

算法设计与分析

课程设计报告

设计题目： 3110 Problem A、3586 Problem C、
教材习题 2.14、补充题 2、补充题 7

专 业 信息与计算科学

班 级 2018 级 2 班

学 生 曾晶晶、汪璐

学 号 631810040303、631810040321

指导教师 韩逢庆

起止时间 2021/01/04~2021/01/15

设计报告	优	良	中	及格	不及格
平时表现	优	良	中	及格	不及格
程序演示	优	良	中	及格	不及格
综合成绩					

2020 年 12 月

课程设计分工表	
曾晶晶	主要负责珍妮的第一次考试、传教士与食人族和加工次序问题的算法设计、概要设计、详细设计、代码撰写及调试。
汪璐	主要负责分子集问题和 24 点游戏的算法设计、概要设计、详细设计、代码撰写及调试，课程设计报告排版，辅助完成另三个问题的详细设计。

一、问题叙述

(一) 珍妮的第一次考试

描述

第一次考试给珍妮带来了许多问题。一个问题对于任何一场考试，珍妮都需要一整天的时间来准备（好消息对于任何一场考试，她只需要一天的时间来准备）。还有一个问题：在一天的考试中，珍妮不能学习任何知识。并且这个主要的问题是：珍妮必须在第 i 场考试之前不早于 t_i 天的期间中准备考试，否则在考试时她会完全忘记所有与第 i 场考试相关的知识。珍妮想尽可能晚地开始准备，但她必须通过所有的考试。帮助珍妮选择她必须开始学习的日子。

输入

输入文件的第一行包含 $n(1 \leq n \leq 50000)$ ——考试数量。下面几行描述了考试。每个描述由三行组成。第一行是主题名称（仅包含拉丁字母的字符串，最大长度为 10）。第二行是格式为 *dd.mm.yyyy* 的考试日期。第三行包含此考试的 $t_i(1 \leq t_i \leq 100000)$ 。所有考试的处于 01.01.1900 至 31.12.2100 之间。回想一下，如果年份可以被 4 整除，而不能被 100 整除，或者可以被 400 整除，那么它就是闰年，有 366 天，多出的日期是 2 月 29 日。

输出

输出珍妮可以开始准备并通过所有考试的最晚日期。以 *dd.mm.yyyy* 格式写入日期。如果不可能通过所有考试，则输出 “Impossible” 一词。

样值输入

样值输入 #1	02.01.1900	样值输入 #2	30.06.2005
3	3	2	2
<i>Philosophy</i>	<i>Physics</i>	<i>Philosophy</i>	
01.01.1900	04.01.1900	29.06.2005	
1	10	1	
<i>Algebra</i>		<i>Algebra</i>	

样值输出

样值输出 #1

30.12.1899

样值输入 #2

Impossible

(二) 传教士与食人族

描述

传教士和食人族的问题通常是这样表述的：三名传教士和三名食人者在河的一边，还有一艘可以容纳一两个人的船。想办法把每个人都带到另一边，而不会把一群传教士留在一个人数超过食人族的地方。现在这个问题已经扩大到更复杂了。有 m 个传教士和 m 个食人族想过河。而且这艘船也被扩大到能够容纳 n 个人。为了让他们都安全过河，最少走几步？请注意，当船过河时，船上必须至少有一名传教士或食人者。

输入

输入的第一行是整数 T ，表示测试用例的数量。每个测试用例用单独的行指定，由两个正数 m 和 n 组成，其中 $m \leq 100000$ 和 $n \leq 1000$ 。

输出

对于每个测试用例，在一行上输出结果。如果问题不能解决，则输出-1 作为结果。

样值输入

2

3 2

20 3

样值输出

11

-1

(三) 加工次序问题

有 10 个工件在一台机床上加工。这 10 个工件中，有些工件必须在另一些工件加工完毕之后才能加工，这些必须先加工的工件称为前期工件。按规定，在工件运抵后 266h 内应加工完毕，否则要支付一定的赔款，赔款数正比于延误的时间，各工件每延误 1h 的赔款额不同。表 1 列出了各工件所需的加工时间、加工次序要求和每延误 1h 的赔款额。设由于机器的故障，10 个工件运抵之后 T 小时才开始加工，要安排一个加工次序，使得支付的赔款最少。

表 1: 加工次序问题表格

工件号	1	2	3	4	5	6	7	8	9	10
加工时间	20	28	25	45	16	12	60	10	20	30
紧前工件	3	8	7	/	1,2,6	8	4	3	5	9
1 小时赔款	12	14	15	10	10	11	12	8	6	7

(四) 分子集问题

设 S 是 n 个不等的正整数的集合, n 为偶数。给出一个算法将 S 划分成字迹 S_1 和 S_2 , 使得 $|S_1| = |S_2| = n/2$, 且

$$\left| \sum_{x \in S_1} x - \sum_{x \in S_2} x \right|$$

达到最大, 即使得两个子集元素之和的差达到最大。

(五) 24 点游戏

(1) 由计算机系统随机生成 4 张扑克牌, 每一张数字介于 1-10(可以重复), 用户利用这 4 个数字及运算符号 “+” “-” “*” “/” 及括号 “(” 和 “)” 从键盘上输入一个计算表达式 (每个数字仅使用一次); 系统显示用户所花的时间 (从用户看到 4 张扑克牌到计算表达式输入完毕), 并判断计算表达式结果是否等于 24, 是则显示 “*Congratulation*”, 否则显示 “*Incorrect*”。

(2) 由用户输入 4 张扑克牌, 每一张数字介于 1-10(可以重复), 计算机程序系统利用运算符号 “+”、“-”、“*”、“/” 及括号 “(” 和 “)” 以及这 4 张扑克牌的数字的各种顺序组合来计算能否得出 24, 能, 则显示表达式, 否则显示 *failed*; 系统显示计算所花的时间 (从用户输入 4 张扑克牌到显示完毕)。测试数据: (2, 2, 7, 7); (4, 4, 10, 10); (1, 2, 3, 8); (1, 2, 4, 6); (1, 5, 5, 5)

二、问题分析

(一) 珍妮的第一次考试

1. 问题要求

对于题目给出的 n 科考试及其对应的复习期限 t_i ，我们需要找到在能保证所有考试都能够在规定的时间内复习完的最晚开始复习的时间。

2. 限制条件

- (1) 任何一门考试，需要且仅需要花一整天来复习；
- (2) 考试当天不能复习；
- (3) 第 i 场考试复习的时间仅为考试前的 t_i 内，否则此门考试，他将通不过；
- (4) $1 \leq n \leq 50000, 1 \leq t_i \leq 100\,000$;
- (5) 同时所有考试的处于 01.01.1900 至 31.12.2100 之间。

(二) 传教士与食人族

1. 问题要求

判断对于题目给的测试用例 m, n 是否能让传教士与食人族全部安全渡到对岸，若能，则输出最少需要的步数，否则输出-1。

2. 限制条件

- (1) $m \geq c$ ，即任何时刻两岸、船上都必须满足传教士人数不少于野人数 ($m=0$ 时除外，既没有传教士)；
- (2) $m + c \leq k$ 船上人数限制在 k 以内。

(三) 加工次序问题

1. 问题要求

在 T 小时后，加工 10 个工件，求出一个加工次序，使得支付的赔款最少。

2. 限制条件

有些工件必须在前期工件加工完毕后才能加工。

(四) 分子集问题

1. 问题要求

针对给定的集合 S ，将其分为两个集合，使得两个集合的元素之和相差最大，返回

元素之和的差值。

2. 限制条件

- (1) 集合平均分成两份，即两个集合的元素个数相等；
- (2) 两个集合的元素之和相差最大。

(五)24 点游戏

1. 问题要求

(1) 对由计算机系统随机生成的 4 张扑克牌，用户输入表达式，系统判断表达式的值是否为 24，并显示用户所花的时间。

(2) 用户输入 4 个数字，系统计算这 4 个数字能否通过运算得到 24，如果能则返回表达式，否则显示 *failed*。

2. 限制条件

- (1) 计算机系统随机生成的 4 个数和用户输入的数字必须介于 1-10；
- (2) 生成和输入的数字可以重复，且每个数字只能使用一次；

三、概要设计

(一) 珍妮的第一次考试

1. 求解思路

为了方便计算,我先将日期映射成数字,从 1600 年 1 月 1 日,一直加,一直加直到年份不超过 2100 年。再将题目给定的考试日期运用二分法映射成数字。然后采用贪心算法进行对给定的考试科目复习的右边界进行排序,然后再采用二分法搜索树状数组的方式寻找最晚复习时间点,二分复习时间段,若中点到区间右端点全都被占用且中点没有被占用,则中点即为最优点,否则继续缩小区间来分。

2. 数据类型

(1) n 是一个整型数,表示给定的考试场数。

(2) $s[10]$ 是一个字符型数组,表示考试名称。

(3) $c[]$ 是一个整数型数组,一个用于统计 $1 \sim i$ 结点有多少天被占用树状数组。

(4) INF 是一个静态整数型,表示无穷大。

(5) $mon[]$ 是一个整数型数组,表示每个月的日期,月份下标从 1 开始,为了让 $m[0]$ 不被选上,设置了一个极大数。

(6) cnt 是一个整型数,记录最后一个日期,即 2100.12.31 在 $date[]$ 中所对应的数字,即下标。

(7) 日期结构体 $Date$, 定义 $date[100000]$ 表示每天的日期结构体数组

$ADT\ Date\{$

数据对象: $D = \{y, m, d | y, m, d \in int, time \text{ 是年}, pay \text{ 是月}, pt \text{ 是日} \}$

数据关系: $R = \{ \langle Date\ cur, Date\ rhs \rangle | cur, rhs \in Date, \text{若 } cur \text{ 的日期早于 } rhs \text{ 的日期, 则 } cur < rhs; \text{若 } cur \text{ 的日期等于 } rhs \text{ 的日期, 则 } cur = rhs; \text{若 } cur \text{ 的日期晚于 } rhs \text{ 的日期, 则 } cur > rhs \}$

基本操作:

$init()$

初始条件: $Date$ 已存在

操作结果: 设置初始日期为 1600.1.1

$get()$

初始条件: $Date$ 已存在

操作结果: 格式输入该日期

add()

初始条件: *Date* 已存在

操作结果: 在当前日期的基础上增加一天, 得到新的日期

print()

初始条件: *Date* 已存在

操作结果: 格式输出该日期

} *ADT Date*

(8) 考试科目, 定义 $p[50000]$ 记录所有给出的考试科目信息

ADT Course{

数据对象: $D = \{End, t, | End, t \in int, End \text{ 是该科目的考试时间}, t \text{ 是该科目允许复习的时长} \}$

数据关系: $R = \{ \langle Course\ a, Course\ b \rangle | a, b \in Course, \text{若 } a \text{ 复习的左边界早于 } b \text{ 复习的左边界, 则 } a < b \}$

基本操作:

sort()

初始条件: 考试科目数组已存在

操作结果: 将数组中的元素根据复习的左边界从小到大, 即从早到晚排序

} *ADT Course*

3. 模块划分及所用算法

(1)*main()*: 获取所给考试科目的信息, 并将考试科目日期转换成数字放入 $date[]$ 中, 将科目信息放入 $p[]$ 数组中, 再应用贪心算法, 对 $p[]$ 中的 n 个日期根据复习的左边界从早到晚排序, 计算得到在规定的时间内复习完的最晚开始复习的时间。

(2)*init()*: 预处理日期, 将从 1600.1.1 开始到 2100.12.31 的日期逐天存入结构体数组 $date[]$ 中, 即先将日期转换为数组

(3)*BinSc(Date &rhs)*: 二分法考试日期, 将所给的日期在 $date[]$ 数组中对应的位置输出

(4)*lowbit(int x)*: 得到非负整数 x 在二进制表示下最低位及其后面的 0 构成的数值

(5)*sum(int x)*: 得到在树状数组 $c[]$ 中第 i 个节点之前有多少场考试

(6)*add(int pos)*: 为当前日期建立树状数组, 在当前日期所对应的树状数组的节点及父节点处 +1

(7)*solve()*: 找到保证所有考试都能够在规定的时间内复习完的最晚开始复习的时间并返回, 否则输出-1。然后运用二分法, 利用树状数组先往右逐步二分当前考试科目的复习时间段, 判断是否有最佳复习时间, 若没找到, 则从左边区间逐步二分复习时间段, 还是没找到则输出-1。

