THE GRAPH

REPRESENTATION

ADJACENCY MATRIX

ADJACENCY LIST

ADJACENCY SET

THE GRAPH
REPRESENTATION

EACH VERTEX IS A NODE

EACH VERTEX HAS A POINTER TO A LINKED LIST

THIS LINKED LIST CONTAINS ALL THE OTHER NODES THIS VERTEX CONNECTS TO DIRECTLY

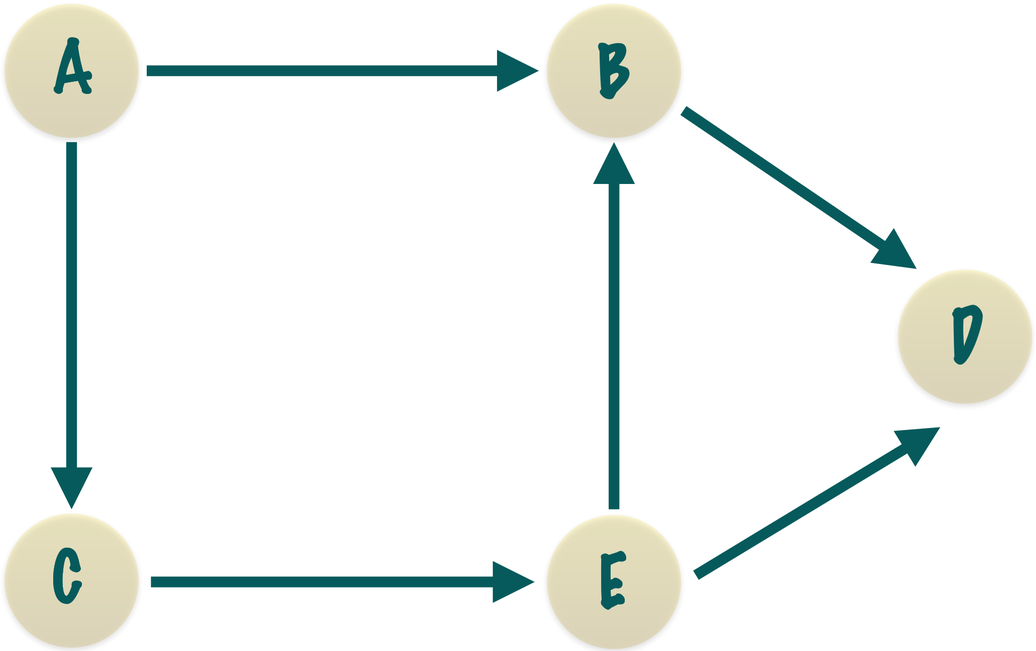
ADJACENCY LIST

IF A VERTEX V HAS AN EDGE LEADING TO ANOTHER VERTEX U

