Quiz 2 Logistics

Taken via D2L. You are not timed, but you have only one attempt
Opens 6:00 AM on Thursday, closes 11:59 PM on Thursday

10-15 Questions

- Short answer, multiple choice, true or false

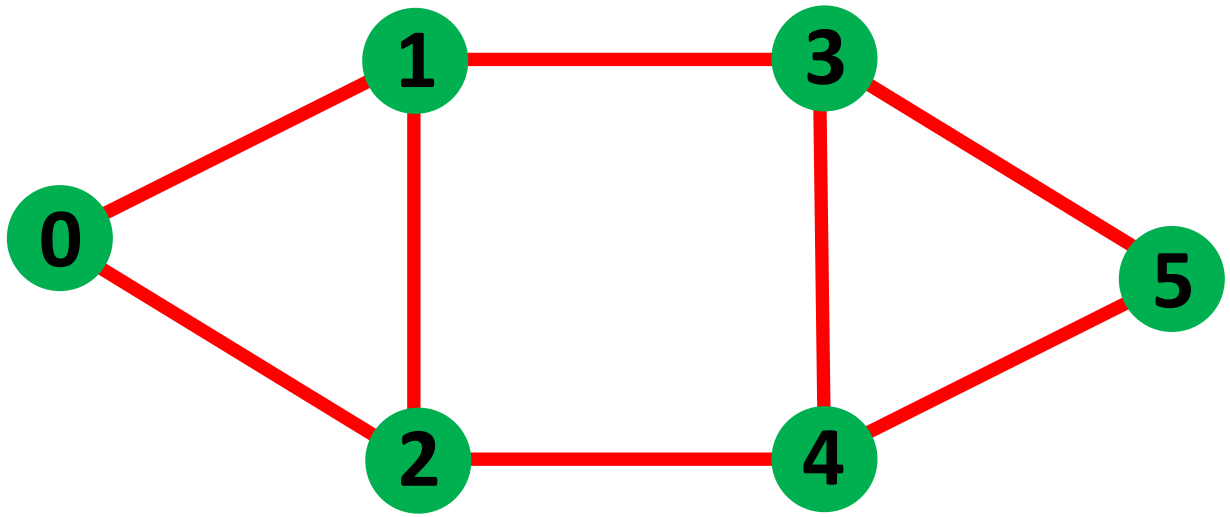
Quiz Content

- Hash Tables
- Hash Functions
- Principles of “good” hash functions and hash tables
- HashMaps and HashSets
- Graphs
- Graph Terminology and Definitions
- Graph Traversal

Graph Searching

CSCI 232

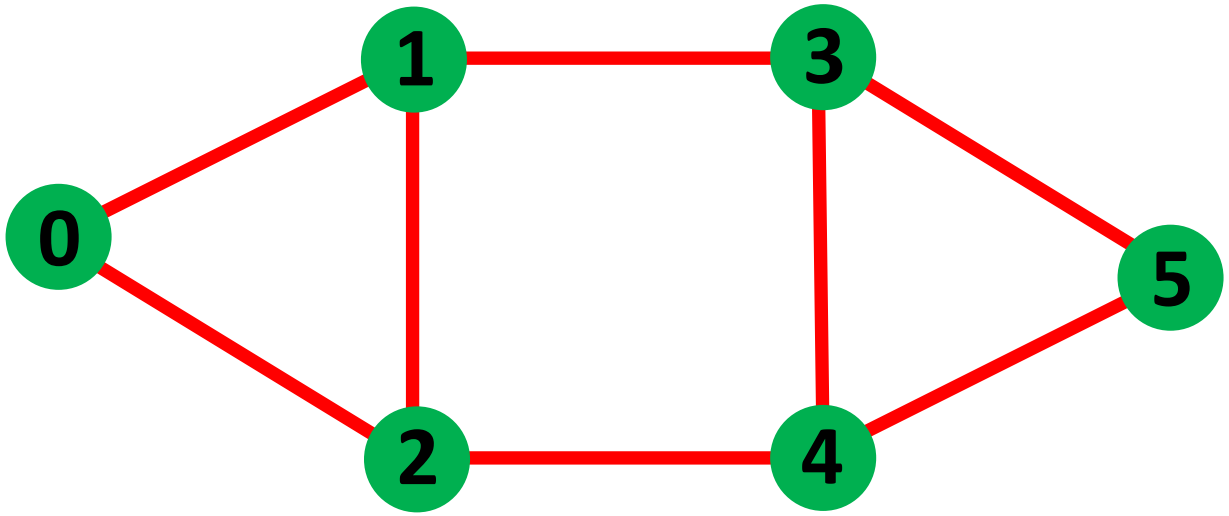
Graphs



Adjacency List

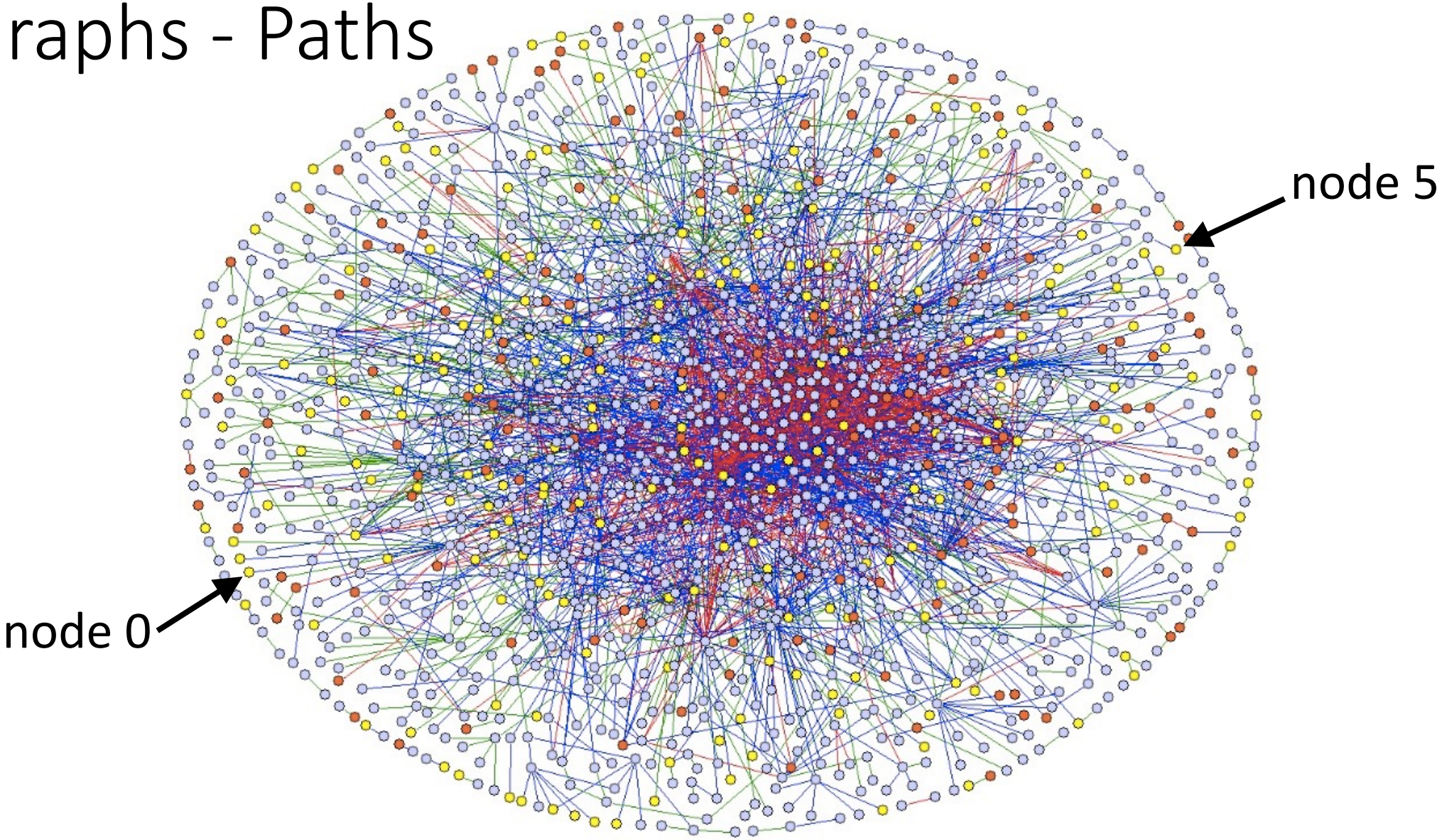
0	→	{1,2}
1	→	{0,2,3}
2	→	{0,1,4}
3	→	{1,4,5}
4	→	{2,3,5}
5	→	{3,4}

Graphs - Paths



Is there a path from node 0 to node 5?

Graphs - Paths



Is there a path from node 0 to node 5?

Graphs - Paths

0	→	**{1,2}**
1	→	**{0,2,3}**
2	→	**{0,1,4}**
3	→	**{1,4,5}**
4	→	**{2,3,5}**
5	→	**{3,4}**

Is there a path from node 0 to node 5?

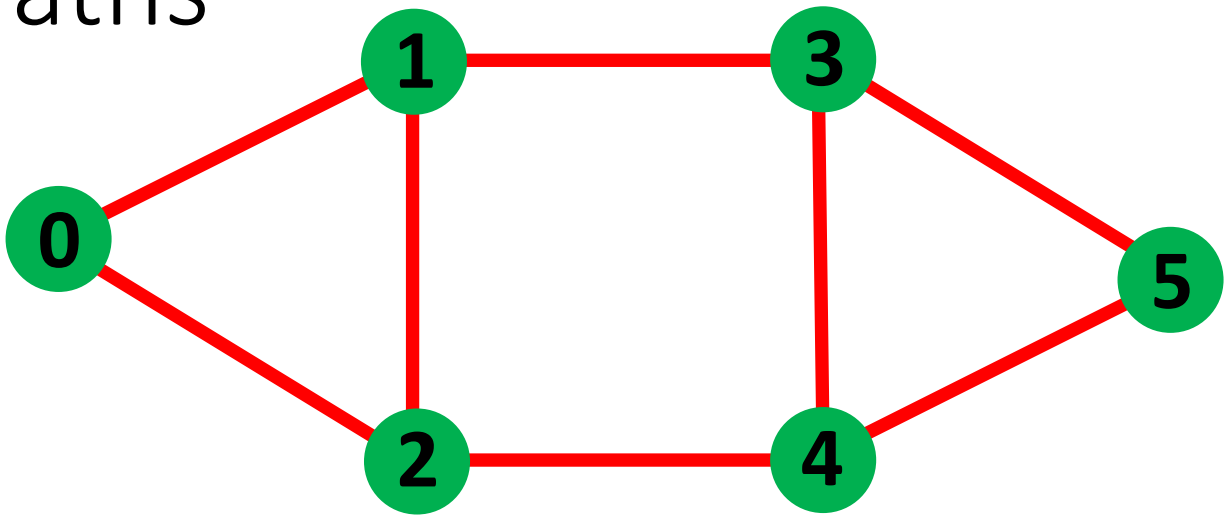
Graphs - Paths

0	→	{1,2}
1	→	{0,2,3}
2	→	{0,1,4}
3	→	{1,4,5}
4	→	{2,3,5}
5	→	{3,4}

We need a better process than "eyeballing it" for finding paths.

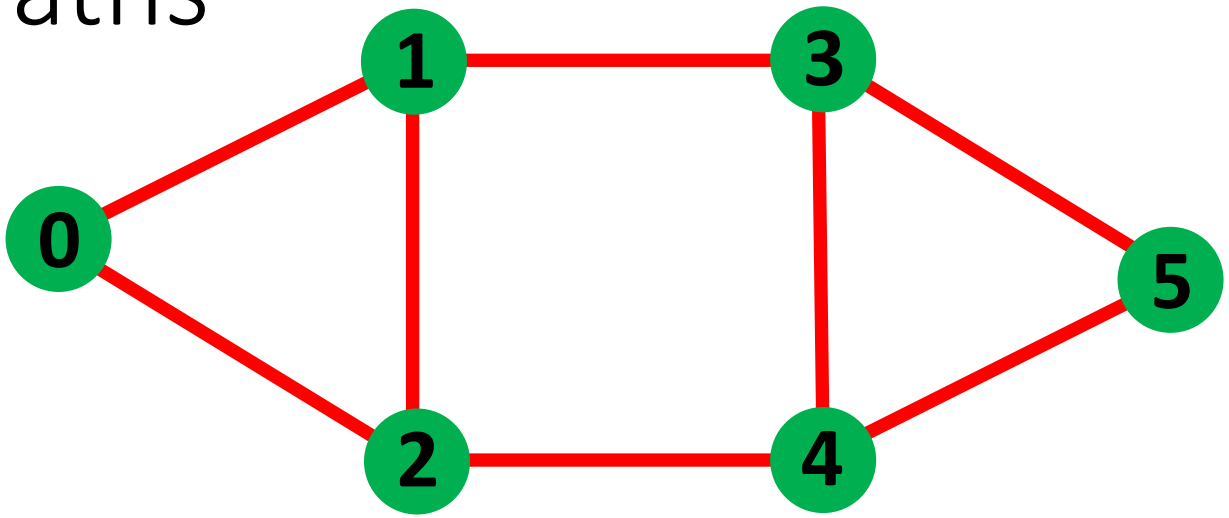
Is there a path from node 0 to node 5?

Graphs - Paths



What is a **generalizable** process to see if there is a path from node 0 to node 5?

Graphs - Paths

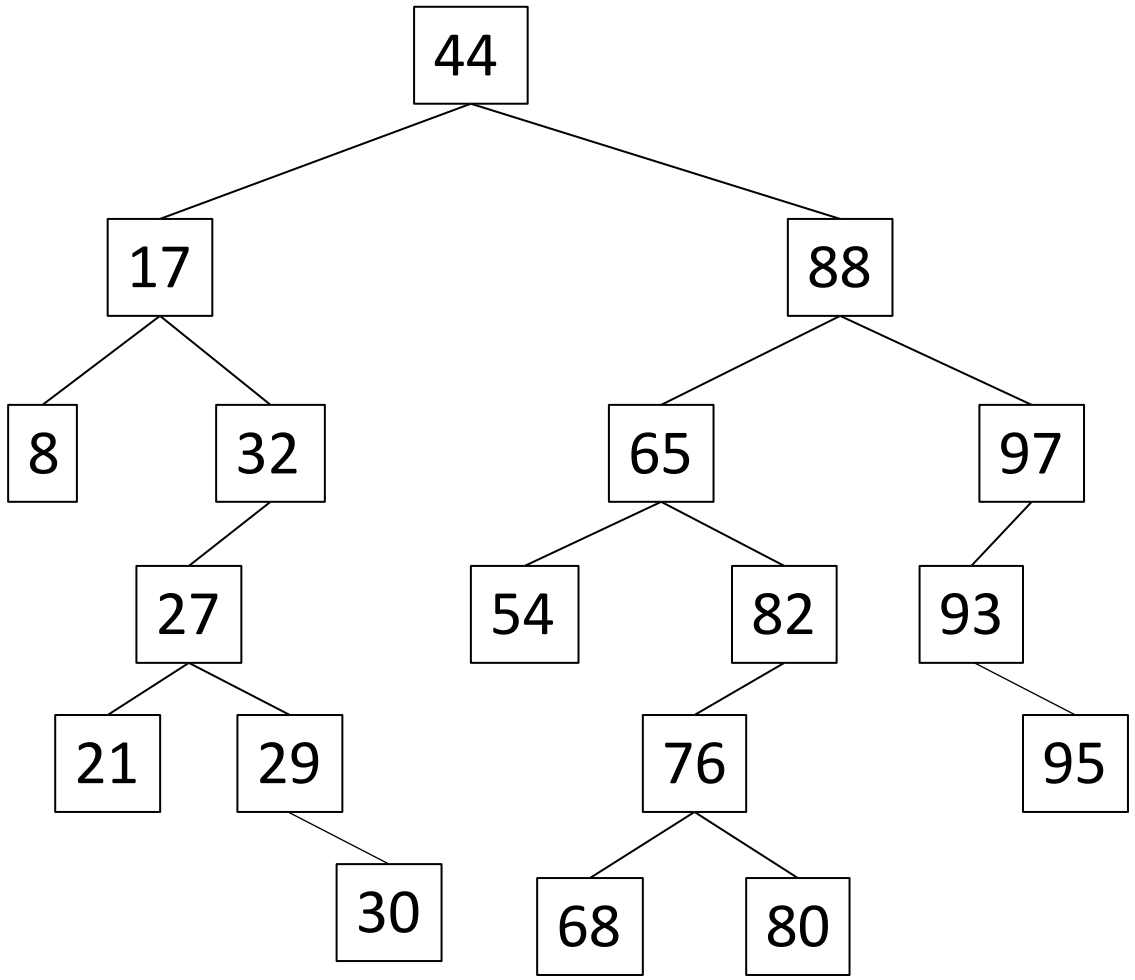


What is a **generalizable** process to see if there is a path from node 0 to node 5?

Start at node 0.
Go to each neighbor.
Check each neighbor's neighbor.
Check each neighbor's neighbor's neighbor….

