

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):

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

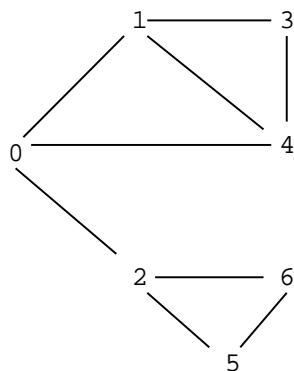
16-2: Depth First Search

```
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);
    }
}
```

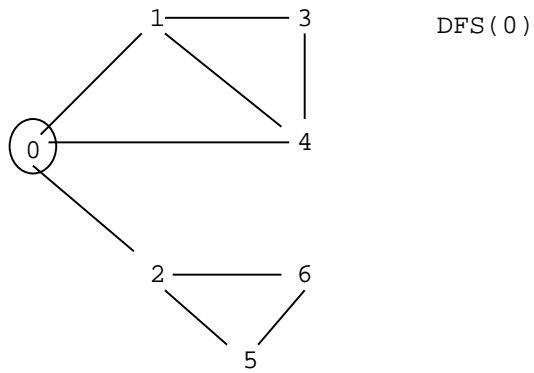
16-3: Depth First Search

- Example
 - Visited nodes circled in red

**16-4: Depth First Search**

- Example

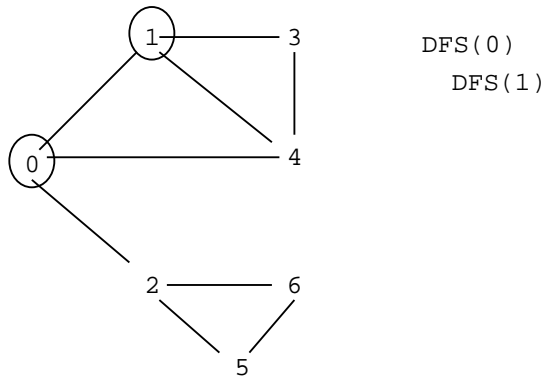
- Visited nodes circled in red



16-5: **Depth First Search**

- Example

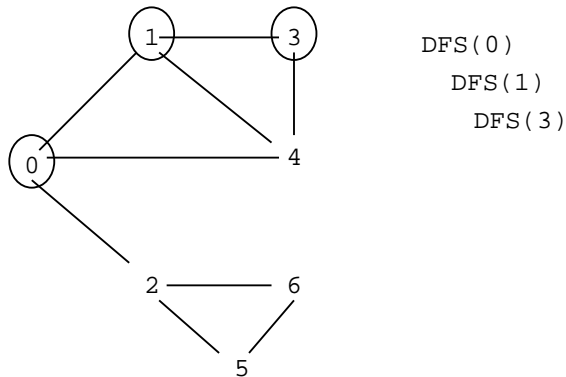
- Visited nodes circled in red



16-6: **Depth First Search**

- Example

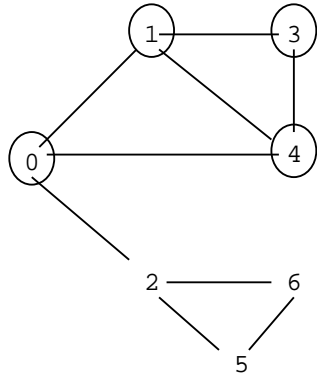
- Visited nodes circled in red



16-7: **Depth First Search**

- Example

- Visited nodes circled in red

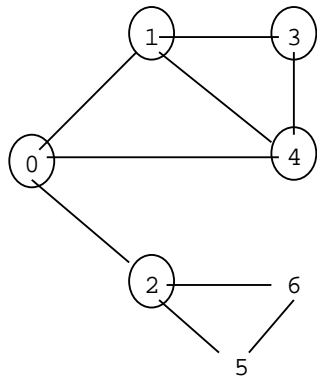


```
DFS(0)
  DFS(1)
    DFS(3)
      DFS(4)
```

16-8: **Depth First Search**

- Example

- Visited nodes circled in red

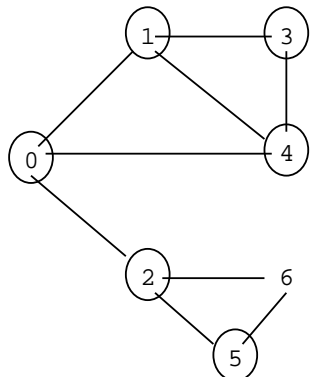


```
DFS(0)
  DFS(1)
    DFS(3)
      DFS(4)
        DFS(2)
```

