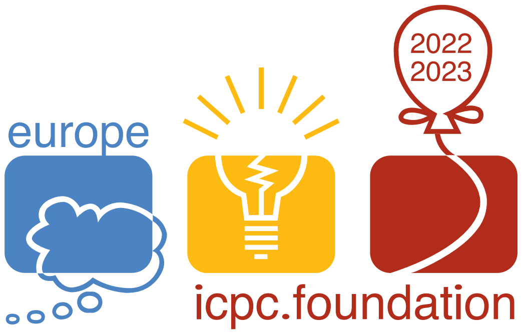
icpc international collegiate programming contest

ICPC Europe Contests

Central Europe Regional Contest

Official Problem Set



A – Bandits

Time limit: 5 s   Memory limit: 512 MiB

There is a kingdom with N villages and $N-1$ bidirectional roads that allow the citizens to travel between any pair of villages following a path consisting of one or more roads. The i-th road connects villages A_i and B_i and has length C_i.

The king has noticed an increasing number of complaints about bandits attacking the merchants travelling along the roads in the kingdom. He has tasked his advisor with solving this problem by hiring loyal groups of thugs that will act as security agencies. Each such security contract guarantees security of all roads in a radius of R_j from the village X_j with the group's headquarters. A road is protected by the contract if it is part of a path of length at most R_j from X_j to some other village. Some roads may be protected by several contracts and are therefore more secure.

Write a program that will process queries about new contracts and answer queries about the security of individual roads, that is the number of contracts currently securing that road.

Input data

The first line contains the number of villages N. The roads connecting these villages are described in the following $N-1$ lines. The description of each road consists of space-separated integers A_i, B_i and C_i, which represent a road of length C_i between villages A_i and B_i. The villages are numbered from 1 to N.

Next line contains the number of queries Q. The following Q lines describe the queries. The query that represents a new security contract starts with character `'+'` and is followed by the headquarters village X_j and security radius R_j. The query about the security of some road starts with character `'?'` and is followed by the number Y_j of that road. The roads are numbered from 1 to $N-1$ in order in which they are given in the input.

Input limits

- $1 \leq N, Q \leq 10^5$
- $0 \leq C_i, R_j \leq 10^9$

Output data

Process the queries in the given order and for every query of type `'?'` output one line with the current number of contracts securing the road Y_j.

Example

Input

```
7
1 2 4
4 2 7
5 1 3
3 6 4
1 6 9
2 7 1
7
+ 2 6
? 3
? 1
+ 6 14
? 1
? 2
? 3
```

Output

```
0
1
2
0
1
```

B – Combination Locks

Time limit: 2 s Memory limit: 256 MiB

Alice and Bob are playing with combination locks. Each of them has a combination lock that consists of N rotating discs with digits `0` to `9` engraved on them. Their friend Charlie doesn't have a lock and has devised a game to keep them occupied. He will keep track of whether the corresponding digits of their locks match and will describe the current situation with a difference pattern string S. The j-th character of S is either `'='` or `'.'` and indicates whether the j-th digits in Alice's and Bob's locks match or not, respectively.

Charlie will officiate the game, while Alice and Bob take turns with Alice starting first. On each move, a player has to change one digit of their combination lock. As Charlie only keeps track of the difference patterns, this pattern has to change for a move to be valid. He is also rather superstitious and has brought a list of patterns P_i that must not appear during the game. Charlie's main task is to enforce the rule that no difference pattern repeats during the course of the game. The player who can't make a move loses the game.

Write a program that will determine the winner of the game if both players play optimally.

Input data

The first line contains the number of test cases T. Each test case starts with a line containing two space-separated integers N and C. This is followed by two lines that describe the starting configuration of Alice's and Bob's combination lock. A lock configuration is a string of N digits. The following C lines describe Charlie's superstitious patterns P_i. The superstitious list doesn't contain duplicates and it is guaranteed that the difference pattern of the starting lock configurations is not on the superstitious list.

Input limits

- $1 \le T \le 20$
- $1 \le N \le 10$
- $0 \le C \le 1000$

Output data

For every test case output one line with the name of the winner.