4. 调用关系图

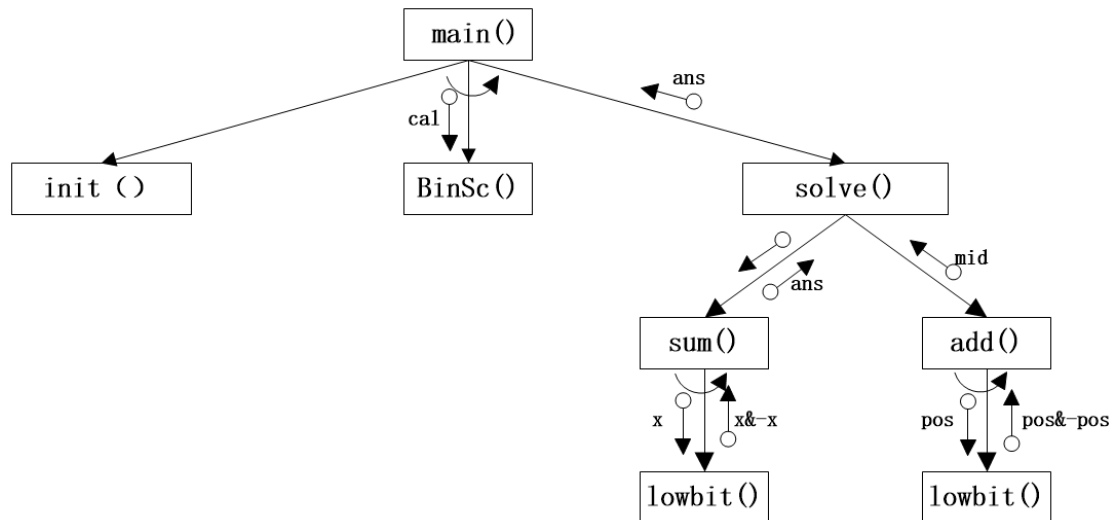


图 1: 珍妮的第一次考试模块调用关系图

(二) 传教士与食人族

1. 求解思路

(1) 状态空间: 用三元组 (m, c, b) 表示左岸状态, 其中: $0 \leq m, c \leq 3, BL \in \{0, 1\}$, 则该问题可以简化为从 $(lm, lm, 1)$ 到 $(0, 0, 0)$ 的状态转换, 同时中间状态要是安全状态, 即满足安全条件。

(2) 操作规则: 该问题主要有两种操作: 从左岸划向右岸和从右岸划向左岸, 以及每次摆渡的传教士和食人族个数。我们可以使用一个二元组 (mm, cc) 来表示每次摆渡的传教士和食人族个数。

(3) 搜索策略: 为了避免重复, 我们将搜索过的状态记录下来, 之后避开搜索这个状态。同时只对安全状态 (传教士与野人的数目相等; 传教士都在左岸; 传教士都在左岸) 进行深度优先搜索。

2. 数据类型

(1) lm 是一个整型数, 表示 lm 个传教士, lm 个食人族。

(2) lk 是一个整型数, 船的最大载客量为 lk 。

(3) (m, c, b) 是一个整数型三元组, 表示左岸状态, 其中 m 表示传教士在左岸的人

数, c 表示野人在左岸的人数, b 表示船是否在左岸, 当 $b = 1$ 时, 表示船在左岸, 当 $b = 0$ 时, 表示船在右岸。

(4) (mm, cc) 是一个整数型二元组, 表示船上的人数, 其中 m 表示传教士在船上的人数; c 表示野人在船上的人数。

(5) ans 是一个整型数, 表示能让传教士与食人族全部安全渡到对岸的最少步数。

(6) ok 是一个布尔型, 表示这个问题是否有解。

(7) 左岸状态顺序栈 *SequenStack*

ADT SequenStack{

数据对象: $D = \{(m, c, b) | m, c, b \in \text{int}; m \text{ 表示传教士在左岸的人数}; c \text{ 表示野人在左岸的人数}; b \text{ 表示船是否在左岸, 当 } b = 1 \text{ 时, 表示船在左岸, 当 } b = 0 \text{ 时, 表示船在右岸}\}$

数据关系: $R = \{<(m_1, c_1, b_1), (m_2, c_2, b_2)> | (m_2, c_2, b_2) \text{ 是 } (m_1, c_1, b_1) \text{ 的下一个左岸状态}\}$

基本操作:

Init_lbank()

操作结果: 返回一个顺序栈 *lbank* 表

*Push_SequenStack(SequenStack * S, State st)*

初始条件: 顺序栈已存在

操作结果: 将状态 *st* 压入栈中

*Pop_SequenStack(SequenStack * S)*

初始条件: 顺序栈已存在

操作结果: 将栈顶的状态弹出

}ADT SequenStack

3. 模块划分及所用算法

(1) *main()*: 对于给定的测试案例将初始状态压入顺序栈中, 然后深度优先遍历所有的可能的解, 最后输出结果

(2) *Init_rule(Rule * rule)*: 应用穷举法找出船上所有可能出现的状态

(3) *visited(SequenStack * lbank, State newst)*: 从栈底开始直到栈顶遍历, 判断是否有相同的状态, 防止陷入循环

(4) *safe(SequenStack * lbank, State newst)*: 判断该状态是否安全

(5) $Pop_SequenStack(SequenStack * S)$: 将顺序栈中栈顶元素出栈

(6) $Push_SequenStack(SequenStack * S, State st)$: 将给定状态入栈

(7) $get_result(SequenStack * lbank)$: 获得栈的长度, 即所有状态的数量

(8) $DFS(SequenStack * lbank, State st, Rule * rule)$: 利用递归深度优先遍历找出测试案例的解

(9) $Search(SequenStack * lbank, State newst, Rule * rule)$: 遍历所有船上人数组合状态寻找一个安全的状态之后入栈, 若在入栈后未找到解则将栈顶元素出栈。

(10) $Init_lbank()$: 初始化 $lbank$ 表, 用来存放状态空间

(11) $init_State(int m, int k)$: 初始化状态 $(lm, lm, 1)$

4. 调用关系图

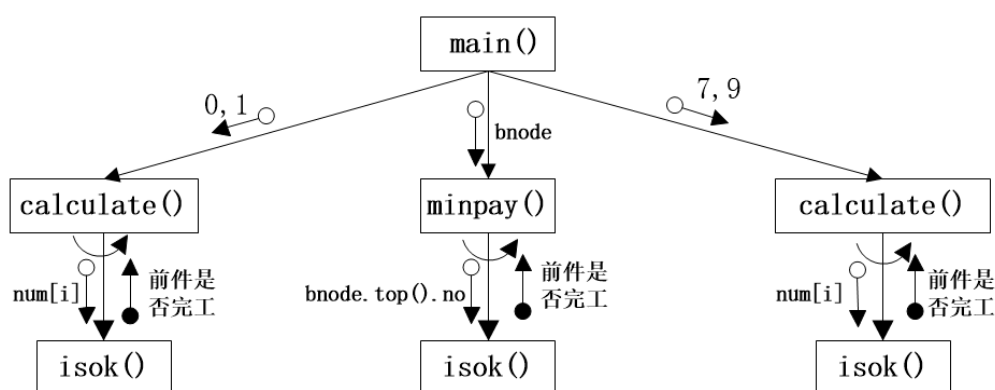


图 2: 传教士与食人族模块调用关系图

(三) 加工次序问题

1. 建立模型

(1) 网络图

用圆圈表示加工一个工件, 用箭头表示加工的次序要求, 即箭头指向的圆圈内的工件必须在箭尾圆圈内的工件加工完毕之后方能开始加工. 据表 1 可画出网络图如图 3 所示, 其中圆圈内的数字表示加工的工件号, 圆圈外的数字表示该工件所需的加工时间.

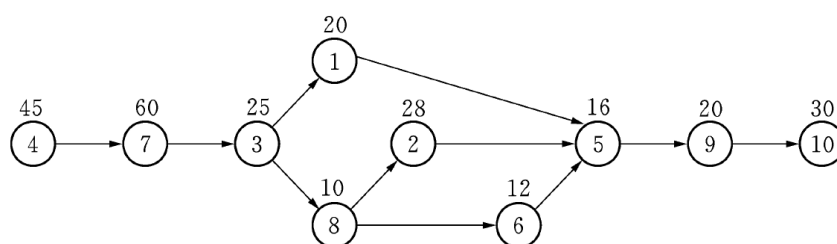


图 3: 加工次序网络图

(2) 优化的目标

假设两个工件加工之间, 机床做准备工作的时间可以忽略. S_1, S_2, \dots, S_{10} 是 $1, 2, \dots, 10$ 的一个排列, 满足由图 5-5 中的网络描述的加工次序约束, 称为可行的加工次序。由于开工延误了 T h, 第 s_j 个工件的完工时间为

$$\sum_{i=1}^j t_{s_i} + T$$

其中 $t_l (l = 1, 2, \dots, 10)$ 表示第 l 个工件的加工时间. 令 w_j 为第 j 个工件延误 1 h 的赔款数, 那么赔款总额为

$$P = \sum_{j=1}^{10} \max \left(\sum_{i=1}^j t_{s_i} + T - 266, 0 \right) w_{s_j}$$

2. 模型的求解 (贪心算法)

从图 5-5 可以看出, 工件 4,7,3 和 5,9,10 的加工次序是确定的, 所以, 我们实际上只需确定工件 1,2,6,8 的加工次序。我采用贪心算法来求解这个极小化问题, 将 3, 1, 2, 6, 8 这四个节点的单位时间赔偿的金额 $pt_i = p_i/t_i$ 从大到小排序, 得到一个优先级队列, 然后在加工次序约束的条件下一次进行加工得到的次序即为最佳次序。

3. 数据类型

(1) $num[]$ 一个静态整数型数组, 表示根据图 3得到的一个前 2 后 3 有序的, 但是中间 5 个无序的工号。

(2) $tt[]$ 是一个静态整数型数组, 表示工号对应的所需加工时间。

(3) $pp[]$ 是一个静态整数型数组, 表示工号对应的 1 小时赔款。

(4) $before[] [3]$ 是一个二位整数型数组, $before[i]$ 表示第 $i+1$ 个工件的前件, 若 $before[i][1] == 0$, 表示该工件不需要前件。

(5) $ready[10]$ 是一个布尔型数组, $ready[i]$ 表示第 i 个工件是否完成, 其初始值是 $false$ 。

(6) $order[10]$ 是一个整数型数组, 用来记录加工次序, 初始值为 0。

(7) T 是一个整型数, 表示给定的延误时间。

(8) $DeadLine = 266$ 是一个整型数, 表示在工件运抵后必须完成的时间。

(9) sum 是一个整型数, 表示赔款总额。

(10) j 是一个整型数, 当工件完成时将工件号填入 $order[j]$ 。

(11) $work$ 是一个整型数, 表示 $\sum_{i=1}^j t_{s_i} + T - 266$, 当其为正时, 表示所超出的时

间。

(12) 工件号 3 ~ 5(不包括 5) 的节点

ADT BiTNode{

数据对象: $D = \{time, pay, pt, no | time, pay, no \in int, pt \in float, time \text{ 是加工时间, } pay \text{ 是每小时赔款, } pt \text{ 是每小时赔款加工时间比, } no \text{ 是工件号} \}$

数据关系: $R = \{ \langle time, pay, pt, no \rangle | time, pay, pt, no \in D, pt = pay \times 0.1 / time \}$
 $\{(BiTNode\ a, BiTNode\ b) | \text{若 } a.pt < b.pt \text{ 则 } a < b\}$

基本操作:

init(int t, int p, int n)

初始条件: 二叉树节点 *BiTNode* 已存在

操作结果: 令 *BiTNode* 中的 $time = t, pay = p, pt = p * 0.1 / t, no = n$

}ADT BiTNode

(13) 优先队列

ADT priority_queue{

数据对象: $D = \{BiTNode\}$

数据关系: $R = \{ \langle a, b \rangle | a, b \in BiTNode, a, b \text{ 按照从 } a.pt, b.pt \text{ 从大到小排序} \}$

基本操作:

priority_queue < BiTNode > bnode

操作结果: 产生一个空的名为 *bnode* 的优先级队列

InitPriQueue()

初始条件: 优先级队列已存在

操作结果: 将五个工件所形成的节点加入队列中

minpay()

初始条件: 优先级队列已存在

操作结果: 根据优先级以及在加工次序约束的条件下获得中间五个所需要的最小的赔款。

push(BiTNode b)

初始条件: 优先级队列已存在

操作结果: 将元素 *b* 按照其 *pt* 值放入队列中恰当的位置

top()

初始条件：优先级队列已存在

操作结果：返回优先级队列队头元素

pop()

初始条件：优先级队列已存在

操作结果：将优先级队列队头元素弹出

empty()

