A 云计算

时空限制：2s 256MB

文件名

cloud.in/cloud.out/cloud.cpp

题目描述

Johnny 创立了 Bytecomp，一家提供云计算能力的公司。这样的公司通常拥有许多快速的计算机，可以让客户在上面进行计算。

Johnny 仍然没有购买任何机器。他去了一家计算机商店，收到了一个包含所有 n 台可用的计算机的清单。每台计算机都可以用三个属性描述：处理器内核数 c_i，时钟频率 f_i 和价格 v_i。这样的计算机包含 c_i 个独立的内核，所以他们可以被分配不同的任务。

当客户订购资源时，她会指定所需的内核数 C_j 和最小的时钟频率 F_j。订单还包含客户愿意支付的价格 V_j。如果接受订单，Bytecomp 需要提供客户所需计算能力的独占访问权。Johnny 需要选择 C_j 个核心（可能来自不同的计算机），且它们的时钟频率至少为 F_j，这些核心不能被分配给其它订单。

帮助 Johnny 赚尽可能多的钱：接受一个最优的订单子集，选择所有计算机的一个子集来满足所有接受了的订单。你的目标是最大化利润，即为客户提供计算能力的收入与购买计算机的成本之差。

输入格式

标准输入的第一行包含一个整数 n，表示商店中可用计算机的数量。

接下来 n 行，每行描述一台计算机，包含三个空格隔开的整数 c_i, f_i, v_i，分别表示内核数，时钟频率和价格。

接下来一行一个整数 m，表示客户的订单数量。

接下来 m 行，每行描述一个订单，包含三个空格隔开的整数 C_j, F_j, V_j，分别表示需要的核心数，最低的允许的时钟频率以及预算。

输出格式

标准输出唯一的一行包含一个整数，表示能够得到的最大利润。

样例 #1

样例输入 #1

```
4
4 2200 700
2 1800 10
20 2550 9999
4 2000 750
3
1 1500 300
6 1900 1500
3 2400 4550
```

样例输出 #1

```
350
```

提示

样例解释

一共有四台可用的计算机和三个订单。

最佳方案是购买两台价格为 700 和 750（总计 1450）的四内核的计算机，然后接受前两个订单获得 $300 + 1500 = 1800$ 的收益。

我们获得了四个时钟频率为 2000 的内核，和四个时钟频率为 2200 的内核。可以将其中六个分配给第二个订单（需要 1900 的时钟频率），再将其中一个分配给第一个订单（需要 1500 的时钟频率），剩下一个核心不使用，这是允许的。

总利润为 $1800 - 1450 = 350$。

数据规模与约定

对于 100% 的数据，$1 \le n, m \le 2 \times 10^3$，$1 \le c_i, C_i \le 50$，$1 \le f_i, F_i, v_i, V_i \le 10^9$。

所有测试数据被划分成若干个有附加限制的子任务，每个子任务中包含若干测试点。

子任务	附加限制	分值
1	$n \le 15$	18
2	$m \le 15$	18
3	$n, m \le 100$，$c_i = C_j = 1$	18
4	$f_i = F_j = 1$	18
5	$v_i = V_j = 1$	18
6	无附加限制	10

B 预处理器

时空限制：1s 512MB

文件名

preprocessor.in/preprocessor.out/preprocessor.cpp

题目描述

宏是 C/C++ 语言的一项特性，它根据预先定义的规则进行文本替换（也被称为 “宏展开”），能够实现定义常量、简化代码重复输入等功能。例如：

```
#define PI 3.14159
double area = PI * r * r;
```

以上代码经过宏展开后变为：

```


double area = 3.14159 * r * r;
```

其中，宏定义命令变成了空行，而其他行中的宏被展开成了规则定义的文本。

C/C++ 语言代码在编译时对宏的处理由**预处理器**完成。你的任务是实现一个简化版的预处理器，要求如下：

- 代码由**行**组成，每行除行末的换行符外，均由可打印 ASCII 字符（ASCII 码范围 $32 \sim 126$）组成。每行要么是**预处理命令**（以 `#` 开头），要么是**普通文本**（其他情况）。
- 预处理器逐行处理代码，
  - 如果是预处理命令，执行该命令，并输出一个空行。
    - 如果是普通文本，对其进行宏展开并输出结果。
- 预处理命令有两种，分别是宏定义命令 `#define` 和取消宏定义命令 `#undef`。
  - 宏定义命令的格式为 `#define <name> <content>`，其中第一部分 `#define` 是命令名，第二部分 `<name>` 是要定义的宏的名字，第三部分 `<content>` 是要定义的宏的展开内容。
    - 取消宏定义命令的格式为 `#undef <name>`，其中第一部分 `#undef` 是命令名，第二部分 `<name>` 是要取消的宏的名字。

