

CS150

算法专题 集训

Day3



2016复赛真题

| | | | | |
|---------|------------|----------|----------|-----------|
| 题目名称 | 买铅笔 | 回文日期 | 海港 | 魔法阵 |
| 题目类型 | 传统型 | 传统型 | 传统型 | 传统型 |
| 目录 | pencil | date | port | magic |
| 可执行文件名 | pencil | date | port | magic |
| 输入文件名 | pencil.in | date.in | port.in | magic.in |
| 输出文件名 | pencil.out | date.out | port.out | magic.out |
| 每个测试点时限 | 1.0秒 | 1.0秒 | 1.0秒 | 1.0秒 |
| 内存限制 | 512 MB | 512 MB | 512 MB | 512 MB |
| 测试点数目 | 20 | 10 | 20 | 20 |
| 每个测试点分值 | 5 | 10 | 5 | 5 |

提交源程序文件名

| | | | | |
|-------------|------------|----------|----------|-----------|
| 对于C++ 语言 | pencil.cpp | date.cpp | port.cpp | magic.cpp |
| 对于C 语言 | pencil.c | date.c | port.c | magic.c |
| 对于Pascal 语言 | pencil.pas | date.pas | port.pas | magic.pas |

注意事项:

1. 文件名（程序名和输入输出文件名）必须使用英文小写。
2. 除非特殊说明，结果比较方式均为忽略行末空格及文末回车的全文比较。
3. C/C++中函数main()的返回值类型必须是int，程序正常结束时的返回值必须是0。
4. 全国统一评测时采用的机器配置为：CPU AMD Athlon(tm) II x2 240 processor, 2.8GHz，内存4G，上述时限以此配置为准。
5. 只提供Linux格式附加样例文件。
6. 评测在NOI Linux下进行。
7. 编译时不打开任何优化选项。

1. 买铅笔

【问题描述】

P 老师需要去商店买 n 支铅笔作为小朋友们参加NOIP 的礼物。她发现商店一共有 3 种包装的铅笔，不同包装内的铅笔数量有可能不同，价格也有可能不同。为了公平起见，P 老师决定只买同一种包装的铅笔。

商店不允许将铅笔的包装拆开，因此P老师可能需要购买超过 n 支铅笔才够给小朋友们发礼物。

现在P 老师想知道，在商店每种包装的数量都足够的情况下，要买够至少 n 支铅笔最少需要花费多少钱。

【输入格式】

从文件`pencil.in` 中读入数据。

输入的第一行包含一个正整数 n ，表示需要的铅笔数量。

接下来三行，每行用两个正整数描述一种包装的铅笔：其中第一个整数表示这种包装内铅笔的数量，第二个整数表示这种包装的价格。

保证所有的 7 个数都是不超过 10000 的正整数。

【输出格式】

输出到文件`pencil.out` 中。

输出一行一个整数，表示P 老师最少需要花费的钱。

【样例1输入】

57

2 2

50 30

30 27

【样例1说明】

铅笔的三种包装分别是：

- 2 支装，价格为 2 ；
- 50 支装，价格为 30 ；
- 30 支装，价格为 27 。

P老师需要购买至少 57 支铅笔。

如果她选择购买第一种包装，那么她需要购买 29 份，共计 $2 \times 29 = 58$ 支，需要花费的钱为 $2 \times 29 = 58$ 。

实际上，P老师会选择购买第三种包装，这样需要买 2 份。虽然最后买到的铅笔数量更多了，为 $30 \times 2 = 60$ 支，但花费却减少为 $27 \times 2 = 54$ ，比第一种少。

对于第二种包装，虽然每支铅笔的价格是最低的，但要够发必须买 2 份，实际的花费达到了 $30 \times 2 = 60$ ，因此P老师也不会选择。

所以最后输出的答案是 54 。

【样例1输出】

54

【样例2输入】

9998

128 233

128 2333

128 666

【样例2输出】

18407

【样例3输入】

9999

101 1111

1 9999

1111 9999

【样例3输出】

89991

【子任务】

子任务会给出部分测试数据的特点。如果你在解决题目中遇到了困难，可以尝试只解决一部分测试数据。

每个测试点的数据规模及特点如下表：

| 测试点 | 整倍数 | 其他特点 |
|-------------|-----|-----------------|
| 1,2,3,4 | √ | 三种包装内的铅笔数量都是相同的 |
| 5,6,7,8 | × | |
| 9,10,11,12 | √ | 后两种包装的铅笔数量是相同的 |
| 13,14,15,16 | × | |
| 17,18 | √ | 没有特殊性质 |
| 19,20 | × | |

