COMS 311: Homework 3
Due: Oct 20, 11:59pm
Total Points: 100

Late submission policy. If you submit by Oct 21, 11:59PM (and after the specified deadline), there will be 10% penalty. That is, if your score is x points, then your final score for this homework after the penalty will be $0.9 \times x$. Similarly, if you submit by Oct 22, 11:59PM (after Oct 21, 11:59PM), there will be 20% penalty.

Submission after Oct 22, 11:59PM will not be graded without (prior) explicit permission from the instructors.

Points. Total points for this homework assignment is 100pts. There will be extra credit test cases worth 20pts. In other words, you may be able to score 120pts out of 100pts.

Learning outcomes.

Design, implement and evaluate algorithms following specifications.

Basic Graph exploration algorithms

0 Preamble

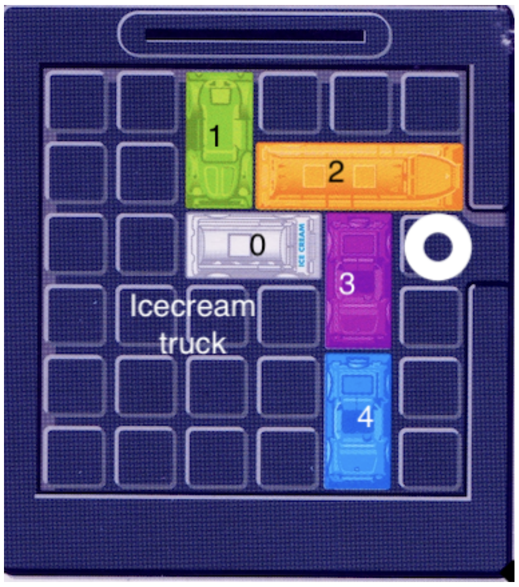
Description of a programming assignment is not a linear narrative and may require multiple readings before things start to click. You are encouraged to consult instructor/Teaching Assistants for any questions/clarifications regarding the assignment. Your programs must be in Java.

1 Problem Description

Scarlet Overkill developed a simple strategy game to keep the minions busy. The game includes a square grid gameboard with 6 rows and 6 columns. The grid is partially occupied by several (toy) vehicles. Consider such a grid in Figure 1. There are 5 vehicles and one of the vehicles is an an icecream truck (white truck numbered 0). One of the grid locations is an *escape location* (marked with white circle).

The rules of the game are as follows:

- Vehicles can only move along either east-west or north-south depending on the way they are placed on the grid. For instance, the vehicles 1, 3 and 4 can move north (up) or south (down); while the vehicles 0 and 2 can move east (right) or west (left).
- Vehicles cannot move over another vehicles.



The objective of the game is to move the vehicles in such a way inside the grid such that the icecream truck can reach the escape location. One can realize the objective by

- move vehicle 0 west by two grid locations (2 moves)
- move vehicle 1 south by three grid locations (3 moves)
- move vehicle 2 west by two grid locations (2 moves)
- move vehicle 3 north by two grid locations (2 moves)
- move vehicle 0 east all the way to the grid location marked by white circle (escape location) (4 moves).

This is called a plan. The length of a plan is computed as a sum of the number moves. For instance, for the above plan the number of moves is 13 moves.

You are "requested" by Scarlet Overkill to write a program, which can take as input some initial setup and output a *shortest* plan (if one exists) that will realize the game objective. (Note that, there can be many shortest plans.)

2 Encoding Description

The grid locations are numbered 1 to 36, where the first row contains the grid locations $1, 2, 3, 4, 5, 6$ (from left to right); second row contains the grid locations $7, 8, 9, 10, 11, 12$ and so on. **The escape location is** 18.

