Practice 1 – Arithmetic Sequence

To pass time during their long trek through the misty mountains and to prepare for the ordeals ahead, Gandalf starts teaching Math to Bilbo and the thirteen dwarves. One of the things he teaches them is the notion of an *arithmetic sequence*. A sequence of numbers, $x_1, ..., x_n$, is an arithmetic sequence if the differences between successive terms in the sequence are the same. In other words, the sequence is an arithmetic sequence if: $x_2 - x_1 = x_3 - x_2 = ... = x_n - x_{n-1}$.

Bilbo and Ori, one of the dwarves, start playing a game to develop their skills. One of them suggests a sequence of numbers, and the other has to quickly tell whether the sequence is an arithmetic sequence, and if yes, then he also has to list the next 5 numbers in the sequence. You are to write a program to help them make sure their answers are correct.

Input/Output Format:

Input: The first line in the test data file contains the number of test cases (< 100). After that, each line contains one test case: the first number is the number of entries, n, in the sequence (provided as an `int`), and the next n numbers are the sequence itself.

Output: For each test case, you are to either output that the sequence is not an arithmetic sequence, or you are to output the next 5 numbers in the sequence. The exact format is shown below in the examples.

Note: We have provided a skeleton program that reads the input and prints the output. All you need to do is provide the body of the following procedure:

```
private static Arraylist<Integer> solveArithmeticSequence(ArrayList<Integer> sequence)
```

The procedure should return the next 5 numbers as an "Arraylist<Integer>", or it should return null (if the sequence is not an arithmetic sequence).

Examples:

The output is shown with an extra blank line so that each test case input is aligned with the output; the first blank line should not be present in the actual output.

Input:	Output:
3	
3 1 2 3	The next 5 numbers after [1, 2, 3] are: [4, 5, 6, 7, 8]
5 4 8 12 16 20	The next 5 numbers after [4, 8, 12, 16, 20] are: [24, 28, 32, 36, 40]
4 1 2 3 5	The sequence [1, 2, 3, 5] is not an arithmetic sequence.

Practice 2 – Shift Letters

Bilbo wants to send a message to the dwarves, currently being held prisoners by the Great Goblin. He has a mechanism to get it to the dwarves, but there is a chance that the orcs may intercept the message. So he decides to encrypt the message using a very simple scheme (sufficient to fool the orcs, who are not known for their IQs).

Specifically, he decides to shift the letters in every word in the message toward the right (wrapping around appropriately) by a small number. For example, shifting "oakenshield" by 3 gives him "eldoakenshi", whereas shifting "gandalf" by 2 gives him "lfganda". He chooses to shift every word by a different number of letters, always less than the number of letters in the word.

You are to write a program to help Bilbo encode his message.

Input/Output Format:

Input: The first line in the test data file contains the number of test cases (< 100). After that, each line contains one test case, a word, w, (provided as a `String`) followed by an integer, n (`int`). You can assume that: $0 < n < length(w)$.

Output: For each test case, you are to output the word formed by shifting every letter in w to the right by n, and wrapping around to the beginning appropriately. The exact format is shown below in the examples.

Note: We have provided a skeleton program that reads the input and prints the output. All you need to do is provide the body of the following procedure:

```
private static String solveShiftLetters(String s, int shiftBy)
```

Examples:

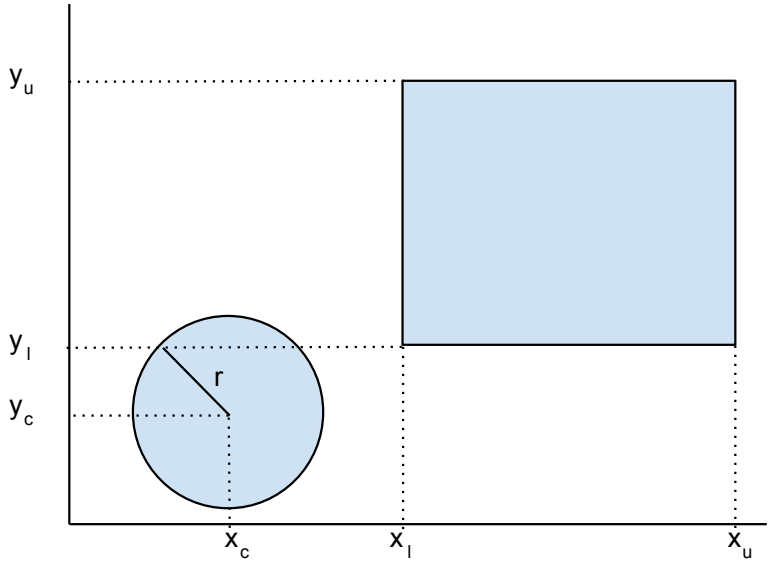
The output is shown with an extra blank line so that each test case input is aligned with the output; the first blank line should not be present in the actual output.

