Report

Implementation of A Star Algorithm.

By

Team A Star

MUHAMMAD ZAIN YUSUF

EESHA IFTIKHAR QAZI

ZAEEM BAIG

Habib University

CS 102

SECTION 05

CONTENTS

ABSTRACT

A-star algorithm is one the Ai's search algorithm. It uses a general formula "f(x)=g(x)+h(x)" where h(x) is heuristic part which estimates the shortest path and g(x) represents path from initial to current position. In this report 3 implementations of this algorithm are discussed with respective output, experimental analysis and calculation of time complexity. first application involves A-star algorithm in maze solving, where maze is an input and shortest path from starting node to goal node is the output. Second implementation uses A-star maze to solve the puzzle by changing input matrix to goal matrix. Last implementation uses this search technique to change input string in goal string.

ALGORITHM INTRODUCTION

Reaching a destination via the shortest route is a daily activity we all do. A-star (also referred to as A*) is one of the most successful search algorithms to find the shortest path between nodes or graphs. It is an informed search algorithm, as it uses information about path cost and also uses heuristics to find the solution.

A* Search algorithm is one of the best and popular techniques used in path-finding and graph traversals. Informally speaking, A* Search algorithms, unlike other traversal techniques, it has “brains”. What it means is that it is really a smart algorithm that separates it from the other conventional algorithms. This fact is cleared in detail in the below sections.

And it is also worth mentioning that many games and web-based maps use this algorithm to find the shortest path very efficiently (approximation).

ALGORITHM 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 pathfinding. 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.

A* achieve **optimality** and **completeness**, two valuable property of search algorithms.

When a search algorithm has the property of **optimality**, it means it is **guaranteed** to find the **best possible solution**. When a search algorithm has the property of **completeness**, it means that if a solution to a given problem **exists**, the algorithm is **guaranteed** to find it.

Now to understand how A* works, first we need to understand a few terminologies:

- **Node** (also called **State**) — All potential position or stops with a unique identification
- **Transition** — The act of moving between states or nodes.
- **Starting Node —** Where to start searching

- **Goal Node —** The target to stop searching.
- **Search Space** — A collection of nodes, like all board positions of a board game
- **Cost** — Numerical value (say distance, time, or financial expense) for the path from a node to another node.
- **g(n)** — this represents the **exact cost** of the path from the **starting node** to any node **n**
- **h(n)** — this represents the heuristic **estimated cost** from node **n** to the goal node.
- **f(n)** — lowest cost in the neighboring node n

EXPLANATION

Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible. Here A* Search Algorithm comes to the rescue.

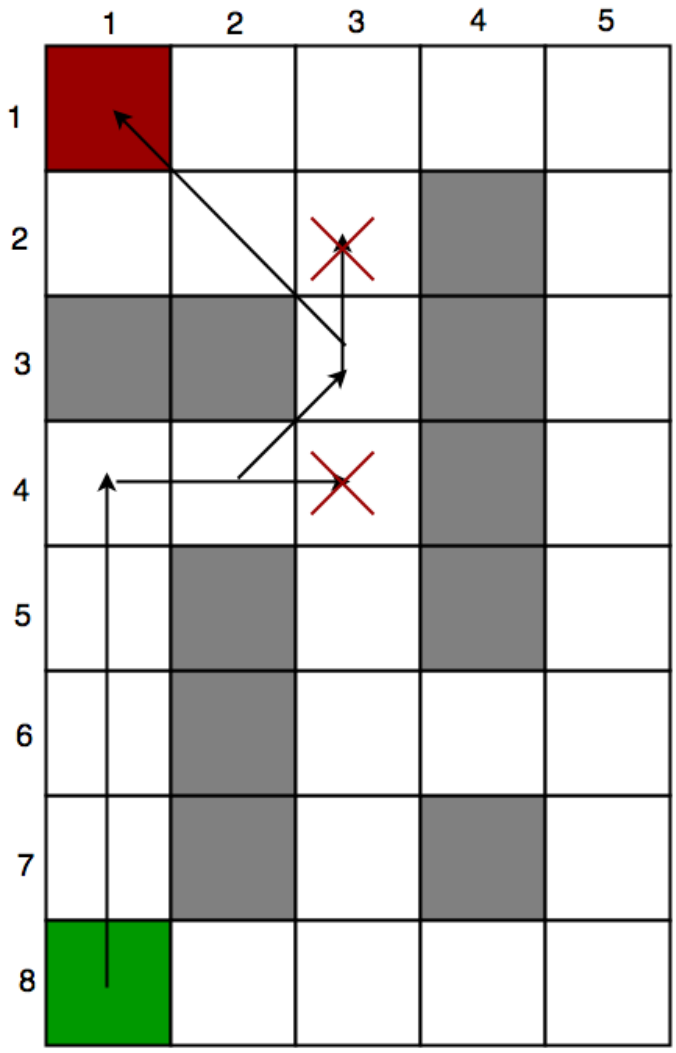
What A* Search Algorithm does is that at each step it picks the node according to a value-'**f**' which is a parameter equal to the sum of two other parameters – '**g**' and '**h**'. At each step it picks the node/cell having the lowest '**f**', and process that node/cell.

