POJ2676

题目描述（POJ2676）：数独是一项非常简单的任务。如下图所示，一张 9 行 9 列的表被分成 9 个 3×3 的小方格。在一些单元格中写上十进制数字 1～9，其他单元格为空。目标是用 1～9 的数字填充空单元格，每个单元格一个数字，这样在每行、每列和每个被标记为 3×3 的子正方形内，所有 1～9 的数字都会出现。编写一个程序来解决给定的数独任务。

1		3				5		9
		2	1		9	4		
			7		4			
3			5		2			6
	6						5	
7			8		3			4
			4		1			
		9	2		5	8		
8		4				1		7

输入：输入数据将从测试用例的数量开始。对于每个测试用例，后面都跟 9 行，对应表的行。在每一行上都给出 9 个十进制数字，对应这一行中的单元格。如果单元格为空，则用 0 表示。

输出：对于每个测试用例，程序都应该以与输入数据相同的格式打印解决方案。空单元格必须按照规则填充。如果解决方案不是唯一的，那么程序可以打印其中任何一个。

输入样例	输出样例
1 103000509 002109400 000704000 300502006 060000050 700803004 000401000 009205800 804000107	143628579 572139468 986754231 391542786 468917352 725863914 237481695 619275843 54396127

题解：本题为数独游戏，为典型的**九宫格问题，可以采用回溯法搜索**。把一个 9 行 9 列的网格再细分为 9 个 3×3 的子网格，要求在每行、每列、每个子网格内都只能使用一次 1～9 的一个数字，即在每行、每列、每个子网格内都不允许出现相同的数字。

0 表示空白位置，其他均为已填入的数字。要求填完九宫格并输出（如果有多种结果，则只需输出其中一种）。如果给定的九宫格无法按要求填出来，则输出原来所输入的未填的九宫格。

用 3 个数组标记每行、每列、每个子网格已用的数字。

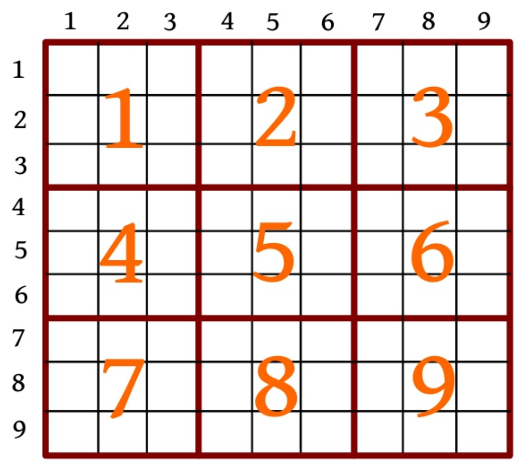
- row[i][x]：用于标记第 i 行中的数字 x 是否出现。
- col[j][y]：用于标记第 j 列中的数字 y 是否出现。
- grid[k][z]：标记第 k 个 3×3 子网格中的数字 z 是否出现。

row 和 col 的标记比较好处理，关键是找出 grid 子网格的序号与行 i、列 j 的关系，即要知道第 i 行 j 列的数字属于哪个子网格。

把一个 9 行 9 列的网格再细分为 9 个 3×3 的子网格，在每个子网格内都不允许出现相同的数字，那么我们将 9 个子网格编号为 1～9，在同一个子网格内不允许出现相同的数字。观察子网格的序号 k 与行 i、列 j 的关系：

- 如果把第 1～3 行转换为 0，第 4～5 行转换为 1，第 7～9 行转换为 2，则 a=(i-1)/3；
- 如果把第 1～3 列转换为 0，第 4～5 列转换为 1，第 7～9 列转换为 2，则 b=(j-1)/3。

行 i、列 j 对应的子网格编号 k=3×a+b+1=3×((i-1)/3)+(j-1)/3+1，如下图所示。



1. 算法设计

（1）预处理输入数据。

（2）从左上角(1,1)开始按行搜索，如果行 i=10，则说明找到答案，返回 1。

（3）如果 map[i][j]已填数字，则判断如果列 j=9，则说明处理到当前行的最后一列，继续下一行第 1 列的搜索，即 dfs(i+1,1)，否则在当前行的下一列搜索，即 dfs(i, j+1)。如果搜索成功，则返回 1，否则返回 0。

（4）如果 map[i][j]未填数字，则计算当前位置(i,j)所属子网格 k=3×((i-1)/3)+(j-1)/3+1。枚举数字 1～9 填空，如果当前行、当前列、当前子网格均未填该数字，则填写该数字并标记该数字已出现。如果判断列 j=9，则说明处理到当前行的最后一列，继续下一行第 1 列的搜索，即 dfs(i+1,1)，否则在当前行的下一列搜索，即 dfs(i, j+1)。如果搜索失败，则回溯归位，继续搜索，否则返回 1。

