Tema 2: Algoritmos de búsqueda

Objetivo: El alumno aplicará el método de búsqueda apropiado a conjuntos de datos residentes tanto en memoria principal como en la memoria secundaria. Diseñará y aplicará algoritmos.

2.1 Generalidades

- El cómputo actual y la facilidad de acceso a internet, han hecho que grandes cantidades de información sean accesibles a prácticamente cualquier persona.
- La capacidad de realizar búsquedas eficientes en tales cantidades de información es fundamental para aprovechar la información de manera eficiente.

2.2 Definición de la operación de búsqueda

- La búsqueda es la operación para recuperar datos almacenados con anterioridad.
- Al hablar de búsqueda, se asume que existe algún elemento (lista, archivo, arreglo, etc.) que contiene datos (llaves) de un tipo en particular y que puede hacerse referencia a alguna clave de éste, utilizando un índice posicional.
- También se asume que la entidad donde se realiza la búsqueda no es vacía.

2.2 Definición de la operación de búsqueda

- Existen 2 categorías fundamentales de búsqueda
  - Búsqueda interna.
  - Búsqueda externa.
- Al igual que en la clasificación de ordenamiento, el tipo de búsqueda se refiere al tipo de memoria donde se va a realizar.

2.3 Búsqueda por comparación de llaves

- Consiste en la realización de la búsqueda comparando los valores de la lista contra el valor deseado.
- Existen 2 algoritmos principales para este tipo de búsquedas

2.3.1 Búsqueda Lineal

- También conocida como búsqueda secuencial, se utiliza cuando la lista que se va a procesar está desordenada, la única opción es revisar secuencialmente toda la lista.
- Este es el algoritmo más sencillo, no es muy eficiente pero garantiza que su resultado es exitoso para cualquier lista.

2.3.1 Búsqueda Lineal

- La tarea importante de un algoritmo de búsqueda es identificar la ubicación de la llave dentro de la lista, por tanto, el algoritmo devolverá el índice en el cual se encuentra el valor buscado.
- Si el valor objetivo no se encuentra en la lista es común que el algoritmo devuelva un valor que se encuentra fuera del rango de los elementos.

2.3.1 Búsqueda Lineal

```
busquedaLineal (lista, x)
    n ← longitud (lista)
    para i ← 1 hasta n
        si x = lista [i]
            regresar i
    regresar −1
```

2.3.2 Búsqueda Binaria

- Consiste en dividir el intervalo de búsqueda en dos partes, comparando el elemento buscado con el que ocupa la posición central en el arreglo.
- Si no son iguales, se redefinen los extremos del intervalo, según el elemento sea mayor o menor que el elemento buscado.
- Éste método funciona exclusivamente con arreglos ordenados.

2.3.2 Búsqueda Binaria

```
busquedaBinaria (lista, x)
    izq ← 1
    der ← longitud (lista)
    mientras izq ≤ der
        medio ← (der + izq) /2
        si x = lista [medio]
            regresar medio
        otro si x < lista [medio]
            der ← medio − 1
        otro si x > lista [medio]
            izq ← medio + 1
    regresar −1
```

2.4 Búsqueda por transformación de llaves

- Los dos métodos analizados anteriormente permiten encontrar un elemento en un arreglo.
- En ambos casos el tiempo de búsqueda es proporcional a su número de elementos.
- Si bien el método de búsqueda binaria es más eficiente que el secuencial, existe la restricción de que el arreglo debe estar ordenado.

2.4 Búsqueda por transformación de llaves

- El método de transformación de claves permite aumentar la velocidad de búsqueda sin necesidad de tener los elementos ordenados.
- El tiempo de búsqueda es independiente del número de componentes del arreglo
- Para un conjunto de datos, cada uno de ellos se identifica con una clave mediante la cual se puede localizar el dato en forma directa

2.4.1 Funciones Hash

- Una función Hash es una función que convierte una clave dada un una dirección dentro del arreglo.

$$H(clave) = dirección$$

- La función hash (H) aplicada a la clave genera un índice, el cual funciona para determinar la posición en un arreglo/lista, que a su vez, permite acceder directamente al elemento.
- El caso mas trivial se presenta cuando las claves son números enteros consecutivos.

