

Thuật toán ứng dụng

Bài thực hành 1

TS. Ban Hà Bằng
TA. Nguyễn Thị Linh





Nội dung

1. Telco data check & query
 2. Maze
 3. Range Minimum Query
 4. Largest Black Subrectangle
-

1. Telco data check & query

Write a C++ program to perform some queries on a telco data (coming from stdin) with the following format:

The first block of data consists of lines (terminated by a line containing #), each line (number of lines can be up to 100000) is under the form:

call <from_number> <to_number> <date> <from_time> <end_time>

which is a call from the phone number <from_number> to a phone number <to_number> on <date>, and starting at time-point <from_time>, terminating at time-point <end_time>

- <from_number> and <to_number> are string of 10 characters (a phone number is correct if it contains only digits 0,1,...,9, otherwise, the phone number is incorrect)
- <date> is under the form YYYY-MM-DD (for example 2022-10-21)
- <from_time> and <to_time> are under the form hh:mm:ss (for example, 10:07:23)

The second block consists of queries (terminated by a line containing #), each query in a line (number of lines can be up to 100000) and belongs to one of the following types:

- ?check_phone_number: print to stdout (in a new line) value 1 if no phone number is incorrect
- ?number_calls_from <phone_number>: print to stdout (in a new line) the number of times a call is made from <phone_number>
- ?number_total_calls: print to stdout (in a new line) the total number of calls of the data
- ?count_time_calls_from <phone_number>: print to stdout (in a new line) the total time duration (in seconds) the calls are made from <phone_number>

1. Telco data check & query

Ví dụ:

Input

```
call 0912345678 0132465789 2022-07-12 10:30:23 10:32:00
call 0912345678 0945324545 2022-07-13 11:30:10 11:35:11
call 0132465789 0945324545 2022-07-13 11:30:23 11:32:23
call 0945324545 0912345678 2022-07-13 07:30:23 07:48:30
#
?check_phone_number
?number_calls_from 0912345678
?number_total_calls
?count_time_calls_from 0912345678
?count_time_calls_from 0132465789
#
```

Output

```
1
2
4
398
120
```

Ý tưởng

Truy vấn 1: ?check_phone_number

Tạo hàm kiểm tra:

- Nếu độ dài của sđt khác 10
-> FALSE
- Nếu sđt bao gồm kí tự không phải trong
đoạn [0..9]
-> FALSE
- Ngược lại -> TRUE





Contents

1. Telco data check & query
 2. Maze
 3. Range Minimum Query
 4. Largest Black Subrectangle
-

1.Telco data check & query

Example:

Input

```
call 0912345678 0132465789 2022-07-12 10:30:23 10:32:00
call 0912345678 0945324545 2022-07-13 11:30:10 11:35:11
call 0132465789 0945324545 2022-07-13 11:30:23 11:32:23
call 0945324545 0912345678 2022-07-13 07:30:23 07:48:30
#
?check_phone_number
?number_calls_from 0912345678
?number_total_calls
?count_time_calls_from 0912345678
?count_time_calls_from 0132465789
#
```

Output

```
1
2
4
398
120
```

Ý tưởng

Truy vấn 2: ?number_calls_from<phone_number>

Vì phone_number là một chuỗi

-> Sử dụng `map<string,int>` để đếm số cuộc gọi từ phone_number



Ý tưởng

Truy vấn 3: ?number_total_calls

Dùng 1 biến đếm số lượng input nhập vào



Ý tưởng

Truy vấn 4: ?count_time_calls_from <phone_number>

- Tạo hàm đếm thời gian

```
int countTime (string ftime, string etime)
{
    int startTime = 3600*((ftime[0]-'0')*10 + ftime[1]-'0')
                  + 60*((ftime[3]-'0')*10 + ftime[4]-'0')
                  + ((ftime[6]-'0')*10 + ftime[7]-'0');
    int endTime = 3600*((etime[0]-'0')*10 + etime[1]-'0')
                 + 60*((etime[3]-'0')*10 + etime[4]-'0')
                 + ((etime[6]-'0')*10 + etime[7]-'0');
    return endTime - startTime;
}
```

- Lưu thời gian của mỗi sdt vào map<string, int>



2. Maze

Một mê cung hình chữ nhật được biểu diễn bởi 0-1 ma trận $N \times M$ trong đó $A[i,j] = 1$ thể hiện ô (i,j) là tường gạch và $A[i,j] = 0$ thể hiện ô (i,j) là ô trống, có thể di chuyển vào. Từ 1 ô trống, ta có thể di chuyển sang 1 trong 4 ô lân cận (lên trên, xuống dưới, sang trái, sang phải) nếu ô đó là ô trống. Xuất phát từ 1 ô trống trong mê cung, hãy tìm đường ngắn nhất thoát ra khỏi mê cung.

