Problem A. Airport Coffee

Source file name:	airportcoffee.c, airportcoffee.cpp, airportcoffee.java, airportcoffee.py
Input:	`Standard`
Output:	`Standard`

Jonna often travels to programming contests by airplane. Since she lives in Helsinki, she often has to first travel to some large airport hub, such as Copenhagen Airport, where she takes a new flight. Unfortunately, flights are often very late. This is especially problematic when taking a connecting flight.

As it happens, Jonna just landed at Copenhagen Airport, trying to make her connection to Heathrow Airport. Since her flight from Helsinki was delayed, she must walk very quickly from her arrival gate to the new departure gate. Normally, Jonna walks at a speed of a centimeters per second. To make matters more difficult, Jonna has a slight coffee addiction, and will walk very sluggishly while not drinking coffee. While the coffee itself does not really affect the walking speed, the resulting grumpiness from not drinking coffee trumps even the worries of a missed flight. When she is drinking coffee, her speed increases to b centimeters per second.

The distance between Jonna's arrival and departure gates is ℓ centimeters, and along the way there are n small coffee carts where Jonna can buy a cup of coffee. When buying a cup of coffee (a practically instant endeavour nowadays, thanks to contactless card payments), she first waits for t seconds, in order to let it cool down. During this time, she will keep walking at the slower pace. Immediately after t seconds pass, she starts drinking the coffee. It takes exactly r seconds to finish the coffee (during which she walks at the faster pace). When the coffee is finished, she will again walk slower.

Note that Jonna is carrying a bag with her left hand, so she can only carry a single cup of coffee at a time. While a bit wasteful, she may throw away a cup that still contains some amount of coffee to purchase a brand new cup.

Can you help Jonna determine where to purchase her coffee(s), in order to get to her departure gate as quickly as possible?

Input

The first line of input contains five integers ℓ, a, b, t and r, where:

- $1 \leq \ell \leq 10^{11}$ is the distance between Jonna's arrival and departure gates in centimeters.
- $1 \leq a < b \leq 200$ are Jonna's walking speeds in centimeters per second when she is not and when she is drinking coffee, respectively.
- $0 \leq t \leq 300$ is the number of seconds Jonna must wait until she can drink her coffee.
- $1 \leq r \leq 1200$ is the number of seconds it takes for Jonna to drink a cup of coffee.

Then follows a line containing an integer $0 \leq n \leq 500\,000$, the number of coffee carts between the two gates. The third and last line of input contains n integers – the positions of the coffee carts, given in ascending distance from the departure gate in centimeters (i.e., each number is between 0 and ℓ, inclusive). No two coffee carts are in the same postion.

Output

First, output a line containing the number of carts where Jonna should purchase coffee. Next, output a single line containing the indices of the coffee carts where Jonna should buy coffee. These indicies should be between 0 and $n-1$, and correspond to the order of the coffee carts in the input. The indices may be output in any order, but each index must be output at most once.

Your answer will be accepted if the time that the proposed coffee purchasing plan takes is within an absolute or relative error of at most 10^{-9} compared to the optimum time.

Illustration of Sample Input 1 and a possible solution. The coffee shops Jonna uses are marked with triangles. The portions where she walks faster due to the effects of coffee are marked with a dotted line. The first coffee cools down 11 000 centimeters from the starting position, and the second after 61 000 centimeters from the starting position.

Example

Input	Output
100000 100 138 60 300 5 5000 20000 50000 55000 75000	2 0 3
100000 78 86 9 560 4 13505 69705 87448 92090	2 0 1

Problem B. Best Relay Team

Source file name:	bestrelayteam.c, bestrelayteam.cpp, bestrelayteam.java, bestrelayteam.py
Input:	`Standard`
Output:	`Standard`

You are the coach of the national athletics team and need to select which sprinters should represent your country in the 4×100 m relay in the upcoming championships.

