COMP9414 - Assignment 2

Hao Chen 5102446

Question 1 - Maze Search Heuristics

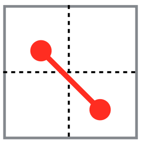
(a)

The Manhattan-Distance dominates the Straight-Line-Distance, and the formula for it is:

$$h(x, y, x_G, y_G) = |x\text{-}x_G| + |y\text{-}y_G|$$

(b - i)

No. The Straight-Line-Distance heuristic is not admissible. In the question we consider the agent having the same cost when it move up, down, left, right or diagonally, while the actual cost when the agent move from one gird centre to the other grid centre, it cost √2, which is larger than 1. Based on the definition of heuristics, the actual cost must lager than the calculated cost, while 1 <= √2, so the Straight-Line-Distance heuristic is not admissible.



(b - ii)

No. If the agent is considered to have the same cost with diagonally, horizontal and vertically, then the diagonally travel will be the same as only travel horizontally or vertically, while the Manhattan-Distance will be a sum of the travel horizontally and vertically.

(c - iii)

$$h(x, y, x_G, y_G) = \begin{cases} |x_G - x| \ \mathit{if} \ x_G - x \geq y - y_G \\ |y - y_G| \ \mathit{if} \ x_G - x < y - y_G \end{cases}$$

Question 2 - Search Algorithms for the 15-Puzzle

(a)

	Start 10	Start 12	Start 20	Start 30	Start 40
UCS	2565	Mem	Mem	Mem	Mem
IDS	2047	13812	5297410	Time	Time
A*	33	26	915	Mem	Mem
IDA*(Man)	29	21	952	17297	112571
IDA*(Mis)	35	87	4345	2105465	Time

(b)

The changed code:

```
misdist(X/Y,X1/Y1, 0) :-        % if X=X1 and Y=Y1, then this grid do not have to be moved, return 0.
    X = X1,
    Y = Y1, !.

misdist(X/Y,X1/Y1, 1).          % if else, then this grid do not have to be moved, return 1.
```

This part of changed code is write the misdist function, which will return 0 if this grid is on the right place, return 1 if it is on the wrong place.

```
totdist([Tile|Tiles], [Position|Positions], D) :-
    misdist(Tile, Position, D1),
    totdist(Tiles, Positions, D2),
    D is D1 + D2.
```

This part is change the way of calculate f(n) from Manhattan-Distance to Count Misplaced Tiles.

(c)

Base on the table shown, we can see that the efficiency of UCS is the lowest, and it has the highest space complexity.
The IDS does not perform good, and it has the second lowest efficiency, it will run out of time at 30 and 40, and it has to 5297410 nodes when start from 20, which is expensive, and it has the highest time complexity.
The A* have a good efficiency, it perform good when it search 10, 20, 30 depth, but it cost too much memory.

The Manhattan-Distance has the best efficiency among all algorithm, and it is the only one who can reach the start 40, while IDA(mis)'s efficiency is becoming slower and slower when puzzle get mess, and it finally run out of time.

Question 3 - Heuristic Path Algorithms for the 15-Puzzle

(a), (c)