Input:	Output:
3	
hobbit 2	Shifting hobbit by 2 positions gives us: ithobb
unexpected 3	Shifting unexpected by 3 positions gives us: tedunexpec
journey 4	Shifting journey by 4 positions gives us: rneyjou

Practice 3 – Hobbiton Farms

The HOA (Home Owners Association) of Hobbiton, the village in the Shire where Bilbo lives, is trying to create a new map showing precisely the boundaries of the houses in Hobbiton. But the unique geometry of Hobbiton is making this complicated, since a land parcel (on which a house is built) may either be a circle or a rectangle.

Before figuring out the boundaries and drawing the new map, the Hobbiton HOA would like to make sure that all its records are correct, and specifically that no two land parcels overlap with each other. It knows how to decide if two land parcels overlap if both of them are circles, or if both of them are rectangles. However, it needs your help when one of the land parcels is a rectangle and the other one is a circle.



Input/Output Format:

Input: The first line in the test data file contains the number of test cases (< 100). After that, each line contains one test case. Each test case comprises 7 numbers (of the type `double`). The first three numbers denote the circle, specifically, *radius*, *x-coordinate of the center*, *y-coordinate of the center*. The next four numbers represent the rectangle, specifically, the $x-$ and $y-$ coordinates of the lower left and upper right corners. More precisely, from the figure above, the 7 numbers are: $r, x_c, y_c, x_l, y_l, x_u, y_u$. You can assume that all the numbers are positive.

Output: For each test case, you are to output whether the circle and the rectangle overlap. The exact format is shown below in the examples.

Note: We have provided a skeleton program that reads the input and prints the output. All you need to do is provide the body of the following procedure:

```
private static boolean solveOverlap(Circle c, Rectangle r)
```

Examples:

The output is shown with an extra blank line so that each test case input is aligned with the output; the first blank line should not be present in the actual output.

Input:	Output:
4	
10.0 50.0 50.0 55.0 55.0 56.0 56.0	The given circle and rectangle overlap.
20.0 50.0 50.0 55.0 55.0 56.0 56.0	The given circle and rectangle overlap.
10.0 50.0 50.0 30.0 30.0 31.0 31.0	The given circle and rectangle do not overlap.
10.5 50.5 50.5 40.0 40.0 41.0 41.0	The given circle and rectangle do not overlap.

Contents

1 Hand Detection

Hobbits like games of chance and skill. As such they play a game very much like our poker, though their decks of cards and rules are a bit different.

- Cards can have one of 21 values (1 to 21 inclusive).
- Cards can have one of 5 suits (Elf, Man, Hobbit, Ent, Orc).

A deck of cards therefore has 105 cards. Each "hand" has 5 cards, just like poker.

Your task is to write a program to identify two types of hands: a "spread" and a "rainbow".

The rules for a "spread" are as follows: the suits don't matter but the values do. The values must be such that the difference between any two card values is not the same as the difference between any other two card values.

For example, a hand with containing cards with values 1, 2, 4, 8, 16 is a spread because any pair of values has a unique difference:

First Card	Second Card	difference
1	2	1
1	4	3
1	8	7
1	16	15
2	2	2
2	4	6
2	8	14
4	8	4
4	16	12
8	16	8

The rule for a "rainbow" is simply that the hand must have exactly one card of each suit.

Input/Output Format:

Input: The first line in the test data file contains the number of test cases (< 100). After that each line will contain a test case, with each card being represented by two integers (thus each test case comprises of 10 integers). The first number is the card's suit (an `int` that takes values between 0 and 4, inclusive), and the second represents the card's suit (an `int` that takes values between 1 and 21, inclusive).

Output: For each test case, the program needs to indicate whether or not that hand of cards has a spread and whether or not that hand of cards has a rainbow. The exact format is shown below.

Note: We have provided a skeleton program that reads the input and prints the output based on two boolean methods that you need to implement:

```
private static boolean detectSpread(Card[] handOfCards)
private static boolean detectRainbow(Card[] handOfCards)
```

Examples:

The output is shown with an extra blank line so that each test case input is aligned with the output; the first blank line should not be present in the actual output.

Input:	Output:
2	
0 1 0 2 0 4 0 8 0 16	Have a spread. Do not have a rainbow.
0 1 1 1 2 1 3 1 4 1	Do not have a spread. Have a rainbow.

2 Anagrams

Gandalf has been trying to read ancient Elvish texts to better understand the history of the One Ring. One of the texts that he has un(middle)earthed looks promising; however, an enchantment appears to have scrambled the letters in each word. For example, "sauron" may have been scrambled to "uosanr", "isildur" may have been scrambled to "druiisl" and son on. The number of letters and the frequency of letters in a word remains unchanged. Two such strings are called *anagrams*.

