Rope数据结构分析

## Rope基本介绍

字符串连接是一种十分常见的操作。当字符串以传统的方式（比如字符数组）存储的时候，连接操作需要花费 O(n) 的时间；而Rope可以高效处理字符串的拼接、查询、删除、及随机访问。Rope的一个典型应用场景是：在一个文本编辑器程序里，用来保存较长的文本字符串。图1给出了Rope结构存储的字符串“Hello\_my\_name\_is\_Simon”。

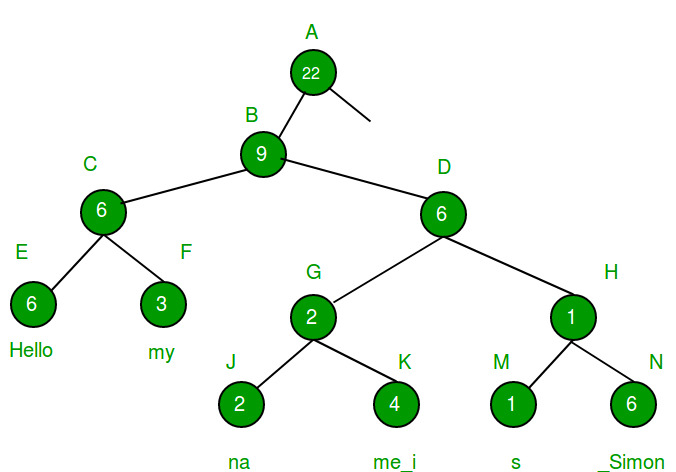


图1. Rope数据结构示例

从图中可看出，Rope是一个二叉树，叶节点包含的是字符串的子串，非叶节点是权重，其值等于左子树叶节点的所有字符之和。一个字符串被分割为两部分，左子树包含了字符串左部分，右子树包含了字符串的右部分，这些部分都是由用户自己确定的。

## Rope的Python实现

Rope数据结构由于是字符串的另一实现，因此其需要支持字符串的基本操作，即：索引，拼接以及分割。

### 索引 index()

该方法返回位置为i的字符。在Python中我使用了\_\_getitem()\_\_ 方法使其更符合Python语法使用。

### 拼接 concat()

该方法拼接两个 Rope结构的字符串至一个Rope中。在Python中我使用 \_\_add\_\_() 方法进行重载，使其操作更符合Python语法。

### 分割 split()

该方法分离字符串s 在i 处的两个字符串，返回两个Rope。

以上代码的实现见附录文件。

## Rope的性能分析

对于索引操作，字符数组的索引复杂度为 ，块状链表如果设置的分块数目得当时，时间复杂度为 。相比之下，Rope使用的基于递归的搜索，时间复杂度为，因为索引值只需同最多层的数据进行比较进而确定搜索索引的位置。

对于拼接操作，字符数组的索引复杂度为 ，因为内存不连续导致需要申请一块新的内存空间进行复制存储；块状链表如果设置的分块数目得当时，时间复杂度为 ；Rope的拼接本身仅需要，因为只需要一个生成一个新的节点并把左右分支分别指向带拼接的Rope即可，但是如果是基于平衡树实现的Rope那么拼接后还需要计算这个新节点的权重，该计算过程的复杂度为，因此整体来看需要的复杂度。

对于分割操作，字符数组的时间复杂度为，块状链表为，Rope的时间复杂度为，但是其分割后的两个子字符串可能还需要进行数据结构的平衡以达到Rope的数据结构。但该平衡过程最差情况下复杂度为，因此整个Rope的分割操作时间复杂度可记为。性能分析总结如表1所示：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 字符数组 | 块状链表 | Rope |
| 索引 |  |  |  |
| 拼接 |  |  |  |
| 分割 |  |  |  |

表1. 字符串常见实现结构性能对比

总结来说，Rope在不需要连续的内存存储空间，在面对文本编辑这种n随时变化且规模不断增大的使用场景他能更好地发挥作用，因为其对应的增加与删除（拼接与分割的组合）所需要的时间更短，但是他需要更多的内存资源用于维护节点的权重，且代码实现与维护也比较麻烦。

## 参考文献

1. [Ropes Data Structure (Fast String Concatenation) - GeeksforGeeks](https://www.geeksforgeeks.org/ropes-data-structure-fast-string-concatenation/)
2. [Rope --高效字符串处理数据结构\_ai\_xiangjuan的博客-CSDN博客\_rope数据结构](https://blog.csdn.net/ai_xiangjuan/article/details/79246289)