1

STANDARD TEMPLATE LIBRARY EN C++

Lic. Rodolfo Catunta

ENERO 2023

UNIVERSIDAD CATÓLICA BOLIVIANA

Contenidos I

1 Introducción

STLs en C++

¿Qué es?

La Standard Template Library es un conjunto de librerías que son soportadas como un estándar por todos los compiladores de C++. Las STLs contienen algoritmos y estructuras de datos usados a menudo en la Programación Competitiva.

include <algorithm>

La STL incluye algoritmos de ordenamiento, busqueda, de permutacion, de búsqueda y otros. Documentación

Estructuras de Datos

La STL incluye estructuras de datos ya implementadas como string, vector, set, map, queue, stack y sus variantes.

2

Strings

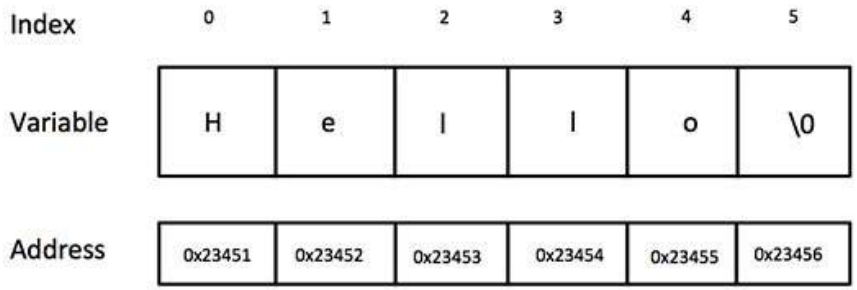
Strings

- Los strings, se utilizan para guardar cadenas de caracteres, inicialmente los caracteres ASCII usuales. Tabla ASCII

```
1 int main(){
2     string s; // String vacio
3     string t = "OBI"; // String con contenido
4     string r(3,'Z'); // String con contenido ZZZ
5     // Operacion básica - Concatenacion
6     string p = s+r;
7     p += t;
8     cout<<p<<"\n"; // Resultado ZZZOBI
9 }
```

Código 1: Constructores de la clase String

Acceso a elementos




```
1 string s = "Hello";
2 cout<<s[4]<<"\n"; // Imprime o
3 s[1]='a'; // Ahora s es "Hallo"
4 // Recorrido por indices
5 for(int i=0;i<s.size();i++){
6     cout<<s[i]<<" ";
7 }
8     cout<<"\n";
```

Código 2: Acceso a los elementos

Recorridos

```cpp
int main(){
    string s = "Olimpiada";
    // Recorrido Tradicional
    for(int i=0;i<s.size();i++){
        cout<<s[i]<<" ";
    }
    cout<<"\n";
    // Recorrido con iteradores
    for(string::iterator it=s.begin();it!=s.end();it++){
        cout<<(*it)<<" ";
    }
    cout<<"\n";
    // Recorrido con C++11 o superiores
    for(auto letra: s){
        cout<<letra<<" ";
    }
    cout<<"\n";
}
```

Código 3: Recorrido de una cadena

Principales Métodos de la clase string

Enlace a la Documentación

```cpp
int main(){
    string s = "Olimpiada";
    cout<<s.size()<<"\n"; // Longitud del string
    // MODIFICADORES
    s.push_back('s'); // Añade una s al final
    s.append(" Informatica"); // Añade un string al final
    s.insert(10," de"); // (pos, s) Añade el string s en la posicion pos
    s.erase(9,1); // (pos, len) Borra un string desde la posicion pos de longitud len
    cout<<s<<"\n";
    // BUSQUEDA
    if(s.find("Olimpiada") != -1){ // Retorna la posicion donde se encuentra el primer match o -1 en caso contrario
        // string encontrado
    }
    else{
        // string NO encontrado
    }
    // SUBSTR
    cout<<s.substr(13,11)<<"\n"; // (pos,len) Crea un substring desde la posicion pos de longitud len
}
```

Código 4: Principales Métodos de un string

3 Vectores

Vectores

- La **clase** vector se encarga de generar estructuras de datos denominadas vectores.
- Los vectores son similares a los arreglos, con la ventaja de que tienen la capacidad de **cambiar de tamaño**.
- Por la razón anterior son usados en ocasiones como reemplazo a los arreglos estáticos.
- Un componente característico de los vectores es que sus elementos pueden ser accedidos de dos formas.
  - Por un indice
  - Por medio de un iterador

