Platinum Sponsor

Silver Sponsor

Technical Sponsor

3rd Gulf Programming Contest

March 13,14 2013

Duration: 10 am - 3 pm

Problem #	***Problem Name***	***Balloon Color***
A	Is your PC smarter…	Purple
B	Tarzan's Puzzle	Red
C	Recycle Truck	Pale Blue
D	Posters	White
E	Construct a Word	Black
F	The Game of Life in 3D!	Lime Green
G	The Cipher Hacker	Yellow
H	Hydrogen Fusion	Silver
I	Train Stations	Pink

Problem A:
Is your PC smarter than a 2nd grader?

Source file: smart.{c | cpp | java}
Input file: smart.in

Children in 2nd grade of elementary school learn mathematical concepts by using simple equations like the following, where the question mark has to be replaced by the correct integer number:

```
2 + ? = 3 + 4
10 - 3 + 6 = ?
2 + ? = 14
```

The equations are using only addition and subtraction. Also no parentheses are used. Your task is to write a program that would be able to produce the correct answer.

Input

The first line of input contains an integer *N* indicating the number of test cases to follow. Each input case is in a line of its own, with spaces separating the numbers and the operators.

Output

For each test case, you should output what the mystery number should be in a separate line.

Sample Input

```
2 + ? = 3 + 4
10 - 3 + 6 = ?
2 + ? = 14
```

Sample Output

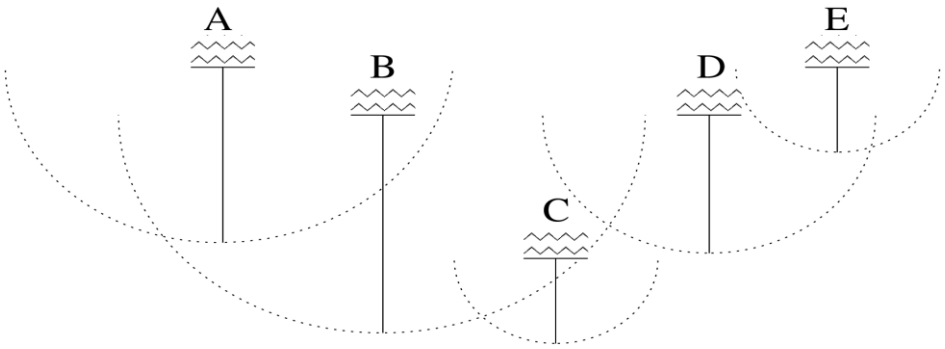
```
5
13
12
```

Problem B:
Tarzan's Puzzle

Source file: `tarzan.{c | cpp | java}`
Input file: `tarzan.in`

Jane is in trouble and Tarzan needs to reach her in a hurry. He has to cross the jungle as fast as possible, but in front of him lies a "sea" of tree ropes. You must help him choose the ones that will get him to Jane in the fewest number of swings (changes of rope) possible.

The tree ropes have different lengths and therefore each has a maximum range. Additionally, each is located at a different height. An example involving ropes lying on the same vertical plane, is shown below:



Tarzan can swing from one rope to the next as long as he does not go above the height of his rope's support point. All ropes are assumed to be resting, i.e. vertical, except the rope Tarzan starts from. So, in the above example, if Tarzan starts from rope A, he can go to B and then to C. However, from C he can go nowhere. Also if he started from D he could go to E but from E he can go nowhere.

Unfortunately, the problem is a bit more complicated as the ropes are not laid out over a line but are scattered in space. For each rope you are given the 3D coordinates of its suspension point, and its length.

Input

The first line of input contains an integer N indicating the number of test cases. The first line of each test case contains three integers: the number of ropes R, and the IDs of the starting and ending ropes (i.e. where Tarzan and Jane are initially). Ropes are numbered consecutively starting from 0 up to R-1, according to the order they appear in the input.

R lines follow, each containing 4 32bit integers: the x, y, and z coordinates of the rope's support (with z being the height from the ground) and the length of the rope. The rope is always shorter or equal to z.

Output

For each test case, you should output the number of ropes Tarzan should use in order to get to Jane. Each result should appear in a separate line.

If a solution is not possible, then you should output “Jane is doomed”.