16-9: **Depth First Search**

- Example

- Visited nodes circled in red

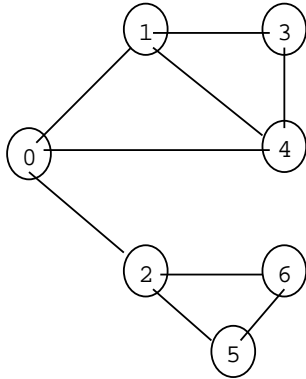


```
DFS(0)
  DFS(1)
    DFS(3)
      DFS(4)
        DFS(2)
          DFS(5)
```

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)
  
```

16-11: Depth First Search

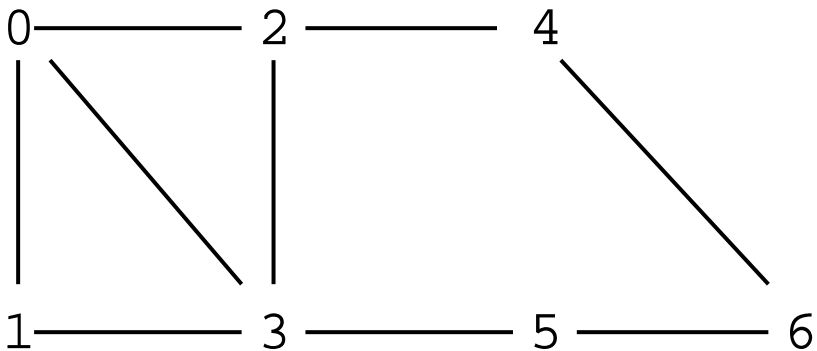
- To visit every node in the graph:

```

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);
}
  
```

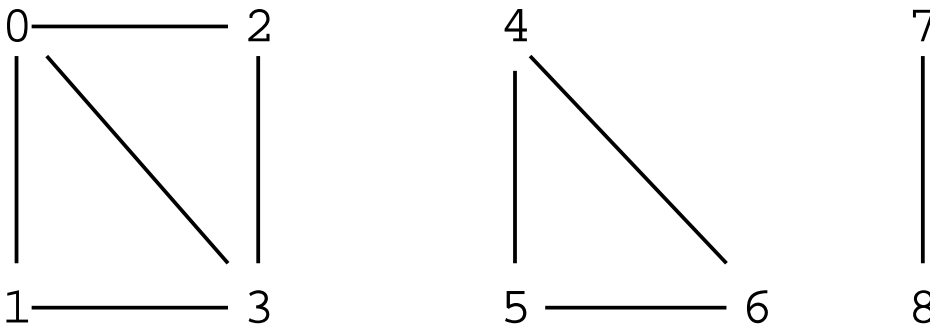
16-12: Depth First Search

- Examples



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

```
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

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

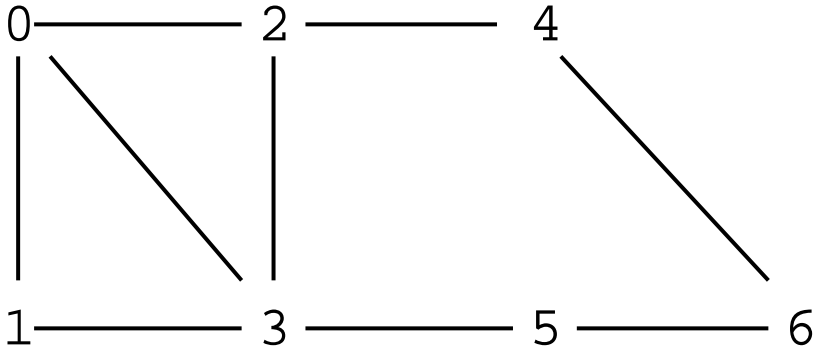
    S.push(new Integer(vertex));
    while (!S.empty()) {
        nextV = ((Integer) S.pop()).intValue();
        if (!Visited[nextV]) {
            Visited[nextV] = true;
            for (tmp = G[nextV]; tmp != null; tmp = tmp.next) {
                S.push(new Integer(tmp.neighbor));
            }
   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

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

16-19: **Breadth First Search**

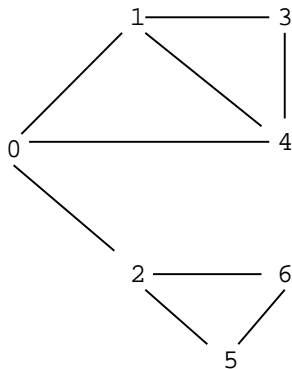
- Coding BFS:
 - Use a queue instead of a stack

```
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()));
            }
        }
    }
}
```

