CS 6385.0W2 - Algorithmic Aspects of Telecommunication Networks - Su19

Assignment 6

Submitted By: ***Yash Pradhan (ypp170130)***

Problem Statement:

Implement the CMST problem.
1. The modified Kruskal's algorithm
2. The Esau-Williams Algorithm
Compare the results with graphs of various sizes (node/edge combinations). There are no parallel edges.

Your program will take as input
1. vertices
2. costs of each edge
3. any other data (as discussed in the class example)
You may create a graph (incident matrix or Adjacency Matrix or adjacency list).

Run the program for at least 4 configurations.

Your submission should consist of
1. Your code
2. Screen shot of each run of the program for the 4 graphs
3. Report describing your implementation and results - comparison between the two algorithms.

Solution:

The source code resides at src/ypp170130, instructions to compile and run the code are given in the src/readme file, the code has detailed commenting to explain its working.

Implementation Details:

There are two files submitted for this assignment, the Graph.java which is a dependency, it defines the Graph class, underlying Vertex, Edges and AdjacencyList for representation of the graph and WMST.java which contains the implementation of modified kruskal's algorithm and Esau-Williams heuristics, it also contains the driver code.

The driver contains 4 different configuration of the graphs, for running the implemented algorithms. I have included the output screenshots of the execution in this report.

Programming Language Used: Java

The Graph.java defines the Graph used for this assignment. It contains Vertex, Edge and AdjacencyList classes, each to represent corresponding entity in graph. A few of the important fields that graph class has are: the adjacency list, count of vertices and edges, the root vertex and constraint W for constructing the capacitated minimum spanning tree.

The construct() method is a way to supply a graph described as a scanner instance. Use it to construct the graph, the input graph should be a strongly connected undirected graph and should contain no parallel edges. The input format for string representation is

"<V> <E> <W> v0 v1 e0 ….."

V is number of vertices
E is number of Edges
W is the constraint, the max number of vertices in each of sub-clusters

The Vertex class represent instances of vertex in the graph. Each vertex has fields used for book keeping, I have used disjoint set union-find algorithm for maintaining the clusters, the union function efficiently identifies cycles hence preventing the same, connects an edge is the constraints are met also representatives of clusters have their fields updated upon union.

The Edge class represents edges in the graph, we have an undirected graph for these algorithms. Each edge instance has the two endpoints, from and to, and weight and label for each edge. The AdjacencyList has Vertex and its corresponding list of edges. The graph stores the list of these instances and that's how the graph is modelled in my code.

The WMST.java has implementation of modified kruskal's and esau-williams heuristic for computing the capacitated minimum spanning tree. In order to print the steps of execution of algorithm, there is a Boolean variable which can be set to true, otherwise set it to false. Having the graph class as a complete API with required methods ready, the implementation looks pretty similar to the algorithm itself. In the modified kruskals algorithm, once we have the graph created, we sort the edges. We continue until we exhaust all edges or we having a spanning tree generated. The union function takes care whether the edge at hand can be selected or not, making sure that we don't form cycles and we don't violate constraints. At the end, once we have verified that we indeed have a spanning tree, the edges that form the tree are printed along with the weight.

The esau-williams heuristic is implemented in its own function, the initialization is done by connecting each of the vertices to root, there is a method that computes the tradeoff for particular vertex in any iteration. I have used the same formula as in the class slides. The termination criteria is when all the tradeoffs are positive. Until then, we run the iterations, compute the tradeoffs, choose the minimum one, make decision whether that edge could be chosen or not, recompute the tradeoff, this goes on until the algorithm terminates. At the end, the edges that make up the tree are printed along with the weight of the capacitated minimum spanning tree.

