1^{o} Aquecimento para Nacional - BH

4 de Novembro de 2016

Sevidor BOCA:
`http://maratona.c3sl.ufpr.br/`

UFPR
UDESC

Instruções Importantes

- Em cada problema, cada arquivo de entrada contém apenas um caso de teste. Sua solução será executada com vários arquivos de entrada.
- Se a solução der erro ou esgotar o tempo limite para um dado arquivo de entrada, você receberá a indicação de erro (estouro de tempo, resposta errada, etc.) para aquele arquivo, e a execução terminará. O arquivo que causou o erro não é identificado. Note que pode haver outros erros, de outros tipos, para outros arquivos de entrada, mas apenas o primeiro erro encontrado é reportado.
- Sua solução será compilada ou executada com a seguinte linha de comando:
  - C: `gcc -static -O2 -lm`
  - C++: `g++ -static -O2 -lm`
  - C++11: `g++ -std=c++11 -static -O2 -lm`
  - Java: `javac`
  - Python: `python3`
  - Pascal: `fpc -Xt -XS -O2`
- Sua solução deve processar cada arquivo de entrada no tempo máximo estipulado para cada problema, dado pela seguinte tabela:

Problema	**Nome**	**Tempo Limite (segundos)**
A	Rhombinoes	1
B	Enumerating Brackets	1
C	Wizard of Odds	1
D	Treasure Hunt	1
E	Planet Destruction	1
F	Laurel Creek	1
G	Trainsorting	1
H	Virtual Friends	1
I	Dominos	1
J	Logo	1

- Todas as linhas, tanto na entrada quanto na saída, terminam com o caractere de fim-de-linha ($\backslash n$), mesmo quando houver apenas uma única linha no arquivo.
- Para submissões em **JAVA**, a classe deverá ter o mesmo nome que o *basename* do problema (leia a linha entre o título e o texto do problema).

A: Rhombinoes

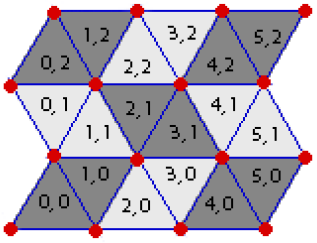
`Arquivo: rhombinoes.[c|cpp|java|py|pas]`

In the game of *Rhombinoes*, you have a board made up entirely of equilateral trianges (see the image), some of which are "*live*" and some are "*dead*". Your goal is to place down as many rhombinoes ("rhombus"-shaped pieces) as possible on the board. Each rhombino should exactly cover two "adjacent" *live* triangles that have a common side, and no two rhombinoes can use the same triangle.

Given the description of the *live* and *dead* triangles of a Rhombino board, what is the maximum number of rhombinoes you can simultaneously place down on the board?

Description of Board

Each triangle in the board has a pair of coordinates (x, y). The bottom-left triangle has coordinates $(0, 0)$ and will always be a triangle with its tip pointed upward. For any given triangle with coordinates (x, y), the triangle adjacent to it on its right-side (if any) has coordinates $(x+1, y)$, and the triangle adjacent to it on its top-side (if any) has coordinates $(x, y+1)$. Left-side and bottom-side adjacency are defined similarly.



Each board has a width W and a height H. A board with width W and height H is the board which consists of all triangles with coordinates (x, y) such that $0 \leq x < W$ and $0 \leq y < H$. For example, the game board in the image has width 6 and height 3. See the image for clarification.

Input

The first line of input contains three space-separated integers W, H, and K. W is the width of the board, H is the height, and K is the number of dead triangles on the board ($1 \leq W \leq 100, 1 \leq H \leq 100, 1 \leq K \leq W \times H \leq 1000$).

Exactly K lines will follow. Each such line will contain a pair of space-separated integers x and y ($0 \leq x < W, 0 \leq y < H$), indicating that the triangle with coordinates (x, y) is a dead triangle. All other triangles are live.

Output

Output a line containing a single integer, the maximum number of rhombinoes you can simultaneously place down on the board.

Sample Input	**Sample Output**
6 3 4 1 1 2 2 4 1 3 0	5

This is the board in the image, with cells $(1, 1)$, $(2, 2)$, $(4, 1)$, and $(3, 0)$ dead.

B: Enumerating Brackets

`Arquivo: brackets.[c|cpp|java|py|pas]`

A *balanced bracket sequence* is a string consisting only of the characters `(` (opening brackets) and `)` (closing brackets) such that each opening bracket has a "matching" closing bracket, and vice versa. For example, `(())()` is a balanced bracket sequence, whereas `(())(()` and `))(()` are not.

