Otro tipo de búsquedas internas

Universidad Distrital Francisco José de Caldas

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Ingeniería de Sistemas

Profesor: Julio Cesar Florez Baez.

Grupo 020-83

Brayan Estiven Aguirre Aristizabal - 20231020156

-

-

Equipo 3

Ciencias de la Computación II

Bogotá D.C

1. Búsquedas por Residuos

1.1 Primera Definición (Cormen)

Abramson define la búsqueda por residuos como un tipo de búsqueda por transformación de claves, en la cual el proceso consiste en calcular el residuo de la división de la clave entre el número de componentes de la estructura de almacenamiento utilizada, como un arreglo.

1.2 Segunda Definición (Knuth)

Una búsqueda por residuos es un algoritmo y método de búsqueda para encontrar un valor particular en una estructura de datos al verificar cada uno de sus elementos. Busca un elemento procesando dígitos individuales y finalmente se somete a un método lineal también para asegurarse de la presencia o ausencia del elemento en la estructura. (Knuth, 1998)

1.3 Tercera Definición (SedgeWick)

Los métodos de búsqueda por residuos son técnicas que analizan secuencialmente los bits individuales de las claves de búsqueda, en lugar de realizar comparaciones completas entre claves en cada iteración. A diferencia de métodos como el hashing, que transforman las claves, estos algoritmos operan directamente sobre los bits originales de las claves, lo que los hace eficientes cuando los bits son accesibles y las claves están uniformemente distribuidas. (Sedgewick, 1983)

Entre sus ventajas destacan:

1. Rendimiento predecible: Ofrecen un tiempo de ejecución estable en el peor caso, evitando la complejidad de estructuras como los árboles equilibrados.
2. Flexibilidad con claves variables: Manejan claves de longitud variable de forma natural.
3. Optimización de espacio: Almacenan fragmentos de las claves dentro de la estructura misma, reduciendo el consumo de memoria.
4. Velocidad competitiva: Superan o igualan la eficiencia de árboles binarios y tablas de dispersión en ciertos escenarios. (Sedgewick, 1983)

2. Tipos de Búsqueda por Residuos

2.1. Arboles de Búsqueda Digital

2.1.1 Primera Definición (Knuth)

Knuth define un **árbol binario de búsqueda (ABB)** como una estructura de datos jerárquica en la que cada nodo contiene una clave y tiene a lo sumo dos hijos: un hijo izquierdo y un hijo derecho. La clave de cada nodo en el árbol sigue la propiedad fundamental de los árboles de búsqueda:

- Las claves en el subárbol izquierdo son menores que la clave del nodo padre.
- Las claves en el subárbol derecho son mayores que la clave del nodo padre.

Ejemplo:

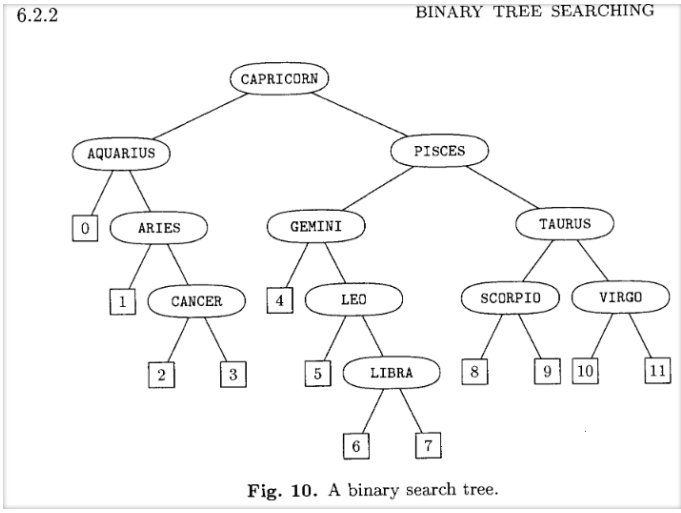


Fig. 10. A binary search tree.

2.1.2 Segunda Definición (Abramson)

Abramson describe un árbol binario de búsqueda como una estructura de datos que facilita la búsqueda de elementos dentro de un conjunto, utilizando la disposición de los nodos y sus conexiones para identificar el valor buscado.

Ejemplo:

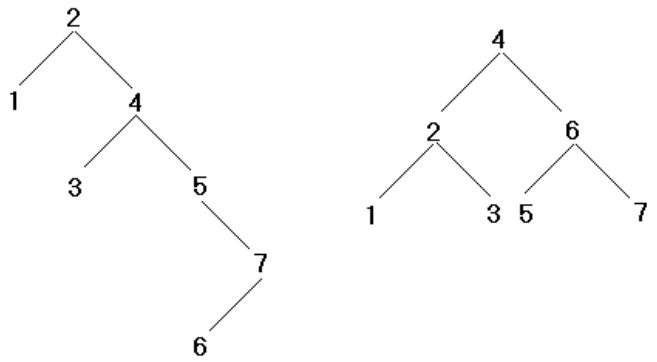
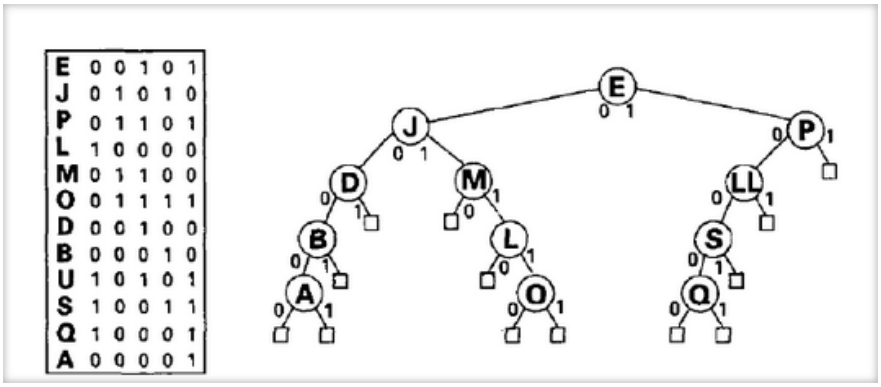


Figura 1:Dos ABB con los mismos elementos.

2.1.3 Tercera Definición (SedgeWick)

Robert Sedgewick describe un árbol binario de búsqueda como una estructura de datos eficiente para almacenar y recuperar elementos de forma ordenada. Según Sedgewick, la propiedad principal de un árbol de búsqueda es que cada nodo contiene una clave, y estas claves se organizan de manera que la búsqueda pueda realizarse de manera eficiente.