Execution Screenshots:

```
YPRADHAN-M-W3Z6:src ypradhan$
YPRADHAN-M-W3Z6:src ypradhan$ java ypp170130/WMST
_____________________________________________
Graph: n: 6, m: 15
0(0) :  (0,1) [5] (0,2) [6] (0,3) [9] (0,4) [12] (0,5) [15]
1(1) :  (1,3) [3] (1,2) [4] (0,1) [5] (1,4) [8] (1,5) [10]
2(1) :  (1,2) [4] (2,4) [5] (0,2) [6] (2,3) [8] (2,5) [12]
3(1) :  (1,3) [3] (3,4) [6] (3,5) [6] (2,3) [8] (0,3) [9]
4(1) :  (2,4) [5] (3,4) [6] (4,5) [7] (1,4) [8] (0,4) [12]
5(1) :  (3,5) [6] (4,5) [7] (1,5) [10] (2,5) [12] (0,5) [15]
_____________________________________________

Modified Kruskals Algorithm



Results:
Spanning Tree Edges:
(1,3) [3]
(1,2) [4]
(0,1) [5]
(4,5) [7]
(0,4) [12]

Weight: 31
_____________________________________________

Esau Williams Heuristic

Results:
Spanning Tree Edges:
(1,3) [3]
(0,1) [5]
(2,4) [5]
(0,2) [6]
(3,5) [6]

Weight: 25
```

Slide Example

```
-------------------------------------------
Graph: n: 7, m: 21
0(0) :  (0,1) [5] (0,2) [6] (0,3) [9] (0,4) [10] (0,5) [11] (0,6) [15]
1(1) :  (0,1) [5] (1,2) [9] (1,3) [6] (1,4) [6] (1,5) [8] (1,6) [17]
2(1) :  (0,2) [6] (1,2) [9] (2,3) [7] (2,4) [9] (2,5) [8] (2,6) [12]
3(1) :  (0,3) [9] (1,3) [6] (2,3) [7] (3,4) [10] (3,5) [5] (3,6) [11]
4(1) :  (0,4) [10] (1,4) [6] (2,4) [9] (3,4) [10] (4,5) [14] (4,6) [9]
5(1) :  (0,5) [11] (1,5) [8] (2,5) [8] (3,5) [5] (4,5) [14] (5,6) [8]
6(1) :  (0,6) [15] (1,6) [17] (2,6) [12] (3,6) [11] (4,6) [9] (5,6) [8]
-------------------------------------------

Modified Kruskals Algorithm



Results:
Spanning Tree Edges:
(0,1) [5]
(3,5) [5]
(0,2) [6]
(1,3) [6]
(2,4) [9]
(4,6) [9]

Weight: 40
-------------------------------------------

Esau Williams Heuristic

Results:
Spanning Tree Edges:
(0,1) [5]
(3,5) [5]
(0,2) [6]
(0,3) [9]
(5,6) [8]
(1,4) [6]

Weight: 39
-------------------------------------------
```

