實做能力加強

temmie

1. 位元運算
2. 字元轉換
3. 陣列運用
4. 區間問題

位元運算

為什麼要學這個？

- 因為電腦的機制，位元運算會比一般運算更快一些
- 在**枚舉**這堂課上，我們會用到很多

進位制

- n 進位代表只用 n 以內的數字組成（如果超過 10 就用英文）
- 二進位的數字代表只用 0 和 1 組成

進位制

- n 進位代表只用 n 以內的數字組成（如果超過 10 就用英文）
- 二進位的數字代表只用 0 和 1 組成
- 如果一個式子有不同進位制的數字，則會用括弧備註在右下角
- 例如：$17_{(10)} = 10001_{(2)}$

二進位轉十進位

- 以 $10001_{(2)}$ 舉例
- 我們可以先把所有 2 的冪次列出來：2^4、2^3、2^2、2^1、2^0

二進位轉十進位

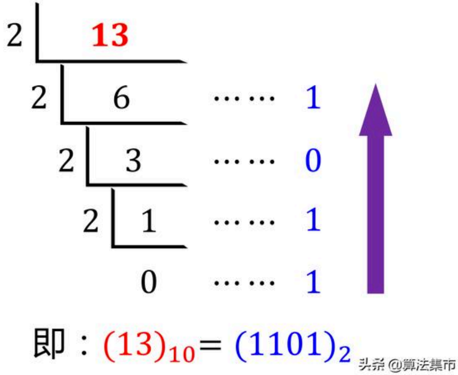
- 以 $10001_{(2)}$ 舉例
- 我們可以先把所有 2 的冪次列出來：2^4、2^3、2^2、2^1、2^0
- 和原本的二進位字串一一對應後相乘 1×2^4、0×2^3、0×2^2、0×2^1、1×2^0

二進位轉十進位

- 以 $10001_{(2)}$ 舉例
- 我們可以先把所有 2 的冪次列出來：2^4、2^3、2^2、2^1、2^0
- 和原本的二進位字串一一對應後相乘 1×2^4、0×2^3、0×2^2、0×2^1、1×2^0
- 將所有數字相加 $1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 17$

十進位轉二進位

- 詳細作法如下圖



-

例題

二進位制轉換

題目連結

給你一個十進位的正整數 n，輸出 n 的二進位

位元運算的類別

- 主要有 OR、AND、XOR、NOT
- 和一般的 or、and、not 不同的是，可以對數字做運算，而不是限制在布林

位元運算的類別

- 主要有 OR、AND、XOR、NOT
- 和一般的 or、and、not 不同的是，可以對數字做運算，而不是限制在布林

- *AND*：「&」表示，如果兩個 bit **都是** 1 就是 1，否則為 0
- *OR*：「|」表示，如果兩個 bit **至少有一個** 1 就是 1，否則為 0
- *XOR*：「^」表示，如果兩個 bit **恰有一個為** 1，否則為 0
- *NOT*：「~」表示，將 1 變成 0，0 變成 1

使用位元運算

- 難道用位元運算還要手動轉進位制？@@

使用位元運算

- 難道用位元運算還要手動轉進位制？@@
- 在 C++ 中，直接對兩個十進位的數字做就可以了
- $17_{(10)} \mid 5_{(10)} = 10001_{(2)} \mid 00101_{(2)} = 10101_{(2)} = 21_{(10)}$

例題

位元運算的大雜燴

題目連結

給你三個參數 x、y、z ，找到 $(x \oplus (y \mid z)) \oplus (x + y)$ 的最小值

其中 $\oplus$ 是 XOR 的意思，$\mid$ 是 OR 的意思

例題

位元運算的大雜燴

題目連結

給你三個參數 x、y、z ，找到 $(x \oplus (y \mid z)) \oplus (x + y)$ 的最小值

其中 $\oplus$ 是 XOR 的意思，$\mid$ 是 OR 的意思

- 參數數量是固定的，那就列出所有情況吧！
- 把六種組合各試一遍就可以了

更多位元運算

- 另外常用的位元運算有左移和右移

更多位元運算

- 另外常用的位元運算有左移和右移

- 左移：「＜＜」表示，代表將所有 bit 左移 n 位
- 右移：「＞＞」表示，代表將所有 bit 右移 n 位

更多位元運算

- 另外常用的位元運算有左移和右移

- 左移：「<<」表示，代表將所有 bit 左移 n 位
- 右移：「>>」表示，代表將所有 bit 右移 n 位

- 例如：$17_{(10)} << 1 = 10001_{(2)} << 1 = 100010_{(2)} = 34_{(10)}$
- 你知道嗎？實際上左移就是乘上 2^n（右移則相反）

