OLP khối không chuyên 2022

Bài 1: Nghệ thuật trừu tượng

Bài này có 1 trường hợp đặc biệt đó là khi n=m=1 thì chỉ có 1 cột và chắc chắn chúng ta luôn thấy cột này.

Còn lại thì đây là một bài tương đối đơn giản. Chúng ta chỉ cần duyệt từng mặt một và kiểm tra từng cột có thể được nhìn thấy ở mặt này hay không bằng cách so sánh độ cao của cột này với độ cao của cột ngay trước nó theo mặt đang được xét mà có thể nhìn thấy được. Nếu nó lớn hơn thì tức là nhìn thấy được và tăng biến đến lên. Mỗi mặt tốn $O(n^2)$ để kiểm tra, do đó kiểm tra từng mặt sẽ tốn $4 * O(n^2)$

Tuy nhiên cần phải xử lý khéo sao cho mỗi cột không bị chọn nhiều lần do nó có thể được nhìn từ nhiều phía.

Các bạn có thể tham khảo cách giải của mình (mặc dù không chắc đây có phải cách tối ưu hay không):

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

using namespace std;

int main(){
    ios :: sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    int n,m,ans;
    cin>>n>>m;
    ans=(n+m)*2-4;
    if(n==1&&m==1){
        cout<<1;
        return 0;
    }
    int a[n][m];
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            cin>>a[i][j];
        }
```

```
    }
    int x[4][n][m];
    map<int,pair<int,int>>z;
    z[0]={0,-1};
    z[1]={0,1};
    z[2]={-1,0};
    z[3]={1,0};
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            if(j==0){
                x[0][i][j]=a[i][j];
                if(a[i][j]==0){
                    ans--;
                }
            }
            else{
                x[0][i][j]=max(x[0][i][j-1], a[i][j]);
            }
        }
        for(int j=m-1;j>=0;j--){
            if(j==m-1){
                if(a[i][j]==0){
                    ans--;
                }
                x[1][i][j]=a[i][j];
            }
            else{
                x[1][i][j]=max(x[1][i][j+1], a[i][j]);
            }
        }
    }
    for(int j=0;j<m;j++){
        for(int i=0;i<n;i++){
            if(i==0){
                if(a[i][j]==0&&j!=0&&j!=m-1){
                    ans--;
                }
                x[2][i][j]=a[i][j];
```

```
            }
            else{
                x[2][i][j]=max(x[2][i-1][j], a[i][j]);
            }
        }
        for(int i=n-1;i>=0;i--){
            if(i==n-1){
                if(a[i][j]==0&&j!=0&&j!=m-1){
                    ans--;
                }
                x[3][i][j]=a[i][j];
            }
            else{
                x[3][i][j]=max(x[3][i+1][j], a[i][j]);
            }
        }
    }
    for(int i=1;i<n-1;i++){
        for(int j=1;j<m-1;j++){
            for(int k=0;k<4;k++){
                if(x[k][i][j]>x[k][i+z[k].first][j+z[k].secon
                    ans++;
                    break;
                }
            }
        }
    }
    cout<<ans;
}
```

Bài 2: Cắt dán

Đây là một bài hình học, tuy nhiên tư tưởng không quá khó. Chỉ cần sử dụng phương pháp duyệt đơn giản. Hai hình đa giác A và B được coi là tương đồng nhau nếu với mỗi cạnh của A luôn có cạnh tương ứng trên B với độ dài bằng nhau và chưa được ghép với cạnh khác. Ngoài ra hai đường chéo cũng vậy.

Nói cách khác, nếu bạn tìm tất cả các khoảng cách giữa hai điểm bất kì trên A và sắp xếp lại theo thứ tự từ nhỏ đến lớn và làm y hệt với B thì A = B khi và chỉ khi hai mảng khoảng cách là giống nhau.

Code tham khảo:

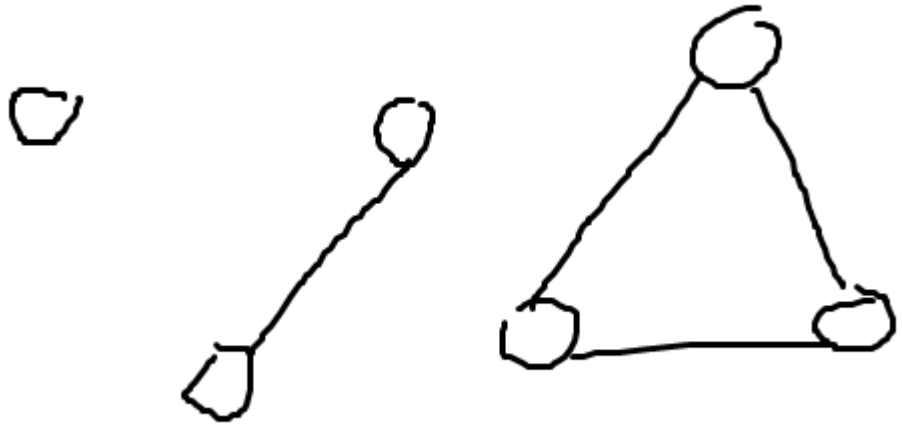
```
#include <bits/stdc++.h>

