Codeforces Beta Round #81

A. Transmigration

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

In Disgaea as in most role-playing games, characters have skills that determine the character's ability to use certain weapons or spells. If the character does not have the necessary skill, he cannot use it. The skill level is represented as an integer that increases when you use this skill. Different character classes are characterized by different skills.

Unfortunately, the skills that are uncommon for the given character's class are quite difficult to obtain. To avoid this limitation, there is the so-called transmigration.

Transmigration is reincarnation of the character in a new creature. His soul shifts to a new body and retains part of his experience from the previous life.

As a result of transmigration the new character gets all the skills of the old character and the skill levels are reduced according to the k coefficient (if the skill level was equal to x, then after transmigration it becomes equal to $[kx]$, where $[y]$ is the integral part of y). If some skill's levels are **strictly less** than 100, these skills are forgotten (the character does not have them any more). After that the new character also gains the skills that are specific for his class, but are new to him. The levels of those additional skills are set to 0.

Thus, one can create a character with skills specific for completely different character classes via transmigrations. For example, creating a mage archer or a thief warrior is possible.

You are suggested to solve the following problem: what skills will the character have after transmigration and what will the levels of those skills be?

Input

The first line contains three numbers n, m and k — the number of skills the current character has, the number of skills specific for the class into which the character is going to transmigrate and the reducing coefficient respectively; n and m are integers, and k is a real number with exactly two digits after decimal point ($1 \le n, m \le 20$, $0.01 \le k \le 0.99$).

Then follow n lines, each of which describes a character's skill in the form "$name\ exp$" — the skill's name and the character's skill level: $name$ is a string and exp is an integer in range from 0 to 9999, inclusive.

Then follow m lines each of which contains names of skills specific for the class, into which the character transmigrates.

All names consist of lowercase Latin letters and their lengths can range from 1 to 20 characters, inclusive. All character's skills have distinct names. Besides the skills specific for the class into which the player transmigrates also have distinct names.

Output

Print on the first line number z — the number of skills the character will have after the transmigration. Then print z lines, on each of which print a skill's name and level, separated by a single space. The skills should be given in the lexicographical order.

Sample test(s)

input

```
5 4 0.75
axe 350
impaler 300
ionize 80
megafire 120
magicboost 220
heal
megafire
shield
magicboost
```

output

```
6
axe 262
heal 0
impaler 225
magicboost 165
megafire 0
shield 0
```

B. Dark Assembly

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Dark Assembly is a governing body in the Netherworld. Here sit the senators who take the most important decisions for the player. For example, to expand the range of the shop or to improve certain characteristics of the character the Dark Assembly's approval is needed.

The Dark Assembly consists of n senators. Each of them is characterized by his *level* and *loyalty* to the player. The level is a positive integer which reflects a senator's strength. Loyalty is the probability of a positive decision in the voting, which is measured as a percentage with precision of up to 10%.

Senators make decisions by voting. Each of them makes a positive or negative decision in accordance with their loyalty. If **strictly more** than half of the senators take a positive decision, the player's proposal is approved.

If the player's proposal is not approved after the voting, then the player may appeal against the decision of the Dark Assembly. To do that, player needs to kill all the senators that voted against (there's nothing wrong in killing senators, they will resurrect later and will treat the player even worse). The probability that a player will be able to kill a certain group of senators is equal to $A / (A + B)$, where A is the sum of levels of all player's characters and B is the sum of levels of all senators in this group. If the player kills all undesired senators, then his proposal is approved.

Senators are very fond of sweets. They can be bribed by giving them candies. For each received candy a senator increases his loyalty to the player by 10%. It's worth to mention that loyalty cannot exceed 100%. The player can take no more than k sweets to the courtroom. Candies should be given to the senators **before** the start of voting.

Determine the probability that the Dark Assembly approves the player's proposal if the candies are distributed among the senators in the optimal way.

