CMPE 480 I.Project Report

Şadi Uysal

2015400162

Outputs for level1.txt

$ python proj1.py level1.txt bfs

11 68 7 7

RRDRRRD

$ python proj1.py level1.txt dfs

62 47 38 38

RRDLLULURDLLURDDLURRDRRRRDLLULDRRDLURD

python proj1.py level1.txt ucs

10 58 8 8

DRRRRRRD

$ python proj1.py level1.txt gs

10 15 8 8

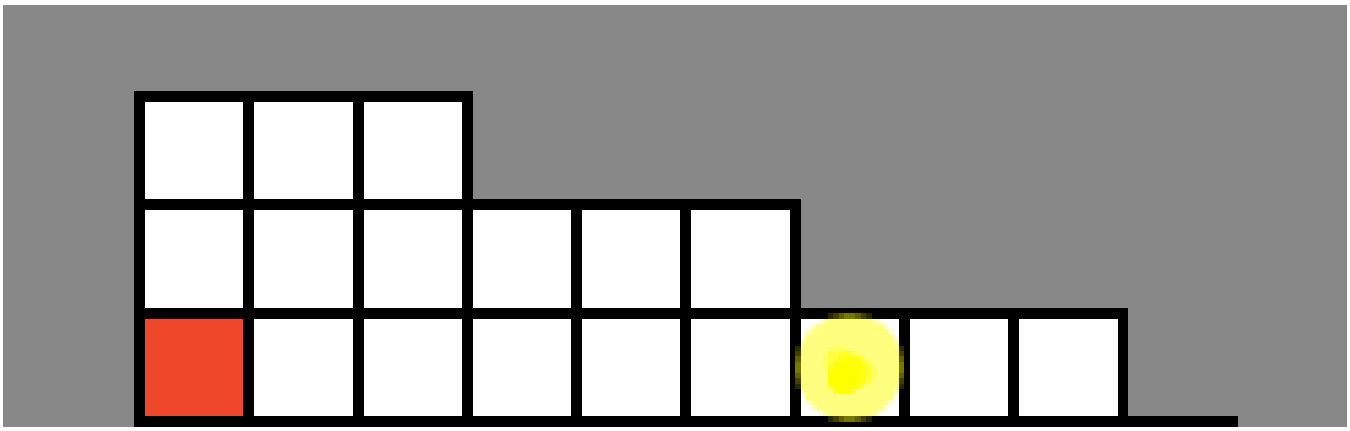
DRRRRRRD

$ python proj1.py level1.txt as

10 13 8 8

DRRRRRRD

Since edge costs between the states not the same for every transition, BFS do not have optimality here. DFS also do not have optimality as we know. For the UCS case, it have optimality since it searches according to cost not the depth of the nodes.



Heuristic: Imagine above scenario which red grid is the start state and the yellow one is the goal state. Firstly i thought about the using manhattan distance as a heuristic value but in the above scenario, even if the manhattan distance is 6 box we can win with just 4 moves. For those similar cases, our average speed is 3/2 box per move. In other words, our number of moves is manhattan distance/average speed(3/2 box per move). So, i decided to use heuristic value(manhattan distance/(3/2)) for special grids in the puzzle and manhattan distance for non-special grids. I found out some special grids that have high potential to have lower moves to win. Those are:

single orientation states which have location>>

(state_x_value-goal_x_value)/3==0 and (state_y_value-goal_y_value)/3==0

vertical orientation states which have location>>

((state_y_value-goal_y_value)-2)/3==0

horizontal orientation states which have location>>

((state_x_value-goal_x_value)-2)/3==0

So, while searching if the program finds any grid that have high potential to have lower moves to win, it chooses that grid for expansion. With doing that, i get great improvement for the number of nodes expanded for greedy search and also for a* search.