Data Structures and Algorithms

Graph Traversals

Chris Brooks

Department of Computer Science

University of San Francisco
Graph Traversals

- Visit every vertex, in an order defined by the topology of the graph.

- Two major traversals:
  - Depth First Search
  - Breadth First Search
Starting from a specific node (pseudo-code):

```java
DFS(Edge G[], int vertex, boolean Visited[]) {
    Visited[vertex] = true;
    for each node w adjacent to vertex:
        if (!Visited[w])
            DFS(G, w, Visited);
}
```
class Edge {
    public int neighbor;
    public int next;
}

void DFS(Edge G[], int vertex, boolean Visited[]) {
    Edge tmp;
    Visited[vertex] = true;
    for (tmp = G[vertex]; tmp != null; tmp = tmp.next) {
        if (!Visited[tmp.neighbor])
            DFS(G, tmp.neighbor, Visited);
    }
}
Example

- Visited nodes circled in red

![Graph Diagram]

19-3: Depth First Search
Example

- Visited nodes circled in red

DFS(0)
Example
- Visited nodes circled in red

DFS(0)
DFS(1)
Depth First Search

Example

- Visited nodes circled in red

```
DFS(0)
  DFS(1)
    DFS(3)
```
Example

- Visited nodes circled in red

DFS(0)
DFS(1)
DFS(3)
DFS(4)
19-8: Depth First Search

Example
- Visited nodes circled in red

DFS(0)
  DFS(1)
    DFS(3)
      DFS(4)
  DFS(2)

Diagram:
- Nodes 0, 1, 2, 3, 4, 5, 6
- Edges connect nodes
- Nodes 0, 1, 2, and 3 visited
- Nodes 4, 5, 6 not visited
Example

- Visited nodes circled in red

DFS(0)
  DFS(1)
  DFS(3)
  DFS(4)
  DFS(2)
  DFS(5)
Example

- Visited nodes circled in red

DFS(0)
  DFS(1)
    DFS(3)
      DFS(4)
    DFS(2)
      DFS(5)
      DFS(6)
To visit every node in the graph:

```java
TraverseDFS(Edge G[]) {
    int i;
    boolean Visited = new Edge[G.length];
    for (i=0; i<G.length; i++)
        Visited[i] = false;
    for (i=0; i<G.length; i++)
        if (!Visited[i])
            DFS(G, i, Visited);
}
```
Examples

0——2——4
|   |   |
1——3——5——6

19-12: Depth First Search
6 Examples

- Depth First Search

```
0 2
/|
/ |
1 3
```

```
4
/|
/ |
5 6
```

```
7
```

```
8
```
Keep track of what nodes we have left using a stack
Recursive version implicitly uses the system stack
Can write DFS non-recursively, using our own stack
DFS, using recursion

```java
void DFS(Edge G[], int vertex, boolean Visited[]) {
    Edge tmp;
    Visited[vertex] = true;
    for (tmp = G[vertex]; tmp != null; tmp = tmp.next) {
        if (!Visited[tmp.neighbor])
            DFS(G, tmp.neighbor, Visited);
    }
}
```
DFS, using stack

```java
void DFS(Edge G[], int vertex, boolean Visited[]) {
    Edge tmp;
    int nextV;
    Stack S = new Stack();

    S.push(new Integer(vertex));
    while (!S.empty()) {
        nextV = ((Integer) S.pop()).intValue();
        if (!Visited[nextV]) {
            Visited[nextV] = true;
            for (tmp = G[nextV]; tmp != null; tmp = tmp.next) {
                S.push(new Integer(tmp.neighbor));
            }
        }
    }
}
```
19-17: Breadth First Search

- **DFS:** Look as *Deep* as possible, before looking wide
  - Examine all descendants of a node, before looking at siblings

- **BFS:** Look as *Wide* as possible, before looking deep
  - Visit all nodes 1 away, then 2 away, then three away, and so on
Examples

0  2  4
   \  /  \
  1  3
   /  \
  5  6
19-19: Breadth First Search

- Coding BFS:
  - Use a queue instead of a stack

```java
void BFS(Edge G[], int vertex, boolean Visited[]) {
    Edge tmp;
    int nextV;
    Queue Q = new Queue();

    Q.enqueue(new Integer(vertex));
    while (!Q.empty()) {
        nextV = ((Integer) Q.dequeue()).intValue();
        if (!Visited[nextV]) {
            Visited[nextV] = true;
            for (tmp = G[nextV]; tmp != null; tmp = tmp.next) {
                Q.enqueue(new Integer(tmp.neighbor()));
            }
        }
    }
}
```
Example

- Visited nodes circled

```
  0 ─ 1 ─ 3
  |   |   |
  4 ─ 2 ─ 6
  |   |
  5
```
Example

- Visited nodes circled

```
Queue:
0
```

```
0 - 1 - 3
|    |
|    |
|    |
2 - 5
```

```
0 - 2 - 6
|    |
|    |
|    |
4 - 3
```
Example

