2006 年北大信息学院《数据结构与算法实习》答案——张铭

一、简答题

1. 数学建模 15 分

(1) （5 分）对于建立的模型一定要做出合理的假设
(2) （5 分）做出假设后要把问题用严格的数学语言表述出来
(3) （5 分）对于自己所作的结论给出严格的数学证明

模型假设：

（1）四条腿一样长，椅脚与地面点接触，四脚连线呈长方形（矩形）
（2）地面高度连续变化，可视为数学上的连续曲面
（3）地面相对平坦，使椅子在任意位置至少三只脚同时着地。

模型建立：

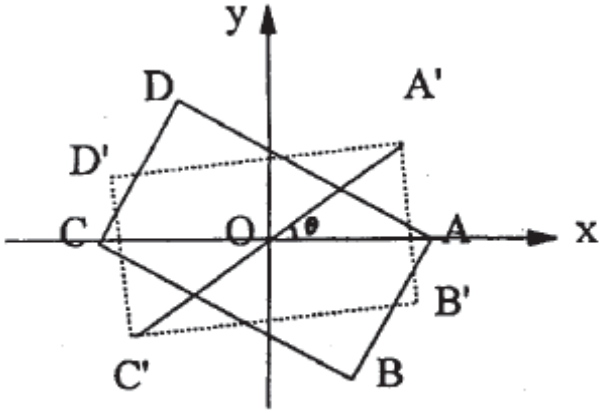


图 1

如图 1， 依次设椅子四脚与地面的接触端点为 A,B,C, D，而矩形 ABCD 的中心 O 为平面直角坐标系的原点，对角线 AC 与 x 轴重合。

设椅子绕中心 O 旋转一个角度θ后，转到 A'B'C'D'的位置。即引人对角线 AC 与 x 轴的夹角θ来表示椅子此时所在的位置。如果用某个变量表示椅子脚与地面的竖直距离，那么当这个距离为零时就是椅子脚着地了。而椅子在不同位置时椅子脚与地面的距离不同，所以这个距离是椅子位置变量θ的函数。设 A, B 两脚端点与地面的距离之和为 f(θ)>= 0.C, D 两 脚端点与地面的距离之和为 g(θ)> =0 。则无论旋转角度θ取何值，由假设(2)知 f(θ)、g(θ)都是连续的。由假设(3)知 f(θ)、g(θ)至少有一个为 0，即 f(θ)g(θ) =0 不失一般性，开始状态时不妨设θ=0，且有 g(θ) =0，f (θ) >0。.因而，改变椅子的位置使四脚同时着地，就归结为如下模型：

已知 f (θ),g (θ)连续，且有 f(θ)g (θ)= 0，g (0) =0, f (0) >0。则存在θ’，使 f (θ’)=0 且 g(θ’)=0

模型求解：

将椅子旋转 π，显然此时矩形两边 AB 与 CD 互换，即有 g(π)>0, f(π)=0 若构造一新函数 h (θ) =f(θ) - g(θ)，考虑区间[0，π]必有 h (θ)连续，且 h(0) = f(0) - g(0) > 0，h(π) = f(π) - g(π) < 0。由连 续 函 数的零点定理知，必存在θ_0 ∈（0，π），使 h(θ_0)=0，即 f(θ_0)=g (θ_0)又总 有 f(θ)g (θ)= 0 成立，于是有 f (θ_0) = 0, g(θ_0)=0，即此时，椅子四脚同时着地。证毕

2. 界面 10 分

列出下表中 3 种界面风格的优缺点以及典型应用场景。

每种各占 3 分，1 分为答题风格分。

交互风格	主要优势	主要不足	典型应用
直接操作	快速直观 易上手	实现难度较大；仅适用于任务或者对象存在可视化的场景	视频游戏 CAD 系统
菜单选择	避免了用户错误操作；较少的击键需求	对于熟练用户来说太慢 如果菜单选项太多可能显得复杂	大多数通用软件系统，如办公软件
表单填充	简单的数据输入；易上手；可核对	占用太多屏幕空间；当用户选项不匹配表单域时会引发问题	求职网申系统，个人贷款处理
命令行语言	强大、灵活	难于学习；贫乏的错误（容错）管理	操作系统，命令与控制系统
自然语言	对普通用户易于理解；容易扩展	需要更多的输入；自然语言理解系统不是那么可靠	信息检索系统，如 Google

