### Week 32: Advanced Graph Algorithms – Bridges, Articulation Points & 2-SAT

**Topics:** - Finding Bridges and Articulation Points (Tarjan’s Algorithm) - Strongly Connected Components (Kosaraju / Tarjan) - 2-SAT Problem and Implication Graphs - Biconnected Components - Low-Link Values and DFS Applications - Applications: Network Reliability, Constraint Satisfaction, Graph Connectivity

**Weekly Tips:** - Bridges are edges whose removal increases the number of connected components. - Articulation points are vertices whose removal disconnects the graph. - Tarjan’s algorithm computes low-link values for bridges and articulation points. - SCC algorithms help solve 2-SAT by detecting cycles in implication graphs. - Biconnected components help in network design and analysis.

**Problem 1: Bridges in a Graph** **Link:** [CSES Road Reparation](https://cses.fi/problemset/task/1673/) **Difficulty:** Advanced

**C++ Solution with Explanation Comments:**

#include <bits/stdc++.h>  
using namespace std;  
vector<int> tin, low;  
vector<vector<int>> adj;  
vector<pair<int,int>> bridges;  
int timer;  
void dfs(int u,int p){  
 tin[u]=low[u]=timer++;  
 for(int v:adj[u]){  
 if(v==p) continue;  
 if(tin[v]!=-1) low[u]=min(low[u],tin[v]);  
 else{  
 dfs(v,u);  
 low[u]=min(low[u],low[v]);  
 if(low[v]>tin[u]) bridges.push\_back({u,v});  
 }  
 }  
}  
int main(){  
 int n,m; cin>>n>>m;  
 adj.assign(n,{}); tin.assign(n,-1); low.assign(n,-1);  
 for(int i=0;i<m;i++){ int u,v; cin>>u>>v; u--; v--; adj[u].push\_back(v); adj[v].push\_back(u); }  
 timer=0;  
 for(int i=0;i<n;i++) if(tin[i]==-1) dfs(i,-1);  
 for(auto [u,v]:bridges) cout<<u+1<<' '<<v+1<<endl;  
}

**Explanation Comments:** - DFS assigns discovery time (tin) and computes low-link values. - An edge is a bridge if low[v] > tin[u]. - Detects critical edges whose removal increases components.

**Problem 2: 2-SAT Solver** **Link:** [Codeforces 2-SAT Tutorial](https://cp-algorithms.com/graph/2SAT.html) **Difficulty:** Advanced

**C++ Solution with Explanation Comments:**

#include <bits/stdc++.h>  
using namespace std;  
int n; vector<vector<int>> adj, radj;  
vector<int> order, comp; vector<bool> visited;  
void dfs1(int u){ visited[u]=true; for(int v:adj[u]) if(!visited[v]) dfs1(v); order.push\_back(u); }  
void dfs2(int u,int cl){ comp[u]=cl; for(int v:radj[u]) if(comp[v]==-1) dfs2(v,cl); }  
int main(){  
 int m; cin>>n>>m; // n variables, m clauses  
 adj.assign(2\*n,{}); radj.assign(2\*n,{});  
 for(int i=0;i<m;i++){  
 int a,b; cin>>a>>b; a--; b--;  
 adj[a^1].push\_back(b); adj[b^1].push\_back(a);  
 radj[b].push\_back(b^1); radj[a].push\_back(a^1);  
 }  
 visited.assign(2\*n,false); order.clear();  
 for(int i=0;i<2\*n;i++) if(!visited[i]) dfs1(i);  
 comp.assign(2\*n,-1); int j=0;  
 for(int i=2\*n-1;i>=0;i--) if(comp[order[i]]==-1) dfs2(order[i],j++);  
 for(int i=0;i<n;i++) if(comp[2\*i]==comp[2\*i+1]) { cout<<"NO"<<endl; return 0; }  
 cout<<"YES"<<endl;  
}

**Explanation Comments:** - Build implication graph from 2-SAT clauses. - Kosaraju’s algorithm finds SCCs. - If variable and its negation are in same SCC, no solution exists. - Efficient for constraints and logic-based graph problems.

**End of Week 32** - Advanced graph algorithms such as bridges, articulation points, SCC, and 2-SAT are essential for ACM-ICPC contests. - Practice DFS low-link, SCC, and implication graph techniques extensively.