UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

MAESTRÍA EN CIENCIAS EXACTAS, SISTEMAS Y DE LA INFORMACIÓN, ESPECIALIDAD EN INTELIGENCIA ARTIFICIAL

TEMA DE TESIS:

Búsqueda de puntos dominantes y compresión de objetos binarios basada en un código de tres cambios ortogonales.

TESIS

PARA OPTAR POR EL GRADO DE

MAESTRO EN CIENCIAS

PRESENTA

Mario Alberto Rodríguez Díaz

TUTOR: Hermilo Sánchez Cruz AÑO: 2009

Por este conducto autorizamos al tesista:

Mario Alberto Rodríguez Díaz

La impresión de su documento final de Tesis, con título "Búsqueda de puntos dominantes y compresión de objetos binarios basada en un código de tres cambios ortogonales", ya que cumple con los requisitos de contenido y forma exigidos en la Universidad Autónoma de Aguascalientes.

Asesor

Dr. Hermilo Sánchez Cruz

Sinodales

Dr. Alejandro Padilla Díaz

Dr. Netzahualcoyotl Castañeda Leyva

Centro de Ciencias Básicas

ISC. MARIO ALBERTO RODRÍGUEZ DÍAZ
PASANTE DE LA MAESTRÍA EN CIENCIAS EXACTAS, SISTEMAS Y DE LA INFORMACIÓN CON ESPECIALIDAD EN INTELIGENCIA ARTIFICIAL
P R E S E N T E .

Estimado (a) Alumno (a) Rodríguez:

Por medio de este conducto me permito comunicar a Usted que habiendo recibido los votos aprobatorios de los revisores de su trabajo de tesis titulado: **"Búsqueda de puntos dominantes y compresión de objetos binarios basada en un código de tres cambios ortogonales"**, hago de su conocimiento que puede imprimir dicho documento y continuar con los trámites para la presentación de su examen de grado.

Sin otro particular me permito saludarle muy afectuosamente.

A T E N T A M E N T E
Aguascalientes, Ags., 22 de junio de 2009
"LUMEN PROFERRE"
EL DECANO

DR. FRANCISCO JAVIER ÁLVAREZ RODRÍGUEZ

UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES
CENTRO DE CIENCIAS BÁSICAS

FJAR,mjda

RESUMEN

En esta tesis se ha implementado el código 3OT (tres cambios ortogonales) para la codificación de imágenes en dos colores (binarias) con uno o más objetos (pixeles en estado encendido) con o sin hoyos (pixeles en estado apagado), además, se desarrolló e implementó en una interfaz amigable un algoritmo para la decodificación del código 3OT generado, con el fin de recuperar la imagen original sin pérdida de información. Esto ha servido de apoyo en la investigación acerca de la aplicación de algoritmos de compresión sobre el código 3OT y se observó que con el algoritmo Aritmético se obtienen mejores resultados que con el estándar internacional JBIG(Joint Bi-level Image Experts Group) para comprimir imágenes binarias y que esta mejoría de compresión es independiente del tamaño en pixeles del contorno de los objetos. Por otro lado, y como otra de las principales aportaciones de la tesis, se implementó un método para la detección de puntos dominantes, aquellos puntos que al ser unidos mediante segmentos de línea recta discreta, generan polígonos que se asemejan a los objetos originales, basado en el análisis del código 3OT, este método fue comparado con los algoritmos más referenciados en la literatura usando imágenes que son ejemplos generalmente usados para evaluar el rendimiento de algoritmos para la detección de puntos dominantes, obteniendo resultados superiores que el mejor de los otros métodos.

(**Palabras clave:** imágenes binarias, objetos binarios, hoyos binarios, código 3OT, algoritmo Aritmético, JBIG, compresión, puntos dominantes)

AGRADECIMIENTOS

A las instituciones que me brindaron su apoyo económico para lograr este trabajo, CONCYTEA y CONACYT. Al programa de becas de PROMEP.

A mis maestros durante el postgrado, especialmente a mi tutor Dr. Hermilo Sánchez Cruz, a mis maestros Dra. Eunice Ponce de León Sentí, Dra. Elva Díaz Díaz, Dr. Alejandro Padilla Díaz, Dr. Netzahualcoyotl Castañeda Leyva.

A mis compañeros, por su apoyo y empatía Lic. Sergio Enríquez Aranda, Ing. Sergio Nevarez, Lic. Leoncio Ibarra.

A la gente que me apoyó profesionalmente para la correcta conclusión del presente trabajo, M.C. Luis Enrique Arámbula Miranda, Ing. Leopoldo Vázquez González, M.C. José Luis Gómez Serrano, Sr. Rodrigo Gómez Córdoba, M.C. Fernando Robles Casillas y a mis alumnos de la Universidad Autónoma de Aguascalientes y del Instituto Tecnológico de Aguascalientes.

A mi familia por todo su amor, mis padres Sra. Irma Esthela Díaz Tapia, Sr. Mario Alberto Rodríguez Calderón, hermano Gerardo Rodríguez Díaz. A mis amigos por todo su apoyo, José Miguel Valdez Pacheco, Nicanor Solís Villegas, Renata Galindo. Muy especialmente, con todo mi amor a Laura por darme esas gotitas, más bien tragos, de tranquilidad cuando más lo he necesitado.

ÍNDICE

Capítulo 1

INTRODUCCIÓN

En este capítulo se muestra, de manera general, en qué consiste el trabajo de la tesis, cuál es su campo de aplicación y las principales aportaciones hechas con el trabajo de investigación, experimentación y desarrollo de la tesis.

1.1 Descripción del Contexto del Problema de Investigación

Las imágenes binarias son un dominio de interés clásico dentro de la industria del fax, porque las máquinas facsímiles necesitan estándares eficientes para aumentar la velocidad de transmisión de documentos. En 1980, un estándar internacional llamado Group 3 fue recomendado por un estudio del CCITT (Consultative Committee for International Telegraphy and Telephony) para este propósito. Este estándar permitió una reducción sustancial de tamaño de almacenamiento sin pérdida de información.

Sin embargo, el trabajo sobre el estándar CCITT Group 3 prosiguió, y en 1984 el CCITT recomendó un nuevo estándar llamado Group 4, aunque no fue muy usado en fax, dicho estándar se usó en imágenes digitales y fue incorporado en el formato TIFF 4.0 (Tagged Image File Format) (Knoll, 2000).

La búsqueda de mejores algoritmos de compresión continuó y en 1993, JBIG (Joint Bi-level Image Experts Group) produjo un nuevo estándar (ISO/IEC 11544) (International Organization for Standarization / International Electrotechnical Comission) conocido como JBIG o JBIG1. Sin embargo, este método, que era sustancialmente mejor que el estándar G4 no fue muy usado, principalmente porque IBM (International Business Machines) poseía varias patentes críticas de la codificación aritmética de la imagen, lo que evitó la implementación libre del estándar.

A finales de los años 90s, se propuso el estándar JBIG2. Aunque hay un modo sin pérdida de información para el JBIG2, este es típicamente usado para compresión con pérdida. El JBIG2 es usado principalmente para comprimir documentos de texto y permite una compresión aproximadamente cinco veces mejor que la obtenida con el formato CCITT/G4.

Los métodos tradicionales para comprimir imágenes binarias consisten en el barrido de las imágenes y el registro de las secuencias de pixeles encendidos para después hacer la compresión de dichas secuencias usando diversos métodos como el algoritmo de Huffman o el algoritmo Aritmético.

Una alternativa a los métodos tradicionales para comprimir imágenes binarias es usar códigos de cadenas de caracteres, los cuáles toman en cuenta la forma de los objetos contenidos en las imágenes para hacer la compresión.

Los códigos de cadena son secuencias de caracteres que se utilizan para representar un conjunto de puntos, que constituyen una línea curva, por lo tanto también pueden ser usados para representar una frontera cuando la curva es cerrada, es decir, el pixel final es vecino del pixel inicial.

El primer método para representar curvas digitales mediante códigos de cadena fue desarrollado por (Freeman, 1961). Para la definición del código de cadena de Freeman, se tiene en cuenta la localización de un pixel (i, j) y sus 8 vecinos en las direcciones cuantizadas de 45° para vecindad 8-conectada.

En (Sánchez-Cruz & Rodríguez-Dagnino, 2005) se propuso un método de compresión para formas binarias sin pérdida de información, basado en un código de cadena similar al 5OT (cinco cambios ortogonales) de (Bribiesca, 2000), pero aplicado en dos dimensiones, este código tiene un alfabeto de tres símbolos y se conoce como código de tres cambios ortogonales o 3OT.

En un artículo seminal (Attneave, 1954) se propusieron varios ejemplos ilustrativos demostrando como la información visual que llega a la retina es altamente redundante. Attneave argumentaba que un objetivo importante de la percepción es reducir la redundancia y "codificar la información entrante en una forma más económica que en la que afecta a los receptores" (Attneave, 1954). Uno de sus ejemplos más famosos es el dibujado de un gato durmiendo, obtenido al identificar 38 puntos de máxima curvatura o puntos dominantes en una imagen real y conectando esos puntos apropiadamente con una recta. El sujeto felino del dibujo es reconocible inmediatamente, apoyando los razonamientos de Attneave de que la percepción involucra la extracción económica de descriptores de objetos por reducción de redundancia en las imágenes originales (Rao, 2002).

Se han desarrollado muchos algoritmos para detectar puntos dominantes, la mayoría de ellos pueden ser clasificados en dos categorías: algoritmos con enfoque de aproximación poligonal o por enfoque de detección de esquinas (Wu, 2003).

Las esquinas se refieren a aquellos puntos en una línea curva donde se puede asociar una discontinuidad identificable en la curvatura media. La detección de una esquina es una función de la magnitud de la discontinuidad, su brusquedad, y las regiones curvas a cada lado de esta sobre las cuales la curvatura media puede ser considerada como uniforme y libre de discontinuidades (Freeman & Davis, 1976). El propósito de la detección de esquinas es detectar puntos significativos con una curvatura extrema.

Los algoritmos para la detección de esquinas pueden categorizarse en dos tipos: los que usan la información contenida en los valores locales de la escala de grises o los que involucran el análisis de las características del contorno de los objetos.

Gracias a sus características, el código 3OT se ha utilizado para el análisis de las formas binarias, no sólo en su procesamiento, como en (Sánchez-Cruz, 2006) que ideó un método para la detección de puntos dominantes en formas binarias identificando patrones en el código 3OT de las mismas.

El método para la detección de puntos dominantes con 3OT presentado en el presente trabajo está inspirado en el método de (Sánchez-Cruz, 2006) y es un método de detección esquinas basado en el análisis de la frontera de los objetos.

1.2. Objetivos y metas del trabajo de investigación

Con el trabajo de esta tesis se han logrado diversos objetivos, y gracias a la conclusión satisfactoria de dichos objetivos se ha llegado a una meta final. A continuación se describen estos objetivos y meta.

1.2.1. Objetivos

1. El primer objetivo del presente trabajo es implementar en una interfaz amigable un algoritmo que permita hacer la codificación al 3OT de imágenes binarias que contengan uno o más objetos binarios y que dichos objetos puedan contener hoyos, es decir grupos de pixeles en estado apagado dentro de objetos binarios.

2. El segundo objetivo es implementar en una interfaz amigable un algoritmo que permita hacer la decodificación del código 3OT, este algoritmo debe permitir recuperar sin pérdida de información las imágenes binarias originales que se hayan codificado en 3OT.
3. Como tercer objetivo, el software generador de código 3OT deberá permitir, además de recuperar las imágenes originales sin pérdida de información a partir del código, extraer las fronteras de los objetos codificados, al permitir generar archivos de imágenes en formato BMP o JPG que contengan únicamente las fronteras que delimitan a los objetos y hoyos de la imagen original, esto servirá de apoyo en la investigación de las imágenes binarias.
4. Usando como apoyo el software generador de código 3OT, se buscará lograr el cuarto objetivo, que consiste en confirmar o refutar la hipótesis que se tiene de que el método propuesto 3OT-Aritmético para el almacenamiento de imágenes con objetos binarios permite optimizar el espacio en memoria utilizado, y es mejor que el estándar internacional JBIG sin importar el tamaño en pixeles de los contornos de los objetos.
5. El quinto objetivo consiste en implementar en el software generador de código 3OT la detección de puntos dominantes en los objetos de imágenes binarias basada en un método de detección de patrones en el código 3OT.
6. El sexto objetivo es extender el método de detección de puntos dominantes con 3OT, al permitir definir nuevos patrones usando expresiones regulares.
7. Como séptimo y último objetivo es necesario el poder comparar el método para la detección de puntos dominantes con 3OT contra los métodos ya existentes, para lo cual se ha desarrollado una medida propia además de haber usado la distancia de Tanimoto (Tanimoto, 1957) para verificar la eficiencia de aproximación poligonal de los algoritmos.

1.2.2. Meta

La meta final del trabajo de tesis es el desarrollo de un sistema de reconocimiento de puntos dominantes y compresión de imágenes en blanco y negro basado en el código de cadena 3OT. Este sistema facilita la investigación que se relacione con la codificación de imágenes binarias mediante código 3OT y tiene el potencial para usarse comercialmente.

En la Figura 1.1 se muestra la relación entre los diferentes objetivos del trabajo de tesis, así como los subproductos generados por ellos.

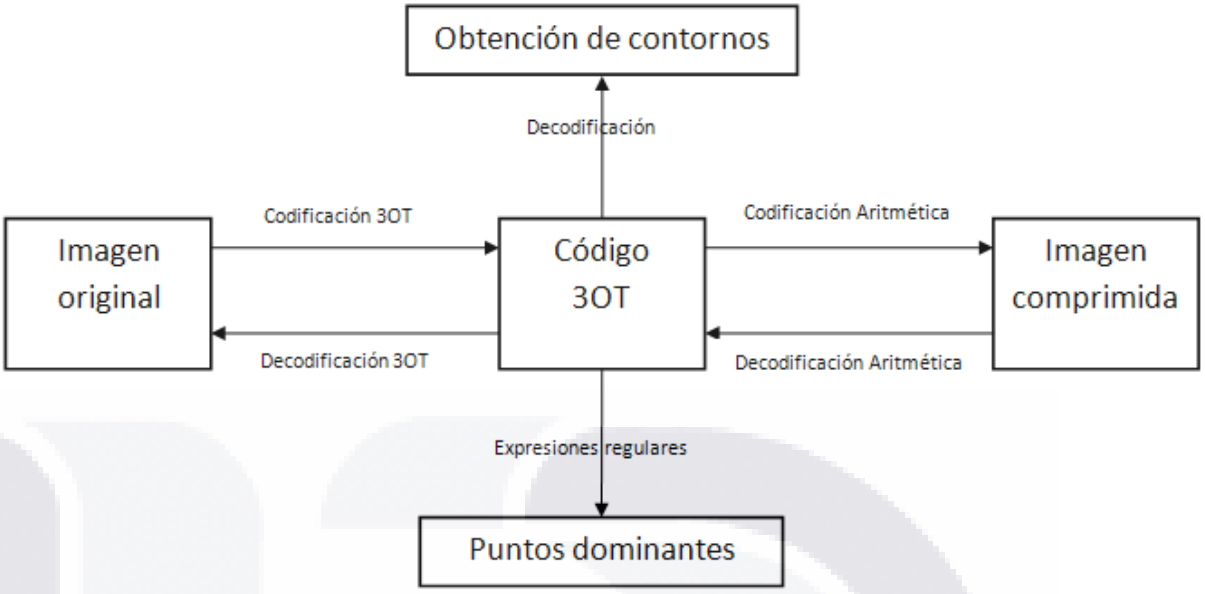


Figura 1.1 Relación entre los objetivos de la tesis.

1.3. Relevancia y Justificación de la Investigación

La información visual se ha convertido en uno de los principales métodos de comunicación en la era moderna. El estudio de las imágenes por computadora se ha desarrollado como un campo de investigación interdisciplinario en el que el enfoque es la adquisición y procesamiento de información visual por medio de computadoras. Entre todos los aspectos subyacentes en la información visual, la forma de los objetos juega un papel especial.

La importancia de las formas como una herramienta para el análisis y clasificación ha llevado a su estudio a muchos campos diversos de la ciencia y la ingeniería. Un paso importante en el análisis computacional de las formas es la representación de las mismas, que se refiere al proceso de representar la forma en una manera adecuada para el almacenamiento y/o análisis. Una clase importante de las técnicas para el análisis de las formas se basa en la representación de las fronteras (contornos) de los objetos. Debido a su rica naturaleza, el contorno es uno de los descriptores de formas comúnmente más usados, y varios métodos para la representación de contornos de objetos 2D se han propuesto en la literatura, cada uno logra, más o menos, el éxito. Las características más deseables de una representación apropiada son la compresión de datos, simplicidad en la codificación y decodificación, escalamiento e

invariancia bajo movimientos rígidos (Marji & Siy, 2003). En el presente trabajo se hacen importantes desarrollos en la investigación de dos tipos de descriptores de formas basados en fronteras: la representación de fronteras mediante códigos de cadena y la detección de puntos dominantes sobre la frontera.

La representación de imágenes binarias mediante códigos de cadenas es un área de estudio que ofrece un amplio campo por ser investigado, hasta ahora se ha hablado poco acerca de la codificación de varios objetos con o sin hoyos en una misma imagen. De ahí ha surgido la necesidad de desarrollar una herramienta que permita hacer la codificación y decodificación de las imágenes con estas características.

Hasta donde se sabe, no existe algún algoritmo para decodificar y reconstruir las imágenes binarias representadas por un código de cadena 3OT, lo cual es una necesidad para poder continuar con las investigaciones en el área.

Se conoce de (Sánchez-Cruz, et al, 2007) que los principales códigos de cadena comprimidos con el algoritmo de Huffman son más eficientes que el estándar internacional JBIG para objetos con fronteras menores a cierto umbral. Se tiene la hipótesis de que al aplicar el algoritmo de compresión Aritmético al código 3OT (método 3OT-Aritmético) se tiene más eficiencia que en el estándar internacional JBIG independientemente del tamaño de las fronteras de los objetos, para poder comprobarlo es necesario el desarrollo de herramientas de apoyo, como software que permita hacer la codificación / decodificación al 3OT de imágenes binarias con varios objetos con o sin hoyos.

Las imágenes binarias son usadas en muchas instancias del procesamiento de datos. Por ejemplo, en imágenes médicas del cerebro: se usan formas binarias para la segmentación de la corteza cerebral antes de un mapeo funcional del cerebro. Hay muchas otras aplicaciones diversas, tales como la identificación de la forma de cerezas (Beyer, et al, 2002), análisis de la estructura molecular del oro o la optimización de diseños aerodinámicos, documentos de texto, mapas, faxes, etc, por nombrar algunas (Rosin & Žunić, 2005).

En visión artificial es frecuente utilizar la morfología matemática para el tratamiento de regiones en el sentido de determinar cómo se pueden cambiar, contar o evaluar. La morfología matemática, que comenzó a desarrollarse a finales de los 60, permanece hoy en día como una parte separada del análisis de imágenes. La morfología matemática está basada en la geometría

y la forma. Las operaciones morfológicas simplifican las imágenes y preservan las formas principales de los objetos.

Las operaciones matemáticas tienen su principal aplicación en imágenes binarias y los fundamentos matemáticos fueron concebidos desde el punto de vista de la posición de los pixeles (que es la base en el tratamiento binario) antes que desde la intensidad.

En visión artificial existe el problema de extraer una figura a partir de una imagen arbitraria, no binaria. Por ejemplo, el tratar de diferenciar a una persona en una foto. Es necesario aislar ciertas partes de las imágenes como entidades significativas. El objetivo consiste en desarrollar un conjunto de técnicas para caracterizar dichas entidades de una manera natural. La cuestión sobre cómo resuelven este problema las personas, ha intrigado durante muchos años a los psicólogos, particularmente a los de la escuela conocida como la Gestalt (Pajares-Martinsanz & de la Cruz-García, 2002). Sin embargo, no ha aparecido una teoría clara sobre dichos esfuerzos; además, hay ilusiones ópticas en las cuales la figura y el fondo aparecen alternantes. Por consiguiente, no existe un algoritmo aplicable con carácter general para extraer las figuras de las imágenes. Por el contrario, hay una variedad de técnicas que son útiles en situaciones particulares, entre ellas por ejemplo la extracción de bordes o las técnicas de binarización donde cualquier punto que excede de un determinado umbral se declara como perteneciente a las figuras subyacentes en la imagen, tal es el planteamiento de (Cheng & Chen, 1999).

Siguiendo la línea de. (Fu, et al, 1988), el problema de la descripción en la visión consiste en extraer las características de un objeto para reconocerlo. Por lo general, los descriptores deben ser independientes de la localización y orientación del objeto y deben contener suficiente información de discriminación para distinguir un objeto de otro, un tipo de descriptores que reúnen dichas características son los códigos de cadena (Pajares-Martinsanz & de la Cruz-García, 2002).

Debido a la utilidad que tienen las imágenes binarias, siempre será conveniente buscar la manera de optimizar el espacio en memoria que se requiere para su almacenamiento.

Por otro lado, los puntos dominantes en objetos binarios son aquellos puntos localizados en la frontera del objeto que tienen la característica de que al ser unidos con segmentos de líneas rectas permiten generar un polígono que tiene un alto grado de aproximación con el objeto al que pertenecen. Se ha sugerido que la información en una curva está concentrada en

los puntos con curvatura extrema (Attneave, 1954). Sirven para simplificar el análisis de imágenes al reducir drásticamente la cantidad de datos a ser procesados, preservando al mismo tiempo información importante acerca de los objetos.

Los puntos dominantes son características esenciales en imágenes 2D. La identificación certera de esos puntos provee información importante en numerosas aplicaciones de visión por computadora tales como visión estereoscópica, detección de movimiento y análisis de escenas. Indican la presencia y localización de objetos, estrechando el problema de búsqueda y haciendo la interpretación de imágenes en alto nivel más fácil. En el procesamiento de imágenes y reconocimiento de patrones, los puntos dominantes son usados para describir la forma de curvas planas, así como reducir la cantidad de datos.

Se sabe que el método desarrollado por (Sánchez-Cruz, 2006) para la detección de puntos dominantes analizando patrones en las cadenas del 3OT puede ser ampliado al permitir implementar nuevos patrones a buscar en las cadenas de símbolos, para lograrlo es necesario el desarrollo de un método que permita definir los patrones de búsqueda, una idea para ello es usar expresiones regulares.

1.4. Descripción General del Enfoque, Métodos y Técnicas de Investigación

Los códigos de cadena son representaciones, en formato de cadenas de caracteres, de líneas que pueden ser rectas o curvas, un objeto binario en una imagen es delimitado por una línea curva cerrada, por lo tanto, los objetos binarios pueden ser descritos mediante códigos de cadena.

El 3OT, es un código de cadena desarrollado por (Sánchez-Cruz & Rodríguez-Dagnino, 2005). Con el presente trabajo se ha logrado hacer tanto la codificación como la decodificación en 3OT de imágenes que pueden contener varios objetos con o sin hoyos. Hasta ahora no se sabe que se haya hecho la reconstrucción de una imagen binaria a partir del código que la representa.

El algoritmo Aritmético es un método de compresión sin pérdida de información para cadenas de caracteres que permite una eficiencia de compresión cercana a la entropía (Said, 2003), de manera que el método 3OT-Arimético consiste en comprimir con el algoritmo Aritmético la cadena de caracteres 3OT que representa a una imagen binaria. El algoritmo para compresión de imágenes binarias 3OT-Aritmético es un método que permite guardar las imágenes en un espacio de almacenamiento menor al requerido por el formato JBIG con

independencia del tamaño en pixeles de las fronteras, el desarrollo de la aplicación que permite codificar / decodificar imágenes binarias al 3OT ha servido de apoyo para demostrar este hecho.

Se hicieron diversas pruebas para comparar el nivel de compresión entre el JBIG y el 3OT-Aritmético, se utilizaron imágenes binarias con la característica de no ser geométricas, sino irregulares, a las que se les aplicó el método 3OT-Aritmético y se registro la cantidad de bytes resultante, luego las mismas imágenes fueron convertidas al formato JBIG para registrar su tamaño de almacenamiento. En todas las pruebas hechas se obtuvo un tamaño de almacenamiento menor con el 3OT-Aritmético que con el JBIG, y en promedio el espacio de almacenamiento necesario para las imágenes comprimidas con 3OT-Aritmético fue un 20% menor que el usado por el JBIG. En el análisis de diferentes métodos de compresión hecho por (Sánchez-Cruz, et al, 2007) se observó que los métodos propuestos en dicho trabajo son mejor que el JBIG sólo para imágenes menores a cierto tamaño, el umbral que separaba esto variaba para los diferentes métodos, una aportación muy importante del presente trabajo es que se encontró que la eficiencia en la compresión del método 3OT-Aritmético encontrada es independiente del tamaño de los objetos.

Para la detección de puntos dominantes se hace un análisis de subcadenas del código 3OT que representa al objeto binario que se procesa. Se verifica si estas subcadenas cumplen con algún patrón específico que representa una región de la curva contenedora de un punto pivote con curvatura extrema o esquina. Al haber encontrado todas las subcadenas que cumplen con el patrón de búsqueda se forman grupos de puntos pivotes que se encuentren a ciertos caracteres del 3OT entre sí y se aplican procesos para eliminar pivotes redundantes en los grupos, tomando aquellos pivotes que representen mejor al grupo al que pertenecen.

En el software generador de código 3OT se implementó la característica de permitir leer los patrones de búsqueda, expresados en un formato similar a expresiones regulares, desde archivos de texto, lo que le permite al usuario definir fácilmente nuevos patrones de búsqueda sin necesidad de modificar el código fuente del programa y recompilar.

Se experimentó con tres patrones en esta tesis que representan cambios de dirección en la curva de la frontera de los objetos. Se desarrolló una medida de la eficiencia para grupos de puntos dominantes y se midieron los resultados del 3OT y los métodos para la detección de puntos dominantes con más referencias en la literatura. Como objetos de medición se usaron

algunas imágenes que son benchmarks, utilizadas generalmente para medir la detección de puntos dominantes con distintos métodos. Los resultados mostraron que el método 3OT tiene mejor eficiencia en la detección de los puntos dominantes que el resto de los métodos analizados. Además, el método 3OT evita la redundancia en los puntos detectados, ya que no se encuentran puntos dominantes sobre las partes rectas de la frontera de los objetos.

1.5. Descripción de Capítulos de la Tesis

- Capítulo 1. Introducción: En este capítulo se muestra, de manera general, en qué consiste el trabajo de tesis, cuál es su campo de aplicación y las principales aportaciones hechas con el trabajo de investigación, experimentación y desarrollo de la tesis.
- Capítulo 2. Marco teórico: El trabajo de experimentación de la tesis tiene fundamentos teóricos de diferentes temas, como lo son imágenes digitales, lenguajes formales, códigos de cadena, teoría de la información y puntos dominantes. En este capítulo se revisan dichos fundamentos teóricos.
- Capítulo 3. Codificación y Decodificación en 3OT: Se explica en qué consisten los algoritmos desarrollados para la codificación y decodificación de imágenes con objetos binarios a código de cadena 3OT. También, incluye una explicación acerca del algoritmo de compresión aritmético y cómo se aplicó a las cadenas de caracteres en 3OT.
- Capítulo 4. Detección de puntos dominantes con 3OT: En esta sección se describe el método para la detección de puntos dominantes basado en 3OT, también se explica en qué consiste la extensión implementada para dicho método.
- Capítulo 5. Software generador de código 3OT: Para la realización del trabajo de experimentación se ha desarrollado un software generador de código 3OT, en este capítulo se describe el sistema programado.
- Capítulo 6. Reporte y discusión de resultados: En este capítulo se muestran los resultados obtenidos de los experimentos con el compresor 3OT-Aritmético y con el algoritmo para la detección de puntos dominantes basado en 3OT para imágenes con objetos binarios. Se describen los objetos de medición usados, los parámetros de medición empleados y se muestran los resultados obtenidos para cada experimento.

- Capítulo 7. Conclusiones: En este capítulo se explica el significado de los resultados obtenidos en el capítulo anterior, así como de sus implicaciones e importancia. También se comenta acerca del potencial que tiene el software generador de código 3OT, describiendo el trabajo futuro que se puede desarrollar usándolo como herramienta de apoyo.

1.6. Aportaciones Principales de la Tesis

Con el presente trabajo se han hecho diversas aportaciones, tanto de investigación como prácticas, las cuáles se enumeran a continuación:

1. Eficiente detección de puntos dominantes. Se implementó un método para la aproximación de formas por medio de polígonos generados al unir puntos dominantes con segmentos de rectas, dichos puntos dominantes son detectados mediante el reconocimiento de patrones dentro del código de cadena 3OT. Este método ha resultado ser más eficiente que los métodos más citados en la literatura, además de evitar la redundancia de información. Las mediciones para la eficiencia de los algoritmos se realizaron con un coeficiente nuevo desarrollado, además de haber usado la distancia de Tanimoto (Tanimoto, 1957).
2. Compresión de imágenes binarias con independencia de tamaño. Por el trabajo de (Sánchez-Cruz, et al, 2007) se conoce que los principales códigos de cadena permiten una mejor eficiencia de compresión para imágenes binarias que el estándar internacional JBIG, pero sólo para imágenes que contienen objetos con fronteras menores a cierto tamaño en pixeles. Con este trabajo se comprobó que el método de compresión 3OT-Aritmético es más eficiente que el estándar internacional JBIG para la compresión de imágenes binarias independientemente del tamaño en pixeles de las fronteras de los objetos en las imágenes.
3. Software con interfaz amigable. Otra aportación muy importante es el desarrollo de una herramienta de apoyo para la investigación.

   En la codificación de imágenes binarias al 3OT históricamente se ha trabajado principalmente con imágenes consideradas simples: con una forma binaria y sin hoyos como la presentada en la Figura 1.2, con el presente trabajo se ha desarrollado un software que permite codificar imágenes más complejas, es decir que pueden contener más de un objeto binario y estos pueden tener hoyos como la presentada en la Figura 1.3.

Figura 1.2 Imagen binaria, Un objeto sin hoyos.

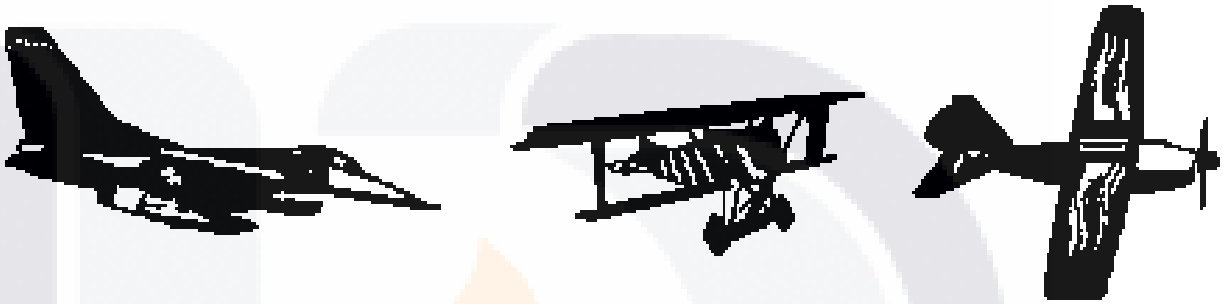


Figura 1.3 Imagen binaria, Tres objetos con varios hoyos.

Otra aportación importante es el hecho de poder recuperar las imágenes originales a partir del código 3OT, se ha implementado en el software un algoritmo que permite interpretar los códigos de cadena para generar las fronteras que estos representan, posteriormente también puede hacerse el rellenado de dichas fronteras para reconstruir la imagen binaria original.

También, se desarrolló un método que permite definir nuevos patrones a reconocer en las cadenas de símbolos 3OT para detectar puntos dominantes, el software lee desde archivos de texto los nuevos patrones escritos en un formato similar a expresiones regulares. Gracias a esta característica del software se puede experimentar fácilmente con nuevos patrones de búsqueda sin tener que programarlos, bastará definirlos en el formato requerido.

Capítulo 2

MARCO TEÓRICO

El trabajo de experimentación de la tesis tiene fundamentos teóricos de diferentes temas, como lo son imágenes digitales, lenguajes formales, códigos de cadena, teoría de la información, y puntos dominantes. En el presente capítulo se revisan dichos fundamentos teóricos.

2.1. Imágenes Digitales

2.1.1. Mallas en dos dimensiones.

$\mathbb{G}$ es un subconjunto finito rectangular de un arreglo plano ortogonal $\mathbb{Z}^2$

$$\mathbb{Z}^2 = \mathbb{Z} \times \mathbb{Z} = \{(i,j): i,j \in \mathbb{Z}\},$$

Fórmula 2.1Plano ortogonal 2D.

donde $\mathbb{Z}$ es el conjunto de todos los enteros. En la Fórmula 2.1 se representa el plano ortogonal en 2D. A $\mathbb{G}$ se le conoce como una *malla* en dos dimensiones y a cada elemento en $\mathbb{G}$ se le conoce como *punto de malla*. Un *vértice de malla* está trasladado (0.5, 0.5) con respecto a un punto de malla. Un par de vértices de malla adyacentes definen una *arista de malla*. Un *cuadro de malla* es definido por cuatro aristas de malla que forman un cuadro (Rosenfeld & Klette, 2004). A una malla $\mathbb{G}$ en informática se le conoce como *mapa de bits*.

Se pueden definir los puntos, vértices, aristas y cuadros de malla en términos de coordenadas cartesianas. Las posiciones cartesianas de los vértices de malla se definen en la Fórmula 2.2

$$(0.5, 0.5) + \mathbb{Z}^2 = \{(i + 0.5, j + 0.5): i,j \in \mathbb{Z}\}.$$

Fórmula 2.2 Vértices de malla, en coordenadas cartesianas.

Una arista de malla conecta un par de vértices adyacentes, en la Fórmula 2.3 se aprecian sus coordenadas cartesianas.

$$(i + 0.5, j + 0.5)(i + 0.5, j + 1.5) \text{ ó } (i + 0.5, j + 0.5)(i + 1.5, j + 0.5)$$

Fórmula 2.3 Arista de malla, en coordenadas cartesianas.

Un cuadro de malla es definido por cuatro aristas de malla donde sucesivas aristas comparten un vértice, en la Fórmula 2.4 se pueden apreciar sus coordenadas cartesianas.

$$(i + 0.5, j + 0.5)(i + 0.5, j + 1.5)(i + 1.5, j + 1.5)(i + 1.5, j + 0.5)$$

Fórmula 2.4 Cuadro de malla, en coordenadas cartesianas.

Definición 2.1 Celdas.

Las dimensiones de los cuadros, aristas y vértices sugieren una terminología alternativa, Un cuadro de malla es llamado 2-Celda, una arista de malla es llamada 1-Celda y un vértice de malla es llamado 0-Celda. La Figura 2.1 ilustra un ejemplo de una malla $\mathbb{G}$ con los tipos de Celda.

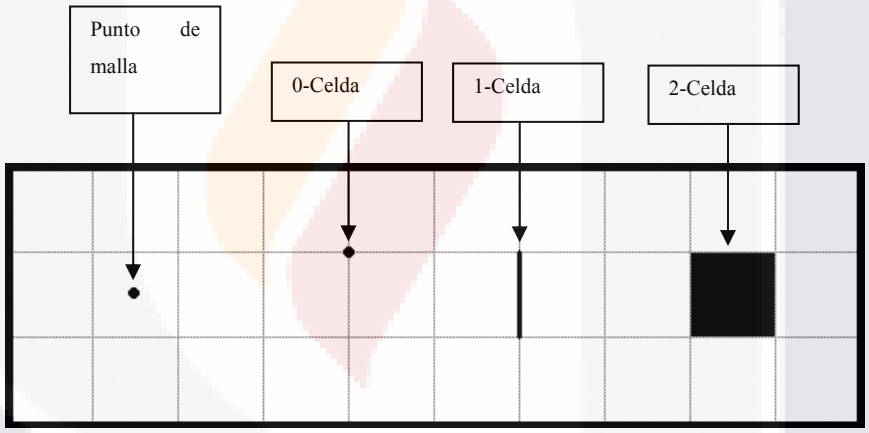


Figura 2.1 Malla G con los tipos de Celda.

En esta terminología, un punto de malla es el centro de una 2-Celda. También, $\mathbb{C}_2^{(i)}$ denotará el conjunto de todas las i-Celdas con (i = 0, 1, 2) en el plano (Rosenfeld & Klette, 2004). También se define una malla $\mathbb{G}$ por medio del modelo de malla de celdas, como puede apreciarse en la Fórmula 2.5.

$$\mathbb{C}_2 = \mathbb{C}_2^{(2)} \cup \mathbb{C}_2^{(1)} \cup \mathbb{C}_2^{(0)}$$

Fórmula 2.5 Malla G en el modelo de malla de celdas.

Definición 2.2 Modelos de mallas.

En el modelo de malla de celdas, una malla $\mathbb{G}$ es $\mathbb{C}_2$, o bien un "bloque" de tamaño $m \times n$ de 2-Celdas en el que la unión $\cup\, \mathbb{G}$ es una región rectangular del plano euclideano. En el modelo de malla de puntos, una malla $\mathbb{G}$ es una malla $\mathbb{Z}^2$ finita o un subarreglo rectangular de tamaño $m \times n$ de $\mathbb{Z}^2$.

2.1.2. Pixeles.

Definición 2.3 Pixel.

Un pixel p es una 2-Celda (cuadro de malla) en el modelo de malla de celdas o un punto de malla (el centro de una 2-Celda) en el modelo de malla de puntos *(Rosenfeld & Klette, 2004)*.

"La palabra pixel es un acrónimo basado en dos palabras inglesas: picture y element. Es decir, quiere significar elemento de imagen y al contraer ambas palabras surgió 'pixel' como abreviatura y término definitivo" (Network-Press.Org::En Tierra Firme, 2003-2008). El término pixel se usó por primera vez en los años 60s por un grupo en el Jet Propulsion Laboratory en Pasadena, California en donde se utilizaron técnicas de computadora para mejorar las imágenes de la Luna enviadas por la sonda espacial Ranger 7 (Pajares-Martinsanz & de la Cruz-García, 2002).

Un pixel es la menor unidad homogénea en color que forma parte de una imagen digital, ya sea esta una fotografía, un fotograma de video o un gráfico.

Ampliando lo suficiente una imagen digital (zoom), por ejemplo en la pantalla de una computadora, pueden observarse los pixeles que componen la imagen. Los pixeles aparecen como pequeños cuadrados o rectángulos en color, en blanco y negro, o en matices de gris. En la Figura 2.2 se muestra una imagen y los pixeles que se aprecian al hacer zoom a una pequeña región.

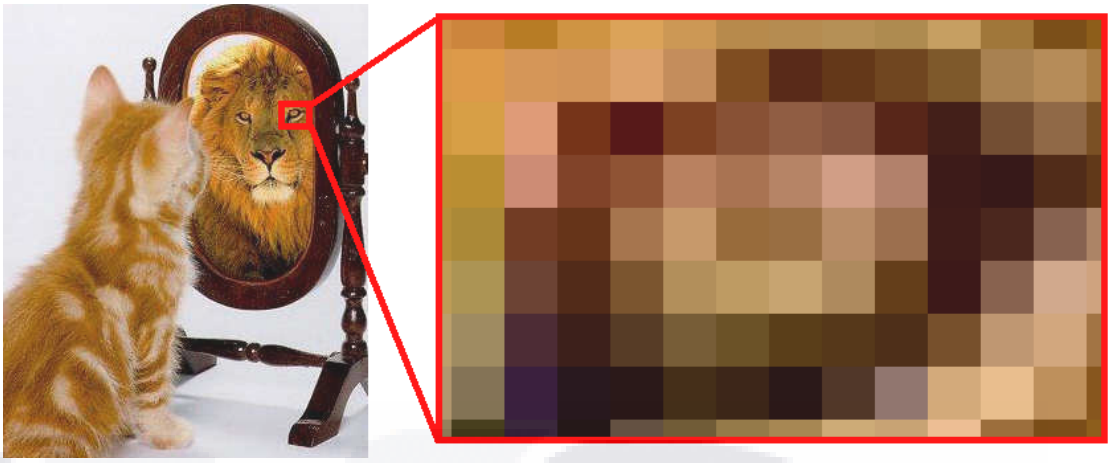


Figura 2.2 Zoom en imagen y pixeles.

2.1.3. Sistema de coordenadas de pixeles.

Definición 2.4 Sistema de coordenadas de pixeles.

En imágenes por computadora generalmente el método más conveniente para expresar locaciones en una imagen es usar coordenadas de pixeles. En este sistema de coordenadas la imagen es tratada como una malla de elementos discretos, ordenados de arriba hacia abajo y de izquierda a derecha comenzando con el elemento 0, como se muestra en la Figura 2.3.

