Group Project

Design Algorithm and Analysis

Presented by :

Aaren

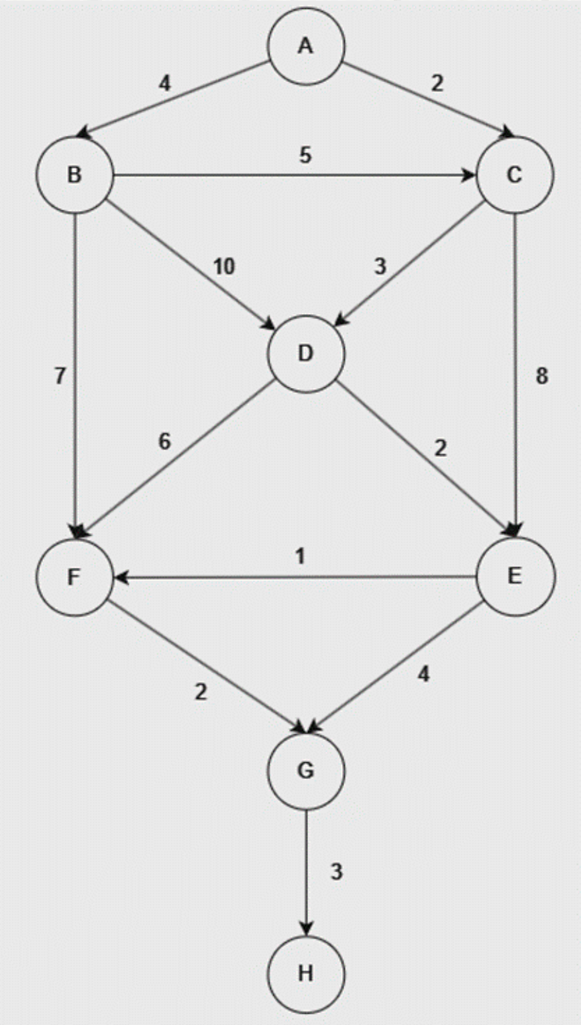
Khugan

Kabilan

Table of contents

Problem Statement

Mr.Kanavathy is organizing a large treasure hunt competition in a historical city where three teams— Team Alpha, Team Bravo, and Team Charlie—start from different landmarks (A, B, and C respectively) and must collect tokens from all other landmarks before reaching the final treasure at Landmark H. The city's layout is represented as a weighted graph with landmarks as nodes and pathways as edges with travel times. The goal is to ensure a fair, efficient, and enjoyable experience for all participants by optimizing their routes.



Geographical Settings: Malaysia
Type of Disaster: Not Applicable

Damage Impact:

- If one team inadvertently has a shorter or less challenging route to the treasure due to suboptimal path planning, it would lead to unfair competition.
- If travel times are not optimized, it could result in delays, causing frustration among participants and spectators.
- In case of an emergency, knowing the shortest paths allows for quicker response times from medical or security teams.
- Longer or more complex routes could increase the risk of participants getting lost or encountering unforeseen hazards

Importance of Advanced Algorithm Design (AAD):

- It handles the complexity of the city map, enabling the use of sophisticated algorithms like Dijkstra's, A*, and Breadth First Search (BFS) to find the shortest, safest routes.
- AAD optimizes resource allocation, enhances real-time adaptability, ensures fairness, improves safety and emergency response, and boosts participant satisfaction.
- AAD also minimizes the impact on the city's infrastructure, ensuring a smooth, enjoyable, and sustainable event.

Goal

- The goal of this project is to develop a robust pathfinding system using Breadth-First Search (BFS) and Dijkstra's algorithm.
- BFS will be used to identify all possible paths from each starting landmark to the final treasure, ensuring that all nodes (landmarks) are visited.
- Dijkstra's algorithm is to determine the shortest possible path for each team, minimizing their travel time and ensuring a balanced competition.
- This approach aims to optimize resource allocation, enhance participant safety, and ensure a smooth event flow.

Model Develpment

Data Types and Structures:

- The city's layout will be represented as a weighted graph, where landmarks are nodes and pathways between them are edges with non-negative weights representing travel times

Objective Function:

- The primary objective is to minimize the total travel time for each team from their respective starting landmarks to the final treasure at Landmark H, while ensuring they visit all other landmarks to collect tokens.

