CLUB DE PROGRAMACIóN COMPETITIVA
UADY

2

C++: Standard Template Library

CLUB DE PROGRAMACIóN COMPETITIVA UADY

STL

La Standard Template Library (STL) es una colección de estructuras de datos genéricas y algoritmos escritos en C++.

En un concurso nos interesa pasar la menor cantidad de tiempo implementando, por lo que tener estas estructuras de uso común ya implementadas con las mejores garantías de complejidad con las que podemos contar es de gran utilidad

Podemos incluirla en su totalidad con

```
#include <bits/stdc++.h>
```

y especificando

```
using namespace std;
```

CLUB DE PROGRAMACIóN COMPETITIVA UADY

Iteradores

Son objetos que pueden iterar sobre los elementos de un contenedor, funcionan como referencias a elementos dentro de estos, como si fueran punteros. Usualmente las funciones de la STL que operan sobre estos reciben como parámetros o devuelven iteradores en lugar de punteros, por lo que es importante saber como manejarlos.

```
vector<int> mi_vector = {1, 2, 3, 4, 3, 4, 7, 2, 3, 1};

auto beg = mi_vector.begin(); //iterador al primer elemento
auto end = mi_vector.end(); //iterador a “después del último elemento”. Marca el fin del contenedor y no debe accederse.

int x = *beg; // 1
int y = *end; // comportamiento indefinido!

sort(beg, end); //ordena el vector
```

CLUB DE
PROGRAMACIóN
COMPETITIVA
UADY

Contenedores

Almacenan datos de cualquier tipo y permiten su acceso de manera conveniente según sea el caso de uso.

Podemos anidarlos casi de cualquier manera:

```
queue<vector<pair<string, int>>> wtf[10];
```
(arreglo de tamaño 10 de colas de vectores de pares de string y entero)

y podemos recorrerlos usando iteradores

```
set<int> mi_set;
for(int elem : mi_set){
    // elem es el elemento de mi_set en la iteración actual
    // podemos omitir el tipo de elem usando auto
}
```

CLUB DE PROGRAMACIóN COMPETITIVA UADY

pair<T, T>

El contenedor pair permite almacenar y mantener vincluados un par de valores.

```
pair<int , int> mi_par;
pair<int , int> mi_par1 = {1, 2};
// mi_par1.first = 1;
// mi_par1.second = 2;
```

Supongamos que tenemos la cantidad de dinero que recibimos en cada día de un mes, ¿Como podemos ordenar los días del mes según la cantidad de dinero obtenido?

CLUB DE PROGRAMACIóN COMPETITIVA UADY

vector<T>

Implementa un arreglo dinámico. A diferencia del array, puede cambiar su tamaño en tiempo de ejecución.

```
   O(1)  vector<int> vec;                          // declaración sin tamaño inicial
   O(n)  vector<int> vec1[25];                     // declaración con tamaño inicial
   O(n)  vector<int> vec2 = {1, 2, 3, 4, 5, 6};    // declaración explícita
   O(1)  vec1[3] = 12;                             // asignar 12 a posición 3 de vec1
   O(1)  vec2.push_back[7];                        // insertar 7 al final de vec2
   O(1)  vec2.pop_back[];                          // remover al final de vec2
         vec2.begin[];                             // primer elemento
         vec2.end[];                               // después del último elemento
O(nlogn) sort[vec2.begin[], vec2.end[]];           // ordenar vec2
   O(1)  vec1.size[]                               // tamaño de vec1 [25]
         vec[10000] = 17;                          // segfault!
```

CLUB DE PROGRAMACIóN COMPETITIVA UADY

stack<T>

Implementa una pila. (LIFO)

```
      stack<int> mi_pila;
O(1)  mi_pila.push(2);          // insertar 2 en el tope de mi_pila
O(1)  mi_pila.pop();            // eliminar elemento en el tope de mi_pila
O(1)  mi_pila.top();            // retornar tope de mi_pila, segfault!
O(1)  mi_pila.empty();          // retorna si mi_pila está vacía (true)
O(1)  mi_pila.size();           // tamaño de mi_pila (0)
```

Tenemos una string de parentesis, ¿Como podemos saber si es una expresión balanceada?

