Problem A
Cuckoo Hashing

One of the most fundamental data structure problems is the dictionary problem: given a set D of words you want to be able to quickly determine if any given query string q is present in the dictionary D or not. Hashing is a well-known solution for the problem. The idea is to create a function $h : \Sigma* \rightarrow [0..n-1]$ from all strings to the integer range 0, 1, ..,$n-1$, i.e. you describe a fast deterministic program which takes a string as input and outputs an integer between 0 and $n--1$. Next you allocate an empty hash table T of size n and for each word w in D, you set $T[h(w)] = w$. Thus, given a query string q, you only need to calculate $h(q)$ and see if $T[h(q)]$ equals q, to determine if q is in the dictionary.

The procedure seems simple enough, but aren't we forgetting something? Of course, what if two words in D map to the same location in the table? This phenomenon, called collision, happens fairly often (remember the Birthday paradox: in a class of 24 pupils there is more than 50% chance that two of them share birthday). On average you will only be able to put roughly $\sqrt{n}$-sized dictionaries into the table without getting collisions, quite poor space usage!

A stronger variant is Cuckoo Hashing[1]. The idea is to use two hash functions h_1 and h_2. Thus each string maps to two positions in the table. A query string q is now handled as follows: you compute both $h_1(q)$ and $h_2(q)$, and if $T[h_1(q)] = q$, or $T[h_2(q)] = q$, you conclude that q is in D. The name "Cuckoo Hashing" stems from the process of creating the table. Initially you have an empty table. You iterate over the words d in D, and insert them one by one. If $T[h_1(d)]$ is free, you set $T[h_1(d)] = d$. Otherwise if $T[h_2(d)]$ is free, you set $T[h_2(d)] = d$. If both are occupied however, just like the cuckoo with other birds' eggs, you evict the word r in $T[h_1(d)]$ and set $T[h_1(d)] = d$. Next you put r back into the table in its alternative place (and if that entry was already occupied you evict that word and move it to its alternative place, and so on). Of course, we may end up in an infinite loop here, in which case we need to rebuild the table with other choices of hash functions. The good news is that this will not happen with great probability even if D contains up to $n/2$ words!

Input

On the first line of input is a single positive integer $1 \leq t \leq 50$ specifying the number of test cases to follow. Each test case begins with two positive integers $1 \leq m \leq n \leq 10000$ on a line of itself: m tells the number of words in the dictionary; and n tells the size of the hash table in the test case. Next follow m lines of which the i:th describes the i:th word (d_i) in the dictionary D by two non-negative integers $h_1(di)$ and $h_2(di)$ less than n giving the two hash function values of the word d_i. The two values may be identical.

The input must be read from the file cuckoo.in.

[1]Cuckoo Hashing was suggested by the danes R. Pagh and F. F. Rödler in 2001

Output

For each test case there should be exactly one line of output either containing the string `successful hashing` if it is possible to insert all words in the given order into the table, or the string `rehash necessary` if it is impossible.

The output must be written to standard output.

Sample Input

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

Sample output

```
successful hashing
rehash necessary
```

Problem B
Optimal Parking

When shopping on Long Street, Michael usually parks his car at some random location, and then walks to the stores he needs. He is very strong, and does not mind carrying all the bags around. Can you help Michael choose a place to park which minimizes the distance he needs to walk on his shopping round?

Long Street is a straight line, where all positions are integer.

Input

The first line of input gives the number of test cases, $1 \leq t \leq 100$. There are two lines for each test case. The first gives the number of stores Michael wants to visit, $1 \leq n \leq 20$, and the second gives their n integer positions on Long Street, $0 \leq xi \leq 99$.

The input must be read from the file parking.in*.*

Output

Output for each test case a line with the minimal distance Michael must walk given optimal parking.

The output must be written to standard output.

