Problem A. Non Absorbing DFA

Input file:	`dfa.in`
Output file:	`dfa.out`
Time limit:	0.5 seconds

In the theory of compilers and languages *finite state machines*, also known as *finite automata* are widely used. Deterministic finite automation (DFA) is an ordered set $\langle \Sigma, U, s, T, \varphi \rangle$ where Σ is the finite set called *input alphabet*, U is the finite set of *states*, $s \in U$ is the *initial state*, $T \subset U$ is the set of *terminal states* and $\varphi : U \times \Sigma \to U$ is the *transition function*.

The input of the automation is the string α over Σ. Initially the automation is in state s. Each step it reads the first character c of the input string and changes its state to $\varphi(u, c)$ where u is the current state. After that the first character of the input string is removed and the step repeats. If when its input string is empty the automation is in the terminal state, it is said that it *accepts* the initial string α, in the other case it *rejects* it.

In some cases to simplify the automation the concept of *nonabsorbing edges* is introduced. That is, in addition to φ the function $\chi : U \times \Sigma \to \{0, 1\}$ is introduced and when making a transition from some state u with some character c, the leading character is removed from the input string only if $\chi(u, c) = 0$. If $\chi(u, c) = 1$, the input string is kept intact and next transition is performed with the new state and the same character.

It is said that such automation accepts some string α if after a number of steps it transits to the terminal state and the input string becomes empty.

Your task is given the DFA with nonabsorbing edges to compute the number of strings of the given length N that it accepts.

Input

The first line of the input file contains Σ — a subset of the English alphabet, several different small letters. Next line contains $K = |U|$ — the number of states of the automation ($1 \le K \le 1\,000$). Let states be numbered from 1 to K. Next line contains S ($1 \le S \le K$) — the initial state, followed by $L = |T|$ — the number of terminal states and then L different integer numbers ranging from 1 to K — the numbers of terminal states.

Next K lines contain $|\Sigma|$ integer numbers each and define φ. Next K lines define χ in a similar way.

The last line of the input file contains N ($1 \le N \le 60$).

Output

Output the only number — the number of different strings of length N over Σ that the given DFA accepts.

Example

dfa.in	dfa.out
ab 2 1 1 2 2 1 1 2 0 1 0 0 3	2

In the given example the two strings accepted by the automation are "`aaa`" and "`abb`".

Problem B. The Towers of Hanoi Revisited

Input file:	`hanoi.in`
Output file:	`hanoi.out`
Time limit:	0.5 seconds

You all must know the puzzle named "The Towers of Hanoi". The puzzle has three pegs and N discs of different radii, initially all disks are located on the first peg, ordered by their radii — the largest at the bottom, the smallest at the top. In a turn you may take the topmost disc from any peg and move it to another peg, the only rule says that you may not place the disc atop any smaller disk. The problem is to move all disks to the last peg making the smallest possible number of moves.

There is the legend that somewhere in Tibet there is a monastery where monks tirelessly move disks from peg to peg solving the puzzle for 64 discs. The legend says that when they finish, the end of the world would come. Since it is well known that to solve the puzzle you need to make $2^N - 1$ moves, a small calculation shows that the world seems to be a quite safe place for a while.

However, recent archeologists discoveries have shown that the things can be a bit worse. The manuscript found in Tibet mountains says that the puzzle the monks are solving has not 3 but M pegs. This is the problem, because when increasing the number of pegs, the number of moves needed to move all discs from the first peg to the last one following the rules described, decreases dramatically.

Calculate how many moves one needs to move N discs from the first peg to the last one when the puzzle has M pegs and provide the scenario for moving the discs.

Input

Input file contains N and M ($1 \le N \le 64$, $4 \le M \le 65$).

Output

On the first line output L — the number of moves needed to solve the puzzle. Next L lines must contain the moves themselves. For each move print the line of the form

```
move <disc-radius> from <source-peg> to <target-peg>
```

if the disc is moved to the empty peg or

```
move <disc-radius> from <source-peg> to <target-peg> atop <target-top-disc-radius>
```

if the disc is moved atop some other disc.

Disc radii are integer numbers from 1 to N, pegs are numbered from 1 to M.

