

根號算法

credit by nkhg (t1016d)
modified by yp155136
2019/5/18

Sprout

暖身：優希的麻將牌

- 優希有 N 張麻將牌，每張麻將牌都有它的美好度 A_i
- 請你從這 N 張麻將牌挑一些，使得美好度的總和恰好為 K

- N <= 40
- K、每個 A_i、A_i 的總和都在 [1, 10^18]



優希的麻將牌

- 我會做 2^N
- 2^40 大約是 10^12
- 好像有點大
- 輸光光(?

Sprout



優希的麻將牌

- 我會做 2^N
- 2^40 大約是 10^12
- 好像有點大
- 輸光光(?

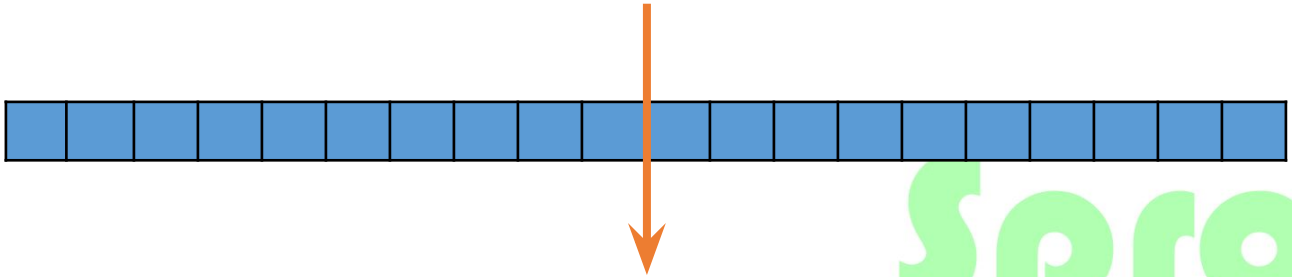
- 突然回講起上個禮拜DP教的內容(???
- 折半枚舉

Sprout

優希的麻將牌

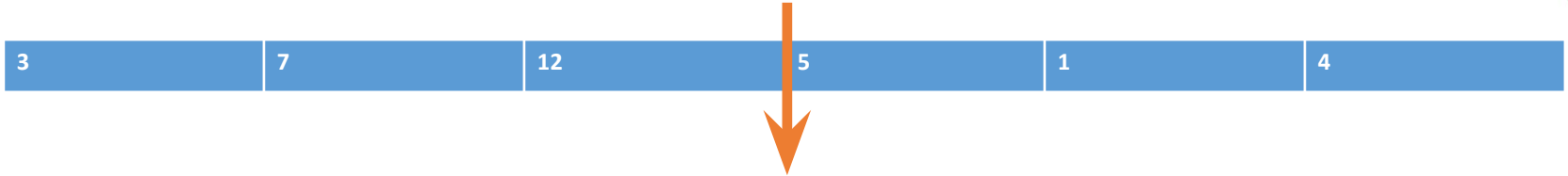
- 如果我們可以把問題分割成「規模相近」的兩部分
- 而且可以用「好方法」把問題的兩個部分合併

- 那就有機會藉此減少時間複雜度！
- 40個物品...任意分割成兩個大小類似的集合



優希的麻將牌

- 左半邊可以組合出什麼組合呢
- {}、{3}、{7}、{12}、{3,7}、{3,12}、{7,12}、{3,7,12}
- 總和分別為 {0,3,7,12,10,15,19,33}



- 那麼，右半邊呢
- {}、{5}、{1}、{4}、{5,1}、{1,4}、{5,4}、{5,1,4}
- 總和分別為 {0,5,1,4,6,5,9,10}

優希的麻將牌

- 如果我們想要湊出總和為 K

- 左半邊總和分別為 {0,3,7,12,10,15,19,33}
- 右半邊總和分別為 {0,5,1,4,6,5,9,10}

- 如果左邊所取的物品，價值總和為 i
- 右邊就必須拿到 M-i



優希的麻將牌

- 分別考慮左邊的所有物品能湊出的價值
- 分別考慮右邊的所有物品能湊出的價值
- 需要多少時間？ 分別需要 O(2^(N/2))

- 我們總共想要價值M
- 枚舉左邊集合的所有價值 i
- 如果在右邊找的到 M-i，那就成功了
- 如果右邊找不到M-i呢？ 那就表示不存在「從左邊集合取出總和 i，右邊集合取出總和 M-i 的方法」

Sprout

優希的麻將牌

- 如何從右邊的集合中尋找 M-i 呢？
- 先排序！ 再搜尋
- 每一個 O(log(2^(N/2)))
- O(2^20 * log2 (2^20)) -> 2 * 10^7

- 雖然不太爽，但至少比原先的 O(2^40) 快多了

Sprout



優希的麻將牌

- 要如何找出一組可行的方案？

```
Struct Mahjong {
    long long sum; //儲存總和
    bool used[N]; //紀錄每個元素是否用到
}
```

Sprout



優希的麻將牌

- Struct Mahjong { //照sum排序
- long long sum; //儲存總和
- int used; //紀錄每個元素是否用到，用一個int就夠了
- } //上面其實也可以用std::pair寫

- 因為搜索的問題本身規模不會太大
- 位元運算是方便好寫的好選擇
- 很快，很節省空間

Sprout

優希的麻將牌

- 實做範例

```
typedef pair<int,int> pii;
int a[N];
int nn = n/2;
vector<pii> v;
for (int i=0;i<(1<<nn);++i) { //i is the bitmask
    long long sum=0; //sum of this bitmask
    for (int j=0;j<nn;++j) {
        if ((1<<j)&i) sum += a[j];
    }
    v.push_back(make_pair(i,sum));
}
```

暖身完畢

- 暖身完畢，進入正題吧～



RMQ(Range minimum query)

- 給你一個長度 N 的序列 A[0], A[2], ..., A[N - 1]
- 緊接著 Q 筆詢問, 每筆詢問是一個數對 x, y
- 請你回答 min{ A[x], A[x + 1], ..., A[y] }
- N <= 10^5, 0 <= A[i] <= 10^9

- A = {5, 2, 8, 1, 7}
- Query = {(0, 2), (1, 4), (2, 2)}
- 答案:2 1 8

Sprout

RMQ(Range minimum query)

```
1 int ans = A[x];
2
3 for (int i = x + 1; i <= y; i++)
4   ans = min(ans, A[i]);
```

- 時間複雜度 O(QN)
- 特性:min(a, b, c) = min(a, min(b, c))
- 可以對一部分數字先求最小值，再從每個部份的最小值中取最小的！

Sprout

RMQ(Range minimum query)

- 一部分？
- 每連續 k 個分一組，先把每組的最小值算好並存下來
- 對於詢問



- 綠色每塊都先算好了，時間花費是塊數：最多 N/k
- 紅色部份每個數字跑一遍，時間花費是一塊大小：最多 2k
- 下面的說明中，綠色會是「塊中元素」，紅色會是「邊邊元素」



RMQ(Range minimum query)

- 每次詢問 O(N/k + k)
  取個好的 k？
- 算幾不等式:N/k + k <= 2*sqrt(N/k * k) = 2*sqrt(N)
- 在 N/k = k 也就是 k = sqrt(N) 時最小

- 所以取 k 為 sqrt(N) 就有了每次詢問 O(sqrt(N)) 的解法了
  - 預處理 O(N)

Sprout

RMQ(Range minimum query)

- 於是大家獲得了人生第一個根號算法 (?)

