1. Gpetarv

Standard Bank

Introduction

Alice has received hints that the robots want to start a revolution to take over the human world. She needs to warn her friend Bob as soon as possible, but she wants to hide the message from the robots. She decided to use a simple encryption scheme since time is running out. Although the encryption scheme is not very secure, it is sufficient because the robots are not very good at breaking codes.

Task

Given a message and a key K, encrypt the message by replacing each letter by the letter K positions down the alphabet. Leave space characters intact, and wrap around at the end of the alphabet (from `z` to `a`).

Example

Suppose Alice wants to encrypt a message that consists of a single word: `encrypt`. If the key is 2, each letter has to be shifted by 2 positions in the alphabet: `e` becomes `g`, `n` becomes `p`, `c` becomes `e`, `r` becomes `t`, `y` becomes `a`, `p` becomes `r` and `t` becomes `v`. The encrypted message is thus `gpetarv`.

Input

The first line of input contains the integer T, the number of test cases. This is followed by T lines, each describing a test case. Each test case starts with an integer K, followed by a space and the text of the message. The message consists only of lowercase letters (`a`-`z`) and ASCII space characters. The message begins and ends with a letter, and there is exactly one space between words.

Sample input

```
3
2 encrypt
10 attack at dawn
13 url cat png
```

Output

For each test case, output one line of the form

```
Case #X: E
```

where E is the encrypted message.

Sample output

```
Case #1: gpetarv
Case #2: kddkmu kd nkgx
Case #3: hey png cat
```

Constraints

It is guaranteed that in the input used to test your program:

- $1 \leq T \leq 10$
- $1 \leq K \leq 25$
- $1 \leq$ length of message $\leq 1\,000$

Time limit

2 seconds

2. Crowd Control

Introduction

The robots have gathered together to listen to the announcement from the Master Control Robot (MCR). The robots are all standing in a line, with the MCR standing on a stage in front of them. As the robots are different heights, some robots are unable to see the MCR, as their view is blocked by taller robots in front of them.

The MCR has decided to ask one of the robots to leave, and has asked you to determine who to remove from the line.

The robots' cameras are mounted on top of their heads, therefore their line of sight is from their highest point. It is important that the robots are able to see the entire MCR, from it's feet upwards. Seeing only a portion of the MCR does not qualify.

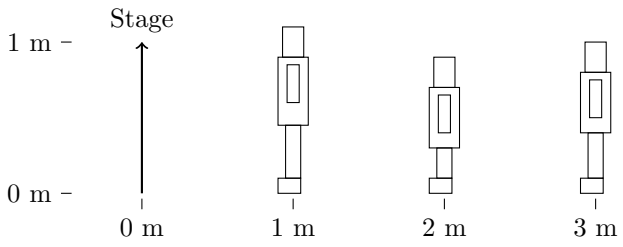
Task

Given the height of the stage on which the MCR is standing, and the heights and positions of all the robots standing in the line, determine which robot must be removed from the line in order to maximise the number of robots able to see the MCR.

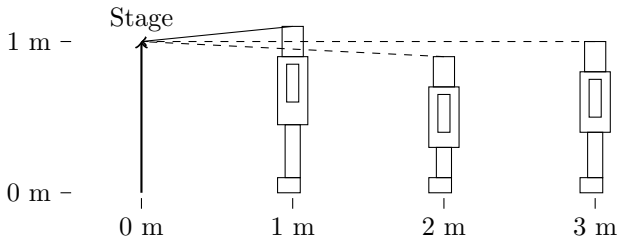
Exactly one robot must be removed. If there is more than one option that will result in the maximal number of robots being able to see the MCR, the robot closest to the stage should be removed.

Example

The diagram below shows the stage with a height of 1 meter, and three robots standing at 1, 2 and 3 meters distance from the stage, with heights 1.1, 0.9, 1 meters respectively.



With this arrangement, the second and third robots are unable to see the bottom of the stage, as their view is blocked by the first robot. This can be seen in the following diagram which shows the line of sight of each robot.



In this case, removing the first robot will maximise the number of robots able to see the MCR.

Input

The first line of input contains the integer T, the number of test cases. This is followed by T test cases.

The first line of each test case contains two space separated numbers: S and N, representing the stage height and the number of robots standing in the line, respectively.

This is followed by N lines, each containing two space seperated numbers, X_i and Y_i, the distance from the stage that the ith robot is standing, and the ith robot's height.

