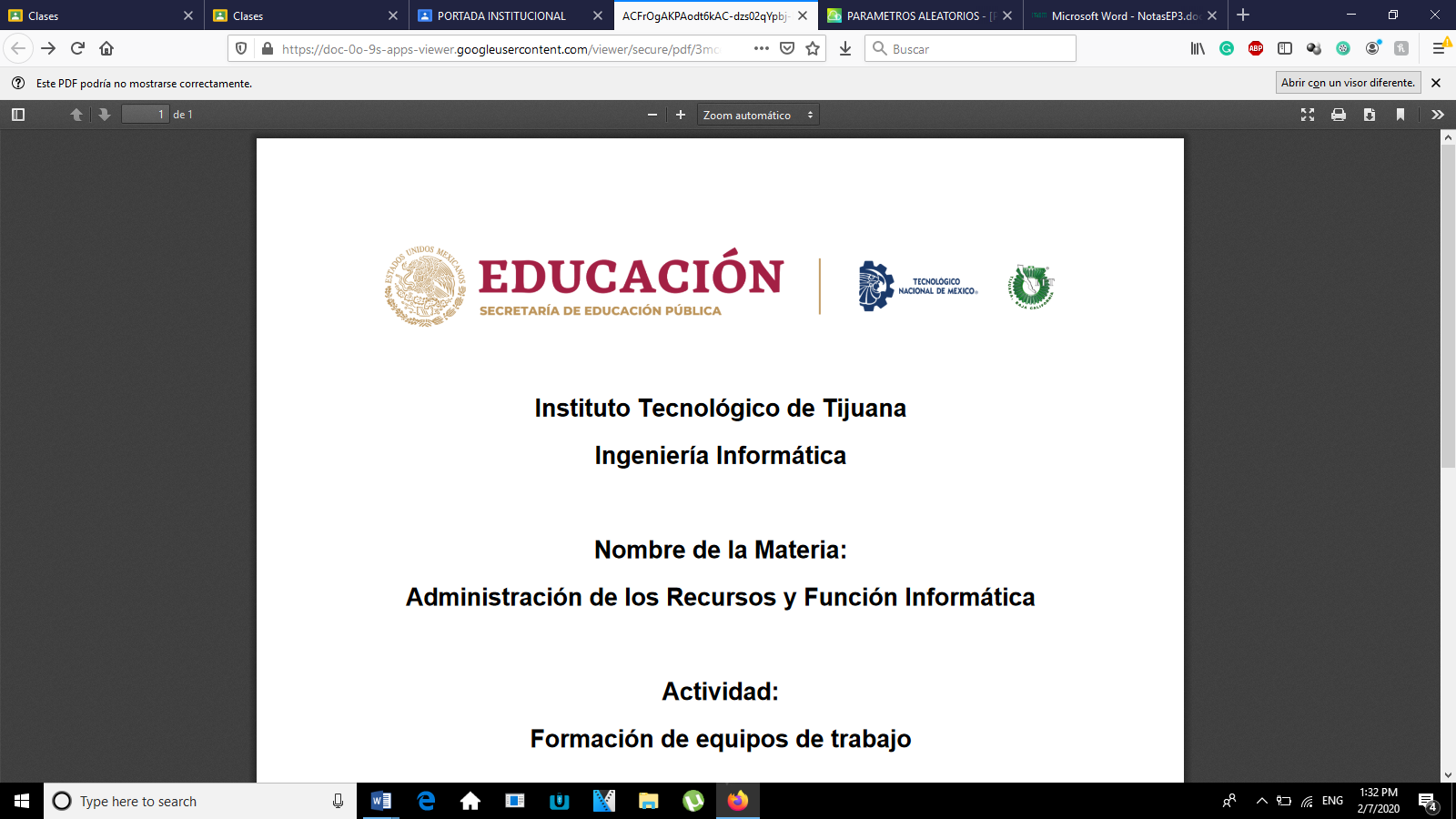
Tecnológico Nacional de México Instituto Tecnológico de Tijuana

