ECE 242: Data Structures and Algorithms- Fall 2016

Project 5: FunWithGraph (DFS,BFS,MSTW using Queue/Stack/Heap)

Due Date: **Deadline 11:50pm Wed December 14, 2016**

Description

The goal of this project is to operate a Graph data structure and implement various graph algorithms such as depth-first search (DFS), breadth-first search (BFS), minimum spanning tree Prim's algorithm for weighted (non-directed) graphs (MSTW). The graphs are defined using a rectangular grid and contain vertices that are defined by (integer) Cartesian coordinates (i,j) (or (x,y)), and edges that connect those vertices with a given weight. The weights of the graph can also be generated randomly. The resulting MST (which is also a graph) is actually a maze. We are then creating a random maze generator.

How to start

The project includes multiple 'source' files:

1. `GraphApp1.java` and `GraphApp2.java`: two application files used for testing (provided).
2. `Graph.java` file containing the class objects: Graph, Edges, Vertex (to complete).
3. `Queue.java` and `Stack.java` files containing the object class queue and stack (provided).
4. `Heap.java` file containing the class Heap to operate on edges (to complete).
5. `StdDraw.java` and `EasyIn.java`, tools (provided).

All the functionalities of the application files (presented in details below) should be successfully implemented to obtain full credit. You need to proceed step-by-step, application by application. A list of methods to implement is described at the end of each section.

`GraphApp1.java`

The code creates a regular grid where all first neighbors are connected. The size of the grid is set by the user. The weights are uniformly equal to 1 for all edges. The code returns some information about the graph (such as connecting edges, total weight and adjacency matrix for small systems).

Figure 1 provides a complete example of a 3x3 grid graph with matrix representation.

Example of code execution:

```
>java GraphApp1

Welcome to Graph App 1
======================

Enter Total Grid Size x:
3
Enter Total Grid Size y:
3

List of edges + weights:
(1,0)<-->(0,0)  1
```

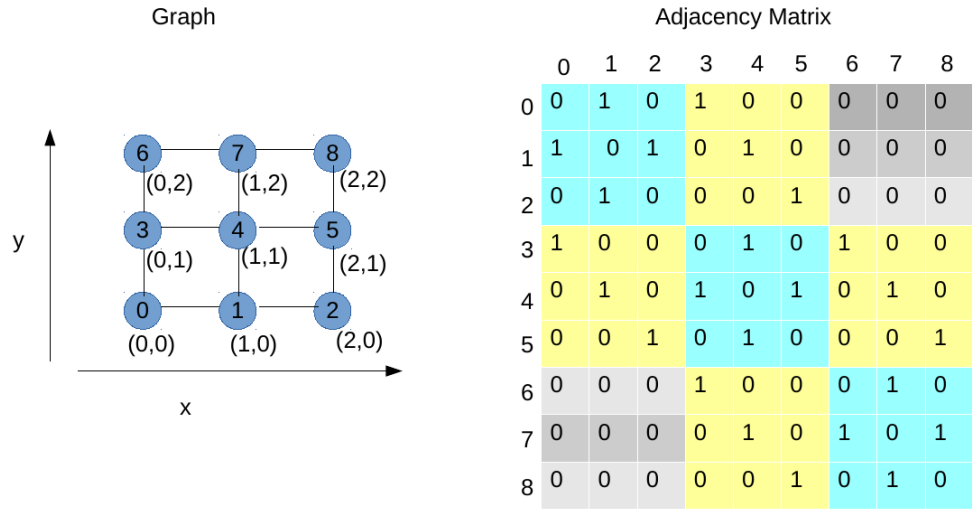

Figure 1: Grid-Graph, local and global coordinates, matrix representation.

```
(2,0)<-->(1,0)  1
(0,1)<-->(0,0)  1
(1,1)<-->(1,0)  1
(1,1)<-->(0,1)  1
(2,1)<-->(2,0)  1
(2,1)<-->(1,1)  1
(0,2)<-->(0,1)  1
(1,2)<-->(1,1)  1
(1,2)<-->(0,2)  1
(2,2)<-->(2,1)  1
(2,2)<-->(1,2)  1
Total weight: 12

Matrix:

  0  1  0  1  0  0  0  0  0
  1  0  1  0  1  0  0  0  0
  0  1  0  0  0  1  0  0  0
  1  0  0  0  1  0  1  0  0
  0  1  0  1  0  1  0  1  0
  0  0  1  0  1  0  0  0  1
  0  0  0  1  0  0  0  1  0
  0  0  0  0  1  0  1  0  1
  0  0  0  0  0  1  0  1  0

What do you want to do? (1:DFS, 2:BFS, 0: Exit)?
```

At this stage a grid-graph (identical to the one in Figure 1) is plotted in color GRAY. On can continue the execution by selecting either DFS or BFS search algorithms. The search starts at the vertex at coordinate (0,0) (i.e. vertex 0). As a result, a MST will be interactively created (the MST is not unique since all the weights are the same), you will display all the edges connections, and the plot will have to be updated interactively (at each time there is a new vertex that is visited, you need to update the plot.. you will use the color RED for the path).

Here is what happens if we continue the execution using DFS:

```
What do you want to do? (1:DFS, 2:BFS, 0: Exit)?
1

From (0,0) to (1,0)
From (1,0) to (2,0)
From (2,0) to (2,1)
From (2,1) to (1,1)
From (1,1) to (0,1)
From (0,1) to (0,2)
From (0,2) to (1,2)
From (1,2) to (2,2)
Press return to continue
```

The resulting MST is shown in Figure 2:

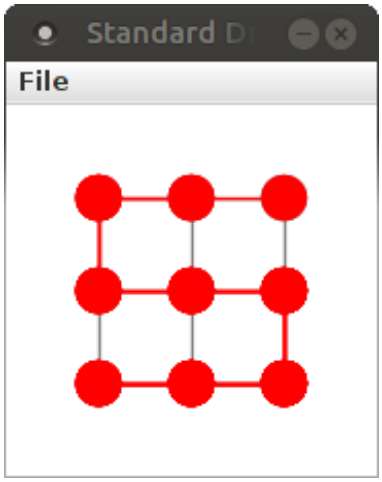

Figure 2: result DFS.

In turn, if we select option 2 for BFS, we get:

```
What do you want to do? (1:DFS, 2:BFS, 0: Exit)?
2
(0,0)
From (0,0) to (1,0)
From (0,0) to (0,1)
From (1,0) to (2,0)
From (1,0) to (1,1)
From (0,1) to (0,2)
From (2,0) to (2,1)
From (1,1) to (1,2)
From (2,1) to (2,2)
Press return to continue
```

The resulting MST is now shown in Figure 3:
What you need to implement in `Graph.java`:

1. A constructor that generates a grid. We consider a rectangular grid where the number of vertices is

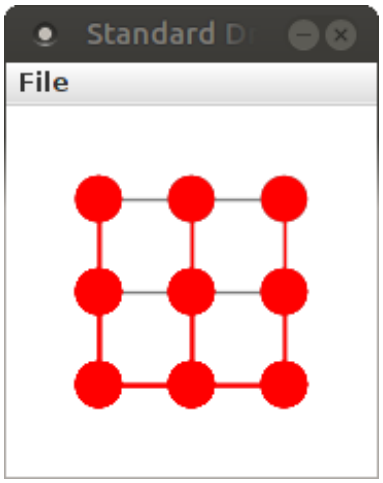

Figure 3: result BFS.

given and equal to nVertex=nx∗ny. Hint: create the list of vertices and initialize the adjacency matrix to 0.

2. The `uniformGridWeight` method that sets the adjacency matrix by adding edges to the graph (all with weight 1). Hint: only the first neighbors of each vertex are connected, you may need also to define the method `addEdge`.

3. The `displayInfoGraph` method, provides information about the graph. Your output should be similar to the example above. The method returns also the total weight of the graph.

4. The `displayAdjMatrix` method, provides the matrix. Your output should be similar to the example above. You also need the simple `getnVertex` method used to check the value of nVertex in the main code.

5. The `dfs` method, performs the depth first search. Your output should be similar to the example above. The search path is displayed interactively (the plot is also updated interactively). You need the class `Stack`, provided here. You will suppose that a max size for the Stack of nVertex.

6. The `bfs` method, performs the breadth first search. Your output should be similar to the example above. The search path is displayed interactively (the plot is also updated interactively). You need the class `Queue`, provided here. You will suppose that a max size for the Queue of nVertex.

7. The `getAdjUnvisitedNode` method used by both `dfs` and `bfs`.

You can now play around with your code, try bigger grid (square or rectangular) and observe (movie-like) the searches taking place.

`GraphApp2.java`

The same grid graph is created but you are now going to use a random generator to select the weights (weights between 1 and 4).

Here an example of execution (would be different for you):

```
>java GraphApp2

Welcome to Graph App 2
======================
```

```
Enter Total Grid Size x:
8
Enter Total Grid Size y:
8
INFO FOR MAIN GRAPH

List of edges + weights:
(1,0)<-->(0,0)  4
(2,0)<-->(1,0)  3
(3,0)<-->(2,0)  4
(4,0)<-->(3,0)  3
(5,0)<-->(4,0)  4
(6,0)<-->(5,0)  4
(7,0)<-->(6,0)  2
(0,1)<-->(0,0)  2
(1,1)<-->(1,0)  4
(1,1)<-->(0,1)  1
(2,1)<-->(2,0)  4
(2,1)<-->(1,1)  3
(3,1)<-->(3,0)  1
(3,1)<-->(2,1)  2
(4,1)<-->(4,0)  2
(4,1)<-->(3,1)  1
(5,1)<-->(5,0)  4
(5,1)<-->(4,1)  3
(6,1)<-->(6,0)  3
(6,1)<-->(5,1)  3
(7,1)<-->(7,0)  2
(7,1)<-->(6,1)  1
(0,2)<-->(0,1)  1
(1,2)<-->(1,1)  1
(1,2)<-->(0,2)  1
(2,2)<-->(2,1)  3
(2,2)<-->(1,2)  2
(3,2)<-->(3,1)  2
(3,2)<-->(2,2)  2
(4,2)<-->(4,1)  2
(4,2)<-->(3,2)  1
(5,2)<-->(5,1)  4
(5,2)<-->(4,2)  4
(6,2)<-->(6,1)  2
(6,2)<-->(5,2)  4
(7,2)<-->(7,1)  1
(7,2)<-->(6,2)  1
(0,3)<-->(0,2)  2
(1,3)<-->(1,2)  2
(1,3)<-->(0,3)  2
(2,3)<-->(2,2)  1
(2,3)<-->(1,3)  3
(3,3)<-->(3,2)  4
(3,3)<-->(2,3)  4
(4,3)<-->(4,2)  2
(4,3)<-->(3,3)  4
(5,3)<-->(5,2)  2
(5,3)<-->(4,3)  3
(6,3)<-->(6,2)  1
(6,3)<-->(5,3)  1
```

```
(7,3)<-->(7,2)  2
(7,3)<-->(6,3)  2
(0,4)<-->(0,3)  3
(1,4)<-->(1,3)  1
(1,4)<-->(0,4)  2
(2,4)<-->(2,3)  3
(2,4)<-->(1,4)  1
(3,4)<-->(3,3)  3
(3,4)<-->(2,4)  2
(4,4)<-->(4,3)  4
(4,4)<-->(3,4)  3
(5,4)<-->(5,3)  1
(5,4)<-->(4,4)  1
(6,4)<-->(6,3)  4
(6,4)<-->(5,4)  1
(7,4)<-->(7,3)  4
(7,4)<-->(6,4)  4
(0,5)<-->(0,4)  1
(1,5)<-->(1,4)  4
(1,5)<-->(0,5)  3
(2,5)<-->(2,4)  2
(2,5)<-->(1,5)  1
(3,5)<-->(3,4)  3
(3,5)<-->(2,5)  1
(4,5)<-->(4,4)  3
(4,5)<-->(3,5)  1
(5,5)<-->(5,4)  4
(5,5)<-->(4,5)  1
(6,5)<-->(6,4)  1
(6,5)<-->(5,5)  2
(7,5)<-->(7,4)  3
(7,5)<-->(6,5)  3
(0,6)<-->(0,5)  1
(1,6)<-->(1,5)  3
(1,6)<-->(0,6)  1
(2,6)<-->(2,5)  1
(2,6)<-->(1,6)  1
(3,6)<-->(3,5)  4
(3,6)<-->(2,6)  4
(4,6)<-->(4,5)  1
(4,6)<-->(3,6)  4
(5,6)<-->(5,5)  2
(5,6)<-->(4,6)  3
(6,6)<-->(6,5)  2
(6,6)<-->(5,6)  2
(7,6)<-->(7,5)  4
(7,6)<-->(6,6)  3
(0,7)<-->(0,6)  1
(1,7)<-->(1,6)  4
(1,7)<-->(0,7)  4
(2,7)<-->(2,6)  3
(2,7)<-->(1,7)  4
(3,7)<-->(3,6)  2
(3,7)<-->(2,7)  2
(4,7)<-->(4,6)  2
(4,7)<-->(3,7)  1
(5,7)<-->(5,6)  1
(5,7)<-->(4,7)  3
```

```
(6,7)<-->(6,6)  4
(6,7)<-->(5,7)  3
(7,7)<-->(7,6)  3
(7,7)<-->(6,7)  2
Total weight: 271

Press return to continue
```

The resulting grid-graph is shown in Figure 4. We note that the method `plot` (provided) uses four different thickness to draw the connections between all the nodes (corresponding to the weights).

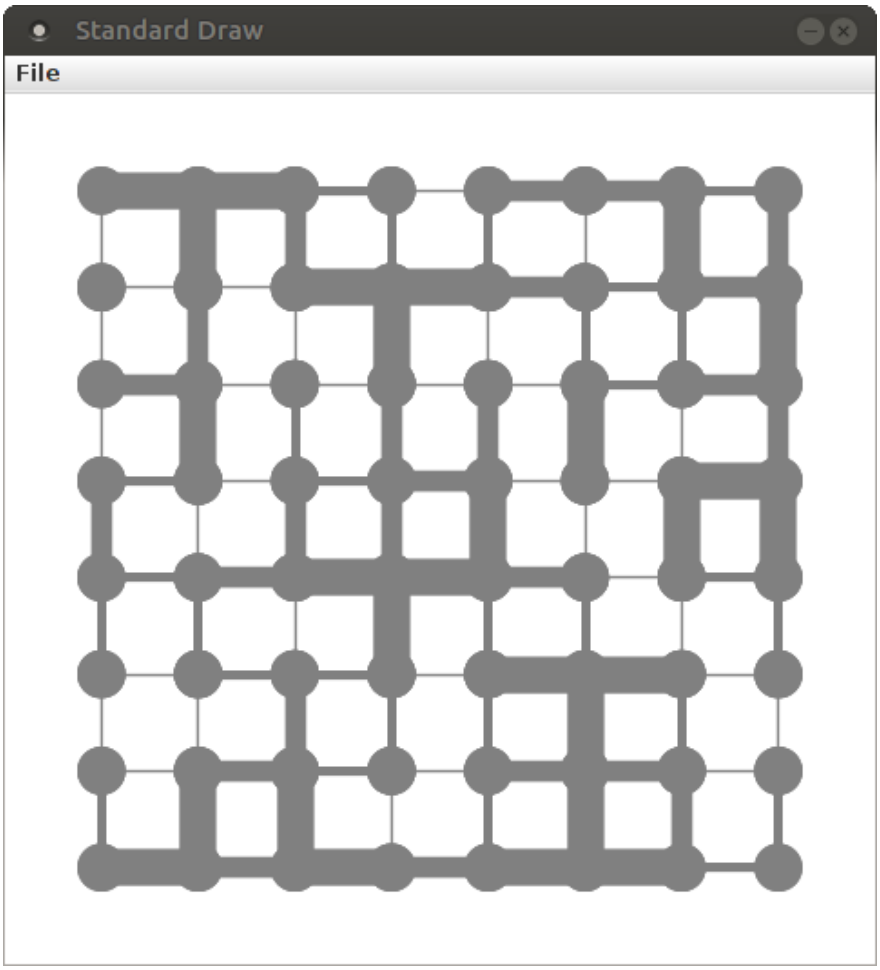

Figure 4: Grid-graph 8x8 with random weights.

The code computes then the MSTW of the original graph (this is also a graph data structure). The number of vertices is the same than the original graph but the number of edges is nVertex-1 (63 in the example). Once the new graph is obtained, the code returns the new info containing the optimized connections with weights as well as the total minimum weight, and the new plot (the MSTw is plotted in BLUE on top of the original GRAY graph).

Here is the example of execution:

```
INFO FOR MST
```

```
List of edges + weights:
(2,0)<-->(1,0)  3
(3,0)<-->(2,0)  4
(7,0)<-->(6,0)  2
(0,1)<-->(0,0)  2
(1,1)<-->(0,1)  1
(3,1)<-->(3,0)  1
(3,1)<-->(2,1)  2
(4,1)<-->(4,0)  2
(4,1)<-->(3,1)  1
(5,1)<-->(5,0)  4
(5,1)<-->(4,1)  3
(7,1)<-->(7,0)  2
(7,1)<-->(6,1)  1
(0,2)<-->(0,1)  1
(1,2)<-->(1,1)  1
(2,2)<-->(1,2)  2
(3,2)<-->(2,2)  2
(4,2)<-->(4,1)  2
(4,2)<-->(3,2)  1
(7,2)<-->(7,1)  1
(7,2)<-->(6,2)  1
(1,3)<-->(1,2)  2
(1,3)<-->(0,3)  2
(2,3)<-->(2,2)  1
(4,3)<-->(4,2)  2
(5,3)<-->(5,2)  2
(6,3)<-->(6,2)  1
(6,3)<-->(5,3)  1
(7,3)<-->(7,2)  2
(1,4)<-->(1,3)  1
(1,4)<-->(0,4)  2
(2,4)<-->(1,4)  1
(3,4)<-->(3,3)  3
(3,4)<-->(2,4)  2
(5,4)<-->(5,3)  1
(5,4)<-->(4,4)  1
(6,4)<-->(5,4)  1
(0,5)<-->(0,4)  1
(2,5)<-->(1,5)  1
(3,5)<-->(2,5)  1
(4,5)<-->(3,5)  1
(5,5)<-->(4,5)  1
(6,5)<-->(6,4)  1
(6,5)<-->(5,5)  2
(7,5)<-->(7,4)  3
(7,5)<-->(6,5)  3
(0,6)<-->(0,5)  1
(1,6)<-->(0,6)  1
(2,6)<-->(2,5)  1
(2,6)<-->(1,6)  1
(4,6)<-->(4,5)  1
(5,6)<-->(5,5)  2
(6,6)<-->(6,5)  2
(7,6)<-->(6,6)  3
(0,7)<-->(0,6)  1
(1,7)<-->(1,6)  4
```

```
(3,7)<-->(3,6)  2
(3,7)<-->(2,7)  2
(4,7)<-->(4,6)  2
(4,7)<-->(3,7)  1
(5,7)<-->(5,6)  1
(7,7)<-->(7,6)  3
(7,7)<-->(6,7)  2
Total weight: 108

Press return to continue
```

The resulting MSTw which also a maze is shown in Figure 5:

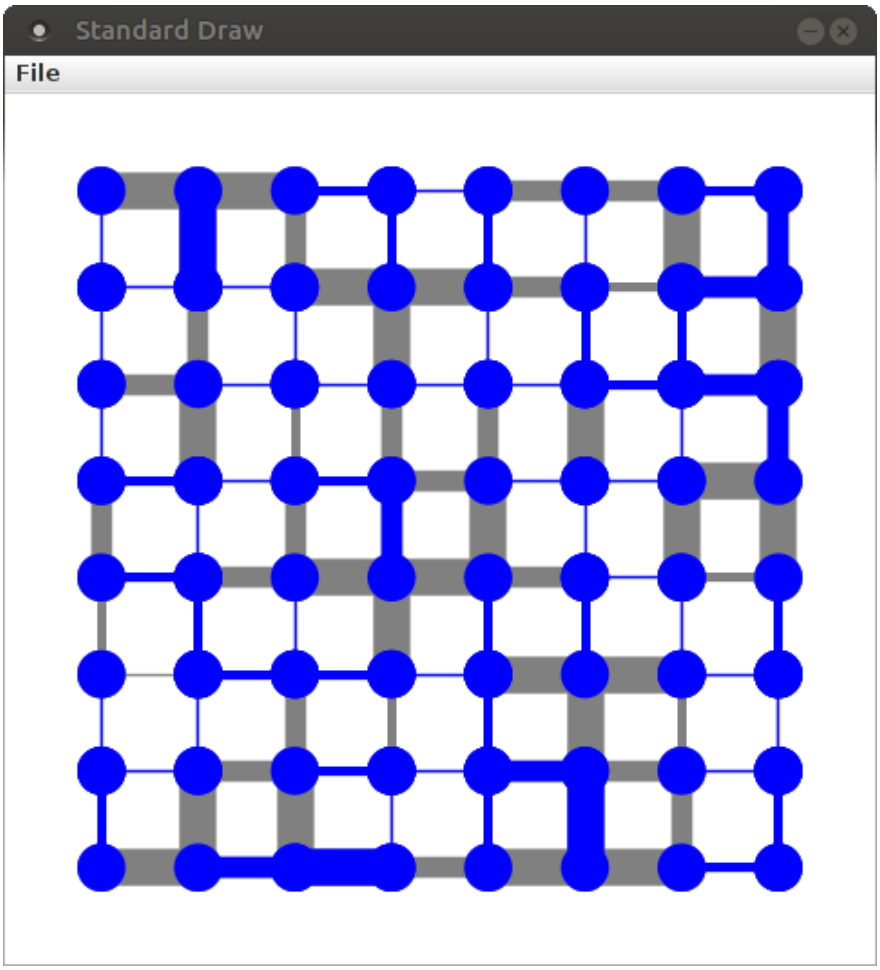

Figure 5: result MSTw, Random Maze 8x8.

You have created a random Maze Generator!

The code proceed by asking if you want to perform dfs or bfs on the new MSTw grid-graph structure. You will use the same method implemented for GraphApp1. Again, everything happens interactively, the display of the edges connections, and the update of the plot (at each time there is a new vertex that is visited, you need to update the plot, so you can see the progression of the RED path on the BLUE graph).

What you need to implement:

1. The method `randomGridWeight` that sets the adjacency matrix by adding edges to the graph (all with random weights between 1 to 4).

2. The method `mstw` that implement the Prim's algorithm for computing the MSTw (that is returned as new grid-graph). You will use a heap to implement the priority queue.

3. The entire class `Heap` that considers the smallest weight for the edges as priority item.

REQUIREMENT, SUBMISSIONS

1. You need to implement step by step the class following the requirement of Applications 1 and 2. The applications will not compile as is. You need to start by commenting out the application files and gradually uncomment them to help the debugging process.

2. Include all information in a README file. Indicate which applications have been fully completed (or what has been partially completed in each application). Submit a single zip file composed of: All your source files (*.java), and your README file.

Grading Policy

This project will be graded out of 100 points:

1. Your program should compile successfully, and overall writing style (10 points)

2. Your program should implement all functionalities and run correctly (90 points).

Extra Credit (no help from TA or instructor)

(10pts) You can implement an option 3 for GraphApp2 that computes the shortest path (Dijkstra's algorithm) from the bottom left point (0,0) to the upper right corner (nVertex-1,nVertex-1) point. You will consider that all the weights (between 1 and 4) are now equal to 1 (to obtain a real maze situation). You will display the path, compute/return the cost (sum of weights) and plot (in GREEN) the path (on top of the BLUE MST graph).