E7 - 2021级程序设计基础第七次练习

A - Sheep的指针

题目背景

*Sheep*在帮同学*debug*的时候曾经碰到了同学的指针逻辑问题，为了帮助同学们更好地理解C语言指针的参数传递，*Sheep*把这个错误提取了出来给同学们分享

*Sheep*给出测试用的这份"奇怪的代码"

```
#include <stdio.h>
char ga[] = "abcdefghijklm";
void my_array_func(char ca[10])
{
    printf(" addr of array param = %#x \n", &ca);
    printf(" addr (ca[0]) = %#x \n", &(ca[0]));
    printf(" addr (ca[1]) = %#x \n", &(ca[1]));
    printf(" ++ca = %#x \n\n", ++ca);
}
void my_pointer_func(char *pa)
{
    printf(" addr of ptr param = %#x \n", &pa);
    printf(" addr (pa[0]) = %#x \n", &(pa[0]));
    printf(" addr (pa[1]) = %#x \n", &(pa[1]));
    printf(" ++pa = %#x \n", ++pa);
}
int main()
{
    printf(" addr of global array = %#x \n", &ga);
    printf(" addr (ga[0]) = %#x \n", &(ga[0]));
    printf(" addr (ga[1]) = %#x \n\n", &(ga[1]));
    my_array_func(ga);
    my_pointer_func(ga);
    return 0;
}
```

*Sheep*在终端运行得到的结果如下

```
/*
输出结果如下：
% test.out
    addr of global array = 0x403010
    addr (ga[0]) = 0x403010
    addr (ga[1]) = 0x403011
    addr of array param = 0x62fe00
    addr (ca[0]) = 0x403010
    addr (ca[1]) = 0x403011
    ++ca = 0x403011
    addr of ptr param = 0x62fe00
    addr (pa[0]) = 0x403010
```

```
    addr (pa[1]) = 0x403011
    ++pa = 0x403011
*/
```

所以奇怪的现象是，数组参数的地址和数组参数的第一个元素的地址竟然不一样？

更奇怪的是，$Sheep$本来用的是64位环境写代码，但是他惊奇的发现这段代码在32位和64位环境下的运行结果还不一样？

题目描述

$Sheep$希望同学们探索一下为什么会出现数组参数的地址和数组参数的第一个元素的地址不一样的情况，所以需要同学们更改上述的代码后输出，让上述代码中输出`数组参数的地址`的位置变为输出`数组参数的第一个元素的地址`。（需要更改的地方**小于**3处，每处更改只涉及一个字符的删除或添加或替换）

这段代码还有一个有趣的点，为了给同学们更好地测试，所以在更改完上述的代码输出后，需要同学们重新读取上述代码中的ga字符串，并输出$sizeof(ga), sizeof(ca), sizeof(pa)$的值

注意，**本题的输出仅供格式示范，答案并不正确**

输入描述

一行一个字符串s，保证字符串s长度$length(s)$满足$1 \leq length(s) \leq 100$

输出描述

先输出上面代码更改后的代码

然后输出三行三个整数，分别为$sizeof(ga), sizeof(ca), sizeof(pa)$

注意这两部分输出之间有一行空行，可以参考样例。

所有输出均为64位环境下的结果。

样例输入

```
abcdefghijklmn
```

样例输出

```
#include <stdio.h>

char ga[] = "abcdefghijklm";

void my_array_func(char ca[10])
{
    printf(" addr of array param = %#x \n", &ca);
    printf(" addr (ca[0]) = %#x \n", &(ca[0]));
    printf(" addr (ca[1]) = %#x \n", &(ca[1]));
    printf(" ++ca = %#x \n\n", ++ca);
}

void my_pointer_func(char *pa)
{
    printf(" addr of ptr param = %#x \n", &pa);
```

```
    printf(" addr (pa[0]) = %#x \n", &(pa[0]));
    printf(" addr (pa[1]) = %#x \n", &(pa[1]));
    printf(" ++pa = %#x \n", ++pa);
}

int main()
{
    printf(" addr of global array = %#x \n", &ga);
    printf(" addr (ga[0]) = %#x \n", &(ga[0]));
    printf(" addr (ga[1]) = %#x \n\n", &(ga[1]));
    my_array_func(ga);
    my_pointer_func(ga);
    return 0;
}

2
5
6
```

