

[Register for SDE Sheet Challenge](#)**takeUforward**[Signin/Signup](#)[Striver's SDE Sheet](#)[Striver's A2Z DSA Course/Sheet](#)[Striver's DSA Playlists](#)[CS](#)[Interview Prep Subjects](#)[Interview Prep Sheets](#)[Striver's CP Sheet](#)January 10, 2023 ▪ [Data Structure / Graph](#)

# G-40: Number of Ways to Arrive at Destination

You are in a city that consists of  $n$  intersections numbered from  $0$  to  $n - 1$  with **bi-directional** roads between some intersections. The inputs are generated such that you can reach any intersection from any other intersection and that there is at most one road between any two intersections.

You are given an integer  $n$  and a 2D integer array 'roads' where  $\text{roads}[i] = [u_i, v_i, \text{time}_i]$  means that there is a road between intersections  $u_i$  and  $v_i$  that takes  $\text{time}_i$  minutes to travel. You want to know in how many ways you can travel from intersection  $0$  to intersection  $n - 1$  in the **shortest amount of time**.

Search

**Latest Video on takeUforward**

St...

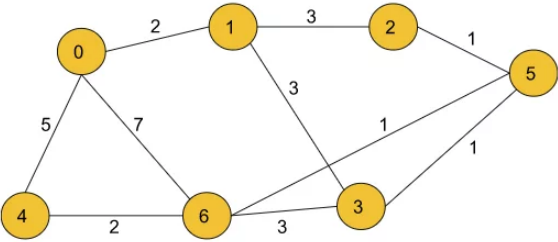


**Latest Video on Striver**

Return *the **number of ways** you can arrive at your destination in the shortest amount of time.*

Since the answer may be large, return it modulo  $10^9 + 7$ .

Example 1:



**Input :**  
n=7, m=10  
edges= [[0,6,7],[0,1,2],[1,2,3],[1,3,3],[1,5,1],[2,5,1],[3,6,1],[4,0,5],[4,6,2],[5,3,1]]

**Output :**  
4

**Explanation :**  
The four ways to get there in 7 minutes are:  
- 0 6  
- 0 4 6  
- 0 1 2 5 6  
- 0 1 3 5 6

Example 2:



Recent Posts

[Floor and Ceil in Sorted Array](#)

[Search Insert Position](#)

[Implement Upper Bound](#)

[Implement Lower Bound](#)

[2023 – Striver’s SDE Sheet Challenge](#)

**Input :**

n=6, m=8

edges= [[0,5,8],[0,2,2],[0,1,1],[1,3,4],[3,5,4],[3,4,5],[4,5,1],[5,0,1]]

**Output:**

3

**Explanation:**

The three ways to get there in 8 minutes are:

- 0 5
- 0 2 5
- 0 1 3 4 5

## Solution

**Disclaimer:** Don't jump directly to the solution, try it out yourself first.

### [Problem Link](#)

**Note:** In case any image/dry run is not clear please refer to the video attached at the bottom.

**Intuition:** Since there can be many paths to reach a destination node from the given

source node, in this problem, we have to find all those paths that are the **shortest** in order to reach our destination. For an easier understanding of this particular problem, we can say that we can divide the problem into partitions such as illustrated below :

From the above picture, we may assume that there will be 3 shortest paths to the destination node. But that may not be the case every time. Let us understand how –  
We assume the total number of ways in which the destination node is reachable by the shortest possible distance be  $\text{ways}[\text{node}]$  where 'node' depicts the destination node and node1, node2 and node3 are the three nodes which act as intermediate nodes that provide shortest paths to the destination. We can say :

$$\text{ways}[\text{node}] = \text{ways}[\text{node1}] +$$

```
ways[node2] + ways[node3]
```

Where,  $\text{ways}[\text{node1}]$ ,  $\text{ways}[\text{node2}]$ , and  $\text{ways}[\text{node3}]$  are the number of shortest paths possible to node1, node2, and node3 respectively from the source node, the sum of which is the total possible shortest paths and that can be hence greater than 3.

### Approach:

This problem is based on Dijkstra's Algorithm where we count all the possible shortest paths from the source to the destination node.

### Initial configuration:

- **Priority Queue:** Define a Priority Queue which would contain pairs of the type  $\{\text{dist}, \text{node}\}$ , where 'dist' indicates the currently updated value of the shortest dist taken to reach from source to the current 'node'.
- **Distance Array:** Define a distance array that would contain the minimum distance from the start node to the current node. If a cell is marked as 'infinity' then it is treated as unreachable/ unvisited.
- **Source Node:** Define the start node from where we have to calculate the total number of shortest paths.
- **Ways Array:** Define a ways array which would contain the number of possible

shortest ways/paths for each node.

