Things that we would talk about

- DFS
- Tree
- Connectivity
Useful website

http://codeforces.com/blog.entry/16221
Recommended Practice Sites

- HKOJ
- Codeforces
- Topcoder
- Csacademy
- Atcoder
- USACO
- COCI
Term in Directed Tree

- **Consider node 4**
  - Node 2 is its parent
  - Node 1, 2 is its ancestors
  - Node 5 is its sibling
  - Node 6 is its child
  - Node 6, 7, 8 is its descendants

- **Node 1 is the root**
DFS Forest

- When we do DFS on a graph, we would obtain a DFS forest. Noted that the graph is not necessarily a tree.
- Some of the information we get through the DFS is actually very useful, such as
  - Starting time of a node
  - Finishing time of a node
  - Parent of the node
Some Tricks Using DFS Order

- Suppose vertex v is ancestor (not only parent) of u
  - Starting time of v < starting time of u
  - Finishing time of v > starting time of u
- $st[v] < st[u] \leq ft[u] < ft[v]$
- $O(1)$ to check if ancestor or not
- Flatten the tree to store subtree information (maybe using segment tree or other data structure to maintain)
- Super useful !!!!!!!!!!!
Partial Sum on Tree

- Given queries, each time increase all node from node $v$ to node $u$ by 1
- Assume node $v$ is ancestor of node $u$
- $\text{sum}[u]++$, $\text{sum}[\text{par}[v]]--$
- Run dfs in root
  
  $$\text{dfs}(v)$$
  
  for all child $u$
  
  $$\text{dfs}(u)$$
  
  $$d[v] = d[v] + d[u]$$
Types of Edges

- **Tree edges**
  - Edges that form a tree

- **Forward edges**
  - Edges that go from a node to its descendants but itself is not a tree edge.

- **Back edges**
  - Edges that go from a node to its ancestor

- **Cross edges**
  - Any other edges, connect in the same tree or different tree.
Determination of Edges

- Let’s say now there is an directed edge \((u, v)\).
  - It is tree edge
    - if parent of \(v\) is \(u\)
  - It is forward edge
    - if parent of \(v\) is not \(u\) (not a tree edge) and
    - Starting time of \(u\) < starting time of \(v\) and
    - Finishing time of \(u\) > finishing time of \(u\)
Determination of Edges

- It is back edge
  - If starting time of u > starting time of v and
  - Finishing time of u < finishing time of v
- It is cross edge
  - If starting time of u > starting time of v and
  - Finishing time of u > finishing time of v
Cut Edges (Bridges)

- Definition - a bridge is an edge of a graph whose deletion increases its number of connected component.
- We could easily found all bridges in $O(N + M)$ using DFS.

AMAZING
Finding Cut Edges with DFS

• Let’s maintain an array, named low. \( \text{low}[\text{v}] = \min(\text{start}_\text{time}[\text{w}] \text{ with at least one vertex } \text{u} \text{ in subtree of } \text{v} \text{ such that there is an edge between } \text{w} \text{ and } \text{u}) \).

• Then for every vertex \( \text{v} \), if its child \( \text{u} \), where \( \text{start}_\text{time}[\text{v}] < \text{low}[\text{u}] \). Edge \( (\text{v}, \text{u}) \) is a cut edge.
Cut Vertices (Articulation Point)

- Definition - a articulation point is a vertex of a graph whose deletion increases its number of connected component.
- Similar technique could be used to find cut vertices.
- For every vertex v, if its child u, where start_time[v] <= low[u] AND (v is not the root or number of child of v > 1).
- Consider the case where n = 2 and there is edge between node 1 and 2. Second condition is necessary as removing 1 or 2 would not increase the number of connected component.
Practice Problems

- **Bridges**:  

- **Articulation Points**:  
  - [http://poj.org/problem?id=1523](http://poj.org/problem?id=1523)
Biconnected Component

- Biconnected component is a maximal biconnected subgraph such that there is no articulation point.

- But for various OI problems, we often use the other definition, the bridge-connected component. When we delete all bridges, the vertices that are connected is in the same bridge-connected component.

- Be sure what you are looking for, biconnected component or bridge-connected component, they sound the same but actually have great difference.

- For example: \( n = 5 \).
  - \((1, 2), (2, 3), (3, 1), (2, 4), (4, 5), (5, 2)\) ← these are edges.
Bridge-Connected Component

- How to find BCC?
- SIMPLE!
  - Mark all bridges
  - DFS from 1 to N but do not use bridges to travel
  - Each time produce a BCC
Shrink Trick

- When we shrink all the bridge-connected component in to a node, we would get a tree.
- This help us reduce some hard graph problems into easy tree problems.
- Because in every bridge-connected component, we have got a property. For every vertex v, it could reach vertex u in two ways without duplicate edges

Problems :
- http://www.spoj.com/problems/GRAFFDEF/
Strongly Connected Component

- **CAUTION:** SCC is discussed based on directed graph, bridges, ap and other stuffs are all based on undirected graph.

- A graph is strongly connected if every vertex $v$, could reach to every vertex $u$ (including $v$ itself). And it is maximal (you could not add edges or vertices in this subgraph from the original without breaking this property).
(picture from wikipedia)
Strongly Connected Component

- One way to find scc in linear time is to use Kosaraju's algorithm.

- We first do dfs, for unvisited node from 1 to n
  
  $\text{dfs}(v) :$
  
  - mark $v$ as visited
  - for all unvisited node $u$ connected to $v$, $\text{dfs}(u)$
  - append $v$ in vector $V$

- We then do dfs again(called $\text{rdfs}(v, k)$), from the back of the vector, each time increasing parameter $k$ by 1
  
  $\text{rdfs}(v, k) :$
  
  - mark $v$ as visited, $v$ is belong to scc group $k$
  - for all unvisited node $v$ connected to $u$, $\text{dfs}(u)$

**** CAREFUL
Strongly Connected Component

• Again, you could apply shrink trick to solve problems easily.

• Here are some practice problems for scc:
  - http://www.spoj.com/problems/TFRIENDS/
Weakly Connected Component

• Again we are discussing it based only on directed graphs.

• A graph is weakly connected if every vertex v and u, at least v could reach u or u could reach v. And the graph is maximal.

• How to find ? I’ll leave it as a practice
  
  - https://judge.hkoi.org/task/M1321
Reference

- [http://codeforces.com/blog/entry/16221](http://codeforces.com/blog/entry/16221)
- [https://en.wikipedia.org/wiki/Strongly_connected_component](https://en.wikipedia.org/wiki/Strongly_connected_component)
Q&A

- If you have further question, you are welcomed to facebook me or message me on codeforces (user name: ulna)