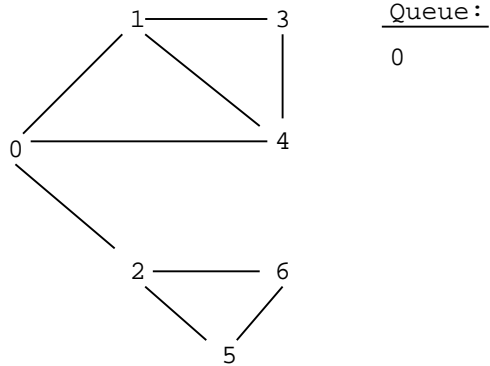
16-20: **Breadth First Search**

- Example
 - Visited nodes circled

16-21: **Breadth First Search**

- Example

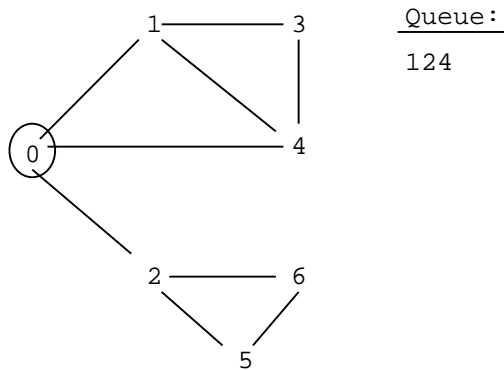
- Visited nodes circled



16-22: **Breadth First Search**

- Example

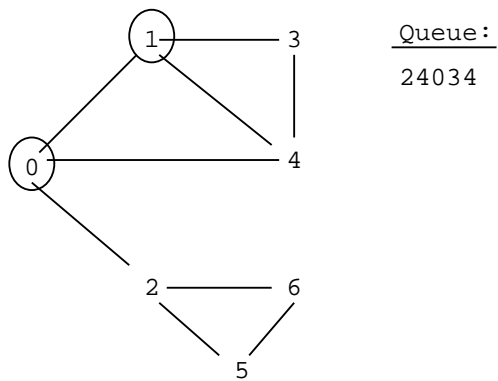
- Visited nodes circled



16-23: **Breadth First Search**

- Example

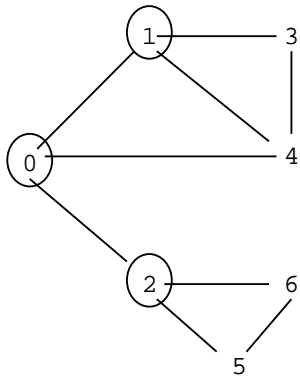
- Visited nodes circled



16-24: **Breadth First Search**

- Example

- Visited nodes circled

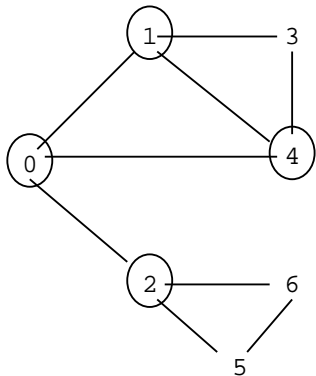


Queue:
4034056

16-25: **Breadth First Search**

- Example

- Visited nodes circled

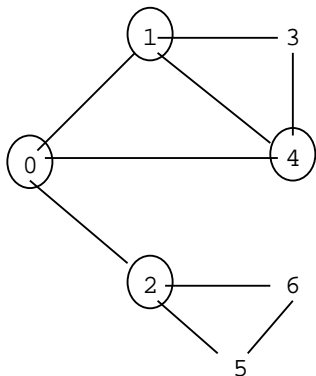


Queue:
034056013

16-26: **Breadth First Search**

- Example

- Visited nodes circled

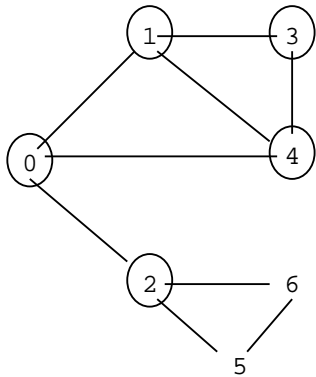


Queue:
34056013

16-27: **Breadth First Search**

- Example

- Visited nodes circled

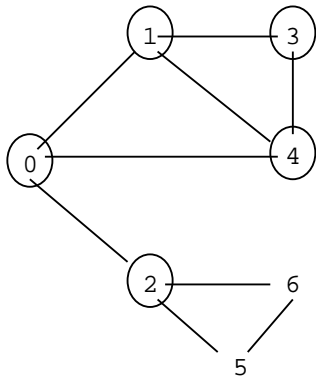


Queue:
405601314

16-28: **Breadth First Search**

- Example

- Visited nodes circled

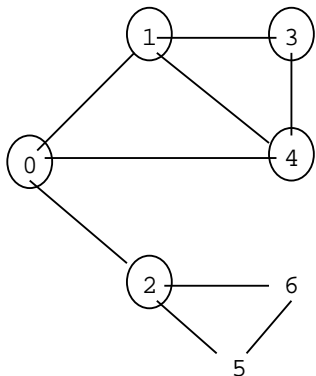


Queue:
05601314

16-29: **Breadth First Search**

- Example

- Visited nodes circled

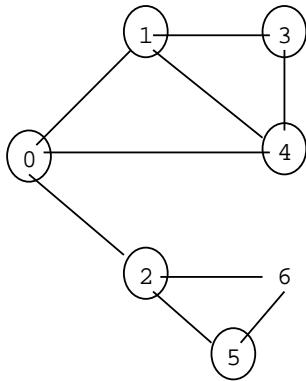


Queue:
5601314

16-30: **Breadth First Search**

- Example

- Visited nodes circled

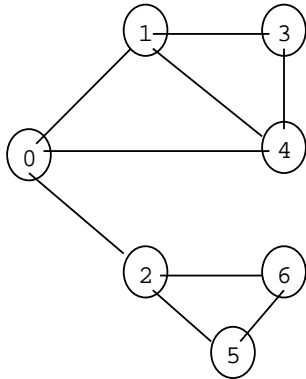


Queue:
60131426

16-31: **Breadth First Search**

- Example

- Visited nodes circled



Queue:
013142625

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):

```
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**

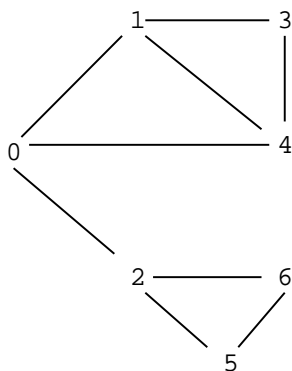
- Alternate version of BFS
