Taking Stock

Last Time
- Depth-First Search

This Time: Uses of DFS
- Topological Sort
- Strongly Connected Components

Depth-First Search

DFS (Initialize and Go)

```
DFS(V, E)
1 for each u in V
2 do color(u) ← GREEN
3 π(u) ← NIL
4 time ← 0
5 for each u in V
6 do if color[u] = GREEN
7 then DFS-visit(u)
```
DFS (Visit Node—Recursive)

DFS-VISIT(\( u \))
1 \( \text{color}(u) \leftarrow \text{YELLOW} \)
2 \( d[u] \leftarrow \text{time}++ \)
3 \text{for each } v \text{ in } \text{Adj}[u] \text{ do if } \text{color}[v] = \text{GREEN} \text{ then } \pi[v] \leftarrow u \\
4 \text{dfs-visit}(v) \)
5 \( \text{color}(u) \leftarrow \text{RED} \)
6 \( f[u] = \text{time}++ \)

Parental Theorem

- Let’s look at the intervals: \([d[v], f[v]]\) for each vertex \( v \in V \).
  (Surely, \( d[v] < f[v] \))
- These tell us about the predecessor relationship in \( G_{\pi} \)

- If I finish exploring \( u \) before first exploring \( v \), \((d[u] < f[v])\) then \( v \) is not a descendant of \( u \). (Or versa vice)
- If \([d[u], f[u]] \subset [d[v], f[v]]\) then \( u \) is a descendent of \( v \) in the DFS tree
- If \([d[v], f[v]] \subset [d[u], f[u]]\) then \( v \) is a descendent of \( u \) in the DFS tree

Analysis of DFS

- Loop on lines 1-3 \( O(|V|) \)
- DFS-VISIT is called \textbf{exactly} once for each vertex \( v \) (Why?)
  Because the first thing you do is paint the node \textbf{YELLOW}
- The Loop on lines 3-6 in calls DFS-VISIT \(|\text{Adj}[v]|\) times for vertex \( v \).
- Since DFS visit is called exactly once per vertex, the total running time to do loop on lines 3-6 is

\[
\sum_{v \in V} |\text{Adj}[v]| = \Theta(|E|).
\]
- Therefore: running time of DFS on \( G = (V, E) \) is \( \Theta(|V| + |E|) \): \textbf{Linear} in the (adjacency list) size of the graph

Graph Review...

Think back to your thorough reading of Appendix B.4 and B.5...

- A \textbf{path} in \( G \) is a sequence of vertices such that each vertex is adjacent to the vertex preceding it in the sequence. Simple paths \textbf{do not} repeat nodes.
- A (simple) \textbf{cycle} is a (simple) path except that the first and last vertices are the same.
- Paths and cycles can either be \textbf{directed} or \textbf{undirected}
- If I say “cycle” or “path,” I will often mean simple, \textbf{undirected} cycle or path
The DFS graph: $G_\pi = (V, E_\pi)$ forms a forest of subtrees

**New Definitions: Tree**

- A tree $T = (V, E)$ is a connected graph that does not contain a cycle
- All pairs of vertices in $V$ are connected by a simple (undirected) path
- $|E| = |V| - 1$
- Adding any edge to $E$ forms a cycle in $T$

A (Undirected) acyclic graph is usually called a forest
- A DAG is a Directed, Acyclic Graph
  - A directed forest
  - A subtree is simply a subgraph that is a tree

Classifying Edges in the DFS Tree

Given a DFS Tree $G_\pi$, there are four type of edges $(u, v)$

- Tree Edges: Edges in $E_\pi$. These are found by exploring $(u, v)$ in the DFS procedure
- Back Edges: Connect $u$ to an ancestor $v$ in a DFS tree
- Forward Edges: Connect $u$ to a descendent $v$ in a DFS tree
- Cross Edges: All other edges. They can be edges in the same DFS tree, or can cross trees in the DFS forest $G_\pi$

Modifying DFS to Classify Edges

DFS can be modified to classify edges as it encounters them...
- Classify $e = (u, v)$ based on the color of $v$ when $e$ is first explored...

- **GREEN**: Indicates Tree Edge
- **YELLOW**: Indicates Back Edge
- **RED**: Indicates Forward or Cross Edge
DFS Undirected Graphs

- In an undirected graph, there may be some ambiguity, as \((u, v)\) and \((v, u)\) are the same edge. The following theorem will help clear things up.

**Thm**

In a DFS of an undirected graph \(G = (V, E)\), every edge is a tree edge or a back edge.

