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 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
16-5: Depth First Search

- Example
  - Visited nodes circled in red

```
DFS(0)
  DFS(1)
```

Diagram:
```
0 -- 1 -- 3
|     |     |
|     |     |
|     |     |
2 -- 5
```
```
16-6: Depth First Search

- Example
  - Visited nodes circled in red
16-7: Depth First Search

- Example
  - Visited nodes circled in red

![Graph diagram showing Depth First Search example]

DFS(0)
DFS(1)
DFS(3)
DFS(4)
16-8: Depth First Search

- Example
  - Visited nodes circled in red

![Diagram of Depth First Search example]
Example

Visited nodes circled in red

DFS(0)
DFS(1)
DFS(3)
DFS(4)
DFS(2)
DFS(5)
16-10: Depth First Search

- 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

```
0 -- 2 -- 4
   |    |
  1 -- 3
```

```
2 -- 4
   |    
  3 -- 5
```

```
4 -- 6
```
16-13: Depth First Search

- Examples

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

```
4 -- 7
|   |
|   |
5 -- 8
```
DFS & Stacks

- 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);
    }
}
```
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
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()));
            }
        }
    }
}
```
Breadth First Search

- Example
  - Visited nodes circled

```
0 ——— 1 ——— 3
   |         |
   v         v
4 ——— 2 ——— 6
    |         |
    v         v
5
```
16-21: Breadth First Search

- Example
  - Visited nodes circled

```
Queue:
0
```

```
0 - 1 - 2 - 5
    /   \
   3     6
   |     /\
   4  0  5
```
Example

Visited nodes circled

Queue: 1 2 4
16-23: Breadth First Search

- Example
  - Visited nodes circled

```
Queue:
2 4 0 3 4
```
16-24: Breadth First Search

- Example
  - Visited nodes circled

```
Queue: 4034056
```
Breadth First Search

- Example
  - Visited nodes circled

```
Queue:
0 3 4 5 6 0 1 3
```
16-26: Breadth First Search

- Example
- Visited nodes circled

Queue: 34 0 5 6 0 1 3
**Example**

- Visited nodes circled

![Breadth First Search Diagram]

Queue: 405601314
Example

Visited nodes circled

Queue:
0 5 6 0 1 3 4 14
16-29: Breadth First Search

- Example
  - Visited nodes circled

```
Queue: 5601314
```
16-30: Breadth First Search

- Example
  - Visited nodes circled

Queue:
60131426
Breadth First Search

- Example
- Visited nodes circled

Queue:
0 1 3 4 2 6 5

Diagram of a graph with nodes numbered 0 to 6 and edges connecting them.
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
Breadth First Search

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));
            }
        }
    }
}
```
**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
- 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
Breadth First Search

- Example
  - Visited nodes circled

![Diagram of Breadth First Search example](attachment:image.png)
16-37: Breadth First Search

- Example
  - Visited nodes circled

```
Queue:
0
```

Diagram:
```
0 -- 1 -- 3
|     |     |
|     |     |
|     |     |
0     4

```

```
2 -- 6
|     |
|     |
|     |
2     5
```
Example

Visited nodes circled
Breadth First Search

- Example
  - Visited nodes circled

Queue: 243
**Breadth First Search**

- **Example**
  - Visited nodes circled

![Diagram of a graph with nodes and edges, showing a Breadth First Search example with visited nodes circled and a queue: 4356.](image-url)
Example

Visited nodes circled

Queue:
3 5 6
Example

Visited nodes circled

Queue:
5 6
**Example**
- Visited nodes circled

Queue:

6
16-44: Breadth First Search

- Example
  - Visited nodes circled

```
Queue:
```

```
0 -- 1 -- 3
|     |
v     v
0 -- 2 -- 6
     |     |
     v     v
      5 -- 4
```
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:
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:
BFS Search Trees

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