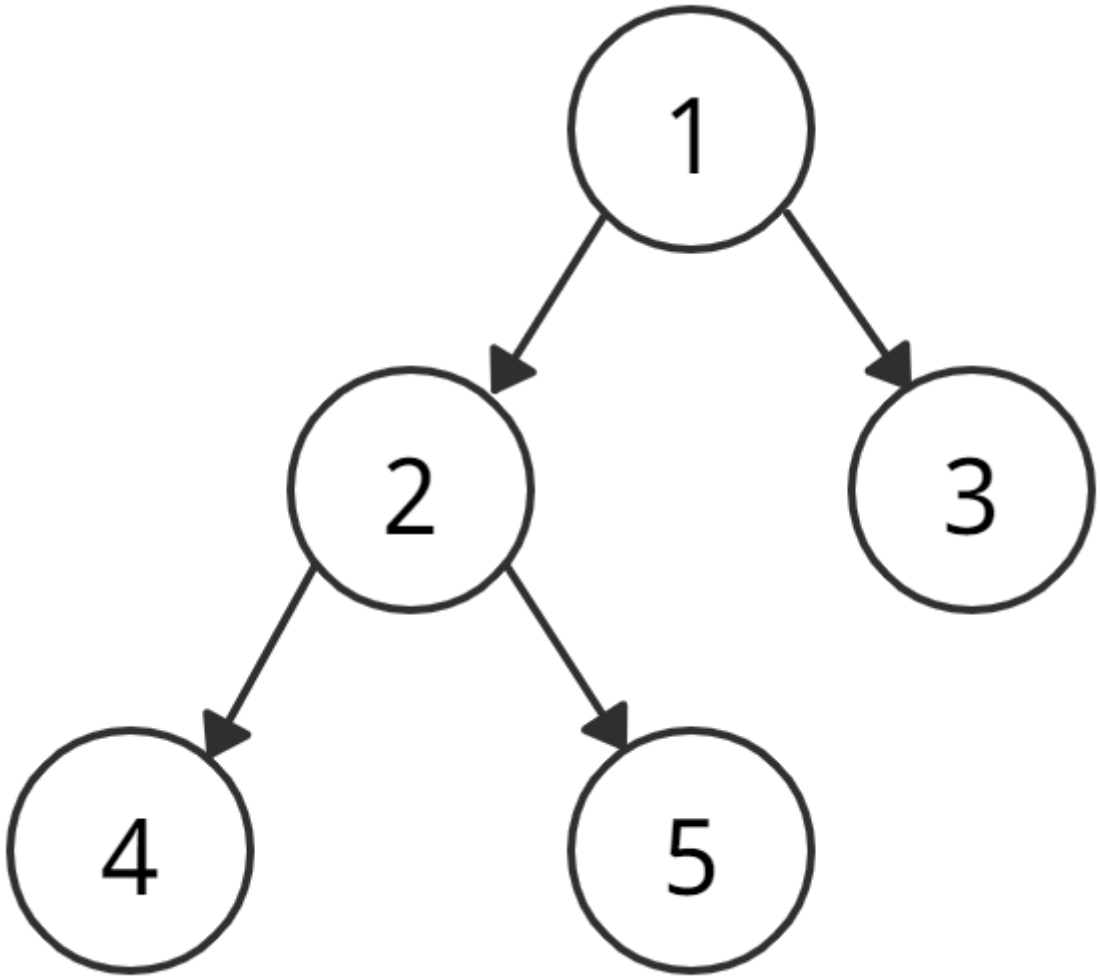
Бинарное Дерево

ПРИМЕЧАНИЕ

Двоичное дерево – древовидная структура данных, в которой у родительских узлов не может быть больше двух детей.



Строим двоичное дерево на Python

Определим метод `__init__()`. Как всегда, он принимает `self`. Также мы передаем в него значение, которое будет храниться в узле. Установим значение узла, а затем определим левый и правый указатель (для начала поставим им значение None).

```
class TreeNode:
  def __init__(self, val=0, left=None, right=None):
    self.val = val
    self.left = left
    self.right = right
```

И… все! Если речь идет о двоичном дереве, единственное, что его отличает от связного списка, это то, что вместо next у нас тут есть left и right.

Давайте построим дерево, которое изображено на схеме в начале статьи. Верхний узел имеет значение 1. Далее мы устанавливаем левые и правые узлы, пока не получим желаемое дерево.

```
tree = TreeNode(1)
tree.left = TreeNode(2)
tree.right = TreeNode(3)
tree.left.left = TreeNode(4)
tree.left.right = TreeNode(5)
```

Создание дерева из list

```
def from_list(l):
    nodes = [TreeNode(v) for v in l]
    kids = nodes[::-1]
    root = kids.pop()
    for node in nodes:
        if node:
            if kids:
                node.left = kids.pop()
            if kids:
                node.right = kids.pop()
    return root

t = TreeNode.from_list([4,2,7,1,3,6,9])
print(t)
```

Еще вариант

```
def creatBTree(data, index):
    pNode = None
    if index < len(data):
```

```
        if data[index] == None:
            return
        pNode = TreeNode(data[index])
        pNode.left = creatBTree(data, 2 * index + 1) # [1, 3, 7, 15,
...]
        pNode.right = creatBTree(data, 2 * index + 2) # [2, 5, 12, 25,
...]
    return pNode

lst = [5,4,8,11,None,13,4,7,2,None,None,None,1]
root = creatBTree(lst, 0)
```