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE: AGOSTO – DICIEMBRE 2021

Ingeniería en Sistemas

Estructura de datos

Investigación unidad 6

Unidad 6

Guillen Martínez Anthony 20210575

**M.C.C. LUZ ELENA CORTEZ GALVAN**

02/10/2021

ÍNDICE

**6.0 - Métodos de Búsqueda 3**

**6.1 - Búsqueda Secuencial 5**

**6.2 - Búsqueda Binaria 7**

**6.3 - Búsqueda por Funciones Hash 11**

**Tabla comparativa 14**

**Conclusiones 15**

**Bibliografías 16**

**6.0 - Métodos de Búsqueda.**

**¿Qué son los métodos de búsqueda?**

La búsqueda es el proceso algorítmico de encontrar un ítem particular en una colección de ítems. Una búsqueda normalmente devuelve True o False según el ítem esté o no presente, respectivamente. En ocasiones, el algoritmo se puede modificar para devolver la posición donde se encuentre el ítem. Para nuestros propósitos, simplemente nos ocuparemos de la pregunta sobre la membrecía.

**¿Para qué se utilizan?**

Los métodos de búsqueda se utilizan para encontrar datos dentro de una lista de datos ordenados o no ordenados dependiendo de la situación necesaria habrá diferentes tipos de búsqueda los cuales tendrán mejores resultados dependiendo de la cantidad de datos o de si estos están ordenados o no.

**¿Cuáles métodos existen?**

Búsqueda Secuencial: en este algoritmo cada elemento es revisado para comprobar si él es el elemento buscado.

Búsqueda de Intervalos: estos algoritmos están diseñados para ser más eficientes con listas de datos que ya estén ordenados.

Búsqueda de punto de salto: es una optimización del algoritmo A\* para redes de costo uniforme. Reduce las asimetrías en el proceso de búsqueda mediante la poda (del inglés pruning) del grafo, es decir, se eliminan algunos nodos con base en las suposiciones respecto a los vecinos del nodo actual, siempre y cuando las condiciones las condiciones generales impuestas a la red sean satisfechas

**Búsqueda de interpolación**

La búsqueda por interpolación se asemeja al método por el cual las personas buscan un nombre en un directorio telefónico (el valor clave por el cual se ordenan las entradas del libro): en cada paso, el algoritmo calcula en qué lugar del espacio de búsqueda restante el elemento buscado podría basarse en los valores de clave en los límites del espacio de búsqueda y el valor de la clave buscada, generalmente mediante una interpolación lineal.

**Búsqueda exponencial**

Hay numerosas maneras para implementarlo siendo la más común determinar el rango en que la llave de búsqueda reside y realizar una búsqueda binaria dentro de dicho rango. Esto demora O (log i) dónde i es la posición de la llave de búsqueda en la lista, si la llave de búsqueda está en la lista, o la posición donde la llave de búsqueda debería estar, si la llave de búsqueda no está en la lista.

**Búsqueda de Fibonacci**  
Es un método de búsqueda en un [array ordenado](https://es.wikipedia.org/w/index.php?title=Array_ordenado&action=edit&redlink=1) usando un [algoritmo de divide y vencerás](https://es.wikipedia.org/w/index.php?title=Algoritmo_de_divide_y_vencer%C3%A1s&action=edit&redlink=1) que disminuye las ubicaciones posibles con la ayuda de los [números de Fibonacci](https://es.wikipedia.org/wiki/N%C3%BAmeros_de_Fibonacci).

**6.1 – Búsqueda Secuencial**

**Concepto/Descripción.**

La búsqueda secuencial es una técnica en la cual se toma un valor clave (Elemento a buscar) y se empieza a comparar con todos los demás elementos, registro por registro, el resultado de la comparación es el índice o posición del elemento y en caso de no encontrar valor, el resultado puede definirse como nulo. Del anterior término, se deriva su nombre de Búsqueda secuencial, pues se comparan secuencialmente todos los elementos, desde el inicio hasta el fin de arreglar uno por uno, hasta que el elemento del arreglo se encuentre o hasta que se llegue al final del arreglo. La existencia se puede asegurar desde el momento en que el elemento es localizado, pero no se puede asegurar la no existencia hasta no haber analizado todos los elementos del arreglo. La búsqueda secuencial funciona de forma lineal y es muy útil en arreglos de pocos elementos, o bien para arreglos que no están ordenados. Su 52 efectividad y velocidad se incrementa si los elementos de un arreglo ya se encuentran ordenados, esto es porque reduce el área de búsqueda eliminando los registros que están sobre el elemento clave.