Sample Input

```
2
5 0 4
0 0 10 10
10 10 20 5
4 4 7 6
5 4 10 9
7 7 10 3
7 0 6
0 0 10 1
0 1 10 1
0 2 10 1
1 1 10 1
2 0 10 1
2 1 10 1
2 2 10 1
```

Sample Output

```
1
4
```

Problem C:
Recycle Truck

Source file: `recycle.(c|cpp|java)`
Input: `recycle.in`

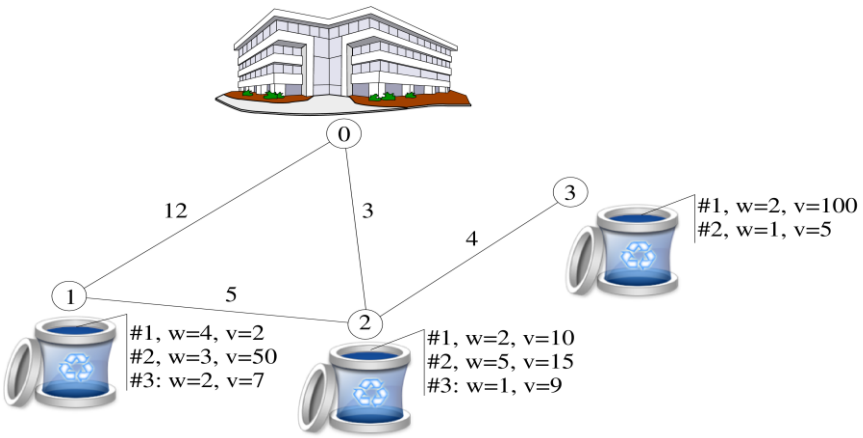
The city you live in has started an experimental effort to collect and recycle garbage using intelligent buckets (eBuckets). The new technology allows for an evaluation of the waste items' value and awards the citizens accordingly. For the effort to succeed and gather political and social support, it needs to suffer only small operational costs, or even achieve a small profitability from marketing the recycled raw materials.

The small scale pilot operation relies on a truck that traverses the available recycle buckets and collects as many of the waste items, as can fit in its cargo space *C*. The truck leaves the recycle plant and traverses the recycle buckets before returning to the recycle plant to drop off the collected items. The items cannot be broken-up when picked-up.

The truck goes out once every day. If the operation is to become a success, one needs to maximize the total value of the recycle items, minus the cost of the truck's journey. This is possible because the eBuckets inform the system operators of the value and volume of the waste items available in each eBucket, ahead of the truck's arrival.

The problem is described in the following manner: a weighted undirected graph with vertices representing the eBuckets and the plant is provided. The edge weights represent the associated monetary cost involved for moving the truck in one direction. Your task is to maximize the total value of the retrieved items that can fit in the truck's cargo bay, minus the cost of the collection process. The graph is known to be connected, i.e. there is a path between every pair of vertices.

An example is shown below:



In the above example, if *C*=12, the maximum benefit would be 157, if items:

- From node 1 : #2, #3
- From node 2 : #1, #3
- From node 3 : #1, #2

were chosen by traversing the graph in this fashion: 0 → 2 → 1 → 2 → 3 → 2 → 0 (the order of 1 and 3 could be switched).

Input

The input starts with an integer *N* representing the number of test cases. Each test case starts with a line holding three positive non-zero integers, *C*, *V* and *E: V* is the number of eBuckets ($V \leq 20$) and *E* the number of edges. The eBuckets are numbered consecutively starting from 1. Node 0 represents the recycling plant. *E* lines follow, each holding three integers A,B and C, where C is the cost of going from A to B (or vice-versa).

These are followed by a line holding *V* numbers $1 \leq M_i \leq 20$ each representing the number of waste items in the corresponding eBuckets. The input is finalized by $\sum M_i \leq 1000$ lines, each holding the volume w_i and value v_i of a waste item, where $w_i \leq C$.

Output

For each test case, you should print the test case number and the maximum profit (or minimum loss) that would result from a single outing of the truck. The restriction is that at least one waste item must be returned to the recycle plant.

Sample Input

```
1
12 3 4
1 2 5
3 2 4
1 0 12
0 2 3
3 3 2
4 2
3 50
2 7
2 10
5 15
1 9
2 100
1 5
```

Sample Output

```
1. 157
```

Problem D:
Posters

Source file: `posters.{c | cpp | java}`
Input file: `posters.in`

In universities, boards are hung in several places across campus for students and student clubs to place their posters and ads. Those posters usually come in different colors and sizes. Students, more often than not, place their posters in a random fashion on the allocated boards, resulting in a lot of wasted space on the boards. To make the problem worse, students almost never remove their old posters. If students who want to place their new posters do not find sufficient space for them, they simply place them over older posters, covering parts of one or more older posters. With all this mess on the boards, what we are interested in is the utilization of space on them. In other words, what percentage of the board is covered by posters?