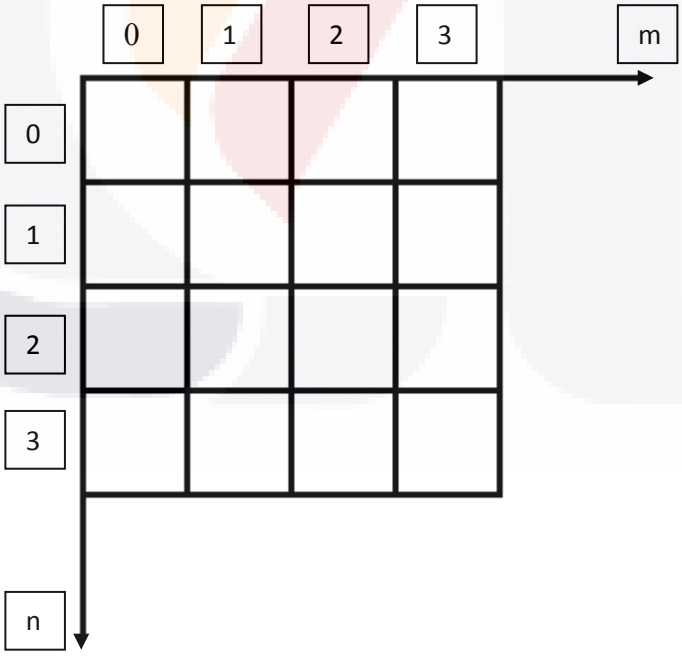


Figura 2.3 Coordenadas de pixeles.

Los algoritmos en el análisis de imágenes son aplicados frecuentemente a los pixeles de una imagen en una secuencia obtenida al escanear la malla $\mathbb{G}$. El escaneo de una malla $\mathbb{G}_{m,n}$ es un mapeo θ uno a uno de los $m \times n$ pixeles de la malla en una secuencia θ(1), …, θ(mn). Los escaneos se usan en programas de procesamiento de imágenes para controlar el orden en que se accede a los pixeles. Un escaneo también puede ser visto como una enumeración de los pixeles; θ(k) es el k-ésimo pixel de la imagen donde ($1 \leq k \leq mn$) (Rosenfeld & Klette, 2004).

Definición 2.5 Orden de línea mayor.

El método estándar de escaneo para una malla $\mathbb{G}_{m,n}$ es línea por línea, con los pixeles en cada línea accedido de izquierda a derecha y las columnas accedidas de arriba hacia abajo; este orden de escaneo es frecuentemente llamado orden de línea mayor. Es un orden lexicográfico derivado de las coordenadas de pixeles: (0, 0), (0, 1), …, (0, n), (1, 0), (1, 1), …., (m, n) donde (0, 0) es la esquina superior izquierda; $(i_1, j_1) < (i_2, j_2)$ si y sólo sí $i_1 < i_2$, ó $i_1 = i_2$ cuando $j_1 < j_2$

En la Figura 2.4 se muestra un ejemplo de escaneo de los pixeles en una malla mediante el orden de línea mayor.

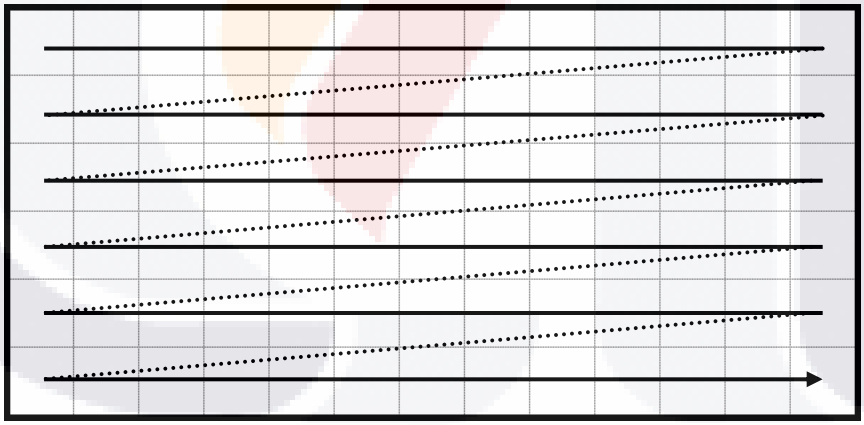


Figura 2.4 Escaneo de orden de línea mayor.

2.1.4. Adyacencia y conectividad.

Definición 2.6 Adyacencia, 4 pixeles.

Dos 2-Celdas c_1 y c_2 son 1-Adyacentes si y sólo si $c_1 \neq c_2$ y $c_1 \cap c_2$ es una 1-Celda. Dos puntos de malla $p_1 = (x_1, y_1)$ y $p_2 = (x_2, y_2)$ son llamados 4-Adyacentes sí y solo sí $|x_1 - x_2| + |y_1 - y_2| = 1$.

En otras palabras, dos 2-Celdas c_1 y c_2 son 1-Adyacentes si y sólo si no son idénticas pero comparten una arista de malla. Haciendo a p_i el centro de c_i con (i=1, 2); entonces c_1 y c_2 son 1-Adyacentes si y solo si p_1 y p_2 son 4-Adyancentes.

Ahora se definen las relaciones A_1 de 1-Adyancencia y A_4 de 4-Adyacencia para los modelos de malla de celdas y malla de puntos respectivamente. En general, escribimos pRq o (p, q) ∈ R si y sólo si p, q están en relación R. En esta notación, tenemos $c_1A_1c_2$ si y sólo si $\{c_1, c_2\} \in A_1$ si y sólo si c_1 y c_2 son 1-Adyacentes, y $p_1A_4p_2$ si y solo si $\{p_1, p_2\} \in A_4$ si y sólo si p_1 y p_2 son 4-Adyacentes. Las relaciones A_1 y A_4 son simétricas e irreflexivas. La Figura 2.5 muestra una relación A_1 y la Figura 2.6 muestra una relación de pixeles A_4.

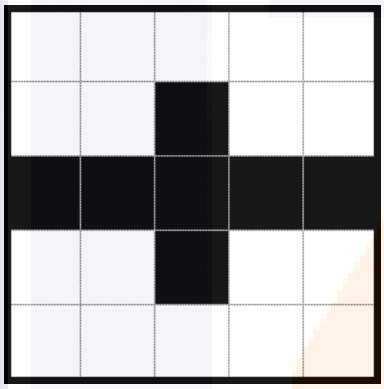


Figura 2.5 Relación A1.

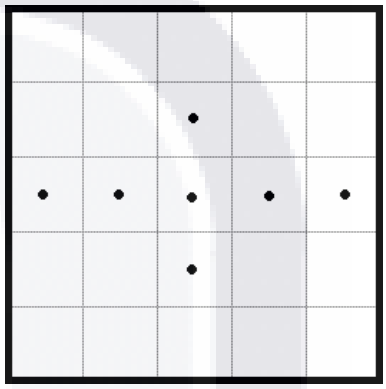


Figura 2.6 Relación A4.

Definición 2.7 Adyacencia, 8 pixeles.

Dos 2-Celdas c_1 y c_2 son llamadas 0-Adyacentes si y solo si $c_1 \neq c_2$ y $c_1 \cap c_2$ contiene al menos una 0-Celda. Dos puntos de malla $p_1= (x_1, y_1)$ y $p_2 = (x_2, y_2)$ son llamados 8-Adyacentes sí y solo sí $\max\{|x_1-x_2|, |y_1-y_2|\}=1$.

En otras palabras, dos 2-Celdas c_1 y c_2 son 0-Adyacentes si y sólo si no son idénticas pero comparten al menos un vértice de malla. Haciendo a p_i el centro de c_i con (i=1, 2); entonces c_1 y c_2 son 0-Adyacentes si y sólo si p_1 y p_2 son 8-Adyancentes. Las relaciones A_0 y A_8 son simétricas e irreflexivas. La Figura 2.7 muestra una relación A_0 y la Figura 2.8 muestra una relación de pixeles A_8.

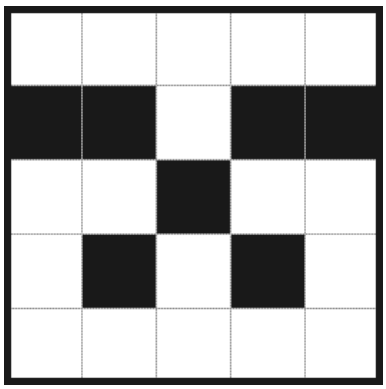


Figura 2.7 Relación A0.

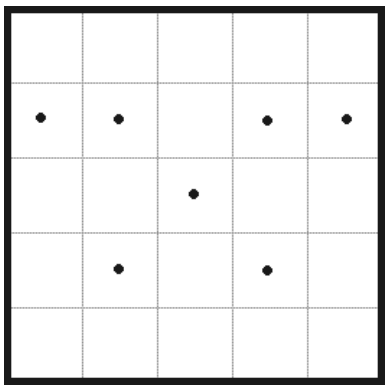


Figura 2.8 Relación A8.

Las 2-Celdas adyacentes se transforman en puntos de malla adyacentes mapeando las 2-Celdas en sus puntos centrales, y los puntos de malla adyacentes se transforman en 2-Celdas adyacentes mapeándolos en las 2-Celdas que los tienen a ellos como puntos centrales. La existencia de este mapeo uno a uno nos permite lo siguiente:

Proposición 2.1. La malla $\mathbb{G}$ definida por las $m \times n$ 2-Celdas y la relación de adyacencia A_0 es isomorfa a la malla definida por los $m \times n$ puntos de malla y la relación de adyacencia A_8. La malla $\mathbb{G}$ definida por las $m \times n$ 2-Celdas y la relación de adyacencia A_1 es isomorfa a la malla definida por los $m \times n$ puntos de malla y la relación de adyacencia A_4. Cualquiera de esas mallas será denotada por $\mathbb{G}_{m,n}$ (Rosenfeld & Klette, 2004).

Definición 2.8 Conexión.

Sea M un subconjunto de $\mathbb{G}$. Se dice que M es i-Conectado para cada i = 0, 1, 4, 8 si para todos p, q∈M existe una secuencia $p_0,\ldots, p_n$ de elementos de M tales que $p_0 = p$, $p_n = q$, y $p_{j-1}A_ip_j$ $(1 \leq j \leq n)$; a tal secuencia se le conoce como i-ruta y se dice que i-conecta a p y q en M.

Los conceptos de 0-, 1-, 4- y 8-Adyacencia y Conectividad fueron introducidos al análisis de imágenes en 1966 (Rosenfeld & Pfaltz, 1966), aunque los prefijos 0-, 1-, 4- y 8- no se habían usado hasta años recientes.

También suele usarse lenguaje geográfico para identificar las relaciones de adyacencia de los pixeles. En este caso, si dos pixeles tienen una relación de adyacencia, se dice que son vecinos.

Para el pixel $p = (x, y) \in \mathbb{Z}^2$ sus vecinos en la relación A_4 son mostrados en la Fórmula 2.6.

$$V_4(p) = \{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\}.$$

Fórmula 2.6 Vecinos en la relación A4.

y para la relación A_8 se ven en la Fórmula 2.7.

$$V_8(p) = \{V_4(p) \cup (x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)\}$$

Fórmula 2.7 Vecinos en la relación A8.

Dos pixeles $p, q \in \mathbb{Z}^2$ se conocen como vecinos 4-Adyancentes si $p \neq q$ y $p \in V_4(p)$.

Dos pixeles $p, q \in \mathbb{Z}^2$ se conocen como vecinos 8-Adyacentes si $p \neq q$ y $p \in V_8(p)$.

A cada uno de los miembros del conjunto $V_8(p)$ se les denomina usando el lenguaje geográfico:

(x+1,y)	vecino *este* de *p*
(x+1, y-1)	vecino *noreste* de *p*
(x, y -1)	vecino *norte* de *p*
(x-1, y-1)	vecino *noroeste* de *p*
(x-1, y)	vecino *oeste* de *p*
(x-1, y+1)	vecino *suroeste* de *p*
(x, y+1)	vecino *sur* de *p*
(x+1, y+1)	vecino *sureste* de *p*

Las relaciones 1-Adyacente (A_1) o 4-Adyacente (A_4) en una imagen *I* es cuando 2 pixeles se consideran vecinos hacia el este, norte, oeste o sur uno del otro. En la Figura 2.9 se ven los vecinos en A_1 del pixel marcado con rojo.

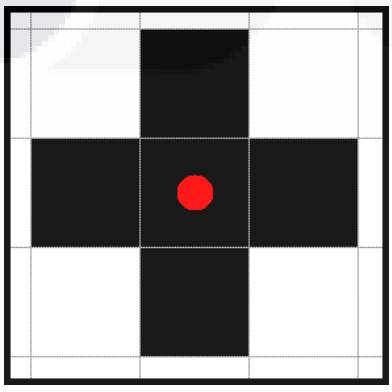


Figura 2.9 Vecinos A1 del pixel marcado con rojo.

Las relaciones Adyacente-0 (A_0) o Adyacente-8 (A_8) en una imagen *I* es cuando 2 pixeles se consideran vecinos hacia el este, noreste, norte, noroeste, oeste, suroeste, sur o sureste uno del otro, esto es que también se conectan de manera diagonal. En la Figura 2.10 se ven los vecinos en A_0 del pixel marcado con rojo.

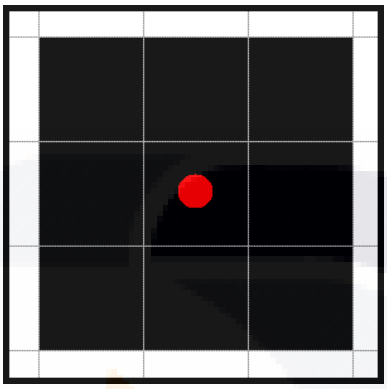


Figura 2.10 Vecinos A0 del pixel marcado con rojo.

2.1.5. Imágenes.

Definición 2.9 Imagen.

Una imagen I en dos dimensiones es una función definida en una malla $\mathbb{G}$. I le asigna un valor I(p) a cada pixel $p \in G$.

Los valores de los pixeles pueden ser enteros, números de punto flotante, vectores, o incluso conjuntos finitos. En las funciones I cada pixel puede tomar sólo un número finito de posibles valores. El rango de valores en una función I usualmente estará en la forma $\{0, \dots, G_{max}\}$, donde $0 \leq I(p) \leq G_{max}$. Cuando $G_{max} = 0$ es el caso trivial de una imagen en blanco *(Rosenfeld & Klette, 2004)*.

En informática, una imagen digital es obtenida mediante un lector electrónico, como un escáner o una cámara, que digitaliza la imagen, transformándola en pixeles. Lo que hace el lector electrónico es transformarla de analógica en digital, transforma lo que podemos ver en datos pasibles de ser trasladados a la computadora en binario.

Independientemente del tipo de sensor utilizado, la imagen que ha de ser tratada por la computadora se presenta digitalizada espacialmente en forma de matriz con una resolución de $m \times n$ elementos. Cada elemento de la matriz denominado pixel tiene un valor asignado que corresponde con el color que representa el punto discreto en la escena captada. La Figura 2.11 muestra una imagen digitalizada cualquiera.

Figura 2.11 Imagen digital, escena del mundo real capturada por una cámara.

Para la relaciones A_1 en $\mathbb{C}_2$ y A_4 en $\mathbb{Z}^2$ el número de pixeles adyacentes a cualquier pixel es siempre 4. Para las relaciones A_0 en $\mathbb{C}_2$ y A_8 en $\mathbb{Z}^2$ es siempre 8. Sin embargo, este número puede variar en las relaciones de adyacencia definidas en imágenes.

Considerando a *I* una imagen definida en la malla $\mathbb{G}$ donde el pixel *p* tiene el valor *I(p)* en $\{0, 1, \ldots, G_{max})$. Decimos que dos pixeles *p* y *q* son *I-equivalentes* si y sólo si *I(p) = I(q)*.

Las relaciones de adyacencia en imágenes están definidas por la localización y el valor de los pixeles. Entonces, en una imagen *I.* Dos pixeles *p* y *q*, son i-Adyacentes si y sólo si pA_iq e *I(p) = I(q)* para i = 0, 1, 4, 8. Para que *p* y *q* sean i-Adyacentes en una imagen, además de estar en la relación A_i deben de ser I-equivalentes.

De manera similar, una i-ruta para cada *i = 0, 1, 4, 8* en una imagen *I* es aquella secuencia de pixeles $p_{0, \ldots, p_n}$ en que $p_{j-1}A_ip_j$ $(1 \leq j \leq n)$ y p_{j-1} es I-equivalente con p_j.

Definición 2.10 Cuerda.

Una i-cuerda para cada i = 0, 1, 4, 8 es una i-ruta de pixeles p_j $(0 \leq j \leq n)$ en una imagen que i-conecta a p_0 con p_n en donde ningún p_j tiene relación de adyacencia con 4 pixeles para i = 0, 8 o con 8 pixeles para i = 1, 4. En otras palabras, una cuerda es aquella ruta de pixeles en una imagen, en donde ningún pixel tiene vecinos en todas las direcciones posibles de la relación de adyacencia usada. En la Figura 2.12 pueden apreciarse los pixeles que conforman una cuerda.

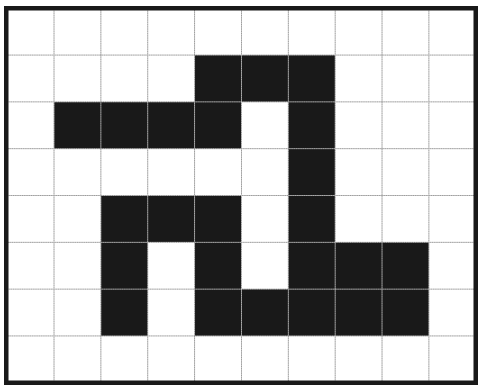


Figura 2.12 Ejemplo de una cuerda.

2.1.6. Imágenes binarias.

Definición 2.11 Imagen binaria.

Una imagen binaria Ib es aquella función I que tiene un rango de valores $\{0, 1\}$, es decir $G_{max} = 1$. Los pixeles cuyo valor es 1, son llamados 1s definen un subconjunto $\langle O \rangle$ de la malla $\mathbb{G}$. Los pixeles cuyo valor es 0, son llamados 0s y definen un subconjunto $\langle \bar{O} \rangle$ de la malla $\mathbb{G}$ *(Rosenfeld & Klette, 2004)*.

Una imagen binaria, también llamada imagen binivel o monocromática, es una imagen de computadora en donde cada pixel es representado por un bit, por lo tanto, cada pixel puede tomar sólo dos posibles valores: apagado(0) o encendido(1); generalmente apagado corresponde al fondo y encendido para los objetos en la imagen.

Ejemplos de formas binarias pueden ser señalamientos como se ve en la Figura 2.13, las letras del presente texto o algunos tipos de mapas como puede apreciarse en la Figura 2.14.

Figura 2.13 Imagen binaria, Stop.

Figura 2.14 Imagen binaria, México.

2.1.7. Objetos binarios.

Definición 2.12 Objeto binario.

Un objeto binario o componente conexa binaria es un conjunto de pixeles $p_0, \ldots, p_n$ en una imagen binaria Ib con valor $Ib(p_i) = 1$ $(0 \leq i \leq n)$ en donde para todo p_i existe una ruta que lo conecta con p_j $(0 \leq j \leq n)$. En otras palabras, un objeto binario es un conjunto conectado de pixeles 1s en una imagen binaria. En la Figura 2.15 se aprecia un ejemplo de imagen binaria con varios objetos binarios en ella.

Figura 2.15 Imagen binaria con ocho objetos binarios, Italia.

Definición 2.13 Frontera.

Una i-frontera de un objeto binario Ob es un conjunto f de pixeles en Ob que conforman una i-cuerda cerrada, para i = 0, 1, 4, 8. En otras palabras, una frontera es un subconjunto de pixeles del objeto binario que no tienen relación de adyacencia con otros pixeles en todas las direcciones posibles.

Se dice que una frontera *separa* o *delimita* dos conjuntos *C1* y *C0* de pixeles, el conjunto *C1* es de pixeles 1s que conforman al objeto binario *Ob* del cual forma parte la frontera, el conjunto *C0* es de pixeles 0s, donde no existe alguna ruta de pixeles $p_0, \ldots, p_n$ donde $p_0 \in C0$ y $p_n \in C1$ y $p_j \notin f$ con $1 \leq j \leq n$.

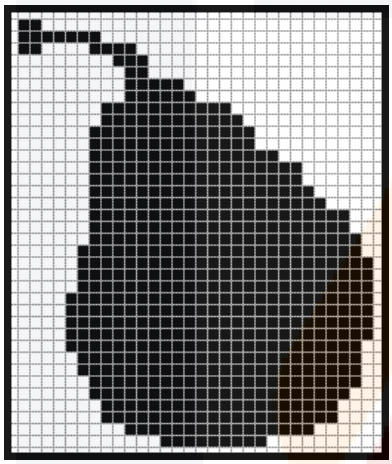


Figura 2.16 Imagen binaria, pera.

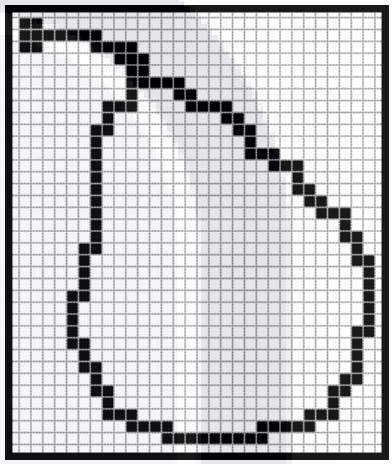


Figura 2.17 Frontera de imagen binaria, pera.

La frontera que se muestra en la Figura 2.16 separa los pixeles en color negro (1s) de los pixeles en color blanco (0s) de la Figura 2.17.

Definición 2.14 Hoyo binario.

Un hoyo binario es un conjunto de pixeles $p_0, \ldots, p_n$ en una imagen binaria Ib con valor $Ib(p_i) = 0$ $(0 \leq i \leq n)$ en donde para todo p_i existe una ruta que lo conecta con p_j $(0 \leq j \leq n)$ y es delimitado por una frontera. En otras palabras, un hoyo binario son los grupos de pixeles apagados (0s) que están rodeados por un objeto binario. En la Figura 2.18 y Figura 2.20 se muestran dos imágenes binarias, en la Figura 2.19 y Figura 2.21 se muestran las respectivas fronteras de esas imágenes.

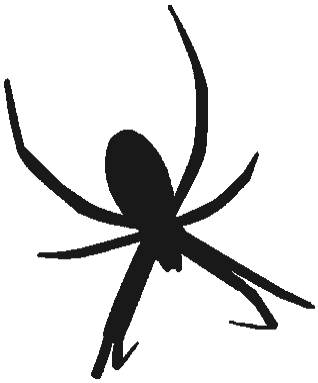


Figura 2.18 Imagen binaria, araña sin hoyos.

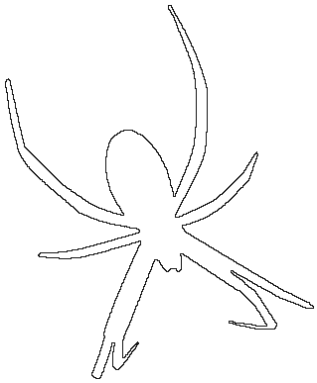


Figura 2.19 Frontera de imagen binaria, araña sin hoyos.

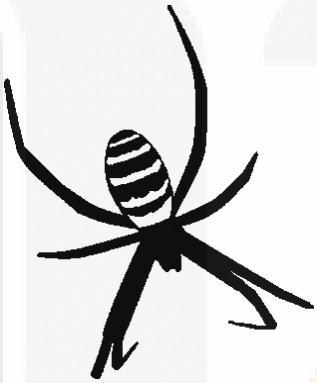


Figura 2.20 Imagen binaria, araña con hoyos.

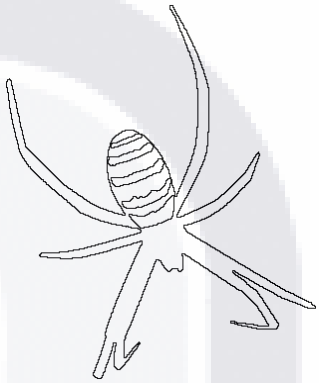


Figura 2.21 Frontera de imagen binaria, araña con hoyos.

Las fronteras que delimitan los hoyos de un objeto se denominan *fronteras internas* y a cada hoyo le corresponde una frontera interna; la frontera que delimita a los objetos binarios se denomina *frontera externa* y es única para cada objeto binario.

2.2. Teoría de Autómatas y Lenguajes Formales

2.2.1. Lenguajes Formales.

Definición 2.15 Alfabeto.

Llamaremos alfabeto, y lo denotaremos en general por una letra griega mayúscula a cualquier conjunto finito y no vacío de elementos que denominaremos símbolos o caracteres.

Así, por ejemplo, el alfabeto de los números arábigos lo podemos definir

$\Lambda = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

O el alfabeto de las letras vocales minúsculas del alfabeto romano es $\beta = \{a, e, i, o, u\}$

Definición 2.16 Cadena.

Una palabra, cadena, string o frase es cualquier secuencia finita de símbolos de un alfabeto, por ejemplo 10008000 es una palabra sobre Λ. La cadena que tiene cero símbolos (se puede formar a partir de cualquier alfabeto) se denomina cadena vacía *(García, et al, 1996)* y la denotaremos por λ.

Definición 2.17 Longitud de cadena.

Llamaremos longitud de una cadena al número de símbolos que contiene. De esta forma la palabra eaea sobre β tiene una longitud de 4.

2.2.2. Teoría de Autómatas

El autómata finito es un modelo formal de un sistema que trabaja con entradas y salidas discretas. El sistema puede estar en cualquiera de las configuraciones de un conjunto finito de configuraciones internas o *estados*. El estado del sistema resume la información concerniente a las anteriores entradas necesaria para determinar el comportamiento del sistema en las subsiguientes entradas.

Los autómatas finitos constituyen un modelo útil para muchos tipos importantes de hardware y software, tales como:

1. Software para el diseño y la verificación del comportamiento de circuitos digitales.
2. El "analizador léxico" de un compilador típico, esto es, la componente del compilador que descompone el texto inicial en unidades lógicas tales como identificadores, palabras reservadas y signos de puntuación.
3. Software para explorar grandes corpus de texto, como conjuntos de páginas web, o para descubrir las apariciones de ciertas palabras, frases u otros patrones.
4. Software para comprobar la corrección de cualquier tipo de sistemas que tengan un número finito de estados diferentes, como los protocolos de comunicación o los protocolos para el intercambio seguro de información.

Hay muchos sistemas o componentes, como los enumerados anteriormente, de los que se puede decir en todo momento que están en cierto "estado", entre un número finito de ellos. El objetivo de un estado es recordar la parte significativa de la historia del sistema. Dado que sólo disponemos de un número finito de estados, normalmente no es posible recordar la historia completa, por lo que el sistema ha de ser diseñado cuidadosamente para que sea

posible recordar los aspectos importantes y olvidar los que no lo son. La ventaja de tener sólo un número finito de estados es que el sistema se puede implementar con un volumen fijo de recursos. Por ejemplo, podría hacerse una implementación de hardware con un circuito, o mediante un programa sencillo que decida en función de un conjunto limitado de datos, o utilizando la situación actual del código para tomar cada decisión (Hopcroft, et al, 2002).

Los autómatas finitos se representan mediante diagramas en donde los estados se representan mediante círculos y las influencias externas que afectan al sistema mediante arcos con etiquetas entre los estados. Uno de los estados del autómata se denomina "estado inicial": es en el que se sitúa inicialmente el sistema, el cual, por convenio, se indica mediante la palabra *Inicio* y una flecha que señala a dicho estado. Frecuentemente es necesario indicar que uno o más estado son estados "finales", o estados "de aceptación", es conveniente que el sistema llegue a uno de esos estado después de una secuencias de entradas sucesivas, por convenio, dichos estados de aceptación suelen señalarse mediante un circulo doble. En la Figura 2.22 se muestra el modelo de autómata finito para un interruptor de encendido/apagado. La Figura 2.23 muestra un autómata finito para reconocer la palabra "Hola" en un texto.

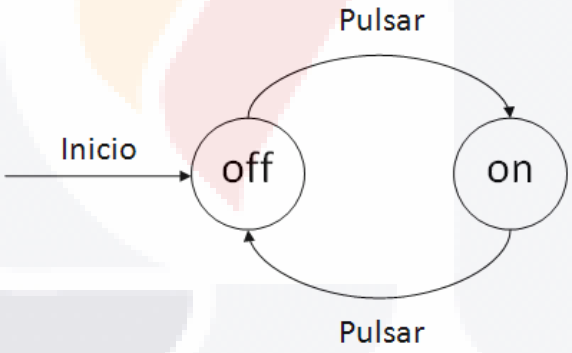


Figura 2.22 Autómata finito, encendido / apagado.

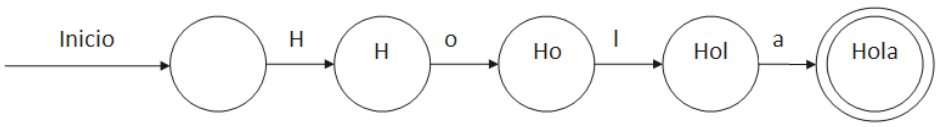


Figura 2.23 Autómata finito, hola.

Los autómatas finitos se dividen en dos tipos dependiendo de su control: determinista, lo que significa que el autómata no puede estar en más de un estado simultáneamente; y no determinista, lo que significa que puede estar en varios estados al mismo tiempo.

2.2.3. Autómatas Finitos Deterministas.

Definición 2.18 Autómata Finito Determinista.

Un autómata Finito Determinista (AFD) es una 5-Tupla de la forma de la Fórmula 2.8.

$$A=(Q, \sum, \delta, q_0, F)$$

Fórmula 2.8 Auómata Finito Determinista.

Siendo,

- Q un conjunto finito de estados
- $\sum$ un conjunto finito de símbolos llamado alfabeto
- δ: $Q \times \sum \rightarrow Q$ una función parcial llamada función de transición
- $q_0 \in Q$ el estadio inicial
- $F \subseteq Q$ el conjunto de estado finales.

Cuando la función de transición es total se dice que el autómata está completamente especificado o es completo.

Podemos entender un AFD como una memoria finita que se encuentra en un estado determinado del conjunto de estados Q, una cinta de entrada donde se escriben cadenas de $\sum$, y una cabeza de lectura que apunta a una determinada posición de la cinta de entrada (García,et al, 1996), tal como se muestra en la Figura 2.24.

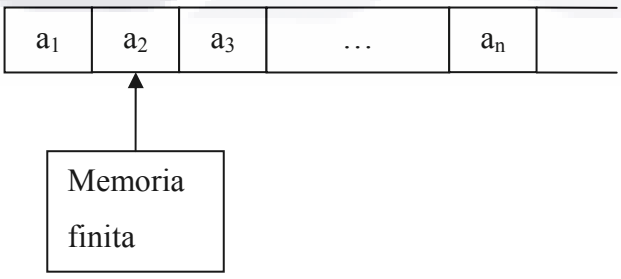


Figura 2.24 Autómata Finito Determinista.

El funcionamiento de esta máquina es el siguiente:

1. Inicialmente la memoria se encuentra en el estado q_0, la cinta de entrada contiene la cadena a analizar por la máquina, escrita a partir de su primera posición, y la cabeza de lectura se encuentra apuntando a esa primera posición de la cinta.
2. Suponiendo el autómata en un estado q y suponiendo que lee en la cinta de entrada un símbolo a la memoria finita pasa al estado δ(q, a) y mueve la cabeza lectura una posición a la derecha.
3. Cuando la cabeza lectora llega a la primera posición vacía de la cinta significa que la cadena de entrada ha sido completamente analizada por la máquina. Se dice que la cadena es aceptada si el estado al que se llega finalmente pertenece al conjunto de estado finales F.

2.2.4. Autómatas Finitos No Deterministas.

Definición 2.19 Autómata Finito No Determinista.

Un Autómata Finito No Determinista (AFN) es una 5-tupla de la forma de la Fórmula 2.8.

Siendo,

- $Q, \sum, q_0 \in Q$ y $F \subseteq Q$ el mismo conjunto de estados, alfabeto de entrada, estado inicial y conjunto de estados finales de la definición del AFD.
- $\delta: Q \times \sum \rightarrow 2^Q$ la función de transición, definida también como una función parcial. 2^Q denota el conjunto potencia o de las partes de Q.

De la misma manera que en el caso del AFD, podemos entender un AFN como un máquina con una memoria finita que lee en una cinta de entrada. Sin embargo, la memoria finita, en cada tiempo, puede estar en cualquier conjunto de estados. Cuando debe determinarse el nuevo estado, leído un símbolo de entrada, se toman todas las posibles decisiones manteniendo a la vez varias copias de la máquina funcionando al mismo tiempo sobre la misma entrada. En definitiva el modelo no determinista permite cero, una o más transiciones a partir de un estado con un determinado símbolo de entrada. Una cadena de entrada se dice que es aceptada por un AFN si existe una secuencia de transiciones, correspondientes a la cadena de entrada, que lleva del estado inicial a algún estado final (García, et al, 1996).

2.2.5. Autómatas Finitos no Deterministas para Búsqueda de Texto.

Supongamos que nos dan un conjunto de cadenas, que llamaremos *palabras clave*, y que queremos contar el número de apariciones de esas palabras. En aplicaciones como ésta, resulta útil diseñar un autómata finito no determinista que indique, al pasar a un estado de aceptación, que se ha leído una de las palabras clave. Se le proporciona al autómata finito no determinista el texto de un documento, un caracter cada vez. El autómata finito no determinista reconocerá la aparición de las palabras en el texto. Existe una forma simple para que un autómata finito no determinista reconozca un conjunto de palabras clave (Hopcroft, et al, 2002).

1. Hay un estado inicial con una transición a sí mismo para cada símbolo de entrada, es decir, para cada caracter ASCII imprimible si estamos examinando texto. Intuitivamente, el estado inicial representa la "conjetura" de que todavía no hemos comenzado a leer una de las palabras clave, incluso aunque hayamos leído algunas letras de una de esas palabras (Hopcroft, et al, 2002).
2. Para cada palabra clave $a_1, a_2, \ldots, a_k$, hay k estados, digamos $q_1, q_2, \ldots, q_k$. Existe una transición desde el estado inicial a q_1 con el símbolo a_1, una transición de q_1 a q_2 con el símbolo a_2, y así sucesivamente. El estado q_k es un estado de aceptación que indica que la palabra clave $a_1, a_2, \ldots, a_k$ ha sido encontrada (Hopcroft, et al, 2002).

Por ejemplo, supongamos que queremos diseñar un autómata finito no determinista que reconozca las apariciones de las palabras web y ebay. La figura 2.32 muestra el diagrama de transiciones del autómata finito no determinista construido utilizando las reglas indicadas. El estado 1 es el estado inicial. Se utiliza $\sum$ para representar el conjunto de todos los caracteres ASCII imprimibles. Los estados 2 a 4 están encargados de reconocer web, mientras que los estados 5 a 8 reconocen ebay.

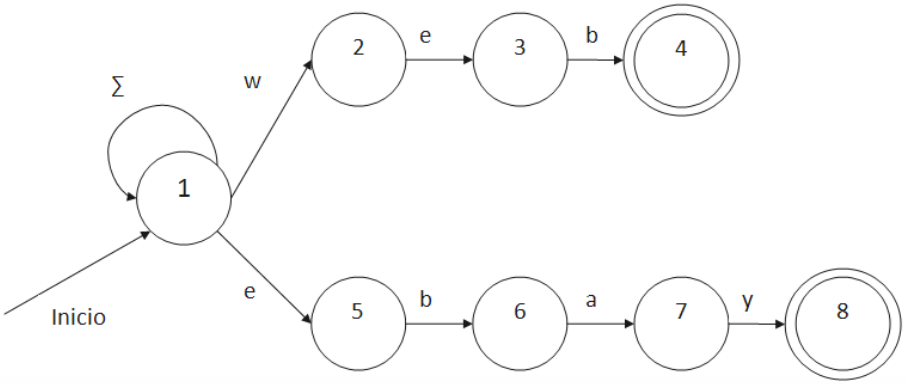


Figura 2.25 Autómata finito, palabras.

2.3. Códigos de cadena.

Definición 2.20 Código de cadena.

Los códigos de cadena son palabras de determinado alfabeto que se utilizan para representar un conjunto de puntos, que constituyen una línea, que puede ser recta o curva, por lo tanto también pueden ser usados para representar una frontera cuando el pixel final de la curva es vecino del pixel inicial.

Las técnicas de códigos de cadena son usadas ampliamente para representar formas binarias ya que permiten preservar la información además de permitir una reducción considerable en los datos (Sánchez-Cruz, et al, 2007).

En la codificación de objetos binarios mediante códigos de cadena se sigue el principio de localizar los objetos en la imagen binaria escaneándola en orden de línea mayor hasta encontrar el pixel del objeto que se encuentre más arriba y a la izquierda, tal pixel pertenece a la frontera externa, una vez encontrado se recorren los pixeles de dicha frontera f en sentido de las manecillas de reloj para la codificación de la parte externa del objeto, después se buscan los hoyos escaneando el objeto binario en orden de línea mayor, para los hoyos o partes internas las fronteras se recorren en sentido contrario de las manecillas del reloj. En la Figura 2.26 se presenta una imagen binaria con un hoyo y se aprecian los sentidos de dirección para la codificación de las fronteras.

Figura 2.26 Imagen binaria, taza, recorrido de las fronteras.

Al hacer el recorrido de la curva que representa la frontera del objeto binario, dos pixeles consecutivos $p_{(i-1)}$, $p_{(i)}$ con coordenadas $(x_{(i-1)}, y_{(i-1)})$ y (x_i, y_i) en una frontera conforman un vector unitario en una de las direcciones x, y del plano cartesiano, donde las direcciones y signos de los vectores $\hat{\jmath}$ y $-\hat{\jmath}$ se han invertido considerando el sistema de coordenadas de pixeles.

Se generan los siguientes vectores dependiendo de las coordenadas de $p_{(i-1)}$ y de $p_{(i)}$ en una relación 1-Adyacente (A_1) o 4-Adyacente (A_4).

$$SA_4 = \{ \hat{\imath}, \hat{\jmath}, -\hat{\imath}, -\hat{\jmath} \}$$

$\hat{\imath}$	cuando	$x_i > x_{(i-1)}$ y $y_i = y_{(i-1)}$	en dirección *este*
$\hat{\jmath}$	cuando	$x_i = x_{(i-1)}$ y $y_i < y_{(i-1)}$	en dirección *norte*
$-\hat{\imath}$	cuando	$x_i < x_{(i-1)}$ y $y_i = y_{(i-1)}$	en dirección *oeste*
$-\hat{\jmath}$	cuando	$x_i = x_{(i-1)}$ y $y_i > y_{(i-1)}$	en dirección *sur*

La Figura 2.27 muestra el recorrido de la frontera de una forma binaria en relación A_4, en la Tabla 2.1 se ve el orden de visita de los pixeles de la Figura 2.27.

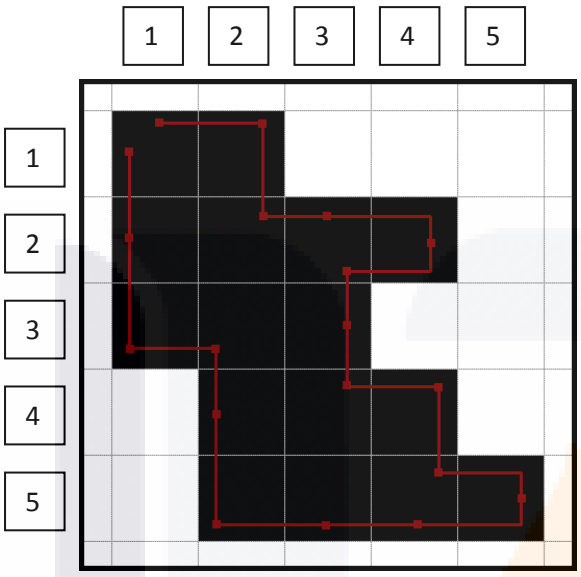


Figura 2.27 Imagen binaria, aleatoria, recorrido de frontera relación A4.