HINT

注意字符的转义，注意换行

注意输出的代码字符串为原代码字符串

$Stockholm$的灵魂三问：

- 我们都知道，对一个非指针型变量（如`int x`）取地址（`&x`），得到的是这个变量在内存中的地址，即存储的位置；那么如果我们对一个指针型变量（如`int *x`）或者一个数组的变量名（如`int x[10]`）取地址（`&x`），这会得到什么呢？
- 再者，我们对一个非指针型变量（如`int x`）取大小（`sizeof(x)`），得到的是这个变量在内存中的占据的空间，即存储空间的大小；我们对一个实参数组的变量名（如`int x[10]`）取大小（`sizeof(x)`），得到的是整个数组在内存中占用的空间大小；那么如果我们对一个指针型变量（如`int *x`）或者形参数组（如`void f(int x[])`）取大小（`sizeof(x)`），这又会得到什么呢？
- 又有甚者，大家初次接触形参和实参这两种不同的变量定义方式：
    - 形参：形参变量只有在函数被调用时才会分配内存，调用结束后，立刻释放内存，所以形参变量只有在函数内部有效，不能在函数外部使用；
    - 实参：实参可以是常量、变量、表达式、函数等，无论实参是何种类型的数据，在进行函数调用时，它们都必须有确定的值，以便把这些值传送给形参，所以应该提前用赋值、输入等办法使实参获得确定值；

对这两种不同的数据类型取大小（`sizeof(x)`），为什么会有天壤之别呢？

建议同学们自己复制上述代码在自己的电脑上跑一跑并思考一下为什么会这样，会对同学们后续对数组/指针参数的传递的理解有很大帮助；如果想要更深入地理解，不妨问问度娘。

至于题目代码中数组`ca[10]`为什么定义的时候只开了10但是仍然没有问题的情况，同学们可以往形参实参的区别这方面思考。

AUTHOR: Oh so many sheep & Stockholm

B - Monica的比较

题目描述

我们知道我们可以使用`strcmp`函数判断两个字符串是否相等，但是这个函数不会给出在不相等情况下，两个字符串具体有多少位不相同。

现在给出两个字符串，将它们左对齐后，你需要统计出有多少位上对应字符不同。

对于长度不同的两个字符串，在较短的字符串后面补齐空字符，具体可见样例。

输入描述

第一行一个整数n，表示数据组数。

接下来$2n$行每行一个字符串，表示要判断的字符串。

其中第$2k-1$和第$2k$行的字符串为同一组数据。$(k=1,2,\cdots,n)$

输出描述

输出共n行。

对于每组数据输出一个整数，表示两个字符串不相同的位数。

样例输入

```
3
izeely
iwee
WewillletthemoonandthestarsShowuswherewearetonight
wewillletthemoonandthestarsshowuswherewearetonight
309
9010
```

