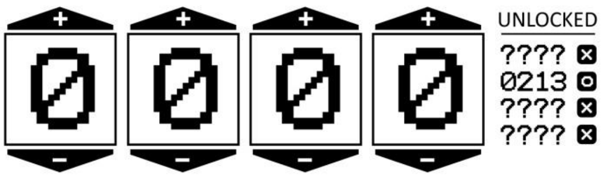
Problem A

Anti Brute Force Lock

Lately, there is one serious problem with Panda Land Safe Box: several safes have been robbed! The safes are using old 4-digits rolling lock combination (you only have to roll the digit, either up or down, until all four of them match the key). Each digit is designed to roll from 0 to 9. Rolling-up at 9 will make the digit become 0, and rolling-down at 0 will make the digit become 9. Since there are only 10000 possible keys, 0000 through 9999, anyone can try all the combinations until the safe is unlocked.

What's done is done. But in order to slow down future robbers' attack, Panda Security Agency (PSA) has devised a new safer lock with multiple keys. Instead of using only one key combination as the key, the lock now can have up to *N* keys which has to be all unlocked before the safe can be opened. These locks works as the following:

- Initially the digits are at 0000.
- Keys can be unlocked in any order, by setting the digits in the lock to match the desired key and then pressing the UNLOCK button.



- A magic JUMP button can turn the digits into any of the unlocked keys without doing any rolling.
- The safe will be unlocked if and only if all the keys are unlocked in a minimum total amount of rolling, excluding JUMP (yes, this feature is the coolest one).
- If the number of rolling is exceeded, then the digits will be reset to 0000 and all the keys will be locked again. In other word, the state of the lock will be reset the cracking is failed.

PSA is quite confident that this new system will slow down the cracking, giving them enough time to identify and catch the robbers. In order to determine the minimum number of rolling needed, PSA wants you to write a program. Given all the keys, calculate the minimum number of rolls needed to unlock the safe.

Input

The first line of input contains an integer *T*, the number of test cases follow. Each case begins with an integer *N* ($1 \le N \le 500$), the number of keys. The next *N* lines, each contains exactly four digits number (leading zero allowed) representing the keys to be unlocked.

Output

For each case, print in a single line the minimum number of rolls needed to unlock all the keys.

Sample Input	Output for Sample Input

4
2 1155 2211
3 1111 1155 5511
3 1234 5678 9090
4 2145 0213 9113 8113
	16
	20
	26
	17

Explanation for the 2nd case:

- Turn `0000` into `1111`, rolls: 4
- Turn `1111` into `1155`, rolls: 8
- Jump `1155` into `1111`, we can do this because `1111` has been unlocked before.
- Turn `1111` into `5511` rolls: 8

Total rolls = 4 + 8 + 8 = 20

Problem B

Bonus Treasure

In an archeology excavation in the Panda Land, the panda researchers found artifacts from the technologically advanced Bracket Panda civilization which lives around year 2008 BGP – Before Giant Panda. The artifacts are in the form of a big stone tablet with inscriptions on it (seems to be some kind of codes) and some locked treasure chests labeled as "bonus" by the ancient pandas, each chest have numbers inscribed on them as well as several switches labeled with '`(`' and '`)`'.

After much research and decoding, the researchers found the following results:

- Codes in the tablet describe a master sequence as the source of all possible switch positions (a.k.a. combination codes for opening the chest). In other words, all valid codes are defined as subsets of particular terms in the sequence. The terms of the master sequence are defined as being the concatenation of all previous terms of the sequence enclosed in the outer parenthesis '`(`' and '`)`', as follows: S_1=`()`, S_2=`(())`, S_3=`(()(()))`, S_4=`(()(())(()(())))`, and so on.

- There are three numbers in each treasure chest. The first number *N*, describes the term number in master sequence containing the code, the second number *K*, is a zero based index describing the position of starting character of the code in the *N*-th sequence term, and the last one *M*, describes the length of the sequence (number of the switches in the chest).

