数算学习随机——作业

start since 2024.3.7 Complied by zxy

前言：做作业时想要积累下来的一些知识点和技巧，因为有时候做完作业太累了就不想整理了，同时有些知识点整理到数据结构学习随记里面了，因此这里面的内容其实不多，不过都是很有用的！

1、.strip() 方法是用于去除字符串两端的空白字符（包括空格、制表符和换行符）。

```
pythonuser_input = "  hello, world   "
cleaned_input = user_input.strip()
print(cleaned_input)  # 输出：hello, world
```

2、`has_parent.index(False)`：这部分代码的作用是返回列表中第一个值为 `False` 的元素的索引（位置）。如果列表中不存在值为 `False` 的元素，则会触发 ValueError 异常。

3、想反转一个列表：`list[::-1]`，而`list.reverse()`是一个操作，不会返回任何值，是在原列表上直接操作。

4、一个四则运算表达式："(3+5)2-6/2"，将其转换为后序表达式为："35+262/-"，根据该后序表达式构建树，树的结构如下：

```
          -
        /   \
       *     /
      / \   / \
     +   2 6   2
    / \
   3   5
```

可以看出来顺序是从left到right，从下到上操作的！

5、树的两序转一序：

```
"""
后序遍历的最后一个元素是树的根节点。然后，在中序遍历序列中，根节点将左右子树分开。
可以通过这种方法找到左右子树的中序遍历序列。然后，使用递归地处理左右子树来构建整个树。
"""
```

6、切片的一堆注意：

在 Python 中，`[s:-1]` 表示从索引 `s` 开始（包括 `s` 所在的位置）到倒数第二个元素（不包括最后一个元素）的切片操作。具体含义如下：

- 如果 `s` 是正数，表示从索引为 `s` 的元素开始（包括该元素），直到倒数第一个元素（不包括最后一个元素）。
- 如果 `s` 是负数，表示从倒数第 `s` 个元素开始（包括该元素），直到倒数第一个元素（不包括最后一个元素）。

举个例子，假设有一个列表 `my_list = [1, 2, 3, 4, 5]`：

- `my_list[1:-1]` 将返回 `[2, 3, 4]`，因为它从索引 `1`（包括索引 `1` 处的元素 `2`）开始，到倒数第二个元素 `4` 结束（不包括最后一个元素 `5`）。
- `my_list[-2:-1]` 将返回 `[4]`，因为它从倒数第二个元素 `4`（包括元素 `4`）开始，到倒数第一个元素结束（不包括最后一个元素 `5`）。

7、递归！（二叉搜索树遍历）

```
def pre_to_post(preorder):
    if not preorder:
        return []
    root=preorder[0]
    left_sub=[x for x in preorder if x<root]
    right_sub=[x for x in preorder if x>root]
    return pre_to_post(left_sub)+pre_to_post(right_sub)+[root]

n=int(input())
preorder=list(map(int,input().split()))
print(' '.join(map(str,pre_to_post(preorder))))
```

好的，让我们逐步模拟一下 `pre_to_post` 函数的递归过程：

1. 首先，我们输入测试数据为：

```
5
4 2 1 3 5
```

1. 运行程序时，首先会执行 `pre_to_post` 函数，并将 `preorder` 列表 `[4, 2, 1, 3, 5]` 传入函数。
2. 因为 `preorder` 列表非空，所以不会执行 `return []`，继续执行后续代码。
3. 在 `preorder` 列表中，`root` 的值为 `4`，然后根据这个 `root` 值，将 `preorder` 划分为左子树和右子树：
    - `left_sub` 列表为 `[2, 1, 3]`
    - `right_sub` 列表为 `[5]`
4. 接着，程序会进行递归调用：
    - 对左子树 `left_sub` 调用 `pre_to_post(left_sub)`，递归过程如下：

        ```
        pre_to_post([2, 1, 3])
        ```

        - `root` 的值为 `2`
        - 将 `left_sub` 划分为 `[1]` 和 `[3]`
        - 递归调用 `pre_to_post([1])` 和 `pre_to_post([3])`
    - 对右子树 `right_sub` 调用 `pre_to_post(right_sub)`，递归过程如下：

        ```
        pre_to_post([5])
        ```

        - `root` 的值为 `5`
        - 左子树和右子树都为空，直接返回 `[5]`
5. 递归调用的结果会根据左右子树的返回值进行拼接，并且加上根节点的值 `[root]`。
    - 对于左子树，递归调用的结果为 `[1, 3]`
    - 对于右子树，递归调用的结果为 `[5]`
6. 将左子树、右子树和根节点的值拼接起来，得到最终的后序遍历结果。

