DFS Movement Algorithm for Mr. X

1. Verbal Explanation

The **DFS Movement Algorithm** (Depth-First Search) is designed to find a long and safe path for Mr. X, ensuring he always remains ahead of the detectives. The algorithm explores possible routes in depth, starting from Mr. X's position and recursively extending the path while maintaining safety conditions.

Safety condition: For each visited field, Mr. X must have a strictly shorter distance to that field than any detective.

Core logic:

1. **Initialization:**
   - Start from the current position of Mr. X.
   - Prepare a list of neighboring fields (each treated as a separate search branch).
   - Keep a record of visited nodes to avoid loops.
   - Initialize memory for the longest safe path found.
2. **Depth Exploration:**
   - For each neighbor, perform a DFS search.
   - Each recursive step extends the current path if:
     (a) The field is not already visited.
     (b) Mr. X's distance to that field is shorter than any detective's.
     (c) The current route to that field is the shortest possible for Mr. X.
3. **Termination Conditions:**
   - The current path reaches the defined minimum satisfactory length.
   - The next field is unsafe (violates the safety condition).
   - The field has already been visited.
   - The maximum allowed depth or turn limit is reached.
4. **Handling Ferries and Black Tickets:**
   - Ferry connections (*ferry edges*) can only be used if Mr. X has a black ticket available.
   - Each use of a ferry consumes one black ticket.
5. **Direction Constraint:**
   - Each neighbor of Mr. X must be checked at least once.
   - DFS may terminate individual branches early, but all initial directions are tested independently.
6. **Result:**
   - If a sufficiently long safe path is found, return it.
   - If no such path exists, return the longest safe path discovered.