Constructores

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main(){
    vector <int> v1; // vector vacio (sin casillas)
    vector <int> v2(8); // vector con 8 casillas vacias
    vector <int> v3(5,10); // vector con 5 casillas cada una con el numero 10
    vector <int> v4(v2); // vector copia de v2
}
```

Acceso a Elementos

```cpp
// Sea un vector v
// v = [7, 4, -1, 8, 5, 0, 1]

// Acceso por indice
cout<<v[0]<<endl; // imprimira 7
int suma = v[2] + v[6]; // suma sera igual a 0

// Accesos especiales
int frente = v.front(); // frente sera igual a 7
int ultimo = v.ultimo(); // ultimo sera igual a 1
```

Funciones de Capacidad

```cpp
// Sea el vector v
// v = [8, 4, 1, -7, 10]

int tam = v.size(); // .size devuelve la cantidad de casillas del vector
// entonces tam sera igual a 5

if(v.empty()){   // .empty devuelve un booleano
     // true si el vector esta vacio
     cout<<"v esta vacio"<<endl;
}
else{
     // false si el vector NO esta vacio
     cout<<"v no esta vacio"<<endl;
}
```

Funciones de Modificación

```
1  // Sea el vector v = [5,-1,7]
2  // Insertar al final
3  v.push_back(4);  // v = [5,-1,7,4]
4  // Quitar del final
5  v.pop_back();  // v = [5,-1,7]
6  // Insertar en una posicion (iterador)
7  v.insert(v.begin(),3);  // v = [3,5,-1,7]
8  // Borrar de una posicion (iterador)
9  v.erase(v.begin()++);  // v = [3,-1,7]
10 // Cambiar de tamanio
11 v.resize(5,0);  // v = [3,-1,7,0,0]
12
13 v.clear();  // Borra el contenido
```

Ordenamiento

```cpp
#include <iostream>
#include <vector>
#include <algorithm> // contiene a sort

using namespace std;

int main(){
    vector<int> v; // Sea v = [7,-8,1,4,0]
    // Orden creciente
    sort(v.begin(),v.end()); // v = [-8,0,1,4,7]
    // Orden decreciente
    sort(v.rbegin(),v.rend()); // v = [7,4,1,0,-8]
}
```

Busqueda Binaria

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main(){
    vector <int> v = {7,-8,1,4,0};
    // Ordenar primero
    sort(v.begin(),v.end()); // v = [-8,0,1,4,7]
    binary_search(v.begin(),v.end(),4); // true
    binary_search(v.begin(),v.end(),10); // false
    // lower bound primer elemento no menor (>=)
    int pos = lower_bound(v.begin(),v.end(),1)-v.begin(); // 2
    pos = lower_bound(v.begin(),v.end(),5)-v.begin(); // 4
    // upper bound primer elemento mayor (>)
    pos = upper_bound(v.begin(),v.end(),1)-v.begin(); // 3
}
```

4

Colas Dobles

Deque

- La clase deque, permite la utilización de colas dobles.
- Las colas dobles son similares a los vectores, con la única diferencia que soportan dos operaciones adicionales.
  - Insercion al frente **push_front(x)**
  - Borrado al frente **pop_front()**
- El termino cola doble, proviene de que esta estructura soporta inserciones y borrados en $O(1)$, tanto al inicio como al final. Lo que hace que tenga "dos colas".

push_front y pop_front

```cpp
#include <iostream>
#include <deque> // Libreria para usar colas dobles

using namespace std;

int main(){
  deque<int>d;
  d.push_back(7); // d = [7]
  d.push_back(4); // d = [7,4]
  d.push_front(20); // d = [20,7,4]
  d.push_front(1); // d = [1,20,7,4]
  d.pop_back(); // d = [1,20,7]
  d.pop_front(); // d = [20,7]
  return 0;
}
```

5

Colas

Colas

- Es una estructura de datos dinámica lineal, que tiene la característica de parecerse a una cola de la vida real.
- Una cola tiene una disciplina FIFO, First In First Out (Primero que llega, Primero que sale)
- En general una cola es un tipo especial de lista, en el que solo se puede acceder al elemento del inicio de la cola.

