Chapter 4
陣列與均攤技巧
Array and Time Complexity Skills

台南女中資訊研究社 38th C++ 進階班課程

TNGS IRC 38th C++ Advanced Course

講師：陳俊安 Colten

陣列建表

建表可以幫助我們在要使用某個數據之時直接利用先前建好的表直接取值

這樣一來可以避免掉許多不必要的的計算，大幅優化時間複雜度

建表常常延伸到許多演算法的概念上，如前綴和、差分、質數篩等等

陣列建表：Ten Point Round #11 (Div. 2) pC Gunjyo 與骰子

先來估看看沒有建表的時間複雜度會是多少吧？

再來估看看使用建表的時間複雜度吧！

將所有骰子的情況建表

```
13      for(int i=1;i≤100;i++) // 第一個骰子
14      {
15          for(int k=1;k≤100;k++) // 第二個骰子
16          {
17              for(int j=1;j≤100;j++) // 第三個骰子
18              {
19                  check[i*k*j]++;
20
21                  check[i+k+j]++;
22
23                  check[i+k*j]++;
24
25                  check[i*k+j]++;
26              }
27          }
28      }
```

建完表之後就可以 O(1) 查詢了耶，好耶

```
30      int q;
31
32      cin >> q;
33
34      while(q--)
35      {
36          int input;
37
38          cin >> input;
39
40          cout << check[input] << "\n";
41      }
```

一維前綴和

- 區間快速求和的技巧
- 一開始先建立另外一個新的陣列 b
  - 定義 b_i = b_{i-1} + a_i
  - 這個時候我們想要求 a 陣列當中 [L , R] 的和
  - 答案就會是 b_R - b_{L-1}
  - 預處理時間複雜度 O(陣列長度)、查詢區間和 O(1)

Sample Code

```
35
36      int n;
37      cin >> n;
38      for(int i=1;i≤n;i++) cin >> a[i];
39      for(int i=1;i≤n;i++) b[i] = b[i-1] + a[i];
40
41      int l,r;
42      cin >> l >> r;
43      cout << b[r] - b[l-1] << "\n";
44
```

Array Skills Assignment 1

- CSES Problem Set Static Range Sum Queries

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to process q queries of the form: what is the sum of values in range $[a, b]$?

Input

The first input line has two integers n and q: the number of values and queries.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Finally, there are q lines describing the queries. Each line has two integers a and b: what is the sum of values in range $[a, b]$?

Output

Print the result of each query.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$
- $1 \le a \le b \le n$

例題 2: Educational Ten Point Round #1 pC. 偷工減料

C. 偷工減料 (Cut corners)

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Colten 有一個全部都是小寫英文字母的組合技 S，只要 Colten 使用這個組合技之後就會把當前在寫的題目 AC，因此 Koying 非常想要效仿 Colten。

於是 Koying 也創造了一個組合技只是效果沒有像 Colten 這麼好，因此有時候還是會得到 WA，因此他決定要把組合技做一些更改。

Koying 的組合技為 T，如果他要改變他組合技內的任何一個字元，那麼他必須花費 1 元作為更改的費用，不過 Koying 沒有那麼多錢，因此他決定偷工減料，只改某個區間 $[L, R]$ 的字元，讓這個區間內的所有字元都變得跟 Colten 的組合技字串中區間 $[L, R]$ 的字元的一樣，舉例來說：假設 S ="abcde" , T ="abedc"，那麼如果 Koying 改變區間 $[1, 3]$ 的話只需要花費 1 元，因為只有第 3 個字元跟 Colten 的不一樣，因此他只需要改第 3 個字元。

Koying 想要多次查詢他如果改變區間 $[L, R]$ 的字元會需要花費多少錢，因此請你設計一個程式幫幫他吧！

Input
只有一筆資料。

第一行輸入一個字串 S。

第二行輸入一個字串 T。

第三行輸入一個正整數 Q 表示接下來將有 Q 組詢問。

接下來將有 Q 行，每行依序輸入兩個正整數 L_i, R_i，表示詢問的區間。

測資範圍限制：

- $1 \leq |S| = |T| \leq 4 \cdot 10^5$
- $1 \leq L_i \leq R_i \leq |S| = |T|$
- $1 \leq Q \leq 100000$
- 保證字串中的所有字元都是小寫英文字母

