Problem A. Cat's Age

Source file name:	Catsage.c, Catsage.cpp, Catsage.java, Catsage.py
Input:	`Standard`
Output:	`Standard`

There are several proposals for converting a cat's age to human age. One such approach is as follows:

1. The first year of cat's age is equivalent to 15 years of human age.
2. The second year of cat's age is equivalent to 9 years of human age.
3. Every year after the second year of cat's age is equivalent to 4 years of human age.

For example, if a cat is 13-year old, the cat would be $(15) + (9) + [(13 - 2) \cdot 4)] = 68$-year old in human age. Note that the above approach implies:

1. If a cat is less than one-year old, each month of cat's age will be equivalent to 15 months of human age. For example, if a cat is 5-month old, the cat would be $(5 \cdot 15) = 75$-month old in human age, which is 6-year-and-3-month old.
2. If a cat is older than one year but less than two-year old, each month of cat's age after the first year will be equivalent to 9 months of human age. For example, if a cat is 1-year-and-7-month old, the cat would be (15 years) + [(7 · 9) months] = (15 years) + [63 months] = 20-year-and-3-month old in human age.
3. If a cat is beyond 2-year old, each month of cat's age after the second year will be equivalent to 4 months of human age. For example, if a cat is 5-year-and-7-month old, the cat would be (15 years) + (9 years) + [(5 − 2) · 4 years] + [(7 · 4) months] = 38-year-and-4-month old in human age.

Given a cat's age, convert it to human age.

Input

There is only one input line; it contains two integers: y $(0 \leq y \leq 20)$, indicating the year part of the cat's age and m $(0 \leq m \leq 11)$, indicating the month part of the cat's age.

Output

Print two integers indicating the cat's age in human age (years and months). Note that the output value for months should not exceed 11.

Example

Input	Output
13 0	68 0
0 5	6 3
1 7	20 3
5 7	38 4
20 11	99 8

Problem B. Overall Ranking

Source file name:	Overall.c, Overall.cpp, Overall.java, Overall.py
Input:	`Standard`
Output:	`Standard`

The Regional Competitions of ICPC (International Collegiate Programming Contest) allow universities to enter more than one team in the contest. The scoresheet for the contest will list the ranking for each team. For example, if UCF has three teams, GT two teams, and Auburn four teams, the final team ranking may look like:

1. Auburn
2. GT
3. UCF
4. UCF
5. UCF
6. Auburn
7. Auburn
8. GT
9. Auburn

The regional contest does not show the "overall performance" of each university which is the average of the team rankings for the university. For the above scoresheet, Auburn has the overall performance of "$(1 + 6 + 7 + 9)/4 = 5.75$"; UCF has the overall performance of "$(3 + 4 + 5)/3 = 4.0$"; and GT has the overall performance of "$(2 + 8)/2 = 5.0$". So, the overall ranking of the universities will be:

1. UCF
2. GT
3. Auburn

Note that lower average indicates better performance, hence higher university ranking.

Assume that the universities will not have the same overall performance, i.e., no need for tiebreaker.

Given the scoresheet for a contest with each university having one or more teams, find the university ranking based on the overall performances.

Input

The first input line contains an integer, n ($2 \leq n \leq 100$), indicating the number of teams in the regional contest. The teams are listed on the following n input lines, one per line, in the order of team ranking at the regional. Each team name will be 1 to 20 letters (uppercase and/or lowercase).

Assume there will be at least two universities in the input.

Output

Print the universities, one per line, in the order of overall performance.

Example

Input	Output
9 Auburn GT UCF UCF UCF Auburn Auburn GT Auburn	UCF GT Auburn
7 UofX UofA UofC UofB UofA UofB UofY	UofX UofC UofA UofB UofY
8 UA UB UA UA UA ua Ua uA	UB UA ua Ua uA

Problem C. Don't Complicate It!

Source file name:	Complicate.c, Complicate.cpp, Complicate.java, Complicate.py
Input:	`Standard`
Output:	`Standard`

The number of parentheses and the level of nesting are usually a good indication of how complicated an expression is.

Given a string containing only "(", ")" and blanks (spaces), compute its complexity. The complexity is defined as follows:

- Each innermost set of parentheses adds 1 to the total complexity.
- Each set of parentheses containing only innermost set of parentheses adds 2 to the total complexity, i.e., one level out will add 2 to the complexity.
- Each set of parentheses at the next level out adds 3 to the complexity, and so on.

