CS 2210a — Data Structures and Algorithms
Assignment 5

Due Date: December 7, 11:55 PM
Total marks: 20

1 Overview

For this assignment you need to write a program that finds a path between two specified locations in a roadmap that satisfies certain conditions specified below. The roadmap has 3 kinds of roads: *public* roads, *private* roads that have a fee to be used, and *construction* roads that are under repair, so the traffic through them is slow.

Your program will receive as input a file with a description of the roadmap, the starting point s, the destination e, the maximum number p of private roads that the path from s to e can use, and the maximum number c of construction roads that the path from s to e can use. Your program then will try to find a path from s to e as specified.

You must store the roadmap as an undirected graph. Every edge of the graph represents a road and every node represents either the intersection of two roads or the end of a road. There are two special nodes in this graph denoting the starting point s and the destination e. A modified depth first search traversal, for example, can be used to find a path as required, if one exists.

2 Classes to Implement

You are to implement at least four Java classes: `Node`, `Edge`, `Graph`, and `MyMap`. You can implement more classes if you need to, as long as you follow good program design and information-hiding principles.

You must write all code yourself. You cannot use code from the textbook, the Internet, other students, or any other sources. However, you are allowed to use the algorithms discussed in class.

For each one of the classes below, you can implement more private methods if you want to, but you cannot implement additional public methods.

2.1 Node

This class represent a node of the graph. You must implement these public methods:

- `Node(int id)`: This is the constructor for the class and it creates a node with the given id. The id of a node is an integer value between 0 and $n - 1$, where n is the number of nodes in the graph.
- `markNode(boolean mark)`: Marks the node with the specified value, either `true` or `false`. This is useful when traversing the graph to know which nodes have already been visited.
- `boolean getMark()`: Returns the value with which the node has been marked.
- `int getId()`: Returns the id of this node.

2.2 Edge

This class represents an edge of the graph. You must implement these public methods:

- `Edge(Node u, Node v, String type)`: The constructor for the class. The first two parameters are the endpoints of the edge. The last parameter is the type of the edge, which for this assignment can be either "public" (if the edge represents a public road), "private" (if the edge represents a private road), or "construction" (if the edge represents a road with construction work on it).
- `Node firstNode()`: Returns the first endpoint of the edge.
- `Node secondNode()`: Returns the second endpoint of the edge.
- `String getType()`: Returns the type of the edge.

2.3 Graph

This class represents an undirected graph. You **must use an adjacency matrix or an adjacency list** representation for the graph. For this class, you must implement all the public methods specified in the provided `GraphADT` interface plus the constructor. Hence, this class must be declared as follows:

```
public class Graph implements GraphADT
```

The public methods in this class are described below.

- `Graph(int n)`: Creates a graph with `n` nodes and no edges. This is the constructor for the class. The ids of the nodes are $0, 1, \ldots,$ `n`-1.
- `Node getNode(int id)`: Returns the node with the specified `id`. If no node with this id exists, the method should throw a `GraphException`.
- `addEdge(Node u, Node v, String edgeType)`: Adds an edge of the given type connecting `u` and `v`. This method throws a `GraphException` if either node does not exist or if in the graph there is already an edge connecting the given nodes.
- `Iterator incidentEdges(Node u)`: Returns a Java Iterator storing all the edges incident on node `u`. It returns null if `u` does not have any edges incident on it.
- `Edge getEdge(Node u, Node v)`: Returns the edge connecting nodes `u` and `v`. This method throws a `GraphException` if there is no edge between `u` and `v`.
- `boolean areAdjacent(Node u, Node v)`: Returns *true* if nodes `u` and `v` are adjacent; it returns *false* otherwise.

The last four methods throw a `GraphException` if `u` or `v` are not nodes of the graph.

2.4 MyMap

This class represents the roadmap. A graph will be used to store the map and to try to find a path from the starting point to the destination satisfying the conditions stated above. For this class you must implement the following public methods:

- `MyMap(String inputFile)`: Constructor for building a graph from the input file specified in the parameter; this graph represents the roadmap. If the input file does not exist, this method should throw a `MapException`. Read below to learn about the format of the input file.
- `Graph getGraph()`: Returns the graph representing the roadmap.
- `int getStartingNode()`: Returns the id of the starting node (this value and the next three values are specified in the input file).