Eventually, we would want to return  $\text{ways}[n-1]$  where  $n = \text{Number of nodes}$ .

The Algorithm consists of the following steps :

- Start by creating an adjacency list, a priority queue that stores the dist-node pairs in the form  $\{\text{dist}, \text{node}\}$  and a dist array with each node initialized with a very large number ( to indicate that the nodes have not been visited initially).
- In addition to the standard configuration of Dijkstra's algorithm, we have one more array in this problem by the name 'ways' which is initialized to '0' for every node when they're unvisited (so the number of ways is 0).
- Now, we push the start node to the queue along with its distance marked as '0' and ways marked as '1' initially because we've just started the algorithm.
- Pop the element from the front of the queue and look out for its adjacent nodes.
- If the current dist value for a number is better than the previous distance indicated by the distance array, we update the distance in the array and push it to the queue. Now, here side by side we also keep the number of ways to the 'node' the same as before.

- If the current dist value is the **same** as the previously stored dist value at the same index, increment the number of ways by 1 at that index.
- We repeat the above steps until the queue becomes empty or till we reach the destination.
- Return the ways[n-1] modulo  $10^9+7$  when the queue becomes empty.

Here's a quick demonstration of the Algorithm's 1st iteration for **example 1** stated above ( all the further iterations would be done in a similar way ) :

***Note: If you wish to see the dry run of the above approach, you can watch the video attached to this article.***

**Code:**

---

**C++ Code**

```

#include <bits/stdc++.h>
using namespace std;

class Solution
{
public:
    int countPaths(int n, vector<vec
    {
        // Creating an adjacency lis
        vector<pair<int, int>> adj[n]
        for (auto it : roads)
        {
            adj[it[0]].push_back({it
            adj[it[1]].push_back({it
        }

        // Defining a priority queue
        priority_queue<pair<int, int>
            vector<pair<i

        // Initializing the dist arr
        // along with their first in
        vector<int> dist(n, 1e9), wa
        dist[0] = 0;
        ways[0] = 1;
        pq.push({0, 0});

        // Define modulo value
        int mod = (int)(1e9 + 7);

        // Iterate through the graph
        // just as we do in Dijkstra
        while (!pq.empty())
        {
            int dis = pq.top().first
            int node = pq.top().seco
            pq.pop();

            for (auto it : adj[node]
            {
                int adjNode = it.fir
                int edW = it.second;

                // This 'if' conditi
                // time we're coming

```



```

        // in PQ and keep th
        if (dis + edW < dist
        {
            dist[adjNode] =
            pq.push({dis + e
            ways[adjNode] = '
        }

        // If we again encou
        // as before, we sim
        else if (dis + edW =
        {
            ways[adjNode] =
        }
    }
}
// Finally, we return the no
// (n-1)th node modulo 10^9+
return ways[n - 1] % mod;
}
};

int main()
{
    // Driver Code.
    int n = 7;

    vector<vector<int>> edges = {{0,
    {3, 5, 1}, {6, 5, 1}, {2, 5, 1},

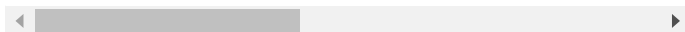
    Solution obj;

    int ans = obj.countPaths(n, edge

    cout << ans;
    cout << endl;

    return 0;
}

```



**Output :**

4

**Time Complexity:**  $O(E \cdot \log(V))$  { As we are using simple Dijkstra's algorithm here, the time complexity will be of the order  $E \cdot \log(V)$  }

Where  $E$  = Number of edges and  $V$  = No. of vertices.

**Space Complexity :**  $O(N)$  { for dist array + ways array + approximate complexity for priority queue }

Where,  $N$  = Number of nodes.