Example

Input

```
3
2 2
12
89
=.
==
3 1
204
101
.==
3 2
000
000
...
==.
```

Output

```
Alice
Bob
Bob
```

Comment

In the first example, the only move for Alice is to change the second digit from 2 to 9. Any other move is invalid because it doesn't change the difference pattern or because it would result in a superstitious pattern. Bob doesn't have a valid move, therefore Alice wins.

C – Constellations

Time limit: 10 s Memory limit: 512 MiB

Astrologists took a hard scientific look at their zodiac horoscope predictions and realised that their methodology doesn't provide future insight better than chance. Instead of looking inwards they blame the stars and historical construction of constellations for their inability to predict the future. They're testing out a new way of constructing constellations that will renew their powers of future-sight.

They need your help to implement their iterative constellation creation system. Initially every star represents its own constellation. In every step you should merge two constellations into one, by picking the constellations that are closest to each other. The distance between two constellations A and B is defined as the average squared Euclidean distance of pairs of stars from each constellation:

$$d(A, B) = \frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} ||a - b||^2.$$

If multiple pairs have the same distance you should merge older constellations first. When comparing two pairs of constellations that could be merged, first compare the distances between constellations. If both pairs are at exactly the same distance, compare them by the age of the older constellation in a pair. If there is still a tie, compare them by the age of the newer constellation in a pair. A constellation's age is defined by the time when it was formed with the last merge, or in case of single-star constellations by the age of the star. The stars in the input are listed from oldest to youngest.

Input data

The first line contains N, the number of stars. The next N lines contain coordinates of stars with two space-separated integers X_i and Y_i.

Input limits

- $2 \leq N \leq 2000$
- $-1000 \leq X_i, Y_i \leq 1000$ for all $1 \leq i \leq N$
- All pairs X_i, Y_i are unique since it's physically impossible for two stars to lie on the same point.

Output data

After every step of the described constellation creation system, print out the size of the newly created constellation. You should output $N - 1$ lines.

Examples

Input

```
3
0 0
-1 0
10 10
```

Output

```
2
3
```

Input

```
4
0 0
0 -1
0 1
0 2
```

Output

```
2
2
4
```

Input

```
4
0 0
0 1
0 -1
0 2
```

Output

```
2
3
4
```

D – Deforestation

Time limit: 2 s Memory limit: 256 MiB

You want to remove a big tree from your property, but it's too big for you to carry all at once. How many pieces do you have to cut it into if the maximum weight you can carry is W?

The tree has a single trunk connected to the ground and can split out into multiple branches. All of those branches can branch out further etc. So each segment of the tree is a continuous mass of wood, which may or may not split out into multiple branches.

You can make cuts at any point on the tree; start, end, or anywhere in the middle of any segment. You can consider branching as an arbitrarily small part of the tree, i.e. you can cut immediately before or after a branch splits off without increasing the weight of the base branch, but it will affect whether the child branches are cut off as a single piece or just one branch is cut off separately.

Input data

The first line of the input will contain W, your carrying capacity. The next line will continue with the description of the first tree segment; its trunk.

A tree segment description is defined recursively. The first line contains two numbers M, weight of the segment, and N, number of branches coming out of the segment at its end. This is followed by N tree segment descriptions, describing each one of the branches.

Input limits

- $1 \leq W, M \leq 10^9$
- $0 \leq N \leq 10^5$
- Total weight of all tree segments will not exceed 10^9.
- Total number of segments will not exceed 10^5.

Output data

Output one number, the number of pieces you have to cut the tree into.

