DATA STRUCTURES AND ALGORITHMS

PROJECT REPORT

Project Title: **The A* Algorithm**

Group Members:

- Muhammad Zain Yousuf
- Eesha Iftikhar Qazi
- Muhammad Zaeem Baig

Project Description:

The algorithm that we selected for our DSA Project is A* algorithm. The A* search algorithm is an Artificial Intelligence (AI) algorithm that is used for graph traversals and path finding. It is frequently used in computer science due to its completeness, optimality and accuracy. A* algorithm has a higher rate of precision because it utilizes heuristics to perform its search.

In our project A*, we will be demonstrating three application of A* which are as follows:

1. Maze Solving
2. 8-Puzzle solving
3. String combination

In maze solving, we will be given a grid of specific dimension along with some obstacles in it. We need to implement A-star to find the shortest and the smartest path from the starting point to the goal. In each iteration, it calculates the value of f(n) by calculating the cost of current node from starting node g(n) and the estimated distance from the current node to the goal node h(n). It selects the node with the least value of f(n), and continue to do this till our goal state is not reached.

In 8-puzzle, it first compares the misplaced tiles in the puzzle, and calculates h(n). Then, it generates sub-states or children, and again calculates h(n) by comparing the misplaced

tiles. In this way, it selects the child with the least h(n) value, and returns the solved 8-puzzle as the goal state is provided.

In string combination, we will be given an unarranged string, a starting state, and we need to convert the string into the arranged form that is our goal state using different combinations. It generates different combinations and calculates f(n). Similarly, selects the combination with least value of f(n), and reach our goal combination in this manner.

Implementation of DSA Techniques:

- Lists
- Tuples
- Strings
- Node Classes and Objects
- Nested Lists
- Priority Queue

Outputs and Results:

Following are the outputs and results of A* applications:

String Combination:

Enter starting string: ytpnho

Enter goal string: python

Starting...

0) ytpnho

1) yptnho

2) pytnho

3) pythno

4) python

Maze Solving:

For 5x5 grid maze,

['_', '_', '_', '|', '_']

['_', '_', '_', '|', '_']

['_', '_', '_', '_', '_']

['_', '_', '_', '|', '_']

['_', '_', '_', '|', '_']

Start = (0, 0)

Goal = (3, 4)

Path = [(0, 0), (1, 1), (2, 2), (2, 3), (3, 4)]

8-Puzzle:

For 3x3 puzzle dimension,

Enter the start state matrix

1 2 3

_ 4 6

7 5 8

Enter the goal state matrix

1 2 3

4 5 6

7 8 _

|

|

\/

1 2 3
_ 4 6
7 5 8

|
|
\/

1 2 3
4 _ 6
7 5 8

|
|
\/

1 2 3
4 5 6
7 _ 8

|
|
\/

1 2 3
4 5 6
7 8 _

Challenges Addressed:

- For our project, first we needed to calculate values for g(n) and h(n) to get f(n). It was hurdle for us to associate lengths in such a manner that would give us better approximation of h(n) and g(n).
- In maze solving and 8-puzzle, we needed to store the maze and puzzle in such a way that would easily be processed on, and that would easily generate f(n). We used lists and nested lists for that purpose.
- Once the heuristics are calculated, it was a difficult task to store the node and its heuristics value in any array that would give us the node with least value of f(n) in O(1) time. We implemented Priority Queue to tackle that challenge in String Combination.
- To store the node and its heuristics value in an array would be a difficult task if both are not associated with each other. Therefore, we used Object-Oriented Programming to create node classes in order to store the node data easily.
- A single A-Star function was insufficient to cope up with the features of the applications. Therefore, we created extra supporting and helping functions to make sure that the function processes data properly.

Completeness:

All the three applications of A* gives correct and accurate outputs as we expected from them. None of the programs crashed or gave errors. Furthermore, codes have been commented in order to provide feasibility to the users. Complete experimental results and detail time complexity analysis have been provided along with the complete codes. Each application uses heuristics to calculate the distance, and all the basic A* features have been applied and implemented in the applications.