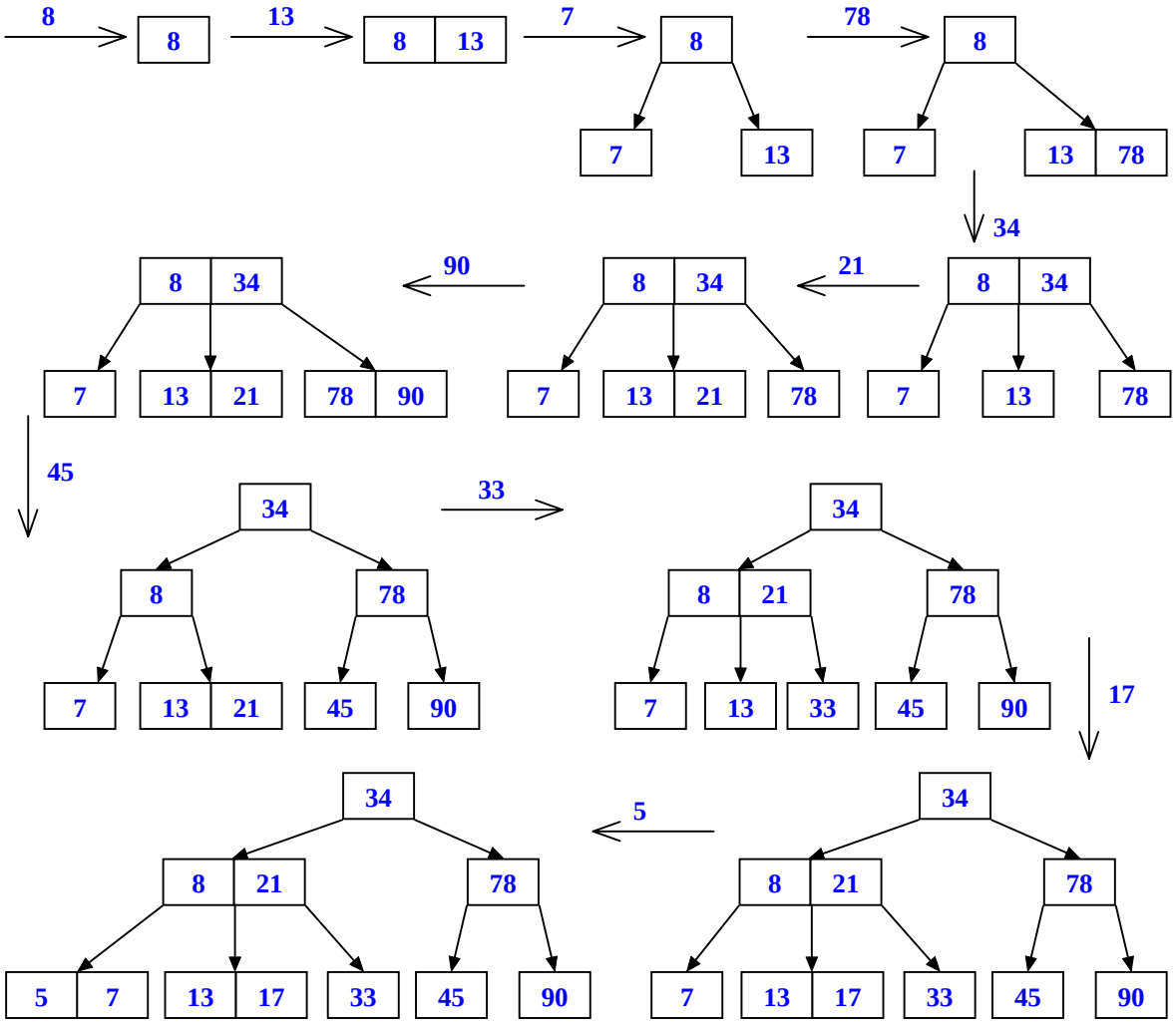
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
Information and Computer Science Department
ICS202: Data Structures
HOMEWORK 3 (Term 062)
Due Date: Monday, May 21st, 2007

Question#1 [16 points:]

