CSCI-SHU 210 Data Structures - 100 Points

HOMEWORK ASSIGNMENT 6 - TREES

PROBLEM 1 – GAME TREE - 60 POINTS

Tic-Tac-Toe, also known as "noughts and crosses," is a classic two-player strategy game typically played on a 3x3 grid. The game's objective is to be the first player to form a horizontal, vertical, or diagonal line of three of their symbols (either "X" or "O") on the grid. Players take turns placing their symbols on empty spaces until one player achieves this goal, resulting in a win for that player. If the grid is completely filled without a winning line, the game ends in a draw.

Consider yourself as player "X". You want to maximize the possibility of winning Tic-Tac-Toe and, therefore, create a game tree containing all possible moves. We split this problem into 3 subtasks:

1. In task 1.A., implement the function *grow()* that generates all possible game moves starting from a given game. Please note that you need to generate a general tree, not a binary tree.
2. In task 1.B., implement the function *evaluate(current_board)* that evaluates the current state of the game. If "X" or "O" score 3 cells horizontally, vertically, or diagonally, X wins, and there are no further subtrees generated. If the board is full, no further moves are possible.
3. In task 1.C., implement the function *propagate()* that utilizes the min-max algorithm to propagate game results to the root node. Please see how the min-max algorithm works below.

What is a Game Tree?

A game tree is a visual representation of all possible moves and outcomes during a game. It starts with the initial state of the board and branches out with every possible move each player can make. Each node in the tree represents a specific game state, including the configuration of the board and whose turn it is.

As the game progresses, the tree expands, creating a branching structure that accounts for all potential sequences of moves. The leaves of the tree represent terminal states where the game ends, either in a win for one player, a draw, or a loss.

What is the min-max algorithm?

The Minimax algorithm is a decision-making algorithm used in two-player games with perfect information. Its objective is to find the best move for a player, assuming that the opponent will also make the best move. The algorithm works by recursively exploring the game tree, starting from the current state. In each node of the tree, the algorithm alternates between two players: the maximizing player (trying to maximize the score) and the minimizing player (trying to minimize the score). Terminal states (end of the game) are assigned scores: positive for a win, negative for a loss, and zero for a draw. These scores are then propagated back up the tree. Maximizing players choose the maximum score among their children while minimizing players choose the minimum score. At the root node, the algorithm selects the move that leads to the child node with the highest score for the maximizing player or the lowest score for the minimizing player. The chosen move is then executed in the actual game. The Minimax algorithm ensures that a player makes the best possible move based on the assumption that the opponent is also playing optimally. It's a fundamental concept in game theory and is used in various board games, including Tic-Tac-Toe.

Requirements

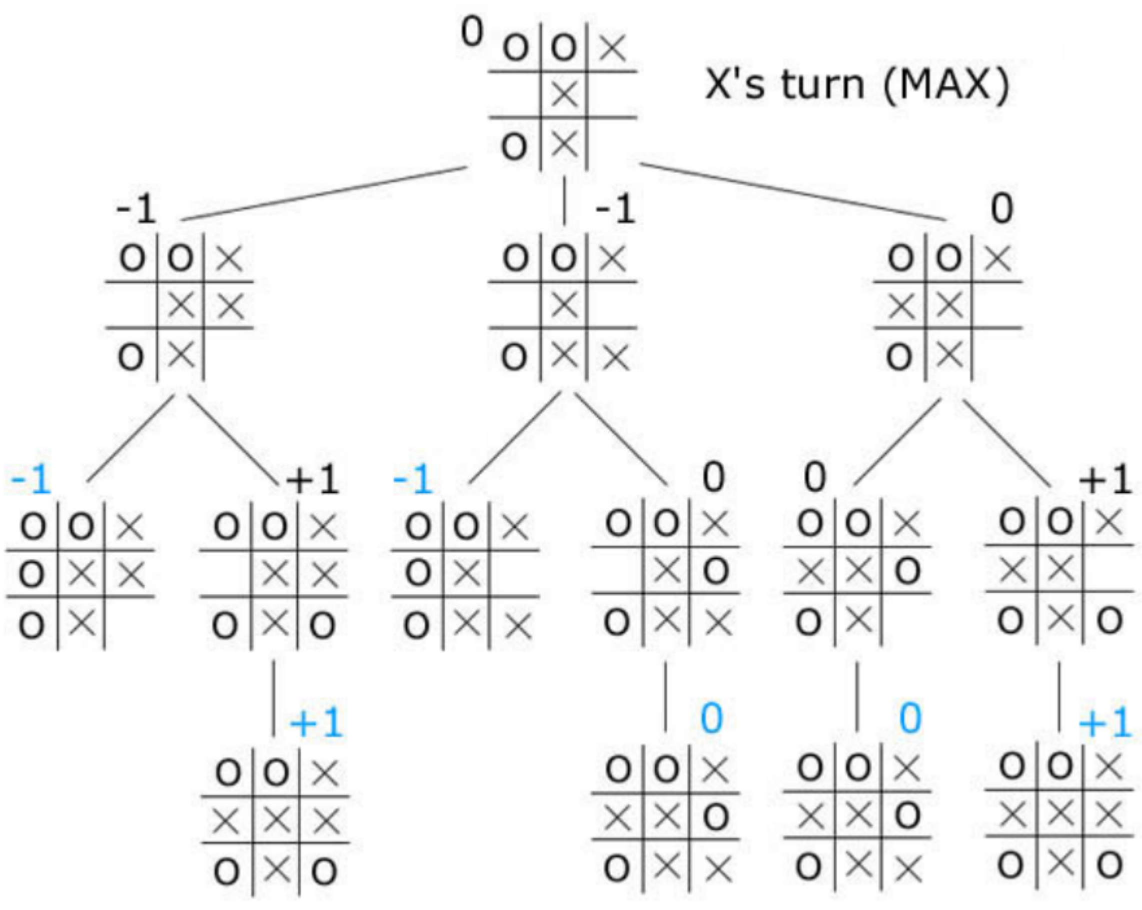
- You are not allowed to alter the provided classes.
- You need to implement all requested functions.
- You have to use "X" and "O" (this is not a zero).

Example 1

```
input_board = [
        ["O",  "O", "X"   ],
        [None, "O", None  ],
        ["O",  "X", None  ]
]

game = TreeNode(input_board, "X")
game.grow()  # this function expands your tree
game.propagate()  # this function min-maxes your tree

print("Should equal 1", game._score)
```



Problem 2 – Linked Binary Tree - 40 Points

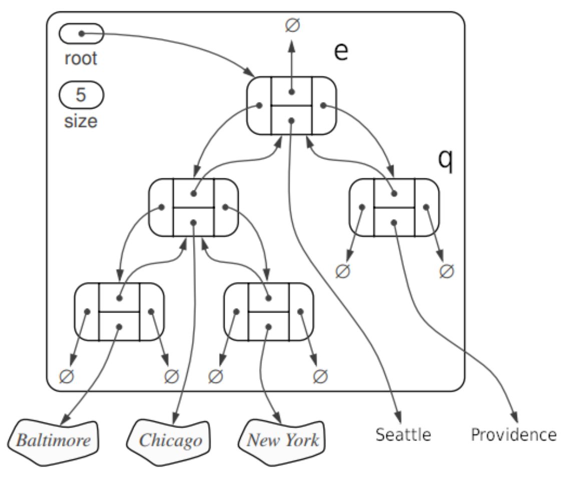
Implement the member function $\_swap(self, e,\ q)$ in the given linked binary tree class. The member function swaps the nodes e and q, and maintains child and parent references.

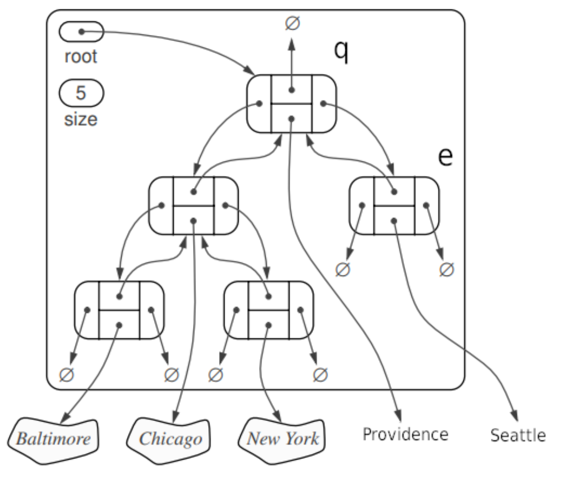
Requirements

- You are not allowed to alter the provided classes. **(-20 Points)**
- You are not allowed just to change node values. **(-20 Points)**
- The time complexity requirement of this method is $O(1)$. **(10 Points)**
- The space complexity requirement of this method is $O(1)$. **(10 Points)**

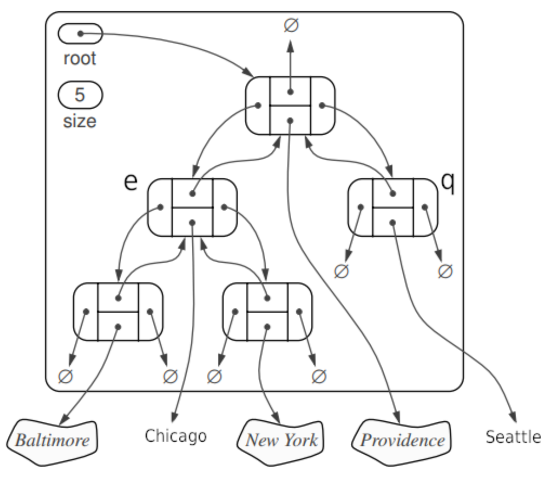
Example

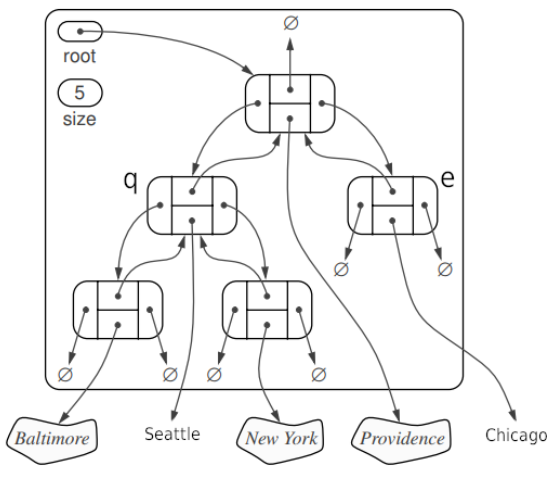
Swapping child and parent relationship





Swapping individual nodes


root
5
size
e
q
Baltimore
Chicago
New York
Providence
Seattle


root
5
size
q
e
Baltimore
Seattle
New York
Providence
Chicago