字元轉換

Ascii

- 我們都知道電腦是儲存 0 跟 1，並不能儲存字元
- 那要怎麼儲存字元呢？

Ascii

- 我們都知道電腦是儲存 0 跟 1，並不能儲存字元
- 那要怎麼儲存字元呢？
- 只要把所有字母編碼就可以啦！
- 實際上我們用的字元都有一個編碼，稱為 Ascii 編碼

Ascii

- $[\,0-9\,]=48\sim57$
- $[\,A-Z\,]=65\sim90$
- $[\,a-z\,]=97\sim122$
- 利用編碼的連續性，就有很多功能可以應用

例題

數字總和

題目連結

你有一個數字 n，請你求出所有位數的總和，保證 n 的位數 $\leq 10^5$

例題

數字總和

題目連結

你有一個數字 n，請你求出所有位數的總和，保證 n 的位數 $\leq 10^5$

- 字串很大，看起來沒辦法用 int 儲存，所以只能用 string 儲存
- 我們可以用 for 得到每個字元，並且**減去'0'**，就可以轉換成數字
- '0'(48)-'0'(48)=0，'1'(49)-'0'(48)=1
- '5'(53)-'0'(48)=5，'9'(57)-'0'(48)=9

例題

凱撒密碼

題目連結

給你一個字串 s，把每個字母向後移 3 位，例如 $\mathsf{A} \to \mathsf{D}$，$\mathsf{Z} \to \mathsf{C}$

例題

凱撒密碼

題目連結

給你一個字串 s，把每個字母向後移 3 位，例如 A $\rightarrow$ D，Z $\rightarrow$ C

- 我們可以將每個字母的值都加上 3
- 如果該字母超過'Z' 的話就減去 26

陣列運用

陣列運用

- 陣列的用途很廣，巧妙的使用陣列可以讓時間複雜度更好
- 用陣列也可以實做很多好用的資料結構，並提供額外的性質

vector

- vector 可以說是更好的陣列，**可以動態的分配空間**
- 陣列做的事 vector 做的到，陣列不能做的事 vector 也可以做的到
- 缺點：可能比陣列慢一點點、大量宣告可能會吃到 MLE

vector 的宣告和使用

vector<type> name(size, value);

- type 是放 vector 裡面東西的型別
- name 則是 vector 的名稱（接下來的介紹都用 vec 表示）
- （可選）size 可以放初始的陣列大小
- （可選）value 可以放初始值

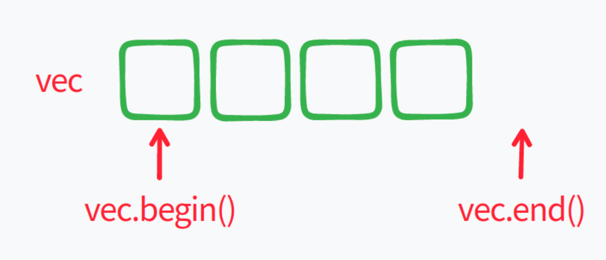
name.method(value1, value2...)

- 以上為 vector 如何使用的模板

vector 的資料位置

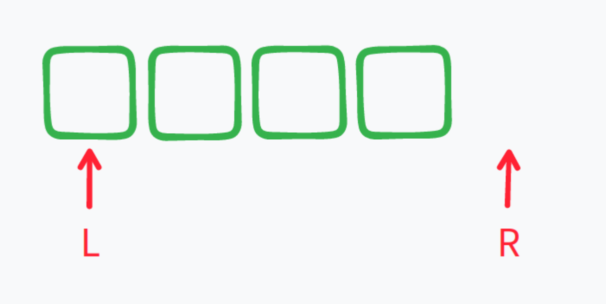
- .begin() 代表 vector 裡第一個資料位置
- .end() 代表 vector 裡最後一個資料**的後一位**位置
- vec[i] 代表 vector 裡的第 i 個資料

vector 的資料位置

- .begin() 代表 vector 裡第一個資料位置
- .end() 代表 vector 裡最後一個資料**的後一位**位置
- vec[i] 代表 vector 裡的第 i 個資料
- 

