IE170: Algorithms in Systems Engineering: Lecture 16

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

February 28, 2007

Taking Stock

Last Time

- The Wonderful World of Graph Theory
- Breadth First Search

This Time

- Finish Breadth-First Search
- Depth-First Search

Graph Search Algorithms

- There are two "workhorse" algorithms for searching graphs that form the basis for many more complicated algorithms.
  - Breadth-First Search (BFS): Search "broadly"
  - Depth-First Search (DFS): Search "deeply"
- BFS: Last Time
- DFS: Today

Recall—BFS

BFS

- Input: Graph $G = (V, E)$, source node $s \in V$
- Output: $d(v)$, distance (smallest # of edges) from s to v $\forall v \in V$
- Output: $\pi(v)$, predecessor of v on the shortest path from s to v

Oh no! DP again

- $\delta(s, v)$: shortest path from s to v
- Lemma: If $(u, v) \in E$, then $\delta(s, v) \leq \delta(s, u) + 1$

BFS

BFS(V, E, s)

```
for each u in V \ {s}
do d(u) ← ∞
    π(u) ← NIL
d[s] ← 0
Q ← ∅
ADD(Q, s)
while Q ≠ ∅
do u ← POLL(Q)
    for each v in Adj[u]
    do if d[v] = ∞
          then d[v] ← d[u] + 1
               π[v] = u
               ADD(Q, v)
```


Analysis

- How many times is each vertex added?
  - Answer: Once. So $|V|$ for add operation
- How many times is adjaceny list of vertex v scanned?
  - Answer: Once. Since $\sum_{v \in V} \text{size}(Adj[v]) = 2|E|$ (for undirected), we have $|E|$ here.
- Running time: $O(|V| + |E|)$: Linear in the input size of the graph (for adjacency list implementation)

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

Compare and Contrast

- BFS: Discovers all nodes at distance k from starting node s before discovering any node at distance $k+1$
- DFS: As soon as we discover a vertex, we explore from it.
- Here we are after creating a different predecessor subgraph

$$G_\pi = (V, E_\pi) \text{ with } E_\pi = \{(\pi[v], v) \mid v \in V, \pi[v] \leq \text{NIL}\}$$

- Not shortest edge-path lengths

DFS Colors

In this implementation, we will use colors:

- GREEN: vertex is undiscovered
- YELLOW: vertex is discovered, but not finished
- RED: vertex is finished.
  - (i.e., we have completely explored everything from this node)

Discovery and Finish Times

- Unique integers from 1 to $2|V|$ denoting when you first discover a vertex and when you are done with it
- $d[v] < f[v] \; \forall v \in V$

DFS (Initialize and Go)

DFS(V, E)

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

DFS (Visit Node—Recursive)

DFS-VISIT(u)

$color(u) \leftarrow$ YELLOW
$d[u] \leftarrow time{++}$
for **each** v **in** $Adj[u]$
do if $color[v] =$ GREEN
    then $\pi[v] \leftarrow u$
        DFS-VISIT(v)

$color(u) \leftarrow$ RED
$f[u] = time{++}$

Example

- Here I will show Java code and an example

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 $|Adj[v]|$ 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

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

- 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

- 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

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

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.

Next Time

- Graphs, Graphs, and more Graphs.
- Additional Hmwk: Problems: 22.2-5, 22.2-6, 22.3-8, 22.4-3