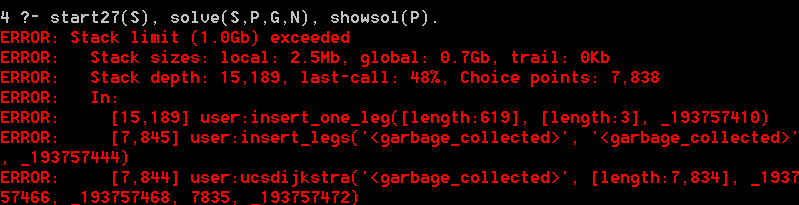
Q1. Search Algorithms for the 15-Puzzle



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 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)

1. UCS: UCS has large space complexity and time complexity as well, which are, where c\* is cost of optimal solution and we assume that every action cost at least , So it is easy to run out of memory.

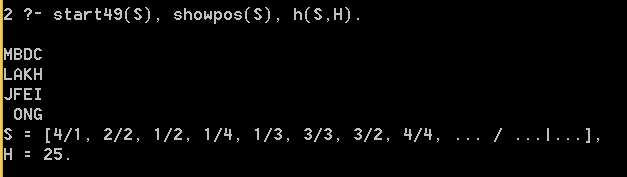
2. IDS: IDS visits all nodes. So it easily run out of time and as a result Its time complexity is O(),space complexity is O(bd), much smaller than UCS.

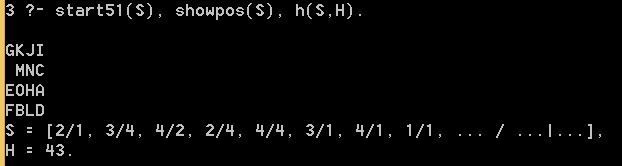
3. A\*: A\* also uses a lot of space, because it will evaluate the total cost in order to find a cheapest way, it keeps all the nodes in the memory. But it less than UCS. It combines the advantages of UCS and greedy search, so it runs faster than UCS and IDS. The space complexity of it is O() for worst situation, and the time complexity of it is based on the every action cost at least ,which is up to O().

4. IDA\*: IDA\* can pass all the test, so it is the most efficient algorithm and perform well in both time and space complexity. The space complexity of it is O(bd) (reduced from exponential growth to linear growth), less than A\*, and the time complexity of it is .

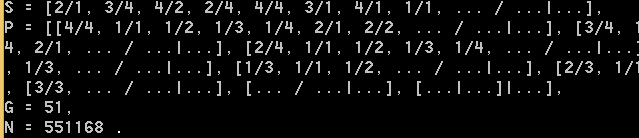
Q2. Deceptive Starting States

1. Start49:



Start51:

1. The number nodes expanded during the search is 551168.



1. This is because it has repetitive search during the search. So it expands so many more nodes.

Q3. Heuristic Path Search

a)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Start49 | | Start60 | | Start64 | |
| IDA\* | 49 | 178880187 | 60 | 321252368 | 64 | 1209086782 |
| 1.2  1.4 | 51  57 | 988332  311704 | 62  82 | 230861  4432 | 66  94 | 431033  190278 |
| Greedy | 123 | 5237 | 166 | 1617 | 184 | 2174 |

b)

the original code:

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

s(Node, Node1, C),

not(member(Node1, Path)), % Prevent a cycle

G1 is G + C,

h(Node1, H1),

F1 is G1 + H1,

F1 =< F\_limit,

depthlim([Node|Path], Node1, G1, F\_limit, Sol, G2).

the changed code(changed part is red):

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

s(Node, Node1, C),

not(member(Node1, Path)), % Prevent a cycle

G1 is G + C,

h(Node1, H1),

**F1 is 0.8\*G1+1.2\*H1**

F1 =< F\_limit,

depthlim([Node|Path], Node1, G1, F\_limit, Sol, G2)

.

This is based on f(n)= G1\*(2-W) + H1\*W, W=1.2.

c). see answer in answer a

d) As w increases, the speed increase and total number of states expanded during the search decreases (which is N), but the length of path increases as well (which is G).

Q4. Maze Search Heuristics

1. .The formula for Straight Line-Distance heuristic :

b).1). No, because based on the definition of heuristics, the actual cost must lager than the calculated cost. it cost √2(straight line), which is less than 1(should be), so it is not admissible.

b).2). No, it is not admissible, because the moves computed by heuristic in (based on Q1a) is 2, more than real moves which is 1. So it is not

