CSE2046 – Analysis of Algorithms
Efficient Way of Wireless Transmitter Problem Solving

Supervisor:
Assistant Prog. Ömer KORÇAK
Senior Lecturer
Computer Science Engineering Department, Marmara University
e-mail: omer.korcak@marmara.edu.tr

Supervisees:
Bilgehan NAL
Student ID: 150114038
Computer Science Engineering Department, Bachelor's Degree
e-mail: bilgehan.nal@marun.edu.tr

Oğuzhan BÖLÜKBAŞ
Student ID: 150114022
Computer Science Engineering Department, Bachelor's Degree
e-mail: oguzhanbolukbas@marun.edu.tr

Table of Contents

1. Abstract

This is the example of solving and simulating the wifi connection problem. We assume a system laptops can communicate between each other with their wifis and we tried to find out the shortest path of first laptop and other laptops.
This work is made for CSE 246 Lecture Marmara University.
Assignment page: http://mimoza.marmara.edu.tr/~omer.korcak/courses/CSE246/HW3_Spring2017.pdf

2. Introdunction

In this problem we have n laptops and these laptops can send a message if it connects to other laptop. Firstly we tried to show connectivity map of laptops on a graph. And we tried to solve our problem with a shortest path algorithm. We use Dijkstra's algorithm for this example and we tested this algorithm for different inputs.

3. Theory

3.1. Dijkstra's algorithm

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.[1][2]

The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes,[2] but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

For a given source node in the graph, the algorithm finds the shortest path between that node and every other.[3]:196–206 It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path algorithm is widely used in network routing protocols, most notably IS-IS and Open Shortest Path First (OSPF). It is also employed as a subroutine in other algorithms such as Johnson's.

Dijkstra's original algorithm does not use a min-priority queue and runs in time $O(|V^2|)$ (where $|V|$ is the number of nodes). The idea of this algorithm is also given in Leyzorek et al. 1957. The implementation based on a min-priority queue implemented by a Fibonacci heap and running in $O(E + |V| \log |V|)$ (where $|E|$ is the number of edges) is due to Fredman & Tarjan 1984. This is asymptotically the fastest known single-source shortest-path algorithm for arbitrary directed graphs with unbounded non-negative weights. However, specialized cases (such as bounded/integer weights, directed acyclic graphs etc.) can indeed be improved further as detailed in § Specialized variants.

In some fields, artificial intelligence in particular, Dijkstra's algorithm or a variant of it is known as uniform-cost search and formulated as an instance of the more general idea of best-first search.[4]

Algorithm

Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.
3. For the current node, consider all of its neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B (through A) will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.
4. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

Description

Suppose you would like to find the shortest path between two intersections on a city map: a starting point and a destination. Dijkstra's algorithm initially marks the distance (from the starting point) to every other intersection on the map with infinity. This is done not to imply there is an infinite distance, but to note that those intersections have not yet been visited; some variants of this method simply leave the intersections' distances unlabeled. Now, at each iteration, select the current intersection. For the first iteration, the current intersection will be the starting point, and the distance to it (the intersection's label) will be zero. For subsequent iterations (after the first), the current intersection will be the closest unvisited intersection to the starting point (this will be easy to find).

From the current intersection, update the distance to every unvisited intersection that is directly connected to it. This is done by determining the sum of the distance between an unvisited intersection and the value of the current intersection, and relabeling the unvisited intersection with this value (the sum), if it is less than its current value. In effect, the intersection is relabeled if the path to it through the current intersection is shorter than the previously known paths. To facilitate shortest path identification, in pencil, mark the road with an arrow pointing to the relabeled intersection if you label/relabel it, and erase all others pointing to it. After you have updated the distances to each neighboring intersection, mark the current intersection as visited, and select the unvisited intersection with lowest distance (from the starting point) – or the lowest label—as the current intersection. Nodes marked as visited are labeled with the shortest path from the starting point to it and will not be revisited or returned to.

Continue this process of updating the neighboring intersections with the shortest distances, then marking the current intersection as visited and moving onto the closest unvisited intersection until you have marked the destination as visited. Once you have marked the destination as visited (as is the case with any visited intersection) you have determined the shortest path to it, from the starting point, and can trace your way back, following the arrows in reverse; in the algorithm's implementations, this is usually done (after the algorithm has reached the destination node) by following the nodes' parents from the destination node up to the starting node; that's why we also keep track of each node's parent.

This algorithm makes no attempt to direct "exploration" towards the destination as one might expect. Rather, the sole consideration in determining the next "current" intersection is its distance from the starting point. This algorithm therefore expands outward from the starting point, interactively considering every node that is closer in terms of shortest path distance until it reaches the destination. When understood in this way, it is clear how the algorithm necessarily finds the shortest path. However, it may also reveal one of the algorithm's weaknesses: its relative slowness in some topologies.

