TỔ CHỨC THE GIFTED BATTLEFIELD
DỰ ÁN INTELLECTUAL SHOWDOWN

Educational Contest 2 - DFS Tree
NĂM HỌC 2023 – 2024

Ngày thi: **24/12/2023**
Thời gian làm bài: **7 ngày**
Hướng dẫn giải gồm 19 trang, 08 bài

Tổng quan đề thi

	Tên bài	Hạn chế thời gian	Hạn chế bộ nhớ
Bài 1	Loại bỏ	2 giây	512 MB
Bài 2	Chuyện trong phòng thi	2 giây	512 MB
Bài 3	Hệ thống	2 giây	512 MB
Bài 4	Tập thể dục	2 giây	512 MB
Bài 5	Giảm thiểu chi phí	2 giây	512 MB
Bài 6	Cuộc thi hát	2 giây	512 MB
Bài 7	Tùy bạn	2 giây	512 MB
Bài 8	Đồ thị lộn xộn	2 giây	512 MB

Nhập xuất theo kiểu chuẩn - standardIO, không sử dụng nhập xuất file.

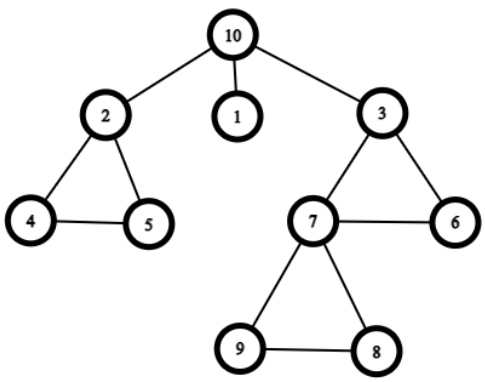
I. Hướng dẫn chung

1. Bài thi của thí sinh được chấm thi bằng phần mềm chấm thi trực tuyến (online judge) trên website https://oj.giftedbat.edu.vn/, sử dụng bộ test của Tổ chức The Gifted Battlefield – Ban Tin học, đúng với đáp án và biểu điểm được nêu trong hướng dẫn chấm thi.
2. Điểm bài thi được xuất từ phần mềm chấm thi.

Bài 1. Loại bỏ

Ý tưởng

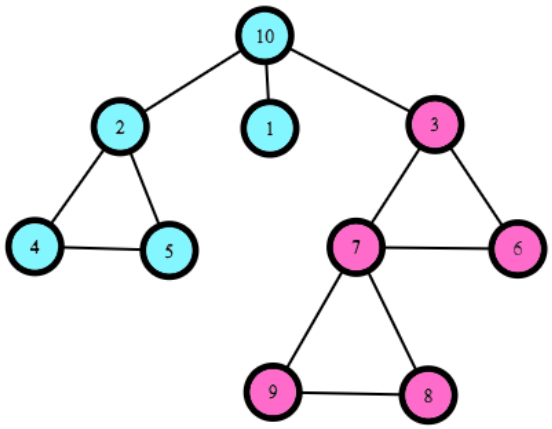
- Ta xem đồ thị như một cây DFS (DFS Tree)[1] trong bài này:
  - Các cạnh thuộc cây DFS được gọi là các "cạnh nét liền".
  - Các cạnh còn lại không thuộc cây DFS được gọi là các "cạnh nét đứt", được tạo ra sau khi quá trình DFS kết thúc và quay về đỉnh ban đầu.
- Cây DFS của test mẫu có hình dạng như sau:



- Các mảng sử dụng trong cây DFS:
  - $num[]$: Thứ tự duyệt DFS của các đỉnh.
  - $low[]$: $low[u]$ cho biết thứ tự nhỏ nhất khi duyệt DFS trên cây con gốc u khi đi xuống theo các cạnh nét liền và đi ngược lên bằng cạnh nét đứt không quá một lần.
  - $tail[]$: Cho biết thời gian kết thúc duyệt DFS của đỉnh đó.

Tìm cầu

- Ta có thể dễ dàng nhận thấy rằng tính liên thông không thể bị ảnh hưởng bởi những cạnh nét đứt (những cạnh không thuộc cây DFS này). Vì thế, các cạnh cầu **không thể là cạnh nét đứt** mà là **các cạnh nét liền**.
- Xét riêng từng thành phần liên thông của đồ thị. Ta có thể xét cây DFS của vùng liên thông G như sau:
  - Với cây con gốc v ta có cha trực tiếp của v là u, ta có cạnh (u, v) có tính chất như sau:
    + Cho A là tập hợp các đỉnh thuộc cây con gốc v, B là tập hợp các đỉnh không thuộc cây con gốc v.
    + Khi đó, sau khi xóa đi cạnh (u, v) thì các đỉnh thuộc tập hợp A vẫn có thể đi trực tiếp đến nhau bằng các cạnh đã cho.
    + Nhưng để các đỉnh thuộc tập hợp B muốn đi qua A hoặc ngược lại thì cần phải qua cạnh (u, v).
  - **Ví dụ**: Xét cạnh nét liền $(10, 3)$, ta có tập A là các đỉnh màu hồng, còn tập B là các đỉnh màu xanh.
    + Nếu loại bỏ cạnh $(10, 3)$, thì các đỉnh màu hồng không thể đi tới các đỉnh màu xanh (hoặc ngược lại) bằng những cạnh nét liền còn lại. Ví dụ như từ đỉnh 7 không thể đi đến đỉnh 2 sau khi xóa cạnh.
    + Như vậy, để các đỉnh trong tập A đi qua các đỉnh trong tập B ta phải thông qua cạnh $(10, 3)$.



