Chuyên đề: CÂY DFS VÀ CÁC BÀI TẬP VỀ LIÊN THÔNG MẠNH

Phần I. Mở đầu

Mục đích và lí do chọn chuyên đề:

Đồ thị là một chuyên đề không thể thiếu trong quá trình bồi dưỡng học sinh giỏi môn Tin học. Trong các đề thi học sinh giỏi luôn có các bài tập thuộc chuyên đề đồ thị, trong đó có các bài tập liên quan đến các thành phần liên thông mạnh.

Cây DFS có ứng dụng hiệu quả trong việc giải quyết các bài toán liên quan đến các thành phần liên thông mạnh. Trong phạm vi chuyên đề này, tôi trình bày về cây DFS và ứng dụng để giải các bài toán liên quan đến các thành phần liên thông mạnh.

Phần II. Một số vấn đề lý thuyết

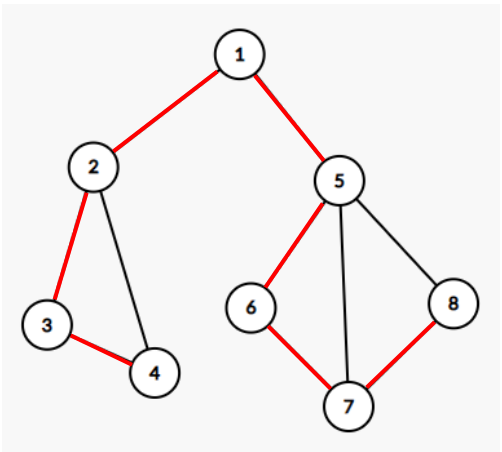
1. Khái niệm liên thông mạnh

Một đồ thị có hướng được gọi là liên thông mạnh khi luôn có đường đi theo các cung định hướng giữa các cặp đỉnh bất kỳ trong đồ thị.

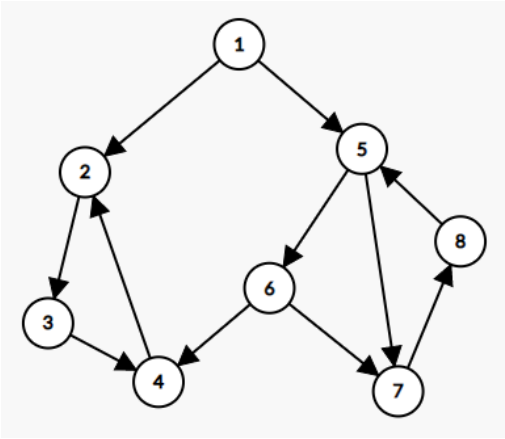
Một thành phần liên thông mạnh là một đồ thị con liên thông mạnh có tính chất "tối đại", tức là nếu thêm bất kỳ một đỉnh khác thuộc đồ thị ban đầu vào đồ thị con thì nó sẽ làm mất tính liên thông mạnh của đồ thị con mới.

2. Cây DFS

Cây DFS được hình thành trong quá trình duyệt đồ thị ưu tiên theo chiều sâu. Trong quá trình duyệt DFS, với mỗi đỉnh v của đồ thị ta có đỉnh par[v] là đỉnh cha của đỉnh v, tức là từ đỉnh par[v] thủ tục DFS gọi đệ quy đến v. Tập hợp các cạnh (par[v] , v) và các đỉnh nối bởi các cạnh này tạo thành cây DFS. Trong hình dưới gồm các cạnh màu đỏ.



Trong đồ thị có hướng, sau khi duyệt DFS, các cung được phân thành các loại như sau:



- Cung thuộc cây DFS (màu đen)

- Cung ngược (màu đỏ) là cung không thuộc cây DFS, nối một đỉnh với tổ tiên của nó

- Cung xuôi (màu tím) là cung không thuộc cây DFS, nối một đỉnh với hậu duệ của nó

- Cung chéo (màu xanh) là cung không thuộc cây DFS, nối giữa hai đỉnh thuộc hai nhánh khác nhau của cây DFS.

