پیاده سازی درخت باینری و چند مسئله کاربردی

۱ پیاده سازی

تعریف کلاس Node

```
class Node:

    def __init__(self, val):
        self.parent = None
        self.left = None
        self.right = None
        self.element = val
```

تعریف کلاس BinaryTree

```
class BinaryTree:

    def __init__(self):
        self.root = None

    def getRoot(self):
        return self.root

    def addroot(self,val):
        if self.root is not None:
           print("Tree is not empty!")
        else:
           self.root = Node(val)
```

تعریف چند متد کلاس .BinaryTree اضافه کردن فرزند چپ و راست به یک راس.

```
def addleft(self,p, val):
    if (p is None):
        print("Empty Reference!")
        return
    if(p.left is not None):
        print("Left child is already present!")
        return
    else:
        p.left = Node(val)
        p.left.parent = p
        return p.left

def addright(self,p, val):
    if (p is None):
        print("Empty Reference!")
        return
    if (p.right is not None):
        print("Right child is already present!")
        return
    else:
        p.right = Node(val)
        p.right.parent = p
        return p.right


def is_leaf(self, p):
    if (p is not None and p.left is None and p.right is None):
        return True
    else:
        return False

def deleteTree(self):
    self.root = None
```

پیاده سازی پیمایش درخت با روش میان ترتیب inOrder

```
def inOrder(self):
       if(self.root is not None):
           self._inOrder(self.root)

   def _inOrder(self, node):
       if(node is not None):
           self._inOrder(node.left)
           print(str(node.element), end=' ')
           self._inOrder(node.right)
```

مجهز کردن کلاس BinaryTree به یک iterator

بعد از انجام این کار، می‌توانیم با استفاده از حلقه for به مقادیر ذخیره شده در رئوس درخت را با ترتیب خاصی دسترسی داشته باشیم. برای مثال با فرض اینکه iterator ایجاد شده است، می‌توانیم قطعه کد زیر را داشته باشیم.

```
T = BinaryTree()


T.addroot('+')
x = T.addleft(T.getRoot(),'3')
y = T.addright(T.getRoot(),'*')
T.addright(y,'2')
z=T.addleft(y,'+')
T.addleft(z,'5')
T.addright(z,'9')

for x in T:
    print(x)
```

برای ایجاد iterator باید متد iter را به کلاسهای BinaryTree و Node اضافه کنیم.

```
class Node:
    def __init__(self, val):
     #   self.parent = None
        self.left = None
        self.right = None
        self.element = val

    def __iter__(self):
        if self.left:
            yield from self.left

        yield self.element

        if self.right:
            yield from self.right
```

همچنین در کلاس BinaryTree متد iter را اضافه می‌کنیم.

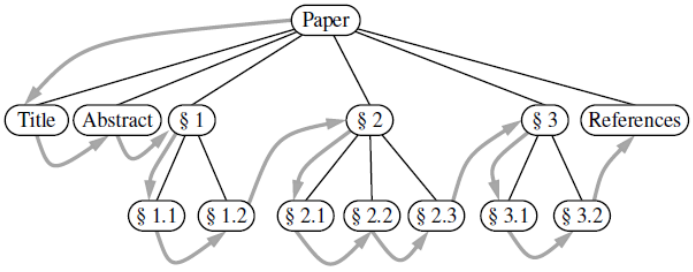
```
    def __iter__(self):
        if self.root is not None:
            yield from self.root
```

۲ چند کاربرد از پیمایشهای درخت باینری

از پیمایشهای preorder و inorder و postorder می‌توان برای حل بعضی از مسائل کابردی استفاده کرد. در زیر چند نمونه را بررسی می‌کنیم.

۱.۲ تولید فهرست

بخشهای مختلف یک کتاب یا مقاله را می‌توان بصورت سلسله مراتبی نمایش داد. برای مثال شکل زیر را ببینید.



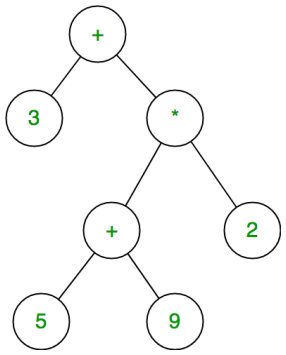
همانطور که در شکل نشان داده شده است، برای اینکه فهرست مطالب را با ترتیب صحیح تولید کنیم از پیمایش preorder استفاده می‌کنیم. اگر بخواهیم مانند شکل زیر، سمت راست، برای هر بخش به تعداد مناسب فاصله از سر خط قرار داده شود، می‌توانیم در تعریف بازگشتی preorder یک متغیر اضافه را جهت رصد کردن عمق راس اضافه کنیم. دقت کنید اینجا درخت لزوما باینری نیست. در قطعه کد زیر فرض شده که متدی به اسم children را داریم که فرزندان یک راس را به ترتیب از چپ به راست تولید میکند.

```
Paper                Paper
Title                  Title
Abstract               Abstract
1                       1
1.1                       1.1
1.2                       1.2
2                       2
2.1                       2.1
...                       ...

(a)                  (b)
```

```
def table_of_content(self):
       if(self.root is not None):
           self._table_of_content(self.root, 0)

   def _table_of_content(self, node, depth):
       if(node is not None):
           print(' '*depth + str(node.element))
           for x in self.children(node):
                 self._table_of_content(self, x, depth+1):
```

۲.۲ محاسبه حاصل یک درخت عبارت

یک نمونه از یک درخت عبارت در شکل زیر نشان داده شده است. برای سادگی فرض کنید همه عملگرها دودویی هستند.



برای محاسبه حاصل درخت عبارت می‌توانیم از پیمایش postorder استفاده کنیم.

```
def evaluate(self):
    return self._evaluate_recur(self.root)

def _evaluate_recur(self, p):
    if self.is_leaf(p):
        return float(p.element) # we assume element is numeric
    else:
        op = p.element
        left_val = self._evaluate_recur(p.left)
        right_val = self._evaluate_recur(p.right)
        if op == '+' : return left_val + right_val
        elif op == '-' : return left_val - right_val
        elif op == '/' : return left_val / right_val
        else: return left_val * right_val # treat x or as multiplication
```

۳ پرانتزبندی درخت عبارت

برای اینکه عبارت مربوط به درخت عبارت را بصورت پرانتزبندی شده تولید کنیم، می‌توانیم از شیوه پیمایش in-Order استفاده کنیم. دقت کنید هر بار که به سمت چپ می‌رویم که پرانتز باز می‌گذاریم و هر بار که پیمایش زیردرخت سمت راست را تمام کردیم یک پرانتز بسته می‌گذاریم. عبارت زیر حاصل اجرای متد parenthesize برای درخت عبارت شکل صفحه قبل است.

$$(3 + ((5 + 9) * 2))$$

```
def parenthesize(self):
    if(self.root != None):
        return self._parenthesize_recur(self.root)


def _parenthesize_recur(self, p):
    if (self.is_leaf(p)):
        print(p.element, end='')
    else:
        print('(' , end='')
        self._parenthesize_recur(p.left)
        print(p.element,end='')
        self._parenthesize_recur(p.right)
        print(')', end='')
```