# Heuristic Function

My heuristic function is defined as the straight-line distance divided by 2, the problem is relaxed by assuming agent can move in any direction with unit cost, and there is no mountain.

Proof:

heuristic is defined as

For monotonicity:

m is child node of n.

when m is generated by horizontal/vertical, according to triangle inequality.

distance(m, n) + distance(m, g) > distance(n, g)

1 + 2 \* h(m) > 2 \* h(n)

½ > h(n) – h(m)

h(n) – h(m) < 2 = cost(n, m)

h(n) <= cost(n, m) + h(m)

when m is generated by diagonal move, according to triangle inequality.

distance(m, n) + distance(m, g) > distance(n, g)

sqrt(2) + 2 \* h(m) > 2 \* h(n)

sqrt(2)/2 > h(n) – h(m)

h(n) – h(m) < sqrt(2)/2 < 1 = cost(n, m)

h(n) <= cost(n, m) + h(m)

therefore the

h(n) <= cost(n, m) + h(m) holds for all n, m.

Hence monotonicity is preserved.

And since h is monotonic, it is also admissible

Therefore, the resulting algorithm is A\*.

Notes: sqrt represents the function compute square root.

Tie breaking rule:

if operator have the same degree of merit, then choose the operator which comes first in following list

[R, RD, D, LD, L, LU, U, RU]

# Screenshot of output and map

The following output is generated by running generate\_solution.py in submission.

please run generate\_solution.py if following screenshot is unclear.



The output files/diagnose mode output can be generated by running generate\_output.sh in submission.

diagnose mode, please run generate\_output.sh if the screenshot is unclear.



### analysis

The runtime complexity for depth limited search is same with O(b powered by l) where b is branching factor and l is the limit.

The limit in my implementation is chosen to be number of normal tile in the map which is bounded by total number of tile in map s^2.

therefore, it’s O(b^(s^2))

The complexity of A algorithm depending on the quality of heuristic function.

For chosen heuristic, delta = max|h – h\*| = s - 2 \* (s^2) / 2 , where s represent of width of square map.

so complexity is O(b ^ (s ^ 2)) in worst case. (such worst case is tested in input8.txt)

time measured in second

input1.txt 0.165sfor DLS and 0.165 for A\*

input2.txt 0.162 for DLS and 0.178 for A\*

input3.txt 0.171 for DLS and 0.173 for A\*

input4.txt 0.175 for DLS and 0.169 for A\*

input5.txt 0.172 for DLS and 0.172 for A\*

input6.txt 0.177 for DLS and 0.194 for A\*

input7.txt 0.183 for DLS and 0.179 for A\*

input8.txt 0.177 for DLS and 0.166 for A\*

input9.txt 0.169 for DLS and 0.172 for A\*

|  |  |  |  |
| --- | --- | --- | --- |
| input testcase number | DLS time | A\* time | DLS time – A\* time |
| 1 | 0.165 | 0.165 | 0 |
| 2 | 0.162 | 0.178 | -0.016 |
| 3 | 0.171 | 0.173 | -0.002 |
| 4 | 0.175 | 0.169 | 0.006 |
| 5 | 0.172 | 0.172 | 0 |
| 6 | 0.177 | 0.194 | -0.017 |
| 7 | 0.183 | 0.179 | 0.004 |
| 8 | 0.177 | 0.166 | 0.011 |
| 9 | 0.169 | 0.172 | -0.003 |
|  |  |  |  |

By measuring the actual runtime of program, we can see that the time difference is not significant. The reason of that could be I did not test the program on large map and upper bound of both algorithm in my implementation is the same.

In the test A\* found optimal path for all test cases (if there is any).

while DLS does not found optimal path for test case 3, 4, 5, 6, 7 and 9. But it still found at least a path if there exist any.

### Note on testing:

I have added a few more test cases for different configuration of map. and there is a file named generate\_testcase.py in submission, which is used in testcase generation. But the generated testcase need some handcraft in order to do more specific testing.