HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Applied Algorithm Lab

Bridges

ONE LOVE. ONE FUTURE.

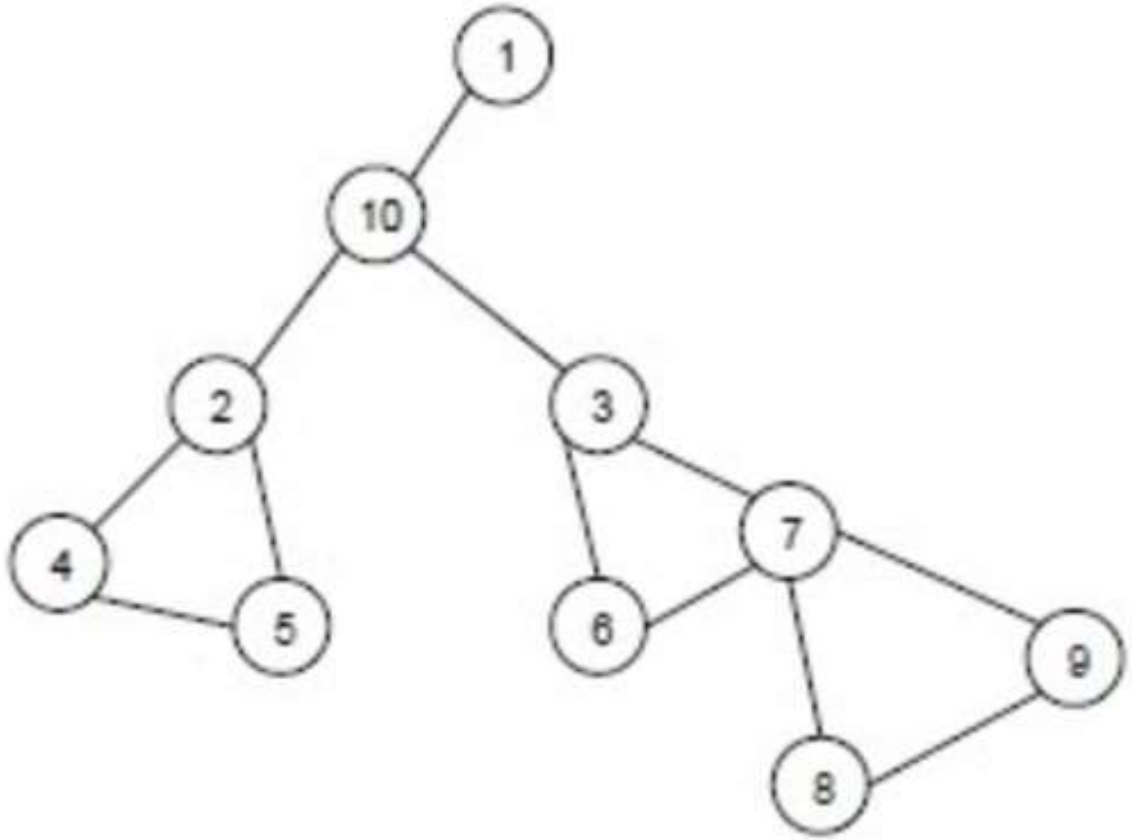
Bridges

- Find articulation points and bridges of an undirected graph.
  - Definition: Remove articulation points/bridges -> the number of connected components in the graph increase
- **Input:** Edge list
  - Line 1 contains N, M
  - M lines follow, containing a pair of 2 integers a, b which is an undirected edge from a to b.
- **Output:**
  - Number of articulation points and the number of bridges.

