Competitive Programming

From Problem 2 Solution in O(1)

Graph Theory

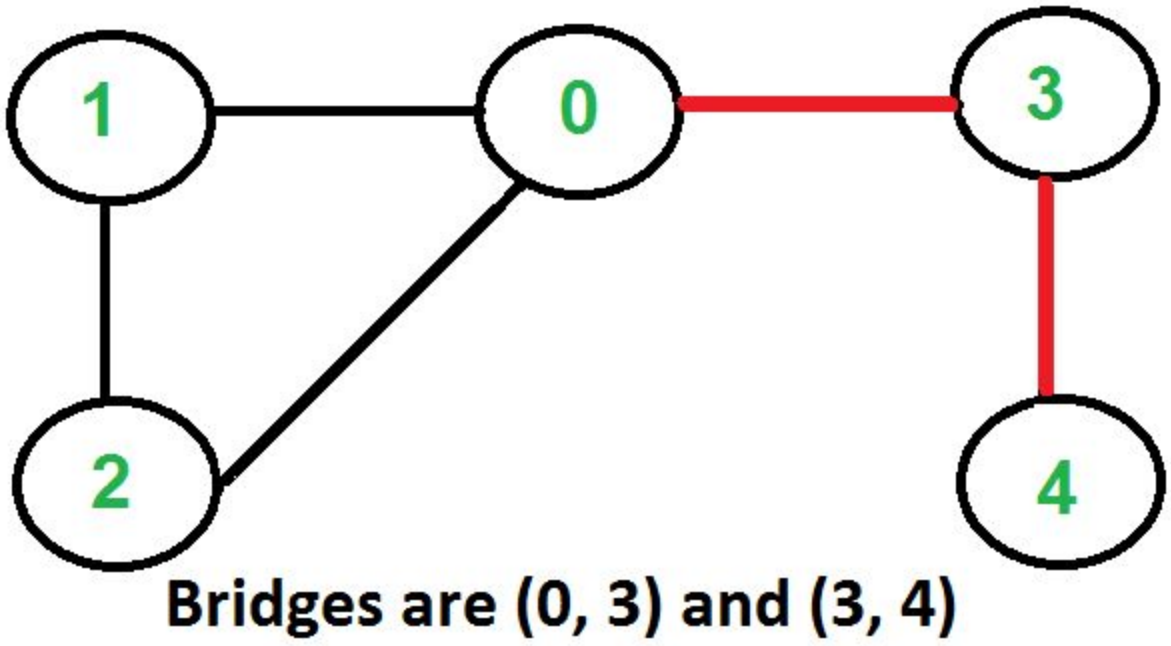
Bridges using Tarjan

Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University

Graph Bridges

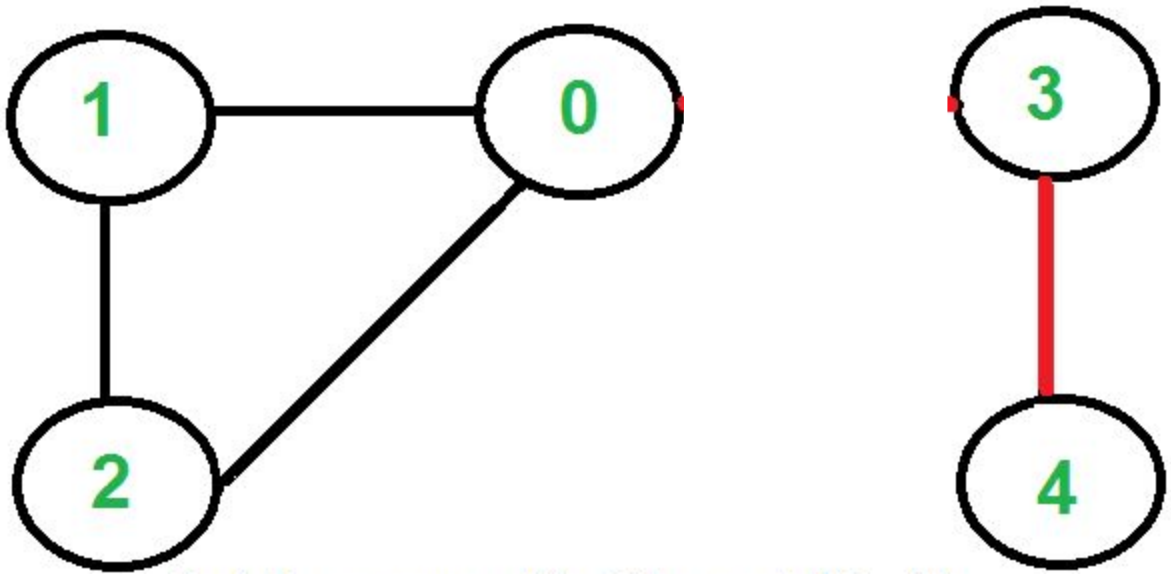
- In **undirected** graph, **removing** a bridge:
  - if graph is fully connected ⇒ disconnects it
  - if not => disconnects a connected component
  - E.g. increases overall components
  - we can compute it using brute force: remove and test connectivity.
  - But very slow
  - Let's see how Tarjan can help us