using namespace std;

struct point{
    double a,b;
};

double distance(point x, point y){
    return sqrt((x.a-y.a)*(x.a-y.a)+(x.b-y.b)*(x.b-y.b));
}

void gendis(vector<double>&dis, point x[]){
    for(int i=0;i<3;i++){
        for(int j=i+1;j<4;j++){
            dis.push_back(distance(x[i],x[j]));
        }
    }
}
int main(){
    ios :: sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    point c[4];
    int ans=0;
    for(int i=0;i<4;i++){
        cin>>c[i].a>>c[i].b;
    }
    vector<double>dis;
    gendis(dis, c);
    sort(dis.begin(),dis.end());
    int t;
    cin>>t;
    while(t--){
```

```
        bool ck=false;
        point x[4];
        for(int i=0;i<4;i++){
            cin>>x[i].a>>x[i].b;
        }
        vector<double>checkdis;
        gendis(checkdis, x);
        sort(checkdis.begin(),checkdis.end());
        for(int i=0;i<6;i++){
            if(dis[i]!=checkdis[i]){
                ck=true;
            }
        }
        if(ck==false){
            ans++;
        }
    }
    cout<<ans;
}
```

Bài 3: Quà Noel

Đây là bài khó nhất đề, và các bạn sẽ bất ngờ khi biết rằng đây là một bài đồ thị.

Giả sử có một đồ thị n đỉnh tương đương n món quà. Nếu một cháu thư i muốn quà u và v thì ta tạo một cạnh nối đỉnh u và đỉnh v, khi đấy ta có các trường hợp:



- Xét trường hợp đầu tiên chính là đỉnh đứng một mình và không có cạnh đến các đỉnh khác. Điều này có nghĩa rằng không có bạn nào muốn nhận món quà tương ứng với đỉnh này.
- Trường hợp thứ hai là trường hợp với 2 đỉnh và 1 cạnh nối duy nhất (2 đỉnh này không nối với các đỉnh khác). Điều này tương đương với có một bạn và chỉ một muốn nhận một trong 2 món quà này. Và vì mỗi người chỉ nhận 1 món quà nên chỉ 1 trong 2 được trao đi.
- Trường hợp cuối cùng là các đỉnh nối với nhau như trong đồ thị dạng tam giác như ở trên. Có 3 đường nối tương đương với 3 bạn muốn 2 trong 3 món quà này. Và ta có thể suy luận rằng hoàn toàn có thể chia 3 món quà này cho 3 bạn sao cho ai cũng có quà.

Vậy từ 3 trường hợp trên chúng ta có một nhận xét như sau:

- TH1: chỉ có 1 đỉnh và 0 cạnh kề → 0 quà
- TH2: 2 đỉnh 1 cạnh kề → 1 quà
- TH3: 3 đỉnh và 3 cạnh kề (Đặt 3 đỉnh là 1, 2, 3 ta có 1→2, 1→3, 2→3) → 3 quà

Vậy ta đưa đến kết luận rằng kết quả của bài toán chính là số cạnh nhiều nhất có thể chọn sao cho chỉ thăm mỗi đỉnh một lần.

Để cài đặt bài này chỉ cần sử dụng thuật toán BFS hoặc DFS để duyệt qua đồ thị. Với mỗi đồ thị H (giả định rằng có nhiều đồ thị không liên kết với nhau) bao gồm có Cv đỉnh và Ce cạnh:

● Nếu H có đúng Cv – 1 cạnh: có Cv – 1 cạnh được chọn
● Nếu H có hơn Cv – 1 cạnh: có Cv cạnh được chọn

Cần đếm số đỉnh Cv và số cạnh Ce của từng đồ thị.

Code tham khảo:

```
#include <bits/stdc++.h>
using namespace std;

const int maxN = 100010;
vector <int> g[maxN];
int cv, ce, visited[maxN];

void dfs(int u){
    visited[u] = 1; cv++;
    for (int v: g[u]){
        ce++;
        if (visited[v] == 0)
            dfs(v);
    }
}

int main(){
int n, m, res = 0;
cin >> n >> m;
while (m--) {
    int u, v;
    cin >> u >> v;
    g[u].push_back(v);
    g[v].push_back(u);
}
for (int u = 1; u <= n; u++){
    if (visited[u]==0){
        ce = 0; cv = 0;
        dfs(u);
        ce /= 2;
        res += ce < cv ? cv - 1 : cv;
        }
}
```

```
cout << res;
}
```

Bài 4: Cửa hàng năng lượng thông minh

Để không bị TLE bài này, thì các bạn cần phải sử dụng một kỹ thuật là sàng. Với kỹ thuật này thì trước khi trả lời các truy vấn thì các bạn cần phải lọc hết các số thỏa mãn yêu cầu của đề bài. Ví dụ với bài này yêu cầu thống kê có bao nhiêu viên pin có mức năng lượng là số lý tưởng - các số có dạng lũy thừa của 2, của 3 hoặc của 5 thì sàng ở đây được sử dụng để lọc các số từ 1 → 1e9 mà chia hết cho 2, 3 và 5 ra một mảng.

Sau đó với mỗi giá trị mức năng lượng thì có thể sử dụng để kiểm tra xem giá trị có nằm trong mảng được sàng hay không. Quá trình này có thể sử dụng chặt nhị phân để giảm độ phức tạp xuống. Nếu có thì đây là giá trị lý tưởng. Ở đây ta tiếp tục sử dụng sàng để lọc ra các viên pin lý tưởng ra một mảng khác.

Cuối cùng với mỗi truy vấn ta chỉ cần tìm lower bound của low và upper bound của high và kết quả của bài toán sẽ là:

$$ans = upper\_bound(high) - lower\_bound(low) + 1$$

Code tham khảo:

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

using namespace std;

int limit = 1e9;
int main(){
    ios :: sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    int n,m;
    cin>>n>>m;
    vector<int>a;
    vector<int>check={1};
    for(int i=2;i<=limit;i*=2){
        check.push_back(i);
    }
```

```
    for(int i=3;i<=limit;i*=3){
        check.push_back(i);
    }
    for(int i=5;i<=limit;i*=5){
        check.push_back(i);
    }
    sort(check.begin(),check.end());
    for(int i=0;i<n;i++){
        int x;
        cin>>x;
        if(binary_search(check.begin(),check.end(),x)==true){
            a.push_back(x);
        }
    }
    sort(a.begin(),a.end());
    while(m--){
        int lo,hi;
        cin>>lo>>hi;
        auto low = lower_bound(a.begin(),a.end(),lo);
        auto high = upper_bound(a.begin(),a.end(),hi);
        high--;
        if(low==a.end()||*high>hi){
            cout<<0<<"\n";
        }
        else{
            cout<<high-low+1<<"\n";
        }
    }
}
```