初始条件：优先级队列已存在

操作结果：判断该优先队列是否为空，为空则返回 *true*，否则返回 *false*

}ADT *priority_queue*

4. 模块划分

(1) *main()*：输出加工次序以及最小赔款

(2) *isok(int n)*：判断号码为 *n* 的工件是否已经完成了，完成了则返回 *true*，否则返回 *false*

(3) *calculate(int be, int en)*：得到顺序工作的起始工件和结束工件所需要的赔款 (*be, en* 分别为其在 *num[]* 数组中的下标)

(4) *InitPriQueue()*：得到一个包含中间五个工件所形成的节点优先队列

(5) *minpay()*：得到中间五个所需要的最小的赔款

5. 调用关系图

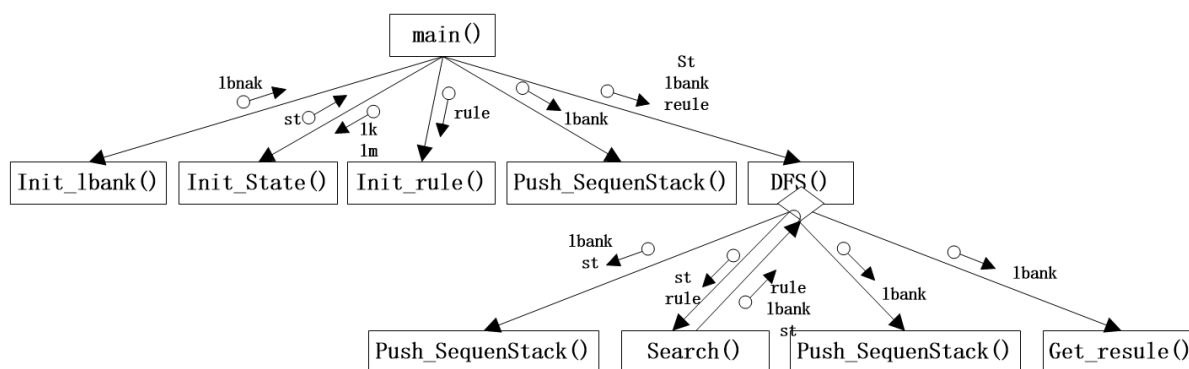


图 4: 加工次数问题模块调用关系图

(四) 分子集问题

1. 抽象数据类型

ADT S{

数据对象: $D = \{S[], S_1[], S_2[] \mid S[], S_1[], S_2[] \text{ 均为为整型数组} \}$

数据关系: $R1 = \{ \langle S[], S_1[], S_2[] \rangle \mid S[] \text{ 为整数集合}, S_1[], S_2[] \text{ 为数组 } S \text{ 的子集} \}$

基本操作:

$sort(int *a, int length)$

操作结果: 将数组 $a[]$ 进行由小到大的排序

$Deliever(int *a, int length)$

操作结果: 将数组 $a[]$ 分为两组

}ADT S

2. 主要模块的算法

由于题目中的集合是用户自己定义的, 因此将集合的数定义为数组进行排序更为方便, 在排序的时候, 考虑到算法的空间复杂度和稳定性, 因此 $sort(int *a, int length)$ 模块采用了穷举法中的冒泡排序法。其优点是: 简单, 空间复杂度较低, 是稳定的。

3. 调用关系图

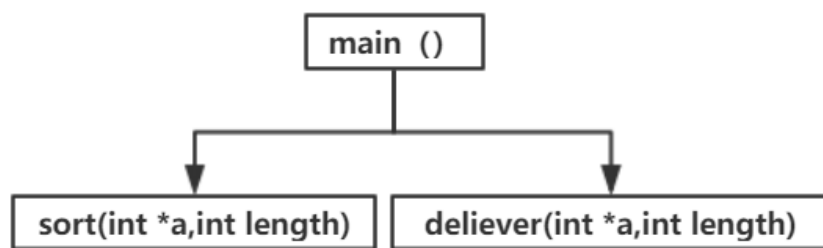


图 5: 分子集问题模块调用关系图

(五)24 点游戏

1. 抽象数据类型

ADT S{

数据对象: $D = \{n_i, o_j \mid n_i \in [1, 10], o_j \in [+,-,\times,\div], i = 1, 2, 3, 4, j = 1, 2, 3\}$

数据关系: $R1 = \{ \langle n_i, o_i \rangle \mid a_{i-1}, a_i \in [1, 10], o_i \in [+,-,\times,\div] i = 1, 2, 3 \}$

基本操作:

$op_24()$

操作结果: 判断用户输入的表达式是否等于 24.

$find_24(n_1, n_2, n_3, n_4)$

操作结果: 判断用户输入的 4 个数字是否存在值为 24 的表达式

}ADT S

2. 主要模块的算法

由于用户输入的 4 个数字能进行带有括号的运算, 因此能够对数字和运算符以及括号进行多种排列组合, 为了保证所有公式都能验证, 并且更方便捕获表达式中分母为 0 的异常, 故 $find_24(n_1, n_2, n_3, n_4)$ 采用了穷举法, 更方便理解。

3. 调用关系图

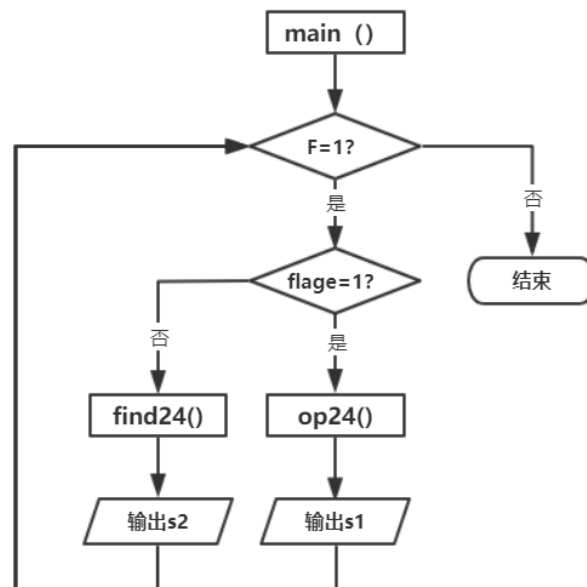


图 6: 分子集问题模块调用关系图

四、详细设计

(一) 珍妮的第一次考试

1. 存储结构

600.1.1 ~ 2100.12.31 之间的日期转换成从 1600.1.1 开始的天数顺序存储在结构体数组 $date[]$ 中。

每个考试科目的考试时间及复习期限按照给出的顺序存储在结构体数组 $p[]$ 中。

同时为了减少时间复杂度和空间复杂度，将考试当天占用的日期和复习所占用的日期在树状数组所对应的下标赋 1，故树状数组可用于统计 $1 \sim i$ 结点有多少天被占用了，而不需要一个日期一个日期的判断是否被占用。

2. 模块的伪代码

```
1      /*BinSc(Date& rhs)*/
2      while l<r do
3          if data[mid]==rhs then min<-(l+r)/2
4          else if rsh<data[mid] then r<-mid-1
5          else l<-mid+1
6      end
```

```
1      /*add(int pos)*/
2      while pos<cnt do
3          c[pos]++
4          pos<-pos+lowbit(pos)
5      end
```

```
1      /*solve()*/
2      for i=n-1 to i!=1 step-1
3          ok<-0
4          while l<r do
5              if (sum(r) - sum(mid) == r - mid)
6                  if (sum(mid) - sum(mid-1) == 0) then add(mid) ok<-1
7                  else r=mid-1
8              else l<-mid+1
9          end
10         if (!ok) then return -1;
```

(二) 传教士与食人族

1. 存储结构

左岸状态以一个三元组的形式顺序存储在栈中。新的状态入栈，此时状态无解时出栈。船上人数是一个二元组，顺序存储在数组中，可供数组下标调用。

2. 模块的伪代码

```
1      /*Init_rule(Rule *rule)*/
2      for i=0 to lk
3          for j=0 to lk-j
4              if i+j==0 do rule[k].get(i,j)
5                  end if
6          end
7      end
```

```
1      /*visited(SequenStack *lbank,State newst)*/
2      if lbank->top!=-1 do
3          for i=0 to lbank->top
4              if newst.m==lbank->data[i].n&&newst.c==lbank->data[i].c
5                  &&newst.b==lbank->data[i].b do return 1
6              endif
7          end
8      endif
```

```
1      /*safe(SequenStack *lbank,State newst)*/
2      if i==0 do
3          if newst.m>0&&newst.m<=lm&&newst.c>0&&newst.c<=lm do
4              if newst.m==0||newst.m==lm||newst.m==newst.c do
5                  return true
6              endif
7          endif
8      endif
```

```
1      /*Search(SequenStack *lbank,State newst,Rule *rule)*/
2      if st.b==1 do
3          for i=0 to Max(lk)
4              st.m=newst.m-rule[i].mm
5              st.c<-newst.c-rule[i].cc
6              st.b<-0
7              flag<-safe
8              if flag==true do
9                  Push_SequenStack(lbank,st)
10                 DFS(lbank,st,rule)
11                 Pop_SequenStack(lbank)
12             end if
13         end
14     else
15         for i=0 to i<Max(lk)
16             st.m<-newst.m+rule[i].mm
17             st.c<-newst.c+rule[i].cc
18             st.b<-1
19             flag<-safe(lbank,st)
20             if flag==true do
21                 Push_SequenStack(lbank,st)
22                 DFS(lbank,st,rule)
23                 Pop_SequenStack(lbank)
24             end if
25         end
26     en if
```

(三) 加工次序问题

1. 存储结构

将所有工件的相关信息分别顺序存储在对应的数组中。对于中间五个工件，考虑到需要将其按照每小时赔款加工时间比进行从大到小排序，采用的是大顶堆的非线性存储结构。

2. 模块的伪代码

```
1      /*isok(int n)*/
2      bool ok=true
3      if before[n-1][0]-1==-1
4          return true
5      for i=0 to 3
6          if before[n-1][i]==0
7              break
8          if ready[before[n-1][i]-1]==false
9              ok<-false
```

```
1      /*calculate(int be,int en)*/
2      for i=be to en
3          if (isok(num[i])==true)
4              work=t[i]+work
5              if work>0
6                  sum=sum+work+pp[i]
7          else
8              break
```

```
1      /*minpay(priority_queue<BiTNode>&bnode)*/
2      for i=2 to 7
3          BiTNode bbi;
4          bbi.init(tt[i],pp[i],num[i]);
5          bnode.push(bbi);
6      priority_queue<BiTNode>temp;
7      while(!empty)
8          if (all down)
9              bnode.pop();
10             while(!temp.empty)
11                 temp.pop()
```

(四) 分子集问题

1. 存储结构

由于该题的主要对象是集合，因此将集合定义为数组，是在物理内存里分配出的一块连续空间按顺序存储，故定义了线性表中的顺序存储结构。

2. 模块的伪代码

```

1      /*sort(int *a,int length)*/
2      for i <- 1 to length step 1
3          for j <- i+1 to 0 step -1
4              if a[j] < a[j - 1]
5                  swap (a[j],a[j - 1])
6              end if
7          end
8      end

```

```

1      /*int deliever(int *a,int length)*/
2      for i <0 to length/2
3          s_1[i]=a[i];
4          s_2[i]=a[i+(length-2)/2+1];
5      end
6      for i <0 to length/2
7          sum1=s_1[i];
8          sum2=s_2[i];
9      end

```

(五)24 点游戏

1. 存储结构

由于该题的主要对象是数字集合、符号集合和表达式，因此将各个集合排列组合放入 *list*，是在物理内存里分配出的一块连续空间按顺序存储，故定义了线性表中的顺序存储结构。

2. 模块的伪代码

```

1      /*op_24()*/
2      for i = 1 to 4
3          a[i]=random(1,10)
4          print(a[i])
5      end
6      game=eval(input())
7      if game==24 then return Congratulation
8      else return Incoorect

