Maze Simulation

Prateek Chaudhury 2019CS10384
Gunjan Kumar 2019CS10353

July 2021

1 Problem Description

There is a policeman who has to patrol the area after every hour in the night. Every time, the policeman starts patrolling from the police station. He visits all the localities and then after visiting all places, he returns back. All the places are connected by one way streets with each other. There are many possible such paths for policeman to visit all the localities. Out of all possible paths, he wants to chose the path which requires minimum distance to be covered. As the policeman cannot figure this out by himself, we have to help him find out the required path.

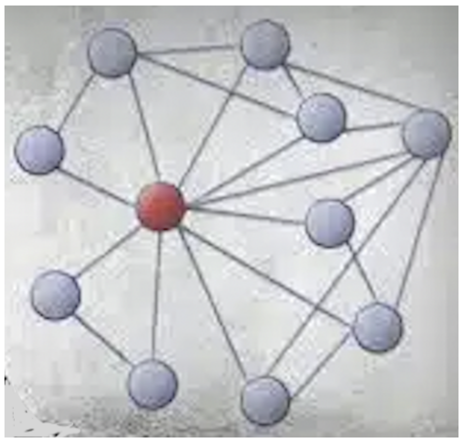

Figure 1: Graph representation

2 Simulation Using Maze

We can change the above problem to a graph problem which can further be mapped to travelling in a maze. Police Station and all the localities are vertices of graph where the edges between them represent the length of one-way routes between the localities. There can be many walks in the graph which starts from Police Station, then visits all other nodes and ends at the Police Station. Out of all such walks, we have to find the walk which is of the minimum length.

Further, the problem can be simulated using a maze. Each cell of maze which is not block represents the non-residential areas where police man need not go. Every other cell represents the nodes which has to be visited. Let's place the police man at the top left corner. Each cell will contain a certain number which represents the distance of the cell to the neighbouring cell using the one-way streets. We have to find a walk that starts from the top left corner, then visit all the cells of the maze and them, return back to the top-right corner by covering the minimum possible distance.

3 Approach 1 : Brute Force

Step 1 : In this approach, we will find all the possible permutations of nodes. Now, each permutation will represent one way of visiting all the nodes of the graph. Cost for each permutation will be equal to the summation of shortest paths between two consecutive vertices in the permutation.

Step 2 : We will use Floyd - Warshall algorithm to pre-process the graph and store the shortest distance between any two pair of nodes.

Step 3 : The shortest path visiting all the vertices of the graph will be minimum of all the possible permutations.

Feasibility :

The time complexity of the Floyd-Warshall algorithm = $O(N^3)$
The number of all different permutations of N nodes = $O(N!)$
Therefore, complexity of brute force is,

$$Complexity = O(V^3 + V \times V!) \tag{1}$$

Hence, total complexity will be $O(V^3 + V \times V!)$ which is not feasible as the factorial grows very fast.

4 Optimised Approach : Modified Dijkstra Algorithm

Step 1 : In this approach, we first declare an 2-D array which stores cost. The cost present at cost[X][Y] represents the shortest path to node X visiting all the nodes represented by the bit-mask Y. Next, a priority queue is declared which stores pairs where first element of pair represents the node X and second element is an bit-mask representing all the node visited during this reaching node X.

Step 2 : Next, we try to start the shortest path from each node by adding each node to the priority queue and turning their only their particular bit on. We also initialise the cost of visiting node X such that the bit-mask has only their particular bit turned on as 0 in the cost array.

Step 3 : Now we use the Dijkstra's algorithm. We iterate over the children nodes of the current node. For each child, we check if the cost value of it visiting a specific subset of nodes(represented by the bits turned on in the bit-mask) greater than the current cost plus the weight of the edge that connects the current node with the child. In this case, we need to update the cost value of the child node. Also, we add the child and the current bit-mask to the priority queue in which we turn the bit of the child node on.

Step 4 : Now we iterate over all the nodes in the maze and check for the cost present at the index with bit-mask having all the bits turned on. The minimum of these values will be our desired result.
Feasibility :

The time complexity of the Dijkstra algorithm = $O(VlogV)$, where V is the vertex
Number of states per node = 2^N, as there are 2^N states of bitmasks
Total number of Nodes = $N \times 2^N$
Therefore, complexity of modified Dijkstra is,

$$Complexity = O(N \times 2^N log(N \times 2^N)) \tag{2}$$

Hence, total complexity is exponential which is not feasible as it grows very fast.

5 Polynomial Time Complexity Not Achievable

As we have shown above the factorial complexity in the brute force approach has been improves to exponential complexity. But still exponential complexity is not feasible for number of nodes being greater than 30-40.

Actually this is a NP hard problem and thus it is not possible to have a polynomial time complexity. Now let's prove that our problem is a NP-hard problem.

Claim : Our problem can be reduced to the Travelling Salesman Problem.

Proof : Calculate the minimum distance between each pair of nodes first. For this, we can use the Floyd-Warshall algorithm. Simply construct the whole graph where the edge between nodes u and v represents the least cost from u to v.

Since, now the graph becomes complete, so our problem reduces to Travelling Salesman Problem. The conversion occurs by using only polynomial time complexity. So, our problem is as hard as the Travelling Salesman Problem.

As we already know that Travelling Salesman Problem is NP-hard, so our problem is also NP-hard.

6 Approximation Algorithm

Since, polynomial time complexity of the problem has not been discovered and most likely don't exist. We will try to find a path which has sum near to the minimum path possible and it is feasible even for higher number of nodes.

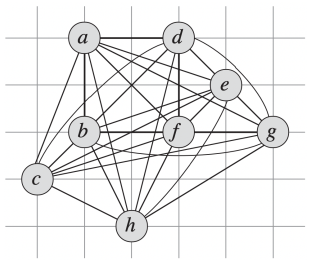

Figure 2: Graph

6.1 Algorithm

Step 1 : Starting from the Police Station as the root, find the Minimum Spanning Tree of the Graph using Prim's algorithm.

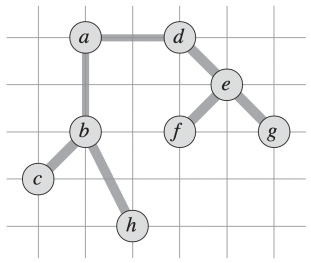

Figure 3: Minimum Spanning Tree, T

Step 2 : The approximated path is obtained by traversing the tree in the following order. $Node \rightarrow LeftSubtree \rightarrow Node \rightarrow RightSubtree \rightarrow Node$

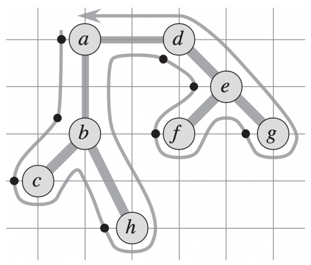

Figure 4: Final path, H

6.2 Correctness of Algorithm

The path returned by the above algorithm clearly visits all the vertices of the graph. Now, we need to show that the length of the path return by the above algorithm is close to the length of the actual such path.

Claim : The length of the path produced by the above algorithm is never more than twice the length of the path of best possible output.

Proof : Let the length of any path (P) is L(P). Let's name the minimum spanning tree as T and The path returned by above algorithm as H. Assume that the best possible path is B.

Now, the length of the path B is never less than the length of the MST, T. This is because MST is the minimum cost tree that connects all vertices. The path B visits all nodes so, its length has to be greater than the length of B.

$$L(B) >= L(T) \tag{3}$$

Now, the length of the path H is twice the length of the MST, T.

$$L(H) = 2 \times L(T) \tag{4}$$

Using above two equations, we get

$$L(H) <= 2 \times L(B) \tag{5}$$

Hence, length of the path H is less than twice the length of the path B.

6.3 Run Time Analysis

Complexity of Algorithm = **Prim's Algorithm** Complexity + **Tree Traversal**

- Simple Implementation

$$Complexity = O(N^2) + O(N) = O(N^2) \tag{6}$$

Even with the implementation of MST-PRIM without using priority queue, the running time complexity of finding the optimal path has reduced to a polynomial complexity from an exponential complexity.

- Implementation using Priority Queue

$$Complexity = O(NlogN) + O(N) = O(NlogN) \tag{7}$$

If we implement the Prim's algorithm using binary min heap **priority queue**, then the time complexity of the algorithm will further improve.

Feasibility : This approach is feasible and will run for even very large number of nodes upto millions. The $O(N^2)$ implementation of the approach can run within seconds for 10^4 nodes while the $O(NlogN)$ implementation of the approach can run within seconds for 10^7 nodes.

6.4 Implementation Plan

First we find the Minimum Spanning Tree of the graph by using Prim's algorithm. We have the list of vertex nodes as V and the adjacency matrix as M. The root node is named 'a'. Then we use a min-priority queue, which is a binary min-heap, which stores all the nodes not yet visited during the execution of the algorithm. Each vertex node has 2 attributes, key which represents minimum weight of any edge connecting to another vertex in the tree, and other attribute being the parent vertex node.

Now, after obtaining the MST, we traverse the MST starting from the root. Traversal at any node is done as, current node, then left sub-tree, then current node, then right sub-tree and finally again through the current node. The nodes are finally stored in the required order in a vector.

PRIM'S MINIMUM SPANNING TREE

```
Q ← min priority queue
M ← adjacency list

for Node in V
    $Node.key \leftarrow \infty$
    $Node.parent \leftarrow null$
    $a.key \leftarrow 0$
    $Q \leftarrow V$

while Q $!= \phi$
    $Node \leftarrow ExtractMin(Q)$
    for each node in M[Node]
        if node in Q and weight(Node, node) < node.key
            $node.key \leftarrow weight(Node, node)$
            $node.parent \leftarrow Node$
```

PATH GENERATION FROM PRIM'S TREE

```
path ← vector of nodes
generatePath(root) :
    if (root == null ) do nothing
    path.add(node)
    generatePath(node.left)
    path.add(node)
    generatePath(node.right)
    path.add(node)
```

DATA STRUCTURES USED

- **1.Binary Min Heap :** Used for finding the implementation of the Min priority queue which has been used in prim's algorithm for finding the minimum spanning tree.
- **2.Vectors :** It has been used for storing the graph in the form of adjacency list. Finally, it has also been used for storing the final path in generatePath function.