3. 等价类划分 10 分
每一个条件 0.5 分

输入条件	有效等价类	无效等价类
是否三角形的三条边	（A>0），（1） （B>0），（2） （C>0），（3） （A+B>C），（4） （B+C>A），（5） （A+C>B），（6）	（A≤0），（7） （B≤0），（8） （C≤0），（9） （A+B≤C），（10） （B+C≤A），（11） （A+C≤B），（12）
是否等腰三角形	（A=B），（13） （B=C），（14） （C=A），（15）	（A≠B）and（B≠C）and（C≠A）（16）
是否等边三角形	（A=B）and（B=C）and（C=A）（17）	（A≠B），（18） （B≠C），（19） （C≠A），（20）

4. 活动选择 10 分

表格占 8 分

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
s_j	4	5	6	7	8	9	10	11	12	13	14

两两相容的最大活动集合占 2 分：{1, 4, 8, 11}。如果顺序有所改变，也判为正确，因为集合不要求顺序。

5. 搜索引擎 10 分

答出下面的三种就可以，每种给 3 分，需要有适当的解释。还有 1 分为创新分。

索引压缩，分块索引，分布式，缓存，增量方式（增量抓取，增量索引）

二、算法填空题

6. 回溯型背包问题填空 8 分，每个空 4 分

//?1　　knap(S-w[n]，n-1)

//?2　　knap(S，n-1)

7．动态规划填空 12 分，每个空 3 分（第 4 空语句虽多，其实类似，每个 1 分）

空段 1：Sim[0][j] = Sim[0][j-1] + Weight[4][GeneB[j]];

空段 2：i++, Cur = !Cur

空段 3：Sim[Cur][0] = Sim[!Cur][0] + Weight[GeneA[i]][4];

空段 4：Sim[Cur][j] = Sim[!Cur][j-1] + Weight[GeneA[i]][GeneB[j]];

Sim[Cur][j] = max(Sim[Cur][j], Sim[Cur][j-1] + Weight[4][GeneB[j]]);

Sim[Cur][j] = max(Sim[Cur][j], Sim[!Cur][j] + Weight[GeneA[i]][4]);

三、 数据结果与算法设计题

8．背包问题动态规划 15 分

(1) 数据结构和算法分析部分占 5 分。如果没有列举出递推方程，最多给 2 分。

(2) 算法部分给 8 分

如果写成递归算法只给 3 分。

(3) 算法分析，占 2 分。

类似整数背包问题。T 相当于背包容量，ti 相当于物体 i 的体积，Si 相当于物体 i 的价值。物体可多选。用动态规划法.

设 F（k，y）表示只允许选择前 k 个题目，耗时不超过 y 时的最大得分。则：

递推方程：

F（k,y）=max{F(k-1,y),F(k,y-tk)+Sk}

F（y）=max{ F(y-tk)+Sk}(1=<k<=n)

边界条件：

F(0,y)=0,

F(k,0)=0,

F(1,y)=S1 y<=t1

0 y>t1

核心代码：

设 F[y]表示当前总时间为 y 时的最大得分，S[k]表示 k 题的得分，T[k]表示 k 题耗时

```
for (k=0;k<n;k++)     {
      for (y=T;y>=T[k];y--)     {
          if (F[y]<F[y-T[k]]+S[k])
              F[y]=F[y-T[k]]+S[k]
    }
}
```

（注：如能举出实例，并列出非递归计算的表格更好）

9. 贪心法 15 分

贪心法算法要点：

1. 对整数区间[left, right]集合按照 right 从小到大排序 （5 分）
2. 对于该有序集合，按序对于每一个区间，分三种情况进行讨论 （6 分）

1）插入 right 和 right-1 到结果集合

2）插入 right 到结果集合

3）不做操作

3. 分析时空复杂度 （4 分）

1）空间复杂度是 O(n)

2）时间复杂度是 O(nlogn)

注：

1. 如果使用冒泡等排序，复杂度为 O(n^2)，适当扣 2 分
2. 如果对区间的 left 排序，则第二步应该考察 left 和 left+1，也算正确，不用扣分
3. 如果采取去除包含小区间的大区间，时间复杂度会达到 O(n^2)，适当扣 2 分

