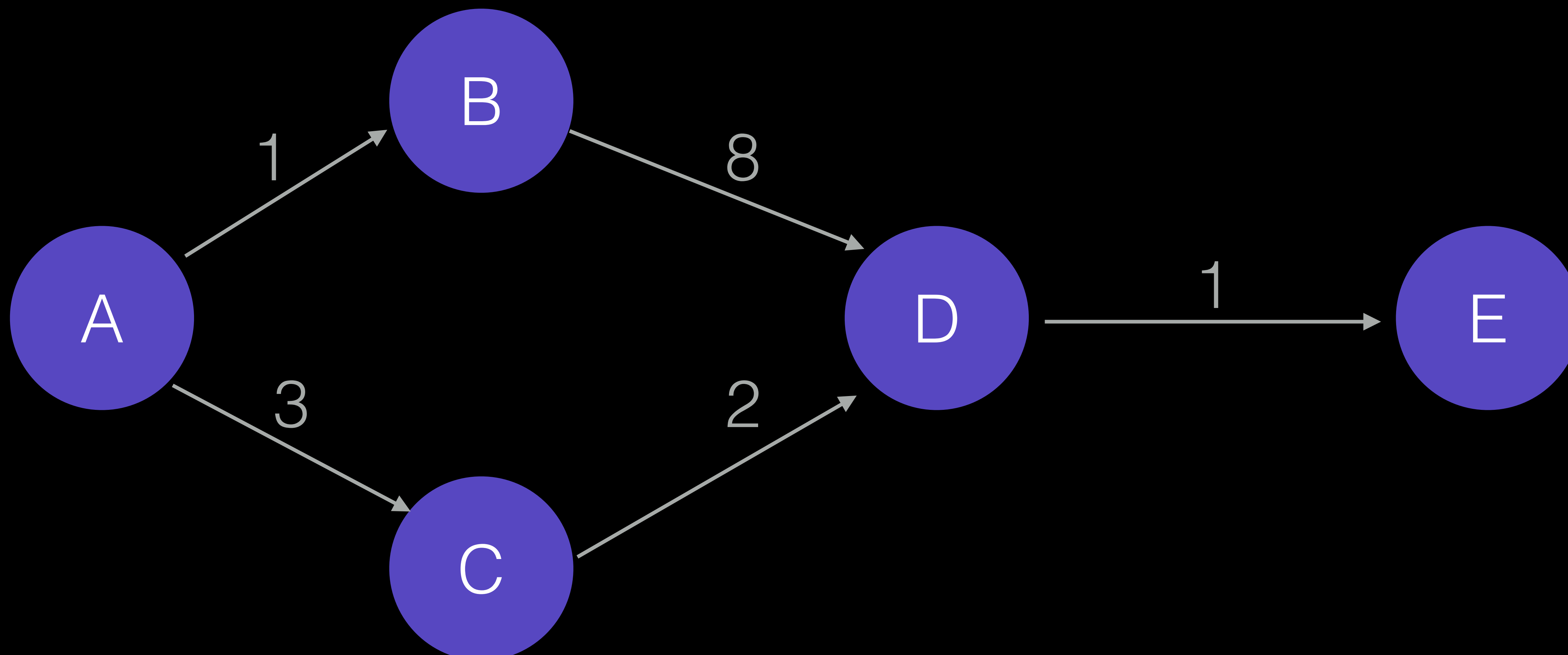
Graph algorithms in their many shapes and sizes

Daniel Epstein, 5/8/14
depstein@cs.washington.edu
https://github.com/depstein/programming-competitions

In mathematics and computer science, **graph theory** is the study of *graphs*, which are mathematical structures used to model pairwise relations between objects. A "graph" in this context is made up of "vertices" or "nodes" and lines called *edges* that connect them. A graph may be *undirected*, In mathematics and computer science, **graph theory** is the study of *graphs*, which are mathematical structures used to model pairwise relations between objects. A "graph" in this context is made up of "vertices" or "nodes" and lines called *edges* that connect them. A graph may be *undirected*,