  - Vậy trong trường hợp không có cạnh giữa 2 tập đỉnh A và tập đỉnh B ngoài cạnh (u, v) thì khi xóa đi cạnh (u, v), G sẽ tách ra thành 2 vùng liên thông A và B. Nhưng nếu tồn tại ít nhất một cạnh nét đứt giữa hai đỉnh thuộc hai tập này, đồ thị vẫn liên thông sau khi xóa cạnh (u, v).
- Từ đó, ta có thể quy bài toán này về câu hỏi **"có tồn tại cạnh nét đứt giữa hai đỉnh thuộc tập đỉnh A và B hay không?"**:
  - Ta có thể đi từ đỉnh v đến đỉnh y có $num[y] = low[v]$ khi đi theo các đường nét liền của cây và đi qua không quá một cạnh nét đứt, với y có thứ tự thăm sớm nhất khi DFS từ v.
  - Nếu y nằm trong tập đỉnh B (tập đỉnh ngoài cây con v) thì y là tổ tiên của v. Hay nói cách khác chắc chắn $num[y] < num[v]$ hoặc $low[v] < num[v]$, điều này có nghĩa tồn tại cạnh nét đứt giữa 1 đỉnh thuộc A và 1 đỉnh thuộc B vì từ một đỉnh không thể đi tới một tổ tiên của nó mà không đi qua các cạnh nét đứt.
  - Vì vậy, phủ định quy luật trên, ta có $low[v] \geq num[v]$ chắc chắn y thuộc tập đỉnh A khi đó không tồn tại cạnh nét đứt nối giữa 1 đỉnh thuộc A với 1 đỉnh thuộc B.

– Nhưng ta có thể thấy rằng $low[v] \leq num[v]$ vì đỉnh v luôn tới được chính nó.

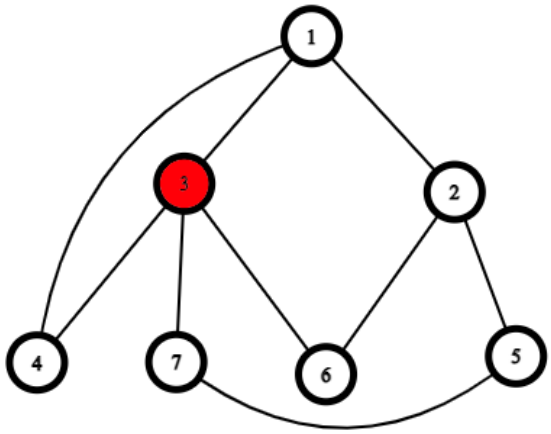
- **Kết luận**: Nếu $low[v] = num[v]$ thì (u, v) là một cạnh cầu trong đồ thị.

Tìm khớp

- Ta sẽ xét riêng từng thành phần liên thông của đồ thị. Xét vùng liên thông G như sau:

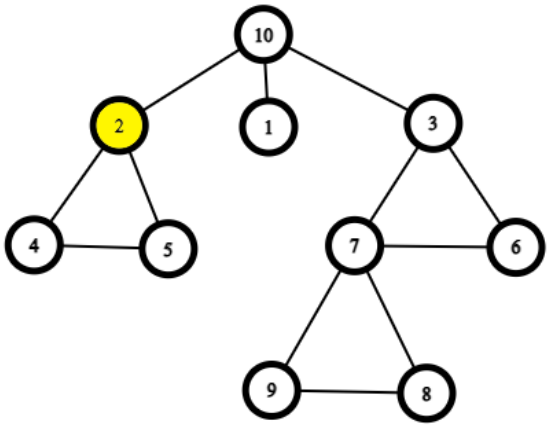
  – Ta có thể dễ dàng kết luận rằng nếu tồn tại ít nhất một đỉnh v là con trực tiếp của cây con gốc u có cung đi ngược lên tổ tiên của u, thì khi loại bỏ u, có thể đồ thị vẫn có tính liên thông vì từ tổ tiên của u đó, có thể đi sang v và qua một vài hay thậm chí tất cả đỉnh có u là tổ tiên, từ đó giữ nguyên số thành phần liên thông của đồ thị. Ngược lại, nếu mọi nhánh con của cây con gốc u đều có cung đi ngược lên đến tổ tiên của u ($low[v] < num[u]$, v là tất cả con trực tiếp của u trên cây DFS) thì chắc chắn đỉnh u không phải là khớp.

    + **Ví dụ**: Đỉnh 3 không phải là khớp vì tất cả 3 nhánh con của nó $(4, 7, 6)$ đều có cung ngược lên tổ tiên của 3.



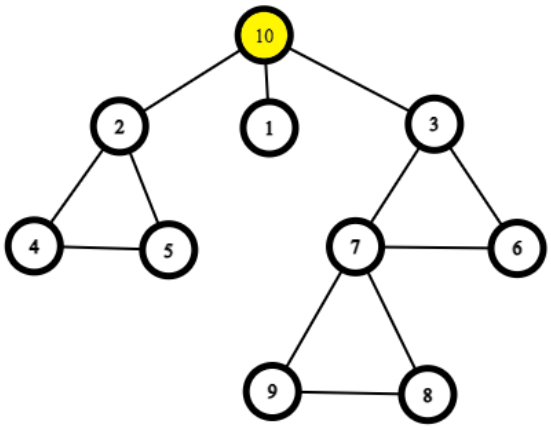
  – Trong trường hợp u không phải là đỉnh gốc của cây DFS: nếu tồn tại một v sao cho $low[v] \geq num[u]$ (với v là một con trực tiếp bất kì của u trên cây DFS) thì đỉnh u là khớp. Điều kiện $low[v] \geq num[u]$ mang ý nghĩa: trong cây con gốc v (v là con trực tiếp của u), không có đỉnh nào có thể đi bằng cung ngược để thăm được các tổ tiên của $u \Rightarrow$ Khi loại bỏ đỉnh u thì cây con gốc v trở thành một miền liên thông của riêng nó mà không có đường đi đến bất kì đỉnh nào khác trong đồ thị.

    + **Ví dụ**: Đỉnh 2 là đỉnh khớp vì thỏa hai điều kiện: không phải là đỉnh gốc cây DFS và các đỉnh con không có cung nối lên tổ tiên của 2.



