Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
Masaryk University

Czech Technical University in Prague

Technical University of Ostrava
University of Žilina
University of West Bohemia

CTU Open Contest 2017

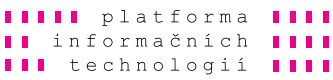
Welcome to CTU Open 2017 programming contest!

This year's contest theme is Amusement Park. Some attractions in the park are easy and some are not, some problems at the contest are easy and some are not. When you are stuck at an attraction you can't leave it, when you are stuck with a problem you can't leave it. Once you master all attractions, you return gladly to the park, once you master all problems, you return gladly to the contest.

Your programs can be written in C, C++, Java or Python programming languages. The choice is yours but you will be fully responsible for the correctness and efficiency of your solutions. We need the correct answer produced by your code in some appropriate time. Nothing else matters. You may choose any algorithm and any programming style.

All programs will read text from the standard input only. All inputs are terminated by the end of file mark. The results must be written to the standard output. You are not allowed to use any other files, communicate over network, or create processes. Input and output formats are described in problem statements and must be strictly followed. Numerical values are separated by one space, if not specified otherwise. Do not print anything more than required. Each printed text line (including the last one) should be terminated by a newline character ("\n"), which is not considered to be a part of that line.

Good luck, thank you for coming to 2017 contest, and see you again at 2018 contest!



This set consists of twelve sheets of paper (including this one) containing eleven problems. Please make sure that you have the complete set.

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
Masaryk University

Czech Technical University in Prague

Technical University of Ostrava
University of Žilina
University of West Bohemia

CTU Open Contest 2017

Amusement Anticipation

`amusement.c`, `amusement.cpp`, `amusement.c11`, `Amusement.java`, `amusement.py`

It is was late Saturday morning at the end of October. The amusement park was going to be open that afternoon for the first time after months of prolonged reconstruction.

Josse and Murry were sitting in the basement of the office building at the park gate. They have just finished debugging their advanced AI system management of all fantastic attractions in the park. "Work is done," said Josse. "Now for some amusement. Can you think of any algorithmic problem?"

"Yes, of course," replied Murry and grinned at his friend from his heavily cluttered desk. "Consider, for example, finite sequences of numbers. For me, the most interesting sequences are those that end with a long arithmetic sequence. In other words, I like arithmetic sequences that span from some index to the very last member. As there may be many such subsequences, in order to truly appreciate how interesting some sequence is, it is necessary to determine which one is the longest. Here is your initial sequence. You have to find the start of the longest continuous arithmetic subsequence spanning to its end."

"OK," said Josse when he marked the correct place in the sequence. "That was easy. What next?" "What next?" Repeated Murry and hesitated for a moment. Then he raised himself determinedly from the chair. "Let's go out to the park and find some tougher problems there!"

Input Specification

There are more test cases. Each case consists of two lines. The first line contains one integer N ($1 \leq N \leq 1\,000$) specifying the length of the sequence. The second line contains a sequence of N integers X_i ($0 \leq X_i \leq 10^9$), separated by spaces.

Output Specification

For each test case, print a single line with the index of the first member of the longest continuous arithmetic subsequence that spans to the end of the given sequence. The index of the first element in the sequence is always 1.

