資料結構 I

Koying

2022-07-04

目錄

前言

- 資料結構是用來儲存資料的工具

前言

- 資料結構是用來儲存資料的工具
- 陣列就是一種資料結構

前言

- 資料結構是用來儲存資料的工具
- 陣列就是一種資料結構
- 比較複雜的問題有時候需要使用比較特殊的資料結構來快速地解決他 ~~(例如線段樹之類的)~~

STL 與 iterator 介紹

什麼是 STL

- 標準模板庫 (Standard templete library)

什麼是 STL

- 標準模板庫（Standard templete library）
- C++ 內建（std 內）含有大量好用容器、資料結構的函式庫

什麼是 STL

- 標準模板庫（Standard templete library）
- C++ 內建（std 內）含有大量好用容器、資料結構的函式庫
- 什麼是模板呢？簡單來講就是一個可以適應各種型態的工具（因為模板的內容很多，想了解更多的可以搜尋 C++ templete）

什麼是 STL

- 標準模板庫（Standard templete library）
- C++ 內建（std 內）含有大量好用容器、資料結構的函式庫
- 什麼是模板呢？簡單來講就是一個可以適應各種型態的工具（因為模板的內容很多，想了解更多的可以搜尋 C++ templete）
- 許多看似複雜的題目，都可以使用 STL 內的容器解決

什麼是 STL

- 標準模板庫（Standard templete library）
- C++ 內建（std 內）含有大量好用容器、資料結構的函式庫
- 什麼是模板呢？簡單來講就是一個可以適應各種型態的工具（因為模板的內容很多，想了解更多的可以搜尋 C++ templete）
- 許多看似複雜的題目，都可以使用 STL 內的容器解決
- 適用各種型態（int, double, char 等）

什麼是 STL

- 標準模板庫（Standard templete library）
- C++ 內建（std 內）含有大量好用容器、資料結構的函式庫
- 什麼是模板呢？簡單來講就是一個可以適應各種型態的工具（因為模板的內容很多，想了解更多的可以搜尋 C++ templete）
- 許多看似複雜的題目，都可以使用 STL 內的容器解決
- 適用各種型態（int, double, char 等）
- 以下內容若出現 T a 的格式代表這個 a 是 T 的型態

學習 STL 的好用工具

- cpp reference: https://en.cppreference.com/w/
- 建中講義: https://tioj.ck.tp.edu.tw/uploads/attachment/11/40/1.pdf

iterator

- 中文叫做迭代器，是一種類似指標的東西（指標也算是某種迭代器）
- 是一種用來存取 STL 內元素的工具

iterator

- 中文叫做迭代器，是一種類似指標的東西（指標也算是某種迭代器）
- 是一種用來存取 STL 內元素的工具
- 想要取得迭代器所指向的內容，需要在前面加上取值運算子（*）

iterator

- 中文叫做迭代器，是一種類似指標的東西（指標也算是某種迭代器）
- 是一種用來存取 STL 內元素的工具
- 想要取得迭代器所指向的內容，需要在前面加上取值運算子（*）
- 迭代器分為隨機存取迭代器、雙向迭代器、單向迭代器

iterator

- 中文叫做迭代器，是一種類似指標的東西（指標也算是某種迭代器）
- 是一種用來存取 STL 內元素的工具
- 想要取得迭代器所指向的內容，需要在前面加上取值運算子（*）
- 迭代器分為隨機存取迭代器、雙向迭代器、單向迭代器
  - 隨機存取迭代器：有點像陣列使用的 arr[i]，隨機存取迭代器可以移動到任意位置

iterator

- 中文叫做迭代器，是一種類似指標的東西（指標也算是某種迭代器）
- 是一種用來存取 STL 內元素的工具
- 想要取得迭代器所指向的內容，需要在前面加上取值運算子（*）
- 迭代器分為隨機存取迭代器、雙向迭代器、單向迭代器
  - 隨機存取迭代器：有點像陣列使用的 arr[i]，隨機存取迭代器可以移動到任意位置
  - 雙向迭代器：一次只能夠往前、往後一個位置（-/++）
  - 單向迭代器：一次只能往後一個位置（++）

iterator

- 中文叫做迭代器，是一種類似指標的東西（指標也算是某種迭代器）
- 是一種用來存取 STL 內元素的工具
- 想要取得迭代器所指向的內容，需要在前面加上取值運算子（*）
- 迭代器分為隨機存取迭代器、雙向迭代器、單向迭代器
  - 隨機存取迭代器：有點像陣列使用的 arr[i]，隨機存取迭代器可以移動到任意位置
  - 雙向迭代器：一次只能夠往前、往後一個位置（-/++）
  - 單向迭代器：一次只能往後一個位置（++）
- STL 有兩種迭代器：::iterator 以及::reverse_iterator

iterator

- 中文叫做迭代器，是一種類似指標的東西（指標也算是某種迭代器）
- 是一種用來存取 STL 內元素的工具
- 想要取得迭代器所指向的內容，需要在前面加上取值運算子（*）
- 迭代器分為隨機存取迭代器、雙向迭代器、單向迭代器
  - 隨機存取迭代器：有點像陣列使用的 arr[i]，隨機存取迭代器可以移動到任意位置
  - 雙向迭代器：一次只能夠往前、往後一個位置（-/++）
  - 單向迭代器：一次只能往後一個位置（++）
- STL 有兩種迭代器：::iterator 以及::reverse_iterator
  - ::iterator 可以使用.begin()、.end() 來存取最前面及最後面
  - ::reverse_iterator 可以使用.rbegin()、.rend() 來存取最後面及最前面

iterator

- 中文叫做迭代器，是一種類似指標的東西（指標也算是某種迭代器）
- 是一種用來存取 STL 內元素的工具
- 想要取得迭代器所指向的內容，需要在前面加上取值運算子（*）
- 迭代器分為隨機存取迭代器、雙向迭代器、單向迭代器
  - 隨機存取迭代器：有點像陣列使用的 arr[i]，隨機存取迭代器可以移動到任意位置
  - 雙向迭代器：一次只能夠往前、往後一個位置（-/++）
  - 單向迭代器：一次只能往後一個位置（++）
- STL 有兩種迭代器：::iterator 以及::reverse_iterator
  - ::iterator 可以使用.begin()、.end() 來存取最前面及最後面
  - ::reverse_iterator 可以使用.rbegin()、.rend() 來存取最後面及最前面
  - STL 的容器是左閉又開 [begin(), end())，因此.end() 以及 rbegin() 是沒有元素存在的，不能夠取值

pair、tuple

平面座標點排序

在一個座標平面上有 n 個座標點，請將所有座標點依照先先排序 x 再排序 y 的方式排序後輸出

平面座標點排序

在一個座標平面上有 n 個座標點，請將所有座標點依照先先排序 x 再排序 y 的方式排序後輸出

- 這種題目你會怎麼做呢？

pair

平面座標點排序

在一個座標平面上有 n 個座標點，請將所有座標點依照先先排序 x 再排序 y 的方式排序後輸出

- 這種題目你會怎麼做呢？
- C++ 的 STL 裡有一種好用的資料結構稱為 pair（數對）
- 顧名思義，可以將兩個數字變成一個數對

pair

平面座標點排序

在一個座標平面上有 n 個座標點，請將所有座標點依照先先排序 x 再排序 y 的方式排序後輸出

- 這種題目你會怎麼做呢？
- C++ 的 STL 裡有一種好用的資料結構稱為 pair（數對）
- 顧名思義，可以將兩個數字變成一個數對
- 有點像是內建好一個含有兩個變數的 struct
- 使用需 include <utility>

pair

- pair<T1, T2> p：宣告一個 pair 其第一個位置的型態為 T1，第二個位置的型態為 T2

pair

- `pair<T1, T2> p`：宣告一個 pair 其第一個位置的型態為 T1，第二個位置的型態為 T2
- `pair<T1, T2> p(a, b)` / `pair<T1, T2> p = make_pair(a, b)` / `pair<T1, T2> p = {a, b}` (C++11)：宣告一個 p 並設為 (a, b)

pair

- pair<T1, T2> p：宣告一個 pair 其第一個位置的型態為 T1，第二個位置的型態為 T2
- pair<T1, T2> p(a, b) / pair<T1, T2> p = make_pair(a, b) / pair<T1, T2> p = {a, b} (C++11)：宣告一個 p 並設為 (a, b)
- p = make_pair(T1 a, T2 b) / p = {T1 a, T2 b} (C++11)：將 p 設為 (a, b)

pair

- pair<T1, T2> p：宣告一個 pair 其第一個位置的型態為 T1，第二個位置的型態為 T2
- pair<T1, T2> p(a, b) / pair<T1, T2> p = make_pair(a, b) / pair<T1, T2> p = {a, b} (C++11)：宣告一個 p 並設為 (a, b)
- p = make_pair(T1 a, T2 b) / p = {T1 a, T2 b} (C++11)：將 p 設為 (a, b)
- p.first：取得第一個值，p.second：取得第二個值

pair

- pair<T1, T2> p：宣告一個 pair 其第一個位置的型態為 T1，第二個位置的型態為 T2
- pair<T1, T2> p(a, b) / pair<T1, T2> p = make_pair(a, b) / pair<T1, T2> p = {a, b} (C++11)：宣告一個 p 並設為 (a, b)
- p = make_pair(T1 a, T2 b) / p = {T1 a, T2 b} (C++11)：將 p 設為 (a, b)
- p.first：取得第一個值，p.second：取得第二個值
- pair 是可以比大小的，其規則為：優先比第一項，第一項相同時比第二項，所以可以直接排序