As the name of the event implies, such a sprint relay consist of 4 legs, 100 meters each. One would think that the best team would simply consist of the 4 fastest 100 m runners in the nation, but there is an important detail to take into account: flying start. In the 2nd, 3rd and 4th leg, the runner is already running when the baton is handed over. This means that some runners – those that have a slow acceleration phase – can perform relatively better in a relay if they are on the 2nd, 3rd or 4th leg.

You have a pool of runners to choose from. Given how fast each runner in the pool is, decide which four runners should represent your national team and which leg they should run. You are given two times for each runner – the time the runner would run the 1st leg, and the time the runner would run any of the other legs. A runner in a team can only run one leg.

Input

The first line of input contains an integer n, the number of runners to choose from ($4 \leq n \leq 500$). Then follow n lines describing the runners. The i'th of these lines contains the name of the i'th runner, the time a_i for the runner to run the 1st leg, and the time b_i for the runner to run any of the other legs ($8 \leq b_i \leq a_i < 20$). The names consist of between 2 and 20 (inclusive) uppercase letters '`A`'-'`Z`', and no two runners have the same name. The times are given in seconds with exactly two digits after the decimal point.

Output

First, output a line containing the time of the best team, accurate to an absolute or relative error of at most 10^{-9}. Then output four lines containing the names of the runners in that team. The first of these lines should contain the runner you have picked for the 1st leg, the second line the runner you have picked for the 2nd leg, and so on. Any solution that results in the fastest team is acceptable.

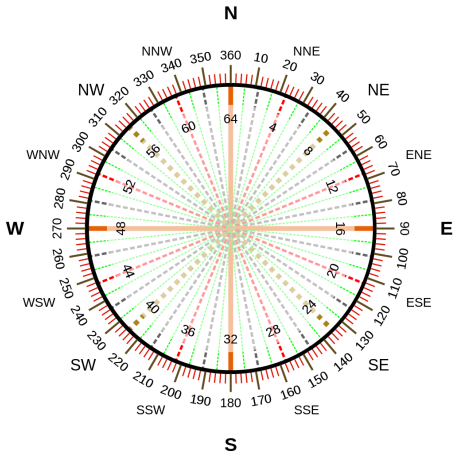
Example

Input	Output
6 ASHMEADE 9.90 8.85 BLAKE 9.69 8.72 BOLT 9.58 8.43 CARTER 9.78 8.93 FRATER 9.88 8.92 POWELL 9.72 8.61	35.54 CARTER BOLT POWELL BLAKE
9 AUSTRIN 15.60 14.92 DRANGE 15.14 14.19 DREGI 15.00 14.99 LAAKSONEN 16.39 14.97 LUNDSTROM 15.83 15.35 MARDELL 13.36 13.20 POLACEK 13.05 12.55 SANNEMO 15.23 14.74 SODERMAN 13.99 12.57	52.670000 MARDELL POLACEK SODERMAN DRANGE

Problem C. Compass Card Sales

Source file name:	compasscard.c, compasscard.cpp, compasscard.java, compasscard.py
Input:	`Standard`
Output:	`Standard`

Katla has recently stopped playing the collectible card game Compass. As you might remember, Compass is a game where each card has a red, a green and a blue angle, each one between 0 and 359, as well as an ID. Since she has stopped playing, Katla has decided to sell all her cards. However, she wants to keep her deck as unique as possible while selling off the cards. Can you help her figure out the order in which she should sell the cards?



To decide how unique a card is in the deck, she proceeds as follows. For each of the three colors she finds the closest other card in both directions, and then computes the angle between these two other cards. For instance if she has three cards with red angles 42, 90 and 110, then the uniqueness values of their red angles are 340, 68, and 312, respectively. If two cards A and B have the same angle, B is considered the closest to A in both directions so that the uniqueness value of A (and B) for that color is 0.

By summing the uniqueness values over the three colours, Katla finds how unique each card is. When selling a card, Katla sells the currently least unique card (smallest uniqueness value). If two cards have the same uniqueness value, she will sell the one with the higher ID first. After each card is sold, the uniqueness values of the remaining cards are updated before selling the next card.