2.4.1 Funciones Hash

- Idealmente, se desea obtener una estructura de datos en la que las operaciones de insertar y buscar sean O(1).
- Se busca diseñar un contenedor que será utilizado para almacenar elementos de un conjunto K, en este contexto los elementos de K se llaman llaves (keys)

2.4.1 Funciones Hash

- La estrategia general es almacenar las llaves en un arreglo. La posición en la cual se debe almacenar está dada por la función hash aplicada a la llave.
- Las funciones hash deben ser simples de calcular y deben asignar direcciones de la manera más uniforme posible.

2.4.1.1 Función hash por módulo

- También llamada “por división” consiste en tomar el residuo de la división de la clave entre un número fijo para todas las claves.
- Dicho número puede estar determinado por el número de componentes del arreglo, o puede ser un número basado en cierto criterio.

$$h(x) = |x| \mod M$$

2.4.1.1 Función Hash por módulo

key	hash ($M = 100$)	key	hash ($M = 100$)
212	12	612	12
618	18	606	6
302	2	772	72
940	40	510	10
702	2	423	23
704	4	650	50

2.4.1.1 Función Hash por módulo

key	hash ($M = 97$)	key	hash ($M = 97$)
212	18	612	30
618	36	606	24
302	11	772	93
940	67	510	25
702	23	423	35
704	25	650	68

2.4.1.2 Función Hash cuadrado

- La operación de la división es bastante lenta comparada con otro tipo de operaciones en la computadora. Si se evita el uso de la división, se puede mejorar el tiempo de ejecución de una función hash.
- La función hash cuadrado consiste en elevar al cuadrado la clave y tomar los dígitos centrales como dirección.

2.4.1.2 Función Hash cuadrado

- El número de dígitos que se debe considerar se encuentra determinado por el rango del índice.
- Sea K la clave del dato a buscar, la función hash cuadrado queda definida así:

$$H(k) = dígitosCentrales(k^2) + 1$$

2.4.1.2 Función Hash cuadrado

- Ejemplo

Sea N = 100 el tamaño del arreglo, y sus direcciones los números comprendidos entre 1 y 100. Sean K1=7259 y K2=9359 dos claves a las que se deben asignar posiciones en el arreglo. Se aplica la fórmula anterior para calcular las direcciones correspondientes a K1, K2.

$$H(K_1) = \text{dígitos\_centrales } (52\ 693\ 081) + 1 = 94$$
$$H(K_2) = \text{dígitos\_centrales } (87\ 590\ 881) + 1 = 91$$

2.4.1.3 Función hash por plegamiento

- Consiste en dividir la clave en partes tomando igual número de dígitos, aunque la última parte puede tener menos.
- Operar con ellas asignando como dirección los dígitos menos significativos.
- La operación entre las secciones se puede realizar por medio de sumas o multiplicaciones.

2.4.1.3 Función hash por plegamiento

- Sea K la clave del dato a buscar. K está formada por los dígitos, d1,d2,…,dn. La función hash por plegamiento queda definida por la siguiente fórmula.

$$H(K) = ((d_1 \ldots d_i) + (d_{i+1} \ldots d_j) + \ldots + (d_1 \ldots d_n)) + 1$$

Tarea

- Buscar en qué consisten las siguientes funciones hash.
  - Por truncamiento
  - Método de Fibonacci
  - 2 funciones hash que no se haya visto en clase

  Describir en qué consisten los métodos e incluir un ejemplo.

