RV College of Engineering®
Mysore Road, RV Vidyaniketan Post,
Bengaluru - 560059, Karnataka, India

Go, change the world®

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Coding Challenge - 1

REACH THE QUEEN PROBLEM

REPORT

Submitted by

Name : MANOJ KUMAR B V **USN : 1RV23CS407**

Under the guidance of

Dr. Chethana R Murthy
Associate Professor,
Dept. of CSE - RVCE

Experiential Learning

Design and Analysis of Algorithms – CD343AI

IV Semester – B Section

AY: 2023-2024

Question

There are N cities and M directed roads connecting them. The king is at city 1 and his queen is at city N. He needs to find the number of ways to reach the queen. The cities are connected in a way that they don't form directed cycles.

Input Format

The first input line has two integers N and M. Then there are M lines describing the roads. Each line has two integers a and b i.e there is a directed road from city a to city b

Output Format:

Print the number of ways modulo 10^9 **+ 7**

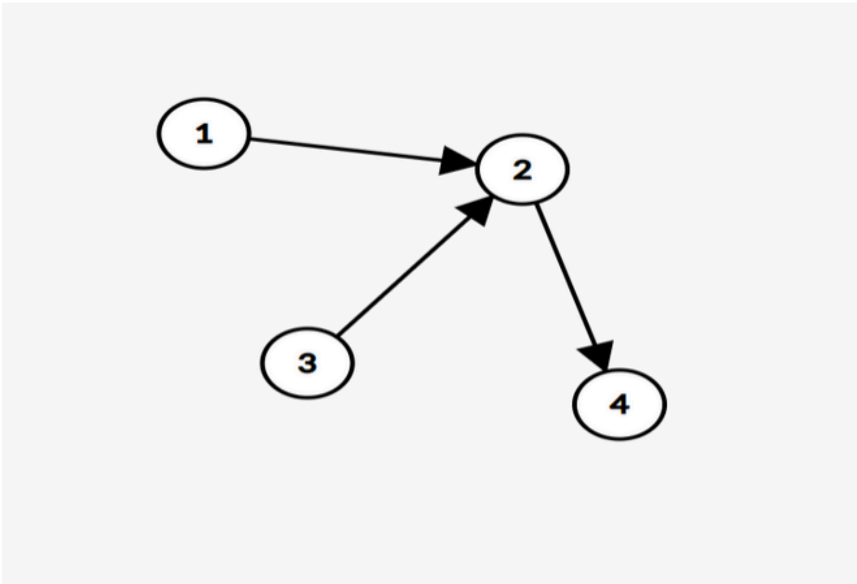
Constraints:

$1 \leq N \leq 2 * 10^5$
$1 \leq M \leq 4 * 10^5$
$1 \leq a, b \leq N$

Sample Input	Sample Output
4 3 1 2 2 4 3 2	1

Time Limit: 3
Memory Limit: 256
Source Limit:

Explanation



There is only 1 way to reach from 1 to 4 i.e 1->2->4

Details on Data Structures and Algorithm

Data Structures:

Graph Representation:

Graph (Adjacency List): The graph is represented using a dictionary where each key is a node, and its value is a list of nodes that it points to. This allows efficient traversal and edge exploration.

In-Degree Array: An array that keeps track of the number of incoming edges for each node. This is essential for Kahn's algorithm to determine the nodes with no incoming edges (in-degree of 0).

2. Topological Sort:

Queue: A deque is used to implement Kahn's algorithm for topological sorting. It helps in efficiently managing the nodes with an in-degree of 0.

3. Dynamic Programming Array:
DP Array: An array where `dp[i]` represents the number of ways to reach node `i` from node 1. This is updated as the graph is traversed in topological order.

Algorithm:

1. Graph Construction and In-Degree Calculation:
For each edge (u, v), add v to the adjacency list of u and increment the in-degree of v.

2. Topological Sorting using Kahn's Algorithm:
Initialize a queue with all nodes having an in-degree of 0.
While the queue is not empty, remove a node, add it to the topological order, and reduce the in-degree of its neighbors by 1. If any neighbor's in-degree becomes 0, add it to the queue.