样例输出

```
3
2
3
```

样例解释

第一组数据，z和w不同，第二个字符串比第一个字符串少两位，可以直接认为是不同的两位。

数据范围

对于100%的数据

字符串长度不超过100，数据组数不超过10组。

字符串中仅包含数字和字母。

AUTHOR: Monica

C - 地址

题目描述

现在已经声明了一个大小为 100 的数组 a，即 `int a[100];`。假设数组的起始地址位置为 `0x00000000`。

现在给定一串字符串，询问这个字符串所对应的地址（用十六进制表示）。

输入描述

本题有多组输入

对于每组输入，输入一行字符串。

输出描述

对于每一组数据，输出一行，为该字符串对应的地址，用形如 `0x0012abcd` 的十六进制数表示。

样例输入

```
a
a+6
&a
&a+6
&a[10]
&a[10]+6
```

样例输出

```
0x00000000
0x00000018
0x00000000
0x00000960
0x00000028
0x00000040
```

数据范围

字符串长度不超过 100。

输入的字符串有且仅有以下几种形式。（其中，$0 \leq x, y \leq 99$）

输入
`&a`
`&a[x]`
`&a[x]+y`
`a`
`a+y`
`&a+y`

HINT

可以自己尝试下不同字符串对应的地址

Author：*Blore*

D - 厉害的指针数组

题目描述

给定n个字符串和m个操作

字符串分别为$A_1, A_2, \cdots, A_N$。

现决定调整它们的次序，对每一个操作，给定两个整数l_1, l_2，表示将字符串A_{l_1}, A_{l_2}交换顺序。

大家去看看HINT。

输入描述

第一行两个整数，分别为n和m。其中$n \leq 100, m \leq 100$。

接下来n行，每行一个字符串，从前到后分别为$A_1, A_2, \cdots, A_N$。（字符串中含有空格，字符串长度不大于100）

接下来m行，每行两个用空格分开的整数，分别为l_1, l_2。

输出描述

共n行，表示$A_1, A_2, \cdots, A_N$。

样例输入

```
3 1
i like programming
i love buaa
stay hungry stay foolish
1 3
```

样例输出

```
stay hungry stay foolish
i love buaa
i like programming
```

HINT

这一题当然可以开一个char型二维数组存下所有的数据，然后通过交换数组中的值来改变字符串次序。但是这样做效率很低。

正如这道题的题目一样，希望同学们可以使用指针数组来A掉这题，体会指针的妙用。可以定义指针数组$lst[]$。

```
char *lst[105];
```

$lst[i]$指向字符串A_i的地址，当需要交换字符串次序时，可以直接修改指针数组的值，这比移动整个字符串要高效得多。

为了避免gets读入空行，建议使用如下读入方式

```
scanf("%d%d ",&n,&m);//第二个%d后面有一个空格
for(int i=1;i<=n;i++)
    gets(a[i]);
```

AUTHOR：Mentor_D

E - Count Sort!

题目背景

给出一个数组比如：`{1,4,1,5,4,1}`，当对其进行排序时，朴素的想法是直接用常用的排序方式来排序（快排、冒泡、插入、选择……）

本题将介绍一种针对数据重复多，值得范围相对较小得排序方式：计数排序；即统计每个数出现次数，再从小到大遍历范围内所有的数，如果某个数出现大于0次，输出相应次数，会惊喜地发现输出结果就是排序结果。

例子当中，从1遍历到5，先输出3个1，再输出2个4，最后输出1个5，1出现了3次，4出现了2次，5出现了1次，得到排序结果：`{1,1,1,4,4,5}`

题目描述

读入一个空格分隔的数组A，输出排序后的结果，为了避免大家用`qsort()`偷懒，除了从小到大输出排序后数组元素a之外，还要输出每个数组元素出现的次数c。

输入描述

输入第一行一个整数N，代表数组长度。

接下来一行N个整数，表示数组A的每个元素a_i。

输出描述

排好序的数组A'从小到大输出所有不重复元素a'和出现次数c'，用冒号分隔，每个数组元素占一行。

样例输入

```
6
1 4 1 5 4 1
```

样例输出

```
1:3
4:2
5:1
```

数据范围

数组元素大小范围：$1 \leq a \leq 10^4$

数组长度范围：$1 \leq N \leq 10^4$

HINT

也许可以不用数组存放数组元素，而是读入一个元素直接计数。

AUTHOR: Shederay

F - 指针判断

题目描述

一个指针所指向的类型可以是**基本类型**、**数组类型**、**指针类型**、**函数类型**等。

- 基本类型：
  - 形如 `char *chPtr` ，称 `chPtr` 的类型为 `char*`，所指向的类型是 `char`
  - 只出现 $int, double, char$
- 数组类型：
  - 形如 `int (*arPtr)[10]`,称 `arPtr` 的类型为 `int(*)[10]`，所指向的类型是 `int[10]`
  - 保证数组的元素类型一定是**基本类型**，比如不会出现 `int* (*p)[10]` （p指向指针数组）
  - 可能出现多维数组的指针，比如 `char (*p)[2][10][10]`
- 指针类型:
  - 形如 `int* *ptrPtr`，称 `ptrPtr` 的类型为 `int**`，所指向的类型是 `int*`
  - 保证该指针类型所指向的类型一定是**基本类型**，比如不会出现 `int** *p` （p指向指针的指针）
- 函数类型：
  - 形如 `double (*funcPtr)(int, int)`，称 `funcPtr` 的类型为 `double(*)(int,int)`，所指向的类型是 `double(int, int)`

