CS 340 Project 2

Due 11:59 PM Tuesday September 26

Project 2

Implement the methods in the UndirectedGraph class shown on the following slides. The class implements an adjacency list representation of an undirected graph and the breadth first search algorithm. The vertices in the adjacency list representation are stored in a singly linked list **without a** sentinel node. The edges for each vertex (what I have referred to as edge list 1 and edge list 2) are stored in singly linked lists **without a** sentinel node.

The list of vertices is stored in **ascending order by name**.

Edge list 1 is stored in **ascending order using the name of the second vertex** in the edge.

Edge list 2 is stored in **ascending order using the name of the first vertex** in the edge.

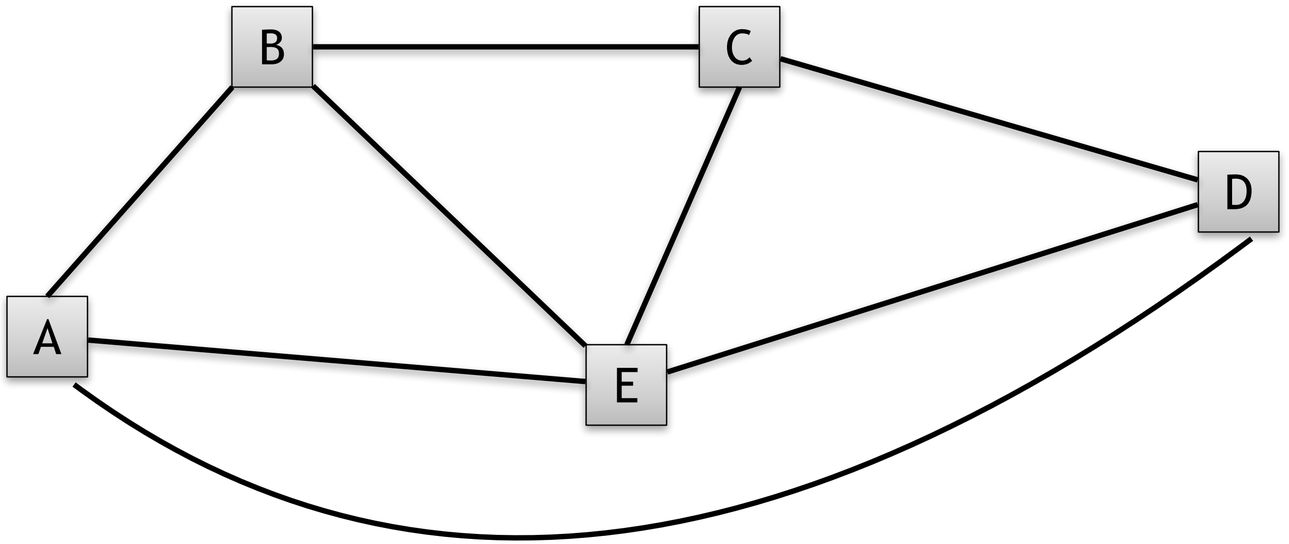
Remember there is only one EdgeNode for each edge. Each EdgeNode is part of 2 lists (edge list 1 and edge list 2)

You can add private methods and instance variables. For each private method include a comment explaining what the method does and for each private instance variable include a comment explaining the value stored in the variable.

