

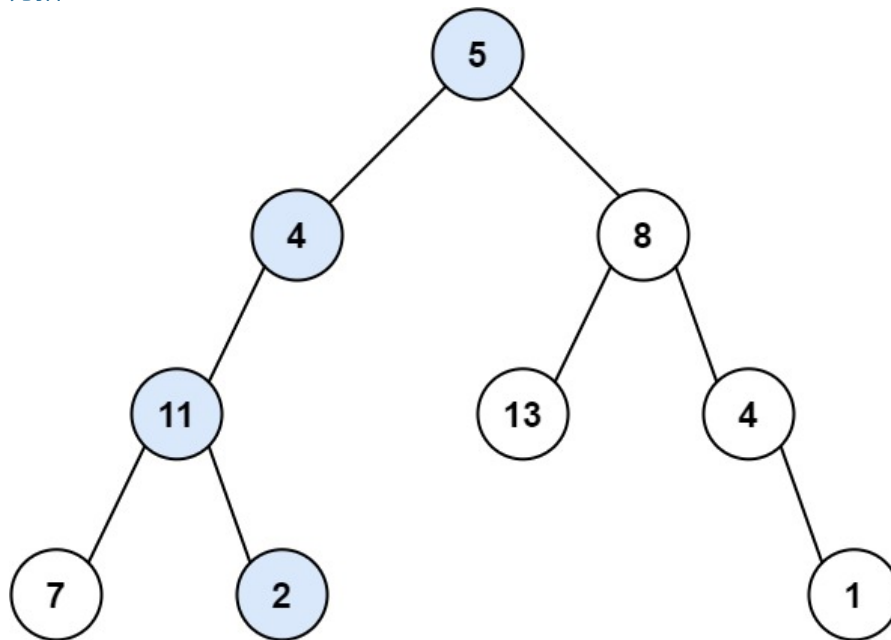
0520

路径总和

给你二叉树的根节点 `root` 和一个表示目标和的整数 `targetSum`，判断该树中是否存在根节点到叶子节点的路径，这条路径上所有节点值相加等于目标和 `targetSum`。

叶子节点 是指没有子节点的节点。

力扣



输入: `root = [5,4,8,11,null,13,4,7,2,null,null,null,1]`, `targetSum = 22` 输出: `true`

完成代码

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def hasPathSum(self, root: TreeNode, targetSum: int) -> bool:
        def ispathsum(root, cursum, target):
            if root is None:
                ##root为空
                return False
            cursum += root.val
            if root.left is None and root.right is None:
```

```

if root.left is None and root.right is None:
    return cursum == target
return ispathsum(root.left,cursum,target) or
ispathsum(root.right,cursum,target)
###时间复杂度为树的遍历，O(N) N为树的节点，空间复杂度为树的高度
O(H)
return ispathsum(root,0,targetSum)

```

截图

执行结果: **通过** 显示详情

执行用时: **48 ms** , 在所有 Python3 提交中击败了 **85.17%** 的用户

内存消耗: **16.4 MB** , 在所有 Python3 提交中击败了 **97.97%** 的用户

标签: 深度优先搜索, 树, 递归

与题解, 分享我的解法

提交结果	执行用时	内存消耗	语言	提交时间	备注
通过	48 ms	16.4 MB	Python3	2021/05/20 14:01	考虑到负数情况...

```

1 # Definition for a binary tree node.
2 # class TreeNode:
3 #     def __init__(self, val=0, left=None, right=None):
4 #         self.val = val
5 #         self.left = left
6 #         self.right = right
7 class Solution:
8     def hasPathSum(self, root: TreeNode, targetSum: int) -> bool:
9         def ispathsum(root,cursum,target):
10             if root is None:
11                 return False
12             cursum += root.val
13             if root.left is None and root.right is None:
14                 return cursum == target
15             return ispathsum(root.left,cursum,target) or ispathsum(root.right,cursum,target)
16         return ispathsum(root,0,targetSum)
17
18 #####时间复杂度为树的遍历，O(N) N为树的节点，空间复杂度为树的高度
19 O(H)
20 return ispathsum(root,0,targetSum)

```

10亿个数中如何高效地找到最大的一个数以及最大的第K个数

关于大数据量的topK问题，涉及到无法一次性将所有数据读入内存，float在32位处理器占用4个字节，如果是1亿，需要占用400M内存，10亿则需要占用4G内存

- 考虑方法 分治 + hash + 小顶堆

1.找到最大的一个数

将10亿数据均匀的分成1000份，每份为100 0000个数，每次需要占用4M内存，选出每份中的最大数，将这1000个最大数再进行排序比较

2.找到最大的K个数

- 通过hash法将这10亿个数字去除重复
- 然后分治，分解成多个小数据集
- 每份数据处理用小顶堆，先取一部分（K）数建堆，和堆顶元素比较，如果比堆顶（最小）元素小，则下一个，如果大，则重新调整小顶堆
- 最后在所有的topK中建堆
- 如果是top词频可以使用分治+ Trie树/hash +小顶堆
- 时间复杂度：O（NlogK）建堆复杂度O（K）

3.不容的应用场景的解决方案。

- (1) 单机+单核+足够大内存

如果有这么大内存，直接在内存中对查询次进行排序，顺序遍历找出出现频率最大的即可。也可以先用HashMap求出每个词出现的频率，然后求出频率最大的10个词。

(2) 单机+多核+足够大内存

这时可以直接在内存总使用Hash方法将数据划分成 n 个partition，每个partition交给一个线程处理，最后一个线程将结果归并。每个线程的处理速度可能不同，快的线程需要等待慢的线程，最终的处理速度取决于慢的线程。而针对此问题，解决的方法是，将数据划分成 $c \times n$ 个partition ($c > 1$)，每个线程处理完当前partition后主动取下一个partition继续处理，知道所有数据处理完毕，最后由一个线程进行归并。

(3) 单机+单核+受限内存

这种情况下，需要将原数据文件切割成一个一个小文件，将原文件中的数据切割成 M 个小文件，如果小文件仍大于内存大小，继续采用Hash的方法对数据文件进行分割，知道每个小文件小于内存大小，这样每个文件可放到内存中处理。

(4) 多机+受限内存

这种情况，为了合理利用多台机器的资源，可将数据分发到多台机器上，每台机器解决本地的数据。可采用hash+socket方法进行数据分发。