ITO5047 – Modules 1 and 2 Assignment

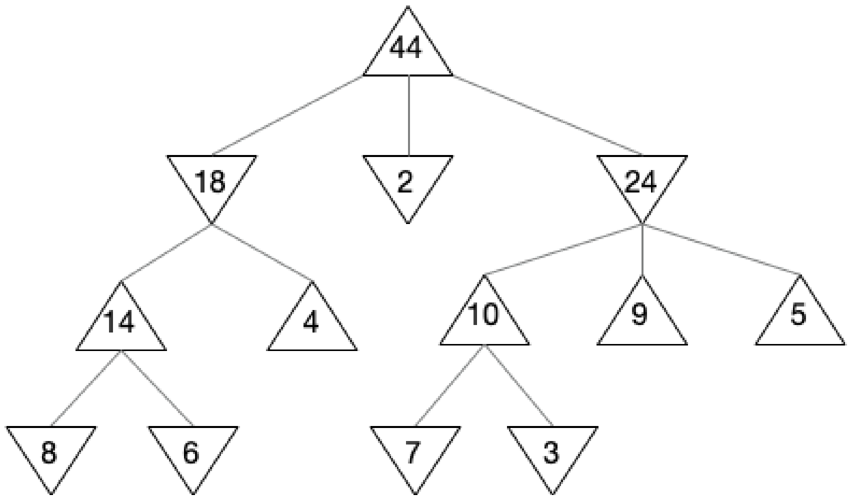
Question 1: Task environment

Consider the task of marathon running, and complete the following:

(a) Use a PEAS description to specify the task environment.
(b) Specify three properties/characteristics of the environment. Explain these properties in the context of the task.
(c) Determine which of the following agent types you would employ and explain why: Simple-reflex, Model-based, Goal-based, or Utility-based.

Question 2: Uninformed search

Use the following (fully expanded) search tree to indicate the order in which nodes are expanded for different types of search algorithms. Assume that '44' is the start node and '3' is the only goal node.



(a) Depth First Search (DFS).
List the nodes according to their order of expansion. Assume that rightmost nodes are expanded first. Moreover, list the nodes in the final search tree—that is, without the nodes deleted by the algorithm.

(b) Breadth First Search (BFS).
List the nodes according to their order of expansion. Assume that rightmost nodes are expanded first.

Question 3: Informed search

Consider a grid of size 3 by 3 that represents a map. In each tile of the grid, there is a letter representing a kind of terrain. There are two types of terrains in the map: A and B. Currently, a robot is at the top right corner. We use the “$*$” symbol to represent the current location of the robot.

	1	2	3
1	A	B	A*
2	A	A	A
3	B	B	B

Assume that the X axis is the horizontal direction while the Y axis is the vertical direction—so the current location of the robot is (3, 1). The robot needs to do some tasks at position (1, 3), depicted in red. It takes 1 minute for the robot to move from one tile to another. However, when the robot is moving from one type of terrain to another type, it has to change its wheels before the moving, which takes two extra minutes. The robot can only move horizontally or vertically (thus, the robot cannot move diagonally).

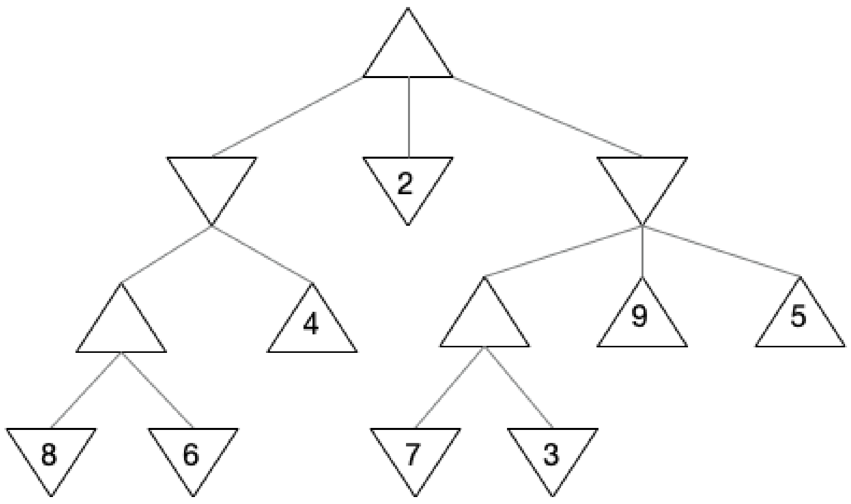
Using the A* algorithm, suppose that we want to find a path that takes the least time to make the robot move from position $(3, 1)$ to position $(1, 3)$. Based on this information:

(a) Greedy best-first search.
Propose an admissible heuristic function for this problem and use it to find a path for the robot using the Greedy first-search algorithm. Draw the search tree generated by the algorithm using your heuristic function, indicating the values of f and h for each node.

(b) Algorithm A*.
Using the heuristic function defiend in part (a), perform the A* algorithm to search for the path and draw the search tree. In such a search tree, indicate the values of g, h and f, and use Roman numbers to indicate the order in which nodes are expanded.

Question 4: Adversarial search

The figure below shows a game tree in which leaf nodes have the stated utility values. Nodes depicted with △ belong to player MAX, while nodes depicted with ▽ belong to MIN.



(a) Minimax algorithm.
Using Minimax, display the computed Minimax values for all nodes in the game tree.

(b) Alpha-Beta pruning.
Perform the α−β search algorithm assuming that the rightmost nodes are expanded first. Then, cross out the branches that will be pruned by the algorithm and in each case indicate if it is an α cut-off or a β cut-off. Also, for all non-leaf nodes in the search tree generated by the α−β search algorithm, display the sequence of α or β values computed by the algorithm at that node (with leftmost values in the sequence having been computed first).

Question 5: Implementation

In this task you will write a program in Python that implements graph search. Your program will take as input a graph G, a matrix of heuristic estimates H, a search method X, a start state S, and a goal state T , and using this information, find a path from S to T . The algorithm should allow the user to use any of the following four search algorithms:

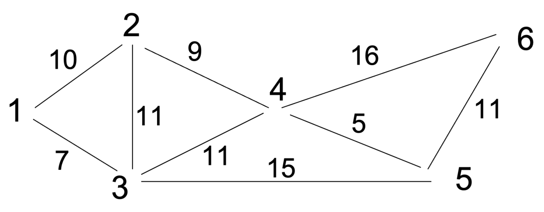
(a) BFS

(b) DFS

(c) Greedy best-first search

(d) A*

HINT: Think about how you can write one version of graph search and simply change the way nodes are added to and removed from the FRONTIER to implement each search strategy. The graph G will be given as an adjacency matrix where the value at G[i][j] represents the cost of getting from node i to node j. If there is no edge between nodes i and j then the G[i][j] value will be None. Assume G is always an undirected graph. For example, the graph on the left would have the adjacency matrix on the right:



	1	2	3	4	5	6
1		10	7			
2	10		11	9		
3	7	11		11	15	
4		9	11		5	16
5			15	5		11
6				16	11	

The matrix H will contain heuristic estimates for the distance between all pairs of vertices in the graph. So H[i][j] will contain a heuristic estimate for going from vertex i to vertex j. From this, given a goal state g, you can construct a heuristic function h that takes a node n and returns a real number greater than or equal to 0 estimating the minimum cost to get from n to g. Remember, if you are implementing A* then your heuristic must be admissible! If it is not then you are implementing A. It need not be consistent but using a consistent heuristic will make things easier for yourself.

Both G and H will be given as csv files. As an example, the adjacency matrix above for the graph G on the left would be represented as:

```
,10,7,,,
10,,11,9,,
7,11,,11,15,
,9,15,,5,16
,,,5,,11
,,,16,11,
```

Likewise, e.g., the (constant) admissible heuristic for all nodes would be represented with the csv file shown below:

```
0,0,0,0,0,0
0,0,0,0,0,0
0,0,0,0,0,0
0,0,0,0,0,0
0,0,0,0,0,0
0,0,0,0,0,0
```

You should write your code in the Python file `a1_q5.py` provided. You should write your graph search code in the `graph_search` function and any other code you need should be written below that. The `a1_q5.py` file has code for getting all the required inputs from the command line, and for creating a heuristic function from H and the specified goal state. You can use the adjacency matrix representation in your graph search function or create whatever kind of graph data structure you would like to use.

To run your Python program from a command line, you should write:

```
python a1_q5.py S T X graph.csv h.csv
```

where we have in the example graph given:

- $\mathtt{S} \in \{1, 2, 3, 4, 5, 6\}$, for the start node.
- $\mathtt{T} \in \{1, 2, 3, 4, 5, 6\}$, for the target node.
- $\mathtt{X} \in \{\mathtt{B}, \mathtt{D}, \mathtt{G}, \mathtt{A}\}$, for algorithm BFS, DFS, GREEDY, and A (star), respectively;
- `graph.csv` for the input graph in csv format; and
- `h.csv` for the heuristic matrix, also in csv format.

The program should then output the path obtained using the specified algorithm X from start node S and target node T. To order nodes in the frontier, use minimum cost with respect to:

- *f(i) = depth(i)*, for BFS and DFS, with BFS prioritising smallest depth and DFS prioritising greatest depth;
- *f(i) = h(i),* for Greedy best-first search; and
- *f(i) = g(i) + h(i)*, for A (star).

Finally, assume that nodes in the graph are associated with natural numbers. So, 1,2,3, and so on and so forth.

Example.
So, if we write:

```
python a1_q5.py 1 5 B graph.csv h.csv
```

to find a path between nodes 1 and 5 using BFS, we would get the path/output:

```
1 3 5
```

representing the path $1 \rightarrow 3 \rightarrow 5$ in the graph.