Unidad 6. Métodos de búsqueda

6.1 Métodos de búsqueda

- 6.1 Secuencial
- 6.2 Binaria
- 6.3 Hashing

6.1 Búsqueda secuencial

Es el método más simple para encontrar un elemento dentro de una estructura de datos, como una lista o un archivo en SSD.

Se basa en revisar cada elemento uno a uno hasta encontrar el valor deseado o recorrer todos los datos sin encontrarlo.

6.1 Búsqueda secuencial

Procedimiento de la búsqueda secuencial

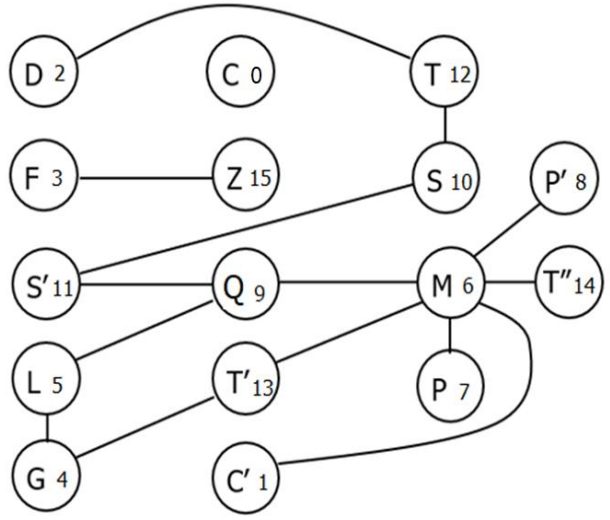
1. Se lee el primer renglón del archivo.
2. Se compara el valor buscado con el valor leído.
   - Si coincide termina la búsqueda y si es necesario se devuelve su posición.
   - Si no hay coincidencia se lee otro valor y se repite el paso 2.
3. Cuando ya no haya datos por leer termina la búsqueda.
4. Si no se encontró el valor buscado, se debe informar.

6.1 Búsqueda secuencial

Ejemplo

- El Archivo ***NombresNodos.txt*** contiene los nombres de las ciudades del grafo de abajo.
- El Archivo ***ArcosAutopistas.txt*** contiene los arcos (como es un grafo no dirigido, cada arco está en los 2 sentidos).
- Hay que escribir un programa para encontrar el grado (número de autopistas que inciden) de cierta ciudad.

C=Canatlán	0
C′=Cuernavaca	1
D=Durango	2
F=Fresnillo	3
G=Guadalajara	4
L=León	5
M=México	6
P=Puebla	7
P′=Pachuca	8
Q=Querétaro	9
S=Saltillo	10
S′=San Luis Potosí	11
T=Torreón	12
T′=Toluca	13
T″=Tulancingo	14
Z=Zacatecas	15



NombresNodo...

Archivo Edición Formato Ver Ayuda

```
Canatlan
Cuernavaca
Durango
Fresnillo
Guadalajara
Leon
Mexico
Puebla
Pachuca
Queretaro
Saltillo
San Luis Potosi
Torreon
Toluca
Tulancingo
Zacatecas
```

Windows (CRLF) UTF-8

Archivo Edición

```
1 6
6 1
2 12
12 2
3 15
15 3
4 5
5 4
4 13
13 4
5 9
9 5
6 9
9 6
6 7
7 6
6 8
8 6
6 14
14 6
9 11
11 9
10 11
11 10
10 12
12 10
6 13
13 6
```

