Tecnológico de Monterrey

Hash Table Casos prueba y complejidades

Programming of data structures

Inés Alejandro Garcia Mosqueda A00834571

Profesores: Luis Ricardo Peña Llamas
Jorge Gonzalez

21/11/2022

En la actividad se implementa una estructura de datos HashTable, donde se plantean dos métodos para manejar colisiones (chaining y quadratic probing)

Para esto se plantearon dos estructuras de HashTable con la diferencia de la implementación de manejo de colisiones (HashTable_Chain, HashTable_Quadratic)

```
class HashTable_Chain{
    int capacity;
    list<int> *table;
    private:
        int getPrime(int); //Complejidad O(n)
        bool checkPrime(int); //Complejidad O(sqrt(n))
        int hashFunction(int); //ComplejidadO(1)
    public:
        HashTable_Chain(){ //Complejidad O(1)
            this->capacity = 0;
            table = new list<int>[capacity];
        };
        HashTable_Chain(int n){ //Complejidad O(nsqrt(n))
            int size = getPrime(n);
            this->capacity = size;
            table = new list<int>[capacity];
        };
        ~HashTable_Chain(){
            delete[] table;
            cout << "\nHashTable_Chain deleted. " << endl;
        };
        void insertItem(int); //O(1)
        void deleteItem(int); //O(1)
        void displayHash();   //O(n+m)
        void fillHash(string); //O(n)
};
```

```
class HashTable_Quadratic{
    int capacity;
    int *table;
    private:
        int getPrime(int); //Complejidad O(n)
        bool checkPrime(int); //Complejidad O(sqrt(n))
        int hashFunction(int); //ComplejidadO(1)
        int quadratic(int,int);
    public:
        HashTable_Quadratic(){ //Complejidad O(1)
            this->capacity = 0;
            table = new int[capacity];
        };
        HashTable_Quadratic(int n){ //Complejidad O(nsqrt(n))
            int size = getPrime(n);
            this->capacity = size;
            table = new int[capacity];
        };
        ~HashTable_Quadratic()
        {
            delete[] table;
            cout << "\nHashTable_Quadratic deleted. " << endl;
        };
        void insertItem(int); //O(n)
        void deleteItem(int); //O(n)
        void displayHash(); //O(n+m)
        void fillHash(string); //O(n)
};
```

Donde la complejidad para las funciones de la implementación con Chain son:

int getPrime(int); //Complejidad O(n)

bool checkPrime(int); //Complejidad O(sqrt(n))

int hashFunction(int); //ComplejidadO(1)

HashTable_Chain(); //Complejidad O(1)

HashTable_Chain(int n) // Complejidad O(nsqrt(n))

void insertItem(int); //O(1)

void deleteItem(int); //O(1)

void displayHash(); //O(n+m)

void fillHash(string); //O(n)

Las complejidades para las funciones con Quadratic probing son:

int getPrime(int); //Complejidad O(n)

bool checkPrime(int); //Complejidad O(sqrt(n))

int hashFunction(int); //ComplejidadO(1)

int quadratic(int,int);

HashTable_Quadratic() //Complejidad O(1)

HashTable_Quadratic(int n) //Complejidad O(nsqrt(n))

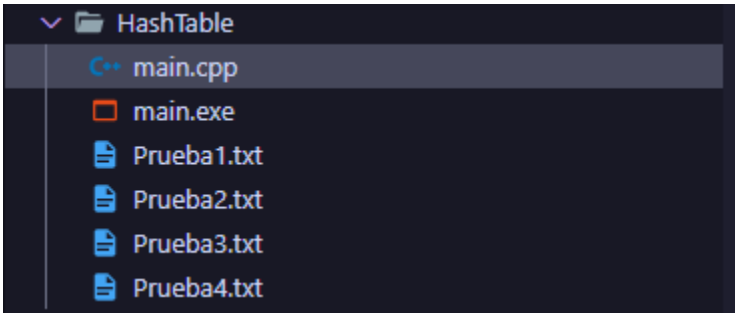
void insertItem(int); //O(n)

void deleteItem(int); //O(n)

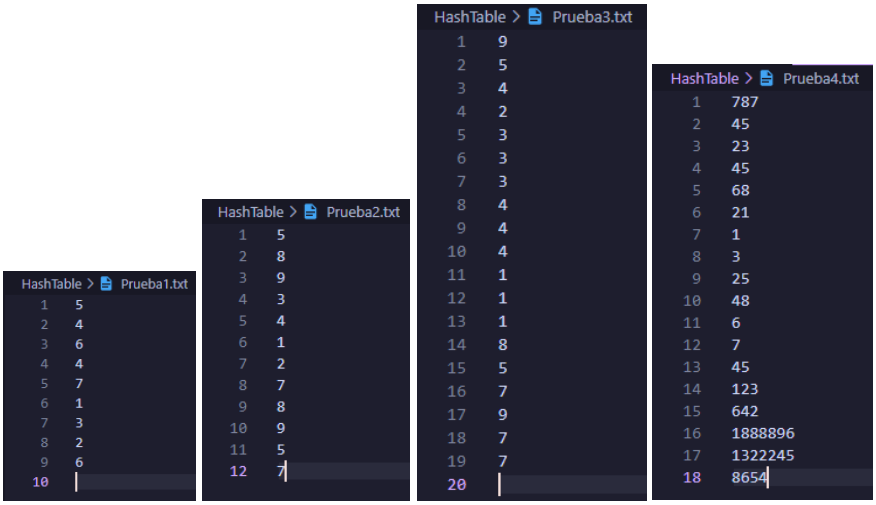
void displayHash(); //O(n+m)

void fillHash(string); //O(n)