  – Nếu u là đỉnh gốc của cây DFS, thì u là đỉnh khớp khi và chỉ khi u có ít nhất 2 nhánh con. Khi u có 2 nhánh con thì đường đi giữa hai đỉnh thuộc hai nhánh con đó bắt buộc phải đi qua u (vì tính chất của cây DFS, sẽ không tồn tại cạnh nét đứt nối giữa hai nhánh con của một đỉnh). Vì thế nên việc loại bỏ đỉnh u sẽ làm tăng số thành phần liên thông của đồ thị.

    + **Ví dụ**: Đỉnh 10 là đỉnh khớp vì đỉnh 10 là đỉnh gốc của cây DFS và có tới 3 nhánh con.



– **Kết luận**: Đỉnh u là đỉnh khớp khi:

+ Đỉnh u không phải là gốc của cây DFS và $low[v] \geq num[u]$ (với v là một con trực tiếp bất kì của u trong cây DFS). **Hoặc**
+ Đỉnh u là gốc của cây DFS và có ít nhất 2 con trực tiếp trong cây DFS.

Code mẫu

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

using namespace std;

const int N = 10010;

int n, m;
bool isKhop[N]; // Đánh dấu đỉnh khớp
int timeDFS = 0, cau = 0;
int low[N], num[N];
vector<int> g[N];

void DFS(int u, int pre) {
    int child = 0; // Đếm lượng con trực tiếp của đỉnh u trong cây DFS
    num[u] = low[u] = timeDFS++; // num[u]=low[u] vì u luôn có thể đi tới chính nó
    for (int v : g[u]) { // Xét mọi nhánh con của u
        if (v == pre) continue;
        if (!num[v]) {
            DFS(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] == num[v]) cau++;
            child++;
            if (u == pre) { // Trường hợp u là đỉnh gốc của cây DFS
                if (child > 1) isKhop[u] = true;
            }
            else if (low[v] >= num[u]) isKhop[u] = true;
        }
        else low[u] = min(low[u], num[v]);
    }
}

int main() {
    cin >> n >> m;
    for (int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    for (int i = 1; i <= n; i++)
        if (!num[i]) DFS(i, i);

    int khop = 0;
```

```
    for (int i = 1; i <= n; i++) khop += isKhop[i];

    cout << khop << ' ' << cau;
}
```

Độ phức tạp: $O(N + M)$.

Bài 2. Chuyện trong phòng thi

- Trước hết ta sẽ sort lại danh sách cạnh theo w tăng dần và chặt nhị phân điểm cắt mid: với những cạnh có trọng số w sao cho $w \leq mid$ thì chúng ta sẽ thêm vào đồ thị.
- Bây giờ bài toán chúng ta cần giải sẽ là cho một đồ thị có hướng, hãy xác định xem có tồn tại một đường đi từ $u_i \to v_i$ hay không.
- Để có thể giải bài toán này ta sẽ nén đồ thị thành các thành phần liên thông mạnh để dễ xử lí.
- Ta có mảng bool $reachable_{i,j} = true$ nếu chúng ta có thể có thể đến thành phần liên thông thứ j khi bắt đầu đi ở thành phần liên thông thứ i (kể cả chính nó).
- Để có thể xây mảng $reachable$ ta sẽ duyệt các thành phần liên thông mạnh theo thứ tự toposort đảo ngược, nếu có cạnh nối giữa hai thành phần liên thông mạnh i, j thì $reachable_i = reachable_i \ OR \ reachable_j$.
- Đáp án sau khi chúng ta chặt nhị phân sẽ là w_{ans}.
- Lúc này độ phức tạp của chúng ta sẽ là $O((n^2 + k) \times \log M)$ vì có $\log M$ lần kiểm tra với chi phí cho mỗi lần là $n^2 + k$.
- Để có thể tối ưu hơn nữa ta sẽ lưu n mảng $reachable$ dưới dạng bitset[2].

Code mẫu

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

const int MaxN = 2e4;
const int MaxM = 1e5;

bitset<MaxN + 1> reachable[MaxN + 1];

struct SCC{
    int n;
    vector<vector<int>> adj,adjRev,sAdj;
    vector<int> order,comp;
    vector<bool> visited;
    SCC(int n) :n(n){
        adj.resize(n + 1);
        adjRev.resize(n + 1);
        comp = vector<int>(n + 1,0);
        visited = vector<bool>(n + 1,0);
    }
    void addEdge(int a,int b){
        adj[a].push_back(b);
        adjRev[b].push_back(a);
    }
    void topoSorting(int u){
        visited[u] = true;
        for(int v : adj[u]){
            if(!visited[v]){
                topoSorting(v);
            }
        }
        order.push_back(u);
    }
```

```
    void dfsSCC(int u,int cnt){
        comp[u] = cnt;
        for(int v : adjRev[u]){
            if(comp[v] == 0){
                dfsSCC(v,cnt);
            }
        }
    }
    void buildSCC(){
        for(int i = 1;i <= n;++i){
            if(!visited[i]){
                topoSorting(i);
            }
        }
        reverse(order.begin(),order.end());
        int cnt = 0;
        for(int u : order){
            if(comp[u] == 0){
                dfsSCC(u,++cnt);
            }
        }
        sAdj.resize(cnt + 1);
        for(int i = 1;i <= n;++i){
            for(int v : adj[i]){
                if(comp[i] != comp[v]){
                    sAdj[comp[i]].push_back(comp[v]);
                }
            }
        }
    }
    void prepare(){
        reverse(order.begin(),order.end());
        for(int u : order){
            reachable[comp[u]].set(comp[u]);
            for(int v : adj[u]){
                if(comp[u] != comp[v]){
                    reachable[comp[u]] |= reachable[comp[v]];
                }
            }
        }
    }
    bool isReachable(int u,int v){
        return reachable[comp[u]].test(comp[v]);
    }
};
struct Edge{
    int u,v,w;
    bool operator < (const Edge &other){
        return  (w < other.w);
    }
};

int n,m,k;
pair<int,int> queries[MaxM + 1];
Edge edgelist[MaxM + 1];

void readData(){
    cin >> n >> m >> k;
    for(int i = 1;i <= m;++i){
        cin >> edgelist[i].u >> edgelist[i].v >> edgelist[i].w;
    }
    for(int i = 1;i <= k;++i){
        cin >> queries[i].first >> queries[i].second;
    }
    sort(edgelist + 1,edgelist + m + 1);
}
bool check(int it){
```

```
    SCC graph(n);
    for(int i = 1;i <= it;++i){
        graph.addEdge(edgelist[i].u,edgelist[i].v);
    }
    graph.buildSCC();
    graph.prepare();
    for(int i = 1;i <= k;++i){
        if(!graph.isReachable(queries[i].first,queries[i].second)){
            return false;
        }
    }
    return true;
}
int solve(){
    int lo = 1,hi = m,ans = m + 1;
    while(lo <= hi){
        int mid = (lo + hi) / 2;
        if(check(mid)){
            ans = mid;
            hi = mid - 1;
        }else{
            lo = mid + 1;
        }
    }
    if(ans == m + 1){
        return -1;
    }
    return edgelist[ans].w;
}
int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    readData();
    cout << solve();
    return 0;
}
```

