**QUESTION 1**

a. False: a lucky DFS might expand exactly d nodes to reach the goal. A∗ largely dominates any graph-search algorithm that is guaranteed to find optimal solutions.

b. True: h(n) = 0 is always an admissible heuristic in any case , since costs are nonnegative.

c. False: A* search is often used in robotics; the space can be discretized or skeletonized.

d. True: depth of the solution matters for breadth-first search, not cost.

e. False: a rook can move across the board in move one, although the Manhattan distance from start to finish is 8.

**QUESTION 2**

**a. Problem formulation**

- State space: States are all possible city pairs (x, y). Initial state is some (x0, y0).

- Successor function: The successors of (x, y) are all pairs (x', y') such that x and x' are adjacent, y and y' are adjacent.

- Goal: Be at (i, i) for some i.

- Step cost function: The cost to go from (x, y) to (x', y') is max(d(x, x'),d(y, y'))

**b. Find an admissible heuristic**

In the best possible case, the two friends head towards each other in a straight line, with an equal amount of time spent on each step they take. They will meet in the middle of the line, and the time taken (in this best possible scenario) is D(x, y) / 2.

Since an admissible heuristic never overestimates the actual cost, even in the best-case scenario, only (iii) D(x, y) / 2 is admissible.

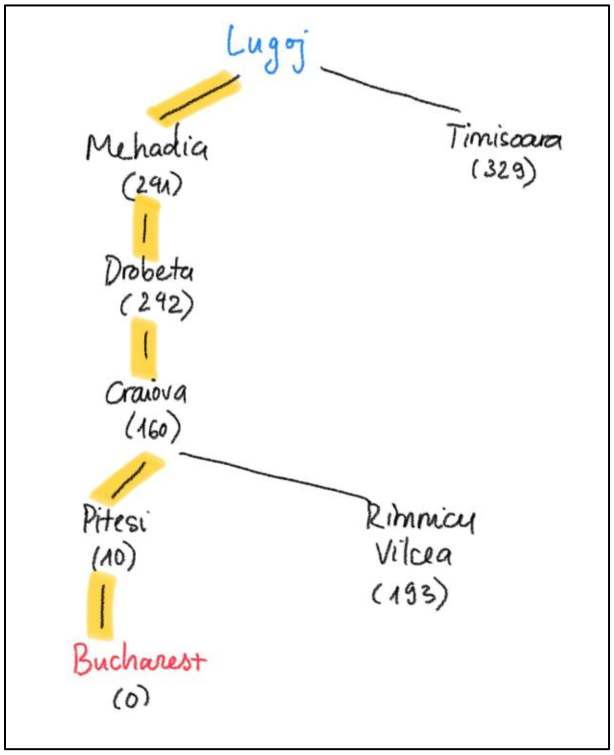
**QUESTION 3**

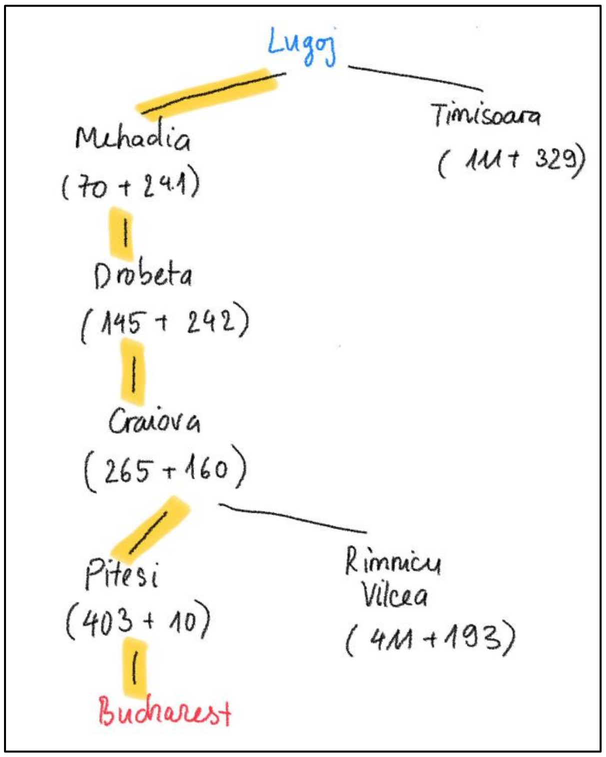
a. Evaluation function f(n) = h(n) (where h(n) is the straight-line distance to Bucharest)

**Answer**: Left figure.

b. Evaluation function f(n) = g(n) + h(n) (where g(n) is the actual cost from root node to current node n, and h(n) is the straight-line distance to Bucharest)

**Answer**: Right figure.

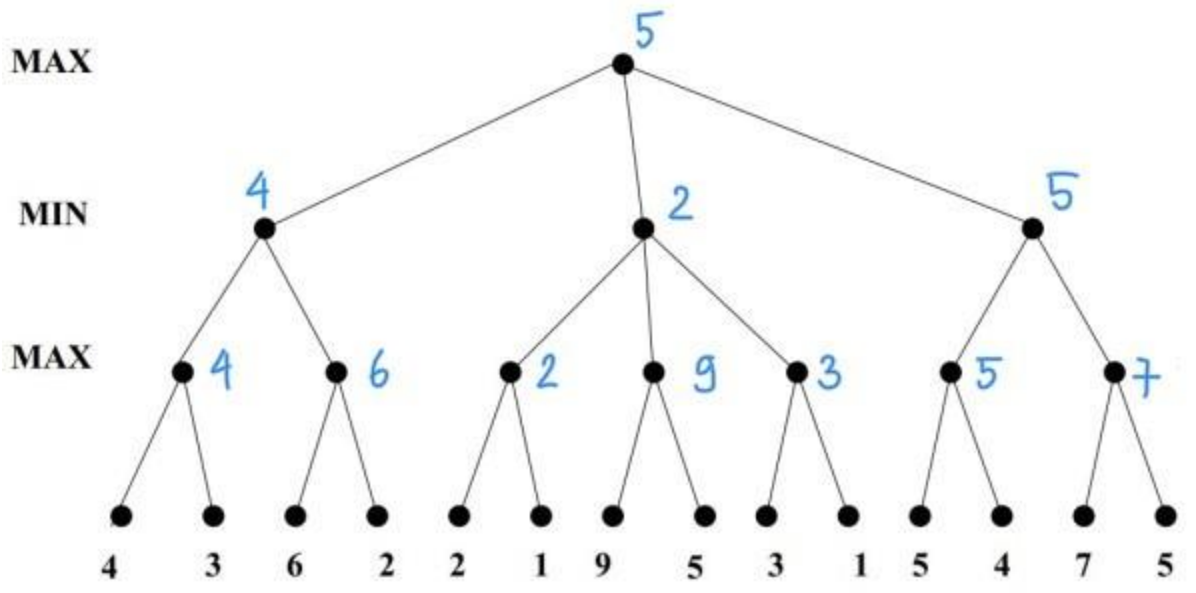




**QUESTION 4**

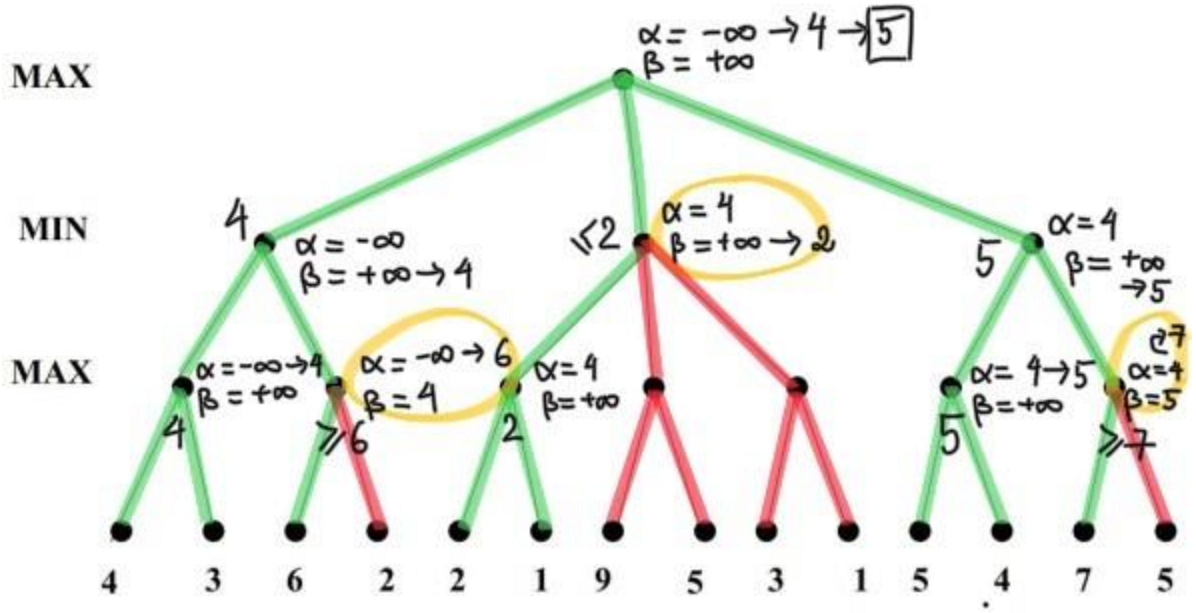
a. The minimizing agent (MIN) chooses a move that minimizes the utility (value of the terminal node).

The maximizing agent (MAX) chooses a move that maximizes the utility (value of the terminal node).

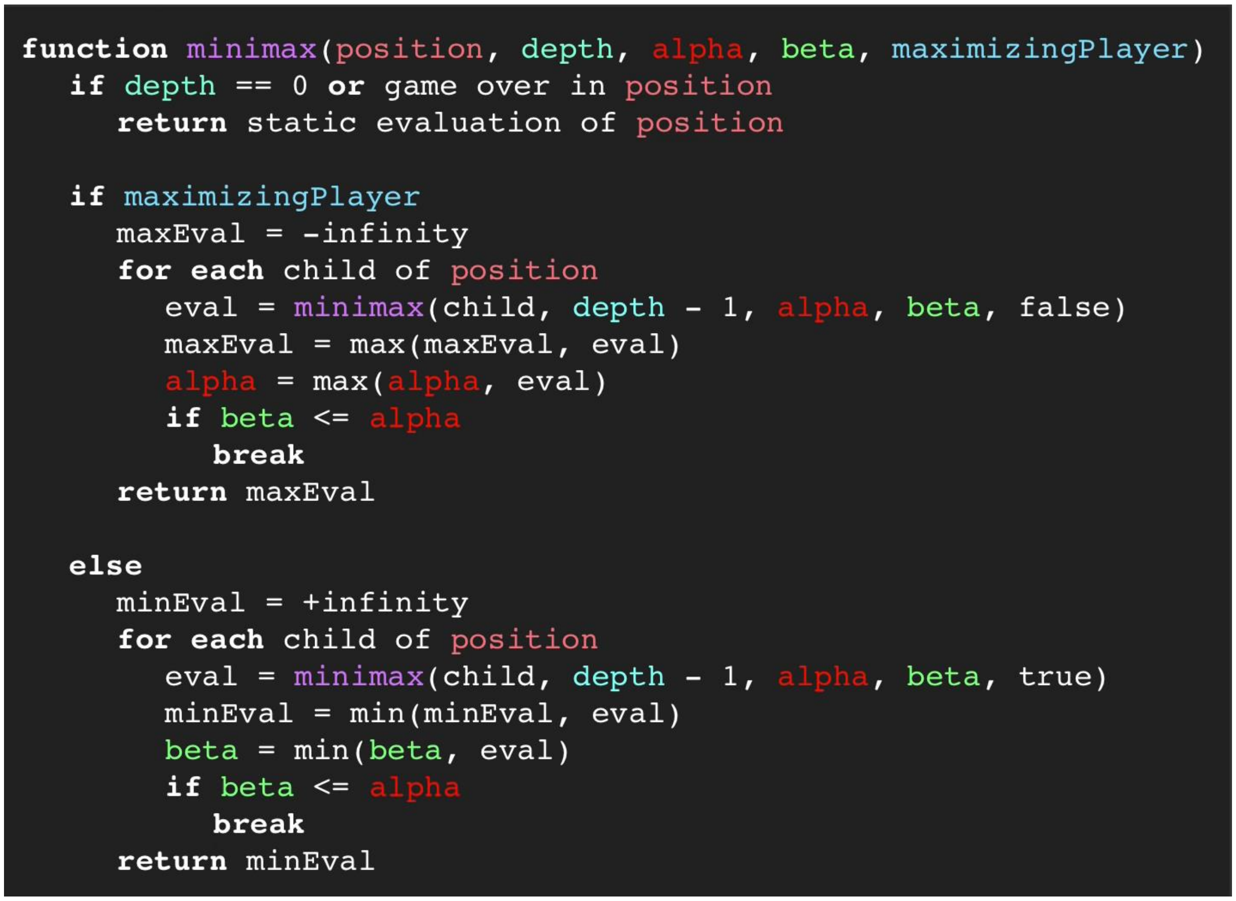


b. Branches in red are pruned. The reason for pruning (alpha being greater than beta) is enclosed in yellow.

To help visualize the alpha-beta pruning process, think of a node as having its own set of alpha and beta values, obtained from its parent the moment when it (the node) is traversed to. These alpha and betas values are updated occasionally, and when beta is not greater than alpha, the branch is pruned.



You can reference this pseudocode to help visualize the process.



```
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
```

**QUESTION 5**

The Manhattan distance heuristic captures more accurately the spatial cost to move a tile from its current position to its goal position. This may lead to a better, and more efficient, search process.

The actual Python code to implement and solve this problem will not be included in this file. However, try to properly define states, with their attributes (e.g. content, size, successor states, heuristic values,…), as well as carefully design the algorithms (Initiation, each iteration, & termination). GBFS or hill-climbing are easier choices, which you can implement first as sanity check. Other algorithms (e.g. local beam, genetic algorithm, simulated annealing) can be more tricky. In the process, try to analyze the performance of different heuristics and different algorithms.