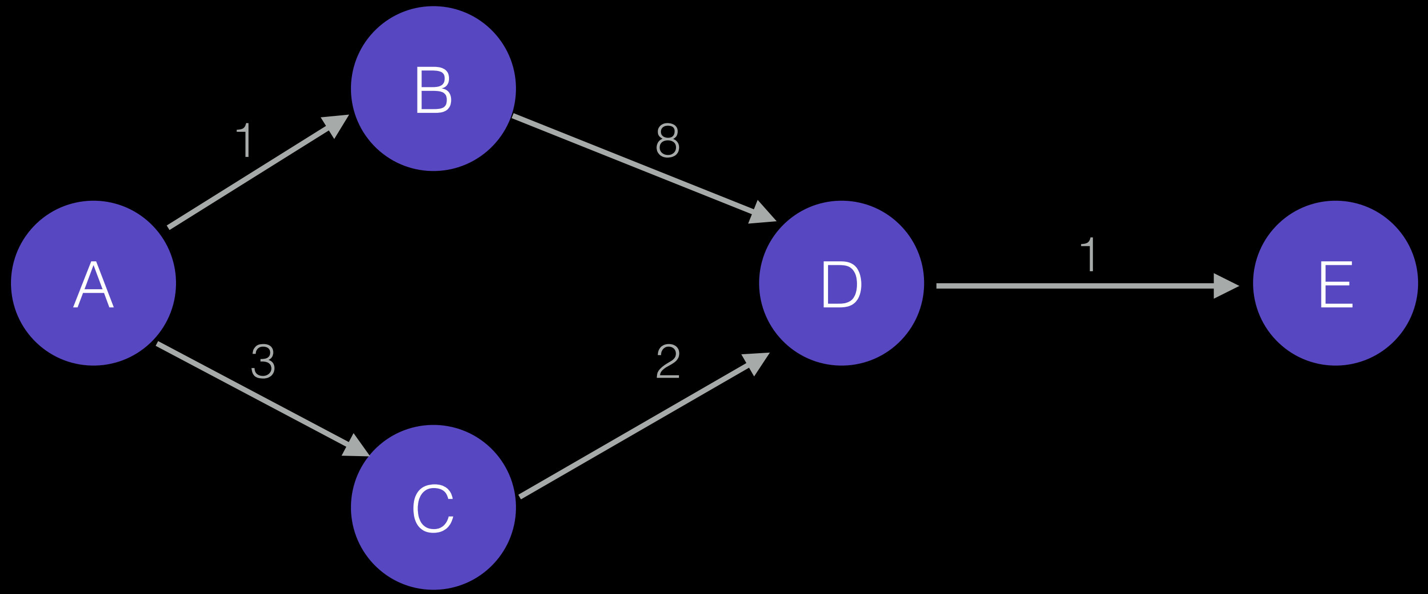
Nodes and Edges!

