ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Thuật toán ứng dụng thực hành

Buổi 1-Data structures & Library

Teams code

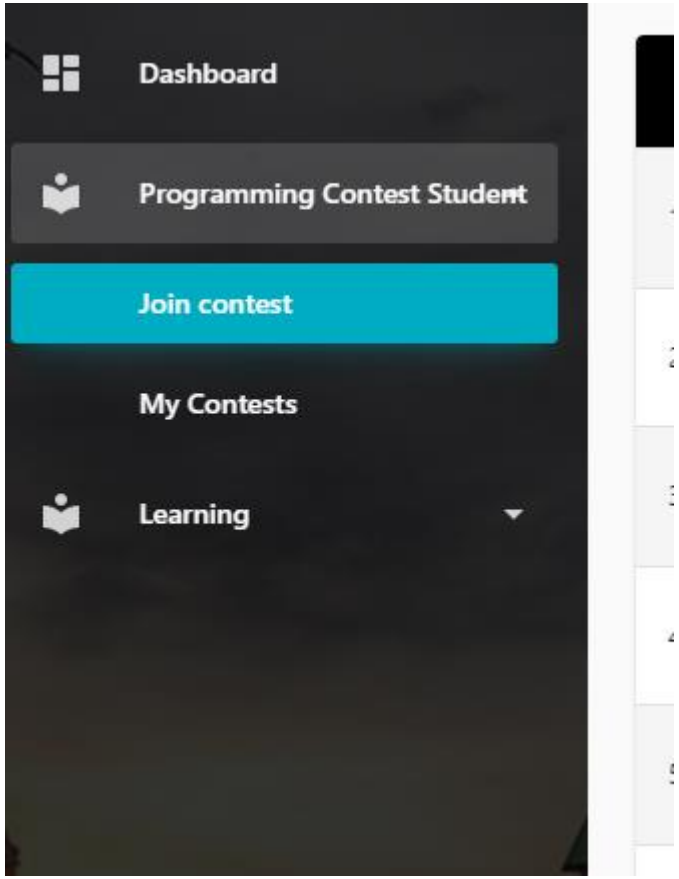
- Team 1: 330dwgh
  - tất cả các vấn đề liên quan đến thực hành, thi giữa kỳ, cuối kỳ, hệ thống submit bài,..
- Team 2: m5d3p76
  - Đề bài, bài chữa các buổi thực hành

- Hệ thống dùng cho thực hành submit code: https://openerp.dailyopt.ai/
- Đăng ký và đăng nhập
- Tại pannel Dashboard chọn Join contest
- Tìm contest

Thuật toán ứng dụng - Applied Algorithm - 20221

- Và ấn nút

REGISTER



Bài tập

1.Telco Data Check & Analyze

2. MAZE

3.Range Minimum Query

4.Largest Black SubRectangle

1.Telco Data Check & Analyze (Kiểm tra và phân tích dữ liệu log cuộc gọi thoại)

- Một nhà mạng muốn thực hiện truy vấn dữ liệu log lịch sử cuộc gọi trong ngày, dữ liệu log này được format dạng:

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

Ý nghĩa của các trường là

- Từ khóa call: đây là log cuộc gọi điện thoại
- <from_number> và <to_number>: là SDT gọi và nhận cuộc gọi, là kiểu xâu ký tự độ dài 10 (chỉ gồm các chữ số 0-9)
- <date>: Là ngày thực hiện cuộc gọi theo định dạng YYYY-MM-DD (VD. 2022-10-21)
- <from_time> và <end_time>: Là thời gian bắt đầu, kết thúc cuộc gọi trong ngày (định dạng theo hh:mm:ss, VD. 10:07:23)

Chú ý:

- Số lượng log cuộc gọi này có thể lớn tới 100000 dòng
- Các tham số ngăn cách với nhau bởi 1 dấu cách trống

1.Telco Data Check & Analyze (Kiểm tra và phân tích dữ liệu log cuộc gọi thoại)