Solution

- 簡化一下題目
- 給定兩個字串，接下來有 Q 筆查詢，答詢區間 [L , R] 有幾個不一樣的字元

Solution

- 開另外一個陣列，把不一樣的字元位置標記成 1
- 查詢 [L , R] = 那一個陣列的該區間有幾個位置是 1
- 如果字元一樣那麼該位置的結果是 0
- 因此就變成查詢該陣列 [L , R] 的總和
- 所以我們只要對該陣列建一個前綴和即可，時間複雜度 O(|S|)

Sample Code

```
15      string s1,s2;
16
17      cin >> s1 >> s2;
18
19      int total = 0; // 當前的總修改次數
20
21      for(int i=0;i<s1.size();i++)
22      {
23          if( s1[i] ≠ s2[i] ) total++; // 字元不一樣，總修改次數 + 1
24
25          prev_sum[i+1] = total; // 為了避免 overflow，因此我們紀錄在 i + 1 的位置
26
27          // 這麼做也可以方便等一下的查詢
28      }
```

Sample Code

```
30      int q;
31
32      cin >> q;
33
34      while(q--)
35      {
36          int l,r;
37
38          cin >> l >> r; // 查詢 [L,R] 的修改次數
39
40          cout << prev_sum[r] - prev_sum[l-1] << "\n"; // 區間和查詢
41      }
```

Codeforces Round 336 pB. Hamming Distance Sum

Genos needs your help. He was asked to solve the following programming problem by Saitama:

The length of some string s is denoted $|s|$. The Hamming distance between two strings s and t of equal length is defined as $\sum_{i=1}^{|s|} |s_i - t_i|$, where s_i is the i-th character of s and t_i is the i-th character of t. For example, the Hamming distance between string "0011" and string "0110" is $|0 - 0| + |0 - 1| + |1 - 1| + |1 - 0| = 0 + 1 + 0 + 1 = 2$.

Given two binary strings a and b, find the sum of the Hamming distances between a and all contiguous substrings of b of length $|a|$.

Input

The first line of the input contains binary string a ($1 \le |a| \le 200\,000$).

The second line of the input contains binary string b ($|a| \le |b| \le 200\,000$).

Both strings are guaranteed to consist of characters '0' and '1' only.

Output

Print a single integer — the sum of Hamming distances between a and all contiguous substrings of b of length $|a|$.

Examples

input	Copy
01 00111	
output	**Copy**
3	

Codeforces Round 336 pB. Hamming Distance Sum

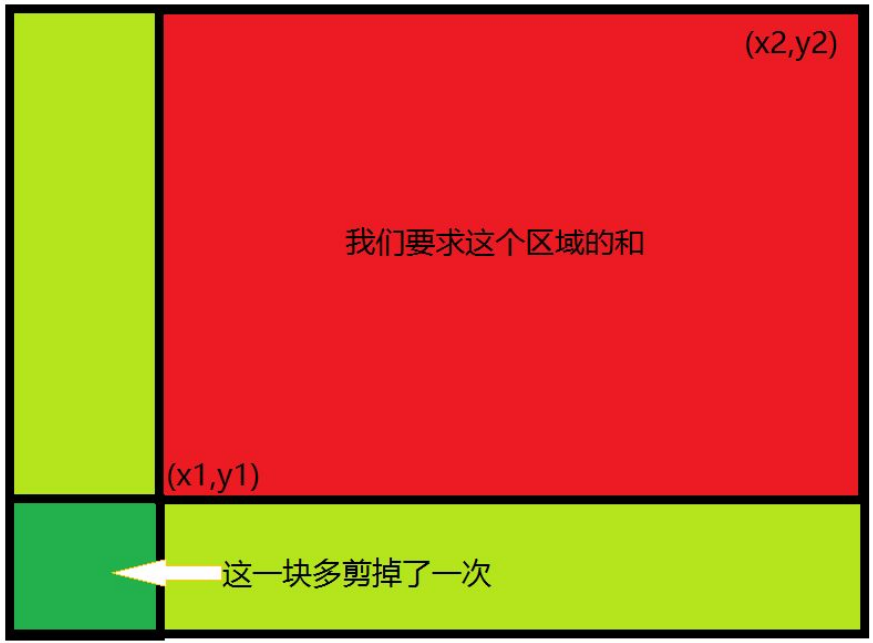
- 我們先觀察一下性質，對於第一個字串的第 i 個字元來說
- 他會掃過區間 [i , |s2| - |s1| + i + 1] 這一個區間
- 如果第 i 個字元是 1
  - 我們只需要知道這一個區間上有幾個 0
  - 就可以知道這一個位置貢獻了多少給答案
- 反之如果第 i 個字元是 0，那就找那一個區間 1 個數量
- 時間複雜度 O(|s1|)