Help Gandalf decipher the text by giving him an oracle that can tell if any two words are anagrams.

Input/Output Format:

Input: The first line in the test data file contains the number of test cases (< 100). For each such test case there are two words on a line, separated by a space. Each word contains less than 100 characters.

Output: For each test case, you are to output whether the words are anagrams, or not. The exact format is shown below.

Note: We have provided a skeleton program that reads the input and prints the output. All you need to do is provide the body of the following procedure:

```
private static boolean solveAnagrams(String first, String second)
```

Examples:

The output is shown with an extra blank line so that each test case input is aligned with the output; the first blank line should not be present in the actual output.

Input:	Output:
3	
blather reblath	blather & reblath are anagrams.
maryland landam	maryland & landam are NOT anagrams.
bizarre brazier	bizarre & brazier are anagrams.

3 Guessing Game I

After a long trek through treacherous mountains, Bilbo and the dwarfs have arrived at the West Gate of the mines of *Puzzlia*, constructed by a mathematically inclined Dwarf King. The gate is of course not open (nothing can be that easy for our heroes), and Dwalin explains to Bilbo that they need to guess a 4-digit number (let's call it "secret") correctly to open the gate. He points out 8 geometrical shapes on top of the gate, 4 circles and 4 squares. Every time a guess is made (by standing in front of the gate and speaking each of the digits loudly), some of the squares and the circles will light up. If the guess is correct, then the gate will open; but if the number is not guessed correctly in 10 attempts, then... well, best not to think of such things.

Dwalin explains the rules about how the circles and squares will light up. Let *guess* denote a guess that is made. For every digit in *guess*: (1) if it is equal to the corresponding digit in *secret* (at the same position), then a circle will light up; (2) if it is equal to some digit in *secret* but at a different position, a square will light up. For example:

- **secret =** 1234, **guess =** 1357: The "1" in *guess* gives us one circle, whereas the "3" in *guess* gives us one square. The other two digits in *guess* do not have a match.

The secret (and the guess) may contain duplicate digits. The rule for handling duplicates is simple: a digit in secret can only match one digit in guess (and vice versa), and exact matches supersede out-of-position matches. So we first check for exact matches, and then look for the second type.

- **secret =** 1234, **guess =** 1122: There is only one exact match, for position 1 (that gives us one circle). The second "1" in guess doesn't match anything in secret (since the "1" in secret is already matched). Then, we get one square for the first "2" in guess, and that's it.
- **secret =** 1311, **guess =** 1122: One circle for exact match at position 1, and one square for an out-of-position match for digit "1".
- **secret =** 0011, **guess =** 0213: Two circles, no squares.

Before trying to guess the secret of the gate, Bilbo would like to make sure he understands the process and develops some strategy. He would like your help with this. Specifically, he would like you to write a program that, given two numbers, tells him how many circles and squares would light up.

Input/Output Format:

Input: The first line in the test data file contains the number of test cases (< 100). After that, each line contains one test case: the first number is the *secret* (provided as an `int`), and the following number is the *guess* (provide as an `int`). Both the numbers are ≤ 9999. Note that the numbers need to be padded with 0's. For example, if one of the numbers is 1, it needs to be treated as 0001.

Output: For each test case, you are to output the numbers of circles and squares that will light up. Exact format shown below.

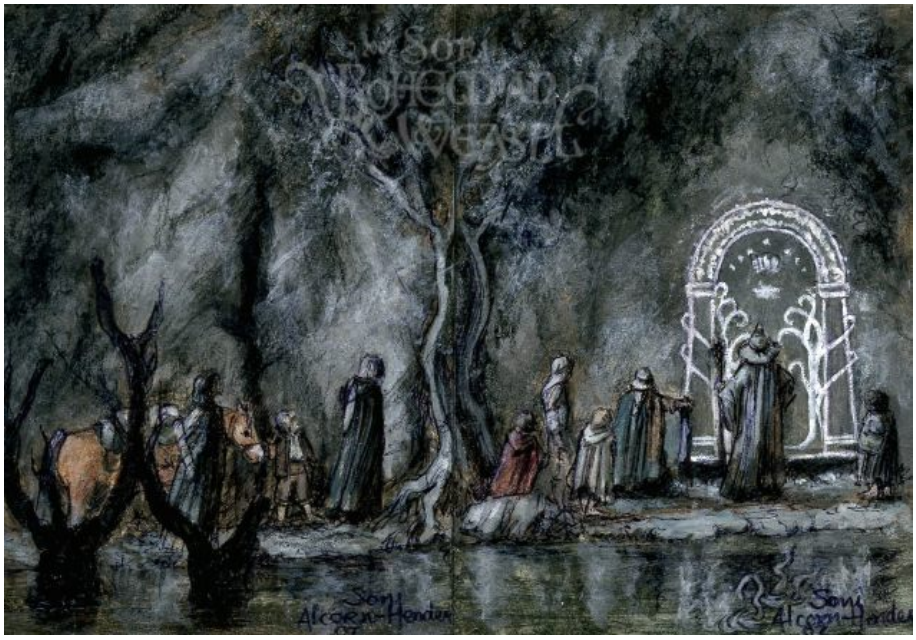
Note: We have provided a skeleton program that reads the input and prints the output. All you need to do is set the appropriate variables in the body of the following procedure:

```
private static void solveGuessingGameI(int secret, int guess)
```