3. Cây DFS và thuật toán Tarjan tìm các thành phần liên thông mạnh

Phần này trình bày một số định lý và nhận xét quan trọng làm cơ sở cho thuật toán Tarjan tìm các thành phần liên thông mạnh (không trình bày lại phần chứng minh đã được trình bày trong sách thầy Lê Minh Hoàng)

Định lý 1: Nếu a, b là hai đỉnh thuộc thành phần liên thông mạnh C thì với mọi đường đi từ a tới b cũng như từ b tới a. Tất cả đỉnh trung gian trên đường đi đó đều phải thuộc C.

Định lý 2: Với một thành phần liên thông mạnh C bất kỳ, tồn tại một đỉnh r thuộc C sao cho mọi đỉnh của C đều thuộc cây con gốc r trong cây DFS.

Định lý 3: Luôn tìm được đỉnh chốt a thỏa mãn: Quá trình tìm kiếm theo chiều sâu bắt đầu từ a không thăm được bất kỳ một chốt nào khác. (Tức là nhánh DFS gốc a không chứa một chốt nào ngoài a) chẳng hạn ta chọn a là chốt được thăm sau cùng trong một dây chuyền đệ quy hoặc chọn a là chốt thăm sau tất cả các chốt khác. Với chốt a như vậy thì các đỉnh thuộc nhánh DFS gốc a chính là thành phần liên thông mạnh chứa a.

Thuật toán Tarjan

Ý tưởng: chọn u là chốt mà từ đó quá trình tìm kiếm theo chiều sâu không thăm thêm bất kỳ một chốt nào khác, chọn lấy thành phần liên thông mạnh đầu tiên là nhánh DFS gốc u, sau đó loại bỏ nhánh DFS gốc u ra khỏi cây DFS. Tiếp theo, tìm một đỉnh chốt v khác mà nhánh DFS gốc v không chứa chốt nào khác, lại chọn thành phần liên thông mạnh thứ hai là nhánh DFS gốc v, loại bỏ nhánh DFS gốc v ra khỏi cây DFS. Tiếp tục như vậy cho đến khi không còn chốt nào khác.

Để xác định một đỉnh v nào đó có phải là chốt hay không, ta có các nhận xét sau:

Nhật xét 1: Nếu các đỉnh thuộc nhánh DFS gốc v không có cung ngược hay cung chéo nào đi ra khỏi nhánh đó thì v là chốt.

Nhận xét 2: Nếu từ một đỉnh nào đó thuộc nhánh DFS gốc v có một cung ngược tới một đỉnh là tiền bối của v thì v không là chốt.

Để thuận tiện trong việc cài đặt, ta dùng các mảng với ý nghĩa như sau:

Mảng Num[u] là thứ tự thăm của đỉnh u trong quá trình duyệt DFS

Mảng Low[u] là giá trị Num[.] nhỏ nhất trong các đỉnh có thể đến được từ một đỉnh v nào đó thuộc nhánh DFS gốc u.

Cách xác định Low[u] như sau:

Trong thủ tục DFS(u), trước hết ta đánh số thứ tự thăm cho đỉnh u là Num[u] và khởi gán Low[u] = Num[u]. Sau đó, với mỗi đỉnh v nối từ u thì có hai khả năng:

Khả năng 1: v đã được thăm thì gán Low[u] = min(Low[u], Num[v])

Khả năng 2: v chưa được thăm thì gọi đệ quy thăm v, sau đó gán Low[u] = min(Low[u], Low[v])

Cài đặt chi tiết sẽ được minh họa thêm ở Bài tập số 1 sau đây.

Phần III. Bài tập

Bài 1: Đếm số thành phần liên thông mạnh - TJALG

Cho đồ thị có hướng G(V, E) có N ($1 \le N \le 10^4$) đỉnh, M ($1 \le M \le 10^4$) cung. Hãy đếm số thành phần liên thông mạnh của G.