UTF-8

6.1 Búsqueda secuencial

Búsqueda secuencial para obtener el identificador de la Ciudad

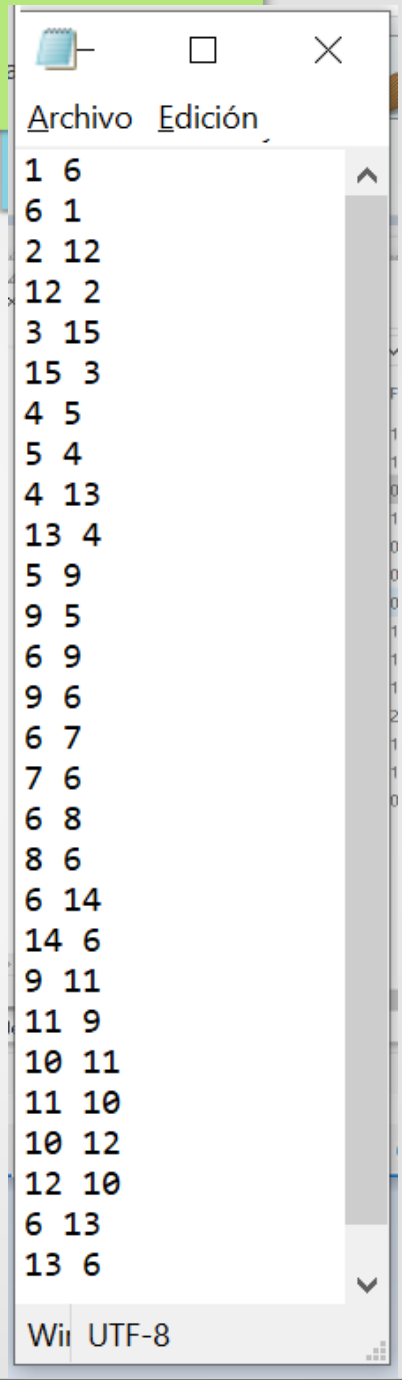
```
cout << "----------------------------------------"<<endl;
cout << "Escriba el nombre de la ciudad a buscar: ";
getline(cin, NombreCiudad);
PosicionCiudad = 0;
bool CiudadEncontrada=false;
while (getline(archivo, NombreNodo)) {
    if (NombreNodo==NombreCiudad){
        CiudadEncontrada=true;
        break;
    }
    PosicionCiudad++;
}
if (!CiudadEncontrada)
    cout << "Ciudad no se encuentra" << endl;
```

Canatlán	0
Cuernavaca	1
Durango	2
Fresnillo	3
Guadalajara	4
León	5
México	6
Puebla	7
Pachuca	8
Querétaro	9
Saltillo	10
San Luis Potosí	11
Torreón	12
Toluca	13
Tulancingo	14
Zacatecas	15

6.1 Búsqueda secuencial

Búsqueda secuencial para obtener el grado de un nodo (número de autopistas que inciden en la ciudad)

```
int Grado=0;
while (getline(archivo2, Linea)) {
    sscanf(Linea.c_str(), "%d %d", &i, &j);
    if (i==PosicionCiudad)
        Grado++;
}
cout << "Numero de Carreteras que inciden en "
     << NombreCiudad << ": " << Grado << endl;
```



```
#include <iostream>
#include <fstream>
#include <conio.h>
using namespace std;
int main(){
    char Opcion; int NumeroNodos,i,j; string NombreNodo, Linea;
    do {
        cout << "5.- Consultar Numero de Autopistas por ciudad\n"; cout << "0.- Terminar\n\n"; cout << "Opcion: ";Opcion = getch();
        cout << Opcion; cout << endl;
        switch (Opcion){
              case '5':{
                  string NombreCiudad; int PosicionCiudad; ifstream archivo("NombresNodos.txt");
                  if(!archivo){
                            cout<<"No existe o no se pudo abrir el archivo NombresNodos.txt"<<endl;
                            return 0;
                  }
                  else{
                            cout << "------------------------------------------------------------"<<endl;
                            cout << "Escriba el nombre de la ciudad a buscar: "; getline(cin, NombreCiudad);
                            PosicionCiudad = 0; bool CiudadEncontrada=false;
                            while (getline(archivo, NombreNodo)) {
                                if (NombreNodo==NombreCiudad){
                                          CiudadEncontrada=true;
                                          break;
                                }
                                PosicionCiudad++;
                            }
                            if (!CiudadEncontrada){
                                cout << "----------------------\n";
                                cout << "Ciudad no se encuentra\n";
                                cout << "----------------------\n"; break;
                            }
                  };
                  archivo.close();
                  ifstream archivo2("Arcos-Autopistas.txt");
                  if(!archivo2){
                            cout<<"No existe o no se pudo abrir el archivo Arcos.txt"<<endl;
                            return 0;
                  }
                  else{
                            int Grado=0;
                            while (getline(archivo2, Linea)) {
                                sscanf(Linea.c_str(), "%d %d", &i, &j);
                                if (i==PosicionCiudad)
                                          Grado++;
                            }
                            cout << "Numero de Carreteras que inciden en " << NombreCiudad << ": " << Grado << endl;
                  };
                  archivo.close();
                  cout << "------------------------------------------------"<<endl;
              }
        }
    } while (Opcion!='0');
    return 0;
}
```

