1 Main Idea

In this problem, we are given an undirected graph $G = (V, E)$ where all edges have the same length of 1. Given two nodes $s, t \in V$, we want to find the number of *distinct* shortest paths from s to t in linear time. The main idea of the algorithm that solves this problem is to use BFS to find the distance from the source node to the other nodes in the graph. Keeping track of the number of paths from the source node to each node, we count the number of shortest paths from the source node to the target node once we reach the target node.

2 Pseudocode

Algorithm 1: Count Distinct Shortest Paths

Input: An undirected graph $G = (V, E)$, source node s, target node t
Output: The number of distinct shortest paths from s to t

```
create a queue Q;
create a hash map distance;
create a hash map numPaths;
for v ∈ V do
    distance[v] = ∞;
    numPaths[v] = 0;
end
distance[s] = 0;
numPaths[s] = 1;
Q.enqueue(s);
while Q is not empty do
    u = Q.dequeue();
    if u = t then
        break;
    end
    for v ∈ u.neighbors do
        if distance[v] = ∞ then
            distance[v] = distance[u] + 1;
            numPaths[v] = numPaths[u];
            Q.enqueue(v);
        end
        else if distance[v] = distance[u] + 1 then
            numPaths[v] += numPaths[u];
        end
    end
end
return numPaths[t];
```

3 Proof of Correctness

To show that this algorithm is correct, we need to show that it will always return the number of distinct shortest paths from the source node to the target node. The algorithm starts by doing BFS from the source node s. Thus, the algorithm will always process nodes in increasing order of distance from the source node. This guarantees that when we reach a node, we have already processed the shortest paths to it. We find the shortest between each node and the source since BFS will always find the shortest path. The *pathCount* hash map correctly keeps track of the shortest paths since it uses the shortest path to the current node to calculate the shortest path to the next node. Thus, when the algorithm finally reaches the target node, it will have the correct number of shortest paths from the source node to the target node.

4 Runtime Analysis

The time complexity of this algorithm is $O(V + E)$. This is because we are doing a BFS traversal of the graph, and each edge is processed at most twice.