Examples

Input	Output

```
1
10 0
```
	`10`

Input	Output

```
7
5 2
7 0
2 0
```
	`2`

Input	Output

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

Comment

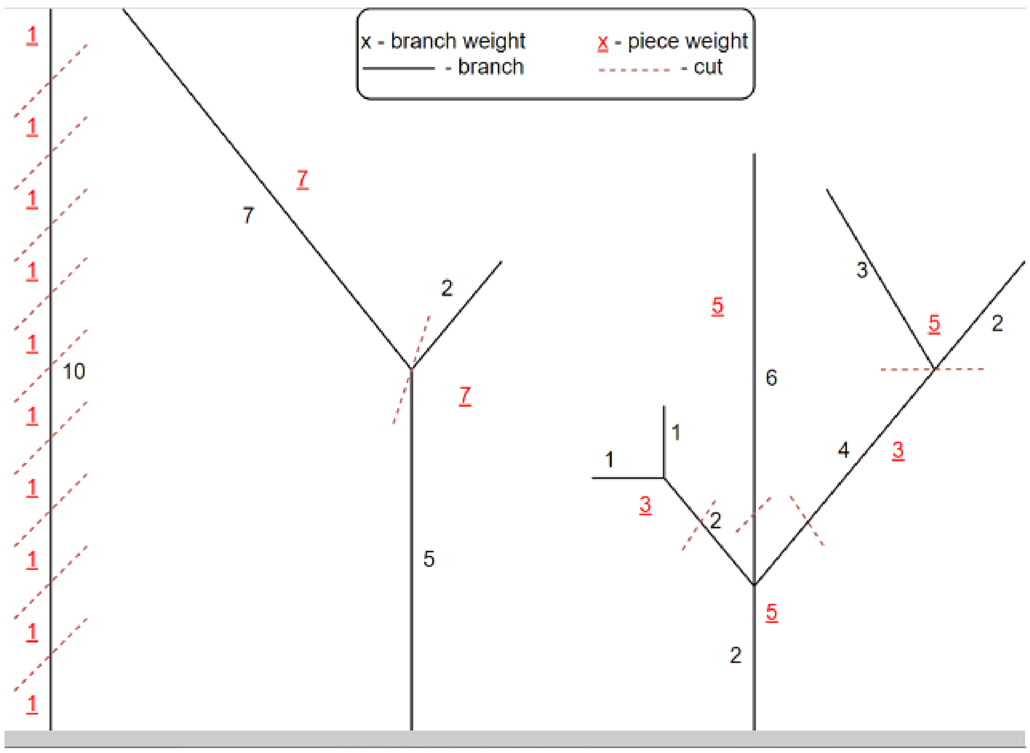


Image shows some possible solutions of sample test cases.

E – Denormalization

Time limit: 5 s Memory limit: 256 MiB

Dr. Brodnik prepared a list $A = [a_1, a_2, \dots, a_N]$ that contained N integers. No one knows what exactly these numbers represented, but it is well known that:

- $1 \leq a_i \leq 10\,000$ for all $1 \leq i \leq N$ and
- their greatest common divisor was 1.

Dr. Hočevar decided to do his colleague a favor and normalized the list, as he thought that it represents a vector in the n-dimensional real vector space. Namely, he calculated the number

$$d = \sqrt{\sum_{i=1}^{N} a_i^2} = \sqrt{a_1^2 + a_2^2 + \cdots + a_N^2}$$

and replaced Dr. Brodnik's list by $[a_1/d, a_2/d, \dots, a_N/d]$. The numbers in this normalized list were also rounded to 12 decimal places for storage. We will refer to the elements of the stored normalized list as $X = [x_1, x_2, \dots, x_N]$. After some time, he realized that it was a mistake and he now wishes to recover the original list A. Of course, no backup of the original has been made. Since Dr. Hočevar is too busy at the moment doing more important tasks, your help will be much appreciated.

As some data was lost due to rounding, he will be happy with any reconstructed list $R = [r_1, r_2, \dots, r_N]$, such that after normalization it would differ from X by at most 10^{-6} in each corresponding element.

Input data

The first line of the input contains an integer N, i.e. the length of the list X. The i-th of the following N lines contains a floating-point number x_i with exactly 12 decimal places. It is guaranteed that the input is valid, i.e. it was really obtained in the described manner from a list of integers with the properties described above.

Input limits

- $2 \leq N \leq 10\,000$
- $0 < x_i < 1$ for all $1 \leq i \leq N$

Output data

The output should contain N lines containing the reconstructed integers $r_1, r_2, \dots, r_N$ in this order. You can output any acceptable solution as described above.

Output limits

- $1 \leq r_i \leq 10\,000$ for all $1 \leq i \leq N$
- $\gcd(r_1, \dots, r_N) = 1$

Example

Input

```
6
0.137516331034
0.165019597241
0.275032662068
0.412548993102
0.825097986204
0.165019597241
```

Output

```
5
6
10
15
30
6
```

F – Differences

Time limit: 2 s *Memory limit: 256 MiB*

We have a list of N strings S_i. All strings have length M and consist only of characters `A`, `B`, `C` and `D`. Let us define the distance between two strings X and Y as the number of indices j, where the strings have different characters ($X_j \neq Y_j$). We know that the list of strings S_i contains precisely one special string that has distance K to all other strings. Note that there might be other pairs of strings with a distance of K. We are experiencing problems finding this special string, so please write a program to help us out.

