Greg Becker
Programming Assignment 6
29/04/2017

Abstract

For programming assignment 6 we had to implement an algorithm written in psuedo-code to create a minimum spanning tree using text files used in lab 4. The algorithm given to us utilized a stack, rather than recursion. To solve this problem some code could be re-used from lab 4. Although we did not recycle the code directly, as its purpose is very different, the method reDFS in lab 4 was looked at as a reference to grasp the algorithm given in the descriptions to the problem of assignment 6.

The msp method, which actually creates the minimum spanning tree, begins by processing the first city and pushing it on the stack. After starting off the problem a loop runs until everything placed on the stack—edges selected in the minimum spanning tree—is removed. Within this loop the lowest value for each distance between cities is computed and selected. Any distance between two cities that is smaller than the current minimum cost (current as in while all of the costs have not been accounted for) is added to the stack. While this loop runs each next step in the final route is printed on screen. It concludes by checking if what is at the top of the stack is the first city and then produces this as the ending city. This method was a lot less heavy than reDFS. Comparisons to the determine the smallest route between two cities were made in nested for loops. No other methods (that we wrote) had to be used. The most striking observation was how quickly minimum spanning trees can be created. To create a tree based on data in TSP29.txt took one second. To use the same data to solve the traveling salesmen problem would take more than a lifetime.

```
TSP12.txt
run:
Start City: 0
closest city 5
closest city 7
closest city 4
closest city 9
closest city 1
closest city 8
closest city 10
closest city 3
closest city 11
closest city 2
closest city 6
End city: 0
BUILD SUCCESSFUL (total time: 1 second)
```

```
TSP13.txt
run:
Start City: 0
closest city 5
closest city 7
closest city 4
closest city 9
closest city 1
closest city 8
closest city 10
closest city 12
closest city 3
closest city 11
closest city 2
closest city 6
End city: 0
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
TSP14.txt
run:
Start City: 0
closest city 5
closest city 7
closest city 4
closest city 9
closest city 1
closest city 8
closest city 10
closest city 12
closest city 3
closest city 11
closest city 2
closest city 6
closest city 13
End city: 0
BUILD SUCCESSFUL (total time: 0 seconds)

TSP15.txt
run:
Start City: 0
closest city 5
closest city 7
closest city 4
closest city 9
closest city 1
closest city 8
closest city 10
closest city 12
closest city 14
closest city 3
closest city 11
closest city 2
closest city 6
closest city 13
End city: 0
BUILD SUCCESSFUL (total time: 0 seconds)

TSP16.txt
run:
Start City: 0
closest city 5
closest city 7
closest city 12
closest city 4
closest city 11
closest city 1
closest city 8
closest city 9
closest city 3
closest city 14
closest city 2
closest city 6
closest city 13
closest city 10
closest city 15
End city: 0
BUILD SUCCESSFUL (total time: 0 seconds)

TSP19.txt
run:
Start City: 0
```

```
closest city 5
closest city 7
closest city 12
closest city 4
closest city 11
closest city 1
closest city 15
closest city 8
closest city 9
closest city 3
closest city 18
closest city 14
closest city 2
closest city 17
closest city 6
closest city 13
closest city 10
closest city 16
End city: 0
BUILD SUCCESSFUL (total time: 1 second)

run:
Start City: 0
closest city 27
closest city 23
closest city 20
closest city 5
closest city 7
closest city 12
closest city 26
closest city 4
closest city 11
closest city 1
closest city 15
closest city 8
closest city 9
closest city 19
closest city 25
closest city 3
closest city 28
closest city 18
closest city 22
closest city 14
closest city 24
closest city 2
closest city 17
closest city 6
closest city 13
closest city 10
closest city 21
closest city 16
End city: 0
BUILD SUCCESSFUL (total time: 1 second)
```