- RMQ 是很經典的問題
  - 線段樹:預處理 O(N) - 詢問 O(log N)
  - sparse table:預處理 O(N log N) - 詢問 O(1)
  - 線性 RMQ:預處理 O(N) - 詢問 O(1)

- 資芽只會教到線段樹

Sprout

實作上

```
/** theoretically **/
scanf("%d", &N);
K = sqrt(N);

/** When solving OJ problems **/
const int MAXN = 1e5 + 100;
const int K = 330;

/** Testing sample inputs **/
const int MAXN = 1e5 + 100;
//const int K = 330;
const int K = 3;

// remember to set K back to 330 before submit
```

實做技巧

- 使用 0-base
- i/K 代表
  i 在哪一塊
- 如果題目輸入
  是1-base的話,
  記得改成0-base
  (???

```
const int N = 100006;
const int K = 330;  //sqrt(N)
int a[N];
int mn[N]; //mn[i] --> minimum value of block i
memset(mn,(0x3f3f3f3f),sizeof(mn));
for (int i=0;i<n;++i) mn[i/K] = min(mn[i/K],a[i]);
//assume that query are [L,R]
int ans = 0x3f3f3f3f;
for (int i=L;i<=R;++i) {
    if (i/K != L/K) break;
    ans = min(ans,a[i]);
}
for (int i=R;i>=L;--i) {
    if (i/K != R/K) break;
    ans = min(ans,a[i]);
}
for (int i=L/K+1;i<R/K;++i) ans = min(ans,mn[i]);
```

序列操作題

- 給你 A[1], A[2], ..., a[N]
- 兩種操作, 總操作數量為Q
  - 修改:把A[X] 設成 Y
  - 查詢:輸出A[L]到A[R]中, 有多少數字比Z小

- N, Q <= 10^5, 0 <= A[i] <= 10^9

- 想想看可以怎麼做？！？

序列操作題

- 優希(X) 老天(O) 告訴你, 在試試看一次分塊～



序列操作題

- 試著把序列每 K 個分成一塊，每塊都維護成 sorted 的序列
  - 預處理 O(N log N)

序列操作題

- 試著把序列每 K 個分成一塊，每塊都維護成 sorted 的序列
  - 預處理 O(N log N)
- 對於查詢操作：
  - 邊邊的元素：O(K) 暴力比較
  - 塊裡面的元素：O(log N) 二分搜有多少數字比 Z 小
  - 總複雜度：O(K + N/K log N)



序列操作題

- 試著把序列每 K 個分成一塊，每塊都維護成 sorted 的序列
  - 預處理 O(N log N)
- 對於查詢操作：
  - 邊邊的元素：O(K) 暴力比較
  - 塊裡面的元素：O(log N) 二分搜有多少數字比 Z 小
  - 總複雜度：O(K + N/K log N)
- 對於修改操作：
  - O(K) 把該塊的 sorted 序列維護好

Sprout

序列操作題

- 試著把序列每 K 個分成一塊，每塊都維護成 sorted 的序列
  - 預處理 O(N log N)
- 對於查詢操作：
  - 邊邊的元素：O(K) 暴力比較
  - 塊裡面的元素：O(log N) 二分搜有多少數字比 z 小
  - 總複雜度：O(K + N/K log N)
- 對於修改操作：
  - O(K) 把該塊的 sorted 序列維護好
- 總複雜度：O(NK + N^2/K log N)，取 K = sqrt(N) ?!?

Sprout

序列操作題

- 試著把序列每 K 個分成一塊，每塊都維護成 sorted 的序列
  - 預處理 O(N log N)
- 對於查詢操作：
  - 邊邊的元素：O(K) 暴力比較
  - 塊裡面的元素：O(log N) 二分搜有多少數字比 z 小
  - 總複雜度：O(K + N/K log N)
- 對於修改操作：
  - O(K) 把該塊的 sorted 序列維護好
- 總複雜度：O(NK + N^2/K log N)，取 K = sqrt(N) ?!?
- 認真算一下，好像取 K = sqrt(N log N) 會比較好
- O(N sqrt(N) log N) v.s. O(N sqrt(N log N))

序列操作題:簡單實做

- 可以使用 std::vector 的 erase, insert, 會比自己寫swap還快一點點(不過複雜度還是O(N))
- 善用 std::lower_bound

```
vector<int> v[N];
//v[i]: sorted array of block i
//modify operation (x, y)
int i = x;
v[i/K].erase(lower_bound(v[i/K].begin(),v[i/K].end(),a[x]));
a[x] = y;
v[i/K].insert(lower_bound(v[i/K].begin(),v[i/K].end(),a[x]),a[x]);
//query operation (x,y,k)
for (int i=x/K+1;i<y/K;++i) {
    ans += upper_bound(v[i].begin(),v[i].end(),k)-v[i].begin();
}
//elements in block x/K & y/K use brute-force
```

中國人插隊問題

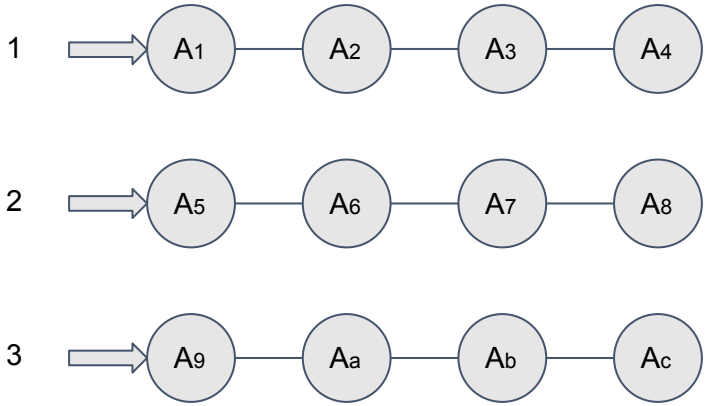
- 一開始給一個長度 N 的序列(index 1 到 N)
- 接著 Q 次操作, 每次可能是
  a. 請在第 i 個位子插入數字 x(index >= i 的數字往後一格)
  b. 拔掉第 i 個位子的數字(index > i 的數字往前一格)
  c. 請你回答第 i 個位子的數字是多少
- 例如
  - 初始:1, 2, 3, 4, 5
  - 操作:(a, 1, 6), (a, 7, 7), (c, 4), (b, 3), (b, 3), (c, 5)
  - 過程:
    - 6, 1, 2, 3, 4, 5
    - 6, 1, 2, 3, 4, 5, 7 (輸出 3)
    - 6, 1, 3, 4, 5, 7
    - 6, 1, 4, 5, 7 (輸出 7)

中國人插隊問題

- 天上傳來一道聲音，叫你再試試看剛剛那樣「分塊」
- 雖然這題用平衡樹炸下去是可以 O(N log N)，不過這裡我們先試試看分塊XD

中國人插隊問題

- 嘗試讓每塊儲存了 K 個數字（若 K = 4）

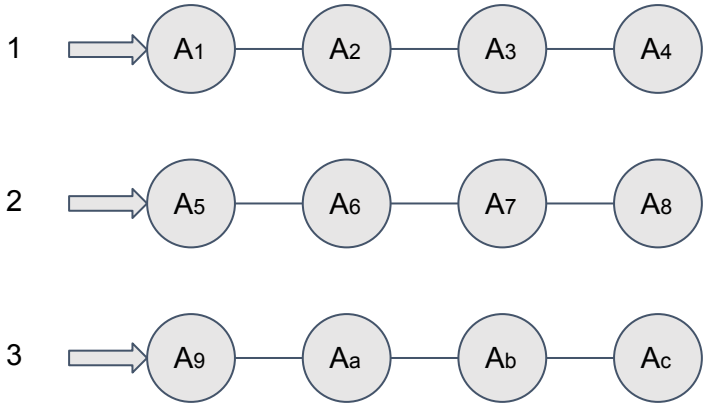


中國人插隊問題

- 一開始給一個長度 N 的序列(index 1 到 N)
- 接著 Q 次操作, 每次可能是
  a. **請在第 i 個位子插入數字 x(index >= i 的數字往後一格)**
  b. 拔掉第 i 個位子的數字(index > i 的數字往前一格)
  c. 請你回答第 i 個位子的數字是多少
- 例如
  - 初始:1, 2, 3, 4, 5
  - 操作:(a, 1, 6), (a, 7, 7), (c, 4), (b, 3), (b, 3), (c, 5)
  - 過程:
    - 6, 1, 2, 3, 4, 5
    - 6, 1, 2, 3, 4, 5, 7 (輸出 3)
    - 6, 1, 3, 4, 5, 7
    - 6, 1, 4, 5, 7 (輸出 7)