(a) [10 Points] Draw the final B-Tree and the intermediate trees that results in inserting the following keys in an initially empty B-tree of **order 3** following the given sequence:

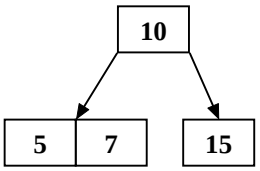
8, 13, 7, 78, 34, 21, 90, 45, 33, 17, 5

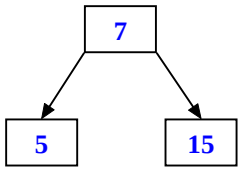
Grading: 1 point for each correct insertion, except the first one.



(b) [6 Points: 3 + 3] What will be the resulting B-Tree when the indicated operation is performed on the given B-Tree:

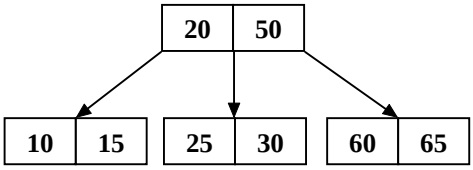
i. Delete key **10** from the following B-Tree of **order 3**.

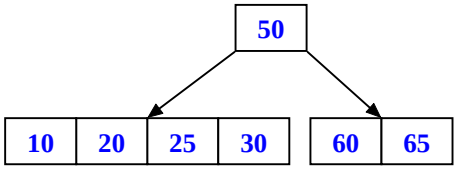




If **10** is swapped with its successor and then deleted, an underflow will happen and a right rotation will be needed. If **10** is swapped with its predecessor and then deleted, there will be no underflow. In both cases the resulting B-Tree will be the same.

ii. Delete key **15** from the following B-Tree of **order 5**.





Deleting 15 will result in an underflow. A rotation is not possible. The underflow node should be merged with the right node. The root node does not underflow after merging.

Question 2 [20 points: 8 + 3 + 3 + 3 + 3] Huffman Coding

A textfile **x** contains the following message:

