程序设计方法与艺术

解题报告

班　　级：2020 级计科 3 班
指导老师：徐本柱
组　　长：2020217920 陈良宇
组　　员：2020217925 段萌
2020217927 卢中源
2020217891 王震

Contents

1 A

1.1 题目描述

给定 N 个正整数，每个数大小不超过 3，那么最多能将这 N 个数分为多少个组，使得每一组的和都是 7 的倍数。

1.2 思路分析

本题要求对给定数量的 1、2、3 进行分组，并求出满足和为 7 的整数倍的最大组数。我们发现，要想分出最大的组数，就要使每组的和尽可能小。通过分析，每组的和只可能是 7、14 或 21，更大的数如 28，一定能够被分解为满足条件的两组或更多组。

1.3 解决方案

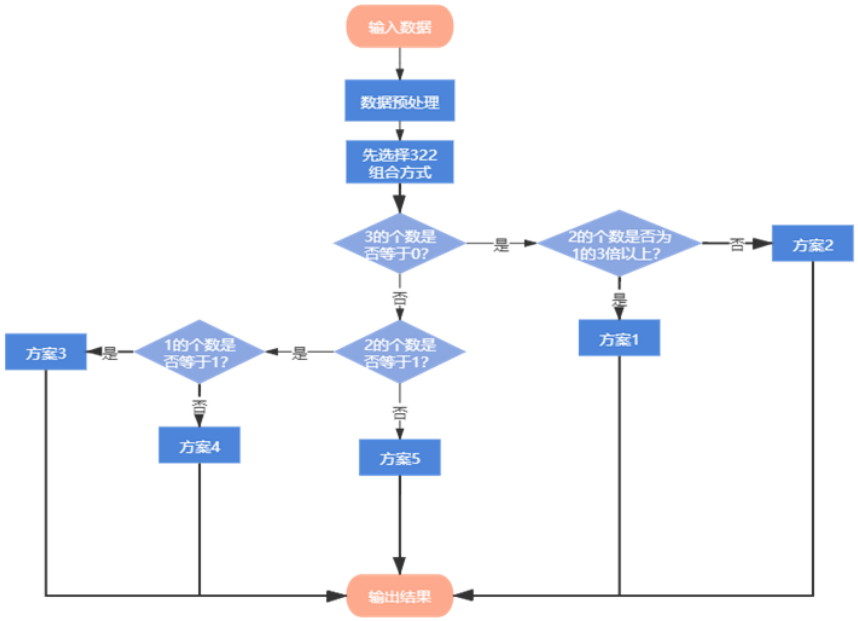

Figure 1: A 题流程图

1.4 程序运行结果

对给定的测试样例，我们得到了正确结果。

Input:	Output:
3	1
4	2
2 1 1	22
6	
0 4 2	
100	
61 24 15	

Figure 2: 测试样例

我们采用在安徽科技大学的公开 oj 上测试, 通过测试并拿到了 AC。测试截图如下:

Figure 3: A 题通过截图

2 B

2.1 题目描述

有一个 1 到 N 的排列 P，给定 P 中任意两数的大小关系，求恢复这个排列。

2.2 思路分析

通过观察题目，需要对所输入的 1，0，-1 进行处理，通过题目所给的关系来进行分析来找到每行数据所代表的值。要通过对应关系来找到解并不容易，但通过观察题目可以发现，每次出现 1 个 1 代表大于一个值，而数据是从 1 至 n，这样问题就很简单了，通过计数 1 来找它的值。

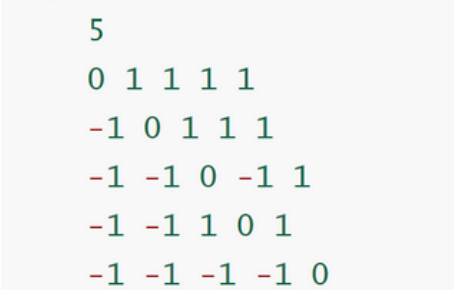

Figure 4: 图例

例如第一行代码有 4 个 1 代表大于 4 行数字，所以该数值为 5

2.3 解决方案

将所有的数据输入，存放于数组之中，将每行的数组遍历，设置计数变量 n=1，当遇到 1 的时候使 n+1。遍历之后 n 存放数组中再次将 n 设置为 1。将每行数据循环结束之后就可以将答案输出得到最终解。

具体流程图如下：

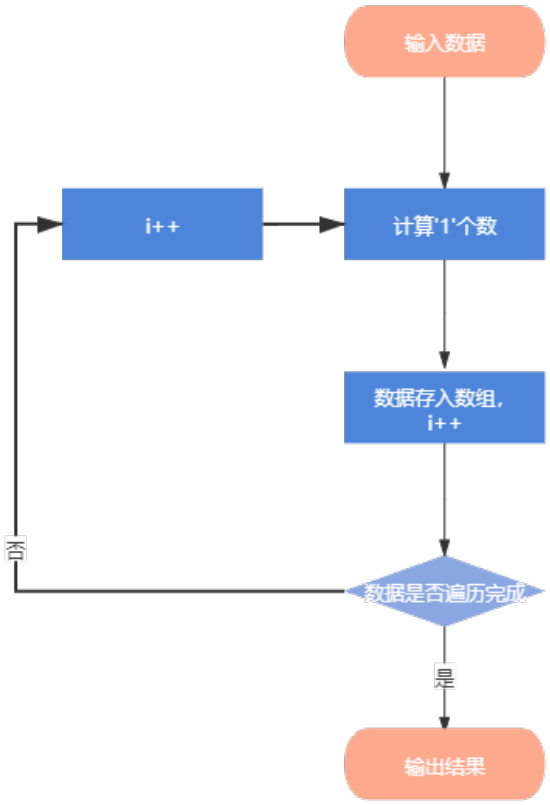

Figure 5: 解决方案流程图

2.4 程序运行结果

对于测试样例, 很轻松就能得到答案:

```
Input:
    5
    0 1 1 1 1
    -1 0 1 1 1
    -1 -1 0 -1 1
    -1 -1 1 0 1
    -1 -1 -1 -1 0
Output:
    5 4 2 3 1
```

Figure 6: B 题测试样例通过截图

我们采用在安徽科技大学的公开 oj 上测试, 通过测试并拿到了 AC。测试截图如下:

Figure 7: B 题通过截图

3 C

3.1 题目描述

给定一个序列，对于序列中的两个数，若标号小的数比标号大的数大，则在两个数之间连一条边，最终得到一个无向图，求最大连通块中的元素个数。

3.2 思路分析

该题目是一道非常经典的利用并查集求解的问题：将每一个数看作顶点，之后对满足条件的点进行连边操作。而如何表示这些顶点之间的关系，是解决本题的关键。通常，我们以数组下标表示顶点的值，但本题中数的大小范围为 10^9，如果开一个 10^9 的数组，内存开销过于巨大，这样显然是不行的。所以我们可以换一个角度考虑：只记录对应数值的标号，因为其唯一确定，所以可以用标号之间的关系构建并查集。这样，数值就只是判断是否建立边的条件，大幅减少了内存的开销。

接下来就是具体的构建过程。以数值的序号为顶点，查找时采用路径压缩的方法，通过将加入的结点的前驱结点即时更新为根结点，可以减少查找时所需要的次数。此外，由于本题需要求出连通块中的元素个数，所以还需要建立一个同样大小的数组来存储每个结点的该项信息，并在每次合并时通过比较更新记录的最大值。

3.3 解决方案

首先建立构建并查集所必需的合并和查找函数，然后依次读取数值，将读取过的数值放在一个临时数组中，将当前被读取的元素与之前的数值相比较，符合题意则进行结点的合并操作。合并之后更新根结点记录的元素个数以及记录的最大值。遍历序列则可得到最终结果。