Sample Input

```
5
1 2 3 4 5
7
1 2 3 4 5 8 8
3
7 5 2
```

Output for Sample Input

```
1
6
2
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
Masaryk University

Czech Technical University in Prague

Technical University of Ostrava
University of Žilina
University of West Bohemia

CTU Open Contest 2017

Pond Cascade

`cascade.c`, `cascade.cpp`, `cascade.c11`, `Cascade.java`, `cascade.py`

The cascade of water slides has been installed in the park recently and it has to be tested. The cascade consists of some number of reservoirs, or "ponds" for short, which are linked into a single sequence connected by water slides. Visitors are expected to start the journey in the topmost pond, then to be washed into subsequent lower ponds, and finally end the journey in the last pond of the cascade, which is also the lowest pond of the cascade. The journey spans all ponds in the cascade.

To test the cascade, each pond has to be completely filled with water. Each pond is attached to a pipe with a valve which, when opened, pours water into the pond. The sizes and capacities of different ponds are different. Fortunately, at least the pipes and the valves are standardized. Therefore, the rate at which water is poured through the valve into a pond is the same for all ponds.

When a pond is filled, water overflows and continues to flow into the next pond. If that pond is also filled, water continues to the next subsequent pond, and so on, until it either reaches some pond which has not been completely filled yet, or it overflows the lowest pond and sinks in the drain at the bottom of the cascade. The time in which the overflowing water reaches the next pond is considered to be negligible. The test starts at time 0 with all ponds being empty. Then, all valves are opened simultaneously. The test stops and the valves are closed when all ponds are filled by water.

The pond capacities and valves flow rate are known, you have to determine the moment when the lowest pond starts to overflow and the first moment when all ponds are filled.

Input Specification

There are more test cases. Each test case consists of two lines. The first line contains two integers N ($1 \leq N \leq 10^5$, $1 \leq F \leq 10^9$) separated by space. N is the number of ponds, F is the rate of water flow through each valve. We consider the flow to be expressed in litres per second. The second line contains a sequence of N integers C_i ($1 \leq C_i \leq 10^9$) separated by spaces and representing the pond capacities expressed in litres. The sequence reflects the order of ponds from the topmost one to the lowest one.

Output Specification

For each test case, print a single line with two numbers separated by space. The first number represents the time in which the lowest pond in the cascade starts to overflow. The second number represents the duration of the test. Both times should be printed in seconds and should be accurate within an absolute or relative error of 10^{-6}.

Sample Input

```
4 2
1 2 3 4
4 3
10 7 3 2
```

Output for Sample Input

```
1.25 1.25
0.66666667 3.33333333
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
Masaryk University

Czech Technical University in Prague

Technical University of Ostrava
University of Žilina
University of West Bohemia

CTU Open Contest 2017

Chessboard Dancing

`chessboard.c`, `chessboard.cpp`, `chessboard.c11`, `Chessboard.java`, `chessboard.py`

Amusement park anniversary organizing group invited a prominent dance company to perform surprise theme dances at the anniversary celebrations. The company prepared four dances with chess theme, conveniently named King performance, Knight performance, Bishop performance and Rook performance.

Each dance is performed by a number of teams of dancers. The unusual setting in the amusement park seems to cause difficulties to a few sensitive company members, however. The main choreographer holds that if a dancer, during a performance, has in his/her sight another dancer of the same team it might sometimes confuse the dancer and tempt him/her to follow erroneously the movements of the teammate. The implication is that the dancers of the same team should be suitably dispersed over the dancing area to be mutually obscured by the members of other teams. Moreover, any arrangement of this kind cannot be shared among the performances, because their styles and movements differ dramatically from each other.

For each of the four dances, the organizing group eventually managed to formulate exact limiting conditions based on company demands:

The performance takes place on a regular $S \times S$ square grid of markers on the floor.

Each dancer occupies one marker, no marker is left unoccupied, no two dancers share a common marker and each dancer remains at the same marker during the performance. Each dancer belongs to exactly one team.

Suppose that the center of each marker coincides with some integer lattice point in a Cartesian grid and that the distance between the center of any marker and the center of its closest neighbour marker is 1.

Let us consider two different markers A and B with their respective centers located at Cartesian coordinates (x_1, y_1) and (x_2, y_2).

- In the King performance, if $|x_1 - x_2| < 2$ and $|y_1 - y_2| < 2$, then A and B must be occupied by members of different teams.
- In the Knight performance, if $|x_1 - x_2| \cdot |y_1 - y_2| = 2$, then A and B must be occupied by members of different teams.
- In the Bishop performance, if $|x_1 - x_2| = |y_1 - y_2|$, then A and B must be occupied by members of different teams.
- In the Rook performance, if $|x_1 - x_2| \cdot |y_1 - y_2| = 0$, then A and B must be occupied by members of different teams.

For each of the four dances, the organizing group wants to know the minimum possible number of teams needed to perform the dance.

Input Specification

There are more test cases. Each test case consists of a single line with integer S ($1 \leq S \leq 1000$) followed by a space and one of four capital letters "`K`", "`N`", "`B`", "`R`". S specifies the size of the marker grid and letters denote King performance, Knight performance, Bishop performance, or Rook performance, respectively.

