Assignment 1 COMP3411
z5311886, Joshua Fish, Assignment 1 COMP3411

Question 1: Search Algorithms for the 15-Puzzle

a) Draw up a table with four rows and five columns. Label the rows as UCS, IDS, A* and IDA* and the columns as start10, start12, start20, start30 and start40.
Run each of the following algorithms on each of the 5 start states: (i)[ucsdijkstra] (ii)[ideepsearch] (iii)[astar] (iv)[idastar]

Figure 1:

	Start10	Start12	Start30	Start40
UCS	2565	Mem	Mem	Mem
IDS	2407	13812	Time	Time
A*	33	26	Mem	Mem
IDA*	29	21	17297	112571

b) Briefly discuss the efficiency of these four algorithms (including both time and memory usage).

From figure 1, it is certain that uniform searches being UCS and IDS are slower and more inefficient algorithms for the 15-puzzle problem compared to informed searches of A* and IDA*. This is because uniform search algorithms expand a larger number of nodes compared to informed search algorithms and this causes larger computational time and memory usage. This can be seen within both algorithms time and space complexities with UCS having a time complexity and space complexity of $O(b^d)$, while IDS has the same time complexity, but a more efficient space complexity of $O(bd)$. The 'b' value refers to the branching factor and the 'd' value refers to the depth of the search, hence as the depth of the search becomes larger a larger number of nodes expand causing an exponentially longer time for both algorithms. However, the informed searches of A* (similar to UCS) and IDS* (similar to IDS) both use a heuristic function that enables the algorithm to have information on the goal state and allows for these algorithms to expand less nodes allowing less computations, while still producing the optimal solution. The A* algorithm can be seen having its memory limit exceeded with higher inputs and this is because it stores the entire level of its states within its memory. Whereas the IDS* algorithm uses a depth-first search allowing for lower memory usage than A*. However, for both of these informed searches their time and space complexity vary depending on the performance of the heuristic function with a better heuristic function substantially reducing time and memory costs.

key: b -> branching factor, d -> depth

Algorithms	Time Complexity	Space Complexity
UCS	$O(b^d)$	$O(b^d)$
IDS	$O(b^d)$	$O(bd)$
A*	Worst case $O(b^d)$	Worst case $O(b^d)$
IDA*	Worst case $O(b^d)$	Worst case $O(bd)$

Question 2) Heuristic Path Search for 15-Puzzle

(a) Run [greedy] for start50, start60 and start64, and record the values returned for G and N in the last row of your table (using the Manhattan Distance heuristic defined in puzzle15.pl).

	start50		start60		start64	
IDA∗	50	14642512	60	321252368	64	1209086782
1.2	52	191438	62	230861	66	431033
1.4	66	116342	82	4432	94	190278
1.6	100	33504	148	55626	162	235848
GREEDY	164	5447	166	1617	184	2174

(b) Now copy idastar.pl to a new file heuristic.pl and modify the code of this new file so that it uses an Iterative Deepening version of the Heuristic Path Search algorithm discussed in the Weak 2 Tutorial Exercise, with w = 1.2. In your submitted document, briefly show the section of code that was changed, and the replacement code.

Code for heuristic.pl:
with original code commented out and substituted for replacement code

```
% Otherwise, use Prolog backtracking to explore all successors
% of the current node, in the order returned by s.
% Keep searching until goal is found, or F_limit is exceeded.
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),
    % Code before the change was F1 is G1 + H1
    F1 is 0.8 * G1 + 1.2 * H1, % Changing the value of w to 1.2 as (2-w) g(n) + w * h(n)
    F1 =< F_limit,
    depthlim([Node|Path], Node1, G1, F_limit, Sol, G2).
```

(c) Run [heuristic] on start50, start60 and start64 and record the values of G and N in your table. Now modify your code so that the value of w is 1.4, 1.6 ; in each case, run the algorithm on the same three start states and record the values of G and N in your table.

(d) Briefly discuss the trade-off between speed and quality of solution for these five algorithms.

The trade-off between speed and quality of the solution for these five algorithms is dependent on the weighting of the heuristic. When the code had originally zero weighting where the function for the heuristic f(n) = g(n) + h(n) the program produced the optimal solution to the problem. This is shown with at start64 the algorithm only making 64 moves as G = 64, but a lot more nodes expanded with N = 1209086782. However, this was a lot slower than when the weighting for the heuristic changed with ‘w’ equalling to 1.2, 1.4 and 1.6. When ‘w’ was equal to these values it enabled the program to be a lot faster with a lower number of nodes being expanded, but not an optimal solution as it was an overestimate. At start64 the algorithm for when w = 1.6 shows this overestimate with a total of 162 moves as G = 162, while the number of nodes expanded is a lot less with only N = 235848. This trade-off between speed and quality is really dependent on the situation that the algorithm is used in and for some cases the optimal solution will be needed to solve a problem, but for others a close solution with a faster output is better and may find solutions that were thought not possible.