# CMPT 310 Assignment 1 Documentation

This documentation will include a summary of the code in the A1.py file. It also includes the test results from the test data given for the assignment. Lastly, I also modified the test data a bit (changing numbers on the board, and changing the starting node and the ending node) to see the reaction of my algorithm. Those will be included at the bottom of the file.

## Code Summary

For this assignment, I used an external python library – numpy. I mostly used it for array manipulation, and for the function numpy.diff() which calculates the difference between elements in a 2d array.
I also used a lot of global variables in this assignment for convenience.

## Classes

node – an object I created, used to track information about an element of the board/matrix. This object is only used for the best-first search.

- x: which row is it on
- y: which item on the row
- value: the heuristic value from this node to the end goal
- parent: which node expanded into this node

a_star_node – an upgrade to node, added an extra attribute called distance. Used for the A* search.

- x: which row is it on
- y: which item on the row
- value: the heuristic value from this node to the end goal plus the distance from the starting node
- parent: which node expanded into this node
- distance: the real path distance from the starting node to this node

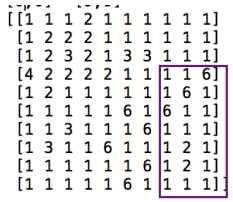
## Global Variables

n – the n value form user input data, the dimension for the matrix
gx, gy – the goal coordinates from the user input data
best_first_search_nodes – a list of nodes that needs to be expanded.
best_first_searched_nodes – a list of searched nodes
a_star_search_nodes, a_star_searched_nodes – same as above, but used for A* search
matrix – the board/environment

## Functions

**readFile** – reads the file and assign variables accordingly
**nodeInList** – a function to see if a node exist in a list
**popSearchNode** – pop off an element from the best-first search nodes
**findNode** – return the detailed information of a node based its coordinates
**createSubMatrix** – creates a submatrix, used to help calculate the heuristic

**findHeuristic** – My heuristic functions takes a submatrix (a rectangle that include both the current node and the end goal) from the board. Takes the average elevation (using numpy.diff()), and add the Manhattan distance.
Eg. The submatrix form (3,7) would be the matrix in the box
And the heuristic value would be 10.4285714286




**matrixDifference** – calculate the absolute difference between 2 elements, use to prevent going over cliffs and in other calculations

**expandBestFirstFringe** – from a given node, expand its neighbours. The function first mark the given node as searched, then it would look at it's neighbours. If the neighbour's elevation isn't too high, and the neighbour hasn't been searched before, it would append the neighbour node into the best_first_search_nodes list. The heuristic value for the neighbour node would be calculated using **findHeuristic** from the neighbour node to the end goal.

**popAStarSearchNode** – similar to **popSearchNode**, it pops a node off the A* search list

**expandAStarFringe** – same as **expandBestFirstFringe**, but the heuristic value also takes the distance to the start into account. The distance to the start is calculated by taking its parent's distance value, adding the difference in elevation, and 1 for mandatory cost of moving.

## Best-First Search

My best first search function first makes an expansion on the starting node. Storing its neighbours in the best_first_search_nodes list. It then takes the minimum of the best_first_search_nodes list and expand on the minimum cost node. As it expands, it tracks which node it expanded in the best_first_searched_nodes list and it store its neighbours in the best_first_search_nodes list. The best_first_search_nodes list includes the coordinates of the node, it's heuristic value (from the function) and it's parent. The algorithm would avoid expanding any node that has already been expanded. It keeps expanding until the goal node appears in the best_first_searched_nodes list. Then it just prints the value and the path.

## A-Star Search

A very similar idea to the best-first search, 90% of the algorithm is the same. The only difference is that in the A* search, I also took the distance from the starting node into account. So the heuristic values from the nodes in the a_star_search_nodes is the sum of the value I got from my heuristic function plus the distance from the starting node. Then the rest of the algorithm is the same, searching through the a_star_search_nodes list of the minimum value to expand.

Test Data Sample

Input Data:

10

[0,0]  [9,9]

[[1 1 1 2 1 1 1 1 1 1]
 [1 2 2 2 1 1 1 1 1 1]
 [1 2 3 2 1 3 3 1 1 1]
 [4 2 2 2 2 1 1 1 1 6]
 [1 2 1 1 1 1 1 1 6 1]
 [1 1 1 1 1 6 1 6 1 1]
 [1 1 3 1 1 1 6 1 1 1]
 [1 3 1 1 6 1 1 1 2 1]
 [1 1 1 1 1 1 6 1 2 1]
 [1 1 1 1 1 6 1 1 1 1]]

