Reflexión Actividad Act 5.1 - Implementación individual de operaciones sobre conjuntos

Héctor Robles Villarreal A01634105

Programación de estructuras de datos y algoritmos fundamentales

Carlos Augusto Ventura Molina

Grupo 11

Tecnológico de Monterrey Campus Guadalajara

Domingo 28 de Noviembre de 2021

Reflexión Actividad Act 5.1 - Implementación individual de operaciones sobre conjuntos

Una hash table es una estructura de datos muy usada en la que se pueden guardar llavse y varios datos asociados a esas llaves. Hace uso de una función llamada “función hash” que lo que hace es convertir las llaves a índices a través de varios métodos como el módulo o una elevación al cuadrado. Estos índices sirven para indicar en donde se van a guardar los datos asociados a esa llave dentro de una tabla de dispersión. Las tablas hash son muy usadas gracias a su baja complejidad computacional. Su eficiencia es increíble, ya que en algunos casos, tiene una complejidad de O(1) ya que simplemente se necesita el índice que genra la función hash para poder acceder a la tabla y encontrar esos valores asociados, sin tener que usar iteraciones. En el peor de los casos su complejidad llega a ser de O(n) ya que cuando hay colisiones, es decir, datos que tienen el mismo índice generado por la función hash, se tiene que hacer iteraciones para lidiar con estas.

Para esta actividad se implementaron dos funciones para crear una tabla hash y lidiar con sus colisiones. La primera fue una función que usaba el método de sondeo cuadrático, y la segunda ponía las colisiones en una lista encadenada.

Clase hash, constructor y función hash:

```
class Hash{
    public:
    // Constructor
    Hash(int);
    // Función hash
    int hashFunction(int);
    //Funcion para manejo de colisiones por sondeo cuadrático
    void sondeoCuadratico(std::pair<int,std::string>);
    //Función para el manejo de colisiones por encadenamiento
    void chain(std::pair <int, std::string>);
    static void printHashCuadratico(Hash);
    static void printChain(Hash);
    private:
        //Numero de filas que tendrá la tabla hash
        int numFilas;
        //Tabla hash
        std::vector<std::pair<int, std::string> > lista;
        std::vector<bool> banderas;
        std::vector<node*> listaChain;

};

Hash::Hash(int n){
    numFilas = n;
    std::pair <int, std::string> par(0," ");
    for (int h = 0; h < numFilas; h++){
        banderas.push_back(false);
        lista.push_back(par);
        nodo = new node();
        nodo->datos = par;
        nodo->sig = NULL;
        listaChain.push_back(nodo);
    }
}
```

Función de sondeo cuadrático:

```
void Hash::sondeoCuadratico(std::pair<int, std::string> elemento){
    int index = hashFunction(elemento.first);
    if (!banderas[index]){
        lista.at(index) = elemento;
        banderas.at(index) = true;
    }
    //Sondeo Cuadratico /O(n)
    else{
        for (int i = 0; i < numFilas; i++){
            int nuevoIndex = (index + i*i) % numFilas;
            if(!banderas[nuevoIndex]){
                lista.at(nuevoIndex) = elemento;
                banderas.at(nuevoIndex) = true;
                break;
            }
        }
    }
}
```