- 保证函数的返回值和参数类型一定是**基本类型**，比如不会出现 `int* (*f)()`，`void (*g)(int*, int[])`
- 函数参数数量一定是有限的（可能为0）

现给你多个不同类型指针的声明，请你分别输出指针的类型、指针名和所指向的类型。

输入描述

多行输入，每行一个字符串，为指针的声明（指针名中只含有英文小写字母），中间**不含有空格**

输出描述

对于每行输入，输出一行，格式为 `<pointerType> <pointerName> -> <elemType>`

其中，`<pointerType>` 表示指针类型，`<pointerName>` 是指针名，`<elemType>` 是所指向的类型（均**不含有空格**）

样例输入

```
double*p
int(*ptr)[12]
char**q
int(*fp)(char,int)
```

样例输出

```
double* p -> double
int(*)[12] ptr -> int[12]
char** q -> char*
int(*)(char,int) fp -> int(char,int)
```

数据范围

数据组数 $\leq 10^4$，每行字符串长度 ≤ 100

AUTHOR: toush1

G - good->perfect

题目描述

程小设获得了一个神奇魔法。该魔法可以将一个字符串中"好"的片段转化为`perfect`字符串。

"好"的片段定义为：形如**字母**`g`**+两个及以上字母**`o`**+字母**`d`,且中间没有其他字符的片段。比如`good` `gooood`是"好"的片段，`god` `gooad`不是"好"的片段。

请编程告诉他，如果使用魔法，将一个字符串所有"好"的片段转化为`perfect`字符串，最后的字符串应该是什么样子。

输入描述

多行输入，每行一个字符串，**仅包含小写字母和空格。**

输出描述

对于每行输入，输出一行，为使用魔法之后的字符串。

样例输入

```
good good dogfood
go od do og fo od
gggooooooooooooddd
```

样例输出

```
perfect perfect dogfood
go od do og fo od
ggperfectdd
```

数据范围

行数不超过50行，字符串长度$|S|$满足$1 \leq |S| \leq 50$。

输入仅包含小写字母，空格，及行末换行符`\n`。

AUTHOR: inf

H - 变0为1

题目描述

有一个 $n \times m$ 的网格，每个格子有一个权值，权值只可能是 0 或 1 。我们称每个格子的权值为该**格子的值**。

如果在时刻 t，某个格子的值为 1，那么在时刻 $t+1$，它会将与它相邻的所有格子的值变为 1 。显然，如果最开始有至少一个格子的值为 1，那么在无限长的时间后，所有格子的值都会变为 1 。

现在告诉你时刻 0 时有 k 个格子的值为 1，并且给出他们的坐标（相对应的，没有给出坐标的其他格子的值都为 0），接下来有 T 个询问，每次询问一个网格的坐标，请你回答该格子的值最早在什么时刻变为 1 。