按照区间的右界进行排序（或者先按照右界排序，再按照左界排序）。

“为什么这么排”的要点：取右界和次大的作为插入结果集合的备选元素，查看当前结果集合和当前整数区间的相交情况，分别决定插入二个，一个，还是不查入。

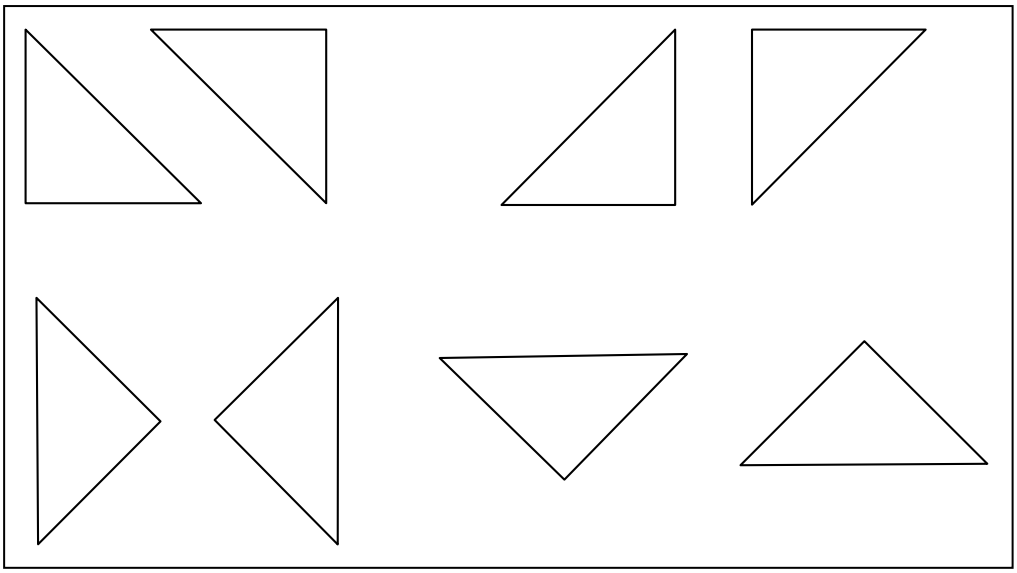
参考核心代码：

```
for(I = 1;I < total;I ++){

    if(right >= interval[i].left){
        if(left < interval[i].left){//正在考察的区间包含上一次找到的一个点
        count ++;//此区间只需要再找一个点
        left = right;//左游标右移
        right = interval[i].right;//右游标移至区间右端点
        }
    }
    else{//区间不包含上一次找到的任何点
        count += 2;//记数加二
        left = interval[i].right-1;//左游标移至区间右端点减一处
        right = interval[i].right;//右游标移至区间右端点
    }
}
```

10. 枚举 15 分

（1）（2 分，每种情况 0.25 分）对每个点进行枚举，只需考虑各点作为等腰直角三角形直角顶点的情形，然后据此搜索各种可能性。由于直角顶点在每个三角形中是唯一的，这样肯定能做到不重复与不遗漏。对于每个顶点，共用 8 种情形：



（2）9 分，下面两个要点各 3 分。

此小题答案不唯一，须包含以下要点 1、记录切割线的情况 2、交点的判断 3、三角形搜索

1、记录切割线的情况

可以使用四个一维数组纪录四种切割线。

```
bool row[51];                    //记录行是否被切
bool line[51];                   //记录列是否被切

//记录左上到右下的连线是否被切，左上到右下连线上的点 x+y 为定值
bool LupToRdown[101];

//记录右上到左下的连线是否被切，左上到右下连线上的点 x-y 为定值
bool RupToLdown[101];
```

均初始化为 false

每输入两个点后，会有如下的处理来修改各数组的值

```
//分别输入两个点的横坐标，纵坐标
cin >> x1 >> y1 >> x2 >> y2;

//分情况修改 bool 数组
//1、行被切
if (y1==y2)
{
    row[y1]=true;
}

//2、列被切
else if (x1==x2)
{
    line[x1]=true;
}
```

```
//3、左上到右下线被切
else if (x1+y1==x2+y2)
{
    LupToRdown[x1+y1]=true;
}
//4、右上到左下线被切
else if (x1-y1==x2-y2)
{
    RupToLdown[x1-y1+N]=true;   //将数组的下标+N 使其不会为负
}
```

