

**SEGUNDA CONSULTA – BUSQUEDAS POR RESIDUOS**

**Presentado a:**

Julio Cesar Florez Baez

**Presentado por:**

Johan Esteban Castaño Martinez - 20191020029

Jhony Alejandro Caro Umbariba - 20191020055

Samuel Andrés Romero Bueno - 20191020127

**Equipo número 1**

Facultad de Ingeniería.

Ciencias de la Computación II.

17 de septiembre de 2022.

**INDICE**

[1. Búsquedas por residuos 3](#_Toc113805475)

[2. Clasificación 4](#_Toc113805476)

[2.1. Arboles Digitales](#_Toc113805478) 4

[2.1.1. Definición de arbol digital. 4](#_Toc113805479)

[2.1.2. Búsqueda por residuos Tries. 6](#_Toc113805479)

2.1.3. [Búsquedas por residuos múltiples 8](#_Toc113805480)

[3. Árboles de Huffman 10](#_Toc113805481)

1. **Búsquedas por residuos:**
   1. Primera definición

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.[[1]](#footnote-1)

* 1. Segunda definición

Varios métodos de búsqueda producen examinando las claves de búsqueda a razón de un bit cada vez, en lugar de hacer comparaciones completas entre claves en cada paso. Estos métodos, denominados métodos de búsqueda por residuos, trabajan con los bits de las propias claves, y no con las versiones transformadas de las claves utilizadas en la dispersión. Al igual que los de ordenación por residuos, estos métodos pueden ser muy útiles cuando los bits de las claves de búsqueda son fácilmente manipulables y los valores de las claves están bien distribuidas.

La ventaja principal de los métodos de búsqueda por residuos es que proporcionan un rendimiento razonable en el peor caso, sin las complicaciones de los árboles equilibrados; también proporcionan un método fácil para utilizar claves de longitud variable; algunos permiten incluso ganar espacio almacenando parte de la clave dentro de la estructura de búsqueda; y, finalmente, permiten un acceso muy rápido a los datos, compitiendo tanto con los árboles binarios de búsqueda como con la dispersión. Sus inconvenientes son que toda inclinación en los datos puede provocar un mal rendimiento por degeneración de los árboles (y todo dato compuesto por caracteres está inclinado) y que algunos de estos métodos pueden malgastar inútilmente el espacio de la memoria. Al igual que con la ordenación por residuos, estos métodos se diseñan para aprovechar las características particulares de las arquitecturas de las computadoras: puesto que utilizan las propiedades digitales de las claves, es difícil, o imposible, hacer implementaciones eficaces en algunos lenguajes de alto nivel.[[2]](#footnote-2)

* 1. Tercera definición:

Recuperar información de la memoria es una de las operaciones básicas de muchas aplicaciones de computador. Y hay bastantes variantes de este problema, donde se puede hacer una diferenciación entre métodos de búsqueda. Una forma de dividir la gran colección de algoritmos de búsqueda es distinguirlos entre algoritmos que usan comparación entre una clave simple y los que están basados en el uso de claves como cadenas de caracteres. Otra forma de distinguirlos son los algoritmos que usan estructuras de datos reconstruidas para facilitar la búsqueda, y, por otro lado, los que solo utilizan una simple y lineal estructura de datos como un arreglo o una lista enlazada.

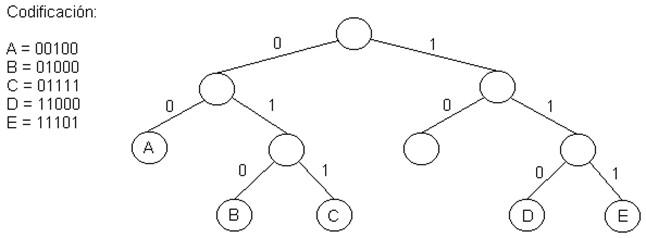
Haciendo un análisis de las definiciones consultadas, se puede decir que una búsqueda por residuos es aquella que utiliza comparaciones completas entre claves en cada paso, es decir esta clave está conformada por una cadena de caracteres, lo cual conforma una clave aún más compleja a diferencia de una que se podría tener en una búsqueda secuencial o lineal. Adicionalmente, también se puede decir que una búsqueda por residuos es aquella que se hace en estructuras no lineales, por ejemplo, en un trie o en un árbol digital de búsqueda.[[3]](#footnote-3)