上表中“整倍数”的意义为：若为“√”，表示对应数据所需要的铅笔数量 n 一定是每种包装铅笔数量的整倍数（这意味着一定可以不用多买铅笔）。

2. 回文日期

【问题描述】

在日常生活中，通过年、月、日这三个要素可以表示出一个唯一确定的日期。

牛牛习惯用 8 位数字表示一个日期，其中，前 4 位代表年份，接下来 2 位代表月份，最后 2 位代表日期。显然：一个日期只有一种表示方法，而两个不同的日期的表示方法不会相同。

牛牛认为，一个日期是回文的，当且仅当表示这个日期的 8 位数字是回文的。现在，牛牛想知道：在他指定的两个日期之间（包含这两个日期本身），有多少个真实存在的日期是回文的。

【输入格式】

从文件 *date.in* 中读入数据。

输入包括两行，每行包括一个 8 位数字。

第一行表示牛牛指定的起始日期 $date_1$ 。

第二行表示牛牛指定的终止日期 $date_2$ 。

保证 $date_1$ 和 $date_2$ 都是真实存在的日期，且年份部分一定为 4 位数字，且首位数字不为 0。

保证 $date_1$ 一定不晚于 $date_2$ 。

【提示】

一个8位数字是回文的，当且仅当对于所有的 i ($1 \leq i \leq 8$) 从左向右数的第 i 个数字和第 $9 - i$ 个数字（即从右向左数的第 i 个数字）是相同的。

例如：

- 对于 2016 年 11 月 19 日，用8位数字 20161119 表示，它不是回文的。
- 对于 2010 年 1 月 2 日，用 8 位数字 20100102 表示，它是回文的。
- 对于 2010 年 10 月 2 日，用 8 位数字 20101002 表示，它不是回文的。

每一年中都有 12 个月份：

其中，1、3、5、7、8、10、12月每个月有31天；4、6、9、11月每个月有30天；而对于2月，闰年时有29天，平年时有28天。

一个年份是闰年当且仅当它满足下列两种情况其中的一种：

1. 这个年份是4的整数倍，但不是100的整数倍；
2. 这个年份是400的整数倍。

例如：

- 以下几个年份都是闰年：2000、2012、2016。
- 以下几个年份是平年：1900、2011、2014。

【输出格式】

输出到文件`date.out` 中。

输出一行，包含一个整数，表示在 $date_1$ 和 $date_2$ 之间，有多少个日期是回文的。

【样例1输入】

20110101

20111231

【样例2输入】

20000101

20101231

【样例1输出】

1

【样例2输出】

2

【样例说明】

对于样例1，符合条件的日期是 20111102。

对于样例2，符合条件的日期是 20011002 和 20100102。

【子任务】

对于 60% 的数据，满足 $date_1 = date_2$ 。

3. 海港

【问题描述】

小K是一个海港的海关工作人员，每天都有许多船只到达海港，船上通常有很多来自不同国家的乘客。

小K对这些到达海港的船只非常感兴趣，他按照时间记录下了到达海港的每一艘船只情况；对于第 i 艘到达的船，他记录了这艘船到达的时间 t_i （单位：秒），船上的乘客数量 k_i ，以及每名乘客的国籍 $x_{i,1}, x_{i,2}, \dots, x_{i,k_i}$ 。

小K统计了 n 艘船的信息，希望你帮忙计算出以每一艘船到达时间为止的 24 小时（24 小时 = 86400 秒）内所有乘船到达的乘客来自多少个不同的国家。

形式化地讲，你需要计算 n 条信息。对于输出的第 i 条信息，你需要统计满足 $t_i - 86400 < t_p \leq t_i$ 的船只 p ，在所有的 $x_{p,j}$ 中，总共有多少个不同的数。