Example

hanoi.in	hanoi.out
5 4	13 move 1 from 1 to 3 move 2 from 1 to 2 move 1 from 3 to 2 atop 2 move 3 from 1 to 4 move 4 from 1 to 3 move 3 from 4 to 3 atop 4 move 5 from 1 to 4 move 3 from 3 to 1 move 4 from 3 to 4 atop 5 move 3 from 1 to 4 atop 4 move 1 from 2 to 1 move 2 from 2 to 4 atop 3 move 1 from 1 to 4 atop 2

Problem C. Hyperhuffman

Input file:	`huffman.in`
Output file:	`huffman.out`
Time limit:	0.5 seconds

You might have heard about Huffman encoding — that is the coding system that minimizes the expected length of the text if the codes for characters are required to consist of an integral number of bits.

Let us recall codes assignment process in Huffman encoding. First the *Huffman tree* is constructed. Let the alphabet consist of N characters, i-th of which occurs P_i times in the input text. Initially all characters are considered to be active nodes of the future tree, i-th being marked with P_i. On each step take two active nodes with smallest marks, create the new node, mark it with the sum of the considered nodes and make them the children of the new node. Then remove the two nodes that now have parent from the set of active nodes and make the new node active. This process is repeated until only one active node exists, it is made the root of the tree.

Note that the characters of the alphabet are represented by the leaves of the tree. For each leaf node the length of its code in the Huffman encoding is the length of the path from the root to the node. The code itself can be constrcuted the following way: for each internal node consider two edges from it to its children. Assign 0 to one of them and 1 to another. The code of the character is then the sequence of 0s and 1s passed on the way from the root to the leaf node representing this character.

In this problem you are asked to detect the length of the text after it being encoded with Huffman method. Since the length of the code for the character depends only on the number of occurences of this character, the text itself is not given — only the number of occurences of each character. Characters are given from most rare to most frequent.

Note that the alphabet used for the text is quite huge — it may contain up to 500 000 characters.

Input

The first line of the input file contains N — the number of different characters used in the text ($2 \le N \le 500\,000$). The second line contains N integer numbers P_i — the number of occurences of each character ($1 \le P_i \le 10^9$, $P_i \le P_{i+1}$ for all valid i).

Output

Output the length of the text after encoding it using Huffman method, in bits.

Example

huffman.in	huffman.out
3 1 1 4	8

Problem D. Little Jumper

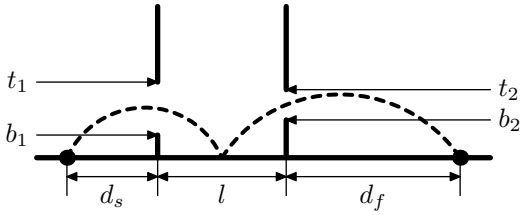
Input file:	`jumper.in`
Output file:	`jumper.out`
Time limit:	0.5 seconds

Little frog Georgie likes to jump. Recently he have discovered the new playground that seems the perfect place to jump.

Recently the new jumping exercise has become very popular. Two vertical walls are placed on the playground, each of which has a hole.

The lower sides of the holes in the walls are on heights b_1 and b_2 respectively, and upper sides on heights t_1 and t_2. Walls are parallel and placed on distance l from each other.

The jumper starts at the distance d_s from the first wall. It jumps through the first hole and lands between the walls. After that from that point he jumps through the second hole. The goal is to land exactly at the distance d_f from the second wall.



Let us describe the jump. The jumper starts from the specified point and starts moving in some chosen direction with the speed not exceeding some maximal speed v, determined by the strength of the jumper. The gravity of g forces him down, thus he moves along the parabolic trajectory.

The jumper can choose different starting speeds and different directions for his first and second jump.

Of course, The jumper must not attempt to pass *through* the wall, although it is allowed to *touch* it passing through the hole, this does not change the trajectory of the jump. The jumper is not allowed to pass through both holes in a single jump.

Find out, what must be the maximal starting speed of the jumper so that he could fulfil the excersise.

Input

Input file contains one or more lines, each of which contains eight real numbers, separated by spaces and/or line feeds. They designate b_1, t_1, b_2, t_2, l, d_s, d_f and g. All numbers are in range from 10^{-2} to 10^3, $t_1 \geq b_1 + 10^{-2}$, $t_2 \geq b_2 + 10^{-2}$.