Examples:

The output is shown with an extra blank line so that each test case input is aligned with the output; the first blank line should not be present in the actual output.

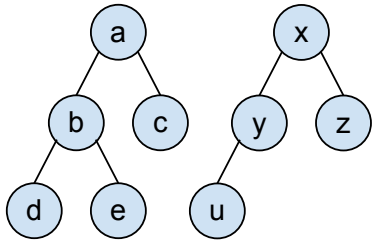
Input:	Output:
5	
1234 1111	For secret = 1234 and guess = 1111, 1 circles and 0 squares will light up.
5678 5678	For secret = 5678 and guess = 5678, 4 circles and 0 squares will light up.
4444 4444	For secret = 4444 and guess = 4444, 4 circles and 0 squares will light up.
1234 1211	For secret = 1234 and guess = 1211, 2 circles and 0 squares will light up.
1122 2211	For secret = 1122 and guess = 2211, 0 circles and 4 squares will light up.



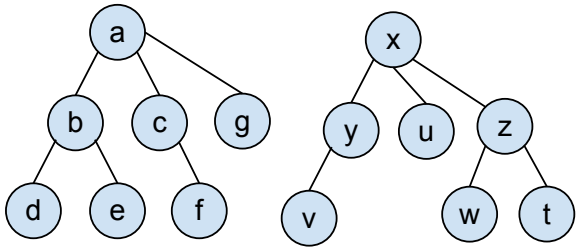
4 Tree Isomorphism

Bilbo meets the love of his life, Oblib, but she seems eerily familiar and he worries that they are related. They both know their ancestry, but can't just compare their ancestry trees because Bilbo knows his ancestor names in the male form, and Oblib knows hers in the female form, and the two forms are not related. Even if they were brother and sister, the names they know their ancestors by would all be different!

Bilbo and Oblib have written out their family trees. Your task is to determine if the two trees are isomorphic, i.e., if there is a one-to-one mapping between the two, ignoring the names. Assume that the trees are rooted, i.e., their roots are fixed. As an example, the two trees on the left below are not isomorphic to each other – there is no way to map the two children of a to the two children of x, since one child of a must have 2 children, but none of the children of x do.



Non-isomorphic Trees



Isomorphic Trees

On the other hand, the two trees on the right are isomorphic. The mapping is: $a \leftrightarrow x$ (the roots must always be mapped to each other), $b \leftrightarrow z$, $c \leftrightarrow y$, $g \leftrightarrow u$, $d \leftrightarrow w$, $e \leftrightarrow t$, $f \leftrightarrow v$. There is one other isomorphism (that swaps the mappings for d and e).

Input/Output Format:

Input: The first line in the test data file contains the number of test cases (< 100). For each such test case there are two lines, each describing a tree. An input line describes the tree by listing node labels encountered in a pre-order traversal, starting at the root. In other words, the trees are traversed from root to leaves and from left to right, and the label of every node along the way is output. After each set of children, there is a hash mark ('#'). All node labels and hash marks are separated by spaces. The example below shows the pre-order traversals for the two examples above.

Output: For each test case, you are to output "Isomorphic." if the trees are isomorphic, or "Not isomorphic." if not, followed by a newline.

Note: We have provided a skeleton program, but you need to provide routines that read in the trees, and compare them. Specifically, you need to provide the body of the following routine:

```
private static boolean solveTreeIsomorphism(String one, String two)
```

Examples:

The output is shown with extra blank lines so that each test case input is aligned with the output; the first blank line should not be present in the actual output.

Input:	Output:
2 a b d # e # # c # # x y u # # z # # a b d # e # # c f # # g # # x y v # # u # z w # t # # #	The two trees are not isomorphic. The two trees are isomorphic.

5 Guessing Game II

While Bilbo has been developing a strategy for guessing the secret to the West gate of Puzzlia, Bombur the dwarf is getting antsy, and thinking "how hard can it be", stands up in front of the door, and starts making guesses at random. Before anyone can stop him, he has already made 8 guesses (unsuccessfully as you might expect). Dwalin was able to write down some of the guesses (but not all) and the gate's responses to those guesses. For example, if the guess was 1234, and the gate's response was to light 1 circle, Dwalin recorded it as: $\langle 1234 :\ 1\ Circle \rangle$.

