Standard Template Library (STL)

B.S. Rodolfo Mercado Gonzales

Standard Template Library

“Es mejor reutilizar que reescribir”

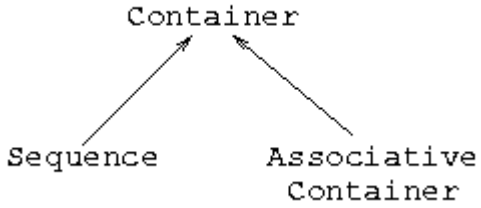
Standard Template Library

- Librería de estructuras de datos y algoritmos que forman parte del estándar de C++.
- Evita que se tenga que programar algo de uso frecuente.
- Presenta conceptos como contenedores e iteradores.

Contenedores

- Estructura que puede almacenar una colección de elementos del mismo tipo.
- Administra su memoria en el heap.
- Se accede a sus elementos a través de iteradores y algunas veces directamente.
- Son implementaciones de estructuras muy comunes en programación.

Contenedores



Contenedores de secuencia

El usuario controla el orden de los elementos. Ejm: vector, list, deque.

Contenedores asociativos

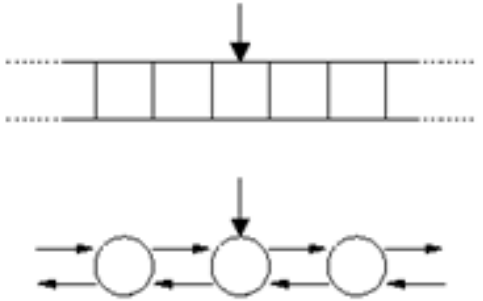
El contenedor controla la posición de los elementos, permite buscar un elemento a través de una llave (key). Ejm: set, multiset, map, multimap.

***Adaptadores de contenedores**

Contenedores implementados en base a otros. Ejm: stack, queue, priority queue

Iteradores

- Es un puntero generalizado que identifica una posición en un contenedor.
- Nos permite recorrer un contenedor en la dirección que nos permita.



Iteradores

- Iterador al primer elemento $contenedor.begin()$
- Ir al siguiente elemento $iterador++$
- Reconocer fin del contenedor $contenedor.end()$
- Acceder al valor de un elemento $*iterador$

Vector

- Contenedor que almacena elemento en posiciones contiguas de memoria.
- Pueden cambiar de tamaño en tiempo de ejecución.
- Permite acceso aleatorio.
- Permite insertar y eliminar un elemento al final en tiempo constante.
- Permite simular una pila.

Vector

```
vector< int > v ; // declarar vector
v.size(); // tamaño del vector O(1)
v[ i ] // acceso O(1)
v.push_back( x ); //agregar un elemento al final O(1)
v.pop_back(); // eliminar último elemento O(1)
v.front(); // obtener primer elemento O(1)
v.back(); // obtener el último elemento O(1)

// insertar elemento x en posición pos O(n)
vector<int>::iterator it;
it = v.begin() + pos ;
v.insert ( it , x);

// eliminar elemento en posicion pos O(n)
v.erase (v.begin() + pos );
```

Problemas

HackerRank – Vector-Sort

HackerRank – Vector-Erase

Parentización Balanceada

Dado un string compuesto de ‘(‘ y ‘)’ , decir si la expresión presenta parentización balanceada.

Ejemplos:

( ) si
(()()) si
(())(no

Problemas

Hackerrank – Balanced Brackets

Codeforces – Plug-in

Deque

- Pueden cambiar de tamaño en tiempo de ejecución.
- Permite acceso aleatorio.
- Permite insertar y eliminar un elemento al inicio y al final en tiempo constante.
- Permite simular una pila y una cola.

Deque

```
deque< int > dq ; // declarar deque
dq.size(); // tamaño del deque O(1)
dq[ i ]; // accesos O(1)
dq.push_back( x ); // agregar un elemento al final  O(1)
dq.push_front( x ); // agregar un elemento al inicio O(1)
dq.pop_back(); // eliminar el último elemento  O(1)
dq.pop_front(); // eliminar el primer elemento O(1)
dq.front(); // obtener primer elemento O(1)
dq.back(); // obtener el último elemento O(1)

// insertar elemento x en posición pos O(n)
deque<int>::iterator it;
it = dq.begin() + pos ;
dq.insert ( it , x);

// eliminar elemento en posicion pos O(n)
dq.erase (v.begin() + pos );
```

Map

- También conocido como diccionario.
- Nos permite almacenar pares de la forma <llave, valor> , donde la clave es única.
- Los elementos son guardados en orden ascendente respecto a su clave.
- Las operaciones de búsqueda, inserción y eliminación se hacen en tiempo logarítmico respecto al tamaño del contenedor.