Input

- Dòng 1: ghi 4 số nguyên dương n, m, r, c trong đó n và m tương ứng là số hàng và cột của ma trận A ($1 \leq n, m \leq 999$) và r, c tương ứng là chỉ số hàng, cột của ô xuất phát.
- Dòng $i+1$ ($i=1, \dots, n$): ghi dòng thứ i của ma trận A

Output

Ghi giá số bước cần di chuyển ngắn nhất để thoát ra khỏi mê cung, hoặc ghi giá trị -1 nếu không tìm thấy đường đi nào thoát ra khỏi mê cung.

2. Maze

Ví dụ

Input

8 12 5 6

1 1 0 0 0 0 1 0 0 0 0 1

1 0 0 0 1 1 0 1 0 0 1 1

0 0 1 0 0 0 0 0 0 0 0 0

1 0 0 0 0 0 1 0 0 1 0 1

1 0 0 1 0 0 0 0 0 1 0 0

1 0 1 0 1 0 0 0 1 0 1 0

0 0 0 0 1 0 1 0 0 0 0 0

1 0 1 1 0 1 1 1 0 1 0 1

Output

7

Ý tưởng

Sử dụng thuật toán BFS để tìm đường

- Lấy vị trí $[r, c]$ từ đỉnh queue ra
- Di chuyển sang 4 ô lân cận:
 - $[r-1, c]$ // lên trên
 - $[r+1, c]$ // xuống dưới
 - $[r, c-1]$ // sang trái
 - $[r, c+1]$ // sang phải
- Nếu ô đó là ô trống ($a[i][j] = 0$) và chưa được đi qua lần nào --> di chuyển sang và thêm ô đó vào queue
- Kết quả là giá trị của ô ở ngoài biên đầu tiên được di chuyển tới



Code



```
#include <bits/stdc++.h>

using namespace std;

typedef pair<int,int> ii;

const int maxN = 999 + 100;

const int oo= 1e9 + 7;

int a[maxN][maxN] , m , n , r , c , d[maxN][maxN];

int dx[] = {1 , 0, -1 , 0} ,
        dy[] = {0 , 1, 0 , -1};

queue<ii> qe;
```

Code

```
int solve(){
    qe.push(ii(r,c));  d[r][c] = 0;  a[r][c] = 1;
    while(!qe.empty()){
        ii u = qe.front(); qe.pop();
        for(int i = 0 ; i < 4 ; i++){
            int x = dx[i] + u.first;          int y = dy[i] + u.second;
            if(x < 1 || x > m || y < 1 || y > n) return d[u.first][u.second] + 1;
            if(a[x][y] != 1){
                d[x][y] = d[u.first][u.second] + 1;
                qe.push(ii(x,y));
                a[x][y] = 1;
            }
        }
    }
    return -1;
}
```



Code

```
int main(){  
    ios_base::sync_with_stdio(false);cin.tie(0);  
    cin >> m >> n >> r >> c;  
    for(int i = 1 ; i <= m ; i++) for(int j = 1 ; j <= n ; j++) cin >> a[i][j];  
    int ans = solve();  
    cout << ans;  
    return 0;  
}
```



3. Range Minimum Query

Given a sequence of n integers a_0, \dots, a_{n-1} . We denote $\text{rmq}(i, j)$ the minimum element of the sequence a_i, a_{i+1}, \dots, a_j . Given m pairs $(i_1, j_1), \dots, (i_m, j_m)$, compute the sum $Q = \text{rmq}(i_1, j_1) + \dots + \text{rmq}(i_m, j_m)$

Input

- Line 1: n ($1 \leq n \leq 10^6$)
- Line 2: a_0, \dots, a_{n-1} ($1 \leq a_i \leq 10^6$)
- line 3: m ($1 \leq m \leq 10^6$)
- Line $k+3$ ($k=1, \dots, m$): i_k, j_k ($0 \leq i_k < j_k < n$)

Output

- Write the value Q

3. Range Minimum Query

Example

Input

16

2 4 6 1 6 8 7 3 3 5 8 9 1 2 6 4

4

1 5

0 9

1 15

6 10

Ouput

6

Ý tưởng

Cách 1: Duyệt

- Với mỗi cặp truy vấn (i, j) nhập vào, ta duyệt hết đoạn từ i tới j để tìm giá trị nhỏ nhất
- Tính tổng tất cả các giá trị nhỏ nhất của từng đoạn đã tìm được ở trên

Độ phức tạp: $O(m \cdot n)$



Ý tưởng

Cách 2:

Với mỗi truy vấn (i, j) , ta cần đưa ra giá trị của phần tử có giá trị nhỏ nhất trong đoạn từ i đến j trong $O(1)$

Ký hiệu $RMQ(i, j)$

--> Với m truy vấn, ta có thể giải quyết bài toán trong vòng $O(m)$



Ý tưởng

Tiền xử lý:

Ta sử dụng mảng $M[0, N-1][0, \log N]$ để lưu giá trị của $\text{RMQ}(i, j)$ với $M[i][j]$ là giá trị nhỏ nhất trong đoạn có độ dài 2^j và bắt đầu ở i .