Graph Bridges



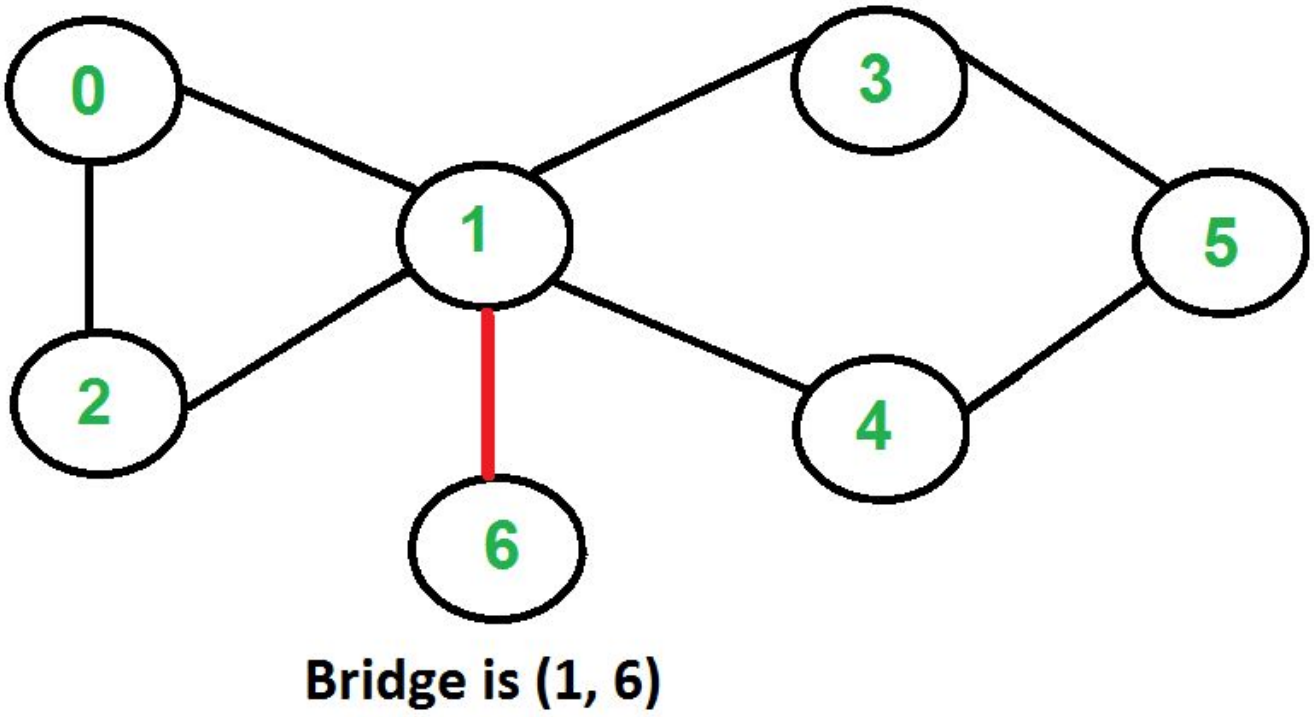
Src: http://d2dskowxfbo68o.cloudfront.net/wp-content/uploads/Bridge1.png

Graph Bridges: Removing (0-3)



