## Question 1: Search Algorithms for the 15-Puzzle (4 marks)

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.

| | Start10 | Start12 | Start20 | Start30 | Start40 |
|---|---|---|---|---|---|
| UCS | 2565 | Mem | Mem | Mem | Mem |
| IDS | 2407 | 13812 | 5297410 | Time | Time |
| A* | 33 | 26 | 915 | Mem | Mem |
| IDA* | 29 | 21 | 952 | 17297 | 112571 |

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

The UCS expands the lowest cost node and the cost is increasing along each path. The time usage efficiency is the lowest compared to the other three as well as the memory usage efficiency, for it runs out of memory from start12.

The IDS is better than the UCS since it successfully solved start10, start12 and start20. However, in start20, the IDS memory usage is over 5 million nodes. Further, it runs out of time from start30. Therefore, the it has a low time usage efficiency (O(b^d)).

The performance of A* is way better than the previous two algorithms, for it has significant improvement in memory usage in start10, start12, and start20. A* combines the advantages of the UCS and Greedy Search, for it keeps the efficiency of Greedy Search but avoid expanding the nodes that have already expanded. But from start30, it runs out of memory. Hence, A* is better than the UCS in memory usage, but it still not enough to solve complex scenarios.

The IDA* is the best in these 4 algorithms since it has the combination of A* and IDS's advantages. Moreover, the IDA* is a low-memory variant, so it is admissible and the fastest.

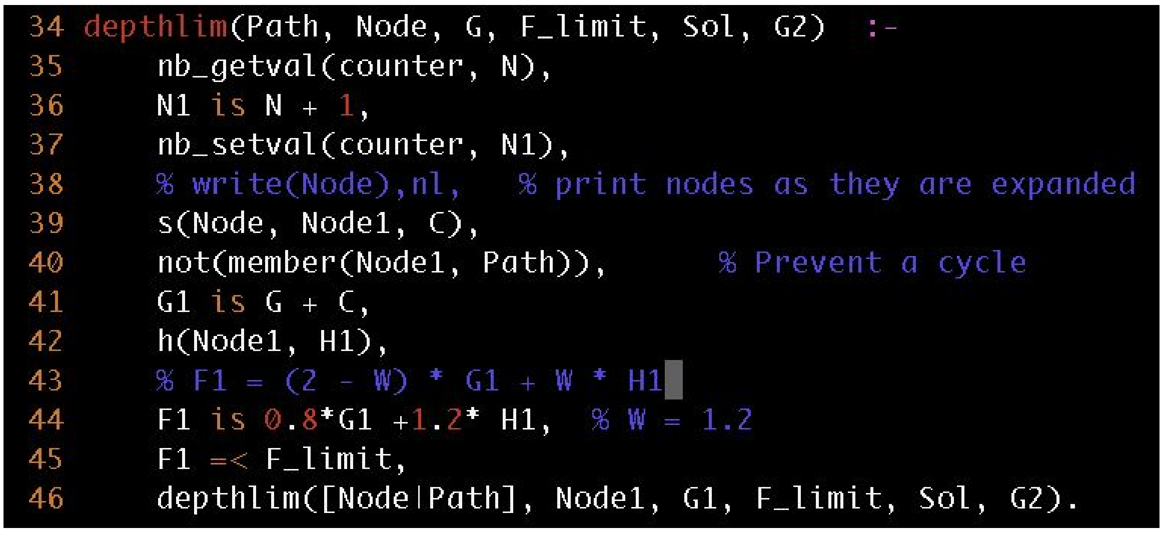
## Question 2: Heuristic Path Search for 15-Puzzle (4 marks)

a) Run [greedy] for start50, start60 and start64, and record theRun [greedy] for start50, start60 and start64, and record the values 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 |
| 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 Week 2 Tutorial Exercise (Module 4 - Tutorial: Heuristic Path Search) with w = 1.2 .
In your submitted document, briefly show the section of code that was changed, and the replacement code.