Problem

Supposing there are n laptops each containing a wireless transmitter. For each laptop i, following information are known:

1. Position, (x_i, y_i),
2. Wireless transmission range, r_i

That is, we can imagine that the wireless range of laptop i is a circle centered at (x_i, y_i) with radius r_i. We say that the laptop i can communicate with laptop j if laptop j is in the wireless range of laptop i. (If wireless range of laptop j is less than wireless range of laptop i, it is possible that laptop i can send message to laptop j, but laptop j may not be able to send message to laptop i.) Of course, not every laptop can communicate with every other laptop, but laptops can send messages by using intermediate laptops as routers. Hop distance h(i, j) is defined as the minimum number of intermediate laptops used to send a message from laptop i to laptop j. For example, if two laptops can communicate directly, the hop distance between them is

a) Given a set of n agents with their positions and wireless ranges, design an efficient algorithm to compute the hop-distance from the first laptop to every other reachable laptop. If agent i is not reachable from agent j then the hop distance h(i, j) will be set to 0. You need to have a pdf document file, report2.pdf. In this document, give a step by step verbal description of your algorithm in detail.

4.Solving The Problem

Shortest Path Algorithm

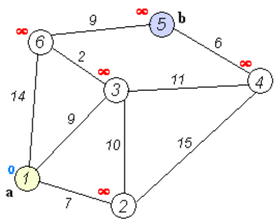
We decide that this problem can be calculated with a shortest path algorithm. If we look at the root of the problem. This problem is a finding a shortest path problem. Therefore we decided to applicate one of the best shortest path algorithm named “Dijkstra’s algorithm”.

Basically, we look at the connectivity of the laptops betwen each other and build a graph from these informations then, we determine the shortest distance of first element to every single elements.

We have a clas named Laptop and each Laptop object keeps the coordinate information and radius information.

We keep the connectivity of the laptops in a graph. We built our graph as an adjacency list. We have an array, size’s of the array is number of laptop and each element of the array keeps a linked list. This graph keeps integer data.(This integer data is the distance of two laptops. Of course these two laptops can communicate between each other.)

We have a method named graphMaker, we look at the communication of the all laptops with other laptops. If there is a connectivity. We update our graph system fort this connectivity.



Lastly, we have a function named “findDistance()”. As we could understand from the name, this method calculates all shortest distances from first laptop to other laptops with dijkstra’s algorithm.

After we applicate “Dijkstra’s algorithm” all laptops’ distances to first laptop are determined. We save this data to an integer array and we write the output(distance array) to an output file.

Steps of the Algorithm:

1- Read the data from the input file
2- We are creating laptop objects with its double values.
3- A graph creation
4- Dijkstra's algorithm applicating.
5- Show the result and write to an output file.

Time Complexity

Conrtucting a graph takes $\Omega(n^2)$ tim, because every laptops are compered with other laptops.

Verticies of the graph: n
Edges of the graph: E

Dijkstra's algorithm time complexity: O(n + E) times.

Conclusion

If we look at the problem detaily. We can figure out that this problem is classical shortest path problem. We could solve this problem with a shortest path algorithm (Dijkstra's Algorithm, Bellman Ford Algorithm etc.) We decided to use Dijkstra's algorithm, but first of all we simulated the problem on a graph using some geometric formulas. After this proccess everything looks available to solve this problem with Dijkstra's algorithm. Therefore after we solve this problem we tested it and our algorithm runs with the dijkstra's algorithm's efficiency.

5.Conclusion

If we look at the problem detaily. We can figure out that this problem is classical shortest path problem. We could solve this problem with a shortest path algorithm (Dijkstra's Algorithm, Bellman Ford Algorithm etc.) We decided to use Dijkstra's algorithm, but first of all we simulated the problem on a graph using some geometric formulas. After this proccess everything looks available to solve this problem with Dijkstra's algorithm. Therefore after we solve this problem we tested it and our algorithm runs with the dijkstra's algorithm's efficiency.

6.References:

[1]: *Frana, Phil (August 2010). "An Interview with Edsger W. Dijkstra". Communications of the ACM.* ***53*** *(8): 41–47. doi:10.1145/1787234.1787249.* What is the shortest way to travel from Rotterdam to Groningen? It is the algorithm for the shortest path which I designed in about 20 minutes. One morning I was shopping with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path.

[2]: *Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs" (PDF). Numerische Mathematik.* ***1****: 269–271. doi:10.1007/BF01386390.*

[3]: *Mehlhorn, Kurt; Sanders, Peter (2008). Algorithms and Data Structures: The Basic Toolbox (PDF). Springer.*

[4]: *Felner, Ariel (2011). Position Paper: Dijkstra's Algorithm versus Uniform Cost Search or a Case Against Dijkstra's Algorithm. Proc. 4th Int'l Symp. on Combinatorial Search.* In a route-finding problem, Felner finds that the queue can be a factor 500–600 smaller, taking some 40% of the running time.