1. **Clasificación de búsqueda por residuos:**
   1. **Arboles Digitales:**
      1. **Definiciones de Árbol Digital:**
         1. Primera definición:

Cuando los elementos de un conjunto se pueden representar como una secuencia de bits (ninguna representación de un elemento en particular es prefijo de otra):

Se puede usar un árbol digital, el cual es un árbol binario donde la posición de un elemento no depende de su valor, sino de su representación binaria, los elementos se almacenan solo en sus hojas, pero no necesariamente todas las hojas contienen elementos. también se conocen como trie.[[4]](#footnote-4)

Ejemplo:

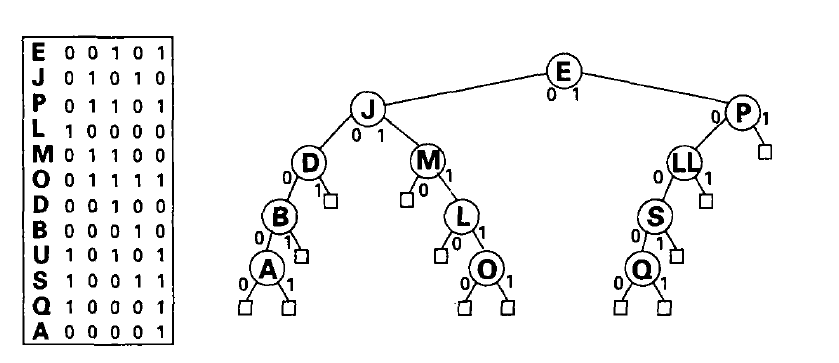


*Imagen 1: Obtenida de “Métodos de búsqueda”.* [*http://ual.dyndns.org/*](http://ual.dyndns.org/)

* + - 1. Segunda definición:

El método de búsqueda por residuos más simple es el de búsqueda digital: el algoritmo es precisamente el mismo que el de búsqueda por árbol binario, excepto que el movimiento por las ramas del árbol no se hace de acuerdo con el resultado de una comparación entre claves, sino con los bits de la clave. En el primer nivel se utiliza el primer bit, en el segundo nivel se utiliza el segundo bit, y así hasta encontrar un nodo externo o una hoja.[[5]](#footnote-5)

Ejemplo:



*Imagen 2: Extraída de “Algorithms” – Sedgewick.[[6]](#footnote-6)*

* + - 1. Tercera definición:

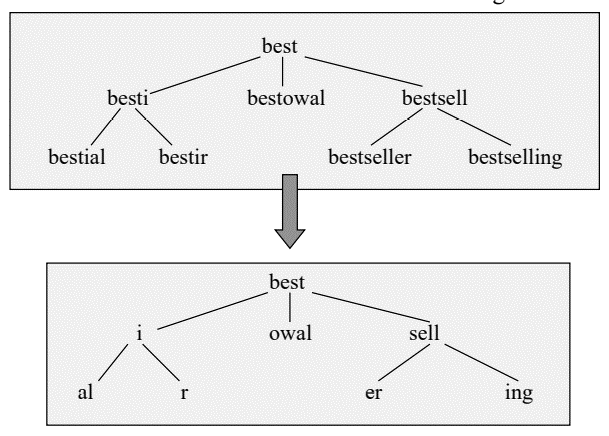
El código es virtualmente el mismo que el de la búsqueda por árbol binario. La única diferencia es que las claves son del tipo clavebits utilizado en la ordenación por residuos y se utiliza la función bits, para tener acceso a los bits individuales, en lugar de las comparaciones entre claves.