$$f(n) = (2-w)g(n) + w\,h(n), \quad \text{where } 0 \leq w \leq 2.$$



```
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 = (2 - W) * G1 + W * H1
    F1 is 0.8*G1 +1.2* H1,  % W = 1.2
    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.

| | 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 | 190287 |
| 1.6 | 100 | 33504 | 148 | 55626 | 162 | 235848 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Greedy | 164 | 5447 | 166 | 1617 | 184 | 2174 |

d) Briefly discuss the tradeoff between speed and quality of solution for these five algorithms.

IDA* generates the most nodes so it’s the least efficient, but the solution is the optimal. In contrast, greedy search expands the least cost node so it’s the most efficient, but it takes the most steps. We can consider IDA* is w = 1 and greedy search is w = 2, and the other three heuristic approaches are w between 1 and 2. As ‘w’ increasing, the speed will increase since the expanded nodes decrease. And since the length of generated path increased, the quality decreases. Therefore, the faster it runs, the worse the solution.

## Question 3: Maze Search Heuristics (4 marks)

Consider the problem of an agent moving around in a 2-dimensional maze, trying to get from its current position (x, y) to the Goal position ($x_G$, $y_G$) in as few moves as possible, avoiding obstacles along the way.

a) The heuristic is called Manhattan Distance heuristic. The formula is $h(x, y, x_G, y_G) = |x - x_G| + |y - y_G|$

b) Now assume that at each time step, the agent can take one step either up, down, left, right or diagonally. When it moves **diagonally**, it travels to the centre of a diagonally neighboring grid square, but a diagonal step is still considered to have the same “cost” (i.e. one “move”) as a horizontal or vertical step (like a King move in Chess).

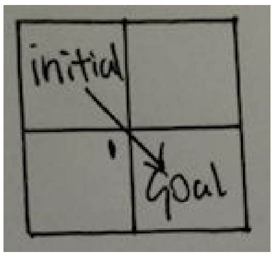


Figure 1

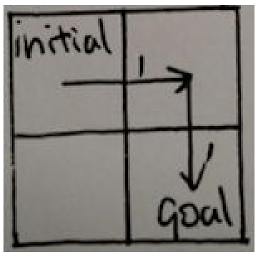


Figure 2

