2016 年全国青少年信息学奥林匹克冬令营

竞赛时间：2016 年 1 月 30 日 8:00-13:00

题目名称	挑战 NPC	论战捆竹竿	鏖战表达式
目录	npc	jie	expr
可执行文件名	npc	jie	expr
输入文件名	npc.in	jie.in	expr.in
输出文件名	npc.out	jie.out	expr.out
每个测试点时限	1 秒	1 秒	1 秒
内存限制	256MB	256MB	1GB
测试点数目	10	20	20
每个测试点分值	10	5	5
是否有部分分	否	否	否
题目类型	传统型	传统型	交互式程序型
是否有附加文件	是	是	是

提交源程序须加后缀

对于 C++ 语言	npc.cpp	jie.cpp	expr.cpp
对于 C 语言	npc.c	jie.c	expr.c
对于 Pascal 语言	npc.pas	jie.pas	expr.pas

编译开关

对于 C++ 语言	-O2 -lm	-O2 -lm	-O2 -lm
对于 C 语言	-O2 -lm	-O2 -lm	-O2 -lm
对于 Pascal 语言	-O2	-O2	-O2

挑战 NPC

【问题描述】

小 N 最近在研究 NP 完全问题，小 O 看小 N 研究得热火朝天，便给他出了一道这样的题目：

有 n 个球，用整数 1 到 n 编号。还有 m 个筐子，用整数 1 到 m 编号。每个筐子最多能装 3 个球。

每个球只能放进特定的筐子中。具体有 e 个条件，第 i 个条件用两个整数 v_i 和 u_i 描述，表示编号为 v_i 的球可以放进编号为 u_i 的筐子中。

每个球都必须放进一个筐子中。如果一个筐子内有**不超过** 1 个球，那么我们称这样的筐子为**半空**的。

求半空的筐子最多有多少个，以及在最优方案中，每个球分别放在哪个筐子中。

小 N 看到题目后瞬间没了思路，站在旁边看热闹的小 I 嘿嘿一笑："水题！"然后三言两语道出了一个多项式算法。

小 N 瞬间就惊呆了，三秒钟后他回过神来一拍桌子："不对！这个问题显然是 NP 完全问题，你算法肯定有错！"

小 I 浅笑："所以，等我领图灵奖吧！"

小 O 只会出题不会做题，所以找到了你——请你对这个问题进行探究，并写一个程序解决此题。

【输入格式】

输入文件 ***npc.in*** 第一行包含 1 个正整数 T，表示有 T 组数据。

对于每组数据，第一行包含 3 个正整数 n, m, e，表示球的个数，筐子的个数和条件的个数。

接下来 e 行，每行包含 2 个整数 v_i, u_i，表示编号为 v_i 的球可以放进编号为 u_i 的筐子。

【输出格式】

输出文件为 ***npc.out***。

对于每组数据，先输出一行，包含一个整数，表示半空的筐子最多有多少个。

然后再输出一行，包含 n 个整数 $p_1, p_2, \ldots, p_n$，相邻整数之间用空格隔开，表示一种最优解。其中 p_i 表示编号为 i 的球放进了编号为 p_i 的筐子。如果有多种最优解，可以输出其中任何一种。

【样例输入 1】

```
1
4 3 6
1 1
2 1
2 2
3 2
3 3
4 3
```

【样例输出 1】

```
2
1 2 3 3
```

【样例输入输出 2】

见选手目录下的 ***npc/npc.in*** 与 ***npc/npc.ans***。

【数据规模和约定】

对于所有数据，$T \leq 5$，$1 \leq n \leq 3m$。保证 $1 \leq v_i \leq n$，$1 \leq u_i \leq m$，且不会出现重复的条件。

保证至少有一种合法方案，使得每个球都放进了筐子，且每个筐子内球的个数不超过 3。

各测试点满足以下约定：

测试点	m	约定
1	≤ 10	$n \leq 20$，$e \leq 25$
2		
3	≤ 100	$e = nm$
4		存在方案使得有 m 个半空的筐子
5		不存在有半空的筐子的方案
6		
7		无
8		
9		
10		

论战捆竹竿

【问题描述】

这是一个美好的下午，小 W 和小 C 在竹林里切磋捆竹竿的技艺。

竹林里有无数根完全一样的短竹子，每一根竹子由 n 节组成。

这些竹子比较特别，每一节都被染上了颜色。可能的颜色一共 26 种，分别用小写英文字母 a 到 z 表示。也就是说，如果把竹子的底端到顶端的颜色按顺序写出来可以排成一个由小写英文字母组成的字符串。

