Estructuras de datos

Miguel Ortiz

Programación competitiva para ICPC

Abril 2023 - Cochabamba, Bolivia

Introducción

- Una estructura de datos es una forma de almacenar datos en una computadora
- Es importante elegir la estructura de datos apropiada para cada problema
- Estructuras de datos sofisticadas y poderosas ya están implementadas en los lenguajes más populares
- Veremos algunas de las más importantes presentes en C++

Arreglos dinámicos

- Un arreglo dinámico puede cambiar su tamaño durante la ejecución del programa
- `vector` en C++
- Puede usarse casi como un arreglo normal

Arreglos dinámicos

Crear un vector vacío y añadir elementos:

```
#include <vector>
...
vector<int> v;
v.push_back(3); // [3]
v.push_back(2); // [3, 2]
v.push_back(5); // [3, 2, 5]
```

Arreglos dinámicos

Crear un vector vacío y añadir elementos:

```
#include <vector>
...
vector<int> v;
v.push_back(3); // [3]
v.push_back(2); // [3, 2]
v.push_back(5); // [3, 2, 5]
```

Se puede acceder a los elementos como en un arreglo normal:

```
cout << v[0] << '\n'; // 3
cout << v[1] << '\n'; // 2
cout << v[2] << '\n'; // 5
```

Arreglos dinámicos

La función `size` devuelve el tamaño del vector:

```
for (int i = 0; i < v.size(); i++) {
  cout << v[i] << " ";
}
// 3 2 5
```

Una forma más corta de iterar sobre los elementos:

```
for (auto x : v) {
  cout << x << " ";
}
// 3 2 5
```

Arreglos dinámicos

Inicializar un vector con 5 elementos:

```
vector<int> v = {3,2,5,1,4};
```

También se puede inicializar con un tamaño y un valor en cada posición:

```
// tamaño 10, lleno de 0s
vector<int> v(10);

// tamaño 10, lleno de -1s
vector<int> v(10, -1);

// tamaño 5, lleno de vectores vacios
vector<vector<int>> v(5, vector<int>());
```

Arreglos dinámicos

Otras funciones:

- `v.pop_back()`: elimina el último elemento
- `v.resize(n)`: cambia el tamaño del vector a *n*
- `v.assign(n, val)`: cambia el tamaño del vector a *n* y llena todas las posiciones con *val*
- `v.clear()`: elimina todos los elementos del vector

Ordenar un vector:

```
#include <algorithm>
...
sort(v.begin(), v.end());
```

Arreglos dinámicos

- Un `string` también es un arreglo dinámico
- Almacena caracteres y tiene funciones para manipularlos

```
string a = "abc";
string b = a + a;
cout << b << '\n'; // abcabc
b[2] = 'x';
cout << b << '\n'; // abxabc
string c = b.substr(2, 3);
cout << c << '\n'; // xab
c.pop_back();
cout << c << '\n'; // xa
```

Conjuntos

- Un `set` es una estructura de datos que almacena un conjunto de elementos (sin repeticiones)
- Se puede insertar y eliminar elementos, y verificar si un elemento está en el conjunto
- C++ tiene dos tipos de sets: `set` y `unordered_set`
- `set` está implementado como un árbol binario de búsqueda balanceado (operaciones en $O(\log n)$)
- `set` mantiene los elementos ordenados
- `unordered_set` está implementado como una tabla de hash (operaciones en $O(1)$ en promedio)

Conjuntos

- La función `insert` añade un elemento al conjunto
- La función `count` devuelve el número de ocurrencias de un elemento en el conjunto (0 o 1)
- La función `erase` elimina un elemento del conjunto

```
#include <set>
...
set<int> s;
s.insert(3);
s.insert(2);
s.insert(5);
s.insert(3); // no se inserta
cout << s.count(3) << '\n'; // 1
cout << s.count(4) << '\n'; // 0
s.erase(3);
cout << s.count(3) << '\n'; // 0
```

Conjuntos

- A diferencia de los vectores, no se puede acceder a los elementos de un conjunto por su índice
- Se puede iterar sobre los elementos de un conjunto de forma similar
- Función `size` devuelve el número de elementos en el conjunto

```
set<int> s = {2, 5, 6, 8};
cout << s.size() << '\n'; // 4
for (auto x : s) {
  cout << x << " ";
}
```

Conjuntos

- C++ también contiene `multiset` y `unordered_multiset`
- La única diferencia es que permiten elementos repetidos

```
multiset<int> s;
s.insert(5);
s.insert(5);
s.insert(5);
cout << s.count(5) << '\n'; // 3
```

- Se debe tener cuidado si solo se quiere eliminar un elemento

```
s.erase(s.find(5));
cout << s.count(5) << '\n'; // 2
s.erase(5);
cout << s.count(5) << '\n'; // 0
```

Mapas

- Un `map` es una estructura de datos que almacena pares de elementos (llave, valor)
- En un arreglo se accede a los elementos por su índice, en un mapa se accede por su llave
- La llave puede ser de cualquier tipo
- Hay dos implementaciones de mapas: `map` y `unordered_map`
- Un `map` está implementado como un árbol binario de búsqueda balanceado (operaciones en $O(\log n)$)
- `map` mantiene los elementos ordenados por su llave
- `unordered_map` está implementado como una tabla de hash (operaciones en $O(1)$ en promedio)

Mapas

- Se añaden elementos usando la llave como si ya existiera en el mapa
- Si se intenta acceder a un elemento que no existe, se crea un nuevo elemento con valor por defecto (0 para enteros)

```
#include <map>
...
map<string, int> m;
m["mono"] = 3;
m["gato"] = 5;
m["cuaderno"] = 2;
cout << m["mono"] << '\n'; // 3
cout << m["tecla"] << '\n'; // 0
```

Mapas

- La función `count` devuelve el número de ocurrencias de una llave en el mapa (0 o 1)
- La función `erase` elimina un elemento del mapa
- Se puede iterar sobre los elementos de un mapa de forma similar a un `set`

```
if (m.count("tecla")) {
  // llave existe
}
m.erase("mono");
for (auto p : m) {
  cout << p.first << " " << p.second << '\n';
}
// cuaderno 2
// gato 5
// tecla 0
```