

《 算法设计与分析 》

实验报告本

|  |  |
| --- | --- |
| 班 级： | **计213** |
| 学 号： | **21013134** |
| 姓 名： | **徐昊博** |
| 指导教师： | **叶琪** |

信息科学与工程学院

2023年12 月

**实验一 递归与分治算法**

**一、实验目的**

**1、**熟悉数据结构和基本的排序和搜索算法，熟悉编程语言的集成开发环境，掌握程序设计与实现的能力，分析算法的复杂度。

2、通过上机实验，要求掌握递归与分治法算法的问题描述、算法设计思想、程序设计和算法复杂性分析等。

**二、实验装置**

**Win7, Visual C++/Java**

**三**、实验内容

（在第一次实验题目库中选择题目，填写题目内容及输入输出要求）

  1、给定一个大小为n的数组nums，返回其中的多数元素。多数元素是指在数组中出现次数大于的元素。假设数组是非空的，并且给定的数组总是存在多数元素。

2、连续数列。给定一个整数数组，找出总和最大的连续数列，并返回总和。

3、  至少有K个重复字符的最长子串。给定一个字符串S和一个整数k，找出S中的最长子串，要求该子串中的每一个字符出现次数都不少于k。返回这一子串的长度。如果不存在这样的子字符串，则返回0。（S仅由小写字母组成）

4、最长的美好子字符串。当一个字符串s包含的每一种字母的大写和小写形式同时出现在s中，就称这个字符串s是美好字符串。比如，“abABB”是美好字符串，因为“A”和“a”同时出现了，且“B”和“b”也同时出现了。然而，“abA”不是美好字符串因为‘b’出现了，而‘B’没有出现。给定一个字符串s，请返回s最长的美好子字符串。如果有多个答案，请返回最早出现的一个。如果不存在美好子字符串，请返回一个空字符串。

5、库存管理。仓库管理员以数组stock形式记录商品库存表，其中stock[i]表示对应商品库存余量。请返回库存余量最少的cnt个商品余量，返回顺序不限。

6、The highest building in our city has only one elevator. A request list is made up with N positive numbers. The numbers denote at which floors the elevator will stop, in specified order. It costs 6 seconds to move the elevator up one floor, and 4 seconds to move down one floor. The elevator will stay for 5 seconds at each stop.  
For a given request list, you are to compute the total time spent to fulfill the requests on the list. The elevator is on the 0th floor at the beginning and does not have to return to the ground floor when the requests are fulfilled.

**Input**

There are multiple test cases. Each case contains a positive integer N, followed by N positive numbers. All the numbers in the input are less than 100. A test case with N = 0 denotes the end of input. This test case is not to be processed.

**Output**

Print the total time on a single line for each test case.

7、A number sequence is defined as follows:  
  
f(1) = 1, f(2) = 1, f(n) = (A \* f(n - 1) + B \* f(n - 2)) mod 7.  
  
Given A, B, and n, you are to calculate the value of f(n).

**Input**

The input consists of multiple test cases. Each test case contains 3 integers A, B and n on a single line (1 <= A, B <= 1000, 1 <= n <= 100,000,000). Three zeros signal the end of input and this test case is not to be processed.

**Output**

For each test case, print the value of f(n) on a single line.

8、A simple mathematical formula for e is  
  
  
  
where n is allowed to go to infinity. This can actually yield very accurate approximations of e using relatively small values of n.

**Output**

Output the approximations of e generated by the above formula for the values of n from 0 to 9. The beginning of your output should appear similar to that shown below.

9、These days, I am thinking about a question, how can I get a problem as easy as A+B? It is fairly difficulty to do such a thing. Of course, I got it after many waking nights.  
Give you some integers, your task is to sort these number ascending (升序).  
You should know how easy the problem is now!  
Good luck!

**Input**

Input contains multiple test cases. The first line of the input is a single integer T which is the number of test cases. T test cases follow. Each test case contains an integer N (1<=N<=1000 the number of integers to be sorted) and then N integers follow in the same line.  
It is guarantied that all integers are in the range of 32-int.

**Output**

For each case, print the sorting result, and one line one case.

10、Larry graduated this year and finally has a job. He’s making a lot of money, but somehow never seems to have enough. Larry has decided that he needs to grab hold of his financial portfolio and solve his financing problems. The first step is to figure out what’s been going on with his money. Larry has his bank account statements and wants to see how much money he has. Help Larry by writing a program to take his closing balance from each of the past twelve months and calculate his average account balance.

**Input**

The input will be twelve lines. Each line will contain the closing balance of his bank account for a particular month. Each number will be positive and displayed to the penny. No dollar sign will be included.

**Output**

The output will be a single number, the average (mean) of the closing balances for the twelve months. It will be rounded to the nearest penny, preceded immediately by a dollar sign, and followed by the end-of-line. There will be no other spaces or characters in the output.

**四、程序运行结果**

**（说明设计思路，解释使用的数据结构，计算时间复杂度）**

1.设计思路：

count\_in\_range 函数用于计算在指定范围内数组中特定元素 target 的出现次数。

majority\_element\_rec 函数是递归的，它将数组分成左右两部分，并分别求出左右两部分的主要元素。

然后检查左右两部分的主要元素在整个区间中出现的次数，如果超过了该区间长度的一半，则返回这个主要元素；否则返回 -1。

majorityElement 函数是主函数，调用 majority\_element\_rec 函数来找出整个数组的主要元素。

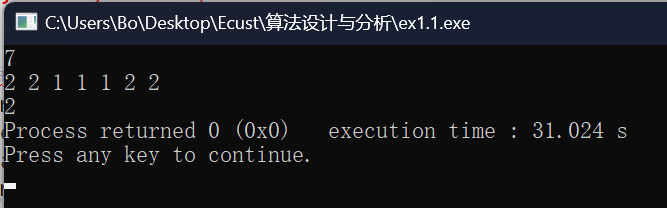
使用的数据结构：

代码中主要使用了基本的数组数据结构来存储整数元素。

时间复杂度分析：

count\_in\_range 函数中的循环对指定范围内的元素进行遍历，时间复杂度为 O(hi - lo)，其中 hi 和 lo 分别是范围的上界和下界。

majority\_element\_rec 函数中进行了递归，每次将数组分成两半，因此时间复杂度可以表示为递归树的高度。最坏情况下，递归深度为 O(log n)，其中 n 是数组的长度。

在最坏情况下，count\_in\_range 函数在每一层递归中都需要线性时间。因此，最终的时间复杂度为 O(n log n)，其中 n 是数组的长度。

2． 设计思路：

Status 结构体用于维护了四个变量：lSum 表示左侧连续子数组的最大和，rSum 表示右侧连续子数组的最大和，mSum 表示跨越中点的最大子数组和，iSum 表示整个区间的总和。

pushUp 函数用于将左右子区间的 Status 结构体合并为一个更大区间的 Status 结构体。

get 函数使用分治的方法，在每个区间内递归地计算出四个变量的值，并使用 pushUp 函数合并左右子区间的结果。

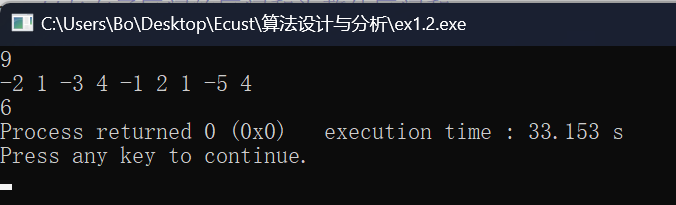
maxSubArray 函数是主函数，调用 get 函数来找出整个数组的最大子数组和。

使用的数据结构：

代码中主要使用了基本的数组数据结构和自定义的结构体 Status 来实现算法。

时间复杂度分析：

get 函数是一个递归函数，将数组分成左右两半，在每个递归步骤中需要进行常数次操作。因此，总的时间复杂度为 O(n)，其中 n 是数组的长度。



3、设计思路：

dfs 函数是这个算法的核心。它通过递归地找到满足条件的最长子串。

该函数使用了分治的思想：首先统计字符串中各个字符出现的次数，然后找出出现次数小于 k 的字符作为分隔符。

如果找到了这样的分隔符，则将字符串分成多个子串，对每个子串递归调用 dfs 函数进行处理。

longestSubstring 函数是主函数，它接受输入字符串和 k 值，并调用 dfs 函数来找出最长的满足条件的子串长度。

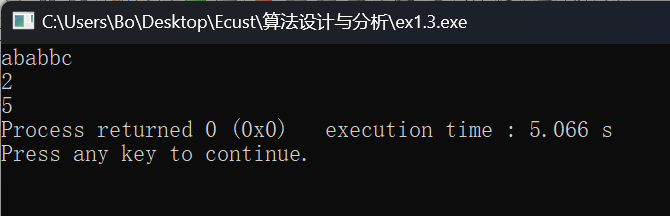
使用的数据结构：

代码中主要使用了字符数组和一些基本的变量（如 int 类型的变量）来存储和处理字符串以及计算结果。

时间复杂度分析：

dfs 函数是递归调用的，其时间复杂度取决于递归的深度和每层的操作复杂度。

在最坏情况下，每次递归都需要遍历一次字符串，并且递归的层数与字符串长度成正比。因此，最坏情况下的时间复杂度是 O(n^2)，其中 n 是字符串的长度。



4、设计思路：

dfs 函数是核心函数，它递归地查找特殊子串。首先检查字符串中是否存在某字符的大写和小写同时出现，并且位置在给定的范围内。若不存在，则将区间划分为两部分并分别递归处理。

如果存在特殊子串（即整个区间内都满足上述条件），则更新 max 变量表示的最大长度，并用 res 数组存储这个特殊子串。

使用的数据结构：

代码使用了字符数组和一些基本的变量来处理字符串和计算结果。

时间复杂度分析：

dfs 函数中的递归在每次迭代中都会对字符串进行查找和递归调用。在最坏情况下，它的时间复杂度可能达到指数级别。

最终结果的输出过程复杂度取决于特殊子串的长度，但是由于整个过程是在递归函数中完成的，因此整体时间复杂度与特殊子串的长度和输入字符串长度相关。



5、设计思路：

Solution 类中的 partition 函数是快速排序中的一部分，用于对数组进行分区。

randomized\_partition 函数是基于随机选取元素的方式进行划分，它调用 partition 函数对数组进行分区。

randomized\_selected 函数是递归实现的，根据快速排序的思想，在每次递归中根据 cnt 和数组元素的个数选择下一步处理的区间。

inventoryManagement 函数是公共接口函数，它初始化随机种子并调用 randomized\_selected 函数，最终返回选取的 cnt 个元素组成的向量。