Các truy vấn dữ liệu log được đưa vào với định dạng bắt đầu bằng dấu ?, trong đó

- ?check_phone_number: in ra màn hình (dòng mới) giá trị 1 nếu các số điện thoại đều hợp lệ
- ?number_calls_from <phone_number>: in ra màn hình (dòng mới) số cuộc gọi được xuất phát từ SDT <phone_number>
- ?number_total_calls: in ra màn hình (dòng mới) tổng số cuộc gọi có trong log
- ?count_time_calls_from <phone_number>: in ra màn hình (dòng mới) tổng thời gian gọi (tính theo second) xuất phát từ SDT <phone_number>

Chú ý:

- Số lượng truy vấn cũng có thể lên tới 100000 dòng
- Các tham số ngăn cách với nhau bởi 1 dấu cách trống

1.Telco Data Check & Analyze

- Example

stdin	stdout
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 2 4 398 120

1.Telco Data Check & Analyze

Gợi ý chung

• Xác định đâu là phần dữ liệu log và đâu là phần truy vấn nhờ ký hiệu #, hoặc nhờ từ khóa call và
dấu ? ở trước, hoặc nhờ ký tự c và ? hoặc # ở đầu dòng

• Đọc các phần của log và tách ra các trường? Các tham số này đều tuân theo format chuẩn tuy nhiên nếu đọc vào thì sẽ bị lỗi do có ký tự - và :

cin>>sdt1>>sdt2>>y1>>m1>>d1>>h1>>m1>>s1>>h2>>m2>>s2

→Vậy có thể dùng dạng scanf("%s%s%d-%2d-%2d%2d:%2d:%2d%2d:%2d:%2d",...)

```
char sdt1[15], sdt2[15];
int y1,mm1,d1,h1,m1,s1,h2,m2,s2;
scanf("%s %s %d-%d-%d %d:%d:%d
%d:%d:%d",sdt1,sdt2,&y1,&mm1,&d1,&h1,&m1,&s1,&h2,&m2,&s2);
```

o tuy nhiên dễ nhất là đọc thành xâu ý tự rồi xử lý tiếp (nhưng bạn sẽ cần thêm việc chuyển xâu thành giá trị số)

• Kiểm tra SDT đúng định dạng hay sai định dạng? Độ dài 10 là đủ?