输入描述

第一行四个用空格分开的整数 $n\ m\ k\ T$，含义见上。

接下来 k 行，每行两个整数 $x\ y$，表示最开始值为 1 的格子的坐标。

接下来 T 行，每行两个整数 $a\ b$，表示询问网格的坐标。

输出描述

共 T 行，第 i 行一个正整数，表示第 i 个询问格最早变为 1 的时刻。

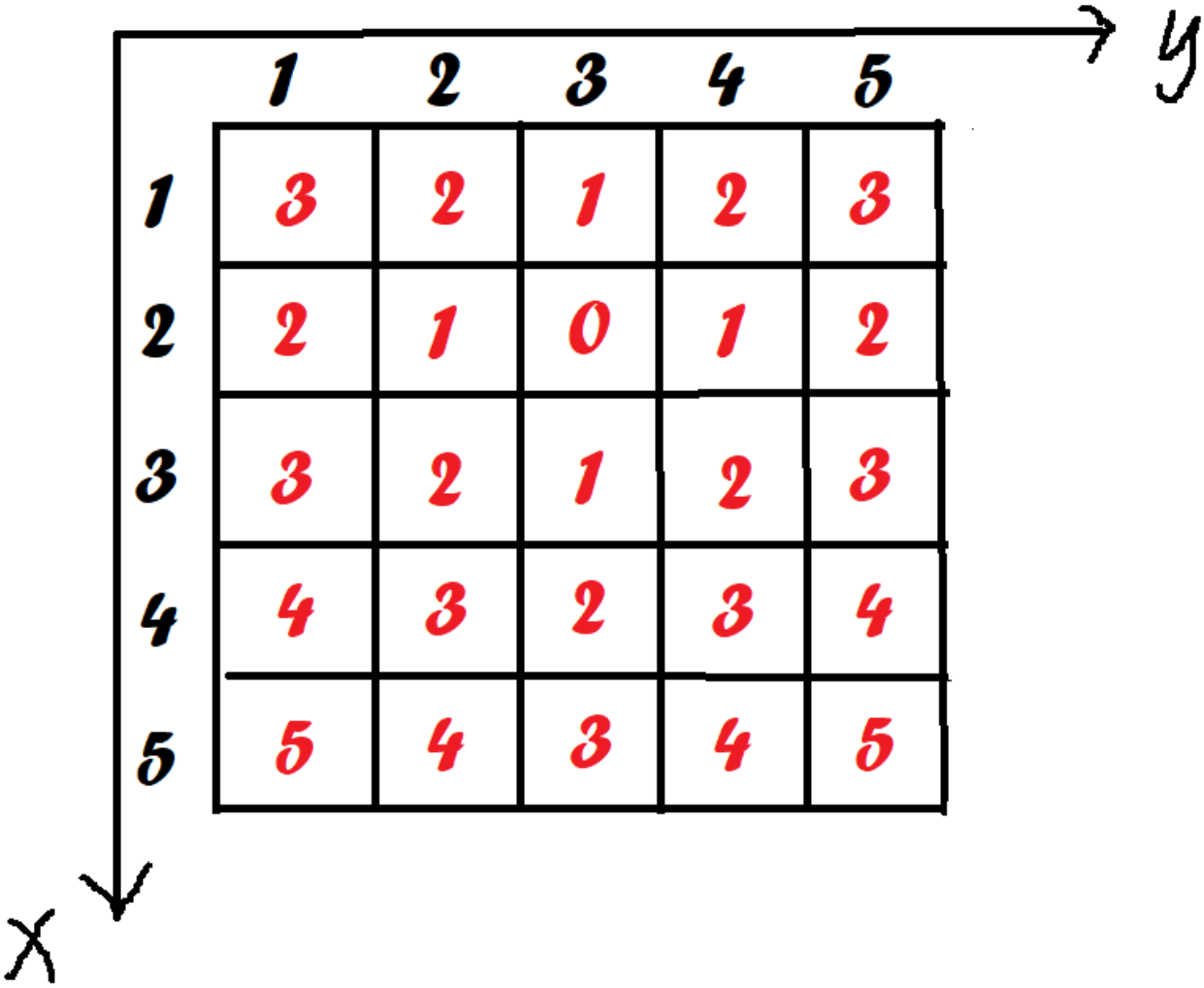
样例输入

```
5 5 1 3
2 3
1 5
5 1
2 3
```

样例输出

```
3
5
0
```

样例解释



如图，格中的红色数字表示该格最早变为 1 的时刻。

数据范围

对于 99% 的数据，$n, m, k, T \leq 100$。

对于 100% 的数据，$n, m \leq 10^9, k \leq 1000, T \leq 10000$ 。

HINT

曼哈顿距离了解一下~

Author : yzh

I - Special Judge

题目背景

各位还记得E5-J 黄金分割 中特殊的评测方式吗?

