Assignment 1
Analysis and Report

Student ID : 28993373
Bhanuka Manesha Samarasekara Vitharana Gamage
bsam0002@student.monash.edu
School of Information Technology

August 24, 2019

1 Proof of the Heuristic Function

The *input1.txt* example is shown below :

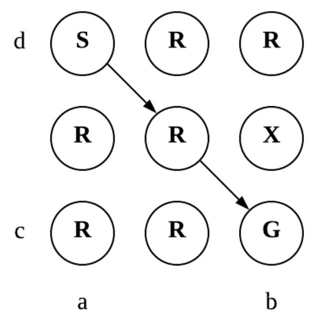

Figure 1: Minimum Heuristic from Start to Goal for *input1.txt*

Using the above example, we will prove that the heuristic is valid. The heuristic used in this case is the maximum between the difference from the current state to the goal state. The derivation of $h(n)$ is stated below:

$$dx = |b - a| \tag{1}$$

$$dy = |d - c| \tag{2}$$

$$h(n) = max(dx, dy) \tag{3}$$

Below is the python implementation for the heuristic:

```python
  def heuristic(self,x,y):
    '''
    method used to calculate the heuristic value given the
    current x and y coordinates
    @param x: current x coordinate
    @param y: current y coordinate
    @return: returns the heuristic value
    '''
    # Calculate the difference in both x and y directions
    dx = abs(x - self.GOAL_COORD[0])
    dy = abs(y - self.GOAL_COORD[1])

    # Returns the max of either the x or y direction
    return max(dx,dy)

```

When determining the heuristic we assume that there are no ridges in the map. Therefore the rules are relaxed compared to the actual rules of the environment. In order for the heuristic to be valid, it needs to be admissible and monotonic. Now let us prove that the above heuristic is Admissible and Monotonic.

1.1 Admissibility

In order for a heuristic to be admissible it needs to satisfy :

$$\forall n \quad h(n) \leq h*(n) \tag{4}$$

Using Figure 1 we can derive the following equations:

Since we get the maximum value between dx and dy as the heuristic, it will always be the minimum amount of diagonal moves between the start and goal state. Therefore we can deduce that any cost will be greater than the heuristic and will never be less than it. So for the best case the heuristic will be equal to the actual cost, but for the worst case the heuristic will be underestimating since the cost of non-diagonal moves are greater than one.

Therefore the above heuristic is admissible as it satisfy Equation 4 and it does not overestimate the cost.

1.2 Monotonicity

In order for a heuristic to be monotonic, it should satisfy the following condition:

$$\forall n \quad h(n) \leq c(m,n) + h(m) \tag{5}$$

where m is a child of n

Using the same example from above (Figure 1), we can prove that the heuristic of any given node is less than or equal to the cost from that node to its successor plus the heuristic of the successor. This is because we take the max difference between the current node and the goal node.

1.3 Is the resulting algorithm A or A*?

Therefore since the actual cost is always used for the $g(n)$ value and not an estimate, we can state that the resulting algorithm is A*.

2 Tie Breaker Implementation

As show below, to implement the tie breaker, we override the Node instance's less than operator:

```
1   def __lt__(self, other):
2     '''
3     This method is used to override the less than operator in
4     python to use the f cost for comparison
5     @param other: the other node to be compared
6     @return: boolean value stating whether its less than or not
7     '''
8     if (self.f < other.f):
9       return True
10    elif(self.f == other.f ):
11      if self.operator in self.best_operators:
12        return True
13      elif other.operator in other.best_operators:
14        return False
15      else:
16        return True
17    else:
18      return False
19
```

As per line 10, if the cost of the nodes are equal, we prioritize the node which was generated using a diagonal operator such as "LU, LD, RU, RD". So the tie breaker implementation will always prioritize paths with diagonals.

3 Test Cases

3.1 Output for all the test cases

Below are the test cases and the resulting path from each algorithm:

3.1.1 input1.txt

```
3
SRR
RRX
RRG

DLS : S-RD-D-R-G 5
A* : S-RD-D-R-G 5
```

3.1.2 input2.txt

```
5
SRRXG
RXRXR
RRRXR
XRXRR
RRRRX

DLS : S-D-D-R-D-D-R-R-U-R-U-U-U-G 24
A* : S-D-D-R-D-D-R-R-U-R-U-U-U-G 24
```

3.1.3 input3.txt

```
        5
        SRXXX
        RRRXG
        XRRRR
        XRRRR
        RXXRX

        DLS : S-RD-RD-RD-RU-U-G 6
        A* :  S-RD-RD-RD-RU-U-G 6