Este programa es un complemento a los programas que se escribieron para Grafos en la unidad 4

6.1 Búsqueda secuencial

Características de la búsqueda secuencial

1. Simple de implementar y comprender.
2. No requiere que la lista esté ordenada.
3. Se puede usar para archivos con cualquier cantidad y tipos de datos.
4. Aunque una aplicación natural es archivos de texto como los del ejemplo o para archivos *.CSV*, se usa ampliamente con archivos binarios o tablas de una base de datos.
5. Puede ser un proceso lento si el numero de datos es muy grande, porque revisa los elementos uno por uno.

6.2 Búsqueda Binaria

La Búsqueda Binaria es un algoritmo **eficiente** para encontrar el valor de una llave dentro de un **archivo ordenado**.

Consiste en dividir repetidamente el espacio del archivo a la mitad, hasta:

- Llegar a un subarchivo de tamaño 1 y encontrar el valor buscado.
- Llegar a un subarchivo de tamaño nulo con lo que se determina que no se ha encontrado el valor.

6.2 Búsqueda Binaria

El archivo debe estar ordenado con base en la llave (en el ejemplo mostrado, el nombre de la ciudad) para que se pueda aplicar el procedimiento de Búsqueda Binaria.

La Búsqueda Binaria está diseñada para archivos que sufren pocos cambios durante el tiempo ya que si se añade un nuevo registro al archivo, tendrá que ordenarse de nuevo en forma completa.

Ciudad	País	Población
Beijing	China	22,596,500
Cairo	Egipto	23,074,200
Chihuahua	México	1,153,540
Chongqing	China	18,171,200
Delhi	India	34,665,600
Dhaka	Bangladesh	24,652,900
Durango	México	686,982
Estanbul	Turquía	16,236,700
Guadalajara	México	5,578,580
Karachi	Paquistán	18,076,800
Kinshasa	Congo	17,778,500
Laos	Nigeria	17,156,400
México	México	22,752,400
Monterrey	México	5,272,360
Mumbai	India	22,089,000
Osaka	Japón	18,921,600
Sao Paulo	Brasil	22,990,000
Shanghai	China	30,482,100
Tokio	Japón	37,036,200

6.2 Búsqueda Binaria

Procedimiento

1. **Se asigna el segmento de búsqueda inicial**:
   - Se establecen dos variables: *izquierda* para el índice del primer registro del archivo y *derecha* para el índice del último.
2. **Se usa una variable para el cálculo de la mitad del segmento de búsqueda**:
   - mitad = int (izquierda + derecha) / 2
3. **Comparación con el elemento buscado**:
   - Si el valor de la mitad es el que buscábamos, o izquierda>derecha la búsqueda termina.
   - Si el valor buscado es mayor que el de la mitad, se modifica el límite izquierdo: ***izquierda = mitad + 1***.
   - Si el valor buscado es menor que el de la mitad, se modifica el límite derecho: ***derecha = mitad - 1***.
   - Se calcula de nuevo: **mitad = int (izquierda + derecha) / 2**
4. **Ciclo**: Se repite el punto 3.

6.2 Búsqueda Binaria

0	Beijing	China	22,596,500
1	Cairo	Egipto	23,074,200
2	Chihuahua	México	1,153,540
3	Chongqing	China	18,171,200
4	Delhi	India	34,665,600
5	Dhaka	Bangladesh	24,652,900
6	Durango	México	686,982
7	Estanbul	Turquía	16,236,700
8	Guadalajara	México	5,578,580
9	Karachi	Paquistán	18,076,800
10	Kinshasa	Congo	17,778,500
11	Laos	Nigeria	17,156,400
12	México	México	22,752,400
13	Monterrey	México	5,272,360
14	Mumbai	India	22,089,000
15	Osaka	Japón	18,921,600
16	Sao Paulo	Brasil	22,990,000
17	Shanghai	China	30,482,100
18	Tokio	Japón	37,036,200

Buscar Durango.

