資料結構

Data Structure

謝嘉穎

Feb. 10 2026

自我介紹

- Blame
- T26 社長
- 2026 全國賽二等獎

自我介紹

- Blame
- T26 社長
- 2026 全國賽二等獎
- APCS 5 5

自我介紹

- Blame
- T26 社長
- 2026 全國賽二等獎
- APCS 5 5
- TOI 入營考沒打進過

自我介紹

- Blame
- T26 社長
- 2026 全國賽二等獎
- APCS 5 5
- TOI 入營考沒打進過
- 想要學資料結構所以來當資料結構講師

一些不重要的東西

在這堂課你可以做以下事情

- 認真聽課
- 覺得太簡單，跑去刷題目

一些不重要的東西

在這堂課你可以做以下事情

- 認真聽課
- 覺得太簡單，跑去刷題目
- 睡覺 (不要打呼就行)

一些不重要的東西

在這堂課你可以做以下事情

- 認真聽課
- 覺得太簡單，跑去刷題目
- 睡覺 (不要打呼就行)
- 舉手提問

一些不重要的東西

在這堂課你可以做以下事情

- 認真聽課
- 覺得太簡單，跑去刷題目
- 睡覺 (不要打呼就行)
- 舉手提問
- 要上廁所自己去

一些不重要的東西

在這堂課你不能做的事情

- 玩遊戲 ex.(Roblox)

一些不重要的東西

在這堂課你不能做的事情

- 玩遊戲 ex.(Roblox)但 Minecraft 也許可以

一些不重要的東西

在這堂課你不能做的事情

- 玩遊戲 ex.(Roblox)但 Minecraft 也許可以
- 後空翻

一些不重要的東西

在這堂課你不能做的事情

- 玩遊戲 ex.(Roblox)但 Minecraft 也許可以
- 後空翻
- 吵鬧

一些不重要的東西

在這堂課你不能做的事情

- 玩遊戲 ex.(Roblox)但 Minecraft 也許可以
- 後空翻
- 吵鬧
- 講我聽不懂的笑話 ex.(3K 打勾)

Slido

上課提問可以用這個網站 (假如舉手太害羞的話)

前言

今天講的東西是資料結構

希望大家都可以聽懂

前言

今天講的東西是資料結構

希望大家都可以聽懂

課程進度應該會由簡到難

不用強迫自己全部都聽懂

前言

今天講的東西是資料結構

希望大家都可以聽懂

課程進度應該會由簡到難

不用強迫自己全部都聽懂

但希望大家都能從這堂課上學到一點東西

課程規劃

下面應該是等下課程會講到的內容

- DSU
- Segment Tree
- BIT
- 分塊
- Sparse Table
- 珂朵莉樹

假如你上面的都會了

課程規劃

下面應該是等下課程會講到的內容

- DSU
- Segment Tree
- BIT
- 分塊
- Sparse Table
- 珂朵莉樹

假如你上面的都會了那就去刷題

前置知識

要完全聽懂以下課程，我猜你需要先會

- 時間複雜度
- 一點點高中數學
- 遞迴 & 分治概念
- STL

前置知識

要完全聽懂以下課程，我猜你需要先會

- 時間複雜度
- 一點點高中數學
- 遞迴 & 分治概念
- STL

但很巧的是，昨天就教過上面的其中三種東西了!!!

介紹資料結構

介紹資料結構

什麼是資料結構?

什麼是資料結構?

什麼是資料結構?

什麼是資料結構?

在電腦科學中，資料結構（英語：data structure）是電腦中儲存、組織資料的方式。by wiki

什麼是資料結構?

什麼是資料結構?

在電腦科學中，資料結構（英語：data structure）是電腦中儲存、組織資料的方式。by wiki

能將資料結構當成一個工具盒

什麼是資料結構?

什麼是資料結構?

在電腦科學中，資料結構（英語：data structure）是電腦中儲存、組織資料的方式。by wiki

能將資料結構當成一個工具盒

在特殊的情況下找到對應的工具盒是非常重要的

Example

例題

有一個隊伍

Q 筆操作，共有 2 種操作

1. 每次從隊伍後端加入編號為 A_i 的人
2. 每次從隊伍前端拔除一個人，並且需要輸出最前端的人是誰

Example

例題

有一個隊伍

Q 筆操作，共有 2 種操作

1. 每次從隊伍後端加入編號為 A_i 的人
2. 每次從隊伍前端拔除一個人，並且需要輸出最前端的人是誰

假如你有認真上 STL 課的話

Example

例題

有一個隊伍

Q 筆操作，共有 2 種操作

1. 每次從隊伍後端加入編號為 A_i 的人
2. 每次從隊伍前端拔除一個人，並且需要輸出最前端的人是誰

假如你有認真上 STL 課的話

queue!

Example

```
1 queue<int> qq;
2 qq.push(a); // add
3 qq.pop(); // remove
```

Example

```
1 queue<int> qq;
2 qq.push(a); // add
3 qq.pop(); // remove
```

而 push 及 pop 就是我們與這個工具盒的互動

總結

也就是說一個能儲存資料、整理資料、拿取資料的工具包

我們其實就可以簡單的稱為 **資料結構**

總結

也就是說一個能儲存資料、整理資料、拿取資料的工具包

我們其實就可以簡單的稱為 **資料結構**

而資料結構之間有些並無好壞之分

總結

也就是說一個能儲存資料、整理資料、拿取資料的工具包

我們其實就可以簡單的稱為 **資料結構**

而資料結構之間有些並無好壞之分

像是要用到 First In First Out 的題目，你用 queue 解

Last in First Out 的題目，你用 stack 解

總結

也就是說一個能儲存資料、整理資料、拿取資料的工具包

我們其實就可以簡單的稱為 **資料結構**

而資料結構之間有些並無好壞之分

像是要用到 First In First Out 的題目，你用 queue 解

Last in First Out 的題目，你用 stack 解

針對這些性質上的差異，我們選取一個 ” **適當**” 的資料結構

STL vs 這堂課的東西

蛤？那這堂資料結構課就是另一堂 STL 囉？

STL vs 這堂課的東西

蛤? 那這堂資料結構課就是另一堂 STL 囉?

簡單來講，我們這堂課所介紹的資料結構

在 C++ 的函式庫裡並沒有

STL vs 這堂課的東西

蛤? 那這堂資料結構課就是另一堂 STL 囉?

簡單來講，我們這堂課所介紹的資料結構

在 C++ 的函式庫裡並沒有

所以我們需要自己將那些工具盒寫出來

DSU

DSU

- 連通性
- 例題一
- 實作
- 例題二
- 優化

圖論的小東西

很遺憾的是圖論是下午才教的

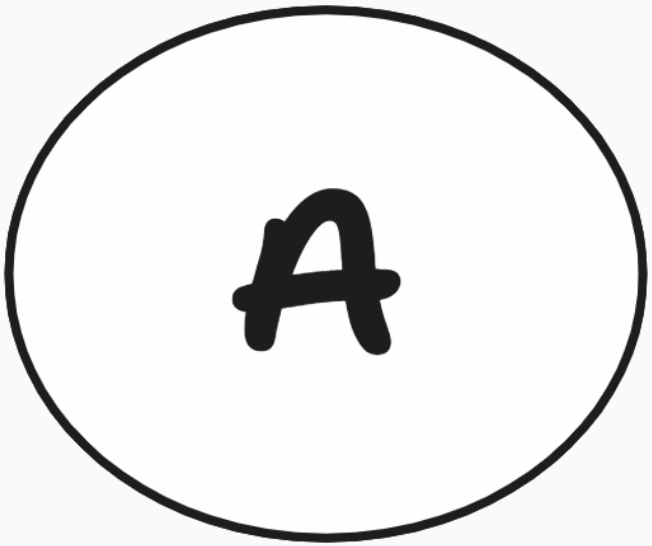
圖論的小東西

很遺憾的是圖論是下午才教的

~~所以我們要先偷點圖論的東西過來教~~

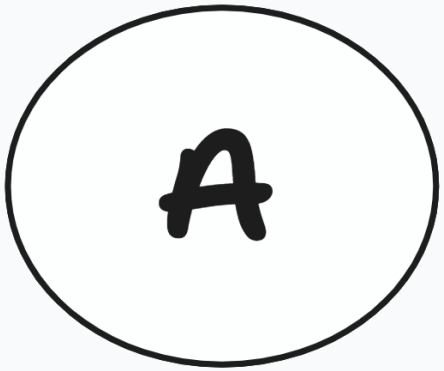
圖論的小東西

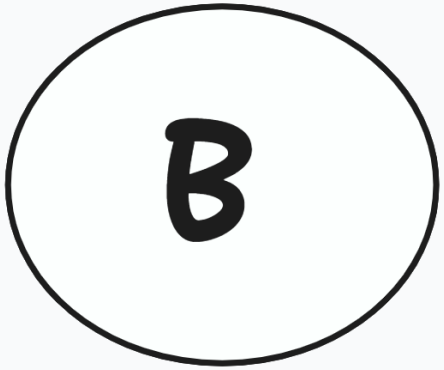
這是一個點 A



圖論的小東西

這是兩個點 A, B



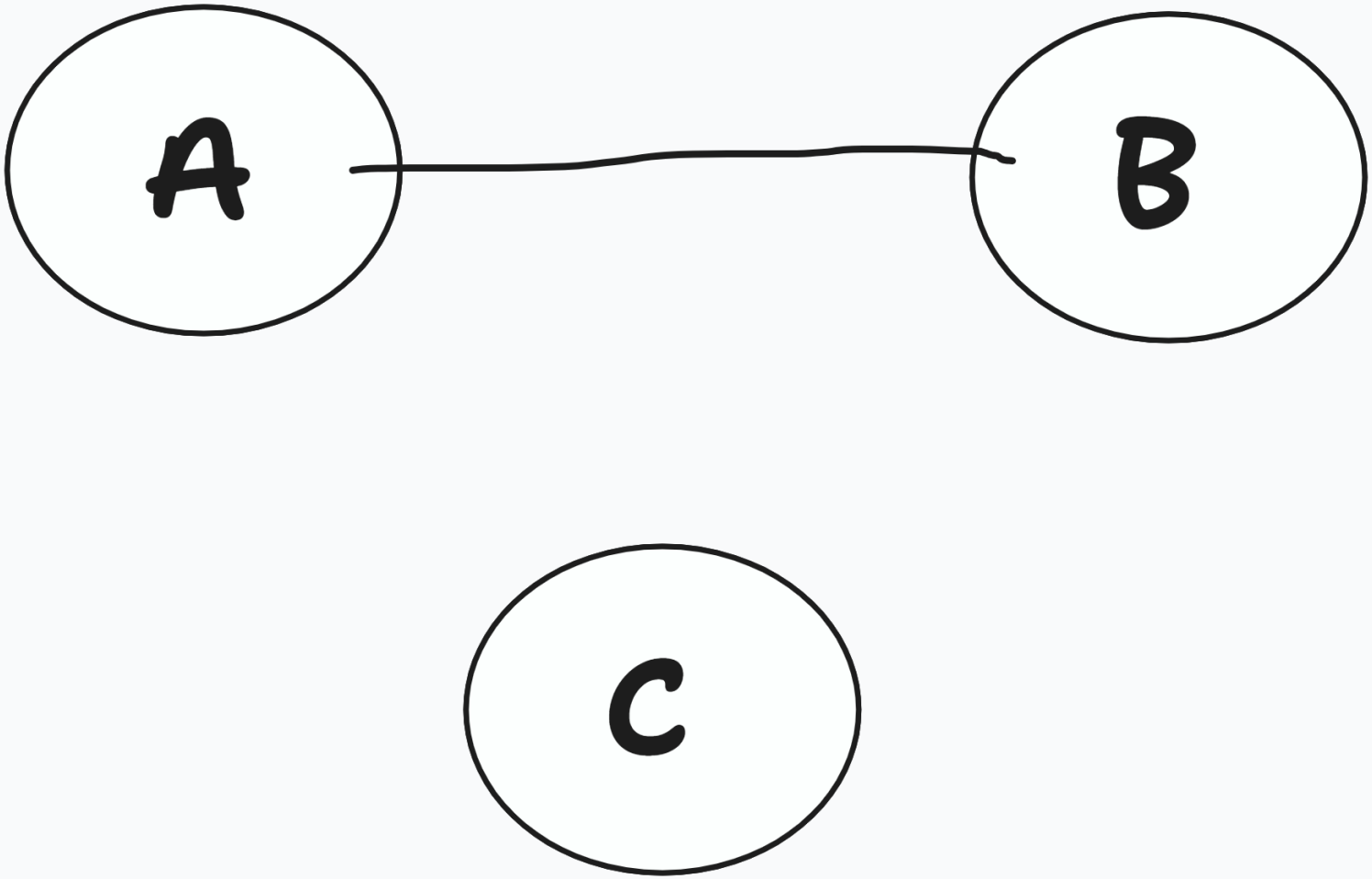


圖論的小東西

A B 之間的線叫做邊

圖論的小東西

多了一個點 C



圖論的小東西

我們稱 A 與 B 為連通

AC or BC 不連通

圖論的小東西

我們稱 A 與 B 為連通

AC or BC 不連通

所以可以稱 AB 為一個連通塊

例題一

例題 (DSU 經典題)

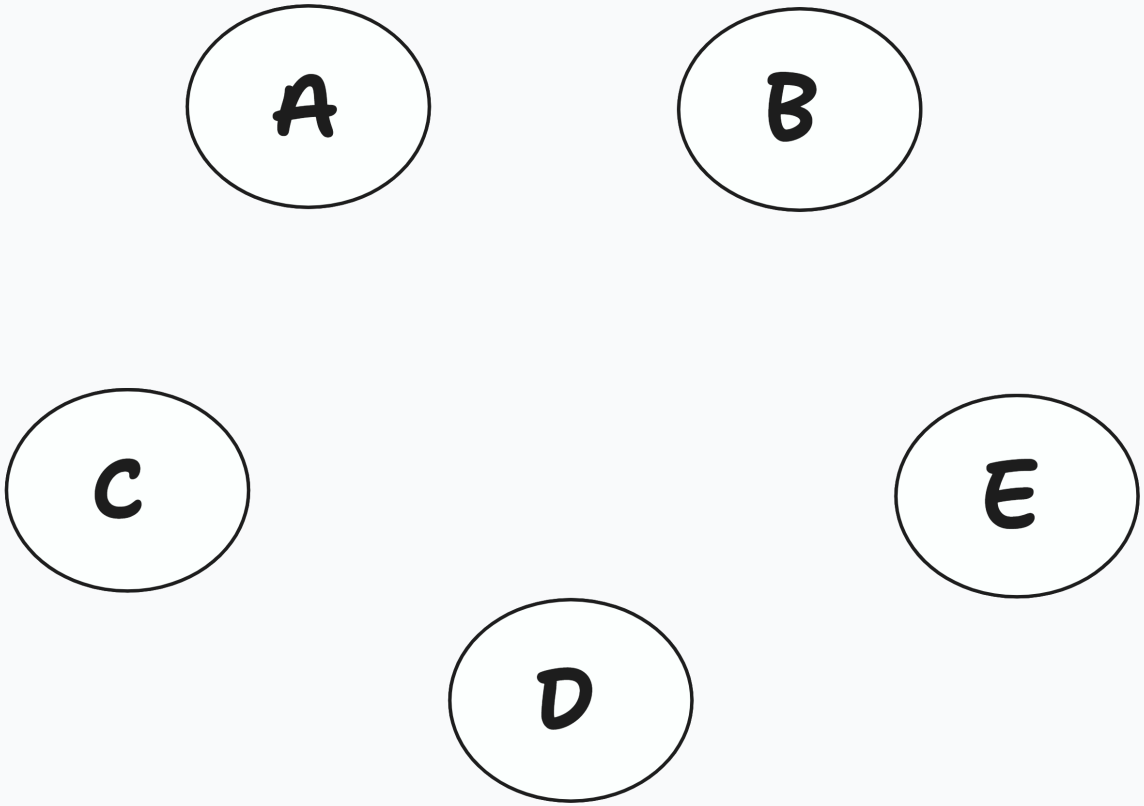
有 N 個點 (編號 $1 \sim N$)，Q 筆詢問

每次有以下兩種操作

1. 在點 A 與點 B 之間連上一條邊
2. 詢問 A 與 B 在當筆詢問的狀況下是否連通

$1 \leq N, Q \leq 1000$

舉個例子來說



定義

我們不妨將問題轉化一下

定義

我們不妨將問題轉化一下

假設 N 個點是 N 個人

且每個人都有自己的頂頭上司

定義

我們不妨將問題轉化一下

假設 N 個點是 N 個人

且每個人都有自己的頂頭上司

那麼將 A B 兩點連接在一起的操作

其實就是將一個人加到另一個的上司中

定義

我們不妨將問題轉化一下

假設 N 個點是 N 個人

且每個人都有自己的頂頭上司

那麼將 A B 兩點連接在一起的操作

其實就是將一個人加到另一個的上司中

這樣詢問就變成 A B 是否在同一個公司內了!