Độ phức tạp: $O((\frac{n^2}{32} + k) \times \log M)$.

Bài 3. Hệ thống

- Ta nhận xét rằng đoạn xung yếu đề bài nói tới thực chất là một **cầu** của đồ thị. Định nghĩa về cầu có thể xem ở đây: VNOI Wiki - DFS Tree[1]. Như vậy, bài toán trở thành: Cần thêm ít nhất bao nhiêu cạnh để đồ thị ban đầu của chúng ta không còn **cầu**.
- Khi này ta có thể nén đồ thị ban đầu lại thành một cây: Với mỗi đỉnh là một miền liên thông không tồn tại cầu, và mỗi cạnh là một cầu. Kĩ thuật này còn được biết đến với tên gọi là **Bridge Tree**.
- Sau khi ta xây dựng được Bridge Tree, ta có bài toán : cần thêm bao nhiêu cạnh vào một cây để cây không còn cầu. Ta có thể sử dụng thuật toán tham lam và thêm các cạnh giữa các đỉnh lá. Gọi số lượng lá là m. Dễ thấy số đáp án bài toán là $(m + 1)/2$.

Code mẫu

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

using namespace std;

vector< vector<int> > adj, adj2;
vector<int> low, in, par;
int dfs_time, tree_node;

void dfs_find(int u) {
    low[u] = in[u] = ++dfs_time;
```

```
    for(int v: adj[u]) if (v != par[u]) {
        if (!in[v]) {
            par[v] = u;
            dfs_find(v);
            low[u] = min(low[u], low[v]);
        }
        else low[u] = min(low[u], in[v]);
    }
}

void dfs_build(int u, int node) {
    for(int v: adj[u]) {
        if (par[v] == u) {
            if (low[v] > in[u]) {
                ++tree_node;
                adj2[node].push_back(tree_node);
                adj2[tree_node].push_back(node);
                dfs_build(v, tree_node);
            }
            else dfs_build(v, node);
        }
    }
}

array<int, 2> dfs(int u, int par) {
    array<int, 2> ans = {u, 0};
    for(int v: adj2[u]) if (v != par) {
        array<int, 2> tmp = dfs(v, u);
        if (tmp[1] > ans[1]) ans = tmp;
    }
    return {ans[0], ans[1] + 1};
}

void solve() {
    int n, m;
    cin >> n >> m;

    dfs_time = 0;
    adj.assign(n + 1, vector<int>());
    adj2.assign(n + 1, vector<int>());
    low.assign(n + 1, 0);
    in.assign(n + 1, 0);
    par.assign(n + 1, 0);
    tree_node = 1;

    while(m--) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    dfs_find(1);
    dfs_build(1, tree_node);

    int leaf = 0;
    for(int i = 1; i <= tree_node; ++i) {
        leaf += (adj2[i].size() == 1);
    }