Обход двоичного дерева

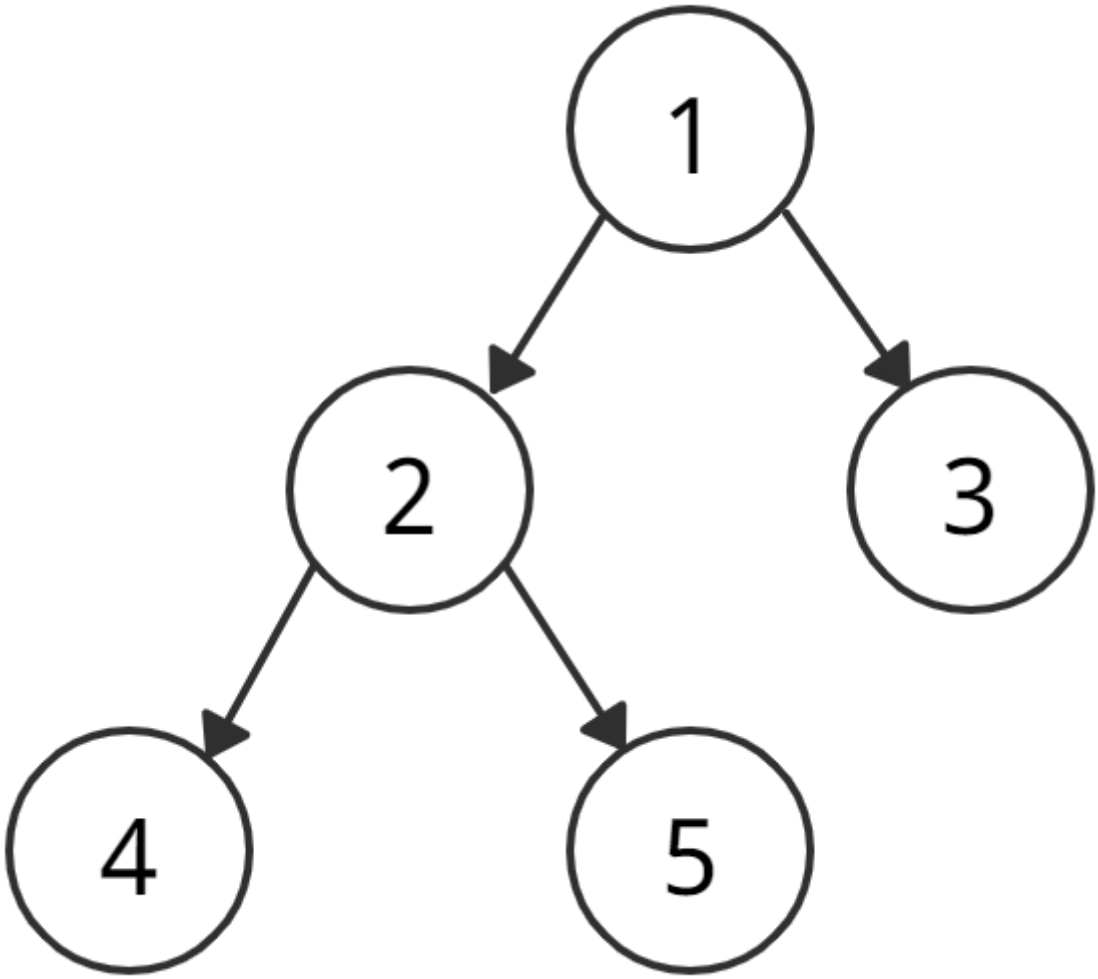
Итак, вы построили дерево и теперь вам, вероятно, любопытно, как же его увидеть. Нет никакой команды, которая позволила бы вывести на экран дерево целиком, тем не менее мы можем обойти его, посетив каждый узел. Но в каком порядке выводить узлы?

Самые простые в реализации обходы дерева

- прямой (Pre-Order)
- обратный (Post-Order)
- центрированный (In-Order)

Вы также можете услышать такие термины, как поиск в ширину и поиск в глубину, но их реализация сложнее, ее мы рассмотрим как-нибудь потом.

Итак, что из себя представляют три варианта обхода, указанные выше? Давайте еще раз посмотрим на наше дерево.



При прямом обходе мы посещаем родительские узлы до посещения узлов-потомков. В случае с нашим деревом мы будем обходить узлы в таком порядке: 1, 2, 4, 5, 3.

Обратный обход двоичного дерева – это когда вы сначала посещаете узлы-потомки, а затем – их родительские узлы. В нашем случае порядок посещения узлов при обратном обходе будет таким: 4, 5, 2, 3, 1.

При центрированном обходе мы посещаем все узлы слева направо. Центрированный обход нашего дерева – это посещение узлов 4, 2, 5, 1, 3.

Давайте напишем методы обхода для нашего двоичного дерева.

Прямой обход (Pre-Order)

Написать обход просто. Прямой обход – это посещение родительского узла, а затем каждого из его потомков. Мы «посетим» родительский узел, выведя его на экран, а затем «обойдем» детей, вызывая этот метод рекурсивно для каждого узла-потомка.

```
# Выводит родителя до всех его потомков
def pre_order(node):
    if node:
        print(node.value)
        pre_order(node.left)
        pre_order(node.right)
```

Дополнительные ссылки

- оригинал статьи
- статья о деревьях в Питоне

Видео материалы

- Задача из Собеседования в Microsoft (Бинарные Деревья)

Примеры на letcode

- https://leetcode.com/problems/binary-tree-maximum-path-sum/
- https://leetcode.com/problems/path-sum/
- https://leetcode.com/problems/invert-binary-tree/

Теги: python programing treenode Data Structure

Отредактировать эту страницу

*Последнее обновление **4 сент. 2023 г.** от **Stavis***