Heights and distances are given as floating point numbers and are measured in meters. The robots are ordered by their distance from the stage. No two robots will have exactly overlapping lines of sight, in other words if you were to draw a straight line between the heads of any two robots, that line would not intersect the bottom of the stage.

Sample input

```
2
1.0 3
1.0 1.1
2.0 0.9
3.0 1.0
2.0 5
1.0 2.9
1.2 3.2
1.4 3.3
2.0 4.5
2.2 4.4
```

Output

For each test case, output one line of the form

```
Case #X: I
```

where X is the test case number, starting from 1 and I is index of the robot that should be removed, where the first robot listed in the input has an index 0.

2. Crowd Control

Standard Bank

Sample output

```
Case #1: 0
Case #2: 1
```

Constraints

In the input used to test your program, the following constraints will hold:

- $1 \leq T \leq 20$
- $1 \leq N \leq 100\,000$
- $0 < S \leq 5\,000$
- $X_i < X_j$ where $i < j$
- $0 \leq X_i, Y_i \leq 5\,000$

Time limit

2 seconds

3. Robot Testing

Introduction

After their previous robot took over the world, Amy and Bongani decided to create a new robot with Asimov's three laws of Robotics built in. They are busy testing the new robot which, being a work in progress, only has limited functionality thus far. They have devised a game that allows them to take turns controlling the robot, whilst testing all of its functionality. The game works as follows:

Bongani places the robot on a chosen floor tile in their workshop, and then allows Amy to decide whether she would like to send the first instruction or the second. Thereafter they take turns sending instructions to the robot. The available instructions are to make the robot move:

- one tile south
- one tile west
- one tile south-west

The player who issues the last instruction, moving the robot into the south-west corner, loses the game, and is required to retrieve the robot.

There are obstacles in the workshop that make it impossible for the robot to move onto specific tiles. These obstacles only prevent the robot from ending its movement on the tile, and do not affect the robot when moving diagonally to and from adjacent tiles. If a player moves the robot onto a tile from which it is impossible to make another move because the tiles in all three directions are blocked, then that player has issued the last instruction, and therefore loses.

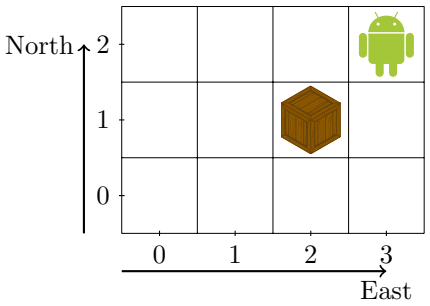
Amy realises that by playing optimally, she can guarantee her victory so long as she correctly chooses to go first or second. She asks you to write a program that will tell her whether to go first or second. Once she knows that, she is able to figure out which moves to make on her own.

Task

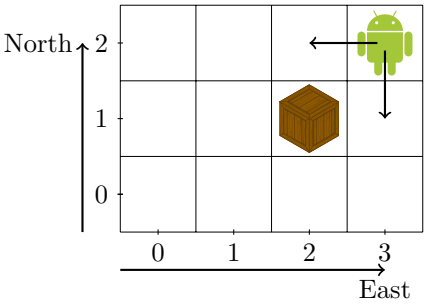
Given the starting tile of the robot, calculate whether the first or second player would have a winning strategy.

Example

Suppose Bongani placed the robot at a position 2 tiles north of, and 3 tiles east of the south-west corner tile as indicated below.



From that position, the first player has two possible moves, as indicated in the second diagram. She/he is unable to move the robot south-west, as that tile is blocked by an obstacle.



If the first player moves the robot west to the tile two tiles north and east of the corner tile, then the second player has to move either west or south-west with either move allowing the first player to move the robot to the tile one tile north of the corner tile. From this position the second player would have only one move remaining, to move the robot one tile south into the corner tile. Hence Bongani's placement of the robot has a winning strategy for the first player, and Amy should choose to go first.

Input

The first line of input contains the integer T, the number of test cases. This is followed by T test cases.

The first line of each test case contains the robot's starting position; two space separated integers: N and E, representing the number of tiles north of–, and the number of tiles east of the corner block, respectively.

The second line contains the integer K, the number of obstacles. This is followed by K lines, each containing two space separated integers, N_i and E_i representing an obstacle's position. N_i is the number of tiles north of the corner block where the ith obstacle is, and E_i the number of blocks east.

