Kian Afkhami

CMSC 207

Final Project

Learning Experience:

This was the first time I had used a nearest neighbor algorithm or tried to solve a problem with a nearest neighbor algorithm or a new algorithm. It was interesting not only figuring out the solutions but also how different algorithms have different outcomes. I ended up creating 2 programs: a nearest neighbor algorithm and another algorithm I will call the method of exhaustion. It finds all possible scenarios of the algorithm and sorts them to find the shortest path. It actually came up with a path that is shorter than the nearest neighbor algorithm!

The most difficult part of this project was creating a new algorithm. Once I came up with the idea, I had to come up with how to store all the different scenarios, then how to find the one with the minimum distance and how to find which data ended up creating the most ideal scenario. The hardest part was figuring out how to generate all the scenarios. I ended up using a recursive method to generate all the possible scenarios.

Overall, I learned a lot about algorithms when completing this project. I think I made the method of exhaustion algorithm overly complicated, by switching variables between int, char, and String. If I were to do it again, I would also try to design both algorithms simultaneously as opposed to designing them separately, which would allow me to make both programs run more similarly, the only difference being the algorithm, rather than being pretty different with the exception of the final calculations.

Nearest Neighbor Algorithm

This algorithm uses a starting point and finds the nearest neighbor of the unvisited location to try and find the shortest path.

Pseudocode:

Public static void main

Numberformat currencyformat

Declare final int MPH equals 60, MPG equals 7, DRIVER\_SALARY equals 1200, HELPER\_SALARY equals 900, HOTEL equals 100, FOOD equals 68, NUM\_CITIES equals 7

Declare final double PRICE\_PER\_GALLON equals 3.32, DRIVER\_PER\_MILE equals 0.56, HELPER\_PER\_MILE equals 0.42, WEAR equals 0.88

Declare int distance equals 0, visits equals 0, e equals 0, hours equals 0

Declare String array cityNames equals to Rockville, Silver Spring, Philadelphia, Pittsburgh, Baltimore, Cleveland, New York City Declare two dimensional int array d = {{0, 13, 142, 225, 40, 352, 227}, {13, 0, 136, 237, 34, 363, 222}, {141, 135, 0, 305, 101, 432, 97}, {226, 237, 304, 0, 248, 133, 371}, {40, 34, 106, 248, 0, 374, 192 }, {352, 364, 431, 133, 375, 0, 462}, {228, 222, 97, 370, 188, 462, 0}}

Declare ArrayList Integer citiesLeft

Declare ArrayList String route

For( i equals 1, I less than or equal to 6, I increment)

Add i to citiesList

Add Rockville to citiesList

driverSalary equals DRIVER\_SALARY

helperSalary equals HELPER\_SALARY

int it equals 0

for(visits equals 1; visits less than 7; visits increment)

int minDist = 99999

for(i equals 0; i is less than citiesLeft.size; i increment)

if (d[e][citiesLeft.get(i)] is less than minDist)

minDist equals d[e][citiesLeft.get(i)]

it equals i

e equals citiesLeft.get(it)

remove e from citiesLeft

add cityNames[e] to route

distance increment equals minDist

add Rockville to route

distance increment equals d[5][0]

totalFuel equals PRICE\_PER\_GALLON times (distance divided by MPG)

driverSalary equals distance times DRIVER\_PER\_MILE

helperSalary equals distance times HELPER\_PER\_MILE

totalTime equals distance divided by MPG

maintenance equals distance times WEAR

double hotelCost equals HOTEL times NUM\_CITIES times 2

double foodCost equals FOOD times NUM\_CITIES times 2

total equals totalFuel plus driverSalary plus helperSalary plus maintenance plus hotelCost plus foodCost

for( i equals 0; i<route.size; i++)

print route.get(i)