The input to your program is a simple text file containing some numbers (integers). The first line will indicate the number (n) of vehicles in the gameboard. This is followed by n

lines where each line contains either 2 or 3 numbers (in ascending order and separated by one or more spaces) indicating the grid locations covered by a vehicle. The first of these n lines contains the grid locations for the icecream truck. The remaining lines contain the grid locations of vehicles in order of vehicle number. For instance, example grid set up in Figure 1 is presented as

```
5
15 16
3  9
10 11 12
17 23
29    35
```

You can assume that the text file will be correctly formatted and will contain semantically meaningful information. For example, there will be no numbers less than 1 or greater than 36 for any grid location; none of the vehicles will cover less than 2 or more than 3 grid locations; none of the vehicles will cover grid locations that are not horizontally or vertically placed. *In short, you do not need to worry about the input file being meaningless (no input validation is needed).*

You are required to implement the following classes and methods.

1. A class `GameBoard` with default constructor. It must contain the following methods:
   (a) `public void readInput(String FileName) throws IOException`
   The objective of the method is to read the input initial game setup from a text file (its name is the parameter) and populate necessary attributes (as you see fit) in the object of type `GameBoard`.
   (b) `public ArrayList<Pair> getPlan()`
   The objective of the method is to compute and return a plan to realize game objective for the initial game setup. The plan is captured in `ArrayList<Pair>`, where the elements of the array list indicates the move of a specific vehicle, which, in turn, is captured as an object of type `Pair` (see below). The first attribute of the pair is the vehicle number and the second attribute is a character: 'e' (for east), 'w' (for west), 'n' (for north, 's' (for south).
   (c) `public int getNumOfPaths()`
   The objective of the method is to compute and return the number of shortest plans that can realize the game objective for the initial game setup.

2. A class `Pair` as follows:

```
class Pair {
        int id;
        char direction;  // {'e', 'w', 'n', 's'}
        public Pair(int i, char d) {  id = i; direction = d; }

        char getDirection() { return direction; }
        int getId() { return id; }

        void setDirection(char d) { direction = d; }
        void setId(int i) { id = i; }
}
```

You can add as many other classes, helper methods as you see fit for developing your solution.

3 Programming Requirements

1. You shall use default package (that is, no package at all). Though it is not recommended in practice for mid-to-large projects, organize your classes as follows:
2. Prepare a file `GameBoard.java`. In this file you will write the above specified classes and any other class that you used for develop your solution. For instance,

```
// import directives
import java.io.IOExceptions;
//

// primary class for this file
public class GameBoard {
   // attributes

   // specified methods
   // Any other helper methods
}

class Pair {
   // as described above
   // Any other helper attributes and methods
}

class <helperclass> {
...
}
```

3. Prepare another file with any name suitable for your setup which only contains the `main` method and declares the object of type `GameBoard` and invokes it methods. You can use the following sample file containing the main method (**there is no constraint on how/what you write in this file as this file will not be part of your submission**).

```
// appropriate import directives
public class HW {
   public static void main(String[] args) throws exception {

       GameBoard gb = new GameBoard();
       gb.readInput(args[0]);

       ArrayList<Pair> path = gb.getPlan();
       for (int i=0; i<path.size(); i++)
            System.out.println(path.get(i).getId() + " "
                                  + path.get(i).getDirection());
       System.out.println(gb.getNumOfPaths());
   }
}
```

Consider that the input for the initial setup (see Figure 1) is provided in a file `1.txt`, then for a correct implementation (assuming `GameBoard.java` and `HW.java` are in the same working directory as `1.txt`), the output for
`javac HW.java; java HW 1.txt` can be

```
0 w
0 w
1 s
1 s
1 s
0 e
0 e
2 w
2 w
3 n
3 n
0 e
0 e
35
```

Note that, there may be many shortest plans for the initial setup; you are required to compute just one of them.

4 Submission Requirement

You are required to submit `GameBoard.java` and nothing else. (Double check before submitting whether you have all the necessary helper classes in the submitted file).