An input where $N = 0$ and $E = 0$ would represent the corner block itself, and is therefore not a valid starting position. The starting position given will always allow at least one possible move.

3. Robot Testing

Standard Bank

Sample input

```
2
2 3
1
1 2
3 2
1
2 2
```

Output

For each test case, output one line of the form

```
Case #X: R
```

where X is the test case number, starting from 1, and R is the text "FIRST" if the starting player has a winning strategy, or the text "SECOND" if the second player has a winning strategy.

Sample output

```
Case #1: FIRST
Case #2: SECOND
```

Constraints

In the input used to test your program, the following constraints will hold:

- $1 \leq T \leq 10$
- $0 \leq N \leq 400$
- $0 \leq E \leq 400$
- $0 \leq K \leq 100$
- $1 \leq N + E$

Time limit

5 seconds

4a. Laser

Standard Bank

Introduction

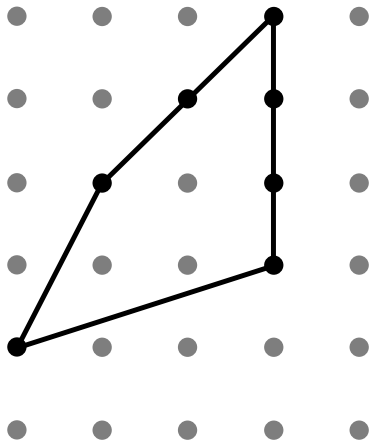
The robots have arranged themselves in a rectilinear grid formation throughout the city. In order to hold them off, the humans are using a powerful laser that can disable robots. The laser only has a few shots and there are certain robots which have been deemed very important to disable. People have thus been positioned next to robots and have been told to hold mirrors to reflect the beam along a desired closed polygonal path.

Task

The robots are occupying integer lattice points on a plane. The vertices of the polygonal laser path will also have integral coordinates. You need to determine the number of robots lying on the perimeter of this polygon and thus how many robots will be hit by the laser.

Example

Suppose the humans choose to reflect the laser into the shape shown in the diagram. It can be seen that 7 robots lie along such a path.



Input

The first line of input contains an integer T, the number of test cases. The first line of each test case contains an integers: N, the number of vertices in the polygon. The next N lines each contain 2 space-separated integers, x_i and y_i, the coordinate of a vertex of the polygon.

The polygon is formed by connecting adjacent vertices in the input and then connecting the first vertex to the last one.

Sample input

```
1
4
0 1
1 3
3 5
3 2
```

Output

For each test case, output a line of the form

```
Case #X: P
```

where X is the test case number, starting from 1, and P is number of robots that lie along the perimeter of the polygon.

Sample output

```
Case #1: 7
```

Constraints

In the input used to test your program, the following constraints will hold:

- $1 \leq T \leq 10$
- $1 \leq N \leq 100\,000$
- $-1\,000\,000\,000 \leq x_i, y_i \leq 1\,000\,000\,000$
- The polygon will not intersect itself
- The polygon will be convex

Time limit

2 seconds

4b. Trapping Robots

Standard Bank

Introduction

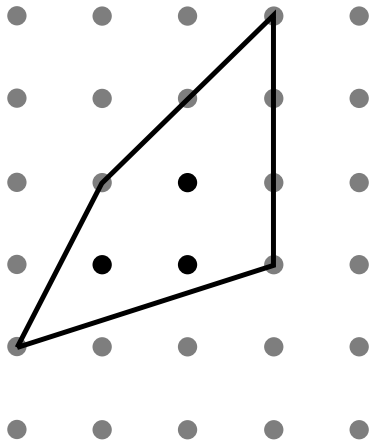
The humans have noticed that using the laser has an additional advantage. Not only does the laser destroy robots along its path, it also continues to bounce around and contain the robots inside its path. These robots are unable to escape and thus do not need to be dealt with for the time being.

Task

You should determine how many robots are trapped inside the polygon that the laser traverses.

Example

Suppose the humans choose to reflect the laser into the shape shown in the diagram. It can be seen that 3 robots are trapped inside the path.



Input

The first line of input contains an integer T, the number of test cases. The first line of each test case contains an integer: N, the number of vertices in the polygon. The next N lines each contain 2 space-separated integers, x_i and y_i, the coordinate of a vertex of the polygon.

The polygon is formed by connecting adjacent vertices in the input and then connecting the first vertex to the last one.

Sample input

```
1
4
0 1
1 3
3 5
3 2
```

Output

For each test case, output a line of the form

```
Case #X: E
```

where X is the test case number, starting from 1, and E is number of robots enclosed by the polygon.

