Student programming contest

bubblecup

Microsoft Development Center Serbia

Problem set & Analysis from the Finals and Qualification rounds

Belgrade, 2017

Scientific committee:
Aleksandar Damjanović
David Milićević
Borna Vukorepa
Nikola Nedeljković
Branko Fulurija
Slavko Ivanović
Ivan Dejković
Marko Rakita
Janko Šušteršič
Miloš Kurškonja
Predrag Ilkić
Nikola Smiljković
Miloš Šuković
Ibragim Ismailov

Qualification analysts:
Ivan Stošić
Vselovod Stepanov
Yuri Shilayaev
Kristijan Verović
Vladimir Milenković
Vladzslav Hlembotskyi
Nikola Herceg
Domagoj Bradac
Michal Seweryn
Balsa Knezevic
Stjepan Požgaj
Aleksa Milisavljević
Ilija Rackov
Krzysztof Maziarz
Alexander Crustev
Nikolay Zhidkov
Juliusz Straszyński
Ivan Paljak
Mihajlo Kocić
Mikhail Mayorov

Cover:
Sava Čajetinac

Typesetting:
Monika Čolić

Volume editor:
Dragan Tomić

Contents

Preface

Greetings, fellow finalists!

It is my great pleasure to wish you a warm welcome and thank you for taking part in this tenth edition of the Bubble Cup in Belgrade, Serbia.

Ten years ago, we set off to create a competition which would boost the success of domestic teams at regional ACM contest in 2008. But, as the years went by, we succeed in transforming the Bubble Cup into one of the most distinguished international competitions. We are very proud of what we achieved with the contest and with the community of folks associated with the contest.

This wouldn't be possible without the students who recognize the Bubble Cup for what it is: a challenging competition worthy of their energy, knowledge and devotion to coding. In the past ten years, more than 3000 students worldwide have taken part in the Bubble Cup during the 1st and 2nd round and we have hosted more than 500 finalists here in Belgrade, Serbia.

This year, we broke our previous record of total number of teams competing by amazing 30%. We are delighted with the number of top teams and amazing coders from Serbia, Croatia, Bulgaria, Poland, Ukraine, Belorussia and Russia, that decided to compete this year and make this the hardest Bubble Cup competition yet.

A special place in the Cup's 10-year long history is reserved for people who have given and are still giving their all to the Bubble Cup. I would like to give special thanks to all previous Bubble Cup directors and all the folks who were the members of the Bubble Cup crew. Also, we are extremely grateful to the Bubble Cup judges who, with their knowledge and enthusiasm, helped elevate the Bubble Cup into what it is today. Of course, we won't stop here. We are going to give our best to make the Bubble Cup more challenging and to make it a more prestigious coding competition with each new year to come. Microsoft Development Center Serbia started its operation in 2005 and is one of the most distinguished operations in Europe. Today it is the place that 200 engineers call home. A considerable number of them were like you today – Bubble Cup finalists. This is a competition that may presents a lot of opportunities to young, ambitious people like yourselves.

The world in which we are living is changing and evolving rapidly. In this fast-changing world coding is still the most in-demand skill across industries. Without a doubt, 2016 was an amazing year for Machine Learning (ML) and Artificial Intelligence (AI) where we made some tremendous progress. Every industry branch is transformed by software today. Therefore, you are the folks that are at the forefront of the transformation of our society. You are leading the 4th industrial revolution.

Some of you are the rising stars who will transform the world and we are happy to accompany you on this tiny part of your journey. We hope you had fun, made new friends and experienced the Bubble Cup 10 as a special adventure. Let's continue to shape the future together. Take the Bubble Cup as an opportunity to advance your technical knowledge and to build relationships that will last you a lifetime.

Sincerely,
Dragan Tomic

MDCS PARTNER Engineer manager/Director

About Bubble Cup

Bubble Cup is a coding contest started by Microsoft Development Center Serbia in 2008 with a purpose of creating a local competition similar to the ACM Collegiate Contest, but soon that idea was overgrown and the vision was expanded to attract talented programmers from the entire region and promote the values of communication, companionship and teamwork.

The format of the competition has remained the same this year. All competitors battled for the place in finals during two qualifications rounds. Top high school and university teams were invited to the finals in Belgrade, where they competed in the traditional five-hour long contest.
All Bubble Cup finalists had a chance to hear about PSI:ML machine learning seminar where they can meet people interested in machine learning, learn and work on projects alongside industry expert from the field.

Microsoft Development Center Serbia (MDCS) was created with a mission to take an active part in the conception of novel Microsoft technologies by hiring unique local talent from Serbia and the region. Our teams contribute components to some of the Microsoft's premier and most innovative products such as SQL Server, Office & Bing. The whole effort started in 2005, and during the last 12 years a vast number of products came out as a result of a great team work and effort.

Our development center is becoming widely recognized across Microsoft as a center of excellence for the following domains: computational algebra engines, pattern recognition, object classification, computational geometry and core database systems. The common theme uniting all the efforts within the development center is applied mathematics. MDCS teams maintain collaboration with engineers from various Microsoft development centers around the world (Redmond, Israel, India, Ireland, Japan and China), and Microsoft researchers from Redmond, Cambridge and Asia.

Microsoft | Development Center Serbia

Bubble Cup finals

The Bubble Cup X Finals were held on September 2, 2017, at the Belgrade Youth Center (Dom Omladine Beograd) in Belgrade. Marko Panic who was the coordinator at the very first Bubble Cup was now the moderator of the Opening Ceremony and he welcomed both the finalists and guests and called on the speakers to greet them too.

Dragan Tomic presented how Bubble Cup started 10 years ago, and where we are now. He talked about MDCS partnership with the industry, academia and official institutions and how only together we could contribute to the society. A few short speeches were given by: Ms. Tanja Matic, the State Secretary at the Ministry of Trade, Tourism and Telecommunications, Ms. Katarina Aleksic, the Advisor to the Minister of Education, Science and Technological Development. Mr. Nebojsa Vasiljevic, the director of the Petlja Foundation and the professor Filip Maric held the presentation on how the portal Petlja.org works and how kids and professors can use the interactive materials for learning coding in elementary schools.

Competition started at 11.30am and lasted till 4.30pm. In the evening, at Dom Omladine Beograd, the award ceremony was held and was later followed by the lounge party organized to honor all the participants.

This is the 10th anniversary edition of the Bubble Cup and we are extremely happy to see an increase in the number of teams that surpassed our expectations. We were so pleased with the number of top teams and red coders that we decided to outdo ourselves this year and make it the toughest Bubble Cup competition to date. Because of that we decided to make an exception and invite 22 teams to the finals, breaking our traditional 20 teams total number. On the other hand, we didn't change the length of the competition which remained in the classical ACM ICPC five-hour format. The university and high school students competed in the same category. Bubble Cup prizes were given to not only to the top 3 overall teams, but also to the teams which were the top 3 in their respective categories.

Since the scoreboard froze there were a lot of changes in the standings and one submission could make the difference between winning one of the first three main prizes or going home empty handed. After starting slowly the **Los Estribos** team (Mateusz Radecki, Maciej Holubowicz, Jan Tabaszewski) ended up victorious and the only team to solve the Bob And Stages problem.The **Jagiellonian Armadillos** (Vladyslav Hlembotskyi, Michal Seweryn, Krzysztof Maziarz) were leading for a long time but in the end they finished in the second place. The biggest change in the freeze time, however, was made by the **Cheeks** (Nikolay Zhiidkov, Petr Smirnov, Vsevolod Stepanov) team that went from 4 solved problems to 7 and grabbed 3rd overall place as the only team with 7 points. In the end, penalty points didn't matter for the first three places as all three teams had unique number of solved problems.

Bubble Cup Finals Results

Results

#	User	Score	Penalty
1	Los Estribos	9	1560
2	Jagiellonian Armadillos	8	981
3	Cheeks	7	1426
4	jos jedan tim	6	423
5	Sek da pomislq	6	772
6	Onda to može... i još bolje!	6	843
7	Almost_the_algotesters	6	844
8	got.in	6	989
9	EC v3.0	6	1029
10	Julek team	6	1134
11	Selski babi	6	1316
12	kindergarten	6	1363
13	Prže Mače Folje	5	664
14	Samo touristicki	4	683
15	airwaves	4	695
16	Selski batki	4	768
17	LimaLi	3	495
18	Krótkie Porty	2	520
19	Nemas sanse, iskoristi ju!	2	837
20	Eukaliptusi	1	114
21	Ex Machina	0	-42
22	Gimnazija Sombor	0	-42

Problem A: Digits

Author:

Borna Vukorepa

Implementation and analysis:

Borna Vukorepa

Aleksandar Damjanović

Statement:

John gave Jack a very hard problem. He wrote a very big positive integer A_0 on a piece of paper. The number is less than 10^{200000}. In each step, Jack is allowed to put '+' signs in between some of the digits (maybe none) of the current number and calculate the sum of the expression. He can perform the same procedure on that sum and so on. The resulting sums can be labeled respectively by A_1, A_2 etc. His task is to get to a single digit number.

The problem is that there is not much blank space on the paper. There are only three lines of space, so he can't perform more than three steps. Since he wants to fill up the paper completely, he will perform exactly three steps.

Jack must not add leading zeros to intermediate results, but he can put '+' signs in front of digit 0. For example, if the current number is 1000100, $10 + 001 + 00$ is a valid step, resulting in number 11.

Input:

In the first line, a positive integer N, representing the number of digits of number A_0. In the second line, a string of length N representing number A_0. Each character is a digit. There will be no leading zeros.

Output:

Output exactly three lines, the steps Jack needs to perform to solve the problem. You can output any sequence of steps which results in a single digit number (and is logically consistent).

Constraints:

- $1 \leq N \leq 200\,000$

Example input 1:	***Example input 2:***	***Example input 3:***
2 10	3 992	4 1234

Example output 1:	***Example output 2:***	***Example output 3:***
10 1+0 1	99+2 1+01 2	123+4 1+2+7 1+0

Explanation:

Example 1: In the first step, we use zero '+' signs, so $A_1 = 10$. In the second step, we place a '+' sign between 1 and 0, so $A_2 = 1 + 0 = 1$. In the third step, we don't need to (and we can't) put any '+' signs, so we get $A_3 = 1$.

Example 2: In the first step, we only put a '+' between the last two digits, so we get $A_1 = 99 + 2 = 101$. In the second step, we place the only '+' sign between the first and the second digit, so $A_2 = 1 + 01 = 1 + 1 = 2$. In the third step, we don't need to (and we can't) put any '+' signs, so we get $A_3 = 2$.

Example 3: In the first step, we use a '+' sign between the last two digits, so $A_1 = 123 + 4 = 127$. On the second step, we place a '+' sign between every two digits, so $A_2 = 1 + 2 + 7 = 10$. In the third step, we place a '+' sign between 1 and 0, so $A_3 = 1 + 0 = 1$.

Time and memory limit: 1s/ 128MB

Solution and analysis:

Let $ds(x)$ be the sum of digits of the number x.

Notice that if $ds(A_0) \leq 198$, then $ds(ds(ds(A_0)))$ is a single digit number, therefore we have solved the case $ds(A_0) \leq 198$.

After a little more thinking, we can extend it to $ds(A_0) \leq 288$, because 289 is the smallest number which can't be reduced to a single digit number in at most two steps. This is because number 199 can be transformed to single digit in 2 steps by using following transformations $1 + 99$, $1 + 0 + 0$. Now we know how to solve $ds(A_0) <= 288$. Now we will solve $288 \leq ds(A_0) \leq 999$.

Let $A_0 = a_0 a_1 \ldots a_{n-1} a_n$ be the decimal representation of A_0.

Consider:

$$X = a_0 a_1 + a_2 a_3 + \ldots + a_{n-1} a_n$$

$$Y = a_0 + a_1a_2 + \ldots + a_{n-2}a_{n-1} + a_n$$

(this is for an odd n, the following results would be the same with an even n).

Now look at: $X + Y = 11ds(A_0) - 9a_n$. (this is easy to be seen by addition).

Since $ds(A_0) \geq 288$, we have $X + Y > 10ds(A_0)$. Thus, we trivially conclude that for example: $X > 5ds(A_0)$.

That means that $X > 1000$. Now look at the following sequence of numbers:

$$a_0 + a_1 + a_2 + a_3 + \ldots + a_{n-1} + a_n,$$

$$a_0a_1 + a_2 + a_3 + \ldots + a_{n-1} + a_n,$$

$$a_0a_1 + a_2a_3 + \ldots + a_{n-1} + a_n,$$

$$\ldots,$$

$$X = a_0a_1 + a_2a_3 + \ldots + a_{n-1}a_n .$$

It is increasing, and the first element has 3 digits, the last has at least 4, so the number of digits has increased at some point. Since the numbers in the sequence are increasing by at most 81 (trivial check), the sequence element, when digit skipping occurred, is at most 1080. Thus, our first step is to put '+' signs as they are in that sequence element. After that, we apply $ds()$ two more times and we will be done (trivial check).

Now we solve $ds(A_0) \geq 1000$.

Consider:

$$X = a_0a_1a_2 + a_3a_4a_5 + \ldots + a_{n-2}a_{n-1}a_n$$

$$Y = a_0 + a_1a_2a_3 + a_4a_5a_6 + \ldots + a_{n-4}a_{n-3}a_{n-2} + a_{n-1} + a_n$$

$$Z = a_0 + a_1 + a_2a_3a_4 + a_5a_6a_7 + \ldots + a_{n-3}a_{n-2}a_{n-1} + a_n.$$

(this is for the n of the form $n = 3k + 2$, the following results would be the same with the other n)

As before, we can easily see that $X + Y + Z > 98ds(A_0)$ ($90ds(A_0)$ is enough), so, for example, $X > 30ds(A_0)$.

Now look at the following sequence of numbers:

$$a_0 + a_1 + a_2 + a_3 + \ldots + a_{n-1} + a_n,$$

$$a_0a_1a_2 + a_3 + a_4 + \ldots + a_{n-1} + a_n,$$

$$a_0a_1a_2 + a_3a_4a_5 + \ldots + a_{n-2} + a_{n-1} + a_n,$$

$$\ldots,$$

$$X = a_0a_1a_2 + a_3a_4a_5 + \ldots + a_{n-2}a_{n-1}a_n$$

It is increasing and since the first number is $ds(A_0)$ and the last one is at least $30ds(A_0)$, the digit skipping occurred somewhere.

Let $M = a_0a_1a_2 + a_3a_4a_5 + \ldots + a_{3k}a_{3k+1}a_{3k+2} + a_{3k+3} + \cdots + a_{n-2} + a_{n-1} + a_n$ be the sequence element right before the digit skipping occurred. Therefore, it is less than $10ds(A_0)$ and at most 999 away from power of 10. Now we continue by 'steps of two', so the next element is:

$$a_0a_1a_2 + a_3a_4a_5 + \ldots + a_{3l}a_{3k+1}a_{3k+2} + (a_{3k+3}a_{3k+4} + a_{3k+5}) + \cdots + a_{n-2} + a_{n-1} + a_n,$$

and the next one:

$$a_0a_1a_2 + a_3a_4a_5 + \ldots + a_{3l}a_{3k+1}a_{3k+2} + (a_{3k+3}a_{3k+4} + a_{3k+5}) + (a_{3k+6}a_{3k+7} + a_{3k+8}) + \cdots + a_{n-2} + a_{n-1} + a_n,$$

Remember that $M < 10ds(A_0)$. Since $X > 30ds(A_0)$, if we continued making steps of three, we would increase our number by at least $20ds(A_0)$ before the end of the procedure. Our steps are steps of two, and each step increases the number by at least $1/11$ of what it would be increased by if we were making steps of three (trivial check). Thus, before the end of our procedure, we will increase M by more than $ds(A_0)$. Since $ds(A_0) \geq 1000$, and remember that M was at most 999 less than the power of 10, the digit skipping will indeed occur.

We thus reach the similar situation as in the previous case and we know what our first step is. We only need to apply $ds()$ two more times after that.