Input data

The first line contains space-separated integers N, M and K. Strings S_i are given in the following N lines.

Input limits

- $2 \leq N, M \leq 10^5$
- $1 \leq K \leq M$
- $NM \leq 2 \cdot 10^7$

Output data

Output the index i of the special string. Strings are numbered from 1 to N as given in the input.

Examples

Input

```
5 10 2
DCDDDCCADA
ACADDCCADA
DBADDCCBDC
DBADDCCADA
ABADDCCADC
```

Output

```
4
```

Input

```
4 6 5
AABAAA
BAABBB
ABAAAA
ABBAAB
```

Output

```
2
```

G – Greedy Drawers

Time limit: 2 s   Memory limit: 256 MiB

Janko has N rectangular notebooks on the table. The i-th notebook has sides of length A_i and B_i. Next to the table is a chest of drawers that consists of N drawers, which have a rectangular shape but can be of different sizes. The j-th drawers has width X_j and depth Y_j. Janko wants to store each notebook in its own drawer. He can rotate the notebooks but will place them in a drawer so that the sides of the notebook are aligned with the sides of the drawer. A notebook fits into the drawer if the length of each side does not exceed the length of the corresponding aligned side of the drawer.

Janko has decided on a procedure to assign notebooks to drawers. For every notebook he will determine the number of drawers that he can fit the notebook into. Similarly, he will determine for every drawer the number of notebooks that would fit into this drawer. Then he will select the object (notebook or drawer) with the smallest number of options. If this object has no options, the procedure stops with a failure. If there are several objects with the same smallest number of options, he will select one uniformly at random. He will assign one of the options to the selected object uniformly at random. If the selected object was a notebook, he will assign it to a random drawer that can fit the notebook. If the selected object was a drawer, he will assign it to a random notebook that fits into the drawer. He will remove the assigned pair (notebook and drawer) and repeat the procedure until all notebooks are assigned to drawers.

Metka has overheard Janko's idea about placing notebooks into drawers. She is convinced that his procedure is flawed and might not succeed. Help her by writing a program that will read the number of notebooks and drawers N and output a list of notebooks and a list of drawers where Janko's random greedy method doesn't necessarily find an assignment of all notebooks to drawers although such an assignment exists.

Input data

The first and only line contains integer the number of notebooks and drawers N.

Input limits

- $150 \leq N \leq 250$

Output data

First, output N lines with space-separated notebook side lengths A_i and B_i. Next, output an empty line followed by another N lines with space-separated drawer dimensions X_j and Y_j. All dimensions should be integers between 1 and 1000, inclusive.

Evaluation

To evaluate the output of your program, we will run Janko's random greedy method on your data (notebook and drawer dimensions). Note that there must exist an assignment of all notebooks to drawers, otherwise your output will be considered as incorrect. Your solution will be evaluated on 20 test cases and Janko's method has to fail on all of them. For every test case we will run Janko's method once with a fixed random seed.

Examples

Input

```
1
```

Output

```
4 3

2 6
```

Input

```
3
```

Output

```
4 4
3 5
6 1

2 7
5 4
5 5
```

Comment

Note that the provided sample inputs and outputs are incorrect. The inputs don't respect the constraint $150 \le N$.

In the first sample, there is a single notebook which doesn't fit into the single drawer, therefore a valid assignment doesn't exist.

In the second sample, Janko's method would successfully assign all notebooks to drawers. First, it would select the last notebook (6×1) or the first drawer (2×7) and assign it to the other one because both have a single option. Now both remaining notebooks fit into both remaining drawers, therefore any assignment will do.

H – Insertions

Time limit: 1 s Memory limit: 256 MiB

We are given three strings, s, t and p. We will denote the length of a string by vertical bars, thus $|s|$ is the length of s and so on. If we *insert* t into s at position k, where $0 \le k \le |s|$, the result is a new string consisting of the first k characters of s, followed by the entirety of t, and finally followed by the remaining $|s| - k$ characters of s. We would like to select k so that the resulting new string will contain the largest possible number of occurrences of p as a substring.