Orden visita	Coordenadas	Vector
0	(1, 1)	
1	(2, 1)	$\hat{\imath}$
2	(2, 2)	$-\hat{\jmath}$
3	(3, 2)	$\hat{\imath}$
4	(4, 2)	$\hat{\imath}$
5	(3, 2)	$-\hat{\imath}$
6	(3, 3)	$-\hat{\jmath}$
7	(3, 4)	$-\hat{\jmath}$
8	(4, 4)	$\hat{\imath}$
9	(4, 5)	$-\hat{\jmath}$
10	(5, 5)	$\hat{\imath}$
11	(4, 5)	$-\hat{\imath}$
12	(3, 5)	$-\hat{\imath}$
13	(2, 5)	$-\hat{\imath}$
14	(2, 4)	$\hat{\jmath}$
15	(2, 3)	$\hat{\jmath}$
16	(1, 3)	$-\hat{\imath}$
17	(1, 2)	$\hat{\jmath}$
18	(1, 1)	$\hat{\jmath}$

Tabla 2.1 Recorrido de frontera, relación A4.

Se generan los siguientes vectores dependiendo de las coordenadas de $p_{(i-1)}$ y de p_i en una relación 0-Adyacente (A_0) u 8-Adyacente (A_8).

$$SA_8 = \{\hat{\imath},\ \hat{\imath}+\hat{\jmath},\ \hat{\jmath},\ -\hat{\imath}+\hat{\jmath},\ -\hat{\imath},\ -\hat{\imath}-\hat{\jmath},-\hat{\jmath},\ \hat{\imath}-\hat{\jmath}\}$$

$\hat{\imath}$	cuando	$x_i > x_{(i-1)}$ y $y_i = y_{(i-1)}$	en dirección *este*
$\hat{\imath}+\hat{\jmath}$	cuando	$x_i > x_{(i-1)}$ y $y_i < y_{(i-1)}$	en dirección *noreste*
$\hat{\jmath}$	cuando	$x_i = x_{(i-1)}$ y $y_i < y_{(i-1)}$	en dirección *norte*
$-\hat{\imath}+\hat{\jmath}$	cuando	$x_i < x_{(i-1)}$ y $y_i < y_{(i-1)}$	en dirección *noroeste*
$-\hat{\imath}$	cuando	$x_i < x_{(i-1)}$ y $y_i = y_{(i-1)}$	en dirección o*este*
$-\hat{\imath}-\hat{\jmath}$	cuando	$x_i < x_{(i-1)}$ y $y_i > y_{(i-1)}$	en dirección *suroeste*
$-\hat{\jmath}$	cuando	$x_i = x_{(i-1)}$ y $y_i > y_{(i-1)}$	en dirección *sur*
$\hat{\imath}-\hat{\jmath}$	cuando	$x_i > x_{(i-1)}$ y $y_i > y_{(i-1)}$	en dirección *sureste*

La Figura 2.28 muestra el recorrido de la frontera de una forma binaria en relación A_8, en la Tabla 2.2 se ve el orden de visita de los pixeles de la Figura 2.28.

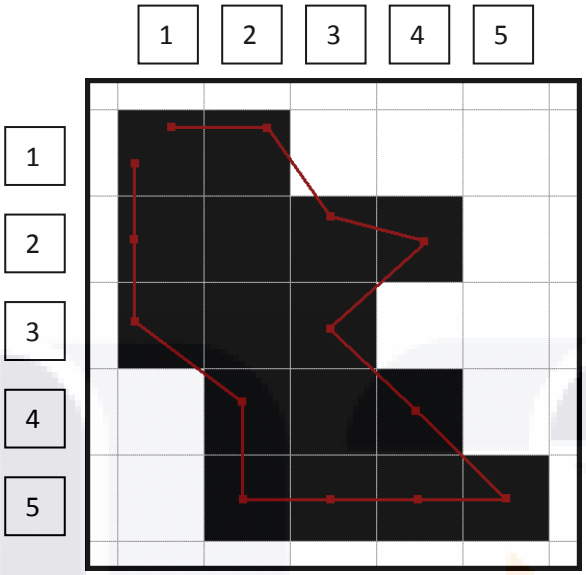


Figura 2.28 Imagen binaria, recorrido frontera, relación A8.

Orden visita	Coordenadas	Vector
0	(1, 1)	
1	(2, 1)	$\hat{\imath}$
2	(3, 2)	$\hat{\imath}$-$\hat{\imath}$
3	(4, 2)	$\hat{\imath}$
4	(3, 3)	-$\hat{\imath}$-$\hat{\imath}$
5	(4, 4)	$\hat{\imath}$-$\hat{\imath}$
6	(5, 5)	$\hat{\imath}$-$\hat{\imath}$
7	(4, 5)	-$\hat{\imath}$
8	(3, 5)	-$\hat{\imath}$
9	(2, 5)	-$\hat{\imath}$
10	(2, 4)	$\hat{\imath}$
11	(1, 3)	-$\hat{\imath}$+$\hat{\imath}$
12	(1, 2)	$\hat{\imath}$
13	(1, 1)	$\hat{\imath}$

Tabla 2.2 Recorrido frontera, relación A8.

Al hacer el recorrido de la frontera f del objeto binario se van registrando los símbolos del código de cadena dependiendo de los vectores generados. Según el código de cadena usado, los sucesivos pixeles se toman en relación 4-Adyacente o en relación 8-Adyacente.

El primer método para representar curvas digitales mediante códigos de cadena fue desarrollado por (Freeman, 1961). Para la definición del código de cadena de Freeman, se tiene en cuenta la localización de un pixel (x, y) y sus 8 vecinos en las direcciones cuantizadas de 45° para relación 8-Adyacente, a cada una de dichas direcciones se les asigna un valor numérico y así al este, noreste, norte, noroeste, oeste, suroeste, sur y sureste les corresponden 0, 1, 2, 3, 4, 5, 6 y 7 respectivamente, este código se conoce como FCCE, los vectores generados por este código pueden verse en la Figura 2.30. También se puede considerar para relación 4-Adyacente, en donde las direcciones posibles difieren en 90° entre sí, y al este, norte, oeste y sur les corresponden los caracteres 0, 1, 2 y 3 respectivamente, este código se conoce como FCCF, los vectores generados por este código pueden verse en la Figura 2.29.

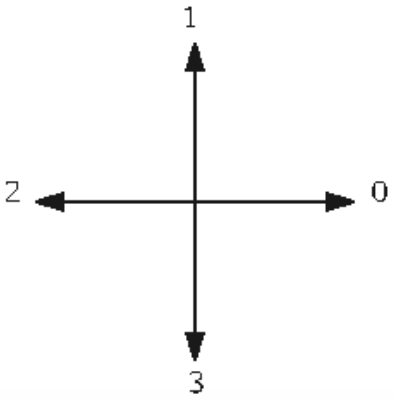


Figura 2.29 Vectores del código FCCF.

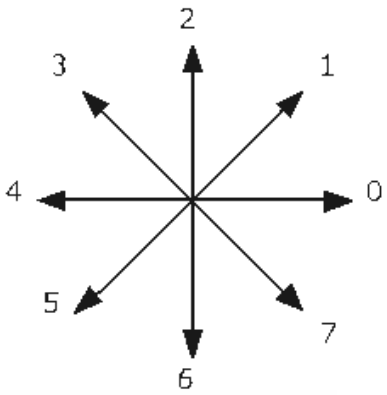


Figura 2.30 Vectores del código FCCE.

(Freeman, 1974) ha declarado que "en general, un proyecto de codificación para estructuras lineales debe satisfacer tres objetivos: (1) debe preservar fielmente la información de interés; (2) debe permitir un almacenamiento compacto y ser conveniente para desplegarse; y (3) debe facilitar cualquier procesamiento requerido. Los tres objetivos están de alguna manera en conflicto entre sí, y cualquier código necesita que se llegue a un equilibrio entre los tres".

El código de Freeman no es el único que se ha desarrollado, en (Papert, 1972) se propuso uno de los códigos de cadena más simple, usando sólo 2 símbolos {0, 1}, en este código se sigue un recorrido con cambios de dirección relativo a través del contorno, y el cero representa giros hacia la derecha y el 1, a la izquierda.

Una variante a los códigos de Freman fue propuesta por (Kui-Liu & Zalik, 2005), donde se consideraron los mismos cambios de dirección, pero en lugar de ser absolutos, son relativos a la dirección seguida al recorrer la curva discreta. Así se asignaron los símbolos del código: 0 para cuando no hay cambio de dirección, 1 para +45° de cambio de dirección, 2 para -45°, 3 para +90°, 4 para -90°, 5 para +135°, 6 para -135° y 7 para 180°. Tal código es conocido como DFCCE y el correspondiente código para relación 4-Adyacente se llama DFCCF.

En (Bribiesca, 1999) propuso un código de cadena basado en el número de vértices que están en contacto en los pixeles de la frontera de la forma binaria, se asigna el símbolo 0 cuando en determinado vértice sólo hay un pixel, esto se da en esquinas exteriores; el símbolo 1 cuando hay dos pixeles en contacto, este caso se presenta en las secciones rectas del contorno; y el símbolo 2 cuando hay tres pixeles en contacto, lo cual sucede en esquinas interiores al recorrer la frontera del objeto binario.

2.3.1. Código 3OT.

En (Bribiesca, 2000) se propuso un código de cadena para representar curvas y cuerdas en tres dimensiones, dicho código está basado en el cambio relativo de direcciones ortogonales, y consiste de 5 símbolos, este código es conocido como 5OT.

En (Sánchez-Cruz & Rodríguez-Dagnino, 2005) se propuso un código de cadena para representar formas binarias sin pérdida de información, basado en un código de cadena similar al 5OT de Bribiesca, pero aplicado en dos dimensiones, este código tiene un alfabeto de tres símbolos y se conoce como 3OT.

Para su codificación se localizan los objetos binarios en la imagen, escaneándola en orden de línea mayor. Las fronteras de los objetos se recorren en sentido de las manecillas del reloj para la frontera externa y en sentido contrario para las internas. El recorrido de los pixeles de las fronteras se hace siguiendo una relación 4-Adyacente, de modo que pueden generarse 4 posibles vectores $SA_4 = \{ \hat{\imath}, \hat{\jmath}, -\hat{\imath}, -\hat{\jmath}\}$ donde $\hat{\imath}$, $\hat{\jmath}$ son vectores unitarios, al pasar de un pixel a otro.

El alfabeto del código 3OT, se define así $\Omega = \{0, 1, 2\}$. Tomando cualquier vector de $\vec{S}_i$ como un vector de referencia en uno de los múltiples pasos necesarios al recorrer la frontera *f*: $\vec{S}_i \equiv \vec{S}_{ref}$.

Al vector $\vec{S}_{ref}$ se le llamará *vector de referencia o vector base*.

Al vector $\vec{S}_{ref+1}$ se le llamará *vector pivote*.

Al vector $\vec{S}_{ref+z+2}$ se le llamará *vector de cambio*, donde *Z* es cero o entero positivo, y su valor depende del número de pasos dados en una misma dirección.

Los símbolos del alfabeto Ω se generan de esta manera:

- 0: Este símbolo se genera cuando el vector de cambio es igual al vector pivote.

  vector de cambio = vector pivote

- 1: Generado cuando el vector de cambio es diferente del vector pivote y el vector de cambio es igual al vector de referencia. Una vez generado el "1", el vector de referencia toma el valor del vector pivote y el vector pivote el valor del vector de cambio.

vector de cambo ≠ vector pivote

y

vector de cambio = vector de referencia

- 2: Generado cuando el vector de cambio es diferente del vector pivote y el vector de cambio es diferente del vector de referencia. Una vez generado el "2", el vector de referencia toma el valor del vector pivote y el vector pivote el valor del vector de cambio.

vector de cambo ≠ vector pivote

y

vector de cambio ≠ vector de referencia

Entonces, cada símbolo de los mostrados en la Figura 2.31 se interpreta así:

$$\textbf{0: } \vec{S}_{ref+1} = \vec{S}_{ref+z+2} \quad \textbf{si} \quad \vec{S}_{ref} \bullet \vec{S}_{ref+1} = 0$$

$$\textbf{1: } \vec{S}_{ref} = \vec{S}_{ref+z+2} \quad \textbf{si} \quad \vec{S}_{ref} \bullet \vec{S}_{ref+1} = 0$$

$$\textbf{2: } \vec{S}_{ref} = -\vec{S}_{ref+z+2} \quad \textbf{si} \quad \vec{S}_{ref} \bullet \vec{S}_{ref+1} = 0$$

donde • denota el producto punto y Z es cero o entero positivo, y su valor depende del número de pasos dados en una misma dirección.

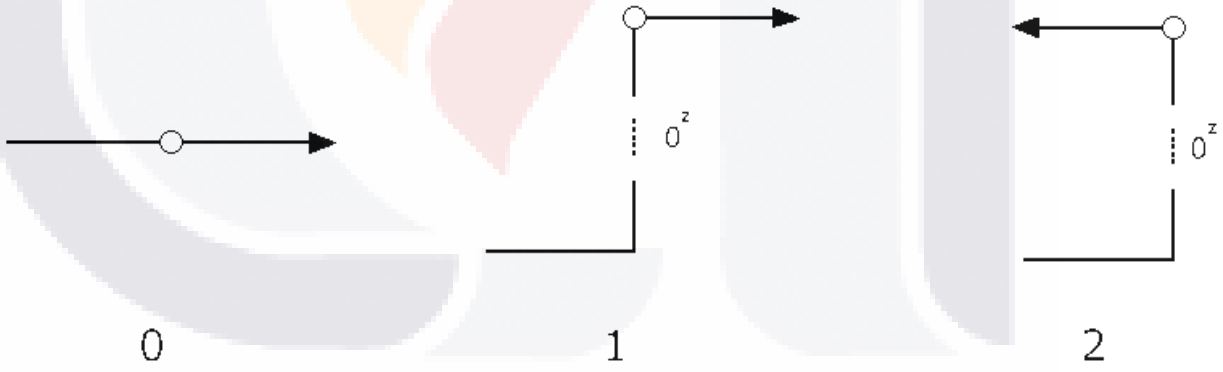


Figura 2.31 Símbolos, código 3OT.

En la codificación de una forma binaria simple, las coordenadas representan la 0-Celda (vértice) superior izquierdo de cada pixel.

En la Figura 2.32 se aprecia el recorrido de los pixeles al codificar en 3OT una imagen binaria aleatoria, en la Tabla 2.3 se muestra una prueba de escritorio.

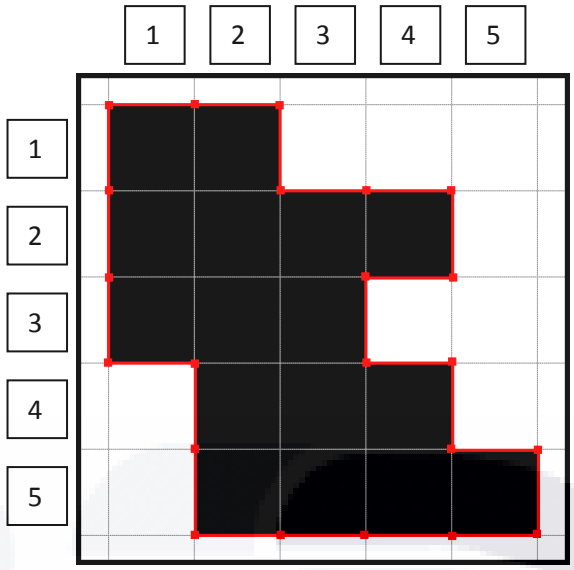


Figura 2.32 Imagen binaria, aleatoria, ejemplo de codificación en 3OT.

Orden de visita	Coordenadas	REFERENCIA	PIVOTE	CAMBIO	Símbolo generado
0	(2, 1)	j	i	i	0
1	(3, 1)	j	i	-j	2
2	(3, 2)	i	-j	i	1
3	(4, 2)	-j	i	i	0
4	(5, 2)	-j	i	-j	1
5	(5, 3)	i	-j	-i	2
6	(4, 3)	-j	-i	-j	1
7	(4, 4)	-i	-j	i	2
8	(5, 4)	-j	i	-j	1
9	(5, 5)	i	-j	i	1
10	(6, 5)	-j	i	-j	1
11	(6, 6)	i	-j	-i	2
12	(5, 6)	-j	-i	-i	0
13	(4, 6)	-j	-i	-i	0
14	(3, 6)	-j	-i	-i	0
15	(2, 6)	-j	-i	j	2
16	(2, 5)	-i	j	j	0
17	(2, 4)	-i	j	-i	1
18	(1, 4)	j	-i	j	1
19	(1, 3)	-i	j	j	0
20	(1, 2)	-i	j	j	0
21	(1, 1)	-i	j	i	2

Tabla 2.3 Generación de código 3OT, imagen binaria aleatoria.

Gracias a sus características, el código 3OT se ha utilizado para el procesamiento de los objetos binarios, no sólo en su análisis, como el trabajo de (Sánchez-Cruz, 2006) que ideó un método para la detección de puntos dominantes en objetos binarios identificando patrones en las palabras del código 3OT.

2.4. Teoría de la información.

Aunque la información es medida a veces por caracteres o dígitos, en la teoría de la información se mide por bits.

Suponiendo que se tira una moneda un millón de veces y se anota la secuencia de resultados. Si se desea comunicar esta secuencia a otra persona, ¿Cuántos bits tomará el mensaje?, si es una moneda con la misma probabilidad para las dos caras, entonces, cada tirada requiere un bit de información. La secuencia entera requerirá un millón de bits.

Pero, suponiendo que la moneda tiene 0.75 de probabilidad de una cara y 0.25 de probabilidad en la otra. Entonces, la secuencia completa puede enviarse en 811,300 bits en promedio. Esto parece implicar que cada tirada de la moneda requiere sólo 0.8113 bits para ser transmitida. ¿Cómo puede ser que una tirada requiere menos de un bit para ser transmitida si el único lenguaje disponible son ceros y unos? Obviamente no se puede. Pero si el objetivo es transmitir la secuencia completa de tiradas y la distribución de probabilidad está parcializada en alguna manera, entonces se puede usar el conocimiento de la distribución para usar un código más eficiente. Otra manera de verlo es que una secuencia de tiros parcializados contiene menos información que una secuencia de tiros no parcializados, así que debe tomar menos bits para transmitirse.

Veamos otro ejemplo. Suponiendo que la probabilidad de obtener águila es de 0.001 y sello es de 0.999. En un millón de tiros de esta moneda, se puede esperar alrededor de 1,000 águilas. En lugar de transmitir el resultado de cada tiro, se puede transmitir sólo el número de la tirada en que haya salido águila; el resto de las tiradas se asume que calló sello. Cada tirada tiene una posición en la secuencia: un número entre 1 y 1,000,000. Un número en ese rango puede codificarse usando sólo 20 bits. Así que si se transmiten 1,000 números de 20 bits, se habrá transmitido toda la información contenida en la secuencia de un millón de tiradas usando sólo alrededor de 20,000 bits.

Puede incluso mejorarse esto, codificando las posiciones absolutas de águilas en la secuencia toma 20 bits por águila, pero esto nos permite transmitir las águilas en cualquier orden. Si se prefiere transmitir las águilas de manera sistemática, haciéndolo en orden del inicio al fin, entonces en lugar de transmitir las posiciones absolutas puede sólo codificar la distancia entre cada águila, lo que requiere menos bits. Por ejemplo, si las primeras 4 águilas ocurrieron en las posiciones 502, 1609, 2454 y 2607, entonces, su codificación como la

distancia entre águilas sería 502, 1107, 845, 153. En promedio, la distancia entre 2 águilas será de 1,000 tiradas; sólo raramente la distancia excederá 4,000 tiradas. Los números en el rango de 1 a 4,000 pueden codificarse en 12 bits. Así que, usando esta convención para la codificación, una secuencia de un millón de tiradas que contiene alrededor de 1,000 águilas puede ser transmitida usando sólo 12,000 bits. Así, una tirada toma sólo 0.012 bits para transmitirse, lo que de nuevo no tiene sentido usando el alfabeto de ceros y unos.

¿Puede inventarse una mejor codificación?, ¿cuál será el límite de la codificación óptima?. Para este caso el límite es de 0.0114 bits por tirada.

El contenido de información de una secuencia es definido como el número de bits requeridos para transmitir la secuencia usando la codificación óptima. Siempre se tiene la libertad de usar una codificación menos eficiente, lo que requerirá más bits, pero eso no incremente la cantidad de información transmitida (Touretzky, 2004).

2.4.1. Entropía.

En (Shannon, 1948) se formuló la teoría de la compresión de datos. Shannon estableció que hay un límite fundamental para la compresión de datos sin pérdida de información. Este límite, llamado nivel de entropía se denota por la letra *H*. El valor exacto de *H* depende de la fuente de información, más específicamente de su naturaleza estadística. Es posible comprimir la fuente sin perder información con un nivel de compresión cercano a *H*. Es matemáticamente imposible hacerlo mejor que *H*(Data-Compression.com A website devotede to the principles and practice of data compression, 2000 - 2007).

Sea $n \in \mathbb{N}$ y $X = \{x_1, \ldots, x_n\}$ un conjunto finito con una distribución de probabilidad de $p = (p_1, \ldots, p_n)$. La entropía *H* se define como en la Fórmula 2.9.

$$H(X) = -\sum_{j=1}^{n} p_j log p_j$$

Fórmula 2.9 Entropía.

2.4.2. Códigos de longitud variable.

Los códigos de longitud variable o métodos estadísticos de compresión aprovechan el conocimiento de la frecuencia de aparición de los símbolos en las cadenas para asignarle menor cantidad de bits a aquellos que aparecen con más frecuencia y asigna más bits entre menos frecuente sea la aparición.

Permiten comprimir cadenas exitosamente y descomprimirlas sin pérdida de información. Con la correcta estrategia de codificación se podría comprimir casi arbitrariamente cerca a la entropía *H* de la cadena.

Por ejemplo, Suponiendo un alfabeto

$$X = \{A, B, C, D, E, F, G\}$$

y una cadena de símbolos

$$M = \{d_i \in X \mid i = 1, \ldots, 100\}$$

Con una cantidad de apariciones de cada elemento de *X* en *M* de

A = 45	D = 16
B = 15	E = 4
C = 14	F = 6

La mayoría de los códigos binarios de representación de símbolos, como el ASCII, son de longitud fija, en ASCII se usan 8 bits para representar $2^8 = 256$ símbolos. Para el ejemplo se necesitan 3 bits por símbolo. El código puede representarse así:

A = 000	D = 011
B = 001	E = 100
C = 010	F = 101

con este código de longitud fija, la cadena *M* será almacenada en 3 * 100 = 300 bits.

Usando un código de longitud variable, como la siguiente asignación:

A = 0	D = 111
B = 101	E = 1101
C = 100	F = 1100

se logra almacenar la cadena *M* en

1*45 + 3*15 + 3*14 + 3*16 + 4*4 + 4*6 = 220 bits

2.5. Puntos Dominantes en fronteras.

La representación eficiente de objetos en una imagen puede ser muy útil en varios campos, tales como reconocimiento de objetos, codificación de video basada en objetos, descripción de formas y compresión de datos. Los objetos pueden ser representados por su frontera o por su forma interior. La representación por frontera es más eficiente y preserva la forma completa. Las fronteras pueden ser representadas de una manera aproximada mediante diversos esquemas de codificación. La aproximación poligonal es un esquema de codificación en el que el subgrupo de puntos de curva (conocidos como puntos dominantes) es determinado y unido con segmentos de líneas rectas. Claramente, tal esquema tiene pérdida de información ya que presenta algo de distorsión o error de aproximación. La presencia de error es usualmente permisible mientras que la distorsión visual sea considerada mínima (Masood, 2008a). Este tipo de representación ha ganado popularidad debido a su simplicidad, localidad, generalidad y compacidad. También simplifica el análisis de imágenes al proveer un grupo de puntos característicos que contienen información casi completa de una frontera. También causa una alta reducción de datos y eficiencia para subsecuentes procedimientos (Masood, 2008b). La Figura 2.33 muestra una imagen binaria con un objeto binario, en la Figura 2.34 se aprecia la misma imagen binaria con puntos dominantes marcados con rojo; la Figura 2.35 muestra el polígono sobrepuesto a la frontera del objeto original.

Figura 2.33 Imagen binaria, delfín.

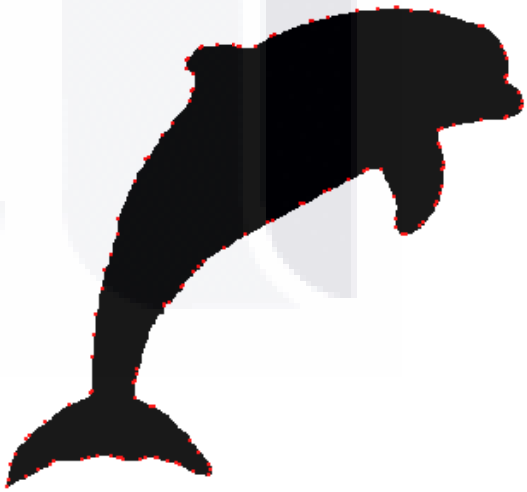


Figura 2.34 Imagen binaria con puntos dominante, delfín.

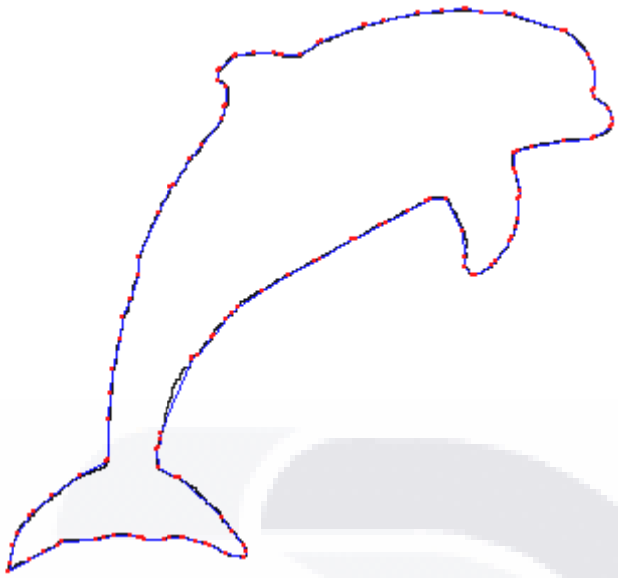


Figura 2.35 Frontera de imagen binaria con puntos dominantes y su polígono aproximado, delfín.

Capítulo 3

CODIFICACIÓN Y DECODIFICACIÓN EN 3OT

En el presente capítulo se explica en qué consisten los algoritmos desarrollados para la codificación y decodificación de imágenes con objetos binarios a código de cadena 3OT. También, incluye una explicación acerca del algoritmo de compresión aritmético y cómo se aplicó a las cadenas de caracteres en 3OT.

3.1. Cambios de dirección.

Antes de comenzar la explicación del algoritmo y las clases creadas es conveniente presentar los 4 tipos de cambio de dirección entre el vector pivote y el vector de cambio: cambio simple, cambio diagonal, sin cambio, en dirección opuesta.

Definición 3.1 Cambio de dirección simple.

Si el vector de cambio es el mismo que el vector pivote rotado 90° en sentido de las manecillas del reloj. Es cuando al cambiar de dirección se hace sobre el pixel actual. Con este tipo de cambio se cumple la condición de que el vector pivote es diferente al vector de cambio, por lo tanto, depende del vector base si se va a representar con un 1 o un 2 del alfabeto Ω este cambio. Ejemplo: Si el vector pivote va hacia el este, un cambio de dirección simple sería como lo representado en la Figura 3.1.

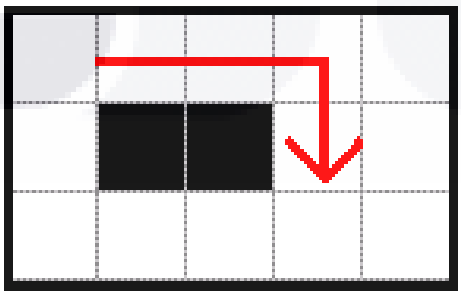


Figura 3.1 Cambio de dirección simple.

Definición 3.2 Cambio de dirección diagonal.

Si el vector de cambio es el mismo que el vector pivote rotado 90° en sentido contrario de las manecillas del reloj. Es cuando al cambiar de dirección se ven implicados otros 2 pixeles,

además del pixel actual. Con este tipo de cambio se cumple la condición de que el vector pivote es diferente al vector de cambio, por lo tanto, depende del vector base si se va a representar con un 1 o un 2 del alfabeto Ω este cambio. Ejemplo: Si el vector pivote va hacia el este, un cambio de dirección diagonal sería como lo representado en la Figura 3.2.

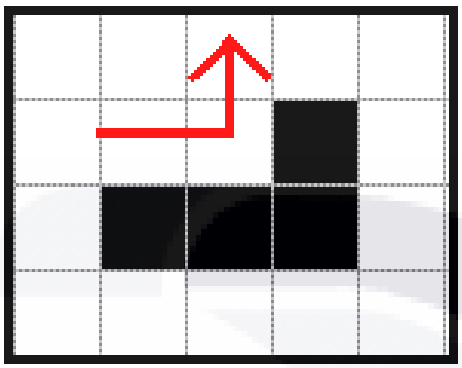


Figura 3.2 Cambio de dirección diagonal.

Definición 3.3 Cambio de dirección nulo.

Es cuando no hay cambio de dirección entre el vector pivote y el vector de cambio. Cuando sucede un cambio de dirección de este tipo se representa con un símbolo 0 del alfabeto Ω. Ejemplo: Si el vector pivote va hacia el este, un cambio de dirección nulo sería como lo representado en la Figura 3.3.

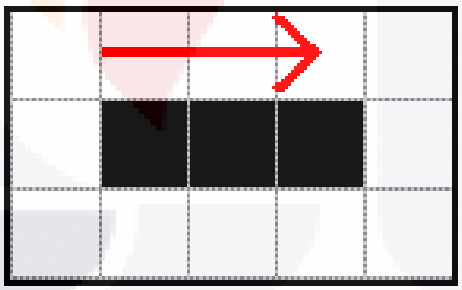


Figura 3.3 Cambio de dirección nulo.

Definición 3.4 Cambio de dirección negativo.

Es cuando el vector de cambio es igual al negativo del vector pivote. Este caso no se presenta en este experimento.

Como se ha visto, hay 3 cambios posibles para cada una de las direcciones que puede tener el vector pivote y son combinaciones entre el vector pivote y el vector de cambio.

3.2. Codificación 3OT

La codificación el 3OT es un paso básico dentro de esta tesis, su algoritmo se basa en el recorrido de la frontera de los objetos u hoyos en imágenes binarias, para la correcta codificación es necesario conocer las posiciones de los siguientes pixeles en el recorrido de la frontera, al conocer dos tipos de posiciones de los sucesivos pixeles se puede asignar el siguiente caracter dentro del alfabeto Ω para la cadena.

Definición 3.5 Pixel vecino.

Cuando se recorre la frontera, se checa el estado del siguiente pixel, el pixel vecino respecto al actual es cualquiera que se encuentre en relación 4-Adyacente con este, es aquel en dirección hacia el este, norte, oeste o sur. Por lo regular se comparará el siguiente pixel en la dirección del vector pivote.

En la Figura 3.4 se muestran marcados con verdes los pixeles vecinos hacia las distintas direcciones posibles del pixel marcado con rojo, el pixel vecino hacia el norte está apagado; los pixeles vecinos hacia el este, sur y oeste están encendidos.

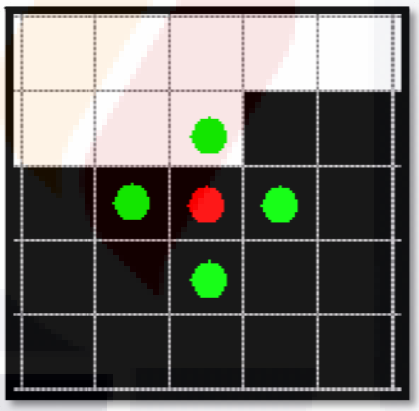


Figura 3.4 Imagen binaria, aleatoria, pixeles vecinos del actual.

Definición 3.6 Pixel diagonal.

El pixel diagonal con respecto al actual es aquel pixel trasladado dos posiciones consecutivas en relación 4-Adyacente; la primera posición es en cualquier dirección este, norte, oeste o sur, la segunda posición es una rotación en 90° de la primera posición.

En la Figura 3.5 se muestran los cuatro pixeles diagonales (verde) con respecto al pixel actual (rojo), El pixel diagonal al norte del actual está apagado; los pixeles diagonales al este, oeste y sur están encendidos.

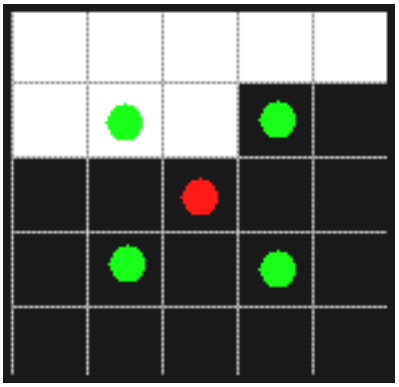


Figura 3.5 Imagen binaria, aleatoria, pixeles diagonales del actual.

En la Figura 3.6 se tiene una imagen binaria aleatoria, que será usada para mostrar los cambios de dirección en la codificación y decodificación del 3OT.

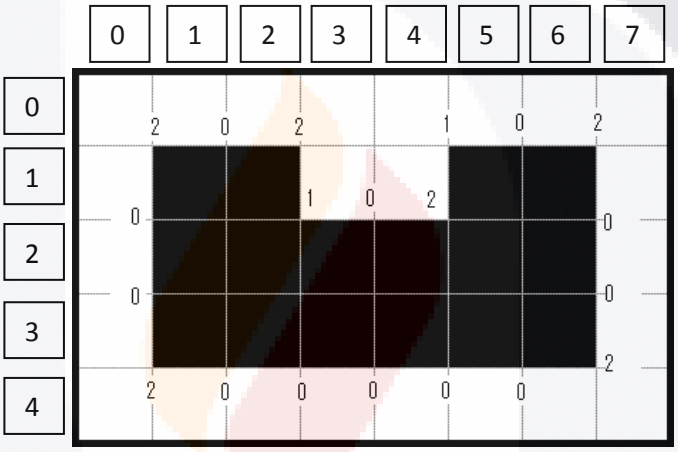


Figura 3.6 Imagen binaria, aleatoria, ejemplo cambios de dirección.

En la Tabla 3.1 se muestran los vectores para cada posición en la codificación de la imagen de la Figura 3.6, las coordenadas representan la 0-Celda (vértice) superior izquierdo de cada pixel.

Orden de visita	Coordenadas	REFERENCIA	PIVOTE	CAMBIO	Símbolo generado
0	(2, 1)	$\hat{\jmath}$	$\hat{\imath}$	$\hat{\imath}$	0
1	(3, 1)	$\hat{\jmath}$	$\hat{\imath}$	$-\hat{\jmath}$	2
2	(3, 2)	$\hat{\imath}$	$-\hat{\jmath}$	$\hat{\imath}$	1
3	(4, 2)	$-\hat{\jmath}$	$\hat{\imath}$	$\hat{\imath}$	0
4	(5, 2)	$-\hat{\jmath}$	$\hat{\imath}$	$\hat{\jmath}$	2
5	(5, 1)	$\hat{\imath}$	$\hat{\jmath}$	$\hat{\imath}$	1
6	(6, 1)	$\hat{\jmath}$	$\hat{\imath}$	$\hat{\imath}$	0
7	(7, 1)	$\hat{\jmath}$	$\hat{\imath}$	$-\hat{\jmath}$	2
8	(7, 2)	$\hat{\imath}$	$-\hat{\jmath}$	$-\hat{\jmath}$	0
9	(7, 3)	$\hat{\imath}$	$-\hat{\jmath}$	$-\hat{\jmath}$	0
10	(7, 4)	$\hat{\imath}$	$-\hat{\jmath}$	$-\hat{\imath}$	2
11	(6, 4)	$-\hat{\jmath}$	$-\hat{\imath}$	$-\hat{\imath}$	0

12	(5, 4)	-ĵ	-î	-î	0
13	(4, 4)	-ĵ	-î	-î	0
14	(3, 4)	-ĵ	-î	-î	0
15	(2, 4)	-ĵ	-î	-î	0
16	(1, 4)	-ĵ	-î	ĵ	2
17	(1, 3)	-î	ĵ	ĵ	0
18	(1, 2)	-î	ĵ	ĵ	0
19	(1, 1)	-î	ĵ	î	2

Tabla 3.1 Generación de código 3OT, imagen binaria aleatoria, ejemplo cambios de dirección.

Los diferentes símbolos pertenecientes al alfabeto Ω que conforman la cadena de símbolos del código 3OT son generados al comparar los vectores de referencia, pivote y de cambio generados al hacer el recorrido de las diferentes fronteras en la imagen. Los cambios de dirección de estos vectores se determinan verificando el estado del siguiente pixel y del pixel diagonal con respecto al pixel actual en dirección del vector pivote.

Los diferentes cambios de dirección se consiguen de esta manera:

- Cambio de dirección simple: Cuando el pixel vecino en dirección del vector pivote del pixel actual en el recorrido de la frontera esté apagado (0) significa un cambio de dirección simple. En la Figura 3.7 se aprecia un cambio de dirección simple, el vector pivote está en dirección hacia la derecha en el pixel marcado con rojo, y el pixel de la derecha está apagado.

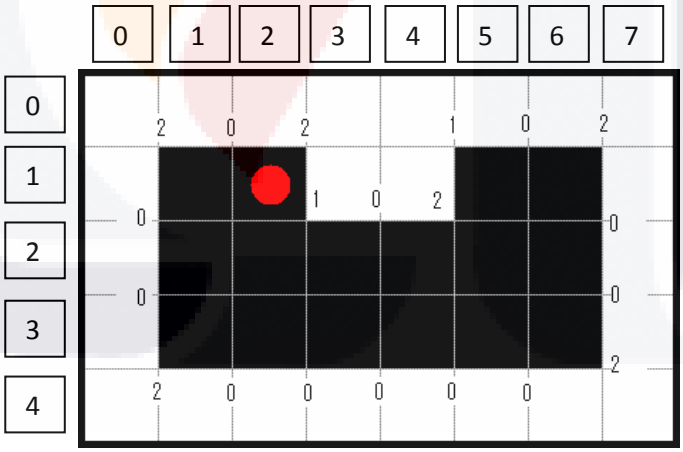


Figura 3.7 Imagen binaria, aleatoria, pixel vecino apagado.

- Cambio de dirección nulo: Quiere decir que el pixel vecino en dirección del vector pivote está encendido(1) y el pixel en diagonal en dirección del vector pivote está apagado(0). En la Figura 3.8 se aprecia un cambio de dirección nulo, el vector pivote

están en dirección hacia la derecha del pixel marcado con rojo, y el pixel de la derecha está encendido y el pixel en diagonal derecha está apagado.

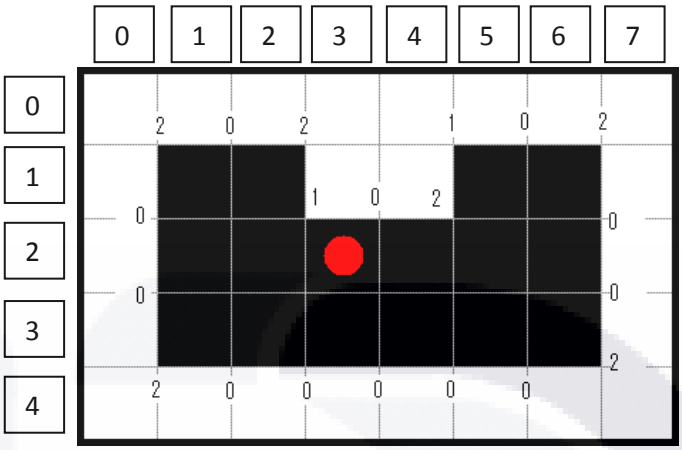


Figura 3.8 Imagen binaria, aleatoria, pixel vecino encendido y pixel diagonal apagado.

- Cambio de dirección diagonal: Quiere decir que el pixel vecino en dirección del vector pivote está encendido(1) y el pixel en diagonal en dirección del vector pivote está encendido(1). En la Figura 3.9 se aprecia un cambio de dirección diagonal, el vector pivote están en dirección hacia la derecha del pixel marcado con rojo, y el pixel de la derecha está encendido y el pixel en diagonal derecha está encendido.

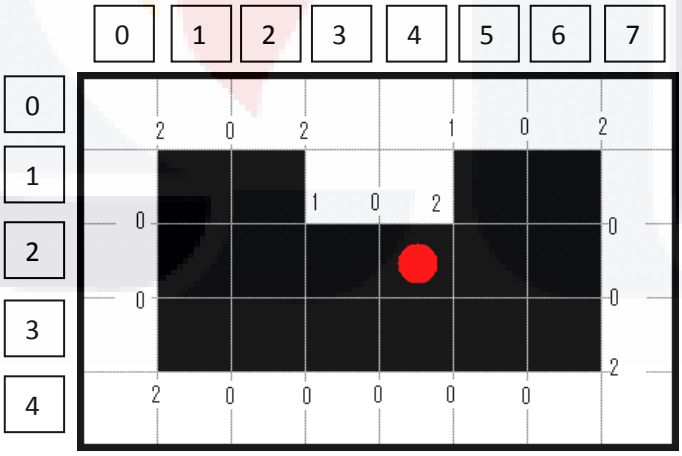


Figura 3.9 Imagen binaria, aleatoria, pixel vecino encendido y pixel diagonal encendido.