Codeforces Round 336 pB. Hamming Distance Sum

```
int pre1[200005],pre2[200005];

signed main(void)
{
	ios_base::sync_with_stdio(false);
	cin.tie(0);

	string s1,s2;
	cin >> s1 >> s2;
	for(int i=1;i<=s2.size();i++)
	{
		pre1[i] = pre1[i-1] + ( s2[i-1] == '0' );
		pre2[i] = pre2[i-1] + ( s2[i-1] == '1' );
	}

	int ans = 0;

	for(int i=0;i<s1.size();i++)
	{
		if( s1[i] == '1' )
		{
			ans += pre1[s2.size()-s1.size()+1+i] - pre1[i];
		}
		else
		{
			ans += pre2[s2.size()-s1.size()+1+i] - pre2[i];
		}
	}

	cout << ans << "\n";

	return 0;
}
```

二維前綴和

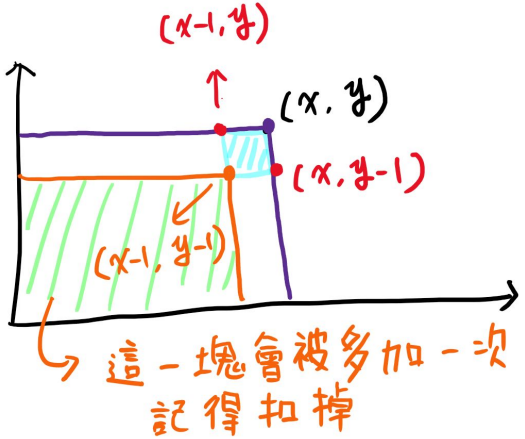
- 給定兩點，快速求這兩個點所圍成的矩形上，所有數字的總和
- 排容原理，算面積



二維前綴和的實作

- 另外建立一個陣列，定義陣列索引[xi][yi] 表示（１，１）到（xi，yi）的矩形面積總和
- 接著利用雙層的迴圈將每一個索引的答案都計算出來備用
- 建立完之後就可以利用這一個陣列來做二維前綴和的查詢了
- 接著我們就來來推出計算二維前綴和的式子吧！

預先建立（1，1）到（x，y）的和



利用排容原理，我們可以推出以下式子：

sum[i][k] :=（1，1）到（i，k）的前綴和

sum[i][k] = sum[i-1][k] + sum[i][k-1] - sum[i-1][k-1] + val[i][k]

做完這個預處理（建表）之後，我們就可以快速地查詢（x1，y1）到（x2，y2）的和了

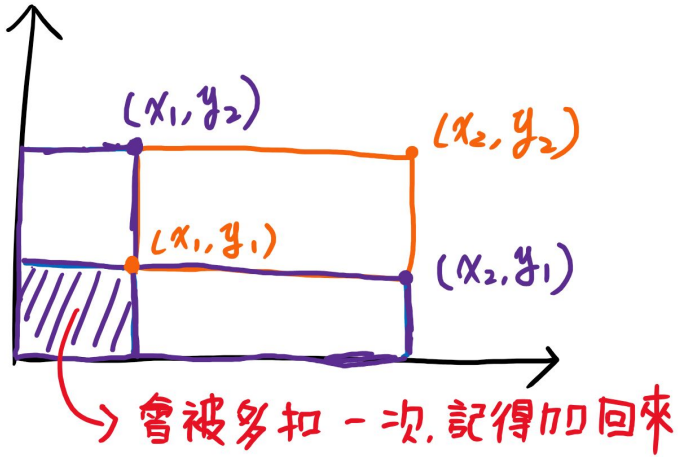
O(1) 查詢矩形上的元素總和

sum[i][k] := (0 , 0) 到 (i , k) 的前綴和

同樣利用排容原理我們

可以推出 (x1,y1) 到 (x2,y2)