Ejemplo:



2.2. Tries de búsqueda por residuos

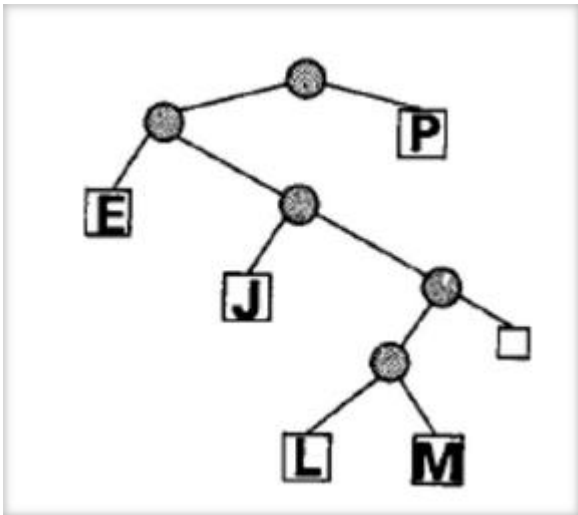
2.2.1 Primera Definición (O'Rourke)

Un trie, o árbol de prefijos, es una estructura de datos en la que cada nodo representa un dígito o carácter de una clave. La búsqueda en un trie se efectúa de manera secuencial, comparando cada dígito de la clave de búsqueda con el correspondiente en el árbol, lo que permite descartar rápidamente grandes subconjuntos de claves que comparten un

mismo prefijo. Esta característica lo hace ideal para aplicaciones como el autocompletado o los diccionarios digitales, donde el procesamiento incremental es fundamental.

La estructura del trie facilita una organización muy eficiente de las claves, ya que permite la reutilización de nodos comunes para claves con prefijos idénticos. De este modo, no solo se optimiza la memoria al evitar redundancias, sino que también se agiliza el proceso de búsqueda al reducir la cantidad de comparaciones necesarias. Esta propiedad es especialmente ventajosa en sistemas con una gran cantidad de entradas donde el rendimiento es crucial.

Ejemplo:

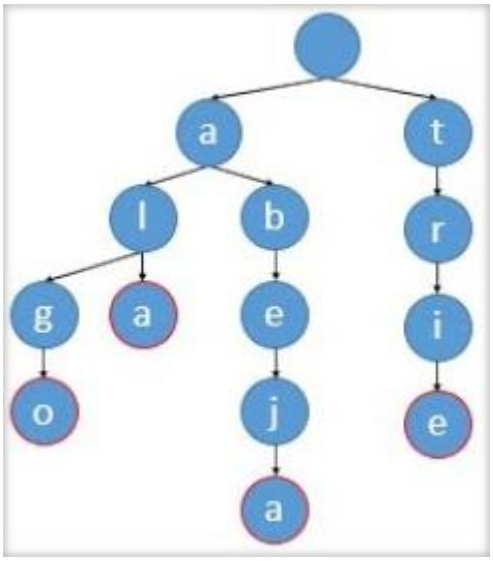


2.2.2 Segunda Definición (Preparata, F. P., & Shamos)

En esta visión, los tries se describen como estructuras jerárquicas que particionan las claves basándose en sus bits o caracteres. Cada nivel del árbol corresponde a una posición en la clave, lo que significa que el tiempo de búsqueda es proporcional a la longitud de la clave en lugar del número total de elementos almacenados. Esto ofrece una ventaja significativa en términos de escalabilidad y predictibilidad del rendimiento.

El diseño de los tries es particularmente útil en escenarios en los que las claves tienen longitudes variables o cuando se maneja un gran conjunto de entradas. La partición por niveles permite que cada comparación se focalice en un dígito específico, eliminando rápidamente ramas enteras que no pueden contener el valor buscado. Este método reduce la complejidad del proceso de búsqueda y mejora la eficiencia en aplicaciones de búsqueda intensiva, como bases de datos de palabras o sistemas de codificación.

Ejemplo:

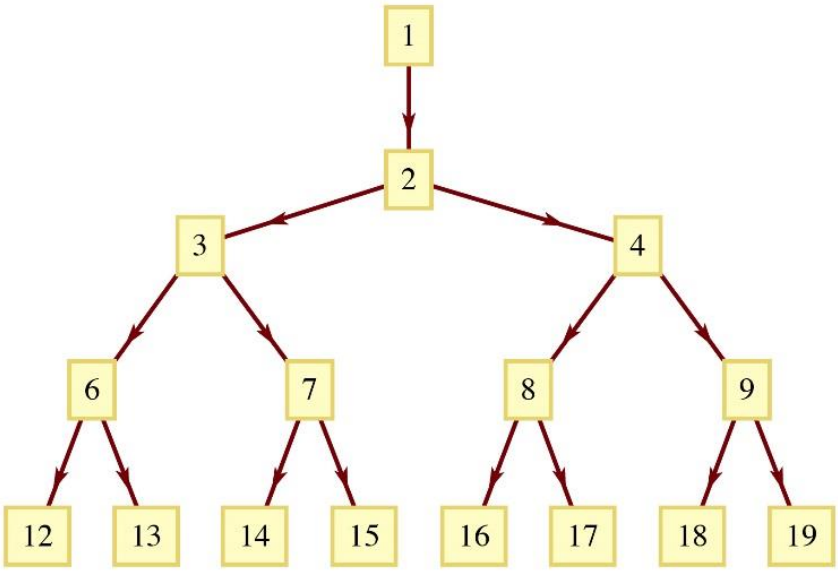


2.2.3 Tercera Definición (Wirth)

Niklaus Wirth plantea que los tries son, en esencia, árboles de decisión en los que cada nodo actúa como un selector de dígitos o bits. Esta metodología evita comparaciones redundantes al procesar cada dígito individualmente, lo que permite descartar subconjuntos completos de claves irrelevantes en cada paso de la búsqueda. La ventaja de este enfoque reside en su capacidad para reducir significativamente el tiempo de búsqueda en grandes conjuntos de datos.

La estructura de un trie facilita además la implementación de algoritmos de búsqueda que requieren una respuesta casi instantánea, ya que la decisión en cada nodo reduce drásticamente el espacio de búsqueda. Este modelo se adapta muy bien a situaciones en las que los datos se distribuyen de forma amplia y se requiere un procesamiento rápido y eficiente, como en sistemas de reconocimiento de patrones o en la indexación de grandes volúmenes de información digital.