main 函数用于接受输入，创建 Solution 对象并调用 inventoryManagement 函数，并打印结果。

使用的数据结构：

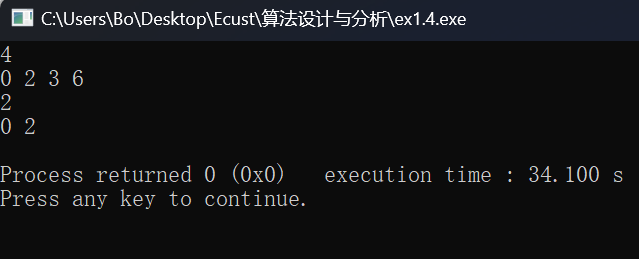
代码主要使用了 vector 来存储库存数组和最终选择的元素。

时间复杂度分析：

在 randomized\_selected 函数中，快速排序的时间复杂度为平均情况下的 O(n log n)，其中 n 是数组的长度。

在最坏情况下，时间复杂度可能会达到 O(n^2)，但因为是随机选取元素进行划分，通常情况下快速排序的时间复杂度为 O(n log n)。

最终 inventoryManagement 函数的时间复杂度主要取决于快速排序的时间复杂度。



6、设计思路：

代码通过一个 while 循环来模拟多组测试用例的处理，直到输入的 n 为 0 才停止。

在每一组测试用例中，首先初始化 floor 为 0，并且 ans 为 0。

接下来通过循环读取每一层楼的目标楼层 f，根据当前楼层与目标楼层之间的关系，计算移动电梯的时间。

每次移动都会消耗一定的时间，根据题目规定的条件，计算出移动的总时间并更新 ans。

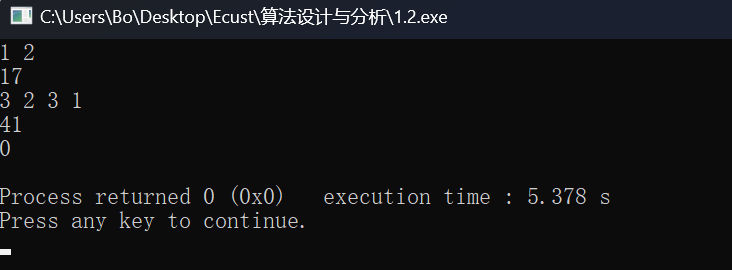
最后输出 ans 表示电梯移动的总时间。

使用的数据结构：

代码主要使用了基本的变量和循环结构，没有使用特定的数据结构。

时间复杂度分析：

代码中的时间复杂度主要取决于循环的次数和每次循环中的计算量。在最坏情况下，代码中的循环次数和输入的楼层数有关。平均为O（n）。



7、设计思路：

seq 函数根据给定的递推式 ans=(A\*seq(A,B,n-1)+B\*seq(A,B,n-2))%7 计算数列中第 n 个数的值。

在 main 函数中，通过一个 while 循环读取输入，直到输入的 a、b、n 全部为 0 才停止。

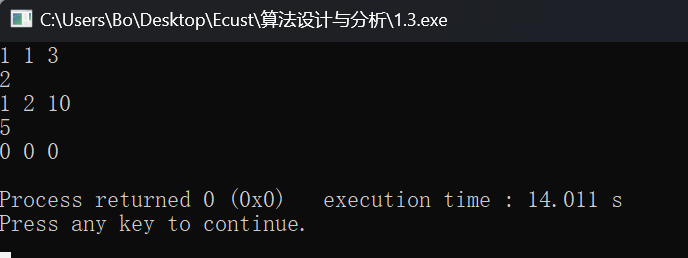
在每次输入后，调用 seq 函数计算出第 n 个数的值，并输出结果。

使用的数据结构：

代码主要使用了基本的变量和循环结构，没有使用特定的数据结构。

时间复杂度分析：

在 seq 函数中，计算第 n 个数的值需要依次递归计算直到递归到 n=1 或 n=2 时终止。因此，该递归的时间复杂度是指数级别的，并且与 n 相关。



8、

设计思路：

fac 函数用于计算阶乘，通过递归实现，计算了给定整数 n 的阶乘。

在 main 函数中，使用循环遍历 i 从 0 到 9，计算数列的前几项和。

数列的第 i 项的值 m 通过不断累加 1/fac(i) 得到，并将每一项的结果输出。

输出结果分为两部分：当 i <= 2 时，输出结果保留小数点后 0 位或 1 位；而对于 i > 2，输出结果保留小数点后 9 位。

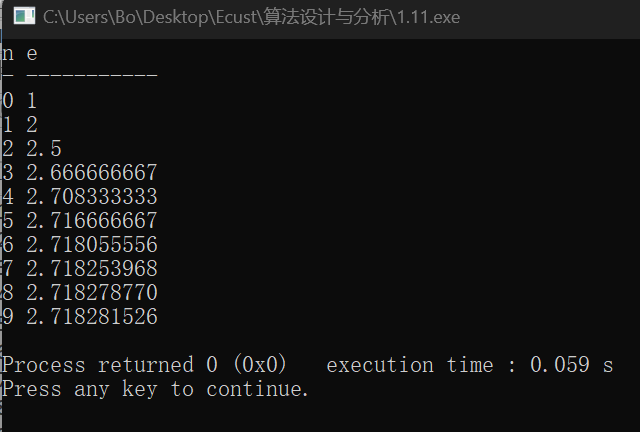
使用的数据结构：

代码主要使用了基本的变量、循环结构和递归函数来计算阶乘。

时间复杂度分析：

fac 函数中采用递归方式计算阶乘，时间复杂度是 O(n)。

在 main 函数中，通过循环遍历求和，整体时间复杂度为 O(1)，因为循环次数是固定的。



9、

设计思路：

使用 int a[1000]; 声明了一个长度为 1000 的整型数组，用于存储每个测试用例中的元素。

通过 cin >> T; 获取测试用例的数量 T。

使用 while (T--) 循环来处理每个测试用例：通过 cin >> n; 获取当前测试用例中数组的元素个数 n。

使用 for 循环读取 n 个元素，并存储到数组 a[] 中。

使用 sort(a, a + n); 对数组 a[] 中的元素进行排序。

使用 for 循环输出排序后的数组元素。

循环直至处理完所有测试用例。

使用的数据结构：

代码主要使用了整型数组 a[] 和基本的输入输出函数来实现排序并输出结果。

时间复杂度分析：

对于每个测试用例，首先需要读取 n 个整数，并对这 n 个整数进行排序。

排序所需时间取决于使用的排序算法，通常情况下 sort() 函数的时间复杂度为 O(n log n)，其中 n 是数组长度。



10、

设计思路：

使用 for 循环，从 1 到 12 输入每个月的存款数额 m。

在每次循环中，将输入的月存款数额 m 除以 12，并将结果累加到变量 ans 中。

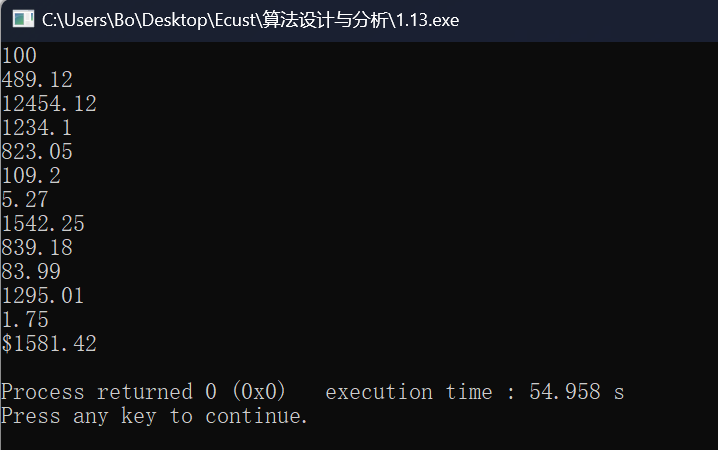
循环结束后，将累加后的结果除以 12（也就是计算了 12 个月的平均值），并将结果输出为一个带有美元符号的金额。

使用的数据结构：

代码主要使用了浮点数变量和循环结构来实现计算平均值并输出结果。

时间复杂度分析：

代码只有一个简单的 for 循环，循环次数固定为 12 次，因此时间复杂度为 O(1)，是一个常数时间的操作。



**实验一成绩：\_\_\_\_\_\_\_\_\_\_\_\_ 教师签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**实验二 动态规划和贪心算法**

**一、实验目的**

1、理解动态规划法的设计思想，分析是否满足最优子结构性质，刻画其结构特征，递归地定义最优值（动态规划方程），以自底向上的方式计算出最优值，构造最优解。掌握动态规划的算法框架和设计策略。

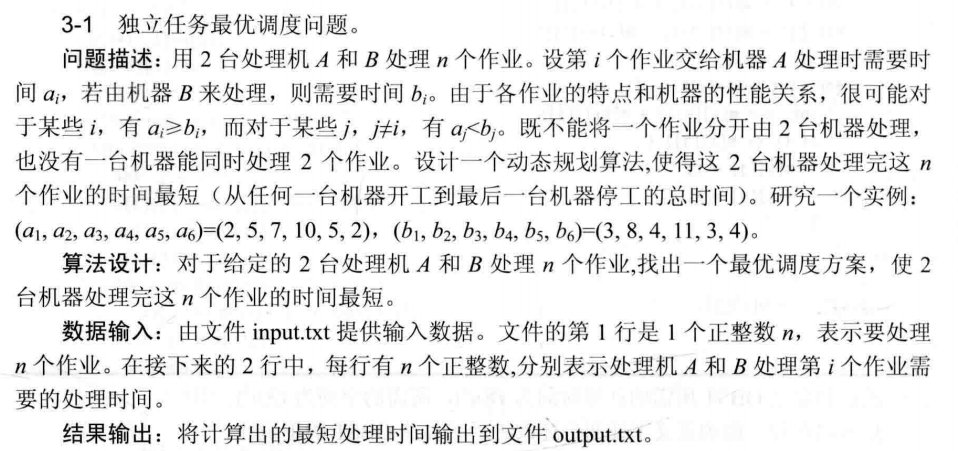
2、理解贪心算法的设计思想，掌握贪心算法的算法框架和设计策略，选取度量标准，逐步构造最优解。

**二、实验装置**

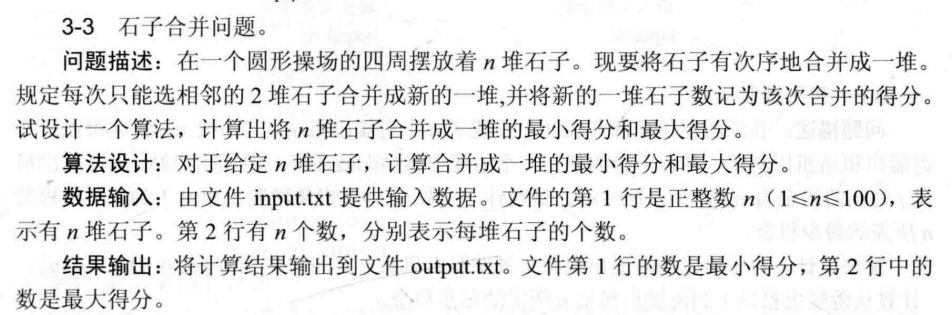
**Win7, Visual C++/Java**

**三、实验内容**

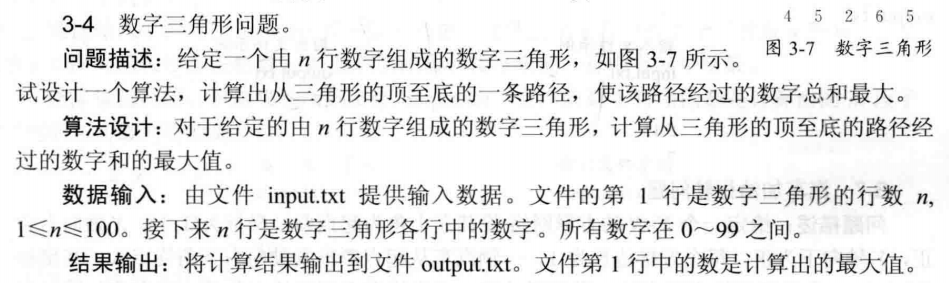
**（**在第一次实验题目库中选择题目，填写题目内容及输入输出要求**）**

1.

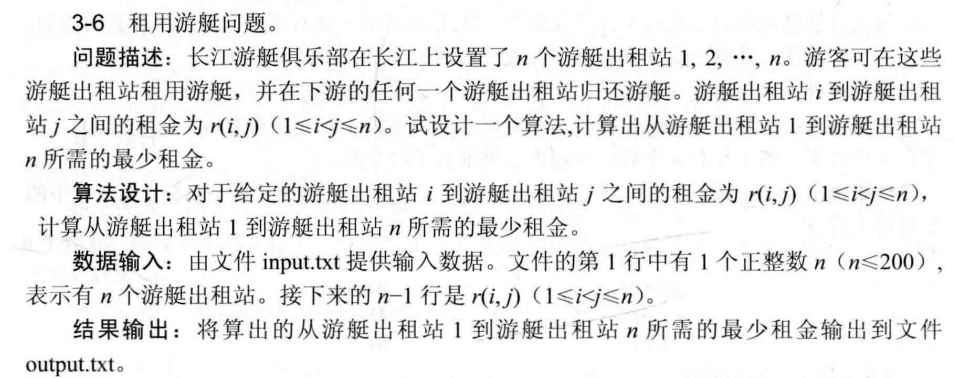
2．

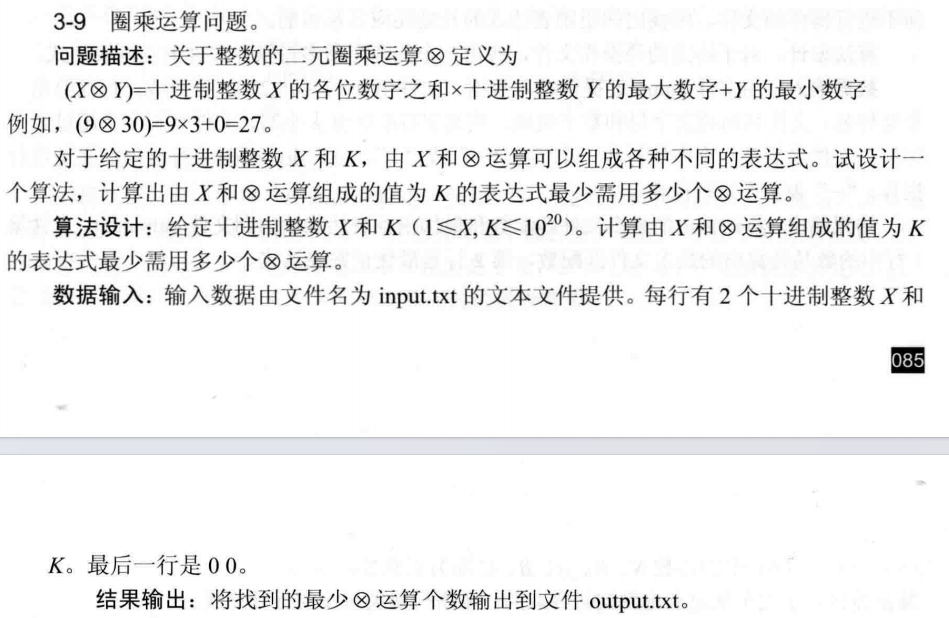


3．



4、



5、

6、

**1. Max Sum**

Problem Description

Given a sequence a[1],a[2],a[3]......a[n], your job is to calculate the max sum of a sub-sequence. For example, given (6,-1,5,4,-7), the max sum in this sequence is 6 + (-1) + 5 + 4 = 14.

Input

The first line of the input contains an integer T(1<=T<=20) which means the number of test cases. Then T lines follow, each line starts with a number N(1<=N<=100000), then N integers followed(all the integers are between -1000 and 1000).

Output

For each test case, you should output two lines. The first line is "Case #:", # means the number of the test case. The second line contains three integers, the Max Sum in the sequence, the start position of the sub-sequence, the end position of the sub-sequence. If there are more than one result, output the first one. Output a blank line between two cases.

7、

**3.Employment Planning**

Problem Description

A project manager wants to determine the number of the workers needed in every month. He does know the minimal number of the workers needed in each month. When he hires or fires a worker, there will be some extra cost. Once a worker is hired, he will get the salary even if he is not working. The manager knows the costs of hiring a worker, firing a worker, and the salary of a worker. Then the manager will confront such a problem: how many workers he will hire or fire each month in order to keep the lowest total cost of the project.