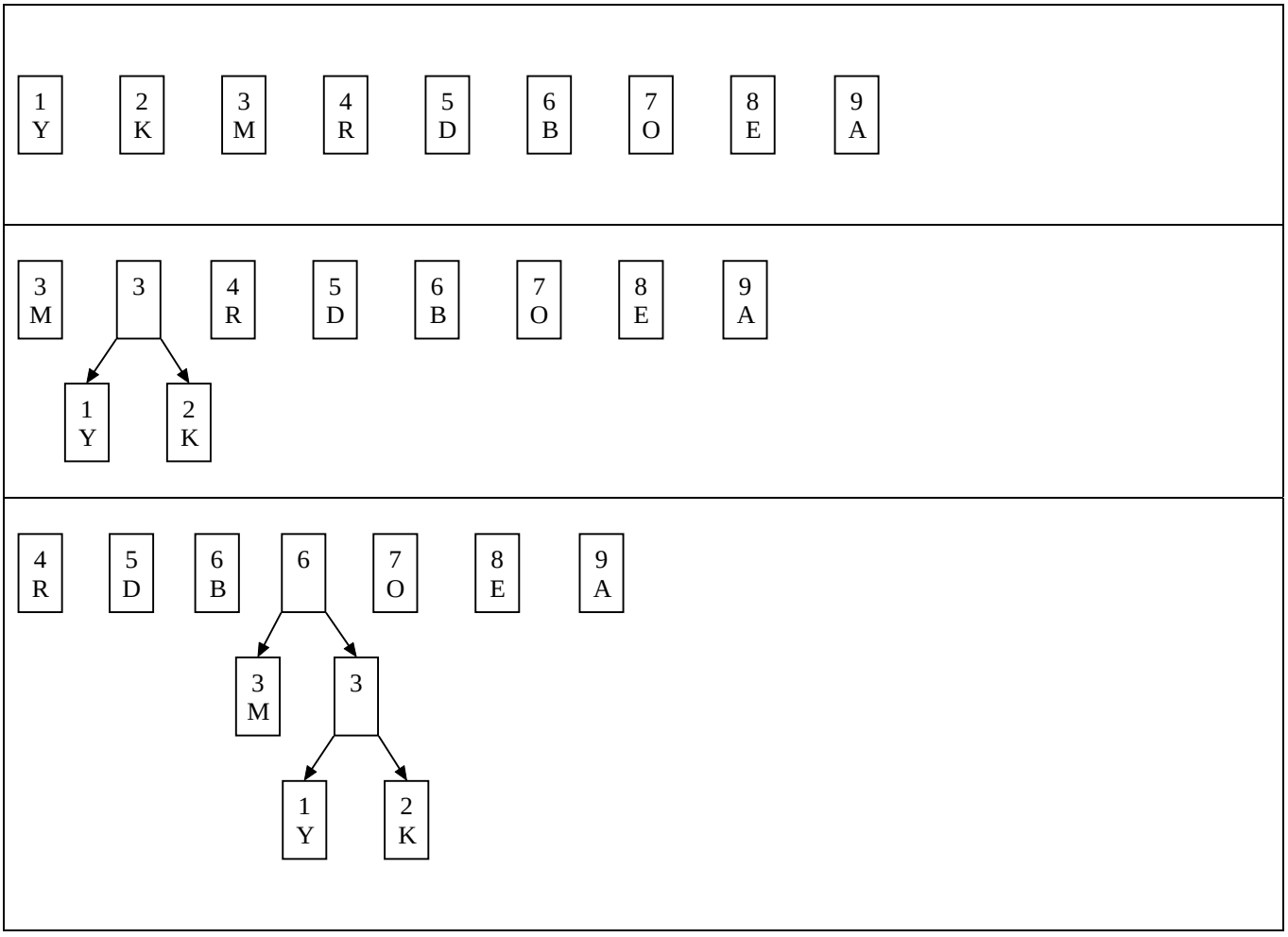
AAAAAAAAAYEEEEEEEEKKOOOOOOORRRRMMMBBBBBBDDDDD

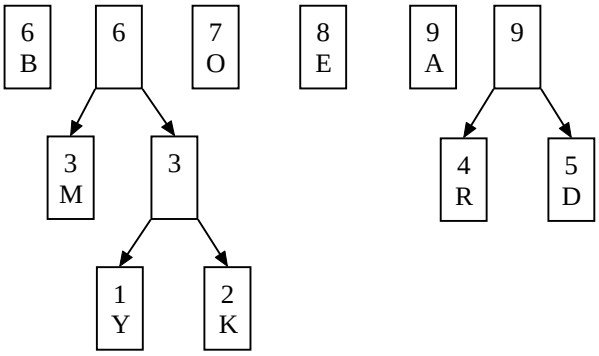
(a) Draw a Huffman code tree for encoding the textfile. **Show all intermediate and the final tree and the way these trees are arranged in a MinHeap priority queue.**

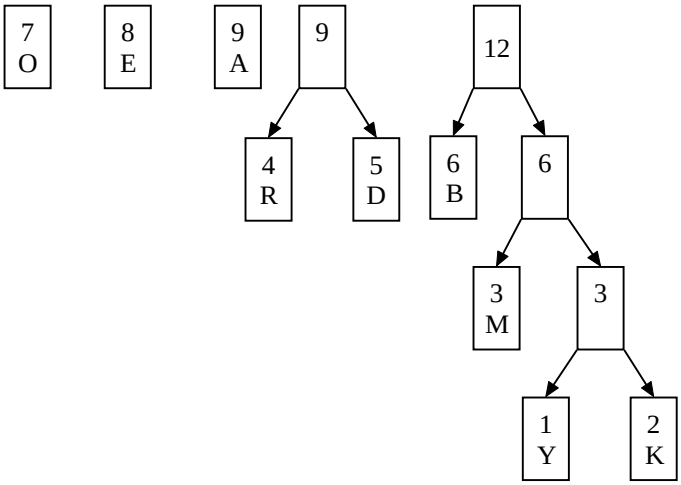
Note: Create a unique Huffman code tree by using the following guidelines:

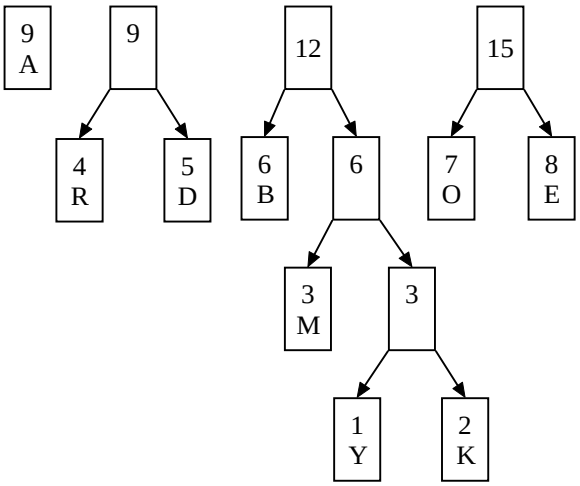
1. If two or more initial one-node trees have the same frequency; arrange them in the MinHeap priority queue in alphabetically increasing order.
2. Whenever two trees are dequeued from the priority queue; the first dequeued tree becomes the left subtree of the merged tree, and the second dequeued tree becomes the right subtree.
3. A merged tree **t** must be inserted **after** all trees in the MinHeap priority queue with the same frequency as that of **t**.
4. The left edge of each node in the final tree is assigned **0** and the right edge of each node is assigned **1**

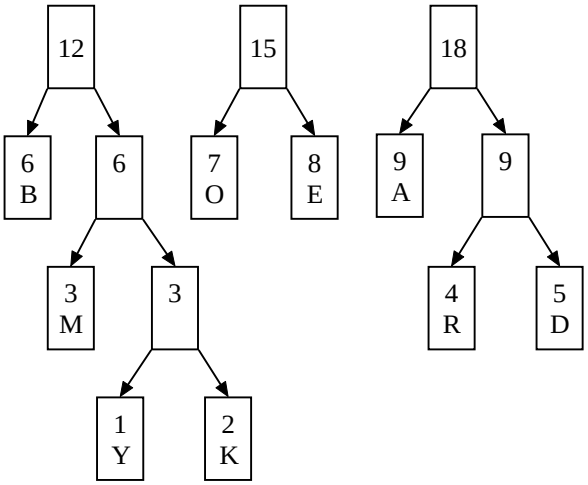
Note: NO GRADE WILL BE GIVEN FOR THIS QUESTION IF THESE 4 GUIDELINES ARE NOT FOLLOWED