CLUB DE PROGRAMACIóN COMPETITIVA UADY

queue<T>

Implementa una cola. (FIFO)

```
      queue<int> mi_cola;
O(1)  mi_cola.push(2);          // insertar 2 en el frente de mi_cola
O(1)  mi_cola.pop();            // eliminar elemento en el frente de mi_cola
O(1)  mi_cola.front();          // retornar frente de mi_cola, segfault!
O(1)  mi_cola.empty();          // retorna si mi_cola está vacía (true)
O(1)  mi_cola.size();           // tamaño de mi_cola (0)
```

Un caso de uso común de las colas es en recorridos en anchura de grafos (BFS)

CLUB DE PROGRAMACIóN COMPETITIVA UADY

deque<T> pronunciado /dɛk/ “Deck”

Es una cola con acceso constante al frente y al fondo.

```
      deque<int> mi_cola;
O(1)  mi_cola.push_back(1);          // Insertar 1 al fondo de mi_cola
O(1)  mi_cola.push_front(2);         // Insertar 2 al frente de mi_cola
O(1)  mi_cola.pop_back();            // Remover el fondo de mi_cola
O(1)  mi_cola.pop_front();           // Remover el frente de mi_cola
O(1)  mi_cola.front();               // Acceder al frente de mi_cola
O(1)  mi_cola.back();                // Acceder al fondo de mi_cola
O(1)  mi_cola.size();                // Tamaño de mi_cola (0)
O(1)  mi_cola.empty();               // Está mi_cola vacía? (true)
O(1)  mi_cola[3];                    // Acceso al tercer elemento.
```

Util cuando necesitas insertar al inicio de un arreglo, que usualmente sería O(n), y sin perder acceso aleatorio a elementos arbitrarios, a cambio de un **factor constante** (ver presentación de complejidad) mucho mayor.

CLUB DE PROGRAMACIóN COMPETITIVA UADY

priority_queue<T, Compare>

Implementa una cola de prioridad. El elemento con mayor prioridad sale primero, determinado por la función de comparación (Mayor por defecto).

```
        priority_queue<int> pq;
O(logn) pq.push(2);                 // insertar 2 en pq
O(logn) pq.push(1000);              // insertar 1000 en pq
O(logn) pq.top();                   // retornar tope de pq (1000)
O(logn) pq.pop();                   // eliminar el tope de pq
   O(1) pq.empty();                 // retorna si pq está vacía (false)
   O(1) pq.size();                  // tamaño de pq (1)
```

Casos de uso comunes son el algoritmo de Dijkstra y calcular el Minimum Spanning Tree de un grafo. En general es útil para procesar elementos según su valor respecto al resto, en lugar del orden en el que se recibieron.

CLUB DE PROGRAMACIóN COMPETITIVA UADY

set<T, Compare>

Implementa un conjunto sin elementos duplicados, ordenado de menor a mayor por defecto.

```
O(nlogn) set<int> mi_set = {4, 5, 7, 3, 4, 4};
    O(1) mi_set.size();                          // tamaño de mi_set (4)
 O(logn) mi_set.insert(2);                       // insertar 2 en mi_set
 O(logn) mi_set.erase(2);                        // remover 2 de mi_set
 O(logn) mi_set.count(4);                        // frecuencia de 4 en mi_set (1)
 O(logn) mi_set.lower_bound(3);                  // iterador al primer x >= 3 (3)
 O(logn) mi_set.upper_bound(3);                  // iterador al primer x > 3 (4)

         // mi_set : {3, 4, 5, 7}
```

Dado que preservan orden podemos comparar dos sets lexicográficamente con operadores de comparación en O(n)

CLUB DE PROGRAMACIóN COMPETITIVA UADY

unordered_set<T>

Implementa un conjunto sin elementos duplicados, sin algún orden en particular

```
O(n)* unordered_set<int> mi_set = {4, 5, 7, 3, 4, 4};
O(1)* mi_set.size();                                    // tamaño de mi_set (4)
O(1)* mi_set.insert(2);                                 // insertar 2 en mi_set
O(1)* mi_set.erase(2);                                  // remover 2 de mi_set
O(1)* mi_set.count(4);                                  // frecuencia de 4 en mi_set (1)

      // mi_set : {3, 4, 5, 7}
```

Tenemos un arreglo de enteros, ¿Cual es la cantidad de elementos distintos?

* Colisiones en la función de hash usada en su implementación pueden causar complejidades lineales en casos muy extremos. Sitios como codeforces suelen incluir testcases ingeniados específicamente para provocar esto.