ZJ a915 二維點排序

有 N 個二維平面上的點，Colten 想將他們以 x 座標為第一順位，y 座標為第二順位的方式由小到大排序

- 可以試試看用預設的排序規則，也可以寫寫看自訂 compare 函式
- 參考程式：a915.cpp

平面座標點排序

在一個座標平面上有 n 個座標點，每個點都有權重 v，請將所有座標點依照 x, y, v 的優先順序排序後輸出

平面座標點排序

在一個座標平面上有 n 個座標點，每個點都有權重 v，請將所有座標點依照 x, y, v 的優先順序排序後輸出

- 用一個 `pair<pair<int, int>, int>`？

平面座標點排序

在一個座標平面上有 n 個座標點，每個點都有權重 v，請將所有座標點依照 x, y, v 的優先順序排序後輸出

- 用一個 `pair<pair<int, int>, int>`？
- 也是可以，但有點難看，有沒有一種可以存三個變數的 `pair` 呢？

tuple

平面座標點排序

在一個座標平面上有 n 個座標點，每個點都有權重 v，請將所有座標點依照 x, y, v 的優先順序排序後輸出

- 用一個 `pair<pair<int, int>, int>`？
- 也是可以，但有點難看，有沒有一種可以存三個變數的 `pair` 呢？
- 或許你要的就是 `tuple`（數組）

tuple

平面座標點排序

在一個座標平面上有 n 個座標點，每個點都有權重 v，請將所有座標點依照 x, y, v 的優先順序排序後輸出

- 用一個 pair<pair<int, int>, int>？
- 也是可以，但有點難看，有沒有一種可以存三個變數的 pair 呢？
- 或許你要的就是 tuple（數組）
- tuple 跟 pair 的宣告很像，但 tuple 可以放入不只兩個
- 使用需 include <tuple>

tuple

- tuple<T1, T2, ..., Tn> t：宣告一個有 n 個成員，每個成員的型態分別為 $T_1, T_2, ..., T_n$ 的 tuple t
- 初始化的方法跟 pair 幾乎一樣，不過如果 make_pair 要改成 make_tuple
- 賦值的方法也基本一樣

tuple

- tuple<T1, T2, ..., Tn> t：宣告一個有 n 個成員，每個成員的型態分別為 $T_1, T_2, ..., T_n$ 的 tuple t
- 初始化的方法跟 pair 幾乎一樣，不過如果 make_pair 要改成 make_tuple
- 賦值的方法也基本一樣
- get<0>(t)：取得 t 的第一個成員

tuple

- tuple<T1, T2, ..., Tn> t：宣告一個有 n 個成員，每個成員的型態分別為 $T_1, T_2, ..., T_n$ 的 tuple t
- 初始化的方法跟 pair 幾乎一樣，不過如果 make_pair 要改成 make_tuple
- 賦值的方法也基本一樣
- get<0>(t)：取得 t 的第一個成員
  - 但這看起來有點麻煩

tuple

- `tuple<T1, T2, ..., Tn> t`：宣告一個有 n 個成員，每個成員的型態分別為 $T_1, T_2, ..., T_n$ 的 `tuple t`
- 初始化的方法跟 `pair` 幾乎一樣，不過如果 `make_pair` 要改成 `make_tuple`
- 賦值的方法也基本一樣
- `get<0>(t)`：取得 `t` 的第一個成員
  - 但這看起來有點麻煩
  - 如果我們有一個 `tuple<int, int, int> t`，我們可以使用 `tie(a, b, c) = t` 來讓 `a, b, c` 分別對應到 `t` 的三個成員
  - 在 C++17 以上的版本，我們可以使用 `auto [a, b, c] = t` 來做到一樣的事情

tuple

- `tuple<T1, T2, ..., Tn> t`：宣告一個有 n 個成員，每個成員的型態分別為 $T_1, T_2, ..., T_n$ 的 `tuple t`
- 初始化的方法跟 `pair` 幾乎一樣，不過如果 `make_pair` 要改成 `make_tuple`
- 賦值的方法也基本一樣
- `get<0>(t)`：取得 `t` 的第一個成員
  - 但這看起來有點麻煩
  - 如果我們有一個 `tuple<int, int, int> t`，我們可以使用 `tie(a, b, c) = t` 來讓 `a, b, c` 分別對應到 `t` 的三個成員
  - 在 C++17 以上的版本，我們可以使用 `auto [a, b, c] = t` 來做到一樣的事情
  - `tuple` 同樣也有排序的比照順序：越前面的成員順位越高

vector、string

- 如果你今天需要一個陣列，但你不知道需要開多大，怎麼辦？

vector

- 如果你今天需要一個陣列，但你不知道需要開多大，怎麼辦？
- 這時候 vector 就是一個很好的工具
- vector 又被一些人稱為動態陣列

vector

- 如果你今天需要一個陣列，但你不知道需要開多大，怎麼辦？
- 這時候 vector 就是一個很好的工具
- vector 又被一些人稱為動態陣列
- 他可以藉由 push_back() 的操作，讓這個陣列大小變得很彈性

vector

- 如果你今天需要一個陣列，但你不知道需要開多大，怎麼辦？
- 這時候 vector 就是一個很好的工具
- vector 又被一些人稱為動態陣列
- 他可以藉由 push_back() 的操作，讓這個陣列大小變得很彈性
- 使用需要 include <vector>

vector

- `vector<T> vec`：宣告一個型別為 `T`，長度為 `0` 的 `vector`
- `vector<T> vec(n)`：宣告一個型別為 `T`，長度為 `n` 的 `vector`，$\mathcal{O}(\mathrm{n})$
- `vector<T> vec(n, T val)`：宣告一個型別為 `T`，長度為 `n` 的 `vector`，且初始化為 `val`

vector

- `vector<T> vec`：宣告一個型別為 `T`，長度為 `0` 的 `vector`
- `vector<T> vec(n)`：宣告一個型別為 `T`，長度為 `n` 的 `vector`，$\mathcal{O}(\mathrm{n})$
- `vector<T> vec(n, T val)`：宣告一個型別為 `T`，長度為 `n` 的 `vector`，且初始化為 `val`
- `vector` 的初始化不需要加 $=$：`vector<int> v{1, 2, 3, 4, 5}`

vector

- vector<T> vec：宣告一個型別為 T，長度為 0 的 vector
- vector<T> vec(n)：宣告一個型別為 T，長度為 n 的 vector，$\mathcal{O}(\mathrm{n})$
- vector<T> vec(n, T val)：宣告一個型別為 T，長度為 n 的 vector，且初始化為 val
- vector 的初始化不需要加 $=$：vector<int> v{1, 2, 3, 4, 5}
- 取值就跟一般的陣列一樣，使用 vec[i] 就可以取得第 i 位的值
- 若是存取第一位、最後一位，可以使用 vec.front()、vec.back()

vector

- `vec.push_back(T val)`：可以在 `vec` 後端新增一個元素，均攤 $\mathcal{O}(1)$
- `vec.pop_back()`：將 `vec` 最後端的元素移除，均攤 $\mathcal{O}(1)$

vector

- `vec.push_back(T val)`：可以在 `vec` 後端新增一個元素，均攤 $\mathcal{O}(1)$
- `vec.pop_back()`：將 `vec` 最後端的元素移除，均攤 $\mathcal{O}(1)$
- `vec.size()`、`vec.empty()`：取得 `vec` 的長度、是否為空，均攤 $\mathcal{O}(1)$

vector

- `vec.push_back(T val)`：可以在 `vec` 後端新增一個元素，均攤 $\mathcal{O}(1)$
- `vec.pop_back()`：將 `vec` 最後端的元素移除，均攤 $\mathcal{O}(1)$
- `vec.size()`、`vec.empty()`：取得 `vec` 的長度、是否為空，均攤 $\mathcal{O}(1)$
- `vec.resize(x)`：將 `vec` 的長度強制變為 x，若 $\mathrm{x} >$ 原本的 `size()`，則可以指定一個參數 (`vec.resize(x, val)`)，可往後補足 `x - size()` 個 `val`，$\mathcal{O}(|\ \mathrm{x} - \mathrm{size}\ |)$

vector

- `vec.push_back(T val)`：可以在 `vec` 後端新增一個元素，均攤 $\mathcal{O}(1)$
- `vec.pop_back()`：將 `vec` 最後端的元素移除，均攤 $\mathcal{O}(1)$
- `vec.size()`、`vec.empty()`：取得 `vec` 的長度、是否為空，均攤 $\mathcal{O}(1)$
- `vec.resize(x)`：將 `vec` 的長度強制變為 x，若 $\mathrm{x} >$ 原本的 `size()`，則可以指定一個參數 (`vec.resize(x, val)`)，可往後補足 `x - size()` 個 `val`，$\mathcal{O}(|\ \mathrm{x} - \mathrm{size}\ |)$
- `vec.clear()`：將 `vec` 清除，$\mathcal{O}(\mathrm{size})$

vector

- `vec.push_back(T val)`：可以在 `vec` 後端新增一個元素，均攤 $\mathcal{O}(1)$
- `vec.pop_back()`：將 `vec` 最後端的元素移除，均攤 $\mathcal{O}(1)$
- `vec.size()`、`vec.empty()`：取得 `vec` 的長度、是否為空，均攤 $\mathcal{O}(1)$
- `vec.resize(x)`：將 `vec` 的長度強制變為 x，若 $\mathrm{x} >$ 原本的 `size()`，則可以指定一個參數 (`vec.resize(x, val)`)，可往後補足 `x - size()` 個 `val`，$\mathcal{O}(|\ \mathrm{x} - \mathrm{size}\ |)$
- `vec.clear()`：將 `vec` 清除，$\mathcal{O}(\mathrm{size})$
- `vec.erase(iterator first, iterator last)`：移除 $[\mathrm{first}, \mathrm{last})$ 的元素，並將 $[\mathrm{last}, \mathrm{end})$ 往前補，若不指定 `last` 則只移除 `first`，$\mathcal{O}(\mathrm{end} - \mathrm{first})$

- 因為是線性的結構，vector 的迭代器是隨機存取迭代器

vector

- 因為是線性的結構，vector 的迭代器是隨機存取迭代器
- ::iterator 宣告：vector<T>::iterator
- ::reverse_iterator 宣告：vector<T>::reverse_iterator

vector

- 因為是線性的結構，vector 的迭代器是隨機存取迭代器
- ::iterator 宣告：vector<T>::iterator
- ::reverse_iterator 宣告：vector<T>::reverse_iterator
- 隨機存取迭代器之間支援加減，若想得到某個 iterator it 是第幾個元素可以用 it - vec.begin() 取得

ZJ f819 圖書館

Colten 借了 N 本書，第 i 本書的編號是 S_i，借了 D_i
每本書最多只能借 50 天，逾期一天罰 5 元，求有哪些書逾期以及 Colten 需要罰多少錢，逾期的書請按照編號排序輸出

ZJ f819 圖書館

Colten 借了 N 本書，第 i 本書的編號是 S_i，借了 D_i
每本書最多只能借 50 天，逾期一天罰 5 元，求有哪些書逾期以及 Colten 需要罰多少錢，逾期的書請按照編號排序輸出

- 將逾期的書綁成 pair 丟進 vector 裡

ZJ f819 圖書館

Colten 借了 N 本書，第 i 本書的編號是 S_i，借了 D_i
每本書最多只能借 50 天，逾期一天罰 5 元，求有哪些書逾期以及 Colten 需要罰多少錢，逾期的書請按照編號排序輸出

- 將逾期的書綁成 pair 丟進 vector 裡
- 排序後將 vector 的所有元素輸出

ZJ f819 圖書館

Colten 借了 N 本書，第 i 本書的編號是 S_i，借了 D_i
每本書最多只能借 50 天，逾期一天罰 5 元，求有哪些書逾期以及 Colten 需要罰多少錢，逾期的書請按照編號排序輸出

- 將逾期的書綁成 pair 丟進 vector 裡
- 排序後將 vector 的所有元素輸出
- 參考程式：f819.cpp

vector 例題

ZJ h082 (APCS 202201 P2)

有 n 個人要比賽，第 i 有其戰力指數 S_i 及應變力 T_i，一開始按照編號 $1 \sim n$ 排列，從前端開始兩兩一組比賽，若人數為奇數則最後一個人在這回合不需比賽
每場比賽的勝負規則如下，求最後贏家（假設對戰的兩人為 a, b）：