具体过程的流程图如下：

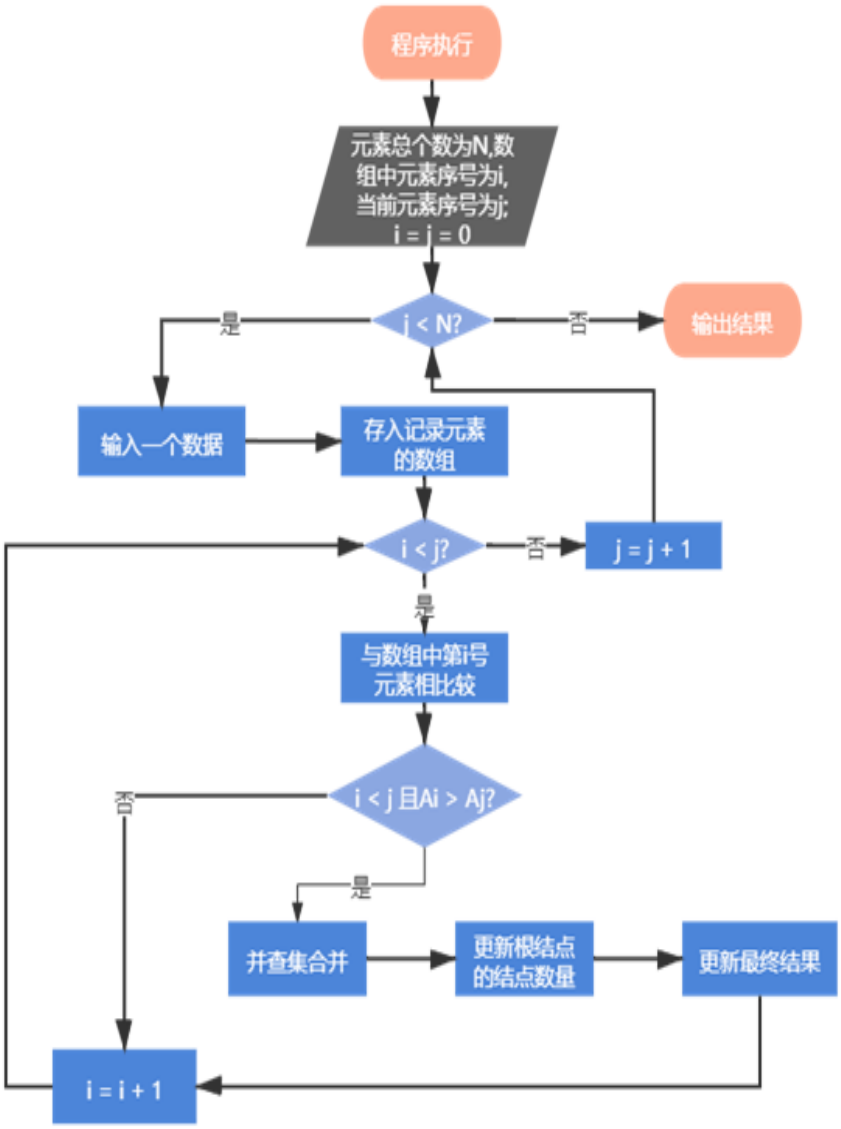

Figure 8: C 题流程图

3.4 程序运行结果

对给定的测试样例，我们得到了如下结果：

```
Input:                    Output:
    5                         5
    4 4 3 2 3

    2                         1
    3 3

    7                         5
    3 4 2 4 2 8 6
```

Figure 9: 测试样例

此外，我们采用在安徽科技大学的公开 oj 上测试，通过测试并拿到了 AC。测试截图如下:

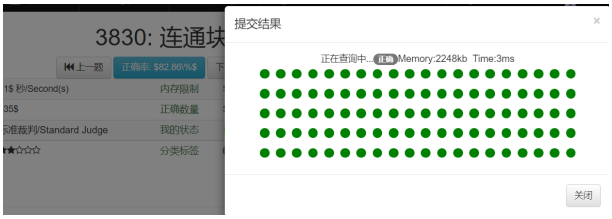

Figure 10: C 题通过截图

4 D

4.1 题目描述

血魔遇到了 N 个敌人。初始血魔生命值为 M，每秒钟血魔的生命值会减少 1。如果血魔的生命值变为 0，血魔会立刻消失，但在此之前他可以击杀敌人来恢复生命值。对于第 i 个敌人需要攻击 Ti 次才能杀死，血魔每秒钟可以攻击一次，每秒钟血魔可以攻击任意一个敌人，并且每次攻击可以同时击中任意多个敌人。敌人一共有 K 种类型，对于每种类型的敌人，血魔最多只能杀死一个。血魔在进行过所有攻击后，他可以立刻恢复他所杀死敌人的最小生命值，在恢复过后不能再继续进行攻击（如果血魔生命值变为 0 的瞬间恢复生命值，血魔不会死亡）。现在血魔想知道，他是否能杀死 K 个敌人，如果能的话，那么杀死 K 个敌人后，生命值最高能为多少?

4.2 思路分析

首先分析题目，题目要求我们判断血魔是否能存活并找出血魔能存活情况下的最大生命值。首先我们来判断血魔是否能存活，存活的前提条件是血魔在生命值丧失之前杀死每种类型的敌人各一个，那么我们将每种类型敌人所需要攻击的次数最小的那个给取出来，依次跟血魔的血量排个序，如果有一个所需攻击次数比血魔血量大的话，那就说明血魔有一种类型的敌人杀不了了，就输出-1，退出程序。在经过上一步的判断之后，就知道血魔已经可以活下来了，接下来就是找出血魔能获得的最大生命值，将每种种类的敌人按照血量从大到小排序，然后在每种种类的敌人中找到血量最高同时能被血魔杀死的敌人，然后从这些敌人中找出血量最高的那个敌人，然后再从这些敌人中找出需要攻击次数最大的那个敌人，然后用血魔的血量减去最大攻击次数加上杀死敌人最高血量就是血魔能获得的最大血量。

4.3 解决方案

先输入数据，将敌人按照类型分配到不同的 vector 中，然后将每种类型的敌人按照需要攻击的次数从小到大排序，之后将每种类型的需要攻击次数 attackTimes 的最小值与血魔的最小血量 hp 进行比较，如果存在有一种类型敌人的最小攻击次数大于血魔的 hp, 那么血魔就死亡, 否则到下一步。将每种类型的敌人按照血量从小到大进行排序, 定义一个下标数组, 用于存放一个种类敌人的下标, 将每种类型的敌人的所需攻击次数与血魔的血量相比, 直到血魔的血量大于等于这个敌人的所需攻击次数, 记录当前敌人下标放入 $index$[x] 中, 找出每种符合类型的敌人的最小血量 $hpMin$ 与最大所需攻击次数 $attackTimesMax$, 那么血魔最终剩余的血量就为:

$$hpEnd = hp + hpMin + attackTimesMax \tag{1}$$

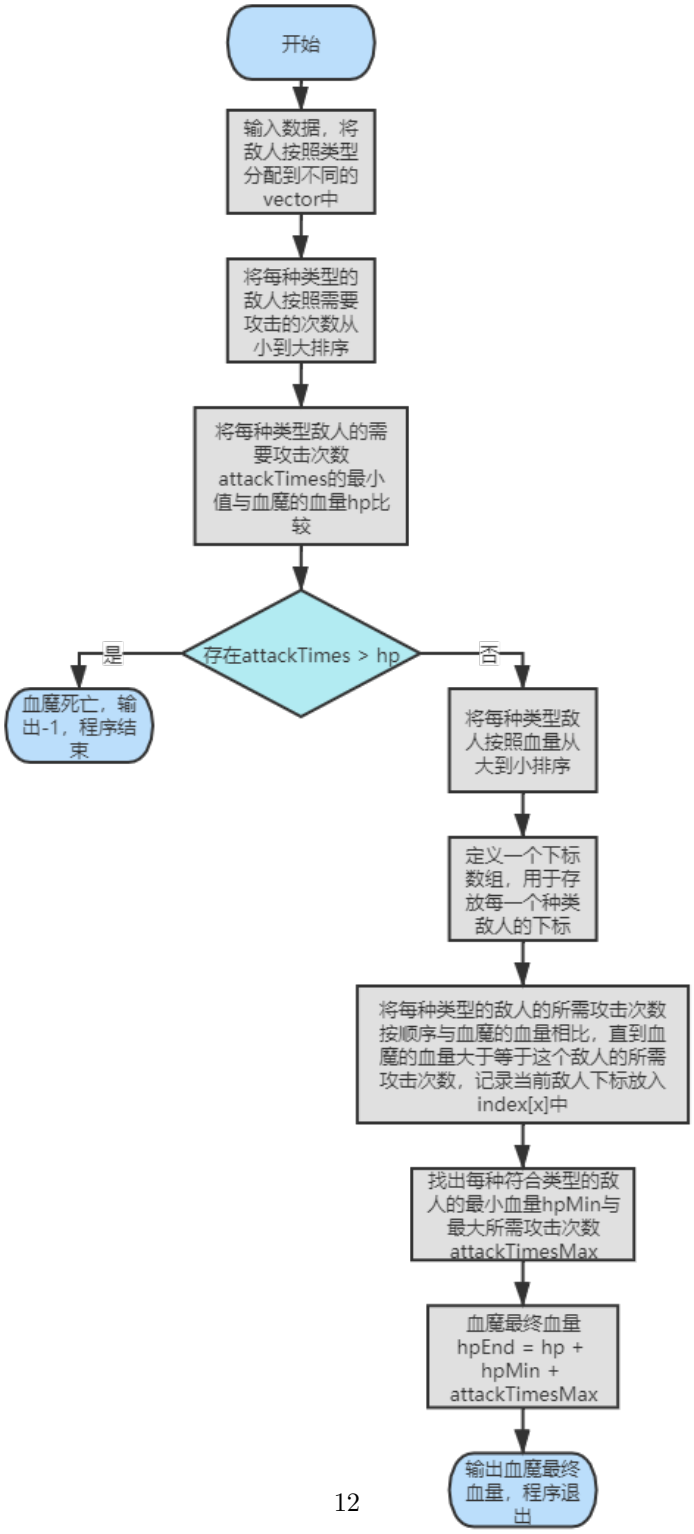

Figure 11: D 题流程图

4.4 程序运行结果

对于测试样例数据, 我们很轻松就能得到结果

```
INPUT:                  OUTPUT:
    2 5 2                   1
    1 2 2
    4 5 1

INPUT:                  OUTPUT:
    5 4 2                   5
    4 2 1
    1 1 1
    4 1 1
    3 2 2
    1 1 2
```

Figure 12: D 题测试样例通过截图

与此同时, 这道题目我们在安徽科技大学的 Oj 上面得到了第一 AC。

Figure 13: D 题通过截图

5 E

5.1 题目描述

给定一个长 n 的字符串 a，然后进行 k 次操作，操作一共有两种。(1,t,x,y) 表示将字符串 a 重置为第 t 次操作后的状态，然后将 a 中第 x 个字符修改为 y，(2,t,x) 表示将字符串 a 重置为第 t 次操作后的状态，然后删除第 x 个字符，x 之后的字符依次向前补齐。假设 s1,s2,...,sk 为每次操作后得到的字符串，现在要求将这 k 个字符串进行排序，输出排序结果。

5.2 思路分析

题目要求对字符串进行处理, 由于需要对字符串进行回溯, 因此程序需要建立相应的存储机制来支持回溯操作, 除此之外, 操作类型共有两种, 并且两种操作之间参数数目也有所不同, 这也对操作的存储以及应用造成了一定麻烦。