The shapes of the boards and all the posters are simple rectangles. All posters will be placed in parallel with the axes of the board they are on.

Input

The first line of the input file contains an integer *N* indicating the number of boards. For each board, the first line contains the width *W*, and height *H* of the board, and the second line contains an integer *P* indicating the number of posters on the board. *P* lines follow, each containing two coordinates *Xi* and *Yi* of the lower left corner of poster *i*, and the width *Wi* and height *Hi* of that poster. The lower left corner of the board has coordinates of (0,0). All numbers in the input file fit in 32-bit signed integers and the number of posters *P* is in the range $0 \leq P \leq 400$.

Output

For each test case, print one line containing the total area covered by the posters followed by the division sign '/' and the total area of the board.

Sample Input

```
2
10 10
1
5 5 1 1
20 20
2
0 0 10 10
5 5 10 10
```

Sample Output

```
1/100
175/400
```

Problem E:
Construct a Word

Source file: `word.{c | cpp | java}`
Input file: `word.in`

"Construct a Word" is a game for kids where part of a word is given and the player has to guess the missing part. We are going to create an even more boring game. Let's call it "Super Rabbit"! In this game, you have a rabbit, a string of upper case letters, and a hidden word. The rabbit has a special super power that allows it to hop over the letters of the given string. But the rabbit can only hop from the letter it's on to an adjacent one. When the rabbit hops over to a letter, that letter can be used once to construct the hidden word. In the beginning of the game, the first hop the rabbit takes lands it on the first letter on the string (this is hop number 1). The rabbit then hops to the next letters one by one until the last letter in the string. When it reaches the last letter in the string, it then turns direction, and the next hop lands it on the previous letter. It then continues to hop over the letters one by one in the opposite direction until it reaches the first letter in the string. It then changes directions again and the next hop lands it on the second letter, and so on. If there is only one letter, the rabbit will keep hopping in its place. The rabbit stops hopping as soon as all the letters required to construct the hidden word are acquired.

Given the string and the hidden word, your task is to find out the number of times the rabbit will hop before it stops.

For example, if the string is "AYTHOOHGIWE" and the hidden word is "HI", the rabbit will hop 9 times to acquire the 'H' and the 'I'. If the string is "QWOEYLHRTS" and the hidden word is "HELLO", the rabbit will hop 14 times (Note the need to acquire the 'L' twice).

Input

The first line of the input file contains an integer *N* indicating the number of test cases. The first line of each test case contains the string *S* which has a maximum size of 1,000 letters, while the second line contains the hidden word *W* with a maximum size of 100 letters. All letters are upper case alphabets and no empty strings will be given in the file.

Output

For each test case, output the number of times the rabbit will hop before it stops. If the rabbit never stops hopping, print "Super Rabbit!"

Sample Input

```
3
AYTHOOHGIWE
HI
QWOEYLHRTS
HELLO
SUPER
RABBIT
```

Sample Output

```
9
14
Super Rabbit!
```

Problem F:
The Game of Life in 3D!

Source file: `life.{c | cpp | java}`
Input file: `life.in`

The extraterrestrial organisms collected at the third planet of *beta-orion* are behaving strangely. Each organism is contained in a cubic cell having unit length in each side. Q such cells are stacked to form a rectangular parallelepiped having dimensions M, N, and H. Therefore, Q = M x N x H. Now, let's come back to the point. What is so strange about the organisms? It is found that an organism in cell *x* lives or dies based on the density of living organisms in its neighborhood. Let us be precise with the following definitions.

Let *Neighborhood*(*x*) = {all cells (including *x*) sharing edge(s) with cell *x*}

S(*x*) = | *Neighborhood*(*x*)| (i.e., total cells in *Neighborhood(x)*)

The organisms evolve through generations. Let *t* be the current generation, and *t*+1 be the next generation. Let *Alive*(*x*, *t*) be true if the organism at cell *x* is alive at generation *t*. Also, let