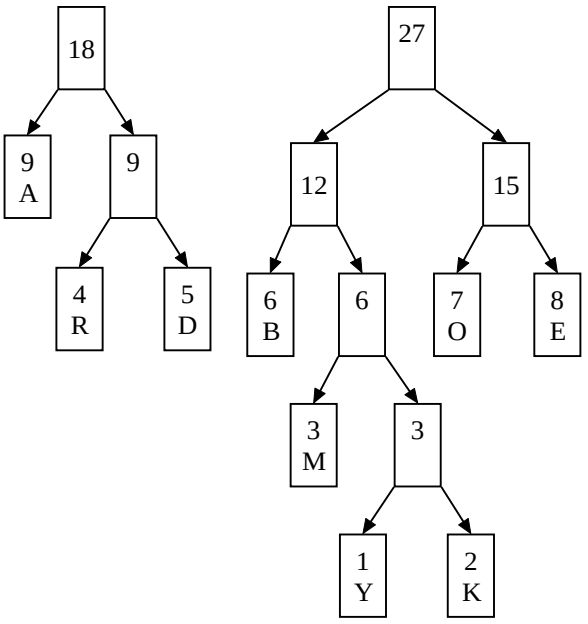


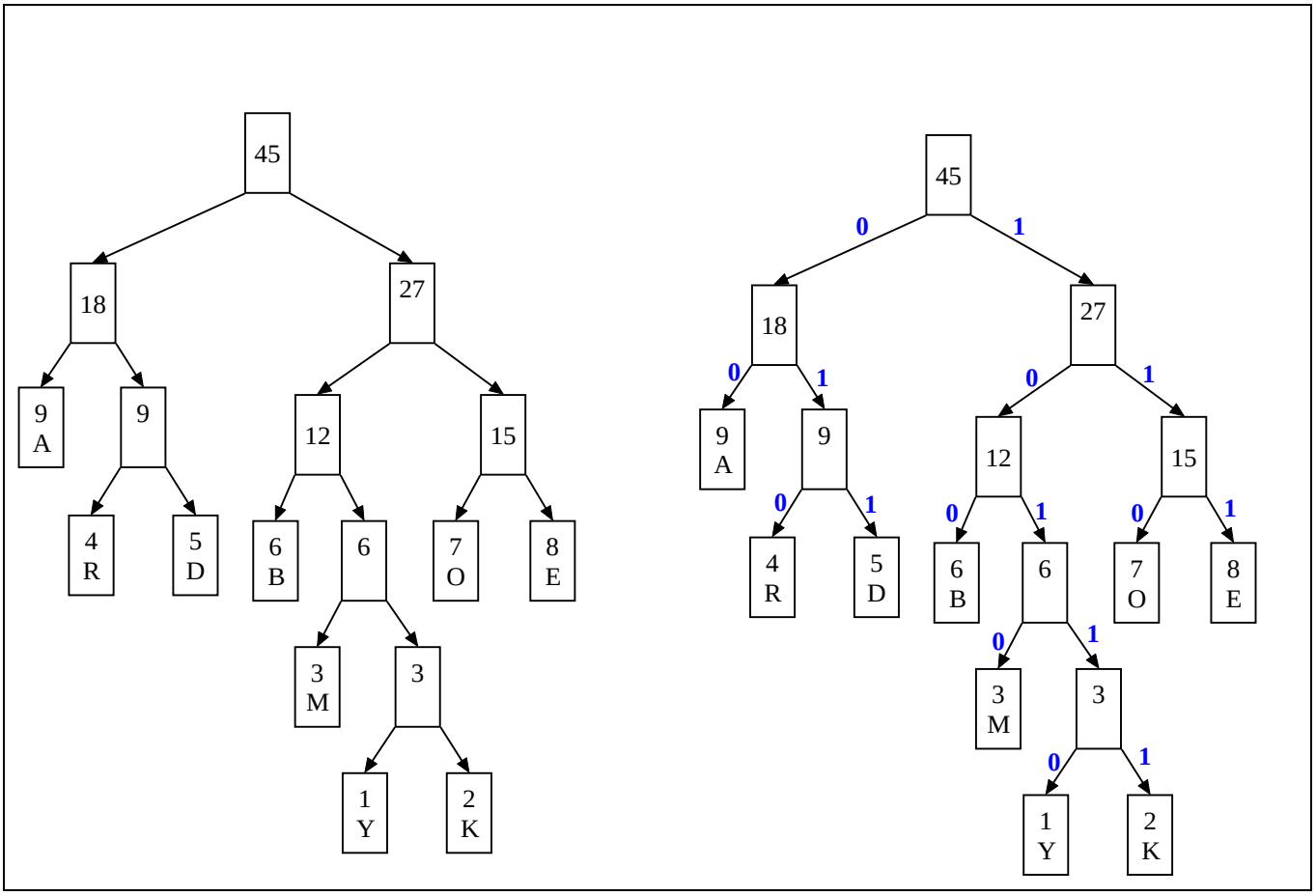

6
B
6
7
O
8
E
9
A
9
3
M
3
4
R
5
D
1
Y
2
K


7
O
8
E
9
A
9
12
4
R
5
D
6
B
6
3
M
3
1
Y
2
K


9
A
9
12
15
4
R
5
D
6
B
6
7
O
8
E
3
M
3
1
Y
2
K


12
15
18
6
B
6
7
O
8
E
9
A
9
3
M
3
4
R
5
D
1
Y
2
K


18
27
9
A
9
12
15
4
R
5
D
6
B
6
7
O
8
E
3
M
3
1
Y
2
K



(b) Write the Huffman codeword of each character in a table similar to the following:

character	Huffman codeword
A	**00**
B	**100**
D	**011**
E	**111**
K	**10111**
M	**1010**
O	**110**
R	**010**
Y	**10110**

(c) Encode the message **READBOOK**

010111000111001101110111

(d) Decode the message **101010110111010010110010** if possible.