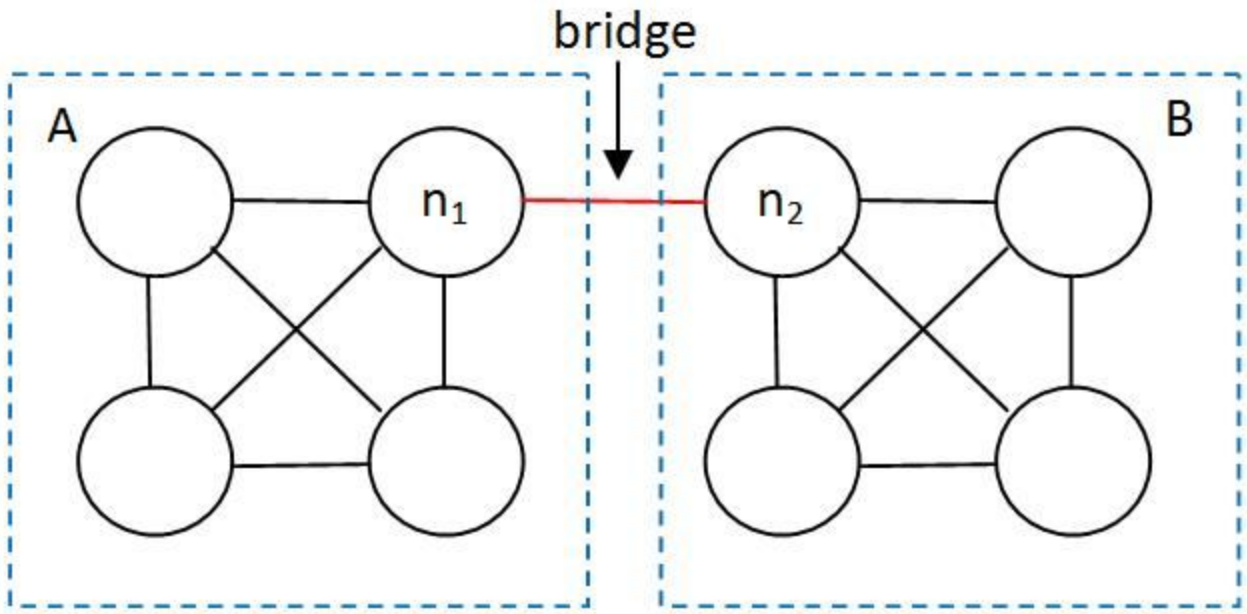
Src: http://d2dskowxfbo68o.cloudfront.net/wp-content/uploads/Bridge1.png

Graph Bridges



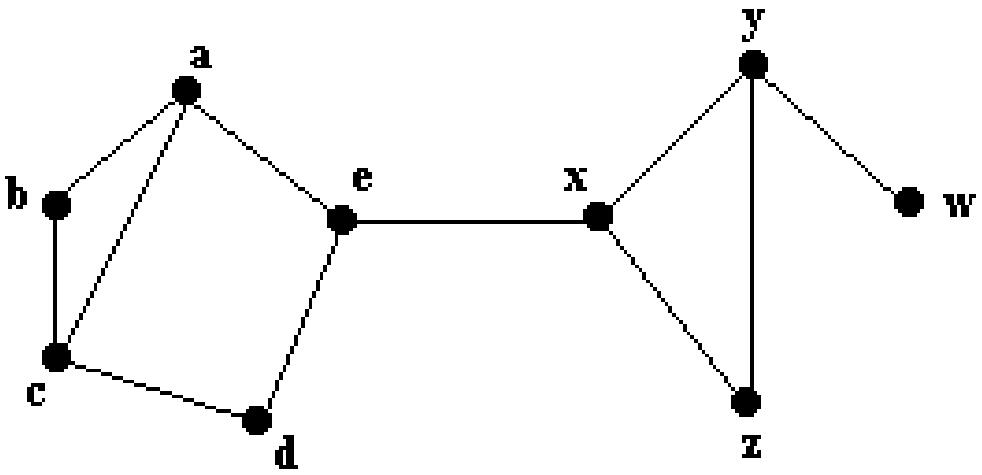
Src: http://d2dskowxfbo68o.cloudfront.net/wp-content/uploads/Bridge2.png

Graph Bridges



Src: https://upload.wikimedia.org/wikipedia/commons/2/21/InterpersonalBridge.jpg

Graph Bridges



Bridges: ex, yw

Every bride is **NOT** part of a **cycle**

Node x and its children have no way to go to the component (a, b, c, d, e) except **through bridge e-x**