Input file contains at most 1000 test cases.

Output

For each line of the input file output the smallest possible maximal speed the jumper must have to fulfil the exercise. If it is impossible to fulfil it, output -1. Your answer must be accurate up to 10^{-4}.

Example

jumper.in	jumper.out
0.3 1.0 0.5 0.9 1.7 1.2 2.3 9.8 0.6 0.8 0.6 0.8 2.4 0.3 1.5 0.7	5.2883 1.3127

Problem E. Quantization Problem

Input file:	`quant.in`
Output file:	`quant.out`
Time limit:	0.5 seconds

When entering some analog data into a computer, this information must be quantized. Quantization transforms each measured value x to some other value $l(x)$ selected from the predefined set L of levels. Sometimes to reduce the influence of the levels set to the information, the group of levels sets L_i is used. The number of levels sets is usually chosen to be the power of 2.

When using the number of levels sets, some additional information should be used to specify which set was used for each quantization. However, providing this information may be too expensive — the better solution would be to choose more levels and use one set. To avoid the specification of the quantization set, the following technique is used. Suppose that n values $x_1, x_2, \ldots, x_n$ are to be quantized and the group of $m = 2^p$ levels sets $\{L_i\}_{i=0}^{m-1}$ each of size $s = 2^q$ is used to quantize it. After quantization x_j is replaced with some number $l_j =\in L_{f(j)}$. Instead of sending l_j, its ordinal number in $L_{f(j)}$ is usually sent, let k_j be the ordinal number of l_j in $L_{f(j)}$ (levels are numbered starting with 0). Take p least significant bits of k_j and say that the number $k_j \& (2^p - 1)$ is the number of the levels set that will be used for next quantization, that is $f(j+1) = k_j \& (2^p - 1)$.

Since the least significant bits of k_j are usually distributed quite randomly, the sets used for quantization change often and weakly depend on values of quantized data, thus this technique provides the good way to perform the quantization.

Usually to perform the quantization the closest to the value level of the levels set is chosen. However, using the technique described, sometimes it pays off to choose not the optimal level, but some other one, the ordinal number of which has other least significant bits, thus choosing another levels set for next measure and providing better approximation of quantized values in the future. Let us call the *deviation* of quantization the value $\sum_{j=1}^{n} |x_j - l_j|$. Your task is given measures and levels sets to choose quantized value for each measure in such a way, that the deviation of quantization is minimal possible.

The first value is always quantized using set L_0.

Input

The first line of the input file contains n ($1 \le n \le 1000$). The second line contains n integer numbers x_i ranging from 1 to 10^6. The next line contains m and s ($1 \le m \le 128$, $m \le s \le 128$). Next m lines contain s integer numbers each — levels of the quantization sets given in increasing order for each set, all levels satisfy $1 \le level \le 10^6$.

Output

First output the minimal possible deviation of the quantization. Then output n integer numbers in range from 0 to $s - 1$. For each input value output the number of the level in the corresponding levels set (k_j) used for this number to achieve the quantization required.

Example

quant.in	quant.out
3 8 8 19 2 4 5 10 15 20 3 7 13 17	5 1 1 3

Problem F. Roads

Input file:	`roads.in`
Output file:	`roads.out`
Time limit:	0.5 seconds

The kingdom of Farland has N cities connected by M roads. Some roads are paved with stones, others are just country roads. Since paving the road is quite expensive, the roads to be paved were chosen in such a way that for any two cities there is exactly one way to get from one city to another passing only the stoned roads.

The kingdom has a very strong bureaucracy so each road has its own ordinal number ranging from 1 to M: the stoned roads have numbers from 1 to $N-1$ and other roads have numbers from N to M. Each road requires some money for support, i-th road requires c_i coins per year to keep it intact. Recently the king has decided to save some money and keep financing only some roads. Since he wants his people to be able to get from any city to any other, he decided to keep supporting some roads in such a way, that there is still a path between any two cities.