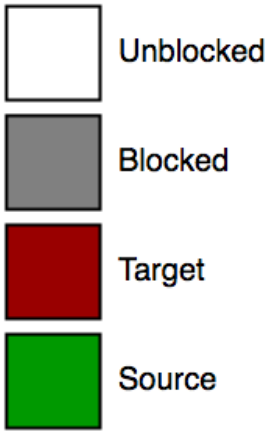
We define '**g**' and '**h**' as simply as possible below

g = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.

h = the estimated movement cost to move from that given square on the grid to the final destination. This is often referred to as the heuristic, which is nothing but a kind of smart guess. We really don't know the actual distance until we find the path, because all sorts of things can be in the way (walls, water, etc.).

So suppose as in the below figure if we want to reach the target cell from the source cell, then the A* Search algorithm would follow the path as shown below. Note that the below figure is made by considering Euclidean Distance as a heuristics





A* Search Algorithm makes the most intelligent choice at each step. Hence you can see that algorithm goes from (4,2) to (3,3) and not (4,3) (shown by cross).

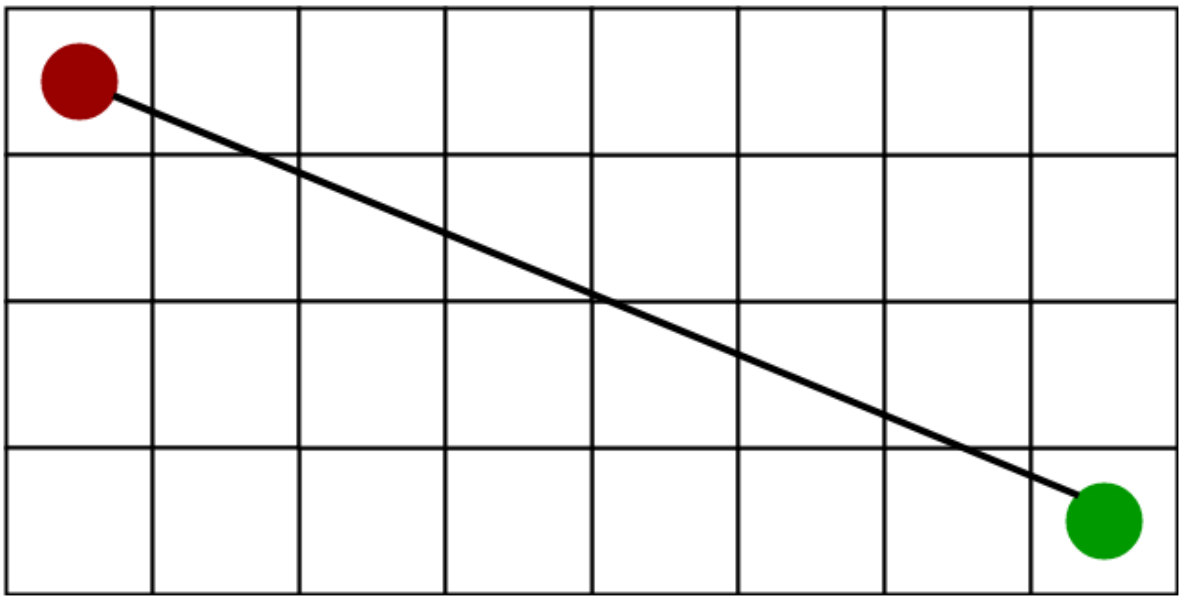
Similarly the algorithm goes from (3,3) to (2,2) and not (2,3) (shown by cross).

EUCLIDEAN DISTANCE

As it is clear from its name, it is nothing but the distance between the current cell and the goal cell using the distance formula

h= $\sqrt{}$(x2 − x1) 2 + (y2 − y1) 2 t

The Euclidean Distance Heuristics is shown by the below figure (assume red spot as source cell and green spot as target cell).