最先想到的是暴力求解 (此方法未尝试), 但是用暴力求解对资源浪费是非常严重的, 问题的本质还是需要进行求解。于是就有了以图的大致思路:

(1) 设计合适的数据结构, 来对变换后的数据和相对应的操作进行存储。

(2) 设计合适的方法, 对字符串的操作过程以及判断过程进行简化, 以此降低时间复杂度。

(3) 实现算法, 进行求解。

5.3 解决方案

具体的解决流程如下:

Figure 14: E 题解决流程图

首先根据题目要求设计合理的数据结构如下:

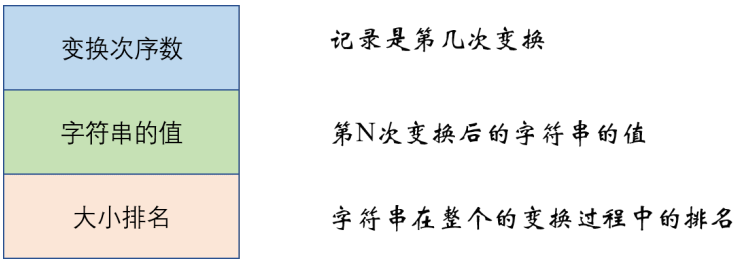

Figure 15: 自主设计的数据结构

之后通过这种数据结构进行存储数据。

对于排序, 有一种比较合适的思想, 就是自主设计一个比较函数去进行比较, 通过 sort 函数可以快速排序得到结果。

之后我们就可以直接输出结果。

5.4 程序运行结果

对于测试样例数据, 我们很轻松就能得到结果:

```
Input:
    3
    aca
    5
    1 0 2 e
    1 1 2 c
    2 2 1
    2 3 1
    1 0 2 c
Output:
    4 2 5 1 3
```

Figure 16: 测试样例结果

与此同时, 这道题目我们在安徽科技大学的 Oj 上面得到了第一 AC。

6 F

6.1 题目描述

有一条无限长的道路，道路可以近似看做一条直线。道路上有 N 个部分需要维修，第 i 个部分的维修的位置是 Vi，Xi 时间开始维修，直到 Yi 时间维修结束。有 Q 个维修工人，第 i 个工人的上班的时间是 Ai，下班时间是 Bi，从 Ai 时间开始，他负责所有维修位置在 [Li,Ri] 范围的道路，直到 Bi 时间结束。他想要知道他的工人上班的每一分钟是否都在进行维修工作，你能编写一个程序回答他吗?

6.2 思路分析

这道题就是要知道每个工人是不是每个时候都有活干，就要先知道每个工人能干活的维修点在哪块，然后在找出每个工人的维修时间是否包括在这些总维修时间中，如果被包括了，那么就说明工人能一直进行维修工作，否则就是不能。

6.3 解决方案

这道题就是先建立数据集，然后在每个工人的工作时间的每个时间点遍历该工人所能够工作的地点，如果工人的每个工作时间点都有对应的工作那么就输出 Yes，如果有某一个时间点没有对应工作就说明该工人不能每一分钟都进行维修工作，输出 No。

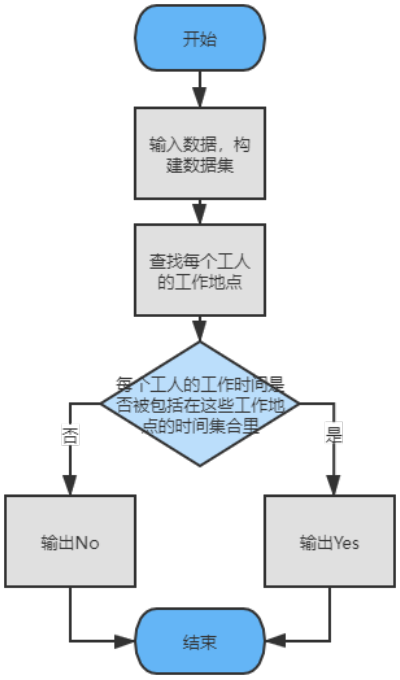

Figure 17: F 题流程图

6.4 程序运行结果

对于测试样例, 很轻松就能得到答案

```
INPUT:                          OUTPUT
    4                               Yes
    5 3 3                           No
    2 4 5                           Yes
    3 1 2                           No
    5 2 3
    4
    2 3 2 2
    1 4 2 3
    2 5 2 4
    3 5 1 5
```

Figure 18: F 题测试样例通过截图

我们采用在安徽科技大学的公开 oj 上测试, 通过测试并拿到了第一 AC。测试截图如下

Figure 19: F 题通过截图

7 G

7.1 题目描述

对于一棵给定的树，如果有边 (a,b)(b,c)(c,d)，但 a、d 之间没有边，则将 ad 连接形成一条边，求最多能连多少边?

7.2 思路分析

此题表面上看起来是一道关于树的问题，但增加的边势必会与之前的边形成一个环，所以本质上这是一道关于图的问题。根据题意，我们要找到与当前顶点相隔两个顶点的顶点，并对其进行是否有边的判断。由于其“分层遍历”的性质，我们很自然地联想到了广度优先搜索。

7.3 解决方案

首先建立一个以邻接链表表示的图，之后根据题意读取并存储顶点和边信息。对于每个顶点，我们在结构体中额外记录了该顶点被访问时的“层数”，和一个“以特定顶点起始的不可访问”的整数。当进行 BFS 时，首先要根据起始顶点的下标更新“是否可以访问”的情况，再更新遍历的层数，最后再将其入列。将遍历到第三层的顶点与起始顶点进行连边判断，并更新连边的个数。

此外，不同于普通搜索的是，新连接的边可能会对接下来的遍历产生影响。比如下图的情况：

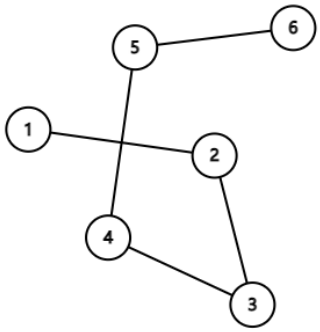

Figure 20: 给定的树

当我们连接了 1、4 之后，会发现原本无法连接的 1、6 也成为了可能连接的边。

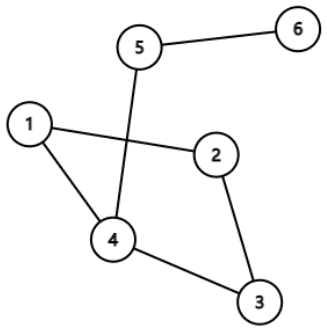

Figure 21: 连接之后的图结构

经过分析，我们发现要将新连接的顶点作为层数为 1 的顶点重新放入队列进行搜索。至此，改造的 BFS 算法已经完成。以下是流程图：

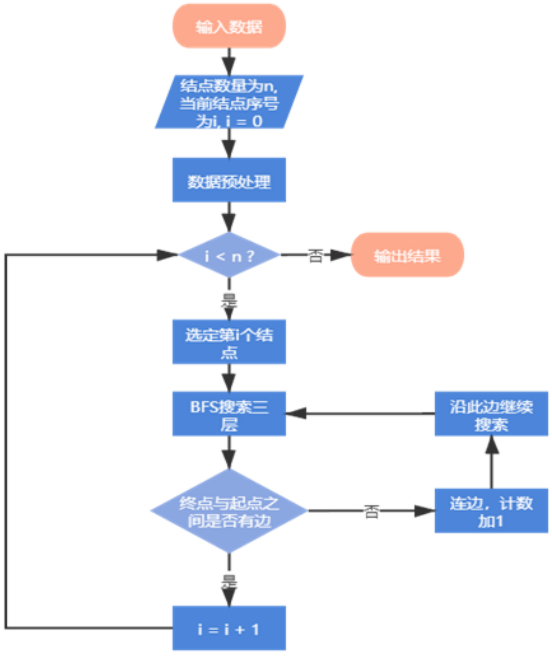

Figure 22: G 题流程图

7.4 程序运行结果

对给定的测试样例，我们得到了正确结果：

```
Input:                Output:
    8                     9
    1 4
    2 4
    3 4
    4 5
    5 6
    5 7
    5 8
```

Figure 23: 测试样例

此外，我们采用在安徽科技大学的公开 oj 上测试，通过测试并拿到了 AC。

测试截图如下:

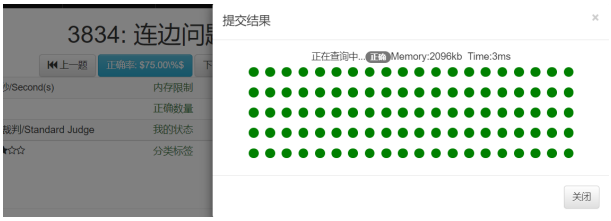

Figure 24: G 题通过截图

8 H

8.1 题目描述

有一个方盘，被分割成了 4 行 4 列共 16 个方格。方盘的主人 Jasonare 最近突然萌生了一个问题，但是任他怎么冥思苦想都得不到答案，他非常沮丧，以至于最近这段时间茶饭不思，他的好朋友 May 看到他这样非常担心，打算把这个问题晒出来，并承诺若有人能将此问题解决，便可以将他珍藏多年的钢铁侠手办限量版赠与对方。

可以看到问题是这样的——你需要在方盘中 4*4 的方格中填数，且填入的数字必须为正整数，Jasonare 事先规定了每一行数字的总和、每一列数字的总和以及两条对角线数字的总和。另外在你开始填数前，他在方盘的任意四个格子事先填好了数字。请问你能否将这个方盘中的方格填满数字，并且满足每一行数字的总和、每一列数字的总和以及两条对角线的总和都与 Jasonare 规定的相同。

