Comp3411 Assignment2

Q1

a)

N	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 the first search(ucsdijkstra) does not use any heuristics and stores every solution which its costs are less than the solution into the memory. The need of ram and time increases geometrically when the tree grows deeper and hence it gets stuck at start20.

Result:
Time efficiency: ★
Space efficiency: ★

The second searching algorithm (IDS) is a little bit smarter than the previous one in the in terms of saving ram. The iterative depth first search does not need to store all the previous result into the ram but it still needs a lot of time to run through every result. As a result, this algorithm takes too much time at start27.

Result:
Time efficiency: ★
Space efficiency: ★★★★★

The third algorithm is A * search. This algorithm needs heuristics to support . Comparing the the UCS, this result massively increase the time efficiency since it does not need to explore as many nodes as that in UCS and still can achieve the same goal (find optimal result) as long as the heuristics is correct. Since it still needs to store all the previous result into the ram, it will eventually run out of memory as the search tree grows large.

Result:
Time efficiency: ★★★★★
Space efficiency: ★★★

The IDA* algorithm is an algorithm that improves the A* search in the aspect of saving memory by giving up a little bit of its speed as a trade off. This algorithm does not store all the paths it founds while searching. Instead, it needs to repeat the search for not storing them. It might sound wasteful to do this, but in practice it does not affect its speed by too much.

Time efficiency: ★★★★
Space efficiency: ★★★★★

Q2
a)
//code
$start49(S), showpos(S), h(S,H).

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

//code
$start51(S), showpos(S), h(S,H).

GKJI
 MNC
EOHA
FBLD
S = [2/1, 3/4, 4/2, 2/4, 4/4, 3/1, 4/1, 1/1, ... / ...|...],
H = 43.

b)
551168

c)
The value of H (heuristic) is too small for the 49 which is significant different from the true cost. The underestimation determines that it has to work through all the possibilities that are not possible to reach the target.

Q3

	Start49		Start60		Start64	
IDA*(1.0)	49	178880187	60	321252368	64	1209086782
1.2	51	988332	62	230861	66	431033
1.4	57	311704	82	3673	94	188917
Greedy(2.0)	133	5237	166	1617	184	2174

a) b) c)

```
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 (2-1.2)*G1 + (1.2)*H1,
    F1 =< F_limit,
    depthlim([Node|Path], Node1, G1, F_limit, Sol, G2).
```

d)
The adjustment of w can be seen as a trade off between time and the quality of of the result. When using A* search which it does not over over-estimate the heuristics, it ensures we can find the optimal result but this takes a lot of time.

The greedy algorithm on the other hand does not guarantee its result is optimal. In practice, its output is very bad. The good part of greedy algorithm is that it is a very fast algorithm.

The other 2 algorithms in between are neither greedy algorithm nor A*. Instead, it is something between them. This algorithm would have a better time performance than A* and better quality of result than greedy. As w increases, the algorithm would act more like greedy and it will perform exactly the same as greedy as w=2. Similarly, as w decreases, it would act more like A * and will perform exactly the same as A* as w=1.

Q4

a) Manhattan distance:

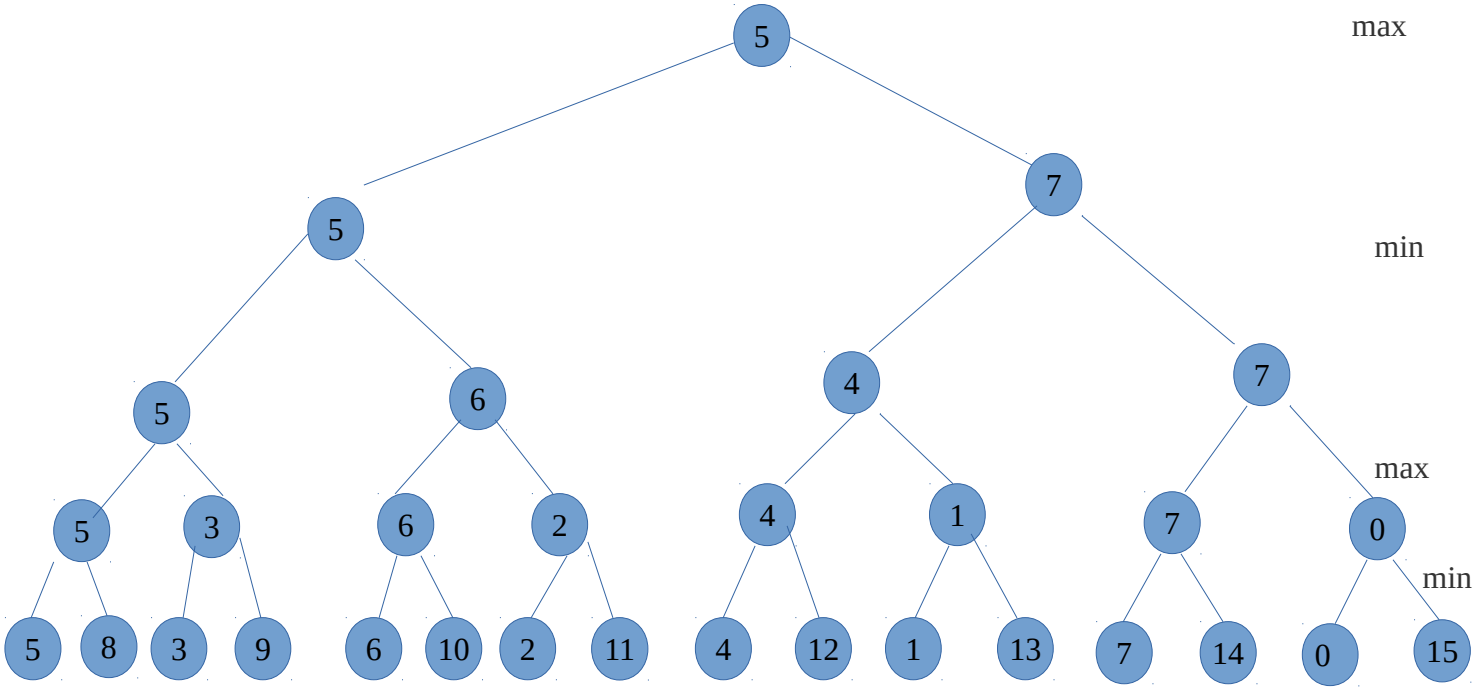
h(x,y,xg,yg) = |(yg-y) |+ |(xg-x)|

b)
(i) No, the straigh-line distance heuristic will not be admissible since it over-estimates the cost of diagnally moves.(1.414 moves compare to 1)
(ii)No, for the same reason above.(2 moves compare to 1)
(iii)

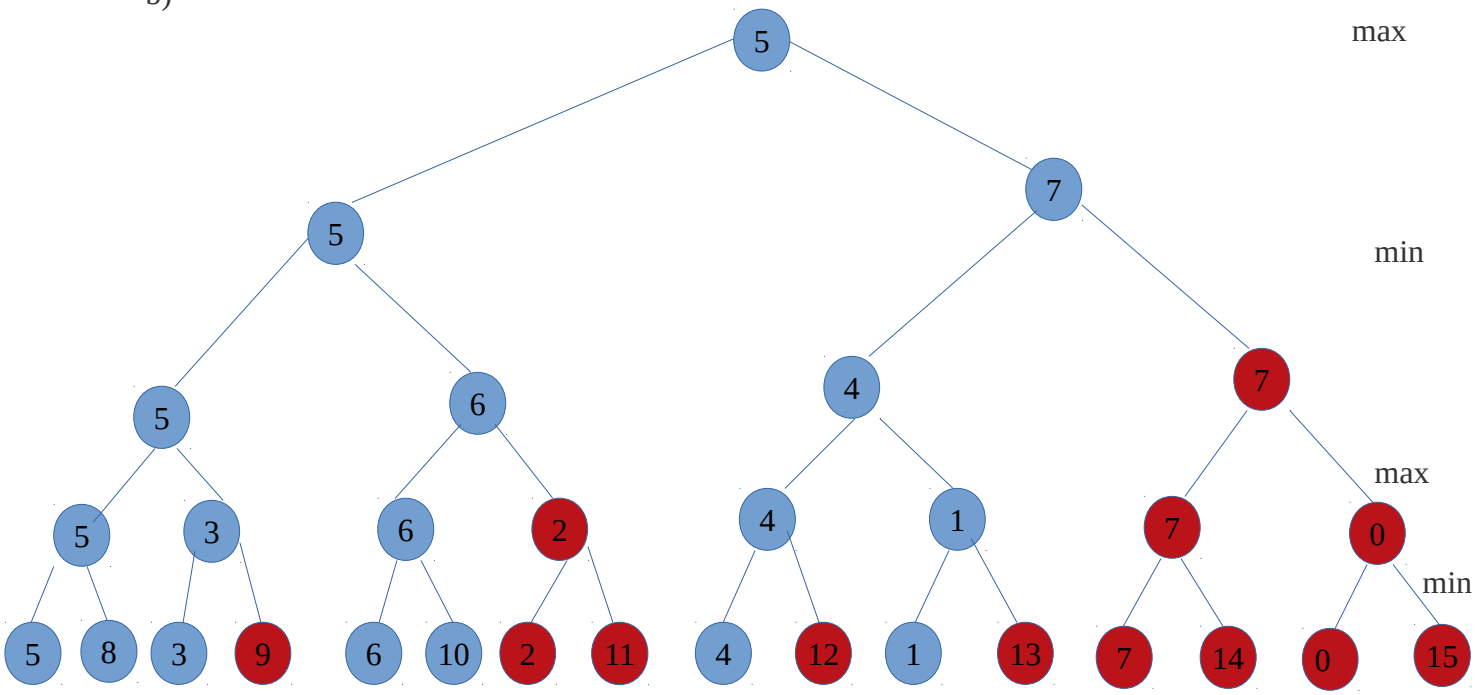
h(x,y,xg,yg) = min(|(yg-y) |, |(xg-x)|) + max(|(yg-y) |, |(xg-x)|) - min(|(yg-y) |, |(xg-x)|)
= max(|(yg-y) |, |(xg-x)|)