Sample Input

```
2
4
24 13 89 37
6
7 30 41 14 39 42
```

Sample output

```
152
70
```

Problem C
Circle of Debt

The three friends Alice, Bob, and Cynthia always seem to get in situations where there are debts to be cleared among themselves. Of course, this is the "price" of hanging out a lot: it only takes a few restaurant visits, movies, and drink rounds to get an unsettled balance. So when they meet as usual every Friday afternoon they begin their evening by clearing last week's debts. To satisfy their mathematically inclined minds they prefer clearing their debts using as little money transaction as possible, i.e. by exchanging as few bank notes and coins as necessary. To their surprise, this can sometimes by harder than it sounds.

Suppose that Alice owes Bob 10 crowns and this is the three friends' only uncleared debt, and Alice has a 50 crown note but nothing smaller, Bob has three 10 crown coins and ten 1 crown coins, and Cynthia has three 20 crown notes. The best way to clear the debt is for Alice to give her 50 crown note to Cynthia, Cynthia to give two 20 crown notes to Alice and one to Bob, and Bob to give one 10 crown coin to Cynthia, involving a total of only five notes/coins changing owners. Compare this to the straight- forward solution of Alice giving her 50 crown note to Bob and getting Bob's three 10 crown notes and all his 1 crown coins for a total of fourteen notes/coins being exchanged!

Input

On the first line of input is a single positive integer, $1 \leq t \leq 50$, specifying the number of test cases to follow. Each test case begins with three integers ab, bc, $ca \leq 1000$ on a line of itself. ab is the amount Alice owes Bob (negative if it is Bob who owes Alice money), bc the amount Bob owes Cynthia (negative if it is Cynthia who is in debt to Bob), and ca the amount Cynthia owes Alice (negative if it is Alice who owes Cynthia).

Next follow three lines each with six non-negative integers $a_{100}, a_{50}, a_{20}, a_{10}, a_5, a_1, b_{100}, ..., b_1$, and $c_{100}, ..., c_1$, respectively, where a_{100} is the number of 100 crown notes Alice got, a_{50} is the number of her 50 crown notes, and so on. Likewise, $b_{100}, ..., b_1$ is the amount of notes/coins of different value Bob got, and $c_{100}, ..., c_1$ describes Cynthia's money. Each of them has at most 30 coins (i.e. $a_{10} + a_5 + a_1$, $b_{10} + b_5 + b_1$, and $c_{10} + c_5 + c_1$ are all less than or equal to 30) and the total amount of all their money together (Alice's plus Bob's plus Cynthia's) is always less than 1000 crowns..

The input must be read from the file debt.in.

Output

For each test case there should be one line of output containing the minimum number of bank notes and coins needed to settle the balance. If it is not possible at all, output the string `impossible`.

The output must be written to standard output.

Sample Input

```
3
10 0 0
0 1 0 0 0 0
0 0 0 3 0 10
0 0 3 0 0 0
-10 -10 -10
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
-10 10 10
3 0 0 0 2 0
0 2 0 0 0 1
0 0 1 1 0 3
```

Sample output

```
5
0
impossible
```

Problem D
Full Tank

After going through the receipts from your car trip through Europe this summer, you realized that the gas prices varied between the cities you visited. Maybe you could have saved some money if you were a bit more clever about where you filled your fuel?

To help other tourists (and save money yourself next time), you want to write a program for finding the cheapest way to travel between cities, based on the prices of gas in each city. We assume that all cars use one unit of fuel per unit of distance, and start with an empty gas tank.

Input

The first line of input gives $1 \leq n \leq 1000$ and $0 \leq m \leq 10000$, the number of cities and roads. Then follows a line with n integers $1 \leq p_i \leq 100$, where p_i is the price of one unit of fuel in the i-th city. Then follow m lines with three integers $0 \leq u, v < n$ and $1 \leq d \leq 100$, telling that there is a road between u and v with length d. Then comes a line with the number $1 \leq q \leq 100$, giving the number of queries, and q lines with three integers $1 \leq c \leq 100$, s and e, where c is the fuel capacity of the vehicle, s is the starting city, and e is the goal.

The input must be read from the file `tank.in`.

Output

For each query, output the price of the cheapest trip from s to e using a car with the given capacity, or `impossible` if there is no way of getting from s to e with the given car.

The output must be written to standard output.

Sample Input

```
5 5
10 10 20 12 13
0 1 9
0 2 8
1 2 1
1 3 11
2 3 7
2
10 0 3
20 1 4
```

Sample output

```
170
impossible
```

Problem E
Worst Weather Ever

“Man, this year has the worst weather ever!”, David said as he sat crouched in the small cave where we had sought shelter from yet another sudden rainstorm.
“Nuh-uh!”, Diana immediately replied in her traditional know-it-all manner.
“Is too!”, David countered cunningly.
Terrific. Not only were we stuck in this cave, now we would have to listen to those two nagging for at least an hour. It was time to cut this discussion short.
“Big nuh-uh. In fact, 93 years ago it had already rained five times as much by this time of year.”
“Duh”, David capitulated, “so it’s the worst weather in 93 years then.”
“Nuh-uh, this is actually the worst weather in 23 years.”, Diana again broke in.
“Yeah, well, whatever”, David sighed, “Who cares anyway?”.

Well, dear contestants, you care, don’t you?

Your task is to, given information about the amount of rain during different years in the history of the universe, and a series of statements in the form “Year X had the most rain since year Y”, determine whether these are true, might be true, or are false. We say that such a statement is true if:

- The amount of rain during these two years and all years between them is known.
- It rained at most as much during year X as it did during year Y.
- For every year Z satisfying $Y < Z < X$, the amount of rain during year Z was less than the amount of rain during year X.

We say that such a statement might be true if there is an assignment of amounts of rain to years for which there is no information, such that the statement becomes true. We say that the statement is false otherwise.

Input

The input will consist of several test cases, each consisting of two parts.

The first part begins with an integer $1 \leq n \leq 50000$, indicating the number of different years for which there is information. Next follow n lines. The i-th of these contains two integers $-10^9 \leq y_i \leq 10^9$ and $1 \leq r_i \leq 10^9$ indicating that there was r_i milliliters of rain during year y_i (note that the amount of rain during a year can be any nonnegative integer, the limitation on r_i is just a limitation on the input). You may assume that $y_i < y_{i+1}$ for $1 \leq i < n$.

The second part of a test case starts with an integer $1 \leq m \leq 10000$, indicating the number of queries to process. The following m lines each contain two integers $-10^9 \leq Y < X \leq 10^9$ indicating two years.

There is a blank line between test cases. The input is terminated by a case where $n = 0$ and $m = 0$. This case should not be processed.

Technical note: Due to the size of the input, the use of cin/cout in C++ might be too slow in this problem. Use scanf/printf instead. In Java, make sure that both input and output is buffered.

The input must be read from the file worst.in.

Output

There should be m lines of output for each test case, corresponding to the m queries. Queries should be answered with `true` if the statement is true, `maybe` if the statement might be true, and `false` if the statement is false.

Separate the output of two different test cases by a blank line.

The output must be written to standard output.

Sample Input

```
4
2002 4920
2003 5901
2004 2832
2005 3890
2
2002 2005
2003 2005

3
1985 5782
1995 3048
2005 4890
2
1985 2005
2005 2015
0
0
```

Sample output

```
false
true

maybe
maybe
```

Problem F
Shopaholic

Lindsay is a shopaholic. Whenever there is a discount of the kind where you can buy three items and only pay for two, she goes completely mad and feels a need to buy all items in the store. You have given up on curing her for this disease, but try to limit its effect on her wallet.

You have realized that the stores coming with these offers are quite selective when it comes to which items you get for free; it is always the cheapest ones. As an example, when your friend comes to the counter with seven items, costing 400, 350, 300, 250, 200, 150, and 100 dollars, she will have to pay 1500 dollars. In this case she got a discount of 250 dollars. You realize that if she goes to the counter three times, she might get a bigger discount. E.g. if she goes with the items that costs 400, 300 and 250, she will get a discount of 250 the first round. The next round she brings the item that costs 150 giving no extra discount, but the third round she takes the last items that costs 350, 200 and 100 giving a discount of an additional 100 dollars, adding up to a total discount of 350.

Your job is to find the maximum discount Lindsay can get.

