

THUẬT TOÁN TARJAN

Thuật toán Tarjan có hai ứng dụng chính: Tìm khốp và cầu trong đồ thị vô hướng và tìm thành phần liên thông mạnh trong đồ thị có hướng.

1. Tìm khốp và cầu trong đồ thị vô hướng

1.1 Định nghĩa:

- Đồ thị vô hướng liên thông khi và chỉ khi tồn tại đường đi giữa mọi cặp đỉnh.
- Thành phần liên thông của một đồ thị là một đồ thị con trong đó giữa bất kỳ 2 đỉnh nào đều có đường đi đến nhau và không thể nhận thêm bất kỳ một đỉnh nào mà vẫn duy trì tính chất trên.
- Khốp: là đỉnh mà nếu ta bỏ nó đi (cùng các cạnh liên thuộc với nó) ra khỏi đồ thị thì số thành phần liên thông của đồ thị sẽ tăng lên.
- Cầu: Là cạnh nếu ta bỏ nó ra khỏi đồ thị thì số thành phần liên thông của đồ thị sẽ tăng lên.
- Đồ thị song liên thông: Đồ thị không có khốp.
- Tổ tiên của một đỉnh có thể là cha của nó hoặc là chính nó.
- Back edge: cạnh đưa đỉnh hiện tại trong hàm DFS quay về một đỉnh đã được duyệt từ trước.

1.2 ý tưởng:

Thuật toán Tarjan áp dụng DFS để hình thành cây DFS.

- Trên cây DFS, đỉnh được thăm trước gọi là đỉnh cha.
- Nếu con của một đỉnh không có đường đi về cha của đỉnh này thì đỉnh này gọi là khốp.

1.3 Thuật toán:

Thuật toán Tarjan là một biến thể của DFS nên độ phức tạp vẫn là $O(V+E)$.

Chúng ta định nghĩa một số biến như sau:

- Num[u]: thứ tự duyệt đỉnh u khi DFS.
- Low[u]: giá trị num[u] nhỏ nhất đạt được trong cây DFS con của đỉnh u.
- Mảng num[]: cho biết thứ tự duyệt DFS của các đỉnh (thứ tự mà mỗi đỉnh bắt đầu duyệt).
- Mảng low[]: Với mỗi đỉnh u, low[u] cho biết thứ tự (giá trị num) nhỏ nhất có thể đi đến được từ u bằng cách đi xuôi xuống theo các cạnh nét liền (các cung trên cây DFS) và kết thúc đi ngược lên không quá 1 lần theo cạnh nét đứt. Ngoài ra ta cũng có thể hiểu ý nghĩa của]low[u] là thứ tự thăm của đỉnh có thứ tự thăm sớm nhất nằm trong cây con gốc u.

II. ỨNG DỤNG CỦA THUẬT TOÁN TARJAN

1. Tìm thành phần liên thông mạnh trong đồ thị có hướng

Bài toán: Cho đồ thị $G(V,E)$ có hướng n ($1 \leq n \leq 10^4$) đỉnh m ($1 \leq m \leq 10^5$) cung. Hãy đếm số thành phần liên thông mạnh của G .

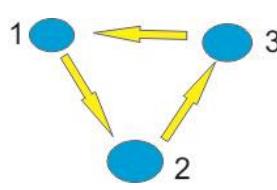
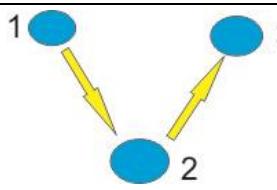
Input

+ Dòng đầu tiên là n,m.

+ M dòng tiếp theo mô tả một cung của G.

Output

Gồm một dòng duy nhất là số TPLT mạnh.

input	output
3 3 1 2 2 3 3 1	1 
3 2 1 2 2 3	3 

```
#include <bits/stdc++.h>
using namespace std;
const int N=10002;
int n,m;
vector<int> a[N];
int num[N],low[N];
int dd[N];
int sl=0;
int d=0;
stack <int> st;
void nhap()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        int u,v;
        cin>>u>>v;
        a[u].push_back(v);
    }
}
void tarjan(int u)
{
    dd[u]=1;
```

```

st.push(u);
num[u]=low[u]=++sl;
for(auto v:a[u])
{
    if(num[v])
        low[u]=min(low[u],num[v]);
    else
    {
        tarjan(v);
        low[u]=min(low[u],low[v]);
    }
}
if(num[u]==low[u])
{
    d++;
int tg;
do
{
    tg=st.top();
    st.pop();
    num[tg]=low[tg]=INT_MAX;
} while(tg!=u);
}
}
main()
{
nhap();
for(int i=1;i<=n;i++)
if(num[i]==0) tarjan(i);
cout<<d;
}

```

Thuật toán:

Ta thấy như sau : Thuật toán Tarjan dựa vào thuật toán DFS để xây dựng

Đầu tiên: Ta duyệt từng đỉnh: ở đây u là đỉnh bắt đầu nếu u chưa được duyệt thì ta dùng mảng dd đánh dấu u đã duyệt sau đó đi kiểm duyệt đỉnh v kề với u.

