2018 ICPC Mid-Atlantic North America Programming Contest

Coachs' Handout

icpc regionals 2018

advancing the art and the sport of competitive programming

icpc.foundation

This is a courtesy copy of the problem set for the Mid-Atlantic Regional contest.

Important!!

The creation of this problem set was a collaboration among multiple Regional Contests scheduled for November 10. These contests will be starting at different times and taking place in different time zones.

In deference to the later-starting contests, please refrain from emailing, blogging, posting, tweeting, or otherwise making any public comments about this problem set until November 11.

Nov. 10, 2018

And now, a word from the head judge

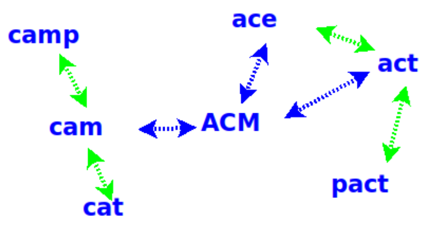
Without a team of creative volunteers, there would be no problem set for this contest each year. It's not too early to start thinking about the 2019 contest!

We do require that problem authors not be coaches of or otherwise directly associated with a participating team. But if you are a team coach and have a colleague who might be interested in contributing, please drop a word to them. If you have a creative team member who will be an alumnus or alumna next year, you might do the same.

Anyone interested in participating as a member of the authoring team for 2019 can send their contact information to this year's head judge for the region, Steven Zeil, at `zeil@cs.odu.edu`.

Problem	Problem Name	Balloon Color
A	Lost is Close to Lose	Orange
B	Orphan Backups	Green
C	Collusion on Two Wheels	Silver
D	Find Poly	Pink
E	Neutral Ground	Red
F	Picking Up the Dice	Yellow
G	Playing the Slots	Black
H	Tightly Packed	Purple

Problem A: Lost is Close to Lose



Better Documents Inc. is contemplating the next generation of word processors. Now, nearly every word processor includes a Spell Checker. BDI, however, is looking forward to replacing that with a true Typo Checker. We've all been caught, after relying on a spell checker, by typing mistakes ("typos") that just happen to wind up as a correctly spelled word, just not the word we intended. BDI hopes to use AI to determine when a word's context suggests that it is out of place and probably should have been a different, but similarly spelled, word.

As a first step in this process, they want to see how common such similar words really are in ordinary text. Write a program to read in paragraphs of text and to produce a list of similarly spelled words occurring in that text.

For the purpose of this program, a *word* is any maximal string of non-whitespace characters containing at least one alphabetic character. *Whitespace* can be either blanks or line terminators ("`\r`" or "`\n`"). The *core* of a word is what you have left after removing any non-alphabetic characters and replacing any upper-case alphabetic characters to their lower-case equivalents.

Two words are considered to be *similarly spelled* if the core of one word can be converted to the core of the other word by a single application of any one of the following transformations:

- Delete a single character.
- Insert a single alphabetic character.
- Replace a single character by a different alphabetic character.
- Transpose (exchange) any two adjacent characters.

Input

Input consists of 1 to 100 lines of text, followed by an end of input marker in the form of a line containing only the string "***".

Each line of text will contain 0 to 80 ASCII characters (not counting line terminators).

Output

For each word core in the text that has 1 or more similarly spelled words, print a line consisting of

1. That word core
2. A colon (":") followed by a blank
3. A list of all similarly spelled word cores (with no duplicates and not containing the core to the left of the colons), in alphabetic order, separated by single spaces.

The lines printed should be in alphabetic order of the word cores to the left of the colon.

If there are no similarly spelled words in the input, print a single line containing the string "***".

