

2014 年安徽省青少年信息学奥林匹克竞赛

中学组试题

第二试

AOI 2014

(请选手仔细阅读本页内容)

比赛用时 (5 小时)

竞赛时间：2014 年 4 月 19 日 8:00–13:00

题目名称	支线剧情	保龄球	奇怪的计算器
目录	story	bowling	calc
可执行文件名	story.exe	bowling.exe	calc.exe
输入文件名	story.in	bowling.in	calc.in
输出文件名	story.out	bowling.out	calc.out
每个测试点时限	1 秒	1 秒	1 秒
内存限制	256M	256M	256M
测试点数目	10	10	10
每个测试点分值	10	10	10
是否有部分分	否	否	否
题目类型	传统型	传统型	传统型

提交源程序须加后缀

对于 Pascal 语言	story.pas	bowling.pas	calc.pas
对于 C 语言	story.c	bowling.c	calc.c
对于 C++ 语言	story.cpp	bowling.cpp	calc.cpp

注意：最终测试时，所有编译命令均不打开任何优化开关。

支线剧情

【故事背景】

宅男 JYY 非常喜欢玩 RPG 游戏，比如仙剑，轩辕剑等等。不过 JYY 喜欢的并不是战斗场景，而是类似电视剧一般的充满恩怨情仇的剧情。这些游戏往往都有很多的支线剧情，现在 JYY 想花费最少的时间看完所有的支线剧情。

【问题描述】

JYY 现在所玩的 RPG 游戏中，一共有 N 个 **剧情点**，由 1 到 N 编号，第 i 个 **剧情点** 可以根据 JYY 的不同的选择，而经过不同的 **支线剧情**，前往 K_i 种不同的新的 **剧情点**。当然如果 K_i 为 0，则说明 i 号剧情点是游戏的一个结局了。

JYY 观看一个支线剧情需要一定的时间。

JYY 一开始处在 1 号剧情点，也就是游戏的开始。

显然任何一个剧情点都是从 1 号剧情点可达的。此外，随着游戏的进行，剧情是不可逆的。所以游戏保证从任意剧情点出发，都不能再回到这个剧情点。

由于 JYY 过度使用修改器，导致游戏的“存档”和“读档”功能损坏了，所以 JYY 要想回到之前的剧情点，唯一的方法就是退出当前游戏，并开始新的游戏，也就是回到 1 号剧情点。JYY 可以在任何时刻退出游戏并重新开始。

不断开始新的游戏重复观看已经看过的剧情是很痛苦，JYY 希望花费最少的时间，看完所有不同的支线剧情。

【输入格式】

从文件 **story.in** 中读入数据。

输入一行包含一个正整数 N 。

接下来 N 行，第 i 行为 i 号剧情点的信息；

第一个整数为 K_i ，接下来 K_i 个整数对， b_{ij} 和 t_{ij} ，表示从剧情点 i 可以前往剧情点 b_{ij} ，并且观看这段支线剧情需要花费 t_{ij} 的时间。

【输出格式】

输出到文件 **story.out** 中。

输出一行包含一个整数，表示 JYY 看完所有支线剧情所需要的最少时间。

【输入样例】

```
6
2 2 1 3 2
```

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

【输出样例】

```
24
```

【样例说明】

JYY 需要重新开始 3 次游戏，加上一开始的一次游戏，4 次游戏的进程是 $1 \rightarrow 2 \rightarrow 4$, $1 \rightarrow 2 \rightarrow 5$, $1 \rightarrow 3 \rightarrow 5$ 和 $1 \rightarrow 3 \rightarrow 6$ 。

【数据规模】

对于 30% 的数据满足 $N \leq 25, \sum_{i=1}^N K_i \leq 100$;

对于 70% 的数据满足 $N \leq 100, \sum_{i=1}^N K_i \leq 2000$;

对于 100% 的数据满足 $N \leq 300, 0 \leq K_i \leq 50, 1 \leq t_{ij} \leq 300, \sum_{i=1}^N K_i \leq 5000$ 。