```

```

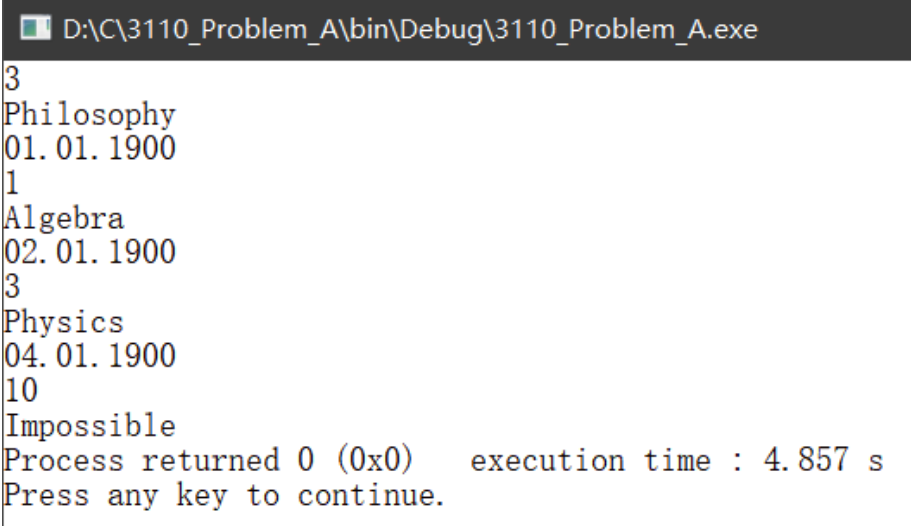
1      /*find_24(n1,n2,n3,n4)*/
2      for a,b,c,d in permutations((n1,n2,n3,n4),4):
3          for o1,o2,o3 in product(['+', '-', '*', '/'], repeat=3):
4              cases = list()
5              cases.append()
6              for expression in cases:
7                  flag=0
8                  try:
9                      if eval(expression) == 24 then
10                         print(expression)
11                         flag=1
12                         return expression
13                     except:
14                         pass
15             if (flag==0) then return failed

```

五、调试分析

(一) 珍妮的第一次考试

1. 调试原因：输出错误，没有得到正确的输出



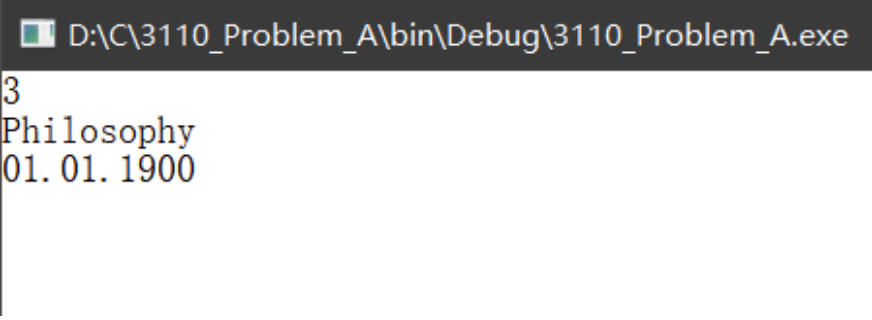
```
D:\C\3110_Problem_A\bin\Debug\3110_Problem_A.exe
3
Philosophy
01.01.1900
1
Algebra
02.01.1900
3
Physics
04.01.1900
10
Impossible
Process returned 0 (0x0)    execution time : 4.857 s
Press any key to continue.
```

图 7: 珍妮的第一次考试第 1 个错误输出图

分析原因: ans 的可能取值是从 0 开始的, 所以当 ans 为 0 时, 也会输出 *Impossible*

修改代码: 将 $!ans$ 改为 $ans \neq -1$

2. 调试原因：输入错误，没有输入完全就不运行了



```
D:\C\3110_Problem_A\bin\Debug\3110_Problem_A.exe
3
Philosophy
01.01.1900
```

图 8: 珍妮的第一次考试第 2 个错误输出图

分析原因: 在将日期转换为数组下标的过程中不能达到完成的条件

修改代码: 将 $l < r$ 改为 $l \leq r$

3. 调试原因：输入后，不能输出结果

```
D:\C\3110_Problem_A\bin\Debug\3110_Problem_A.exe
3
Philosophy
01. 01. 1900
1
Algebra
02. 01. 1900
3
Physics
04. 01. 1900
```

图 9: 珍妮的第一次考试第 3 个错误输出图

分析原因: 中点的位置取多了一位

修改代码: 将 $mid = (l + r) / 2 + 1$ 改为 $mid = (l + r) / 2$

4. 调试原因: 输入后, 不正常退出程序

```
D:\C\3110_Problem_A\bin\Debug\3110_Problem_A.exe
2
Philosophy
29. 06. 2005
1
Algebra
30. 06. 2005
2
```

图 10: 珍妮的第一次考试第 4 个错误输出图

分析原因: 在计算给定节点之前有多少场考试中, 其父节点位置找错了

修改代码: 将 $sum()$ 函数中的 $x += lowbit(x)$; 改为 $x -= lowbit(x)$;

(二) 传教士与食人族

1. 调试原因: 没有输出

```
State init_State(int m, int k)
{
    State st;
    st.m=1m; //左岸3位传教士
    st.c=1m; //左岸3位野人
    st.b=1k; //船在左岸
    return st;
}

//初始化1bank表
```

图 11: 传教士与食人族第 1 个错误程序图

分析原因：在初始化状态时，三元组 (m,c,b) 中的 *b* 被初始化为船的限载人数

修改代码：将 *lk* 改为 1

2. 调试原因：没有输出

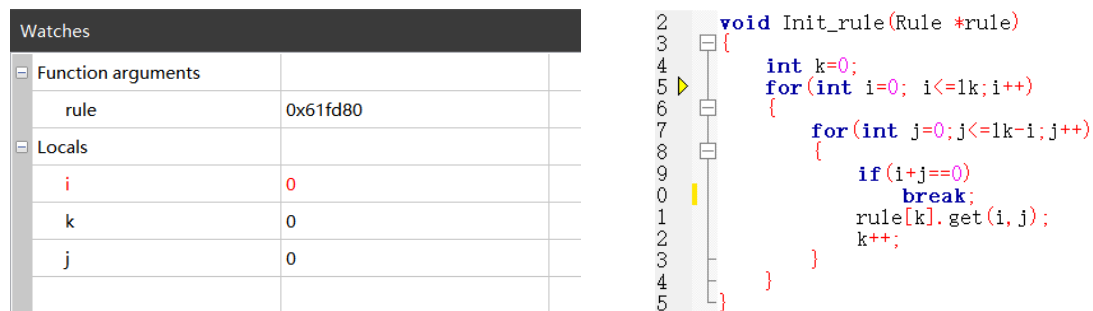


图 12: 传教士与食人族第 2 个错误调试图

分析原因：在遍历船上所有可能出现的状态时，船上传教士人数为 0 时的情况被跳过了

修改代码：将 *break* 改为 *continue*

3. 调试原因：结果输出错误，应为 11

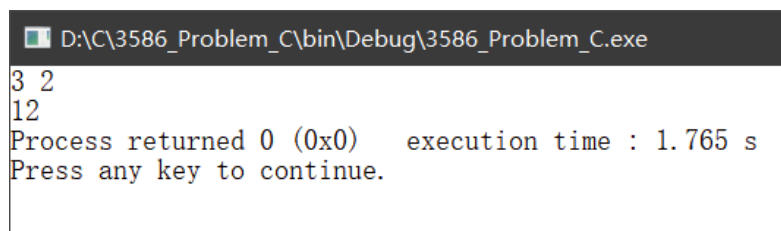


图 13: 传教士与食人族第 3 个错误输出图

分析原因：将顺序栈从栈底遍历到栈顶累加得到的数为正确解的所有状态和，但是初始状态不算在步数中间

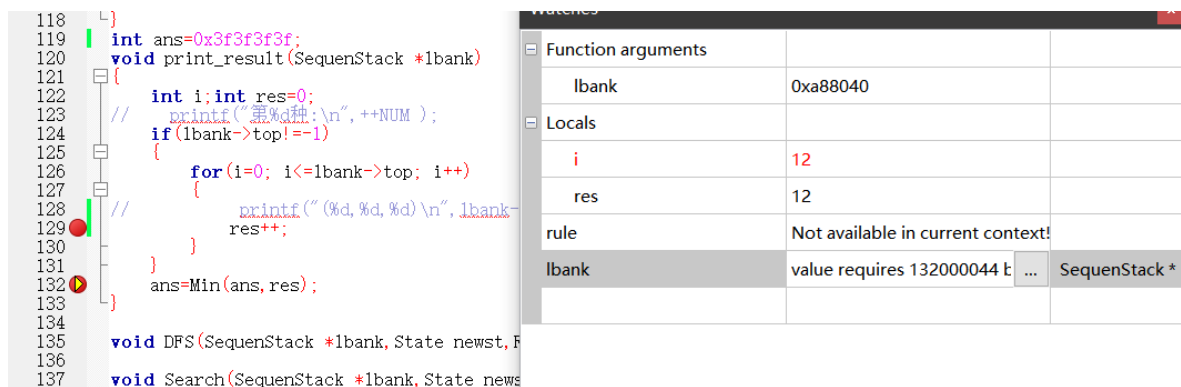


图 14: 传教士与食人族第 3 个错误调试图

修改代码：将遍历顺序栈修改为直接访问栈顶 *top* 的大小

4. 调试原因：结果输出错误，应为-1

```
D:\C\3586_Problem_C\bin\Debug\3586_Problem_C.exe
20 3
1061109566
Process returned 0 (0x0)    execution time : 4.485 s
Press any key to continue.
```

图 15: 图片名传教士与食人族第 4 个错误输出图

分析原因：没有设置未找到解时的正确输入，故输出了 *ans* 的初始值

修改代码：添加状态变量 *ok*，当有解的时候赋值 *false*，输出 *ans*，否则输出-1

(三) 加工次序问题

1. 调试原因：*error : nomatchingfunctionforcallto'Artifact::Artifact()'*

```
1
2 typedef struct Artifact //以链队列的结构定义各个工件
3 {
4     int time;//加工时间
5     int pay;//每小时赔款
6     int be[3]={0};
7     int no;//工件号
8     Artifact(int t,int p,int n):time(t),pay(p),no(n) {}; //构造方法
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Log: Build: Release 在 Bu2 中 (编译器: GNU GCC Compiler) ===
warning: 'typedef' was ignored in this declaration
In function 'void InitArt(SqQueue&, int)':
error: no matching function for call to 'Artifact::Artifact()'
note: candidate: 'Artifact::Artifact(int, int, int)'
note: candidate: 'Artifact::Artifact()' [no constructor defined]

图 16: 加工次序问题第 1 个错误提示图

分析原因：在结构体 *Artifact* 中，没有写无参的构造函数

修改代码：将构造函数改写成为自定义的初始化函数

2. 调试原因：*error : expectedinitializerbefore'.' token*

```
114 for(int i=0;i<2;i++)
115 {
116     Artifact aa.init(tt[i],pp[i],num[i];
117     EnQueue(a,aa,2);
118 }
119
```

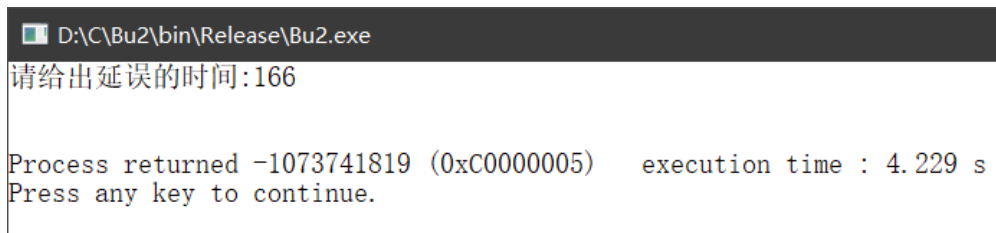
Log: Build: Release 在 Bu2 中 (编译器: GNU GCC Compiler) ===
warning: 'typedef' was ignored in this declaration
In function 'void InitArt(SqQueue&, int)':
warning: converting to non-pointer type 'int' from NULL [-Wconversion-null]
warning: unused variable 'art' [-Wunused-variable]
In function 'void Init(SqQueue&, BiNode&, SqQueue&):'
error: expected initializer before '.' token
error: 'aa' was not declared in this scope

图 17: 加工次序问题第 2 个错误提示图

分析原因：必须先定义结构体变量后才能使用结构体中的函数

修改代码：先定义，然后使用

3. 调试原因：输入后，程序不正常返回



```
D:\C\Bu2\bin\Release\Bu2.exe
请给出延误的时间:166

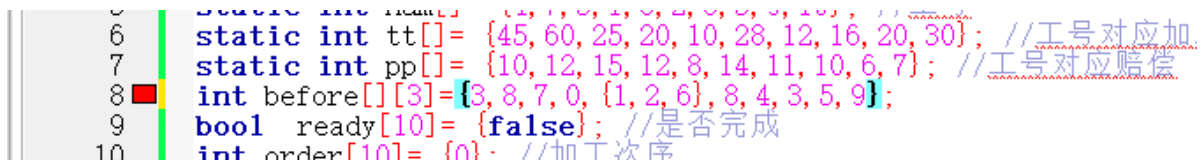
Process returned -1073741819 (0xC0000005)    execution time : 4.229 s
Press any key to continue.
```

图 18: 加工次序问题第 3 个错误输出图

分析原因：顺序队列和二叉树节点不能进行连接

修改代码：将顺序队列形式删掉，二叉树删掉，保留二叉树节点形成优先队列

4. 调试原因：*error : braces around scalar initializer for type 'int'*