Graph Representation



Arrays

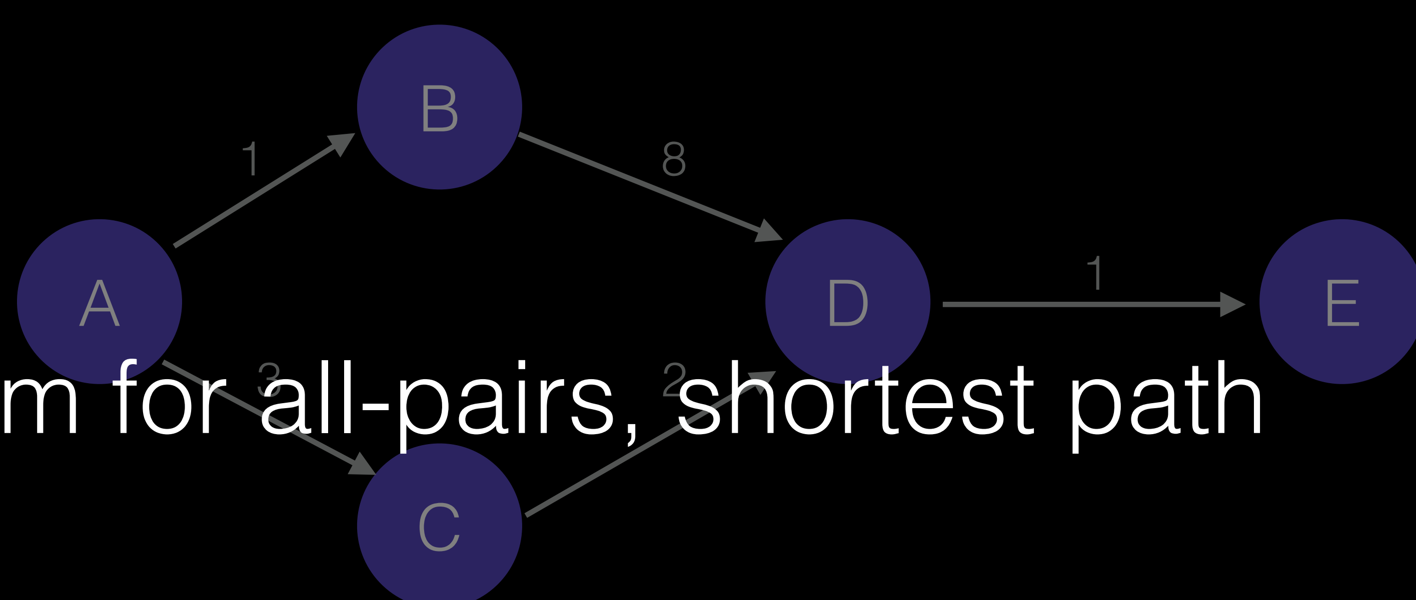
```java
import java.io.*;
import java.util.*;

public class arrays {
  public static int[][] dist = new int[5][5];

  public static void main(String[] args) {
    for(int i=0;i<5;i++)
      for(int j=0;j<5;j++) {
        if(i != j)
          dist[i][j] = 1000; // Not using Integer.MAX_VALUE to avoid integer overflowing
      }
    //Initialize graph as described
    dist[0][1] = 1;
    dist[0][2] = 3;
    dist[1][3] = 8;
    dist[2][3] = 2;
    dist[3][4] = 1;
  }
}
```