小 W 和小 C 都是捆竹竿的高手，他们知道怎样才能把零散的短竹子捆成一整根长竹竿。初始时你拿着一根短竹子作为当前的竹竿。每次你可以选择一根短竹子，短竹子底端若干节（可以是 0 节）与竹竿的最上面若干节对应地一节一节捆起来，而短竹子前面剩下的节伸出去，这样就得到了一根更长的竹竿。注意，竹子的底端是靠近根部的那一端，不可以颠倒。

小 W 对竹竿的审美要求很高，他捆竹竿时有一个癖好：如果两根竹子的某两节被捆在了一起，那么它们的颜色必须相同。

我们假设一根短竹子从底端到顶端每节的颜色为 aba。

那么两根竹子可以首尾捆在一起，可以得到一根颜色为 abaaba 的竹竿；也可以将第一根顶端的一节 a 与第二根底端的一节 a 捆在一起，得到一根颜色为 ababa 的竹竿；还可以直接将每一节都对应起来，捆成一根颜色为 aba 的竹竿。

假设我们在颜色为 ababa 的竹竿顶端再捆一根竹子，则可以捆成 ababaaba，abababa 和 ababa 三种不同的情况。

但是小 C 在这个问题上有不同的看法，他认为小 W 捆不出很多种长度不同的竹竿。小 W 非常不服，于是他找到了你——现在请你求出在竹竿长度不超过 w 的情况下，小 W 可以捆出多少种**长度不同**的竹竿。其中，竹竿的长度指从底端到顶端的竹子的节的个数。

注意：如果 $w < n$，则没有合法的长度，此时答案为 0。

【输入格式】

输入文件 ***jie.in*** 第一行包含 1 个整数 T，为数据组数。

每组数据的第一行包含 2 个正整数 n 和 w，表示短竹子的长度和竹竿的长度上限。

每组数据的第二行包含一个长度为 n 的字符串，该字符串仅由小写英文字母构成，表示短竹子从底端到顶端每节的颜色。

【输出格式】

输出文件为 ***jie.out***。

输出共 T 行，每行包含一个整数表示捆成竹竿的不同长度种数。

【样例输入 1】

```
1
4 11
bbab
```

【样例输出 1】

```
5
```

【样例解释 1】

可以捆成长度不超过 11 的竹竿有 6 种不同的情况：

bbab

bbabbab

bbabbbab

bbabbabbab

bbabbabbbab

bbabbbabbab

后两种竹竿长度相同，因此不同长度的竹竿共有 5 种。长度分别为：4，7，8，10，11。

【样例输入 2】

```
2
44 1000
baaaaaabaabbaaabbbbabbbaaabbbababaaabaaabaaa
41 1000
abaabbabaaabaabbbbbbbbbbbbbababbbbaaabaabbb
```

【样例输出 2】

```
195
24
```

【样例输入输出 3】

见选手目录下的 ***jie/jie.in*** 与 ***jie/jie.ans***。

【数据规模和约定】

对于所有的测试数据，保证所有的字符串均由小写字母构成。

各测试点满足以下约定：

数据点	T	n	w	约束
1	5	≤ 10	≤ 10	s 仅包含字母 a 和 b
2		≤ 20	≤ 20	
3		≤ 100	$\leq 10^{18}$	无
4				
5		$\leq 10^3$		
6				
7		$\leq 5 \times 10^4$	$\leq 10^5$	
8				
9				
10				
11		$\leq 7 \times 10^4$	$\leq 10^{18}$	
12				
13		$\leq 10^5$		
14				
15				
16				
17		$\leq 5 \times 10^5$		
18				
19				
20				

麈战表达式

【问题描述】

这是一道交互题。

我们需要处理这样一类表达式：它由 n 个元素和 $n-1$ 个运算符构成，两个相邻元素之间有且仅有一个运算符，且表达式中没有括号。例如 $a+b\times c$ 就是一个这样的表达式。

这些运算符一共有 k 种，每种的优先级都不同。为了方便，我们用 1 到 k 的整数来表示这些运算符，并且**数字越大的优先级越高**。

这些运算符都满足交换律和结合律，即对任意的元素 a,b,c 和运算符 $\sim$，满足

$$a\sim b=b\sim a$$

$$a\sim(b\sim c)=(a\sim b)\sim c$$

我们记一开始的表达式为第 0 个版本。

有 m 个操作，每个操作为以下几种之一：

1. 元素修改：对某一个版本，修改某个位置上的元素；
2. 运算符修改：对某一个版本，修改某两个元素之间的运算符；
3. 翻转：对某一个版本，将第 l 个元素到第 r 个元素之间的所有元素（包括第 l 个和第 r 个元素）和运算符翻转。