- `int getDestinationNode()`: Returns the id of the destination node.
- `int maxPrivateRoads()`: Returns the maximum number allowed of private roads in the path from the starting node to the destination.
- `int maxConstructionRoads()`: Returns the maximum number allowed of construction roads in the path from the starting node to the destination.
- `Iterator findPath(int start, int destination, maxPrivate, maxConstruction)`: Returns a Java Iterator containing the nodes of a path from the `start` node to the `destination` node such that the path uses at most `maxPrivate` private roads and `maxConstruction` construction roads, if such a path exists. If the path does not exist, this method returns the value `null`. For example for the roadmap described below the Iterator returned by this method should contain the nodes `0, 1, 5, 6,` and `10`.

3 Implementation Issues

3.1 Input File

The input file is a text file with the following format:

```
SCALE
START
END
WIDTH
LENGTH
PRIVATE_ROADS
CONSTRUCTION_ROADS
RHRHRH···RHR
HXHXHX···HXH
RHRHRH···RHR
HXHXHX···HXH
⋮
RHRHRH···RHR
```

Each one of the first seven lines of the input file contains one number:

- `SCALE` is the scale factor used to display the map on the screen. Your program will not use this value. If the map appears too small on your screen, you must increase this value. Similarly, if the map is too large, choose a smaller value for the scale.
- `START` is the starting node.
- `END` is the destination node.
- `WIDTH` is the width of the map. The roads of the map are arranged in a grid. The number of vertical roads in each row of this grid is the width of the map.
- `LENGTH` is the length of the map, or the number of horizontal roads in each column of the grid. So the total number of nodes in the corresponding graph is `WIDTH` $\times$ `LENGTH`.
- `PRIVATE_ROADS` is the maximum allowed number of private roads in the path from `START` to `END`.
- `CONSTRUCTION_ROADS` is the maximum allowed number of construction roads in the path from `START` to `END`.

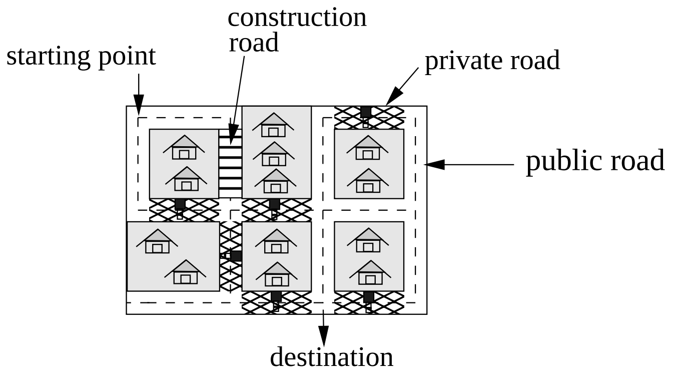
For the rest of the input file, `R` can be any of the following characters: '+' or 'B'. `H` could be 'B', 'V', 'C', or 'P'. The meaning of the above characters is as follows:

- '+': denotes either and intersection of two roads, or a dead end
- 'V': denotes a private road
- 'P': denotes a public road
- 'C': denotes a construction road
- 'B': denotes a block of houses, a building, or a park

Each line of the file (except the first eight lines) must have the same length because, as mentioned above, the only roadmaps that we will consider are grid roadmaps. Here is an example of an input file:

```
30
0
10
4
3
1
1
+P+B+V+
PBCBPBP
+V+V+P+
BBVBPBP
+P+V+V+
```

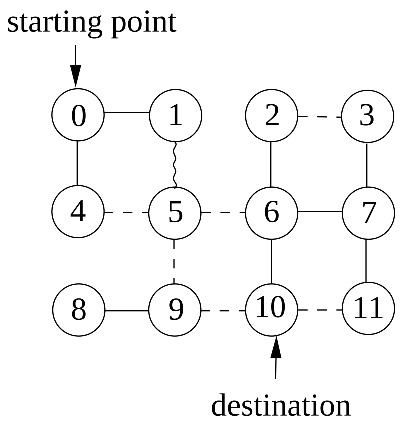
This input represents the following roadmap



For the above example, a path from node with id 0 to node with id 10 is sought that uses at most one private and one construction roads.