Input

The first line of input gives the number of test scenarios, $1 \leq t \leq 20$. Each scenario consists of two lines of input. The first gives the number of items Lindsay is buying, $1 \leq n \leq 20000$. The next line gives the prices of these items, $1 \leq pi \leq 20000$.

The input must be read from the file shopaholic.in.

Output

For each scenario, output one line giving the maximum discount Lindsay can get by selectively choosing which items she brings to the counter at the same time.

The output must be written to standard output.

Sample Input

```
1
6
400 100 200 350 300 250
```

Sample output

```
400
```

Problem G
Machined Surfaces

An imaging device furnishes digital images of two machined surfaces that eventually will be assembled in contact with each other.The roughness of this final contact is to be estimated.

A digital image is composed of the two characters, "X" and " " (space). There are always 25 columns to an image, but the number of rows, N, is variable. Column one (1) will always have an "X" in it and will be part of the left surface. The left surface can extend to the right from column one (1) as contiguous X's. Similarly, column 25 will always have an "X" in it and will be part of the right surface. The right surface can extend to the left from column 25 as contiguous X's.

```
Digital-Image View of Surfaces

       Left         Right

       XXXX         XXXXX  <-- 1
       XXX        XXXXXXX
       XXXXX         XXXX
       XX          XXXXXX
       .                .
       .                .
       .                .
       XXXX          XXXX
       XXX         XXXXXX  <-- N
       |                |
       1               25
```

In each row of the image, there can be zero or more space characters separating the left surface from the right surface. There will never be more than a single blank region in anyrow.

Foreach image given, you are to determine the total "void" that will exist after the left surface has been brought into contact with the right surface. The "void" is the total count of the spaces that remains between the left and right surfaces after they have been brought into contact.

The two surfaces are brought into contact by displacing them strictly horizontally towards each other until a rightmost "X" of the left surface of some row is immediately to the left of the leftmost "X" of the right surface of that row. There is no rotation or twisting of these two surfaces as they are brought into contact; they remain rigid, and only move horizontally.

Note: The original image may show the two surfaces already in contact, in which case no displacement enters into the contact roughness estimation.

Input

The input consists of a series of digital images. Each image data set has the following format:

First line – A single unsigned integer, N, with value greater than zero (0) and less than 13. The first digit of N will be the first character on a line.

Next N lines – Each line has exactly 25 characters; one or more X's, then zero or more spaces, then one or more X's.

The end of data is signaled by a null data set having a zero on the first line of an image data set and no further data.

The input must be read from the file surface.in.

Output

For each image you receive as a data set, you are to reply with the total void (count of spaces remaining after the surfaces are brought into contact). Use the default output for a single integer on a line.

The output must be written to standard output.

Sample Input

```
4
XXXX                XXXXX
XXX               XXXXXXX
XXXXX                XXXX
XX                 XXXXXX
2
XXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXX
1
XXXXXXXXX              XX
0
```

Sample output

```
4
0
0
```

Problem H
Flatland

Once upon a time there was *Flatland*, a world whose inhabitants believed was a 2D rectangular region. Flatlanders (the people inhabiting Flatland) assumed that if somebody traveled long enough in one direction, he/she would fall down over the edge of Flatland. Animated Caribbean Movies (ACM) plans to produce a film about Flatland. Before the script of the film is approved, ACM wants to become familiar with life as it was in Flatland by simulating how the world could evolve from given initial situations and some conditions determining life and death.

Flatlanders were nomads by nature as they were always traveling: all of them traveled at the same speed rate on straight lines but each individual had its own direction. As you may imagine, if one observed the life of a lonely Flatlander, he/she would eventually reach Flatland's edge and die. Nevertheless, if two Flatlanders collided, then their fate was improved as their directions changed: the resulting directions were as the former ones reflected in a mirror bisecting the angle between the former directions of the crashing inhabitants. So, a Flatlander survived because a collision with another one could change his/her direction.