Declaración y Creación

```cpp
#include <iostream>
#include <queue> // Para usar colas

using namespace std;

int main(){
  queue<int> q; // Cola de enteros
  queue<char> q; // Cola de caracteres
}
```

Inserción y Borrado de Elementos

```cpp
#include <iostream>
#include <queue> // Para usar colas

using namespace std;

int main(){
  queue<int> q;
  q.push(5); // q = 5
  q.push(2);  // q = 5 2
  q.push(1); // q = 5 2 1
  q.pop(); // q = 2 1
}
```

Acceso y Funciones de Capacidad

```cpp
int main(){
  queue<int> q; // Sea q con varios elementos
  cout<<q.front()<<endl; // .front obtiene el primer elemento de la cola
  cout<<q.back()<<endl; // .back obtiene el ultimo elemento de la cola
  cout<<q.size()<<endl; // .size obtiene el tamanio de la cola
  if(q.empty()){ // .empty determina si la cola esta vacia.
    // true
    cout<<"Cola vacia"<<endl;
  }
  else{
    // false
    cout<<"Cola no vacia"<<endl;
  }
}
```

Proceso de Recorrido de una Cola

```cpp
#include <iostream>
#include <queue> // Para usar colas

using namespace std;

int main(){
  queue<int> q;
  //... sea q con varios elementos

  // RECORRIDO
  while(!q.empty()){// Mientras la cola no este vacia
    cout<<q.front()<<endl; // imprimir el primer elemento de la cola
    q.pop();// sacar el primer elemento de la cola
  }
}
```

6

Pilas

Pilas

- Es una estructura de datos dinámica lineal, que tiene la característica de parecerse a una pila de objetos.
- Una pila tiene una disciplina LIFO, Last In First Out (Ultimo que llega, Primero que sale)
- En general una pila es un tipo especial de lista, en el que solo se puede acceder al elemento de la cima de la pila.

Declaración y Creación

```cpp
#include <iostream>
#include <stack> // Para usar pilas

using namespace std;

int main(){
  stack<int> q; // Pila de enteros
  stack<char> q; // Pila de caracteres
}
```

Inserción y Borrado de Elementos

```cpp
#include <iostream>
#include <stack> // Para usar pilas

using namespace std;

int main(){
  stack<int> p;
  p.push(5); // p = 5
  p.push(2);  // p = 2 5
  p.push(1); // p = 1 2 5
  p.pop(); // p = 2 5
}
```

Acceso y Funciones de Capacidad

```cpp
int main(){
  stack<int> p; // Sea p con varios elementos
  cout<<p.top()<<endl; // .front obtiene el elemento de la cima de la pila
  cout<<p.size()<<endl; // .size obtiene el tamanio de la pila
  if(p.empty()){ // .empty determina si la pila esta vacia.
    // true
    cout<<"Pila vacia"<<endl;
  }
  else{
    // false
    cout<<"Pila no vacia"<<endl;
  }
}
```

Proceso de Recorrido de una Pila

```cpp
#include <iostream>
#include <stack> // Para usar pilas

using namespace std;

int main(){
  stack<int> p;
  //... sea p con varios elementos

  // RECORRIDO
  while(!p.empty()){// Mientras la pila no este vacia
    cout<<p.top()<<endl; // imprimir el elemento de la cima de la pila
    p.pop();// sacar el primer elemento de la pila
  }
}
```

7

Set

Problema de Motivación

Problema

Dada una lista de N números enteros se quiere saber cuantos números distintos tiene la lista.

Límites

- $1 \leq N \leq 10^6$

Conjunto

- Los conjuntos son estructuras de datos no lineales (guardado en forma de **árbol binario balanceado**) que permiten guardar elementos no repetidos y en un cierto orden.
- Los conjuntos por naturaleza guardan elementos en orden creciente, es decir, que guardara números de menor a mayor y, por otro lado, cadenas, en orden lexicográfico.
- Los conjuntos se pueden utilizar por ejemplo para los siguientes casos:
  - Contar cuantos elementos no repetidos tiene una lista.
  - Mantener una lista ordenada, en cierto orden.
  - Simula una cola de prioridad*
  - Realizar estructuras mas complejas en las que se necesite un orden.

Visualización

Para generar arboles binarios balanceados se pueden usar las siguientes técnicas*:

- AVL (Adelson-Velskii Landis)
- Red-Black Trees

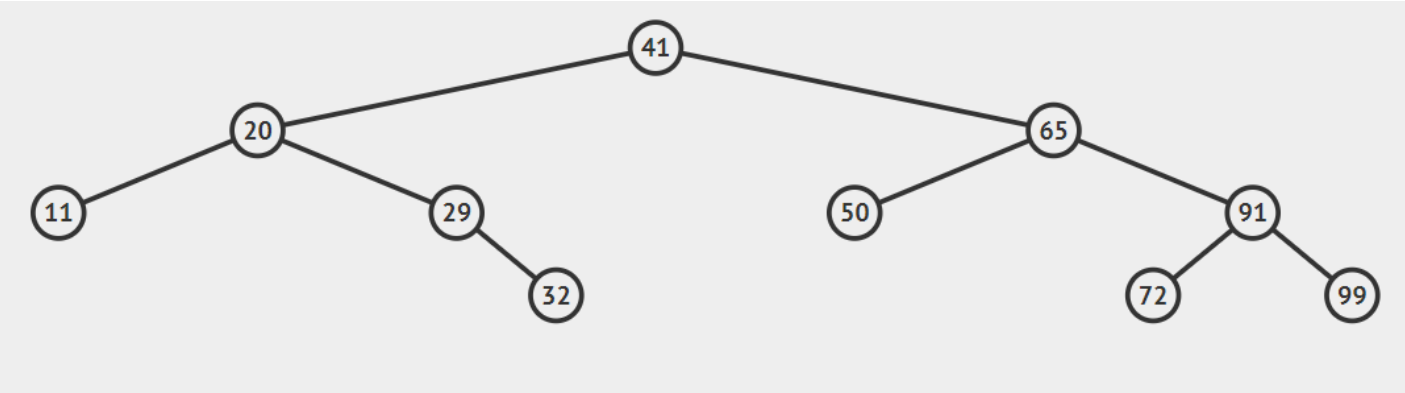


Figure 1: Ejemplo de un árbol binario balanceado, forma en la que C++ guarda e implementa el uso de set.

Constructores

```cpp
#include <iostream>
#include <set> // Libreria para usar conjuntos
using namespace std;

int main(){
  set <int> conj; // conjunto que ordenara en orden creciente
  set <int, greater<int> > conj_dec; // conjunto que ordenara en orden
      decreciente

  return 0;
}
```

Funciones Modificadoras

```cpp
#include <iostream>
#include <set> // Libreria para usar conjuntos
using namespace std;

int main(){
  set <int> conj; // conjunto que ordenara en orden creciente

  conj.insert(4); // inserta el elemento 4
  conj.insert(7); // inserta el elemento 7
  conj.insert(1); // inserta el elemento 1

  conj.erase(1); // elimina el elemento 1 del conjunto

  conj.clear(); // elimina todo el contenido del conjunto
  return 0;
}
```

Funciones de Capacidad

```cpp
int main(){
  set <int> conj; // conjunto que ordenara en orden creciente

  conj.insert(4); // inserta el elemento 4
  conj.insert(7); // inserta el elemento 7
  conj.insert(1); // inserta el elemento 1

  cout<<conj.size()<<endl; // imprimira 3 ya que es el tamanio del set

  if(conj.empty()){ // .empty() nos indica si el set esta o no vacio
    //True
    cout<<"El conjunto esta vacio"<<endl;
  }
  else{
    //False
    cout<<"El conjunto NO esta vacio"<<endl;
  }
}
```

Funciones de Consulta

```
1  set <int> conj; // conjunto que ordenara en orden creciente
2  conj.insert(4); conj.insert(7); conj.insert(1); conj.insert(10);
3  cout<<conj.count(7)<<endl; // imprira 1 ya que el 7 aparece una vez en el set
4  if(conj.find(3)!=conj.end()){ // .find() devuelve un iterador apuntando al
       elemento
5    cout<<"3 esta en el conjunto"<<endl;
6  }else{ // si no encuentra el elemento .find() devuelve un iterador a .end()
7    cout<<"3 no esta en el conjunto"<<endl;
8  }
9  cout<<*(conj.lower_bound(3))<<endl; // imprimira 4 ya que es el primer elemento
       no menor que 6
10 cout<<*(conj.lower_bound(7))<<endl; // imprimira 7 ya que es el primer elemento
       no menor que 7
11 cout<<*(conj.upper_bound(1))<<endl; // imprimira 4 ya que es el primer elemento
       mayor que 2
12 cout<<*(conj.upper_bound(4))<<endl; // imprimira 7 ya que es el primer elemento
       mayor que 4
```

Recorrido

```cpp
int main(){
  set <int> conj; // conjunto que ordenara en orden creciente

  conj.insert(4); // inserta el elemento 4
  conj.insert(7); // inserta el elemento 7
  conj.insert(1); // inserta el elemento 1
  conj.insert(10); // inserta el elemento 10

  // Recorrido Tradicional
  set<int>::iterator it; // Creacion del iterador
  for(it=conj.begin();it!=conj.end();it++){
    cout<<*it<<" "; // Acceso al elemento dentro del iterador
  }
  cout<<endl;
  return 0;
}
```