It might seem to you that keeping the stoned roads would be the good idea, however the king did not think so. Since he did not like to travel, he did not know the difference between traveling by a stoned road and travelling by a muddy road. Thus he ordered you to bring him the costs of maintaining the roads so that he could order his wizard to choose the roads to keep in such a way that the total cost of maintaining them would be minimal.

Being the minister of communications of Farland, you want to help your people to keep the stoned roads. To do this you want to fake the costs of maintaining the roads in your report to the king. That is, you want to provide for each road the fake cost of its maintaining d_i in such a way, that stoned roads form the set of roads the king would keep. However, to lower the chance of being caught, you want the sum $\sum|c_i - d_i|$ to be as small as possible.

You know that the king's wizard is not a complete fool, so if there is the way to choose the minimal set of roads to be the set of the stoned roads, he would do it, so ties are allowed.

Input

The first line of the input file contains N and M ($2 \le N \le 60$, $N-1 \le M \le 400$). Next M lines contain three integer numbers a_i, b_i and c_i each — the numbers of the cities the road connects ($1 \le a_i \le N$, $1 \le b_i \le N$, $a_i \ne b_i$) and the cost of maintaining it ($1 \le c_i \le 10\,000$).

Output

Output M lines — for each road output d_i that should be reported to be its maintainance cost so that the king would choose first $N-1$ roads to be the roads to keep and the sum $\sum|c_i - d_i|$ is minimal possible.

Example

roads.in	roads.out
4 5 4 1 7 2 1 5 3 4 4 4 2 5 1 3 1	4 5 4 5 4

Problem G. Robbers

Input file:	`robbers.in`
Output file:	`robbers.out`
Time limit:	0.5 seconds

N robbers have robbed the bank. As the result of their crime they chanced to get M golden coins. Before the robbery the band has made an agreement that after the robbery i-th gangster would get X_i/Y of all money gained. However, it turned out that M may be not divisible by Y.

The problem which now should be solved by robbers is what to do with the coins. They would like to share them fairly. Let us suppose that i-th robber would get K_i coins. In this case unfairness of this fact is $|X_i/Y - K_i/M|$. The total unfairness is the sum of all particular unfairnesses. Your task as the leader of the gang is to spread money among robbers in such a way that the total unfairness is minimized.

Input

The first line of the input file contains numbers N, M and Y ($1 \le N \le 1000, 1 \le M, Y \le 10000$). N integer numbers follow - X_i ($1 \le X_i \le 10000$, sum of all X_i is Y).

Output

Output N integer numbers — K_i (sum of all K_i must be M), so that the total unfairness is minimal.

Example

robbers.in	robbers.out
3 10 4 1 1 2	2 3 5

Problem H. Toral Tickets

Input file:	`tickets.in`
Output file:	`tickets.out`
Time limit:	0.5 seconds

On the planet Eisiem passenger tickets for the new mean of transportation are planned to have the form of tores.

Each tore is made of a single rectangular black rubber sheet containing $N \times M$ squares. Several squares are marked with white, thus encoding the ticket's source and destination.

When the passenger buys the ticket, the ticket booking machine takes the rubber sheet, marks some squares to identify the route of the passenger, and then provides it to the passenger. The passenger next must glue the ticket.

The ticket must be clued the following way. First two its sides of greater length are glued together, forming a cylinder. Next cylinder base circles, each of which has the length equal to the length of the short side of the original rubber sheet, are glued together. They must be glued in such a way, that the cells, sides of which are glued, first belonged to the same row of the sheet.

The resulting tore is the valid ticket.

Note that if the original sheet is square, there are two topologically different ways to make a tore out of a rubber sheet.

Ticket material is so perfect and gluing quality is so fine, that no one is able to find the seam, and this leads to some problems. First, the same tore can be obtained using different sheets. More of that, the same sheet can lead to tores that look a bit different.

Now the transport companies of Eisiem wonder, how many different routes they can organize, so that the following conditions are satisfied:

- tickets for different routes are represented by different tores;
- if some rubber sheet was marked to make the tore for some route, it cannot be used to make the tore for another route.

Help them to calculate the number of routes they can organize.

Input

The first line of the input file contains N and M ($1 \le N, M \le 20$).

Output

Output the number of routes Eisiem transport companies can organize.

Example

tickets.in	tickets.out
2 2	6
2 3	13