Programming Assignment 6

Trevor Goodwin

C202-Fall 2016

12-10-2016

In programming assignment 6 we are tasked with finding the shortest path tree of an adjacency matrix (Traveling Salesman Problem). For this assignment, one can recycle the "Lab 5" text files that allow us to build the two-dimensional adjacency matrix mentioned above. The method used before differs in the fact that it builds paths based solely on cost and will not display the output data until the lowest cost has been found. The spanning tree has a much faster running time since it is not actually dependent on cost; rather it is dependent on the distance between nodes.

The algorithm for this program that is given uses a data structure called a "Stack" instead of the original DFS (Depth First Search) which used a data structure called an "ArrayList (An array of unbounded size)". The algorithm then uses an array that only evaluates to two outputs (True or False, i.e. Boolean). This tells the loop that we will create later on if a certain city has already been added to the path tree ("Stack") or not. We then introduce a loop that increments only while the "Stack" is has values to examine, and inside this loop we introduce specific expressions that set Boolean values as well as values to the "Stack". We also create a variable that contains the largest numerical integer value and make it smaller as we go along till we have a satisfactory result. Once our loops conditions can no longer be met, we pop all the values out of the "Stack" until its empty and we are kicked completely out of the loop.

After running this program, the results were adequately surprising. For instance, when running with twelve cities, there was not much of a difference, but as the number of cities increased, the exhaustive tasks thrown at the CPU at Lab 5 were no match for the efficiency of this Programming Assignment. The actual execution times of this programming assignment only differed by only 600,000 nanoseconds starting with 12 cities and going to 29 cities. This is understandable as the complexity of lab five increases exponentially with every new city added. Below I will add my time comparisons of Lab 5 versus Programming Assignment 6:

- 12 Cities: (L5: 4 seconds) (PA6: 370965 Nanoseconds)
- 13 Cities: (L5: 12 seconds) (PA6: 413561 Nanoseconds)
- 14 Cities: (L5: 13 seconds) (PA6: 397705 Nanoseconds)
- 15 Cities: (L5: 31 seconds) (PA6: 389778 Nanoseconds)
- 16 Cities: (L5: 13 Minutes 21 Seconds) (PA6: 412240 Nanoseconds)
- 19 Cities: (L5: Ran for 3 days without output) (PA6: 475001 Nanoseconds)
- 29 Cities: (L5: Ran for 1 week without output) (PA6: 1160747 Nanoseconds)

Actual screen output listed below (NEXT PAGE)

```
run:
Help! I need a number of cities!
12 tsp
Type 'tsp':
0 5 3 8 4 1 11 6 7 10 9 2
Cost: 2690
Time taken to run: 379208 nanoseconds.
BUILD SUCCESSFUL (total time: 2 seconds)

run:
Help! I need a number of cities!
13 tsp
Type 'tsp':
0 5 3 8 4 1 11 6 7 10 9 2 12
Cost: 2779
Time taken to run: 524549 nanoseconds.
BUILD SUCCESSFUL (total time: 3 seconds)

run:
Help! I need a number of cities!
14 tsp
Type 'tsp':
0 5 3 8 4 1 13 11 6 7 10 9 2 12
Cost: 3521
Time taken to run: 375244 nanoseconds.
BUILD SUCCESSFUL (total time: 1 second)

run:
Help! I need a number of cities!
15 tsp
Type 'tsp':
0 5 3 8 4 1 13 14 12 2 9 10 7 6 11
Cost: 3874
Time taken to run: 528183 nanoseconds.
BUILD SUCCESSFUL (total time: 4 seconds)
```

```
run:
Help! I need a number of cities!
16 tsp
Type 'tsp':
0 5 11 8 4 1 9 3 14 13 10 15 12 7 6 2
Cost: 5935
Time taken to run: 414222 nanoseconds.
BUILD SUCCESSFUL (total time: 4 seconds)

run:
Help! I need a number of cities!
19 tsp
Type 'tsp':
0 5 11 8 4 1 9 3 14 18 15 12 7 6 10 13 17 16 2
Cost: 6812
Time taken to run: 506382 nanoseconds.
BUILD SUCCESSFUL (total time: 1 second)

run:
Help! I need a number of cities!
29 tsp
Type 'tsp':
0 27 5 11 8 4 20 1 19 9 3 14 18 24 6 22 26 23 7 15 12 17 13 21 16 10 28 25 2
Cost: 11258
Time taken to run: 1157443 nanoseconds.
BUILD SUCCESSFUL (total time: 2 seconds)
```