* + 1. **Búsqueda por residuos tries:**
       1. Primera definición:

Un caso particular de árboles digitales son los Tries, estos son árboles que fueron propuestos independientemente por Rene de la Briandais en Proceedings Western Joint Computer Conference 1959 ypor Edward Fredkin en 1960, a este último le deben su nombre; el mismo proviene de la extracción deletras de la palabra retrieval (recuperación) pero como su pronunciación en inglés, “tri”, se confundíacon tree (árbol, que también se pronuncia “tri”) se tomó como convención pronunciarlo como “try” queen ingles se pronuncia “trai”. Esta es una estructura de datos adecuada para realizar búsquedas rápidasde cadenas en un texto grande, se utiliza en aplicaciones tales como búsquedas de patrones, indexacióny compresión de texto, recuperación de secuencias de texto, biología computacional, etc.

El trie es un árbol M-ario, que permite el almacenamiento de cadenas de símbolos de un alfabetofinito de cardinalidad M. Cada nodo interno del árbol es un vector de M posiciones. Cada posición enel vector se corresponde con cada carácter del alfabeto y contiene un puntero que es un posible caminoa recorrer durante una búsqueda.[[7]](#footnote-7)

Ejemplo:

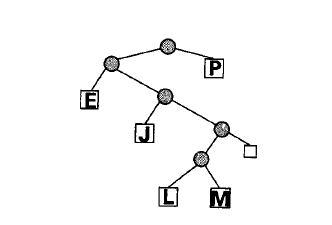


*Imagen 3: Extraída de “Técnicas Avanzadas de Programación” – Javier Campos*

* + - 1. Segunda definición:

La idea es no almacenar claves en los nodos internos del árbol, sino poner todas en los nodos externos. Esto es, en lugar de utilizar z para los nodos terminales de la estructura, se ponen nodos que contienen las claves de búsqueda. Así pues, se tienen dos tipos de nodos en la estructura: nodos internos, que sólo contienen enlaces a otros nodos, y nodos terminales que contienen claves y no enlaces. (Fredkin denominó a este método trie porque es útil para la extracción («retrieval»), palabra que suele pronunciarse (drai-b) o simplemente (drab). Para buscar una clave en una estructura como ésta, es preciso moverse por las ramas de acuerdo con sus bits, al igual que anteriormente, pero sin comparar la clave con nada hasta que no se alcance un nodo externo. Cada clave del árbol se almacena en un nodo terminal del camino descrito por el conjunto de los primeros bits de la clave y, como cada clave de búsqueda termina en un nodo terminal, se necesita una comparación completa de las claves para terminar la búsqueda.[[8]](#footnote-8)

Ejemplo:



*Imagen 4: Extraída de “Algorithms” – Sedgewick.[[9]](#footnote-9)*

* + - 1. Tercera definición:

Tiene dos defectos molestos: la «ramificación de una sola vía», que provoca la creación de nodos extra en el árbol, y la existencia de dos tipos diferentes de nodos, lo que complica en cierta forma el código (en especial el de una inserción).

En el contexto presente, trie permite la búsqueda de N claves de longitud arbitraria en un árbol que tiene exactamente N nodos, necesitando sólo una comparación completa por búsqueda.

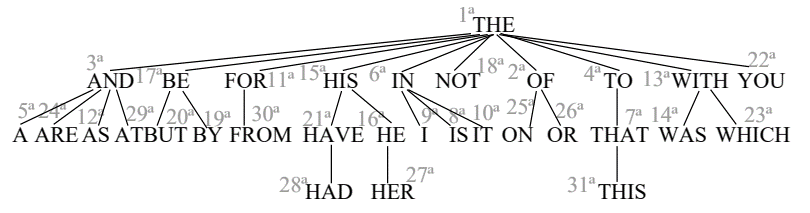
La ramificación unidireccional se evita gracias a un simple recurso: cada nodo contiene el índice del bit que se deberá comprobar para decidir qué camino tomar cuando se salga de este nodo. Se evitan los nodos terminales reemplazando los enlaces hacia los nodos externos por enlaces que apuntan hacia niveles superiores del árbol, lo que lleva a la noción habitual de los nodos normales del árbol, con una clave y dos enlaces. Pero en trie, las claves de los nodos no se utilizan para controlar la búsqueda en el recorrido hacia abajo del árbol; simplemente se almacenan como referencia para cuando se alcance el fondo del árbol.[[10]](#footnote-10)