定義

蛤這樣還是好難喔!

定義

蛤這樣還是好難喔!

我們再轉化一下題目

一開始每 N 個人都各擁有一家公司

定義

蛤這樣還是好難喔!

我們再轉化一下題目

一開始每 N 個人都各擁有一家公司

每次連接在一起的時候就等於其中一家公司收購了另外一家公司

定義

蛤這樣還是好難喔!

我們再轉化一下題目

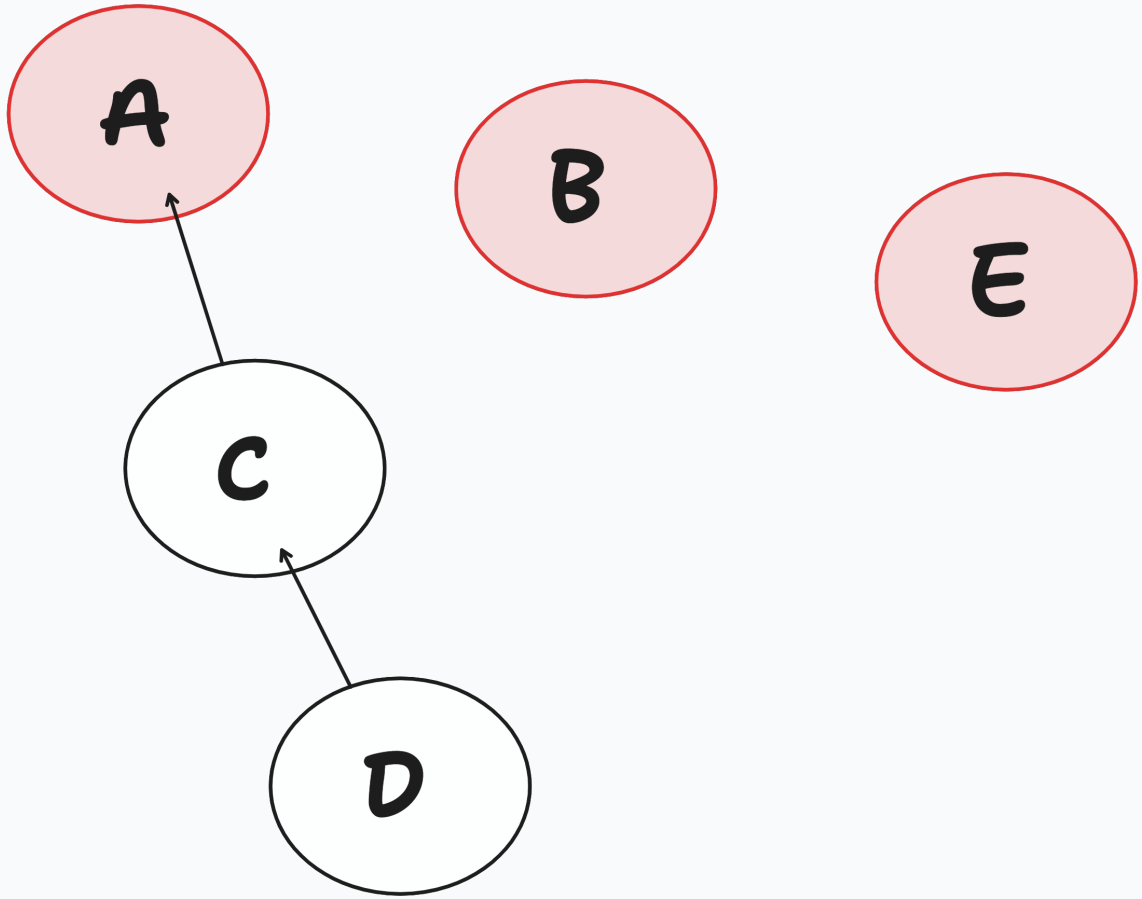
一開始每 N 個人都各擁有一家公司

每次連接在一起的時候就等於其中一家公司收購了另外一家公司

而我們定義每個人僅有至多 1 個頂頭上司

定義

我們將圖畫出來看看



定義

操作一做的事情就是將兩家公司的 CEO 抓出來

將一個人的頂頭上司設成另一個 CEO

定義

操作一做的事情就是將兩家公司的 CEO 抓出來

將一個人的頂頭上司設成另一個 CEO

操作二則是問兩個人隨著頂頭上司的關係一直走

看最終達到的 CEO 是否相同

實作

在實作上，我們為了方便，可以將每個 CEO 的頂頭上司指向自己

實作

在實作上，我們為了方便，可以將每個 CEO 的頂頭上司指向自己

```
1 vector<int> boss(N+1); // N 個人 的頂頭上司
2 for(int i = 1; i <= N; i++) boss[i] = i; // 一開始
     大家皆為自己的 boss
```

實作

我們可以採用遞迴的方式找 CEO

```
1 int find_CEO(int a) { // 找 a 的 CEO
2   if(boss[a] == a) return a; // a 是 CEO 了
3   return find_CEO(boss[a]); // boss[a] 的 CEO 就是
        自己的 CEO
4 }
```

實作

詢問的話就分別呼叫兩個人的 find_CEO 就好了

```
if(find_CEO(a) != find_CEO(b)) {
  // 沒連通
} else {
  // 連通了
}
```

實作

剩下只剩下連接了!

```
int a_CEO = find_CEO(a);
int b_CEO = find_CEO(b);
if(a_CEO == b_CEO) {
  // 兩的點早就連通了
  // 所以不用操作
} else {
  boss[a_CEO] = b_CEO;
}
```

複雜度分析

我們來分析一下上面的複雜度

複雜度分析

我們來分析一下上面的複雜度

我們發現上面的 code 的瓶頸都是在 `find_CEO` 函式

那麼 `find_CEO` 複雜度為多少呢?

複雜度分析

`find_CEO` 實際上在做的事情是跳鏈

所以複雜度應該是 O(鏈長)

複雜度分析

`find_CEO` 實際上在做的事情是跳鏈

所以複雜度應該是 O(鏈長)

而很明顯的，假如我先做 $N - 1$ 一次操作一

這樣我的圖就會是一條很長很長的鏈

複雜度分析

`find_CEO` 實際上在做的事情是跳鏈

所以複雜度應該是 O(鏈長)

而很明顯的，假如我先做 $N-1$ 一次操作一

這樣我的圖就會是一條很長很長的鏈

所以複雜度為 $O(N)$

複雜度分析

我們做 Q 次 `find_CEO`

所以總共複雜度為 $O(Q \times N) = O(NQ)$

小結

上述轉化問題為每個節點為單向出邊以及每個連通塊中使用最高頂點來操作的方式

小結

上述轉化問題為每個節點為單向出邊以及每個連通塊中使用最高頂點來操作的方式

我們稱之為 DSU(Disjoint Set Union)

中文叫做並查集

例題二

例題 (TOJ 701 電學大師)

一張 N 個點的圖

Caido 與 Same 在圖上玩

Q 次指令，有兩種

1. Caido 站在點 x，Same 站在點 y，問兩點是否有直接連接或者間接連接
2. 將點 x 與點 y 連接起來

$1 \leq N \leq 10^5$ $1 \leq Q \leq 2 \cdot 10^5$

例題二

假如你剛剛有在認真聽課

例題二

假如你剛剛有在認真聽課

你會發現這根本例題一阿!

例題二

假如你剛剛有在認真聽課

你會發現這根本例題一阿!

太天真了...

- $1 \leq N \leq 10^5$
- $1 \leq Q \leq 2 \cdot 10^5$

例題二

假如你剛剛有在認真聽課

你會發現這根本例題一阿!

太天真了...

- $1 \leq N \leq 10^5$
- $1 \leq Q \leq 2 \cdot 10^5$

$O(NQ)$ 在這題過不了了!

路徑壓縮

既然我們剛剛的瓶頸是在 `find_CEO`

那我們就來想想看怎麼優化他!

路徑壓縮

發現一個性質:

我們好像只在乎員工以及 CEO 的關係?

路徑壓縮

發現一個性質:

我們好像只在乎員工以及 CEO 的關係?

性質 (DSU 性質)

對於每個員工 $\rightarrow$ 自己的 CEO

我們根本不用知道路徑上走過誰!

路徑壓縮

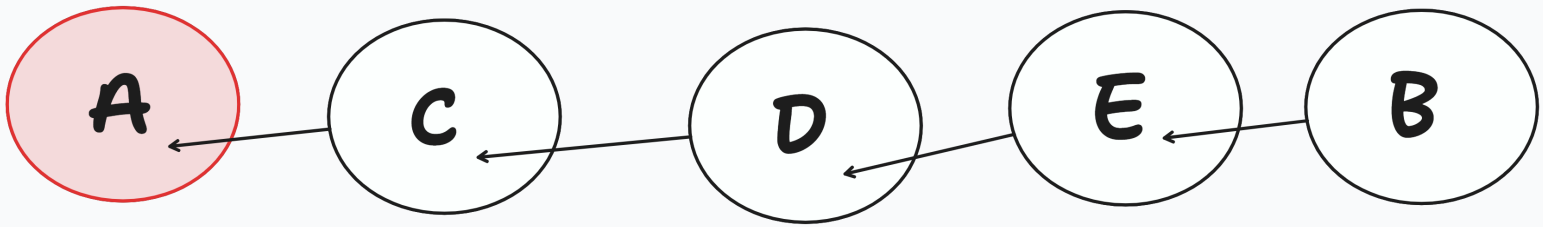
推論

我們可以在每次詢問將路徑鏈上的節點們拆掉

使得他們各自的頂頭上司皆為 CEO

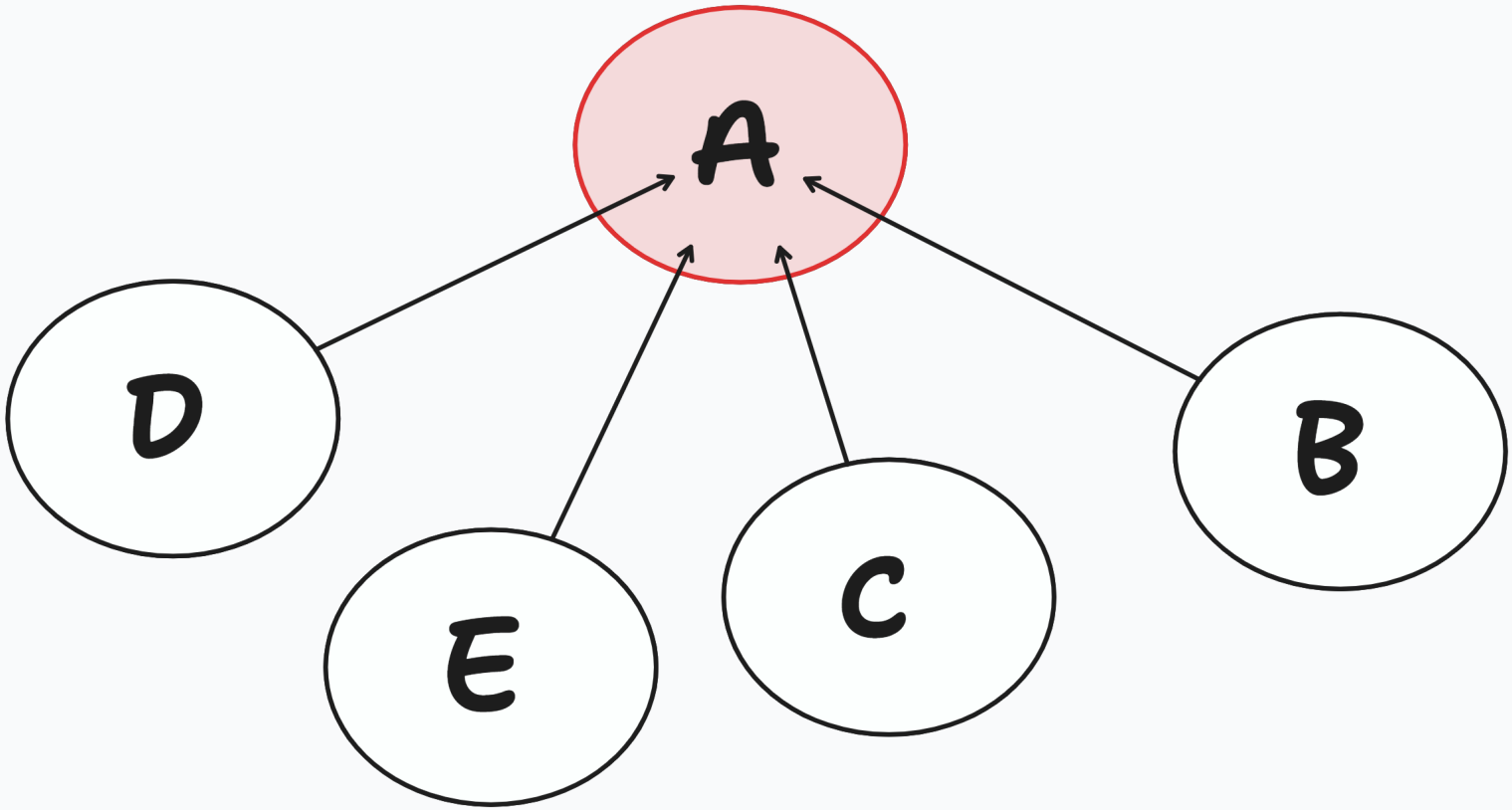
路徑壓縮

舉個例子



路徑壓縮

舉個例子



路徑壓縮 – 實作

將 find_CEO 函式改寫

```
1 int find_CEO(int a) {
2   if(boss[a] == a) return a;
3   boss[a] = find_CEO(boss[a]);
4   return boss[a];
5 }
```

路徑壓縮 – 時間複雜度

按照路徑壓縮實作後，你會發現你通過例題二了!

路徑壓縮 – 時間複雜度

按照路徑壓縮實作後，你會發現你通過例題二了!

但為什麼呢?

路徑壓縮 – 時間複雜度

按照路徑壓縮實作後，你會發現你通過例題二了!

但為什麼呢?

