Colombian Collegiate Programming League
CCPL 2013

Contest 7 -- July 6

Search, Sorting, and Graphs

Problems

This set contains 8 problems; pages 1 to 15.

(Borrowed from several sources online.)

Page

A - Stammering Aliens 1
B - Beehives 3
C - Cucumber Market 5
D - Distributing Ballot Boxes 6
E - Tagalog Dictionary 8
F - Peer Review 9
G - Texas Summers 11
H - Haunted Graveyard 13

Official site `http://programmingleague.org`

Official Twitter account @CCPL2003

A - Stammering Aliens

Source file name: `aliens.c, aliens.cpp` *or* `aliens.java`

Dr. Ellie Arroway has established contact with an extraterrestrial civilization. However, all efforts to decode their messages have failed so far because, as luck would have it, they have stumbled upon a race of stuttering aliens! Her team has found out that, in every long enough message, the most important words appear repeated a certain number of times as a sequence of consecutive characters, even in the middle of other words. Furthermore, sometimes they use contractions in an obscure manner. For example, if they need to say *bab* twice, they might just send the message *babab*, which has been abbreviated because the second *b* of the first word can be reused as the first *b* of the second one.

Thus, the message contains possibly overlapping repetitions of the same words over and over again. As a result, Ellie turns to you, S.R. Hadden, for help in identifying the gist of the message.

Given an integer m, and a string s, representing the message, your task is to find the longest substring of s that appears at least m times. For example, in the message *baaaababababbababbab*, the length-5 word *babab* is contained 3 times, namely at positions 5, 7 and 12 (where indices start at zero). No substring appearing 3 or more times is longer (see the first example from the sample input). On the other hand, no substring appears 11 times or more (see example 2).

In case there are several solutions, the substring with the rightmost occurrence is preferred (see example 3).

Input

The input contains several test cases. Each test case consists of a line with an integer m ($m \geq 1$), the minimum number of repetitions, followed by a line containing a string s of length between m and 40000, inclusive. All characters in s are lowercase characters from 'a' to 'z'. The last test case is denoted by $m = 0$ and must not be processed.

The input must be read from standard input.

Output

Print one line of output for each test case. If there is no solution, output `none`; otherwise, print two integers in a line, separated by a space. The first integer denotes the maximum length of a substring appearing at least m times; the second integer gives the rightmost starting position of this substring.

The output must be written to standard output.

Sample Input

```
3
baaaababababbababbab
11
baaaababababbababbab
3
cccccc
0
```

Sample Output

```
5 12
none
4 2
```

B - Beehives

Source file name: `beehives.c, beehives.cpp` *or* `beehives.java`

Bees are one of the most industrious insects. Since they collect nectar and pollen from flowers, they have to rely on the trees in the forest. For simplicity they numbered the n trees from 0 to $n-1$. Instead of roaming around all over the forest, they use a particular list of paths. A path is based on two trees, and they can move either way i.e. from one tree to another in straight line. They don't use paths that are not in their list.

As technology has been improved a lot, they also changed their working strategy. Instead of hovering over all the trees in the forest, they are targeting particular trees, mainly trees with lots of flowers. So, they planned that they will build some new hives in some targeted trees. After that they will only collect their foods from these trees. They will also remove some paths from their list so that they don't have to go to a tree with no hive in it.

Now, they want to build the hives such that if one of the paths in their new list go down (some birds or animals disturbs them in that path) it's still possible to go from any hive to another using the existing paths.

They don't want to choose less than two trees and as hive-building requires a lot of work, they need to keep the number of hives as low as possible. Now you are given the trees with the paths they use, your task is to propose a new bee hive colony for them.

Input

Input starts with an integer T ($T \leq 50$), denoting the number of test cases.

Each case starts with a blank line. Next line contains two integers n ($2 \leq n \leq 500$) and m ($0 \leq m \leq 20000$), where n denotes the number of trees and m denotes the number of paths. Each of the next m lines contains two integers u v ($0 \leq u, v < n$, $u \neq v$) meaning that there is a path between tree u and v. Assume that there can be at most one path between tree u to v, and needless to say that a path will not be given more than once in the input.

The input must be read from standard input.

Output

For each case, print the case number and the number of beehives in the proposed colony or `impossible` if it's impossible to find such a colony.

The output must be written to standard output.

Sample input

```
3

3 3
0 1
1 2
2 0

2 1
0 1

5 6
0 1
1 2
1 3
2 3
0 4
3 4
```

Sample output

```
Case 1: 3
Case 2: impossible
Case 3: 3
```

C - Cucumber Market

Source file name: `cucumber.c, cucumber.cpp` *or* `cucumber.java`

Cucumber Boy is young and loves cucumbers. Therefore, Cucumber Boy will go to the cucumber market to buy some cucumbers.

Different cucumbers may have different costs in yen. Cucumber Boy's mother gave him budget yen. However, he does not understand money well.

He just chooses some unique cucumbers he likes. If the total price of the chosen cucumbers is not greater than budget yen, he can buy them, otherwise he cannot.

You are given a list of prices, the budget, and the minimum number of unique cucumbers Cucumber Boy wants to buy. Your task is to determine if Cucumber Boy can buy any number of unique cucumbers whose count is at least the minimum number of cucumbers.

Input

The input consists of several test cases. The first line of each test case contains three integer numbers c, b, and m ($1 \leq c \leq 50$, $1 \leq b \leq 10^6$, $1 \leq m \leq c$) separated by blanks. The number c indicates the number of cucumbers initially selected by Cucumber Boy, b represents his budget, and m the minimum number of cucumbers that he wishes to buy from the cucumbers initially selected. Then one line follows with a list of c integer numbers, blank separated, indicating the prices of the c cucumbers initially chosen.

The input ends with $c = b = m = 0$, which should not be processed as a test case.

The input must be read from standard input.

Output

For each test case answer `YES` if Cucumber Boy can buy any set of m unique cucumbers from the cucumbers c with budget b, and `NO` if there is some set of m cucumbers that is too expensive for him.

The output must be written to standard output.

Sample Input	**Sample Output**

`4 1110 3`
`1000 1 10 100`
`4 1109 3`
`1000 1 10 100`
`0 0 0`
	`YES`
	`NO`

D - Distributing Ballot Boxes

Source file name: `boxes.c, boxes.cpp` *or* `boxes.java`

Today, an important event is taking place in Spain: General Elections. Every single resident of the country aged 18 or over is asked to vote in order to choose representatives for the Congress of Deputies and the Senate. You do not need to worry that all judges will suddenly run away from their supervising duties, as voting is not compulsory.

The administration has a number of ballot boxes, those used in past elections. Unfortunately, the person in charge of the distribution of boxes among cities was dismissed a few months ago due to financial restraints. As a consequence, the assignment of boxes to cities and the lists of people that must vote in each of them is arguably not the best. Your task is to show how efficiently this task could have been done.

The only rule in the assignment of ballot boxes to cities is that every city must be assigned at least one box. Each person must vote in the box to which he/she has been previously assigned. Your goal is to obtain a distribution which minimizes the maximum number of people assigned to vote in one box.

In the first case of the sample input, two boxes go to the first city and the rest to the second, and exactly 100000 people are assigned to vote in each of the (huge!) boxes in the most efficient distribution. In the second case, $1, 2, 2$ and 1 ballot boxes are assigned to the cities and 1700 people from the third city will be called to vote in each of the two boxes of their village, making these boxes the most crowded of all in the optimal assignment.

Input

The first line of each test case contains the integers N $(1 \leq N \leq 500000)$, the number of cities, and B $(N \leq B \leq 2000000)$, the number of ballot boxes. Each of the following N lines contains an integer a_i, $(1 \leq a_i \leq 5000000)$, indicating the population of the i-th city.

A single blank line will be included after each case. The last line of the input will contain `-1 -1` and should not be processed.

The input must be read from standard input.

Output

For each case, your program should output a single integer, the maximum number of people assigned to one box in the most efficient assignment.

The output must be written to standard output.

Sample Input

```
2 7
200000
500000

4 6
120
2680
3400
200

-1 -1
```

Sample output

```
100000
1700
```

E - Tagalog Dictionary

Source file name: `dictionary.c, dictionary.cpp` *or* `dictionary.java`

In the first half of the 20th century, around the time that Tagalog became the national language of the Philippines, a standardized alphabet was introduced to be used in Tagalog school books (since then, the national language has changed to a hybrid ‘‘Pilipino’’ language, as Tagalog is only one of several major languages spoken in the Philippines).

Tagalog’s 20-letter alphabet is as follows:

```
a b k d e g h i l m n ng o p r s t u w y
```

Note that not all letters used in English are present, ‘k’ is the third letter, and ‘ng’ is a single letter that comes between ‘n’ and ‘o’.

You are compiling a Tagalog dictionary, and for people to be able to find words in it, the words need to be sorted in alphabetical order with reference to the Tagalog alphabet.

Input

The input consists of several test cases. Each test case consists of a single line containing a list of Tagalog words, blank separated. Each test case contains at most 50 words.

The end of the input is given with a line containing the word `END`.

The input must be read from standard input.

Output

For each test case, output the given list of Tagalog words, blank separated, in Tagalog alphabetical order.

The output must be written to standard output.

Sample input	**Sample output**

`abakada alpabet tagalog ako`
`siya niya kaniya ikaw ito iyon`
`kaba baka naba ngipin nipin`
`END`
	`abakada ako alpabet tagalog`
	`kaniya ikaw ito iyon niya siya`
	`baka kaba naba nipin ngipin`

F - Peer Review

Source file name: `review.c, review.cpp` *or* `review.java`

For scientific conferences, scientists submit papers presenting their ideas, and then review each other's papers to make sure only good papers are presented at the conference. Each paper must be reviewed by several scientists, and scientists must not review papers written by people they collaborate with (including themselves), or review the same paper more than once.

You have been asked to write a program to check if your favorite conference is doing things right. Whether a paper is being reviewed too much, too little, or by the wrong people - the organizers must know before it is too late!

Input

The first line in each test case has two integers, K $(1 \leq K \leq 5)$ and N $(1 \leq N \leq 1000)$. K is the number of reviews that each paper will receive, while N is the number of papers to be reviewed. The conference only accepts papers with a single author, and authors can only present a single paper at the conference.

Each of the next N lines describes an author and includes the name of the institution to which the author belongs, followed by the list of the K papers he or she has been requested to review. It is assumed that researchers from the same institution collaborate with each other, whereas researchers from different institutions don't. All institution names are shorter than 10 characters, and contain only upper or lowercase letters and no whitespace. Since we have as many papers as authors, papers are identified by their author's index; paper 1 was written by the first author in the list, and paper N was written by the last author.

The end of the test cases is marked with a line containing $K = 0$ and $N = 0$. You should generate no output for this line.

The input must be read from standard input.

Output

For each test case, your program should output `NO PROBLEMS FOUND` (if all rules are being followed) or `P PROBLEMS FOUND`, where P is the number of rule violations found (counting at most 1 violation per paper). If there is exactly one rule violation overall, your program should output `1 PROBLEM FOUND`.

The output must be written to standard output.

Sample input	**Sample output**

`2 3`
`UCM 2 3`
`UAM 1 3`
`UPM 1 2`
`2 3`
`UCM 2 3`
`UAM 1 2`
`UPM 2 2`
`0 0`
	`NO PROBLEMS FOUND`
	`3 PROBLEMS FOUND`

G - Texas Summers

Source file name: `texas.c`, `texas.cpp` *or* `texas.java`

Summer in Texas can be very hot. But Texans are tough, and in a weird way some of them enjoy the heat; for even the heat is bigger in Texas. But the heat can be quite uncomfortable for computer science students who are new to Texas and are not used to the sweating it can cause.

These students want to minimize the amount they sweat while walking from their dormitory to class. When they enter the sunshine, they begin sweating. The longer they stay in the sun, the more they sweat. The amount they sweat is proportional to the square of how long they have been continuously exposed to the sun. Put another way, if they are in the sun continuously for s seconds, they will sweat Cs^2 gallons, where C is a constant which is different for different students. But if they find a shady spot along the way, their continuous sun exposure is broken and they stop sweating immediately. They can then sit in the shade and cool off completely before continuing on their journey to class. Of course, leaving the shade means they begin sweating again.

Write a program that helps a student find a path from the dormitory to class which minimizes the total amount of sweat she expends.

Input

Input consists of several test cases. Each test case begins with a line containing an integer $0 \leq n \leq 2500$, the number of shady spots. Each of the next n lines contains a pair of integers x y specifying the coordinates of a shady spot. No two shady spots have the same coordinates. Following the shady spots are two more lines in the same format which specify the coordinates of the student's dormitory and class, respectively.

The last test case is denoted by $n = -1$ and must not be processed.

The input must be read from standard input.

Output

For each test case, print a path the student can take to get from her dormitory to class. The path should minimize the total sweat produced along the whole path. Print the path as indexes of the shady spots (from the order given in the input, with the first shady spot having index 0). If the best path contains no shady spots, output a single “`-`”. If there are multiple paths that minimize the total sweat, print any. There must be a blank line between cases.

The output must be written to standard output.

Sample input

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

Sample output

```
1
2

1
3
5
4
2

-
```

H - Haunted Graveyard

Source file name: `haunted.c, haunted.cpp` *or* `haunted.java`

Tonight is Halloween and Scared John and his friends have decided to do something fun to celebrate the occasion: crossing the graveyard. Although Scared John does not find this fun at all, he finally agreed to join them in their adventure. Once at the entrance, the friends have begun to cross the graveyard one by one, and now it is the time for Scared John. He still remembers the tales his grandmother told him when he was a child. She told him that, on Halloween night, ‘‘haunted holes’’ appear in the graveyard. These are not usual holes, but they transport people who fall inside to some point in the graveyard, possibly far away. But the scariest feature of these holes is that they allow one to travel in time as well as in space; i.e., if you fall inside a ‘‘haunted hole’’, you appear somewhere in the graveyard a certain time before (or after) you entered the hole, in a parallel universe otherwise identical to ours.

The graveyard is organized as a grid of $W \times H$ cells, with the entrance in the cell at position (0,0) and the exit at $(W-1,\ H-1)$. Despite the darkness, Scared John can always recognize the exit, and he will leave as soon as he reaches it, determined never to set foot anywhere in the graveyard again. On his way to the exit, he can walk from one cell to an adjacent one, and he can only head to the North, East, South or West. In each cell there can be either one gravestone, one ‘‘haunted hole’’, or grass:

- If the cell contains a gravestone, you cannot walk over it, because gravestones are too high to climb.
- If the cell contains a ‘‘haunted hole’’ and you walk over it, you will appear somewhere in the graveyard at a possibly different moment in time. The time difference depends on the particular ‘‘haunted hole’’ you fell into, and can be positive, negative or zero.
- Otherwise, the cell has only grass, and you can walk freely over it.

He is terrified, so he wants to cross the graveyard as quickly as possible. And that is the reason why he has phoned you, a renowned programmer. He wants you to write a program that, given the description of the graveyard, computes the minimum time needed to go from the entrance to the exit. Scared John accepts using ‘‘haunted holes’’ if they permit him to cross the graveyard quicker, but he is frightened to death of the possibility of getting lost and being able to travel back in time indefinitely using the holes, so your program must report these situations.

Figure 1 illustrates a possible graveyard (the second test case from the sample input). In this case there are two gravestones in cells $(2,1)$ and $(3,1)$, and a ‘‘haunted hole’’ from cell $(3,0)$ to cell $(2,2)$ with a difference in time of 0 seconds. The minimum time to cross the graveyard is 4 seconds. If you do not use the ‘‘haunted hole’’, you need at least 5 seconds.

Input

The input consists of several test cases. Each test case begins with a line containing two integers W and H ($1 \leq W, H \leq 30$). These integers represent the width W and height H of

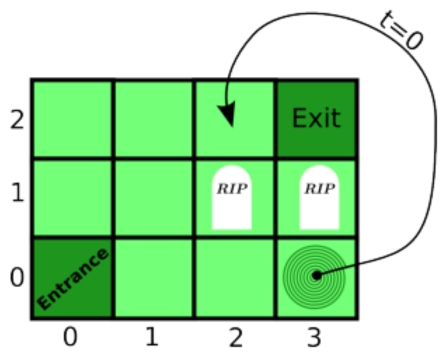

Figure 1: Sample graveyard.

the graveyard. The next line contains an integer G ($G \geq 0$), the number of gravestones in the graveyard, and is followed by G lines containing the positions of the gravestones. Each position is given by two integers X and Y ($0 \leq X < W$ and $0 \leq Y < H$).

The next line contains an integer E ($E \geq 0$), the number of "haunted holes", and is followed by E lines. Each of these contains five integers X_1, Y_1, X_2, Y_2, T. (X_1, Y_1) is the position of the "haunted hole" ($0 \leq X_1 < W$ and $0 \leq Y_1 < H$). (X_2, Y_2) is the destination of the "haunted hole" ($0 \leq X_2 < W$ and $0 \leq Y_2 < H$). Note that the origin and the destination of a "haunted hole" can be identical. T ($-10000 \leq T \leq 10000$) is the difference in seconds between the moment somebody enters the "haunted hole" and the moment he appears in the destination position; a positive number indicates that he reaches the destination after entering the hole. You can safely assume that there are no two "haunted holes" with the same origin, and the destination cell of a "haunted hole" does not contain a gravestone. Furthermore, there are neither gravestones nor "haunted holes" at positions $(0, 0)$ and $(W-1, H-1)$.

The input will finish with a line containing 0 0, which should not be processed.

The input must be read from standard input.

Output

For each test case, if it is possible for Scared John to travel back in time indefinitely, output `Never`. Otherwise, print the minimum time in seconds that it takes him to cross the graveyard from the entrance to the exit if it is reachable, and `Impossible` if not.

The output must be written to standard output.

Sample input	**Sample output**

`3 3`
`2`
`2 1`
`1 2`
`0`
`4 3`
`2`
`2 1`
`3 1`
`1`
`3 0 2 2 0`
`4 2`
`0`
`1`
`2 0 1 0 -3`
`0 0`
	`Impossible`
	`4`
	`Never`