Examples

Example 1

Sample Input

```
Lost is Close to Lose

"Better Documents Inc. wants to add Typo Checking in to the
next generation of word processors," he said.
***
```

Sample Output

```
close: lose
he: the
in: inc is
inc: in
is: in
lose: close lost
lost: lose
the: he
```

Example 2

Sample Input

```
The fox said, "When?"
"Not till 12 o’clock", replied the hen.
"That clock is stopped, it will never strike.", he said.
***
```

Sample Output

```
clock: oclock
he: hen the
hen: he when
is: it
it: is
oclock: clock
the: he
till: will
when: hen
will: till
```

Example 3

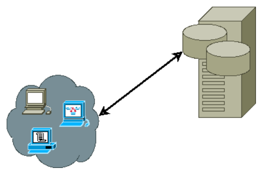
Sample Input

```
There are no similar words
in this input set.
***
```

Sample Output

```
***
```

Problem B: Orphan Backups



GigantoCorp has a problem. They have removed all tape drives from their data center and are doing all of their backups to disk storage. However, they seem to have used more disk space than their backup software indicates should be in use. They think that there are files on the backup storage that are not in the backup software index. They would like your team to write a program to determine if there are“orphan” files on the backup storage that are not in the index or “orphan” index entries that have no corresponding files on the backup storage.

Your program will be provided with two lists – an index of valid backup image names, followed by a list of backup file names.

Your program is to compare the two lists and determine if there are either backup images in the index that have with no files or files that are not associated with any backup image in the index.

Input

Input is structured in two blocks.

The first is a list of backup images, one per line. Each backup image name consists of 1 to 32 printable ASCII characters (no spaces). There may be up to 100 000 images. Each image name is unique, and they may occur in any order.

The end of this block is signaled by an empty line.

The second block contains a list of backup file names, one per line, left justified. A backup file name has the format

imageName_integer_type

where *imageName* is a backup image name in the format described above, *integer* is an integer in the range $0 \ldots 2\,000\,000\,000$ denoting the time in seconds since January 1, 1970 at 00:00 GMT (the “UNIX epoch”), and *type* is an uppercase alphabetic character string of 1 to 6 characters. There may be up to 300 000 backup file names. File names are unique, and they may occur in any order.

The end of the second block is signaled by the end of file.

Output

First, print the list of files with no index entries, one per line, by printing the character “F”, a single space, and then the file name.

After all the orphan files are printed, print the orphan index entries one per line by printing the character “I”, a single space, and then the backup image name.

Entries in each list are to be printed in ASCII lexicographic order.

If every file has an index entry and every index entry has one or more files, print only a line containing the string “No mismatches.”.

Example

Example 1

Sample Input

```
payroll.xls
projects.doc
employees.dat
products.txt

payroll.xls_1539199053_INCR
employees.dat_1539199053_INCR
payroll.xls_1539112653_INCR
employees.dat_1539112653_INCR
payroll.xls_1539026253_FULL
employees.dat_1539026253_FULL
customers.dat_1539026253_FULL
```

Sample Output

```
F customers.dat_1539026253_FULL
I products.txt
I projects.doc
```

Problem C: Collusion on Two Wheels

Two bicycle courier services have been competing in Metro City for many years, stealing customers back and forth from one another. Recently, they have come to realize that they would be better off if they could attract new customers instead. A problem is that, over time, each company's customers have become so widely scattered across the city that some of their delivery times are unacceptably slow, leading to customer dissatisfaction and a poor reputation.

The companies would like to divide up their current customers so that each company could run ads saying "We guarantee delivery in no more than M minutes", hoping to attract new customers. The streets of Metro City are laid out on a grid, with buildings at integer coordinates of the grid. The couriers must travel along those roads – they cannot cut through buildings. It takes one minute to travel one unit of distance in the x or y direction.

Divide up the current customer base to minimize the longest delivery time required by either company to have a courier travel from one customer of that company to any other customer of the same company.

* A delivery is considered to have been completed when the courier reaches the (x,y) address of the customer. No time is counted for wandering the hallways of the customer's building.

* It is acceptable that a courier traveling from one customer to another will pass by customers of the same or of the other company. No extra time delay is accrued for riding past a customer.

* If it should happen that one company winds up with only a single customer, that company puts someone on site to deliver messages within that one building. This is considered a zero delivery time.

Input

Input starts with 1 line containing an integer N, the number of customers to be divided up. $2 < N \leq 1\,000$.

This is followed by N lines, each containing a pair of integers x and y denoting the position of one customer. $0 \leq x, y \leq 1\,000$.

Output

Print a single line containing the longest delivery time required by the two companies (i.e., the maximum of the two longest delivery times offered by each company separately).

Examples

Example 1

Sample Input

```
6
1 1
4 1
1 5
10 10
10 8
7 10
```

Sample Output

```
7
```

Example 2

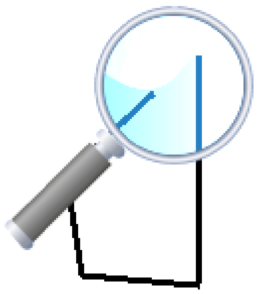
Sample Input

```
7
0 0
100 100
0 100
100 0
50 50
0 50
100 50
```

Sample Output

```
100
```

Problem D: Find Poly



Consider a set of line segments in a 2D plane.

For any set of line segments L, define $P(L)$ as the set of all endpoints of the line segments in L.

Two line segments are said to be *connected* if they share an endpoint.

Given a set of line segments U, we say that a *geometric figure* ("figure" for short) is a set S of one or more line segments, $S \subseteq U$, for which

1. for any two points p_1 and p_2 in $P(S)$, we can reach p_2 from p_1 by tracing along one or more connected line segments, and
2. for every line segment e in S, all line segments of U that are connected to e are also in S.

A *polygon* is a geometric figure S for which it is possible to trace a path starting from some endpoint p and ending at p using every line segment in S exactly once and visiting each point in $P(S)$ other than p, exactly once, visiting p only at the beginning and end of the path.

See figure 1 which has 10 figures of which a, b, e, and f are polygons. The dots are the end points of the line segments.

Note that b is self-intersecting but that the intersection is not at the end points of the intersecting line segments. Similarly c and d as well as e and f intersect but are not connected.

Your task is to count the total number of figures and identify how many are polygons.

Input

The input is a series of lines terminated by end-of-file. Each line will have one or more line segments of the form:

```
(x1,y1),(x2,y2);
```

where `(x1,y1)` is one end point and `(x2,y2)` is the other end point. $0 \leq x1, y1, x2, y2 \leq 99$. All coordinates are integers.

The separator characters, "(),;", may be preceded or followed by white space. A line may be at most 100 characters long.

There will be at most 200 line segments. A given line segment will only appear once in the input and none will be of length 0.

Line segments are not directed so the order of the end points in the line segment is not significant. The order of the line segments in the input is also not significant.

Output

Print a single line containing two integers separated by a single space. The first number should be the total number of figures and the second should be the number of polygons found.