Input

There is only one input line; it contains a valid expression. The expression (string) will be 2-60 characters, each character being either "(", ")" or blank (space). Note that the blanks can be anywhere but the string will not exceed 60 characters (including the blanks). Assume that there will be at least one set of parentheses in the input, i.e., the input will not be all blanks.

Note that a valid expression satisfies the following:

1. All the parentheses are matched, i.e., every opening parenthesis has a corresponding closing parenthesis.
2. The matched parentheses are in the correct order, i.e., an opening parenthesis comes before its corresponding closing parenthesis.

Note again that the input is a valid expression, i.e., you do not need to check for errors.

Output

Print the complexity of the expression.

Example

Input	Output
`()()()()()`	`5`
`       (    ()   ( )  ()  (      )    )`	`6`
`(     (())      ()    (())         )`	`10`
`    ((()))   ()   (   (())  ()   (())  )`	`17`
`(((((((((((((((((((())))))))))))))))))))`	`210`

Explanation

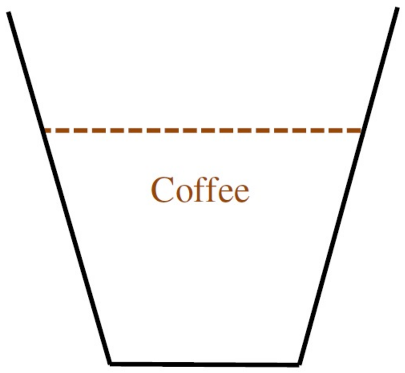
The last Example Input has parentheses nested 20 levels.

Problem D. How Much Coffee is Left?

Source file name:	Coffee.c, Coffee.cpp, Coffee.java, Coffee.py
Input:	`Standard`
Output:	`Standard`

As a budding computer scientist, you've realized that it's very important to have your morning coffee. Furthermore, it's good to know when the coffee will run out so you can properly plan when to refill.

Your coffee mug is the shape of a cross section of a cone. Here is a horizontal view:



You start the morning with a full cup. After some amount of time, you note the depth of the coffee remaining - this is the distance from the bottom of the cup to the surface of the coffee (note that the surface of the coffee is a plane parallel to the bottom of the cup). Given this information, calculate how much more time it will take before the coffee is finished, assuming that you drink coffee at the same constant rate the whole time.

Given the information about the cup and elapsed drinking time, determine how many minutes it will take to finish the coffee.

Input

There is only one input line; it contains five space separated integers:

- r $(1 \le r \le 100)$, the radius of the bottom of the cup in inches,
- s $(r < s \le 1000)$, the radius of the top of the cup in inches,
- h $(2 \le h \le 1000)$, the height of the cup in inches,
- m $(1 \le m \le 1000)$, the number of minutes that coffee has been drank,
- d $(1 \le d < h)$, the depth of the remaining coffee in inches.

Output

Print a single floating point number on a line by itself: the number of minutes it will take to finish the cup of coffee. Any answer within an absolute tolerance of 10-6 will be accepted.

Example

Input	Output
`3 7 5 15 4`	`30.284316723`
`10 50 100 47 50`	`12.469387755`
`5 6 12 30 7`	`35.081603090`

Problem E. Time to Eat

Source file name:	Timetoeat.c, Timetoeat.cpp, Timetoeat.java, Timetoeat.py
Input:	`Standard`
Output:	`Standard`

The UCF Programming Team has made it to the World Contest Finals (WF), thanks to the great team members and coaches. Fortunately for Dr. Orooji (Team Faculty Advisor), WF is in a city with streets running only horizontally and vertically (this means the chance of Dr. O getting lost is less). The city can be described as a two-dimensional grid with R rows and C columns. The team hotel is at the upper-left corner (cell with indices 1, 1). The contest will be at the bottom-right corner (cell with indices R, C).

The UCF group will start at the hotel and needs to be at the contest site. From a cell, the group can walk into one of the four neighboring cells (north, south, east, west). Note that the cells on the boundary do not have four neighbors. The group would like to make it to the contest with the fewest number of steps - moving from a cell to a neighboring cell is considered taking a step.

One complication with the trip from the hotel to the contest site is that the UCF group gets hungry and needs to eat after they've taken certain number of steps. For example, if they have to eat after taking 10 steps, then their 10^{th} step (or an earlier step) must walk them into a cell with food (sub shop).

The need to eat means the group may not be able to take the shortest path from the hotel to the contest site because the sub shops may not be on that path. Fortunately, there are enough sub shops at different spots (cells) such that the group can eat when needed and make it to the contest site, though they may take a few extra steps than the straight path from the hotel to the contest site.