• Tính thời gian thoại: Chỉ là thời gian trong cùng 1 ngày

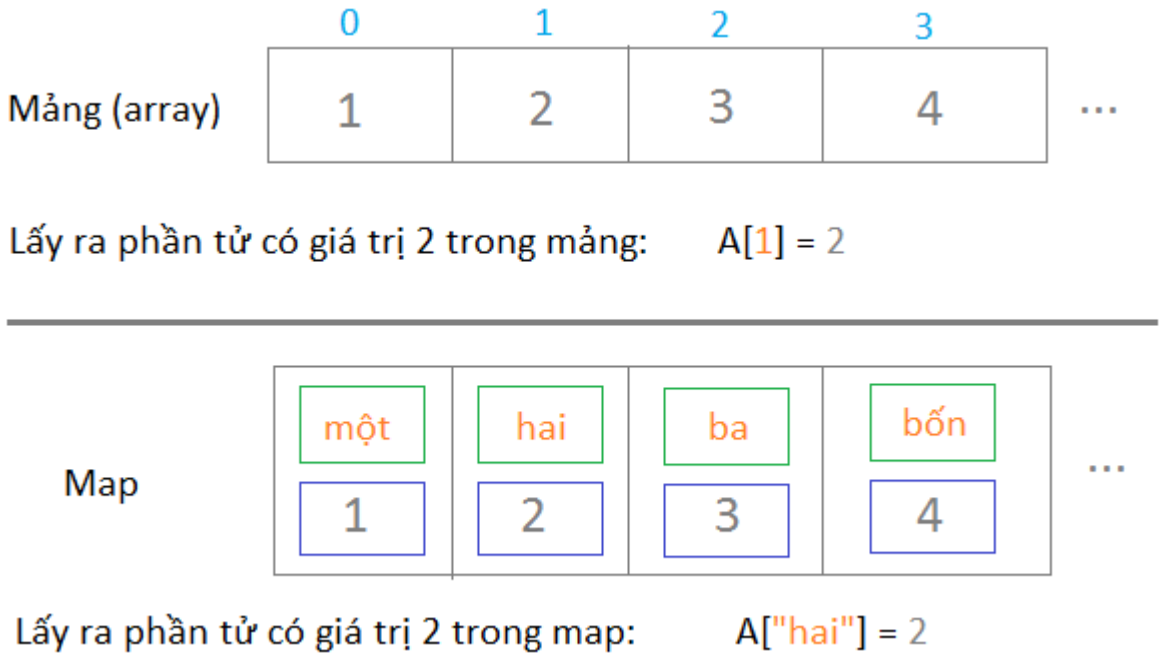
1.Telco Data Check & Analyze

Sử dụng map

```
map <kiểu dữ liệu của key, kiểu dữ liệu của value> someMap;
```

- Trong đó kiểu dữ liệu của key thường là string, kiểu dữ liệu của value thường là giá trị số như int, float, ..
- Ví dụ:

```
map<string, int> A;
// Khởi tạo một map A
// Thêm vào map A một số phần tử.
 A[“một"] = 1;
 A[“hai"] = 2;
 A[“ba"] = 3;
```



Implementation

```
#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;
}
```

Implementation

```
map <string,int> numberCalls, timeCall;
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!="#");
```

Implementation

```
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

- Một mê cung hình chữ nhật được biểu diễn bởi 0-1 ma trận NxM 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 <= n,m <= 999) và r, c tương ứng là chỉ số hàng, cột của ô xuất phát.
  - Dòng i+1 (i=1,...,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.

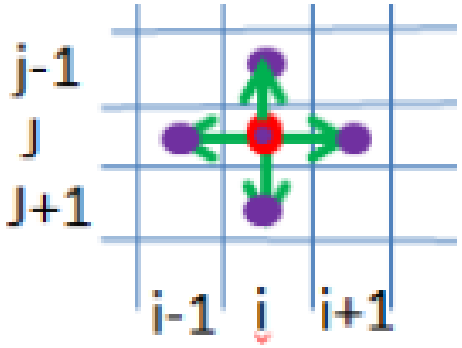
2.MAZE

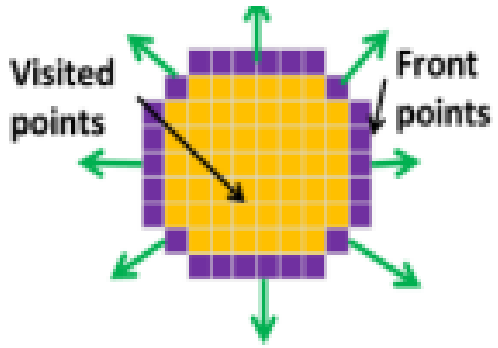
- Example

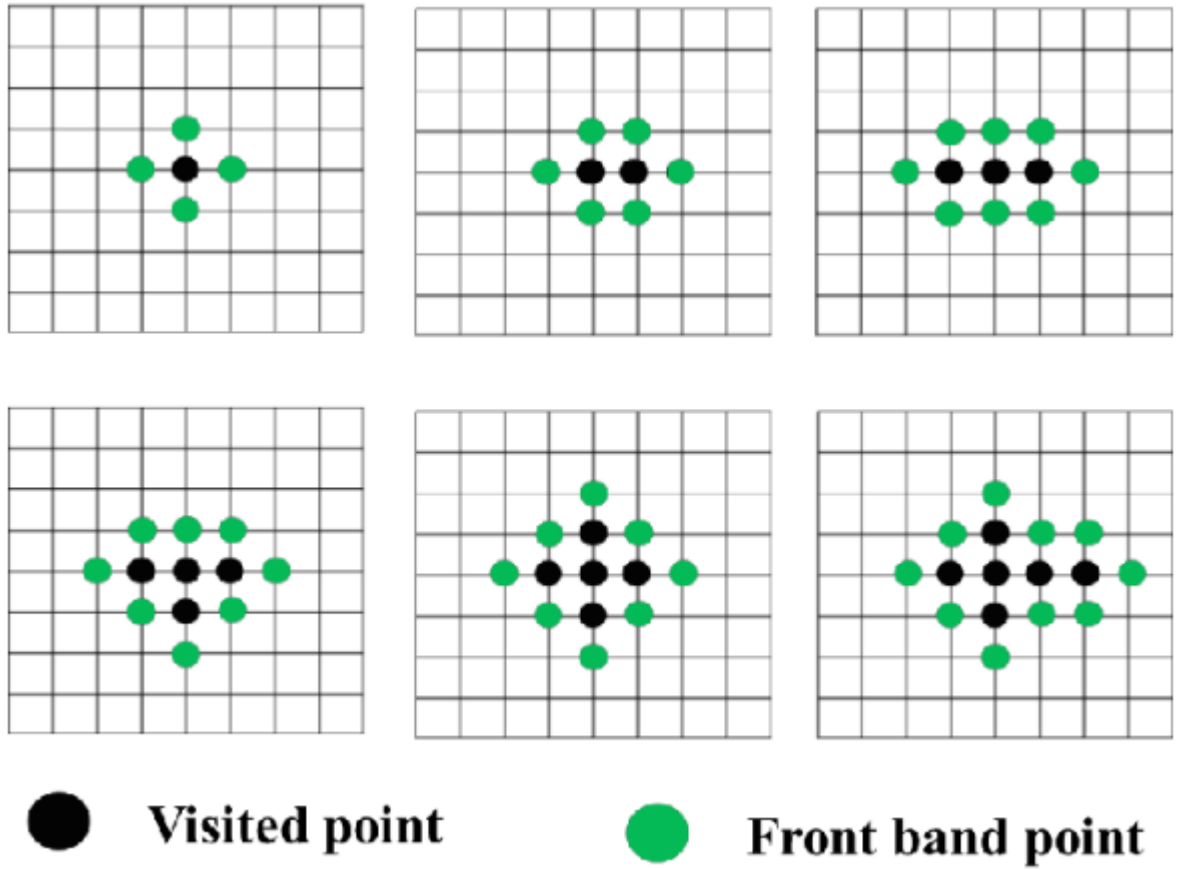
stdin	stdout
8 12 5 6 1 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 1 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 1 1 1 0 1 0 1	7

Hint

- Nếu đã có hình dạng mê cung, dùng thuật toán lan – tìm theo chiều rộng – BFS, lan từ vị trí hiện tại ra xung quanh theo 4 hướng cho tới khi gặp biên nào đầu tiên là được







Hint

- Dùng pair để gộp thông tin tọa độ hàng, cột thành 1 điểm.
- Dùng queue để lưu danh sách điểm sẽ xét đến

Implementation