8.2 思路分析

首先对题目进行定性, 数字谜盘问题由来已久, 之前就有幻方此类的问题 (下文称此问题为幻方), 对于这道题目刚开始的想法是使用线性代数的知识去寻找是否有简便求解方法, 但是在对幻方进行不同的各种变换之后我们发现不管是经过怎么样的变换, 其对应的行列式的值以及对应矩阵的秩始终不满足求解需求, 也就是说, 我们始终在相同的条件中兜圈子, 于是自己尝试对求解这个过程进行分析。

对本问题分析, 我们试着对其进行求解, 我遵循着以下的求解法则:

(1) 每一个位置的数字都对应着他们的重要程度, 这个重要程度根据其所在行列 (如果处于对角线上或者反对角线上就还有这二者) 的和有关, 越大的和也就意味着这个位置越重要。

(2) 我们每一次选择都是贪心的, 也就是说, 我们始终是要先填满越重要的位置

(3) 在填充失败的时候, 我们考虑使用回溯, 将其进行回溯, 具体体现为在重要程度大的地方先填充次重要的, 再将当前位置的数值进行回溯。

8.3 解决方案

我们首先对幻方进行初始化, 在这个过程之中, 将整个幻方的数值都初始化为 1, 由于对幻方的数值都初始化为 1 以及我们有给定的四个点的数据, 于是我们需要对幻方的相应的行列以及对角线的和减去相对应的数值大小, 预处理到此完毕。

之后我们以列和为基准, 寻找最大的列和, 之后寻找最大的行和, 由于第一列和第四列和第一行的交点位置和对角线也有关系, 同理第二列和第三列与第二行和第三行与对角线也有关系, 我们首先考虑这些位置。

寻找到最优的位置之后, 我们在这个位置增加 1, 那么在之后, 我们根据此位置来更改行和, 列和, 对角线和, 之后重复此操作。

在发现数据无解的时候, 我们需要对数据进行回溯运算, 也就是将上一次的不同位置的地方减去数值 1, 这也就意味着我们需要设立变量来对上一个不同位

置的坐标进行记录。

具体的流程图如下:

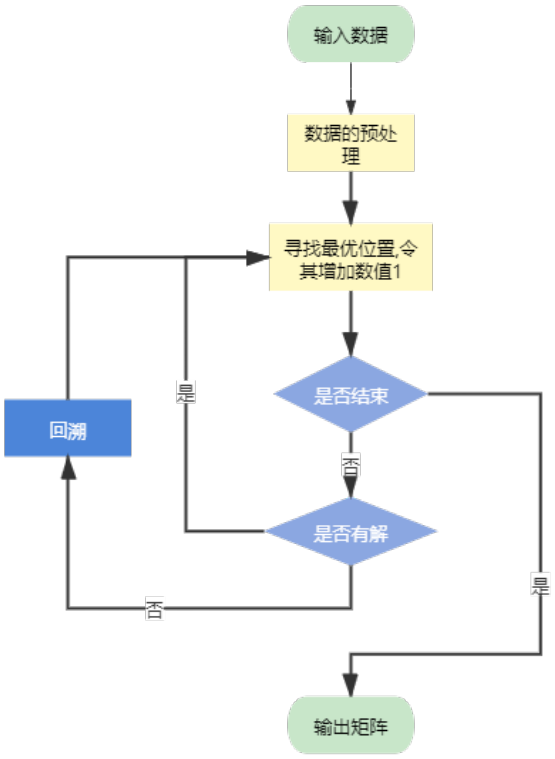

Figure 25: H 题流程图

8.4 程序运行结果

对于测试样例, 很轻松就能得到答案:

```
Input:
    4 4 4 4 4 4 4 4 4 4
    0 0 1
    1 1 1
    2 2 1
    3 3 1
Output:
    1 1 1 1
    1 1 1 1
    1 1 1 1
    1 1 1 1
```

Figure 26: H 题测试样例通过截图

我们采用在安徽科技大学的公开 oj 上测试, 通过测试并拿到了第一 AC。测试截图如下:

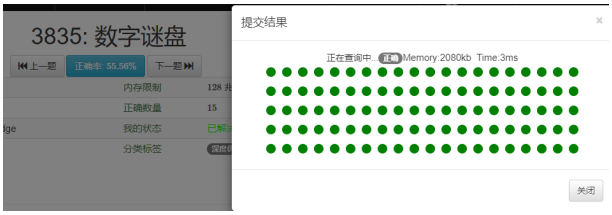

Figure 27: H 题通过截图

9 I

9.1 题目描述

地球即将毁灭，科学家们派遣了一支队伍前往银河系外进行调查，寻找未来人类能居住的行星。幸运的是，在一个星球上，科学家们搜集到了许多 RNA 片段，科学家们把这些 RNA 片段运送往地球的实验基地中进行研究。经过几周的研究，他们发现这些 RNA 片段中存在着许多未知的病毒！

每个 RNA 片段都由 ACT 三种碱基组成，经过研究发现，如果一个 RNA 能够翻译出一个 AACCTT 无限循环的蛋白质序列，那么这个 RNA 就存在病毒！RNA 翻译蛋白质时，首先会随机选择一个位置的碱基，将该碱基翻译为蛋白质，然后继续随机选择相邻位置的碱基进行翻译，直到结束。现在给出一个 RNA，你需要判断这个 RNA 是否存在病毒。

9.2 思路分析

这道题是要找出是否会出现循环的 AACCTT, 就从所有碱基序列中找出一个碱基 A 开始递归，从相邻碱基找出下一个所需要的碱基，如果从这个 A 碱基开始能够构成 AACCTT 序列就说明这个 A 碱基能构成 AACCTT 序列，如果在之后每次新序列 AACCTT 建立完毕，如果这个末尾 T 的相邻碱基中有之前生成过完整序列的 A，就说明这个序列能够生成循环的 AACCTT 序列，即存在病毒。

9.3 解决方案

1。定义一个碱基数组，用于存放整个 RNA 片段与其中碱基对相邻的信息。

2。从所有碱基序列中找到一个 A 碱基, 从这个 A 序列开始递归创建 AACCTT 序列。

3。每次递归到一个碱基，就观察到有没有序列所需要的下一个碱基，如果有，就递归到下一个碱基，如果没有，就回溯到上一个碱基。

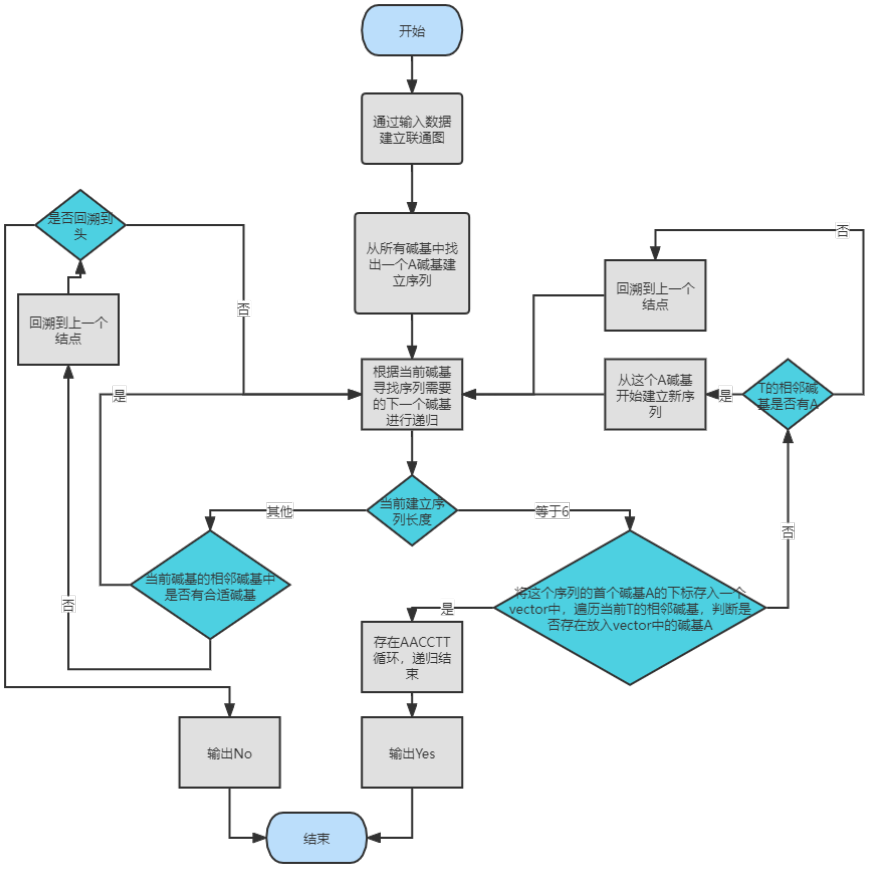

Figure 28: I 题解决流程图

9.4 程序运行结果

```
INPUT:
    6 7
    AACCTT
    1 2
    1 6
    2 3
    3 4
    4 5
    1 2
    5 6
OUT:
    YES
```

Figure 29: I 题测试样例

我们采用在安徽科技大学的公开 oj 上测试, 通过测试并拿到了第一 AC。测试截图如下

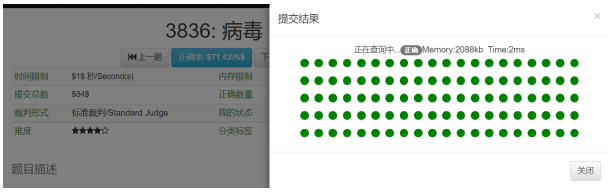

Figure 30: I 题通过截图

10 J

10.1 题目描述

Alice 和 Bob 在玩游戏，两个人分别有一个计分板，记录各自的得分。得分 X 的字典序严格小于得分 Y，那么就认为得分 X 高于得分 Y。Bob 想要自己的分数高于 Alice，他选择了自己计分板中一些相邻位置交换。为了不让 Alice 发现，Bob 必须交换尽量少的次数，请写一个程序帮助 Bob。