* + 1. **Búsqueda por residuos múltiples:**
       1. Primera definición:

Partiendo de la problemática de los tries, donde, la mayoría de los nodos internos tienen un solo hijo, lo que aumenta el coste en espacio. La idea es implementar un árbol binario, pero, evitando las bifurcaciones de una sola dirección. Patricia (Practical Algorithm To Retrieve Information Coded In Alphanumeric) es una representación de un trie en la que todos los nodos con un solo hijo “se mezclan” con sus padres.[[11]](#footnote-11)

Ejemplo:

La búsqueda en al árbol es binaria, pero puede ampliarse fácilmente a m-aria (m > 2), para un alfabeto con m símbolos, así:



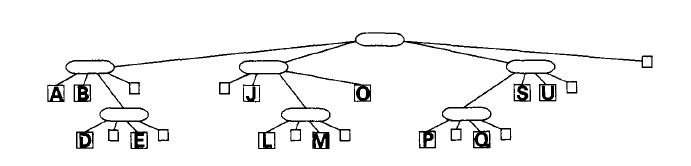
*Imagen 5: Extraída de “Árbol de búsqueda digital” – Javier Campos[[12]](#footnote-12)*

Los datos insertados en el mismo orden, pero en un árbol digital de búsqueda de orden 27.

* + - 1. Segunda definición:

En la ordenación por residuos se vio que se pueden obtener importantes mejoras en la velocidad considerando varios bits a la vez. Esto es también cierto en la búsqueda por residuos: examinando m bits a la vez, se puede aumentar la velocidad en un factor 2m. Sin embargo, hay una situación que impone algo más de cuidado al aplicar esta idea, lo que no fue necesario en la ordenación por residuos. El problema es que considerar m bits a la vez implica utilizar nodos con M = 2m enlaces, lo que puede conducir a derrochar un volumen considerable de espacio por los enlaces no utilizados.[[13]](#footnote-13)

Ejemplo:



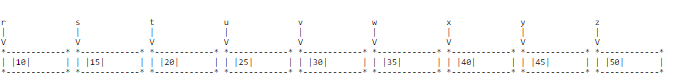
*Imagen 6: Extraída de “Algorithms” – Sedgewick.[[14]](#footnote-14)*

* + - 1. Tercera definición:

Los Árboles de Búsqueda de M Vías (M-Way Search Trees) son árboles en los cuales todos los nodos son de grado menor o igual a M, y tienen las siguientes propiedades

* Si T es un Árbol de Búsqueda de M Vías y está vacío:
  + T tiene el valor de un apuntador nulo
* Si T es un Árbol de Búsqueda de M Vías y no esta vacío y T tiene el valor de un apuntador a un nodo del tipo n, A0, (K1, A1, B1), (K2, A2, B2), ..., (Kn, An, Bn) donde:
  + n, igual o menor a M-1, es el número de claves que tiene el nodo.
  + Ki, i variando de 1 a n, son valores de claves.
  + Ai, i variando de 0 a n, son apuntadores a un subárbol de T.
  + Bi, i variando de 1 a n, son apuntadores a los registros de clave Ki.
  + Ki debe ser menor que Ki+1, i variando de 0 a n.
  + Todos los valores de clave del subárbol Ai deben ser menores que el valor de clave Ki+1 para i variando de 0 a n-1.
  + Todos los valores de clave en el subárbol An son mayores que el valor de clave Kn.
  + Los subárboles de T, apuntados por Ai, i variando de 0 a n, son también Arboles de Búsqueda de M Vías.[[15]](#footnote-15)

