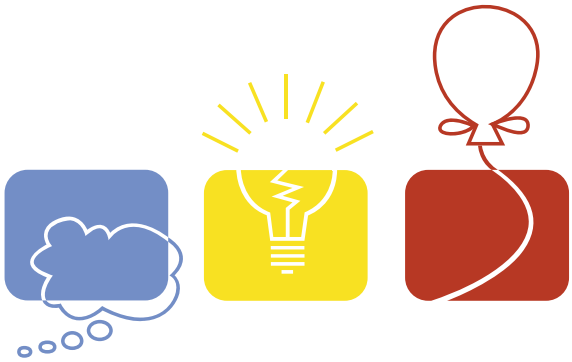
NWERC 2006

The 2006 ACM Northwestern European Programming Contest
KTH - Royal Institute of Technology, Stockholm, Sweden



acm International Collegiate Programming Contest

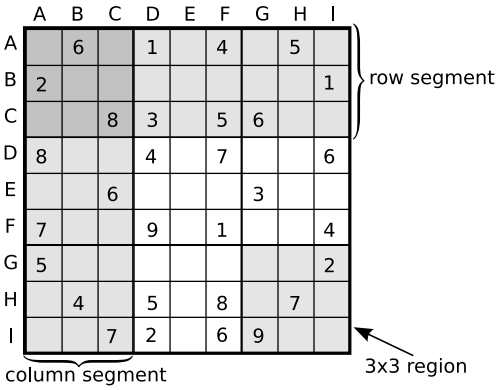
IBM event sponsor

The Problem Set

Almost blank page

Problem A: Sudoku

Oh no! Bill just realized that the sudoku puzzle he had spent the last ten minutes trying to solve essentially was last week's puzzle, only rotated counterclockwise. How cheap! Couldn't the magazine afford to make a new one every week? Of course, he had no way of knowing about this before he started to solve it, as the holes to fill with digits were other than last week. Nevertheless, realizing that this week's puzzle was a simple derivative of last week's certainly took the fun out of solving the rest of it.



The sudoku board consists of 9×9 cells. These can be grouped into 3×3 *regions* of 3×3 cells each. Some of the cells are filled with a digit 1 through 9 while the rest of them are left empty. The aim of the game is to fill each empty cell with a digit $1 \ldots 9$ so that every row, every column and every region contains each of the numbers $1 \ldots 9$ exactly once. A proper sudoku puzzle always has exactly one solution.

Help Bill avoid unpleasant surprises by creating a program that checks whether an unsolved sudoku puzzle is in fact derived from an earlier puzzle by simple operations.

The allowed operations are:

1. Rotating the entire puzzle clockwise or counterclockwise.
2. Swapping two columns within a 3×9 column segment.
3. Swapping two rows within a 9×3 row segment.
4. Swapping entire row or column segments.
5. Applying a permutation f of the digits $1 \ldots 9$ to every cell (i.e. replace x by $f(x)$ in every cell).

An operation is considered being performed on the sudoku solution (rather than on the unsolved puzzle) and always guarantees that if the board before the transformation was a solution to a sudoku puzzle, it still is afterwards.

Input

The input starts with the number of test cases $0 \leq N \leq 50$ on a single line.

Then for every test case follow nine lines describing last week's puzzle solution, from top to bottom. Each line corresponds to a row in the puzzle and consists of nine digits ($1 \ldots 9$), describing the contents of the cell from left to right.

Last week's solution is followed by nine lines describing this week's unsolved puzzle. Here, also, every line corresponds to a puzzle row and every digit ($0 \ldots 9$) describes the contents of a cell. 0 indicates that the cell is empty. The rows are presented ordered from top to bottom, and within each row, the cells are ordered from left to right.

After every test case except the last one follows a blank line. Every unsolved puzzle is guaranteed to be uniquely solvable and last week's solution is always a proper sudoku solution.

Output

For every test case, output `Yes` if the sudoku puzzle can be derived from the given solved puzzle using the allowed operations, or `No` if this is not possible.