我们记第 i 次操作之后得到的表达式为第 i 个版本。

在每个操作之后，你需要求出当前表达式的值。

你只能通过调用一个函数 $F(a,b,op)$ 来得到 a op b 的结果，且调用这个函数的次数有一定限制。

【任务描述】

你需要实现以下几个函数：

- $init(test\_id,n,m,k,a,ops)$

  我们首先会调用这个函数。其中：
  - $test\_id$ 为测试点编号，n,m,k 意义见题目描述；
  - a 为一个大小为 n 的数组，$a[i]$ 表示最初表达式里的第 i 个元素的值（下标从 0 开始）；
  - ops 为一个大小为 n 的数组，$ops[i]$ 表示第 i 个元素与第 $i-1$ 个元素之间的运算符；
  - 注意 $ops[0]$ 没有意义，请不要尝试去访问它。

- $modify\_data(id, pos, x)$

  元素修改操作：对第 id 个版本，修改第 pos（$0 \le pos < n$）个元素为 x，并将修改后的表达式作为一个新的版本。你需要返回修改后表达式的值。

- $modify\_op(id, pos, new\_op)$

  运算符修改操作：对第 id 个版本，修改第 pos（$1 \le pos < n$）和第 $pos - 1$ 个元素之间的运算符为 new_op，并将修改后的表达式作为一个新的版本。你需要返回修改后表达式的值。

- $reverse(id, l, r)$

  翻转操作：对第 id 个版本，将第 l 个元素到第 r 个元素之间的所有元素（包括第 l 个和第 r 个元素，$0 \le l \le r < n$）和运算符翻转，并将修改后的表达式作为一个新的版本。你需要返回修改后表达式的值。

其中表达式里的元素（$init$ 函数中 a 数组里的元素，$modify\_data$ 中的 x，和每次返回的表达式的值）均为 $Data$ 类型，这种类型在交互库里提供了定义。

你可以调用一个函数 F 来得到两个元素做某个运算的结果：

- $F(a, b, op)$

  返回将 a, b 两个元素做运算 op（$1 \le op \le k$）之后的值。

请注意：$Data$ 类型中的变量 x 只对交互库有意义，你不需要也不应该访问这个变量。另外请保证传入函数 F 的 a, b 和三种操作函数的返回值为 $init$、$modify\_data$ 或 F 中提供的元素，否则，在使用下发的交互库进行测试时，会发生未知行为（如返回错误的结果，或运行错误等），在最终评测时，该测试点会被判为 0 分。

【实现细节】

你需要且只能提交一个源文件 ***expr.cpp/c/pas*** 实现上述函数，并遵循下面的命名和接口。

对 C/C++语言的选手：

你需要包含头文件 ***expr.h***。

$Data$ 类型的定义：

```
typedef struct {
    int x;
} Data;
```

最终评测时，$Data$ 类型的定义与题面中给出的一样。

你需要实现的函数：

```
void init(
    int test_id, int n, int m, int k,
    const Data *a, const int *ops
);
Data modify_data(int id, int pos, Data x);
Data modify_op(int id, int pos, int new_op);
Data reverse(int id, int l, int r);
```

函数 F 的接口信息如下：

```
Data F(Data a, Data b, int op);
```

你需要在本题目录下使用如下命令编译得到可执行程序：

对于 C 语言

```
gcc grader.c expr.c -o expr -O2 -lm
```

对于 C++语言

```
g++ grader.cpp expr.cpp -o expr -O2 -lm
```

对 Pascal 语言的选手：

你需要使用单元 ***graderhelperlib***。

$Data$ 类型的定义：

```
type Data = record
    x : longint;
end;
```

最终评测时，$Data$ 类型的定义与题面中给出的一样。

你需要实现的函数：

```
procedure init(
    test_id, n, m, k : longint;
    const a : array of Data;
    const ops : array of longint
);
function modify_data(
    id, pos : longint;
    x : Data
) : Data;
function modify_op(
    id, pos, new_op : longint
) : Data;
function reverse(id, l, r : longint) : Data;
```

函数 F 的接口信息如下：

```
function F(
    a, b : Data;
    op : longint
) : Data;
```

你需要在本题目录下使用如下命令编译得到可执行程序：

```
fpc grader.pas -o"expr" -O2
```

【如何开始答题】

本题目录下，有针对每种语言的样例源代码 ***expr_sample.cpp/c/pas***，选择你所需的语言，将其复制为 ***expr.cpp/c/pas***，按照本节前文中提到的方式进行编译，即能通过编译得到可执行程序。

