# THUẬT TOÁN ỨNG DỤNG

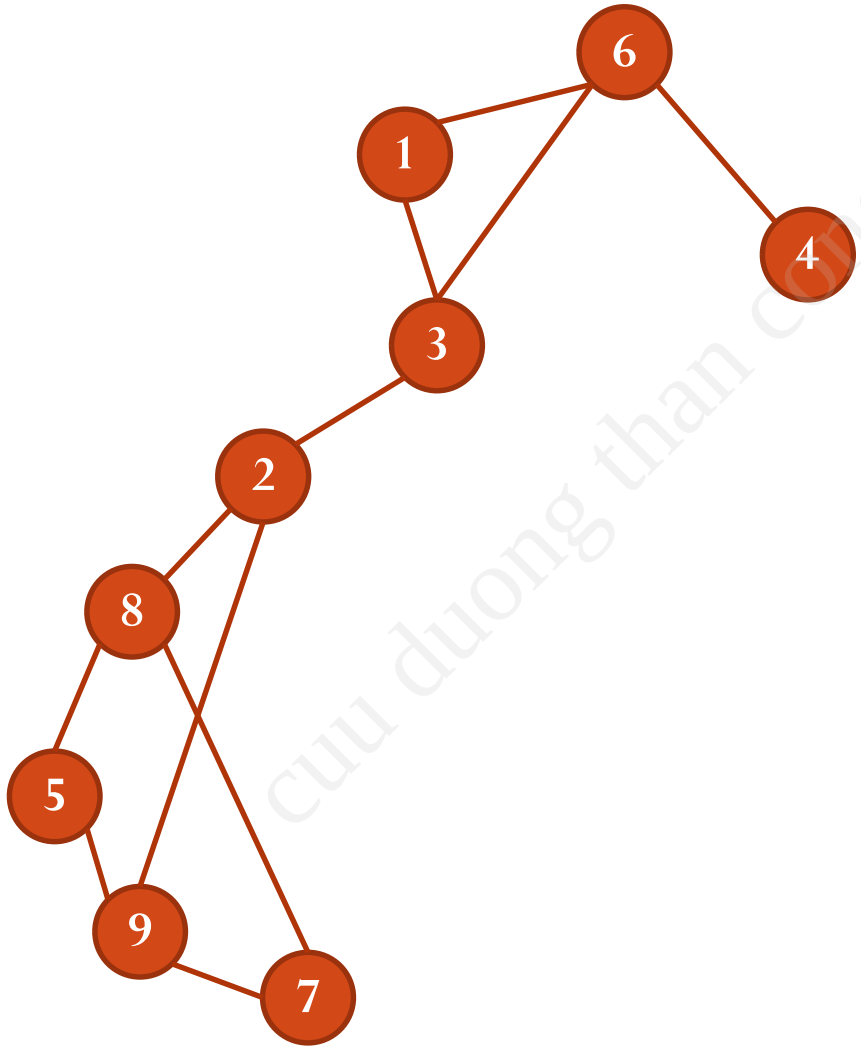
## Tarjan DFS algorithm for finding Bridges and Articulation Points

Phạm Quang Dũng
Bộ môn KHMT
dungpq@soict.hust.edu.vn

# Duyệt theo chiều sâu

- Cây DFS
  - DFS xuất phát từ một đỉnh cho phép thăm các đỉnh con cháu của nó trên cây DFS
- Cấu trúc dữ liệu duy trì
  - num[v]: thời điểm đỉnh v được thăm
  - low[v]: giá trị num nhỏ nhất của các đỉnh x sao cho có cạnh ngược (u,x) với u là 1 đỉnh con cháu nào đó của v