    cout << (leaf + 1) / 2;
}

signed main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    solve();
```

```
    return 0;
}
```

Độ phức tạp: $O(n)$.

Bài 4. Tập thể dục

- "*Huy* chọn một số ngôi làng để đi bộ qua sao cho nếu bỏ bất kỳ ngôi làng nào ra thì *Huy* vẫn có thể đi qua các ngôi làng còn lại được bằng những đường nối các làng trong số đó".
- Ta có thể thấy, với các ngôi làng thỏa mãn con đường đi tập thể dục của *Thanh Huy*, tạo thành một đồ thị song liên thông, vậy ta phát biểu lại bài toán:
- Nối một cạnh lại để số đỉnh trong miền song liên thông lớn nhất là lớn nhất.

Ý tưởng

- Đây là một ứng dụng của **Block cut tree**. Bạn đọc có thể tìm hiểu thêm ở đây: Khái niệm Block cut tree[3].
- Sau khi biến đổi đồ thị gốc thành BCT, bài toán có nhiều cách giải, có thể kể đến như:
  - Với các đỉnh gốc ban đầu, gán trọng số cho đỉnh đó là -1.
  - Với các đỉnh ảo tượng trưng cho các miền song liên thông, gán trọng số đỉnh đó là độ lớn của miền song liên thông.

    $\rightarrow$ Bài toán được biến đổi thành tìm đường kính của cây có trọng số.
  - Phần còn lại để lại cho bạn đọc tự nghĩ.

Code mẫu

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

using namespace std;
const int maxn = 1e6 + 5;
int n, m, cnt, ans, ncc, f;
int num[maxn], low[maxn], node[maxn], val[maxn], h[maxn];
pair<int, int> mx[maxn], edge;
vector<int> adj[maxn], bcc[maxn];
stack<pair<int, int>> st;

inline void add(int u, int v)
{
    adj[u].emplace_back(v);
    adj[v].emplace_back(u);
}

void read()
{
    cin >> n >> m;
    for (int i = 1, x, y; i <= m; i++)
        cin >> x >> y,
        add(x, y);
}

void dfs(int x, int p)
{
    num[x] = low[x] = cnt++;
    for (int i : adj[x])
    {
        if (i == p) continue;
        if (!num[i])
        {
```

```
            st.push({x, i});
            dfs(i, x);
            low[x] = min(low[x], low[i]);
            if (low[i] >= num[x])
            {
                ncc++;
                bcc[ncc].emplace_back(x);
                pair<int, int> tmp, cmp = {x, i};
                while (tmp != cmp)
                {
                    tmp = st.top();
                    st.pop();
                    bcc[ncc].emplace_back(tmp.second);
                }
                ans = max(ans, (int)bcc[ncc].size());
            }
        }
        else low[x] = min(low[x], num[i]);
    }
}

void get_ans(int val, int x, int y)
{
    if (ans < val)
    {
        ans = val;
        edge = {x, y};
    }
}

void dfs3(int x = 1, int p = 1)
{
    h[x] = h[p] + 1;
    for (int i : adj[x]) if (i != p) dfs3(i, x);
}

void dfs2(int x = 1, int p = 1)
{
    mx[x] = {val[x], node[x]};
    vector<pair<int, int>> v;
    for (int i : adj[x]) if (i != p)
    {
        dfs2(i, x);
        v.emplace_back(mx[i]);
    }
    sort(v.begin(), v.end(), greater<pair<int, int>>());
    if (v.size()) get_ans(val[x] + v[0].first, (x <= n ? x : bcc[x - n].back()), v[0].
    second);
    if (v.size() >= 2) get_ans(val[x] + v[0].first + v[1].first, v[0].second, v[1].second);
    if (v.size() && v[0].first > 0)
        mx[x] = {val[x] + v[0].first, v[0].second};
}

bool cmp(int a, int b)
{
    return h[a] > h[b];
}

void solve()
{
    for (int i = 1; i <= n; i++)
        if (!num[i]) dfs(i, i);
    for (int i = 1; i <= n; i++)
        adj[i].clear();
    f = 1;
    for (int i = 1; i <= ncc; i++)
        for (int j : bcc[i])
```

```
            add(i + n, j);
    dfs3();
    for (int i = 1; i <= n; i++)
        val[i] = -1,
        node[i] = i;
    for (int i = n + 1; i <= n + ncc; i++)
    {
        val[i] = bcc[i - n].size();
        sort(bcc[i - n].begin(), bcc[i - n].end(), cmp);
        node[i] = bcc[i - n].front();
    }
    edge = {1, 1};
    dfs2();
    cout << edge.first << " " << edge.second;
}

signed main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
//    freopen("test.inp", "r", stdin);
//    freopen("test.out", "w", stdout);
    read();
    solve();
}
```

Độ phức tạp: $O(n \times \log n)$.

Bài 5. Giảm thiểu chi phí

Lời giải

- Để dễ hình dung, ta coi thành phố của ta như một đồ thị, với mỗi thành phố là một đỉnh và mỗi con đường là một cạnh.
- Ta nhận xét rằng một con đường quan trọng chính là một **cầu** của đồ thị. Định nghĩa về cầu có thể xem ở đây: VNOI Wiki - DFS Tree[1].
- Khi này ta có thể nén đồ thị ban đầu lại thành một cây: Với mỗi đỉnh là một miền liên thông không tồn tại cầu, và mỗi cạnh là một cầu. Kĩ thuật này còn được biết đến với tên gọi là **Bridge Tree**.
- Sau khi ta xây dựng được Bridge Tree, ta nhận xét rằng sẽ có một vài đỉnh trong cây này không cần thiết khi giải bài toán với k đỉnh đề cho. Và vì ta cần tìm đáp án tối ưu, ta sẽ loại bỏ những đỉnh này.
- Vì một đỉnh trong cây mới này thể hiện cho một tập các đỉnh của đồ thị ban đầu, nên ta có thể định nghĩa một đỉnh **trọng yếu** trên cây là một đỉnh sao cho bên trong nó có chứa một trong số k đỉnh cần xét.
- Gọi **Final Tree** là cây cuối cùng sau khi đã loại bỏ những đỉnh không cần thiết.
- Khi này nếu ta đặt gốc của cây mới tại một trong số các đỉnh **trọng yếu**, thì điều kiện cần và đủ để một đỉnh thuộc Final Tree là trong cây con của nó tồn tại một đỉnh **trọng yếu**.
- Gọi d = đường kính (diameter) của cây Final Tree.
- Đáp án sẽ chính bằng số cạnh trong cây Final Tree trừ cho d.

Code mẫu

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

const int N = 1e5 + 5;

```

```cpp
vector<int> adj[N];
int low[N], num[N], color[N];
stack<int> st;

bool init[N], need[N];
int timer = 0, node = 0;

void dfs(int u = 1, int p = 0){ // dựng cây Bridge Tree
    st.push(u);
    low[u] = num[u] = ++timer;
    for(int v : adj[u]){
        if(v == p) continue;
        if(!num[v]){
            dfs(v, u);
            low[u] = min(low[u], low[v]);
        }
        else low[u] = min(low[u], num[v]);
    }
    if(low[u] > num[p]){
        ++node;
        while(1){
            int v = st.top(); st.pop();
            color[v] = node;
            need[node] |= init[v];
            if(v == u) break;
        }
    }
}

vector<int> adj2[N];
bool ok[N]; // kiểm tra đỉnh có thuộc Final Tree hay không

void dfs2(int u, int p = -1){ // dựng cây Final Tree
    if(need[u]) ok[u] = 1;
    for(int v : adj2[u]){
        if(v != p){
            dfs2(v, u);
            ok[u] |= ok[v];
        }
    }
}

int edge = 0, diameter = 0;

int dfs3(int u, int p = -1){ // tìm bán kính của cây Final Tree
    int mx1 = 0, mx2 = 0;
    for(int v : adj2[u]){
        if(v != p && ok[v]){
            int cur = dfs3(v, u);
            if(cur >= mx1){
                mx2 = mx1;
                mx1 = cur;
            }
            else mx2 = max(mx2, cur);
        }
    }
    diameter = max(diameter, mx1 + mx2);
    return mx1 + 1;
}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n, m, k; cin >> n >> m >> k;
    while(m--){
        int u, v; cin >> u >> v;
```