Sample input	**Sample output**
2 963174258 178325649 254689731 821437596 496852317 735961824 589713462 317246985 642598173 060104050 200000001 008305600 800407006 006000300 700901004 500000002 040508070 007206900 534678912 672195348 198342567 859761423 426853791 713924856 961537284 287419635 345286179 010900605 025060070 870000902 702050043 000204000 490010508 107000056 040080210 208001090	Yes No

Problem B: The SetStack Computer

Background from Wikipedia: "Set theory is a branch of mathematics created principally by the German mathematician Georg Cantor at the end of the 19th century. Initially controversial, set theory has come to play the role of a foundational theory in modern mathematics, in the sense of a theory invoked to justify assumptions made in mathematics concerning the existence of mathematical objects (such as numbers or functions) and their properties. Formal versions of set theory also have a foundational role to play as specifying a theoretical ideal of mathematical rigor in proofs."

Given this importance of sets, being the basis of mathematics, a set of eccentric theorist set off to construct a supercomputer operating on sets instead of numbers. The initial SetStack Alpha is under construction, and they need you to simulate it in order to verify the operation of the prototype.

The computer operates on a single stack of sets, which is initially empty. After each operation, the cardinality of the topmost set on the stack is output. The cardinality of a set S is denoted $|S|$ and is the number of elements in S. The instruction set of the SetStack Alpha is `PUSH`, `DUP`, `UNION`, `INTERSECT`, and `ADD`.

- `PUSH` will push the empty set $\{\}$ on the stack.
- `DUP` will duplicate the topmost set (pop the stack, and then push that set on the stack twice).
- `UNION` will pop the stack twice and then push the union of the two sets on the stack.
- `INTERSECT` will pop the stack twice and then push the intersection of the two sets on the stack.
- `ADD` will pop the stack twice, add the first set to the second one, and then push the resulting set on the stack.

For illustration purposes, assume that the topmost element of the stack is

$$A = \{\{\}, \{\{\}\}\}$$

and that the next one is

$$B = \{\{\}, \{\{\{\}\}\}\}.$$

For these sets, we have $|A| = 2$ and $|B| = 2$. Then:

- `UNION` would result in the set $\{\ \{\}, \{\{\}\}, \{\{\{\}\}\}\ \}$. The output is 3.
- `INTERSECT` would result in the set $\{\ \{\}\ \}$. The output is 1.
- `ADD` would result in the set $\{\ \{\}, \{\{\{\}\}\}, \{\{\},\{\{\}\}\}\ \}$. The output is 3.

Input

An integer $0 \leq T \leq 5$ on the first line gives the cardinality of the set of test cases. The first line of each test case contains the number of operations $0 \leq N \leq 2\,000$. Then follow N lines each containing one of the five commands. It is guaranteed that the SetStack computer can execute all the commands in the sequence without ever popping an empty stack.

Output

For each operation specified in the input, there will be one line of output consisting of a single integer. This integer is the cardinality of the topmost element of the stack after the corresponding command has executed. After each test case there will be a line with `***` (three asterisks).

Sample input	**Sample output**
2	0
9	0
PUSH	1
DUP	0
ADD	1
PUSH	1
ADD	2
DUP	2
ADD	2
DUP	***
UNION	0
5	0
PUSH	1
PUSH	0
ADD	0
PUSH	***
INTERSECT	

Problem C: Pie

My birthday is coming up and traditionally I'm serving pie. Not just one pie, no, I have a number N of them, of various tastes and of various sizes. F of my friends are coming to my party and each of them gets a piece of pie. This should be one piece of one pie, not several small pieces since that looks messy. This piece can be one whole pie though.

My friends are very annoying and if one of them gets a bigger piece than the others, they start complaining. Therefore all of them should get equally sized (but not necessarily equally shaped) pieces, even if this leads to some pie getting spoiled (which is better than spoiling the party). Of course, I want a piece of pie for myself too, and that piece should also be of the same size.

What is the largest possible piece size all of us can get? All the pies are cylindrical in shape and they all have the same height 1, but the radii of the pies can be different.

Input

One line with a positive integer: the number of test cases. Then for each test case:

- One line with two integers N and F with $1 \leq N, F \leq 10\,000$: the number of pies and the number of friends.
- One line with N integers r_i with $1 \leq r_i \leq 10\,000$: the radii of the pies.

