Report

COP 5536 Fall 2013

Programming Project

Implementation of Prims Algorithm

Name : Ankit Sharma

UF ID : 24868901

Email : ankitsharma13@ufl.edu

Make File

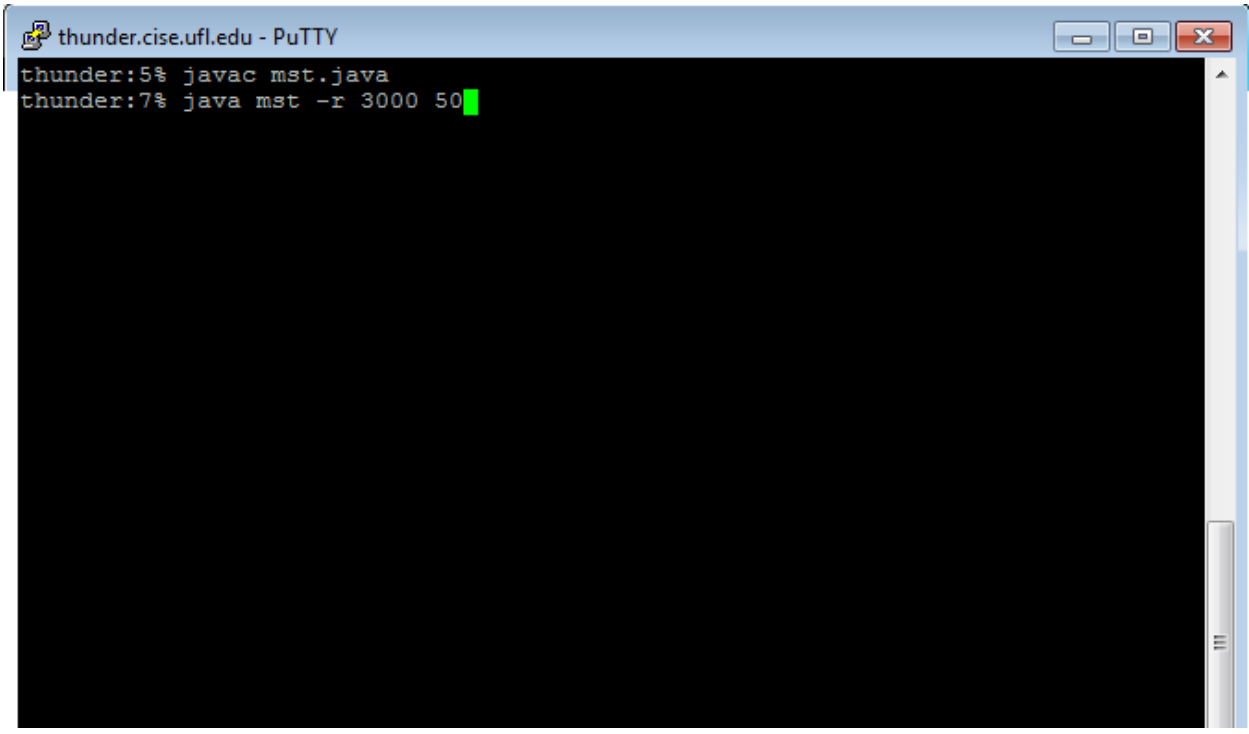
Compiling and Execution Steps :

1. Pogram is compiled and executed using java compiler
2. Go to the directory where the mst.java file is put. Execute the steps below :-

Random Mode :-

Compile File :- javac mst.java
Execute File :- java mst –r V D

-r - Random Mode
V – Number of Vertices in the Graph
D – Density of Graph (%)



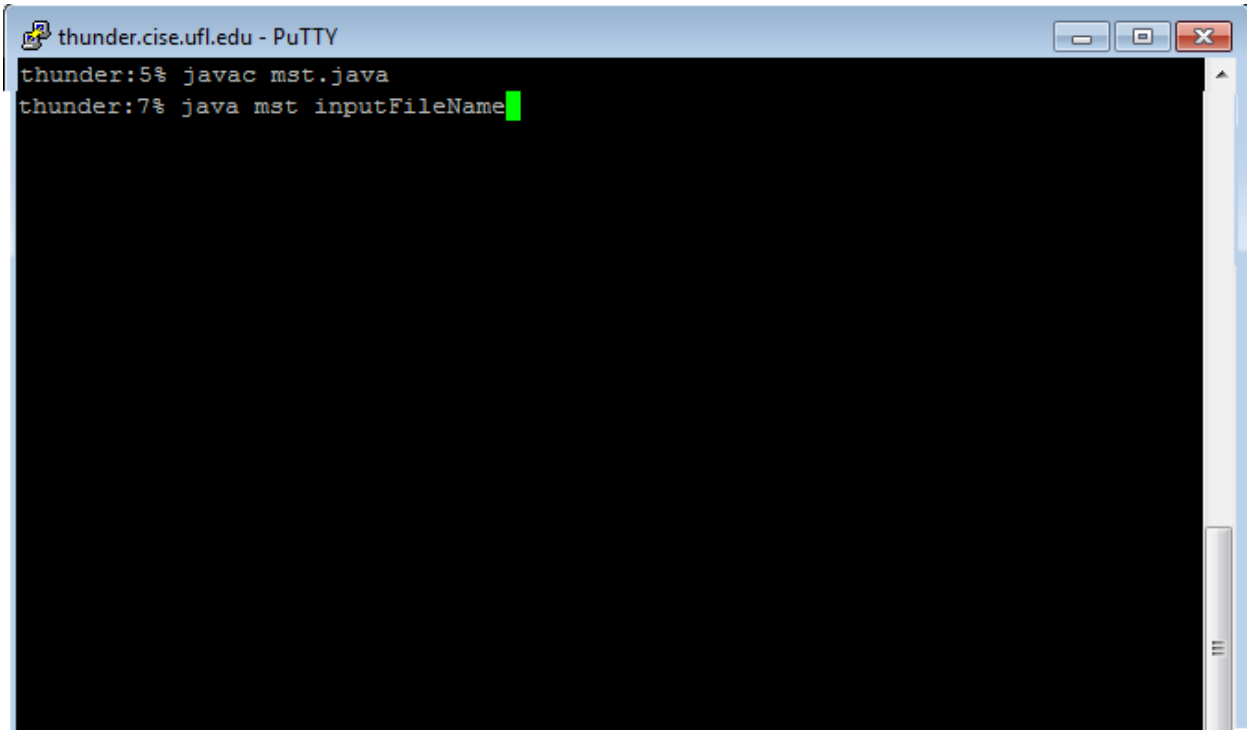
User Input Mode :-

1. Put the input file(*inputFIleName)* in the same directory as mst.java
2. Provide proper data in proper format in the input file for MST

Compile File :- javac mst.java

Fibonacci Scheme (-f),
Execute File :- java mst –f *inputFIleName*

Simple Scheme (-s),
Execute File :- java mst –s *inputFIleName*



PROGRAM STRUCUTRE

The program consists of the following classes:

1. **mst:** This is the main class. It takes different types of input i.e the random mode (-r) & file scheme mode(-f) (which is simple scheme and Fibonacci schema)

   Important Functions and Constructors:
   - simpleSchemeMST :- This function is called for simple scheme generation
   - fibonacciSchemeMST :- This function is called for generating MST using Fibonacci heaps based on priority queue

2. **Graph:** This is the very essential class. Its function is to create the graph. It in turns calls the simple scheme MST and the Fibonacci MST.

   Important Functions and Constructors:
   - **Graph(int V,int D) :**When random mode is selected this constructor is called. It takes two parameters viz. V (vertex) and D (density).
   - **Graph(String file)** : Takes File name to be read.
   - **depthFirstSearch(Graph graph)** : This function checks whether the graph is connected or not.
   - **simpleSchemeMST()**:Calculate MST using simple scheme (array).
   - **fibonacciSchemeMST()** : Calculate MST using simple scheme (array).
   - **insertPQ(int V,int W)**: This functions is used by the mstw() to put the new elements into the priority.

3. **vertex**: The class defines the structure of each node.

   Important Functions and Constructors:
   - **vertex(int ame,AdjacentNode adjNode, boolean visited)** : This constructor creates the node object.

4. **AdjacentNode** : This class creates the Adjacent node of each vertex using the linked list

   Important Functions and Constructors:
   - **AdjacentNode(int vertexId,int weight, AdjacentNode nextNode)** : Assigns the weight and the next node to the given vertex.

5. **edges**: Simple scheme to calculate MST uses this to keep a check on edges.

6. **PriorityQueue:** This class contains functions required for the simple priority Q implementation

   Important Functions and Constructors
   - **public void insert(edges item):** Inserts edge in the priority queue and maintained
   - **public edges removeMinimum():** The important functions which gives the edges to be included in MST
   - **removeN(int n):** Implements the remove based on the indexed is used by the insert
     search(int searchIndex) : important function, it find the element at the given node

7. **Fibonacci:** Implements Fibonacci heap
   - Enters new node and returns it
   - **removeMin:** Essential function. Removes minimum element and returns it to MST
   - **consolidate:** It is called after remove min to consolidate the graph
   - **link:** to link two subtrees generated
   - **decreaseKey, cascadingKey,cut :** implement the function of Fibonacci heap as and how it is

8. **FibonacciHeapNode :** This class defines structure of Fibonacci node.

EXECUTION DETAILS

- Program is run with different number of vertices (1000, 3000 & 5000) and with different density (10% to– 100%)
- Below tables show the output result for both simple scheme and Fibonacci scheme

Nodes : 1000

Density	Simple		Fibonacci	
	Time (ms)	Cost	Time (ms)	Cost
10	168	13608	306	13664
20	269	6713	604	6705
30	365	4459	1237	4318
40	391	3362	1780	3390
50	497	2860	2150	2841
60	518	2483	2635	2495
70	587	2231	1480	2260
80	812	1934	3605	1954
90	903	1854	3801	1835
100	878	1744	4550	1730

Node: 3000

Density	Simple		Fibonacci	
	Time (ms)	Cost	Time (ms)	Cost
10	2891	13254	23034	13307
20	6056	7529	38011	7538
30	9057	5793	38719	5838
40	13554	4509	25770	4499
50	7846	4026	40852	4026
60	7055	3657	40094	3645
70	10018	3448	41873	3469
80	9770	3448	36387	3343
90	10206	3238	32635	3226
100	11464	3182	3189	34167

Node: 5000

Density	Simple		Fibonacci	
	Time (ms)	Cost	Time (ms)	Cost
10	6283	14191	50950	14198
20	12793	8813	55309	8902
30	18913	6755	69186	6774
40	23654	5842	71698	5858
50	33184	5482	92279	5464
60	39031	5296	112467	5281
70	44367	5156	135874	5149
80	43661	5082	122774	5080
90	-	Out of heap	-	Out of heap
100	-	Out of heap	-	Out of heap