Problem B: Neural Network Country

Author:
Nikola Nedeljković

Implementation and analysis:
Janko Šušteršič
Nikola Nedeljković

Statement:

Due to the recent popularity of the *Deep learning* new countries are starting to look like *Neural Networks*. That is, the countries are being built *deep* with many layers, each layer possibly having many *cities*. They also have *one* entry, and *one* exit point. There are exactly L layers, each having N cities. Let us look at the two adjacent layers L_1 and L_2. Each city from the layer L_1 is connected to each city from the layer L_2 with the travelling cost c_{ij} for $i, j \in (1, 2, .., N)$, and each pair of adjacent layers has the same cost in between their cities as any other pair (they just stacked the same layers, as usual). Also, the travelling costs to each city from the layer L_2 are same for all cities in the L_1, that is c_{ij} is the same for $i \in (1, 2, .., N)$, and fixed j. *Doctor G.* needs to speed up his computations for this country so he asks you to find the number of paths he can take from *entry* to *exit* point such that his travelling cost is divisible by given number M.

Input:

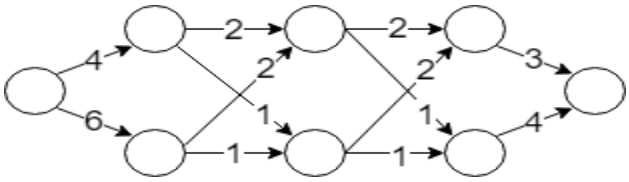
The first line of input contains N – the number of cities in each layer, L – the number of layers, and M. Second, third and fourth line contain N integers denoting costs from *entry* point to the first layer, costs between adjacent layers as described above, and costs from the last layer to the *exit* point.

Output:

Output a single integer, the number of paths *Doctor G.* can take which have total cost divisible by M, modulo $10^9 + 7$.

Example input:	*Example output:*
2 3 13 4 6 2 1 3 4	2

Example explanation:



This is a country with 3 layers, each layer having 2 cities. Paths $6 \to 2 \to 2 \to 3$, and $6 \to 2 \to 1 \to 4$ are the only paths having total cost divisible by 13. Notice that input edges for layer cities have the same cost, and that they are same for all layers.

Constraints:

- $1 \leq N \leq 10^6$
- $2 \leq L \leq 10^5$
- $2 \leq M \leq 100$
- $0 \leq costs < M$

Time and memory limit: 2s / 256 MB

Solution and analysis:

Let's ignore constraints for a second and try to solve the problem using a classical dynamic approach. We can easily see that if we know in how many ways we can reach some layer for every modulo up to M, we can simply calculate number of ways we can reach the next layer for every modulo by iterating over weights in $O(N)$. That means we have matrix $Dp_{L \times M}$, where $Dp[i][j]$ is equal to the number of roads up to i-th layer whose total cost is equal to j modulo m. The transition between the layers is calculated as follows:

$$Dp[i][j] = \sum_{k=0}^{N-1} Dp[i-1][(j - w[k]) \bmod M],$$

where w is a zero-base indexed array of weights between the layers. Given the array of costs between the starting point and the first layer, a, the base is calculated as $Dp[0][a[k]] \mathrel{+}= 1$, for every k from 0 to $N-1$. The last layer is a bit tricky. In order to calculate the final result, we need to know not only $Dp[L][0..N-1]$, but also which of those roads end in which nodes. We can achieve that by calculating Dp up to the layer $L-1$, and computing the last step manually, by "merging" the costs of the last two sets of edges, by simply adding $w[i]$ and $b[i]$, where b is the array of edge costs between the last layer and the finish point, for every i from 0 to $N-1$, and outputting the following result modulo Q:

$$res = \sum_{i=0}^{N-1} Dp[L-1][(-w[i] - b[i]) \bmod M],$$

giving the overall time complexity of $O(L \cdot N \cdot M)$, and memory complexity of $O(L \cdot M)$, both of them exceeding the given limits. Of course, memory is not a problem if we see that we need not to save the whole matrix, but only the last column.

We will reduce time complexity step by step. If we notice that it doesn't matter in which node we are currently in (except in the last step), we can speed up the computation by saving the number of occurrences of each $w[i]$ modulo M into an array $num[i]$. Now Dp matrix is calculated in $O(L \cdot M^2)$ using formula:

$$Dp[i][j] = \sum_{k=0}^{M-1} num[k] \cdot Dp[i-1][(j-k) \bmod M].$$

Now, a keen eye can spot multiplication of a matrix and a vector by looking at this formula. Indeed, we can compute a transition matrix of dimensions $M \times M$ which, multiplied by a vector of the current state (number of roads which have the total cost $i = 0..M-1$, modulo M up to some layer), gives us the next state vector, corresponding to the next layer, resulting in time complexity $O(M^3 \cdot \log L)$, using modular matrix exponentiation. The last layer still has to be treated separately, in the same way as described before.

Problem C: Property

Author:

Aleksandar Damjanović

Slavko Ivanović

Implementation and analysis:

Aleksandar Damjanović

Slavko Ivanović

Statement:

Bill is a famous matematician in BubbleLand. Thanks to his revolutionary math discoveries he was able to make enough money to build a beautiful house. Unfortunately, for not paying property tax on time, court decided to punish Bill by making him lose a part of his property.

Bill's property can be observed as a convex regular $2n$-sided polygon $A_0A_1 \ldots A_{2n-1}A_{2n}$, $A_{2n} = A_0$ with sides of the **exactly** 1 meter in length.

Court rules for removing part of his property are as follows:

- Split every edge $A_kA_{k+1}, k = 0 \ldots 2n-1$ in n **equal parts** of size $\frac{1}{n}$ with points $P_0, P_1, \ldots P_{n-1}$
- On every edge $A_{2k}A_{2k+1}, k = 0 \ldots n-1$ court will choose **one point** $B_{2k} = P_i$ for some $i = 0, \ldots, n-1$ such that $\bigcup_{i=0}^{n-1} B_{2i} = \bigcup_{i=0}^{n-1} P_i$
- On every edge $A_{2k+1}A_{2k+2}, k = 0 \ldots n-1$ Bill will choose **one point** $B_{2k+1} = P_i$ for some $i = 0, \ldots, n-1$ such that $\bigcup_{i=0}^{n-1} B_{2i+1} = \bigcup_{i=0}^{n-1} P_i$
- Bill gets to keep property inside of $2n$-sided polygon $B_0B_1 \ldots B_{2n-1}$

Luckily, Bill found out all B_{2k} points the court chose. Even though he is a great mathematician, his house is very big and he has a hard time calculating. Therefore, he is asking you to help him choose points in order to maximize his property area.

Input:

The first line contains one integer number *n* representing number of edges of Bill's $2n$-sided polygon house.

The second line contains n distinct integer numbers $B_{2k}, k = 0 \ldots n-1$, separated by a single space, representing points the court chose. If $B_{2k} = i$, the court chose point P_i on side $A_{2k}A_{2k+1}$.

Output:

Output contains n distinct numbers separeated by a single space representing points Bill shoud choose in order to maximize the property area. If there are multiple solutions that maximize the area, return any solution which maximizes the area.

Constraints:

- $2 \leq n \leq 50\,000$
- $0 \leq B_{2k} \leq n-1, k = 0 \ldots n-1$
- $\bigcup_{i=0}^{n-1} B_{2i} = \{0, 1, 2, \ldots n-1\}$

Example input:

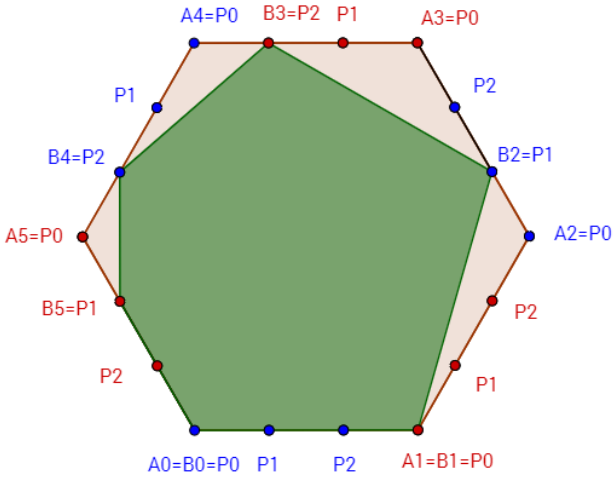
```
3

0 1 2
```

Example output:

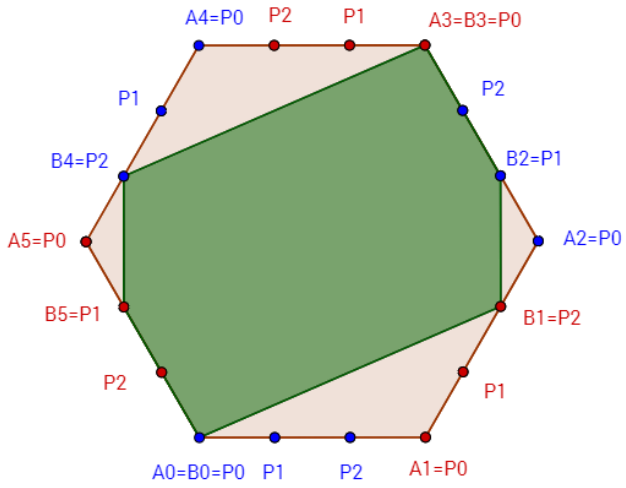
```
0 2 1
```

Explanation:

Court chose points $B_0 = P_0$, $B_2 = P_1$ and $B_4 = P_2$ as described in the image below. If Bill chooses points $B_1 = P_0$, $B_3 = P_2$ and $B_5 = P_1$ he will achieve the maximum area as shown in the image below:



On the other hand, if Bill chooses points $B_1 = P_2$, $B_3 = P_0$ and $B_5 = P_1$, the area of his green property will be smaller:



Time and memory limit: 0.5s / 128 MB

Solution and analysis:

Let's first figure out what is the easiest way to calculate the area of the $B_0B_1 \ldots B_{2n-1}$.

We can easily determine the area of of $B_0B_1 \ldots B_{2n-1}$ polygon by subtracting a purple triangle area from the initial $A_0A_1 \ldots A_{2n-1}$ polygon (Figure 1).

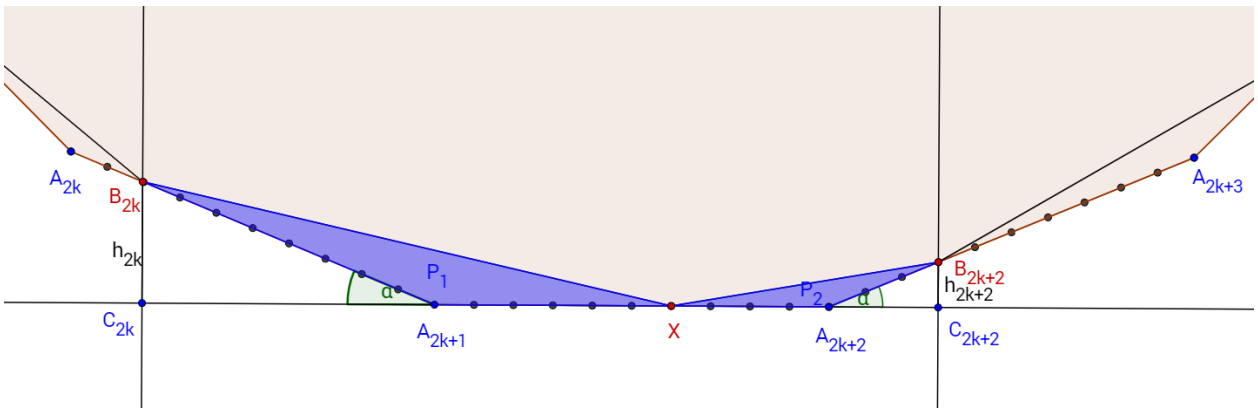


Figure 1

Determining the maximum polygon area $B_0B_1 \dots B_{2n-1}$ we can achieve is equivalent to determining the minimum purple triangle area we can achieve, which is a much easier task.

Let's say the court chose points B_{2k} and B_{2k+2} (Figure 1). We now need to determine our point $X = B_{2k+1}$ to minimize sums of areas P_1 and P_2.

$$P_1 + P_2 = \frac{A_{2k+1}X * B_{2k}C_{2k}}{2} + \frac{XA_{2k+2} * B_{2k+2}C_{2k+2}}{2}$$

$$P_1 + P_2 = \frac{x * B_{2k}C_{2k}}{2} + \frac{(s-x) * B_{2k+2}C_{2k+2}}{2}$$

$$P_1 + P_2 = \frac{s * B_{2k+2}C_{2k+2}}{2} + \frac{x * (B_{2k}C_{2k} - B_{2k+2}C_{2k+2})}{2}$$

$$P_1 + P_2 = const + \frac{x * (h_{2k} - h_{2k+2})}{2}$$

Therefore, to minimize the total triangle area, we need to choose right x for $h_{2k} - h_{2k+2}$. Because our points represent a permutation of $\{P_0, P_1, \dots, P_n\}$, best way to choose points will be to take minimum x when $h_{2k} - h_{2k+2}$ is maximal, and maximum x, when $h_{2k+1} - h_{2k+2}$ is minimal.

Writing algorithm is now easy. For polygon create the array $diffh[k]$ = $(h_{2k} - h_{2k+2}, k)$, sort it in an ascending order and give to $sortedDiffh[k].edge$ point P_{n-k-1}.

The overall algorithm complexity is $O(NlogN)$.

Problem D: Exploration plan

Author:
Branko Fulurija

Implementation and analysis:
Branko Fulurija
Ivan Dejković

Statement:

The competitors of Bubble Cup X gathered after the competition and discussed what is the best way to get to know the host country and its cities.

After exploring the map of Serbia for a while, the competitors came up with the following facts: the country has V cities which are indexed with numbers from 1 to V, and there are E bi-directional roads that connect the cites. Each road has a weight (the time needed to cross that road). There are N teams at the Bubble Cup and the competitors came up with the following plan: each of the N teams will start their journey in one of the V cities, and some of the teams share the starting position.

They want to find the **shortest** time T, such that every team can move in these T minutes, and the number of **different** cities they **end up in** is at least K (because they will only get to know the cities they end up in). A team doesn't have to be on the move all the time, if they like it in a particular city, they can stay there and wait for the time to pass.

Please help the competitors to determine the **shortest** time T so it's possible for them to end up in at least K different cities or print -1 if that is impossible no matter how they move.

Note that there could exist multiple roads between some cities.

Input:

The first line contains four integers: V, E, N and K, the number of cities, the number of roads, the number of teams and the smallest number of different cities they need to end up in. The second line contains N integers, the cities where the teams start their journey. The next E lines contain information about the roads in following format: A_i B_i T_i, which means that there is a road connecting cities A_i and B_i, and you need T_i minutes to cross that road.

Output:

Output a single integer that represents the **minimal** time the teams can move for, such that they **end up in** at least K different cities or output -1 if there is no solution.

Example input:

```
6 7 5 4
5 5 2 2 5
1 3 3
1 5 2
1 6 5
2 5 4
2 6 7
3 4 11
3 5 3
```

Example output:

```
3
```

Example explanation:

Three teams start from city 5, and two teams start from city 2. If they agree to move for 3 minutes, one possible situation would be the following: Two teams in city 2, one team in city 5, one team in city 3 , and one team in city 1. And we see that there are four different cities the teams end their journey at.

Constraints:

- $1 \leq V \leq 600$
- $1 \leq E \leq 20\,000$
- $1 \leq N \leq 200$
- $1 \leq K \leq N$
- $1 \leq Ai, Bi \leq V$
- $1 \leq Ti \leq 10\,000$
- The result will be no greater than 1731311 if the solution exists

Time and memory limit: 2s / 256 MB

Solution and analysis:

The first step we should do is to precompute the All-pairs shortest path table. We can do this with Dijkstra's algorithm from every node in time $O(V^2 logE)$ or with Floyd Warshall in time $O(V^3)$.

