Algorithm Unibo

Introduzione al progetto

Giovanni Spadaccini, Angelo Huang

Università di Bologna

t.me/algo_unibo

Table of Contents

In CP (Competitive Programming) andiamo ad utilizzare molte classi della STL di C++, andiamo a prendere un pò di famigliarità

STL C++

Oggi andremo a vedere un po' le basi dei contenitori di c++, e algoritmi dell'std.

NOTA: sembreranno una lunga lista di nomi, però la cosa bella è utilizzarle nella pratica!

Provate a runnare gli snippets mentre andiamo a mostrarli.

Cosa fare attenzione

- Complessità!

Cosa fare attenzione

- Complessità!
- Proprietà mantenute dalle strutture di dati
  - e.g. FIFO se uso la queue
  - e.g. LIFO se uso la stack
  - e.g. unicità degli elementi nei set.

Saranno i vostri amici migliori per i problemi di CP :D

Array dinamici

Vettori sono la struttura che ci permette di utilizzare gli array dinamici. Si possono accedere tramite la libreria vector (inclusa in bits/stdc++.h) Le operazioni principali:

- sono l'accesso ad un elemento costo $O(1)$
- Aggiunta di un elemento all'inizio costo ammortizzato, push_back $O(1)$
- Rimozione di un elemento all'inizio costo ammortizzato, pop_back() $O(1)$

```
vector<int> v(2,5); // {5,5}
v.push_back(4)// {5,5,4}
v.pop_back() // {5,5}
```

Queue e Stack

Queue LIFO, e Stack FIFO che hanno accesso, inserimento e rimozione tutti in $O(1)$ ammortizzato.

```
queue<int> q;
q.push(4);q.push(5);q.push(6);//{4,5,6}
q.front()// = 4
q.pop();//{5,6}
q.front()// = 5
stack<int> s;
s.push(4);s.push(5);s.push(6);//{4,5,6}
s.back()// =6
s.pop();//{4,5}
s.front()// = 5
```

Stack con i vector

Osservazione: Possiamo simulare gli stack con i vectors!

Infatti potremmo vedere la *push* della stack come un *push_back* del vector, stessa cosa con pop.

```
stack<int> s;
s.push(4);s.push(5);s.push(6);//{4,5,6}
s.pop();//{4,5}

// stessa cosa di
vector<int> v;
v.push_back(4); v.push_back(5); v.push_back(6);
s.pop_back();
```

Set

I set permettono l'inserimento, l'eliminazione e verificare la presenza in $O(\log n)$. Esistono anche i multiset per tenere il conto di ogni elemento quante volte è inserito, e gli unordered_set che usano gli hash e ammortizzano le operazioni a $O(1)$.

```
set<int> s;
s.insert(3);// {3}
s.insert(3);// {3}
s.insert(5);// {3,5}
cout << s.count(3) << "\n"; // 1
cout << s.count(4) << "\n"; // 0
s.erase(3); // {5}
s.insert(4); // {4,5}
cout << s.count(3) << "\n"; // 0
cout << s.count(4) << "\n"; // 1
```

Heap

Priority queue è la struttura che in c++ implemente la heap, questa struttura permette l'inserimento e la rimozione in $O(\log n)$ e accedere al massimo in $O(1)$.

```
priority_queue<int> q;
q.push(3);  q.push(5); // {5,3}
q.push(7); q.push(2);  // {7,5,3,2}
cout << q.top() << "\n"; // 7
q.pop(); // {5,3,2}
q.push(6); // {6,5,3,2}
cout << q.top() << "\n"; // 6
```

Heap con set

Possiamo anche utilizzare i set come se fossero dei heap, ma gli elementi devono essere distinti fra di loro!

```
set<int> s;

// inserimento heap è come inserire nel set.
s.insert(1);

// togliere dalla cima:
s.erase(s.begin());
```

Map

Le map permettono di indicizzare da attraverso un valore un altra variabile, l'eliminazione e verificare la presenza e l'accesso in $O(\log n)$.
Esistono unordered_map che usano gli hash e ammortizzano le operazioni a $O(1)$. anche se solitamente non vengono mai utilizzati (per certi input sono lineari)

```
map<string,int> m;
m["monkey"] = 4;
m["banana"] = 3;
m["harpsichord"] = 9;
cout << m["banana"] << "\n"; // 3
if (m.count("aybabtu")) {
    // key exists
}
for (auto x : m) {
    cout << x.first << " " << x.second << "\n";
}
```

Iteratori

Gli iteratori sono delle strutture intorno ai pointers. Aiutano a regolare l'accesso alle strutture (così non vai a fare segfault ma hai un errore :D).
Su tutte le classi precedenti possiamo trovare queste funzioni:

- .begin(), ritorna l'iteratore all'inizio del container
- .end(), ritorna l'iteratore che punta alla memoria dopo il container
- .rbegin(), simile a begin(), ma cominci al contrario.
- .rend(), ritorna l'iteratore che punta alla memoria prima il container

Questi si comportano proprio come se fossero dei pointer, puoi andare al prossimo o al precedente con gli operatori ++, o - -;
Demo con i set: https://replit.com/@angelohuang2/C-For-CP#set.cpp

Operazioni sugli array

Operazioni sugli array/vector.

- `find([begin_iterator], [end_iterator], [item])` - O(n)
- `sort([begin_iterator], [end_iterator])` - O(log n)
- `lower_bound([begin_iterator], [end_iterator], [item])` - O(log n)
- `upper_bound([begin_iterator], [end_iterator], [item])` - O(log n)

Esempi: https://replit.com/@angelohuang2/C-For-CP#stl_functions.cpp

Esercizi

Esercizi

- Repetitions: https://cses.fi/problemset/task/1069
- Subarray Sums I: https://cses.fi/problemset/task/1660
- Restaurant Customers: https://cses.fi/problemset/task/1619/
- Movie Festival: https://cses.fi/problemset/task/1629
- Exponentiation: https://cses.fi/problemset/task/1095
- https://codeforces.com/contest/1791/problem/D
- Subarray Sums II: https://cses.fi/problemset/task/1661