数据结构与算法学习笔记

引言

我大概是从大一的时候的时候就开始看算法书，我还记得那个时候一道分解自然数的算法，我真的是死活都弄不明白，递归对于我来说太过于难以理解，简直就是违反了天理啊，至少不是一个普通人的思维。时至今日，即将成为一名研究生，暑假之余重新拾起算法书，对于以前令人头疼的图论，搜索，动态规划，我竟然发现现在自己可以很容易就能理解并写出代码，虽然也是要调试很久（我可能需要写它20遍闭着眼睛也能写出来吧）我想这就是进步，只要坚持不放弃人总是会前进的，哪怕是前进得缓慢，一步一步走总是会到达终点。学着用计算机的思维去看待一个问题，什么是计算机思维，我认为就是那一堆循环分支指令和函数调用，计算机和人不同特有的强大的计算能力可以让我们重复去做一件事，那我们的问题是否也可以分割成一些相同的小问题让计算机循环去做，这就是所谓的分治的思想，计算机函数调用可以调用自己本身，我不知道这是哪个天才设计的计算机语言，这就是伟大的递归，递归的数学模型其实就是归纳法，如果子问题存在嵌套的关系并有相同的模型那就可以使用递归。最后最近学的Python做一张本篇文章关键词的WordCloud如下图所示。



一、暴力搜索与回溯

1. 概述

回溯算法实际上一个类似枚举的搜索尝试过程，主要是在搜索尝试过程中寻找问题的解，当发现已不满足求解条件时，就“回溯”返回，尝试别的路径。回溯法是一种选优[搜索](https://baike.baidu.com/item/%E6%90%9C%E7%B4%A2/2791632" \t "https://baike.baidu.com/item/%E5%9B%9E%E6%BA%AF%E7%AE%97%E6%B3%95/_blank)法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法，而满足回溯条件的某个状态的点称为“回溯点”。许多复杂的，规模较大的问题都可以使用回溯法，有“通用解题方法”的美称。

1. 搜索分类

2.1 DFS深度优先搜索

1、全排列

2、求集合的子集（可用二进制法转换为全排列问题）

3、有几种分法的问题（分解自然数、分苹果，八皇后问题）

2.2 BFS 广度优先搜索

1、最短路径问题（打印路径步数）

1. 搜索剪枝策略

搜索算法可以算是一种万能解法，但是往往题目的解空间是呈指数性增长的，特别是递归形式的深度优先搜索往往会出现栈溢出的现象，需要对算法进行优化以获得更好的性能。在搜索算法优化中，剪枝，就是通过某种判断，避免一些不必要的遍历过程，形象的说，就是剪去了搜索树中的某些“枝条”，故称剪枝。应用剪枝优化的核心问题是设计剪枝判断方法，即确定哪些枝条应当舍弃，哪些枝条应当保留的方法。

剪枝的方法可以分为两类，一是可行性剪枝，二是最优性剪枝。可行性剪枝就是判断继续搜索是否得到答案，如果 不能则直接回溯，最优性剪枝又称为上下界剪枝，是一种重要的搜索剪枝策略。它记录当前结果的最优值，如果当前节点 无法产生比当前最优解更优的解，则可以提前回溯。

4、算法框架

当搜索树的深度在三层以下的枚举直接采用for循环比较简单，若大于三层的则利用递归的深度优先搜索。

4.1深度搜索框架

**procedure** **try**(i:**integer**);

**begin**

**if** i>n **then** 输出结果

**else** **for** j:=下界 **to** 上界 **do**

x[i]:=h[j];

**if** 可行{满足限界函数和约束条件} **then**

**begin**

置值;

**try**(i+1);

**end**;

**end**;

4.2 宽度搜索框架

**procedure** BFS(i:**integer**);

**begin**

Queue p

第一个节点入队；

设置第一个节点的参数（已访问过等信息）

While 队列不为空 then

取队首元素；

for 能够到达的节点

If 满足约束条件

入队；

If 到达目标节点 break

**end**;

1. **贪心与动态规划**
2. 贪心与动态规划的比较

贪心和动态规划都是用解决最优化问题的方法，它们的思想都是用局部最优解去递推出全局最优解。贪心特在，可以证明，每一个子树的根的值不取决于下面叶子的值，而只取决于当前问题的状况。换句话说，不需要知道一个节点所有子树的情况，就可以求出这个节点的值。由于贪心算法的这个特性，它对解空间树的遍历不需要自底向上，而只需要自根开始，选择最优的路，一直走到底就可以了。贪心算法不一定能找到最优解，动态规划可以找到最优解。