```
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    while(k--){
        int u; cin >> u;
        init[u] = 1;
    }

    dfs();
    if(st.size()){
        ++node;
        while(st.size()){
            int u = st.top(); st.pop();
            color[u] = node;
        }
    }

    for(int u = 1; u <= n; ++u)
        for(int v : adj[u])
            if(color[u] != color[v])
                adj2[color[u]].push_back(color[v]);

    for(int u = 1; u <= node; ++u){
        sort(adj2[u].begin(), adj2[u].end());
        adj2[u].erase(unique(adj2[u].begin(), adj2[u].end()), adj2[u].end());
    }

    int root;
    for(int u = 1; u <= node; ++u)
        if(need[u]){
            root = u;
            break;
        }

    dfs2(root);
    for(int u = 1; u <= node; ++u)
        if(ok[u])
            ++edge;
    --edge;

    dfs3(root);
    cout << edge - diameter;
    return 0;
}
```

Độ phức tạp: $O(n+m)$.

Bài 6. Cuộc thi hát

Prerequisites

- Cặp ghép[4] (Chỉ cần hiểu định nghĩa).
- Thành phần liên thông mạnh[5].

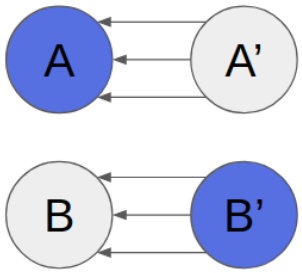
Lời giải

- Ta coi các mối quan hệ giữa bạn nam thứ i và nữ thứ i là một cạnh trong một cặp ghép hoàn hảo[6], các mối quan hệ còn lại là các cạnh không được ghép của cặp ghép.
  - Gọi S là tập bạn nam và bạn nữ được chọn thỏa yêu cầu bài toán.
  - Gọi tập gồm các bạn nam từ 1 tới n là L, tập gồm các bạn nữ từ 1 tới n là R.
  - Định nghĩa:
    + A là tập các phần tử trong $L \cap S$.

+ B là tập các phần tử trong L không xuất hiện trong A.
+ A' là tập các phần tử trong cặp ghép thuộc R có đường đi tới A.
+ B' là tập các phần tử còn lại trong R không thuộc A'.

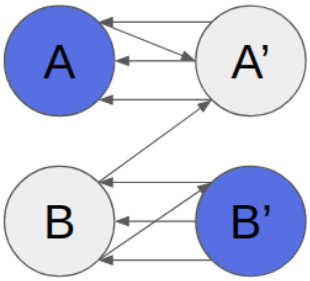
– Dễ thấy rằng $S = A \cup B'$ vì các đỉnh ở R không được có đường đi đến A. Ta định hướng các cạnh của đồ thị như sau:
+ Với các cạnh thuộc cặp ghép, ta sẽ định hướng nó từ R qua L.
+ Với các cạnh còn lại, ta sẽ định hướng chúng từ L sang R.

– Sau khi thêm các cạnh thuộc cặp ghép, ta có đồ thị như sau:



Hình 1: [7] Đồ thị sau khi thêm các cạnh thuộc cặp ghép hoàn hảo

– Sau khi thêm đầy đủ các cạnh của đồ thị, đồ thị của ta nhìn như sau:



Hình 2: [7] Đồ thị đầy đủ

- Nhận thấy rằng các đỉnh ở A không có đường đi tới B' vì theo cách định hướng và chia tập sẽ không có cạnh từ A tới B', A tới B, A' tới B và A' tới B' . Ta sẽ sử dụng thuật toán tìm các thành phần liên thông mạnh Tarjan hoặc Kosaraju để nhóm đồ thị thành các miền liên thông mạnh. Ta có các trường hợp sau:
  – Nếu chỉ có một thành phần liên thông mạnh thì bài toán không có đáp án thỏa do với mọi đỉnh thuộc R sẽ luôn có đường đi từ các đỉnh thuộc L tới đỉnh đó. Vì thế chúng ta sẽ không có cách chọn thỏa điều kiện các giám khảo và thí sinh không được quen biết nhau.
  – Nếu như có ít nhất 2 thành phần liên thông mạnh thì ta chọn một thành phần làm gốc và gọi nó là Q. Giả sử $B' = Q \cap R$ và gọi A là tất cả những nút thuộc L không có cạnh trong cặp ghép với các nút thuộc B'. Ta chỉ cần chứng minh rằng không có nút nào của A nằm trong Q. Thật vậy, xét các đỉnh thuộc $L \cap Q$, giả sử các đỉnh này nằm trong cùng miền liên thông mạnh với các đỉnh thuộc B' nên bắt buộc phải tồn tại đường đi từ các đỉnh này tới các đỉnh thuộc B'. Mà theo cách định hướng cạnh, các đỉnh thuộc A không có đường đi tới B' như đã nói nên các đỉnh thuộc A sẽ nằm ở các miền liên thông mạnh còn lại, qua đó thỏa điều kiện không có đường đi từ A tới B'.
- Đáp án của tập hợp chúng ta cần tìm là $B' \cup A$.

Tips : Ta có thể lưu một đỉnh tượng trưng cho cạnh giữa bạn nam và bạn nữ thứ i thay vì lưu $2n$ đỉnh để tiết kiệm bộ nhớ cũng như thuận lợi cho việc cài đặt.

Code mẫu:

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

using namespace std;
const int N = 1e6 + 5, MOD = 1e9 + 7;
int n, m, label[N], num[N], low[N], timedfs = 0, sccs = 0;
bool del[N];
vector<int> adj[N];
stack<int> st;
void dfs(int i){
    num[i] = low[i] = ++timedfs;
    st.push(i);
    for(int x : adj[i]){
        if(del[x]) continue;
        if(!num[x]){
            dfs(x);
            low[i] = min(low[i], low[x]);
        } else{
            low[i] = min(low[i], num[x]);
        }
    }
    if(low[i] == num[i]){
        int u = -1; ++sccs;
        while(u != i){
            u = st.top(); st.pop();
            del[u] = true, label[u] = sccs;
        }
    }
}
void solve(){
    cin >> n >> m;
    for(int i = 1; i <= m; i++){
        int x, y; cin >> x >> y;
        if(x == y) continue;
        adj[x].push_back(y);
    }
    for(int i = 1; i <= n; i++){ if(!num[i]){
        dfs(i);
    }}
    vector<int> v1, v2;
    for(int i = 1; i <= n; i++){
        if(label[i] == 1) v1.push_back(i);
        else v2.push_back(i);
    }
    if(sccs == 1){
        cout << "No\n";
    } else{
        cout << "Yes\n";
        cout << v1.size() << " " << v2.size() << endl;
        for(int x : v1) cout << x << " ";
        cout << endl;
        for(int x : v2) cout << x << " ";
        cout << endl;
    }

    timedfs = sccs = 0;
    for(int i = 1; i <= n; i++) adj[i].clear(), del[i] = 0, num[i] = low[i] = 0;
    while(st.size()) st.pop();
}
signed main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    //freopen(task".inp", "r", stdin);
    //freopen(task".out", "w", stdout);
    int t = 1; cin >> t;
    while(t--){
```

```
        solve();
    }
    return 0;
}
```

