深度优先和广度优先搜索 DFS, BFS

笔记本: 程序设计与算法
创建时间: 2020/9/23 0:54 **更新时间:** 2020/9/25 1:06
作者: Jie Zhong
URL: https://shimo.im/docs/UdY2UUKtliYXmk8t/read

深度优先和广度优先搜索 DFS, BFS
搜索 - 遍历

- 每个结点都要访问一次
- 每个结点仅仅要方位一次

深度优先搜索(Depth first search) 模板

```
#递归
visited = set()

def dfs(node, visited):
    if node in visited: # terminator
        # already visited
        return

    visited.add(node)

    # process current node here.
    ...
    for next_node in node.children():
        if next_node not in visited:
        dfs(next_node, visited)
```

```
#非递归
def DFS(self, tree):

    if tree.root is None:
    return []

    visited, stack = [], [tree.root]

    while stack:
        node = stack.pop()
        visited.add(node)

        process (node)
        nodes = generate_related_nodes(node)
        stack.push(nodes)

    # other processing work
    ...
```

广度优先搜索(Breadth first search)模板

```
def BFS(graph, start, end):
    queue = []
    queue.append([start])
    visited.add(start)

    while  queue:
        node = queue.pop()
        visited.add(node)
        process(node)
        nodes =  generate_related_nodes(node)
        queue.push(nodes)
```

贪心算法 Greedy
贪心算法是一种在每一步选择中都采取在当前状态下最好或最优（即最有 利）的选择，从而希望导致结果是全局最好或最优的算法。

贪心算法与动态规划的不同在于它对每个子问题的解决方案都做出选择，不 能回退。动态规划则会保存以前的运算结果，并根据以前的结果对当前进行 选择，有回退功能。

- 贪心法可以解决一些最优化问题, 然而对于工程和生活中的问题，贪心法一般不能得到我们所要求的答案。
  - 求图中的最小生成树、
  - 求哈夫曼编码
- 一旦一个问题可以通过贪心法来解决，那么贪心法一般是解决这个问题的最 好办法。由于贪心法的高效性以及其所求得的答案比较接近最优结果，贪心 法也可以用作辅助算法或者直接解决一些要求结果不特别精确的问题。

二分查找
二分查找的前提条件

1. 目标函数单调性（单调递增或递减）
2. 存在上下界(bound)
3. 能够通过索引访问(index accessible)

代码模板

```
// Java
public int binarySearch(int[] array, int target) {
    int left = 0, right = array.length - 1, mid;
    while (left <= right) {
        mid = (right - left) / 2 + left;

        if (array[mid] == target) {
            return mid;
        } else if (array[mid] > target) {
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }
    return -1;
}
```

分析题： 使用二分查找，寻找一个半有序数组 [4, 5, 6, 7, 0, 1, 2] 中间无序的地方
（说明：同学们可以将自己的思路、代码写在学习总结中）
问题描述：寻找一个半有序数组 [4, 5, 6, 7, 0, 1, 2] 中间无序的地方，此数组无序的地方在下标为4的位置，也就是7后面是0，突然由升序变为降序
分析：

- 寻找半有序数组中间无序的地方相当于寻找到数组的最小值，[4, 5, 6, 7, 0, 1, 2] 的最小值是 0
- 使用二分法，
  - 初始l = 0, r = nums.length
  - 循环当 l < r，终止条件为左指针右指针重合，此位置为就是数组的最小值，开始变无序的地方
  - 如果nums[mid] > nums[r] 说明无序的地方在后半部分，l = mid + 1
  - 如果nums[mid] < nums[left] 说明无序的地方在前半部分, r = mid

代码：

```
public int findMin(int[] nums) {
        int l = 0;
        int r = nums.length - 1;
        while (l < r) {
            int mid = l + (r -l)/2;
```

```
            if (nums[r] < nums[mid]) {//后半部分无序，min一定出现在后半部分
                l = mid + 1;
            } else {//后半部分有序(包括前半部分有序或无序），min一定出现在前半部分
                r = mid;
            }
        }
        return nums[l];
    }
```