Given the description of the city, determine the minimum number of steps needed for the UCF group to go from their hotel to the contest site, taking into account that they need to eat after taking certain number of steps (or before taking that many steps if a sub shop is not exactly at that position on the path).

Input

The first input line contains four integers: R $(1 \leq R \leq 50)$, indicating the number of rows in the grid, C $(1 \leq C \leq 50)$, indicating the number of columns in the grid, F $(1 \leq F \leq 100)$, indicating the number of steps the group can take and then need to eat, and S $(1 \leq S \leq 100)$, indicating the number of sub shops in the grid.

Each of the next S input lines contains two integers $(1 \leq r \leq R,\ \ 1 \leq c \leq C)$ providing the row and column number for a sub shop. Assume that all sub shops are at different locations and they are not at the hotel or contest site.

Output

Print the minimum number of steps needed for the UCF group to go from their hotel to the contest site.

Example

Input	Output

10 6 5 7
1 6
5 4
8 1
9 1
8 3
10 1
2 5
	18

5 10 5 3
1 5
2 7
5 6
	13

Explanation

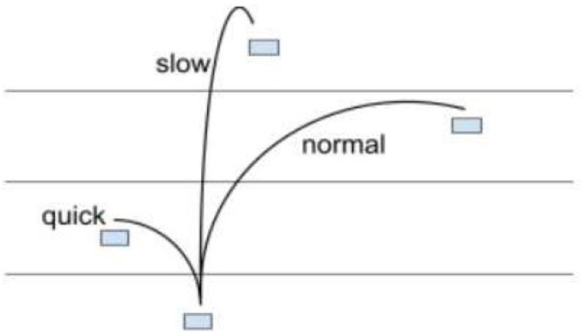
For the first example Input/Output:

The group must take the path that goes through the sub shops at locations (2, 5), (5, 4) and (8, 3), since they need to eat every 5^{th} step (or before 5^{th} step).

Problem F. Tower Climbing

Source file name:	Tower.c, Tower.cpp, Tower.java, Tower.py
Input:	`Standard`
Output:	`Standard`

You are speed running a game where you need to jump on various platforms and reach the top of the tower. The tower has multiple uniformly spaced layers. Each layer contains a single platform. The platform is so narrow that it can be thought of as a point. Depending on upgrades, your character can jump at most k layers up (in a single jump). The time it takes to jump from one point (platform) to another point is defined by the following equation:



$$Time((x_1,\ y_1),(x_2,\ y_2)) = |x_1 - x_2| + (y_1 - y_2)^2$$

where x_1 and x_2 represent (respectively) the x location of your starting and ending points (platforms) in a jump, and y_1 and y_2 represent (respectively) the layer for the starting and ending platforms for the jump.

Note that the layers are numbered sequentially starting at 1, i.e., the y coordinate of platforms going up are 1, 2, 3, etc. Note also that, as indicated by the $Time$ formula and shown in the picture, the time it takes to jump to a nearby layer is less than the time it takes to jump to a layer farther away.

You start at the first layer and finish at the last layer. You'd like to confirm that you have the optimal strategy, time wise.

Given the x location of the platforms and the maximum height you can jump, determine the least amount of time to reach the top of the tower.

Input

The first input line contains two integers: n $(2 \le n \le 2 \cdot 10^4)$, indicating the number of layers, and k $(1 \le k \le 8)$, indicating the maximum height you can jump. Each of the next n input lines contains an integer (between 1 and 10^9, inclusive), indicating the x coordinate of a platform. The platforms are provided in order from starting point to ending point (going up the tower), i.e., the y coordinate for platforms will be 1, 2, 3, etc. (so the y coordinates are provided in the input implicitly rather than explicitly).

Output

Print the total amount of time it takes to reach the top (last) platform from the bottom (first) platform.

Example

Input	Output

4 3
5
4
10
6
	8

2 1
99
100
	2

Problem G. Palindromic Primes

Source file name:	Palindromic.c, Palindromic.cpp, Palindromic.java, Palindromic.py
Input:	`Standard`
Output:	`Standard`

A prime number is an integer greater than 1 which is only divisible by the integers 1 and itself, and no other positive integers. A palindromic number is one that is the same when read forwards or backwards, when written with no leading zeros. A palindromic prime is a prime number that is also a palindrome. The first few palindromic primes are 2, 3, 5, 7, 11 and 101.

