cpshCP 03. STL/basic DS

3-2. standard template library(STL)

先備知識

什麼是STL?

標準模板庫(standard template library), 簡稱STL,

是C++中集合了常見資料結構和演算法的一個函式庫,

裡面有很多現成的資料結構可以直接用(ex. queue/stack),

就不用每次用到時都要手刻一個了。

(不過還是建議要去了解它們的運作原理/實作方式, 而不是只會用而已)

模板(template)

在學習STL之前，我們要先了解模板(template)究竟是什麼。

在設計一些容器的時候(ex. stack)，我們不會事先了解使用者到底想存什麼型別的資料在裡面，可能是字元，也可能是數字、字串、指標等等的...

如果要寫出能夠應付各種儲存型別的容器，

用一般的方式會需要寫好多份類似的code，

為了處理這種狀況，C++就衍生出了”模板”的概念，將型別參數化。

模板(template)

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

template<class T>
class dot {
  T x;
  T y;
};

int main() {
  dot<int> d1;  //宣告一個int型別的dot class
  dot<double> d2; //宣告一個double型別的dot class
}
```

如何使用模板(template)

通常一個STL的容器宣告方式如下

```
class<T> name;
```

其中container代表容器名稱、T(type)代表容器所儲存的型別、name則是賦予容器的名稱。

如何使用模板(template)

舉例來說，我今天想要宣告一個stack，那程式會像這樣。

```
#include<stack>

using namespace std;
int main() {
  stack<int> s1;  //a stack storing integer
  stack<char> s2;  //a stack storing char
}
```

迭代器(iterator)

迭代器是用來遍歷容器的一個東西，你可以把它想成比較有智慧的指標(?

你可以對一個迭代器使用*取得他指向的資料(ex. *ite),

也可以用++/--去移動這個迭代器到下一個/前一個位置(ex. ite++),

在一些可以隨機存取的容器中甚至能直接當成一般的指標用。

(ex. *(ite + 7))

迭代器(iterator)

宣告的方式則是

```
container::iterator name;
```

順帶一提, STL中的容器大部分都會提供.begin()和.end()函式

來取得指向該容器**第一個元素/最後一個元素的下一個位置**的迭代器喔~

迭代器(iterator)

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

using namespace std;
int main() {
  set<int> se;
  se.insert(1);
  se.insert(2);
  se.insert(-5);
  se.insert(6);

  //用迭代器遍歷set, 直到ite指到set的尾端
  for(set<int>::iterator ite = se.begin(); ite != se.end(); ite++)
    cout << (*ite) << '\n';
  //用C++的auto feature自動幫你找型別
  for(auto ite = se.begin(); ite != se.end(); ite++)
    cout << (*ite) << '\n';
  //用for-each遍歷set
  for(int X : se) {
    cout << X << '\n';
  }

  return 0;
}
```

一些STL的資料結構

STL中容器都有的函式

size() : 回傳該容器目前的元素數量

empty() : 回傳一個布林值，如果是true代表容器是空的，反之。

stack(堆疊)

標頭: <stack>

宣告: stack<T> name;

常用函式:

push(T data) : 把一個型別T的資料放到stack頂端。

pop() : 把stack最頂端的資料移除。

top() : 回傳stack最頂端的資料。

以上函式都是O(1)。

```cpp
#include<stack>
#include<iostream>

using namespace std;
int main() {
  stack<int> sta;

  sta.push(1);
  sta.push(3);
  sta.pop();
  cout << sta.top() << '\n';
  //output: 1

  return 0;
}
```

queue(佇列)

標頭: <queue>

宣告: queue<T> name;

常用函式:

push(T data) : 把型別為T的資料
放到queue的後端。

pop() : 把queue中最前端的資料移除。

front() : 回傳queue中最前端的資料。

以上函式都是O(1)。

```cpp
#include<queue>
#include<iostream>

using namespace std;
int main() {
  queue<int> que;

  que.push(3);
  que.push(4);
  que.push(2);
  que.pop();

  cout << que.front() << '\n';
  //output: 4

  return 0;
}
```

deque(雙向佇列)