Bridges

- Example

Input	Output
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 Explain: Articulation points: 10, 2, 3, 7 Bridges: (2-10), (10-3), (10-1)



Bridges

- Idea to solve: DFS
  - Store graph in adjacency list: vector<vector<int>> adj(N);

Bridges

- **DFS**
- DFS tree and Num, Low structure
- Finding bridges
- Finding articulation points

Depth First Search

- DFS is a basic graph traversing technique (to visit all nodes and edges in graph).
  - DFS can answer if there exist a path from node u to node v in graph or not, and show the path if exists.
  - DFS can also answer from u, we can goes to which nodes on graph G.
- The traversing order in DFS follow LIFO – Last In First Out mechanism, start from some beginning node u.
  - We may use backtracking recusion or stack
- Complexity: $O(|V| + |E|)$, where V is node set and E is edge set of G, since each node and each edge is visited once.

Programming

- Graph G = (V, E) represented by adjacency lists A[1…n]
- Marking array:
  - visited[u] = true, if u is visited
  - visited[u] = false, otherwise

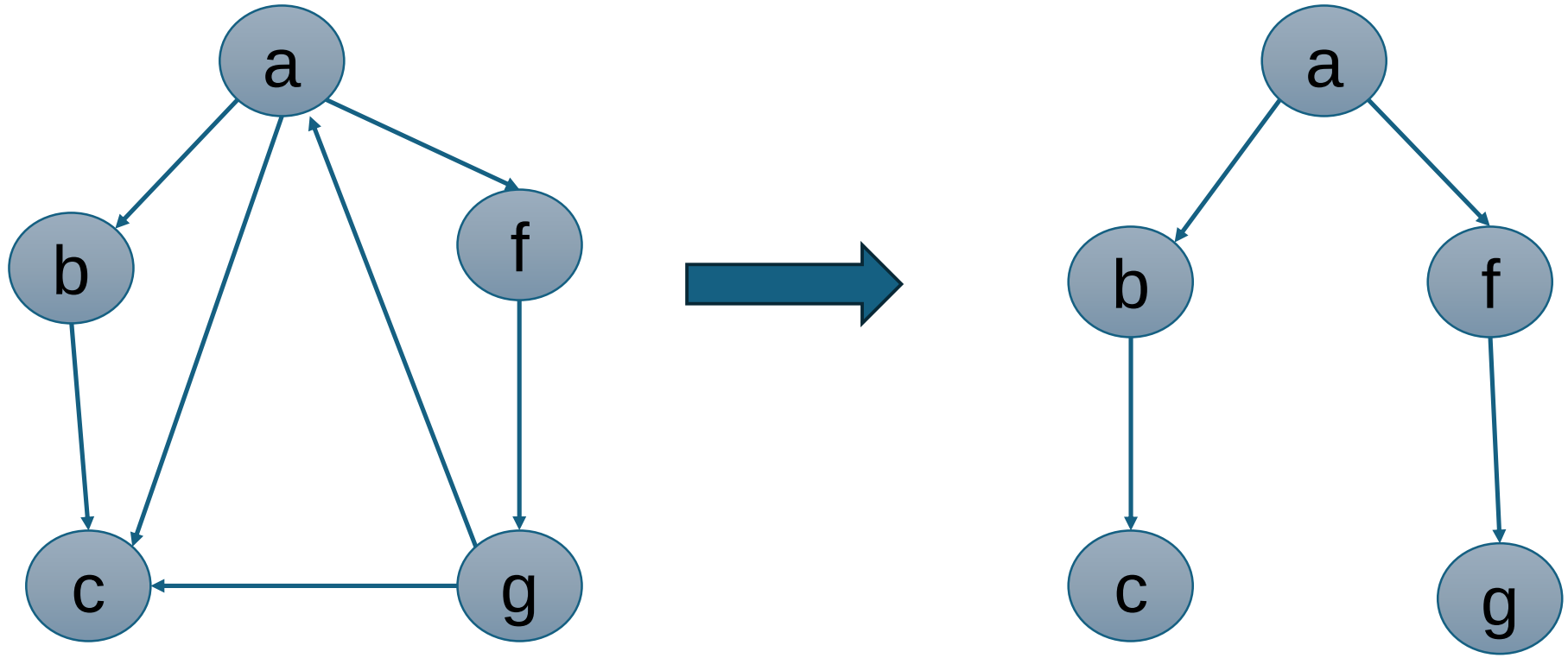
```
DFS(u) {
   visit(u); // assign visited[u] = true
   for v in A[u] do {
      if not visited[v] then {
          DFS(v);
      }
   }
}
DFS(){
  for u in V  do {   visited[u] = false;  }
  for u in V do {
     if not visited[u] then
        DFS(u);
  }
}
```