2.4.2 Colisiones

- La elección de un método adecuado para resolver colisiones es tan importante como la elección de una buena función hash.
- Normalmente, cualquiera que sea el método elegido resulta costoso tratar las colisiones.
- Se debe encontrar una función que ofrezca la mayor uniformidad en la distribución de claves
- Los métodos más utilizados para resolver las colisiones son…

2.4.2.1 Reasignación

- Para la reasignación de elementos, existen sub-métodos.
- El método de **prueba lineal** consiste en que una vez que se detecta una colisión, se recorre el arreglo secuencialmente a partir del punto de colisión.
- La desventaja es que puede haber un fuerte agrupamiento alrededor de ciertas claves, mientras que otras zonas del arreglo pueden permanecer vacías

2.4.2.1 Reasignación

Prueba Lineal

$$D = H(K)$$

$$D' = D + 1$$

$$D'' = D' + 1$$

$$D''' = D'' + 1$$

2.4.2.1 Reasignación

- El método de **prueba cuadrática** es similar al anterior. La diferencia consiste en que en prueba cuadrática, las direcciones alternativas se generan como D+1, D+2, D+4, D+8 etc.

$$D = H(K)$$

$$D' = D + 2$$

$$D'' = D + 4$$

$$D''' = D + 8$$

…

2.4.2.1 Reasignación

- El método de **doble dirección Hash** consiste en que una vez que se detecta la colisión, se genera otra dirección aplicando otra función hash a la dirección previamente obtenida

$$D = H(K)$$

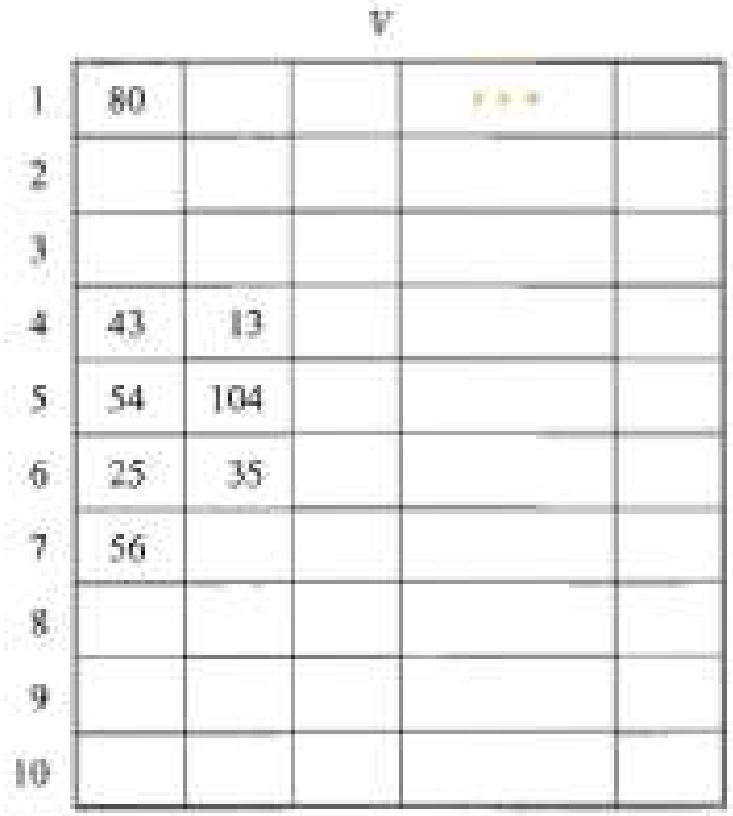
$$D' = H_2(D)$$

$$D'' = H_3(D')$$

2.4.2.2 Arreglos anidados

- El método de arreglos anidados consiste en que cada elemento del arreglo tenga otro arreglo en el cual se almacenen los elementos que colisionan.
- Al trabajar con arreglos se depende del espacio que haya sido asignado a estos.