MYERROR

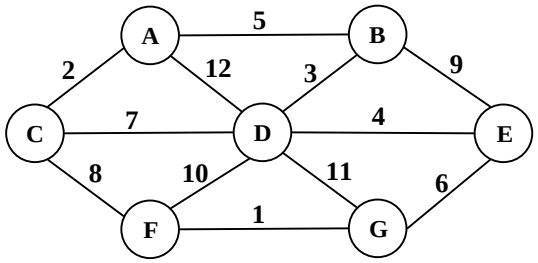
(e) Calculate the number of bits to transmit the encoded file **x**.

character	A	B	D	E	K	M	O	R	Y
codeword	00	100	011	111	10111	1010	110	010	10110
bits	2	3	3	3	5	4	3	3	5
frequency	9	6	6	8	2	3	7	4	1
Bits*frequency	18	18	18	24	10	12	21	12	5

Number of bits transmitted = Sum of bits * frequency = 128

Question 3 [13 points: 6 + 7] Graph Traversals and Tracing

Consider the following graph:



(a) List the vertices in the order they will be visited by each of the following traversal methods. If at any point you have more than one vertex to visit, you must visit the vertices in increasing alphabetic order.

i. [2 points] Pre-order Depth-First Traversal starting at vertex A:

A, B, D, C, F, G, E

ii. [2 points] Post-order Depth-First Traversal starting at vertex B:

B, A, C, D, E, G, F

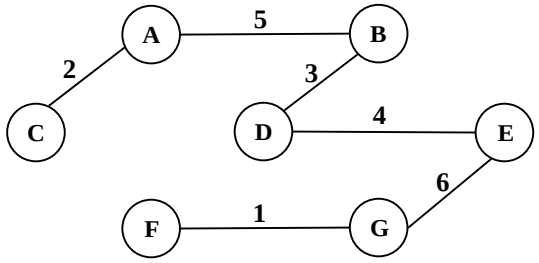
iii. [2 points] Breadth-First Traversal starting at vertex G:

G, D, E, F, A, B, C

(b) [6+1 points] Use the above graph to trace the execution of Prim's algorithm as it finds the minimum-cost spanning tree starting from vertex C. Draw the resulting minimum-cost spanning tree. Use the following table for tracing Prim's algorithm.

Pass:	initial	1	2	3	4	5	6	7	weight	Predecessor
Active vertex:		**C**	**A**	**B**	**D**	**E**	**G**			
A	∞	2							**2**	**C**
B	∞	∞	5						**5**	**A**
C	0								**0**	**-**
D	∞	7	7	3					**3**	**B**
E	∞	∞	∞	9	**4**				**4**	**D**
F	∞	8	8	8	8	8	**1**		**1**	**G**
G	∞	∞	∞	∞	11	**6**			**6**	**E**

The resulting minimum-cost spanning tree is:



Question 4 [34 points: 10 + 20 + 4] Graph Implementation

The **GraphAsArrayLists** class uses the concept of adjacency list to represent a graph. If you have done Lab#10 (Graphs), you will have complete implementation of the **GraphAsArrayLists** class.

*Note: If you have not done Lab#10, make sure that your **GraphAsArrayLists** class implements all the abstract methods. Otherwise your class will not compile.*

In this question, you need to make a copy of the **GraphAsArrayLists** class and add the following methods to it**.**

(a) public void deleteEdge(String from, String to)

