Universidad de Concepción
Facultad de Ingeniería
Departamento de Ing. Informática y Cs
de la Computación

Profesor
Diego Seco

Laboratorio 6:

Estructuras de Datos (503220)

Estudiante: Nicolás Araya (2018448613)

1) ADTMap:

ADTMap.h > ADTMap

```h
#ifndef Map
#define Map
#include <iostream>
using namespace std;

class ADTMap{
    public:
        virtual void insert(pair<string, int>) = 0;
        virtual void erase(string) = 0;
        virtual int at(string) = 0;
        virtual int size() = 0;
        virtual bool empty() = 0;
};
#endif
```

2) MapB:

MapB.h > MapB > colisiones

```h
#include "ADTMap.h"
#include <vector>

using namespace std;

class MapB : public ADTMap{
    private:
        long long int colisiones, colisionesAt;
        int sz;
        int N;
        pair<string, int> null;
        vector<pair<string,int>> hash;
        void insertHash(pair<string,int>);
        void newHash();
    public:
        MapB(int);
        void insert(pair<string, int>);
        void erase(string);
        int at(string);
        int size();
        bool empty();
        void print();
        void printColisiones();
};
```

MapB.cpp > ...

```cpp
#include "MapB.h"
#include <iostream>

using namespace std;

MapB::MapB(int N){
    this->N=N;
    sz=0;
    null = {"", INT16_MIN};
    hash.assign(N+1, null);
    colisiones=0;
    colisionesAt=0;
}
```

```cpp
void MapB::insertHash(pair<string,int> par){
    int suma=0;
    for(int i = 0; i<par.first.size(); i++){
        suma += par.first[i];
    }
    int hashresult = suma%N;
    if(hash[hashresult] == null) hash[hashresult] = par;
    else{
        if(hash[hashresult].first == par.first) return;
        hashresult++;
        colisiones++;
        if(hashresult>N-1) hashresult = 0;
        while(hash[hashresult] != null ){
            hashresult++;
            colisiones++;
            if(hashresult>N-1) hashresult = 0;
        }
        hash[hashresult] = par;
    }

}

void MapB::newHash(){
    vector<pair<string, int>> aux;
    N = N*2;
    aux.assign(N+1, null);
    for(int i = 0; i<hash.size(); i++){
        if(hash[i]!=null){
            string s = hash[i].first;
            long long int suma=0;
            for(int i = 0; i<s.size(); i++){
                suma += s[i];
            }
            int hashresult = suma % N;
            if(aux.at(hashresult)!=null){
                hashresult++;
                if(hashresult>N-1) hashresult = 0;
                while(aux[hashresult]!=null){
                    hashresult++;
                    if(hashresult>N-1) hashresult = 0;
                }
            }
            aux[hashresult] = hash[i];
            }
    }
    hash = aux;
}
```

```
62  void MapB::insert(pair<string,int> par){
63      if(size()>=N/2) newHash();
64      insertHash(par);
65      sz++;
66
67  }
```

```
68  void MapB::erase(string s){
69      int suma=0;
70      for(int i = 0; i<s.size(); i++){
71          suma+=s[i];
72      }
73      int hashresult = suma%N;
74      if(hash[hashresult]!=null){
75          if(hash[hashresult].first==s) {
76              hash[hashresult]=null;
77              sz--;
78          }
79          else{
80              hashresult++;
81              if(hashresult>N-1) hashresult = 0;
82              while(hash[hashresult].first!=s){
83                  hashresult++;
84                  if(hashresult>N-1) hashresult=0;
85                  if(hashresult==suma%N) break;
86              }
87              if(hash[hashresult].first==s){
88                  hash[hashresult] = null;
89              }
90          }
91      }
92  }
```

```
93  int MapB::at(string s){
94      int suma=0;
95      for(int i = 0; i<s.size(); i++){
96          suma+=s[i];
97      }
98      int hashresult = suma%N;
99      if(hash[hashresult]!=null){
100         if(hash[hashresult].first==s) return hash[hashresult].second;
101         else{
102             hashresult++;
103             colisionesAt++;
104             if(hashresult>N-1) hashresult = 0;
105             while(hash[hashresult].first!=s){
106                 hashresult++;
107                 colisionesAt++;
108                 if(hashresult>N-1) hashresult=0;
109                 if(hashresult==suma%N) break;
110             }
111             if(hash[hashresult].first==s) return hash[hashresult].second;
112             else return INT16_MIN;
113         }
114     }
115 }
```

```cpp
int MapB::size(){
    return sz;
}
bool MapB::empty(){
    if(size()==0) return true;
    return false;
}

void MapB::print(){
    for(int i = 0; i<hash.size(); i++){
        if(hash[i]!=null) cout<<hash[i].first<<" "<<hash[i].second<<endl;
    }
}
void MapB::printColisiones(){
    cout<<"MapB"<<endl<<"Colisiones Insert: "<<colisiones<<endl<<"Colisiones at: "<<colisionesAt<<endl;
}
```