PROJECT IMPLEMENTATION

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

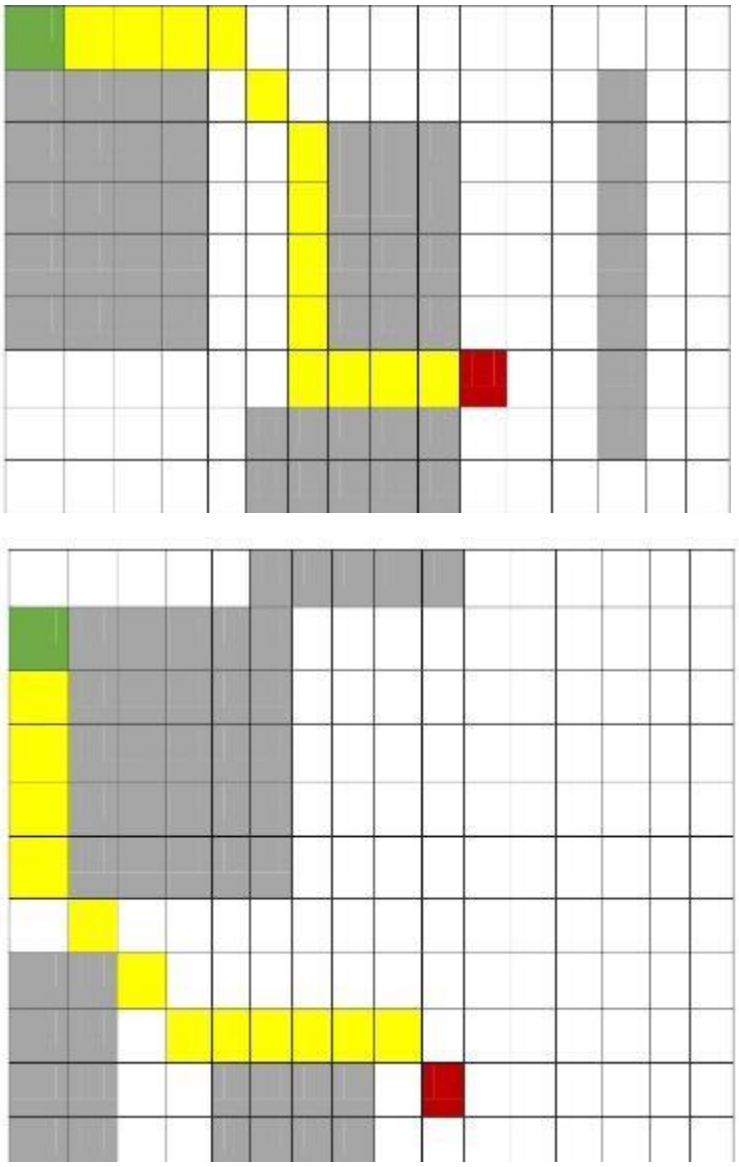
2. 8-Puzzle solving

3. String combination

Maze Solving

In maze solving, we took a grid of specific dimensions along with some obstacles in it. We then implemented 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 the 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.

A-star algorithm has vast applications and is used in a variety of fields and programs. From robotics, machine learning and AI based applications to games, maps and graphs, programmers highly rely upon A-star algorithm. Therefore, in our project too, we will be implementing A-star for path-finding. As it is stated above, A-star is widely used in Game Development, and in our project, we will be giving a grid of specific dimensions 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. Similar to what is implemented in games, if you want to go from one place to another or find a specific character in the game, you are directed towards it, and is given the path to the desired location by the game engine itself. On the backhand, it is actually A-star that is implemented in the game. Therefore, we will be solving this kind of problem in our project. The following are a few examples to give insights of our project.



Here, in this demonstration, yellow denotes the path, green starting point and red destination. We can see how smartly the algorithm chooses the path to the goal, and cuts between the obstacles marked by the gray color. That is why it is said that this is the algorithm with “brains” in it.

Maze Solving Output:

For 5x5 grid maze,

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

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

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

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

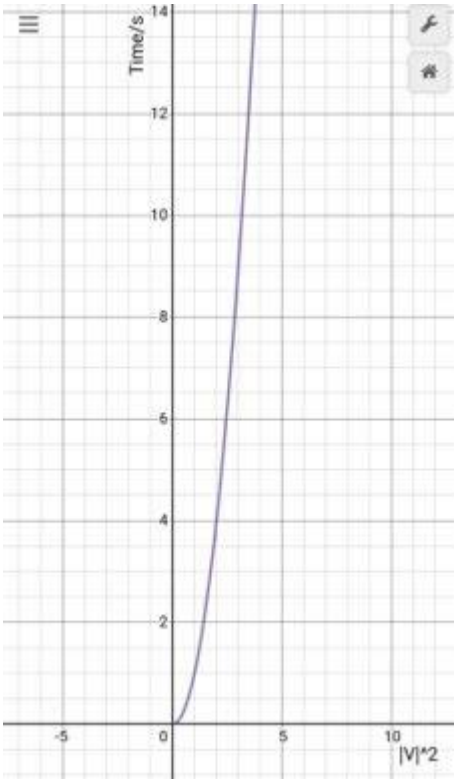
['_', '_', '_', '|', '_']

Start = (0, 0)

Goal = (3, 4)

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

TIME COMPLEXITY OF MAZE SOLVING ALGORITHM.