2. Flowcharts

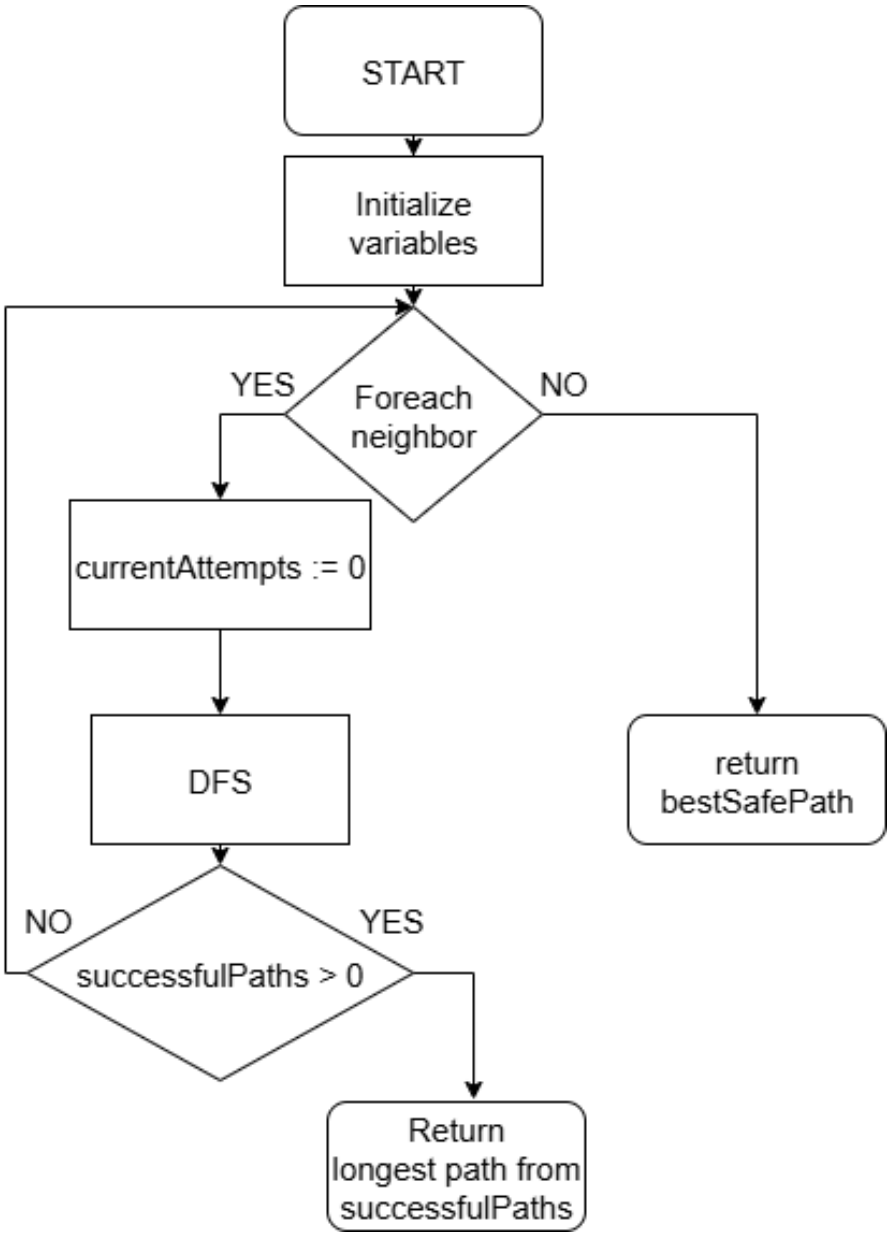

Figure 1: Overall DFS Movement Flowchart

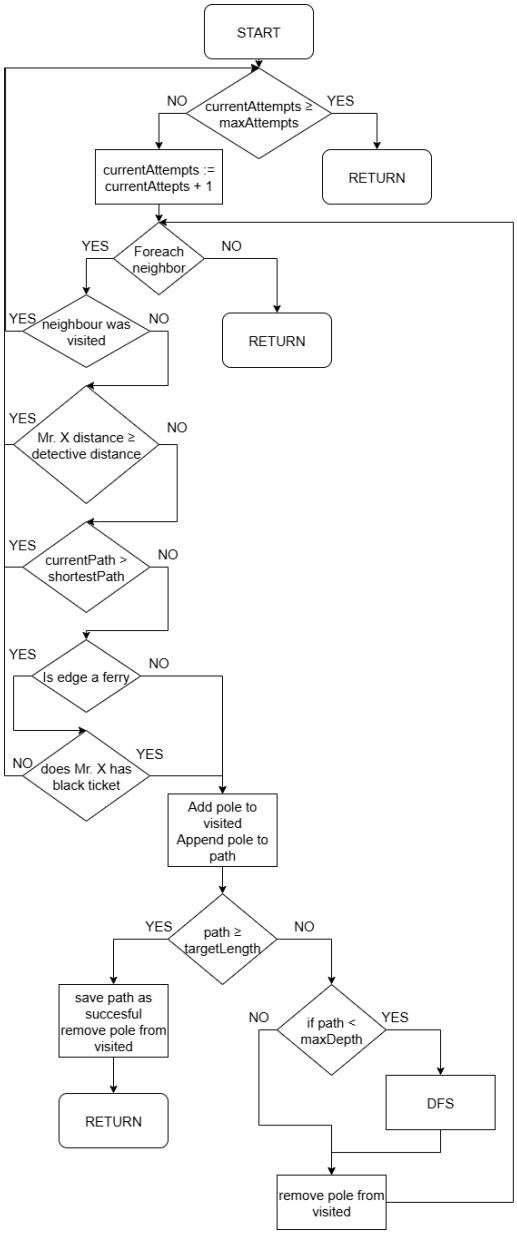

Figure 2: Detailed Flowchart of the Recursive DFS Function

3. Pseudocode

Algorithm 1: DFS Movement Algorithm for Mr. X (with black ticket handling, safety and per-neighbor attempt limits)

Input: $posX$, $posDetectives$, $graph$, $blackTickets$, parameters $targetLength$, $maxDepth$, $maxAttemptsPerNode$
Output: Longest safe path found

Function `DFS`(*current, path, visited, blackTickets, attempts*)**:**
  if $attempts[current] \geq maxAttemptsPerNode$ **then**
    return
  end
  $attempts[current] \leftarrow attempts[current] + 1$;
  foreach *neighbor in graph.neighbors*(*current*) **do**
    if *neighbor in visited* **then**
      continue;
    end
    if $distance(posX, neighbor) \geq min(distance(detective_i, neighbor))$ **then**
      continue;
    end
    if $currentPathLengthTo(neighbor) > shortestPath(posX, neighbor)$ **then**
      continue;
    end
    if *edge*(*current*, *neighbor*) *is ferry* **then**
      if $blackTickets == 0$ **then**
        continue;
      end
      $blackTickets \leftarrow blackTickets - 1$;
    end
    add *neighbor* to *visited*;
    append *neighbor* to *path*;
    if $|path| \geq targetLength$ **then**
      save *path* as successful result;
      remove *neighbor* from *visited*;
      remove last node from *path*;
      return
    end
    if $|path| < maxDepth$ **then**
      `DFS`(*neighbor, path, visited, blackTickets, attempts*);
    end
    remove *neighbor* from *visited*;
    remove last node from *path*;
  end
Initialize $visited \leftarrow \{posX\}$;
Initialize $bestPath \leftarrow []$;
Initialize $successfulPaths \leftarrow []$;
Initialize $neighbors \leftarrow graph.neighbors(posX)$;
foreach *n in neighbors* **do**
  Initialize $attempts_n \leftarrow$ empty map of node→count;
  `DFS`(*n, [posX, n], visited, blackTickets,* $attempts_n$);
  if $|successfulPaths| > 0$ **then**
    return longest path from $successfulPaths$;
  end
end
return $bestSafePath$;