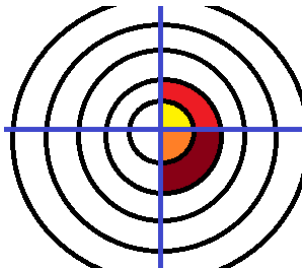
Koray Emre ŞENEL
150117037

CSE2046 ANALYSIS OF ALGORITHM ASSIGNMENT 3 REPORT

In this assignment, we are asked to design a algorithm to solve travelling salesman problem.

First part of code reads the input file and writes it into 2 arraylist(One for to save the original inputs to use it later). This part also sum x and y's of the objects that called as cities. The sum is done to find mean of the cities x and y's, and this is used to put the cities in a coordinate system.



After reading the inputs, by the help of the mean mover method rearrenges the x and y's of cities and set the radius and quadrant for all of them. In my algorithm, I need radius to divide them and quadrants to get better results. I tried to show what I imagine to do by dividing in this painting. The darker the color goes the later they got added in path. To do the dividing I needed an expression and I write some random expressions and check their growth. I like the $\log_{10}(n)^{\log(n)}$most but while improving the project changed it to $\log_{10}(n)^{\log(n)}/ (n)^{1/2}$.

Then there is method called arrSplit to make dividing and put the cities that in range to an arraylist.

```
while(current < inputArr.get(size-1).R + range ) {
    double xytotals [][] = new double[2][4];
    while(inputArr.get(i).R < current + range && close1) {
        switch(j) {
        case (0):
            if (inputArr.get(i).X >= 0) {
                Cities objt = new Cities(inputArr.get(i).ID,inputArr.get(i).X,inputArr.get(i).Y);
                cityArr0.get(inputArr.get(i).quadrant).add(objt);
                xytotals[0][inputArr.get(i).quadrant] += inputArr.get(i).X;
                xytotals[1][inputArr.get(i).quadrant] += inputArr.get(i).Y;
            }
            break ;
        case (1):
            if (inputArr.get(i).X < 0) {
```

I wanted to find better solution so dividing the half is done for 4 times these are the starting cities x's are bigger or equal to 0 , y's are bigger or equal to 0 , x's are less than 0 and y's are less than 0. I said starting cities because as u see in the coordinate system I used only the right side of system. I left left side to coming back to the starting point. So the left side must start from the furthest circle and get closer to origin.

```
while(currentStart < inputArr.get(size-1).R + range ) {
    double xytotals [][] = new double[2][4];
while(inputArr.get(size-k).R > current2-range && close2 ) {
    switch(j) {
    case (0):
        if (inputArr.get(size-k).X < 0) {
            Cities objt = new Cities(inputArr.get(size-k).ID,inputArr.get(size-k).X,inputArr.get(size-k).Y);
            cityArr1.get(inputArr.get(size-k).quadrant).add(objt);
            xytotals[0][inputArr.get(size-k).quadrant] += inputArr.get(size-k).X;
```

These iterations are done after putting the cities in the range in a arraylist . They are also seperated by their quadrants . The mover is used again to find cities close to each other. lastCity is a cities object and used to maintain the path from last of the path that found. The firstTime boolean checks if it is the first time of the getting back to started city . And it allows to use lastCity .

```
for (int a = 0 ; a<4 ; a++) {
if(!cityArr0.get(a).isEmpty()) {
mover(cityArr0.get(a),xytotals[0][a],xytotals[1][a]);
lastCity = pathConstruct(cityArr0.get(a),lastCity,path);
}
for(int a = 0 ; a<4 ;a++)
if(!cityArr1.get(a).isEmpty()) {
    mover(cityArr1.get(a),xytotals[0][a],xytotals[1][a]);
    if(firstTime) {
        lastCityBack = pathConstruct(cityArr1.get(a),lastCity,pathBack);
        firstTime = false;
        continue;
    }
    lastCityBack = pathConstruct(cityArr1.get(a),lastCityBack,pathBack);
    }
```

The pathConstruct method is used to create the paths. In the start of the method it checks the lastCity and finds the closest to that in the arraylist. After finding the closest of lastCity it adds it to path. The added citiys index is stored and it search the nearest to that city by looking to arraylist.

