ADS Project 2013

NAME : Rohini Kar

UFID: 67308301

UF Email Account : rohinikar@ufl.edu

COMPILATION:

Compiler used- standard JDK

Steps for Compilation and Running- The main class name is mst.java. We have to run this file

Input required :

Fibonacci Scheme:-

Random mode: We have to provide "-r", the value for "n" and the value for "d" as the 3 arguments when we first run the mst.java

"-r" here refers to random mode.

"n" refers to the number of nodes(have to provide a value).

"d" refers to the density percentage(give only the integer(i.e. 10 or 20 or 100) and not with the "%" symbol appended)

User Input mode: We have to provide "-f" and the value for "the file path" as the 2 arguments when we first run the mast.java

"-f" here refers to the Fibonacci scheme.

"the file path" denotes the location where the particular file is stored.

Simple Scheme:-

Random mode: We have to provide "-r", the value for "n" and the value for "d" as the 3 arguments when we first run the mast.java

"-r" here refers to random mode.

"n" refers to the number of nodes(have to provide a value)..

“d” refers to the density percentage(give only the integer(i.e. 10 or 20 or 100) and not with the “%” symbol appended)

User Input mode: We have to provide “-s” and the value for “the file path” as the 2 arguments when we first run the mast.java

“-s” here refers to the Simple scheme.

“the file path” denotes the location where the particular file is stored.

Objective:

To find PRIM’s Minimum Cost Spanning Tree using

a)Simple Scheme

b)Fibonacci Scheme

Function Prototypes and Structure of the Programs:

We have 5 class files:

- mst
- RandomGeneration
- DFS
- SimpleScheme
- FibonacciScheme

mst.java----- This the main class of the program. When we first run this file, we need to provide arguments specified above for

1)*Simple Scheme in User Input mode*

OR

2)*Simple Scheme in Random mode*

OR

3)*Fibonacci Scheme in User Input mode*

OR

4)*Fibonacci Scheme in Random mode*

If using User Input mode for Simple Scheme:

After getting the required arguments, the mst.java calls the SimpleScheme.java by passing the file name in the constructor of the class SimpleScheme.java. The SimpleScheme.java prints out the final minimum spanning tree and it's cost.

If using User Input mode for Fibonacci Scheme:

After getting the required arguments, the mst.java calls the FibonacciScheme.java by passing the file name in the constructor of the class FibonacciScheme.java. The FibonacciScheme.java prints out the final minimum spanning tree and it's cost.

If using Random mode for Simple Scheme:

After getting the required arguments, the mst.java calls the RandomGeneration.java by passing the arguments(number of nodes & density percentage) in the constructor of the class RandomGeneration.java. The RandomGeneration.java forms a matrix corresponding to the given density and passes this matrix to DFS.java by calling it's constructor. The DFS.java checks to see if the matrix obtained from RandomGeneration.java is connected or not. If the graph is found to be not connected, the RandomGeneration.java generates another graph and calls the DFS.java. This process continues till we find the proper connected graph. After this, the mst.java calls the SimpleScheme.java and the SimpleScheme.java finally gives the minimum spanning tree and it's cost. The time taken by the SimpleScheme.java is also calculated from the mst.java.

If using Random mode for Fibonacci Scheme:

After getting the required arguments, the mst.java calls the RandomGeneration.java by passing the arguments(number of nodes & density percentage) in the constructor of the class RandomGeneration.java. The RandomGeneration.java forms a matrix corresponding to the given density and passes this matrix to DFS.java by calling it's constructor. The DFS.java checks to see if the matrix obtained from RandomGeneration.java is connected or not. If the graph is found to be not connected, the RandomGeneration.java generates another graph and calls the DFS.java. This process continues till we find the proper connected graph. After this, the mst.java calls the Fibonacci Scheme.java and the Fibonacci Scheme.java finally gives the minimum spanning tree and it's cost. The time taken by the Fibonacci Scheme.java is also calculated from the mst.java.

Different functions used in the different classes & their utilities:-

1)**mst.java** – This class has the main method. From this method, we call the RandomGeneration.java or SimpleScheme.java or FibonacciScheme.java.