1. $S_a \times T_a \geq S_b \times T_b$：
   a 獲勝，$S_a = S_a + \frac{S_b \times T_b}{2T_a}$，$T_a = T_a + \frac{S_b \times T_b}{2S_a}$，$S_b = S_b + \frac{S_b}{2}$，$T_b = T_b + \frac{T_b}{2}$
2. $S_a \times T_a < S_b \times T_b$：
   b 獲勝，S_b **變為** $S_b + \frac{S_a \times T_a}{2T_b}$，$T_b$ **變為** $T_b + \frac{S_a \times T_a}{2S_b}$，$S_a = S_a + \frac{S_a}{2}$，$T_a = T_a + \frac{T_a}{2}$

vector 例題

ZJ h082 (APCS 202201 P2)

有 n 個人要比賽，第 i 有其戰力指數 S_i 及應變力 T_i，一開始按照編號 $1 \sim n$ 排列，從前端開始兩兩一組比賽，若人數為奇數則最後一個人在這回合不需比賽
每場比賽的勝負規則如下，求最後贏家（假設對戰的兩人為 a, b）：

1. $S_a \times T_a \geq S_b \times T_b$：
   a 獲勝，$S_a = S_a + \frac{S_b \times T_b}{2T_a}$，$T_a = T_a + \frac{S_b \times T_b}{2S_a}$，$S_b = S_b + \frac{S_b}{2}$，$T_b = T_b + \frac{T_b}{2}$
2. $S_a \times T_a < S_b \times T_b$：
   b 獲勝，S_b 變為 $S_b + \frac{S_a \times T_a}{2T_b}$，$T_b$ 變為 $T_b + \frac{S_a \times T_a}{2S_b}$，$S_a = S_a + \frac{S_a}{2}$，$T_a = T_a + \frac{T_a}{2}$

- 開兩個 vector a, b：第 i 輪的比賽名單、第 $i+1$ 輪的比賽名單

vector 例題

ZJ h082 (APCS 202201 P2)

有 n 個人要比賽，第 i 有其戰力指數 S_i 及應變力 T_i，一開始按照編號 $1 \sim n$ 排列，從前端開始兩兩一組比賽，若人數為奇數則最後一個人在這回合不需比賽
每場比賽的勝負規則如下，求最後贏家（假設對戰的兩人為 a, b）：

1. $S_a \times T_a \geq S_b \times T_b$：
   a 獲勝，$S_a = S_a + \frac{S_b \times T_b}{2T_a}$，$T_a = T_a + \frac{S_b \times T_b}{2S_a}$，$S_b = S_b + \frac{S_b}{2}$，$T_b = T_b + \frac{T_b}{2}$
2. $S_a \times T_a < S_b \times T_b$：
   b 獲勝，S_b 變為 $S_b + \frac{S_a \times T_a}{2T_b}$，$T_b$ 變為 $T_b + \frac{S_a \times T_a}{2S_b}$，$S_a = S_a + \frac{S_a}{2}$，$T_a = T_a + \frac{T_a}{2}$

- 開兩個 vector a, b：第 i 輪的比賽名單、第 $i+1$ 輪的比賽名單
- 在第 i 輪時將可以比第 $i+1$ 輪的選手放入 b 中，第 i 輪結束後再將 a 替換為 b
- 參考程式：h082.cpp

string

- string 很像是一個 vector<char>，但因為很常用到，所以有一些額外的功能
- 需 include <string>

string

- string 很像是一個 vector<char>，但因為很常用到，所以有一些額外的功能
- 需 include <string>
- string 的宣告：string s

string

- string 很像是一個 vector<char>，但因為很常用到，所以有一些額外的功能
- 需 include <string>
- string 的宣告：string s
- string 可以做加減，a += b 可以將 b 接到 a 後面，$\mathcal{O}(\text{size}_a + \text{size}_b)$

string

- string 很像是一個 vector<char>，但因為很常用到，所以有一些額外的功能
- 需 include <string>
- string 的宣告：string s
- string 可以做加減，a += b 可以將 b 接到 a 後面，$\mathcal{O}(\text{size}_a + \text{size}_b)$
- 如果你需要轉成 C 式字串，可以使用 s.c_str()
- 如果要取得字串 s 的子字串，可以使用 s.substr(first, length)，會得到 $[\text{first}, \text{first} + \text{length})$ 的子字串，若是用 s.substr(first) 則會得到 $[\text{first}, \text{end})$ 的子字串

string

- string 很像是一個 vector<char>，但因為很常用到，所以有一些額外的功能
- 需 include <string>
- string 的宣告：string s
- string 可以做加減，a += b 可以將 b 接到 a 後面，$\mathcal{O}(\text{size}_a + \text{size}_b)$
- 如果你需要轉成 C 式字串，可以使用 s.c_str()
- 如果要取得字串 s 的子字串，可以使用 s.substr(first, length)，會得到 $[\text{first}, \text{first} + \text{length})$ 的子字串，若是用 s.substr(first) 則會得到 $[\text{first}, \text{end})$ 的子字串
- 兩個字串可以使用比較運算子比大小，但需要注意複雜度是 $\mathcal{O}(\texttt{max}(\text{size}_a, \text{size}_b))$

string

- string 很像是一個 vector<char>，但因為很常用到，所以有一些額外的功能
- 需 include <string>
- string 的宣告：string s
- string 可以做加減，a += b 可以將 b 接到 a 後面，$\mathcal{O}(\text{size}_a + \text{size}_b)$
- 如果你需要轉成 C 式字串，可以使用 s.c_str()
- 如果要取得字串 s 的子字串，可以使用 s.substr(first, length)，會得到 $[\text{first}, \text{first} + \text{length})$ 的子字串，若是用 s.substr(first) 則會得到 $[\text{first}, \text{end})$ 的子字串
- 兩個字串可以使用比較運算子比大小，但需要注意複雜度是 $\mathcal{O}(\textbf{max}(\text{size}_a, \text{size}_b))$
- 取得長度請使用.size()，.strlen 的複雜度會是可怕的 $\mathcal{O}(\text{size})$

deque

deque

- 覺得 vector 只能 push_back 很難用嗎？deque（雙向佇列）就可以達成 push_front()

deque

- 覺得 vector 只能 push_back 很難用嗎？deque（雙向佇列）就可以達成 push_front()
- deque 可以看成是加強版的 vector，不但能做到 vector 能做的事，還能夠在 front push 以及 pop
- 但有一個缺點是速度比 vector 還要慢很多
- 使用需 include <deque>

deque

- 覺得 vector 只能 push_back 很難用嗎？deque（雙向佇列）就可以達成 push_front()
- deque 可以看成是加強版的 vector，不但能做到 vector 能做的事，還能夠在 front push 以及 pop
- 但有一個缺點是速度比 vector 還要慢很多
- 使用需 include <deque>
- 除了有 push_front() 跟 pop_front() 之外，其他都跟 vector 一樣

deque 例題

ZJ i400 字串解碼 (APCS 202206 P2)

有一個字串加密系統，給你加密的規則以及加密過後的字串，求原始字串
加密規則請看 ZJ

deque 例題

ZJ i400 字串解碼 (APCS 202206 P2)

有一個字串加密系統，給你加密的規則以及加密過後的字串，求原始字串
加密規則請看 ZJ

- 加密過程是將 S 的第一個/最後一個字元接到 T 的後面

deque 例題

ZJ i400 字串解碼 (APCS 202206 P2)

有一個字串加密系統，給你加密的規則以及加密過後的字串，求原始字串
加密規則請看 ZJ

- 加密過程是將 S 的第一個/最後一個字元接到 T 的後面
- 解密過程就是將 T 的最後一個字元接到 S 的第一個/最後一個

deque 例題

ZJ i400 字串解碼 (APCS 202206 P2)

有一個字串加密系統，給你加密的規則以及加密過後的字串，求原始字串
加密規則請看 ZJ

- 加密過程是將 S 的第一個/最後一個字元接到 T 的後面
- 解密過程就是將 T 的最後一個字元接到 S 的第一個/最後一個
- 利用 deque 反著做來解碼即可
- 參考程式：i400.cpp

deque 例題

CF 1352G Special Permutation

構造出一個 $1 \sim n$ 的排列 p 滿足 $2 \leq |p_i - p_{i+1}| \leq 4, (i < n)$

deque 例題

CF 1352G Special Permutation

構造出一個 $1 \sim n$ 的排列 p 滿足 $2 \leq |p_i - p_{i+1}| \leq 4, (i < n)$

- 可以觀察到只要 $n \leq 3$ 必定無解

deque 例題

CF 1352G Special Permutation

構造出一個 $1 \sim n$ 的排列 p 滿足 $2 \leq |p_i - p_{i+1}| \leq 4, (i < n)$

- 可以觀察到只要 $n \leq 3$ 必定無解
- 對於 $n = 4$，我們可以得到一個排列是 $[3, 1, 4, 2]$

deque 例題

CF 1352G Special Permutation

構造出一個 $1 \sim n$ 的排列 p 滿足 $2 \leq |p_i - p_{i+1}| \leq 4, (i < n)$

- 可以觀察到只要 $n \leq 3$ 必定無解
- 對於 $n = 4$，我們可以得到一個排列是 $[3, 1, 4, 2]$
- 當 $n = 5$ 時，可以在 $n = 4$ 的排列的右邊加上 5，就可以構造出一個合法的排列 $3, 1, 4, 2, 5$

deque 例題

CF 1352G Special Permutation

構造出一個 $1 \sim n$ 的排列 p 滿足 $2 \leq |p_i - p_{i+1}| \leq 4, (i < n)$

- 可以觀察到只要 $n \leq 3$ 必定無解
- 對於 $n = 4$，我們可以得到一個排列是 $[3, 1, 4, 2]$
- 當 $n = 5$ 時，可以在 $n = 4$ 的排列的右邊加上 5，就可以構造出一個合法的排列 $3, 1, 4, 2, 5$
- 當 $n = 6$ 時則是在 $n = 5$ 的排列的左邊加上一個 6，變成 $[6, 3, 1, 4, 2, 5]$

deque 例題

CF 1352G Special Permutation

構造出一個 $1 \sim n$ 的排列 p 滿足 $2 \leq |p_i - p_{i+1}| \leq 4, (i < n)$

- 可以觀察到只要 $n \leq 3$ 必定無解
- 對於 $n = 4$，我們可以得到一個排列是 $[3, 1, 4, 2]$
- 當 $n = 5$ 時，可以在 $n = 4$ 的排列的右邊加上 5，就可以構造出一個合法的排列 $3, 1, 4, 2, 5$
- 當 $n = 6$ 時則是在 $n = 5$ 的排列的左邊加上一個 6，變成 $[6, 3, 1, 4, 2, 5]$
- 以此類推，我們可以在 $n = 4$ 的基礎上不斷的在右端以及左端加上元素，就能夠構造出一組解了