3. Path Counting using Dynamic Programming:
Initialize `dp[1]` to 1, as there is one way to reach node 1 (starting point).
For each node in the topological order, update the dp array for its neighbors.
Specifically, for each neighbor, add the current node's dp value to the neighbor's dp value, taking modulo ($10^9 + 7$).

Code Snippets Screenshots

Enter your code or Upload your code as file. Save Python 3.8 (python 3.8.10)

```python
from collections import defaultdict, deque
import sys

MOD = 10**9 + 7

def count_paths(N, M, edges):
    graph = defaultdict(list)
    indegree = [0] * (N + 1)

    # Build the graph and compute indegree of each node
    for u, v in edges:
        if 1 <= u <= N and 1 <= v <= N:  # Check to ensure nodes are within bounds
            graph[u].append(v)
            indegree[v] += 1

    # Kahn's algorithm for topological sorting
    topo_order = []
    queue = deque([i for i in range(1, N + 1) if indegree[i] == 0])

    while queue:
        node = queue.popleft()
        topo_order.append(node)
        for neighbor in graph[node]:
            indegree[neighbor] -= 1
            if indegree[neighbor] == 0:
                queue.append(neighbor)

    # Dynamic programming to count paths
    dp = [0] * (N + 1)
```

20:17 vscode

Enter your code or Upload your code as file. Save Python 3.8 (python 3.8.10)

```python
    dp = [0] * (N + 1)
    dp[1] = 1

    for node in topo_order:
        for neighbor in graph[node]:
            dp[neighbor] = (dp[neighbor] + dp[node]) % MOD

    return dp[N]

# Reading input
input = sys.stdin.read
data = input().strip().split()

N = int(data[0])
M = int(data[1])
edges = []

index = 2
for _ in range(M):
    a = int(data[index])
    b = int(data[index + 1])
    if 1 <= a <= N and 1 <= b <= N:  # Ensure edges are within valid range
        edges.append((a, b))
    index += 2

# Calculating the number of ways to reach city N
result = count_paths(N, M, edges)
print(result)
```

20:17 vscode

20:17 vscode

Test against custom input

Compile & Test code | Submit code

Log ID: 260581871 / Jun 26, 2024 01:07 PM IST (Asia/Kolkata)

RESULT: Sample Test Cases Passed

Refer judge environment

Note: When you **Compile & Test code**, the code is run against sample inputs. When you **Submit code**, the code is run against sample input as well as multiple hidden test cases. In order to solve the problem, your code must pass all of the test cases.

Time (sec)	Memory (KiB)	Language
0.018994	2	Python 3.8

Input

```
4 3
1 2
2 4
3 2
```

Output

```
1
```

Expected Correct Output

```
1
```

20:17 vscode

Test against custom input

Compile & Test code | Submit code

Submission ID: 94680595

RESULT: Accepted

Refer judge environment

Score	Time (sec)	Memory (KiB)	Language
0	1.79864	79492	Python 3.8

Input	Result	Time (sec)	Memory (KiB)	Score	Your output	Correct output	Diff
Input #1	Accepted	0.017469	2	10			
Input #2	Accepted	0.017213	2	10			
Input #3	Accepted	0.017336	2	10			
Input #4	Accepted	0.016812	2	10			
Input #5	Accepted	0.01673	2	15			
Input #6	Accepted	0.568596	79492	15			
Input #7	Accepted	0.633895	79492	15			
Input #8	Accepted	0.510593	79492	15			

Time, Space Efficiency

Time Complexity:
Graph Construction: (O(M)) where M is the number of edges. Each edge is processed once.
Topological Sorting (Kahn's Algorithm): (O(N + M)) where N is the number of nodes and M is the number of edges. Each node and edge is processed once.
Dynamic Programming Update: (O(N + M)). Each node and edge is processed in the order given by the topological sort.

Overall time complexity is (O(N + M)).

Space Complexity:

Graph Storage: (O(N + M)). Adjacency list representation takes space proportional to the number of nodes and edges.
In-Degree Array: (O(N)**).**
Queue for Topological Sort: (O(N)) in the worst case.
DP Array: (O(N)).

Overall space complexity is (O(N + M)).

Conclusion

The algorithm efficiently calculates the number of distinct paths from node 1 to node N in a DAG using topological sorting and dynamic programming. The time and space complexities are both (O(N + M)), making it suitable for large graphs within the problem constraints.