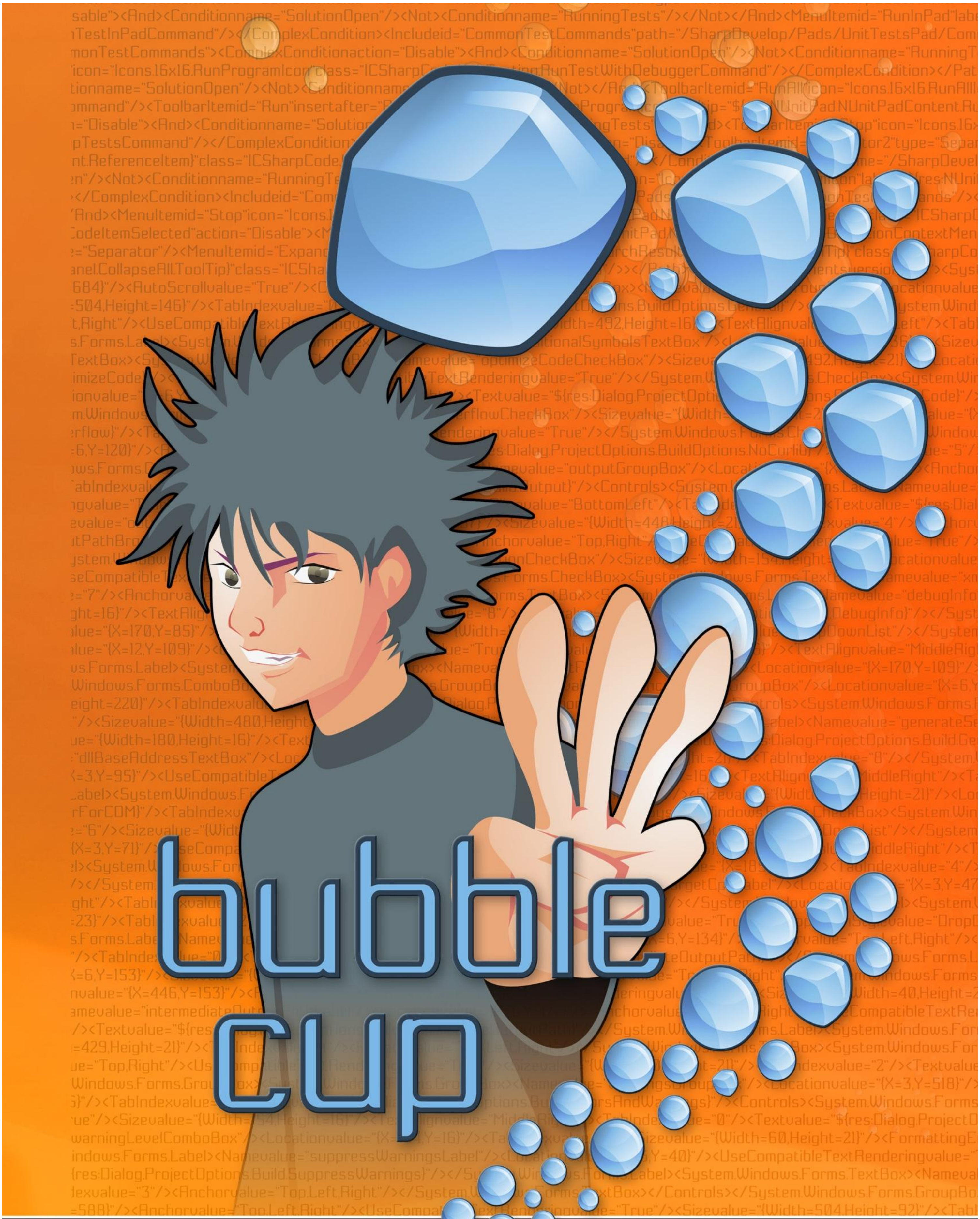


bubble cup

student
programming
contest

Microsoft | Development Center Serbia

MDCS

bubble cup
Student programming contest
Microsoft Development Center Serbia

BUBBLE CUP 2010

Student programming contest
Microsoft Development Center Serbia

Tasks and Solutions

Belgrade, 2010

Authors:

Milan Novaković
Milan Vugdelija
Andreja Ilić
Andrija Jovanović
Mladen Radojević
Dimitrije Filipović
Nikola Mihajlović

Cover:

Sava Čajetinac

Typesetting:

Andreja Ilić

Proofreader:

Slobodanka Jovanović

Volume editor:

Dragan Tomić

Contents

Preface

Greetings, fellow contestants!

It is my great pleasure to express our warm welcome to all of you and thank you for coming and taking part in the third edition of the Bubble Cup in Belgrade, Serbia.

Our team at Microsoft Development Center Serbia (MDCS) is composed of many contestants in various disciplines over the last 10 to 20 years. Solving difficult problems is part of our DNA. As soon as the MDCS team had grown to a respectable size we looked for an opportunity to give back to our community. Our members are folks like YOU who spend countless hours reading, learning, practicing and improving algorithms.

We live in the information age. The applications of computer science have changed the world, and they will continue to do so. Some of you are rising stars who will transform the world and we are happy to accompany you on this tiny part of your journey.

Today, we are in the 3rd edition of Bubble Cup. Initially, this was a local contest, but over the last two years it has become a regional event. I hope that Bubble Cup will continue to develop and grow and that the competition level will further increase in the future.

In addition to competing hard, I hope that you will have fun and come back to Belgrade for more editions of BubbleCup in the future.

Sincerely,
Dragan Tomic
Group Manager - MDCS

About Bubble Cup and MDCS

Microsoft Development Center Serbia (MDCS) has been working for 5 years on various Microsoft products and services. Around 40 core staff (developers, testers and program managers) have been working to investigate, design and implement solutions which are be used throughout the world. MDCS was founded by Bodin Drešević in Belgrade and is currently led by Dragan Tomić. Divided into small teams, people in MDCS worked on projects for Windows, Office, SQL Server, Live Labs and Bing groups and impacted many products such as Windows 7, Office 14 and SQL Server 2008. The key values MDCS shows throughout the work are strong applied math knowledge, machine learning expertise and core understanding of relational database systems. The mission for MDCS is to lead the development of new technologies and to help the region to do the same.

Bubble Cup is a student programming contest originally designed as a training for the ACM ICPC. The goal is to help young people perfect their coding skills and prepare them for the world-wide competition. Bubble Cup started in 2008 and now, two years later, it increased its reach to regional countries with stronger than ever teams from Croatia, Romania and Serbia. The competition was organized into 2 online qualification rounds and a final round, which takes place in Belgrade, Serbia every year. The official website of the competition is www.bubblecup.org, and it includes rules, details, problem sets, information about competitors and organizers and an archive of previous competitions.

I’m a BubbleCupper… what about you?

bubble cup
Student programming contest
Microsoft Development Center Serbia

Tasks and Solution
from Finals and Qualifications

Finals

The finals of the third Bubble Cup were held on 11 September 2010 at the School of Electrical Engineering in Belgrade. Fifteen teams competed in solving nine problems. The competition lasted five hours, and the goal was to solve as many problems as possible, but also as quickly as possible – if two or more teams solved the same number of problems, the one who needed the least time was ranked best. Additionally, teams received bonus points depending on their qualification results, but for each problem there were time penalties if a team had incorrect submissions before managing to solve it.

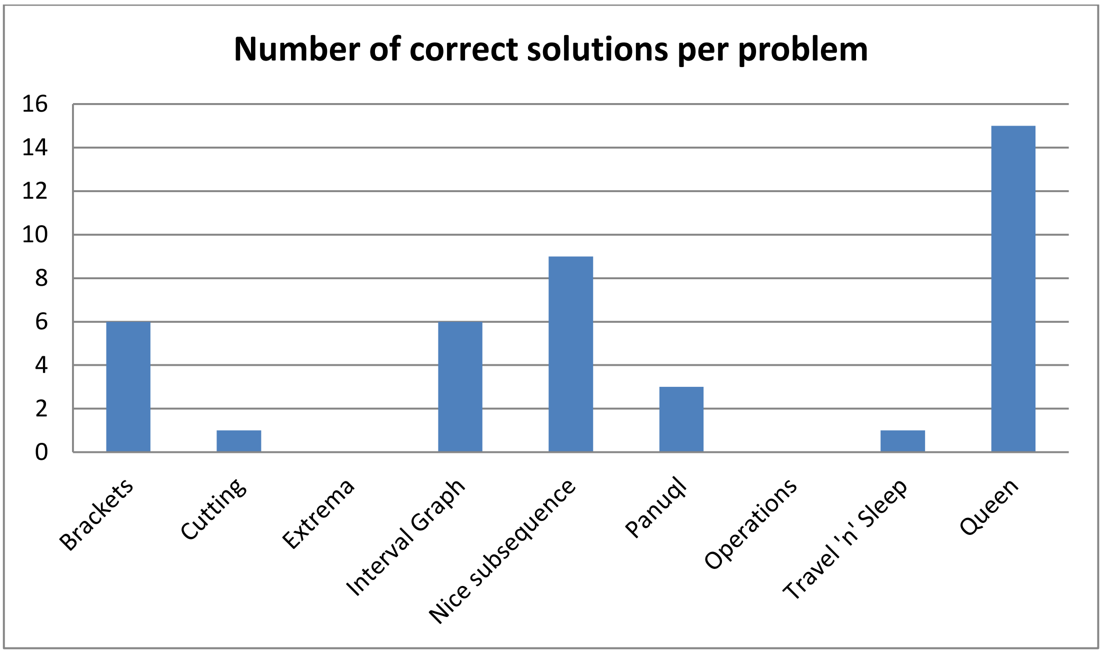
The problems were of varying difficulty – on one end, one problem was solved by every team, while on the other there were two problems that no team managed to solve (and for one of those no one even attempted to submit a solution!).

Team **mljivo** (members: Tomislav Grbin, Luka Dondjivić and Davor Jerbić, all from the faculty FER Zagreb) won the competition. They managed to solve five problems and edged out **Suit Up!** (Ivan Katanić, Marin Smiljanić and Stjepan Glavina, all high school students), who also had five solved problems but a larger time penalty. The third place went to **Prongrammers** (Slobodan Mitrović, Rajko Nenadov and Nemanja Škorić, from PMF Novi Sad), who were the quicker of the two teams with four solved problems.

The final scoreboard

Rank	Team name	Team crew	Points	Penalty
01	**Mljivo**	Tomislav Grbin Luka Dondivic Davor Jerbić	5	392
02	**Suit Up!**	Ivan Katanic Marin Smiljanic Stjepan Glavina	5	876
03	**Prongrammers**	Slobodan Mitrovic Rajko Nenadov Nemanja Skoric	4	623
04	**I like it RAF**	Vanja Petrovic Tankovic Nenad Božidarević Milan Tomić	4	863
05	**The Ninjas**	Nikola Milosavljevic Aleksandar Trokicic Marko Djikic	3	210
06	**Rivals Aren't Frightened (yet)**	Maja Kabiljo Igor Kabiljo Milos Stankovic	3	300
07	**ZBrains**	Mircea Dima Flaviu Pepelea Duta Vlad	3	740
08	**ššuga**	Anton Grbin Viktor Braut Vjekoslav Giacometti	2	245
09	**S-Force**	Dusan Zdravkovic Stefan Stojanovic Dimitrije Dimic	2	247
10	**Eštaf**	Matija Osrečki Ognjen Dragoljevic Goran Gasic	2	280
11	**NS Boys**	Demjan Grubic Boris Grubic Mario Cekic	2	517
12	**Burek**	Frane Kurtović Adrian Satja Kurdija Tomislav Gudlek	1	-19
13	**Nameless**	Nemanja Marsenic Nikola Trkulja Mikloš Kalozi	1	3
14	**Strawhats**	Damir Ferizovic Daniel Ferizovic Dominik Gleich	1	70
15	**Seek & Destroy**	Predrag Ilkic Aleksandar Milovanovic Dejan Pekter	1	82

Statistics



ID	Problem name	Number of teams with correct solutions	Number of teams with at least one submission attempt	Total percentage of accepted submissions
A	Brackets	6	14	12%
B	Cutting	1	1	100%
C	Extrema	0	0	/
D	Interval Graph	6	12	10%
E	Nice Subsequence	9	15	13%
F	Panuql	3	7	15%
G	Operations	0	4	0%
H	Travel 'n' Sleep	1	3	7%
I	Queen	15	15	55%

Problem A: Brackets

Author: ***Milan Vugdelija*** *Implementation and analysis:* ***Milan Vugdelija***

Statement:

You are given an array of n strings, and each string contains only open and closed brackets.

Find out if those strings can be sorted in such a way that after the concatenation of all strings, a valid arrangement of brackets is achieved (like as in a math expression after removing all other characters).

Input:

The first line contains the positive integer n ($1 \leq n \leq 100{,}000$), the number of strings. Each of the next n lines contains a sequence of ‘(‘ and ‘)’ characters, up to the end of the line.
Total number of all characters in all strings does not exceed 10,000,000 (ten millions).

Output:

The output consists of one word:

- “yes” (without quotes) if the required arrangement of strings exists
- “no” if it doesn’t exist

Example input:

```
3
(()
(
))
```

Example output:

```
yes
```

Time and memory limit: 3s / 64MB

Solution and analysis:

In this problem each string is equivalent to a string starting with zero or more closed brackets, followed by zero or more open brackets. For example, underlined brackets are matching and can be removed from the string: “ $\underline{()}$) ($\underline{(())}$(”, reducing it to: “) ((”. So, each string is completely characterized with two integer attributes: the number of unmatched closed brackets at the beginning of a string and the number of unmatched open brackets at the end of a string.

Let’s introduce the following notation:

$UO[i]$ − Number of unmatched open brackets at the end of i-th string;
$UC[i]$ − Number of unmatched open brackets at the beginning of i-th string;
$\mathrm{B}[i] = \mathrm{UO}[i] - \mathrm{UC}[i]$ − Bracket balance of i-th string, which can also be negative.

In the previous example $UC = 1,\ UO = 2,\ B = 1$.

If $\sum_{i=1}^{n} B[i] \neq 0$, it is clearly impossible to arrange the strings as required.

Otherwise (if $\sum_{i=1}^{n} B[i] = 0$), we can first sort the strings according to the following criteria:

- First we put all strings with positive (i.e. non-negative) balance, and then all strings with negative balance.
- UC should be increasing among strings with positive balance, and UO should be decreasing among strings with negative balance.
- If two strings with positive balance have the same UC (or two strings with negative balance have the same UO), we first put the one with higher B.

For the global string (obtained by concatenation of given strings), we want to check that at each point the number of closed brackets does not exceed the number of open brackets, i.e. that balance at each position is non-negative.

It is not difficult to prove that for any two consecutive strings the suggested order maximizes the lowest balance over all positions in the global string. Consequently, if a solution exists, it can be obtained by sorting as described. Let's prove this.

We can look at the string as an ordered triple $(UC[i], UO[i], B[i])$. Let's assume that these strings are arranged in correct form. In other words:

$$UC[1] = 0$$
$$UC[i] \leq B[1] + B[2] + \cdots + B[i-1] \text{ , for } i \in [2, n]$$

First, let us prove that a nonnegative balanced string can be moved in front of negative ones. The necessary and sufficient condition for this is to prove that, if we have two successive strings with indexes i and $i+1$, where $B[i] < 0$ and $B[i+1] \geq 0$, we can swap them. Denote $B[1] + \cdots B[i]$ as $SUM[i]$. Now from $UC[i] \leq SUM[i-1]$ and $UC[i+1] \leq SUM[i-1] + B[i]$ we have that $UC[i+1] \leq SUM[i-1]$ and $UC[i] \leq SUM[i-1] + B[i+1]$ because $B[i]$ is negative and $B[i+1]$ is nonnegative. Of course, because this is a successive string, described transformation does not affect the rest of strings. With this operation we only increase the required differences.

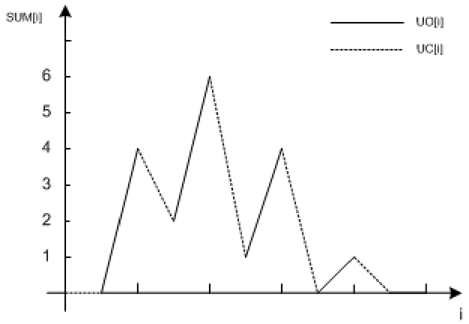


Figure 1. We can look at the arrangement as **Dyck lattice path** with $UC[i]$ and $UO[i]$ as jups.

Now we can look at the strings with positive balance and the strings with negative balance as two sub-problems for sorting. For strings with positive balances, if we swap successive strings so that one with smaller UC goes first, again we have only strengthened the inequalities. On the other hand, for negative ones this is not so obvious. But if we put this ‘on paper’ we get this (again for successive strings):

$$\begin{array}{ccc} SUM[i-1] \geq UC[i] & & SUM[i-1] \geq UC[i+1], \qquad \text{because } SUM[i-1] > SUM[i] \\ SUM[i] \geq UC[i+1] & \Rightarrow & SUM[i-1] + UO[i+1] - UC[i+1] \geq UO[i+1] + UC[i] - UO[i] \geq UC[i] \\ UO[i+1] \geq UO[i] & & \end{array}$$

Using the above transformation over successive strings, starting from a proper arrangement we can generate a new proper arrangement – which is also the output of our sort with the above criteria.

Therefore, it is enough to check strings in described order. If for this order balance stays non-negative (and is zero at the end), the answer is “yes”, but otherwise “no”.

Implementation:

- Read the strings and sort them as described,
- Check whether balance is non-negative at all points in concatenated string and zero at the end.

Complexity

Time complexity is obviously $O(n\log(n) + N)$, where N is the total number of characters in all given strings. Memory complexity is $O(N + n)$.

Problem B: Cutting

Author: ***Andreja Ilić*** *Implementation and analysis:* ***Andreja Ilić***

Statement:

Given an integer m and an integer sequence a of length n, you have to split the given sequence into consecutive subsequences. The sum of elements in any subsequence must be less than or equal to m. Let M be the sum of maximal elements of the subsequences. Your task is to find the split that minimizes M.

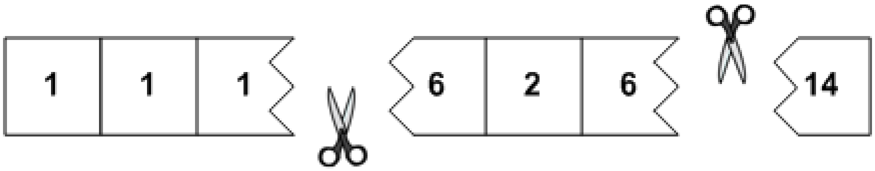


Figure 1. One possible cutting for the given example bellow

Input:

The first line contains two positive integers n and m ($1 \le n \le 100\ 000$ and $1 \le m \le 10^9$), where n is the number of elements in the given sequence and m is the maximal allowed sum of elements in a subsequence. The following line contains n integers – elements of the sequence. All elements are in the range $[0, 10^6]$.

Output:

The output consists of one integer:

- "-1" (without quotes) if a solution does not exist
- otherwise, the minimal sum of maximal elements for any split(M)

Example input:

```
7 14
1 1 1 6 2 6 14
```

Example output:

```
21
```

Time and memory limit: 1s / 64MB

Solution and analysis:

Firstly, we can see that a cutting exists if all elements of the given sequence are smaller than or equal to m. This is the first thing that we are going to check. From now on, we are assuming that all elements are not greater than m.

Let's start thinking backwards – not from the sequence itself but from the subsequences. If we denote subsequence $a_i, a_{i+1}, \dots, a_j$ as $a[i, j]$, then in the final cutting the last subsequence has the form $a[k, n]$ for some $1 \le k \le n$. Now we can say that the final solution is $\max\{a_k, a_{k+1}, \dots, a_n\} + d[k-1]$, where

$d[k-1]$ is the optimal cutting for the first $k-1$ elements of a. This cutting has to be optimal, because otherwise the cutting for the whole sequence would not be optimal either.

This smells like dynamic programming. Let's define array d of length n as:

$$d\ [k] = \text{optimal value of cutting sequence } a[1,k]$$

We have the following recurrent relation between the elements of d:

$$d\ [k] = \min_{bound[k] \le i \le k} \{d\ [i-1] + \max\{a[i,k]\}\},$$

where $bound[k]$ is the minimal index such that the sum of elements of $a[bound[k],k]$ is less than or equal to m. In other words, if the element a_k is the right boundary of some subsequence, then its left boundary has to be in the above segment. For the base of the dynamic programming algorithm, we can define $d[0]=0$ and $d[1]=a[1]$. The final solution is stored in the element $d[n]$.

Implementation:

The tricky part of this problem was implementation. Let's see how we can initialize bounds fast. Array $bound$ is non-decreasing ($bound[k] \le bound[k+1]$). When we want to initialize the element $bound[k]$, we only have to look in the segment $[bound[k-1],k]$. If we accumulate the current sum in this segment, the initialization of the whole array can be implemented in linear time.

```
==================================================================================
bound [1] = 1;
currentSum = a [1];
for k = 2 to n do
        bound [k] = bound [k - 1];
        currentSum = currentSum + a [k];
        while (currentSum> m)
                currentSum = currentSum - a [bound [i]];
                bound [i] = bound [i] + 1;
==================================================================================
```

Algorithm for bound initialization

What about array d? Naive implementation of the above recurrent relation leads to time complexity of $O(n^2)$, which is very slow for our constraints. The key observation is that we do not need all indices from the segment $[bound[k],k]$ when we want to find a minimum. We only need indices from the set

$$I_k = \{k, bound[k]\} \cup \{i \in [bound[k]-1, k-1] | a_i > a_{i+1}, \dots, a_k\}$$

Therefore, we have to check for boundaries and only for elements that are strictly greater than ones before them. This is intuitively clear, because if we have some maximum in a subsequence we want to stretch to the left as long as we can.

We can store these indices in a list. When we move from k-th to $(k+1)$-th element, we remove only some elements from the head and some elements from the tail of this list. From the beginning we are going to remove indices that are smaller than $bound[k+1]$. Since every index in the list represents an element that is greater than the ones before, from the end of the list we are going to remove indices if the corresponding elements are less than or equal to $a[k+1]$. After that we are going to add new element k at the end of the list. All of this is possible because both the indices in the list and their corresponding elements are sorted in a strictly increasing order.

And what about the minimum of these elements? Theoretically, this list can be very long. Well, we are going to store values $d\ [i] + \max\{a[i+1,k]\}$ in a heap structure (all of them except for boundaries). When we move to a new element, as we remove something from the list, we remove the corresponding element from the heap. In the end, only for the last element of the list, which had the value $\max\{a[last+1,k-1]\}$ before adding the new one, is going to change – it becomes $a[k] = \max\{a[last+1,k]\}$.

```
==================================================================================
     d [0] = 0 and d [1] = a [1];
     add in heap (1, a [1]);        // index and value
     add in list 1;                 // index of elements in heap
     for k = 2 to n do
            while first index in list is less than bound [k]
                   remove it from heap;
                   remove it from list;

            while last element in list is less than or equal to a [k]
                   remove it from heap;
                   remove it from list;

            if (heap is not empty)
                   remove last element lastElement from heap;
                   add inheap(lastElement, d [lastElement] + a [k]);

            maxInBound = max (a [k], a [firstElementInList]);
            d [k] = max (a [k] + d [k - 1],
                   d [bound [k] - 1] + maxInBound,
                   min in heap);

            add in heap (k, d [k]);
            add in list (k);
==================================================================================
```

Pseudo-code for described algorithm

Complexity:

Initialization of bounds is linear (as we have seen). Every element from the sequence is going to be added to the heap (and list) only once and removed from it at most once. Initialization of elements $d\ [k]$ requires one call for finding a minimum in the heap. This leads us to total time complexity of $O(n \log n)$.