Input: Cho trong tệp TJALG.INP có cấu trúc:

- Dòng đầu tiên ghi hai số N và M lần lượt là số đỉnh và số cung của G.

- M dòng tiếp theo, dòng thứ i ghi hai số u_i và v_i cho biết có cung nối từ đỉnh u_i đến đỉnh v_i ($1 \le u_i, v_i \le N$)

Output: Ghi ra tệp TJALG.OUT gồm một dòng ghi một số là số lượng thành phần liên thông mạnh.

Ví dụ:

TJALG.INP	TJALG.OUT
8 11 1 2 1 5 2 3	3

3 4 4 2 5 6 5 7 6 4 6 7 7 8 8 5	

Hướng dẫn:

Đây là bài tập cơ bản nhất, đếm số thành phần liên thông mạnh của một đồ thị có hướng, chỉ cần áp dụng thuật toán Tarjan là giải quyết được.

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

using namespace std;

const int maxN = 10005;

int n, m, cnt = 0, scc = 0, low[maxN], num[maxN];
bool fre[maxN];
vector <int> g[maxN];
stack <int> st;

void dfs(int u) {
    cnt++;
    num[u] = low[u] = cnt; //khoi gan Num[], Low[]
    st.push(u);
    for (int ii = 0; ii < g[u].size(); ii++) {
        int v = g[u][ii];
        if (fre[v]) continue;
        if (num[v] != 0) low[u] = min(low[u], num[v]);
        else {
            dfs(v);
            low[u] = min(low[u], low[v]);
        }
    }
    //xac dinh u la dinh chot
    if (low[u] == num[u]) {
        scc++; //tang so luong thanh phan lien thong manh
        int v;
        do {
            v = st.top();
            fre[v] = true; //loai dinh v khoi cay DFS
            st.pop();
        }
        while (v != u);
    }
}
int main() {
```

