ACM International Collegiate Programming Contest
2006 East Central Regional Contest
Ashland University
Sheridan Institute of Technology
University of Cincinnati
University of Michigan – Dearborn
November 11, 2006

Sponsored by IBM

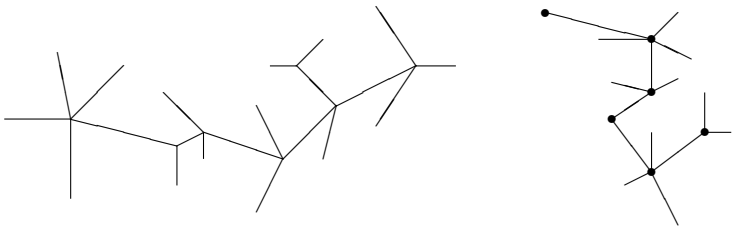
Rules:

1. There are **eight** questions to be completed in **five hours**.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. The allowed programming languages are C, C++ and Java.
4. All programs will be re-compiled prior to testing with the judges' data.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contestant officials (e.g., that might generate a security violation).
6. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
7. All communication with the judges will be handled by the PC^2 environment.
8. Judges' decisions are to be considered final. No cheating will be tolerated.

Problem A: Caterpillar

An undirected graph is called a *caterpillar* if it is connected, has no cycles, and there is a path in the graph where every node is either on this path or a neighbor of a node on the path. This path is called the *spine* of the caterpillar and the spine may not be unique. You are simply going to check graphs to see if they are caterpillars.

For example, the left graph below is not a caterpillar, but the right graph is. One possible spine is shown by dots.



Input

There will be multiple test cases. Each test case starts with a line containing n indicating the number of nodes, numbered 1 through n (a value of $n = 0$ indicates end-of-input). The next line will contain an integer e indicating the number of edges. Starting on the following line will be e pairs n_1 n_2 indicating an undirected edge between nodes n_1 and n_2. This information may span multiple lines. You may assume that $n \leq 100$ and $e \leq 300$. Do not assume that the graphs in the test cases are connected or acyclic.

Output

For each test case generate one line of output. This line should either be

Graph g is a caterpillar.

or

Graph g is not a caterpillar.

as appropriate, where g is the number of the graph, starting at 1.

Sample Input

```
22
21
1 2 2 3 2 4 2 5 2 6 6 7 6 10 10 8 9 10 10 12 11 12 12 13 12 17
18 17 15 17 15 14 16 15 17 20 20 21 20 22 20 19
16
15
1 2 2 3 5 2 4 2 2 6 6 7 6 8 6 9 9 10 10 12 10 11 10 14 10 13 13 16 13 15
0
```

Sample Output

```
Graph 1 is not a caterpillar.
Graph 2 is a caterpillar.
```

Problem B: Hie with the Pie

The Pizazz Pizzeria prides itself in delivering pizzas to its customers as fast as possible. Unfortunately, due to cutbacks, they can afford to hire only one driver to do the deliveries. He will wait for 1 or more (up to 10) orders to be processed before he starts any deliveries. Needless to say, he would like to take the shortest route in delivering these goodies and returning to the pizzeria, even if it means passing the same location(s) or the pizzeria more than once on the way. He has commissioned you to write a program to help him.

Input

Input will consist of multiple test cases. The first line will contain a single integer n indicating the number of orders to deliver, where $1 \leq n \leq 10$. After this will be $n + 1$ lines each containing $n + 1$ integers indicating the times to travel between the pizzeria (numbered 0) and the n locations (numbers 1 to n). The j^{th} value on the i^{th} line indicates the time to go directly from location i to location j without visiting any other locations along the way. Note that there may be quicker ways to go from i to j via other locations, due to different speed limits, traffic lights, etc. Also, the time values may not be symmetric, i.e., the time to go directly from location i to j may not be the same as the time to go directly from location j to i. An input value of $n = 0$ will terminate input.

Output

For each test case, you should output a single number indicating the minimum time to deliver all of the pizzas and return to the pizzeria.