Memory complexity is $O(n + MaxElement)$, because we must store information of positions in heap structure. Also, we have to pay attention to cumulative sums and use int64 for storing this information.

Test data:

Test corpus for this problem contains 30 test cases. Short description of test cases is given in Table 1.

Num	n	m	maximal $a\ [i]$	Solution	Desciption
01	10	20	15	37	By hand
02	20	1000	46	46	You don't need to cut

03	100	10000	978	5621	Random
04	1000	100000	32746	-1	-1
05	100	1000	452	452	Sum is equal to M
06	10000	100000	100000	834993296	Every element is one subsequence
07	50000	100000000	132767	7830539	~ 50 subsequences of 1.000 elements
08	99999	100000000	132867	15540259	~ 100 subsequences of 10.00 elements
09	99999	98765432	19753	296266	~ 10 subsequences of 10.000 elements
10	99999	99999999	40007	1159863	~ 10 subsequences of 10.000 elements
11	99999	99999999	32768	556893	Random
12	99999	99999999	532767	273208529	~ 1000 subsequences of 100 elements
13	99999	99999999	32768	556910	changing big - small subsequence
14	100000	7654321	123456	12345600000	Every element is equal to M
15	1	100	50	50	One element
16	100000	98765432	999987	479320035	Random monotonic subsequences
17	99999	100000000	9999	9999	Many zeros
18	100000	10000	9999	999900000	Monotonic down subsequences
19	99999	67834589	987655	-1	One element M + 1 and all ones
20	80000	100000000	532767	221021568	~ 10 subsequences of 10.000 elements
21	90000	10000	6012	92936424	Monotonic up subsequences
22	999999	9999999	999998	4064318963217351	Monotonic down subsequences
23	80000	100000000	999982	306422910	Random monotonic subsequences
24	80000	100000000	100	100	Random small
25	80000	100000000	49999	1449530	~ 20 subsequences of 10.000 elements
26	30000	100	0	0	All zeros
27	10000	1000	999	9990000	All equal to M – 1
28	100000	100000000	999996	355237902	Random monotonic subsequences
29	100000	100000000	504243	528239	~ 3 subsequences of 30.000 elements
30	100000	100000000	532767	275873972	~ 1000 subsequences of 1000 elements

Problem C: Extrema

Author: ***Andreja Ilić*** *Implementation and analysis:* ***Nikola Mihajlović*** ***Andreja Ilić***

Statement:

Let's define a function f as $f(x_1, \dots, x_n) = \sum_{i=1}^{n} a_i x_i$, where $a_i \in [0,1]$ and $\sum_{i=1}^{n} a_i = 1$.
Given two points X and Y from R^n and value $f(X) = C$, find minimum and maximum value for $f(Y)$.

Input:

The first line of input contains the number n ($1 \le n \le 100.000$). The second line contains numbers $x_1, \dots, x_n$ separated by a space. The third line contains $y_1, \dots, y_n$. The final line contains the number$C$. It is guaranteed that there will always be coefficients a_i for which $f(X) = C$ satisfying the above conditions.

Output:

The first line of output should contain minimum value for $f(Y)$ rounded to two decimal places, and the second line should contain the maximum value for $f(Y)$, also rounded to two decimal places.

Example input:

```
3
0 2 1
0 0 1
0.75
```

Example output:

```
0.00
0.75
```

Time and memory limit: 1s / 64MB

Solution and analysis:

This problem requires some math skills. At first sight, it seems to be a kind of linear programming problem, but it can be solved quite elegantly.

We have a function f and we know that it has the form $f(x_1, \dots, x_n) = \sum_{i=1}^{n} a_i x_i$, where $a_i \in [0,1]$ and $\sum_{i=1}^{n} a_i = 1$. The coefficients satisfying these conditions are called **barycentric**. We can easily spot the following property in one-dimensional space: given $x_1, \dots, x_n$ and x, $x \in [\min\{x_i\}, \max\{x_i\}]$ there exists a function f as defined above such that $x = f(x_1, \dots, x_n) = \sum_{i=1}^{n} a_i x_i$. It is enough to vary coefficients for minimum and maximum of x_i, the rest can be 0. Now we know that C must be between these two values.

Extending this to the two-dimensional case is harder. This is stated by the following theorem.

Theorem: Given points $(x_1, y_1), \dots, (x_n, y_n)$ and (x, y) from R^2, (x, y) are in the **convex hull** of $(x_1, y_1), \dots, (x_n, y_n)$ iff there exists a function f as defined above such that $(x, y) = (f(X), f(Y))$, where $X = (x_1, \dots, x_n)$ and $Y = (y_1, \dots, y_n)$.

Again, we can accomplish this by varying just the coefficients a_i for points which are vertices of the convex hull, the rest can be 0.

From the theorem, we conclude that the possible values for $f(Y)$ are just projections of the points which belong to the convex hull to the y-axis. The additional constraint $f(X) = C$ restricts the set of possible points to the ones which lie on the intersection of the convex hull and the line $x = C$. This intersection is a segment (or just a point in extreme case), so the final solution will be the boundaries of this segment.

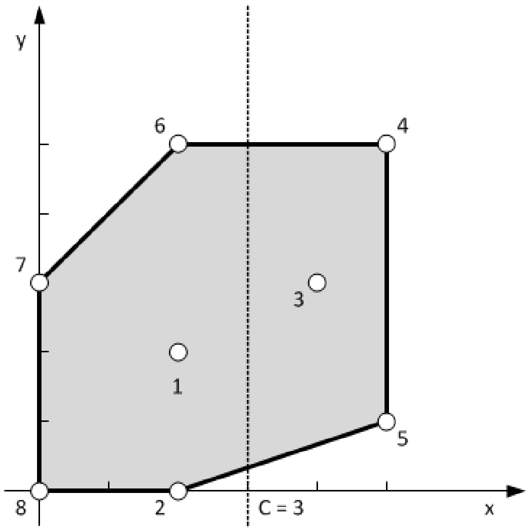


Figure 1. Point and corresponding convex hull for points
$X = (2,2,4,5,5,2,0,0)$ and $Y = (2,0,3,5,1,5,3,0)$ and value $C = 3$

Implementation and complexities

The described idea can be implemented easily: find the convex hull for the given points and find where the line $x = C$ intersects it. Finding the convex hull has the complexity of $O(n \log n)$. Finding the intersections is linear, because we only need to check for consecutive vertices of the convex hull. This leads to the final complexity of $O(n \log n)$.

Problem D: Interval Graph

Author: ***Andreja Ilić*** *Implementation and analysis:* ***Milan Novaković***

Statement:

For a set of closed intervals on real line, one the can construct an *interval graph*. Represent each interval with a different graph vertex and connect two vertices if and only if two corresponding intervals have common points.

Does a the given tree represent an *interval graph* for some set of intervals?

Input:

The first line contains positive integer n $(1 \leq n \leq 1000000)$ — the number of nodes in a tree. The nodes are numbered by IDs: $0, 1, 2, \ldots, n-1$. The node 0 is the root node of the tree.
The next n lines describe children for all nodes.
Line i (each od n lines) lists all children of the node with ID i.
The first integer in the line is c_i, the number of child nodes of node i. The next c_i integers in the same line are IDs of those child nodes.

Output:

The output consists of one line:

- "yes" (without quotes) if the given tree represents an interval graph
- "no" if it doesn't

Example input:

```
3
1 2
0
1 1
```

Example output:

```
yes
```

Example input:

```
7
31 2 3
14
15
16
0
0
0
```

Example output:

```
no
```

Time and memory limit: 3s / 64MB

Solution and analysis:

First, note that no three intervals can have a common point. If that were the case, the interval graph would have a triangle and wouldn't be a tree.
Now consider one interval and all intervals that have common points with it.
The intervals that are nested in that interval cannot therefore have any more common points with other intervals. They generate only one edge in the interval graph.
Intervals that are not nested contain one or both end points of the interval we are considering, therefore we can have no more than two intervals that have common points with considered interval and are not nested in it.

So, if we prune all one-edge sub-graphs corresponding to nested intervals, each vertex in the remaining graph can have at most degree two. In other words, the graph on Figure 1. can't be a sub-graph of the pruned graph.

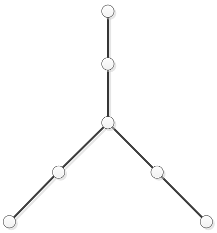


Figure 1. Forbidden structure for interval tree

We can see that if the tree satisfies this property, it is an interval graph. In the pruned graph every edge has degree one or two (if the graph is connected), therefore it is just a sequence of edges. We can therefore construct a sequence of intervals that correspond to this graph, and include nested intervals for additional one-edge sub-graphs that were pruned.

Conclusion is that not having the graph on Figure 1. for a sub-graph is a necessary and sufficient condition for the tree to be an interval graph.

Interval trees are a very important subclass of intersection graphs and perfect graphs. The generalization of above statement is a famous result of **Lekkerkerker and Boland** given below:

Theorem. A graph is an interval graph if and only if it contains none of the graphs shown in Figure 2. as an induced sub-graph.

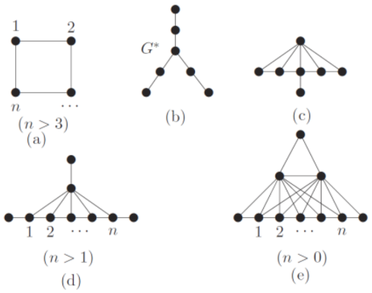


Figure 2. Forbidden structures for interval graphs

Implementation:

For the nodes on the first two levels, the described condition is equivalent to not having more than two sub-trees of depth one or more.
For nodes on deeper levels, the described condition is equivalent to not having more than one sub-tree of depth one or more.
This check could be easily done by **depth-first search** in linear time.
For simplicity, this could be broken into the following steps:

- Traverse tree and for each node calculate maximal depth of the sub-tree under it;
- Traverse tree and for each node calculate the number of children sub-trees with depth one or more;
- Traverse tree and for each node, check the condition taking in consideration the level of the node.

These steps could be done in one tree traversal.

Since depth of the tree can be up to 1,000,000, recursive DFS cannot be used due to stack limitations. Iterative DFS is not much harder to implement. For techniques on how to refactor recursion to iteration, a good resource is ***The Art of Computer Programming***.

Complexity

Time and memory complexity for DFS are $O(n)$and $O(\log(n))$ respectively, but storing the tree structure requires $O(n)$ memory. Overall, both memory and time complexity are linear: $O(n)$.

Problem E: Nice Subsequence

Author: ***Andreja Ilić*** *Implementation and analysis:* ***Dimitrije Filipović*** ***Andrija Jovanović***

Statement:

Given an array a of n integers, find the longest *nice* subsequence of consecutive elements.

The subsequence $a[i], \ldots, a[j]$, $i < j$, is *nice* if

a) $a[i] \leq a[j]$
b) $a[i] \leq a[k] \leq a[j]$, for all $k \in [i, j]$

Input:

First line contains one positive integer n ($2 \leq n \leq 1{,}000{,}000$), where n is the number of elements in the given array. Each of the next n lines contains one integer which represents an element of the array. Elements are in range $[0, 2 \times 10^9]$.

Output:

The output consists of one integer number:

- "-1" (without quotes) if *nice* subsequence doesn't exist
- Length of the longest *nice* subsequence

Example input:	**Example output:**
6	5
1	
3	
4	
2	
5	
0	

Time and memory limit: 3s / 64MB

Implementation and analysis:

First, we can see that a nice subsequence doesn't exist if the array is monotonically decreasing.

A naïve solution would be to find the longest nice subsequence ending with each index j and then find the longest of those, but that's too slow. We need to somehow use the information we have already obtained to speed up the search. For that purpose we will create a stack of nice subsequences we have obtained (we will call it S). It will initially be empty.

We traverse the array, starting from the right. We want to keep some properties of S invariant:

- All subsequences on the stack will either be nice, or have length 1.
- The subsequences on the stack will always be sorted so that their left boundary values (minimums) are decreasing (the largest value is at the top), and their right boundary values (maximums) are increasing (the smallest value is at the top).
- The subsequences will be sorted by their left boundary and they will not overlap each other.

We will always keep a "current" monotone subsequence, which we will denote c, and as we traverse the array from end to beginning, as long as the values of the elements are decreasing we can keep adding them to c. When we arrive to an element that breaks monotonicity (it is larger than its neighbor on the right), we want to push c to the stack (monotone sequences are nice by definition) - but first we have to perform some operations in order to ensure that S will continue to have the desired properties. Namely:

If $\min(c) \leq \min(top(S))$ and $\max(c) \leq max(top(S))$, the subsequence which spans from the left boundary of c to the right boundary of $top(S)$ is also nice, so we can expand c to match this subsequence and pop from S.

If $\max(c) > \max(top(S))$, we can discard $top(S)$, because it means that any subsequence we find in the future cannot be nice if it stretches further than the right boundary of c.

Either of these steps can be repeated several times. Finally, when $min(c) > \min(top(S))$ and $\max(c) \leq max(top(S))$, we push c to the stack and continue the traversal.

It is not hard to check that the properties of S we have highlighted will continue to hold after any of these steps are performed. Of course, we will keep a variable holding the best result we have found so far, and if we come across a nice subsequence longer than that value during any of these steps, we update the result.

Let's now try to sketch a proof that, after the entire array is processed, we will have found the correct result. We are only looking at subsequences on the stack, and we know that they will always be nice, so we will obviously never return a result larger than the correct one. What remains to be shown is that the longest subsequence will always be found by this algorithm.

If the longest nice subsequence is the subsequence $[i, j]$, we know that $a[i-1] > a[i]$ (or i is the first element) and $a[j] > a[j+1]$ (or j is the last element). So, both $a[i-1]$ and $a[j]$ will cause breaks in monotonicity, although in general they won't be in the same monotone subsequence. This means that we need to make sure that the sequence starting with i (s_i) will eventually merge with the sequence ending with j (s_j). But that is simple: $\min(s_i) \leq \min(top(S))$ because otherwise $[i, j]$ would not be nice at all, so the subsequences between s_i and s_j on the stack will either get merged into s_i or be discarded. Finally, when s_j becomes the top of S, s_i will merge with it, because, again, $[i, j]$ being nice implies that $\max(s_i) < \max(s_j)$. This means that $[i, j]$ will definitely be processed at some point, which means that the proof is finished.

Complexity:

Since the number of monotone subsequences in the array cannot be larger than n, the main part of the algorithm essentially consists of $O(n)$ "push" and $O(n)$ "pop" operations on the stack, making the overall time complexity of the solution linear.

Buxkdop F: Panuql

Authors: ***Milan Novaković,***
Andreja Ilić

Implementation and analysis: ***Milan Novaković***

Tnanopozn:

A boufoin panuql qt a panuql jqnc qznomou odopoznt qz jcqic oaic njx zoqmckxuqzm odopoznt auo uodanqyodw buqpo (ix-buqpo), azv nco aktxdrno yadro xf oaic odopozn qt muoanou ncaz xzo. Oaic odopozn cat rb nx fxru zoqmckxut.

Wxr auo mqyoz a panuql $A_{m \times n}$ jqnc qznomou odopoznt $a_{ij}, 1 \le i \le m, 1 \le j \le n$.

Icoie qf ncouo olqtnt a boufoin panuql $B_{m \times n}$ jcouo b_{ij} vqyqvot a_{ij} fxu oyouw $1 \le i \le m, 1 \le j \le n$.

Qzbrn:

Nco fqutn dqzo ixznaqzt bxtqnqyo qznomout m, n $(1 \le m, n \le 80)$ — nco vqpoztqxzt xf nco panuql $A_{m \times n}$. Oaic xf nco zoln m dqzot ixznaqzt a tohrozio xf n qznomout tobauanov kw tbaiot, uobuotoznqzm odopoznt a_{ij} $(2 \le |a_{ij}| \le 1000)$ xf nco panuql $A_{m \times n}$.

Xrnbrn:

Xrnbrn ixztqtnt xf xzo vqmqn: :

- "1" (jqncxrn hrxnot) qf a boufoin panuql olqtnt
- "0" qf qn vxotz'n olqtn

Olapbdo qzbrn:	**Olapbdo xrnbrn:**

2 2
6 4
10 9
	1

Olapbdo qzbrn:	**Olapbdo xrnbrn:**
1 3 4 6 9	0

DON'T PANIC ☺

Nqpo azv popxuw dqpqn: 0.5t / 64PK

Solution and analysis:

The statement of this problem was put through a cipher and presented to the competitors in encrypted form, which they had to decipher before they could start solving the actual problem.

Since the other eight problems were unencrypted and the texts had the same basic shape, starting by trying to compare the ciphertext with them was a good idea. It is noticeable that certain words repeat multiple times, and looking at the other problems reveals that they probably correspond to certain important phrases, for example "input", "output", "integer", with unchanged number of letters in a word. This implies that the cipher is a simple substitution cipher, and also reveals the encrypted values for many of the letters.

Going in this direction starts producing text that is already somewhat intelligible, so we are encouraged to continue: we know where keywords such as "statement" and "problem" are located, and we can guess the remaining letters in frequently occurring words: "the", "line". The rest is easy: most words will have only one of two letters left encrypted, and simple common sense should be enough to finish the job. Also, a cool thing is that when you translate “DON’T PANIC” you get “LET’ MATCH”.

Translated problem looks like this:

Statement:

A perfect matrix is a matrix with integer elements in which each two neighboring elements are relatively prime (co-prime), and the absolute value of each element is greater than one. Each element has up to four neighbors.

You are given a matrix $\mathrm{A}_{\mathrm{m}\times\mathrm{n}}$ with integer elements $\mathrm{a}_{\mathrm{ij}}, 1 \leq \mathrm{i} \leq \mathrm{m}, 1 \leq \mathrm{j} \leq \mathrm{n}$.

Check if there exists a perfect matrix $\mathrm{B}_{\mathrm{m}\times\mathrm{n}}$ where b_{ij} divides a_{ij} for every $1 \leq \mathrm{i} \leq \mathrm{m}, 1 \leq \mathrm{j} \leq \mathrm{n}$.

Input:

The first line contains positive integers m, n $(1 \leq \mathrm{m}, \mathrm{n} \leq 80)$ — the dimensions of the matrix $\mathrm{A}_{\mathrm{m}\times\mathrm{n}}$. Each of the next m lines contains a sequence of n integers separated by spaces, representing elements a_{ij} $(2 \leq |\mathrm{a}_{\mathrm{ij}}| \leq 1000)$ of the matrix $\mathrm{A}_{\mathrm{m}\times\mathrm{n}}$.

Output:

Output consists of one digit:

- “1” (without quotes) if a perfect matrix exists
- “0” if it doesn’t exist

The condition that neighboring elements are relatively prime can be stated in the form that neighboring elements cannot have the same prime factors. Since b_{ij} divides a_{ij}, candidates for prime factors for b_{ij} are prime factors of a_{ij}. Clearly, if there exists a perfect matrix, we can transform it to another perfect matrix where all b_{ij} are prime numbers, by omitting all but one prime factors of b_{ij}. Resulting perfect matrix has all prime elements and neighboring elements are different. Therefore, it is sufficient to check if a perfect matrix with prime elements and described properties exists to get the final answer.

In view of these conclusions, problem can be restated in the following way: For every element in the original matrix A, pick one of its prime factors, so that two neighboring elements have different factors picked. This is closely related to the graph coloring problem, which is NP, so backtrack is the way to go.

Implementation:

The most naive backtrack implementation is too slow. A few improvements can be made. Prime numbers up to 1,000 can be pre-computed and included in the source file.

All numbers up to 1,000 can also be factored to prime factors offline and included in the source file. If some element a_{ij} has a unique prime divisor among its neighbors, this prime divisor can be picked for the perfect matrix.

If some element a_{ij} has only one prime divisor, this prime divisor cannot be picked from its neighbors.

The last observation is the crucial one for speeding up the backtrack algorithm. What should be noted here is that once a prime divisor is removed from the list of possible primes for all neighboring elements, the process can continue if one of the neighboring elements is left with only one prime divisor. This can propagate as long as there are changes made.

Complexity:

Time complexity is exponential. Memory complexity in most implementations shouldn't be larger than $O(mnp)$, where p is the number of different prime factors.

Problem G: Operations

Author: ***Mladen Radojević*** *Implementation and analysis:* ***Mladen Radojević***

Statement:

You are given an array of n characters in the form of $DSDSDS \dots DSD$ (n is odd) where D ∈ {'0'…'9'} and S ∈ {'+','-','=','>','<'}.

Find out the maximal number of non-overlapping correct expressions (a correct expression is a substring of the given string which starts and ends with a digit, has exactly one comparison operator ('=' or '>' or '<'), and is mathematically correct).

Input:

The first line contains a positive integer n (n is odd, and $n < 5{,}000{,}000$). The next line consists of n characters in the form described above (without any spaces between characters).

Output:

The output consists of the integer number which represents the maximal number of non overlapping correct expressions.

Example input:	Example output:
7 7-5<3=5	1
Example input:	**Example output:**
11 2+5<6-4<5=3	2

Time and memory limit: 3s / 64MB

Implementation and analysis:

It is not hard to check that the maximal number of correct expressions can be achieved using the following algorithm (Greedy algorithm):

- Find the first comparison operator for which we can obtain the correct expression, take the correct expression which contains that comparison operator and for which the rightmost character has the smallest index in the original array.
- Start looking for new correct expression from the position of first digit after the previously found correct expression.

Checking if it exists and finding the optimal correct expression (optimal in the meaning described above) that contains some fixed comparison operator and that starts from some particular position (on the left

side of comparison operator) can be done in the following way:
First we calculate all possible values on the left side of the comparison operator. Then we scan digit by digit on the right side of the comparison operator, calculating the value of the expression on the right side by considering the digit and checking if that value, with the comparison operator and any value from the left side, gives a true statement.

Checking if a value from right side, the comparison operator and any value from the left give a true statement can be done by sorting values from the left side in non-decreasing order. Then, if the comparison operator is ‘=’, use binary search to check if that value exists in a set from left. If the comparison operator is ‘<’, check if the value from the right is greater than the first value from left in the sorted array, and in case when the comparison operator is ‘>’, check if the value from the right is less than the last value from the left in the sorted array. Alternatively, since the minimal possible value on the left is $-9 \cdot (n-1)/2$ and the maximal is $9 \cdot (n-1)/2$, another approach would be to have an array of $9 \cdot (n-1) + 1$ elements, so that for each value calculated on the left we join one element of the array, and while filling that array we can calculate the minimal and maximal value from all those that we were using in filling, so we can easily check if a value from the right exists on the left, just by looking in the corresponding place in the new array. If the value from the right is greater than some value from the left we can check using min, and whether the value from the right is less than some value from left we can check using max. We can also use min and max for initializing array for each new comparison operator.

Example: 3+8-6=2+4<3+2

First we calculate values on the left side of ‘=’. These are 6,2,5. Then we go from ‘=’ to the right. The first potential value is 2. Check if it is found in the set from the left. If it is, meaning that we found one correct expression, start looking for a new one from ‘4’. The next comparison operator is ‘<’. The only value on its left side is 4. The first digit on the right is 3. Considering that 3 is not greater than any value from the left (in this case just 4) we continue. The next digit is 2, so the next potential value is 5. Since 5 is greater than 4, we do have a new correct expression. We get to the end of the array, so the maximal number of non-overlapping correct expressions is two.

Complexity:

It is obvious that the complexity of the solution which uses a sorted array is $O(n\log(n))$ and that the complexity of the second solution is $O(n)$.

Memory complexity is $O(n)$ in either case.