```
#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];
    //a[maxN][maxN] đánh dấu một ô đã xét hay chưa
    //d[maxN][maxN] lưu độ dài đường đi từ ô xuất phát
int dx[] = {1 , 0, -1 ,  0} ,
    dy[] = {0 , 1,  0 , -1};
queue<ii> qe; //lưu các ô lân cận sẽ được xét
```

Implementation

```
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;
}
```

Implementation

```
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 (Giá trị nhỏ nhất trong khoảng)

- Cho 1 đoạn gồm n số nguyên với giá trị $a_0,\ldots, a_{n-1}$,
  ta định nghĩa rmq(i, j) là giá trị nhỏ nhất trong đoạn từ a_i tới a_j(giá trị số nhỏ nhất trong các số a_i, $a_{i+1}, \ldots, a_j$).

- Ví dụ dãy 10 phần tử 1,5,3,7,8,43,23,5,12,7 thì

  rmq(0,9) = 1
  rmq(1,9) = 3
  rmq(3,5) = 7

- Với đầu vào là *m* đoạn $(i_1, j_1),\ldots, (i_m, j_m)$, giá trị tổng của các rmq định nghĩa trên *m* cặp được tính như sau Q = $rmq(i_1, j_1) + \ldots + rmq(i_m, j_m)$

- Input
  Dòng 1: là số nguyên *n* ($1 <= n <= 10^6$)
  Dòng 2: chứa giá trị các phần tử trong đoạn ban đầu $a_0, \ldots, a_{n-1}$ ($1 <= a_i <= 10^6$)
  Dòng 3: là giá trị *m* ($1 <= m <= 10^6$)
  Các dòng tiếp theo từ k+3 (k = 1, . . ., *m*): là các cặp giá trị i_k, j_k ($0 <= i_k < j_k < n$)

