Student ID: z5192086
Student Name: Pan Luo
File Description: COMP9414 Assignment2

Question 1: Search Algorithms for the 15-Puzzle

(a)

	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)

For UCS:

Time: $O(b^{\lceil C^*/\varepsilon \rceil})$ where C^* = cost of optimal solution, and assume every action costs at least ε.

Space: $O(b^{\lceil C^*/\varepsilon \rceil})$ ($b^{\lceil C^*/\varepsilon \rceil} = b^d$ if all step costs are equal).

UCS is a generate view above BFS which contains the step cost in expand every node. As the table shows, it is highly likely for UCS to exceed memory limit. Because it still keeps lots of nodes in memory which likes BFS.

For IDS:

Time: $O(b^d)$

Space: O (bd)

The space complexity for IDS reduce a lot because it only keeps one branch the memory. So there is no memory exceeding. But the time complexity is too high that some longer goal can not find in particular time.

For A*:

Time: Exponential in [relative error in h × length of solution]

Space: Keeps all nodes is memory

The core of A* algorithm is the heuristic function h(n)=f(n)+g(n), where f(n) is the estimated cost of cheapest solution through node n and g(n) is the cost from initial node to node n. So the choose of h(n) will influence A* algorithm a lot. A* keeps every node in memory so as the table shows it sometimes exceeding memory.

For ida*:

Ida* is similar to a* but it pre-compute some states instead of using algorithm to find them. So ida* does not need so much memory.

Question 2: Deceptive Starting States:

(a)

For start49(S), the result is:

MBDC
LAKH
JFEI
 ONG
S = [4/1, 2/2, 1/2, 1/4, 1/3, 3/3, 3/2, 4/4, ... / ...|...],
H = 25.

For start51(S), the result is:
GKJI
 MNC
EOHA
FBLD
S = [2/1, 3/4, 4/2, 2/4, 4/4, 3/1, 4/1, 1/1, ... / ...|...],
H = 43.

(b)
When use idastar for start51,the number of expended nodes is 551168.

(c)
For ida* algorithm, like a*, the true path length is just one composition of heuristic function h(n), another is the estimated cost from initial node to node n. So maybe start49 and start51 has similar true path length to the goal, but they differ a lot in the estimated cost from initial node.

Question 3: Heuristic Path Search:

	Start49		Start60		Start64	
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) the original code is:

```
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).
```

Now, I modify the red part(F1 is G1 + H1) because the original code means w=1 in the function f(n) = (2-w)g(n) + wh(n). So when w=1.2, the function should be f(n) = 0.8g(n) + 1.2h(n).

So the red part should replace by F1 is 0.8*G1+1.2*H1.

(d)The objective function for heuristic algorithm is f(n)=(2-w)g(n) + wh(n).

The original version of code means w=1. The greedy search means w=2.
As the table shows, when w increases, G(the length of path) increases slightly, but N(the number of expanded node) decreases dramatically.

When w increases from 1 to 2. The speed of finding the goal is more faster, but the algorithm do not return the optimal solution which path is more longer. So it is a tradeoff between speed and optimal. What's more, when w=2(greedy search), it is likely to get stuck in loops, so it is also a problem.

Question 4: Maze Search Heuristics

(a)In this case, Manhattan distance dominates the Straight-Line-Distance. The formula of Manhattan distance is $h(x, y, x_G, y_G) = |x-x_G|+|y-y_G|$

(b)

(i)No, the Straight-LineDistance heuristic is not heuristic. Because as the assumption says "a diagonal step is still considered to have the same "cost" (i.e. one "move") as a horizontal or vertical step". But in actual the diagonal move is √2. So the estimated cost is large than true cost which is against the definition of admissible.

(ii)No, because as the formula shows, Manhattan distance is $h(x, y, x_G, y_G) = |x-x_G|+|y-y_G|$. Considering the diagonal cost, Manhattan cost is 2, but the true cost is 1. So the eatimate cost is large than the true cost which is against the definition of admissible.

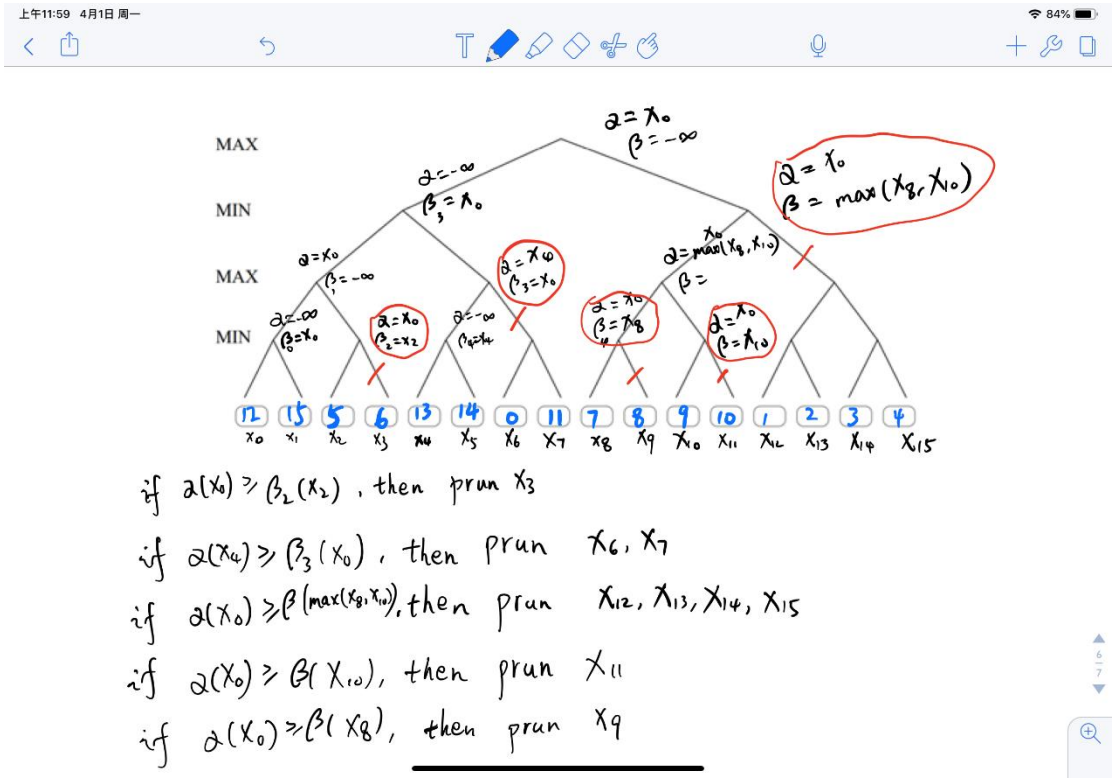
(iii)According to the assumption, the true cost of two points is the absolute value of the distance in one axis. Because the diagonal cost is also 1, so the gap between one axis is the total true cost. So the formula is