**Ventajas**

* Es un método sumamente simple que resulta útil cuando se tiene un conjunto de datos pequeños (Hasta aproximadamente 500 elementos)
* Es fácil adaptar la búsqueda secuencial para que utilice una lista enlazada ordenada, lo que hace la búsqueda más eficaz.
* Si los datos buscados no están en orden es el único método que puede emplearse para hacer dichas búsquedas.

**Desventajas**

* Este método tiende a ser muy lento.
* Si los valores de la clave no son únicos, para encontrar todos los elementos con una clave particular, se requiere buscar en todo el arreglo, lo que hace el proceso muy largo.

**6.2 - Búsqueda Binaria**

**Concepto/Descripción**

La búsqueda binaria es el método, donde si el arreglo o vector está bien ordenado, se reduce sucesivamente la operación eliminando repetidas veces la mitad de la lista restante.

Este método se puede aplicar tanto a datos en listas lineales como en árboles binarios de búsqueda. Los prerrequisitos para la búsqueda binaria son:

* La lista debe estar ordenada, en un orden específico de acuerdo al valor de la clave.
* Debe conocerse el número de elementos.

Si el conjunto de elementos es grande, el tiempo de búsqueda se puede reducir utilizando el siguiente algoritmo de tipo divide y vencerás:

* Se divide el elemento en dos partes.
* Se determina la parte que debe contener la clave buscada.
* Se repite el proceso en esa parte.

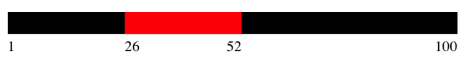
**Funcionamiento**

El proceso comienza comparando el elemento central del arreglo con el elemento buscado. Si ambos coinciden finaliza la búsqueda. Si no ocurre así, el elemento buscado será mayor o menor en sentido estricto que el elemento central del arreglo. Si el elemento buscado es mayor se procede a hacer búsqueda binaria en el subarray superior, si el elemento buscado es menor que el contenido de la casilla central, se debe cambiar el segmento a considerar al segmento que está a la izquierda de tal sitio central

Para poder implementar un algoritmo en un lenguaje de programación, necesitarás entender un algoritmo hasta sus últimos detalles. ¿Cuáles son las entradas del problema? ¿Las salidas? ¿Qué variables deben crearse, y qué valores iníciales deben tener? ¿Qué pasos intermedios deben tomarse para calcular otros valores y calcular en última instancia la salida? ¿Estos pasos repiten instrucciones que se pueden escribir en forma simplificada al usar un bucle?

Veamos cómo describir cuidadosamente la búsqueda binaria. La idea principal de la búsqueda binaria es llevar un registro del rango actual de intentos razonables. Digamos que estoy pensando en un número entre uno y 100, justo como en el juego de adivinar. Si ya intentaste decir 25 y te dije que mi número es más grande, y ya intentaste decir 81 y te dije que mi número es más chico, entonces los números en el rango de 26 a 80 son los únicos intentos razonables. Aquí, la sección roja de la recta numérica contiene los intentos razonables, y la sección negra muestra los intentos que hemos descartado:

En cada turno, haces un intento que divide el conjunto de intentos razonables en dos rangos de aproximadamente el mismo tamaño. Si tu intento no es correcto, entonces te digo si es muy alto o muy bajo, y puedes eliminar aproximadamente la mitad de los intentos razonables. Por ejemplo, si el rango actual de los intentos razonables es de 26 a 80, intentarías adivinar a la mitad del camino, (26 + 80) / 2(26+80)/2 left parenthesis, 26, plus, 80, right parenthesis, slash, 2, o 53. Si después te digo que 53 es demasiado alto, puedes eliminar todos los números de 53 a 80, dejando 26 a 52 como el nuevo rango de intentos razonables, reduciendo a la mitad el tamaño del rango.



