Problem A
Three Squares

Input File: A.in
Output File: standard output
Time Limit: 1 seconds (C/C++)
Memory Limit: 256 megabytes

There are ***N*** distinct points located at integer coordinates in the plane. We want to place three identical axis-parallel squares on the plane, such that each of the ***N*** points is located inside (or on the borders of) at least one of the squares. What is the minimum possible side length of the squares?

Input
The first line contains the integer number ***N*** (**$1 \le N \le 100000$**). The next ***N*** lines contain two space-separated integers each, the **x** and **y** coordinates of one of the points (**$0 \le x, y \le 10^9$**).

Output
Print the minimum possible side length of the 3 squares, such that all the ***N*** points are covered by them.

Sample input	Sample output

5
0 0
10 0
0 1
2 2
9 3
	2

Explanation
You can place the 3 squares with their bottom-left corners at the following coordinates: (0,0), (8,0), (7,1).

Problem B
Favorite music

Input File: B.in
Output File: standard output
Time Limit: 1 second (C/C++)
Memory Limit: 256 megabytes

Al likes music a lot. He has ***n*** favorite music fragments he particularly enjoys listening to. It is well known that music can be expressed using notes. For simplicity, we'll ignore pauses, different note durations and octaves. We'll assume that a music fragment is just a non-empty sequence of notes (without chords, so no two notes are played at the same time) in the common letter notation:

C, C#, D, D#, E, F, F#, G, G#, A, A#, B.

What is better than listening to some favorite music? Only listening to more favorite music! Unfortunately, Al doesn't have a lot of time for that. Therefore, he'd like to know how much time does he need to hear a couple of his favorite fragments at a time. These fragments may overlap, but the notes in both of them must go in the same order and without any insertions in between.

Input
The first line contains an integer ***n*** ($1 \le n \le 3 \times 10^5$) – the number of music fragments. Each of the next ***n*** lines contains a sequence of notes without spaces describing one fragment. It's guaranteed that all fragments are different. The total length of the strings representing all the fragments is not greater than 3×10^5. The next line contains an integer ***q*** ($1 \le q \le 10^5$) – the number of questions Al has in mind. Each of the next ***q*** lines contains two different fragment indices. Fragments are numbered from 1 to ***n*** as they appear in the input.

Output
For each question, print on a separate line the minimum time needed to hear both given fragments. Assume that each note lasts for one time interval.

Sample input	Sample output

`3`
`ABCAB`
`AC`
`ACAB`
`3`
`1 2`
`1 3`
`2 3`
	`7`
	`7`
	`4`

`5`
`EBEBGF#DEEBEBF#GABBEBECBGAAADAEADA`
`EECAGGECDECBAACDEECAAAFABCAGAAAA`
`EECAGGECDECBAACDECEFGCGFEFDCDDDD`
`G#A#CDD#FD#DCA#G#A#CDD#FDD#FGG#A#G#GFD#DD#FGG#A#`
`GG#A#CDD#DCFD#DGFD#DCA#CDD#FGFD#A#CA#G#GFD#DCC`
`3`
`2 3`
`4 5`
`5 1`
	`64`
	`63`
	`66`

Southeastern European Regional Programming Contest
Bucharest, Romania – Vinnytsya, Ukraine
October 22, 2016

Problem C
Castle

Input File: C.in
Output File: standard output
Time Limit: 0.5 seconds (C/C++)
Memory Limit: 256 megabytes

K. has stumbled upon a weird game while playing on his computer. The game consists of an initial string ***S*** of length ***N*** (1 ≤ ***N*** ≤ 1000) and an empty set **T**. The following events might occur during the game:

- a character is added at the end of ***S***, thus increasing its length by 1
- the string ***S*** is added to the set ***T***
- the game master inquires: “How many strings in ***T*** are suffixes of ***S***?”. A suffix of ***S*** is a substring which can start at any position in ***S***, but must finish on the last position of ***S***.

Because K. wants to go visit a famous castle near his hometown, you must help him deal with the game as quickly as possible.

Input

The first line of the input contains two integers: ***N***, the length of the initial string ***S*** and ***E***, the number of events (***E*** ≤ 1200000).

The second line describes the string ***S***; the string consists only of lowercase Roman alphabet (a-z).

