Extracting Trees from Graphs

- Graphs are in general unwieldy creatures
- Rooted Trees allow structured top-down approach

Fortunately, one can view a graph as a rooted tree

- with some extra edges added.

To do so, we do Depth-First Search

Depth-First Search

For each node u:

- $d[u]$ is the time it is discovered, and is colored gray
- $f[u]$ is when it is finished, and is colored black

$$
\begin{array}{l}
\text{DFS}(u) \quad // \ u \text{ is white} \\
\quad color[u] \leftarrow \text{gray} \\
\quad d[u] \leftarrow \text{current\_time} \\
\quad \textbf{foreach } w \text{ where } u \rightarrow w \in E \\
\qquad \textbf{if } color[w] = \text{white} \\
\qquad\quad T \leftarrow T \cup \{u \rightarrow w\} \\
\qquad\quad \mathrm{DFS}(w) \\
\quad f[u] \leftarrow \text{current\_time} \\
\quad color[u] \leftarrow \text{black}
\end{array}
$$

Edge Classification

Depth-First Search produces, for each (strongly) connected component, a tree T

- and some extra edges.

For an Undirected graph, each extra edge from u to w is

- a back edge: w is in T an ancestor of u, but not its parent.

For a Directed graph, each extra edge is either

- a back edge: w is in T an ancestor of u, possibly its parent or even u itself; or
- a forward edge: w is in T a descendant of u, but not a child of u; or
- a cross edge: of u and w, neither is an ancestor of the other

Generic Depth First Search

The algorithm DFS can be augmented 5 places:

DFS(u) // u is white
  $color[u] \leftarrow$ gray; $d[u] \leftarrow$ current_time
  PreNode action
  foreach w where $u \rightarrow w \in E$
    if $color[w] =$ white
      PreEdge action
      $T \leftarrow T \cup \{u \rightarrow w\}$
      $\mathrm{DFS}(w)$
      PostEdge action
    else
      OtherEdge action
  $color[u] \leftarrow$ black; $f[u] \leftarrow$ current_time
  PostNode action

Running time: $\Theta(|V| + |E|)$

Topological Sort

For a directed graph, a topological sort is a list of the nodes such that

whenever $u \to w$ is an edge
then u comes before w in the list

- if the graph has a cycle, that task is impossible
- but otherwise it is possible, as we shall see by using Depth-First Search

DFS for Topological Sort

Recall that DFS may produce 4 kinds of edges:

- back edges: then there is a cycle, so task impossible
- forward edge: target finished
  before source discovered target
- cross edge: target finished
  before source was discovered
- tree edge: target will finish
  before source finishes

We conclude: unless task is impossible,

if edge $u \rightarrow w$
then w finishes before u finishes

Thus the finish times yield (reverse) topological sort!

Algorithm for Topological Sort

For the generic DFS, we need

1. to report error when we see a back edge
2. to print a node when it is finished.

We do 2 in PostNode and 1 in OtherEdge:

```
TopologicalSort(u)
    color[u] ← gray
    foreach (u, w) ∈ E
        if color[w] = white
            TopologicalSort(w)
        else
            if color[w] = gray    // back edge
                Error: cyclic graph
    color[u] ← black
    print u
```

Running time: $\Theta(|V| + |E|)$

Articulation Points

For a connected undirected graph (V, E), an articulation point is a node whose removal makes the graph disconnected.

To detect articulation points, we may run DFS:

- the root is an articulation point iff
  it has at least two children

and thus we just need to run DFS $|V|$ times but this takes total time in $\Theta(|V| \cdot |E|)$.

To run DFS only once, we observe:

- a non-root node u is an articulation point iff u has a child that cannot reach above u by a back edge, perhaps preceded by one or more tree edges.

To keep track of that, we let highest$[u]$ be the highest node that u can reach by going down zero or more tree edges, and then going up at most one back edge.

DFS for Articulation Points

highest (abbreviated h) must be

- initialized (PreNode)
- updated after child has been processed (PostEdge)
- updated after seeing back edge (OtherEdge)

We must check if u is articulation point when

- a child of u has been processed (PostEdge)
- u is the root (PostNode)

Algorithm for Articulation Points

ArticulationPoints(u)
- $color[u] \leftarrow$ gray
- $h[u], d[u] \leftarrow$ current_time
- **foreach** $(u, w) \in E$
  - **if** $color[w] =$ white
    - $T \leftarrow T \cup \{(u, w)\}$
    - ArticulationPoints(w)
    - **if** $\mathrm{h}[w] \geq d[u]$ **and** u is not root
      - u is articulation point
    - $\mathrm{h}[u] \leftarrow \min(\mathrm{h}[w], \mathrm{h}[u])$
  - **else if** $d[w] < d[u]$ and $(w, u) \notin T$
    - $\mathrm{h[u]} \leftarrow \min(d[w], \mathrm{h}[u])$
- $color[u] \leftarrow$ black
- **if** u is root and has ≥ 2 children in T
  - u is articulation point

Running time: $\Theta(|V| + |E|) = \Theta(|E|)$.