The next step is to notice that we can use a binary search to find the answer, because if the teams end up in at least K cities in some time T, they can do that in every greater time (they can just remain in the same cities as in the time T).

The final step to our solution is to create a function $bool\ can(int\ T)$, that for some time T, which we guess in our binary search, tells if the requirements are met (at least K different cities). This can be

solved as follows: We will create a bipartite graph, where on the left side we have every starting city of our teams and on the right side the remaining cities. Then for every starting city on the left, we will create an edge to every city on the right that can be reached within time T (here we use our APSP table). Now for some fixed time T, we have a bipartite graph with starting cities on the left, that have an edge to every city they can reach within time T. After we have this graph, it's not hard to see that we now have a maximum bipartite matching problem. We just have to check whether the MBP is greater than or equal to K. Our bipartite graph will have at most N vertices on the left, and at most NV edges, so if we use Ford Fulkerson algorithm for matching the time complexity of this part will be $O(N^2V)$.

The total time complexity is: $O(V^2 \log E + N^2V\ log(\text{maxAnswer}))$

Problem E: Casinos and travel

Author:

Borna Vukorepa

Implementation and analysis:

Borna Vukorepa

Predrag Ilkić

Statement:

John has just bought a new car and is planning a journey around the country. Country has N cities, some of which are connected by bidirectional roads. There are $N-1$ roads and every city is reachable from any other city. Cities are labeled from 1 to N.

John first has to select from which city he will start his journey. After that, he spends one day in a city and then travels to a randomly choosen city which is directly connected to his current one and which he has not yet visited. He does this until he can't continue obeying these rules.

To select the starting city, he calls his friend Jack for advice. Jack is also starting a big casino business and wants to open casinos in some of the cities (max 1 per city, maybe nowhere). Jack knows John well and he knows that if he visits a city with a casino, he will gamble exactly once before continuing his journey.

He also knows that if John enters a casino in a good mood, he will leave it in a bad mood and vice versa. Since he is John's friend, he wants him to be in a good mood at the moment when he finishes his journey. John is in a good mood before starting the journey.

In how many ways can Jack select a starting city for John and cities where he will build casinos such that no matter how John travels, he will be in a good mood at the end? Print answer $mod\ 10^9 + 7$.

Input:

In the first line, a positive integer N, the number of cities.

In the next $N-1$ lines, two numbers a, b separated by a single space meaning that cities a and b are connected by a bidirectional road.

Output:

Output one number: the number of ways Jack can make his selection $mod\ 10^9 + 7$.

Constraints:

- $1 \leq N \leq 200\,000$
- $1 \leq a, b \leq N$

Example input 1:

```
2

1 2
```

Example input 2:

```
3

1 2

2 3
```

Example output 1:

```
4
```

Example output 2:

```
10
```

Example 1 explanation:

If Jack selects city 1 as John's starting city, he can either build 0 casinos, so John will be happy all the time, or build a casino in both cities, so John would visit a casino in city 1, become unhappy, then go to city 2, visit a casino there and become happy and his journey ends there because he can't go back to city 1. If Jack selects city 2 for start, everything is symmetrical, so the answer is 4.

Example 2 explanation:

If Jack tells John to start from city 1, he can either build casinos in 0 or 2 cities (total 4 possibilities). If he tells him to start from city 2, then John's journey will either contain cities 2 and 1 or 2 and 3. Therefore, Jack will either have to build no casinos, or build them in all three cities. With other options, he risks John ending his journey unhappy. Starting from 3 is symmetric to starting from 1, so in total we have $4 + 2 + 4 = 10$ options.

Time and memory limit: 1s / 256 MB

Solution and analysis:

After examining the problem, it is easy to see that the statement can be reduced to:

Given a tree, in how many ways can you select a root and color every node black or white such that all paths from root to any leaf node (except the root) have even number of black nodes.

Assume we have selected a root and color the tree arbitrarily. Select any path from root to some leaf. Notice that whatever the number of black nodes is on this path, the parity can be adjusted by changing the color of the corresponding leaf if needed. Therefore, for every path mentioned in the problem, its parity is ultimately determined by the leaf node color and all the other nodes can be colored in any way. Therefore, the number of colorings for a fixed root is $2^{N-|leaves|}$. We only need to count, for every

root, how many leaves are there. That is easy. If a root is a leaf itself, the number of leaves of such rooted tree is the number of leaves of our unrooted tree, minus the root, otherwise it is just the number of leaves of the unrooted tree.

The explicit formula is (N is the number of nodes and L the number of leaves in the rooted tree):

$$S = L * 2^{N-L+1} + (N - L) * 2^{N-L}$$

Problem F: Product transformation

Author:

Nikola Nedeljković

Implementation and analysis:

Janko Šušteršič

Nikola Nedeljković

Statement:

Consider an array A with N elements, all being the same number a. Define the product transformation as a *simultaneous* update $A_i = A_i * A_{i+1}$, that is multiplying each element to the element right to it for $i \in (1, 2, .., N-1)$, with the last number A_N remaining the same. For example, if we start with an array A with $a = 2$ and $N = 4$ after one product transformation $A = [4, 4, 4, 2]$, and after two product transformations $A = [16, 16, 8, 2]$. Your simple task is to calculate the array A after M product transformations. Since the numbers can get quite big you should output them modulo Q.

Input:

The first and only line of input contains four integers N, M, a, Q.

Output:

You should output the array A from left to right, space separated.

Constraints:

- $7 \leq Q \leq 10^9 + 123$
- The multiplicative order of a number a modulo Q, $\phi(a, Q)$ is prime.
- $1 \leq M, N < \phi(a, Q) \leq 10^6 + 123$
- $2 \leq a \leq 10^6 + 123$

Example input:

```
2 2 2 7
```

Example output:

```
1 2
```

Example explanation:

After 2 transformations $A = [8, 2]\ mod\ 7 = [1, 2]$.

Time and memory limit: 2s / 64 MB

Note:

The multiplicative order of a number a modulo Q $\phi(a, Q)$, is the smallest natural number x such that $a^x mod\ Q = 1$. For example, $\phi(2, 7) = 3$.

Solution and analysis:

For example, let's consider an array $A = [a, a, a, a, a], (N = 5)$. After four product transformations that array becomes $A = [a^{16}, a^{15}, a^{11}, a^{5}, a]$. By writing only one example, one cannot see the pattern that easily. If we write out multiple examples for different $M's$, we might notice that the differences of exponents have somewhat familiar structure. In this case, written from left to right, we have $1\ 4\ 6\ 4$, and that resembles binomials, right? Indeed, it is not hard to prove mathematically, relying on recurrent formula $C_k^n = C_{k-1}^{n-1} + C_k^{n-1}$, that after M product transformations i-th element of zero-base indexed array A is:

$$A_i = a^{C_0^M + C_1^M + \cdots + C_{N-i-1}^M}.$$

Given that the multiplicative order of number a modulo Q, $p = \phi(a, Q)$ is prime, we can speed up the computation by using Sieve of Eratosthenes for finding primes in $O(p \cdot \log\log p)$, and asking if $a^x \ mod \ Q = 1$ in $O(\log\ x)$. If we approximate the prime-counting function $\pi(x)$ with $\pi(x) \approx \frac{x}{\ln x}$, it gives us an overall complexity of $O(p + p \cdot \log\log p)$ which is a bit faster than a solution which doesn't use Sieve of Eratosthenes, which runs in $O(p \cdot \log p)$ with a big multiplicative constant, knowing that calculating modulus is expensive.

Now, let us denote $D_i = C_0^M + C_1^M + \cdots + C_{N-i-1}^M$, for simplicity. Given the number p which we have previously determined as $p = \phi(a, Q)$, it stands:

$$a^{D_i} \ mod \ Q = a^{D_i \ mod \ p} \ mod \ Q.$$

Since p is also a prime number, we can find each C_k^n in $O(1)$ if we precompute all factorials and their inverses up to N (which can be done in $O(N \cdot \log N)$, given that p is prime and bigger than N, as stated in the constraints). Now we can find all $D_i's$ easily, in linear time. We also do a standard modular exponentiation algorithm to output final result in each iteration, yielding total complexity of

$$O(p \cdot \log\log p + N \cdot \log N + N \cdot \log p).$$

Problem G: Bathroom terminal

Author:

Aleksandar Damjanović

Implementation and analysis:

Miloš Šuković

Miloš Kruškonja

Statement:

Smith wakes up at the side of a dirty, disused bathroom, his ankle chained to pipes. Next to him is tape-player with a hand-written message "Play Me". He finds a tape in his own back pocket. After putting the tape in the tape-player, he sees a key hanging from a ceiling, chained to some kind of a machine, which is connected to the terminal next to him. After pressing a Play button a rough voice starts playing from the tape:

"Listen up Smith. As you can see, you are in pretty tough situation and in order to escape, you have to solve a puzzle.

You are given N strings which represent words. Each word is of the maximum length L and consists of characters 'a'-'e'. You are also given M strings which represent patterns.

Pattern is a string of length $\leq L$ and consists of characters 'a'-'e' as well as the maximum 3 characters '?'. Character '?' is an unknown character, meaning it can be equal to any character 'a'-'e', or even an empty character.

For each pattern find the number of words that matches with the given pattern. After solving it and typing the result in the terminal, the key will drop from the ceiling and you may escape.

Let the game begin."

Help Smith escape.

Input:

The first line of input contains two integers N and M, representing the number of words and patterns respectively. The next N lines represent each word, and after those N lines, following M lines represent each pattern.

Output:

Output contains M lines and each line consists of one integer, representing the number of words that match the corresponding pattern.

Constraints:

- $1 \leq N \leq 100\,000$
- $1 \leq M \leq 5000$
- $0 \leq L \leq 50$

Example input:

```
3 1
abc
aec
ac
a?c
```

Example output:

```
3
```

Example explanation:

If we switch '?' with 'b', 'e' and with empty character, we get 'abc', 'aec' and 'ac' respectively.

Time and memory limit: 2s / 256 MB

Solution and analysis:

Let's first put all given words in trie of maximum depth L in $O(N * L)$ time. Once this is done every node in the trie will contain how many words end at that node.

Now for each pattern we need to check how many words inside the trie satisfy the pattern. For every character 'a'-'e' in pattern we iterate though trie character by character. When we reach '?' we need to recursively sum the count of all possible letter choices, by this you have to recursively process all children of the current trie node, by continuing iteration. For the empty character, you have to stay at current trie node, but process the next character in the pattern, also beware of patterns with multiple consecutive '?' such as 'a???b' or 'a??' and patterns like '?aaaa?', with same caracters between two '? ', because while searching the trie, you may find same words multiple times. To solve this, for each pattern make a set of trie node pointers, which point to end nodes of found words for current pattern, so when you find a word next time, first check if the word is in the set, before counting it.

As there can be maximum 3 '?' character in a pattern, calculating how many strings in a trie satisfy a pattern will be performed $O(5\text{^}3 * M * L)$.

Also there is a bit slower solution. Use lexicographic sort on input words, generate all possible words from every pattern, and search every generated word in the sorted word vector. This solution is slower by O(logN).

Problem H: Bob and stages

Author:

Aleksandar Damjanović

Ivan Dejković

Implementation and analysis:

Ivan Dejković

Slavko Ivanović

Statement:

The citizens of BubbleLand are celebrating thier 10th anniversary so they decided to organize a big music festival. Bob got a task to invite N famous singers who would sing on the fest. He was too busy placing stages for their performances that he totally forgot to write the invitation e-mails on time, and unfortunately he only found K available singers. Now there are more stages than singers leaving some of the stages empty. Bob would not like if citizens of BubbleLand noticed empty stages and found out that he was irresponsible.

Because of that he decided to choose exactly K stages that form a **convex** set, make large posters as edges of that convex set and hold festival inside. While those large posters will make it impossible for citizens to see empty stages outside Bob still needs to make sure they don't see any of the empty stages inside that area

Since lots of people are coming, he would like that the festival area is as large as possible. Help him calculate the **maximum** area that he could obtain respecting the conditions. If there is no such area, the festival cannot be organized and the answer is 0.00.

Input:

The first line of input contains two integers N and K, separated with one empty space, representing number of stages and number of singers, respectively. Each of the next N lines contains two integers X_i and Y_i, representing the coordinates of the stages.

Output:

Output contains only one line with one number, rounded to two decimal points: the maximal festival area.

Example input:

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

Example output:

```
10.00
```

Example explanation:

From all possible convex polygon with 4 vertices and no other vertex inside, the largest is one with points (0, 0), (2, 1), (4, 4) and (1, 5).

Constraints:

- $3 \leq N \leq 1000$
- $3 \leq K \leq min(N, 50)$
- $0 \leq Xi, Yi \leq 10^9$
- There are no three or more collinear points

Time and memory limit: 2s / 256 MB

Solution and analysis:

For each point p from the given points we will find the maximal area of all convex K-gons which have p as leftmost vertex. After setting point p and removing all points to the left of p, we sort the rest of the points by angle around p. If we connect the points in the order they are sorted in, and the first and the last point with p, we get a star-shaped polygon P_p. Each convex polygon that has p as leftmost vertex must lie inside P_p. Next, we compute the visibility graph VG_p in such a polygon and use dynamic programming to find the polygon with maximum area.

We will construct the visibility graph during one counter-clockwise scan around the polygon. Let's say that we have M points in P_p different from p. We won't include point p in visibility graph. When we visit p_i we construct all incoming edges of p_i. With each vertex p_i we maintain a queue Q_i that stores the starting points of some of the incoming edges of p_i in counter-clockwise order. It contains those points p_j such that ji is an edge of the visibility graph and we have not yet reached another point p_k with $k > i$ such that jk is an edge of the visibility graph. The following pseudo code describes the algorithm for computing the visibility graph in a star-shaped polygon.

```
procedure CreateVisibilityGraph
    for i := 1 to M do Q_i := ∅ end
    for i := 1 to M - 1 do Proceed(i, i + 1) end

procedure Proceed(i, j)
    while Q_i ≠ ∅ and IsLeftTurn(Front(Q_i)i, ij) do
        Proceed (FRONT(Q_i), j)
        Pop(Q_i)
    end
    Connect(i, j)
    Push(i, Q_j)
```

Time complexity for computing visibility graph is $O(|VG|)$ since every call of the Proceed adds one edge to the VG.

Next, we will use dynamic programming over the visibility graph for computing the maximal area of all polygons with p as leftmost vertex. Each of these polygons is uniquely determined by one convex chain, a subset of the VG, which has K-2 edges (obtained by removing two edges from p). For each edge e of the VG, and for each d, $1 \leq d \leq m$-2 we will determine $DP[e][d]$ - the maximum area of all polygons whose corresponding convex chain starts with e and which is d edges long.
We will treat the vertices clockwise. Assume that we are at some vertex p_i. Let the incoming edges of p_i be in_1,..., in_{inmax} and the outgoing edges out_1,..., out_{outmax} both ordered counter-clockwise by angle. Note that the algorithm for computing the visibility graph inside P gives us the edges in this order. For all outgoing edges we know the maximal areas of polygons which corresponding chain starts there.
We will treat the incoming edges in the reversed order, starting at $inmax$. For this first incoming edge we look at all outgoing edges that form a convex angle with it. Let m_{d-1} be the maximal value of $DP[o][d-1]$ among them. Then $DP[inmax][d] = m + AreaOfTriangle(p, inmax)$. Clearly, all outgoing edges that form a convex angle with (i, j) form a convex angle with $(i, j\text{-}1)$. Hence, we don't have to check them again. Finding the maximal area takes time $O(K * |VG|)$, since we look at each edge twice.

The overall complexity of the algorithm is $O(KN^3)$.