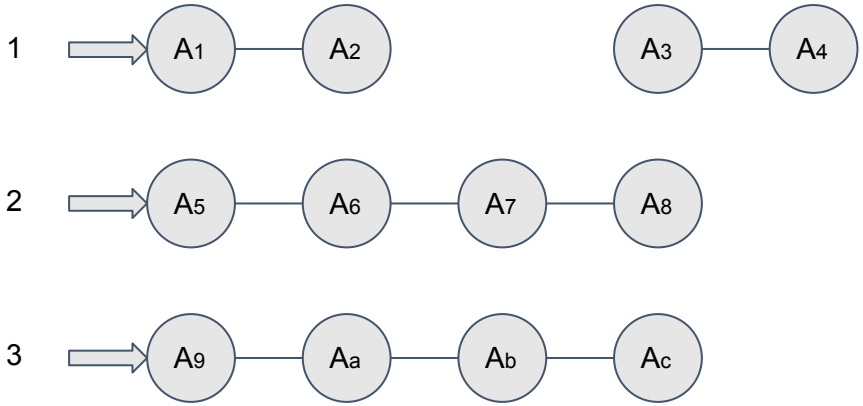
中國人插隊問題

- 增加一個數字(第 3 個位置插入 B)



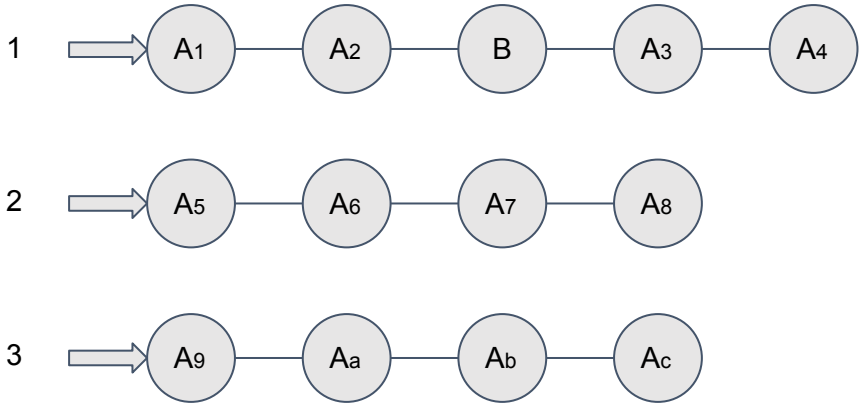
中國人插隊問題

- 增加一個數字(第 3 個位置插入 B)



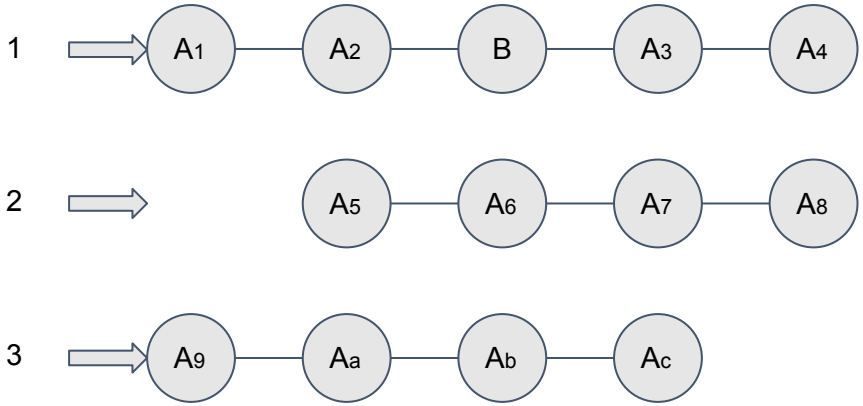
中國人插隊問題

- 增加一個數字(第 3 個位置插入 B)



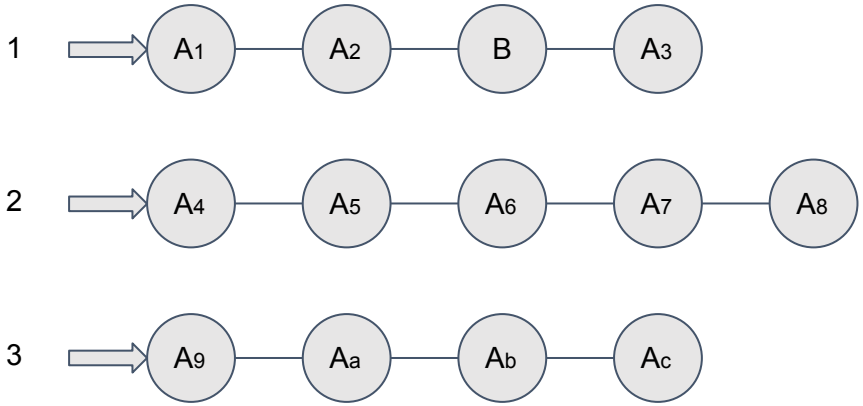
中國人插隊問題

- 增加一個數字(第 3 個位置插入 B)



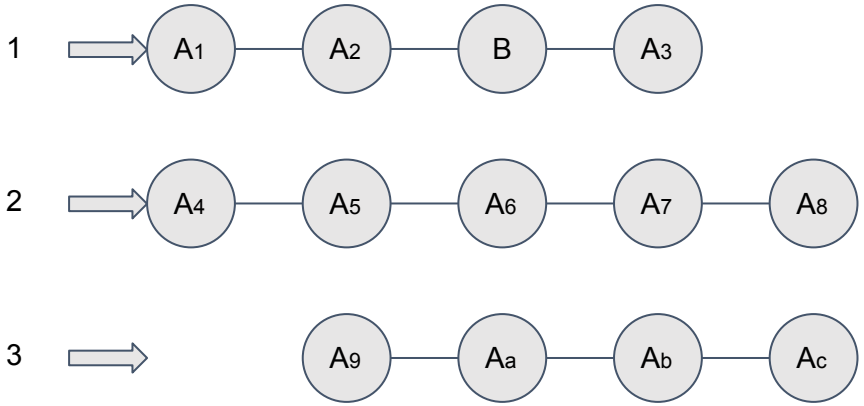
中國人插隊問題

- 增加一個數字(第 3 個位置插入 B)



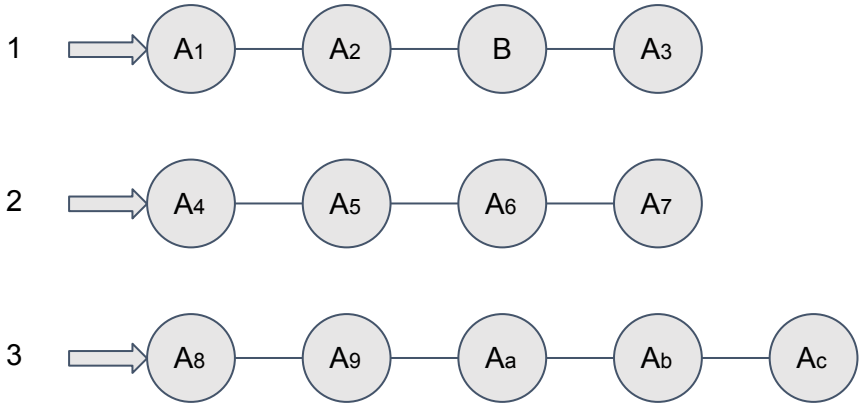
中國人插隊問題

- 增加一個數字(第 3 個位置插入 B)