Binary Search Tree - Traversal

```
public void depthFirst(Node n) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

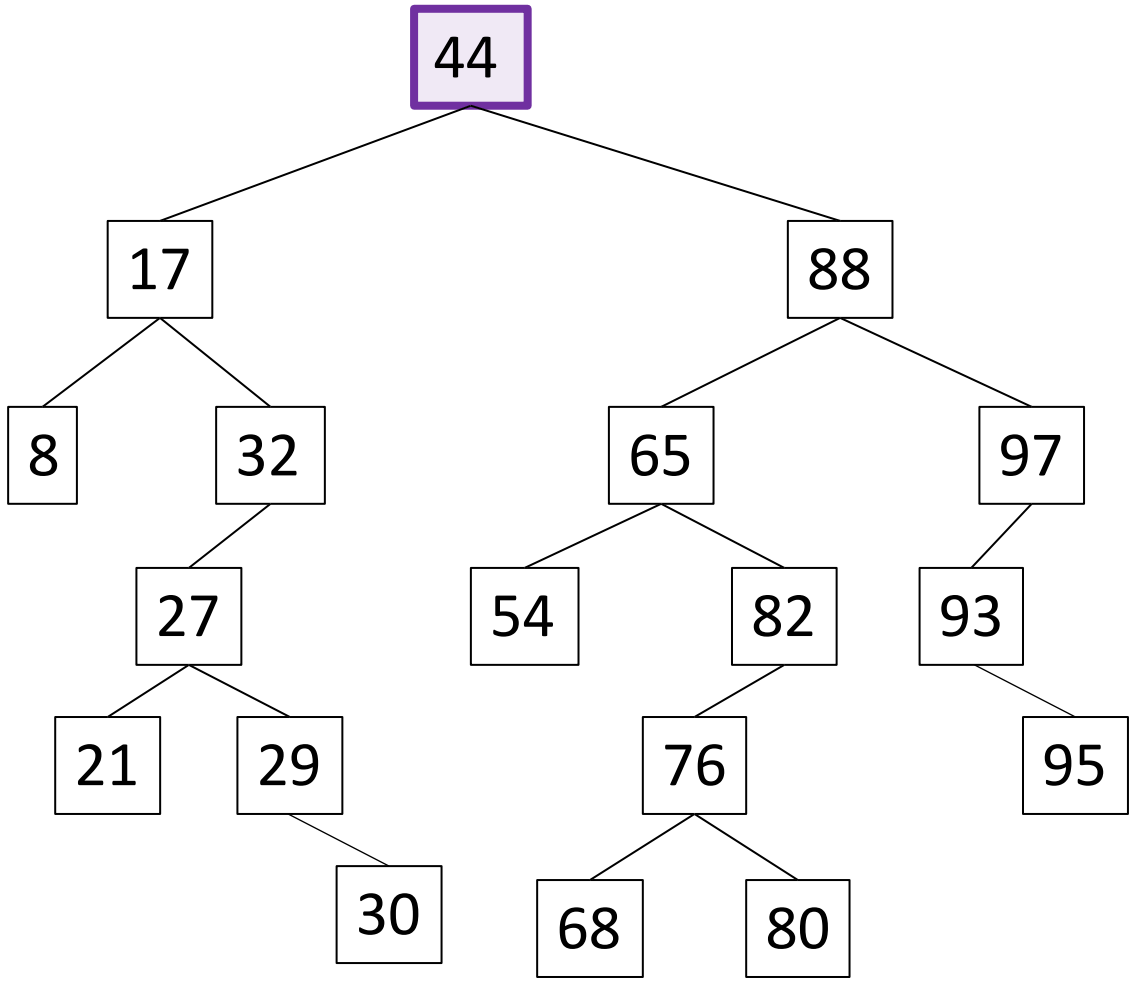


Recursion:

- **Calling a method from inside itself.**
- **Solve the problem by solving identical smaller problems.**
- **What is the "smaller problem"?**
  - **Process the left side, then process the right side.**

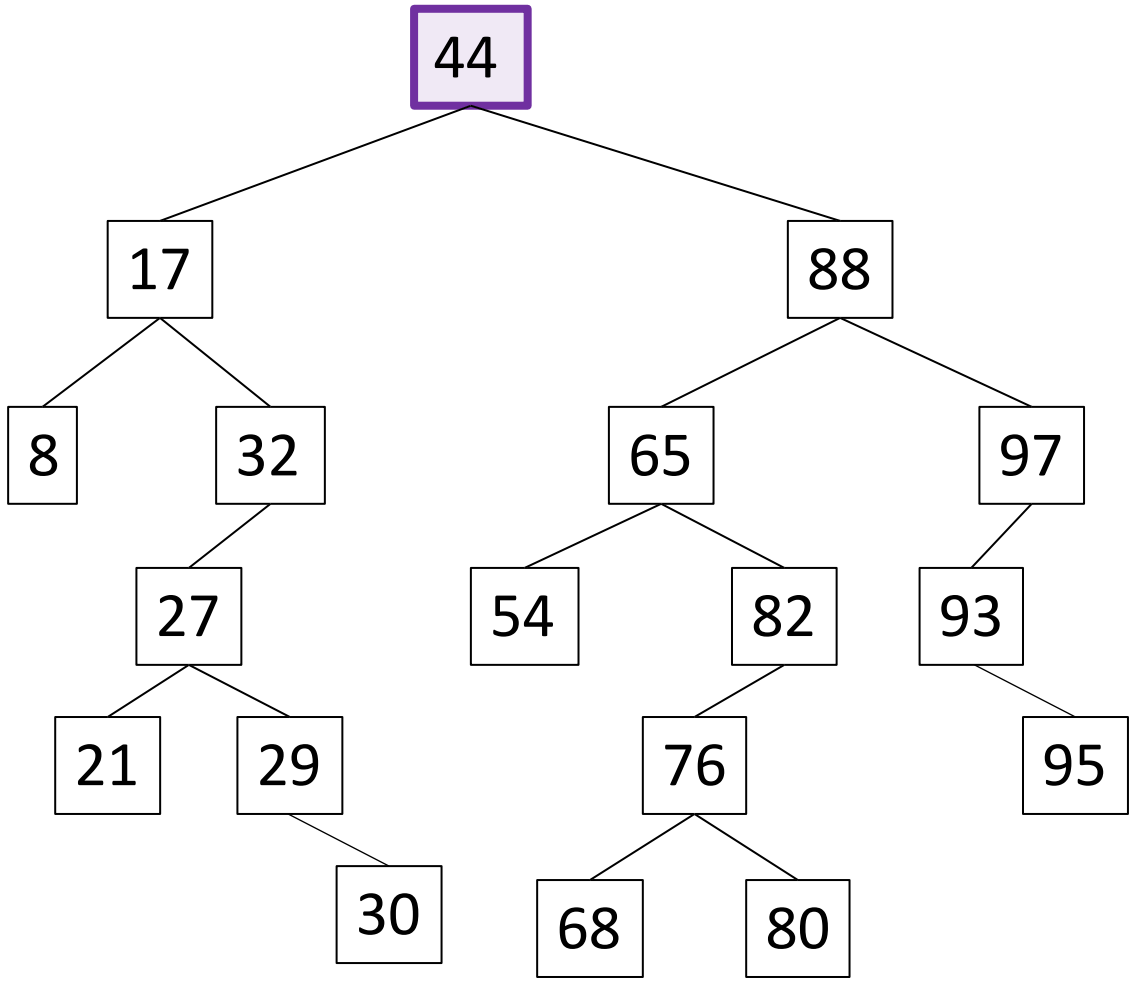
Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Binary Search Tree - Traversal

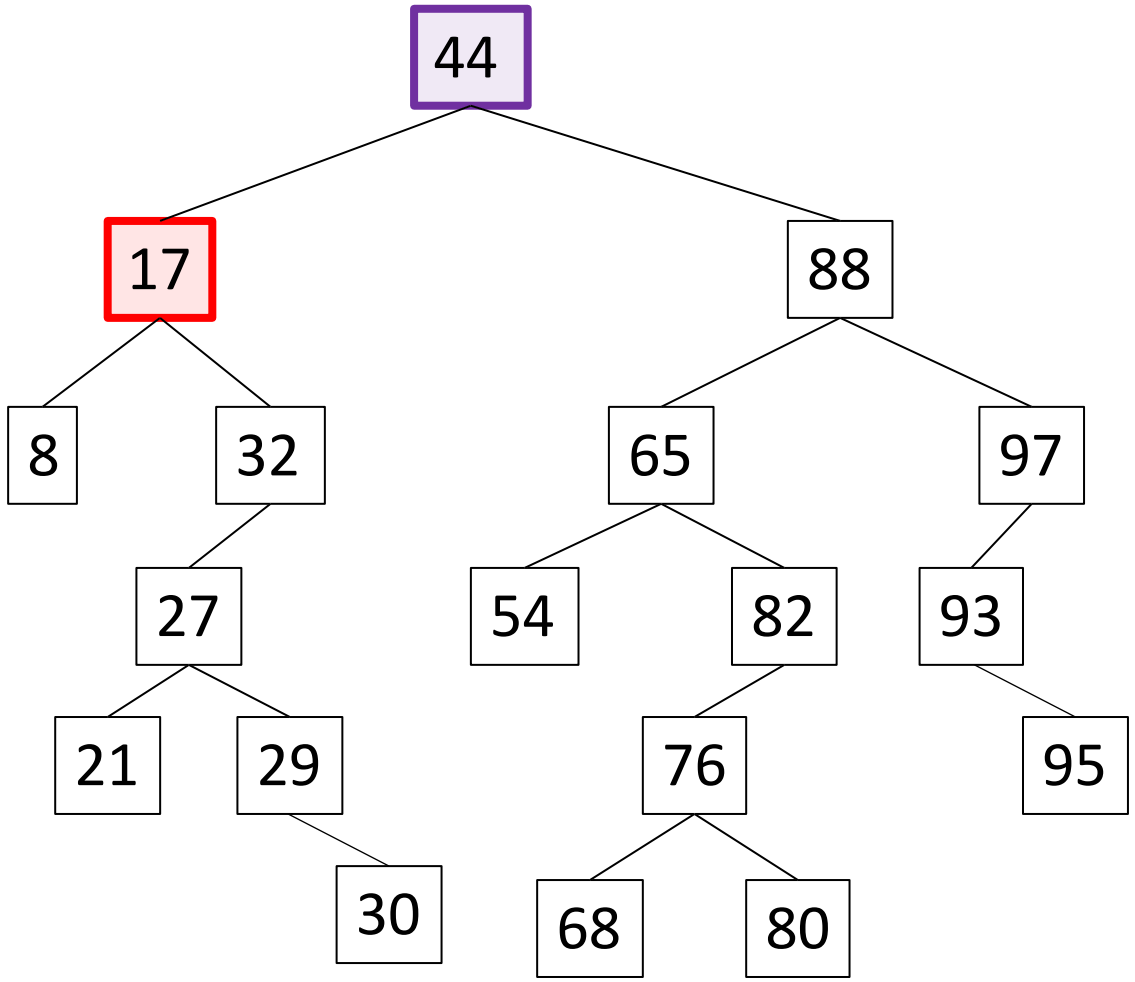
```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

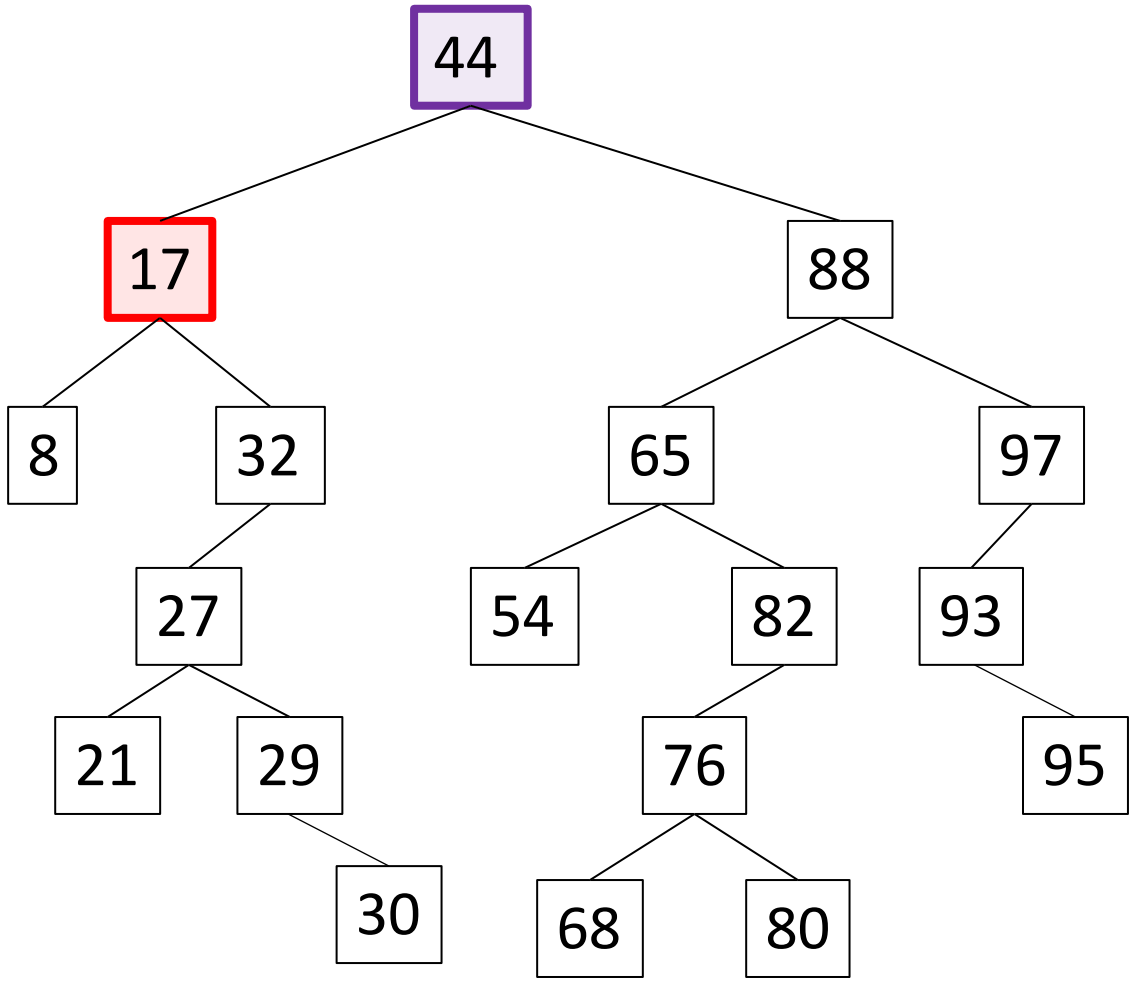
```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

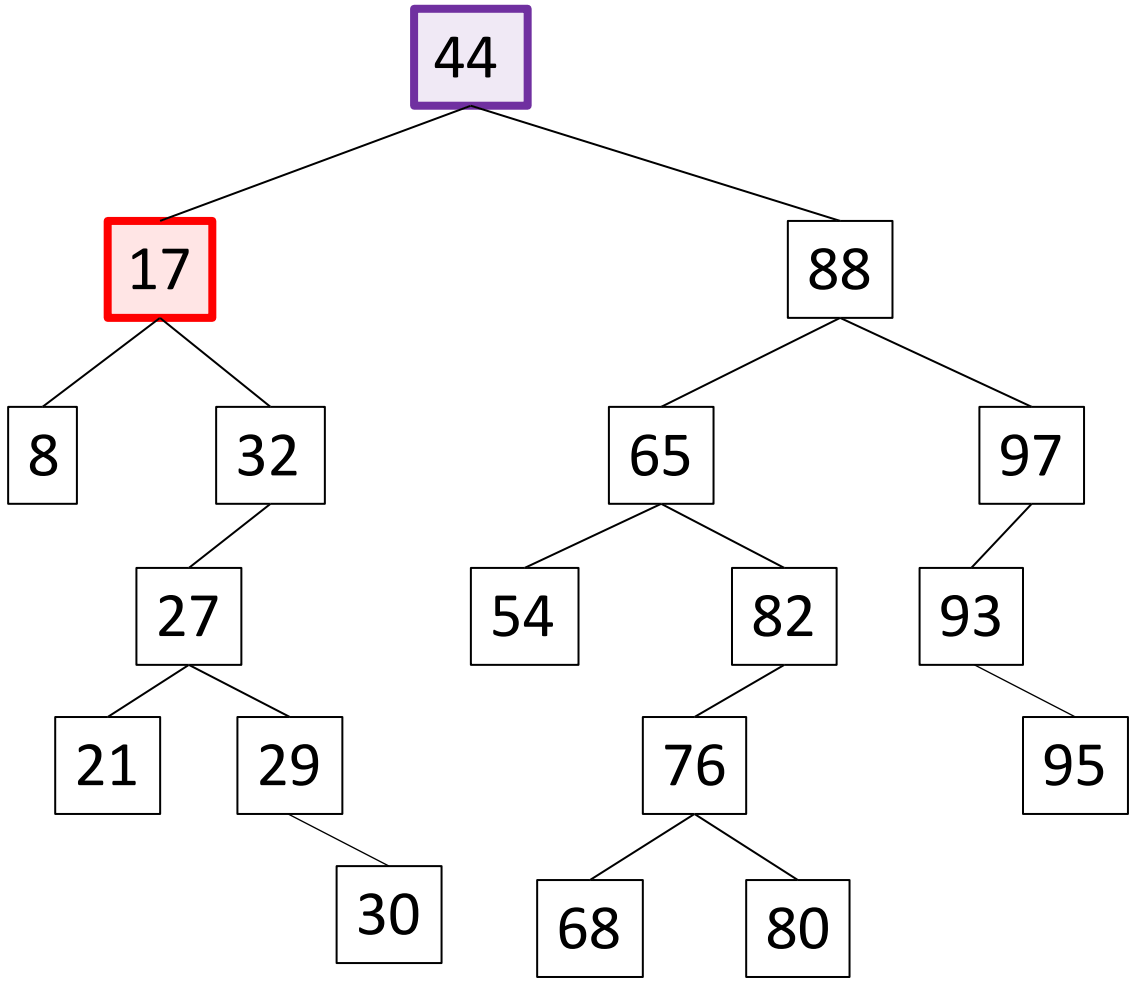
```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