double-ended queue，簡稱deque，

是可以在兩端插入/刪除資料的資料結構。

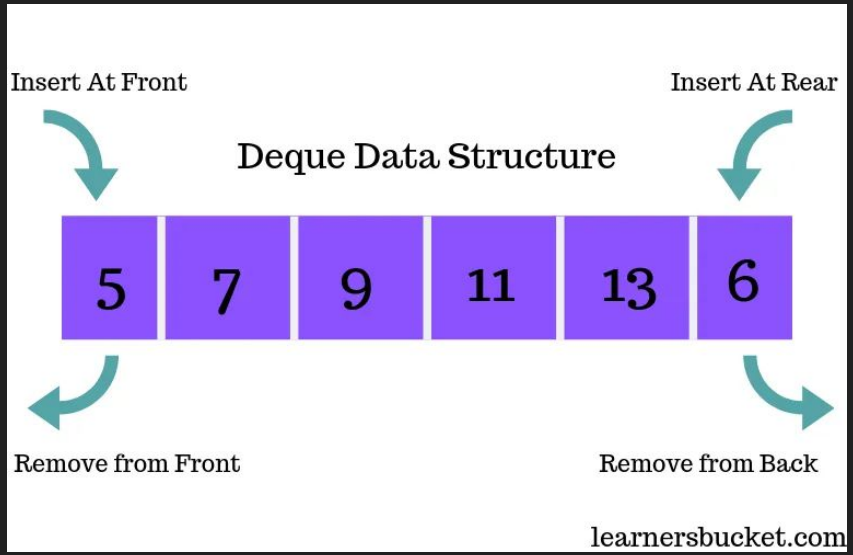
標頭: <deque>

宣告: deque<T> name;

常用函式:

push_back(T data) : 把型別為T的資料放到deque後端。

push_front(T data) : 把型別為T的資料放到deque前端。



deque(雙向佇列)

常用函式:

pop_back() : 把deque最後端的資料移除。

pop_front() : 把deque最前端的資料移除。

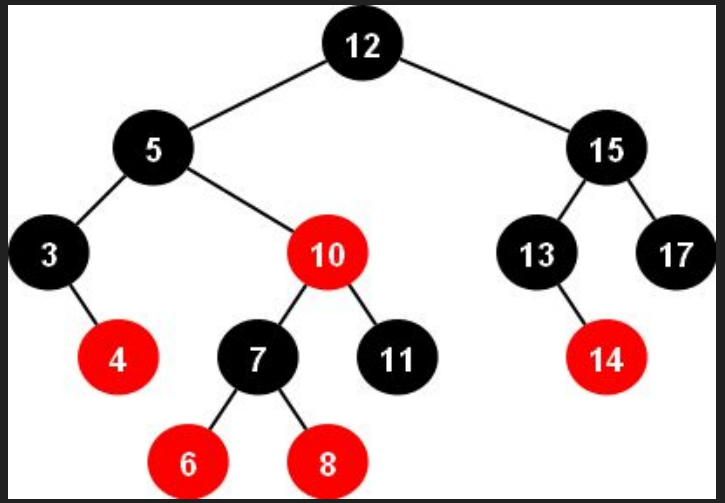
back() : 回傳deque最後端的元素。

front() : 回傳deque最前端的元素。

以上函式都是O(1)。

```
#include<deque>
#include<iostream>

using namespace std;
int main() {
  deque<int> deq;

  deq.push_back(2);
  deq.push_front(3);
  deq.push_back(7);
  deq.pop_back();

  cout << deq.front() << ' '
       << deq.back() << '\n';
  //output: 3 2

  return 0;
}
```

set(集合)



set, 是一個能夠在O(lg size)做到

1. 插入/移除set中的元素
2. 尋找set中是否有某個元素x
3. 尋找set中第一個大於/小於x的元素是多少

的一個資料結構,

實作上是使用red-black tree(紅黑樹), 由於這東西過於複雜,

動輒幾百行, 所以這邊只教怎麼使用他, 不提實作細節。

set(集合)

標頭: <set>

宣告: set<T> name;

常用函式:

insert(T data) : 插入一個型別T的資料到set裡面，複雜度O(lg size)。

erase(T data) : 把set中為data的資料移除，複雜度O(lg size)。

clear() : 清除set中所有資料。複雜度O(size)。

set(集合)

常用函式:

lower_bound(T data) : 回傳指向第一個”**大於等於**”data的資料的迭代器，若不存在就回傳end()，複雜度O(lg size)。

upper_bound(T data) : 同lower_bound()，只是變成”**大於**”。

find(T data) : 回傳指向data的迭代器，若不存在就回傳end()，複雜度O(lg size)。

begin() : 回傳指向set第一個元素的迭代器，複雜度O(1)。

end() : 回傳指向set最後一個元素往後一格的迭代器，複雜度O(1)。

set(集合)

ps. 要獲得小於x的第一個元素可以用--upper_bound(x)，小於等於同理。

```cpp
#include<set>
#include<iostream>

using namespace std;
int main() {
  set<int> se;

  se.insert(1000);
  se.insert(-1234);
  se.insert(789);
  se.erase(-1234);

  int val = 789;
  if (se.find(val) == se.end())
    cout << val << " is in the set\n";
  else
    cout << val << " is not in the set\n";

  if (se.lower_bound(val) != se.end())
    cout << (*se.lower_bound(val)) <<
         << "is the first element >= " val << '\n';
  //output: 789 is the first element >= 789

  if (se.upper_bound(val) != se.end())
    cout << (*se.upper_bound(val)) <<
         << "is the first element > " val << '\n';
  //output: 1000 is the first element > 789

  return 0;
}
```