Ejemplo:

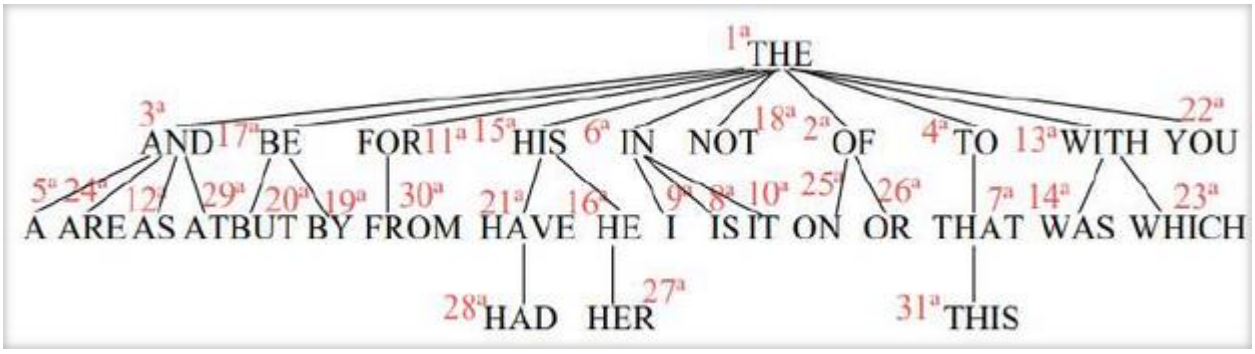


2.3. Árboles de Búsqueda por residuos Múltiples

2.3.1 Primera Definición (Bentley)

Se definen los árboles de búsqueda por residuos múltiples como estructuras que aplican una serie de funciones de residuo (o hash) de manera secuencial en distintos niveles del árbol. Cada nivel utiliza una función diferente para dividir el espacio de claves, lo que mejora la distribución de las entradas y reduce las colisiones que suelen presentarse en métodos basados en un único residuo. El proceso de aplicar múltiples funciones permite abordar de forma más equilibrada conjuntos de datos con alta dimensionalidad.

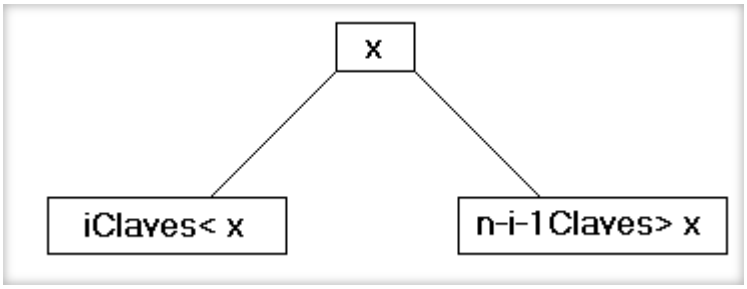
Ejemplo



2.3.2 Segunda Definición (Cormen)

Desde otra perspectiva, estos árboles se conciben como estructuras en las que cada nodo contiene un arreglo de hijos, indexado mediante residuos calculados con bases distintas. Esta técnica combina lo mejor de los métodos de los tries con las ventajas del hashing múltiple, permitiendo que la estructura se adapte dinámicamente a distribuciones no uniformes y a claves con múltiples componentes. El uso de diferentes bases en cada nivel facilita la segmentación fina del espacio de claves, lo que es particularmente útil en aplicaciones de alta complejidad.

Ejemplo:

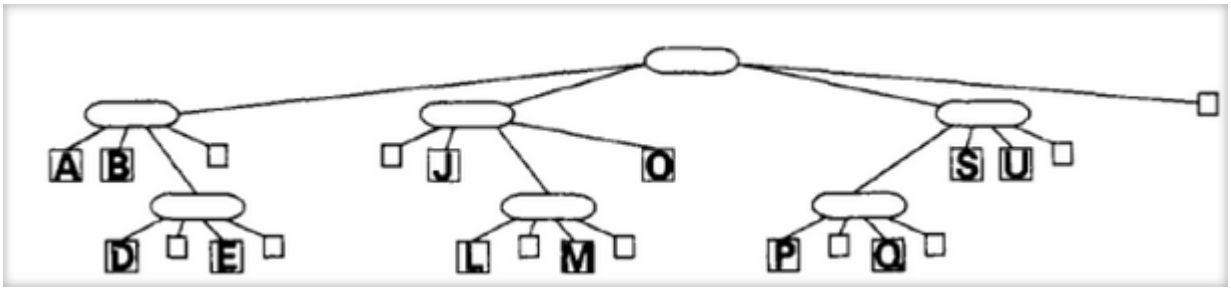


2.3.3 Tercera Definición (Wirth)

Los árboles de búsqueda por residuos múltiples se pueden ver como una generalización de los árboles digitales clásicos. En lugar de procesar un solo dígito o bit por nivel, estos árboles utilizan funciones de residuo para seleccionar subárboles en cada etapa de la búsqueda. Esto permite adaptar la estructura a distribuciones de claves que no sean uniformes, optimizando la eficiencia al dividir el espacio de búsqueda en función de criterios más sofisticados.

La utilización de funciones de residuo en cada nivel ofrece una mayor granularidad en la división del espacio, lo que puede resultar en una reducción significativa de las colisiones y, por ende, en una mejora del tiempo de búsqueda. Esta técnica es especialmente valiosa en sistemas en los que la diversidad de las claves es alta y la uniformidad en su distribución no está garantizada, proporcionando una solución adaptable y robusta frente a diferentes escenarios de consulta.