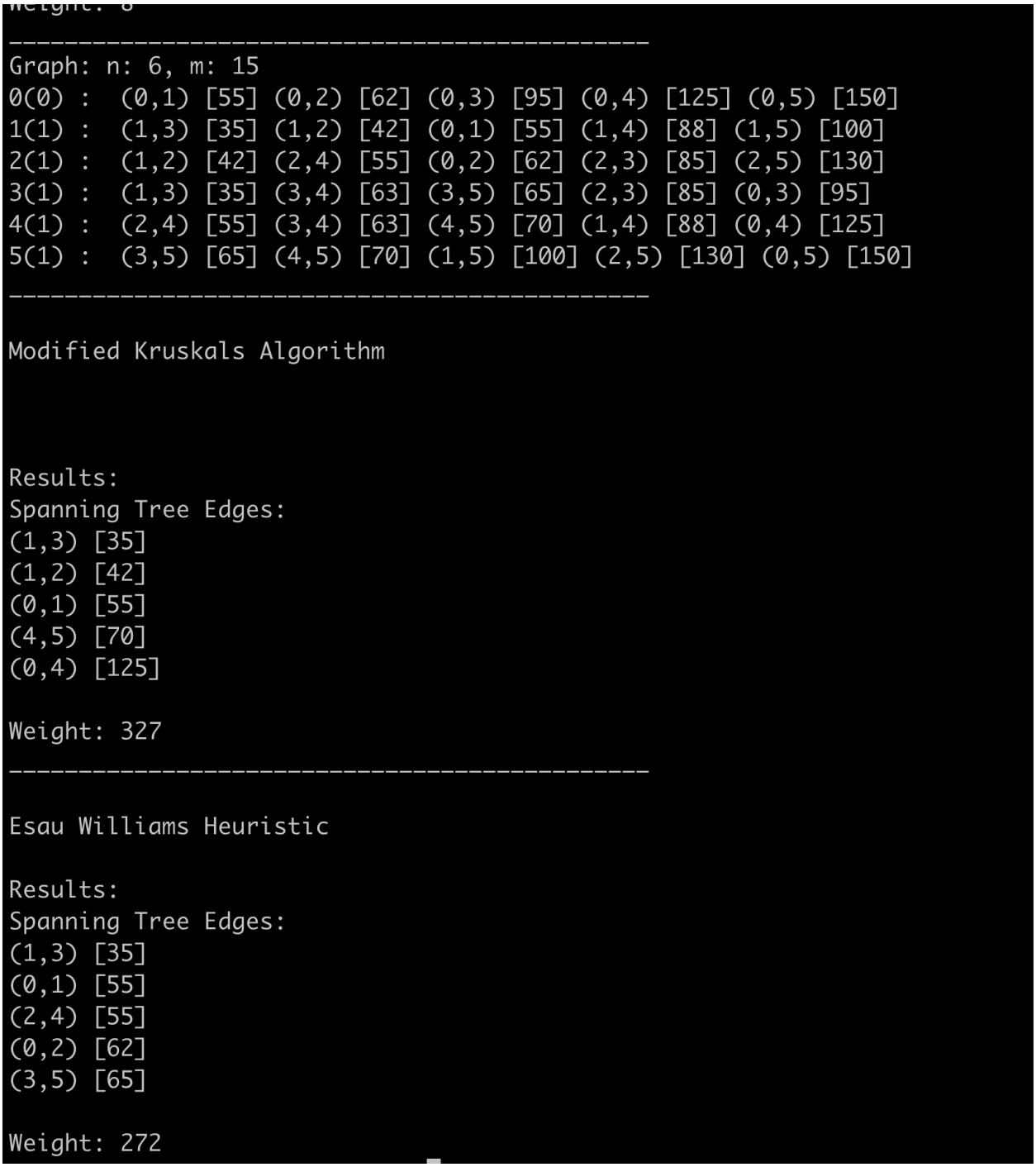
Handout Example

Link: http://www.pitt.edu/~dtipper/2110/CMST_example.pdf

```
Graph: n: 7, m: 21
0(0) :  (0,1) [2] (0,2) [10] (0,3) [10] (0,4) [2] (0,5) [10] (0,6) [10]
1(1) :  (0,1) [2] (1,2) [1] (1,3) [10] (1,4) [10] (1,5) [10] (1,6) [10]
2(1) :  (0,2) [10] (1,2) [1] (2,3) [1] (2,4) [10] (2,5) [10] (2,6) [10]
3(1) :  (0,3) [10] (1,3) [10] (2,3) [1] (3,4) [10] (3,5) [10] (3,6) [10]
4(1) :  (0,4) [2] (1,4) [10] (2,4) [10] (3,4) [10] (4,5) [1] (4,6) [10]
5(1) :  (0,5) [10] (1,5) [10] (2,5) [10] (3,5) [10] (4,5) [1] (5,6) [1]
6(1) :  (0,6) [10] (1,6) [10] (2,6) [10] (3,6) [10] (4,6) [10] (5,6) [1]
-------------------------------------------

Modified Kruskals Algorithm



Results:
Spanning Tree Edges:
(1,2) [1]
(2,3) [1]
(4,5) [1]
(5,6) [1]
(0,1) [2]
(0,4) [2]

Weight: 8
-------------------------------------------

Esau Williams Heuristic

Results:
Spanning Tree Edges:
(0,1) [2]
(4,5) [1]
(0,4) [2]
(5,6) [1]
(1,2) [1]
(2,3) [1]

Weight: 8
```

My Own Example

Both were able to determine the best tree in this case.



Random Weights on the Slides Example

Conclusion:

As we see the general performance of esau-williams is better than kruskal's algorithm. But this is a hard problem, and Esau Williams is just a heuristic, it does not guarantee finding an optimal solution :)

References:

- Class Notes/Slides
- Example: http://www.pitt.edu/~dtipper/2110/CMST_example.pdf
- Reference Material: http://www.pitt.edu/~dtipper/2110/Slides6.pdf