题目描述

程设助教秋月在上周出了一道超难超难的题目，而且只给出了两组测试数据。杉叶嘉觉得秋小月这题也太毒瘤了，根本没法debug，于是她逼迫秋月更改了评测机的给分方式，具体规则如下：

在提交的程序的输出文件和答案的输出文件进行比较时：

1. 如果程序输出比答案输出长（行数多），那么这个数据点得0分。
2. 对于某个数据点，如果程序输出的**第一个错误**出现在第$k+1$行，而这个输出答案文件总共有n行，那么在这个数据点，这个程序可以得到k/n的分数。
3. 如果程序输出文件比答案输出文件短（行数少），那么依然依照2处的说明给分（如果答案的某行有值，而程序输出的某行为空，这行算作错误输出）。
4. 比较时忽略行末的空格，以及每份文件最后的空行。

输入描述

输入若干行字符串，表示两份“文件”。每份“文件”由两部分组成，第一行为文件名，接下来若干行连续字符串为文件内容。

首先输入程序输出文件，其文件名为`prog.out`

然后输入答案输出文件，其文件名为`ans.out`

整个输入部分以一行`--END--`结束，这行不算作任何文件的文件名或内容。

输出描述

输出两行，第一行是一个在$[0,1]$中的保留四位小数的浮点数，表示得分。

第二行表示评测信息，有以下4种：

- `[Wrong Answer]: Your output is too long.`：表示程序输出比答案输出长（行数多），得0分。
- `[Wrong Answer]: On line %d column %d, read %c, expect %c.`：表示程序输出的第一个错误出现在第几行（line）的第几个字符（column），`read`后面的`%c`对应程序输出，`expect`后面的`%c`对应答案输出。如果程序输出或者答案输出的这个位置为空，那么`%c`的对应位置输出`?`
- `[Wrong Answer]: Your output is too short.`：表示程序输出比答案输出短（行数少），但是程序输出了的每一行都是正确的。
- `[Accepted]`：表示程序得分为1，完全正确。

样例输入1

```
prog.out
1
```

```
2
3
4
5
6
16
8
9
10
ans.out
1
2
3
4
5
6
17
8
9
10
--END--
```

样例输出1

```
0.6000
[Wrong Answer]: On line 7 column 2, read 6, expect 7.
```

样例输入2

```
prog.out
1
```

```
ans.out
1
--END--
```

样例输出2

```
1.0000
[Accepted]
```

样例输入3

```
prog.out
1
111aaas3455gg
ans.out
1
1
--END--
```

样例输出3

```
0.5000
[Wrong Answer]: On line 2 column 2, read 1, expect ?.
```

数据范围

对于所有数据：每行不超过100个字符，每个输入不超过200行。

所有"文件"均由若干行连续的字符串组成，这些字符串中只有大小写字母和数字。

每行的中间没有空格,只有行末可能会有空格。

本题的所有输入数据文件中不存在`'\r'`符号。

所有文件没有中间空行，只有文件末尾可能会有空行。

AUTHOR: 梁秋月

J - 工科元素分析

题目背景

在提瓦特大陆上有七种元素：火、水、风、雷、草、冰、岩。

绝大部分不同元素两两之间可以互相发生反应，反应类型如下表：

元素反应	火	水	风	雷	草	冰	岩
火	无	蒸发	扩散	超载	燃烧	融化	结晶
水	蒸发	无	扩散	感电	无	冻结	结晶
风	扩散	扩散	无	扩散	无	扩散	无
雷	超载	感电	扩散	无	无	超导	结晶
草	燃烧	无	无	无	无	无	无
冰	融化	冻结	扩散	超导	无	无	结晶
岩	结晶	结晶	无	结晶	无	结晶	无

在本题中，暂不考虑一次超载两次伤害、强水可蒸发两次等等条件，不考虑控制状态等等结果，不考虑暴击率和暴击效果的影响，不考虑怪物的防御力和元素抗性等数值的影响，在仅考虑单次正常伤害的条件下，**简化**元素反应机制，做出如下规定：

增幅反应：

1. 蒸发：不造成额外伤害，加强本次攻击伤害。
2. 融化：不造成额外伤害，加强本次攻击伤害。

聚变反应：

3. 超载：造成额外伤害，不加强本次攻击伤害。
4. 扩散：造成额外伤害，不加强本次攻击伤害。
5. 燃烧：造成额外伤害，不加强本次攻击伤害。
6. 感电：造成额外伤害，不加强本次攻击伤害。
7. 超导：造成额外伤害，不加强本次攻击伤害。