	Start 50		Start 60		Start 64	
IDA*	50	1462512	60	321252368	64	1209086782
1.2	52	191438	62	230861	66	431033
1.4	66	116174	82	3673	94	188917
1.6	100	34647	148	55626	162	235852
1.8	236	6942	314	8816	344	2529
Greedy	164	5447	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
    s(Node, Node1, C),
    not(member(Node1, Path)),      % Prevent a cycle
    G1 is G + C,
    h(Node1, H1),
    W is 1.2 % W = 1.2/1.4/1.6/1.8
    F1 is  (2-W) * G1 + W*H1,
    F1 =< F_limit,
    depthlim([Node|Path], Node1, G1, F_limit, Sol, G2).
```

The changed code is add W as a parameter when calculate the F.The objective function is f(n) = (2-w)*g(n) + w*h(n).

(d)

This algorithm reduces to Uniform Cost Search when w=0, to A*Search when w=1 and to Greedy Search when w=2. When the W is growing from 1.2 to 1.8, the length of the path is growing while the number of nodes expanded is getting less and less. So the quality is becoming bad but the speed is getting faster when W changes from 1.2 to 1.8.

Question 4 - Game Tress and Pruning

(a)

Name X(0) to X(15) follow the sequence from left to right, we can set X(0) < X(1), X(2) < X(3), X(4) < X(5), X(6) < X(7), X(8) < X(9), X(10) < X(11), X(12) < X(13), X(14) < X(15).

Based on the alpha-beta algorithm, we can deduce the regular below:

When X(0) ≥ X(2), cut X(3).

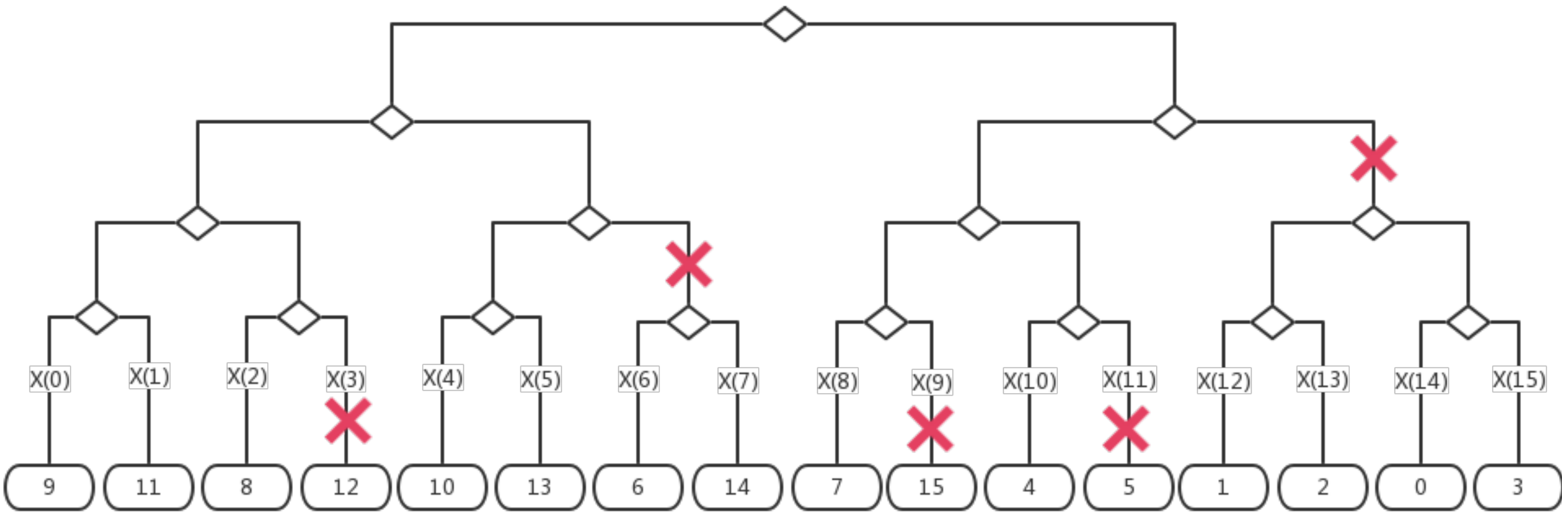
When X(4) ≥ X(0), cut X(6), X(7).

When X(0) ≥ X(8), cut X(9).

When X(0) ≥ X(10), cut X(11).

When X(0) ≥ X(8) or X(0) ≥ X(10), cut X(12), X(13), X14, X(15).

One of the possible tree is shown in the graph below.



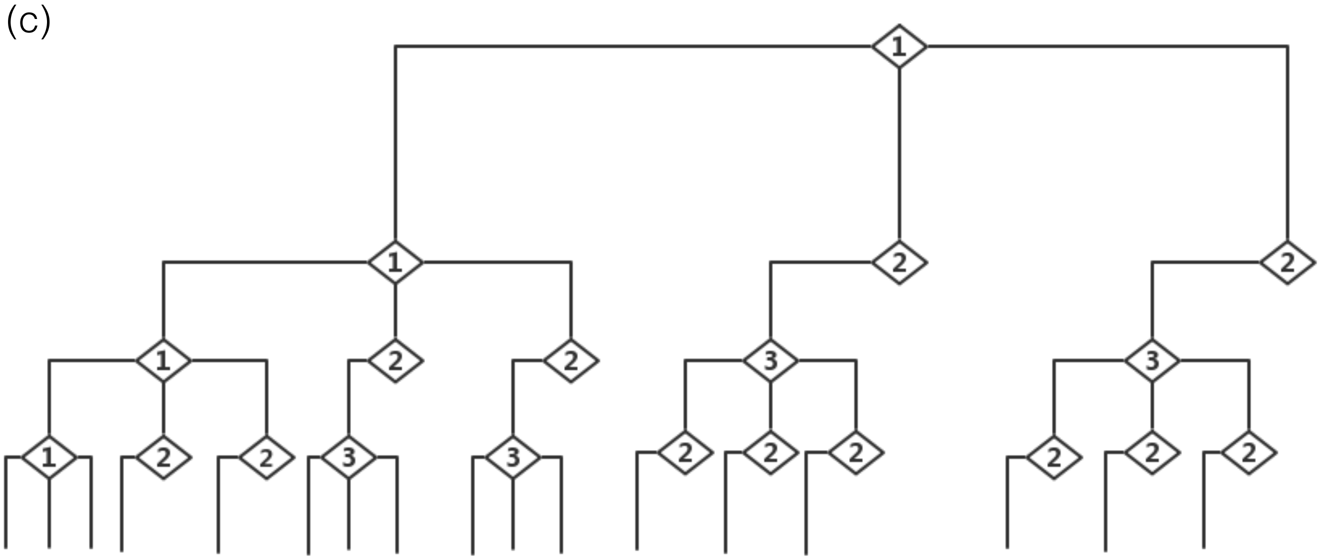
(b)

There are 7 of the original leaves are evaluated together.

The leaves are cut are already shown in the graph.

As shown in the graph the path with red X means that this path is cut.

(c)



In this case that alpha-beta algorithm can cut as many nodes as possible, then the node can be divided into 3 types(1,2,3). Node level 1 is the PV-nodes. The node level 2 is the brother node of node level 1, the child node of level 2 is 3, and the child of level 3 is node level 2.

For node 1 and 3 need to explore all nods below it, and for the node level 2 just have to explore the left child node, while the other child can be cut.

(d)

The time complexity of alpha-beta search is O(b^(d/2)). With an branching factor of b, and a search depth of d plies, the maximum number of leaf node positions evaluated is O(b*b*...*b) = O(b^d). If the best move s are always first, the number of leaf evaluated is about O(b*1*b*1*...*b) for odd depth and O(b*1*b*1*...*1) for even depth, which is O(b^(d/2)).