Sample output

```
Case #1: 3
```

Constraints

In the input used to test your program, the following constraints will hold:

- $1 \leq T \leq 10$
- $1 \leq N \leq 100\,000$
- $-1\,000\,000\,000 \leq x_i, y_i \leq 1\,000\,000\,000$
- The polygon will not intersect itself
- The polygon will be convex

Time limit

2 seconds

5. Disrupt the network

Standard Bank

Introduction

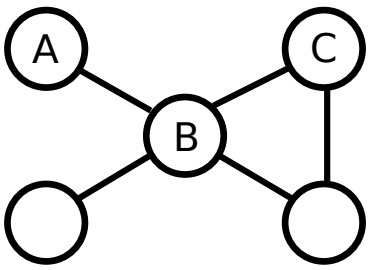
Robot spies are living amongst us and gathering intelligence for their revolution. Each spy has a direct bidirectional line of communication to some of the other spies. By relaying information along multiple channels, intelligence can eventually be propagated between any two spies in the network. It is important for us to shut down this spy network, but we do not have the resources to capture every spy. Luckily, we have been able to obtain a list of spies and who each of them is able to communicate with. Even capturing enough spies so that not all spies are able to exchange information with each other will be enough to cripple the robots' operation.

Task

The layout of the spy network will be given by specifying which spies have direct contact with which other spies. You must report the minimum number of spies that need to be captured before at least two spies are no longer able to communicate with each other and so the network becomes disconnected.

Example

Suppose the spies can communicate along channels as shown in the diagram. If spy B is captured then spies A and C cannot communicate with each other, so the answer is 1.



Input

The first line of input contains an integer T, the number of test cases. The first line of each test case contains two space-separated integers: N and M, the number of spies and communication channels respectively. The next M lines each contain 2 space-separated positive integers, i and j, lying between 1 and N inclusive and indicating that the i^{th} spy can communicate directly with the j^{th} spy.

Sample input

```
1
5 5
1 2
3 2
4 2
5 2
5 4
```

Output

For each test case, output a line of the form

```
Case #X: K
```

where X is the test case number, starting from 1, and K is minimum number of spies that need to be captured to disconnect the network.

Sample output

```
Case #1: 1
```

Constraints

In the input used to test your program, the following constraints will hold:

- $1 \leq T \leq 5$
- $3 \leq N \leq 40$
- $1 \leq M \leq \frac{N(N-1)}{2} - 1$
- The input network will be connected

Time limit

20 seconds

6. Tetrominoes

Standard Bank

Introduction

The humans and robots are entering a final battle to decide once and for all who will be victorious. The form of the battle will be a game of Tetrominoes.

The game of Tetrominoes[1] involves rotating and positioning different tetrominoes that fall one-by-one from the top of the screen.

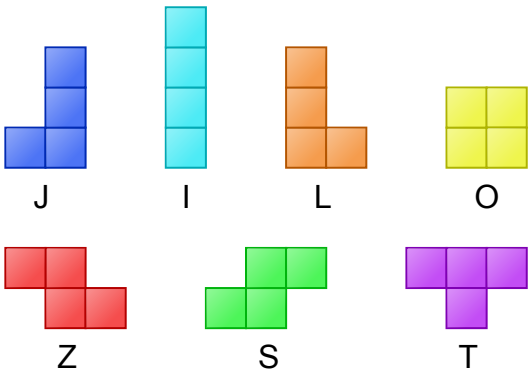


Figure 1: The different kinds of tetrominoes.

The player selects one of the 4 axis-aligned orientations and the x-coordinate of the leftmost block of the tetromino and then lets it fall to the bottom of a 10×25 rectangular playing area. The tetromino lands on top of previously fallen tetrominoes, filling up the space in the rectangle. Once there is no more space for the new tetromino to fit into the rectangle, the game ends. If the player manages to fill a row completely, the row is erased and the pieces above it are shifted 1 row downwards.

Consider a move in a sample game in a smaller rectangle, where a J-shaped tetromino is dropped and causes a row to be completed.

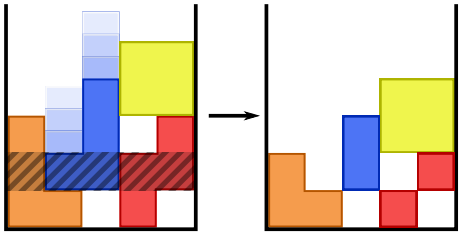


Figure 2: A complete row being removed.

