Advanced Competitive Programming

國立成功大學ACM-ICPC程式競賽培訓隊
nckuacm@imslab.org

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

Week 2
I/O & Standard Template Library

STL 和 Coding 小知識

Outline

- 基礎競程姿勢
- 基礎 I/O
  - I/O 加速
- 常用 STL
  - vector, string, queue, stack, list, set, map, priority_queue, sort 超多

基礎競程姿勢

基礎競程姿勢

記憶體**空間**的規範各競賽都不相同

通常得考慮：

- 遞迴深度
- 使用的變數多寡
- 程式碼長度

基礎競程姿勢

而競賽都以秒為單位去做**時間**限制

- 例如 1 秒、 3 秒、10 秒

基礎競程姿勢

- 基本上競程選手幾乎使用 C++，較不會使用 C、Python、Java、Go 等語言
- 有幾點原因：
  - C++ 很快
  - C++ 相對 C 有很多寫好的資料結構可以直接用
  - C++ 在幾乎所有比賽中都能使用
  - C++ 對於操作陣列、指標的難度最低
- 但若有其他語言更為適合的情況，也可以試試

基礎競程姿勢--常見資料型態

基礎競程姿勢--常見資料型態

- 需要注意常見的資料型態範圍
  - 就算是選手常因沒有小心估算範圍而得到 WA 或 RE
- int x: $|x| \leq 2 \times 10^9$
- long long x: $|x| \leq 9 \times 10^{18}$
- ~~float~~ x: 共 6 位精確度
  - 例如 123.456789 後面的 789 是不準確的
- double x: 共 15 位精確度

基礎競程姿勢--常見資料型態

- 需要注意所要使用的記憶體量，基本上陣列不要超過 10^6
  - 若需要超過請仔細計算
- 程式的記憶體用量也可以稱作空間複雜度
  - 同樣可以套你學過的 $big\ O$ 表示
- 如果開很大需要開全域變數

基礎競程姿勢--演算法的效率

基礎競程姿勢--演算法的效率

- 2 倍、3 倍、甚至 10 倍的常數倍優化不是競賽時最優先考慮的要點
- 我們所設計的演算法必須根據輸入規模 N 而定。

基礎競程姿勢--演算法的效率

需要仔細估算所使用演算法的效率

- 同樣可以套你學過的 $big\ O$ 表示

Big O

- $Big\ O$ 表示法

$$f(N) = O\big(g(N)\big)$$
$$\Leftrightarrow \exists M, c > 0. \forall N > M. |f(N)| \ \leq c \cdot |g(N)|$$

- 意思是說在 N 足夠大的時候，存在正數 c 使得 $c \cdot |g(N)|$ 大於等於 $|f(N)|$

Big O

- 例如估計的時間函數: $f(x) = x^2 + x + 1$
- 在 x 很大的時候，主要影響整個函數值的大小是平方項
- 這時我們可以說 $f(x) = O(x^2)$

Big O

- 設輸入規模為 N，常見的複雜度有：
- $O(1) \leq O(logN) \leq O(N) \leq O(NlogN) \leq O(N^k) \leq O(k^N) \leq O(N!) \leq O(N^N)$
- 其中 k 為常數 (不隨輸入規模成長)

俗話說：大約 10^7 以內都算安全

合理的時間複雜度

假設題目:

規模為 N

而你:

設計出的演算法複雜度為 $O(N^2 logN)$

合理的時間複雜度

$x = N^2 logN$ 得落在 $x \leq 10^7$左右

這樣的複雜度才不容易 TLE

- 也就是說如果 $N = 10^5$
- 那就得重新設計演算法
- 因為此時 $x = 10^{10} \times \log(10^5)$ 超大ggggg

基礎 I / O

基礎 I / O

- 一般常見有兩種
  - cin/cout
  - scanf/printf

基礎 I / O

- 若你習慣使用 cin/cout 需在 main 的一開始加入：
  - ios::sync_with_stdio(false);
  - cin.tie(0);
- 因為預設中在執行 cin 之前 cout 會直接 flush (將緩衝區的內容輸出到螢幕)
- C++ 有個換行操作是 std::endl，將會強制進行 flush，建議也用'\n'取代
- 並請不要混用 cin/cout、scanf/printf
- 如果你使用 scanf/printf 請跳過本頁

基礎 I / O

- 所以你的 code 可能一開始會長這樣

```cpp
#include <bits/stdc++.h>
#define endl '\n'

using namespace std;
int main() {
    int n, m, i, j, k;
    ios::sync_with_stdio(false);
    cin.tie(0);

    return 0;
}
```

常用 STL

常用 STL

- STL 全名 Standard Template Library
  - 簡單來說就是幫你寫好很多東西讓你加快~~變笨~~
- STL 可以套在 STL 裡面
  - 然而 STLSTL 又可以套在 STL 裡面
    - 然而 STLSTLSTL 又可以套在 STL 裡面
      - 然而 STLSTLSTLSTL又可以套在 STL 裡面
        - 然而 STLSTLSTLSTLSTL 又可以套在 STL 裡面
- 這裡只教基礎用法，比較詳細的請看課程教材或文件

Vector

- vector 就是比較好用的陣列
- 他的用法與許多 STL 的資料結構很像
- #include <vector> // 其他 STL 的引入方法類似
- 宣告：vector<int> v; // 不用陣列大小
- 把東西丟進去 v.push_back(123);
- v:

Index	0	1	2	3	4
Value	123				

- 你可以用 v[0] 來取得這個 123

Vector

繼續

```
v.push_back(23);
v.push_back(3);
```

v 會變成

Index	0	1	2	3	4
Value	123	23	3		

Vector

`v.size()` 取得大小

Index	0	1	2	3	4
Value	123	23	3		

`v.size() == 3`

Iterator(迭代器)

- 用法類似指標，可以說是指標的加強版
- 常用於遍歷容器，如 vector、map、list

Iterator 用法

```
vector<int> v;
vector<int>::iterator iter;
for(iter = v.begin(); iter != v.end(); iter++) {
  cout << *iter << endl;
}
// v.begin(): v 的起始地址
// v.end(): v 的末端地址 + 1 (由於左閉右開)
```


Made by 培訓團隊群 CCNS.

String

- string 就是比較好用的 char str[n]
- 宣告：string s; // 不用字串大小
- 你可以直接 cin >> s 或是 cout << s
- 也可以直接指定 s = "ccns"

Index	0	1	2	3	4
Value	c	c	n	s	

- 你可以用 s[0] 來取得'c'

String

Index	0	1	2	3	4
Value	c	c	n	s	

- 你可以用 s.length() 來取得 s 的長度
- s.length() == 4

String

- vector 擁有的自帶函數 string 也有，如：
  - string::push_back()
  - string::assign()
- 也可以把 string 轉型態成 char[] 的字串型態：
  - string::c_str()

- string 也可以直接相加 s = stra + strb
- 甚至可以直接排序(之後會提到)

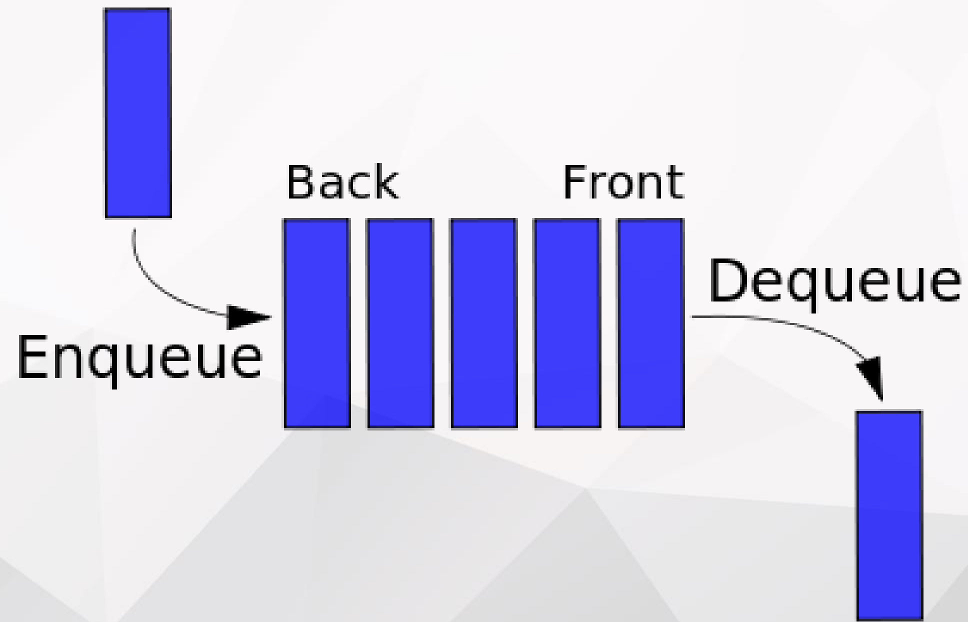
getline

```
cin.ignore();

while (getline(cin, s)) {
  if (s.empty()) break;
    :
    .
}
```

Queue(佇列)

- 先進先出，如排隊般的特性
- 與接下來的 Stack、Linked List 有類似的操作
- push()
- pop()
- front()
- empty()
- size()



Queue 操作

```
#include <queue>
queue<string> q;
q.push("c");
q.push("c");
q.push("n");
q.push("s");
q.front(); // return "c"
q.pop();
```

Queue 操作

```
while(!q.empty()) {
    // do something
}
while(q.size()) {
    // do something
}
```

Queue 例題

- Uva 10935
- 題目說明：

現在有一疊牌，每當輸入一個 n (n ≤ 50) 代表牌有 n 張，依序編號從 1 到 n， 每次操作會將最頂端的牌的編號輸出，並拿掉該牌，然後將新成為頂端的牌放到牌的最末端，直到剩下最後一張牌，輸出最後一張牌的編號。

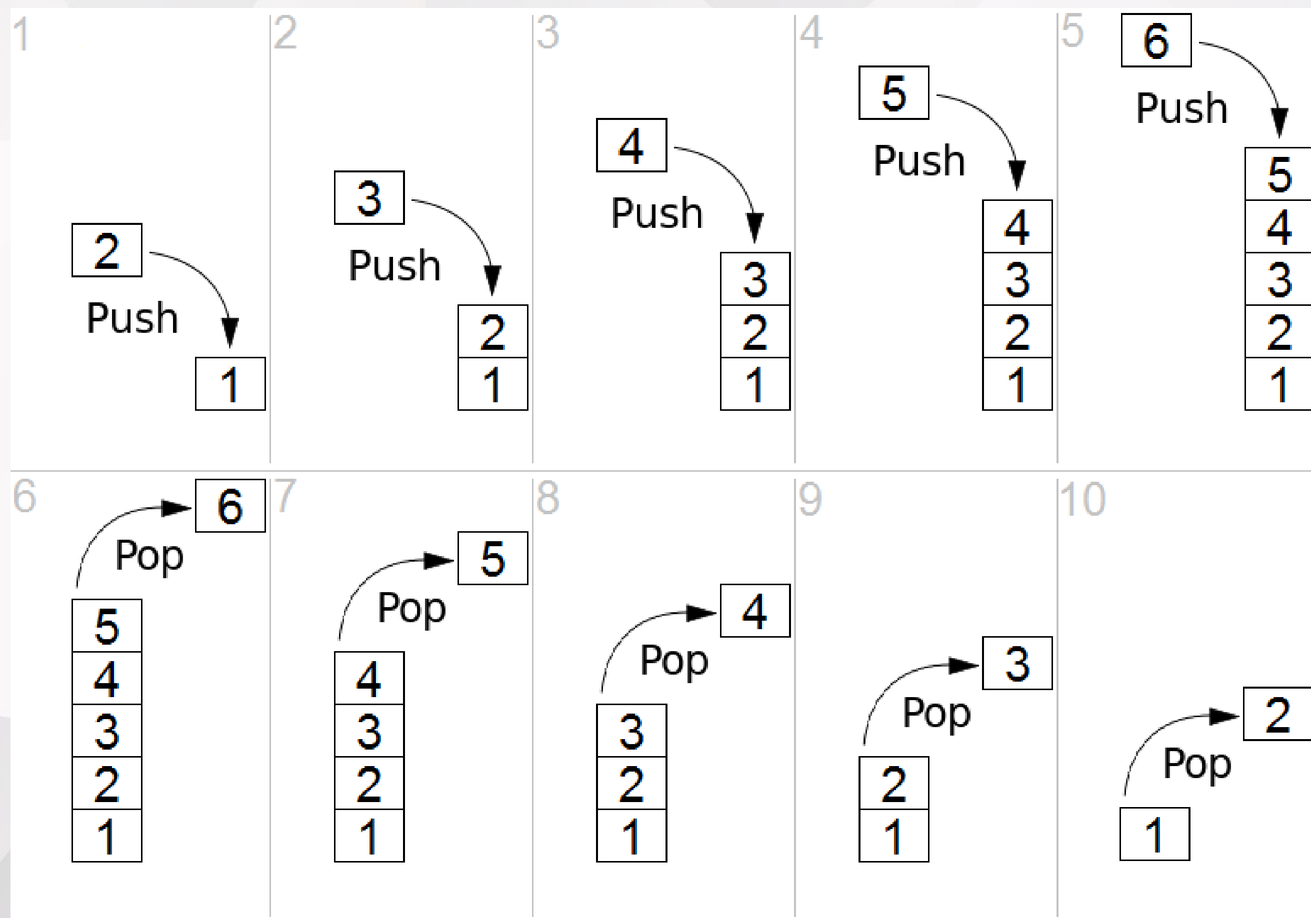
範例輸入輸出

```
Input:
  7
Output:
  Discarded cards: 1, 3, 5, 7, 4, 2
  Remaining card: 6
```