3.3. Rellenado de fronteras con cola de línea

Un procedimiento de apoyo que será usado en la codificación de varios objetos binarios con o sin hoyos en una imagen binaria será el rellenado de fronteras. El método usado para lograrlo es el conocido como cola de líneas, que consiste en almacenar en una cola una serie de líneas horizontales, estas líneas contienen los pixeles a partir de los cuáles se comenzará el rellenado de las siguientes líneas, las siguientes líneas se tomarán a partir de los vecinos norte y/o sur de estos pixeles.

Los pasos de este método de relleno se describen a continuación:

1. Se define un punto inicial arbitrario que se encuentre dentro de la frontera que se rellenará.
2. Se modifica el estado de los pixeles al nuevo valor a partir del punto inicial hacia la izquierda y derecha hasta encontrar pixeles que tengan el mismo color al que se quiere rellenar o que sean parte de la frontera.
3. Los pixeles que se han modificado de estado en el paso anterior conforman una línea horizontal, esta línea se introduce en una cola.
4. Se saca una línea de la cola y se analizan los vecinos norte y sur de cada pixel de la línea, si alguno de estos pixeles analizados es diferente al color a rellenar, entonces se procede con ellos de la misma manera que en el paso uno.

En la Figura 3.10 se presenta una frontera sin rellenar, en la Figura 3.11 el primer pixel a partir de donde se comenzará el rellenado.

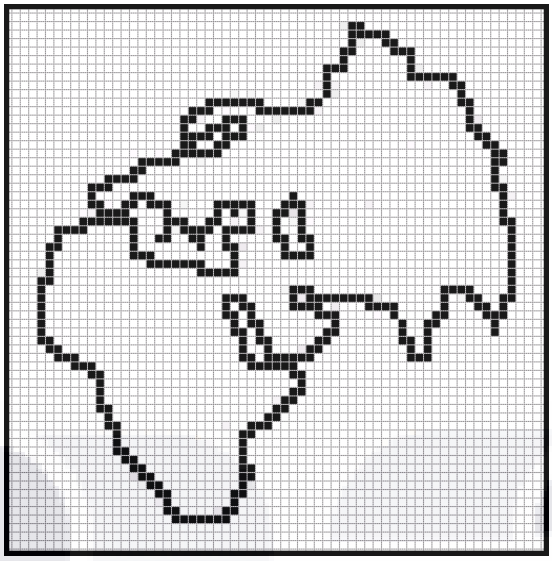


Figura 3.10 Frontera binaria, Euroasiaticoafricano, ejemplo rellenado, estado inicial.

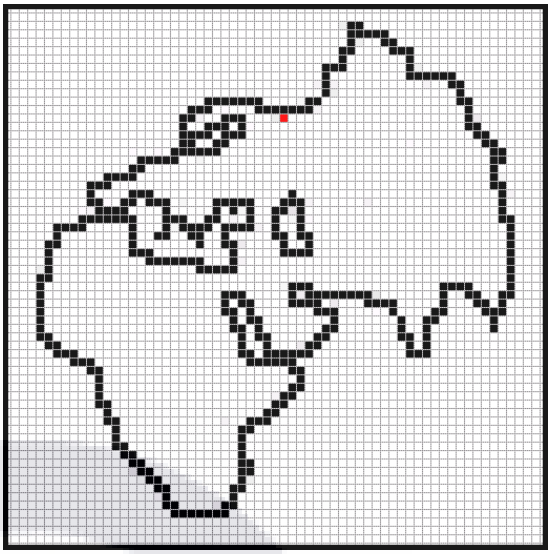


Figura 3.11 Frontera binaria, Euroasiaticoafricano, ejemplo rellenado, primer pixel.

La Figura 3.12 muestra el rellenado hacia la izquierda a partir del punto inicial; la Figura 3.13 muestra el rellenado hacia la derecha.

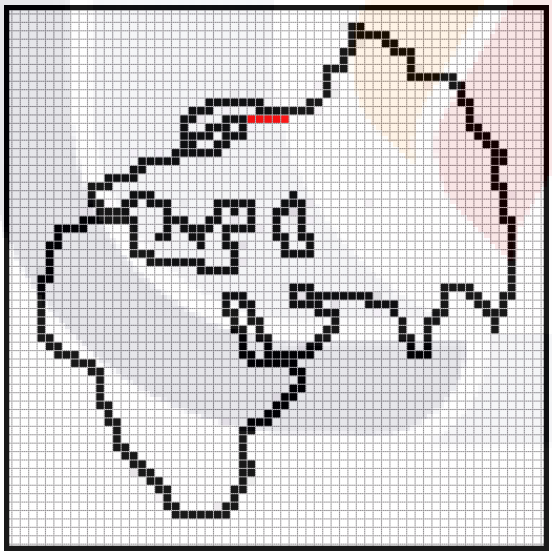


Figura 3.12 Frontera binaria, Euroasiaticoafricano, ejemplo rellenado, izquierda primera línea.

Figura 3.13 Frontera binaria, Euroasiaticoafricano, ejemplo rellenado, derecha primera línea.

En la Figura 3.14 se muestra el rellenado de la segunda línea en la cola (norte de la primera línea), en la Figura 3.15 se muestra el rellenado de la tercera línea en la cola (sur de la primera línea).

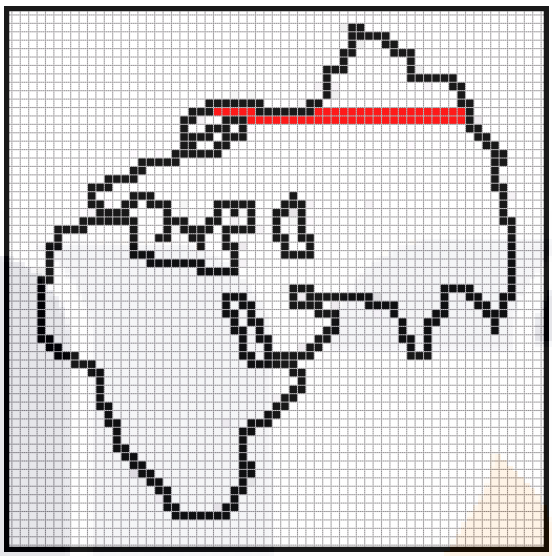


Figura 3.14 Frontera binaria, Euroasiaticoafricano, ejemplo rellenado, segunda línea.

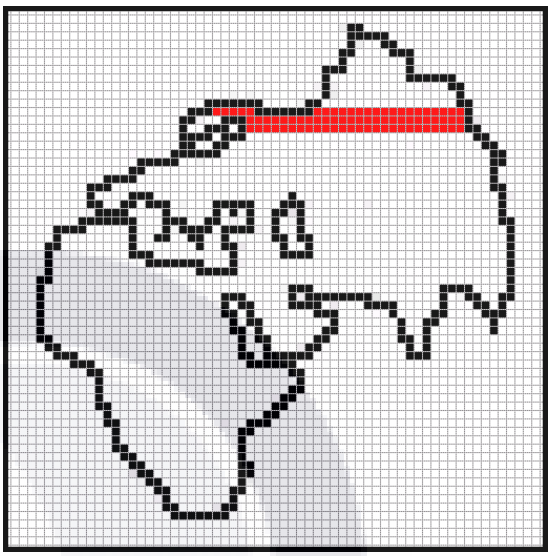


Figura 3.15 Frontera binaria, Euroasiaticoafricano, ejemplo rellenado, tercera línea.

La Figura 3.16 muestra el rellenado de la cuarta línea en la cola (norte de la segunda línea) y la Figura 3.17 el rellenado de la quinta línea en la cola (sur de la segunda línea).

Figura 3.16 Frontera binaria, Euroasiaticoafricano, ejemplo rellenado, cuarta línea.

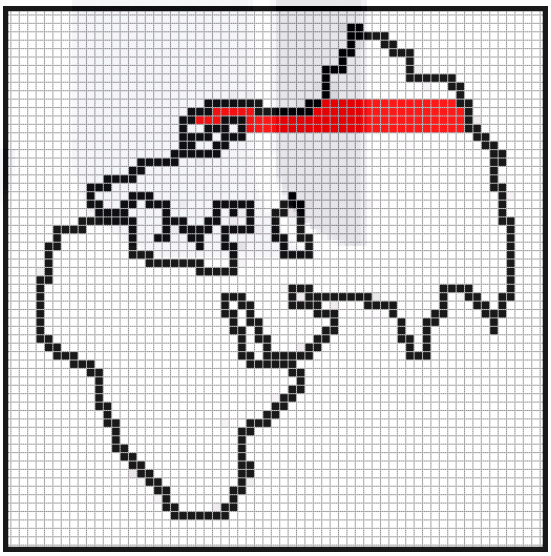


Figura 3.17 Frontera binaria, Euroasiaticoafricano, ejemplo rellenado, quinta línea.

3.4. Búsqueda de objetos y hoyos en imágenes binarias.

Una de las principales aportaciones de la tesis es que permite hacer la codificación de varios objetos binarios que pueden contener o no hoyos, para lograrlo es necesario poder identificar en una malla $\mathbb{G}$ los pixeles que constituyen a los diferentes objetos y hoyos para hacer su codificación.

Definición 3.7 Fondo de búsqueda.

Identificar los objetos en una imagen supone que todos los objetos se encuentran sobre un fondo de pixeles 0s. El fondo de búsqueda es una imagen auxiliar que tendrá las mismas dimensiones que la imagen original y tendrá, inicialmente, todos sus pixeles en estado apagado, una vez localizada la frontera de un objeto en la imagen original, esta y todo su relleno será marcado en el fondo de búsqueda, de manera que sirva como indicador para ignorar las coordenadas de dichos pixeles en la búsqueda del siguiente objeto en la imagen original. Por lo tanto, para decir que se ha encontrado la frontera de un objeto binario nuevo en una imagen digital, se hará escaneo en orden de línea mayor simultáneamente en la imagen original y en el fondo de búsqueda y si se encuentra un pixel que en la imagen original está encendido y en el fondo de búsqueda está apagado, este pixel será el pixel más hacia arriba y a la izquierda del siguiente objeto binario.

La Figura 3.18, Figura 3.19, Figura 3.20 y Figura 3.21 muestran cada una dos mapas de bits, de los cuales el de la izquierda representa a la imagen binaria original y el de la derecha el fondo de búsqueda.

La Figura 3.18 muestra la imagen binaria original y el fondo de búsqueda en su estado inicial antes de comenzar la búsqueda de formas en la imagen binaria.

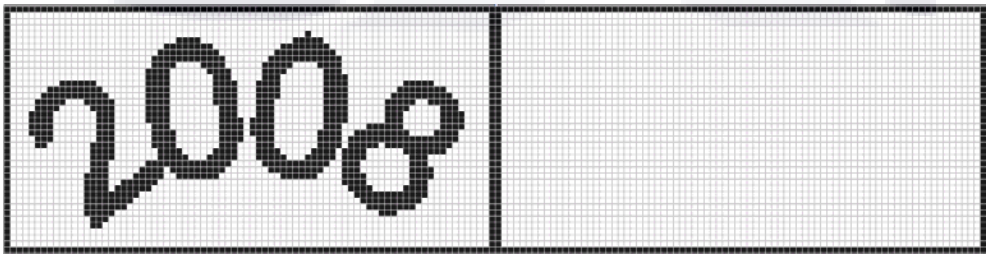


Figura 3.18 Imágenes binarias, 2008, ejemplo búsqueda de objetos binarios, estado inicial.

La Figura 3.19 muestra la frontera del objeto binario encontrado marcada con rojo y el estado del fondo de búsqueda después de haber ejecutado el algoritmo por primera vez.

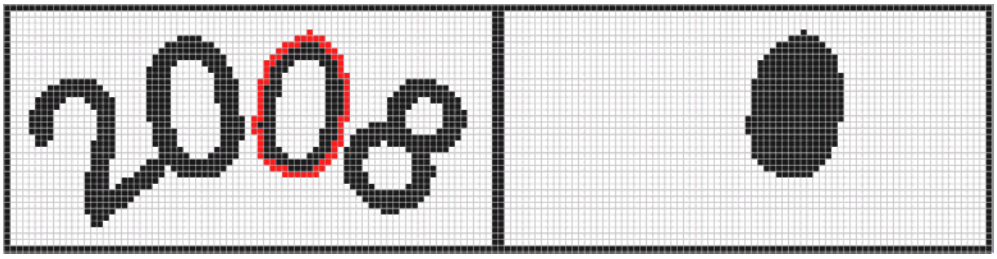


Figura 3.19 Imágenes binarias, 2008, ejemplo búsqueda de objetos binarios, primer objeto marcado.

La Figura 3.20 muestra la frontera del objeto binario encontrado marcada con rojo y el estado del fondo de búsqueda después de haber ejecutado el algoritmo por segunda vez.

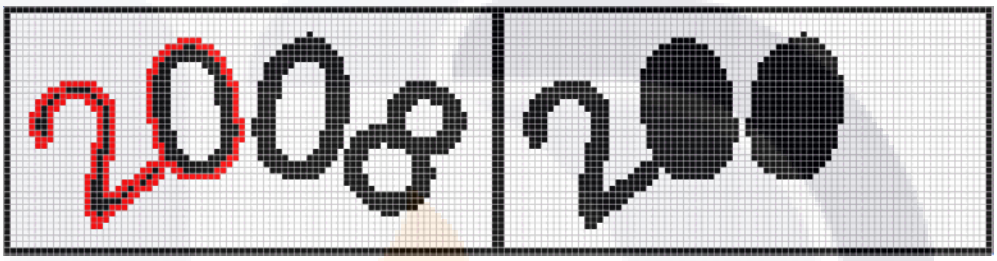


Figura 3.20 Imágenes binarias, 2008, ejemplo búsqueda de objetos binarios, segundo objeto encontrado.

La Figura 3.21 muestra la frontera del objeto binario encontrado marcada con rojo y el estado del fondo de búsqueda después de haber ejecutado el algoritmo por tercera vez.

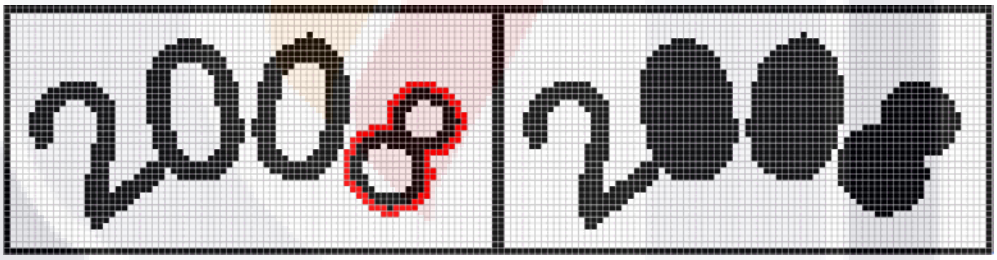


Figura 3.21 Imágenes binarias, 2008, ejemplo búsqueda de objetos binarios, tercer objeto encontrado.

Definición 3.8 Forma de búsqueda.

La búsqueda de hoyos se hace sobre un grupo conectado de pixeles en estado encendido, objeto binario. La forma de búsqueda consiste de una imagen con las mismas dimensiones que la imagen binaria original y con todos sus pixeles en 0s excepto aquellos delimitados por la frontera externa del objeto sobre el que se buscan hoyos. Por lo tanto, para decir que se ha encontrado un hoyo nuevo sobre el objeto binario actual se hará, simultáneamente, un escaneo en orden de línea mayor sobre la imagen original y la forma de búsqueda y si se encuentra un pixel en estado apagado sobre la imagen original y un pixel en estado encendido sobre la

forma de búsqueda, dicho pixel será el elemento más hacia arriba y a la izquierda de la frontera del siguiente hoyo binario en el objeto.

La Figura 3.22, Figura 3.23, Figura 3.24 muestran cada una dos mapas de bits, de los cuales el de la izquierda representa a la imagen binaria y el de la derecha la forma de búsqueda.

La Figura 3.22 muestra la imagen binaria original y la forma de búsqueda en su estado inicial antes de comenzar la búsqueda de hoyos del tercer objeto binario.

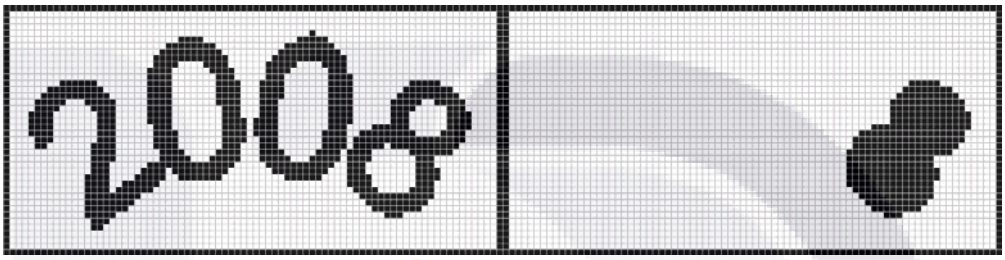


Figura 3.22 Imágenes binarias, 2008, ejemplo búsqueda de hoyos, estado inicial.

La Figura 3.23 muestra la frontera del hoyo encontrado marcada con rojo y el estado de la forma de búsqueda después de haber ejecutado el algoritmo por primera vez.

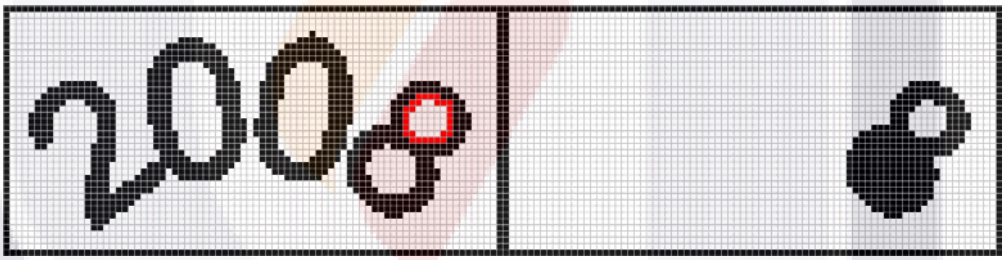


Figura 3.23 Imágenes binarias, 2008, ejemplo búsqueda de hoyos, primer hoyo encontrado.

La Figura 3.24 muestra la frontera del hoyo encontrado marcada con rojo y el estado de la forma de búsqueda después de haber ejecutado el algoritmo por segunda vez.

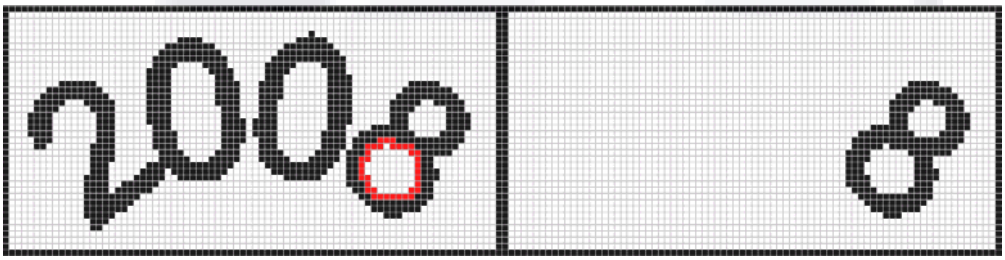


Figura 3.24 Imágenes binarias, 2008, ejemplo búsqueda de hoyos, segundo hoyo encontrado.

A continuación se aprecia una secuencia habitual para la codificación de una imagen binaria con 3 objetos con varios hoyos cada uno, La imagen original se aprecia en la Figura 3.25.

Figura 3.25 Imagen binaria, 3 aviones, ejemplo orden de búsqueda de objetos y hoyos.

Se comienza la codificación del avión de la derecha, ya que es el primer objeto encontrado al recorrer la imagen en orden de línea mayor. Se muestra marcado el contorno de dicho objeto en la Figura 3.26.

Figura 3.26 Imagen binaria, 3 aviones, ejemplo orden de búsqueda de objetos y hoyos, primer objeto.

Después, se codifican los hoyos del primer objeto, el 3er avión, haciéndole zoom, en la Figura 3.27 se muestra marcado el primer hoyo encontrado, en la Figura 3.28 el segundo hoyo y se prosigue hasta encontrar el último hoyo, que se aprecia en la Figura 3.29.

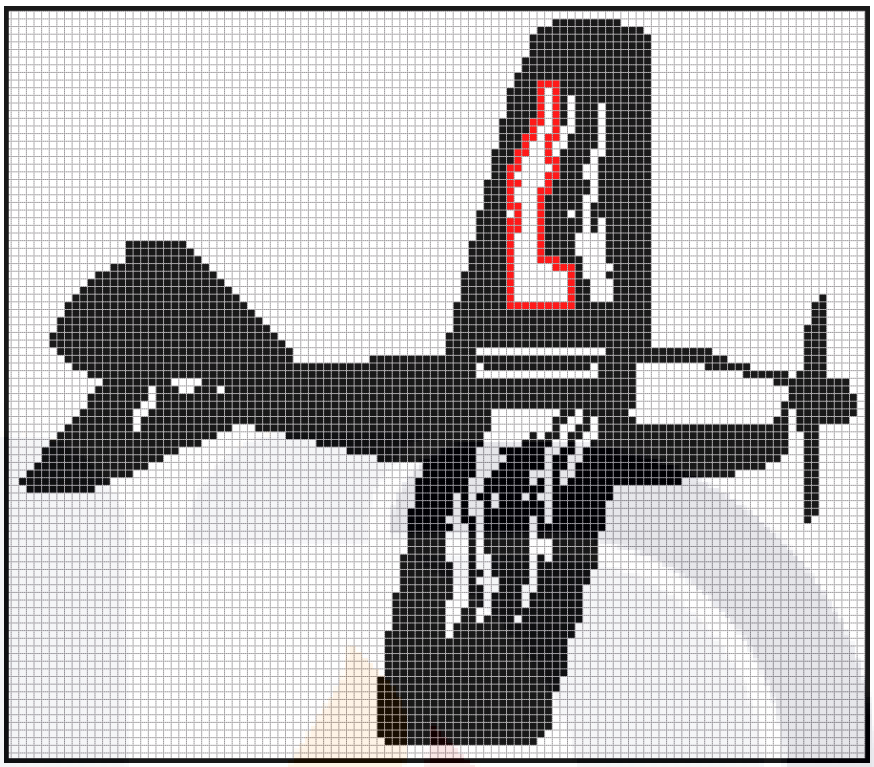


Figura 3.27 Imagen binaria, 3 aviones, ejemplo orden de búsqueda de objetos y hoyos, primer hoyo del primer objeto.

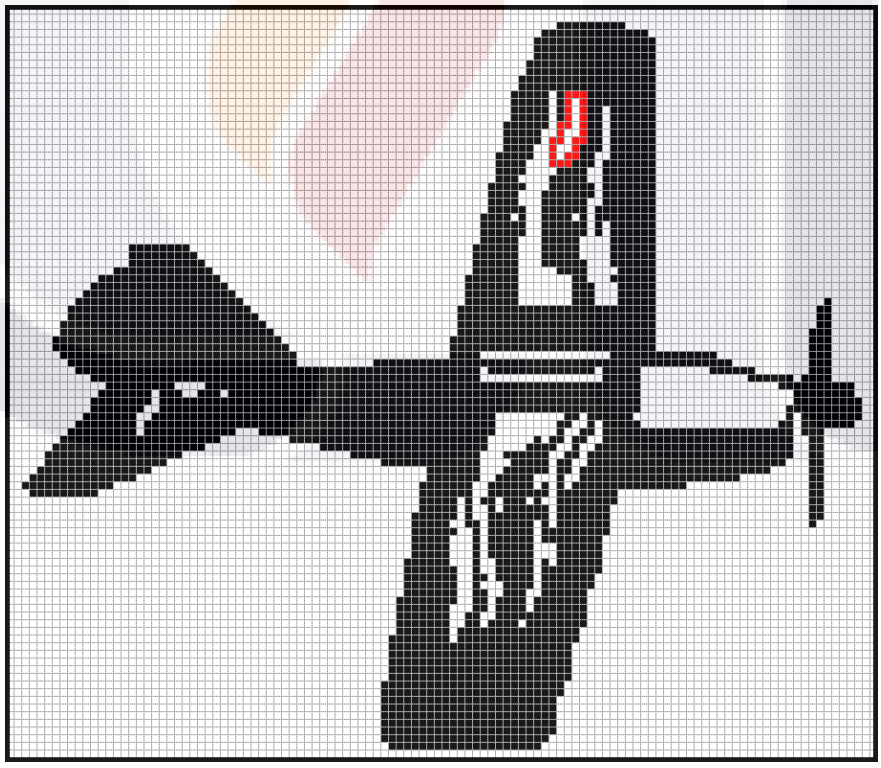


Figura 3.28 Imagen binaria, 3 aviones, ejemplo orden de búsqueda de objetos y hoyos, segundo hoyo del primer objeto.

Y se prosigue con el resto de los hoyos, hasta codificar el último hoyo de la imagen

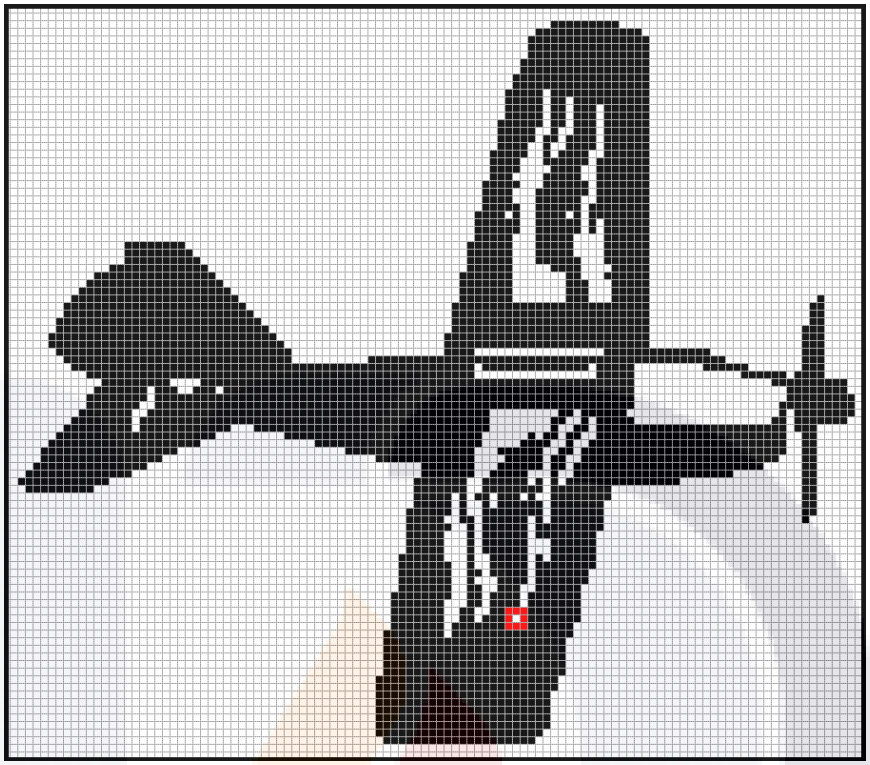


Figura 3.29 Imagen binaria, 3 aviones, ejemplo orden de búsqueda de objetos y hoyos, último hoyo del primer objeto.

Se procede buscando el siguiente objeto, marcado en la Figura 3.30 y codificando su frontera.

Figura 3.30 Imagen binaria, 3 aviones, ejemplo orden de búsqueda de objetos y hoyos, segundo objeto.

Se codifican los hoyos del segundo objeto, comenzando por el primer hoyo, Figura 3.31.

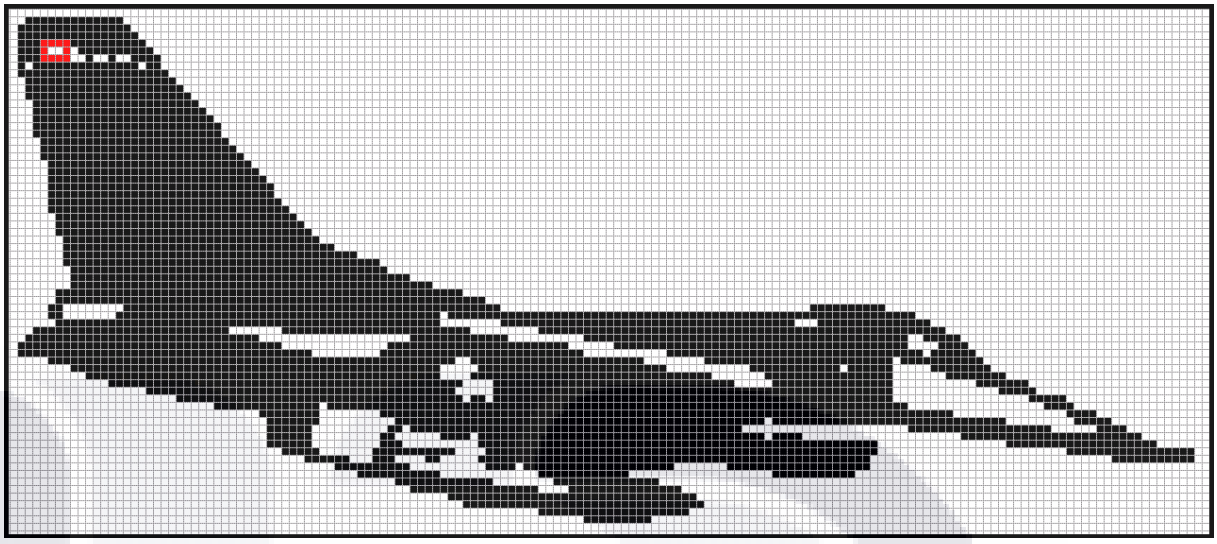


Figura 3.31 Imagen binaria, 3 aviones, ejemplo orden de búsqueda de objetos y hoyo, primer hoyo del segundo objeto.

Hasta codificar el último hoyo del segundo objeto, Figura 3.32.

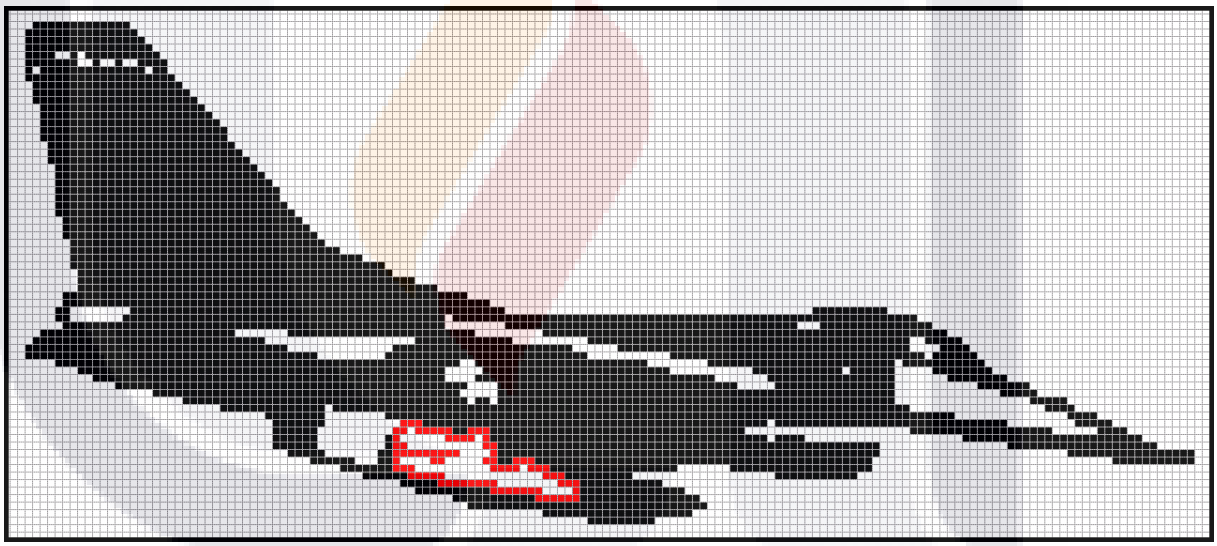


Figura 3.32 Imagen binaria, 3 aviones, ejemplo orden de búsqueda de objetos y hoyo, último hoyo del segundo objeto.

Al terminar la codificación de todos los hoyos del primer avión, es decir, del segundo objeto encontrado, se continúa la búsqueda de objetos, hasta localizar el siguiente y último objeto, que es el avión de en medio, con su frontera marcada en la Figura 3.33, se codifica igualmente el contorno exterior del objeto.

Figura 3.33 Imagen binaria, 3 aviones, ejemplo orden de búsqueda de objetos y hoyo, tercer objeto.

Y se procede, al igual que en los otros objetos, primer hoyo en la Figura 3.34.

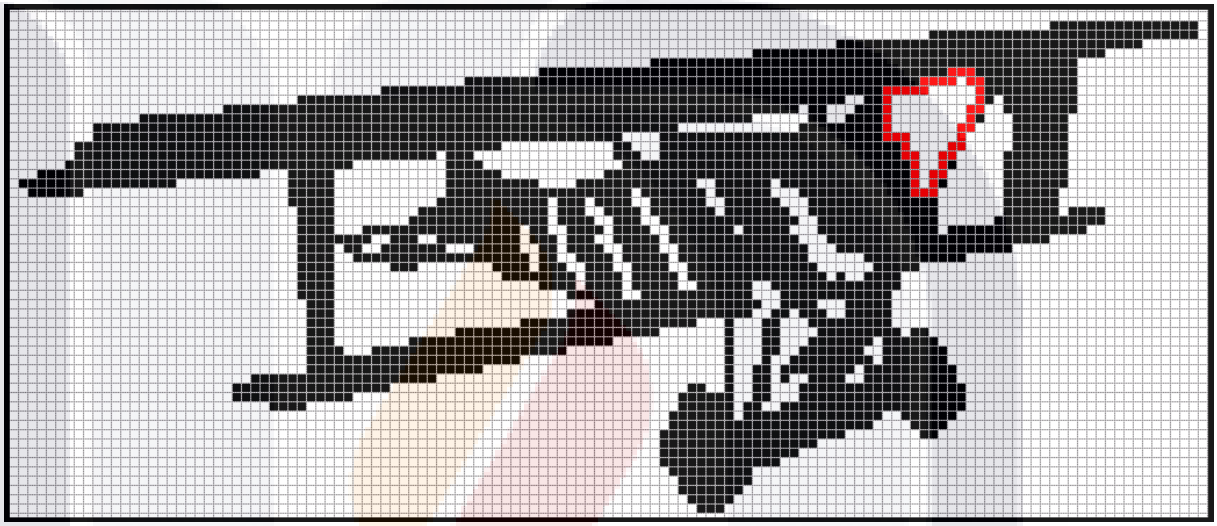


Figura 3.34 Imagen binaria, 3 aviones, ejemplo orden de búsqueda de objetos y hoyo, primer hoyo del tercer objeto.

Hasta codificar el último hoyo del tercer objeto, marcado en la Figura 3.35.

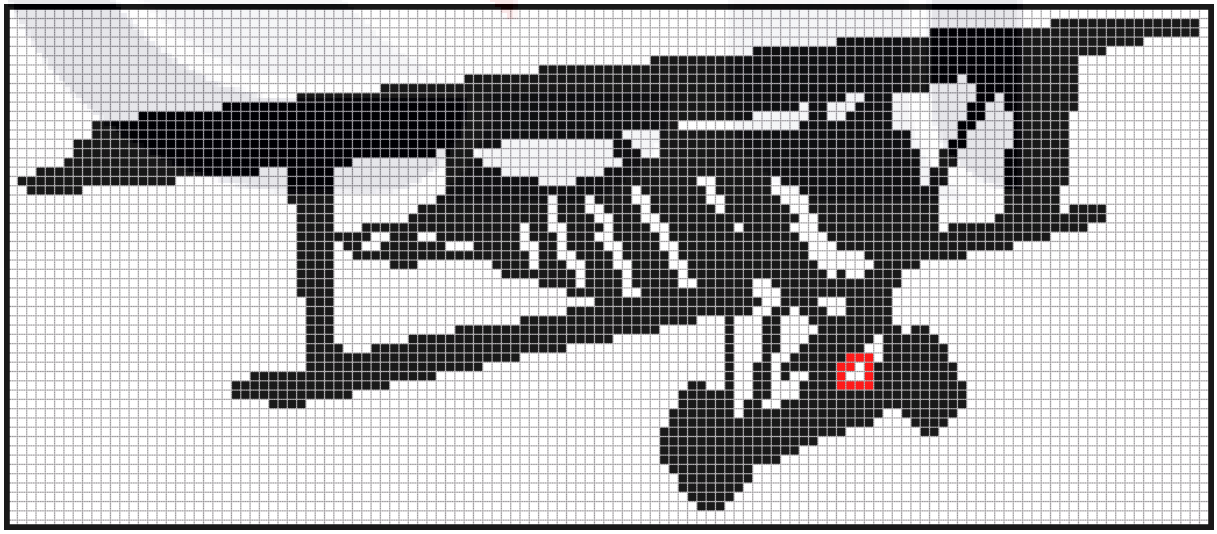


Figura 3.35 Imagen binaria, 3 aviones, ejemplo orden de búsqueda de objetos y hoyo, último hoyo del tercer objeto.

3.5. Decodificación 3OT.

En contraparte a la codificación, en donde los símbolos del alfabeto Ω se generan analizando si existen cambios de dirección simple, nulo o en diagonal en las fronteras; en la decodificación se irán graficando dichos cambios dependiendo de los vectores de dirección que se generan al analizar los símbolos leídos de la cadena. Los diferentes cambios de dirección se consiguen de esta manera:

- Cambio de dirección simple: Se genera cuando el vector de cambio actual es igual al vector pivote del símbolo anterior rotado 90° hacia la derecha. En la Figura 3.36 se aprecia un cambio de dirección simple, el vector pivote anterior (del pixel marcado con verde) apunto hacia la derecha; el vector de cambio actual (del pixel marcado con rojo) apunta hacia abajo

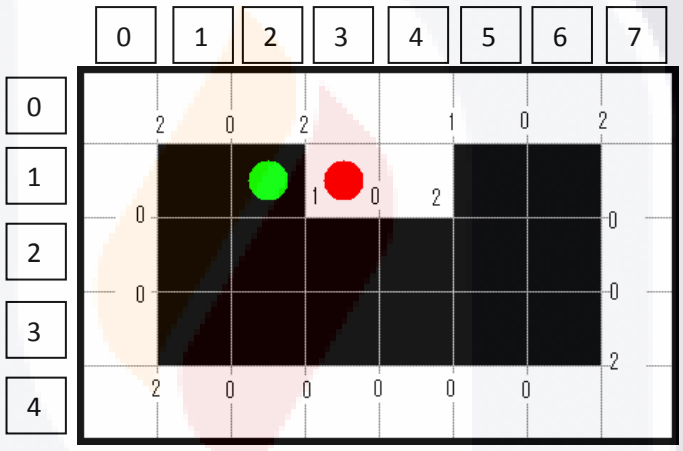


Figura 3.36 Imagen binaria, aleatoria, cambio de dirección simple en decodificación,

- Cambio de dirección nulo: Se genera cuando el vector de cambio actual es igual al vector pivote del símbolo anterior. En la Figura 3.37 se aprecia un cambio de dirección nulo, el vector pivote anterior (del pixel marcado con verde) apunto hacia la derecha; el vector de cambio actual (del pixel marcado con rojo) apunta hacia la derecha.

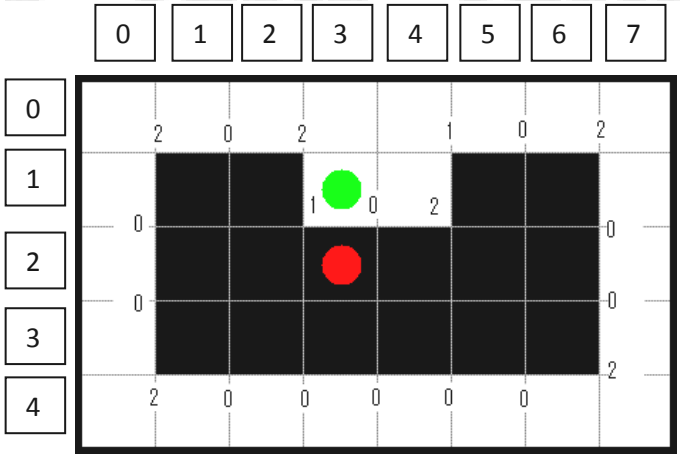


Figura 3.37 Imagen binaria, aleatoria, cambio de dirección nulo en decodificación.