的總和式子如下：



sum[x2][y2] - sum[x1-1][y2] - sum[x2][y1-1] + sum[x1-1][y1-1]

Sample Code

```
for(int i=0;i<n;i++)
{
    int x,y,k;

    cin >> x >> y >> k;

    x++; // 讓 x 跟 y 座標都 + 1，避免等等 overflow

    y++;

    a[x][y] += k;
}
```

Sample Code

```
34      for(int i=1;i≤1001;i++)
35      {
36          for(int k=1;k≤1001;k++)
37          {
38              s[i][k] = s[i-1][k] + s[i][k-1] - s[i-1][k-1] + a[i][k];
39
40              // s[i][k] = ( 1 , 1 ) 到 ( i , k ) 所圍成的矩形當中的籌碼數量
41              // 在輸入時把 x++ y++ 就是為了避免這裡 overflow
42          }
43      }
```

Sample Code

```
45      int q;
46
47      cin >> q;
48
49      while(q--)
50      {
51          int a,b,c,d;
52
53          cin >> a >> b >> c >> d; // x1,y1 跟 x2,y2
54
55          a++;
56
57          b++; // 由於輸入時我們將座標的 x y 都 + 1，因此這邊也要將 x y + 1
58
59          c++;
60
61          d++;
62
63          cout << s[c][d] - s[a-1][d] - s[c][b-1] + s[a-1][b-1] << "\n";
64      }
```

教學題：[TPR #23 H2 區間求和問題【Two-dimensional Version】]