Ejemplo

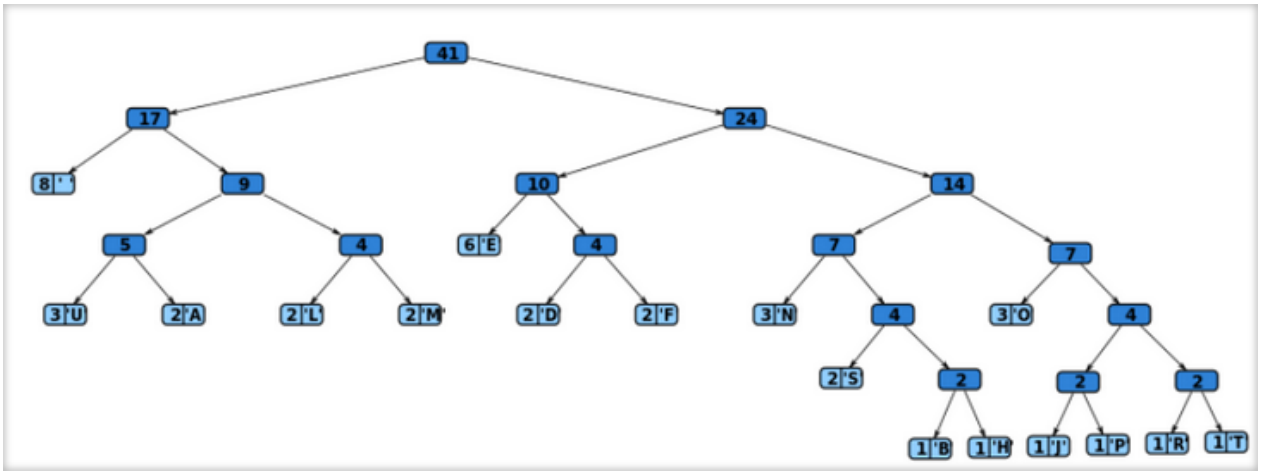


3. Arboles de Huffman

3.1 Primera Definición (O'Rourke)

Los árboles de Huffman se definen como estructuras binarias diseñadas para minimizar el costo esperado de codificación. En este esquema, cada símbolo del conjunto de datos se representa mediante una hoja, y la longitud del código asignado a cada símbolo es inversamente proporcional a su frecuencia de aparición. La construcción del árbol se realiza mediante un proceso voraz (greedy) que combina iterativamente los dos nodos de menor frecuencia, garantizando así la optimalidad de la codificación.

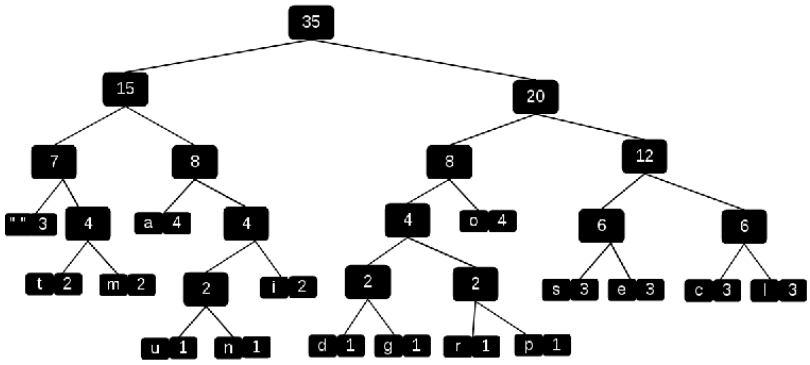
Este método es fundamental en la compresión de datos, ya que asegura que los símbolos más frecuentes utilicen códigos más cortos, reduciendo el tamaño global del mensaje sin pérdida de información. La relación directa entre frecuencia y longitud del código convierte a los árboles de Huffman en una herramienta imprescindible en sistemas de compresión sin pérdida, optimizando tanto el almacenamiento como el ancho de banda requerido para la transmisión de datos.



3.2 Segunda Definición (Tanenbaum)

Tanenbaum sugiere que un árbol de Huffman se describe como una estructura de tipo greedy, en la que la construcción se basa en la fusión progresiva de nodos con frecuencias bajas hasta formar la raíz del árbol. Cada fusión garantiza que los símbolos menos frecuentes se combinen, resultando en códigos más largos, mientras que los símbolos comunes obtienen códigos más cortos. Esta característica es esencial para lograr la máxima eficiencia en la codificación.

Ejemplo

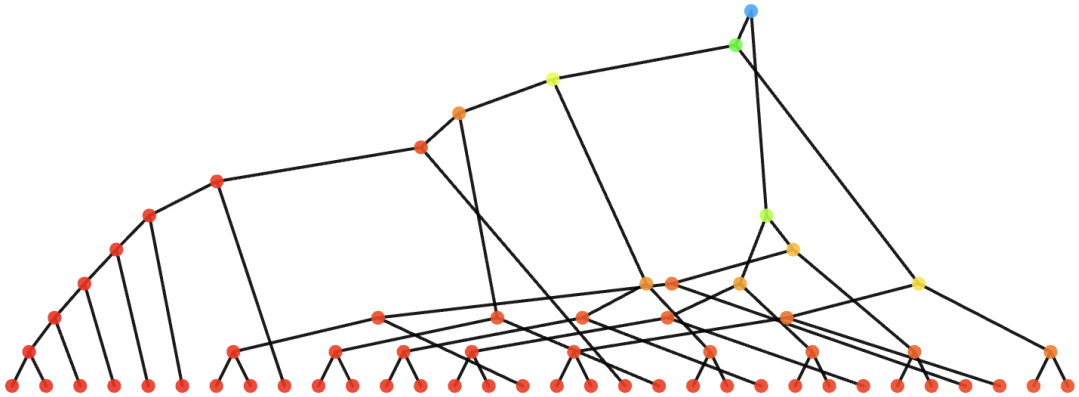


3.3 Tercera Definición (Sayood)

En la visión de Sayood, los árboles de Huffman se entienden como una aplicación concreta de la teoría de la codificación óptima de Shannon-Fano. Aquí, la entropía del sistema dicta la longitud mínima promedio de los códigos, haciendo que el árbol de Huffman sea una representación práctica de estos principios teóricos. La distribución de la frecuencia de los símbolos es la base para asignar códigos que reflejen la importancia relativa de cada símbolo en el mensaje.

Esta técnica se destaca por su capacidad para alcanzar la eficiencia teórica en la compresión de datos, ya que utiliza la información estadística del conjunto de símbolos para optimizar la representación. La relación entre entropía y longitud de código es aprovechada para construir una estructura que no solo minimiza el costo de codificación, sino que también facilita una decodificación inmediata y sin errores. La solidez de este método ha llevado a su adopción en numerosos estándares de compresión modernos.