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

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

Question 5: Game Trees and Pruning

(a)



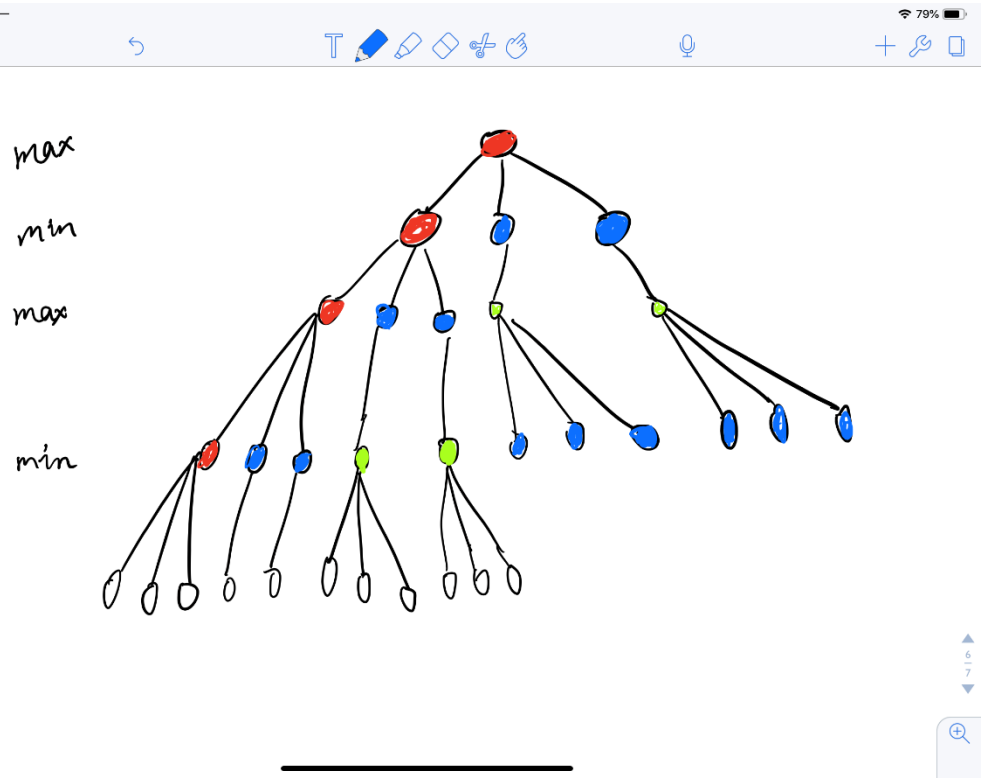
According to the requirement, all the value of left child node should smaller than that right child node. So $X_0<X_1$, $X_2<X_3$, ……$< X_{14}<X_{15}$, and in order to prun as many nodes as possible, when the value of $X_0>=X_2$, then prun X3. When $X_4>=X_0$, prun X_6 and X_7. When $X_0>=X_8$, prun X_9. When $X_0>=X_{10}$, prun X_{11}. When $x_0>=\max(X_8,X_{10})$, prun $X_{12},X_{13},X_{14},X_{15}$.

Given 0 to 15, the possible value of this tree is shown by blue figures.

(b)

As is shown in the picture, 9 nodes are pruned which invovled $X_3,X_6,X_7,X_9,X_{11},X_{12},X_{13},X_{14},X_{15}$, and 7 nodes are evaluated which invovled $X_0,X_1,X_2,X_4,X_5,X_8,X_9$

(c)



As is shown in the picture, the red nodes mean PV nodes. The blue nodes mean starting from a PV-node but proceeding through non-PV nodes. The green nodes mean the lowest child of blue nodes.

For the blue nodes, only need to evaluate 1/3 of its children. And for red and green nodes, all of their children should be evaluated.

So the total number of evaluated nodes are: [(3+1+1)+3+3]+[1+1+1]+[1+1+1] = 17

(d) With “perfect ordering,” the time complexity of alpha-beta search = $O(b^{m/2})$ where b means the number of branches and m means the search depth. Because the total number of nodes are b^m, but if the best moves always examined first, then for the odd number depth, branches need to evaluate all their nodes. But for even number branches, only half of the child need to evaluate. So the time complexity reduce from b^m to $b^{m/2}$.