MapG:

```cpp
#include "ADTMap.h"
#include <vector>

using namespace std;

class MapG : public ADTMap{
    private:
        int z;
        int sz;
        int N;
        long long int colisiones, colisionesAt;
        vector<pair<string,int>> hash;
        pair<string, int> null;
        void insertHash(pair<string,int>);
        void newHash();

    public:
        MapG(int);
        void insert(pair<string, int>);
        void erase(string);
        int at(string);
        int size();
        bool empty();
        void print();
        void printColisiones();
};
```

MapG.cpp > MapG::printColisiones()

```cpp
#include "MapG.h"
#include <iostream>

using namespace std;

MapG::MapG(int N){
    this->N=N;
    sz=0;
    null = {"", INT16_MIN};
    hash.assign(N+1, null);
    z=33;
    colisiones=0;
    colisionesAt=0;
}

void MapG::insertHash(pair<string,int> par){
    long long int suma=0;
    for(int i = 0; i<par.first.size(); i++){
        suma = suma*z + par.first[i];
    }
    int hashresult = suma%N;
    if(hash[hashresult] == null) hash[hashresult] = par;
    else{
        if(hash[hashresult].first == par.first) return;
        hashresult++;
        colisiones++;
        if(hashresult>N-1) hashresult = 0;
        while(hash[hashresult] != null ){
            hashresult++;
            colisiones++;
            if(hashresult>N-1) hashresult = 0;
        }
        hash[hashresult] = par;
    }
}
```

```cpp
void MapG::newHash(){
    vector<pair<string, int>> aux;
    N = N*2;
    aux.assign(N+1, null);
    for(int i = 0; i<hash.size(); i++){
        if(hash[i]!=null){
            string s = hash[i].first;
            long long int suma=0;
            for(int i = 0; i<s.size(); i++){
                suma = suma*z + s[i];
            }
            int hashresult = suma % N;
            if(aux.at(hashresult)!=null){
                hashresult++;
                if(hashresult>N-1) hashresult = 0;
                while(aux[hashresult]!=null){
                    hashresult++;
                    if(hashresult>N-1) hashresult = 0;
                }
            }
            aux[hashresult] = hash[i];
            }
    }
    hash = aux;
}
```

```cpp
void MapG::insert(pair<string,int> par){
    if(size()>=N/2) newHash();
    insertHash(par);
    sz++;

}
```

```cpp
void MapG::erase(string s){
    long long int suma=0;
    for(int i = 0; i<s.size(); i++){
        suma = suma*z + s[i];
    }
    int hashresult = suma%N;
    if(hash[hashresult]!=null){
        if(hash[hashresult].first==s) {
            hash[hashresult]=null;
            sz--;
        }
        else{
            hashresult++;
            if(hashresult>N-1) hashresult = 0;
            while(hash[hashresult].first!=s){
                hashresult++;
                if(hashresult>N-1) hashresult=0;
                if(hashresult==suma%N) break;
            }
            if(hash[hashresult].first==s){
                hash[hashresult] = null;
            }
        }
    }
}
```

```cpp
int MapG::at(string s){
    long long int suma=0;
    for(int i = 0; i<s.size(); i++){
        suma = suma*z + s[i];
    }
    int hashresult = suma%N;
    if(hash[hashresult]!=null){
        if(hash[hashresult].first==s) return hash[hashresult].second;
        else{
            hashresult++;
            colisionesAt++;
            if(hashresult>N-1) hashresult = 0;
            while(hash[hashresult].first!=s){
                hashresult++;
                colisionesAt++;
                if(hashresult>N-1) hashresult=0;
                if(hashresult==suma%N) break;
            }
            if(hash[hashresult].first==s) return hash[hashresult].second;
            else return INT16_MIN;
        }
    }
}
int MapG::size(){
```

```cpp
int MapG::size(){
    return sz;
}
bool MapG::empty(){
    if(size()==0) return true;
    return false;
}

void MapG::print(){
    for(int i = 0; i<hash.size(); i++){
        if(hash[i]!=null) cout<<hash[i].first<<" "<<hash[i].second<<endl;
    }
}
void MapG::printColisiones(){
    cout<<"MapG"<<endl<<"Colisiones Insert: "<<colisiones<<endl<<"Colisiones at: "<<colisionesAt<<endl;
}
```