# DFS(6)



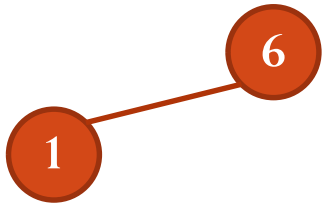

3

# DFS(6)

6

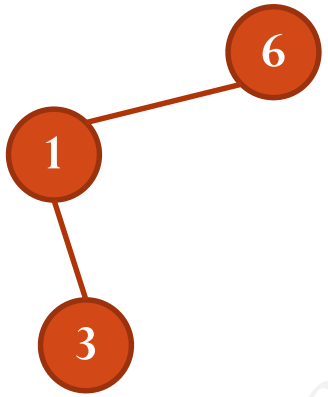
num[6] = 1, low[6] = 1

# DFS(6)



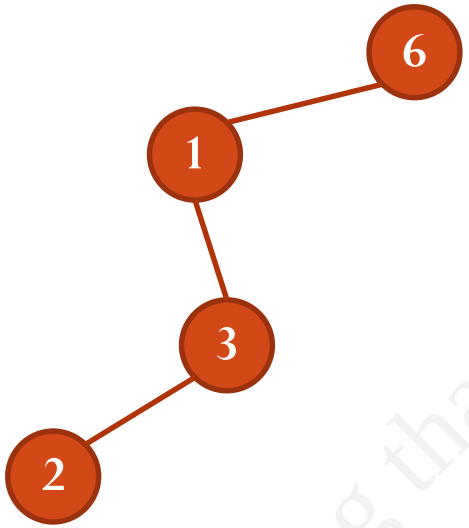

num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2

# DFS(6)



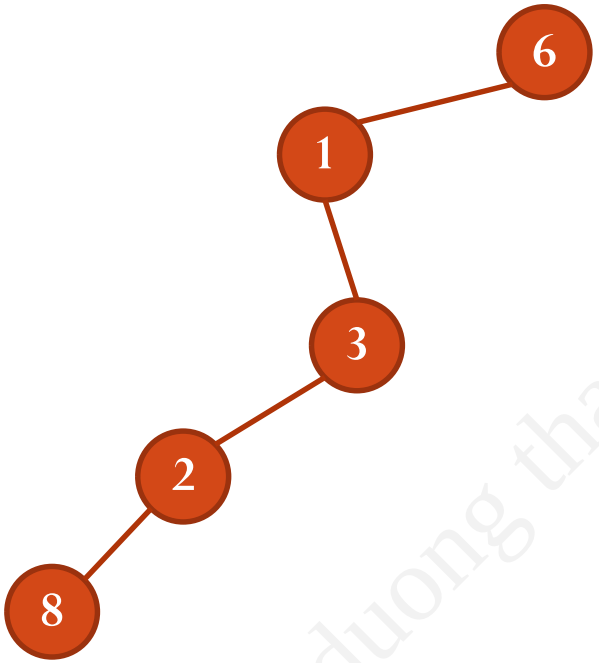

num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3