References

[1] David P. Dobkin, Herbert Edelsbrunner, Mark H. Overmars, *Searching for Empty Convex Polygons* (1988)

Problem I: Dating

Author:
Branko Fulurija

Implementation and analysis:
Branko Fulurija
David Milićević

Statement:

This story is happening in a town named BubbleLand. There are n houses in BubbleLand. In each of these n houses lives a boy or a girl. People there really love numbers, and everyone has a favorite number f. That means that the boy or the girl that lives in the $i-th$ house has a favorite number equal to f_i .

The houses are numerated with numbers from 1 to n.

The houses are connected with $n-1$ bi-directional roads and you can travel from any house to any other house in the town. There is exactly one path between every pair of houses.

A new dating agency had opened their offices in BubbleLand and the citizens were very excited. They immediately sent q questions to the agency and each question was in the following format:

$a\ b$ – asking how many ways there are to choose a couple (boy and girl) that have the same favorite number and live in one of the houses on a unique path from house a to house b.

Help the dating agency answer the questions and grow their business.

Input:

The first line contains an integer n, the number of houses in the town.

The second line contains n integers, where the i-th number is 1 if a boy lives in the i-th house or 0 if a girl lives in the i-th house.

The third line contains n integers, where the i-th number represents the favorite number f_i of the girl or the boy that lives in the i-th house.

The next $n-1$ lines contain information about the roads and the i-th line contains two integers a_i and b_i which means that there exists a road between these two houses.

The following line contains an integer q, the number of questions.

Each of the following q lines represents a question and consists of two integers a and b.

Output:

For each of the q questions output a single number, the answer to the citizens' questions.

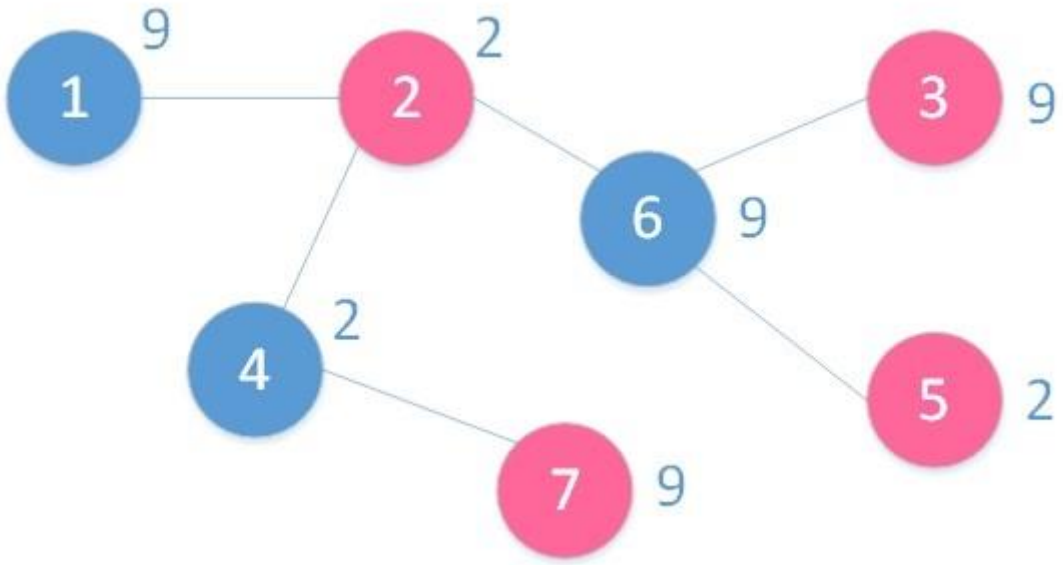
Example input:

```
7
1 0 0 1 0 1 0
9 2 9 2 2 9 9
2 6
1 2
4 2
6 5
3 6
7 4
2
1 3
7 5
```

Example output:

```
2
3
```

Example explanation:



Blue nodes represent houses where boys live and pink those where girls live, and the numbers beside nodes represent their favorite numbers.

In the first question from house 1 to house 3, the potential couples are: (1, 3) and (6, 3).

In the second question from house 7 to house 5, the potential couples are: (7, 6), (4, 2) and (4, 5).

Constraints:

- $1 \leq n \leq 10^5$
- $1 \leq q \leq 10^5$
- $1 \leq f_i \leq 10^9$

Time and memory limit: 3s / 64MB

Solution and analysis:

This problem is solved with Mo's algorithm on tree.

Flatten the tree in an array by doing a modified DFS preorder traversal. For every node, we must calculate $ST[u]$ and $EN[u]$.

$ST[u]$ represents the start time when we entered the node u in our DFS and $EN[u]$ represents the time when we finished exploring the node u and its subtree.

$ST[u]$ and $EN[u]$ will give us the position of the node u in the flattened array.

Now when we must answer query $u - v$, we split the path $u - LCA$ and $LCA - v$. Core of the algorithm is to see what ranges in our arrays ST and EN we should consider:

Case 1: $LCA == u$:

In this case, our query range would be $[\ ST[u],\ ST[v]\ ]$.

Case 2: $LCA\ != u$

In this case, our query range would be $[\ EN[u],\ ST[v]\ ]\ +\ [\ ST[LCA],\ ST[LCA]\ ]$.

We should ignore every node that appears twice or zero on that range (if it appears twice that means we finished processing that node and it isn't a part of that path). With all this considered we can solve the problem with Mo's algorithm by decomposing the queries in buckets of the size $sqrt(n)$. While moving L and R pointers we add and remove nodes and calculate answer on the fly with a counter array for nodes with girls and for nodes with boys. For example, when we add a node with a boy, we can do the following: $res\ +=\ countGirl[favoriteNumber]$. Similarly, for removing nodes from range. Time complexity $O(n\ sqrt(n))$

References

[1] Mo's Algorithm on Trees: http://codeforces.com/blog/entry/43230

Qualifications

The qualifications were split into two rounds with ten problems in each round and 25 days for contestants to solve them. Teams were given one point for each successfully solved problem in the first qualification round. Successfully solving a problem from second round provided all teams two points, besides challenge problem which was worth maximum six points.

This year Bubble Cup website was fully integrated with Petlja having ability for the first time to create custom online programming competitions.

We are extremely happy with our partnership with Sphere Online Judge, Caribbean Online Judge and DMOJ who provided problems for Bubble Cup qualifications this year, for the most diverse problem set yet. During first round problems from Sphere Online Judge and Caribbean Online Judge were used, while for second round there was at least one problem from all three online judge platforms.

Because so many elite teams applied, problems in Round 2 were especially difficult making competitors not only having to optimize on time, but also on space. Even though all problems from qualification rounds were solved still this remains as one of the toughest Round 2 qualification problem sets in Bubble Cup history.

Competitors from all around the world participated in the qualifications. Teams that qualified for the finals were from Serbia, Croatia, Poland, Latvia, Ukraine, Belarus and Russia.

Qualification results

Num	Problem name	Accepted solutions
01	Amusement Park	106
02	City Major	43
03	FF's Divisors	150
04	Game with Psyho	113
05	Homework with Fibonacci	64
06	I am a Good Sublist	74
07	Legend of Heta	76
08	Maximum Child Sum	62
09	On Time	65
10	Sauron's Army	125

Num	Problem name	Accepted solutions
01	Akame	27
02	Arrow	36
03	Line Sweep	37
04	Mine and Tree	32
05	Primitive Pythagorean Pairs	17
06	Periodic Function	5
07	Travelling Knight	2
08	Delivery Game	41
09	[Challenge] Lawnmower	49
10	Terminus Est	25

We continued with our tradition that the contestants are the ones who are writing the solutions for qualifications problems. You should note that these solutions are not official - we cannot guarantee that all of them are accurate in general

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

Problem R1 01: Amusement Park

Fox Ciel is in the Amusement Park and now she is in a queue in front of the Ferris wheel. There are people (or foxes more precisely) in the queue: we use first people to refer to one at the head of the queue, and n-th people to refer to the last one in the queue.

There will be k gondolas, and the way we allocate gondolas looks like this:

- When the first gondolas come, the q_1 people at the head of the queue go into the gondolas.
- Then, when the second gondolas come, the q_2 people at the head of the remaining queue go into the gondolas.

...

- The remain q_k people go into the last (k-th) gondolas.

Note that q_1, q_2, ..., q_k must be positive. You can get from the statement that $\sum q_i = \mathrm{n}$ and $q_i > 0$.

You know, people don't like to share the gondolas with strangers, so your task is to find an optimal allocation way (that is, to find an optimal sequence q) to make people happy. For every pair of people i and j there exists a value uij which denotes a level of unfamiliarity. You can assume for $u_{ij} = u_{ji}$, for all i, j $(1 \leq \mathrm{i}, \mathrm{j} \leq \mathrm{n})$ and $u_{ii} = 0$ for all i $(1 \leq \mathrm{i} \leq \mathrm{n})$. Then an unfamiliar value of gondolas is the sum of the levels of unfamiliar between any pair of people that is in the gondolas.

A total unfamiliar value is the sum of unfamiliar values for all gondolas. Help Fox Ciel to find the minimal possible total unfamiliar value for some optimal allocation.

Input

The first line contains two integers n and k $(1 \leq \mathrm{n} \leq 1000 \text{ and } 1 \leq \mathrm{k} \leq \min(\mathrm{n}, 200))$ - the number of people in the queue and the number of gondolas. Each of the following n lines contains n integers - matrix u, $(0 \leq u_{ij} \leq 9, u_{ij} = u_{ji} \text{ and } u_{ii} = 0)$.

Output

Print an integer - the minimal possible total unfamiliar value.

Examples

Example1:

Input	Output
5 2 0 0 1 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0	0

Examle2:

Input	Output
8 3 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0	7

Time and memory limit: 4s/ 128MB

Solution and Analysis:

This problem is about dynamic programming. At first, let's consider first auxiliary dynamic: $f[i][j]$ – what is the value of the total level of unfamiliar if we take on board people with numbers from i to j. Obviously, we understand that this value is equal to

$$\sum_{i \le i1 \le j1 \le j} a[i1][j1].$$

It's the same as $$\frac{\sum_{i \le i1, j1 \le j} a[i1][j1]}{2}.$$

Therefore we should only calculate the sum of all elements in the rectangular in matrix a. We can perform it by usual partial sums, when we primarily calculate values

$$sum[i][j] = \sum_{0 \le i1 \le i, 0 \le j1 \le j} a[i1][j1]$$

by easy dynamic:

$$sum[i][j] = sum[i-1][j] + sum[i][j-1] - sum[i-1][j-1] + a[i][j]$$

(it's easy to see that we count every element once). After that, we can easily obtain

$$f[i][j] \text{ as } sum[j][j] - sum[i-1][j] - sum[i][j-1] + sum[i][i].$$

So now we can consider the main dynamic $dp[n][k]$- what is the minimal level of unfamiliar if we want to seat the first n people in k gondolas. To calculate next states, let's iterate over all possible numbers of people to take in next gondola (for example, s).

Then, we can update state $dp[n+s][k+1]$ with the value $(dp[n][k] + f[k][k+s])$.

The total complexity is $O(n^2)$ for preprocessing and $O(n^2k)$ for main dp (we have $O(nk)$ states and linear number of transitions in every state).

Resource: Caribbean Online Judge
Solution by:
Name: Nikolay Zhidkov
School: Saint Petersburg Academic University
E-mail: zhidkov102@gmail.com

Problem R1 02: City Major

Toby has got a lot of popularity since he defeated Humbertov Moralov in the Primoshkas war. He was elected mayor of Tobyland and now he is now running for the elections again. Unfortunately for him, the Rottweiler association controls all of the communications media. This association doesn't like Toby because they believe that only big and strong dogs should rule the city, and Toby is only a small and cute dog.

Hence Toby plans to create a new TV station that supports his cause. Toby is short on budget to buy the antennas, but he knows that he doesn't need to cover the entire population to win the elections. Toby's city is represented as a non-self-intersecting polygon, and, given the location of the TV station, your task is to find the minimum radius of coverage around the TV station such as that at least p percent of the population (p% of the polygon area) is inside the radius of coverage. Assuming that the TV station is located in the origin $(0, 0)$, the TV station could be inside or outside of the city polygon.

Input

The input consists of several test cases. Each test case begins with a line with two integers $3 \leq N \leq 100$ and $1 \leq P \leq 100$ representing the number of vertices and the percentage of coverage required. Then follows N lines with two integers xi and yi $(-100000 \leq xi, yi \leq 100000)$ indicating the position of vertex number i. The end of the input is given by the end of file.

Output

For each test case, print a single line containing the minimum radius of coverage as explained above. Print your answer with 2 decimal digits.

Examples

Example1:

Input	Output
4 100 1 1 1 -1 -1 -1 -1 1	1.41

Example2:

Input	Output
3 39 0 0 0 2 2 0	1.00

Time and memory limit: 1s/ 64MB

Solution and analysis

Since the percentage of the population inside the radius grows with the radius, the minimum radius which covers p percent of the area can be found with a binary search. The only thing that needs to be presented is how to compute the area of the intersection of the disk and the polygon.

If $p_1, \ldots, p_n$ are the vertices of a non-self-intersecting polygon P in a counter-clockwise order and $p_i = (x_i, y_i)$ for $i = 1, \ldots, n$, then, by the so-called shoelace formula, the area of the polygon is equal to

$$\sum_{i=1}^{n} \frac{x_i y_{i+1} - x_{i+1} y_i}{2}$$

where $(x_{n+1}, y_{n+1}) = (x_1, y_1)$. To understand how the area of the intersection of the polygon with the disk can be computed, we need to understand why the shoelace formula works.

ith term of the sum in the formula is half of the cross product of p_i and p_{i+1}, that is the signed area of a triangle T_i with vertices $(0,0)$, p_i, and p_{i+1}, which is the area of T_i if the vertices $(0,0), p_i, p_{i+1}$ are in the counter-clockwise order, or minus the area of T_i if the vertices are in clockwise order. Thus, the shoelace formula says that the signed area of P is the sum of the signed areas of the triangles $T_1, \ldots, T_n$.

To see why this is true, let us consider any region R of a plane, such that no ray $\{(tx_i, ty_i) : t \geq 0\}$ and no line segment $[(x_i, y_i), (x_{i+1}, y_{i+1})]$ intersects R. Then, when summing the signed areas of the triangles, if R is outside of the polygon, we add the area of R as many times as we subtract it, and if R is inside the polygon, then we add it one more time than we subtract it (see Figure 1 for an illustration).

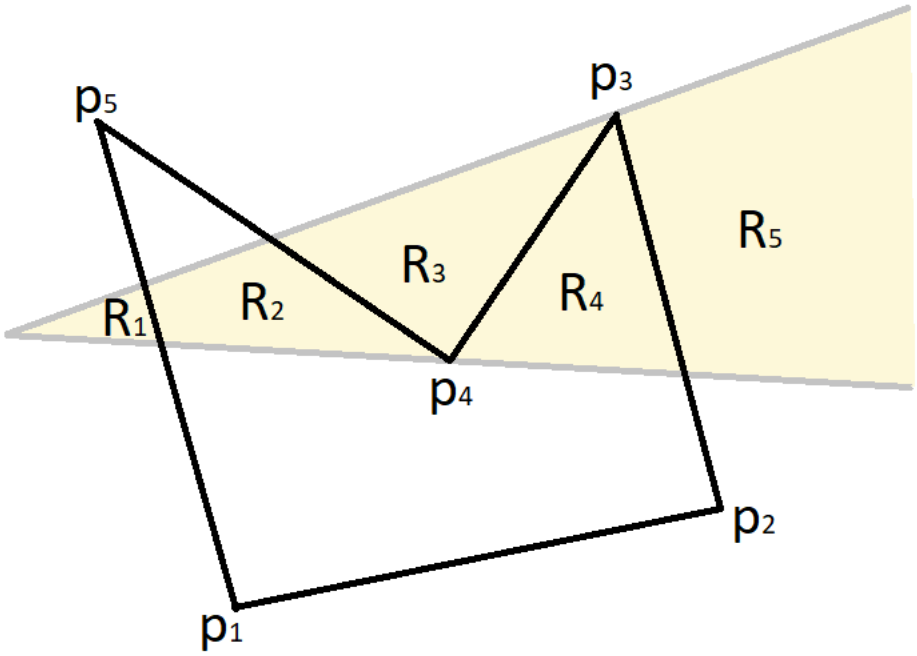


Figure 1: The area of the region R_1 is added twice in the shoelace formula (when adding the signed area of the triangles T_2 and T_4) and subtracted twice (when adding the signed area of the triangles

T_3 and T_5). The area of the region R_2 is added twice and subtracted once. The area of the region R_3 is added once and subtracted once. The area of the region R_4 is added once and never subtracted. The area of the region $R_5$is not added nor subtracted.