Constraints:

1. **Mandatory Visit:** Each team must visit all landmarks at least once before reaching the treasure.
2. **Travel Time:** Paths should minimize the sum of travel times between landmarks.
3. **Fairness:** The shortest path calculated should be the same for all teams to ensure a fair competition.
4. **Safety:** Paths must avoid unsafe areas or pathways marked as hazardous.
5. **Environmental Impact:** Paths should minimize repeated use of specific pathways to prevent environmental degradation.

Why finding optimal solution is important

- **Fair Competition:** It's critical to guarantee that every team has an equal opportunity to gather every token and get to the ultimate treasure without facing any unfair advantages. The fairness of the competition would be jeopardised if one team found a more efficient way to finish the work much faster than the others.

- **Time Efficiency:** Identifying the shortest path for every team reduces the overall amount of time needed, as travel times between landmarks differ. This keeps competitors interested and cuts down on downtime by ensuring the competition goes smoothly and ends on time

- **Safety:** Reducing the amount of time people spend travelling also improves safety, particularly in ancient cities where pathways may be congested or challenging to negotiate. It guarantees that players can finish the search both swiftly and securely.

Algorithm Suitability

Algorithm	Suitability	Strengths	Weakness
Sorting	Low	Simple and well understood	Does not address the pathfinding problem
Divide and Conquer (DAC)	Low to Moderate	Useful for breaking down complex problems	May not handle interconnected paths efficiently
Dynamic Programming (DP)	High	Suitable for problems with multiple optimal sub paths	Implementation can be complex
Greedy Algorithms	Moderate to High	• Simple and easy to implement • Efficient in terms of time and space	Can fail for complex pathfinding tasks with dependencies
Graph Algorithms	Very High	• Directly applicable to pathfinding and shortest path problems • Can handle weighted graphs and various constraints	• Can be computationally intensive for very large graphs • Requires a good understanding of graph theory and algorithms

ALGORITHMS DEFINITION

Breadth first search

- Used to find **all the possible paths**
- Breadth-First Search (BFS) is a way to explore a graph level by level. You start at a given node, then visit all its direct neighbors, then the neighbors' neighbors, and so on. BFS ensures that all nodes at the current level are visited before moving to the next level.

Dijkstra's Algorithm

- Used to find the **shortest path**
- Dijkstra's Algorithm finds the shortest path from one node to all other nodes in a graph with weighted edges. It starts at the initial node and always explores the cheapest path to each next node, updating paths as shorter ones are found.

HOW IT WORKS

Breadth first search

1. Start at the node you want to explore from.
2. Enqueue the starting node and mark it as visited.
3. Dequeue a node from the front of the queue.
4. Visit all its unvisited neighbors, mark them as visited, and enqueue them.
5. Repeat steps 3-4 until the queue is empty.

Dijkstra's Algorithm

1. Start at the node you want to find the shortest paths from.
2. Set the distance to the start node to 0 and all other nodes to infinity.
3. Pick the node with the smallest distance (initially the start node).
4. Update the distances to its neighboring nodes if you find a shorter path.
5. Mark this node as visited.
6. Repeat steps 3-5 until all nodes are visited.

TIME COMPLEXITY OF ALGORITHMS

Breadth First Algorithm (BFS)

Best Case: O(V + E)

The shortest paths are found almost immediately, with minimal exploration. For example, if the starting landmark is directly connected to the treasure, BFS quickly identifies the path with minimal node and edge traversal. The algorithm efficiently completes with little computational effort.

Average Case: O(N * b^d)

The algorithm explores multiple paths from each starting landmark to the treasure. For example, the teams need to navigate through several intermediate landmarks, expanding nodes level-by-level until viable paths are found. This involves moderate exploration and computational effort as BFS systematically searches through possible routes.

Worst Case: O(N * b^d)

The algorithm must explore every possible path extensively before finding the goal nodes. For instance, if the treasure is located at the maximum depth and all other paths need to be considered, BFS will exhaustively traverse every node and edge. This leads to significant computational effort as the algorithm explores the entire graph.

Dijkstra Algorithm

Best Case: O(N * V log V)

The shortest paths are quickly found with minimal updates and relaxations. For instance, if the graph is sparse or the shortest paths are evident early, Dijkstra's algorithm efficiently processes the nodes and edges with minimal computational effort. The teams quickly find their routes to the treasure.