Stack(堆疊、堆棧、棧)

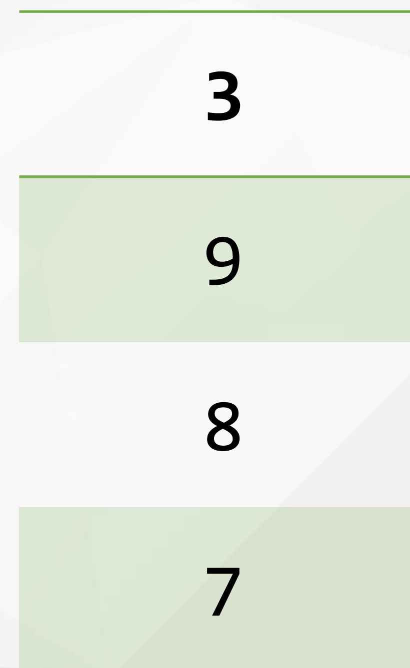
- 後進先出，就像把東西疊起來
- push()
- pop()
- top()
- empty()
- size()

Stack



Stack 操作

```
#include <stack>
stack<int> s;
s.push(7);
s.push(8);
s.push(9);
s.push(3);
s.top();  // return 3
s.pop();
s.top();  // return 9
```



Stack 操作

```
while(!s.empty()) {
  // do something
}
while(s.size()) {
  // do something
}
```

Stack 例題

- Uva 514
- 題目說明：

[開火車開起來]

範例輸入輸出

Input:

```
5
1 2 3 4 5
5 4 1 2 3
```

Output:

```
Yes
No
```

List(鏈結串列)

- 愛怎麼進出都可以
- push_back()
- pop_back()
- push_front()
- pop_front()
- back()
- front()
- insert()
- erase()

List 操作

```
#include <list>
list<int> li;
li.push_back(3);   // 3
li.push_back(4);   // 3<->4
li.push_front(2); // 2<->3<->4
li.pop_front();    // 3<->4
```

List 操作

```
list<int>::iterator iter;
for(iter = li.begin(); iter != li.end(); iter++) {
  cout << *iter << endl;
}
```