Since the 10th guess must be the correct one, there is now only one guess that can be made with which to find more information about the secret number. Bilbo would like your help in that. Specifically, given up to 8 guess-response pairs, you are to first decide if there is already sufficient information to identify the secret correctly. If yes, you are to output the secret. However, if not: you are also to decide whether you can use the remaining guess to elicit sufficient information to identify the secret.

For example, if Dwalin had only recorded one guess-response pair, say $\langle 1234 :\ 1\ Circle \rangle$, there are too many possibilities left to be able to identify the correct secret with just one more guess. On the other hand, given the guess-response pairs: $\langle 5888 :\ 3\ Circles \rangle$, $\langle 1234 : \rangle$, $\langle 4567 : \rangle$, we can deduce that there are only three possibilities for the secret word: 0888, 8888, and 9888. There are several guesses that can uniquely identify the secret. For example, if we were to guess 0988 next, we will get different answers depending upon the secret. If the secret were 0888, we would get $\langle 0988 :\ 3\ circles \rangle$, but for the other two, we would get: $\langle 0988 :\ 2\ circles \rangle$, and $\langle 0988 :\ 2\ circles,\ 1\ square \rangle$ respectively. Thus, the response of the gate would enable us to identify the secret. Note that, none of the remaining possibilities are good guesses in this situation: if we were to guess 0888, and if the gate's response were 3 circles, we wouldn't know the secret.

There are several other guesses that can uniquely identify the secret, e.g., 0898, 0889, 0009. In such cases, you should output the one that is numerically the smallest. In this case, the output should be 0009.

Input/Output Format:

Input: The first line in the test data file contains the number of test cases (≤ 1000). After that, each line contains one test case: the first number is the number of guess-response pairs recorded, n ($0 < n \leq 8$), and the next $3n$ numbers denote the triples: $\langle guess, num\_circles, num\_squares \rangle$. Note that the guesses are implicitly padded with 0's. For example, if one of the guesses is 1, it needs to be treated as 0001.

Output: For each test case, you need to distinguish between three cases: the given guess-response pairs are sufficient to deduce the secret correctly (`CASE_SUFFICIENT_TO_GUESS_SECRET`), there is one more guess that can uniquely identify the secret (`CASE_ONE_ADDITIONAL_GUESS_SUFFICIENT`), or there is not sufficient information to find the secret with one more guess (`CASE_INSUFFICIENT_INFORMATION`). In the first and second cases, the secret and the guess should be output respectively. If there are multiple choices for the guess, you should output the one that is numerically the first. Exact format shown below.

Note: We have provided a skeleton program that reads the input and prints the output. All you need to do is to complete the body of the following procedure:

```
private static int[] solveGuessingGameII(GuessResponse[] rounds)
```

The procedure returns an array of 2 integers: in the first integer, you should note which of the three cases applies, and the second integer should be the secret or the guess accordingly (if the first integer is `CASE_INSUFFICIENT_INFORMATION`, the second integer is ignored).

Examples:

The output is shown with an extra blank line so that each test case input is aligned with the output; the first blank line should not be present in the actual output.

Input:	Output:
3 1 1234 1 0 3 5888 3 0 1234 0 0 4567 0 0 4 5888 3 0 1234 0 0 4567 0 0 0009 0 0	 There is no guess that can uniquely identify the secret. There is a guess that can uniquely identify the secret: 0009 The secret is: 8888

6 Ent Numbers

Ents are a race of beings in Middle-Earth who closely resemble trees, and are ancient shepherds of the forest. When Bilbo visits the forest of Fangorn to meet with the Ents, he notices very very long numbers in their texts, and asks Treebeard (eldest of the Ents) about them. Treebeard explains that the numbers refer to the convergence rates of certain sequences (that were introduced to the Ents by a mathematician named Reuben Goodstein). After a few days of explanation, Bilbo understands the basic process through which such numbers were generated.

Consider a sequence defined by two numbers, N and B. The first number in the sequence is N itself. However, to get the next number, we decrease the number (N) by 1, but we increase the base B by 1.

Say we start with $N = 42$ and $B = 10$. We will write it as `(4)(2) Base 10`, parenthesizing the digits as shown. Hence the next number will be: `(4)(1) Base 11`, which is `45 Base 10`. We get this by first reducing the number by one (which gives `(4)(1) Base 10`) and then increasing the base by 1 (while keeping the digits in the number the same). The next number will be: `(4)(0) Base 12`, which is `48 Base 10`. The table below shows how the sequence continues.