- 要對整個 vector 操作丟.begin() 跟.end() 就好了

vector 的使用

- 以後介紹的函式，除非有特別標注，否則都為以下的方法
- 開始都是包含，結束則不包含
- 書寫的形式為 [L, R)
- 

vector 的輸入方式

- 分配好空間，並且用 cin >> [i]
- 不分配空間，用 vec.push_back(value) 將元素放入最後一位

- 要用哪一個？如果後續沒有要操作就用第一個，否則用另一個

vector 的各種常用操作

- $O(1)$，.size()，取得 vector 的大小
- $O(1)$，.empty()，回傳 vector 是否為空
- $O(\mid size - n \mid)$，.resize(n, [val])
  重設 vector 的大小，並設為 val（可選）
- $O(1)$，.pop_back()，清除 vector 最後面的元素
- $O(1)$，.clear()，清空 vector
- $O(1)$，.front()，回傳最前面的元素
- $O(1)$，.back()，回傳最後面的元素

vector 的各種常用操作

- $O(n)$，*max_element(L, R)，取得 [L, R) 的最大值
- $O(n)$，*min_element(L, R)，取得 [L, R) 的最小值
- $O(1)$，swap(vec1, vec2)，將 vec1 跟 vec2 對調
- $O(n)$，fill(L, R, val)，將 [L, R) 設為 val
- $O(n \log n)$，sort(L, R)，將 [L, R) 排序