- 我之前出的教學題
- 題敘有詳細教學，編了一個有趣的故事xD

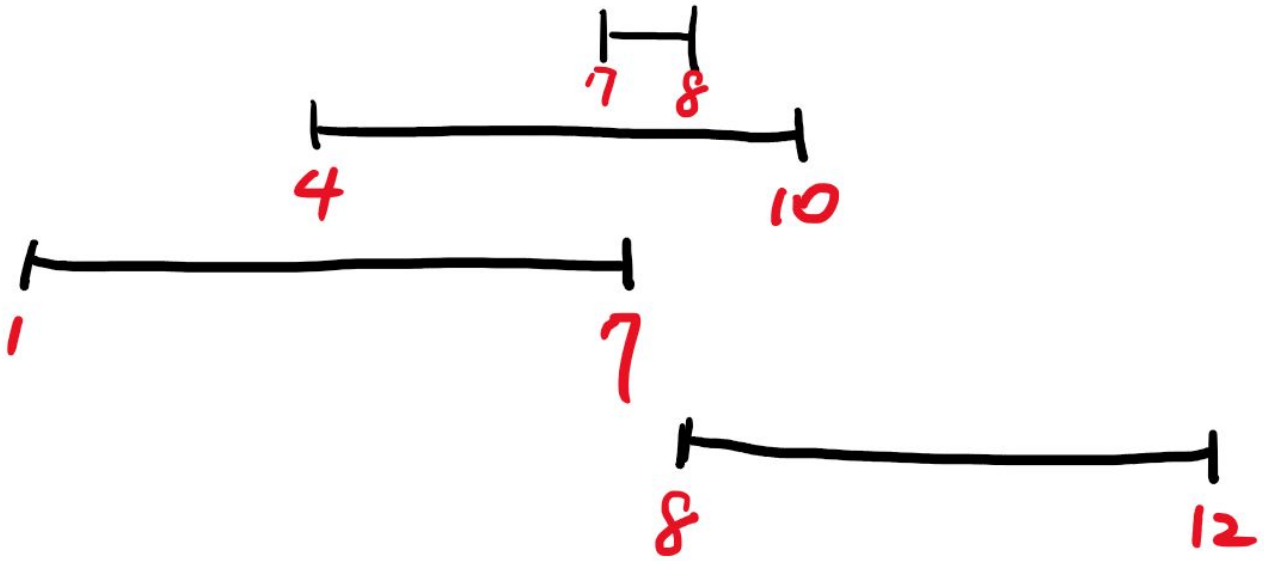
差分

差分，是任意兩個相鄰的值所相減形成的差的簡稱

例子：{ 1,5,6 } 這個序列的差分序列為 { 5 - 1 , 6 - 5 } = { 4 , 1 }

差分非常類似於標記的概念，只是這個技巧在解決問題的時後常常不太好聯想到，因此經驗的累積是非常重要的！！！

差分例題：Ten Point Round #8 pD. 公園調査



建立一個陣列，index 表示第 index 個小時增加多少人

```
7 int people[10005];
```

建立一個陣列，index 表示第 index 個小時增加多少人

```
7 int people[10005];
```

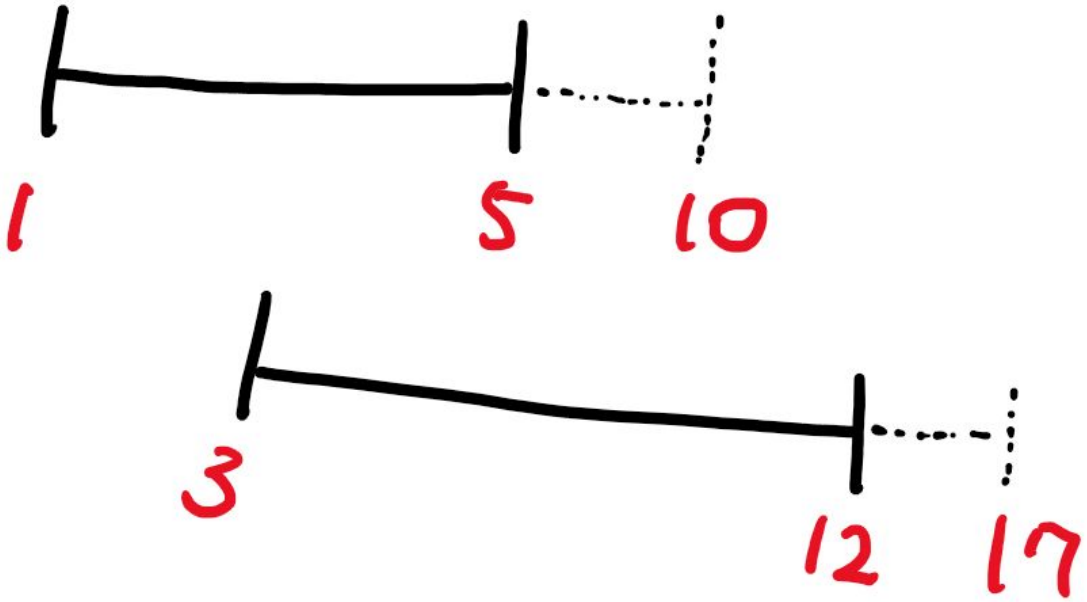
輸入處理

```
19      for(int i=0;i<n;i++)
20      {
21          int in,out;
22
23          cin >> in >> out;
24
25          people[in]++; // 第 in 小時會多 1 個人
26
27          people[out]--; // 第 out 小時會少 1 個人
28      }
```

計算答案

```
30      int ans = 0,total = 0; // total 是當前總人數
31
32      for(int i=1;i≤10000;i++) // 檢查第 1 個小時到第 10000 個小時的總人數
33      {
34          total += people[i]; // 第 i 個小時的人數 = 當前總人數 + 第 i 個小時的人數增加量
35
36          ans = max(ans,total);
37      }
38
39      cout << ans << "\n";
```

差分練習題：[Educational Ten Point Round #1 pB. 北門路上的科學家]



差分進階題：Codeforces Round 802 pC. Helping the Nature

- 現在有 n 個土壤，每一個土壤的水分是 ai
- 有三種操作
  - 選擇一個數字 i，把 [1 , i] 的土壤水分都 -1
  - 選擇一個數字 i，把 [i , n] 的土壤水分都 -1
  - 把全部的土壤都 +1
- 如果最後想要讓土壤水分都變成 0，最少只需要操作幾次即可？
- 1 <= n <= 2e5