CF 1293A - ConneR and the A.R.C. MarkLand-N

有一個N層樓的建築，每一層都有一間餐廳，但是其中有k層的餐廳正在整修，

而ConneR目前在第s層，請問ConneR最少要爬幾層樓才能找到一個沒有整修的餐廳用餐呢?

(2 <= N <= 10 ^ 9, 1 <= k <= min(N - 1, 1000))

https://codeforces.com/problemset/problem/1293/A

CF 1293A - ConneR and the A.R.C. MarkLand-N

這題有一個簡單的做法, 就是把那k層樓用陣列存起來,

並從s同時往上往下搜, 每次要詢問某層樓有沒有在整修時就暴力掃過整個陣列,

每次詢問要O(k), 最多詢問O(k)次, 整體複雜度O(k^2)。

CF 1293A - ConneR and the A.R.C. MarkLand-N

這題有一個簡單的做法, 就是把那k層樓用陣列存起來,

並從s同時往上往下搜, 每次要詢問某層樓有沒有在整修時就暴力掃過整個陣列。

每次詢問要O(k), 最多詢問O(k)次, 整體複雜度O(k^2)。

我們能不能做到更好?

CF 1293A - ConneR and the A.R.C. MarkLand-N

如果我們不用陣列儲存整修的餐廳，改成用set，

由於set支援O(lg size)的插入和查詢，所以一次的查詢會降為O(lg k)，

整體複雜度變成O(k*lg k)。

map

set的強化版，同時儲存鍵(key)和值(value)，

除了set擁有的功能以外，

還能在O(lg size)的時間找到對應一個key的value。

```cpp
#include<map>
#include<iostream>

using namespace std;
int main() {
  map<int, int> ma;

  ma[2] = 5;
  ma[7] = 1;

  cout << ma[2] << '\n';
  //output: 5

  return 0;
}
```

map

標頭: <map>

宣告: map<K, T> name; (K代表鍵的型別, T代表值的型別)

常用函式:

- set所擁有的函式
- map[key] : 回傳對應key的value, 複雜度O(lg size)。
- map[key] = data : 把對應key的value設為data, 複雜度O(lg size)。

CF 903C - Boxes Packing

給你N個箱子，其中第i個箱子大小為a_i，

你可以任意地把較小的箱子放到較大的箱子裡，

被放進去的箱子被稱為”看不到的箱子”。

請問這些箱子在經過收納後，最少可以有多少個”看得見的箱子”?

(1 <= N <= 5000, 1 <= a_i <= 10^9)

https://codeforces.com/contest/903/problem/C

CF 903C - Boxes Packing

可以觀察到，如果有一些不同大小的箱子各一，那它們一定可以收納成一個箱子。

因此，答案就是所有大小的箱子中，出現最多次的那一個大小的箱子數。

但是實作上要怎麼知道哪個大小的箱子出現最多次呢?

CF 903C - Boxes Packing

可以觀察到，如果有一些不同大小的箱子各一，那它們一定可以收納成一個箱子。

因此，答案就是所有大小的箱子中，出現最多次的那一個大小的箱子數。

但是實作上要怎麼知道哪個大小的箱子出現最多次呢?

開一個跟值域一樣大的陣列，大小為i的箱子數就記錄在arr[i]?

CF 903C - Boxes Packing

可以觀察到，如果有一些不同大小的箱子各一，那它們一定可以收納成一個箱子。

因此，答案就是所有大小的箱子中，出現最多次的那一個大小的箱子數。

但是實作上要怎麼知道哪個大小的箱子出現最多次呢?

開一個跟值域一樣大的陣列，大小為i的箱子數就記錄在arr[i]?

很可惜，這樣的複雜度是O(max a_i) (a_i <= 10 ^ 9)，沒辦法通過時限。

CF 903C - Boxes Packing

可以觀察到，如果有一些不同大小的箱子各一，那它們一定可以收納成一個箱子。

因此，答案就是所有大小的箱子中，出現最多次的那一個大小的箱子數。

但是實作上要怎麼知道哪個大小的箱子出現最多次呢?