Input

The first line of input contains an integer n, the number of cards ($1 \leq n \leq 10^5$). Then follows n lines. Each of these n lines contains 4 integers r, g, b, id ($0 \leq r, g, b < 360$, $0 \leq \text{id} < 2^{31}$), giving the red, green and blue angles as well as the ID of a card. No two cards have the same ID.

Output

Output n lines, containing the IDs of the cards in the order they are to be sold, from first (least unique) to last (most unique).

Example

Input	Output
3 42 1 1 1 90 1 1 2 110 1 1 3	2 3 1
4 0 0 0 0 120 120 120 120 240 240 240 240 0 120 240 2017	2017 240 120 0

Problem D. Distinctive Character

Source file name:	distinctive.c, distinctive.cpp, distinctive.java, distinctive.py
Input:	`Standard`
Output:	`Standard`

Tira would like to join a multiplayer game with n other players. Each player has a character with some features. There are a total of k features, and each character has some subset of them.

The similarity between two characters A and B is calculated as follows: for each feature f, if both A and B have feature f or if none of them have feature f, the similarity increases by one.

Tira does not have a character yet. She would like to create a new, very original character so that the maximum similarity between Tira's character and any other character is as low as possible.

Given the characters of the other players, your task is to create a character for Tira that fulfils the above requirement. If there are many possible characters, you can choose any of them.

Input

The first line of input contains two integers n and k, where $1 \leq n \leq 10^5$ is the number of players (excluding Tira) and $1 \leq k \leq 20$ is the number of features.

Then follow n lines describing the existing characters. Each of these n lines contains a string of k digits which are either 0 or 1. A 1 in position j means the character has the j'th feature, and a 0 means that it does not have the j'th feature.

Output

Output a single line describing the features of Tira's character in the same format as in the input. If there are multiple possible characters with the same smallest maximum similarity, any one of them will be accepted.

Example

Input	Output
3 5 01001 11100 10111	00010
1 4 0000	1111

Problem E. Emptying the Baltic

Source file name:	emptyingbaltic.c, emptyingbaltic.cpp, emptyingbaltic.java, emptyingbaltic.py
Input:	`Standard`
Output:	`Standard`

Gunnar dislikes forces of nature and always comes up with innovative plans to decrease their influence over him. Even though his previous plan of a giant dome over Stockholm to protect from too much sunlight (as well as rain and snow) has not yet been realized, he is now focusing on preempting the possible effects climate change might have on the Baltic Sea, by the elegant solution of simply removing the Baltic from the equation.

First, Gunnar wants to build a floodbank connecting Denmark and Norway to separate the Baltic from the Atlantic Ocean. The floodbank will also help protect Nordic countries from rising sea levels in the ocean. Next, Gunnar installs a device that can drain the Baltic from the seafloor. The device will drain as much water as needed to the Earth's core where it will disappear forever (because that is how physics works, at least as far as Gunnar is concerned). However, depending on the placement of the device, the entire Baltic might not be completely drained – some pockets of water may remain.

To simplify the problem, Gunnar is approximating the map of the Baltic using a 2-dimensional grid with 1 meter squares. For each square on the grid, he computes the average altitude. Squares with negative altitude are covered by water, squares with non-negative altitude are dry. Altitude is given in meters above the sea level, so the sea level has altitude of exactly 0. He disregards lakes and dry land below the sea level, as these would not change the estimate much anyway.

Water from a square on the grid can flow to any of its 8 neighbours, even if the two squares only share a corner. The map is surrounded by dry land, so water never flows outside of the map. Water respects gravity, so it can only flow closer to the Earth's core – either via the drainage device or to a neighbouring square with a lower water level.

Gunnar is more of an idea person than a programmer, so he has asked for your help to evaluate how much water would be drained for a given placement of the device.

Input

The first line contains two integers h and w, $1 \leq h, w \leq 500$, denoting the height and width of the map.

Then follow h lines, each containing w integers. The first line represents the northernmost row of Gunnar's map. Each integer represents the altitude of a square on the map grid. The altitude is given in meters and it is at least -10^6 and at most 10^6.