deque 例題

CF 1352G Special Permutation

構造出一個 $1 \sim n$ 的排列 p 滿足 $2 \leq |p_i - p_{i+1}| \leq 4, (i < n)$

- 可以觀察到只要 $n \leq 3$ 必定無解
- 對於 $n = 4$，我們可以得到一個排列是 $[3, 1, 4, 2]$
- 當 $n = 5$ 時，可以在 $n = 4$ 的排列的右邊加上 5，就可以構造出一個合法的排列 $3, 1, 4, 2, 5$
- 當 $n = 6$ 時則是在 $n = 5$ 的排列的左邊加上一個 6，變成 $[6, 3, 1, 4, 2, 5]$
- 以此類推，我們可以在 $n = 4$ 的基礎上不斷的在右端以及左端加上元素，就能夠構造出一組解了
- 這個操作可以使用 deque 輕鬆地達成
- 參考程式：1352G.cpp

stack

stack

- stack（堆疊），就像是疊盤子一樣，一層一層往上疊，先拿也是拿最上面的

stack

- stack（堆疊），就像是疊盤子一樣，一層一層往上疊，先拿也是拿最上面的
- 有著 LIFO（Last In First Out）的特性
- 使用需要 include <stack>

stack

- stack<T> st：宣告一個型別為 T 的 stack
- st.size()、st.empty()：同 vector

stack

- stack<T> st：宣告一個型別為 T 的 stack
- st.size()、st.empty()：同 vector
- st.push(T val)：在 stack 上方加入一個元素 val
- st.pop()：移除最上方的元素

stack

- stack<T> st：宣告一個型別為 T 的 stack
- st.size()、st.empty()：同 vector
- st.push(T val)：在 stack 上方加入一個元素 val
- st.pop()：移除最上方的元素
- st.top()：取得最上方的元素

stack 例題

ZJ b304 00673 - Parentheses Balance (括號匹配問題)

給一個由小括號 () 及中括號 [] 組成的字串，問是否合法

stack 例題

ZJ b304 00673 - Parentheses Balance（括號匹配問題）

給一個由小括號（）及中括號 [] 組成的字串，問是否合法

- 以小括號為例，若）的前面是（，那就可以將其消除

stack 例題

ZJ b304 00673 - Parentheses Balance (括號匹配問題)

給一個由小括號 () 及中括號 [] 組成的字串，問是否合法

- 以小括號為例，若) 的前面是 (，那就可以將其消除
- 利用 stack 存等待消除的 (還有 [，當走到) 或是] 時，若 st.top() 是 (或是 [，就將其 pop

stack 例題

ZJ b304 00673 - Parentheses Balance (括號匹配問題)

給一個由小括號 () 及中括號 [] 組成的字串，問是否合法

- 以小括號為例，若) 的前面是 (，那就可以將其消除
- 利用 stack 存等待消除的 (還有 [，當走到) 或是] 時，若 st.top() 是 (或是 [，就將其 pop
- 不合法的條件有三種

stack 例題

ZJ b304 00673 - Parentheses Balance (括號匹配問題)

給一個由小括號 () 及中括號 [] 組成的字串，問是否合法

- 以小括號為例，若) 的前面是 (，那就可以將其消除
- 利用 stack 存等待消除的 (還有 [，當走到) 或是] 時，若 st.top() 是 (或是 [，就將其 pop
- 不合法的條件有三種
  1. 沒有 (或是 [可以消除

stack 例題

ZJ b304 00673 - Parentheses Balance (括號匹配問題)

給一個由小括號 () 及中括號 [] 組成的字串，問是否合法

- 以小括號為例，若) 的前面是 (，那就可以將其消除
- 利用 stack 存等待消除的 (還有 [，當走到) 或是] 時，若 st.top() 是 (或是 [，就將其 pop
- 不合法的條件有三種
  1. 沒有 (或是 [可以消除
  2. 當走到) 或是] 時，st.top() 不是 (或是 [

stack 例題

ZJ b304 00673 - Parentheses Balance (括號匹配問題)

給一個由小括號 () 及中括號 [] 組成的字串，問是否合法

- 以小括號為例，若) 的前面是 (，那就可以將其消除
- 利用 stack 存等待消除的 (還有 [，當走到) 或是] 時，若 st.top() 是 (或是 [，就將其 pop
- 不合法的條件有三種
  1. 沒有 (或是 [可以消除
  2. 當走到) 或是] 時，st.top() 不是 (或是 [
  3. 全部走完之後還有剩下的未匹配括號

stack 例題

ZJ b304 00673 - Parentheses Balance (括號匹配問題)

給一個由小括號 () 及中括號 [] 組成的字串，問是否合法

- 以小括號為例，若) 的前面是 (，那就可以將其消除
- 利用 stack 存等待消除的 (還有 [，當走到) 或是] 時，若 st.top() 是 (或是 [，就將其 pop
- 不合法的條件有三種
  1. 沒有 (或是 [可以消除
  2. 當走到) 或是] 時，st.top() 不是 (或是 [
  3. 全部走完之後還有剩下的未匹配括號
- 參考程式：b304.cpp

stack 例題

ZJ b304 00673 - Parentheses Balance（括號匹配問題）

給一個由小括號（）及中括號［］組成的字串，問是否合法

這題的實作有幾個細節：

1. 輸入有可能有空白行，使用一般的輸入字串會讀不到

stack 例題

ZJ b304 00673 - Parentheses Balance (括號匹配問題)

給一個由小括號 () 及中括號 [] 組成的字串，問是否合法

這題的實作有幾個細節：

1. 輸入有可能有空白行，使用一般的輸入字串會讀不到
2. 需要使用 `getline()`

stack 例題

ZJ b304 00673 - Parentheses Balance (括號匹配問題)

給一個由小括號 () 及中括號 [] 組成的字串，問是否合法

這題的實作有幾個細節：

1. 輸入有可能有空白行，使用一般的輸入字串會讀不到
2. 需要使用 `getline()`
3. 輸入完整數之後的換行有可能會被 `getline` 讀進去，因此輸入完 `n` 之後需要先 `cin.ignore()`

stack 例題

TIOJ 1012 Rails

用文字比較難描述，請看原題

TIOJ 1012 Rails

用文字比較難描述，請看原題

- 維修鐵路有長度限制，所以我們能不用維修鐵路就不用

TIOJ 1012 Rails

用文字比較難描述，請看原題

- 維修鐵路有長度限制，所以我們能不用維修鐵路就不用
- 分兩個步驟，第一步驟先不使用維修鐵路

stack 例題

TIOJ 1012 Rails

用文字比較難描述，請看原題

- 維修鐵路有長度限制，所以我們能不用維修鐵路就不用
- 分兩個步驟，第一步驟先不使用維修鐵路
- 開三個 stack st, tmp, B 分別代表車站、維修鐵路、B 方向

stack 例題

TIOJ 1012 Rails

用文字比較難描述，請看原題

- 維修鐵路有長度限制，所以我們能不用維修鐵路就不用
- 分兩個步驟，第一步驟先不使用維修鐵路
- 開三個 stack st, tmp, B 分別代表車站、維修鐵路、B 方向
- 第一步驟有兩種選項：往車站走、往 B 走

stack 例題

TIOJ 1012 Rails

用文字比較難描述，請看原題

- 維修鐵路有長度限制，所以我們能不用維修鐵路就不用
- 分兩個步驟，第一步驟先不使用維修鐵路
- 開三個 stack st, tmp, B 分別代表車站、維修鐵路、B 方向
- 第一步驟有兩種選項：往車站走、往 B 走
- 能往 B 走就盡量先往 B 走，否則就先放在車站

stack 例題

TIOJ 1012 Rails

用文字比較難描述，請看原題

- 維修鐵路有長度限制，所以我們能不用維修鐵路就不用
- 分兩個步驟，第一步驟先不使用維修鐵路
- 開三個 stack st, tmp, B 分別代表車站、維修鐵路、B 方向
- 第一步驟有兩種選項：往車站走、往 B 走
- 能往 B 走就盡量先往 B 走，否則就先放在車站
- 當有一節車廂可以往 B 走時，有可能可以讓放在車站的車廂也可以往 B 走，所以每次往 B 走時都要額外判斷車站內的車廂是否也可以往 B 走

stack 例題

TIOJ 1012 Rails

用文字比較難描述，請看原題

- 第一步驟有兩種選項：st 跟 B，第二步驟就變成 tmp 跟 B

stack 例題

TIOJ 1012 Rails

用文字比較難描述，請看原題

- 第一步驟有兩種選項：st 跟 B，第二步驟就變成 tmp 跟 B
- 做跟第一步驟一樣的事情，能往 B 走就往 B 走，否則就放在 tmp

stack 例題

TIOJ 1012 Rails

用文字比較難描述，請看原題

- 第一步驟有兩種選項：st 跟 B，第二步驟就變成 tmp 跟 B
- 做跟第一步驟一樣的事情，能往 B 走就往 B 走，否則就放在 tmp
- 有車廂可以往 B 走時都額外判斷維修鐵路內的車廂能不能往 B 走

stack 例題

TIOJ 1012 Rails

用文字比較難描述，請看原題

- 第一步驟有兩種選項：st 跟 B，第二步驟就變成 tmp 跟 B
- 做跟第一步驟一樣的事情，能往 B 走就往 B 走，否則就放在 tmp
- 有車廂可以往 B 走時都額外判斷維修鐵路內的車廂能不能往 B 走
- 參考程式：1012.cpp

queue

queue

- queue（佇列）跟 stack 恰好相反，是類似排隊的運作方式

queue

- queue（佇列）跟 stack 恰好相反，是類似排隊的運作方式
- 有著 FIFO（First In First Out）的特性
- 使用需要 include <queue>

queue

- queue<T> q：宣告一個型別為 T 的 queue
- q.size()、q.empty()：同 vector

queue

- queue<T> q：宣告一個型別為 T 的 queue
- q.size()、q.empty()：同 vector
- q.push(T val)：在 queue 的前端加入一個元素 val
- q.pop()：移除最前方的元素

queue

- queue<T> q：宣告一個型別為 T 的 queue
- q.size()、q.empty()：同 vector
- q.push(T val)：在 queue 的前端加入一個元素 val
- q.pop()：移除最前方的元素
- q.front()：取得最前方的元素
- q.back()：取得最後方的元素

queue 例題

ZJ e447 queue 練習

queue 的基本操作

priority_queue

priority_queue

- 在 queue 內，還包含了一個非常強大的資料結構：priority_queue（優先佇列）

priority_queue

- 在 queue 內，還包含了一個非常強大的資料結構：priority_queue（優先佇列）
- priority_queue 是由一種叫做 binary heap 的結構建立的，可以支援取得最值（可能是最大也可能是最小，看你怎麼設定）

priority_queue

- 在 queue 內，還包含了一個非常強大的資料結構：priority_queue（優先佇列）
- priority_queue 是由一種叫做 binary heap 的結構建立的，可以支援取得最值（可能是最大也可能是最小，看你怎麼設定）
- priority_queue 的宣告方法為：priority_queue<T, Con, Cmp>，其中 Con 代表的是使用的容器，主要可以用 vector 跟 deque，而 Cmp 是排序的規則，以下舉兩種最常見的例子：