In this code, while loop runs over |V| times in the worst case, and we have a “for” loop inside it which also runs for |V| times. Therefore, it can be calculated that the time complexity is polynomial (quadratic), and thus has a parabolic graph as shown below.

f(|V|) = |V|(|V|+1) / 2

f(|V|) = |V|^2

TIME EXECUTION OF MAZE SOLVING ALGORITHM.

8x8	0.00025
9x9	0.00027
10x10	0.00031

APPLICATION OF A*

String Combination.

In this application of A*, we took an unarranged string, a starting state, and we needed to convert the string into the arranged form that is our goal state using different combinations. We did this by generating random combinations of our starting goal, and we may reach to our goal state, however, that is really inefficient, and we may never reach our goal state and end up in an infinite loop. Here, A* again gives the optimal path to reach to our goal state using the same technique. First, it generates children of state by rotating the string in random manner and generates $f(n)$ by calculating the distance of each letter as follows: - Calculates $g(n)$ each time when we move away from the starting letter. In this case, it keeps adding the index of the letter. - Calculates $h(n)$ by comparing the index of the letter in the starting state to the goal state - Adds $g(n)$ and $h(n)$ to generate $f(n)$ of each child. Secondly, it stores the child and its $f(n)$ value in in a Priority Queue, and when we are done with generating child, we simply get the child state from Priority Queue which stores the value in an arranged ascending order. In this manner, we can get the child of minimum cost value of $f(n)$ and reach our goal in the shortest path possible.

String Combination Output:

Enter starting string: ytpnho
Enter goal string: python

Starting...

