Programming Assignment 6

Name: Dewey Kincheloe

Assignment Number: Programming Assignment 6

Assignment Title: Programming Assignment 6

Due Date: December 2, 2016

Programming Assignment 6

Abstract

Problem Description:

This program solves the Traveling Sales Person problem. It inputs the number of cities the sales person is to visit and the file that contains the “cost” to travel to each of these cities from the starting location. The goal of this program is to calculate the cheapest path that allows the sales person to visit each of the cities only once and return to the starting city then output that cost as well as a list of the cities visited.

The file that contains the “cost” to each city is read and used to populate the adjacency matrix.

I’m still working on this assignment.

```
/**
 * C202 Assignment 6
 * asst6.java
 * Purpose: This program explores the algorithms for computing the tour cost of
 * a traveling salesman problem.
 *
 * @author Dewey Kincheloe
 * @version 1.0 12/2/2016
 */

import java.util.*;
import java.io.*;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;
import java.util.Stack;
import java.io.File;
import java.io.FileNotFoundException;

public class asst6 <E extends Comparable<E>>
{
   //Attributes
   private int CITI; // Can't create until the number of cities is known
   private int[][] adjacency;
   private int bestcost = Integer.MAX_VALUE;
   private ArrayList<Integer> bestpath;
   private Stack<Integer> pathStack;
   private int city, startCity, closestCity, currentCity, min = 0;
   //public int [] visitedCities;
```

```
private Boolean minFlag = true;


//Constructors
public asst6(int number)
{
        CITI = number;
        adjacency = new int[CITI][CITI];
        bestpath = new ArrayList<>();
    pathStack = new Stack<>();
}
// The populateMatrix constructor populates the fname file.
public void populateMatrix(String fname)
{
        File f = new File(fname);
        try
    {
        Scanner input = new Scanner(f);
        int i,j;
        for(i = 0; i < CITI && input.hasNext(); i++)
        {
                for(j = i; j < CITI && input.hasNext(); j++)
           {
              if(i == j)
              {
                        adjacency[i][j] = 0;
              }
              else
              {
                        int value = input.nextInt();
                        adjacency[i][j] = value;
```

```
                        adjacency[j][i] = value;
                }
                    }
        }
        input.close();
    }
        catch(IOException e)
    {
        System.out.println("File reading failed!");
        }
}


//Methods
//The cost method receives an integer array and calculates the path cost.
public int cost(ArrayList<Integer> path)
{
    int cost = 0;
        for(int i = 0; i < path.size()-1; i++)
    {
        cost += adjacency[path.get(i)][path.get(i+1)];
    }
        if(path.size() == CITI)
    {
        cost += adjacency[path.get(path.size()-1)][0];
        }
        return cost;
}
// The tspdfs method receives an integer array and an arraylist (cities visited
// and to be visited) then recursively searches for a better path with a cheaper cost.
```

```
    public void tspdfs(ArrayList<Integer> partialTour, ArrayList<Integer>
remainingCities)
    {
        if(remainingCities.size() == 0)
      {
        int tourCost = cost(partialTour);
        if(tourCost < bestcost)
        {
          bestcost = tourCost;
          bestpath = new ArrayList<>(partialTour);
        }
      }
      else
      {
        for(int i = 0; i < remainingCities.size(); i++)
        {
          ArrayList<Integer> temp = new ArrayList<>(partialTour);
          temp.add(remainingCities.get(i));
          int tourCost = cost(temp);
          if(tourCost < bestcost)
          {
            ArrayList<Integer> temp2 = new ArrayList<>(remainingCities);
            temp2.remove(i);
            tspdfs(temp, temp2);
          }
        }
      }
    }

    public boolean notVisited(int [] visitedCities, int checkIfVisited)
```

```
{
   for(int i = 0; i < visitedCities.length; i++)
   {
      if(visitedCities[i] == checkIfVisited)
      {
         return false;
      }
   }
   return true;
}

public void addVisited(int[] visitedCities, int closestCity)
{
   int i = 0;
   while(visitedCities[i] != -1 && i < visitedCities.length)
   {
      i++;
   }
   visitedCities[i] = closestCity;
}

public void pathSearch(int[] visitedCities, int number)
{
   pathStack.push(startCity);
   closestCity = startCity;
   minFlag = false;

   System.out.println("Starting City = 0");
   while(!pathStack.isEmpty())
   {
```

```
        currentCity = pathStack.peek();
        min = Integer.MAX_VALUE;
        for(int i = 1; i < number; i++)
        {
           if(adjacency[currentCity][i] != 0 && notVisited(visitedCities, i))
           {
              if(adjacency[currentCity][i] < min)
              {
                 min = adjacency[currentCity][i];
                 closestCity = i;
                 minFlag = true;
              } //end if
           } //end if
        } // end for

        if(minFlag)
        {
           addVisited(visitedCities, closestCity);
           pathStack.push(closestCity);
           minFlag = false;
        } //end if

     pathStack.pop();
     } //endwhile
}

//The main method controls program execution.
public static void main(String[] args)
{
```

```
Scanner in = new Scanner(System.in);
System.out.println("Enter number of cities ");
int number = in.nextInt();
System.out.println("Enter the file name ");
String input = in.next();
   asst6 tsp = new asst6(number);

tsp.populateMatrix(input);

int[] visitedCities = new int[number];
visitedCities[0] = 0;
for(int i = 0; i < number; i++)
{
   visitedCities[i] = -1;
   }

tsp.pathSearch(visitedCities, number);

   ArrayList<Integer> partialT = new ArrayList<>();
   partialT.add(0);
   ArrayList<Integer> remainingT = new ArrayList<>();
   for(int i = 1; i < number; i++)
{
   remainingT.add(i);
   }
long sum = 0;
long start = System.nanoTime();
   tsp.tspdfs(partialT, remainingT);
long stop = System.nanoTime();
   sum += (stop-start);
```

```
            System.out.println(sum);
            tsp.output();
        }

            // The output method displays the best path and cost.
            public void output()
            {
                System.out.println();
            System.out.println("Best path and cost");
                for(int i = 0; i < bestpath.size(); i++)
            {
                System.out.println(bestpath.get(i) + " ");
                }
            System.out.println("cost =  " + bestcost);
        }
    }
```

```
run:
Enter number of cities
12
Enter the file name
tsp12.txt
Starting City = 0
1480302373

Best path and cost
0
5
3
9
```