```
6 static int tt[] = {45, 60, 25, 20, 10, 28, 12, 16, 20, 30}; //工号对应加工
7 static int pp[] = {10, 12, 15, 12, 8, 14, 11, 10, 6, 7}; //工号对应赔偿
8 int before[3][3] = {3, 8, 7, 0, {1, 2, 6}, 8, 4, 3, 5, 9};
9 bool ready[10] = {false}; //是否完成
10 int order[10] = {0}; //加工次序
```



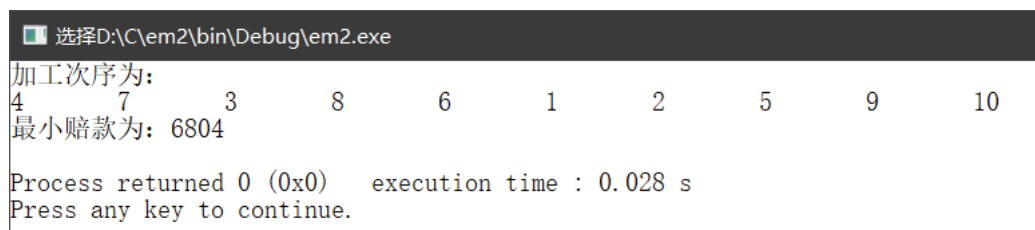
文件	行	信息
D:\C\em2\main...	8	error: braces around scalar initializer for type 'int'
D:\C\em2\main...		In function 'bool isok(int)':

图 19: 加工次序问题第 4 个错误提示图

分析原因：没有设置未找到解时的正确输入，故输出了 *ans* 的初始值

修改代码：添加状态变量 *ok*，当有解的时候赋值 *false*，输出 *ans*，否则输出-1

5. 调试原因：输入后，结果不正确



```
选择D:\C\em2\bin\Debug\em2.exe
加工次序为:
4       7       3       8       6       1       2       5       9       10
最小赔款为: 6804

Process returned 0 (0x0)    execution time : 0.028 s
Press any key to continue.
```

图 20: 加工次序问题第 5 个错误输出图

分析原因：排序错误，应该是从大到小排序

修改代码：将结构体 *BiTNode* 中的比较操作符函数中的 *>* 改为 *<*

(四) 分子集问题

调试原因：出现 `error: invalid conversion from 'int' to 'int*' [-fpermissive]` 的错误提示，如图 21

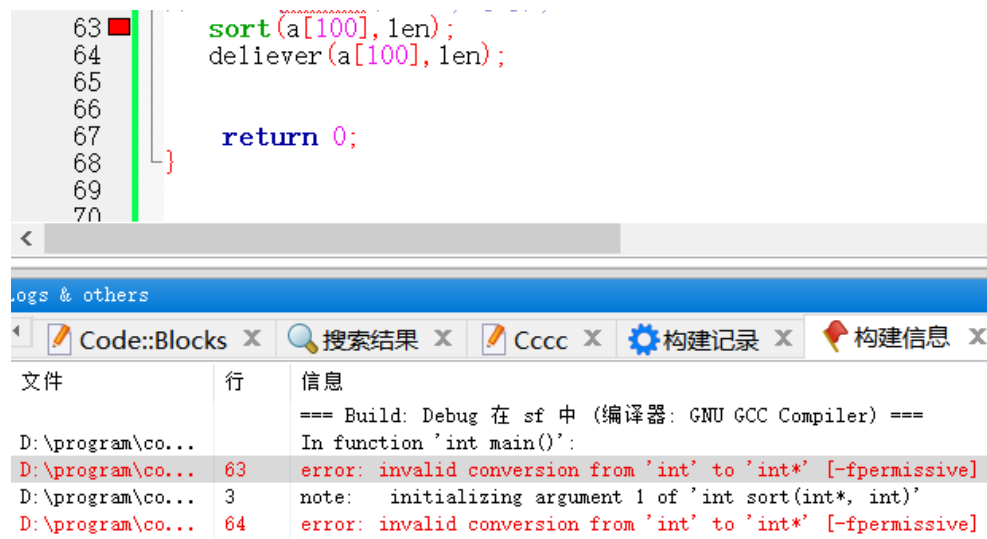


图 21: 分子集问题第一次错误 error

分析原因：`sort` 的第一个参数是指针型的，即数组 `a` 的地址，因此调用 `sort` 时，传入的参数不能是数组，直接传入地址即可。

修改代码：将 `sort(a[100], len)` 修改为 `sort(a, len)`，如图 22。

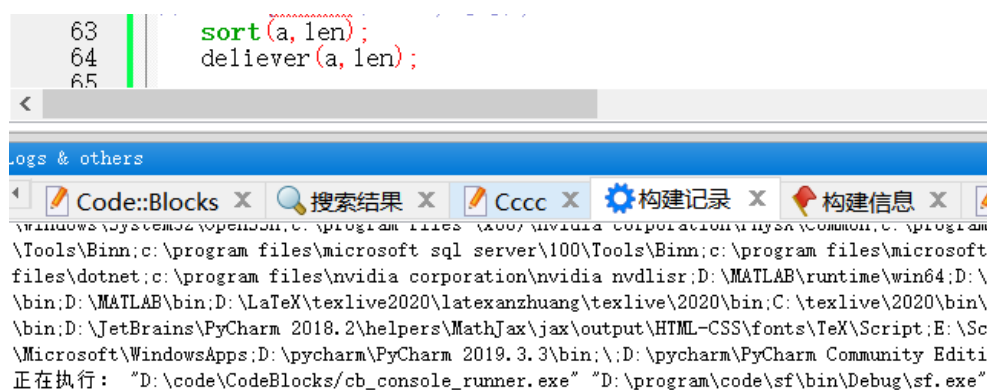


图 22: 分子集问题第一次修改

(五) 24 点游戏

调试原因：出现 `TypeError` 错误提示，如图 23，

分析原因：因为 `input` 获得的为字符，需要强制转换为 `int` 型才能带入 `find24`

```

D:\pycharm\ppg\venv\Scripts\python.exe D:/pycharm/ppg/venv/tupian.py
1
5
6
3
Traceback (most recent call last):
  File "D:/pycharm/ppg/venv/tupian.py", line 32, in <module>
    find_24(a,b,c,d)
  File "D:/pycharm/ppg/venv/tupian.py", line 6, in find_24
    cases.append('%d%s%d%s%d%s%d'%(a,o1,b,o2,c,o3,d))
TypeError: %d format: a number is required, not str

Process finished with exit code 1

```

图 23: 24 点游戏第一次错误 TypeError

修改代码: 将 a, b, c, d 转换为 int 型, 如图 24, 最后成功运行。

```

a=int(input())
b=int(input())
c=int(input())
d=int(input())
find_24(a,b,c,d)

```

图 24: 24 点游戏第一次修改

调试原因: 如果没有找到值为 24 的公式则输出一个 *failed*, 但是程序多次出现 *failed*, 且在末行出现公式, 如图 25。

D:\pycharm\ppg\venv\Scripts\	failed
9	failed
5	failed
3	failed
1	failed
failed	failed
failed	failed
failed	failed
failed	failed
failed	9+5*3*1 = 24
failed	
failed	Process finished with exit c
failed	

图 25: 24 点游戏第二次错误 failed 出现异常

分析原因: $print("failed")$ 语句在 for 循环内, 导致每循环一次, 若不存在表达式, 就会执行一次 $print$ 语句。

修改代码: 将 $if - print$ 语句提出 for 循环, 设定了一个 $flage = 0$ 作为不存在标

志，当存在 24 点表达式的时候， $flag = 1$ ，在 *for expression incases* 循环外，设置 *if* 函数，如果 $flag = 0$ ，说明没有相应的表达式，则输出 *failed*，否则，输出表达式。如图 26。

```

36         for expression in cases:
37             flag=0;#对flag进行初始化，用于记录是否存在值为24的表达式，
38             # 如果flag=0，表示没有表达式，反之表达式存在
39             try:    #捕获表达式中分母为0的异常
40                 if eval(expression) == 24: #如果表达式的值等于24
41                     print('%s = 24' % expression) #输出这个表达式
42                     flag=1 #将1赋值给flag，表明表达式存在
43                     return #返回表达式
44             except: #如果分母等于0
45                 pass #忽略该式
46         if (flag==0): #如果表达是不存在
47             print("failed") #输出: failed

```

图 26: 24 点游戏第二次修改

运用测试数据：1、1、1、1；4、4、10、10，测试结果如图 37，程序正常运行。

```

D:\pycharm\ppg\venv\Scripts\python.exe D:\pycharm\ppg\venv\Scripts\python.exe D:
1      4
1      4
1      10
1      10
failed  (10*10-4)/4 = 24

Process finished with exit code 0

```

图 27: 24 点游戏第二次调试后正常运行

为了限制用户输入的数字在 1-10 之间，提升数据的安全性，我们引入了 *while* 循环对输入的数字进行判断，代码如图 28：

```

a=int(input('请输入4张扑克牌，每一张数字介于1-10（可以重复）：\n'))
b=int(input())
c=int(input())
d=int(input())
while a<1 or b<1 or c<1 or d<1 or a>10 or b>10 or c>10 or d>10:
    print("输入扑克牌不合法")
    a = int(input('请输入4张扑克牌，每一张数字介于1-10（可以重复）：\n'))
    b = int(input())
    c = int(input())
    d = int(input())
print("输入完毕")
find_24(a,b,c,d)

```

图 28: 24 点游戏第三次修改

测试数据：56、13、4、1

测试数据：1、2、9、11

测试数据：1、5、4、6

测试结果：如图 29

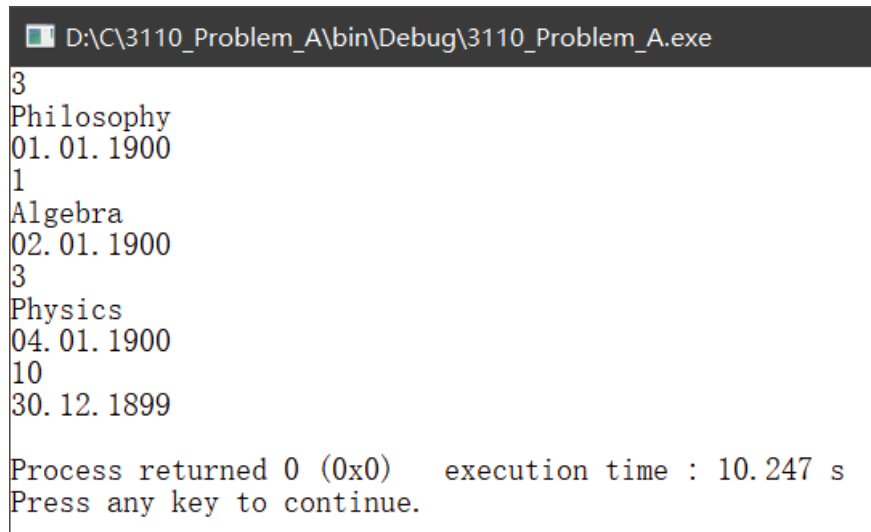
```
请输入4张扑克牌，每一张数字介于1-10（可以重复）：
56
13
4
1
输入扑克牌不合法
请输入4张扑克牌，每一张数字介于1-10（可以重复）：
1
2
9
11
输入扑克牌不合法
请输入4张扑克牌，每一张数字介于1-10（可以重复）：
-
9
11
输入扑克牌不合法
请输入4张扑克牌，每一张数字介于1-10（可以重复）：
1
5
4
6
输入完毕
6/(5/4-1) = 24
Process finished with exit code 0
```

图 29: 24 点游戏第三次调试后运行结果

六、结果分析 (含复杂性分析)

(一) 珍妮的第一次考试

1. 正确的输入及其输出结果

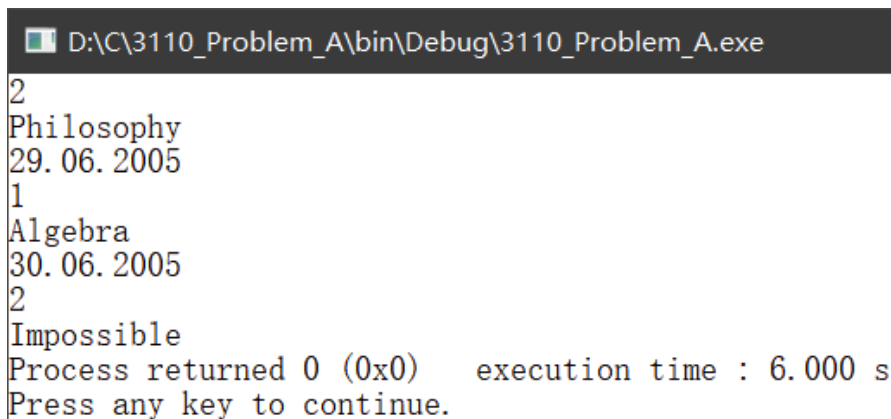


