## 所解决问题的分类

显然，使用回溯法解决的问题可以分为两大类：

1. 满足性问题：需要搜索出满足条件的所有解，这类问题最有代表性的是图的m着色问题
2. 最值问题：需要搜索出所有满足条件的解中的最值解，最有代表性的有：最大团问题

## 2.最值问题的实现总结

在最值问题中会搜索到满足条件的很多解，但是只选择一个最值解作为最终解，因此解决此类问题可以借鉴迭代法求最值的思路:使用cn记录当前搜索到解的值，使用bestn记录当前最优值，在if（i>n）中进行判断只有当cn>bestn是才进行更新。

## 满足性问题的实现总结

一旦到达叶子结点，就表明搜索到了一个满足条件的解，这样就可以进行输出了。

## 构造最优解的实现总结

在子集树问题中，最优解的构造是相对简单的：使用数组X[n]来记录，如果选择的是左边，取值1，如果选择的是右边，则取值为0.（因为子集树正好是完全二叉树，所以每个结点只连接两条边，所以对应的取值也只有两个）

## 回滚思想在回溯法中的广泛使用（非常重要的思想）

在数据库中我学习了事务的概念，在操作系统课上我学习了原子操作，它们两者有着共同的特点就是：操作如果成功就执行下去；如果失败就要回滚，即把之前的操作都要撤销掉，使对象的状态回到没有执行这个操作的状态。这个思想真是太重要了，这不，在回溯法中我就看到了它的影子。

void Loading::BackTrack(int i){//在本函数中，i是从0开始的

r -= w[i];

if (cw + w[i] <= c){//搜索左子树

x[i] = 1;

cw += w[i];

BackTrack(i + 1);

x[i] = 1;

cw -= w[i];

}

if (cw + r > bestw){ x[i] = 0;

BackTrack(i + 1);

}

r += w[i];

}

请注意上面相同颜色的部分：发现它们执行的是完全相反的操作，也就是后句的操作是对前句操作的撤销。

我们还能够注意到的一点是：递归操作放在了回滚操作之间，下面解析这样写的精妙所在：

如果BackTrack(i + 1)能够成功地执行下去，这就表明操作成功，则会使对象的相应状态发生改变（r -= w[i]; x[i] = 1; cw -= w[i];）。如果BackTrack(i + 1)没有成功地执行下去，则前面的操作需要撤销掉，即进行回滚。

BackTrack(i + 1)如果能够成功地层层向下执行到叶子结点，则就知道了一个完整解。