Lecture 14: Graphs II: Depth-First Search

Lecture Overview

- Depth-First Search
- Edge Classification
- Cycle Testing
- Topological Sort

Recall:

- **graph search**: explore a graph
e.g., find a path from start vertex \( s \) to a desired vertex

- **adjacency lists**: array \( \text{Adj} \) of \(|V|\) linked lists
  - for each vertex \( u \in V \), \( \text{Adj}[u] \) stores \( u \)'s neighbors, i.e., \( \{ v \in V \mid (u, v) \in E \} \)
  (just outgoing edges if directed)

For example:

![Adjacency Lists](image)

Figure 1: Adjacency Lists

**Breadth-first Search (BFS):**

Explore level-by-level from \( s \) — find shortest paths
Depth-First Search (DFS)

This is like exploring a maze.

![Depth-First Search Frontier](image)

Figure 2: Depth-First Search Frontier

**Depth First Search Algorithm**

- follow path until you get stuck
- backtrack along breadcrumbs until reach unexplored neighbor
- recursively explore
- careful not to repeat a vertex

**Example**

See Figure 4.  

**Edge Classification**

- to compute this classification (back or not), mark nodes for duration they are “on the stack”
- *Question to ponder:* Which of these edges can occur in an undirected graph?
parent = {s: None}

DFS-visit (V, Adj, s):

\[\text{start} \rightarrow \text{for } v \text{ in } \text{Adj}[s]: \]
\[\quad \text{if } v \text{ not in parent:} \]
\[\quad \quad \text{parent}[v] = s \]
\[\quad \quad \text{DFS-visit } (V, \text{Adj}, v) \]

\[\text{finish} \rightarrow \]

DFS (V, Adj)

\[\text{parent} = {} \]

\[\text{for } s \text{ in } V: \]
\[\quad \text{if } s \text{ not in parent:} \]
\[\quad \quad \text{parent}[s] = \text{None} \]
\[\quad \quad \text{DFS-visit } (V, \text{Adj}, s) \]

Figure 3: Depth-First Search Algorithm

![Diagram of depth-first search algorithm with vertices a, b, c, d, e, f and edges labeled with numbers 1 through 8, showing forward, back, and cross edges.]

**Answer:** Only tree and back edges.

**Analysis**

- DFS-visit gets called with a vertex \( s \) only once (because then \( \text{parent}[s] \) set)
  \[\Rightarrow \text{time in DFS-visit} = \sum_{s \in V} |\text{Adj}[s]| = O(E)\]

- DFS outer loop adds just \( O(V) \)
  \[\Rightarrow O(V + E) \text{ time (linear time)}\]
Cycle Detection

Graph G has a cycle $\iff$ DFS has a back edge

Proof

For $\Leftarrow$, assume that $v_0, \ldots, v_k$ is a cycle in the graph and $v_0$ is the first node in the cycle visited by the DFS traversal.

We claim that $(v_k, v_0)$ is a back edge.
• before visit to \(v_i\) finishes, 
  will visit \(v_{i+1}\) (& finish):
  will consider edge \((v_i, v_{i+1})\)
  \(\implies\) visit \(v_{i+1}\) now or already did

• \(\implies\) before visit to \(v_0\) finishes, 
  will visit \(v_k\) (& didn’t before)

• \(\implies\) before visit to \(v_k\) (or \(v_0\)) finishes, 
  will see \((v_k, v_0)\) as back edge

### Job scheduling

Given Directed Acyclic Graph (DAG), where vertices represent tasks & edges represent dependencies, order tasks without violating dependencies

![Dependence Graph: DFS Finishing Times](image)

**Figure 6:** Dependence Graph: DFS Finishing Times

### Source:

Source = vertex with no incoming edges

= schedulable at beginning \((A,G,I)\)

### Attempt:

BFS from each source:
• from A finds A, BH, C, F

• from D finds D, BE, CF ← slow . . . and wrong!

• from G finds G, H

• from I finds I

**Topological Sort**

Reverse of DFS finishing times (time at which DFS-Visit\(v\) finishes)

\[
\begin{align*}
&\text{DFS-Visit}(v) \\
&\quad \ldots \\
&\quad \text{order.append}(v) \\
&\text{order.reverse()}
\end{align*}
\]

**Correctness**

For any edge \((u, v)\) — \(u\) ordered before \(v\), i.e., \(v\) finished before \(u\)

- if \(u\) visited before \(v\):
  - before visit to \(u\) finishes, will visit \(v\) (via \((u, v)\) or otherwise)
  - \(\implies v\) finishes before \(u\)

- if \(v\) visited before \(u\):
  - graph is acyclic
  - \(\implies u\) cannot be reached from \(v\)
  - \(\implies\) visit to \(v\) finishes before visiting \(u\)