Output:

```
The Best First Search has successfully found a path.
['0,0', '0,1', '0,2', '0,3', '1,3', '2,3', '3,3', '3,4', '3,5', '4,5', '4,4', '5,4', '6,4', '6,5',
'7,5', '7,6', '7,7', '8,7', '9,7', '9,8', '9,9']
It used a total of 20 steps.
The sum cost of the steps (including elevation difference) is 22
The following nodes were searched using the best first search method.
['0,0', '0,1', '0,2', '0,3', '1,3', '2,3', '3,3', '3,4', '3,5', '3,6', '3,7', '4,7', '3,8', '2,8',
'2,9', '1,9', '0,9', '1,8', '4,6', '5,6', '2,7', '0,8', '1,7', '4,5', '0,7', '4,4', '5,4', '6,4',
'6,5', '7,5', '7,6', '7,7', '8,7', '9,7', '9,8', '9,9', ]
The for a total of 36 nodes.
The A* Search has successfully found a path.
['0,0', '1,0', '2,0', '2,1', '3,1', '4,1', '5,1', '5,2', '5,3', '6,3', '6,4', '6,5', '7,5', '7,6',
'7,7', '8,7', '9,7', '9,8', '9,9']
It used a total of 18 steps.
The sum cost of the steps (including elevation difference) is 20
The following nodes were searched using the A* search method.
['0,0', '0,1', '1,0', '0,2', '2,0', '1,1', '1,1', '0,3', '1,2', '2,1', '2,1', '3,1', '1,3', '1,3',
'4,1', '3,2', '2,3', '3,3', '3,4', '0,4', '0,5', '0,6', '0,7', '0,8', '0,9', '1,9', '2,9', '1,8',
'1,7', '2,8', '3,8', '2,8', '2,7', '3,7', '4,7', '3,7', '1,6', '5,1', '6,1', '2,2', '2,2', '5,2',
'4,2', '1,5', '1,4', '1,4', '5,3', '6,3', '7,3', '8,3', '9,3', '9,4', '8,4', '8,5', '6,4', '6,5',
'7,5', '7,6', '7,7', '8,7', '9,7', '9,8', '9,9', ]
The for a total of 63 nodes.
```

Input Data:

10

[0,0]  [9,4]

[[1 1 1 2 1 1 1 1 1 1]
 [1 2 2 2 1 1 1 1 1 1]
 [1 2 3 2 1 3 3 1 1 1]
 [4 2 2 2 2 1 1 1 1 6]
 [1 2 1 1 1 1 1 1 6 1]
 [1 1 1 1 1 6 1 6 1 1]
 [1 1 3 1 1 1 6 1 1 1]
 [1 3 1 1 6 1 1 1 2 1]
 [1 1 1 1 1 1 6 1 2 1]
 [1 1 1 1 1 6 1 1 1 1]]

Output:

```
The Best First Search has successfully found a path.
['0,0', '0,1', '0,2', '0,3', '1,3', '2,3', '3,3', '3,4', '4,4', '5,4', '6,4', '6,3', '7,3', '8,3',
'8,4', '9,4']
It used a total of 15 steps.
The sum cost of the steps (including elevation difference) is 17
The following nodes were searched using the best first search method.
['0,0', '0,1', '0,2', '0,3', '1,3', '2,3', '3,3', '3,4', '4,4', '5,4', '6,4', '6,3', '7,3', '8,3',
'8,4', '9,4', ]
The for a total of 16 nodes.
The A* Search has successfully found a path.
['0,0', '0,1', '0,2', '0,3', '1,3', '2,3', '3,3', '4,3', '5,3', '6,3', '7,3', '8,3', '8,4', '9,4']
It used a total of 13 steps.
The sum cost of the steps (including elevation difference) is 15
The following nodes were searched using the A* search method.
['0,0', '0,1', '0,2', '1,0', '2,0', '0,3', '1,2', '1,3', '1,3', '1,1', '1,1', '2,3', '3,3', '2,1',
'3,1', '3,2', '2,1', '4,1', '3,4', '2,2', '4,2', '4,2', '0,4', '4,3', '1,4', '1,4', '5,2', '5,2',
'5,1', '2,4', '5,3', '4,4', '6,1', '6,3', '7,3', '8,3', '8,4', '9,3', '9,4', ]
The for a total of 39 nodes.
```

Input Data:

10

[0,0] [9,9]

[[1 1 1 2 1 1 1 1 1 1]
[1 2 2 2 1 1 1 1 1 1]
[1 2 3 2 1 3 3 1 1 1]
[4 2 2 2 2 1 1 1 1 6]
[1 2 1 1 1 1 1 1 6 1]
[1 1 1 1 1 6 1 6 1 1]
[1 1 3 1 1 1 6 1 1 1]
[1 3 1 1 6 1 6 1 2 1]
[1 1 1 1 1 1 6 1 2 1]
[1 1 1 1 1 6 1 1 1 1]]