Average Case: O(N * (V + E) log V)

The teams need to process a significant portion of the graph to find the shortest paths. For example, multiple updates and edge relaxations are required as the teams navigate through several landmarks. The algorithm involves a moderate computational effort to identify the shortest routes for each team.

Worst Case: O(N * (V + E) log V)

The graph is densely connected, and extensive updates and relaxations are necessary. For example, if the teams must consider numerous pathways with many nodes and edges, Dijkstra's algorithm will handle a large number of computations to find the shortest paths. This results in maximum computational effort as the algorithm processes the entire graph extensively.

Overall Algorithm

Best Case:
O(V + E + N * V log V)

All three teams quickly find the shortest paths from their starting landmarks to the final treasure at landmark H with minimal exploration. For instance, if the pathways are direct and the shortest paths are evident early in the process, the algorithm efficiently completes without extensive traversal or updates.This results in a streamlined process where the initial BFS finds possible paths swiftly and Dijkstra's algorithm quickly identifies the shortest routes for each team.

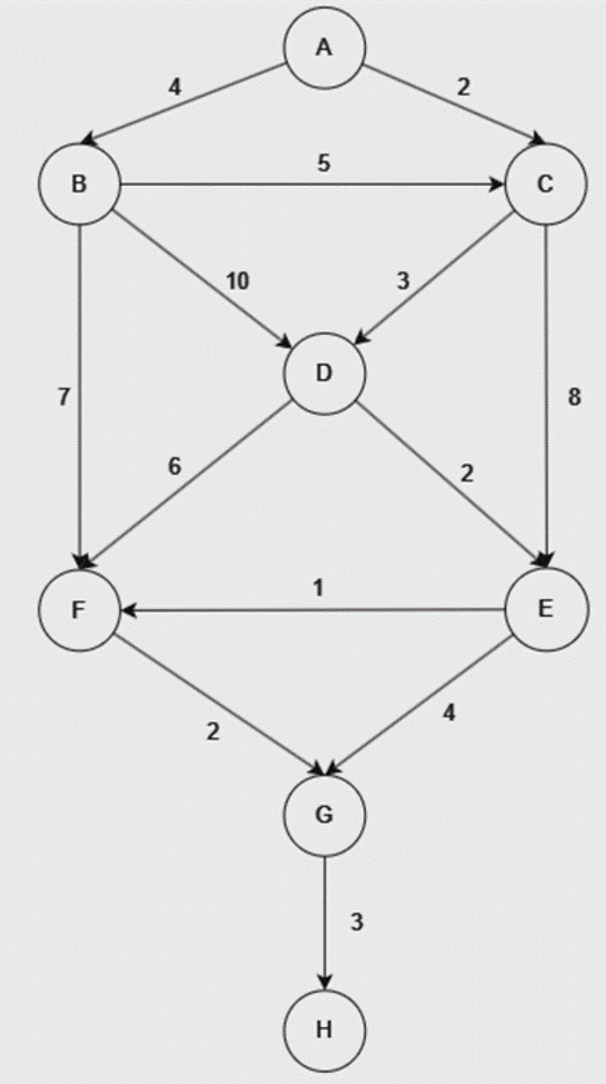
Average Case:
O(V + E + N * b^d + N * (V + E) log V)

The teams need to navigate through several intermediate landmarks, requiring moderate exploration of paths. Team Alpha, Bravo, and Charlie might need to explore multiple routes and landmarks before reaching the treasure. The BFS will explore a fair number of paths before determining viable options, and Dijkstra's algorithm will process a significant portion of the graph to find the shortest paths.

Worst Case:
O(V + E + N * b^d + N * (V + E) log V)

The teams must exhaustively explore all nodes and edges before finding the shortest paths to the treasure. For instance, if the treasure is located at the farthest possible landmark, and the pathways are complex and dense, the BFS will have to explore every possible path extensively. Dijkstra's algorithm will need to handle a large number of updates and relaxations due to the dense connections leading to maximum computational effort and resource usage.