1. izquierda=0, derecha=18, mitad=9.
2. izquierda=0, derecha=8, mitad=4.
3. izquierda=5, derecha=8, mitad=6.
4. Se encontró y se informan los datos completos:
   - País: México
   - Población: 686,982

Buscar Puebla.

1. izquierda=0, derecha=18, mitad=9.
2. izquierda=10, derecha=18, mitad=14.
3. izquierda=15, derecha=18, mitad=16.
4. izquierda=15, derecha=15, mitad=15.
5. izquierda=16, derecha=15.
6. No se encontró.

6.2 Búsqueda Binaria

Ventajas

- **Eficiencia:**
  - Tiene una orden de complejidad temporal de **O(log n)**, lo que significa que el número de comparaciones necesarias para encontrar un elemento es muy bajo comparado con búsqueda secuencial.
- **Simplicidad**:
  - El algoritmo es simple lo que permite entenderlo e implementarlo fácilmente.
- **Uso:**
  - Es muy útil para búsqueda en archivos muy grandes que cambian poco o nada.
  - Es la base de los índices de árboles B, B+ de Bases de datos.

6.2 Búsqueda Binaria

Desventajas

- **El archivo debe estar ordenado permanentemente:**
  - Si no está ordenado, tendrá que reordenarse y eso implica un tiempo de proceso que puede no ser conveniente.
- **Solo es aplicable en estructuras de acceso directo:**
  - Vectores o archivos binarios en los que cada elemento o registro tiene un índice asociado.
- **Innecesaria para archivos pequeños:**
  - Para conjuntos de datos pequeños, una búsqueda secuencial es suficiente.
- **Solo adecuado para llaves únicas**:
  - Si se necesita encontrar todos los elementos que cumplen una cierta condición, la búsqueda binaria no es la mejor opción, ya que solo encontrará el primer valor que cumpla con la condición .

6.2 Búsqueda Binaria

Ejercicio 1

Escriba un programa para implementar este algoritmo de manera sencilla usando un vector con valores numéricos generados al azar.

- El programa deberá indicar si se encuentra o no cierto valor y cuantas comparaciones hizo para encontrar el dato o para determinar que no se encuentra.

6.3 Hashing

Es una técnica muy eficiente para asignar el lugar en un archivo a nuevos datos y consultarlos posteriormente mediante un algoritmo que en el mejor caso es O(1) y en el peor O(n).

Se logra usando una tabla llamada Hash, que consiste en el espacio de un archivo para asignar idealmente **un lugar único** a cada valor de un conjunto de datos posibles.

6.3 Hashing

Conceptos

1. **Función hash**:
    - Convierte el valor de un identificador único (la llave) en un índice de un archivo (el archivo es la tabla Hash).
2. **Colisiones**:
    - Ocurren cuando la función Hash genera el mismo índice para dos diferentes llaves.
    - Una técnica común para resolver las colisiones es usar listas encadenadas para cada índice del archivo.
3. **Tabla Hash**:
    - Espacio de un archivo donde los registros se almacenan y recuperan utilizando la función Hash.

6.3 Hashing

Implementación

1. Tabla Hash:

- Cada índice del archivo es la cabeza de una lista encadenada para almacenar el primer registro almacenado en la posición correspondiente (de acuerdo a la llave) y las colisiones.
- Una posible función Hash sencilla:
  - Se suman los códigos ASCII de cada uno de los valores de la llave y al resultado final se le aplica el residuo contra el tamaño de la tabla Hash (espacio de un archivo destinado para la tabla Hash).

6.3 Hashing

1. Tabla Hash (continuación):

- Se recomienda que el tamaño sea un número primo.
- Si las llaves son los números de control de los alumnos activos en el tecno, el tamaño del archivo sería un número primo cercano a 6000, por ejemplo, 7001.
  - Los 7001 registros se deben crear como un espacio cerrado (los registros del archivo del 0 al 7000).
  - El espacio para las colisiones sería el espacio abierto a partir del registro 7001.