定理

使用路徑壓縮的 DSU，單筆操作均攤複雜度為 $O(\log N)$

路徑壓縮 – 均攤複雜度

均攤複雜度是啥?

我們可以想像每筆操作花的時間有可能不一樣

路徑壓縮 – 均攤複雜度

均攤複雜度是啥?

我們可以想像每筆操作花的時間有可能不一樣

第一筆操作走到很長的鏈，則他花費的時間為 $O(N)$

第二筆操作對於某連通塊的 CEO 求 CEO，則他花費的時間為 $O(1)$

路徑壓縮 – 均攤複雜度

均攤複雜度是啥?

我們可以想像每筆操作花的時間有可能不一樣

第一筆操作走到很長的鏈，則他花費的時間為 $O(N)$

第二筆操作對於某連通塊的 CEO 求 CEO，則他花費的時間為 $O(1)$

所以均攤複雜度是想跟你講，當我們總共做 Q 次詢問的時候

我們將所有操作的單筆複雜度加總，然後除上 Q

這就是單筆均攤複雜度

路徑壓縮 – 時間複雜度

那我們來感性理解一下路徑壓縮為什麼均攤是 $O(\log N)$

路徑壓縮 – 時間複雜度

那我們來感性理解一下路徑壓縮為什麼均攤是 $O(\log N)$

我們想像每次操作都會將長鏈剖掉，形成一個個深度為 1 的短鏈

路徑壓縮 – 時間複雜度

那我們來感性理解一下路徑壓縮為什麼均攤是 $O(\log N)$

我們想像每次操作都會將長鏈剖掉，形成一個個深度為 1 的短鏈

那麼我們前面消除的長鏈肯定會影響到後面操作所詢問的鏈長度

路徑壓縮 – 時間複雜度

那我們來感性理解一下路徑壓縮為什麼均攤是 $O(\log N)$

我們想像每次操作都會將長鏈剖掉，形成一個個深度為 1 的短鏈

那麼我們前面消除的長鏈肯定會影響到後面操作所詢問的鏈長度

再通靈一點，我們就可以說服自己複雜度是 $O(\log N)$ 量級了

路徑壓縮 – 時間複雜度

那我們來感性理解一下路徑壓縮為什麼均攤是 $O(\log N)$

我們想像每次操作都會將長鏈剖掉，形成一個個深度為 1 的短鏈

那麼我們前面消除的長鏈肯定會影響到後面操作所詢問的鏈長度

再通靈一點，我們就可以說服自己複雜度是 $O(\log N)$ 量級了

至於為什麼是感性理解而不是理性理解呢?

啟發式合併

我們來聽另一種優化方式

叫做啟發式合併

啟發式合併

我們來聽另一種優化方式

叫做啟發式合併

我們這次不往跳鏈長的方向優化了!

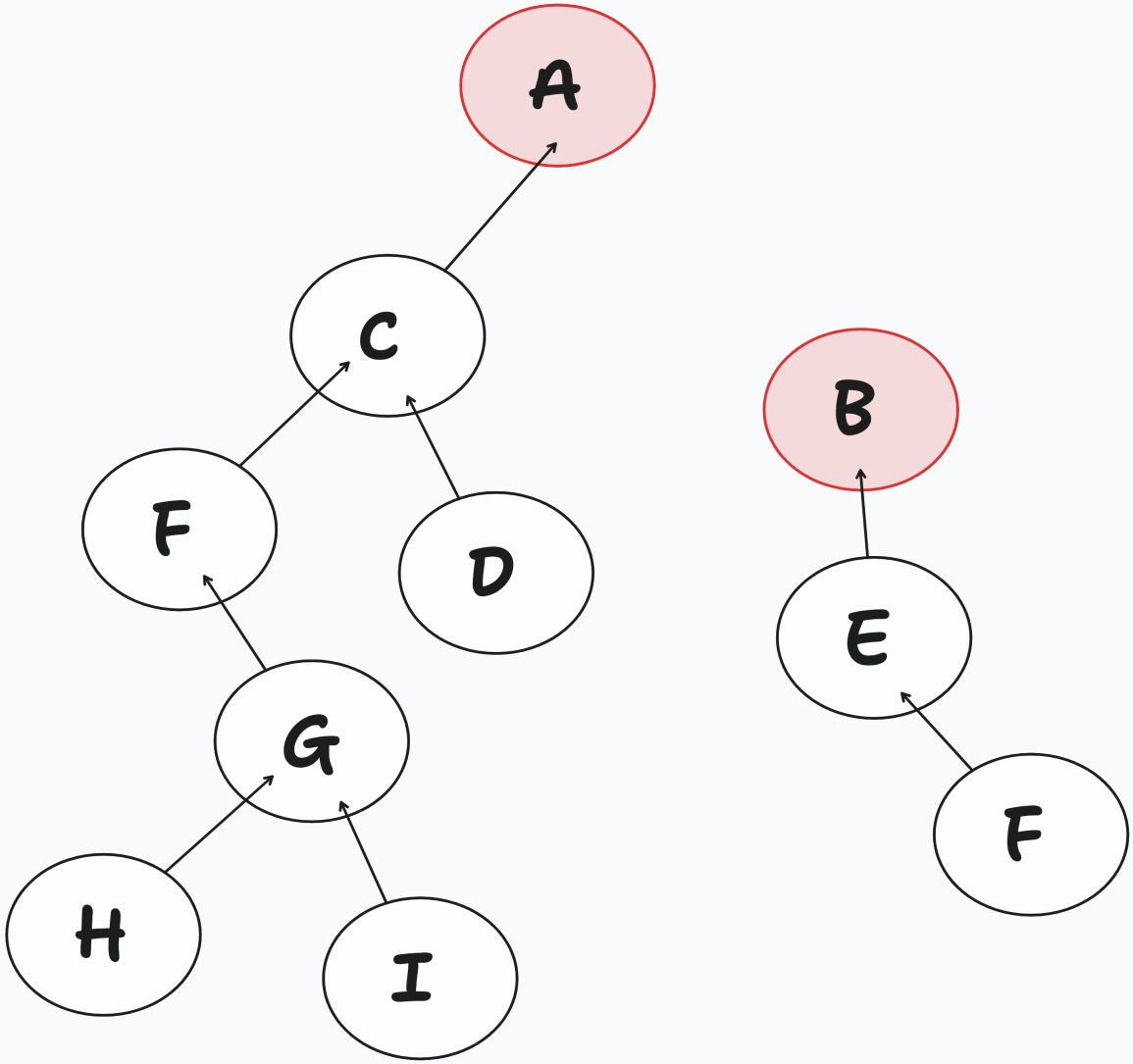
我們嘗試優化合併 (Union) 操作

啟發式合併

觀察當我們擁有兩個連通塊的時候

我們怎麼合併應該才是最佳的?

啟發式合併



啟發式合併

我們發現了

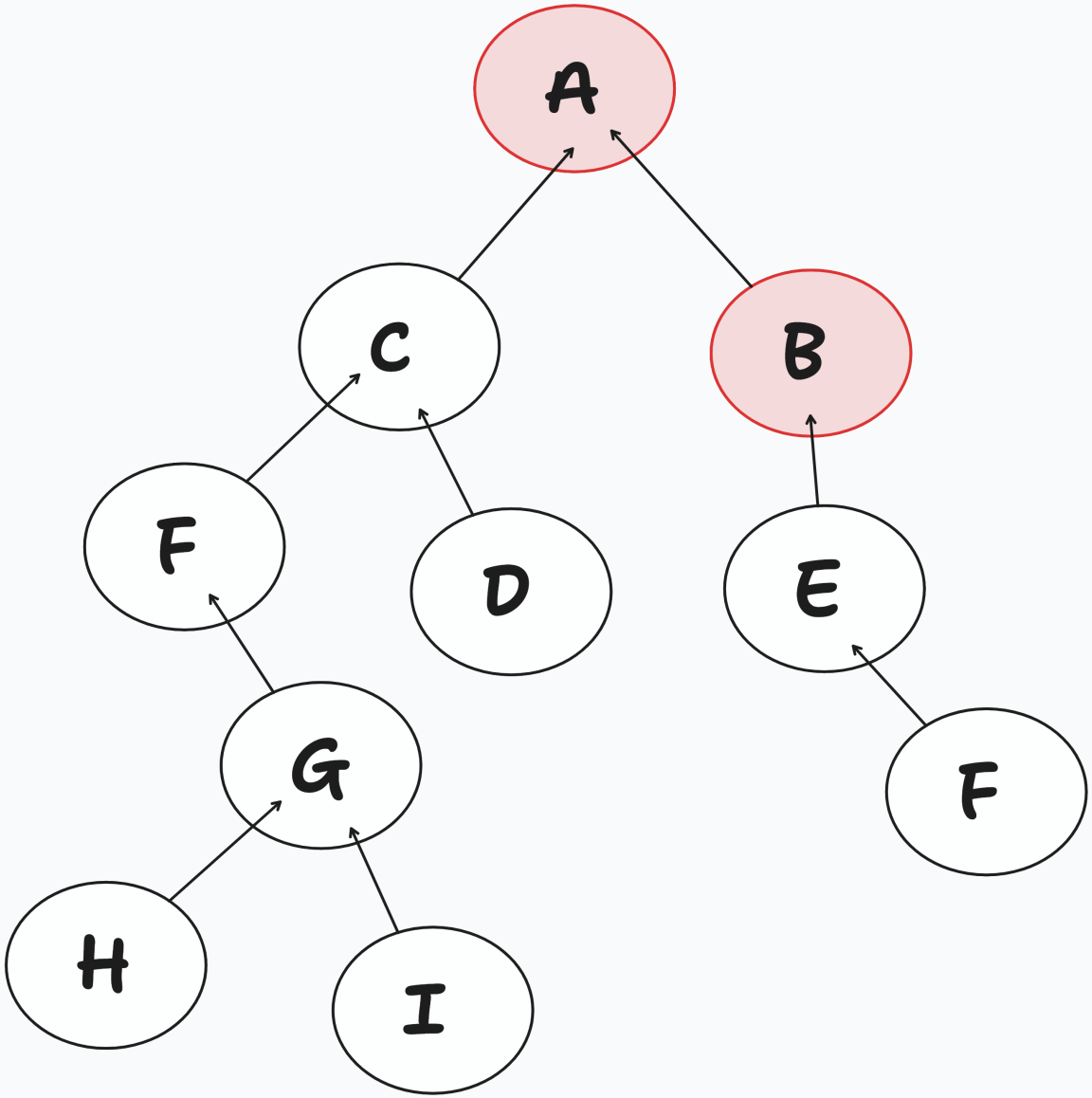
- 深度小的往深度大的指
- 節點數量少的往節點數量多的指

這樣會是最佳的

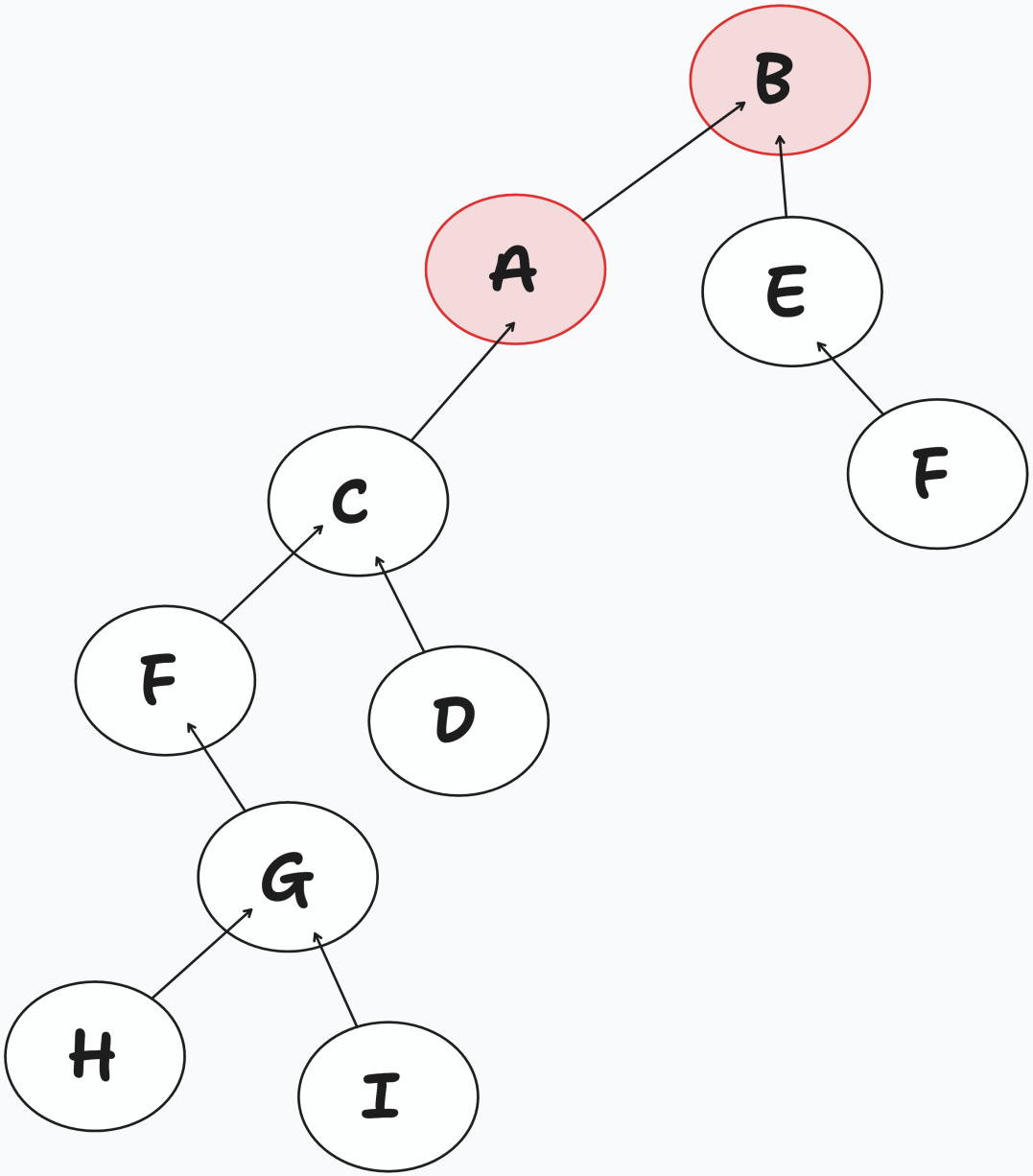
啟發式合併

我們只討論節點少的對節點多的，因為深度大概一樣可以證

啟發式合併



啟發式合併



啟發式合併 – 時間複雜度

定理

每次 Union 操作的時候

將節點少的連通塊指向節點多的連通塊上

最終圖上的最長鏈只有 $\log N$ 量級

啟發式合併 – 時間複雜度

Proof.

當按照由小指向大的合併規則時

我們將 $N-1$ 條邊皆建立出來

我們會發現，每條邊只會讓至多 $\lfloor \frac{\text{節點數量}}{2} \rfloor$ 的節點在他的下面

啟發式合併 – 時間複雜度

Proof.