Sample Input

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

Sample Output

```
8
```

Problem C: Mahershalalhashbaz, Nebuchadnezzar, and Billy Bob Benjamin Go to the Regionals

The Association for Computing Machinery (ACM) is considering new rules for its regional programming contests, in part to solve some software problems. For instance, the program that prints out the badges for each team was designed so that the same font size is used to print all badges for that team. However, this means that if one member of the team has a very long name, then a very small font will be used to print all of the team's badges, and this wouldn't look very nice for someone with an extremely short name like "AL".

The initial solution proposed by the ACM was to put a limit on the length of contestants' names. Someone pointed out that this would discriminate against teams from certain regions where bigger names are more common (for instance, children in the home towns of well-known actors are more likely to be named after that actor – imagine how many children in Oakland, California have been named after the actor Mahershalalhashbaz Ali since he became one of the stars of the television series *The 4400*).

As a compromise, the ACM decided to change the rule to require that "no team member's name can have length more than two away from the average length of all the team member's names." Using this rule, a team consisting of "MAHERSHALALHASHBAZ", "AL", and "BILL" would be disqualified (average name length is 8, so AL and BILL are not within 2). However, "MAHERSHALALHASHBAZ", "NEBUCHADNEZZAR", and "BILLYBOBBENJAMIN" would be okay (average name length is 16, and all team member names have length within 2 of this number).

Given the names of n students, determine whether or not they can be placed into teams of k members each so that each team meets the requirements of the new ACM rule.

Input

Input will consist of multiple test cases. Each test case begins with a line consisting of two positive integers n and k, where $n \leq 1000, k \leq 8$, and n is divisible by k. Following this are n lines, each containing a single name consisting only of upper case letters with no embedded, leading, or trailing blanks. These are the names of the n students who need to be organized into teams of size k each. No name will exceed 80 characters. The last test case is followed by a line containing two zeros.

Output

For each test case, output the case number (starting at 1) followed by either the word "yes" (meaning that it is possible to organize the students into teams of size k so that no student on that team has a name whose length is greater than distance 2 from the average name lengths of the members on that team), or "no" if it is not possible. Use the format shown in the sample output. Insert a blank line between cases.

Sample Input

```
3 3
MAHERSHALALHASHBAZ
AL
BILL
6 3
MAHERSHALALHASHBAZ
AL
NEBUCHADNEZZAR
BILL
BILLYBOBBENJAMIN
JILL
0 0
```

Sample Output

```
Case 1: no

Case 2: yes
```

Problem D: The Mastermind Master's Mind

The Advancement of Collegiate Mastermind (hey, that's ACM again...weird!) is an organization which (among other things) holds classes for college students to improve their Mastermind skills. Recall that basic Mastermind is a two-player game which works as follows: The first player – the codemaker – creates a secret 4-color code, each color taken from one of six colors (we'll use A,B,C,D,E and F for the colors). The other player – the codebreaker – now must try to guess the codemaker's code. After each guess, the codemaker uses black and white pegs to tell the codebreaker two things: the number of correct colors in the correct positions (the black pegs), and the number of remaining correct colors that are in the wrong positions (the white pegs). For example, if the true code is ABCC, and the codebreaker makes the guess ACCD, then the response would be 2 black and 1 white; if the guess was CCAA, the response would be 3 white. The codebreaker continues making guesses until he receives 4 blacks. More advanced versions of Mastermind increase both the length of the code and the number of colors to choose from.

The ACM's master instructor is Dee Sifer, and she has a unique ability: when given a set of n guesses and responses, she can immediately determine what the best $(n+1)^{\text{st}}$ guess should be. For each possible $(n+1)^{\text{st}}$ guess, Dee calculates (in her head) the number of codes left for each possible response she could get to that guess. The maximum of these numbers over all responses is called the *uncertainty* of the guess. The "best" guess is the one with the minimum uncertainty. For example, suppose that you get to a point in a game where you've narrowed down the answer to only three possible codes: ABBB, ABBC or ABCB. If your next guess is ABBB, there would be two possible responses: 4 blacks (leaving 0 remaining possibilities left) or 3 blacks (leaving 2 remaining possibilities – ABBC and ABCB). However, if instead of ABBB you try ABBC, then there are 3 possible responses: 4 blacks (leaving 0 possibilities), 3 blacks (leaving 1 possibility – ABBB) and 2 blacks and 2 whites (also leaving 1 possibility – ABCB). Thus ABBC would be a better guess in this case, since the uncertainty it leaves is 1, whereas the uncertainty for ABBB is 2.

This is all well and good, except for one thing. You have been selected as Dee's successor, and you do NOT have Dee's ability to pick the minimum uncertainty guess. Dee has been dropping hints (in code) that she plans to retire soon, so you need a program to help you simulate her ability.

Input

Input will consist of multiple test cases. The first line of the input file will contain a single integer indicating the number of test cases. Each test case will consist of several lines. The first line will contain three integers: l c n, where l is the length of the code being guessed, c is the number of colors to choose from, and n is the number of guesses made so far. These values will satisfy $1 \leq l \leq 15, 1 \leq c \leq 20, 0 \leq n \leq 10$. The values of l and c will be such that the total possible number of codes will be ≤ 32768. After this will come n lines of the form

`guess` b w

where `guess` is a length-l character string specifying a guess, and b and w are the number of black and white pegs in the response. All colors will be uppercase letters taken from the first c letters of the alphabet. For each test case, the guesses given will limit the total number of possible solutions to ≤ 1500.

Output

For each test case, output a single line containing the best guess and its uncertainty. Use a single blank to separate the guess from the uncertainty. If there is more than one guess with the same minimum uncertainty, use the one which comes first lexicographically.

Sample Input

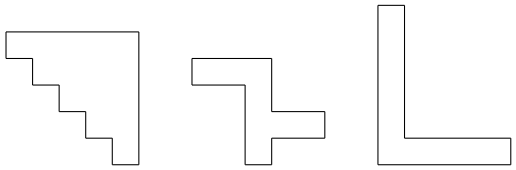
```
3
4 6 2
AABC 1 2
BEAC 0 3
4 6 1
ABCD 0 0
3 20 4
ABE 1 0
ROM 1 0
INK 1 0
MOB 0 2
```

Sample Output

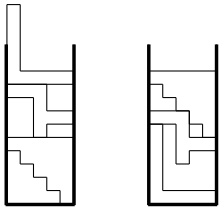
```
ABCD 4
AEEE 3
IBM 0
```

Problem E: Plaque Pack

The *Knick Knack Plaque Shack* designs plaques of unusual shapes. All the plaques are 1 inch deep, and have a wide variety of shapes, some of which are shown below.



Ben Fitt is one of several workers in the shipping department (part of the Knick Knack Plaque Shack Pack, as they like to call themselves). Each day he is assigned the task of shipping all the plaques of a certain width to the various department stores which sell them. He has at his disposal boxes with a depth of 1 and a width equal to the plaques' width. As the plaques come off the assembly line, he fits them into the boxes one at a time. When placed in a box, each plaque will slide down until some part of it touches the topmost plaque already in the box (or the bottom of the box if it is the first plaque). For example, if the leftmost plaque above came off the assembly line first, followed by the middle and then the rightmost, they would stack up one on top of the other as shown on the left. If they came off the assembly line in reverse order, they would stack up as shown on the right.



When a plaque comes off the assembly line which will not fit into the box (i.e., it sticks up over the top), Ben closes that box, ships it off, and starts a new box. In the above examples, the height of the boxes is only 12, so it would take two boxes for the first ordering of plaques, but only one for the second.

During his free moments between packing plaques, Ben wonders what it would be like if hundreds of computer programmers tried to write code to simulate this monotonous drudgery.

Input

Input will consist of multiple test cases. Each test case will start with a line containing three integers n w b, where n indicates the number of plaques to ship, w indicates the width of each plaque, and b indicates the height of each shipping box. These values will lie in the ranges $1 \ldots 100$, $1 \ldots 10$ and $1 \ldots 100$, respectively. Following this line will be n specifications of plaque shapes. Each shape specification starts with a single line containing the integer height h of the plaque ($1 \leq h \leq 10$ and $h \leq b$). Following this will be h lines containing w characters each, where each character is either `'X'` (indicating a part of the plaque) or `'.'`, indicating empty space. The order in which the plaques appear in the input is the order in which they must be packed in the boxes, and rotating or inverting the plaques is not allowed. The input file will end with the line 0 0 0.