Output

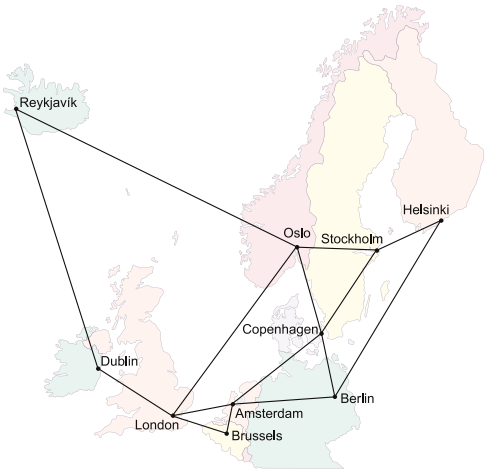
For each test case, output one line with the largest possible volume V such that me and my friends can all get a pie piece of size V. The answer should be given as a floating point number with an absolute error of at most 10^{-3}.

Sample input	**Sample output**
3 3 3 4 3 3 1 24 5 10 5 1 4 2 3 4 5 6 5 4 2	25.1327 3.1416 50.2655

Almost blank page

Problem D: Ticket to Ride

Ticket to Ride is a board game for up to 5 players. The goal of the game is to set up train lines (and to thwart the opponents' attempts at setting up their train lines). At the beginning of play, each player is assigned four train lines. A player may choose to discard as many of these four assignments as she likes. Each assignment has a score, corresponding to its difficulty (so, typically, a train line between e.g. Stockholm and Tokyo would be worth more than a train line between e.g. Stockholm and Utrecht). At the end of the game, each player gets points for the assignments that they have successfully completed, and penalty points for the assignments that they have failed to complete.



An assignment consists of a pair of cities that are to be connected by a series of shorter railway routes. A route can be claimed (for a certain cost associated with the route), but things are complicated by the fact that there is only a limited number of routes, and once a player claims a route, none of the other players can claim it. A player has successfully set up a train line between two cities if there is a path between the two cities using only routes that have been claimed by this player. For simplicity, we will ignore all additional aspects of the game (including the actual process of claiming routes and additional ways to score points).

For instance, if your assignment is to connect Stockholm and Amsterdam in the Figure above, you would probably want to claim the routes between Stockholm and Copenhagen, and between Copenhagen and Amsterdam. But if another player manages to claim the route between Copenhagen and Stockholm before you, your train line would have to use some other routes, e.g. by going to Copenhagen via Oslo.

In this problem, we will consider the rather bold strategy of trying to complete all four assignments (typically, this will be quite hard). As a preliminary assessment of the difficulty of achieving this, we would like to calculate the minimum cost of setting up all four lines assuming that none of the other players interfere with our plans. Your job is to write a program to determine this minimum cost.

Input

The input consists of several (at most 20) games to be analyzed. Each game starts with two integers $1 \leq n \leq 30$, $0 \leq m \leq 1\,000$, giving the number of cities and railway routes in the map, respectively. Then follow n lines, giving the names of the n cities. City names are at most 20 characters long and consist solely of lower case letters (`'a'-'z'`).

After this follow m lines, each containing the names of two different cities and an integer $1 \leq c \leq 10\,000$, indicating that there is a railway route with cost c between the two cities. Note that there may be several railway routes between the same pair of cities. You may assume that it is always possible to set up a train line from any city to any other city.

Finally, there will be four lines, each containing the names of two cities, giving the four train line assignments.

The input is terminated by a case where $n = m = 0$. This case should not be processed.

Output

For each game, output a single line containing a single integer, the minimum possible cost to set up all four train lines.

Sample input

```
10 15
stockholm
amsterdam
london
berlin
copenhagen
oslo
helsinki
dublin
reykjavik
brussels
oslo stockholm 415
stockholm helsinki 396
oslo london 1153
oslo copenhagen 485
stockholm copenhagen 522
copenhagen berlin 354
copenhagen amsterdam 622
helsinki berlin 1107
london amsterdam 356
berlin amsterdam 575
london dublin 463
reykjavik dublin 1498
reykjavik oslo 1748
london brussels 318
brussels amsterdam 173
stockholm amsterdam
oslo london
reykjavik dublin
brussels helsinki
2 1
first
second
first second 10
first first
first first
second first
first first
0 0
```

Sample output

```
3907
10
```

Problem E: The Bookcase

No wonder the old bookcase caved under the massive piles of books Tom had stacked on it. He had better build a new one, this time large enough to hold all of his books. Tom finds it practical to have the books close at hand when he works at his desk. Therefore, he is imagining a compact solution with the bookcase standing on the back of the desk. Obviously, this would put some restrictions on the size of the bookcase, it should preferably be as small as possible. In addition, Tom would like the bookcase to have exactly three shelves for aesthetical reasons.