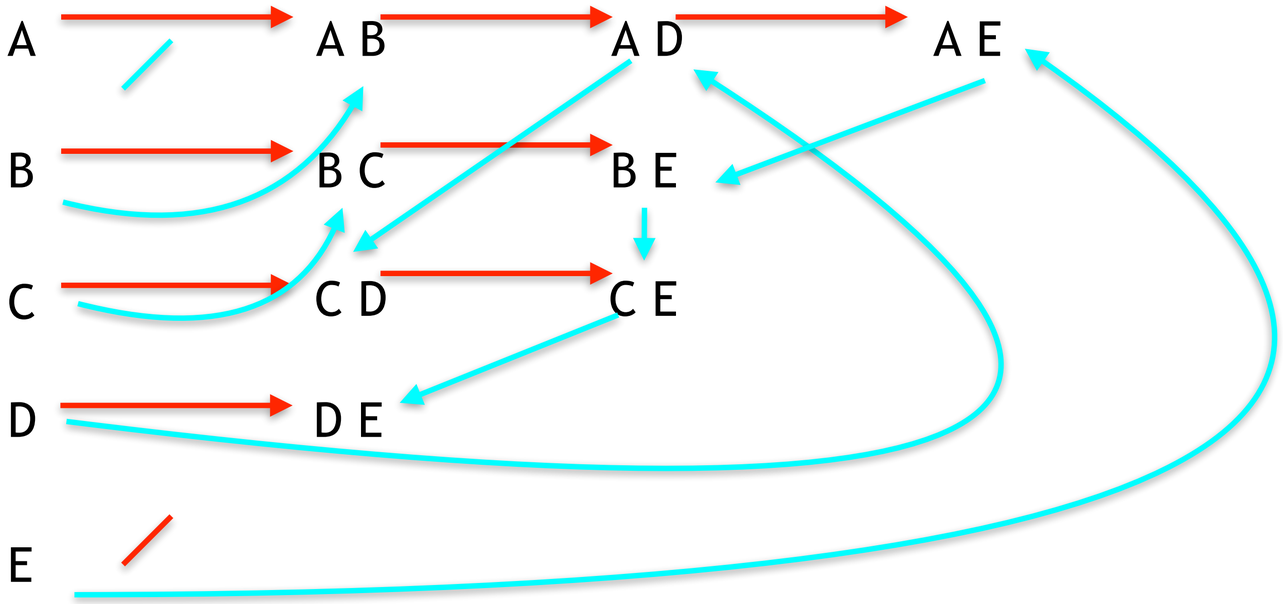
At the top of the java file put a comment with your name in it.

In this project you must not use any Java list implementations to implement the vertex list or the edge lists. You must implement the vertex list and the edge lists using the vertex node and edge node classes shown on the following slides. You can use a Java class for the FIFO Queue needed in the bfs.

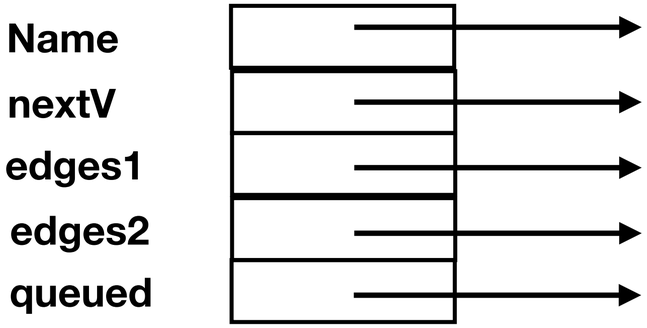
Undirected Graph



Adjacency List
Undirected Graph



VertexNode



Space for the reference to the name

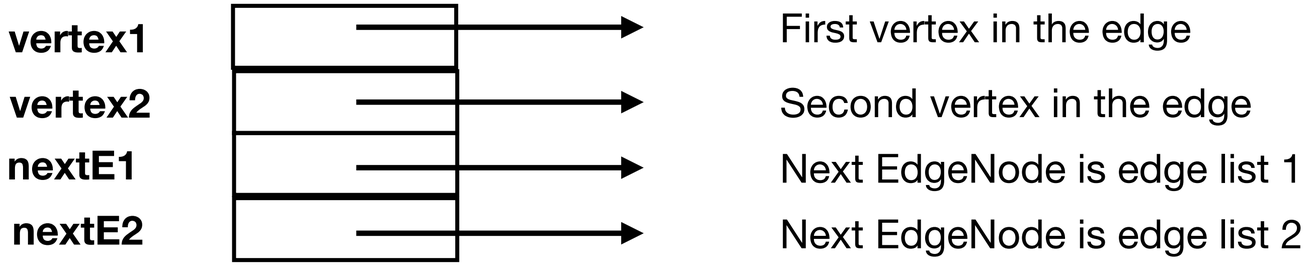
Next VertextNode in the list of vertices