MapDH:

MapDH.h > MapDH > colisiones

```cpp
#include "ADTMap.h"
#include <vector>

using namespace std;

class MapDH : public ADTMap{
    private:
        long long int colisiones, colisionesAt;
        int sz;
        int z;
        int N;
        vector<pair<string,int>> hash;
        pair<string, int> null;
        void insertHash(pair<string,int>);
        void newHash();
    public:
        MapDH(int);
        void insert(pair<string, int>);
        void erase(string);
        int at(string);
        int size();
        bool empty();
        void print();
        void printColisiones();
};
```

MapDH.cpp > MapDH::printColisiones()

```cpp
#include "MapDH.h"
#include <iostream>

using namespace std;

MapDH::MapDH(int N){
    this->N=N;
    sz=0;
    null = {"", INT16_MIN};
    hash.assign(N+1, null);
    z=33;
    colisiones=0;
    colisionesAt=0;
}

void MapDH::insertHash(pair<string,int> par){
    long long int suma=0;
    for(int i = 0; i<par.first.size(); i++){
        suma = suma*z + par.first[i];
    }
    int j=0;
    int hashresult = (suma%N + j*(13-(suma%13)) ) % N;
    while(hash[hashresult]!=null){
        if(hash[hashresult].first==par.first) return;
        j++;
        colisiones++;
        hashresult = (suma%N + j*(13-(suma%13)) ) % N;
    }
    hash[hashresult]=par;
    sz++;
}
```

```cpp
void MapDH::newHash(){
    vector<pair<string, int>> aux;
    N = N*2;
    aux.assign(N+1, null);
    for(int i = 0; i<hash.size(); i++){
        if(hash[i]!=null){
            string s = hash[i].first;
            long long int suma=0;
            for(int i = 0; i<s.size(); i++){
                suma = suma*z + s[i];
            }
            int j=0;
            int hashresult = (suma%N + j*(13-(suma%13)) ) % N;
            while(aux[hashresult]!=null){
                j++;
                hashresult = (suma%N + j*(13-(suma%13)) ) % N;
            }
            aux[hashresult] = hash[i];
        }
    }
    hash = aux;
}
```

```cpp
void MapDH::insert(pair<string,int> par){
    if(size()>=(N/2)) newHash();
    insertHash(par);

}
```

```cpp
 void MapDH::erase(string s){
     long long int suma=0;
     for(int i = 0; i<s.size(); i++){
         suma = suma*z + s[i];
     }
     int j = 0;
     int hashresult = (suma%N + j*(13-(suma%13)) ) % N;
     if(hash[hashresult]!=null){
         if(hash[hashresult].first==s) {
             hash[hashresult]=null;
             sz--;
         }
         else{
             while(hash[hashresult].first!=s){
                 j++;
                 hashresult = (suma%N + j*(13-(suma%13)) ) % N;
                 if(hashresult == (suma%N) % N) return;
             }
             hash[hashresult] = null;
             sz--;
         }
     }

 }
```

```cpp
int MapDH::at(string s){
    long long int suma=0;
    for(int i = 0; i<s.size(); i++){
        suma = suma*z + s[i];
    }
    int j = 0;
    int hashresult = (suma%N + j*(13-(suma%13)) ) % N;
    if(hash[hashresult]!=null){
        if(hash[hashresult].first==s) return hash[hashresult].second;
        else{
            while(hash[hashresult].first!=s){
                j++;
                colisionesAt++;
                hashresult = (suma%N + j*(13-(suma%13)) ) % N;
                if(hashresult == (suma%N) % N) return INT16_MIN;
            }
            return hash[hashresult].second;
        }
    }
}
```