10.2 思路分析

最初看到这题时候感觉就像找到最少次数使用几个数字来表示另一个较大数字，办法就是让大的数字先多次使用，具体想法大致类似，如果只用将 a 移动到前边就能完成这次排列最好不过了，可还需要分辨先将别的字母移动的方法，如果只移动 b 或别的就可以完成，所以设置多次循环，来解决不同优先级的移动问题。

10.3 解决方案

题目要求输入两行字母，进行比较两行字母的字典序，如果 Bob 的字符串字典序小于 alice 的需要用最少的步数来进行调整，使用最少步骤需要进行运行次数的比较，多种方式找使最少次数完成目标的方式。应该优先将 a 移到前边，依次将字典序高的优先移动，进行多次循环，每次改变优先移动字符，找到最优解。将字母放入数组 a[]。进行循环如果字符串 a[]>b[] 退出循环，否则继续下一步将 a[i] 中字符与字符串中字符进行对比如果搜索到那么将 flag 置为 1，如果 flag 为 1 那么就进行字母的移动，否则找下一个字母进行移动。如果 a[]>b[] 那么循环就结束，将 flag 再次置为 0。进行下一个优先级字母的移动过程，最终将移动次数存入数组。所有遍历结束之后，在进行数组中次数的排序找出最少次数，如果不能成立，那么 flag1 将会为 0 会输出-1，表示这次不能完成。

具体流程图如下：

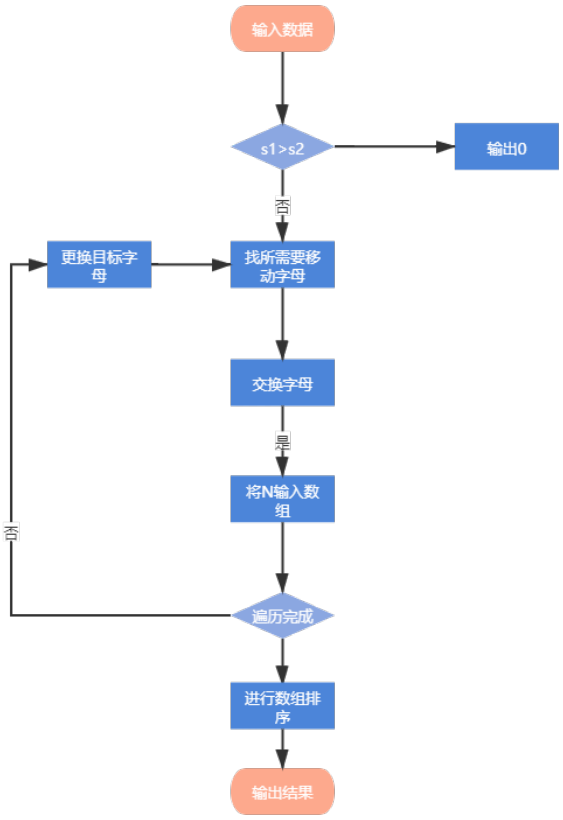

Figure 31: 解决方案流程图

10.4 程序运行结果

对于测试样例, 很轻松就能得到答案:

```
Input:
    hgfedcba
    abcdefgz
Output:
    28
```

Figure 32: J 题测试样例通过截图

我们采用在安徽科技大学的公开 oj 上测试, 通过测试并拿到了 AC。测试截图如下:

Figure 33: J 题通过截图

11 K

11.1 题目描述

在斗罗大陆，有两个强大的帝国，分别是天斗帝国和星罗帝国。由于武魂殿从中挑拨离间，两大帝国之间的关系日益紧张。在天斗帝国太子雪清河（由千仞雪伪装）的蛊惑下，天斗帝国的雪夜大帝决定向星罗帝国宣战。

然而星罗帝国的实力也不容小觑，双方经过长达数月的对峙，仍然没有一方占据优势。现在天斗帝国这边决定派出在用毒方面颇负盛名的毒斗罗，而星罗帝国这边则派出了 n 个魂斗罗应战，他们打算群起而攻之。战局对毒斗罗相当不利，因此毒斗罗决定使出范围攻击，其攻击范围可以看作一个矩形区域，若是接触到这种攻击，哪怕是魂斗罗也会在顷刻间死亡。但是由于之前魂力被消耗大半，毒斗罗想尽可能节省魂力，已知攻击的范围越大，魂力消耗就越大 (即矩形区域的面积越大，魂力消耗越大)。若这次攻击能够完全覆盖到 n 个敌人，那么这次攻击就是成功的，否则这次攻击就是失败的。现在已知星罗帝国 n 个魂斗罗的位置，请问毒斗罗该如何进攻才能使得他的魂力消耗最少，又能确保这次攻击是成功的。请求出能让他攻击成功的最小矩形区域面积。

11.2 思路分析

首先对题目进行定性, 通过初步阅读题目我们可以知道这是一个关于使用矩形将所有坐标点都包含进去的问题, 在这个问题中, 很容易有一个误区就是矩形是正着的, 也就是说, 其长和宽和 x 轴,y 轴平行, 当有新的点进来的时候, 我们就延长矩形的长和宽, 让矩形将点包含进去, 这个思路是显而易见的, 但却是一个陷阱。因为当矩形倾斜的时候有可能更小面积的矩形就能满足题目需求, 如下图:

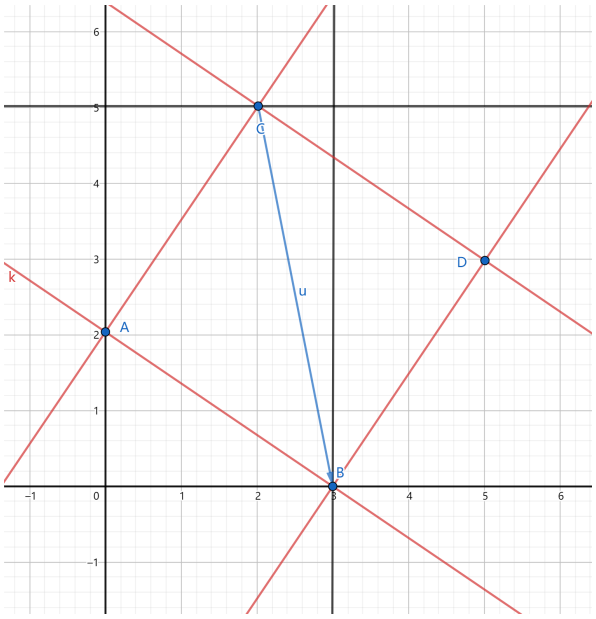

Figure 34: 倾斜的矩形对面积需求可能更小

在这里可以清晰地看出, 对给定的三个点 A,B,C, 三角形 ABC 是倾斜的矩形 ABCD 面积的一半, 却达不到不倾斜的矩形的一半面积 (在 A 处做一个平行于 X 轴的直线即可易证得出), 那么我们需要考虑的就是覆盖给定面积的最小倾斜矩形, 这个给定的区域如何去求解, 这就变成了我们需要面对的主要问题。

对于所求的区域, 由于在这个区域内部的点我们并不需要考虑, 所以我们将其看成一个多边形即可, 这个问题类似于如何围篱笆才能最节省的问题相似, 我们假想有一条无限长的绳子, 绳子绕着某一个点伸展后逆时针旋转, 直到绳子旋转的部分与已经被点阻挡住的部分相遇才停止, 绳子在旋转过程中碰到的那些点就是我们所要求解的边界点, 我们将最外围的绳子看成是这个多边形的边, 那么就可以得到这个最小区域数学化的多边形。它实际上是一个求解凸包的过程 (下文称这个区域为凸包)。

得到多边形之后, 我们需要对其进行一个操作, 我们可以很直观地看出, 最小矩形的某一条边一定与凸包的一条边重合, 那么我们可以假设有一条边从凸包的第一条边为基准, 来进行不断旋转和其他边重合, 同时求解在和各个边重合的情况下能覆盖凸包的最小矩形的面积。设凸包有 n 条边, 依靠边 i 得到的最小矩形的面积为 S_i, 我们所要求解的最小矩形面积的 S_m, 那么就有:

$$S_m = min\{S_1, S_2, S_3...S_n\} \tag{2}$$

最后可以得到的效果如下:

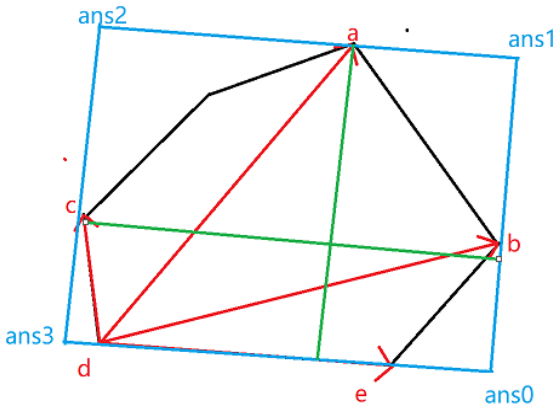

Figure 35: 求解结果示意图

可以看到, 在最终结果的示意图中, 整个凸包被矩形紧紧包围, 这时矩形的面积达到约束条件下的最小。

11.3 解决方案

大体的流程图如下:

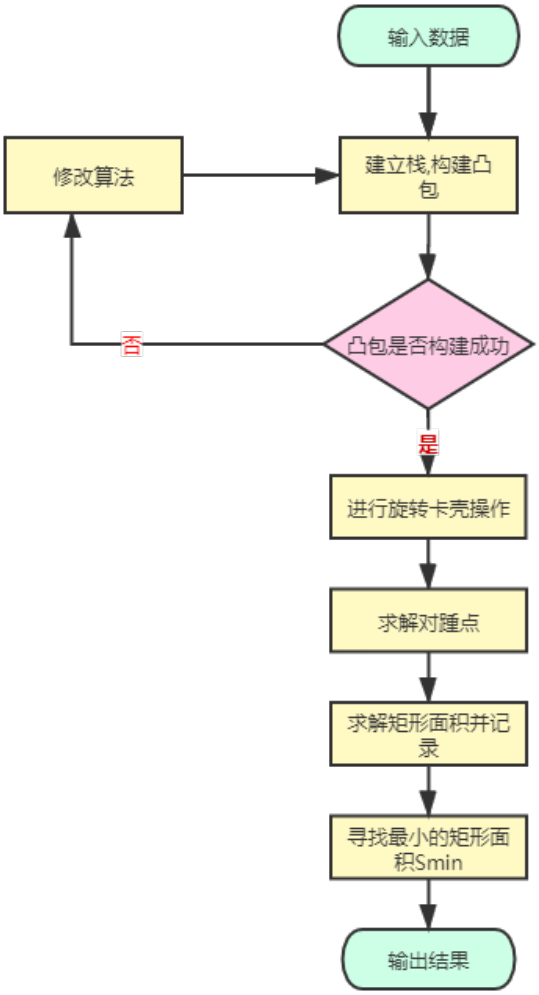

Figure 36: 解决方案流程图

在求解凸包的时候, 使用栈, 将点不断地压栈, 判断新的点是否在所围成区域的内部, 如果不在, 那么将新的点加入凸包的点集, 如此构建下去直至最后, 凸包构建完成。

在求解的时候, 主要的运算是求解对踵点, 对踵点的求解步骤略有复杂, 但是内部几何关系简单, 仍是可以利用几何关系求解。之后利用对踵点, 可以使用向量运算快速求解矩形的面积。

11.4 程序运行结果

对于测试样例, 很轻松就能得到答案:

```
Input:
    3
    -1  0
    1   0
    0   1
Output:
    2.00000
```

Figure 37: K 题测试样例通过截图

我们采用在安徽科技大学的公开 oj 上测试, 通过测试并拿到了第一 AC。测试截图如下:

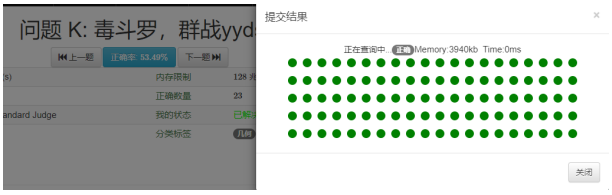

Figure 38: K 题通过截图

12 附录: 代码与实现

12.1 A 题

```cpp
#include <iostream>
#include <cstdio>
#include <algorithm>
using namespace std;
int main() {
    ios_base::sync_with_stdio(false);
    int n;cin >> n;  // 组数
    int a[1000];
    for (int i = 0; i < n; ++i) {
        int x = 0;
        int num, num1, num2, num3;    // 数字总数
        cin >> num >> num1 >> num2 >> num3;
        int n1 = num1;int n2 = num2;int n3 = num3;
        while (n3 && n2 > 1) {
            ++x;
            --n3;
            n2 -= 2;
        }
        if (n3 == 0)
            if (3 * n1 <= n2)
                x = x + n1 + (n2 - 3 * n1) / 7;
            else
                x = x + n2 / 3 + (n1 - n2 / 3 - 3) / 7 + 1;
        else if (n2 == 1) {  // 有3且n2为1
            if (n1 > 1) {
                ++x;
                --n3;
                --n2;
                n1 -= 2;
                if (2 * n1 < n3)
                    ++x;}
            else
                x = x + (n3 - 3) / 7;
        }
        else
            if (n1 * 2 <= n3)
                x = x + n1 + (n3 - 2 * n1) / 7;
            else
                x = x + n3 / 2 + (n1 - 4) / 7 + 1;
        a[i] = x;
    }
    for (int i = 0; i < n; ++i)
        printf("%d\n", a[i]);
    return 0;
}
```

12.2 B 题

```
#include <iostream>
int a[1000][1000];
int b[1000];
using namespace std;
int main()
{

    int n;
    int i, j;
    int s,k;
    k = 1;

    cin >> n;
    i = n;
    j = n;
    for (i = 0;i < n;i++)
    {
        for (j = 0;j < n;j++)
        {
            cin >> s;
            a[i][j] = s;

        }
    }
    i = 0;
    j = 0;
    for (i = 0;i < n;i++)
    {
        for (j = 0;j < n;j++)
        {
            if (a[i][j] == 1)
            {
                k++;
           }

        }
        b[i] = k;
        k = 1;
    }
    for (i = 0;i < n;i++)
    {
        cout << b[i]<<" ";
    }

}
```

12.3 C 题

```
#include <iostream>
using namespace std;
const int nmax = 1010;
int father[nmax];
int sum[nmax];
int tmp[nmax];
int ans = 1;
int find(int x) {
    if (father[x] == x)
        return x;
    else {
        father[x] = find(father[x]);
        return father[x];
    }
}
void Union(int x, int y) {
    int fx = find(x);
    int fy = find(y);
    if (fx != fy) {
        father[fy] = fx;
        sum[fx] += sum[fy];
        ans = ans < sum[fx] ? sum[fx] : ans;
    }
}
void init(int n) {
    for (int i = 0; i < n; ++i) {
        father[i] = i;
        sum[i] = 1;
    }
}
int main() {
    int n;
    ios_base::sync_with_stdio(false);
    cin >> n;init(nmax);
    for (int i = 0; i < n; ++i) {
        int number;
        cin >> number;
        tmp[i] = number;
        for (int j = 0; j < i; ++j) {
            if (tmp[j] > number) {
                Union(j, i);
            }
        }
    }
    cout << ans;
    return 0;
}
```

12.4 D 題

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
struct enemy
{
        int hp;
        int attackTimes;
        int type;
        bool isDead;
};
bool canAllKill(bool cankill[], int n) {
        for (int i = 0; i < n; i++)
                if (cankill[i] == false)
                        return false;
        return true;
}
int findMin(int nums[], int n) {
        int min = nums[0];
        for (int i = 0; i < n; i++)
                if (nums[i] > min)
                        min = nums[i];
        return min;
}
int main() {
        int nums_enemy, hp_Hunger, type_enemy;
        cin >> nums_enemy >> hp_Hunger >> type_enemy;
        vector<vector<enemy>> enemys;
        vector<int> type;
        for (int i = 0; i < nums_enemy; i++)
        {
                enemy temp;
                cin >> temp.hp >> temp.attackTimes >> temp.type
                    ;
                temp.isDead = false;

                if (type.size() == 0)
                {
                        type.push_back(temp.type);
                        vector<enemy> a;
                        a.push_back(temp);
                        enemys.push_back(a);
                        continue;
                }
                bool isCount = false;
                for (int i = 0; i < type.size(); i++)
                        if (type[i] == temp.type)
                                isCount = true;
```

```
        if (isCount)
                for (int i = 0; i < enemys.size(); i++)
                        if (enemys[i][0].type == temp.
                            type)
                        enemys[i].push_back(temp);
        else
        {
                vector<enemy> a;
                a.push_back(temp);
                enemys.push_back(a);
        }
}
int* min = new int[type_enemy];
for (int i = 0; i < enemys.size(); i++)
{
        min[i] = enemys[i][0].attackTimes;
        for (int j = 0; j < enemys[i].size(); j++)
                if (enemys[i][j].attackTimes < min[i])
                    min[i] = enemys[i][j].attackTimes;
if (hp_Hunger < findMin(min, type_enemy))
        cout << "-1";
else
{
        for (int i = 0; i < type_enemy; i++)
        {
                sort(enemys[i].begin(), enemys[i].end()
                    , [](enemy& a, enemy& b) {
                        return a.hp > b.hp;
                        });
        }
        int* index = new int[type_enemy];
        for (int i = 0; i < type_enemy; i++)
                index[i] = 0;
        bool* cankill = new bool[type_enemy];
        for (int i = 0; i < type_enemy; i++)
                cankill[i] = false;

        while (canAllKill(cankill, type_enemy))
        {
                for (int i = 0; i < type_enemy; i++)
                {
                        if (enemys[i][index[i]].
                            attackTimes > hp_Hunger)
                                ++index[i];
                        else
                                cankill[i] = true;
                }
        }
        int hpConsume = enemys[0][index[0]].attackTimes
            ;
```

```
                for (int i = 0; i < type_enemy; i++)
                        if (enemys[i][index[i]].attackTimes >
                            hpConsume)
                                hpConsume = enemys[i][index[i
                                    ]].attackTimes;
                int hpReply = enemys[0][index[0]].hp;
                for (int i = 0; i < type_enemy; i++)
                        if (enemys[i][index[i]].hp < hpReply)
                                hpReply = enemys[i][index[i]].
                                    hp;
                hp_Hunger = hp_Hunger - hpConsume + hpReply;
                cout << hp_Hunger;
        }
}
```

12.5 E 题