- Output: in ra giá trị Q

3.Range Minimum Query

- Example

stdin	stdout
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	6

Hint

- mảng A là tĩnh, tức là, các phần tử không được chèn hoặc xóa trong một loạt các truy vấn
- xử lý trước mảng phù hợp thành cấu trúc dữ liệu đảm bảo trả lời truy vấn nhanh hơn.
- Một giải pháp đơn giản là tính toán trước tất cả các truy vấn có thể có, tức là tối thiểu của tất cả các mảng con của A và lưu trữ chúng trong một mảng B sao cho B [i, j] = min (A [i… j]);

→ truy vấn phạm vi tối thiểu có thể được giải quyết trong thời gian không đổi bằng cách tra cứu mảng trong B

Có thể có $\Theta(n^2)$ truy vấn cho mảng length-n

Hint

- lưu trữ các truy vấn tối thiểu của đoạn được tính toán trước không phải cho mọi đoạn[i, j], mà chỉ đối với các đoạn có kích thước là lũy thừa của 2.
- Có Θ (log n) truy vấn như vậy cho mỗi vị trí bắt đầu i, do đó kích thước của bảng B là Θ (n log n).
- Giá trị của B [i, j] là chỉ số của giá trị nhỏ nhất trong đoạn A [i… i + 2^j-1].
- Điền vào bảng cần thời gian Θ (n log n), với các chỉ số của giá trị nhỏ nhất tính bằng:
  - Nếu A [B [i, j-1]] ≤ A [B [i + 2^{j-1}, j-1]], thì B [i, j] = B [i, j-1];
  - Ngược lại, B [i, j] = B [i + 2^{j-1}, j-1].
- Sau bước tiền tính toán này, truy vấn RMQ (l, r) giờ đây có thể được trả lời bằng cách tách thành 2 truy vấn riêng biệt:
- truy vấn được tính toán trước với phạm vi từ l đến giá trị được ghi nhớ lớn nhất nhỏ hơn r.
- truy vấn của một khoảng có cùng độ dài có r là ranh giới bên phải của nó.

Implementation

```
#include <bits/stdc++.h>
using namespace std;
int n;
int M[30][1000000];
int A[1000000];
void preprocessing(){
    for(int j = 0; (1 << j) <= n; j++){
        for(int i = 0; i < n; i++) M[j][i] = -1;
    }
    for(int i = 0; i < n; i++) M[0][i] = i;
    for(int j = 1; (1 << j) <= n; j++){
        for(int i = 0; i + (1 << j) - 1 < n; i++){
            if(A[M[j-1][i]] < A[M[j-1][i+(1 << (j-1))]]) M[j][i] = M[j-1][i];
else  M[j][i] = M[j-1][i + (1 << (j-1))];
        }
    }
}
```

Implementation

```
int rmq(int i, int j){
    int k = log2(j-i+1);
    int p2k = (1 << k);//pow(2,k);
    if(A[M[k][i]] <= A[M[k][j-p2k+1]]){
        return M[k][i];
    }else{
        return M[k][j-p2k+1];
    }
}
```

Implementation

```
int main(){
    scanf("%d",&n);
    for(int i = 0; i < n; i++)   scanf("%d",&A[i]);

    preprocessing();

    int ans = 0; int m;
     scanf("%d",&m);
     for(int i = 0; i < m; i++){
         int I,J;  scanf("%d%d",&I,&J);
         ans += A[rmq(I,J)];
     }
     cout << ans;
     return 0;
}
```

4.Largest Black SubRectangle

- Một hình chữ nhật kích thước *n* x *m* được chia thành các ô vuông con 1 x 1 với 2 màu đen hoặc trắng. Hình chữ nhật được biểu diễn bởi ma trận *A*(*n* x *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 <= *n, m* <= 1000)
  - · Dòng i+1 (i = 1,…, *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

stdin	stdout
4 4 0 1 1 1 1 1 1 0 1 1 0 0 1 1 1 0	6