Arrays

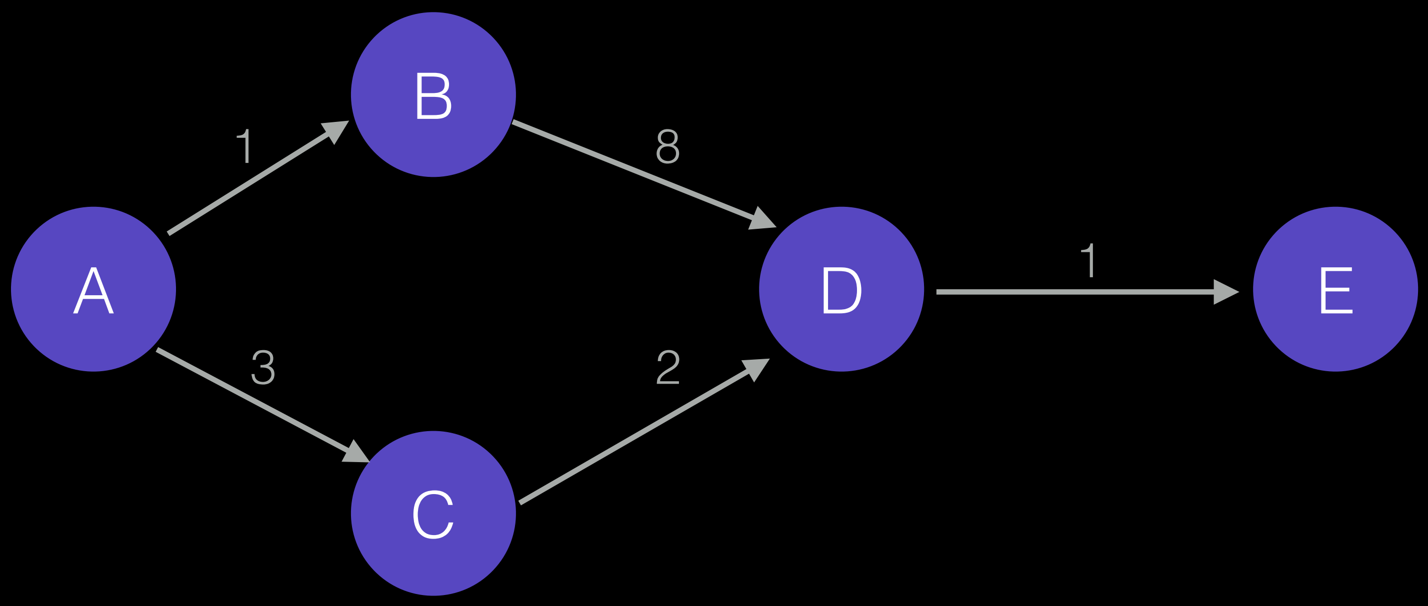
- Know the max number of nodes
- Adjacency Matrix
  - Easy to look up edge weight/edge existence
  - Slow to get all edges out of a node
  - Uses N^2 memory
- I tend to only use them for all-pairs, shortest path

```java
import java.io.*;
import java.util.*;

public class arrays {
  public static int[][] dist = new int[5][5];

  public static void main(String[] args) {
    for(int i=0;i<5;i++)
      for(int j=0;j<5;j++) {
        if(i != j)
          dist[i][j] = 1000; // Not using Integer.MAX_VALUE to avoid integer overflowing
      }
    //Initialize graph as described
    dist[0][1] = 1;
    dist[0][2] = 3;
    dist[1][3] = 8;
    dist[2][3] = 2;
    dist[3][4] = 1;
  }
}
```



Dedicated classes

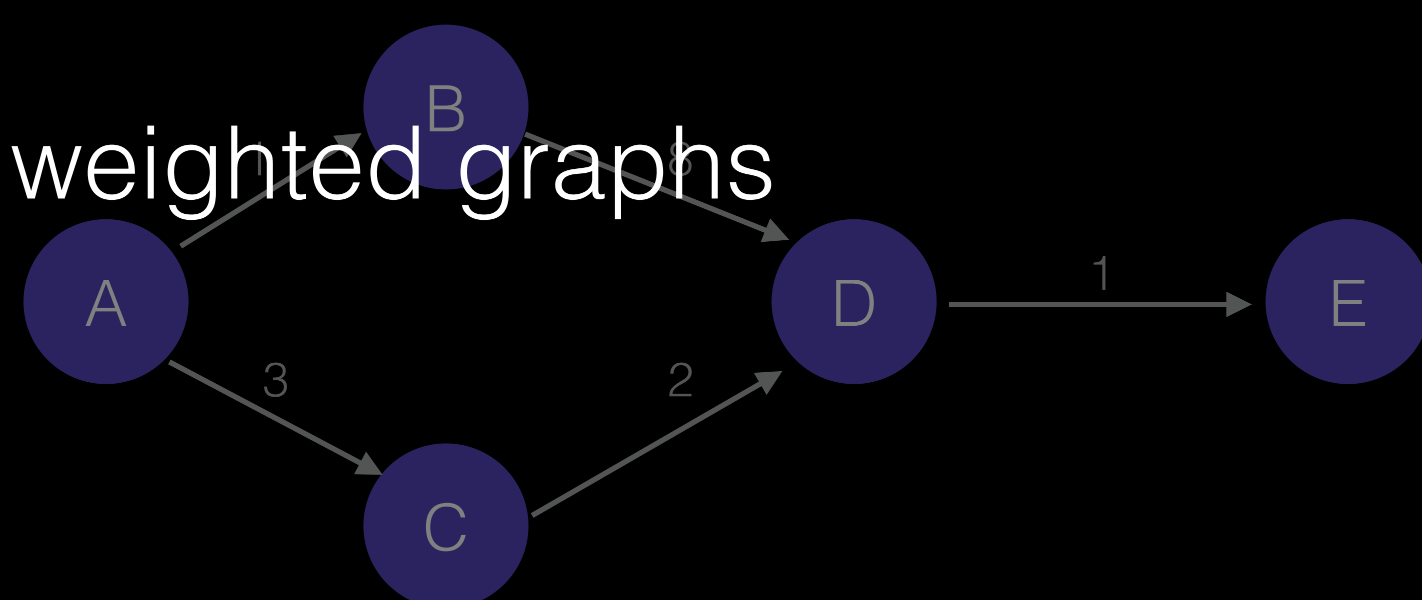
```
class Node {
    public ArrayList<Edge> neighbors = new ArrayList<Edge>();
}

class Edge {
    public Node dest;
    public int distance;

    public Edge(Node dest, Node source, int distance) {
        this.dest = dest;
        this.distance = distance;
        source.neighbors.add(this);
    }
}
```