Map

```
map< string, int > Edad; // declarar map
Edad.size(); // tamaño del map O(1)
Edad["Marco"] // acceso O(log n)
Edad["Marco"] = 30; // insertar elemento O(log n)
Edad.erase("Marco"); // eliminar elemento O(log n)

map<string,int>::iterator it; //iterador para map
it = Edad.find("Marco"); // busca elemento por clave O(log n)
```

Problemas

HackerRank – Maps-STL

UVA 10420 – List of Conquests

Codeforces – Restoring Password

Set

- Nos permite almacenar llaves (elementos únicos).
- Los elementos son guardados en orden ascendente.
- Las operaciones de búsqueda, inserción y eliminación son en tiempo logarítmico respecto al tamaño del contenedor.

Set

```
set< int > S // declarar set
S.size() // tamaño del set O(1)
S.insert(x) // insertar elemento  O(log n)
S.erase(x) // eliminar elemento O(log n)

set< int >::iterator it; //iterador para set
it = S.find(x); // busca elemento O(log n)
```

Problemas

HackerRank – Sets-STL

Codeforces 228A – Is your horseshoe on the other hoof?

HackerRank Codesprint 2 – Minimum Loss

Cola de Prioridad

- El primer elemento de la cola será el de mayor prioridad.
- Inserta en tiempo logarítmico.
- Elimina el elemento de mayor prioridad en tiempo logarítmico.
- Acceso al elemento de mayor prioridad en tiempo constante.

Cola de Prioridad

```
priority_queue <int> Q; //declara cola de prioridad
Q.size(); // tamaño de cola de priorida
Q.push(x); // inserta elemento O(log n)
Q.top(); // devuelve el elemento de mayor prioridad O(1)
Q.pop(); // elimina el elemeto de mayor prioridad O(log n)*/
```

Problemas

Inicialmente usted tiene una caja vacía. Se le darán q $(q \leq 10^5)$ consultas de 3 tipos :

1 x : insertar un elemento con valor x $(x \leq 10^5)$.

2 : eliminar el elemento de menor valor.

3 : imprimir el elemento de menor valor.

Entrada

6
1 7
1 3
2
3
1 9
3

Salida

7
7

Algoritmos STL

- Sort $O(n\ logn)$

Permite ordenar eficientemente los elementos de un arreglo, vector o deque.

```
sort( arr, arr + n ); // arreglo

sort( v.begin(), v.end() ); // vector o deque
```

Algoritmos STL

- Sort $O(n\,logn)$

Alguna veces tenemos que definir nuestro operador **<**

```
struct tupla{
    int x, y;
    tupla(){ //constructor vacio
    }
    tupla(int a, int b){ //constructor definido
        x = a, y = b;
    }
    //sobrecarga de operador <
    bool operator < ( const tupla &tup )const{
        if(x != tup.x) return x < tup.x;
        return y < tup.y;
    }
};
```

Algoritmos STL

- Binary Search $O(\log n)$

Permite saber de manera eficiente si un elemento x está presente en un arreglo, vector o deque, para esto el contenedor debe estar ordenado de manera ascendente.

```
bool isPresent = binary_search( v.begin(), v.end(), val );
```

Algoritmos STL

- Lower bound $O(\log n)$

Dado un arreglo/contenedor ordenado ascendentemente, retorna un puntero/iterador a la posición del primer elemento que es mayor o igual a un elemento x.

```
int *p = lower_bound( A, A + n, 3 );
vector<int> :: iterator it = lower_bound( v.begin(), v.end(), x );
```

Algoritmos STL

- Upper Bound $O(\log n)$

Dado un arreglo/contenedor ordenado ascendentemente, retorna un puntero/iterador a la posición del primer elemento que es mayor a un elemento x.

```
int *p = upper_bound( A, A + n, 3 );
vector<int> :: iterator it = upper_bound( v.begin(), v.end(), x );
```

Problemas

Codeforces – Rank List

Referencias

- Giménez, Omer. Guías de Programación C++ STL.
- Oualline, Steve. Practical C++ programming.
- University of Helsinski. Algorithm Libraries
  https://www.cs.helsinki.fi/u/tpkarkka/alglib/k06/

¡ Good luck and have fun !