0) ytpnho

1) yptnho

2) pytnho

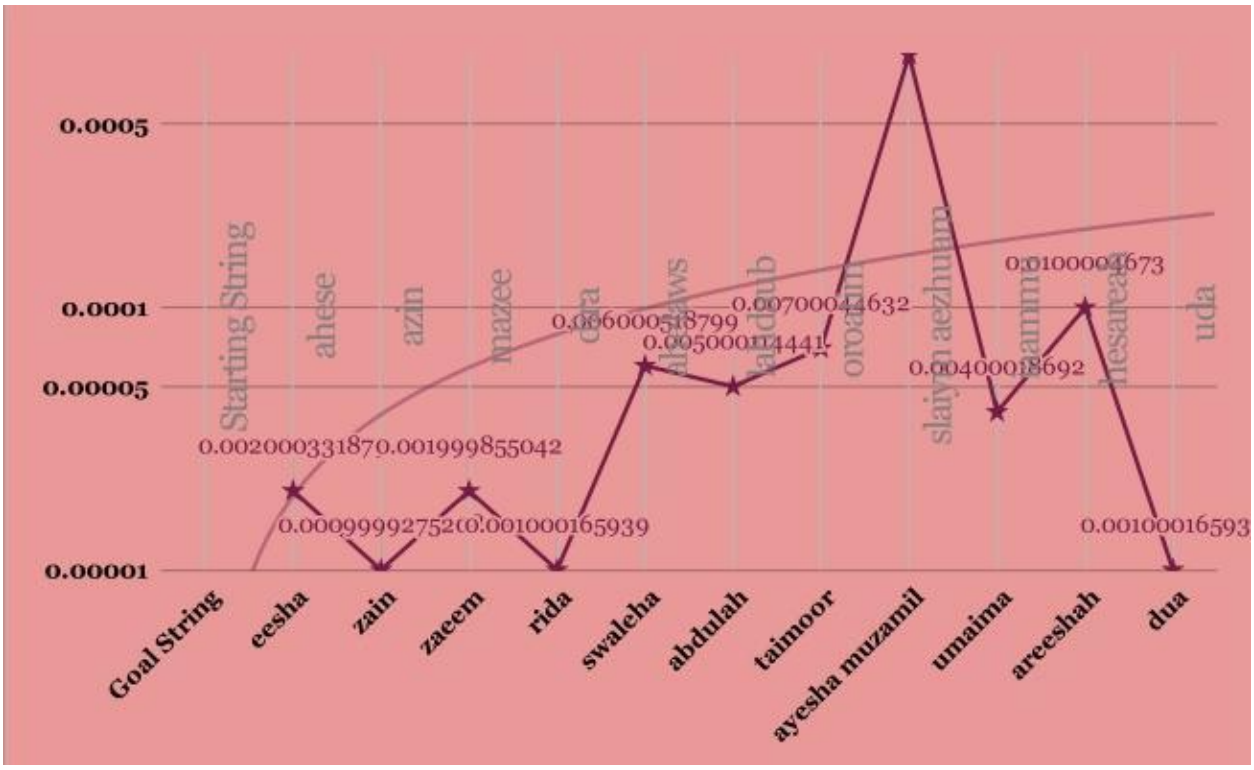
3) pythno

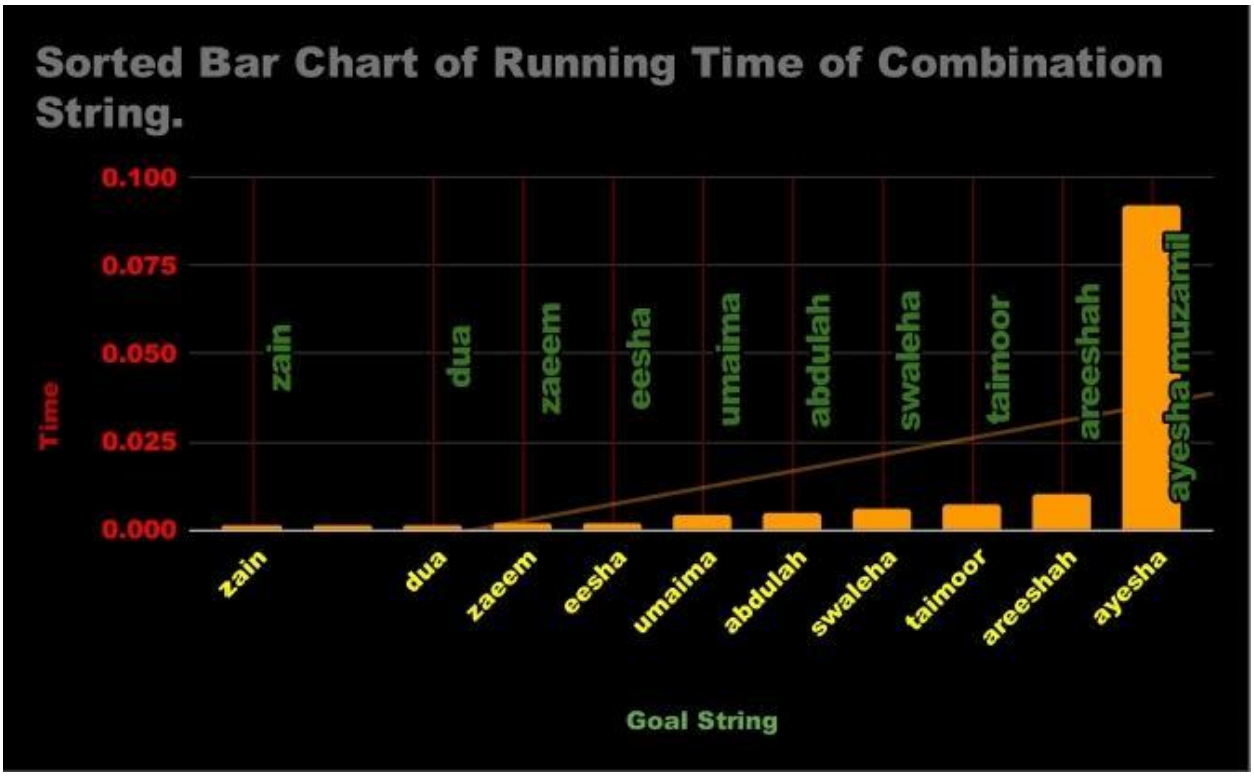
4) python

Time Execution

No. of times.	Starting String	Goal String	Time
1	ahese	eesha	0.002000331879
2	azin	zain	0.000999927520
3	mazee	zaeem	0.001999855042
4	dira	rida	0.001000165939
5	ahelaws	swaleha	0.006000518799
6	lahdaub	abdulah	0.005000114441
7	oroatim	taimoor	0.00700044632
8	slaiym aezhuam	ayesha muzamil	0.09200525284
9	iaammu	umaima	0.00400018692
10	hesareah	areeshah	0.0100004673
11	uda	dua	0.001000165939

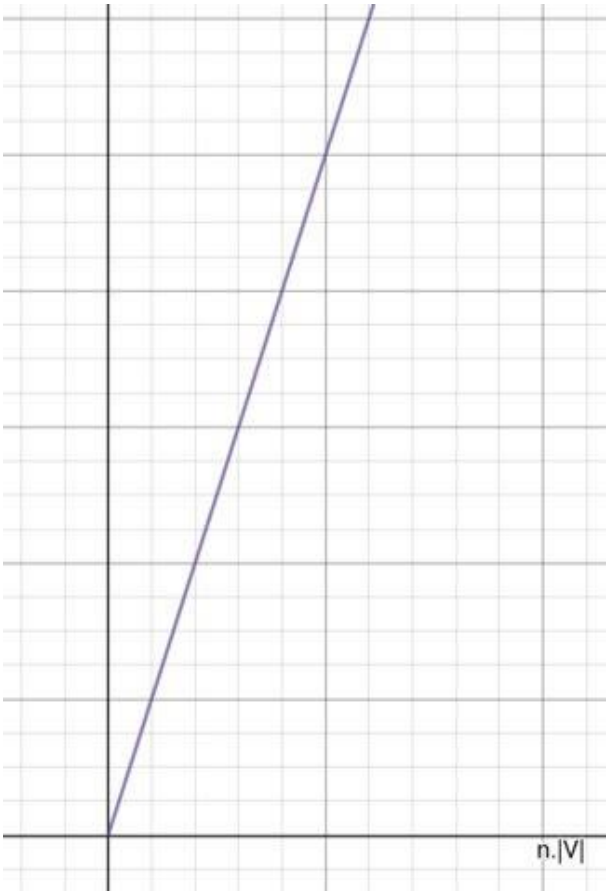
We ran ten different outputs of different lengths and levels of complexities and achieved the following result.