Problem H: Travel ‘n’ sleep

Author: ***Milan Novaković*** *Implementation and analysis:* ***Andrija Jovanović***

Statement:
You are the manager of a company and you want to send some of your employees to a big company meeting, which starts t days from now. The city where the meeting will be held is very far away from your headquarters, so they will have to travel for a couple of days, passing through some other cities and making pauses to sleep and rest during the journey. You have a map that assigns numbers between 1 and n to the cities and shows which of these cities have direct routes between each other. All the employees start from your headquarters (city 1) on the first day. On any given day, each employee can choose either to travel between two connected cities or to stay where he is and rest, and they all have to reach the meeting place (city n) and must not be late for the meeting.

There is just one small problem: your employees hate each other, so you can never allow two or more of them to be in the same city at the same time (except at the start and the end of their journeys, of course). It is allowed for someone to enter a city on the same day when someone else is leaving, however. You kind of hate all of them too, so you don’t want to allow anyone to stay in your headquarters or to return there during the journey.

The meeting is quite important, so you would like to send as many people there as possible, and now you want to calculate exactly how many is that.

Input:
The first line contains three numbers, n $(2 \leq n \leq 50)$, t $(1 \leq t \leq 30)$ and m $(1 \leq m \leq 500)$. Each of the following m lines contains two different integers, the numbers of connected towns. All routes are two-way.

Output:
The output consists of exactly one non-negative integer, the maximal number of people that can reach town n from town 1 in t or less days.

Example input:

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

Example output:

```
2
```

Explanation:
On the first day, the first person can go to city 2 and the second can go to city 3, and they will both reach city 4 on the second day.

Time and memory limit: 1s / 64MB

Implementation and analysis:

Looking at the problem statement carefully, it is noticeable that this problem is fairly similar to the problem of finding maximum flow in a graph. There are a few difficulties, however. Paths in our graph depend on a time component, while maximum flow assumes that edges have constant capacities. Also, we need to make sure that only one path can include any single vertex at a point in time. So what we need to do is try to find a way to transform our graph into one that is more suited to the max-flow constraints.

First, we will transform every vertex v of the original graph (except one!) into t vertices of the form (v, t_i), with the idea that one vertex of the new graph will represent a single point in space and time. With this, the original graph is turned into a graph in which we always know whether a particular route is available or not. To be more precise, for each edge uv in the original graph, the new graph will have directed edges from vertex (u, t_i) to $(v, t_i + 1)$ and from (v, t_i) to $(u, t_i + 1)$ for t_i between 0 and $t - 1$. We also have to account for the possibility of staying in the same city on a particular day, so every vertex (u, t_i) should also have an edge towards $(u, t_i + 1)$ for $t_i < t$. We will not do this for city 1, however, in order to eliminate staying in this city or returning to it later.

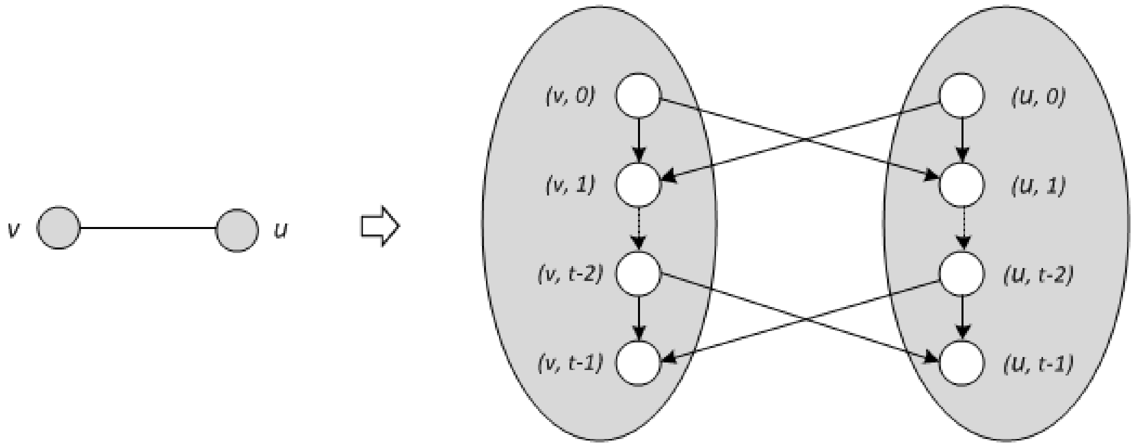


Figure 1. Vertex transformation with time parameter

Now we have a graph we can traverse without paying special attention to the time component. The other problem is ensuring that no two people can be in the same city on the same day, and we can do this using another easy trick to transform the graph: we split every vertex $w = (v, t_i)$ into an “in” vertex w_{in} and an “out” vertex w_{out}, and add an edge of capacity 1 from w_{in} to w_{out}. All edges that went to v should be redirected to w_{in}, while all edges that went out from w should now start from w_{out}.

The only thing remaining is to define the starting and the finishing vertex (the source and the sink) for the flow in our new graph. The source is easy: it is the out vertex corresponding to city 1, where everyone has to start. We do not have a single finishing vertex, however - all vertices corresponding to city n are valid finishing points. We will get around this by adding yet another (!) vertex, v_{sink}, and adding edges of unlimited capacity from all vertices $(v, t_i)_{in}$ to v_{sink}. (If “unlimited” is a problem for the computer to understand, any relatively large number will do.) It is now ensured that the solution to the problem is the maximum flow of the transformed graph.

Complexity:

The constraints are such that most standard algorithms for finding maximum flow will work, so feel free to pick your favorite one. For example, the Edmonds–Karp algorithm is relatively easy to implement, and its complexity is $O(V \cdot E^2)$ for a graph with V vertices and E edges. The number of vertices in the transformed graph is approximately $V = 2nt$, while the number of edges is approximately $E = (m + n) \cdot t$. This looks like bad news, but a closer look at the algorithm reveals that it consists of iterating breadth-first searches ($O(E)$ time), and that each iteration augments the flow. In the general case the number of iterations can be $O(V \cdot E)$, but here it is easy to see that the flow can never be larger than $n - 1$, since only $n - 1$ people have a city to go to on any given day. This means that the overall time complexity is actually $O(E \cdot n)$, which reduces to $O(mnt)$, and that should be well within the time limit. The space complexity primarily depends on the size of the transformed graph, which, if we use lists of edges for storage, is $O(V + E) = O\big((m + 3n) \cdot t\big)$.

Problem I: Queen

Author: ***Milan Vugdelija*** *Implementation and analysis:* ***Milan Vugdelija***

Statement:
Compute how many squares on average does a queen attack on a generalized chess board $n \times n$.

The queen attacks a square if it is on the same row, column or diagonal. For example, the queen denoted by the letter Q in the image bellow attacks 17 squares marked with dots:

	○		○		
○	○	○			
○	Q	○	○	○	○
○	○	○			
	○		○		
	○			○	

Input:
The first line contains positive integer n ($1 \leq n \leq 1{,}000{,}000$), the number of lines and columns of a board.

Output:
The output consists of one real number rounded to exactly three decimal places, the average number of fields attacked by a queen.

Example input:
3

Example output:
6.222

Time and memory limit: 0.5s / 64MB

Solution and analysis:

From each of $n \cdot n$ fields, a queen attacks $n - 1$ fields in its row, and $n - 1$ in its column.
Let's now count the fields on the same up-left to down-right diagonal. Counting diagonal by diagonal, we get

$$D = 2 \cdot 1 + 3 \cdot 2 + \cdots + n(n-1) + (n-1)(n-2) + \cdots + 3 \cdot 1 + \ 2 \cdot 1$$

For up-right to down-left diagonals we obviously obtain the same result. This gives the total number of fields that are attacked from all positions of the queen:

$$S = 2D + n^2(2n-2)$$

After summation, we obtain

$$S = \frac{10n^3 - 12n^2 \ + 2\,n}{3}$$

and the average number of attacked fields is

$$M = \frac{S}{n^2} = \frac{10n - 12 \ + \frac{2}{n}}{3}$$

Summation can be done by using math, or a computer program. In the latter case, time complexity will be linear (instead of constant) and care should be taken of overflow / precision.

Implementation:

Trivial: just read n and write $\frac{10n-12 \ + \frac{2}{n}}{3}$.

Complexity

As mentioned before, time complexity is $O(1)$ if computation is done mathematically, and $O(n)$ if computation is done programmatically.

Memory complexity is $O(1)$ in any case.

Test data:

The case $n = 1$ is an interesting example, so it should be included. Other tests should include odd and even numbers, as well as small and big numbers, to check different cases, time complexity and computation accuracy.

Qualifications

BubbleCup continued to increase in popularity in this year's edition. A total of 44 teams were involved and solved at least one problem, representing the highest turnout in its brief history. The tournament continued on the path of slowly becoming genuinely international, with multiple teams from Serbia, Croatia, Bosnia and Herzegovina, Macedonia and Romania all involved. The majority of teams consisted of university students but many younger teams competed as well, and some of them achieved very good results - six high school teams managed to pass through to the finals.

The qualifications were split into two rounds, with ten problems in each round and 25 days for the contestants to solve them. The first round lasted throughout April, and teams earned one point for each successfully solved problem. The second round was in May, and problems in this round were worth two points each (a rule change from the previous years).

The problems for both rounds were chosen from the publicly available archives at the Timus Online Judge site (acm.timus.ru). The first round is designed to be easier, and numbers confirm that - the most difficult problem according to the statistics (Mouse) was solved 17 times, and only one other problem was solved less than 30 times. The second round increased the difficulty considerably, resulting in one problem that no team managed to solve (Cockroach Race), while four other problems were solved 10 times or less.

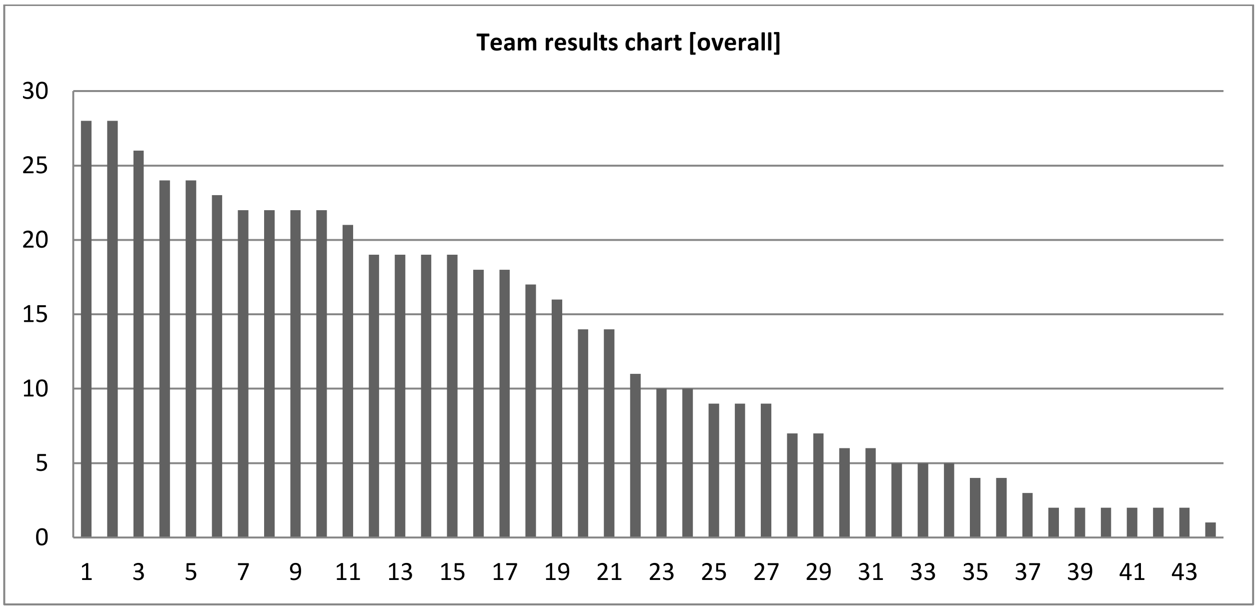
Num	Problem name	ID	Accepted solutions
01	Like Comparisons	1177	32
02	Mouse	1199	17
03	Asteroid Landing	1232	29
04	Evacuation Plan	1237	33
05	Bus Routes	1434	46
06	Brainfuck	1552	34
07	Dean's debts	1580	44
08	Pharaohs' Secrets	1584	31
09	Vasya Ferrari	1666	45
10	The Most Complex Number	1748	54

Table 1. Statistics for Round 1

Num	Problem name	ID	Accepted solutions
01	Cockroach Race	1369	00
02	Lights	1464	10
03	Fat hobbits	1533	21
04	Aztec Treasure	1594	10
05	Abstractionism to the People	1649	04
06	The Hobbit or Three and Back Again 2	1663	16
07	Asterisk	1670	22
08	Mortal Kombat	1676	22
09	Sniper shot	1697	09
10	Periodic sum	1749	25

Table 2. Statistics for Round 2

The fifteen teams with the highest number of points qualified for the finals. In another rule change from the previous contests, the results in the qualifications continue to matter - each team gets bonus time points in the finals based on their number of points in the qualifying rounds.



The explanations of the solutions for all 20 problems are provided in this booklet. They were written by a number of different people, some by contestants and some by MDCS BubbleTeam, and you should note that they are not official - we cannot guarantee that all of them are accurate in general. (Still, a correct implementation should pass all of the test cases on the Timus site.) Important algorithms and data structures are marked **bold**. It is assumed that you are familiar with them - if you are not, you can easily find information about them in literature or on the Internet.

The organizers would like to express their gratitude to everyone who participated in writing the solutions.

Problem R1 01: Like Comparisons (ID: 1177)

Time Limit: 1.0 second

Memory Limit: 16 MB

Development team of new DBMS asks you to write subroutine for the ‘like’ operator.
‘Like’ operator works as following. It returns true if text string matches specified template. Template is a text string containing any symbols or following special sequences:

%	matches any number of any characters
_	matches any single character
[c1-c2]	matches any single character in the range c1-c2
[c1c2c3…cN]	matches any single character of the set {c1,c2,c3,…,cN}
[^c1-c2]	matches any single character not in the range c1-c2
[^c1c2c3…cN]	matches any single character not in the set {c1,c2,c3,…,cN}

Input

First line contains number of tests N ≤ 1000. Next N lines contain comparisons in the following format:

```
'string' like 'template'
```

String or template may contain any symbols with ASCII codes 32-255. Inner entrance of apostrophe symbol (ASCII 39) into string or template is encoded by double apostrophe symbol. Maximal length of string or template is 100 symbols.

Output

For each of N comparisons output single 'YES' or 'NO' at a line.

Sample

input	output
15 'abcde' like 'a' 'abcde' like 'a%' 'abcde' like '%a' 'abcde' like 'b' 'abcde' like 'b%' 'abcde' like '%b' '25%' like '_5[%]' '_52' like '[_]5%' 'ab' like 'a[a-cdf]' 'ad' like 'a[a-cdf]' 'ab' like 'a[-acdf]' 'a-' like 'a[-acdf]' '[]' like '[[]]' '''''' like '_''' 'U' like '[^a-zA-Z0-9]'	NO YES NO NO NO NO YES YES YES YES NO YES YES YES NO

Solution:

The key to this problem lies in the fact that both strings which should be compared are short (up to 100 symbols). Therefore we can use **dynamic programming** to solve this problem. We will create the matrix m, the size of which can be up to 100x100B = 10kB. By 'character' we are going to imply the character for the given string or special sequence which is described with a symbol at position i in the second string. Matrix element $m[i][j]$ will indicate if the first i characters of the first string A are 'like' the first j characters of the second string B (template). Having this in mind we can create the recurrent relation to populate the entire matrix. The recurrent relation is presented below:

$$m[i][j] = \begin{array}{ll} true & , i = 0 \text{ and } j = 0 \\ false & , (i = 0 \text{ or } j = 0) \text{ and } (i + j > 0) \\ true & , match(A[i], B[j]) \text{ and } m\ [i-1][j-1] \\ true & , equal(B[j], \%) \text{ and } (m[i-1][j-1] \text{ or } m[i][j-1] \text{ or } m[i-1][j]) \\ false & , \text{in all other cases} \end{array}$$

The explanation for this formula is the following: if the character $A[i]$ is equal to the character $B[j]$, and if the first $i - 1$ characters of the string A are 'like' the first $j - 1$ characters of the string B, then the first i characters of the string A are 'like' the first j characters of the string B. If a character $B[j]$ is equal to %, there are three different cases: $m[i-1][j-1]$, $m\ [i][j-1]$ and $m[i-1][j]$. The factor $m[i-1][j-1]$ is there because the character % can match any character, including $A[i]$; $m[i][j-1]$ is there because the character % can be substituted with zero characters; finally, $m[i-1][j]$ is there because the character $B[j]$ = '%' is already included in substitution of the character $A[i]$ so the character B$[j]$ = '%' can also substitute the character A$[j-1]$. The element $m[length(A)[length(B)]$ will show if the string A is 'like' the string B.

The function $match$ checks if two given special sequences can produce/cover at least one common character. Since the strings can contain only ASCII characters with codes in range 32-255, this function can be implemented by producing the two character sets (up to 255 characters each), which are defined by the given special sequences, and check if these character sets have an intersection. During the implementation of this function, we should pay attention to the possibility that a special sequence can be defined recursively (like '[[]]', or '[%]').

The function $equals$ checks if the given special sequences can produce/cover exactly the same character sets. For example, % is equivalent to [%] or [_%].

The time complexity of this algorithm is $O(nm)$.

Solution by:

Name: ***Miloš Milovanović***
School: The Faculty of Electrical Engineering, Belgrade
E-mail: milmil@microsoft.com

Problem R1 02: Mouse (ID: 1199)

Time Limit: 2.0 second
Memory Limit: 16 MB

In the kitchen lives a mouse. There are also a cat and a piece of cheese in the kitchen. The coordinates of the cheese and the mouse are known, and the cat is sleeping. Finally, there is some furniture in the kitchen. The furniture is a set of convex polygons. The mouse wants to get to the cheese unnoticed. A point of the route is called dangerous if the distance to the nearest piece of furniture is greater than 10 cm. It is required to find the least dangerous route for the mouse, i.e., the route in which the sum of the lengths of dangerous segments is minimal.

Input

In the first line there are four numbers x_m, y_m, x_c, y_c separated with a space. They are the coordinates of the mouse (x_m, y_m) and of the cheese (x_c, y_c). In the second line there is the number of pieces of furniture N ($0 \le N \le 100$). The next N lines describe these pieces. Each description starts with the number of vertices of the corresponding polygon K ($3 \le K \le 10$), given in a separate line. Each of the next K lines contains two numbers, which are the coordinates of the corresponding vertex. It is known that the distance between any two points of different polygons is greater than 20 cm (so that it would be easier for the cat to catch the mouse). Neither the mouse nor the cheese are inside any of the polygons. All the coordinates are given in meters and have no more than three fractional digits. The absolute values of coordinates do not exceed 10^5.

Output

You should give the mouse's route in the form of a broken line. In the first line output the number of its vertices (including the initial and final ones). Then give the coordinates of the vertices, two numbers per line, accurate to 10^{-4}. Each segment of the broken line must be either entirely dangerous or entirely safe (with the possible exception of its endpoints). The broken line must contain no more than 1000 vertices.

Sample

input	output

1.0 1.5 0.0 1.5
1
4
0.0 0.0
0.0 1.0
1.0 1.0
1.0 0.0
	4
	1.0 1.5
	1.0 1.1
	0.0 1.1
	0.0 1.5

Solution:

The idea is basically simple (like most geometric problems), but problems may arise in the implementation of which special care should be taken. The solution consists of three steps:

1. Determination of the shortest distance between any two polygons;
2. Finding the shortest path in a graph;
3. Reconstruction of line segments of the final path.

The first step consists solely of geometry. It is convenient to define a class of two-dimensional vector to simplify implementation of the further calculations, whereby the points can be represented as radius-vectors from the origin. It provides easy manipulation of points, calculation of distances, application of operators such as cross and dot product, etc.

When determining the shortest distance between two polygons, it is necessary to determine the distance from a point (the polygon vertex) to the given line segment (the other polygon edge). For final path reconstruction, it is also necessary to determine the point on the line segment that is closest to a given point. Figure 1 shows (in two cases) the shortest distance d from the point P and line segment AB, and the point C on line segment, closest to the point P. (Shown distance is also the shortest distance of those two polygons.)

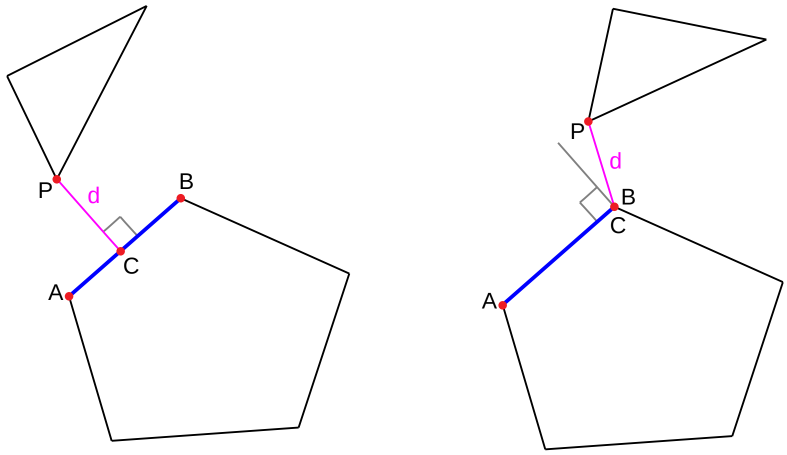


Figure 1. The shortest distance from a point to the line segment

The shortest distance between two polygons can be trivially determined by taking the minimum distance between all pairs (the first polygon vertex, the other polygon edge) and vice versa. The complexity of this approach is $O(n \cdot m)$, where n and m are the numbers of vertices of the first and second polygon, respectively. For this task, this is good enough. By using a somewhat more sophisticated algorithm (e.g. **rotating calipers**), the complexity of this step can be reduced to $O(m + n)$.

The second step involves graph theory. Each polygon is represented by a vertex of a complete weighted graph, where edge weights correspond to the shortest distances between the two polygons. For simplicity, the mouse and the cheese can also be treated as polygons (composed of only one point, or degenerate triangle with three equal vertices, or alike). In addition, the correction of edge weights is performed in a way to subtract $10\ cm$ for each edge side associated with actual polygon (subtraction is not performed for the mouse and the cheese). The shortest path in this graph is also the requested path with a minimal length of the "dangerous" route. A typical algorithm for determining the shortest path in a graph is **Dijkstra's algorithm**.

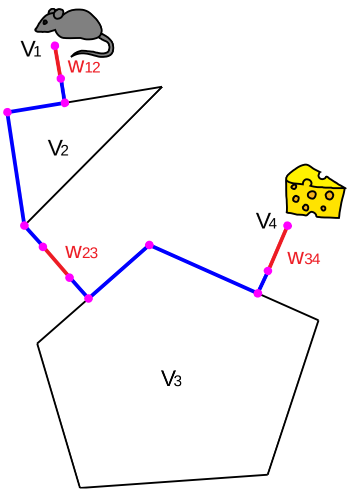


Figure 2. The final path reconstruction

Finally, the third step performs the reconstruction of linear segments of the required path (Figure 2). Based on the shortest path in the graph, the polygon traversal sequence is known. The specified path consists of the line segments along the edges of polygons, and the line segments that connect two polygons by the shortest route. It is necessary to make sure that the line segment which connects two polygons is broken down into three parts: two "safe" of $10\ cm$ length from the edge of each polygon, and a "dangerous" one, which connects the previous two.