 - Previous code marks nodes as VISITED as they are removed from the queue
 - We could also mark nodes as VISITED when they are placed on the queue
- How does execution differ?

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?
- 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

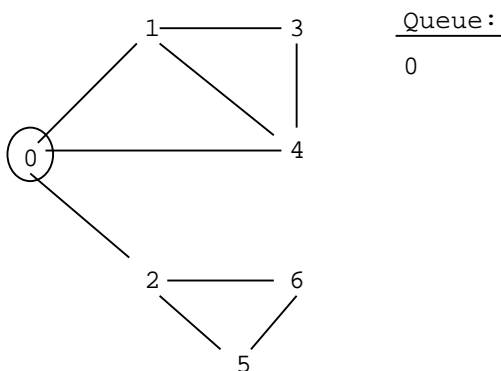
16-36: **Breadth First Search**

- Example
 - Visited nodes circled



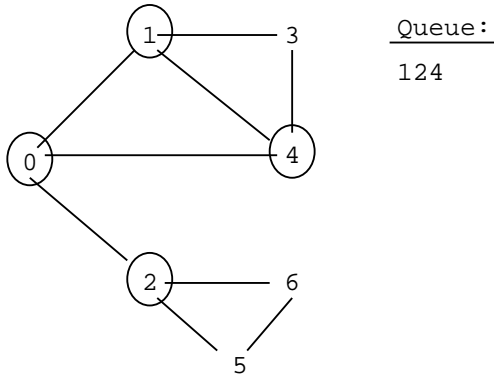
16-37: **Breadth First Search**

- Example
 - Visited nodes circled



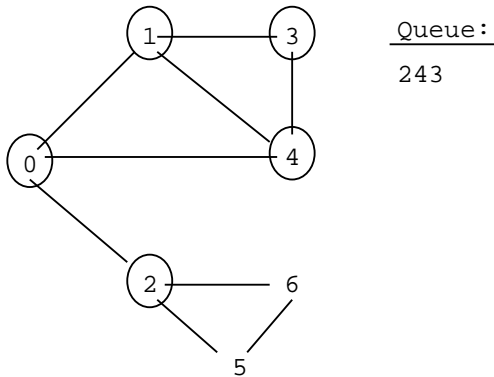
16-38: **Breadth First Search**

- Example
 - Visited nodes circled



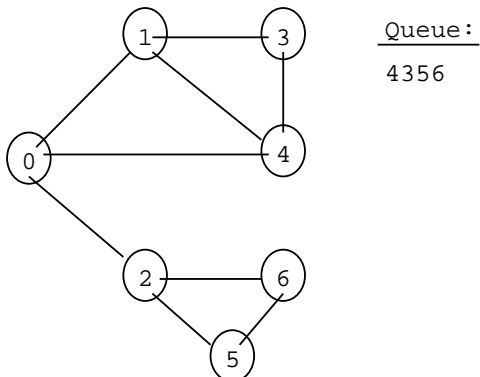
16-39: **Breadth First Search**

- Example
 - Visited nodes circled



16-40: **Breadth First Search**

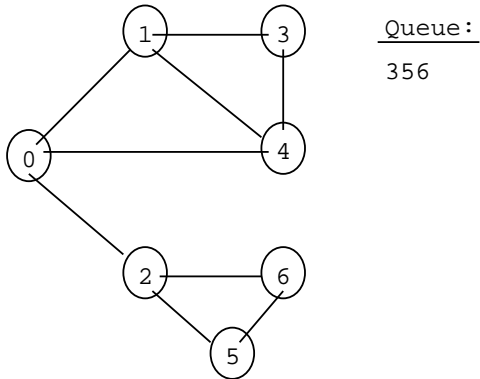
- Example
 - Visited nodes circled