The following ***E*** lines describe the events. Each of these lines contains an integer ***p***, describing the event type.

If ***p*** is 1, then it is followed by a character (a-z), which will be added at the end of ***S***.

If ***p*** is 2, then the string ***S*** is added in ***T***.

If ***p*** is 3, then you must respond to the query “How many strings in T are suffixes of the current S?”

Output

The output should consist of a line containing an integer for each type 3 event in the input, which represents the answer to the query.

Note: ***T*** is a set, so it doesn't contain duplicates.

Note: Large input: for C **scanf** and **printf** recommended; for C++, add “`cin.tie(NULL); ios::sync_with_stdio(false);`” before reading; for Java use **BufferedReader** and **BufferedWriter**.

Sample input	Sample output	Explanation
`1 11`	`1`	
`a`	`2`	**Initially S is “a”.**
`2`		**After the first event T becomes {“a”}.**
`1 b`		**After the second and third event S becomes**
`1 a`		**“aba”.**
`2`		**After the fourth event T becomes {“a”, “aba”}.**
`2`		**After the fifth event T becomes {“a”, “aba”}.**
`1 c`		**After the sixth and seventh event S becomes**
`1 a`		**“abaca”.**
`3`		**The result of the query is 1 (“a”).**
`1 b`		**After the ninth and tenth event S becomes**
`1 a`		**“abacaba”.**
`3`		**The result of the query is 2 (“a” and “aba”).**

Problem D
Reading Digits

Input File: D.in
Output File: standard output
Time Limit: 0.1 seconds (C/C++)
Memory Limit: 256 megabytes

Bob has a very special way of encoding strings formed with digits. For instance, he encodes “1211” as: “one of one, one of two, two of one”, or, more precisely: “111221”. Bob’s encoding of the latter string is: “312211”. We call this a “*two-times re-encoding of 1211”.* Bob likes repeating this process several times.

You are given a string of digits which represents the “***k****th re-encoding of a string* ***s***”. The string ***s*** contains only non-zero digits (i.e. [1-9]). Also, it is not possible to have a sequence of more than 9 repeating digits in ***s***. You must find the digit which lays on the ***pos*** position (starting from 0) of the string ***s***.

The input consists of two lines. The first line contains the values ***k*** and ***pos***. The second line contains the ***k***th re-encoding of ***s***. We have $1 < k < 40$ and $0 \le pos \le 100000$.
The output is the digit from position pos of ***s***.

Sample input	Sample output

`2 0`
`312211`
	`1`

`2 1`
`312211`
	`2`

`1 3`
`312211`
	`2`

`3 0`
`1321123113`
	`1`

`3 1`
`1321123113`
	`2`

`3 2`
`1321123113`
	`3`

Problem E
Exam

Input File: E.in
Output File: standard output
Time Limit: 1.5 seconds (C/C++)
Memory Limit: 256 megabytes

Some universities use an evaluation system according to which students can get from 0 to 100 points, out of which from 0 to 75 - during the semester, and from 0 to 25 - at the final exam. The final grade is determined based on the sum of the semester and the exam points as follows:

Sum of points	European grade
90—100	A
82—89	B
75—81	C
68—74	D
60—67	E
35—59	FX

If a student gets strictly less than 35 points during the semester, he or she is not allowed to pass the exam; we'll assume in this problem that such students' names have been previously crossed out from the list.

If one reads the column of European grades in the exam list from top to bottom, he or she can find various "words". For example, if there are consecutive sums of points such as 92, 75 and 66, they are marked as A, C and E respectively, and form the "word" ACE. In case of FX, both letters (first F, then X) appear in the "word".

It is impossible to know the exam results beforehand. But the lecturer knows both the approximate level of knowledge of each student and the difficulty of the exam. So, for all students, the lecturer can estimate the probabilities (in percentages) of each possible number of points the student can earn at the exam: i.e. the probability that the student will get 0 points, 1 point, ..., 25 points – for a total of 26 nonnegative integers, whose sum equals 100. Points, obtained by every student during the semester, are also known (as concrete numbers from 35 to 75, without any probabilities).

The lecturer is a great esthete and dislikes situations when the "word", produced by European marks, contains "unpleasant" substrings, according to his taste.

Your task is to write a program to find the probability that no "unpleasant" substring will occur.