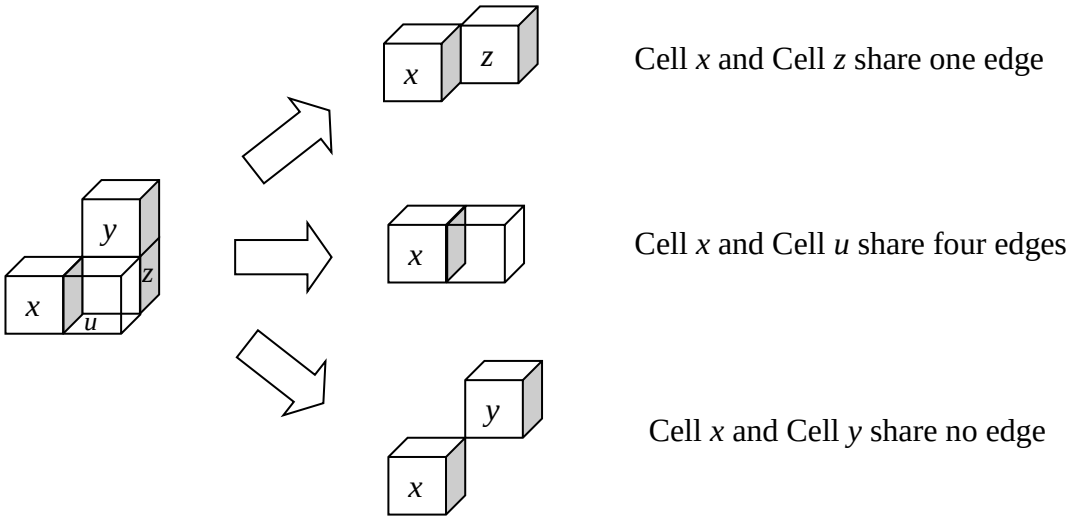
D(x, t) = Density of *Neighborhood(x)* in generation *t* = total number of live organisms in *Neighborhood*(*x*) in generation *t*.

Therefore, the rule of evolution is as follows:

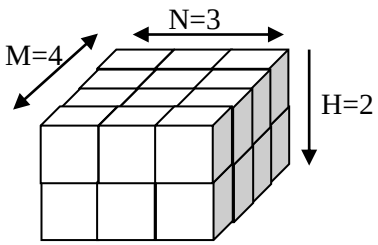
$$Alive(x, t+1) = \begin{cases} \text{true, if } \left\lfloor \frac{S(x)}{3} \right\rfloor \le D(x,t) \le \left\lfloor \frac{2S(x)}{3} \right\rfloor \\ \text{false, otherwise} \end{cases};$$

Also, *Alive*(*x*, *0*) = *true* if the sum *i* + *j* + *k* is even, where (*i*, *j*, *k*) are the indices (corresponding to the first, second, and the third dimension) of cell *x* within the parallelepiped. The cell indices range from (0,0,0) to (*M*-1,*N*-1,*H*-1).

Given a particular parallelepiped cell stack, you are to determine the total number of organisms that are alive at a certain generation.



Illustrating several cases of edge sharing



Illustrating a parallelepiped with M=4, N=3, H=2; so total cells, Q=24

Input

There may be several scenarios in the input. Each scenario starts with a line containing three positive integers: M, N, and H (0 < M, N, H <= 100). This follows a number of lines indicating the generation number (non-negative integer <= 1000,000), one per line. A blank line indicates the end of one scenario. Then another scenario starts with another set of M, N, and H. The input ends with a line containing the string "GAME OVER".

Output

For each scenario, print the number of alive organisms for each generation mentioned in the input.

Sample Input

```
10 10 10
100
5000
20

20 20 20
100
5000
500

GAME OVER
```

Sample Output

```
596
0
612
3984
3596
3688
```

Problem G:
The Cipher Hacker

Source file: `cipher.{c | cpp | java}`
Input file: `cipher.in`

Cryptography is the technique of encrypting (i.e., transforming) plain messages to an unreadable form using a secret key. One of the oldest applications of cryptography dates back to the Roman emperor Julius Caesar, who used his own technique, called the *Caesar cipher*. A similar, but more complex cryptographic technique is to substitute one letter with another, which is called substitution cipher. For example, suppose the letters *f*, *o*, and *x* are replaced with *p*, *d*, and *b*, respectively. Therefore, the word *fox* would become *pdb* after the substitution. In this example, *fox* is called the "plaintext" and *pdb* is the corresponding "ciphertext". In order to crack (or hack) a cipher (i.e., derive the plaintext from the ciphertext), the hacker needs to know the *key*, i.e., which letter has been substituted with which. In this problem, the hacker has access to a dictionary of plaintext words, and a ciphertext. Can you help the hacker to break the cipher?