Given two integers, L and H, determine the number of palindromic primes that are between L and H, inclusive.

Input

The first and only input line will contain two space separated integers: L $(2 \leq L \leq 10^{12})$ and H $(L \leq H \leq 10^{12})$, the lower and upper bounds (inclusive) for the search.

Output

Print a single integer on a line by itself: the number of palindromic primes between L and H, inclusive.

Example

Input	Output
2 101	6
238 382	3
93139 97879	15

Problem H. Balanced Strings

Source file name:	Balance.c, Balance.cpp, Balance.java, Balance.py
Input:	`Standard`
Output:	`Standard`

Being an individual obsessed with balancing and trying to find strings that meet that criteria, you have begun a new endeavor to study the curious structure of what we will call balanced strings.

Balanced strings are strings that maintain an equal ratio of consonants to vowels in all of their substrings. What? You don't know the difference between a consonant and a vowel? Vowels are the characters 'a', 'e', 'i', 'o', 'u' and sometimes 'y'. Actually, you don't like the character 'y' because this makes the problem much harder. What was the difficulty level of this problem again? Oh yeah Medium! We can't have a problem about consonants and vowels that makes 'y' sometimes a vowel! That would make the problem a Hard and too many hard problems is a very bad thing. Being obsessed with balance, you have decided that 'y' will be included with the vowels. That way, there are 6 vowels and 20 consonants. A nice balanced even number of both! Oh! I almost forgot! A consonant is any letter that is not a vowel. Also to simplify things (this is a medium problem after all!), we will consider strings composed of lowercase letters only.

Now you are ready to understand balanced strings! A balanced string is a string that has an equal number of consonants and vowels in all of its substrings. Well... not all of its substrings. Just the substrings of even length! For example, the string "orooji" has the following set of even-length substrings: {"or", "ro", "oo", "oj", "ji", "oroo", "rooj", "ooji", "orooji"}. In that set the following substrings are unbalanced: {"oo", "oroo", "ooji", "orooji"}. That is, the substrings do not contain an equal number of vowels and consonants. So, the string "orooji" is not balanced. But a string like "arup" is balanced. The requisite even-length substrings are: {"ar", "ru", "up", "arup"} and all these substrings are balanced (have the same number of vowels and consonants), thus the string "arup" is balanced. Note that a balanced string may contain an odd number of characters, e.g., the string "ali" is balanced since all its even-length substrings ({"al", "li"}) are balanced.

Now you want to take words and erase some of the letters and replace them with new letters to form balanced strings. Since this is a medium problem, we've taken the liberty of erasing some of the letters for you and replaced them with '`?`' characters. See! The problem is already halfway solved!

Given a string of lowercase letters and '`?`' characters, count the number of ways to replace all the '`?`' with lowercase letters such that the string results in a balanced string. Two ways are considered different if there exists some i such that the i^{th} character in one string differs from the i^{th} character of the other string. Note that all the '`?`' do not have to be replaced by the same letter.

Input

The first input line contains a positive integer, n, indicating the number of strings to balance. The strings are on the following n input lines, one string per line. Each string contains only lowercase letters and/or '`?`' characters. Assume each input string has at least one character and at most 100 characters.

Output

For each string, output "`String #d: v`" where v is the number of ways to replace the questions marks with lower case letters. It is guaranteed that v will fit in a signed 64-bit integer for the strings we provide. Leave a blank line after the output for each string.

Example

Input	Output
7 orooji al? a?i g?ha a?u? ????????????????? arup	String #1: 0 String #2: 6 String #3: 20 String #4: 6 String #5: 400 String #6: 1117952409600000000 String #7: 1

Explanation

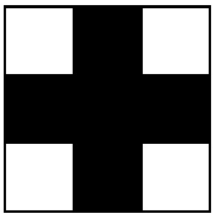
The first and last strings in Example Input do not have any ‘?’. The first input is not balanced so there is no way of replacing all ‘?’ to make it balanced, thus the output is zero. However, the last input is balanced so there is one way of replacing all ‘?’ to make it balanced, thus the output is 1.

Problem I. Reach for the Stars

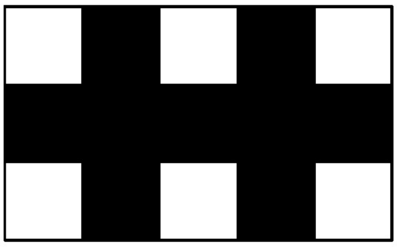
Source file name:	Stars.c, Stars.cpp, Stars.java, Stars.py
Input:	`Standard`
Output:	`Standard`

When you were a little kid, you would stamp papers randomly but your parents were still so impressed by your art. Now you are dealing with the UCF Programming Team Coaches and you need to be more creative to impress them. Good luck!