- Cambio de dirección diagonal: Se genera cuando el vector de cambio actual es igual al vector pivote del símbolo anterior rotado 90° hacia la izquierda. En la Figura 3.38 se aprecia un cambio de dirección diagonal, el vector pivote anterior (del pixel marcado con verde) apunto hacia la abajo; el vector de cambio actual (del pixel marcado con rojo) apunta hacia la derecha.

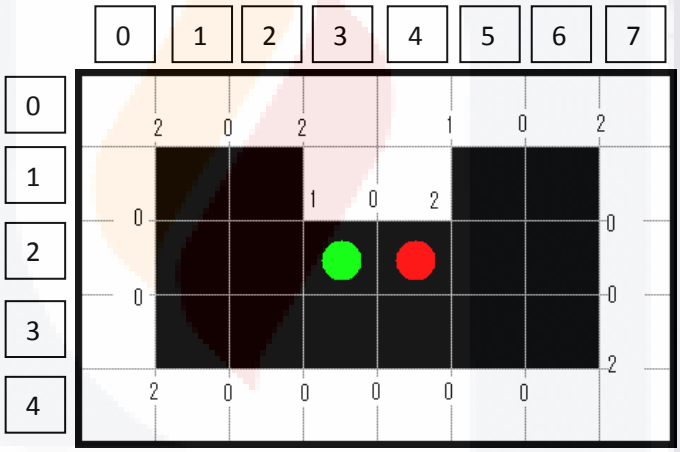


Figura 3.38 Imagen binaria, aleatoria, cambio de dirección diagonal en decodificación.

3.6. Algoritmo aritmético.

Básicamente, la compresión consiste en tomar una cadena de símbolos y transformarlos en códigos. Si la compresión es eficiente, los códigos resultantes ocuparán menor espacio que la cadena original (Abel, 2004 - 2009). Una vez obtenido el código 3OT que representa a la forma binaria se le aplica el algoritmo de compresión aritmético para permitir optimizar su almacenamiento.

El aritmético es un algoritmo estadístico de compresión, ya que utiliza las propiedades estadísticas de la fuente para mejorar la codificación. Se trata de aprovechar la redundancia de información de la fuente para conseguir esa compresión.

Está basado en las probabilidades de repetición de los mensajes de entrada. La idea principal es asignar a cada símbolo de la cadena original un intervalo [0 .. 1), cada intervalo es dividido en varios subintervalos, cuyos tamaños representan la probabilidad actual de aparición de los símbolos correspondientes del alfabeto definido para el código. El subintervalo del símbolo codificado se toma luego como intervalo del próximo símbolo a codificar. La salida es el intervalo del último símbolo.

El algoritmo aritmético es un método estadístico de compresión, ya que el número de bits usados para codificar cada símbolo varía de acuerdo a la probabilidad de su aparición en el mensaje original. Si un símbolo tiene una probabilidad baja de aparición en el mensaje utiliza muchos bits en memoria y los símbolos que aparecen más frecuentemente utilizan menos bits en memoria (Nelson, 1991).

Para llevar a cabo la codificación aritmética se ejecutan dos pasos: primero hay que obtener la probabilidad de aparición de cada símbolo del alfabeto usado en el mensaje, posteriormente habrá que asignar un rango de valores dentro de la distribución de probabilidad obtenida para cada símbolo leído del mensaje original.

El primer paso del algoritmo aritmético consiste en calcular la probabilidad de aparición de cada símbolo del alfabeto dentro de la cadena a comprimir.

Por ejemplo, se tiene la cadena M en el alfabeto $\Omega = \{0, 1, 2\}$, con una longitud de 30

M=002102100200002002102100200002

M es una cadena de símbolos que representa la forma binaria de la Figura 3.39 en código 3OT:

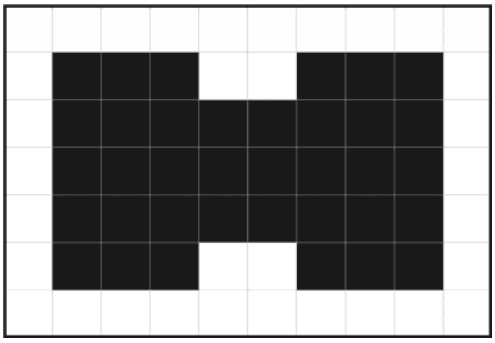


Figura 3.39 Imagen binaria, aleatoria, ejemplo de algoritmo aritmético.

En la Tabla 3.2 se muestra la distribución de probabilidad símbolos del alfabeto 3OT:

Símbolo	*Apariciones*	*Probabilidad*	*Prob Acumulada*	*Rango_inferior*	*Rango_superior*
0	*18*	*0.6*	*0.6*	*0*	*0.6*
1	*4*	*0.1333*	*0.73*	*0.6*	*0.73*
2	*8*	*0.2666*	*1*	*0.73*	*1*

Tabla 3.2 Distribución probabilidad, ejemplo de algoritmo aritmético.

De la tabla 2.4 se obtiene la Tabla 3.3, con los rangos de probabilidad para cada símbolo

Símbolo	*Rango_inferior*	*Rango_superior*
0	*0*	*0.6*
1	*0.6*	*0.73*
2	*0.73*	*1*

Tabla 3.3 Rangos de probabilidad, ejemplo de algoritmo aritmético.

Una vez que se tiene la distribución de probabilidad de los símbolos del alfabeto en la cadena, se procede a asignar un rango de valores a cada símbolo, dependiendo de su posición en la cadena y de la probabilidad que tiene de aparición. El Pseudocódigo 3.1 se usa para obtener los rangos de salida.

Asignación de rangos de valores en algoritmo aritmético

Permite generar un número flotante que representa una cadena de símbolos utilizando los rangos de probabilidad de aparición de los mismos.

Variables	**Tipo**	**Descripción**
Rango	Flotante	Lleva el registro de cuál debe ser la proporción para el siguiente rango de valores.
L_inferior	Flotante	Especifica el límite inferior generado.
L_superior	Flotante	Especifica el límite superior generado.
Rango_inferior_del_ simbolo	Flotante	Es el límite inferior del rango para el símbolo correspondiente en la cadena, se obtiene de la tabla 2.5.
Rango_superior_del_ símbolo	Flotante	Es el límite superior del rango para el símbolo correspondiente en la cadena, se obtiene de la tabla 2.5.

Pseudocódigo

1. L_inferior = 0;
2. L_superior = 1;
3. Hacer lo siguiente para todos los símbolos del mensaje:

3.1. Rango = L_superior – L_inferior;
3.2. L_inferior = L_inferor + (Rango * Rango_inferior_del_símbolo);
3.3. L_superior = L_inferior + (Rango * Rango_superior_del_símbolo);

Pseudocódigo 3.1 Asignación de rangos de valores en algoritmo aritmético.

Y para la codificación de los primeros símbolos del mensaje tenemos los resultados mostrados en la Tabla 3.4.

Símbolo del mensaje	Rango	L_inferior	L_superior
		0	1
0	1	0	0.6
0	0.6	0	0.36
2	0.36	0.2628	0.36
1	0.0972	0.32112	0.333756
0	0.012636	0.32112	0.3287016
2	0.0075816	0.326654568	0.3287016

Tabla 3.4 Asignación de rangos, ejemplo de algoritmo aritmético.

Entonces, hasta este momento se tiene que el valor que representa al mensaje 002102 es cualquier número dentro del rango

[0.326654568, 0.3287016)

Se puede tomar algún valor que requiera menos dígitos para su representación

002102 = 0.327

Es importante tener en cuenta que el código obtenido no representa únicamente al mensaje codificado, sino que representa a todos los mensajes que comienzan de la misma manera que dicho mensaje.

Por ejemplo:

El código 0.326654568 representa al mensaje

002102

Así como también representa a los siguientes mensajes:

002102000

0021021

002102112

00210210

Por lo tanto, es necesario almacenar el tamaño del mensaje original para conocer un punto de paro al hacer la decodificación.

El código aritmético es un algoritmo sin pérdida de información, siempre y cuando se cuente con la distribución de probabilidad original al momento de decodificar (Said, 2003).

Para lograr la decodificación se procede de manera inversa a como se codificó, para lograrlo es necesario contar con tres datos de información, que deben ser guardados en el archivo en que se almacene el código:

1. La distribución de probabilidad en la cadena original: es necesario conocerla porque de otra manera no sería posible asignar los símbolos decodificados.
2. El valor flotante que representa a la cadena original: se partirá de este valor para comenzar la decodificación, usando la distribución de probabilidad del mensaje se podrá asignar los símbolos.
3. Longitud de la cadena original o la cantidad de símbolos que lo componen: este dato es necesario como condición de paro para la decodificación.

La manera de decodificar es primero ver a que rango pertenece el valor flotante y la salida será el símbolo de dicho rango, después hay que extraer el rango de este símbolo del valor flotante. El Pseudocódigo 3.2 sirve para obtener la cadena original a partir del valor flotante.

Decodificación de código aritmético

A partir de un número flotante y utilizando los rangos de probabilidad de aparición de los símbolos del alfabeto genera la cadena original.

Variables	Tipo	Descripción
Longitud	Entero	Tiene almacenada la longitud de la cadena original.
Rango	Flotante	Lleva el registro de cuál debe ser la proporción para el siguiente rango de valores.
Numero_flotante	Flotante	Es el número que está siendo decodificado
Rango_inferior_del_simbolo	Flotante	Es el límite inferior del rango para el símbolo correspondiente en la cadena, se obtiene de la tabla 2.5.
Rango_superior_del_símbolo	Flotante	Es el límite superior del rango para el símbolo correspondiente en la cadena, se obtiene de la tabla 2.5.

Pseudocódigo

1. Hacer lo siguiente Longitud veces
    1.1. Rango = Rango_superior_del_símbolo – Rango_inferior_del_símbolo;
    1.2. Numero_flotante = Numero_flotante – Rango_inferior_del_símbolo;
    1.3. Numero_flotante = Numero_flotante / Rango
    1.4. El Símbolo de esta iteración será el que su rango contenga a Numero_flotante.

Pseudocódigo 3.2 Decodificación de código aritmético.

En la Tabla 3.5 se muestran los resultados de las primeras cinco iteraciones del Pseudocódigo 3.2

Numero_flotante	Símbolo	Rango
0.326654568	0	0.6
0.54442428	0	0.6
0.9073738	2	0.2666
0.665318	1	0.1333
0.49	0	0.6
0.81668	2	0.2666

Tabla 3.5 Decodificación aritmética, ejemplo de algoritmo aritmético.

Capítulo 4

DETECCIÓN DE PUNTOS DOMINATES CON 3OT

En esta sección se describe el método para la detección de puntos dominantes basado en 3OT, también se explica en qué consiste la extensión implementada para dicho método.

4.1. Subcadenas.

Para la detección de puntos dominantes usando código 3OT se hace un reconocimiento de patrones de secuencias de símbolos que representan cambios de dirección específicos a través de la frontera de los objetos binarios codificados.

4.1.1. Generación de subcadenas a analizar.

La búsqueda de los patrones se hace sobre subcadenas m_i, con $0 \leq i < \mathcal{L}$, de alguna cadena M de longitud $\mathcal{L}$ definida sobre el alfabeto Ω, la longitud de las subcadenas m_i es de $\ell + 1$, donde ℓ es un parámetro de ajuste del algoritmo y es definido por el usuario.

La cadena M, que contiene el código 3OT que representa al objeto al que se le detectarán los puntos dominantes, debe ser procesada para obtener una matriz con el número de filas igual a la longitud de las subcadenas (ℓ+1), y con el número de columnas igual a la longitud de la cadena M ($\mathcal{L}$). Cada columna de esta matriz es una subcadena (m_i) que será analizada para ver si se ajusta a algún patrón. La primer columna o subcadena m_0 se generará tomando ℓ+1 (por ejemplo 11) caracteres contiguos de M comenzando por el primer elemento de M, la segunda columna m_1 se generará tomando como primer elemento el segundo carácter de M y será de la misma longitud que m_0, se prosigue de esta manera hasta la generación de la columna $m_{\mathcal{L}-(\ell+1)}$, a partir de la columna $m_{\mathcal{L}-\ell}$ y hasta la última columna $m_{\mathcal{L}-1}$ se generarán tomando los primeros elementos de la columna a partir del i actual y hasta el último elemento de M y los elementos faltantes se tomarán a partir del primer elemento de M hasta completar el tamaño para la subcadena m_i (ℓ+1). Como ejemplo se utilizará el objeto binario de la Figura 4.1, que tiene la cadena 3OT con $\mathcal{L}$=1002 de la Tabla 4.1. Utilizando ℓ=10 se genera la matriz de la Tabla 4.2.

Figura 4.1 Imagen binaria, Perrito.

000021001101101101100000000011110110000011000000000011001101100000000110110110010101000000000002111001111101111101111102101000011011000011011111101010101110101111110111111111111111110111110111100101011011000011001100000110111101111111111111100101110111011101101010010111101000111101100111101101101101101101111010010100000000000000000000021110011101000101111110111101111011000002002111111111111111101100110011000211101111010000211011111111111111111101101011111110011111100000000000210101111011110110002111111101120110000200000011011011011000000101111110011000000011000021000000110110110101111100000211100011100000110000012011001101101111110101011110211121212110011011101111111111011110101110000000000021011001100110111011101101100002101111011011101110100020100011020011000000000000011210111101010101011011110000000002100001100011000000011001102210110110000011011000000021001101100111111010110110110110001100000000002101101101010111111000000202011101010110002111110100000000002010110011011011111011111111

Tabla 4.1 Código 3OT, Perrito.

m_0	m_1	m_2	$m_3 \ldots m_{\ell-(\ell+1)-1}$	$m_{\ell-(\ell+1)}$	$m_{\ell\ell}$	$m_{\ell\ell+1}$	$m_{\ell\ell+2} \ldots m_{\ell-2}$	$m_{\ell-2}$	$m_{\ell-1}$
0	0	0	…	1	1	0	…	1	1
0	0	0	…	1	0	1	…	1	0
0	0	2	…	0	1	1	…	0	0
0	2	1	…	1	1	1	…	0	0
2	1	0	…	1	1	1	…	0	0
1	0	0	…	1	1	1	…	0	2
0	0	1	…	1	1	1	…	2	1
0	1	1	…	1	1	1	…	1	0
1	1	0	…	1	1	1	…	0	0
1	0	1	…	1	1	0	…	0	1
0	1	1	…	1	0	0	…	1	1

Tabla 4.2 Subcadenas de longitud 11, Perrito.

4.1.2. Partes de las subcadenas.

Definición 4.1 Cambio de dirección notable

Las secuencias de caracteres del alfabeto Ω representan segmentos de líneas, que pueden ser rectas o curvas. Un cambio de dirección notable es cuando se presentan dos segmentos de curva de alguna longitud determinada con poca curvatura, contiguos y que tienen diferentes pendientes entre sí y están unidos por un punto (esquina) donde se hace notable el cambio de pendiente entre los dos segmentos.

Para la detección de puntos dominantes es necesario localizar aquellas subcadenas que tengan secuencias de caracteres que representen un cambio de dirección notable sobre la curva de la frontera del objeto analizado, para lograrlo se ha optado por dividir a las subcadenas en tres partes con la intención de diferenciar los dos segmentos de baja curvatura con diferente pendiente y el punto, esquina, que las une.

Al definir los patrones de búsqueda de subcadenas es necesario especificar las tres partes claramente, las cuales se separan escribiendo cada una entre paréntesis. También habrá que detallar las características que deben reunir, estas características son: la longitud de cada una de las partes que no debe ser mayor que el tamaño total de la subcadena (ℓ+1) para cada parte y la sumatoria de las tres debe ser igual a ℓ+1; otra característica a definir es que caracteres se presentarán en cada parte de la subcadena, estos se escriben separándolos con el símbolo de adición (+) y significa que puede aparecer cualquiera de estos símbolos si y sólo si son especificados dentro de los paréntesis; también, se puede utilizar el parámetro q para indicar que algún símbolo debería aparecer cierta cantidad de veces como mínimo dentro de la parte de subcadena.

Definición 4.2 Pivotes de subcadena

Como las cadenas del alfabeto Ω representan el contorno de un objeto en una imagen binaria, entonces cada uno de los símbolos de esta cadena tiene asociado un punto del contorno del objeto. El objetivo de analizar si las subcadenas m_i cumplen con algún patrón es para detectar los puntos dominantes del objeto, si una subcadena cumple con un patrón es necesario tener bien definido cuál o cuáles símbolos de ella serán considerados puntos dominantes, al definir las tres partes de subcadena habrá que dejar por lo menos una parte con tamaño igual a uno, a estas partes de subcadena con tamaño uno se les denomina pivotes de subcadena.

4.1.3. Agrupamiento de subcadenas.

Al encontrar las subcadenas que cumplen con algún patrón es posible encontrar algunas en que sus pivotes están separados por alguna cantidad de caracteres que pueden considerarse como muy cercanos o incluso pueden estar adyacentes. Esta situación es indeseable, porque si en los pixeles marcados como puntos dominantes sus coordenadas están muy cercanas se está agregando información redundante al conjunto de puntos dominantes.

Definición 4.3 Agrupamiento de subcadenas.

El agrupamiento de subcadenas consiste en para cada subcadena que cumpla con algún patrón S_i comparar si la posición de su caracter pivote se encuentra a una cantidad menor o igual a ν de caracteres con respecto a la posición del pivote de la siguiente subcadena que cumple con el patrón S_i, si esto es verdad, las subcadena se irán introduciendo a una lista hasta encontrar alguna subcadena que su pivote esté a una distancia mayor a ν. El valor de ν es un parámetro de ajuste para el algoritmo de detección de puntos dominantes.

4.1.4. Operaciones sobre grupos de subcadenas.

Una vez que se han agrupado las subcadenas, se puede aplicar a los diferentes grupos formados dos operadores para evitar tener subcadenas que marcan puntos dominantes muy cercanos entre sí. Si se decide aplicar estos operadores, a las subcadenas agrupadas se les conocerá como subcadenas candidatas y el resultado de los operadores generará las esquinas.

El primer operador es:

Definición 4.4 Toma de inicio y fin de grupos.

Cuando en un grupo de subcadenas existen más de dos elementos, puede suceder que el pivote de la primer subcadena se encuentre a una distancia de caracteres mayor a ν con respecto al pivote de la última subcadena, si este es el caso puede optarse por tomar como esquinas sólo la primera y la última subcadena, dejando como candidatas a las subcadenas intermedias.

El segundo operador para los grupos de subcadenas es:

Definición 4.5 Promediado de grupos.

Si el pivote de la primer subcadena de un grupo está a una distancia menor o igual a ν que el pivote de la última subcadena puede optarse por tomar como esquina sólo al pixel especificado por el caracter que tenga como posición el promedio de las posiciones de los pivotes de todas las subcadenas del grupo. De esta manera sólo queda una esquina definida por el grupo.

4.2. Patrones.

Se han definido tres patrones que serán usados para detectar subcadenas con puntos esquinas, estos patrones representan cambios de dirección en la frontera del objeto binario. El software generador de código 3OT permite al usuario definir fácilmente nuevos patrones de búsqueda, basta con expresarlos en un formato similar a expresiones regulares. Los siguientes patrones son el resultado de la experimentación con una cantidad de patrones y son los que arrojaron mejores resultados.

4.2.1. Patrón S_1.

Definición 4.6 Patrón S_1.

El primer patrón consiste en cambios de dirección a través de la frontera de aproximadamente 45°, en la Fórmula 4.1 se muestra la definición del patrón S_1 para la detección de puntos dominantes. Representa un segmento de línea con poca curvatura horizontal o vertical (mayoría de 0s) seguida por un cambio de dirección (1 o 2) pivote y después un segmento de línea con poca curvatura diagonal (mayoría de 1s); y su reflejo. En la Figura 4.2 se muestra el cambio de dirección representado por el patrón S_1.

$$S_1 = (0^q+1)^{1/2}(1)(0+1^q)^{1/2} + (0+1^q)^{1/2}(1)(0^q+1)^{1/2} + (0^q+1)^{1/2}(2)(0+1^q)^{1/2} + (0+1^q)^{1/2}(2)(0^q+1)^{1/2}$$

Fórmula 4.1 Patrón S_1.

Figura 4.2 Cambio de dirección en Patrón S_1.

En la Figura 4.3 se muestran las esquinas encontradas con el patrón S_1 en una parte de la imagen binaria de la Figura 4.1, estas esquinas se encontraron usando como parámetros $\ell = 10$, $\nu = 9$ y $q = 3$.

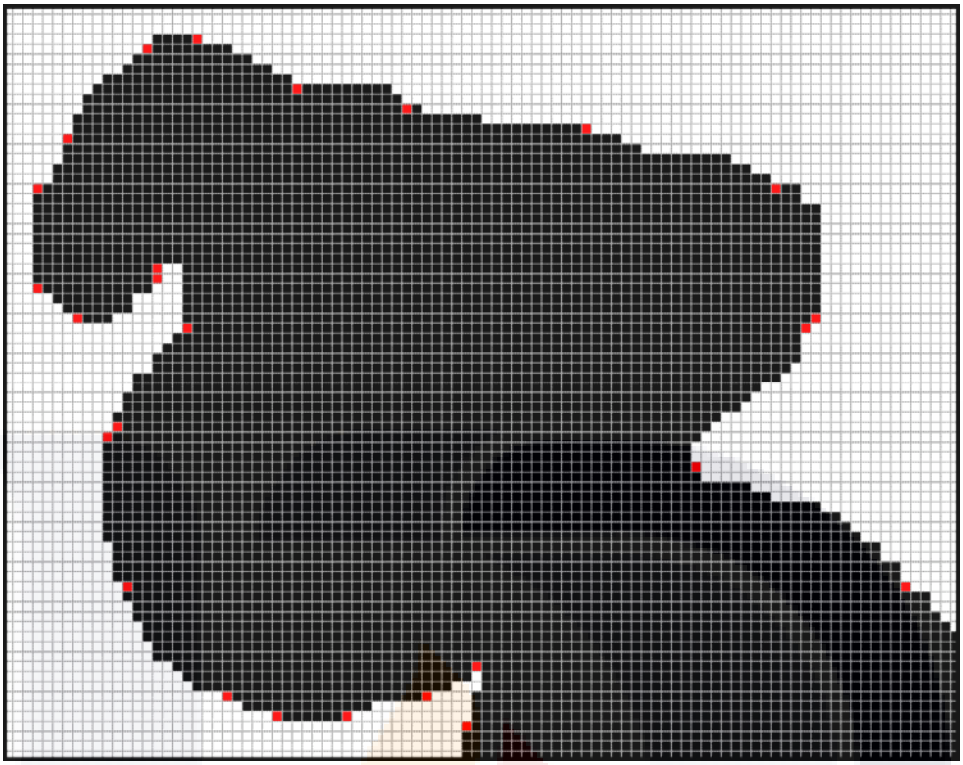


Figura 4.3 Imagen binaria con subcadenas S_1 marcadas, Perrito.

4.2.2. Patrón S_2.

Definición 4.7 Patrón S_2.

El segundo patrón es aproximadamente "escalones" o cambios ligeros sobre una línea con poca curvatura, en la Fórmula 4.2 se muestra la definición del patrón S_2 para la detección de puntos dominantes. Representa un segmento de línea con poca curvatura horizontal o vertical (mayoría de 0s) seguida de un pivote con un cambio de dirección (1) y después otra línea con poca curvatura horizontal o vertical (mayoría de 0s). En la Figura 4.4 se muestra un cambio de dirección representado por el patrón S_2.

$$S_2 = (0^q+1)^{l/2}(1)(0^q+1)^{l/2}$$

Fórmula 4.2 Patrón S_2.

Figura 4.4 Cambio de dirección en Patrón S_2.

En la Figura 4.5 se muestran las esquinas encontradas con el patrón S_2 en una parte de la imagen binaria de la Figura 4.1, estas esquinas se encontraron usando como parámetros $\ell = 10$, $\nu = 9$ y $q = 3$.

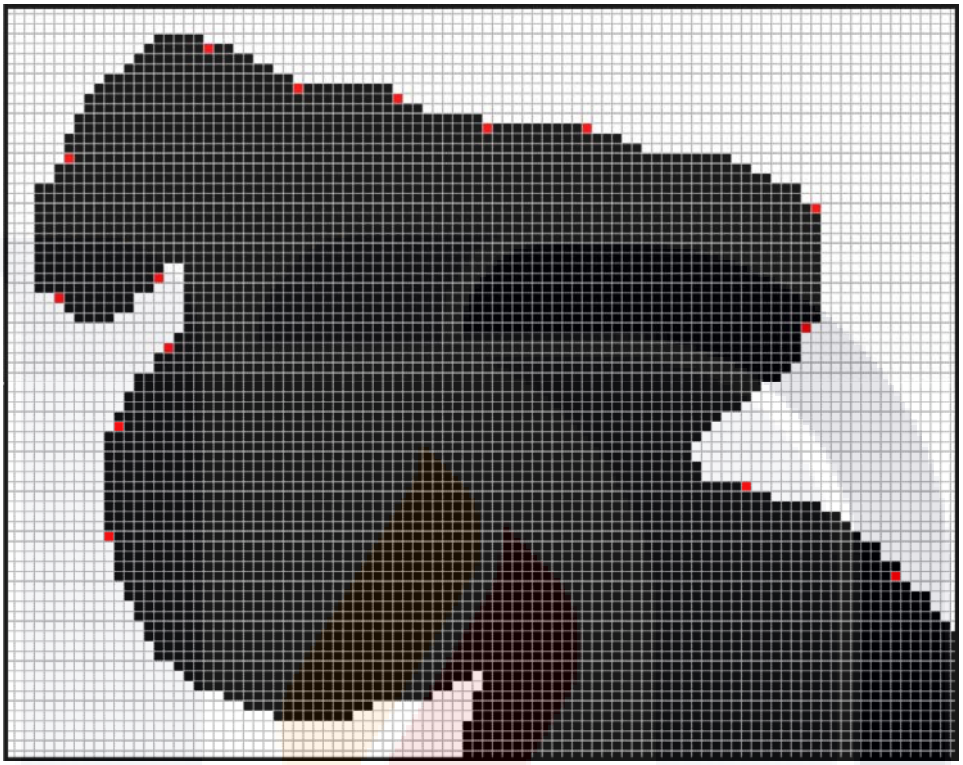


Figura 4.5 Imagen binaria con subcadenas S_2 marcadas, Perrito.

4.2.3. Patrón S_3.

Definición 4.8 Patrón S_3.

El tercer patrón consiste de secciones simétricas en las que hay retornos de dirección, en la Fórmula 4.3 se muestra la definición del patrón S_3 para la detección de puntos dominantes. Significa un cambio de dirección (1 ó 2) pivote, seguido de un segmento de línea recta horizontal o vertical (sólo 0s) y por último un pivote con cambio de dirección en sentido contrario (2) al primer pivote del patrón. En la Figura 4.6 se muestra un cambio de dirección representado por el patrón S_3.

$$S_3 = (1+2)(0^q)^{l-1}(2)$$

Fórmula 4.3 Patrón S_3.

Figura 4.6 Cambio de dirección en Patrón S_3.

En la Figura 4.7 se muestran las esquinas encontradas con el patrón S_3 en una parte de la imagen binaria de la Figura 4.1, estas esquinas se encontraron usando como parámetros $\boldsymbol{\ell} = 15$, $\nu = 9$ y $\boldsymbol{q} = 3$.

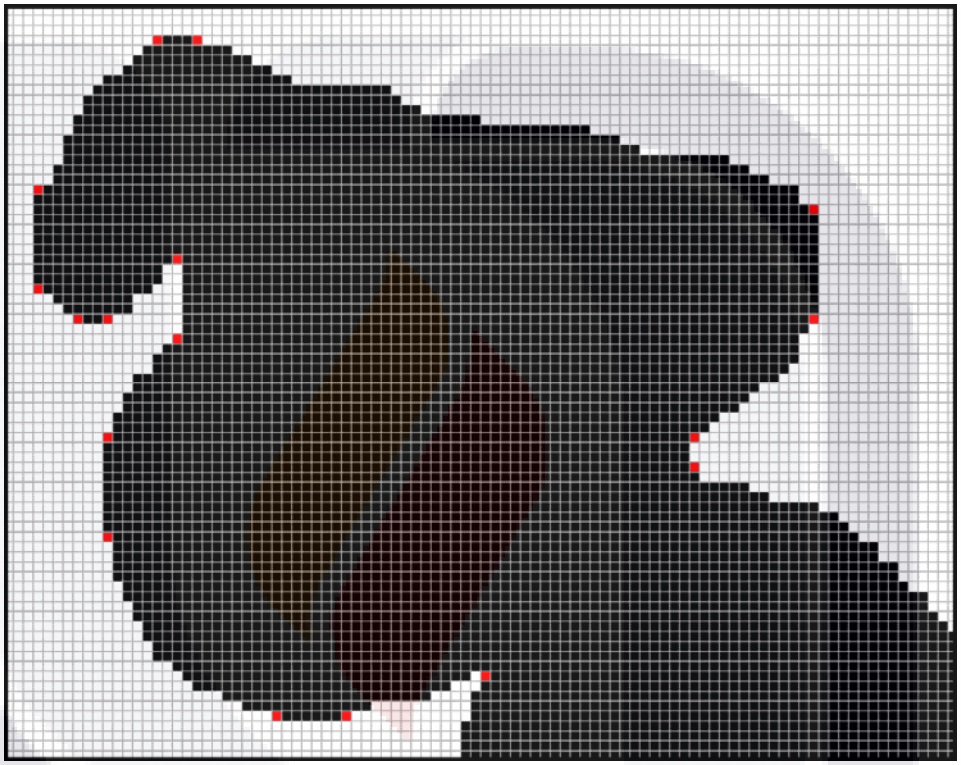


Figura 4.7 Imagen binaria con subcadenas S_3 marcadas, Perrito.

En la Figura 4.8 se muestra una parte de la frontera (negro) del objeto binario de la Figura 4.1 con los puntos dominantes en la Figura 4.3, la Figura 4.5 y la Figura 4.7 marcados (rojo) y unidos mediante segmentos de recta (azul).

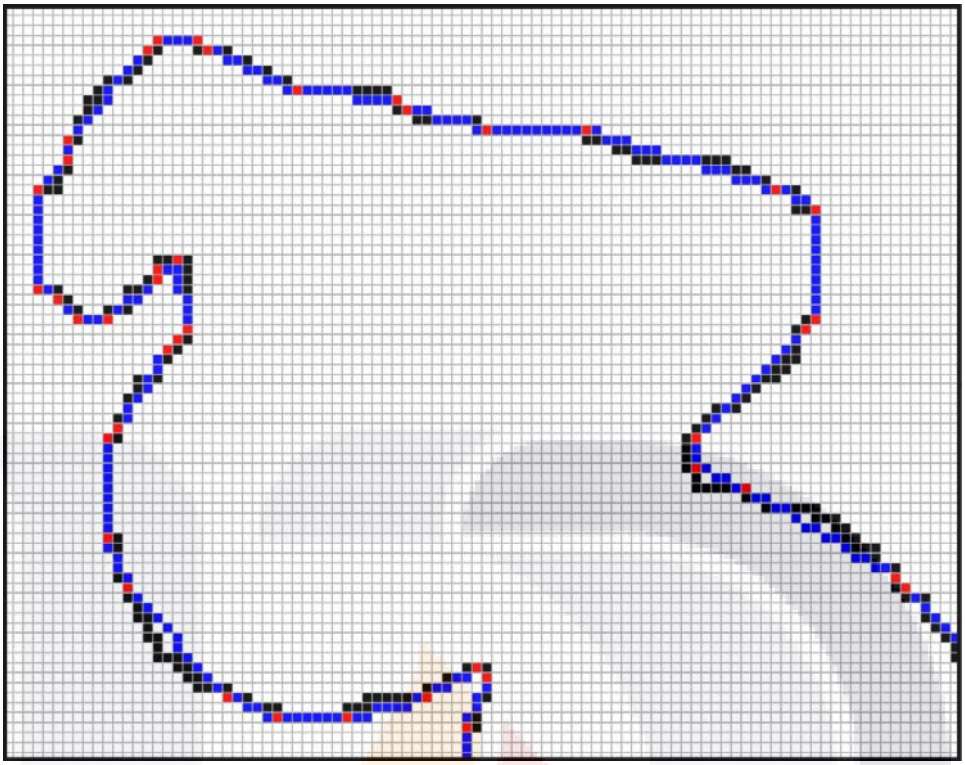


Figura 4.8 Frontera de imagen binaria con puntos dominantes y su polígono aproximado, Perrito.

4.3. Patrones como expresiones regulares.

Los patrones S_1, S_2 y S_3 son expresiones regulares porque son aceptados por un autómata finito no determinista (AFN). Construyendo los autómatas para los patrones tenemos que el alfabeto es $\sum = \Omega = \{0, 1, 2\}$.

El conjunto de estados para aceptar S_3 es $Q_3 = \{q_0, q_1, q_2\}$, la función de transición es δ_3: $Q_3 \times \sum \rightarrow Q_3$. Mientras que el conjunto de estados para aceptar S_1 y S_2 es $Q_{1,2} = \{q_0, q_1\}$ y la función de transición es $\delta_{1,2}$: $Q_{1,2} \times \sum \rightarrow Q_{1,2}$.

Así que, S_3 puede ser interpretado como un lenguaje aceptado por el autómata finito no determinista (AFN) de la Figura 4.9. Mientras que los patrones S_1 y S_2 son aceptados por el AFN de la Figura 4.10. Finalmente, la Figura 4.11 muestra el AFN que acepta los tres patrones.

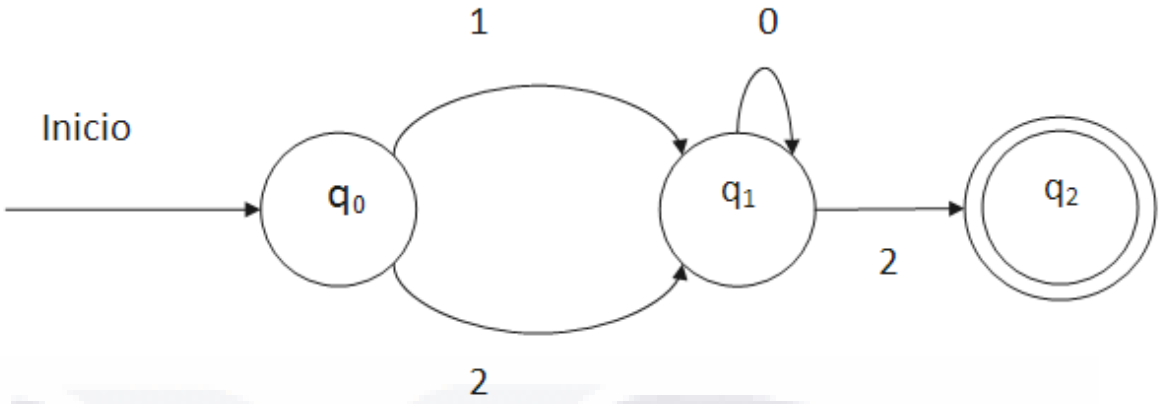


Figura 4.9 Autómata Finito no Determinista, Patrón S_3.

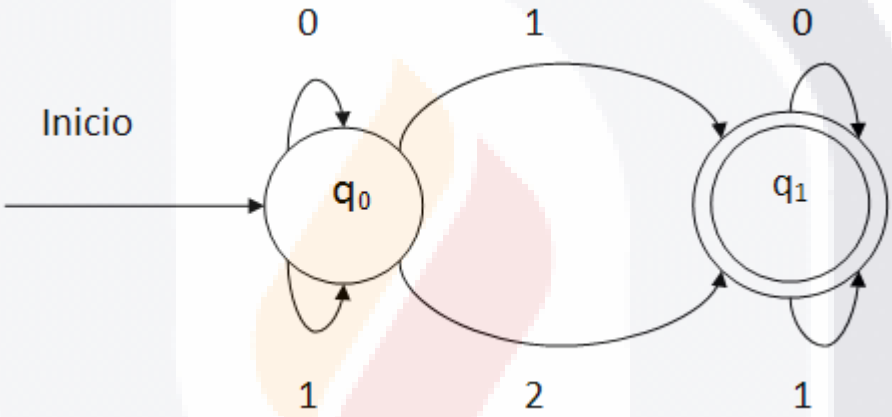


Figura 4.10 Autómata Finito no Determinista, Patrones S_1 y S_2.

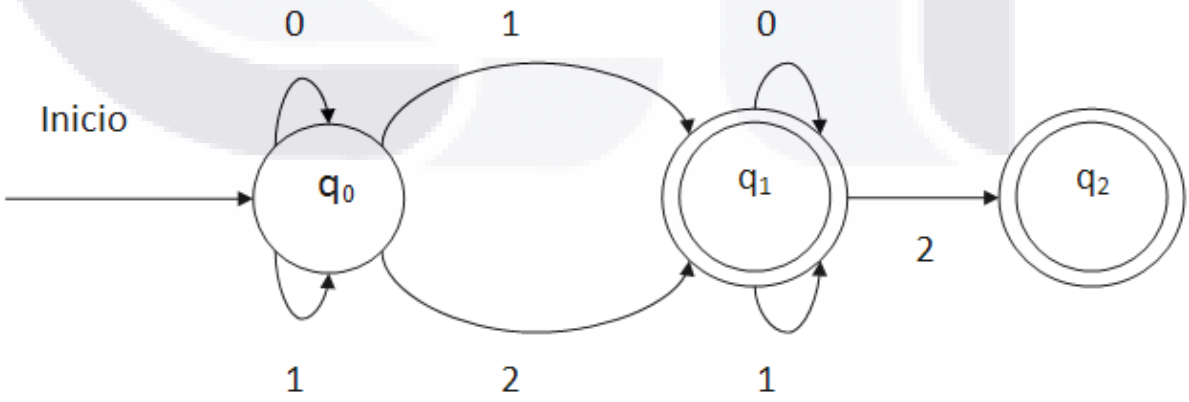


Figura 4.11 Autómata Finito no Determinista, Patrones S_1, S_2 y S_3.

Capítulo 5

SOFTWARE GENERADOR DE CÓDIGO 3OT

Para la realización del trabajo de experimentación se ha desarrollado un software generador de código 3OT, en este capítulo se describe el sistema programado.

5.1. Descripción.

El software generador de código 3OT se programó usando el lenguaje de programación C#, siguiendo una metodología orientada a objetos. Este software consiste de una ventana principal con cuatro pestañas:

5.1.1. Pestaña 3OT.

En esta pestaña del software se muestra la interfaz para codificar / decodificar el código 3OT, consta de dos paneles principales; en el primero que se encuentra en la parte superior de la ventana donde se cargan imágenes en formato JPG o BMP que contengan objetos binarios o se muestran las imágenes decodificadas a partir de las cadenas 3OT y en el segundo panel, ubicado en la parte inferior de la ventana, es donde se muestra o se establece el código 3OT. Entre ambos paneles se encuentran controles para la codificación y decodificación, en la parte izquierda está el botón para generar el código, junto con dos check box con opciones para la generación, se tiene la opción de eliminar el ruido al codificar, esto es, eliminar del código generado ciertos patrones de caracteres que representan pixeles aislados en el contorno o bien pequeños hoyos de un pixel, también está la opción para codificar con o sin coordenadas de los pixeles iniciales de las fronteras de los objetos y hoyos en la imagen; en la derecha está el botón para generar la imagen a partir del código insertado, se tiene la opción de generar únicamente el contorno, sin hacer el rellenado de las fronteras. En la Figura 5.1 se muestra la pestaña 3OT.