Input

The first line contains three integers n, k and A ($1 \le n, k \le 8$, $1 \le A \le 9999$).

Then n lines follow. The i-th of them contains two numbers — b_i and l_i — the i-th senator's level and his loyalty.

The levels of all senators are integers in range from 1 to 9999 (inclusive). The loyalties of all senators are integers in range from 0 to 100 (inclusive) and all of them are divisible by 10.

Output

Print one real number with precision 10^{-6} — the maximal possible probability that the Dark Assembly approves the player's proposal for the best possible distribution of candies among the senators.

Sample test(s)

input
```
5 6 100
11 80
14 90
23 70
80 30
153 70
```
output
```
1.0000000000
```

input
```
5 3 100
11 80
14 90
23 70
80 30
153 70
```
output
```
0.9628442962
```

input
```
1 3 20
20 20
```
output
```
0.7500000000
```

Note

In the first sample the best way of candies' distribution is giving them to first three of the senators. It ensures most of votes.

It the second sample player should give all three candies to the fifth senator.

C. Item World

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Each item in the game has a level. The higher the level is, the higher basic parameters the item has. We shall consider only the following basic parameters: attack (`atk`), defense (`def`) and resistance to different types of impact (`res`).

Each item belongs to one class. In this problem we will only consider three of such classes: `weapon`, `armor`, `orb`.

Besides, there's a whole new world hidden inside each item. We can increase an item's level travelling to its world. We can also capture the so-called residents in the Item World

Residents are the creatures that live inside items. Each resident gives some bonus to the item in which it is currently located. We will only consider residents of types: `gladiator` (who improves the item's `atk`), `sentry` (who improves `def`) and `physician` (who improves `res`).

Each item has the size parameter. The parameter limits the maximum number of residents that can live inside an item. We can move residents between items. Within one moment of time we can take some resident from an item and move it to some other item if it has a free place for a new resident. We cannot remove a resident from the items and leave outside — any of them should be inside of some item at any moment of time.

Laharl has a certain number of items. He wants to move the residents between items so as to equip himself with weapon, armor and a defensive orb. The weapon's `atk` should be largest possible in the end. Among all equipping patterns containing weapon's maximum `atk` parameter we should choose the ones where the armor's `def` parameter is the largest possible. Among all such equipment patterns we should choose the one where the defensive orb would have the largest possible `res` parameter. Values of the parameters `def` and `res` of weapon, `atk` and `res` of armor and `atk` and `def` of orb are indifferent for Laharl.

Find the optimal equipment pattern Laharl can get.

Input

The first line contains number n ($3 \le n \le 100$) — representing how many items Laharl has.

Then follow n lines. Each line contains description of an item. The description has the following form: "$name\ class\ atk\ def\ res\ size$" — the item's name, class, basic attack, defense and resistance parameters and its size correspondingly.

- $name$ and $class$ are strings and atk, def, res and $size$ are integers.
- $name$ consists of lowercase Latin letters and its length can range from 1 to 10, inclusive.
- $class$ can be "`weapon`", "`armor`" or "`orb`".
- $0 \le atk, def, res \le 1000$.
- $1 \le size \le 10$.

It is guaranteed that Laharl has at least one item of each class.

The next line contains an integer k ($1 \le k \le 1000$) — the number of residents.

Then k lines follow. Each of them describes a resident. A resident description looks like: "$name\ type\ bonus\ home$" — the resident's name, his type, the number of points the resident adds to the item's corresponding parameter and the name of the item which currently contains the resident.

- $name$, $type$ and $home$ are strings and $bonus$ is an integer.
- $name$ consists of lowercase Latin letters and its length can range from 1 to 10, inclusive.
- $type$ may be "`gladiator`", "`sentry`" or "`physician`".
- $1 \le bonus \le 100$.

It is guaranteed that the number of residents in each item does not exceed the item's size.

The names of all items and residents are pairwise different.