Ejemplo



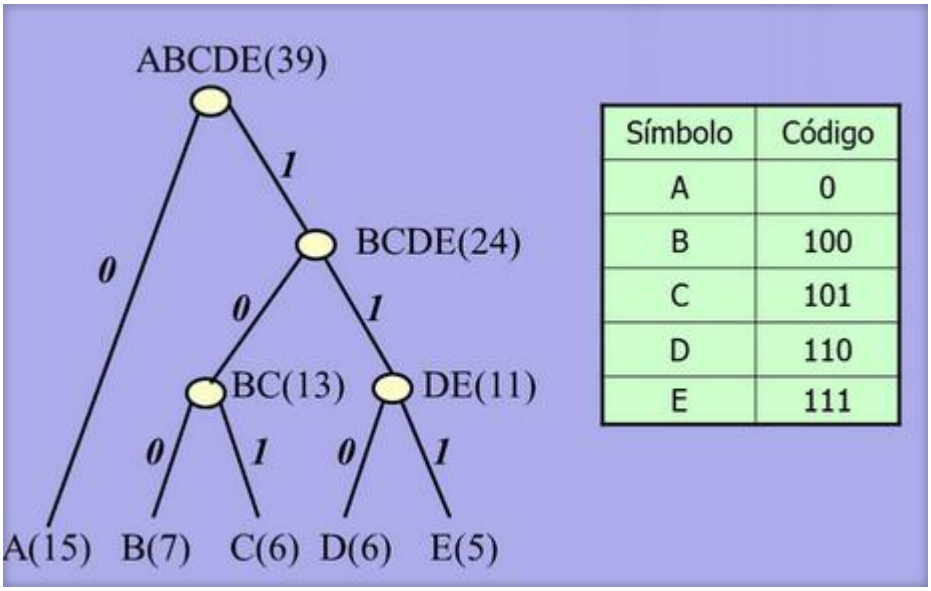
4. Códigos de Huffman

4.1 Primera Definición (sedgeWick)

Los códigos de Huffman se definen como códigos de prefijo variable asignados a símbolos en función de su frecuencia de aparición. La propiedad de prefijo único garantiza que ningún código pueda ser confundido con la concatenación de otros, lo que permite una decodificación inmediata y sin ambigüedad. Este tipo de codificación es altamente eficiente, ya que asigna códigos más cortos a los símbolos que se repiten con mayor frecuencia y códigos más largos a los menos comunes.

La aplicación de estos códigos en sistemas de compresión de datos se basa en la idea de minimizar la longitud total del mensaje codificado. Al eliminar la posibilidad de solapamientos en los códigos (debido a la propiedad de prefijo), se facilita la decodificación sin necesidad de marcadores adicionales. Esta característica resulta especialmente valiosa en entornos de transmisión de datos, donde la velocidad y la precisión en la recuperación de la información son críticas.

Ejemplo:



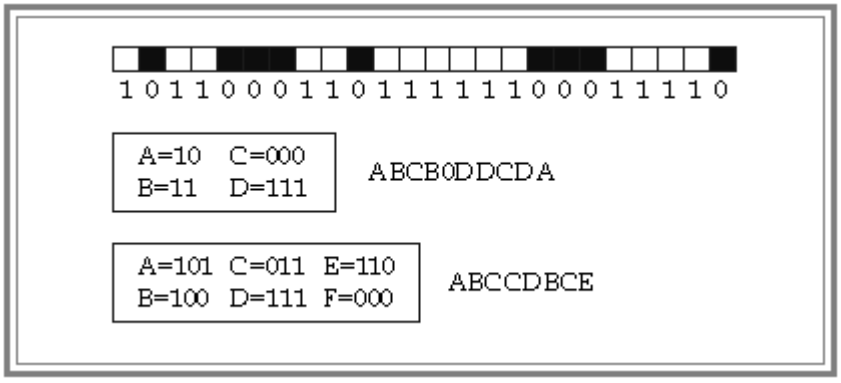
Símbolo	Código
A	0
B	100
C	101
D	110
E	111

4.2 Segunda Definición (Nelson)

Desde el punto de vista de Nelson, los códigos de Huffman se consideran óptimos para fuentes estacionarias, ya que minimizan la longitud esperada del mensaje bajo la restricción de tener códigos con prefijos únicos. El diseño de estos códigos se fundamenta en la distribución probabilística de los símbolos, de manera que aquellos que aparecen con mayor frecuencia reciben códigos más compactos. Esta adaptación a la frecuencia de ocurrencia convierte a los códigos de Huffman en una solución ideal para la compresión sin pérdida.

La estructura de estos códigos permite una reducción significativa en el tamaño de los datos, al tiempo que asegura que la información pueda ser reconstruida de forma exacta. La optimización se logra a través de un proceso que equilibra la frecuencia de cada símbolo con la longitud del código asignado, resultando en una representación eficiente y práctica para una amplia variedad de aplicaciones, desde la compresión de archivos hasta la transmisión en redes de comunicación.

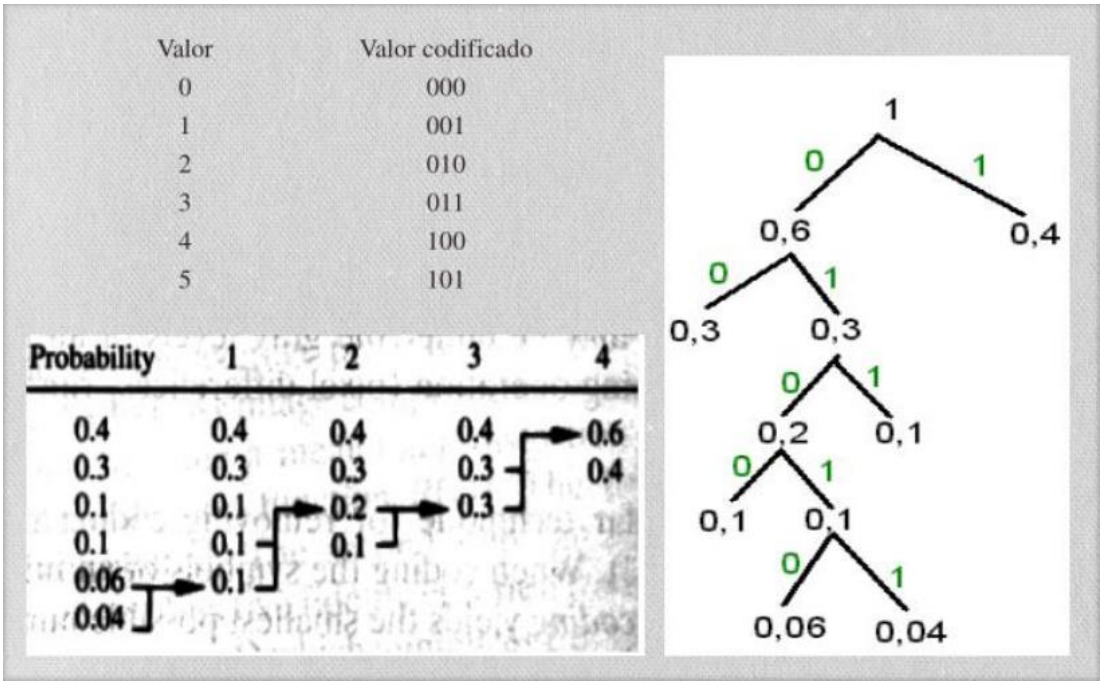
Ejemplo:



4.3 Tercera Definición (Gallager)

Gallager enfatiza que los códigos de Huffman son instantáneamente decodificables, lo que significa que cada símbolo puede ser identificado tan pronto se recibe su código completo, sin necesidad de esperar a que se acumulen bits adicionales. Esta propiedad de decodificación inmediata es esencial en aplicaciones donde la latencia debe minimizarse, como en sistemas de transmisión en tiempo real o en dispositivos con recursos limitados.