Los casos de prueba se llevaron de la siguiente manera



Dentro de la carpeta existen 4 archivos de texto que contienen diferentes casos como prueba, ordenados con el número de prueba que se leen en la función main()




```int main(){

    const int ArchivosPrueba = 4;
    string arrPruebas[ArchivosPrueba];
    for (int i = 0; i < sizeof(arrPruebas)/ sizeof(arrPruebas[0]); i++)
    {
        string prueba = "Prueba" + to_string(i+1) + ".txt";
        arrPruebas[i] = prueba;
    }

    for (int i = 0; i < sizeof(arrPruebas)/ sizeof(arrPruebas[0]); i++)
    {
        if (i>0)
        cout<<"\n.....\n";

        cout << "\n\nTEST: " << arrPruebas[i] << endl;
        HashTable_Chain hash_chain;
        HashTable_Quadratic hash_quadratic;
        hash_chain.fillHash(arrPruebas[i]);
        hash_quadratic.fillHash(arrPruebas[i]);

        cout<<"\nChaining implementation\n";
        hash_chain.displayHash();
        cout<<"\nQuadratic probing implementation\n";
        hash_quadratic.displayHash();
        cout<<"\n.....\n";
    }

    return 0;
}```
```

Como resultado de los casos de prueba se obtienen los resultados esperados para ambas implementaciones en cada caso:

```
TEST: Prueba1.txt

Chaining implementation
table[0]
table[1] --> 1
table[2] --> 2
table[3] --> 3
table[4] --> 4 --> 4
table[5] --> 5
table[6] --> 6 --> 6
table[7] --> 7
table[8]
table[9]
table[10]

Quadratic probing implementation
table[0] --> 6
table[1] --> 1
table[2] --> 2
table[3] --> 3
table[4] --> 4
table[5] --> 5
table[6] --> 6
table[7] --> 7
table[8]
table[9] --> 4
table[10]

.........................

HashTable_Quadratic deleted.

HashTable_Chain deleted.
```

```
TEST: Prueba2.txt

Chaining implementation
table[0]
table[1] --> 1
table[2] --> 2
table[3] --> 3
table[4] --> 4
table[5] --> 5 --> 5
table[6]
table[7] --> 7 --> 7
table[8] --> 8 --> 8
table[9] --> 9 --> 9
table[10]
table[11]
table[12]

Quadratic probing implementation
table[0] --> 5
table[1] --> 1
table[2] --> 2
table[3] --> 7
table[4] --> 4
table[5] --> 5
table[6]
table[7] --> 7
table[8] --> 8
table[9] --> 9
table[10] --> 8
table[11] --> 9
table[12] --> 3

.........................

HashTable_Quadratic deleted.

HashTable_Chain deleted.
```

```
TEST: Prueba3.txt

Chaining implementation
table[0]
table[1] --> 1 --> 1 --> 1
table[2] --> 2
table[3] --> 3 --> 3 --> 3
table[4] --> 4 --> 4 --> 4 --> 4
table[5] --> 5 --> 5
table[6]
table[7] --> 7 --> 7 --> 7
table[8] --> 8
table[9] --> 9 --> 9
table[10]
table[11]
table[12]
table[13]
table[14]
table[15]
table[16]
table[17]
table[18]

Quadratic probing implementation
table[0] --> 2
table[1] --> 1
table[2] --> 3
table[3] --> 1
table[4] --> 4
table[5] --> 7
table[6] --> 4
table[7] --> 7
table[8] --> 7
table[9] --> 9
table[10] --> 8
table[11] --> 5
table[12] --> 3
table[13]
table[14] --> 5
table[15] --> 4
table[16] --> 1
table[17] --> 4
table[18] --> 3

.........................

HashTable_Quadratic deleted.

HashTable_Chain deleted.
```

```
TEST: Prueba4.txt

Chaining implementation
table[0]
table[1] --> 1
table[2] --> 21
table[3] --> 3
table[4] --> 23
table[5]
table[6] --> 6 --> 25
table[7] --> 45 --> 7 --> 45 --> 45
table[8] --> 787
table[9] --> 8654 --> 123
table[10] --> 48
table[11] --> 1888896 --> 68
table[12]
table[13]
table[14]
table[15] --> 642
table[16] --> 1322245
table[17]
table[18]

Quadratic probing implementation
table[0] --> 787
table[1] --> 1
table[2] --> 21
table[3] --> 3
table[4] --> 25
table[5] --> 7
table[6] --> 6
table[7] --> 45
table[8] --> 45
table[9] --> 8654
table[10] --> 48
table[11] --> 1888896
table[12] --> 68
table[13] --> 45
table[14]
table[15] --> 642
table[16] --> 1322245
table[17] --> 23
table[18] --> 123

.........................

HashTable_Quadratic deleted.

HashTable_Chain deleted.
```