Dedicated classes

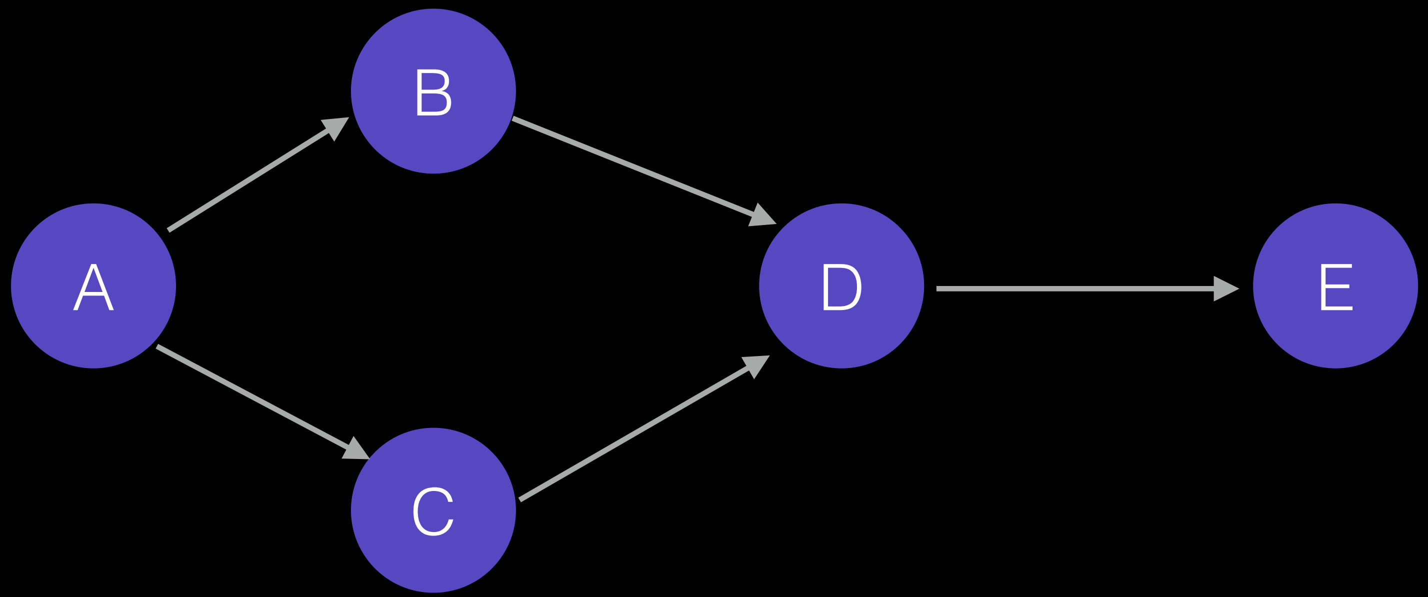
- Well-organized
- Adjacency Lists
    - Easy to get all edges out of a node
    - Difficult to find a particular edge
- This is what I use for weighted graphs

```java
class Node {
    public ArrayList<Edge> neighbors = new ArrayList<Edge>();
}

class Edge {
    public Node dest;
    public int distance;

    public Edge(Node dest, Node source, int distance) {
        this.dest = dest;
        this.distance = distance;
        source.neighbors.add(this);
    }
}
```