After finding the nearest city,its index is stored az destination and it also added to path. By removing the first city that found it blocks the writing a city more then once. Then it does the same things until arraylist is cleared. The method returns the last element of the path as lastCity to use it in another call.

```
while(arr.size()>1) {
    int destination = 0 ;
    min = 999999999;
    int i = 0 ;
while(index + i < arr.size()-1) {
    i++;
    if(i == 100)
        break;
}
for(int back = index+i  ; back >= 0;back-- ) {
    if(back == index ) continue;
    if(distance(arr.get(index),arr.get(back)) < min) {
            min = distance(arr.get(index),arr.get(back));
            destination = back;
        }
    }
path.add(inputArrStart.get((int) arr.get(destination).ID));
arr.remove(index);
    if(destination < index)
index = destination;
    else
        index = destination-1;
}
arr.remove(0);
return path.get(path.size()-1);
```

At the end of this operations, we ended up with two arraylist that has a path in it so we have to combine them into one. To do this there is a method called pathCombine. It combines the two path one that starts from one half near to origin and other is other halfs further to origin. As I mention before this is only 1 of the 4 paths so I have to check their lengths and find the nearest to optimal. This basic method used for this job.

```
private static int[] pathLengthcheck(ArrayList<ArrayList> pathArr) {
    int length = 0 ;
    int min = 999999999;
    int returns[] = new int [2];
    int best = 0;
    int pathLength[] = new int[4];
    for (int i = 0 ; i<4 ; i++) {
        int size = pathArr.get(i).size();
        length = 0 ;
        for(int j = 0 ;j<size-1;j++) {
            int dist = distance((Cities) pathArr.get(i).get(j),(Cities) pathArr.get(i).get(j+1));
            length = length + dist;
        }
        int getBack = distance((Cities)pathArr.get(i).get(size-1),(Cities)pathArr.get(i).get(0));
        length = length + getBack;
        pathLength[i] = length;
    }
    for (int i = 0 ; i<4 ; i++) {
        if(pathLength[i] < min) {
        min = pathLength[i];
        best = i;
    }
    }
    returns[0] = best;
    returns[1] = pathLength[best];
    return returns;

}
```

```
private static void outputPrint(ArrayList<Cities> arr, int length) throws IOException {
     FileWriter writer = new FileWriter(txtOutputName);
     writer.write((length + System.lineSeparator()));
    for(int i = 0 ; i<arr.size() ; i++)
        writer.write(((int) arr.get(i).ID) + System.lineSeparator());

      writer.close();
}
```

To give the output I used this method.

Solution verifier results.

```
C:\Users\koray>cd C:\Users\koray\eclipse-workspace\Algorithm

C:\Users\koray\eclipse-workspace\Algorithm>python tsp-verifier.py test-input-4.txt test-output-4.txt
solution found of length  13484

C:\Users\koray\eclipse-workspace\Algorithm>python tsp-verifier.py test-input-3.txt test-output-3.txt
solution found of length  87661564

C:\Users\koray\eclipse-workspace\Algorithm>python tsp-verifier.py test-input-2.txt test-output-2.txt
solution found of length  338801

C:\Users\koray\eclipse-workspace\Algorithm>python tsp-verifier.py test-input-1.txt test-output-1.txt
solution found of length  3463

C:\Users\koray\eclipse-workspace\Algorithm>python tsp-verifier.py example-input-1.txt example-output-1.txt
solution found of length  153995

C:\Users\koray\eclipse-workspace\Algorithm>python tsp-verifier.py example-input-2.txt example-output-2.txt
solution found of length  3463

C:\Users\koray\eclipse-workspace\Algorithm>python tsp-verifier.py example-input-3.txt example-output-3.txt
solution found of length  2081530
```

In arrSplit method there is a loop that calls a method that uses loop so I think the time complexity of the code is $O(n^2)$.