The 1st line of the input file contains the number of students ***N*** ($3 \le N \le 4096$). Each of the following ***N*** lines contains 27 space-separated integers — the number of semester points (from 35 to 75), and 26 probabilities corresponding to 0, 1, 2, ..., 25 exam points (each probability is non-negative, their total sum is 100). The next line contains the number ***K*** (1 ≤ ***K*** ≤ 1024) of "unpleasant" words according to lecturer's opinion. Each of the following ***K*** lines contains an "unpleasant" word. It is guaranteed that each of these ***K*** lines contains only uppercases and ends with the end-of-line symbol. The length of each "unpleasant" word is in the range 2...1024 and the total sum of lengths of all "unpleasant" words does not exceed 32768.

Your program should write to standard output exactly one floating-point number on a single line – the probability (in percentages) that the lecturer will be satisfied. Floating-point format may be any of the standard ones (using decimal point, not decimal comma). The answer will be accepted iff its relative or absolute error does not exceed 1e–6.

Input	**Output**
3 72 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 2 3 5 7 9 14 16 21 12 5 1 55 0 0 0 1 2 3 4 5 6 7 8 8 9 8 8 7 6 5 4 3 2 2 1 1 0 0 55 0 0 0 1 2 3 4 5 6 7 8 8 9 8 8 7 6 5 4 3 2 2 1 1 0 0 2 DE WAW	79.5

The letter W may not occur in grades, so we can skip the word WAW and look for DE only. The 1st student's sum will be at least 72+10=82 points, so his mark can't be D. Thus, the "unpleasant" word DE will occur if and only if the 2nd student gets from 13 to 19 points (the probability is 8%+8%+7%+6%+5%+4%+3%=41%) and the 3rd – from 5 to 12 (the probability 3%+4%+5%+6%+7%+8%+8%+9%=50%). So, the word DE will appear with probability 0.41*0.5=0.205, and will not appear with probability 1–0.205=0.795 (i.e. 79.5%).

Problem F
Letters

Input File: F.in
Output File: standard output
Time Limit: 0.2 seconds (C/C++)
Memory Limit: 256 megabytes

We build the infinite sequence of uppercase letters (A-Z) which starts with single-letter strings (A, B, …, Z), continues with two-letter strings (AA, AB, …, AZ, BA, BB, …, BZ, …, ZZ), then three-letter strings and so on. The same-length strings are ordered lexicographically. We are interested in finding which letter sits at a given index in the sequence.

Input
The letter index, ranging between 0 and $2*10^9$.

Output
The letter at the specified index.

Sample input	Sample output
0	A
25	Z
50	A
100	B
250	E
500	J
1000	S

Southeastern European Regional Programming Contest
Bucharest, Romania – Vinnytsya, Ukraine
October 22, 2016

Problem G
Pokemons

Input File: G.in
Output File: standard output
Time Limit: 0.3 seconds (C/C++)
Memory Limit: 1 megabytes

Jim is fond of Pokemons, and plays all kinds of games involving them. Now, he plays a trading game. He knows the price of a Pokemon on each day over the next ***n*** days (Pokemons have the same price, no matter the type). He starts with a given amount of money, and picks one day to buy as many Pokemons (including fractions of them) as he can afford - he must spend ***all*** his money, and then sell them ***all*** on some ***subsequent*** day. Obviously, he wants to maximize his profit (or minimize the losses), and must decide very fast. Can you help him?

The input file starts with the amount of money Jim will use. The next line contains the number ***n*** **(**$1 < n \leq 10^6$**)** of days. Starting from a different line follows the ***n*** space-separated prices of Pokemons for each day.

The output file contains the maximum profit (may be negative also), a real number with 2 decimal digits (0.005 is 0.01 while 0.0049 is 0.00; -0.005 is -0.01 and -0.0049 is -0.00).

The sample describes an instance of the game. The first line of the input contains the amount of money, the next one the number of days, the following line contains the prices per day of the Pokemons.

