GRAPH TRAVERSAL

Lecture 18
CS 2110 — Spring 2019
“Graphs”, topic 8: DFS, BFS
Graph Representations

Adjacency list

1 → 2 → 3
2 → 4
3 → 2 → 4
4

Adjacency matrix

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
4

**Review: Sparse vs. Dense**

*(improved definitions in Lec 17)*
Graph Search
Graph Traversal

**Goal:** *visit* each vertex that is reachable from some starting vertex

**And:** even if there are many paths to a node, *visit only once*

Two algorithms: DFS, BFS
Intuition: one person exploring a maze

Depth-First Search (DFS)
/** Visit every node reachable along a path of unvisited nodes from node v.
   Precondition: v is unvisited. */

void dfs(Vertex v) {
    mark v visited;
    for all edges (v,u):
        if u is unmarked:
            dfs(u);
}

dfs(1) visits the nodes in this order: 1, 2, 3, 5, 7, 8
Poll #1
/** Visit every node reachable along a path of unvisited nodes from node v.  
   Precondition: v is unvisited. */

void dfs(Vertex v) {
    mark v visited;
    for all edges (v,u):
        if u is unmarked:
            dfs(u);
}
DFS Space Efficiency

void dfs(Vertex v) {
    mark v visited;
    for all edges (v,u):
        if u is unmarked:
            dfs(u);
}

Suppose graph has V vertices and E edges

Space required?
• Mark for each vertex: O(V)
• Frame for each recursive call: O(V)

Worst case: O(V)
DFS Time Efficiency

```c
void dfs(Vertex v) {
    mark v visited;
    for all edges (v,u):
        if u is unmarked:
            dfs(u);
}
```

Time required?
- **Mark** each vertex: $O(V)$
- **Recursive call** for on each unvisited vertex: $O(V)$
- **Find each edge**
  - in adj list: $O(E)$: Worst case: $O(V+E)$
  - in adj matrix: $O(V^2)$: Worst case: $O(V^2)$
Variant: Iterative DFS

Same algorithm; non-recursive implementation

```java
void dfs(Vertex u) {
    Stack s= new Stack();
    s.push(u);
    while (s is not empty) {
        u= s.pop();
        if (u not visited) {
            visit u;
            for each edge (u, v):
                s.push(v);
        }
    }
}
```

Visit order was 1, 7, 8, 5, 2, 3: differs from before because of order edges processed
Breadth-First Search (BFS)

Intuition: Search party fanning out in all directions
Breadth-First Search

Idea: Iteratively process the graph in "layers" moving further away from the source vertex.
Breadth-First Search

Idea: Iteratively process the graph in "layers" moving further away from the source vertex.

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

Visit order was 1, 2, 5, 7, 3, 8
Poll #2
/** Visit all vertices reachable on unvisited paths from u. */

```java
void bfs(int u) {
    Queue q = new Queue
    q.add(u);
    while (q is not empty) {
        u = q.remove();
        if (u not visited) {
            visit u;
            for each (u, v):
                if (v not encountered) {
                    mark v as encountered;
                    q.add(v);
                }
            }
        }
    }
}
```

Idea: Don’t put vertex in queue if already encountered
/** Visit all vertices reachable on unvisited paths from u. */
void bfs(int u) {
    Queue q = new Queue
    q.add(u);
    while (q is not empty) {
        u = q.remove();
        if (u not visited) {
            visit u;
            for each (u, v):
                if (v not encountered) {
                    mark v as encountered;
                    q.add(v);
                }
        }
    }
}
### Comparing Traversal Algorithms

<table>
<thead>
<tr>
<th></th>
<th>DFS (recursive)</th>
<th>DFS &amp; BFS (iterative, improved*)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time</strong></td>
<td>$O(V+E)$ or $O(V^2)$</td>
<td>$O(V+E)$ or $O(V^2)$</td>
</tr>
<tr>
<td><strong>Space</strong></td>
<td>$O(V)$</td>
<td>$O(V)$</td>
</tr>
</tbody>
</table>

*Without improvement, space becomes $O(E)$*