當按照由小指向大的合併規則時

我們將 $N-1$ 條邊皆建立出來

我們會發現，每條邊只會讓至多 $\lfloor \frac{\text{節點數量}}{2} \rfloor$ 的節點在他的下面

換句話說，就是每個點最多只會走 $O(\log_2 N)$ 條邊

也就是最長鏈長度僅 $\log N$ □

啟發式合併 – 實作

將 Union 操作改成以下

```
// union a & b
int a_CEO = find_CEO(a);
int b_CEO = find_CEO(b);
if(a_CEO != b_CEO) { // 不在同一個連通塊上
  if(size[a_CEO] < size[b_CEO])
    swap(a_CEO, b_CEO); // 使得 a_CEO 節點數量較大
    size[a_CEO] += size[b_CEO]; // 將節點數量合併
    boss[b_CEO] = s_CEO;
}
```

兩種合併方式比較

	路徑壓縮	啟發式合併
find_CEO 複雜度	均攤 $O(\log N)$	$O(\log N)$

發現到兩種方式總體複雜度皆為 $O(Q \log N)$

$1 + 1 = \alpha(n)$

DSU 的複雜度最佳也就這樣嗎?

$1 + 1 = \alpha(n)$

DSU 的複雜度最佳也就這樣嗎?

我們發現剛剛所講的路徑壓縮 & 啟發式合併

二者好像根本不衝突?

$1 + 1 = \alpha(n)$

DSU 的複雜度最佳也就這樣嗎?

我們發現剛剛所講的路徑壓縮 & 啟發式合併

二者好像根本不衝突?

所以我們可以將兩種寫法一起使用!

定理

使用路徑壓縮以及啟發式合併的 DSU

單次操作均攤複雜度為 $O(\alpha(N))$

定理

使用路徑壓縮以及啟發式合併的 DSU

單次操作均攤複雜度為 $O(\alpha(N))$

什麼是 αN?

定理

使用路徑壓縮以及啟發式合併的 DSU

單次操作均攤複雜度為 $O(\alpha(N))$

什麼是 αN?

它叫做反阿克曼函數，是一個很小很小的數字

小到我們可以將它視為常數

定理

使用路徑壓縮以及啟發式合併的 DSU

單次操作均攤複雜度為 $O(\alpha(N))$

什麼是 αN?

它叫做反阿克曼函數，是一個很小很小的數字

小到我們可以將它視為常數

舉個例子，$\alpha(2^{2^{10^{19729}}}) = 4$

Segment Tree

Segment Tree

- 例題一
- 例題二
- 單點修改
- 區間修改?

例題一

例題 (CSES Static Range Minimum Queries)

N 個數字 $x_1, x_2, \ldots, x_N$

有 Q 筆詢問，每次詢問區間 $[l, r]$ 之間的最小值是多少

$1 \leq N, Q \leq 2 \cdot 10^5$

暴力解

我們考慮暴力解

也就是每次將區間 $[l, r]$ 暴力掃過

暴力解

我們考慮暴力解

也就是每次將區間 $[l, r]$ 暴力掃過

複雜度 $O(NQ) = O(4 \cdot 10^{10})$

爆掉了

分治思想

考慮將每次詢問變成分治查詢

分治思想

考慮將每次詢問變成分治查詢

每次將區間 $[l, r]$ 分成 $[l, mid]$ 以及 $[mid + 1, r]$ 詢問

定義 $f(l, r)$ 是區間 $[l, r]$ 的數字最小值

那麼 $f(l, r) = \min(f(l, mid), f(mid + 1, r))$

分治思想

蛤？這樣複雜度不是更糟嗎？

每次還是要遞迴到 $f(l,l), f(l+1,l+1), \ldots, f(r,r)$ 吧？

分治思想

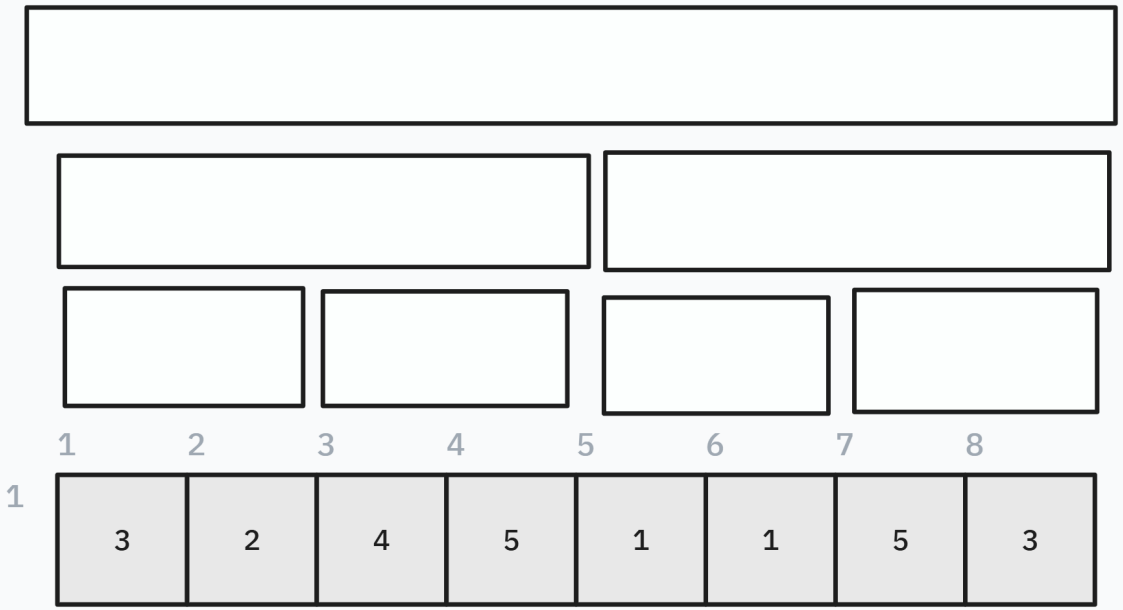
蛤? 這樣複雜度不是更糟嗎?

每次還是要遞迴到 $f(l,l), f(l+1,l+1), \ldots, f(r,r)$ 吧?

真的嗎?

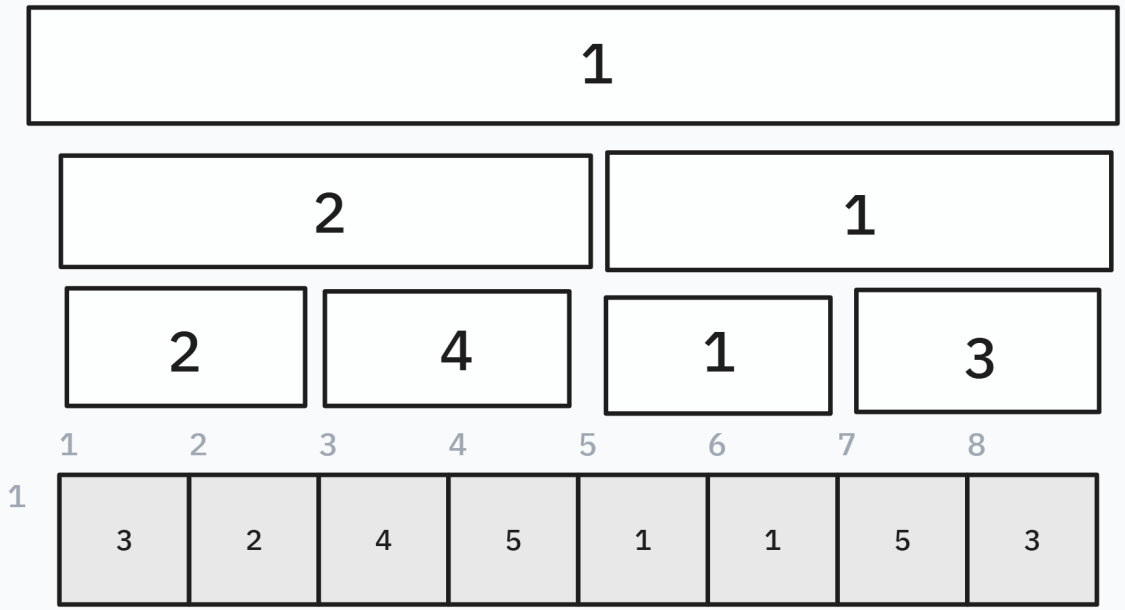
分治思想

我們將函數的區間當作一個塊狀，視覺化出來



分治思想

把區間答案填上去



分治思想

觀察一下

發現了什麼?

分治思想

觀察一下

發現了什麼?

我們將答案記錄下來的話

好像有些區間根本不用再拆分下去吧?

分治思想

按照分治的塊狀分法並將區間答案放到陣列上

形成的一棵很像樹的形狀

這種資料結構就叫做 Segment Tree

中文叫做線段樹

定義結構

那我們要怎麼將每個區間答案存到陣列上?

定義結構

那我們要怎麼將每個區間答案存到陣列上?

我們先定義區間 $[1, N]$ 為編號 1

那麼區間 $[1, mid]$ 為編號 2

區間 $[mid + 1, N]$ 為編號 3

定義結構

那我們要怎麼將每個區間答案存到陣列上?

我們先定義區間 $[1, N]$ 為編號 1

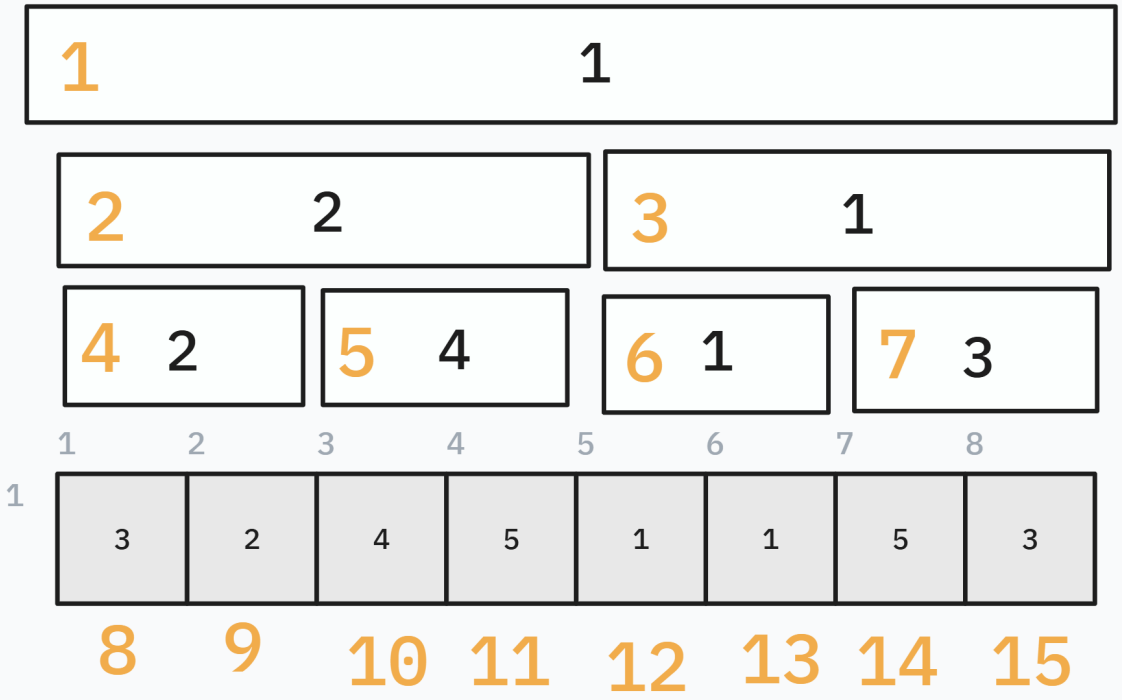
那麼區間 $[1, mid]$ 為編號 2

區間 $[mid + 1, N]$ 為編號 3

以此類推下，就是假設區間 $[l, r]$ 編號為 x

那麼 $[l, mid]$ 就是 $2 \cdot x$，$[mid + 1, r]$ 就是 $2 \cdot x + 1$

定義結構



定義結構

蛤那我對於一個長度為 N 的區間

我要開大小為多少的陣列阿?

定義結構

蛤那我對於一個長度為 N 的區間

我要開大小為多少的陣列阿?

定理 (Segment Tree 的大小)

對於一個長度為 N 的區間

線段樹大小至多開 $4N$ 個

Proof.

最底下那層節點數為 N

往上一層為 $\frac{N}{2}$

再往上為 $\frac{N}{4}$ 以此類推

則總節點數為 $N + \frac{N}{2} + \frac{N}{4} + \ldots + \frac{N}{2^{\log_2 N}}$

根據瘋狂的計算，你可以發現 $= 4N$ □

定義結構

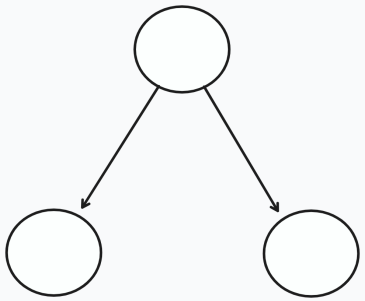
蛤？不會數學怎麼辦？

定義結構

蛤？不會數學怎麼辦？

根據編號規則來說，我們每次應該都會分成一棵完整二元樹

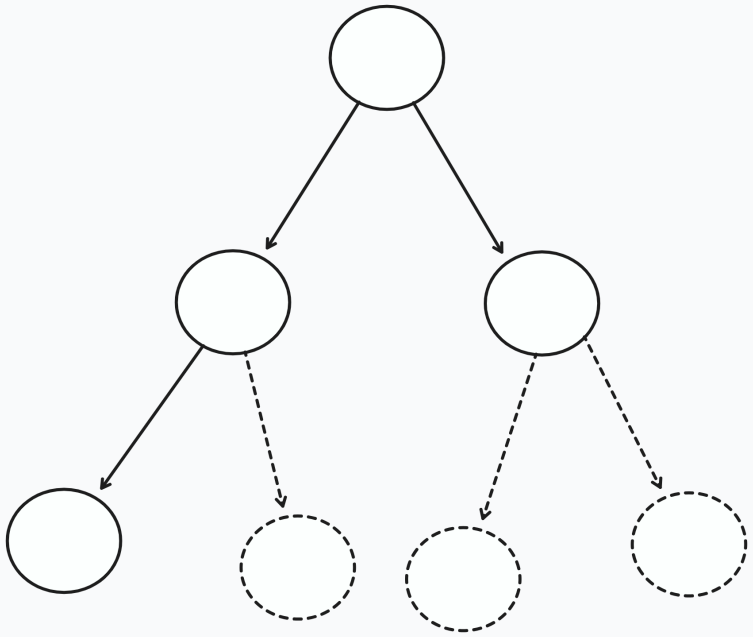
所以假如 N 為二的冪次，節點數量應該是 $2N - 1$



定義結構

發現在 $2N - 1$ 個節點後，又多了一個節點

那麼就會使得整個二元樹多了整整一層，也就需要至多 $4N$ 個節點了



建樹實作

宣告

```
vector<int> tree(n*4+1); // 存樹的資料
vector<int> v(n+1); // 1-based 陣列
```

建樹實作

build 函式

```
1  void build(int l, int r, int id) {
2    if(l == r) { // 已經到最下面的節點(葉節點)
3      tree[id] = v[l];
4      return;
5    }
6    int mid = (l+r)/2;
7    build(l, mid, id*2); // 遞迴往左邊區間
8    build(mid+1, r, id*2+1); // 遞迴往右邊區間
9    tree[id] = min(tree[id*2], tree[id*2+1]); // 記得
         要把下面的節點資訊 pull 上來
10 }
```

建樹實作

呼叫

```
1 build(1, n, 1);
```

詢問

那我們做完了建樹了!

我們要怎麼在詢問中正確的找到對應的節點?

詢問

那我們做完了建樹了!

我們要怎麼在詢問中正確的找到對應的節點?

直接從編號 1 的節點開始做分治 $+$ 剪枝!