Para el juego de adivinar, podemos llevar un registro del conjunto de intentos razonables al usar unas cuantas variables. Sea la variable minminm, i, n el intento razonable mínimo actual para esta ronda, y sea la variable maxmaxm, a, x el intento razonable máximo actual. La entrada (o input en inglés) al problema es el número nnn, el mayor número posible que tu oponente está pensando. Suponemos que el menor número posible es uno, pero sería fácil modificar el algoritmo para tener el menor número posible como una segunda entrada.

Aquí está una descripción paso a paso de cómo usar la búsqueda binaria para jugar el juego de adivinar:

* Sea min = 1min=1m, i, n, equals, 1 y max = nmax=nm, a, x, equals, n.
* Adivina el promedio de maxmaxm, a, x y minminm, i, n, redondeado hacia abajo de modo que sea un entero.
* Si adivinaste el número, detente. ¡Lo encontraste!
* Si el intento fue demasiado bajo, haz que minminm, i, n sea uno más grande que el intento.
* Si el intento fue demasiado alto, haz que maxmaxm, a, x sea uno menos que el intento.
* Regresa al paso dos.

Podríamos hacer que la descripción fuera todavía más precisa al describir claramente las entradas y las salidas del algoritmo y al clarificar qué queremos decir con instrucciones como "adivina un número" y "detente". Pero esto es suficiente detalle por ahora.

Algoritmo.

Se toma el elemento central y se divide el array en dos:

{1,2,3,4}-5-{6,7,8,9}

Como el elemento buscado (3) es menor que el central (5), debe estar en el primer subarray: {1,2,3,4}

Se vuelve a dividir el array en dos:

{1}-2-{3,4}

Como el elemento buscado es mayor que el central, debe estar en el segundo subarray: {3,4}

Se vuelve a dividir en dos:

{}-3-{4}

Como el elemento buscado coincide con el central, lo hemos encontrado.

Si al final de la búsqueda todavía no lo hemos encontrado, y el subaray a dividir está vacío {}, el elemento no se encuentra en el array. La implementación sería:

**Ventajas**

* Se puede aplicar tanto a datos en listas lineales como en árboles binarios de búsqueda.
* Es el método más eficiente para encontrar elementos en un arreglo ordenado.

**Desventajas**

* Este método funciona solamente con arreglos ordenados, por lo cual si nos encontramos con arreglos que no están en orden, este método, no nos ayudaría en nada.

**6.3 - Búsqueda por Funciones Hash**

Concepto/Descripción

La función hash es la que se va a encargar de transformar las claves en direcciones de la tabla. Se puede definir la “función ideal” como aquella que distribuya a todos los elementos lo más uniformemente posible sobre la gama de valores índice, es decir, si tenemos una tabla que puede almacenar N items, entonces requerimos de una función que transforme claves a enteros en el rango [0, N-1], que la salida de la función tenga aproximadamente la misma probabilidad para cada entero, y que la distribución de las claves no esté ligada a patrón alguno. La función de Hash que se seleccione debe ser calculable de modo eficiente, es decir, estar compuesta de un número reducido de operaciones aritméticas básicas.

La implementación de la función hash depende del tipo de clave. No va a ser la misma si la clave es un entero, un real o una cadena.

Dentro de las funciones más comunes para la implementación de hashing se encuentran:

* Función Módulo: Se toma el resto de la división entre la clave y el número de N de items. Estadísticamente se puede verificar que, para una mayor uniformidad en la distribución, N debe ser un número primo, o al menos que sea divisible por pocos números. H(k) = (K mod N).
* Función Cuadrado: Se eleva al cuadrado la clave y se toman los dígitos centrales como dirección. El número de dígitos a tomar es determinado por el rango del índice (por ejemplo, si tengo 100 índices, tomo solo 2 dígitos). H(k) = dig\_centrales(k^2).
* Sumatoria de valores ASCII: Se toma el resto entre la sumatoria de los valores ASCII de la cadena de caracteres y el número N (tamaño de la tabla). Como char es un valor entero que es como mucho 127, para valores de N grandes, se necesita que las claves sean de una longitud considerable, de otra manera la distribución no va a ser uniforme.
* Plegamiento: consiste en dividir el número en diferentes partes, y operar con ellas (normalmente con suma o multiplicación). También se produce una colisión. Por ejemplo, si se dividen los números de 8 cifras en 3, 3 y 2 cifras y se suman, dará otro número de tres cifras (y si no, se recogen las tres últimas cifras): 13000000 --> 130=130+000+00.