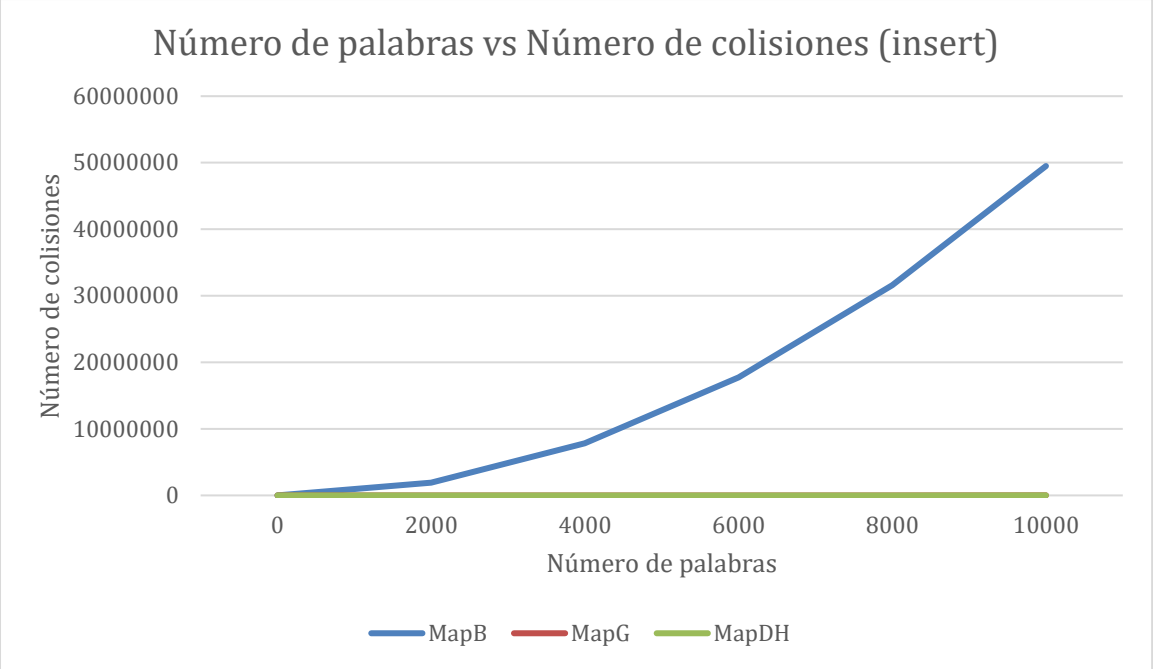
```cpp
int MapDH::size(){
    return sz;
}
bool MapDH::empty(){
    if(size()==0) return true;
    return false;
}

void MapDH::print(){
    for(int i = 0; i<N; i++){
        if(hash[i]!=null) cout<<hash[i].first<<" "<<hash[i].second<<endl;
    }
}
void MapDH::printColisiones(){
    cout<<"MapDH"<<endl<<"Colisiones Insert: "<<colisiones<<endl<<"Colisiones at: "<<colisionesAt<<endl;
}
```

3)