2. 算法实现

```
bool dfs(int i,int j){
    if(i==10)
        return 1;
    bool flag=0;
    if(map[i][j]){
        if(j==9)
            flag=dfs(i+1,1);
        else
            flag=dfs(i,j+1);
        return flag?1:0;
    }
    else{
        int k=3*((i-1)/3)+(j-1)/3+1;
        for(int x=1;x<=9;x++){//枚举数字 1～9 填空
            if(!row[i][x]&&!col[j][x]&&!grid[k][x]){
                map[i][j]=x;
                row[i][x]=1;
                col[j][x]=1;
                grid[k][x]=1;
                if(j==9)
                    flag=dfs(i+1,1);
                else
                    flag=dfs(i,j+1);
                if(!flag){ //回溯，继续枚举
                    map[i][j]=0;
                    row[i][x]=0;
                    col[j][x]=0;
                    grid[k][x]=0;
                }
                else
                    return 1;
            }
        }
    }
    return 0;
}
```

POJ1190

题目描述（POJ1190）：制作一个体积为 Nπ的 M 层生日蛋糕，每层都是一个圆柱体。设从下往上数第 i（$1 \le i \le M$）层蛋糕是半径为 R_i、高度为 H_i 的圆柱。当 $i<M$ 时，要求 $R_i>R_{i+1}$ 且 $H_i>H_{i+1}$。由于要在蛋糕上抹奶油，所以为了尽可能节约经费，希望蛋糕外表面（底层的下底面除外）的面积 Q 最小。令 Q=Sπ，对给出的 N 和 M，找出蛋糕的制作方案（适当的 R_i 和 H_i 的值），使 S 最小。除 Q 外，以上所有数据皆为正整数。

输入：输入包含两行，第 1 行为 N（N≤10 000），表示制作的蛋糕的体积为 Nπ；第 2 行为 M（M≤20），表示蛋糕的层数。

输出：单行输出一个正整数 S（若无解，则 S=0）。

输入样例	输出样例
100 2	68

提示：圆柱体积 $V=\pi R^2H$，侧面积 $A'=2\pi RH$，底面积 $A=\pi R^2$。

题解：本题为在体积和层数一定的情况下，找到合适的半径和高度，使蛋糕表面积最小。可以采用回溯法搜索求解。

1. 预处理

从顶层向下计算出最小体积和面积的最小可能值。在从顶层（即第 1 层）到第 i 层的最小体积 minv[i]成立时，第 i 层的半径和高度都为 i。此时只计算侧面积，对上表面积只在底层计算一次，底层的底面积即总的上表面积。

```
void init(){
   minv[0]=mins[0]=0;
   for(int i=1;i<22;i++){
      minv[i]=minv[i-1]+i*i*i;
      mins[i]=mins[i-1]+2*i*i;
   }
}
```

2. 算法设计

dep 指当前深度；sumv、sums 分别指当前体积和、面积和；r、h 分别指当前层半径、高度。

（1）从底层 m 层向上搜，当 dep=0 时，搜索完成，更新最小面积。

（2）剪枝技巧：

- 如果当前体积加上剩余上面几层的最小体积大于总体积 n，则退出；
- 如果当前面积加上剩余上面几层的最小面积大于最小面积，则退出；
- 如果当前面积加上剩余面积（剩余体积折算）大于最小面积，则退出。

（3）枚举半径 i，按递减顺序枚举 dep 层蛋糕半径的每一个可能值 i，第 dep 层的半径最小值为 dep。

- 如果 dep=m，sums=i×i，底面积作为外表面积的初始值（总的上表面积，以后只需计算侧面积）。
- 计算最大高度 maxh，即 dep 层蛋糕高度的上限，(n-sumv-minv[dep-1])表示第 dep 层的最大体积。
- 枚举高度 j，按递减顺序枚举 dep 层蛋糕高度的每一个可能值 j，第 dep 层的最小高度值为 dep。
- 递归搜索子状态，层次为 dep-1，体积和为 sumv+i×i×j，面积和为 sums+2×i×j，半径为 i-1，高度为 j-1，即 dfs(dep-1,sumv+i×i×j,sums+2×i×j,i-1,j-1)。