CLUB DE PROGRAMACIóN COMPETITIVA UADY

multiset<T, Compare>

Implementa un conjunto que admite elementos duplicados, ordenado de menor a mayor por defecto.

```
O(nlogn) multiset<int> mi_set = {4, 5, 7, 3, 4, 4};
    O(1) mi_set.size();                          // tamaño de mi_set (6)
 O(logn) mi_set.insert(2);                       // insertar 2 en mi_set
 O(logn) mi_set.erase(2);                        // remover 2 de mi_set
 O(logn) mi_set.count(4);                        // frecuencia de 4 en mi_set (3)
 O(logn) mi_set.lower_bound(4);                  // iterador al primer x >= 4
 O(logn) mi_set.upper_bound(4);                  // iterador al primer x > 4

         // mi_set : {3, 4, 4, 4, 5, 7}
         //              |        |
         //           lb(4)    ub(4)
```

CLUB DE PROGRAMACIóN COMPETITIVA UADY

map<Key, T, Compare>

Funciona como un set, pero almacena pares llave-valor, ordenado de menor a mayor por defecto.

```
         map<int, string> mi_map;
O(logn)  mi_map[1] = “Hola, ”;     // Crear elemento con llave 1 y asignarle el valor “Hola, ”
O(logn)  mi_map[666] = “mundo”;    // Crear elemento con llave 666 y asignarle el valor “mundo”
   O(1)  mi_map.size();            // Tamaño de mi_map (2)
O(logn)  mi_map.count(1212);       // Frecuencia del elemento con llave 1212 (0)
O(logn)  mi_map.erase(444);        // Elimina 444, devuelve el numero de elementos eliminados (0)
```

Usar “[x]” crea el elemento con llave x y lo inicializa al elemento nulo del tipo de dato del valor.

CLUB DE PROGRAMACIóN COMPETITIVA UADY

unordered_map<Key, T>

Funciona como un set, pero almacena pares llave-valor, sin orden particular

```
        unordered_map<int, int> mi_map;
O(1)*   mi_map[1] = 2;              // Crear elemento con llave 1 y asignarle el valor 2
O(1)*   mi_map[666] = 3;            // Crear elemento con llave 666 y asignarle el valor 3
O(1)*   mi_map.size();              // Tamaño de mi_map (2)
O(1)*   mi_map.count(1212);         // Frecuencia del elemento con llave 1212 (0)
O(1)*   mi_map.erase(444);          // Elimina 444, devuelve el numero de elementos eliminados (0)
O(1)*   mi_map[666]++;              // Incrementa en 1 el valor asociado a la llave 666 (4)
O(1)*   mi_map[33]++;               // Crea el elemento con llave 33, lo inicializa en 0 y suma 1
```

Ahora no necesitamos declarar un arreglo de tamaño muy grande para implementar una cubeta o arreglo de frecuencias!

* Presenta el mismo problema que unordered_set

CLUB DE PROGRAMACIóN COMPETITIVA UADY

Otros contenedores

Usados prácticamente nunca, salvo casos extremadamente específicos, o que usualmente es posible solucionar con contenedores más comunes.

`multimap<Key, T, Compare>`	Analogo a multiset
`unordered_multimap<Key, T>`	unordered_map con duplicados
`unordered_multiset<Key, T>`	unordered_set con duplicados
`list<T>`	Implementa una lista doblemente enlazada
`forward_list<Key, T>`	Implementa una lista enlazada simple

CLUB DE PROGRAMACIóN COMPETITIVA UADY

Algoritmos

Son funciones que implementan algoritmos de uso común.

O(log min(a, b)))	`gcd(a, b)`	Retorna el MCD de a y b.
O(1)	`max(a, b)`	Retorna el máximo entre a y b.
O(1)	`min(a, b)`	Retorna el mínimo entre a y b.
O(1)	`swap(pos1, pos2)`	Intercambia los valores entre las posiciones pos1 y pos2.
O(n)	`reverse(begin, end)`	Invierte el orden en el intervalo comprendido entre begin y end.
O(nlogn)	`sort(begin, end, cmp)`	Ordena el intervalo usando la función de comparación cmp.
O(n)	`next_permutation(begin, end)`	Encuentra la siguiente permutación del intervalo.
O(n)	`prev_permutation(begin, end)`	Encuentra la permutación anterior del intervalo.

CLUB DE
PROGRAMACIóN
COMPETITIVA
UADY

Para consultar

en.cppreference.com

cplusplus.com/reference/


CLUB DE
PROGRAMACIóN
COMPETITIVA
UADY