詢問 – 實作

```
1  int query(int query_l, int query_r, int l, int r,
       int id) {
2    if(query_l <= l && r <= query_r) return tree[id];
3    int mid = (l+r)/2;
4    if(qr <= mid) return query(query_l, query_r, l,
         mid, id*2);
5    else if(ql > mid) return query(query_l, query_r,
         mid+1, r, id*2+1);
6    else {
7      return min(query(query_l, mid, l, mid, id*2)
8                 ,query(mid+1, query_r, mid+1, r, id
                     *2+1));
9    }
10 }
```

詢問 – 實作

呼叫

```
cout << query(l, r, 1, n, 1) << '\n';
```

分析時間複雜度

這樣詢問複雜度為什麼是好的?

分析時間複雜度

這樣詢問複雜度為什麼是好的?

性質

在線段樹上

區間詢問 $[l, r]$ 至多被拆解成 $log_2 N$ 個線段

證明省略~~(但聽說清大特選面試有考過)~~

分析時間複雜度

所以單次詢問的複雜度為 $O(\log N)$

建樹複雜度為 $O(4N) = O(N)$(不是 $O(N \log N)$ 喔!)

總共複雜度為 $O(N + Q \log N)$

例題二

例題

CSES Dynamic Range Sum Queries N 個數字 $x_1, x_2, \ldots, x_N$

Q 筆操作，有兩種操作

1. 更改 $x_k \to u$
2. 詢問區間 $[l, r]$ 的總和

$1 \leq N, Q \leq 2 \cdot 10^5$

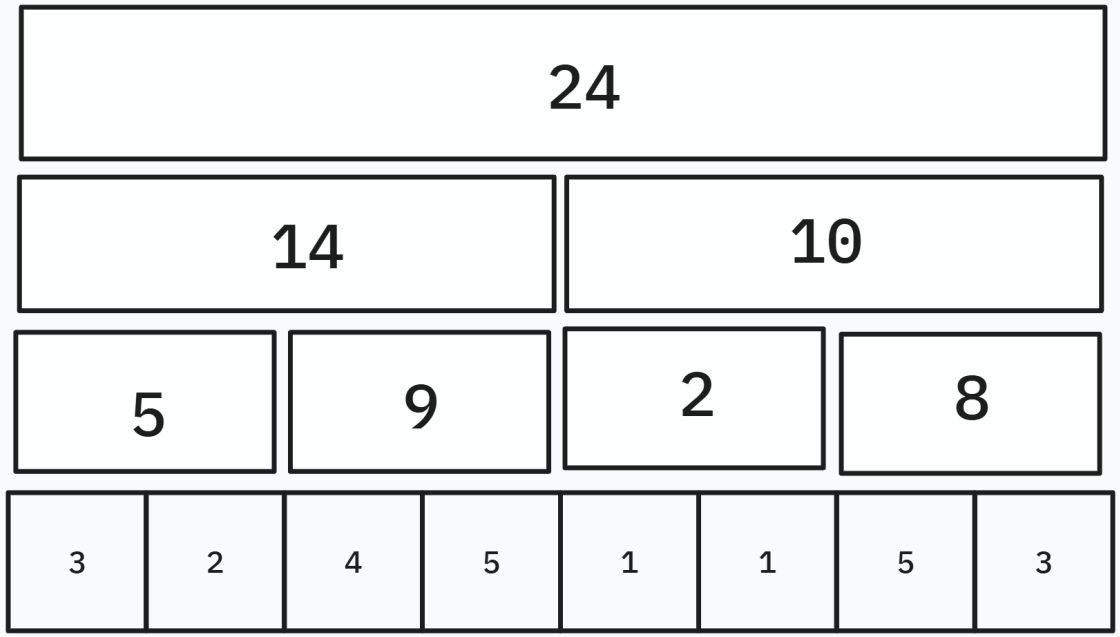
單點修改

多了個單點修改了欸!

我可以在線段樹上修改嗎?

單點修改

觀察一下



單點修改

性質

線段樹上一次單點修改至多只會修改 $\lceil \log_2 N \rceil$ 個節點

單點修改

性質

線段樹上一次單點修改至多只會修改 $\lceil \log_2 N \rceil$ 個節點

Proof.

樹高 $\lceil \log_2 N \rceil$ □

單點修改 – 實作

```
1  void update(int update_i, int update_val, int l,
       int r, int id) {
2    if(l == r) {
3      tree[id] = update_val;
4      return;
5    }
6    int mid = (l+r)/2;
7    if(update_i <= mid) update(update_i, udpate_val,
         l, mid, id*2);
8    else update(update_i, update_val, mid+1, r, id
         *2+1);
9    tree[id] = min(tree[id*2], tree[id*2+1]);
10 }
```

分析時間複雜度

單點修改複雜度 $O(\log N)$，與詢問一樣

所以總體複雜度一樣為 $O(N + Q \log N)$

區間修改?

做 $r - l + 1$ 次單點修改?

區間修改?

做 $r - l + 1$ 次單點修改?

時間複雜度為 $O(NQ \log N)$ 欸!

區間修改?

做 $r - l + 1$ 次單點修改?

時間複雜度為 $O(NQ \log N)$ 欸!

區間修改有兩種方法

- 懶人標記
- 永久化標記

上述二者都能在 $O(\log N)$ 的時間做到區間修改

只是我猜我講不完，所以沒打算教

根號分塊

根號分塊

- 例題一
- 例題二
- 單點修改
- 例題三
- 區間修改
- 分塊 vs 線段樹

例題一

例題 (CSES Static Range Minimum Queries)

N 個數字 $x_1, x_2, \ldots, x_N$

有 Q 筆詢問，每次詢問區間 $[l, r]$ 之間的最小值是多少

$1 \leq N, Q \leq 2 \cdot 10^5$

例題一

剛剛我們已經知道怎麼使用線段樹去解這題了

但是線段樹好像又臭又長，不太好寫欸?

例題一

剛剛我們已經知道怎麼使用線段樹去解這題了

但是線段樹好像又臭又長，不太好寫欸?

$1 \leq N, Q \leq 2 \cdot 10^5$

線段樹解法複雜度為 $O(N + Q \log N) \approx 3 \cdot 10^6$

例題一

剛剛我們已經知道怎麼使用線段樹去解這題了

但是線段樹好像又臭又長，不太好寫欸?

$1 \leq N, Q \leq 2 \cdot 10^5$

線段樹解法複雜度為 $O(N + Q \log N) \approx 3 \cdot 10^6$

好快喔!

例題一

剛剛我們已經知道怎麼使用線段樹去解這題了

但是線段樹好像又臭又長，不太好寫欸?

$1 \leq N, Q \leq 2 \cdot 10^5$

線段樹解法複雜度為 $O(N + Q \log N) \approx 3 \cdot 10^6$

好快喔!

我們能犧牲一點執行時間，使得我們做法更好寫嗎?

定義結構

我們可以發現線段樹所做的事情是

將該層區間切成好幾個小塊

再兩兩相鄰合併，將資訊合併往上拉至上面一層

定義結構

我們可以發現線段樹所做的事情是

將該層區間切成好幾個小塊

再兩兩相鄰合併，將資訊合併往上拉至上面一層

我們真的需要這麼多節點嗎?

定義結構

我們可以發現線段樹所做的事情是

將該層區間切成好幾個小塊

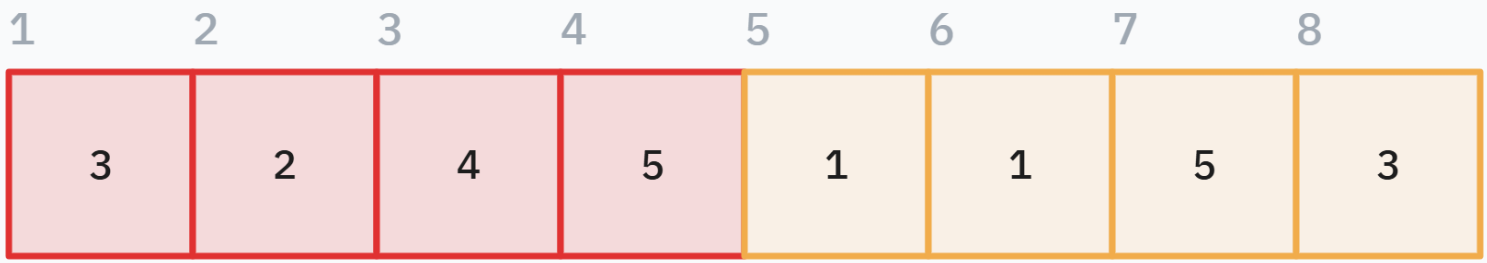
再兩兩相鄰合併，將資訊合併往上拉至上面一層

我們真的需要這麼多節點嗎?