However, there were bad news when more than two Flatlanders collided in a single crash: in that case all of them died, disappearing right there. Note that some Flatlanders could die at the same time. If this was the case, the last name of the list of deads was remembered as the *Last Dead One* in that moment (to simplify, we assume they used our modern English alphabet and lexicographic order). The survivors venerated the name of the Last Dead One until a new last dead appeared (and some Flatlander disappeared).

ACM's film begins with a given population in Flatland, where names, positions, and directions of every single individual are known. ACM wants you to help them to determine which would be the name of the last Last Dead One in the whole Flatland's life.

Input

There are NC test cases to solve, $0 < NC < 100$. The first line of the input file has NC. After that, for each testcase, a set of lines:

- the first line contains a number n, the number of Flatlanders in the initial world ($1 \leq n \leq 100$);
- the second line contains two positive integer numbers B and H separated by a space, representing the dimensions of Flatland ($2 \leq B, H \leq 100$). Coordinates in Flatland are points (i, j), with $0 \leq i \leq B$, $0 \leq j \leq H$. Flatland's edges are points with coordinates of the form $(0, j), (i, 0), (B, j)$ or (i, H);
- n lines (one per Flatlander) with four numbers and one string: x, y, d_1, d_2 and *name* separated by a blank. (x, y) represents the position of the Flatlander ($0 < x < B$, $0 < y < H$,

and two Flatlanders cannot start in the same position), (d_1, d_2) represents the direction: (d_1, d_2) is a point on some Flatland's edge, so that the Flatlander is moving towards it; *name* is a string of one to 10 alphabetical uppercase characters, which represents the name of the Flatlander. You may assume that every Flatlander has a unique name.

The input must be read from the file flatland.in*.*

Output

For each given case, output one line with the name of the Last Dead One.

The output must be written to standard output.

Sample input	**Output for the sample input**
`2`	`ALICE`
`2`	`BOB`
`20 23`	
`1 1 0 0 BOB`	
`3 3 3 0 ALICE`	
`3`	
`20 23`	
`2 2 4 0 ALICE`	
`4 2 2 0 BOB`	
`1 3 0 3 CHARLES`	

Problem I
Finding Seats Again

A set of n^2 computer scientists went to the movies. Fortunately, the theater they chose has a square layout: n rows, each one with n seats. However, these scientists are not all from the same research area and they want to seat together. Indeed, there are K independent research groups of scientists among them (no scientist belongs to two of them) with a distiguished leader for each group. Then the leader bought the tickets for his whole group, and he did it in such a way that all his group could seat occupying a rectangular set of seats (and everyone in this set of seats belongs to the same group). Every group was placed satisfying this bizarre condition, although the scientists did not care where the actual assigned areas were.

The usher was informed of the situation and he decided to annotate in a theater map a satisfactory seats deploying. He thought that if he wrote the position of each group's leader in the map indicating besides the corresponding group size, he could tell where to accomodate every scientist. But he discovered that it is not so easy!

The usher asks for your help. You must tell him a way to place the K rectangular areas with the given sizes, and with the corresponding leader for each group seated where it was originally assigned.

Input

Input consists of several test cases, each one defined by a set of lines:

- the first line in the case contains two numbers n and K separated by blanks, with n representing the size of the theater ($0 < n < 20$) and K the number of groups ($K \leq 26$);
- the next n lines describe the usher's map. A one-digit decimal number in the map indicates the seat of a leader and the size of his group. A point indicates that no leader will sit there.

The end of the input is indicated by the line
```
0 0
```

The input must be read from the file seats.in.

Output

For each test case, display an answer consisting in n lines each one of them with n characters representing a seat occupation for the theater. Each group is assigned to an uppercase letter and all of its members are identified with that letter. No two groups are assigned to the same letter.

The output must be written to standard output.

Sample input	**Output for the sample input**
`3 3`	`ABB`
`3.4`	`ABB`
`...`	`ACC`
`.2.`	`AAAABCC`
`7 18`	`DDDDBEF`
`...4.2.`	`GHIIBEF`
`...45..`	`GHJKBEF`
`222..3.`	`LLJKBMM`
`...2..3`	`NOJPQQQ`
`.24...2`	`NOJPRRR`
`...2.3.`	
`22..3..`	
`0 0`	