# DFS(6)



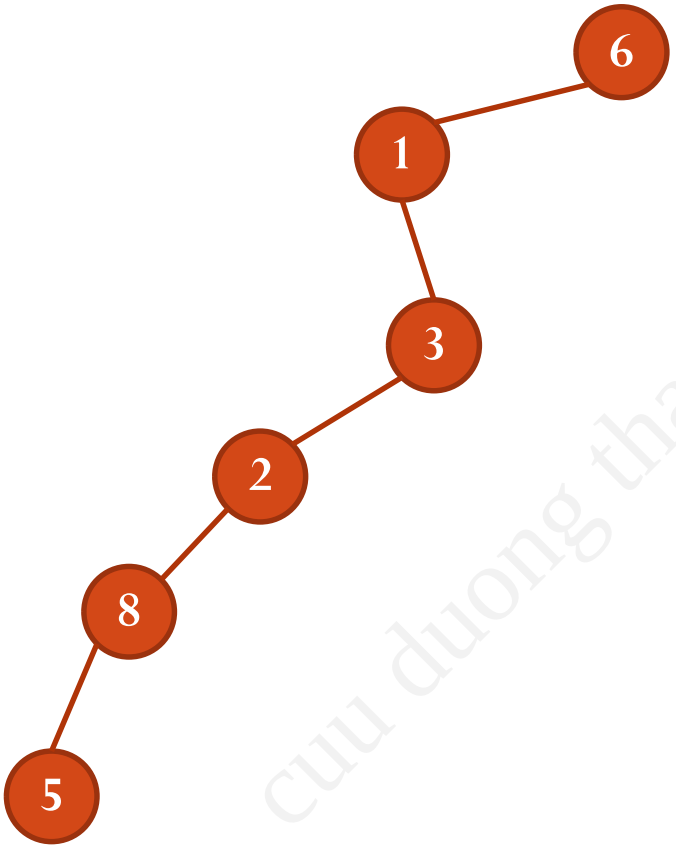

num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4

# DFS(6)



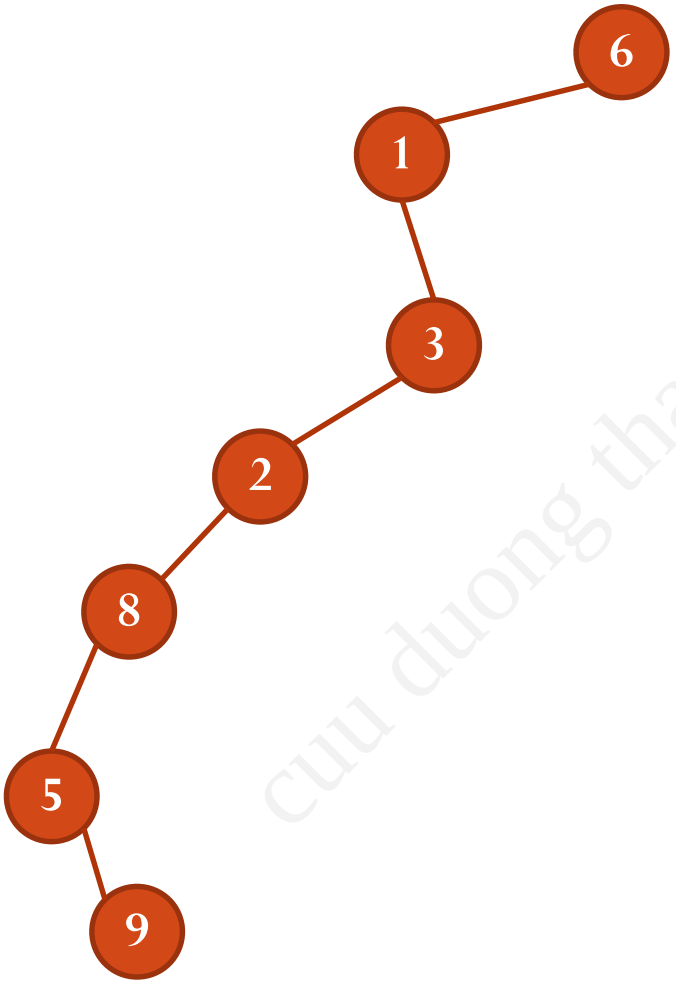

num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = 5

# DFS(6)



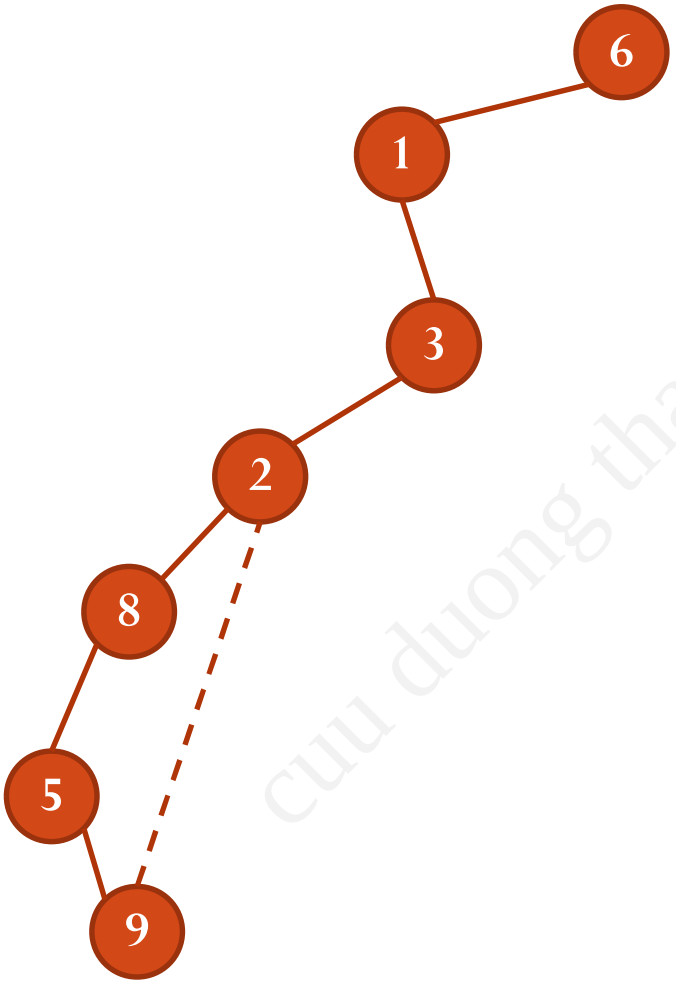

num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = 5
num[5] = 6, low[5] = 6