i) Assuming that the cost of a path is the total number of moves to reach the goal, is the Straight-Line-Distance heuristic still admissible? Explain why.

The Straight-Line-Distance heuristic is not admissible since it overestimate the optimal cost. For instance, as the pictures shown above, from initial to goal, the cost calculated by SLD is sqrt(2), however, the optimal solution is 1 (Figure 1)

ii) Is your heuristic from part(a) still admissible? Explain why.

No, the cost calculated by Manhattan Distance heuristic is 2 (Figure 2), however, the optimal solution is 1 (Figure 1).

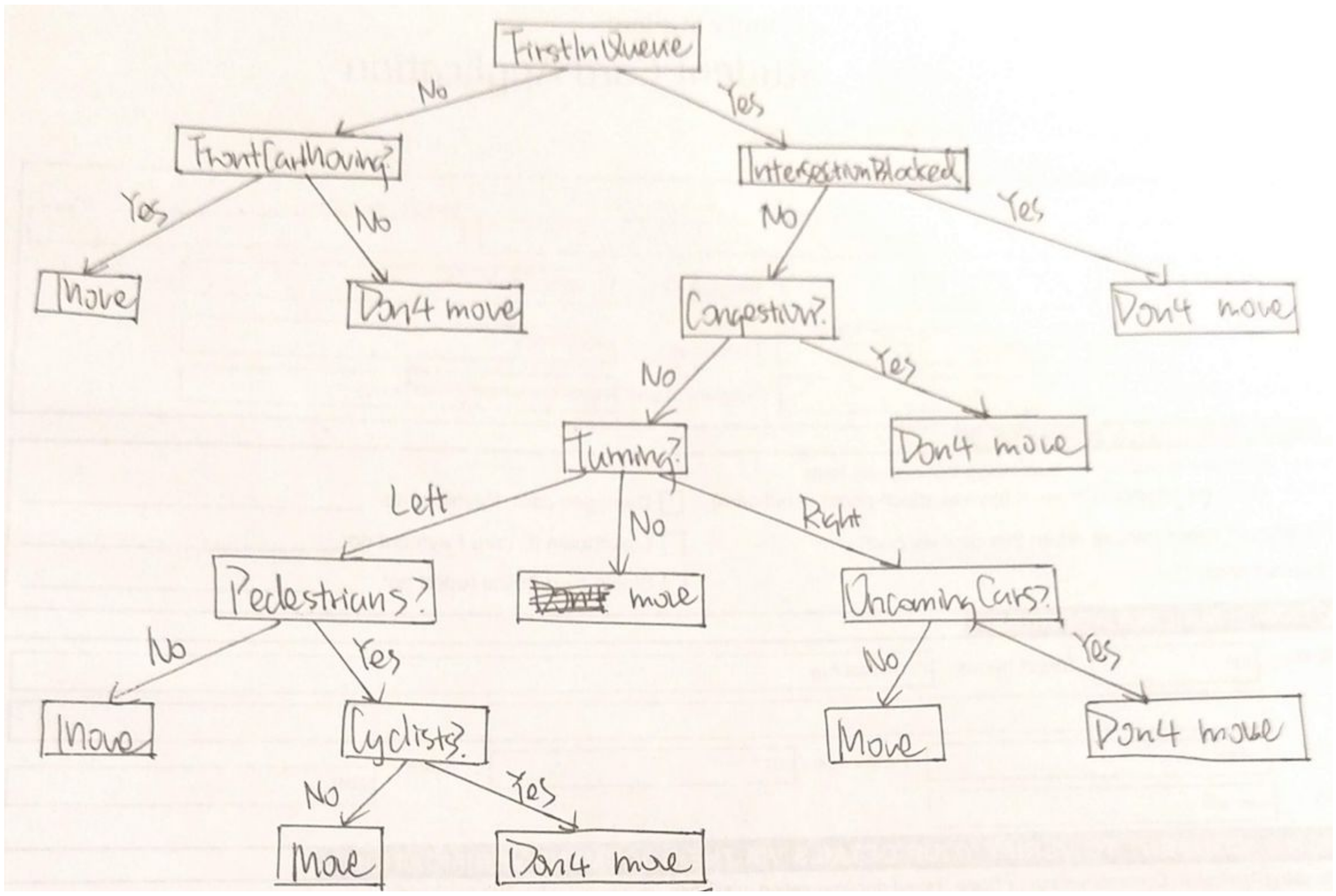
iii) Try to devise the best admissible heuristic you can for this problem, and write a formula for it in the format:

$h(x, y, x_G, y_G) = \max(|x - x_G|, |y - y_G|)$

## Question 4:Decision trees (4 marks)

In this question you need to draw a decision tree for the problem of deciding whether to move forward at a road intersection, given that the light has just turned green. (Hint: think of real-world situations, e.g. pedestrians, cyclists, other cars …).

Briefly explain what you have presented with your decision tree. Can this knowledge be used by an agent? If yes, explain why and if no, why not.



- First In Queue ⇔ If the car is the first in the queue
- Front Car Moving ⇔ if the car in front of it is moving
- Intersection Blocked ⇔ if the intersection is blocked
- Congestion ⇔ if there's traffic congestion
- Turning ⇔ if the car want to turn left or right or go straight
- Pedestrians ⇔ if there are pedestrians
- Cyclists ⇔ if there are cyclists

Well, it can be used by an agent. However, it can't be used in realistic since it didn't cover the scenarios that drivers or pedestrians or cyclists disobey the traffic rules. It only works when everyone carefully follow the rules which is impossible in the real world.

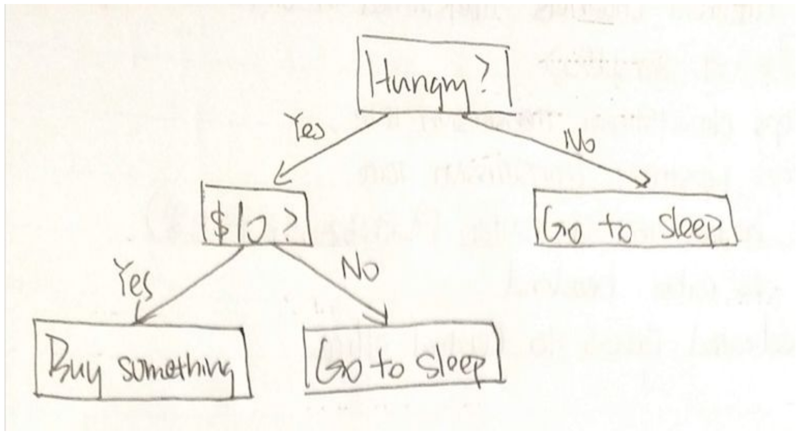
## Question 5:Decision trees (4 marks)

Suppose we generate a training set from a decision tree and then apply decision-tree learning to that training set.