```
    freopen("TJALG.INP","r",stdin);
    freopen("TJALG.OUT","w",stdout);
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
    }
    for (int i = 1; i <= n; i++) num[i] = 0;
    for (int i = 1; i <= n; i++)
        if (num[i] == 0) dfs(i);
    cout << scc;
    return 0;
}
```

Bài 2: Công ty đa cấp – SCOM

Công ty bán hàng đa cấp SCOM có tất cả N người tham gia và được đánh số thứ tự từ 1 đến N, mỗi người có một số điện thoại và một người có thể biết số điện thoại của một số người khác. Việc liên lạc thông tin trong công ty được quy định thực hiện theo hình thức nhắn tin dây chuyền: Giám đốc sẽ nhắn tin cho một số người, những người nhận được tin nhắn của giám đốc thì sẽ nhắn tiếp cho những người trong công ty mà người này biết số điện thoại và người nhận được tin nhắn lại tiếp tục nhắn cho những người mà họ biết số điện thoại,…

Hiện giám đốc đang có một thông tin quan trọng muốn nhắn đến cho tất cả mọi trong công ty nhưng thay vì ông phải nhắn tin đến tất cả mọi người thì ông muốn chọn một số ít nhất người để nhắn tin mà theo quy định nhắn tin dây chuyền của công ty thì tất cả mọi người trong công ty đều nhận được tin nhắn.

Bạn hãy giúp ông giám đốc xác định xem ông cần nhắn tin đến ít nhất là bao nhiêu người (giả sử giám đốc biết số điện thoại của tất cả mọi người trong công ty)

Input: cho trong file SCOM.INP có cấu trúc:

- Dòng đầu ghi hai số nguyên dương N, M ($1 \le N, M \le 10^5$)

- M dòng tiếp theo, mỗi dòng ghi hai số nguyên dương u và v cho biết người u biết số điện thoại của người v.

Output: ghi ra file SCOM.OUT gồm 1 dòng ghi số duy nhất là số lượng người ít nhất giám đốc cần nhắn tin.

Ví dụ:

SCOM.INP	SCOM.OUT
12 15 1 3 3 6 6 1 6 8 8 12 12 9 9 6 2 4 4 5 5 2 4 6 7 10 10 11 11 7 10 9	2

Hướng dẫn:

Xem mỗi nhân viên là một đỉnh của đồ thị, quan hệ biết số điện thoại là các cung của đồ thị có hướng. Dễ thấy các đỉnh trong cùng thành phần liên thông mạnh thì chỉ cần nhắn tin đến một đỉnh, như vậy học sinh có thể nghĩ đáp án của bài toán là số thành phần liên thông mạnh. Tuy nhiên, do có thể từ một đỉnh thuộc thành phần liên thông mạnh này có thể có cung đến một đỉnh thuộc thành phần liên thông mạnh khác nên đáp án chỉ là số thành phần liên thông mạnh không có cung vào.

Để kiểm tra một thành phần liên thông mạnh không có cung vào, có thể thực hiện như sau:

- Dùng một mảng ltm[u] = s, với ý nghĩa u thuộc thành phần liên thông mạnh thứ s

- Xét mỗi cung (u,v), nếu ltm[u] ≠ ltm[v] thì lúc này đánh dấu thành phần liên thông mạnh chứa v là có cung vào (vao[ltm[v]] = true).

- Xét lại các thành phần ltm để đếm số thành phần ltm không có cung vào.

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

using namespace std;

const int N = 100005;
const int oo = 0x3c3c3c3c;
```

```
int n, m, Num[N], Low[N], cnt = 0;
vector<int> a[N];
deque<int> st;
int ltm[N];
int Count = 0;
pair<int,int> r[N];
bool vao[N];

void visit(int u) {
    Low[u] = Num[u] = ++cnt;
    st.push_back(u);
    for (int o = 0;o < a[u].size();o++) {
        int v = a[u][o];
        if (Num[v])
            Low[u] = min(Low[u], Num[v]);
        else {
            visit(v);
            Low[u] = min(Low[u], Low[v]);
        }
    }
    if (Num[u] == Low[u]) {
        Count++;
        int v;
        do {
            v = st.back();
            ltm[v] = Count;
            st.pop_back();
            Num[v] = Low[v] = oo;
        } while (v != u);
    }
}
int main() {
    freopen("SCOM.INP","r",stdin);
    freopen("SCOM.OUT","w",stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= m; i++) {
        int x, y;
        cin >> x >> y;
        a[y].push_back(x);
        r[i] = make_pair(x,y);
    }
    for (int i = 1; i <= n; i++)
        if (!Num[i]) visit(i);
    for (int i = 1;i <= m;i++)
        if (ltm[r[i].first] != ltm[r[i].second]) vao[ltm[r[i].second]] = true;
    int res = 0;
    for (int i = 1;i <= Count;i++)
        if (vao[i] == false)
            res++;
    cout << res;
    return 0;
}
```

Bài 3: Các thành phố trung tâm – CENCITY

Đất nước ALLCITY là một đất nước rất phát triển, các đơn vị hành chính đều là các thành phố, có N thành phố được đánh số từ 1 đến N. Có M đường một chiều nối giữa các cặp thành phố của đất nước này. Chính quyền thành phố muốn chọn một thành phố để đặt thủ đô của đất nước, họ muốn chọn thành phố đặt thủ đô phải đảm bảo tất cả các thành phố khác đều có đường đi đến thành phố này.

Bạn làm việc cho cơ quan quy hoạch của đất nước và được giao nhiệm vụ tìm tất cả các thành phố có thể làm ứng viên để chính quyền xem xét lựa chọn đặt thủ đô tại một trong các thành phố ứng viên này.

***Input*:** cho trong file CENCITY.INP có cấu trúc như sau:

- Dòng đầu tiên ghi hai số N và M ($1 \le N \le 10^5, 1 \le M \le 2x10^5$)

- M dòng tiếp theo, mỗi dòng ghi hai số u và v cách nhau dấu cách cho biết có đường một chiều từ thành phố u tới thành phố v.