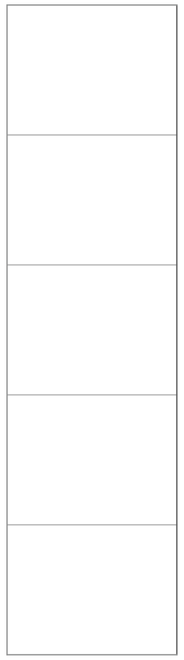
THEN U IS PRESENT IN V'S LINKED LIST

THE GRAPH

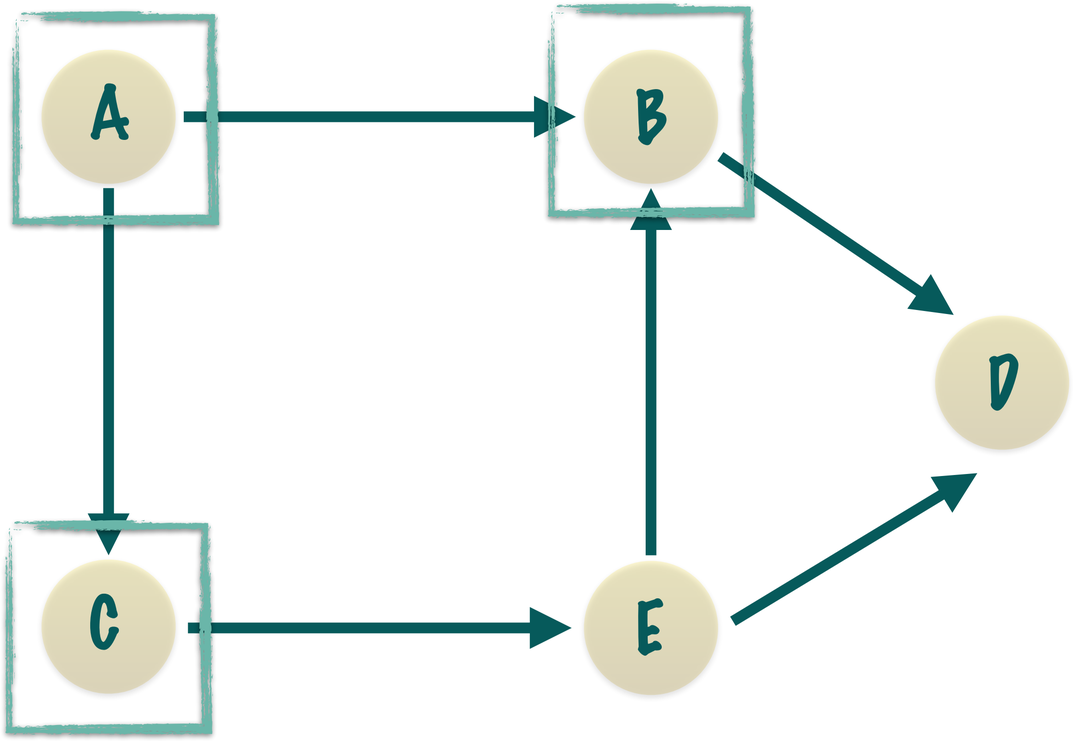
REPRESENTATION



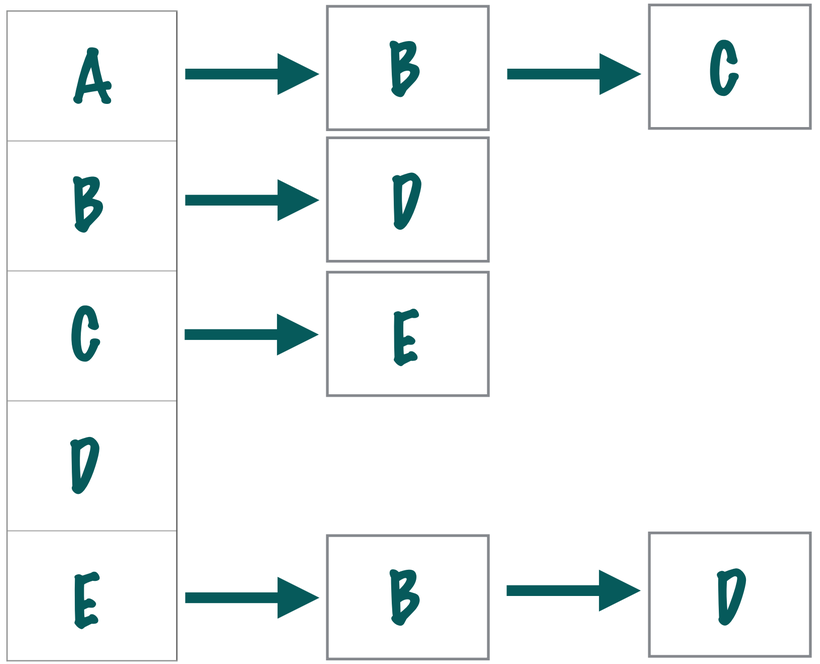
ADJACENCY LIST



THE GRAPH
REPRESENTATION

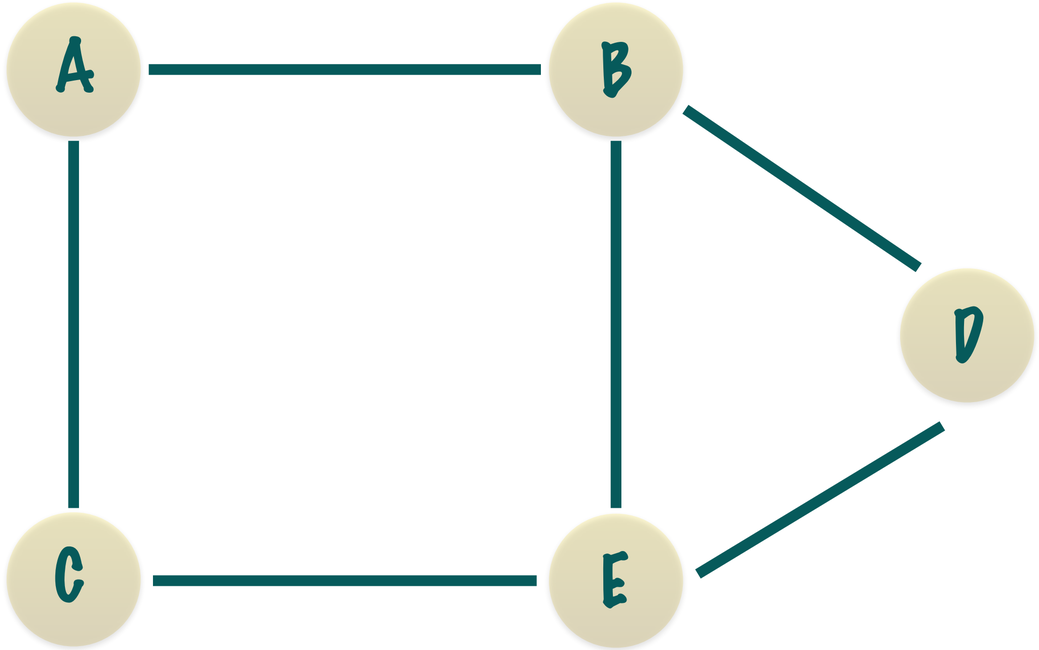


ADJACENCY LIST

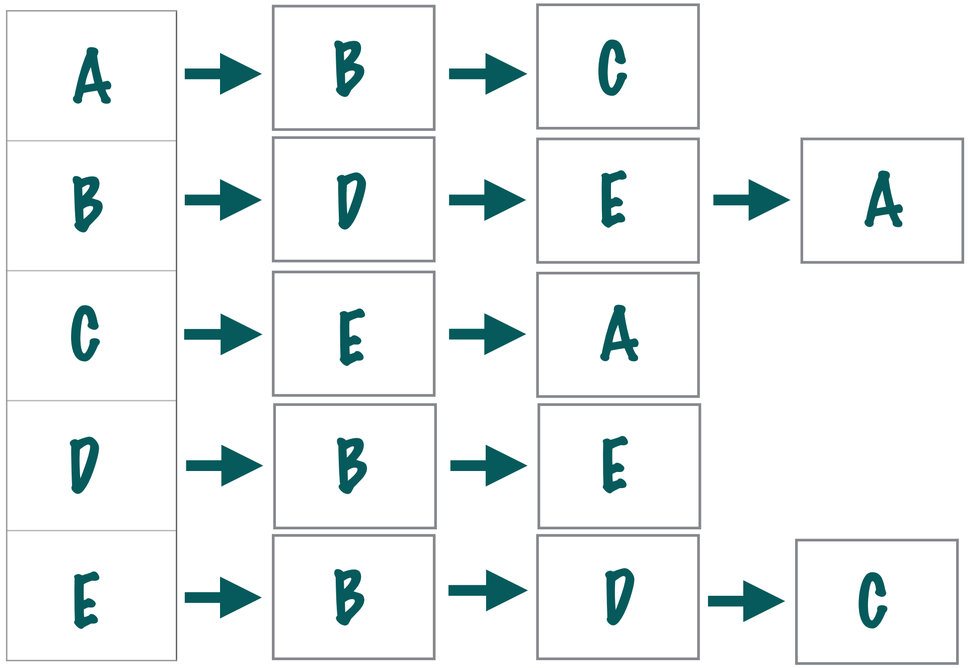