Ejemplo archivo con 9 registros:



*Imagen 7: Extraída de “Estructura de Datos: Árbol de Búsqueda de M Vías (M-Way Search Tree)”.[[16]](#footnote-16)*

1. **Arboles de Huffman:**
   1. Primera definición:

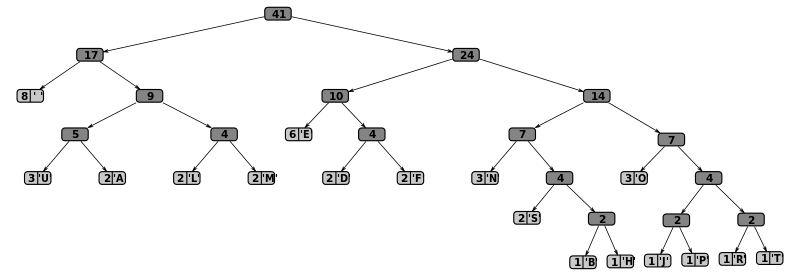
Los arboles de búsqueda en realidad se comportan mejor para cadenas comprimidas. La mayoría de los esquemas de compresión basados en modelos modernos utilizan la codificación Huffman, la codificación de Huffman tiene el efecto de suavizar la entrada, por lo que construir el árbol tiene un menor gasto de memoria Los arboles Huffman consisten en la creación de un árbol binario en el que se etiquetan los nodos hoja con los caracteres, junto a sus frecuencias, y de forma consecutiva se van uniendo cada pareja de nodos que menos frecuencia sumen, pasando a crear un

nuevo nodo intermedio etiquetado con dicha suma. Se procede a realizar esta acción hasta que no quedan nodos hoja por unir a ningún nodo superior, y se ha formado el árbol binario.

Posteriormente se etiquetan las aristas que unen cada uno de los nodos con ceros y unos (hijo derecho e izquierdo, respectivamente, por ejemplo). El código resultante para cada carácter es la lectura, siguiendo la rama, desde la raíz hacia cada carácter (o viceversa) de cada una de las etiquetas de las aristas.[[17]](#footnote-17)

Ejemplo:

El siguiente es un árbol de Huffman generado para las frecuencias de apariciones exactas del texto "ESTO ES UN EJEMPLO DE UN ARBOL DE HUFFMAN.

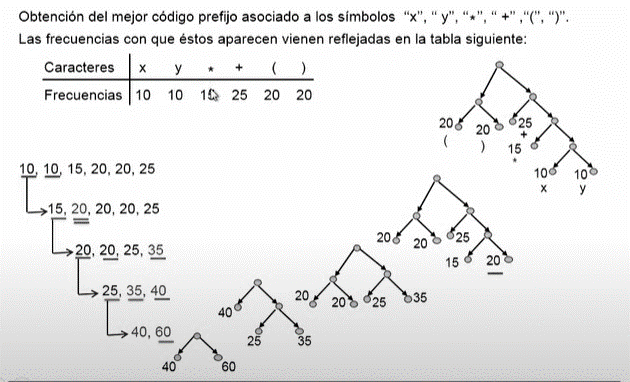


*Imagen 7: Ejemplo de un árbol de Huffman (Redor 2008).*

* 1. Segunda definición:

Se desarrolla un método óptimo para codificar un conjunto de mensajes que consta de un número finito de miembros. Un código de redundancia mínima es uno construido de tal manera que él se minimiza el número medio de dígitos de codificación por mensaje.[[18]](#footnote-18)

Ejemplo:



*Imagen 8: Extraído de Codificación de Huffman (Conejero, A).[[19]](#footnote-19)*

* 1. Tercera definición:

