Searching or Exploring a Graph

There are at least 3 fundamental different methods:

1) DFS (depth-first search)
2) BFS (breadth-first)
3) Random Walk.

Each generates a spanning tree.
Namely 1) First-visit tree.
   i.e. the set of edges used to first visit a vertex.

   These trees are very different.

Next 2 lectures will be DFS.

We start with Basic Depth-first search.

Input: $G = (V, E)$ (Directed graph)
      $S \in V$ (Start vertex)
Basic Depth First Search

**Alg**: DFS(G)
1) \( \forall u \in V \) \( \text{color}(u) \leftarrow \text{white} \); \( \text{time} \leftarrow 0 \)
2) \( \forall u \in V \) if \( \text{color}(u) = \text{white} \) then DFS-Visit(\( u \))
   (what order?)

**Alg**: DFS-Visit(\( u \))
1) \( \text{color}(u) \leftarrow \text{gray} \); \( \text{push}\text{-time}(u) \leftarrow \text{time}++ \)
2) \( \forall v \in \text{Adj}(u) \)
   if \( \text{color}(v) = \text{white} \) then DFS-Visit(\( v \))
3) \( \text{color}(u) \leftarrow \text{black} \); \( \text{pop}\text{-time}(u) \leftarrow \text{time}++ \)

**Note**: \( \text{dfs}(u) = \text{push}\text{-time}(u) \)
An Example

4-types
- Tree
- Back-edge
- Cross
- Forward
Testing Edge Types

Consider time that edge \( e \) is first used.

Tree \( (e) \) iff \( \text{color}(v) = \text{white} \)

Back Edge \( (e) \) iff \( \text{color}(v) = \text{gray} \)

\( \text{color}(v) = \text{black} \) iff \( \text{Cross}(e) \) or \( \text{Forward}(e) \)

\( \text{color}(v) = \text{black} \) & \( \text{dfs}(u) < \text{dfs}(v) \) \( \Rightarrow \) forward edge

\( \text{Cross} \) edge
Thm. The intervals \([\text{push}(u), \text{pop}(u)]\) are well nested in

\[\text{push}(u) < \text{push}(v) < \text{pop}(v) < \text{pop}(u)\]

or

\[\text{push}(u) < \text{pop}(u) < \text{push}(v) < \text{pop}(v)\]

\begin{tabular}{c|c}
(type, edge) & pop \\
\hline
Tree & \text{pop}(v) < \text{pop}(u) \\
back & \text{pop}(v) > \text{pop}(u) \\
cross & \text{pop}(v) < \text{pop}(u) \\
forward & \\
\end{tabular}

Thm. If \(G\) is a DAG & \((u, v) \in E\) then

\[\text{pop}(v) < \text{pop}(u)\]

DAG = Directed Acyclic Graph
**Topological Sort**

**Def:** If \( G = (V, E) \) is a DAG then an ordering \( x_1, \ldots, x_n \) of \( V \) is a topological sort if

\[
(x_i, x_j) \in E \implies i < j
\]

**Thm:** For a DAG reverse pop times is a topological sort.

ie \( a \rightarrow b \implies \text{pop}(a) > \text{pop}(b) \)

**Thm:** Topological Sort is \( O(n+m) \) time

Assume \( G \) is a DAG.
Thm. The following are equivalent:

a) $G$ has a cycle
b) Every DFS generates a back edge.
c) Some DFS generates a back edge

Proof: b) $\Rightarrow$ c) $\Rightarrow$ a) Easy.

a) $\Rightarrow$ b)

Suppose $C$ is a cycle in $G$, DFS.

Assume that $x_i$ is first vertex searched

$$C = x_i \rightarrow x_2 \rightarrow \ldots \rightarrow x_k$$

Claim: $(x_k, x_i)$ in a back edge

$$\text{push}(x_i) < \text{push}(x_k) < \text{pop}(x_k) \leq \text{pop}(x_i)$$
Biconnected Components

\( G \) is undirected

\( G \) is connected if \( \forall v, w \in V \exists \text{ path from } v \text{ to } w. \)

\( \forall v \in V \) is an articulation point if \( \exists \) distinct \( x, y \) s.t.

All paths from \( x \) to \( y \) visit \( v \).

Def \( G \) is biconnected if \( \nexists \) an articulation point

a graph consisting of a single edge is called a trivial biconnected graph.

Def A biconnected component is a maximal subgraph which is biconnected
Using DFS for Biconnectivity

**Theorem:** In undirected case all edges are tree or back edges.

**Definition:**
\[
\text{low}(v) = \min \{ \text{dfs}(w) \mid \exists u, u \text{ descendent of } v \land u \rightarrow w \text{ back edge }, \exists U \{ \text{dfs}(v) \} \}
\]
The Articulation Points after DFS

Thm Suppose G is connected & we have run DFS.
1) Leaves are not Arts.
2) The root is an Art iff #children \geq 2
3) If u is not leaf and not the root then
   \[ u \text{ is an Art iff } \exists \text{ child } v \text{ of } u \text{ st } \text{low}(v) \geq \text{dfs}(u). \]

Pr 1) If v is a leaf then T\{-v\} is connected.
2) If r is root with 1 child then T\{-r\} is connected.

Since any path from one child to the other uses root.
**Proof of case 3.**

\( \Rightarrow \) Suppose \( u \) is not leaf or root and \( \exists x, y \notin \text{root} \) all paths from \( x \) to \( y \) use \( u \).

3 subcases:

a) \( x, y \in \text{subtree}(u) \) (false, empty)

b) \( x, y \in \text{subtree}(u) \)

c) \( x \in \text{subtree} \& y \in \text{subtree} \)

3b) Suppose \( \text{low}(x) \& \text{low}(y) < \text{dfs}(u) \)

\( \text{(contra!)} \)

\( \text{WLOG } \text{low}(x) \geq \text{dfs}(u) \)

3c) \( \text{low}(x) < \text{dfs}(u) \) \( \text{contra!} \)

\( \Leftarrow \) \( v = \text{child}(u) \& \text{low}(v) \geq \text{dfs}(u) \)

then \( u \) separates \( v \) from root

\[ \text{Diagram of tree with root U, child V, and leaf Y} \]
Example

Arts 2, 4
**Depth First Search to compute low(v)**

\[ \text{Alg: } \text{DFS}(G) \]

1) \( \forall u \in V \) \( \text{color}(u) \leftarrow \text{white} \); \( \text{time} \leftarrow 0 \)
2) \( \forall u \in V \) if \( \text{color}(u) = \text{white} \) then \( \text{DFS-Visit}(u) \)
   (what order?)

\[ \text{Alg: } \text{DFS-Visit}(u) \]

1) \( \text{color}(u) \leftarrow \text{gray} \); \( \text{push-time}(u) \leftarrow \text{time} + 1 \)
   \( \text{low}(u) \leftarrow \text{DFS}(u) \)
2) \( \forall v \in \text{Adj}(u) \)
   if \( \text{color}(v) = \text{white} \) then \( \text{DFS-Visit}(v) \)
3) \( \text{color}(u) \leftarrow \text{black} \); \( \text{pop-time}(u) \leftarrow \text{time} + 1 \)
   3a) if \( (u,v) \) is back edge then
       \( \text{low}(u) \leftarrow \min\{\text{low}(u), \text{dfs}(v)\} \)
   3b) if \( (u,v) \) is tree edge then
       \( \text{low}(u) \leftarrow \min\{\text{low}(u), \text{low}(v)\} \)