差分進階題：Codeforces Round 802 pC. Helping the Nature

- 我們先把土壤的水分建立一個差分序列
  - $b_i = a_i - a_{(i-1)}$
    - $b_1 = a_1$
- **我們現在的目標就是要讓差分序列全部變成 0**
- 觀察看看三種操作使用後對差分序列的改變是什麼？

差分進階題：Codeforces Round 802 pC. Helping the Nature

- 第一種操作（把 [1 , i] 都減少 1 ）
  - 會使差分序列的第 i + 1 項 +1
  - 但差分序列的第 1 項會 -1
- 第二種操作（把 [i , n] 都減少 1 ）
  - 會使差分序列的第 i 項 -1
- 第三種操作（全部的土壤都+1 ）
  - 會使差分序列的第 1 項 +1

差分進階題：Codeforces Round 802 pC. Helping the Nature

- 問題就變轉換成
- 我們要把差分序列全部變成 0
- 我們要透過那三種操作來處理
  - 再講的簡單一點
  - 我們有把任意位置 +1 -1 的操作
  - 現在要知道把所有位置變成 0 的最小操作次數
- 把問題轉換成這樣之後，就變成一個超級簡單的題目囉！

均攤：CSES Problem Set Common Divisors

這題會需要跟枚舉的概念作搭配

提示：注意值域，枚舉最大公因數

如果數字 K 是這組序列某幾個數的最大公因數

- 表示，這個序列一定有至少兩個數字是 K 的倍數
- 上面這一點是這一題的解題關鍵！！！
- 因此我們就去枚舉最大公因數 K
- 看看這一個序列裡面有幾個數字是 K 的倍數
- 只要有 2 個以上就表示 K 是某兩個數字的最大公因數
    - K 由大枚舉到小 (因為要找最大公因數)

先觀察看看值域

x_i 不會超過 10 的 6 次方這代表什麼？

答案絕對不會超過 10 的 6 次方！

Constraints

- $2 \leq n \leq 2 \cdot 10^5$
- $1 \leq x_i \leq 10^6$

那我們一開始先將數列的狀態紀錄到一個陣列裡面

```
21      for(int i=0;i<n;i++)
22      {
23          cin >> a[i];
24
25          check[a[i]]++; // 定義 check[index] 表示這個序列當中有 index 這個數字的個數
26      }
```

接著我們枚舉答案，並看看這個答案是否合法

```
28      int ans = 1; // 如果都找不到，答案會是 1
29
30      for(int i=1000000;i≥2;i--) // i 表示我們枚舉的答案
31      {
32          int total = 0; // 這個序列當中，可以被 i 整除的數量
33
34          for(int j=1;i * j ≤ 1000000;j++) // 可以被 i 整除了數字 1i,2i,3i,4i (枚舉 i 的倍數)
35          {
36              if( check[ i * j ] ≥ 1 ) total += check[i*j]; // 發現這個序列有 i 的倍數
37
38              if( total ≥ 2 ) break; // 如果已經找到至少 2 個數字了，就可以結束了
39          }
40
41          if( total ≥ 2 )
42          {
43              ans = i;
44
45              break;
46          }
47      }
48
49      cout << ans << "\n";
```

時間複雜度？均攤在每一次的枚舉最大公因數中

- O(10^6 / 2 + 10^6 / 3 + ⋯ + 10^6 / 10^6)
  - O(10^6 * log(10^6))

```
28    int ans = 1; // 如果都找不到，答案會是 1
29
30    for(int i=1000000;i≥2;i--) // i 表示我們枚舉的答案
31    {
32        int total = 0; // 這個序列當中，可以被 i 整除的數量
33
34        for(int j=1;i * j ≤ 1000000;j++) // 可以被 i 整除了數字 1i,2i,3i,4i (枚舉 i 的倍數)
35        {
36            if( check[ i * j ] ≥ 1 ) total += check[i*j]; // 發現這個序列有 i 的倍數
37
38            if( total ≥ 2 ) break; // 如果已經找到至少 2 個數字了，就可以結束了
39        }
40
41        if( total ≥ 2 )
42        {
43            ans = i;
44
45            break;
46        }
47    }
48
49    cout << ans << "\n";
```