Độ phức tạp: $O(n)$.

Bài 7. Tùy bạn

Lời giải

- Xây dựng cây DFS cho đồ thị và nhìn vào cấu trúc cạnh ngược của cây.
  - Nếu tìm được một cạnh ngược nối hai đỉnh có khoảng cách lớn hơn hoặc bằng $\lceil \sqrt{n} \rceil$, thì ta đã tìm được một chu trình thỏa công việc 2.
  - Nếu không tìm được, đồ thị đơn chứng tỏ mỗi đỉnh có ít hơn $\lceil \sqrt{n} \rceil - 1$ cạnh ngược nối lên trên (theo nguyên lý Dirichlet). Vậy nếu ta xét chọn đỉnh theo thứ tự **con trước cha sau**, nếu đỉnh hiện tại vẫn chọn được (trước đây chưa chọn đỉnh nào kề nó) thì nên chọn nó vào tập tổ hợp độc lập hiện tại. Để ý sau khi chọn như vậy, chỉ có chưa tới $\lceil \sqrt{n} \rceil$ đỉnh khác bị cấm chọn sau này. Do đó có thể chọn được một tổ hợp độc lập thỏa công việc 1.
- Có thể còn có rất nhiều giải thuật khác và heuristic để giải bài này.

Code mẫu

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

int n, m;
int sqrtn;
vector<vector<int>> adj;
vector<bool> vis(n, false);
vector<int> depth(n);
vector<int> parent(n);
vector<bool> is_bad(n, false);
vector<int> independent_set;

void dfs(int u, int p) {
    vis[u] = true;
    parent[u] = p;
    depth[u] = p == -1 ? 0 : depth[p] + 1;

    for (int v : adj[u]) {
        if (v == p) {
            continue;
        }

        if (!vis[v]) {
            dfs(v, u);
            continue;
        }

        if (depth[u] - depth[v] + 1 >= sqrtn) { // big cycle found
            cout << 2 << '\n';
            cout << depth[u] - depth[v] + 1 << '\n';
            do {
                cout << u + 1 << ' ';
                u = parent[u];
            } while (u != v);
            cout << u + 1 << '\n';
            exit(0);
        }
    }

```

```
    if (!is_bad[u]) {
        independent_set.push_back(u);
        is_bad[u] = true;
        for (int v : adj[u]) {
            is_bad[v] = true;
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> n >> m;

    sqrtn = 1;
    while (sqrtn * sqrtn < n) sqrtn++;

    adj.resize(n);
    vis.resize(n, false);
    depth.resize(n);
    parent.resize(n);
    is_bad.resize(n, false);

    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        u--;
        v--;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    dfs(0, -1);

