Artificial Intelligence – Agent - based and Adversarial Search Report

Background:

Agent-based Search:

Being a computational model to simulate the interactions of autonomous agents along with its actions, agent based modelling is used to gain better understanding of the behaviours of a system and reason its outcomes.

Most agent-based models are composed of:
(1) numerous agents specified at various scales (typically referred to as agent-granularity);
(2) decision-making heuristics;
(3) learning rules or adaptive processes;
(4) an interaction topology; and
(5) an environment.

Uninformed Search Algorithms
Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search. There are types as below:

- Depth – First Search (DFS):

Depth first search is an algorithm to traverse or search tree or graph data structures. Beginning at the root node (selection of arbitrary nodes as the root nodes of the graph) it keeps on exploring as far as possible along each branch until it has to backtrack.

- Breadth – First Search (BFS):

Even Breadth first search algorithm is used to traverse or search tree or graph or data structures. Starting at the root node, it explores all the neighbour nodes at present depth level before proceeding for another node at next level.

- Uniform Cost Search (UCS):

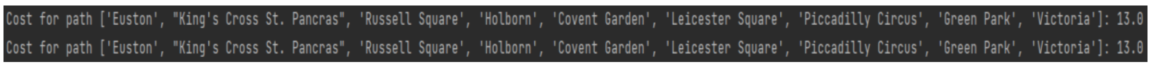
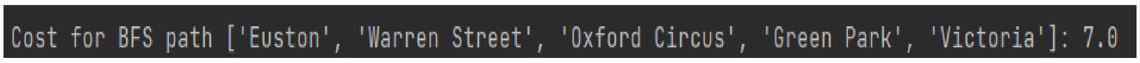
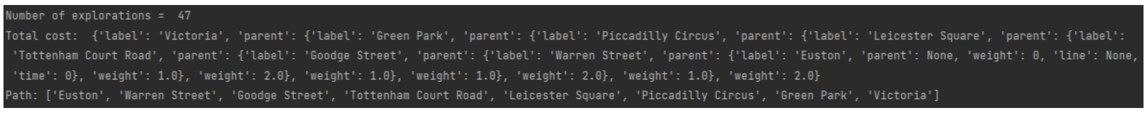
  Different than DFS and BFS, Uniformed cost search let' s us know that traversing through different edges could have different costs. Its goal is to check for a path which has the least cumulative costs.

The following has been demonstrated in the Q 2.1 stated below.

2.1 Implement DFS, BFS, and USC
Implement DFS, BFS, and USC to find routes from a starting station to a destination station. For a path found, your program should print/display meaningful information (e.g. stations, cost, number of node expanded). The route from Euston to Victoria is a good one to test on. Describe in the report how your program works.
You firstly need to think about how to represent a state and how to construct a new state from each

station reachable from the one in the current state but not already visited in the current path so far (hint: if you are unsure, think about the common features of DFS, BFS, and USC).
This question carries 25% of the marks for this coursework.

Stations considered: Euston to Victoria

Type of search	Output
Depth-first search	Along with path it has also computed cost.  `Cost for path ['Euston', "King's Cross St. Pancras", 'Russell Square', 'Holborn', 'Covent Garden', 'Leicester Square', 'Piccadilly Circus', 'Green Park', 'Victoria']: 13.0` `Cost for path ['Euston', "King's Cross St. Pancras", 'Russell Square', 'Holborn', 'Covent Garden', 'Leicester Square', 'Piccadilly Circus', 'Green Park', 'Victoria']: 13.0`
Breadth-first search	Along with path it also showed the cost which is being computed.  `Cost for BFS path ['Euston', 'Warren Street', 'Oxford Circus', 'Green Park', 'Victoria']: 7.0`
Uniformed cost search	Along with path it also showed the cost which is being computed.  `Number of explorations =  47` `Total cost: {'label': 'Victoria', 'parent': {'label': 'Green Park', 'parent': {'label': 'Piccadilly Circus', 'parent': {'label': 'Leicester Square', 'parent': {'label': 'Tottenham Court Road', 'parent': {'label': 'Goodge Street', 'parent': {'label': 'Warren Street', 'parent': {'label': 'Euston', 'parent': None, 'weight': 0, 'line': None, 'time': 0}, 'weight': 1.0}, 'weight': 2.0}, 'weight': 1.0}, 'weight': 1.0}, 'weight': 2.0}, 'weight': 1.0}, 'weight': 2.0}` `Path: ['Euston', 'Warren Street', 'Goodge Street', 'Tottenham Court Road', 'Leicester Square', 'Piccadilly Circus', 'Green Park', 'Victoria']`