Ejemplo:



5. Búsqueda por Rangos

5.1 Primera Definición (Preparata, F. P., & Shamos)

La búsqueda por rangos se define como una técnica para recuperar todos aquellos elementos cuyas claves se encuentren dentro de un intervalo específico. Se suelen utilizar estructuras de datos especializadas, como árboles-B o árboles de segmentos, que permiten realizar estas consultas en tiempo logarítmico. Este método es fundamental en aplicaciones de bases de datos y sistemas de información geográfica, donde la filtración de datos en función de un rango es una operación común y crítica.

El principal atractivo de la búsqueda por rangos reside en su capacidad para manejar grandes conjuntos de datos sin necesidad de escanear la totalidad de los elementos. Al utilizar índices o estructuras balanceadas, se reduce drásticamente la cantidad de comparaciones requeridas, lo que se traduce en una mejora sustancial del rendimiento en entornos con altas demandas de consulta. La metodología es escalable y se adapta bien a sistemas dinámicos donde los datos se actualizan con frecuencia.

5.2 Segunda Definición (Gallager)

Desde la óptica de espacios multidimensionales, la búsqueda por rangos se aborda mediante estructuras especializadas como los árboles k-d, que dividen recursivamente el espacio de datos. Este método es especialmente útil cuando se trabaja con información que se extiende a lo largo de varias dimensiones (por ejemplo, coordenadas geográficas), ya que permite descartar rápidamente regiones irrelevantes y concentrar la búsqueda en áreas específicas. La división recursiva facilita el manejo de consultas complejas y de alto volumen.

La estructura resultante de estos métodos permite que la consulta se realice de forma incremental y que se aproveche la organización espacial inherente a los datos. Esto no solo optimiza el tiempo de respuesta, sino que también reduce la carga computacional al evitar el procesamiento innecesario de datos fuera del rango especificado. El enfoque se adapta bien a aplicaciones de visualización y análisis de grandes conjuntos de datos multidimensionales, donde la eficiencia es esencial.

5.3 Tercera Definición (Knuth)

Knuth plantea la búsqueda por rangos como una operación fundamental en el manejo de bases de datos, en la que el uso de índices balanceados permite responder consultas de intervalo en tiempos muy competitivos. En este enfoque, la estructura de índices garantiza que la consulta solo recorra la parte del conjunto de datos que efectivamente cae dentro del rango especificado, evitando el recorrido completo del conjunto. Esto es

especialmente valioso en sistemas donde el rendimiento en la recuperación de datos es crítico.

El empleo de índices balanceados y estructuras optimizadas permite que la búsqueda por rangos se ejecute con una complejidad logarítmica, lo que es determinante para la escalabilidad del sistema. La precisión y la rapidez en la respuesta a consultas de intervalo hacen de este método una técnica robusta para entornos de bases de datos, en los cuales la eficiencia operativa y la capacidad de respuesta ante consultas complejas son aspectos prioritarios.

6. Tipos de Búsqueda por Rangos

6.1 Métodos Elementales

6.1.1 Primera Definición (Bentley)

El término **"métodos elementales"** generalmente se refiere a técnicas básicas y directas para resolver problemas de consulta sobre rangos en estructuras de datos. Estas técnicas suelen ser menos eficientes que las soluciones avanzadas, pero son más fáciles de implementar y entender.

Algunos de los **métodos elementales** incluyen:

1. **Búsqueda Lineal**
    a. Si se tiene un array de longitud N, se puede recorrer todo el rango de interés y calcular el resultado en O(n).
    b. Es ineficiente para múltiples consultas, pero puede ser útil si solo se hace una búsqueda.
2. **Uso de Prefix Sums (Sumas Acumuladas)**
    a. Se preprocesa un array acumulado en O(n) y luego se responde una consulta en O(1).
    b. Aplicable a problemas como suma de elementos en un rango.
3. **Sparse Tables o tablas dispersas (para funciones idempotentes, como mínimo/máximo)**
    a. son una estructura de datos estática que permite responder consultas de rangos de manera eficiente, particularmente para operaciones **idempotentes** como el mínimo, máximo, máximo común divisor (GCD), o operaciones bit a bit (AND, OR).

Ejemplo: Queremos calcular la suma de los números entre las posiciones 2 y 5 en el arreglo [3, 1, 4, 1, 5, 9, 2].

- **Búsqueda Lineal**: Se recorren los elementos 4, 1, 5, 9 y se suman manualmente: 4 + 1 + 5 + 9 = 19 (O(n)).
- **Prefix Sums**: Se preprocesa el arreglo [3, 4, 8, 9, 14, 23, 25], y la suma se obtiene en O(1) con: Prefix[5] - Prefix[1] = 23 - 4 = 19.
- **Sparse Table**: Si queremos el mínimo en el rango [2,5], usamos la tabla precalculada y respondemos en O(1).

6.1.2 Segunda Definición (Preparata, F. P., & Shamos)

En el ámbito de la computación, el término "métodos elementales" se refiere a técnicas básicas y sencillas utilizadas para resolver problemas de consulta por rangos en estructuras de datos, las cuales, aunque menos eficientes que soluciones más avanzadas, destacan por su facilidad de implementación y comprensión. Entre estos métodos se encuentra la búsqueda lineal, que consiste en recorrer un array de longitud N para calcular el resultado en un rango específico con una complejidad de O(n), siendo ineficiente para múltiples consultas pero práctica si solo se realiza una única búsqueda. Otro enfoque es el uso de sumas acumuladas (prefix sums), donde se preprocesa un array acumulado en O(n) para luego responder consultas en O(1), ideal para operaciones como la suma de elementos en un rango. Además, están las sparse tables, diseñadas para funciones idempotentes como el mínimo o el máximo, que requieren un preprocesamiento en O(n log n) pero permiten consultas en O(1), aunque no admiten actualizaciones eficientes. Estos métodos representan soluciones fundamentales y accesibles para problemas de rangos, especialmente cuando la simplicidad es prioritaria sobre la optimización.

Ejemplo: Un sistema de ventas almacena los ingresos diarios en un arreglo. Queremos saber cuánto se vendió entre los días 10 y 20.