Input

The input may contain several data sets. Each data set contains three lines. First line contains the months of the project planed to use which is no more than 12. The second line contains the cost of hiring a worker, the amount of the salary, the cost of firing a worker. The third line contains several numbers, which represent the minimal number of the workers needed each month. The input is terminated by line containing a single '0'.

Output

The output contains one line. The minimal total cost of the project.

8、

Problem Description

Over centuries ago, mankind faced a new enemy, the Titans. The difference of power between mankind and their newfound enemy was overwhelming. Soon, mankind was driven to the brink of extinction. Luckily, the surviving humans managed to build three walls: Wall Maria, Wall Rose and Wall Sina. Owing to the protection of the walls, they lived in peace for more than one hundred years.

But not for long, a colossal Titan appeared out of nowhere. Instantly, the walls were shattered, along with the illusory peace of everyday life. Wall Maria was abandoned and human activity was pushed back to Wall Rose. Then mankind began to realize, hiding behind the walls equaled to death and they should manage an attack on the Titans.

So, Captain Levi, the strongest ever human being, was ordered to set up a special operation squad of N people, numbered from 1 to N. Each number should be assigned to a soldier. There are three corps that the soldiers come from: the Garrison, the Recon Corp and the Military Police. While members of the Garrison are stationed at the walls and defend the cities, the Recon Corps put their lives on the line and fight the Titans in their own territory. And Military Police serve the King by controlling the crowds and protecting order. In order to make the team more powerful, Levi will take advantage of the differences between the corps and some conditions must be met.

The Garrisons are good at team work, so Levi wants there to be at least M Garrison members assigned with continuous numbers. On the other hand, members of the Recon Corp are all elite forces of mankind. There should be no more than K Recon Corp members assigned with continuous numbers, which is redundant. Assume there is unlimited amount of members in each corp, Levi wants to know how many ways there are to arrange the special operation squad.

Input

There are multiple test cases. For each case, there is a line containing 3 integers N (0 < N < 1000000), M (0 < M < 10000) and K (0 < K < 10000), separated by spaces.

Output

One line for each case, you should output the number of ways mod 1000000007.

9、

有一头母牛，它每年年初生一头小母牛。每头小母牛从第四个年头开始，每年年初也生一头小母牛。请编程实现在第n年的时候，共有多少头母牛？

**Input**

输入数据由多个测试实例组成，每个测试实例占一行，包括一个整数n(0<n<55)，n的含义如题目中描述。  
n=0表示输入数据的结束，不做处理。

**Output**

对于每个测试实例，输出在第n年的时候母牛的数量。  
每个输出占一行。

10、

We give the following inductive definition of a “regular brackets” sequence:

* the empty sequence is a regular brackets sequence,
* if *s* is a regular brackets sequence, then (*s*) and [*s*] are regular brackets sequences, and
* if *a* and *b* are regular brackets sequences, then *ab* is a regular brackets sequence.
* no other sequence is a regular brackets sequence

For instance, all of the following character sequences are regular brackets sequences:

(), [], (()), ()[], ()[()]

while the following character sequences are not:

(, ], )(, ([)], ([(]

Given a brackets sequence of characters *a*1*a*2 … *an*, your goal is to find the length of the longest regular brackets sequence that is a subsequence of *s*. That is, you wish to find the largest *m* such that for indices *i*1, *i*2, …, *im* where 1 ≤ *i*1 < *i*2 < … < *im* ≤ *n*, *ai*1*ai*2 … *aim* is a regular brackets sequence.

Given the initial sequence ([([]])], the longest regular brackets subsequence is [([])].

**Input**

The input test file will contain multiple test cases. Each input test case consists of a single line containing only the characters (, ), [, and ]; each input test will have length between 1 and 100, inclusive. The end-of-file is marked by a line containing the word “end” and should not be processed.

**Output**

For each input case, the program should print the length of the longest possible regular brackets subsequence on a single line.

**四、程序运行结果**

**（说明设计思路，解释使用的数据结构，计算时间复杂度）**



设计思路：

从名为 input.txt 的文件中读取输入数据。

输入的第一行是整数 n，表示任务数量。

后续的两行分别是数组 a[] 和 b[] 中的元素。

使用动态规划求解问题。

最终将得到的最小处理时间写入到名为 output.txt 的文件中。

使用的数据结构：

n, a[], b[]: 用于存储任务数量和任务处理机 a、b 的时间。

dp[][]: 二维数组，用于动态规划。dp[i][j] 表示前 i 个任务中，用时小于等于 j 时 b 处理机的用时。

时间复杂度分析：

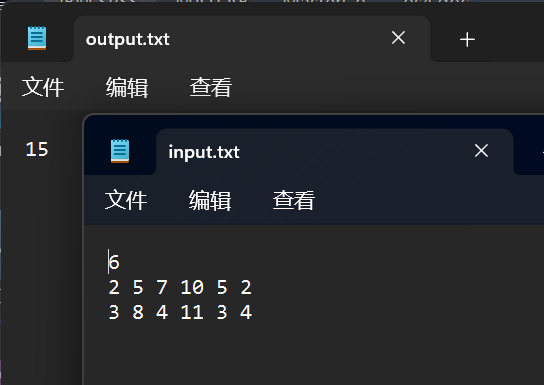
从 input.txt 文件中读取数据的时间复杂度为 O(n)，其中 n 是任务数量。

初始化 dp 数组的时间复杂度为 O(n \* sum)，其中 sum 表示数组 a[] 中所有元素的总和。

动态规划求解的过程中，嵌套了两个循环，时间复杂度为 O(n \* sum)。

计算最小处理时间的过程中，又有一个循环，时间复杂度为 O(sum)。

因此，总体时间复杂度为 O(n \* sum)。



2．

设计思路：

从标准输入中读取一个整数 n，表示序列的长度。

接下来读取 n 个整数，保存在数组 a[] 中，并计算数组 sum[]，其中 sum[i] 表示数组 a[] 前 i 个元素的和。

使用动态规划解决问题。MIN[i][j] 和 MAX[i][j] 分别表示将序列从位置 i 划分到位置 j 之间的最小和最大值。

通过嵌套的循环，遍历不同区间的可能性，更新最小和最大值。

最终输出序列划分成不同区间所能得到的最小和最大值。

使用的数据结构：

n, a[], sum[]: 整型数组，用于存储序列长度、序列元素和前缀和。

MIN[][] 和 MAX[][]: 二维数组，用于动态规划。MIN[i][j] 和 MAX[i][j] 表示将序列从位置 i 划分到位置 j 之间的最小和最大值。

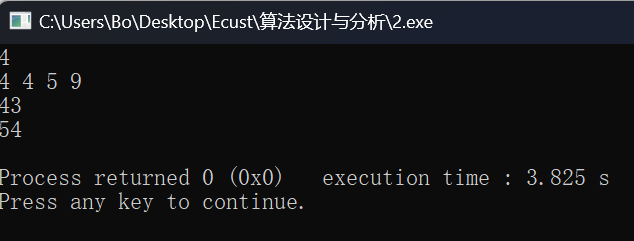
时间复杂度分析：

读取输入数据的时间复杂度为 O(n)，其中 n 表示序列长度。

填充 MIN[][] 和 MAX[][] 的初始值的时间复杂度为 O(n^2)。

动态规划的求解过程中有三个嵌套循环，时间复杂度为 O(n^3)。

因此，总体时间复杂度为 O(n^3)。



3．

设计思路：

从标准输入中读取一个整数 n，表示三角形矩阵的行数。

读取三角形矩阵的每一行，保存在二维数组 a[][] 中。

使用动态规划求解最大路径和问题。f[i][j] 表示到达第 i 行第 j 列的最大路径和。

通过遍历三角形矩阵，更新动态规划数组 f[][] 中的值。

找到最后一行中的最大值，即为最大路径和。

使用的数据结构：

n, a[][]: 整型变量和二维数组，用于存储三角形矩阵的行数和具体元素。

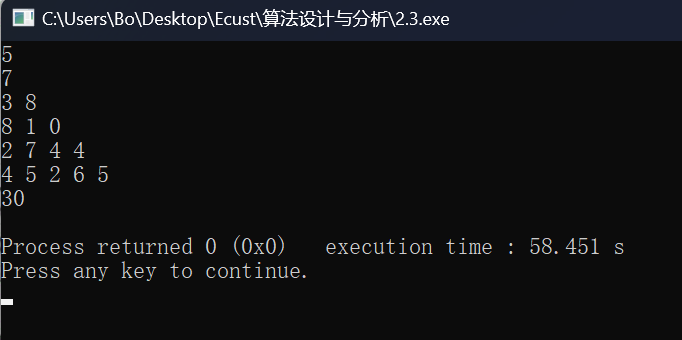
时间复杂度分析：

读取输入数据的时间复杂度为 O(n^2)，其中 n 表示三角形矩阵的行数。

动态规划求解过程中有两个嵌套循环，时间复杂度为 O(n^2)。

遍历最后一行找到最大值的时间复杂度为 O(n)。

因此，总体时间复杂度为 O(n^2)。



4、

设计思路：

从标准输入中读取一个整数 n，表示一个图的节点数量。

读取 n 行数据，填充二维数组 r[][]，表示图中节点之间的路径长度。

使用动态规划求解最短路径问题。dp[i] 表示从节点 1 到节点 i 的最短路径长度。

通过遍历节点，更新动态规划数组 dp[] 中的值，得到最终节点 n 的最短路径长度。

使用的数据结构：

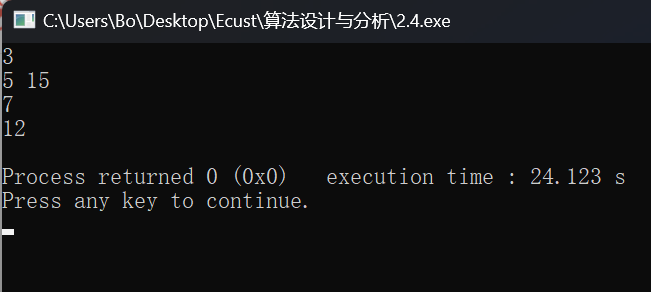
n, r[][]: 整型变量和二维数组，用于存储节点数量和节点之间的路径长度。

时间复杂度分析：

读取输入数据的时间复杂度为 O(n^2)，其中 n 表示节点数量。

动态规划求解过程中有两个嵌套循环，时间复杂度为 O(n^2)。

因此，总体时间复杂度为 O(n^2)。



5、

设计思路：

从标准输入中读取两个字符串 s1 和 s2。

将字符串 s1 转换为整数 len（字符串长度）和大整数 big（预设的较大值）。

将字符串 s2 转换为整数 kk。

根据输入的字符串和整数初始化 num[][] 数组。

使用动态规划计算数字转换的最小步数。

如果找到转换步数小于最大值 MAX，则输出最小步数，否则输出 -1。

使用的数据结构：

s1, s2: 字符串，用于存储输入的两个字符串。

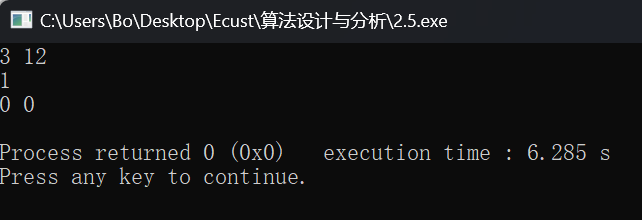
num[][]: 二维数组，用于记录字符串转换数字时的状态（初始值、最小值、最大值等）。

时间复杂度分析：

字符串转整数操作的时间复杂度为 O(s)，其中 s 表示字符串长度。

初始化部分时间复杂度为 O(big \* 10)，其中 big 表示预设的较大值。

动态规划部分使用了两层嵌套循环，时间复杂度最坏情况下可能达到 O(big^2 \* big)。



6、

设计思路：

从标准输入中读取测试用例数量 test。

对每个测试用例：读取该用例中数组的长度 n 和数组元素。

使用动态规划计算最大子段和并记录在数组 d[] 中，其中 d[i] 表示以第 i 个元素结尾的最大子段和。

找到最大子段和的值 max 以及其起始位置 f 和结束位置 e。

输出该测试用例的结果。

使用的数据结构：

test, n, a[], d[]: 整型变量和数组，用于存储测试用例数量、数组长度以及数组元素。

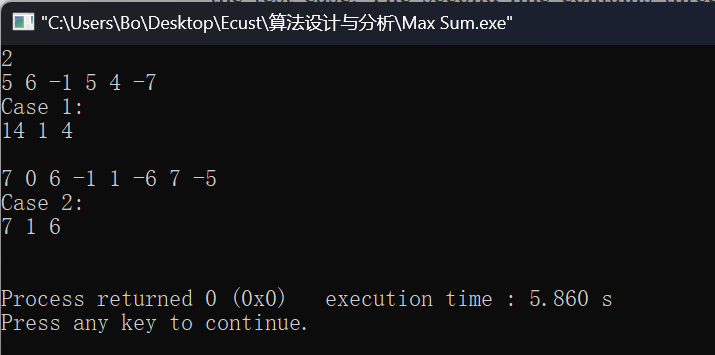
max, f, e: 整型变量，用于记录最大子段和的值以及其起始位置和结束位置。