16-41: **Breadth First Search**

- Example

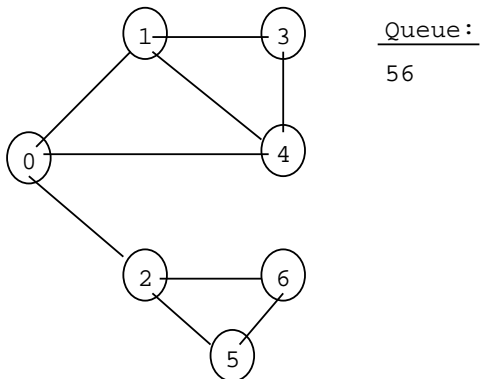
- Visited nodes circled



16-42: **Breadth First Search**

- Example

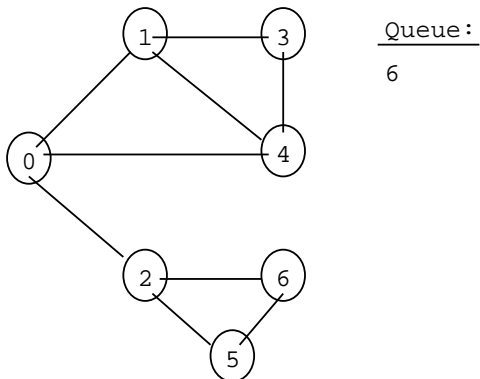
- Visited nodes circled



16-43: **Breadth First Search**

- Example

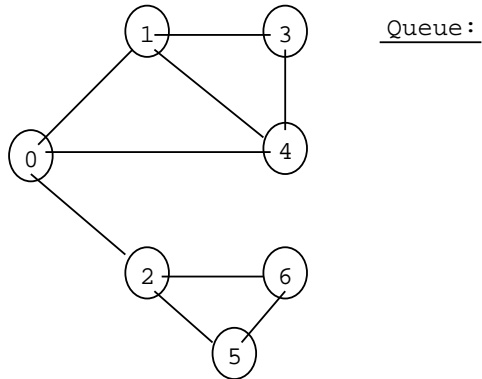
- Visited nodes circled



16-44: **Breadth First Search**

- Example

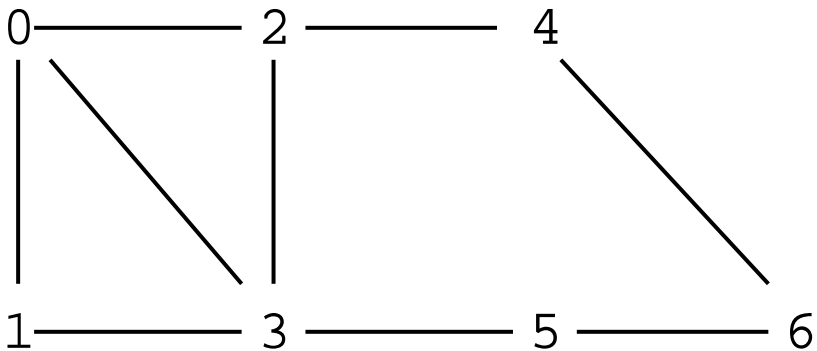
- Visited nodes circled

16-45: **Search Trees**

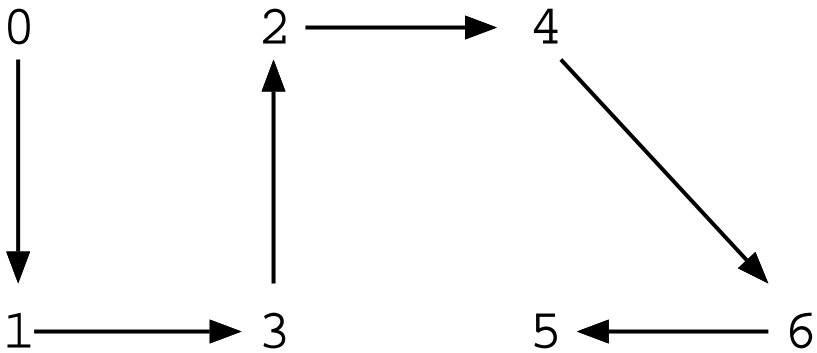
- Describes the order that nodes are examined in a traversal
- Directed Tree
 - Directed edge from v_1 to v_2 if the edge (v_1, v_2) was followed during the traversal

16-46: **DFS Search Trees**

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

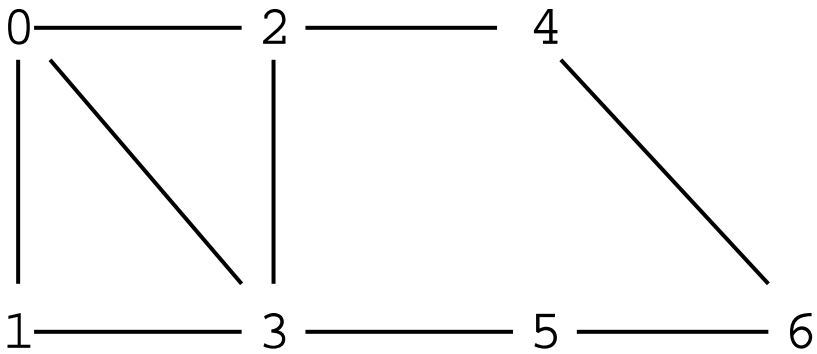
16-47: **DFS Search Trees**