To implement each of the methods, there are three files created named as – UCS_implementation, BFS_implementation and DFS_implementation. In all them, it is divided into sections:

- Function to collect data from CSV file:
  - Def data() / def tube_data()

- Function for the displaying o the graph:
  - def show_weighted_graph(networkx_graph, node_size, font_size, fig_size):

- Function to compute cost:
  - def compute_path_cost(graph, path):

The functions for collection of data , displaying of graph and cost computation are kept in the same format for all of them and have major adaptations from the lab sessions Lab03 and Lab04.

- Next in each file, there are 3 different functions for the implementations of each algorithms, named as:
  - Function for DFS algorithm:
    def my_depth_first_graph_search(nxobject, initial, goal, compute_exploration_cost=False, reverse=False):

  - Function for BFS algorithm:
    def bfs_implementation(graph, origin, destination, counter = 0, reverse=False):

The major adaptations of DFS and BFS algorithms have been taken from the Lab03 sessions.

  - Implementing of UCS algorithm:
    def usc(nxobject, initial, goal,): defined to implement the algorithm
    Priority_Q.py: To implement the Priority Queueing

2.2 Compare DFS, BFS, and USC

Now make a detailed comparison of the three search methods, and write a short analysis of what you find. Make sure you try out a good number of different routes, including the ones below, and, once you've done so, base your comparison on the relative costs, and the relative number of nodes expanded, of the different methods. Is any one method consistently the best? What are the advantages and disadvantages of the methods, in comparison one to another? How does the way the Tube Map is structured affect the outcomes? Write a short report (no more than 2 pages of 10 point font for this part) of what you find and explain what you have discovered.

Routes to include in your comparison:

- Canada Water to Stratford
- New Cross Gate to Stepney Green
- Ealing Broadway to South Kensington
- Baker Street to Wembley Park

This question carries 15% of the marks for this coursework.

Stations	Canada Water to Stratford
Depth-first search	Along with path it has also computed cost. Cost for path ['Canada Water', 'Canary Wharf', 'North Greenwich', 'Canning Town', 'West Ham', 'Stratford']: 15.0 Cost for path ['Canada Water', 'Canary Wharf', 'North Greenwich', 'Canning Town', 'West Ham', 'Stratford']: 15.0
Breadth-first search	Along with path it also showed the cost which is being computed. Cost for BFS path ['Canada Water', 'Canary Wharf', 'North Greenwich', 'Canning Town', 'West Ham', 'Stratford']: 15.0
Uniformed cost search	Along with path it also showed the cost which is being computed. Number of explorations = 181 Total cost: {'label': 'Stratford', 'parent': {'label': 'West Ham', 'parent': {'label': 'Bromley-by-Bow', 'parent': {'label': 'Bow Road', 'parent': {'label': 'Mile End', 'parent': {'label': 'Stepney Green', 'parent': {'label': 'Whitechapel', 'parent': {'label': 'Shadwell', 'parent': {'label': 'Wapping', 'parent': {'label': 'Rotherhithe', 'parent': {'label': 'Canada Water', 'parent': None, 'weight': 0, 'line': None, 'time': 0}, 'weight': 1.0}, 'weight': 1.0}, 'weight': 1.0}, 'weight': 2.0}, 'weight': 3.0}, 'weight': 2.0}, 'weight': 1.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 3.0} Path: ['Canada Water', 'Rotherhithe', 'Wapping', 'Shadwell', 'Whitechapel', 'Stepney Green', 'Mile End', 'Bow Road', 'Bromley-by-Bow', 'West Ham', 'Stratford']
Stations	New Cross Gate to Stepney Green
Depth-first search	Along with path it has also computed cost. Cost for path ['New Cross Gate', 'Surrey Quays', 'Canada Water', 'Canary Wharf', 'North Greenwich', 'Canning Town', 'West Ham', 'Stratford', 'Mile End', 'Stepney Green']: 27.0 Cost for path ['New Cross Gate', 'Surrey Quays', 'Canada Water', 'Canary Wharf', 'North Greenwich', 'Canning Town', 'West Ham', 'Stratford', 'Mile End', 'Stepney Green']: 27.0
Breadth-first search	Along with path it also showed the cost which is being computed. Cost for BFS path ['New Cross Gate', 'Surrey Quays', 'Canada Water', 'Rotherhithe', 'Wapping', 'Shadwell', 'Whitechapel', 'Stepney Green']: 14.0
Uniformed cost search	Along with path it also showed the cost which is being computed. Number of explorations = 128 Total cost: {'label': 'Stepney Green', 'parent': {'label': 'Whitechapel', 'parent': {'label': 'Shadwell', 'parent': {'label': 'Wapping', 'parent': {'label': 'Rotherhithe', 'parent': {'label': 'Canada Water', 'parent': {'label': 'Surrey Quays', 'parent': {'label': 'New Cross Gate', 'parent': None, 'weight': 0, 'line': None, 'time': 0}, 'weight': 4.0}, 'weight': 2.0}, 'weight': 1.0}, 'weight': 1.0}, 'weight': 1.0}, 'weight': 2.0}, 'weight': 3.0} Path: ['New Cross Gate', 'Surrey Quays', 'Canada Water', 'Rotherhithe', 'Wapping', 'Shadwell', 'Whitechapel', 'Stepney Green']
Stations	Ealing Broadway to South Kensington
Depth-first search	Along with path it has also computed cost. Cost for path ['Ealing Broadway', 'Ealing Common', 'North Ealing', 'Park Royal', 'Alperton', 'Sudbury Town', 'Sudbury Hill', 'South Harrow', 'Rayners Lane', 'West Harrow', 'Harrow-on-the-Hill', 'Northwick Park', 'Preston Road', 'Wembley Park', 'Finchley Road', 'Baker Street', 'Bond Street', 'Green Park', 'Victoria', 'Sloane Square', 'South Kensington']: 57.0 Cost for path ['Ealing Broadway', 'Ealing Common', 'North Ealing', 'Park Royal', 'Alperton', 'Sudbury Town', 'Sudbury Hill', 'South Harrow', 'Rayners Lane', 'West Harrow', 'Harrow-on-the-Hill', 'Northwick Park', 'Preston Road', 'Wembley Park', 'Finchley Road', 'Baker Street', 'Bond Street', 'Green Park', 'Victoria', 'Sloane Square', 'South Kensington']: 57.0
Breadth-first search	Along with path it also showed the cost which is being computed.

