AI VIET NAM – COURSE 2024

Object Oriented Programming - Day 14

Ngày 24 tháng 6 năm 2024

Ngày thực hiện:	24/06/2024
Người thực hiện:	Đinh Thị Tâm
Nguồn:	AIO2024 - Day 14
Nguồn dữ liệu (nếu có):	Link Sources code
Từ khóa:	Object Oriented Programming
Người tóm tắt:	Đinh Thị Tâm

1. Mô tả
   Lập trình hướng đối tượng (OOP) là một phương pháp lập trình tập trung vào việc tạo ra các "đối tượng", là những đơn vị cơ bản đại diện cho các thực thể trong thế giới thực. Mỗi đối tượng có các thuộc tính (data) và phương thức (hành vi) riêng. OOP mang lại nhiều lợi ích cho việc lập trình, bao gồm:

   - Tái sử dụng mã: OOP cho phép bạn viết mã có thể được sử dụng lại nhiều lần cho các mục đích khác nhau. Điều này giúp tiết kiệm thời gian và công sức, đồng thời giúp mã dễ dàng bảo trì hơn.
   - Tính linh hoạt: OOP giúp bạn dễ dàng thay đổi và mở rộng mã của mình. Khi bạn cần thêm tính năng mới, bạn chỉ cần tạo ra các đối tượng mới hoặc sửa đổi các đối tượng hiện có.
   - Tính bảo trì: OOP giúp mã của bạn dễ đọc và dễ hiểu hơn. Điều này giúp bạn dễ dàng tìm và sửa lỗi, đồng thời giúp những người khác dễ dàng hiểu mã của bạn.

2. **Bài tập:** Giả sử bạn đang quản lý một cửa hàng bán đồ điện tử. Cửa hàng bán nhiều loại sản phẩm khác nhau, bao gồm điện thoại, máy tính xách tay, tivi, v.v. Mỗi sản phẩm có các thuộc tính chung như tên, giá, nhà sản xuất và số lượng hàng tồn kho.Cửa hàng cũng cần theo dõi các đơn hàng của khách hàng, bao gồm thông tin khách hàng, sản phẩm đã mua và số lượng. Yêu cầu:

   - Tạo lớp Cây: Định nghĩa lớp Node đại diện cho một nút trong cây, bao gồm các thuộc tính như giá trị (value), con trái (left), và con phải (right). Định nghĩa lớp Tree đại diện cho cây, bao gồm thuộc tính gốc (root). Triển khai các phương thức khởi tạo (constructor) cho cả hai lớp Node và Tree.
   - Duyệt cây: viết phương thức duyệt hết các phần tử có trong cây theo phương pháp duyệt theo chiều ngang (BFS).
   - Tìm kiếm: Viết phương thức để tìm kiếm một giá trị cụ thể trong cây. Phương thức trả về True nếu giá trị được tìm thấy, False nếu không.
   - Thêm: Viết phương thức để thêm một giá trị mới vào cây. Cây nên được giữ cân bằng sau khi thêm.

- Xóa: Viết phương thức để xóa một giá trị cụ thể khỏi cây. Cập nhật các liên kết trong cây sau khi xóa.