All words and numbers in the input are separated by single spaces.

Output

Print on the first line the name of the weapon in the optimal equipping pattern; then print the number of residents the weapon contains; then print the residents' names.

Print on the second and third lines in the same form the names of the armor and defensive orb as well as the residents they contain.

Use single spaces for separation.

If there are several possible solutions, print any of them.

Sample test(s)

input
4 sword weapon 10 2 3 2 pagstarmor armor 0 15 3 1 iceorb orb 3 2 13 2

```
longbow weapon 9 1 2 1
5
mike gladiator 5 longbow
bobby sentry 6 pagstarmor
petr gladiator 7 iceorb
teddy physician 6 sword
blackjack sentry 8 sword
```

output

```
sword 2 petr mike
pagstarmor 1 blackjack
iceorb 2 teddy bobby
```

input

```
4
sword weapon 10 2 3 2
pagstarmor armor 0 15 3 1
iceorb orb 3 2 13 2
longbow weapon 9 1 2 1
6
mike gladiator 5 longbow
bobby sentry 6 pagstarmor
petr gladiator 7 iceorb
teddy physician 6 sword
blackjack sentry 8 sword
joe physician 6 iceorb
```

output

```
longbow 1 mike
pagstarmor 1 bobby
iceorb 2 petr joe
```

Note

In the second sample we have no free space inside the items, therefore we cannot move the residents between them.

D. Entertaining Geodetics

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

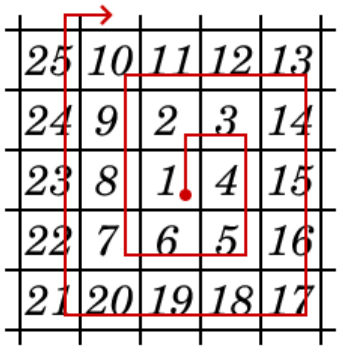
The maps in the game are divided into square cells called Geo Panels. Some of these panels are painted. We shall assume that the Geo Panels without color are painted the transparent color.

Besides, the map has so-called Geo Symbols. They look like pyramids of different colors (including Geo Symbols of the transparent color). Each Geo Symbol is located on one Geo Panel, and each Geo Panel may contain no more than one Geo Symbol.

Geo Symbols can be eliminated. To understand better what happens when a Geo Symbol is eliminated, let us introduce some queue to which we will put the recently eliminated Geo Symbols.

Let's put at the head of the queue a Geo Symbol that was eliminated just now. Next, we will repeat the following operation:

Extract the Geo Symbol from the queue. Look at the color of the panel containing the given Geo Symbol. If it **differs from transparent** and differs from the color of the Geo Symbol, then all Geo Panels of this color are repainted in the color of the given Geo Symbol (transparent Geo Symbols repaint the Geo Panels transparent). Repainting is executed in an infinite spiral strictly in the following order starting from the panel, which contained the Geo Symbol:



In other words, we select all the panels that need to be repainted and find their numbers in the infinite spiral whose center is placed in the position of the given Geo Symbol. After that, we repaint them in the order of the number's increasing.

If a panel contains another Geo Symbol and this panel is being repainted, then the Geo Symbol is removed from the field and placed at the tail of the queue.

After repainting the Geo Symbol is completely eliminated and the next Geo Symbol is taken from the head of the queue (if there is any) and the process repeats. The process ends if the queue is empty.

See the sample analysis for better understanding.

You know the colors of all the Geo Panels and the location of all the Geo Symbols. Determine the number of repaintings, which will occur if you destroy one of the Geo Symbols.

Input

The first line contains two integers n and m ($1 \le n, m \le 300$) — the height and the width of the map (in cells).

Then follow n line; each of them contains m numbers — the Geo Panels' colors.

Then follow n more lines; each of them contains m numbers — the Geo Symbols' description. -1 means that the given position contains no Geo Symbol. Otherwise, the number represents the color of the Geo Symbol in the given position.