THE GRAPH

REPRESENTATION



ADJACENCY LIST



THE GRAPH
REPRESENTATION

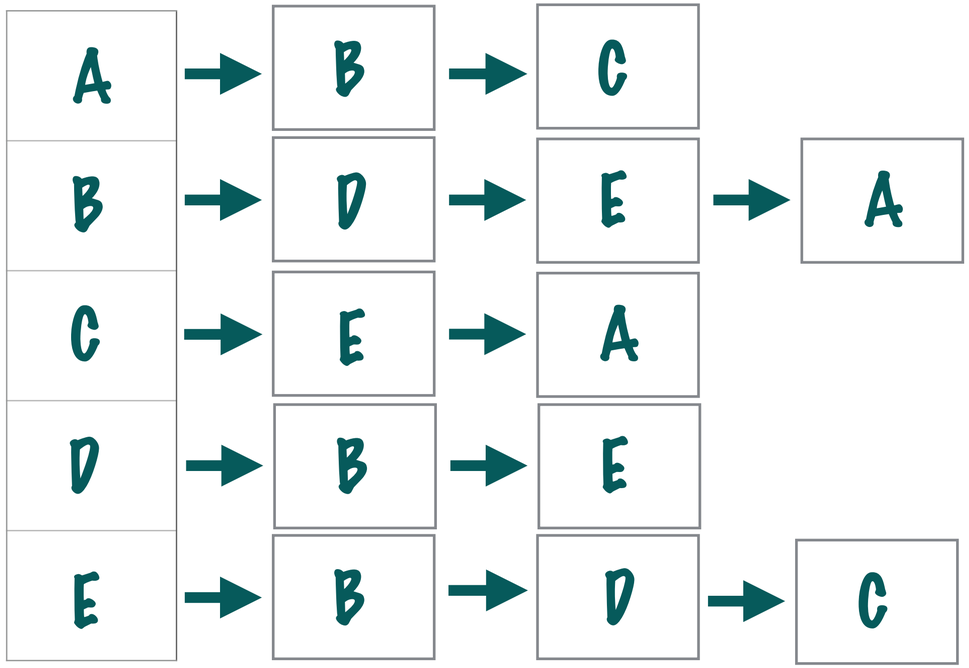
```
class Node {
   int vertexId;
   Node next;
}
```

OR

```
class Node {
   int vertexId;
   List<Node> nodes;
}
```

ADJACENCY LIST

```
class Graph {
   List<Node> vertices;
}
```



THE GRAPH

REPRESENTATION

ADJACENCY LIST

ADJACENCY LISTS ARE NOT THE BEST REPRESENTATIONS OF GRAPHS

THEY HAVE SOME MAJOR DOWNSIDES

THE ORDER OF THE VERTICES IN THE ADJACENCY LISTS MATTER

THE SAME GRAPH CAN HAVE MULTIPLE REPRESENTATIONS

CERTAIN OPERATIONS BECOME TRICKY E.G. DELETING A NODE INVOLVES LOOKING THROUGH ALL THE ADJACENCY LISTS TO REMOVE THE NODE FROM ALL LISTS