- **Búsqueda Lineal**: Se suman los valores manualmente de los días 10 al 20.
- **Prefix Sums**: Se usa un arreglo acumulado y se restan dos valores para obtener la suma en O(1).
- **Sparse Table**: Si quisiéramos conocer el día con la menor venta en ese rango, la respuesta se obtiene en O(1)

6.1.3 Tercera Definición (Tanenbaum)

Los **métodos elementales en algoritmos de búsqueda** son técnicas fundamentales utilizadas para localizar elementos en estructuras de datos, caracterizadas por su simplicidad y aplicabilidad en escenarios básicos. Estos métodos incluyen principalmente dos enfoques. Los *prefix sums* y los *sparse tables*. Los **prefix sums** son acumulados parciales que registran la suma de todos los elementos desde el inicio del arreglo hasta una posición específica, permitiendo calcular la suma de cualquier subarreglo en tiempo constante mediante la diferencia entre dos prefix sums. Por su parte, los **sparse tables** son una estructura pre calculada que almacena resultados para rangos de longitud potencia de 2 (1, 2, 4, 8...), diseñada para operaciones asociativas como mínimo, máximo o Máximo Común Divisor.

Ejemplo: Un sistema de monitoreo de temperatura guarda valores cada hora. Queremos saber la temperatura mínima entre la 1:00 p.m. y las 6:00 p.m.

- **Prefix Sums**: Se usa un acumulado para calcular promedios de temperatura en segundos.
- **Sparse Table**: Se preprocesa el mínimo por rangos y se obtiene en O(1)

6.2. Métodos de la rejilla

6.2.1 Primera Definición (SedgeWick)

Los métodos de la rejilla se fundamentan en la partición espacial del área de búsqueda mediante una cuadrícula o grilla regular. Cada celda de la grilla agrupa un subconjunto de datos, lo que permite que las consultas de vecindad se realicen únicamente en las celdas relevantes. Esta técnica es ampliamente utilizada en aplicaciones geométricas y de gráficos por computadora, donde la localización espacial de los elementos es determinante para la eficiencia de la búsqueda.

La segmentación en celdas posibilita descartar rápidamente grandes regiones que no pueden contener el objeto o dato buscado, lo que reduce la cantidad de comparaciones y acelera el proceso de consulta. Esta técnica es particularmente efectiva en entornos donde la distribución de los datos es homogénea y se requiere una rápida identificación de vecinos o elementos cercanos, optimizando la utilización de recursos y mejorando el rendimiento general del sistema.

Ejemplo: Un dron debe encontrar la estación de carga más cercana en un campo de 100x100 metros.

- Se divide el área en celdas de 10x10 metros.
- Solo se buscan estaciones en la celda actual y sus vecinas, en lugar de analizar los 10,000 puntos.

Esto reduce la cantidad de comparaciones y acelera la búsqueda.

6.2.2 Segunda Definición (Berg)

Según Berg, los métodos de la rejilla son especialmente eficaces en aplicaciones de renderización 3D y en la detección de colisiones, ya que permiten pre calcular qué celdas se intersectan con un objeto en movimiento o en un escenario estático. La pre asignación de elementos a celdas de la grilla facilita la reducción de comparaciones innecesarias, ya que solo se examinan aquellas celdas que tienen una alta probabilidad de contener elementos relevantes para la consulta.

Este enfoque de partición espacial se destaca por su capacidad para gestionar de forma eficiente escenas complejas, donde el procesamiento en tiempo real es crucial. La utilización de la rejilla permite que el sistema se enfoque únicamente en las regiones de interés, reduciendo la sobrecarga computacional y mejorando la respuesta en aplicaciones críticas, como en videojuegos o simulaciones físicas.

Ejemplo: En un juego, hay 1000 proyectiles y 500 enemigos moviéndose.

- En lugar de comparar cada proyectil con cada enemigo (*1000×5001000 \times 500*1000×500), se divide el mapa en celdas.
- Cada proyectil solo revisa enemigos en su celda o celdas cercanas.

Esto optimiza la detección de colisiones.

6.2.3 Tercera Definición (González)

En el ámbito del procesamiento de imágenes, González aplica el concepto de rejilla para optimizar operaciones de filtrado y convolución. En esta perspectiva, la rejilla actúa como

una máscara que permite acceder a píxeles vecinos de manera ordenada, lo que facilita la aplicación de filtros y algoritmos de transformación en la imagen. Este método permite procesar grandes volúmenes de datos visuales de manera eficiente, al segmentar la imagen en regiones manejables.

La implementación de métodos de rejilla en procesamiento digital no solo mejora la velocidad de los algoritmos de filtrado, sino que también contribuye a una mayor precisión en la manipulación de la información espacial. Al dividir la imagen en celdas, se logra un acceso más directo y estructurado a los datos, lo que es fundamental en aplicaciones de reconocimiento de patrones y en el desarrollo de sistemas de visión por computadora de alto rendimiento.

Ejemplo: Se aplica un filtro de desenfoque a una imagen.

- En lugar de analizar cada píxel individualmente, la imagen se divide en bloques de 8x8 píxeles.
- El desenfoque se calcula para cada bloque en lugar de procesar píxel por píxel, acelerando el procesamiento.

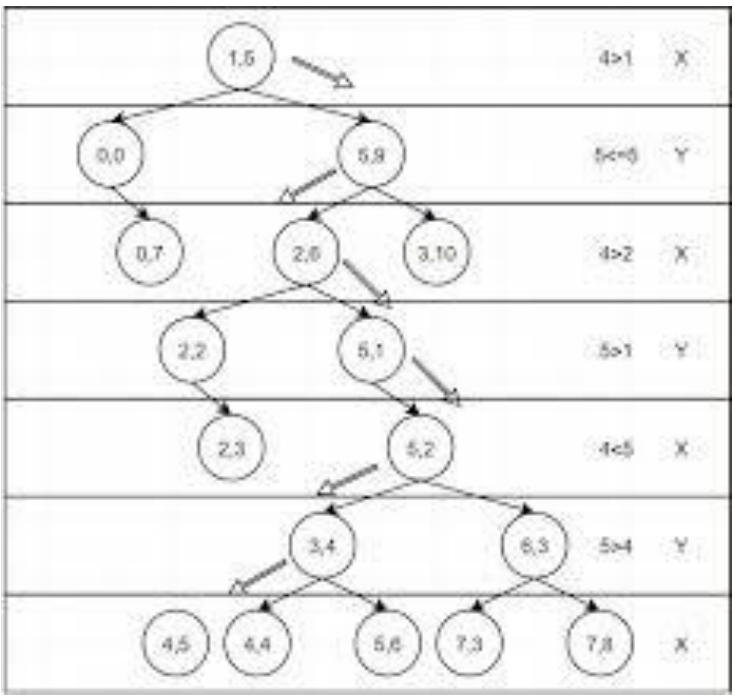
6.3. Arboles 2D (KD)

6.3.1 Primera Definición (Tanenbaum)

Los árboles k-d (k-dimensionales) se definen como estructuras binarias que dividen recursivamente el espacio en hiperplanos alternantes, utilizando ejes distintos en cada nivel de la estructura. En cada nodo, la decisión de partición se basa en una comparación con un punto de división que segmenta el espacio en dos regiones, facilitando la búsqueda y la organización de puntos en espacios multidimensionales. Esta técnica es esencial para manejar conjuntos de datos con más de una dimensión, permitiendo búsquedas eficientes en espacios complejos.