All colors are integers from 0 to 10^9. 0 represents the transparent color.

The last line contains two integers x and y ($1 \le x \le n$, $1 \le y \le m$) — the row and the column where the Geo Symbol is placed that needs to be eliminated. The rows are numbered from top to bottom, the columns are numbered from left to right. Coordinates are 1-based. It is guaranteed that the position with coordinates (x, y) contains a Geo Symbol.

Output

Print the single number — the total number of repaintings after the Geo Symbol is eliminated.

Please, do not use the `%lld` specificator to read or write 64-bit integers in C++. It is preferred to use the `cout` stream (you may also use the `%I64d` specificator).

Sample test(s)

input
5 5 9 0 1 1 0 0 0 3 2 0 1 1 1 3 0 1 1 1 3 0 0 1 2 0 3

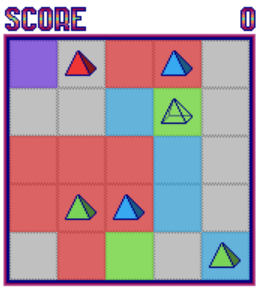
```
-1 1 -1 3 -1
-1 -1 -1 0 -1
-1 -1 -1 -1 -1
-1 2 3 -1 -1
-1 -1 -1 -1 2
4 2
```

output

```
35
```

Note

All actions of the sample you can see on the following picture:



If your browser does not support APNG and you see just static image, you can see GIF version of this image by the following link: http://212.193.37.254/codeforces/images/geo_slow.gif

E. Lift and Throw

time limit per test: 1.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a straight half-line divided into segments of unit length, which we will call positions. The positions are numbered by positive integers that start with 1 from the end of half-line, i. e. $1, 2, 3$ and so on. The distance between the positions is the absolute difference between the respective numbers.

Laharl, Etna and Flonne occupy some positions on the half-line and they want to get to the position with the largest possible number. They are originally placed in different positions.

Each of the characters can perform each of the following actions **no more than once**:

- Move a certain distance.
- Grab another character and lift him above the head.
- Throw the lifted character a certain distance.

Each character has a *movement range* parameter. They can only move to free positions, assuming that distance between those positions doesn't exceed the movement range.

One character can lift another character if the distance between the two characters equals 1, and no one already holds that another character. We can assume that the lifted character moves to the same position as the person who has lifted him, and the position in which he stood before becomes free. A lifted character cannot perform any actions and the character that holds him cannot walk.

Also, each character has a *throwing range* parameter. It is the distance at which this character can throw the one lifted above his head. He can only throw a character to a free position, and only when there is a lifted character.

We accept the situation when one person grabs another one who in his turn has the third character in his hands. This forms a "column" of three characters. For example, Laharl can hold Etna while Etna holds Flonne. In this case, Etna and the Flonne cannot perform any actions, and Laharl can only throw Etna (together with Flonne) at some distance.

Laharl, Etna and Flonne perform actions in any order. They perform actions in turns, that is no two of them can do actions at the same time.

Determine the maximum number of position at least one of the characters can reach. That is, such maximal number x so that one of the characters can reach position x.

Input

The first line contains three integers: Laharl's position, his movement range and throwing range. The second and the third lines describe Etna's and Flonne's parameters correspondingly in the similar form. It is guaranteed that the three characters occupy distinct positions. All numbers in the input are between 1 and 10, inclusive.

Output

Print a single number — the maximum ordinal number of position which either Laharl, Etna or Flonne can reach.

Sample test(s)

input
9 3 3 4 3 1 2 3 3
output
15

Note

Let us explain how to reach position 15 in the sample.

Initially Laharl occupies position 9, Etna — position 4 and Flonne — position 2.

First Laharl moves to position 6.

Then Flonne moves to position 5 and grabs Etna.

Laharl grabs Flonne and throws to position 9.

Flonne throws Etna to position 12.

Etna moves to position 15.