2、交点的判断

引入二维数组

```
    bool isCrossPoint[101][101];    //记录点是否是交点
```

将数组大小定为 101*101 是为了使所有交点坐标都为整数。交点的判断由下面的函数实现：

```
void JudgeCrossPoints(bool isCrossPoint[][101], bool row[], bool line[], bool
LupToRdown[], bool RupToLdown[], int M, int N)
{
    int x;
    int y;

    //枚举矩形内小正方形的顶点(只要过某一点有任意的两条线的组合,这一点就是交点)
    for (x=0; x<=M; x++)
    {
        for (y=0; y<=N; y++)
        {
            if (row[y] && line[x]
                || row[y] && LupToRdown[x+y]
                || row[y] && RupToLdown[x-y+N]
                || line[x] && LupToRdown[x+y]
                || line[x] && RupToLdown[x-y+N]
                || LupToRdown[x+y] && RupToLdown[x-y+N])
            {
                isCrossPoint[2*x][2*y]=true;//注意有两倍关系
            }
            else
            {
                isCrossPoint[2*x][2*y]=false;
            }
        }
    }

    //枚举矩形内小正方形的中心（这些点只有可能是斜线交出，可简化判断）
    for (x=0; x<M; x++)
```

```
    {
        for (y=0; y<N; y++)
        {
            if (LupToRdown[x+y+1] && RupToLdown[x-y+N])
            //x+y+1=(2*x+1+2*y+1)/2
            //x-y+N=(2*x+1-2*y-1)/2+N
            {
                isCrossPoint[2*x+1][2*y+1]=true;
            }
            else
            {
                isCrossPoint[2*x+1][2*y+1]=false;
            }
        }
    }
}
```

3、三角形搜索

我们以上图中第一个三角形为剖析对象，根据上面的数据结构和赋值，进行判断：

函数原形均类似于

```
int SearchTriangle1(bool isCrossPoint[][101], bool LupToRdown[], int startx, int starty, int M, int N)
```

若能找到则返回 1，否则返回 0。

要注意的是，并不是所有的搜索函数中 M 和 N 都用到了，之所以都这样定义是为了统一性。

各函数的实现就是沿着边寻找到两个最近的点，然后再看这三点是否会形成一个三角形。

搜索 Triangle1 的实现如下：

```
int SearchTriangle1(bool isCrossPoint[][101], bool LupToRdown[], int startx, int starty, int M, int N)
{
    if (startx==2*M || starty==2*N) //判断是否是不需要搜索的边界
        return 0;

    int searchx=startx;
    int searchy=starty;
    //以下两段 while 是沿着所搜寻三角形的直角边寻找离函数传入的点最近的两个交点
    while (1)
    {
        searchx += 2;
        if (isCrossPoint[searchx][starty])
            break;
    }
    while (1)
```

```
    {
        searchy += 2;
        if (isCrossPoint[startx][searchy])
            break;
    }
    //判断找到的两个点是否能形成 triangle1，即他们之间是否有斜 45 度的连线
    if (searchx+starty != startx+searchy || !LupToRdown[(searchx+starty)/2])
        return 0;
    return 1;
}
```

其他所有搜索函数均相当类似，只要根据每个三角形的具体形状再分别修改一些寻找以及判断的语句就可以了。

答案须包含以上三个要素；代码简洁，注释到位，思路清晰为评分要点

（3）代价分析 4 分

时间复杂度分析：首先，对所有点进行搜索就有了一个 M*N 的复杂度；其次，在每一个点处的函数调用，又需要一个搜索的过程，不过在一般的情况下，这个搜索过程会小一些。最不幸的情况是搜索 max(m,n)次左右。而 M 和 N 是一个数量级，因此总的时间复杂度为 O（n^3）

空间复杂度分析：主要用了四个数组，而且根据题目的最大规模，该题是静态分配了四个数组，在问题范围内，空间开销不会发生变化。空间复杂度为 O(max(m,n))