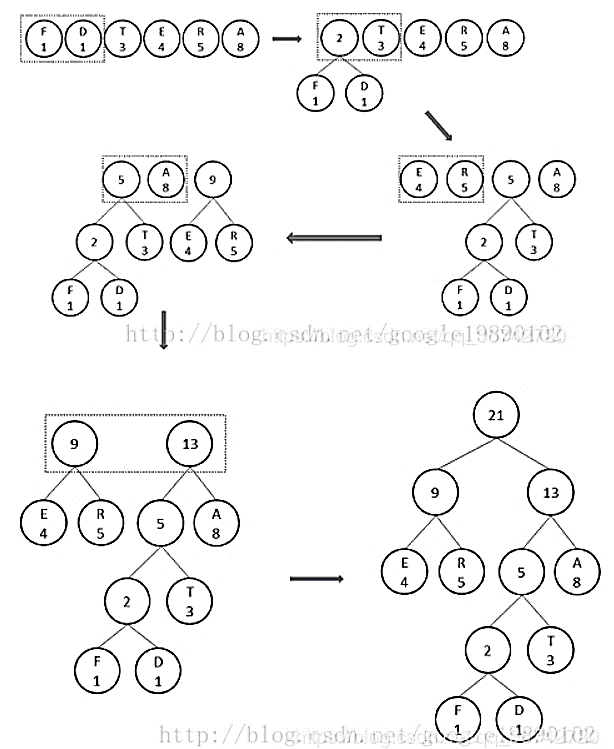
Es un tipo de estructura de datos que se utiliza para la compresión o encriptación de datos mediante el estudio de la frecuencia de aparición de caracteres. El algoritmo funciona a partir de un conjunto dado de símbolos con sus respectivos pesos. Los pesos son la frecuencia de aparición en una cadena. El algoritmo se basa en el uso de un árbol binario dónde las hojas representan los símbolos del conjunto de entrada. Para conseguir el código de Huffman asociado a cada símbolo únicamente hay que seguir las aristas que unen la raíz con la hoja determinada.

Estructura de datos de Huffman:

El algoritmo de Huffman permite encontrar un código con las propiedades especificadas. La idea del algoritmo consiste en llegar a formar un árbol binario que represente el código. En los nodos terminales (hojas) de este árbol se almacenarán los colores que se desea codificar junto con la probabilidad de aparición de cada uno de ellos, cumpliéndose que los colores más probables están más próximos a la raíz que los menos probables. De manera que el camino desde la raíz del árbol a cualquier hoja del mismo representará el código para el color de esa hoja, de acuerdo con el criterio de que los subárboles izquierdos se etiquetan con un 0 y los subárboles derechos lo hacen con un 1. Según este criterio, los colores con mayor probabilidad de aparición tendrán los códigos de menor longitud, mientras que los que tengan una probabilidad de aparición menor tendrán asociado un código de mayor longitud.

Codificación de Huffman:

La técnica funciona creando un árbol binario de nodos. Un nodo puede ser un nodo hoja o un nodo interno. Inicialmente, todos los nodos son nodos hoja, que contienen el carácter en sí, el peso (frecuencia de aparición) del carácter. Los nodos internos contienen peso de carácter y enlaces a dos nodos secundarios. Como convención común, un nodo 0 representa seguir al hijo izquierdo, y un nodo 1 representa seguir al hijo correcto. Un árbol terminado tiene n nudos de hojas y n-1 nodos internos.[[20]](#footnote-20)



*Imagen 9: Extraído de “Estructura de datos y algoritmo”.[[21]](#footnote-21)*

**Bibliografía**

[1]. Abramson, N. (1981). Teoría de la Información y Codificación. Madrid: McGRAW-HILL

[2]. Conejero, A., & Jordán, C. (19 de febrero de 2013). S4.12- Codificación de Huffman. UPV, Valencia, España. Obtenido de <https://www.youtube.com/watch?v=W6WZT12ruGQ&t=730s>

[3]. Fano, R. (1961). The Transmission of Information. MIT Res. Lab. Electron. Tech.,149.