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



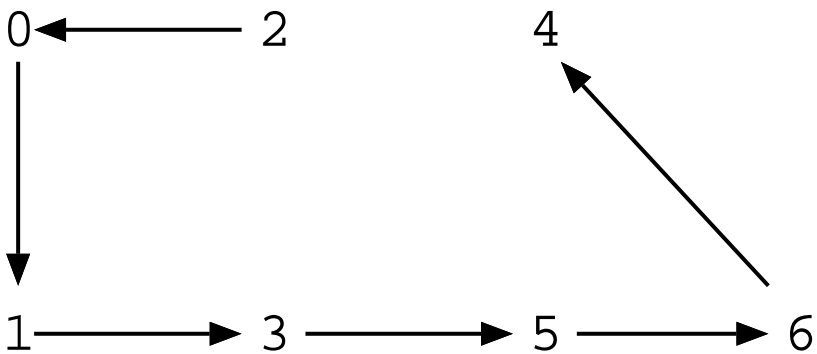
16-48: DFS Search Trees

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



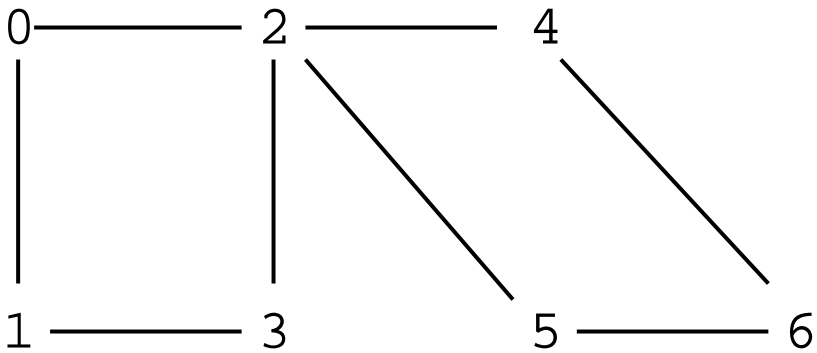
16-49: DFS Search Trees

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



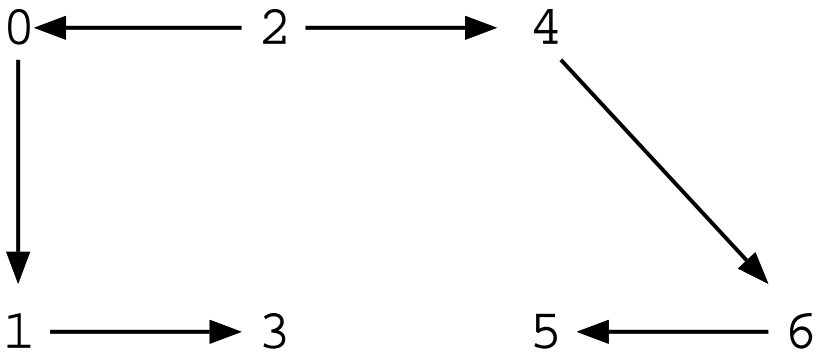
16-50: DFS Search Trees

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



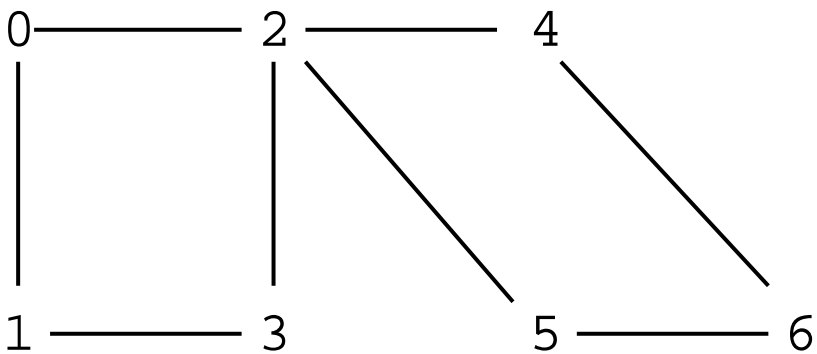
16-51: DFS Search Trees

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



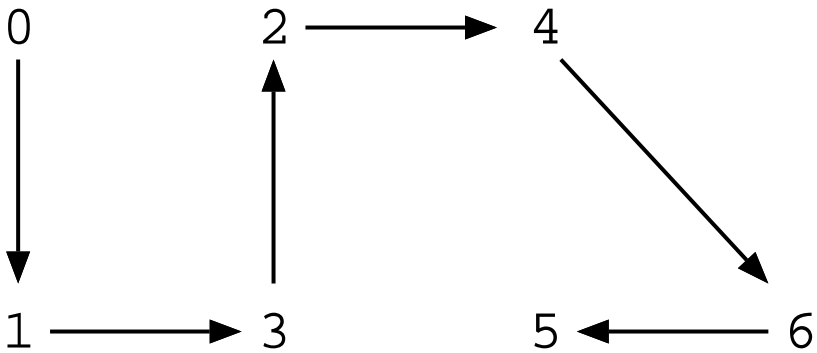
16-52: DFS Search Trees

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



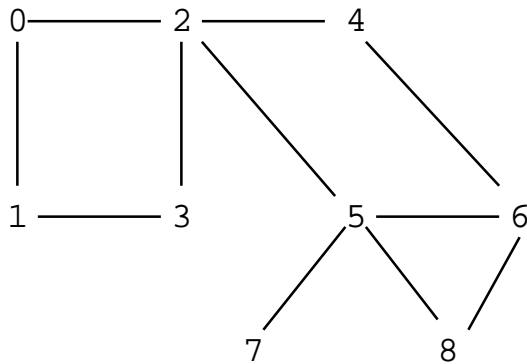
16-53: DFS Search Trees

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



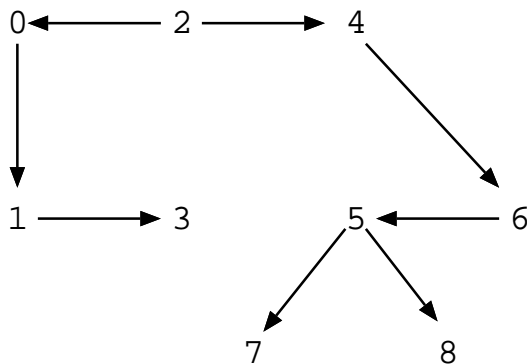