priority_queue

- 在 queue 內，還包含了一個非常強大的資料結構：priority_queue（優先佇列）
- priority_queue 是由一種叫做 binary heap 的結構建立的，可以支援取得最值（可能是最大也可能是最小，看你怎麼設定）
- priority_queue 的宣告方法為：priority_queue<T, Con, Cmp>，其中 Con 代表的是使用的容器，主要可以用 vector 跟 deque，而 Cmp 是排序的規則，以下舉兩種最常見的例子：
  - priority_queue<int, vector<int>, less<int» pq / priority_queue<int> pq：將最大值優先取出
  - priority_queue<int, vector<int>, greater<int» pq：將最小值優先取出

priority_queue

- 在 queue 內，還包含了一個非常強大的資料結構：priority_queue（優先佇列）
- priority_queue 是由一種叫做 binary heap 的結構建立的，可以支援取得最值（可能是最大也可能是最小，看你怎麼設定）
- priority_queue 的宣告方法為：priority_queue<T, Con, Cmp>，其中 Con 代表的是使用的容器，主要可以用 vector 跟 deque，而 Cmp 是排序的規則，以下舉兩種最常見的例子：
  - priority_queue<int, vector<int>, less<int» pq / priority_queue<int> pq：將最大值優先取出
  - priority_queue<int, vector<int>, greater<int» pq：將最小值優先取出
- C++ 內建的 less 是指前一個小於後一個，而 pq 的 top 可以視為是最後一個元素，所以當你的 Cmp 是 less 時，就代表是取出最大值

priority_queue

- 在 queue 內，還包含了一個非常強大的資料結構：priority_queue（優先佇列）
- priority_queue 是由一種叫做 binary heap 的結構建立的，可以支援取得最值（可能是最大也可能是最小，看你怎麼設定）
- priority_queue 的宣告方法為：priority_queue<T, Con, Cmp>，其中 Con 代表的是使用的容器，主要可以用 vector 跟 deque，而 Cmp 是排序的規則，以下舉兩種最常見的例子：
  - priority_queue<int, vector<int>, less<int» pq / priority_queue<int> pq：將最大值優先取出
  - priority_queue<int, vector<int>, greater<int» pq：將最小值優先取出
- C++ 內建的 less 是指前一個小於後一個，而 pq 的 top 可以視為是最後一個元素，所以當你的 Cmp 是 less 時，就代表是取出最大值
- 需要注意的是，在宣告一個 pq 時，若在後面加上 iterator 的 first, last (priority_queue<int> pq(first, last))，則 pq 會將 $[\text{first}, \text{last})$ 的元素都放入 pq 內

priority_queue

- pq.size()、pq.empty()、pq.top() 都跟其他 STL 的用法一樣

priority_queue

- pq.size()、pq.empty()、pq.top() 都跟其他 STL 的用法一樣
- pq.push(T val)：將 val 放進 pq 內，$\mathcal{O}(\log n)$
- pq.pop()：將 pq 最頂端的元素移除，$\mathcal{O}(\log n)$

priority_queue

- pq.size()、pq.empty()、pq.top() 都跟其他 STL 的用法一樣
- pq.push(T val)：將 val 放進 pq 內，$\mathcal{O}(\log n)$
- pq.pop()：將 pq 最頂端的元素移除，$\mathcal{O}(\log n)$
- 若是在建構的時候直接指定要放進的內容物的話，複雜度會是 $\mathcal{O}(\text{size})$，一個一個放進去則是 $\mathcal{O}(\text{size} \log \text{size})$

priority_queue 例題

TPR #7 pC 中位數

有一個長度為 n 的數列 a，輸出 n 個數字，第 i 個數字代表 $a_1 \sim a_i$ 的中位數

priority_queue 例題

TPR #7 pC 中位數

有一個長度為 n 的數列 a，輸出 n 個數字，第 i 個數字代表 $a_1 \sim a_i$ 的中位數

- 利用兩個 priority_queue left, right，一個優先取出最大，一個優先取出最小，分別代表左半部以及右半部

priority_queue 例題

TPR #7 pC 中位數

有一個長度為 n 的數列 a，輸出 n 個數字，第 i 個數字代表 $a_1 \sim a_i$ 的中位數

- 利用兩個 priority_queue left, right，一個優先取出最大，一個優先取出最小，分別代表左半部以及右半部
- 對於一個新插入的元素 a_i，若 $a_i \leq$ left.top()，那就將他 push 到 left 裡，反之則 push 到 right 裡

priority_queue 例題

TPR #7 pC 中位數

有一個長度為 n 的數列 a，輸出 n 個數字，第 i 個數字代表 $a_1 \sim a_i$ 的中位數

- 利用兩個 priority_queue left, right，一個優先取出最大，一個優先取出最小，分別代表左半部以及右半部
- 對於一個新插入的元素 a_i，若 $a_i \leq$ left.top()，那就將他 push 到 left 裡，反之則 push 到 right 裡
- 如果 $|$left.size() - right.size()$| > 1$

priority_queue 例題

TPR #7 pC 中位數

有一個長度為 n 的數列 a，輸出 n 個數字，第 i 個數字代表 $a_1 \sim a_i$ 的中位數

- 利用兩個 priority_queue left, right，一個優先取出最大，一個優先取出最小，分別代表左半部以及右半部
- 對於一個新插入的元素 a_i，若 $a_i \leq$ left.top()，那就將他 push 到 left 裡，反之則 push 到 right 裡
- 如果 $|$left.size() - right.size()$| > 1$
  - 若 left.size() > right.size()，將 left.top() push 到 right 內，並執行 left.pop()
  - 若 right.size() > left.size() 就做反向的操作

priority_queue 例題

TPR #7 pC 中位數

有一個長度為 n 的數列 a，輸出 n 個數字，第 i 個數字代表 $a_1 \sim a_i$ 的中位數

- 利用兩個 priority_queue left, right，一個優先取出最大，一個優先取出最小，分別代表左半部以及右半部
- 對於一個新插入的元素 a_i，若 $a_i \leq$ left.top()，那就將他 push 到 left 裡，反之則 push 到 right 裡
- 如果 $|$left.size() - right.size()$| > 1$
  - 若 left.size() > right.size()，將 left.top() push 到 right 內，並執行 left.pop()
  - 若 right.size() > left.size() 就做反向的操作
- 時間複雜度：$\mathcal{O}(n \log n)$

priority_queue 例題

CSES 1161 Stick Divisions

有一根長度為 n 的棍子，需要將它分成長度為 $d_1, d_2, ..., d_m$ 的 m 根棍子
每次可以將一根棍子分成兩個，所需的費用是該棍子的長度，求最少費用是多少

priority_queue 例題

CSES 1161 Stick Divisions

有一根長度為 n 的棍子，需要將它分成長度為 $d_1, d_2, ..., d_m$ 的 m 根棍子
每次可以將一根棍子分成兩個，所需的費用是該棍子的長度，求最少費用是多少

- 換個角度想，將把棍子分開想成是將很多棍子合起來

priority_queue 例題

CSES 1161 Stick Divisions

有一根長度為 n 的棍子，需要將它分成長度為 $d_1, d_2, ..., d_m$ 的 m 根棍子
每次可以將一根棍子分成兩個，所需的費用是該棍子的長度，求最少費用是多少

- 換個角度想，將把棍子分開想成是將很多棍子合起來
- 會發現到，每次都取最小的兩根棍子合併會是最佳的策略

priority_queue 例題

CSES 1161 Stick Divisions

有一根長度為 n 的棍子，需要將它分成長度為 $d_1, d_2, ..., d_m$ 的 m 根棍子
每次可以將一根棍子分成兩個，所需的費用是該棍子的長度，求最少費用是多少

- 換個角度想，將把棍子分開想成是將很多棍子合起來
- 會發現到，每次都取最小的兩根棍子合併會是最佳的策略
- 利用取出最小值的 priority_queue 來實作
- 參考程式：Stick_Divisions.cpp

set

set

- set（集合）是一個自平衡二元查找樹，簡單來說就是一個樹狀的資料結構
- 跟數學中的集合很像，不會有重複的元素

set

- set（集合）是一個自平衡二元查找樹，簡單來說就是一個樹狀的資料結構
- 跟數學中的集合很像，不會有重複的元素
- 支援 $\mathcal{O}(\log n)$ 的插入、刪除、搜尋
- 需要 `include <set>`

set

- set（集合）是一個自平衡二元查找樹，簡單來說就是一個樹狀的資料結構
- 跟數學中的集合很像，不會有重複的元素
- 支援 $\mathcal{O}(\log n)$ 的插入、刪除、搜尋
- 需要 `include <set>`
- `set<T> st`：宣告一個型別為 `T` 的 `set`

set

- st.size(), st.empty() 同其他容器

set

- st.size(), st.empty() 同其他容器
- st.insert(T val)：將 val 加入 st 中，$\mathcal{O}(\log \text{size})$

set

- `st.size()`, `st.empty()` 同其他容器
- `st.insert(T val)`：將 `val` 加入 `st` 中，$\mathcal{O}(\log \text{size})$
- `st.find(T val)`：找到 `val` 所在位置的 `iterator`，若找不到則回傳 `st.end()`，$\mathcal{O}(\log \text{size})$
- `st.erase(T val)`：在 `st` 中刪除 `val`，$\mathcal{O}(\log \text{size})$

set

- `st.size()`, `st.empty()` 同其他容器
- `st.insert(T val)`：將 `val` 加入 `st` 中，$\mathcal{O}(\texttt{log}\,\text{size})$
- `st.find(T val)`：找到 `val` 所在位置的 `iterator`，若找不到則回傳 `st.end()`，$\mathcal{O}(\texttt{log}\,\text{size})$
- `st.erase(T val)`：在 `st` 中刪除 `val`，$\mathcal{O}(\texttt{log}\,\text{size})$
- `st.erase(iterator first, iterator last)`：刪除 $[\text{first}, \text{last})$ 之間的元素，$\mathcal{O}(\text{count})$

set

- `st.size()`, `st.empty()` 同其他容器
- `st.insert(T val)`：將 `val` 加入 `st` 中，$\mathcal{O}(\texttt{log}\,\text{size})$
- `st.find(T val)`：找到 `val` 所在位置的 `iterator`，若找不到則回傳 `st.end()`，$\mathcal{O}(\texttt{log}\,\text{size})$
- `st.erase(T val)`：在 `st` 中刪除 `val`，$\mathcal{O}(\texttt{log}\,\text{size})$
- `st.erase(iterator first, iterator last)`：刪除 $[\text{first}, \text{last})$ 之間的元素，$\mathcal{O}(\text{count})$
- `st.count(T val)`：取得 `val` 在 `st` 中的數量，$\mathcal{O}(\texttt{log}\,\text{size} + \text{count})$