Solution by:

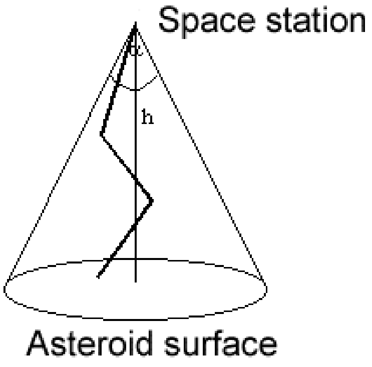
Name: ***Ognjen Dragoljević***

School: The Faculty of Electrical Engineering and Computing, University of Zagreb

E-mail: ognjen.dragoljevic@gmail.com

Problem R1 03: Asteroid Landing (ID: 1232)

Time Limit: 1.0 second
Memory Limit: 16 MB



A guided probe is launched from the space station located at the distance of h from the surface of a large asteroid. The probe must land at the asteroid. The probe moves straight forward for a fixed distance d, after that it receives a new command from the station. The command defines the new direction for the movement. Each movement of the probe must help it to get closer to the surface. The control signals from the station are transmitted only within a cone having a vertex angle of α.

So, the trajectory of the probe is a broken line with segments of equal length, which is lying inside the cone described above. The last segment of the trajectory must also be of length d, lie inside the transmission cone and end at the surface of the asteroid.

Your task is to determine if it is possible to perform the landing of the probe taking into consideration the above conditions. If the landing is possible, then find the trajectory of minimal length including the coordinates of the ends of each segment. The landing point must be found, too.

The coordinates of the points are Cartesian. Ox and Oy lie on the surface of the asteroid, and Oz passes through the space station.

Input

h ($0 < h < 100$), d ($h/1000 \le d \le 10*h$), α (the angle is in radians, $0.1 \le \alpha \le 3$). All numbers are float.

Output

n — the number of segments in the trajectory, or −1, if landing is impossible

x_1 y_1 z_1

x_2 y_2 z_2

…

x_n y_n z_n — the coordinates of the points where the probe receives control signals, and the landing point. All coordinates must be calculated to within 0.0001.

Sample

input	output

11 5 2	3
	0 3 7
	3 3 3
	3 -1 0

Solution:

The distance h of the asteroid from the surface can be written as

$$h = kd + r,$$

where k is a nonnegative integer. If $r = 0$, the asteroid can land on the surface in k movements by just moving down, i.e. in the direction of the surface. Otherwise, it will need $k + 1$ movements.

Let's find the maximum possible distance from the surface after i movements — $Z_{max}(i)$ for every i. We can see that maximal distance is achieved if the asteroid is moving as in Figure 1. Note that we are moving only in the xz plane, so y is always 0.

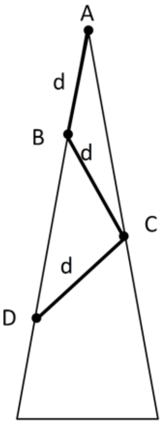
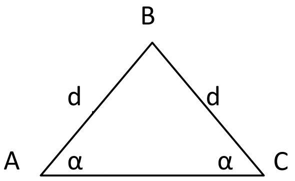
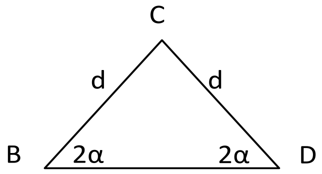


Figure 1. Asteroid moving

We know that $Z_{max}(0) = h$, and $Z_{max}(1) = h - d \cdot cos(\alpha/2)$. Observe the (isosceles) triangle formed by two consecutive movements and lateral surface of the cone (figures below show first two such triangles).





It can be shown that i-th such triangle has base angle of size $i \cdot \alpha$, and therefore the size of its base side is:

$$a_i = 2d \cdot cos(i\alpha).$$

Knowing that, we can compute $Z_{max}(i)$:

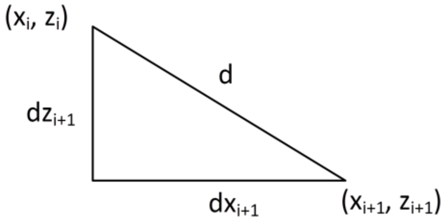
$$Z_{max}(i) = Z_{max}(i-2) - a_i \cdot cos(\alpha/2).$$

If $i\alpha > \pi$ (the sum of angles in a triangle), then such triangle is not formed and $Z_{max}(i) = Z_{max}(i-1)$ for such i.

Having determined Z_{max} we can find z_i coordinates of points where the probe receives the control signal.
If $Z_{max}(k+1) < 0$ then the solution does not exist.
If $Z_{max}(k+1) = 0$ then all z_i coordinates will equal $Z_{max}(i)$.
Else find $Z_{max}(i)$ such that $Z_{max}(\quad) \geq (k+1-i) \cdot d$ and $Z_{max}(i+1) < (k-i) \cdot d$. First i points will

have z_i coordinate equal $Z_{max}(i)$, and the rest will have $z_i = (k + 1 - i) \cdot d$ (the probe is moving down vertically in that part).

Having found z_i coordinates, we can find x_i coordinates.



$$dz_i = z_{i-1} - z_i$$
$$dx_i^2 = d^2 - dz_i^2$$

x_i coordinates are changing in alternating directions:

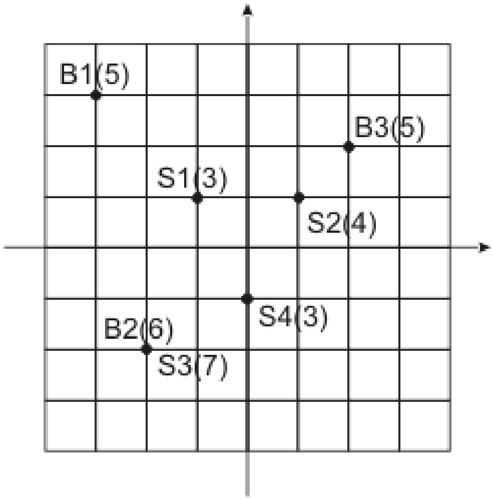
$$x_i = x_{i-1} + (-1)^i dx_i,$$

y_i is always zero as mentioned, so we have found our solution.

Solution by:
Name: ***Luka Donđivić***
School: The Faculty of Electrical Engineering and Computing, Zagreb
E-mail: ldondjivic@yahoo.com

Problem R1 04: Evacuation Plan (ID: 1237)

Time Limit: 1.0 second
Memory Limit: 16 MB



The City has a number of municipal buildings and a number of fallout shelters that were build specially to hide municipal workers in case of a nuclear war. Each fallout shelter has a limited capacity in terms of a number of people it can accommodate, and there's almost no excess capacity in The City's fallout shelters. Ideally, all workers from a given municipal building shall run to the nearest fallout shelter. However, this will lead to overcrowding of some fallout shelters, while others will be half-empty at the same time.

To address this problem, The City Council has developed a special evacuation plan. Instead of assigning every worker to a fallout shelter individually (which will be a huge amount of information to keep), they allocated fallout shelters to municipal buildings, listing the number of workers from every building that shall use a given fallout shelter, and left the task of individual assignments to the buildings' management. The plan takes into account a number of workers in every building - all of them are assigned to fallout shelters, and a limited capacity of each fallout shelter - every fallout shelter is assigned to no more workers then it can accommodate, though some fallout shelters may be not used completely.

The City Council claims that their evacuation plan is optimal, in the sense that it minimizes the total time to reach fallout shelters for all workers in The City, which is the sum for all workers of the time to go from the worker's municipal building to the fallout shelter assigned to this worker.

The City Mayor, well known for his constant confrontation with The City Council, does not buy their claim and hires you as an independent consultant to verify the evacuation plan. Your task is to either ensure that the evacuation plan is indeed optimal, or to prove otherwise by presenting another evacuation plan with the smaller total time to reach fallout shelters, thus clearly exposing The City Council's incompetence.

During initial requirements gathering phase of your project, you have found that The City is represented by a rectangular grid. The location of municipal buildings and fallout shelters is specified by two integer numbers and the time to go between municipal building at the location (X_i, Y_i) and the fallout shelter at the location (P_j, Q_j) is $D_{i,j} = |X_i - P_j| + |Y_i - Q_j| + 1$ minutes.

Input

The input consists of The City description and the evacuation plan description. The first line consists of two numbers N and M separated by a space. N ($1 \le N \le 100$) is a number of municipal buildings in The City (all municipal buildings are numbered from 1 to N). M ($1 \le M \le 100$) is a number of fallout shelters in The City (all fallout shelters are numbered from 1 to M).

The following N lines describe municipal buildings. Each line contains there integer numbers X_i, Y_i, and B_i separated by spaces, where X_i, Y_i ($-1000 \le X_i, Y_i \le 1000$) are the coordinates of the building, and B_i ($1 \le B_i \le 1000$) is the number of workers in this building.

The description of municipal buildings is followed by M lines that describe fallout shelters. Each line contains three integer numbers P_j, Q_j, and C_j separated by spaces, where P_i, Q_i ($-1000 \le P_j, Q_j \le 1000$) are the coordinates of the fallout shelter, and C_j ($1 \le C_j \le 1000$) is the capacity of this shelter.

The description of The City Council's evacuation plan follows on the next N lines. Each line represents an evacuation plan for a single building (in the order they are given in The City description). The evacuation plan of i^{th} municipal building consists of M integer numbers $E_{i,j}$ separated by spaces. $E_{i,j}$ ($0 \le E_{i,j} \le 1000$) is a number of workers that shall evacuate from the i^{th} municipal building to the j^{th} fallout shelter.
The plan is guaranteed to be valid. Namely, it calls for an evacuation of the exact number of workers that are actually working in any given municipal building according to The City description and does not exceed the capacity of any given fallout shelter.

Output

If The City Council's plan is optimal, then write the single word OPTIMAL. Otherwise, write the word SUBOPTIMAL on the first line, followed by N lines that describe your plan in the same format as in the input. Your plan need not be optimal itself, but must be valid and better than The City Council's one.

Sample

input	output

3 4
-3 3 5
-2 -2 6
2 2 5
-1 1 3
1 1 4
-2 -2 7
0 -1 3
3 1 1 0
0 0 6 0
0 3 0 2
	SUBOPTIMAL
	3 0 1 1
	0 0 6 0
	0 4 0 1

3 4
-3 3 5
-2 -2 6
2 2 5
-1 1 3
1 1 4
-2 -2 7
0 -1 3
3 0 1 1
0 0 6 0
0 4 0 1
	OPTIMAL

Solution:

This is another interesting graph theory problem. The question is how to send workers from buildings to shelters and make a better solution than the one given in the task (the new plan does not need to be optimal, but it must be valid).

We see that there are two types of shelters, the ones that are not completely filled up with workers and the ones that are. For the second type it is important that if we want to move a worker into the shelter, first we have to move one out.

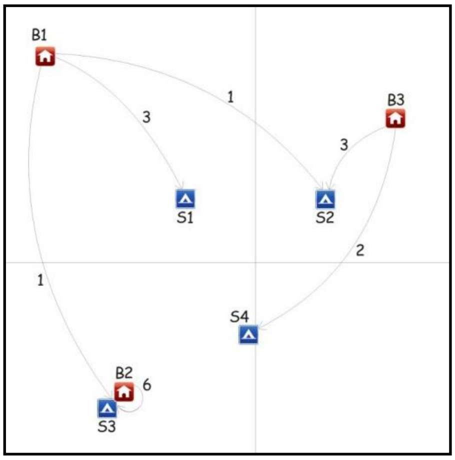


Figure 1. Example from the problem statement

Let's make a weighted directed graph$G = (V, E)$ in which nodes represent shelters, and for every non-empty shelter u there is an edge from it to every other shelter v.The weights of these edges are given by the following formula: $W_{u,v} = \min\{dist(x,u) - dist(x,v)\}$ for every building x which has a worker in shelter u, and where $dist(x,u)$ is the distance from building x to shelter u. This formula represents the minimal difference in the distances needed to transfer a worker from shelter u to shelter v.

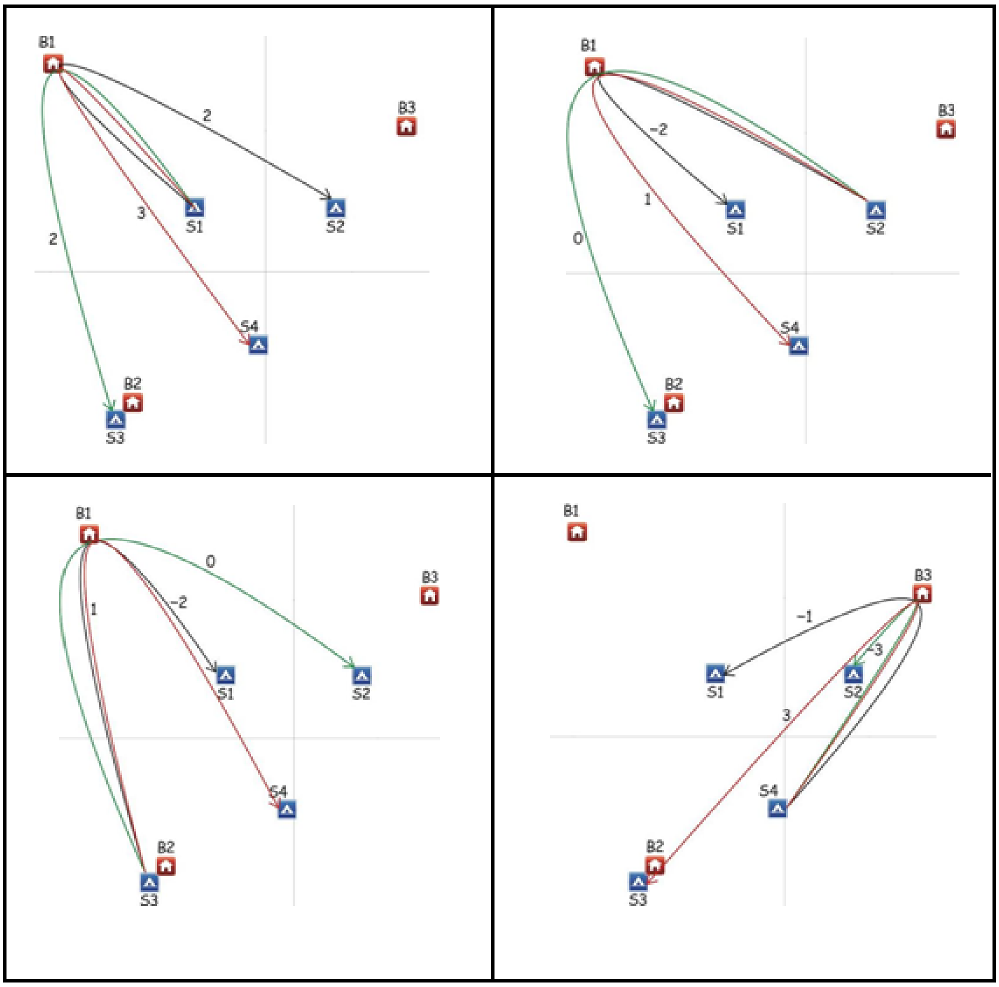


Figure 2. Described graph in the example

Notice that an edge can be negative, which represents improvement. Now, it is easy to see that the only possible improvement is either a negative cycle in this graph or a negative path which ends in a shelter that is not full. A negative cycle/path is one for which the sum of all its edges is negative. We get the solution by rotating workers along the path/cycle that we have found.

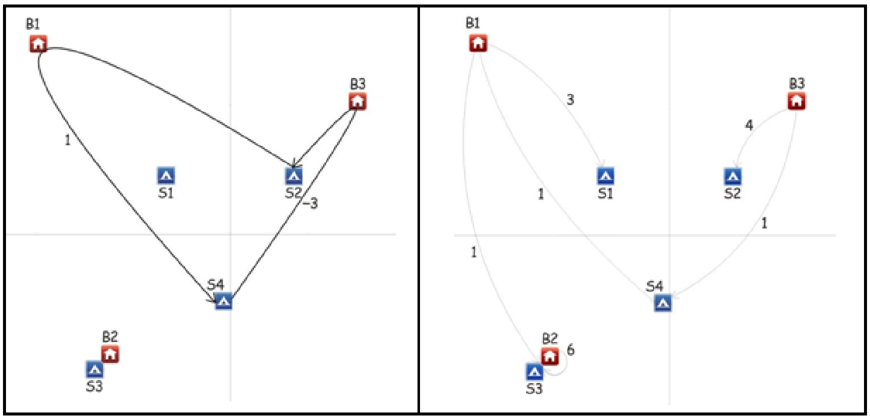


Figure 3. Solution for the example

The easiest way of finding a negative cycle or path is the **Bellman-Ford algorithm**, which has complexity of $O(|V||E|)$. Since the number of vertices is M and the number of edges is not bigger than M^2, the complexity of this algorithm is $O(M^3)$.

Solution by:

Name: Dušan Zdravković, Dimitrije Dimić, Stefan Stojanović

School: "SvetozarMarković" High School,Niš

E-mail: zdravkovicdusan@hotmail.com, dimke92@yahoo.com, dolarlord@gmail.com

Problem R1 05: Bus Routes (ID: 1434)

Time Limit: 3.0 second

Memory Limit: 32 MB

The Vasyuki University is holding an ACM contest. In order to help the participants make their stay in the town more comfortable, the organizers composed a scheme of Vasyuki's bus routes and attached it to the invitations together with other useful information.

The Petyuki University is also presented at the contest, but the funding of its team is rather limited. For the sake of economy, the Petyuki students decided to travel between different locations in Vasyuki using the most economical itineraries. They know that buses are the only kind of public transportation in Vasyuki. The price of a ticket is the same for all routes and equals one rouble regardless of the number of stops on the way. If a passenger changes buses, then he or she must buy a new ticket. And the Petyuki students are too lazy to walk. Anyway, it easier for them to write one more program than to walk an extra kilometer. At least, it's quicker.

And what about you? How long will it take you to write a program that determines the most economical itinerary between two bus stops?

P.S. It takes approximately 12 minutes to walk one kilometer.

Input

The first input line contains two numbers: the number of bus routes in Vasyuki N and the total number of bus stops M. The bus stops are assigned numbers from 1 to M. The following N lines contain descriptions of the routes. Each of these lines starts with the number k of stops of the corresponding route, and then k numbers indicating the stops are given ($1 \le N \le 1000$, $1 \le M \le 10^5$, there are in total not more than 200000 numbers in the N lines describing the routes). In the N+2nd line, the numbers A and B of the first and the last stops of the required itinerary are given (numbers A and B are never equal).

Output

If it is impossible to travel from A to B, then output −1. Otherwise, in the first line you should output the minimal amount of money (in roubles) needed for a one-person travel from A to B, and in the second line you should describe one of the most economical routes giving the list of stops where a passenger should change buses (including the stops A and B).

Sample

input	output

3 10
5 2 4 6 8 10
3 3 6 9
2 5 10
5 9
	3
	5 10 6 9

Solution:

This was a very interesting problem and proved to be one of the easiest. The problem can be approached in several ways – but they all contain a few graph search algorithms. The problem can be solved very elegantly, which will be shown here.

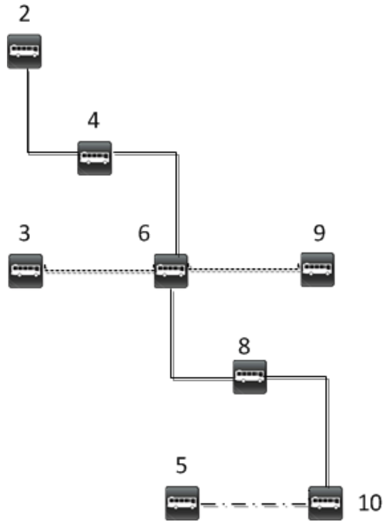


Figure 1. Example from the problem statement

The question is how to find a path between two bus stops with the smallest number of bus changes. Denote with $G = (V, E)$ graph with nodes corresponding to bus stops, and edges between consecutive bus stops in routes. We can look at bus routes as paths in graph G. Now we can generate a graph with bus stops as nodes and find the minimal path using **breadth-first search (BFS)**. This idea works, but the implementation can be a bit tricky, so we will try to implement it in a slightly different way. We are going to construct a new graph with vertices for both bus stops and routes. Now, for every route we are going to add edges to its stops. This is very cool, isn't it? The corresponding graph would look like this:

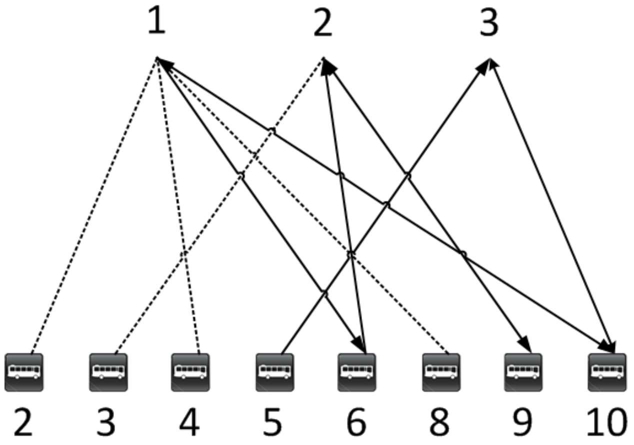


Figure 2.Generated graph for the example case

As we can see, this graph is **bipartite**. It has one nice feature – the distance between any two bus stops on the same route is two edges, so we can just find the shortest path between two given bus stops and these vertices are going to be bus stops where we have to change routes. The easiest way to implement this is to store routes as nodes with indexes $M+1, M+2, \dots, M+N$.

The number of edges in this graph is equal to the sum of lengths of all routes. The complexity of the BFS algorithm is linear on the number of edges, which brings us to the final complexity of $O(SUM\_OF\_ROUTES\_LEN)$, which is less than 200,000.

Solution by:
*Name***:** ***Andreja Ilić***
School: The Faculty of Mathematics and Sciences, Niš
*E-mail***:** *ilic_andrejko@yahoo.com*

Problem R1 06: Brainfuck (ID: 1552)

Time Limit: 2.0 second

Memory Limit: 64 MB

Chairman of "Horns and hoofs" company, Mr. Phunt, decided to start advertising campaign. First of all, he wants to install an indicator panel on the main square of the city that will show advertisements of the company. So he charged the manager of the company, Mr. Balaganov, to do this job. After analyzing offers of indicator panels, Balaganov ordered one at a price of only $19999.99. But when it was delivered, a little problem was found. The panel was programmable, but the instruction set of the processor was a subset of brainfuck language commands. The commands that processor was capable to execute were '>', '<', '+', '–' and '.', which are described in the table below. Moreover, this panel had very little memory for the program, so not every program typing a particular string will fit into memory. Now Balaganov wants to know the minimal program that will output the given string. But because he is not very good at programming, he asks you to solve this problem. The brainfuck program is a sequence of commands executed sequentially (there are some exceptions, but panel processor cannot execute such commands). The brainfuck machine has, besides the program, an array of 30000 byte cells initialized to zeros and a pointer into this array. The pointer is initialized to point to the leftmost byte of the array.

Command	Description
>	Increment the pointer (to point to the next cell to the right). If the pointer before increment points to the rightmost byte of the array, then after increment it points to the leftmost byte.
<	Decrement the pointer (to point to the next cell to the left). If the pointer before decrement points to the leftmost byte of the array, then after increment it points to the rightmost byte.
+	Increment (increase by one) the byte at the pointer. If the value of the cell before increment is 255 then it becomes 0.
–	Decrement (decrease by one) the byte at the pointer. If the value of the cell before decrement is 0 then it becomes 255.
.	Output the value of the byte at the pointer.

Input

Input has one line containing the string brainfuck program must output. Every character of the string is a small English letter ('a'–'z'). The length of the string is not greater than 50. You may assume that optimal program will not have to modify more than four memory cells.

Output

Input has one line containing the string brainfuck program must output. Every character of the string is a small English letter ('a'–'z'). The length of the string is not greater than 50. You may assume that optimal program will not have to modify more than four memory cells.