Now, we will describe how to calculate the area of the intersection of a polygon P with a $(0,0)$-origined disk with radius r. We denote the intersection by I. If no side of the polygon intersects the disk, then I is simply the disk, so the area can be easily calculated. From now on we assume that at least one side of P intersects the disk.

The boundary of I can be represented as a union of curves $S_1, \ldots, S_m$, where each S_i is either a line segment lying on a side of P, or a circular arc lying on the boundary of the disk. For any S_i, if the endpoints of S_i are collinear with $(0,0)$, then we define a_i and b_i to be the endpoints of S_i in any order. Otherwise we define a_i and b_i to be the endpoints of S_i in such an order, that when moving from a_i to b_i along S_i, we traverse the boundary of (a component of) I in a counter-clockwise direction.

For $i = 1, \ldots, m$, if S_i is a line segment, we define v_i to be the signed area of the triangle with the vertices $(0,0)$, a_i and b_i. If S_i is a circular arc, we define v_i as the (unsigned) area of the circular sector enclosed by S_i and segments from $(0,0)$ to a_i and from $(0,0)$ to b_i. Recall that the area of a circular sector with radius r and angle θ is $\frac{r^2\theta}{2}$.

We claim that the area of I is $\sum_{i=1}^{m} v_i$. Note that this is a generalization of the shoelace formula (these two formulas are identical for sufficiently large r). The same reasoning as in the case of the shoelace formula, but for a region R which does not intersect any ray $\{ta_i : t \geq 0\}$ or $\{tb_i : t \geq 0\}$ and none of the curves S_i, proves the formula for the area of I. The reason why we take the unsigned area of the circular sectors is because if S_i is an arc and we move from a_i to b_i along S_i, then we move in counter-clockwise with respect to the origin $(0,0)$. This is not necessarily the case if S_i is a line segment.

Finally observe that the representation of the boundary of the I as a union of the curves $S_1, \ldots, S_m$ can be computed in $O(n \log n)$ time: First intersect every side of the polygon with the disk to obtain those of $S_1, \ldots, S_m$ that are line segments ($O(n)$ time). Then find all the intersections $c_1, \ldots, c_k$ of the sides of P with the boundary of the disk ($O(n)$ time), sort them in a counter-clockwise order ($O(n \log n)$ time), and determine which of the arcs from c_i to c_{i+1} are lying on the boundary of I by inspecting the slope of the sides of P incident to c_i or c_{i+1}.

Resource: Caribbean Online Judge

Solution by:

Name: Michal Seweryn
School: Jagiellonian University
E-mail: michalsew@gmail.com

Problem R1 03: FF's Divisors

Little Felix Fuente a.k.a. FF wants to solve his math assignment. His teacher has taught him that a number n has $d1, d2, d3, \ldots, di$ divisors (1 and itself included). If $p = d1 * d2 * d3 * \ldots * di$, help FF find the total number of divisors of p.

Input

Several lines but no more than 120. Each line of input contains an integer n $(1 \leq n \leq 107)$. The input ends with a line containing 0.

Output

For each input line output the number of divisors of p.

Examples

Input	Output
1 2 3 4 0	1 2 2 4

Time and memory limit: 1s/ 64MB

Solution and analysis:

The solution to the problem lies in finding the prime factors of the divisors di– the divisors of p. Once we do that, it's a problem of finding the best way to multiply all those divisors, and get the final result for the test case.

We start off by constructing a sieve of Eratosthenes up to the largest single divisor (which is 10^7). We use a modified version of the popular algorithm, where we also store the primes in a separate array, as well as keeping an array which links the actual number to the index in the prime array. Once we get a number from the input, we first check whether this number is a square, because we will need to process its root separately. We iterate through the numbers smaller than the root of this number, and process each divisor, as well as the result of the division.

Processing the divisors is the main part of the solution. Firstly, we check if the number is a prime. If it is a prime, we add it to the final list of prime factors. Otherwise we find its prime factors using our sieved array, and add each prime to the final list. We do this in a particular way. Every time we encounter a prime divisor, we check whether we've added it to the list, if we haven't, we add it to the list. Once it's added to the list, we just increase its counter by 1 every time we encounter it. We check

whether it was added by looking it up in an array of added factors, it's the size of the primes array, and it's directly proportional to it.

Once we've processed all the divisors, we just iterate through the prime factor array, multiply them, and we get our solution. While doing this, we also reset the arrays for the next test case.

Resource: Caribbean Online Judge
Solution by:
Name: Mihajlo Kocic
School: Faculty of Science and Mathematics, University of Nis
E-mail: kocicmihajlo@gmail.com

Problem R1 04: Game with Psycho

This is an interesting game. There are n psychos standing in the line. Each psycho is assigned a unique integer from 1 to n. At each step, every psycho who has an id greater than the psycho to his right (if exists) kills his neighbor to the right in the line. Note that a psycho might kill and get killed at the same step.

You're given the initial arrangement of the psychos in the line. Calculate how many steps are needed to the moment of time such that nobody kills his neighbor after that moment. Look notes to understand the statement more precisely.

Input

The first line of the input contains an integer T ($1 \leq T \leq 30$), meaning the number of the test cases. For each test case: The first line of input contains integer n denoting the number of psychos, ($1 \leq n \leq 105$). In the second line, there will be a list of n distinct integers separated by a single space, each one in then range from 1 to n, inclusive, indicating the IDs of the psychos in the line from left to right.

Output

Print the number of steps, so that the line remains the same afterwards.

Examples

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

Explanation

Note: In the sample line of the psychos transforms as follows:
$[10\ 9\ 7\ 8\ 6\ 5\ 3\ 4\ 2\ 1] \rightarrow [10\ 8\ 4] \rightarrow [10]$. So, there are two steps.

Time and memory limit: 2s/ 128MB

Solution and analysis

The most important observation here is that the time of death of a chosen psycho is entirely independent of what's on his right side. Let's denote $t[i]$ as the round when the psycho on the i^{th} position is alive, or -1 if he never dies. If we can compute the entire array t, our answer is the maximum of these values plus one, as this is the last round when some of these psychos die.

Obviously $t[1] = -1$, as he is the leftmost one. Now for given index $i > 1$ we want to deduce $t[i]$ from previous values, that is $t[1], t[2], \ldots, t[i-1]$.

Let's call a psycho shorter if he has lower identifier number, and taller if otherwise.

If i^{th} psycho is the tallest amongst the first i psychos $t[i] = -1$. Otherwise, there exists a psycho on the left that is taller than him. Let's take the rightmost one and denote it's index j. If there are no psychos between j and i, that means that i^{th} psycho dies on the first round, and $t[i] = 0$. Otherwise there are some psychos between j and i. As these psychos are shorter than i^{th} psycho, all of them must be removed before i^{th} dies.

Let's look at the moment when already every psycho between i and j have died. At that moment, the psycho on the left of i^{th} must be taller or equal to j^{th}. So in the next round i^{th} psycho dies.

To conclude, formula for $t[i]$ is as follows

$$t[i] = \begin{cases} -1, & no\ higher\ psycho\ on\ the\ left \\ 1 + \max(-1, t[j+1], \dots, t[i-1]), & j\ is\ the\ index\ of\ a\ higher\ psycho \end{cases}$$

This can be computed by using interval tree, which in total yields complexity $O(n * \log(n))$, but we can come up with $O(n)$ algorithm using stack.

We would like to keep only interesting psychos, that is, psychos who can influence the results on the right. It turns out that stack of psychos of increasing height (so the shortest psycho is on top) is enough.

Initially the stack is empty. When a new psycho comes, we store the maximum time of the shorter psychos on the left. We pop shorter psychos until the stack is empty, or there is a taller psycho on the top. If the stack is empty, the answer is -1, otherwise it's 1 + the maximum of the death times of encountered shorter psychos we popped from the stack. At the end, we push the new psycho on the stack. Do notice that information from the popped psychos won't affect any future result, for 2 reasons:

- None of these psychos can be the rightmost taller psycho, as the new psycho is taller
- None of these psychos can have the maximum time of psycho in between, as the new psycho has higher time.

Each psycho is popped and pushed at most once, so complexity of such algorithm is $O(n)$.

Resource: Caribbean Online Judge
Solution by:

Name: Juliusz Straszyński
School: University of Warsaw
E-mail: julek@straszynski.pl

Problem R1 05: Homework with Fibonacci

Rodrigo Díaz de Vivar (1043 - July 10, 1099), known as Cid Campeador ("The lord-master of military arts"), was a Castilian nobleman, a military leader, a ferocious warrior, and a diplomat that fought against the Moors. El Cid's legendary martial abilities have fueled his reputation as an outstanding battlefield commander and one of the greatest heroes in Spain's history.

Before becoming a great warrior, El Cid was a mathematics and puzzles lover in his childhood. But one time, he got stuck with the following problem:

You are given a sequence of integers $a1, a2, \ldots, an$. Your task is to perform over the sequence m consecutive operations of the following type:

1. For given numbers li and ri you've got to calculate $\sum f(ak) : k = l_i..r_i$, where $f(0) = f(1) = 1$ and $f(i) = f(i-1) + f(i-2) : i \geq 2$.
2. For a group of three numbers l_i, r_i, d_i you should increase value a_x by d_i for all x, $(l_i, \leq x \leq r_i)$

Input

The first line contains the number of test cases (at most 15). For each test case: The first line contains one integer n $(1 \leq n \leq 105)$ - the number of integers in the sequence. The second line contains n integers $a1, a2, \ldots, an$ $(1 \leq a_i \leq 107)$, separated by spaces. The third line contains one integer m $(1 \leq m \leq 105)$ - the number of operations. Then follow m lines that describe the operations. Each line starts with an integer $t_i (1 \leq t_i \leq 2)$ indicating the operation type:

- If $t_i = 1$, then next follow two integers $l_i$and $r_i (1 \leq li \leq r_i \leq n)$, separated by spaces.
- If $t_i = 2$, then next follow three integers l_i, $r_i$and di $(1 \leq l_i \leq r_i \leq n, 0 \leq d_i \leq 100)$, separated by spaces.

Output

For each query where $t_i = 1$, print the calculated sum modulo 1000000007 $(10^9 + 7)$.

Examples

Input	Output
1 5 3 1 2 5 6 3 1 1 3 2 2 4 2 1 2 5	6 42

Time and memory limit: 10s/ 128MB

Solution and analysis:

Before we go to the solution of this problem, let us consider what happens with the value $f(x)$ if we increase x by d.

If d is equal to 1, then by the definition of Fibonacci numbers: $f(x+1)=f(x)+f(x-1)$. If d is larger than 1 following equation is correct: $f(x+d)=f(x)*f(d-1)+f(x-1)*f(d)$.This equation can be proven by induction, with base being $d=1$. Let us notice that if we have sum of some Fibonacci values $\sum f(a_i)$ and given d following equation is correct:

$$\sum f(a_i+d)=\sum(f(d-1)*f(a_i)+f(d)*f(a_i-1))=f(d-1)*\sum f(a_i)+f(d)*\sum f(a_i-1)$$

Now let us consider two segment trees. One keeps sum of values $f(x)$ in segment, and other one, sum of values $f(x-1)$ in segment. When processing question queries, we output sum of values of the first segment tree in given segment. When processing update queries we calculate new values (with the above given formula) for nodes in given segment in the both segment trees. In order to speed up the performance of this algorithm we will use lazy propagation.

Resource: Caribbean Online Judge
Solution by:

Name: Aleksa Milisavljević
School: Matematicka gimnazija
E-mail: aleksamilis@gmail.com

Problem R1 06: I am a Good Sublist

Ted has been studying numeric lists (lists of positive integer numbers not necessarily different and not necessarily in order). He has learned that a good sublist of a given list L is such sublist whose sum of elements is no greater than a given number M. A maximal good sublist is a good sublist S of L such that for every element x of L not in S if x is added to S it becomes bad. Ted wants to know how many maximal good sublists are there of a given list. Can you help him?

Note: A sublist S of the list L is a list obtained by removing some (maybe none or all) of the elements of the list L.

Input

First line of input contains integers N and M separated by spaces. Second line of input contains N integers li, $1 \leq i \leq N$ the elements of the list separated by spaces. It is granted that

$1 \leq N \leq 1000$, $1 \leq M \leq 10000$ and $1 \leq l_i \leq 1000$.

Output

Output in a single line the number of maximal good sublists modulo 1000000007 $(10^9 + 7)$.

Examples

Input	Output
3 10 5 3 6	2

Time and memory limit: 1s/ 64MB

Solution:

Let us sort the input list, and denote it as L. We will consider sublists S of the list L. Let us observe that for a sublist S to be good, it must hold:

$$sum(S) \leq M$$

$$sum(S) \geq M - min(L \backslash S) + 1$$

where $sum()$ denotes the sum of the elements of a sublist, $min()$ the smallest element in a sublist, and \ is set difference.

Let us fix index i, and assume the i-th element in the (sorted) list L is the first element that will not be contained in the sublist S. For each i, we will calculate the number of good sublists S with this property. Summing up those quantities will give the answer for our problem.

Let us denote the i-th element of L as x. From the definition of i it follows that elements $1, \ldots, i-1$ must be contained in S, let their sum be t. We see that the number of ways to suitably choose the remaining elements of S is exactly the number of ways to choose a sublist of elements with indications $i+1, i+2, \ldots, N$ summing up to at least $M-t-x+1$, and at most $M-t$. Note that some care must be exercised when the bounds on the sum of S are negative.

It will be enough if we are able to answer queries of the form: how many sublists of some suffix of L exist, such that they sum up to at most V (for some V). This, in turn, is a standard dynamic programming exercise: for each possible suffix of L, and sum V, we keep the number of sublists of the considered suffix that sum up to at most V. A single information of this kind can be computed in a constant time, given the information that was found previously. Note that the range of values for V is small, because we are only interested in $V \leq M$.

The dynamic programming described above runs in time $O(NM)$. Having computed all the cases, the remaining part of the solution works in $O(N)$. This gives a total running time of $O(NM)$, which is enough to pass all test cases.

Resource: Caribbean Online Judge
Solution by:
Name: Krzysztof Maziarz
School: Jagiellonian University
E-mail: krzysztof.s.maziarz@gmail.com

Problem R1 07: Legend of Heta

Heta is a conventional name for the historical Greek alphabet letter Eta (H) and several of its variants, when used in their original function of denoting the consonant /h/ (Wikipedia). Because of whispers of N'Zoth the corruptor, Heta now wants to destroy all existing alphabets (fiction).

Heta has a spell book containing spells to delete a string. There are several spells in that book. Armed with the spellbook, Heta starts his journey to complete his mission. On his way, Heta found a very long string. To delete that string, Heta reads that string letter by letter from the first letter. If at any point he found a substring that's present in his spellbook, that spell will be cast, and that substring will be destroyed. Then he continues until he reaches the end of that string and no more spells can be used. If there are multiple spells that can be used at one time, the spell that appears first in the book is used. Determine what's left of the very long string after Heta is done!

Input

First line is a string containing A-Z. String can contain from 1 to 100000 characters. Next line of input is N, the number of spells in the spellbook $1 \leq N \leq 100$. Next N lines contain spells sorted by appearance in the spellbook. Each spell is a string containing A-Z with length from 1 to 100.

Output

One line containing the string after Heta is done doing his magic.