Head of edge list 1 for this vertex

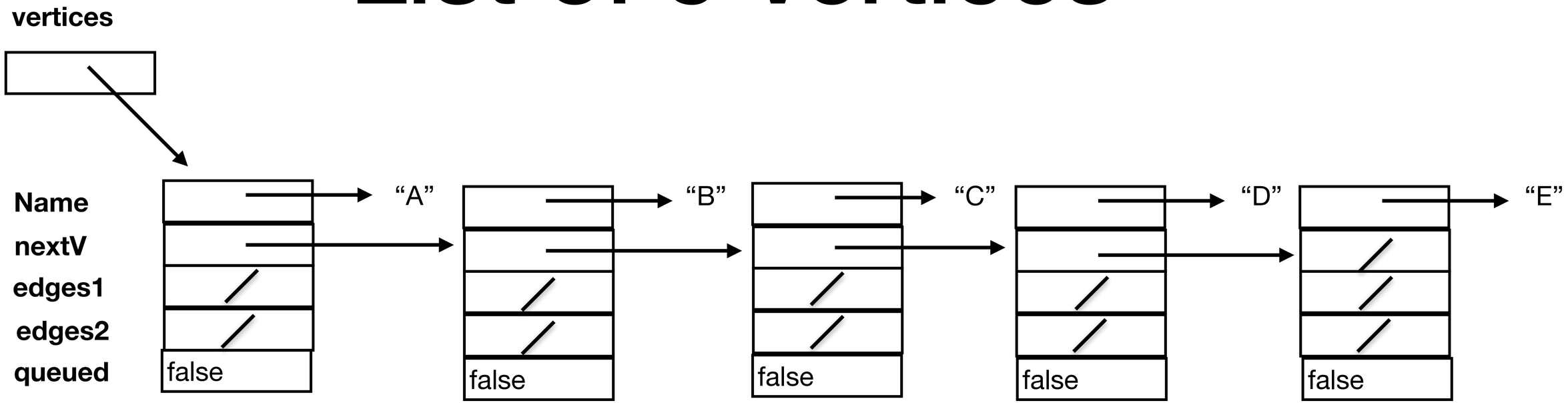
Head of edge list 2 for this vertex

boolean value used in Breadth First Search (BFS)

