دانشگاه آزاد اسلامی شیراز - دانشکده مهندسی کامپیوتر

ساختمان داده ها

استاد : دکتر اسکندری

ترم اول سال تحصیلی 1403-1404

هفته هفتم

گرد آورندگان : امیر حسین همتی ، حمید رضا نامجومنش

توضیحات :

در صورت مشاهده لوگوی سبز open ai در بالای سوال خود شما می توانید از هوش مصنوعی در پاسخ به سوال خود استفاده کنید اما باید

سوالی را که از هوش مصنوعی پرسیدید اسکرین شات گرفته و در پاسخنامه خود قرار دهید.

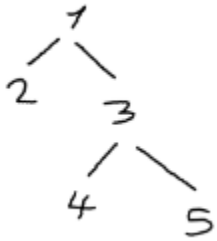
در صورت مشاهده لوگوی قرمز open ai در بالای سوال خود شما نمی توانید از هوش مصنوعی در پاسخ به سوال خود استفاده کنید و باید با

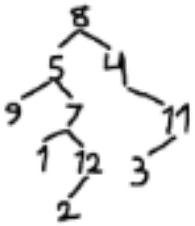
دانش خود به این سئوال پاسخ دهید.

سوالات :

سوال 1 : چرا و در چه شرایطی استفاده از درخت ها به جای لیست ها و آرایه ها مفید تر است با یک مثال توضیح دهید .

سوال 2 : pre order , in order , post order درخت های زیر را بنویسید .





سوال 3 : کدی را پیاده سازی کنید که تعداد گره های یک درخت را مشخص کند (فقط یک راه حل ندارد می توان آن را به نحو های مختلف پیاده سازی نمود) .

سوال 4 : با توجه به کد های زیر به سوالات زیر پاسخ دهید .

- همان طور که می‌بینید دو تابع برای اضافه کردن راس به درخت وجود دارد (add_new_node_1 و add_new_node_2). فرق این دو پیاده‌سازی مختلف در چیست؟
- تابع find_in_subtree چگونه عمل می‌کند؟
- چگونه می‌توان اندازه زیر درخت فرزند چپ ریشه را به دست آورد؟

```
class TreeNode:
    def __init__(self, label):
        self.parent = None
        self.left_child = None
        self.right_sibling = None
        self.label = label

    def __str__(self):
        return "TreeNode(%s)" % str(self.label)
```

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

    def assign_root(self, label):
        assert self.root is None
        self.root = TreeNode(label)

    def is_empty(self):
        return self.root is None

    def add_new_node_1(self, parent, label):
        new_node = TreeNode(label)
        left_child = parent.left_child
        parent.left_child = new_node
        new_node.right_sibling = left_child
        new_node.parent = parent
        return new_node

    def add_new_node_2(self, parent, label):
        new_node = TreeNode(label)
        new_node.parent = parent
        if parent.left_child is None:
            parent.left_child = new_node
        else:
            left_child = parent.left_child
            while left_child.right_sibling is not None:
                left_child = left_child.right_sibling
            left_child.right_sibling = new_node
        return new_node

    def add_new_node(self, parent, label):
        return self.add_new_node_2(parent, label)

    def find_in_subtree(self, label, node):
        if node.label == label:
            return node

        child = node.left_child
        while child is not None:
            result = self.find_in_subtree(label, child)
            if result is not None:
                return result
            child = child.right_sibling
        return None

    def find_by_label(self, label):
        if self.is_empty():
            return None
        return self.find_in_subtree(label, self.root)

    def add_new_node_by_label(self, parent_label, label):
        self.add_new_node(self.find_by_label(parent_label), label)

    def get_subtree_size(self, node):
        if node is None:
            return 0

        count = 1
        child = node.left_child
        while child is not None:
            count += self.get_subtree_size(child)
            child = child.right_sibling

        return count

    def get_size(self):
        if self.is_empty():
            return 0
        return self.get_subtree_size(self.root)
```