資料結構

temmie

1. Stack
2. Queue
3. Deque
4. Priority_queue
5. Pair
6. Set
7. Multiset
8. Map
9. Bitset
10. 雑項
11. 題目

Stack

Stack

- 一種後進先出 (Last-In-First-Out, LIFO) 的線性資料結構
- 和疊盤子一樣，要放只能疊上去，要拿只能從最上面拿

Stack

使用時機：

- 消除元素（括號序列）
- 單調堆疊（stack 裡的東西保持單調性）
- 四則運算解析
- 模擬遞迴

Stack

現成的工具：

- stack<type> name
  - .push(val)
  - .pop()
  - .top()
  - .size()
  - .empty()

Queue

Queue

- 一種先進先出 (First-In-First-Out, FIFO) 的線性資料結構
- 和排隊一樣，無法插隊，先到先走

Queue

使用時機：

- BFS
- 只需要動到頭尾

Queue

現成的工具：

- queue<type> name
  - .push(val)
  - .pop()
  - .front()
  - .back()
  - .size()
  - .empty()

Deque

Deque

- 簡單來說就是 Stack + Queue
- **常數非常大**，如非必要就不要使用
- 可以自己用陣列實作

Deque

現成的工具：

- deque<type> name
  - .push_front(val)
  - .push_back(val)
  - .pop_front()
  - .pop_back()
  - .front()
  - .back()
  - .size()
  - .empty()
  - .clear()
  - name[i]

Priority_queue

Priority_queue

- 一個可以自動排序的 Queue，但實際上是樹狀結構
- 會讓 Queue 由大到小（或是相反）排序
- 比起我們待會會講的 Set / Multiset 還要快

Priority_queue

使用時機：

- 解決貪心問題

Priority_queue

現成的工具：

- priority_queue<type>: 由大到小
- priority_queue<type, vector<type>, greater<type> >: 由小到大
- .push(val)
- .pop()
- .top()
- .size()
- .empty()

Pair

Pair

- 和我們之前學的 Struct 一樣
- 只有兩個空間，前面叫做 first 後面叫做 second
- 如果不想用 Struct 就可以用 Pair

Priority_queue

現成的工具：

- pair<type1, type2>，兩個元素
- pair<type1, pair<type2, type3> >，三個元素
- .first
- .second

Set

Set

- 一種平衡二元樹，可以先忽略實作方法
- 使用時機非常多，不單單只是做為「集合」使用

Set

使用時機：

- 去重
- 動態排序（動態做二分搜能做的事）
- 快速知道前/後一個元素為何

Set

現成的工具：

- set<type> name
- .begin()
- .end()
- .find(val)，return iterator
- .size()
- .empty()

Set

現成的工具：

- .insert(val)，return pair<iterator, bool>
  - $O(\log n)$
  - iterator 為插入的位置
  - bool 為是否插入成功（如果已經存在就回傳 False）

Set

現成的工具：

- .erase(iterator)，return iterator

  .erase(iterator_left, iterator_right)，return iterator

  .erase(val)，return bool
  - $O(\log n)$
  - 傳入 iterator，則回傳下個元素的 iterator
  - 傳入數值，則回傳有是否被刪除

Set

現成的工具：

- upper_bound / lower_bound(val)，return iterator
  - $O(\log n)$
  - iterator 為找到的位置

Unordered_set

- 和 set 差不多，但是不會有排序功能
- insert 和 erase 變成 $O(1)$
- 通常會用在一個叫做**雜湊**的技巧

Multiset

Multiset

- 和 set 差不多，但沒有去重的功能

Multiset

現成的工具：

- .insert(val)，return iterator
  - $O(\log n)$
  - iterator 為插入的位置

Multiset

現成的工具：

- .erase(iterator)，return iterator

  .erase(iterator_left, iterator_right)，return iterator

  .erase(val)，return bool
  - $O(\log n)$
  - 傳入 iterator，則回傳下個元素的 iterator
  - 傳入數值，則**所有和該數值相同的都會被刪除**
  - 如果只想要刪除一個則使用.erase(.find(val))

Multiset

現成的工具：

- .count(val)，return int
  - $O($ 元素個數 $)$
  - 回傳元素個數，時間複雜度**與元素個數成正比**，因此不該使用 Multiset 計算元素數量

Map

Map

- Map 有「映照」的意思，也就是將 key 對照成 value
- 同樣是以樹狀結構實作
- 同樣有自動排序

Map

使用時機：

- 沒辦法用陣列的索引值儲存
- 將資料做對應

Map

注意事項：

- Map 的 iterator 都是指向一個 pair，前者是 Key，後者是 Value
- 不要資料結構上癮，**如果索引值較小就應該用陣列**
- 在尋找元素之前，應該先確保該元素存在（用.find 檢查），否則會因為該位置被初始化而導致時間複雜度爛掉

Map

現成的工具：

- map<type1, type2> name
- .begin()
- .end()
- .find(val) return iterator
- .size()
- .empty()
- name [key]

Bitset

Bitset

- 比起 bool[]，是以精確的 bit 來實作，空間少上許多
- 做位元運算超快，甚至可以快到 $\frac{1}{64}$ 倍

Bitset

使用時機：

- 需要儲存 01 的資料
- 大量用到位元運算
- 壓常數

Bitset

現成的工具：

- bitset<固定大小> name
- .count()
- .size()
- .all()
- .any()
- .set()
- .reset()
- .to_string()
- .to_ulong()
- .to_ullong()
- name [key]

雑項

雜項

以下是我沒講到的，它們出場率很低

- Tuple
- Unordered_map
- List

題目

題目

- STring
- 括號配對
- 最近較小數字
- 儲存站
- LR 排列
- 木棒組合
- 藤原千花與字串
- 木棒切割