B	N Base B	N Base 10	Explanation
10	(4)(2)	42	Sequence is defined by the two starting numbers
11	(4)(1)	4 * 11 + 1 = 45	
12	(4)(0)	4 * 12 + 0 = 48	
13	(3)(11)	3 * 13 + 11 = 50	Note that, we subtract 1 first (in the previous base) and then increase the base.
14	(3)(10)	3 * 14 + 10 = 52	
15	(3)(9)	3 * 15 + 9 = 54	

Although it seems like the sequence will continue to increase like this, it actually eventually converges to zero at base B = 192 (i.e., the sequence goes to `(1) Base 191`, and then to `(0) Base 192`). The table below shows the sequence for another starting point, and another base. As you can see, the sequence can grow very very rapidly (at Base 100, the sequence reaches 1000707070428).

B	N Base B	N Base 10
2	(1)(1)(0)(0)(1)(0)(0)	100
3	(1)(1)(0)(0)(0)(1)(1)	976
4	(1)(1)(0)(0)(0)(1)(0)	5124
5	(1)(1)(0)(0)(0)(0)(3)	18753
6	(1)(1)(0)(0)(0)(0)(2)	54434
7	(1)(1)(0)(0)(0)(0)(1)	134457

Treebeard explains to Bilbo that this sequence always converges to zero, even though it may first increase to very large numbers. Bilbo finds this hard to believe, and would like your help to understand this better. Specifically given two starting numbers N and B, he would like you to tell him if the sequence converges to 0 before the base reaches 2^{60} (which is pretty much the largest number he can imagine).

Input/Output Format:

Input: The first line in the test data file contains the number of test cases (< 100). For each such test case there are two integers on a line, N and B, separated by a space.

Output: For each test case, you are to decide whether the sequence converges to 0 before the base reaches 2^{60}. If yes, you are to output the base at which it reaches zero. Exact output format is shown below.

Note: We have provided a skeleton program that reads the input and prints the output. All you need to do is to complete the body of the following procedure:

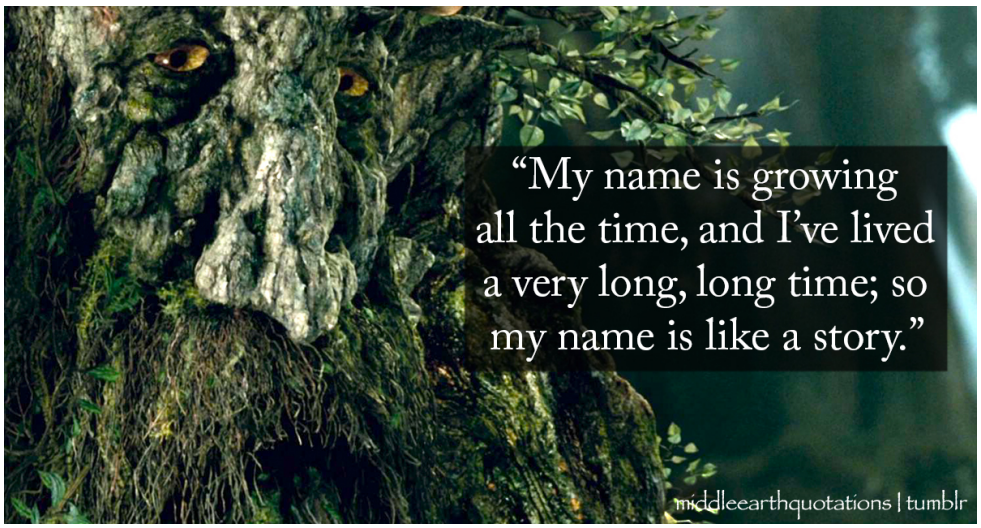
```
private static long solveGoodstein(long N, long B)
```

The procedure should return the base at which the sequence converges to 0, or -1 if it doesn't before the base reaches 2^{60}.

Examples:

The output is shown with an extra blank line so that each test case input is aligned with the output; the first blank line should not be present in the actual output.

Input:	Output:
5	
42 10	Starting with N = 42 and B = 10, sequence converges to 0 at Base 192.
100 2	Starting with N = 100 and B = 2, sequence did not converge to 0 before 2^60.
12 10	Starting with N = 12 and B = 10, sequence converges to 0 at Base 24.
15 8	Starting with N = 15 and B = 8, sequence converges to 0 at Base 30.
20 5	Starting with N = 20 and B = 5, sequence converges to 0 at Base 80.



7 King Thrór's Gold Problem

The Dwarf king Thrór wants to give some of his bars of gold to reward his k best warriors. The dwarf unit of currency is called a Mirian. Thrór has n different bars whose values in Mirians are $\{v_1, \ldots, v_n\}$. He wants to give away a total target value of T Mirians to these warriors by handing out k gold bars. Not all of his warriors are equally deserving, and so they will generally receive bars of different values.