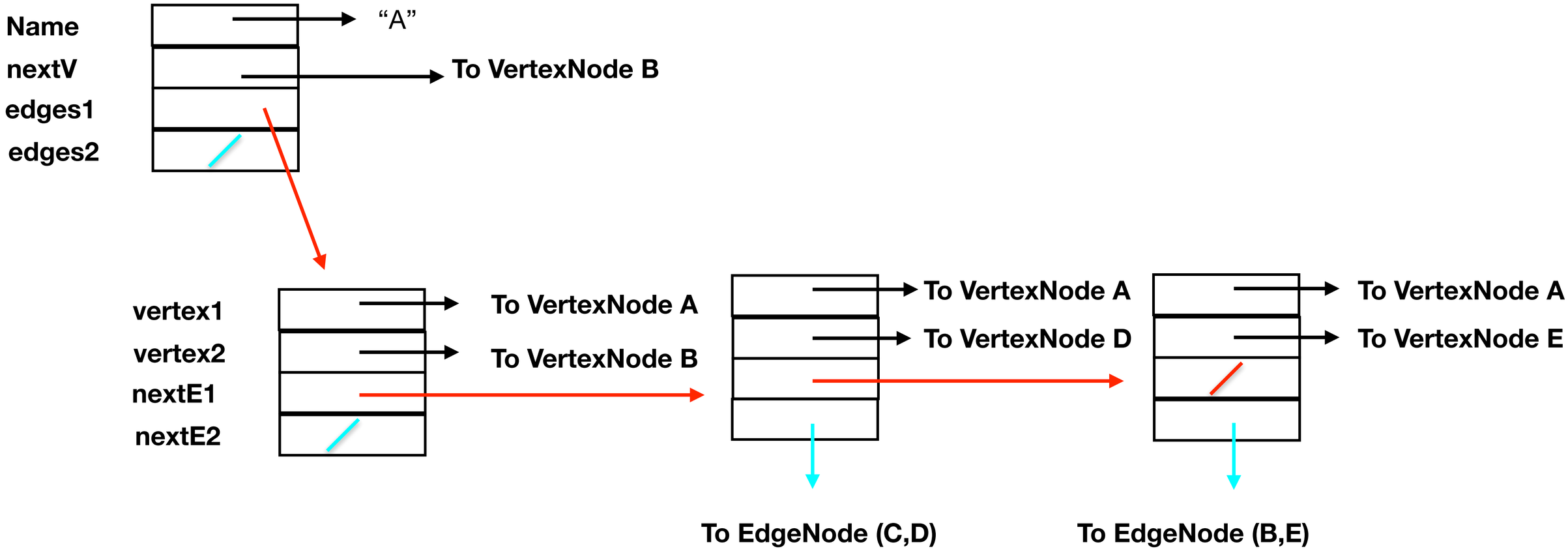
EdgeNode


vertex1
vertex2
nextE1
nextE2
First vertex in the edge
Second vertex in the edge
Next EdgeNode is edge list 1
Next EdgeNode is edge list 2

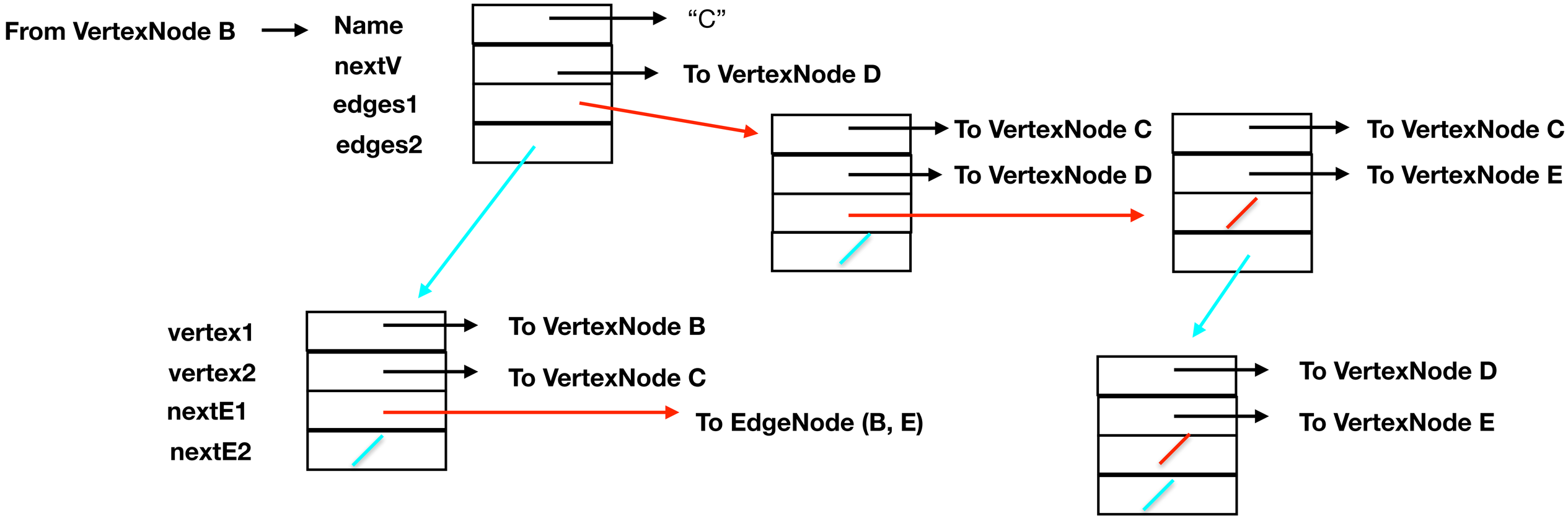
List of 5 Vertices



Detailed Partial View of Adjacency Lists



Detailed Partial View of Adjacency Lists



Undirected Graph

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

public class UndirectedGraph {

  private class VertexNode {
    private String name;
    private VertexNode nextV; //next vertex in the vertex list
    private EdgeNode edges1; //head of the edge list 1 for this vertex
    private EdgeNode edges2; //head of the edge list 2 for this vertex
    private boolean queued;


    private VertexNode(String n, VertexNode v) {
      name = n;
      nextV = v;
      edges1 = null;
       edges2 = null;
       queued = false;
    }
  }
```

Undirected Graph