Q5

a)

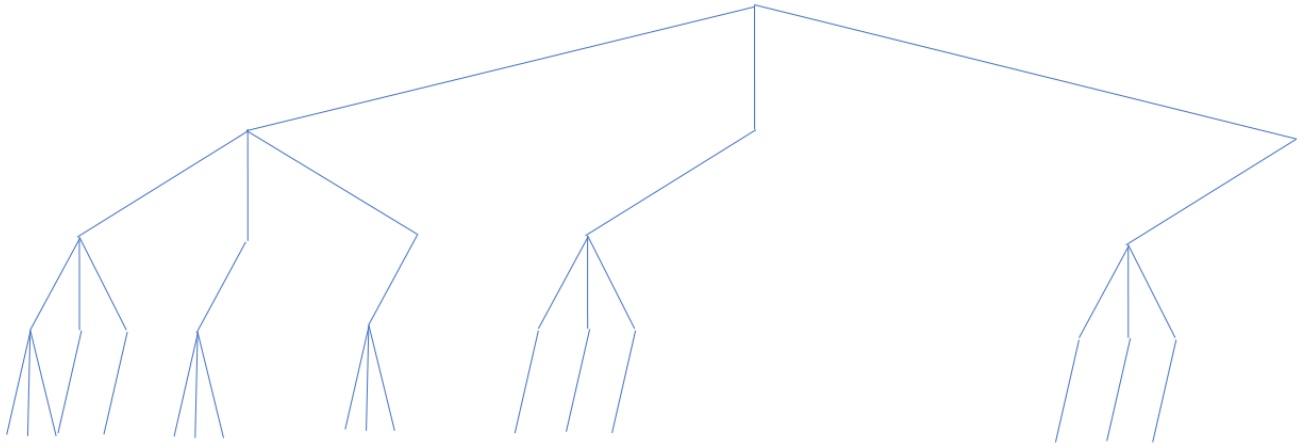

max
min
max
min
5
7
5
4
7
5
6
4
7
5
3
6
2
4
1
7
0
5
8
3
9
6
10
2
11
4
12
1
13
7
14
0
15

b)


max
min
max
min
5
7
5
4
7
5
6
4
7
5
3
6
2
4
1
7
0
5
8
3
9
6
10
2
11
4
12
1
13
7
14
0
15

c)

there are 17 out of 81 are expanded, the rest are pruned.



d)
b = branching factor
d = depth

O(b^(d/2))

since it changes from O(b*b*b*b..)= b ^ d to

Odd(b*1*b*1*b*1…) * Even(b*1*b*1*b*1…)
which is O(b^(d/2))