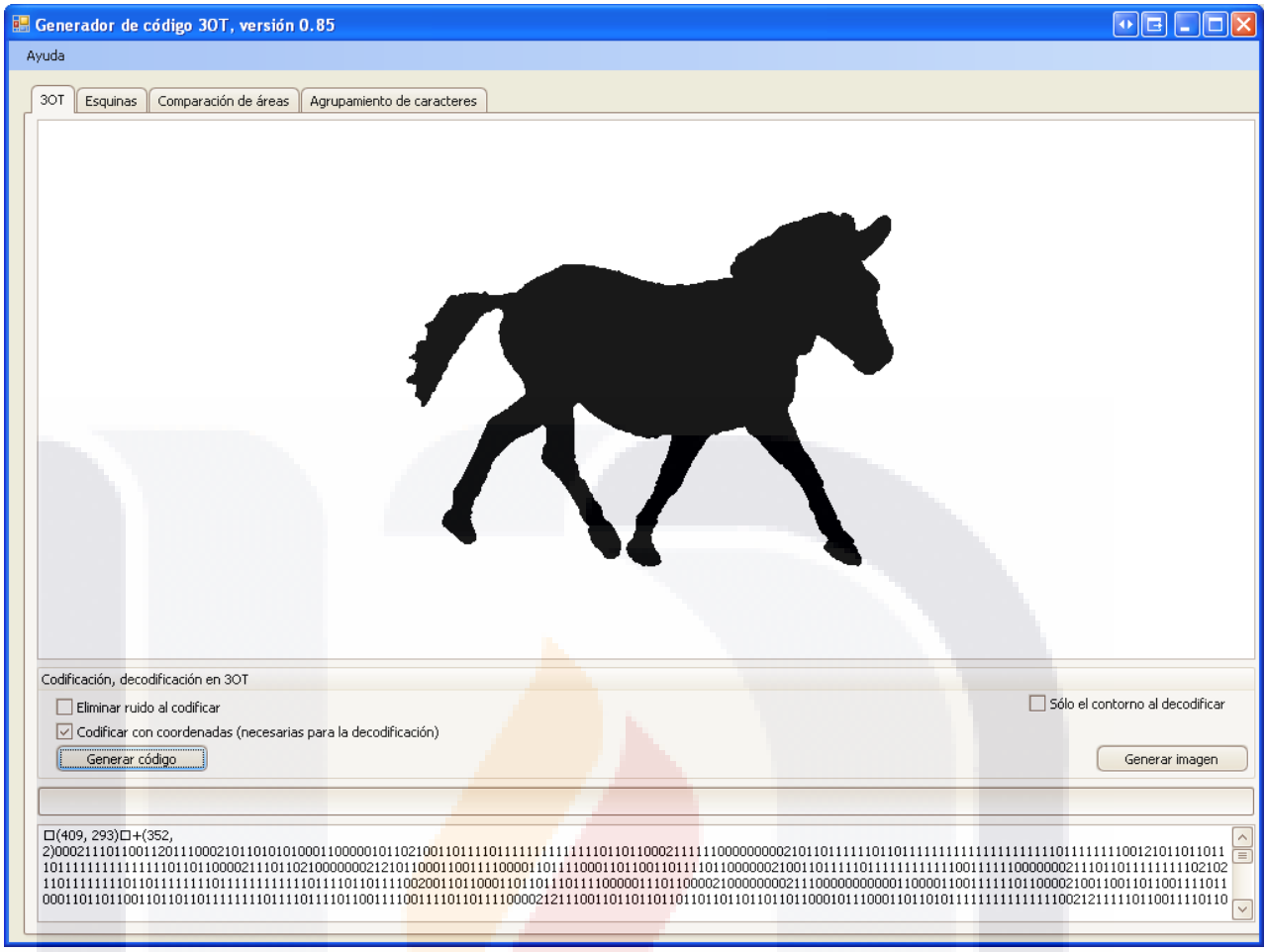


Figura 5.1 Software generador de código 3OT, pestaña 3OT.

5.1.2. Pestaña Esquinas.

En la pestaña de esquinas se presenta la interfaz de la herramienta para la detección de puntos dominantes basada en 3OT, para poder utilizarla es necesario haber generado previamente el código 3OT para alguna imagen binaria en la pestaña anterior. Esta pestaña tiene, en la parte superior izquierda la definición de los patrones a buscar dentro del código 3OT, dichos patrones se cargan a partir de algún archivo de texto, para seleccionar el archivo de texto a usar se usa el botón de "Cargar patrones", al hacer click en el botón aparece el cuadro de dialogo para seleccionar el archivo, una vez seleccionado el archivo con los patrones se puede decidir cuales usar haciendo click en el check box correspondiente. En la parte inferior a los patrones se muestran opciones para la detección de puntos dominantes, aquí se decide si se va a tomar sólo el inicio y fin para los grupos de subcadenas mayores a ν y

si se va o no a promediar los grupos de subcadenas menores a ν; tiene también un spin edit donde se decide el tamaño de ν y un botón para generar las subcadenas, una vez presionado este botón, comenzará la búsqueda de las subcadenas que se ajusten a los parámetros especificados. Las subcadenas generadas se muestran en el list box en la parte derecha de la pantalla, donde se especifica para cada subcadena si representa un punto dominante, un candidato a punto dominante o si no se ajusta a ningún patrón. Se puede seleccionar cualesquiera subcadenas y el punto pivote que representan será graficado en el par de paneles de la parte inferior de la pestaña, el panel de la izquierda contiene la imagen original y el de la derecha una imagen del mismo tamaño que la original pero con todos sus pixeles apagados. Una vez seleccionada más de una subcadena, se puede mandar a unir con líneas rectas los puntos graficados usando el botón de "Unir puntos seleccionados".

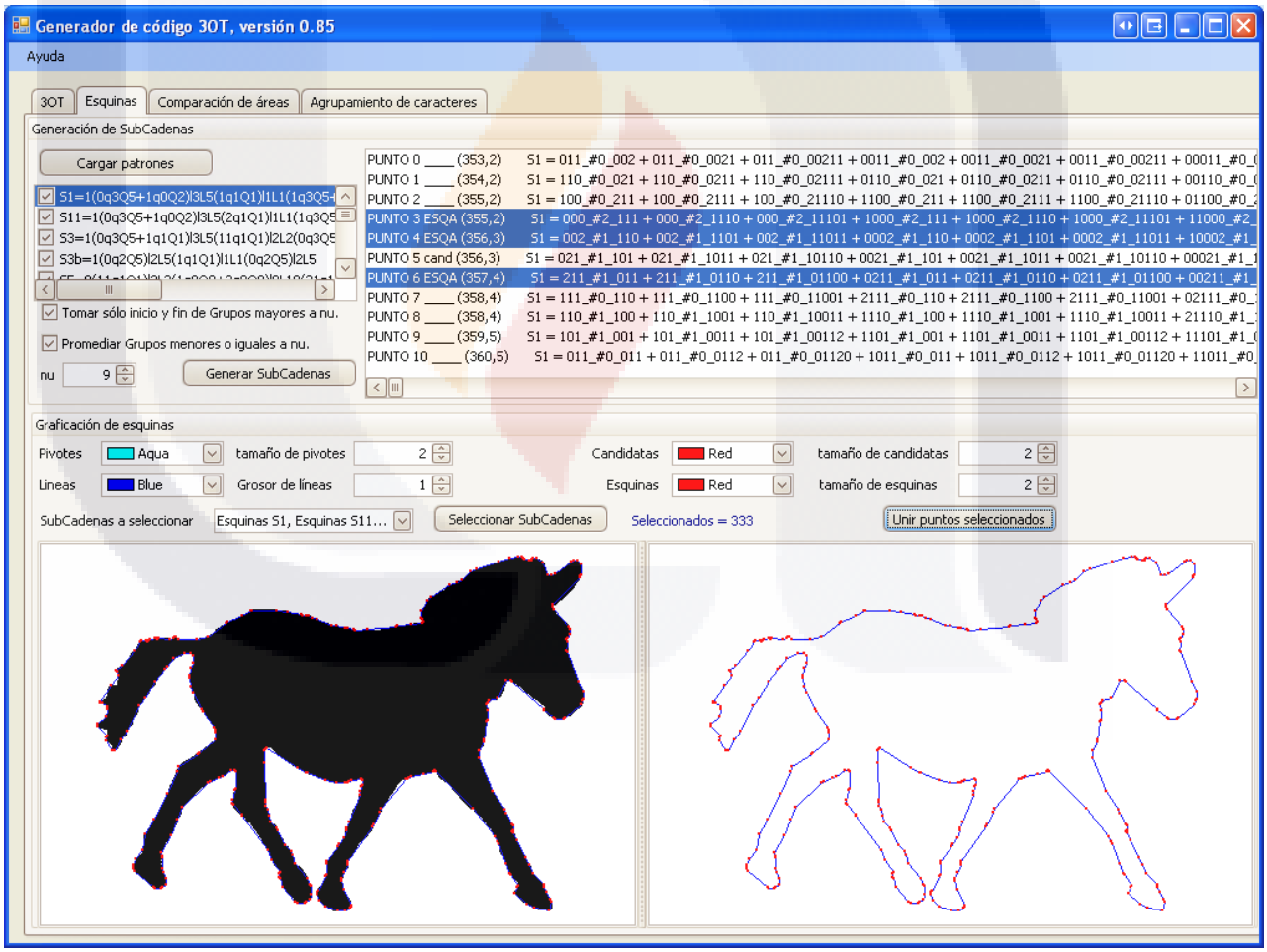


Figura 5.2 Software generador de código 3OT, pestaña Esquinas.

También se puede mandar seleccionar las subcadenas que sean candidatas y/o puntos dominantes de uno o varios patrones de manera automática, esto se logra con el check combo box de "Subcadenas a seleccionar", donde se puede decidir qué tipo de subcadenas seleccionar. Se puede especificar, además, el tamaño y color de los puntos seleccionados, así como de las líneas que unirán estos puntos usando los controles en la parte de en medio de la pestaña. En la Figura 5.2 se presenta la pestaña de Esquinas.

5.1.3. Pestaña Comparación de áreas.

En esta pestaña se muestra un par de paneles para imágenes, en la izquierda está la imagen original y en la derecha el polígono generado con las esquinas encontradas en la pestaña anterior.

Figura 5.3 Software generador de código 3OT, pestaña Comparación de áreas.

Esta pestaña muestra la cantidad de pixeles encendidos en la imagen original y los pixeles encendidos en el polígono. Además, sirve de apoyo para medir que tanto se asemejan ambas imágenes al contabilizar los pixeles en que coinciden ambas (PA), los pixeles encendidos en la original y apagados en el polígono (P+) y los pixeles encendidos en el polígono y apagados en la original (P-). La pestaña de comparación de áreas se muestra en la Figura 5.3.

5.1.4. Pestaña Agrupamiento de caracteres.

Esta pestaña es de apoyo para trabajos de experimentación futuros con el código 3OT que están fuera del alcance de esta tesis, se pretende generar variaciones al código 3OT agrupando por subcadenas y asignando nuevos símbolos a estos grupos. La Figura 5.4 muestra la pestaña de agrupamiento de caracteres.

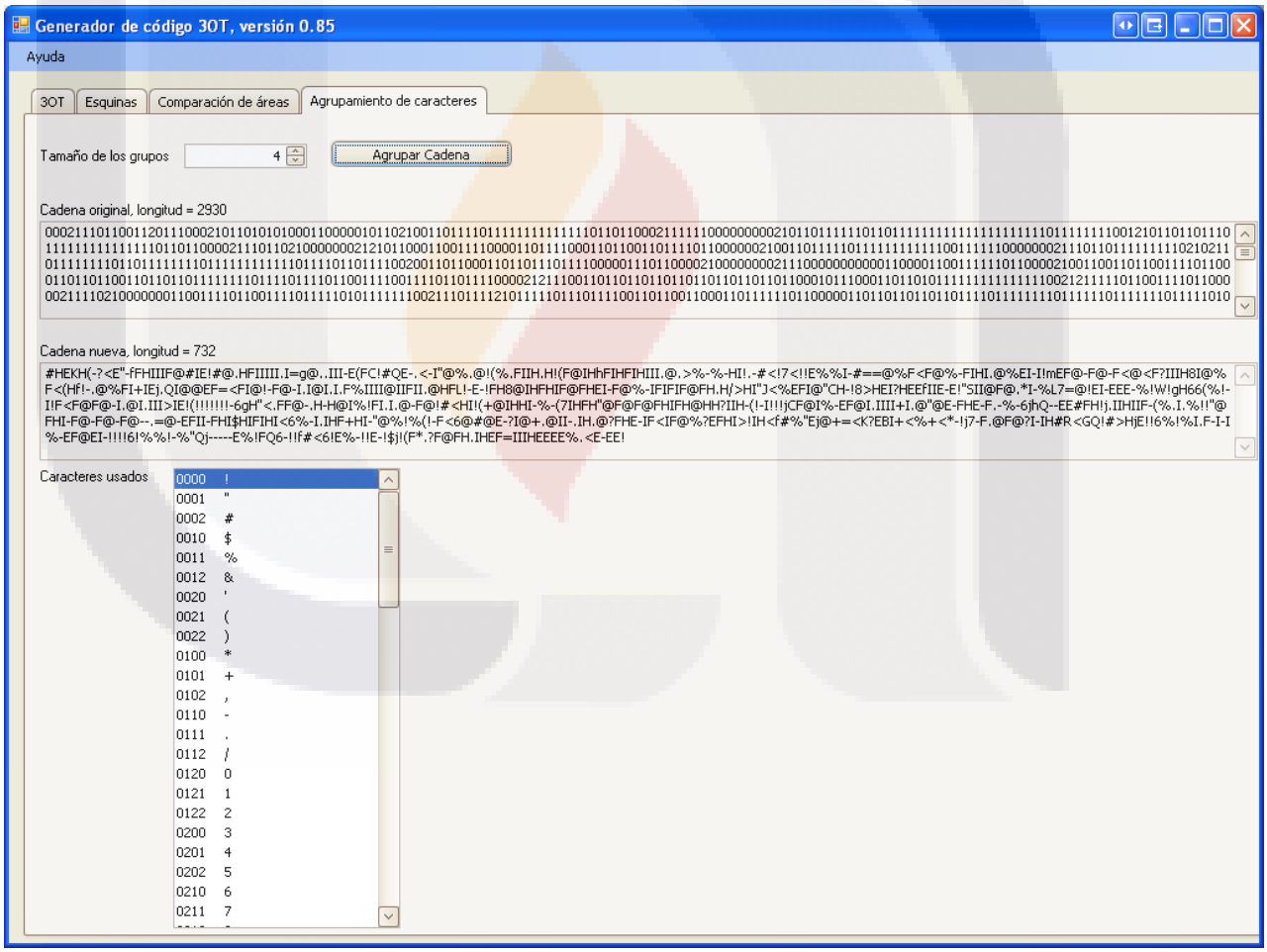


Figura 5.4 Software generador de código 3OT, pestaña Agrupamiento de caracteres.

5.2. Definición de nuevos patrones de búsqueda.

Los patrones S1, S2 y S3 que se explicaron en el capítulo anterior fueron implementados en el software generador de código 3OT, cuando se hizo evidente la necesidad de implementar nuevos patrones se encontró el inconveniente de tener que modificar el código fuente del sistema y tener que recompilar el programa. De ahí surgió la idea, que pronto se convirtió en uno de los objetivos de esta tesis, de desarrollar una metodología que permitiera definir nuevos patrones a usar en el software generador de código 3OT de manera independiente a la programación. Para lograr esto se pensó que lo más conveniente sería usar archivos de entrada definidos por el usuario que contengan las reglas para los patrones. La manera de solucionar esto fue crear una serie de reglas, a manera de expresiones regulares, para especificar los patrones. Para poder explicar cómo se definen los nuevos patrones, es necesario explicar algunos conceptos nuevos usados.

Definición 5.1 Secuencia de caracteres.

Una secuencia de caracteres es una regla que especifica que un grupo o cadena de caracteres, en este caso pertenecientes al alfabeto Ω, determinada tiene que aparecer cualquier cantidad de veces mayor o igual a q y menor o igual a Q dentro de otra cadena de caracteres del mismo alfabeto para decir que se cumple. En la Fórmula 5.1 se muestra las veces que aparece una secuencia de caracteres dentro de una cadena.

$$q \leq apariciones \leq Q$$

Fórmula 5.1 Veces de aparición de una secuenia de caracteres.

Para definir una secuencia de caracteres primero hay que especificar el grupo de caracteres de la secuencia, después se usa la letra q seguida de su tamaño y por último la letra Q seguida de su tamaño. Por ejemplo:

- 0q3Q10 significa que la secuencia "0" tiene que aparecer cualquier cantidad de veces mayor o igual a tres y menor o igual a diez dentro de otra cadena.

Cumplen	No cumplen
000121	0012122
0122000	21210112

- 0q3Q3 significa que la secuencia “0” tiene que aparecer tres veces dentro de otra cadena.

Cumplen	No cumplen
021020	01222
00011122	00
202001122	0000

- 21q4Q5 significa que la secuencia “21” debe de aparecer cuatro o cinco veces dentro de otra cadena.

Cumplen	No cumplen
21212121	000
212021222211121121	2121012
1212100021211121	201201201201

- 2q0Q0 significa que la secuencia “2” no puede aparecer dentro de otra cadena

Cumplen	No cumplen
12121121212	121211212120
221212112121	2212121012121

Es posible unir varias secuencias de caracteres usando el operador de +, al unir varias secuencias, se debe cumplir con cada una de ellas. Por ejemplo,

- 0q2Q3+12q1Q1 la secuencia “0” debe aparecer dos o tres veces y la secuencia “12” debe aparecer una vez.

Cumplen	No Cumplen
00012	121200
120022	000012
2120100	120

- 1q0Q0+2q0Q0 No pueden aparecer las secuencias “1” y “2”

Cumplen	No cumplen
000000000	20001
00	0000011

Definición 5.2 Condición.

Una condición especifica las secuencias de caracteres que tendrá cada una de las partes (inicio de subcadena, pivote de subcadena o fin de subcadena) de una subcadena a analizar, también especifica el tamaño en caracteres que tendrá la parte definida, que será mayor o igual a l y menor o igual a L. En la Fórmula 5.2 se muestra el tamaño que puede tener la parte de una subcadena definida por una condición.

$$l \leq tamaño \leq L$$

Fórmula 5.2 Tamaño de la parte de una subcadena.

Para definir una condición se escriben entre paréntesis las secuencias de caracteres que deberán cumplirse, seguido de la letra l con su tamaño en número y por último la letra L con su número. Por ejemplo:

- ()l8L8 Cualquier cadena de tamaño ocho.

Cumplen	No cumplen
01222210	1221021
00001220	100001200
22120012	122

- (0q2Q3+12q1Q1)l5L10 la secuencia “0” debe aparecer dos o tres veces y la secuencia “12” debe aparecer una vez dentro de una cadena que puede ser de tamaño mayor o igual a cinco y menor o igual a diez.

Cumplen	No cumplen
00122	0012
2221200220	00022
1202202	22212002201

- (1q0Q0+2q0Q0)l1L3 no pueden aparecer las secuencias “1” y “2” dentro de una cadena que puede ser de tamaño mayor o igual a uno y menor o igual a dos.

Cumplen	No cumplen
0	01
00	102

- (012q2Q2)l3L5 la secuencia “012” debe aparecer dos veces dentro de una cadena de tamaño mayor o igual a tres y menor o igual a cinco.
  Para esta condición no existe cadena que la cumpla.

Definición 5.3 Regla.

Una regla es un conjunto de tres condiciones que especifican cada una de las partes de una subcadena: inicio de subcadena, pivote de subcadena y fin de subcadena.

Las subcadenas que se analizan con estos patrones son parte de la cadena en código 3OT que representa a una imagen binaria, por lo tanto, cada caracter de la subcadena analizada especifica a algún pixel. En la regla también se especifica que caracter se usará para marcar el pixel que será el punto dominante encontrado: si se usa el índice 0, el punto dominante será el pixel representado por el primer caracter del inicio de subcadena; con 1, el punto dominante será el pixel representado por el primer caracter del pivote de subcadena; con 2, el punto dominantes será el pixel representado por el primer caracter del fin de subcadena y con 3 el punto dominante será el pixel representado por el último caracter del fin de subcadena.

En una regla, cada una de las tres condiciones se analizan por separado, para que una subcadena cumpla con una regla deben cumplirse las tres condiciones.

Para especificar una regla se escribe primero el índice a graficar, seguido de las condiciones en orden para el inicio, pivote y fin de subcadena. Por ejemplo:

- 1()l5L5(0q0Q0+1q0Q0)l1L1()l5L5 representa al patrón S_1, visto en la sección anterior, de longitud 11, donde se graficará el primer carácter del pivote
- 2(11q1Q1)l2L2(1q0Q0+2q0Q0)1L5(21q1Q1)l2L2 representa una regla donde como inicio de subcadena está la secuencia “11” una vez, como pivote se encuentra cualquier cantidad mayor a 0 y menor a 6 de “0” y como fin de subcadena una secuencia “21” y se graficará el pixel representado por el carácter “2”.
  Se muestran todas las subcadenas que cumplen con la regla:
  11012
  110012
  1100012
  11000012

110000012

Definición 5.4 Patrón.

Un patrón especifica una o más reglas que pueden aplicarse sobre las subcadenas, para decir que se cumple un patrón la subcadena necesita cumplir con por lo menos una de las reglas. Las diferentes reglas se definen separándolas con el caracter *. Para definir un patrón, primero se escribe el nombre que tendrá seguido del símbolo de = y luego se escriben las reglas. Por ejemplo:

- S2=1(0q3Q5)l5L5(0q0Q0+2q0Q0)l1L1(1q3Q5)l5L5*1(1q3Q5)l5L5(0q0Q0+2q0Q0)l1L1(0q3Q5)l5L5 especifica al patrón S_2, visto en la sección anterior, de longitud 11, donde se graficará el primer caracter del pivote.

Capítulo 6

REPORTE Y DISCUSIÓN DE RESULTADOS

En el presente capítulo se muestran los resultados obtenidos de los experimentos con el compresor 3OT-Aritmético y con el algoritmo para la detección de puntos dominantes basado en 3OT para imágenes con objetos binarios. Se describen los objetos de medición usados, los parámetros de medición empleados y se muestran los resultados obtenidos para cada experimento.

6.1. Análisis y Discusión de Resultados

Para poder decir que un algoritmo es mejor que otro es necesario manejar algunos términos apropiados.

Definición 6.1 Eficiencia.

Son los recursos computacionales que necesita la ejecución de un programa de computadora (Brassard & Bratley, 1996), por ejemplo, tiempo de CPU, uso de memoria, ancho de banda, etc.

Para el análisis de la eficiencia del presente trabajo, se medirá un solo recurso: el uso de memoria de almacenamiento, ya que el objetivo principal es la compresión de las imágenes binarias.

Definición 6.2 Eficacia.

La eficacia de un algoritmo va enfocada a sus resultados, entre más cerca se encuentren a la solución óptima, más eficaz será el algoritmo.

Es muy importante tener en cuenta este parámetro de medición para la detección de puntos dominantes, ya que por ser un método de compresión con pérdida de información es necesario tener cuidado de no alejarse demasiado del resultado óptimo.

Definición 6.3 Robustez.

Es el balance entre la eficiencia y eficacia necesarias para la sobrevivencia en muchos ambientes diferentes (Goldberg, 1989).

Es de suma importancia que los métodos de compresión de datos puedan emplearse para muestras de datos variados para no limitar su uso a datos que reúnan ciertas características.

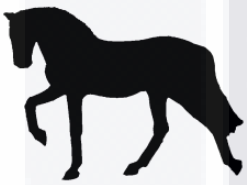
6.1.1. Resultados del método de compresión 3OT-Aritmético

En (Sánchez-Cruz, et al, 2007) se comparó la capacidad de compresión de siete códigos de cadena recientes, incluyendo el 3OT y también el método JBIG; después le aplicaron el algoritmo de compresión de Huffman a las cadenas de todos los códigos usados, sus experimentos arrojaron mejores resultados que el compresor JBIG. Encontraron que los mejores códigos para representar objetos binarios fueron el DFCCE y 3OT comparados con el JBIG considerando objetos con una 8-frontera no mayor a un umbral de 13000 pixeles.

La presente investigación se llevó a cabo utilizando los códigos DFCCE y 3OT, ya que fueron los dos mejores códigos en (Sánchez-Cruz, et al, 2007). Más aún, el alfabeto Ω, del código 3OT, está compuesto de sólo tres símbolos, lo que lo hace adecuado para ser manejado con codificación aritmética, mejor aún que los códigos compuestos por alfabetos más grandes, incluyendo el DFCCE, que tiene ocho símbolos.

Una vez que se han obtenido los códigos 3OT y DFCCE, se aplicó el algoritmo Aritmético a las cadenas resultantes y se encontró que este método tiene un mejor desempeño que el JBIG. Para comparar los resultados con la compresión propuesta en (Sánchez-Cruz, et al, 2007) basada en el algoritmo de Huffman aplicado al DFCCE, también se computó dicho método. Así, se obtuvo una cantidad M_{CODE}, dada en bytes. También, se aplicó el compresor JBIG y se obtuvo la cantidad M_{JBIG}, en bytes también. Se procesaron con estos métodos de compresión 16 objetos binarios, que se aprecian en la Tabla 6.1 junto con la información de su tamaño original.

Objeto	Tamaño imagen	8-Frontera del objeto	Imagen del objeto escalada
Ant	235 ×250	1484	
Bat	551 ×267	1444	
Btrfy	667 ×822	2682	
Btrfly2	600 ×451	1473	
Bus	300 ×250	653	
Camel	640 ×726	3446	

Snail	640 ×633	2557	
Coco	302 ×87	773	
Football	640 ×850	3482	
Dog	694 ×851	4634	
Horse	814 ×600	3679	
Lion	382 ×380	1577	
Plane	1801 ×1039	9591	

Map	648 ×648	4140	
Moto	960 ×738	5954	
Skull	1391 ×1333	6861	

Tabla 6.1 Objetos de medición, Resultados 3OT-Aritmético.

Para analizar los resultados, es necesario definir la eficiencia de compresión con respecto al JBIG.

Definición 6.4 Eficiencia con respecto al JBIG.

La eficiencia de compresión para imágenes binarias de algún código (CODE) con respecto del estándar JBIG se define como en la Fórmula 6.1.

$$Eficiencia\ M_{CODE} = 1 - M_{CODE} / M_{JBIG}$$

Fórmula 6.1 Eficiencia con respecto al JBIG.

En la Tabla 6.2 y Tabla 6.3 se reportan los principales resultados de este trabajo. Se presentan los valores de la 8-frontera, también el tamaño en memoria resultante de aplicar cada método de compresión comparado: 3OT-Aritmético, DFCCE-Aritmético, DFCCE-Huffman y JBIG, y la eficiencia con respecto al JBIG. Como puede apreciarse el 3OT-Aritmético y el DFCCE-Aritmético mejoran los niveles de compresión y son mejores que el DFCCE-Huffman y el que el JBIG.

Objeto	8-Frontera	M_{JBIG}	$M_{DFCCE\text{-}Huffman}$	$M_{DFCCE\text{-}Aritmético}$	$M_{3OT\text{-}Aritmético}$
Ant	1484	398	336	311	**309**
Bat	1444	392	323	297	**284**
Btrfy	2682	694	608	532	**507**
Btrfly2	1473	439	328	306	**283**
Bus	653	205	133	129	**115**
Camel	3446	746	715	662	**659**
Snail	2557	658	548	512	**502**
Coco	773	230	159	154	**145**
Football	3482	817	728	**674**	702
Dog	4634	1101	1001	936	**883**
Horse	3679	776	783	722	**680**
Lion	1577	435	356	338	**322**
Plane	9591	2211	2190	1957	**1789**
Map	4140	1031	847	**793**	848
Moto	5954	1391	1315	1211	**1133**
Skull	6861	1453	1358	**1210**	1298

Tabla 6.2 Tamaño en bytes de los métodos de compresión, Resultados 3OT-Aritmético.

Objeto	8-Frontera	M_{JBIG}	Eficiencia $M_{DFCCE\text{-}Huffman}$	Eficiencia $M_{DFCCE\text{-}Aritmético}$	Eficiencia $M_{3OT\text{-}Aritmético}$
Ant	1484	398	0.16	0.22	**0.22**
Bat	1444	392	0.18	0.24	**0.28**
Btrfy	2682	694	0.12	0.23	**0.27**
Btrfly2	1473	439	0.25	0.30	**0.36**
Bus	653	205	0.35	0.37	**0.44**
Camel	3446	746	0.04	0.11	**0.12**
Snail	2557	658	0.17	0.22	**0.24**
Coco	773	230	0.31	0.33	**0.37**
Football	3482	817	0.11	**0.18**	0.14
Dog	4634	1101	0.09	0.15	**0.20**
Horse	3679	776	-0.01	0.07	**0.12**
Lion	1577	435	0.18	0.22	**0.26**
Plane	9591	2211	0.01	0.11	**0.19**
Map	4140	1031	0.18	**0.23**	0.18
Moto	5954	1391	0.05	0.13	**0.19**
Skull	6861	1453	0.07	**0.17**	0.11

Tabla 6.3 Eficiencia con respecto al JBIG de los métodos de compresión, Resultados 3OT-Aritmético.

Como puede verse, en la Figura 6.1, existe una relación lineal entre el algoritmo Aritmético, aplicado al 3OT; los Algoritmos de Huffman y Aritmético aplicados al DFCCE; y el JBIG, con el tamaño de la 8-frontera.

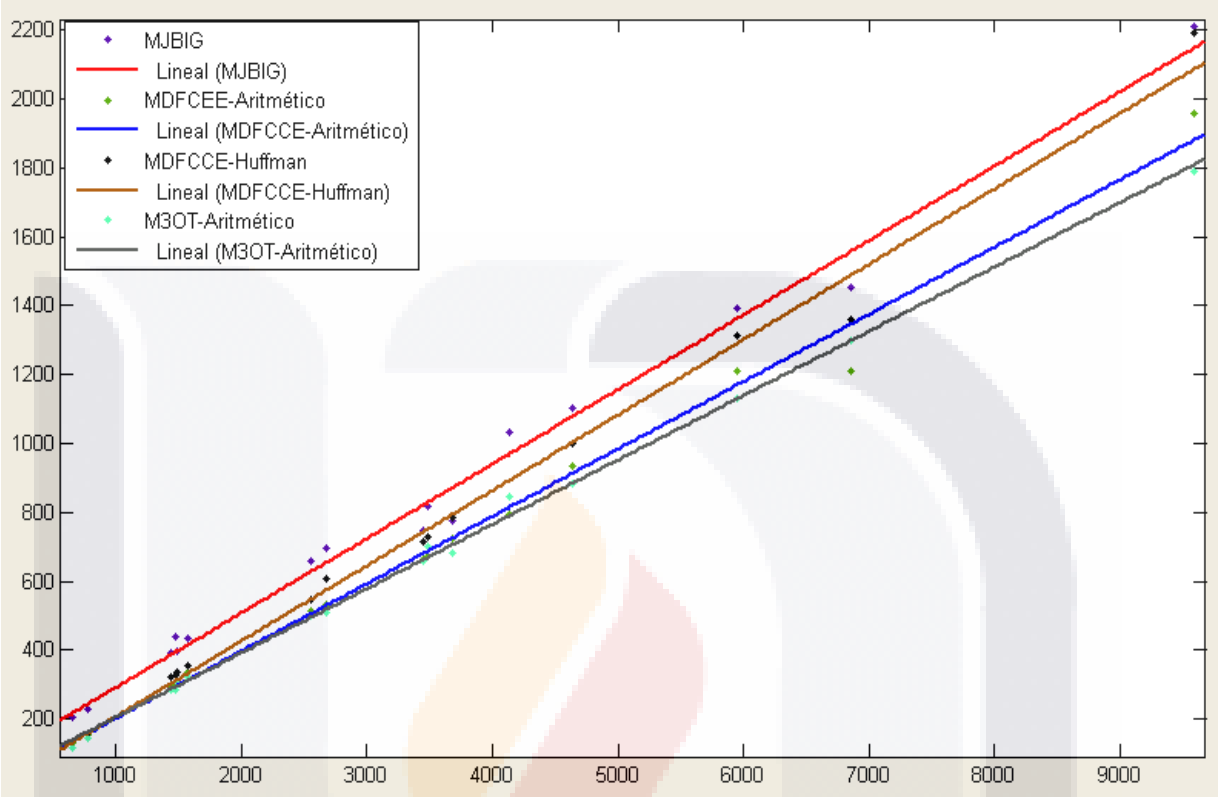


Figura 6.1 Relación del MCODE con la 8-frontera, Resultados 3OT-Aritmético.

Mientras que la Figura 6.2 muestra una relación exponencial de la forma de la Fórmula 6.2 entre la eficiencia de $M_{DFCCE\text{-}Aritmético}$ y $M_{DFCCE\text{-}Huffman}$ con respecto al JBIG y la 8-frontera y una relación gaussiana de la forma de la Fórmula 6.3 con respecto al JBIG y la 8-frontera.

$$y = a \times e^{(b \times x)} + c \times e^{(d \times x)}$$

Fórmula 6.2 Relación exponencial de la eficiencia $M_{DFCCE\text{-}Aritmético}$ y $M_{DFCCE\text{-}Huffman}$ vs 8-frontera.

$$y = a_1 \times e^{-(\frac{x-b_1}{c_1})^2} + a_2 \times e^{-(\frac{x-b_2}{c_2})^2} + a_3 \times e^{-(\frac{x-b_3}{c_3})^2}$$

Fórmula 6.3Relación gaussiana de la eficiencia M3OT-Aritmético vs 8-frontera

Para el caso del DFCCE-Huffman, aparece un comportamiento similar en (Sánchez-Cruz, et al, 2007), en donde la eficiencia de DFCEE y la 8-frontera fue graficada, y se aplicó el algoritmo de Huffman. Sin embargo, la mejora ahora es que el gráfico está más alejado de la eficiencia cero, y es dado por el código 3OT-Aritmético.

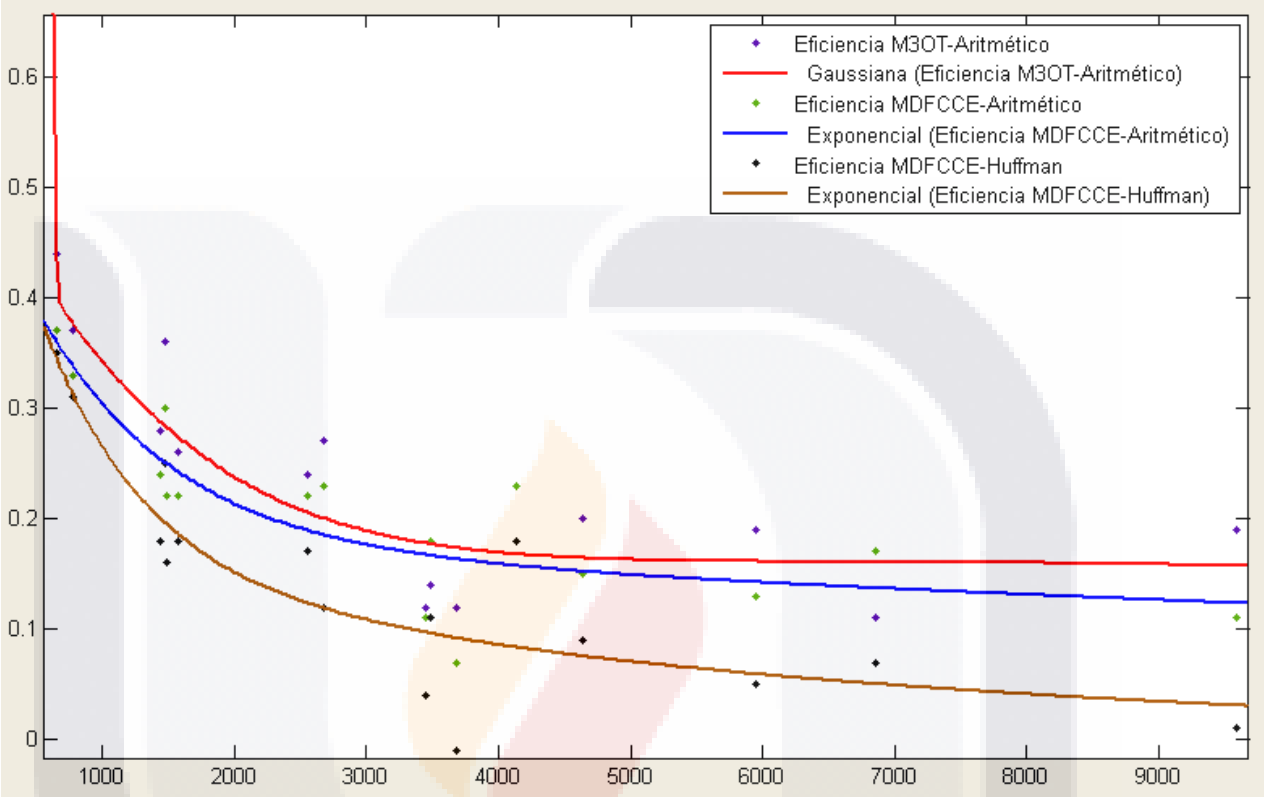


Figura 6.2 Relación de la eficiencia con la 8-frontera, Resultados 3OT-Aritmético.

Es importante observar en la Figura 6.1, que la tendencia del código DFCCE-Huffman sugiere que se cruzará con la tendencia del JBIG, por supuesto, en un punto más alejado de las 9000 unidades de la 8-frontera, un comportamiento similar se encontró en (Sánchez-Cruz, et al, 2007) para este código. Sin embargo, la tendencia del 3OT-Aritmético y DFCCE-Aritmético sugiere que nunca se cruzarán con la tendencia del JBIG, más aún, la pendiente para la función ajustada del 3OT-Aritmético es la menor de las cuatro.

Este análisis nos permite decir que es mejor usar el código 3OT para representar imágenes binarias siempre que haya objetos irregulares en ellas. Este análisis y la tendencia de la eficiencia sugieren que para todas las fronteras codificadas con 3OT, en que el algoritmo de

compresión de cadenas aritmético sea aplicado se tiene mejor desempeño de compresión que el JBIG, incluyendo fronteras más grandes que las encontradas en la referencia (Sánchez-Cruz, et al, 2007).

También, la distribución de puntos obtenida, sugiere que hay una muestra de objetos mayor para ser representada por el 3OT y que es mejor que la del DFCCE encontrada en (Sánchez-Cruz, et al, 2007), en donde el algoritmo de Huffman fue el más efectivo para comprimir objetos binarios.

Está claro que para cada hoyo y cada objeto en una imagen se necesitan algunos bits extras para representar las coordenadas de inicio. En el caso del objeto Moto, se codificaron 24 hoyos. Cada hoyo codificado requiere dos coordenadas de inicio, en el "peor de los casos" 960 columnas y 738 líneas pueden ser codificadas por 20 bits, multiplicados por 24, dan 480 bits. De esta manera se requerirán 60 bytes extras para codificar el objeto de la moto. Obviamente, no todos los 24 hoyos tienen coordenadas (960, 738), esta cantidad es el "peor de los casos", y, sin embargo, esto no cambia las tendencias encontradas.

6.1.2. Resultados del método de detección de puntos dominantes usando el código 3OT

Uno de los principales objetivos de este trabajo fue la implementación del método para la detección de puntos dominantes con 3OT para poder compararlo contra los diferentes métodos ya existentes; para lograrlo se ha desarrollado una medida que nos permite ver que tan bueno es un algoritmo de detección de puntos dominantes con respecto a los demás.

Para poder decir que un método de detección de puntos dominantes es más robusto que otro es necesario comparar los resultados que generan, por lo que se debe analizar el conjunto de puntos dominantes encontrados usando el código 3OT y los grupos de puntos encontrados por otros métodos.

Evidentemente, una buena forma de medir la eficacia de un conjunto de puntos dominantes es ver que tanto se asemeja o que tan diferente es el polígono P que estos representan con respecto al objeto O original, de manera que se creó el parámetro para medir el área de error ΔA, la cual es la suma de C^+ (pixeles existentes en la forma original, que no existen en el polígono generado) con C^- (pixeles existentes en el polígono generado, pero no

existen en la forma original). En la Fórmula 6.4 se muestra la expresión para los pixeles de diferencia entre el objeto original y el polígono generado.

$$\Delta A = C^{+} + C^{-}.$$

Fórmula 6.4 Diferencia de pixeles entre objeto original y polígono generado.

En la Figura 6.3 se muestra un objeto binario O con puntos dominantes marcados; en la Figura 6.4 se muestra el polígono formado por los puntos dominantes sobrepuesto en el mismo objeto; en la Figura 6.5 se muestran en color rojo los pixeles C^{+}, en azul los pixeles C^{-} y en negro la intersección del polígono generado y el objeto original $O \cap P$, en la Figura 6.6 se muestra el polígono generado con los puntos dominantes sobrepuesto sobre la frontera del objeto original. Para este ejemplo se usaron patrones de búsqueda modificados, intencionalmente malos para hacer notable la diferencia entre el polígono generado y el objeto original.

Figura 6.3 Imagen binaria con puntos dominantes, ardilla.

Figura 6.4 Imagen binaria con polígono aproximando, ardilla.

Figura 6.5 Imagen binaria con pixeles C^{+} y C^{-}, ardilla.

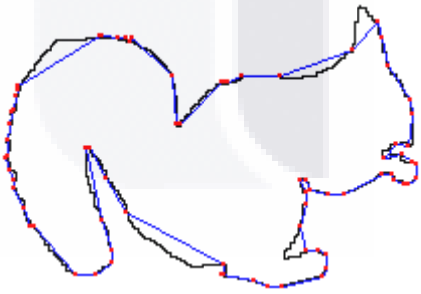


Figura 6.6 Frontera de imagen binaria con puntos dominantes y su polígono aproximado, ardilla

Entonces cuando ΔA tiende a cero el polígono se aproxima al objeto original, así que entre menor sea este valor es más eficaz el conjunto de puntos dominantes.