The last line contains two integers i and j, $1 \leq i \leq h, 1 \leq j \leq w$, indicating that the draining device is placed in the cell corresponding to the j'th column of the i'th row. You may assume that position (i, j) has negative altitude (i.e., the draining device is not placed on land).

Output

Output one line with one integer – the total volume of sea water drained, in cubic meters.

Example

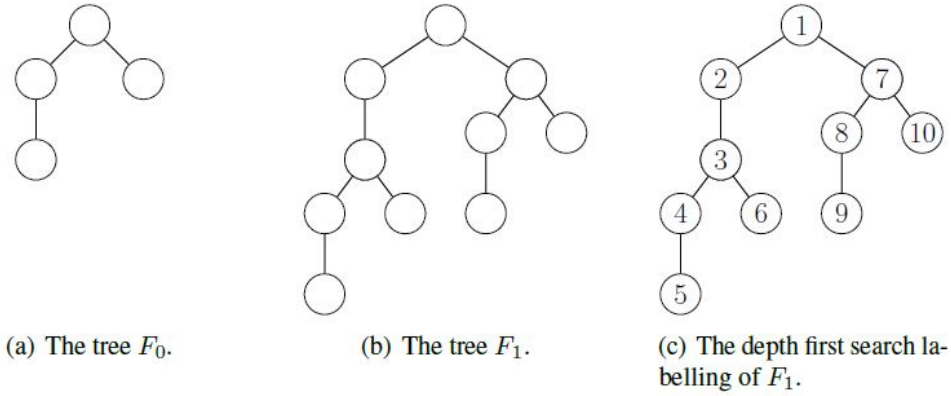
Input	Output
3 3 -5 2 -5 -1 -2 -1 5 4 -5 2 2	10
2 3 -2 -3 -4 -3 -2 -3 2 1	16

Problem F. Fractal Tree

Source file name:	fractaltree.c, fractaltree.cpp, fractaltree.java, fractaltree.py
Input:	`Standard`
Output:	`Standard`

A fractal tree F_i is defined in the following way. First, a rooted tree F_0 is given, which contains at least 2 vertices. F_i is then defined recursively in the following manner. Consider the set of vertices S which are leaves in F_{i-1}. For each vertex v in S, we replace it with a copy of F_0, such that v corresponds to the root of F_0.

Now, consider the tree F_k, for a given k. In this tree, we perform a depth-first search, where we visit all vertices of the tree recursively. At a certain vertex, we first recurse into the subtree of the leftmost child of the vertex, then the second leftmost child, and so on, until we have visited all the vertices in the subtree of the vertex. Assign integer labels to the vertices in the order they were visited, starting at 1. See Figure 1 for an example.



(a) The tree F_0. (b) The tree F_1. (c) The depth first search labelling of F_1.

Figure F.1: Illustration of Sample Input 1.

Given a set of queries consisting of pairs of vertices, your task is to find the distance between the two vertices. The distance is defined as the number of edges on the (unique) simple path between the two vertices.

Input

First, the tree F_0 is given. The first line of input contains the number of vertices $2 \leq n \leq 100\,000$ in F_0. The vertices are numbered 0 to $n-1$, with 0 being the root vertex. Then follows a line containing $n-1$ integers $p_1, \ldots, p_{n-1}$. For each $1 \leq i \leq n-1$, the parent of node i in F_0 is p_i. It holds that $p_i < i$. Within the tree, the left-to-right ordering of the vertices correspond to their numbering, in ascending order (i.e. the lowest-numbered child is the leftmost child).

The third line of input contains an integer $0 \leq k < 2^{30}$. Then follows a line containing an integer q, $1 \leq q \leq 100\,000$, the number of queries. Finally, there are q lines containing the queries. Each query is given by two distinct integers a and b, the labels of two vertices of F_k. You may assume that a and b are valid labels (i.e., they are between 1 and the number of vertices of F_k), and that they are at most 2^{30}.