Sample

input	output
a	+++++++++++++++++++++++++++++++++++ +++++++++++++++++++++++++++++++++++ +++++++++++++++++++++++++++.

Hint

Please note that the sample output is divided into several lines only for convenience. In the real output whole program must be printed on a single line.

Solution:

Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy.
(Alan Perlis, from *Epigrams on Programming)*

If we look at the sentence in the task description, “You may assume that optimal program will not have to modify more than four memory cells”, we can conclude that these four cells are consecutive, and that the starting cell has to be one of them. That gives us four possibilities to begin with, and each of them is solved in the same fashion as follows.

The solution uses **dynamic programming**. Every state can be described with six integers: values of the four “critical” cells, the number of the cell (0…3) the pointer is currently at, and the position of the letter in the string which is to be printed next. To reduce the number of states in the memory, we can make the following observations. First, the values of the four cells are either 0 or ASCII values of the letters ‘a’ – ‘z’ after each letter is printed; this reduces the number of possible values in each cell from 256 to 27. Furthermore, we do not need the number of the cell the pointer is currently at, because if we know which letter was printed last, we simply find the cell which has the value of that letter. Therefore, the total number of states reduces to $27^4 \cdot 50 = 26{,}572{,}050$, which is still much more than the number of states that can actually be reached. For this reason, to implement the dynamic programming approach we use **recursion with memoization** - in this way we will process only the states that we can get to.

In the process of finding the best solution for a state in the recursion, our goal is to choose a cell which will contain the value of the next letter to be printed, so that the number of remaining brainfuck commands (moving the pointer to the chosen cell, changing the cell value and recursively solving the rest of the string from the obtained state) is minimal. In order to reconstruct the optimal sequence of commands, we can run a procedure which, knowing the minimal number of commands for each state (already calculated by the dynamic programming algorithm), simply finds this sequence.

Note:
The **brainfuck** programming language is an esoteric programming language (programming language designed as a test of the boundaries of computer programming language design, as a proof of concept, or as a joke) noted for its extreme minimalism. It is a Turing tar-pit, designed to challenge and amuse programmers, and is not suitable for practical use.

Solution by:
*Name**:** **Adrian Satja Kurdija***
School: 5th High School, Zagreb; Faculty of Mathematics and Sciences, Zagreb
E-mail: askurdija@gmail.com

Problem R1 07: Dean's Debts (ID: 1580)

Time Limit: 2.0 second

Memory Limit: 64 MB

N students of one university took part in the Yekaterinozavodsk training camp. When they returned home, it turned out that they had spent much of their own money for the tickets to Yekaterinozavosk and back, for their lodgings, food and registration fees. The students came to the dean of their department and asked him to compensate the costs of the trip. The dean listened to them carefully and gave some amounts of money (possibly different) to all of them. The next day two of these students came to the dean and told that the two of them had been given A_1 rubles less than they had spent jointly. On the next day, the situation repeated itself: a pair of students claimed that the dean owed them A_2 rubles. The situation repeated itself for a few days more. Finally, on the M-th day a pair of students told the dean that they two had spent together A_M rubles more than the dean had paid them. After that, the students lost any hope and stopped visiting the dean. Then the dean took the notes with the students' demands and decided to calculate how much he owed each of them. But it turned out to be not so easy!

Input

The first line contains integers N and M separated by a space (2 ≤ N ≤ 1000; 1 ≤ M ≤ 100000). The following M lines contain the demands of pairs of students who visited the dean. The (i + 1)-st line contains three integers separated by spaces: the numbers of two students who visited the dean on the i-th day and the amount of money A_i they asked for. The students are numbered from 1 to N. The number A_i is an integer in range from −10000 to 10000. A negative number means that the students got from the dean more that they spent. It is known that no pair of students visited the dean more than once.

Output

If the dean can determine uniquely how much money he owes each of the students, write these sums with two digits after the decimal point: in the i-th line output the amount he owes the i-th student. The numbers can be negative; this means that the student owes the dean (sometimes it happens!). If it is impossible to find these amounts, output "IMPOSSIBLE".

Sample

input	output

3 3
1 2 2
2 3 4
3 1 6
	2.00
	0.00
	4.00

4 3
1 2 2
1 3 4
1 4 6
	IMPOSSIBLE

Solution:

The problem, in essence, asks of us to solve a system of linear equations and print the solution if it is unique. It is less general than that, however, because our equations have a rigidly defined shape - they all have the form of $s_i + s_j = A_{ij}$. We will try to make use of that.

Let's start solving the system by hand. If we have (like in the example) $s_1 + s_2 = 2$, $s_2 + s_3 = 4$ and $s_3 + s_1 = 6$, we can subtract the first equation from the second to get $s_1 + (-s_3) = -2$. Then, adding the third equation to this, we get $s_1 + s_1 = 4$, and it is easy to go further from there.

Now we have to generalize this approach to get the solution. We will define a set of states (which represent equations) and rules for transitions between states (which simulate derivation). All equations we will need can be written as $\pm s_i \pm s_j = A_{ij}$. This can be represented by a state that specifies the left-hand side — the numbers i and j along with the coefficients in front of the variables, which are always $+1$ or -1 in this problem. Of course, the state will have the number A_{ij} associated with it as well. We can also note that, since negating the whole equation does not change anything, it is safe to assume that the first coefficient (in front of s_i) is always $+1$ and only remember the second one, halving the number of states.

We get the starting set of states from the input. (We can add some trivial states to this set as well: $s_i - s_i = 0$ for all i). If we are in a state corresponding to the equation $s_i + cs_j = A$, the rules of transition are the following:

- if $i = j$ and $c = 1$, we can calculate $s_i = A/2$
- if $i = j, c = -1$ and $A \neq 0$, we have reached a contradiction
- if we have the value for one of s_i or s_j, we can get the other one using simple arithmetic
- otherwise, for all k, if we have a state that corresponds, e.g., to the equation $s_j + c_k s_k = B$, we can transition to the state $s_i + (-cc_k)s_k$ with the right-hand side $A - cB$ (and similarly for other possibilities – when s_j is the second variable, or s_i is the variable we need to eliminate)

We can traverse the space of states using a variant of **depth-first search**, making sure to pass along all required values while we travel. We can get the following results:

- a contradiction occurred somewhere: just print out "IMPOSSIBLE"
- we got all the values: this is our solution, so print it out
- we did not get the values of all s_i: since the search is exhaustive, there is no other way that can get the values of variables we did not get, therefore print out "IMPOSSIBLE"

Since the number of states is limited to $2N^2 = 2 \cdot 10^6$, and we won't visit any state more than once, it follows that we have more than enough time for the algorithm to finish. The implementation should be relatively straightforward.

Solution by:
Name: ***Andrija Jovanović***
School: *School of Computing (RAF), Belgrade*
E-mail: *ja.andrija@gmail.com*

Problem R1 08: Pharaohs' Secrets (ID: 1584)

Time Limit: 1.0 second

Memory Limit: 64 MB

When programmer Alex was in Egypt, he not only swam in the Red Sea and went sightseeing, but also studied history. When Alex visited the place where an archeological dig of an ancient temple was carried out, an excavation worker complained to him that they had to drag very heavy statues from place to place every day. This was because some Egyptologist had read in an ancient papyrus that if the statues were arranged in a special order, then some ancient hiding-place would open. When the temple had been dug out, these statues had stood as soldiers, forming a rectangle. Some statues were identical, so there were several types of statues. They were to be arranged into a rectangle of the same dimensions on the same place with all rows and columns symmetric with respect to their middles. This meant that the statues standing in the same row or column at equal distances to its ends had to be of the same type.

Alex offered his help. He wants to find the way to transform the rectangle into a symmetric one by means of the minimal number of moves.

Input

The first line contains the dimensions of the rectangle n and m (2 ≤ n, m ≤ 20). These integers are even. Each of the next n lines contains m lowercase English letters. Each letter denotes the type of the statue that stands in the rectangle at this position.

Output

Output the minimal number of statues that should be moved in order to make a symmetric rectangle. It is guaranteed that this is possible.

Sample

input	output
4 4 abxa xyyb xyyx abba	2

Hint

The arrangement in the example can be transformed to a symmetric one in only two moves: first the statue of the type **x** from the upper row should be moved to the place in the rightmost column where there is the statue of the type **b**, and this statue then should moved to the place where the first statue stood. After all moves each place must be occupied by exactly one statue, but during the moving process there can be several statues at the same place.

Solution:

Let us enumerate rows and columns of the given rectangle top-bottom from 1 to n and left-right from 1 to m, and denote by (i,j) the position at the intersection of i-th row and j-th column. Suppose that a statue a is located at (i,j), $i \leq n/2$, $j \leq m/2$. If this rectangle is symmetric, then, by applying symmetry on i-th row and j-th column, it follows that statues of the same type (a) must be located at $(i, m-j+1$ and $(n-i+1, j)$. Applying symmetry once more, we get that location $(n-i+1, m-j+1)$ must also contain a statue of type a.

Now, denote with $A_{i,j} = \{(i,j),(i,m-j+1),(n-i+1,j),(n-i+1,m-j+1)\}$. By the above discussion, it can be easily proven that:

An $n \times m$ rectangle is symmetric if and only if for all $i \leq n/2, j \leq m/2$,
statues in $A_{i,j}$ are of the same type.

It follows that we need to put the statues of the same type in each of these $\frac{nm}{4}$ quadruplets of locations. But which type in which quadruplet? Let's divide nm given statues in $\frac{nm}{4}$ quadruplets, so that each quadruplet contains the statues of the same type (there may be more quadruplets with the same type of statues). Denote the quadruplets with B_i, $1 \leq i \leq nm/4$. Such division exists because it is guaranteed that solutions exist. If we decide to put the statues of type B_k on the locations $A_{i,j}$, we need to remove all the statues from these locations which are not of B_k type. This is the "cost" of our choice and we need to minimize the sum of costs for all quadruplets. Note that we only consider removal cost because when we bring statues to some quadruplet $A_{i,j}$, we are removing them from another quadruplet.

Let's transform this problem a bit. Denote by $G(V,E)$ the complete bipartite graph with partitions $A = \{A_{i,j} | i \leq \frac{n}{2}, j \leq \frac{m}{2}\}$ and $B = \{B_k | k \leq \frac{nm}{4}\}$. For each edge $e \in E$ between $A_{i,j}$ and B_k, we assign it weight $w(e)$ to be the number of the statues on locations $A_{i,j}$ which are not of type B_k. It is obvious that $w(e)$ is the above mentioned cost. Note that $|A| = |B| = \frac{nm}{4}$. In this notation, our goal is to find a maximum matching M in graph G, such that $\sum_{e \in M} w(e)$ is minimized (notice that any perfect matching in G uniquely defines an assignment of statues to quadruplets of locations).

But this is the well-known **minimum weighted matching in bipartite graph** problem, which can be solved using the famous **Hungarian algorithm**. Complexity of this algorithm is $O(V^3)$, in our case $O(\left(\frac{nm}{4}\right)^3)$.

Solution by:

Name: ***Nikola Milosavljević***
School: The Faculty of Mathematics and Sciences, Niš
E-mail: nikola5000@gmail.com

Problem R1 09: Vasya Ferrari (ID: 1666)

Time Limit: 0.5 second
Memory Limit: 64 MB

Vasya, nicknamed Ferrari, has to solve an equation of fourth degree with integer coefficients $x^4+ax^3+bx^2+cx+d = 0$. Vasya wants to factorize the polynomial in the left part of this equation to the maximal possible number of multipliers with integer coefficients to reduce the problem to solving several equations of lower degree.

Input

4 integers: a, b, c, d — the coefficients of the polynomial, with absolute values not exceeding 20000.

Output

If the polynomial can't be factorized to multipliers with integer coefficients, you should output a single line “Irreducible”. In the other case output the factorization of the polynomial as a product of several polynomials with integer coefficients, enclosed in parentheses. You shouldn't delimit the multipliers with spaces and output monomials with zero coefficients. Coefficients and degrees equal to 1 should be omitted, except the monomial “1”.

Sample

input	output
`0 0 0 0`	`(x)(x)(x)(x)`
`-4 -3 24 45`	`(x2+3x+3)(x2-7x+15)`
`1 1 1 1`	`Irreducible`

Solution:

In this problem we have to reduce a fourth degree equation by factorizing the polynomial on the left-hand side to the largest number of factors with integer coefficients, so that the equation can be solved by solving two, three or more equations. The following quartic equation is given:

$$x^4 + ax^3 + bx^2 + cx + d = 0$$

A polynomial of the fourth degree can be factorized in several ways:

- the product of a third-degree polynomial with a first-degree polynomial
- the product of two second-degree polynomials
- the product of two first-degree polynomials and one second-degree
- the product of four first-degree polynomials

First of all, we will try to factorize our polynomial as a product of two quadratic polynomials.

$$x^4 + ax^3 + bx^2 + cx + d = 0 = (x^2 + a_1x + b_1) \cdot (x^2 + a_2x + b_2)$$

From this observation we get a system of equations:

$$a = a_1 + a_2$$
$$b = b_1 + b_2 + a_1a_2$$

$$c = a_1 b_2 + a_2 b_1$$
$$d = b_1 b_2$$

We can transform them into the following:

$$a_1(a - a_1) + b_1 + b_2 - b = 0$$
$$(-a_1)^2 + aa_1 + b_1 + b_2 = 0$$

It seems impossible to solve this directly. The easiest way to get around this, keeping in mind that we are only looking for integer solutions, would be to find all divisors of d, which is the set of all possible values for b_1 and b_2, and iterate through them. Knowing b_1 and b_2 we can solve the quadratic equation for a_1 and immediately obtain a_2 through one of the above equations.

So, if we have found a possible way to factorize our polynomial as a product of two quadratic polynomials, we are left with this equation:

$$(x^2 + a_1 x + b_1) \cdot (x^2 + a_2 x + b_2) = 0$$

Two quadratic equations follow:

$$(x^2 + a_1 x + b_1) = 0$$
$$(x^2 + a_2 x + b_2) = 0$$

We will try to factorize them separately. (Maybe we will not be able to factorize one or both of them). In order to factorize one quadratic equation into a product of two linear equations, all we need to know are the roots of the given equation, x_1 and x_2. By finding the roots we can write one of our quadratic equations as $(x - x_1) \cdot (x - x_2)$. We still mustn't forget that we are only interested in the integer roots.

We will stop here because the polynomial can't be factorized further. So far this solution seems OK, but sometimes our polynomial of the fourth degree can't be written as the product of two polynomials of the second degree. In this case we will try to factorize it as the product of polynomials of first and third degree.

$$x^4 + ax^3 + bx^2 + cx + d = 0 = (x + d_1) \cdot (x^3 + a_1 x^2 + b_1 x + d_2)$$
$$a = a_1 + d_1$$
$$b = b_1 + a_1 d_1$$
$$c = d_2 + b_1 d_1$$
$$d = d_1 d_2$$

We have a system of equations once again. And again, finding all divisors of d will help us to find the other variables. If we are successful, there is no need to go any further. Obviously, the linear polynomial can't be reduced, and if it were possible to factorize the third degree polynomial, it would reduce to the previous case - meaning that the original polynomial can be factorized into two quadratic polynomials, and we have already excluded that possibility. If we weren't able to factorize the polynomial in either of these ways, it is irreducible. There are some special cases when the constant term of our quartic equation is equal to zero ($d = 0$).

Solution by:

Name: ***Daniel Ferizović***
School: MSŠ Bosanski Petrovac
E-mail: dani.f@live.de

Problem R1 10: The Most Complex Number (ID: 1748)

Time Limit: 1.0 second
Memory Limit: 64 MB

Let us define a complexity of an integer as the number of its divisors. Your task is to find the most complex integer in range from 1 to n. If there are many such integers, you should find the minimal one.

Input

The first line contains the number of testcases t (1 ≤ t ≤ 100). The i-th of the following t lines contains one integer n_i ($1 \le n_i \le 10^{18}$).

Output

For each testcase output the answer on a separate line. The i-th line should contain the most complex integer in range from 1 to n_i and its complexity, separated with space.

Sample

input	output

5
1
10
100
1000
10000
	1 1
	6 4
	60 12
	840 32
	7560 64

Solution:

Statistically speaking, this problem was the easiest one in the qualification rounds (it was solved 54 times). This number theory problem is a very nice example for everyone who wants to try competing in programming. One of the reasons for this is that it can be solved in many ways. We are going to present three of them in brief here.

Firstly, let's try to formalize the problem a little bit. We have to find the minimal number from the segment $[1, n]$ that has the maximal complexity – the number of its divisors. The **Fundamental Theorem of Arithmetic** tells us that we can write any integer (greater than 1) as a unique product (up to the ordering of the factors) of the prime numbers. The number $m \in [1, n]$ can be represented as

$$m = {p_1}^{\alpha_1}{p_2}^{\alpha_2} \dots {p_k}^{\alpha_k}$$

where p_i are different prime numbers and α_i are nonnegative integer numbers. The nice thing about this representation is that the number x divides m if and only if in its representation ${p_1}^{\beta_1}{p_2}^{\beta_2} \dots {p_k}^{\beta_k}$ we have $\beta_i \le \alpha_i$ for all i. Now, if we denote the complexity of the number m as $d(m)$, we have that the following property holds:

$$d(m) = (\alpha_1 + 1)(\alpha_2 + 1) \dots (\alpha_k + 1)$$

because every prime p_i from the factorization of m can be included 0, 1... α_i times.

From this we can see that values p_i of the prime divisors are not important for calculating the complexity – only the exponents α_i. We need the minimal number that has maximal complexity, so we can observe that the prime factors of this number are going to be 2,3,5 … in order. If we skip a prime number, than we can replace the biggest prime with the skipped one and get a smaller number with the same complexity. This is very good, isn't it? Also, from the same argument we conclude that the degrees of the primes are sorted in non-decreasing order ($\alpha_i \geq \alpha_{i+1}$).

What is the largest prime that we have to consider? If we start to multiply all primes in order, the first time we get a result larger than10^{18} is after multiplying by 47. Therefore, only the first 15 prime numbers can appear in the factorization. We can similarly determine the maximal exponents for each prime factor. (Note that there are better bounds than these!)

Prime numbers	2	3	5	7	11	13	17	19	23	29	31	37	41	43	47
Maximal degree	59	37	25	21	17	16	14	14	13	12	12	11	11	11	10

Table of the first 15 prime number and corresponding maximal degrees

Now, we can solve the problem in many ways:

- **Backtrack**

We can backtrack through the set of all the numbers with the above characteristics. There are not more than 3.000 of them.

```
=================================================================================
       Function:      Solve
       Input:         num – current generated number (passed by value)
                      index – current prime number
                      deg [] – current degrees (for num)
       --------------------------------------------------------------------------
       if index is bigger than 15 then
              return;
       if num is better than current solution
              update current solution;
       while (num <= n) and (deg [k] <= deg [k – 1])
              solve (num, index + 1, deg);
              multiply num with index-th prime number;
              deg [k] = deg [k] + 1;
       deg [k] = 0;
=================================================================================
```

Pseudo code for backtracking function

- **Pre-calculation**

The number of different possible solutions is very small, so another idea is to pre-calculate all possibilities — we don't need to pay attention to how quick the algorithm is — and store them in arrays. After that we only need to traverse these arrays and fetch the solution.

```
    long num[] = {1, 2, 4, 6, 12, 24, 36, 48, 60, 120, 180, 240, 360, 720, 840, 1260, 1680, 2520, 5040, 7560, 10080, 15120, 20160, 25200, 27720, 45360, 50400, 55440, 83160, 110880, 166320,
221760, 277200, 332640, 498960, 554400, 665280, 720720, 1081080, 1441440, 2162160, 2882880, 3603600, 4324320, 6486480, 7207200, 8648640, 10810800, 14414400, 17297280, 21621600,
32432400, 36756720, 43243200, 61261200, 73513440, 110270160, 122522400, 147026880, 183783600, 245044800, 294053760, 367567200, 551350800, 698377680, 735134400, 1102701600,
1396755360, 2095133040, 2205403200L, 2327925600L, 2793510720L, 3491888400L, 4655851200L, 5587021440L, 6983776800L, 10475665200L, 13967553600L, 20951330400L, 27935107200L,
41902660800L, 48886437600L, 64250746560L, 73329656400L, 80313433200L, 97772875200L, 128501493120L, 146659312800L, 160626866400L, 240940299600L, 293318625600L,
321253732800L, 481880599200L, 642507465600L, 963761198400L, 1124388064800L, 1606268664000L, 1686582097200L, 1927522396800L, 2248776129600L, 3212537328000L, 3373164194400L,
4497552259200L, 6746328388800L, 8995104518400L, 9316358251200L, 13492656777600L, 18632716502400L, 26985313555200L, 27949074753600L, 32607253879200L, 46581791256000L,
48910880818800L, 55898149507200L, 65214507758400L, 93163582512000L, 97821761637600L, 130429015516800L, 195643523275200L, 260858031033600L, 288807105787200L,
391287046550400L, 577614211574400L, 782574093100800L, 866421317361600L, 1010824870255200L, 1444035528936000L, 1516237305382800L, 1732842634723200L, 2021649740510400L,
2888071057872000L, 3032474610765600L, 4043299481020800L, 6064949221531200L, 8086598962041600L, 10108248702552000L, 12129898443062400L, 18194847664593600L,
20216497405104000L, 24259796886124800L, 30324746107656000L, 36389695329187200L, 48519593772249600L, 60649492215312000L, 72779390658374400L, 74801040398884800L,
106858629141264000L, 112201560598327200L, 149602080797769600L, 224403121196654400L, 299204161595539200L, 374005201994424000L, 448806242393308800L, 673209363589963200L,
748010403988848000L, 897612484786617600L, 1000000000000000001L};

    long com[] = {1, 2, 3, 4, 6, 8, 9, 10, 12, 16, 18, 20, 24, 30, 32, 36, 40, 48, 60, 64, 72, 80, 84, 90, 96, 100, 108, 120, 128, 144, 160, 168, 180, 192, 200, 216, 224, 240, 256, 288, 320, 336, 360, 384,
400, 432, 448, 480, 504, 512, 576, 600, 640, 672, 720, 768, 800, 864, 896, 960, 1008, 1024, 1152, 1200, 1280, 1344, 1440, 1536, 1600, 1680, 1728, 1792, 1920, 2016, 2048, 2304, 2400, 2688, 2880,
3072, 3360, 3456, 3584, 3600, 3840, 4032, 4096, 4320, 4608, 4800, 5040, 5376, 5760, 6144, 6720, 6912, 7168, 7200, 7680, 8064, 8192, 8640, 9216, 10080, 10368, 10752, 11520, 12288, 12960,
13440, 13824, 14336, 14400, 15360, 16128, 16384, 17280, 18432, 20160, 20736, 21504, 23040, 24576, 25920, 26880, 27648, 28672, 28800, 30720, 32256, 32768, 34560, 36864, 40320, 41472,
43008, 46080, 48384, 49152, 51840, 53760, 55296, 57600, 61440, 62208, 64512, 65536, 69120, 73728, 80640, 82944, 86016, 92160, 96768, 98304, 103680, 0};
```

- **Dynamic programming**

We can define two lists as follows:

$qComplexity\ [i] -$ the list of all complexities for a number generated with the first i prime numbers and smaller than 10^{18}
$qMinimal\ [i]$ –the list of the minimal numbers for corresponding complexities

For the starting values we can use $qComplexity[0] = \{1\}$and $qMinimal[0] = \{1\}$. Now, we can get the values for other elements of these lists, going through already calculated elements. Of course, we have to sort these lists so we can quickly find the given complexity.