Bridges

- DFS
- **DFS tree and Num, Low structure**
- Finding bridges
- Finding articulation points

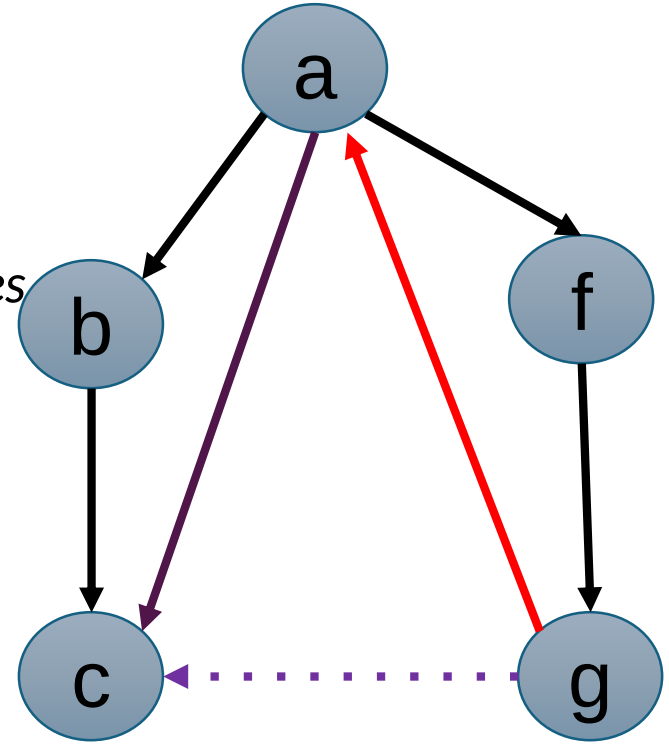
DFS tree

- Trace of the DF search will construct a tree, called DFS tree



DFS tree

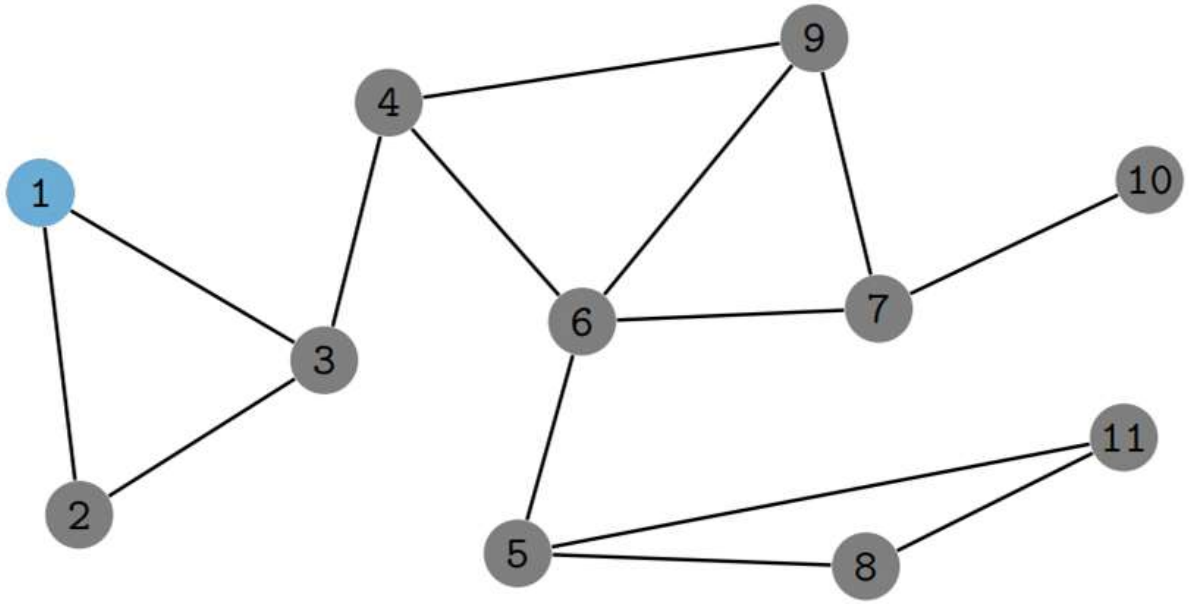
- Trace of the DF search will construct DFS tree
- Some type of edge in DFS:
  - Tree Edge: edge in DFS tree, *e.g. black edges in figure*
  - Back Edge: edge from descendants to ancestors, *e.g. red edges*
  - Forward Edge: edge from ancestors to descendants, *e.g. blue edges*
  - Crossing Edge: edge between non-relational nodes, *e.g. purple edges*