Unweighted Graphs

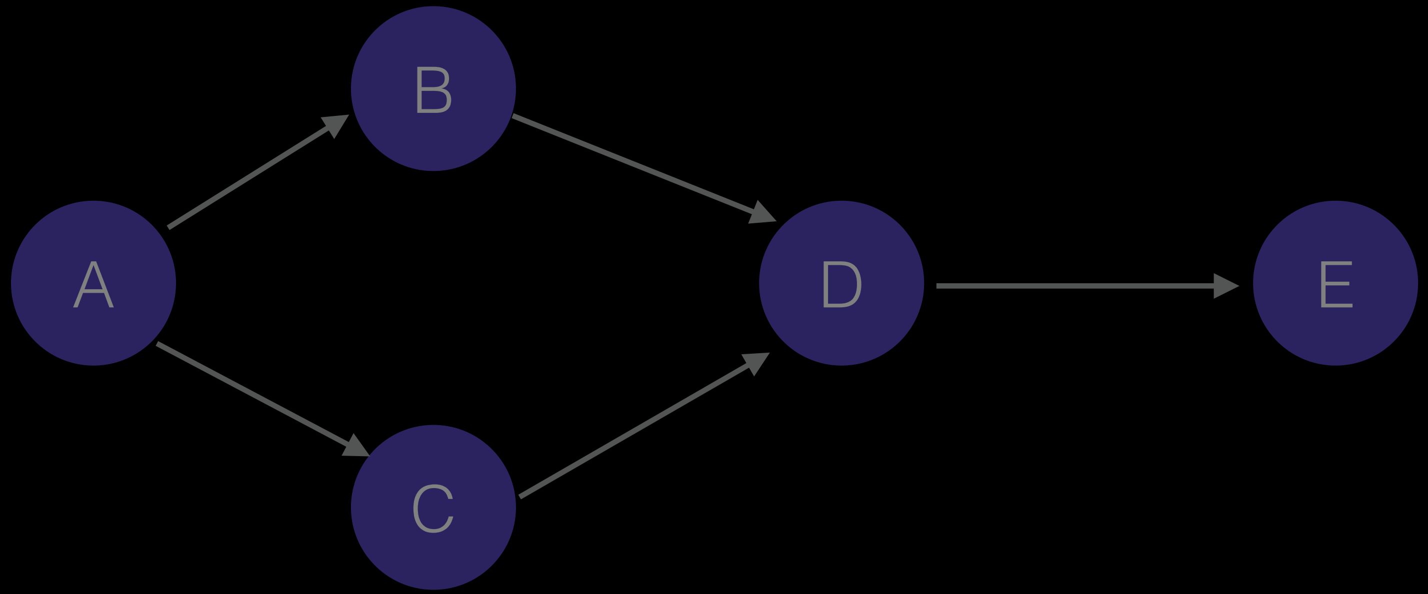
```
class Node {
    public ArrayList<Node> neighbors = new ArrayList<Node>();
}
```