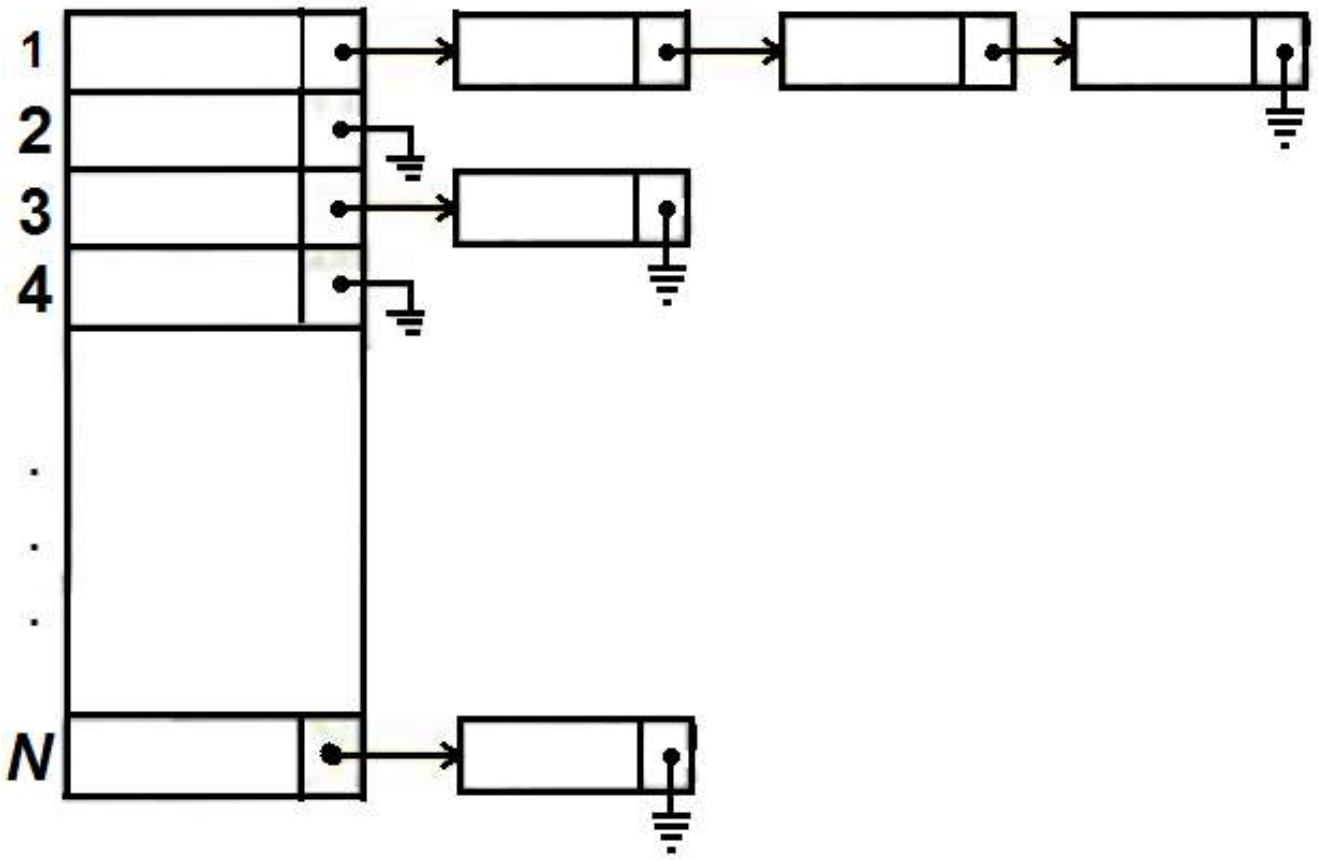
2.4.2.2 Arreglos anidados



2.4.2.3 Encadenamiento

- El método de encadenamiento consiste en que cada elemento del arreglo tenga un apuntador a una lista ligada la cual se irá generando y almacenará los valores que colisionan
- Este método es el más eficiente debido al dinamismo propio de las listas.
- Cualquiera que sea el número de colisiones que se presenten se pueden resolver sin inconvenientes.

2.4.2.3 Encadenamiento



2.4.3 Hash

- **Compromiso espacio tiempo**

Los métodos hash son un buen ejemplo del compromiso espacio-tiempo

Si no existiera límite en la memoria, se podría buscar cualquier dato con un solo acceso usando la llave como dirección de memoria

Si no existiera una restricción en el tiempo se necesitaría muy poca cantidad de memoria adicional utilizando búsqueda secuencial