Title

Reasoning

To solve this problem we can use the bfs algorithm to find the shortest path, then if we use bfs we can keep tracking how many possible paths there are that reach the output in the same ammount of time. We can do this by slightly modifying the algorithm to have a *count* array that updates similarly to the distance array. Whenever we connect a new node (that is not in *seen*) then we can give it the *count* that the current node has. If it is a node that is already connected, then we can update count. If we have the same shortest distance to this node, then we add the node we are connecting from's count to its count. At the end, the *count* of the target node should be the highest possible count of shortest paths to the target.

Psudocode

```
find_max_shortest_paths(G=(V,E), s, t):
    for every vertex v:#O(n)
        seen[v] = false
        dist[v] = inf
        count[v] = 0

    count[s] = 1
    dist[s] = 0
    beg = 1
    end = 2
    Q[1] = s
    seen[s] = true
    while beg<end:#O(m)
        head=Q[beg]
        for every neighbor u of head:#O(deg(u))
            if not seen[u]:
                Q[end++] = u
                dist[u] = dist[head]+1
                seen[u] = true
                count[u] = count[v]
            else if (dist[u] == dist[v] + 1):
                count[u] += count[v]
        beg ++
```

Proof

Proof of the algorithm here.

Running Time

Estimate

The running time is $\Theta(?)$ or $O(?)$

Running Time Reasoning

Reasoning Here