1. 动态规划分类及典型例题

2.1 背包问题

1、0-1背包问题

### 01背包的状态转换方程 f[i,j] = Max{ f[i-1,j-Wi]+Pi( j >= Wi ),  f[i-1,j] }

f[i,j]表示在前i件物品中选择若干件放在承重为 j 的背包中，可以取得的最大价值。

2、部分背包问题（每种物品可以选多件）

**状态转移方程 f [ i ] [ j ]=Max{f[i-1][j-n\*wi}+n\*pi(0<=n<=n[i]);**

第i件物品有n[i]+1种选法

3、 完全背包问题（每种物品可以选无限件）

**状态转移方程 f [ i ] [ j ]=Max{f[i-1][j-k\*wi}+k\*pi(0<=k<=j/w[i]);**

第i件物品有n[i]+1种选法

变形: weight[i]：可以换成其他的约束条件，比如规定选几个

Value[i]：当有正负的时候，也要维护一个最小的价值数组

前i物品：当出现在选择物品之间有其他约束条件的时候，比如不能连续选，就要以i为结尾

在不止一种选择的时候要循环，或者要遍历前面的f[i]

2.2 区间模式（一维dp，二维dp）

最长连续递增子序列

L(i)=0 (a[i]<a[i-1]);

L(i)=L(i-1)+1; (a[i]>a[i-1]);

变形：当不要求连续的时候 ：L(i)=Max{L(i-k)+1,0<k<=i-1,a[i]>a[i-k]};

最长匹配子串

c[i][j]=c[i-1][j-1]+1(c1[i]=c2[j])

c[i][j]=0;（连续）c[i][j]=Max(c[i-1][ j ],c[ i ][ j-1] )(不要求连续)

字符串相似度（通过最少的添加删除修改的次数使两个字符串一样）

c[i][j]=c[i-1][j-1]+1(c1[i]=c2[j])

c[i][j]=Max{c[i-1][j-1],c[i-1][j],c[i][j-1]}//修改，删除

1. **图的基本算法**

1、深度优先遍历，递归和非递归

2、广度优先遍历

3、Dijkstra求最短路问题

算法简单描述

1. 输入：一个加权连通图，其顶点集合为V，边集合为e，起始顶点Vs;
2. 初始化: 标记各顶点的访问情况，每个其余顶点到源顶点的距离;
3. 循环图的顶点数次：

每次从还没有访问过的节点中选出离起始顶点距离最近的顶点Vm;

标记Vm已访问；

对所有还没有访问过的节点，如果加入顶点Vm后离源顶点的距离更近则更新到源顶点的距离；

4、求最小生成树

4.1 Kruskal算法 （并查集） //无向图的最小生成树

**算法简单描述**

1. .记Graph中有v个顶点，e个边
2. .新建图Graphnew，Graphnew中拥有原图中相同的e个顶点，但没有边
3. .将原图Graph中所有e个边按权值从小到大排序
4. .循环：从权值最小的边开始遍历每条边 直至图Graph中所有的节点都在同一个连通分量中

if 这条边连接的两个节点于图Graphnew中不在同一个连通分量中

添加这条边到图Graphnew中

4.2 Prim算法

**算法简单描述**

1. .输入：一个加权连通图，其中顶点集合为V，边集合为E；
2. .初始化：Vnew = {x}，其中x为集合V中的任一节点（起始点），Enew = {},为空；

3).重复下列操作，直到Vnew = V：

a.在集合E中选取权值最小的边<u, v>，其中u为集合Vnew中的元素，而v不在Vnew集合当中，并且v∈V（如果存在有多条满足前述条件即具有相同权值的边，则可任意选取其中之一）；

b.将v加入集合Vnew中，将<u, v>边加入集合Enew中；

4).输出：使用集合Vnew和Enew来描述所得到的最小生成树。