2020 年 CCF 非专业级软件能力认证
提高级第二轮
2020 CCF CSP-S2

时间：2020 年 11 月 7 日 14:30 ～ 18:30

题目名称	儒略日	动物园	函数调用	贪吃蛇
题目类型	传统型	传统型	传统型	传统型
目录	julian	zoo	call	snakes
可执行文件名	julian	zoo	call	snakes
输入文件名	julian.in	zoo.in	call.in	snakes.in
输出文件名	julian.out	zoo.out	call.out	snakes.out
时间限制	1.0 秒	1.0 秒	2.0 秒	2.0 秒
内存限制	256 MB	256 MB	256 MB	256 MB
测试点数目	10	20	20	20

提交源程序文件名

C++ 语言	julian.cpp	zoo.cpp	call.cpp	snakes.cpp
C 语言	julian.c	zoo.c	call.c	snakes.c
Pascal 语言	julian.pas	zoo.pas	call.pas	snakes.pas

编译选项

C++ 语言	-lm
C 语言	-lm
Pascal 语言	

注意事项（请选手仔细阅读）

1. 文件名（程序名和输入输出文件名）必须使用英文小写。
2. 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。
3. 提交的程序代码文件的放置位置请参照各省的具体要求。
4. 因违反以上三点而出现的错误或问题，申诉时一律不予受理。
5. 若无特殊说明，结果的比较方式为全文比较（过滤行末空格及文末回车）。
6. 程序可使用的栈内存空间限制与题目的内存限制一致。
7. 全国统一评测时采用的机器配置为：Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz，内存 32GB。上述时限以此配置为准。
8. 只提供 Linux 格式附加样例文件。
9. 评测在当前最新公布的 NOI Linux 下进行，各语言的编译器版本以其为准。

儒略日（julian）

【题目描述】

为了简便计算，天文学家们使用儒略日（Julian day）来表达时间。所谓儒略日，其定义为**从公元前 4713 年 1 月 1 日正午 12 点到此后某一时刻间所经过的天数**，不满一天者用小数表达。若利用这一天文学历法，则每一个时刻都将被均匀的映射到数轴上，从而得以很方便的计算它们的差值。

现在，给定一个不含小数部分的儒略日，请你帮忙计算出该儒略日（一定是某一天的中午 12 点）所对应的公历日期。

我们现行的公历为格里高利历（Gregorian calendar），它是在公元 1582 年由教皇格里高利十三世在原有的儒略历（Julian calendar）的基础上修改得到的（注：儒略历与儒略日并无直接关系）。具体而言，现行的公历日期按照以下规则计算：

1. 公元 1582 年 10 月 15 日（含）以后：适用格里高利历，每年一月 31 天、二月 28 天或 29 天、三月 31 天、四月 30 天、五月 31 天、六月 30 天、七月 31 天、八月 31 天、九月 30 天、十月 31 天、十一月 30 天、十二月 31 天。其中，闰年的二月为 29 天，平年为 28 天。当年份是 400 的倍数，或日期年份是 4 的倍数但不是 100 的倍数时，该年为闰年。
2. 公元 1582 年 10 月 5 日（含）至 10 月 14 日（含）：不存在，这些日期被删除，该年 10 月 4 日之后为 10 月 15 日。
3. 公元 1582 年 10 月 4 日（含）以前：适用儒略历，每月天数与格里高利历相同，但只要年份是 4 的倍数就是闰年。
4. 尽管儒略历于公元前 45 年才开始实行，且初期经过若干次调整，但今天人类习惯于按照儒略历最终的规则反推一切 1582 年 10 月 4 日之前的时间。注意，**公元零年并不存在**，即公元前 1 年的下一年是公元 1 年。因此公元前 1 年、前 5 年、前 9 年、前 13 年……以此类推的年份应视为闰年。

【输入格式】

输入文件名为 julian.in。

第一行一个整数 Q，表示询问的组数。

接下来 Q 行，每行一个非负整数 r_i，表示一个儒略日。

【输出格式】

输出文件名为 julian.out。

对于每一个儒略日 r_i，输出一行表示日期的字符串 s_i。共计 Q 行。

s_i 的格式如下：

1. 若年份为公元后，输出格式为 Day Month Year。其中日（Day）、月（Month）、年（Year）**均不含前导零**，中间用一个空格隔开。例如：公元 2020 年 11 月 7 日正午 12 点，输出为 7 11 2020。