    以上两种预处理命令中，相邻两部分之间都严格用一个空格分隔。`<name>` 是由大小写字母和数字以及下划线组成的**标识符**（一个或多个字符），`<content>` 可以包含任意可打印 ASCII 字符（零个或多个字符）。一个宏定义的**有效范围**是从它定义所在行开始到后续最近的宏名匹配的取消定义所在行为止（如果没有对应的取消定义，则有效范围一直覆盖到文件结束）。

对普通文本进行宏展开时，将一行文本中每段**连续极长**的大小写字母和数字以及下划线视为标识符（而不是其中一部分），其余为**其他字符**。从左到右依次对文本中的标识符进行宏展开：

1. 如果该标识符是有效的宏名，则用对应的展开内容替换它，此时该宏名进入正在展开的状态，直到本流程结束；否则原样保留宏名。例如，若宏 `A` 定义为 `b`，则文本 `A` 展开结果为 `b`（发生替换），文本 `B` 展开结果仍然为 `B`（未定义，不替换），文本 `AA` 展开结果仍然为 `AA`（`AA` 是不同于 `A` 的另一个标识符，未定

义），而文本 `A*B` 展开结果为 `b*B`。
2. 替换发生后，如果展开内容中包含标识符，重复应用以上的展开操作，称为 “多次展开”。例如，若宏 `A` 定义为 `B`，宏 `B` 定义为 `c`，则文本 `A` 的展开结果为 `c`。
3. 如果待展开的宏名与正在进行展开的某个宏名相同，称为 “递归展开”，此时该宏名不再展开。本规则用来防止无限递归展开。例如，若宏 `A` 定义为 `B+a`，宏 `B` 定义为 `A+b`，则文本 `A` 展开结果为 `A+b+a`，由于最初的 `A` 处于正在展开状态，因此 `A+b+a` 里的 `A` 不再展开。
4. 其他字符原样保留。

注意：出于简化的目的，本题的要求与 C/C++ 语言标准里的描述不完全一致，请以上面的要求为准。最明显的区别是本题只有标识符和其他字符两类词法单元，没有数值、字符串、注释等。

输入格式

输入的第一行包含一个正整数 n，表示要处理的代码行数。

接下来的 n 行是要处理的代码。

输出格式

输出 n 行，为输入逐行预处理后的结果。

样例 #1

样例输入 #1

```
5
#define BEGIN {
#define END }
#define INTEGER int
class C BEGIN INTEGER x; END;
INTEGER main() BEGIN C c; END
```

样例输出 #1

```
class C { int x; };
int main() { C c; }
```

样例 #2

样例输入 #2

```
见附件中的 preprocessor/preprocessor2.in
```

样例输出 #2

```
见附件中的 preprocessor/preprocessor2.ans
```

样例 #3

样例输入 #3

```
见附件中的 preprocessor/preprocessor3.in
```

样例输出 #3

```
见附件中的 preprocessor/preprocessor3.ans
```

提示

【数据范围】

对 20% 的数据，不会出现宏定义命令 `#define` 和宏取消定义命令 `#undef`。

对另外 20% 的数据，不会出现多次展开的情况，且不会出现宏取消定义命令 `#undef`。

对另外 20% 的数据，不会出现多次展开的情况。

对另外 20% 的数据，不会出现递归展开的情况。

对其余数据，无特殊限制。

对 100% 的数据，$n \leq 100$，输入的每行字符数都不超过 100，且保证输出的每行字符数都不超过 1000（字符数均不计行末换行符）。保证输入数据中的预处理命令都是合法的，包含但不限于：

- `#` 字符只会出现在预处理命令所在行的第一个字符的位置，其他任何位置（包括预处理命令和普通文本）都不会出现 `#` 字符。
- 宏定义和取消定义命令的格式是正确的，严格遵循题面所描述的格式。
- 同一个宏在取消定义之前不会被再次定义。
- 要取消定义的宏在之前被定义过且还没有被取消过。