## Java Code

```
import java.util.*;

class Pair {
    int first;
    int second;
    public Pair(int first, int
second) {
        this.first = first;
        this.second = second;
    }
}

class Solution {

    static int countPaths(int n,
List < List < Integer >> roads) {

        // Creating an adjacency
list for the given graph.
        ArrayList < ArrayList <
Pair >> adj = new ArrayList < >
();
        for (int i = 0; i < n;
i++) {
            adj.add(new ArrayList
< > ());
        }
    }
}
```

```

        int m = roads.size();
        for (int i = 0; i < m;
i++) {

adj.get(roads.get(i).get(0)).add(n
ew Pair(roads.get(i).get(1),
roads.get(i).get(2)));

adj.get(roads.get(i).get(1)).add(n
ew Pair(roads.get(i).get(0),
roads.get(i).get(2)));
        }

        // Defining a priority
queue (min heap).
        PriorityQueue < Pair > pq
= new PriorityQueue < Pair > ((x,
y) -> x.first - y.first);

        // Initializing the dist
array and the ways array
        // along with their first
indices.
        int[] dist = new int[n];
        int[] ways = new int[n];
        for (int i = 0; i < n;
i++) {
            dist[i] = (int) 1e9;
            ways[i] = 0;
        }
        dist[0] = 0;
        ways[0] = 1;
        pq.add(new Pair(0, 0));

        // Define modulo value
        int mod = (int)(1e9 + 7);

        // Iterate through the
graph with the help of priority
queue
        // just as we do in
Dijkstra's Algorithm.
        while (pq.size() != 0) {

```

```
        int dis =
pq.peek().first;
        int node =
pq.peek().second;
        pq.remove();

        for (Pair it :
adj.get(node)) {
            int adjNode =
it.first;
            int edW =
it.second;

            // This 'if'
condition signifies that this is
the first

            // time we're
coming with this short distance,
so we push

            // in PQ and keep
the no. of ways the same.
            if (dis + edW <
dist[adjNode]) {
                dist[adjNode]
= dis + edW;
                pq.add(new
Pair(dis + edW, adjNode));
                ways[adjNode]
= ways[node];
            }

            // If we again
encounter a node with the same
short distance

            // as before,
we simply increment the no. of
ways.

            else if (dis +
edW == dist[adjNode]) {
                ways[adjNode]
= (ways[adjNode] + ways[node]) %
mod;
            }
}
```

```

    }
    }
    // Finally, we return the
    no. of ways to reach
    // (n-1)th node modulo
    10^9+7.
    return ways[n - 1] % mod;
}
}

```

```

class Main {

    public static void
    main(String[] args) {

        int n = 7;
        List < List < Integer >>
        roads = new ArrayList < > () {
            {
                add(new
                ArrayList<Integer>
                (Arrays.asList(0, 6, 7)));
                add(new
                ArrayList<Integer>
                (Arrays.asList(0, 1, 2)));
                add(new
                ArrayList<Integer>
                (Arrays.asList(1, 2, 3)));
                add(new
                ArrayList<Integer>
                (Arrays.asList(1, 3, 3)));
                add(new
                ArrayList<Integer>
                (Arrays.asList(6, 3, 3)));
                add(new
                ArrayList<Integer>
                (Arrays.asList(3, 5, 1)));
                add(new
                ArrayList<Integer>
                (Arrays.asList(6, 5, 1)));
                add(new
                ArrayList<Integer>
                (Arrays.asList(2, 5, 1)));
            }
        }
    }
}

```

```

        add(new
ArrayList<Integer>
(Arrays.asList(0, 4, 5)));
        add(new
ArrayList<Integer>
(Arrays.asList(4, 6, 2)));

    }
};
    Solution obj = new
Solution();
    int ans =
obj.countPaths(n, roads);

    System.out.print(ans);
    System.out.println();
}
}

```

### Output :

4

**Time Complexity:**  $O(E \cdot \log(V))$  { As we are using simple Dijkstra's algorithm here, the time complexity will be or the order  $E \cdot \log(V)$  }

Where  $E$  = Number of edges and  $V$  = No. of vertices.

**Space Complexity :**  $O(N)$  { for dist array + ways array + approximate complexity for priority queue }

Where,  $N$  = Number of nodes.

Special thanks to [Priyanshi Goel](#) for contributing to this article on takeUforward. If you also wish to share

your knowledge with the takeUforward  
fam, [please check out this article](#). If you  
want to suggest any  
improvement/correction in this article  
please mail us at [write4tuf@gmail.com](mailto:write4tuf@gmail.com)

---

« Previous Post

**G-39: Minimum  
Multiplications to  
Reach End**

Next Post »

**Count Square  
Submatrices with All  
1s | DP on  
Rectangles : DP 56**

Load Comments

---



The best place to learn data structures, algorithms, most asked coding interview questions, real interview experiences free of cost.

Follow Us



**DSA Playlist**

Array Series

Tree Series

Graph Series

**DSA Sheets**

Striver's SDE Sheet

Striver's A2Z DSA Sheet

SDE Core Sheet

**Contribute**

Write an Article

**Copyright © 2023 takeuforward | All rights reserved**