Output Specification

For each test case, print a single line with one integer denoting the minimum number of teams which are necessary to perform the given dance on a marker grid of the given size.

Sample Input

```
2 N
8 R
2 B
1 K
```

Output for Sample Input

```
1
8
2
1
```

Czech ACM Student Chapter
Charles University in Prague
Slovak University of Technology
Masaryk University

Czech Technical University in Prague
Technical University of Ostrava
University of Žilina
University of West Bohemia

CTU Open Contest 2017

Equinox Roller Coaster

`equiroaster.c`, `equiroaster.cpp`, `equiroaster.c11`, `Equiroaster.java`, `equiroaster.py`

Equiroaster is an abbreviation of EQUInox ROller coASTER, a new and grand attraction which is going to be built in the park vicinity the next year.

The name of the roller coaster comes from the fact that it is the first roller coaster of its class with a construction and shape directly linked to major astronomical events. Twice a year, at equinox, the visitors will enjoy an unforgettable ride enhanced by additional visual effects. Also, the advertising value of equinox rides is expected to be very high.

To meet the expectations, Equiroaster must be built very precisely. The alignment of its tracks with East-West direction must be precise. The tracks will be supported by four main piers and the alignment of the Southern and the Northern pair of piers with East-West direction must be equally precise.

Equiroaster will be built in the field close to other park attractions. The field geology is suspected to be somewhat less favorable to large building projects and thus a detailed survey of the field was commissioned. A grid perfectly aligned with East-West direction (or equivalently, with North-South direction) and with grid points one meter apart was projected onto the field. Each grid point was geologically evaluated and marked as stable or unstable.

All four main piers of Equiroaster must be built on stable grid points. It is hoped that there are enough stable grid points in the field to allow for a really extensive design of the coaster. However, the building technology dictates that the base of the four main piers must form a perfect square in order to reliably support the massive weight of the coaster.

The task is to determine the maximum possible distance between the piers on one side of the coaster according to the specifications described above.

Input Specification

There are more test cases. Each case starts with a line containing one integer N specifying the number of stable grid points ($1 \leq N \leq 100\,000$). Next, there are N lines each of which specifies x- and y-coordinates of a grid point. You should suppose that the grid axes are aligned with East-West and with South-North directions and also that the coordinates are given in whole meters. All grid points in one test case are unique. The absolute value of each coordinate is at most 10^9.

Output Specification

For each test case, print a single line with one integer denoting the maximum possible distance in meters between the centers of two piers on one side of the coaster. If it is not possible to place the piers on stable grid points according to the specifications, print 0 on the output line.

