CS1110 Spring 2010 Assignment A6
Fibonacci Bees

Introduction

In A6, you will implement Danaus' methods learn() and run(). Method learn() will require you to exhaustively explore a graph —the map on which the butterfly flies. The map is easily viewed as a graph. The nodes are the tiles. An edge leads from a node to all the neighboring flyable tiles. Method run() will require you to design an algorithm to parsimoniously collect a set of flowers scattered about the graph.

Collaboration policy

You may do this assignment with one other person. If you are going to work together, then, as soon as possible —and certainly a few days before you submit the assignment— get on the CMS for the course and do what is required to form a group. Both people must do something before the group is formed: one proposes, the other accepts.

If you do this assignment with another person, *you must work together*. It is against the rules for one person to do some programming on this assignment without the other person sitting nearby and helping. You should take turns "driving" —using the keyboard and mouse.

With the exception of your CMS-registered partner, you may not look at anyone else's code, in any form, for any version of this assignment in this semester or previous ones, in any form. You may not show or give your code to another person in the class.

Getting help

If you don't know where to start, if you don't understand testing, if you are lost, etc., please SEE SOMEONE IMMEDIATELY —a course instructor, a TA, a consultant. Or, ask a question and look for answers on the course Piazza. Do not wait. A little in-person help can do wonders. See the course homepage for contact information.

Step 1. Method learn

public TileState[][] learn()

In A3, you implemented a boustrophedonic implementation of learn(). Now, implement learn() according to the method's full specification in AbstractButterfly.java. You will have to exhaustively explore the map —viewed as a graph— and return a TileState[][] array with a TileState for all flyable tiles. Implement learn() using Depth First Search. Study carefully the lecture slides on DFS before beginning.

Step 2. Method run

public void run(List<Long> flowerIds)

After a call on method learn() has completed, the simulator will populate the map with a few more flowers and then call your method run() with a list of flower ID's. Method run() must collect the flowers, using function collect(). It must not collect any other flowers. For more information, read the specification of run() in AbstractButterfly.java.

In A6, we impose an additional performance constraint on run(). For a map m with n flyable tiles, your implementation of run() must fly to fewer than n tiles on m —in other words, it must not visit all flyable tiles. This constraint is designed to motivate you to design a clever algorithm that takes advantage of the information you gathered in learn().

Testing

Unit testing is vital to verify the correctness of software. Unit testing code of any reasonable scale requires unit tests to be repeatable, scalable, and exhaustive. Thus far, your ability to test Danaus software has been limited. Aside from providing seeds to Danaus via -s <seed>, you did not know how to construct arbitrary maps and use them in testing testing (although you could have found out by reading the information about Danaus).

Danaus supports the construction and loading of arbitrary maps encoded in XML files. Maps are located in directory /res/maps. Consult tutorial.xml for a sample map file with a brief tutorial as comments. We have included other map files that constitute good test cases. However, a solution that correctly completes the provided maps does not guarantee a flawless solution. We encourage you to create your own unit tests. Consult Danaus' man pages or the Danaus manual for information on simulating on map files.

Here is a quick suggestion. To run using map1.xml, in the Run Configuration, put the program argument

```
-f res/maps/map1.xml
```

To do the same thing but without the GUI, use two program arguments:

```
-h  -f res/maps/map1.xml
```

Important suggestion

It will take time to get both learn() and run() right. You have more work than usual to do because you get to think about what has to be done and design your solution appropriately, using helper methods where necessary. It doesn't make sense to start on method run(..) until you absolutely *know* that method learn() is implemented correctly.

We suggest you complete testing of learn() by 25 April, so that you have time to enough time to work on run(..).

About the deadline for submission

The deadline of 6 May is firm. We don't expect to accept late submissions. Reason: We don't have too much time before the final (12 May) to do the grading of A6 and whatever else has to be done before the final.

Deliverables

Checklist

Before you submit your assignment, check to see that you have completed the following.

- The fields of Butterfly are annotated with the class invariant.
- Each method of Butterfly has a good javadoc specification.
- Your submission is well designed and follows the course style guidelines. For example, methods are of reasonable length (no more than 20-50 lines) and duplicated-looking code has been replaced by calls on methods.
- You submission makes clear in comments what algorithms you are using where —e.g. DFS, Dijkstra's shortest path algorithm.
- You have placed the total time spent on the assignment at the top of Butterfly in the following format
  /*Time Spent: XX hours YY minutes. */

Submission

On the CMS, submit files Butterfly.java.