- 內容很多，以知道有這些功能為主，多練習題目自然就會記起來了
- 特別注意，有些 $O(n)$ 不要用太多次

vector 的遍歷

- ```cpp
  for (auto x : v){
      cout << x << " ";
  }
  ```
- 其中 x 就會是 vector 裡的所有元素
- 想要改值？回想前一節講的吧！
  ```

例題

圖書館

題目連結

給你 n 本書，每本書都有編號 a_i 和借閱日期 b_i

如果借閱日期超過 100 天就代表逾期，每超過 1 天就要罰 5 元罰金

請求出所有逾期的書的編號，以及總罰金

例題

圖書館

題目連結

給你 n 本書，每本書都有編號 a_i 和借閱日期 b_i

如果借閱日期超過 100 天就代表逾期，每超過 1 天就要罰 5 元罰金

請求出所有逾期的書的編號，以及總罰金

- 我們可以把所有逾期的書都裝進 vector 裡，是不是簡單又方便呢？

二維 vector 的宣告和使用

vector<vector<type>>name(size1, vector<int>(size2, value))

- 以上如同 int[ ][ ]
- 不過實在是太長了，通常我還是會用 int[ ][ ]

vector<type> name[size]

- 這也是二維陣列，不過每一項都可以是不同大小
- 在**圖論**這個單元我們會很常用到

該學哪個？一如往常的，兩個都要學

二維 vector 的比較

例題

vector 練習

題目連結

給你 n 個人，並且有 k 組朋友關係，由小到大輸出每個人的朋友有哪些

每組朋友關係有兩個整數 a、b，代表 a 和 b **互相為**朋友

(記得 IO 加速)

例題

vector 練習

題目連結

給你 n 個人，並且有 k 組朋友關係，由小到大輸出每個人的朋友有哪些
每組朋友關係有兩個整數 a、b，代表 a 和 b **互相為**朋友
(記得 IO 加速)

- 這題其實就是圖的儲存！我們留到圖論在說

區間問題

區間問題概述

- 在競程裡面，我們有相當多的問題是有關於「區間」的
- 通常會有兩個功能「修改」跟「查詢」
- 這次我們會介紹兩個非常常見且基礎的方法

例題

接下來我們用幾個問題來暖身

- $1+3+2+1+3+2+1+3+3+2+1+2+3$

例題

接下來我們用幾個問題來暖身

- $1+3+2+1+3+2+1+3+3+2+1+2+3$
- $1+3+2+1+3+2+1+3+3+2+1+2+3+5$

例題

接下來我們用幾個問題來暖身

- $1+3+2+1+3+2+1+3+3+2+1+2+3$
- $1+3+2+1+3+2+1+3+3+2+1+2+3+5$
- $1+3+2+1+3+2+1+3+3+2+1+2+3+5+3$

例題

接下來我們用幾個問題來暖身

- 1+3+2+1+3+2+1+3+3+2+1+2+3
- 1+3+2+1+3+2+1+3+3+2+1+2+3+5
- 1+3+2+1+3+2+1+3+3+2+1+2+3+5+3

- 4+1+3+1+3+3+4+3+1+2+1+2+1

例題

接下來我們用幾個問題來暖身

- $1+3+2+1+3+2+1+3+3+2+1+2+3$
- $1+3+2+1+3+2+1+3+3+2+1+2+3+5$
- $1+3+2+1+3+2+1+3+3+2+1+2+3+5+3$

- $4+1+3+1+3+3+4+3+1+2+1+2+1$
- $4+1+3+1+3+3+4+3+1+2+1+2+1$

例題

接下來我們用幾個問題來暖身

- 1+3+2+1+3+2+1+3+3+2+1+2+3
- 1+3+2+1+3+2+1+3+3+2+1+2+3+5
- 1+3+2+1+3+2+1+3+3+2+1+2+3+5+3

- 4+1+3+1+3+3+4+3+1+2+1+2+1
- 4+1+3+1+3+3+4+3+1+2+1+2+1

- 如果一個一個計算是不是很慢呢？讓我們透過一些技巧加速

前綴和概念

- 前綴和可以處理**沒有修改，區間查詢**的問題
- 其中每個查詢都可以在 $O(1)$ 完成
- 非常非常非常重要的技巧，很常出現

前綴和概念

- 
- 上圖的藍色區域怎麼算呢？

前綴和概念

- 
- 上圖的藍色區域怎麼算呢？
- 只要算出正方形的面積，並且減去扇形，即可求解
- 讓我們看看這個概念放到陣列上會變成怎樣

前綴和概念

- 

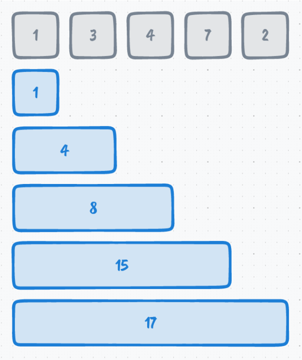
- 我們可以算出所有數字的總和，然後扣除灰色的部份

前綴和概念

- 

- 我們可以算出所有數字的總和，然後扣除灰色的部份
- 嗯？這樣應該算的數字更多，因此更慢不是嗎？
- 實際上，我們可以儲存這些數字，要用時再扣除，這樣就很快啦！

前綴和概念

- 

- 試著把上面的藍色數值兩兩相減看看，然後觀察他們的性質吧
- 如果我想要獲得一個區間的和，該怎麼求得呢？
- 所有藍色的值需要一個一個計算嗎？還是有更快的方法？

前綴和概念

- 稍微整理一下，要找到一個區間 [L, R] 的和
- 則將 [0, R] - [0, L-1] 即可
- 而 [0, n] 的值可以從 [0, n-1] $+ a_n$ 得到

前綴和實做

```
// declare
int n, tmp;
vector<int> v(1);   // 預先開一個空間，防止 [0, L-1] 變成負數
vector<int> pre(1); // 預先開一個空間，防止 [0, n-1] 變成負數

// input
cin >> n;
for (int i=1 ; i<=n ; i++){ // start from 1
    cin >> tmp;
    v.push_back(tmp);

    // get prefix sum
    pre.push_back(pre[i-1]+v[i]);
}
```

- 上面這個過程叫做預處理，可以看出時間複雜度為 $O(n)$
- 我們定義 pre_i 為 $\sum_{j=0}^{i} a_j$

例題

前綴和陣列實作

題目連結

如題，試著實作出我們剛剛介紹的前綴和陣列

前綴和實做