	Cost for BFS path ['Ealing Broadway', 'Ealing Common', 'Acton Town', 'Turnham Green', 'Hammersmith', 'Barons Court', "Earls' Court", 'Gloucester Road', 'South Kensington']: 20.0
Uniformed cost search	Along with path it also showed the cost which is being computed. Number of explorations = 39 Total cost: {'label': 'South Kensington', 'parent': {'label': 'Gloucester Road', 'parent': {'label': "Earls' Court", 'parent': {'label': 'West Kensington', 'parent': {'label': 'Barons Court', 'parent': {'label': 'Hammersmith', 'parent': {'label': 'Goldhawk Road', 'parent': {'label': "Shepherd's Bush", 'parent': {'label': 'White City', 'parent': {'label': 'East Acton', 'parent': {'label': 'North Acton', 'parent': {'label': 'West Acton', 'parent': {'label': 'Ealing Broadway', 'parent': None, 'weight': 0, 'line': None, 'time': 0}, 'weight': 3.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 3.0}, 'weight': 3.0}, 'weight': 1.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 1.0} Path: ['Ealing Broadway', 'West Acton', 'North Acton', 'East Acton', 'White City', "Shepherd's Bush", 'Goldhawk Road', 'Hammersmith', 'Barons Court', 'West Kensington', "Earls' Court", 'Gloucester Road', 'South Kensington']
Stations	Baker Street to Wembley Park
Depth-first search	Along with path it has also computed cost. Cost for path ['Baker Street', 'Finchley Road', 'Wembley Park']: 13.0 Cost for path ['Baker Street', 'Finchley Road', 'Wembley Park']: 13.0
Breadth-first search	Along with path it also showed the cost which is being computed. Cost for BFS path ['Baker Street', 'Finchley Road', 'Wembley Park']: 13.0
Uniformed cost search	Along with path it also showed the cost which is being computed. Number of explorations = 261 Total cost: {'label': 'Wembley Park', 'parent': {'label': 'Preston Road', 'parent': {'label': 'Northwick Park', 'parent': {'label': 'Harrow-on-the-Hill', 'parent': {'label': 'West Harrow', 'parent': {'label': 'Rayners Lane', 'parent': {'label': 'South Harrow', 'parent': {'label': 'Sudbury Hill', 'parent': {'label': 'Sudbury Town', 'parent': {'label': 'Alperton', 'parent': {'label': 'Park Royal', 'parent': {'label': 'North Ealing', 'parent': {'label': 'Ealing Common', 'parent': {'label': 'Acton Town', 'parent': {'label': 'Chiswick Park', 'parent': {'label': 'Turnham Green', 'parent': {'label': 'Stamford Brook', 'parent': {'label': 'Ravenscourt Park', 'parent': {'label': 'Hammersmith', 'parent': {'label': 'Barons Court', 'parent': {'label': 'West Kensington', 'parent': {'label': "Earls' Court", 'parent': {'label': 'Gloucester Road', 'parent': {'label': 'South Kensington', 'parent': {'label': 'Sloane Square', 'parent': {'label': 'Victoria', 'parent': {'label': 'Green Park', 'parent': {'label': 'Bond Street', 'parent': {'label': 'Baker Street', 'parent': None, 'weight': 0, 'line': None, 'time': 0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 1.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 1.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 3.0}, 'weight': 2.0}, 'weight': 3.0}, 'weight': 2.0}, 'weight': 3.0}, 'weight': 2.0}, 'weight': 3.0}, 'weight': 3.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 2.0}, 'weight': 3.0} Path: ['Baker Street', 'Bond Street', 'Green Park', 'Victoria', 'Sloane Square', 'South Kensington', 'Gloucester Road', "Earls' Court", 'West Kensington', 'Barons Court', 'Hammersmith', 'Ravenscourt Park', 'Stamford Brook', 'Turnham Green', 'Chiswick Park', 'Acton Town', 'Ealing Common', 'North Ealing', 'Park Royal', 'Alperton', 'Sudbury Town', 'Sudbury Hill', 'South Harrow', 'Rayners Lane', 'West Harrow', 'Harrow-on-the-Hill', 'Northwick Park', 'Preston Road', 'Wembley Park']

It can be seen that at times UCS has not been much efficient as the path computed has exceeded to what the DFS and BFS versions had to offer. From the data, we can notice that using BFS to compute cost and path has been more effective. Not only did it provide shorter path lengths but also cost less than or equal to the ones computed by DFS.

2.3 Extending the cost function
Consider how you might improve the cost function over simply adding the time required between stations. Implement and use your new cost function in USC. Explain your new cost function and what you found in the report. This question carries 10% of the marks for this coursework.

Similar to DFS and BFS, the cost computation has been implemented in the UCS_implementation.py file.

Informed Search Algorithms:
The algorithms contains information on the goal state that benefits in more efficient searching. This information is obtained by something called a heuristic.

Heuristic: A heuristic h is defined as -
h(x) = Estimate of distance of node x from the goal node.
Lower the value of h(x), closer is the node from the goal.
Strategy: Expansion of the node that is closest to the goal state, i.e. expand the node with a lower h value.

The following has been demonstrated in the Q 2.4.

2.4 Heuristic search
Given that you know the zone(s) a station is in, consider how you might use this information to focus the search in the right direction and implement your heuristic search. You should explain in detail in the report your heuristic - if your implementation has gone astray, but your idea was good, you will get at least some of the credit.
This question carries 20% of the marks for this coursework.

As we know the heurestics take the node which has the lower value of h(x). So to implement it, there is mathematical modifications of reducing the weight by '2' in the def heuristic() function for computation.

<u>Adversarial Search:</u>

In adversarial search, the problem is analysed which can be caused when the agent is trying to plan ahead of the other agent or opponent (in case of a game). This search has one of its applications in game-playing where are two or more than two players. For the game playing, the agents are put in a competitive environment with conflicting goals to be achieved by them. The players always try to defeat one another in order to win. Thus giving rise to adversarial search.

The mathematical concept of this search is based on **'Game Theory'. According to game theory, a game is played between two players. To complete the game, one has to win the game and the other looses automatically.'**

Types of algorithms in adversarial search:

- <u>MinMax Algorithm:</u>

MinMax algorithm is a decision-making strategy which is used to minimize the loss chances and maximize the winning chances. Strategy of the game can be stated as follows:

- MIN: Decrease the chances of MAX to win the game.
- MAX: Increases his chances of winning the game.

MinMax algorithm follows the concept of Depth-First Search and where it backtracks by picking one of the best move out of several choices. It can be demonstrated in the example shown in 3.1 .

Pseudo code:

```
function minimax(position, depth, maximizingPlayer)
    if depth == 0 or game over in position
        return static evaluation of position
    if maximizingPlayer
        maxEval = -infinity
        for each child of position
            eval = minimax(child, depth - 1, false)
            maxEval = max(maxEval, eval)
        return maxEval
    else
        minEval = +infinity
        for each child of position
            eval = minimax(child, depth - 1, true)
            minEval = min(minEval, eval)
        return minEval
// initial call
minimax(currentPosition, 3, true)
```

3.1 Play optimally (MINIMAX algorithm)

Knowing that you start (top of the tree is you), complete the blank nodes in the provided tree using the MINIMAX approach. This question carries 10% of the marks for this coursework.

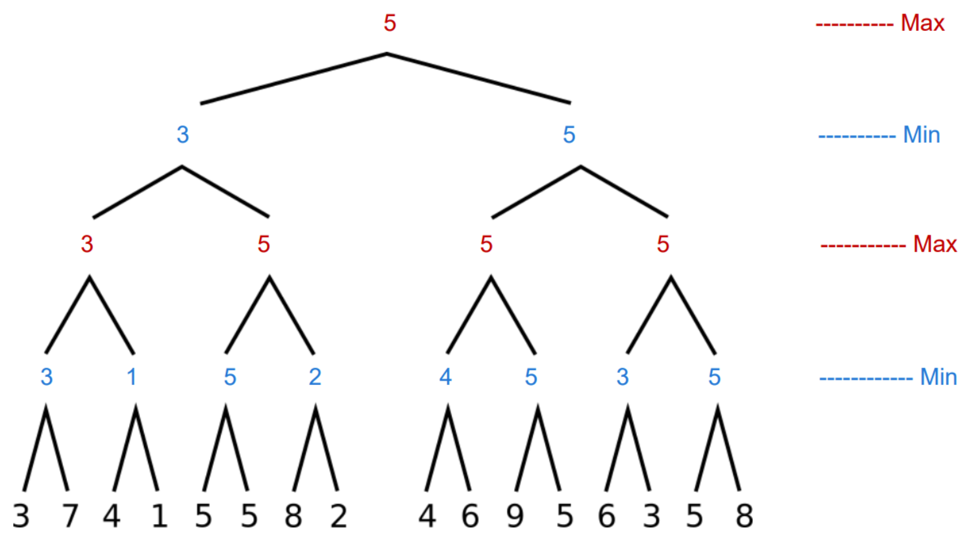


Figure 1: Tree of the game

- Alpha-Beta Pruning:

The Alpha-Beta pruning is the advanced version of MINMAX algorithm but with more time complexity. As it deeply explores each nodes in the tree to provide best path among all the available path. It can be demonstrated in the example shown in 3.2 .

Note: Alpha-beta pruning technique can be applied to trees of any depth, and it is possible to prune the entire subtrees easily. [3]

<u>Pseudo code:</u>

```
function minimax(position, depth, alpha, beta, maximizingPlayer)
    if depth == 0 or game over in position
        return static evaluation of position
    if maximizingPlayer
        maxEval = -infinity
        for each child of position
            eval = minimax(child, depth - 1, alpha, beta false)
            maxEval = max(maxEval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha
                break
        return maxEval
    else
        minEval = +infinity
        for each child of position
            eval = minimax(child, depth - 1, alpha, beta true)
            minEval = min(minEval, eval)
            beta = min(beta, eval)
            if beta <= alpha
                break
        return minEval
// initial call
minimax(currentPosition, 3, -∞, +∞, true)
```

3.2 Play optimally, quicker thinking! (α-β pruning)

Suppose there is a mean arbiter that doesn't like too much hesitation at the beginning. Perform α-β pruning on the tree you filled, searching from left to right.

Which branches are pruned and why?

The branches which does not affect the outcome are always prunes in the algorithm. In the figure, the pruned branch has been marked with a cross mark over it. As it can be seen that the $\alpha \geq \beta$, where $\alpha = 5$ and $\beta = 3$. Hence, it is pruned.

This question carries 10% of the marks for this coursework.

See Figure below

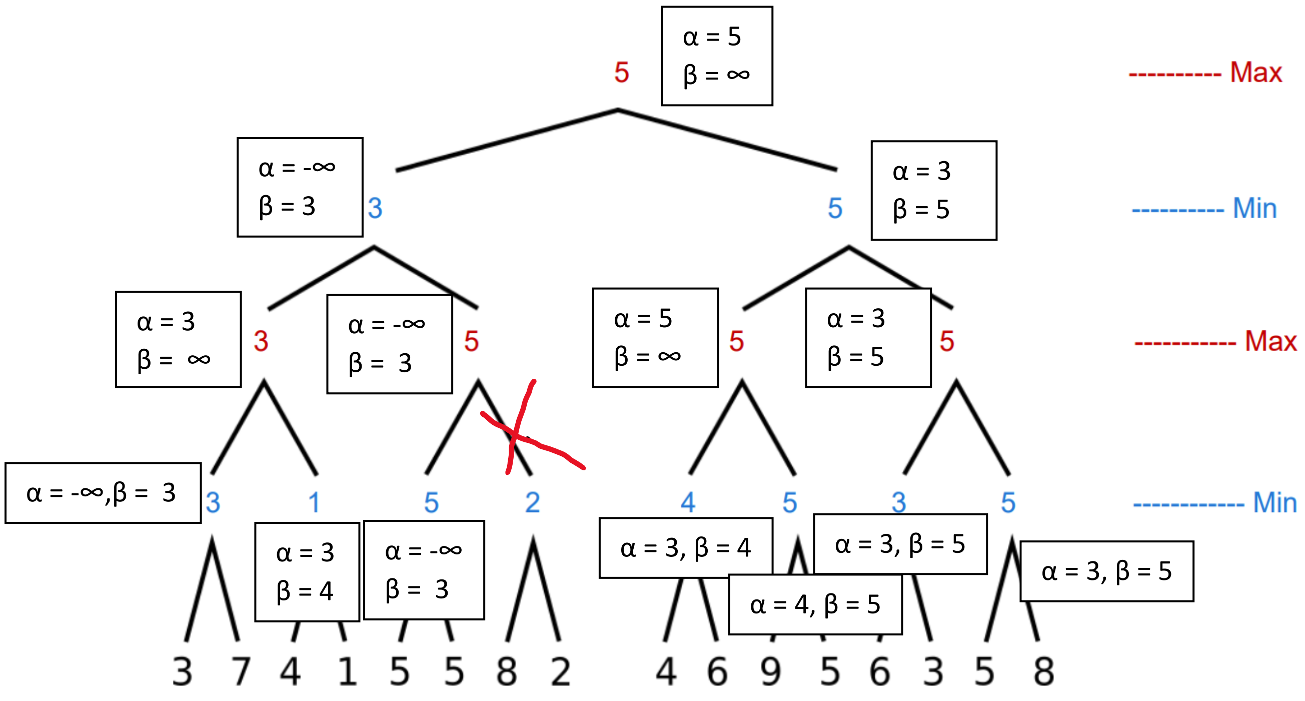


Figure 1: Tree of the game

3.3 Entry free to play

Suppose the entry fee to play the game is x ≥ 0 units. In particular, this is the money that the MAX player pays the MIN player to be allowed to start the game. The rest of the game is as before. So for instance, if the terminal state of the game is the left-most leaf in Figure 1, then the MAX player wins from the MIN player 3 units (so the MIN player has to pay MAX player 3 units). Note that this is independent of the entry fee. So effectively, in this scenario, the MAX player has won 3 − x units overall, exactly how much the MIN player has lost.

1. What are the ranges of values for x that will be still worth playing the game for the MAX player (to enter the game at all)?
The game is worth playing if the maximiser plays in the range of values of x ≤ 5.

2. For a fixed value of x, do you prefer to be the first player (MAXimiser) or the second player (MINimiser)? Very briefly explain your answer.

Values of x	Player
x ≤ 5	MAXimiser would win
x ≥ 5	MINimiser would win

This question carries 10% of the marks for this coursework.

References:

1. https://www.tutorialandexample.com/adversarial-search-in-artificial-intelligence/
2. https://www.tutorialandexample.com/minimax-strategy/

3. https://www.tutorialandexample.com/alpha-beta-pruning/
4. https://www.youtube.com/watch?v=l-hh51ncgDI
5. https://pastebin.com/VSehqDM3
6. https://pastebin.com/rZg1Mz9G
7. https://en.wikipedia.org/wiki/Agent-based_model
8. https://www.geeksforgeeks.org/search-algorithms-in-ai/