2)**RandomGeneration.java** – This class sends the randomly generated matrix to DFS.java. We pass the arguments(# of nodes & density percentage) to the constructor of RandomGeneration.java from mst.java.

getMatrix()**-** This method returns the proper connected randomly generated graph. In this method, we randomly generate two numbers within n(where n is the input number of nodes) and assign it to 2 vertices. We also generate another number using Random(1000)+1 as specified in the project requirements and assign it to the edge existing between the two vertices. This process continues till we find a proper connected graph corresponding to the input density.

getEdges()- This method returns the number of edges of the generated graph.

3)**DFS.java** – This class receives the randomly generated graph in it's constructor from RandomGeneration.java and returns true or false if the graph is connected or not respectively.

depthFirstSearch**()** – This method takes as argument a first node of the graph and the number of nodes. It checks if the graph is connected or not by checking if all the vertices are visited and returns true or false.

4)**SimpleScheme.java** – This class finds the minimum cost spanning tree for the simple scheme.

primFunction() – This function takes as arguments either the file path or the arguments(matrix, number of nodes & number of edges) according to User Input mode or Random mode and calls another function called createMST(). This method returns a boolean value.

getData() - This function takes as arguments either the file path or the arguments(matrix, number of nodes & number of edges) according to User Input mode or Random mode. It gets the data i.e. the vertices and the edge weights between two vertices.

mstGeneration() – This method takes as argument a random vertex and the edges present in the graph in order to find the minimum spanning tree.

5)**FibonacciScheme.java** - This class finds the minimum cost spanning tree for the Fibonacci scheme.

heapify() - This method takes as arguments either the file path or the arguments(matrix, number of nodes & number of edges) according to User Input mode or Random mode

ifPresent() – This method checks to see if a particular combination is present in the used hash map.

getData() – This method takes as arguments either the file path or the arguments(matrix, number of nodes & number of edges) according to User Input mode or Random mode and returns another matrix containing edges.

addNode() – This method is used to add a node into the Fibonacci heap.

removeMin() – This method is used to remove the minimum element from the heap. It returns a node.

meld() – It melds two trees comparing their degrees after remove min operation. It returns a boolean value.

merge() – It is used to merge two nodes while melding.

ifMin() – This method is used to find if a minimum element exists in the heap. It returns a boolean value.

removeNode() – This method is used to remove a given particular node . It takes as argument a particular node.

show() – This is used to print the heap.

updateChild() – This method is used to update the child pointers.

updateData() - This method returns a boolean value and sets the data for the node.

updateDegree() – This method is used to update the degree of a particular node. It takes as input a particular node and returns a boolean value.

Output:

An example:

On running the Simple Scheme using “-r”, “5”, 100” as arguments, the following output is obtained-

```
Density:100.0
final connected Graph
Counter value:5
Final result after DFS
*****************************
      DENSITY = 100.0%
Counter value:25.0
*****************************
588  132  705  734  442
22  861  873  18  337
244  728  230  854  672
971  409  406  778  971
671  467  760  252  120
Cost: 783

1->3
1->0
1->4
3->2

Total time taken by Simple Scheme in Random mode:123
```

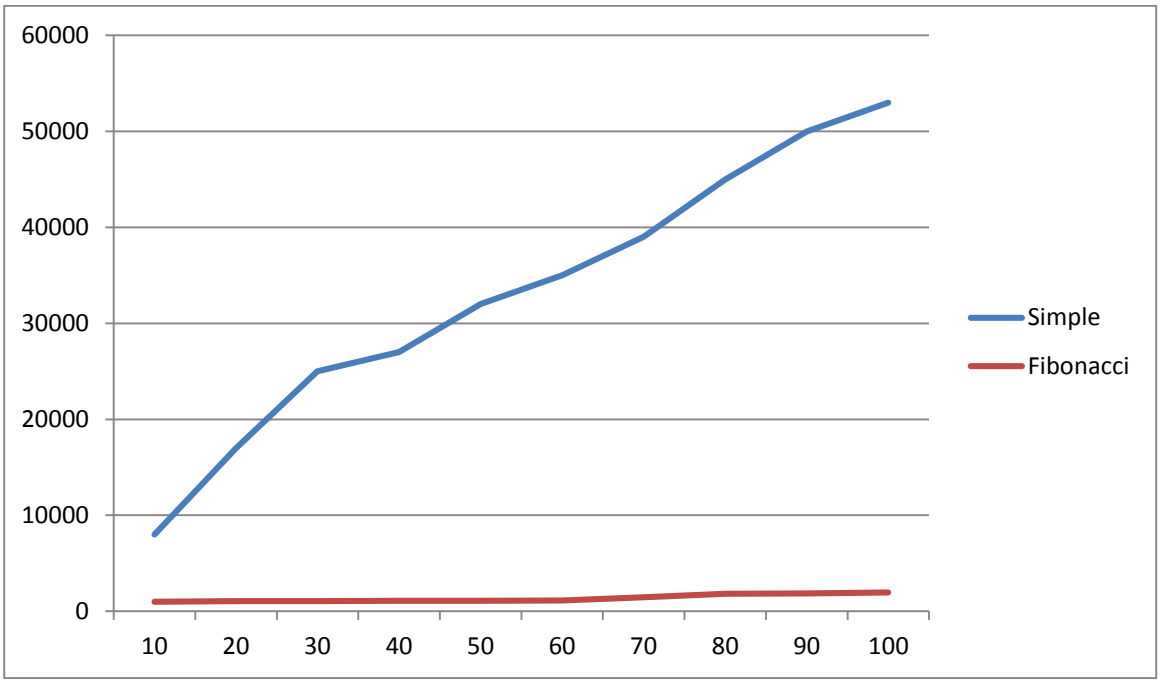
Summary of result comparison:

Expectation-

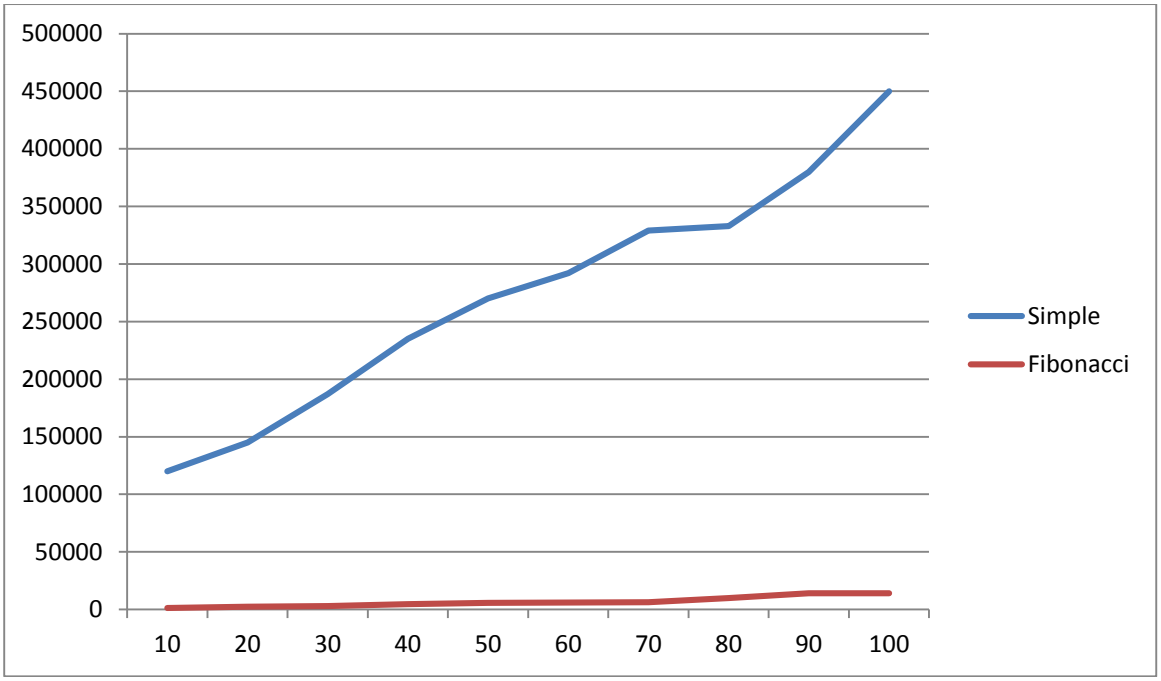
The Simple Scheme is supposed to take an overall time complexity of O(n^2) for finding the minimum cost spanning tree where n is the number of edges. This is because each extract minimum operation takes O(n) time and there are n such operations.
Whereas
The Fibonacci Scheme is supposed to take an overall time complexity of O(nlogn+e) where n is the number of nodes and e is the number of edges. Each remove min operation of Fibonacci heap takes O(log n) time and thus total extract minimum operations take O(nlog n) time.

Outputs received:

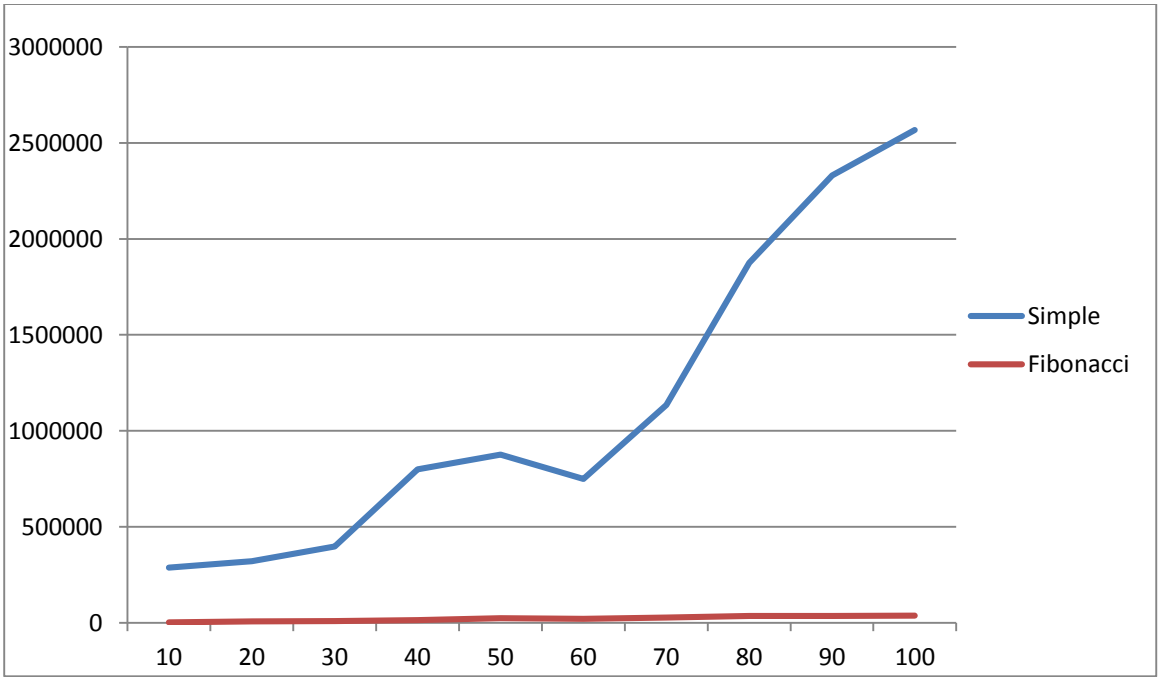
Below graphs have been plotted:-



Plot for 1000 nodes with density along x-axis and time along y-axis. The time is measured in milliseconds.



Plot for 3000 nodes with density along x-axis and time along y-axis. The time is measured in milliseconds.



Plot for 5000 nodes with density along x-axis and time along y-axis. The time is measured in milliseconds.

The performance measurements in tabular format:

Density	1000 nodes		3000 nodes		5000 nodes	
	Simple	Fibonacci	Simple	Fibonacci	Simple	Fibonacci
10	8000	1000	120000	1200	287000	3000
20	17000	1040	145000	2450	320000	6990
30	25020	1070	187000	3000	398000	9080
40	27000	1080	235000	4770	800000	13350
50	32100	1100	270000	5800	877000	24320
60	35000	1130	292000	6090	750000	19980
70	39090	1450	329000	6200	1135000	27210
80	45010	1830	333000	10050	1876000	35930
90	50000	1850	380000	14000	2331000	36000
100	53000	1970	450000	14150	2567000	37880

The Simple Scheme is working fine both in the User Input mode and the Random mode. However, the Fibonacci Scheme works intermittently and the below graphs have been plotted with data at different times. In the mst.java class file, if we select the Random mode, the Simple scheme & the Fibonacci scheme both run, but due to the Fibonacci scheme not working properly, we sometimes get an error(even if the Simple scheme works properly).

The differences observed in the actual runtime of the programs with that of the expected sometimes may be due slow machines and sometimes because of less heap capacity. Also, sometimes, issues have been faced because of different input file format and the way the program is supposed to read the file.