时间复杂度分析：

外部循环遍历测试用例，时间复杂度为 O(test)。

内部使用了两次循环，对数组进行了遍历操作，时间复杂度为 O(n)。

因此，总体时间复杂度为 O(test \* n)。



7、

设计思路：

从标准输入中循环读取测试用例，直到遇到输入为 0 的情况。

对每个测试用例进行如下操作：读取输入的三个整数 n, x, y, z。

根据 n 的值，依次读取数组 a[] 的值，并找到数组元素的最大值 m。

初始化动态规划数组 dp[][]，大小为 dp[15][20005]，并将其初始化为一个较大的值。

使用动态规划计算最小代价。

输出最小代价。

使用的数据结构：

n, x, y, z, m: 整型变量，用于存储输入的值以及数组元素的最大值。

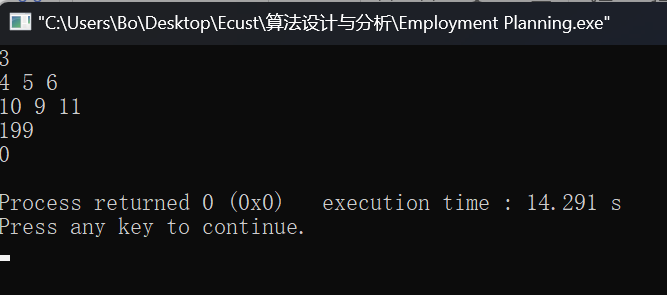
a[]: 整型数组，用于存储每个测试用例中的数组元素。

dp[][]: 二维整型数组，用于动态规划的状态转移。

时间复杂度分析：

外部循环对每个测试用例进行遍历，时间复杂度为 O(T)，其中 T 表示测试用例的数量。

内部使用了三次嵌套循环，时间复杂度较高，在最坏情况下达到 O(n \* m^2)，其中 n 表示数组长度，m 表示数组元素的最大值。



8、

设计思路：

从标准输入中读取三个整数 n, m, k。

使用动态规划数组 dp[][]，通过循环计算状态值，并在特定条件下进行状态转移。

计算 dp[n][1] + dp[n][2] + dp[n][3] 的值，再进行一些调整，得到最终的结果 ans。

输出结果 ans。

使用的数据结构：

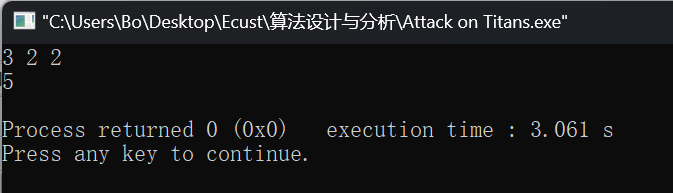
n, m, k: 整型变量，存储输入的三个整数。

dp[][]: 二维长整型数组，用于动态规划的状态转移。

时间复杂度分析：

整体代码使用了嵌套循环，并对 dp[][] 进行了两轮状态的计算和更新，因此总体时间复杂度较高，为 O(n)。

具体时间复杂度因输入值的大小而异。



9、

设计思路：

使用循环将数组 f[] 的前四个值初始化为 1、2、3、4。

使用循环计算并填充数组 f[] 的剩余值，根据规则 f[i] = f[i-1] + f[i-3]。

通过读取标准输入的整数 n，输出数组 f[] 在位置 n 处的值。

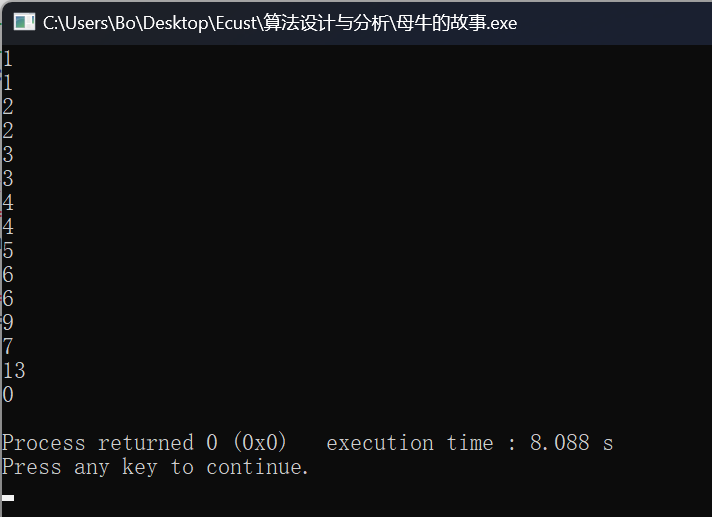
使用的数据结构：

n: 整型变量，用于接收输入的整数。

f[]: 无符号长整型数组，存储数列的值。

时间复杂度分析：

循环初始化数组 f[] 和计算数组 f[] 的值，时间复杂度为 O(N)，其中 N 为数组 f[] 的长度。



10、

使用一个 while 循环不断地读取输入的字符串。

当输入的字符串为 "end" 时，退出循环。

对于每个输入的字符串 a：初始化变量 ans 为 0，用于记录匹配的括号数量。

使用变量 left1 和 left2 分别表示当前未匹配的左括号 '(' 和 '[' 的数量。

逐个扫描字符串中的字符，对于 '('、')'、'['、']' 分别检查：如果是左括号 '(' 或者 '['，分别增加对应的未匹配左括号的数量 left1 或 left2。

如果是右括号 ')' 或者 ']'，判断是否有足够的未匹配左括号可以与之匹配：如果有对应的未匹配左括号，就将对应的未匹配左括号数量减少，并且增加 ans 的值（表明匹配成功）。

输出当前字符串中成功匹配的括号对数 ans。

数据结构：

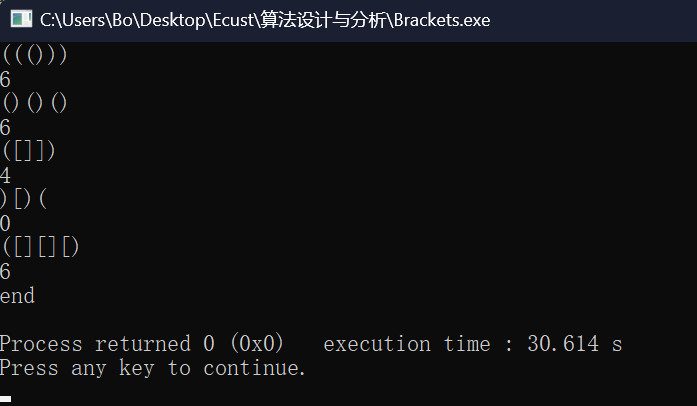
string a: 用于存储输入的字符串。

int left1, int left2: 分别表示未匹配的左圆括号 '(' 和左方括号 '[' 的数量。

int ans: 用于记录匹配成功的括号对数。

时间复杂度：

对于每个输入字符串，程序会线性地扫描其中的字符，因此时间复杂度为 O(n)，其中 n 是输入字符串的长度。



**实验二成绩：\_\_\_\_\_\_\_\_\_\_\_\_ 教师签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**实验三 回溯算法**

**一、实验目的**

**1、**理解回溯法的设计思想，回溯法是一个既带有系统性又带有跳跃性的搜索算法。掌握从包含问题的所有解的解空间树中，按照深度优先的策略，从根结点出发搜索解空间树的过程。掌握回溯法的算法框架和设计策略。

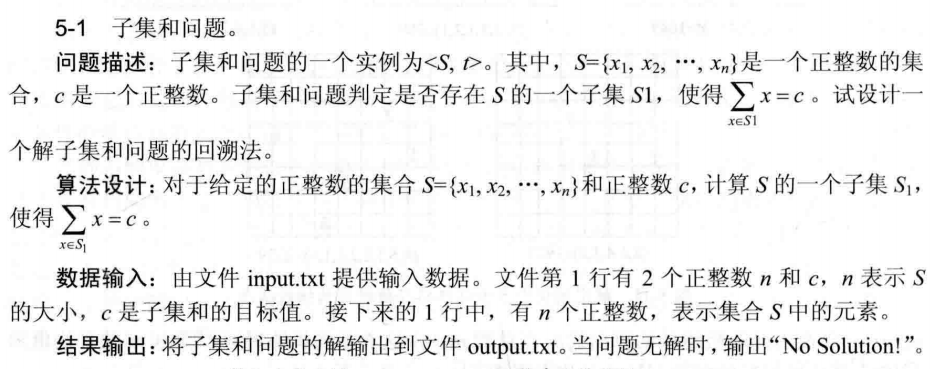
**二、实验装置**

**Win7, Visual C++/Java**

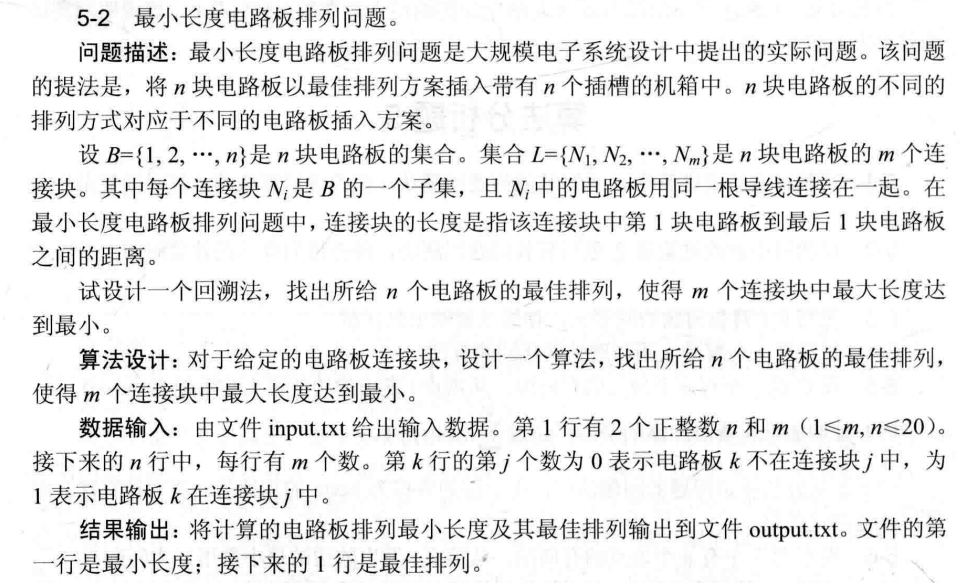
**三、实验内容**

**（在第三次实验题目库中选择题目，填写题目内容及输入输出要求）**

1.



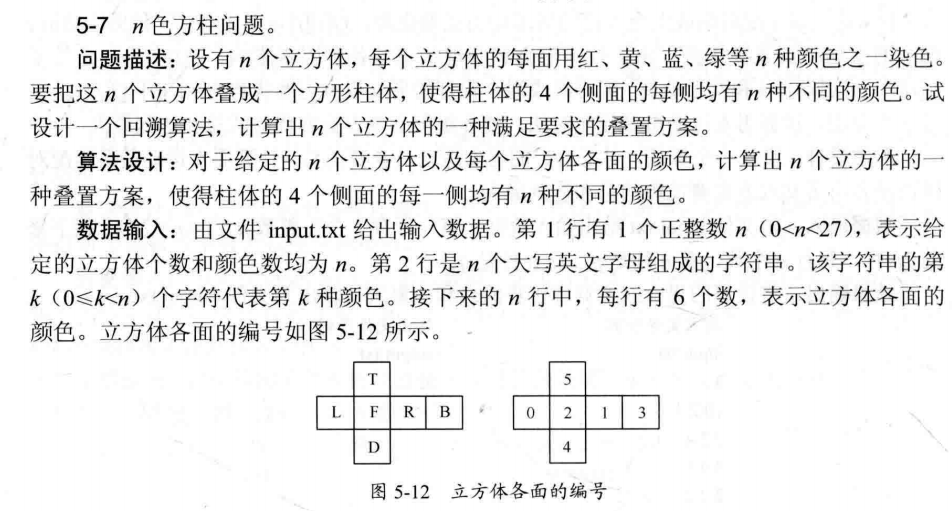
2．

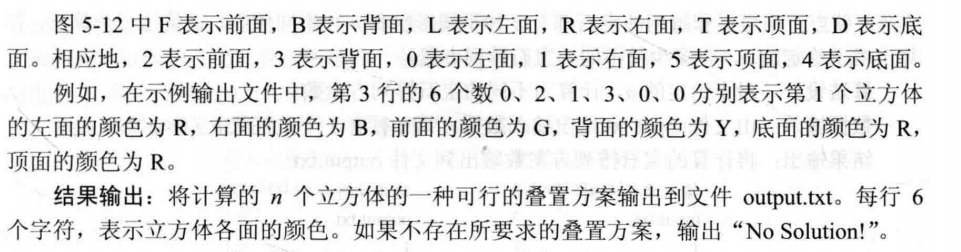


3．

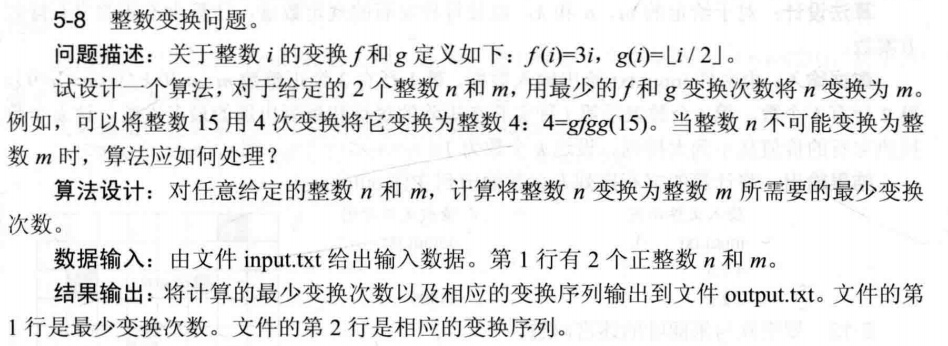


4、





5、



6、

【问题描述】

Gardon 昨天给小希布置了一道作业，即根据一张由不超过 5000 的 N(3<=N<=100)个正整数组成的数表两两相加得到 N\*(N-1)/2 个和，然后再将它们排序。例如，如果数表里含有四个数 1，3，4，9，那么正确答案是 4，5，7，10，12，13。小希做完作业以后出去玩了一阵，可是下午回家时发现原来的那张数表不见了，好在她做出的答案还在，你能帮助她根据她的答案计算出原来的数表么？

【输入形式】

包含多组数据，每组数据以一个 N 开头，接下来的一行有按照大小顺序排列的 N\*(N-1)/2 个数，是小希完成的答案。文件最后以一个 0 结束。

假设输入保证解的存在性和唯一性。

【输出形式】

对于每组数据，输出原来的数表。它们也应当是按照顺序排列的。

7、

【问题描述】

呃……变形课上Harry碰到了一点小麻烦,因为他并不像Hermione那样能够记住所有的咒语而随意的将一个棒球变成刺猬什么的,但是他发现了变形咒语的一个统一规律:如果咒语是以a开头b结尾的一个单词,那么它的作用就恰好是使A物体变成B物体.

