16-0: **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);
}
```

16-2: **Depth First Search**

```java
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);
    }
}
```

16-3: **Depth First Search**

- Example
  - Visited nodes circled in red

```
1 3
/
0 4 2 6
```

16-4: **Depth First Search**
• Example
  • Visited nodes circled in red

Depth First Search

16-5: Depth First Search

• Example
  • Visited nodes circled in red

Depth First Search

16-6: Depth First Search

• Example
  • Visited nodes circled in red

Depth First Search

16-7: Depth First Search
Graph Traversals

• Example
  • Visited nodes circled in red

16-8: Depth First Search

16-9: Depth First Search

16-10: Depth First Search
Graph Traversals

• Example
  • Visited nodes circled in red

![Graph Diagram](image)

DFS(0)
  DFS(1)
  DFS(3)
  DFS(4)
  DFS(2)
  DFS(5)
  DFS(6)

16-11: Depth First Search

• To visit every node in the graph:

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

![Graph Diagram](image)

16-13: Depth First Search

• Examples
16-14: **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

16-15: **DFS & Stacks**

- 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);
    }
}
```

16-16: **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
```

16-19: **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()));
      }
    }
  }
}
```

16-20: **Breadth First Search**

- Example
  - Visited nodes circled

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

16-21: **Breadth First Search**

- Example
16-22: **Breadth First Search**

- Example
  - Visited nodes circled

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

Queue: 0

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

Queue: 0 1 2 4

16-23: **Breadth First Search**

- Example
  - Visited nodes circled

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

Queue: 0 1 2 4

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

Queue: 0 1 2 4 3 6

16-24: **Breadth First Search**

- Example
Graph Traversals

BFS & DFS

- Visited nodes circled

\[\text{Queue:} \quad 4 \quad 0 \quad 3 \quad 4 \quad 0 \quad 5 \quad 6 \]

16-25: Breadth First Search
- Example
  - Visited nodes circled

\[\text{Queue:} \quad 0 \quad 3 \quad 4 \quad 0 \quad 5 \quad 6 \quad 0 \quad 1 \quad 3 \]

16-26: Breadth First Search
- Example
  - Visited nodes circled

\[\text{Queue:} \quad 0 \quad 3 \quad 4 \quad 0 \quad 5 \quad 6 \quad 3 \quad 4 \quad 0 \quad 6 \quad 1 \quad 3 \]

16-27: Breadth First Search
- Example
Graph Traversals

16-28: **Breadth First Search**

- Example
  - Visited nodes circled

16-29: **Breadth First Search**

- Example
  - Visited nodes circled

16-30: **Breadth First Search**

- Example
• Visited nodes circled

16-31: Breadth First Search

• Example

16-32: 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

16-33: 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));
      }
    }
  }
}
```

16-34: 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

16-35: **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?

16-36: **Breadth First Search**
• Example
  • Visited nodes circled

16-37: **Breadth First Search**
• Example
  • Visited nodes circled

Queue:

<table>
<thead>
<tr>
<th>Queue:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
16-38: **Breadth First Search**
- Example
  - Visited nodes circled

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

16-40: **Breadth First Search**
- Example
  - Visited nodes circled

16-41: **Breadth First Search**
• Example
  • Visited nodes circled

16-42: **Breadth First Search**

• Example
  • Visited nodes circled

16-43: **Breadth First Search**

• Example
  • Visited nodes circled

16-44: **Breadth First Search**
• Example
  • Visited nodes circled

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

16-45: **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

16-46: **DFS Search Trees**

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

```
0 2 4
1 3
```

16-47: **DFS Search Trees**

• Starting from node 0, adjacency list sorted by vertex number:
16-48: DFS Search Trees

- Starting from node 2, adjacency list sorted by vertex number:

16-49: DFS Search Trees

- Starting from node 2, adjacency list sorted by vertex number:

16-50: DFS Search Trees

- Starting from node 2, adjacency list sorted by vertex number:
16-51: **DFS Search Trees**

- Starting from node 2, adjacency list sorted by vertex number:

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

16-52: **DFS Search Trees**

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

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

16-53: **DFS Search Trees**

- Starting from node 0, adjacency list sorted by vertex number:
16-54: DFS Search Trees

- Starting from node 2, adjacency list sorted by vertex number:

16-55: DFS Search Trees

- Starting from node 2, adjacency list sorted by vertex number:

16-56: BFS Search Trees

- Starting from node 0, adjacency list sorted by vertex number:
16-57: BFS Search Trees

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

\[0 \rightarrow 2 \rightarrow 4\]
\[1 \rightarrow 3 \rightarrow 5 \rightarrow 6\]

16-58: BFS Search Trees

- Starting from node 2, adjacency list sorted by vertex number:

\[0 \rightarrow 2 \rightarrow 4\]
\[1 \rightarrow 3 \rightarrow 5 \rightarrow 6\]

16-59: BFS Search Trees

- Starting from node 2, adjacency list sorted by vertex number:
Graph Traversals

BFS & DFS

16-60: DFS in Directed Graphs

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

16-61: DFS in Directed Graphs

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