其實只要用map，把key當成箱子大小，value當成大小為key的箱子數，

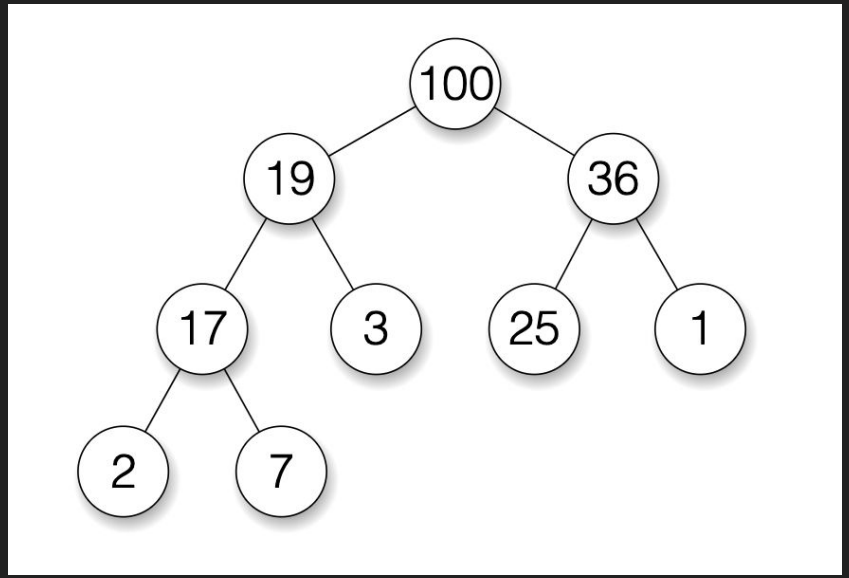
就可以輕鬆地記錄各種大小的箱子數量是多少，

最後再用for-each遍歷整個map尋找最大值即可，整體複雜度O(n * lg n)。

priority queue(優先佇列)

priority queue是用來快速詢問容器中最小/最大的元素是多少的一個資料結構,

實作上是使用heap(堆積), 有興趣的可以查查看資料, 實作並不會很難。



priority queue(優先佇列)

標頭: <queue>

宣告: priority_queue<T, container, comparator>

其中container代表作為priority_queue的容器，一般都是使用vector<T>，

而comparator則是用來比較大小的函式，通常使用greater<T>/less<T>。

宣告範例:

查詢最小值的優先佇列: priority_queue<T, vector<T>, greater<T> > name;

查詢最大值的優先佇列: priority_queue<T, vector<T>, less<T> > name;

priority queue(優先佇列)

常用函式:

push(T data) : 插入型別為T的資料T進入priority_queue，複雜度O(lg size)。

pop() : 把最小/最大的資料從priority_queue中移除，複雜度O(lg size)。

top() : 回傳priority_queue中最小/最大的資料，複雜度O(1)。

clear() : 清空priority_queue，複雜度O(size)。

priority queue(優先佇列)

```cpp
#include<queue>
#include<iostream>

using namespace std;
int main() {
  priority_queue<int, vector<int>, less<int> > mxPq;
  //宣告一個最大堆
  priority_queue<int, vector<int>, greater<int> > mnPq;
  //宣告一個最小堆

  mxPq.push(3);
  mxPq.push(5);
  mxPq.push(-1);
  mxPq.pop();

  cout << mxPq.top() << '\n';
  //output: 3

  mnPq.push(3);
  mnPq.push(5);
  mnPq.push(-1);

  cout << mnPq.top() << '\n';
  //output: -1

  return 0;
}
```

vector(動態陣列)

由於C++提供的一般陣列在編譯期間就固定大小了,

有時候用起來不太方便, 因此STL提供了vector這個能自由改變大小的動態陣列。

ps. 之後在學圖論的時候會經常用到, 建議要熟練。

vector(動態陣列)

標頭: <vector>

宣告: vector<T> name(size, initial value);

常用函式:

[index] : 存取索引為index的值，複雜度O(1)。

push_back(T data) : 增加一格到vector尾端，並設值為data，複雜度O(1)。

pop_back() : 把vector最後一格移除，複雜度O(1)。

clear() : 清空vector，複雜度O(size)。

vector(動態陣列)

