HW4 Report

Zhiwei Jiang

December 5, 2024

1 Introduction

In this report, we address two problems related to graph traversal and shortest path calculation. The first problem deals with a shortest path problem with additional constraints, and the second focuses on determining the minimum initial health points (HP) required for a hero to traverse a map with monsters and attack powers.

2 Problem 1

2.1 Introduction

The problem involves determining the shortest path in an undirected, weighted graph where the objective is to pass through specific edges in a specified order. The challenge lies in finding an optimal path that incorporates these additional constraints while minimizing the total distance.

2.2 Possible Solutions

Several potential methods for solving this problem:

2.2.1 Brute Force Search

A brute force approach examines all possible paths that pass through the required edges. This method explores all permutations of the required edges and finds the shortest valid path. However, the brute force approach suffers from an exponential time complexity, making it impractical for larger graphs.

2.2.2 Modified DFS with Backtracking

DFS with backtracking can explore paths from the start node to the end node, ensuring that all required edges are visited. However, it lacks the ability to prioritize shorter paths, which can lead to excessive exploration of suboptimal routes. The time complexity remains high due to the exhaustive nature of DFS.

2.2.3 Dynamic Programming on Graphs

Dynamic programming, similar to the Held-Karp algorithm for the Traveling Salesman Problem, can store intermediate results for different states. However, it faces high space complexity, making it unsuitable for large graphs due to the curse of dimensionality.

2.3 Chosen Solution: Modified Dijkstra Algorithm with State Tracking

The selected solution is a **Modified Dijkstra Algorithm** with state tracking. This approach extends the classic Dijkstra algorithm by maintaining a state that tracks which required edges have been visited. A priority queue is used to explore the shortest paths efficiently.

2.3.1 Algorithm Overview

The algorithm maintains a priority queue that stores tuples of the form (current_distance, current_node, current_state), where `current_state` is represented by a bitmask. The algorithm iteratively selects the node with the shortest known distance and updates the state upon visiting a required edge.

2.3.2 Complexity Analysis

The time complexity of the approach is approximately:

$$O((V \cdot 2^k + E) \cdot (\log V + k))$$

where V is the number of nodes, E is the number of edges, and k is the number of required edges.

2.3.3 Strengths of the Chosen Solution

- **Optimal Path Guarantee**: The modified Dijkstra algorithm guarantees the optimal solution by always selecting the shortest path. - **Efficient State Representation**: The use of a bitmask to track visited edges efficiently manages the traversal state. - **Scalability for Small k**: The approach is scalable for problems where the number of required edges is small.

2.4 Comparison with Other Solutions

Compared to brute force and DFS-based methods, the modified Dijkstra approach offers significant improvements in time complexity and practical feasibility, particularly for larger graphs.

2.5 Conclusion

In conclusion, the Modified Dijkstra Algorithm provides an efficient and optimal solution for the shortest path problem with the additional constraint of passing through specific edges. This solution balances computational efficiency with optimality, making it a suitable choice for the problem at hand.

3 Problem 3

3.1 Introduction

The problem involves determining the minimum initial health points (HP) that a hero named Don needs in order to traverse a map represented as an undirected graph. Don must reach the destination with exactly zero HP and zero spirit points (SP), while passing through edges that have associated monsters with attack power. The goal is to find the minimum initial HP Don needs to start with in order to reach the destination while maintaining these conditions.

3.2 Possible Solutions

Several approaches can be used to solve this problem:

1. **Depth-First Search (DFS):** DFS can explore all possible paths between the start and end nodes, but it may not efficiently handle large graphs, leading to high computational costs.
2. **Breadth-First Search (BFS):** BFS can find the shortest path in terms of the number of edges but does not prioritize paths based on the minimum HP.
3. **Dijkstra's Algorithm / BFS with Priority Queue:** A modified version of Dijkstra's algorithm can be used to prioritize paths with the least HP cost. This is a suitable approach as it minimizes the HP required by exploring the least costly paths first.
4. **Dynamic Programming (DP):** DP could be used by defining states for the minimum HP required at each node with varying levels of SP. However, this would be complex and computationally expensive.

3.3 Chosen Approach

The chosen solution is **BFS with Priority Queue (Min-Heap)**. This approach is ideal for the problem because:

- **Optimal Path Exploration:** By using a priority queue, we explore nodes based on the least HP cost, which aligns perfectly with the goal of minimizing initial HP.

- **Handling Dynamic States:** The BFS with priority queue is suitable for state-changing problems. The state includes the current HP, the node, and the SP, which allows for efficient exploration.
- **Efficiency:** This approach avoids redundant calculations and ensures efficient exploration of paths by pruning suboptimal ones early on.

3.4 Solution Details

The implemented solution works as follows:

- **Graph Representation:** The graph is represented using an adjacency list, where each node has a list of neighbors and corresponding attack power on the edges.
- **Priority Queue (Min-Heap):** A min-heap is used to prioritize nodes with the least HP. Each entry includes the current HP, the node, and the SP.
- **State Management:** We calculate the damage as damage $= \frac{\text{attack\_power}}{\text{SP}}$ for each edge. SP decreases by 1 with each edge traversed.
- **Termination Condition:** The traversal continues until we reach the start node with the minimum HP, ensuring that both HP and SP are zero.

3.5 Why This Solution Is Better

The BFS with priority queue is better than other alternatives for the following reasons:

- **Optimal Path Selection:** It always explores the node with the least HP cost, ensuring that we find the optimal path.
- **Pruning Suboptimal Paths:** The priority queue allows us to discard suboptimal paths early, unlike DFS which might explore them deeply.
- **Scalability:** This approach scales well for large graphs, avoiding the complexity of DFS and DP, which would be computationally expensive for large graphs.

3.6 Conclusion

In conclusion, the BFS with priority queue (min-heap) approach was selected as it efficiently handles the dynamic state changes and minimizes the initial HP required. This solution is computationally feasible, scales well for large graphs, and successfully calculates the minimum initial HP needed for Don to complete her journey.