Announcements

- For the next lecture, you MUST watch the tutorial on the shortest path algorithm beforehand: http://www.cs.cornell.edu/courses/cs2110/2017fa/online/shortestPath/shortestPath.html
- Thursday’s lecture will assume that you understand it. Watch the tutorial once or twice and execute the algorithm on a small graph.

Graphs

Representing Graphs

Graph Algorithms

- Search
  - Depth-first search
  - Breadth-first search
- Shortest paths
  - Dijkstra’s algorithm
- Spanning trees
  - Algorithms based on properties
    - Minimum spanning trees
  - Prim’s algorithm
  - Kruskal’s algorithm

Search on Graphs

- Given a graph \((V, E)\) and a vertex \(u \in V\)
- We want to "visit" each node that is reachable from \(u\)

There are many paths to some nodes.

How do we visit all nodes efficiently, without doing extra work?
Depth-First Search

/** Visit all nodes reachable on unvisited paths from u. 
Precondition: u is unvisited. */
public static void dfs(int u) {
    visited[u] = true;
    for all edges (u,v):
        if(!visited[v]):
            dfs(v);
}

Intuition: Recursively visit all vertices that are reachable along unvisited paths.

Suppose there are \( n \) vertices that are reachable along unvisited paths and \( e \) edges:

- Worst-case running time: \( O(n + e) \)
- Worst-case space: \( O(n) \)

Depth-First Search Iteratively

/** Visit all nodes reachable on unvisited paths from u. 
Precondition: u is unvisited. */
public static void dfs(int u) {
    Stack s= (u);
    while (s is not empty ):
        u = s.pop();
        if(u not visited) {
            visit u;
            for each edge (u, v):
                s.push(v);
        }
}

Intuition: Iteratively process the graph in "layers" moving further away from the source node.

Breadth-First Search

/** Visit all nodes reachable on unvisited paths from u. 
Precondition: u is unvisited. */
public static void bfs(int u) {
    Queue q= (u);
    while (q is not empty ) {
        u = q.remove();
        if(u not visited) {
            visit u;
            for each edge (u, v):
                q.add(v);
        }
    }

Intuition: Iteratively process the graph in "layers" moving further away from the source node.

Analyzing BFS

/** Visit all nodes reachable on unvisited paths from u. 
Precondition: u is unvisited. */
public static void bfs(int u) {
    Queue q= (u);
    while (q is not empty ) {
        u = q.remove();
        if(u not visited) {
            visit u;
            for each edge (u, v):
                q.add(v);
        }
    }

Suppose there are \( n \) vertices that are reachable along unvisited paths and \( e \) edges:

- Worst-case running time: \( O(n + e) \)
- Worst-case space: \( O(e) \)
Comparing Search Algorithms

<table>
<thead>
<tr>
<th>DFS</th>
<th>BFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visits: 1, 2, 3, 5, 7, 8</td>
<td>Visits: 1, 2, 5, 7, 3, 8</td>
</tr>
<tr>
<td>Time: $O(n + e)$</td>
<td>Time: $O(n + e)$</td>
</tr>
<tr>
<td>Space: $O(n)$</td>
<td>Space: $O(e)$</td>
</tr>
</tbody>
</table>