Algorithms and Analysis - Task C and Task D

Student ID: S4006840, Natchanon Laoharawee

Task C Methodology - Algorithms considered:

Approach 1 - Breadth-first search only (maximise shortest-path)

The main idea of my breadth-first search only approach:

- For every cell, append the neighbours to the back of a queue.
- Dequeue from the front of the queue (order for neighbours of a cell depends on implementation).
- Visit the dequeued cell.
- Repeat until exit is found.
- (Modification for Task C)
- Save the exit coordinates to a set. (We will use this set to avoid finding the same paths).
    - If the exit coordinates are already in the set for the current entrance, then continue searching.
- Save the path found as a global key-value pair (Exit coordinates, path). Path is a list that contains the cells visited for the path (we can deduce the total distance from this list).
- If the exit coordinates has a better total distance where:

  $$Min(t_{dist}\ ,\ t'_{dist}),$$

  where t_dist is the total distance from an entrance to an exit.
    - Then we change the total distance for that (exit, path) key-value pair.
- Keep doing the above steps until we find all exits for a specific entrance.
- Repeat this with all entrances.

Brief analysis:

- Since the algorithm always picks a cell to visit that is closest to the starting node, the algorithm is guaranteed to find the shortest-path, but at the cost of searching unnecessarily. This is a brute-force strategy to consider all possible best-paths for an entrance to all exits and is my first thought to solve this problem.
- Exploring a cell is the basic operation in BFS, which would be at worst case O(N), where N is the number of cells in the maze. Though, there are many iterations of BFS, so the time complexity is closer to $O(E_{exit} * N * E_{entrances})$, where E is the number of exits or entrances in the maze (as we are trying all possible combinations of exits and entrances to maximise shortest-path).

Approach 2 - Hybrid approach, A* search and Dijkstra's algorithm-inspired search (Best-first search algorithm)

The main idea of my hybrid approach:

- Find all coordinates of exits using recursive back-tracking (or DFS) solver from a single entrance. Save them to a set.
- Begin A* search and Dijkstra's inspired search.
- Start at a random exit that has not been finalised.
- When not at an entrance:
- For the current cell, we will have to calculate the weights of neighbours.
- The weighting for a specific neighbour is given by the following equation:
    - Weight = distance from nearest exit + current path distance.

$$W_{neighbour} = d_{exit} + d_{path},$$

Where W_neighbour is the weight for the current neighbour considered,
d_exit is the distance from the nearest entrance to the neighbour (we will use Manhattan distance here)
d_path is the number of nodes in the path.

- Add the neighbours with their calculated weights into a priority queue.
- Dequeue from the priority queue and visit the cell returned from queue.
- Repeat until an entrance has been found.
- When an entrance has been found, this will be the optimum entrance for the given exit, save it to a finalised key-value pair (entrance coordinate, path).
- Repeat until all exits have found a shortest path.

Brief analysis:

- For very large values of N (number of cells in maze), this hybrid approach may be more efficient than the previous approach as it only considers one iteration of DFS (to find all exits), and then iterates through exits using a A* search-style algorithm that guarantees shortest-paths to an entrance. It may be less efficient when there are a large number of entrances to consider upon each calculation of the weights compared to the brute-force method I proposed with BFS, but should negate much of the unnecessary visits of a BFS algorithm running multiple times over on a particular entrance (that occurs until all exits are visited).

Task C Implementation:

- Did not get to complete implementation, encountered challenges when coding.

Task C Conclusion:

- The hybrid approach should be more efficient for very large mazes, with a few number of entrances, but the brute-force strategy may be more efficient for smaller mazes.

Task D

...