常用函式:

resize(size_t size, T initial value) : 將vector配置到大小剛好為size,

多的格子移除、不夠的格子則是分配到足夠, 並設為value, 複雜度O(size)。

begin() : 回傳指向第一個元素的迭代器, 複雜度O(1)。

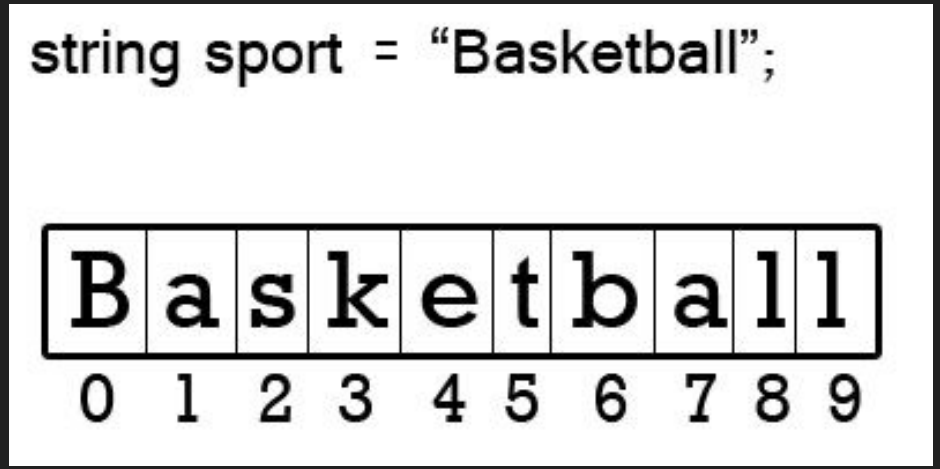
end() : 回傳指向最後一個元素的下一格的迭代器, 複雜度O(1)。

vector(動態陣列)

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

using namespace std;
int main() {
  vector<int> vec(3, 1);
  //{1, 1, 1}

  vec.push_back(2);
  vec.push_back(5);
  //{1, 1, 1, 2, 5}

  vec.pop_back();
  //{1, 1, 1, 2}

  for(int i = 0; i < vec.size(); i++)
    cout << vec[i] << ' ';
  cout << '\n';
  //output: 1 1 1 2

  return 0;
}
```

string(字串)

基本上就是vector<char>做成的，加上一些額外的功能。



string(字串)

標頭: <string>

宣告: string name;

輸入/輸出:

cin >> string; (輸入不含空格的字串)

getline(cin, str); (輸入一整行, 包含空格)

cout << string;

string(字串)

常用函式:

所有vector擁有的函式

+= str：把str加到string的尾端，複雜度O(len(str))。

[index]：存取string中索引為index的字元，複雜度O(1)。

substr(size_t pos, size_t len)：回傳從index pos開始、長度為len的string。

string(字串)

```cpp
#include<string>
#include<iostream>

using namespace std;
int main() {
  string str = "abcd";
  str += "efg";

  cout << str << '\n';
  //output: abcdefg
  cout << str.substr(2, 3) << '\n';
  //output: cde

  return 0;
}
```

其他常用的STL工具

sort(排序)

標頭: <algorithm>

函式: sort(iterator begin, iterator end)

將給定範圍[begin, end)中的資料由小到大排序，複雜度O(size * lg size)。

sort(排序)

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

using namespace std;
int main() {
  vector<int> vec(5);
  for(int i = 0; i < 5; i++)
    vec[i] = 5 - i;
  //{5, 4, 3, 2, 1}

  sort(vec.begin(), vec.end());
  //{1, 2, 3, 4, 5}

  int arr[5] = {3, 1, 2, 5, 7};
  sort(arr, arr + 5);
  //{1, 2, 3, 5, 7}

  return 0;
}
```

pair

把兩個變數綁在一起的一個工具(ex. 儲存二維座標x, y的時候)。

標頭: <utility>

宣告: pair<T1, T2> name;

常用函式:

first : 存取第一個資料。

second : 存取第二個資料。

make_pair(T data1, T data2) : 製作一個pair。

pair

範例(儲存一個二維平面上的點):

```cpp
#include<utility>
#include<iostream>

using namespace std;
int main() {
  int x, y;
  cin >> x >> y;
  pair<int, int> dot = make_pair(x, y);

  cout << dot.first << ' ' << dot.second << '\n';
  //output: x y

  return 0;
}
```

swap(交換)

一個交換元素時方便的小工具, 讓你不用為了交換寫三行code。

函式: swap(T data1, T data2)

範例:

```cpp
#include<utility>
#include<iostream>

using namespace std;
int main() {
  int a = 3, b = 5;
  swap(a, b);

  cout << a << ' ' << b << '\n';
  //output: 5 3

  return 0;
}
```

結語

這次講的STL工具可以不用一口氣全部記下來,

有用到再回來查, 就會慢慢熟悉了。

上面提到的都只是最常用的東西,

想知道更多/更詳細關於STL的東西可以上cppreference.com查詢。