3. 算法实现

```
void dfs(int dep,int sumv,int sums,int r,int h){
    if(!dep){
        if(sumv==n&&sums<best) best=sums;
        return ;
    }
    if(sumv+minv[dep]>n||sums+mins[dep]>best||sums+2*(n-sumv)/r >best) return;
    for(int i=r;i>=dep;i--){
        if(dep==m) sums=i*i;
        int maxh=min((n-sumv-minv[dep-1])/(i*i),h);
        for(int j=maxh;j>=dep;j--)
            dfs(dep-1,sumv+i*i*j,sums+2*i*j,i-1,j-1);
    }
}
```

说明：

（1）**初始参数 r 和 h 均为 n。**因为体积 $V=\pi R^2H$，因此体积为 nπ时，$n=R^2H$，半径和高度均不会超过 n，**半径和高度均大于或等于当前层。**

（2）剩余面积折算。体积 $V=\pi R^2H$，侧面积 $A'=2\pi RH$，$2V/R=A'$，因此将剩余体积折算成剩余侧面积为 2×(n-sumv)/r。

POJ1011

题目描述（POJ1011）：乔治拿来一组等长的木棒，将它们随机砍断，使得每一节木棒的长度都不超过 50 个长度单位。然后他又想把这些木棍恢复到原来的状态，但忘记了初始时有多少木棒及木棒的初始长度。请计算初始时原木棒的最小可能长度。每一节木棒的长度均为大于零的整数。

输入：输入包含多组数据，每组数据都包括两行。第 1 行是一个不超过 64 的整数，表示砍断之后共有多少节木棒。第 2 行是截断以后所得到的各节木棒的长度。在最后一组数据之后是一个 0。

输出：对每组数据，都单行输出原木棒的最小长度。

输入样例	输出样例
9 5 2 1 5 2 1 5 2 1 4 1 2 3 4 0	6 5

题解：

1. 算法设计

本题由切割后的木棒长度推测原木棒的最小长度，可以枚举原木棒的最小长度，使用回溯法搜索及剪枝优化即可解决。可以用拼接的方法反向推测，根据现有木棒拼接成多个等长的原木棒。例如，1 2 3 4，最多可以拼接成两根等长木棒 4+1、3+2，原木棒的最小长度为 5。例如，5 2 1 5 2 1 5 2 1，最多可以拼接成 4 根等长木棒 5+1、5+1、5+1、2+2+2，原木棒的最小长度为 6。

（1）枚举长度。木棒的总长度为 sumlen，最长木棒的长度为 maxlen。因为切割后最长为 maxlen，那么原木棒的长度必然大于或等于 maxlen。如果原木棒只有一根，那么原木棒的长度就是 sumlen。如果原木棒多于一根，那么原木棒的长度一定小于或等于 sumlen/2。从 maxlen 到 sumlen/2，从小到大枚举所有可能的原木棒长度，通过深度优先搜索尝试能否组合成原木棒，如果尝试成功，则当前木棒的长度为原木棒的最小可能长度。

（2）组合顺序。对**木棒长度从大到小排序**，如果**从小到大排序则会超时**。因为小木棒比大木棒灵活性更好，所以先考虑较长的木棒，然后用较短的木棒组合成原棒，更容易成功。好比往箱子装东西，尽量先装大的，然后用小的填补空隙，如果先把小的装进去，大的就可能放不下，或者装不满。用一维数组 used[]标记当前状态下木棒是否已使用组合原棒。

（3）剪枝技巧。

- 剪枝技巧 1：从小到大枚举，第 1 个满足条件的原木棒长度 InitLen 必然是最短的。
- 剪枝技巧 2：原木棒是等长的，因此 sumlen%InitLen=0。
- 剪枝技巧 3：如果当前木棒已使用或者与前一个未使用的木棒长度相等，则无须再搜索。
- 剪枝技巧 4：组合新木棒时，若搜索完所有木棒后都无法组合，则说明该木棒无法在当前组合方式下组合，不用往下搜索，直接返回上一层。

2. 算法实现

```
bool dfs(int len,int index,int num){//当前组合长度、当前搜索起点、已用的木棒数量
    if(num==n)
        return true;
    for(int i=index;i<n;i++){
        if(used[i]||(i&&!used[i-1]&&stick[i]==stick[i-1]))//已使用或与上一个相同
            continue;
        used[i]=true;//标记使用
        if(len+stick[i]<InitLen){//还未组合成功
            if(dfs(len+stick[i],i+1,num+1)) //选中 stick[i]继续组合
                return true;
        }
        else if(len+stick[i]==InitLen){//组合成功一根
            if(dfs(0,0,num+1)) //重新开始组合下一根木棒
                return true;
        }
        used[i]=false;//回溯归位
        if(len==0)//尝试完毕，仍然无法成功
            break;
    }
    return false;
}
```