0.0005
0.0001
0.00005
0.00001
Starting String
ahese
azin
mazee
0.0002000331870.001999855042
0.000999927520.001000165939
0.00700044632
0.0100004673
0.00400018692
0.00100016593
Goal String
eesha
zain
zaeem
rida
swaleha
abdulah
taimoor
ayesha muzamil
umaima
areeshah
dua


Sorted Bar Chart of Running Time of Combination String.
0.100
0.075
0.050
0.025
0.000
Time
zain
dua
zaeem
eesha
umaima
abdulah
swaleha
taimoor
areeshah
ayesha muzamil
Goal String
ayesha

Time Complexity

This implementation has a worst-case time complexity of O(n.|V|) where ‘V’ represent vertices and ‘n’ is the length of the input string.



ANOTHER APPLICATION OF A*

8 Puzzle Solve

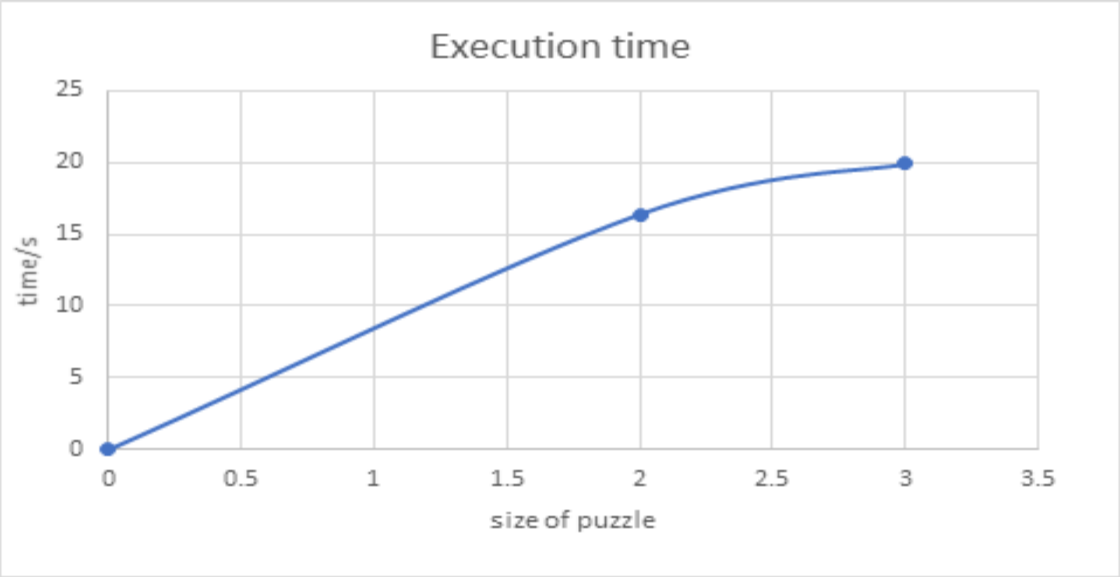
In this project, we solved the 8-puzzle problem. An 8-puzzle problem is a popular puzzle game that consists of 3 rows and 3 columns. We were given 8 known tiles and one empty tile which can be replaced or moved between the tiles in order to obtain the goal state. We implemented A* to reach our goal state in an optimal manner. Here, the tiles will be the Nodes, and the start and goal configurations are the States. Each time when we move the empty node to reach our goal state, the initial state will get updated, and we need to keep track of each node state. Now, A* comes into play here. It generates new states, processes the data of the state, and keeps into the recorded list or visited list so that we may not end up processing one state multiple times. So, how does the A* algorithm finds the optimal path to the goal state? Again, it used the above-stated methods to do so. The state with least value of $f(n)$ is selected until we reached the destination. It generated the value of $f(n)$ as follows:

- - $f(n) = g(n) + h(n)$
- - Where $g(n)$ is the number of nodes traversed from the start node to the current node
- - $h(n)$ is the number of misplaced tiles by comparing the current state and the goal state.