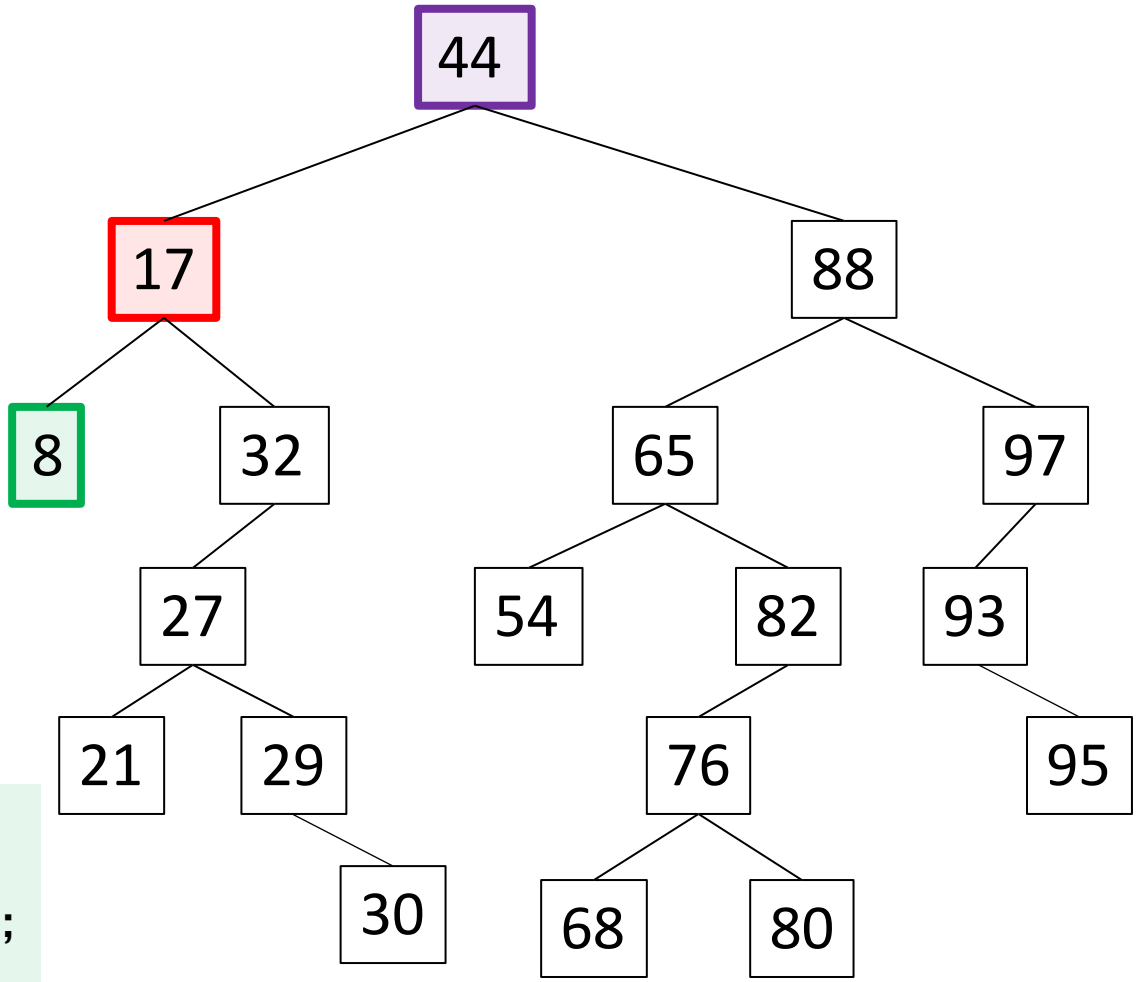


Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

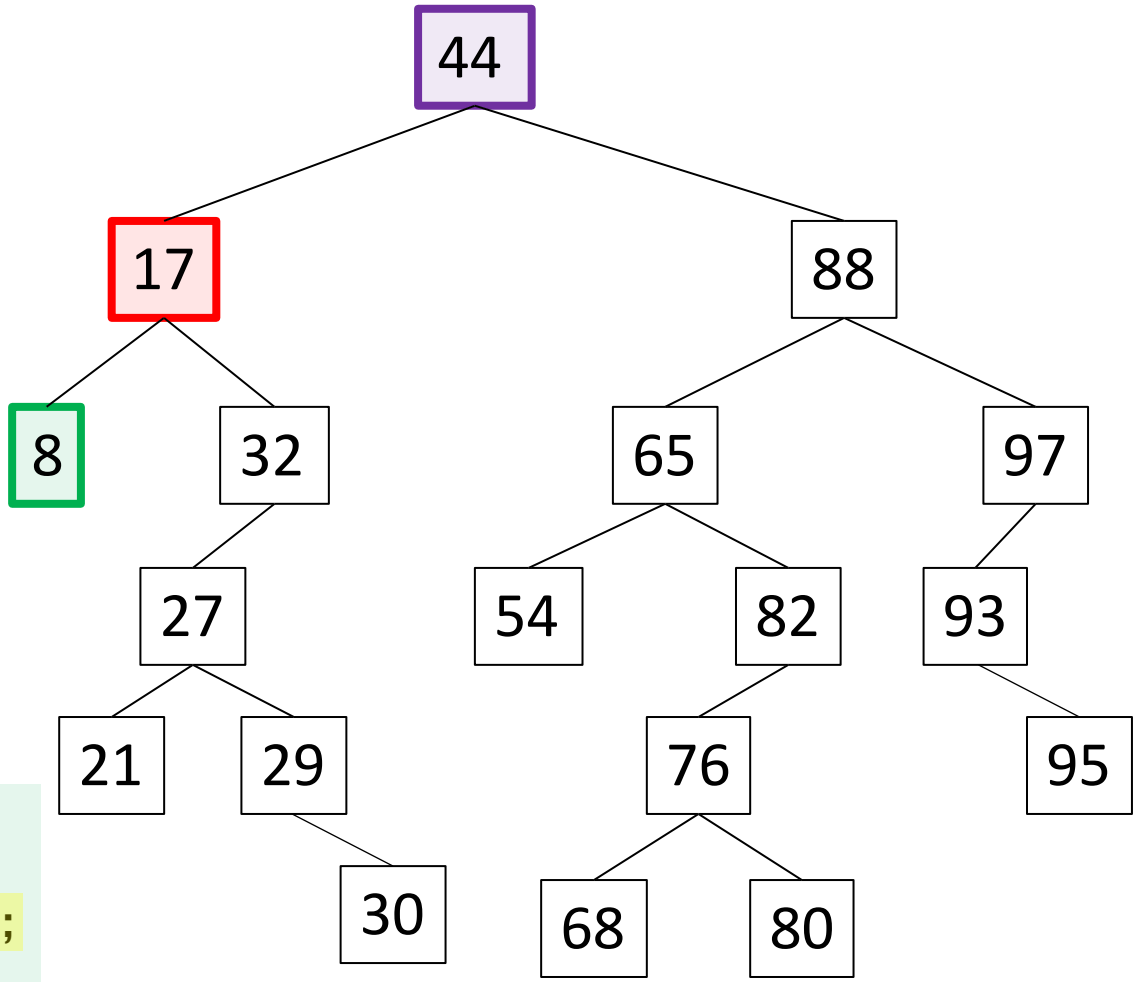


Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

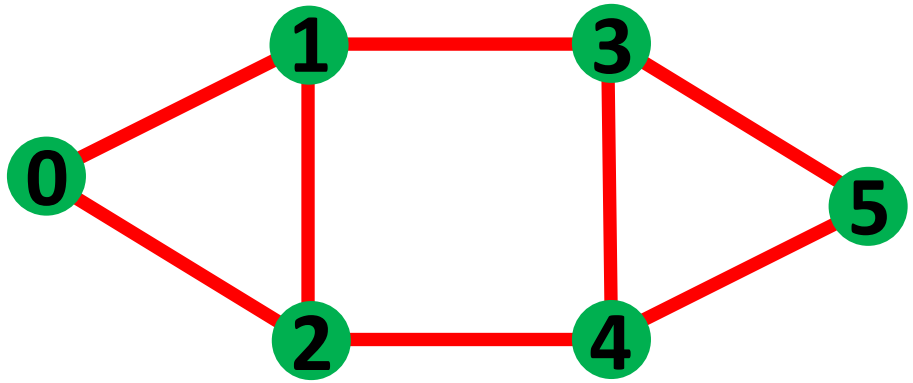
```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



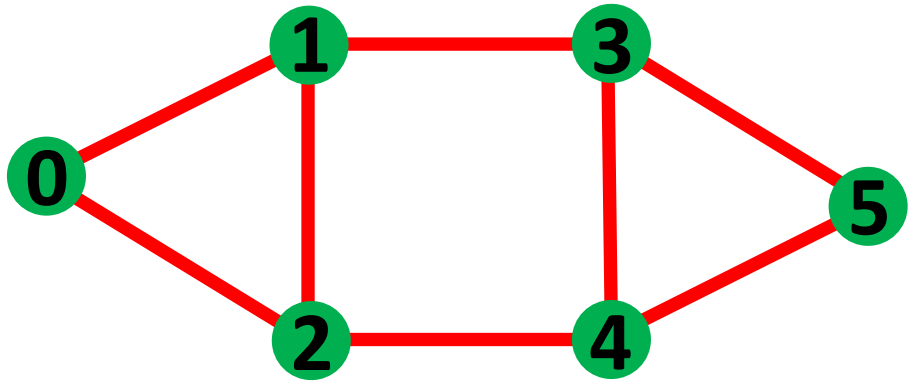
Graphs - Traversal

```
public void depthFirst(Node n) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



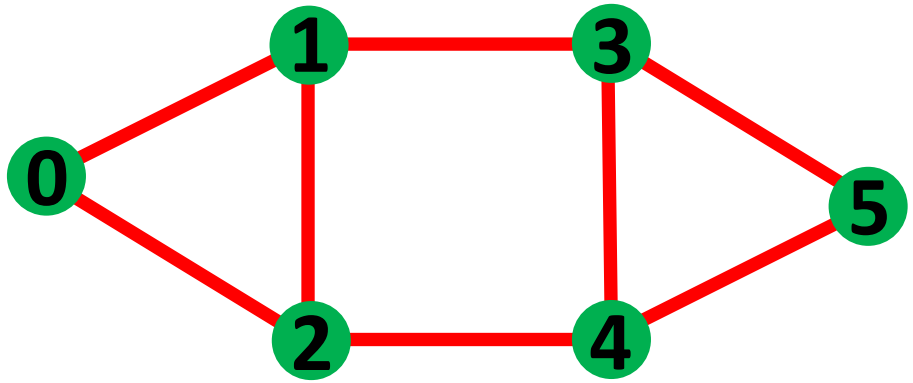
Graphs - Traversal

```
public void depthFirst(Node n) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



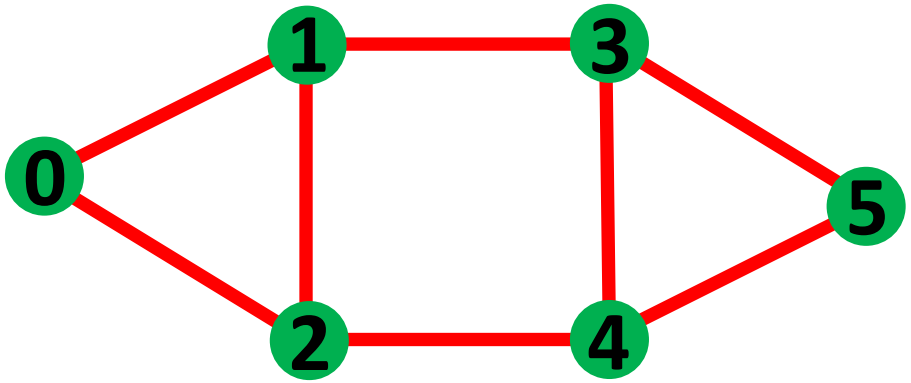
Graphs - Traversal

```
                        int
public void depthFirst(~~Node~~ n) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



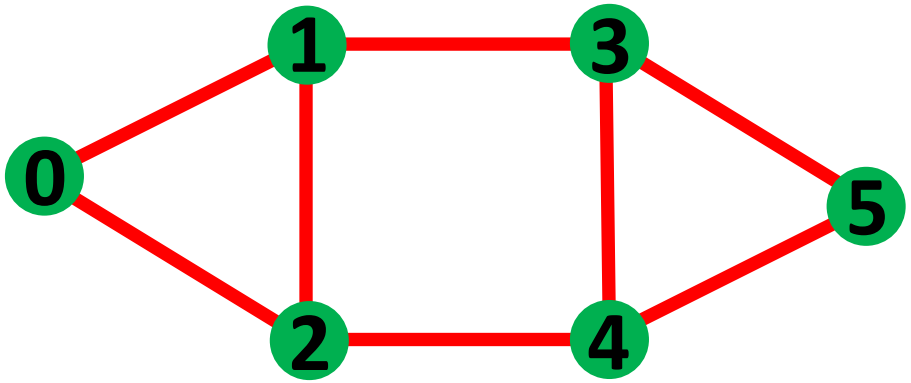
Graphs - Traversal

```
                        int
public void depthFirst(Node n) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



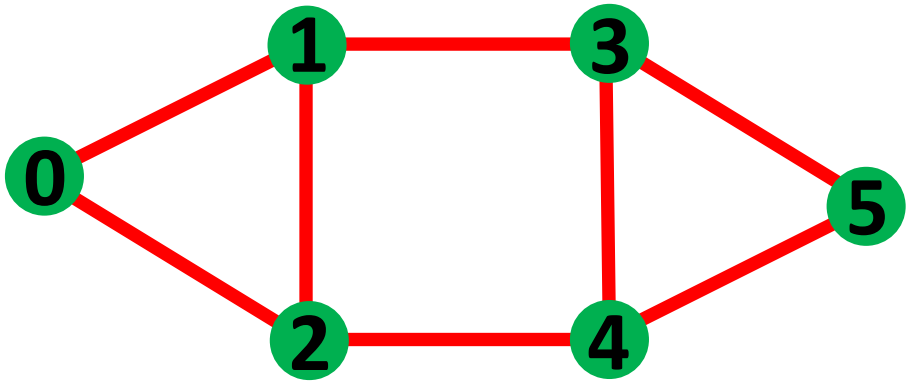
Graphs - Traversal

```
                                int
public void depthFirst(Node n) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



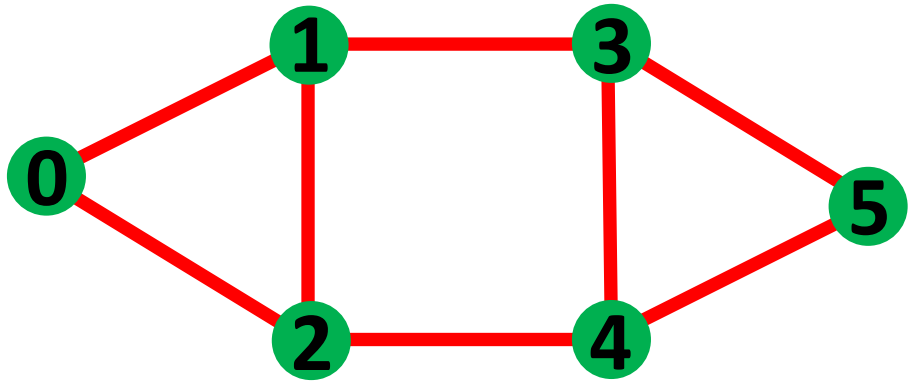
Graphs - Traversal

```
                                int
public void depthFirst(Node n) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



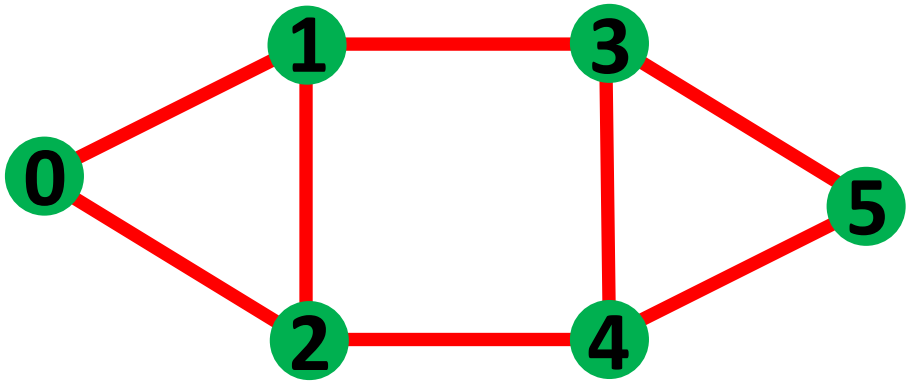
Graphs - Traversal

```
                                int
public void depthFirst(~~Node~~ n) {
    ~~if (n != null) {~~
        System.out.println(n~~.getValue()~~);
        ~~depthFirst(n.getLeft());~~
        ~~depthFirst(n.getRight());~~
    ~~}~~
}
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
```



Graphs - Traversal

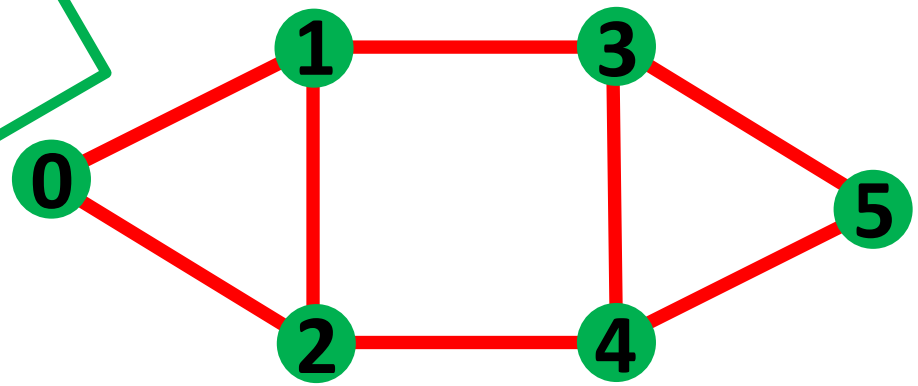
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Graphs - Traversal

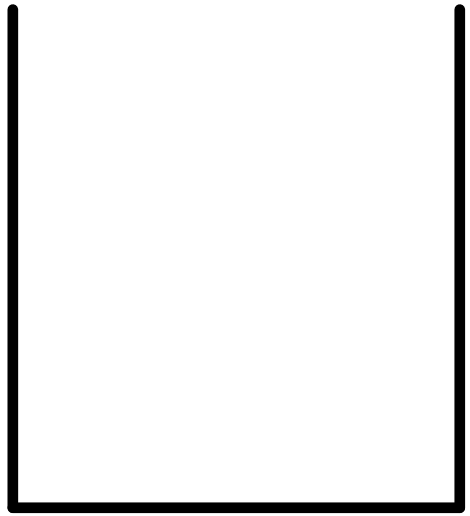
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)
        depthFirst(neighbor);
    }
}
```

CODE



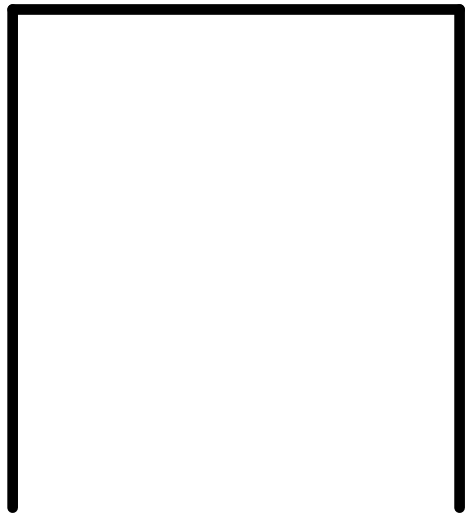
Graphs - Traversal

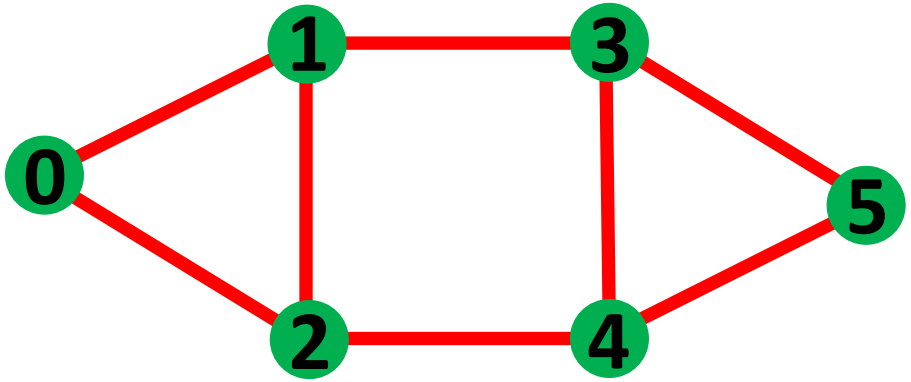
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Run-time Stack

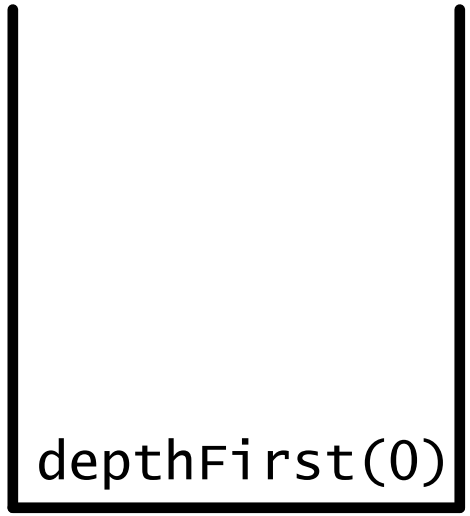
Output





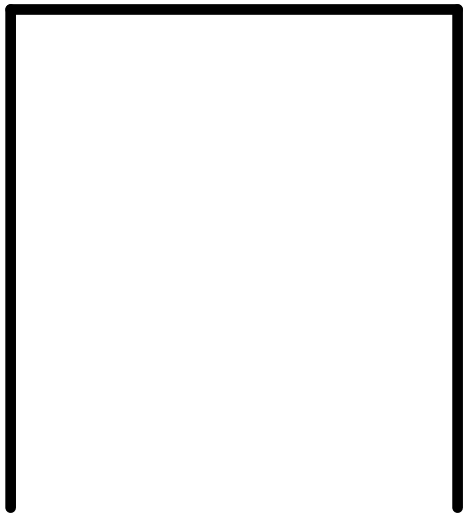
Graphs - Traversal

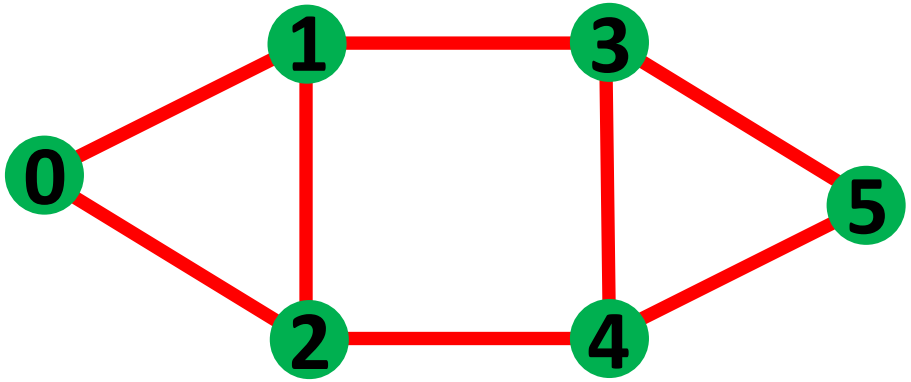
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Run-time Stack

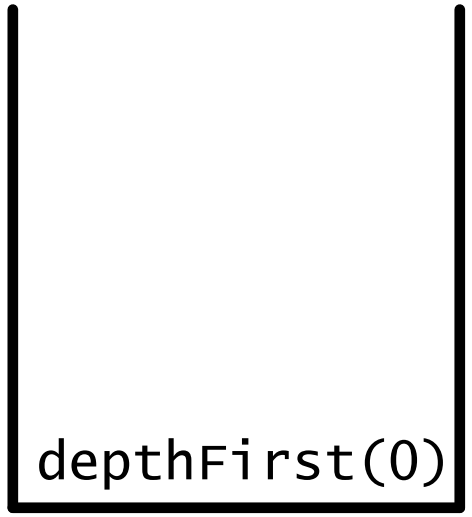
Output





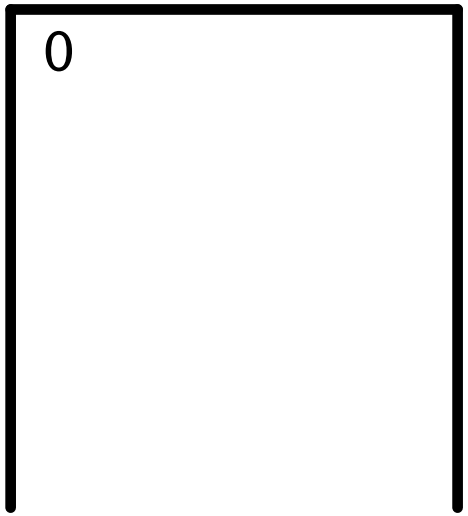
Graphs - Traversal

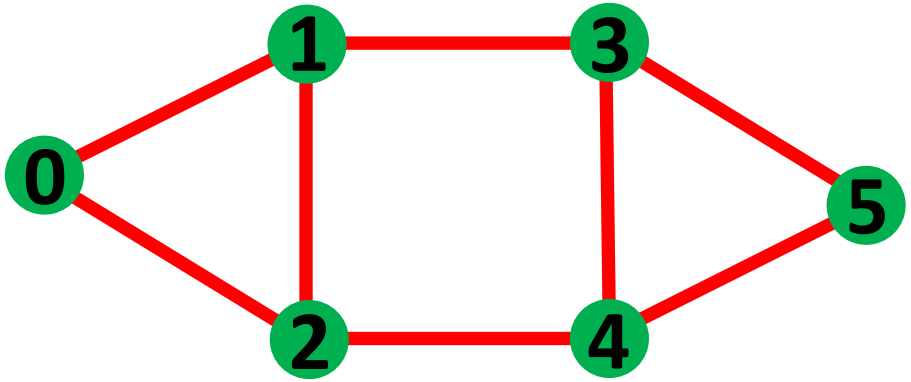
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Run-time Stack