我們考慮將每 B 個東西就分成一整塊

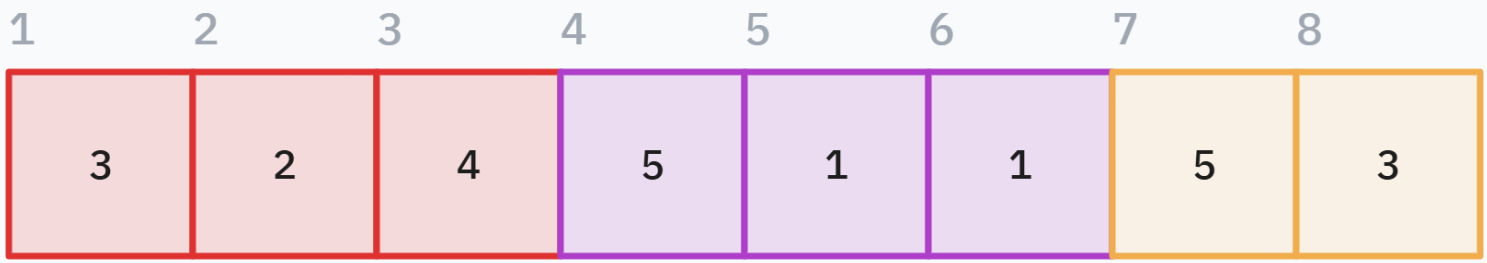
定義結構

$$B = 4$$



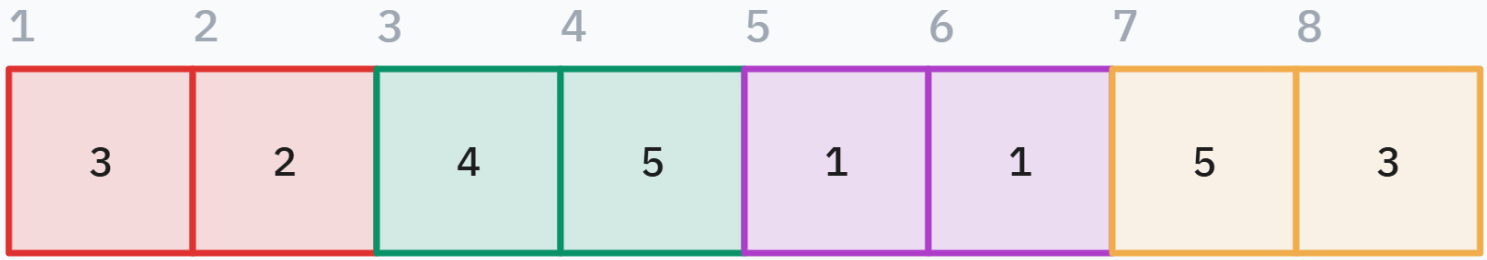
定義結構

$$B = 3$$



定義結構

$$B = 2$$



定義結構

也就是

如果 $B = 3$

$1 \sim 3$ 為第一塊

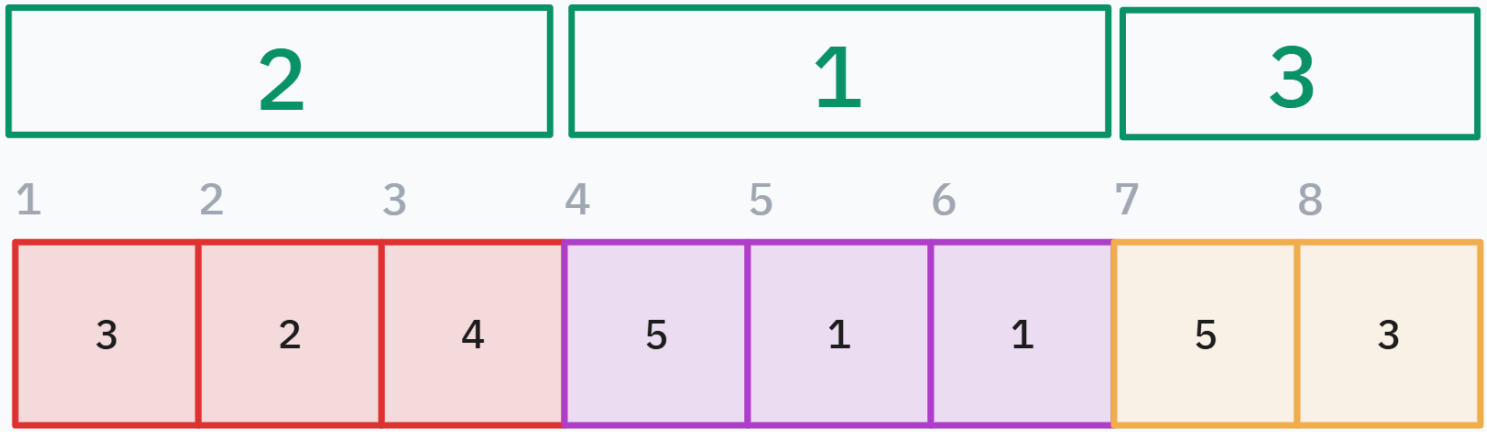
$4 \sim 6$ 為第二塊

$7 \sim 8$ 為第三塊

定義結構

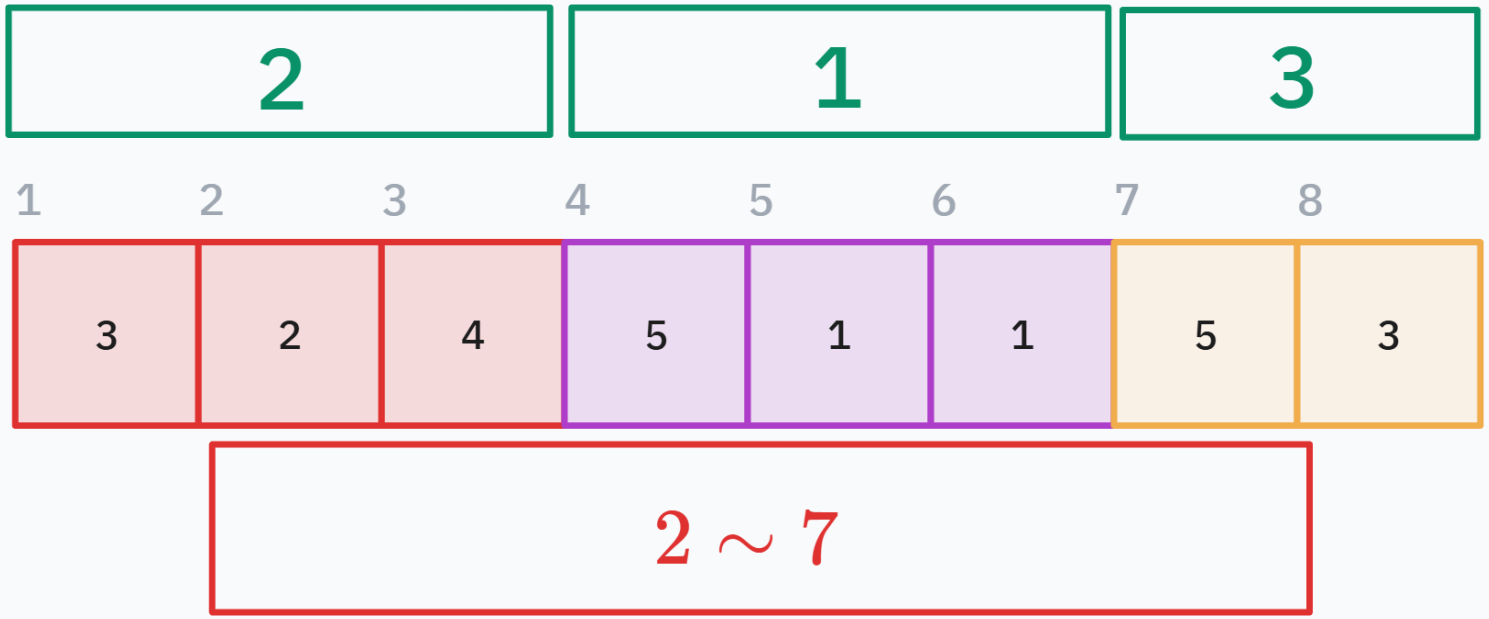
那麼我們對每一組計算整組的答案

$$B = 3$$

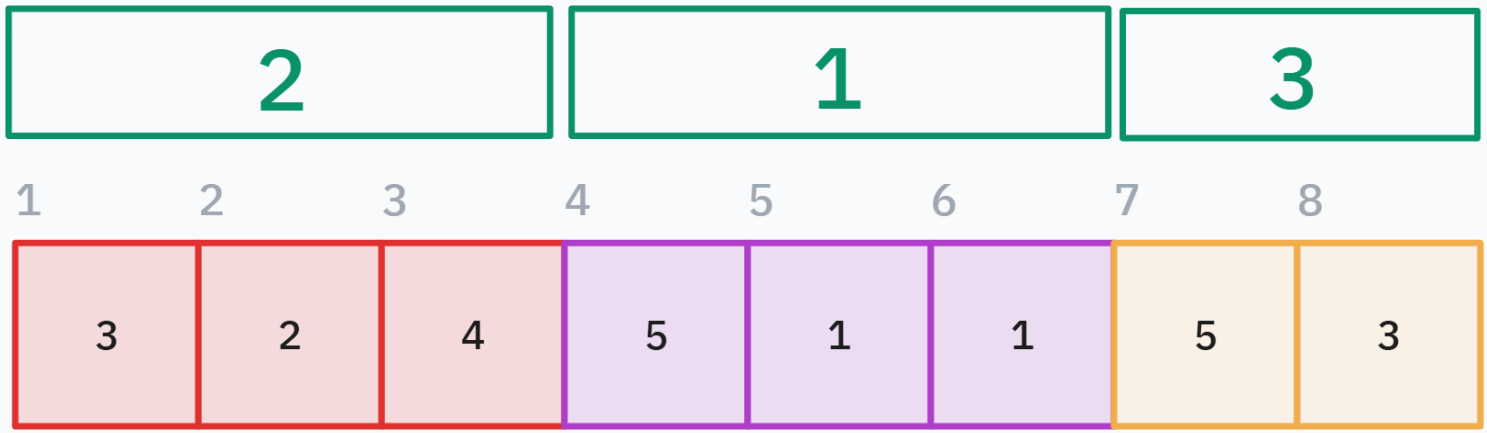


定義結構

觀察詢問依照我們剛剛的塊狀分法會長怎樣?



定義結構



分塊的詢問

發現每次的詢問會裂成好幾段

分塊的詢問

發現每次的詢問會裂成好幾段

分別為最左邊/最右邊的 **碎塊**

還有中間的 **整塊**

分塊的詢問

發現每次的詢問會裂成好幾段

分別為最左邊/最右邊的 **碎塊**

還有中間的 **整塊**

那麼我們就可以透過整塊的答案加速計算了!

塊狀分法?

但我們究竟要分成多少個一塊?

$B = ?$

塊狀分法?

但我們究竟要分成多少個一塊?

$B = ?$

B 可以等於 $1, 2, \ldots, N$

我究竟要取多少才會使得我的分法時間複雜度是合理的?

塊狀分法?

定理 (普通分塊的塊狀大小)

$B = \sqrt{N}$ 為最佳分法

我們可以來簡單證明一下

推論 (普通分塊的塊狀大小)

當我們討論每次詢問的複雜度時

一次詢問最糟情況一定是要讓我們 **碎塊數量最大 & 整塊數量也要最多**

那麼我們可以合理推測 $l = 2, r = N - 1$ 應該是最糟情況

推論 (普通分塊的塊狀大小)

當我們討論每次詢問的複雜度時

一次詢問最糟情況一定是要讓我們 **碎塊數量最大 & 整塊數量也要最多**

那麼我們可以合理推測 $l = 2, r = N - 1$ 應該是最糟情況

因為 $\frac{N}{B} - 2$ 個整塊，$(B - 1) \times 2$ 個碎塊

推論 (普通分塊的塊狀大小)

當我們討論每次詢問的複雜度時

一次詢問最糟情況一定是要讓我們 **碎塊數量最大 & 整塊數量也要最多**

那麼我們可以合理推測 $l = 2, r = N - 1$ 應該是最糟情況

因為 $\frac{N}{B} - 2$ 個整塊，$(B - 1) \times 2$ 個碎塊

那麼我們每次的詢問複雜度為 $O(\frac{N}{B} - 2 + 2 \times (B - 1))$

$= O(\frac{N}{B} + B)$

塊狀分法?

假如你會一點高中數學

Proof.

要使得 $\frac{N}{B} + B$ 最小化 $(B \geq 1)$

發現 $\frac{N}{B}$ 與 B 皆為正整數

塊狀分法?

假如你會一點高中數學

Proof.

要使得 $\frac{N}{B} + B$ 最小化 $(B \geq 1)$

發現 $\frac{N}{B}$ 與 B 皆為正整數

所以使用算幾不等式!

$$\frac{N}{B} + B \geq 2 \cdot \sqrt{\frac{N}{B} \times B} = 2 \cdot \sqrt{N}$$

塊狀分法?

假如你會一點高中數學

Proof.

要使得 $\frac{N}{B} + B$ 最小化 $(B \geq 1)$

發現 $\frac{N}{B}$ 與 B 皆為正整數

所以使用算幾不等式!

$$\frac{N}{B} + B \geq 2 \cdot \sqrt{\frac{N}{B} \times B} = 2 \cdot \sqrt{N}$$

所以最佳情況發生在 $\frac{N}{B} = B = \frac{2 \cdot \sqrt{N}}{2} = \sqrt{N}$

每次詢問複雜度為 $O(2 \cdot \sqrt{N}) = \sqrt{N}$ □

實作

在分塊實作部分上，比較建議使用 0-based

因為編號 i 所在的塊可以使用 $\lfloor \frac{i}{B} \rfloor$ 算出

宣告

```cpp
int B = sqrt(n); // 塊大小
vector<int> v(n); // 0-based 陣列
vector<int> B_val(n); // 整塊資訊
```

實作

先預處理整塊資訊

```
for(int i = 0; i < n; i++) {
  if(i%B == 0) { // 整塊的最左邊
    B_val[i/B] = v[i];
  } else {
    B_val[i/B] = min(B_val[i/B], v[i]);
  }
}
```

實作

詢問

```cpp
// 詢問 [l, r]
int ans = 1e9; // 初始化很大的數字
for(int i = l; i <= r;) {
  if((i/B)*B >= l && (i/B)*B + B-1 <= r) { // 目前所在的整塊是否在區間內
    ans = min(ans, B_val[i/B]);
    i += B;
  } else {
    ans = min(ans, v[i]);
    i++;
  }
}
```

例題二

例題 (CSES - Dynamic Range Minimum Queries)

N 個數字 $x_1, x_2, \ldots, x_N$

有 Q 筆詢問，有兩種操作

1. 更改 $x_k \to u$
2. 詢問區間 $[a, b]$ 的最小值

$1 \leq N, Q \leq 2 \cdot 10^5$

例題二

多了單點操作，分塊還可做嗎?

例題二

多了單點操作，分塊還可做嗎?

我們發現，對於一個值 x_i

他影響到的東西只有他所在的整塊以及自己本身!

例題二

多了單點操作，分塊還可做嗎?

我們發現，對於一個值 x_i

他影響到的東西只有他所在的整塊以及自己本身!

所以我們可以修改該值，然後暴力更新該個整塊資訊

時間複雜度

複雜度很好證明

我們暴力更新整塊資訊，最多也才遍歷 B 個東西，也就是 $\sqrt{N}$ 個

時間複雜度

複雜度很好證明

我們暴力更新整塊資訊，最多也才遍歷 B 個東西，也就是 $\sqrt{N}$ 個

所以預處理花費 $O(N)$

詢問總共花費 $O(Q\sqrt{N})$

更改總共花費 $O(Q\sqrt{N})$

總時間複雜度 $O(N + Q\sqrt{N})$

例題三

例題 (CSES Range Update Queries)

有 N 個數字 $x_1, x_2, \ldots, x_N$

有 Q 筆詢問，有兩種操作

1. 將區間 $[a, b]$ 的元素都加上 u
2. 詢問 x_k 的值

$1 \leq N, Q \leq 2 \cdot 10^5$

區間修改?

換成區間修改，單點查詢，這樣分塊要怎麼做?

區間修改?

換成區間修改，單點查詢，這樣分塊要怎麼做?

我們發現，區間修改根本跟區間詢問一樣吧?

我們可以用同一種切法，把區間修改切成整塊以及碎塊

區間修改?

換成區間修改，單點查詢，這樣分塊要怎麼做?

我們發現，區間修改根本跟區間詢問一樣吧?

我們可以用同一種切法，把區間修改切成整塊以及碎塊

對於碎塊就暴力修改

對於整塊就紀錄整塊的一起加值!

區間修改實作

```
1  // 修改 [l, r] += x
2  // B_add 是整塊的一起加值量
3  for(int i = l; i <= r; i++) {
4    if((i/B)*B >= l && (i/B)*B + B-1 <= r) { // 目前
         所在的整塊是否在區間內
5      B_add[i/B] += x;
6      i += B;
7    } else {
8      v[i] += x;
9      i++;
10   }
11 }
```

區間修改實作

單點詢問

```
1 cout << v[k] + B_add[k/B] << '\n';
```

區間修改/區間查詢

同理，根號分塊其實可以做到區間修改/區間查詢!

區間修改/區間查詢

同理，根號分塊其實可以做到區間修改/區間查詢!

分塊 vs 線段樹

這邊先教根號分塊的區間修改，不教線段樹的區間修改

是因為線段樹的區間修改的懶人標記較為抽象，根號分塊的區間修改比較容易理解

分塊 vs 線段樹

這邊先教根號分塊的區間修改，不教線段樹的區間修改

是因為線段樹的區間修改的懶人標記較為抽象，根號分塊的區間修改比較容易理解

但別忘了！根號分塊的好實作性是由犧牲複雜度導致的

$\sqrt{N}$ 與 $\log N$ 還是差蠻多的!

分塊 vs 線段樹

這邊先教根號分塊的區間修改，不教線段樹的區間修改

是因為線段樹的區間修改的懶人標記較為抽象，根號分塊的區間修改比較容易理解

但別忘了！根號分塊的好實作性是由犧牲複雜度導致的

$\sqrt{N}$ 與 $\log N$ 還是差蠻多的!

BTW 其實有題目是只能用分塊解，但不能用線段樹解的

BIT

BIT

例題一

BIT 這麼強喔?

區間修改?

例題一

例題

CSES Dynamic Range Sum Queries N 個數字 $x_1, x_2, \ldots, x_N$

Q 筆操作，有兩種操作

1. 更改 $x_k \to u$
2. 詢問區間 $[l, r]$ 的總和

$1 \leq N, Q \leq 2 \cdot 10^5$

例題一

假如你剛剛的剛剛有認真聽課

這是線段樹的例題二

例題一

假如你剛剛的剛剛有認真聽課

這是線段樹的例題二

剛剛的線段樹感覺好長好難刻阿!

有沒有很好刻，且複雜度不會退化到 $\sqrt{N}$ 的解法?

轉化問題

我們先將問題轉化一下

區間詢問 $[l, r]$ 的和

其實可以轉化成 $pre_r - pre_{l-1}$ 吧?

轉化問題

我們先將問題轉化一下

區間詢問 $[l, r]$ 的和

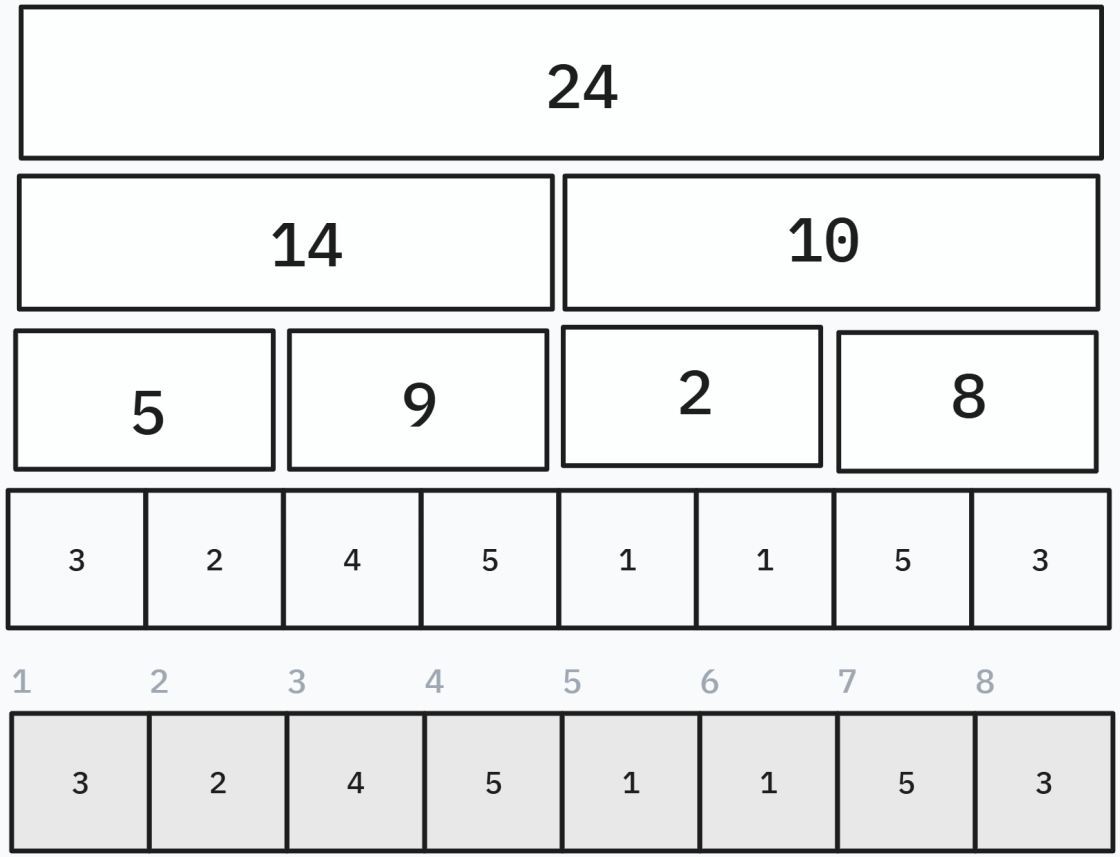
其實可以轉化成 $pre_r - pre_{l-1}$ 吧?