THE GRAPH
REPRESENTATION

ADJACENCY MATRIX
ADJACENCY LIST
ADJACENCY SET

THE GRAPH
REPRESENTATION

ADJACENCY SET

THIS IS VERY SIMILAR TO AN ADJACENCY LIST

INSTEAD OF A LINKED LIST TO MAINTAIN THE ADJACENT VERTICES

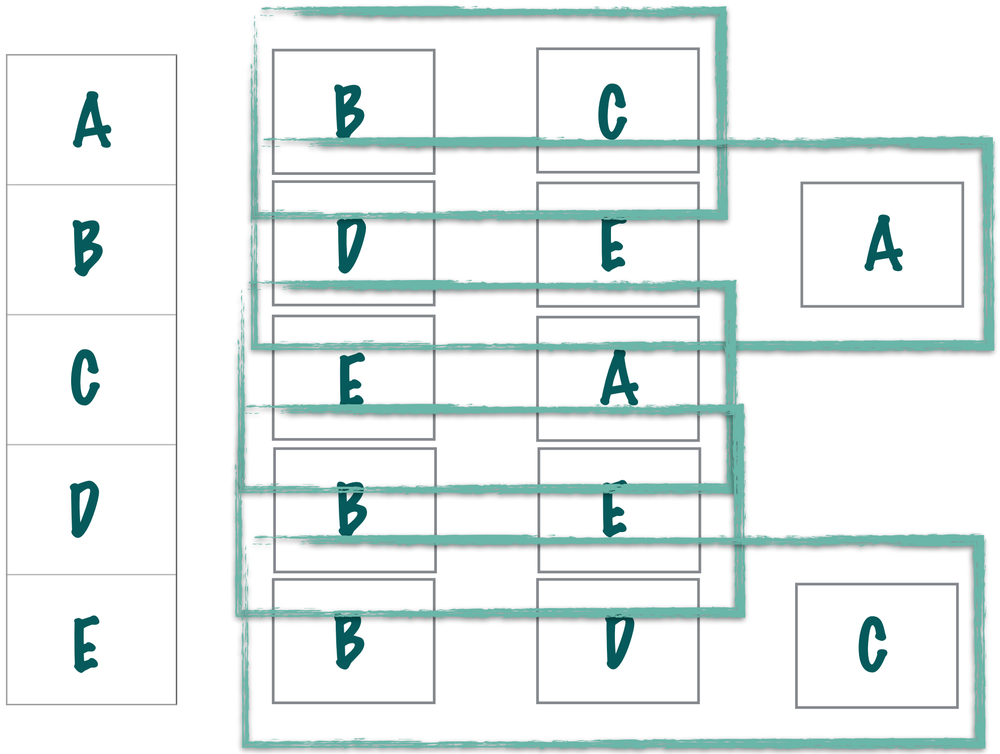
USE A SET

THE GRAPH
REPRESENTATION

```
class Node {
   int vertexId;
   Set<Node> nodes;
}


class Graph {
   List<Node> vertices;
}
```

ADJACENCY SET



THE GRAPH

REPRESENTATION

ADJACENCY SET

NOW LET'S SEE SOME CODE...

THE VERTEX NODE

```
public static class Node {

    private int vertexNumber;
    private Set<Integer> adjacencySet = new HashSet<>();

    public Node(int vertexNumber) {
        this.vertexNumber = vertexNumber;
    }

    public int getVertexNumber() {
        return vertexNumber;
    }

    public void addEdge(int vertexNumber) {
        adjacencySet.add(vertexNumber);
    }

    public List<Integer> getAdjacentVertices() {
        List<Integer> sortedList = new ArrayList<>(adjacencySet);

        Collections.sort(sortedList);

        return sortedList;
    }
}
```

A CLASS WHICH REPRESENTS A VERTEX

EACH NODE HOLDS A SET OF ADJACENT VERTICES

EACH VERTEX HAS AN INDEX OR UNIQUE NUMBER ASSOCIATED WITH IT

HELPER METHOD TO ADD AN EDGE WITH THIS NODE AS THE SOURCE

GET THE ADJACENT VERTICES FOR THIS NODE

THE ADJACENCY SET

```
public class AdjacencySetGraph implements Graph {

    private List<Node> vertexList = new ArrayList<>();
    private GraphType graphType = GraphType.DIRECTED;
    private int numVertices = 0;

    public AdjacencySetGraph(int numVertices, GraphType graphType) {
        this.numVertices = numVertices;
        for (int i = 0; i < numVertices; i++) {
            vertexList.add(new Node(i));
        }
        this.graphType = graphType;
    }
```

THIS IMPLEMENTS THE GRAPH INTERFACE - THE USE OF THE ADJACENCY SET IS AN IMPLEMENTATION DETAIL

SET UP A LIST OF VERTEX NODES, REMEMBER EACH NODE HOLDS A SET OF ADJACENT VERTICES

THIS CAN BE A DIRECTED OR UNDIRECTED GRAPH

INITIALIZE THE VERTEX LIST, AND OTHER INFORMATION IN THE CONSTRUCTOR

ADJACENCY SET - ADD EDGE

```java
@Override
public void addEdge(int v1, int v2) {
    if (v1 >= numVertices || v1 < 0 || v2 >= numVertices || v2 < 0) {
        throw new IllegalArgumentException("Vertex number is not valid: " + v1 + ", " + v2);
    }

    vertexList.get(v1).addEdge(v2);
    if (graphType == GraphType.UNDIRECTED) {
        vertexList.get(v2).addEdge(v1);
    }
}
```

SPECIFY THE VERTICES THE EDGE CONNECTS - V1 IS THE SOURCE VERTEX AND V2 IS THE DESTINATION VERTEX

ENSURE THE VERTICES ARE VALID

ADD V2 TO THE SET OF NODE V1

IF THE GRAPH IS UNDIRECTED THEN THE CONNECTION GOES BOTH WAYS - SET V1 TO THE SET OF NODE V2

ADJACENCY SET - GET ADJACENT VERTICES

```java
@Override
public List<Integer> getAdjacentVertices(int v) {
    if (v >= numVertices || v < 0) {
        throw new IllegalArgumentException("Vertex number is not valid: " + v);
    }

    return vertexList.get(v).getAdjacentVertices();
}
```

ENSURE THE VERTEX IS VALID

JUST RETURN THE ADJACENT VERTICES FROM THE NODE CLASS