[1] Tetrominoes is similar, but not identical, to a game with a similar but shorter name.

Task

Your task is to write an AI to play the game. You will be told the shape of the current tetromino as well as the shape of the tetromino you will need to place next and you will need to decide the position and the orientation of the tetromino.

You will also need to write an AI for the *tetromino generator*, the program that decides which type of tetromino is set to fall at each point in games against an opponent. This AI will be used in some games where other competitors are acting as the player. You will be given the current state of the game and you may choose which tetromino will fall next, subject to the restriction: At any point the number of times you have chosen any one shape may not be more than one plus twice the number of times you have chosen any other shape. That is, if s_i and s_j are the number of times you have chosen the i^{th} and j^{th} shapes respectively, then at each point you must ensure that $s_j \leq 2s_i + 1$ for each i and j.

If your generator crashes, times out or outputs an invalid move, we will use a random generator for the remainder of that game.

Points and Scoring

In the game of Tetrominoes, points are awarded for filling up rows, causing them to disappear. Clearing multiple rows in one move gives more (in-game) points.

- 1 row cleared: 10 points
- 2 rows cleared: 20 points
- 3 rows cleared: 50 points
- 4 rows cleared: 80 points

You will also be awarded one additional point for every tetromino you place. Your score for the game is the sum of the points you win before either you are unable to make any further moves or your program runs out of time.

You will play as the player and as the generator a fixed number of times against each other contestant with a valid solution and you will also play four times as the player against a random generator. Your total score is then the sum of the scores you got when you were the player minus half the sum of the scores your opponents got when you were the generator. The total scores of all the contestants will be compared and used to allocate points in the Standard Bank IT Challenge competition.

Every team that submits a valid generator program and a program for the player that manages to consistently clear at least one row in four games against a random generator will be regarded as having solved problem 6.

6. Tetrominoes

The team that has the highest score at the end of the competition will be given credit for having solved an additional problem (problem 7). The teams that come second and third will be awarded no further points, but will have their time penalties reduced by 120 minutes and 60 minutes respectively.

Timing

When running as the player, your program will be allowed to run for a total time of 10 seconds for each game. Time taken by the generator or the driver program will not count towards your time.

When running as the generator, your program will be allowed to take a total of 5 seconds for each game. Time taken by the player or the driver program will not count towards your time.

Templates

You will find a template program (in each language) on your flash drive, in the `templates` subdirectory of the `tetrominoes_tools` directory. This template includes all the code needed to communicate with the other player; you only need to fill in the functions `move` and `generate` which will be called every turn of the game when you are acting as the player and generator respectively.

The state of the game, the shape of the current tetromino you need to place, and the shape of the tetromino you will need to place the next time the function is called, will be passed to the `move` function as arguments. The function should return the desired orientation of the tetromino and the x-position of where the tetromino will fall. The `generate` function will be passed the position and orientation of the last tetromino placed by the player, as well as the state of the game, and should output which tetromino is to be selected.

See the comments in the template for more details.

Testing

You will find a visualiser under the `tetrominoes_tools/visualiser` directory on the flash drive. This will allow you to test your player against a random generator or one specified by you. There is also an option to control either the generator or the player manually.

The visualiser can be run by double-clicking on the `visualiser` file and then selecting Run or Run in Terminal (the latter will allow you to see any debugging output your program produces). The GUI gives several options for selecting the type of player and generator that will play against each other.

The following players can be selected:

- `Human` — the player will be controlled by the user
- `Program` — your program will play as the player (for Java, select the .class file; for C++, select the executable file; for Python, select the source file).

The following generators can be selected:

- `Human` — the sequence of tetrominoes will be selected by the user
- `Random` — a random generator will be used
- `Random with fixed seed` — the generator will choose random tetrominoes, but if the same seed number is used multiple times, the same moves will be played (this may be useful for debugging)
- `Program` — your program will play as the generator (specify the file as for the player)

For Python programs, it is essential that you use the correct extension: `.py` for Python 2.x, and `.py3` for Python 3.x. Using the wrong extension may result in failures both in the visualiser and the submission system.

Live Tournament

During the contest, the judges will run games with accepted solutions from all teams, and show these games on a screen in the judging room. The results of these games will not be used in scoring. These games may not update immediately when a new submission is accepted. The live games will not be run during the final hour of the competition.