Harry已经将他所会的所有咒语都列成了一个表,他想让你帮忙计算一下他是否能完成老师的作业,将一个B(ball)变成一个M(Mouse),你知道,如果他自己不能完成的话,他就只好向Hermione请教,并且被迫听一大堆好好学习的道理。

【输入形式】

测试数据有多组。每组有多行，每行一个单词,仅包括小写字母,是Harry所会的所有咒语.数字0表示一组输入结束.

【输出形式】

如果Harry可以完成他的作业,就输出”Yes.”,否则就输出”No.”(不要忽略了句号)

8、

在N\*N的方格棋盘放置了N个皇后，使得它们不相互攻击（即任意2个皇后不允许处在同一排，同一列，也不允许处在与棋盘边框成45角的斜线上。你的任务是，对于给定的N，求出有多少种合法的放置方法。

Input

共有若干行，每行一个正整数N≤10，表示棋盘和皇后的数量；如果N=0，表示结束。

Output

共有若干行，每行一个正整数，表示对应输入行的皇后的不同放置数量。

9、

【问题描述】

8球是一种台球竞赛的规则。台面上有7个红球、7个黄球以及一个黑球，当然还有一个白球。对于本题，我们使用如下的简化规则：红、黄两名选手轮流用白球击打各自颜色的球，如果将该颜色的7个球全部打进，则这名选手可以打黑球，如果打进则算他胜。如果在打进自己颜色的所有球之前就把黑球打进，则算输。如果选手不慎打进了对手的球，入球依然有效。

现在给出打进的球（白球除外）的顺序，以及黑球由哪方打进，你的任务是判定哪方是胜者。

假设不会有一杆同时打进一颗黑球和其他彩球。

【输入形式】

输入包含多组数据。每组数据第一行是一个整数NN(1≤N≤15)，表示打进的球的个数，N=0表示结束。随后有一行，包含N个字符，依序表示打进的是何种球。如果是B，表示是红方打进的黑球，如果是L，表示是黄方打进的黑球。如果是Y则表示是黄球，R表示红球。字符间没有空格。

所有输入都满足如下条件：最后一颗球打进时这局比赛正好结束，而且打进的红球和黑球都不超过7个。

【输出形式】

对每组数据，输出一行。如果红方胜，输出Red；黄方胜，输出Yellow。

10、

【问题描述】

A DFS(digital factorial sum) number is found by summing the factorial of every digit of a positive integer.

For example ,consider the positive integer 145 = 1!+4!+5!, so it's a DFS number.  
  
Now you should find out all the DFS numbers in the range of int( [1, 2147483647] ).  
  
There is no input for this problem. Output all the DFS numbers in increasing order. The first 2 lines of the output are shown below.

**四、程序运行结果**

**（说明设计思路，解释使用的数据结构，计算时间复杂度）**



设计思路：

输入：从标准输入中获取整数 n（集合元素数量）和整数 c（目标和），以及 n 个整数表示集合元素。

搜索算法：使用回溯法进行搜索。search 函数采用递归方法，尝试将集合中的元素依次加入子集，并根据当前子集和与目标值的大小关系进行相应的操作。

使用 vis 数组标记集合中元素的状态，subSum 变量记录当前子集的和，flag 变量标记是否找到符合条件的子集。

当子集和等于目标值时，输出符合条件的子集元素。

如果子集和小于目标值，继续添加下一个元素到子集中；如果子集和大于目标值，回溯到上一个状态并尝试其他可能的组合。

使用的数据结构：

a[] 数组存储集合元素。

vis[] 数组存储集合元素的状态，用于标记是否已经包含在子集中。

subSum 变量记录当前子集的和。

flag 变量标记是否找到符合条件的子集。

基本的整型变量用于索引、临时存储和条件判断。

时间复杂度分析：

最坏情况下，搜索所有可能的子集，时间复杂度为指数级别。每个元素有两种状态（包含或不包含），因此复杂度为 O(2^n)。



2．

设计思路：

输入：从标准输入中获取整数 n（电路板数量）和整数 m（连接块数量），以及一个二维数组 B[][] 表示电路板在连接块中的排列情况。

回溯算法：使用回溯法进行搜索。backtrack 函数通过尝试交换元素的位置，产生所有可能的排列方式，并通过 len 函数计算每种排列的最小长度。

len 函数计算当前排列的最小长度，根据电路板在连接块中的位置来确定连接块的边界，并计算每个连接块的长度。

不断更新最小长度 (bestd) 以及对应的最优排列 (bestx[])。

最后输出最小长度以及最优排列。

使用的数据结构：

x[] 数组存储当前排列的顺序。

B[][] 二维数组表示电路板在连接块中的排列。

low[] 和 high[] 数组记录每个连接块的最左边和最右边电路板的位置。

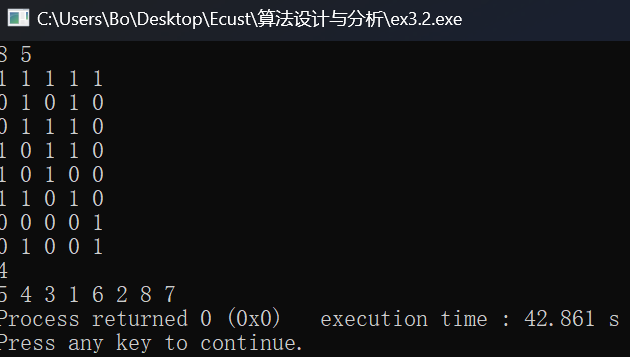
bestx[] 数组保存最优排列的顺序。

基本的整型变量用于索引、临时存储和条件判断。

时间复杂度分析：

回溯算法的时间复杂度取决于所有可能的排列数量。

对于每个元素，都有 n! 种可能的排列，其中 n 是电路板的数量。因此，最坏情况下的时间复杂度为 O(n!)



3．

设计思路：

输入：从标准输入中获取整数 n（男女运动员的数量），以及两个二维数组 boy[][] 和 girl[][]，分别表示男女双方的竞赛优势情况。

DFS搜索：使用深度优先搜索进行搜索匹配。dfs 函数通过尝试不同的匹配方式，计算男女双方的竞赛优势总和，并更新最大值 Max。

在搜索过程中，使用剪枝来优化搜索过程，当当前已计算的竞赛优势总和与剩余可能的最大值无法超过当前最大值 Max 时，进行剪枝。

对于每个男生，计算他与所有女生匹配后的竞赛优势，并记录最大的竞赛优势 maxSum。

最后输出最大的竞赛优势总和 Max。

使用的数据结构：

boy[][] 和 girl[][] 二维数组分别表示男女双方的竞赛优势情况。

res[][] 二维数组用于存放男运动员与女运动员匹配后的双方竞赛优势。

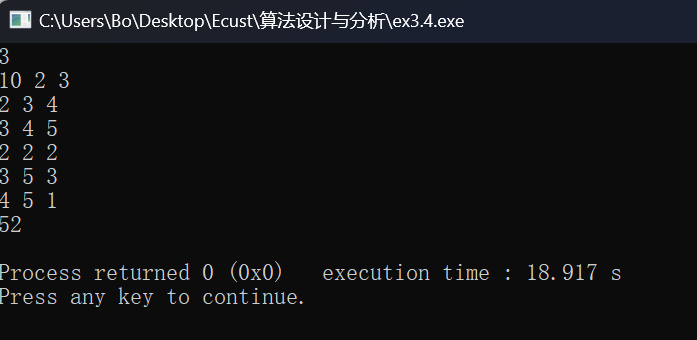
maxSum[] 数组保存每个男生匹配后可达到的最大双方竞赛优势。

book[] 数组用于标记女运动员是否已经匹配。

基本的整型变量用于索引、临时存储和条件判断。

时间复杂度分析：

该算法通过DFS搜索所有可能的匹配方式。在最坏情况下，每个男生都有 n 个选择，因此有 n^n 种可能的匹配方式。



4、

设计思路：

输入：从标准输入中获取整数 n（立方体数量）、颜色数组 color[] 和二维数组 board[][]，表示每个立方体六个面的颜色情况。

搜索算法：使用深度优先搜索（DFS）寻找解决方案。search 函数尝试构建两个子图，每个子图代表一个相邻的立方体表面颜色相同的情况。

通过回溯和剪枝的方式搜索所有可能的方案，直到找到满足条件的解或者遍历完所有可能情况。

如果找到解，则输出每个立方体的六个面的颜色，如果没有解，则输出 "No Solution!"。

使用的数据结构：

board[][] 二维数组存储了每个立方体的六个面的颜色情况。

solu[][] 二维数组用于存储解决方案中每个立方体的六个面的颜色。

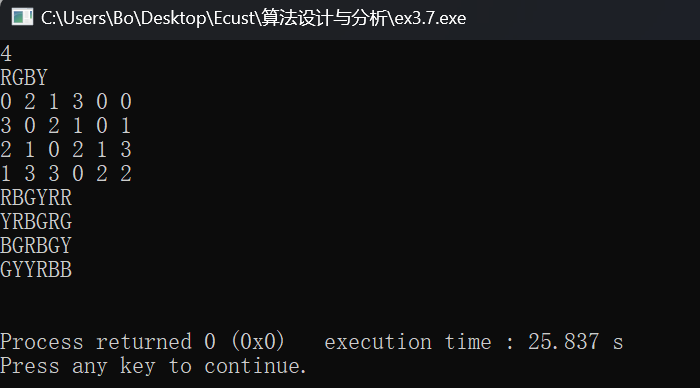
used[] 数组用于标记每个立方体的使用情况。

color[] 数组记录每个颜色的种类。

基本的整型变量用于索引、临时存储和条件判断。

时间复杂度分析：

该算法是一个暴力搜索算法，它尝试了所有可能的情况。对于每个立方体，有 6 种不同的排列方式，总共有 n 个立方体，所以搜索的时间复杂度为 O(6^n)。



5、

设计思路：

输入：从标准输入中获取两个整数 n 和 m。

DFS搜索：通过递归深度优先搜索（DFS）找到将整数 n 变换为整数 m 所需的最小操作次数，并记录变换过程。使用递归函数 dfs 进行搜索，每一层尝试两种变换操作：左子树是乘以 3，右子树是除以 2。

如果在任意一层找到满足条件的变换，记录变换过程中的操作路径，左子树记录 "f"，右子树记录 "g"。

通过不断增加搜索深度（即变换次数），直到找到变换成功的路径。

使用的数据结构：

基本的整型变量用于存储输入值、深度、变换次数以及当前操作路径。

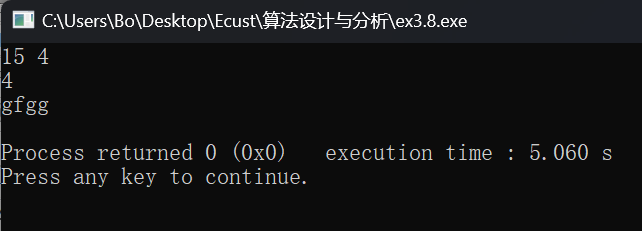
字符串 str 用于记录变换的路径，"f" 代表乘以 3 的操作，"g" 代表除以 2 的操作。

时间复杂度分析：

在每个节点，进行了两次操作：乘以 3 或除以 2。

因为搜索深度递增，最坏情况下，时间复杂度是指数级别的，取决于变换的次数。

在这种情况下，时间复杂度为 O(2^k)，其中 k 是将 n 转换为 m 的最小操作次数。



6、

设计思路：

输入处理：从输入中获取整数 n，然后将 s[1], s[2], ..., s[num]（num = n\*(n-1)/2）存储在数组 s[] 中。

尝试求解：通过尝试枚举 a[1] 和 a[2]（即 s[1] 和 s[2]），计算出 a[3] 的值，并根据 a[1] 和 a[2] 的值尝试推算出其他元素 a[3], a[4], ..., a[n] 的值。通过遍历可能的 a[1] + a[3] = s[2] 的情况，并检查是否存在唯一解。

