DFS Templates

1. Tìm thành phần liên thông

Input:

```
int n;                                  // số đỉnh
vector<vector<int> > adj;               // vector kề
```

Output:

```
vector<int> LT;                         // LT[u]=số hiệu thành phần liên thông của u
int slt;                                // Số lượng TP liên thông
```

Code:

```
void DFS(int u) {
    LT[u]=slt;
    for(int v: adj[u])
    if (cl[v]==0) DFS(v);
}

int main() {
    ....
    LT.resize(n+1,0);
    slt=0;
    for(int u=1;u<=n;++u) if (LT[u]==0) {++slt, DFS(u);}
    ....
}
```

2. Tìm đường đi có từ điển nhỏ nhất

Input:

```
int n;                                  // số đỉnh
vector<vector<pair<int,int> > > adj;    // first-đỉnh kề,second -  màu
// Chú ý các đỉnh kề được liệt kê theo màu tăng dần
```

Output:

```
vector<pair<int,int> > Prev;            // Mảng để tìm danh sách màu
```

Code:

```
vector<int> cl;
void DFS(int u) {
    cl[u]=1;
    for(pair<int,int> id: adj[u]) {
        int v=id.first, c=id.second;
        if (cl[v]==0) {
            Prev[v]={u,c};
            cl[v]=1;
            DFS(v);
        }
    }
}

int main() {
    ....
    cl.resize(n+1,0);
    Prev[xp]={0,0}
    DFS(xp);
    // Tìm đường đi từ điển nhỏ nhất từ xp đến t.In ra danh sách màu
```

```
    if (cl[t]) {
        vector<int> path;
        int u=t;
        while (u!=xp) {
            path.push_back(Prev[u].second);
            u=Prev[u].first;
        }
        reverse(path.begin(), path.end());
        for(int c : path) cout << c << ' ';
    }
}
```

3. Tìm đỉnh khớp

Input:

```
int n;
vector<vector<int> > adj;
```

Output:

```
vector<int> Khop;
```

Code:

```
vector<int> cl, num, low;
int id=0;
void DFS(int u,int dad) {
    int cnt=0;
    cl[u]=1;
    num[u]=low[u]=++id;
    for(int v: adj[u]) {
        if (v==dad) ++cnt;
        if (cnt>1 || v!=dad) {
            if (cl[v]==0) {
                DFS(v);
                low[u]=min(low[u],low[v]);
                if (low[v]>=num[u]) ++khop[u];
            } else low[u]=min(low[u],num[v]);
        }
    }
}

int main() {
    ....
    cl.resize(n+1,0); num.resize(n+1,0); low.resize(n+1,0); id=0;
    for(int u=1;u<=n; ++u) if (cl[u]==0) {
        int socon=0;
        cl[u]=1;
        for(int v: adj[u]) if (cl[v]==0) {
            ++socon;
            DFS(v,u);
        }
        khop[u]=socon-1;
    }
}
```

4. Tìm cạnh cầu

Input:

```
int n;                          // Số đỉnh
```

```
vector<vector<int> > adj;        // Vector kề
int m;                           // Số cạnh
vector<pair<int,int> > E;        // Danh sách cạnh
```

Output:

```
vector<int> cau;                 // cau[i]=1 nếu cạnh thứ i là cầu
```

Code:

```
vector<int> cl, num, low, Prev;
int id=0;
void DFS(int u,int dad) {
    int cnt=0;
    cl[u]=1;
    num[u]=low[u]=++id;
    for(int v: adj[u]) {
        if (v==dad) ++cnt;
        if (cnt>1 || v!=dad) {
            if (cl[v]==0) {
                Prev[u]=u; DFS(v);
                low[u]=min(low[u],low[v]);
            } else low[u]=min(low[u],num[v]);
        }
    }
}

int main() {
    ....
    cl.resize(n+1,0); num.resize(n+1,0); low.resize(n+1,0); id=0;
    Prev.resize(n+1,0);
    for(int u=1;u<=n; ++u) if (cl[u]==0) DFS(v,0);
    // Xác định cầu
    cau.resize(m+1,0);
    for(int i=1;i<=m;++i) {
        int u=E[i].first, v=E[i].second;
        if (Prev[u]==v) swap(u,v);
        if (Prev[v]==u && low[v]>num[u]) cau[i]=1;
    }
}
```