-
```cpp
// queries
cin >> q;
for (int i=0 ; i<q ; i++){
    // 尋找 [l, r] 的區間和
    cin >> l >> q;
    cout << pre[r]-pre[l-1] << endl;
}
```
- 每次查詢都直接拿陣列裡面的值，時間複雜度為 $O(q)$

例題

一維區間和問題

題目連結

給你一個有 n 個正整數的陣列 arr

給予 q 個查詢，每個查詢都有兩個正整數 L_i，R_i

對於每個查詢求出 [L_i, R_i] 的總和

保證 $1 \leq n, q \leq 2 \times 10^5$

例題

二維區間和問題

題目連結

給你一個 $n \times n$ 大小的森林，森林的每一個座標都是空地或是樹

給予 q 個查詢，每個查詢包含 y_{1_i}、x_{1_i}、y_{2_i}、x_{2_i} 對於每個查詢輸出該範圍有多少顆樹

- 請你想想看，如何運用同樣的概念做二維的區間查詢呢？

例題

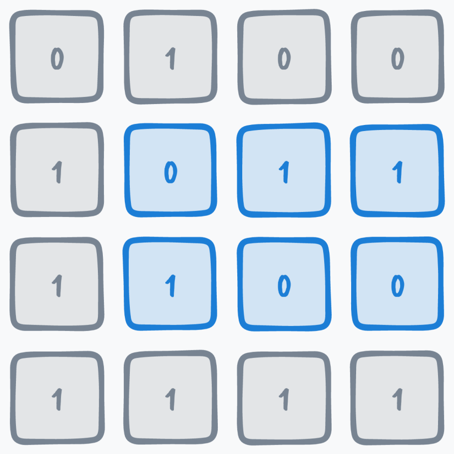
二維區間和問題

題目連結

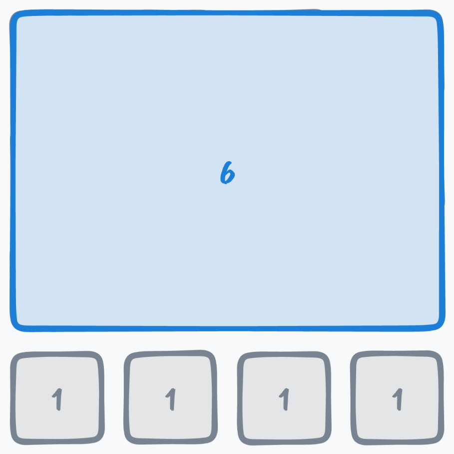
給你一個 $n \times n$ 大小的森林，森林的每一個座標都是空地或是樹
給予 q 個查詢，每個查詢包含 y_{1_i}、x_{1_i}、y_{2_i}、x_{2_i} 對於每個查詢輸出該範圍有多少顆樹

- 請你想想看，如何運用同樣的概念做二維的區間查詢呢？
- 我們一樣儲存 $(0,0)$ 到 (x,y) 的區間和
- 並用相同的**扣除不需要的範圍**的概念

例題

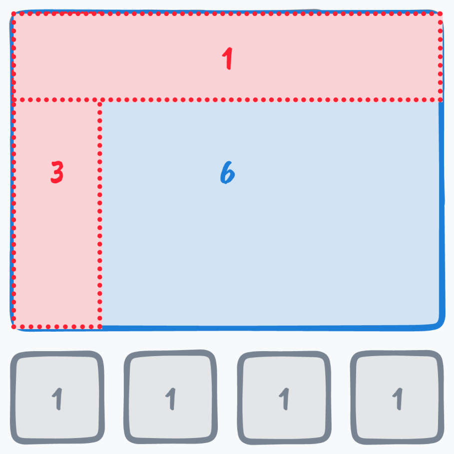
- 

例題

- 

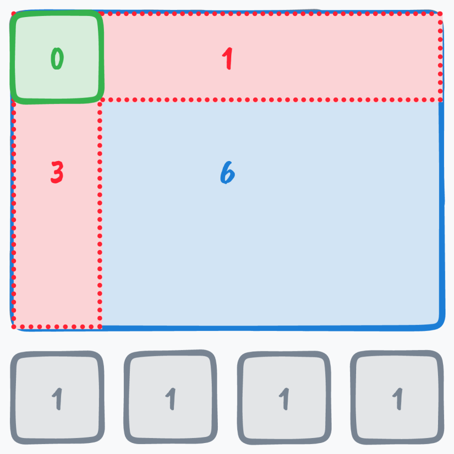
- 先把最大的問題加起來

例題

- 

- 將不需要的範圍刪除

例題

- 

- 把多刪除的範圍加回去

例題

區間 xor 問題

題目連結

給你一個有 n 個正整數的陣列 arr

給予 q 個查詢，每個查詢都有兩個正整數 L_i，R_i

