Visiting Cities Hackerrank Solution Python

Visiting cities hackerrank solution python is a popular problem found on the HackerRank platform, a competitive programming website that provides coding challenges and contests for developers and programmers. The problem typically revolves around determining the best route for visiting a set of cities while adhering to specific constraints. This article will explore a comprehensive guide to solving this problem using Python, including an overview of the problem statement, a breakdown of the approach to the solution, a detailed explanation of the code, and some tips for optimizing your solution.

Understanding the Problem Statement

The visiting cities problem usually describes a scenario where you have a group of cities, each uniquely identified by a number. You are given a start city and a destination city, and you need to find a way to visit each city exactly once before returning to the start city. The challenge is to determine if such a route is possible and, if so, to calculate the distance or cost of traversing that route.

Here's a simplified version of the problem statement:

- You are given a number of cities and a set of direct paths between them, each with an associated distance.
- You need to determine if it is possible to visit all cities starting from a given city and returning to it.
- If possible, compute the minimum distance required to complete the trip.

Breaking Down the Approach

To solve the visiting cities problem effectively, we can break down the approach into several key steps:

1. Graph Representation

The cities and their connections can be represented as a graph. In this representation:

- Each city is a node.
- Each direct path between cities is an edge, with weights corresponding to the distance between the cities.

You can use a dictionary or an adjacency list to represent this graph.

2. Depth-First Search (DFS) or Backtracking

To explore all possible routes, we can use DFS or a backtracking algorithm. This approach will allow us to traverse the graph while keeping track of visited cities and calculating the total distance traveled.

3. Checking Validity

While traversing, we need to ensure that:

- Each city is visited exactly once.
- We return to the starting city after visiting all other cities.

4. Distance Calculation

As we traverse the graph, we keep a running total of the distance traveled. At the end of the traversal, if we have successfully visited all cities, we can compare this distance with previously calculated distances to find the minimum.

5. Edge Cases

Consider edge cases such as:

- No paths exist between certain cities.
- More cities than paths.
- All cities are connected.

Implementing the Solution in Python

Now that we have a clear approach, let's implement the solution in Python. Below is an example code that demonstrates how to solve the visiting cities problem using DFS:

```python
def visitingCities(n, paths):
from collections import defaultdict

Create a graph representation using an adjacency list
graph = defaultdict(list)
for u, v, d in paths:
graph[u].append((v, d))
graph[v].append((u, d)) Since the graph is undirected

Variables to keep track of the minimum distance
min_distance = float('inf')

Function to perform DFS
def dfs(city, visited, current_distance, count):
nonlocal min_distance

If all cities have been visited and we're back to the start
if count == n and city == 0:
min_distance = min(min_distance, current_distance)
return

Visit adjacent cities
for neighbor, distance in graph[city]:
if not visited[neighbor]:
visited[neighbor] = True Mark as visited
dfs(neighbor, visited, current_distance + distance, count + 1)
visited[neighbor] = False Unmark after backtracking

Start DFS from the first city (0)
visited = [False] n
visited[0] = True Starting city visited
dfs(0, visited, 0, 1) Start DFS

return min_distance if min_distance != float('inf') else -1

Example usage
```

```
n = 4 Number of cities
paths = [(0, 1, 10), (1, 2, 10), (2, 3, 10), (3, 0, 10), (0, 2, 15)]
print(visitingCities(n, paths)) Output minimum distance or -1 if not possible
```

Code Explanation

Let's break down the code step by step:

1. Graph Representation:
- We use a `defaultdict` from the `collections` module to create an adjacency list for the graph. Each city points to a list of tuples, where each tuple represents a neighboring city and the distance to it.

2. DFS Function:
- The `dfs` function takes the current city, a list to track visited cities, the current distance traveled, and a count of visited cities.
- If we have visited all cities and returned to the starting point, we update the minimum distance.

3. Visiting Cities:
- We begin DFS from the starting city (city 0 in this case), marking it as visited.
- We recursively call the `dfs` function for each unvisited neighboring city, updating the visited list accordingly.

4. Returning the Result:
- After executing DFS, we check if we found a valid route. If `min_distance` remains `inf`, it means no valid route exists, and we return -1.