Recorrido de un Conjunto

```cpp
int main(){
  set <int> conj; // conjunto que ordenara en orden creciente
  set <int, greater<int> > conj_dec; // conjunto que ordenara en orden
      decreciente

  conj.insert(4); // inserta el elemento 4
  conj.insert(7); // inserta el elemento 7
  conj.insert(1); // inserta el elemento 1
  conj.insert(10); // inserta el elemento 10

  // RECORRIDO en C++ 11
  for(auto elemento: conj){
    cout<<elemento<<" ";
  }
  cout<<endl;
}
```

8

Map

Problema de Motivación

Problema

Dada una lista de N números enteros se quiere saber cual es el número que más se repite en la lista.

Límites

- $1 \leq N \leq 10^6$

Mapas

- Los mapas son estructuras de datos no lineales que permiten guardar pares de elemento $llave \rightarrow valor$.
- Los mapas no pueden guardar llaves repetidas y solo pueden guardar un valor por cada llave (este valor puede ser de cualquier tipo valido de dato o estructura de datos).
- Los mapas por naturaleza guardan sus llaves en orden creciente, es decir, que guardara números de menor a mayor y, por otro lado, cadenas, en orden lexicográfico.
- Los mapas se pueden utilizar por ejemplo para los siguientes casos:
  - Realizar una tabla de Frecuencias
  - Simular una tabla de Hash (Tabla de Correspondencias)
  - Realizar una función de mapeo.

Visualización

Los mapas tienen la misma forma de **árbol binario balanceado** que los sets, con la característica adicional que cada valor del árbol, puede guardar un valor adicional.

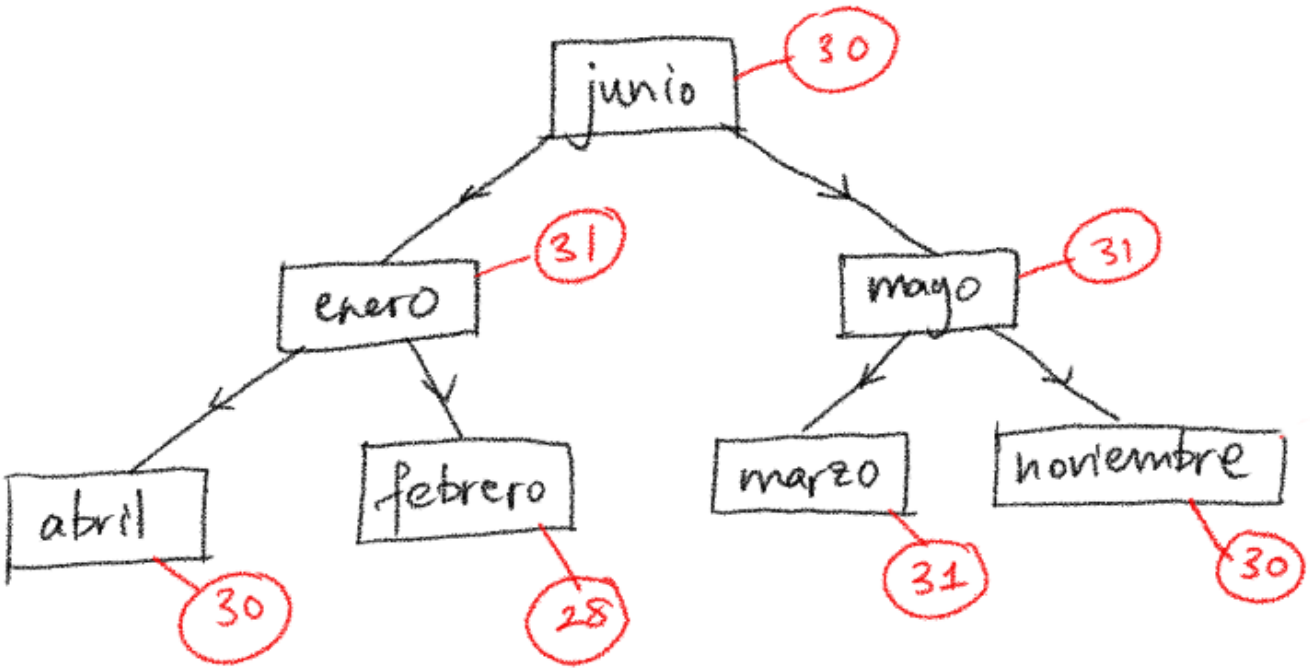


Figure 2: Ejemplo de un mapa de llave string y valor int.