***Output*:** ghi vào file CENCITY.OUT có cấu trúc như sau:

- Dòng thứ nhất ghi một số nguyên T là số lượng thành phố ứng viên

- Nếu T > 0 thì dòng tiếp theo ghi T số là số thứ tự của các thành phố ứng viên, các số ghi cách nhau một dấu cách và ghi theo thứ tự tăng dần.

Ví dụ:

CENCITY.INP	CENCITY.OUT
4 4 1 2 2 3 3 2 4 1	2 2 3

Hướng dẫn:

Các thành phố ứng viên phải thuộc cùng một thành phần liên thông mạnh L không có cung ra (vì nếu giả sử một thành phố u nào đó có cung ra một thành phố v nào khác không thuộc L thì từ v không thể đến được các thành phố thuộc L, nếu từ v đến được thành phố thuộc L thì v phải thuộc L). Như vậy bài toán trở thành tìm thành

phần liên thông mạnh không có cung ra. Lưu ý thêm là nếu có nhiều hơn một thành phần liên thông mạnh không có cung ra thì kết quả T sẽ bằng 0.

Có thể giải quyết bài này như bài 2 ở trên. Tuy nhiên, đoạn code sau đây sẽ không áp dụng thuật toán TARJAN để tìm các thành phần liên thông mạnh mà dùng thuật toán khác được mô tả qua các bước như sau:

- Xây dựng đồ thị graph để biểu diễn bài toán ban đầu, xây dựng đồ thị graphT là đồ thị đảo của graph.

- DFS1 trên đồ thị graph để đánh thứ tự duyệt xong của các đỉnh từ đỉnh duyệt xong đầu tiên đến đỉnh duyệt xong sau cùng.

- DFS2 trên đồ thị graphT theo thứ tự đỉnh duyệt xong sau cùng đến đỉnh duyệt xong đầu tiên để xác định các thành phần liên thông mạnh.

- Đếm số lượng thành phần liên thông mạnh không có cung ra và đưa ra kết quả bài toán.

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

#define MAXN 200001

using namespace std;

vector<int> graph[MAXN], graphT[MAXN], hadvis;
bool visited[MAXN];
int comp[MAXN], out[MAXN];

void dfs1(int u) {
    visited[u] = true;
    for (int i = 0; i < graph[u].size(); ++i) {
        if (!visited[graph[u][i]]) {
            dfs1(graph[u][i]);
        }
    }
    hadvis.push_back(u);
}

void dfs2(int u, int c) {
    visited[u] = false;
    comp[u] = c;
    for (int i = 0; i < graphT[u].size(); ++i) {
        if (visited[graphT[u][i] ]) {
            dfs2(graphT[u][i], c);
        }
    }
}

int main() {
    freopen("CENCITY.INP","r",stdin);
    freopen("CENCITY.OUT","w",stdout);
```

```
    ios_base::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);
    int t, i, N, v, j, cntltm, M, u, cnt1, cnt2;
    scanf( "%d%d", &N, &M );
    for (i = 1; i <= N; ++i) {
        visited[i] = false;
        graph[i].clear();
        graphT[i].clear();
        out[i] = 0;
    }
    for (i = 1; i <= M; ++i) {
        scanf("%d%d", &u, &v);
        graph[u].push_back(v);
        graphT[v].push_back(u);
    }
    for (i = 1; i <= N; ++i) {
        if (!visited[ i ]) {
            dfs1(i);
        }
    }
    cntltm = 0;
    for (i = hadvis.size() - 1; i >= 0; --i) {
        if (visited[hadvis[i]]) {
            dfs2(hadvis[i], cntltm++);
        }
    }
    for (i = 1; i <= N; ++i) {
        for (j = 0; j < graph[ i ].size(); ++j) {
           if (comp[i] != comp[graph[i][j]]) {
                out[comp[i]] = 1;
            }
        }
    }
    cnt1 = 0;
    for (i = 0; i < cntltm; ++i) {
        if (out[i] == 0) {
            ++cnt1;
        }
    }
    if ( cnt1 > 1 ) {
        printf("0");
    }
    else {
        cnt2 = 0;
        for (i = 1; i <= N; ++i) {
            if (out[comp[i]] == 0 ) {
                ++cnt2;
            }
        }
        printf("%d\n", cnt2);
        for (i = 1; i <= N; ++i) {
            if (out[comp[i]] == 0) {
                printf("%d ", i);
            }
```