Se observa en la tabla de la derecha, que se producen colisiones si se aplica este algoritmo sencillo.

```
int FuncionHash(string Cadena, int TamanioTabla){
    long Suma=0;
    for(int i=0;i<Cadena.length();i++){
         Suma+=Cadena[i];
    }
    return Suma % TamanioTabla;
}
```

Espacio cerrado

Espacio abierto

0			
1			
	.....		
399	24041220	MARTINEZ ROSA MARIA	-1
400	24041221	RODRIGUEZ MARIA	-1
401			
402			
403			
404	23040452	GONZALEZ JOSÉ	-1
	.....		
414	24040893	LOPEZ ALICIA	7001
415			
416	25040876	ROMERO FRANCISCO	-1
	.....		
7000			
7001	25040883	GARCIA JOAQUIN	-1

6.3 Hashing

2. **Añadir**:
   - Al añadir un nuevo registro, se calcula el índice según la Función Hash y se añade a la lista encadenada que se encuentra en la posición del índice calculado.
     - La lista enlazada puede quedar ordenada o no, sin embargo, en teoría no debe ser necesario ya que en caso de que el número de colisiones sea grande, se debe cambiar la función Hash.
     - El primer nodo de la lista encadenada va a quedar en la posición calculada por la función Hash, los que siguen quedarán en el archivo luego del último índice de la tabla (Si es 7001 el tamaño de la tabla, los índices van de 0 a 7000, por lo que del índice 7001 en adelante se usará para las colisiones).

6.3 Hashing

3. **Consultar**

   - Para consultar, se calcula la posición de acuerdo a la función Hash y se recorre la lista encadenada (nodos que están en los índices posteriores al último de la tabla Hash, en este caso del 7001 en adelante) hasta encontrar la llave buscada o determinar que no existe.

Ejemplo algunas funciones Hash sencillas.

```
#include <iostream>
#include <cstring>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <cmath>
using namespace std;

int FuncionHash1(string Cadena, int TamanioTabla){
      long Suma=0;
      for(int i=0;i<Cadena.length();i++){
            //cout << Cadena[i] << " " << (int)Cadena[i] << endl;
            Suma+=Cadena[i];
      }
      return Suma % TamanioTabla;
}

int FuncionHash2(string Cadena, int TamanioTabla){
      long Suma=0;
      for(int i=0;i<Cadena.length();i++){
            //cout << Cadena[i] << " " << (int)(i+1)*Cadena[i];
            //cout << (i+1)*Cadena[i] << endl;
            Suma+=(i+1)*Cadena[i];
      }
      return Suma % TamanioTabla;
}
```

Ejemplo algunas funciones Hash sencillas.

```
int FuncionHash3(string Cadena, int TamanioTabla){
	long long Suma=0;
	for(int i=0;i<Cadena.length();i++){
		//cout << Cadena[i] << " " << (int)Cadena[i];
		//cout << (long long)pow(Cadena[i],(i+1))*Cadena[i] << endl;
		Suma+=(long long)pow(Cadena[i],(i+1))*Cadena[i];
	}
	return Suma % TamanioTabla;
}


int main() {
	do{
		string NumControl;
		cout << "Escriba un Numero de Control: ";cin >> NumControl;
		cout << "Indice segun Funcion 1: " << FuncionHash1(NumControl, 7001) << endl;
		cout << "Indice segun Funcion 2: " << FuncionHash2(NumControl, 7001) << endl;
		cout << "Indice segun Funcion 3: " << FuncionHash3(NumControl, 7001) << endl;
	}while(true);
	return 0;
}
```

6.3 Hashing

- Los algoritmos de las diapositivas anteriores son funciones Hash simples diseñadas con fines didácticos.
- Hay algoritmos muy robustos que se usan para diversas aplicaciones pero que podemos usar para búsqueda Hashing.
  - Por ejemplo los de la familia **SHA** (*Secure Hash Algorithm*).
  - *SHA-256,* por ejemplo, produce una cadena de 32 bytes.
  - Pueden tomarse algunos de los bytes iniciales, interpretarlos en sistema decimal y aplicar el residuo para la Función Hash.

6.3 Hashing

Ejercicio

Solicite a una de las herramientas de IA que le genere un programa en C++ para el problema planteado de almacenar una cadena de 8 dígitos correspondiente al numero de control en una *Tabla Hashing* manejando las colisiones con listas encadenadas.