print distance

print driverSalary

print helperSalary

print maintenance

print totalTime

print totalFuel

print hotelCost

print foodCost

print total

Test Case:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | Expected Output | Actual Output | Total Distance Traveled | Passed Test |
| MPH = 60, MPG = 7, DRIVER\_SALARY = 1200, HELPER\_SALARY = 900, HOTEL = 100, FOOD = 68, NUM\_CITIES = 7, PRICE\_PER\_GALLON = 3.32, DRIVER\_PER\_MILE = 0.56, HELPER\_PER\_MILE = 0.42, WEAR = 0.88  City Names: Rockville, Silver Spring, Philadelphia, Pittsburgh, Baltimore, Cleveland, New York City  Distance between cities:  {{0, 13, 142, 225, 40, 352, 227}, {13, 0, 136, 237, 34, 363, 222}, {141, 135, 0, 305, 101, 432, 97}, {226, 237, 304, 0, 248, 133, 371}, {40, 34, 106, 248, 0, 374, 192 }, {352, 364, 431, 133, 375, 0, 462}, {228, 222, 97, 370, 188, 462, 0}}; | Order of cities: Rockville, Silver Spring, Baltimore, Philadelphia, New York City, Pittsburgh, Cleveland, Rockville  Distance: 1105  Driver: 1818.80  Helper: 1364.10  Maintenance: 972.40  Driving Time: 18.416666  Fuel Cost: 521.24  Hotel Cost: 1400.00  Food Cost: 952.00  Total Cost: 7028.54 | Rockville  Silver Spring  Baltimore  Philadelphia  New York City  Pittsburgh  Cleveland  Rockville  Total Distance: 1105  Driver Total: $1,818.80  Helper Total: $1,364.10  Maintenence Cost: $972.40  Total Driving Time: 18. 416666666666668  Total Fuel Cost: $521.24  Hotel Cost: $1,400.00  Food Cost: $952.00  Total Cost: $7,028.54 | 1105 | Yes |

Method of Exhaustion Algorithm

This algorithm finds every possible scenario and sorts through them in order to find the scenario with the smallest possible distance.

Pseudocode:

Declare static two dimensional array allRoutes with dimensions 720 and 8

Declare static int fCounter equals 0

Public static void main(String[] args)

Declare final int MPH equals 60, MPG equals 7, DRIVER\_SALARY equals 1200, HELPER\_SALARY equals 900, HOTEL equals 100, FOOD equals 68,

NUM\_CITIES equals 7

Declare final double PRICE\_PER\_GALLON equals 3.32, DRIVER\_PER\_MILE equals 0.56, HELPER\_PER\_MILE equals 0.42, WEAR equals 0.88

Declare int minDistance equals 99999

Declare two dimensional int array d = {{0, 13, 142, 225, 40, 352, 227}, {13, 0, 136, 237, 34, 363, 222}, {141, 135, 0, 305, 101, 432, 97}, {226, 237, 304, 0, 248, 133, 371}, {40, 34, 106, 248, 0, 374, 192 }, {352, 364, 431, 133, 375, 0, 462}, {228, 222, 97, 370, 188, 462, 0}}

Declare double driverSalary equals 0, helperSalary equals 0, totalFuel equals 0, maintenance equals 0, totalTime equals 0

Declare String array cityNames equals to Rockville, Silver Spring, Philadelphia, Pittsburgh, Baltimore, Cleveland, New York City

Declare String cities equals 123456

Declare int array finalRoute with size 8

Call exhaustionMethod with parameters : cities.toCharArray(), 0

For (int r equals 0, r is less than 720, r increment)

Distance equals 0

For (int t equals 0; t less than 7; t increment)

Distance addition assingment d[allRoutes[r][t]]allRoutes[r][t plus 1]]