Output

For each query (a, b), in the same order as given in the input, output the distance in F_k between the vertices labelled a and b.

Example

Input	Output
4 0 1 0 1 10 1 2 1 4 1 6 1 8 1 10 5 10 6 8 9 3 7 10 8 9	1 3 3 2 2 6 5 5 1 1

Problem G. Galactic Collegiate Programming Contest

Source file name:	gcpc.c, gcpc.cpp, gcpc.java, gcpc.py
Input:	`Standard`
Output:	`Standard`

One hundred years from now, in 2117, the International Collegiate Programming Contest (of which the NCPC is a part) has expanded significantly and it is now the Galactic Collegiate Programming Contest (GCPC).

This year there are n teams in the contest. The teams are numbered $1, 2, \ldots, n$, and your favorite team has number 1.

Like today, the score of a team is a pair of integers (a, b) where a is the number of solved problems and b is the total penalty of that team. When a team solves a problem there is some associated penalty (not necessarily calculated in the same way as in the NCPC – the precise details are not important in this problem). The total penalty of a team is the sum of the penalties for the solved problems of the team.

Consider two teams t_1 and t_2 whose scores are (a_1, b_1) and (a_2, b_2). The score of team t_1 is better than that of t_2 if either $a_1 > a_2$, or if $a_1 = a_2$ and $b_1 < b_2$. The rank of a team is $k + 1$ where k is the number of teams whose score is better.

You would like to follow the performance of your favorite team. Unfortunately, the organizers of GCPC do not provide a scoreboard. Instead, they send a message immediately whenever a team solves a problem.

Input

The first line of input contains two integers n and m, where $1 \leq n \leq 10^5$ is the number of teams, and $1 \leq m \leq 10^5$ is the number of events.

Then follow m lines that describe the events. Each line contains two integers t and p ($1 \leq t \leq n$ and $1 \leq p \leq 1000$), meaning that team t has solved a problem with penalty p. The events are ordered by the time when they happen.

Output

Output m lines. On the i'th line, output the rank of your favorite team after the first i events have happened.

Input	Output
3 4 2 7 3 5 1 6 1 9	2 3 2 1

Problem H. Hubtown

Source file name:	hubtown.c, hubtown.cpp, hubtown.java, hubtown.py
Input:	`Standard`
Output:	`Standard`

Hubtown is a large Nordic city which is home to n citizens. Every morning, each of its citizens wants to travel to the central hub from which the city gets its name, by using one of the m commuter trains which pass through the city. Each train line is a ray (i.e., a line segment which extends infinitely long in one direction), ending at the central hub, which is located at coordinates $(0, 0)$. However, the train lines have limited capacity (which may vary between train lines), so some train lines may become full, leading to citizens taking their cars instead of commuting. The city council wishes to minimize the number of people who go by car. In order to do this, they will issue instructions stating which citizens are allowed to take which train.

A citizen will always take the train line which is of least angular distance from its house. However, if a citizen is exactly in the middle between two train lines, they are willing to take either of them, and city council can decide which of the two train lines the citizen should use. See Figure H.1 for an example.

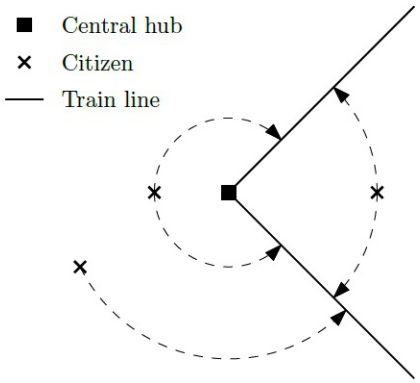


Figure H.1: Illustration of Sample Input 1. The dashed arrows indicate which train lines the citizens are closest to (note that we are measuring angular distances, not Euclidean distance).

Your task is to help the council, by finding a maximum size subset of citizens who can go by train in the morning to the central hub, ensuring that each of the citizens take one of the lines they are closest to, while not exceeding the capacity of any train line. For this subset, you should also print what train they are to take.