5 Suggestions

For this problem, you may want to use hashing. The following is an example scenario. Consider that the element being hashed contains a key with some attribute (say `c`) of type `int[]` and some object of type `MyClass`. That is, in Java terms

```
  HashMap<HashKey, MyClass> keeping_track = new HashMap();
```

To effectively use such a HashMap, we will consider that attribute valuations for `c` uniquely identifies the object of type `MyClass`. The `HashKey` class, this case, will be defined as

```
class HashKey {
  int[] c;  // attribute

  public HashKey(int[] inputc) {
    c = new int[inputc.length];
    c = inputc;
  }


  public boolean equals(Object o) {
    boolean flag = true;
    if (this == o) return true; // same object

    if ((o instanceof HashKey)) {
       HashKey h = (HashKey)o;
       int[] locs1 = h.c;
       int[] locs = c;
       if (locs1.length == locs.length) {
           for (int i=0; i<locs1.length; i++) {
           // mismatch
              if (locs1[i] != locs[i]) {
                 flag = false;
                 break;
              }
           }
        }
        else   // different size
```

```
           flag = false;
    }
    else    // not an instance of HashKey
       flag = false;
       return flag;
  }

/*
 * (non-Javadoc)
 * @see java.lang.Object#hashCode()
 */
  public int hashCode() {
    return Arrays.hashCode(c); // using default hashing of arrays
  }
}
```

6 Some more examples

Consider the following initial set up

```
4
15 16
5 11  17
27  33
28 29 30
```

The output plan can be

```
0 w
0 w
1 s
2 n
2 n
2 n
2 n
0 e
0 e
3 w
3 w
1 s
1 s
0 e
0 e
```

For the initial set up

```
8
13 14
2 3 4
5 6
11 12
15 21
18 24
25 26 27
30 36
```

the output plan can be

```
1 w
2 w
3 w
5 n
5 n
6 e
6 e
7 n
7 n
6 e
4 s
0 e
0 e
0 e
4 n
6 w
7 s
0 e
```

For the initial setup

```
8
13 14
1 2 3
4 10
6 12
15 21
19 25 31
26 27 28
29 30
```

the output plan can be

```
2 s
1 e
1 e
2 s
3 s
1 e
4 n
4 n
0 e
5 n
5 n
6 w
2 s
0 e
0 e
4 s
1 w
3 n
0 e
```

Recall that there can be many shortest plan for an initial setup. Your computed solution may be correct and may not exactly match the solution presented above.

7 Postscript

1. You must follow the given specifications. Method names, classnames, return types, input types. Any discrepancy may result in lower than expected grade even if "everything works".

2. In the above problem, there are several data structure/organization that are left for you to decide. Please do not ask questions related to such data structure/organization. Part of the exercise is to understand and assess a good way to organize data that will allow effective application of methods/algorithms.

3. You will have to think about how to model the problem into a graph-based problem and apply your knowledge of graph algorithms to address the original problem. Please do not ask questions about how to model the problem as a graph-based problem and/or what graph algorithms to use. Part of the exercise is to understand and assess a good way to represent/reduce a problem to a known problem for which we know an efficient algorithm.

4. Start reading and sketching the strategy for implementation **as early as possible**. That does not mean starting to "code" without putting much thought on what to code and how to code it. This will also help in resolving all doubts about the assignment before it is too late. Early detection of possible pitfalls and difficulties in the implementation will help in reducing the `endTime-startTime` for this assignment.

5. Both correctness and efficiency are important for any algorithm assignment. Writing a highly efficient incorrect solution *will* result in low grade. Writing a highly inefficient correct solution *may* result in low grade. In most cases, the brute force algorithm is unlikely to be the most efficient. Use your knowledge from lectures, notes, book-chapters to design and implement algorithms that are correct and efficient.

6. Test your code extensively (starting with individual methods). Your submission will be assessed using test cases that are different from the ones provided as part of this assignment specification. Your grade will primarily depend on the number of these test cases for which your submission produces the correct result.