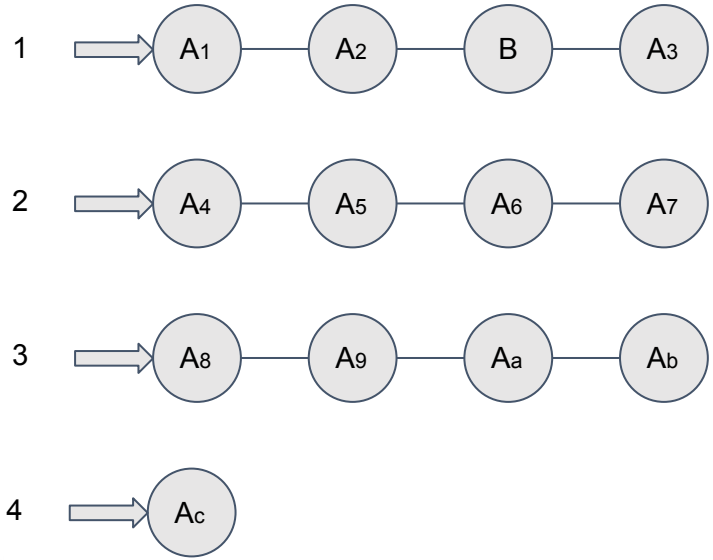
中國人插隊問題

- 增加一個數字(第 3 個位置插入 B)



中國人插隊問題

- 增加一個數字(第 3 個位置插入 B)

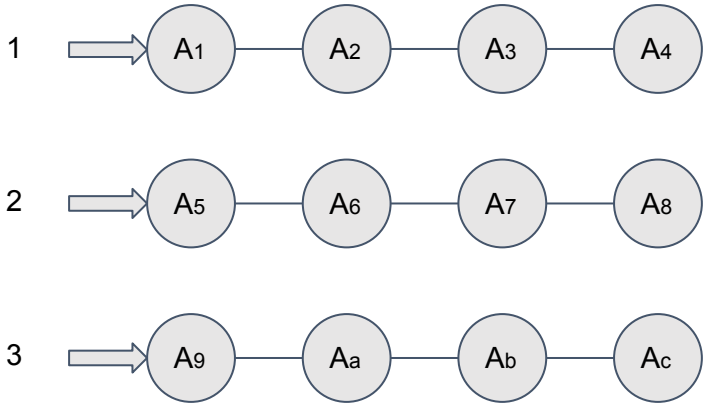


中國人插隊問題

- 一開始給一個長度 N 的序列(index 1 到 N)
- 接著 Q 次操作, 每次可能是
  a. 請在第 i 個位子插入數字 x(index >= i 的數字往後一格)
  b. 拔掉第 i 個位子的數字(index > i 的數字往前一格)
  c. 請你回答第 i 個位子的數字是多少
- 例如
  - 初始:1, 2, 3, 4, 5
  - 操作:(a, 1, 6), (a, 7, 7), (c, 4), (b, 3), (b, 3), (c, 5)
  - 過程:
    - 6, 1, 2, 3, 4, 5
    - 6, 1, 2, 3, 4, 5, 7 (輸出 3)
    - 6, 1, 3, 4, 5, 7
    - 6, 1, 4, 5, 7 (輸出 7)

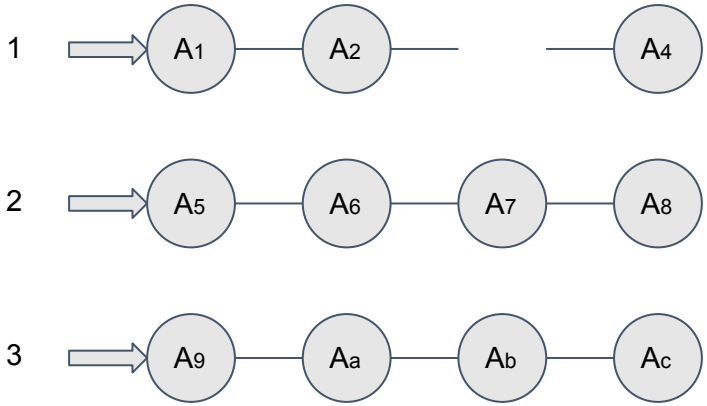
中國人插隊問題

- 拔掉一個數字(拔掉第 3 個位置)



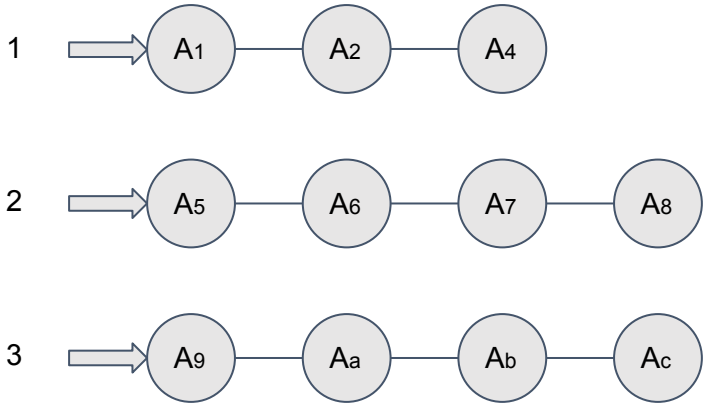
中國人插隊問題

- 拔掉一個數字(拔掉第 3 個位置)



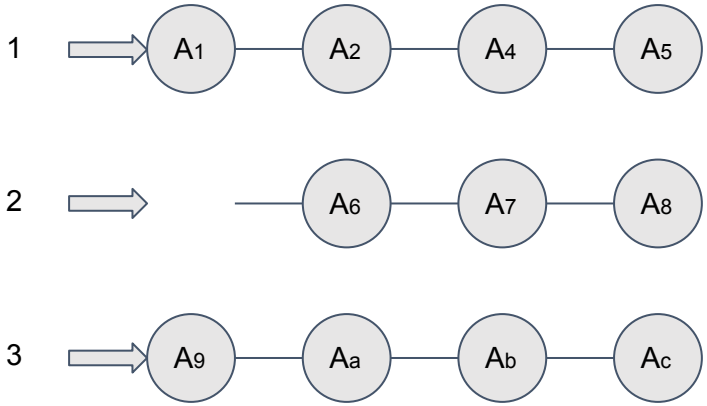
中國人插隊問題

- 拔掉一個數字(拔掉第 3 個位置)



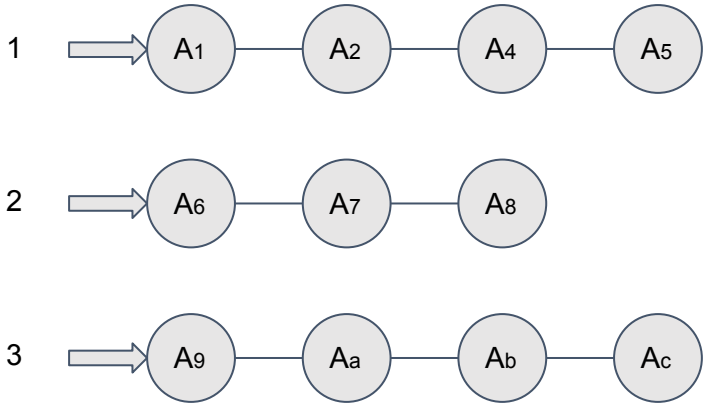
中國人插隊問題

- 拔掉一個數字(拔掉第 3 個位置)



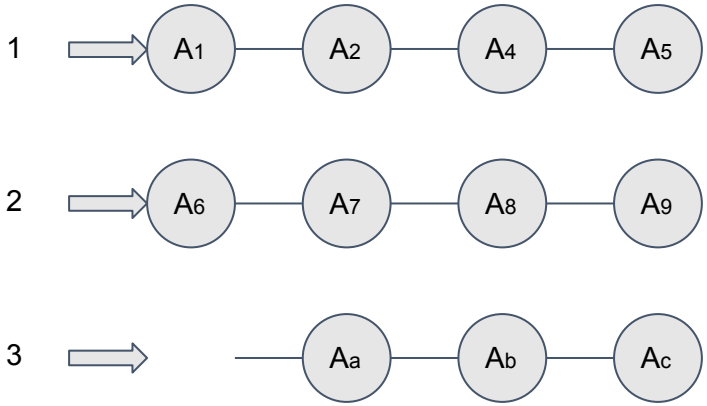
中國人插隊問題

- 拔掉一個數字(拔掉第 3 個位置)