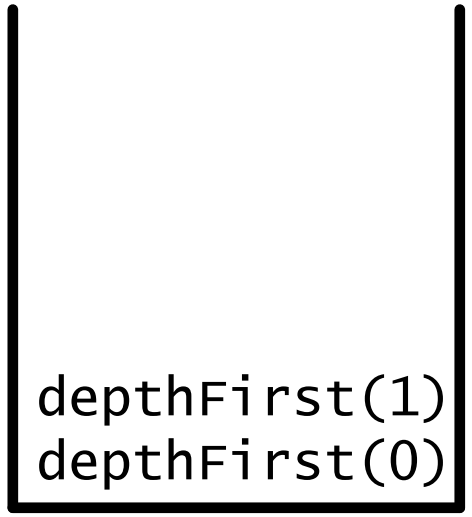
Output





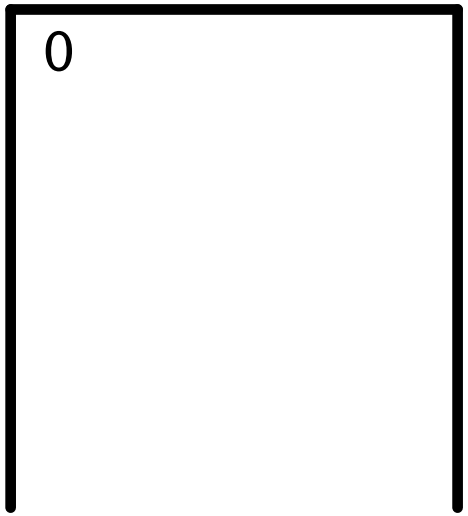
Graphs - Traversal

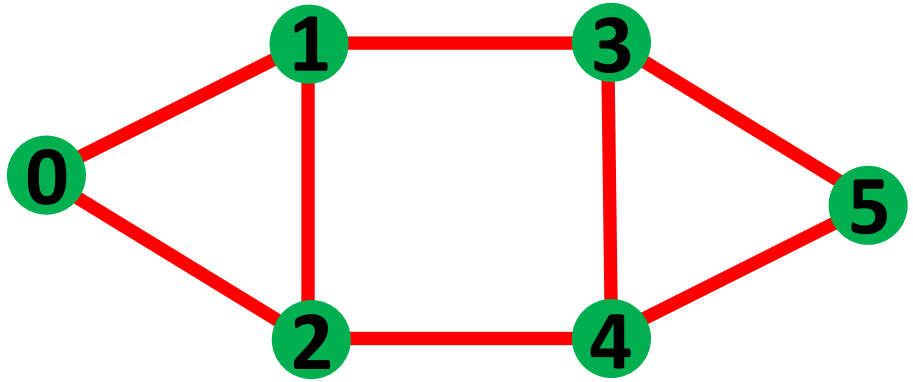
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Run-time Stack

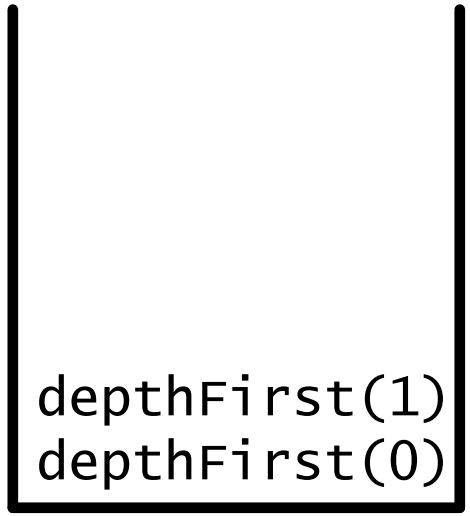
Output





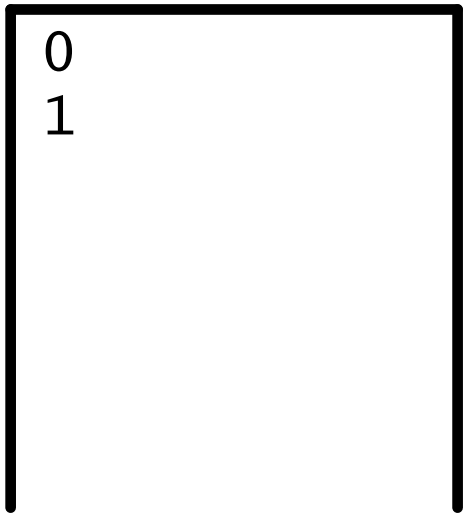
Graphs - Traversal

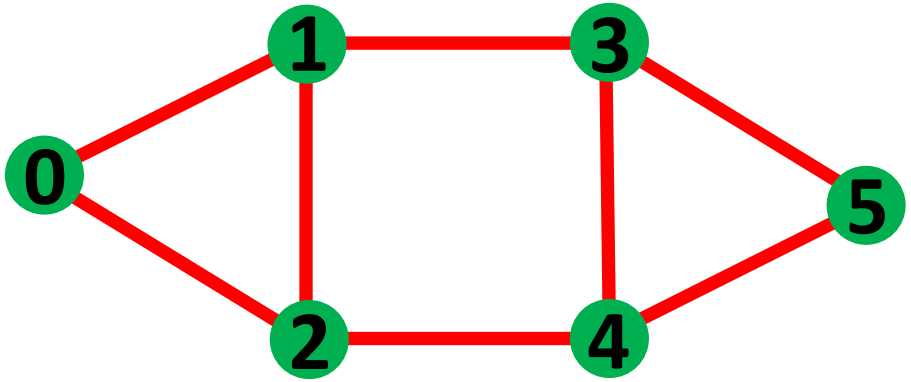
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Run-time Stack

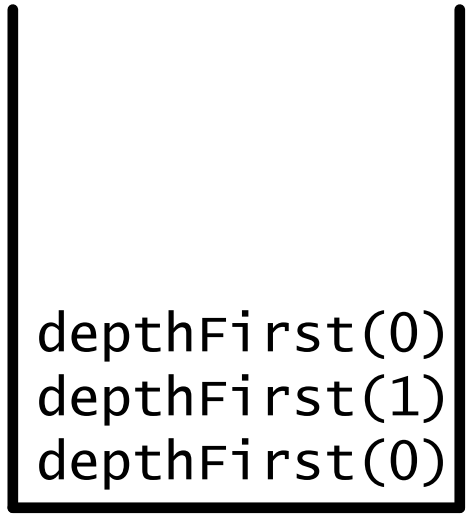
Output





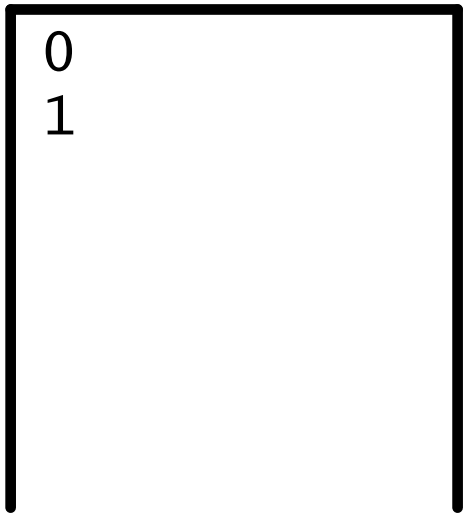
Graphs - Traversal

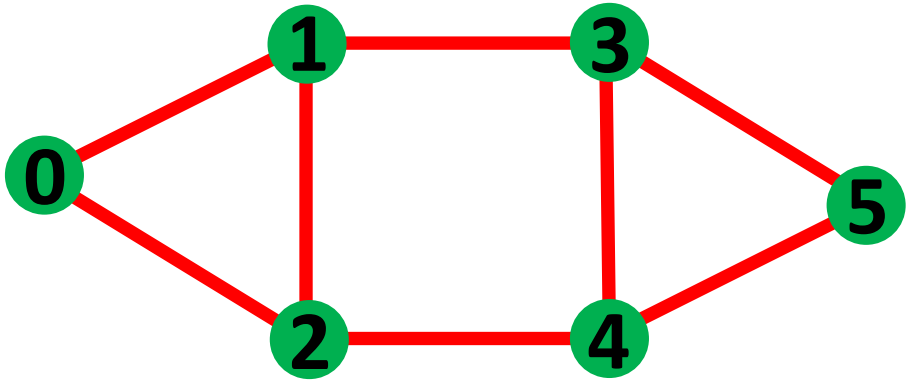
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Run-time Stack

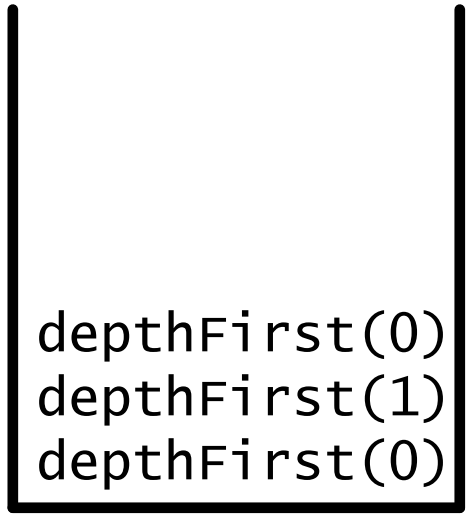
Output





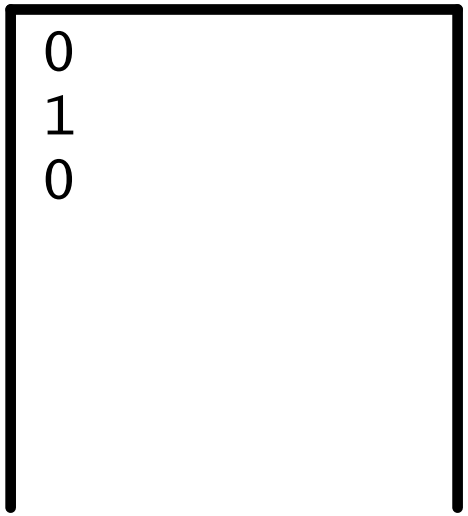
Graphs - Traversal

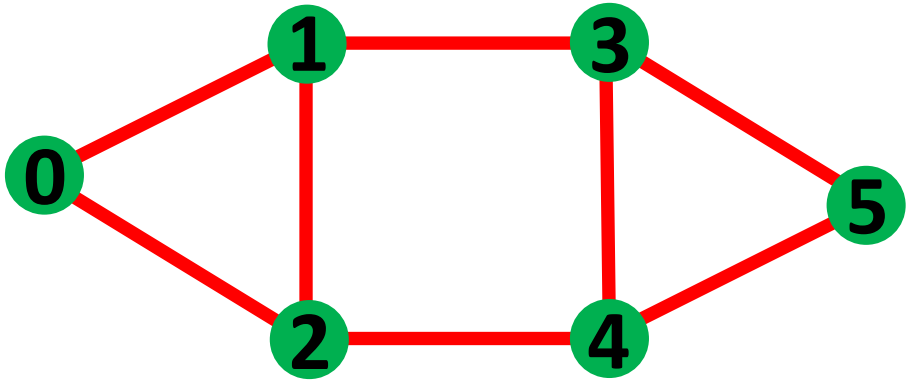
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Run-time Stack

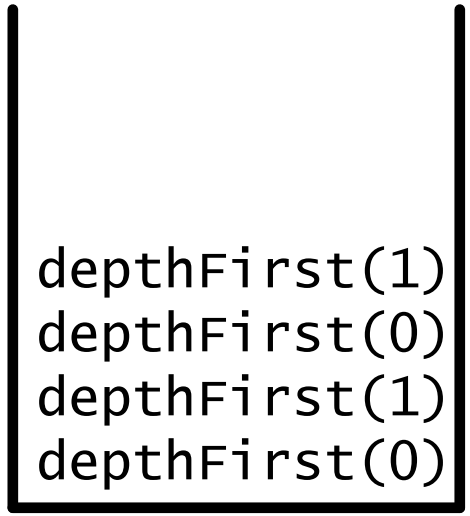
Output





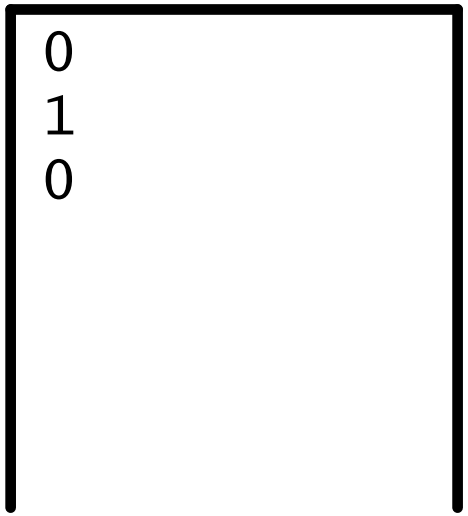
Graphs - Traversal

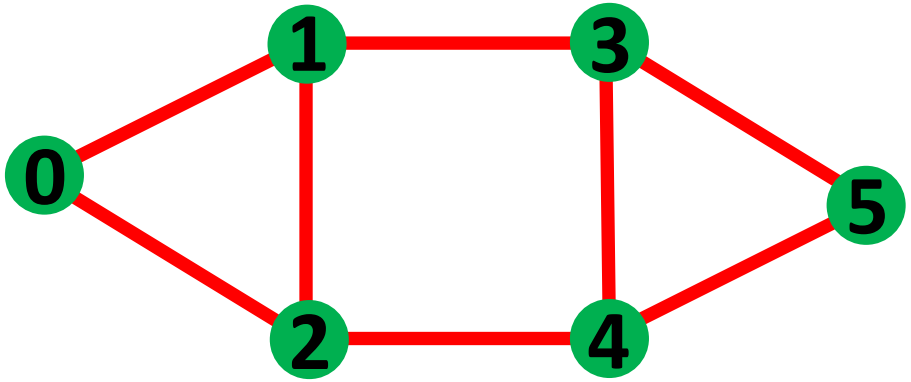
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Run-time Stack

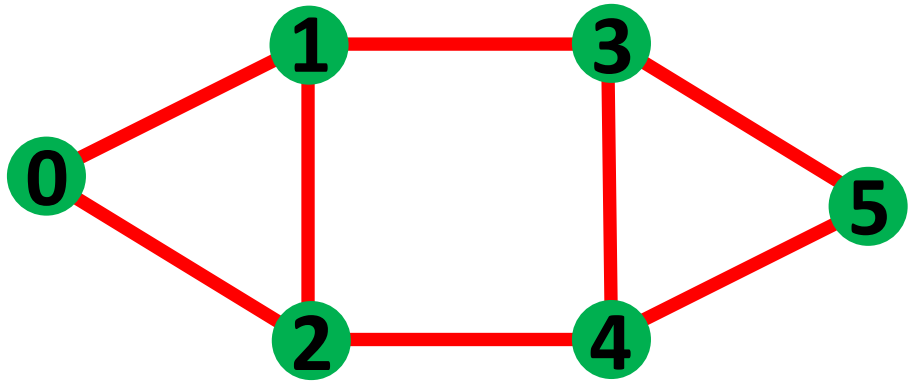
Output





Graphs - Traversal

```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Output

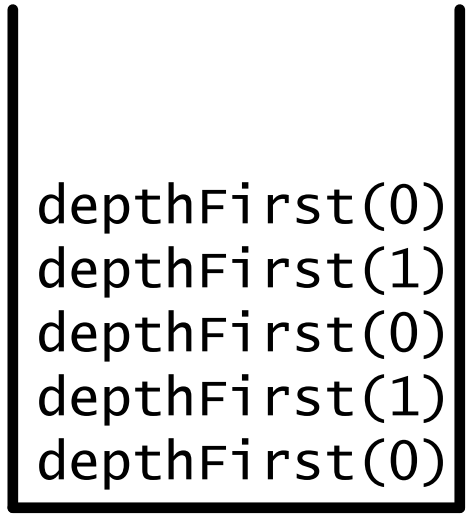
```
0
1
0
1
```

```
depthFirst(1)
depthFirst(0)
depthFirst(1)
depthFirst(0)
```

Run-time Stack

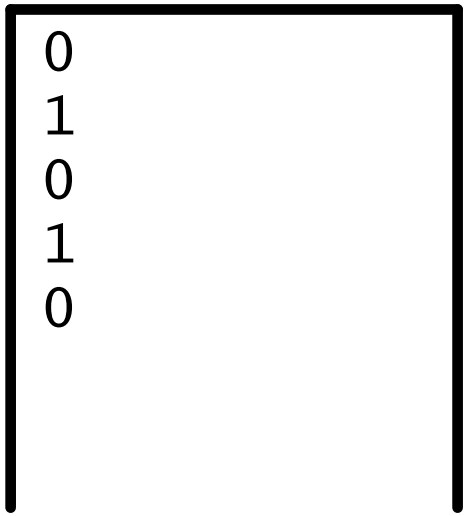
Graphs - Traversal

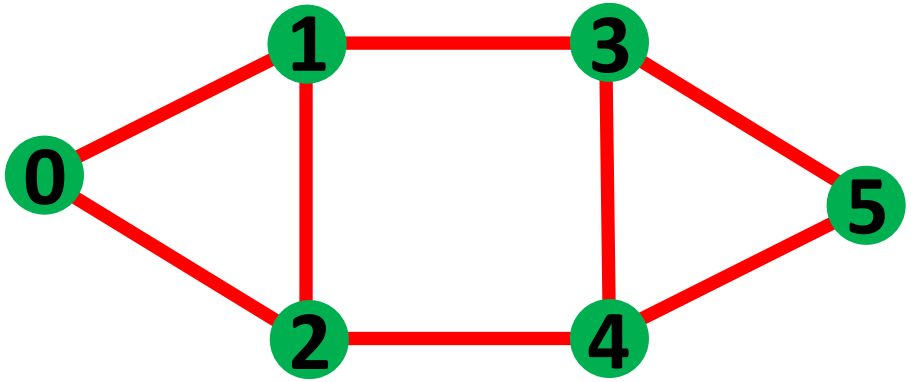
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Run-time Stack

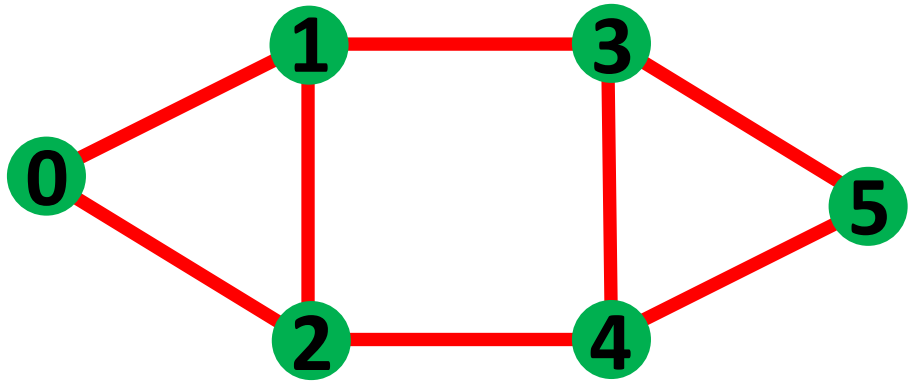
Output





Graphs - Traversal

```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Output

```
0
1
0
1
0
1
```

```
depthFirst(1)
depthFirst(0)
depthFirst(1)
depthFirst(0)
depthFirst(1)
depthFirst(0)
```

Run-time Stack

Graphs - Traversal

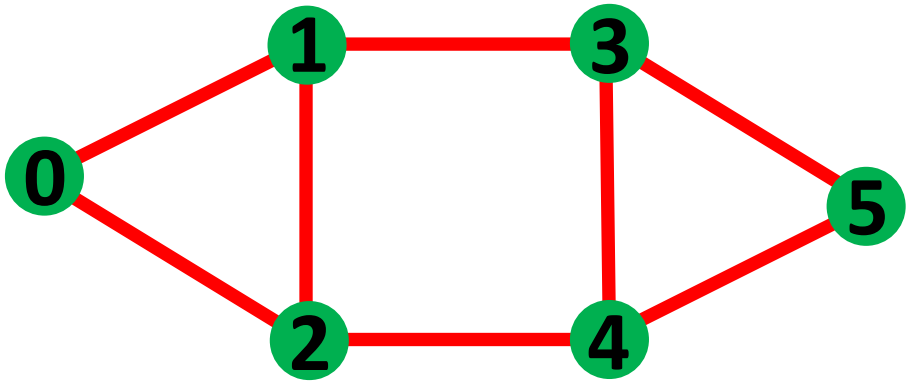
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```

```
depthFirst(0)
depthFirst(1)
depthFirst(0)
depthFirst(1)
depthFirst(0)
depthFirst(1)
depthFirst(0)
```