```
D:\C\3110_Problem_A\bin\Debug\3110_Problem_A.exe
3
Philosophy
01.01.1900
1
Algebra
02.01.1900
3
Physics
04.01.1900
10
30.12.1899

Process returned 0 (0x0)    execution time : 10.247 s
Press any key to continue.
```

图 30: 珍妮的第一次考试正确输入结果

2. 错误的输入及其输出结果



```
D:\C\3110_Problem_A\bin\Debug\3110_Problem_A.exe
2
Philosophy
29.06.2005
1
Algebra
30.06.2005
2
Impossible

Process returned 0 (0x0)    execution time : 6.000 s
Press any key to continue.
```

图 31: 珍妮的第一次考试错误输入结果

3. 时间复杂度分析: 该问题的时间主要花寻找最晚该复习日期的 `solve()` 函数中。记该问题给了 N 门考试, 则最好的情况是, 该问题最早考试时间的前一天没被占用, 即是最优解, 故最好时间复杂度为 $O(N\log N)$; 最坏的情况是, 测试案例没有最优点, 即没有解, 则 $O(N\log N)$

4. 空间复杂度分析: 由于该问题需要通过预处理将从 1600.1.1 开始到 2100.12.31 的日期全部转换成数字保存在数组中, 所以最占用空间的是树状数组 `c[200000]` 和日期数组 `date[200000]`, 故空间复杂度为 $O(200000)$ 。

(二) 传教士与食人族

1. 正确的输入及其输出结果

```
D:\C\3586_Problem_C\bin\Debug\3586_Problem_C.exe
1
3 2
11

Process returned 0 (0x0)   execution time : 3.616 s
Press any key to continue.
```

图 32: 加工次序问题正确输入结果

2. 错误的输入及其输出结果

```
D:\C\3586_Problem_C\bin\Debug\3586_Problem_C.exe
1
5 2
-1

Process returned 0 (0x0)   execution time : 4.666 s
Press any key to continue.
```

图 33: 加工次序问题正确输入结果

3. 时间复杂度分析：该问题的时间主要花在遍历解空间的 `Search()` 和 `DFS()` 相互递归。记限制食人族和传教士人数为 N ，船上可承载的人数为 K ，则最好的情况是，深度遍历第一轮可能状态空间即可得到解，故最好时间复杂度为 $O(NK^2)$ ；最坏的情况是，测试案例没有最优点，即没有解，故 $O(NK^2)$

4. 空间复杂度分析：由于该问题需要将深度遍历得到的可能状态保存在栈中，故空间复杂度为 $O(NK)$ 。

(三) 加工次序问题

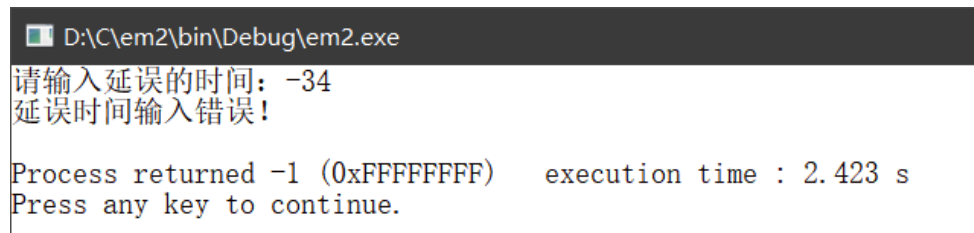
1. 正确的输入及其输出结果

```
D:\C\em2\bin\Debug\em2.exe
请输入延误的时间: 166
加工次序为:
4      7      3      8      6      1      2      5      9      10
最小赔款为: 6804

Process returned 0 (0x0)   execution time : 2.016 s
Press any key to continue.
```

图 34: 加工次序问题正确输入结果

2. 错误的输入及其输出结果



```
D:\C\em2\bin\Debug\em2.exe
请输入延误的时间: -34
延误时间输入错误!

Process returned -1 (0xFFFFFFFF)   execution time : 2.423 s
Press any key to continue.
```

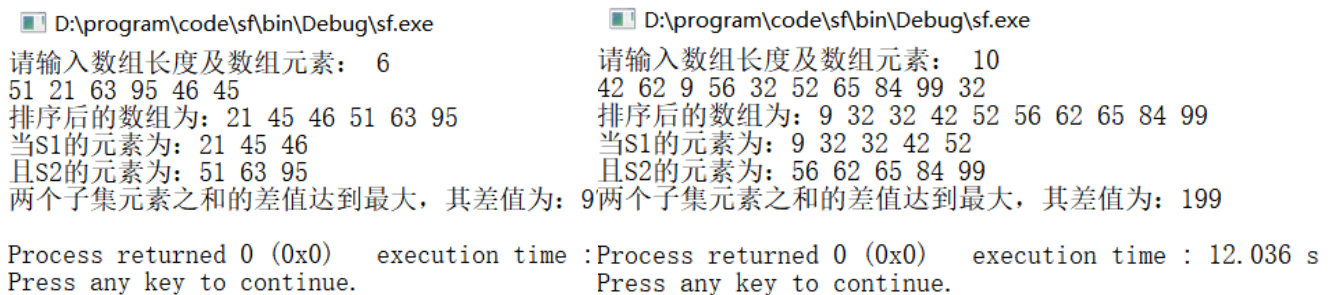
图 35: 加工次序问题正确输入结果

3. 时间复杂度分析: 该问题的时间主要花在对于有多个前件及后件的优先队列求赔偿的 `minpay()` 函数中。记该问题有 N 个工件, 每个工件有 M 个前件, 则最好的情况是, 该问题全部都是顺序的, 每个工件的前件、后件最多只有一个, 故最好时间复杂度为 $O(MN)$; 最坏的情况是, 每个工件的前件、后件都有 M 个, 则 $O(M^2N)$ 。

4. 空间复杂度分析: 该问题的空间复杂度为 $O(1)$ 。

(四) 分子集问题

1. 输出结果



```
D:\program\code\s\bin\Debug\s\sf.exe
请输入数组长度及数组元素: 6
51 21 63 95 46 45
排序后的数组为: 21 45 46 51 63 95
当S1的元素为: 21 45 46
且S2的元素为: 51 63 95
两个子集元素之和的差值达到最大, 其差值为: 9

Process returned 0 (0x0)   execution time :
Press any key to continue.

D:\program\code\s\bin\Debug\s\sf.exe
请输入数组长度及数组元素: 10
42 62 9 56 32 52 65 84 99 32
排序后的数组为: 9 32 32 42 52 56 62 65 84 99
当S1的元素为: 9 32 32 42 52
且S2的元素为: 56 62 65 84 99
两个子集元素之和的差值达到最大, 其差值为: 199

Process returned 0 (0x0)   execution time : 12.036 s
Press any key to continue.
```

图 36: 分子集问题输出结果

2. 算法的时间复杂度

由于算法运用了冒泡排序算法, 和 `for` 循环求和, 因此有 $T(n) = O(n^2) + O(n)$ 因此算法的时间复杂度为 $O(n^2)$ 。

3. 算法的空间复杂度

由于冒泡排序的最优的空间复杂度就是开始元素顺序已经排好了, 则空间复杂度为 0, 最差的空间复杂度就是开始元素逆序排序了, 则空间复杂度为 $O(n)$, 平均的空间复杂度为: $O(1)$ 。故算法的空间复杂度为 $O(1)$ 。

(五)24 点游戏

1. 输出结果



图 37: 24 点游戏第二次调试后正常运行

其中，问题一的正确输出如图 38、错误输出如图 39：

进入24点游戏请输入1，测试数字是否能得到24请按2，是否继续该游戏？
1
欢迎来到24点游戏
四张扑克分别为为： 6 9 2 6
请输入由四个数字组成的表达式： ((6+9)*2)-6
Congratulation!, 本次耗时 11.8051943778991
是否继续该游戏？
是
欢迎来到24点游戏
四张扑克分别为为： 2 7 1 7
请输入由四个数字组成的表达式： 2+7+1+7
Incorrect, 本次耗时 4.991381883621216 秒
是否继续该游戏？
否
是否进入测试数据程序？
是
欢迎进入测试数据程序！
请输入4张扑克牌，每一张数字介于1-10（可以重复）：
2
2
7
7
2*(7-2+7) = 24
所花时间为： 0.014958381652832031 秒
是否继续该游戏？
是
欢迎进入测试数据程序！
请输入4张扑克牌，每一张数字介于1-10（可以重复）：
4
4
10
5

是否继续该游戏？
是
欢迎来到24点游戏
四张扑克分别为为： 2 7 1 7
请输入由四个数字组成的表达式： 2+7+1+7
Incorrect, 本次耗时 4.991381883621216 秒
是否继续该游戏？
否
是否进入测试数据程序？
是
欢迎进入测试数据程序！
请输入4张扑克牌，每一张数字介于1-10（可以重复）：
1
2
4
6
(2-1)*4*6 = 24
所花时间为： 0.03989410400390625 秒
是否继续该游戏？
是
欢迎进入测试数据程序！
请输入4张扑克牌，每一张数字介于1-10（可以重复）：
1
5
5
5

图 38: 问题一正确输出

图 39: 问题一错误输出

问题二的运用测试数据输出得到如图 40、41:

```
欢迎进入测试数据程序!
请输入4张扑克牌, 每一张数字介于1-10 (可以重复)
2
2
7
7
2*(7-2+7) = 24
所花时间为: 0.014958381652832031 秒
是否继续该游戏?
是
欢迎进入测试数据程序!
请输入4张扑克牌, 每一张数字介于1-10 (可以重复)
4
4
10
10
(10*10-4)/4 = 24
所花时间为: 0.08876204490661621 秒
是否继续该游戏?
是
欢迎进入测试数据程序!
请输入4张扑克牌, 每一张数字介于1-10 (可以重复)
1
2
3
8
(1+3+8)*2 = 24
所花时间为: 0.015955209732055664 秒
```

图 40: 问题二输出