10

8

4

2

1

11

6

7

cost = 821

BUILD SUCCESSFUL (total time: 10 seconds)

run:

Enter number of cities

13

Enter the file name

tsp13.txt

Starting City = 0

5416769834

Best path and cost

0

5

3

9

10

8

4

12

2

1

11

6

7

cost = 864

BUILD SUCCESSFUL (total time: 13 seconds)

run:

Enter number of cities

14

Enter the file name

tsp14.txt

Starting City = 0

34511632875

Best path and cost

0

5

3

8

4

12

2

9

10

13

1

11

6

7

cost = 958

BUILD SUCCESSFUL (total time: 58 seconds)

run:

Enter number of cities

15

Enter the file name

tsp15.txt

Starting City = 0

67611305512

Best path and cost

0

5

3

9

10

8

4

2

12

14

13

1

11

6

7

cost = 876

BUILD SUCCESSFUL (total time: 1 minute 16 seconds)

run:

Enter number of cities

16

Enter the file name

tsp16.txt

Programming Assignment 6

Starting City = 0

2885300040727

Best path and cost

0

5

11

8

4

2

1

9

12

3

14

13

10

6

15

7

cost = 1222

BUILD SUCCESSFUL (total time: 48 minutes 12 seconds)

run:

Enter number of cities

19

Programming Assignment 6

Enter the file name

tsp19.txt

Starting City = 0