16-54: **DFS Search Trees**

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



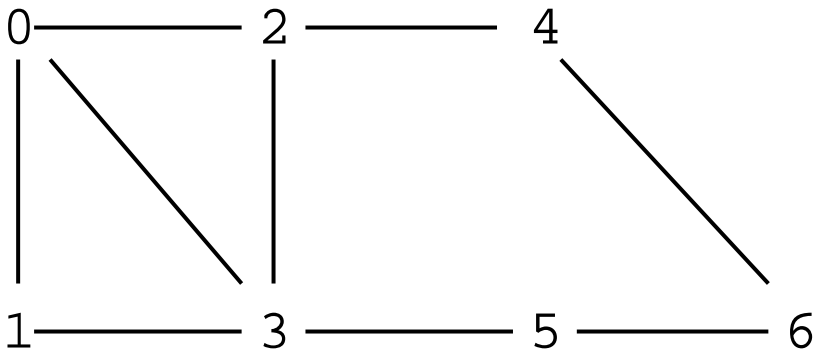
16-55: **DFS Search Trees**

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

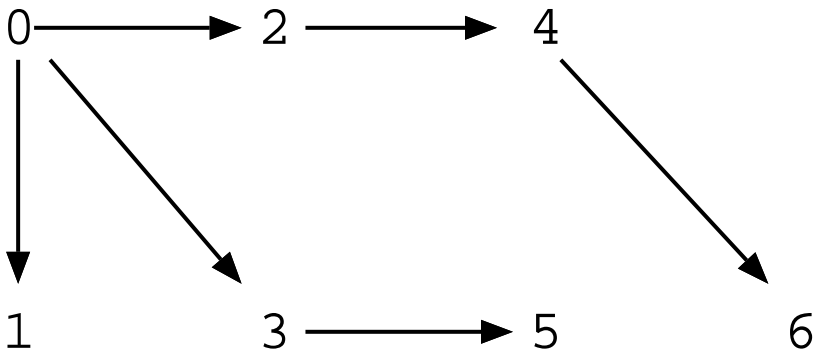


16-56: **BFS Search Trees**

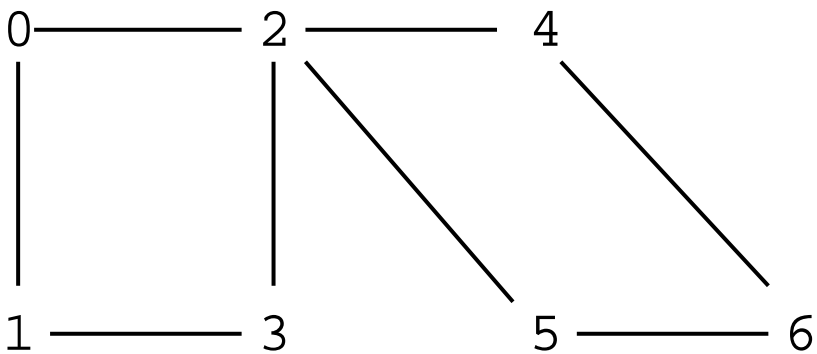
- Starting from node 0, adjacency list sorted by vertex number:

16-57: **BFS Search Trees**

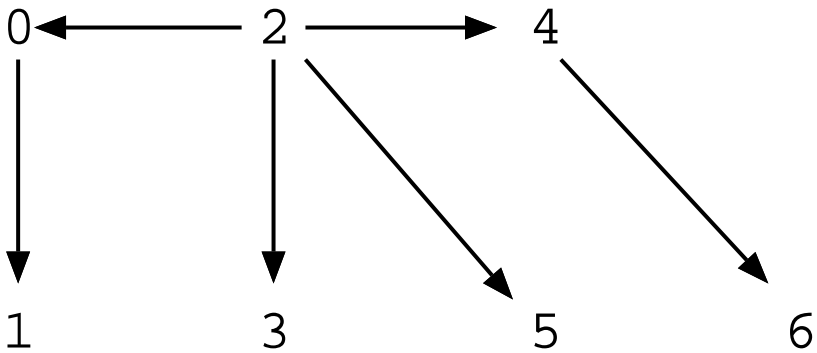
- Starting from node 0, adjacency list sorted by vertex number:

16-58: **BFS Search Trees**

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

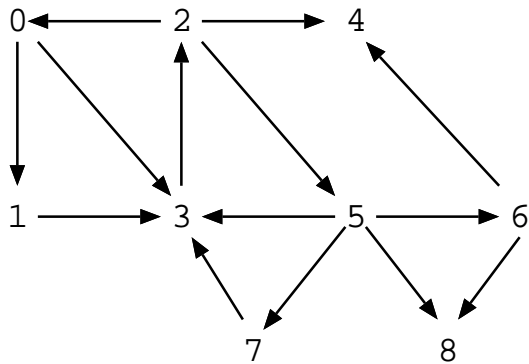
16-59: **BFS Search Trees**

- 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:



16-61: DFS in Directed Graphs

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