Constructores

```cpp
#include <iostream>
#include <map> // Libreria para usar mapas
using namespace std;

int main(){
  map <string,int> mapa; // Crea un mapa de llave tipo string y valor int
}
```

Funciones Modificadoras

```cpp
#include <iostream>
#include <map> // Libreria para usar mapas
using namespace std;

int main(){
  map <string,int> mapa; // Crea un mapa de llave tipo string y valor int
  mapa["enero"]=31; // inserta la llave enero con el valor 31
  mapa["febrero"]=28; // inserta la llave febrero con el valor 28
  mapa.insert(make_pair("junio",30)); // inserta la llave junio con el valor 30
  mapa.insert(make_pair("septiembre",30)); // inserta la llave septiembre con el
      valor 30

  mapa.erase("junio"); // elimina el elemento con llave "junio" del mapa

  mapa.clear(); // elimina todo el contenido del mapa
}
```

Funciones de Capacidad

```cpp
int main(){
  map <string,int> mapa;
  mapa["enero"]=31;
  mapa["febrero"]=28;
  mapa.insert(make_pair("junio",30));
  mapa.insert(make_pair("septiembre",30));

  cout<<mapa.size()<<endl; // imprimira 4 ya que es el tamanio del mapa

  if(mapa.empty()){ // .empty() nos indica si el mapa esta o no vacio
    //True
    cout<<"El mapa esta vacio"<<endl;
  }
  else{
    //False
    cout<<"El mapa NO esta vacio"<<endl;
  }
}
```

Funciones de Consulta

```
1  map <string,int> mapa;
2  mapa["enero"]=31;   mapa["febrero"]=28; mapa["junio"]=30; mapa["septiembre"]=30;
3  cout<<mapa.count("enero")<<endl; // imprira 1 ya que la llave enero aparece una vez en el mppa
4  if(mapa.find("mayo")!=mapa.end()){ // .find() devuelve un iterador apuntando a la llave indicada
5    cout<<"mayo esta en el mapa"<<endl;
6  }else{ // si no encuentra la llave .find() devuelve un iterador a .end()
7    cout<<"mayo no esta en el mapa"<<endl;
8  }
9  cout<<(mapa.lower_bound("mayo"))->first<<endl; // imprimira septiembre
10 cout<<(mapa.lower_bound("septiembre"))->first<<endl; // imprimira septiembre
11 cout<<(mapa.upper_bound("abril"))->first<<endl; // imprimira enero
12 cout<<(mapa.upper_bound("febrero"))->first<<endl; // imprimira junio
```

Recorrido

```cpp
int main(){
  map <string,int> mapa; // Crea un mapa de llave tipo string y valor int
  mapa["enero"]=31; // inserta la llave enero con el valor 31
  mapa["febrero"]=28; // inserta la llave febrero con el valor 28
  mapa.insert(make_pair("junio",30)); // inserta la llave junio con el valor 30
  mapa.insert(make_pair("septiembre",30)); // inserta la llave septiembre con el valor 30

  // Recorrido Tradicional
  map<string,int>::iterator it; // Creacion del iterador
  for(it=mapa.begin();it!=mapa.end();it++){
    cout<<it->first<<" "<<it->second<<endl; // Acceso al elemento dentro del iterador
    // first para la llave
    // second para el valor
  }
}
```

Recorrido de un Mapa

```cpp
int main(){
  map <string,int> mapa; // Crea un mapa de llave tipo string y valor int
  mapa["enero"]=31; // inserta la llave enero con el valor 31
  mapa["febrero"]=28; // inserta la llave febrero con el valor 28
  mapa.insert(make_pair("junio",30)); // inserta la llave junio con el valor 30
  mapa.insert(make_pair("septiembre",30)); // inserta la llave septiembre con el
      valor 30

  // RECORRIDO en C++ 11
  for(auto elemento: mapa){
    cout<<elemento.first<<" "<<elemento.second<<endl;
  }
}
```

9

Colas de Prioridad (Monticulos)

Problema de Motivación

Problema

Dada una lista de N números enteros distintos en la que se pueden hacer las siguientes operaciones.

- Ingresar un nuevo número a la lista
- Borrar el número mas grande

Se quiere responder a Q operaciones de cualquiera de los tipos mencionados, indicando cuál es el número más grande de la lista en ese momento.