對於每個查詢求出 [L_i, R_i] 所有值做 xor 後的值

保證 $1 \leq n, q \leq 2 \times 10^5$

- 仍然是使用相同的概念，不過 xor 的值要怎麼消除呢？
- tip: $A \oplus A = 0$

例題

區間 xor 問題

題目連結

給你一個有 n 個正整數的陣列 arr

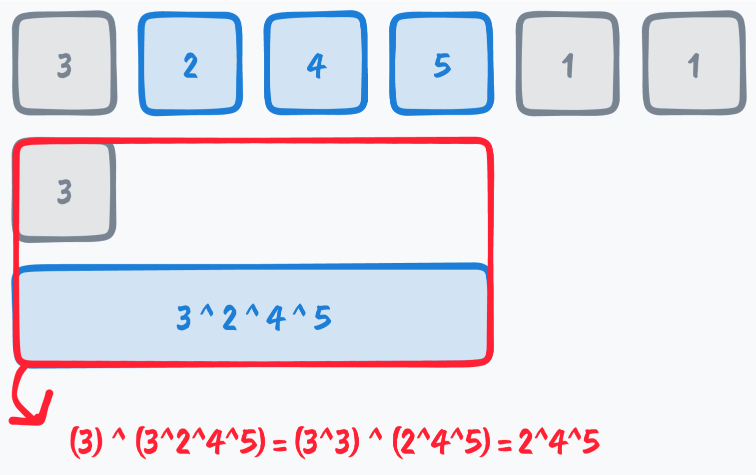
給予 q 個查詢，每個查詢都有兩個正整數 L_i，R_i

對於每個查詢求出 [L_i, R_i] 所有值做 xor 後的值

保證 $1 \leq n, q \leq 2 \times 10^5$

- 仍然是使用相同的概念，不過 xor 的值要怎麼消除呢？
- tip: $A \oplus A = 0$
- 我們可以可以做出類似前綴和陣列的東西，不過加的功能變成 xor
- 在查詢時，也使用 xor 消除

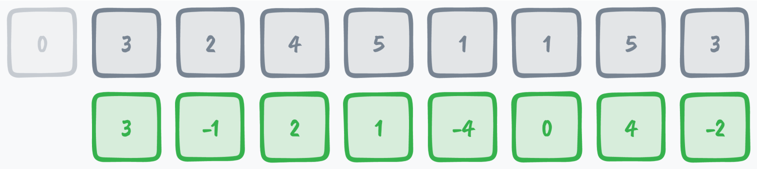
例題



差分概念

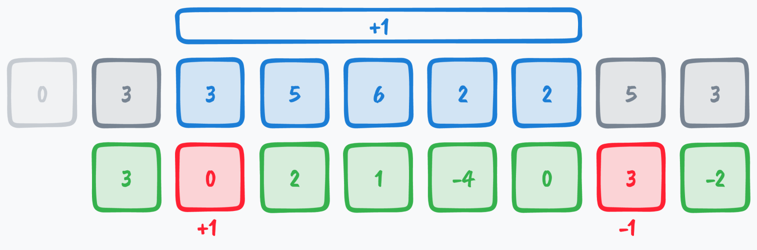
- 還記得我們說過前綴和的使用時機嗎？
- 讓我們把問題倒過來，有沒有辦法做到**區間修改，沒有查詢**
- 比起前綴和，差分的概念較為抽象

差分概念

- 

- 在差分裡，我們會儲存跟前面數字的差，以方便修改

差分概念

- 

- 可以透過圖示發現，修改區間的值實際上只會動到兩格
- 因為區間內的值的變化量一樣，所以差一樣 $(a - b = (a + x) - (b + x))$

差分概念

- 稍微整理一下，我們會有初始的差分序列
- 每次修改就是對 L 加上 value，而 R+1 加上 -value
- 如果需要尋找一個點的值，則要從左到又加上所有變化量

例題

差分陣列實作

[題目連結]

如題，試著實做出我們剛剛介紹的前綴和陣列

例題

疊鬆餅

題目連結

你有 n 個鬆餅需要從底層疊到頂層，每個鬆餅都有一個溼潤度 a_i，代表將該鬆餅放置頂層後，會讓底下 a_i 個（包含自己）鬆餅都沾上奶油請你輸出疊完這 n 個鬆餅後哪些鬆餅沾上了奶油

保證 $1 \leq n \leq 2 \times 10^5$ 且 $0 \leq a_i \leq n$