2. 若年份为公元前，输出格式为 Day Month Year BC。其中年（Year）输出该年份的数值，其余与公元后相同。例如：公元前 841 年 2 月 1 日正午 12 点，输出为 1 2 841 BC。

【样例 1 输入】

```
3
10
100
1000
```

【样例 1 输出】

```
11 1 4713 BC
10 4 4713 BC
27 9 4711 BC
```

【样例 2 输入】

```
3
2000000
3000000
4000000
```

【样例 2 输出】

```
14 9 763
15 8 3501
12 7 6239
```

【样例 3】

见选手目录下的 julian/julian3.in 与 julian/julian3.ans。

【数据范围与提示】

测试点编号	$Q =$	$r_i \leq$
1	1000	365
2		10^4
3		10^5

4	10000	3×10^5
5		2.5×10^6
6	10^5	
7		5×10^6
8		10^7
9		10^9
10		答案年份不超过10^9

动物园（zoo）

【题目描述】

动物园里饲养了很多动物，饲养员小 A 会根据饲养动物的情况，按照《饲养指南》购买不同种类的饲料，并将购买清单发给采购员小 B。

具体而言，动物世界里存在 2^k 种不同的动物，它们被编号为 $0 \sim 2^k - 1$。动物园里饲养了其中的 n 种，其中第 i 种动物的编号为 a_i。

《饲养指南》中共有 m 条要求，第 j 条要求形如“如果动物园中饲养着某种动物，满足其编号的二进制表示的第 p_j 位为 1，则必须购买第 q_j 种饲料”。其中饲料共有 c 种，它们从 $1 \sim c$ 编号。本题中我们将动物编号的二进制表示视为一个 k 位 01 串，第 0 位是最低位，第 $k-1$ 位是最高位。

根据《饲养指南》，小 A 将会制定饲料清单交给小 B，由小 B 购买饲料。清单形如一个 c 位 01 串，第 i 位为 1 时，表示需要购买第 i 种饲料；第 i 位为 0 时，表示不需要购买第 i 种饲料。

实际上根据购买到的饲料，动物园可能可以饲养更多的动物。更具体地，如果将当前未被饲养的编号为 x 的动物加入动物园饲养后，饲料清单没有变化，那么我们认为动物园当前还能饲养编号为 x 的动物。

现在小 B 想请你帮忙算算，动物园目前还能饲养多少种动物。

【输入格式】

输入文件名为 zoo.in。

第一行包含四个以空格分隔的整数 n、m、c、k。分别表示动物园中动物数量、《饲养指南》要求数、饲料种数与动物编号的二进制表示位数。

. 第二行 n 个以空格分隔的整数，其中第 i 个整数表示 a_i。

接下来 m 行，每行两个整数 p_i，q_i 表示一条要求。

数据保证所有 a_i 互不相同，所有的 q_i 互不相同。

【输出格式】

输出文件名为 zoo.out。

仅一行一个整数表示答案。

【样例 1 输入】

```
3 3 5 4
1 4 6
0 3
2 4
2 5
```

【样例 1 输出】

13

【样例 1 解释】

动物园里饲养了编号为 1、4、6 的三种动物，《饲养指南》上 3 条要求为：

1. 若饲养的某种动物的编号的第 0 个二进制位为 1，则需购买第 3 种饲料。
2. 若饲养的某种动物的编号的第 2 个二进制位为 1，则需购买第 4 种饲料。
3. 若饲养的某种动物的编号的第 2 个二进制位为 1，则需购买第 5 种饲料。

饲料购买情况为：

1. 编号为 1 的动物的第 0 个二进制位为 1，因此需要购买第 3 种饲料；
2. 编号为 4、6 的动物的第 2 个二进制位为 1，因此需要购买第 4、5 种饲料。