Given two bracket sequences A and B of the same length, we say that A is *lexicographically smaller* than B (and write $A < B$) if:

- A and B differ in at least one position, and
- A has a `(` and B has a `)` in the left-most position in which A and B differ

For example `(())()` $<$ `()()()` because they first differ in the second position from the left, and the first string has an `(` in that position, whereas the second string has a `)`. For a given length N, the $<$ operator defines an *ordering* on all balanced bracket sequences of length N. For example, the ordering of the sequences of length 6 is:

1. `((()))`
2. `(()())`
3. `(())()`
4. `()(())`
5. `()()()`

Given a length N and a positive integer M, your task is to find the M^{th} balanced bracket sequence in the ordering.

Input

You will be given an even integer N ($2 \leq N \leq 2000$), and a positive integer M. It is guaranteed that M will be no more than 10^{18} and no more than the number of balanced bracket sequences of length N (whichever is smaller).

Output

Output the M^{th} balanced bracket sequence of length N, when ordered lexicographically.

Sample Input	**Sample Output**
`6 4`	`()(())`

C: Wizard of Odds

`Arquivo: wizard.[c|cpp|java|py|pas]`

You have just completed a brave journey to see *The Wizard of Odds*, who agrees to grant you any wish, so long as you can complete the following puzzle:

The Wizard starts by telling you two integers: N and K. He then secretly selects a number from 1 to N (inclusive), and does not tell you this number.

Your goal is to correctly guess the secret number. Before guessing, you are allowed to ask K "true/false" questions about the number, for example, "Is the number even?" or "Is the number between 7 and 10?", or "Is the number 17 or 22?", or "Is the number prime?". And the Wizard will answer with "true" or "false" to each question. The Wizard will always answer honestly. After answering the K questions, you must guess the number. If you win (guess the number correctly), you will be granted your wish; but if the Wizard wins (you guess incorrectly), you will be turned into a flying monkey.

(Formally, you can think of a "question" as a function from $\{1, 2, ..., N\}$ to {true, false}, and the Wizard will answer by telling you whether the value of the function is true or false for his secret number.)

Assuming that you have been told N and K, can you always exactly determine the Wizard's secret number (and guarantee that you win) using only K questions?

Input

The input consists of a single line containing two integers N and K ($2 \leq N \leq 10^{100}, 0 \leq K \leq N$), separated by a single space. Note: These inputs might NOT fit into a 64-bit integer.

Output

Output "`Your wish is granted!`" (without the quotes) if it is possible for you to guarantee that you win the game (regardless of the number the Wizard picks). Otherwise, print "`You will become a flying monkey!`" (without the quotes) if it is not possible.

Sample Input	**Sample Output**
`8 3`	`Your wish is granted!`

Sample Input	**Sample Output**
`1234567890987654321 2`	`You will become a flying monkey!`

D: Treasure Hunt

`Arquivo: treasure.[c|cpp|java|py|pas]`

Jill has created a new smartphone game that leads players to find a treasure. The app uses the phone's GPS to determine the player's location. The app then tells the player the direction in which to go on a route to find the treasure. When the player reaches some specific location, the app rewards the player with a (virtual) treasure.

Can you help the player determine how long it will take to find the treasure?

Input

The first line of input contains two integers R and C, each between 1 and 200, inclusive. These integers define the number of rows and columns in the playing area, respectively. The next R lines of input describe the playing area. Each line contains exactly C letters, and each letter defines the action to take in each location in the playing area. There are five possible letters: `N` indicates a move to the previous row, `S` indicates a move to the next row, `W` indicates a move to the previous column, `E` indicates a move to the next column, and `T` indicates the location of the treasure. Exactly one location contains the treasure.

Output

The player begins playing at the location in the first column of the first row. The player follows the directions at each location. If the player eventually reaches the treasure by following the directions, output a line containing an integer, the number of moves required to reach the treasure. If the directions cause the player to leave the playing area, output a line containing the word "`Out`" (without the quotes). If the directions cause the player to stay in the playing area but never reach the treasure, output a line containing the word "`Lost`" (without the quotes).

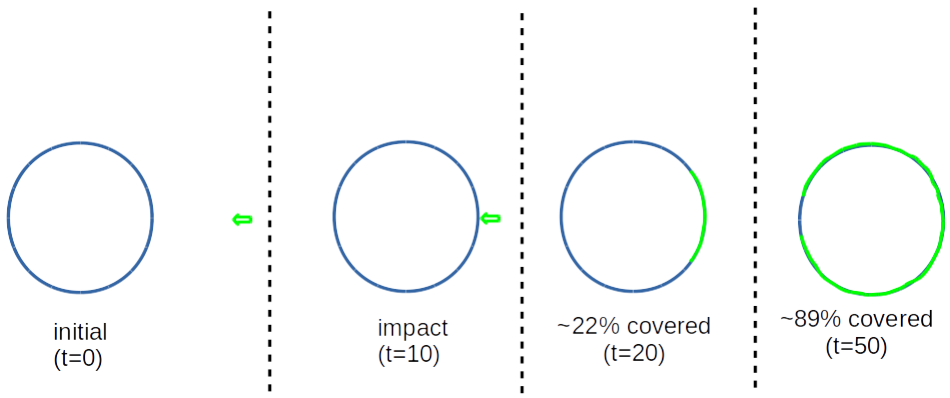
Sample Input	**Sample Output**
2 2 ES TW	3