Output

For each test case, output a single line containing the maximum height of the plaques in each box, in the order in which they are filled.

Sample Input

```
3 5 12
5
XXXXX
.XXXX
..XXX
...XX
....X
4
XXX..
..X..
..XXX
..X..
6
X....
X....
X....
X....
X....
XXXXX
3 5 12
6
X....
X....
X....
X....
X....
XXXXX
4
XXX..
..X..
..XXX
..X..
5
XXXXX
.XXXX
..XXX
...XX
....X
0 0 0
```

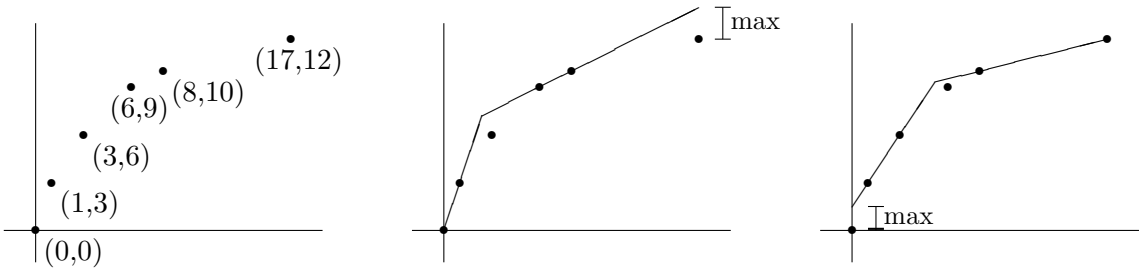
Sample Output

```
9 6
10
```

Problem F: Roofing It

Bill Eaves owns the *Shingle Minded* roofing company which is a sub-contracting firm specializing in putting roofs on buildings. Often, Bill will receive a design for a roof from some young hot-shot architect which, though it may have some aesthetic appeal, is totally impractical. In these cases, Bill will begin negotiations with the architect and the client to find a simpler design. Bill's only concern is that the roof be convex, to allow rain, snow and frisbees to roll off easily. The architect's main concern is that the maximum height between his original roof and the compromise roof be as small as possible. The client's main concern is to get out of this meeting as soon as possible and get back to watching TV.

The architect's plans for the roof come in the form of a series of n points (x_i, y_i) specifying the outline of the roof as seen from the side of the house. The roofs are always symmetrical, so the architect only shows the front side of the roof (from its start at the front of the house to its peak). Once Bill gets these plans and a decision is made on how many sections the convex roof should have, he must decide how to place the sections so as to 1) make sure that all the original (x_i, y_i) points lie on or below the new roof, and 2) to minimize the maximum vertical distance between any of the original (x_i, y_i) points and the new roof. All sections must lie on at least two of the initial points specified by the architect. An example is shown below. On the left are the initial points from the architect. The next two pictures show an approximation of the roof using only two sections. While both of these are convex, the second of the two is the one which minimizes the maximum distance.



Input

Input will consist of multiple test cases. Each case will begin with two positive integers n and k ($2 \le n \le 100, 1 \le k < n$) indicating the number of points used in the original roof plan and the number of sections used in the compromise roof. These will be followed by n lines each containing two floating points numbers x_i y_i, specifying the n points in the roof plan. These values will be given in increasing order of x, and the last point will be guaranteed to have the highest y value of any of the points. All values will be between 0.0 and 10000.0. The last case is followed by a line containing 0 0 which indicates end-of-input and should not be processed.

Output

For each test case, output the maximum distance between the best compromise roof and the initial points, rounded to the nearest thousandth.