DFS tree: Num and Low structure

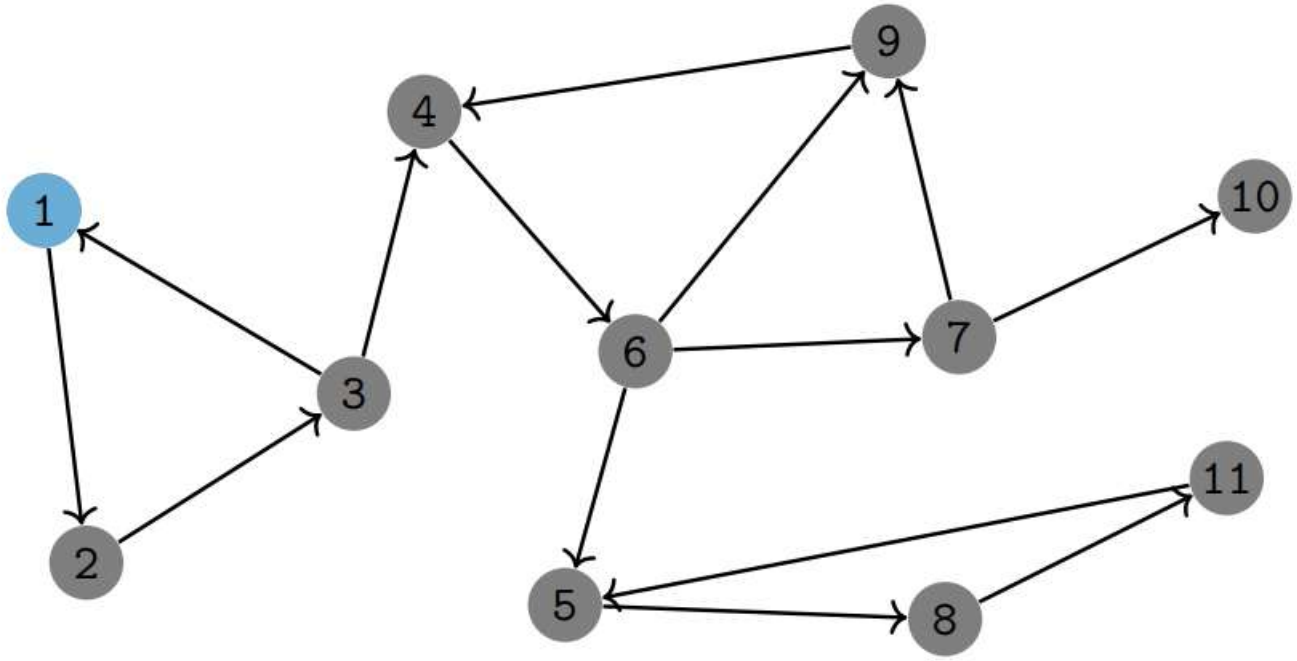
- Defining 2 arrays Num and Low for each node of DFS tree.
- $Num[u]$: visiting order of u in DFS traversal
- $Low[u]$: the minimum value of:
  - $Num[v]$ if (v,u) is back edge
  - $Low[v]$:

    ~minimum num[v] where v and descendents can visit



i	1	2	3	4	5	6	7	8	9	10	11
Num[i]	1	2	3	4	6	5	9	7	10	11	8
Low[i]	1	1	1	4	6	4	4	6	4	11	6

Example



i	1	2	3	4	5	6	7	8	9	10	11
Num[i]	1	2	3	4	6	5	9	7	10	11	8
Low[i]	1	1	1	4	6	4	4	6	4	11	6

DFS and Num, Low programming by recursion

- p[v]: father of v in DFS
- Num[u] = 0: node u is not visited
- Num[u] > 0: node u is visited and Num[u] is the order

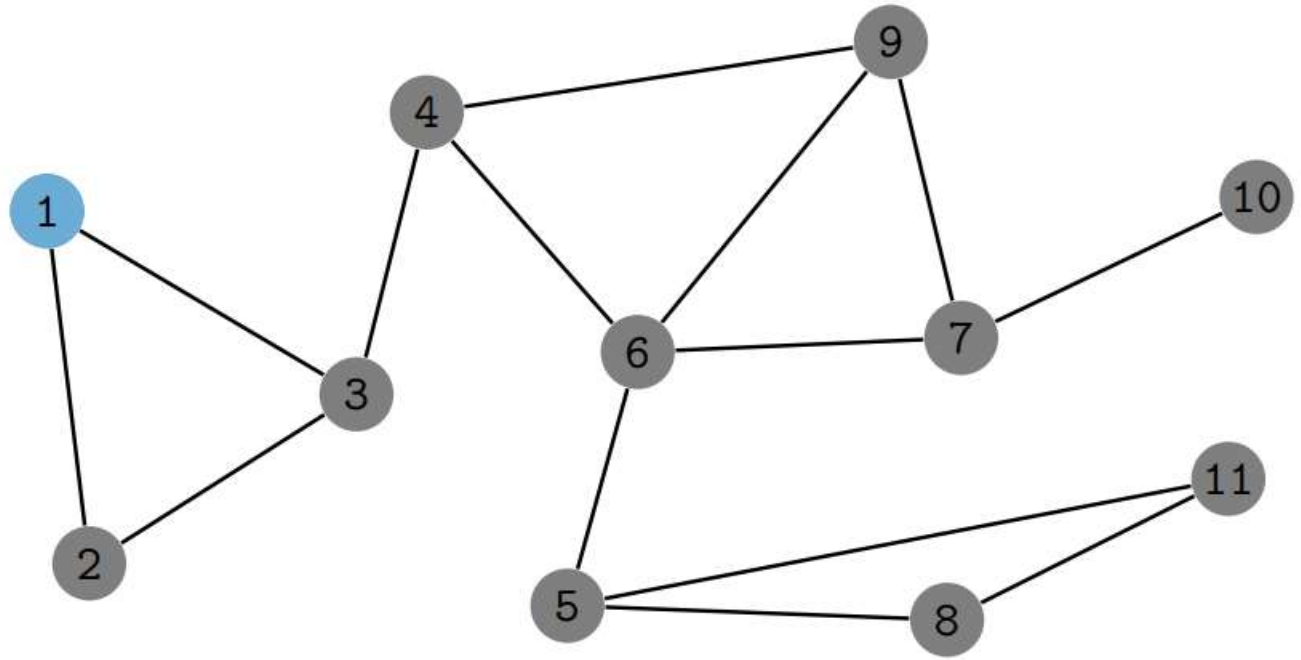
```
DFS(u) {
   T += 1; Num[u] = T; Low[u] = T;
   for v in A[u] do {
      if v = p[u] continue;
      if Num[v] > 0 then { // v was visited
         Low[u] = min(Low[u], Num[v]);
      } else {
         p[v] = u;
         DFS(v);
         Low[u] = min(Low[u], Low[v]);
      }
   }
}
```