E: Planet Destruction

`Arquivo: planet.[c|cpp|java|py|pas]`

Darth Vader is back to his favourite hobby: destroying planets (or their population to be precise). He just discovered that the rebel leadership has gathered at a planet named Watooine, so he must act quickly to eliminate the threat. Unfortunately, the Empire is in a pretty bad recession right now, so there aren't any Death Stars left around. Therefore, Vader's plan involves dropping several containers with deadly viruses onto the planet's surface.

To make things simpler for you, we will model this problem in 2D. Watooine has radius R and is centred at $(0, 0)$. There are K Empire spaceships, each of which will simultaneously launch a rocket with a container towards point $(0, 0)$. Once the rocket hits the surface of the planet, a virus will start spreading along the surface in the shape of a circle of ever increasing radius. Since we are modelling the problem in 2D, the virus spreads from the point of impact at the same rate in both directions, clockwise and counterclockwise along the circumference of the circle that represents the surface of the planet. Each spaceship has a custom rocket speed, and each virus has a custom spread speed. Note that each virus spreads along the surface of the planet - it cannot pass through the planet to reach the other end.



Your task is to determine how much time will it take for Watooine to become completely infected - i.e. every point on the planet surface has been reached by a virus. Good luck!

Input

The first line contains T, the number of test cases, followed by the descriptions of the T test cases. Each test case description has the following structure: The first line contains two integers R, the radius of the planet (in meters), and K, the number of spaceships. The next K lines each contain 4 integers: the X-coordinate of the ship, the Y-coordinate of the ship, the rocket speed and the virus spread speed (in meters per second). It is guaranteed that no ships are inside the planet.

Both speeds are between 1 and 1000000, both coordinates are between -1000000 and 1000000, R is between 1 and 1000000. K is between 1 and 10000, and T is between 1 and 100.

Output

For each test case, print one line containing one real number, the time it takes for the planet to become completely infected. Round and print the number with exactly 4 decimal places.

Sample Input	Sample Output
1 100 1 200 0 10 7	54.8799

F: Laurel Creek

`Arquivo: laurel.[c|cpp|java|py|pas]`

Laurel Creek is a perilous river that divides the campus into two halves and contains dangerous inhabitants such as geese and beavers. Your task in this problem is to find a way to cross the river without getting wet.

To do so, you will take advantage of several tree stumps in the middle of the river. A tree stump provides a safe place for you to stand as you ponder your next move. To get from one stump to another, you walk along logs that connect the stumps.

In cases where no log connects to the stump you wish to reach, all is not lost. You may pick up any log adjacent to the stump on which you are standing and put it down somewhere else so that it leads to the stump you wish to reach. In order for a log to be considered adjacent to a stump, it must be oriented in the appropriate direction; for example the log in `S-S` is adjacent to the two stumps, but the log in `S|S` is not considered adjacent to the two stumps.

Each tree stump is located at a point on a square grid. Two stumps are designated as the beginning and end point of the crossing. Any two stumps lying in the same row or column of the grid may be connected by a log. At any point in time, you may perform one of the following legal moves:

- Traverse a log adjacent to the tree stump you are standing on to the tree stump at the opposite end of the log.
- Pick up a log adjacent to the tree stump you are standing on. You may not hold more than one log at a time.
- Put down the log that you are holding so that it connects the stump you are standing on to some other stump. The log must be of precisely the right length to reach the other stump. The log must rest in the water: you may not use a log to connect two stumps if there is a third stump directly between them, or if the log would cross some other log already in the water.

Input

The first line of input contains one integer specifying the number of test cases to follow. Each test case begins with a line containing two integers $1 \leq r \leq 15$ and $1 \leq c \leq 15$ specifying the number of rows and columns in the grid. Each of the next r lines of input contains c characters with the following meaning: The character `S` denotes a stump. The characters `B` and `E` denote the beginning and end stumps of the crossing, respectively. A consecutive sequence of `-` or `|` characters in a line denotes a single log whose length is proportional to the number of symbols. The character `.` denotes an empty grid point containing only water. There will never be more than fifteen stumps in the river.

Output

For each test case, output a line containing a single integer, the minimum number of moves in which the end stump can be reached from the initial configuration. If it is not possible to reach the end stump from the initial configuration, output a line containing the integer 0.