使用 vis[] 数组标记已经使用过的元素，并尝试确定其他元素的值。

在确定 a[j]（j > 3）的过程中，确保 a[1] + a[j]，a[2] + a[j]，a[3] + a[j] 等组合在 s[] 数组中存在。

如果所有的 a[i]（1 ≤ i ≤ n）都能确定，则输出这些数值；如果无法确定满足条件的数值，则输出空行。

使用的数据结构：

整型数组 a[] 用于存储最终解的结果。

整型数组 s[] 用于存储输入的 s[i] 值。

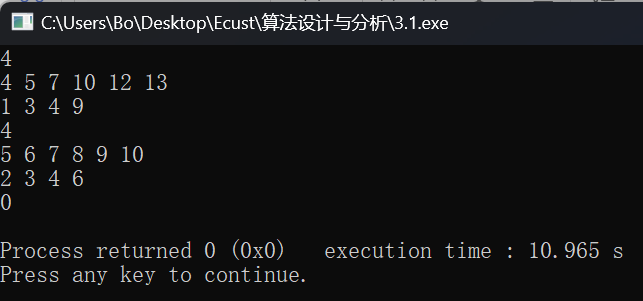
布尔型数组 vis[] 用于标记已经使用过的元素。

时间复杂度分析：

这段代码采用了暴力枚举的方法来尝试求解，因此时间复杂度较高。

主要的时间消耗在于两个循环嵌套的部分，其中外层循环尝试不同的 a[1] + a[2] 的可能组合，内层循环遍历尝试确定其他 a[j] 的值。

如果存在解，那么时间复杂度取决于成功尝试确定每个 a[j] 的过程，最坏情况下为 O(n^3)。



7、

设计思路：

输入处理：从标准输入中逐行读取字符串，表示有向边。当读取到以 '0' 开头的字符串时，表示输入结束。

DFS搜索：通过深度优先搜索，尝试从字符 'b' 出发，寻找是否存在一条路径可以达到字符 'm'。使用递归函数 dfs 进行深度优先搜索。对于每个字符串，检查其是否以当前字符开头，并且未被访问过。

如果能从当前字符出发找到字符 'm'，将 ans 设为 true。

在每次读取到 '0' 开头的字符串时，进行DFS搜索并输出结果，然后清空数据，准备接收新的输入。

使用的数据结构：

二维字符数组 c[][] 用于存储输入的字符串。

布尔数组 flag[] 用于标记每个字符串是否被访问过。

整数 cnt 用于记录输入的字符串数量。

布尔变量 ans 用于记录是否找到满足条件的路径。

时间复杂度分析：

这段代码采用了深度优先搜索的方式进行判断，每次搜索都需要遍历字符串数组。

在每次读取到 '0' 开头的字符串后，进行DFS搜索，搜索过程中需要遍历所有字符串进行匹配。

时间复杂度取决于字符串的数量和长度，在最坏情况下，时间复杂度为 O(N\*M)，其中 N 为字符串的数量，M 为字符串的最大长度。



8、

设计思路：

输入处理：从标准输入中读取一个整数 n 表示棋盘的大小，其中 n = 0 时输入结束。

深度优先搜索：使用递归函数 dfs 进行深度优先搜索，尝试在每一行中放置一个皇后，确保满足条件不被其他皇后攻击。isLeagal 函数用于检查当前位置 (row, col) 是否可以放置皇后，遍历已经放置的皇后位置，检查是否在同一列或对角线上。

在每行尝试放置皇后，当放置完成一种合法的放置方式后，递归地尝试下一行。

如果成功放置了 N 个皇后，则将 num 计数器加一，表示找到了一个符合条件的放置方法。

按照要求输出找到的合法放置方式的数量。

使用的数据结构：

二维字符数组 result[][] 用于存储棋盘状态，表示放置皇后的情况。

一维整型数组 d[] 用于存储每一行皇后放置的列数。

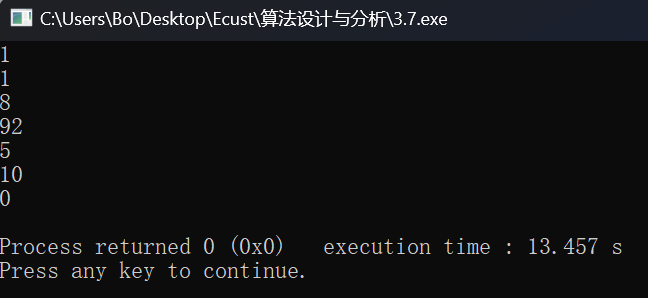
整数 num 用于记录符合条件的放置方式的数量。

时间复杂度分析：

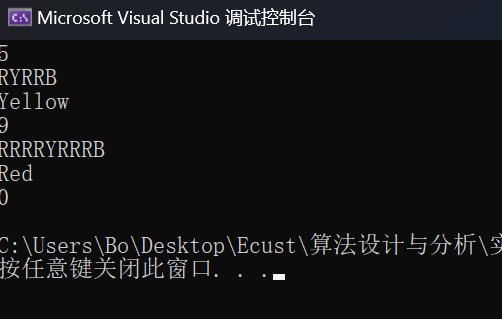
该段代码通过回溯算法解决 N 皇后问题，对于每个位置都进行了检查，所以时间复杂度较高。

主要的时间消耗在深度优先搜索的递归过程中，对每个位置都进行了合法性的检查，最坏情况下时间复杂度为 O(N^N)。

对于较大规模的 N（比如 N > 10），将会有大量的搜索空间，程序运行时间会随着 N 的增加而急剧增长。



9、



10、

设计思路：

输入处理：循环读取整数 n，表示后续输入的长度。当 n 为 0 时结束输入。

对于每个 n，接着读取 n 个字符，字符可能是 'R'（代表红色）或 'Y'（代表黄色）。

根据最后一个字符（a[n]）的值判断最终输出结果：如果最后一个字符为 'B'，则根据红色数量（'R' 的个数）判断输出结果。若红色数量等于 7，则输出 "Red"，否则输出 "Yellow"。

如果最后一个字符为 'L'，则根据黄色数量（'Y' 的个数）判断输出结果。若黄色数量等于 7，则输出 "Yellow"，否则输出 "Red"。

使用的数据结构：

int n, i, red = 0, yellow = 0;：整数变量用于读取输入和计数红色、黄色的字符个数。

char a[20];：字符数组，用于存储输入的字符。

while (1) { ... }：无限循环，直到输入的 n 为 0 时退出。

时间复杂度分析：

这段代码的时间复杂度主要取决于输入的大小 n。外层的 while 循环是根据输入来判断是否继续循环的。

内层的 for 循环遍历了每个输入的字符，并对 'R' 和 'Y' 进行了计数，因此时间复杂度为 O(n)。

整体上，时间复杂度为 O(n)。



**实验三成绩：\_\_\_\_\_\_\_\_\_\_\_\_ 教师签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**实验四 分支限界算法**

**一、实验目的**

**1、**理解分支限界法的设计思想，掌握分支限界法的算法框架和设计策略。

**2、**通过上机实验，要求掌握算法设计思想、程序设计和算法复杂性分析等。

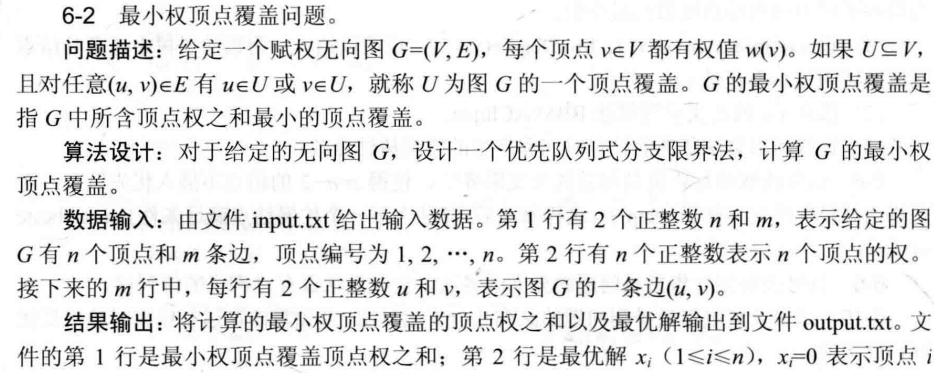
**二、实验装置**

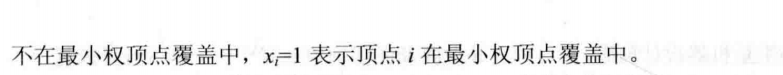
**Win7, Visual C++/Java**

**三、实验内容**

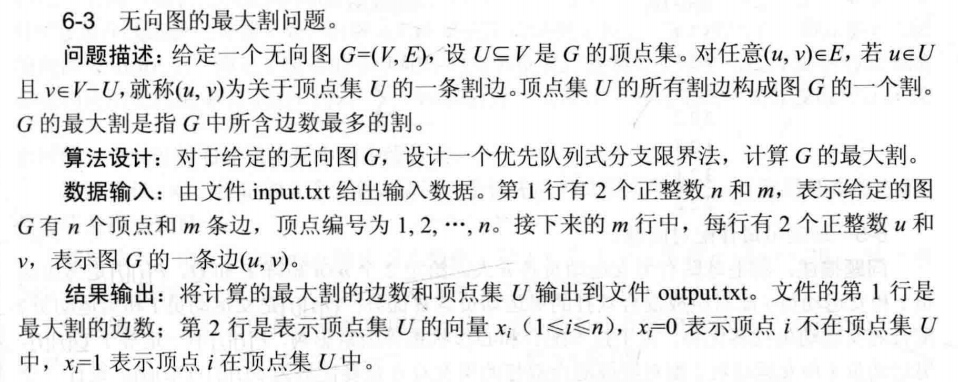
**（在第四次实验题目库中选择题目，填写题目内容及输入输出要求）**

1.

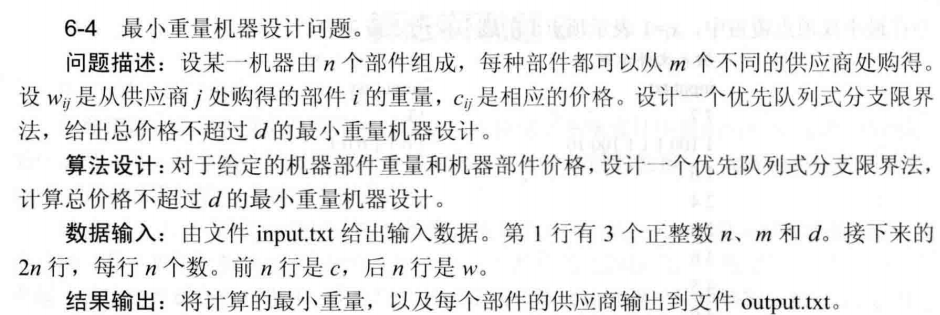




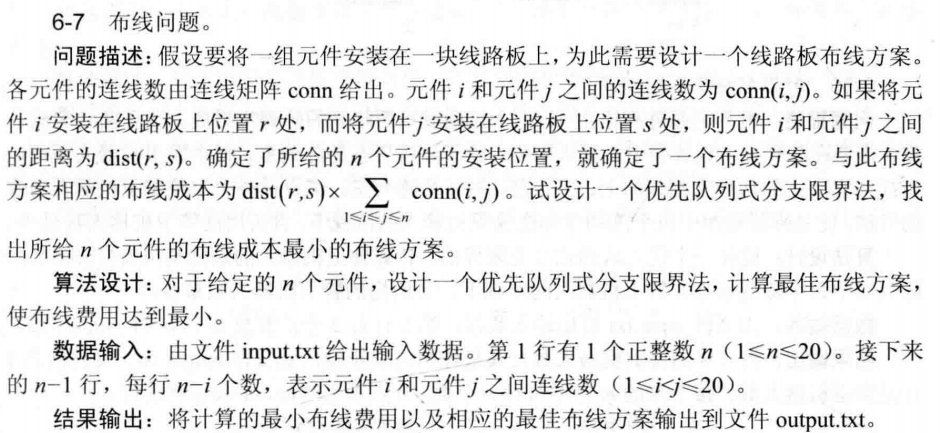
2．



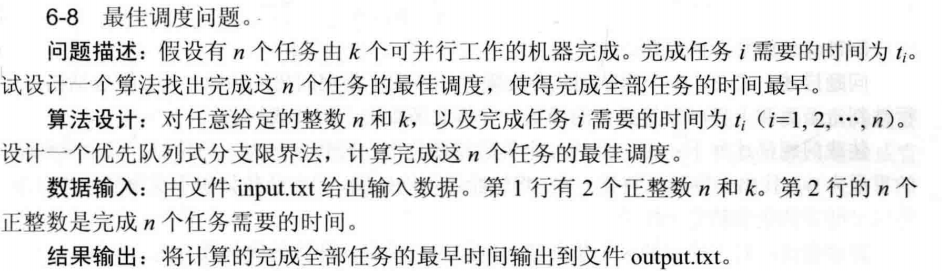
3．



4、



5、



6、

Ignatius被魔王抓走了,有一天魔王出差去了,这可是Ignatius逃亡的好机会.  
  
魔王住在一个城堡里,城堡是一个A\*B\*C的立方体,可以被表示成A个B\*C的矩阵,刚开始Ignatius被关在(0,0,0)的位置,离开城堡的门在(A-1,B-1,C-1)的位置,现在知道魔王将在T分钟后回到城堡,Ignatius每分钟能从一个坐标走到相邻的六个坐标中的其中一个.现在给你城堡的地图,请你计算出Ignatius能否在魔王回来前离开城堡(只要走到出口就算离开城堡,如果走到出口的时候魔王刚好回来也算逃亡成功),如果可以请输出需要多少分钟才能离开,如果不能则输出-1.