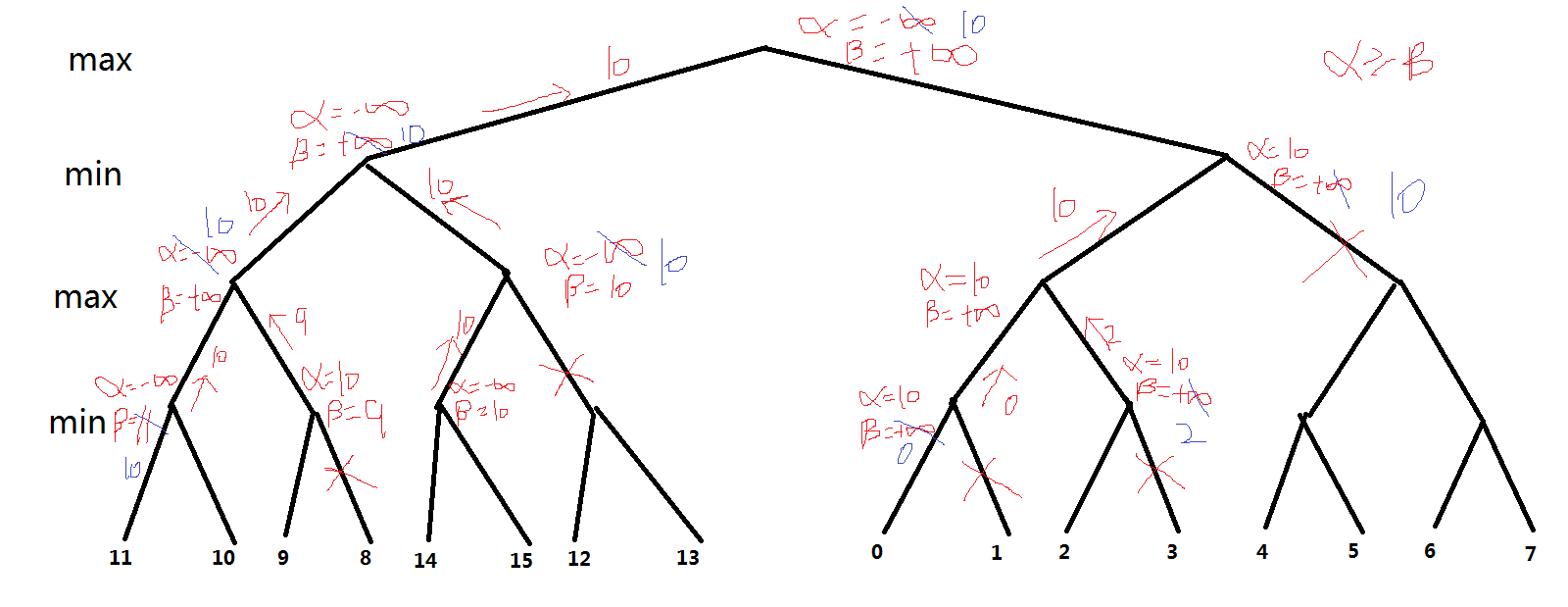
admissible.

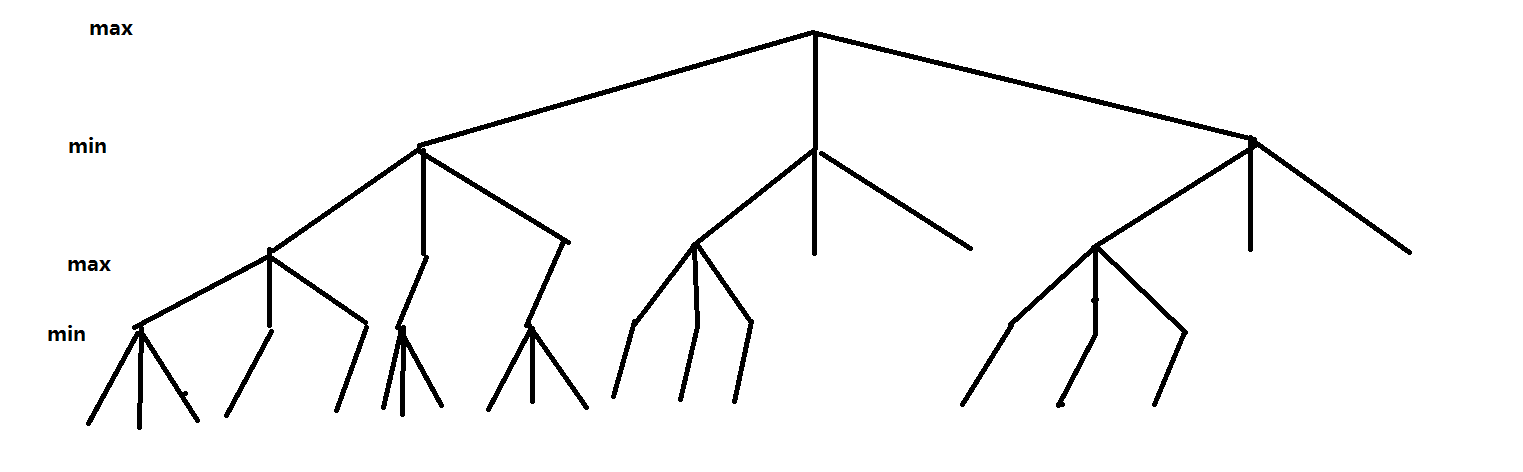
b).3). The best admissible heuristic I can think of for this problem is



Q5. Game Trees and Pruning

a)



1. As we can see in question a, there are 1+2+1+1+4=9 nodes are pruned and left 7 are evaluated.
2. 

The first three nodes we need to explore, if the other left leaves are less than the first parent we have chosen, then we back the data to the second parent. So if middle and right sub-tree of the left sub-tree is larger, then we can prune after visiting the first three leaves of each left sub-tree. Then we have the root that with value of the leftmost sub-tree, as long as the rest value is less than is this value, we can prune. So the final result is showed above.

There are only 3+1+1+3+3+6=17 of the original 81 tree leaves are evaluated.

The time complexity is about , where b = branching factor, d = depth of the tree. If all leaves are unpruned, the amount of leaves can be computed as b\*b\*b\*…\*b(total d times b) which is , So the average time complexity is about .