注意：你只能选择一种语言进行作答，即你本题的试题目录下不能同时存在多个语言的 ***expr.cpp/c/pas***。

接下来你需要修改这个文件的实现，以达到题目的要求。

【如何测试你的程序】

交互库将从文件 ***expr.in*** 读入以下格式的数据：

第 1 行包含 4 个整数 $test\_id, n, m, k$，需保证 n, m 不超过 20000，k 不超过 100。

第 2 行仅包含 1 个整数 lim，表示 F 函数的调用次数的限制，需保证 lim 不超过 10^7。

第 3 行包含 $n-1$ 个整数，第 i（$1 \le i < n$）个整数表示第 i 个运算符。

接下来 m 行，每行先是一个整数 id，然后是对第 id 个版本的一个操作。接下来 2 或 3 个整数，描述一个操作。每个操作必须为下列之一：

- 1 pos
  修改第 pos（$0 \le pos < n$）个元素
- 2 pos new_op
  修改第 pos（$1 \le pos < n$）和第 $pos-1$ 个元素之间的运算符为 new_op。
- 3 l r
  将第 l 个元素到第 r 个元素之间的所有元素（包括第 l 个和第 r 个元素）和运算符翻转。

读入完成之后，交互库将调用 $init$ 函数。然后再根据数据调用 m 次 $modify\_data$，$modify\_op$ 或 $reverse$ 函数。最后交互库将会用某种（选手们

不必知道的）方式来计算你的所有函数返回值的**校验和**，并输出到文件 ***expr.out*** 中。交互库还会输出你调用 F 函数的次数。

如果传入 F 函数的参数非法（op 不在 1 到 k 的范围内，或 a,b 不是由交互库提供的元素），那么交互库会将校验和输出为非法值 -1。（因此该测试点得 0 分），然后在下面一行输出错误的详细信息。

如果要使用自己的输入文件进行测试，请保证输入文件按照以上格式，否则不保证程序能正确运行。

【评分方法】

最终评测时只会收取 ***expr.cpp/c/pas***，修改本题目录中的其他文件对评测无效。

题目首先会受到和传统题相同的限制。例如编译错误会导致整道题目得 0 分，运行错误、超过时间限制、超过空间限制等会导致相应测试点得 0 分等。

你只能访问自己定义的和交互库给出的变量及其对应的内存空间，尝试访问其他空间将可能导致编译错误或运行错误。

若程序正常结束，则会开始检验正确性。如果答案不正确，或 F 函数的调用次数超过了测试点的限制，该测试点得 0 分。否则该测试点得满分。

题目中所给的时间、空间限制为你的代码和交互库加起来可以使用的时间和空间。我们保证，对于任何合法的数据，任何语言任何版本的交互库（包括下发给选手的和最终评测时用的），正常运行所用的时间不会超过 0.2s，正常运行所用的空间不会超过 64MB，也就是说，选手实际可用的时间至少为 0.8s，实际可用的空间至少为 960MB。

【样例输入 1】

```
0 5 5 2
1000
2 1 1 1
0 1 1
1 2 2 2
2 3 1 3
0 3 0 3
0 3 1 1
```

【样例输出 1】

```
120400404
```

【样例解释 1】

这是使用试题目录中的 ***grader*** 和正确的源程序得到的输出中的校验和。若你的程序得到了不同的校验和，那么你的程序在本样例下是错误的。

我们用 + 和 × 表示这两种运算符，用小写字母表示元素，则每一个版本的表达式为如下形式：

- 第 0 个版本：$a \times b + c + d + e$
- 第 1 个版本：$a \times f + c + d + e$
- 第 2 个版本：$a \times f \times c + d + e$
- 第 3 个版本：$a \times d + c \times f + e$
- 第 4 个版本：$d + c + b \times a + e$
- 第 5 个版本：$a \times b + c + d + e$

【样例输入输出 2】

见选手目录下的 ***expr/expr.in*** 与 ***expr/expr.ans***。

【数据规模和约定】

各测试点满足以下约定：

$test\_id$	$n=$	$m=$	$k=$	$lim=$	其他约定
1	10	10	1	10^7	无
2	1000	1000	5	10^7	
3	10000	10000	1	10^7	没有 $reverse$ 操作， 第 i 个操作的 id 为 $i-1$
4	20000	20000	1	10^7	
5	10000	10000	1	10^7	第 i 个操作的 id 为 $i-1$
6	20000	20000	1	10^7	
7	10000	10000	1	10^7	无
8	20000	20000	1	10^7	
9	15000	15000	3	10^7	第 i 个操作的 id 为 $i-1$
10	15000	15000	5	10^7	
11	15000	15000	3	10^7	无
12	15000	15000	5	10^7	
13	15000	15000	50	10^7	第 i 个操作的 id 为 $i-1$
14	20000	20000	50	10^7	
15	20000	20000	100	10^7	
16	20000	20000	100	10^7	
17	15000	15000	50	10^7	无
18	20000	20000	50	10^7	
19	20000	20000	100	10^7	
20	20000	20000	100	10^7	