CSE 373: Data Structures and Algorithms

Lecture 21: Graphs III
Depth-first search

- **depth-first search (DFS)**: finds a path between two vertices by exploring each possible path as many steps as possible before backtracking
  - often implemented recursively
DFS example

• All DFS paths from A to others (assumes ABC edge order)
  – A
  – A -> B
  – A -> B -> D
  – A -> B -> F
  – A -> B -> F -> E
  – A -> C
  – A -> C -> G

• What are the paths that DFS did not find?
DFS pseudocode

• Pseudo-code for depth-first search:
  
  \[
  \text{dfs}(v_1, v_2):
  \]
  \[
  \text{dfs}(v_1, v_2, \emptyset)
  \]
  \[
  \text{dfs}(v_1, v_2, \text{path}):
  \]
  \[
  \text{path} \, += \, v_1.
  \]
  \[
  \text{mark} \, v_1 \, \text{as visited}.
  \]
  \[
  \text{if} \, v_1 \, \text{is} \, v_2:
  \]
  \[
  \text{path} \, \text{is found}.
  \]
  \[
  \text{for each unvisited neighbor} \, v_i \, \text{of} \, v_1
  \]
  \[
  \text{where there is an edge from} \, v_1 \, \text{to} \, v_i:
  \]
  \[
  \text{if} \, \text{dfs}(v_i, v_2, \text{path}) \, \text{finds a path,} \, \text{path} \, \text{is found}.
  \]
  \[
  \text{path} \, -= \, v_1. \, \text{path is not found}.
  \]
DFS observations

• guaranteed to find a path if one exists
• easy to retrieve exactly what the path is (to remember the sequence of edges taken) if we find it
• optimality: not optimal. DFS is guaranteed to find a path, not necessarily the best/shortest path
  – Example: DFS(A, E) may return A -> B -> F -> E
Another DFS example

• Using DFS, find a path from BOS to LAX.
Breadth-first search

- **breadth-first search (BFS)**: finds a path between two nodes by taking one step down all paths and then immediately backtracking
  - often implemented by maintaining a list or queue of vertices to visit
  - BFS always returns the path with the fewest edges between the start and the goal vertices
BFS example

• All BFS paths from A to others (assumes ABC edge order)
  – A
  – A -> B
  – A -> C
  – A -> E
  – A -> B -> D
  – A -> B -> F
  – A -> C -> G

• What are the paths that BFS did not find?
BFS pseudocode

• Pseudo-code for breadth-first search:
  \[
  \text{bfs}(v_1, v_2):
  \]
  \[
  \text{List} := \{v_1\}.
  \]
  \[
  \text{mark } v_1 \text{ as visited.}
  \]
  
  \[
  \text{while List not empty:}
  \]
  \[
  v := \text{List.removeFirst()}.\]
  \[
  \text{if } v \text{ is } v_2:\n  \]
  \[
  \text{path is found.}
  \]
  
  \[
  \text{for each unvisited neighbor } v_i \text{ of } v\n  \text{ where there is an edge from } v \text{ to } v_i:\n  \]
  \[
  \text{mark } v_i \text{ as visited}\n  \]
  \[
  \text{List.addLast}(v_i).\]
  \]
  \[
  \text{path is not found.}
  \]
BFS observations

• *optimal*: in unweighted graphs, optimal. (fewest edges = best)
  – In weighted graphs, not optimal.
    (path with fewest edges might not have the lowest weight)

• *disadvantage*: harder to reconstruct what the actual path is once you find it
  – conceptually, BFS is exploring many possible paths in parallel, so it's not easy to store a Path array/list in progress

• observation: any particular vertex is only part of one partial path at a time
  – We can keep track of the path by storing *predecessors* for each vertex (references to the previous vertex in that path)
Another BFS example

• Using BFS, find a path from BOS to LAX.
DFS, BFS runtime

• What is the expected runtime of DFS, in terms of the number of vertices $V$ and the number of edges $E$?

• What is the expected runtime of BFS, in terms of the number of vertices $V$ and the number of edges $E$?

• Answer: $O(|V| + |E|)$
  – each algorithm must potentially visit every node and/or examine every edge once.
  – why not $O(|V| \times |E|)$?

• What is the space complexity of each algorithm?
public class VertexInfo<V> {
    public V v;
    public boolean visited;

    public VertexInfo(V v) {
        this.v = v;
        this.clear();
    }

    public void clear() {
        this.visited = false;
    }
}