所以我們現在變成了，我們需要一個資料結構

使得我們可以快速的求得前綴和 & 單點修改前綴和

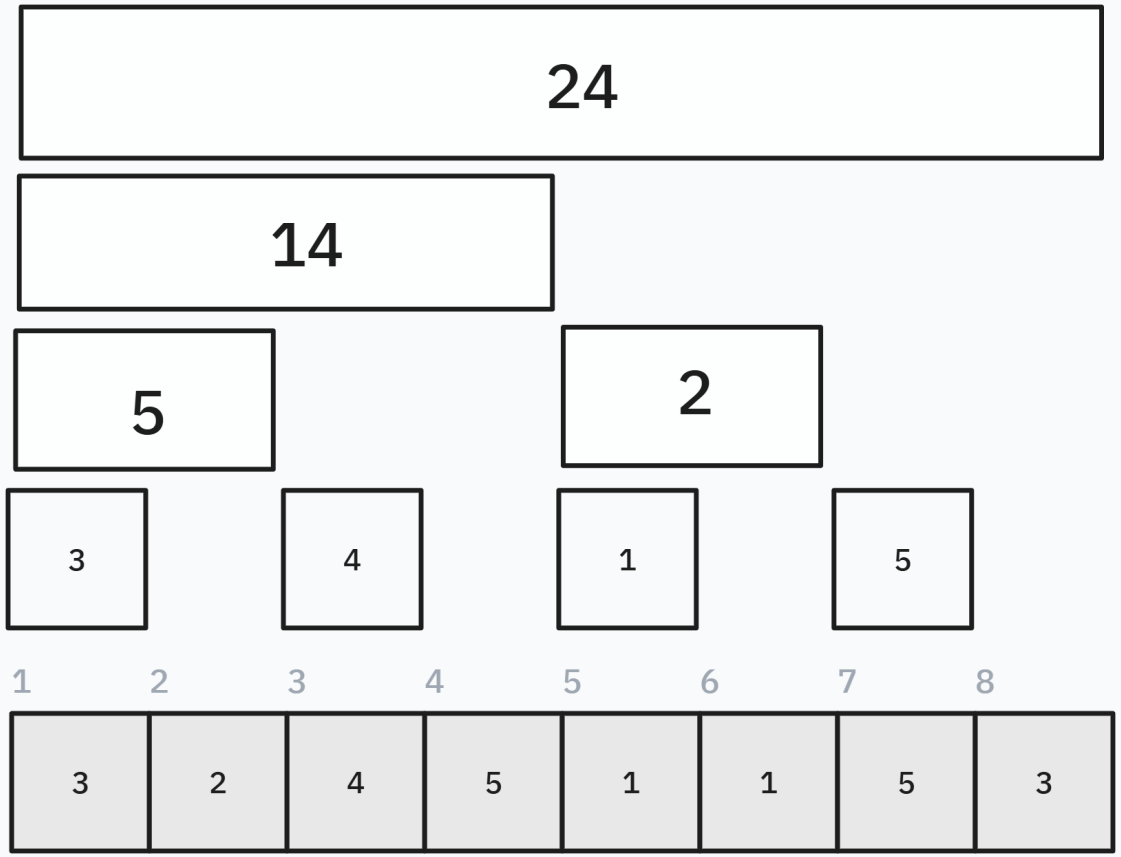
定義結構

這是線段樹的結構



定義結構

我們將每一個節點的右子樹砍掉，觀察一下發生了什麼事情?



定義結構

我們發現每次詢問 $1 \sim i$ 的前綴和 pre_i

好像是每次往上找第一個覆蓋 i 的點

然後把 i 設為 左端點 $- 1$

以此類推，直到最左邊

最終加每一塊總和加起來就是答案了!

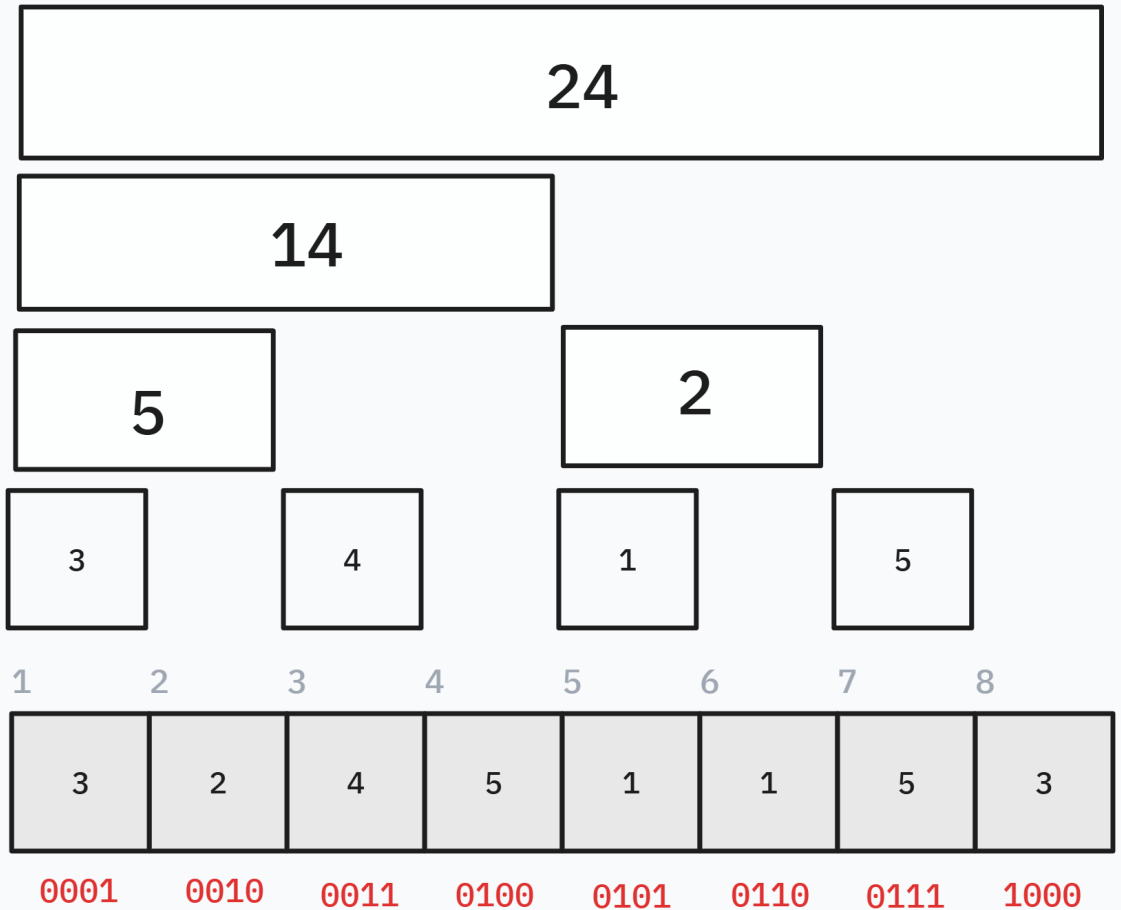
定義結構

我們要怎麼快速地找左端點以及塊編號呢?

定義結構

我們要怎麼快速地找左端點以及塊編號呢?

我們引入一下二進制，觀察一下 1 ~ 8 的二進制長怎樣



定義結構

我們發現，當 i 在 2^k 的位元是 1 時

那麼我們的答案就會用到第 k 層的 $i \to 1$ 遇到的第一個塊

定義結構

我們發現，當 i 在 2^k 的位元是 1 時

那麼我們的答案就會用到第 k 層的 $i \to 1$ 遇到的第一個塊

更關鍵的觀察是，每一塊的右端點都不一樣欸!

所以我們的右端點可以用 i 來表示

lowbit(i)

先引入一個東西，它叫做 lowbit(i)

定義 (lowbit(i))

假設 i 這個數字在二進制下的最低 1 的位置為 k

那麼 $\text{lowbit}(i) = 2^k$

舉個例子來講

$13_{(2)} = 1101$

所以 $k = 0, \text{lowbit}(13) = 2^0 = 1$

$16_{(2)} = 10000$

所以 $k = 4, \text{lowbit}(16) = 2^4 = 16$

定義結構

假如你超級會觀察，你會發現

$pre_i =$ (以i為結尾的塊) + (以$i -$ lowbit(i)為結尾的塊) + ...

所以我們一直減掉 lowbit(i) 就可以一直我左上角跳過去了!

定義結構

假如你超級會觀察，你會發現

$pre_i =$ (以i為結尾的塊) + (以$i -$ lowbit(i)為結尾的塊) + ...

所以我們一直減掉 lowbit(i) 就可以一直我左上角跳過去了!

假如你沒觀察出來的話，
~~那你可以先背起來，留著你變強的時候思考~~

定義結構

然後你再再再超會觀察，你會發現

$i + \text{lowbit}(i)$ 會是該個塊往上被第一個覆蓋的塊編號

所以更改 i 這個點的話，我們就一直加 $\text{lowbit}(i)$，直到超出編號 N

整個路徑上的塊就是你會修改到的所有塊

實作

BIT 的實作用 1-based 才會對，0-based 會有另一個比較麻煩的實作方法

宣告

```
vector<int> bit(n+1); // 1-based
```

lowbit(x)

```
x&-x
```

實作

單點修改

```
1 void update(int i, int x) { // v[i] += x
2   while(i <= n) {
3     bit[i] += x;
4     i += i&-i;
5   }
6 }
```

實作

查詢 pre_i

```
int query(int i) {
  int re = 0;
  while(i > 0) {
    re += bit[i];
    i -= i&-i;
  }
  return re;
}
```

實作

所以查詢區間 $[l, r]$

就是 $pre_r - pre_{l-1}$

```
cout << query(r) - query(l-1) << '\n';
```

BIT 這麼強喔?

BIT 常數比線段樹還小

BIT 實作量也超級少

那我們為什麼還需要線段樹?

BIT 犧牲了什麼?

BIT 本質上是維護前綴和達到快速運算

而有些題目不能用前綴和

BIT 犧牲了什麼?

BIT 本質上是維護前綴和達到快速運算

而有些題目不能用前綴和

更正確來說應該是

$\max(1, 2, 3, 4, 5) = 5$

你沒辦法從 5 推出來 $\max(1, 2, 3, 4, 5)$

區間修改

BIT 其實可以做到區間修改

只是需要一點數學推導

提示一下:

在 BIT 存差分

名詞定義

BIT 其實是縮寫

它叫做 Binary Index Tree

而他也有一個別稱叫做 Fenwick Tree

小結

小結

小結

剛剛教了四個東西

- DSU

小結

剛剛教了四個東西

- DSU
- Segment Tree

小結

剛剛教了四個東西

- DSU
- Segment Tree
- 根號分塊

小結

剛剛教了四個東西

- DSU
- Segment Tree
- 根號分塊
- BIT

小結

除了 DSU

其他三個本質上都是在解差不多的序列問題

小結

除了 DSU

其他三個本質上都是在解差不多的序列問題

只是有些人比較特化到某一類題目

有些人犧牲複雜度，使得可以做更多事情

他們各有不同，所以建議都學一下

小結

	Segment Tree	BIT	分塊
單點查詢	$O(\log N)$	$O(\log N)$	$O(1)$
單點修改	$O(\log N)$	$O(\log N)$	$O(\sqrt{N})$
區間查詢	$O(\log N)$	$O(\log N)$	$O(\sqrt{N})$
區間修改	$O(\log N)$	$O(\log N)$	$O(\sqrt{N})$

Sparse Table

Sparse Table

- 介紹
- 例題一

前言

首先你要先會倍增

前言

首先你要先會倍增

然後講師真的覺得 Sparse Table 很沒用

但因為聽說 OI 題很常考

所以我們還是學一下

性質

性質 (答案疊加性)

$$\max(1,2,3,4,5) = \max(\max(1,2,3,4), \max(2,3,4,5))$$

所以 min/max 其實具有疊加性

例題一

例題 (CSES Static Range Minimum Queries)

N 個數字 $x_1, x_2, \ldots, x_N$

有 Q 筆詢問，每次詢問區間 $[l, r]$ 之間的最小值是多少

$1 \leq N, Q \leq 2 \cdot 10^5$

例題一

考慮靜態詢問 $[l, r]$ 的最小值

例題一

考慮靜態詢問 $[l, r]$ 的最小值

發現只要你很會預處理

我們其實可以把 $[l, l+???]$ 區間與 $[r-???, r]$

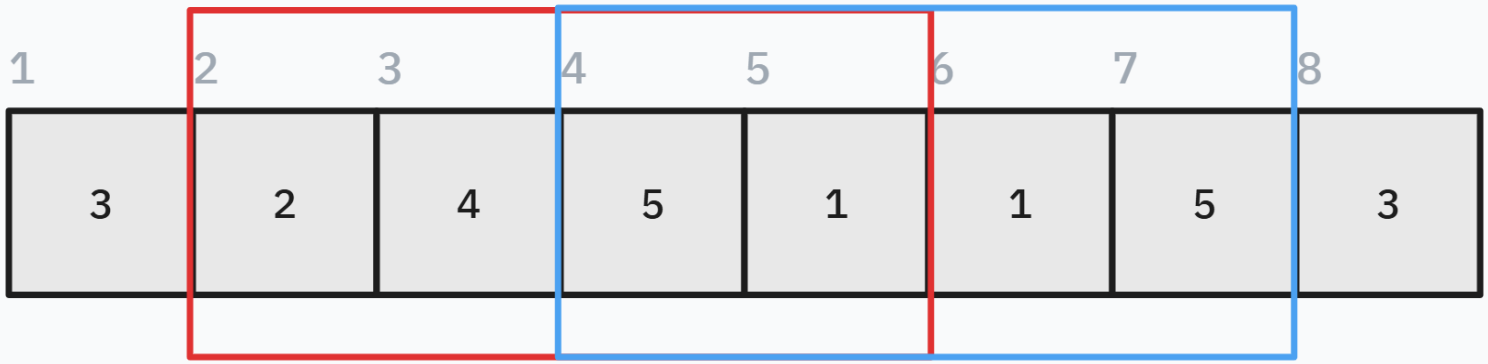
疊在一起，然後求 $\min$

例題一

有點抽象?

例題一

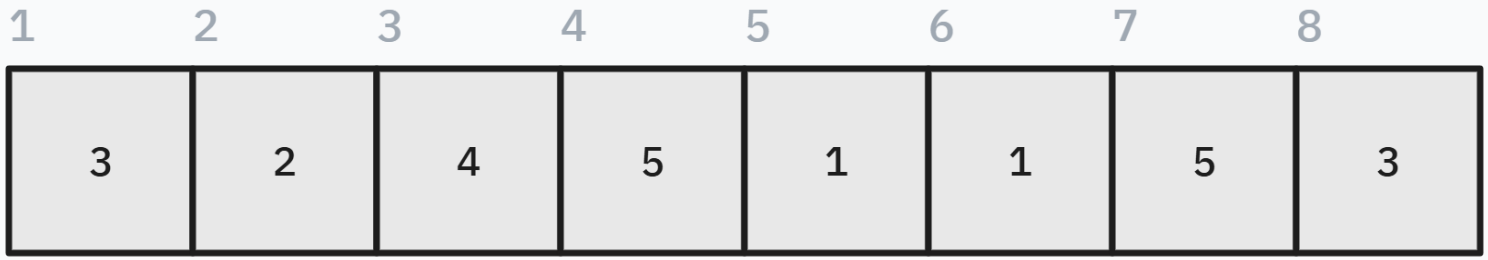
有點抽象?



