回溯法感想

回溯法是一个很好用、很有效的方法，也是一个相对比较简单的算法。其实在接触回溯法之前，我习惯性称这个方法为深度优先搜索，大概意思就是，找到所有能够满足题目条件的解法。

在系统的学习回溯法之后，我对回溯法又有了新的认识。首先是回溯法的解空间树，分为子集树和排列树。子集树的算法时间复杂度是O(M^n)，也就是每一棵子树都要展开M个分枝，然后继续搜索下去；排列树的时间复杂度是O(n!)，其实就是一个全排列的模板。

严谨地说，回溯法有"通用的解题法"之称。用它可以系统地搜索一个问题的所在解或任一解。回溯法是一个即带有系统性又带有跳跃性的所搜算法。在包含问题的所有解的解空间树中，按照深度优先搜索的策略，从根结点出发深度探索解空间树。当探索到某一结点时，要先判断该结点是否包含问题的解，如果包含，就从该结点出发继续探索下去，如果该结点不包含问题的解，则逐层向其祖先结点回溯。（其实回溯法就是对隐式图的深度优先搜索算法）。若用回溯法求问题的所有解时，要回溯到根，且根结点的所有可行的子树都要已被搜索遍才结束。而若使用回溯法求任一个解时，只要搜索到问题的一个解就可以结束。

在回溯法里面还有一个很重要的内容，就是剪枝。剪枝函数又分为约束函数和限界函数，约束函数就是减去不符合题目条件的分枝，限界函数就是减去不可能成为最优解的函数。在实际的编程过程中，这两个函数往往就封装在一个函数中，在进入下一层递归之前，进行判断调用。

回溯法还有一个优点，不管是子集树还是排列树，解题都有固定的模板，十分方便。

```
// 排列树
void backtrack (int t)
{
  if (t>n) output(x);
  else
      for (int i=t;i<=n;i++)
     {
        swap(x[t], x[i]);
        if (legal(t))
            backtrack(t+1);
        swap(x[t], x[i]);
      }
}
```

```
// 子集树
void backtrack (int t)
{
  if (t>n) output(x);
    else
      for (int i=0;i<=1;i++) {
        x[t]=i;
        if (legal(t)) backtrack(t+1);
      }
}
```

需要注意，期末考试要考察绘制可行解的树，在平时要多加练习，加深理解。