# DFS(6)



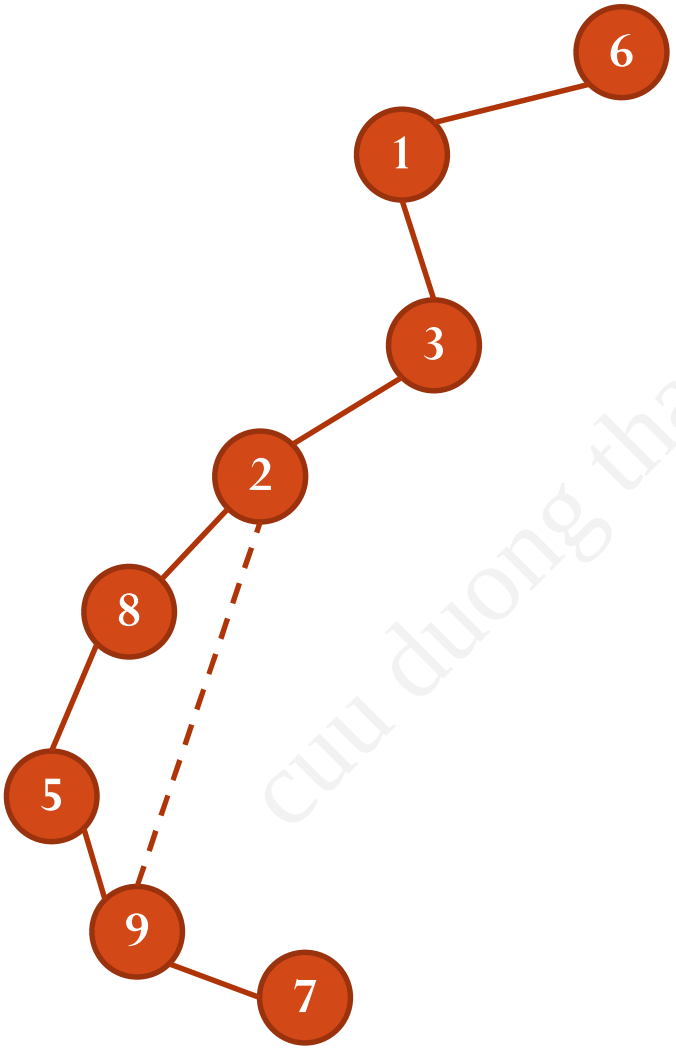

num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = 5
num[5] = 6, low[5] = 6
num[9] = 7, low[9] = 7