Problem and Output

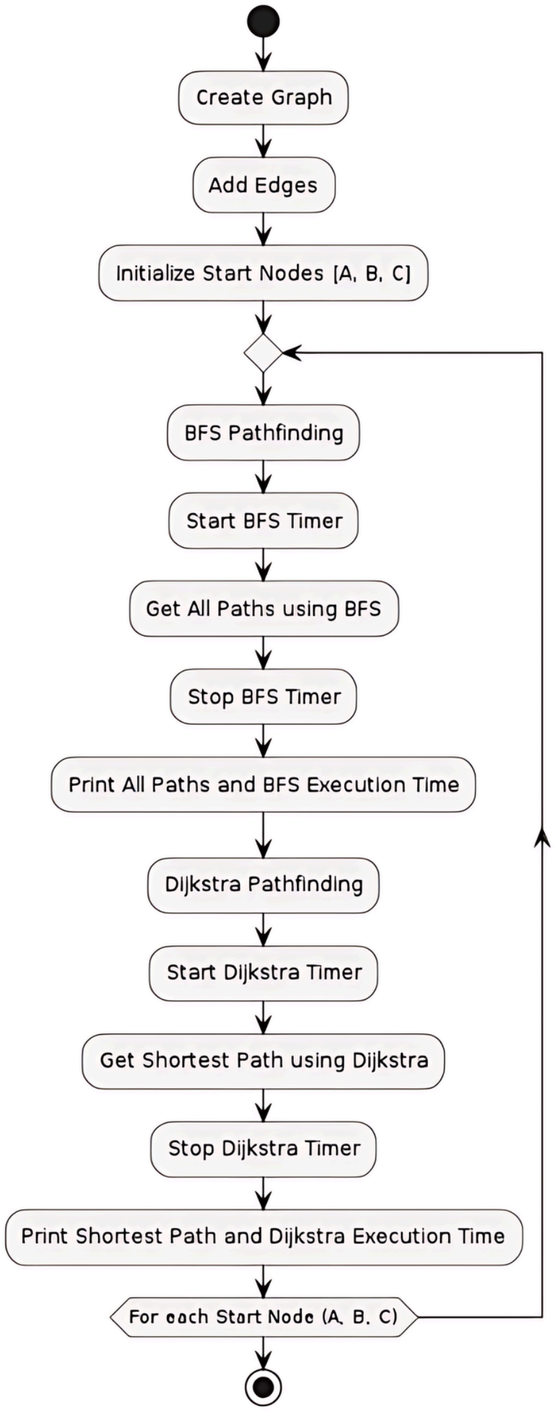



```
All paths from A to H:
[A, B, F, G, H] with distance 16
[A, C, E, G, H] with distance 17
[A, B, C, E, G, H] with distance 24
[A, B, D, E, G, H] with distance 23
[A, B, D, F, G, H] with distance 25
[A, C, D, E, G, H] with distance 14
[A, C, D, F, G, H] with distance 16
[A, C, E, F, G, H] with distance 16
[A, B, C, D, E, G, H] with distance 21
[A, B, C, D, F, G, H] with distance 23
[A, B, C, E, F, G, H] with distance 23
[A, B, D, E, F, G, H] with distance 22
[A, C, D, E, F, G, H] with distance 13
[A, B, C, D, E, F, G, H] with distance 20
Shortest path from A to H: [A, C, D, E, F, G, H] with distance 13
Dijkstra Execution time: 3.5004 ms

All paths from B to H:
[B, F, G, H] with distance 12
[B, C, E, G, H] with distance 20
[B, D, E, G, H] with distance 19
[B, D, F, G, H] with distance 21
[B, C, D, E, G, H] with distance 17
[B, C, D, F, G, H] with distance 19
[B, C, E, F, G, H] with distance 19
[B, D, E, F, G, H] with distance 18
[B, C, D, E, F, G, H] with distance 16
Shortest path from B to H: [B, F, G, H] with distance 12
Dijkstra Execution time: 0.0743 ms

All paths from C to H:
[C, E, G, H] with distance 15
[C, D, E, G, H] with distance 12
[C, D, F, G, H] with distance 14
[C, E, F, G, H] with distance 14
[C, D, E, F, G, H] with distance 11
Shortest path from C to H: [C, D, E, F, G, H] with distance 11
Dijkstra Execution time: 0.0301 ms
```

Flowchart


Create Graph
Add Edges
Initialize Start Nodes [A, B, C]
BFS Pathfinding
Start BFS Timer
Get All Paths using BFS
Stop BFS Timer
Print All Paths and BFS Execution Time
Dijkstra Pathfinding
Start Dijkstra Timer
Get Shortest Path using Dijkstra
Stop Dijkstra Timer
Print Shortest Path and Dijkstra Execution Time
For each Start Node (A, B, C)

Thank you very much!