也就是说，**你不需要做任何语法和语义的错误检查**。

【提示】

本题进行输入时建议使用 C++ 语言的按行读入字符串功能，示例如下：

```cpp
#include <iostream>
#include <string>
using namespace std;
string line;
// 从 cin 读入一行，放入 line 中（换行符被舍弃）
getline(cin, line);
```

也可以使用 C 语言提供的 `fgets` 函数，示例如下：

```
#include <stdio.h>
#define MAX_LEN 200
char line[MAX_LEN];
// 从 stdin 读入一行，放入 line 中（包含换行符）
fgets(line, MAX_LEN, stdin);
```

注意：在读取行数 n 之后可能需要额外读取一行以忽略其后的换行符。

C 吃

时空限制：1s 256MB

文件名

food.in/food.out/food.cpp

题目描述

小 A 很喜欢吃东西。

小 A 面前有 n 份食物，第 i 份有参数 a_i 和 b_i。小 A 可以按照**任意顺序**吃掉这 n 份食物。当她吃掉编号为 i 的食物时，她可以选择将自己的体重乘以 a_i 或者将自己的体重加上 b_i。每份食物只能吃恰好一次。

小 A 的初始体重为 1，请求出她吃完 n 份食物后能达到的**最大**体重。答案可能很大，你只需要输出其对 (10^9+7) 取模后的结果。

注意：你需要最大化体重并将该最大值对 (10^9+7) 取模，而非最大化体重对 (10^9+7) 取模的结果。

输入格式

第一行输入一个整数 n 表示食物的数量。第二行 n 个整数 $a_1, a_2, \ldots, a_n$，第三行 n 个整数 $b_1, b_2, \ldots, b_n$，表示每份食物的参数。

输出格式

输出一个整数，表示小 A 可以得到的最大体重对 (10^9+7) 取模后的结果。

样例 #1

样例输入 #1

```
5
1 2 3 4 5
100 200 300 400 500
```

样例输出 #1

```
18060
```

提示

【样例解释 #1】

以下方案可以达到最大体重：

- 吃掉第一份食物并选择将体重增加 100，体重变为 101；

- 吃掉第二份食物并选择将体重增加 200，体重变为 301；
- 吃掉第三份食物并选择将体重乘 3，体重变为 903；
- 吃掉第四份食物并选择将体重乘 4，体重变为 3612；
- 吃掉第五份食物并选择将体重乘 5，体重变为 18060。

【样例 #2】

见附件中的 `food/food2.in` 与 `food/food2.ans`。

该组样例满足 $n \leq 10$ 和特殊性质 E。

【样例 #3】

见附件中的 `food/food3.in` 与 `food/food3.ans`。

该组样例满足 $n \leq 20$ 和特殊性质 E。

【样例 #4】

见附件中的 `food/food4.in` 与 `food/food4.ans`。

该组样例满足 $n \leq 2000$。

【样例 #5】

见附件中的 `food/food5.in` 与 `food/food5.ans`。

该组样例满足特殊性质 A。

【样例 #6】

见附件中的 `food/food6.in` 与 `food/food6.ans`。

该组样例满足特殊性质 C。

【样例 #7】

见附件中的 `food/food7.in` 与 `food/food7.ans`。

该组样例满足特殊性质 D。

【样例 #8】

见附件中的 `food/food8.in` 与 `food/food8.ans`。

该组样例满足特殊性质 B。

【样例 #9】

见附件中的 `food/food9.in` 与 `food/food9.ans`。

【数据范围】

对于 100% 的测试数据，$1 \leq n \leq 5 \times 10^5$，$1 \leq a_i, b_i \leq 10^6$。

测试点编号	$n \leq$	特殊性质
1	10	DE
2	10	E
3	10	AE
4	10	E
5	20	DE
6	20	E
7	20	E
8	20	E
9	2000	D
10	2000	无
11	2000	无
12	2000	无
13	5×10^5	BD
14	5×10^5	B
15	5×10^5	C
16	5×10^5	C
17	10^5	无
18	10^5	无
19	5×10^5	无
20	5×10^5	无

特殊性质 A：$a_i = 1$。
特殊性质 B：$a_i \geq b_i$。
特殊性质 C：a_i, b_i 在 $[1, 10^6]$ 内独立均匀随机生成。
特殊性质 D：$a_i \geq 2$。
特殊性质 E：$a_i \leq 4$。