【输入格式】

从文件`port.in` 中读入数据。

第一行输入一个正整数 n ，表示小K统计了 n 艘船的信息。

接下来 n 行，每行描述一艘船的信息：前两个整数 t_i 和 k_i 分别表示这艘船到达海港的时间和船上的乘客数量，接下来 k_i 个整数 $x_{i,j}$ 表示船上乘客的国籍。

保证输入的 t_i 是递增的，单位是秒；表示从小K第一次上班开始计时，这艘船在第 t_i 秒到达海港。

保证 $1 \leq n \leq 10^5$ ， $k_i \geq 1$ ， $\sum k_i \leq 3 \times 10^5$ ， $1 \leq x_{i,j} \leq 10^5$ ， $1 \leq t_{i-1} < t_i \leq 10^9$ 。

其中 $\sum k_i$ 表示所有的 k_i 的和， $\sum k_i = k_1 + k_2 + \cdots + k_n$ 。

【输出格式】

输出到文件`port.out` 中。

输出 n 行，第 i 行输出一个整数表示第 i 艘船到达后的统计信息。

【样例1输入】

```
3
1 4 4 1 2 2
2 2 2 3
10 1 3
```

【样例1输出】

```
3
4
4
```

【样例1说明】

第一艘船在第 1 秒到达海港，最近 24 小时到达的船是第一艘船，共有 4 个乘客，分别是来自国家 4,1,2,2，共来自 3 个不同的国家；

第二艘船在第 2 秒到达海港，最近 24 小时到达的船是第一艘船和第二艘船，共有 $4 + 2 = 6$ 个乘客，分别是来自国家 4,1,2,2,2,3，共来自 4 个不同的国家；

第三艘船在第 10 秒到达海港，最近 24 小时到达的船是第一艘船、第二艘船和第三艘船，共有 $4 + 2 + 1 = 7$ 个乘客，分别是来自国家 4,1,2,2,2,3,3，共来自 4 个不同的国家。

【样例2输入】

```
4
1 4 1 2 2 3
3 2 2 3
86401 2 3 4
86402 1 5
```

【样例2输出】

```
3
3
3
4
```

【样例2说明】

第一艘船在第 1 秒到达海港，最近 24 小时到达的船是第一艘船，共有 4 个乘客，分别是来自国家 1,2,2,3，共来自 3 个不同的国家；

第二艘船在第 3 秒到达海港，最近 24 小时到达的船是第一艘船和第二艘船，共有 $4 + 2 = 6$ 个乘客，分别是来自国家 1,2,2,3,2,3，共来自 3 个不同的国家；

第三艘船在第 86401 秒到达海港，最近 24 小时到达的船是第二艘船和第三艘船，共有 $2 + 2 = 4$ 个乘客，分别是来自国家 2,3,3,4，共来自 3 个不同的国家；

第四艘船在第 86403 秒到达海港，最近 24 小时到达的船是第二艘船、第三艘船和第四艘船，共有 $2 + 2 + 1 = 5$ 个乘客，分别是来自国家 2,3,3,4,5，共来自 4 个不同的国家。

【子任务】

- 对于 10% 的测试点, $n = 1, \sum k_i \leq 10, 1 \leq x_{i,j} \leq 10, 1 \leq t_i \leq 10$;
- 对于 20% 的测试点, $1 \leq n \leq 10, \sum k_i \leq 100, 1 \leq x_{i,j} \leq 100, 1 \leq t_i \leq 32767$;
- 对于 40% 的测试点, $1 \leq n \leq 100, \sum k_i \leq 100, 1 \leq x_{i,j} \leq 100, 1 \leq t_i \leq 86400$;
- 对于 70% 的测试点, $1 \leq n \leq 1000, \sum k_i \leq 3000, 1 \leq x_{i,j} \leq 1000, 1 \leq t_i \leq 10^9$;
- 对于 100% 的测试点, $1 \leq n \leq 10^5, \sum k_i \leq 3 \times 10^5, 1 \leq x_{i,j} \leq 10^5, 1 \leq t_i \leq 10^9$ 。

4. 魔法阵

【问题描述】

六十年一次的魔法战争就要开始了，大魔法师准备从附近的魔法场中汲取魔法能量。

大魔法师有 m 个魔法物品，编号分别为 $1, 2, \dots, m$ 。每个物品具有一个魔法值，我们用 x_i 表示编号为 i 的物品的魔法值。每个魔法值 x_i 是不超过 n 的正整数，可能有多多个物品的魔法值相同。

大魔法师认为，当且仅当四个编号为 a, b, c, d 的魔法物品满足 $x_a < x_b < x_c < x_d$ ， $x_b - x_a = 2(x_d - x_c)$ ，并且 $x_b - x_a < (x_c - x_b) \div 3$ 时，这四个魔法物品形成了一个魔法阵，他称这四个魔法物品分别为这个魔法阵的A物品，B物品，C物品，D物品。

现在，大魔法师想要知道，对于每个魔法物品，作为某个魔法阵的A物品出现的次数，作为B物品的次数，作为C物品的次数，和作为D物品的次数。

【输入格式】

从文件`magic.in` 中读入数据。

输入文件的第一行包含两个空格隔开的正整数 n 和 m 。

接下来 m 行，每行一个正整数，第 $i + 1$ 行的正整数表示 x_i ，即编号为 i 的物品的魔法值。

保证 $1 \leq n \leq 15000, 1 \leq m \leq 40000, 1 \leq x_i \leq n$ 。每个 x_i 是分别在合法范围内等概率随机生成的。

【输出格式】

输出到文件`magic.out` 中。

共输出 m 行，每行四个整数。第 i 行的四个整数依次表示编号为 i 的物品作为A,B,C,D 物品分别出现的次数。

保证标准输出中的每个数都不会超过 10^9 。

每行相邻的两个数之间用恰好一个空格隔开。

【样例1输入】**【样例1输出】**

30 8

4 0 0 0

1

0 0 1 0

24

0 2 0 0

7

0 0 1 1

28

1 3 0 0

5

0 0 0 2

29

0 0 2 2

26

0 0 1 0

24

【样例1说明】

共有 5 个魔法阵，分别为：

物品 1,3,7,6，其魔法值分别为 1,7,26,29；

物品 1,5,2,7，其魔法值分别为 1,5,24,26；

物品 1,5,7,4，其魔法值分别为 1,5,26,28；

物品 1,5,8,7，其魔法值分别为 1,5,24,26；

物品 5,3,4,6，其魔法值分别为 5,7,28,29。

以物品 5 为例，它作为A 物品出现了 1 次，作为B 物品出现了 3 次，没有作为C 物品或者D 物品出现，所以这一行输出的四个数依次为 1,3,0,0。

此外，如果我们将输出看作一个 m 行 4 列的矩阵，那么每一列上的 m 个数之和都应等于魔法阵的总数。所以，如果你的输出不满足这个性质，那么这个输出一定不正确。你可以通过这个性质在一定程度上检查你的输出的正确性。

【样例2输入】

15 15

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

【样例2输出】

5 0 0 0

4 0 0 0

3 5 0 0

2 4 0 0

1 3 0 0

0 2 0 0

0 1 0 0

0 0 0 0

0 0 0 0

0 0 1 0

0 0 2 1

0 0 3 2

0 0 4 3

0 0 5 4

0 0 0 5

【子任务】

每个测试点的详细数据范围见下表。

| 测试点编号 | n | m |
|-------|---------|---------|
| 1 | = 10 | = 12 |
| 2 | = 15 | = 18 |
| 3 | = 20 | = 25 |
| 4 | = 30 | = 35 |
| 5 | = 40 | = 50 |
| 6 | = 50 | = 70 |
| 7 | = 65 | = 100 |
| 8 | = 80 | = 125 |
| 9 | = 100 | = 150 |
| 10 | = 125 | = 200 |
| 11 | = 150 | = 250 |
| 12 | = 200 | = 350 |
| 13 | = 250 | = 500 |
| 14 | = 350 | = 700 |
| 15 | = 500 | = 1000 |
| 16 | = 700 | = 2000 |
| 17 | = 1000 | = 5000 |
| 18 | = 2000 | = 10000 |
| 19 | = 5000 | = 20000 |
| 20 | = 15000 | = 40000 |