**Input**

输入数据的第一行是一个正整数K,表明测试数据的数量.每组测试数据的第一行是四个正整数A,B,C和T(1<=A,B,C<=50,1<=T<=1000),它们分别代表城堡的大小和魔王回来的时间.然后是A块输入数据(先是第0块,然后是第1块,第2块......),每块输入数据有B行,每行有C个正整数,代表迷宫的布局,其中0代表路,1代表墙.(如果对输入描述不清楚,可以参考Sample Input中的迷宫描述,它表示的就是上图中的迷宫)  
  
特别注意:本题的测试数据非常大,请使用scanf输入,我不能保证使用cin能不超时.在本OJ上请使用Visual C++提交.

**Output**

对于每组测试数据,如果Ignatius能够在魔王回来前离开城堡,那么请输出他最少需要多少分钟,否则输出-1.

7、

Speakless很早就想出国，现在他已经考完了所有需要的考试，准备了所有要准备的材料，于是，便需要去申请学校了。要申请国外的任何大学，你都要交纳一定的申请费用，这可是很惊人的。Speakless没有多少钱，总共只攒了n万美元。他将在m个学校中选择若干的（当然要在他的经济承受范围内）。每个学校都有不同的申请费用a（万美元），并且Speakless估计了他得到这个学校offer的可能性b。不同学校之间是否得到offer不会互相影响。“I NEED A OFFER”，他大叫一声。帮帮这个可怜的人吧，帮助他计算一下，他可以收到至少一份offer的最大概率。（如果Speakless选择了多个学校，得到任意一个学校的offer都可以）。

**Input**

输入有若干组数据，每组数据的第一行有两个正整数n,m(0<=n<=10000,0<=m<=10000)  
后面的m行，每行都有两个数据ai(整型),bi(实型)分别表示第i个学校的申请费用和可能拿到offer的概率。  
输入的最后有两个0。

**Output**

每组数据都对应一个输出，表示Speakless可能得到至少一份offer的最大概率。用百分数表示，精确到小数点后一位。

8、

大家一定觉的运动以后喝可乐是一件很惬意的事情，但是seeyou却不这么认为。因为每次当seeyou买了可乐以后，阿牛就要求和seeyou一起分享这一瓶可乐，而且一定要喝的和seeyou一样多。但seeyou的手中只有两个杯子，它们的容量分别是N 毫升和M 毫升 可乐的体积为S （S<101）毫升　(正好装满一瓶) ，它们三个之间可以相互倒可乐 (都是没有刻度的，且 S==N+M，101＞S＞0，N＞0，M＞0) 。聪明的ACMER你们说他们能平分吗？如果能请输出倒可乐的最少的次数，如果不能输出"NO"。

**Input**

三个整数 : S 可乐的体积 , N 和 M是两个杯子的容量，以"0 0 0"结束。

**Output**

如果能平分的话请输出最少要倒的次数，否则输出"NO"。

 9、

可怜的公主在一次次被魔王掳走一次次被骑士们救回来之后，而今，不幸的她再一次面临生命的考验。魔王已经发出消息说将在T时刻吃掉公主，因为他听信谣言说吃公主的肉也能长生不老。年迈的国王正是心急如焚，告招天下勇士来拯救公主。不过公主早已习以为常，她深信智勇的骑士LJ肯定能将她救出。  
现据密探所报，公主被关在一个两层的迷宫里，迷宫的入口是S（0，0，0），公主的位置用P表示，时空传输机用#表示，墙用\*表示，平地用.表示。骑士们一进入时空传输机就会被转到另一层的相对位置，但如果被转到的位置是墙的话，那骑士们就会被撞死。骑士们在一层中只能前后左右移动，每移动一格花1时刻。层间的移动只能通过时空传输机，且不需要任何时间。

**Input**

输入的第一行C表示共有C个测试数据，每个测试数据的前一行有三个整数N，M，T。 N，M迷宫的大小N\*M（1 <= N,M <=10)。T如上所意。接下去的前N\*M表示迷宫的第一层的布置情况，后N\*M表示迷宫第二层的布置情况。

**Output**

如果骑士们能够在T时刻能找到公主就输出“YES”，否则输出“NO”。

10、

一个规则的实心十二面体，它的 20个顶点标出世界著名的20个城市，你从一个城市出发经过每个城市刚好一次后回到出发的城市。

**Input**

前20行的第i行有3个数,表示与第i个城市相邻的3个城市.第20行以后每行有1个数m,m<=20,m>=1.m=0退出.

**Output**

输出从第m个城市出发经过每个城市1次又回到m的所有路线,如有多条路线,按字典序输出,每行1条路线.每行首先输出是第几条路线.然后个一个: 后列出经过的城市.参看Sample output

**四、程序运行结果**

**（说明设计思路，解释使用的数据结构，计算时间复杂度）**



设计思路：

输入 n（顶点个数）和 m（边的个数），以及每个顶点的权重 W[] 和边的信息 E[]。

创建 find\_min\_cover 函数来递归搜索最小顶点覆盖。采用回溯的方法尝试所有可能的选择。

find\_min\_cover 函数中，采用 DFS（深度优先搜索）的方式，对每个顶点有两种选择：选中或不选中。

在每个顶点的选择过程中，检查是否所有边都被覆盖，如果是则更新最小覆盖和对应的顶点选择情况。

输出最小覆盖和相应的顶点选择情况。

使用的数据结构：

int n, m;：存储顶点个数和边的个数。

int W[MAX], select[MAX], bestS[MAX];：存储顶点权重、当前选择情况和最优选择情况。

struct E：用来存储边的左右端点信息。

bool isCover(int nowSelect[])：检查当前选择的顶点是否覆盖了所有的边。

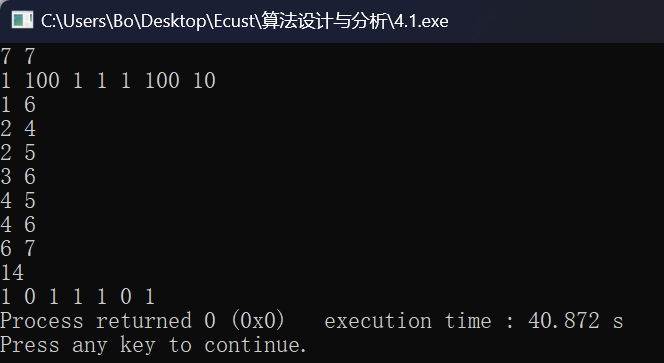
void find\_min\_cover(int nowSelect[], int position, int sum)：递归搜索最小顶点覆盖的函数。

时间复杂度分析：

递归函数 find\_min\_cover 中有一个深度优先搜索，对每个顶点有两种选择（选中或不选中）。因此，时间复杂度是指数级别的

在递归过程中，需要遍历每个顶点和每条边来判断覆盖情况，因此涉及遍历操作。

整体时间复杂度为 O(2n×(n+m))。



2．

设计思路：

读取顶点数 n 和边数 m，以及图的连接关系。

采用回溯法尝试将所有的顶点划分到两个集合中，并计算每个集合之间的最大边数。

定义 cal\_max\_cut 函数来进行回溯，对每个顶点有两种选择：放入第一个集合或放入第二个集合。

递归地进行划分，并在递归结束后计算划分的两个集合之间的边数，并更新最大边数及最优划分位置。

输出最大边数及对应的最优划分结果。

使用的数据结构：

int n, m;：存储顶点数和边数。

bool is\_connected[MAX\_NODE][MAX\_NODE];：邻接矩阵，表示图的连接关系。

int cut\_pos[MAX\_NODE];：记录最优划分的位置。

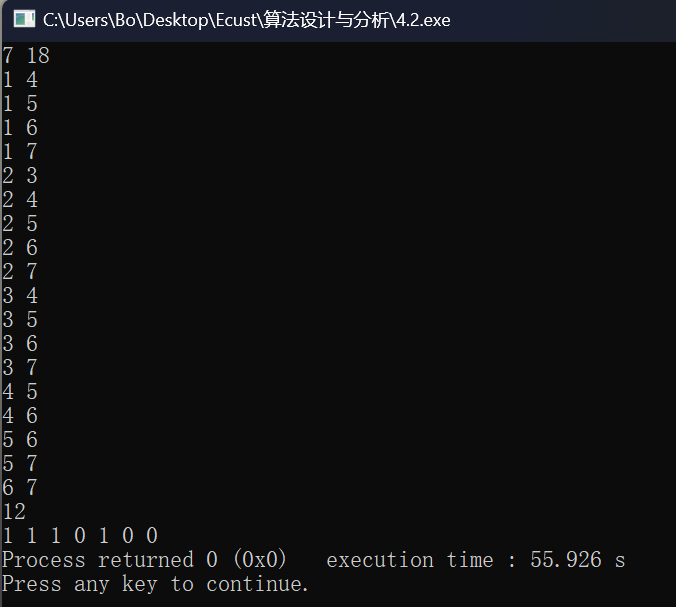
void cal\_max\_cut(int n, int m, bool is\_connected[][MAX\_NODE], int cut[], int i)：递归搜索最大边数的函数。

时间复杂度分析：

递归函数 cal\_max\_cut 中对于每个顶点都有两种选择，因此时间复杂度为 O(2n)。

在递归过程中，需要检查每个顶点的连接关系，最坏情况下需要遍历所有顶点对，因此涉及遍历操作。

整体时间复杂度为 O(2n×n2)，其中 �n 表示顶点数。



3．

设计思路：

输入部件数量 n，供应商数量 m，和价格上限 d，以及每个部件在不同供应商的价格和重量信息。

采用优先队列的分支定界法求解问题，通过优先队列（最小堆）来管理节点。

定义 Node 结构体表示每个节点的状态，包括当前已选机器的重量和价值，层次、供货商等信息。

使用 createNode 函数创建节点，并根据计算出的优先级设置节点的优先级。

实现 constraint 函数进行可行性剪枝和限界剪枝，判断节点扩展的可行性。

采用回溯法，在满足约束条件的前提下，根据价格上限 d 寻找最小重量的机器组合。

使用的数据结构：

Node 结构体：包含节点的重量、价值、供货商、层次、优先级等信息。

int\*\* c, int\*\* w：分别存储价格和重量的二维数组。

priority\_queue<Node> heap：优先队列，用于按照节点的优先级进行排序和管理节点。

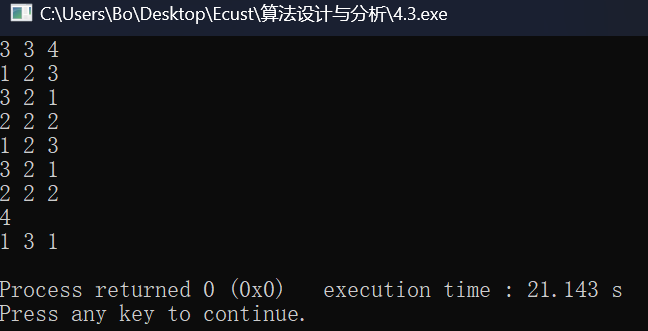
时间复杂度分析：

初始化价格和重量数组的时间复杂度为 O(n×m)，其中 n 为部件数量，m 为供货商数量。

因为是分支定界法，最坏情况下需要遍历所有可能的组合，节点的个数为 O(mn)。

在扩展节点时，根据价格上限和约束条件，进行可行性剪枝和限界剪枝，优先队列插入和删除操作的复杂度为 O(logn)。

整体算法的时间复杂度为 O(mn×logn)。



4、

数据结构：

line[maxn][maxn]: 存储城市之间的距离信息。

dp[maxs]: 存储状态压缩下的动态规划结果，表示访问每个状态（已访问城市的集合）的最小花费。

cnt[maxs]: 存储每个状态下已访问城市的距离和。

pre[maxs]: 记录状态转移路径，用于恢复最优路径。

核心思路：

使用状态压缩表示已访问的城市集合，通过二进制位来表示城市的访问情况。

动态规划的状态 dp[i] 表示已经访问了城市集合 i 的最小花费。

cnt[i] 表示访问状态 i 下已访问城市的总距离和。

通过状态转移方程 dp[i] = min(dp[i], dp[i - qpow[j]] + cnt[i - qpow[j]]) 来更新最小花费和路径。

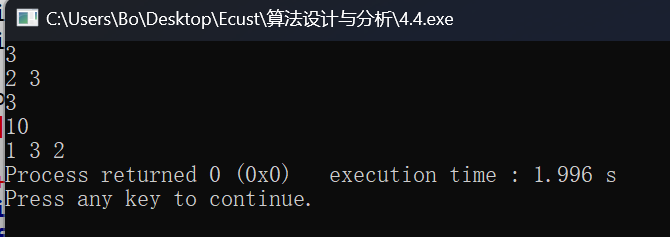
时间复杂度分析：

对于初始化和输入城市之间的距离信息，时间复杂度为 O(n^2)，其中 n 是城市数量。

cnt 数组的计算复杂度为 O(2^n \* n)，因为需要遍历所有状态并计算每个状态下的城市距离和。

动态规划的时间复杂度为 O(2^n \* n)，因为有 2^n 个状态需要更新，每个状态的更新需要检查当前状态的所有子集。

最后的路径恢复部分时间复杂度为 O(n)。



5、

数据结构：

Node 类：表示调度的节点，包含了节点所需的时间、当前时间、选择数组（记录每个任务的分配情况）和机器数组（记录每个机器上已经分配的时间）。

Schedule 类：用于执行调度算法，包括主要的调度逻辑和计算最小时间的功能。

主要逻辑：

Node 类用于表示调度问题中的节点。compute 函数用于更新节点的时间。

Schedule 类负责执行调度算法。在 ExSchedule 函数中，使用队列进行 BFS，搜索所有可能的调度情况。

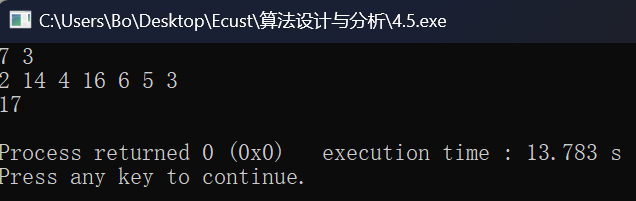
在 BFS 过程中，每个节点代表一种可能的调度情况。通过尝试为当前任务分配不同的机器，并计算时间，更新节点状态，然后将符合条件的节点加入队列进行下一轮搜索。