```
是否继续该游戏?
是
欢迎进入测试数据程序!
请输入4张扑克牌, 每一张数字介于1-10 (可以重复):
1
2
4
6
(2-1)*4*6 = 24
所花时间为: 0.03989410400390625 秒
是否继续该游戏?
是
欢迎进入测试数据程序!
请输入4张扑克牌, 每一张数字介于1-10 (可以重复):
1
5
5
5
(5-1/5)*5 = 24
所花时间为: 0.03406047821044922 秒
是否继续该游戏?
否
是否进入数独游戏?
否
是否退出程序?
是
退出程序成功!
```

图 41: 问题二输出

2. 算法的时间复杂度

由于算法运用了穷举法, 且有三个 *for* 循环, 因此算法的时间复杂度为 $O(n^3)$ 。

3. 算法的空间复杂度

虽然是三次 *for* 循环, 前两次都执行了 4 次循环, 最后一次执行了 11 次循环, 总计算量就是 19, 故空间复杂度是 $O(1)$ 。

七、设计心得体会

一、汪璐的心得体会

这次的课程设计开始让我感觉很艰难，最后出成果的时候，内心充满了喜悦。

记得我和晶晶一开始选题并不是 24 点，开始选的是第三个题，关于董事长的开会安排，但是看了两天之后，我们最终放弃了，由于时间紧迫，我们从下午放弃到第二天中午之前就要写出来这个程序，也就意味着这个题，写出代码的时间大概只有半天了，其中还包括睡觉的时候。所以那天晚上我们熬夜到四点钟，最终写出来了代码的框架，然后第二天早起，对代码进行不断的完善，最终完成了一个我俩都比较满意的程序。这段时间也有程序写不出来的抓狂，两人没思路或者程序跑出来的结果不是我们想要的答案的时候，就会疯狂吐槽，来回踱步。感觉一起做课程设计不仅仅是完成了一个任务，这中间也有很多回忆。

我们都明白，算法编程能力不可能是突飞猛进的，一切都是量变最终积累成质变！所以我们坚持啃完一篇篇文章，做完一个个程序，就在某个时间节点，我明显感觉到我的编程能力进步了，能够实现内心的更多想法，而不是这个好难啊，好像写不出来，换一个思路吧。以前我总以为，编程就是要告诉计算机去做某事，就只需要编写自己心里想的代码，问题就解决了。这次的算法设计，我明白了仅仅写完代码是不行的，我们还要去优化，去实现更多的可能；同时我也明白了不能过度依赖网络上的算法，因为不去自己设计，不走自己的路，很难知道自己真的需要什么，如果只会啃别人的东西，有时总有一种懈怠，总觉得别人都做的这么牛了，也不缺我一个。我们要坚定自己，无论结果如何，只要未来的自己可以骄傲的看待目前的过程，那就是我的胜利。

二、曾晶晶的心得体会

在一学期的算法分析与设计的课程学习中，我感受到了，算法非常难但又非常有用。想要掌握一种算法很难，熟练运用它更是难上加难，但同时很多问题的解决，程序的编写都要依赖它，很多人为它如痴如醉。一个好的算法就能减少一个问题的计算时间。

我们选的问题都挺难，曾经一度想要放弃，却还是坚持了下来，终于完成了它。这次的课程设计让我对算法的理解在理论的基础上更深了一层，可谓实践出真知。我的一个收获是，在编写程序时，一定要多问问自己：这个操作对象用什么存储结构比较好？这个函数用什么算法实现更好？这个数值的设置到底正不正确？.....，只有把自己逼成十万个为什么，才能最终获得理想的程序。

我的第二个收获是：自学了一些 C++。C++ 和我们这学期学的 Java 程序设计有点类似，都是面向对象程序设计，但是它里面有一些自带的头文件，如 `<queue>` 等都能帮我们平时编程节省很多时间和精力。在进行算法设计与分析的程序设计过程中，我还顺便复习了数据结构相关知识，如栈、队列、二叉树等。程序设计实验真是一举多得呢，让我们在理论的基础上得到实践。

算法的学习对于培养一个人的逻辑思维能力是有极大帮助的，它可以培养我们养成思考分析问题，解决问题的能力。大多数问题动态规划都可以解决，但是太慢。因此有些问题贪心确实比动态快的多。不过相对的贪心解决问题并不如动态规划精细，得不来的不一定是最优解，只能说是相对最优，根据情况选择不同的算法解决问题才是王道。在现实中，有很多问题往往需要我们把其所有可能穷举出来，然后从中找出满足某种要求的可能或最优的情况，从而得到整个问题的解。回溯算法就是解决这种问题的“通用算法”，有“万能算法”之称。如果一个问题可以同时用几种方法解决，贪心算法应该是最好的选择之一。

通过对算法设计与分析的课程设计实践，我掌握了许多经典的算法思想，思维创新能力和实践能力得到了有效的提高，并且一题多解的情况让我对不同的算法有了更加深刻的认识。这些知识在我今后的学习中将会得到更深层次的理解与应用。

参考文献

- [1] 孙成存, 李凯, 赵克. 产生式系统分析和应用 [J]. 电子科技,2004(09):49-52.
- [2] 常友渠, 肖贵元, 曾敏. 贪心算法的探讨与研究 [J]. 重庆电力高等专科学校学报,2008(3):40-42,47.
- [3] 屈婉玲, 刘田, 张立昂, 王捍贫. 算法设计与分析 (第 2 版)[M]. 北京: 清华大学出版社,2016.
- [4] 殷人昆. 数据结构 (C 语言描述)[M]. 北京: 清华大学出版社,2012.

附录 (源代码)

(一) 珍妮的第一次考试

```
1  #include <iostream>
2  #include <cstdio>
3  #include <cstring>
4  #include <algorithm>
5  #define Min(a,b) (a) > (b) ? (b) : (a)
6  #define Max(a,b) (a) > (b) ? (a) : (b)
7
8  using namespace std;
9
10 int n; // 考试场数
11 char s[10]; // 考试名称
12 int c[1000000]; // 树状数组
13 const int INF = 0x3f3f3f3f; // 设置INF为无穷大, 并令其不可改变
14 int mon[] = {23333,31,28,31,30,31,30,31,31,30,31,30,31};
15 // 月份下标从1开始, 为了让m[0]不被选上, 设置了一个极大数
16
17 bool is_leap(int& y)
18 {
19     // c++中的true转换成整型是1, false转换成整型是0
20     // 判断是否是闰年, 是则返回true, 否则返回false
21     if (y % 400 == 0 || (y % 4 == 0 && y % 100 != 0))
22         return true;
23     return false;
24 }
25
26 struct Date
27 {
28     // 定义日期结构体
29     // C++中的struct可以有函数
30     int y,m,d; // 定义成员变量年、月、日
31     void get()
32     {
33         scanf("%d.%d.%d",&d, &m, &y);
34     }
35     void init()
36     {
37         y = 1600;
38         m = d = 1;
39     }
40     void add()
41     {
42         // 在当前日期的基础上增加一天
43         d++; // 日期加1
44         if (m != 2)
45         {
46             // 当前日期的月份不是2月
47             if (d > mon[m])
48             {
49                 // 若超出给定的月份天数, 进入下一个月
50                 d = 1;
51                 ++m;
52                 if (m > 12)
53                 {
54                     // 进入下一个年份
55                     m = 1;
56                     y++;
57                 }
58             }
59         }
60     }
61 }
```

```

57         }
58     }
59 }
60 else
61 {
62     if (d > is_leap(y) + 28)
63     {
64         //根据是否是闰年判断天数是否超出
65         m = 3;
66         d = 1;
67     }
68 }
69 }
70 //结构体内嵌比较函数:
71 bool operator == (const Date& rhs) const
72 {
73     //两个结构体相等的条件是年月日都相等
74     return y == rhs.y && m == rhs.m && d == rhs.d;
75 }
76 bool operator < (const Date& rhs) const
77 {
78     //当前结构体<rhs结构体的条件:
79     return y < rhs.y || (y == rhs.y && m < rhs.m) || ((y == rhs.y && m == rhs.m && d < ...
80         rhs.d));
81 }
82 void print()
83 {
84     printf("%02d.%02d.%d\n",d,m,y);
85 }
86 } date[1000000]; //每天的日期结构体数组
87
88 int cnt;
89 void init()
90 {
91     //预处理年份
92     //将从1600.1.1开始到2100.12.31的日期逐天存入结构体数组date[]中
93     Date tmp;
94     tmp.init();
95     cnt = 0; //从0开始给date[]数组赋日期
96     while(tmp.y < 2101)
97     {
98         date[cnt++] = tmp; //将当前的日期对象存入date[]中
99         tmp.add(); //得到下一天的日期
100     }
101 }
102
103 int BinSc(Date& rhs) //LHS: Left-Hand-Side; RHS: Right-Hand-Side
104 {
105     //二分法考试日期与date[]数组中所匹配的日期对应的数字为mid
106     int l = 0, r = cnt;
107     while(l <= r)
108     {
109         int mid = (l + r) / 2; //l+r的中点的下界
110         if (date[mid] == rhs)
111             return mid;
112         else if (rhs < date[mid])
113             r = mid - 1;
114         else l = mid + 1;
115     }
116     return -1; //0~cnt区间内若没找到该日期
117 }
118
119 struct Course

```

```

118 {
119     int End ,t;
120     bool operator < (const Course& rhs) const
121     {
122         //当前结构体<rhs结构体的条件：复习的左边界早
123         return End-t < rhs.End - rhs.t;
124     }
125 } p[50000];
126
127 int lowbit(int x)
128 {
129     // 非负整数x在二进制表示下最低位及其后面的0构成的数值
130     return x & -x;
131 }
132
133 int sum(int x)
134 {
135     //得到该节点之前有多少场考试
136     int ans = 0;
137     while(x)
138     {
139         ans += c[x];
140         x -= lowbit(x);
141     }
142     return ans;
143 }
144
145 void add(int pos)
146 {
147     //建立树状数组
148     //c[pos]节点的父节点为c[pos+lowbit(pos)]
149     //将父节点及本身全部赋为1
150     //也就是说在当前日期所对应的树状数组的节点处+1，为了完整性，其父节点也需要+1
151     while (pos < cnt)
152     {
153         c[pos]++;
154         pos += lowbit(pos);
155     }
156 }
157
158 int solve()
159 {
160     int ans = INF, res;
161     for (int i = n - 1; i != -1; --i) //从排序后的数组的最后一个开始
162     {
163         int l = p[i].End - p[i].t, r = p[i].End;
164         bool ok = 0;
165         res = -1; //每轮循环中，res都被初始化为-1
166         while(l ≤ r)
167         {
168             //二分中点的点
169             //当中点到区间右端点全都被占用了，并且中点没有被占用这就是一个最合理的点！
170             //否则，继续缩小小区间来分！
171             int mid = (l + r) / 2; //二分法
172             if (sum(r) - sum(mid) == r - mid)
173                 //判断复习周期中点到考试当天这段区间是否全部被占用了（存在考试，或复习日）
174                 {
175                     if (sum(mid) - sum(mid-1) == 0) //判断中点是否被占用，若没有则为最合理点
176                     {
177                         res = Max(res, mid);
178                         add(mid); //将复习日加到树状数组中
179                         ok = 1;

```

```

180         break;
181     }
182     else
183         r = mid - 1; //若被占用了，则从
184     }
185     else //没有被全部占用则缩小区间,右端点不变，复习周期减半
186         l = mid + 1;
187     }
188     if (!ok) return -1;
189     ans = Min(ans, res); //取需要准备的时间（早的
190 }
191 return ans;
192 }
193
194 int main()
195 {
196     init();
197     Date cal;
198     scanf("%d", &n);
199     memset(c, 0, sizeof(c)); //给树状数组清零
200     for (int i = 0; i < n; i++)
201     {
202         scanf("%s", s); //获得考试名称
203         cal.get(); //获得考试日期
204         //找到该考试日期所对应的数字，并赋给该科目复习的截止日期End
205         //即考试当天不能复习
206         p[i].End = BinSc(cal);
207         add(p[i].End);
208         scanf("%d", &p[i].t); //获得考试复习时间，即需要在考试前t天内进行复习
209     }
210     sort(p, p+n); //对n个日期根据复习的左边界从早到晚排序
211     int ans = solve();
212     if (ans != -1)
213         date[ans].print();
214     else
215         printf("Impossible");
216     return 0;
217 }

```

(二) 传教士与食人族

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #define MAXSIZE 1000000
4  #define Max(lk) ((lk)+((1+lk)*(lk)/2))
5  #define Min(a,b) (a) > (b) ? (b) : (a)
6  int lm=0,lk=0; //lm个传教士，lk个食人族，船的最大载客量为lk
7
8  //定义船上人数组合 (mm, cc)
9  typedef struct Rule
10 {
11     int mm;
12     int cc;
13     void get(int m, int c) //值
14     {
15         this->cc=c;
16         this->mm=m;
17     }
18 }Rule;
19

```



```

20 //枚举船上所有可能出现的的状态
21 void Init_rule(Rule *rule)
22 {
23     int k=0;
24     for(int i=0; i<=lk; i++)
25     {
26         for(int j=0; j<=lk-i; j++)
27         {
28             if(i+j==0)
29                 continue;
30             rule[k].get(i, j);
31             k++;
32         }
33     }
34 }
35
36 //用三元组(m, c, b)表示左岸状态;
37 //其中m表示传教士在左岸的人数; c表示野人在左岸的人数;
38 //b表示船是否在左岸, 当b=1时, 表示船在左岸, 当b=0时, 表示船在右岸。
39 typedef struct State
40 {
41     int m;
42     int c;
43     int b;
44 } State;
45
46 //定义一个顺序栈存放状态, 数据结构为数组
47 typedef struct SequenStack
48 {
49     State data[MAXSIZE];
50     int top;
51 } SequenStack;
52
53 //判断状态是否被访问过,
54 int visited(SequenStack *lbank, State newst)
55 {
56     int i=0;
57     if(lbank->top!=-1)
58     { //判断是否是空表
59         for(i=0; i!=lbank->top; i++)
60             { //从栈底开始直到栈顶遍历, 判断是否有相同的状态
61                 if(newst.m==lbank->data[i].m&&newst.c==lbank->data[i].c&&newst.b==lbank->data[i].b)
62                 {
63                     return 1;
64                 }
65             }
66     }
67     return 0; //如果是空表则返回
68 }
69
70 //判断该状态是否安全
71 bool safe(SequenStack *lbank, State newst)
72 {
73     int i=visited(lbank, newst); //判断是否被访问过, 访问过: 1
74     if(i==0)
75     {
76         if(newst.m>0&&newst.m<=lm&&newst.c>0&&newst.c<=lm)
77         { //限制条件: 0 m, c 3
78             if(newst.m==0||newst.m==lm||newst.m==newst.c)
79                 { //安全条件:
80                     // 1. 传教士都不在左岸;
81                     // 2. 传教士都在左岸;