Input

The first line of input contains two integers n and m, where $0 \leq n \leq 200\,000$ is the number of citizens, and $1 \leq m \leq 200\,000$ is the number of train lines.

The next n lines each contain two integers x and y, the Cartesian coordinates of a citizen's home. No citizen lives at the central hub of the city.

Then follow m lines, each containing three integers x, y, and c describing a train line, where (x, y) are the coordinates of a single point (distinct from the central hub of the city) which the train line passes through and $0 \leq c \leq n$ is the capacity of the train line. The train line is the ray starting at $(0, 0)$ and passing through (x, y).

All coordinates x and y (both citizens' homes and the points defining the train lines) are bounded by 1000 in absolute value. No two train lines overlap, but multiple citizens may live at the same coordinates.

Output

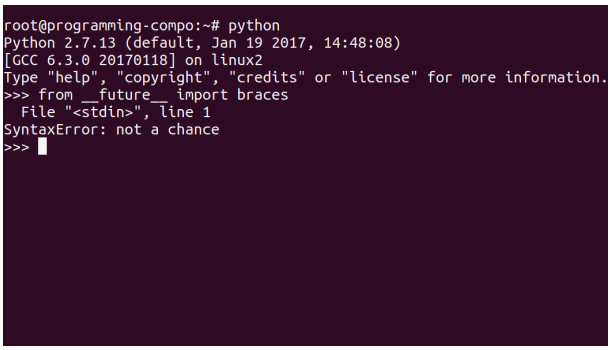
First, output a single integer s – the maximum number of citizens who can go by train. Then, output s lines, one for each citizen that goes by train. On each line, output the index of the citizen followed by the index of the train line the citizen takes. The citizens are numbered from 0 to $n-1$ in the order they are given in the input. The trains are numbered from 0 to $m-1$ in the order they are given in the input. The output lines may be given in any order.

Input	Output
3 2 2 0 -1 0 -2 -1 1 -1 1 1 1 2	3 0 1 1 1 2 0
6 3 1 1 1 1 1 1 -1 1 -1 1 0 1 -1 0 2 0 1 2 1 0 2	6 0 2 1 2 2 1 5 1 3 0 4 0

Problem I. Import Spaghetti

Source file name:	importspaghetti.c, importspaghetti.cpp, importspaghetti.java, importspaghetti.py
Input:	`Standard`
Output:	`Standard`

You just graduated from programming school and nailed a Python programming job. The first day at work you realize that you have inherited a mess. The *spaghetti* design pattern was chosen by the previous maintainer, who recently fled the country. You try to make sense of the code, but immediately discover that different files depend cyclically on each other. Testing the code, in fact running the code, has not yet been attempted.



As you sit down and think, you decide that the first thing to do is to eliminate the cycles in the dependency graph. So you start by finding a shortest dependency cycle.

Input

The first line of input contains a number n, $1 \leq n \leq 500$, the number of files. Then follows one line with n distinct names of files. Each name is a string with at least 1 and at most 8 lower case letters '`a`' to '`z`'. Then follow n sections, one section per file name, in the order they were given on the second line. Each section starts with one line containing the name of the file and an integer k, followed by k lines, each starting with "`import`".

Each "`import`" line is a comma-space separated line of dependencies. No file imports the same file more than once, and every file imported is listed in the second line of the input. Comma-space separated means that every line will start with "`import`", then have a list of file names separated by "`, `" (see sample inputs for examples). Each import statement is followed by at least one file name.

Output

If the code base has no cyclic dependencies, output "`SHIP IT`". Otherwise, output a line containing the names of files in a shortest cycle, in the order of the cycle (i.e., the ith file listed must import the $(i+1)$st file listed, and the last file listed must import the first). If there are many shortest cycles, any one will be accepted.