Optimizing the Solution

While the above solution works, it may not be efficient for larger graphs due to its exponential time complexity. To optimize:

- Memoization: Store results of previously computed states to avoid redundant calculations.
- Dynamic Programming: Implement a more sophisticated approach using dynamic programming to reduce the state space.
- Heuristic Methods: For very large graphs, consider heuristic approaches like Genetic Algorithms or Ant Colony Optimization.

Conclusion

The visiting cities problem on HackerRank offers an excellent opportunity to practice graph traversal techniques and deepen your understanding of Python programming. By representing the cities as a graph and employing a systematic approach, you can effectively determine the best route to visit all specified cities. The provided solution, while straightforward, can be optimized further depending on the problem's constraints. As you practice, consider exploring various graph algorithms and techniques to enhance your coding skills and problem-solving abilities.

Frequently Asked Questions

What is the 'Visiting Cities' problem on HackerRank?

The 'Visiting Cities' problem on HackerRank typically involves optimizing the route to visit a series of cities while minimizing the total travel cost or distance, often requiring the use of graph algorithms.

What data structures are commonly used to solve the 'Visiting Cities' problem?

Common data structures include graphs (using adjacency lists or matrices), priority queues (for Dijkstra's algorithm), and sets or dictionaries for tracking visited nodes.

Which algorithm is most suitable for solving the 'Visiting Cities' problem?

Dijkstra's algorithm is often suitable for finding the shortest paths in weighted graphs, while other algorithms like Floyd-Warshall may be used for all-pairs shortest paths.

How do you represent cities and roads in Python for this problem?

Cities can be represented as nodes in a graph, and roads as edges with weights. You can use a dictionary to map each city to its connected cities and their respective travel costs.

What are some common pitfalls to avoid when solving the 'Visiting Cities' problem?

Common pitfalls include not handling negative weights correctly, forgetting to check for cycles in directed graphs, and inefficiently implementing the algorithm leading to timeouts on larger inputs.

Can you provide a simple Python code snippet to implement Dijkstra's

algorithm for this problem?

Certainly! Here's a simple snippet:

```python
import heapq

def dijkstra(graph, start):
min_heap = [(0, start)]
distances = {city: float('inf') for city in graph}
distances[start] = 0

while min_heap:
current_distance, current_city = heapq.heappop(min_heap)

if current_distance > distances[current_city]:
continue

for neighbor, weight in graph[current_city].items():
distance = current_distance + weight
if distance < distances[neighbor]:
distances[neighbor] = distance
heapq.heappush(min_heap, (distance, neighbor))
return distances
```

How can I test my solution for the 'Visiting Cities' problem?

You can test your solution by creating unit tests with sample inputs and expected outputs. Additionally, use HackerRank's test cases and edge cases to validate your implementation.

What resources can help improve my problem-solving skills for 'Visiting Cities' on HackerRank?

Resources include HackerRank's tutorials, competitive programming blogs, YouTube channels focused on algorithm challenges, and practice problems on platforms like LeetCode and Codeforces.

Visiting Cities Hackerrank Solution Python

visit□□□ing□□□visitting□□visiting□ - □□□□
visiting 1□□□□ □ ['vɪzɪtɪŋ] □ ['vɪzɪtɪŋ] 2□□□□ n. □□□□□□□□ adj. □□□ □□visit□ □□□□ □□. 3□□□□ London is a city worth visiting. □□□□□□□□□□□ 4□□□□ ...

visit□□□□□visiting□□□□□ - □□□□
visit□□□□□visiting□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□ visit□□□□□;□□ □□□ He wanted to visit his brother in Worcester. □□□□□□□□□□ ...

visit□□□□□□ - □□□□
They are visiting the Great Wall of China today. □□□□□□□□□□□□□□□ □ The tourists visited the Louvre Museum in Paris. □□□□□□□□□□□□□□□□ □ 4□Pay a visit ...

□□□□□□□□□□□□□ - □□□□
Dec 1, 2024 · □□□□□□□□□□□□□Warmly welcome the delegation of XXX for their visit.It is with great pleasure that we extend our warmest welcome to the delegation of XXX who are ...

