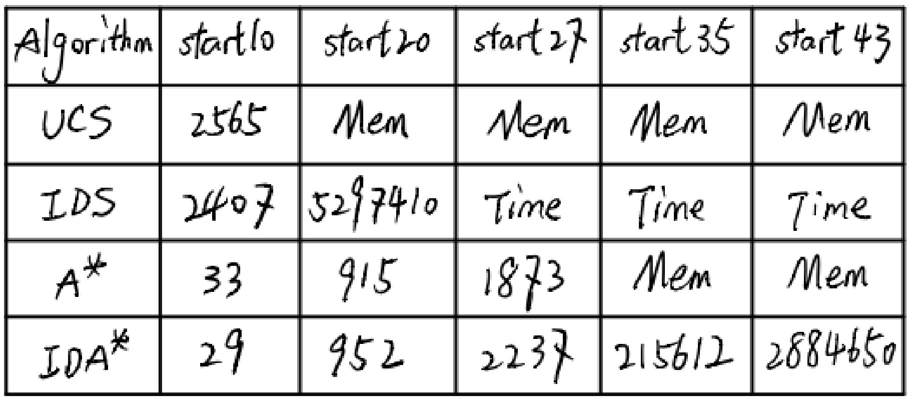
# COMP9414
# Assignment 2
Zheyuan Xu (z5190669)

## Question 1: Search Algorithms for the 15-Puzzle

(a)



| Algorithm | start10 | start20 | start27 | start35 | start43 |
|---|---|---|---|---|---|
| UCS | 2565 | Mem | Mem | Mem | Mem |
| IDS | 2407 | 5297410 | Time | Time | Time |
| A* | 33 | 915 | 1873 | Mem | Mem |
| IDA* | 29 | 952 | 2237 | 215612 | 2884650 |

(b)

*Uniform cost search:*

Terrible performance in solving Puzzle 15. The space complexity is the highest among the four algorithms.

*Iterative Deepening search:*

It generates lots of nodes in start20 and has the highest time complexity among the four algorithms.

*A* search:*

It has good performance in solving Puzzle 15, but as situation gets more complicated, this algorithm will run out of memory.

*Iterative Deepening A* search:*

The most efficient one among the four algorithms in solving Puzzle 15 and it's the only one can solve nodes out in start35 and start43.

Question 2: Deceptive Starting States

(a)

By running the Prolog program, the heuristic value for start49 is 25, for start51 the heuristic value is 43.

(b)

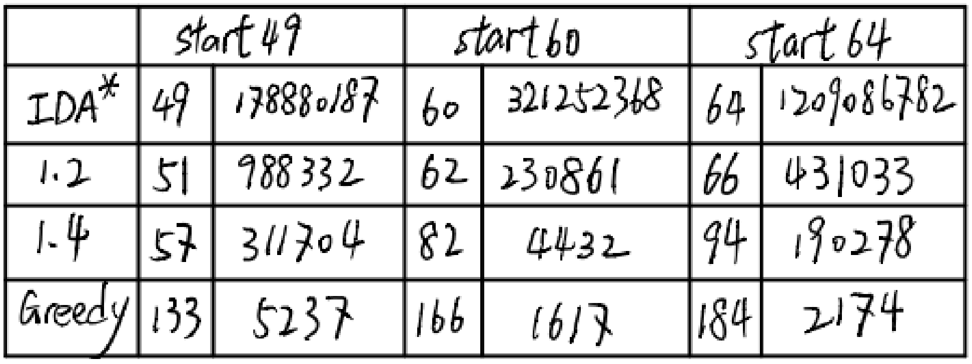
By running the Prolog program, the number of nodes expanded starting from start51 is 551168.

(c)

As we can see from the previous questions, when start49 the heuristic value is 25 and for start51 it's 43. The heuristic value in start49 is smaller means the probability of visiting the same nodes for start49 is much higher than start51. IDA* is a combination of Iterative Deepening search and A* search, a parameter thereshold controls the depth of iteration, a higher heuristic value can help to reduce the time running from start to the thereshold depth, however a smaller value can easy to be trapped into a low searching efficiency problem since there may have many iterations. So the time complexity between these two can be huge.

Question3: Heuristic Path Search

(a)



| | start 49 | | start 60 | | start 64 | |
|---|---|---|---|---|---|---|
| IDA* | 49 | 178880187 | 60 | 321252368 | 64 | 1209086782 |
| 1.2 | 51 | 988332 | 62 | 230861 | 66 | 431033 |
| 1.4 | 57 | 311704 | 82 | 4432 | 94 | 190278 |
| Greedy | 133 | 5237 | 166 | 1617 | 184 | 2174 |

(b)

```
depthlim(Path, Node, G, F_limit, Sol, G2) :-
    nb_getval(counter, N),
    N1 is N + 1,
    nb_setval(counter, N1),
    % write(Node), nl,    % print nodes as they are expanded
    Not(member(Node1, Path)),    % Prevent a cycle
    G1 is G + C,
    h(Node1, H1),
    F1 is 0.8*G1 + 1.2*H1,      % changed part
    F1 =< F_limit,
    Depthlim([Node| Path], Node1, G1, F_limit, Sol, G2).
```

(c)

Changed part: F1 is 0.6*G1 + 1.4*H1

(d)

Actually when w = 1, it's IDA* search and when w = 2, it's Greedy search. We can see from the experiment results that: when w become larger, path become longer and no longer optimal, but the number of expand nodes become smaller, which will decrease the time complexity.