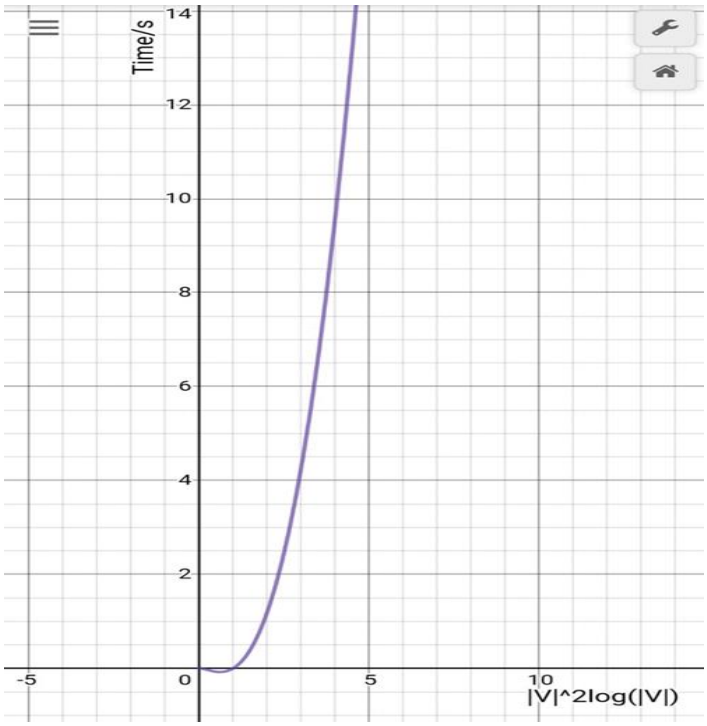
- - It generates the value of $h(n)$ by comparing the initial and the goal states, and the number of misplaced tiles represent the value of $h(n)$ and $g(n)$ is the cost of moving the empty tile from the starting node.
- - It will create the children of the current state by moving the empty in four directions, UP, DOWN, LEFT and RIGHT, and calculates the value of $f(n)$ as mentioned above.
- - It keeps generating sub-state of each parent state, and calculates $f(n)$ until we reach our goal state. In this manner, we can find the optimal path to solve the 8-puzzle.

Time Complexity And Running Time:

0	0.00015
2	16.4
3	19.9



This implementation has time complexity of $O(|V|^2 \log|V||)$ where 'n' is length of input while 'V' represents vertices.



Implementation of DSA Techniques:

Lists

Tuples

Strings

Node Classes and Objects

Nested Lists

Priority Queue

Challenges Addressed:

- For our project, first, we needed to calculate values for g(n) and h(n) to get f(n). It was a hurdle for us to associate lengths in such a manner that would give us a 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 an array that would give us the node with the 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 that the function processes data properly.

