Articulation Points and Bridges

- **DFS Tree**
  - If we perform DFS on a graph such that an already visited node is not visited again then the DFS forms a tree structure called a DFS tree.
  - There can be multiple DFS trees.
- **Back Edge**
  - Edge that connects a node x to an already visited node in graph.
  - Back edge leads to cycle in DFS tree
  - *Span Edge*: Edges included in DFS tree
- **Discovery Time Instance**
  - The first moment in time when u visit a node in dfs.
- **Articulation Point**
  - A vertex/node in a graph is called articulation point if by removing the node and its corresponding edges, we split the graphs into more components,i.e., the number of connected components increase.
- **Bridge**
  - An edge is a bridge if by removing it the number of connected components increase in a graph.
- **Lowest Time unit**
  - It is the minimum discovery time of a node that any backedge is pointing in the subtree of current node.

Algorithm and Implementation

- Pick a random vertext and run a DFS from it.
- At any intermediate stage of the DFS: If the current edge (u, v) is such that none of the vertices `v` have a back-edge to any of the ancestors of `u`, the `u` is an articulation point.
- In case of the `root`, if number of child is >= 2, then it is an articulation Point.
- **Implementation Idea:**
  - If a node **x** has a **child** with *lowest time unit* >= **discovery time of x**, then **x** is an *articulation point*.
  - If a node **x** has a **child** with *lowest time unit* > **discovery time of x**, then **x** is a *bridge*.

```
std::vector<std::list<int>> g;

std::vector<int> low((int)(1e5)+2);
```

```cpp
std::vector<int> disctime((int)(1e5)+2, -1);
int dtime = 1;

std::vector<int> art_pt;
std::vector<pii> bridges;

void dfs(int src, int parent = -1){
    low[src] = disctime[src] = dtime;
    dtime++;

    int children = 0;
    for(auto &ne:g[src]){
        if(ne == parent)
            continue;
        if(disctime[ne] == -1){ // not visited
            dfs(ne, src);
            low[src] = std::min(low[src], low[ne]);

            if(parent!=-1 and low[ne]>=disctime[src]){
                art_pt.push_back(src);
            }

            if(low[ne] > disctime[src]){
                bridges.push_back({src, ne});
            }
            ++children;
        } else {
            low[src] = std::min(low[src], disctime[ne]);
        }
    }

    if(parent==-1 and children>=2){
        art_pt.push_back(src);
    }
}
```