AUTOMATED DRONE SIMULATOR

GROUP 14

CB.EN.U4CSE21322-JASWANTH G

CB.EN.U4CSE21328-K ABHISHEK

CB.EN.U4CSE21329-K HITEESH RAJU

CB.EN.U4CSE21361-SV SAI SATYA

PROBLEM STATEMENT

Design an automated drone simulator that operates in a 3D plane. The stimulator should enable the drone to navigate to a destination while avoiding obstacles, calculate the speed and time taken for the journey, and provide detailed analysis of the path, including nearby obstacles, altitude variations, and estimated time of arrival.

HYBRID DATA STRUCTURES DESCRIPTION:

Speed and Time Calculations:

Queue: Use a queue to store timestamps and corresponding locations during the drone's movement. This enables the calculation of speed by measuring the distance travelled between consecutive timestamps.

Priority Queue: Maintain a priority queue to track time-based events, such as scheduled stops or waypoint arrivals, which can be used to estimate the time taken for the entire journey.

Navigation and Path Planning:

Graph: Represent the 3D space as a graph, where each node represents a location or waypoint, and edges represent possible paths between them. This allows for efficient path planning algorithms like Dijkstra's algorithm or A* search to find the optimal route to the destination.

Priority Queue: Use a priority queue to prioritize waypoints or paths based on factors like distance, altitude, or estimated time of arrival. This helps in determining the next target location for the drone's navigation.

Path Analysis:

Graph: Analyse the graph representing the drone's path to extract relevant information, such as the altitude variations between waypoints or the path's topology. Algorithms like depth-first search or breadth-first search can be utilized for this purpose.

Spatial Indexing: Utilize spatial indexes to retrieve and analyse nearby obstacles or features along the drone's path, such as changes in altitude or terrain variations.

OPERATIONS AND THEIR TIME COMPLEXITY:

NAVIGATION AND PATH PLANNING:

The space complexity for graph representation is O(N + E), where N is the number of nodes and E is the number of edges.

The time complexity for path planning algorithms like Dijkstra's algorithm or A* search is O((N + E)log N), where N is the number of nodes and E is the number of edges.

The space complexity for a priority queue is O(K), where K is the number of elements stored in the priority queue.

The time complexity for priority queue operations is typically O(log K), where K is the number of elements in the priority queue.

SPEED AND TIME CALCULATIONS:

The space complexity for a queue is O(K), where K is the number of elements stored in the queue.

The time complexity for enqueue and dequeue operations in a queue is O(1) on average.

The space complexity for a priority queue is O(K), where K is the number of elements stored in the priority queue.

The time complexity for priority queue operations is typically O(log K), where K is the number of elements in the priority queue.

PATH ANALYSIS:

The space complexity for graph analysis algorithms like DFS or BFS is O(N), where N is the number of nodes in the graph.

The time complexity for graph analysis algorithms like DFS or BFS is O(N + E), where N is the number of nodes and E is the number of edges in the graph.

The space complexity for spatial indexing data structures is O(K), where K is the number of objects or features represented in the spatial index.

The time complexity for querying a spatial index is typically O(log K + M), where K is the number of objects in the spatial index and M is the number of results retrieved.

JUSTIFICATION OF TIME AND SPACE COMPLEXITY:

Graph Analysis (Depth-First Search or Breadth-First Search):

Space Complexity: The space complexity of graph analysis algorithms like depth-first search (DFS) or breadth-first search (BFS) depends on the number of nodes in the graph. If there are N nodes, the space complexity is O(N) since you may need to maintain a data structure (such as a visited set or queue) to track visited nodes.

Time Complexity: The time complexity of DFS and BFS also depends on the number of nodes and edges in the graph, denoted by N and E, respectively. In the worst case, both DFS and BFS visit all nodes and edges once, resulting in a time complexity of O(N + E).

Spatial Indexing:

Space Complexity: The space complexity of spatial indexing data structures like R-trees or k-d trees depends on the number of objects or features represented. If there are K objects, the space complexity is typically O(K) for storing the spatial index.

Time Complexity: The time complexity of querying a spatial index depends on the structure used. For example, for R-trees or k-d trees, the time complexity is generally O(log K + M), where K is the number of objects in the spatial index and M is the number of results retrieved.

SPEED AND TIME CALCULATIONS:

Queue:

Space Complexity: The space complexity of a queue depends on the number of elements stored in it. If there are K elements in the queue, the space complexity is O(K).

Time Complexity: The time complexity of enqueue and dequeue operations in a queue is O(1) on average, providing efficient insertion and retrieval of elements.

Priority Queue:

Space Complexity: The space complexity of a priority queue depends on the number of elements stored in it. If there are K elements in the priority queue, the space complexity is O(K).

Time Complexity: The time complexity of operations on a priority queue depends on the underlying implementation. Binary heaps, for example, provide O(log K) time complexity for insertion and deletion operations, where K is the number of elements in the priority queue.

NAVIGATION AND PATH PLANNING:

Graph Representation:

Space Complexity: The space complexity of representing a graph depends on the number of nodes and edges. For a graph representing a 3D space, if there are N nodes, the space complexity would be O(N) for storing the nodes and O(E) for storing the edges, where E is the number of edges in the graph.

Path Planning Algorithms (Dijkstra's algorithm or A* search):

Time Complexity: The time complexity of Dijkstra's algorithm and A* search depends on the number of nodes and edges in the graph, denoted by N and E, respectively. In the worst case scenario, the time complexity is O((N + E)log N) for Dijkstra's algorithm and O((N + E)log N) for A* search, where the logarithmic factor arises from the use of a priority queue to select the next node.

Priority Queue:

Space Complexity: The space complexity of a priority queue depends on the number of elements stored in it. If there are K elements in the priority queue, the space complexity is O(K).

Time Complexity: The time complexity of operations on a priority queue depends on the underlying implementation. Binary heaps, for example, provide O(log K) time complexity for insertion and deletion operations, where K is the number of elements in the priority queue.