(a) Code

```
# Tree
from graphviz import Graph
from collections import deque


class Node:
    def __init__(self, key):
        self.value = key
        self.left = None
        self.right = None
        self.height = 1  # chieu cao cua cay hien tai


class Tree:
    def __init__(self):
        self.root = None

    def insert_node(self, key):
        if self.root is None:
            self.root = Node(key)
        else:
            self._insert(self.root, key)
            # self.root.balance+=1

    def _insert(self, node, key):
        if key < node.value:
            if node.left is None:
                node.left = Node(key)
            else:
                self._insert(node.left, key)
        else:
            if node.right is None:
                node.right = Node(key)
            else:
                self._insert(node.right, key)
        # 2. update heigth of node
        node.height = 1 + max(self.get_height(node.left),
                              self.get_height(node.right))

        # 3. kiem tra can bang cua node
        balance = self.get_balance(node)

        # neu bi mat can bang:

        # TH 1 - Left Left (xoay right)
        if balance > 1 and key < node.left.value:
            return self.rotate_right(node)

        # TH 2 - Right Right (xoay left)
        if balance < -1 and key > node.right.key:
            return self.rotate_left(node)

        # TH3 - Left Right (xoay left right)
        if balance > 1 and key > node.left.key:
            node.left = self.rotate_left(node.left)
            return self.rotate_right(node)
```

```
57
58          # TH 4 - Right Left (xoay right left)
59          if balance < -1 and key < node.right.key:
60              node.right = self.rotate_right(node.right)
61              return self.rotate_left(node)
62
63      def get_height(self, node):
64          if not node:
65              return 0
66          return node.height
67
68      def get_balance(self, node):
69          if not node:
70              return 0
71          return self.get_height(node.left) - self.get_height(node.right)
72
73      def get_min_value_node(self, node):
74          if node is None or node.left is None:
75              return node
76          return self.get_min_value_node(node.left)
77
78      def rotate_left(self, z):
79          y = z.right
80          T2 = y.left
81
82          # Perform rotation
83          y.left = z
84          z.right = T2
85
86          # Update heights
87          z.height = 1 + max(self.get_height(z.left), self.get_height(z.right))
88          y.height = 1 + max(self.get_height(y.left), self.get_height(y.right))
89
90          # Return the new root
91          return y
92
93      def rotate_right(self, y):
94          x = y.left
95          T2 = x.right
96
97          # Perform rotation
98          x.right = y
99          y.left = T2
100
101         # Update heights
102         y.height = 1 + max(self.get_height(y.left), self.get_height(y.right))
103         x.height = 1 + max(self.get_height(x.left), self.get_height(x.right))
104
105         # Return the new root
106         return x
107
108     def pre_order(self, root):
109         if not root:
110             return
111         print(f"{root.key} ", end="")
112         self.pre_order(root.left)
113         self.pre_order(root.right)
114
115     # ham search
116     def search(self, key):
```

```
        return self._search(self.root, key) is not None

    def _search(self, node, key):
        if node is None or node.value == key:
            return node

        if key < node.value:
            return self._search(node.left, key)

        return self._search(node.right, key)

    def delete(self, root, key):
        # 1. xoa node trong cay AVL
        if not root:
            return root

        if key < root.value:
            root.left = self.delete(root.left, key)
        elif key > root.value:
            root.right = self.delete(root.right, key)
        else:
            if root.left is None:
                return root.right
            elif root.right is None:
                return root.left

            temp = self.get_min_value_node(root.right)
            root.value = temp.value
            root.right = self.delete(root.right, temp.value)

        if not root:
            return root

        # 2. cap nhat chieu cao cua node
        root.height = 1 + max(self.get_height(root.left),
                              self.get_height(root.right))

        # 3. Kiem tra tinh can bang cua cay
        balance = self.get_balance(root)

        # Neu bi mat can bang:

        # TH1 - Left Left (xoay phai)
        if balance > 1 and self.get_balance(root.left) >= 0:
            return self.rotate_right(root)

        # TH 2 - Left Right (xoay trai phai)
        if balance > 1 and self.get_balance(root.left) < 0:
            root.left = self.rotate_left(root.left)
            return self.rotate_right(root)

        # TH 3 - Right Right (xoay trai)
        if balance < -1 and self.get_balance(root.right) <= 0:
            return self.rotate_left(root)

        # TH 4 - Right Left (xoay phai trai)
        if balance < -1 and self.get_balance(root.right) > 0:
            root.right = self.rotate_right(root.right)
            return self.rotate_left(root)

```

```
        return root

# ham duyet cay Bfs


def traversal_bfs(root):
    if root is None:
        return []

    result = []
    queue = deque([root])

    while queue:
        node = queue.popleft()
        result.append(node.value)

        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)

    return result

# Inorder tree traversal (Left - Root - Right)


def inorderTraversal(root):
    if not root:
        return
    inorderTraversal(root.left)
    print(root.value, end=" ")
    inorderTraversal(root.right)


# main
bst = Tree()
elements = [50, 30, 20, 40, 70, 60, 80, 25, 45, 90, 75]
for x in elements:
    bst.insert_node(x)
print('inoreder traversal BST tree:')
inorderTraversal(bst.root)
print('\nTraversal tree by BFS:')
result = traversal_bfs(bst.root)
print(result)
# search
x = 40
if bst.search(x) == True:
    print(f'Found {x} in BST tree ')
    bst.delete(bst.root, x)
    print(f'BST tree after delete {x} is:')
    print(traversal_bfs(bst.root))
else:
    print(f'Not found {x} in BST tree ')
```

(b) Kết quả thực thi chương trình

```
inoreder traversal BST tree:
25 40 45 50 60 70 75 80 90
Traversal tree by BFS:
[50, 40, 70, 25, 45, 60, 80, 75, 90]
```

```
Found 40 in BST tree
BST tree after delete 40 is:
[50, 45, 70, 25, 60, 80, 75, 90]
```