Ví dụ:

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
2	4	3	1	6	7	8	9	1	7

$M[1][0] = 4$

$M[1][1] = 3$

$M[1][2] = 1$



Ý tưởng

Để tính $M[i][j]$, ta xét M của 2 nửa đầu và nửa cuối của đoạn, mỗi phần sẽ có độ dài 2^{j-1} :

$$M[i][j] = \min (M[i][j - 1], M[i + 2^{j-1} - 1][j - 1])$$

```
void process2(int M[MAXN][LOGMAXN], int A[MAXN], int N)
{
    int i, j;

    // Khởi tạo M với các khoảng độ dài 1
    for (i = 0; i < N; i++)
        M[i][0] = A[i];

    // Tính M với các khoảng dài  $2^j$ 
    for (j = 1; 1 <= j <= N; j++)
        for (i = 0; i + (1 <= j) - 1 < N; i++)
            if (M[i][j - 1] < M[i + (1 <= (j - 1))][j - 1])
                M[i][j] = M[i][j - 1];
            else
                M[i][j] = M[i + (1 <= (j - 1))][j - 1];
}
```



Ý tưởng

Để tính $RMQ(i,j)$ ta dựa vào 2 đoạn con độ dài 2^k phủ hết $[i,j]$,
với $k = \lfloor \log(j-i+1) \rfloor$:

$$RMQ[i][j] = \min (M[i][k], M[j - 2^k + 1][k])$$



```
long long result = 0;
int i, j, k;
cin >> m;
for (int t=1; t<=m; t++) {
    cin >> i >> j;
    k = log2(j-i+1);
    result += min(M[i][k], M[j-(1<<k)+1][k]);
}
```

Độ phức tạp tổng quát của thuật toán này là $\langle O(N \log N), O(m) \rangle$

4. Largest Black SubRectangle

Một hình chữ nhật kích thước $n \times m$ được chia thành các ô vuông con 1×1 với 2 màu đen hoặc trắng. Hình chữ nhật được biểu diễn bởi ma trận $A(n \times m)$ trong đó $A(i, j) = 1$ có nghĩa ô hàng i , cột j là ô đen và $A(i, j) = 0$ có nghĩa ô vuông hàng i cột j là ô trắng. Hãy xác định hình chữ nhật con của bảng đã cho bao gồm toàn ô đen và có diện tích lớn nhất.

- Dữ liệu
 - · Dòng 1: chứa số nguyên dương n và m ($1 \leq n, m \leq 1000$)
 - · Dòng $i+1$ ($i = 1, \dots, n$): chứa hàng thứ i của ma trận A
- Kết quả
 - · Ghi ra diện tích của hình chữ nhật lớn nhất tìm được

4. Largest Black SubRectangle

Example

Input

4 4

0 1 1 1

1 1 1 0

1 1 0 0

1 1 1 0

Output

6

Ý tưởng

Với mỗi hàng của ma trận. Tính $h[i]$ là “chiều cao” của cột i . Ví dụ:

0	1	1	1
1	1	1	0
1	1	0	0
1	1	1	0



0	1	1	1
1	2	2	0
2	3	0	0
3	4	1	0

Ý tưởng

- Gọi $L[i]$ là vị trí đầu tiên về bên trái có giá trị nhỏ hơn $h[i]$
 $R[i]$ là vị trí đầu tiên về bên phải có giá trị nhỏ hơn $h[i]$
- Sau khi có được mảng $h[i]$, sử dụng kĩ thuật tìm min, max trên một đoạn tính tiến bằng deque để tính L, R một cách nhanh chóng
- Với mỗi cột i , hình chữ nhật lớn nhất có thể với cột đó là
$$S = (R[i] - L[i] - 1) * h[i]$$



Ý tưởng

Tính L

```
struct pack { int h, id; };  
queue<pack> q;  
// calculate left  
for (int j=1; j<=m; j++) {  
    // loai bo nhung phan tu lon hon h[j]  
    while (!q.empty() && q.back().h >= h[j]) q.pop();  
    if (q.empty()) l[j] = 0;  
    else l[j] = q.back().id;  
    // day phan tu j vao queue  
    q.push({h[j], j});  
}
```



Ý tưởng

Tính R

```
// khởi tạo lại queue
while (!q.empty()) q.pop();
for (int j=m; j>=1; j--) {
    // loại bỏ nhng phần tử lớn hơn h[j]
    while (!q.empty() && q.back().h >= h[j]) q.pop();
    if (q.empty()) r[j] = m+1;
    else r[j] = q.back().id;
    // đẩy phần tử j vào queue
    q.push({h[j], j});
}
```



Độ phức tạp: $O(m*n)$