Sample Input	**Sample Output**
1 7 11 ....S...... ....|...... B---S...... ....S.S...E	10

G: Trainsorting

`Arquivo: trainsorting.[c|cpp|java|py|pas]`

Erin is an engineer. She drives trains. She also arranges the cars within each train. She prefers to put the cars in decreasing order of weight, with the heaviest car at the front of the train.

Unfortunately, sorting train cars is not easy. One cannot simply pick up a car and place it somewhere else. It is impractical to insert a car within an existing train. A car may only be added to the beginning and end of the train.

Cars arrive at the train station in a predetermined order. When each car arrives, Erin can add it to the beginning or end of her train, or refuse to add it at all. The resulting train should be as long as possible, but the cars within it must be ordered by weight.

Given the weights of the cars in the order in which they arrive, what is the longest train that Erin can make?

Input

The first line contains an integer $0 \leq n \leq 2000$, the number of cars. Each of the following n lines contains a non-negative integer giving the weight of a car. No two cars have the same weight.

Output

Output a single integer giving the number of cars in the longest train that can be made with the given restrictions.

Sample Input	**Sample Output**
3 1 2 3	3

H: Virtual Friends

`Arquivo: friends.[c|cpp|java|py|pas]`

These days, you can do all sorts of things online. For example, you can use various websites to make virtual friends. For some people, growing their social network (their friends, their friends' friends, their friends' friends' friends, and so on), has become an addictive hobby. Just as some people collect stamps, other people collect virtual friends.

Your task is to observe the interactions on such a website and keep track of the size of each person's network. Assume that every friendship is mutual (if Fred is Barney's friend, then Barney is also Fred's friend).

Input

The first line of input contains one integer specifying the number of test cases to follow. Each test case begins with a line containing an integer F, the number of friendships formed, which is no more than 100000. Each of the following F lines contains the names of two people who have just become friends, separated by a space. A name is a string of 1 to 20 letters (uppercase or lowercase).

Output

Whenever a friendship is formed, print a line containing one integer, the number of people in the social network of the two people who have just become friends.

Sample Input	**Sample Output**
1 3 Fred Barney Barney Betty Betty Wilma	2 3 4

I: Dominos

`Arquivo: dominos.[c|cpp|java|py|pas]`

Dominos are lots of fun. Children like to stand the tiles on their side in long lines. When one domino falls, it knocks down the next one, which knocks down the one after that, all the way down the line. However, sometimes a domino fails to knock the next one down. In that case, we have to knock it down by hand to get the dominos falling again.

Your task is to determine, given the layout of some domino tiles, the minimum number of dominos that must be knocked down by hand in order for all of the dominos to fall.

Input

The first line of input contains one integer specifying the number of test cases to follow. Each test case begins with a line containing two integers, each no larger than 100000. The first integer n is the number of domino tiles and the second integer m is the number of lines to follow in the test case. The domino tiles are numbered from 1 to n. Each of the following lines contains two integers x and y indicating that if domino number x falls, it will cause domino number y to fall as well.

Output

For each test case, output a line containing one integer, the minimum number of dominos that must be knocked over by hand in order for all the dominos to fall.

Sample Input	**Sample Output**
1 3 2 1 2 2 3	1

J: Logo

`Arquivo: logo.[c|cpp|java|py|pas]`

Logo is a programming language built around a turtle. Commands in the language cause the turtle to move. The turtle has a pen attached to it. As the turtle moves, it draw lines on the page. The turtle can be programmed to draw interesting pictures.

We are interested in making the turtle draw a picture, then return to the point that it started from. For example, we could give the turtle the following program:

```
fd 100 lt 120 fd 100 lt 120 fd 100
```

The command `fd` causes the turtle to move forward by the specified number of units. The command `lt` causes the turtle to turn left by the specified number of degrees. Thus the above commands cause the turtle to draw an equilateral triangle with sides 100 units long. Notice that after executing the commands, the turtle ends up in the same place as it started. The turtle understands two additional commands. The command `bk` causes the turtle to move backward by the specified number of units. The command `rt` causes the turtle to turn right by the specified number of degrees.

After executing many commands, the turtle can get lost, far away from its starting position. Your task is to determine the straight-line distance from the turtle's position at the end of its journey back to the position that it started from.

Input

The first line of input contains one integer specifying the number of test cases to follow. Each test case starts with a line containing one integer, the number of commands to follow. The commands follow, one on each line. Each test case will contain no more than 1000 commands.

Output

For each test case, output a line containing a single integer, the distance rounded to the nearest unit.

Sample Input	**Sample Output**
1 5 fd 100 lt 120 fd 100 lt 120 fd 100	0