保龄球

【故事背景】

JYY 很喜欢打保龄球，虽然技术不高，但是还是总想着的高分。这里 JYY 将向你介绍他所参加的特殊保龄球比赛的规则，然后请你帮他得到尽量多的分数。

【问题描述】

一场保龄球比赛一共有 N 个轮次，每一轮都会有 10 个木瓶放置在木板道的另一端。每一轮中，选手都有两次投球的机会来尝试击倒全部的 10 个木瓶。对于每一次投球机会，选手投球的得分等于这一次投球所击倒的木瓶数量。选手每一轮的得分是他两次机会击倒全部木瓶的数量。

对于每一个**轮次**，有如下三种情况：

1、“**全中**”：如果选手第一次尝试就击倒了全部 10 个木瓶，那么这一轮就称为“全中”。在一个“全中”轮中，由于所有木瓶在第一次尝试中都被击倒，所以选手不需要再进行第二次投球尝试。同时，在计算总分时，选手在下一轮的得分将会被乘 2 计入总分。

2、“**补中**”：如果选手使用两次尝试击倒了 10 个木瓶，那么这一轮就称为“补中”。同时，在计算总分时，选手在下一轮中的第一次尝试的得分将会被乘以 2 计入总分。

3、“**失误**”：如果选手未能通过两次尝试击倒全部的木瓶，那么这一轮就被称为“失误”。同时，在计算总分时，选手在下一轮的得分会被计入总分，没有分数被翻倍。

此外，如果第 N 轮是“全中”，那么选手可以进行一次附加轮：也就是，如果第 N 轮是“全中”，那么选手将一共进行 $N+1$ 轮比赛。显然，在这种情况下，第 $N+1$ 轮的分数一定会被加倍。

附加轮的规则只执行一次。也就是说，即使第 $N+1$ 轮选手又打出了“全中”，也不会进行第 $N+2$ 轮比赛。因而，附加轮的成绩不会使得其他轮的分数翻番。

最后，选手的总得分就是附加轮规则执行过，并且分数按上述规则加倍后的每一轮分数之和。

JYY 刚刚进行了一场 N 个轮次的保龄球比赛，但是，JYY 非常不满意他的得分。

JYY 想出了一个办法：他可以把记分表上，他所打出的所有**轮次**的顺序重新排列，这样重新排列之后，由于翻倍规则的存在，JYY 就可以得到更高的分数了！

当然了，JYY 不希望做的太假，他希望保证重新排列之后，所需要进行的轮数和重排前所进行的轮数是一致的：比如如果重排前 JYY 在第 N 轮打出了“全中”，那么重排之后，第 N 轮还得是“全中”以保证比赛一共进行 $N+1$ 轮；同样的，如果 JYY 第 N 轮没有打出“全中”，那么重排过后第 N 轮也不能是全中。

请你帮助 JYY 计算一下，他可以得到的最高的分数。

【输入格式】

从文件 **bowling.in** 中读入数据。

第一行包含一个整数 N ，表示保龄球比赛所需要进行的轮数。

接下来包含 N 或者 $N+1$ 行，第 i 行包含两个非负整数 x_i 和 y_i ，表示 JYY 在这一轮两次投球尝试所得到的分数， x_i 表示第一次尝试， y_i 表示第二次尝试。