For larger values of *N*, the master sequence's term can become extremely long as the term's length is in the order of 2^N (as you may have figured out previously). Therefore as an elite programmer in Panda Land, you are given a task to write a program for decoding the combination codes - all that the program needs to do is printing the correct combination code required to open the treasure box, given the three numbers (*N*, *K*, *M*) inscribed on the treasure chest.

Input

The input consists of several cases. Each case contains three integer: *N* ($1 \le N \le 31$), *K* ($0 \le K < 2^N$) and *M* ($1 \le M \le \min\{10000, 2^N - K\}$). The input is terminated by a line where $N = K = M = 0$.

Output

For each case, output the correct combination of code required to open it (which is the substring from the *N*-th master sequence term, with length *M* starting from the *K*+1-th character of the term.)

Sample Input

```
3 0 6
3 4 4
3 5 2
3 6 1
3 7 1
0 0 0
```

Output for Sample Input

```
(()(()
()))
))
)
)
```

Problem C

Panda Land 7: Casino Island

There's a special island in Panda Land called Casino Island where all gambling activities are legalized. Here, a well known betting game called Tipu-Tipu is held every month.

Each gambler could place only one bet. Each bet should consists of *m* distinct numbers from 1 to *L* (*m* is chosen by the gambler). Each month, the game owner will announce a sequence of *L* distinct numbers (the 'master') and each bet will be matched to this sequence for a prize. The prize will be given if and only if the bet match perfectly to the master's prefix. In other word, a bet of *m* numbers, should match the first *m* numbers of the master. And yes, a higher *m* will gives you a higher prize but also a higher risk of not gaining anything.

For example:
The prizes for *m* from 1 to *L* respectively are: 10, 20, 30, 40, 50 and 60.
The master is: 1 2 3 4 5 6

1st bet	: 1	*m* = 1, match!,	prize = 10
2nd bet	: 1 2	*m* = 2, match!,	prize = 20
3rd bet	: 1 2 3 4	*m* = 4, match!,	prize = 40
4th bet	: 1 2 4 3	*m* = 4, not match,	prize = 0
5th bet	: 3 1	*m* = 2, not match,	prize = 0

Total prize = 10 + 20 + 40 + 0 + 0 = 70.

Of course the owner of the game wants to minimize the total of prize. So he asked you to write a program to find a master sequence which gives the minimum possible total of prize. If there are several of them, find the one which comes first lexicographically.

Input
The first line of input contains an integer *T*, the number of test cases to follow. Each case begins with two integers *L* ($3 \le L \le 30$) and *N* ($1 \le N \le 100000$). The next line contains *L* numbers representing the prizes for *m* = 1, 2, …, *L* respectively. Each prize will be between 1 and 100, inclusive. The next *N* lines describe all the *N* bets. Each bet is stated in a line, beginning with an integer *m* ($1 \le m \le L$) – the length of the bet, and followed by *m* distinct numbers describing the bet. Each number in the bet will be between 1 and *L*, inclusive.

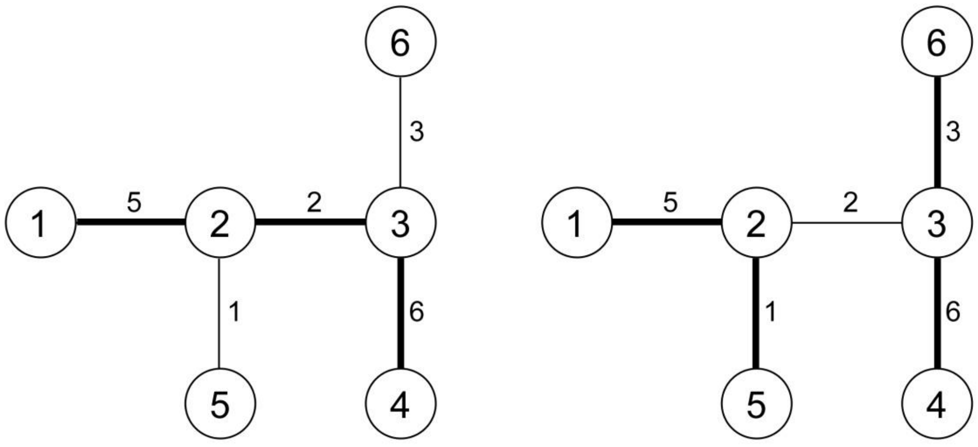
Output
For each case, print in a single line the master sequence which gives the minimum possible total prize. If there's more than one such sequence, output the one that comes first lexicographically. Each number in the sequence should be separated by a single space.

Sample Input	Output for Sample Input

```
2
4 3
1 2 3 4
2 1 2
4 2 1 3 4
4 2 1 4 3
4 13
1 2 3 4
2 2 1
4 1 2 3 4
4 1 2 4 3
2 1 3
2 1 4
2 2 3
2 2 4
2 3 1
2 3 2
2 3 4
2 4 1
2 4 2
2 4 3
```
	```
	1 3 2 4
	1 3 2 4
	```

Problem D
Disjoint Paths

Path in a graph is defined to be disjoint if there is no common edge and vertex that belongs to more than one path. Given a connected graph with *N* nodes and *N*-1 weighted edges, we are interested in finding a set of up to *K* paths in the graph which are disjoint to each other. The set of paths should be chosen so that the sum of all weight of edges of the paths is the maximum possible.



The example above shows two configurations of disjoint paths in the same graph. When the limit of paths to be drawn (K) is one (shown on the left), the maximum sum of all the weight of its edges is 13 (5+2+6). However, when we are allowed to draw up to 2 disjoint paths, we can draw two paths whose sum of edge weights evaluates to 15 (shown on the right).

Input

The input begins with a line containing an integer *T*, the number of test cases follow. Each case begins with two non-negative integers *N* and *K* ($K \le N \le 60$). The next $N - 1$ following lines each will contains three integers: *A*, *B* and *D* ($1 \le A, B \le N$; and $|D| \le 10000$) which means that there is an undirected edge from node *A* to node *B* with weight *D*.

Output

For each case, print in a single line the maximum possible sum of weight of up to *K* disjoint paths in the given graph.

Sample Input	Output for Sample Input

2
6 1
1 2 5
3 2 2
3 4 6
6 3 3
2 5 1
6 2
1 2 5
3 2 2
3 4 6
6 3 3
2 5 1
	13
	15

Problem E

Expert Enough?

Auto-mobile Charting & Manufacturing (ACM) is a company that specializes in manufacturing automobile spare parts. Being one of the leading automotive companies in the world, ACM are sure to keep up the latest information in that world. In the 100-year anniversary of the company, ACM compiled a huge list of range of prices of any automobiles ever recorded in the history. ACM then wants to develop a program that they called Automobile Expert System or AES for short.

The program receives a price *P* as an input, and searches through the database for a car maker in which *P* falls in their range of lowest price *L* and highest price *H* of car they ever made. The program then output the car maker name. If the database contains no or more than one car maker that satisfies the query, the program produce output “UNDETERMINED” (without quotes). Not so expert, huh? You are about to develop that program for ACM.

Input

The input begins with a line containing an integer T ($T \le 10$), the number of test cases follow. Each case begins with the size of the database D ($D < 10000$). The next each of D lines contains M, L and H ($0 < L < H < 1000000$) which are the name of the maker (contains no whitespace and will never exceeds 20 characters), the car’s lowest price the maker ever made, and the car’s highest price the maker ever made respectively. Then there is the number of query Q ($Q < 1000$). follows. Each of the next Q lines contains an integer P ($0 < P < 1000000$), the query price.

Output

Output for each query should be one line containing the name of the maker, or the string “UNDETERMINED” (without quotes) if there is no maker or more than one maker that satisfies the query. You should separate output for different case by one empty line.

Sample Input

```
1
4
HONDA 10000 45000
PEUGEOT 12000 44000
BMW 30000 75900
CHEVROLET 7000 37000
4
60000
7500
5000
10000
```

Output for Sample Input

```
BMW
CHEVROLET
UNDETERMINED
UNDETERMINED
```

Problem F

Free Parentheses

You are given a simple arithmetic expression which consists of only addition and subtraction operators. For example:

```
1 - 2 + 3 - 4 - 5
```

You are free to put any parentheses to the expression anywhere you want and as many as you want. However it should be a valid expression after you put the parentheses. The question is how many different numbers can you make?

For example, adding parentheses to the above expression can give you 6 different values:

```
1 - 2 + 3 - 4 - 5      =  -7
1 - (2 + 3 - 4 - 5)    =   5
1 - (2 + 3) - 4 - 5    = -13
1 - 2 + 3 - (4 - 5)    =   3
1 - (2 + 3 - 4) - 5    =  -5
1 - (2 + 3) - (4 - 5)  =  -3
```

Input

There will be many expressions in the input. Each expression is written in one line. The expression consists of only *N* (2 ≤ *N* ≤ 30) non-negative number less than 100, separated by addition or subtraction operators. There will be no operator before the first number.

Output

For each expression, print the number of different values that can be derived from the expression by adding any number of parentheses.

Sample Input	Output for Sample Input
1 - 2 + 3 - 4 - 5 38 + 29 - 91 54 - 18 + 22 + 74	6 1 3

Problem G

Greatest K-Palindrome Substring

Palindrome is a string that can be read in the same way in either forward or backward direction. For example: ABBA is a palindrome, MOM is also a palindrome, but MATE is not. A non-palindrome string can be transformed into a palindrome by changing some of its characters. We call a string a k-palindrome if it can be turned into a palindrome by changing at most *k* characters. By this definition, a regular palindrome string is 0-palindrome.

Given a string *S* of length *N* that contains only lowercase characters ('a'…'z') and an integer *k*, find the longest substring of *S* which is k-palindrome.

Input

The first line of the input contains an integer *T*, the number of test cases to follow. Each case consists of string *S* (1 ≤ |*S*| ≤ 1000) and integer *K* (0 ≤ *K* ≤ |*S*|). String *S* consists of lowercase characters ('a' … 'z') only. |*S*| denotes the length of string *S*.

Output

For each case, print in a single line: the length of the longest substring of S which is k-palindrome.

Sample Input	**Output for Sample Input**
`3`	`4`
`abba 0`	`3`
`mate 1`	`8`
`zabcddcbxy 1`	

Problem H
Hyper-Mod

Hyper operator $a^{(n)}b$ is a family of operators, where each level n operator is defined as iteration of the previous level (n – 1) operator. It is defined recursively as follows:

$$a^{(n)}b = \begin{cases} b+1, & if\ n=0 \\ a, & if\ n=1, b=0 \\ 0, & if\ n=2, b=0 \\ 1, & if\ n\geq 3, b=0 \\ a^{(n-1)}\left(a^{(n)}(b-1)\right) & if\ n\geq 1, b\geq 1, a\geq 0 \end{cases}$$

With some algebraic manipulations, it can be shown that the operators for n = 1, 2 and 3 correspond respectively to addition, multiplication and exponentiation; as given below:

$$a^{(1)}b = a + \{1+1+1+\cdots+1\} = a+b$$
$$a^{(2)}b = a+a+a+\cdots+a = a\times b$$
$$a^{(3)}b = a\times a\times a\times\cdots\times a = a^b$$

For values of n > 3, $a^{(n)}b$ typically corresponds to a (very) tall exponentiation tower even for small values of a and b, which makes it difficult or even impossible to calculate exactly. Nevertheless, as they are being iterations (or iterations of iterations of …) of exponents, their modulus can still be calculated by means of modular hyper-exponentiation (a form of iterated modular exponentiation). Your task is to compute $a^{(n)}b$ mod m, given the values of a, b, m and n.

Input
The first line of input contains an integer T (not more than 100), the number of test cases follow. The next T lines of input describe the input for each test case on one line, in the form of m n a b. You can safely assume that $0 \leq n \leq 10$; $0 \leq a, b \leq 1000$; and $1 \leq m \leq 10000$.

Output
For each of the test cases, print the test case number followed by the value of $a^{(n)}b$ mod m on one line. The sample output shows the exact format for printing the test case number.

Sample Input	Output for Sample Input

4
78 1 123 456
257 3 16 16
2008 4 8 3
1000 4 13 27
	Case #1: 33
	Case #2: 1
	Case #3: 816
	Case #4: 53

Problem I
ICPC Team Strategy

ICPC (International Collegiate Programming Contest), as you might know, is a team programming contest for college students. Each team consists of exactly three students and they will work on a number of programming problems.

Andi, Budi and Chandra plan to participate in this year ICPC as a team. As for their team strategy, they come up with a simple one:

- In the first 20 minutes of 5 hours contest, they will read through all problems. Then each of them will assign a number to each problem. This number indicates how many minute(s) it will take for him to get the problem accepted (correct solution). Then they will start to code, meaning that they only have 280 minutes to really work on the problems. You may assume that they always get accepted on time whenever they work on a problem.
- There's only one computer for each team, so it's not possible for them to code two different problems simultaneously.
- To avoid any brain fatigue and adrenaline rush (because they attend competitions so frequently), they decided to switch role after each problem, such that none of them will work at the computer for more than one problem consecutively.
- They want to solve as many problems as they can. The order of problem to be solved does not matter.

Input

The first line of input contains an integer T, number of test cases to follow. Each case begins with an integer N ($1 \le N \le 12$) in one line, denoting the number of problems. The second line contains N integers, $A_{1..N}$ ($1 \le A_i \le 300$), denoting the total time (in minutes) needed by Andi to solve i^{th} problem. The third and fourth line will correspond to the total time needed by Budi and Chandra respectively and will have the same input format as the second line.

Output

For each case, print in a single line containing the maximum total number of problem that can be solved by that team.

Sample Input

```
2
3
100 100 80
190 120 90
120 150 100
4
50 20 300 300
200 100 30 250
140 120 100 100
```

Output for Sample Input

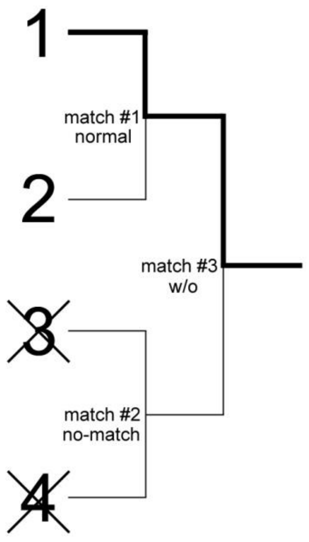
```
2
4
```

Explanation for 1st sample case:
Actually Andi could solve all the three problems alone, but the team has decided that none of them should work at the computer for more than one problem consecutively.

Explanation for 2nd sample case:
The team can solve all the problems. Here is one solution:
- Let Budi work on Problem-2 (100 minutes).
- Switch to Andi and let him work on Problem-1 (50 minutes).
- Switch to Budi again and let him work on Problem-3 (30 minutes).
- Finally, switch to Chandra and let him work on Problem-4 which (100 minutes).

Overall, they need 100 + 50 + 30 + 100 = 280 minutes.

Problem J
Jollybee Tournament



In Jollybee Chess Championship 2008, there are a number of players who have withdrawn themselves from the championship of 64 players (in this problem, we generalized it into 2^N players). Due to the nature of the competition, which is a regular knock-out tournament, and also the short notice of the withdrawals, some matches had been walkover matches (also known as a w/o, a victory due to the absent of the opponent).

If both players are available then there will be a normal match, one of them will advance to the next phase. If only one player is available then there will be a walkover match, and he/she will automatically advance. If no player is available then there will be no match.

In the left figure, the player #3 and #4 are withdrawn from the tournament, leaving a total of one w/o match (at match #3).

Given the list of players who withdraw right before the tournament start, calculate how many w/o matches to happen in the whole tournament, assuming that all of the remaining players play until the end of the tournament (winning or knocked-out).

Input
The first line of input contains an integer *T*, the number of test cases to follow. Each case begins with two integers, *N* ($1 \le N \le 10$) and *M* ($0 \le M \le 2^N$). The next line contains *M* integers, denoting the players who have withdrawn themselves right before the tournament. The players are numbered from 1 to 2^N, ordered by their position in the tournament draw.

Output
For each case, print in a single line containing the number of walkover matches.

Sample Input	Output for Sample Input

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

- This page is intentionally left blank –