Wondering how small his bookcase could be, he models the problem as follows. He measures the height h_i and thickness t_i of each book i and he seeks a partition of the books in three non-empty sets S_1, S_2, S_3 such that $\left(\sum_{j=1}^{3} \max_{i \in S_j} h_i\right) \times \left(\max_{j=1}^{3} \sum_{i \in S_j} t_i\right)$ is minimized, i.e. the area of the bookcase as seen when standing in front of it (the depth needed is obviously the largest width of all his books, regardless of the partition). Note that this formula does not give the exact area of the bookcase, since the actual shelves cause a small additional height, and the sides cause a small additional width. For simplicity, we will ignore this small discrepancy.

Thinking a moment on the problem, Tom realizes he will need a computer program to do the job.

Input

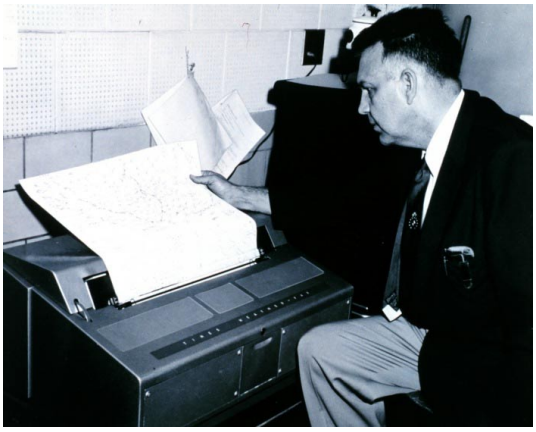
The input begins with a positive number on a line of its own telling the number of test cases (at most 20). For each test case there is one line containing a single positive integer N, $3 \leq N \leq 70$ giving the number of books. Then N lines follow each containing two positive integers h_i, t_i, satisfying $150 \leq h_i \leq 300$ and $5 \leq t_i \leq 30$, the height and thickness of book i respectively, in millimeters.

Output

For each test case, output one line containing the minimum area (height times width) of a three-shelf bookcase capable of holding all the books, expressed in square millimeters.

Sample input	**Sample output**
2 4 220 29 195 20 200 9 180 30 6 256 20 255 30 254 15 253 20 252 15 251 9	18000 29796

Problem F: Printer Queue



The only printer in the computer science students' union is experiencing an extremely heavy workload. Sometimes there are a hundred jobs in the printer queue and you may have to wait for hours to get a single page of output.

Because some jobs are more important than others, the Hacker General has invented and implemented a simple priority system for the print job queue. Now, each job is assigned a priority between 1 and 9 (with 9 being the highest priority, and 1 being the lowest), and the printer operates as follows.

- The first job J in queue is taken from the queue.
- If there is some job in the queue with a higher priority than job J, then move J to the end of the queue without printing it.
- Otherwise, print job J (and do not put it back in the queue).

In this way, all those important muffin recipes that the Hacker General is printing get printed very quickly. Of course, those annoying term papers that others are printing may have to wait for quite some time to get printed, but that's life.

Your problem with the new policy is that it has become quite tricky to determine when your print job will actually be completed. You decide to write a program to figure this out. The program will be given the current queue (as a list of priorities) as well as the position of your job in the queue, and must then calculate how long it will take until your job is printed, assuming that no additional jobs will be added to the queue. To simplify matters, we assume that printing a job always takes exactly one minute, and that adding and removing jobs from the queue is instantaneous.

Input

One line with a positive integer: the number of test cases (at most 100). Then for each test case:

- One line with two integers n and m, where n is the number of jobs in the queue ($1 \leq n \leq 100$) and m is the position of your job ($0 \leq m \leq n-1$). The first position in the queue is number 0, the second is number 1, and so on.
- One line with n integers in the range 1 to 9, giving the priorities of the jobs in the queue. The first integer gives the priority of the first job, the second integer the priority of the second job, and so on.

Output

For each test case, print one line with a single integer; the number of minutes until your job is completely printed, assuming that no additional print jobs will arrive.