```
        }
    }
    return 0;
}
```

Bài 4: Gán nhãn – LABLE

Cho một đơn đồ thị có hướng (có thể có khuyên) gồm N đỉnh và M cạnh. Các đỉnh được đánh chỉ số từ 1 đến N. Với mỗi đỉnh u từ 1 đến N, hãy xác định giá trị cho F(u):

F(u) = 0, nếu như không có đường đi từ 1 đến u

F(u)= 1, nếu như chỉ có duy nhất một đường đi từ 1 đến u

F(u) = 2, nếu như có nhiều hơn 1 đường đi từ 1 đến u, nhưng số đường đi đếm được là hữu hạn

F(u) = -1, nếu như có vô số đường đi từ 1 đến u

Chú ý, một đường đi có thể đi qua một đỉnh hoặc một cung nhiều lần!

***Input*:** cho trong file LABLE.INP có cấu trúc như sau:

- Dòng đầu tiên gồm 2 số N và M, tương ứng số đỉnh và số cạnh

- M dòng tiếp theo, mỗi dòng gồm 2 số u và v, tương ứng có cung từ đỉnh u đến đỉnh v

***Output*:** ghi ra file LABLE.OUT có cấu trúc như sau:

- Gồm một dòng duy nhất, gồm N số nguyên, số thứ u tương ứng với F(u). Các số cách nhau bởi 1 khoảng trắng

Ví dụ:

LABLE.INP	LABLE.OUT
6 7 1 4 1 3 3 4 4 5 2 1 5 5 5 6	1 0 1 2 -1 -1

Ràng buộc

- 20% số tests có 1 <= N <= 1000, 1 <= M <= 5000
- 80% số tests có 1 <= N, M <= 300000

Hướng dẫn:

Sử dụng thuật toán tìm các thành phần liên thông mạnh, các đỉnh thuộc cùng thành phần liên thông mạnh với đỉnh 1 thì đáp án là -1. Ngược lại thì xây dựng một DAG từ các thành phần liên thông mạnh để xác định các giá trị F còn lại

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

void solve() {
    int n, m;
    cin >> n >> m;
    vector<vector<int>> e(n);
    vector<bool> loop(n, false);
    for (int i = 0, u, v; i < m; ++i) {
        cin >> u >> v;
        --u; --v;
        e[u].push_back(v);
        if (u == v) {
            loop[u] = true;
        }
    }
    vector<int> own(n, -1);
    vector<bool> dagloop;
    vector<int> dagcnt;
    int comps = 0;
    {
        int timer = 0;
        vector<int> num(n, -1);
        vector<int> low(n, -1);
        stack<int> st;
        function<void(int)> dfs = [&](int u) {
            num[u] = low[u] = timer++;
            st.push(u);
            for (auto v: e[u]) {
                if (own[v] != -1) {
                    continue;
                }
                if (num[v] != -1) {
                    low[u] = min(low[u], num[v]);
                } else {
                    dfs(v);
                    low[u] = min(low[u], low[v]);
                }
            }
            if (num[u] == low[u]) {
                dagcnt.push_back(0);
                dagloop.push_back(false);
```

```
                while (true) {
                    int v = st.top();
                    st.pop();
                    own[v] = comps;
                    dagcnt.back() ++;
                    dagloop.back() = dagloop.back() || loop[v] || (dagcnt.back()
> 1);
                    if (v == u) {
                        break;
                    }
                }
                ++comps;
            }
        };
        dfs(0);
    }
    vector<vector<int>> ee(comps);
    vector<int> dagdeg(comps, 0);
    for (int u = 0; u < n; ++u) {
        for (auto v: e[u]) {
            if (own[u] != -1 && own[v] != -1 && own[u] != own[v]) {
                ++dagdeg[own[v]];
                ee[own[u]].push_back(own[v]);
            }
        }
    }
    assert(dagdeg[own[0]] == 0);
    vector<int> dp(comps, 0);
    dp[own[0]] = 1;
    queue<int> q;
    q.push(own[0]);
    while (!q.empty()) {
        auto u = q.front();
        q.pop();
        for (auto v: ee[u]) {
            --dagdeg[v];
            dagloop[v] = dagloop[v] || dagloop[u];
            dp[v] = min(dp[v] + dp[u], 2);
            if (dagdeg[v] == 0) {
                q.push(v);
            }
        }
    }
    for (int i = 0; i < n; ++i) {
        if (own[i] == -1) {
            cout << "0 ";
            continue;
        }
        if (dagloop[own[i]]) {
            cout << "-1 ";
            continue;
        }
        cout << dp[own[i]] << " ";
    }
```

```
}

int main() {
    freopen("LABLE.INP","r",stdin);
    freopen("LABLE.OUT","w",stdout);
    ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
    solve();
    return 0;
}
```