If distance is less than minDistance

minDistance equals distance

for( I equals 0; I is less than 8; I increment)

finalRoute[i] equals allRoutes[r][i]

for(I equals 0; I is less than 7; I increment)

print d[finalRoute[i]][finalRoute[i+1]]

print "The cities are visited in the following order: "

for(I equals 0; I is less than 8 I increment)

print cityNames[finalRoute[i]]

totalFuel equals PRICE\_PER\_GALLON times (minDistance divided by MPG)

driverSalary equals minDistance times DRIVER\_PER\_MILE

helperSalary equals minDistance times HELPER\_PER\_MILE

totalTime equals minDistance divided by MPG

maintenance equals minDistance times WEAR

double hotelCost equals HOTEL times NUM\_CITIES times 2

double foodCost equals FOOD times NUM\_CITIES times 2

total equals totalFuel plus driverSalary plus helperSalary plus maintenance plus hotelCost plus foodCost

print distance

print driverSalary

print helperSalary

print maintenance

print totalTime

print totalFuel

print hotelCost

print foodCost

print total

public static void swap(char array place, int i, int j)

char temp equals place[i]

place[i] equals place[j]

place[j] equals temp

public static void exhaustionMethod(char array place, int currentIndex)

if currenrIndex equals to place.length minus 1

Declare ArrayList of type Integer route

for( c equals 0; c less than place.length; c increment)

route.add(Character.getNumericValue(place[c]))

route.add(0,0)

route.add(0)

for (u equals 0; u less than 8; u increment)

allRoutes[fCounter][u] equals route.get(u)

fCounter increment

for( i equals currentIndex; i less than place.length; i increment)

call swap method with parameters: place, currentIndex, i

call exhaustionMethod method with parameters: place, currentIndex + 1

call swap method with parameters: place, currentIndex, i

Test Case:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | Expected Output | Actual Output | Total Distance Traveled | Passed Test |
| MPH = 60, MPG = 7, DRIVER\_SALARY = 1200, HELPER\_SALARY = 900, HOTEL = 100, FOOD = 68, NUM\_CITIES = 7, PRICE\_PER\_GALLON = 3.32, DRIVER\_PER\_MILE = 0.56, HELPER\_PER\_MILE = 0.42, WEAR = 0.88  City Names: Rockville, Silver Spring, Philadelphia, Pittsburgh, Baltimore, Cleveland, New York City  Distance between cities:  {{0, 13, 142, 225, 40, 352, 227}, {13, 0, 136, 237, 34, 363, 222}, {141, 135, 0, 305, 101, 432, 97}, {226, 237, 304, 0, 248, 133, 371}, {40, 34, 106, 248, 0, 374, 192 }, {352, 364, 431, 133, 375, 0, 462}, {228, 222, 97, 370, 188, 462, 0}}; | Order of cities: Rockville, Pittsburgh, Cleveland, New York City, Philadelphia, Baltimore, Silver Spring, Rockville  Distance: 1065  Driver: 1796.40  Helper: 1347.30  Maintenance: 937.20  Driving Time: 17.75  Fuel Cost: 504.64  Hotel Cost: 1400.00  Food Cost: 952.00  Total Cost: 6937.54 | 225  133  462  97  101  34  13  The cities are visited in the following order:  Rockville  Pittsburgh  Cleveland  New York City  Philadelphia  Baltimore  Silver Spring  Rockville  Total Distance: 1065  Driver Total: $1,796.40  Helper Total: $1,347.30  Maintenance Cost: $937.20  Total Driving Time: 17.75  Total Fuel Cost: $504.64  Hotel Cost: $1,400.00  Food Cost: $952.00  Total Cost: $6,937.54 | 1065 | Yes |

Conclusion:

The Method of Exhaustion Algorithm was able to save 1 hour of driving time, 40 miles and a total of $91 in total costs compared to the most optimal result found from the Nearest Neighbor Algorithm. The Nearest Neighbor algorithm only considers the distance to the next city when calculating the minimum distance, rather than the implications of that decision. The Method of Exhaustion Algorithm was able to consider all possibilities and thus found the most optimal route and in my opinion was the more precise algorithm. However, in larger scenarios, the nearest neighbor algorithm would probably be more efficient. The nearest neighbor algorithm only stores one scenario, while the Method of Exhaustion Algorithm stores every single possible scenario and may require more resources to run.