我们用 $x_i = 10, y_i = 0$ 表示一个“全中”轮。输入数据保证 $x_i + y_i \leq 10$ 。

读入数据存在 $N+1$ 行，当且仅当 $x_N = 10$ 且 $y_N = 0$ 。

【输出格式】

输出到文件 **bowling.out** 中。

输出一行一个整数，表示 JYY 最大可能得到的分数。

【样例输入】

```
2
5 2
10 0
3 7
```

【样例输出】

```
44
```

【样例说明】

按照输入顺序，JYY 将得到 37 分。

最佳方案是将 3 个轮次排列成如下顺序：

```
3 7
10 0
5 2
```

【数据规模与约定】

对于 20% 的数据满足 $N \leq 9$ ；

对于 50% 的数据满足 $N \leq 20$ ；

对于 100% 的数据满足 $N \leq 50$ 。

奇怪的计算器

【故事背景】

JYY 有个奇怪的计算器，有一天这个计算器坏了，JYY 希望你能帮助他写一个程序来模拟这个计算器的运算。

【问题描述】

JYY 的计算器可以执行 N 条预设好的指令。每次 JYY 向计算器输入一个 正整数 X ，计算器就会以 X 作为初始值，接着依次执行预设的 N 条指令，最后把最终得出的结果返回给 JYY。

每一条指令可以是以下四种指令之一：（这里 a 表示一个 正整数。）

1、 $+ a$ ：表示将当前的结果加上 a ；

2、 $- a$ ：表示将当前的结果减去 a ；

3、 $* a$ ：表示将当前的结果乘以 a ；

4、 $@ a$ ：表示将当前的结果加上 $a * X$ （ X 是一开始 JYY 输入的数）。

计算器用于记录运算结果的变量的存储范围是有限的，所以每次运算结束后会有计算结果溢出的问题。

JYY 的计算器中，存储每计算结果的变量只能存储 L 到 R 之间的 正整数，如果一次指令执行过后，计算结果超过了 R ，那么计算器就会自动把结果变成 R ，然后再以 R 作为当前结果继续进行之后的计算。同理，如果运算结果小于 L ，计算器也会把结果变成 L ，再接着计算。

比如，假设计算器可以存储 1 到 6 之间的值，如果当前的计算结果是 2，那么在执行 $+ 5$ 操作之后，存储结果的变量中的值将会是 6。虽然 $2+5$ 的实际结果是 7，但是由于 7 超过了存储范围的上界，所以结果就被自动更正成了上界的大小，也就是 6。

JYY 一共想在计算器上输入 Q 个值，他想知道这 Q 个值输入计算器之后，分别会得到什么结果呢？

【输入格式】

从文件 `calc.in` 中读入数据。

输入文件的第一行包含三个正整数， N ， L 和 R ；

第接下来 N 行，每行一个指令，每个指令如题述，由一个字符和一个正整数组成，字符和正整数中间有一个空格隔开；

第 $N+2$ 行包含一个整数 Q ，表示 JYY 希望输入的数的数量；

第接下来 Q 行每行一个正整数，第 k 个正整数 X_k 表示 JYY 在第 k 次输入的整数。

【输出格式】

输出到文件 *calc.out* 中。

输出 Q 行每行一个正整数，第 k 行的整数表示输入 X_k 后，依次经过 N 个指令进行计算所得到的结果。

【样例输入】

```
5 1 6
+ 5
- 3
* 2
- 7
@ 2
3
2
1
5
```

【样例输出】

```
5
3
6
```

【样例说明】

当 JYY 输入 2 时，计算器会进行 5 次运算，每一次运算之后得到的结果分别是 6（实际计算结果为 7 但是超过了上界），3，6，1（实际结果为 -1 但是低于了下界）和 5（由于一开始输入的是 2，所以这一次计算为 $1+2\times 2$ ）。

【数据规模】

对于 10% 的数据满足 $N, Q \leq 100$ 并且指令仅含有 + 和 -；

对于 30% 的数据满足 $N \leq 3000$ 并且指令值仅含有 + 和 -；

对于 50% 的数据满足 $N \leq 3000$ ， $R, a \leq 10^6$ 并且指令值仅含有 +，- 和 *；

对于额外 30% 的数据满足指令值仅含有 +，- 和 @；

对于 100% 的数据满足 $1 \leq N, Q \leq 10^5$ ， $1 \leq L \leq X_k \leq R \leq 10^9$ ， $1 \leq a \leq 10^9$ 。