Run-time Stack

Output

```
0
1
0
1
0
1
0
```



Graphs - Traversal

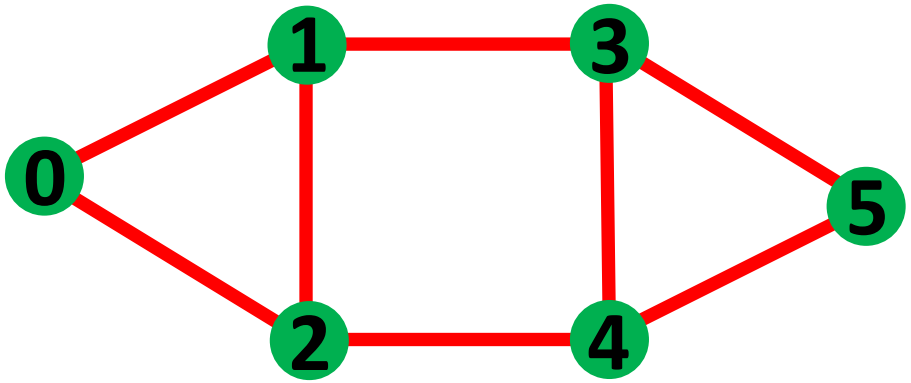
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```

```
depthFirst(1)
depthFirst(0)
depthFirst(1)
depthFirst(0)
depthFirst(1)
depthFirst(0)
depthFirst(1)
depthFirst(0)
```

Run-time Stack

Output

```
0
1
0
1
0
1
0
1
```



Graphs - Traversal

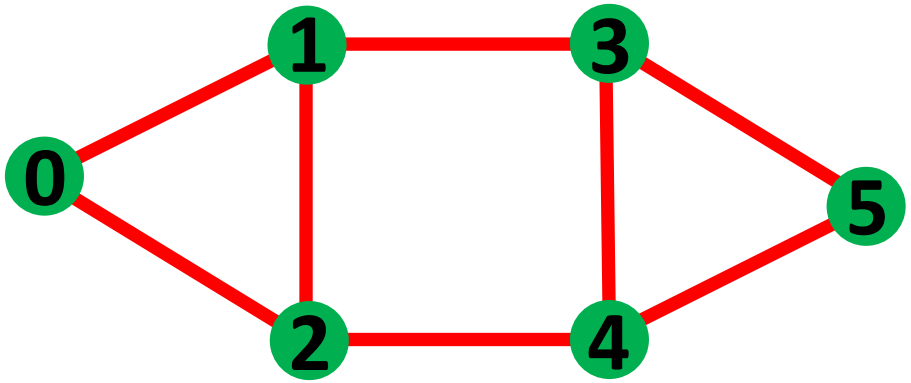
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```

```
depthFirst(0)
depthFirst(1)
depthFirst(0)
depthFirst(1)
depthFirst(0)
depthFirst(1)
depthFirst(0)
depthFirst(1)
depthFirst(0)
```

Run-time Stack

Output

```
0
1
0
1
0
1
0
1
0
```



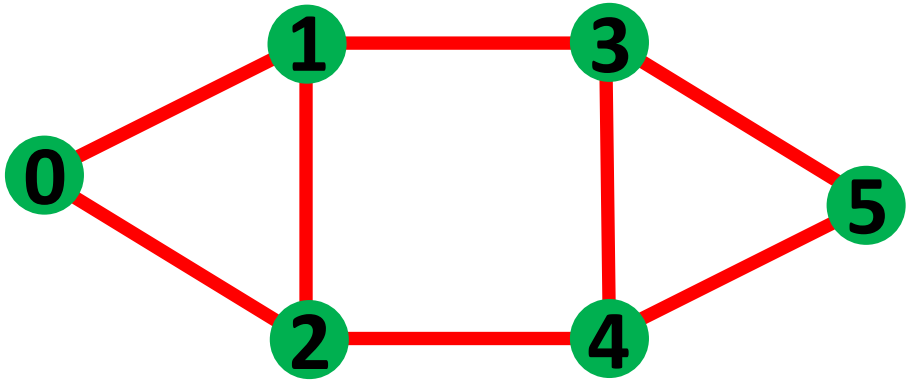
Graphs - Traversal

```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```

```
depthFirst(0)
depthFirst(1)
depthFirst(0)
depthFirst(1)
depthFirst(0)
depthFirst(1)
depthFirst(0)
depthFirst(1)
depthFirst(0)
```

Run-time Stack

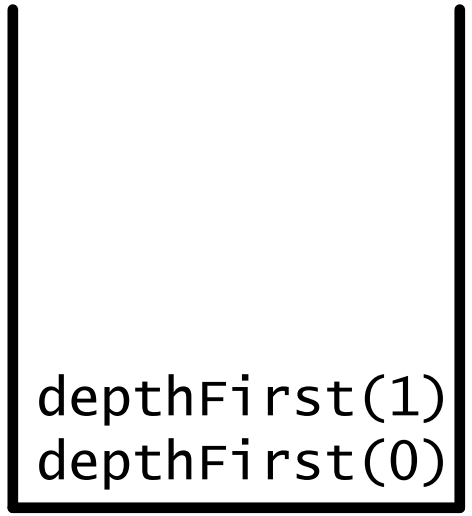
STACK OVERFLOW ERROR



Graphs - Traversal

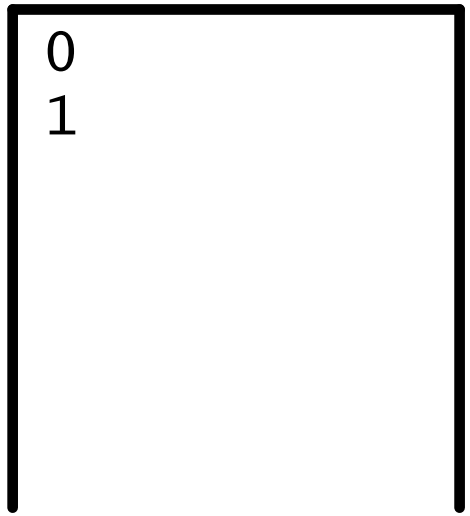
Where is the problem?

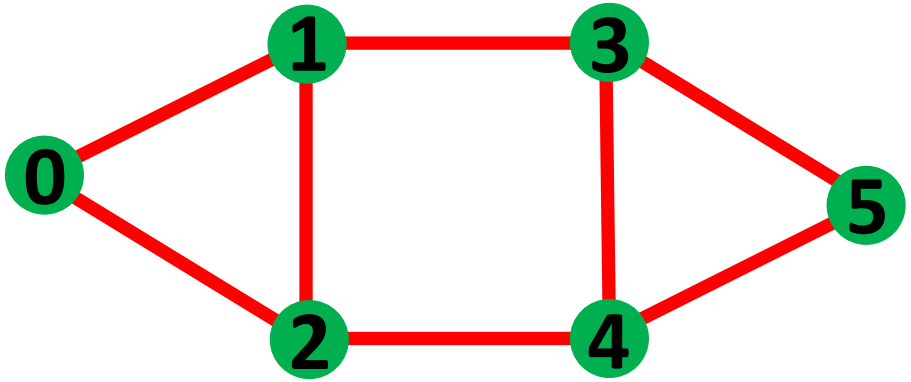
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Run-time Stack

Output

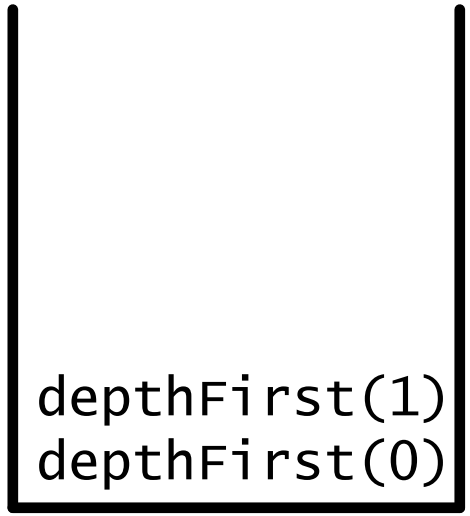




Graphs - Traversal

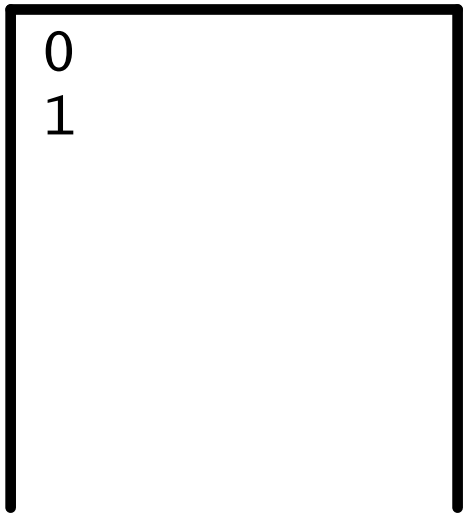
What neighbors should we call `depthFirst()` on?

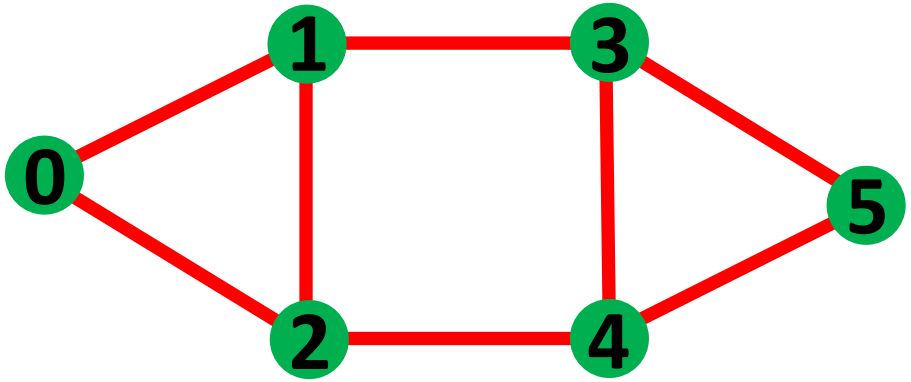
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Run-time Stack

Output



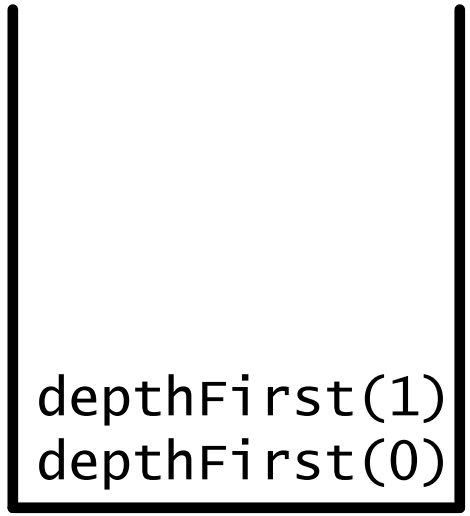


Graphs - Traversal

What neighbors should we call `depthFirst()` on?

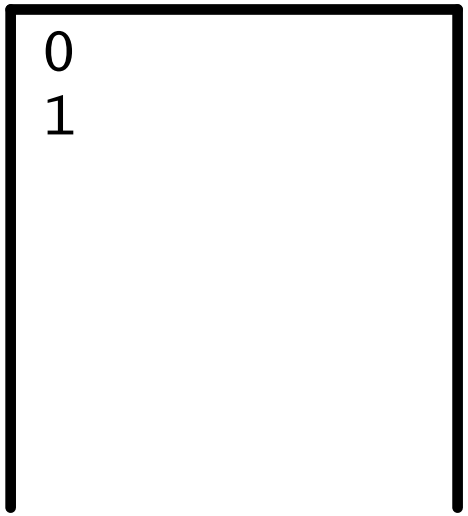
Neighbors that have not already been visited.

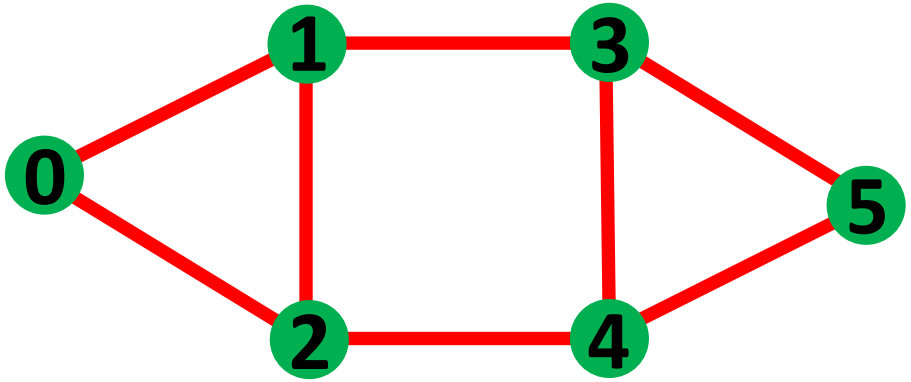
```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



Run-time Stack

Output





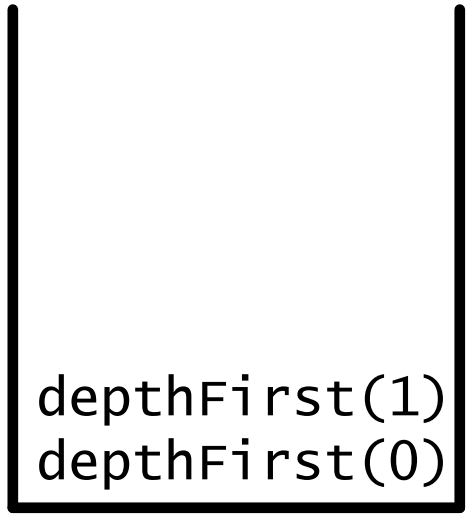
Graphs - Traversal

```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```

What neighbors should we call `depthFirst()` on?

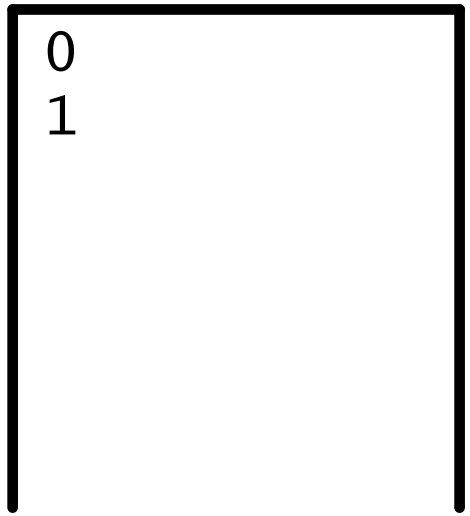
Neighbors that have not already been visited.

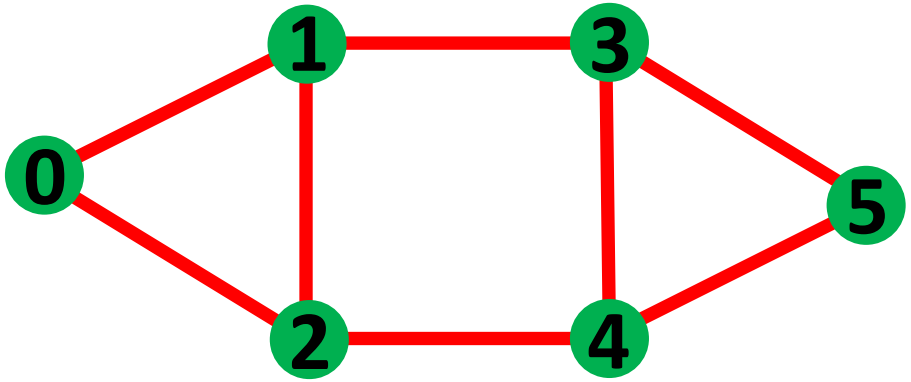
How can we do that?



Run-time Stack

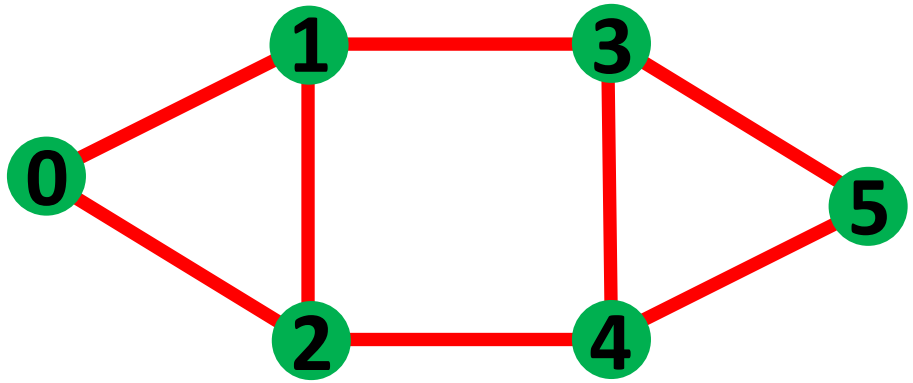
Output





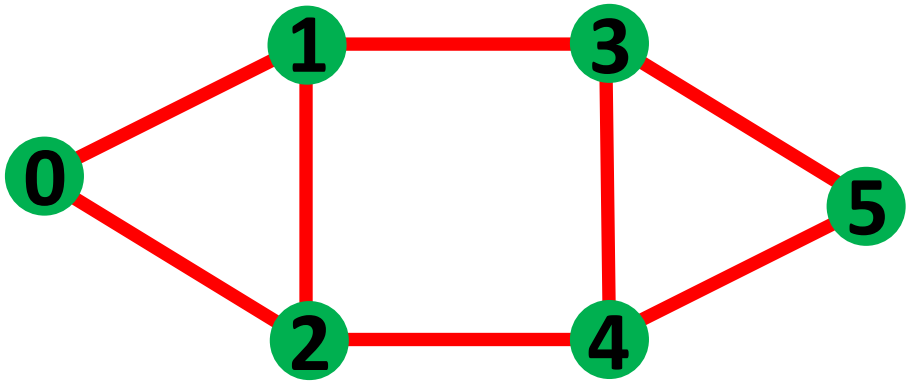
Graphs - Traversal