# DFS(6)



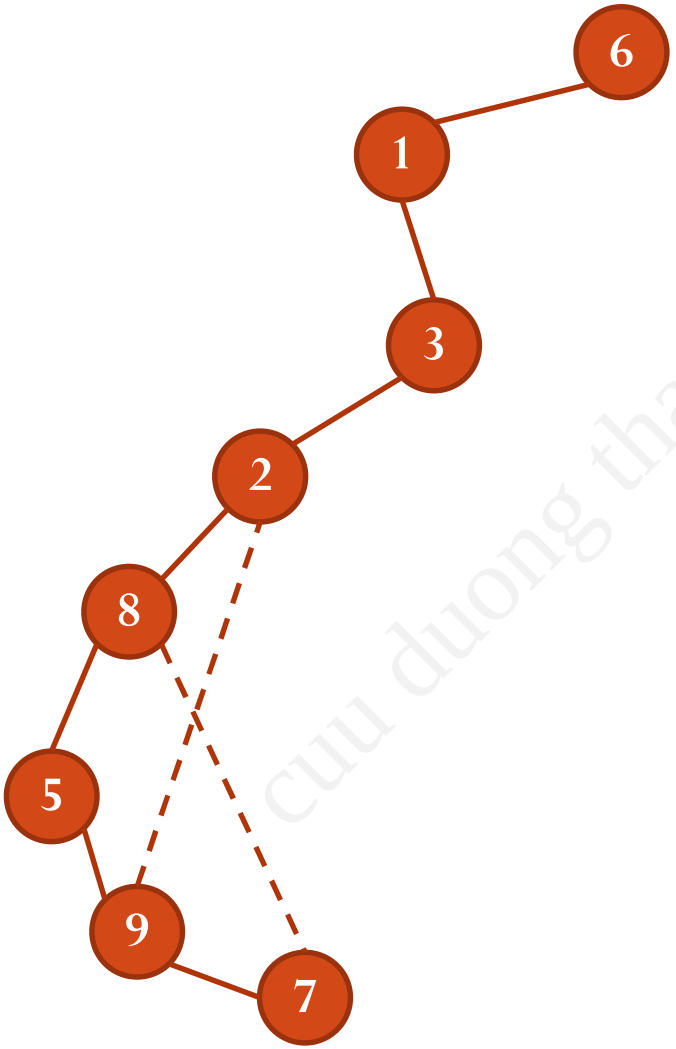

num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = 5
num[5] = 6, low[5] = 6
num[9] = 7, low[9] = num[2] = 4

# DFS(6)



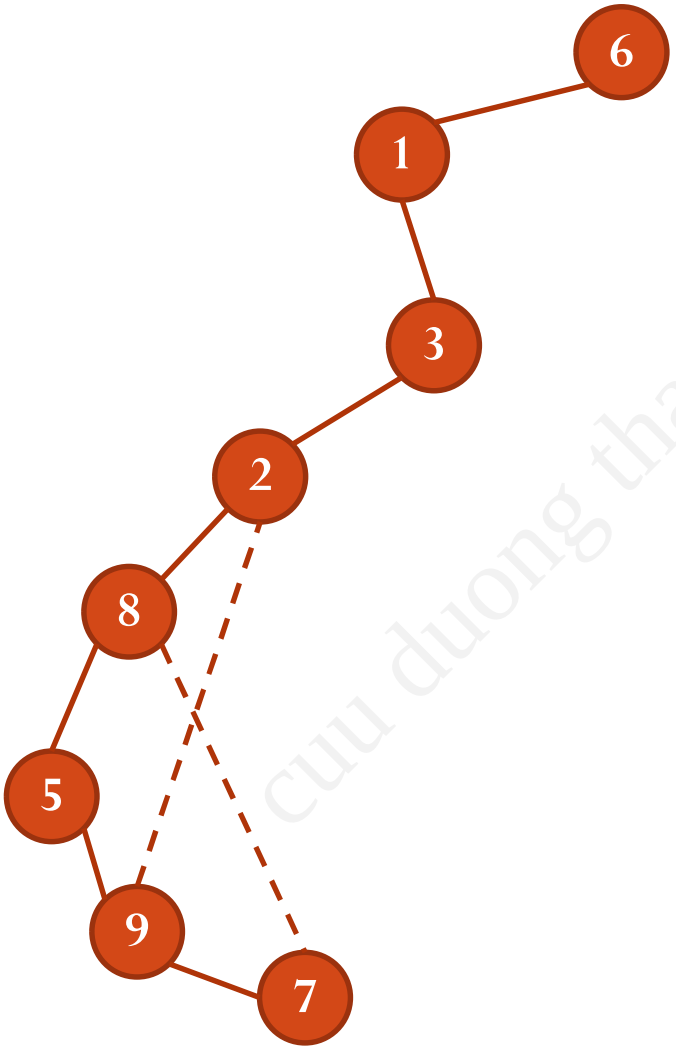

num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = 5
num[5] = 6, low[5] = 6
num[9] = 7, low[9] = num[2] = 4
num[7] = 8, low[7] = 8