Límites

- $1 \leq N \leq 10^4$
- $1 \leq Q \leq 10^6$

Colas de Prioridad

- Una cola de prioridad, es en realidad una Estructura Abstracta de Datos, denominada **montículo (heap)**.
- Los montículos tienen forma de **árbol binario**, con las características adicionales de tener que ser un **árbol completo** y cumplir con la **propiedad de montículo (heap-property)**
- **Propiedad de Montículo** esta propiedad nos dice que para todo subárbol con raíz en el nodo x los elementos a la izquierda y a la derecha del nodo x deben ser menores (o mayores) al elemento en el nodo x.

Visualización

- **Max-Heap** La raíz del árbol es el elemento máximo.
- **Min-Heap** La raíz del árbol es el elemento mínimo.

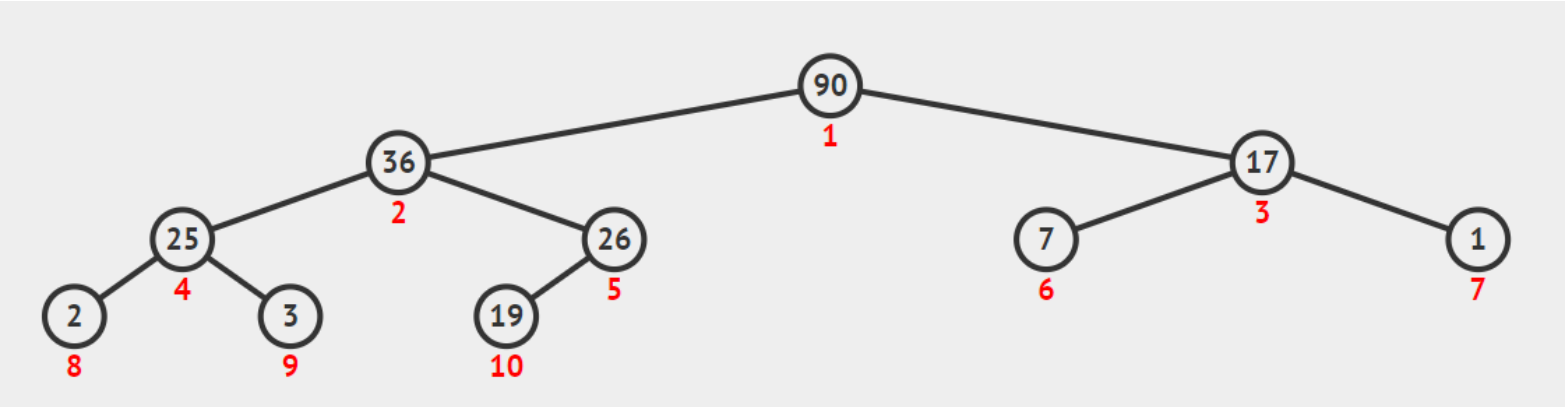


Figure 3: Ejemplo de Max-Heap, priority_queue es un max-heap por defecto

Constructores

```cpp
#include <iostream>
#include <queue> // Para usar priority queue
using namespace std;

int main(){
  priority_queue<int>q_max; // Cola de Prioridad (Max-Heap)

  priority_queue <int, vector<int>, greater<int> > q_min; // Cola de
      Prioridad (Min-Heap)

  return 0;
}
```

Funciones Modificadoras

```cpp
#include <iostream>
#include <queue>
using namespace std;

int main(){
   priority_queue<int>pq; // Cola de Prioridad (Max-Heap)
   pq.push(7); // aniade 7 a la cola de prioridad
   pq.push(14); // aniade 14 a la cola de prioridad
   pq.push(1); // aniade 1 a la cola de prioridad
   cout<<pq.top()<<endl; // Imprimira 14 pues es el elemento mas grande
   pq.pop(); // Quita el elemento mas grande
   cout<<pq.top()<<endl; // Imprimira 7 pues ahora es el elemento mas grande
   return 0;
}
```

Funciones de Capacidad

```cpp
int main(){
  priority_queue<int>pq; // Cola de Prioridad (Max-Heap)
  pq.push(7); // aniade 7 a la cola de prioridad
  pq.push(14); // aniade 14 a la cola de prioridad
  pq.push(1); // aniade 1 a la cola de prioridad
  cout<<pq.size()<<endl; // Imprimer 3 ya que la cola de prioridad tiene 3 elementos

  if(pq.empty()){ // .empty() nos dice si la cola esta vacia
    //True
    cout<<"La cola de prioridad esta vacia"<<endl;
  }
  else{
    //False
    cout<<"La cola de prioridad NO esta vacia"<<endl;
  }
  return 0;
}
```