List 例題

『陸行鳥大賽車』

開車開起來

set

- 是數學上使用的集合結構
- 元素不會重複，例如 {1, 2, 2, 3, 1} = {1, 2, 3}

set

- 是數學上使用的集合結構
- 元素不會重複，例如 {1, 2, 2, 3, 1} = {1, 2, 3}
- 元素之間必須有序(可比大小)
- 插入、刪除、查詢 的複雜度都為 O(log N)

set member function

`insert(a)`
插入元素 a

`erase(l, r)`
把 [l, r) 位置上的元素移除，其中 l, r 型態為 `iterator`

`erase(a)`
移除元素 a

set member function

`find(a)`
指向元素 a 的迭代器，若 a 不存在則回傳 `end()`

`count(a)`
元素 a 是否存在

set 遍歷

```
int mints[] = {75,23,65,42,13,75,65};
set<int> myset(myints,myints+7);

for (auto it = myset.begin(); it != myset.end(); it++)
  std::cout << ' ' << *it;
```

Output:

13 23 42 65 75 // 按照順序

CF 1157A Reachable Numbers

CF 1157A Reachable Numbers

為了產生不曾出現過的數字
只能持續套用函數 f

CF 1157A Reachable Numbers

為了產生不曾出現過的數字
只能持續套用函數 f

所以考慮使用 f 的複雜度
在除以 10 以前，最多也只會加 9 次 1
所以對於 n，不斷操作 f 複雜度為 O(log n)

Made by 培訓團隊群 CCNS.

CF 1157A Reachable Numbers

若遇到**已經見過**的數字，則不會再出現不曾見過的數字了

```
set<int> S;
while(!S.count(n)) {
  S.insert(n);

  n++;
  while(n % 10 == 0) n /= 10;
}
printf("%d\n", S.size());
```

map

- 是鍵(key)值(value)對(pair)的一種實作

- 每個元素都是 pair
- 與 set 一樣，這些 pair 要有序

- 插入、刪除、查詢 的複雜度都為 O(log N)

map

```
map<char,string> mymap;
```

資料型態

map

```
map<char,string> mymap;

mymap['a']="an element";
mymap['b']="another element";
mymap['c']=mymap['b'];
```

map

```
map<char,string> mymap;

mymap['a']="an element";
mymap['b']="another element";
mymap['c']=mymap['b'];

cout << mymap['b']; //another element
cout << mymap['a']; //an element
```

map v.s. array(c style)

```
mymap['b']="apple";
cout << mymap['b’];
```

新增與取值的操作為O(logN)

map v.s. array(c style)

```
mymap['b']="apple";
cout << mymap['b’];
```

新增與取值的操作為O(logN)

```
a[0]=18;
cout << a[0];
```

新增與取值的操作為O(1)

map v.s. array(c style)

*用**非負整數**索引找資料*

```
int a[5] = {3, 7, 2, 7, 5};
```

索引型態

map v.s. array(c style)

- 在 array 中：

*資料型態*可以自由定義

*索引型態*卻只能是**非負整數**

***用非負整數**索引找資料*

```
int a[5] = {3, 7, 2, 7, 5};
```

資料型態

索引型態

map 遍歷

```
map<char,int> mymap;
mymap['b'] = 100, mymap['a'] = 200, mymap['c'] = 300;

for (auto it = mymap.begin(); it != mymap.end(); it++)
  cout << it->first << ", " << it->second << endl;
```

輸出：

```
a, 200
b, 100
c, 300
```

CF 1133C Balanced Team

CF 1133C Balanced Team

與其記錄某 skill 附近有多少 skill 相差為 5
不如記錄有多少 skill 與某 skill 相差為 5

CF 1133C Balanced Team

與其記錄某 skill 附近有多少 skill 相差為 5
不如記錄有多少 skill 與某 skill 相差為 5

```
for(int i = 0; i < n; i++) {
   scanf(“%d”, &a[i]);
   for(int k = 0; k <= 5; k++) cnt[a[i]+k]++;
}
```


Made by 培訓團隊群 CCNS.