set

- `st.size()`, `st.empty()` 同其他容器
- `st.insert(T val)`：將 `val` 加入 `st` 中，$\mathcal{O}(\texttt{log}\,\text{size})$
- `st.find(T val)`：找到 `val` 所在位置的 `iterator`，若找不到則回傳 `st.end()`，$\mathcal{O}(\texttt{log}\,\text{size})$
- `st.erase(T val)`：在 `st` 中刪除 `val`，$\mathcal{O}(\texttt{log}\,\text{size})$
- `st.erase(iterator first, iterator last)`：刪除 $[\text{first}, \text{last})$ 之間的元素，$\mathcal{O}(\text{count})$
- `st.count(T val)`：取得 `val` 在 `st` 中的數量，$\mathcal{O}(\texttt{log}\,\text{size} + \text{count})$
- `st.lower_bound(T val)`：在 `st` 中找到第一個指向元素 $\geq \text{val}$ 的迭代器，$\mathcal{O}(\texttt{log}\,\text{size})$
- `st.upper_bound(T val)`：在 `st` 中找到第一個指向元素 $> \text{val}$ 的迭代器，$\mathcal{O}(\texttt{log}\,\text{size})$

set

- `st.size()`, `st.empty()` 同其他容器
- `st.insert(T val)`：將 `val` 加入 `st` 中，$\mathcal{O}(\texttt{log}\,\text{size})$
- `st.find(T val)`：找到 `val` 所在位置的 `iterator`，若找不到則回傳 `st.end()`，$\mathcal{O}(\texttt{log}\,\text{size})$
- `st.erase(T val)`：在 `st` 中刪除 `val`，$\mathcal{O}(\texttt{log}\,\text{size})$
- `st.erase(iterator first, iterator last)`：刪除 $[\text{first}, \text{last})$ 之間的元素，$\mathcal{O}(\text{count})$
- `st.count(T val)`：取得 `val` 在 `st` 中的數量，$\mathcal{O}(\texttt{log}\,\text{size} + \text{count})$
- `st.lower_bound(T val)`：在 `st` 中找到第一個指向元素 $\geq \text{val}$ 的迭代器，$\mathcal{O}(\texttt{log}\,\text{size})$
- `st.upper_bound(T val)`：在 `st` 中找到第一個指向元素 $> \text{val}$ 的迭代器，$\mathcal{O}(\texttt{log}\,\text{size})$
- `set::iterator` 是雙向迭代器，且會由小到大排序，將迭代器往前或往後移動時的複雜度為 $\mathcal{O}(1)$

set 例題

ZJ d129 00136 - Ugly Numbers

定義一個醜數可以質因數分解為 $2^{x} \times 3^{y} \times 5^{z} (x, y, z \geq 0)$
求第 1500 個醜數

set 例題

ZJ d129 00136 - Ugly Numbers

定義一個醜數可以質因數分解為 $2^x \times 3^y \times 5^z (x, y, z \geq 0)$
求第 1500 個醜數

- 第 1500 個醜數超過一億，枚舉會 TLE

set 例題

ZJ d129 00136 - Ugly Numbers

定義一個醜數可以質因數分解為 $2^x \times 3^y \times 5^z (x, y, z \geq 0)$
求第 1500 個醜數

- 第 1500 個醜數超過一億，枚舉會 TLE
- 一個醜數 i 除了 $2, 3, 5$ 之外，必定是另外一個醜數 j 乘上 $2/3/5$ 得到

set 例題

ZJ d129 00136 - Ugly Numbers

定義一個醜數可以質因數分解為 $2^x \times 3^y \times 5^z (x, y, z \geq 0)$
求第 1500 個醜數

- 第 1500 個醜數超過一億，枚舉會 TLE
- 一個醜數 i 除了 $2, 3, 5$ 之外，必定是另外一個醜數 j 乘上 $2/3/5$ 得到
- 利用 set 會自動排序、去重的特性，我們可以先將 $2, 3, 5$ 放入 set，然後再利用 iterator 由小到大將每個數字乘上 $2, 3, 5$ 的數再放入 set 中

set 例題

ZJ d129 00136 - Ugly Numbers

定義一個醜數可以質因數分解為 $2^x \times 3^y \times 5^z (x, y, z \geq 0)$
求第 1500 個醜數

- 第 1500 個醜數超過一億，枚舉會 TLE
- 一個醜數 i 除了 $2, 3, 5$ 之外，必定是另外一個醜數 j 乘上 $2/3/5$ 得到
- 利用 set 會自動排序、去重的特性，我們可以先將 $2, 3, 5$ 放入 set，然後再利用 iterator 由小到大將每個數字乘上 $2, 3, 5$ 的數再放入 set 中
- 時間複雜度：$\mathcal{O}(n \log n)$

set 例題

ZJ f607 切割費用 (APCS 202101 P3)

有一根棍子位在 $[0, \mathrm{L}]$，Colten 會對這個棍子切 n 次，每次的切割點是 $\mathrm{x_i}$，每次切割的費用為切割的棍子長度

例如：有一根棍子位在 $[0, 10]$，Colten 在 5 的位子切一刀，那麼這次切割的費用就是 10

$(\mathrm{n} \leq 2 \times 10^5, \mathrm{L} \leq 10^7)$

set 例題

ZJ f607 切割費用 (APCS 202101 P3)

有一根棍子位在 $[0, \mathrm{L}]$，Colten 會對這個棍子切 n 次，每次的切割點是 $\mathrm{x_i}$，每次切割的費用為切割的棍子長度
例如：有一根棍子位在 $[0, 10]$，Colten 在 5 的位子切一刀，那麼這次切割的費用就是 10
$(\mathrm{n} \leq 2 \times 10^5, \mathrm{L} \leq 10^7)$

- $\mathrm{L} \leq 10^7$，暴力做顯然不實際

set 例題

ZJ f607 切割費用 (APCS 202101 P3)

有一根棍子位在 $[0, \mathrm{L}]$，Colten 會對這個棍子切 n 次，每次的切割點是 $\mathrm{x_i}$，每次切割的費用為切割的棍子長度
例如：有一根棍子位在 $[0, 10]$，Colten 在 5 的位子切一刀，那麼這次切割的費用就是 10
$(\mathrm{n} \leq 2 \times 10^5, \mathrm{L} \leq 10^7)$

- $\mathrm{L} \leq 10^7$，暴力做顯然不實際
- 我們需要一個能夠支援插入又能夠支援找到左右點的資料結構

set 例題

ZJ f607 切割費用 (APCS 202101 P3)

有一根棍子位在 $[0, \mathrm{L}]$，Colten 會對這個棍子切 n 次，每次的切割點是 $\mathrm{x_i}$，每次切割的費用為切割的棍子長度
例如：有一根棍子位在 $[0, 10]$，Colten 在 5 的位子切一刀，那麼這次切割的費用就是 10
$(\mathrm{n} \leq 2 \times 10^5, \mathrm{L} \leq 10^7)$

- $\mathrm{L} \leq 10^7$，暴力做顯然不實際
- 我們需要一個能夠支援插入又能夠支援找到左右點的資料結構
- 利用 set 紀錄切割點，使用 lower_bound 找到目前切割的棍子右界，至於左界可以使用 upper_bound 的前一個來得到

set 例題

ZJ f607 切割費用 (APCS 202101 P3)

有一根棍子位在 $[0, \mathrm{L}]$，Colten 會對這個棍子切 n 次，每次的切割點是 $\mathrm{x_i}$，每次切割的費用為切割的棍子長度
例如：有一根棍子位在 $[0, 10]$，Colten 在 5 的位子切一刀，那麼這次切割的費用就是 10
$(\mathrm{n} \leq 2 \times 10^5, \mathrm{L} \leq 10^7)$

- $\mathrm{L} \leq 10^7$，暴力做顯然不實際
- 我們需要一個能夠支援插入又能夠支援找到左右點的資料結構
- 利用 set 紀錄切割點，使用 lower_bound 找到目前切割的棍子右界，至於左界可以使用 upper_bound 的前一個來得到
- 需要注意切割點有可能本來就被切過的情況

set 例題

ZJ f607 切割費用 (APCS 202101 P3)

有一根棍子位在 $[0, \mathrm{L}]$，Colten 會對這個棍子切 n 次，每次的切割點是 $\mathrm{x_i}$，每次切割的費用為切割的棍子長度
例如：有一根棍子位在 $[0, 10]$，Colten 在 5 的位子切一刀，那麼這次切割的費用就是 10
$(\mathrm{n} \leq 2 \times 10^5, \mathrm{L} \leq 10^7)$

- $\mathrm{L} \leq 10^7$，暴力做顯然不實際
- 我們需要一個能夠支援插入又能夠支援找到左右點的資料結構
- 利用 set 紀錄切割點，使用 lower_bound 找到目前切割的棍子右界，至於左界可以使用 upper_bound 的前一個來得到
- 需要注意切割點有可能本來就被切過的情況
- 題目中的切割點並沒有按照切割順序排，需要另外做處理
- 參考程式：f607.cpp

multiset

- multiset（多重集）是允許重複元素的 set
- 使用方法跟 set 一模一樣

multiset

- multiset（多重集）是允許重複元素的 set
- 使用方法跟 set 一模一樣
- 如果在 st.erase(val) 中放入的參數是值而不是 iterator，則會刪掉所有 st 中的 val

multiset 例題

ZJ d424 00105 - The Skyline Problem

有多棟從 $\mathrm{L_i}$ 延伸到 $\mathrm{R_i}$，高度為 $\mathrm{H_i}$ 的建築物
求城市的天際線

multiset 例題

ZJ d424 00105 - The Skyline Problem

有多棟從 $\mathrm{L_i}$ 延伸到 $\mathrm{R_i}$，高度為 $\mathrm{H_i}$ 的建築物
求城市的天際線

- 我們可以利用一個 vector<pair<int, int» v，以 first 紀錄座標，second 紀錄高度，若高度為正則代表是左界，反之為右界

multiset 例題

ZJ d424 00105 - The Skyline Problem

有多棟從 L_i 延伸到 R_i，高度為 H_i 的建築物
求城市的天際線

- 我們可以利用一個 vector<pair<int, int» v，以 first 紀錄座標，second 紀錄高度，若高度為正則代表是左界，反之為右界
- 若有一個點不是任何建築物的左右界，則他一定不會是天際線的轉折處，因此只需要考慮 v 中的座標即可

multiset 例題

ZJ d424 00105 - The Skyline Problem

有多棟從 L_i 延伸到 R_i，高度為 H_i 的建築物
求城市的天際線

- 我們可以利用一個 vector<pair<int, int» v，以 first 紀錄座標，second 紀錄高度，若高度為正則代表是左界，反之為右界
- 若有一個點不是任何建築物的左右界，則他一定不會是天際線的轉折處，因此只需要考慮 v 中的座標即可
- 利用 multiset<int, greater<int» 維護 v 中每個座標的最高點，如果高度為正那就 insert，反之就 erase