我們可以把紅色區塊跟藍色區塊疊起來求得答案

例題一

舉個例子



例題一

我們要怎麼預處理?

要對哪些??? 預處理?

例題一

我們要怎麼預處理?

要對哪些??? 預處理?

我們發現一個區間他很小的話

我們對於大區間的預處理就不能套在他們上面了

例題一

我們要怎麼預處理?

要對哪些??? 預處理?

我們發現一個區間他很小的話

我們對於大區間的預處理就不能套在他們上面了

對於大區間來說，小區間的預處理就不夠用了

方法

我們考慮只對於??? $= 2^k$ 做預處理

方法

我們考慮只對於??? $= 2^k$ 做預處理

那麼總共也才做 $\log_2 N$ 次

每次最多只會有 N 個區間

也就是花 $O(N \log N)$ 的時間預處理

方法

定理

對於一個區間 $[l, r]$

我們必定可以找到兩個 2^k 長度的區間

使得兩區間疊起來的範圍為 $[l, r]$

方法

定理

對於一個區間 $[l, r]$

我們必定可以找到兩個 2^k 長度的區間

使得兩區間疊起來的範圍為 $[l, r]$

證明可以用反證法，反正很簡單

實作

宣告

```
vector<vector<int> > sparse_table(32, vector<int>(n+1));
vector<int> v(n+1); // 1-based
```

預處理

```
for(int i = 1; i <= n; i++) sparse_table[0][i] = v[i];
for(int k = 1; k <= 30; k++) {
  for(int i = 1; i <= n; i++) {
    sparse_table[k][i] = min(sparse_table[k-1][i],
        sparse_table[k-1][i+(1<<(k-1))]);
  }
}
```

詢問

```
// 詢問 [l, r] min
int k = __lg(r-l+1);
int length = (1<<k);
cout << min(sparse_table[k][l], sparse_table[k][r-
    length+1]) << '\n';
```

時間複雜度

分析一下時間複雜度

時間複雜度

分析一下時間複雜度

預處理 $O(N \log N)$

查詢可以達到神奇的 $O(1)$!

時間複雜度

分析一下時間複雜度

預處理 $O(N \log N)$

查詢可以達到神奇的 $O(1)$!

只是通常比較不會有題目卡你 Q 超大，然後 N 超小

時間複雜度

分析一下時間複雜度

預處理 $O(N \log N)$

查詢可以達到神奇的 $O(1)$!

只是通常比較不會有題目卡你 Q 超大，然後 N 超小

通常用到 Sparse Table 的時候，都是好難的題目

時間複雜度

分析一下時間複雜度

預處理 $O(N \log N)$

查詢可以達到神奇的 $O(1)$!

只是通常比較不會有題目卡你 Q 超大，然後 N 超小

通常用到 Sparse Table 的時候，都是好難的題目

但遇到靜態區間取 min，Sparse Table 比線段樹好刻!

柯朵莉樹

柯朵莉樹

例題一

例題一

例題 (TOJ 960 快樂線段樹 2.0)

給一個長度為 N 的陣列 A

Q 筆操作，兩種

1. 詢問區間 $[l, r]$ 是否皆相同
2. 區間改值 $[l, r] \to x$

$1 \leq N \leq 2 \cdot 10^6$

$1 \leq Q \leq 10^6$

什麼是珂朵莉?

什麼是珂朵莉?

什麼是珂朵莉?

簡介

簡單來講，珂朵莉是由一題 Codeforces 上的題目而出名的技巧

精隨感覺就是用 set 亂做

定義

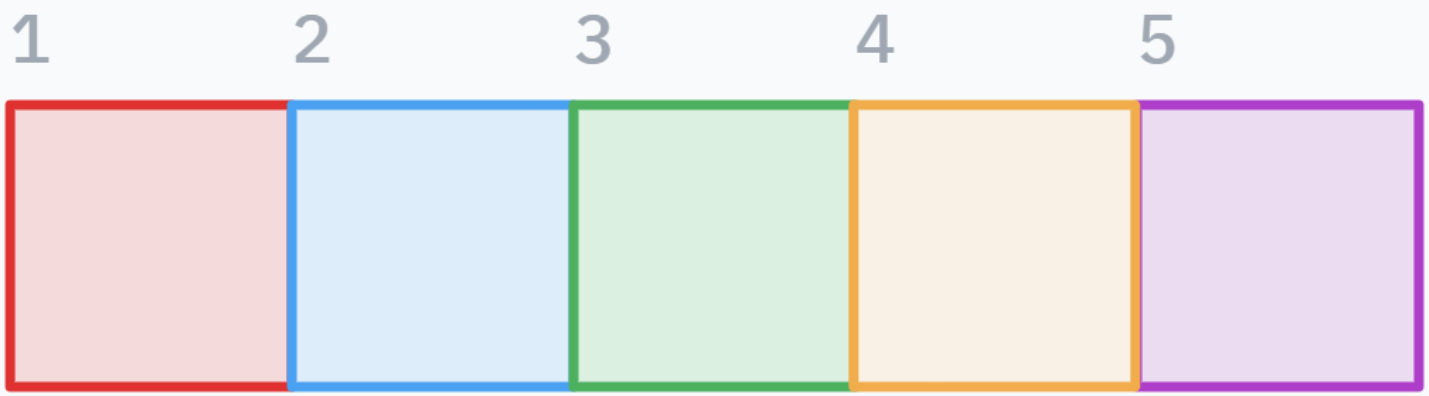
首先我們先將數值相同的當作同一種顏色

那麼陣列上就會出現不同種顏色

定義

首先我們先將數值相同的當作同一種顏色

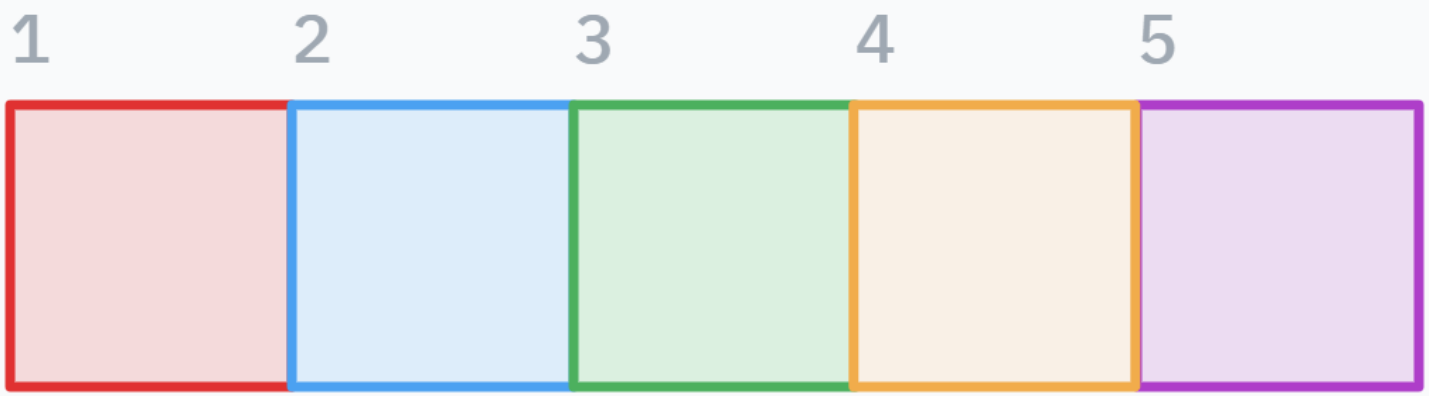
那麼陣列上就會出現不同種顏色



定義

首先我們先將數值相同的當作同一種顏色

那麼陣列上就會出現不同種顏色



那麼區間改值就是把一個區間段塗上一種顏色

定義

我們定義一個節點 (i, C)

代表著從第 i 個格子開始是顏色 C

定義

我們定義一個節點 (i, C)

代表著從第 i 個格子開始是顏色 C

那麼我們是否可以使用以上定義來表達出一個陣列呢?

定義

舉個例子

1 2 3 3 3 5 4 4

其實可以表示為

$(1,1)$ $(2,2)$ $(3,3)$ $(6,5)$ $(7,4)$

作法

我們發現把節點放到 set 上面

set 會自動將 (i, C) 按照 i 去做排序

使得會按照 index 排序

作法

我們發現把節點放到 set 上面

set 會自動將 (i, C) 按照 i 去做排序

使得會按照 index 排序

這時候我們就能用二分搜去找到包含在區間內的最左邊的節點了了!!

作法

那我們要怎麼判斷區間 $[l, r]$ 是否相同?

作法

那我們要怎麼判斷區間 $[l, r]$ 是否相同?

區間 $[l, r]$ 僅被一個節點包含!

實作

宣告

```
set<pair<int, int> > se; // 儲存 (i, C) 的 set
vector<int> v(n+1); // 1-based 陣列
se.insert({0, 0}); // 左邊的邊界
se.insert({n+1, 0}); // 右邊的邊界
```

預處理

```
  for(int i = 1; i <= n; i++) {
    if(se.empty() || se.back()->second != v[i]) {
      se.insert({i, v[i]});
    }
  }
```

實作

區間塗色

```
// [l, r] -> x
auto lb = se.lower_bound(make_pair(l, 0));
int last_color = -1;
if(lb->first != l) { // 第 l 格的顏色起始點在前面
  last_color = prev(lb)->second;
}
while(lb->first <= r) {
  last_color = lb->second;
  lb = se.erase(lb);
}
if(lb->first != r+1) se.insert({r+1, last_color});
se.insert({l, x});
```

實作

詢問

```
auto lb = se.lower_bound(make_pair(l+1, 0));
if(lb->first > r) {
  // 同樣顏色
} else {
  // 不一樣
}
```

時間複雜度

對於珂朵莉的時間複雜度證明

有點困難

時間複雜度

對於珂朵莉的時間複雜度證明

有點困難

所以感性理解每次修改區間的時候會刪掉好多個節點

所以這樣是好的

時間複雜度

對於珂朵莉的時間複雜度證明

有點困難

所以感性理解每次修改區間的時候會刪掉好多個節點

所以這樣是好的

反之，假如你的珂朵莉沒有做到每一次區間詢問很多個點的時候，把點都刪掉

那麼它的複雜度會壞掉

fun fact

珂朵莉樹又稱 ODT(老司機樹)

講題 + 實作時間

講題 + 實作時間

- DSU 的各種應用
- Segment Tree
- BIT
- 珂朵莉樹

例題 (TOJ 791 遜砲音音)

N 點，M 邊

Q 筆詢問，每次拔掉一條邊 $a \leftrightarrow b$

對於每一筆詢問輸出當前的連通塊數量

$2 \leq N \leq 2 \cdot 10^5$

$1 \leq M \leq \min(\frac{N(N-1)}{2}, 2 \cdot 10^5)$

$1 \leq Q \leq M$

TOJ 309 群集問題

例題

N 個點，Q 筆詢問

兩種詢問

1. 新增一條邊 $a \leftrightarrow b$
2. 問 a 所在的連通塊是否存在環

$1 \leq N, Q \leq 10^5$

TOJ 266 水桶和水球

例題 (TOJ 266 水桶和水球)

一開始有 N 個水桶，一開始水桶都是空的

有 Q 筆操作，兩種

1. 在 p 號水桶內放入一顆價值為 v 的水球
2. 從區間 $[l, r]$ 的水桶中拿出價值最大的那顆，輸出出來，然後丟掉 (假如有一樣的以水桶編號大的優先)

$1 \leq N, Q \leq 10^4$

TOJ 11 bubble

例題 (TOJ 11 bubble)

給一個長度為 N 的陣列 A

問至少要進行多少次相鄰交換的操作才會使得整個陣列被排序好

$3 \leq N \leq 2 \cdot 10^6$

$1 \leq A_i \leq N$

TOJ 12 遙控器

例題 (TOJ 12 遙控器)

有 $N \times M$ 個按鍵

Q 筆操作，兩種操作

1. 把第 i 列第 j 行改成 k
2. 詢問在矩形 $(x_1, y_1), (x_2, y_2)$ 之內的按鍵總和

$1 \leq N, M \leq 3000$

$1 \leq Q \leq 10^4$

TOJ 748 批量種田

例題 (TOJ 748 批量種田)

N 個田地，各自上面有農作物 $c_1, c_2, \ldots, c_N$

Q 筆操作，每次將區間 $[l, r]$ 種上作物 x_i

根據每筆操作，輸出在種上新作物之前，區間 $[l, r]$ 有多少個不一樣的農作物

$1 \leq N, Q \leq 3 \cdot 10^5$

資結中毒

資結中毒

TOJ 490 246 三硝基甲苯

例題 (TOJ 490 246 三硝基甲苯)

給你 N 個數字，Q 筆詢問

每次將 $[1, i]$ 的區間塗上顏色

最後問你這個區間長怎樣

$1 \leq N, Q \leq 10^6$

CSES Static Range Sum Queries

例題 (CSES Static Range Sum Queries)

給你長度為 N 的陣列 x

Q 筆詢問

每次詢問區間 $[a, b]$ 的總和

$1 \leq N, Q \leq 2 \cdot 10^5$

沒有 judge 的題

例題

給你長度為 N 的陣列 A

Q 筆詢問

每次詢問區間 $\max(\max_{1\le i<L}(A_i), \max_{R<j\le N}(A_j))$

Q&A

一大堆题目

一大堆題目

TOJ 816 模糊的地圖

習題 (TOJ 816 模糊的地圖)

N 個點，Q 筆詢問

三種操作

1. 連接 a b 二點
2. 詢問 a b 是否連通
3. 問圖上總共有多少個連通塊

$1 \leq N, Q \leq 10^5$

TOJ 27 遙控器 2

習題 (TOJ 27 遙控器 2)

有 N 個按鍵

Q 筆操作，兩種操作

1. 把第 A 個按鍵改成 B
2. 假設 x 是 $[A, B]$ 的最大值，輸出 $\lfloor \frac{x}{2} \rfloor$

$1 \leq N, Q \leq 2 \cdot 10^6$

TOJ 1026 最短排序

習題 (TOJ 1026 最短排序)

給你一個長度為 N 的陣列 A

求你至多不用排序多少個前綴長度

使得後面的元素排序後

整體逆序數對數量 $\leq X$

$1 \leq N \leq 2 \cdot 10^6$

CSES Hotel Queries

習題 (CSES Hotel Queries)

有 N 間旅館，還有 M 組遊客

第 i 間旅館能塞的下 h_i 個人

第 j 組遊客的人數為 r_j

根據先後順序，將每組遊客安排至當下能塞的下的房間當中，並輸出會到哪間房間

房間可以塞不同組遊客，假如沒有房間可以住就輸出 0

$1 \leq N, M \leq 2 \cdot 10^5$

習題 (TOJ 33 好多學生)

N 個數字，Q 筆詢問

每次詢問問區間 $[l, r]$ 的**區間眾數**出現的次數

$1 \leq N \leq 8 \times 10^4$

$1 \leq Q \leq 10^5$ **強制在線**

一些酷東西

一些酷東西

資結好多東西喔

廢話

假如你上面的東西都學完了

假如你上面的東西都學完了

那你可以學

- 懶標線段樹
- 動態開點線段樹
- Trie
- 回滾 DSU
- 帶權 DSU
- 李超線段樹
- 持久化線段樹
- Treap
- 線段樹優化建圖
- 時間線段樹
- 小波樹
- 吉如一線段樹
- Splay Tree
- Link Cut Tree
- 貓樹
- Gomory-Hu tree

假如我真的有講到這邊的話

那我真的很強哈哈

Thanks for Listening