Data Structures and Algorithms

CS245-2016S-16

Graph Traversals

BFS & DFS

David Galles

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
16-1: Depth 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 Edge 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
16-4: Depth First Search

- Example
  - Visited nodes circled in red

DFS(0)
Example

Visited nodes circled in red

DFS(0)
DFS(1)
Example

Visited nodes circled in red

DFS(0)
DFS(1)
DFS(3)
16-7: Depth First Search

- Example
  - Visited nodes circled in red

```
DFS(0)
DFS(1)
DFS(3)
DFS(4)
```
**Example**

- Visited nodes circled in red

DFS(0)
DFS(1)
DFS(3)
DFS(4)
DFS(2)
Depth First Search

- 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);
}
```
16-12: Depth First Search

- Examples

Diagram:

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

16-13: Depth First Search

- Examples

```
0 -- 2
|   |   |
|   |   |   
|   |   |   
|   |   |   |
1 -- 3
```
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 & Stacks

- 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));
            }
        }
    }
}
```
16-17: **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
16-18: Breadth First Search

- Examples

```
0 --- 2 --- 4
 |   |   |
|   |   |
1 --- 3   5 --- 6
```
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[next] = true;
            for (tmp = G[nextV]; tmp != null; tmp = tmp.next) {
                Q.enqueue(new Integer(tmp.neighbor()));
            }
        }
    }
}
```
16-20: Breadth First Search

- Example
  - Visited nodes circled

![Graph diagram showing nodes and connections for breadth first search example.](image-url)
Example
- Visited nodes circled
Example

Visited nodes circled

Queue: 1 2 4
**Example**

- Visited nodes circled

```
Queue:
2 4 0 3 4
```
Example
- Visited nodes circled

Graph:
- Nodes: 0, 1, 2
- Edges: 0-1, 0-3, 1-4, 2-5, 2-6

Queue:
- 0, 1, 2, 3, 4, 5, 6

16-24: Breadth First Search
16-25: Breadth First Search

• Example
  • Visited nodes circled

Queue: 034056013
Breadth First Search

- Example
  - Visited nodes circled

```
Queue:
3 4 5 6 0 1 3
```

Diagram:
```
0 -- 2 -- 6
|    |
|    |
0 -- 4 -- 1
```

Nodes 3, 4, and 6 are in the queue.
Breadth First Search

- Example
  - Visited nodes circled

Queue: 405601314
Example

Visited nodes circled

Queue:
0 5 6 0 1 3 1 4
Breadth First Search

Example

- Visited nodes circled

Queue: 5 6 0 1 3 1 4

Diagram:
- Nodes 0, 1, 2, 3, 4, 5, 6
- Connections between nodes
Breadth First Search

- Example
  - Visited nodes circled

![Graph Example](image-url)
Example

Visited nodes circled

Queue:
0 1 3 1 4 2 6 2 5

Breadth First Search
Alternate version of BFS
- Previous code marks nodes as VISITED as they are removed from the queue
- We could also mark nodes as VISITED when they are placed on the queue
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));
            }
        }
    }
}
Alternate version of BFS

- Previous code marks nodes as VISITED as they are removed from the queue
- We could also mark nodes as VISITED when they are placed on the queue

How does execution differ?
Alternate version of BFS

- Previous code marks nodes as VISITED as they are removed from the queue
- We could also mark nodes as VISITED when they are placed on 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

Breadth First Search

Diagram of a graph with nodes labeled 0 to 6 and edges connecting them.
Example

Visited nodes circled

Queue:
0

Breadth First Search
Breadth First Search

- Example
  - Visited nodes circled
16-39: **Breadth First Search**

- Example
  - Visited nodes circled

![Graph](image)

*Queue:* 243
Breadth First Search

- Example
  - Visited nodes circled

Queue: 4 3 5 6
Breadth First Search

- Example
  - Visited nodes circled

![Graph Diagram]

Queue: 356
**Example**

- Visited nodes circled

```
Queue:
56
```
16-43: Breadth First Search

• Example
  • Visited nodes circled

Queue:
6

Diagram of a graph with nodes 0, 1, 2, 3, 4, 5, and 6 connected by edges, illustrating the breadth-first search algorithm.
Breadth First Search

- Example
  - Visited nodes circled

.Queue:
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:
• 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
```

DFS Search Trees
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:
BFS Search Trees

- Starting from node 0, adjacency list sorted by vertex number:
16-58: BFS Search Trees

- Starting from node 2, adjacency list sorted by vertex number:
Starting from node 2, adjacency list sorted by vertex number:
16-60: DFS in Directed Graphs

- Starting from node 0, adjacency list sorted by vertex number:
Starting from node 0, adjacency list sorted by vertex number: