Atividade 2 - Pratica

Nome: Maverick Alekyne de Sousa Ribeiro - 541062 Nome: Ana Livia Sousa Davi Taveira - 536158

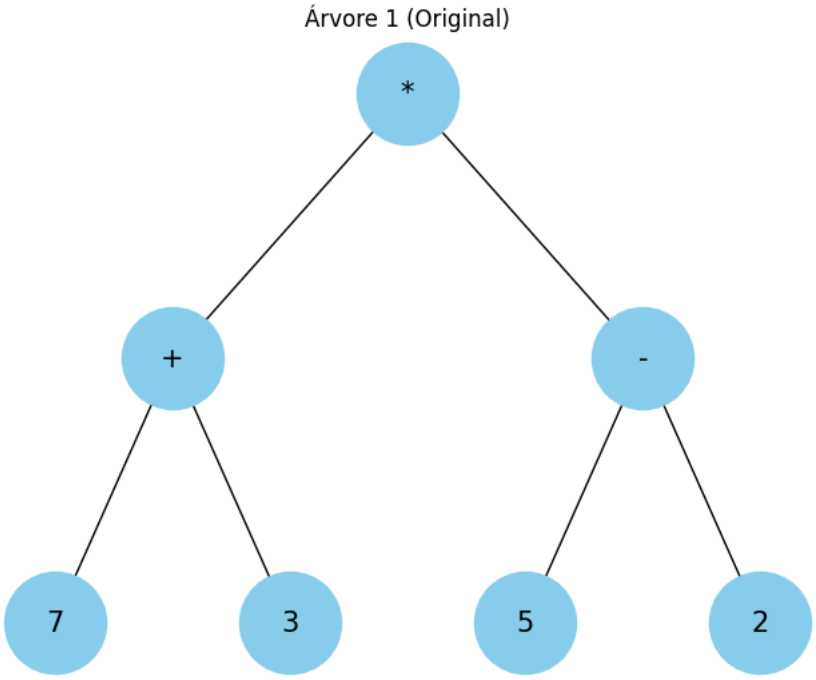
Professor: Hitalo Joseferson

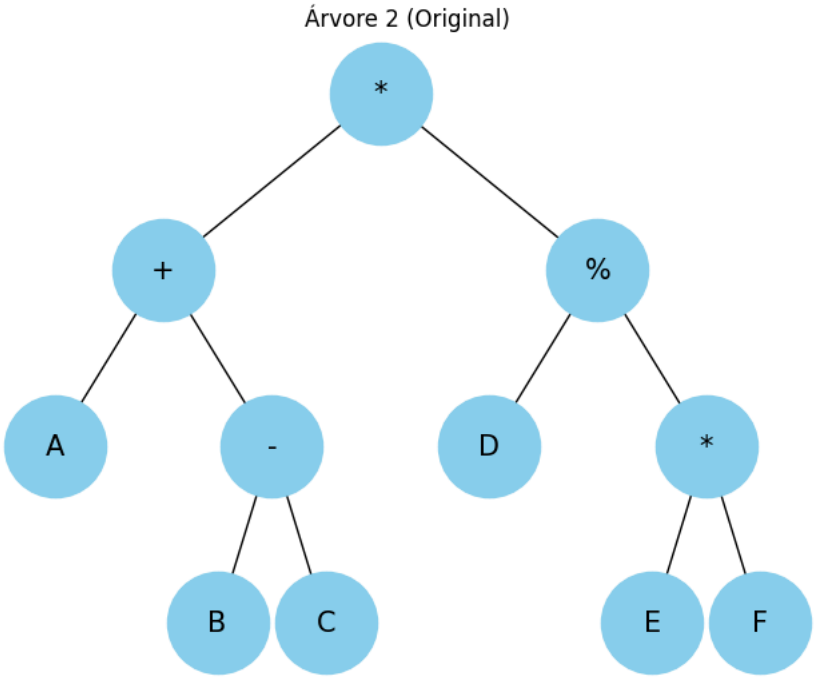
Escreva um programa em qualquer linguagem de programação que implemente os seguintes passos:

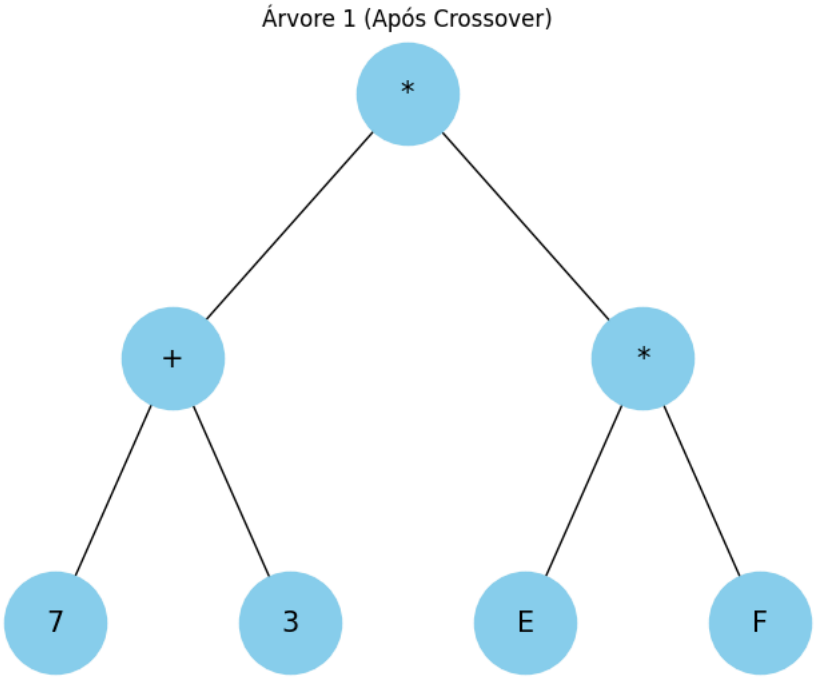
1. Criar as seguintes árvores binárias:
2. Seleciona um vértice aleatório nas duas árvores;
3. Realiza o crossover para criar dois filhos;
4. Exibe as árvores referentes aos pais e os dois filhos

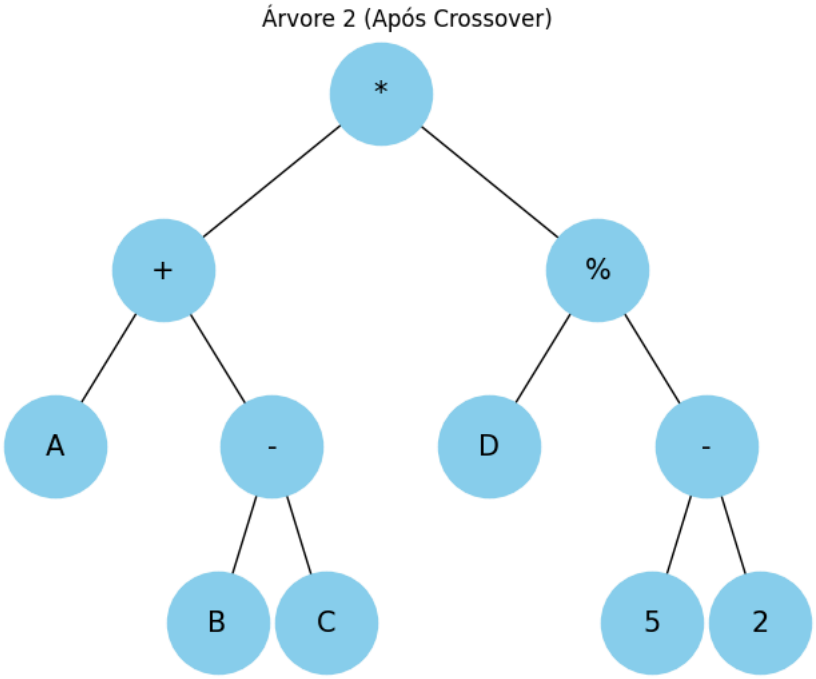
```
import random
import matplotlib.pyplot as plt
import networkx as nx

# Definindo as expressões
expr1 = "((7 + 3) * (5 - 2)) / ((4 - 3) * (3 - 1))"
expr2 = "(A + ((B - C)) * (D % (E * F)))"

# Classe Node
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

# Função para construir a árvore binária a partir de uma expressão
def build_tree(expr):
    stack = []
    operators = set(['+', '-', '*', '/', '%'])

    for char in expr:
        if char.isalnum() or char in operators:
            node = Node(char)
            stack.append(node)
        elif char == ')':
            right = stack.pop()
            operator = stack.pop()
            left = stack.pop()
            operator.left = left
            operator.right = right
            stack.append(operator)

    return stack[0]

# Função para desenhar a árvore
def draw_tree(node, pos=None, graph=None, level=0, width=2.):
    if pos is None:
        pos = {node: (0, 0)}
    if graph is None:
        graph = nx.Graph()

    width = width / 2

    if node.left:
        pos[node.left] = (pos[node][0] - width, pos[node][1] - 1)
        graph.add_edge(node, node.left)
        pos, graph = draw_tree(node.left, pos=pos, graph=graph, level=level + 1, width=width)

    if node.right:
        pos[node.right] = (pos[node][0] + width, pos[node][1] - 1)
        graph.add_edge(node, node.right)
        pos, graph = draw_tree(node.right, pos=pos, graph=graph, level=level + 1, width=width)

    return pos, graph

# Função para plotar a árvore
def plot_tree(tree, title):
    pos, graph = draw_tree(tree)
    labels = {node: node.value for node in pos}
    nx.draw(graph, pos, labels=labels, with_labels=True, node_size=3000, node_color="skyblue", font_size=15)
    plt.title(title)
    plt.show()

# Função para encontrar todos os nós em uma árvore
```

```
    def find_all_nodes(node):
        nodes = []
        if node:
            nodes.append(node)
            nodes.extend(find_all_nodes(node.left))
            nodes.extend(find_all_nodes(node.right))
        return nodes

    # Função para encontrar um nó aleatório em uma árvore
    def find_random_node(node):
        nodes = find_all_nodes(node)
        return random.choice(nodes) if nodes else None

    # Função para encontrar o pai de um nó
    def find_parent(root, node):
        if root is None or root == node:
            return None
        if root.left == node or root.right == node:
            return root
        parent = find_parent(root.left, node)
        if parent is not None:
            return parent
        return find_parent(root.right, node)

    # Função para realizar o crossover entre duas árvores
    def crossover(tree1, tree2):
        # Encontre um nó aleatório em cada árvore
        node1 = find_random_node(tree1)
        node2 = find_random_node(tree2)

        if node1 and node2:
            # Encontrar os pais dos nós selecionados
            parent1 = find_parent(tree1, node1)
            parent2 = find_parent(tree2, node2)

            # Substituir a subárvore em cada árvore
            if parent1:
                if parent1.left == node1:
                    parent1.left = node2
                else:
                    parent1.right = node2
            if parent2:
                if parent2.left == node2:
                    parent2.left = node1
                else:
                    parent2.right = node1

        return tree1, tree2

    # Criando as duas árvores
    tree1 = build_tree(expr1)
    tree2 = build_tree(expr2)

    # Plotando as árvores originais
    plot_tree(tree1, "Árvore 1 (Original)")


    plot_tree(tree2, "Árvore 2 (Original)")

    # Realizando o crossover
    new_tree1, new_tree2 = crossover(tree1, tree2)

    # Plotando as árvores resultantes após o crossover
    plot_tree(new_tree1, "Árvore 1 (Após Crossover)")


    plot_tree(new_tree2, "Árvore 2 (Após Crossover)")
```


Árvore 1 (Original)
*
+
-
7
3
5
2


Árvore 2 (Original)
*
+
%
A
-
D
*
B
C
E
F


Árvore 1 (Após Crossover)
*
+
*
7
3
E
F



Comece a programar ou gere código com IA.