Bài 5: Cung cấp nhu yếu phẩm – FOODS

Thành phố S có N thị trấn nhỏ, thị trấn i có C[i] người dân sinh sống và giữa 2 thành phố có tối đa một con đường một chiều đi từ thành phố này đến thành phố kia.

Hiện nay thành phố đang chịu ảnh hưởng nghiêm trọng của dịch bệnh, mỗi người dân bị hạn chế đi ra ngoài, bởi vậy nên chính quyền thành phố phải cử bộ đội đi cung cấp nhu yếu phẩm cho mọi người. Giả sử các anh bộ đội xuất phát tại thị trấn i, họ sẽ thông qua những con đường một chiều để đi đến những thị trấn khác, thăm và phát nhu yếu phẩm cho tất cả người dân sinh sống trên thị trấn họ đi qua (*lưu ý: mỗi người dân chỉ được thăm và phát nhu yếu phẩm 1 lần*).

Yêu cầu đặt ra là với mỗi thị trấn xuất phát, thì có thể thăm và phát nhu yếu phẩm cho tối đa bao nhiêu người dân?

***Input*:** cho trong file FOODS.INP có cấu trúc như sau:

- Dòng đầu tiên gồm 2 số N và M, tương ứng số thị trấn trong thành phố S, và số con đường một chiều

- Dòng tiếp theo gồm N số nguyên tương ứng số dân sinh sống tại từng thị trấn

- M dòng tiếp theo, mỗi dòng gồm 2 số u và v, tương ứng có con đường từ thị trấn u đến thị trấn v

***Output*:** ghi ra file FOODS.OUT có cấu trúc như sau:

- Gồm một dòng duy nhất ghi N số nguyên, số thứ u tương ứng với số người dân tối đa có thể được thăm và phát nhu yếu phẩm nếu như xuất phát tại thị trấn u. Các số cách nhau bởi 1 khoảng trắng.

Ví dụ:

FOODS.INP	FOODS.OUT

7 8 1 2 3 4 5 6 7 1 2 2 3 3 4 4 5 5 2 5 6 5 7 4 7	22 21 21 21 21 6 7

Ràng buộc

- 30% số tests có 1 <= N, M <= 1000
- 70% số tests có 1 <= N, M <= 100000

Hướng dẫn:

Sử dụng thuật toán tìm các thành phần liên thông mạnh, xây dựng DAG từ những thành phần liên thông mạnh và thứ tự TOPO, sau đó quy hoạch động trên thứ tự TOPO này