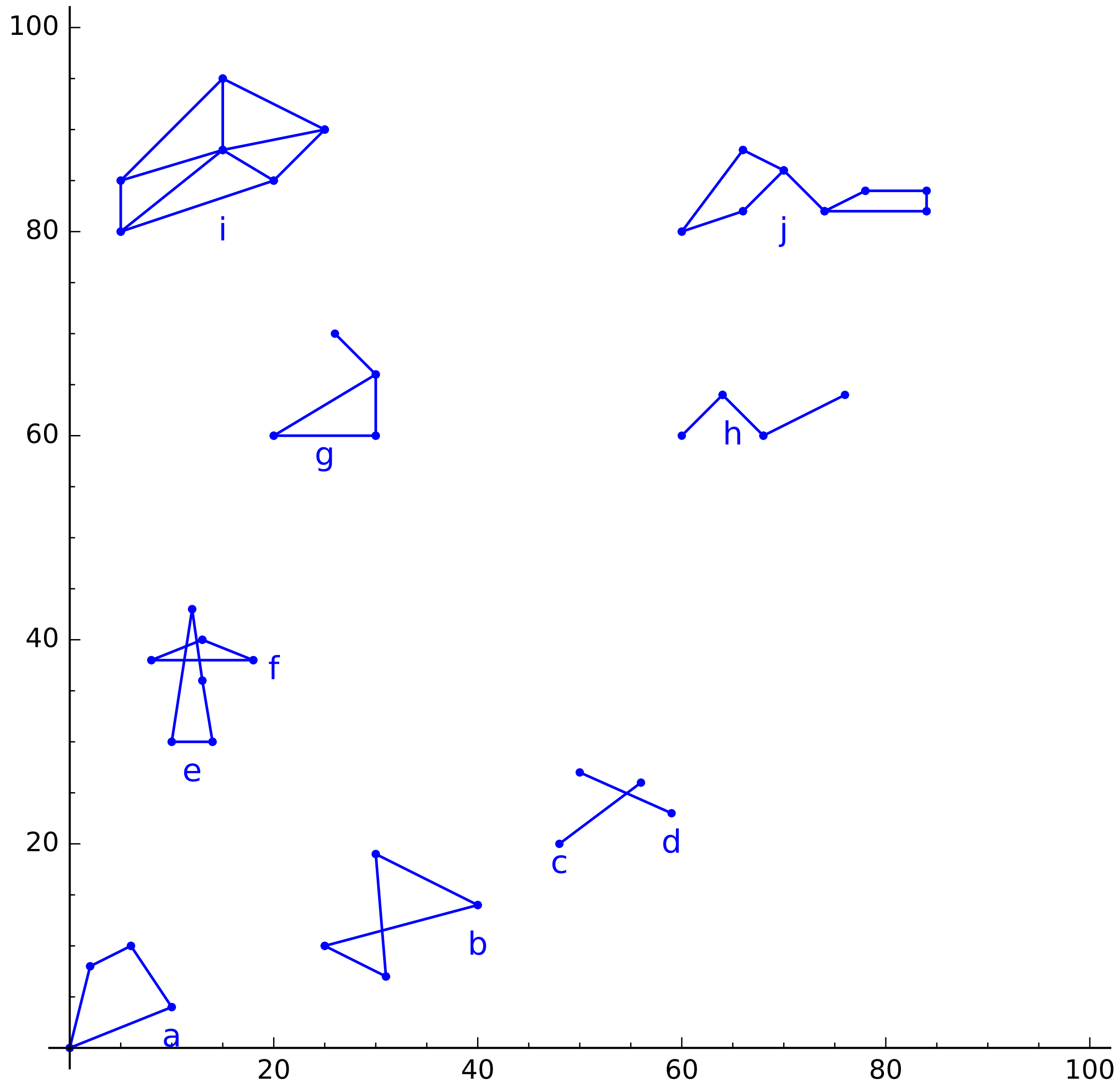


Figure 1: 10 figures, 4 polygons

Examples

In the first example below, the points correspond to the picture in Figure 1.

Example 1

Sample Input

```
(84,84),(78,84);(68,60),(64,64);(20,85),(15,88);(0,0),(2,8);
(30,60),(30,66);(13,40),(18,38);(15,88),(15,95);(18,38),(8,38);
(31,7),(25,10);(30,66),(26,70);(40,14),(30,19);(5,85),(15,88);
(48,20),(56,26);(84,84),(84,82);(66,82),(70,86);(15,95),(25,90);
(70,86),(66,88);(59,23),(50,27);(15,88),(5,80);(78,84),(74,82);
(60,80),(66,82);(5,85),(5,80);(25,10),(40,14);(20,85),(25,90);
(20,60),(30,66);(13,36),(14,30);(30,60),(20,60);(64,64),(60,60);
(31,7),(30,19);(15,88),(25,90);(68,60),(76,64);(8,38),(13,40);
(5,85),(15,95);(0,0),(10,4);(10,30),(14,30);(74,82),(70,86);
(10,30),(12,43);(6,10),(10,4);(5,80),(20,85);(6,10),(2,8);
(60,80),(66,88);(84,82),(74,82);(12,43),(13,36);
```

Sample Output

```
10 4
```

Example 2

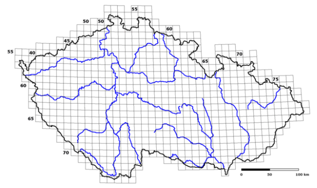
Sample Input

```
(45,26),(88,34);(39,6),(67,8);(73,52),(92,38);(63,35),(18,61);
(34,23),(46,10);(2,75),(86,47);(26,18),(95,36);(59,78),(49,95);
(63,95),(67,80);(23,12),(33,46);(33,1),(46,10);(63,78),(2,75);
(2,33),(11,31);(99,98),(18,5);(88,34),(49,95);(25,43),(46,10);
(66,1),(2,75);(17,59),(2,33);(85,7),(85,59);(51,56),(63,95);
(1,48),(46,7);(66,49),(57,84);(59,78),(63,35);(0,49),(69,83);
(75,82),(51,56);(67,8),(92,38);(25,43),(86,47);(54,33),(12,42);
(87,12),(57,84);(37,92),(84,90);(18,61),(12,42);(66,49),(95,36);
(85,7),(73,52);(1,48),(99,98);(53,26),(11,31);(34,23),(86,47);
(22,91),(55,59);(23,12),(75,6);(66,1),(33,1);(33,46),(97,41);
(87,12),(66,49);(92,38),(8,19);(54,33),(97,41);(45,26),(7,53);
(39,6),(85,59);(63,78),(34,23);(76,9),(18,5);(67,80),(17,59);
(25,43),(66,1);(75,82),(53,26);(0,49),(18,61);(12,42),(19,3);
(33,1),(63,78);(76,9),(46,7);(8,19),(84,90);(8,19),(37,92);
```

Sample Output

```
7 2
```

Problem E: Neutral Ground



Two kingdoms had been at war for a long time, until the emperor intervened to bring an end to the conflict. The territory in question comprises an M by N rectangular grid. At the emperor's insistence, the two kings have withdrawn their troops until no two opposing troops are in adjacent squares of the map (adjacent being horizontal or vertical – diagonal is not considered).

The emperor proposes to designate certain squares of the map as neutral territory. Neither king will be allowed to move troops into those squares, and the emperor's own forces will patrol them to be sure that both kings observe these rules.

The emperor is frugal and does not want to commit more soldiers to this effort than absolutely necessary. His generals have marked each square of the map with the number of soldiers required to secure that square. What remains is to choose which of those squares should be patrolled.

Write a program to determine the minimum number of soldiers that the emperor will need to be deploy to guarantee that the troops of one kingdom cannot move, in one or more steps, into squares occupied by the troops of the second kingdom (moving horizontally or vertically) without encountering the emperor's own soldiers.

Input

Input begins with a line containing 2 integers, w and h, denoting the width and height of the map. $1 \leq w, h \leq 40$.

This is followed by h lines. Each line contains w characters, left justified. These characters will be 'A' or 'B', designating a position held by king A or king B, or a single numeric digit, designating a currently unoccupied position that can be secured by the use of that number of soldiers. For example, a '2' would indicate that two soldiers must be deployed to that square to secure it against passage of other troops. A '0' indicates terrain that is impassible – the emperor need not commit soldiers there because the kingdom troops cannot pass through that square.

No 'A' will be adjacent, horizontally or vertically, to any 'B'.

There will be at least one 'A' and one 'B' in the input.

Output

Print a single line containing an integer denoting the minimum number of soldiers that the emperor must deploy to guarantee that there is no open path between any 'A' position and any 'B' position, using any combination of horizontal or vertical moves.

Examples

Example 1
Sample Input

```
8 5
A11111AA
AA7B111A
111BB111
11BBB111
11BBB11B
```

Sample Output

```
13
```

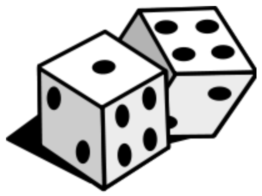
Example 2
Sample Input

```
25 6
A211111321231111111111111
2A211111000111111BB11111
AA2111110001111111BBBB1111
A2111114111411111BBBB1111
AA2111110001111111BB11111
AA2111111011111111111111
```

Sample Output

```
2
```

Problem F: Picking Up the Dice



Two players are playing a game with a set of K six-sided dice. One player calls out a number in the range $K \ldots 6K$ and the other tries to roll that number. After the first roll, the player is allowed to pick up any number $(0 \ldots K)$ of dice and re-roll them.

Given the number of dice, the target number the player wants to roll, and the set of numbers the player obtained on the first roll, what number of dice should the player pick up to maximize their chances of getting the target number on the second roll?

Input

Input begins with a line containing 2 integers, K, the number of dice, and T, the target number. $2 \leq K \leq 24$, $K \leq T \leq 6K$.

The next line contains K integers, indicating the numbers that were rolled on each of the dice on the first roll. All will be integers in the range $1 \ldots 6$.

Output

Print a single line containing an integer denoting the number of dice that the roller should pick up and re-roll in order to maximize the chances of getting an overall sum of T. (The roller will be able to choose which dice to pick up, but you are only asked to print the number of dice, not which ones.)

If there are more than one numbers of dice that can be picked up to achieve the same probability of getting to T, print the smallest such value.

Examples

Example 1
Sample Input

```
3 9
5 4 1
```

Sample Output

```
1
```

Example 2
Sample Input

```
4 13
2 2 2 2
```

Sample Output

```
3
```

Example 3

Sample Input

```
18 90
1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6
```

Sample Output

```
12
```

Example 4

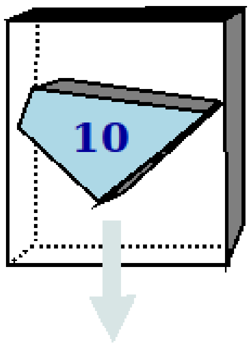
Sample Input

```
6 21
1 2 3 4 5 6
```

Sample Output

```
0
```

Problem G: Playing the Slots



The small nation of Erratica prides itself on defying conventions established by the more "boring" countries around the world. One of their more obvious distinctions lies in the design of their coinage. Believing that a person should be easily able to identify the value a coin in a pocket or change purse by touch alone, Erratica designs its coins as polygons. For stability, the coins are convex – there are no notches cut into the coins. But the overall shapes themselves can be quite irregular.

Erratica Vending, the only manufacturer of vending machines in the nation, has been sent the design for a new coin to be issued. Their machines are designed so that coins enter through a slot into channel shaped as a rectangular prism with the slot being an open face of the prism. The channel is narrow because the coins are thin, but long enough to contain the entire coin. From the outside of the machine, the slot appears as a rectangular hole.

The company wants to know what would be the smallest slot size they will need so that the new coin can be slipped, after some rotation, into the slot.

Input

Input begins with a line containing N, the number of sides to the polygonal coin. $3 \leq N \leq 20$.

This is followed by N lines, each containing two real numbers x and y, the coordinates of a vertex of the polygon. $0.0 \leq x, y \leq 100.0$

All N vertices will be distinct, and the vertices will be presented in an order proceeding clockwise around the perimeter of the coin.

Output

Print a single line with a real number, to two decimal places precision, denoting the minimum slot size allowing the coin to pass through. Outputs will be accepted that are within ± 0.01 of the judges' answer.

Examples

Example 1

Sample Input

```
3
0 0
0.71 3.54
4.21 4.21
```

Sample Output

```
2.00
```

Example 2

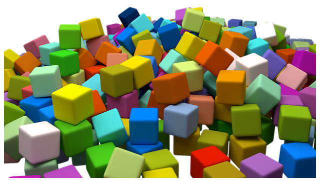
Sample Input

```
6
10 12.5
10 17.5
15 20
20 17.5
20 12.5
15 10
```

Sample Output

```
8.94
```

Problem H: Tightly Packed



Consider packing widgets for shipping where widgets cannot be stacked upon each other (2D packing). Each widget has a 1x1 footprint and is 1 unit high.

Boxes are available in any W by H by 1 size such that $H/2 \leq W \leq 2H$, with W and H being integers. The company wants to minimize the amount of packing material that will be needed to fill empty squares in a box.

Given N, the number of widgets to be shipped, what is the smallest number of squares that will be left empty when those widgets are packed for shipping?

Input

Input consists of one line containing an integer N, the number of widgets to be packed. $1 \leq N \leq 10^{16}$.

Output

Print a single line containing an integer denoting the minimum number of empty squares.

Examples

Example 1
Sample Input

```
47
```

Sample Output

```
1
```

Example 2
Sample Input

```
523
```

Sample Output

```
2
```

Example 3
Sample Input

```
10000000000001
```

Sample Output

```
6
```