Project-I Report

ADVANCE DATA STRUCTURE - NAME:-ROHIT GARG- UFID:-17622194- rohit.garg@ufl.edu

Deliverables:-

1. edge.java :- The edge node declaration
2. fHeap.java :- Fibonacci heap operations implementation
3. fHeapNode.java :- Fibonacci heap node declaration
4. GenGraph.java :- Generating graph with the given input of number of nodes and density And returns the adjacency list.
5. GraphFile.java :- Read text file and make a graph and return the adjacency list.
6. mstSimple.java :- Simple scheme implementation of prim's MST algorithm.
7. mstFheap.java :- Fibonacci heap scheme implementation of prim's MST algorithm.
8. mst.java :- Main file of the project means that contains main function definition.
9. Project1report.pdf :- Project report
10. Garg_Rohit.zip :- Contains all the source files mentioned above

Compilation:-

- JDK 1.7.25 (Just in time compiler)
- Eclipse IDE
- Extract the Garg_Rohit.zip to Garg_Rohit folder
- Javac *.java (set path variable first of JDK/bin folder)
- JAVA mst 1 n d // JAVA 1 4 6
- JAVA mst 2 filepath, filename must be absolute path.
  // Java 2 "C:\Users\Rohit\workspace\PrimAlgo\src\project1"
  In both cases it will run for both simple scheme and Fibonacci heap scheme

User options

Random Mode

In the random mode I am generating a graph which is going to store as adjacency list with the given input of number of nodes and density (number of edges). Number of edges cannot exceed n*(n-1)/2, where n= number of nodes.

Steps

- Adjacency list data structure declaration :- ArrayList<LinkedList<edge>> neighbors
- To generate an edge set i = random (n), j = random (n) and cost = random (1000) + 1, and add the edge into the graph when edge (i, j) is *not* in the graph. Add edge to the appropriate link list.
- Then check using BFS traversal the graph is connected or not , if not repeat the above two steps
- Return the adjacency list.

How to use the mode

- "JAVA mst 1 n d", here 1 is for random mode, n and d are number of nodes and density respectively.
- It will generate the graph first and then pass the adjacency list to both simple scheme and Fibonacci heap scheme and print the runtime as output.

User Mode

- User need to provide file name as input and it will print at runtime and total weight as output
- JAVA mst 2 filepath , here 2 for user mode , filepath must be absolute
- User need to give the absolute path of the file
- First row must represent the number of nodes and density.

Class definition

Class edge.java

Class edge {

```
	int v1;
	int v2;
	int weight;

	public edge(int a,int b, int c){} // Constructor
}
```

Class fHeapNode.java

```
public class fHeapNode {
	int data;                // Vertex number
	fHeapNode child;        // child field
	fHeapNode left;         // Left sibling
	fHeapNode right;        // Right sibling
	fHeapNode parent;       // Parent node
	int degree;             // no. of childs
	double key;                // key value
	boolean mark;           // if child is removed then its true


	// Constructor
	public fHeapNode(int a , double b ){}
	public double getKey(){}
```

Class fHeap.java

```
public class fHeap {

	private fHeapNode minNode;   //min Node
	private int numNodes;       // Total number of nodes

	public fHeap(){} // its a collection of tree so nothing to intialize
	public boolean checkEmpty(){} //checking empty or not
	public void clear(){} // Clear the heap structure
	public fHeapNode minElement(){} // return minimum element
	public int getSize(){return numNodes;}
	public void insert(fHeapNode node, double x){}//Insert in heap here
	public void decreaseKey(fHeapNode x, double k){}// Decrease key
	private void cascadingCut(fHeapNode tempParent) {} // Cascading cut
	private void cut(fHeapNode x, fHeapNode tempParent) {}// Normal cut
	public fHeap meld(fHeap x, fHeap y){} // Melding two heaps here
	public fHeapNode removeMin(){} // Remove minimum node from heap
	private void pairwiseCombine(){} // After remove min pairwise combine
	private void merge(fHeapNode y, fHeapNode x){} //Merging two fib nodes
	public void delete(fHeapNode x){} // Delete any node in heap
	public fHeapNode search(int x){} //Breath first search in heap
}
```

Class GenGraph.java

```
public class GenGraph {
	private int nodes;
	private int density;
	private ArrayList<LinkedList<edge>> neighbourlist= null; // neighbour
	private int adjMatrix [] [] = null;



	public GenGraph(int a, int b){} // Constructor
	public void makeGraph(){} // Randomly generates graph here
	public boolean checkConnected(){}  //check graph connected or not
	public void print(){} // Print the whole graph
	public ArrayList<LinkedList<edge>> getNeighbourlist(){}//return
adjacency list
	public int[][] getMatrix(){} // return adjacency matrix
	public int size(){} // return size
}
```

Class GraphFile.java

```
public class GraphFile {
      private int size;
      private int density;
      private ArrayList<LinkedList<edge>> neighbourlist =null;


  public void fileRead(String fName) throws IOException {} // File read
  public void print(){}                          //Print adjacency list here
  public ArrayList<LinkedList<edge>> getNeighbourlist(){} //return graph
  public int getSize(){return size;}            // return size
  public int getDensity(){return density;}      // return density

  }
```

Class mstSimple.java

```
public class mst {
      private ArrayList<LinkedList<edge>> neighbour =null;
      private int size=0;

public mst(ArrayList<LinkedList<edge>> object,int x){} // Constructor
public double start(){}                                 //Main algo
private int minVertex(double [] dist, boolean [] v){} //return min cost
vertex at each level
```

Class mstFheap.java

```
public class mstFheap {
      private fHeap f;
      private double totalCost =0;
      private double [] keyList= null;

      //Graph elements
      private int size=0;
      private ArrayList<LinkedList<edge>> neighbour =new
ArrayList<LinkedList<edge>>();
      public ArrayList<fHeapNode> nodes = new ArrayList<fHeapNode>();
      public double cost(){return totalCost;}


      public mstFheap(ArrayList<LinkedList<edge>> object,int size){}
//Constructor
      public void start(){} // Main algorithm
      public void print(){}// Output

      }
```

Class mst.java

```
	public class project1 {
	public static void main(String args[]) throws IOException{} // Main
class
	}
```

Pitfalls

- "Java Heap Size" Out of memory problem for higher density, due to this reason unable to get the performance of that.

Output

- Output in random node comes in parts.
- Run for 10% to 100% for N=1000 and giving me the correct output
- Run for 10% to 70 % for N=3000 and giving me the correct output
- Run for 10% to 30% for N=5000 and giving me the correct output

Findings

N=1000

	Simple scheme run time (ms)	Fibonacci heap run time (ms)
10%	16	31
20%	31	46
30%	46	63
40%	63	93
50%	94	125
60%	140	156
70%	172	219
80%	219	266
90%	312	344
100%	344	375

N=3000

	Simple scheme run time (ms)	Fibonacci heap run time (ms)
10%	125	140
20%	395	437
30%	843	918
40%	1511	1594
50%	2469	2578

60%	3727	3823
70%	5428	5346
80%	7310	7401
90%	-	-
100%	-	-

"-"= Java heap size out of memory

N=5000

Simple scheme run time (ms) Fibonacci heap run time (ms)

10%	501	505
20%	1801	1826
30%	4493	5001
40%	-	-
50%	-	-
60%	-	-
70%	-	-
80%	-	-
90%	-	-
100%	-	-

"-"= Java heap size out of memory