Sample Input

```
3
0 0
0 1
1 0
8
0 10
0 20
0 30
10 0
10 10
10 20
20 0
20 10
```

Output for Sample Input

```
0
10
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
Masaryk University

Czech Technical University in Prague

Technical University of Ostrava
University of Žilina
University of West Bohemia

CTU Open Contest 2017

Forest Picture

`forest.c`, `forest.cpp`, `forest.c11`, `Forest.java`, `forest.py`

The game "Draw a forest picture" is quite popular among younger visitors of the amusement park. The number of players in the game is virtually unlimited and nearly everybody becomes a winner. The game is simple. At the beginning, a leader of the game describes briefly a picture of a forest which he or she had seen recently. Then the players are given some paper and crayons and they have to reproduce the image as best as they can. Everybody who hands in at least partial image of virtually any piece of any forest anywhere on Earth depicted in any style wins a small reward in the form of a chocolate or a fruit.

In this problem, you have to implement an imitation of the game. As you are much more experienced than the younger players, your drawing has to meet the specifications exactly. The image you have to reproduce depicts a glade — an open area within a woodland which is somewhat less populated by the trees than the surrounding thick forest. The image has to be printed in the "ascii-art" style.

An $M \times M$ image is represented by a canvas consisting of M rows, each row containing M characters. Each pixel in the image is represented by some printable ASCII character on the canvas. The coordinates of the pixels in the image correspond to the coordinates of the characters on the canvas. The coordinates of the pixels in the bottom left corner and in the top right corner of the image are $(0, 0)$ and $(M - 1, M - 1)$, respectively. The x-coordinate of the pixel in the bottom right corner of the image is $M - 1$.

Each pixel in the image depicts either grass or a part of a standing tree or a tree stump. A pixel depicting grass is represented by a single dot character ("`.`", ASCII code 46) on the canvas. Standing trees and tree stumps are depicted by more pixels, their representation on the canvas follows.

A standing tree has a positive height S and it consists of four parts: The roots, the tree trunk, the branches, and the treetop. The roots are represented by three horizontally adjacent characters: underscore, vertical bar, underscore ("`_|_`", ASCII codes 95, 124, 95). The tree trunk is represented by S vertically adjacent vertical bars ("`|`", ASCII code 124) located immediately above the center of the tree roots. The branches consist of S left branches located immediately to the left of the tree trunk, and S right branches located immediately to the right of the tree trunk. Each branch is adjacent to the tree trunk. A left branch is represented by a single forward slash character ("`/`", ASCII code 47), a right branch is represented by a single backslash character ("`\`", ASCII code 92). A treetop is represented by a single caret character ("`^`", ASCII code 94) located immediately above the topmost character of the tree trunk.

A tree stump consists of three horizontally adjacent pixels represented by characters underscore, lowercase letter "`o`", underscore ("`_o_`", ASCII codes 95, 111, 95).

Note that a standing tree or a tree stump may appear in the image only partially or may not appear in the image at all, depending on its coordinates. See the sample data below for additional illustration of this fact.

Input Specification

There are more test cases. Each case starts with a line containing two integers M, N separated by space ($1 \leq M \leq 100$, $1 \leq N \leq 10^5$). Next, there are N lines, each line contains a triple of integers S, X, Y, separated by spaces, which describe one standing tree or one tree stump. The values of X and Y represent the coordinates of the center of either tree roots or a tree stump. In case of $S = 0$ the triple describes a stump. In case of $S > 0$ the triple describes a standing tree with height S. It holds $0 \leq S \leq 9, -10^9 \leq X, Y \leq 10^9$.

It is guaranteed that no parts of two different standing trees and/or tree stumps should be depicted by the same pixel.

Output Specification

For each test case, print the canvas with the image of the glade. The top row of the canvas should be the first printed row of the image. The bottom row of the canvas should be the last printed row of the image. The printout should be decorated by a square border made of asterisk characters ("*", ASCII code 42), the thickness of the border should be one pixel. The border should frame the canvas tightly, that is, there should be no spaces between the border and canvas neither horizontally nor vertically. Print one empty line after each case.

Sample Input

```
3 2
0 5 5
9 1 0
8 10
3 3 2
0 2 1
1 -1 -1
0 -1 2
3 0 6
6 4 7
0 7 4
3 8 -1
5 5 -5
9 2 -10
```

Output for Sample Input

```
*****
*/|\*
*/|\*
*_|_*
*****

**********
*|\._|_..*
*|_.^....*
*../|\...*
*../|\._o*
*../|\...*
*_._|_../*
*._o_.^./*
*\.^./|\/*
**********
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
Masaryk University

Czech Technical University in Prague

Technical University of Ostrava
University of Žilina
University of West Bohemia

CTU Open Contest 2017

Shooting Gallery

`gallery.c`, `gallery.cpp`, `gallery.c11`, `Gallery.java`, `gallery.py`

The popular shooting gallery in the park is an attraction which is simple and intricate at the same time. It consists of a row of ducks sitting on one horizontal perch. The ducks are not necessarily all of the same kind, some of them may belong to different, albeit easily distinguishable, species.

The shooting proceeds in rounds. In each round, the shooter fires two shots. Each shot can hit at most one duck. A round is considered to be a good round if the shooter hits two ducks of the same species. A round is considered to be a bad round if the shooter either hits less than two ducks or if he/she hits two ducks of different species.

The shooter can ask for a round if and only if both following conditions are met:

- There are at least two ducks of the same species on the perch.
- The shooter is either asking for his/her first round in the shooting (the perch is full of ducks) or his/her previous round in the current shooting was a good round.