a) Show working examples with 2 attributes, with 3 attributes and with 4 attributes.

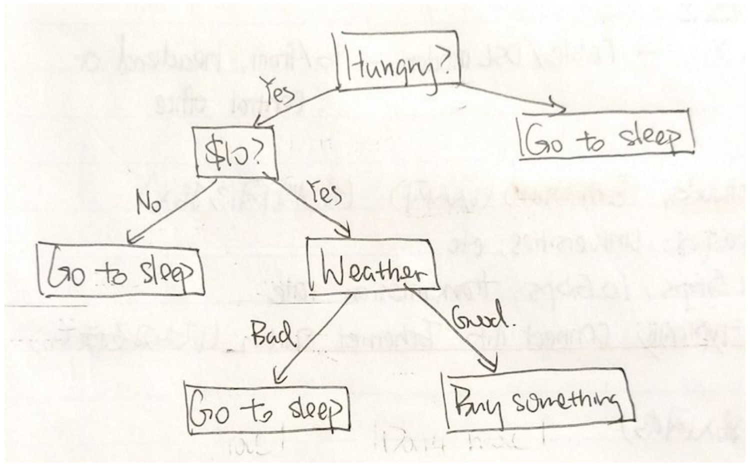
Scenario: whether go out to buy something to eat.

**2 attributes:**



| Example | Input Attributes | | Goal |
|---|---|---|---|
| | Hungry | $10 | Buy something to eat |
| X1 | Yes | Yes | Yes |
| X2 | Yes | No | No |
| X3 | No | Yes | No |
| X4 | No | No | No |

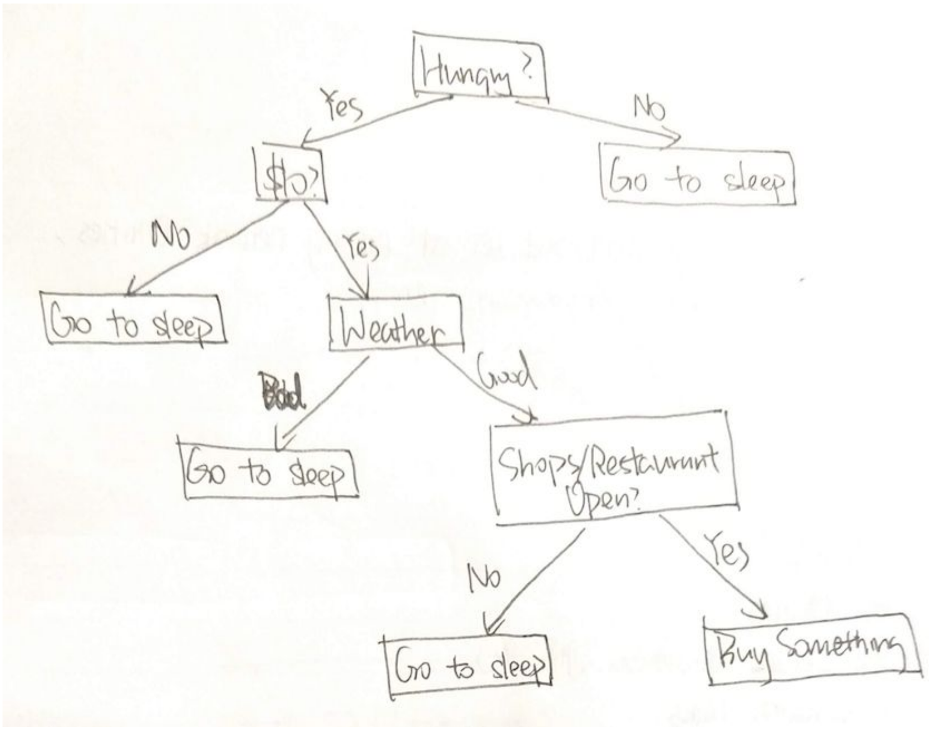
**3 attributes:**



| Example | Input Attributes | Goal |
|---|---|---|

| | Hungry | $10 | Weather Good? | Buy something to eat |
|---|---|---|---|---|
| X1 | Yes | Yes | Yes | Yes |
| X2 | Yes | Yes | No | No |
| X3 | Yes | No | Yes | No |
| X4 | Yes | No | No | No |
| X5 | No | Yes | Yes | No |
| X6 | No | Yes | No | No |
| X7 | No | No | Yes | No |
| X8 | No | No | No | No |

## 4 attributes:



| Example | Input Attributes | | | | Goal |
|---|---|---|---|---|---|
| | Hungry | $10 | Weather Good? | Shops/Restaurant open? | Buy something to eat |
| X1 | Yes | Yes | Yes | Yes | Yes |
| X2 | Yes | Yes | Yes | No | No |
| X3 | Yes | Yes | No | Yes | No |
| X4 | Yes | Yes | No | No | No |

| | | | | | |
|---|---|---|---|---|---|
| X5 | Yes | No | Yes | Yes | No |
| X6 | Yes | No | Yes | No | No |
| X7 | Yes | No | No | Yes | No |
| X8 | Yes | No | No | No | No |
| X9 | No | Yes | Yes | Yes | No |
| X10 | No | Yes | Yes | No | No |

b) Is it the case that the learning algorithm will eventually return the correct tree as the training-set size goes to infinity? Why or why not?

Well, the learning algorithm will eventually learn the tree but may not return the correct tree when the training set generates all possible combination of input attributes.
The possible functional space is huge and difficult to handle. It's nearly impossible to find the exact same tree. If the method randomly selects the value of each attribute, when the training-set size goes to infinity, the possibility of generating all combinations goes to one. Hence, there is a chance to find a logically equivalent function.