D 垃圾回收

时空限制：2s 512MB

文件名

garbage.in/garbage.out/garbage.cpp

题目描述

通常的情况下，编程语言在管理内存时进行如下的选择：

- 让用户进行手动内存管理（C、C++、Rust 等），这会收获很好的性能，但是给用户提供了很大的编程负担。
- 使用垃圾回收系统（Java、Go 等），这需要维护一个运行时系统，并且在内存使用和程序性能方面造成了许多不可预测的负担。

尽管存在许多的问题，目前最通用的自动化内存管理手段始终为 Tracing Garbage Collector。这种做法的最基础的思路是维护对象间的引用关系，形成一张图，每次回收时通过扫描引用关系推导出已经无法被访问到的对象，释放它们占用的内存。而这种传统的做法最大的问题在于维护引用链需要造成很大的开销，并且随着维护的对象越多，扫描的代价也会越大。

小 L 是一个喜欢思考的女孩子，她发现维护 Garbage Collector 是一件非常复杂的事情，于是她决定考虑一个更简单的模型（注意它与任何现实中的 GC 规则可能是完全不同的！）。

对于一个 n 个点 m 条边的无向图，没有重边自环，点和边均从 1 开始标号。其中每个节点代表一个占用了一定内存的对象，每条边对应一个引用关系（注意这里的引用关系是**无向**的），程序从第 0 秒开始运行，在第 $q+1$ 秒结束运行。对于 $i=1,2,3,\ldots,q$ 的每个时刻 i 发生以下两种操作之一：

- DELETE i，删除边 (x_i,y_i)，保证不会删除已经被删除的边。
- GC， 进行一次内存回收，即杀死所有从起点出发不能访问到的点，释放它们占用的内存。（注意这里对节点的删除不会删除与这些点相连的边）

你可以认为这些操作是被瞬间执行完成的，在所有操作执行后，也就是第 $q+1$ 秒，程序结束，删除所有剩余的节点（包括 1 号点）。

第 i 个点占用的内存为 a_i，现在请你求出 $\sum_{i=1}^{n} a_i \cdot alive_i$，这里 $alive_i$ 表示第 i 个点存活的时间，在第 0 秒，所有节点都是存活的。

输入格式

输入的第一行是三个正整数 n,m,q，分别表示对象的个数，引用关系的个数，操作的个数。

接下来的 m 行每行两个正整数 x_i,y_i 表示第 i 条引用关系的两个端点。

接下来的 q 行每行为以下两种形式之一，第 i 行表示在第 i 秒发生的操作：

- 字符串 DELETE 和正整数 x，含义见【题目描述】。
- 字符串 GC，含义见【题目描述】。

接下来的一行有 n 个正整数 $a_1,a_2,\ldots,a_n$，表示每个对象占用的内存大小。

输出格式

输出一行一个整数表示答案，含义见【题目描述】。

样例 #1

样例输入 #1

```
6 6 8
1 2
2 3
2 4
1 4
2 5
1 6
GC
DELETE 5
DELETE 3
GC
DELETE 1
GC
DELETE 2
GC
1 2 3 4 5 6
```

样例输出 #1

```
149
```

样例 #2

样例输入 #2

```
样例 2 见附件 garbage2.in
本组数据满足测试点 6 的限制。
```

样例输出 #2

```
样例 2 见附件 garbage2.ans
```

样例 #3

样例输入 #3

```
样例 3 见附件 garbage3.in
本组数据满足测试点 11 的限制。
```

样例输出 #3

```
样例 3 见附件 garbage3.ans
```

提示

【样例 1 解释】

在第 4 秒时，节点 5 被删除。

在第 6 秒时，节点 $2,3$ 被删除。

在第 9 秒时，节点 $1,4,6$ 被删除。

答案即 $5\times 4+(2+3)\times 6+(1+4+6)\times 9=20+30+99=149$。

【数据规模与约定】

对于全部数据，$1\le n,m,q\le 4\times 10^5$，$1\le a_i\le 10^8$。

具体的数据规模与约定见下表。

测试点编号	n	m	q	特殊限制
$1\sim 2$	≤ 500	≤ 500	≤ 500	
$3\sim 5$	≤ 3000	≤ 3000	≤ 3000	
$6\sim 10$	≤ 5000	≤ 5000	≤ 5000	
$11\sim 14$	$\le 2\times 10^5$	$n-1$	$\le 2\times 10^5$	保证一开始图是一棵树
$15\sim 16$	$\le 2\times 10^5$	$\le 2\times 10^5$	$\le 2\times 10^5$	
$17\sim 20$	$\le 4\times 10^5$	$\le 4\times 10^5$	$\le 4\times 10^5$	