Examples

Example1:

Input	Output
KUKUKAKIKUKAKEKKUKAKAKKUKAKUKAKU 2 KEK UK	KAKIKAKAKAKKAKAKU

Example2:

Input	Output
HEATHLEDGER 2 HEATH LEDGER	

Example3:

Input	Output
KAPKAPPAPAK 1 KAPPA	K

Example4:

Input	Output
CABAI 3 ABA AB B	CAI

Time and memory limit: 1s/ 128MB

Solution:

The idea of the solution is to implement efficiently Heta's algorithm - the program goes over the initial string $\boldsymbol{S}$, checks at each position $\boldsymbol{i}$ whether a spell ends there and if so erases its corresponding substring.

To erase the substrings efficiently we suggest using a separate stack-like data structure $\boldsymbol{S}'$ where the currently processed part of $\boldsymbol{S}$ is stored. When the program iterates to the next character of S, it is pushed to S'. When a spell ending at i is found, the characters which form it are popped one by one - it isn't an issue to do so, since each character of S is popped at most once, requiring $O(N)$ operations in total. I recommend using *std::vector* instead of *std::stack* because it also supports $O(1)$ random access - this is important for the next part of the solution.

Since the maximal length of the spells (let's denote it with $\boldsymbol{MaxSpell}$) is relatively short (*100*), for a given end index i, it isn't a problem to iterate over all possible starting positions for a spell $\boldsymbol{j}$ (let $\boldsymbol{S'_{j,i}} = \boldsymbol{S'_j} \ldots \boldsymbol{S'_i}$) and check whether $S'_{j,i}$ is a spell. For each position, there are at most $\boldsymbol{MaxSpell}$ starting indexes, $|S| \times MaxSpell$ substrings to be checked in total. By comparing them naively, we get an $O(|S| \times MaxSpell \times N)$ algorithm. This solution could be improved by comparing the substrings using *hashes* instead.

Firstly, to be able to compute the hashes of substrings of S' efficiently, a rolling hash function is required, e.g.:

$$H(a) = (a_1 p^0 + a_2 p^1 + a_3 p^2 + \ldots + a_k p^{k-1}) \bmod M$$

where p and $M$are prime numbers, $p \approx 100$, $M \approx 10^9$ and $k = |a|$. For a fixed end index i and $j = i$ we have $H(S'_{j,i}) = S_i$ and for every other $j < i$ we have

$$H(S'_{j,i}) = (S'_j + p \times H(S'_{(j+1),i})) \bmod M$$

This way each $H(S'_{j,i})$ can be calculated in $O(1)$.

Secondly, we need a fast way to determine which spell (if any) corresponds to a specific hash. This could be done by storing the index of the spell with a given hash in an *unordered_map*, which supports

$O(1)$ access on average. Then for every end position i we compute the values of $H(S'_{j,i})$ and determine which spells (if any) they correspond to. We take the one with the smallest index (let its length be k) and we pop the last k characters of S', thereby erasing the spell. Of course, if no spell is found ending at i the algorithm continues without deleting anything.

We process each substring in $O(1)$ and have a combined complexity of $O(|S| \times MaxSpell)$. There is another approach which doesn't use hashes and does $MaxSpell$ concurrent traversals of a *Trie* with all spells instead. It is, however, more complex and for this reason isn't covered here.

Resource: Sphere Online Judge
Solution by:
Name: Alexander Crustev
School: High School of Mathematics, Varna
E-mail: aleks.tcr@gmail.com

Problem R1 08: Maximum Child Sum

A rooted tree is being built. Initially, there is only one node in the tree, having number 1 and value 0. You are to perform Q ($Q \leq 200000$) queries, each of them is one of the following two types:

- 1 X Y - Add a new vertex to the tree with parent X (It's guaranteed that node X is already in the tree) and value Y ($1 \leq Y \leq 109$). Its number will be the smallest positive integer that is not a number of a node yet. For example, the first query of this type will add a vertex with number 2, then 3, then 4 and so on.
- 2 X - Consider the children of node X. For each of them, let's sum up the values of all nodes in their subtrees. You are asked to print the maximum of those sums.

Input

The first line contains an integer Q - the number of queries. Each of the next Q lines contains one of the queries.

Output

Print the answers for all queries of the second type, one answer per line.

Examples

Input	Output
7 1 1 3 2 1 2 2 1 2 5 2 1 1 1 4 2 1	3 0 8 8

Time and memory limit: 2s/ 128MB

Solution:

Firstly, we can input the entire tree and set the initial values of the nodes to zero. We can do this since the value of zero doesn't affect the queries. This way, we have transformed the problem of adding nodes to the tree to the problem of setting values of certain nodes.

Let's run a *dfs* from the root (node 1) and list the nodes in the order they are traversed (this permutation is called Euler walk). It can be easily seen that a subtree of a node is represented as a segment in the permutation. The beginning and the end of the segment can be obtained using the discovery and finish times of that particular node. Therefore, the sum of node's values in the subtree

is equal to the sum of the values on the segment so we can use a segment tree to add nodes and query the sum in a subtree in $O(log\ n)$ time complexity, where n stands for the number of nodes.

We can also add a node in $O(log\ n)$ time complexity, but it takes us $O(number\_of\_children * log\ n)$ time complexity to answer a query, which gives an overall complexity of $O(n * q * log\ n)$, which is too slow.

Let $K = 400$. Queries for nodes with at most K children will be answered in the previously described manner, which is acceptable for the given time limit.

Queries for nodes with more than K children will be answered in $O(1)$. To achieve this, when adding a node in the tree we need to traverse all the nodes with more than K children between the added node and the root of the tree. Let X be the node that is being added, Y_k the k-th node on the path from X to the root which has more than K children, and let Z_k be the child of node Y_k which is on the path from the X to Y_k (notice that Z_k is uniquely determined). We need to add the value of X to the sum of the subtree of Z_k and, if necessary, update the solution for Y_k to be the sum of the nodes in the subtree of Z_k. This makes the complexity of adding a node to be $O(log\ n + n / K)$. The overall complexity is good enough for this task.

Resource: Sphere Online Judge
Solution by:
Name: Stjepan Požgaj
School: Faculty of Science and Mathematics, University of Croatia
E-mail: stjepan.pozgaj1@gmail.com

Problem R1 09: On Time

Classes begin at 8:00 am at the University of Cienfuegos (UCf). Bob doesn't want to be late, but he doesn't want to wake too early either. There are several ways by which he could get to the UCf from his home: he could walk, or take a bus, or a combination of both.

Using a map of the city of Cienfuegos, Bob has identified N locations he could visit on his way to the UCf including the latter and his home, some of which are bus stops. He named all locations with unique integer numbers from 1 to N; by default, he named his home as 1 and the University as N. Bob can walk from location a to location b if there is a street connecting them. To measure the length of every street Bob used the time (in seconds) it takes to walk through it.

Every bus serving in the city has a route, a starting time S and a periodicity T. A bus route can be represented as a sequence of unique locations on Bob's map: $c_1, c_2, \ldots, c_R$; $c_i != c_j$ for $i != j$. Every location in this description is a bus stop. The bus takes 1 second to move from c_i to $c_i + 1$. At time S the bus stops in the location c_1, after 1 second traveling stops in c_2, and so on until it reaches the last stop in c_R. Also, every T seconds (counting from time S) there is a bus in c_1 starting the same operation. Note that a bus can start the route at $S + T * k$ (in location c_1, k as a Natural Number), while another bus from the same route haven't yet reached c_R.

Bob have gathered all the information necessary to find out at what time (with the accuracy of a second) he can leave home to arrive to the UCf before classes begin.

Input

A line with the numbers N ($2 \leq N \leq 104$, locations in Bob's map), M ($0 \leq M \leq 105$, number of streets), B ($0 \leq B \leq 102$, number of bus routes) and P ($1 \leq P \leq 86399$, the time in seconds when classes begin).

Next M lines describing each street with the numbers ai, bi ($1 \leq ai, bi \leq N$),

li ($1 \leq li \leq 103$); meaning that Bob can walk from ai to bi, or from bi to ai, in li seconds. Next B lines describing a bus with the numbers $S, T, R, c_1, c_2, \ldots, c_R$

($0 \leq S, T \leq 86399, 2 \leq R \leq 10, 1 \leq c_i \leq N$); meaning that this bus route begins operating at time S, repeats every T seconds and goes through locations c_1, c_2, ... and finishes at c_R.

Output

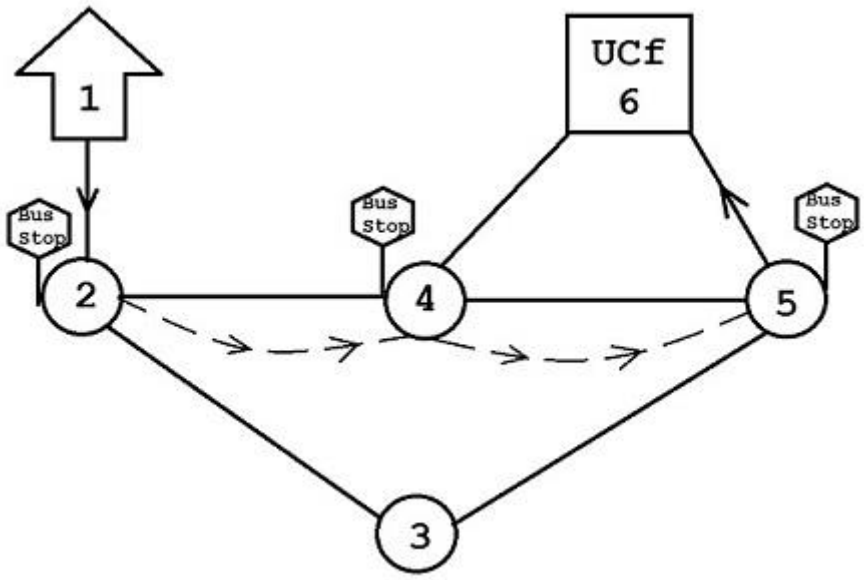
Print a single integer, the greatest time (in seconds) when Bob should leave home in order to arrive at the UCf before time P (when classes begin). If Bob needs to leave home the day before (negative time) just print "sleep at the UCf". You may assume that there is always a path from location 1 to N.

Examples

Input	Output
6 7 1 480 1 2 5 2 4 30 4 6 15 2 3 20 4 5 15 3 5 10 5 6 5 105 20 3 2 4 5	460

Explanation:

In the example, Bob leaves home at 460, reaches location 2 at 465 by walking, at 465 takes the bus from location 2 to location 4 and from location 4 to location 5, reaching 5 at 467, finally walks to the UCf (location 6) reaching there at 472. Classes begin at 480, so he got earlier.



Time and memory limit: 1s/ 64MB

Solution and analysis

The solution algorithm consists of 3 parts:

1. Constructing the graph.
2. Finding the quickest path from start to finish.

3. Searching for the best starting time.

Constructing the graph:

Each node in the graph will be depicted by X objects (X being the number of paths from that node to its neighbors), which consist of 4 properties:

- It's a neighbor.
- The time required to walk to the neighbor.
- The time at which the first bus arrives to that node (goes to the neighbor).
- The period at which every other bus arrives (goes to the neighbor).

The objects will be stored in the C++ vector container.

Finding the quickest path from start to finish:

The algorithm used for finding the path is *Dijsktra*, with some modifications, which takes the starting time as a parameter. First, we make a heap of objects consisting of the 2 properties, the current node N, and the total time required to get to it T, the whole heap will be sorted by a *least amount of time required*, which will make sure that the top object represents the node to which the path is the quickest. Then, iterating through the graph, we check if there is a bus line going through N and to it's neighbor. If there is, we check whether it is worth to wait for the next bus or not.

Yes:

We make a new object representing the next node whose T will be N's time + wait time + 1

No:

We make a new object representing the next node whose T will be N's time + travel time

Finally, we pop the previous and push the new object on to the heap and continue the same until the 'school' node is at the top of the heap, which means we have found the quickest path for the starting time. We empty the heap and check if we have made it to school in time, the function should return 1 if we did and 0 if we didn't.

Searching for the best starting time:

We initially call the function from time 0, to check if he can't make it to school at all, in which case the solution would be "sleep at the UCf". If that is not the case, using binary-search and calling the function, we find the largest time at which he can start and get to school.

Resource: Caribbean Online Judge
Solution by:
Name: Ilija Rackov
School: High School "Veljko Petrovic", Sombor
E-mail: ilajdzaa@gmail.com

Problem R1 10: Sauron's Army

After the last war of the five armies, Sauron wants to make his army more powerful. For that he has found an error that needs to be corrected, the soldiers are tired of seeing the same preceding soldier in the line. Then he needs to know in how many ways he can arrange each line so that no soldier sees the same preceding soldier from the last war. Given the line of N soldiers compute the amount of arrangements that meet the rules above.

Input

The only line of input contains an integer N $(1 \leq N \leq 500)$.

Output

Print a line with the amount of arrangements $mod\ 10^9 + 7$ (1000000007).

Examples

Input	Output
3	3

Explanation:

For the sample input the possible permutations are $[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]$. If the first permutation was used in the last war, then the possible permutations that meet the rules are $[1,3,2], [2,1,3], [3,2,1]$.

Time and memory limit: 1s/ 64MB

Solution and analysis

Without losing generality, we can assume that Sauron's army from the last war was the lexicographically smallest permutation, i.e., $[1,2,\ldots,N]$. Therefore, we are asked to compute the number of permutations with no rising successions. A permutation π is said to have no rising successions if $(\forall i)\ \pi(i+1) \neq \pi(i)+1$.

Let $f(n)$ be the number of permutations with no rising successions of length n. We shall prove the following recurrence:

$$f(0) = f(1) = 1$$

$$n \geq 3, f(n) = (n-1) \cdot f(n-1) + (n-2) \cdot f(n-2)$$

Obviously, we want to compute $f(N)$ which can easily be done in $O(N)$.

Proof: Note that we can build any permutation of length n by inserting a number n at the beginning, at the end or between any two elements of a permutation of length $(n-1)$.

Similarly, we can obtain any permutation with no rising successions by either:

1. Inserting $n$in a permutation with no rising successions of length $(n-1)$ such that it is not preceded by $(n-1)$.
2. Inserting n in a permutation with exactly one rising succession so that it breaks the rising succession.

Obviously, the former insertion (1) can be achieved in $(n-1)\cdot f(n-1)$ ways since we can insert the number n anywhere except immediately after $n-1$.

The only thing left to derive is the number of permutations of length n that have exactly one rising succession. We will construct those permutations from permutations without rising successions of length $n-1$.

Suppose we have fixed such permutation of length $(n-1)$. We can now build $(n-1)$ permutations of length n that have a single rising succession using the following algorithm:

```
p <- fixed permutation with no rising successions of length n-1
for k in [1,2,...,n-1]:
  p' <- p
  for i in [1,2,...,n-1]:
    if p'(i) >= k:
      p'(i) <- p'(i) + 1
  insert k in p' so that it precedes (k+1)
```

This process will yield all permutations of length n that have exactly one rising succession. Since the process is reversible, we can conclude that the total number of such permutations equals $(n-1)\cdot f(n-1)$.

Finally, the latter insertion (2) can be achieved in $(n-2)\cdot f(n-2)$ ways. Combining (1) and (2) we have proved our recurrence.

For a more general analysis of permutations with successions, you can check out this awesome blog post.

Resource: Caribbean Online Judge

Solution by:
Name: Ivan Paljak
School: University of Zagreb
E-mail: ipaljak@gmail.com

Problem R2 01: Akame

Akame is training on an infinite 2D Cartesian plane. There is an infinitely long, rigid and vertical string at each integer x-coordinate. She takes out her katana, Murasame, and makes N infinitely long slashes on the plane. Each slash can be seen as a line with the equation $Ax + By = C$, for some A , B , and C. A slash will never be a vertical line.

Each string Akame slashes will immediately be broken into two smaller strings. For example, a string originally from $(5, -\infty)$ to $(5, \infty)$ severed by the line $0x + 1y = 3$ will be broken into the two strings $(5, -\infty)$ to $(5,3)$ and $(5,3)$ to $(5, \infty)$. A string is considered disconnected if it is of a finite length. For example, the string from $(1,9)$ to $(1, \infty)$ is not of finite length, but the string from $(8, -3)$ to $(8,4)$ is.

There are M points on strings Akame wants to disconnect. A point is said to be disconnected if it is either slashed or the string it's on is disconnected. To measure her performance, Akame would like to know the earliest moment (that is, after which slash) each point is disconnected. The slashes are numbered from 1 to N .

Input

The first line of input will have N and M , separated by a single space.

The next N lines of input will each describe one of Akame's slashes, with A , B , and C separated by spaces.

The next M lines of input will each describe a point Akame wants to disconnect, with X_i and Y_i .

Output

There should be M lines of output. The i−th line should contain the smallest index of the slash that after it is performed, the i−th point is disconnected. If the i−th point is never disconnected, print -1 instead.

Constraints:

$$1 \leq N \leq 500000$$

$$1 \leq M \leq 500000$$

$$0 \leq A, |C| \leq 30000$$

$$1 \leq |B| \leq 30000$$

$$0 \leq |X\_i|, |Y\_i| \leq 10^9$$

Examples

Example1:

Input	Output
3 5 0 1 3 0 1 5 0 1 7 1 2 1 3 1 4 1 5 1 6	-1 1 2 2 3

Example2:

Input	Output
10 10 81 -36 72 64 -69 24 23 47 47 83 36 18 1 25 77 12 -81 -3 13 -90 -34 23 19 -34 19 23 78 50 -96 82 56 59 85 47 46 -23 1 13 -74 -69 23 99 -98 72 -31 -65 -78 7	2 3 4 -1 2 -1 4 2 4 -1

Time and memory limit: 10s/ 1024MB

Solution:

We are going to make a binary tree in the kind of way we make merge sort in $N * log(N)$

In that binary, each node will store interval from L to R inclusive of course.

What will be in that interval? Linear equations we get from adding them one by one, that being said, in that node we store both upper and lower envelope of linear equations added between moment L and R inclusive.

We build that binary tree like with merge sort method. Only in this method firstly we check which of the two lower nodes is the one with the bigger slope (in case of lower envelope and vice versa in case of upper envelope).

Now we need to find the minimum time for each of the points when it gets surrounded by envelopes.

Firstly, we send first slash to separate points in two groups, upper or lower. To upper, if the first slash is below our point, or lower, if the first slash is above our point.

We first push all of the points in the first node of the binary tree (which holds all slashes between the start and the end), if our point never gets a slash it needs (lower or upper according to need) in first node, then that point gets output -1, or so to say never gets cut off by definition of the task.

For the points which are still alive after that first check, we try to push all of them in the left node (which holds all slashes between $1\ldots\frac{n}{2}$, if a point gets cut off there, then her time of getting cut off is $\leq \frac{n}{2}$ and we push her to the left, or else we push our point to the right.

You actually do 2 of these tours, one for the upper and one for the lower group of points described before.

Overall complexity $O(N * log(N))$

Resource: DMOJ
Solution by:
Name: Kristijan Verović
School: Highschool Lucijana Vranjanina, Samobor
E-mail: kiki.verovic@gmail.com

Problem R2 02: Arrow

Bruce is playing the game Green Arrow. In the game, the ground is the x-axis and the targets are a number of vertical line segments in the first quadrant. There is no intersection between any two targets. Moreover, no target touches the x-axis. Bruce can control a game character, always located at $(0,0)$, to shoot an arrow towards the first quadrant with an angle θ ($0° < \theta < 90°$) .There is no air resistance in the game. Thus, the trajectory of the arrow will be a standard parabola. The targets which are crossed by the trajectory are hit in the game, including those with only endpoints crossed by the trajectory.

In challenge mode, there are a few rounds. In the first round, there is only one target. If Bruce hits the target, he can pass to the second round. In the second round, the target in the first round will appear and a new target will be added. Bruce must hit both targets, so that he can pass to the next round. Similarly, in the k-th round, the targets in the previous $k-1$ rounds will appear, and a new target is added. Bruce can pass to the next round only if he can make a shot to hit all targets in this round. Bruce has hacked the game and gets the locations of all targets. He wants to know the maximum number of rounds he can successfully pass. It is possible that Bruce can win the game, in which case there are no more rounds to pass after that.

Input

The first line of input will consist of one integer, N $(1 \leq N \leq 100\,000)$, the number of rounds in the game. Each of the following N lines will consist of three integers, x_i $(0 < x_i < 109)$, y_{i1} and y_{i2} $(0 < y_{i1} < y_{i2} < 109)$, which are the target i's horizontal position, the low endpoint, and the high endpoint, respectively.

Output

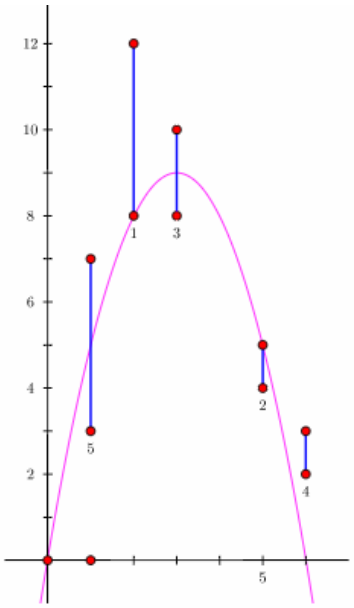
Output one integer, the maximum number of rounds Bruce can pass.

Examples

Input	Output
5 2 8 12 5 4 5 3 8 10 6 2 3 1 3 7	3

Explanation:

The sample case is as shown in the following figure:



Time and memory limit: 4s/ 256MB

Solution and analysis:

First of all, let's think what is an arrow in this problem. Every arrow is some parabola $y = Ax^2 + Bx + C$. And to be an arrow this parabola must satisfy that $A \leq 0$ and it passes through $(0,0)$. From the second condition, we have that $0 = 0 + 0 + C <=> C = 0$. So, every arrow is a parabola

$$y = Ax^2 + Bx.$$

Let's consider a segment $(x_s, y_1, y_2), y_1 < y_2$ and let's think when our parabola intersects this segment. Then, it needs to be satisfied that

$$Ax_s{}^2 + Bx_s \leq y_2 \ \& \ Ax_s{}^2 + Bx_s \geq y_1.$$

Let's consider our coefficients A and B as two unknown variables. Then, there is a parabola which intersects all the segments, if a linear programming on this two variables and corresponding inequalities for each segment has a solution. After that, we can just do a binary search to get an answer to the original problem.

But a linear programming is quite slow for constraints in our problem. But we have only two variables here, so LP can be solved efficiently because every inequality is a half-plane in R^2. So, we just need to check if a set of half-planes has nonempty intersection. It's a well-known problem and can be solved in $O(n\ log\ n)$ or $O(n)$ (yeah, a randomized algorithm with such expected complexity does exist). Or we can have a set with half-planes sorted by angle and maintain this set after inserting a new half-plane. Then our complexity would be $O(n\ log\ n)$.

Resource: DMOJ

Solution by:

Name: Vladyslav Hlembotskyi
School: Jagiellonian University, Khmelnytskyi
E-mail: nobikik9@gmail.com

Problem R2 03: Line Sweep

Preoccupied with coding, you've allowed your room to become quite dusty. You'd better sweep it before your parents find out!

Your room can be represented by a two-dimensional grid of cells, with N rows and M columns. Each cell either contains a piece of furniture, or is empty and must be swept. At least one cell is empty.

To get the job done, you'll be using a broom, of course - in particular, a linear broom. A linear broom of length X can cover a vertical line of X consecutive cells. One sweep of the broom consists of placing it down, and then moving it horizontally by any distance. However, at all times during a sweep, the broom must remain entirely within the boundaries of the room, and all X of the cells it covers must be empty. All of the cells that the broom moves over during a sweep are then rid of dust.

The bigger the better, so you'd like to purchase a broom of the largest size possible such that it's possible to sweep your entire room with it. Your room is completely swept once every empty cell has been involved in at least one sweep of the broom. After deciding on the size of your broom, you'd also like to minimize the number of sweeps required to get the job done.

Input

The first line of input contains three integers N, M and F, which are the number of rows, columns and pieces of furniture respectively ($1 \leq N \leq 2000$; $1 \leq M \leq 2000$; $1 \leq F < N * M$).

The next F lines contain values $r, c, (1 \leq r \leq N, 1 \leq c \leq M)$ which is the row and column position of furniture. No two pieces of furniture will be at the same coordinates.

Output

The output is composed of two numbers. The first line contains the integer width of the largest broom that can be purchased. The second line contains the number of sweeps required to sweep all empty cells.

Examples

Input	Output
5 7 2 3 4 5 7	2 4

Time and memory limit: 2s/ 256MB

Solution:

This problem can be solved easily by using greedy approach. Let's find the biggest broom width first. We initiate two arrays, $u[][]$ and $d[][]$.

$u[i][j]$ represents the distance to the first piece of furniture on the column above (i, j), and $d[i][j]$ is the distance to the first piece of furniture on the column below (i, j). The width of the broom (w) is the minimal value of $u[i][j] + d[i][j] - 1$ among all empty cells.

Next up is finding the number of swipes. Notice that the topmost cell of the broom (the tip) will not pass over the same cell twice and that it will have to pass through some specific cells. We initiate a new array $p[][]$, with $p[i][j] = 0$ for all (i, j).

If $p[i][j] == 1$, the tip of the broom will have to pass through (i, j). We will call a cell (i, j) invalid if it either contains a piece of furniture or lies outside the grid. For each empty cell (i, j), we check if the cell above is invalid. If it is, set the $p[i][j] = 1$.

If the cell below is invalid, set the $p[i - w + 1][j] = 1$.

Now we iterate through each row and for each cell (i, j) with $p[i][j] == 1$ we extend the tip of the broom to the right as far as we can, setting $p[i][j + 1] = 1$, $p[i][j + 2] = 1$, ..., for each free cell as long as all of the $w - 1$ cells below are also free. Then we extend the tip of the broom to the left for each row. After that, we swipe all the cells that we marked using the $p[][]$ array. The number of swipes is equal to the total number of continuous subsequences of 1s for each row of $p[][]$.

All of the remaining unswept cells form disjoint rectangular regions for which the number of swipes is easily calculated. Both time and memory complexities are $O(n * m)$.

Resource: DMOJ

Solution by:
Name: Nikola Herceg
School: Faculty of Science and Mathematics, University of Zagreb
E-mail: nikolaherceghr@gmail.com

Problem R2 04: Mine and Tree

In her spare time, Mine enjoys game shows and mathematics. In particular, she really likes geometry and graph theory. Today, she is a participant on the popular game show GeomeTree, where contestants show off their mathematical skills by performing operations on a point in the 2D coordinate plane, all while on a tree. Mine is in it to win it!

Each contestant is given an undirected graph with N $(1 \leq N \leq 105)$ vertices and $N-1$ edges such that there is a unique path between every pair of vertices - a tree. On each vertex of the tree, an operation is written on it. The operation may either be "rotate your point by θ $(0 \leq \theta < 360)$ degrees counterclockwise around the origin", "translate your point by a vector (dx, dy)

$(-105 \leq dx, dy \leq 105)$, or "move the point towards (but not past) another point (p, q)

$(-105 \leq p, q \leq 105)$ such that the new distance between these points is $P\%$ $(0 \leq P < 100)$ of the old distance". Contestants perform M $(1 \leq M \leq 105)$ rounds of queries on this tree. Each round, there are two possible queries the contestant needs to perform.

The first kind of query is when the contestant is given two vertices of the tree (u, v) $(1 \leq u, v \leq N)$ and a point (x, y) $(-105 \leq x, y \leq 105)$. The contestant has to move the point from vertex u of the tree to vertex v of the tree, performing every operation written on the vertices to the point they were given along the way (including the operations on vertices u and v). Operations are cumulative, and you should perform the operations sequentially.

The second kind of query is when the operation on a tree changes. The new operation will still be either a rotation, translation, or moving operation in accordance with the constraints above.

The first-place prize in the game show is unlimited riches along with a platinum pass to a love hotel for two. We know which prize Mine is really interested in. However, all the other contestants are writing programs to play this ridiculously difficult game show for them. Since Mine is in a huge pinch right now, she has hired you to help her win! You have no choice but to write a program that helps her, otherwise you will be blasted into pieces by Pumpkin!

Input

All the numbers in the input are integers.

The first line of input will have N and M, separated by a single space.

The next N lines will each have the operation of each vertex of the tree, in order from $1..N$.

The operation is one of the following:

- $R\ \theta$ - rotate by θ degrees operation
- $T\ dx\ dy$ - translate by (dx, dy) operation
- $M\ p\ q\ P$ - translate by (p, q) such that the new distance is $P\%$ of the original distance operation.

The next $N-1$ lines describe the graph. Each line is a pair of vertices which are connected by an edge in the graph. There will be no self-loops or duplicate edges, and it is guaranteed that the resulting graph is a tree.

The next M lines will describe the queries. If the line begins with:

- Q it will be followed by $u\ v\ x\ y$, indicating the first kind of query
- U it will be followed by u and the new operation on vertex u, in the same format as above, indicating the second kind of query

Output

For each Q query, output two space-separated real numbers, the final coordinates of the input point. Both numbers should be within an absolute or relative error of at most 10^{-6}.

Examples

Example1:

Input	Output
1 1 M 4 4 75 Q 1 1 0 0	1.000000 1.000000

Example2:

Input	Output
2 3 R 45 R 135 Q 1 1 100 100 Q 1 2 100 100 Q 2 2 100 100	0.000000 141.421356 -100.000000 -100.000000 -141.421356 0.000000

Example3:

Input	Output
5 8 R 90 T 1 1 R 45 T 3 2 R 0 1 2 2 3 3 4 4 5 Q 1 1 0 0 Q 3 3 -4 9 U 5 R 180 Q 1 5 1 1 Q 5 1 1 1 U 2 T 3 -1 Q 2 3 3 5 Q 3 1 13 37	0.000000 0.000000 -9.192388 3.535534 -1.585786 -3.414214 -3.121320 1.707107 1.414214 7.071068 -34.355339 -13.970563

Time and memory limit: 5s/ 256MB

Solution and analysis:

Long story short:

All three kinds of operations are affine angle-preserving plane transformations (we'll call such transformations nice), and a composition of nice transformations is also a nice transformation. This composition is also associative, so we can preprocess the tree using heavy-light decomposition, so we can easily find the composition of the transformations along some path in the tree.

Now for a more detailed solution:

Imagine a triangle on the 2D coordinate plane, with its vertices given in the counter-clockwise (CCW) order. If we rotate this triangle around the origin by θ degrees (operation 1), the result will also be a triangle with its vertices in CCW order, similar to the original one. Also, if we translate it (operation 2) or scale it with respect to some point (operation 3), the result will also be a triangle in CCW order, **similar** to the original. So, all three of our operations are nice.

Imagine we have a sequence $f = f_1, f_2, \dots, f_k$, $f\big((x,y)\big) = f_k\left(f_{k-1}\big(\dots f_1\big((x,y)\big)\dots\big)\right)$ of such nice transformations. To describe the composition of nice transformations, we can simply compute $f((0,0))$ and $f((1,0))$. These two points are sufficient to fully describe any nice transformation. To see why this is true, think of the triangle $(0,0),(1,0),(x,y)$. It will be mapped by f to $f\big((0,0)\big), f\big((1,0)\big), f\big((x,y)\big)$. If we know $f\big((0,0)\big), f\big((1,0)\big)$, we can reconstruct $f((x,y))$ because these two triangles are similar, and their vertices are in the same order.

To be able to answer the queries from the task, we need to preprocess the given tree by decomposing it into a set of vertex-disjoint paths, such that there are $O(log\ n)$ different paths on the path from any node to some arbitrarily chosen root. This is a well-known technique, and it's called heavy-light decomposition (HLD).