Bridges

- DFS
- DFS tree and Num, Low structure
- **Finding bridges**
- Finding articulation points

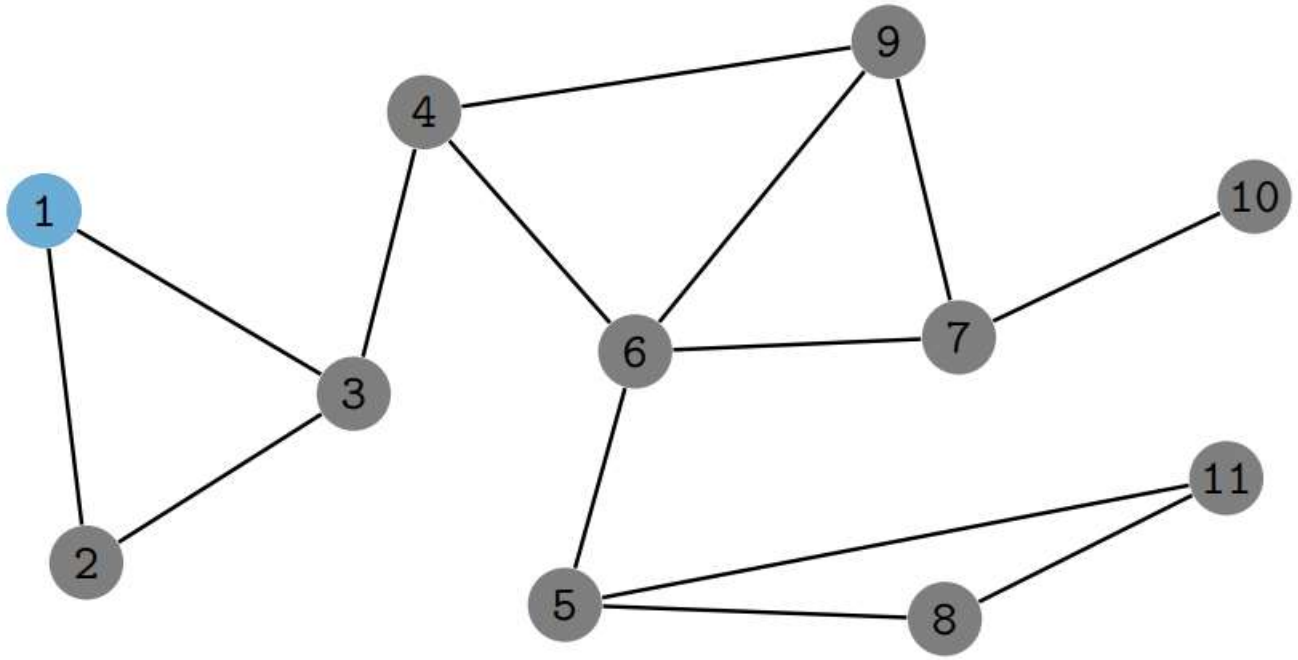
Finding bridges

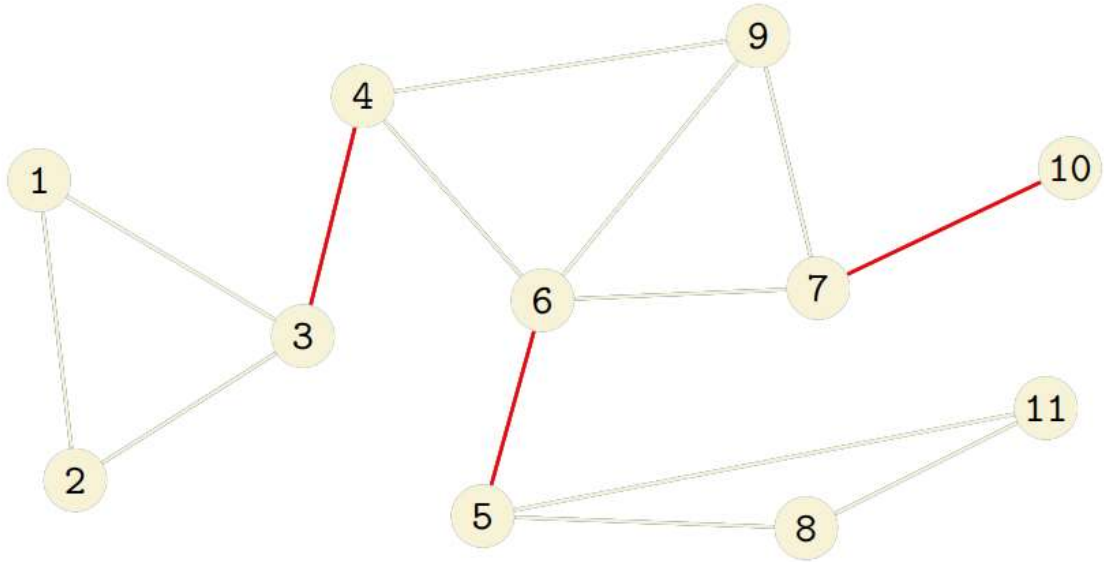
- **Definition**: Bridge is an edge where if we remove it, the number of connected components in the graph increases.
- **Note**: A forwarde edge (u, v) is an edge if and only if $Low[v] > Num[u]$



Finding bridges

- **Note**: A forwarde edge (u, v) is an edge if and only if $Low[v] > Num[u]$





i	1	2	3	4	5	6	7	8	9	10	11
Num[i]	1	2	3	4	6	5	9	7	10	11	8
Low[i]	1	1	1	4	6	4	4	6	4	11	6

Programming

- p[v]: father of v in DFS
- Num[u] = 0: node u is not visited
- Num[u] > 0: node u is visited and Num[u] is the order
- update low[u]
- check bridge (u,v)

```
DFS(u) {
   T += 1; Num[u] = T; Low[u] = T;
   for v in A[u] do {
      if v = p[u] continue;
      if Num[v] > 0 then { // v was visited
         Low[u] = min(Low[u], Num[v]);
      } else {
         p[v] = u;
         DFS(v);
         Low[u] = min(Low[u], Low[v]);
         if Low[v] > Num[u] then (u,v) is a bridge;
      }
   }
}
```

Programming - Implementation

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

const int N = 1e5 + 2;

int timeDfs = 0, bridge = 0, cntJoint = 0;
vector<vector<int>> adj(N);  // Danh sách kề của đồ thị
vector<int> low(N);          // low[u] là số hiệu nhỏ nhất của các đỉnh có thể đến được từ u
                             // bằng cả cây DFS và một số cạnh ngoài cây DFS
vector<int> num(N);          // num[u] là số hiệu của đỉnh u trong quá trình duyệt DFS
vector<bool> joint(N);       // joint[u] là true nếu u là điểm phân cực
```

Ý tưởng cài đặt bằng đệ quy - Implementation

