Tushara Vepakomma

Student ID: 801044941

Date: 04/17/2018

Report for Algorithms and Data Structures Project 2

# Problem Statement in brief:

# Finding a shortest path tree from Charlotte to the five major cities in Florida: Jacksonville, Tallahassee, Orlando, Tampa, & Miami and finding shortest distance trip from charlotte to visit all above 5 cities in Florida and then back to charlotte.

# Algorithms used for implementation:

1. Dijkstra’s algorithm and Brute Force algorithm

**Dijkstra’s algorithm**

Dijkstra's algorithm is used for finding the shortest paths between nodes in a graph. In this algorithm one single node is considered as the source node and finds the shortest paths from the source node to all other nodes in the graph. Thus, it produces a shortest path tree. To explain this algorithm in brief, the source node is considered as the initial node. We will mark all other nodes as unvisited. We assign every node some distance. For initial node we set the distance to zero and we set infinity to all other nodes initially. Now for the current node, we consider all its unvisited neighbours and calculate the distances from the current node. We compare the newly calculated distance with the already present value and assign the smaller value. Once all the neighbours of the current node is done, we mark this node as visited. Now we move to the next unvisited node with smallest distance and repeat the above step i.e.; checking the neighbours and mark it as visited. Once the destination node is marked as visited, the algorithm can be stopped. Otherwise the above steps have to be repeated.

**Brute Force Algorithm:**

Brute Force algorithm will examine all possible ways to solve a problem and will pick the best solution among those ways. So, brute force will cover all possible solutions and select the best one. If we need to go from node A to node C in a shortest distance, the brute force algorithm will look into all the combination of roads we can take and will pick the best possible route to travel from A to C. It will iterate through all possible solutions until a valid solution is found.

Approach:

# 1.The shortest path from charlotte to each of the 5 cities is calculated using Dijkstra’s algorithm. The minimum heap is created of size equal to number of vertices in the graph. Every node in the heap contains vertex number and distance value of the vertex. Then we insert the source node into the heap and make its distance as 0. The distance assigned to all other vertices is infinite. While the heap doesn’t become empty we follow the below steps:

1. We will fetch the minimum distance vertex from heap. Then we loop through all the adjacent vertices of this extracted vertex and do the following for every vertex.
2. If there is a shorter path from extracted vertex(say u) to the adjacent vertex(v), and the adjacent vertex is in min heap, then update the distance of adjacent vertex if its distance value is more than u-v and distance value of u.

Finally, we print the array distance to print all the shortest paths.

2.For shortest distance trip, our goal is to find the shortest distance trip starting from Charlotte and then visit 5 cities and back to Charlotte. As per the code:

**generateAllPossibleRoutes** method generates all the possible combinations of 5 cities - [Jacksonville, Orlando, Tampa, Miami, Tallahassee] and returns an array list of array list of cities, each in different order. We recursively call the same method and perform swap for each recursion.  I used Brute force algorithm i.e. 5 factorial so 120 array lists will be generated as an output of this method. Please note this is an array list of array list of type string.

**CalculateSDTrip method** takes the list of all possible routes from generateAllPossibleRoutes method and loops through each possible route and finds the shortest distance for each route. For example, for each possible Route until possibleRoutes.size()

1. Get the current item in array list in possibleRoutes array list. i.e. [Jacksonville, Tallahassee, Orlando, Tampa, Miami].
2. Add the destination to this array list i.e. Charlotte. Now the array list looks like [Jacksonville, Tallahassee, Orlando, Tampa, Miami, Charlotte].
3. Set the source vertex as Charlotte.
4. Loop thru this array list and do the following for each city in the array list.
   1. Call shortestPath method to calculate the shortest path from source vertex to all the cities in the map. Output contain distances for all the cities from source vertex.
   2. Get the weight only for the current city i.e index of the current iteration.
   3. Add this weight to the previous weight for each iteration. This is to get the total weight for all cities.
   4. Call the getRoute method to store the route followed to reach this destination. i.e. in pathUsed
   5. If the current city is Charlotte, then this is the return journey so store the route in different variable i.e. in returnPath
   6. Change the source vertex to the current city for the next iteration.
5. Now we have the total distances/weight for one route, example: [Jacksonville, Tallahassee, Orlando, Tampa, Miami, Charlotte]
6. If total distance/weight of the previous route is greater than the new total distance/weight, then store the new total distance/weight into finalWeight variable.
7. Removing the destination(charlotte) that I have added in the beginning for this route.
8. Repeat this process for each possible route. i.e.120 iterations.

# Findings:

|  |  |  |  |
| --- | --- | --- | --- |
| Finding | Source node | Destination node | Shortest distance |
| Shortest distance from charlotte to each of the 5 cities | Charlotte | Jacksonville | 388 |
|  | Charlotte | Tallahassee | 474 |
|  | Charlotte | Orlando | 533 |
|  | Charlotte | Tampa | 587 |
|  |  | Miami |  |
|  | Charlotte |  | 740 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Shortest Distance Trip:**

**Total Distance(Shortest): 1758**

Source Path(Reverse-Order):

Miami,VeroBeach,Cocoa,DaytonaBeach,Jacksonville,Savannah,Columbia,Charlotte

Jacksonville,DaytonaBeach,Cocoa,VeroBeach,Miami

Tallahassee,Jacksonville

Orlando,Jct.Int.75,Tallahassee

Tampa,Orlando

Return Path(Reverse-Order): Charlotte,Columbia,Savannah,Jacksonville,Jct.Int.75,Tampa

**Interpretation of the above output:**

Total Distance(Shortest): 1758 - This is the total distance starting from charlotte and travel to 5 major cities and then return to the charlotte.

 Source Path(Reverse-Order):

There are 5 cities to travel. so we can see 5 lines.

When I say reverse-order, this is the format of each line below: [First Destination out of 5 cities], [ Path taken to reach First Destination], [Starting City]

so, in the first line below,

Miami is the first destination out of 5 cities

Path taken from Charlotte to Miami is Columbia, Savannah, Jacksonville,DaytonaBeach, VeroBeach, Miami

Starting City is Charlotte for the first line....

Similar interpretation for rest of the lines.....

Miami, VeroBeach, Cocoa,DaytonaBeach,Jacksonville,Savannah,Columbia,Charlotte   -

Jacksonville, DaytonaBeach,Cocoa,VeroBeach,Miami

Tallahassee,Jacksonville

Orlando,Jct.Int.75,Tallahassee

Tampa,Orlando

Return Path(Reverse-Order): Charlotte,Columbia,Savannah,Jacksonville,Jct.Int.75,Tampa  - Last destination out of 5 cities was Tampa as you can see the 5th line. So this the return path from Tampa to Charlotte.....when we read it in reverse order.....

https://ssl.gstatic.com/ui/v1/icons/mail/images/cleardot.gif

# Conclusion:

So for calculating the shortest path from a source vertex to remaining vertices it is good to use Dijkstra’s algorithm. Dijkstra’s algorithm is used in finding shortest paths. It is widely used in google maps, geographical maps. The running time of Dijkstra’s algorithm is O(E log V) where V is no. of vertices and E is total number of edges. The disadvantage of Dijkstra’s algorithm is it cannot handle negative edges. For calculating shortest distance trip we can use Brute Force algorithm, it will examine all possible ways to solve a problem and will pick the best solution among those ways. The disadvantage of Brute Force approach is it is not suitable for solving problems that have hierarchical structure.