文件使用

```
#include <fstream>
ifstream fin;    //文件读操作, 存储设备读入到内存中
ofstream fout;   //文件写操作, 内存写入存储设备
fin.open("fstream.in");
fout.open("fstream.out");
fin.close();    //显式调用关闭函数, 当程序执行结束, 文件
               //会自动关闭
fout.close();
fin >> n;    //从流中读入数据
fout << "The sum is " << sum << endl;    //向流中写入
数据
```

文件的使用

```
#include <cstdio>
freopen("freopen.txt", "r", stdin); //将输入文件重
定向到标准输入
freopen("freopenout.txt", "w", stdout); //将标准输
出重定向到文件
cin >> n;
cout << "sum is " << sum << endl;
freopen("con", "w", stdout);
cout << "sum is " << sum << endl;
fclose(stdout); //关闭文件
freopen("con", "w", stdout); //恢复, 重新打开控制台
设备
```

代码调试及Debug

Simple and Stupid

把变量的值直接输出到屏幕上

1.生成调试

| | | |
|---|----------------|---------|
|  | 编译[C] | F9 |
|  | 运行[R] | F10 |
|  | 编译运行[o] | F11 |
|  | 全部重新编译[R](U) | F12 |
|  | 语法检查[S] | |
|  | 检查当前文件语法[S](V) | Ctrl+F9 |
| 参数(W)... | | |
| 编辑Make文件 | | |
|  | 清除[I] | |
|  | 性能分析[f] | |
|  | 删除性能信息[Z] | |
| | 转到断点(X) | F2 |
| | 切换断点[T] | F4 |
|  | 调试[D] | F5 |
|  | 停止执行(Y) | F6 |

调试

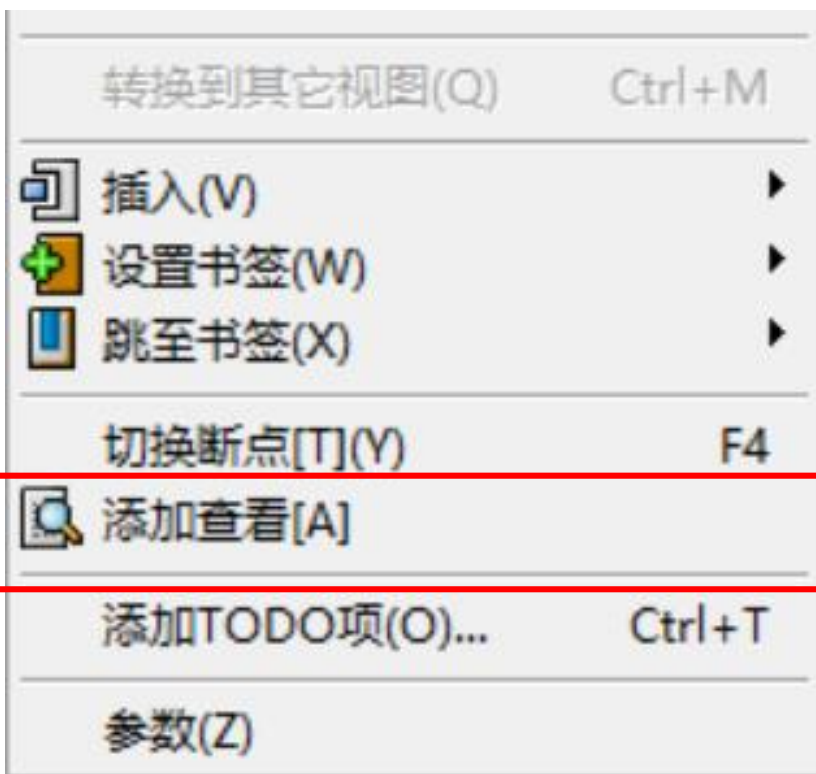
2.设置断点

断点,就是当程序执行到该地点的时候,停止,不继续执行。在断点处, 可以查看数据的值, 状态等信息。



3.添加查看

查看变量的值



4.单步执行

每次执行程序的一行

| | | | | |
|---------|------------|---------|-------|-------|
| ✓ 调试[D] | 添加查看[A] | 下一步[N] | 跳过[S] | 下一条语句 |
| ✗ 停止执行 | 查看CPU窗口[V] | 单步进入[i] | 跳过函数 | 进入语句 |

