Graph Traversal

**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>
```

**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

JavaHyperText Topics

“Graphs”, topic 8: DFS, BFS

Review: Sparse vs. Dense

(improved definitions in Lec 17)
Depth-First Search

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

Depth-First Search

/** 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);
}

Suppose graph has V vertices and E edges

DFS Space Efficiency

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

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

Worst case: O(V)

DFS Time 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

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**

```java
void dfs(Vertex u) {
    Stack s = new Stack();
    s.push(u);
    while (!s.isEmpty()) {
        u = s.pop();
        if (!u.isVisited) {
            u.visit();
            for each edge (u, v):
                s.push(v);
        }
    }
}
```

Same algorithm; non-recursive implementation

Visit order was 1, 7, 8, 5, 2, 3: differs from before because of order edges processed

---

**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.isEmpty()) {
        u = q.remove();
        if (!u.isVisited) {
            u.visit();
            for each (u, v):
                if (!v.isEncountered) {
                    mark v as encountered;
                    q.add(v);
                }
        }
    }
}
```

Idea: Don’t put vertex in queue if already encountered

Visit order was 1, 2, 5, 7, 3, 8

---

**Poll #2**

---

**Improved BFS**

```java
/** Visit all vertices reachable on
unvisited paths from u. */
void bfs(int u) {
    Queue q = new Queue();
    q.add(u);
    while (!q.isEmpty()) {
        u = q.remove();
        if (!u.isVisited) {
            u.visit();
            for each (u, v):
                if (!v.isEncountered) {
                    mark v as encountered;
                    q.add(v);
                }
        }
    }
}
```
Analyzing BFS

```c
/** 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);
                }
        }
    }
}
```

Same as DFS.
Time: O(V)
Space: O(V)

Comparing Traversal Algorithms

- **DFS (recursive)**
  - Time: O(V+E) or O(V^2)
  - Space: O(V)

- **DFS & BFS (iterative, improved*)**
  - Time: O(V+E) or O(V^2)
  - Space: O(V)

*Without improvement, space becomes O(E)