Input

The input file starts with a dictionary of plaintext words on a single line. Two consecutive words will be separated by a space. The total number of words in the dictionary will be less than 1000. Also, no word will contain more than 10 characters. All characters are lowercase letters ('a'-'z'). The next line will contain the ciphertext. Two ciphertext words will also be separated by a space, and there will be no more than 100 words of ciphertext. After the ciphertext, there may be another dictionary – ciphertext pair, or a line containing only the word "END", indicating the end of input.

Output

For each dictionary – ciphertext pair, output the key, i.e., the substitution for each plaintext letter in the ciphertext. If there are more than one possible keys, then print the lexicographically smallest one. If there is no valid key, then output should read "no key found". In the first sample output, the key is "`xyzmnprqostuvwdefhgijklbca`", which means 'a' is substituted with 'x', 'b' with 'y', 'c' with 'z', 'd' with 'm', ……'z' with 'a'. Therefore, the plaintext word *fox* is substituted with the ciphertext word *pdb*, the plaintext word *quick* is substituted with the ciphertext word *fjozt*, and so on. Note that in a key each letter appears exactly once.

Sample Input

```
a quick brown fox jumps over the lazy dogs
yhdlw iqn mdrg x fjozt uxac dknh sjveg pdb
a quick brown fox jumps over the lazy dogs
dknh uxac mdrg fjozt pdb sjveg iqn
a b ac ace aceg acegi bd bdf bdfh bdfhj ack ackm ackmo bdl bdln
m n moq moqsu np npr nprt nprtv mow mowy mowya npx upxz
a b ac ace aceg acegi bd bdf bdfh bdfhj ack ackm ackmo bdl bdln
npxz npr npx np nprt moq nprtv moqsu mow mowy mowya
END
```

Sample Output

```
xyzmnprqostuvwdefhgijklbca
xlzmnprqostuvwdefhgijkybca
no key found
mnopqrstuvwxyzabcdefghijkl
```

Problem H:
Hydrogen Fusion

Source file: `fusion.{c | cpp | java}`
Input file: `fusion.in`

Hydrogen fusion is the process that produces the powerful explosions in stars, such as our sun. In a nutshell, two hydrogen atoms are heated and squeezed together to generate one helium atom. Scientists are trying to mimic this process to generate clean energy. To do so, hydrogen atoms need to be blasted with laser beams from multiple sources. At least three laser sources are needed and should be placed within equal distance from the hydrogen atoms.

Write a program that takes the position of three leaser sources as input and computes the position of the hydrogen atoms, as well as the required distance *d*, between the hydrogen and the laser sources. The three laser sources will never be co-linear.

Input

Your input will be several lines, each consisting of three points. The points are represented as $x_1\ y_1$, $x_2\ y_2$, $x_3\ y_3$. Note that commas separate points, and spaces separate *x* from *y* values. Your input is terminated with a line containing -1.

Output

For each of the input lines, print: $x_h\ y_h$, *d*, where $x_h\ y_h$ is the point of hydrogen placement, followed by the distance *d*. Round all output values to the nearest integer.

Sample Input

```
1 1, 1 7, 4 4
9 12, 5 5, 6 8
-10 -20, 0 0, -33 -20
-1
```

Sample Output

```
1 4, 3
18 3, 13
-21 -2, 22
```

Problem I:
Train Stations

Source file: `train.{c | cpp | java}`
Input file: `train.in`

A train station is a railway facility where trains regularly stop to load or unload passengers or freight. In a large city, train stations are given numbers; 1, 2, 3, 4, … (max 10000 cities) . Every station is connected to other stations. A connection from 1 to 2 means that the train can go from station 1 to station 2, but not necessarily the other way.

Input

The first line of the input is the number of test cases. For each test case, the first line is the number of stations. After that each station takes 2 lines in the input file. The first line of each station has the station number (an integer), and the second line of each station has a list of all the stations that can be accessed directly from the current station. This list is terminated by a 0.

Output

For each case, print all the possible tours for a visitor in the format shown below. The tour should visit every station without visiting a station more than once. If there is more than one valid tour, print the tours in ascending order of their stations. If no such tour exists print "NO TOURS".

Sample Input

```
1
6
1
2 3 0
2
3 0
3
2 5 0
4
0
6
5 4 0
5
6 4 0
```

Sample Output

```
CASE: 1
Tour 1: 1 2 3 5 6 4
```