Solution by:
*Name**:** **Andreja Ilić***
School: The Faculty of Mathematics and Sciences, Niš
E-mail: ilic_andrejko@yahoo.com

Problem R2 01: Cockroach Race (ID: 1369)

Time Limit: 5.0 second
Memory Limit: 32 MB

At last, the spring came. Buds swell on the trees, the snow has almost thawn out. More and more often you can hear birds' sonorous twittering from the outside. Less and less students you can see at the USU math-mech department. Even the cockroaches, usual inhabitants of the hostels, show up very rarely.

What's the connection between these phenomena, you may ask. The answer is the Day of Mathematician and Mechanician celebration, which will begin really soon. At the same time, the traditional cockroach race will take place in the USU. That's what the students are occupied with now - they are training their pets. Everyone wants his pet to become the prize-winner and to receive the proud name of "Magaz".

The race rules are somewhat unusual. Every round, some kind of sweets are placed in N points of the racing area. Together with sweets, M cockroaches are released. N cockroaches that reach these little delights of cockroaches' life, will make it to the next round. During the race all spectators have an unique opportunity to place bets and to win a lot of money. But the totalizator organizers are puzzled, they cannot understand how to calculate the probabilities of cockroaches' victories quickly and without mistakes. This is absolutely required to make the maximum profit out of their enterprise. Math-mech is rather big department and everyone here wants to participate.

You are to determine, for each of N pieces of sweet, which of the cockroaches is closest to that piece. This will help to determine the race leaders.

Input

The first line of the input contains the number M ($1 \le M \le 100000$). M lines follow, containing 2 numbers each — these are coordinates of the cockroaches at the present moment. (M + 2)nd line of the input stream contains the number N ($0 \le N \le 10000$). N following lines contain coordinates of sweet pieces. All coordinates are floating point numbers ($-10000.0 \le x, y \le 10000.0$). The distance between any two cockroaches is not less than 10^{-3}. Also the distance between any two sweets is not less than 10^{-3}.

Output

For each piece of "Cockroach Sweets" you should output all cockroaches closest to that piece in ascending order of their numbers separated by spaces.

Sample

input	output

4
0 0
1 0
0 1
1 2
2
0 0
0 2
	1
	3 4

Solution:

This task is **extremely difficult**. Two most known approaches for spatial nearest neighbor search are **Voronoi diagram** and **k-d trees**. Other popular data structures are not good choice since they're optimized for different things other than speed: **R-trees** are optimized for I/O operations and **Quad-trees** have advantage over *k-d trees* only by being able to parallelize better. Picking any of these structures, which are hard to implement on their own, will not fully solve the problem. In order to address all speed issues, a number of various specific tricks and optimizations need to be done. A good reference is a book *The design and analysis of spatial data structures*.

The *k-d tree* is binary tree, usually balanced, in which every node is one point. Every node divides spatial region into two sub-spaces by line parallel to either x or y axis alternatively. Root for the tree is picked as median point when all points are sorted by x axis, and division line is parallel to y axis. Both of distinct regions are then recursively divided: roots for the sub-trees are medians when points are sorted by y axis, and both sub-spaces are further divided by lines parallel to x axis.

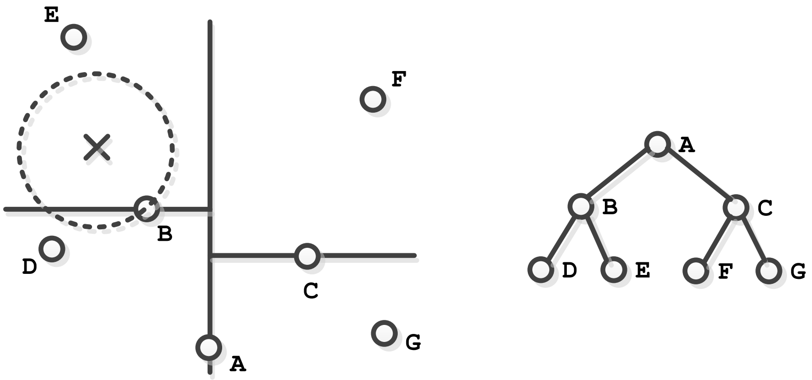


Figure 1. Example of k-d tree

Once the k-d tree is constructed from points representing positions of bugs, we need to search for nearest neighbor for each of points representing positions of sweets. Search of k-d tree is done in following way:

- Starting from the root node, move down recursively to find the region where the point would belong if it were element of the tree. The last node visited determines the current best.

- Moving upwards from the leaf node to the root node, check whether there's a need to check points in other sub-regions (on the other side of the splitting line).
    - If the circle with the center with query point and radius of the current best intersects current splitting line, there's a possibility that some points in the neighboring sub-region (corresponding to the sibling node) can have nearest neighbor candidates. Those regions should be processed in the same manner recursively.
    - If the circle does not intersect current splitting line, the entire branch corresponding to the sibling node is discarded.

At each point, current best is maintained. When root of the tree is reached, current best is final nearest neighbor.

Construction of k-d tree construction can be done in $O(M \cdot \log M)$ time. Queries could be done in $O(N \log M)$ time on average and $O(MN)$ in worst case.

Solution by:

Name: ***Milan Novaković***
School: The Faculty of Electrical Engineering, Belgrade
E-mail: milan.novakovic@microsoft.com

Problem R2 02: Light (ID: 1464)

Time Limit: 3.0 second
Memory Limit: 32 MB

Santa Claus Petrovich moved to a new hut. It consists of only one room. Its floor has the form of a simple polygon (not necessarily convex) with N vertices. It was dark in the hut at first, but then Petrovich hung a lamp at the point with projection (X_0, Y_0). Which area of the room is illuminated by the lamp?

Input

The first line contains the coordinates of the lamp (X_0, Y_0). You may regard the lamp as a material point. The second line contains the integer $3 \le N \le 50000$. In the next N lines there are coordinates (X_i, Y_i) of vertices of the N-gon. The vertices are given in the counter-clockwise order. All the coordinates are given as pairs of real numbers separated with a space, $0 \le X_i, Y_i \le 1000$. The coordinates contain not more than four fractional digits. It is guaranteed that the lamp is strictly inside the room.

Output

Output the area S of the illuminated part of the room. The area must be given with accuracy of at least two fractional digits.

Sample

input	output

`1.0 1.0`
`6`
`0 0`
`3 0`
`3 2`
`2 2`
`2 3`
`0 3`
	`8.00`

Solution:

In problems like this, it could be very useful to find a way to break down the calculation on a single complex object into calculations on multiple simpler objects. For this problem, this means finding a way to calculate the lighted area.

Let us try to calculate the lighted area within the given angle in the point (X_0, Y_0). There are a few major reasons why we should do that:

(a) the way the light lights in the circle around (X_0, Y_0) is just a special case of an angle,

(b) any circle can be divided into a finite number of non-overlapping angles,

(c) if the light within given angle lights only one edge, then calculating lighted area is quite simple - it is just a triangle.

Comparing (a) and (c), it is obvious that such calculation in some cases can be complicated, but in other cases it is very straightforward. As mentioned at the beginning we will try to separate the part (a) into (c)-s

in the following way:
Suppose we have divided the circle in some way (doing (b)) so far. Take any angle, and consider the following two possibilities:

(1) the light within the angle lights only one edge (Figure 1),
(2) the light within the angle lights at least two edges (Figure 2).

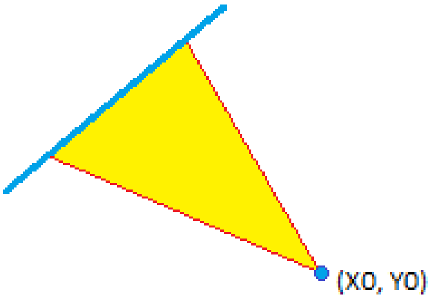


Figure 1. The light within the angle lights only one edge

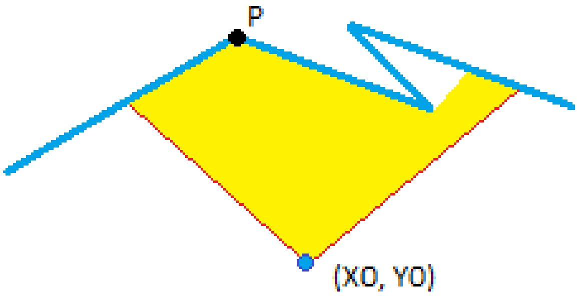


Figure 2. The light within the angle lights at least two edges

We have already mentioned that *(1)* is a simple case, so let us play with *(2)*. If the light within the angle lights at least two edges $e_1$and e_2, then there exists a point P over which light passes from e_1 before switching to e_2. We separate the angle by line $P-(X_0,Y_0)$ and replace the current angle with two just obtained (smaller) angles and continue the procedure. Such a point P we will call "switching point". This procedure raises two very important questions: how can we find a switching point, and is the described procedure finite?

Instead of asking "how we can find", let us ask "when we are sure" there is no switching point in the given angle. A switching point can occur as an intersection of two edges or as a vertex of an edge. The first is impossible by conditions of the problem statement, so a switching point can only be a vertex. Considering the fact that whenever a switching point occurs it has to be a vertex of an edge, we immediately give the answer to the second question - the described procedure is finite, because we have at most n vertices.

Now we can come up with an easy procedure for choosing the angle which does not contain any point within it, except maybe at the lines of the angle. The following pseudo-code describes this procedure:

```
=================================================================================
for each vertex V
   let angle(V) be angle between line V-(X0, Y0) and x-axis
let A be array of sorted vertices in ascending order by their angle(V) value
make A to be a circular array
for each two (V1, V2) adjacent vertices in A
  angle V1-(X0, Y0)-V2 is angle which does not contain switching point
=================================================================================
```

Note that in this way we got angles described by case*(b)*, and each of those angles is of type *(1)*. Calculating lighted area within any angle $\measuredangle(V_1,(X_0,Y_0),V_2)$ of type *(1)* can be done in the following way:

```
=================================================================================
01  doublelightedArea(vertex V1, vertex V2)
02    Xs = (V1.x + V2.x) / 2
03    Ys = (V1.y + V2.y) / 2
04    let A be array of edges. A contains an edge e only iff interior of
       angle V1-(X0, Y0)-V2 contains at least one point of e
05    for each edge e in A
06     let dist(e) be a distance from (X0, Y0) and intersection point between e and line
       (Xs, Ys)-(X0, Y0)
07    sort A in ascending order according by dist(e) value
08    let minE be the first edge of A
09    let (XV1, YV1) be the intersection point between minE
       and line (V1.x, V1.y)-(X0, Y0)
10    let (XV2, YV2) be the intersection point between minE
       and line (V2.x, V2.y)-(X0, Y0)
11    return areOfTriangle(X0, Y0, XV1, YV1)
=================================================================================
```

Note that if there exists at least one point of an edge e in the interior of an angle, like mentioned on the line 4, then the interior of the angle does not contain vertex of e. On the contrary, the angle will contain a switch point and it would not be of type *(1)*. Also, note that once we have chosen minimal edge on the line 8, we have chosen the only edge lighted within the angle. By contrast, if we suppose there exists another edge e_1, different than e, lighted within the same angle it would mean e_1 and e intersect, which is impossible.

Let us summarize what we have concluded so far - for each angle, and there are exactly n of them, we can in $O(n\,log\,n)$ steps decide lighted area within it. We need $O(n\,log\,n)$ steps per one angle because of sorting method on the line 07. So, using described methods we obtained an algorithm with time complexity $O(n^2 log\,n)$. Well, taking into account time limit, such algorithm is slow.

Now we will go a few steps back and consider again the point of what is the main difference between asking "how we can find" and "when we are sure" there is no switching point in the given angle. Let us take a look and see in which way these questions affect simple test case. In the Figure 3 is shown what algorithm should detect if we ask the first question, and in the Figure 4 what the algorithm is supposed to find if we have asked the second question. Although we have less area calculations once we give answer to the first question, it shows that asking the latter question is more efficient. We simplified the question, in some cases got a few more area calculations, but got a lot easier method for deciding whether there is or not a switch point.

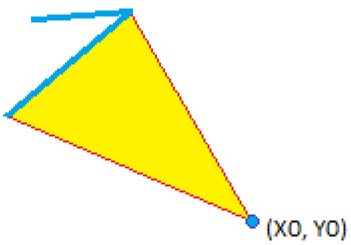


Figure 3. What should algorithm detect if we ask the first question

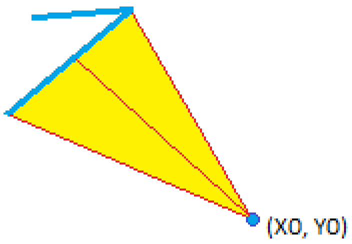


Figure 4. What algorithm is supposed to find if we have asked the second question

Let us move on and conclude how we can make a more efficient algorithm. We are going to analyze lines 04 and 07, and what we have said about them. Obviously, no matter which angle and which type of it we have chosen, and no matter which two edges e_1 and e_2 we have chosen, as long as e_1 nor e_2 make a switch point within the angle, once e_1 is greater (line 07) than e_1 it will always be greater than e_2. This is a very important observation. To explain it, we will choose three consecutive vertices V_1, V_2 and V_3, and consider two edges e_1 and e_2 which both have at least one point in interior of the angles $\measuredangle(V_1, (X_0,\ Y_0), V_2)$ and $\measuredangle(V_2, (X_0, Y_0), V_3)$. A call of the function *lightedArea*(V_1, V_2) will compare edge e_1 against e_2, and call *lightedArea*(V_2, V_3) will also compare e_1 against e_2. The latter one is unnecessary; we already know what is greater from the previous step. This is the key of the optimization. Instead of sorting edges over and over, we should update structure of sorted edges. Note that the suitable structure we need does not have to answer what is i-th ordered element; we need only minimal/maximal element (the line 08). The structure should be dynamic because we will possibly add new edges to it, when parameter V_1 is equal to the starting vertex of the edge, or remove some edges from it, when parameter V_1 is equal to the end vertex of some edge. The structure that fulfills those requests is a heap.

By using the heap, lines 05, 06 and 07 are replaced by maintaining edges that should be removed or added as described -- according to their end/start points and the parameters $V_1$and V_2. During sweep through all angles each edge will be added and removed exactly once. Adding to the heap or removing an element from the heap is done in $O(log\ size\_of(heap))$. The size of the heap is at most n, so for sure we do not need more then $O(log\ n)$ operations for adding or removing. Operation on the line 08 is executed in $O(1)$. Just to remind ourselves, we sweep over n angles and sort n vertices. Overall time complexity of this approach is $O(n\ log\ n)$. Finally, we designed an efficient algorithm that runs in time.

In the end, take a look at an example - how we make the angles and light them.

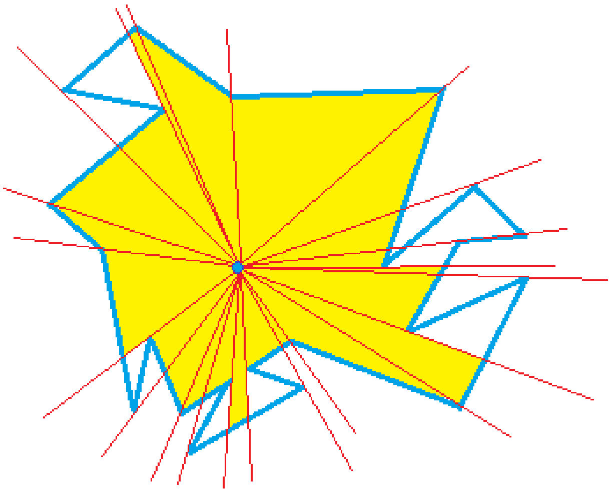


Figure 5. Example of cutting a polygon into angles.

Solution by:

Name: ***Slobodan Mitrović***
School: The Faculty of Mathematics and Sciences, Novi Sad
E-mail: boba5555@gmail.com

Problem R2 03: Fat Hobbits (ID: 1533)

Time Limit: 1.0 second
Memory Limit: 64 MB

None of the hobbits can fight Mordor's army on his own. Gandalf have chosen N hobbits from Shire to form a platoon that will go on a new campaign against Mordor. But some of the hobbits refuse to go because they are afraid that other hobbits in the platoon will call them fat. More exactly, each hobbit refuses to go on the campaign if there will be at least one hobbit with smaller weight in the platoon. Fortunately, hobbits don't know their exact weights. They can only compare their weights using a pan balance, and there is only one pan balance in Shire. Some pairs of hobbits used it to determine which of them was heavier. All hobbits know the results of all weighings. Gandalf is sure that there are no two hobbits with the same weight. Help Gandalf to choose from the N hobbits as many hobbits as possible provided that they will agree to go on the campaign together. Remember that hobbits are clever creatures and know that if, for example, Sam is heavier than Pippin and Pippin is heavier than Frodo, then Sam is heavier than Frodo.

Input

The first line contains the number N of hobbits which were primarily chosen by Gandalf (2 ≤ N ≤ 100). The hobbits are numbered from 1 to N. In the next N lines there is a matrix N × N, which shows the results of weighings. If hobbits with numbers i and j weighed themselves against each other and it turned out that hobbit i was heavier, then there is 1 at the intersection of row i and column j. All other elements are zeros.

Output

In the first line output the maximal number of hobbits in the platoon. In the second line, give their numbers.

Sample

input	output

2
0 1
0 0
	1
	2

3
0 0 0
0 0 0
0 0 0
	3
	1 2 3

Solution:

To solve this problem, we can use some ideas from the graph theory. We can represent hobbits as nodes with an edge from node A to node B if and only if hobbit A knows he is heavier than hobbit B. We are asked to find the **maximum independent set (MIS)** in such a graph. This is an NP-hard problem in general, but this graph has a special structure:

1) There are no cycles (*antisymmetry*)
2) If there is an edge from node A to node B and from node B to node C then there is an edge from node A to node C (*transitivity*)

This graph, therefore, corresponds to a **partial order relation**.

Dilworth's theorem states that cardinality of the maximum **antichain** in a partially ordered set is equal to the minimum number of chains.

In this case, the maximum antichain is the maximum clique in the complemented graph, which is equal to maximum independent set in the original graph, and the minimum number of chains is the minimum number of paths that cover the entire graph.

In order to find these paths, we can build a bipartite graph with hobbits on both the left and the right side. There is an edge between node A (a node on the left side) and node B (a node on the right side) if and only if hobbit A knows he is heavier than hobbit B. This graph contains duplicate nodes from the original graph on both sides and edges are directed according to the partial order relation defined among hobbits. **Max-flow min-cut theorem** states that the set of edges with flow is equal to matching because this is a bipartite graph.

Once we have the appropriate matching, we need to find the nodes that are in MIS, which can be done by using **Konig's theorem**.

This procedure can be implemented in $O(V^3)$ with 50-60 lines of code, but there is a faster algorithm which runs in $O(V^2 + E\sqrt{V})$ where V is the number of nodes and E is the number of edges in the graph.

Outline of the $O(V^3)$ complexity algorithm:

1) Find transitive closure using the **Floyd-Warshall** algorithm in $O(V^3)$
2) Duplicate nodes and create bipartite graph
3) Find matching in $O(VE)$
4) Find MIS

Outline of the $O(V^2 + E\sqrt{V})$ complexity algorithm:

1) Find transitive closure in $O(V^2)$ – hint: this graph is acyclic
2) Duplicate nodes and create bipartite graph
3) Find matching using **Hopcroft-Karp** in $O(E\sqrt{V})$
4) Find MIS

Solution by:

Name: ***Boris Grubić***
School: "Jovan Jovanović Zmaj" Grammar School
E-mail: borisgrubic@gmail.com

Problem R2 04: Aztec Treasure (ID: 1594)

Time Limit: 1.0 second

Memory Limit: 64 MB

During the recent excavations in Teotihuacan archeologists found a strange casket, the contents of which was probably used during the legendary corbans held by Montezuma, and a lot of equal rectangular bone pieces of size 1 × 2.

Archeologists found out that in order to open the casket you should tile the rectangular covering of this casket with bone pieces in a specific way. Pieces cannot overlap and intersect the border of the covering. Archeologists are afraid to break the casket, so they just want to try all possible ways of tiling. Your task is to calculate the number of such ways.

Input

The only line of the input contains two space-separated integers l and w, the length and the width of the casket's covering (1 ≤ l, w ≤ 100).

Output

Output the number of ways of tiling modulo $10^9 + 7$.

Sample

input	output
3 4	11

Solution:

The problem is to calculate the number of ways to cover a rectangular board of dimensions $n \times m$ with domino tiles of size 1×2. The solution is more complex than the mere description of the problem suggests! Temperley & Fisher (1961) and Kasteleyn (1961) independently came to the closed form of the problem solution (1).

$$\prod_{j=1}^{m}\prod_{k=1}^{n}\left(4\cos^2\frac{j\,\pi}{m+1}+4\cos^2\frac{k\,\pi}{n+1}\right)^{1/4} \qquad (1)$$

However, computing the solution using the previous formula requires the work with high-precision floating-point numbers, and circumvents the intuitive combinatorial approach. The following description of the solution is not based on a direct evaluation of the previous formula, but on graph theory and matrix algebra.

Representation of the board with planar graph and perfect matching

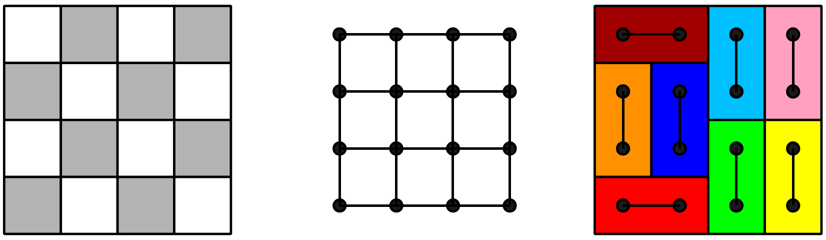


Figure 1. Representation of the board with the planar graph and perfect matching

If the board is turned into a graph (Figure 1.) by replacing the squares with vertices and putting an edge between adjacent squares, the required number of possible tilings is the number of perfect matchings in the resulting graph. Particularly, putting dominoes on the board corresponds to selecting edges from the graph such that no two edges share a vertex (i.e. domino tiles do not overlap). A matching in the graph is a set of pairwise non-adjacent edges. Matching is said to be perfect if every vertex of the graph is included in it. Of course, the graph can only have a perfect matching if the number of vertices is even. It follows that the product of the board dimensions must be even, that is, the dimensions should not both be odd (otherwise there is no complete covering).

Although there is a polynomial algorithm to determine the perfect matching of an arbitrary graph, counting the number of perfect matchings in a general graph has been shown to be *#P-complete* (much harder than *NP-complete*). Nevertheless, for certain special cases, among which are the planar graphs, there are efficient algorithms. A graph is said to be planar if it can be placed in the *2D* plane in such a way that its edges do not intersect.

Counting perfect matchings in the planar graphs

(Those less keen on mathematics can skip the following paragraph)
Let $G(V,E)$ be a graph on n vertices, where n is even. The following definitions are introduced:

- $A(G)$ – the adjacency matrix of G ($A_{ij} = 1\ if\ f\ (i,j) \in E, 0\ otherwise$).
- $PM(n)$ - set of all partitions of n elements into pairs without regard to the order. Each element of $PM(n)$ can be thought of as a permutation of the integers between 1 and n (where edges are represented by consecutive pairs of two numbers), and gives a potential perfect matching of G.
- $r(G) = \sum_{M \in PM(n)} \prod_{(i,j) \in M} A_{ij}$ - the number of perfect matchings. As each element M of the set PM is a permutation, *each vertex* appears *exactly once*, which is a necessary condition for a perfect matching. However, in order for the condition to be sufficient, it is necessary that graph contains each edge comprised in the particular element M. If any edge does not exist, associated A_{ij} element will be zero and the entire product will be zero which will not affect the sum. If all edges exist, the product will be 1 and the sum will be increased accordingly. Since the set PM contains all the necessary permutations (permutations regardless to order of the elements within pairs), all the perfect matchings will be counted.
- $Pf(A') = \sum_{M \in PM(n)} Sgn(M) \prod_{(i,j) \in M} A'_{ij}$ - the *Pfaffian* of a matrix A' where $sgn(M)$ is the sign of M as a permutation of n elements (depending on the parity of number of equivalent transpositions). If the signs of adjacency matrix A are adapted forming new matrix A'so that

$\forall M \in PM \mid \prod_{(i,j)\in M} A'_{ij} = Sgn(M)$ holds, G' is said to be a *Pfaffian* orientation of G, and $r(G) = Pf(A')$ holds.

- According to the theorem (Kasteleyn, 1963), every planar graph has a *Pfaffian* orientation that can be found in polynomial time. Let G be a planar graph. Then G can be oriented efficiently so that each face has an odd number of lines oriented clockwise, and this is a *Pfaffian* orientation of G.
- According to the theorem (Muir, 1882), for *skew-symmetric* matrices ($A'_{ij} = -A'_{ji}$) the formula $Pf(A')^2 = Det(A')$ holds.

In the previous paragraph it was shown that the number of perfect matchings can be computed as the square root of the determinant if the graph can be represented by a corresponding *skew-symmetric* adjacency matrix in a *Pfaffian* orientation. *Skew-symmetry* is achieved by a systematic orientation of edges so that for edge $(i \rightarrow j)$ element $A_{ij} = 1$ and $A_{ji} = -1$. The lattice graph can be trivially oriented into a *Pfaffian* orientation as shown in figure 2.

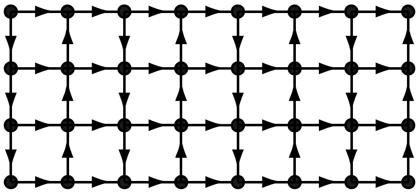


Figure 2. *Pfaffian* orientation of a lattice graph

Problems with direct implementation

It is easy to see that the direct implementation of the previous results is not efficient enough. Specifically, the resulting graph has $n \cdot m$ vertices, and computing the determinant of a generic matrix is $O(p^3)$ which gives the overall complexity of $O(n^3m^3)$. Furthermore, the problem arises in extracting the square root of the determinant, because of the inability to determine the root sign, which apparently prevents the use of the modulo arithmetic. For example: $10000 \equiv 11449 \equiv 18\ (mod\ 23)$, $\sqrt{10000} = 100 \equiv 8\ (mod\ 23)$, $\sqrt{11449} = 107 \equiv 15\ (mod\ 23)$. Knowing the $18\ (mod\ 23)$, without whole results (10000 and 11449), it is not possible to determine which is the correct positive root (8 or 15) since both $8^2 \equiv 18$ and $15^2 \equiv 18\ (mod\ 23)$. The inability to use the modulo arithmetic requires to use *big numbers*, which makes procedure more computationally complex.

Reducing the complexity by studying the structural properties of the adjacency matrix

By studying the structural properties of the adjacency matrix (Figure 3.) several things can be noticed. One of the most important properties is that it is a tridiagonal block matrix consisting of $n \times n$ submatrices of size $m \times m$. In further calculations, all the elements are previously mentioned submatrices of size $m \times m$, and the corresponding complexity of the operations over them is $O(m^3)$. Furthermore, as will be shown later, it is advisable to swap m and n if necessary, so that m is smaller! The determinant of the tridiagonal matrix can be calculated in $O(n)$ where n is the dimension of the matrix, which gives the overall complexity of $O(n \cdot m^3)$. The determinant is calculated by reducing the matrix to a triangular one and multiplying the diagonal elements (Figure 4.).

	1			1															
-1		1			-1														
	-1		1			1													
		-1					-1												
-1					1			1											
	1			-1		1			-1										
		-1			-1		1			1									
			1			-1					-1								
				-1					1			1							
					1			-1		1			-1						
						-1			-1		1			1					
							1			-1					-1				
								-1					1			1			
									1			-1		1			-1		
										-1			-1		1			1	
											1			-1					-1
												-1					1		
													1			-1		1	
														-1			-1		1
															1			-1	

Figure 3. *Skew-symmetric* adjacency matrix for $5 * 4$ board

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & \ddots & 0 \\ 0 & 0 & \ddots & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 \end{bmatrix}$$

Structure of submatrix A

A_1	B_2			
C_2	A_2	B_3		
	C_3	A_3	B_4	
		C_4	A_4	B_5
			C_5	A_5

D_1	B_2			
	D_2	B_3		
		D_3	B_4	
			D_4	B_5
				D_5

Figure 4. Tridiagonal matrix

The diagonal elements of the triangular matrix are calculated recursively by the relations (2) and (3). It is clear that the elements on the three diagonals are equal ($A_i = A, B_i = B, C_i = -B$), and for further structural properties the expression $B_n \cdot D_{n-1}^{-1} \cdot C_n$ can be replaced with D_{n-1}^{-1}, which produces the relation (4).

$$D_1 = A_1 \tag{2}$$
$$D_n = A_n - B_n \cdot D_{n-1}^{-1} \cdot C_n \tag{3}$$
$$D_n = A - D_{n-1}^{-1} \tag{4}$$

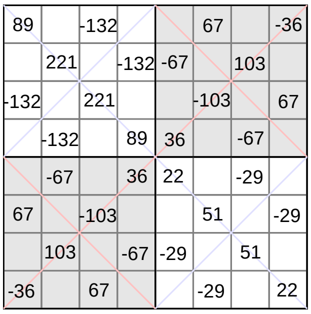
The final determinant is equal to the determinant of the element K_n which is determined by the relations (5) and (6). Further rearranging gives the relation (7).

$$K_1 = D_1 = A \tag{5}$$
$$K_n = D_n \cdot K_{n-1} \tag{6}$$
$$K_n = A \cdot K_{n-1} - K_{n-2} \tag{7}$$

The recursive relation (7) can be efficiently solved by a matrix exponentiation (8), (9) and (10). Complexity of this approach is $O(\log(n) m^3)$.

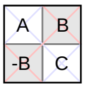
$$M = \begin{bmatrix} A & -I \\ I & \emptyset \end{bmatrix} \tag{8}$$
$$\begin{bmatrix} K_n \\ K_{n-1} \end{bmatrix} = M \cdot \begin{bmatrix} K_{n-1} \\ K_{n-2} \end{bmatrix} \tag{9}$$
$$\begin{bmatrix} K_n \\ K_{n-1} \end{bmatrix} = M^n \cdot \begin{bmatrix} I \\ \emptyset \end{bmatrix} \tag{10}$$

By further exploiting the structural properties, the complexity can be additionally improved by a constant factor. The matrix M^n consists of 4 submatrices (Figure 5. and 6.) which are (anti)symmetrical about the major and minor diagonal. Furthermore, half the elements of the submatrices are zero, and two submatrices are equal (up to sign). Knowing the properties, it is sufficient to calculate asymptotically less than a tenth of the elements ($\frac{1}{4} * \frac{1}{2} * \frac{3}{4} = \frac{3}{32}$), which is equivalent to tenfold speedup!



89		-132			67		-36
	221		-132	-67		103	
-132		221			-103		67
	-132		89	36		-67	
	-67		36	22		-29	
67		-103			51		-29
	103		-67	-29		51	
-36		67			-29		22

Figure 5. Matrix M^n for $n = 8, m = 4$



A	B
-B	C

Figure 6. Matrix M^n for even n

Note: These structural properties are valid only for even powers of M, but this is easy to overcome by powering the square of the matrix M, and finally, if necessary make one additional matrix multiplication ($M^n = M^{2q+r} = M^{2q} * M^r$, where $r = 0\ or\ 1$). All these structural properties and preservation of them can be proved by mathematical induction.

Extracting the square root of determinant in modulo arithmetic

Although there are algorithms for extracting the square root in modulo arithmetic (Cipolla, Pocklington, Tonelli–Shanks), in this case they can be avoided due to the structural properties of diagonal elements. By an intuitive inspection the following can be observed: When calculating the determinant of the element K_n, if, instead of all, only every second diagonal element is taken into product, the result obtained is exactly the square root of the determinant!

For example, by performing Gaussian eliminations on the matrix in figure 5, the following diagonal elements are obtained: $\{89,221,314606769,257918564\} (\bmod 1000000007)$. Product of every second element is: $89 * 31460676 \equiv 2245$ or $221 * 257918564 \equiv 2245$ which is precisely the square root of the determinant. For $m = 7$ and $n = 8$ the diagonal elements are $\{90,236,755555630,915254365,558804690,580166320,855770133\} (\bmod 1000000007)$. Product of every second element is: $90 * 755555630 * 558804690 * 855770133 \equiv 236 * 915254365 * 580166320 \equiv 1292697$.

It still remains to determine the sign of the root. As in the previous step, by observation it can be concluded that the sign depends on the remainder of division of m and n by 4. The exact expression is given in the source code.

Curiosities

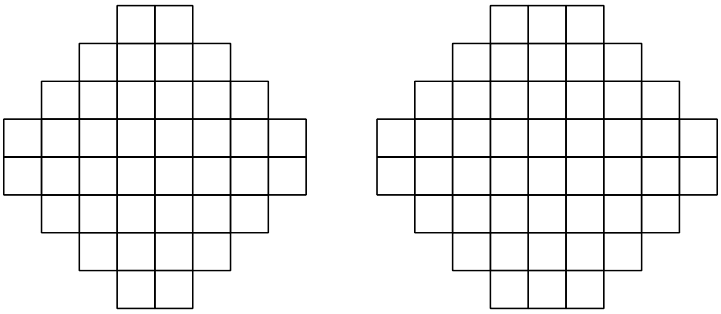


Figure 7. Aztec diamond of order 4, and the augmented Aztec diamond

The number of tilings of a region is very sensitive to boundary conditions, and can change dramatically with apparently insignificant changes in the shape of the region. This is illustrated by the number of tilings of an Aztec diamond of order n, where the number of tilings is $2^{n(n+1)/2}$. If this is replaced by the "augmented Aztec diamond" of order n with 3 long rows in the middle rather than 2, the number of tilings drops to the much smaller number $D(n,n)$, a *Delannoy* number, which has only exponential rather than super-exponential growth in n. For the "reduced Aztec diamond" of order n with only one long middle row, there is only one tiling. Asymptotically, an $m \times n$ board has about 1.339^{mn} tilings.

Solution by:

*Name**: Ognjen Dragoljević***
School: The Faculty of Electrical Engineering and Computing, University of Zagreb
E-mail: ognjen.dragoljevic@gmail.com

Problem R2 05: Abstractionism to the People (ID: 1649)

Time Limit: 1.0 second

Memory Limit: 64 MB

I will never paint again," Dunno answered. *"I paint and paint, and nobody is ever thankful. Everybody keeps scolding me."*

The great abstractionist artist Herman Brooks invented a new style in painting—bactorgaphy. Of course, you want to know what kind of a style it is. That's simple: every painting is live, quite literally. Herman paints with bacteria.

Such a painting is a real work of art. It's a sight worth seeing—the fascinating canvas plays with two or three hundred different shades. But how could this wonder be shown to the people? Photography or video just can't convey the entire range of colors, and Herman still doesn't have a museum (modern art custodians don't like innovative ideas, and there's no point arguing with them). In addition, the painting can be seen in detail only under a microscope. Finally, it was decided to make several thousand copies of the best paintings and sell them as souvenirs. However, there is a problem. Herman, as a real creator, doesn't want to make copies himself, and the hired bioengineers unanimously claim that a copy can only be made if the exact sequence of populating the canvas with bacteria is known. Your task is to restore this sequence.

To help you fulfill the task, the bioengineers provided you with the following information.

- A finished painting is a rectangular canvas divided into equal square cells with bacteria.
- Before the process of painting is started, the canvas is thoroughly disinfected. All its cells are empty and contain no bacteria.
- In each cell of the canvas there can be at most four bacteria.
- The painting process consists in settling successively one bacterium into a free cell of the canvas. When the bioengineers do this, the numbers of bacteria in the adjacent (top, bottom, left, and right) populated cells increase by one. If the number of bacteria in a certain cell becomes 5, then 4 of them die because of overcrowding.
- It is impossible to settle a bacterium into a cell that is already populated, because it leads to an unpredictable reaction that can damage the whole painting.

Input

The first line contains the dimensions of the canvas n and m (1 ≤ n, m ≤ 20). The description of the painting follows in the form of the table with n lines containing m integers each. In every cell of this table the number of bacteria in the corresponding cell of the painting is written. These numbers range from 1 to 4.

Output

If it is impossible to obtain the described painting by means of the procedure available to the bioengineers, output "No". If you managed to find a sequence that makes it possible to create a copy of Herman's masterpiece, output "Yes" in the first line, and in the following lines give this sequence. Each of these lines must contain two integers, which are the number of line and number of column of the next cell to be populated.

Sample

input	output

3 3
2 2 1
3 1 3
1 2 2
	Yes
	2 2
	2 1
	1 1
	1 2
	2 3
	1 3
	3 3
	3 2
	3 1

Solution:

Denote the original matrix with $A[m][n]$. The values of the fields of matrix A are in the range $1 \le A[i][j] \le 4$, but they can also take the value 0 at some points.

Let $TotalNeighbors(i,j)$ denote the number of neighbors of the field (i,j) (two for corners, three for edges and four for all other fields). Next, let's create a new matrix of the size $m \times n$ and call it $Degree$. At any moment, $Degree[i][j]$ represents the number of neighbors of the field $A[i][j]$ not equal to zero at that time (we will call them *current neighbors*).

An easy thing to notice is that the value of the element in the matrix that was filled last has to be 1. This leads us to the idea to try finding a field (i,j) where $A[i][j] = 1$ and $Degree[i][j] = TotalNeighbors(i,j)$. If $TotalNeighbors(i,j) < 4$, we can safely conclude that this 1 was obtained by filling the field after filling all its neighbors. Then we can set $A[i][j]$ to 0 and decrease the values of all its neighbors by one (if some of these values were 1, they become 4) and update all corresponding fields in the matrix $Degree$. However, if $TotalNeighbors(i,j) = 4$, there is another possibility - (i,j) could have been filled before its neighbors, increasing four times to go back to 1. How do we get around this?

Unfortunately, in the general case, we can't conclude locally (using just data from an area around this field) which of these two options is the correct one. So **backtrack** is a good way to go. We can try one possibility, and if we get stuck, then try the other one.

We have already described what we have to do if 1 was obtained by filling that field after filling its neighbors. For the second option, we set $A[i][j]$ to 0 and update all neighboring fields in $Degree$, but we don't decrease the values in its neighboring fields. Note that both options represent simulating a step of the matrix-filling process, but the first choice represents a "backward" step in time, while the second one is a step in the same direction in which the matrix was originally filled. We can keep all "forward" (first) steps and "backward" (last) steps we have made in separate lists, and use these lists to reconstruct the filling order in the end.

This is a valid solution, but it can be too slow for this problem. So we can notice some other properties of the matrix: if there are positions (i,j) for which $Degree[i][j] = A[i][j] - 1$, the field (i,j) is filled before all its current neighbors and we can make a forward step by filling (i,j). Further, if

$A[i][j] = 1$ and $Degree[i][j] < 4$, the field (i, j) was filled after all its current neighbors and we can make a backwards step by filling it. This considerably reduces the number of choices we will have to make.
If during the execution of the algorithm the matrix A becomes a zero matrix, we have reached our goal, we output “Yes“ and the filling sequence. On the other hand, if we are in a situation where we cannot make further progress by applying any of the described operations, the output is $'No'$.

The time complexity of the algorithm is exponential. The memory complexity is (mn) .

Solution by:

Name: Mladen Radojevic
School: The Faculty of Electrical Engineering, University of Belgrade
E-mail: mladen0211@yahoo.com

Problem R2 06: The Hobbit or There and Back Again 2 (ID: 1663)

Time Limit: 0.5 second

Memory Limit: 64 MB

Old Bilbo collects songs and sagas of all races of Middle-earth. Every twenty years he leaves Rivendell for a year to travel through N cities of the Middle-earth, numbered from 1 to N (Rivendell has number 1), and comes back at the end of the journey. Nineteen years have passed since Bilbo's last journey, so he started to prepare for travelling. From his last journey Bilbo remembers that there is a warder at the entrance to each city. He asks the travelers what city they came from and requires an entrance fare depending on that. Time passed, and the entrance fee has changed since the last journey. From the King of Elves Elrond Bilbo has learnt that, if a traveler arrives to a city with number A from a city with number B, the warder will require exactly $P_A \cdot [1000/P_B]$ gold coins, where P_i is the population of city with number i and [X] denotes the floor of X. Officials think that it will stimulate the population outflow from the bigger cities to the smaller ones. Bilbo knows a population of all cities of Middle-earth and supposes that it will not change during the year of his journey. As usual, before the journey he would like to know the order of visiting the cities which will minimize the total amount of money paid.

Input

The first line contains an integer N. $2 \le N \le 1000$. The second line contains N integers $P_1, \ldots, P_N$, delimited with spaces — the populations of the cities of Middle-earth. All P_i lie in range from 1 to 1000.

Output

Output N integers — order of visiting N cities which minimizes the total entrance fee. Remember that Bilbo starts his travel from the city with number 1, visits each city exactly once and returns to the city with number 1 only in the end. If there are several solutions, you may output any one of them.

Sample

input	output
4 10 3 5 4	1 4 2 3

Solution:

The problem of finding an optimal route that goes through every city exactly once and returns to the starting city is known as **The Traveling Salesman Problem (TSP)**. It is known that TSP is an NP-complete problem, which means that it is unlikely to have a solution of polynomial time complexity. However, if we know some details about the input, we could use them to design an efficient (polynomial) solution. So, what do we know about the input graph? We know it is a complete graph with N vertices (cities), and we know that the edge weights have an additional property – they are calculated from weights of endpoint vertices (weight of city a is P_a).

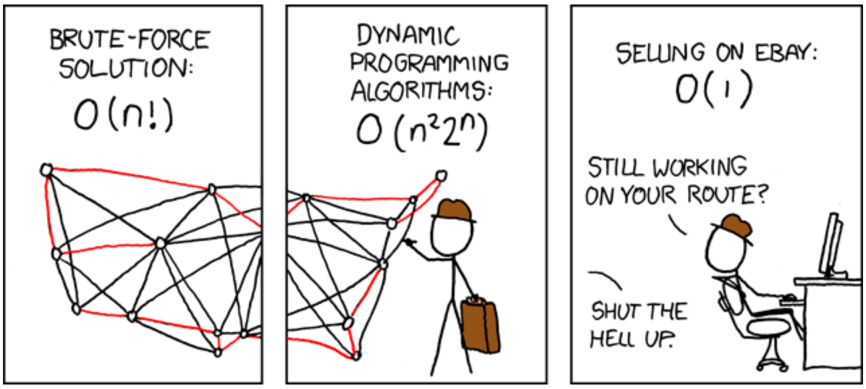

BRUTE-FORCE SOLUTION: O(n!)
DYNAMIC PROGRAMMING ALGORITHMS: O(n²2ⁿ)
SELLING ON EBAY: O(1)
STILL WORKING ON YOUR ROUTE?
SHUT THE HELL UP.

Figure 1. Traveling Salesman Problem comics
Taken from xkcd.com – A web comic of Romance, Sarcasm, Math, and Language

Since the route we are looking for is a cycle, we can choose any city to be the starting one. Let it be the city with the smallest weight (let's call it city A, and the one with the highest weight city B). We can separate this cycle into two vertex disjoint paths (except for starting and ending vertex) – path from A to B, and path from B to A. Let's take a closer look at the path from A to B, $A = x_1 \rightarrow\ x_2 \rightarrow x_3 \rightarrow \cdots \rightarrow x_k = B$. The cost of this path is $\sum_{i=1}^{k-1} x_{i+1} \left\lfloor \frac{1000}{x_i} \right\rfloor$. The crucial observation here is that the optimal path that goes from A to B, through the cities $x_2, x_3, \ldots, x_{k-1}$, is the one in which cities are visited in increasing order according to their weights. We will now give a sketch of the proof.

Let $a_1, a_2, \ldots, a_k$ be the order of the cities $x_1, x_2, \ldots, x_k$ according to their weights. We would like to show that among all the permutations π of $\{2, \ldots, k\}$, the one for which the sum $S = \sum_{i=1}^{k-1} a_{\pi(i)} \left\lfloor \frac{1000}{a_i} \right\rfloor$ is smallest is the permutation $\pi(i) = i + 1$. Without loss of generality, let's suppose that in the optimal permutation (the one for which the summation achieves the minimum), $\pi(1) = k\ \neq 2$. Since π is a permutation, there must exist some j such that $\pi(j) = 2$. If we swap these two values, $\pi(1) \leftarrow 2$and $\pi(j) \leftarrow k$, new summation will obviously be at least as small as summation S. By repeating this swapping for every i such that $\pi(i) \neq i + 1$, we eventually get the desired permutation as the optimal one. Note that during this process we might encounter some permutations for which the summation does not correspond to any ordering of the cities or corresponds to ordering of the cities which does not end in city B. This is, however, not the problem because we have actually proved a stronger statement than needed.

It is straightforward now to translate this observation into a **dynamic programming** solution. We will add one city at a time, in the order of increasing weights, and keep track of the last city added to both paths (this is the only information we need, because of the proven observation). This way we will also keep these two paths vertex-disjoint, and every city will be added exactly once. Since there are at most N^2 possible states, and every state needs constant computing time, the time complexity is $O(N^2)$.

Solution by:
Name: ***Rajko Nenadov***
School: The Faculty of Mathematics and Sciences, Novi Sad
E-mail: rajkon@gmail.com

Problem R2 07: Asterisk (ID: 1670)

Time Limit: 0.5 second

Memory Limit: 64 MB

Recently Cuckooland mathematicians have invented a new binary operation “asterisk”, which uses sequences as its arguments. Operation just appends the first sequence to the second. For example (2, 4) * (1, 3) = (1, 3, 2, 4). “Asterisk” operations in one expression are performed in order from the leftmost to the rightmost, but this order can be changed with brackets (operations in brackets are performed earlier). E. g. (3) * ((1, 5) * (2, 7)) = (2, 7, 1, 5, 3). Notice that if a sequence element is represented by an expression, then this expression is calculated first and then all nested brackets in this sequence are removed. For example, (1, ((2) * (3)), 4) = (1, (3, 2), 4) = (1, 3, 2, 4).

Now cuckoolanders want to use this operation for generating permutations. More precisely, they want to obtain a given permutation from permutation (1, 2, …, N) by adding brackets, commas and asterisks and evaluating the resulting expression.

The formal definition of expression follows.

```
     <expression> ::= <sequence>[*<sequence>…]
     <sequence> ::= (<sequence element>[,<sequence element>…])
     <sequence element> ::= <number> | <expression>
     <number> ::= 1|2|…|N
```

Input

The first line contains an integer N (1 ≤ N ≤ 10000). The second line contains a permutation of numbers from 1 to N. These numbers are separated by spaces.

Output

Output a single line — correct expression, the result of which is the given permutation. Numbers from 1 to N should appear in ascending order. The length of the expression should not exceed 100000 symbols. In case there is no such expression output “IMPOSSIBLE”. Note that expression must not contain spaces and all sequences must be enclosed in brackets.