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

int main() {
    freopen("FOODS.INP","r",stdin);
    freopen("FOODS.OUT","w",stdout);
    ios::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);
    int n, m;
    cin >> n >> m;
    vector<vector<int>> e(n);
    vector<int> c(n);
    for (int i = 0; i < n; ++i) {
        cin >> c[i];
    }
    for (int i = 0, u, v; i < m; ++i) {
        cin >> u >> v;
        --u;
        --v;
        e[u].push_back(v);
    }
    vector<int> own(n, -1);
    vector<long long> cnt;
    int comps = 0;
    {
```

```
        int timer = 0;
        vector<int> num(n, -1);
        vector<int> low(n, -1);
        stack<int> st;
        function<void(int)> dfs = [&](int u) {
            num[u] = low[u] = timer++;
            st.push(u);
            for (auto v: e[u]) {
                if (own[v] != -1) {
                    continue;
                }
                if (num[v] != -1) {
                    low[u] = min(low[u], num[v]);
                } else {
                    dfs(v);
                    low[u] = min(low[u], low[v]);
                }
            }
            if (num[u] == low[u]) {
                cnt.push_back(0);
                while (true) {
                    int v = st.top();
                    st.pop();
                    own[v] = comps;
                    cnt.back() += c[v];
                    if (v == u) {
                        break;
                    }
                }
                ++comps;
            }
        };
        for (int i = 0; i < n; ++i) {
            if (own[i] == -1) {
                dfs(i);
            }
        }
    }
    vector<vector<int>> ee(comps);
    vector<int> deg(comps, 0);
    for (int u = 0; u < n; ++u) {
        for (auto v: e[u]) {
            if (own[u] != own[v]) {
                ++deg[own[u]];
                ee[own[v]].push_back(own[u]);
            }
        }
    }
    queue<int> q;
    vector<long long> dp(comps, 0);
    for (int u = 0; u < comps; ++u) {
        if (deg[u] == 0) {
            q.push(u);
            dp[u] = cnt[u];
```

```
        }
    }
    while (!q.empty()) {
        auto u = q.front();
        q.pop();
        for (auto v: ee[u]) {
            --deg[v];
            dp[v] = max(dp[v], dp[u] + cnt[v]);
            if (deg[v] == 0) {
                q.push(v);
            }
        }
    }
    for (int i = 0; i < n; ++i) {
        cout << dp[own[i]] << " ";
    }
    return 0;
}
```

Link Tests và Code mẫu:

https://drive.google.com/drive/folders/1_6fBHZHyhEaJcwlPyGIDvAoZ6YWGj9JL?usp=sharing

Một số bài tập tham khảo khác

1. https://vn.spoj.com/problems/MESSAGE/
2. https://vn.spoj.com/problems/TREAT/
3. https://codeforces.com/contest/427/problem/C
4. https://codeforces.com/problemset/problem/894/E
5. https://codeforces.com/contest/22/problem/E

Kết luận

Cây DFS có nhiều ứng dụng trong việc giải quyết một số bài toán về đồ thị, đặc biệt là các bài toán liên quan đến các thành phần liên thông mạnh. Ứng dụng cây DFS và thuật toán Tarjan có thể tìm các thành phần liên thông mạnh một cách dễ dàng, làm cơ sở để giải quyết một số bài toán liên quan. Chuyên đề đã trình bày đầy đủ năm bài toán và đưa thêm một số bài tập tham khảo. Mong nhận được sự quan tâm, góp ý của quý thầy cô đồng nghiệp.

Tài liệu tham khảo

- Sách giáo khoa chuyên tin tập 1,2,3 – Nhà xuất bản giáo dục năm 2009.
- Giải thuật và lập trình của TS. Lê Minh Hoàng ĐHSP Hà Nội.
- http://vnoi.info/wiki/Home
- https://vn.spoj.com/
- https://www.hackerrank.com/

- https://codeforces.com/
- https://www.codechef.com
- https://community.topcoder.com