Assignment 4 – Tree & Graph

Deadline: 23:59, May 23

If you miss the deadline but submit the assignment before 23:59, May 25, you'll get a 50% penalty. If you submit later than May 25, you'll get 0.

Problem 1: Binary Tree

Function Requirements:

1. Create a class ***BinaryTree*** with fundamental methods such as AddLeft(), AddRight(), etc. The binary tree should be built by adding nodes one by one.

2. Implement 4 traversal methods: pre-order, in-order, post-order, and level-order. The return value of these methods should be ***String***.

3. Build a binary tree by the given pre-order and in-order Strings, and then return the post-order traversal result. Similarly, build a binary tree by the given post-order and in-order Strings, and then return the pre-order traversal result.

```java
public class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int x){
        val = x;
        left = null;
        right = null;
    }
}

public class BinaryTree {
    public TreeNode root;

    public boolean AddLeft(TreeNode parent, TreeNode left){}
    public boolean AddRight(TreeNode parent, TreeNode right){}
    public String TraversalInOrder(){}
    public String TraversalPreOrder(){}
    public String TraversalPostOrder(){}
    public String TraversalLevelOrder(){}
    public String PreIn2Post(String preTraversal, String inTraversal){}
    public String InPost2Pre(String inTraversal, String postTraversal){}
}
```

Problem 2: Dijkstra’s Shortest Path Algorithm

Function Requirement:

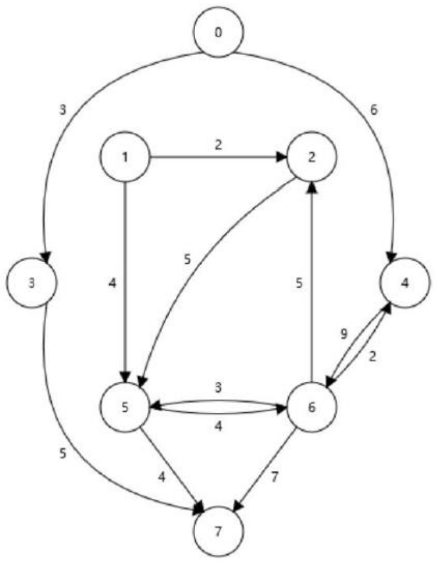
1. Create a class ***Graph*** to represent a graph.
2. Method readGraphFile(String path): The input could be either directed or undirected graph. The input data of the graph will be stored in a **file**. The first line contains exactly one integer that is 1 or 0. 1 means that this graph is directed, while 0 means an undirected graph. The following lines will be all edges of the graph, and the format of each line should be “origin destination weight” (e.g., 0 3 3 means that the edge start from vertex 0 and point to vertex 3 with a weight of 3).
3. Print out adjacency list and adjacency matrix by methods printAdjacencyList() and printAdjacencyMatrix().
4. Print out the shortest paths starting from one given vertex to all other vertices, including the total weights of the paths, by method shortestPath(int startVertex). Also, print out the shortest path starting from a given vertex and pointing to another vertex, including the total weight of the path, by method shortestPath(int startVertex, int endVertex).

Special cases:

1. The indexes of the vertices include all non-negative integers starting from 0 and ending at the largest integer in the first two columns of the input file. The weight of each edge should be a positive integer.
2. The graph may not be fully connected. However, we guarantee that there will be at least an edge connected with each vertex.
3. The output format should be as same as the format shown in the given test cases. See ***GraphTest.java***.
4. We guarantee that there will not be more than one edge connecting two vertices in an undirected graph, and for a directed graph, there will not be any edges that both start from vertex v_1 and point to vertex v_2.

E.g. 1

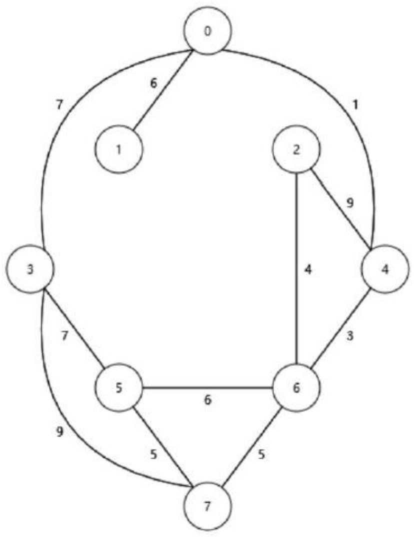
```
1
0 3 3
0 4 6
1 2 2
1 5 4
2 5 5
3 7 5
4 6 9
5 6 4
5 7 4
6 2 5
6 5 3
```



Vertex	Known	Cost	Path	
0	T	0	-1	0
1	F	INF	-1	No Path
2	T	20	6	0 4 6 2
3	T	3	0	0 3
4	T	6	0	0 4
5	T	18	6	0 4 6 5
6	T	15	4	0 4 6
7	T	8	3	0 3 7

E.g. 2

```
0
0 1 6
0 3 7
0 4 1
2 4 9
2 6 4
3 5 7
3 7 9
4 6 3
5 6 6
5 7 5
6 7 5
```



Vertex	Known	Cost	Path	
0	T	0	-1	0
1	T	6	0	0 1
2	T	8	6	0 4 6 2
3	T	7	0	0 3
4	T	1	0	0 4
5	T	10	6	0 4 6 5
6	T	4	4	0 4 6
7	T	9	6	0 4 6 7

```java
public class Graph {

    public void readGraphFile(String strFile) throws IOException {}
    public String printAdjacencyList(){}
    public String printAdjacencyMatrix(){}
    public String ShortestPath(int startVertex){}
    public String ShortestPath(int startVertex, int endVertex){}
}
```

Notice:

1. Please do not change any codes in the given classes ***BinaryTree*** and ***Graph***, and all methods should be written as exactly same as the example shown above, including their names, parameters, and return types. You can add new methods to this class, but do not change any given methods or attributes.

2. Part of the test cases will be given to you, but we will use more test cases to grade your submissions.

3. This assignment needs more work and is more difficult compared to "MaxProfit" and "Rain", please start to do it as early as possible.

4. If you failed to implement a method, please write an empty method into your code to avoid compile errors. Besides, before you submit your assignment, please have a look at the import part of your codes in case there are some strange packages that may lead to a compile error. Note that any compile error will lead to a 0 grade.

5. Since our test cases will be written by JUNIT and the test environment will be JDK 1.8, please try to avoid using new features later than Java 8. We'll try to change these new features to some usable codes, while if we fail to do so, you'll get a zero because this is a compile error.

6. Please submit all files that your program has to use, and compress them into a **zip** or an **rar** package, which should have a same structure as the structure of "***assignment4.zip***". You need to declare "`package problem1`" or "`package problem2`" at the top of the **Java** files.

7. Please avoid being a "buzzer beater" and try to submit your assignment at least 3 minutes earlier than the deadline, in case there is a difference between the time of Blackboard System and your device's time. The submission time will only depend on the time of the Blackboard System.

April 30th, 2021