You are given a star shaped stamp like the one below. The black area is covered in ink and the white area is not. When the stamp hits the paper, it leaves a mark for each cell of ink that hits the paper.



For example, the image below can be made with two stampings. Notice the stamp must always remain axis-aligned when hitting the paper. We also require that the stamp be completely contained within the paper. Note a cell of paper stamped once with black ink is indistinguishable from a cell of paper stamped multiple times with black ink. Note also that cells and stamp line up properly, i.e., a cell is either covered completely by the stamp or not covered at all, i.e., the stamp will not cover part of a cell.



Given a black and white image, determine the minimum number of times, if possible, you would need to stamp the paper with the star stamp to end up with the design specified.

Input

The first input line contains a positive integer, n, indicating the number of images to evaluate. Each image starts with a line containing two integers, r and c, $(1 \leq r \leq 9, 1 \leq c \leq 9)$, representing the number of rows and columns, respectively. The next r input lines contains c characters each. The characters are either '`.`', representing a blank cell of the image and '`#`', representing a cell of the image covered in ink.

Output

For each image, output "`Image #d: v`" where v is the minimum number of stampings required to make the image. Replace v with "`impossible`" (without quotes) if it is not possible to form the image using the star shaped stamp. Leave a blank line after the output for each test case.

Example

Input	Output
5	Image #1: 0
1 1	
.	Image #2: impossible
1 1	
#	Image #3: 1
3 3	
.#.	Image #4: 2
###	
.#.	Image #5: 5
3 5	
.#.#.	
#####	
.#.#.	
4 7	
.##.#..	
######.	
.######	
..#..#.	

Problem J. You Shall Pass

Source file name:	Passing.c, Passing.cpp, Passing.java, Passing.py
Input:	`Standard`
Output:	`Standard`

Students failing COT 3100 (Introduction to Discrete Structures) has become a large concern for the professors at UCF. Matt and Sean have compiled a large amount of data, and after extensively analyzing it, they have generated some interesting probabilities. For each student, there are two basic probabilities known: the probability they will pass Matt's class and the probability they will pass Sean's class. Also known is a large table of probabilities that a student will pass based on being in the same class as another student. When a pair of students are in the same class, they may get together to form a study group, increasing their probability of passing the class. For each pair of students, i, j, a value a_{ij} is known. If students i and j are in the same class then student i is a_{ij} more likely to pass due to a study group. That is, i's probability of passing increases by a_{ij}. For example, if the initial probability of student i passing is 0.4 and $a_{ij} = 0.2$, then i's probability of passing with j in her class becomes 0.6 $(0.4 + 0.2 = 0.6)$.

Now Matt and Sean are trying to find a way to split the students into two classes such that the expected number of people passing is maximized.

Given the initial passing probabilities of the students and the passing probabilities from study groups, you are to determine the maximum expected number of passing students obtainable by splitting the students into two classes. Note that every student will be in exactly one class but one class may be empty.

Input

The first input line contains a positive integer, g, indicating the number of semesters to check. The data for each semester starts with an integer n $(2 \le n \le 50)$, which is the number of students who sign up for Discrete (COT 3100) in that semester. The following two input lines contain n decimal values each. The i-th value of the first line represents the probability of the i-th student passing Matt's class, and the i-th value of the second line represent the probability of the i-th student passing Sean's class. This is followed by n lines. The i-th line contains n non-negative decimal numbers. The j-th number on each line is a_{ij} as specified above.

For any given class configuration, assume that at no time a student will have a probability of passing higher than 1 or less than 0 (even after considering study groups). Each decimal number in input will be of the form "`#.##`", where "`#`" denotes a digit from 0 to 9, inclusive (i.e., each input value will be given to exactly two decimal places).

Output

For each semester, output a single decimal value representing the maximum expected number of students passing. Round the answers to two decimal places (e.g., 1.234 rounds to 1.23 and 1.235 rounds to 1.24).

Example

Input	Output
2	1.50
2	2.65
0.75 0.25	
0.25 0.75	
0.00 0.20	
0.20 0.00	
3	
0.20 0.60 0.95	
0.40 0.40 0.95	
0.00 0.00 0.55	
0.00 0.00 0.35	
0.00 0.00 0.00	