# DFS(6)



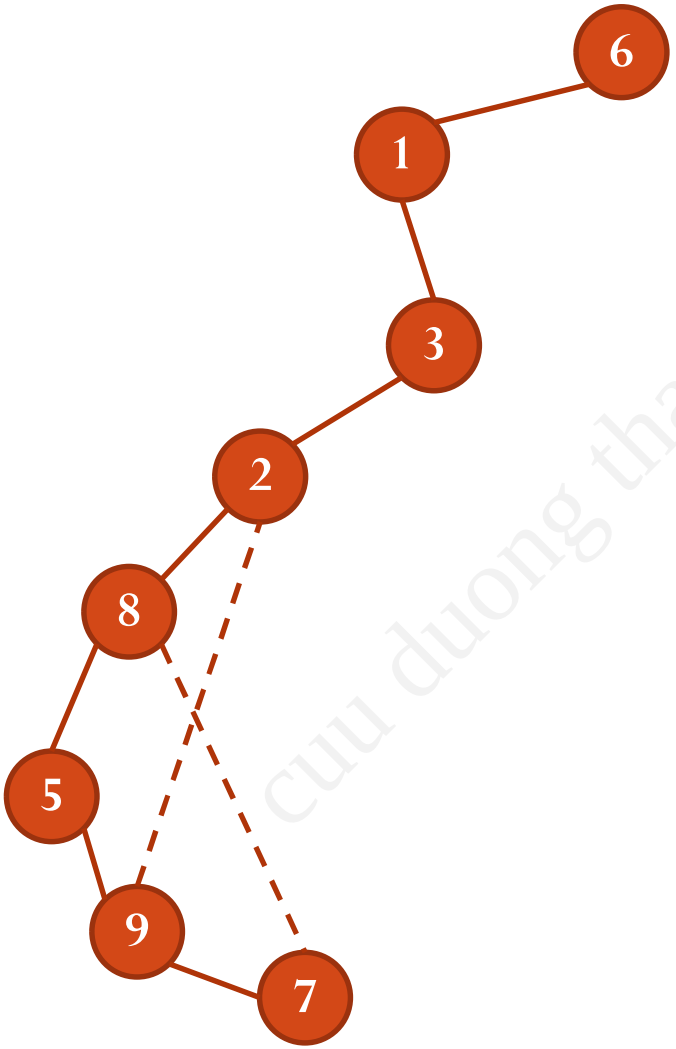

num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = 5
num[5] = 6, low[5] = 6
num[9] = 7, low[9] = num[2] = 4
num[7] = 8, low[7] = num[8] = 5

# DFS(6)



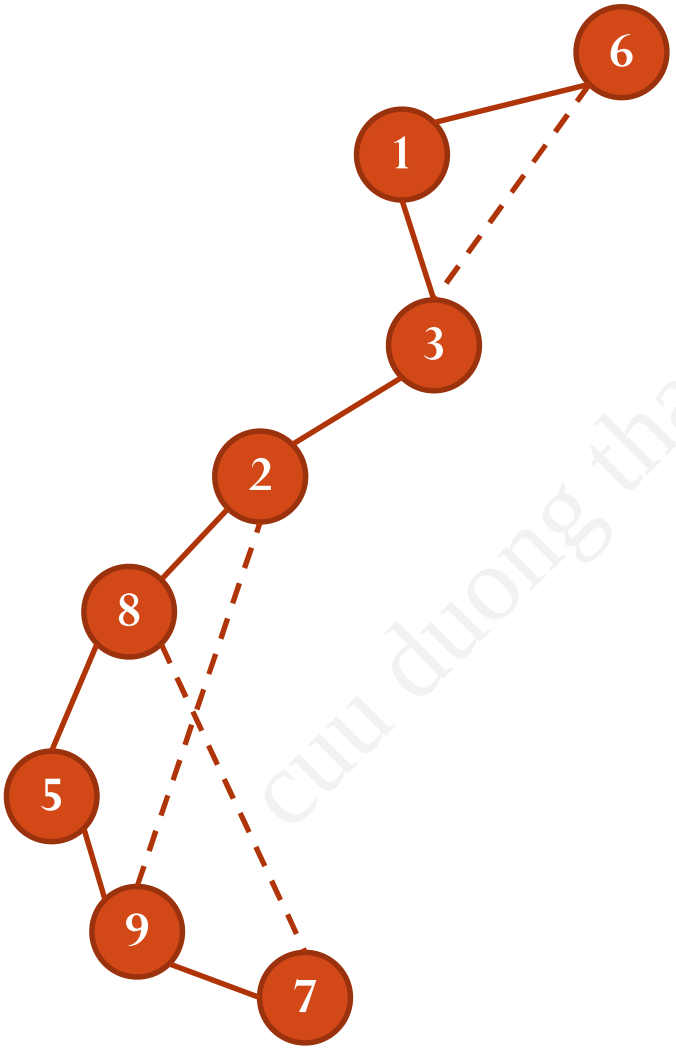

num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = 5
num[5] = 6, low[5] = low[9] = 4
num[9] = 7, low[9] = num[2] = 4
num[7] = 8, low[7] = num[8] = 5

