**算法实现题 5-1**

**子集和问题的一个实例为〈S,t〉。其中，S={ x1，x2，…，xn}是一个正整数的集合，c 是一个正整数。子集和问题判定是否存在 S 的一个子集 S1，使得 S1 中的所有元素之和等于 c。试设计一个解子集和问题的回溯法。**

1、构造问题的解空间

该问题解空间的实质是子集树的形式。设待加入的数的集合为{1,2,3……n}那么对于每一个数都有要加入和不要加入两种情况，所以问题的解空间可以看做是 n+1 层的完全二叉树，对这个 n+1 层的完全二叉树进行遍历就可以得到问题的可行解。

2、剪枝函数设计

因为该问题要求的是一个子集，使得该子集中的元素和为 c。那么可以知道我们要求的是问题的一个可行解，那么采用约束的方法进行剪枝函数的设计。当我们遍历到问题解空间的第 i 层的时候（i<n），如果这个时候的数值加上之前已经计算出来的总和所得的结果大于 c 的时候，那么就把以第 i 层以这个数为根的子数剪掉。

3、算法结束条件

当遍历到叶节点的时候，如果这个时候所求得的总和与 c 相等，那么就就输出问题的可行解。

**算法实现题 5-3**

**设某一机器由 n 个部件组成，每一种部件都可以从 m 个不同的供应商处购得。设 wij 是从供应商 j 处购得的部件 i 的重量，cij 是相应的价格。**

**试设计一个算法，给出总价格不超过 c 的最小重量机器设计。**

本题解法类似背包问题，即最优规划，采用回溯法求解。用动态数组存放选购方案，通过递归函数进行零件的搜索和方案的剪枝，最终得到最小重量的设计方案。

由于题目已经给出总价格的上限，因此算法通过使用回溯来选择合适的机器使得在总价格不超过 d 时得到的机器重量最小。首先初始化当前价格 cp=0，当前重量 cw=0，此外，还要设置一个变量 sum 表示选择机器的总重量，初始化其为每个部件从 1 号供应商购买的重量。在循环选择 i 号机器时，判断从 j 号供应商购买机器后的价格是否大于总价格，如果不大于则选择，否则不选，继续选择下一供应商进行判断。在得到一个合适的供应商后，继续选择下一机器的供应商，从第一个选到最后一个供应商。当所有机器选择结束后，判断得到的总重量是否比之前的 sum 小，如果小就赋给 sum，然后从这一步开始，回溯到上一机器，选择下一合适供应商，继续搜索可行解，直到将整个排列树搜索完毕。这样，最终得到的 sum 即为最优解。

当然，考虑到算法的时间复杂度，还可以加上一个剪枝条件，即在每次选择某一机器时，再判断选择后的当前重量是否已经大于之前的 sum，如果大于就没必要继续搜索了，因为得到的肯定不是最优解。

算法：

```
void search(int i){
   if(i>n){
   if(cw<sum)
      sum = cw;
      return ;
}
for(int j=1;j<=m;j++)
{
```

```
        cw+=w[i][j];
        cp+=c[i][j];
        if(cw<sum && cp<=d)
            backtrack(i+1);
            cw-=w[i][j];
            cp-=c[i][j];
        }
    }
```

此函数遍历排列树的时间复杂度为 O(n!)，故该算法的时间复杂度为 O(n!)。

**算法实现题 5-6**

**设 S 是正整数集合，S 是一个无和集，当且仅当 x,y 属于 S，蕴含 x+y 不属于 S。对于任意的正整数 k，如果可将{1,2……k}划分为 n 个无和子集 S1,S2,S3……Sn，称正整数 k 是 n 可分的。记 F(n)=max{k|k 是 n 可分的}。试设计一个算法，对任意给定的 n，计算 F(n)的值。**