Visited nodes circled

Queue:
1 2 4
Example

Visited nodes circled

Queue:
2 4 0 3 4

19-23: Breadth First Search
Example

- Visited nodes circled

```
Queue:
4034056
```
6 Example
   ▲ Visited nodes circled

Queue:
0 3 4 5 6 0 1 3

Diagram:
0 -> 1 -> 3
0 -> 2 -> 6
0 -> 4

Department of Computer Science — University of San Francisco – p.27/63
Example

Visited nodes circled

Queue:
3 4 0 5 6 0 1 3
Example

Visited nodes circled

Queue:
405601314
Example

Visited nodes circled

Queue:
05601314
Example

- Visited nodes circled

Queue: 5601314
Example

Visited nodes circled

```
Queue:
60131426
```
Example

Visited nodes circled

Queue:
013142625
Previous code differs slightly from what is in the text

- Textbook marks nodes as VISITED when they are placed on the queue
- Previous code marks nodes as VISITED as they are removed from the queue
6 Coding BFS (Alternate version):

```java
void BFS(Edge G[], int vertex, boolean Visited[]) {
    Edge tmp;
    int nextV;
    Queue Q = new Queue();
    Visited[vertex] = true;
    Q.enqueue(new Integer(vertex));
    while (!Q.empty()) {
        nextV = ((Integer) Q.dequeue()).intValue();
        for (tmp = G[nextV]; tmp != null; tmp = tmp.next) {
            if (!Visited[tmp.neighbor]) {
                Visited[tmp.neighbor] = true;
                Q.enqueue(new Integer(tmp.neighbor));
            }
        }
    }
}
```
Previous code differs slightly from what is in the text
- Textbook marks nodes as VISITED when they are placed on the queue
- Previous code marks nodes as VISITED as they are removed from the queue

How does execution differ?
19-35: Breadth First Search

Previous code differs slightly from what is in the text

- Textbook marks nodes as VISITED when they are placed on the queue
- Previous code marks nodes as VISITED as they are removed from the queue

How does execution differ?

- Version I: A vertex is added to the queue for each edge in the graph (so the same vertex can be added to the queue more than once
- Version II: Each vertex is added to the queue at most once
Example

- Visited nodes circled

```
0 -- 1 -- 3
  |    |
0 -- 2 -- 5

0 -- 4
```

19-36: Breadth First Search
Example

- Visited nodes circled

```
Queue:
0
```

Diagram:

```
0
\(\rightarrow\) 1 \(\rightarrow\) 3
\(\rightarrow\) 4
\(\rightarrow\) 2 \(\rightarrow\) 6
\(\rightarrow\) 5
```
Example

- Visited nodes circled

```
Queue:  
124
```

0 1 2 3 4 5 6

Department of Computer Science — University of San Francisco — p.40/63
Example

- Visited nodes circled

Queue: 243
Example

Visited nodes circled

Queue: 4356
Example

Visited nodes circled

Queue:
3 5 6
Example

Visited nodes circled

Queue:
5 6
Example

- Visited nodes circled

```
Queue:
  6
```

```
0 ———- 1 ———- 3
    |         |
    v         v
  4 ———- 2 ———- 6
    |             |
    v             v
      5
```
Example

- Visited nodes circled

```
Queue:
```

![Diagram of a graph showing Breadth First Search example with nodes and edges marked.]

Department of Computer Science — University of San Francisco – p.46/63
19-45: Search Trees

- Describes the order that nodes are examined in a traversal

- Directed Tree
  - Directed edge from $v_1$ to $v_2$ if the edge $(v_1, v_2)$ was followed during the traversal
Starting from node 0, adjacency list sorted by vertex number:

1 3 5 6
0 2 4

19-46: DFS Search Trees
Starting from node 0, adjacency list sorted by vertex number:
6 Starting from node 2, adjacency list sorted by vertex number:

```
0 — 2 — 4
|   |   |
1   3   5   6
```
Starting from node 2, adjacency list sorted by vertex number:
Starting from node 2, adjacency list sorted by vertex number:
Starting from node 2, adjacency list sorted by vertex number:
Starting from node 0, adjacency list sorted by vertex number:
Starting from node 0, adjacency list sorted by vertex number:
Starting from node 2, adjacency list sorted by vertex number:
Starting from node 2, adjacency list sorted by vertex number:
Starting from node 0, adjacency list sorted by vertex number:
Starting from node 0, adjacency list sorted by vertex number:
Starting from node 2, adjacency list sorted by vertex number:

```
0 ——— 2 ——— 4
   |     |     |
1 ——— 3 ——— 5 ——— 6
```
Starting from node 2, adjacency list sorted by vertex number:
6 Starting from node 0, adjacency list sorted by vertex number:
Starting from node 0, adjacency list sorted by vertex number: