

Problem Set 2

Tasks that are marked with * are optional for groups with only two members and will not count into their score.

1 Spaghetti Code

You have been hired by a software company to maintain an ancient piece of spaghetti code until it is no longer needed. The code is highly convoluted and barely maintainable, with new bugs emerging every week. Your task is to fix them. At the beginning of each week, you have two options: either spend a small amount of time fixing the bugs or fully refactor the code to make the time required for future bug fixes negligible. You must make this decision without knowing when your employer will suddenly decide that your code is obsolete and replace it.

Note: This is an interactive problem

Input: The first line consists of two integers: a (the time required to fix bugs in an unrefactored code per week) and b (the time needed to fully refactor the code). Maintaining refactored code takes time 0. After this, the server will either output ‘continue’ (indicating another week of maintenance) or ‘stop’ (indicating the code has become obsolete)

Output: After each ‘continue’ you must output one of the following three actions:

- ‘FIX’ (fix the code without refactoring)
- ‘REFACTOR’ (refactor the code)
- ‘NOTHING’ (you refactored the code and no longer need to spend time on it)

If the server outputs ‘stop’, print the total amount of time spent on your actions. The total time has to match with your actions and is only allowed to be at most **twice** the amount of the optimal offline solution to the problem.

In the following example the server output is *cursive*. The server asks for 6 weeks of runtime and the optimal offline solution would be 5 - refactoring code in the first week. Since $7 \leq 2 \cdot 5$, this solution is valid.

Server Output	Your Output
1 5	
<i>continue</i>	FIX
<i>continue</i>	FIX
<i>continue</i>	REFACTOR
<i>continue</i>	NOTHING
<i>continue</i>	NOTHING
<i>continue</i>	NOTHING
<i>stop</i>	7

2 Road Maintenance*

Task: You start your new job as manager of the road maintenance depot of a district in Manhattan. Your district has the shape of a polygon, and the road system has the shape of a perfect grid (so the Broadway is located in a different district). To get an overview you want to count all the crossroads that you will be responsible for in the future.

Input: The input starts with a line containing the number m of corners of the polygon. Then, m lines follow; where the i -th line contains the x - and y -coordinate of the i -th corner. The coordinates are all integers.

You may assume that the polygon is non-self-intersecting, but it is possible that more than 2 corners are located on a line.

Output: Output the number of crossroads that are located in your district. You are also responsible for the crossroads that lie on the boundary.

Sample Input:

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

Sample Output:

```
10
```

3 k -means I

Task: A crucial part of the implementation of many clustering algorithms is finding the closest center to a point. In this task, we consider a special case of this problem where the points and centers are 1-dimensional, i.e., $P \subseteq \mathbb{R}$ and $C \subseteq \mathbb{R}$. Your task is to solve the following problem: For each query point $x \in \mathbb{R}$, decide whether the closest point in C is at distance at most R from x . If so, report the point closest to x .

Input: Each input file starts with a line with two integers and a floating point number. The first number c (integer) is the number of centers, the second number q (integer) is the number of queries, and the third number is R (floating point number with two positions after the point). In the following c lines, each line has a floating point number (two positions after the point) representing a center. The subsequent q lines contain a floating point number (two positions after the point) each, representing a query point.

Output: For each of the q query points, write a line with **none in range** if the closest point is at distance $> R$, and otherwise, write a line with the closest center (with two positions after the point). If there are two closest centers $c_1, c_2 \in \mathbb{R}$, choose the smaller one.

Sample Input:

```
2 3 1.00
0.00
1.00
1.10
0.50
10.00
```

Sample Output:

```
1.00
0.00
none in range
```

4 k -means II

Task: Now the task is to solve a more realistic version of the last task. The points now live in \mathbb{R}^d with the Euclidean metric, and in addition, the task is dynamic: Instead of getting all centers upfront, we have to produce centers on the fly. More precisely, do the following: Read a stream of points. Compute a set of centers C ; in the beginning, C is empty. For each query point $x \in \mathbb{R}^d$, decide whether the closest point in C is at distance at most R from x . If so, ignore x . If not, add x to C .

Input: The first line contains two numbers. The first number $d \leq 100$ is an integer, the dimension of the points. The second number is a floating point number with three positions after the point, it is the threshold value R . After that, a (not previously specified) number of lines follows, each containing a point, given by d numbers separated by spaces. The numbers are floating points with three positions after the point.

Output: Output a single integer: The number of points that are added to the center set.

Sample Input:

```
2 1.000
0.000 0.000
1.000 1.000
0.500 0.500
2.000 1.000
2.500 0.500
```

Sample Output:

```
3
```

5 Emergency Response

You are in charge of the emergency response team for a big fair and need to make sure that fire extinguishers are evenly distributed along the exhibition site. Since the number of fire extinguishers at your disposal is limited, you need to make a selection of booths that get equipped. You want to ensure that the maximum response time (i.e. the furthest distance of any booth to the nearest fire extinguisher) is minimal, so any fire can be dealt with fairly quickly. Since the event is starting soon and there is not enough time to calculate the optimal distribution, you are content with finding a solution within a factor 2 of the optimum.

Input: The first line of the input contains the number n of booths and the number k of fire extinguishers. Then, n lines follow, each containing two integers that describe the x and y location of each booth

Output: Compute a number that is at least d^* and at most $2d^*$, with d^* being the maximum distance of the optimal solution. To avoid rounding problems, **output the square of the number**, i.e., output X^2 with $d^* \leq X \leq 2d^*$.

Sample Input:

```
5 2
0 0
0 3
1 3
6 6
6 7
```

Sample Output:

```
9
```

In this example, 3 is the optimum value. Thus, outputting 9 is allowed and optimal. However, solutions up to 6 would be permitted, i.e., outputs up to $6^2 = 36$ are accepted.

6 Rectangular Fields

Farmer William has an old, rectangular orchard of pear trees. Originally, it was completely filled with trees, but over the years many trees had to be felled. Now, William wants to use the free space to keep sheep. He has the brilliant idea to use the pear trees as corners for the pasture's fence. He wants to change the sheep's pasture every day. In addition, every day's pasture should be rectangular. How many days can William go without repeating any one pasture?

Remark: The organizer is not aware of a solution in `Python` that adheres to the time limit. It is recommended to solve the problem in `C++` or `Java`.

Input: The first line of the input has the number r of rows of the orchard. Every following line consists of a string in $\{0,1\}^r$. The position of pear trees are marked with 1. You can assume that r is bounded from above by 2000.

Output: The number of subgrids (axis-parallel rectangles) such that all corners are marked with a 1.

Sample Input:

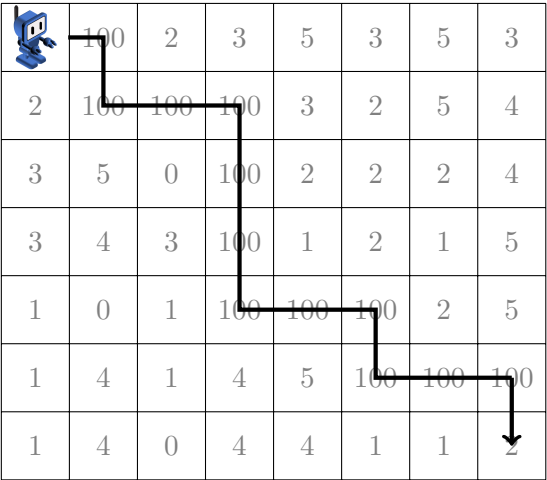
```
3
111
110
011
```



Sample Output:

2

7 Treasure Hunt*

In your new favorite mobile game, you are controlling a little robot that collects gem stones on a rectangular field with n rows each containing m squares (the values of n and m depend on the level that you have reached). The robot starts in the upper left corner at position $(1,1)$ and has to end at position (n,m) . You may only move him down or to the right in each step, i.e., if we denote the robots position in row i , column j by (i,j) , you may move him to $(i+1,j)$ or to $(i,j+1)$, respectively. At each position that the robot reaches, he collects all gem stones that lie at that position. Your goal is to collect as many gem stones as possible.



	100	2	3	5	3	5	3
2	100	100	100	3	2	5	4
3	5	0	100	2	2	2	4
3	4	3	100	1	2	1	5
1	0	1	100	100	100	2	5
1	4	1	4	5	100	100	100
1	4	0	4	4	1	1	

Input: The first line contains the integer $n \in \{1, \dots, 1000\}$, the second line contains the integer $m \in \{1, \dots, 1000\}$. The following n lines contain m non-negative integers each. In the i th of these lines, the j th number is the number of gems at position (i,j) .

Output: An integer, the maximum number of gem stones that may be collected.

Sample Input:

```
3
4
1 12 22 5
0 0 6 0
1 8 0 1
```

Sample Output:

```
42
```

8 Pizzeria Advertisement

You want to advertise the grand opening of your pizzeria in your home town. In order to reach the largest amount of people, you want to place an advertisement poster at intersections in such a way that everyone traveling within the city passes at least one poster. Here, traveling means starting at an intersection and driving to a different intersection. However, the poster printing is expensive, and thus you ideally want to minimize the amount of posters required. Unfortunately, there is not much time left until the opening of your pizzeria and a friend that studies computer science told you that your problem will likely take a very long time to solve exactly. So you accept the next best thing and are happy if you print at most double the minimum number of posters required.

Input: Your home town is given as a graph where roads are represented by edges and intersections are represented by nodes. The first line of the input contains two numbers, the number of nodes n and the number of edges m . This is followed by m lines containing two integers $a, b \in \{0, \dots, n-1\}$. Each such pair describes an edge in the graph.

Output: The number of posters required so that it is impossible to travel along any path (of at least one edge) in the graph without seeing a poster. This number does not have to be the minimal number required, but it may not deviate by factor of more than 2 from this number.

Sample Input:

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

Sample Output:

```
2
```

While 2 is the optimum solution, values up to 4 would be fine as well.