无伤反应：

8. 冻结：不造成额外伤害，不加强本次攻击伤害。
9. 结晶：不造成额外伤害，不加强本次攻击伤害，但生成护盾。

题目描述

初始数据：每个角色的等级为`Level`，攻击力（又称“白值”）为`ATK`，防御力为`DEF`，元素精通为`Elemental_Mastery`，对每种元素都有一个元素伤害加成百分比Δ，**每种元素的伤害加成不一定相等。**

前四个变量均为自然数，后面的变量为实数。

原伤害计算公式：

$$原伤害 = ATK \times (1 + \Delta_{原本元素})$$

元素精通对**增幅反应**的增伤百分比λ_1计算公式：

令α, β分别为$Elemental_Mastery \div 80$的商和余数，则有：

$$增伤系数\lambda_1 = (2.5 - （1.5 - \frac{0.15 \times \beta}{80}) \times 0.9^{\alpha}) \times 100\%$$

元素精通对**聚变反应**的增伤百分比λ_2计算公式：

$$增伤系数\lambda_2 = 2.4 \times 25 \times \frac{Elemental\_Mastery}{9 \times (Elemental\_Mastery + 1400)} \times 100\%$$

元素精通对结晶生成的**元素护盾强度**的增加百分比λ_3计算公式：

$$增强系数\lambda_3 = \frac{4.44}{1 + \frac{1400}{Elemental\_Mastery}} \times 100\%$$

特例：当`Elemental_Mastery==0`时，此式取极限$\lim(Elemental_Mastery) \to 0$。

每种元素反应伤害的计算方式如下：

1. 增幅反应：
    - 蒸发：以一定比例提升本次攻击伤害。
        - 当第一次附着的元素是水元素时，火元素去打水元素，造成原本火元素伤害150%的火元素伤害，无额外伤害；
        - 当第一次附着的元素是火元素时，水元素去打火元素，造成原本水元素伤害200%的水元素伤害，无额外伤害。
    - 融化：以一定比例提升本次攻击伤害。

- 当第一次附着的元素是冰元素时，火元素去打冰元素，造成原本火元素伤害200%的火元素伤害，无额外伤害；
- 当第一次附着的元素是火元素时，冰元素去打火元素，造成原本冰元素伤害150%的冰元素伤害，无额外伤害。

增幅反应原伤害的计算公式：
$$\text{原本元素伤害} = ATK \times (1 + \Delta_{\text{原本元素}}) \times \lambda_1$$

2. 聚变反应：
    - 超载：原伤害不变，额外造成一次一定量的火元素伤害。
    - 扩散：原伤害不变，额外造成一次一定量的对应元素伤害。

      对应元素：当风元素遇到`火/水/雷/冰`元素时将触发扩散反应，将`火/水/雷/冰`元素扩散到四周，这时称`火/水/雷/冰`元素为对应元素。
    - 燃烧：原伤害不变，额外造成一次一定量的火元素伤害。
    - 感电：原伤害不变，额外造成一次一定量的雷元素伤害。
    - 超导：原伤害不变，额外造成一次一定量的冰元素伤害。

    聚变反应的基础伤害与角色等级有关：

$Level \in$	$[1,20]$	$[21,30]$	$[31,40]$	$[41,50]$	$[51,60]$	$[6170]$	$[71,90]$
基础伤害	36.3	61.3	92.9	145	221	324	521

    聚变反应的反应系数：

元素反应	燃烧	超导	扩散	超载	感电
反应系数	0.4	1	1.2	4	4.8

    对于燃烧反应，反应伤害计算公式为：
    $$\text{反应伤害} = \text{基础伤害} \times \text{反应系数}$$
    对于扩散反应，反应伤害计算公式为：
    $$\text{反应伤害} = \text{基础伤害} \times \text{反应系数} \times (1 + \lambda_2) \times (1 + \Delta_{\text{对应元素}})$$
    对于其他聚变反应，反应伤害计算公式为：
    $$\text{反应伤害} = \text{基础伤害} \times \text{反应系数} \times (1 + \lambda_2)$$

