

算法笔记

笔记本： 我的第一个笔记本

创建时间： 2020/6/22 12:57

更新时间： 2020/7/19 18:48

作者： So fly

URL: <https://shimo.im/docs/ZBghMEZWix0Lc2jQ/read>

递归

循环的一种特殊形式，通过函数体来进行的循环。

递归代码的基本逻辑

基本逻辑与伪代码

1. 首先写出递归的终止条件；
2. 写出每层递归层，需要执行的相关逻辑代码；
3. 写出进入下一层的代码，调用函数本身，并传入参数；
4. 如果有需要，整理或者清理当前层的状态。

```
public void recur ( int level, int param) {  
    //terminator  
    if (level > MAX_VALUE)  
        return;  
    //process current logic  
    process(level, param);  
    //drill down  
    recur(level:level + 1, newParam);  
    //restore current satus}
```

注意要点

1. 不要进行人肉递归；

2. 找到最近最简方法，即找到可重复单元；
3. 使用数学归纳法的思维；

分治和回溯

1. 递归的一种特殊形式，在递归的模板上增加了分而治之的一步。
 2. “回溯”算法也叫“回溯搜索”算法，主要用于在一个庞大的空间里搜索我们所需的问题的解。我们每天使用的“搜索引擎”就是帮助我们在庞大的互联网上搜索我们需要的信息。“搜索”引擎的“搜索”和“回溯搜索”算法的“搜索”意思是一样的。
2. “回溯”指的是“状态重置”，可以理解为“回到过去”、“恢复现场”，是在编码的过程中，是为了节约空间而使用的一种技巧。而回溯其实是“深度优先遍历”特有的一种现象。之所以是“深度优先遍历”，是因为我们要解决的问题通常是在一棵树上完成的，在这棵树上搜索需要的答案，一般使用深度优先遍历。

```
public void divide_conquer(problem, param1, param2, ....) {  
    //terminator  
    //process: split big problem(分治的关键)  
    //drill down:  
    //1. conquer subproblems  
    //2. process and generate the final result  
    //revert the current level states  
}
```

深度优先、广度优先搜索

搜索遍历的特性

1. 每个节点都要访问一次；
2. 每个节点仅访问一次；
3. 对于节点的访问顺序不限，可以分为：
 - 深度优先 dfs
 - 广度优先 bfs
 - 优先级优先

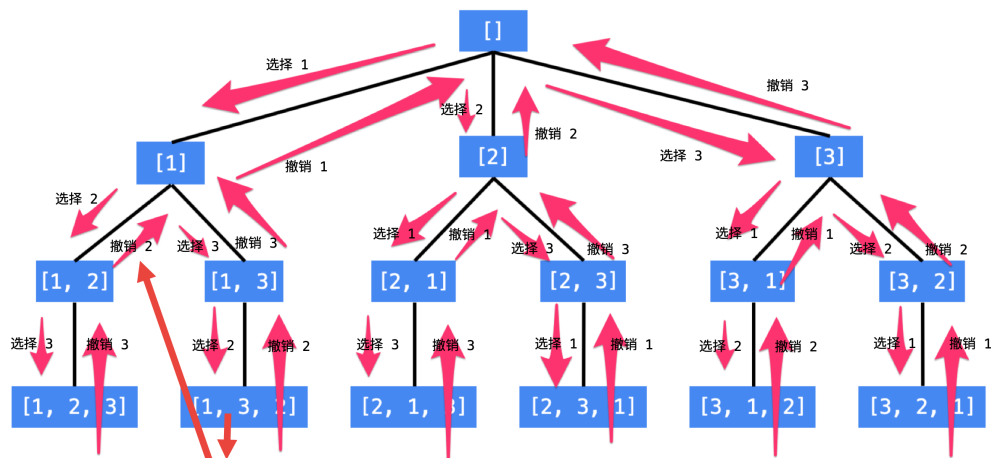
深度优先搜索

深度优先遍历示例代码

```
public class dfs(node) {  
    if (node contains in visited) {  
        //already visited  
        return;  
    }  
    visited.add(node);  
    //process logic  
    //drill down  
    dfs(node.left);  
    dfs(node.right);  
}
```

深度优先的遍历

- 从根节点的子节点中一直遍历至叶子节点，再进行回溯，从新进行遍历。可以用栈的数据结构进行模拟。
- 在遍历的过程中，从深层结点回到浅层结点的过程中所做的操作就叫“回溯”。



广度优先遍历

广度优先遍历示例代码

```
def BFS(graph, start, end):
```

```
visited = set()
queue = []
queue.append([start])

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

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

# other processing work
```

广度优先的遍历

将根节点放入队列中，再确定跟节点的子节点，将这些子节点再放入队列中，使用递归的方式，分别再确认出子节点的子节点，直到递归到叶子节点。

可以使用队列，或者双端队列来进行模拟。

贪心算法

- 在每一步选择中都采取在当前状态下最好或最优，即最有利选择，从而希望导致结果是全局最优的一种算法
- 特点1：贪心算法是在局部直接做出最优选择，且不能回退；
- 特点2：贪心算法效率较高，且如果一个问题可以用贪心算法来进行解决，那么贪心算法得到得结果一般都是接近最优解的。
- 贪心算法的适用场景：问题能够分解成子问题来解决，子问题的最优解能够地推到最终问题的最优解。这种子问题最优解称为最优子结构。
- 适用贪心算法的重点：证明该问题如何能够使用贪心算法得到最优解。

二分查找

二分查找的前提

- 目标函数单调性（单调递增递减），即整个数组是有序的；
- 存在上下界

- 能够通过索引访问

二分查找的模板

```
left, right = 0, len(array) - 1
while left <= right:
    mid = (left + right) // 2
    if array[mid] == target:
        # find the target!!
        break or return result
    elif array[mid] < target:
        left = mid + 1
    else:
        right = mid - 1
```

动态规划

动态规划的要点

1. 动态规划和递归或者分治没有根本上的区别，DP关键是看有无最优子结构；
2. DP与其他递归算法的共性：**找到重复子结构**；
3. DP与其他递归算法的差异：存在最优子结构，中途可以淘汰次优解。
4. 递推与递归的区别：递推是找到最小重复子问题后，自底向上进行计算，递归则是自顶向下进行计算。