La distancia de Tanimoto (Tanimoto, 1957) satisface el concepto de una métrica y es usada en diferentes conjuntos que tengan elementos discretos, se calcula con la Fórmula 6.5.

$$D(P,O) = \frac{n_O + n_P - 2n_{O\cap P}}{n_O + n_P - n_{O\cap P}}$$

Fórmula 6.5 Distancia de Tanimoto.

Donde n_O y n_P son la cantidad de elementos de los conjuntos O y P respectivamente, y $n_{O\cap P}$ es el número de elementos en la intersección de ambos conjuntos.

Tomando esto en cuenta, se define el parámetro ε, que consiste en multiplicar $D(P, O)$ por la cantidad de puntos dominantes encontrados $\#DP$. El valor de ε se muestra en la Fórmula 6.6.

$$\varepsilon = D(P,O) \times \#DP$$

Fórmula 6.6 Fórmula para calcular la falta de robustez de un algoritmo de detección de puntos dominantes.

De manera que ε es inversamente proporcional a la robustez del algoritmo para la detección de puntos dominantes analizado.

Para la medición Se generaron los polígonos aproximados para ocho objetos binarios con los métodos clásicos IPAN(Image and Pattern Analysis Group), (Rosenfeld & Johnston, 1973), (Rosenfeld & Weszka, 1975), (Freeman & Davis, 1976), (Beus & Tiu, 1987) para la detección de puntos dominantes en fronteras de objetos utilizando las herramientas encontradas en la página de IPAN, http://visual.ipan.sztaki.hu/corner/corner_click.html, el formato en que la página regresa los resultados es el mostrado en la Figura 6.8 donde se ve un ejemplo de salida para la entrada del objeto mostrado en la Figura 6.7.

Figura 6.7 Imagen binaria, Benito.

Figura 6.8 Contorno con puntos dominantes, Benito.

Este formato de salida no puede ser utilizado para la comparación de algoritmos para la detección de puntos dominantes propuesta, debido a que no es el polígono aproximado formado por la unión de los puntos dominantes, únicamente sobrepone los puntos sobre la frontera del objeto original, Para generar el polígono aproximado se creó un programa de computadora en C++ Builder que se explica en el Pseudocódigo 6.1.

Generación de polígono aproximado a partir del formato de IPAN Genera el polígono aproximado que representa la salida de datos devuelta por la página de IPAN.		
Variables	**Tipo**	**Descripción**
Entrada	Bitmap	Tiene almacenada la imagen binaria original.
CodigoFreeman	String	Almacena el código de Freeman de la imagen Entrada.
Salida	Bitmap	Tiene almacenada la imagen devuelta por la página de IPAN.
Coordenadas	Vector de pares de enteros.	Almacena las coordenadas de los puntos dominantes localizados en Salida.

Pseudocódigo

1. Generación del código de Freeman para el objeto de la imagen de Entrada.
2. Utilizando el código de Freeman que se obtuvo en el paso 1 se puede recorrer el contorno de la figura de Salida y conforme se encuentren los puntos marcados se guardan en Coordenadas.
3. Se utiliza Coordenadas para unir los puntos en el orden en que se encontraron.

Pseudocódigo 6.1 Decodificación de código aritmético.

La Figura 6.9 muestra el polígono aproximado después de procesar con el Pseudocódigo 6.1 las imágenes de Entrada y Salida mostradas en la Figura 6.7 y Figura 6.8 respectivamente.

Figura 6.9 Imagen binaria, polígono aproximado Benito.

Los objetos de medición que se usaron en la comparación de los algoritmos de detección de puntos dominantes se muestran en la Tabla 6.4 junto con su cantidad de pixeles 1s (n_O), se utilizaron estos objetos ya que son benchmarks clásicos utilizados por IPAN para usarse sobre algoritmos de detección de puntos dominantes.

Objeto	**Imagen del objeto escalada**	n_O
Pencils		48703
Benito		33869
Inglat		28446

Ballard		40703
Plane		20508
Circles		33843
Stick		5344
Buble		42818

Tabla 6.4 Objetos de medición, Resultados detección de puntos dominantes 3OT.

Después de contabilizar el ΔA para los polígonos resultantes y el objeto original y calcular ε con la Fórmula 6.6 y la distancia de Tanimoto con la Fórmula 6.5 para cada objeto y cada método, se obtuvieron los resultados que muestran en la Tabla 6.5

Imagen	Método	#PD	C^+	C^-	n_P	$n_{O \cap P}$	D(P,O)	ε
Pencils	IPAN	53	20	1241	49924	48683	0.0252	1.3382
48703	RJ	52	110	903	49496	48593	0.0204	1.0619
	RW	49	77	910	49536	48626	0.0199	0.9748
	FD	59	46	1179	49836	48657	0.0246	1.4489
	BT	49	307	920	49316	48396	0.0247	1.2116
	3OT	48	398	581	48886	48305	0.0199	0.9535
Benito	IPAN	68	265	713	34317	33604	0.0283	1.9231
33869	RJ	68	127	729	34471	33724	0.0258	1.7523
	RW	58	154	768	34483	33715	0.0266	1.5439
	FD	64	147	736	34458	33722	0.0255	1.6331
	BT	59	218	688	34339	33651	0.0262	1.5468
	3OT	64	680	368	33557	33189	0.0306	1.9591
Inglat	IPAN	76	434	918	28930	28012	0.0460	3.4993
28446	RJ	84	615	877	28708	27831	0.0509	4.2741
	RW	60	741	964	28669	27705	0.0580	3.4784
	FD	82	121	1123	29448	28325	0.0421	3.4498
	BT	96	69	973	29350	28377	0.0354	3.4003
	3OT	79	268	762	28940	28178	0.0353	2.7859
Ballard	IPAN	74	48	1021	41676	40655	0.0256	1.8959
40703	RJ	82	86	923	41540	40617	0.0242	1.9877
	RW	75	18	974	41659	40685	0.0238	1.7852
	FD	109	17	1076	41762	40686	0.0262	2.8516
	BT	83	272	880	41311	40431	0.0277	2.2994
	3OT	85	189	388	40902	40514	0.0140	1.1936
Plane	IPAN	80	219	1094	21383	20289	0.0608	4.8625
20508	RJ	93	536	1135	21107	19972	0.0772	7.1803
	RW	98	220	1039	21327	20288	0.0584	5.7262
	FD	114	146	1270	21632	20362	0.0650	7.4123
	BT	78	446	1245	21307	20062	0.0777	6.0634
	3OT	76	359	346	20495	20149	0.0338	2.5693

Imagen	Método	#PD	C^+	C^-	n_P	$n_{O \cap P}$	D(P,O)	ε
Circles	IPAN	55	125	754	34472	33718	0.0254	1.3974
33843	RJ	55	0	840	34683	33843	0.0242	1.3321
	RW	53	0	947	34790	33843	0.0272	1.4427
	FD	54	3	722	34562	33840	0.0210	1.1326
	BT	57	7	712	34548	33836	0.0208	1.1860
	3OT	59	359	232	33716	33484	0.0173	1.0233
Stick	IPAN	44	40	395	5699	5304	0.0758	3.3351
5344	RJ	53	32	436	5748	5312	0.0810	4.2913
	RW	54	1	470	5813	5343	0.0810	4.3746
	FD	34	61	467	5750	5283	0.0909	3.0893
	BT	36	77	430	5697	5267	0.0878	3.1611
	3OT	39	370	51	5025	4974	0.0780	3.0434
Bubble	IPAN	59	10	974	43782	42808	0.0225	1.3257
42818	RJ	53	7	973	43784	42811	0.0224	1.1861
	RW	55	32	1016	43802	42786	0.0239	1.3150
	FD	63	201	724	43341	42617	0.0212	1.3384
	BT	55	155	734	43397	42663	0.0204	1.1227
	3OT	43	408	79	42489	42410	0.0114	0.4882

Tabla 6.5 Robustez de los diferentes métodos de detección de puntos dominantes..

En la tabla 4.6 se muestra el promedio de ε y la distancia de Tanimoto de los ocho objetos analizados para cada método.

Métrica	IPAN	RJ	RW	FD	BT	3OT
ε	2.4471	2.8832	2.5801	2.7945	2.4989	**1.7520**
Distancia de Tanimoto	0.0387	0.0408	0.0399	0.0396	0.0401	**0.0300**

Tabla 6.6 Promedios de ε y la distancia de Taminoto, Resultados detección de puntos dominantes 3OT.

En la Figura 6.10 se muestra un histograma con los promedios de ε mostrados en la Tabla 6.6; en la Figura 6.11 se muestra un histograma con los promedios de la distancia de Tanimoto mostrados en la Tabla 6.6.

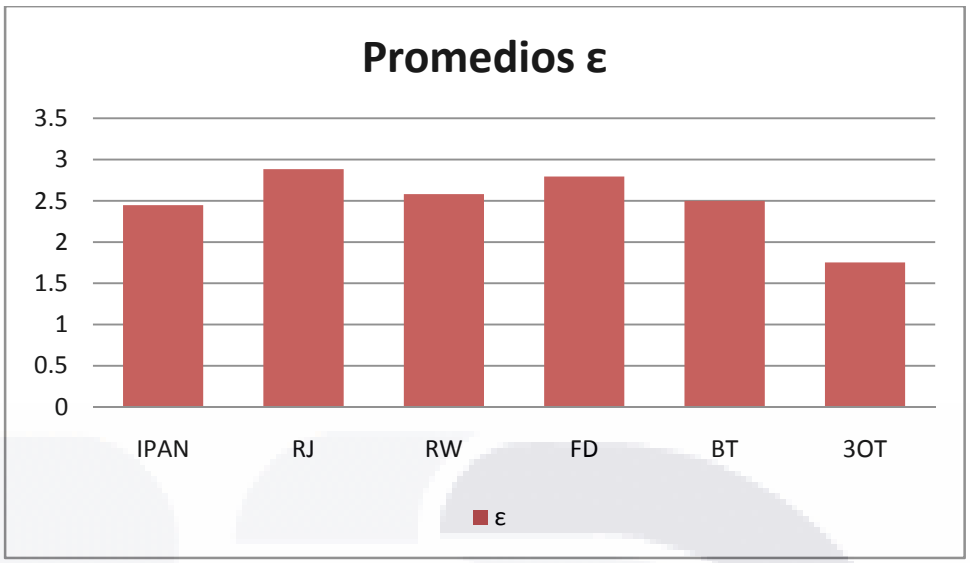


Figura 6.10 Promedios de ε, Resultados detección de puntos dominantes 3OT.

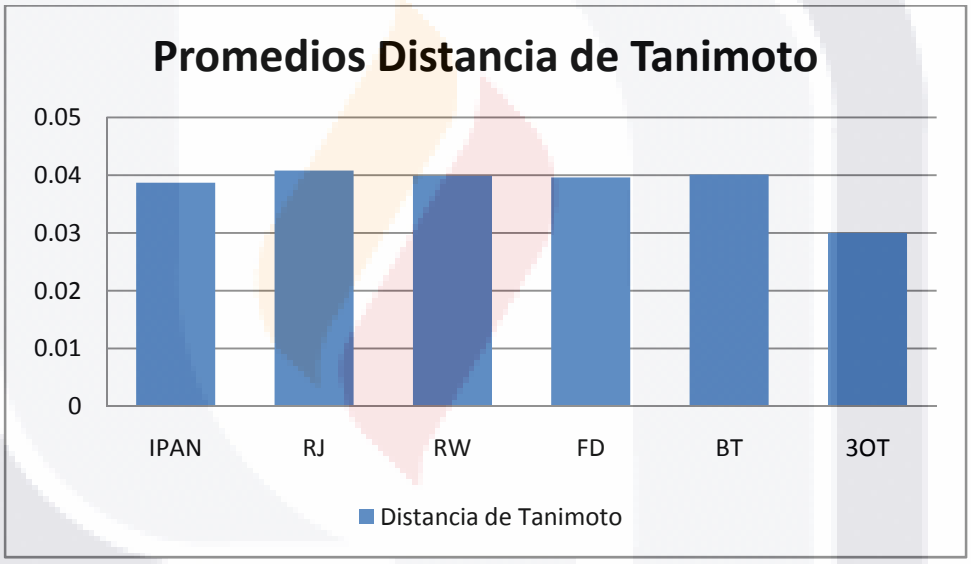


Figura 6.11 Promedios de la distancia de Tanimoto, Resultados detección de puntos dominantes 3OT.

Son muy importantes los resultados de ε y la distancia de Tanimoto obtenidos para el código 3OT, porque resultan ser mejor que los de los otros métodos analizados, los cuáles son de los métodos para la detección de puntos dominantes más referenciados en la literatura. El valor de ε para el 3OT (1.776) es menor que el ε para el segundo mejor método, el de IPAN (2.547), lo que significa una mejoría de 43%, ya que ε es inversamente proporcional a la robustez del algoritmo que evalúa, y con respecto a la distancia de Tanimoto (Tanimoto, 1957) (0.0387 para IPAN y 0.0300 para 3OT) se tiene una mejoría del 29%.

Capítulo 7

CONCLUSIONES

En este capítulo se explica el significado de los resultados obtenidos en el capítulo anterior, así como de sus implicaciones e importancia. También se comenta acerca del potencial que tiene el software generador de código 3OT, describiendo el trabajo futuro que se puede desarrollar usándolo como herramienta de apoyo.

7.1. Conclusiones de Resultados Obtenidos.

Una de las principales aportaciones del presente trabajo es que ha dado fruto a un par de herramientas: el software generador de código 3OT y la extensión del método para la detección de puntos dominantes. Además, estas herramientas se han aplicado con éxito en un par de experimentos: la compresión de imágenes binarias y la detección de puntos dominantes.

El software generador de código 3OT, permite codificar/decodificar imágenes binarias en forma sencilla y práctica, esto ha facilitado la investigación acerca de estas imágenes representadas con este código.

La extensión del método para la detección de puntos dominantes con el 3OT, es una herramienta con mucho potencial, ya que permite la fácil implementación de nuevos patrones, lo que facilita la investigación para determinar que patrones funcionan mejor para detectar los puntos dominantes. Con esto se puede buscar cuales son los mejores patrones especializados para objetos con características determinadas y cuales patrones generales para objetos con características diversas.

Se ha usado el algoritmo aritmético aplicado al código de cadena 3OT, y se han comparado los resultados con el compresor JBIG. Se encontró una mejor eficiencia de compresión que la obtenida al comprimir la cadena 3OT con el algoritmo de Huffman. Todos los objetos de medición fueron comprimidos mejor por el 3OT-Aritmético que por el estándar JBIG. Indudablemente, el 3OT-Aritmético y el DFCCE-Aritmético brindan mejores niveles de compresión comparados con el JBIG y con el DFCCE-Huffman, aún mas, 81% de los objetos

de la muestra fueron comprimidos mejor con el 3OT-Aritmético comparados con el DFCCE-Aritmético.

Los tamaños de las imágenes binarias estuvieron en el rango de 235×250 a 1801×1039, con 8-frontera de 1444 hasta 9591. Los objetos de muestra fueron elegidos de manera que no fueran geométricos, sino irregulares, esto evita la ventaja en redundancia a lo largo de las irregularidades en los contornos.

Para evaluar el desempeño del algoritmo para la detección de puntos dominantes se ha desarrollado una métrica propia, además de evaluar con la distancia de Tanimoto (Tanimoto, 1957). Se han usado imágenes benchmarks típicas, ya que es un problema que se ha atacado con métodos diversos, en la literatura los métodos más referenciados son los de IPAN, (Rosenfeld & Johnston, 1973), (Rosenfeld & Weszka, 1975), (Freeman & Davis, 1976), (Beus & Tiu, 1987), por lo tanto, al encontrar que el 3OT tiene un valor que es 43% con ε y 29% con la distancia de Tanimoto mejor que el método más óptimo de los anteriores es un resultado significativo.

7.2. Trabajo Futuro

Se conoce por (Alcaraz-Corona, et al, 2009) que el agrupamiento de símbolos de los códigos de cadena mejora los niveles de compresión en imágenes binarias. Actualmente, se está experimentando con el agrupamiento de símbolos en el código 3OT para la compresión de imágenes binarias, con el objetivo de comparar el rango de compresión con respecto al tamaño en pixeles de las fronteras de los objetos.

Como trabajo futuro se buscará aplicar el método de compresión de imágenes binarias 3OT-Aritmético en el caso de objetos que representen mapas y árboles.

También, se atacará el problema de representar con el 3OT-Aritmético imágenes más complejas; por ejemplo, aquellas con muchos más objetos y hoyos que las representadas aquí, ya que cada frontera ocupa un par de coordenadas de inicio, lo que requiere más espacio de almacenamiento, así que habrá que buscar la manera adecuada de almacenar dicha información de manera eficiente, para evitar que se dispare la cantidad de bits resultantes. En este punto también se buscará hacer recursiva la manera en que se codifican los objetos y

hoyos, para permitir codificar / decodificar imágenes aún más complejas, como aquellas que tienen objetos dentro de hoyos y así sucesivamente.

Recientemente se ha encontrado un método para comprimir cadenas de caracteres agrupándolas por combinaciones de los símbolos de su alfabeto, es una idea prometedora que habrá que analizar y usarla para el código 3OT.

Se continuará analizando el método de detección de puntos dominantes con el 3OT, para tratar de identificar que patrones son los que permiten la mejor detección de dichos puntos para los objetos binarios con características diversas. También se buscarán patrones especializados para objetos binarios con características específicas.

Se tiene la creencia de que la detección de puntos dominantes puede aplicarse con éxito para el Reconocimiento Óptico de Caracteres (OCR), se comenzará la investigación al respecto.

Como otro trabajo futuro está el hacer más eficiente en cuanto a tiempo de ejecución al software generador de código 3OT y a la extensión del método para la detección de puntos dominantes; se buscará utilizar estructuras de datos más eficientes, hacerle refinamientos a los algoritmos y se considera el implementarlos en otra tecnología. Mejorar el rendimiento en cuanto a tiempo de ejecución de estas herramientas puede significar su uso en un producto comercial, no sólo como herramientas de apoyo en la investigación.

APÉNDICE A. GLOSARIO DE TÉRMINOS

- Algoritmo: En matemáticas y ciencias de la computación, es una lista bien definida, ordenada y finita de operaciones que permite hallar la solución a un problema.
- Análisis de imágenes: Se encamina a determinar ciertas estructuras elementales tales como bordes o regiones, así como las relaciones entre ellas.
- ASCII: American Standard Code for Information Interchange (Código Americano Estándar para el Intercambio de Información). Es un código de caracteres de longitud fija basado en el alfabeto latino tal como se usa en el inglés moderno y otras lenguas occidentales.
- Bit: Binary digit (digito binario). Es la unidad mínima de información. Permite representar 2 posibles valores : {apagado, encendido}
- Byte: mordisco. Compuesto por 8 bits, es la unidad básica de información. Permite representar 256 valores: {0, 1, 2, … , 255}
- CCITT: Consultative Committee for International Telegraphy and Telephony (Comité Consultivo Internacional Telegráfico y Telefónico). Organización que establece estándares internacionales sobre telecomunicaciones. Actualmente es conocido como ITU-T.
- Código 3OT, código de tres cambios ortogonales, desarrollado por Sánchez-Cruz y Rodríguez-Dagnino.
- Código DFCCE, Directional Freeman Chain Code Eight, código direccional de ocho direcciones de Freeman.
- Código DFCCF, Directional Freeman Chain Code Four, código direccional de cuatro direcciones de Freeman.
- Código FCCE, Freeman Chain Code Eight, código de ocho direcciones de Freeman.
- Código FCCF, Freeman Chain Code Four, código de cuatro direcciones de Freeman.
- Facsímil: es una copia o reproducción muy precisa, casi perfecta, de un documento generalmente antiguo y de gran valor, como un libro, un manuscrito, un mapa o un dibujo a mano alzada.

- Fax: es una tecnología de las telecomunicaciones usada para transferir copias(facsímiles) de documentos, usando dispositivos económicos que operan sobre las redes telefónicas.
- Gestalt: Es una corriente de la psicología moderna, postula que la mente configura, a través de ciertas leyes, los elementos que llegan a ella a través de los canales sensoriales (percepción) o de la memoria (pensamiento, inteligencia y resolución de problemas). En nuestra experiencia del medio ambiente, esta configuración tiene un carácter primario por sobre los elementos que la conforman, y la suma de estos últimos por sí solos no podría llevarnos, por tanto, a la comprensión del funcionamiento mental. Este planteamiento se ilustra con el axioma *el todo es más que la suma de sus partes*, con el cual se ha identificado con mayor frecuencia a esta escuela psicológica.
- IBM: International Business Machines. Es una empresa que fabrica y comercializa herramientas, programas y servicios relacionados con la informática.
- IEC: International Electrotechnical Comission (Comisión Electrotécnica Internacional). Es una organización de normalización de los campos electrónico y tecnologías relacionadas.
- IPAN: Image and Pattern Analysis Group,
- ISO: International Organization for Standarization (Organización para la Estandarización Internacional). Nace después de la Segunda Guerra Mundial, es el organismo encargado de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación para todas las ramas industriales a excepción de la eléctrica y electrónica.
- ITU-T: Telecomunication Standarization Sector (Sector para la Estandarización de las Telecomunicaciones). Organización que establece estándares internacionales sobre telecomunicaciones. Antes conocida como CCITT.
- JBIG: Joint Bi-level Image Experts Group. Es un grupo de expertos auspiciado por los cuerpos de estandarización internacionales ISO e ITU encargado de desarrollar estándares para el almacenamiento de imágenes binivel, han desarrollado los formatos JBIG1 y JBIG2, que son los formatos estándar a nivel mundial hoy en día.

- Pixel: Picture Element (Elemento de pintura). Es la unidad básica de color programable en un monitor o imagen de computadora. Es una unidad lógica, no física.
- Procesamiento de imágenes: Implica la manipulación de las imágenes vistas como señales digitales, para extraer la información más elemental subyacente.
- TIFF: Tagged Image File Format (formato de archivo de imágenes con etiqueta) Es un formato de almacenamiento de imágenes con la característica de que los archivos, además de los datos de la imagen propiamente dicha, contienen "etiquetas" en las que se guarda información sobre las características de la imagen, que sirve para su tratamiento posterior.

APÉNDICE B. TIPOS DE DATOS ESTÁNDAR USADOS

Se han utilizado diversos tipos de datos estándar en el desarrollo del trabajo:

Definición 0.1 Tipo de dato, byte.

Puede tomar valores enteros en el rango del 0 al 255. Utilizado para representar valores enteros chicos como el estado del bit o las posibles direcciones que toman los vectores. Ocupa un byte en memoria.

Definición 0.2 Tipo de dato, int.

Puede tomar valores enteros en el rango del -2,147,483,648 al 2,147,483,467. Utilizado para describir datos que necesitan tomar valores enteros grandes como contadores y para representar coordenadas. Ocupa 4 bytes en memoria.

Definición 0.3 Tipo de dato, bool.

Puede tomar 2 valores: true o false. Se utiliza para preguntar estados o verificar si se cumplen condiciones. Ocupa un byte en memoria.

Definición 0.4 Tipo de dato, char.

Puede tomar 256 valores que son los símbolos del código ASCII. Usado para representar símbolos, se utiliza para almacenar los símbolos del alfabeto Ω. Ocupa un byte en memoria.

Definición 0.5 Tipo de dato, string.

Puede almacenar cadenas de símbolos. Se utiliza para almacenar cadenas del alfabeto Ω que representan fronteras en código 3OT. Ocupa un byte multiplicado por la longitud de la cadena almacenada.

Definición 0.6 Definición 3.10. Tipo de dato, Bitmap.

Puede almacenar imágenes. Mapa de bits en español, se utiliza para representar una malla $\mathbb{G}$ en la computadora, este objeto contiene la imagen con los objetos binarios que se codificarán al 3OT, también sirve para almacenar las imágenes generadas al decodificar cadenas del alfabeto Ω. Ocupa $m \times n$ bits multiplicados por la cantidad de bits necesarios para representar G_{max} en binario.

APÉNDICE C. ESTRUCTURAS DE DATOS USADAS.

Para el desarrollo del trabajo se utilizaron 2 estructuras de datos clásicas, que son las colas y listas, las cuáles pueden almacenar diferentes tipos de datos manteniendo una misma funcionalidad para su manejo, a continuación se presentan sus respectivas definiciones.

Definición 0.1 Estructura de dato, Cola.

Es una estructura lineal de datos. Una cola es un grupo ordenado de elementos homogéneos en el que los nuevos elementos se añaden por un extremo (el final) y se quitan por el otro extremo (el frente). En las colas el elemento que entró primero sale también primero, por ello se las llama como listas FIFO (first – in, first – out) "primero en entrar, primero en salir". Por eso, se usan para almacenar datos que necesitan ser procesados según el orden de llegada.

C= C (1), C(2), ..., C(N)

Las eliminaciones se realizan al principio de la lista frente (front), y las inserciones se realizan en el otro extremo final (rear).

La implementación de las colas se representará con la palabra Queue y el tipo de dato que almacenará se definirá inmediatamente después entre símbolos de menor que (<) y mayor que (>). Por ejemplo, una lista de int se define así: *Queue<int>*.

Definición 0.2 Estructura de dato, Lista.

Una lista es una de las estructuras de datos fundamentales, y puede ser usada para implementar otras estructuras de datos. Consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y una o dos referencias (punteros) al nodo anterior y/o posterior. El principal beneficio de las listas enlazadas respecto a los arreglos convencionales es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento en la memoria o el disco, permitiendo que el orden de recorrido de la lista sea diferente al de almacenamiento.

La implementación de las listas se representará con la palabra List y el tipo de dato que almacenará se definirá inmediatamente después entre símbolos de menor que (<) y mayor que (>). Por ejemplo, una lista de int se define así: *List<int>*.

APÉNDICE D. TIPOS DE DATOS DISEÑADOS PARA LA CODIFICACIÓN Y DECODIFICACIÓN DEL 3OT.

Para la programación de los algoritmos que permiten hacer la Codificación y Decodificación del 3OT se han creado varias clases de apoyo, que se describen a continuación:

Definición 0.1 Clase, EstadoDelBit

Clase estática. Aquí están las constantes para manejar los estados que puede tomar un bit o pixel. La Figura 0.1 muestra el diagrama de clase para EstadoDelBit.

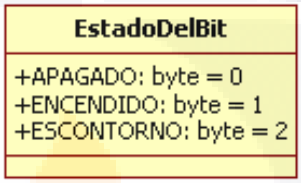


Figura 0.1 Diagrama de clase, EstadoDelBit

- APAGADO: byte = 0. Constante. Se usará para representar el estado de pixel apagado (0) o en blanco.
- ENCENDIDO: byte = 1. Constante. Representa el estado de pixel encendido (1) o que es parte de un objeto binario.
- ESCONTORNO: byte = 2. Constante. Representa el estado que tomarán el conjunto de pixeles que conforman alguna frontera de un objeto binario, se navegará sobre los pixeles del mapa de bits que tengan este valor al hacer la codificación o decodificación del 3OT.

Definición 0.2 Clase, VectorDirección.

Clase estática. Se definen constantes con los valores que van a poder tomar los vectores de dirección. También se definen algunas funciones sencillas para su manipulación. La Figura 0.2 muestra el diagrama de clase para VectorDirección.

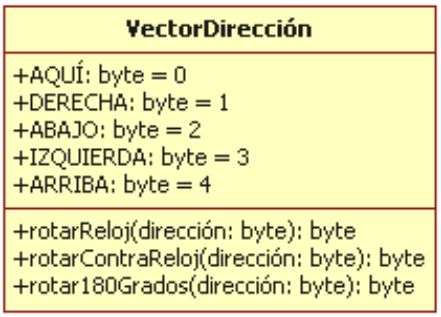


Figura 0.2 Diagrama de clase, VectorDirección.

- AQUÍ: byte = 0. Constante. Auxiliar y se utiliza cuando no se desea utilizar ninguna de las direcciones para los vectores.
- DERECHA: byte = 1. Constante. Representa un vector $\hat{\imath}$.
- ABAJO: byte = 2. Constante. Representa un vector $\hat{\jmath}$.
- IZQUIERDA: byte = 3. Constante. Representa un vector $-\hat{\imath}$.
- ARRIBA: byte = 4. Constante. Representa un vector $-\hat{\jmath}$.
- rotarReloj(dirección: byte): byte: Regresa el valor que representa al vector enviado como parámetro restándole 90º.
- rotarContraReloj(dirección: byte): byte. Regresa el valor que representa al vector enviado como parámetro sumándole 90º.
- rotar180Grados(dirección: byte): byte. Regresa el valor que representa al vector enviado sumándole 180º.

Definición 0.3 Clase, LineaARellenar.

Almacena datos de coordenadas necesarios para hacer el rellenado de fronteras con cola de líneas, dicho procedimiento se ve en el Pseudocódigo 0.1. Representa una línea horizontal de pixeles. La Figura 0.3 muestra el diagrama de clase para LineaARellenar.

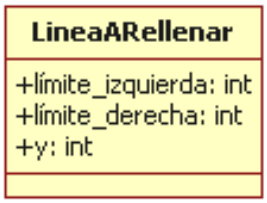


Figura 0.3 Diagrama de clase, LineaARellenar.

- límite_izquierda: int. Límite izquierdo de la línea horizontal.
- límite_derecha: int. Límite derecho de la línea horizontal.
- y: int. Posición en y o altura de la línea horizontal

Definición 0.4 Clase, ExploradorDeBitmaps.

Esta clase es muy importante, realiza el trabajo necesario para navegar el mapa de bits que contiene las formas binarias que se están codificando. También sirve en la decodificación, ya que también permite establecer los estados del bit sobre el que se encuentra en determinado momento. Puede verse como un punto abstracto que se mueve sobre los pixeles del mapa de bits y obtiene o modifica la información contenida en estos. La Figura 0.4 muestra una versión simplificada del diagrama de clase de ExploradorDeBitmaps.

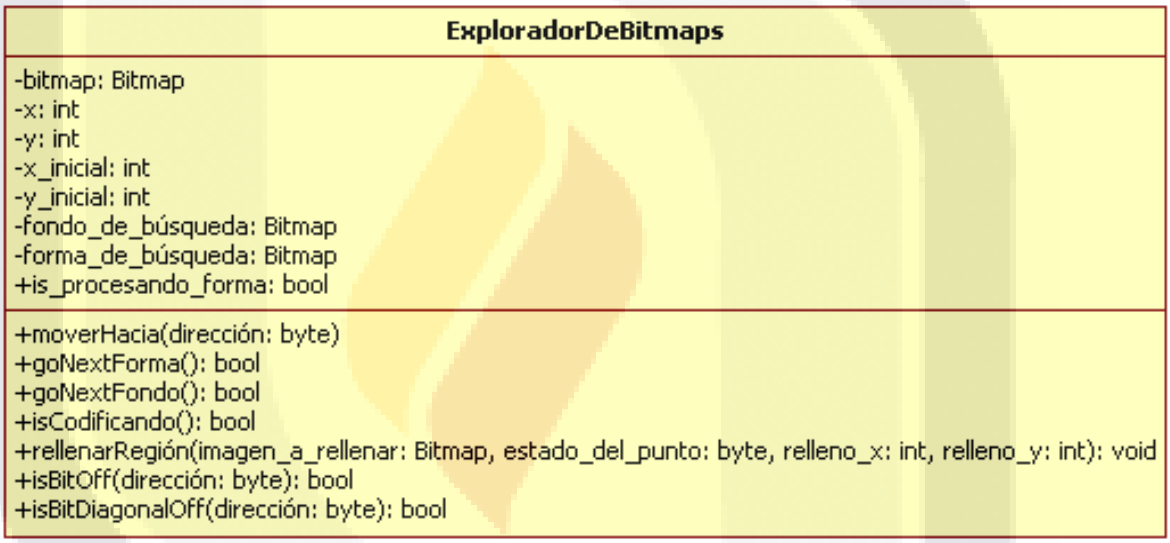


Figura 0.4 Diagrama de clase, ExploradorDeBitmaps.

- imagen_binaria: Bitmap. Es el mapa de bits que se está codificando o generando en la decodificación.
- x: int. Posición en el eje x en que se encuentra el ExploradorDeBitmaps.
- y: int. Posición en el eje y en que se encuentra el ExploradorDeBitmaps.
- x_inicial: int. Posición en el eje x en que se comienza el recorrido de una frontera el ExploradorDeBitmaps.

- y_inicial: int. Posición en el eje y en que se comienza el recorrido de una frontera el ExploradorDeBitmpas.
- fondo_de_búsqueda: Bitmap. Es un mapa de bits auxiliar para la localización de los diferentes objetos binarios en la imagen. Mismas dimensiones que imagen_binaria.
- forma_de_búsqueda: Bitmap. Es un mapa de bits auxiliar para la localización de los diferentes hoyos en los diferentes objetos binarios en la imagen. Mismas dimensiones que imagen_binaria.
- is_procesando_forma: bool. Debido a que la codificación de la frontera externa de los objetos binarios se hace en sentido de las manecillas del reloj y las fronteras internas en sentido contrario, es necesario registrar el tipo de frontera que se está procesando, cuando se establece en true significa frontera externa, cuando es false significa frontera de hoyo.
- moverHacia(dirección: byte): bool. Esta función mueve una unidad el ExploradorDeBitmaps en la dirección especificada, modifica los valores x, y.
- goNextForma(): bool. Localiza el siguiente objeto binario en el mapa de bits y establece los valores x, y, x_inicial, y_inicial que corresponden a la esquina superior izquierda de la forma binaria encontrada, regresa true si encontró algún objeto binario, false de lo contrario
- goNextFondo(): bool. Localiza el siguiente hoyo en el objeto binario que está siendo procesado y establece los valores x, y, x_inicial, y_inicial que corresponden a la esquina superior izquierda del hoyo encontrado, regresa true si encontró algún hoyo en el objeto binario, false de lo contrario.
- isCodificando(): bool. Regresa un valor false mientras no se haya terminado de codificar el objeto binario, esto lo hace revisando que los valores de x, y no hayan tomado los valores x_inicial, y_inicial de nueva cuenta.
- rellenarRegion(imagen_a_rellenar: Bitmap, estado_del_punto: byte, relleno_x: int, relleno_y: int): void. En imagen_a_rellenar, establece al estado_del_punto todos los pixeles que se conecten con el pixel de las coordenadas (relleno_x, relleno_y) incluyendo dicho pixel.

- isBitOff(dirección: byte): bool. Revisa si el pixel contiguo al ExploradorDeBitmaps en la dirección especificada está apagado (0), regresa true si es así. Ejemplo: En la Figura 3.4 suponemos que el ExploradorDeBitmaps se encuentra en la posición con el pixel marcado con rojo, los pixeles marcados con verde son los que se revisarán.

  idBitOff(VectorDirección.DERECHA) *regresa false*
  idBitOff(VectorDirección.ABAJO) *regresa false*
  idBitOff(VectorDirección.IZQUIERDA) *regresa false*
  idBitOff(VectorDirección.ARRIBA) *regresa true*

- isBitDiagonalOff(dirección: byte): bool Esta función es similar a isBitOff, pero en lugar de buscar el pixel contiguo revisa el pixel en diagonal. Ejemplo: En la Figura 3.5 suponemos que el ExploradorDeBitmaps se encuentra en la posición con el pixel marcado con rojo, los pixeles marcados con verde son los que se revisarán.

  idBitDiagonalOff(VectorDirección.DERECHA) *regresa false*
  idBitDiagonalOff(VectorDirección.ABAJO) *regresa false*
  idBitDiagonalOff(VectorDirección.IZQUIERDA) *regresa false;*
  idBitDiagonalOff(VectorDirección.ARRIBA) *regresa true;*

Definición 0.5 Clase, Codificador3OT.

Esta clase codifica los objetos binarios con sus hoyos que existan en un objeto del tipo Bitmap. La cadena de símbolos generada se almacena en un String. La Figura 0.5 muestra el diagrama de clase para Codificador3OT.

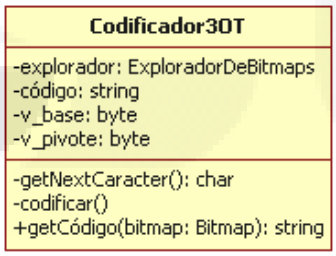


Figura 0.5 Diagrama de clase, Codificador3OT.

- explorador: ExploradorDeBitmaps. Este objeto será el encargado de navegar a través del mapa de bits que contiene los objetos binarios a codificar, identificará las fronteras y el estado de los pixeles.

- código: String. Aquí se almacenará la cadena en el alfabeto Ω que representa los objetos binarios con sus hoyos del mapa de bit que se está codificando al 3OT.
- v_base: byte. Registra la dirección que mantiene el vector de referencia.
- v_pivote: byte. Registra la dirección que mantiene el vector pivote.
- getNextCaracter(): char. Regresa el siguiente símbolo en la cadena que se está generando, este símbolo lo genera dependiendo de los estados de v_base, v_pivote y de lo que regresen las funciones isBitOff e isBitDiagonalOff del explorador.
- codificar(). Controla el orden en que se generan los códigos que representan las fronteras de los diferentes objetos y hoyos en la imagen.
- getCódigo(bitmap: Bitmap): string. Interfaz externa de la clase que recibe un Bitmap que contiene los objetos binarios a codificar, regresa un string que contiene la cadena de símbolos en Ω correspondiente.

Definición 0.6 Clase, Decodificador3OT.

Esta clase decodifica una cadena de símbolos en Ω contenida en un objeto de tipo string, generando un objeto de tipo Bitmap que contiene una representación gráfica de la malla 𝔾 con los objetos binarios. La Figura 0.6 muestra el diagrama de clase para Decodificador3OT.

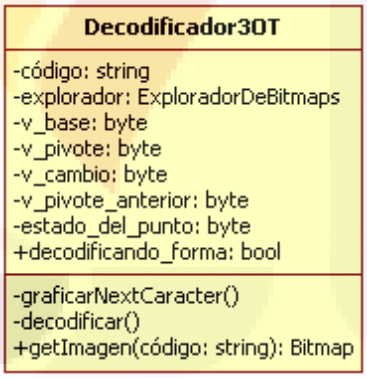


Figura 0.6 Diagrama de clase, Decodificador3OT.

- código: string. De este objeto se leerá el código de símbolos en Ω que se decoficará.
- explorador: ExploradorDeBitmaps. Sirve para navegar a través del mapa de bits que se generará y establecer los valores correspondientes de sus pixeles.

- v_base: byte. Registra la dirección que mantiene el vector de referencia.
- v_pivote: byte Registra la dirección que mantiene el vector pivote.
- v_cambio: byte Registra la dirección que mantiene el vector de cambio.
- v_pivote_anterior: byte. Registra la dirección previa que mantiene el vector pivote.
- estado_del_punto: byte. Registra el estado del punto que se está estableciendo para los pixeles que se grafican.
- decodificando_forma: bool. Variable auxiliar para registrar si se está codificando un objeto o un hoyo, si es un objeto se establece a true, si es un hoyo a false.
- graficarNextCaracter(). Mueve el explorador y establece el valor del pixel en el bitmap dependiendo de los valores que mantengan v_cambio, v_pivote_anterior y estado_del_punto.
- decodificar(). Lee la cadena 3OT símbolo por símbolo y asigna valores a v_base, v_pivote, v_pivote_anterior y v_cambio dependiendo del símbolo leído.
- getImagen(código: String): Bitmap. Interfaz externa de la clase que recibe un string que contiene la cadena de símbolos en Ω que será decodificada, regresa un Bitmap que representa la malla $\mathbb{G}$ con la Imagen que contiene lo decodificado.

APÉNDICE E. PSEUDOCÓDIGO Y PRUEBA DE ESCRITORIO DE CODIFICACIÓN 3OT.

Generación de símbolos del alfabeto Ω

Genera el siguiente símbolo del alfabeto Ω dependiendo de los vectores de referencia, pivote y de cambio generados al hacer el recorrido de una frontera.

Variables	Tipo	Descripción
explorador	ExploradorDeBitmap	Objeto encargado de navegar a través de los pixeles de la frontera a codificar en el mapa de bits contenedor de la imagen binaria. También permite identificar los cambio de dirección .
next_caracter	char	Almacena el símbolo generado que será devuelto por la función.
v_base	byte	Contiene el estado del vector de referencia, inicialmente tendrá el valor que haya sido asignado por el Pseudocódigo 0.3.
v_pivote	byte	Contiene el estado del vector pivote, inicialmente tendrá el valor que haya sido asignado por el Pseudocódigo 0.3.
v_auxiliar	byte	Variable auxiliar que almacenará el estado con que inicia v_base al entrar a la función. Esto con el fin de conocer el valor anterior.