```
#include<iostream>
#include<algorithm>
#include<string.h>
using namespace std;
typedef long long ll;
const ll N = 4e6 + 10;
const int M = 600;
int rank[600][2];
int shell[M][4];
struct node{
    int rank;
    int last_order;
    int order;
    string s;
    node * next;
};

bool less_than(string a,string b)
{
    int i,j;
    for ( i =0,j=0; max(i,j)<min(a.length(),b.length()); i++,j
        ++)
    {
        while (a[i]=='0') i++;
        while (b[j]=='0') j++;
        if(a[i]<b[j])
        return true;
        if(a[i]>b[j])
        return false;
    }
    return false;
}
node * create_list(int k,string a)
{
    node * head = new node;
    head->order = 0;
    head->last_order = -1;
    head->s = a;
    head->rank =0;
    node *p2 = head;
    node *p3= head;
    for(int i =0 ;i<k;i++)
    {
        node *p1 = new node;
        p2 = head;
        while (p2->order != shell[i][1])
        {
            p2 = p2->next;
```

```
        }
        p1->s = p2->s;
        if(shell[i][0]==1) p1->s[shell[i][2]] = shell[i][3];
        if(shell[i][0]==2) p1->s[shell[i][2]] = '0';
        p2 = head;
        while (p2)
        {
            if(!less_than(p1->s,p2->s))
            {
                p1->rank = p2->rank;
                p2->rank--;
            }
            else{
                p1->rank = p2->rank-1;
            }
            p2 = p2->next;
        }
        p1->order = i+1;
        p1->last_order = shell[i][1];
        p1->next = nullptr;
        p3->next = p1;
        p3 = p1;
    }
    cout<<endl;
    while (head)
    {
        cout<<head->last_order<<"  "<<head->s<<endl;
        head = head->next;
    }

    return head;
}
int main()
{
    ll n;
    cin>>n;
    string a = new char[n];
    cin>>a;
    int k;
    cin>>k;
    for(int i=0;i<k;i++)

    for(int i = 0;i<k;i++)
    {
        cout<<shell[i][0]<<shell[i][1]<<shell[i][2]<<endl;
    }
    // create_list(k,a);
    return 0;
}
```

12.6 F 题

```
#include<iostream>
#include<vector>

using namespace std;

struct ServicePoint
{
        int vi;
        int xi;
        int yi;
};

struct Worker
{
        int li;
        int ri;
        int ai;
        int bi;
};


int main() {
        int n;
        cin >> n;

        ServicePoint* poses = new ServicePoint[n];
        for (int i = 0; i < n; i++)
        {
                cin >> poses[i].vi >> poses[i].xi >> poses[i].
                    yi;
        }

        int q;
        cin >> q;

        Worker* workers = new Worker[q];
        vector<vector<int>> pos_to_worker;

        for (int i = 0; i < q; i++)
        {
                cin >> workers[i].li >> workers[i].ri >>
                    workers[i].ai >> workers[i].bi;

                vector<int> temp;
                for (int j = 0; j < n; j++)
                {
                        if (poses[j].vi >= workers[i].li &&
                            poses[j].vi <= workers[i].ri)
```

```
                        {
                                temp.push_back(j);
                        }
                }
                pos_to_worker.push_back(temp);
        }

        for (int i = 0; i < q; i++)
        {
                if (pos_to_worker[i].empty())
                {
                        cout << "No" << endl;
                        break;
                }

                int time = workers[i].ai;
                int count = 0;
                while (!pos_to_worker[i].empty() && time <=
                    workers[i].bi && count < pos_to_worker[i].
                    size())
                {
                        for (int j = 0; j < pos_to_worker[i].
                            size(); j++)
                        {
                                if (poses[pos_to_worker[i][j]].
                                    xi <= time && poses[
                                    pos_to_worker[i][j]].yi >=
                                    time)
                                {
                                        time = poses[
                                            pos_to_worker[i][j
                                            ]].yi + 1;
                                }
                                else if (poses[pos_to_worker[i
                                    ][j]].yi == time)
                                {
                                        time++;
                                }
                        }
                        ++count;
                }
                if (time > workers[i].bi)
                        cout << "Yes" << endl;
                else
                    cout << "No" << endl;
        }
}
```

12.7 G 题

```
#include <cstdio>
#include <iostream>
#include <list>
#include <algorithm>
#include <queue>
using namespace std;
struct Node
{
    int pos = 0;
    int visited = 0;  // 以特定顶点起始的不可访问
    int num = 0;    // 被调用的层数
    list<int> lst;  // 边列表
};
int main() {
    int n;
    cin >> n;
    int tmp1 = n + 1;
    Node* G = new Node[tmp1];
    int x, y;
    int sum = 0;    // 总数
    for (int i = 0; i < n - 1; ++i) {
        cin >> x >> y;
        G[x].pos = x;
        G[x].lst.push_back(y);  // 初始化边信息
        G[y].pos = y;
        G[y].lst.push_back(x);
    }
    for (int i = 1; i < n + 1; ++i) {
        G[i].num = 0;   // 格式化
        G[i].visited = i;   // 自身不可访问
        queue<Node> q;
        q.push(G[i]);
        while (!q.empty()) {
            Node tmp = q.front();
            q.pop();
            if (tmp.num != 3) {
                for (auto j : tmp.lst) {
                    if (G[j].visited != i) {
                        G[j].visited = i;   // 不可访问
                        G[j].num = tmp.num + 1; // 更深一层
                        q.push(G[j]);
                    }
                }
            }
            else {
                if (find(tmp.lst.begin(), tmp.lst.end(), i) ==
                    tmp.lst.end()) {
                    ++sum;
```

```
                    G[i].lst.push_back(tmp.pos);
                    G[i].visited = i;
                    G[tmp.pos].lst.push_back(i);
                    G[tmp.pos].num = 1;
                    q.push(G[tmp.pos]);
                }
            }
        }
    }
    printf("%d", sum);
    delete[] G;
    return 0;
}
```

12.8 H 题

```
#include<iostream>
#include<algorithm>
using namespace std;
typedef long long ll;
const int N = 1010;
int q[4][4] = {1};
int al[4][3];
int res[10];
bool is_known[4][4] = {false};

bool slove()
{
    for(int i=0;i<10;i++)
    {if(res[i]!=0)
    return false;}
    return true;
    int max_pos_col = 0;
    for(int i = 0 ;i<4;i++)
    {
        if(res[max_pos_col]<res[i])
        max_pos_col = i;
    }
    int max_other = 4;
    for(int i =4 ;i<10;i++)
    {
        if(res[max_other]<res[i])
        if(!is_known[max_pos_col][i])
            max_other = i;

    }
    if(max_other==8){
    q[max_pos_col][max_pos_col]+=1;
    res[max_pos_col]--;
    res[max_other]--;
    res[max_pos_col+4]--;
    }
    if(max_other==9)
    {
        q[max_pos_col][3-max_pos_col]+=1;
        res[max_pos_col]--;
        res[max_other]--;
        res[7-max_pos_col]--;
    }
    if(max_other!=8&&max_other!=9)
    {
        q[max_pos_col][max_other]+=1;
        res[max_pos_col]-=1;
        res[max_other]-=1;
```

```
    }
    for(int i=0;i<10;i++)
    {if(res[i]!=0)
    return false;}
    return true;

}

int main()
{
    for(int i =0;i<4;i++)
    for(int j = 0;j<4;j++)
    q[i][j] =1;
    int sum = 0;
    for(int i=0;i<10;i++)
    {cin>>res[i];
    res[i]-=4;
    sum+=res[i];
    }
    for(int i=0;i<4;i++)
    {
        cin>>al[i][0]>>al[i][1]>>al[i][2];
        q[al[i][0]][al[i][1]] = al[i][2];
        res[al[i][0]]-=al[i][2];
        res[al[i][1]+4] -= al[i][2];
        if(al[i][0]==al[i][1])
        res[8]-=al[i][2];
        if (al[i][0]+al[i][1]==3)
        res[9]-=(al[i][2]-1);
        is_known[al[i][0]][al[i][1]]=true;
    }
    for (int i = 0; i < sum/2; i++)
    {
        slove();
    }

    for(int i=0;i<4;i++)
   {for(int j = 0;j<4;j++)
    cout<<q[i][j]<<" ";
    cout<<endl;}
    return 0;

}
```

12.9 I 題

```
#include<iostream>
#include<vector>
#include<string>
#include<queue>

using namespace std;

struct node;

bool isHaveVirus(node* nodes, int n);
bool isHaveVirus(node* nodes, int strlen, vector<int>& indexA,
    int index, int n);

struct node
{
        char base;
        vector<int> AdjIndex;
};

struct Ts_with_everA
{
        int index_headA;
        vector<int> index_tailT;
};

int main() {
        int n, m;
        cin >> n >> m;

        node* nodes = new node[n];
        for (int i = 0; i < n; i++)
        {
                cin >> nodes[i].base;

        }


        for (int i = 0; i < m; i++)
        {
                int ai, bi;
                cin >> ai >> bi;
                if (ai == bi)
                {
                        nodes[ai - 1].AdjIndex.push_back(bi -
                            1);
                }
                else
                {
```

```
                        nodes[ai - 1].AdjIndex.push_back(bi -
                            1);
                        nodes[bi - 1].AdjIndex.push_back(ai -
                            1);
                }
        }

        bool flag = isHaveVirus(nodes, n);

        if (flag)
        {
                cout << "Yes";
        }
        else
        {
                cout << "No";
        }