```
13  void dfs(int u, int pre) {
14      int child = 0;  // Số lượng con trực tiếp của đỉnh u trong cây DFS
15      num[u] = low[u] = ++timeDfs;
16      for (auto v : adj[u]) {
17          if (v == pre) continue;
18          if (!num[v]) {  // Nếu đỉnh v chưa được duyệt, thực hiện DFS trên nó
19              dfs(v, u);
20              low[u] = min(low[u], low[v]);
21              if (low[v] == num[v]) bridge++;  // Cập nhật số cầu
22              child++;
23              if (u == pre) {  // Nếu u là đỉnh gốc của cây DFS
24                  if (child > 1) joint[u] = true;  // Nếu có nhiều hơn một con, u là điểm phân cực
25              } else {
26                  if (low[v] >= num[u]) joint[u] = true;  // Nếu không, xét điều kiện để u là điểm phân cực
27              }
28          } else {
29              low[u] = min(low[u], num[v]);  // Nếu v đã được duyệt, cập nhật low[u]
30          }
31      }
32  }
```

Bridges

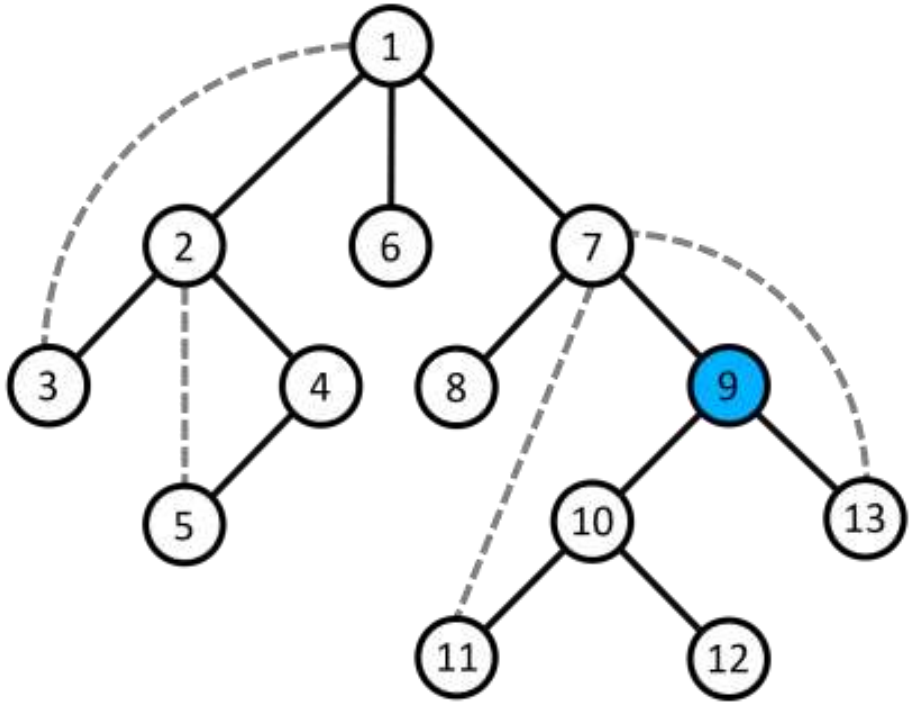
- DFS
- DFS tree and Num, Low structure
- Finding bridges
- **Finding articulation points**

Finding articulation points

- **Definition**: In undirected graph, an articulation point is a node where if we remove it and its adjacent edge, the number of connected components in graph increase.
- **Note**: Node u is an articulation point iff:
  - Node u is the root of DFS tree and

$$Low[v] \geq Num[u]$$

    where v is a direct child of u in DFS tree
  - Or u is the root of DFS tree having at least 2 direct children



Finding articulation points - Implementation

```
35  int main() {
36      ios_base::sync_with_stdio(0);
37      cin.tie(0);
38      cout.tie(0);
39      int n, m;
40      cin >> n >> m;
41      int x, y;
42      for (int i = 1; i <= m; i++) {
43          cin >> x >> y;
44          adj[x].push_back(y);
45          adj[y].push_back(x);
46      }
47      for (int i = 1; i <= n; i++) {
48          if (!num[i])
49              dfs(i, i);
50      }
51      for (int i = 1; i <= n; i++) {
52          cntJoint += joint[i];  // Đếm số điểm phân cực
53      }
54      cout << cntJoint << " " << bridge;  // In kết quả
55      return 0;
56  }
```

THANK YOU !