# DFS(6)



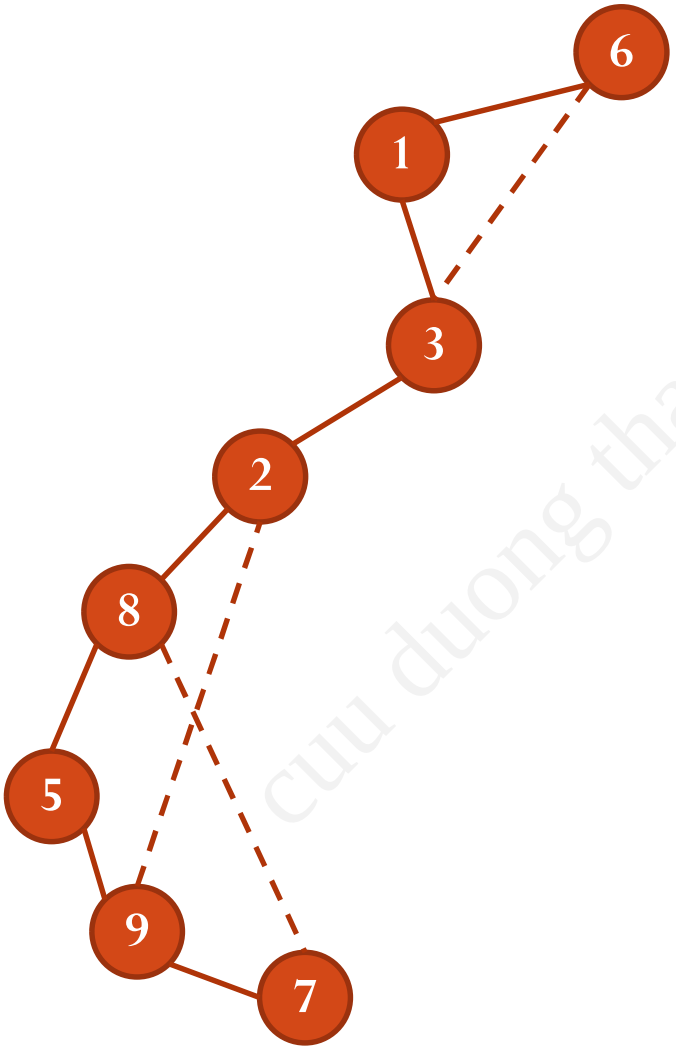

num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = 3
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = low[5] = 4
num[5] = 6, low[5] = low[9] = 4
num[9] = 7, low[9] = num[2] = 4
num[7] = 8, low[7] = num[8] = 5

