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 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:

```
  a
  |   |
  v   v
  |   |
  b -- c
```

```
  a
  |   |
  v   v
  |   |
  b  c
```

```
  a
  |   |
  v   v
  |   |
  b  c
```

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.

![DFS exploration diagram](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

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

![DFS algorithm flowchart](image)

**Figure 3: Depth-First Search Algorithm**

search from start vertex s (only see stuff reachable from s)

explore entire graph (could do same to extend BFS)
Example

Figure 4: Depth-First Traversal

Edge Classification

Figure 5: Edge Classification

• to compute this classification (back or not), mark nodes for duration they are “on the stack”

• only tree and back edges in undirected graph

Analysis

• DFS-visit gets called with a vertex \( s \) only once (because then 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

(<=) tree edges

$\Rightarrow$ is a cycle

back edge: to tree ancestor

(=>) consider first visit to cycle:

$\Rightarrow$ before visit to $v_i$ finishes,
will visit $v_{i+1}$ (& finish):
will consider edge $(v_i, v_{i+1})$
$\Rightarrow$ visit $v_{i+1}$ now or already did

$\Rightarrow$ before visit to $v_0$ finishes,
will visit $v_k$ (& didn’t before)

$\Rightarrow$ 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
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)

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

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

\[ \text{U} \rightarrow \text{V} \]

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