Có 2 trường hợp của v: Nếu v chưa duyệt ta tiến hành duyệt v và lúc này:

$\text{Low}[u]=\min(\text{low}[u], \text{low}[v])$ // nghĩa là low sẽ là mảng để lưu thứ tự duyệt(duyệt đỉnh u) nhỏ nhất của cây DFS xuất phát từ đỉnh u.

Ngược lại nếu duyệt rồi thì u lúc này là đỉnh trước đỉnh v hiển nhiên giá trị :

`Low[u]=min(low[u],num[v]) // bằng min của low tại vị trí duyệt đó với num của số ta duyệt trước đó.`

Nếu `low[u]=num[u]` ta đếm được 1 TPLTM,

Vấn đề ở đây làm sao ta loại được các đỉnh vừa được đếm TPLTM đi thì ta đã dùng 1 mảng stack mảng này lưu các đỉnh đã duyệt rồi //

Hiển nhiên để bỏ qua nó ta dùng 1 biến trung gian để lấy các đỉnh đó ra khỏi stack và gán cho giá trị INT_MAX để loại bỏ (vì ta chỉ lấy min)

Ngoài ra: ở đây ta phải dùng do ... while // vì có trường hợp xét đến chính nó

`While(tg!=u)//` trường hợp bằng u ta vẫn xét và do ...while vẫn là thêm nếu trường hợp `tg=u`,

Như vậy: Đối với đồ thị có hướng thuật toán Tarjan đếm TPLTM trong đồ thị có hướng với độ phức tạp ($O(n+m)$)

2. Thuật toán Tarjan đếm cầu và khớp trong đồ thị vô hướng

Bài toán:

Xét đơn đồ thị vô hướng $G = (V, E)$ có $n(1 \leq n \leq 10000)$ đỉnh và $m(1 \leq m \leq 50000)$ cạnh. Người ta định nghĩa một đỉnh gọi là khớp nếu như xoá đỉnh đó sẽ làm tăng số thành phần liên thông của đồ thị. Tương tự như vậy, một cạnh được gọi là cầu nếu xoá cạnh đó sẽ làm tăng số thành phần liên thông của đồ thị.

Vấn đề đặt ra là cần phải đếm tất cả các khớp và cầu của đồ thị G .

Input

+Dòng đầu: chứa hai số tự nhiên n, m .

+M dòng sau mỗi dòng chứa một cặp số (u, v) ($u \leftrightarrow v$, $1 \leq u \leq n$, $1 \leq v \leq m$)

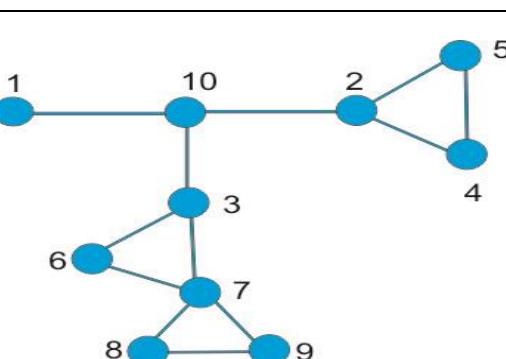
Output

Gồm một dòng duy nhất ghi hai số, số thứ nhất là số khớp, số thứ hai là số cầu của G

Example:

10 12
1 10
10 2
10 3
2 4
4 5
5 2
3 6
6 7
7 3
7 8
8 9
9 7

4 3



Bài toán này có 3 cầu và 4 khớp

```

#include <bits/stdc++.h>
using namespace std;
const int N=10002;
int n,m;
vector<int> a[N];
int num[N],low[N];
int dd[N];
int sl=0,d=0;
bool node[N];
int edge=0;
stack <int> st;

void nhap()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        int u,v;
        cin>>u>>v;
        a[u].push_back(v);
        a[v].push_back(u);
    }
}
void tarjan(int u, int p)
{
    int child=0;
    num[u]=low[u]=++sl;
    for(auto v:a[u])
        if(v!=p)
        {
            if(num[v])
                low[u]=min(low[u],num[v]);
            else
            {
                tarjan(v,u);
                child++;
                low[u]=min(low[u],low[v]);
                if(low[v]>=num[v]) edge++;
                if(u==p)
                {
                    if(child>=2)
                        node[u]=true;
                }
            }
        }
}

```

```
    }
} else
{
    if(low[v]>=num[u])
        node[u]=true;
}
}
}

main()
{
    nhap();
    for(int i=1;i<=n;i++)
        if(num[i]==0) tarjan(i,i);
    int dem=0;
    for(int i=1;i<=n;i++)
        if(node[i]) dem++;
    cout<<dem<<" "<<edge;
}
```