由于在当前动物园中加入一种编号为 0,2,3,5,7,8,⋯,15 之一的动物，购物清单都不会改变，因此答案为 13。

【样例 2 输入】

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

【样例 2 输出】

2

【样例 3】

见选手目录下的 zoo/zoo3.in 与 zoo/zoo3.ans。

【数据范围与提示】

对于 20%的数据：$k \le n \le 5$，$m \le 10$，$c \le 10$，所有的 p_i 互不相同。
对于 40%的数据：$n \le 15$，$k \le 20$，$m \le 20$，$c \le 20$。
对于 60%的数据：$n \le 30$，$k \le 30$，$m \le 1000$。
对于 100%的数据：$0 \le n, m \le 10^6$，$0 \le k \le 64$，$1 \le c \le 10^8$。

函数调用（call）

【题目描述】

函数是各种编程语言中一项重要的概念，借助函数，我们总可以将复杂的任务分解成一个个相对简单的子任务，直到细化为十分简单的基础操作，从而使代码的组织更加严密、更加有条理。然而，过多的函数调用也会导致额外的开销，影响程序的运行效率。

某数据库应用程序提供了若干函数用以维护数据。已知这些函数的功能可分为三类：

1. 将数据中的指定元素加上一个值；
2. 将数据中的每一个元素乘以一个相同值；
3. 依次执行若干次函数调用，保证不会出现递归（即不会直接或间接地调用本身）。

在使用该数据库应用时，用户可一次性输入要调用的函数序列（一个函数可能被调用多次），在依次执行完序列中的函数后，系统中的数据被加以更新。某一天，小 A 在应用该数据库程序处理数据时遇到了困难：由于频繁而低效的函数调用，系统在执行操作时进入了无响应的状态，他只好强制结束了数据库程序。为了计算出正确数据，小 A 查阅了软件的文档，了解到每个函数的具体功能信息，现在他想请你根据这些信息帮他计算出更新后的数据应该是多少。

【输入格式】

输入文件名为 call.in。

第 1 行一个正整数 n，表示数据的个数。

第 2 行 n 个整数，第 i 个整数表示下标为 i 的数据的初始值为 a_i。

第 3 行一个正整数 m，表示数据库应用程序提供的函数个数。函数从 $1\sim m$ 编号。

接下来 m 行中，第 j（$1\le j\le m$）行的第一个整数为 T_j，表示 j 号函数的类型：

1. 若 $T_j=1$，接下来两个整数 P_j,V_j 分别表示要执行加法的元素的下标及其增加的值；
2. 若 $T_j=2$，接下来一个整数 V_j 表示所有元素所乘的值；
3. 若 $T_j=3$，接下来一个正整数 C_j 表示 j 号函数要调用的函数个数，随后 C_j 个整数 $g_1^{(j)},g_2^{(j)},\cdots,g_{C_j}^{(j)}$ 依次表示其所调用的函数的编号。

第 $m+4$ 行一个正整数 Q，表示输入的函数操作序列长度。

第 $m+5$ 行 Q 个整数 f_i，第 i 个整数表示第 i 个执行的函数的编号。

【输出格式】

输出文件名为 call.out。

一行 n 个用空格隔开的整数，按照下标 $1 \sim n$ 的顺序，分别输出在执行完输入的函数序列后，数据库中每一个元素的值。**答案对 998244353 取模。**

【样例 1 输入】

```
3
1 2 3
3
1 1 1
2 2
3 2 1 2
2
2 3
```

【样例 1 输出】

```
6 8 12
```

【样例 1 解释】

1 号函数功能为将 a_1 的值加一。2 号函数功能为所有元素乘 2。3 号函数将先调用 1 号函数，再调用 2 号函数。

最终的函数序列先执行 2 号函数，所有元素的值变为 2,4,6。

再执行 3 号函数时，先调用 1 号函数，所有元素的值变为 3,4,6。再调用 2 号函数，所有元素的值变为 6,8,12。

【样例 2 输入】

```
10
1 2 3 4 5 6 7 8 9 10
8
3 2 2 3
3 2 4 5
3 2 5 8
2 2
3 2 6 7
1 2 5
1 7 6
2 3
3
1 2 3
```