Thrór asks his brother Frór to help him to determine all the different ways in which he can form T as the sum of the values of k bars from his stock. For example, suppose that he has $n = 10$ bars of values $V = \{1, 2, 4, 5, 10, 11, 13, 15, 17, 19\}$ Mirians, respectively. He wants to distribute $k = 3$ bars, and he wants to give a target amount of $T = 20$ Mirians. Frór determines that there are four different possible ways to do this:

$$1 + 2 + 17 \qquad 1 + 4 + 15 \qquad 2 + 5 + 13 \qquad 4 + 5 + 11.$$

If, instead, there had been $k = 4$ warriors, there would be only two solutions:

$$1 + 2 + 4 + 13 \qquad 1 + 4 + 5 + 10$$

You are to write a program to help brother Frór determine the number of solutions. Given the target amount T, the number of warriors k, and the values of bars $V = \{v_1, \ldots, v_n\}$, your program is to output the number of different ways of forming T as the sum of k different values from V. If the number of solutions is 20 or fewer, your program should also list all the solutions in lexicographically increasing order. That is, all solutions involving the smallest bar are listed first, among those, solutions with the second smallest bar are listed first, and so on.

Input/Output Format:

Input: The first line of the input file contains the number of test cases (≤ 100). Each case has the same format:

- A line containing T, the desired target value, and k, the number of warriors. (You may assume that $T \leq 500$ and $k \leq 50$.)
- A line containing n, the number of gold bars. (You may assume that $n \leq 100$.)
- The next n positive integers represent the values of the bars $v_1, \ldots, v_n$. (You may assume that the values are distinct and are given in increasing order.)

We have provided a main program that will read and print a summary of the input. You need only provide the body of the following function, which computes and outputs the answer. The parameter `target` represents T, the parameter `nWarriors` represents k, and the parameter `V` is an n element array of values given in increasing order.

```
private static void goldSum(int target, int nWarriors, int[] V)
```

You may assume that the number of solutions will fit into a single long integer, i.e., it will be less than $2^{63} - 1$.

Output: After computing the number of possible solutions, your program will output the number of solutions on a single line (preceded by the string "Number of solutions = "). If this number is less than or equal to 20, it will then output each solution in lexicographically increasing order on a separate line, with the values separated by white spaces (blanks or tabs).

Examples:

The input and output are shown with extra blank lines so that first line in each test case input is aligned with the corresponding output; the blank lines should not be present in the actual output.

Input:	Output:
2	
20 3	Test case 1
10	Target sum = 20
1 2 4 5 10 11 13 15 17 19	Number of warriors = 3
	V = 1, 2, 4, 5, 10, 11, 13, 15, 17, 19
	Number of solutions = 4
	1 2 17
	1 4 15
	2 5 13
	4 5 11
100 9	Test case 2
19	Target sum = 100
1 2 4 5 6 8 9 10 12 13 15	Number of warriors = 9
16 17 18 19 20 21 23 24	V = 1, 2, 4, 5, 6, 8, 9, 10, 12, 13, 15, 16, 17, 18, 19, 20, 21, 23, 24
	Number of solutions = 1491

8 Maximum Damage

The orcs and the elves have been fighting for a long time, and the war between them has come to a critical juncture. The elves have detected all the locations for the orc camps, and marked them on a map of the battle ground. To simplify their calculations, elves have laid out a coordinate grid over the battleground, and they treat each unit square as a point (with integer coordinates). So the location of each orc camp is given by a pair of integers, denoting the x- and y-coordinates. The elves would like to build a limited number of Elf stations to attack those camps. Each of the stations is capable of inflicting 1 unit of damage to each of the orc camps within a distance R (from the station). Thus the total damage that a single Elf station can inflict is given by the number of orc camps within distance R from it.

However, "distance" between two points on the battle ground is not the standard straight-line distance between them. This is because the horse trails that the elves would use to launch the attacks are laid horizontally and vertically, in a grid-like manner. So, given two locations P_1 and P_2 on the battle ground , the distance between them is:

$$\mathbf{D}(P_1, P_2) = |P_1.x - P_2.x| + |P_1.y - P_2.y|$$

where $P_1.x$ and $P_1.y$ denote the x- and y-coordinates of P_1. Given the above discretization assumption, all the coordinates are positive integers.

The elves have also noticed that parts of the battle ground are too rugged to build an Elf station on – they call such locations "obstacles". With all the information in hand, the elves would like to identify the ideal locations to build a given number T of Elf stations, so as to inflict the maximum possible damage on the orc camps.