Unweighted Graphs

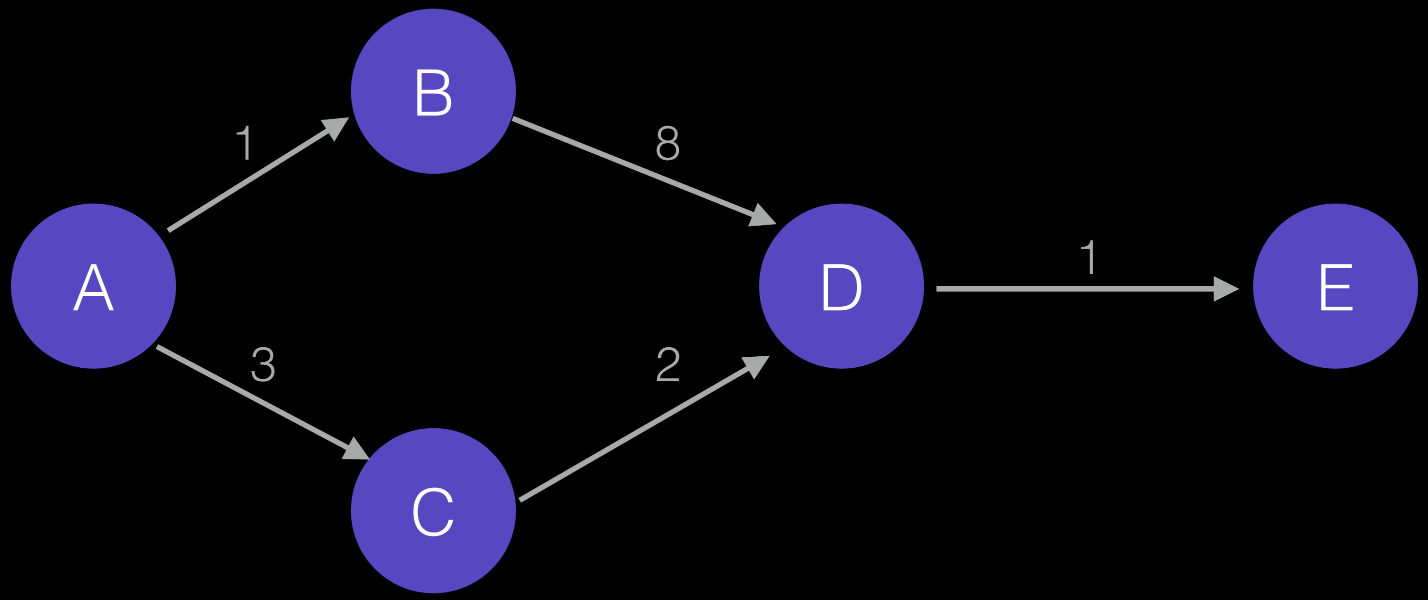
- Easy. You don’t need an edge class!

```
class Node {
    public ArrayList<Node> neighbors = new ArrayList<Node>();
}
```