```
private class EdgeNode {
  private VertexNode vertex1; //first vertex in the edge
  private VertexNode vertex2; //second vertex in the edge
                              //vertex1.name < vertex2.name
  private EdgeNode nextE1; //next edge in edge list 1
  private EdgeNode nextE2; //next edge in edge list 2

  private EdgeNode(VertexNode v1, VertexNode v2) {
  //PRE: v1.name < v2.name
    vertex1 = v1;
    vertex2 = v2;
    nextE1 = null;
     nextE2 = null;
  }
  private void setNextE1(EdgeNode e) {
    nextE1 = e;
  }
  private void setNextE2(EdgeNode e) {
    nextE2 = e;
  }
}
```

Undirected Graph

```
private VertexNode vertices; //head of the vertex list

public UndirectedGraph() {
  vertices = null; //the vertex list is initially empty(no sentinel node is used)
  numVertices = 0;
}
public void addVertex(String s) {
   //PRE: The vertex list is sorted in ascending order (based on the name)
   /* insert a new vertex into the vertex list and
    keep the vertex list sorted in ascending order (based on the name)
   */

}
```

Undirected Graph

```
public void addEdge(String n1, String n2) {
/*PRE: the vertices with names n1 and n2 have already been added
          and the edge list 1 for lesser of n1 and n2 is sorted in
          ascending order based on the greater of n1 and n2
          and the edge list 2 for the greater of n1 and n2
          is sorted in ascending order based on the lesser of n1 and n2
 */
 /*  add the new edge to the adjacency list and keep the edge lists sorted
     see the diagrams on the previous slides for more details
 */

}
```

Undirected Graph

```
public String bfs(String v) {
 /* if a breadth first search beginning with a vertex with name v contains all
    vertices in the graph then return a string of the vertex names (separated by a space) in
    a breadth first search order otherwise return the empty string.
 */

 }
```

BFS Algorithm

Create a FIFO queue
Mark all vertices as not queued (queued = false)
Add the starting vertex to the queue and for the starting vertex set queued = true
while the queue is not empty
    let x be the vertex removed from the queue
    add x to the bfs ordering
    for each vertex y adjacent to x where y has not been queued
        add y to the queue and set queued = true for y

    increment the number of items in the bfs by 1

If the number of vertices in the bfs equals the number of vertices in the graph return the bfs
otherwise return the empty string //this is not the only choice but is the simple one we will make now

Undirected Graph

```
public void printGraph() {
   /*print the graph.
     each line should contain a vertex, n, following
     by the vertices adjacent to the n as shown in the
     example on the following slide
   */

}
```

Undirected Graph

```
Node: A Edges1: B D E   Edges2:
Node: B Edges1: C E   Edges2: A
Node: C Edges1: D E   Edges2: B
Node: D Edges1: E   Edges2: A C
Node: E Edges1:   Edges2: A B C D
```

Undirected Graph

```
  public static void main(String args[]) throws IOException{
   //this code assumes the syntax of the import file is correct
    BufferedReader b = new BufferedReader(new FileReader(args[0]));
    UndirectedGraph g = new UndirectedGraph();
    String line = b.readLine();
    Scanner scan = new Scanner(line);
    while (scan.hasNext()) {
      g.addVertex(scan.next());
    }
    line = b.readLine();
    while (line != null) {
      scan = new Scanner(line);
      g.addEdge(scan.next(), scan.next());
      line = b.readLine();
    }
    g.printGraph();
    System.out.println("BFS Starting at " + args[1] + ": " + g.BFS(args[1]));
    System.out.println("BFS Starting at " + args[2] + ": " + g.BFS(args[2]));

  }
}
```

Project 2 Submission

- Upload one zip file to Canvas. The zip file must contain **only one file called UndirectedGraph.java**. Do not upload your whole Eclipse project!
- I have included a simple test driver to help you test your code. I could use a different test driver.
- **This project must be submitted on time.**