```
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



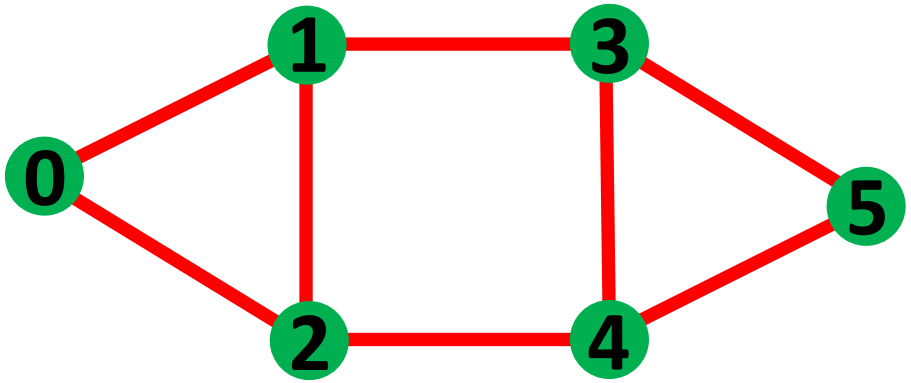
Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        depthFirst(neighbor);
    }
}
```



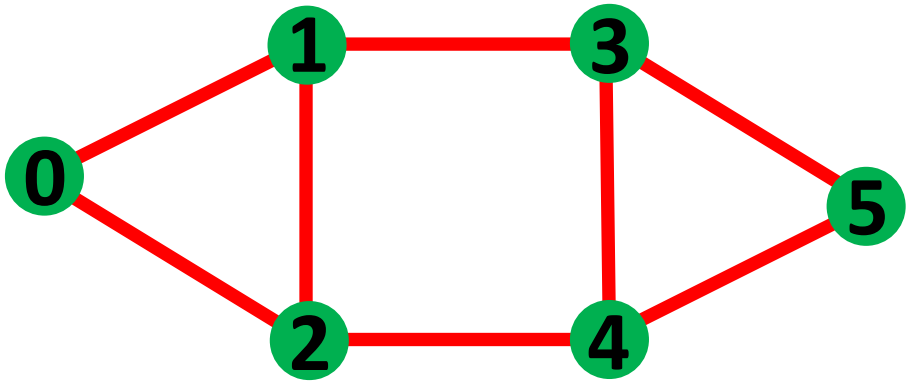
Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



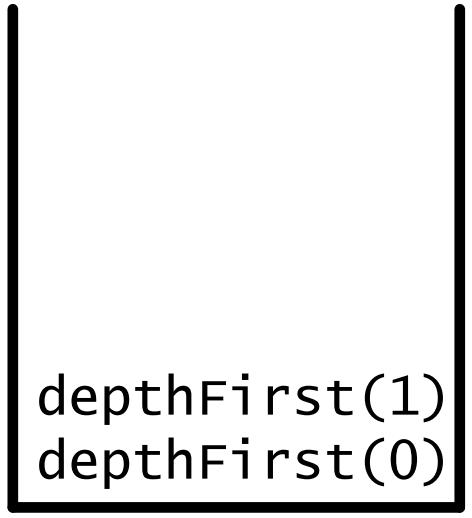
Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



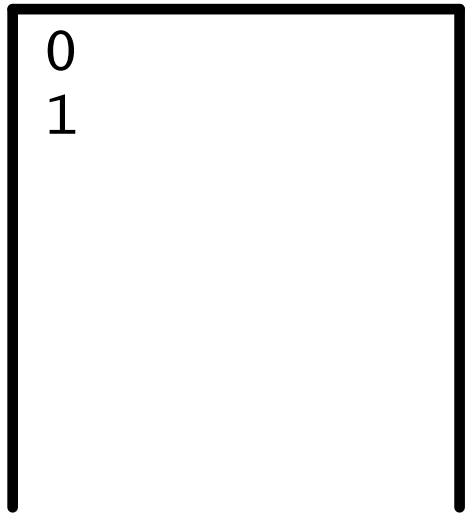
Graphs - Traversal

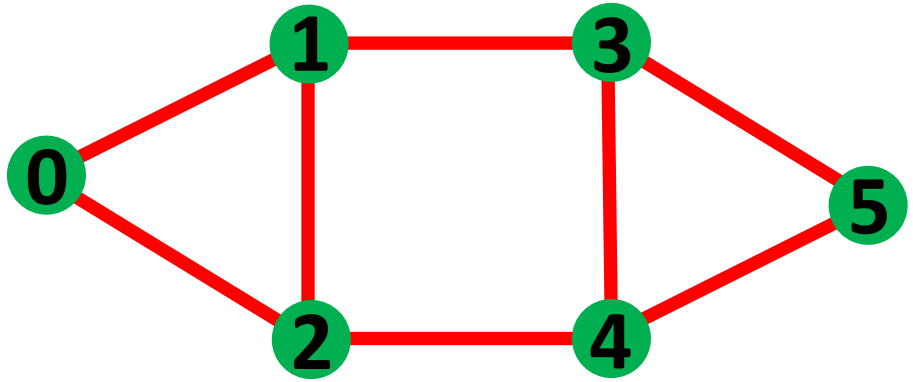
```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Run-time Stack

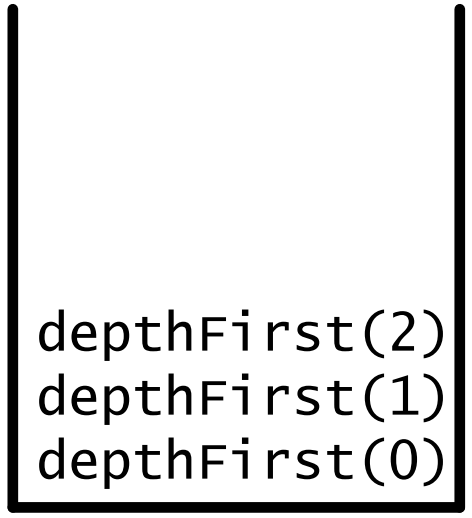
Output





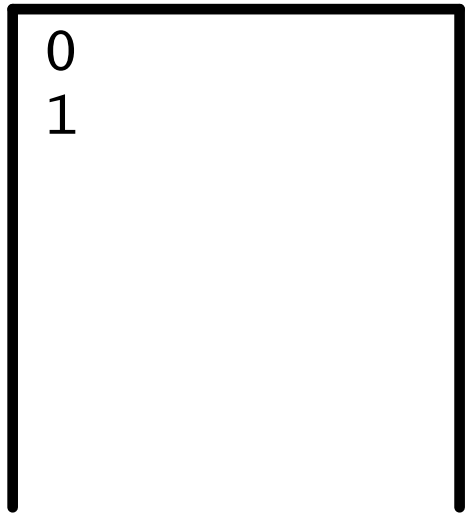
Graphs - Traversal

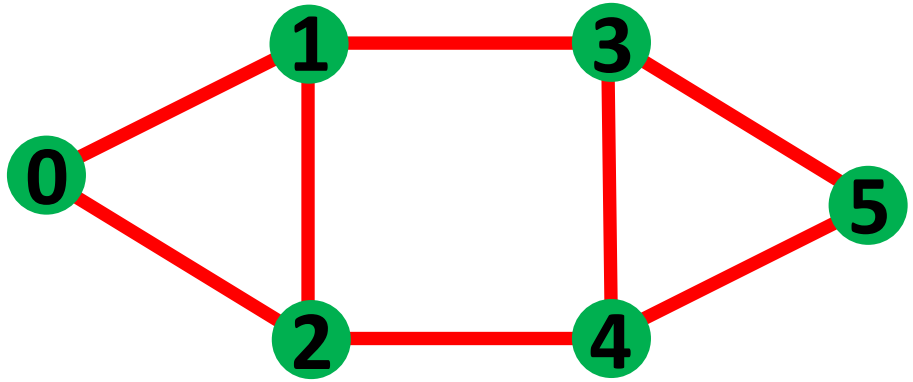
```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Run-time Stack

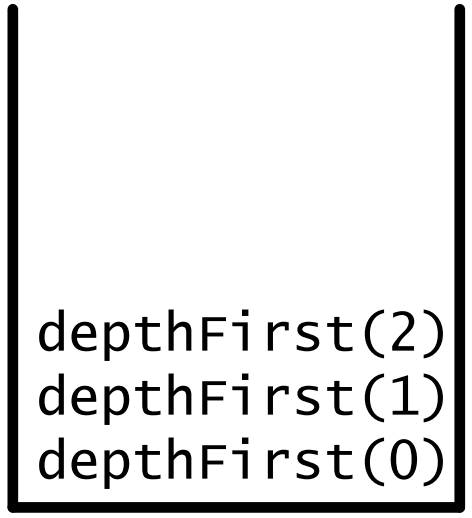
Output





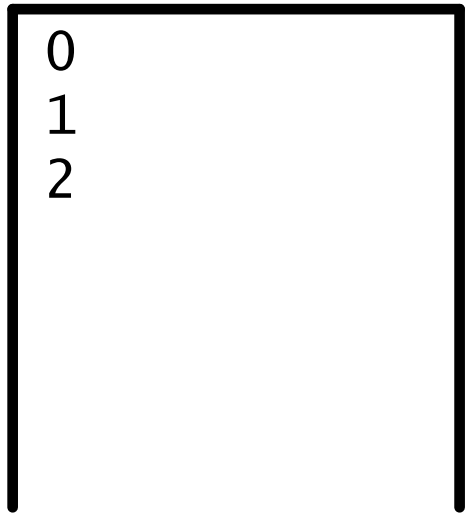
Graphs - Traversal

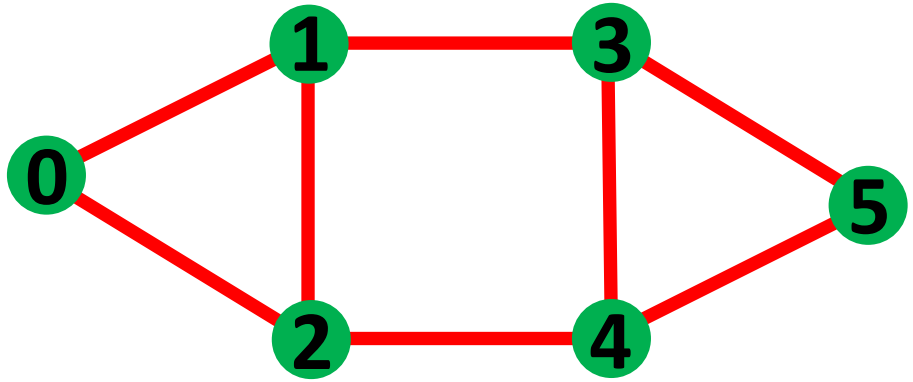
```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Run-time Stack

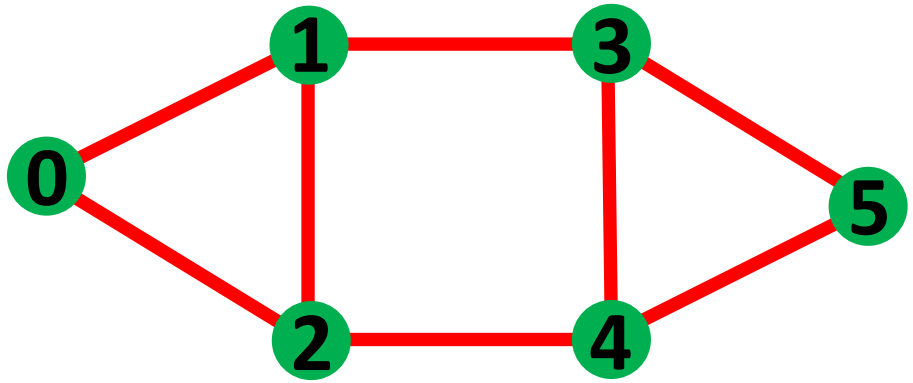
Output





Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Output

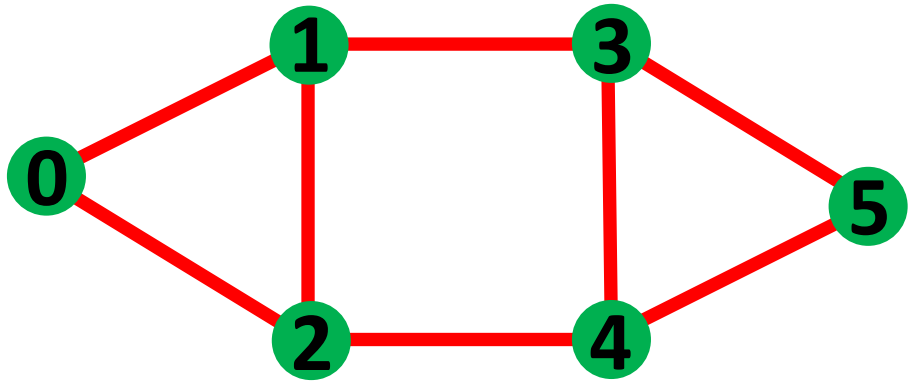
0
1
2

depthFirst(4)
depthFirst(2)
depthFirst(1)
depthFirst(0)

Run-time Stack

Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Output

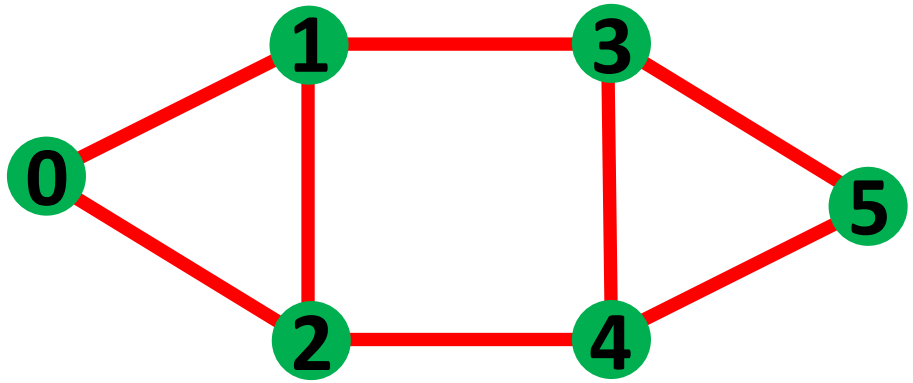
0
1
2
4

depthFirst(4)
depthFirst(2)
depthFirst(1)
depthFirst(0)

Run-time Stack

Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Output

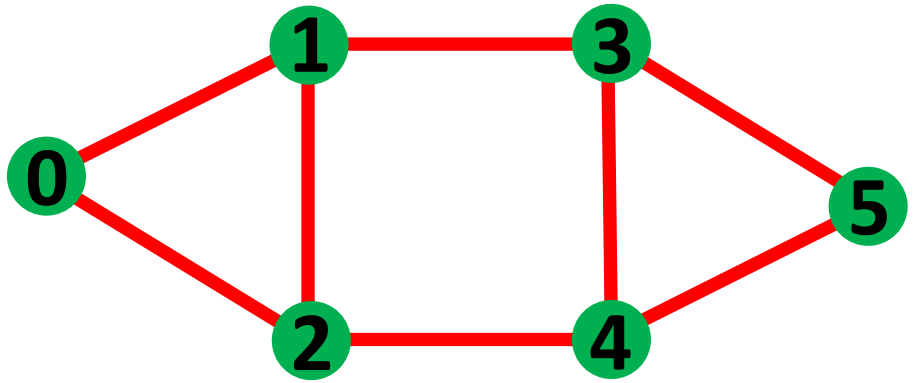
0
1
2
4

depthFirst(3)
depthFirst(4)
depthFirst(2)
depthFirst(1)
depthFirst(0)

Run-time Stack

Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Output

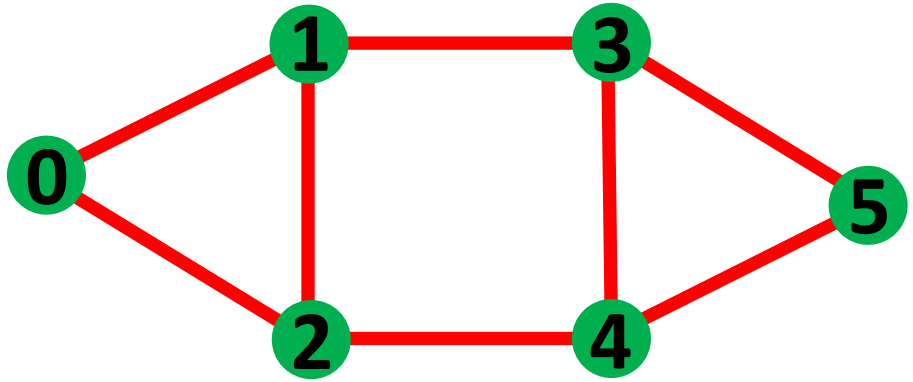
0
1
2
4
3

depthFirst(3)
depthFirst(4)
depthFirst(2)
depthFirst(1)
depthFirst(0)

Run-time Stack

Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



depthFirst(5)
depthFirst(3)
depthFirst(4)
depthFirst(2)
depthFirst(1)
depthFirst(0)

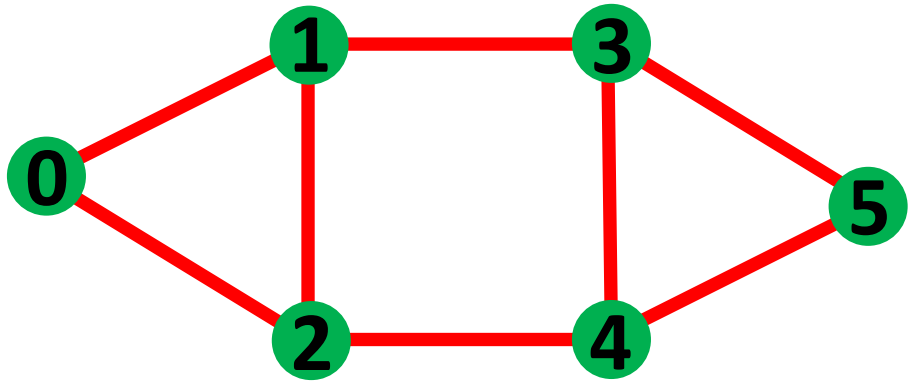
Run-time Stack

Output

0
1
2
4
3

Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



depthFirst(5)
depthFirst(3)
depthFirst(4)
depthFirst(2)
depthFirst(1)
depthFirst(0)

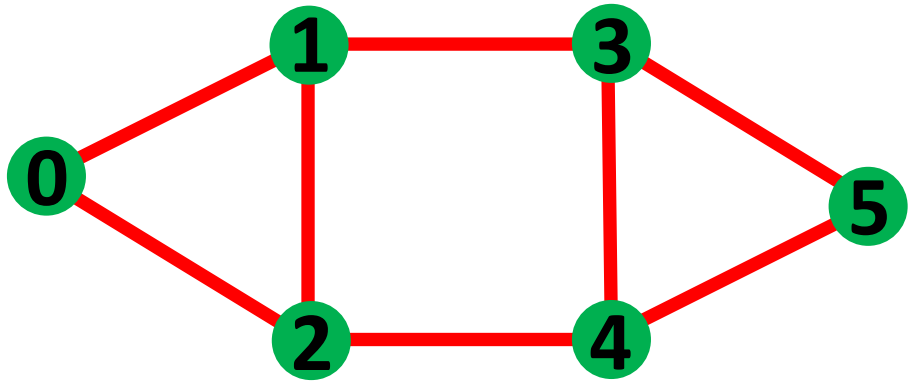
Run-time Stack

Output

0
1
2
4
3
5

Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Output

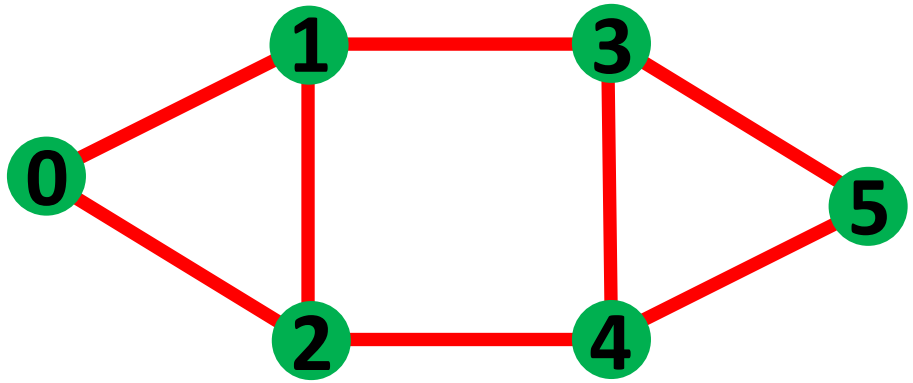
0
1
2
4
3
5

depthFirst(3)
depthFirst(4)
depthFirst(2)
depthFirst(1)
depthFirst(0)

Run-time Stack

Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Output

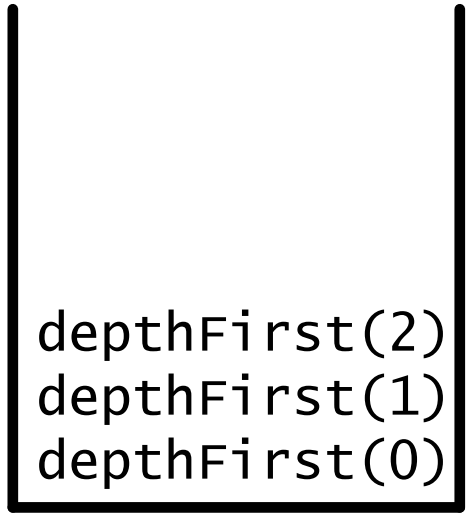
0
1
2
4
3
5

depthFirst(4)
depthFirst(2)
depthFirst(1)
depthFirst(0)