multiset 例題

ZJ d424 00105 - The Skyline Problem

有多棟從 L_i 延伸到 R_i，高度為 H_i 的建築物
求城市的天際線

- 我們可以利用一個 vector<pair<int, int» v，以 first 紀錄座標，second 紀錄高度，若高度為正則代表是左界，反之為右界
- 若有一個點不是任何建築物的左右界，則他一定不會是天際線的轉折處，因此只需要考慮 v 中的座標即可
- 利用 multiset<int, greater<int» 維護 v 中每個座標的最高點，如果高度為正那就 insert，反之就 erase
- 利用另外一個 vector<pair<int, int» ans 來儲存每個可能座標的最高點

multiset 例題

ZJ d424 00105 - The Skyline Problem

有多棟從 L_i 延伸到 R_i，高度為 H_i 的建築物
求城市的天際線

- 我們可以利用一個 vector<pair<int, int» v，以 first 紀錄座標，second 紀錄高度，若高度為正則代表是左界，反之為右界
- 若有一個點不是任何建築物的左右界，則他一定不會是天際線的轉折處，因此只需要考慮 v 中的座標即可
- 利用 multiset<int, greater<int» 維護 v 中每個座標的最高點，如果高度為正那就 insert，反之就 erase
- 利用另外一個 vector<pair<int, int» ans 來儲存每個可能座標的最高點
- 從頭到尾掃描一次，若某個座標的高度不等於上個座標的高度就將其輸出

multiset 例題

ZJ d424 00105 - The Skyline Problem

有多棟從 L_i 延伸到 R_i，高度為 H_i 的建築物
求城市的天際線

- 我們可以利用一個 vector<pair<int, int» v，以 first 紀錄座標，second 紀錄高度，若高度為正則代表是左界，反之為右界
- 若有一個點不是任何建築物的左右界，則他一定不會是天際線的轉折處，因此只需要考慮 v 中的座標即可
- 利用 multiset<int, greater<int» 維護 v 中每個座標的最高點，如果高度為正那就 insert，反之就 erase
- 利用另外一個 vector<pair<int, int» ans 來儲存每個可能座標的最高點
- 從頭到尾掃描一次，若某個座標的高度不等於上個座標的高度就將其輸出
- 注意可能會有左右界一樣的建築物

multiset 例題

ZJ d424 00105 - The Skyline Problem

有多棟從 L_i 延伸到 R_i，高度為 H_i 的建築物
求城市的天際線

- 我們可以利用一個 vector<pair<int, int» v，以 first 紀錄座標，second 紀錄高度，若高度為正則代表是左界，反之為右界
- 若有一個點不是任何建築物的左右界，則他一定不會是天際線的轉折處，因此只需要考慮 v 中的座標即可
- 利用 multiset<int, greater<int» 維護 v 中每個座標的最高點，如果高度為正那就 insert，反之就 erase
- 利用另外一個 vector<pair<int, int» ans 來儲存每個可能座標的最高點
- 從頭到尾掃描一次，若某個座標的高度不等於上個座標的高度就將其輸出
- 注意可能會有左右界一樣的建築物
- 參考程式：d424.cpp

multiset 例題

CSES 1076 Sliding Median

求每個區間為 k 的中位數

multiset 例題

CSES 1076 Sliding Median

求每個區間為 k 的中位數

- 剛剛提到的中位數由於不需要維護區間大小，也就是不會有過期的問題，因此可以直接用 pq

multiset 例題

CSES 1076 Sliding Median

求每個區間為 k 的中位數

- 剛剛提到的中位數由於不需要維護區間大小，也就是不會有過期的問題，因此可以直接用 pq
- 但現在需要考慮過期的問題，會需要刪除指定數值的元素，所以需要使用能夠支援 find 的 multiset

multiset 例題

CSES 1076 Sliding Median

求每個區間為 k 的中位數

- 剛剛提到的中位數由於不需要維護區間大小，也就是不會有過期的問題，因此可以直接用 pq
- 但現在需要考慮過期的問題，會需要刪除指定數值的元素，所以需要使用能夠支援 find 的 multiset
- 實作方法跟 priority_queue 很像，只是多了一個刪除指定元素的動作

multiset 例題

CSES 1076 Sliding Median

求每個區間為 k 的中位數

- 剛剛提到的中位數由於不需要維護區間大小，也就是不會有過期的問題，因此可以直接用 pq
- 但現在需要考慮過期的問題，會需要刪除指定數值的元素，所以需要使用能夠支援 find 的 multiset
- 實作方法跟 priority_queue 很像，只是多了一個刪除指定元素的動作
- 參考程式：Sliding_Median.cpp

map

map

- map（映射）由 key 以及 value 組成，每個 key 都會對應到一個 value
- 跟 set 一樣是樹狀的結構，支援 $\mathcal{O}(\log n)$ 的插入、刪除、搜尋
- 需 include <map>

map

- map（映射）由 key 以及 value 組成，每個 key 都會對應到一個 value
- 跟 set 一樣是樹狀的結構，支援 $\mathcal{O}(\log n)$ 的插入、刪除、搜尋
- 需 include <map>
- map<T1, T2> mp：宣告一個 key 為 T1 型別，value 為 T2 型別的 map，可以想像成是一個 set<pair<T1, T2»

map

- map（映射）由 key 以及 value 組成，每個 key 都會對應到一個 value
- 跟 set 一樣是樹狀的結構，支援 $\mathcal{O}(\log n)$ 的插入、刪除、搜尋
- 需 include <map>
- map<T1, T2> mp：宣告一個 key 為 T1 型別，value 為 T2 型別的 map，可以想像成是一個 set<pair<T1, T2»
- map 其實是由 pair 組成的結構，所以 iterator 指向的元素為一個 pair，想要取出 key 需要用.first, 取出 value 需要用.second

map

- `mp.size()`、`mp.empty()`、`mp.erase(iterator first, iterator last)`、`mp.erase(T1, key)`、`mp.find(T1 key)`、`mp.count(T1 key)`、`mp.lower_bound(T1, key)`、`mp.upper_bound(T1, key)`：同 set

map

- mp.size()、mp.empty()、mp.erase(iterator first, iterator last)、mp.erase(T1, key)、mp.find(T1 key)、mp.count(T1 key)、mp.lower_bound(T1, key)、mp.upper_bound(T1, key)：同 set
- mp.insert(pair<T1 a, T2 b>)：在 mp 中加入一個元素，a 是 key, b 是 value

map

- `mp.size()`、`mp.empty()`、`mp.erase(iterator first, iterator last)`、`mp.erase(T1, key)`、`mp.find(T1 key)`、`mp.count(T1 key)`、`mp.lower_bound(T1, key)`、`mp.upper_bound(T1, key)`：同 set
- `mp.insert(pair<T1 a, T2 b>)`：在 mp 中加入一個元素，a 是 key, b 是 value
- `mp[T1 key]`：查詢 key 對應到的 value 是什麼，$\mathcal{O}(\log \text{size})$
- `mp[T1 key] = value`：更改 key 對應到的 value，$\mathcal{O}(\log \text{size})$

map

- `mp.size()`、`mp.empty()`、`mp.erase(iterator first, iterator last)`、`mp.erase(T1, key)`、`mp.find(T1 key)`、`mp.count(T1 key)`、`mp.lower_bound(T1, key)`、`mp.upper_bound(T1, key)`：同 set
- `mp.insert(pair<T1 a, T2 b>)`：在 mp 中加入一個元素，a 是 key, b 是 value
- `mp[T1 key]`：查詢 key 對應到的 value 是什麼，$\mathcal{O}(\log \text{size})$
- `mp[T1 key] = value`：更改 key 對應到的 value，$\mathcal{O}(\log \text{size})$
  - 注意若 key 原本不在 mp 裡，則會自動插入一個 (key, 0) 進去

map 例題

TPR #8 PH 名次數列

有一個數列，求每個數字的名次，名次計算方法為：若數列中有 k 個數字比 $\mathrm{a_i}$ 小，那 $\mathrm{a_i}$ 的名次就是 $\mathrm{k}+1$

例如：$10, 10, 20, 20, 30$ 的名次數列為 $1, 1, 2, 2, 3$

map 例題

TPR #8 PH 名次數列

有一個數列，求每個數字的名次，名次計算方法為：若數列中有 k 個數字比 a_i 小，那 a_i 的名次就是 $k+1$
例如：$10, 10, 20, 20, 30$ 的名次數列為 $1, 1, 2, 2, 3$

- 首先我們需要先將他排序，這個用一般的陣列就可以了

map 例題

TPR #8 PH 名次數列

有一個數列，求每個數字的名次，名次計算方法為：若數列中有 k 個數字比 a_i 小，那 a_i 的名次就是 $k+1$
例如：$10, 10, 20, 20, 30$ 的名次數列為 $1, 1, 2, 2, 3$

- 首先我們需要先將他排序，這個用一般的陣列就可以了
- 但是我們還需要去掉重複的，想到去重就會想到 set

map 例題

TPR #8 PH 名次數列

有一個數列，求每個數字的名次，名次計算方法為：若數列中有 k 個數字比 a_i 小，那 a_i 的名次就是 $k+1$
例如：$10, 10, 20, 20, 30$ 的名次數列為 $1, 1, 2, 2, 3$

- 首先我們需要先將他排序，這個用一般的陣列就可以了
- 但是我們還需要去掉重複的，想到去重就會想到 set
- 將元素通通放到 set 中後，從小到大遍歷並用一個變數計算目前是第幾個就可以得到每個數字的名次是多少了

map 例題

TPR #8 PH 名次數列

有一個數列，求每個數字的名次，名次計算方法為：若數列中有 k 個數字比 a_i 小，那 a_i 的名次就是 $k+1$
例如：$10, 10, 20, 20, 30$ 的名次數列為 $1, 1, 2, 2, 3$

- 首先我們需要先將他排序，這個用一般的陣列就可以了
- 但是我們還需要去掉重複的，想到去重就會想到 set
- 將元素通通放到 set 中後，從小到大遍歷並用一個變數計算目前是第幾個就可以得到每個數字的名次是多少了
- 但知道還不夠，我們需要把他記錄起來，每個數字都會對應到一個名次，這其實就是 map 的 key 以及 value

map 例題

TPR #8 PH 名次數列

有一個數列，求每個數字的名次，名次計算方法為：若數列中有 k 個數字比 a_i 小，那 a_i 的名次就是 $k+1$
例如：$10, 10, 20, 20, 30$ 的名次數列為 $1, 1, 2, 2, 3$