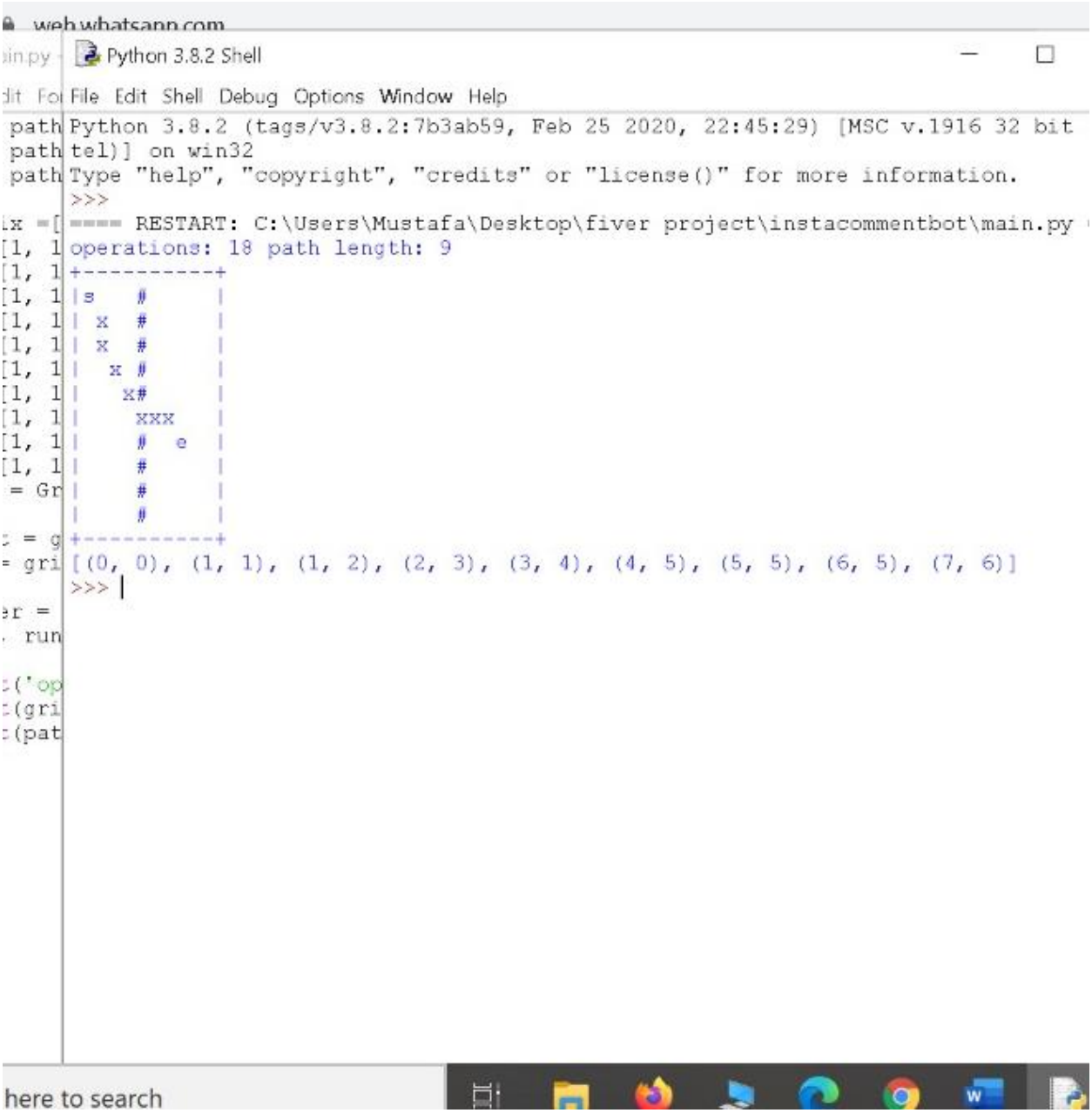
VALIDATION

For validating the outcome of our project and to confirm the correct implementation of A* algorithm, our team took the aid of the Python library and used an in-built library function to confirm the output result. We took the library function from the following website: https://pypi.org/project/pathfinding.

```
from pathfinding.core.diagonal_movement import DiagonalMovement
from pathfinding.core.grid import Grid
from pathfinding.finder.a_star import AStarFinder
matrix = [[1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
      [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
      [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
      [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
      [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
      [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
      [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
      [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
      [1, 1, 1, 1, 0, 1, 1, 1, 1, 1]]
grid = Grid(matrix=matrix)


start = grid.node(0, 0)
```

```
end = grid.node(3, 3)


finder = AStarFinder(diagonal_movement=DiagonalMovement.always)
path, runs = finder.find_path(start, end, grid)


print('operations:', runs, 'path length:', len(path))
print(grid.grid_str(path=path, start=start, end=end))
print(path)
```

It Generated following results:



It's not exactly the same as the output of our algorithm but that's precisely because the inbuilt library function is using the DiagonalMovement function.

COMPLETENESS AND OUTCOME

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.

CONCLUSION

One of the most prominent reasons why we chose this algorithm is that it is an Artificial Intelligence algorithm. It is commonly known as the algorithm with brains. Thus, it gave us some exposure of AI, and how it works since it is an emerging field, and most of us want to pursue this in the future. Furthermore, we were introduced to some new concepts of data structures and algorithms such as how to create nodes and the heuristics part in the algorithm. Therefore, we also had more exposure towards learning some background of AI algorithms and heuristic, and some handful knowledge of how to create classes and nodes. Moreover, the other dominant outcome of this project was to be introduced to GitHub, an online platform for developers where we collaborated with our colleagues to write codes and other documentation. Therefore, it was really beneficial for us as we can corporate our skills in future as we are more acquainted with GitHub now. Other than that, we also learned how to work in a team more productively, and it helped us hone our soft skills which are key aspects for becoming successful in today's world.

References

A* Search Algorithm. (2018, September 07). Retrieved June 17, 2020, from https://www.geeksforgeeks.org/a-search-algorithm/

A* search algorithm. (2020, June 04). Retrieved June 17, 2020, from https://en.wikipedia.org/wiki/A*_search_algorithm

Payne, T. (2013, December 21). *Let's Learn Python #20 - A* Algorithm* [video]. YouTube. https://www.youtube.com/watch?v=ob4faIum4kQ&t=194s

Swift, N. (2020, May 29). Easy A* (star) Pathfinding. Retrieved June 17, 2020, from https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2

Sonawane, A. (2020, January 08). Solving 8-Puzzle using A* Algorithm. Retrieved June 17, 2020, from https://blog.goodaudience.com/solving-8-puzzle-using-a-algorithm-7b509c331288