Recorrido

```cpp
#include <iostream>
#include <queue>
using namespace std;

int main(){
   priority_queue<int>pq;
   pq.push(7); // aniade 7 a la cola de prioridad
   pq.push(14); // aniade 14 a la cola de prioridad
   pq.push(1); // aniade 1 a la cola de prioridad
   //RECORRIDO
   while(!pq.empty()){
      cout<<pq.top()<<endl; // Obtenemos el elemento
      pq.pop(); // Lo sacamos de la cola de prioridad
   }
   return 0;
}
```

10 Tablas Hash

Tablas Hash

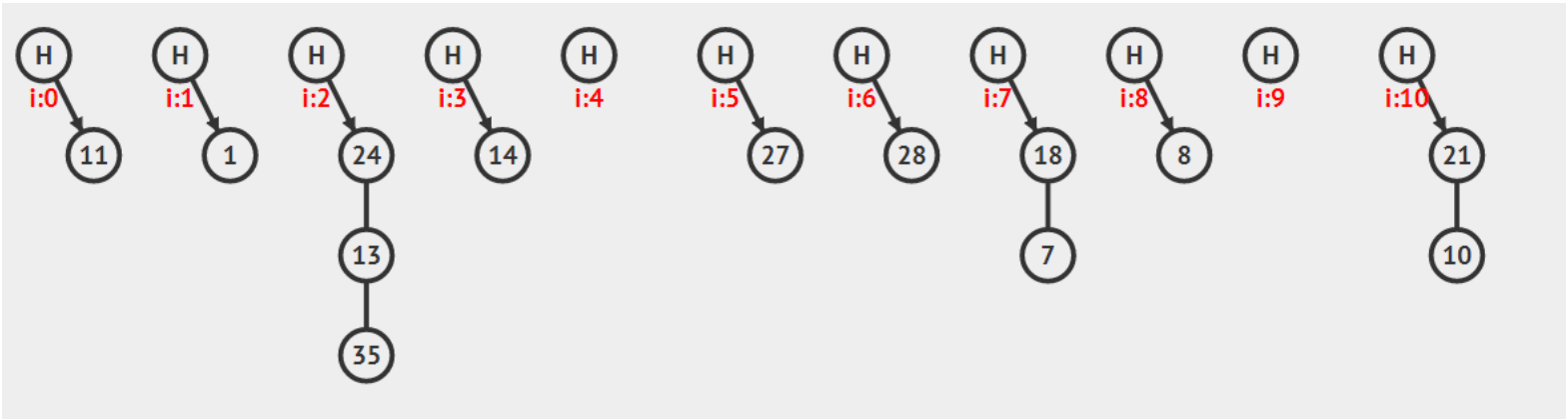
Las tablas Hash son una estructura de datos no lineal eficiente para implementar una Tabla de Datos Abstractos (ADT) que requiere operaciones muy rápidas, muy cercanas a $O(1)$.

- Inserción
- Búsqueda/Acceso
- Actualización
- Borrado

Para que una Tabla Hash sea exitosa se debe tener lo siguiente:

- Una buena función de Hashing
- Mecanismos de solución de colisiones

Visualización Tabla Hash



Implementación en C++ 11 (unordered_set)

```cpp
#include <iostream>
#include <unordered_set>
using namespace std;
int main(){
    unordered_set<string> us = {"Lucas","Raul","Andrea","Juan","Maria"};
    for(string elemento: us){
        cout<<elemento<<" ";
    }
    cout<<endl; // Maria Lucas Raul Juan Andrea
    // Find O(k)
    if(us.find("Rodolfo")!=us.end())
        cout<<"Rodolfo esta en el conjunto"<<endl;
    // Borrado
    us.erase("Juan"); // O(k)
}
```

Implementación en C++ 11 (unordered_map)

```cpp
#include <iostream>
#include <unordered_map>
using namespace std;
int main(){
    unordered_map<string,int> um;
    um["Pepe"]=3;
    um["Marcelo"]=4;
    um.insert(make_pair("Luka",7));
    um.insert(make_pair("Iker",1));
    um.insert(make_pair("Andres",9));
    for(auto elemento: um){
        cout<<elemento.first<<" "<<elemento.second<<endl;
    }
    // Find O(k)
    if(um.find("Rodolfo")!=um.end())
        cout<<"Rodolfo esta en el conjunto"<<endl;
    // Borrado
    um.erase("Pepe"); // O(k)
}
```

¡Gracias!