根据上述过程，最终的后序遍历结果为 `[1, 3, 2, 5, 4]`。

8、去重：

`dict.fromkeys()` 是一个 Python 字典（`dict`）类的方法，用于创建一个新的字典，该字典的键来自于指定的序列（例如列表、元组等），而对应的值都设置为一个指定的默认值。

语法如下：

```
dict.fromkeys(seq[, value])
```

参数说明：

- `seq`：指定序列，可以是列表、元组、集合等可迭代对象。
- `value`：可选参数，用于设置字典中所有键对应的默认值，默认为 `None`。

示例：

```
python# 使用列表作为序列
seq = ['a', 'b', 'c']
d = dict.fromkeys(seq)
print(d)  # 输出: {'a': None, 'b': None, 'c': None}

# 指定默认值
d = dict.fromkeys(seq, 10)
print(d)  # 输出: {'a': 10, 'b': 10, 'c': 10}
```

该方法在某些情况下很有用，如创建一个具有默认值的字典，或者将一个可迭代对象的值用作字典的键，并为每个键设置相同的默认值。

去重：

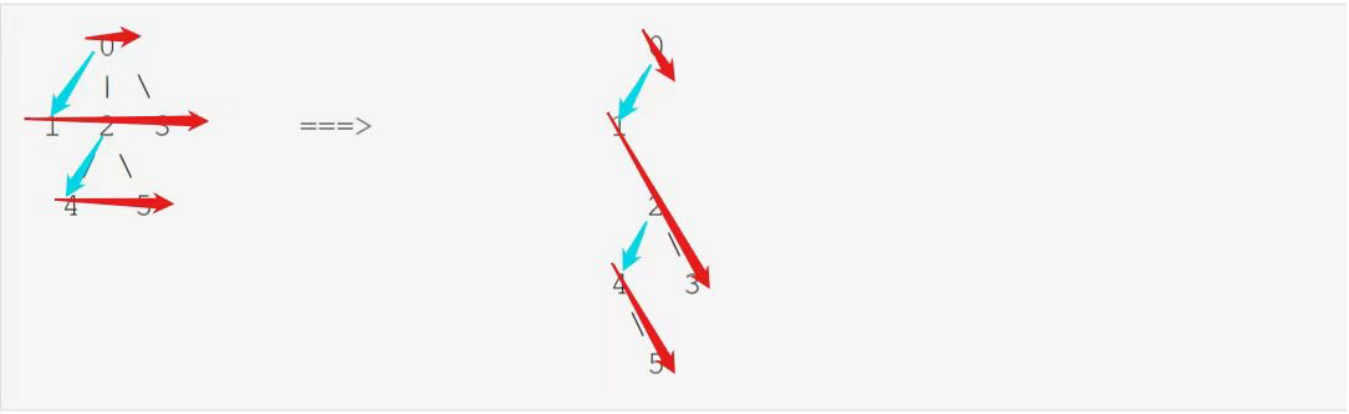
```
l=list(dict.fromkeys(l))
```

9、字典转字符串输出：

```
print(' '.join(map(str, traversal)))
```

10、转化成二叉树：

我们都知道用"左儿子右兄弟"的方法可以将一棵一般的树转换为二叉树，如：



现在请你将一些一般的树用这种方法转换为二叉树，并输出转换前和转换后树的高度。

11、树的深搜：

深度优先搜索（Depth-First Search，DFS）是一种用于图和树的遍历算法。在树中，深度优先搜索从根节点开始，沿着树的深度尽可能远的搜索每条分支，直到无法继续为止，然后回溯到上一个节点，继续搜索未被访问的分支，直到整棵树都被搜索完毕。

12、enumerate( )用法：

`enumerate()` 函数在 Python 中用于将一个可迭代对象包装成一个枚举对象，同时提供了索引和对应值的迭代。在这个情境下，`self.child` 应该是一个可迭代对象，而 `enumerate(self.child)` 则会返回一个枚举对象，其中包含了 `self.child` 中每个元素的索引和对应的值。

例如，如果 `self.child` 是一个列表，那么 `enumerate(self.child)` 将返回一个包含元组 `(index, value)` 的枚举对象，其中 `index` 是索引，`value` 是 `self.child` 中对应的值。

13、tip:

在Python中，`print(*ans)` 的意思是将列表、元组或者其他可迭代对象 `ans` 中的元素打印出来，使用空格分隔。相当于将 `ans` 中的元素逐个打印，并以空格分隔。

举个例子，如果 `ans` 是一个包含元素 `1, 2, 3` 的列表，那么 `print(*ans)` 将会打印出：

```
1 2 3
```

这种写法相当于把 `ans` 中的元素作为 `print` 函数的参数进行传递，而不需要使用循环或者其他操作来逐个打印元素。