此题是子集选取问题，其解空间是一棵子集树，可以套用搜索子集树的回溯法框架。由于搜索空间很大，用搜索时间控制搜索深度，即超过一定时间就把这个分枝剪掉。

每个数 i 依次放入当前情况下能放的集合，继续递归，放不下去了则返回到上一个数，尝试放入另一个集合，这样遍历到所有需要的情况。

约束条件也就是剪枝条件是 x+y 不属于当前子集（x，y 分别属于当前子集），结束条件是搜索时间上限。

```
void operation(int num) {//运行函数
  if(num > optimal_value) {//更新数据
    optimal_value = num - 1;
    for(int i = 1; i <= n; i++) {
      for(int j = 0; j <= assemble[i][0]; j++)
        optimal_assemble[i][j] = assemble[i][j];
      }
    }
    /*t1 = clock();
    if((t1 - t0) > 0.1)//将运行时间过长的分支裁掉
      return;
    t0 = t1;*/
    for(int i = 1; i <= n; i++){
      if(sum[i][num] == 0){
        sum[i][num] = 1;
        for(int j = 1; j <= assemble[i][0]; j++)
          sum[i][num + assemble[i][j]]++;
        assemble[i][++assemble[i][0]] = num;
        operation(num + 1);
        for(j = 1; j < assemble[i][0]; j++)
          sum[i][num + assemble[i][j]]--;
        sum[i][num] = 0;
        assemble[i][assemble[i][0]--] = 0;
      }
```

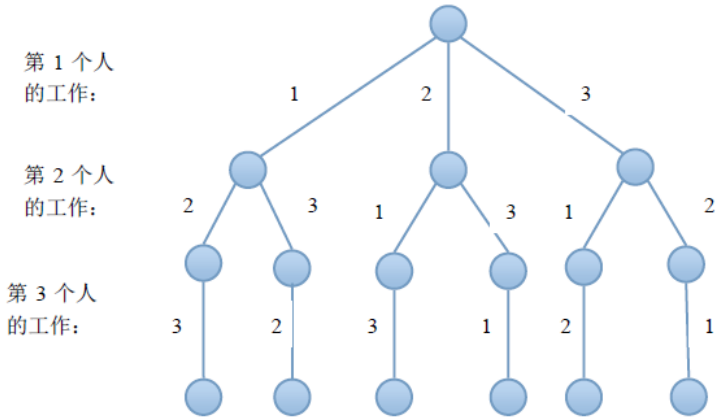
```
    }
}
```

**算法实现题 5-13**

**设有 n 件工作分配给 n 个工人，将工作 i 分配给第 j 个人所需的费用为 cij。试设计一个算法，为每一个人都分配 1 件不同的工作,并使总费用达到最小。**

1、解空间的定义和组织

使用回溯法对本题进行求解，需要遍历每一种工作分配情况，构造的解空间树的形式是一棵排列树。本题的最终解的形式应该是{x1,x2,...,xn}，其中 xi=1,2,...,n 表示分配给第 i 个人的工作号，解空间为{x1,x2,...,xn | x=1,2,...,n,1≤i≤n}。此问题的具体解空间树形态如下：




2、剪枝函数设计

由于本题要找的是最优解，所以应采用限界函数来删除不必要的搜索。

采用当前已知的最优解 S 为标准，对于有 n 个人和 n 件工作的工作分配问题来说，假设当前搜索层次为第 i 层（1<i<n)的某结点 E

如果当前路径长度＜S，则以 E 为扩展结点继续向下一层搜索；

如果当前路径长度＞＝S，则表明以 E 为根结点的子树中不包含最优解，将以 E 为根结点的子树中所有结点都置为死结点，算法向 E 最近的祖先活结点回溯。

3、算法的结束条件

由于本题要找的是最优解，所以只有遍历完所有路径，即所有结点都被置为死结点之后，算法流程结束。输出当前的最优解。

**算法实现题 5-20**

**原始部落 byteland 中的居民们为了争夺有限资源，经常发生冲突，几乎每个居民都有他的仇敌，部落酋长为了组织一支保卫部落的队伍，希望从部落的居民中选出最多的居民入伍，并保证队伍中任何 2 个人都不是仇敌。**

**给定 byteland 部落中居民间的仇敌关系，计算组成部落卫队的最佳方案。**

1、解空间

问题的解空间本质是子集树的形式。对于每个居民是否加入到卫队中都有加入和不加入两种情况，即 0 和 1 两种情况。所以，所以问题的解空间可以看做是 n+1 层的完全二叉树，对这个 n+1 层的完全二叉树进行遍历就可以得到问题的最优解。