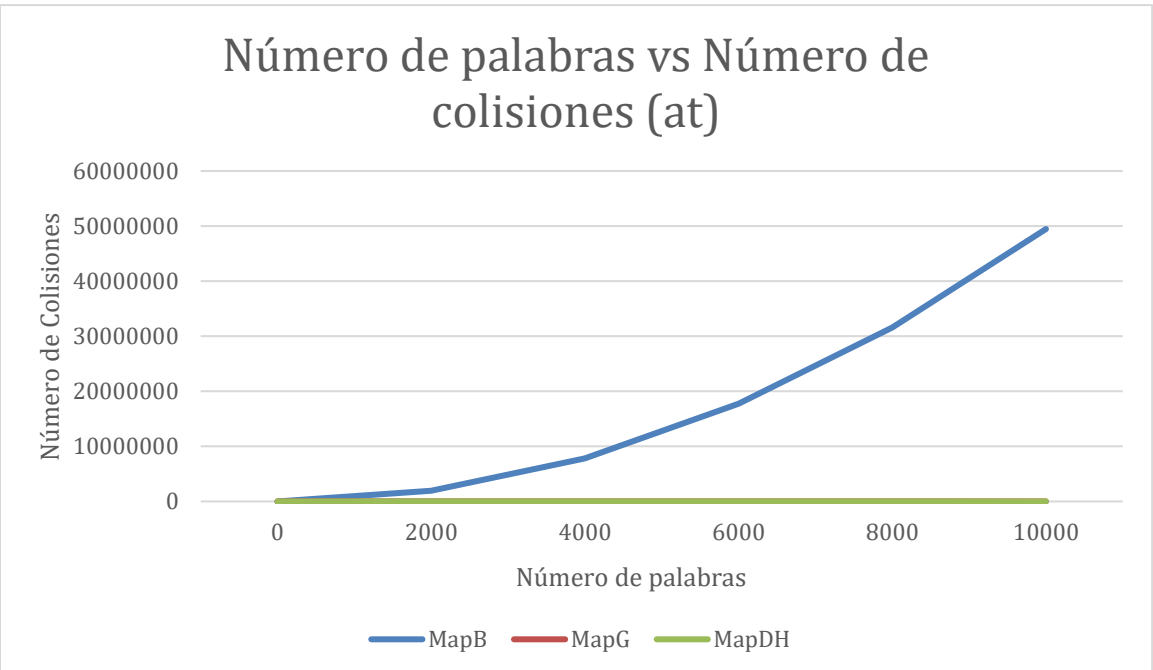
Método Insert:

N	Número de colisiones en Insert		
	MapB	MapG	MapDH
2000	1912923	1312	1160
4000	7810661	2855	2289
6000	17725295	4529	3415
8000	31613341	5891	4756
10000	49500933	7417	5838



Método at:

N	Número de colisiones en at		
	MapB	**MapG**	**MapDH**
2000	**1912923**	**900**	**772**
4000	**7810661**	**1913**	**1530**
6000	**17722446**	**3016**	**2379**
8000	**31603103**	**3961**	**3260**
10000	**49466358**	**5047**	**4039**



Análisis Teórico:

Los métodos insert y at, en pero caso son del orden O(n) ya que en peor caso podrían generarse n-1 colisiones. En todo caso se trata de un caso muy puntual y especifico. Al realizar rehashing, el tiempo es O(n+m), con m igual al nuevo espacio añadido. Considerando que cada vez se duplica el espacio este es O(2n).

Análisis experimental:

De los gráficos y datos tabulados se puede concluir que el MapB, es una forma muy mala de implementar un Map, principalmente por la función Hash implementada, la cual se basaba en sumar cada componente del string, lo que produce una gran cantidad de colisiones. También se puede concluir la importancia de utilizar una buena función de Hash comparando el MapB y MapG ya que, pese a que ambos utilizan linear probing, la buena función de Hash utilizada en MapG (Acumulación polinomial) hizo disminuir la cantidad de colisiones en gran medida. Y respecto a MapDH, el cual pasa del linear probing al doble hashing, utilizando la misma función de Hash que en MapG. Las colisiones si bien son menores, tampoco hay una diferencia tan marcada. En conclusión, el factor más importante para la implementación de Map es utilizar una buena función de Hash, como lo es la acumulación polinomial.