```

```

82             //          3. 传教士与野人的数目相等.
83             return true;
84         }
85     }
86 }
87 return false; //若该状态访问过则会陷入循环, 不安全
88 }
89
90 int Pop_SequenStack(SequenStack *S)
91 { //将栈顶元素出栈
92     if(S->top== -1)
93     {
94         printf("The linear list is empty,poping to stack failed! \n");
95         return 0;
96     }
97     else
98     {
99         S->top--;
100        return 1;
101    }
102 }
103
104
105 int Push_SequenStack(SequenStack *S, State st)
106 {
107     if(S->top>=MAXSIZE-1)
108     {
109         printf("The linear list is full,pushing to stack failed! \n");
110         return 0;
111     }
112     S->top++;
113     S->data[S->top]=st; //将当前状态入栈
114     return 1;
115 }
116
117 int ans;
118 bool ok; //判断是否有解
119 void get_result(SequenStack *lbank)
120 { //获得栈的长度, 即所需步数
121     int res=0;
122     if(lbank->top!= -1)
123     {
124         res=lbank->top;
125     }
126     ans=Min(ans, res);
127     ok=true;
128 }
129
130 void DFS(SequenStack *lbank, State newst, Rule *rule);
131
132 void Search(SequenStack *lbank, State newst, Rule *rule)
133 {
134     int i;
135     bool flag;
136     State st;
137     st.m=newst.m;
138     st.c=newst.c;
139     st.b=newst.b;
140     if(st.b==1)
141     { //船在左岸
142         for(i=0; i<Max(lk); i++)
143             { //船上人数组合一一尝试

```

```

144         //根据所选择组合更新左岸状态，乘船离开左岸，状态减去数量
145         st.m=newst.m-rule[i].mm;
146         st.c=newst.c-rule[i].cc;
147         st.b=0; //船停靠在右岸
148         flag=safe(lbank,st); //判断该状态是否安全
149         if(flag==true)
150         {
151             Push_SequenStack(lbank,st); //将当前状态入栈
152             DFS(lbank,st,rule);
153             Pop_SequenStack(lbank);
154         }
155     }
156 }
157 else
158 { //船在右岸
159     for(i=0; i<Max(lk); i++)
160     { //船上人数组合一一尝试
161         //根据所选择组合更新左岸状态，乘船返回左岸，状态加上数量
162         st.m=newst.m+rule[i].mm;
163         st.c=newst.c+rule[i].cc;
164         st.b=1; //船停靠在左岸
165         flag=safe(lbank,st);
166         if(flag==true) //判断该状态是否安全
167         {
168             Push_SequenStack(lbank,st); //将当前状态入栈
169             DFS(lbank,st,rule);
170             Pop_SequenStack(lbank);
171         }
172     }
173 }
174 }
175
176 //利用递归深度优先遍历找出问题的解
177 void DFS(SequenStack *lbank, State st, Rule *rule)
178 {
179     //当左岸传教士和野人正确运送到右岸的状态(0,0,0)被达到时，输出结果
180     if(st.m==0&&st.c==0&&st.b==0)
181     {
182         get_result(lbank); //找到解之后输出结果
183     }
184     else //否则继续递归查找
185     {
186         Search(lbank,st,rule);
187     }
188 }
189
190 //初始化状态
191 State init_State(int m,int k)
192 {
193     State st;
194     st.m=m; //左岸3位传教士
195     st.c=m; //左岸3位野人
196     st.b=1; //船在左岸
197     return st;
198 }
199
200 //初始化lbank表
201 SequenStack * Init_lbank()
202 {
203     SequenStack *lbank; //声明一个顺序栈lbank表
204     lbank=(SequenStack *) malloc(sizeof(SequenStack)); //分配空间
205     lbank->top=-1; //栈顶位置为-1

```

```

206     return lbank;
207 }
208
209 int main(int argc, char **argv)
210 {
211     int kase;
212     scanf("%d",&kase);
213     for(int i=0;i<kase;i++)
214     {
215         ans=0x3f3f3f3f;
216         ok=false;
217         scanf("%d %d",&lm,&lk);
218         SequenStack *lbank=Init_lbank(); //初始化一个lbank表：用于存放刚生成的节点
219         State st=init_State(lm,lk); //初始化状态（3，3，1）
220         Rule rule[Max(lk)];
221         Init_rule(rule);
222         Push_SequenStack(lbank,st);
223         DFS(lbank,st,rule);
224         if(ok==true)
225             printf("%d\n",ans);
226         else
227             printf("-1\n");
228     }
229     return 0;
230 }

```

(三) 加工次序问题

```

1     #include <iostream>
2     #include <queue>
3     using namespace std;
4
5     static int num[] = {4,7,3,1,8,2,6,5,9,10}; //工号
6     static int tt[] = {45,60,25,20,10,28,12,16,20,30}; //工号对应加工时间
7     static int pp[] = {10,12,15,12,8,14,11,10,6,7}; //工号对应赔偿
8     int ...
9         before[][3]={{3},{8},{7},{0},{1,2,6},{8},{4},{3},{5},{9}}; //before[i]表示第i+1个工件的前件
10    bool ready[10] = {false}; //ready[i]表示第i个是否完成
11    int order[10] = {0}; //加工次序
12    int T; //假定延误的时间为166
13    int DeadLine=266;
14    int sum; //赔款总额
15    int j=0;
16    int work; //当其为正时，表示所超出的时间
17
18    struct BiTNode //定义了工件号3~5（不包括5）的结构体
19    {
20        int time; //加工时间
21        int pay; //每小时赔款
22        float pt; //单位时间赔款
23        int no; //工件号
24        friend bool operator < (BiTNode a, BiTNode b) //重载比较操作符函数
25        {
26            return a.pt < b.pt;
27        }
28        void init(int t, int p, int n) //自定义的构造函数
29        {
30            this->time = t;
31            this->pay = p;
32            this->no = n;

```

```

32         pt = p*0.1/t;
33     }
34 };
35 priority_queue<BiTNode>bnode; //定义优先级队列
36
37 bool isok(int n)//判断工件的紧前工件是否完成了
38 {
39     bool ok=true;
40     if (before[n-1][0]-1==-1)//第一个可以直接开始加工
41         return true;
42     for (int i=0;i<3;i++)
43     {
44         if (before[n-1][i]==0)
45         {
46             break;
47         }
48         else if (ready[before[n-1][i]-1]==false)
49         {
50             //只要一个前件没有完成则不可加工
51             ok=false;
52             break;
53         }
54     }
55     return ok;
56 }
57
58 void calculate(int be,int en)
59 {
60     //计算顺序工序区间的赔款
61     for (int i=be; i<=en; i++,j++)
62     {
63         if (isok(num[i])==true)
64         {
65             //如果前件已完成，即该工件已经准备好了加工
66             work+=tt[i];
67             if (work>0)//如果已经超期work天
68             {
69                 sum+=work*pp[i];//则需要增加赔款
70             }
71             order[j]=num[i];
72             ready[num[i]-1]=true;//下一道工序可进行
73         }
74         else
75         {
76             cout<<num[i]<<"的前件暂未完工"<<endl;
77         }
78     }
79 }
80
81 void InitPriQueue()
82 {
83     for (int i=2; i<7; i++)//中间五个
84     {
85         BiTNode bbi;
86         bbi.init(tt[i],pp[i],num[i]);
87         bnode.push(bbi);
88     }
89 }
90
91 void minpay(priority_queue<BiTNode>&bnode)
92 {
93     InitPriQueue();

```

```

94     priority_queue<BiTNode>temp; //中间队列
95     while(!bnode.empty())
96     {
97         if (isok(bnode.top().no)==true) //若对应的前件全部完成了
98         {
99             work+=bnode.top().time;
100             sum+=work*bnode.top().pay;
101             order[j]=bnode.top().no;
102             ready[bnode.top().no-1]=true;
103             bnode.pop();
104             j++;
105             while(!temp.empty())
106             {
107                 bnode.push(temp.top());
108                 temp.pop();
109             }
110         }
111         else //否则先放到中间队列中
112         {
113             temp.push(bnode.top());
114             bnode.pop();
115         }
116     }
117 }
118
119 int main()
120 {
121     cout<<"请输入延误的时间：";
122     scanf("%d",&T);
123     if (T<0)
124     {
125         cout<<"延误时间输入错误！"<<endl;
126         return -1;
127     }
128     work=T-DeadLine;
129     calculate(0,1); //计算前两个的赔款
130     minpay(bnode); //计算中间五个的赔款
131     calculate(7,9); //计算后三个的赔款
132     cout<<"加工次序为："<<endl;
133     for(int i=0 ; i<10; i++)
134     {
135         cout<<order[i]<<"\t";
136     }
137     cout<<endl;
138     cout<<"最小赔款为："<<sum<<endl;
139 }

```

(四) 分子集问题

```

1     #include <stdio.h>
2
3     int sort(int *a,int length){
4         //参数：数组a[],数组长度，功能：对数组a进行排序
5         int i,j,flage;//i,j为循环指示器，flage为交换值
6         for(i=0;i<length-1;i++){
7             for(j=0;j<length-1-i;j++){
8                 if(a[j]>a[j+1]){
9                     flage=a[j];
10                    a[j]=a[j+1];
11                    a[j+1]=flage; //完成交换

```

```

12         }
13     }
14 }
15 //输出排序后的数组
16 printf("排序后的数组为: ");
17 for(i=0;i<length;i++){
18     printf("%d ",a[i]);
19 }
20 printf("\n");
21 return *a;
22 }
23
24 int deliever(int *a,int length){
25     //参数: 数组a[], 数组长度, 功能: 将其分为两组
26     int len=length/2;
27     //int i;
28     int s_1[500]; //将数组分为s_1,s_2两个
29     int s_2[500];
30     for(int i=0;i<length;i++){
31         s_1[i]=a[i];
32         s_2[i]=a[i+(length-2)/2+1];
33     }
34     int sum1,sum2; //引入求和
35     sum1=sum2=0;
36     printf("当S1的元素为: ");
37     for(i=0;i<len;i++){ //循环输出s1的元素 且对s1的元素进行求和
38         printf("%d ",s_1[i]);
39         sum1+=s_1[i]; //求和
40     }
41     printf("\n且S2的元素为: ");
42     for(i=0;i<len;i++){
43         printf("%d ",s_2[i]);
44         sum2+=s_2[i];
45     }
46     printf("\n两个子集元素之和的差值达到最大, 其差值为: %d\n",
47     sum1>sum2?(sum1-sum2):(sum2-sum1));
48     return 0;
49 }
50
51 int main(){
52     int a[1001];
53     int i;
54     int len; //输入数组的长度
55     printf("请输入数组长度及数组元素: ");
56     scanf("%d",&len);
57     for(i=0;i<len;i++){ //输入
58         scanf("%d",&a[i]);
59     }
60     sort(a,len);
61     deliever(a,len);
62     return 0;
63 }

```

(五)24 点游戏

```

1 import time #用于计算程序耗时
2 import random #用于生成随机数组
3 from itertools import permutations, product #用于对list进行排列组合
4 def op_24(): #计算用户表达式是否能够等于24
5     a = random.randint(1, 10)#随机生成数值为1-10的牌

```



```

68         if p == '是' or p == 'yes' or p == 'YES' or p == 'Yes' :
69             print("退出程序成功！ ")
70             F=0
71
72     while(flage==2 and F==1):
73         print("欢迎进入测试数据程序！ ")
74         a = int(input('请输入4张扑克牌，每一张数字介于1-10（可以重复）：\n'))
75         b = int(input())
76         c = int(input())
77         d = int(input())
78         while a < 1 or b < 1 or c < 1 or d < 1 or a > 10 or b > 10 or c > 10 or d > 10:
79             print("输入扑克牌不合法")
80             a = int(input('请输入4张扑克牌，每一张数字介于1-10（可以重复）：\n'))
81             b = int(input())
82             c = int(input())
83             d = int(input())
84             print("输入完毕,计时开始")
85         start = time.time()
86         find_24(a, b, c, d)
87         end = time.time()
88         print("所花时间为： ", end - start, "秒")
89         print("是否继续该游戏？ ")
90         p = input()
91         if p == '是' or p == 'yes' or p == 'YES' or p == 'Yes' or p == 'ok' or p == 'OK' ...
92             or p == 'Ok':
93             flage = 2
94         else:
95             print("是否进入数独游戏？ ")
96             p = input()
97             if p == '是' or p == 'yes' or p == 'YES' or p == 'Yes' or p == 'ok' or p == ...
98                 'OK' or p == 'Ok':
99                 flage = 1
100             else:
101                 print("是否退出程序？ ")
102                 p = input()
103                 if p == '是' or p == 'yes' or p == 'YES' or p == 'Yes' or p == 'ok' or p ...
104                     == 'OK' or p == 'Ok':
105                     print("退出程序成功！ ")
106                     F = 0

```