

# Bài thực hành số 1

- 1.Telco data check & query
- 2.MAZE
- 3.Range Minimum Query
- 4.Larget black subrectangle

# 1.Telco data check & query

---

Bài toán: Viết chương trình C++ thực hiện công việc truy vấn vào dữ liệu điện thoại.

Dòng dữ liệu có dạng:

**call <from\_number> <to\_number> <date> <from\_time> <end\_time>**

**<from\_number>** : xâu 10 kí tự chữ số (nếu khác, số không hợp lệ)

**<date>**: theo dạng YYYY-MM-DD (ví dụ 2022-10-21)

**<from-time>, <to-time>** : theo dạng hh:mm:ss (ví dụ 10:07:23)

Giới hạn lên đến  $10^5$  dòng.

# 1.Telco data check & query

---

Dòng truy vấn: có 4 loại truy vấn.

?**check\_phone\_number**: in 1 nếu không có số nào sai định dạng, in ra 0 nếu ngược lại.

?**number\_calls\_from** <**phone\_number**>: số cuộc gọi đi của số <phone number>.

?**number\_total\_calls**: tổng số cuộc gọi.

?**count\_time\_calls\_from** <**phone\_number**>: tổng thời gian gọi đi của <phone number>.

Giới hạn lên đến  $10^5$  truy vấn.

# 1.Telco data check & query

---

Thuật toán:

?**check\_phone\_number** : Kiểm tra từng số điện thoại theo định dạng.

?**number\_total\_calls**: Đếm số lượt gọi.

?**number\_calls\_from** <**phone\_number**>: sử dụng cấu trúc dữ liệu map

?**count\_time\_calls\_from** <**phone\_number**>: sử dụng cấu trúc dữ liệu map

## 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

#

# 1.Telco data check & query

---

Cấu trúc dữ liệu map: quản lí một cặp key/value.

Khởi tạo: `map<string,int> numberCalls.`

Tìm kiếm giá trị với key: `numberCalls[key]`

*Các thao tác trong map (thêm, tìm kiếm) có độ phức tạp  $O(\log n)$*

Độ phức tạp của thuật toán:  $O(n \cdot \log n)$

# 1.Telco data check & query

---

```
#include <bits/stdc++.h>
using namespace std;
bool checkPhone (string s){
    if (s.length() != 10) return false;
    for (int i=0; i<s.length(); i++)
        if (!(s[i]>='0' && s[i]<='9')) return false;
    return true;
}
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;
}
map <string,int> numberCalls, timeCall;
```

# 1.Telco data check & query

---

```
int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(NULL);
    cout.tie(NULL);
    string type;
    int totalCalls = 0;
    int incorrectPhone = 0;
    do {
        cin >> type;
        if (type == "#") continue;
        ++totalCalls;
        string fnum, tnum, date, ftime, etime;
        cin >> fnum >> tnum >> date >> ftime >> etime;
        if (!checkPhone(fnum) || !checkPhone(tnum)) ++incorrectPhone;
        numberCalls[fnum]++;
        timeCall[fnum] += countTime(ftime, etime);
    } while (type!="#");
```



# 1.Telco data check & query

---

```
do {
    cin >> type;
    if (type == "#") continue;
    if (type == "?check_phone_number") {
        if (incorrectPhone == 0) cout << 1 << endl; else cout << 0 << endl;
    } else if (type == "?number_calls_from") {
        string phone; cin >> phone;
        cout << numberCalls[phone] << endl;
    } else if (type == "?number_total_calls")
        cout << totalCalls << endl;
    else if (type == "?count_time_calls_from") {
        string phone; cin >> phone;
        cout << timeCall[phone] << endl;
    }
} while (type != "#");
return 0;
}
```

## 2.MAZE

---

- Bài toán: Cho một mê cung và vị trí xuất phát ban đầu. Tìm số bước ít nhất để thoát khỏi mê cung. Chỉ được di chuyển theo 4 hướng: trái, phải, trên, dưới.
- Các giới hạn:  $m, n < 1000$
- Output: Một số nguyên là số bước ít nhất. Nếu không tìm thấy, in -1

## 2.MAZE

---

- Thuật toán: thực hiện tìm kiếm theo chiều rộng (BFS) từ điểm xuất phát. Nếu có thể ra được ngoài, in ra kết quả. Ngược lại, in ra -1.

## 2.MAZE

---

```
#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;
```

## 2.MAZE

---

```
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;
}
```



## 2.MAZE

---

```
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

---

Bài toán: Cho một dãy số  $n$  phần tử  $a_0, \dots, a_{n-1}$ . gọi  $\text{rmq}(i, j)$  là phần tử bé nhất của  $a_i, a_{i+1}, \dots, a_j$ . Cho  $m$  cặp  $(i_1, j_1), \dots, (i_m, j_m)$ , Tính tổng  $Q = \text{rmq}(i_1, j_1) + \dots + \text{rmq}(i_m, j_m)$ .

Input: Dòng 1 chứa số nguyên  $n$  ( $1 \leq n \leq 10^6$ )

Dòng 2 chứa  $n$  số nguyên  $a_0, a_1, \dots, a_{n-1}$  ( $1 \leq a_i \leq 10^6$ )

Dòng 3 chứa số nguyên  $m$ .

Dòng các dòng tiếp theo chứa  $m$  cặp  $i_k, j_k$  ( $0 \leq i_k < j_k < n$ )

Output: in ra giá trị  $Q$



### 3.Range Minimum Query

- Thuật toán: Sử dụng RMQ.  
Gọi mảng  $M[k][i]$  là giá trị min trong đoạn từ  $i$  đến  $i + 2^k - 1$

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

$M[1][0] = 1$

$M[1][1] = 2$

$M[1][2] = 3$

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

$$M[i][j] = \begin{cases} M[i][j-1], & A[M[i][j-1]] \leq A[M[i + 2^{j-1} - 1][j-1]] \\ M[i + 2^{j-1} - 1][j-1], & \text{otherwise} \end{cases}$$

# 4. Largest black subrectangle

---

- Bài toán: Tìm hình chữ nhật lớn nhất chứa các chữ số 1.
- Input: Dòng 1 chứa Số nguyên dương  $m, n$  ( $1 \leq m, n \leq 1000$ )
- Dòng tiếp theo chứa các hàng của ma trận.
- Output: Diện tích hình chữ nhật lớn nhất tìm được.

# 4. Largest black subrectangle

- Thuật toán: Thực hiện với mỗi hàng của ma trận. Tính  $h[i]$  là “chiều cao” của cột  $i$ . Ví dụ:

4 4

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

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

$h[i]$

# 4. Largest black subrectangle

- Sau khi có được mảng  $h[i]$ , sử dụng kĩ thuật tìm min, max trên một đoạn tịnh tiến bằng deque.
- $L[i]$  là vị trí bên trái đầu tiên bé hơn  $h[i]$ .
- $R[i]$  là vị trí bên phải đầu tiên bé hơn  $h[i]$ .

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

3	4	1	0
---	---	---	---

L	0	1	0	0
R	3	3	4	5

## 4. Largest black subrectangle

---

- Với mỗi cột  $i$ , hình chữ nhật lớn nhất có thể với cột đó là  $S = (R[i] - L[i] - 1) * h[i]$