Output:

```
No path found using best first search.
The following nodes were searched using the best first search method.
['0,0', '0,1', '0,2', '0,3', '1,3', '2,3', '3,3', '3,4', '3,5', '3,6', '3,7', '4,7', '3,8', '2,8',
'2,9', '1,9', '0,9', '4,6', '5,6', '1,8', '2,7', '0,8', '1,7', '4,5', '0,7', '1,6', '4,4', '5,4',
'6,4', '6,5', '7,5', '8,5', '8,4', '9,4', '9,3', '9,2', '9,1', '8,3', '9,0', '8,2', '8,1', '7,3',
'7,2', '8,0', '6,3', '5,3', '7,0', '5,2', '4,3', '6,0', '6,1', '0,6', '7,1', '6,2', '5,1', '2,6',
'2,5', '4,2', '1,5', '5,0', '0,5', '3,2', '1,4', '2,4', '4,0', '3,1', '4,1', '0,4', '1,2', '2,1',
'2,2', '1,1', '1,0', '2,0', '3,0', ]
The for a total of 75 nodes.
No path found using best first search.
The following nodes were searched using the A* search method.
['0,0', '0,1', '1,0', '0,2', '2,0', '1,1', '1,1', '0,3', '1,2', '2,1', '2,1', '3,1', '1,3', '1,3',
'4,1', '3,2', '2,3', '3,3', '3,4', '0,4', '0,5', '0,6', '0,7', '0,8', '0,9', '1,9', '2,9', '1,8',
'1,7', '2,8', '3,8', '2,8', '2,7', '3,7', '4,7', '3,7', '1,6', '5,1', '6,1', '2,2', '2,2', '5,2',
'4,2', '1,5', '1,4', '1,4', '5,3', '6,3', '7,3', '8,3', '9,3', '9,4', '8,4', '8,5', '6,4', '6,5',
'7,5', '4,3', '9,2', '2,4', '2,4', '3,5', '3,6', '4,6', '5,6', '5,4', '4,5', '3,0', '4,4', '1,8',
'8,2', '6,0', '7,0', '8,0', '9,0', '9,1', '8,1', '7,1', '6,2', '6,2', '7,2', '5,0', '4,0', '2,6',
'2,6', '2,5', '2,5', '3,0', '4,4', '8,1', '8,2', '7,2', ]
The for a total of 92 nodes.
```

Input data:

10

[9,9] [2,9]

```
[[1 1 1 2 1 1 1 1 1 1]
 [1 2 2 2 1 1 1 1 1 1]
 [1 2 3 2 1 3 3 1 1 1]
 [4 2 2 2 2 1 1 1 1 6]
 [1 2 1 1 1 1 1 1 6 1]
 [1 1 1 1 1 6 1 6 1 1]
 [1 1 3 1 1 1 6 1 1 1]
 [1 3 1 1 6 1 1 1 2 1]
 [1 1 1 1 1 1 6 1 2 1]
 [1 1 1 1 1 6 1 1 1 1]]
```

Output:

```
The Best First Search has successfully found a path.
['9,9', '8,9', '7,9', '6,9', '6,8', '6,7', '7,7', '7,6', '7,5', '6,5', '6,4', '5,4', '4,4', '4,5',
'4,6', '4,7', '3,7', '2,7', '2,8', '2,9']
It used a total of 19 steps.
The sum cost of the steps (including elevation difference) is 19
The following nodes were searched using the best first search method.
['9,9', '8,9', '7,9', '6,9', '5,9', '6,8', '9,8', '7,8', '8,8', '5,8', '6,7', '7,7', '8,7', '4,9',
'9,7', '7,6', '7,5', '9,6', '6,5', '6,4', '5,4', '4,4', '4,5', '4,6', '5,6', '4,7', '3,7', '2,7',
'2,8', '2,9', ]
The for a total of 30 nodes.
The A* Search has successfully found a path.
['9,9', '9,8', '9,7', '8,7', '7,7', '7,6', '7,5', '6,5', '6,4', '5,4', '4,4', '4,5', '4,6', '4,7',
'3,7', '2,7', '2,8', '2,9']
It used a total of 17 steps.
The sum cost of the steps (including elevation difference) is 17
The following nodes were searched using the A* search method.
['9,9', '8,9', '7,9', '6,9', '9,8', '5,9', '8,8', '7,8', '6,8', '9,7', '8,7', '7,7', '5,8', '5,8',
'6,7', '9,6', '4,9', '7,6', '7,5', '6,5', '8,5', '6,4', '5,4', '8,4', '4,4', '6,3', '4,5', '5,3',
'4,3', '8,3', '9,4', '7,3', '4,6', '5,6', '5,2', '4,2', '8,2', '4,7', '3,7', '2,7', '2,8', '2,9', ]
The for a total of 42 nodes.
```