Question 4: Maze Search Heuristics

(a)

Manhattan Distance:

$$h(x,y,x_G,y_G) = |x-x_G| + |y-y_G|$$

b(i)

No.

Let's say we want to move from (0,0) to (2,1), one choice is, we first go right from (0,0) to (1,0) and then we move diagonally from (1,0) to (2,1). So there is total 2 cost. However, $h_{SLD} = \sqrt{2^2+1} = \sqrt{5}$ , which is larger than 2, so it's not admissible.

b(ii)

No.

Similar to b(i), if we consider the cost of a path is the total number of moves to reach the goal, it costs 2 when we move from (0,0) to (2,1). However, the Manhattan Distance is $|0-2| + |0-1| = 3$, which is larger than 2, so it's not admissible.

b(iii)

The best formula for this problem is:

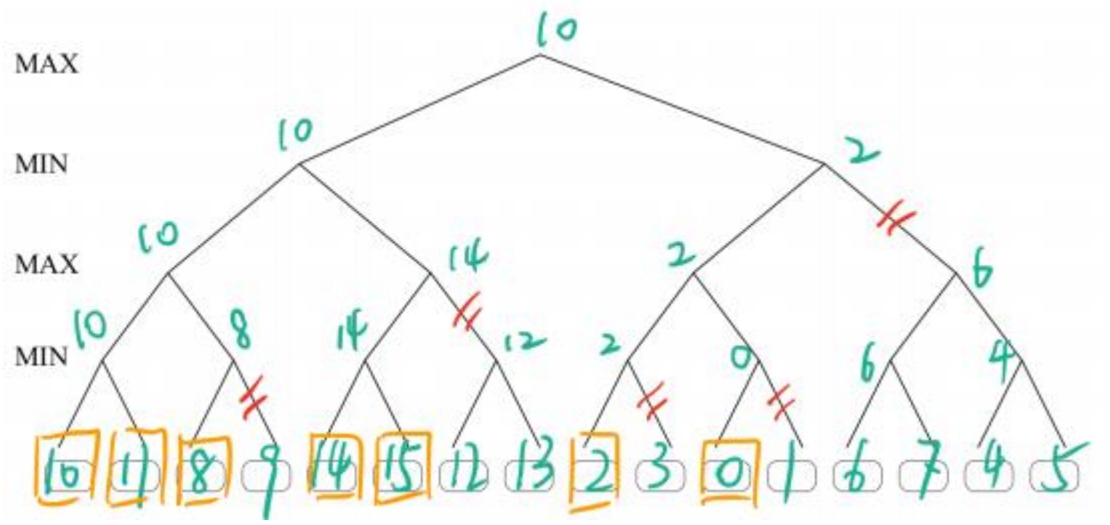
$$h(x,y,x_G,y_G) = \max(|x-x_G|, |y-y_G|)$$

The cost is the maximum value of travel in either direction, so it must be admissible.

Question 5: Game Trees and Pruning

(a) and (b)

From the hint we know that: if we want to prunes as many nodes as possible, then we should put smaller values first on the left leaves, there exists many solutions, here is the one I think of:



First, we traversing to 10, then 11, since α = -∞ < 10 ,10 < 11, so the min node is assigned 10, and the max node above is also assigned 10. Since 10 < β = ∞, we traversing to the right subtree and reach 8, then 8 is assigned to the min node. Since α = 10 > β = 8, the max node will not consider to pick the right subtree, 9 can be cut off.

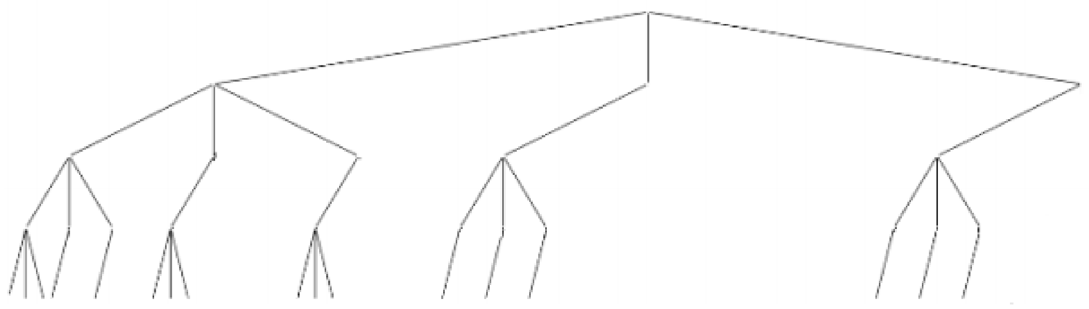
Then we move to the right subtree of the left subtree, since β = 10 > α = -∞, we reach 14, then 15, now α is updated to 14, min node will not consider 14 because it already got 10 on the left subtree, so 12, 13 will be cut off.

We keep doing similar calculation on the right subtree and finally we will find that only 7 leaves need to be evaluated(which are 10, 11, 8, 14, 15,2, 0 in this case), the rest can be pruned.

(c)

In this case, the visit order is important. We should ensure min nodes visit smaller values first and max nodes visit larger values first, then we can prune other branches.

The most ideal scenario is as follows (we ignore branches can be pruned):



As we can see, only 17 leaves need to be evaluated.

(d)

If the best move is always exam first, then it means we need to expand all children at a max node and only need to expand one child at a min node. The reason is: Player1 needs to find the best move among all the possibility, however for Player2, he only needs to find a worse case for Player1.

Hence for a tree with constant branching factor b and depth d, The time complexity is $O(b*1*b*1*b*\cdots) = O(b^{d/2})$.

----------End of Assignment----------