```

3.1.4 input4.txt

```
        7
        RRRXRRR
        RXRRRXR
        RXXXXXR
        RRXSXXR
        XRXRXXR
        XRXRXXR
        XRRRXGR

        DLS : S-D-D-D-L-L-U-U-U-L-U-U-U-R-R-D-R-R-U-R-R-D-D-D-D-D-D-L-G 54
        A* : S-D-D-D-L-L-U-U-U-L-U-U-U-R-R-D-R-R-U-R-R-D-D-D-D-D-D-L-G 54

```

3.1.5 input5.txt

```
        5
        SRRRG
        RRRRR
        XXXXX
        RRRRR
        RRRRR

        DLS : S-RD-R-R-RU-G 6
        A* : S-RD-RU-RD-RU-G 4

```

3.1.6 input6.txt

```
        100
        Too big to display here...

        DLS : S-RD-R-D-D-D-R-R-RD-RU-R-R-RD-RD-RD-RU-U-U-U-U-R-R-R-R-RD-RD-RD-RD-RD-RU-U-U-U-U-R-R-R-R-RD-RD-RD-RD-RD-RU-U-U-U-U-R-R-R-R-RD-RD-RD-RD-RD-RU-U-U-U-U-R-R-R-R-RD-RD-RD-RD-RD-RU-U-U-U-U-R-R-R-R-RD-RD-RD-RD-RD-RU-U-U-U-U-R-R-R-R-RD-RD-RD-RD-RD-RU-U-U-U-U-R-R-R-R-RD-RD-RD-RD-RD-RU-U-U-U-U-R-R-R-R-RD-RD-RD-D-D-D-D-G 226
        A* : S-RD-RU-R-R-RU-RD-R-R-U-RU-RD-RD-RD-D-R-R-R-R-RU-RD-RD-RD-RU-RU-R-R-R-R-RU-RD-RD-RD-RU-RU-R-R-R-R-RU-RD-RD-RD-RU-RU-R-R-R-R-RU-RD-RD-RD-RU-RU-R-R-R-R-RU-RD-RD-RD-RU-RU-R-R-R-R-RU-RD-RD-RD-RU-RU-R-R-R-R-RU-RD-RD-RD-RU-RU-R-R-R-R-RD-LD-RD-RD-RD-LD-RD-G 147

```

3.1.7 input7.txt

```
4
XRGR
SXRR
RRXR
RRRX


DLS : NO-PATH
A* : NO-PATH

```

3.1.8 input8.txt

```
6
SRRRRR
RRRXXR
RXRRRR
RRXRXR
XRRRRR
GRRRXR

DLS : S-RD-R-D-R-D-D-LD-L-L-G 16
A* : S-D-D-D-R-D-D-L-G 14

```

3.1.9 input9.txt

```
6
RSRXGR
RXRXRR
RRXRXR
RRRRXR
RXRXRR
RRRRRR

DLS : S-L-D-D-RD-R-D-D-R-R-RU-U-U-U-LU-G 25
A* : S-L-D-D-RD-R-D-D-R-R-RU-U-U-U-LU-G 25

```

4 Analysis

In order to perform an analysis, multiple test cases were generated and tested on the two algorithms. Below is a table with the time taken for each input by the two algorithms. Please do note that each time is an average of five run.

Input File	**Time Taken**											
	DLS						**A***					
	1	2	3	4	5	**Average**	1	2	3	4	5	**Average**
input1.txt	0.00024	0.00017	0.00023	0.00017	0.00017	**0.00020**	0.00028	0.00028	0.00028	0.00033	0.00028	**0.00029**
input2.txt	0.00068	0.00048	0.00047	0.00048	0.00049	**0.00052**	0.00067	0.00067	0.00065	0.00067	0.00067	**0.00067**
input3.txt	0.00035	0.00034	0.00035	0.00035	0.00035	**0.00035**	0.00081	0.00068	0.00058	0.00057	0.00057	**0.00064**
input4.txt	0.00073	0.00073	0.00072	0.00072	0.00084	**0.00075**	0.00152	0.00152	0.00150	0.00152	0.00150	**0.00151**
input5.txt	0.00035	0.00027	0.00027	0.00027	0.00027	**0.00029**	0.00029	0.00029	0.00028	0.00027	0.00029	**0.00028**
input6.txt	0.08533	0.08353	0.08298	0.08362	0.08385	**0.08386**	10.35549	11.44547	10.61212	11.73250	11.42238	**11.11359**
input7.txt	0.00026	0.00021	0.00023	0.00022	0.00022	**0.00023**	0.00022	0.00026	0.00023	0.00023	0.00023	**0.00023**
input8.txt	0.00061	0.00059	0.00049	0.00049	0.00050	**0.00054**	0.00081	0.00082	0.00087	0.00083	0.00083	**0.00083**
input9.txt	0.00090	0.00089	0.00089	0.00087	0.00088	**0.00089**	0.00102	0.00103	0.00102	0.00104	0.00103	**0.00103**

Table 1: Time Taken for each Algorithm on each input file