Run-time Stack

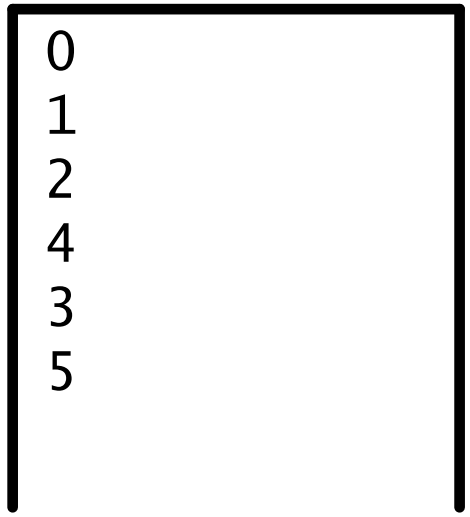
Graphs - Traversal

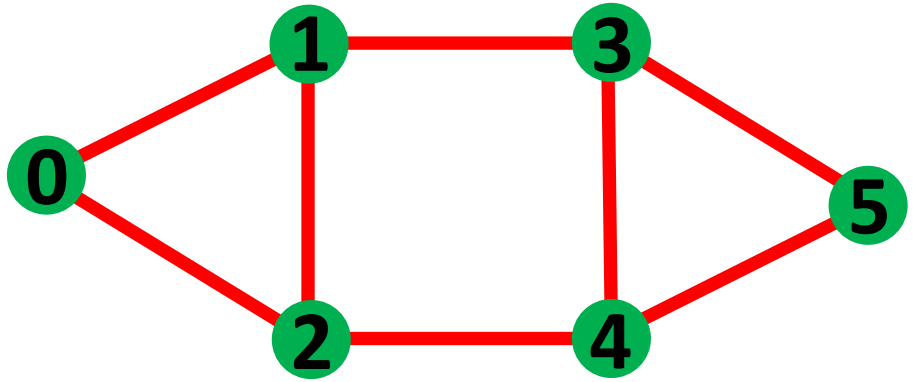
```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Run-time Stack

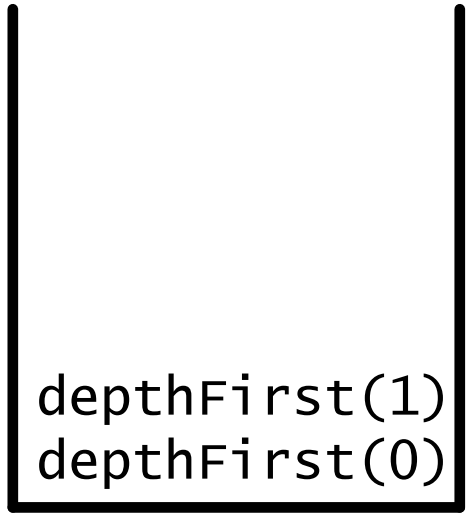
Output





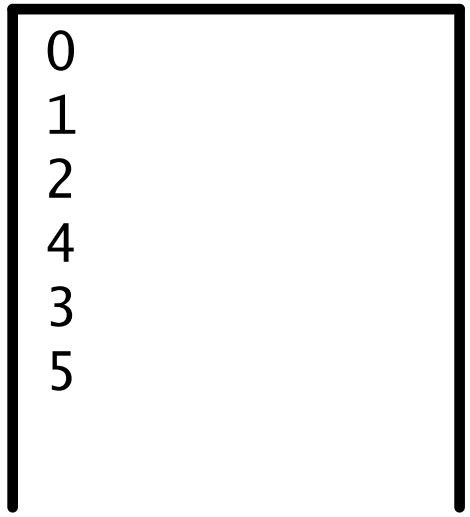
Graphs - Traversal

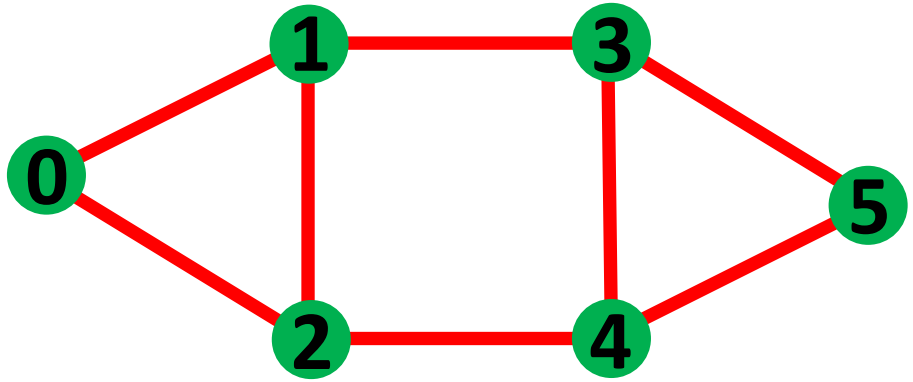
```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Run-time Stack

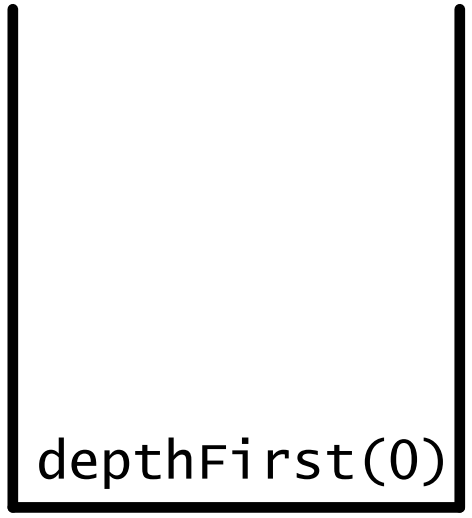
Output





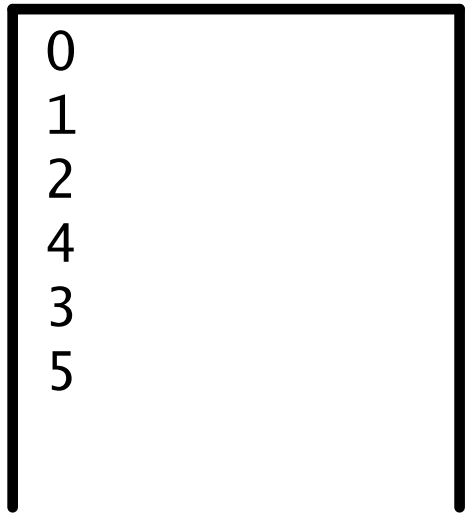
Graphs - Traversal

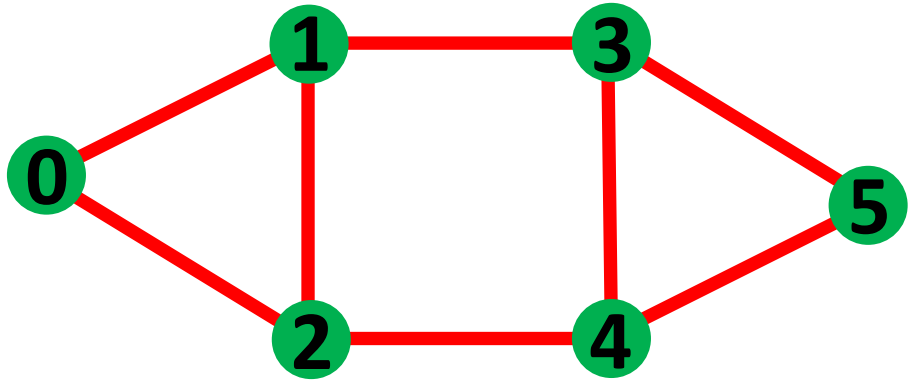
```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Run-time Stack

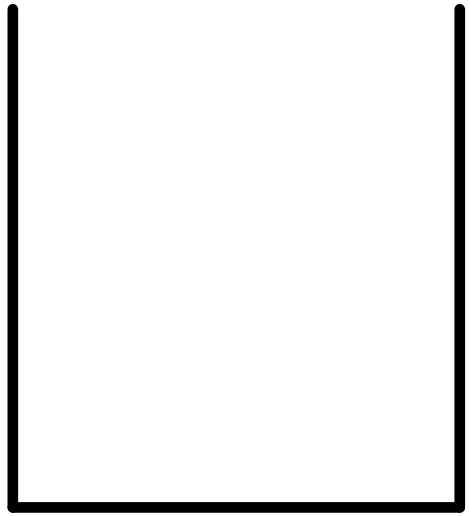
Output





Graphs - Traversal

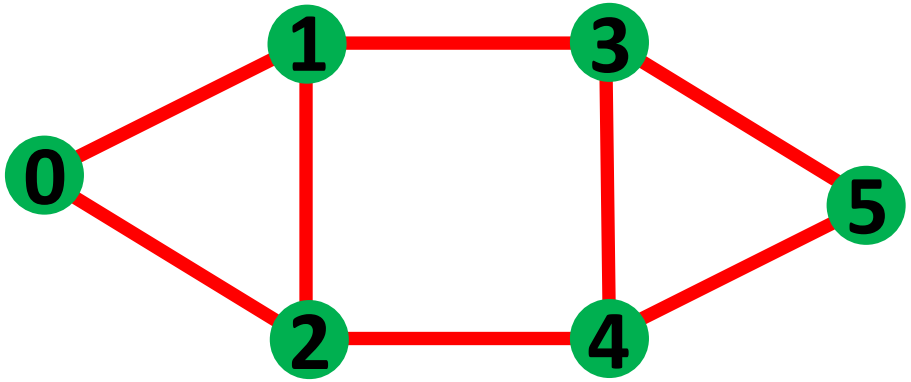
```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Run-time Stack

Output

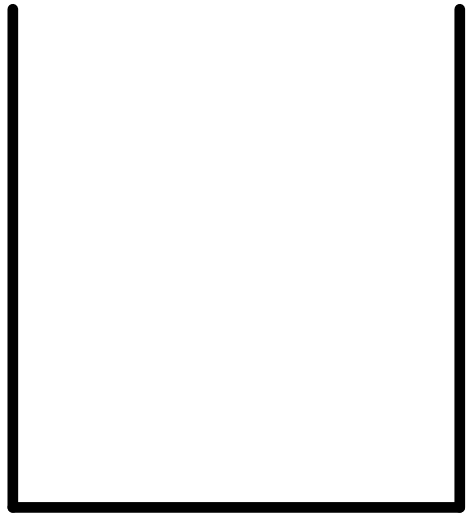
0
1
2
4
3
5



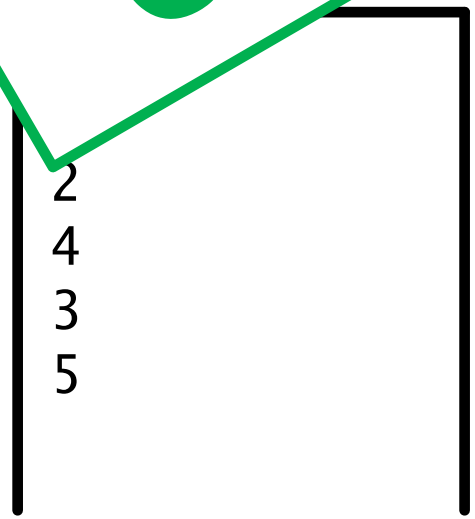
Graphs - Traversal

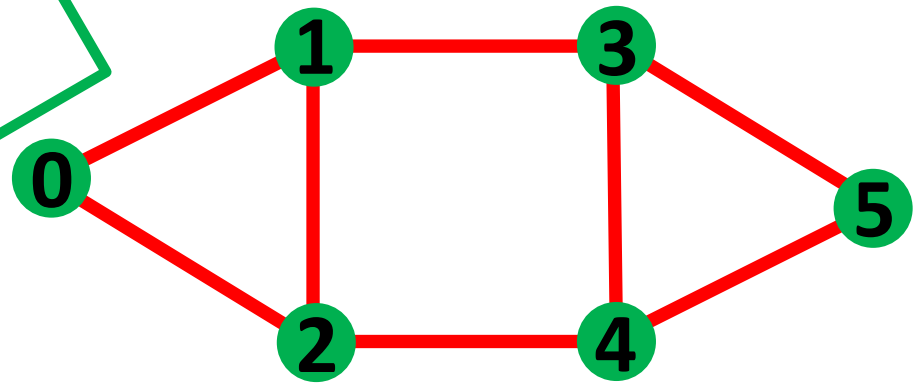
```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbor
        if (!visited[neighbor])
            depthFirst(neigh
        }
    }
}
```

CODE



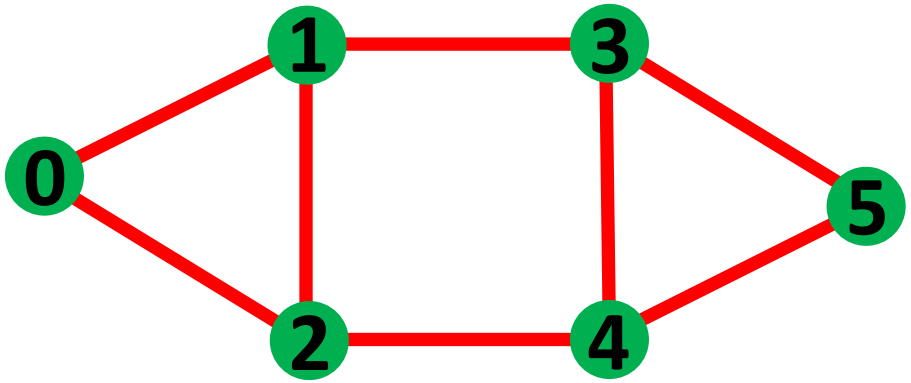
Run-time Stack





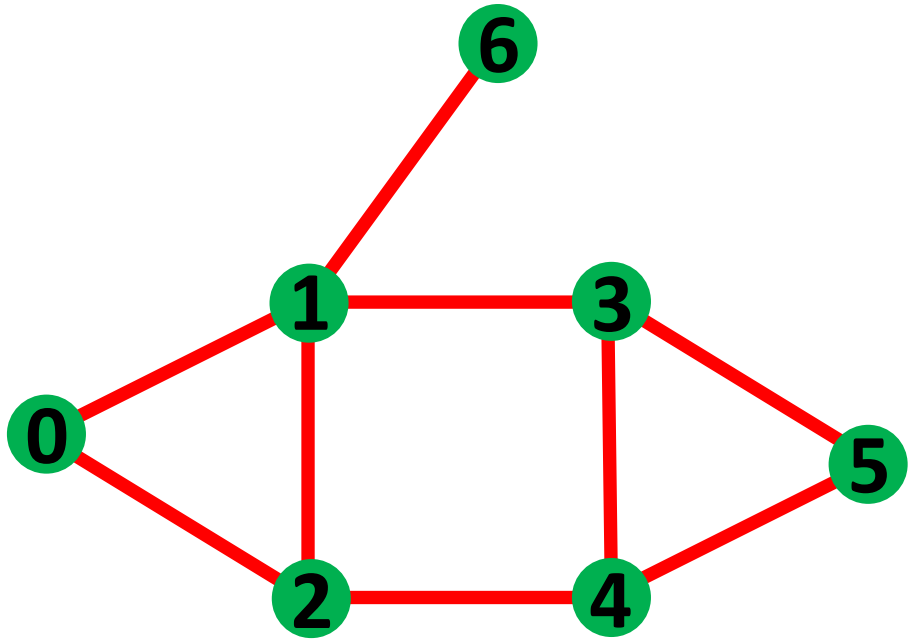
Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



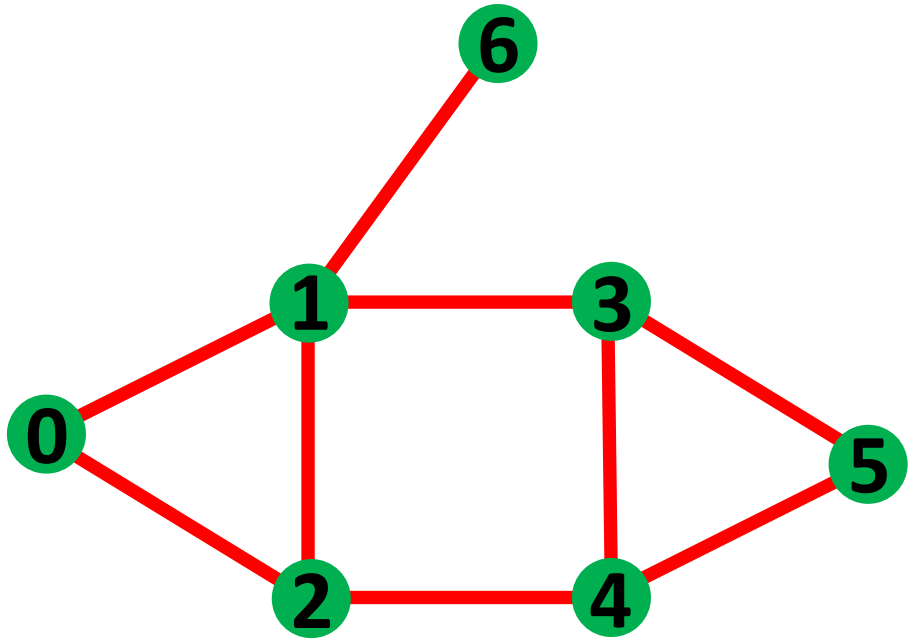
Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



depthFirst(5)
depthFirst(3)
depthFirst(4)
depthFirst(2)
depthFirst(1)
depthFirst(0)

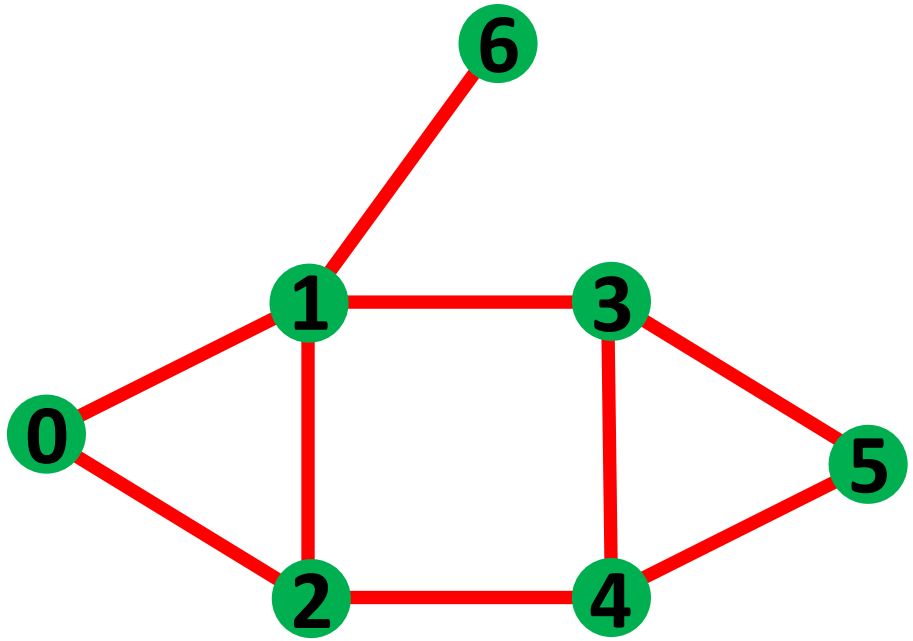
Run-time Stack

Output

0
1
2
4
3
5

Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Output

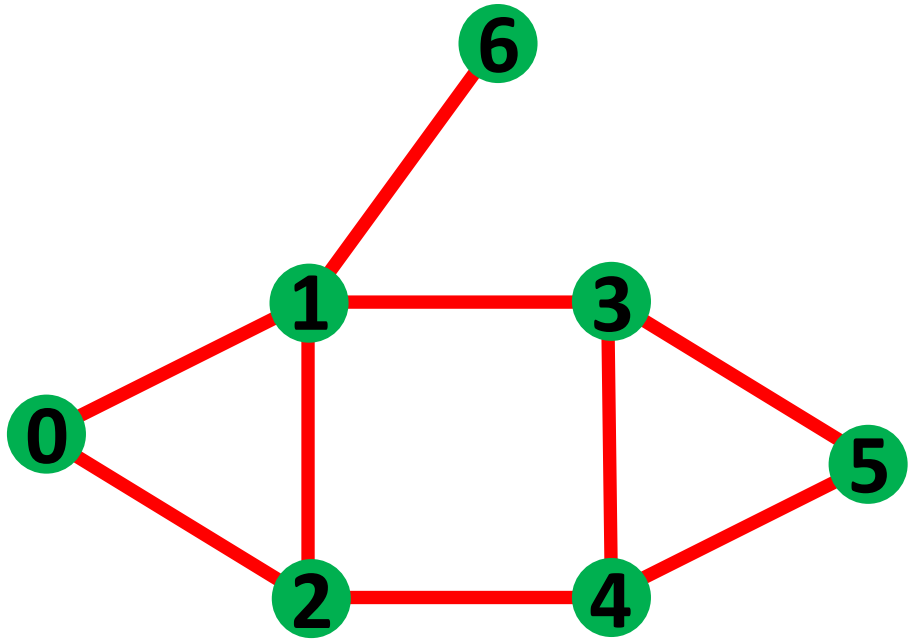
0
1
2
4
3
5

depthFirst(3)
depthFirst(4)
depthFirst(2)
depthFirst(1)
depthFirst(0)