Sample input	**Sample output**
3 1 0 5 4 2 1 2 3 4 6 0 1 1 9 1 1 1	1 2 5

Problem G: Prime Path

The ministers of the cabinet were quite upset by the message from the Chief of Security stating that they would all have to change the four-digit room numbers on their offices.
— It is a matter of security to change such things every now and then, to keep the enemy in the dark.
— But look, I have chosen my number 1033 for good reasons. I am the Prime minister, you know!
— I know, so therefore your new number 8179 is also a prime. You will just have to paste four new digits over the four old ones on your office door.
— No, it's not that simple. Suppose that I change the first digit to an 8, then the number will read 8033 which is not a prime!
— I see, being the prime minister you cannot stand having a non-prime number on your door even for a few seconds.
— Correct! So I must invent a scheme for going from 1033 to 8179 by a path of prime numbers where only one digit is changed from one prime to the next prime.

Now, the minister of finance, who had been eavesdropping, intervened.
— No unnecessary expenditure, please! I happen to know that the price of a digit is one pound.
— Hmm, in that case I need a computer program to minimize the cost. You don't know some very cheap software gurus, do you?
— In fact, I do. You see, there is this programming contest going on...
Help the prime minister to find the cheapest prime path between any two given four-digit primes! The first digit must be nonzero, of course. Here is a solution in the case above.

```
1033
1733
3733
3739
3779
8779
8179
```

The cost of this solution is 6 pounds. Note that the digit 1 which got pasted over in step 2 can not be reused in the last step – a new 1 must be purchased.

Input

One line with a positive number: the number of test cases (at most 100). Then for each test case, one line with two numbers separated by a blank. Both numbers are four-digit primes (without leading zeros).

Output

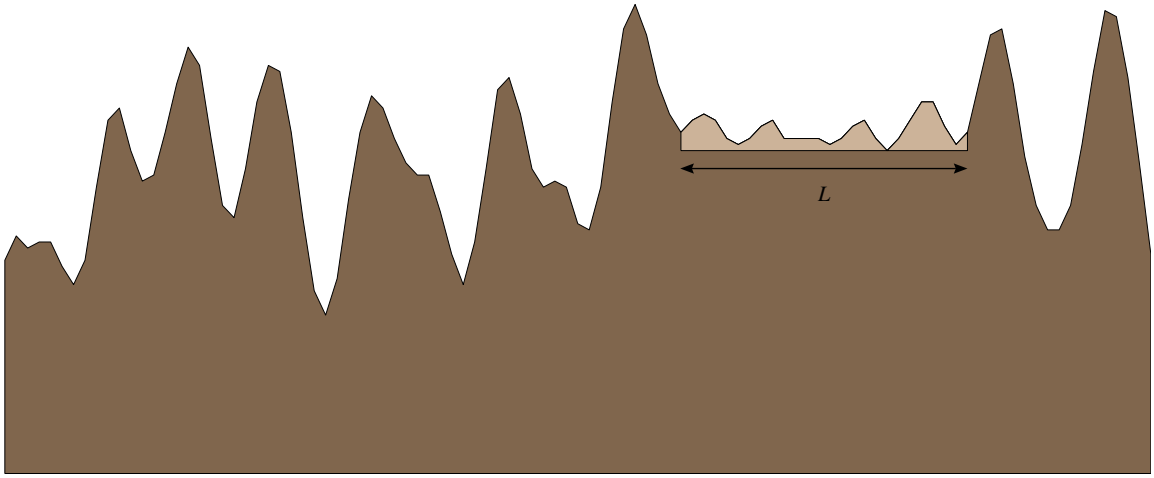
One line for each case, either with a number stating the minimal cost or containing the word `Impossible`.

Sample input	**Sample output**
3 1033 8179 1373 8017 1033 1033	6 7 0

Problem H: Lineland's Airport

Lineland is a strange country. As the name suggests, it's shape (as seen from above) is just a straight line, rather than some two-dimensional shape. The landscape along this line is very mountainous, something which occasionally leads to some problems. One such problem now occurs: in this modern era the king wants to build an airport to stimulate the country's economy. Unfortunately, it's impossible for airplanes to land on steep airstrips, so a horizontal piece of land is needed. To accommodate for the larger airplanes, this strip needs to have length at least L.

