Long Project - 2

PROJECT REPORT

Group – G13
Diksha Sharma (dxs134530) and Sharol Clerit Pereira (scp140130)
Implementation of Advanced Algorithms and Data Structures (CS6301) – Spring 2016
February 21, 2016

Instructor: Dr. Balaji Raghavachari

Introduction

In this project we aim to implement Edmond's Branching Algorithm to find a minimum spanning tree (MST) is a directed graph.

Methodology / Program Execution Steps

Following are the methods in the program:

1. public static Edge returnZeroWeightEdge(Vertex v)
   This method finds if there is any incoming edge to the input vertex with 0 weight. The vertex should be an active vertex in the graph. If there is no edge with 0 weight or no incoming edge at all – this method returns null.
2. public static int countActiveVertices()
   This method counts the number of active vertices in the graph at any moment. We need this method for the logic to verify if all the active vertices are reachable from root vertex using 0 weight edges.
3. public static Edge returnZeroWeightEdgeOutgoing(Vertex v)
   This method returns a zero weight outgoing edge for the input vertex. If no such edge exists then it returns null.
4. public static ArrayList<Vertex> findRemainingVertices(ArrayList<Vertex> verticesList)
   This method returns the list of vertices still not reachable using 0 weight edges from the root vertex. We use the list returned from this method and check it for existence of cycles. The vertices should be active vertices at that point in the graph.
5. public static boolean areAllVerticesReachableFromRoot()
   This method checks if all the active vertices in the graph at any point are reachable from root vertex. If it returns true then the next stage is to start

expanding the contracted cycles. If it returns false, there are more cycles possible in the graph that we should contract before starting to expand the cycles found.

6. public static void findMSTpath(ArrayList<Vertex> vertices)
   This method is called when all the active vertices at any moment in the input graph are reachable from the root vertex. This method adds all those 0 weight active edges in the MST list. We use this list later during expansion phase and replace all edges to or from a super vertex.
7. public static void findZeroWeightCycle(ArrayList<Vertex> verticesList)
   This method walks through the input vertices list and finds a zero weight cycle in it. If a cycle is found then the cycle is passed as argument to contractVertex() method to create a super vertex in the graph and make the vertices and edges part of the cycle inactive. The edges to and from this cycle are added to the new super vertex created except the edges which are within the vertices part of the cycle.
8. public static ArrayList<Edge> removeEdgesNotInCycle(ArrayList<Edge> cycle)
   This method finds the edges that form the cycle in the input edges set. It removes all those vertices which are not repeated in the input edge set.
9. public static void main(String[] args) throws FileNotFoundException
   This method starts the execution of the program. It reads the graph and calls the edmondBranchingMST() method for algorithm execution.
10. public static void reduceWeight(Vertex v)
    This method reduces the weight of all active incoming edges to an active vertex. First we calculate the min weight of all active incoming edges to a vertex. Then we subtract the minimum weight from all the active edges incoming to the input vertex. The value by which we reduce the weight of edges for each vertex is added to get the MST weight for the input graph.
11. public static void edmondBranchingMST()
    This method is the caller for all the methods in the program except main. We check first if we can reach all the vertices the input graph using zero weight edges. If yes then the process is complete. Else, we start by reducing the weight of the edges of the graph and then rechecking if all vertices can be reached from root using zero weight edges. We continue reducing weight, finding cycles in vertices not reachable from root and then contracting them to one super vertex. Once all active vertices can be reached from root, if any cycles were contracted before, we expand them now till all cycles are expanded. If the number of vertices is less than or equal to 50, we output the edges in MST and the weight of MST else we output the weight of the MST.
12. public static void contractVertex(Vertex vStartVertex, ArrayList<Edge> cycle)
    This method contracts the cycle given to it as input. It starts by creating a new super vertex in the graph. Then it makes all the edges in the cycle inactive. The method then iterates through incoming and outgoing edges of all active vertices that have edges to or from the input cycle and add them to the new super vertex we created. All the new edges added are marked as active and once all the edges are added to the super vertex, the vertex is marked as active too. All the vertices that were part of the cycle are marked inactive.
13. public static ArrayList<Edge> rearrangeCycle(Vertex v, ArrayList<Edge> cycle

Rearranges the cycle edges so the first edge starts from input vertex. Also removes the last edge in the cycle so that the cycle is not formed again

14. public static void unContractCycles()
    This method expands the contracted cycles. If an edge in MST is from one vertex to another neither of which are super vertices, then nothing is done. However, if an edge is coming from a super vertex or going to a super vertex, we find the contracted cycle details and add the edges to the MST to expand that vertex.
15. New fields added to Vertex class:
    a. public boolean activeVertex
       Boolean field to show if a node is still active and not contracted
    b. public int minWeight = 0
       Stores the minimum weight of all edges incoming to this vertex
    c. public boolean bContractedVertex
       To indicate if a vertex is a super vertex or not – false for not a super vertex.
16. New fields added to Edge class:
    a. public int OriginalWeight
       Weight of the edge
    b. public boolean activeEdge
       Boolean field to know if this edge needs to be considered for MST calculation or not

Experimental Setup

There are 4 source files in the submission folder:

- Vertex.java – contains the vertex implementation
- Edge.java – contains the edge implementation
- Graph.java – contains the graph implementation. We use only the directed graph components
- Edmond.java – contains the driver program which evaluates the graph and executes the Edmond's Branching Algorithm on the input graph. It generates the weight of minimum spanning tree (MST) with the edges in MST if the number of vertices in the input graph is less than or equal to 50 else it outputs only the weight of MST on system console.

Input arguments:

The driver program takes the full path of the file containing the input graph's details. The input file should have the format as below:

Sample input file:

5 7

1 5 8

1 4 7

1 3 6

4 3 3

3 5 6

5 3 2

5 2 1

The program does not perform any exception handling to determine if the input graph is correct and the details provided in the file are accurate. A correct graph is assumed by the program.

Test Results

For the input file 0-lp2.txt following output (unformatted) is produced:

```
Edges in MST:
(1,4)
(4,3)
(3,5)
(5,2)
Weight of MST:  17
Time: 1 msec.
Memory: 1 MB / 127 MB.
```