[4]. Huffman, D. (1952). A Method for the Construction of Minimum-Redundancy Codes. Proceedings of the I.R.E, 1098-1101.

[5]. Molina, R. (s.f.). Tema 3: Codificación Huffman. 8. Obtenido de <http://www.kramirez.net/RI/Material/Internet/T3_CODIGO_DE_HUFFMAN.pdf>

[6]. Nilsson, S. (1996). Radix Sorting & Searching. Lund: LUTEDX.

[7]. Ramachandran, R. (2011). Radix Search- an Alternative to Linear Search. Journal of E- Technology, 154-158.

[8]. Sedgewick. R. (1946). Algorithms. Providence: Addison Wesley.

[9]. (2015) Estructuras de Datos y Algoritmos: Árboles Digitales: Trie y Patricia. Universidad Nacional de San Luis

[10]. Campos J. “Técnicas Avanzadas de Programación”

[11]. Estructura de datos y algoritmo: árbol de Huffman y codificación de Huffman. (s.f.). <https://programmerclick.com/article/1833650549/>

[12]. techiedelight. (18 de Septiembre de 2022). Obtenido de <https://www.techiedelight.com/es/huffman-coding/>

[13]. Facultad de Ciencias Exactas, Ingeniería y Agrimensura. (5 de Septiembre de 2022). Estructura de Datos: Arbol de Busqueda de M Vias (M-Way Search Tree). Obtenido de https://www.fceia.unr.edu.ar/estruc/2005/arbomway.htm

1. (Ramachandran, R. Radix Search- an Alternative to Linear Search, pág.154-158) [↑](#footnote-ref-1)
2. (Sedgewick, R. Algorithms., pág. 45) [↑](#footnote-ref-2)
3. (Nilsson, S. Radix Sorting &amp; Searching. pág. 45) [↑](#footnote-ref-3)
4. (Conejero, A.,Codificación de Huffman., pág. 45) [↑](#footnote-ref-4)
5. (Sedgewick, R. Algorithms., pág. 45) [↑](#footnote-ref-5)
6. (Sedgewick, R. Algorithms., pág. 45) [↑](#footnote-ref-6)
7. (Universidad Nacional de San Luis,. Estructuras de Datos y Algoritmos: Árboles Digitales: Trie y Patricia.) [↑](#footnote-ref-7)
8. (Sedgewick, R. Algorithms., pág. 45) [↑](#footnote-ref-8)
9. (Sedgewick, R. Algorithms., pág. 45) [↑](#footnote-ref-9)
10. (techiedelight,. huffman-coding) [↑](#footnote-ref-10)
11. (Campos,. “Técnicas Avanzadas de Programación”,. pág. 45) [↑](#footnote-ref-11)
12. (Campos,. “Técnicas Avanzadas de Programación”,. pág. 45) [↑](#footnote-ref-12)
13. (Sedgewick, R. Algorithms., pág. 45) [↑](#footnote-ref-13)
14. (Sedgewick, R. Algorithms., pág. 45) [↑](#footnote-ref-14)
15. (Facultad de Ciencias Exactas, Ingeniería y Agrimensura) [↑](#footnote-ref-15)
16. (Facultad de Ciencias Exactas, Ingeniería y Agrimensura) [↑](#footnote-ref-16)
17. (Abramson, N., Teoría de la Información y Codificación., pág. 45) [↑](#footnote-ref-17)
18. (Huffman, D., A Method for the Construction of Minimum-Redundancy Codes., pág. 45) [↑](#footnote-ref-18)
19. (Conejero, A.,Codificación de Huffman., pág. 45) [↑](#footnote-ref-19)
20. (techiedelight,. huffman-coding) [↑](#footnote-ref-20)
21. (Estructura de datos y algoritmo: árbol de Huffman y codificación de Huffman.) [↑](#footnote-ref-21)