Input	Output
4 a b c d a 1 import d, b, c b 2 import d import c c 1 import c d 0	c
5 classa classb myfilec execd libe classa 2 import classb import myfilec, libe classb 1 import execd myfilec 1 import libe execd 1 import libe libe 0	SHIP IT
5 classa classb myfilec execd libe classa 2 import classb import myfilec, libe classb 1 import execd myfilec 1 import libe execd 1 import libe, classa libe 0	classa classb execd

Problem J. Judging Moose

Source file name:	judgingmoose.c, judgingmoose.cpp, judgingmoose.java, judgingmoose.py
Input:	`Standard`
Output:	`Standard`

When determining the age of a bull moose, the number of tines (sharp points), extending from the main antlers, can be used. An older bull moose tends to have more tines than a younger moose. However, just counting the number of tines can be misleading, as a moose can break off the tines, for example when fighting with other moose. Therefore, a point system is used when describing the antlers of a bull moose.

The point system works like this: If the number of tines on the left side and the right side match, the moose is said to have the even sum of the number of points. So, "an even 6-point moose", would have three tines on each side. If the moose has a different number of tines on the left and right side, the moose is said to have twice the highest number of tines, but it is odd. So "an odd 10-point moose" would have 5 tines on one side, and 4 or less tines on the other side.

Can you figure out how many points a moose has, given the number of tines on the left and right side?

Input

The input contains a single line with two integers ℓ and r, where $0 \leq \ell \leq 20$ is the number of tines on the *left*, and $0 \leq r \leq 20$ is the number of tines on the *right*.

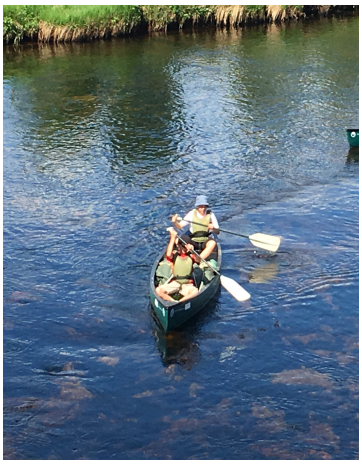
Output

Output a single line describing the moose. For even pointed moose, output "`Even` x" where x is the points of the moose. For odd pointed moose, output "`Odd` x" where x is the points of the moose. If the moose has no tines, output "`Not a moose`"

Input	Output
2 3	Odd 6
3 3	Even 6
0 0	Not a moose

Problem K. Kayaking Trip

Source file name:	kayaking.c, kayaking.cpp, kayaking.java, kayaking.py
Input:	`Standard`
Output:	`Standard`



You are leading a kayaking trip with a mixed group of participants in the Stockholm archipelago, but as you are about to begin your final stretch back to the mainland you notice a storm on the horizon. You had better paddle as fast as you can to make sure you do not get trapped on one of the islands. Of course, you cannot leave anyone behind, so your speed will be determined by the slowest kayak. Time to start thinking; How should you distribute the participants among the kayaks to maximize your chance of reaching the mainland safely?

The kayaks are of different types and have different amounts of packing, so some are more easily paddled than others. This is captured by a speed factor c that you have already figured out for each kayak. The final speed v of a kayak, however, is also determined by the strengths s_1 and s_2 of the two people in the kayak, by the relation $v = c(s_1 + s_2)$. In your group you have some beginners with a kayaking strength of s_b, a number of normal participants with strength s_n and some quite experienced strong kayakers with strength s_e.

Input

The first line of input contains three non-negative integers b, n, and e, denoting the number of beginners, normal participants, and experienced kayakers, respectively. The total number of participants, $b + n + e$, will be even, at least 2, and no more than 100 000. This is followed by a line with three integers s_b, s_n, and s_e, giving the strengths of the corresponding participants ($1 \leq s_b < s_n < s_e \leq 1\,000$). The third and final line contains $m = \frac{b+n+e}{2}$ integers $c_1, \ldots, c_m$ ($1 \leq c_i \leq 100\,000$ for each i), each giving the speed factor of one kayak.

Output

Output a single integer, the maximum speed that the slowest kayak can get by distributing the participants two in each kayak.

Input	Output
3 1 0 40 60 90 18 20	1600
7 0 7 5 10 500 1 1 1 1 1 1 1	505