CF 1133C Balanced Team

因此得知 x 附近有 `cnt[x]` 這麼多 skill 與他相差為 5

接著找出哪個區間記錄值最大

```
int best = 1;
for(int i = 0; i < n; i++)
  best = max(best, cnt[a[i]]);
```

CF 1255C League of Leesins

CF 1255C League of Leesins

題目中的例子 (1,4,2),(4,2,3),(2,3,5)
可以很自然地推得數列 1,4,2,3,5，(畢竟就是這樣來的

欸可以先暫停自己想

CF 1255C League of Leesins

題目中的例子 (1,4,2),(4,2,3),(2,3,5)
可以很自然地推得數列 1,4,2,3,5

如果拿到數字 3，只知道可能後面接 4, 2 或是 2, 5
但如果是 1, 5 可以分別知道後面是 4, 2 和 2, 3

CF 1255C League of Leesins

如果拿到數字 3，只知道可能後面接 4, 2 或是 2, 5
但如果是 1, 5 可以分別知道後面是 4, 2 和 2, 3

```
scanf("%d%d%d", &x, &y, &z);
tp[{x, y}].push_back(z);
tp[{y, x}].push_back(z);
tp[{y, z}].push_back(x);
tp[{z, y}].push_back(x);
tp[{z, x}].push_back(y);
tp[{x, z}].push_back(y);
```

CF 1255C League of Leesins

那麼若是有確定兩個數字 p1, p2，
就能利用 tp 推出下一個數字

CF 1255C League of Leesins

```
bool vis[maxn] = {};
for(int i = 0;; i++) {
   printf("%d ", p1);

   vis[p1] = true;
   if(i == n-1) break;

   auto v = tp[{p1, p2}];
   p1 = p2;
   p2 = vis[v[0]]? v[1] : v[0];
}
```

CF 1255C League of Leesins

完整程式碼詳見 第二週教材

priority_queue

- 優先隊列 (priority queue) 是隊列 (queue) 的一個變種
- 每次以**優先度**作為查找和移除的依據

priority_queue

- 優先隊列 (priority queue) 是隊列 (queue) 的一個變種
- 每次以**優先度**作為查找和移除的依據

- dequeue 會先選容器中**優先度最大**的元素
- front 也先選容器中**優先度最大**的元素

priority_queue

- 優先隊列 (priority queue) 是隊列 (queue) 的一個變種
- 每次以**優先度**作為查找和移除的依據

- dequeue 會先選容器中**優先度最大**的元素
- front 也先選容器中**優先度最大**的元素

- 時間複雜度為每次進出 O(log N)

priority_queue

```
priority_queue<T> pq
```
pq 為空的優先隊列，且各元素型別為 T

```
top()
```
優先度最大的元素

```
push(a)
```
將 a 加進優先隊列

```
pop()
```
將優先度最大的一個元素移除

priority_queue

```
priority_queue<int> mypq;
mypq.push(30);
mypq.push(100);
mypq.push(25);
mypq.push(40);

while (!mypq.empty()) {
   int now = mypq.top();
   mypq.pop();
   cout << ' ' << now;
}
```

Output:

100 40 30 25

priority_queue

與 set 以及 map 一樣，裡面的元素(型態)們必須有序

```
struct XXX {
  int code, weight;
  bool operator<(const XXX &lhs) const {
    return weight < lhs.weight;
  }
};
```

GCJ Kickstart Round E 2018 B Milk Tea

好題，自己讀教材

sort

顧名思義

就是對一些東西排序

sort

對陣列排序

```
int a[100];
for(int i = 0; i < 100; i++)
  scanf("%d", &a[i]);

sort(a, a+100);
```

sort

如果是自訂的結構

```
struct T { int val, num; };
vector<T> v;
```

sort

如果是自訂的結構

```
struct T { int val, num; };
vector<T> v;
```

就得自己訂比較函數

```
bool cmp(const T &a, const T &b) {
  return a.num < b.num;
}
```

sort

將 cmp 加到第三個參數

```
sort(v.begin(), v.end(), cmp);
```

當然也可以直接把匿名函數寫進去：

```
sort(v.begin(), v.end(), [](T a, T b) {
  return a.num < b.num;
});
```

Questions?