Pseudocódigo

```
1.  v_auxiliar = v_base;
2.  if (explorador.isBitOff(v_pivote)) //Significa cambio de dirección simple
      2.1.    v_base = v_pivote;
      2.2.    explorador.moverHacia(v_pivote);
      2.3.    if (explorador.is_procesando_forma)
        2.3.1.      v_pivote = rotarReloj(v_pivote);
      2.4.    else
        2.4.1.      v_pivote rotarContraReloj(v_pivote);
      2.5.    if (v_auxilar == v_pivote)
        2.5.1.      next_caracter = '1';
      2.6.    else
        2.6.1.      next_caracter = '2';
      2.7.    return next_caracter;
3.  else if (explorador.isBitDiagonalOff(v_pivote)) //Significa cambio de dirección nulo
      3.1.    explorador.moverHacia(v_pivote);
      3.2.    next_caracter = '0';
      3.3.    return next_caracter;
4.  else  //Significa un cambio de dirección diagonal
      4.1.    v_base = v_pivote
      4.2.    explorador.moverHacia(v_pivote);
      4.3.    if (explorador.is_procesando_forma)
        4.3.1.      v_pivote = rotarContraReloj(v_pivote);
      4.4.    else
        4.4.1.      v_pivote rotarReloj(v_pivote);
      4.5.    if (v_auxilar == v_pivote)
        4.5.1.      next_caracter = '1';
      4.6.    else
        4.6.1.      next_caracter = '2';
      4.7.    return next_caracter;
```

Pseudocódigo 0.1 Generación de símbolos del alfabeto Ω.

En la Figura 0.1 se aprecian los pixeles sobre los que va navegando el objeto de la clase ExploradorDeBitmap al hacer 5 iteraciones del Pseudocódigo 0.1. En la Tabla 0.1 se muestra una prueba de escritorio de las mismas iteraciones.

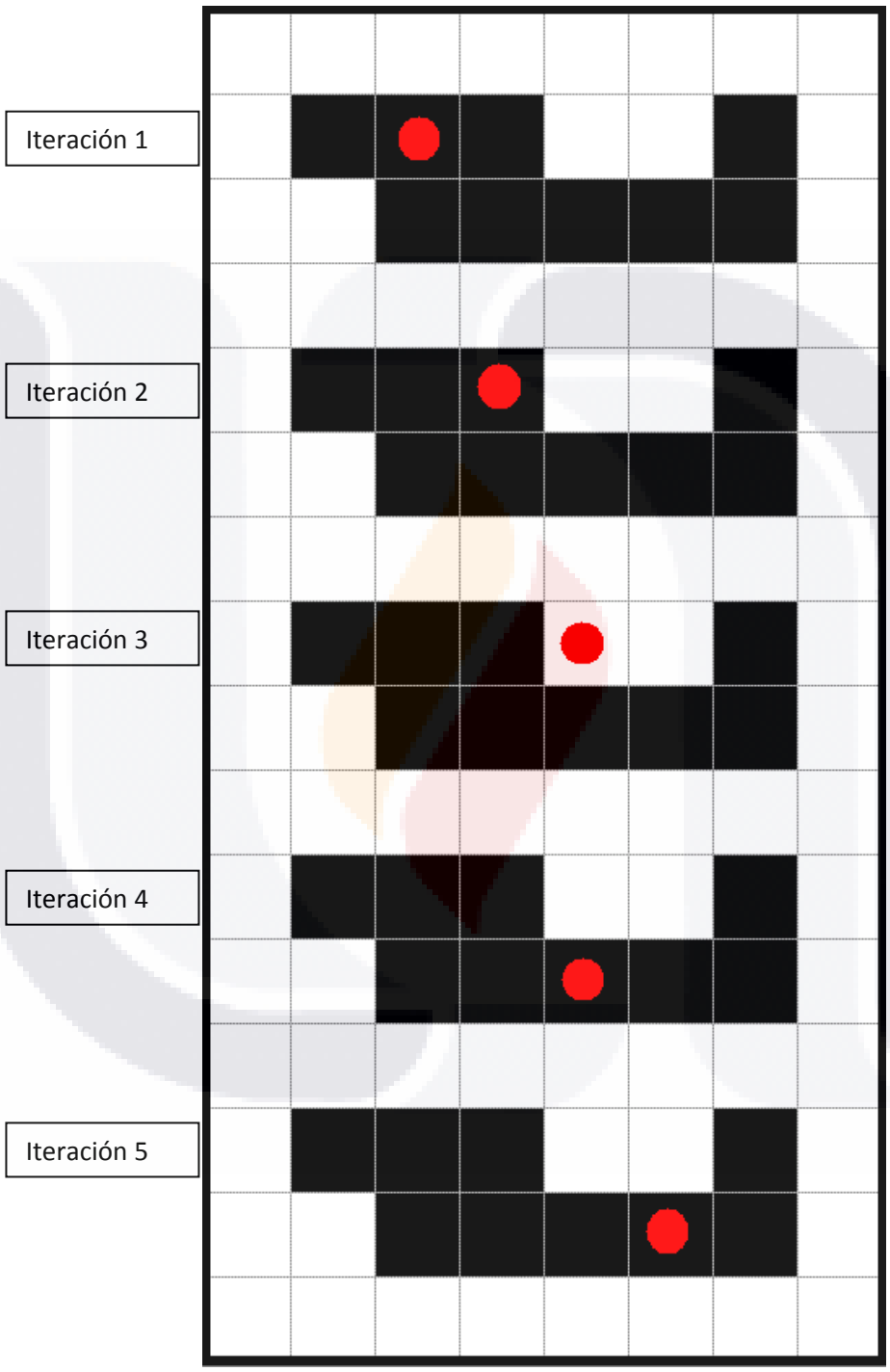


Figura 0.1 Imagen binaria, aleatoria, ejemplo generación de símbolos, 5 iteraciones.

Eventos sucedidos	Valores de las variables tomados en esta iteración
Posición inicial	v_base es ARRIBA v_pivote es DERECHA v_auxiliar es ARRIBA
1. Iteración 1: a. Revisa que se cumpla la condición 2, la cual no se cumple. b. Revisa que se cumpla la condición 3, la cual sí se cumple. i. Mueve el explorador hacia v_pivote(DERECHA) ii. Agrega un "0" al código iii. Va al paso 1, haciendo que v_auxiliar apunte hacia v_base(ARRIBA).	 next_caracter es '0' v_auxiliar es ARRIBA
2. Iteración 2: a. Revisa que se cumpla la condición 2, la cual sí se cumple. i. v_base apunta hacia v_pivote(DERECHA) ii. Mueve el explorador hacia v_pivote(DERECHA) iii. v_pivote apunta hacia v_pivote rotado reloj(ABAJO) iv. como v_auxiliar es diferente de v_pivote entonces agrega un "2" al código v. Va al paso 1, haciendo que v_auxiliar apunte hacia v_base(DERECHA)	 v_base es DERECHA v_pivote es ABAJO next_caracter es '2' v_auxiliar es DERECHA
3. Iteración 3 a. Revisa que se cumpla la condición 2, la cual no se cumple. b. Revisa que se cumpla la condición 3, la cual no se cumple. c. Entra a 4 i. v_base apunta hacia v_pivote(ABAJO) ii. Mueve el explorador hacia v_pivote(ABAJO) iii. v_pivote apunta hacia rotado contra reloj (DERECHA) iv. como v_auxiliar es igual que v_pivote entonces agrega un "1" al código v. Va al paso 1, haciendo que v_auxiliar apunte hacia v_base(DERECHA)	 v_base es ABAJO v_pivote es DERECHA next_caracter es '1' v_auxiliar = ABAJO
4. Iteración 4: a. Revisa que se cumpla la condición 2, la cual no se cumple. b. Revisa que se cumpla la condición 3, la cual sí se cumple. i. Mueve el explorador hacia v_pivote(DERECHA) ii. Agrega un "0" al código iii. Va al paso 1, haciendo que v_auxiliar apunte hacia v_base(ABAJO).	 next_caracter es '0' v_auxiliar es ABAJO
5. Iteración 5: a. Revisa que se cumpla la condición 2, la cual no se cumple. b. Revisa que se cumpla la condición 3, la cual no se cumple. c. Entra a 4 i. v_base apunta hacia v_pivote(DERECHA) ii. Mueve el explorador hacia v_pivote(DERECHA) iii. v_pivote apunta hacia rotado contra reloj (ARRIBA) iv. como v_auxiliar es diferentel que v_pivote entonces agrega un "2" al código v. Va al paso 1, haciendo que v_auxiliar apunte hacia v_base(DERECHA)	 v_base es DERECHA v_pivote es ARRIBA next_caracter es '2' v_auxiliar es DERECHA

Tabla 0.1 Generación de símbolos del alfabeto Ω, ejemplo, 5 iteraciones.

APÉNDICE F. PSEUDOCÓDIGO DEL RELLENADO DE FRONTERAS CON COLA DE LÍNEAS.

En el Pseudocódigo 0.1 se describe el algoritmo implementado en la función rellenarRegion(imagen_a_rellenar: Bitmap, estado_del_punto: byte, relleno_x: int, relleno_y: int): void de la clase ExploradorDeBitmap para tal propósito.

Rellenado de fronteras con cola de líneas.

Establece a un color todos los pixeles que estén delimitados por una frontera en una imagen binaria. Implementado en la función rellenarRegion(imagen_a_rellenar: Bitmap, estado_del_punto: byte, relleno_x: int, relleno_y: int): void de la clase ExploradorDeBitmap. En imagen_a_rellenar, establece al estado_del_punto todos los pixeles que se conecten con el pixel de las coordenadas (relleno_x, relleno_y) incluyendo dicho pixel.

Variables	Tipo	Descripción
línea_auxiliar	LineaARellenar	Auxiliar para crear nuevos objetos de este tipo que serán procesados.
líneas_a_rellenar	Queue<LineaARellenar>	Cola de líneas de pixeles a rellenar.
imagen_a_rellenar	Bitmap	Imagen donde se hará el rellenado.
estado_del_punto	byte	Estado del punto al que se rellenará.
estado_inicial	byte	Estado del punto inicial.
relleno_x	int	Posición en x donde se comenzará el rellenado.
relleno_y	int	Posición en y donde se comenzará el rellenado.

Pseudocódigo

1. Se almacena en estado_inicial el estado que tiene el pixel con las coordenadas (relleno_x, relleno_y).
2. Se crea una línea_auxiliar, de la siguiente manera:
   - 2.1. A partir del pixel inicial(relleno_x, relleno_y) se escaneará imagen_a_rellenar pixel por pixel hacia la izquierda hasta encontrar el primer pixel con estado diferente a estado_inicial o bien el límite izquierdo de la imagen. Al hacer esta búsqueda se irá modificando el estado de los pixeles escaneados al estado_del_punto. Se asignará el valor en x final al línea_auxiliar.límite_izquierdo.
   - 2.2. A partir del pixel inicial(relleno_x, relleno_y) se escaneará imagen_a_rellenar pixel por pixel hacia la derecha hasta encontrar el primer pixel con estado diferente a estado_inicial o bien el límite derecho de la imagen. Al hacer esta búsqueda se irá modificando el estado de los pixeles escaneados al estado_del_punto. Se asignará el valor en x final al línea_auxiliar.límite_derecho.
   - 2.3. Se asigna relleno_y a línea_auxiliar.y.
   - 2.4. Se introduce línea_auxiliar en líneas_a_rellenar.
3. Mientras líneas_a_rellenar no esté vacía.
   - 3.1. Saca una línea de la cola líneas_a_rellenar y asigna los datos a línea_auxiliar.
   - 3.2. Para cada uno de los pixeles en línea auxiliar hace lo siguiente.
     - 3.2.1. Si el pixel vecino norte tiene un estado igual a estado_inicial, hace lo siguiente.
       - 3.2.1.1. Crea una línea_auxiliar de la misma manera que en el paso 2 para el pixel vecino norte del pixel actual.
     - 3.2.2. Si el pixel vecino sur tiene un estado igual a estado_inicial, hace lo siguiente.
       - 3.2.2.1. Crea una línea_auxiliar de la misma manera que en el paso 2 para el pixel vecino sur del pixel actual.

Pseudocódigo 0.1 Rellenado de fronteras con cola de líneas.

APÉNDICE G. PSEUDOCÓDIGO DE BÚSQUEDA DE OBJETOS Y HOYOS EN IMÁGENES BINARIAS.

La búsqueda de objetos y hoyos en imágenes binarias la ejecuta la clase ExploradorDeBitmap, específicamente en las funciones goNextForma() y goNextFondo(), el orden en que se ejecutan estas funciones es controlado por la función codificar() de la clase Codificador3OT. A continuación se presentan los pseudocódigos para dichas funciones.

Búsqueda de objetos binarios en mapa de bits

Recorre la imagen binaria en orden de línea mayor con el fin de localizar el siguiente objeto binario para codificar. Implementado en la función goNextForma(): bool de la clase ExploradorDeBitmap.

Variables	Tipo	Descripción
imagen_binaria	Bitmap	Aquí se hará la búsqueda de los objetos binarios. El estado de sus pixeles depende de la imagen binaria que representa.
fondo_de_búsqueda	Bitmap	Es un mapa de bits auxiliar para la localización de los diferentes objetos binarios en la imagen. Mismas dimensiones que imagen_binaria. Antes de entrar por vez primera a este código tiene todos sus pixeles en estado APAGADO.
x_inicial	int	Registrará la coordenada en x en que se localice el objeto binario encontrado.
y_inicial	int	Registrará la coordenada en y en que se localice el objeto binario encontrado.
is_procesando_forma	bool	Para registrar que se comenzará la codificación de una frontera externa si se encuentra un objeto binario.

Pseudocódigo

1. Recorre los pixeles de imagen_binaria y fondo_de_búsquesa en orden de línea mayor al mismo tiempo.
   1.1. if (encuentra pixel en imagen_binaria con valor ENCENDIDO y en fondo_de_búsqueda está APAGADO).//Significa que se localizó un objeto binario en imagen_binaria.
      1.1.1. x_inicial = coordenada en x del pixel encontrado;
      1.1.2. y_inicial = coordenada en y del pixel encontrado;
      1.1.3. is_procesando_forma = true;
      1.1.4. regresa true;
2. regresa false;
3. En caso de que se haya encontrado objeto binario al salir de esta función se procede con la codificación de la frontera externa de dicho objeto. Este procedimiento se ve en el Pseudocódigo 0.1.
4. Al conocer la frontera del objeto binario encontrado se procede a establecer en estado ENCENDIDO en fondo_de_búsqueda todos los pixeles que sus coordenadas coincidan con los pixeles delimitados por la frontera externa del objeto binario encontrado. Este procedimiento se ve en el Pseudocódigo 0.1.

Pseudocódigo 0.1 Búsqueda de objetos binarios en mapa de bits.

Búsqueda de hoyos en objetos binarios

Recorre la imagen binaria en orden de línea mayor con el fin de localizar el siguiente hoyo a codificar en el objeto binario actual. Implementado en la función goNextFondo(): bool de la clase ExploradorDeBitmap.

Variables	Tipo	Descripción
imagen_binaria	Bitmap	Aquí se hará la búsqueda de los hoyos. El estado de sus pixeles depende de la imagen binaria que representa.
forma_de_búsqueda	Bitmap	Es un mapa de bits auxiliar para la localización de los diferentes hoyos en el objeto binario actual. Mismas dimensiones que imagen_binaria. Al entrar a buscar los hoyos de un objeto binario por vez primera tiene todos los pixeles delimitados por la frontera externa del objeto binario en estado ENCENDIDO y el resto de los pixeles en APAGADO.
x_inicial	int	Registrará la coordenada en x en que se localice el hoyo encontrado.
y_inicial	int	Registrará la coordenada en y en que se localice el hoyo encontrado.
is_procesando_forma	bool	Para registrar que se comenzará la codificación de una frontera interna si se encuentra un hoyo.

Pseudocódigo

1. Recorre los pixeles de imagen_binaria y forma_de_búsquesa en orden de línea mayor al mismo tiempo.
    1.1. if (encuentra pixel que en imagen_binaria tiene valor APAGADO y en forma_de_búsqueda está ENCENDIDO).//Significa que se localizó un hoyo en el objeto binario actual.
        1.1.1. x_inicial = coordenada en x del pixel encontrado;
        1.1.2. y_inicial = coordenada en y del pixel encontrado;
        1.1.3. is_procesando_forma = false;
        1.1.4. regresa true;
2. regresa false;
3. En caso de que se haya encontrado hoyo en el objeto binario actual al salir de esta función se procede con la codificación de la frontera de dicho hoyo. Este procedimiento se ve en el Pseudocódigo 0.1.
4. Al conocer la frontera del hoyo encontrado se procede a establecer en estado APAGADO en forma_de_búsqueda todos los pixeles que sus coordenadas coincidan con los pixeles delimitados por la frontera del hoyo encontrado. Este procedimiento se ve en el Pseudocódigo 0.1

Pseudocódigo 0.2 Búsqueda de hoyos en objetos binarios.

Orden de búsqueda de objetos binarios y hoyos

Controla el orden en que se hará la búsqueda y codificación de los diferentes objetos binarios y hoyos dentro de la imagen binaria original. Implementado en la función codificar() de la clase Codificador3OT.

Variables	Tipo	Descripción
explorador	ExploradorDeBitmap	Objeto encargado de hacer la búsqueda de objetos binarios y hoyos.
código	String	Aquí se irán concatenando los sucesivos símbolos en el alfabeto Ω que constituyen la cadena de símbolos que representa las fronteras de los objetos binarios y sus hoyos en 3OT. Inicialmente está vacío.
v_base	byte	En esta variable se establecerá el valor del vector de referencia al iniciar la codificación de un objeto binario o de un hoyo.
v_pivote	byte	En esta variable se establecerá el valor del vector pivote al iniciar la codificación de un objeto binario o de un hoyo.

Pseudocódigo

1. Concatena en código las dimensiones del mapa de bits a codificar.
2. Mientras explorador.goNextForma() regrese true //Mientras se encuentren formas binarias en la imagen.
   - 2.1. v_base = ARRIBA;
   - 2.2. v_pivote = DERECHA;
   - 2.3. Concatena en código un carácter '+' y las coordenadas en x e y del explorador, es decir, las coordenadas donde se encontró la forma binaria.
   - 2.4. Mientras explorador.isCodificando() regrese true //Mientras no se haya terminado de codificar el objeto binario.
     - 2.4.1. Concatena en código el siguiente símbolo en el alfabeto Ω, el cuál es regresado por la función getNextCaracter() de la clase Codificador3OT. Este procedimiento se ve en el algoritmo 3.5.
   - 2.5. Mientras explorador.goNextFondo() regrese true //Mientras se encuentren hoyos en el objeto binario.
     - 2.5.1. v_base = IZQUIERDA;
     - 2.5.2. v_pivote = ABAJO;
     - 2.5.3. Concatena en código un carácter '-' y las coordenadas en x e y del explorador, es decir, las coordenadas donde se encontró el hoyo.
     - 2.5.4. Mientras explorador.isCodificando() regrese true //Mientras no se haya terminado de codificar el hoyo.
       - 2.5.4.1. Concatena en código el siguiente símbolo en el alfabeto Ω, el cuál es regresado por la función getNextCaracter() de la clase Codificador3OT. Este procedimiento se ve en el Pseudocódigo 0.1.

Pseudocódigo 0.3 Orden de búsqueda de objetos binarios y hoyos.

APÉNDICE H. PSEUDOCÓDIGO Y PRUEBA DE ESCRITORIO DE LA DECODIFICACIÓN 3OT.

El proceso de decodificación consta de dos algoritmos que son ejecutados sucesivamente para cada símbolo de la cadena que contiene el código 3OT. Primero un algoritmo que lee un símbolo de la cadena 3OT y asigna valores a los vectores de dirección dependiendo del símbolo leído (algoritmo 3.7), posteriormente se ejecuta un algoritmo que hace la graficación de las fronteras que son generadas dependiendo de los valores almacenados en los vectores de dirección (algoritmo 3.6). La lectura de símbolos y asignación de valores a los vectores de dirección se ha implementado en la función decodificar() de la clase Decodificador3OT, y la graficación de las fronteras de la imagen binaria en el mapa de bits se implementó en la función graficarNextCaracter() de la clase Decodificador3OT.

Graficación de las fronteras generadas al decodificar el 3OT

Establece los valores de los pixeles de las fronteras en la imagen generada al hacer la decodificación de una cadena de símbolos en 3OT. Este algoritmo está implementado en la función graficarNextCaracter() de la clase Decodificador3OT.

Variables	Tipo	Descripción
explorador	ExploradorDeBitmap	Objeto de la clase ExploradorDeBitmap, encargado de moverse a través del mapa de bits sobre el que se graficarán las fronteras generadas y de establecer sus valores correspondientes a los pixeles.
v_pivote_anterior	byte	Almacena el valor del vector pivote en el paso anterior. El estado inicial depende de la función que llama a este algoritmo.
v_cambio	byte	Almacena el valor del vector de cambio. El estado inicial depende de la función que llama a este algoritmo.

Pseudocódigo

1. if (v_cambio == v_pivote_anterior) //Grafica un cambio de dirección nulo
   1.1. explorador.moverHacia(v_cambio);
   1.2. establece a encendido el pixel actual del explorador.
2. else if (v_cambio = rotarReloj(v_pivote_anterior) //Grafica un cambio de dirección simple
   2.1. establece a encendido el pixel actual del explorador.
3. else if (v_cambio = rotarContraReloj(v_pivote_anterior) //Grafica un cambio de dirección diagonal
   3.1. explorador.moverHacia(v_pivote_anterior);
   3.2. establece a encendido el pixel actual del explorador.
   3.3. explorador.moverHacia(v_cambio);
   3.4. establece a encendido el pixel actual del explorador.

Pseudocódigo 0.1 Graficación de las fronteras generadas al decodificar el 3OT.

Asignación de valores a los vectores de dirección

Lee uno por uno los símbolos de la cadena 3OT y va modificando los valores de los vectores de dirección dependiendo del símbolo actual. Este algoritmo está implementado en la función decodificar() de la clase Decodificador3OT.

Variables	Tipo	Descripción
código	String	Contiene la cadena de símbolos en el alfabeto Ω que será decodificada, además contiene información adicional que indica si una subcadena de símbolos a decodificar representa una frontera externa o un hoyo, las dimensiones del mapa de bits original y las coordenadas donde se graficarán los objetos y hoyos.
decodificando_forma	bool	Variable auxiliar para registrar si se está codificando un objeto o un hoyo, si es un objeto se establece a true, si es un hoyo a false.
v_base	byte	Almacena el valor del vector de referencia.
v_pivote	byte	Almacena el valor del vector pivote.
v_pivote_anterior	byte	Almacena el valor del vector pivote en el paso anterior.
v_cambio	byte	Almacena el valor del vector de cambio.

Pseudocódigo

4. Para cada símbolo en código hará lo siguiente:
    4.1. Switch (símbolo actual de código)
        4.1.1. Case '+': //Comenzará la codificación de una frontera externa de un objeto binario.
            4.1.1.1. if (decodificando_forma == true)
                4.1.1.1.1. Rellena la última frontera decodificada con pixeles encendidos (1s). Este procedimiento se ve en el Algoritmo 3.1.
            4.1.1.2. else
                4.1.1.2.1. Rellena la última frontera decodificada con pixeles apagados (0s). Este procedimiento se ve en el Algoritmo 3.1.
            4.1.1.3. v_base = ARRIBA;
            4.1.1.4. v_pivote = DERECHA;
            4.1.1.5. v_cambio = AQUÍ;
            4.1.1.6. decodificando_forma = true;
            4.1.1.7. break;
        4.1.2. Case '-'://Comenzará la codificación de una frontera de un hoyo.
            4.1.2.1. if (decodificando_forma == true)
                4.1.2.1.1. Rellena la última frontera decodificada con pixeles encendidos (1s). Este procedimiento se ve en el Algoritmo 3.1.
            4.1.2.2. else
                4.1.2.2.1. Rellena la última frontera decodificada con pixeles apagados (0s). Este procedimiento se ve en el Algoritmo 3.1.
            4.1.2.3. v_base = IZQUIERDA;
            4.1.2.4. v_pivote = ABAJO;
            4.1.2.5. v_cambio = AQUÍ;
            4.1.2.6. decodificando_forma = false;
            4.1.2.7. break;
        4.1.3. Case '(': //Paréntesis con coordenadas.
            4.1.3.1. if (son los primeros paréntesis encontrados en toda la cadena de símbolos)
                4.1.3.1.1. Crea el mapa de bits del objeto explorador con las dimensiones especificadas por las coordenadas de los paréntesis, establece todos sus pixeles en apagado.
            4.1.3.2. else
                4.1.3.2.1. Establece las coordenadas en x e y del explorador a las especificadas por los paréntesis.

```
                4.1.3.3.    break;
        4.1.4.      Case '0':
                4.1.4.1.    v_cambio = v_pivote; // S_ref+1 = S_ref+z+2
                4.1.4.2.    v_pivote _anterior = v_pivote;
                4.1.4.3.    break;
        4.1.5.      Case '1':
                4.1.5.1.    v_cambio = v_base; // S_ref = S_ref+z+2
                4.1.5.2.    v_base = v_pivote; // S_ref • S_ref+1 = 0
                4.1.5.3.    v_pivote_anterior = v_pivote;
                4.1.5.4.    v_pivote = v_cambio;
                4.1.5.5.    break;
        4.1.6.      Case '2':
                4.1.6.1.    v_cambio = rotar180Grados(v_base); // S_ref = -S_ref+z+2
                4.1.6.2.    v_base = v_pivote; // S_ref • S_ref+1 = 0
                4.1.6.3.    v_pivote_anterior = v_pivote;
                4.1.6.4.    v_pivote = v_cambio;
                4.1.6.5.    break;
    4.2.    graficarNextCaracter() //Manda a graficar, Este procedimiento se ve en el algoritmo 3.6
```

Pseudocódigo 0.2 Asignación de valores a los vectores de dirección.

En la Figura 0.1 se aprecian los pixeles que se van generando al hacer 5 iteraciones del Pseudocódigo 0.2 En la Tabla 0.1 se muestra una prueba de escritorio de las mismas iteraciones.

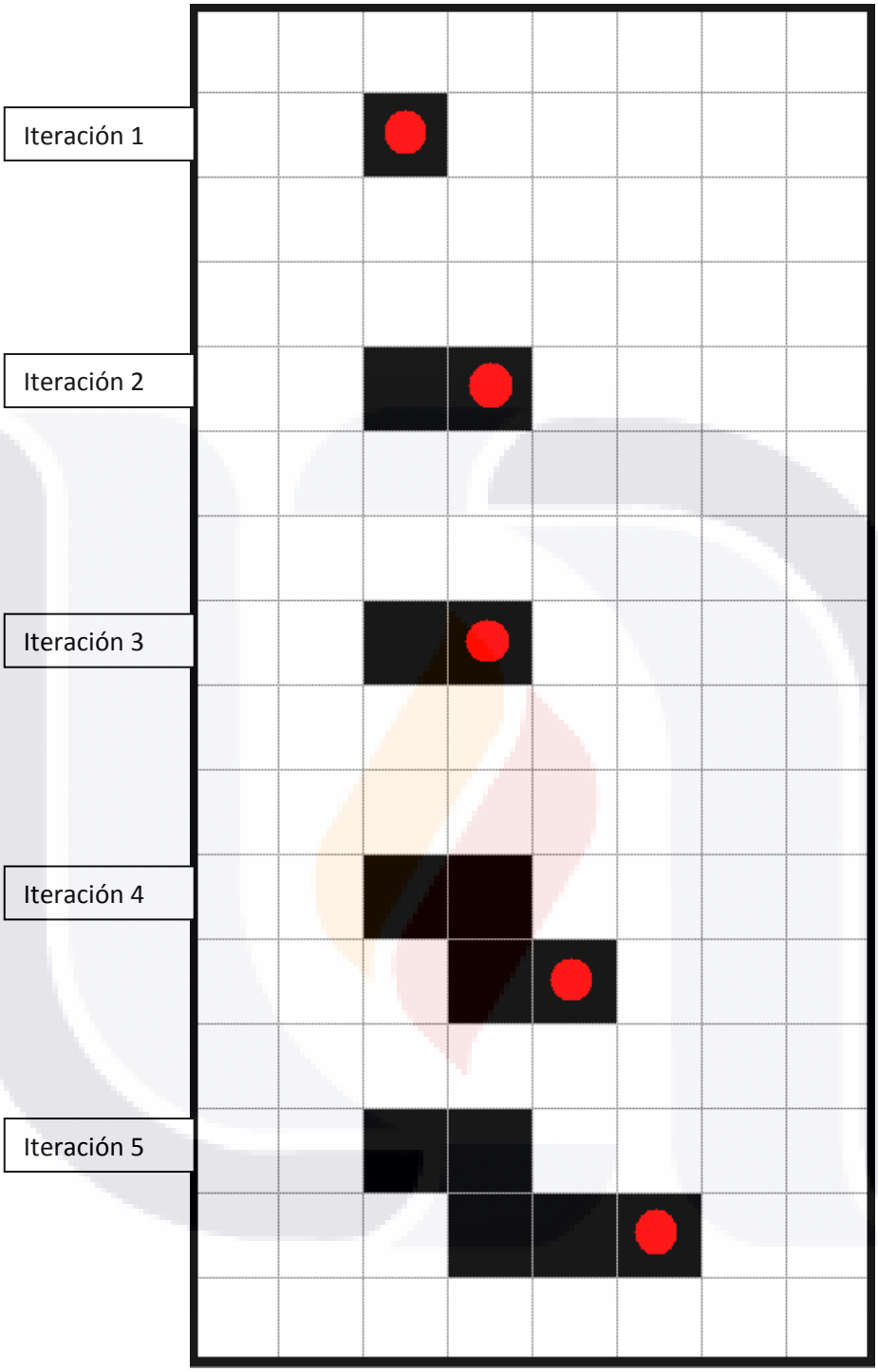


Figura 0.1 Imagen binaria, aleatoria, ejemplo graficación de pixeles, 5 iteraciones.

Eventos sucedidos	Valores de las variables tomados en esta iteración
Posición inicial, se ha encontrado un símbolo '+' y se han establecido las coordenadas en el mapa de bits del explorador que se obtienen de los número entre paréntesis. Se procede con la codificación de la frontera…	código = 002102120200002112 v_base es ARRIBA v_pivote es DERECHA v_pivote_anterior es AQUÍ v_cambio es AQUI
1. Iteración 1: 1.1. Revisa el símbolo actual que es '0', entra a 4.1.4 1.1.1. v_cambio apunta hacia v_pivote (DERECHA) 1.1.2. v_pivote_anterior apunta hacia v_pivote (DERECHA) 1.1.3. llama a la función graficarNextCaracter() y como v_cambio es igual a v_pivote_anterior, resulta en // cambio de dirección nulo 1.1.3.1. mueve el explorador hacia v_cambio(DERECHA) 1.1.3.2. establece a encendido el pixel actual del explorador.	v_cambio es DERECHA v_pivote_anterior es DERECHA
2. Iteración 2: 2.1. Revisa el símbolo actual que es '0', entra a 4.1.4 2.1.1. v_cambio apunta hacia v_pivote (DERECHA) 2.1.2. v_pivote_anterior apunta hacia v_pivote (DERECHA) 2.1.3. llama a la función graficarNextCaracter()y como v_cambio es igual a v_pivote_anterior, resulta en //cambio de dirección nulo 2.1.3.1. mueve el explorador hacia v_cambio(DERECHA) 2.1.3.2. establece a encendido el pixel actual del explorador.	v_cambio es DERECHA v_pivote_anterior es DERECHA
3. Iteración 3 3.1. Revisa el símbolo actual que es '2', entra a 4.1.6 3.1.1. v_cambio apunta hacia v_base rotado 180° (ABAJO) 3.1.2. v_base apunta hacia v_pivote (DERECHA) 3.1.3. v_pivote_anterior apunta hacia v_pivote(DERECHA) 3.1.4. v_pivote apunta hacia v_cambio (ABAJO) 3.1.5. llama a la función graficarNextCaracter()y como v_cambio es igual a v_pivote_anterior rotado 90° hacia la derecha, resulta en //cambio de dirección simple 3.1.5.1. establece a encendido el pixel actual del explorador.	v_cambio es ABAJO v_base es DERECHA v_pivote_anterior es DERECHA v_pivote es ABAJO
4. Iteración 4: 4.1. Revisa el símbolo actual que es '1', entra a 4.1.5 4.1.1. v_cambio apunta hacia v_base (DERECHA) 4.1.2. v_base apunta hacia v_pivote (ABAJO) 4.1.3. v_pivote_anterior apunta hacia v_pivote(ABAJO) 4.1.4. v_pivote apunta hacia v_cambio (DERECHA) 4.1.5. llama a la función graficarNextCaracter()y como v_cambio es igual a v_pivote_anterior rotado 90° hacia la izquierda, resulta en //cambio de dirección diagonal 4.1.5.1. mueve el explorador hacia v_pivote_anterior(ABAJO) 4.1.5.2. establece a encendido el pixel actual del explorador. 4.1.5.3. mueve el explorador hacia v_cambio(DERECHA) 4.1.5.4. establece a encendido el pixel actual del explorador.	v_cambio es DERECHA v_base es ABAJO v_pivote_anterior es ABAJO v_pivote es DERECHA

5. Iteración 5: 5.1. Revisa el símbolo actual que es '0', entra a 4.1.4 5.1.1. v_cambio apunta hacia v_pivote (DERECHA) 5.1.2. v_pivote_anterior apunta hacia v_pivote (DERECHA) 5.1.3. llama a la función graficarNextCaracter() y como v_cambio es igual a v_pivote_anterior, resulta en // cambio de dirección nulo 5.1.3.1. mueve el explorador hacia v_cambio(DERECHA) 5.1.3.2. establece a encendido el pixel actual del explorador	v_cambio es DERECHA v_pivote_anterior es DERECHA

Tabla 0.1 Graficación de pixels a partir de los símbolos del alfabeto Ω, ejemplo, 5 iteraciones.

BIBLIOGRAFÍA

Abel, J. (2004 - 2009). *The Data Compression Resource on the Internet*. Obtenido de www.data-compression.info

Alcaraz-Corona, S., Neri-Calderón, R. A., & Rodríguez-Dagnino, R. M. (2009). Efficient Bilevel Image Compression by Grouping Symbols of Chain Coding Techniques. *Optical Engineering Volume 48, Issue 3* , 037001-1 - 037001-14.

Attneave, F. (1954). Some Informational Aspects of Visual Perception. *Psychological Review Volume 61, Issue 3* , 183-193.

Beus, H., & Tiu, S. (1987). An Improved corner detection algorithm based on chain coded plane curves. *Pattern Recognition Volume 20, Issue 3* , 291 - 296.

Beyer, M., Hahn, R., Peschel, S., Harz, M., & Knoche, M. (2002). Analysing fruit shape in sweet cherry (Prunus avium L.). *Scientia Horticulturae Volume 96, Issues 1-4* , 139 - 150.

Brassard, G., & Bratley, P. (1996). *Fundamentals of Algorithmics.* Prentice-Hall.

Bribiesca, E. (2000). A chain code for representing 3D curves. *Pattern Recognition Volume 33, Issue 5* , 755 - 765.

Bribiesca, E. (1999). A new chain code. *Pattern Recognition Volume 32, Issue 2* , 235 - 251.

Cheng, H., & Chen, Y. (1999). Fuzzy partition of two-dimensional histogram and its application to thresholding. *Pattern Recognition Volume 32, Issue 5* , 824 - 843.

Data-Compression.com A website devotede to the principles and practice of data compression. (2000 - 2007). Obtenido de http://www.data-compression.com

Echávarri-Aguinaga, L., Neri-Calderón, R., & Rodríguez-Dagnino, R. (2007). Compression rates comparison of entropy coding for three-bit chain codes of bilevel images. *Optical Engineering Volume 46, Issue 8* , 087007 - 1 - 087007 - 7.

Freeman, H. (1974). Computer processing of line drawing images. *ACM Computing Surveys Volume 6, Issue 1* , 57 - 97.

Freeman, H. (1961). On the encoding of arbitrary geometric configurations. *IRE Trans. Electron. Comput. EC Volume 10* , 260 - 268.

Freeman, H., & Davis, L. (1976). A Corner-Finding Algorithm for Chain-Coded Curves. *IEEE Transactions on Computers Volume C-26, Issue 3* , 297 - 303.

Fu, K., González, R., & Lee, C. (1988). *Robótica: Control, Detección, Visión e Inteligencia.* Madrid: McGraw-Hill.

García, P., Pérez, T., Ruiz, J., Segarra, E., Sempere, J., & Vázquez de Parga, M. (1996). *Apuntes sobre la teoría de autómatas y lenguajes formales.* Valencia: Universidad Politécnica de Valencia.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

Hopcroft, J., Montwani, R., & Ullman, J. (2002). *Introducción a la Teoría de Autómatas, Lenguajes y Computación.* Madrid: Addison Wesley.

Knoll, A. (2000). *Compression of bi-level images, compressor performance report.* Praga: Librería Nacional de la República Checa.

Kui-Liu, Y., & Zalik, B. (2005). An efficient chain code with Huffman coding. *Pattern Recognition Volume 38, Issue 4* , 553 - 557.

Marji, ,. M., & Siy, P. (2003). A new algorithm for dominant points detection and polygonization of digital curves. *Pattern Recognition Volume 36, Issue 10* , 2239 - 2251.

Masood, A. (2008b). Dominant point detection by reverse polygonization of digital curves. *Image and Vision Computing Volume 26, Issue 5* , 702 - 715.

Masood, A. (2008a). Optimized polygonal approximation by dominant point deletion. *Pattern Recognition Volume 41, Issue 1* , 227 - 239.

Nelson, M. (1991). Arithmetic Coding + Statistical Modeling = Data Compression. *Dr. Dobb's Journal* .

Network-Press.Org::En Tierra Firme. (2003-2008). Obtenido de http://www.network-press.org/

Pajares-Martinsanz, G., & de la Cruz-García, J. (2002). *Visión por computador: imágenes digitales y aplicaciones.* México D.F.: Alfaomega Ra-Ma.

Papert, S. (1972). Uses of technology to enhance education. *Technical Report 298, AI lab, MIT* .

Rao, R. P. (2002). Awakening a Sleeping Cat: A Review of "Information Theory and the Brain" edited by R.Baddeley, P. Hancock and P.Földiák. *Neural Networks Volume 15, Issue 7* , 927-929.

Rosenfeld, A., & Johnston, E. (1973). Angle detection on digital curves. *IEEE Transactions on Computers Volume 22* , 875 - 878.

Rosenfeld, A., & Klette, R. (2004). *Digital Geometry: Geometric Methods for Digital Picture Analysis.* San Francisco, CA: Morgan Kaufmann.

Rosenfeld, A., & Pfaltz, J. (1966). Sequential operations in digital picture processing. *J.ACM Volume 13* , 471 - 494.

Rosenfeld, A., & Weszka, J. (1975). An improved method of angle detection on digital curves. *IEEE Transactions on Computers Volume C-24, Issue 9* , 940 - 941.

Rosin, P., & Žunić, J. (2005). Measuring Rectilinearity. *Computer Vision and Image Understanding Volume 99, Issue 2* , 175 - 188.

Said, A. (2003). Arithmetic Coding. *Elsevier Science* , 101 - 151.

Sánchez-Cruz, H. (2006). A Proposal Method for Corner Detection with an Orthogonal Three-Direction Chain Code. *Lecture Notes in Computer Science Volume 4179* , 161 - 172.

Sánchez-Cruz, H., & Rodríguez-Dagnino, R. M. (2005). Compressing bilevel images by means of a three-bit chain code. *Optical Engineering Volume 44, Issue 9* , 097004-1 - 097004-8.

Sánchez-Cruz, H., Bribiesca, E., & Rodríguez-Dagnino, R. (2007). Efficiency of chain codes to represent binary objects. *Patter Recognition Volume 40, Issue 6* , 1660 - 1674.

Shannon, C. (1948). A Mathematical Theory of Communication. *The Bell System Technical Journal Volume 27* , 379-423, 623-656.

Tanimoto, T. (1957). *IBM Internal Report 17th Nov. 1957.*

Touretzky, D. S. (2004). Basics of Information Theory.

Wu, W.-Y. (2003). Dominant point detection using adaptative bending value. *Image and Vision Computing Volume 21, Issue 6* , 517 - 525.

ÍNDICE DE FIGURAS

ÍNDICE DE DEFINICIONES

ÍNDICE DE FÓRMULAS

ÍNDICE DE TABLAS

ÍNDICE DE PSEUDOCÓDIGO