}



bool isHaveVirus(node* nodes, int n)
{
        int strlen = 0;
        vector<int> indexA;
        int index = -1;
        return isHaveVirus(nodes, strlen, indexA, index, n);
}

bool isHaveVirus(node* nodes, int strlen, vector<int>& indexA,
    int index, int n)
{
        if (index == -1)
        {
                for (int i = 0; i < n; i++)
                {
                        if (nodes[i].base == 'A')
                        {
                                indexA.push_back(i);
                                strlen = 1;
                                return isHaveVirus(nodes,
                                    strlen, indexA, i, n);
                        }
                }
                return false;
        }

        if (strlen == 6)
        {
```

```
            for (int i = 0; i < nodes[index].AdjIndex.size
                (); i++)
            {
                    for (int j = 0; j < indexA.size(); j++)
                    {
                            if (nodes[index].AdjIndex[i] ==
                                 indexA[j])
                            {
                                    return true;
                            }
                    }
            }

            //vector<int> indexNextA;
            strlen = 1;

            for (int i = 0; i < nodes[index].AdjIndex.size
                (); i++)
            {
                    if (nodes[nodes[index].AdjIndex[i]].
                        base == 'A')
                    {
                            //indexNextA.push_back(nodes[
                                index].AdjIndex[i]);
                            return isHaveVirus(nodes,
                                strlen, indexA, nodes[index
                                ].AdjIndex[i], n);
                    }
            }

            return false;
    }
    else if (strlen == 0 || strlen == 1)
    {
            strlen++;
            for (int i = 0; i < nodes[index].AdjIndex.size
                (); i++)
            {
                    if (nodes[nodes[index].AdjIndex[i]].
                        base == 'A')
                    {
                            return isHaveVirus(nodes,
                                strlen, indexA, nodes[index
                                ].AdjIndex[i], n);
                    }
            }
            indexA.pop_back();
            return false;
    }
    else if (strlen == 2 || strlen == 3)
```

```
        {
                strlen++;
                for (int i = 0; i < nodes[index].AdjIndex.size
                    (); i++)
                {
                        if (nodes[nodes[index].AdjIndex[i]].
                            base == 'C')
                        {
                                return isHaveVirus(nodes,
                                    strlen, indexA, nodes[index
                                    ].AdjIndex[i], n);
                        }
                }
                indexA.pop_back();
                return false;
        }
        else if (strlen == 4 || strlen == 5)
        {
                strlen++;
                for (int i = 0; i < nodes[index].AdjIndex.size
                    (); i++)
                {
                        if (nodes[nodes[index].AdjIndex[i]].
                            base == 'T')
                        {
                                return isHaveVirus(nodes,
                                    strlen, indexA, nodes[index
                                    ].AdjIndex[i], n);
                        }
                }
                indexA.pop_back();
                return false;
        }
}
```

12.10 J 题

```
#include <iostream>
#include<string>
#include<algorithm>
using namespace std;
char a[26] = {
    'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
        'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
    'u', 'v', 'w', 'x', 'y', 'z'
};
int b[26];
char a2[100];
char b2[100];
char a3[100];
int main() {
    string a1;
    string b1;
    string c1;
    string d1;
    int s1, s2;
    int i, k, d = 101;
    int i1 = 0, n = 0, i2;
    char s;
    bool flag = 0, flag1 = 0;
    getline(cin, a1);
    getline(cin, b1);
    s1 = a1.length();
    s2 = b1.length();
    for (i = 0; i < s1; i++)
        a2[i] = a1[i];
    for (i = 0; i < s2; i++)
        b2[i] = b1[i];
    for (i = 0; i < s1; i++)
        a3[i] = a1[i];
    c1 = a1;
    for (i2 = 0; i2 < 26; i2++) {
        for (i = 0; i < 26; i++) {
            for (k = 0; k < s1; k++) {
                if (a1[k] == a[i]) {
                    d = k;flag1 = 1;}
                if (flag1 == 1) {
                    while (d != i1) {
                        if ((a1[d] > a1[d - 1]) == 1)
                            break;
                        d1 = a1;
                        s = a1[d - 1];
                        a1[d - 1] = a1[d];
                        a1[d] = s;
                        d--;
```

```
                        if(d1!=a1)
                        n++;
                        cout << a1 <<n<< endl;
                        if ((a1 < b1) == 1)
                            break;
                    }
                }
                if (d == i1)
                    i1++;
                flag1 = 0;
                if ((a1 < b1) == 1) {
                    flag = 1;
                    break;
                }
            }
            if ((a1 < b1) == 1) {
                flag = 1;
                break;
            }
        }
        i = 0;
        flag = 0;
        if ((a1 < b1) == 1) {
            flag = 1;
        }
        if (flag == 0) {
            n = 31000;
        }
        cout << a1 << endl;
        i1 = 0;
        d = 101;
        b[i2] = n;
        n = 0;
        s = a[0];
        a[0] = a[i2];
        a[i2] = s;
        a1 = c1;
    }
    sort(b, b + 26);
    if (flag == 0)
        cout << "-1";
    else
        cout << b[0];
    return 0;
}
```

12.11 K 題

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>
#define x first
#define y second
using namespace std;
typedef pair<double, double> PDD;
const int N = 50010;
const double eps = 1e-12, INF = 1e20;
const double PI = acos(-1);
int n;
PDD q[N];
PDD ans[N];
double min_area = INF;
int stk[N], top;
bool used[N];
int sign(double x)
{
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}
int dcmp(double x, double y)
{
    if (fabs(x - y) < eps) return 0;
    if (x < y) return -1;
    return 1;}
PDD operator+ (PDD a, PDD b)
{return {a.x + b.x, a.y + b.y};}
PDD operator- (PDD a, PDD b)
{return {a.x - b.x, a.y - b.y};}
PDD operator* (PDD a, double t)
{return {a.x * t, a.y * t};}
PDD operator/ (PDD a, double t)
{return {a.x / t, a.y / t};}
double operator* (PDD a, PDD b)
{return a.x * b.y - a.y * b.x;}
double operator& (PDD a, PDD b)
{return a.x * b.x + a.y * b.y;}
double area(PDD a, PDD b, PDD c)
{return (b - a) * (c - a);}
double get_len(PDD a)
{return sqrt(a & a);}
double project(PDD a, PDD b, PDD c)
{return ((b - a) & (c - a)) / get_len(b - a);}
PDD norm(PDD a)
{return a / get_len(a);}
```

```
PDD rotate(PDD a, double b)
{return {a.x * cos(b) + a.y * sin(b), -a.x * sin(b) + a.y * cos
    (b)};}
void get_convex()
{
    sort(q, q + n);
    for (int i = 0; i < n; i ++ )
    {
        while (top >= 2 && sign(area(q[stk[top - 2]], q[stk[top
            - 1]], q[i])) >= 0)
            used[stk[ -- top]] = false;
        stk[top ++ ] = i;
        used[i] = true;
    }
    used[0] = false;
    for (int i = n - 1; i >= 0; i -- )
    {
        if (used[i]) continue;
        while (top >= 2 && sign(area(q[stk[top - 2]], q[stk[top
            - 1]], q[i])) >= 0)
            top -- ;
        stk[top ++ ] = i;
    }
    reverse(stk, stk + top);
    top -- ;
}
void rotating_calipers()
{
    for (int i = 0, a = 2, b = 1, c = 2; i < top; i ++ )
    {
        auto d = q[stk[i]], e = q[stk[i + 1]];
        while (dcmp(area(d, e, q[stk[a]]), area(d, e, q[stk[a +
            1]])) < 0) a = (a + 1) % top;
        while (dcmp(project(d, e, q[stk[b]]), project(d, e, q[
            stk[b + 1]])) < 0) b = (b + 1) % top;
        if (!i) c = a;
        while (dcmp(project(d, e, q[stk[c]]), project(d, e, q[
            stk[c + 1]])) > 0) c = (c + 1) % top;
        auto x = q[stk[a]], y = q[stk[b]], z = q[stk[c]];
        auto h = area(d, e, x) / get_len(e - d);
        auto w = ((y - z) & (e - d)) / get_len(e - d);
        if (h * w < min_area)
        {
            min_area = h * w;
            ans[0] = d + norm(e - d) * project(d, e, y);
            ans[3] = d + norm(e - d) * project(d, e, z);
            auto u = norm(rotate(e - d, -PI / 2));
            ans[1] = ans[0] + u * h;
            ans[2] = ans[3] + u * h;
        }
```

```
    }
}

int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i ++ ) scanf("%lf%lf", &q[i].x, &q[i
        ].y);
    get_convex();
    rotating_calipers();

    int k = 0;
    for (int i = 1; i < 4; i ++ )
        if (dcmp(ans[i].y, ans[k].y) < 0 || !dcmp(ans[i].y, ans
            [k].y) && dcmp(ans[i].x, ans[k].x))
            k = i;

    printf("%.5lf\n", min_area);

    return 0;
}
```

13 写在最后

通过程序设计艺术课程, 我们小组的同学第一次接触到了 ACM 这一程序设计赛事, 不仅在课堂上学习到了 STL 容器, 搜索回溯, 计算几何, 动态规划等内容, 并且在私下开展了属于自己的探索, 在这个探索过程中, 我们不仅独立做题, 而且在遇见困难的时候积极地互相讨论, 在这个过程中, 可以说是收获颇丰, 而我们的程序设计能力也相对于之前有了长足进步, 可以说, 这门课程为我们带来了太多太多. 前一段时间宣城校区开展了程序设计校赛, 我们小组几乎全员参加, 并都取得了一定的成绩, 这种欢欣和喜悦是前所未有的, 目前, 小组中也有同学在积极参与我校 ACM 实验室所组织的培训与招新活动, 相信在将来, 我们因为程序设计艺术这门课得到的收获会越来越多, 同时也衷心地希望程序设计艺术这门课程可以在提升学生能力方面继续发光发热, 希望我校的 ACM 成绩越来越好, 学生们的能力也因为这门课越来越强。