When the shooter cannot ask for another round, the shooting is terminated and the success of the shooting is evaluated and, maybe, rewarded.

To increase the much beloved acoustic effect of shooting and to complicate matters a bit further an additional and unique feature is added to the shooting process. Each time when a shooter fires a good round, the system immediately reduces the number of ducks on the perch. All ducks which were not sitting between the two ducks shot in the round are shot down by an automatic gun. Sometimes, this feature drastically reduces the number of ducks on the perch and makes it impossible for the shooter to ask for more rounds, and thus effectively terminates the shooting.

The goal of the shooting is to fire maximum possible number of subsequent good rounds. Obviously, a shooter with an excellent aim is also in need of a good strategy which would allow him/her to prolong the shooting as much as possible by picking suitable pairs of ducks to be shot down in each good round.

Input Specification

There are more test cases. Each test case consists of two lines. The first line contains one integer N ($1 \leq N \leq 5000$) representing the number of ducks on the perch. The second line contains a sequence of N positive integers D_i ($1 \leq D_i \leq 10^4$), separated by spaces. Each value in the sequence represents the species of one duck on the perch in order from left to right. Same species are marked by the same values, different species are marked by different values.

Output Specification

For each test case, print a single line with one integer representing the maximum number of subsequent good rounds which can be fired at the ducks on the perch.

Sample Input

```
3
6 6 6
12
3 14 15 92 65 35 89 79 32 38 46 26
12
3 1 4 1 5 9 2 6 5 3 5 9
7
2 7 1 8 2 8 1
4
1 6 1 8
11
1 2 4 8 16 32 16 8 4 2 1
6
1 2 3 1 2 3
```

Output for Sample Input

```
1
0
2
2
1
5
1
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
Masaryk University

Czech Technical University in Prague

Technical University of Ostrava
University of Žilina
University of West Bohemia

CTU Open Contest 2017

Ice cream samples

`icecream.c`, `icecream.cpp`, `icecream.c11`, `Icecream.java`, `icecream.py`

To encourage visitors active movement among the attractions, a circular path with ice cream stands was built in the park some time ago. A discount system common for all stands was also introduced. When a customer buys ice cream at some stand, he is automatically granted a discount for one day at the next stand on the path. When visitors start at any stand and follow systematically the discount directions to the next stands, they eventually traverse the whole circular path and return back to the stand they started at.

Ice creams of various brands are sold at the stands. Additionally, each stand sells a nice sample box which contains small samples of popular ice cream brands. The number of samples in the box depends on the stand and various stands may put different brands into their sample boxes. Each box contains samples of one or more brands. A brand may be represented by one or more samples in the box, or it may be completely missing. Each stand sells only one type of sample box (the brands of the samples in the box are always the same for that particular stand).

Quido and Hugo are going to exploit the discount system for their own benefit. They decided to start at some stand and then continue in the direction of the discounts buying one ice cream sample box at each stand they visit in a consecutive sequence. Their goal is to collect at least one sample of each ice cream brand sold in the park. Simultaneously, to respect their stomach capacities, they want to minimize the total number of ice cream samples they buy.

Input Specification

There are more test cases. Each case starts with a line containing two integers N, K separated by space ($1 \leq N, K \leq 10^6$). N is the number of ice cream stands, K is the total number of different ice cream brands sold at all stands. The brands are labeled by numbers $1, 2, \ldots, K$. Next, there are N lines describing stands in their visiting order. Each such line contains the list of brands of all ice cream samples sold in the sample box at that particular stand. Each list starts with one positive integer L, describing its length, followed by L integers. Each list item represents the brand of one ice cream sample in the sample box sold at this stand. You may assume that even if a visitor buys one sample box at each stand, he/she will collect at most 10^7 ice cream samples.

Output Specification

For each test case, print a single line with one integer denoting the minimum number of ice cream samples Quido and Hugo have to buy in order to obtain a sample of each ice cream brand sold in the park. If it is impossible to obtain samples of all brands output -1.

Sample Input

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

Output for Sample Input

```
4
3
-1
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
Masaryk University