Sample input	Sample output

`100.7`
`5`
`1 2.88 3.05 4.33 5.5`
	`453.15`

Problem H
Pub crawl

Input File: H.in
Output File: standard output
Time Limit: 0.3 seconds (C/C++)
Memory Limit: 256 megabytes

Al just arrived in Dublin. He is going to spend his cash on the famous Dublin activity - the pub crawl. The goal is to drink a pint of Guinness in as many different pubs in the city as possible, not visiting any pub twice. There are ***n*** pubs in Dublin. Al gets drunk very fast, so he sees pubs as points on a plane, and his path from a pub to the next one as a straight line connecting these points, though the actual path may involve going along different streets, around buildings or even walking in circles trying to find the next pub. Al doesn't care about those details. The only thing he cares about is that every turn he makes is a left turn, he enjoys going left. This means for every three consecutive pubs in his route the third one must lie in the left half plane with respect to the directed line from the first pub to the second. It is known that builders in Dublin also enjoy pub crawl, so they never managed to build at least three pubs on a straight line. Help Al to find out how many pubs he can visit and plan the route for him.

Input

The first line contains an integer ***n*** ($1 \le n \le 5000$) – the number of pubs.
Each of the next ***n*** lines contains two integers ***x*** and ***y*** ($-10^9 \le x, y \le 10^9$) – pub coordinates. Different pubs are located in different points.

Output

The first line should contain an integer ***m*** – the maximum number of pubs Al can visit.
The next line should contain ***m*** indices of pubs in the order Al should visit them. Pubs are numbered from 1 to ***n*** in the same order as they appear in the input.

Sample input	Sample output

5
0 4
3 0
7 11
9 1
13 8
	5
	5 1 4 3 2

Problem I
Cubes

Input File: I.in
Output File: standard output
Time Limit: 0.5 seconds (C/C++)
Memory Limit: 256 megabytes

Write a program that takes a natural number N and decomposes it as a sum of the minimum number of exact natural cubes. The program should find m_1, m_2, …, m_k, such that each m_i is a natural number, $m_1^3 + m_2^3 + \ldots + m_k^3 = N$, and k is minimal.

Input

The only line of the input file contains the number N ($1 \leq N \leq 44{,}777{,}444$).

Output

Your program should write exactly two lines. The first line contains the number k - the minimum number of natural cubes. The second line contains k space-separated natural numbers - that raised to the power of 3 sum to N.

Sample input	Sample output

`42`	`7`
	`2 2 2 2 2 1 1`

`43`	`3`
	`3 2 2`

Problem J
Marathon

Input File: J.in
Output File: standard output
Time Limit: 0.2 seconds (C/C++)
Memory Limit: 256 megabytes

You've began running long distances. Simplistically, the road on which athletes race is a straight, infinite line. At some point in time all participants split into ***n*** groups of k_i people each.
Each group at this point in time is at coordinate x_i. We know that if a group consists of ***D*** people, the group's speed is **100/*D***. All groups move in the direction of coordinate's growth. If one team catches another, they merge and their speed changes accordingly (more than two groups can merge simultaneously).
Since the road is infinite, from some point in time no more merges are possible.
You, as a beginner, are interested in the number of groups remaining and the number of people in each of them.

Input

The first line contains an integer ***n*** ($1 \le n \le 10^5$). Each of the following ***n*** lines contains the number k_i of people in the corresponding group and its coordinate x_i (x_i - real numbers with no more than three decimal digits and their absolute values do not exceed 10^4, $1 \le k_i \le 100$, all the coordinates are different).

Output

The first line of output contains the number of groups ***m***. The second line contains ***m*** integers - the number of people in each of these groups, in any order.

Sample input	Sample output

4
1 0
2 9000
4 1
3 10000
	2
	5 5

Problem K
Cutting

Input File: K.in
Output File: standard output
Time Limit: 0.2 seconds (C/C++)
Memory Limit: 256 megabytes

You have two strings and you need to check whether you can cut the second one into three pieces so that the first string can be concatenated from these three pieces.

Input

The input file contains two non-empty strings, each on a separate line. Each string consists of at most 5000 lowercase roman letters. Strings have the same length and each letter has the same number of occurrences in both strings.

Output

Print YES if it is possible to cut the string as described, and NO in the opposite case. If you print YES, the following three lines should contain the parts of the second string in the correct concatenation order. These parts cannot be empty. If you have several options to cut, print any of them.

Sample input	Sample output

`Beast`
`betas`
	`YES`
	`be`
	`as`
	`t`

`Royalitem` `Romeitaly`	`NO`