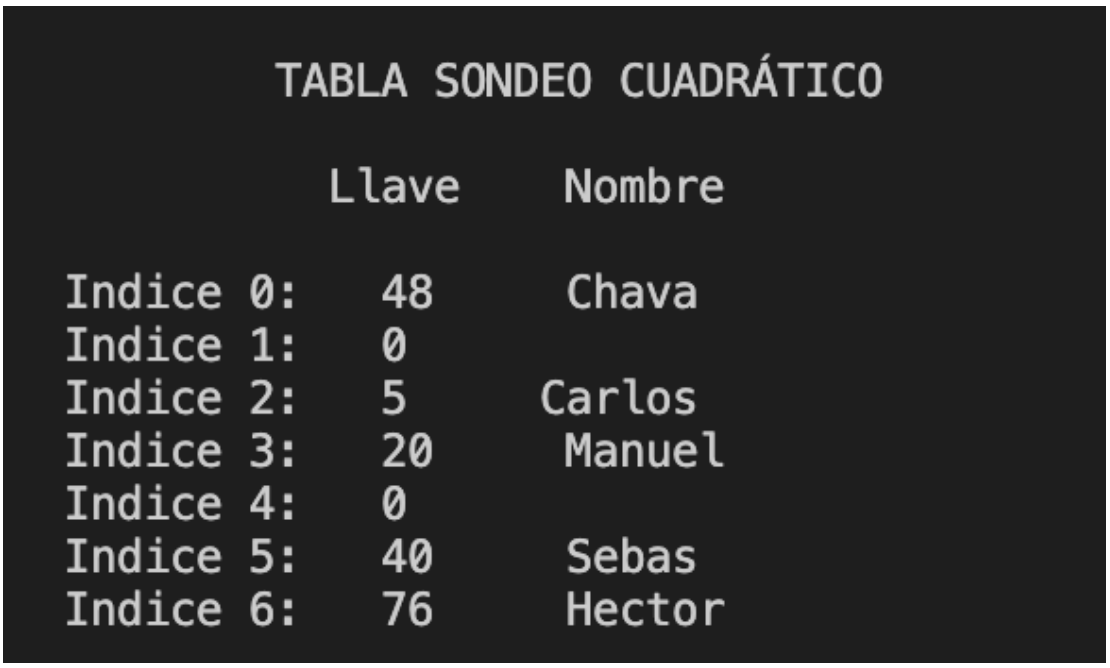
Esta función, como mencionado con anterioridad, lo que hace es que, en caso de que haya una colisión, genera un nuevo índice para la llave que tuvo la colisión, a través de el método de ((índice + n*n) % número de elementos). En el mejor de los casos, el segundo índice generado será un espacio libre en la tabla, sin embargo, hay ocasiones en las que no es así, y en ese segundo índice también hay una colisión, por lo que en cada iteración, n sube uno, por lo que empieza siendo 1, luego, 2 y así sucesivamente. La función sigue iterando y encuentra un espacio vacío en la tabla para la colision. Su complejidad es de O(n) ya que si hay una colision, tiene que iterar a través de toda la tabla para checar los índices. También puede que el segundo índice genere otra colision, por lo que se tiene que volver a iterar.

Implementación en main:

```
int main(){
    Hash tablaSC(7);

    pair<int, string> par1(76,"Hector");
    pair<int, string> par2(40,"Sebas");
    pair<int, string> par3(48,"Chava");
    pair<int, string> par4(5, "Carlos");
    pair<int, string> par5(20, "Manuel");

    tablaSC.sondeoCuadratico(par1);
    tablaSC.sondeoCuadratico(par2);
    tablaSC.sondeoCuadratico(par3);
    tablaSC.sondeoCuadratico(par4);
    tablaSC.sondeoCuadratico(par5);

    Hash::printHashCuadratico(tablaSC);



    Hash tablaChain(10);
```

Se genera primero una tabla hash vacía. Después, se generan unos valores de tipo pair, que contienen tanto la llave como un nombre asociado a esta. Estos se introducen en la tabla y se van checando las colisiones. En caso de que haya colisión, se busca un nuevo índice para esa tabla. Cuando acaba, imprime la tabla

Salida:



Algunos espacios quedan con 0 ya que ninguna llave generó ese índice a través de la función hash. Pero por ejemplo, aquí, se usa el método de módulo. Son 7 elementos por lo que la función hash usará elemento mod 7. El 20 y el 76, ambos dan 6 como residuo, por lo que ahí hay una colision. Manuel se agrega en el índice 3 porque hace el método comentado con anterioridad (6+ 2*2) % 7 dá como resultado 3.