# DFS(6)



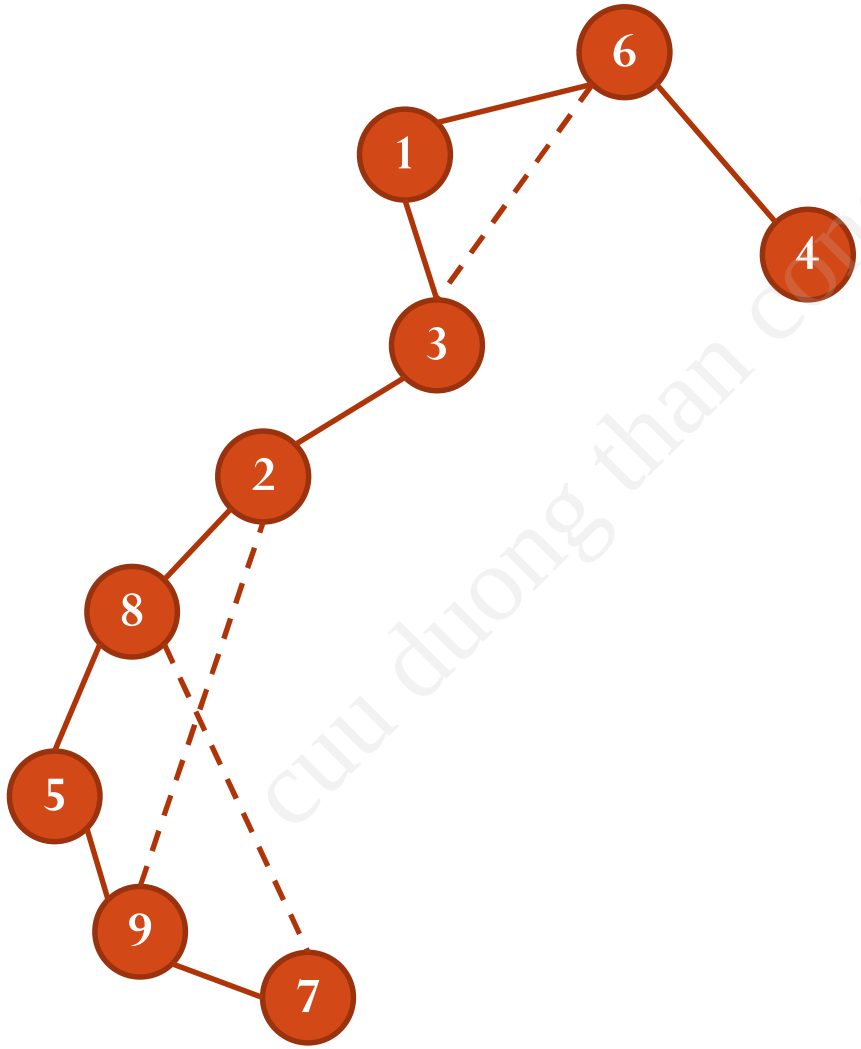

num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = 2
num[3] = 3, low[3] = num[6] = 1
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = low[5] = 4
num[5] = 6, low[5] = low[9] = 4
num[9] = 7, low[9] = num[2] = 4
num[7] = 8, low[7] = num[8] = 5

# DFS(6)




num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = low[3] = 1
num[3] = 3, low[3] = num[6] = 1
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = low[5] = 4
num[5] = 6, low[5] = low[9] = 4
num[9] = 7, low[9] = num[2] = 4
num[7] = 8, low[7] = num[8] = 5

# DFS(6)




num[6] = 1, low[6] = 1
num[1] = 2, low[[1] = low[3] = 1
num[3] = 3, low[3] = num[6] = 1
num[2] = 4, low[2] = 4
num[8] = 5, low[8] = low[5] = 4
num[5] = 6, low[5] = low[9] = 4
num[9] = 7, low[9] = num[2] = 4
num[7] = 8, low[7] = num[8] = 5
num[4] = 9, low[4] = 9

# Sample code

```
#include <bits/stdc++.h>
using namespace std;
const int N = 10000;
int n,m;
vector<int> A[N];
bool visited[N];
int num[N];
int low[N];
int t;
vector<pair<int,int> > bridges;
void input(){
    ios_base::sync_with_stdio(0); cin.tie(0);
    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);
    }
}
```

# Sample code

```
void dfs(int s, int ps){
    // DFS from s with ps is the parent of s in the DFS tree
    t++;
    num[s] = t;
    low[s] = num[s];
    visited[s] = true;
    for(int i = 0;i < A[s].size(); i++){
        int v = A[s][i];
        if(v == ps) continue;
        if(visited[v]){
            low[s] = min(low[s],num[v]);
        }else{
            dfs(v,s);
            low[s] = min(low[s],low[v]);
            if(low[v] > num[s]){
                // discover a bridge (s,v)
                bridges.push_back(make_pair(s,v));
            }
        }
    }
}
```

# Sample code

```
void init(){
    for(int v = 1; v <= n; v++) visited[v] = false;
}
void solve(){
    init();
    t = 0;
    for(int s = 1; s <= n; s++){
        if(!visited[s]){
            dfs(s,-1);
        }
    }
    cout << "bridges = ";
    for(int i = 0; i < bridges.size(); i++){
        cout << "(" << bridges[i].first << "," << bridges[i].second << ") ";
    }
}
int main(){
    input();
    solve();
}
```