La estructura del árbol k-d se adapta muy bien a la búsqueda de vecinos más cercanos y a las consultas ortogonales, ya que cada división reduce drásticamente la región de búsqueda. Este método es ampliamente utilizado en aplicaciones como la minería de datos, la visión por computadora y los sistemas de navegación, donde la organización espacial y la rapidez en la consulta son factores determinantes para el rendimiento del sistema.

Ejemplo:

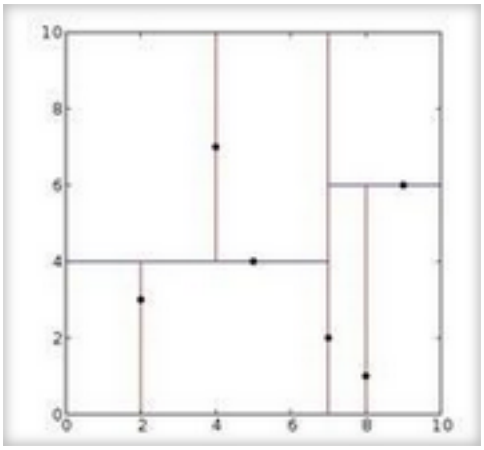


6.3.2 Segunda Definición (Gallager)

De acuerdo con esta perspectiva, los árboles k-d permiten realizar búsquedas ortogonales en datos multidimensionales de manera muy eficiente gracias a la selección inteligente de puntos de división. Cada partición se elige de forma que se balancee la distribución de puntos en ambos subárboles, lo que garantiza una estructura equilibrada y un tiempo de respuesta óptimo en las consultas. La división sucesiva del espacio en función de diferentes ejes posibilita que la búsqueda se concentre únicamente en las regiones relevantes.

La capacidad de descartar rápidamente grandes regiones del espacio no solo mejora el rendimiento en la búsqueda, sino que también permite que el algoritmo se escale a conjuntos de datos de alta dimensión. Este enfoque es particularmente valioso en escenarios donde se requieren búsquedas rápidas y precisas en espacios complejos, como en aplicaciones de análisis geoespacial o en sistemas de recomendación basados en características multidimensionales.

Ejemplo:

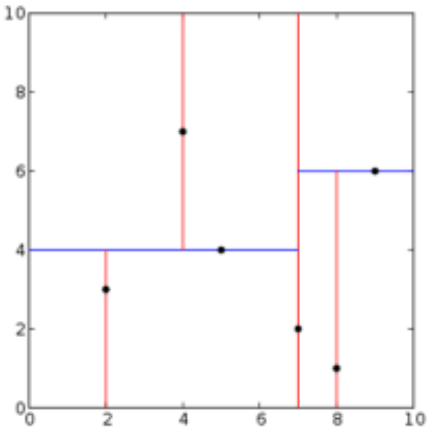


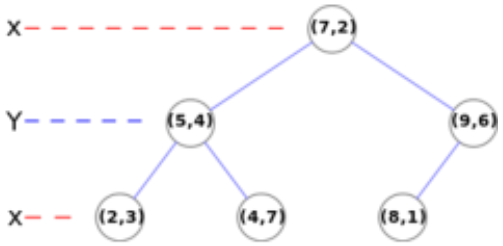
6.3.3 Tercera Definición (Friedman)

Friedman destaca que los árboles k-d son especialmente útiles en algoritmos de “nearest neighbor”, donde la estructura jerárquica permite descartar de forma rápida regiones alejadas del punto de consulta. La técnica se basa en la capacidad del árbol para organizar los datos de forma que, tras cada comparación, se pueda eliminar un conjunto completo de puntos que no pueden ser los más cercanos. Esto se traduce en una mejora considerable en el tiempo de búsqueda en aplicaciones críticas.

La organización en árbol facilita que, durante la búsqueda, se realicen comparaciones solo con aquellos nodos que se encuentran en la vecindad del punto consultado. Este proceso selectivo reduce la carga computacional y permite que la búsqueda se realice de forma casi instantánea, lo que es esencial en aplicaciones en tiempo real como la robótica, el reconocimiento de patrones y los sistemas de navegación, donde la rapidez y precisión son imperativas.

Ejemplo:





7. Bibliografía

- Abramson, N. (1981). *Teoría de la información y codificación*. Recuperado de https://www.cartagena99.com/recursos/alumnos/apuntes/Teoria_de_la_Informacion_y_codificacion-Norman_Abramson_ebook-spanish_.pdf
- Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM, 18*(9), 509–517.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3ª ed.). MIT Press.
- de Berg, M., Cheong, O., van Kreveld, M., & Overmars, M. (2008). *Computational geometry: Algorithms and applications* (3ª ed.). Springer.
- Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software, 3*(3), 209–226.
- Gallager, R. G. (1978). *Principles of digital communication*. Cambridge, MA: Cambridge University Press.
- González, R. C., & Woods, R. E. (2004). *Digital image processing* (2ª ed.). Prentice Hall.
- Knuth, D. E. (1998). *The art of computer programming, Volume 3: Sorting and searching*. Recuperado de https://www.haio.ir/app/uploads/2022/01/The-Art-of-Computer-Programming-Volume-3-Sorting-and-Searching-by-Donald-E.-Knuth-z-lib.org_.pdf
- Nelson, M. (1996). *Data compression: Techniques and applications*. Prentice Hall.
- O'Rourke, J. (1998). *Computational geometry in C*. Cambridge University Press.
- Preparata, F. P., & Shamos, M. I. (1985). *Computational geometry: An introduction*. Springer.

- Sayood, K. (2017). *Introduction to data compression* (5ª ed.). Elsevier.
- Sedgewick, R. (1983). *Algorithms*. Recuperado de https://dsp-book.narod.ru/Algorithms.pdf
- Tanenbaum, A. S., & Austin, T. (2013). *Structured computer organization* (6ª ed.). Pearson.
- Wirth, N. (1976). *Algorithms + data structures = programs*. Prentice-Hall.