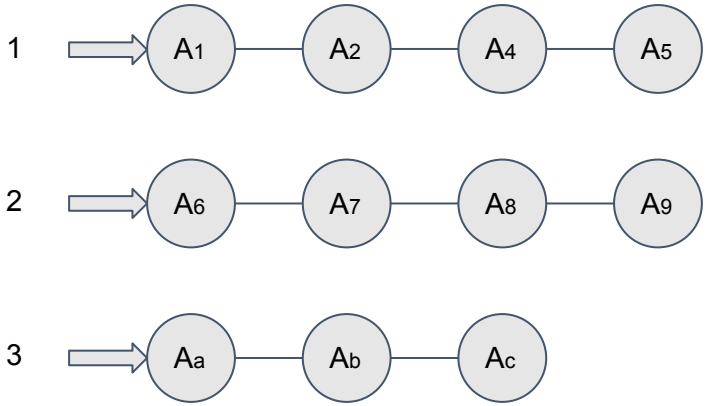
中國人插隊問題

- 拔掉一個數字(拔掉第 3 個位置)



中國人插隊問題

- 拔掉一個數字(拔掉第 3 個位置)

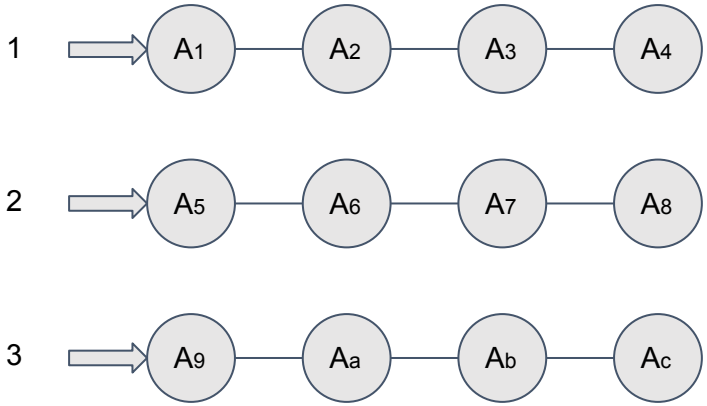


中國人插隊問題

- 一開始給一個長度 N 的序列(index 1 到 N)
- 接著 Q 次操作, 每次可能是
  a. 請在第 i 個位子插入數字 x(index >= i 的數字往後一格)
  b. 拔掉第 i 個位子的數字(index > i 的數字往前一格)
  c. 請你回答第 i 個位子的數字是多少
- 例如
  - 初始:1, 2, 3, 4, 5
  - 操作:(a, 1, 6), (a, 7, 7), (c, 4), (b, 3), (b, 3), (c, 5)
  - 過程:
    - 6, 1, 2, 3, 4, 5
    - 6, 1, 2, 3, 4, 5, 7 (輸出 3)
    - 6, 1, 3, 4, 5, 7
    - 6, 1, 4, 5, 7 (輸出 7)

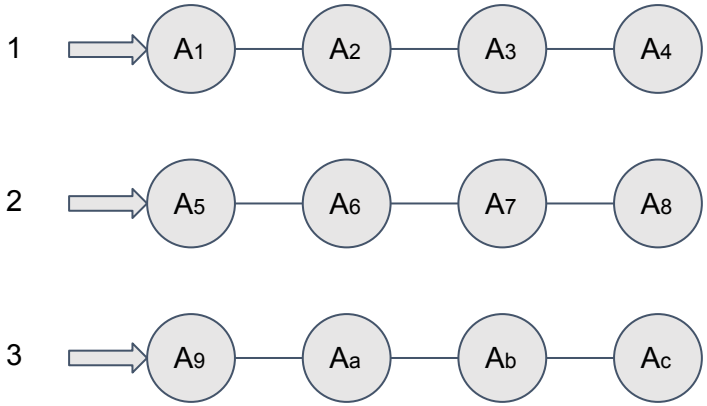
中國人插隊問題

- 第 i 個數字:第 _____ 塊的第 _____ 個



中國人插隊問題

- 第 i 個數字:第 (i - 1) / K + 1 塊的第 (i - 1) % K 個



中國人插隊問題

- 如果有一個可以 O(1) 從頭尾兩端刪除或增加的資料結構來維護每一塊的話
  (內部的查找、刪除都是與儲存的大小成線性)
  - 插入:該塊內部 O(K)、之後每塊都要修一下 O(N / K)
  - 刪除:該塊內部 O(K)、之後每塊都要修一下 O(N / K)
  - 詢問:該塊內部 O(K)
- 找到第 i 塊?
  - 把一堆這種資料結構開成陣列:O(1)
  - 把一堆這種資料結構串起來(linked list):O(N / K)
- 總之全部合起來就是 O(K + N / K) => 又是根號了!

中國人插隊問題

- **如果**有一個可以 O(1) 從頭尾兩端刪除或增加的資料結構來維護每一塊的話
（內部的查找、刪除都是與儲存的大小成線性）

- 方法很多，例如 雙向鏈結串列(Doubly linked list)，或者是STL的std::deque

- 這邊的實做就交給大家囉～可以試試看前面提到用 /K 知道位於哪一塊的作法～

Sprout

第 z 大

- 你有一個容器，一開始是空的
- 題目有三種操作，操作總數量是 Q:
  - 加值:把 y 個數字 x 丟到容器裡面
  - 減值:把 y 個數字 x 從容器中丟掉
  - 查詢:詢問這個容器的第 z 大的元素

- 1 <= Q, x <= 10^5, y <= 10^9 , Z保證合法

Sprout

第 2 大

- 當然，這題可以用平衡樹、線段樹輕鬆過
- 不過老天爺還是希望你可以使用分「塊」

第 2 大

- 當然，這題可以用平衡樹、線段樹輕鬆過
- 不過老天爺還是希望你可以使用分「塊」
- 不過是什麼「塊」呢？

第 z 大

- 對題目的「值域」分塊
- 對這題來講，就是對數字 x 去分塊

第 Z 大

- 對題目的「值域」分塊
- 對這題來講，就是對數字 x 去分塊
- 假設把值域 K 個做一塊，並且維護好每一塊的總和
  - 也就是維護每一塊中，每一個數字出現過得次數和

第 Z 大

- 對題目的「值域」分塊
- 對這題來講，就是對數字 x 去分塊
- 假設把值域 K 個做一塊，並且維護好每一塊的總和
  - 也就是維護每一塊中，每一個數字出現過得次數和
- 加值、減值操作：
  - O(1) 更新那個數字出現的次數和，以及相對應塊的總和

第 Z 大

- 對題目的「值域」分塊
- 對這題來講，就是對數字 x 去分塊
- 假設把值域 K 個做一塊，並且維護好每一塊的總和
  - 也就是維護每一塊中，每一個數字出現過得次數和
- 加值、減值操作：
  - O(1) 更新那個數字出現的次數和，以及相對應塊的總和
- 查詢操作：
  - 先花 O(C/K) 的時間，找到相對應的數字在哪一塊
    - C代表值域，本題來講C = 2*10^5

Sprout

第 Z 大

- 對題目的「值域」分塊
- 對這題來講，就是對數字 x 去分塊
- 假設把值域 K 個做一塊，並且維護好每一塊的總和
  - 也就是維護每一塊中，每一個數字出現過得次數和
- 加值、減值操作：
  - O(1) 更新那個數字出現的次數和，以及相對應塊的總和
- 查詢操作：
  - 先花 O(C/K) 的時間，找到相對應的數字在哪一塊
    - C代表值域，本題來講C = 10^5
  - 再花 O(K) 的時間，確認那個數字是哪一個數字