Over the years, the inhabitants of Lineland have become very proficient in flattening pieces of land. Given a piece a land, they can remove rock quickly. They don't want to add rock for that may lead to an unstable landing strip. To minimize the amount of effort, however, they want to remove the least amount of rock necessary to reach their goal: a flat piece of land of length L. What is this minimum amount? Because of the low-dimensional nature of Lineland, the amount of rock that needs to be removed is measured as the total area of land above the place where the landing strip is placed, rather than the volume (so in the Figure below, the amount of land removed is given by the lightly shaded area).



Input

One line with a positive number: the number of test cases (at most 25). Then for each test case:

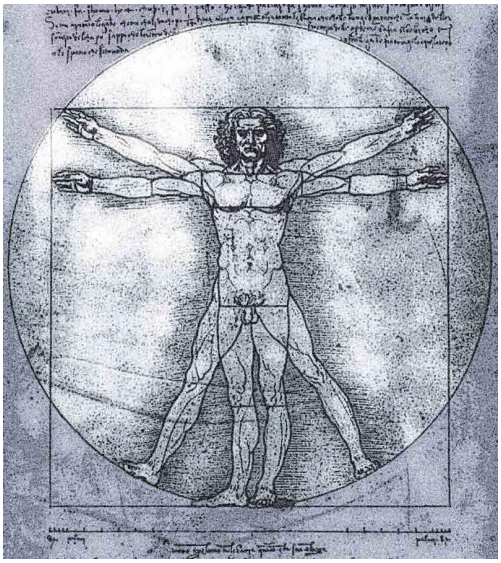
- One line with an integer N, $2 \leq N \leq 500$, the number of points, and an integer L, $1 \leq L \leq 10\,000$, the necessary length to flatten.
- N lines with two integers x_i and y_i with $0 \leq x_i, y_i \leq 10\,000$ describing the landscape of Lineland. The x_i are in (strictly) ascending order. At position x_i the height of the landscape is y_i. Between two x_i the landscape has constant slope. (So the landscape is piecewise linear). The difference between x_N and x_1 is greater than or equal to L.

Output

For each test case, output one line with the minimum amount of rock which must be removed in order to build the airport. The answer should be given as a floating point number with an absolute error of at most 10^{-3}.

Sample input	**Sample output**
4 3 5 0 2 4 2 14 0 4 3 0 2 2 0 4 0 5 3 3 10 10 2 30 2 35 7 2 777 222 333 4444 5555	0.9000 0.3750 0.0000 373362.4867

Problem I: Leonardo's Notebook



— I just bought Leonardo's secret notebook!
Rare object collector Stan Ucker was really agitated but his friend, special investigator Sarah Keptic was unimpressed.
— How do you know it is genuine?
— Oh, it must be, at that price. And it is written in the da Vinci code.
Sarah browsed a few of the pages. It was obvious to her that the code was a substitution cipher, where each letter of the alphabet had been substituted by another letter.
— Leonardo would have written the plain-text and left it to his assistant to encrypt, she said. And he must have supplied the substitution alphabet to be used. If we are lucky, we can find it on the back cover!
She turned up the last page and, lo and behold, there was a single line of all 26 letters of the alphabet:

QWERTYUIOPASDFGHJKLZXCVBNM

— This may be Leonardo's instructions meaning that each A in the plain-text was to be replaced by Q, each B with W, etcetera. Let us see...
To their disappointment, they soon saw that this could not be the substitution that was used in the book. Suddenly, Stan brightened.
— Maybe Leonardo really wrote the substitution alphabet on the last page, and by mistake his assistant coded that line as he had coded the rest of the book. So the line we have here is the result of applying some permutation TWICE to the ordinary alphabet!
Sarah took out her laptop computer and coded fiercely for a few minutes. Then she turned to Stan with a sympathetic expression.
— No, that couldn't be it. I am afraid that you have been duped again, my friend. In all probability, the book is a fake.
Write a program that takes a permutation of the English alphabet as input and decides if it may be the result of performing some permutation twice.

Input

The input begins with a positive number on a line of its own telling the number of test cases (at most 500). Then for each test case there is one line containing a permutation of the 26 capital letters of the English alphabet.

Output

For each test case, output one line containing `Yes` if the given permutation can result from applying some permutation twice on the original alphabet string ABC...XYZ, otherwise output `No`.

Sample input	**Sample output**
2 QWERTYUIOPASDFGHJKLZXCVBNM ABCDEFGHIJKLMNOPQRSTUVWXYZ	No Yes