Thus, for example, inserting t = `aba` into s = `ab` at position $k = 0$ results in the string `abaab`; at $k = 1$, in the string `aabab`; and at $k = 2$, in the string `ababa`. If we are interested in occurrences of p = `aba`, then the best position to insert t into s is $k = 2$, where we get two occurrences: ***aba***`ba` and `ab`***aba*** (as this example shows, occurrences of p are allowed to overlap). If, on the other hand, we were interested in occurrences of p = `aa`, then the best choices of k would be $k = 0$ and $k = 1$, which result in one occurrence of p, whereas $k = 2$ results in 0 occurrences of p.

Input data

The first line contains the string s, the second line the string t, and the third line the string p.

Input limits

- $1 \le |s| \le 10^5$
- $1 \le |t| \le 10^5$
- $1 \le |p| \le 10^5$
- All the strings consist only of lowercase letters of the English alphabet.

Output data

Output one line containing the following four integers, separated by spaces:

1. The maximum number of occurrences of p we can get after inserting t into s at position k, if we choose the position k wisely.
2. The number of different k's (from the range $0, 1, \ldots, |s|$) where this maximum number of occurrences of p is attained.
3. The minimum value of k where the maximum number of occurrences of p is attained.
4. The maximum value of k where the maximum number of occurrences of p is attained.

Examples

Input

```
ab
aba
aba
```

Output

```
2 1 2 2
```

Input

```
abaab
aba
ababa
```

Output

```
1 3 1 5
```

Input

```
eeoeo
eoe
eeo
```

Output

```
2 3 1 4
```

Comment

The first of these three examples is the one discussed earlier in the problem statement.

I – Money Laundering

Time limit: 2 s   Memory limit: 256 MiB

Consider a company A that made a 100 € of profit this year. The company's owners are Ivan with 52.8% ownership share and Robi with a 47.2% ownership share. Naturally, the profits are shared proportionally to the shares with Ivan receiving 52.8 € and Robi 47.2 €.

They will have to pay tax on the received profits, but would like to avoid doing so, if at all possible. Sadly, the ownership structure of their company is too simple and it's easily discoverable how much profits each of them received.

For the next year, they prepare a plan. They make a shell company B and change the ownership shares. Ivan now only owns 1% of company A, Robi only 2%, the company B owns a 49% share of A and A owns 48% of itself. Company B has a similar ownership structure: 70% ownership share belongs to B itself, 25% to A, 3% to Ivan and 2% to Robi.

Looking naively, Ivan and Robi have very small ownership shares. However, we are interested in the ownership shares of *ultimate beneficial owners*, the persons who will ultimately profit, which are Ivan and Robi in our case. We wish to determine their ultimate ownership shares, which turn out to be approximately equal to what they were before the introduction of B.

Ultimate ownership shares can be determined as follows: let the company A have 100 € of profit and B have 0 €. The profits are paid out to all direct owners in proportion to their ownership share. However, since A and B are partial owners of themselves, they receive a part of the profit. To determine the ultimate share of the ultimate beneficial owners, we repeat the procedure – any profits that A and B receive are paid out again, with Ivan and Robi getting a share, as well as A and B. This is repeated ad infinitum until (theoretically, after an infinite number of steps) all money is paid out to the ultimate beneficial owners, and the ratio of the final sums received by Ivan and Robi is by definition equal to their ultimate share of A.

For a given structure of companies, determine the shares of the ultimate beneficial owners. However, the companies do not form a random network of ownership, but are structured in industrial sectors. Companies within sectors may form arbitrary ownership structures, but this is not true for companies in different sectors. If companies P and Q belong to different sectors, it cannot happen that

- P would own a (potentially indirect) share of Q and
- Q would own a (potentially indirect) share of P.

One or none of these statements could be true, but not both.

Input data

The first line contains two space-separated integers c and p, representing the number of companies and number of persons, respectively. Then c lines follow, and i-th of them contains the description of i-th company. The line contains an integer k_i, the number of owners, and then k_i entries of the form $o_{i,j} : p_{i,j}$, where $o_{i,j}$ is the designation of the j-th owner (person or company) and $p_{i,j}$ is their share in percentages. The share will have exactly one decimal place.

Input limits

- $1 \leq c, p \leq 10^3$
- $1 \leq \sum_{i=1}^{n} k_i \leq 10^4$
- $o_{i,j}$ can have two forms: `P`x or `C`y, indicating that the owner is the x-th person or y-th company, respectively. It is guaranteed that $1 \leq x \leq p$, and $1 \leq y \leq c$ holds.
- $k_i \geq 1$
- $0 < p_{i,j} \leq 100$
- $\sum_{j=1}^{k_i} p_{i,j} = 100$
- The identifiers $\{o_{i,j}\}_{j=1}^{k_i}$ are unique, i.e. each owner is listed at most once.
- The number of companies belonging to each sector is less than 10.
- Each company has at least one ultimate beneficial owner. For example, a scheme where A would own a 100% of B and B a 100% of A is forbidden.

Output data

Output the ultimate ownership shares of all persons in all companies. The i-th line should include shares of all persons in the i-th company, including persons with no share. The share is between 0 and 1. Shares in a line should be separated by a space. The answer will be considered correct if its absolute or relative error is less than 10^{-4}.

Examples

Input

```
2 2
2 P1:50.1 P2:49.9
2 P1:23.4 P2:76.6
```

Output

```
0.501000 0.499000
0.234000 0.766000
```

Input

```
2 2
2 P1:50.0 P2:50.0
3 P1:20.0 P2:30.0 C1:50.0
```

Output

```
0.500000 0.500000
0.450000 0.550000
```

Input

```
2 2
4 P1:1.0 P2:2.0 C2:49.0 C1:48.0
4 C2:70.0 C1:25.0 P1:3.0 P2:2.0
```

Output

```
0.528358 0.471642
0.540299 0.459701
```

Input

```
3 2
5 P1:1.0 P2:2.0 C2:49.0 C1:38.0 C3:10.0
4 C2:70.0 C1:25.0 P1:3.0 P2:2.0
2 P1:20.0 P2:80.0
```

Output

```
0.373228 0.626772
0.411024 0.588976
0.2 0.8
```

J – Mortgage

Time limit: 3 s Memory limit: 512 MiB

Andrej is a typical modern student, dreaming to buy a house one day. Since buying real property is no piece of cake, he is planning out his life and trying to figure out exactly how and when he will be able to afford one. To buy a house, he aims to take a mortgage loan that will then need to be paid back in multiple payments over the course of several months. For each of the next n months of his life, he will earn the income a_i that can be spent on the mortgage (other costs have already been accounted for, hence a_i can be negative). He is now looking at a list of various properties and mortgage loans and is trying to figure out which of them he can afford.

Suppose that he takes a mortgage that involves paying x units of money over the course of k months, starting in month i, and ending in month $i + k - 1$. Each of these months, he needs to be able to pay x units of money. If he has any leftover income in month i, i.e. $a_i > x$, he can save the rest and use it towards some of the future payments (same for any leftover money in months $i + 1$ to $i + k - 1$). However, he cannot count on saving any money prior to month i, regardless of the income in those months. He will spend it all on his current rent and avocado toast.

You are given the list of Andrej's income for the next n months and a list of m different time intervals. The i-th time interval is defined by two numbers, s_i, and k_i. The mortgage loan starts on the month s_i and lasts for k_i months, i.e. the last payment is done on the month $s_i + k_i - 1$. For each of the time intervals, determine what the largest monthly payment that Andrej can afford is.

Input data

The first line contains two integers, n and m, the number of months, and the number of different time intervals, respectively. The second line contains n space-separated integers, $a_1, \ldots, a_n$, Andrej's income over the next n months. This is followed by m lines describing different time intervals, each line containing two space-separated integers s_i and k_i.

Input limits

- $1 \leq n, m \leq 2 \cdot 10^5$
- $-10^9 \leq a_i \leq 10^9$
- $1 \leq s_i \leq n; \forall i$
- $1 \leq k_i$ and $s_i + k_i - 1 \leq n; \forall i$

Output data

Print out m lines, one for each time interval. Print out the largest integer amount of monthly payment that Andrej can afford to pay for the i-th mortgage. If the number is strictly smaller than 0, print `"stay with parents"` (without quotation marks).