第 Z 大

- 對題目的「值域」分塊
- 對這題來講，就是對數字 x 去分塊
- 假設把值域 K 個做一塊，並且維護好每一塊的總和
  - 也就是維護每一塊中，每一個數字出現過得次數和
- 加值、減值操作：
  - O(1) 更新那個數字出現的次數和，以及相對應塊的總和
- 查詢操作：
  - 先花 O(C/K) 的時間，找到相對應的數字在哪一塊
    - C代表值域，本題來講C = 10^5
  - 再花 O(K) 的時間，確認那個數字是哪一個數字
- 總複雜度為O(Q(C/K + K))，取 K = sqrt(C)

Sprout

小咲的玩具

- NPSC 2019 國中組初賽 pB
- 你有 K 個 vector<int> v[K], vector 裡面總共會有 N 的元素
- Q 筆詢問, 每筆詢問給兩個 vector<int> 的 index a, b, 請求出

```
long long ans=0;
for (int i:v[a]) for (int j:v[b]) ans += min(i,j)
```

Sprout

小咲的玩具

- 不是序列題，要怎麼分塊QAQ



小咲的玩具

- 先不管這個，現在假設詢問的 vector<int> 的 size 分別是 x，y，並且 x < y。
- 有沒有什麼方法可以得到答案（？）

Sprout



小咲的玩具

- 先不管這個，現在假設詢問的 vector<int> 的 size 分別是 x，y，並且 x < y。
- 有沒有什麼方法可以得到答案（？）
- 方法一：O(xy) 暴力枚舉

Sprout



小咲的玩具

- 先不管這個，現在假設詢問的 vector<int> 的 size 分別是 x，y，並且 x < y。
- 有沒有什麼方法可以得到答案（？）
- 方法一：O(xy) 暴力枚舉
- 方法二：O(x + y) ，使用 雙指針(two pointer)

Sprout

小咲的玩具

- 先不管這個，現在假設詢問的 vector<int> 的 size 分別是 x，y，並且 x < y。
- 有沒有什麼方法可以得到答案（？）
- 方法一：O(xy) 暴力枚舉
- 方法二：O(x + y)，使用 雙指針(two pointer)
  方法二不知道的話沒關係^^
- 方法三：O(x log y)
  先維護好前綴和(prefix sum)，之後對於 x 的每個元素，在 y 二分搜出哪個位置比他小

Sprout

小咲的玩具

- 方法三: O(x log y)
  先維護好前綴和(prefix sum), 之後對於 x 的每個元素 z ,
  在 y 二分搜出哪個位置比他小
- 比 z 小的元素, 題目所求就是那些元素的總和
- 比 z 大的元素, 題目所求就是 z * 比他大的元素的個數

```
for (int i:v[x]) {
    int pos = lower_bound(v[y].begin(),v[y].end(),i) - v[y].begin();
    --pos;
    if (pos < 0) ans += i*1ll*SZ(v[y]);
    else ans += pre[y][pos] + i*1ll*(SZ(v[y])-pos-1);
}
```



小咲的玩具

- 把那先 vector<int> 分成兩類
  - 胖 vector : size > K
  - 瘦 vector : size <= K

Sprout



小咲的玩具

- 把那先 vector<int> 分成兩類
  - 胖 vector : size > K
  - 瘦 vector : size <= K
- 先預處理詢問是兩個胖 vector 的所有答案





小咲的玩具

- 把那先 vector<int> 分成兩類
  - 胖 vector : size > K
  - 瘦 vector : size <= K
- 先預處理詢問是兩個胖 vector 的所有答案
  - 複雜度為: O((N/K)^2 K log N) = O(N^2/K log N)

Sprout

小咲的玩具

- 把那先 vector<int> 分成兩類
  - 胖 vector : size > K
  - 瘦 vector : size <= K
- 先預處理詢問是兩個胖 vector 的所有答案
  - 複雜度為: O((N/K)^2 K log N) = O(N^2/K log N)
- 對於其他的詢問(胖&瘦 或者 瘦&瘦), 就直接算答案

Sprout

小咲的玩具

- 把那先 vector<int> 分成兩類
  - 胖 vector : size > K
  - 瘦 vector : size <= K
- 先預處理詢問是兩個胖 vector 的所有答案
  - 複雜度為: O((N/K)^2 K log N) = O(N^2/K log N)
- 對於其他的詢問(胖&瘦 或者 瘦&瘦), 就直接算答案
  - 複雜度為: O(N K log N)

Sprout

小咲的玩具

- 把那先 vector<int> 分成兩類
  - 胖 vector : size > K
  - 瘦 vector : size <= K
- 先預處理詢問是兩個胖 vector 的所有答案
  - 複雜度為: O((N/K)^2 K log N) = O(N^2/K log N)
- 對於其他的詢問(胖&瘦 或者 瘦&瘦), 就直接算答案
  - 複雜度為: O(N K log N)
- 總複雜度為: O(N^2/K log N + NK log N)

Sprout

小咲的玩具

- 把那先 vector<int> 分成兩類
  - 胖 vector : size > K
  - 瘦 vector : size <= K
- 先預處理詢問是兩個胖 vector 的所有答案
  - 複雜度為: O((N/K)^2 K log N) = O(N^2/K log N)
- 對於其他的詢問(胖&瘦 或者 瘦&瘦), 就直接算答案
  - 複雜度為: O(N K log N)
- 總複雜度為: O(N^2/K log N + NK log N)
- 取 K = sqrt(N) , 可得複雜度為 O(N^1.5 log N)

Sprout

在線與離線

對於剛剛很多題這種有多個詢問、操作的題目

- 在線 (online):程式/演算法 必須對前一個詢問或操作做出回答，之後才能知道下一個詢問或操作。
  (換句話說就是寫程式讀了一行就要馬上處理它)
  (不用懷疑，一定有辦法作到這件事情XD)

- 離線 (offline):程式/演算法 不須先對前一個詢問或操作做出回答，就能知道下一個詢問或操作
  (換句話說你的程式可以先把全部輸入讀進來再來想辦法)

在線與離線

- 為什麼要區分在線與離線呢？
- 哪一個比較簡單？

在線與離線


Google
娜娜奇
全部
圖片
影片
新聞
地圖
更多
設定
工具
請耐心等候至下一個整點（15:00 UTC+8）。
屆時 Google 會處理所有這段時間的搜尋，
您也會在該時間得到您的搜尋結果。

在線與離線

- 將來有很多離線比較簡單的例子在等著大家 A_A



在線與離線

- 其實不用等到將來，剛剛就有一個小小的例子(?)
- 剛剛的序列操作題，使用分塊，就是使用在線的算法
- 而有很多離線算法，可以在時間複雜度更好情況下揍掉這題
  - EX: CDQ 分治，複雜度 O(N (logN)^2)

Sprout

Counting Triangles

- 給你一個 N 個點、M 條邊的簡單無向圖，請算出這張圖裡有幾個「三角形」？
- 假設點被編號為 0 到 N - 1，三角形定義為一組數對 (x, y, z) 使得 0 <= x < y < z < N 且 (x, y)、(y, z)、(z, x) 均有邊

- 3 <= N <= 10^5
- 0 <= M <= 10^5

Sprout

Counting Triangles

- 假設我們能夠 O(1) 回答某個給定的 pair (x, y) 是否為圖上的一條邊

- 那麼指定任一個點 v, 我們能夠
  a. 在 O(M) 時間內算出圖中有幾個包含 v 的三角形
  b. 在 O(d^2) 時間內算出圖中有幾個包含 v 的三角形
     (其中 d 為 v 的點度)
- 每個三角形包含恰 3 個點,
  所以對所有點算出「包含它的三角形個數」加總除以 3,
  就是答案了！