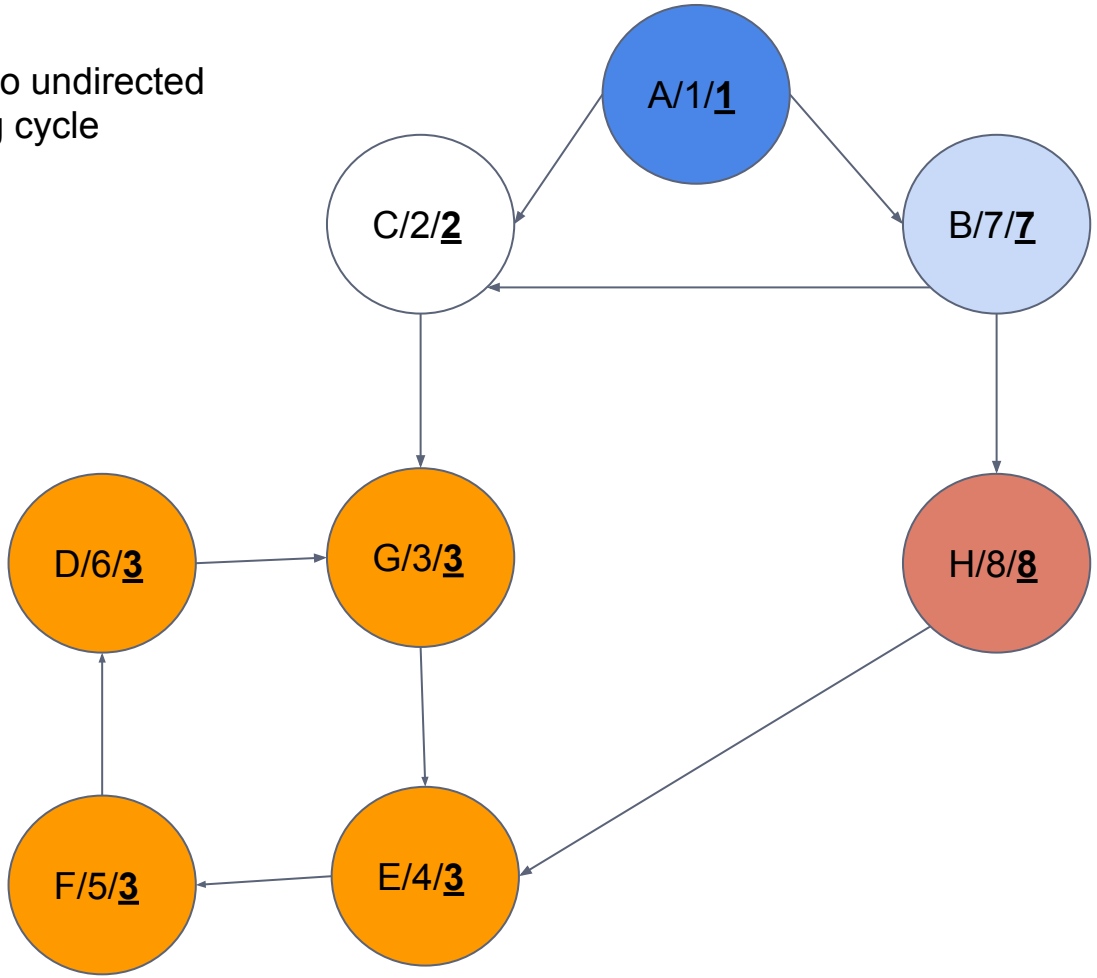
Src: http://www-math.ucdenver.edu/~wcherowi/courses/m4408/ln2ap8.gif

Detecting Graph Bridges

- Recall SCC
  - **Directed** graph
  - Tarjan detects maximal nodes with **edges on cycle** = SCC
  - It detects some edges NOT on a cycle (component graph)
- Simply if edge is going to root of an SCC?
  - then nothing in this SCC can go upper more
  - removing this edge = disconnects this SCC
- But we are undirected graph?!
  - Let's modify tarjan