Example

Input

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

Output

```
4
3
8
stay with parents
0
```

Comment

For the first interval, a monthly payment of 4 units is the largest Andrej can afford. For a monthly payment of 5, he would run out of money for his final payment. Negative income on month 6 means that Andrej cannot afford any mortgage for interval 4, regardless of its size.

K – Skills in Pills

Time limit: 1 s   Memory limit: 256 MiB

An unnamed protagonist of this task received amazing e-mail offers for wondrous pills that will enhance their cognitive and all other sorts of abilities. After carefully analysing all offers and side effects, he has decided that he will order 2 types of pills, let's call them A and B. He needs to take pill A every k days and pill B every j days. He will follow this meticulously over the next n days.

More formally, in the next n days, there should be no k consecutive days where he does not take pill A and no j consecutive days where pill B is not taken. However, there is a twist – the two pills are highly potent and must not be taken on the same day, lest horrible side effects should happen. Given this constraint, what is the smallest number of pills that he needs to take to meet these requirements?

Input data

You are given three space-separated integers, k, j, and n.

Input limits

- $2 \leq n \leq 10^6$
- $2 \leq k, j \leq n$

Output data

Print one number – the minimum number of pills that need to be taken. It is easy to prove that a solution always exists for the given constraints.

Examples

Input **Output**

```
2 3 8
```

```
6
```

Input **Output**

```
2 3 11
```

```
9
```

Input **Output**

```
3 7 100
```

```
48
```

Comment

In the first case, we can take pill A on days 2, 4, 5, and 7, and pill B on days 3 and 6, giving the sequence `.ABAABA.` In the second case, the best approach is to take pills in sequence `.ABAABAABA.` which requires taking 9 pills.

L – The Game

Time limit: 1 s   Memory limit: 256 MiB

Vladimir is the loneliest child in the neighbourhood. No other kid likes to play with him. His parents decided to cheer him up so they bought him a card game called *The Game*. This card game is for up to 5 players, but it can also be played in the *solo* (i.e. single-player) mode.

The package contains 98 *regular* playing cards that are labeled by integers $2, 3, \ldots, 99$. In addition to these, there are 4 special *direction* cards. Two of them are labeled with the number 1 (followed by an up arrow) and the other two are labeled with 100 (followed by a down arrow).

In the initial phase of the game, the pile of regular cards is shuffled and placed face down on the table – this will be the *draw* pile. The four direction cards are placed in a column; the two cards labeled 1 have to be at the top. There should also be enough space on the right-hand side of each direction card – this is where regular cards will be laid during the play. The card labeled 1 initiates an *ascending row*, while a card labeled 100 initiates a *descending row*. In the solo mode, the player draws the top 8 cards from the draw pile, one by one, and puts them in his hand.

After the initial phase the game starts. On each turn the player has to play two cards from his hand according to the following rules:

- A card may be placed at the end of an ascending row if it is larger than the last (i.e. right-most) card in the row.
- A card may be placed at the end of a descending row if it is smaller than the last card in the row.
- A card with a smaller label may be placed at the end of an ascending row, or a card with a larger label may be place at the end of a descending row, if the absolute difference between its value and the value of the last card in the row is exactly 10. This move is called the *backwards trick*. Note that because of this extra rule, the values of the cards in an *ascending row* are not necessarily ascending (and similarly, the values of the cards in a *descending row* are not necessarily descending).

After playing two cards from the hand, the player should draw two new cards from the draw pile, one by one. This concludes his turn. If the draw pile is empty, he continues playing in the same way without drawing new cards. The game ends when the player has no cards left in his hand (in that case the player *beats the game*) or if he cannot play any of the remaining cards in his hand (in that case the player has *lost the game*).