Sprout

Counting Triangles

- 假設我們能夠 O(1) 回答某個給定的 pair (x, y) 是否為圖上的一條邊

- 按照點度是否大於 K 分兩條
  - 大於 K 的: 作法 a.（O(M)）
    -
  - 不超過 K 的:作法 b.（O(d^2)）
    -

Counting Triangles

- 假設我們能夠 O(1) 回答某個給定的 pair (x, y) 是否為圖上的一條邊

- 按照點度是否大於 K 分兩條
  - 大於 K 的: 作法 a. (O(M))
    - 這種點不超過 2M / K 個, 總共 O(M^2 / K)
  - 不超過 K 的:作法 b. (O(d^2))
    - 這種點最多 N 個, 每個 d 最大到 K, 總共 O(N * K^2)

Sprout

Counting Triangles

- 假設我們能夠 O(1) 回答某個給定的 pair (x, y) 是否為圖上的一條邊

- 按照點度是否大於 K 分兩條
  - 大於 K 的: 作法 a. (O(M))
    - 這種點不超過 2M / K 個, 總共 O(M^2 / K)
  - 不超過 K 的:作法 b. (O(d^2))
    - 這種點最多 N 個, 每個 d 最大到 K, 總共 O(N * K^2)

$$O(\sum d_i^2) = O(\sum K \times d_i) = O(K \times \sum d_i) = O(K \times M)$$



Counting Triangles

- 假設我們能夠 O(1) 回答某個給定的 pair (x, y) 是否為圖上的一條邊

- 按照點度是否大於 K 分兩條
  - O(M^2 / K + MK)
  - 又是算幾砸下去，取 K = sqrt(M)
  - 得到 O(M sqrt(M))

- 於是這題被完美的解決了(?)

Sprout



Counting Triangles

- **假設**我們能夠 O(1) 回答某個給定的 pair (x, y) 是否為圖上的一條邊
- 那麼指定任一個點 v，我們能夠
  a. 在 O(M) 時間內算出圖中有幾個包含 v 的三角形
     •
  b. 在 O(d^2) 時間內算出圖中有幾個包含 v 的三角形（其中 d 為 v 的點度）
     •

Sprout



Counting Triangles

- **假設**我們能夠 O(1) 回答某個給定的 pair (x, y) 是否為圖上的一條邊
- 那麼指定任一個點 v，我們能夠
  a. 在 O(M) 時間內算出圖中有幾個包含 v 的三角形
     - 開一條 bool 陣列 adj，adj[i] 表示 v 與 i 之間是否有邊
  b. 在 O(d^2) 時間內算出圖中有幾個包含 v 的三角形（其中 d 為 v 的點度）
     -

Sprout

Counting Triangles

- **假設**我們能夠 O(1) 回答某個給定的 pair (x, y) 是否為圖上的一條邊
- 那麼指定任一個點 v，我們能夠
  a. 在 O(M) 時間內算出圖中有幾個包含 v 的三角形
     - 開一條 bool 陣列 adj，adj[i] 表示 v 與 i 之間是否有邊
  b. 在 O(d^2) 時間內算出圖中有幾個包含 v 的三角形（其中 d 為 v 的點度）
     - 我們只在乎這邊給出的所有 (x, y) 詢問總共有幾個為 true
       而且照 a. 的作法，只要一堆連續的詢問有共同端點我們就會做了

Sprout

Counting Triangles

- **假設**我們能夠 O(1) 回答某個給定的 pair (x, y) 是否為圖上的一條邊
- 那麼指定任一個點 v，我們能夠
  a. 在 O(M) 時間內算出圖中有幾個包含 v 的三角形
     - 開一條 bool 陣列 adj，adj[i] 表示 v 與 i 之間是否有邊
  b. 在 O(d^2) 時間內算出圖中有幾個包含 v 的三角形（其中 d 為 v 的點度）
     - 我們只在乎這邊給出的所有 (x, y) 詢問總共有幾個為 true
       而且照 a. 的作法，只要一堆連續的詢問有共同端點我們就會做了

       離線！把全部詢問照端點分類存起來最後再做

Sprout



Counting Triangles

- 開一條 bool 陣列 adj, adj[i] 表示 v 與 i 之間是否有邊
  這條陣列怎麼寫？

Sprout

Counting Triangles

- 開一條 bool 陣列 adj, adj[i] 表示 v 與 i 之間是否有邊
這條陣列怎麼寫？

```
1 std::fill(adj, adj + N, false);
2 for (int u : edge[v])
3   adj[u] = true;
```

Counting Triangles

- 開一條 bool 陣列 adj, adj[i] 表示 v 與 i 之間是否有邊
這條陣列怎麼寫？

```
1 int adj[MAXN] = {};
2 int round = 0;
3 /** do something **/
4 round++;
5 for (int u : edge[v])
6   adj[u] = round;
7
8 if (adj[u] == round) // u is adjacent to v
```

Counting Triangles

- 另一個結論可能比較簡潔但思路似乎沒那麼自然的作法（不過它或許會給大家一些啟示）
  - 對於一條邊 (u, v)，我們可以用 O(max(d_u, d_v)) 算出包含這條邊的三角形個數(d_u, d_v 是 u 和 v 的點度)
    - 只要一開始把每個點的相鄰點排序過
  - 如果 (u, v) 中點度較大的那個點，我們有它的相鄰矩陣，那麼同一件事可以做到 O(min(d_u, d_v))

Sprout



Counting Triangles

- 把點度 >= sqrt(M) 的點叫做「重點」，其他叫「輕點」
  - 重點只有 2*sqrt(M) 個
    - 所以把「重點」對所有點的相鄰矩陣建出來
    - 空間也只要 O(N sqrt(M))

- 對所有邊計算它被幾個三角形包含
  - 輕點-輕點：用 O(max(d_u, d_v)) 的作法，也才 O(sqrt(M))
  - 輕點-重點：用 O(min(d_u, d_v)) 的作法，也才 O(sqrt(M))
  - 重點-重點：用 O(min(d_u, d_v)) 的作法，也才 ...?

Sprout



Counting Triangles

- 對一個重點 u，如果計算某條邊 (u, v) 的 cost O(min(d_u, d_v)) 是 d_u
  - 表示這條邊連接的點 v 具有 d_v > d_u
  - 這種點 v 最多只有 2M / d_u 個
  - 所以以 d_u 為 cost 的邊的花費加總起來也才
    - (2M / d_u) * d_u = 2M = O(M)

- 重點只有 O(sqrt(M)) 個，它們的總花費就是 O(M sqrt(M))
- 三種情況都在 O(M sqrt(M)) 內，解決！

Sprout



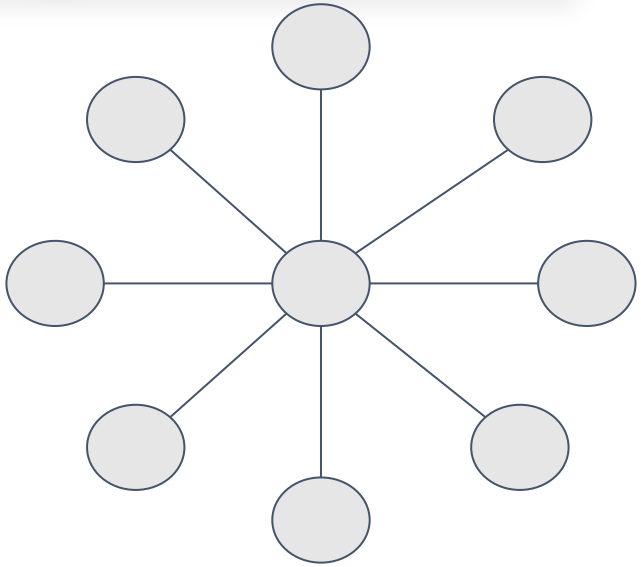
Counting Triangles

- 把點度 >= sqrt(M) 的點叫做「重點」，其他叫「輕點」
  - 重點只有 2*sqrt(M) 個
    - 所以把「重點」對所有點的相鄰矩陣建出來
    - 空間也只要 O(N sqrt(M))

- 對所有邊計算它被幾個三角形包含
  - 輕點-輕點：用 O(max(d_u, d_v)) 的作法，也才 O(sqrt(M))
  - 輕點-重點：用 O(**max**(d_u, d_v)) 的作法，也才 O(sqrt(M))
  - 重點-重點：用 O(min(d_u, d_v)) 的作法，也才 ...?