当节点的时间达到最小时间，并且所有任务都已分配完毕时，更新最小时间和最优解。

时间复杂度分析：

算法的时间复杂度主要取决于节点状态的搜索。

在 ExSchedule 函数中，使用 BFS 进行搜索，最坏情况下需要遍历所有可能的状态。由于每个节点有 k 种选择（机器的数量），因此时间复杂度约为 O(k^n)，其中 n 是任务数量，k 是机器数量。



6、

设计思路：

使用 BFS 来搜索三维迷宫中从起点到终点的最短路径。

通过队列来实现 BFS，在每一步中探索所有可能的移动方向，并将满足条件的节点加入队列。

针对限制最大步数，当超过步数限制时，直接返回 -1 表示无法到达终点。

在每个节点处，判断当前位置是否可行，然后更新位置并继续搜索，直到找到终点或超过最大步数。

数据结构：

struct ch：定义了表示迷宫中某个位置的结构体，包括 x、y、z 坐标和当前步数 step。

queue<ch> ha：使用 STL 中的队列来存储待探索的节点。

int map[51][51][51]：表示三维迷宫的地图。

bool biao[51][51][51]：标记数组，用于记录迷宫中的某个位置是否被访问过。

时间复杂度分析：

设迷宫的尺寸为 a \* b \* c，t 为最大步数限制。

BFS 的时间复杂度为 O(V + E)，其中 V 是顶点数，E 是边数。

在三维迷宫中，顶点数 V 为 a \* b \* c，边数 E 大致在 O(V) 级别。

因此，总体时间复杂度约为 O(V)。

最差情况下，当起点无法到达终点或者最短路径超过 t 步时，时间复杂度为 O(V)。



7、

设计思路：

使用动态规划（DP）来解决问题，其中 dp[i] 表示预算为 i 时获得至少一份录取通知的最大概率。

通过遍历每个学校的费用和录取概率，更新 dp 数组。

类似于 01 背包问题的思路，对于每个学校，从大到小遍历预算，并更新最大概率。

数据结构：

int n, m, a[maxn]：n 表示预算，m 表示学校个数，a[maxn] 存储每个学校的费用。

double b[maxn], dp[maxn]：b[maxn] 存储每个学校的录取概率，dp[maxn] 存储预算为 i 时的最大录取概率。

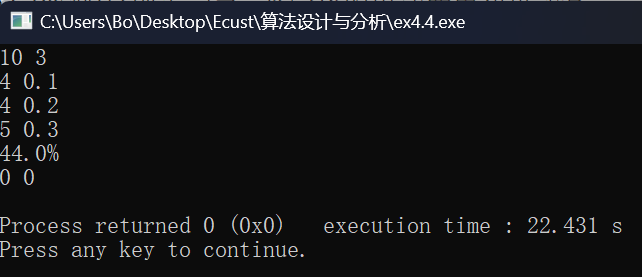
时间复杂度分析：

主要循环在两个嵌套的 for 循环中。

外部循环遍历学校数量 m，内部循环遍历预算 n。

在每次循环中，更新 dp 数组的值。

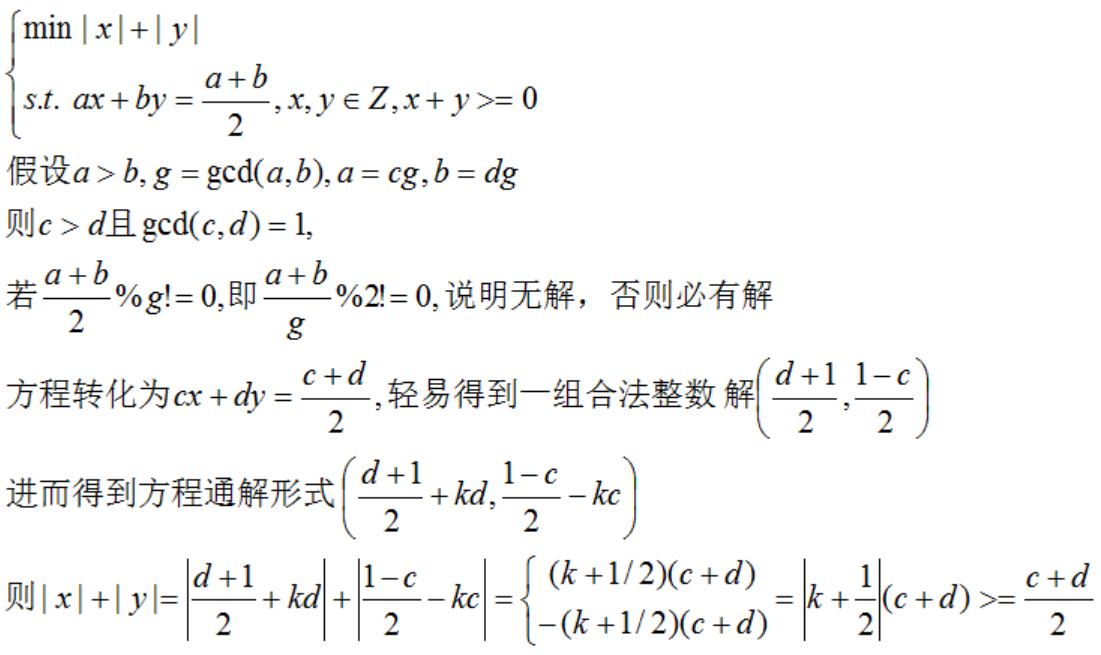
因此，总体时间复杂度约为 O(m \* n)。



8、

设计思想：

设两个小瓶子容积分别为a,b，问题转化成通过两个小瓶子的若干次倒进或倒出操作得到(a+b)/2体积的可乐，设两个小瓶子被倒进或倒出x次和y次（这里的x和y是累加后的操作，即x=第一个瓶子倒出的次数-倒进的次数，y=第二个瓶子倒出的次数-倒进的次数），那么问题转化成：



数据结构和函数：

int gcd(int a, int b)：计算两个整数的最大公约数。采用递归方式实现欧几里德算法。

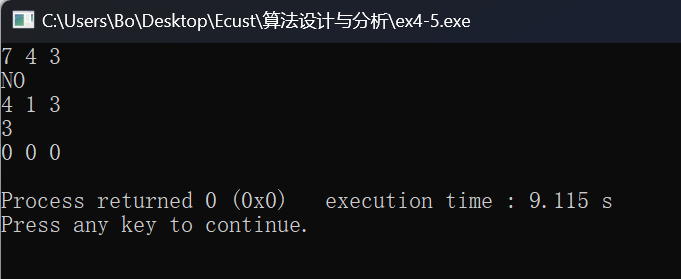
主函数中的 while 循环读取输入的三个整数，并进行处理，直到输入的三个整数之和为零时停止。

时间复杂度分析：

代码中的主要计算是求解最大公约数 gcd 函数。

时间复杂度主要由 gcd 函数决定，其时间复杂度为 O(log(min(a, b)))。

因此，整个程序的时间复杂度可以看作是 O(log(min(a, b)))。



9、

设计思路：

使用 BFS 进行路径搜索，以寻找从起点到达终点的最短路径。

每一步中，按照上、下、左、右四个方向进行搜索，并考虑传送门的情况。

通过两个状态数组 book 和字符矩阵 a 来记录位置是否访问过以及迷宫的布局。

在 BFS 中，若能找到满足条件的路径（不超过步数限制并能到达终点），则输出 "YES"；否则输出 "NO"。

数据结构和函数：

int book[2][15][15]：用于标记迷宫中位置是否访问过的状态数组。

char a[2][15][15]：存储迷宫布局信息的字符矩阵。

struct node：表示迷宫中位置的结构体，包含 x、y、z 坐标和步数 step。

使用 queue<node> Q 来实现 BFS，存储待搜索的节点。

时间复杂度分析：

主要时间消耗在 BFS 部分的搜索。

迷宫规模为 n\*m，总共有两层，因此状态空间大致为 2 \* n \* m。

BFS 的时间复杂度通常为 O(V + E)，其中 V 表示节点数量，E 表示边数量。在这个情况下，V 和 E 都大致在 O(n \* m) 的数量级。

因此，整体时间复杂度约为 O(n \* m)。



10、

设计思路：

使用 DFS 深度优先搜索算法来找到符合条件的环。

从给定的起点出发，在图中进行深度优先遍历。

当搜索到特定长度的路径时，检查该路径是否满足条件（是否存在回到起点的连接），如果满足条件，则输出这条环。

数据结构和函数：

int a[25][3]：存储图的信息，表示每个节点连接的节点。

bool b[25]：标记数组，标记节点是否被访问过。

int c[25]：存储当前路径上经过的节点。

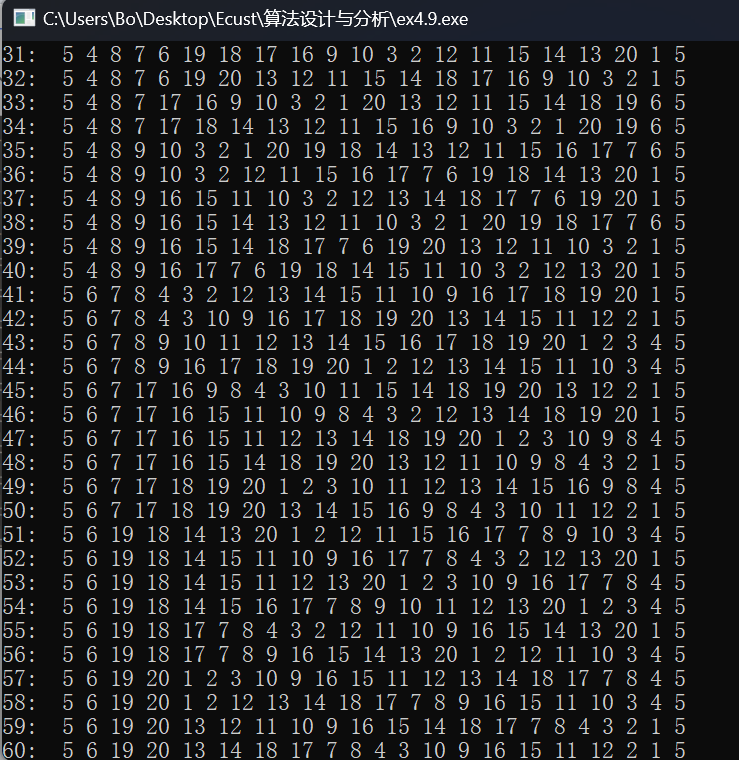
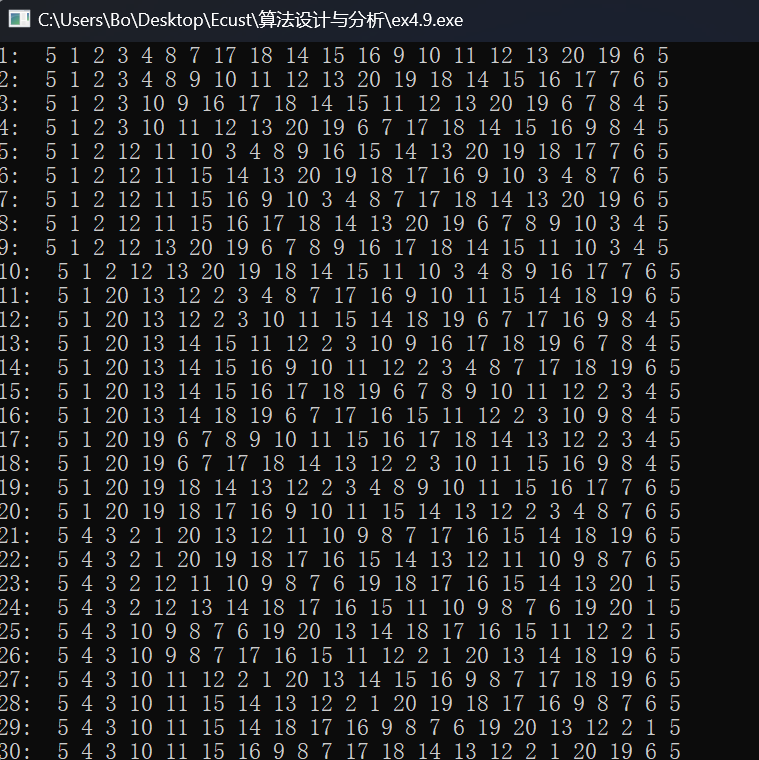
void dfs(int m, int step)：DFS 函数，用于搜索环的路径。

时间复杂度分析：

DFS 的时间复杂度主要受图的顶点数量和边数量影响。

由于每个节点最多有 3 条边（a[i][j] 中 j 的取值范围是 0 到 2），因此每个节点的搜索时间复杂度大致在 O(3^20) 左右。

在给定的循环中，尝试每个节点作为起点，使用 DFS 进行搜索。



**实验四成绩：\_\_\_\_\_\_\_\_\_\_\_\_ 教师签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**