    // dfs terminated without exitting => big independent set found
    cout << 1 << '\n';
    independent_set.resize(sqrtn);
    for (int u : independent_set) {
        cout << u + 1 << ' ';
    }
    cout << '\n';
    exit(0);
}
```

Độ phức tạp: $O(n)$.

Bài 8. Đồ thị lộn xộn

Tính chất bài toán

- **Nhận xét** 1: Với cùng kích thước n, thứ tự mảng $a[]$ đề bài cho không quan trọng. Tất cả thứ tự khác nhau của mảng $a[]$ sẽ đều cho ra cùng một kết quả.
- **Lý do:** Với mọi kết quả của một thứ tự ***dfs*** bất kỳ được cho trong mảng $a[]$, ta luôn có thể **đánh số lại đỉnh của đồ thị** sao cho thứ tự ***dfs*** mới là $1, 2, 3, ..., n$. Ví dụ, với $a[] = \{3, 1, 2\}$, ta đánh số lại đỉnh 3 thành 1, đỉnh 1 thành 2, và đỉnh 2 thành 3, như vậy thứ tự ***dfs*** mới sẽ trở thành $\{1, 2, 3\}$. Việc đánh số này không làm tăng hay giảm số đồ thị thỏa điều kiện nên không ảnh hưởng đến kết quả bài toán.

  $\Rightarrow$ Đề bài có thể được định nghĩa lại thành: Cho n, đếm số đồ thị vô hướng liên thông khác nhau gồm n đỉnh sao cho đồ thị đó tồn tại một thứ tự ***dfs*** có dạng $1, 2, 3, ..., n$.
- **Nhận xét** 2: Thứ tự ***dfs*** của đồ thị sẽ được đặc trưng bởi **DFS Tree** của đồ thị đó. Mỗi đồ thị sẽ dựng ra một và chỉ một DFS Tree, và mỗi DFS Tree sẽ cho ra được một vài thứ tự ***dfs*** phân biệt.

  $\Rightarrow$ Như vậy, ta cần đếm số lượng DFS Tree gồm n đỉnh thỏa điều kiện: trong tất cả thứ tự ***dfs*** mà DFS Tree đó cho ra, tồn tại một thứ tự có dạng $1, 2, 3, ..., n$.

- **Ghi chú:**
  + Tất cả n đỉnh của DFS Tree đều phải liên thông
  + 2 DFS Tree được coi là khác nhau khi tồn tại cạnh (u, v) trong cây này nhưng không tồn tại trong cây kia.
  + Gốc của DFS Tree là đỉnh 1 (vì ta sẽ ***dfs*** từ gốc của cây và ta muốn có được thứ tự $1, 2, 3, ...n$).

Hướng giải quyết

Gợi ý: *Quy hoạch động*

- Ta lần lượt đặt các đỉnh $1, 2, 3, ...n$ theo thứ tự từ nhỏ đến lớn vào DFS Tree.
- Gọi $dp[u][h]$ là số DFS Tree thỏa: cây có u đỉnh, đỉnh u đang nằm ở độ sâu h (**độ cao đánh số từ** 0), h cũng chính là **số lượng tổ tiên** của đỉnh u, và cây thỏa điều kiện cho ra được thứ tự ***dfs*** có dạng $1, 2, 3, ...n$.
- **Công thức truy hồi:**

$$dp[1][0] = 1$$
$$dp[u][h] = 2^{h-1} \times \sum_{k=h-1}^{H} dp[u-1][k]$$

  Với H là độ sâu tối đa có thể của cây DFS Tree gồm $u-1$ đỉnh (thực tế $H = u-2$).

- **Giải thích:**
  - Khi đặt đỉnh thêm đỉnh u vào đồ thị, ta phải đặt đỉnh u làm con của một đỉnh nằm trên được đi từ $u-1$ về đỉnh 1 (bao gồm cả $u-1$ và 1). Sau khi gọi hàm $dfs(u-1)$, ta đã có dãy $1, 2, 3, ..., u-1$. Khi đó, hoặc là hàm sẽ tiếp tục gọi tới $dfs(u)$ (nếu u là con của $u-1$), hoặc là hàm sẽ callback về một trong những tổ tiên của $u-1$ (sẽ không có thao tác thêm đỉnh vào $a[]$ trong quá trình này), rồi từ đỉnh này tiếp tục gọi $dfs(u) \Rightarrow$ Thỏa điều kiện cho ra thứ tự ***dfs*** $a[]$ mới là $\{1, 2, 3, ..., u-1, u\}$.
  - Số DFS Tree thỏa điều kiện "đặt được đỉnh u tại độ sâu h, đồng thời u là con của một đỉnh nằm trên được đi từ $u-1$ về 1"chính là biểu thức $\sum_{k=h-1}^{H} dp[u-1][k]$ (số lượng cây có đỉnh $u-1$ nằm ở độ sâu ít nhất là $h-1$). Nếu $u-1$ nằm ở độ sâu $h-1$, ta nối u làm con của $u-1$ thì u sẽ ở độ cao h; còn nếu $u-1$ ở độ sâu $> h-1$, sẽ tồn tại một tổ tiên của $u-1$ nằm ở độ sâu $h-1$, khi đó ta có thể nổi u làm con của đỉnh này.
  - Vì ta đang đếm số lượng DFS Tree, ta sẽ cần xét cả trường hợp cạnh ngược (back edge) của cây. Khi thêm đỉnh u vào độ sâu h, có $h-1$ tổ tiên (không tính đỉnh cha) mà u có thể nối cạnh ngược vào $\Rightarrow$ Có 2^{h-1} cách nối cạnh ngược.

Code mẫu

Sau khi hiểu được ý tưởng đằng sau công thức quy hoạch động, việc cài đặt bài toán trở nên khá đơn giản. Ta chỉ cần sử dụng các vòng *for* lồng nhau để xử lý bài toán như nhiều bài tập QHĐ khác. Lưu ý ta cần sử dụng kỹ thuật cộng dồn hoặc *prefix sum* để tính nhanh tổng $\sum_{k=h-1}^{H} dp[u-1][k]$.

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

using namespace std;

const int N = 5e3, M = 1e5, mod = 998244353;

int n;
long long dp[N + 5][N + 5];
long long power[N + 5];

void process() {
    cin >> n;
```

```
    for (int i = 1; i <= n; i++) {
        int x;
        cin >> x;
    }

    power[0] = 1;
    for (int i = 1; i <= n; i++) power[i] = power[i - 1] * 2 % mod;

    dp[1][0] = 1;
    for (int i = 2; i <= n; i++) {
        long long sum = 0;
        for (int j = i - 1; j >= 1; j--) {
            sum += dp[i - 1][j - 1];
            sum %= mod;
            dp[i][j] = sum * power[j - 1] % mod;
        }
    }

    long long res = 0;
    for (int i = 0; i < n; i++) res += dp[n][i];
    cout << res % mod;
}

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

    process();

    return 0;
}
```

Độ phức tạp: $O(n^2)$.

Tài liệu

[1] Cây DFS (Depth-First Search Tree) và ứng dụng - VNOI. (n.d.). https://vnoi.info/wiki/algo/graph-theory/Depth-First-Search

[2] Confusing bitsets - Codeforces. (n.d.). https://codeforces.com/blog/entry/71076

[3] Wikipedia contributors. (2023, August 12). Biconnected component # Block-cut tree. https://en.wikipedia.org/wiki/Biconnected_component#Block-cut_tree

[4] Cặp ghép I: Giới thiệu — Matching I: Introduction. (2020, November 12). Giải Thuật Lập Trình. https://www.giaithuatlaptrinh.com/?p=2169

[5] Đồ Thị - Thành Phần Liên Thông Mạnh | chuyenhalong.ucode.vn - Lập trình tương lai của bạn. (n.d.). https://chuyenhalong.ucode.vn/blog/do-thi-thanh-phan-lien-thong-manh-3541

[6] Wikipedia contributors. (2023, June 19). Perfect matching. https://en.wikipedia.org/wiki/Perfect_matching

[7] Editorial of Codeforces Round #594 (on the problems of Moscow Team Olympiad) - Codeforces. (n.d.). https://codeforces.com/blog/entry/70720