2、剪枝函数的设计

因为该问题要求的是部落居民中选出最多的居民入伍，所以要求的是问题的最优解。那么剪枝函数的设计要从约束和限界两个方面进行考虑。

设居民为无向图的每个顶点，然后居民间如果是友好关系的话表示这两个顶点之间有边相连，如果是敌对关系的话两个顶点之间没有线相连。

从约束出发进行剪枝，与最大团问题类似，考虑新顶点，（也就是新加入的居名）同当前团中各顶点的连线是否在边集合 E 中，换句话说就是新加入的居民是否和之前已经加入的居民会存在敌对的关系，如果存在的话，就剪去。

从限界出发进行剪枝，与最大团问题类似，考虑当前已获得完全子图中顶点的数 cn(假设对于某一层的某个节点，目前已经分析过的顶点为 i 个，那么剩下的节点个数为 n-i，那么如果 cn+(n-i)要小于目前已经获得的最优解中的顶点数 bestn 的话，则说明以该顶点为根节点的子树中不会包含该问题的最优解，换句话说，你在某一层已经知道目前的最优的加入居民的个数，然后现在已经加入的居民的个数加上甚至剩下的所有居民的个数都不会比现在这个最优的居民的个数来得大的时候，就没有必要继续对这个顶点为根的子数进行遍历，所以剪去。

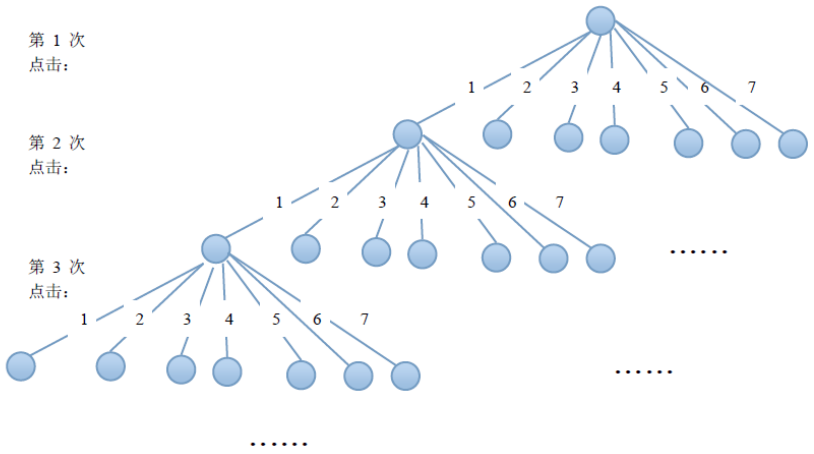
3、结束条件

到达叶子节点的时候，如果找到最优解，则输出，否则输出没有存在最优解。

**青蛙换位问题**

1、解空间

本题中的每一步操作实际上就是用鼠标点击一块石头，如果这块石头上有青蛙，且青蛙可以跳跃，则跳跃。使用回溯法对本题进行求解，每次有 7 种点击的情况，需要遍历每一种点击情况，并获得点击的序列，构造的解空间树的形式是一棵完全 7 叉树。本题的最终解的形式应该是{x1,x2,...,xn}，其中 xi=1,2,...,n 表示第 i 次鼠标应点击的石头号，解空间为{x1,x2,...,xn | x=1,2,...,7,1≤i≤n}。此问题的具体解空间树形态如下：




2、剪枝函数设计

由于本题要找的是可行解，所以应采用约束函数来删除不必要的搜索。具体约束条件如下：

a.当前石头上必须要有青蛙；

b.当前点击的青蛙面向的前方第一块或第二块石头为空石头；

c.青蛙不会跳出这 7 块石头。

3、结束条件

由于本题要找的是可行解，所以只要获得一个状态，满足左边的青蛙都到了右边，右边的青蛙都到了左边，就可以停止运算，并输出相应的路径了。

4、算法复杂度分析

本题的每一层跳跃都有 7 种可能，假设共跳了 n 次最终完成了转换，则此算法的复杂度为$O(7^n)$。