Función de encadenamiento por colisión:

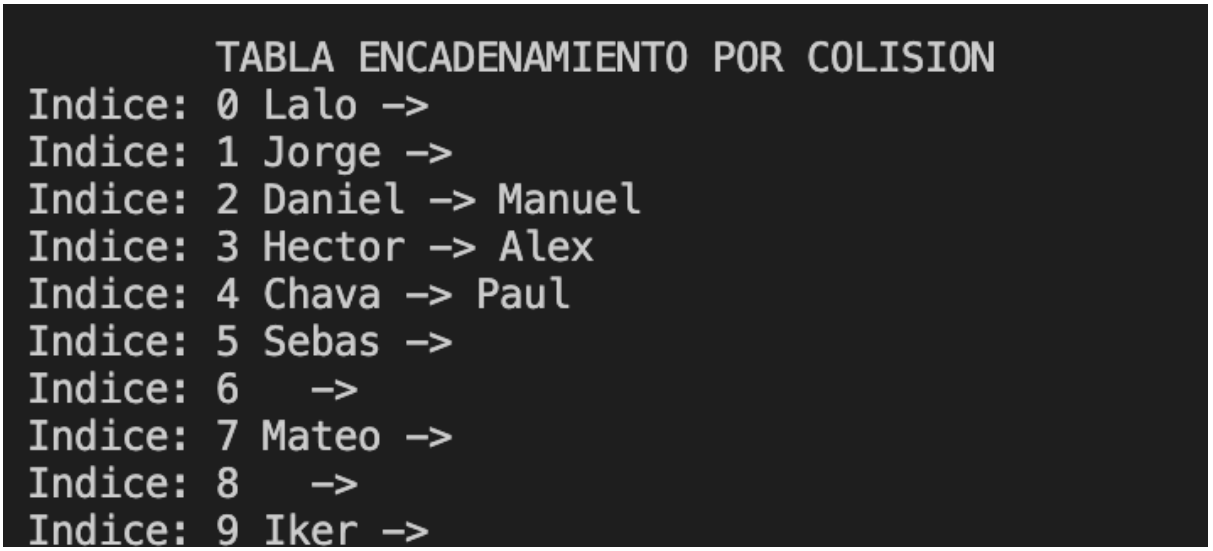
```
Hash tablaChain(10);

pair<int, string> parc0(0, "Lalo");
pair<int, string> parc1(1, "Jorge");
pair<int, string> parc2(2, "Daniel");
pair<int, string> parc3(3, "Hector");
pair<int, string> parc4(4, "Chava");
pair<int, string> parc5(5, "Sebas");
pair<int, string> parc6(2, "Manuel");
pair<int, string> parc7(3, "Alex");
pair<int, string> parc8(4, "Paul");
pair<int, string> parc9(9, "Iker");
pair<int, string> parc10(7, "Mateo");

tablaChain.chain(parc1);
tablaChain.chain(parc2);
tablaChain.chain(parc3);
tablaChain.chain(parc4);
tablaChain.chain(parc6);
tablaChain.chain(parc3);
tablaChain.chain(parc7);
tablaChain.chain(parc5);
tablaChain.chain(parc0);
tablaChain.chain(parc7);
tablaChain.chain(parc8);
tablaChain.chain(parc9);
tablaChain.chain(parc10);

Hash::printChain(tablaChain);
```

En esta función, lo que se hace es ir metiendo las colisiones dentro de una lista ligada. Cada fila de la tabla hash es una lista ligada a la cual se le pueden ir agregando elementos cada vez que surga una colision en ese índice.

Implementación en main:

Aquí también se crean varis pares para introducirlos en una tabla vacía de hash, en caso de colisiones, se genera la lista encadenada.

Salida:



Hay algunos índices que se quedan vacíos porque la función hash no generó esos índices.

Aquí se puede ver que hubo varias colisiones, y estas se agregaron a la lista encadenada de cada índice.