3.2 Graph Construction

The roadmap is represented as a graph in which node ids are numbered consecutively, starting at zero from left to right and top to bottom. For example, the above roadmap is represented with this graph:



where dotted edges represent private roads, solid edges represent public roads, and wavy edges represent construction roads. In the `MyMap` class you need to keep a reference to the starting and destination nodes.

3.3 Finding a Path

Your program must find **any** path from the starting vertex to the destination vertex that uses at most the specified number of private and construction roads. The path can use any number of public roads. If there are several such paths, your program might return any one of them.

The solution can be found, for example, by using a modified DFS traversal. While traversing the graph, your algorithm needs to keep track of the vertices along the path that the DFS traversal has followed. If the current path has already used all allowed private roads, then no more private-road edges can be added to it. Similarly, if the current path has already used all allowed construction roads, no more construction-road edges can be added to the path.

For example, consider the above graph and let the maximum allowed number of private roads be 1 and the maximum allowed number of construction roads be 1. Assume that the algorithm visits vertex 0, then 4, and then 5. As the algorithm traverses the graph, all visited vertices get marked. While at vertex 5, the algorithm cannot next visit vertices 6 or 9, since then two private roads would have been used by the current path. Hence, the algorithm goes next to vertex 1. However, the destination cannot be reached from here, so the algorithm must go back to vertex 5, and then back to vertices 4 and 0. Note that vertices 1, 5 and 4 must be unmarked when the algorithm steps back, as otherwise the algorithm will not be able to find a solution. Next, the algorithm will move from vertex 0 to vertex 1, and then to 5. Since edge $(1, 5)$ represents a construction road, the current path has used 1 construction road and no private roads. The algorithm can then move to node 6 and then to 10 as the rest of the path uses only one private road. Therefore, the solution produced by the algorithm is: `0, 1, 5, 6,` and `10`.

You do not have to implement the above algorithm if you do not want to. Please feel free to design your own solution for the problem.

4 Code Provided

You can download from the course's website the following files: `TestGraph.java`, `DrawMap.java`, `Board.java`, `Path.java`, `GraphADT.java`, `GraphException`, `MapException`, and several image files. The `main` method is in class `Path.java` , so to run your program you will type

```
java Path input_file
```

You can also type

```
java Path input_file delay
```

where `delay` is the number of milliseconds that the program will wait before drawing the next edge of the solution.

You can use `TestDict.java` to test your implementation of the `Graph.java` class. You can also download from the course's website some examples of input files that we will use to test your program.

5 Hints

Recall that the java class `Iterator` is an interface, so you cannot create objects of type `Iterator`. The methods provided by this interface are `hasNext(), next()`, and `remove()`. An Iterator can be obtained, for instance, from a `Vector` or `Stack` object by using the method `iterator()`. For example, if your algorithm stores the path from the starting node to the destination in a `Stack S`, then an iterator can be obtained from `S` by invoking `S.iterator()`.

6 Coding Style

Your mark will be based partly on your coding style.

- Variable and method names should be chosen to reflect their purpose in the program.
- Comments, indenting, and white spaces should be used to improve readability.
- No unnecessary instance variables must be used in your code.
- All instance variables must be declared `private`, to maximize information hiding.

7 Marking

Your mark will be computed as follows.

- Program compiles, produces meaningful output: 2 marks.
- Tests for the `Graph` class: 4 marks.
- Tests for the `MyMap` class: 4 marks.
- Coding style: 2 marks.
- `Graph` implementation: 4 marks.
- `MyMap` implementation: 4 marks.

8 Handing In Your Program

You must submit an electronic copy of your program through OWL. Please **DO NOT** put your code in sub-directories. Please **DO NOT** submit a .zip file or other compressed file containing your code; submit the individual .java files.

You can submit your program more than once if you need to. We will take the latest program submitted as the final version, and will deduct marks accordingly if it is late.