Cody Main

Programming Assignment 6

Traveling Salesman Problem (Using Dijkstra's Algorithm)

December 2, 2016

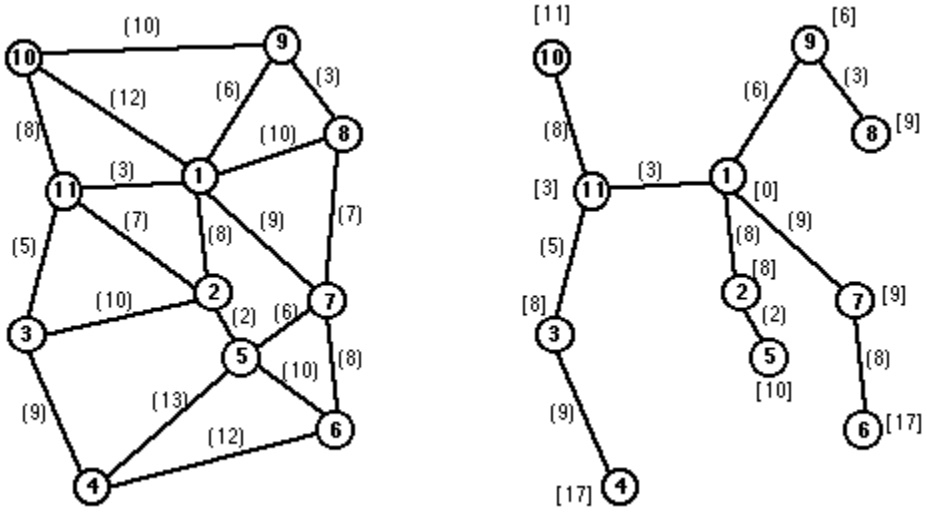
Analysis:

In this particular programming assignment, the objective is to solve the traveling salesman problem using Dijkstra's Algorithm instead of using a Recursive Depth First Search. The objective is to show the time differences between the two algorithms for solving the traveling salesman problem.

In order to successfully complete this assignment, the program must read in a file, place the values into an adjacency matrix, perform Dijkstra's algorithm, write out the shortest path between cities according to the starting city, and list the time taken to complete the operations.

Design:

The design of the Binary Search Tree is displayed below…



This illustration shows how Dijkstra's Algorithm is designed. The algorithm is constructed using a boolean array of visited cities. For each city that is visited that city's index becomes "true" and it does not need to be visited anymore. That is why the paths to each city are not in a series. The algorithm will check each city and choose the shortest path is that city has not been visited. If the city has been visited, it will keep checking to find cities that have not been visited.

The shortest path is set as a minimum value of a city and each city is placed into a stack. When a city's shortest path has been found, the city will pop out of the stack. This continues until the stack is empty and all the cities have been marked as visited in the array.

Observations:

After implementing all of the coding for this assignment, there are two observances that can be noted in the output that are important contrasts to using a Recursive Depth First Search to solve this problem…

1) The time taken to solve the problem is drastically reduced. This is because of the time complexities of Dijkstra's Algorithm compared to that of the Recursive Depth First Search. The time complexity of Dijkstra's Algorithm is n^2 (n squared). This is much different and more efficient than the time complexity of n! (n factorial).

2) The shortest path between cities may not be the shortest path overall between cities when using Dijkstra's Algorithm. This is because Dijkstra's Algorithm is a greedy algorithm. A greedy algorithm is an algorithm that decides the best option for its current state when running a program compared to the overall best option. Using a Recursive Depth First Search traverses every possible combination of paths between cities to find the true shortest distance between all the cities, whereas Dijkstra's Algorithm only finds the shortest distance between all the cities depending on the location of the first city.

Output:

The output has been displayed below according to the assignment specifications…

```
/**
 * Output of the program for each file read.
 *
 *
 * tsp12.txt
 *
 * run:
 * Start City 0
 * Closest City: 5
 * Closest City: 3
 * Closest City: 8
 * Closest City: 4
 * Closest City: 1
```

* Closest City: 11

* Closest City: 6

* Closest City: 7

* Closest City: 10

* Closest City: 9

* Closest City: 2

* Time to Complete: 0.003323907 seconds.

* BUILD SUCCESSFUL (total time: 0 seconds)

*

*

*

*

* tsp13.txt

*

* run:

* Start City 0

* Closest City: 5

* Closest City: 3

* Closest City: 8

* Closest City: 4

* Closest City: 1

* Closest City: 11

* Closest City: 6

* Closest City: 7

* Closest City: 10

* Closest City: 9

* Closest City: 2

* Closest City: 12

* Time to Complete: 0.003392437 seconds.

* BUILD SUCCESSFUL (total time: 0 seconds)
*
*
*
*
* tsp14.txt
*
* run:
* Start City 0
* Closest City: 5
* Closest City: 3
* Closest City: 8
* Closest City: 4
* Closest City: 1
* Closest City: 13
* Closest City: 11
* Closest City: 6
* Closest City: 7
* Closest City: 10
* Closest City: 9
* Closest City: 2
* Closest City: 12
* Time to Complete: 0.009684846 seconds.
* BUILD SUCCESSFUL (total time: 0 seconds)
*
*
*
*
* tsp15.txt

*

* run:

* Start City 0

* Closest City: 5

* Closest City: 3

* Closest City: 8

* Closest City: 4

* Closest City: 1

* Closest City: 13

* Closest City: 14

* Closest City: 12

* Closest City: 2

* Closest City: 9

* Closest City: 10

* Closest City: 7

* Closest City: 6

* Closest City: 11

* Time to Complete: 0.003134862 seconds.

* BUILD SUCCESSFUL (total time: 0 seconds)

*

*

*

*

* tsp16.txt

*

* run:

* Start City 0

* Closest City: 5

* Closest City: 11

* Closest City: 8

* Closest City: 4

* Closest City: 1

* Closest City: 9

* Closest City: 3

* Closest City: 14

* Closest City: 13

* Closest City: 10

* Closest City: 15

* Closest City: 12

* Closest City: 7

* Closest City: 6

* Closest City: 2

* Time to Complete: 0.003877814 seconds.

* BUILD SUCCESSFUL (total time: 0 seconds)

*

*

*

*

* tsp19.txt

*

* run:

* Start City 0

* Closest City: 5

* Closest City: 11

* Closest City: 8

* Closest City: 4

* Closest City: 1

* Closest City: 9

* Closest City: 3

* Closest City: 14

* Closest City: 18

* Closest City: 15

* Closest City: 12

* Closest City: 7

* Closest City: 6

* Closest City: 10

* Closest City: 13

* Closest City: 17

* Closest City: 16

* Closest City: 2

* Time to Complete: 0.003263413 seconds.

* BUILD SUCCESSFUL (total time: 0 seconds)

*

*

*

*

* tsp29.txt

*

* run:

* Start City 0

* Closest City: 27

* Closest City: 5

* Closest City: 11

* Closest City: 8

* Closest City: 4

* Closest City: 20

* Closest City: 1

* Closest City: 19

* Closest City: 9

* Closest City: 3

* Closest City: 14

* Closest City: 18

* Closest City: 24

* Closest City: 6

* Closest City: 22

* Closest City: 26

* Closest City: 23

* Closest City: 7

* Closest City: 15

* Closest City: 12

* Closest City: 17

* Closest City: 13

* Closest City: 21

* Closest City: 16

* Closest City: 10

* Closest City: 28

* Closest City: 25

* Closest City: 2

* Time to Complete: 0.003935473 seconds.

* BUILD SUCCESSFUL (total time: 0 seconds)

*

*/