**Funcionamiento**

Aunque alguna otra técnica pueda desempeñarse mejor en situaciones particulares, la técnica del residuo de la división proporciona generalmente la mejor distribución. Ninguna función hash se desempeña siempre mejor que las otras. El método del medio del cuadrado puede aplicarse en archivos con factores de cargas bastantes bajas para dar generalmente un buen desempeño. El método de pliegues puede ser la técnica más fácil de calcular pero produce resultados bastante erráticos, a menos que la longitud de la llave sea aproximadamente igual a la longitud de la dirección. Si la distribución de los valores de llaves no es conocida, entonces el método del residuo de la división es preferible. Note que el hashing puede ser aplicado a llaves no numéricas. Las posiciones de ordenamiento de secuencia de los caracteres en un valor de llave pueden ser utilizadas como sus equivalentes "numéricos". Alternativamente, el algoritmo hash actúa sobre las representaciones binarias de los caracteres.

**Algoritmo**

Para formar un conjunto H de funciones de Hash se llevan a cabo los siguientes pasos:

Paso 1

Seleccionar el tamaño de la tabla m tal que se primó.

Paso 2

Descomponer la llave x en r +1 bytes esto es x = < x0,x1,x2….xr>, donde Xi pertenece {0,1,2,… m-1}

Equivalente a escribir x en base m.

Paso 3

Sea a = <a0,a1,….ar> una secuencia de r + 1 elementos seleccionados aleatoriamente tal que al perteneciente a {0,1,2,…m-1} hay m^r+1 posibles secuencias

Paso 4

Definir a la función hash 

Paso 5

El conjunto H de funciones hash es: 

**Ventajas**

* Se pueden usar los valores naturales de la llave, puesto que se traducen internamente a direcciones fáciles de localizar.
* Se logra independencia lógica y física, debido a que los valores de las llaves son independientes del espacio de direcciones.
* No se requiere almacenamiento adicional para los índices.

**Desventajas**

* No pueden usarse registros de longitud variable
* El archivo no está clasificado
* No permite llaves repetidas
* Solo permite acceso por una sola llave

**Tabla comparativa**

**Capitales**

| Método de búsqueda | Complejidad temporal | Complejidad espacial | Numero de pasadas | Numero de comparaciones | Numero de intercambios |
| --- | --- | --- | --- | --- | --- |
| Secuencial | 19.343954 | 13736 | 1 | 30 | 0 |
| Binaria (Quicksort) | 12.378755 | 13944 | 41 | 132 | 51 |

**Números de control**

| Método de búsqueda | Complejidad temporal | Complejidad espacial | Numero de pasadas | Numero de comparaciones | Numero de intercambios |
| --- | --- | --- | --- | --- | --- |
| Secuencial | 23.071731 | 12736 | 1 | 39 | 0 |
| Binaria (Quicksort) | 14.844015 | 12752 | 75 | 265 | 108 |
| Hash | 21.867020 | 13168 | 2 | 29 | 50 |

**Conclusiones**

Los métodos de búsqueda juegan un papel muy importante en la estructura de los datos, cada método tiene sus ventajas y desventajas, que son diferentes a la cantidad de datos a procesar, por lo que sepas utilizar herramientas de comparación y podrás hacer grandes cálculos. Big O notation usa símbolos para comprender la mejor situación para cada situación. Ayuda a buscar una gran cantidad de registros con un modelo de eficiencia en el momento adecuado. A través de su tecnología, podemos poner listas en otras listas y luego ordenarlas, o comparar pares clave-valor e intercambiarlos si no están en la posición correcta.

**Bibliografías**

<http://artemisa.unicauca.edu.co/~nediaz/EDDI/cap02.htm>

<http://odin.fi-b.unam.mx/salac/practicasEDAII/eda2_p5.pdf>

<https://www.geeksforgeeks.org/searching-algorithms/>

<https://uneginginf05.es.tl/M-e2-todo-de-Busqueda-Secuencial-y-Binaria.htm>