Example: Suppose that the player's initial hand (i.e. the first 8 cards which he has drawn) is:

```
69, 17, 59, 32, 31, 77, 87, 89
```

He may decide to play the card 89 (putting it in the first descending row) and the card 17 (putting it in the second ascending row). The state of all four rows after the move is:

```
1 ->
1 -> 17
100 <- 89
100 <-
```

Then he has to pick up two more cards from the draw pile – suppose these two cards are 84 and 3 – and his hand becomes:

```
69, 59, 32, 31, 77, 87, 84, 3
```

In the second turn he might want to play the card 3 (in the first ascending row) and card 87 (in the first descending row, after card 89). The state of all four rows after the move:

```
1 -> 3
1 -> 17
100 <- 89, 87
100 <-
```

Vladimir played the game for a few times and he could not always beat the game. Since he hates losing the game, you should write a computer program that will inspect the draw pile and predict the outcome of the game. This will help Vladimir to decide whether he wants to play it or not.

You should also know that Vladimir is a very logical and predictable person. He plays according to the following rules.

- When he draws a card, he places it in his hand on the far-right side.
- He will always play a card from his hand according to his list of priorities:
  1. If one or more cards allow him to do the backward trick, he will use the leftmost such card. If that card can be used for the backward trick in different rows, he will use the top-most amongst these rows.
  2. Otherwise, he plays a card in the regular way. He will select the card to play, and the row in which to put it, in such a way as to minimize the absolute value of the difference between the value of the card that is being played and the last card in the row. If several cards attain the minimum, he will use the left-most amongst these cards. Finally, if there are several choices of where to play this card, he will choose the top-most row.

Your program should find the final state of the game.

Input data

The first (and only) line of the input contains 98 space-separated integers, i.e. some permutation of the set $\{2, 3, \ldots, 99\}$ that represents the initial draw pile. The cards are listed in order from top to bottom of the draw pile.

Output data

The output contains six lines. The first four lines describe the four rows of cards on the table. The fifth line lists the cards that remained in the player's hand (if any) while the last line lists the cards that remained in the draw pile (if any). Print an empty line in case of an empty list. Cards in the four rows and in the hand should be ordered from left to right, while the cards in the last line, which represents the remainder of the draw pile, should be ordered from top to bottom as in the input data. See also the sample outputs.

Examples

Input

```
96 69 40 94 35 7 53 88 10 89 47 37 16 61 24 46 90 6 33 25 63 73 26 81 2 45
    77 75 48 57 66 34 59 92 44 11 31 18 9 52 91 50 8 98 5 64 86 62 83 4 19
    3 27 97 28 36 23 76 58 30 38 12 39 78 41 56 80 67 70 99 13 42 17 49 84
      22 32 29 54 71 51 74 79 95 72 15 87 21 65 68 60 85 55 43 93 20 82 14
```

Output

```
1 7 10 16 6 9 11 18 31 62 64 83 86 91 92 97 98 99
1 2 5 8 19 23 27 28 30 36 38 39 41 56 58 67 70 76 78 80 84 74 79 95
100 96 94 89 88 69 61 53 47 46 40 37 35 33 26 25 24 34 44 42 22 32 29 17
                                                                  13 12 4 3
100 90 81 77 75 73 66 63 59 57 52 50 48 45 21 15
49 54 71 51 72 87 65 68
60 85 55 43 93 20 82 14
```

Input

```
87 31 58 56 82 93 9 68 65 41 26 64 3 11 5 84 24 46 16 30 14 85 52 12 91 75
   96 17 47 37 76 69 78 49 25 28 48 81 95 63 34 43 27 74 80 62 53 83 40 71
     72 35 23 21 51 66 55 61 67 32 38 29 60 39 4 18 20 77 7 94 59 42 79 10
      92 97 57 2 86 33 89 90 88 19 22 99 45 44 73 70 50 6 15 98 54 13 36 8
```

Output

```
1 9 11 16 24 14 17 26 28 30 31 34 62 74 78 80 81 71 72 83 95 96 97 99
1 3 5 12 25 27 29 38 39 42 59 60 66 67 57 77 79 86 89 90 92 94 98 88
100 93 87 82 68 65 64 58 56 46 41 37 47 43 53 51 61 55 45 44 33 22 20 19
                                                               15 13 10 8 6
100 91 85 84 76 75 69 63 52 49 48 40 35 32 23 21 18 7 4 2
73 70 50 54 36
```

Note. The inputs and outputs contain lines that do not fit on an A4 page. If a line starts with one or more spaces, it is in fact continuation of the previous line. The actual data (see the judge system) does not contain these line-breaks.