Run-time Stack

Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Output

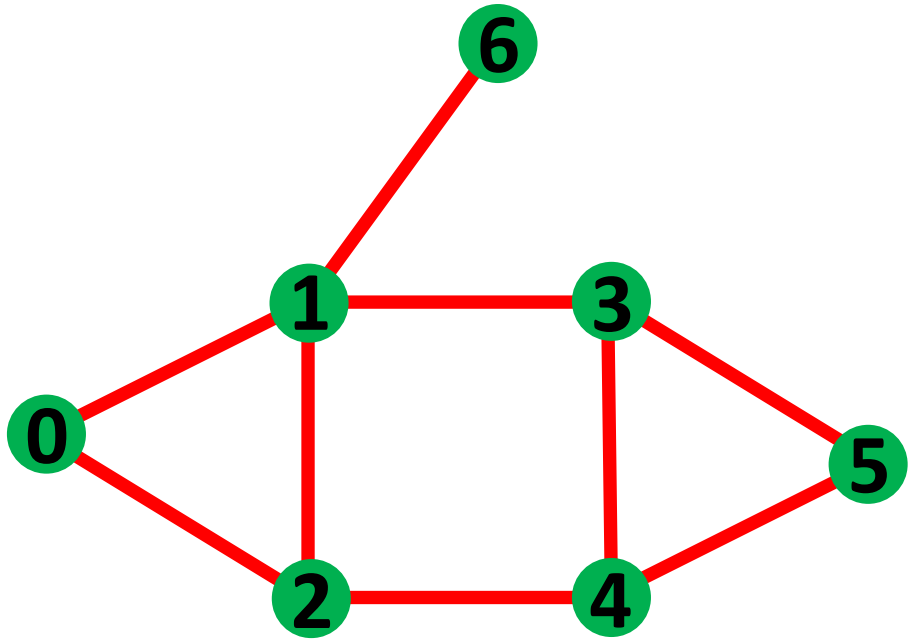
0
1
2
4
3
5

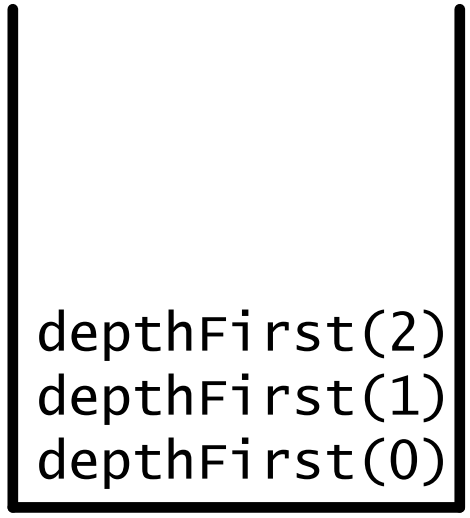
depthFirst(4)
depthFirst(2)
depthFirst(1)
depthFirst(0)

Run-time Stack

Graphs - Traversal

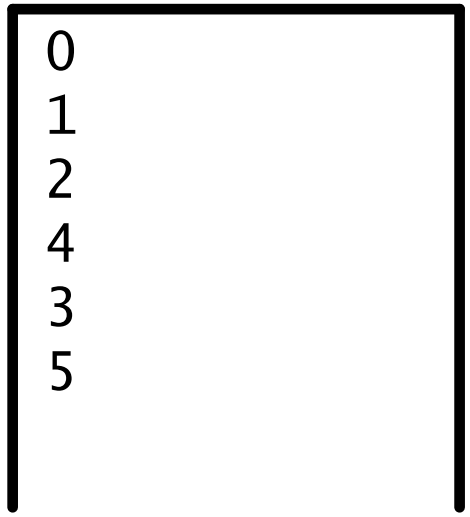
```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```





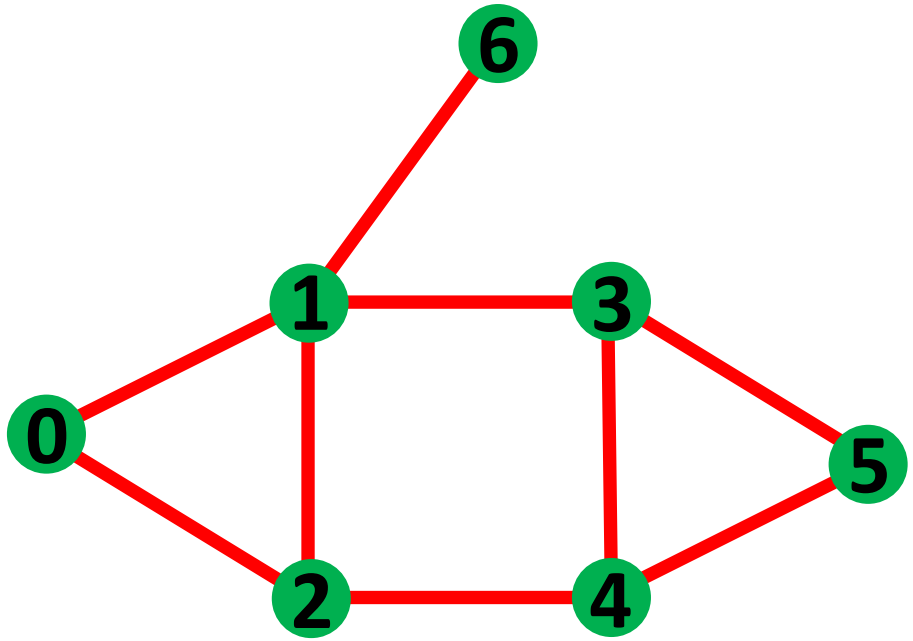
Run-time Stack

Output



Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Output

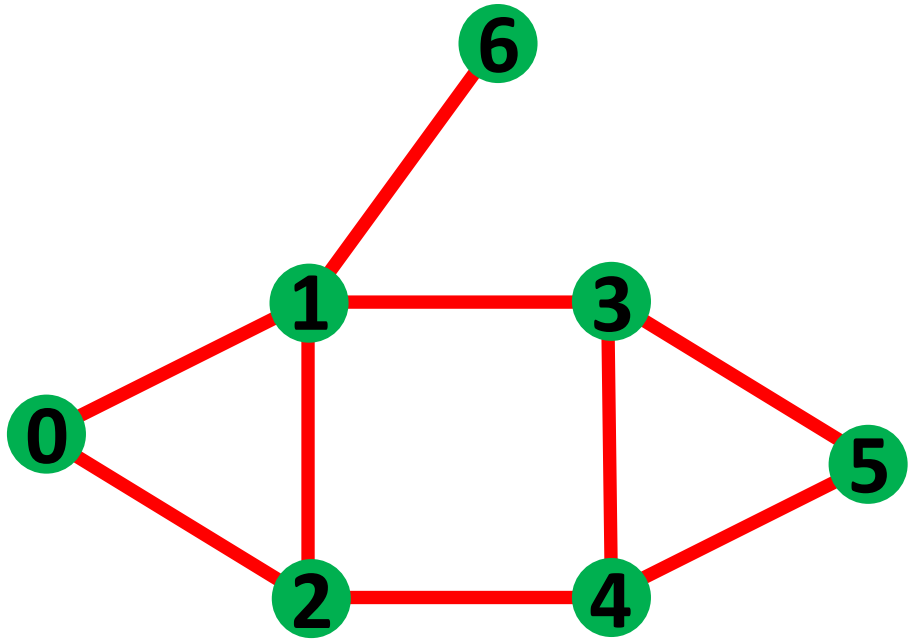
0
1
2
4
3
5

depthFirst(1)
depthFirst(0)

Run-time Stack

Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Output

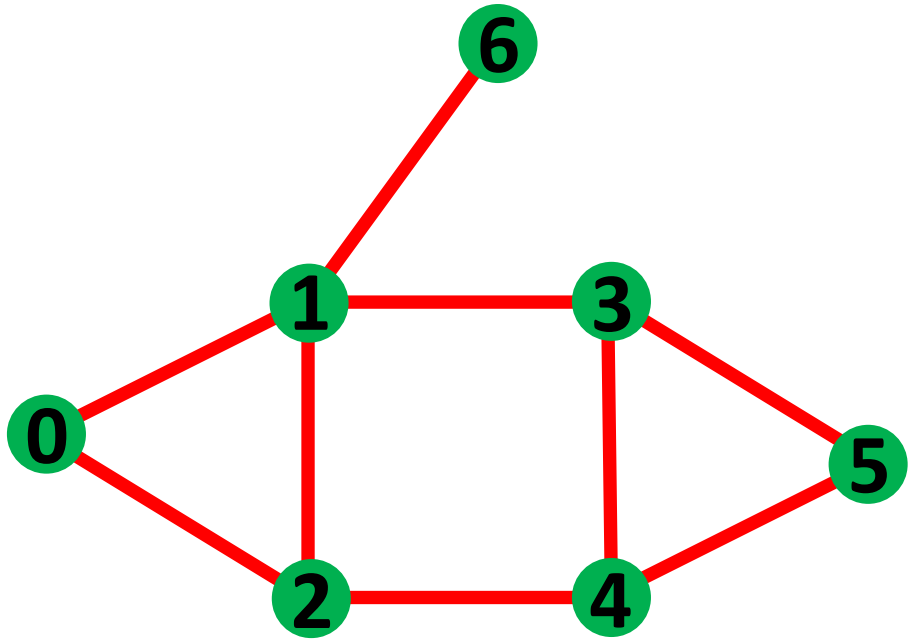
0
1
2
4
3
5

depthFirst(6)
depthFirst(1)
depthFirst(0)

Run-time Stack

Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Output

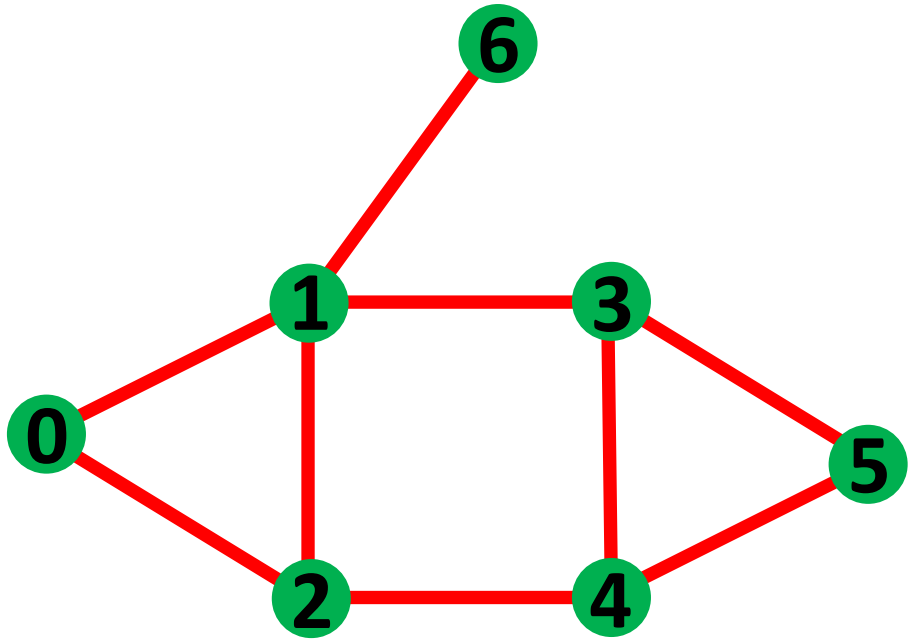
0
1
2
4
3
5
6

depthFirst(6)
depthFirst(1)
depthFirst(0)

Run-time Stack

Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Output

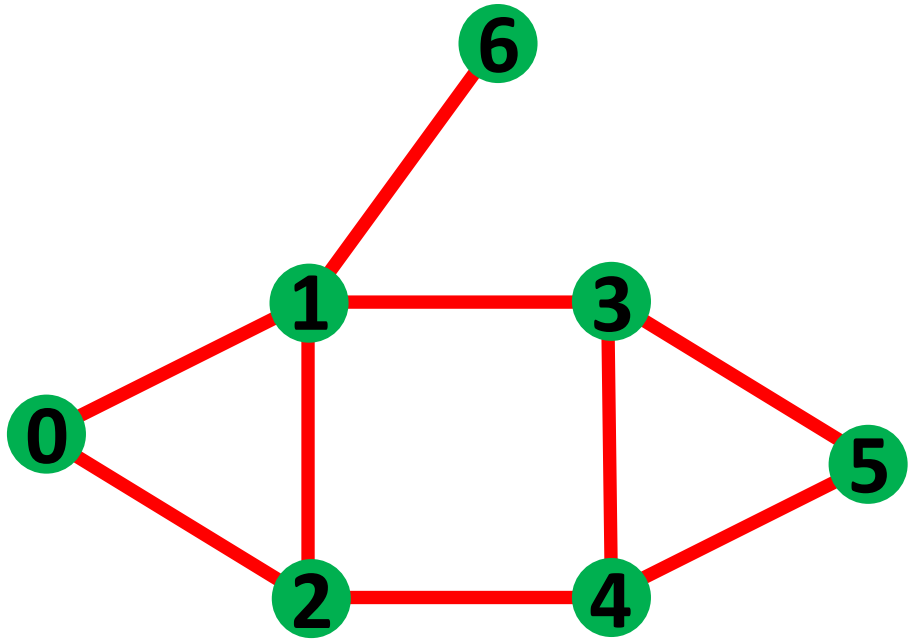
0
1
2
4
3
5
6

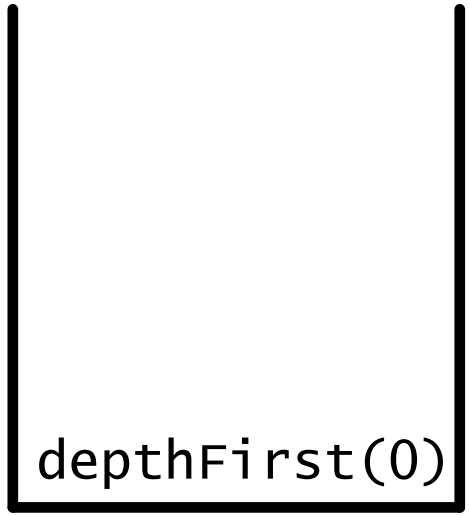
depthFirst(1)
depthFirst(0)

Run-time Stack

Graphs - Traversal

```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```





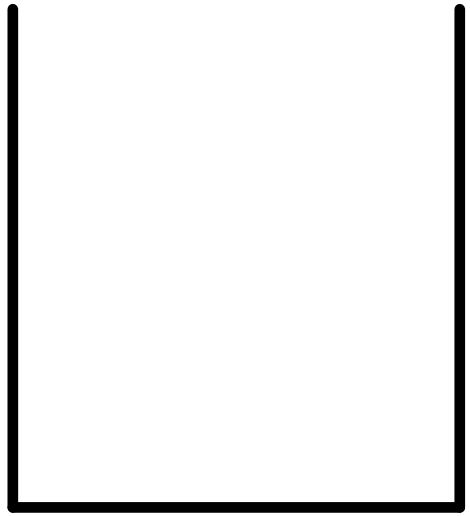
Run-time Stack

Output

0
1
2
4
3
5
6

Graphs - Traversal

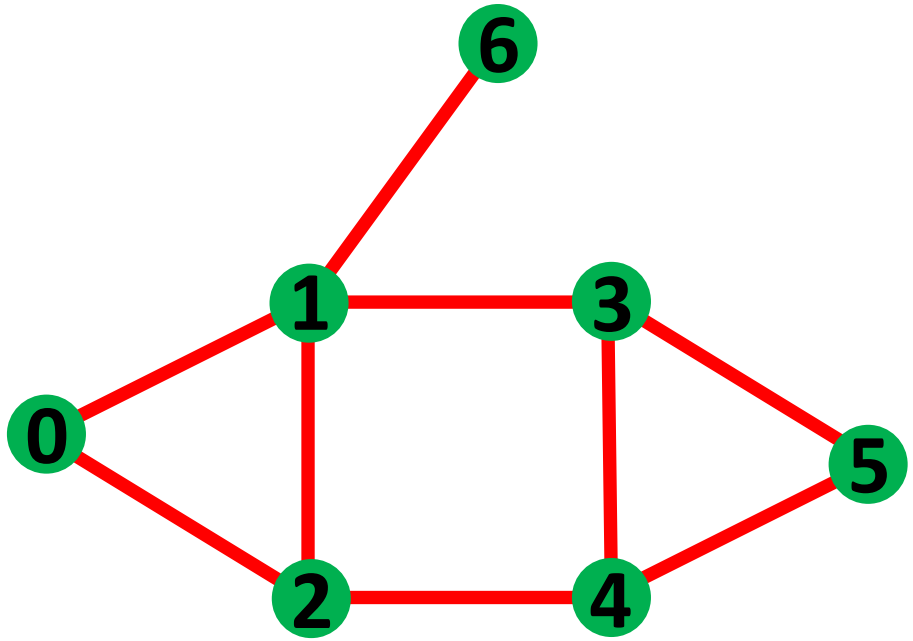
```
private boolean visited[] = new boolean[getNumVertices()];
public void depthFirst(int n) {
    System.out.println(n);
    visited[n] = true;
    for (int neighbor : getNeighbors(n)) {
        if (!visited[neighbor]) {
            depthFirst(neighbor);
        }
    }
}
```



Run-time Stack

Output

0
1
2
4
3
5
6



Graphs - Paths

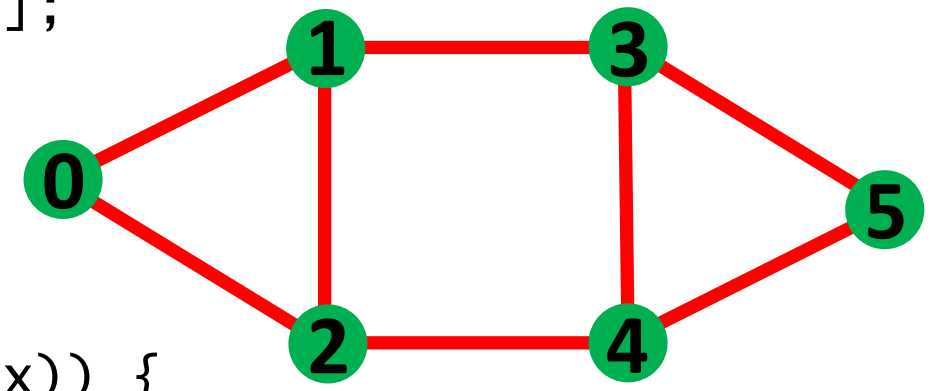
```
private boolean[] visited;

public DepthFirstSearch(Graph graph, int startVertex) {
    visited = new boolean[graph.getNumVertices()];
    dfs(graph, startVertex);
}

private void dfs(Graph graph, int vertex) {
    visited[vertex] = true;

    for (int neighbor : graph.getNeighbors(vertex)) {
        if (!visited[neighbor]) {
            dfs(graph, neighbor);
        }
    }
}

public boolean reachable(int endVertex) {
    return visited[endVertex];
}
```



Graphs - Paths

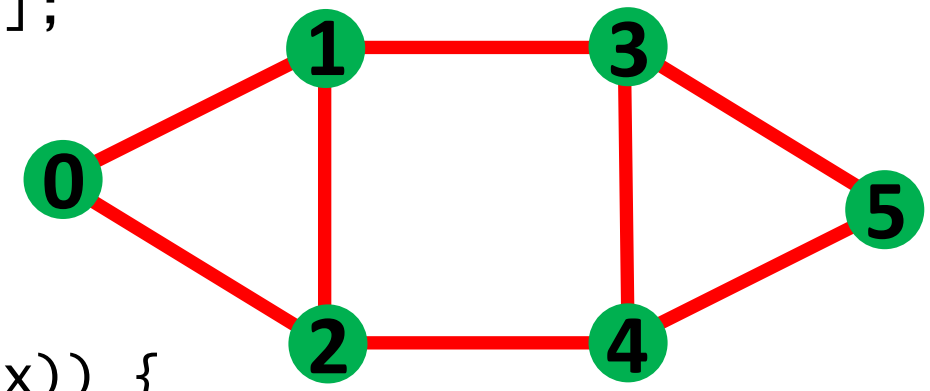
```
private boolean[] visited;

public DepthFirstSearch(Graph graph, int startVertex) {
    visited = new boolean[graph.getNumVertices()];
    dfs(graph, startVertex);
}

private void dfs(Graph graph, int vertex) {
    visited[vertex] = true;

    for (int neighbor : graph.getNeighbors(vertex)) {
        if (!visited[neighbor]) {
            dfs(graph, neighbor);
        }
    }
}

public boolean reachable(int endVertex) {
    return visited[endVertex];
}
```



How do we get actual paths between vertices?