Sample Input

```
6 2
0.0 0.0
1.0 3.0
3.0 6.0
6.0 9.0
8.0 10.0
17.0 12.0
0 0
```

Sample Output

```
1.500
```

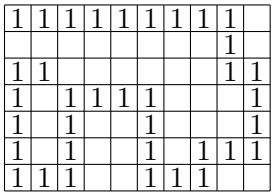
Problem G: Snakes on a Plane

Assume we have an $n \times m$ grid of squares, each filled with either 0 or 1. A *snake* is a connected sequence of grid squares which has the following properties:

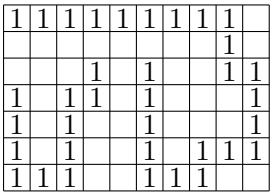
1. Each snake square has a 1 in it
2. Each snake square touches exactly two other snake squares (north/south/east/west), except the first and last square in the sequence (the head and tail of the snake)

A *maximal snake* is one in which we cannot add a 1 to either end without either lengthening the snake, combining two snakes together, or violating rule 2 above.

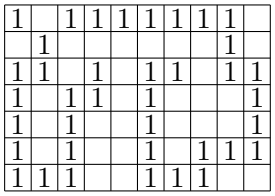
The examples below show grids with and without maximal snakes (all empty squares have 0's in them). Notice that the second grid does not have a maximal snake since you can add a 1 at the end of either snake to get a larger snake.



One maximal snake



No maximal snakes



Three maximal snakes

For this problem, you will be given grids and must count the number of maximal snakes in each.

Input

Input will consist of multiple test cases. The first line of each test case will contain two positive integers n m indicating the number of rows and columns in the grid (the maximum value of each will be 200). The next n lines will consist of m characters (either '0' or '1') specifying the grid. The last case is followed by a line containing 0 0 which indicates end-of-input and should not be processed.

Output

For each test case, output a single line containing the number of maximal snakes in the grid.

Sample Input

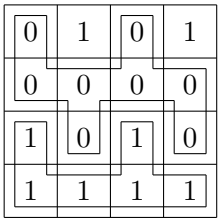
```
7 10
1111111110
0000000010
1100000011
1011110001
1010010001
1010010111
1110011100
7 10
1111111110
0000000010
0001010011
1011010001
1010010001
1010010111
1110011100
7 10
1011111110
0100000010
1101011011
1011010001
1010010001
1010010111
1110011100
0 0
```

Sample Output

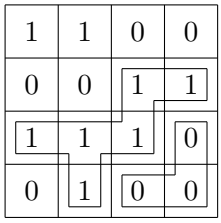
```
1
0
3
```

Problem H: Stake Your Claim

The designers at Gazillion Games Inc. have come up with a new, relatively simple game called "Stake Your Claim". Two players – 0 and 1 – initially select two values n and m and an $n \times n$ board is created with m 0's and m 1's randomly placed on the board. Starting with player 0, each player puts his/her number in one of the empty squares on the board. After the board is filled, each player's score is equal to the largest connected region on the board filled with that player's number (where a connected region is one where for any two squares in the region a path exists consisting of only N/S/E/W moves). The player with the highest score wins, and is awarded the difference between his/her score and the score of the other player. Two examples of finished games are shown below, with the largest connected regions for each player outlined. Note in the second example that the two sections with 2 0's each are not connected.



Player 0's score: 8
Player 1's score: 6
Player 0 awarded 2 points



Player 0's score: 3
Player 1's score: 6
Player 1 awarded 3 points

In order to test how good this game is, the gang at Gazillion has hired you to write a program which can play the game. Specifically, given any starting configuration, they would like a program to determine the best move for the current player, i.e., the score which maximizes the points awarded to that player (or minimizes those awarded to the player's opponent).

Input

Input will consist of multiple test cases. Each test case will start with a line containing a positive integer n (≤ 8) indicating the size of the board. Next will come n lines describing the current board layout (row 0 first, followed by row 1, etc). Each of these lines will contain n characters taken from `'0'`, `'1'` and `'.'`, where `'.'` represents an empty square. The first character will be in column 0, the second in column 1, etc. The number of 0's on the board will either be equal to the number of 1's or one greater, and there will be between 1 and 10 (inclusive) empty squares. The last case is followed by a line containing 0 which indicates end-of-input and should not be processed.

Output

For each test case, output a single line containing two items: the coordinates of the best move for the player and the best point total achieved by that player. In case of ties, print the move which comes first lexicographically. Use the format shown in the sample output.

Sample Input

```
4
01.1
00..
.01.
...1
4
0.01
0.01
1..0
.1..
0
```

Sample Output

```
(1,2) 2
(2,2) -1
```