3. 无伤反应：
    - 冻结：保持原伤害不变，无额外伤害。
    - 结晶：不造成额外伤害，不加强本次攻击伤害，但生成对应元素的护盾。

      元素护盾的基础强度与角色等级有关：

$Level \in$	$[1,30]$	$[31,40]$	$[41,50]$	$[51,60]$	$[61,70]$	$[71,80]$	$[81,90]$
基础强度	303.8	585	786.8	1030.1	1314.8	1596.8	1851.1

      护盾具有强度，与角色的防御力有关，计算公式为：
      $$\text{强度} = DEF + \text{基础护盾强度} \times (1 + \lambda_3)$$

给出角色的基本数值，询问元素反应，具体方式见输入、输出格式。

输入描述

为简化题目，本题使用数字来对应每种元素：

数字	1	2	3	4	5	6	7
元素	火	水	风	雷	草	冰	岩

第一行一个正整数q，表示询问的次数。

每次询问的格式如下：

第一行四个自然数$Level, ATK, DEF, Elemental_Mastery$。

第二行有七个非负实数，分别表示每种元素的伤害加成百分比$\Delta_i, i \in [1, 7]$（**注意是百分制**，输入无百分号$\%$）。

第三行两个正整数u, v，分别表示附着在怪物上的元素对应数字u和角色攻击打出伤害的元素对应数字v。

输出描述

输出q行，每行对应着一次元素反应，格式如下：

- 若无元素反应，输出一个数表示原伤害；
- 对于**增幅反应**，输出一个数表示总伤害；
- 对于**聚变反应**，输出两个数，**先输出原伤害，再输出反应伤害**，两个数间用一个空格隔开；
- 对于**无伤反应**：
    - 当元素反应是冻结时，输出`"Frozen"`，不包含引号；
    - 当元素反应是结晶时，输出一个数表示护盾强度。

所有输出的数字均四舍五入为整数。

样例输入

```
3
90 1511 1136 19
61.6 0 0 0 0 0 0
2 1
90 1015 1211 220
15 0 0 0 0 0 0
4 1
90 4333 1136 19
102.1 0 0 0 0 0 0
6 1
```

样例输出

```
3793
1167 3971
18138
```

样例解释

（样例输入数据来源于胡桃实机测试）

第一组数据中90级胡桃攻击力1511，防御力1136，元素精通19，增伤系数$\lambda_1 = 103.5625\%$，只有61.6%的火元素伤害加成，用火元素打水元素，蒸发伤害加成为150%，故总伤害为：
$1511 \times (1 + 61.6\%) \times 103.5625\% \times 150\% = 3793.146405 \approx 3793$

第二组数据中90级胡桃攻击力1015，防御力1211，元素精通220，增伤系数$\lambda_2 \approx 90.5349794\%$，只有$15\%$的火元素伤害加成，用火元素打雷元素，超载伤害反应系数为4，基础伤害为521，故原伤害不变：
$1015 \times (1 + 15\%) = 1167.25 \approx 1167$

额外的超载伤害为：
$521 \times 4 \times (1 + 90.5349794\%) = 3970.7489707 \approx 3971$

第三组数据中90级胡桃（开E技能后）攻击力4333，防御力1136，元素精通19，增伤系数$\lambda_1 = 103.5625\%$，只有102.1%的火元素伤害加成，用火元素打冰元素，融化伤害加成为200%，故总伤害为：
$4333 \times (1 + 102.1\%) \times 103.5625\% \times 200\% = 18137.9217513 \approx 18138$

数据范围

对于20%的数据：$q \leq 10$；

对于100%的数据：$q \leq 10^5$。

100%的数据均保证：

$1 \leq Level \leq 90, 100 \leq ATK, DEF \leq 10000, 0 \leq Elemental\_Mastery \leq 1000, 1 \leq u, v \leq 7$

$\forall i \in [1, 7], i \in \mathbb{N}, 0\% \leq \Delta_i \leq 200\%$

HINT

你可能需要用到`<math.h>`当中一些函数：

- `pow(a,b)`：求a^b；
- `round(x)`：求x的四舍五入结果。

AUTHOR: Stockholm