[10 points] This method removes the edge between the vertices whose labels are passed in as Strings **from** and **to**. If such an edge does not exist, the method should throw an appropriate exception. If the graph is undirected, the method should also remove the reverse of the edge.

```
public void deleteEdge(String from, String to){//total points = 10
      Edge edge = this.getEdge(from, to);//1 point
      if (edge == null)// 1 point
            throw new IllegalArgumentException("No such edge");//1 point
      int index = getIndex(from);//1 point
      edges[index].extract(edge);//3 points
      /*Instead of calling extract method as above,
       * the student may write a loop to find the edge in the linked list
       * and then remove it by adjusting the next reference of the
       * previous element.
       * This approach is also acceptable
       */
      this.numberOfEdges--;// 1 points
      // delete the reverse edge as well for undirected graph
      if (!this.isDirected() && this.getEdge(to, from)!=null)//1 points
            this.deleteEdge(to, from);// 1 points
}
```

(b) public void deleteVertex(String label)

[20 points] This method removes the vertex whose label is passed in. If such a vertex does not exist, the method should throw an appropriate exception. The method should also remove all emanating and all incident edges for the deleted vertex.

```
	public void deleteVertex(String label){//total points = 20
		int vertexIndex = getIndex(label);// 1 point
		if (vertexIndex == -1) // 1 point
			throw new IllegalArgumentException("No such vertex");//1 point
		//count emanating edges so that edges count can be adjusted later
		int emCount = 0;
		MyLinkedList.Element e = edges[vertexIndex].getHead();
		while (e!=null){
			emCount++;
			e = e.next;
		}// 2 points for the above part

		//shifting all higher index vertices and adjacency lists
		for (int i = vertexIndex; i < this.numberOfVertices - 1; i++){// 2
point
			//this will delete the vertex
			vertices[i] = vertices[i+1];// 1 point
			// this will delete the emanating edges
			edges[i] = edges[i+1];// 1 point
		}
		this.numberOfVertices--;// 1 point
		this.numberOfEdges -= emCount;// 1 point
		vertices[numberOfVertices] = null;// 1 point
		edges[numberOfVertices].purge();// 1 point
		// deleting the incident edges
		for (int i = 0; i < this.numberOfVertices; i++){// 1 point
			e = edges[i].getHead();// 1 point
			while (e!=null){// 1 point
				Edge eg = (Edge) e.getData();// 1 point
				if (eg.getToVertex().getLabel().equals(label)){// 1 point
					e.extract();// 1 point
					// or edges[i].extract(eg);
					numberOfEdges--;// 1 point
				}
			}
		}
	}
```

(c) public static void main(String[] args)

[4 points] This method should test the above two methods. Create an object of **GraphAsArrayLists** class and populate it with some vertices and edges. Remove few vertices and edges. Print the graph at various stages to see the effect of the above methods.

No specific solution.

Question 5 [17 points: 5 + 2 + 10] Graph Application

Think about a realistic problem which can be represented as finding ***Shortest-Path*** problem or finding ***Minimum-Cost Spanning Tree*** problem. You will have to introduce the problem and solve it with one of the graph algorithms discussed in the lecture.
Example problems are:

- Creating a minimum-cost railway or vehicle road network that connects all major cities of a particular country.
- Using an existing railway, bus or airline network of stations, finding the minimum in cost, shortest in distance, or fastest in time rout to reach from one place to another.
- Or you may choose any other problem …

Do the followings for the problem:

(a) [5 points] Describe the problem with the help of a pictorial graph. Your graph must have at least 7 vertices and as many edges.

(b) [2 points] Represent the graph as a text file so that it can be read by your program (as done in the lab).

(c) [10 points] Write a java class **MyGraphProblem** which contains the **main** method (and other methods if needed). In the **main** method, a graph object must be initialized by reading the text file created in the previous step. Then extra information is collected from the user if needed (e.g., name of the city the user wants to start traveling from/to). The method then calls an appropriate method from the **ics202.Algorithms** class to find a solution. It then prints the result properly.

No specific solution.

Important Notes:

- Your report for this homework must be **word-processed** and must follow the **homework submission template** format, which you can get in the downloadables section of the WebCT.
- Diagrams can be drawn using a tool (Visio, SmartDraw, etc) or by hand.
- All the classes for this homework must be stored in a package **`ics202.hw04`.**
- You must import the necessary packages needed for your program.
- You need to submit two things:
    1. A printed copy of your report at the beginning of your class on the due date.
    2. Submit your entire **`ics202`** package into the webCT under the Assignments option.