评估:

静态查错

静态查错，即不测试数据，而是通过反复地看代码来检查。

主要检查方面：

1. 检查是否有变量名打错，语法是否正确，代码和设计想法是否相符。
2. 分析代码的逻辑是否严谨正确，是否能在所有情况下都能正确运行。
3. 验证是否在所有边界情况都能得到正确的解，包括数组是否开够，会不会有 $n=0$ 的情况等等。

静态查错是最有效的查错方法，尽量让自己的程序模块化（即一个函数做一件事情），然后检查的时候边看边打上注释，这样能让自己更清醒地判断，也方便以后能轻松阅读自己的程序。

动态查错

用数据来验证代码的正确性

1. 构造测试数据：一道合格的试题，应该包括三种测试数据。
小数据：可以人工构造，也可以枚举生成。测试小数据目的是检验算法的正确性。

大数据：一般用数据生成器产生。测试大数据目的是检验算法的效率。

极端数据：极端数据非常关键。检查程序在极端情况下程序会不会爆，会不会出错。

2. 极端数据种类

“擦边球”：恰好在边界附近，且欲越界的数据。

边界大小：取最大或最小值，最多或最少值加减1的数据。

“0”：输入0，或者在计算中出现0。

“1”：只有1个数据，或者在循环、迭代中只有1个元素。

程序模板

- ✓ 形成自己的代码书写风格
- ✓ 打造属于自己的程序模板
- ✓ 常用经典算法默写

常见错误

1. 思路错误:

如果算法错误, 要重新写代码。

如果是功能缺陷, 只需修补一下。不过, 修补之后要调试, 防止产生新错误。

2. 语法错误: 可以通过编译器提示信息查出来。

3. 书写错误: 把j错写成i, 把“==”错写成“=”.....这种错误很容易犯, 并且不容易查出来

4. 忘记初始化: sum、max、min忘初始化, 或者用不应该的数初始化

5. 中间值溢出: $lcm = a * b / gcd(a, b)$; 如果a和b都比较大, 很容易发生溢出。

6. 同名变量导致混乱: 最简单的办法是保证每个变量都不重名, 忽略大小写之后仍然不重名。

7. 格式错误、文件名错误、文件放错位置.....



注意事项

答案是检验程序正误的唯一标准。至于怎么得出答案，可以采用任何手段

- 要严格按照题目规定格式来输入、输出文件。格式错误和计算结果错误等效——0分。
- 输入和输出要在指定文件中进行。直接从屏幕上输入、输出，得0分。
- 文件结束应该有换行符。
- 程序必须正常退出。在`main()`中忘记`return 0`，或者返回其他数值，即使答案正确也是0分。

养成良好编程习惯

不要把变量名打错。

不要随便地复制和粘贴。完成后马上检查所有的字母。

起一个有意义的变量名。

定义变量和指针后立刻初始化！

不要使用有冲突的名称，即使是大小写不同也不行。

指出数组的下限是0，还是1？

写简单、精炼的注释。

解释难懂的代码。

用注释把不同模块分割开。

合理的空白和缩进

保留下所有的代码版本

checklist

程序崩溃

- ❑ 在运行程序之前是否放了一个in文件？文件名是否正确？
- ❑ 数组下标是否出现了负数，或者超过了上限？
- ❑ 是不是递归的深度太大？
- ❑ 在函数内部是否有大数组？如果有，请把它挪到外面。
- ❑ 是不是变量忘初始化了？或者是变量名打错了？
- ❑ 循环条件中的字母对吗？不等号的方向正确吗？
- ❑ 指针是否初始化？

checklist

- ❑ 是不是算法效率太低，导致程序一直在慢慢地计算？
- ❑ 是不是有死循环？循环终止条件打错了？字母错了？多写或少写了等号？
- ❑ 变量初始化了吗？
- ❑ 变量名打错了吗？
- ❑ 是否在运算过程中发生了溢出？
- ❑ 变量精度够大吗？
- ❑ 数组够大吗？（定义a[100]，结果正好101个元素）
- ❑ 如果使用字符串函数，字符串末尾是不是'\0'？
- ❑ 是否把“==”错打成“=”，或者在不等号中错误地包含等号？