【样例 2 输出】

36 282 108 144 180 216 504 288 324 360

【样例 3】

见选手目录下的 call/call3.in 与 call/call3.ans。

【数据范围与提示】

测试点编号	$n, m, Q \leq$	$\sum C_j$	其他特殊限制
$1 \sim 2$	1000	$= m - 1$	函数调用关系构成一棵树
$3 \sim 4$		≤ 100	无
$5 \sim 6$	20000	≤ 40000	不含第 2 类函数或不含第 1 类函数
7		$= 0$	无
$8 \sim 9$		$= m - 1$	函数调用关系构成一棵树
$10 \sim 11$		$\leq 2 \times 10^5$	无
$12 \sim 13$	10^5		不含第 2 类函数或不含第 1 类函数
14		$= 0$	无
$15 \sim 16$		$= m - 1$	函数调用关系构成一棵树
$17 \sim 18$		$\leq 5 \times 10^5$	无
$19 \sim 20$		$\leq 10^6$	

对于所有数据：$0 \leq a_i \leq 10^4$，$T_j \in \{1,2,3\}$，$1 \leq P_j \leq n$，$0 \leq V_j \leq 10^4$，$1 \leq g_k^{(j)} \leq m$，$1 \leq f_i \leq m$。

贪吃蛇（snakes）

【题目描述】

草原上有 n 条蛇，编号分别为 $1,2,\cdots,n$。初始时每条蛇有一个体力值 a_i，我们称编号为 x 的蛇实力比编号为 y 的蛇强当且仅当它们当前的体力值满足 $a_x > a_y$，或者 $a_x = a_y$ 且 $x > y$。

接下来这些蛇将进行决斗，决斗将持续若干轮，每一轮实力最强的蛇拥有选择权，可以选择吃或者不吃掉实力最弱的蛇：

1. 如果选择吃，那么实力最强的蛇的体力值将减去实力最弱的蛇的体力值，实力最弱的蛇被吃掉，退出接下来的决斗。之后开始下一轮决斗。
2. 如果选择不吃，决斗立刻结束。

每条蛇希望在自己不被吃的前提下在决斗中尽可能多吃别的蛇（显然，蛇不会选择吃自己）。

现在假设每条蛇都足够聪明，请你求出决斗结束后会剩几条蛇。

本题有多组数据，对于第一组数据，每条蛇体力会全部由输入给出，之后的每一组数据，会相对于上一组的数据，修改一部分蛇的体力作为新的输入。

【输入格式】

输入文件名为 snakes.in。

第一行一个正整数 T，表示数据组数。

接下来有 T 组数据，对于第 1 组数据，第一行一个正整数 n，第二行 n 个非负整数表示 a_i。

对于第 2 组到第 T 组数据，每组数据：

第一行第一个非负整数 k 表示体力修改的蛇的个数。

第二行 $2k$ 个整数，每两个整数组成一个二元组 (x,y)，表示依次将 a_x 的值改为 y。一个位置可能被修改多次，以最后一次修改为准。

【输出格式】

输入文件名为 snakes.out。

输出 T 行，每行一个整数表示最终存活的蛇的条数。

【样例 1 输入】

```
2
3
11 14 14
3
1 5 2 6 3 25
```

【样例 1 输出】

```
3
1
```

【样例 1 解释】

第一组数据，第一轮中 3 号蛇最强，1 号蛇最弱。若 3 号蛇选择吃，那么它将在第二轮被 2 号蛇吃掉。因此 3 号蛇第一轮选择不吃，3 条蛇都将存活。

对于第二组数据，3 条蛇体力变为 5,6,25。第一轮中 3 号蛇最强，1 号蛇最弱，若它选择吃，那么 3 号蛇体力值变为 20，在第二轮中依然是最强蛇并能吃掉 2 号蛇，因此 3 号蛇会选择两轮都吃，最终只有 1 条蛇存活。

【样例 2 输入】

```
2
5
13 31 33 39 42
5
1 7 2 10 3 24 4 48 5 50
```

【样例 2 输出】

```
5
3
```

【样例 3】

见选手目录下的 snakes/snakes3.in 与 snakes/snakes3.ans。

【样例 4】

见选手目录下的 snakes/snakes4.in 与 snakes/snakes4.ans。

【数据范围与提示】

对于 20%的数据：$n = 3$。

对于 40%的数据：$n \leq 10$。

对于 55%的数据：$n \leq 2000$。

对于 70%的数据：$n \leq 5 \times 10^4$。

对于 100%的数据：$3 \leq n \leq 10^6$，$1 \leq T \leq 10$，$0 \leq k \leq 10^5$，$0 \leq a_i, y \leq 10^9$。保证每组数据（包括所有修改完成后的）的 a_i 以不降顺序排列。