Sprout

Counting Triangles



Sprout



Counting Triangles

- 把點度 >= sqrt(M) 的點叫做「重點」，其他叫「輕點」
  - 重點只有 2*sqrt(M) 個
    - 所以把「重點」對所有點的相鄰矩陣建出來
    - 空間也只要 O(N sqrt(M))

- 對所有邊計算它被幾個三角形包含
  - 輕點-輕點：用 O(max(d_u, d_v)) 的作法，也才 O(sqrt(M))
  - 輕點-重點：用 O(min(d_u, d_v)) 的作法，也才 O(sqrt(M))
  - 重點-重點：用 O(**max**(d_u, d_v)) 的作法，也才 ...?

Sprout

Counting Triangles



Sprout



Counting Triangles

- 重點整理與啟發
  - 第一種作法枚舉點、第二種作法枚舉邊
  - 找出枚舉後的複雜度關鍵是什麼，對那個東西分大小
- 第一種作法用到的技巧
  - 離線
  - 可重複利用並快速歸零的 bool array
- 第二種作法用到的技巧
  - 「重點」很少，可以維護一個「重點」x「所有點」的二維陣列
- 小心估複雜度

Sprout



Mo’s algorithm

https://zerojudge.tw/ShowProblem?problemid=b417

- 先來看一道經典問題:「區間 種樹(X) 眾數(O)」
- 給你長度為 N 的序列，並且有 Q 筆詢問，每筆詢問的內容是詢問 [L, R] 出現最多次數的數字出現的個數。

- N, Q <= 10^5

Sprout

Mo’s algorithm

- 發明者：中國隊隊長莫濤
- 莫濤在解決「小Z的襪子」這題是，想到的演算法 https://www.lydsy.com/JudgeOnline/problem.php?id=2038
- 於是乎，這個算法就被稱為「莫隊算法」，簡稱「莫隊」、「Mo's algo」



Mo’s algorithm

- 莫隊算法(Mo's algorithm)是一個離線的算法
- 先把詢問「按照某種順序」排序過後，開始暴力硬算

- 題目的詢問從 [L, R] 轉移到 [L, R+1 or R-1] 時，必須在一個很短的時間內轉移。Ex: O(1) or O(log N)

- 題目的詢問從 [L, R] 轉移到 [L+1 or L-1, R] 時，必須在一個很短的時間內轉移。Ex: O(1) or O(log N)

Sprout

Mo’s algorithm

- 假設上面那張投影片的所有轉移時間都是 O(1)
- 那在「經過某種順序排序」後的詢問之下，時間複雜度就是 O(L指針移動的次數) + O(R指針移動的次數) + O(Q * 得到答案的複雜度)



Mo’s algorithm

- 假設上面那張投影片的所有轉移時間都是 O(1)
- 那在「經過某種順序排序」後的詢問之下，時間複雜度就是 O(L指針移動的次數) + O(R指針移動的次數) + O(Q * 得到答案的複雜度)
- 莫隊算法就是擬定一個好的排序方法，使得 O(L指針移動的次數) = O(R指針移動的次數) = O(N sqrt(N))

Sprout

Mo’s algorithm

- 先把序列每 K 個分成一塊。



Mo’s algorithm

- 先把序列每 K 個分成一塊。
- 排序詢問的方法為:
  - 先按照 L 所在的塊排序
  - 如果 L 所在的塊相同, 則按照 R 排序

Sprout

Mo’s algorithm

- 先把序列每 K 個分成一塊。
- 排序詢問的方法為:
  - 先按照 L 所在的塊排序
  - 如果 L 所在的塊相同, 則按照 R 排序
- 認真分析一下複雜度:
  - 對於 L 指針:
  - 對於 R 指針:



Mo’s algorithm

- 先把序列每 K 個分成一塊。
- 排序詢問的方法為:
  - 先按照 L 所在的塊排序
  - 如果 L 所在的塊相同, 則按照 R 排序
- 認真分析一下複雜度:
  - 對於 L 指針:
    O(N * K + N)
  - 對於 R 指針:
    O(N/K * N + N)

Sprout

Mo’s algorithm

- 先把序列每 K 個分成一塊。
- 排序詢問的方法為:
  - 先按照 L 所在的塊排序
  - 如果 L 所在的塊相同, 則按照 R 排序
- 認真分析一下複雜度:
  - 對於 L 指針:
    O(N * K + N)
  - 對於 R 指針:
    O(N/K * N)
- 取 K = sqrt(N) , 大勝利～
- 於是乎, 大家就得到一個 O(N sqrt (N))的區間眾數的解法

Mo’s algorithm

- 參考實做方式:詢問的排序

```
struct Query {
  int L, R, block, i;
  bool operator<(const Query& q) const {
    return block == q.block ? R < q.R : block < q.block;
  }
} query[maxm];
```

Mo’s algorithm

- 參考實做方式:指針的移動方式

```
for (int i = 0, L = 1, R = 0; i < M; ++i) {
  while (R < query[i].R) add(++R);
  while (R > query[i].R) sub(R--);
  while (L > query[i].L) add(--L);
  while (L < query[i].L) sub(L++);
  ans[query[i].i] = make_pair(most, freq[most]);
}
```

Mo’s algorithm

- 參考實做方式:指針移動的加值/減值

```
void add(int bound) {
  freq[cnt[S[bound]]]--;
  cnt[S[bound]]++;
  freq[cnt[S[bound]]]++;
  while (freq[most + 1]) most++;
}

void sub(int bound) {
  freq[cnt[S[bound]]]--;
  cnt[S[bound]]--;
  freq[cnt[S[bound]]]++;
  while (!freq[most]) most--;
}
```

Sprout



Mo’s algorithm

- 進階版:帶修改莫隊
- 複雜度是 O(N^(5/3))
- 就留給大家上網查資料囉

Sprout



Mo’s algorithm

- 許多序列操作題，如果一直在跟 O(N log N), O(N (log N)^2) 奮鬥的話
- 不如可以考慮寫寫莫隊，搞不好有更優秀的解答～

Sprout



題目

https://codeforces.com/contest/785/problem/E

- 給你一個長度為 N 的序列，一開始是 A[1] = 1, A[2] = 2, ..., a[N] = N
- Q 筆操作，每筆操作會給 X, Y ，請執行 swap(A[X],A[Y])
- 每筆操作完後，請輸出當前有多少對「逆序數對」

- N <= 2 * 10^5, Q <= 5 * 10^4
- TL = 4sec, ML = 512MB
- 當然這題有很多其他的方法、資料結構可以揍掉這題

Sprout



題目

https://oj.icpc.tw/c/28/D

- 給你一個 N 個點、M 條邊的簡單無向圖
- Q 筆詢問，每筆詢問輸入一個環，請計算這個環有幾個「弦」
- 「弦」的定義：一條邊包含兩個環上的點，但是這條邊沒有在環上

- N, M, 每個詢問的環大小總和 <= 3 * 10^5

Sprout



題目

區間第 K 大

- 給你一個長度為 N 的序列，並且有 Q 筆詢問
- 每筆詢問的內容是：[L, R]之中，第 K 大的數字是多少
- N, Q, 值域 <= 10^5

- 可能要把上面教的兩個算法合併在一起 (???

Sprout



題目

SRM 675 Div.1 Middle

- 給你 n, x0, a, b, 表示有一個長度 n 的數列:
  - x[0] = x0
  - for (int i = 1; i < n; i++)
    - x[i] = (x[i - 1] * a + b) % (10^9 + 7)
- 你要回答 Q 個詢問, 每個詢問給你一個 k (0 <= k < n), 請你回答數列 x 中第 k 小的數字(k=0 為最小)
- n <= 5 * 10^6
- Q <= 100
- 時限 10sec、額外儲存空間限制 1MB

Sprout