□□□□□□ exchange student □ visiting student □□□
□□□□□ □□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□non-degree student□□□□□□□□□□□□□□□□□□□□□□□□□□□ ...

visiting fellow □visiting scholar □□□□□□_□□□□
visiting fellow □visiting scholar□□□□□□□□□□□□□□□□□□□□ □□□□□□ 1□visiting fellow□□□□□□□ 2□visiting scholar□□□□□□ □□□□□□ 1□visiting ...

□□□□□□namecard □□ business card? - □□□□
namecard □ business card □□□□□□□□ 1□business card □□□□□□□□□□□□□□□□□□□□□□□□□ □□When we met, he gave me his business card. □□□□□□□□ ...

travel□tour□journey□trip□□□□□□□□□□□□_□□□□
travel□tour□journey□trip□□□□ 1□Journey (n.)---“□□”,“□□”.□□□□,□□□□□□□□□□. 2□Tour (n.)---“□□”.□□□□□□□□□□□□□□□□□□□□□. 3□Trip (n.)---“□□”.□□□□ ...

□□□□□□□□□□□□ welcome to visit us □ □□□□ ...
Dec 24, 2010 · □□□□□welcome to visiting us□ □□□□□□□□□□□□□□welcome to visit us welcome□□□□□□□□□□□□ 1□□□□□□□welcome to China2□□□□□T

□□□□□*having visited* □*visiting*□□ - □□□□
Jun 11, 2022 · □□□□□having visited □visiting□□□□□□□□□□□□□□□□□□□□□□□□□visit□□□□□□□□□□□□□visited□□□□□□□□□□□□□□□□□□ ...

visit□□□ing□□□visitting□□visiting□ - □□□□
visiting 1□□□□ □ ['vɪzɪtɪŋ] □ ['vɪzɪtɪŋ] 2□□□□ n. □□□□□□□□ adj. □□□ □□visit□ □□□□ □□. 3□□□□ London is a city worth visiting. □□□□□□□□□□□ 4□□□□ ...

visit□□□□□visiting□□□□□ - □□□□
visit□□□□□visiting□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□ visit□□□□□;□□ □□□ He wanted to visit his brother in Worcester. □□□□□□□□□□ ...

visit□□□□□□ - □□□□
They are visiting the Great Wall of China today. □□□□□□□□□□□□□□□ □ The tourists visited the Louvre Museum in Paris. □□□□□□□□□□□□□□□□□□ □ 4□Pay a visit …

□□□□□□□□□□□□□□□ - □□□□
Dec 1, 2024 · □□□□□□□□□□□□□□Warmly welcome the delegation of XXX for their visit.It is with great pleasure that we extend our warmest welcome to the delegation of XXX who are …

□□□□□□ exchange student □ visiting student □□□
□□□□□□ □□□□□ □□non-degree student□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ …

visiting fellow □visiting scholar □□□□□□□_□□□□
visiting fellow □visiting scholar□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□ 1□visiting fellow□□□□□□□□□ 2□visiting scholar□□□□□□□ □□□□□□□ 1□visiting …

□□□□□□□namecard □□ business card? - □□□□
namecard □ business card □□□□□□□□□ 1□business card □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□When we met, he gave me his business card. □□□□□□□□□ …

travel□tour□journey□trip□□□□□□□□□□□□□□_□□□□
travel□tour□journey□trip□□□□□ 1□Journey (n.)---“□□”,“□□”.□□□□□,□□□□□□□□□□□. 2□Tour (n.)---“□□”.□□□□□□□□□□□□□□□□□□□□□□□□□□. 3□Trip (n.)---“□□”.□□□□□ …

□□□□□□□□□□□□□□ welcome to visit us □ □□□□□ …
Dec 24, 2010 · □□□□□□welcome to visiting us□ □□□□□□□□□□□□□□□□welcome to visit us welcome□□□□□□□□□□□□□ 1□□□□□□□□□welcome to China2□□□□□□T

□□□□□□*having visited* □*visiting*□□ - □□□□
Jun 11, 2022 · □□□□□□having visited □visiting□□□□□□□□□□□□□□□□□□□□□□□□□□□□□visit□□□□□□□□□□□□□□□□□visited□□□□□□□□□□□□□□□□□□□□ …

Discover the optimal Python solution for the Visiting Cities HackerRank challenge. Enhance your coding skills and tackle this problem effectively. Learn more!

Back to Home