The battle ground is described by a rectangular map where each location is annotated using one of three: '*', 'O' and 'X'. An asterisk '*' represents a blank space (a location at which an Elf station can be built), a letter 'O' represents an orc camp and a letter 'X' means an obstacle (a location that none of the Elf stations can be built on). A legal battleground map is shown below.

```
**O**
*OXO*
*****
**O**
```

Input/Output Format:

Input: The first line in the test data file contains the number of test cases (< 15). Each of the test cases is continued one after another.

For each test case, the first line contains 4 integers: N, M, T, R. The first two, N and M, indicate the size of the battleground along the y and x axes, respectively. T represents the number of Elf stations that can be placed on the map. R represents the maximum attack distance for each Elf station.

Following N lines, each of which contains a string of M characters, represents the map of the battleground.

Assume: $1 \leq N, M \leq 1000, 1 \leq T \leq 10^6, 1 \leq R \leq 10^8$.

Output: For each test case, you are to output one line containing only one integer, representing the maximum harm that can be inflicted upon the orc camps.

Note: We have provided a skeleton program that reads the input and prints the output. All you need to do is provide the body of the following procedure:

```
private static BigInteger solveMaximumDamage(
    String[] mapOfKingdom, int numMaxElfStations, int radiusOfAttacks)
```

The procedure should return the maximum damage in a BigInteger.

Examples:

The output is shown with extra blank lines so that first line in each test case input is aligned with the corresponding output; the blank lines should not be present in the actual output.

Input:	Output:
2	
3 3 2 1	Maximum damage = 4
O	
OXO	
***	
4 5 3 2	Maximum damage = 8
O	
OXO	
*****	
O	

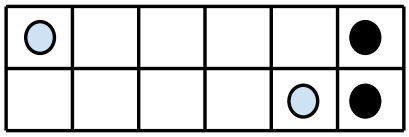
9 Checker Board

After capturing Gandalf, Bilbo, and the dwarfs while they were trying to make their way through the Misty Mountains, the Great Goblin makes Gandalf an offer. The two of them play a game that the goblin has devised. If Gandalf wins, they can all go free, but if the Great Goblin wins, then Gandalf must help him recover the treasures of Moria. Your goal is to help Gandalf decide whether there is a way he can win the game.

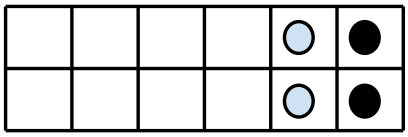
The game is played on a rectangular checkerboard (the size of the board varies), that contains white and black checkers (corresponding to the two players) in a given arbitrary placement. In each row, there can be at most one white checker and at most one black checker. Starting with the white player, the two players take turns sliding their checkers. In each turn, a player can choose one of his checkers and slide it along its row (there is no movement across rows) to any empty position; however, you cannot jump any other checker (if there is another checker on the board) or go outside of the board.

The player who cannot slide on his turn loses the game.

As an example, consider the initial placement as shown in the figure on the left below. With the first move, the first player can slide the white checker in the first row all the way to the right, thus preventing the second player from being able to move, and winning. On the other hand, in the second game, though it looks highly favorable to the first player, he loses because each of his moves creates a move for the second player, and eventually all his white checkers will get blocked on the other (left) side of the board.



White can win by moving first checker all the way to the right



Every move by white creates a move for black, and black eventually wins

Input/Output Format:

Input: The first line in the test data file contains the number of test cases (≤ 20). Each of the test cases is continued one after another. Each test case begins with 4 numbers M (the number of rows), N (the number of columns), P (number of white checkers), and Q (number of black checkers) on a line. Assume that $0 \leq M, N \leq 300$, and $0 \leq P, Q \leq M$. Next P lines describe the positions of the white checkers: the i^{th} line contains two numbers, $1 \leq r_i \leq M, 1 \leq c_i \leq N$, that describe the position of i^{th} white checker. Thus the i^{th} checker is located in r_i^{th} row, in the c_i^{th} column. Similarly the next Q lines describe the positions of the black checkers.

Output: Print "W" if the player with the white checkers can win the game (assuming the player makes the best possible moves), and print "B" if other player can win the game. Print "T" if the game can continue forever without either player being able to force the other player to lose.

Note: We have provided a skeleton program that reads the input and prints the output. All you need to do is provide the body of the following procedure:

```
private static char solveCheckerBoard(int M, int N, int[][] white, int[][] black)
```

The procedure should return a character 'W', 'B' or 'T' representing the result of the game. 'white' is a $P \times 2$ integer array the first column of which represents the checker's row number and the second represents its column number. 'black' is a $Q \times 2$ integer array the first column of which represents the checker's row number and the second represents its column number.

Examples:

The output is shown with extra blank lines so that first line in each test case input is aligned with the corresponding output; the blank lines should not be present in the actual output.

Input:	Output:
2	
2 6 2 2	W
1 1	
2 5	
1 6	
2 6	
2 6 2 2	B
1 5	
2 5	
1 6	
2 6