- 首先我們需要先將他排序，這個用一般的陣列就可以了
- 但是我們還需要去掉重複的，想到去重就會想到 set
- 將元素通通放到 set 中後，從小到大遍歷並用一個變數計算目前是第幾個就可以得到每個數字的名次是多少了
- 但知道還不夠，我們需要把他記錄起來，每個數字都會對應到一個名次，這其實就是 map 的 key 以及 value
- 參考程式：TPR8H.cpp
- 這個操作就是有名的離散化

unordered_set、unordered_map

- C++ 中有另外一種 set、map 是 unordered 的，利用了 hash 的原理使得操作的複雜度消去掉一個 log 變為 $\mathcal{O}(1)$
- 使用上跟一般的 set、map 一樣，只是在前面加上了 unordered_

unordered_set、unordered_map

- C++ 中有另外一種 set、map 是 unordered 的，利用了 hash 的原理使得操作的複雜度消去掉一個 log 變為 $\mathcal{O}(1)$
- 使用上跟一般的 set、map 一樣，只是在前面加上了 unordered_
- 由於是使用 hash，所以元素都不會排序，因此不能使用 lower_bound、upper_bound
- 迭代器是單向迭代器

unordered_set、unordered_map

- C++ 中有另外一種 set、map 是 unordered 的，利用了 hash 的原理使得操作的複雜度消去掉一個 log 變為 $\mathcal{O}(1)$
- 使用上跟一般的 set、map 一樣，只是在前面加上了 unordered_
- 由於是使用 hash，所以元素都不會排序，因此不能使用 lower_bound、upper_bound
- 迭代器是單向迭代器
- hash 會有碰撞的風險（兩個 key hash 出來的值相同），如果沒有自訂 hash function 的話可能會因為碰撞而導致常數爆表，如果是在 codeforces 等有賽後 hack 的比賽中一定要使用自訂 hash function，不然就會吃到滿滿的紅色 -1

bitset

bitset

- bitset 可以看成是一個優化過的 bool 陣列

bitset

- bitset 可以看成是一個優化過的 bool 陣列
- 一般的 bool 要佔一個 byte, bitset 只需要佔一個 bit
- 運行起來非常快速，對常數優化有很大的幫助
- 還可以對同大小的 bitset 做位元運算

bitset

- bitset 可以看成是一個優化過的 bool 陣列
- 一般的 bool 要佔一個 byte, bitset 只需要佔一個 bit
- 運行起來非常快速，對常數優化有很大的幫助
- 還可以對同大小的 bitset 做位元運算
- 需 include <bitset>

bitset

- bitset<size> b(a)：宣告一個長度為 size 的 bitset，並將其初始化為 a，a 可為整數、字串，若沒有初始值則會預設設為 0

bitset

- bitset<size> b(a)：宣告一個長度為 size 的 bitset，並將其初始化為 a，a 可為整數、字串，若沒有初始值則會預設設為 0
- b[i]：取得第 i 個位元的值

bitset

- `bitset<size> b(a)`：宣告一個長度為 size 的 `bitset`，並將其初始化為 a，`a` 可為整數、字串，若沒有初始值則會預設設為 0
- `b[i]`：取得第 i 個位元的值
- `b.size()`：取得有幾個位元，$\mathcal{O}(1)$
- `b.count()`：回傳有幾個 1，$\mathcal{O}(\mathrm{size})$

bitset

- `bitset<size> b(a)`：宣告一個長度為 size 的 `bitset`，並將其初始化為 a，`a` 可為整數、字串，若沒有初始值則會預設設為 0
- `b[i]`：取得第 i 個位元的值
- `b.size()`：取得有幾個位元，$\mathcal{O}(1)$
- `b.count()`：回傳有幾個 1，$\mathcal{O}(\text{size})$
- `b.set()`：將 `b` 的所有位元初始化為 1，$\mathcal{O}(\text{size})$
- `b.reset()`：將 `b` 的所有位元初始化為 0，$\mathcal{O}(\text{size})$

bitset

- `bitset<size> b(a)`：宣告一個長度為 size 的 `bitset`，並將其初始化為 a，`a` 可為整數、字串，若沒有初始值則會預設設為 0
- `b[i]`：取得第 i 個位元的值
- `b.size()`：取得有幾個位元，$\mathcal{O}(1)$
- `b.count()`：回傳有幾個 1，$\mathcal{O}(\text{size})$
- `b.set()`：將 `b` 的所有位元初始化為 1，$\mathcal{O}(\text{size})$
- `b.reset()`：將 `b` 的所有位元初始化為 0，$\mathcal{O}(\text{size})$
- `b.flip()`：將 `b` 的所有位元反轉，$\mathcal{O}(\text{size})$

bitset

- `bitset<size> b(a)`：宣告一個長度為 size 的 `bitset`，並將其初始化為 a，`a` 可為整數、字串，若沒有初始值則會預設設為 0
- `b[i]`：取得第 i 個位元的值
- `b.size()`：取得有幾個位元，$\mathcal{O}(1)$
- `b.count()`：回傳有幾個 1，$\mathcal{O}(\text{size})$
- `b.set()`：將 `b` 的所有位元初始化為 1，$\mathcal{O}(\text{size})$
- `b.reset()`：將 `b` 的所有位元初始化為 0，$\mathcal{O}(\text{size})$
- `b.flip()`：將 `b` 的所有位元反轉，$\mathcal{O}(\text{size})$
- `b.to_string()`：將 `b` 轉為字串
- `b.to_ulong()`：將 `b` 轉為 `unsigned long`

bitset

- `bitset<size> b(a)`：宣告一個長度為 size 的 `bitset`，並將其初始化為 a，`a` 可為整數、字串，若沒有初始值則會預設設為 0
- `b[i]`：取得第 i 個位元的值
- `b.size()`：取得有幾個位元，$\mathcal{O}(1)$
- `b.count()`：回傳有幾個 1，$\mathcal{O}(\text{size})$
- `b.set()`：將 `b` 的所有位元初始化為 1，$\mathcal{O}(\text{size})$
- `b.reset()`：將 `b` 的所有位元初始化為 0，$\mathcal{O}(\text{size})$
- `b.flip()`：將 `b` 的所有位元反轉，$\mathcal{O}(\text{size})$
- `b.to_string()`：將 `b` 轉為字串
- `b.to_ulong()`：將 `b` 轉為 `unsigned long`
- `b1 (位元運算) b2`：將 `b1`、`b2` 做位元運算，$\mathcal{O}(\text{size})$

ABC 240C Jumping Takahashi

Colten 原本在 0 的位置，接著他會跳 n 次，第 i 次可以選擇往前跳 a_i 或是 b_i 格問最後能不能剛好到達 x $(n \leq 100, x \leq 10000)$

ABC 240C Jumping Takahashi

Colten 原本在 0 的位置，接著他會跳 n 次，第 i 次可以選擇往前跳 a_i 或是 b_i 格問最後能不能剛好到達 x $(n \leq 100, x \leq 10000)$

- 嘗試將能否到達的位置以 0、1 表示

ABC 240C Jumping Takahashi

Colten 原本在 0 的位置，接著他會跳 n 次，第 i 次可以選擇往前跳 a_i 或是 b_i 格問最後能不能剛好到達 x $(n \leq 100, x \leq 10000)$

- 嘗試將能否到達的位置以 0、1 表示
- vis[i][j] 為跳第 i 次是否能夠跳到 j，並將 vis[0][0] 設為 true

ABC 240C Jumping Takahashi

Colten 原本在 0 的位置，接著他會跳 n 次，第 i 次可以選擇往前跳 a_i 或是 b_i 格問最後能不能剛好到達 x $(n \leq 100, x \leq 10000)$

- 嘗試將能否到達的位置以 0、1 表示
- vis[i][j] 為跳第 i 次是否能夠跳到 j，並將 vis[0][0] 設為 true
- bitset 可以做位元運算，當然也包含左移與右移

ABC 240C Jumping Takahashi

Colten 原本在 0 的位置，接著他會跳 n 次，第 i 次可以選擇往前跳 a_i 或是 b_i 格問最後能不能剛好到達 x $(n \leq 100, x \leq 10000)$

- 嘗試將能否到達的位置以 0、1 表示
- vis[i][j] 為跳第 i 次是否能夠跳到 j，並將 vis[0][0] 設為 true
- bitset 可以做位元運算，當然也包含左移與右移
- 所以第 i 次能夠跳到的格子就是第 i - 1 次能夠跳到的格子移動 ai 或是 bi 所得到的格子

ABC 240C Jumping Takahashi

Colten 原本在 0 的位置，接著他會跳 n 次，第 i 次可以選擇往前跳 a_i 或是 b_i 格問最後能不能剛好到達 x $(n \leq 100, x \leq 10000)$

- 嘗試將能否到達的位置以 0、1 表示
- vis[i][j] 為跳第 i 次是否能夠跳到 j，並將 vis[0][0] 設為 true
- bitset 可以做位元運算，當然也包含左移與右移
- 所以第 i 次能夠跳到的格子就是第 i - 1 次能夠跳到的格子移動 ai 或是 bi 所得到的格子
- 所以我們就得到了一個式子：vis[i] = vis[i - 1] « ai |vis[i - 1] « bi (bitset 中 0 在最右邊)
- 參考程式：ABC240C

bitset 例題

ZJ f630 (2020 全國賽 PE 共同朋友)

給定每個人有哪些朋友，求有多少對 (a, b) 滿足 $a < b$ 且 a, b 至少有一位共同好友

bitset 例題

ZJ f630 (2020 全國賽 PE 共同朋友)

給定每個人有哪些朋友，求有多少對 (a, b) 滿足 $a < b$ 且 a, b 至少有一位共同好友

- 將一個人的好友關係寫成 0-1 bit

bitset 例題

ZJ f630 (2020 全國賽 PE 共同朋友)

給定每個人有哪些朋友，求有多少對 (a, b) 滿足 $a < b$ 且 a, b 至少有一位共同好友

- 將一個人的好友關係寫成 0-1 bit
  - 例如總共有 5 人，某人與 $2, 3$ 為好友，則可以寫成 01100

bitset 例題

ZJ f630 (2020 全國賽 PE 共同朋友)

給定每個人有哪些朋友，求有多少對 (a, b) 滿足 $a < b$ 且 a, b 至少有一位共同好友

- 將一個人的好友關係寫成 0-1 bit
  - 例如總共有 5 人，某人與 $2, 3$ 為好友，則可以寫成 01100
- a, b 是否有共同好友就是 a, b 的好友關係經過 AND 運算之後有至少一個 bit 不為 0

bitset 例題

ZJ f630 (2020 全國賽 PE 共同朋友)

給定每個人有哪些朋友，求有多少對 (a, b) 滿足 $a < b$ 且 a, b 至少有一位共同好友

- 將一個人的好友關係寫成 0-1 bit
  - 例如總共有 5 人，某人與 $2, 3$ 為好友，則可以寫成 01100
- a, b 是否有共同好友就是 a, b 的好友關係經過 AND 運算之後有至少一個 bit 不為 0
- bitset 可以直接做位元運算

bitset 例題

ZJ f630（2020 全國賽 PE 共同朋友）

給定每個人有哪些朋友，求有多少對 (a, b) 滿足 $a < b$ 且 a, b 至少有一位共同好友

- 將一個人的好友關係寫成 0-1 bit
  - 例如總共有 5 人，某人與 $2, 3$ 為好友，則可以寫成 01100
- a, b 是否有共同好友就是 a, b 的好友關係經過 AND 運算之後有至少一個 bit 不為 0
- bitset 可以直接做位元運算
- 時間複雜度：$\mathcal{O}(n^3)$，使用 count() 的話會 TLE，需要改用 any()，若有任意 bit 為 1 any() 就會回傳 true
- 參考程式：f630.cpp