---

DAG Gum it!

- DAGs are good at modeling processes and structures that have a partial order \((\prec)\)
  - \(A \prec B\) and \(B \prec C\) \(\Rightarrow\) \(A \prec C\)
  - May have neither \(A \prec B\) nor \(B \prec C\)
- Think of a partial order as “the way in which you must do tasks” to ensure successful completion
- Sometimes it doesn’t matter if you do \(A\) first or \(B\) first...

---

Spring Break on My Mind!

- Put ice in shaker (A)
- Pour gin\(^a\) in shaker (B)
- Pour vermouth\(^b\) in shaker (C)
- Stir\(^c\) (D)
- Strain (E)
- Put ice in glass (F)
- Remove ice from glass (G)
- Pour in glass (H)
- Add olive to glass (I)
- Enjoy! (J)

\(^a\)Preferably Boodles  
\(^b\)Very Little  
\(^c\)Never shake

---

Topological Sort

- We would like to produce a valid order for making a martini
- A **topological sort** of a directed acyclic graph (DAG) is a linear ordering of its nodes which is compatible with the partial order \(\prec\) induced on the nodes.
- \(u \prec v\) if there’s a directed path from \(u\) to \(v\) in the DAG.
- An equivalent definition is that each node comes before all nodes to which it has edges.
- i.e. \(u\) must be done before \(v\)
- Every DAG has at least one topological sort, and may have many.
Topological Sort

**Topological Sort: The Whole Algorithm**
- DFS search the graph
- List vertices in order of decreasing finishing time

**Why Does This Work?**
- Show that \((u, v) \in E \Rightarrow f[v] < f[u]\)
- When we explore \((u, v)\), \(u\) is **YELLOW**

Why Does This Work? (cont.)

What color is \(v\)?
- Is \(v\) **YELLOW**?
  - **No**, since then DAG would have a cycle
- Is \(v\) **GREEN**?
  - Then it becomes descendant of \(u\) and (by theorem), \(d[u] < d[v] < v[f] < f[u]\)
- Is \(v\) **RED**?
  - If so, then we’re finished and \(f[v] < f[u]\) since we’re still exploring \(u\)

Therefore if \((u, v) \in E, f[v] < f[u]\)

Quite Enough Done

Strongly Connected Components

- Given a directed graph \(G = (V, E)\), a strongly connected component of \(G\) is a maximal set of vertices \(C \subseteq V\) such that \(\forall u, v, \in C\) there exists a directed path both from \(u\) to \(b\) and from \(v\) to \(u\)
- The algorithm uses the transpose of a directed graph \(G = (V, E)\), where the orientations are flipped:
  \[
  G^T = (V, E^T), \text{ where } E^T = \{(v, u) \mid (u, v) \in E\}
  \]
- What is running time to create \(G^T\)?
- Note: \(G\) and \(G^T\) have the same Strongly Connected Components
  - \(u\) and \(v\) are both reachable from each other in \(G\) if and only if they are both reachable with the orientations flipped \((G^T)\).

Finding Strongly Connected Components

- Call DFS\((G)\) to topologically sort \(G\)
- Compute \(G^T\)
- Call DFS\((G^T)\) but consider vertices in topologically sorted order (from \(G\))
- Vertices in each tree of depth-first forest for SCC
Component Graph

- $G^{SCC}(G) = (V^{SCC}, E^{SCC})$
- $V^{SCC}$ has one vertex for each strongly connected component of $G$
- $e \in E^{SCC}$ is there is an edge between corresponding SCC’s in $G$

Lemma

$G^{SCC}$ is a DAG

- For $C \subseteq V$, $f(C) \overset{\text{def}}{=} \max_{v \in C}\{f[v]\}$

More Lemma

Lemma

Let $C$ and $C'$ be distinct SCC in $G$,
- if $(u, v) \in E$ and $u \in C, v \in C'$, then $f(C) > f(C')$
- if $(u, v) \in E^T$ and $u \in C, v \in C'$, then $f(C) < f(C')$
- If $f(C) > f(C')$, there is no edge from $C$ to $C'$ in $G^T$

Why SCC Works. (Intuition)

- DFS on $G^T$ starts with SCC $C$ such that $f(C)$ is maximum. Since $f(C) > f(C')$, there are no edges from $C$ to $C'$ in $G^T$
- This means that the DFS will visit only vertices in $C$
- The next root has the largest $f(C')$ for all $C' \neq C$. DFS visits all vertices in $C'$, and any other edges must go to $C$, which we have already visited..

Next Time

- Spanning Trees