Czech Technical University in Prague

Technical University of Ostrava
University of Žilina
University of West Bohemia

CTU Open Contest 2017

Dark Ride with Monsters

`monsters.c`, `monsters.cpp`, `monsters.c11`, `Monsters.java`, `monsters.py`

A narrow gauge train drives the visitors through the sequence of chambers in the Dark Ride attraction. The chambers are occupied by IT monsters which are specially programmed to scare the visitors in various wicked ways. There is one monster in each chamber. For strange and obscure reasons, some of the monsters might have been installed in wrong chambers. The task of Freddie and Morcia, who themselves are employees and not monsters, is to reinstall the monsters in the correct chambers.

To avoid any additional confusion or threat, Freddie and Morcia work in episodes. In one episode, they choose two distinct chambers. Freddie picks the monster in one of the chambers and transports it to other chamber, while Morcia picks the monster in the other chamber and transports it to the chamber from which Freddie has just picked his monster. Thus, Freddie and Morcia effectively swap the monsters in two chambers during one episode. After some number of episodes, all monsters should be placed in correct chambers. Swapping monsters is a tedious task. Quite understandably, Freddie and Morcia want to minimize the number of episodes.

Input Specification

There are more test cases. Each test case consists of two lines. The first line contains one integer N ($1 \leq N \leq 2 \cdot 10^5$) specifying the number of chambers. The chambers are labeled $1, 2, \ldots, N$ and also the monsters are labeled $1, 2, \ldots, N$. The label of each chamber should coincide with the label of a monster correctly installed in it. The second line contains N labels of the monsters which are currently installed in the chambers. The monster denoted by the first label on the line is installed in the chamber 1, the monster denoted by the second label on the line is installed in the chamber 2, etc.

Output Specification

For each test case, print a single line with one integer denoting the minimum number of episodes required to install the monsters in correct chambers.

Sample Input

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

Output for Sample Input

```
0
2
3
3
2
9
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
Masaryk University

Czech Technical University in Prague

Technical University of Ostrava
University of Žilina
University of West Bohemia

CTU Open Contest 2017

Go Northwest!

`northwest.c`, `northwest.cpp`, `northwest.c11`, `Northwest.java`, `northwest.py`

Go Northwest! is a game usually played in the park main hall when occasional rainy weather discourages the visitors from enjoying outdoor attractions.

The game is played by a pair of players and it is based entirely on luck, the players can hardly influence its outcome.

The game plan consists of one large map on the wall with N distinct towns displayed on it. Before the start of the game, the leader of the game hands a bowl filled with N identical balls to each player. Inside each ball, there is a reference to some town on the map. Each bowl contains references to all towns on the map.

When the game starts, one of the players randomly draws one ball from his/her bowl, opens it and reveals the town inside. Next, the other player randomly draws one ball form his/her bowl, opens it and also reveals the town inside.

If both towns are the same, the game ends in a draw. If the towns are not the same, the leader of the game highlights the towns on the map. If one of the towns lies exactly Northwest to the other, the first player wins the game. If one of the towns lies exactly Northeast to the other, the second player wins the game. In all other cases, the game ends in a draw. Two towns are considered to lie exactly Northwest or Northeast from each other, if the angle between the line connecting the towns and the horizontal axis is exactly 45 degrees.

It is clear that the chance of winning the game depends substantially on the layout of the towns on the map. All town coordinates are known in advance. You are asked to find the probability that there is a winner in the game.

Input Specification

There are more test cases. Each case starts with a line containing one integer N ($1 \leq N \leq 10^5$) specifying the number of towns on the map. Next, there are N lines each of which contains two integers x and y separated by space and specifying the coordinates of one town on the map. The coordinates are standard Cartesian coordinates in the plane. The absolute value of any coordinate does not exceed 10^9.

Output Specification

For each test case, print a single line with one floating point number P denoting the probability that there is a winner in the game. P should be printed with the maximum allowed error of 10^{-6}.

Sample Input

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

Output for Sample Input

```
0.25
0.666667
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
Masaryk University

Czech Technical University in Prague

Technical University of Ostrava
University of Žilina
University of West Bohemia

CTU Open Contest 2017

Punching Power

`power.c`, `power.cpp`, `power.c11`, `Power.java`, `power.py`

The park management finally decided to install some popular boxing machines at various strategic places in the park. In fact, to compensate for the previous lack of machines, they decided to install as many machines as possible. Surprisingly enough, the park is not going to be choked with new machines because there are some quite serious legal limitations regarding the locations of the machines. The management has marked all possible boxing machine locations and their respective coordinates on the park plan. Additionally, they also have to respect manufacturer security rule: The distance between any two boxing machines has to be at least 1.3 meters.

Help the management to establish the maximum possible number of boxing machines which can be installed in the park.

Input Specification

There are several test cases. Each case starts with a line containing one integer N which specifies the number of possible boxing machine locations in the park ($1 \leq N \leq 2000$). Next, there are N lines representing the location coordinates, each line describes one location by a pair of integer coordinates in meters. All locations in one test case are unique. Each coordinate is non-negative and less than or equal to 10^9.

You are guaranteed that all locations form a single connected group, that is, it is possible to start in any location and reach any other location by a sequence of steps, each of which changes exactly one coordinate by 1, without leaving the area suitable for placing boxing machines.

Output Specification

For each test case, print a single line with one integer representing the maximum number of boxing machines which can be installed in the park.

Sample Input

```
4
0 0
0 1
1 0
1 1
6
0 1
1 0
1 1
1 2
1 3
2 2
```

Output for Sample Input

```
2
4
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
Masaryk University

Czech Technical University in Prague

Technical University of Ostrava
University of Žilina
University of West Bohemia

CTU Open Contest 2017

Treetop Walkway

`walkway.c`, `walkway.cpp`, `walkway.c11`, `Walkway.java`, `walkway.py`

The amusement park is a real landscape park which means that there are significant portions of forest standing among the attractions. A huge treetop walkway allows the visitors to explore the unusual world of tree canopies and enjoy spectacular views of hills and lakes in the region.

The walkway consists of wooden platforms installed close to the treetops in different heights above the ground and connected by narrow wooden paths. Each path connects exactly two platforms. Number of paths connected to one platform may vary. Each platform can be reached from any other platform without leaving the walkway.

To improve the safety standards and to alleviate occasional bottleneck problems in some parts of the walkway, the management decided to make all paths strictly one-way for the visitors. After each path was assigned a direction, someone found out that a connection problem appeared. It turned out that there may be platforms which would not be accessible from some other platforms if the visitors followed only prescribed path directions and did not leave the walkway. As the direction choice was made by the management on the basis of various "aesthetical and functional considerations", changing the already-assigned directions is not an option.

To fix the problem, it was decided to build additional one-way paths. Those will connect selected platforms in such manner that any platform will be accessible from any other platform without leaving the walkway. All other properties of the walkway will remain unchanged. They are not going to build any new platform.

They want to minimize the number of new paths by careful selection of the platforms to be connected and also by correct direction settings of the new paths.

Input Specification

There are more test cases. Each case starts with a line containing two integers N, M separated by space ($1 \leq N \leq 10^5, 0 \leq M \leq 2 \cdot 10^5$). N is the number of treetop platforms, M is number of current treetop paths. The platforms are labeled by integers $1, 2, \ldots, N$. Next, there are M lines, each of them specifies one treetop path and its direction. The line contains the labels of two different platforms separated by space, the planned direction of the path is from the first platform on the line to the second one. All paths in the input are unique. In the final walkway arrangement, there will be at most two paths connecting any two given platforms A and B, one in the direction $A \to B$ and another in the direction $B \to A$.

Output Specification

For each test case, first print a single line with a single integer R specifying the minimum number of additional treetop paths which ensure the reachability of any platform from any other platform. On the next R lines, print all the new paths in the same format as in the input. If there are more solutions of the problem, any of them will be accepted.

Sample Input

```
5 6
1 2
2 3
3 1
2 4
4 5
5 4
4 3
2 1
3 1
4 1
6 5
1 4
1 5
1 6
2 4
3 4
```

Output for Sample Input

```
1
4 1
3
4 2
3 4
1 3
3
4 1
6 2
5 3
```