CS310: Advanced Data Structures and Algorithms

Fall 2019 Programming Assignment 2

Due: Monday, November 11 2019 before midnight

Goals

This assignment aims to help you:

- Practice graphs
- Try greedy algorithms

In this assignment we begin our systematic study of algorithms and algorithm patterns.

Reading

- Graphs (K&T, chapter 3, S&W Chapter 4.1-4.2)
- Greedy algorithms (K&T chapter 4, S&W Chapter 4.4)
- Recursion (from CS210)

Advice

Before writing the code try to run a small example on paper. Think what you would do if you were given the set of instructions or hints and had to do it without a computer. Then start programming. It is always advisable, but especially important in this programming assignment. This assignment is about 80% reading, 20% coding...

Questions

1. **Graphs** (based on the "small world phenomenon" exercise from S&W, see here:

   http://www.cs.princeton.edu/courses/archive/spring03/cs226/assignments/bacon.html.

   Briefly, the assignment asks you to read in a file containing information about films and actors, and produce a histogram of the Kevin Bacon numbers (or anyone else... The center of the universe, Bacon in this case, is given as a parameter).

   Your task is to write a class that, when given the graph with the shortest paths from Bacon (or any other actor), takes an actor's name as a command line and produces this actor's Bacon number and the shortest path to Kevin Bacon. The name format is "Last, First" . You may use the `DegreesOfSeparation` class for ideas (see S&W code), but the output has to be different. Name your class `DegreesOfSeparationBFS` .

   Make sure that you understand the `SymbolGraph` class very well before you start coding. An illustration appears in the Undirected Graphs class notes.

   The command line parameters should be as in `DegreesOfSeparation` (it's probably easiest this way) plus the "sink" actor's name. That is – four command line parameters: The input movie file name, the separator, the "center" (Kevin Bacon in this case) and the actor you want to query. Notice that the original implementation reads the query from the standard input, but I am asking that you read it as a command line parameter. **Do not print the Bacon histogram.**

Example:

```
java -cp .:../algs4.jar DegreesOfSeparationBFS movies.txt "/" "Bacon, Kevin" "Kidman, Nicole"
```

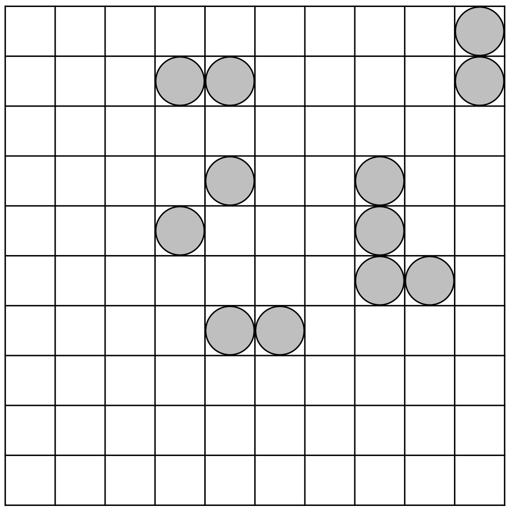
Output:

```
Done reading movies.txt
Kidman, Nicole has a bacon number of 2.
Kidman, Nicole was in the movie My Life (1993 I) with de Sosa, Ruth
de Sosa, Ruth was in the movie Planes, Trains & Automobiles (1987) with Bacon, Kevin
```

2. **Greedy recursive search:** Consider an $N \times N$ grid in which some squares are occupied (see figure below). The squares belong to the same group if they share a common edge. In the figure there is one group of four occupied squares, three groups of two squares, and two individual occupied squares. Assume the grid is represented by a 2 dimensional array. Write a program that:

   (a) Compute the size of a group when a square in the group is given.

   (b) Lists all group members.



For the class setup, use the attached `Grid.java`. The attached file contains a main, the Grid, and the Spot class. You will have to write only the groupSize method and a helper functions.

Here is one idea: set up a local Set of Spots to hold spots you've found so far in the cluster. From the current spot-position, greedily add as many direct neighbor spots as possible. For each neighbor spot that is actually newly-found, call recursively from that position (but make sure to not double-count See the full version of the `MakeChange.java` code in the DP class notes for an example of a recursion helper method – you need a recursion helper here to fill the set. Once the set is filled with spots in the group, the top-level method just returns the group. Just print its size and contents then. Notice that you may use the S&W code but it's probably best if you just use standard java. In this case you don't have to add `algs4.java` to your compilation and running classpath.

Usage: `java Grid 3 7` for example, to search from (3,7), the top occupied square of the L-shaped group. This case should print out:

```
4
(3 , 7)
(4 , 7)
(5 , 7)
(5 , 8)
```

To print all the group members you can take advantage of the Spot class toString function.

3. Dijkstra's algorithm modification: Modify Dijkstra's algorithm (code is available at the algs4 library, DijkstraSP.java), to implement a "tie breaker". When relaxing the edge, i.e., comparing `distTo[w]` to `distTo[v] + e.weight()`, if `distTo[w] > distTo[v] + e.weight()` then proceed as usual. However, if there are two paths with equal weights leading to a vertex, the one with the *smaller number of edges* will be selected. The big-O runtime should stay the same! So you should also keep track of the number of edges along a path. Before you code, make sure you understand Dijkstra's algorithm or at least its outline. Think about all you have to change in the algorithm to make it work in the same big-O. Name your class `DijkstraTieSP.java` . Have it take two command line arguments – a file to read the graph from and a starting vertex number.

   Test it on the attached file, `tinyEDW2.txt`, which is a modified version of S&W's `tinyEDW.txt` , containing a case where a tie breaker is needed.

Setup for Linux

As in the previous assignment, I recommend you test your code in a linux environment. You can create a directory `pa2` with three subdirectories: `src`, `classes` and `lib`. The compilation and run should be as in PA1 (notice that for the grid question you don't need the `algs4` library).

Delivery

The following files should be uploaded to Gradescope:

- DegreesOfSeparationBFS.java: usage java `java -cp .:../lib/algs4.jar DegreesOfSeparationBFS movies.txt "/" "Bacon, Kevin", "Kidman, Nicole"`. The usage includes the quotes since there is a space separating the last and first name . On my home laptop the full movies.txt file caused a stack overflow. You can use a reduced file size for testing. if you did so and what file you used.
- `Grid.java`, usage: `java Grid 3 7` (or any two numbers. Don't assume 3 and 7 will be the only test).
- `DijkstraTieSP.java`, usage: `java -cp .:../lib/algs4.jar DijkstraTieSP.java tinyEDW2.txt 0` (or any other starting vertex. Don't assume it's 0!).
- The `memo.txt` file will be graded manually (you still need to upload it). In your `memo.txt` file state the following:
  - Whether you used late days and if so – how many
  - In question 3 – what is the difference in the paths returned by the original implementation of Dijkstra's algorithm and the one you implemented? Specifically, run the `algs4` DijktsraSP implementation from vertex 0 and save the output to the `memo.txt` file. Then run your version, save it to the `memo.txt` file. Explain the differences briefly.