HashMaps

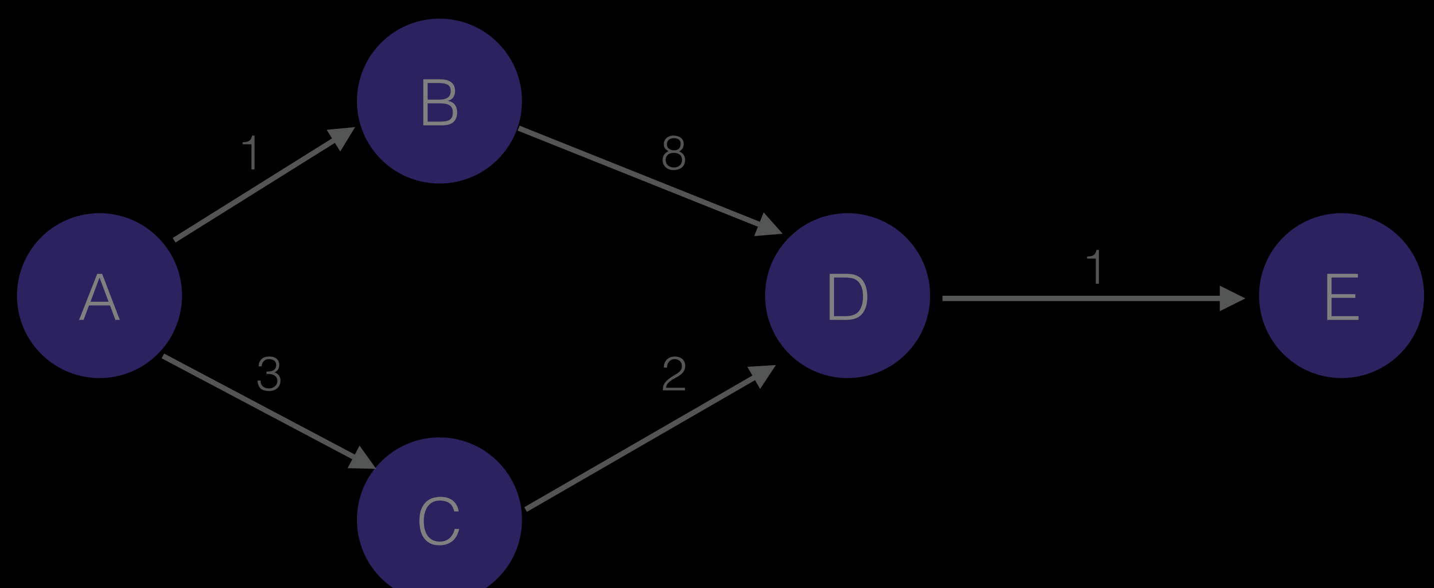
```
//For a weighted graph
class Node {
   public HashMap<Node, ArrayList<Node, Integer>> neighbors = new HashMap<Node,
ArrayList<Node, Integer>>();
}

//For a weighted graph, naming the nodes with Strings
public HashMap<String, HashMap<String, Integer>> neighbors = new HashMap<String,
HashMap<String, Integer>>();

//For an unweighted graph, naming the nodes with Strings
public HashMap<String, ArrayList<String>> neighbors = new HashMap<String,
ArrayList<String>>();
```



HashMaps

- A little harder to organize, debug
- But if you get used to it, can write shorter code

```java
//For a weighted graph
class Node {
   public HashMap<Node, ArrayList<Node, Integer>> neighbors = new HashMap<Node, ArrayList<Node, Integer>>();
}

//For a weighted graph, naming the nodes with Strings
public HashMap<String, HashMap<String, Integer>> neighbors = new HashMap<String, HashMap<String, Integer>>();

//For an unweighted graph, naming the nodes with Strings
public HashMap<String, ArrayList<String>> neighbors = new HashMap<String, ArrayList<String>>();
```



Graph Algorithms

- Searches (Breadth-First, Depth-First)
- Shortest Path (Dijkstra's)
- Minimum Spanning Tree (Prim's, Kruskal's)
- Topological Sort
- Negative-Edge Shortest Path (Bellman-Ford)
- All-Pairs, Shorest Path (Floyd-Warshall)
- Max Flow (Ford-Fulkerson, Edmonds-Karp, Preflow Push)
- Minimum-Cost Flow

Graph Algorithms

- Searches (Breadth-First, Depth-First)
- Shortest Path (Dijkstra’s)
- Minimum Spanning Tree (Prim’s, Kruskal’s)
- Topological Sort
- Negative-Edge Shortest Path (Bellman-Ford)
- All-Pairs, Shorest Path (Floyd-Warshall)
- Max Flow (Ford-Fulkerson, Edmonds-Karp, Preflow Push)
- Minimum-Cost Flow

Graph algorithms in their many shapes and sizes

Daniel Epstein, 5/8/14
depstein@cs.washington.edu
https://github.com/depstein/programming-competitions