Recall DFS# and LowestLink

If we converted all edges to undirected all graph becomes one big cycle

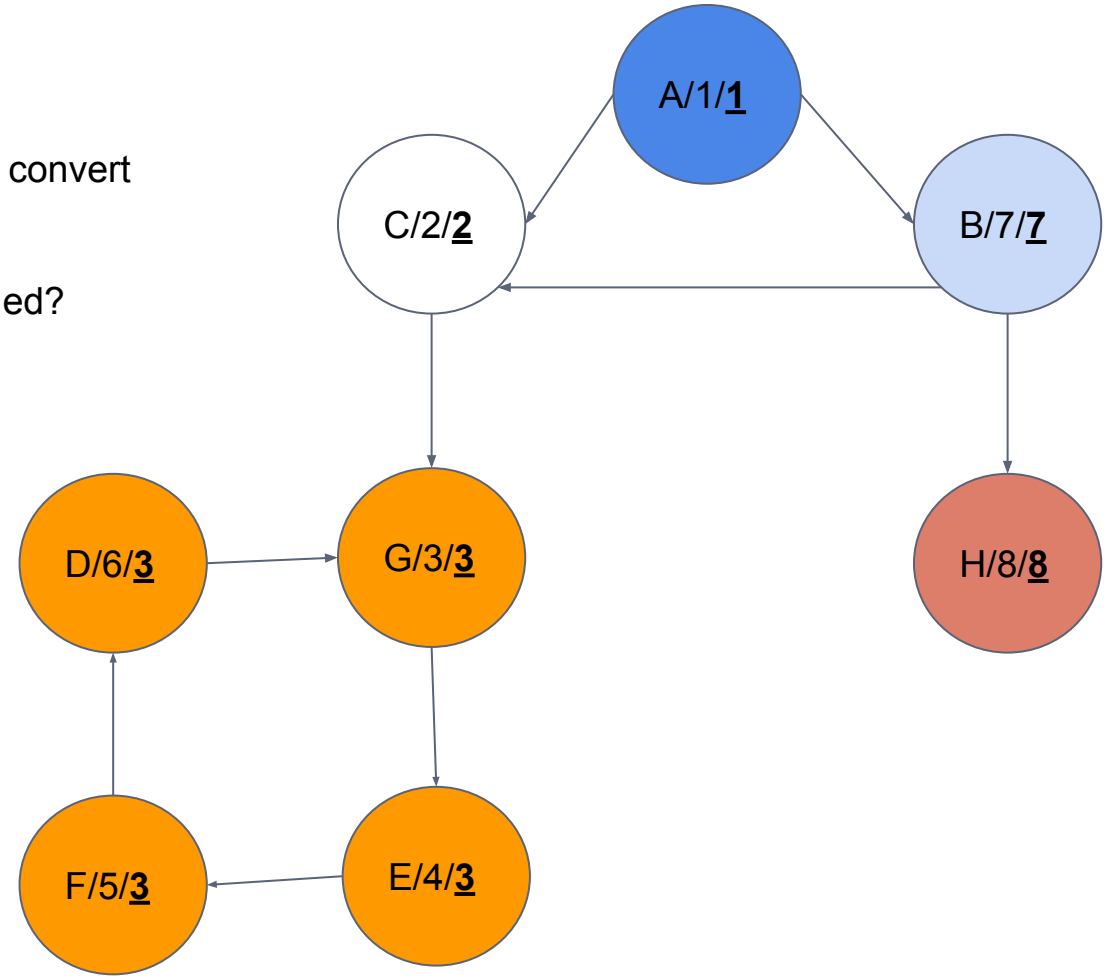


Recall DFS# and LowestLink

Let's modify the graph

This is a bit different..let's convert to undirected

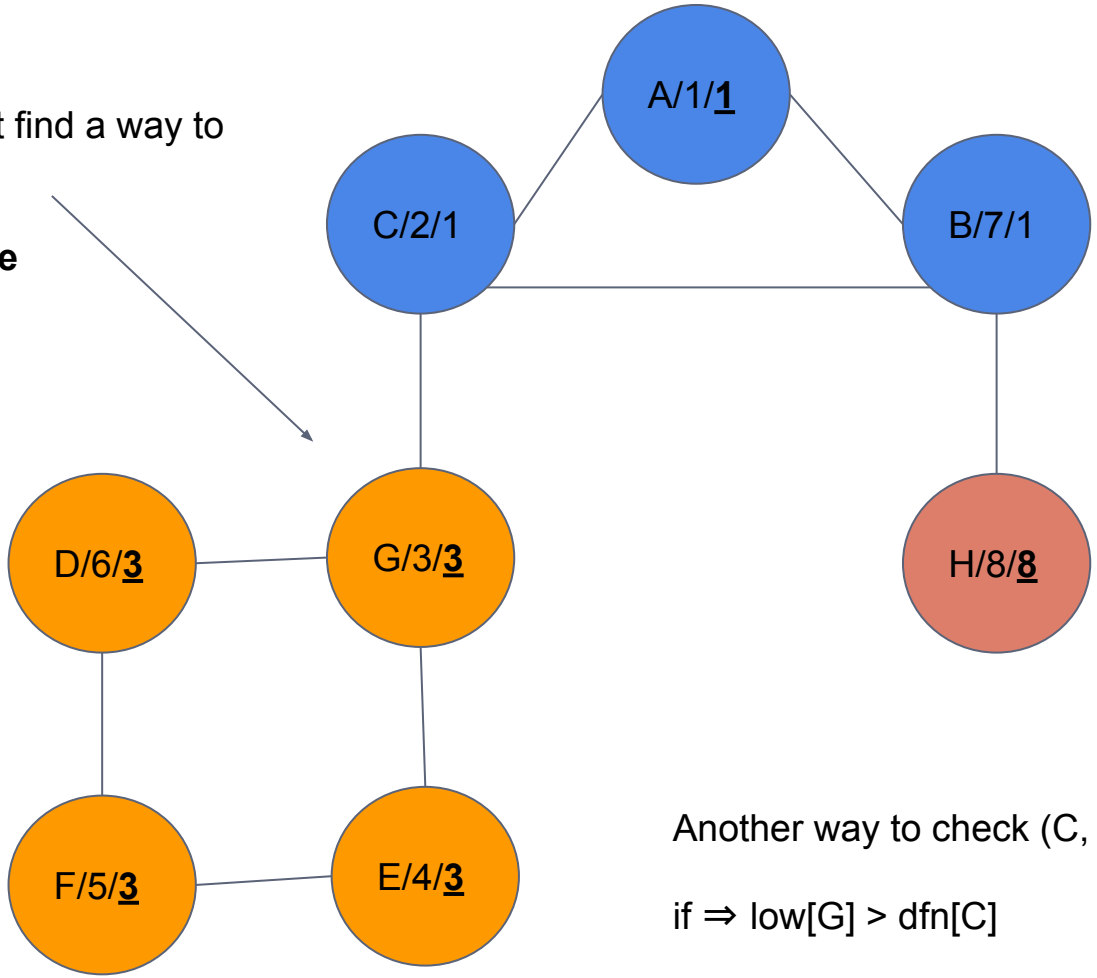
how lowlink can be changed?



Recall DFS# and LowestLink

G dfn # = low link #
⇒ G and its children can't find a way to go back to C?!

Edge from C to G is **bridge**



Another way to check (C, G) is bridge:

if ⇒ low[G] > dfn[C]

that is G lowest ancestor is under C

Tarjan: Directed vs undirected

- (u, v) is same as (v, u)
  - whenever dfs goes from u to v
  - never let it back from v to u
  - classical handling: dfs(u, parent) to know ur last edge
- edge to a visited node?
  - recall: in directed graph it has 2 cases
  - whenever node has a visited child, it must be your ancestor
  - why? because if node v out of dfs stack, then v couldn't reach u...and u can't reach v
  - hence there is 1 case only..and no need to keep who in stack

Tarjan for bridges

```
void tarjan(int node, int par) {
    lowLink[node] = dfn[node] = ndfn++;

    rep(i, adjList[node]) {
        int ch = adjList[node][i];
        if (dfn[ch] == -1) {
            tarjan(ch, node);
            lowLink[node] = min(lowLink[node], lowLink[ch]);
        } else if (ch != par)   // don't go to dad
            lowLink[node] = min(lowLink[node], dfn[ch]);
    }

    // recall: first dfs node has low # = dfs #
    if (lowLink[node] == dfn[node] && par != -1)
        cout<<"bridge "<<par<<" "<<node<<"\n";
}
```

Tarjan for bridges (other way)

```
void tarjan(int node, int par) {
    lowLink[node] = dfn[node] = ndfn++;

    rep(i, adjList[node]) {
        int ch = adjList[node][i];
        if (dfn[ch] == -1) {
            tarjan(ch, node);
            lowLink[node] = min(lowLink[node], lowLink[ch]);

            if (lowLink[ch] == dfn[ch])
            //if (lowLink[ch] > dfn[node])  similar too
                cout<<"bridge "<<node<<" "<<ch<<"\n";
        } else if (ch != par)   // don't go to dad
            lowLink[node] = min(lowLink[node], dfn[ch]);
    }
}
```

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً