REPORT

Name: Anindya Nandy

UFID: 80999501

e-mail: a.nandy89@gmail.com

Language : **Java:** 1.7.0_45; Java HotSpot(TM) Client VM 24.45-b08
Runtime : Java(TM) SE Runtime Environment 1.7.0_45-b18
IDE : NetBeans IDE 7.4

Compilation: compile all the .java files using javac from prompt. Eg. javac <filename>.java

Input : 1. java mst r <n> <density> (random mode)

2. java mst f <filename>.txt (file mode with fibonachhi heap)

3.java mst s <filename>

output: mincost<cost>

completion time<time>

13 //edges

0 1

1 2

Program Structure

To execute the program run mst.java which has the main function.

The user has to give command line input depending on the program will execute either in random mode or user input mode.

In the random mode mst.java calls its member function random() to generate random integer values to populate the ALnode class which basically stores the graph .Once the graph is populated primsusingfibo() is called to generate the prims algo using fibonachhi heap and prims()is called to generate prims algo using simple scheme.

In the user input mode mst.java calls its member function readFile() to extract values from file to populate the ALnode class which basically stores the graph .Once the graph is populated primsusingfibo() is called to generate the prims algo using fibonachhi heap and prims()is called to generate prims algo using simple scheme.

LIST OF CLASSES

1.fibo.java

<u>class variables:</u>

public int cost;

public int u,v;

public fibo left;

public fibo right;

public fibo parent;

public fibo child;

public boolean childcut;

public int degree;

<u>FUNCTIONS:</u>

void getcost(),void setcost()

void getu(),void getv(),void set(),void setv()

void setleft(),void setright(),void getleft(),void getright()

void setparent(),void getparent(),void getchild(),void setchild()

void setdegree(),void getdegree()

2.Oper.java //implentation of fibonacchi heap with all its functionalities

class variables

fibo head;

private fibo first;

private fibo min;

private fibo last;

private fibo temp1;

private int count = 0;

fibo[] ar;

FUNCTIONS

public boolean isEmpty() //check if fibo heap is empty or not

public void insert(int item, int a, int b) //Inserting elements into the fibo heap

public fibo removeMin() //removing min from heap

public void addtosubtree(fibo newchild) //adding child nodes to the chain of parent pointer in case parent is removed

public void combine() //doing pairwise combine

public void reCombineNew(fibo tpar1) // to do pairwise combine if nodes with same degree found

public void updateMin() // to update minimum pointer after min is removed

3.edges.java //class to store edges generated by the prims algo

class variables

private int ufinal;

private int vfinal;

FUNCTIONS

public int getUfinal()

public void setUfinal(int ufinal)

public int getVfinal()

public void setVfinal(int vfinal)

4.ALNode.java //class to store data as adjacency list

CLASS VARIABLES

private int cost;

private int u;

private int v;

private ALNode next;

FUNCTIONS

public int getCost()

public void setCost(int cost)

public int getU()

public void setU(int u)

public int getV()

public void setV(int v)

public ALNode getNext()

public void setNext(ALNode next)

5.mst.java //class to implement all the MST functionalities

CLASS VARIABLES

int e, v, u1, v1, w = 0;

int i, j;

int[][] ar = {{0, 3, 0, 10}, {3, 0, 3, 2}, {0, 3, 0, 4}, {10, 2, 4, 0}};

int mincost = 0;

ArraylistNode[] AL;

public static String f;

ArrayList<edges> er = new ArrayList<edges>();

FUNCTIONS

public static void main(String args[]) throws IOexception, IOException

public void random(int n, int d) //random graph generator

public void readFile(String f) throws FileNotFoundException, IOException //graph generator using file

public void addToAL(int u2, int v2, int cost)//graph generator using adjacency list

public ArraylistNode getStart(int u)

public void primusingfibo() //finding MST using prims algo utilizing fibonacchi heap

public void prim()   //finding MST using prims algo utilizing the simple scheme

6.ArraylistNode.java //implementation of adjacency list to store the vertices of the graph

CLASS VARIABLES

private int cost;

private int u;

private int v;

private ArraylistNode next;

FUNCTIONS

public int getCost()

public void setCost(int cost)

public int getU()

public void setU(int u)

public int getV()

public void setV(int v)

public ArraylistNode getNext()

public void setNext(ArraylistNode next)

COMPARISON:

<u>RUNTIME</u>

Expected: The run time of prims algorithm to find MST using fibonachhi heap is expected to be faster than the simple scheme as the run time complexity of fibonachhi heap is nlogn+v (n-number of nodes,v-edges) and for the simple scheme its is n^2 .

Resultant runtime: As expected the run time of fibonachhi heap was faster than the simple scheme as evident from the graphs below.But for smaller values of n and density it is sometimes observed that the run time of fibonachhi heap is slower as compares to the simple scheme. A possible reason for this could be the overhead incurred in creating the fibonachhi heap.

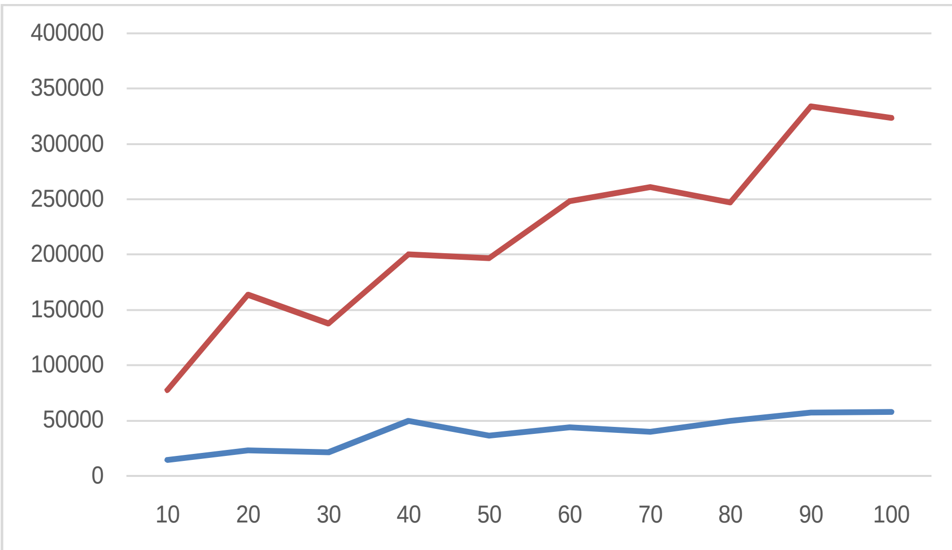
x axis- density(in percentage)

y axis- Time (milliseconds)

Fibonacci heap: Blue

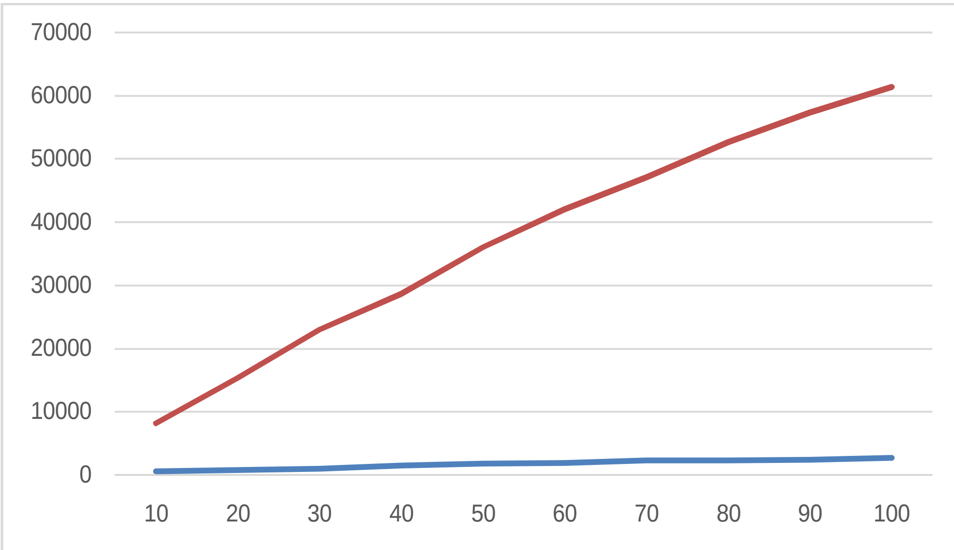
Simple Scheme: Orange

N=3000



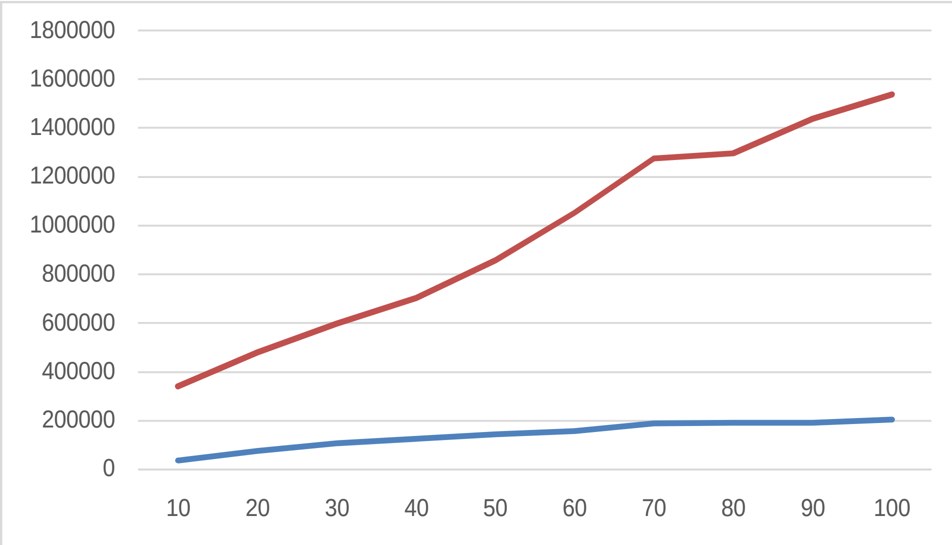
input value	density	fibonacci heap	SIMPLE scheme
3000	10	620	8237
3000	20	810	15416
3000	30	1053	23070
3000	40	1523	28675
3000	50	1761	36127
3000	60	1875	42108
3000	70	2284	47097
3000	80	2310	52748
3000	90	2410	57323
3000	100	2690	61409

N=1000



input value	density	fibonacchi heap	SIMPLE scheme
1000	10	620	8237
1000	20	810	15416
1000	30	1053	23070
1000	40	1523	28675
1000	50	1761	36127
1000	60	1875	42108
1000	70	2284	47097
1000	80	2310	52748
1000	90	2410	57323
1000	100	2690	61409

N=5000



input value	density	fibonacci heap	array scheme
5000	10	36615	341137
5000	20	74962	481352
5000	30	107412	597424
5000	40	123981	702570
5000	50	143544	857821
5000	60	155605	1052368
5000	70	186503	1275613
5000	80	189432	1296535
5000	90	189655	1436528
5000	100	202754	1536985