IE170: Algorithms in Systems Engineering: Lecture 17

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

March 2, 2007

Taking Stock

Last Time

- Depth-First Search

This Time: Uses of DFS

- Topological Sort
- Strongly Connected Components

Depth-First Search

DFS

- Input: Graph $G = (V, E)$
  - No source vertex here. Works for undirected and directed graphs.
  - We focus on directed graphs today...
- Output: Two timestamps for each node $d(v)$, $f(v)$,
- Output: $\pi(v)$, predecessor of v
  - not on shortest path necessarily

DFS (Initialize and Go)

$\text{DFS}(V, E)$

for each u **in** V
do $\mathit{color}(u) \leftarrow \text{GREEN}$
$\quad \pi(u) \leftarrow \text{NIL}$
$\mathit{time} \leftarrow 0$
for each u **in** V
do if $\mathit{color}[u] = \text{GREEN}$
$\qquad$ **then** $\text{DFS-VISIT}(u)$

DFS (Visit Node—Recursive)

DFS-VISIT(u)

1 $\textit{color}(u) \leftarrow$ YELLOW
2 $d[u] \leftarrow \textit{time}{++}$
3 **for each** v **in** $\textit{Adj}[u]$
4 **do if** $\textit{color}[v] =$ GREEN
5   **then** $\pi[v] \leftarrow u$
6   DFS-VISIT(v)
7
8 $\textit{color}(u) \leftarrow$ RED
9 $f[u] = \textit{time}{++}$

Parenthesis 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_π

---

1. If I finish exploring u before first exploring v, $(d[u] < f[v])$ then v is not a descendant of u. (Or versa vice)
2. If $[d[u], f[u]] \subset [d[v], f[v]]$ then u is a descendent of v in the DFS tree
3. 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 exactly once for each vertex v (Why?)
  - Because the first thing you do is paint the node YELLOW
- The Loop on lines 3-6 in calls DFS-VISIT $|\textit{Adj}[v]|$ 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

$$\sum_{v \in V} |\textit{Adj}[v]| = \Theta(|E|).$$

- Therefore: running time of DFS on $G = (V, E)$ is $\Theta(|V| + |E|)$: 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 path in G is a sequence of vertices such that each vertex is adjacent to the vertex preceding it in the sequence. Simple paths do not repeat nodes.
- A (simple) cycle is a (simple) path except that the first and last vertices are the same.
- Paths and cycles can either be directed or undirected
- If I say "cycle" or "path," I will often mean simple, undirected cycle or path

I Can't See the Forest Through the...

- 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

More Definitions

- 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_π, there are four type of edges (u, v)

1. **Tree Edges**: Edges in E_π. These are found by exploring (u, v) in the DFS procedure
2. **Back Edges**: Connect u to an ancestor v in a DFS tree
3. **Forward Edges**: Connect u to a descendent v in a DFS tree
4. **Cross Edges**: All other edges. They *can* be edges in the same DFS tree, or can cross trees in teh DFS forest G_π

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 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 $B \prec C \Rightarrow A < C$
  - 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

Task Relations

- $A \prec B$
- $A \prec C$
- $B \prec D$
- $C \prec D$
- $D \prec E$
- $F \prec G$
- $G \prec H$
- $G \prec I$
- $E \prec H$
- $H \prec J$
- $I \prec J$

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

1. DFS search the graph
2. 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), $d[u] < d[v] < f[v] < f[u]$
- 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

1. Call $\mathrm{DFS}(G)$ to topologically sort G
2. Compute G^T
3. Call $\mathrm{DFS}(G^T)$ but consider vertices in topologically sorteded order (from G)
4. Vertices in each tree of depth-first forest for SCC

Component Graph

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

Lemma

G^{SCC} is a DAG

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

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