GRAPH SEARCH

Lecture 17
CS 2110 Fall 2017
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

Adjacency List

1 → 2 → 4
2 → 3
3
4 → 2 → 3

Adjacency Matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
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?
**Visit all nodes reachable on unvisited paths from u.**
Precondition: u is unvisited.

```java
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.

dfs(1) visits the nodes in this order: 1, 2, 3, 5, 7, 8
Depth-First Search

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

```java
/** 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);
}
```

Suppose there are $n$ vertices that are reachable along unvisited paths and $e$ edges:

Worst-case running time? $O(n + e)$
Worst-case space? $O(n)$
public class Node {
    boolean visited;
    List<Node> neighbors;

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

Intuition: Visit all vertices that are reachable along unvisited paths from the current 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);// Not Java!
    while (q is not empty) {
        u = q.remove();
        if (u not visited) {
            visit u;
            for each (u, v):
                q.add(v);
        }
    }
}
Analyzing BFS

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

```java
/** Visit all nodes reachable on unvisited paths from u.
   Precondition: u is unvisited. */
public static void bfs(int u) {
    Queue q = (u); // Not Java!
    while (               ) {
        u = q.remove();
        if (u not visited) {
            visit u;
            for each (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

DFS
- Visits: 1, 2, 3, 5, 7, 8
- Time: $O(n + e)$
- Space: $O(n)$

BFS
- Visits: 1, 2, 5, 7, 3, 8
- Time: $O(n + e)$
- Space: $O(e)$