5. Tìm thành phần liên thông mạnh

Input:

```
int n;
vector<vector<int> > adj;
```

Output:

```
vector<int> LT;
vector<int> TP;
int slt;
```

Code:

```
vector<int> s, cl, num, low;
int sn=0,id=0;
void DFS(int u) {
    cl[u]=1;
    num[u]=low[u]=++id;
    s[++sn]=u;
```

```
    for(int v: adj[u]) {
        if (cl[v]==0) {
            DFS(v);
            low[u]=min(low[u],low[v]);
        } else if (cl[v]==1) low[u]=min(low[u],num[v]);
    }
    if (num[u]==low[u]) {
        ++slt; int v;
        do {
            v=s[sn--];
            LT[v]=slt;
            TP.push_back(v);
            cl[v]=2;
        } while (v!=u);
    }
}

int main() {
    ....
    cl.resize(n+1,0); num.resize(n+1,0); low.resize(n+1,0);
    LT.resize(n+1,0);
    id=0;
    for(int u=1;u<=n;++u) if (cl[u]==0) DFS(u);
    ....
}
```

6. Tìm thành phần song liên thông cạnh

Input:

```
int n;
vector<vector<int> > adj;
```

Output:

```
vector<int> SLT;
vector<int> TP;
int slt;
```

Code:

```
vector<int> s, cl, num, low;
int id=0, sn=0;
void DFS(int u,int dad) {
    cl[u]=1;
    num[u]=low[u]=++id;
    s[++sn]=u;
    int cnt=0;
    for(int v: adj[u]) {
        if (v==dad) ++cnt;
        if (cnt>1 || v!=dad) {
            if (cl[v]==0) {
                DFS(v,u);
                low[u]=min(low[v],low[u]);
            } else low[u]=min(low[u],num[v]);
        }
    }
    if (low[u]==num[u]) {
        ++slt; int v;
        do {
            v=s[sn--];
```

```
            SLT[v]=slt;
            TP.push_back(v);
        }
    }
}

int main() {
    ....
    cl.resize(n+1,0); num.resize(n+1,0); low.resize(n+1,0);
    SLT.resize(n+1,0);
    for(int u=1;u<=n;++u) if (cl[u]==0) DFS(u,0);
    ...
}
```

7. Tìm thành phần song liên thông đỉnh

Input:

```
int n;
vector<vector<int> > adj;
```

Output:

```
vector<vector<int> > SLT;
```

Code:

```
vector<int> s, cl, num, low;
int sn=0, id=0;
void DFS(int u,int dad) {
    cl[u]=1;
    num[u]=low[u]=++id;
    s[++sn]=u;
    int cnt=0;
    for(int v: adj[u]) {
        if (v==dad) ++cnt;
        if (cnt>1 || v!=dad) {
            if (cl[v]==0) {
                DFS(v,u);
                low[u]=min(low[u],low[v]);
            } else low[u]=min(low[u],num[v]);
        }
    }
    if (low[u]>=num[dad]) {
        int v;
        vector<int> newSLT;
        do {
            v=s[sn--];
            newSLT.push_back(v);
        } while (v!=u);
        if (low[u]==num[dad]) newSLT.push_back(dad);
        SLT.push_back(newSLT);
    }
}

int main() {
    ....
    cl.resize(n+1,0); low.resize(n+1,0); num.resize(n+1,0);
    for(int u=1;u<=n;++u) if (cl[u]==0) DFS(u,0);
    ....
}
```

8. Sắp xếp topo trên DAG

Input:

```
int n;
vector<vector<int> > adj;
```

Output:

```
vector<int> TP;
```

Code:

```
vector<int> cl;
void DFS(int u) {
    cl[u]=1;
    for(int v: adj[u]) {
        if (cl[v]==0) DFS(v);
    }
    TP.push_back(u);
}

int main() {
    ....
    cl.resize(n+1,0);
    for(int u=1;u<=n;++u) if (cl[u]==0) DFS(u);
    reverse(TP.begin(), TP.end());
    ....
}
```