Sample

input	output
4 3 4 2 1	(1)*(2)*(3,4)
6 5 1 2 6 4 3	IMPOSSIBLE

Solution:

This task can be viewed as a **dynamic programming problem**. Before discussing the algorithm, one can make a few useful observations. First of all, applying asterisk operation changes the order of the arguments. Let us assume that we have a correct expression. The order of applying asterisk operations can be changed using brackets, but one can always find the first asterisk that should be applied except when there exists none. In other words, correct expression has one of the following two forms:

a) $(1,2,3,\ldots,n)$
b) $(\ldots A \ldots) * (\ldots B \ldots)$

The first form is trivial and essentially nothing needs to be done, but the second form requires some analysis. After applying the asterisk operation, arguments A and B are going to change order, so the second part of the final permutation has to match part A, and the first part with part B. The key observation is that both of these parts are segments of natural numbers. The same principle can be applied to both of these parts recursively. Similarly, one can analyze behavior when applying the comma separator. In this case, part A matches the first part of the permutation, and part B the second one.

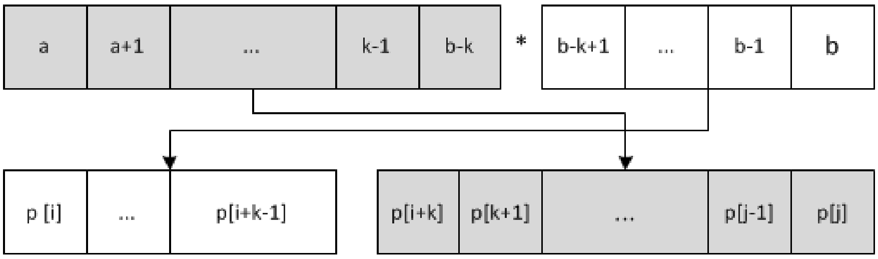


Figure 1. Matching parts after asterisk operation

One can define $d\ [i][j][a][b]$ to be $true$ if $(a, a+1, \ldots, b)$ can be transformed into $(p[i], p[i+1], .., p[j])$, where p is the given permutation, and $false$ otherwise. This is a large matrix to initialize, but another key observation is that it can be computed using greedy approach. In other words, if $d[i][j][a][b]$ is $true$ then there is a split of segment $[a, b]$ with asterisk operation in any two parts that are valid. Parts are valid if corresponding parts (permutation – segments) represent the same elements (up to the ordering). As can be seen in Figure 1, one only has to check if $(p[i], \ldots, p[i+k-1])$ is a permutation of numbers in segment $[b-k+1, b]$. Since the elements are different, this is reduced to checking that minimum is equal to $b-k+1$ and maximum to b. Similar principle can be applied in the case where comma separator is applied, but parts do not change places.

Because of the greedy approach, there is no need to compute or store the whole matrix d. Instead, recursion can be used with break condition a$= b$. First, it should be checked whether segment $[a, b]$ can be split in parts like in Figure 1. If not, it should be checked whether comma separator can be applied. If any of these attempts is successful, the same principle needs to be applied recursively to the two parts. If neither can be applied, then there is no correct expression for the given permutation.

```
===========================================================================================
       Function:      Solve (simulates d [i][j][a][b])
       Input:         i, j - segment from permutation p [i],… ,p [j]
                      a, b - segment [a,b] that has to match to p [i],… ,p [j]
       ----------------------------------------------------------------------------
       if (i == j)
              return true;

       // Asterisk operator
       minP = maxP = p [i];
       for k = 1 to j - I do
              update minP and maxP with p [k];
              if (minP = b - k + 1) and (maxP = b)
                     if (solve (i + k, j, a, b - k) && solve (i, i + k - 1, b - k + 1, b))
                            return true;
```

```
09                      else
10                              return false;

        // Comma operator
11      minP = maxP = p [i];
12      for k = 1 to j - I do
13              update minP and maxP with p [k];
14              if (minP = a) and (maxP = a + k - 1)
15                      if (solve (i, i + k - 1, a, a + k - 1) && solve (i + 1, j, a + k, b))
16                              return true;
17                      else
18                              return false;

19      return false;
=====================================================================================
```

The reconstruction of the correct expression can be made using the same principle. Once it is determined that asterisk or comma can be applied (lines 08 and 16), it is pushed to stack. Another way is to have a similar function like the one above, which will print the solution during construction (of course, one must know that a solution exists).

Solution by:
Name: ***Andreja Ilić***
School: The Faculty of Mathematics and Sciences, Niš
E-mail: ilic_andrejko@yahoo.com

Problem R2 08: Mortal Kombat (ID: 1676)

Time Limit: 1.0 second

Memory Limit: 64 MB

Once every generation, there is a tournament known as Mortal Kombat, which was designed by the Elder Gods for the main purpose to save Earthrealm from the dark forces of Outworld. If the forces of Outworld win the tournament ten consecutive times, the Emperor will be able to invade and conquer Earthrealm. Thus far, Outworld has won nine straight victories, making the upcoming tournament the tenth, and possibly final one, for the Earthrealm.

From Wikipedia, the free encyclopedia

There are N monsters and M best human fighters participating in the Mortal Kombat. According to the tournament rules, each monster should fight one of the humans (different monsters should fight different humans). If at least one monster wins, the Eathrealm will be conquered by the Emperor of the Outworld. However, the humans can choose the competitors and the order of battles.

The thunder god Raiden, protector of the Earthrealm, should choose the fighters in such a way that all Earth warriors will win their battles. For each monster and each Earth warrior it is known whether the Earth warrior can win the monster. First of all, the fighters for the first battle should be chosen.

For example, suppose that Liu Kang wants to fight Goro, but he is the only warrior able to defeat Shang Tsung, while Goro can be defeated by other warriors, such as Johnny Cage. So, even if Liu Kang will defeat Goro in the first battle, it will inevitably lead to the conquest of the Earth, because later Shang Tsung will defeat his opponent. This means that the pair Liu Kang vs. Goro should not be selected for the first fight.

Find out which pairs cannot be chosen by Raiden if he wants to save the freedom of humanity.

Input

The first line contains integers N and M (1 ≤ N ≤ 300; N ≤ M ≤ 1500). Next lines contain the binary matrix A with N rows and M columns. A_{ij} = 1 if and only if j-th Earth warrior can defeat i-th monster.

Output

Output matrix B with N rows and M columns. B_{ij} should be equal to one if the first battle cannot be held between i-th monster and j-th human, and zero otherwise.

Sample

input	output

4 4
1111
1000
1111
1111
	1000
	0111
	1000
	1000

4 5
10000
10000
10000
10000
	11111
	11111
	11111
	11111

Solution:

It is quite obvious that this problem is a variation of the **bipartite matching problem**, which can be solved using a standard **augmenting-path algorithm**.

Denote with n and m number of monsters and fighters, respectively. Now we can construct a bipartite graph $G(V, E)$ with partitions A(monsters) and B(fighters), where an edge exists between the nodes $a \in A$ and $b \in B$ if fighter b can defeat monster a. We can consider any schedule of the battles as a matching in G. Now the problem is to check for every edge $ab \in G$ if there exists a perfect matching in G with respect to A (i.e. matching of size n) such that ab belongs to that matching.

The easiest (and obvious) way to do that is to simply loop over all the edges in G, and for every edge remove its two vertices and check if the remaining graph has the matching of size $n - 1$ by applying a maximum matching algorithm. The complexity of this approach is $O(nm \cdot MatchingAlgorithm)$, which is not good enough for this problem. The key idea is to change the approach and not try to *find* perfect matching for every edge, but, instead, try to *change* some fixed perfect matching M so that the observed edge becomes part of it. Let's explain how.

Let M be an arbitrary matching of size n in our graph G. Denote by G_M an oriented graph obtained from G, in which every edge $ab \in E$ is directed from partition A to partition B if $ab \notin M$, and directed in opposite direction otherwise. Note that any path in G_M contains edges from M and $E \backslash M$ alternatively. Next theorem gives us a way to efficiently implement our approach (all notations are from our graph G):

Theorem: Let ab $(a \in A, b \in B)$ be an arbitrary edge in G and let M be an arbitrary matching of size n in G. Then there exists a perfect matching in G with respect to A which contains ab if and only if at least one of the following statements is true:

i) There exists a path in G_M from vertex b to vertex a;

ii) There exists a path in G_M from vertex b to some vertex which does not belong to M (free vertex).

Proof: (⇒) Suppose there exists a perfect matching $M`$ in $G$ which contains $ab$. Then it is obvious that we can obtain $M`$ from M by changing some (possibly all) matching pairs of vertices from A (ie. by doing some reconnections). Since the order of reconnection is irrelevant, we can choose this one: First, put ab in M(reconnect a). If $bc \in M$ for some $c \in A$, then we must reconnect c (to appropriate pair d as in$M`$). If $de \in M$ for some $e \in A$, we must reconnect $e$ and so on. $M`$ exists, which means that this alternating path of ours must end somewhere and the only possibilities for end vertices are a (to make cycle) or some free vertex (no need for new reconnection). It is not hard to see that this implies i) or ii).

(⇐) If there exists a path P from b to a in G_M (and$ab \notin M$), then $P \cup ab$ form an alternating cycle of even length. If we "invert" the edges on this cycle, in a way that we use all the edges that have not been in matching for a new matching and throw out the old ones, we get the valid matching of size n which contains ab. By analogy, if ii) holds, we have an alternating path of even length, and after we inverse that path and change match-pair of a to be vertex b, we get the requested matching. □

Notice that cases when $ab \in M$ or when b is a free vertex are covered by i) and ii). Suppose now that there exists $u \in A$, such that $ub \in M$. It is easy to see that, instead of checking for i) or ii), it suffices to check if there is a path between u and a or u and some free vertex (because bu is the only outgoing edge from b). Can we efficiently check if there is a path between any two nodes in AinG_M? The answer is yes, that is precisely what **Transitive Closure algorithm** does (it works just like **Floyd-Warshall**, but is only interested in path existence, not its length). If we denote all free vertex in G by x (we consider that set as one vertex), then after calculating transitive closure for vertex set $A \cup x$, we can answer the main question for any edge ab in $O(1)$ time, as shown in the next (C++ like) pseudo-code ($path[u][v]$ denotes existence of path inG_M):

```
bool check( int a, int b)
{
      if ($ab \in M$ or $b$ is a free vertex)
            return true;
      $u$ = neighbor of $b$ such that $ub \in M$;
      if ($path[u][a]$ || $path[u][x]$)
            return true;
      return false;
}
```

The complexity for finding arbitrary maximum matching M is $O(VE)$ if the standard augmenting-path based algorithm is used. It is easy to prove that the complexity for our graph G is $O(n^3)$, since we are interested only in the size n. For transitive closure, we need preprocessing to determine direct neighbors between vertices in A (in our case, neighbors at distance 2, since G is bipartite). It can be done in $O(E)$ by looping over all edges, because for every $b \in B$, there is at most one outgoing edge in G_M with b as start vertex. It follows that time complexity for transitive closure is $O(nm + n^3)$. After all this pre-calculation, the complexity for checking edges is linear on the number of edges, thus overall complexity of our algorithm is $O(nm + n^3)$.

Solution by:

Name: ***Nikola Milosavljević***

School: Faculty of Mathematics and Sciences, Niš

E-mail: nikola5000@gmail.com

Problem R2 09: Sniper Shot (ID: 1697)

Time Limit: 1.0 second

Memory Limit: 16 MB

There is a sniper at point S. His mission is to eliminate an enemy of the state, who rides his bicycle along a straight line from point A to point B. The bullet flies along a striaght line with infinite speed. There are n rectangular parallelepiped-shaped skyscrapers in the city. The bullet can't fly through the skyscraper but can touch its border. Of course, the sniper will make a deadly shot as soon as possible. Your task is to calculate the coordinates of the enemy at the moment of the shot.

Input

The first line contains space-separated coordinates of S: s_x, s_y, s_z ($s_z \geq 0$). The second line contains space-separated coordinates of points A and B: a_x, a_y, b_x, b_y. The enemy of the state moves on the surface of earth, so his z-coordinate is always equal to zero. The third line containts an integer n ($0 \leq n \leq 100$). Each of the following n lines contains space-separated numbers l_x, l_y, r_x, r_y, h ($l_x < r_x$; $l_y < r_y$; $h > 0$)—coordinates of the opposite corners of the bottom of the current skyscraper and its height. The sides of the skyscrapers are parallel to the corrdinate axes. All coordinates and heights are integers and don't exceed 100 by their absolute values. It is guaranteed that no two skyscrapers have common points, the point S doesn't lie inside or on the border of the skyscraper and the segment AB doesn't have common points with any of the skyscrapers.

Output

If the enemy of the state cannot be eliminated, output “Impossible”. In the other case output the coordinates of the enemy of the state precise up to 10^{-7}.

Sample

input	output

0 0 2
-4 4 4 4
2
-3 2 -1 3 10
1 -1 4 2 20
	-1.3333333333 4.0000000000

0 0 2
4 1 4 -1
1
1 -1 3 1 10
	Impossible

Solution:

We have to find the point on the line segment AB closest to A which is not sheltered by a skyscraper. First, we will find all points sheltered by a skyscraper.

Let’s note a few things about the problem:

- The bullet will fly in the plane defined by points A, B and S. We denote the plane by π.
- Every side of skyscraper shelters is an (possibly empty) interval of points on the line

segment AB.

- Instead of a skyscraper as a whole we can observe its four vertical sides independently.
- If both upper corners of a skyscraper side are placed “below” π plane, the skyscraper side doesn’t shelter any point of the line AB and we can remove it from the observations
- If only one upper corner of the skyscraper side is “below” π plane, then the line connecting those two corners intersects π. We can observe only the part of the skyscraper side from the corner that is “above” plane π to the intersection.

After removing all skyscraper sides (and the parts of skyscraper sides) “below” π, we can move problem to 2D by projecting the skyscraper sides and the sniper point to xy plane. Every skyscraper side is now represented by a line segment, say P_iR_i. For every line segment we find the intersection of lines $S'P_i$ and $S'R_i$ with the line AB (where S' is the xy-plane projection of S). If the points P_i and R_i are between S' and the corresponding intersection, then the interval between the two intersections is sheltered by the corresponding skyscraper side. The union of all such intervals gives us all sheltered points on the line AB and now it’s easy to find the solution.

Note: however, after finding the union of the intervals we have to take into consideration to which skyscraper a skyscraper side belongs. We will denote a position on the line AB by a parameter. When two sides shelter parameter intervals $< a, b >$ and $< b, c >$ then the point at the parameter value b is also sheltered if both lines belong to the same skyscraper.

The described algorithm requires:

- Finding a plane given by 3 points
- Determining whether a point is “above” or “below” given plane
- Finding a line given by 2 points in 3D
- Intersecting plane and a line in 3D
- Finding a line given by 2 points in 2D
- Intersecting two lines in 2D
- Determining the middle of 3 collinear points in 2D
- Finding union of intervals

Solution by:

Name: ***Luka Donđivić***

School: The Faculty of Electrical Engineering and Computing, Zagreb

E-mail: ldondjivic@yahoo.com

Problem R2 10: Periodic Sum (ID: 1749)

Time Limit: 1.0 second

Memory Limit: 64 MB

Let S(t) be the sum of integers represented by all substrings of the decimal representation of t. For example, S(1205) = 1 + 2 + 0 + 5 + 12 + 20 + 05 + 120 + 205 + 1205 = 1575. Note that some substrings can have leading zeros. Let F(t, k) be the number which decimal representation is obtained by repeating the decimal representation of t k times. For example, F(1205, 3) = 120512051205. Given numbers p, k and m, calculate S(F(p, k)) modulo m.

Input

The first line contains one integer p ($1 \le p < 10^{100000}$). The second line contains two space-separated integers k and m ($1 \le k, m \le 10^9$).

Output

Output the answer on a single line.

Sample

input	output
1205 3 999999999	847123538

Solution:

This is medium **number theory problem**. The main idea is to apply several times efficient algorithm for calculating the exponent a^n.

```
================================================================================
        Function:      Modular_Power
        Input:         a – the base
                       n – exponent
                       m – modulo
        -------------------------------------------------------------------
        result = 1;
        while (n > 0)
                result = (result * result) mod m;
                if (n mod 2 = 1) then
                        result = (result * a) mod m;
                n = n div 2;
        return (result);
================================================================================
```

Pseudo code for exponent calculation

The complexity of above algorithm is $O(\log n)$.

For $n = \overline{a_1 a_2 \ldots a_s}$ we have

$$S(n) = (a_1 + a_2 + \ldots + a_s) + (\overline{a_1 a_2} + \overline{a_2 a_3} + \ldots + \overline{a_{s-1} a_s}) + \ldots + \overline{a_1 a_2 \ldots a_s}$$

$$S(n) = \sum_{i=1}^{s} a_i \cdot i \cdot (1 + 10 + 10^2 + \ldots + 10^{s-i}) = \sum_{i=1}^{s} a_i \cdot i \cdot \frac{10^{s-i+1} - 1}{9}.$$

In order to calculate $S(F(x,n))$ modulo m, we will need two additional arrays for prefix and suffix sums:

$$SP(n) = a_1 + \overline{a_1 a_2} + \overline{a_1 a_2 a_3} + \ldots + \overline{a_1 a_2 \ldots a_s} = \sum_{i=1}^{s} a_i \cdot \frac{10^{s-i+1} - 1}{9}$$

$$SS(n) = a_s + \overline{a_{s-1} a_s} + \overline{a_{s-2} a_{s-1} a_s} + \ldots + \overline{a_1 a_2 \ldots a_s} = \sum_{i=1}^{s} a_i \cdot i \cdot 10^{s-i}.$$

For integer n, let $|n|$ denotes the number of digits in decimal representation of n. The following recurrent formulas for $F(x,n)$ and $F(x,n-1)$ hold:

$$S(F(x,n)) = S(F(x,n-1)) + S(x) + |F(x,n-1)| \cdot SP(x) + SS(F(x,n-1)) \cdot \left(\frac{10^{|x|} - 1}{9} - 1 \right)$$

$$SP(F(x,n)) = SS(F(x,n-1)) + SS(x) + F(x,n-1) \cdot \left(\frac{10^{|x|} - 1}{9} - 1 \right)$$

$$SS(F(x,n)) = SS(F(x,n-1)) + SS(x) \cdot 10^{|F(x,n-1)|} + |x| \cdot F(x,n-1)$$

$$F(x,n) = 10^{|x|} \cdot F(x,n-1) + x.$$

Similarly, for the double values $F(x,2n)$ and $F(x,n)$ we have:

$$S(F(x,2n)) = 2 \cdot S(F(x,n)) + |F(x,n)| \cdot SP(F(x,n)) + SS(F(x,n)) \cdot \left(\frac{10^{|F(x,n)|} - 1}{9} - 1 \right)$$

$$SP(F(x,2n)) = 2 \cdot SP(F(x,n)) + F(x,n) \cdot \left(\frac{10^{|F(x,n)|} - 1}{9} - 1 \right)$$

$$SS(F(x,n)) = SS(F(x,n))(10^{|F(x,n)|} + 1) + F(x,n) \cdot |F(x,n)|$$

$$F(x,2n) = F(x,n) \cdot (10^{|F(x,n)|} + 1).$$

Using these recurrent formulas one can design $O(\log p + \log k)$ solution. Division by 9 can be handled by considering three cases concerning the greatest common divisor of m and 9, or by establishing another recurrent formula.

The second solution is based on the fast matrix exponential calculation. For given linear recursive homogeneous sequence $T_n = c_{d-1} T_{n-1} + c_{d-2} T_{n-2} + \ldots + c_0 T_{n-d}$, let C be the following matrix

$$C = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -c_0 & -c_1 & -c_2 & \cdots & -c_{d-1} \end{pmatrix}.$$

By simple induction there holds:

$$\begin{pmatrix} T_n \\ T_{n+1} \\ \vdots \\ T_{n+d-1} \end{pmatrix} = C^n \cdot \begin{pmatrix} T_0 \\ T_1 \\ \vdots \\ T_{d-1} \end{pmatrix}.$$

For $x = \overline{a_1 a_2 ... a_s}$ let

$$S_k = a_k + \overline{a_{k-1} a_k} + ... + \overline{a_1 a_2 ... a_k} \ .$$

With the starting value $S_1 = a_1$, the following recurrent formula holds

$$S_k = 10 \cdot S_{k-1} + k \cdot a_k \ .$$

Let B_k be the sum of all numbers from $F(x,k)$ that end with the last digit (the sum of all suffixes). Similarly as in the first solution, we have $B_1 = S_s$ and

$$B_k = S_s + 10^s \cdot B_{k-1} + x \cdot s(k-1) \ .$$

Let $A_k = B_1 + B_2 + ... + B_k$. Using $A_{k+1} - A_k = B_k$ and $A_k - A_{k-1} = B_{k-1}$, we get the following:

$$A_{k+1} - A_k = S_s + 10^s(A_k - A_{k-1}) + x \cdot s(k-1)$$

$$A_{k+1} = (1+10^s)A_k - 10^s A_{k-1} + (S_s + x \cdot s(k-1))$$

By writing the same relation for A_k and subtraction, we get fourth-order homogeneous recurrent relation for A_k

$$A_k = (1+10^s)A_{k-1} - 10^s A_{k-2} + (S_s + x \cdot s(k-2))$$

$$A_{k+1} - A_k = (1+10^s)A_k - 10^s A_{k-1} - (1+10^s)A_{k-1} + 10^s A_{k-2} + x \cdot s$$

$$A_{k+1} = (2+10^s)A_k - (1+2\cdot 10^s)A_{k-1} + 10^s A_{k-2} + x \cdot s$$

$$A_k = (2+10^s)A_{k-1} - (1+2\cdot 10^s)A_{k-2} + 10^s A_{k-3} + x \cdot s$$

$$A_{k+1} - A_k = (2+10^s)A_k - (1+2\cdot 10^s)A_{k-1} + 10^s A_{k-2} - (2+10^s)A_{k-1} + (1+2\cdot 10^s)A_{k-2} - 10^s A_{k-3}$$

$$A_{k+1} = (3+10^s)A_k - (3+3\cdot 10^s)A_{k-1} + (1+3\cdot 10^s)A_{k-2} - 10^s A_{k-3} \, .$$

Furthermore, let C_k^i be the sum of all numbers from $F(x,k)$ that end with $((k-1)s+i)$-th digit. Similarly, we have $C_1^i = S_i(x)$ and

$$C_k^i = S_i + 10^i \cdot B_{k-1} + x_i \cdot s(k-1) \, ,$$

where $x_i = \overline{a_1 a_2 ... a_i}$. Next, we set

$$C_k = \sum_{i=1}^{s} C_k^i = \sum_{i=1}^{s} S_i + B_{k-1} \cdot \sum_{i=1}^{s} 10^i + s(k-1) \cdot \sum_{i=1}^{s} x_i \ .$$

The sums $D_1 = \sum_{i=1}^{s} S_i$, $D_2 = \sum_{i=1}^{s} 10^i$ and $D_3 = s \cdot \sum_{i=1}^{s} x_i$ can be calculated iteratively, and therefore $C_k = D_1 + B_{k-1} D_2 + (k-1) D_3$. Therefore,

$$S(F(x,k)) = \sum_{i=1}^{k} C_i = k \cdot D_1 + D_2 \cdot \sum_{i=1}^{k} B_{i-1} + D_3 \cdot \sum_{i=1}^{k} (i-1)$$

$$S(F(x,k)) = k \cdot D_1 + D_2 \cdot A_{k-1} + D_3 \cdot \frac{k(k-1)}{2}\ .$$

To summarize, first we need to calculate the sums D_1, D_2, D_3 iteratively and value of 10^s, together with the first four values B_1, B_2, B_3, B_4. Then apply the fast matrix exponential to find the value of A_{k-1} and finally compute $S(F(x,k))$.

Solution by:

Name: ***Andreja Ilić***
School: Faculty of Mathematics and Sciences, Niš
E-mail: ilic_andrejko@yahoo.com

Fasten your seatbelts, the adventure continues…

See you next year!

Bubble Cup Crew