After finding this decomposition, we will process every path of the decomposition by building a segment tree, where each node is a nice transformation, and we will do this in both directions, both upwards (towards the root) and downwards (away from it). Now the path from any node u to any other node v consists of a path from u towards the root, stopping at some node w, and then away from the root towards v. This "turning point" w is called the lowest common ancestor of u and v and can also be found in logarithmic time using appropriate preprocessing.

To answer the first type of query, we find the resulting composition of nice transformations using HLD and segment tree techniques and apply the resulting transformation to the given point. To answer the second type of query we update the segment tree corresponding to the path which contains the given node.

The first query type works in $O(\log^2 n)$ and the update query works in $O(log\ n)$. Preprocessing can be done in $O(n)$. Time complexity is thus $O(n\ +\ q\log^2 n)$. Memory complexity is $O(n)$.

Resource: DMOJ
Solution by:
Name: Ivan Stošić
School: Faculty of Science, University of Nis
E-mail: ivan100sic@gmail.com

Problem R2 05: Primitive Pythagorean Pairs

Do you know the Pythagorean theorem? Definitely yes. Do you know about Pythagorean triples? Maybe not. Do you know primitive Pythagorean pairs? Definitely not, because Bruce has not defined it yet! Let's start from the Pythagorean triples. A Pythagorean triple consists of three positive integers a, b and c, such that $a^2 + b^2 = c^2$. If a and b are also coprime, the pair (a, b) is named a primitive Pythagorean pair, like $(3, 4)$. Bruce will tell you how to generate primitive Pythagorean pairs. Given two positive integers m and n $(m > n)$, if m and n are coprime and $m - n$ is odd, a primitive Pythagorean pair (a, b) can be calculated by $a = m^2 - n^2$ and $b = 2 * m * n$. There are an infinite number of primitive Pythagorean pairs.

Now Bruce has N cards, such of which has a positive integer $v_i (i = 1,2, \ldots, N)$. He wants to pick up a few cards so that there are no primitive Pythagorean pairs among the selected cards. Could you please help Bruce to get the total number of different selections? Two selections are different if and only if one card is in one selection, but is not in the other selection. Bruce has to select at least one card in each selection.

Input

The first line of input will consist of one integer, N $(1 \leq N \leq 1\,000\,000)$, the number of cards Bruce has. The second line of input will consist of N positive integers $v_1, v_2 .. v_n$, where v_i $(1 \leq v_i \leq 200\,000$ or $20\,000 \leq v_i \leq 1\,000\,000$ for all $i = 1..n)$ is the value of card i $(1 \leq i \leq N)$.

Output

Output one integer, the number of different selections modulo $10^9 + 7$.

Examples

Input	Output
4 5 12 35 5	8 2

Explanation

There are two primitive Pythagorean pairs $(5, 12)$ and $(12, 35)$. So, there are 8 different selections: $\{5\}, \{12\}, \{35\}, \{5\}, \{5,35\}, \{35,5\}, \{5,5\}, \{5, 35, 5\}$.

Time and memory limit: 2s/ 256MB

Solution and analysis

Let's define $G = (V, E)$, such that its set of vertices is $\{v_1, v_2, \ldots, v_n\}$ and two vertices a, b are connected with a bidirectional edge iff $\sqrt{a^2 + b^2} \in \mathbb{N}$ and $(a, b) = 1$. The problem asks for the number of independent vertex sets in this graph.

First, note that we need to extend our solution to work when $v_1, v_2, \ldots, v_n$ are not unique. For this we need to find the unique values $u_1, \ldots, u_k$ along with their multiplicities in v. If vertices $u_{i_1}, \ldots, u_{i_m}$ with multiplicities $b_{i_1}, \ldots, b_{i_m}$ form an independent set, if we count such sets with multiplicities, the number is

$$\left(2^{b_{i_1}} - 1\right)\left(2^{b_{i_2}} - 1\right) \ldots \left(2^{b_{i_m}} - 1\right).$$

Let's solve a simpler version of the problem: G is a tree.

Let r be an arbitrarily chosen root. For each vertex x, we will calculate the number of independent sets, considering only the subtree rooted at x, in two scenarios: if x is a part of such independent set ($d_{x,1}$) and if it is not ($d_{x,0}$). Let $y_1, \ldots, y_p$ be the children of vertex x. The recurrence relation is

$$d_{x,1} = \left(2^{mult(x)} - 1\right) \prod_{i=1}^{p} (d_{y_i,0})$$

$$d_{x,0} = \prod_{i=1}^{p} (d_{y_i,0} + d_{y_i,1})$$

where $mult(x)$ is the multiplicity of vertex x. The solution for the whole tree is given by $Sol(T) = d_{r,0} + d_{r,1}$. This solution works in linear time.

Let's make it a bit more complicated: G is a forest.

If $T_1, \ldots, T_q$ are all trees of the forest, the solution for the forest is $Sol(T_1) * Sol(T_2) * \ldots * Sol(T_q)$, as all the sets can be chosen independently for each tree.

Now to the actual problem. Our graph is neither a tree, nor a forest, nor a simple cycle, nor are we sure if it is connected or not. But, we can empirically verify that the difference between the number of edges and the number of vertices is not too large, in each connected component of any input graph. The constraints may give a hint: Either all v_i are no greater than 200000 or they are all between 20000 and 10^6.

For a general graph, we can find the solution for it by finding its connected components, solving them independently and multiplying these solutions. So, let's focus on finding the solution for a connected graph $G = (V, E)$.

Imagine we chose a few vertices V, removed them (let's call that set C), and that we are left with a forest. Let's call such a set of vertices a splitting set. For every subset D of C (there are exactly $2^{|C|}$ of them), let's find the number of independent sets in G such that all vertices in D appear in those independent sets and no vertex from $C \backslash D$ appears. This can be done by modifying the dynamic programming approach shown above: For each vertex u which is adjacent to some vertex in D, we force $d_{u,1} = 0$, as u cannot be chosen.

Since we can find the solution for every subset D of C in linear time in the number of vertices in G, the complexity is $O(|V| * 2^{|C|})$. We just need to find a sufficiently small splitting set.

As it turns out, the following algorithm does a good job. It finds a small splitting set of the given connected graph:

If there is a vertex with degree 0 or 1, remove it from the graph (**do not** add it to the splitting set S)

Otherwise, pick a vertex with the highest degree, remove it from the graph and add it to S.

If there are no vertices left, stop.

To see why S is a splitting set, add the removed vertices (those not in S) in reverse order. Each added vertex will have degree 0 or 1 and so a cycle cannot be formed.

In practice, this always finds a splitting set of size at most 2 when run on the connected components of graphs induced on $V = \{1, \ldots, 200000\}$ and $V = \{20000, \ldots, 10^6\}$.

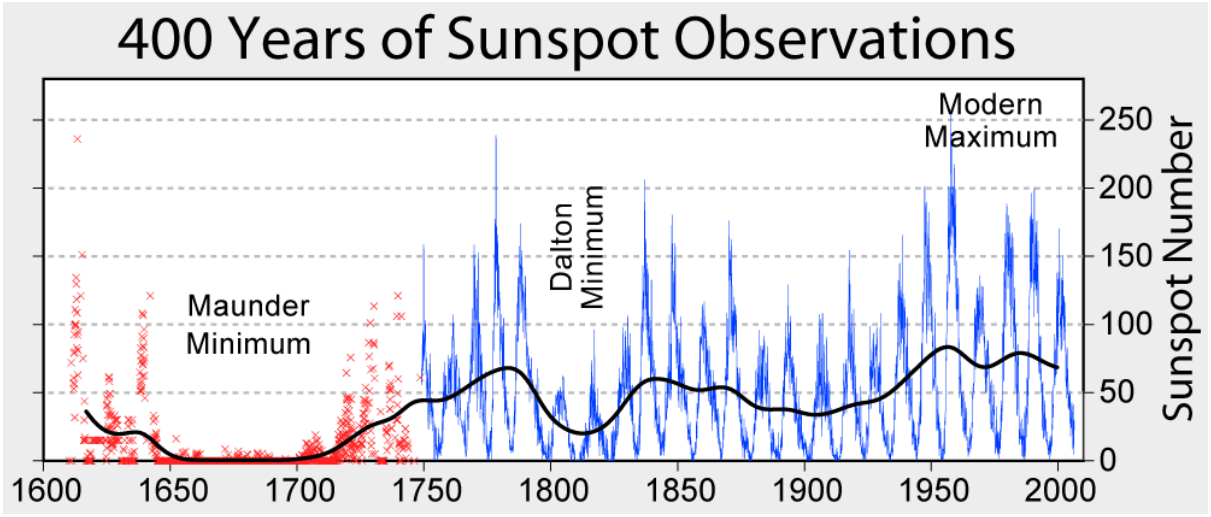
To further strengthen our solution, we can always use these splitting sets on our input graph, and not the ones we would find if we ran the algorithm on the input graph, because there is a possibility that our algorithm would perform poorly (i.e. produce a large splitting set) for some specially constructed input graph.

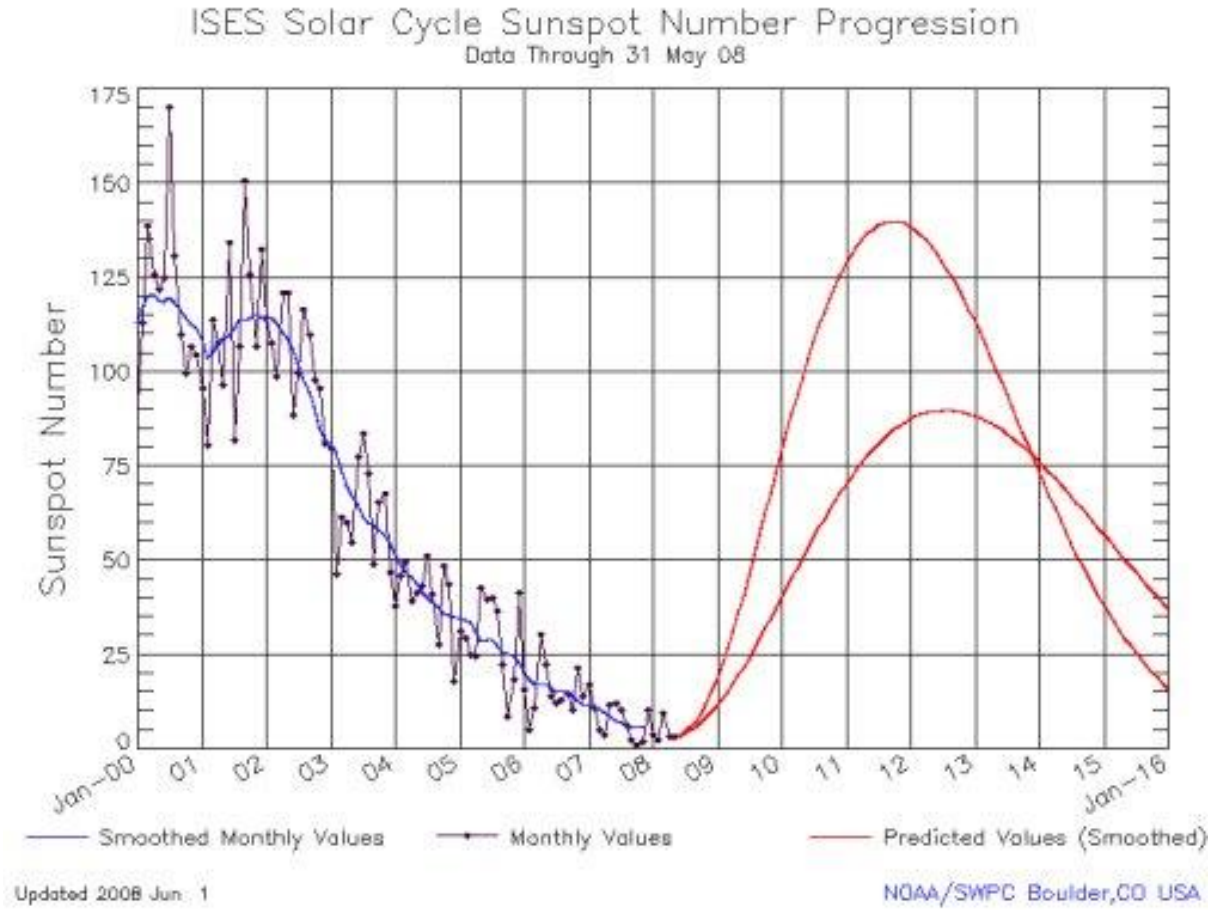
Time complexity: $O(N * 2^{O(1)})$. Memory complexity: $O(N * O(1))$. The constant factors come from the properties of input ($v_i \leq 200000$ or $20000 \leq v_i \leq 10^6$). In general, the time complexity is exponential.

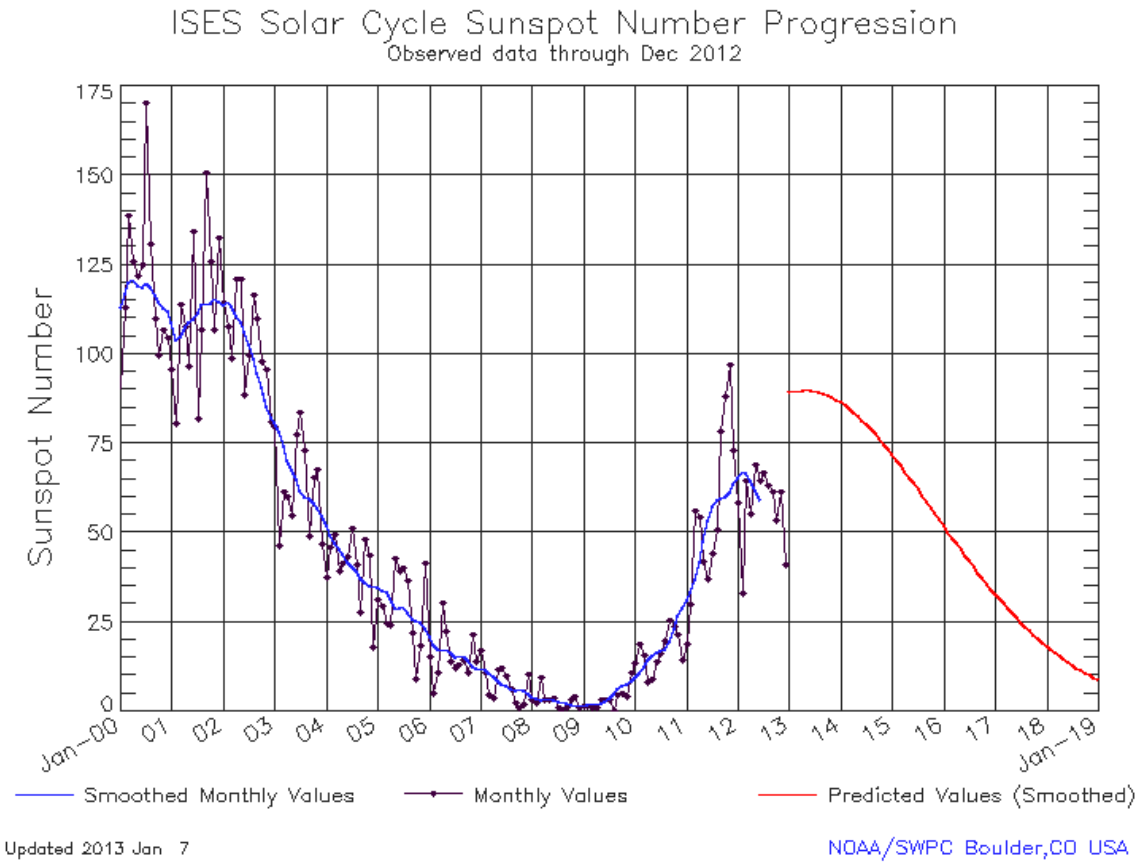
Resource: DMOJ
Solution by:
Name: Ivan Stošić
School: Faculty of Science, University of Niš
E-mail: ivan100sic@gmail.com

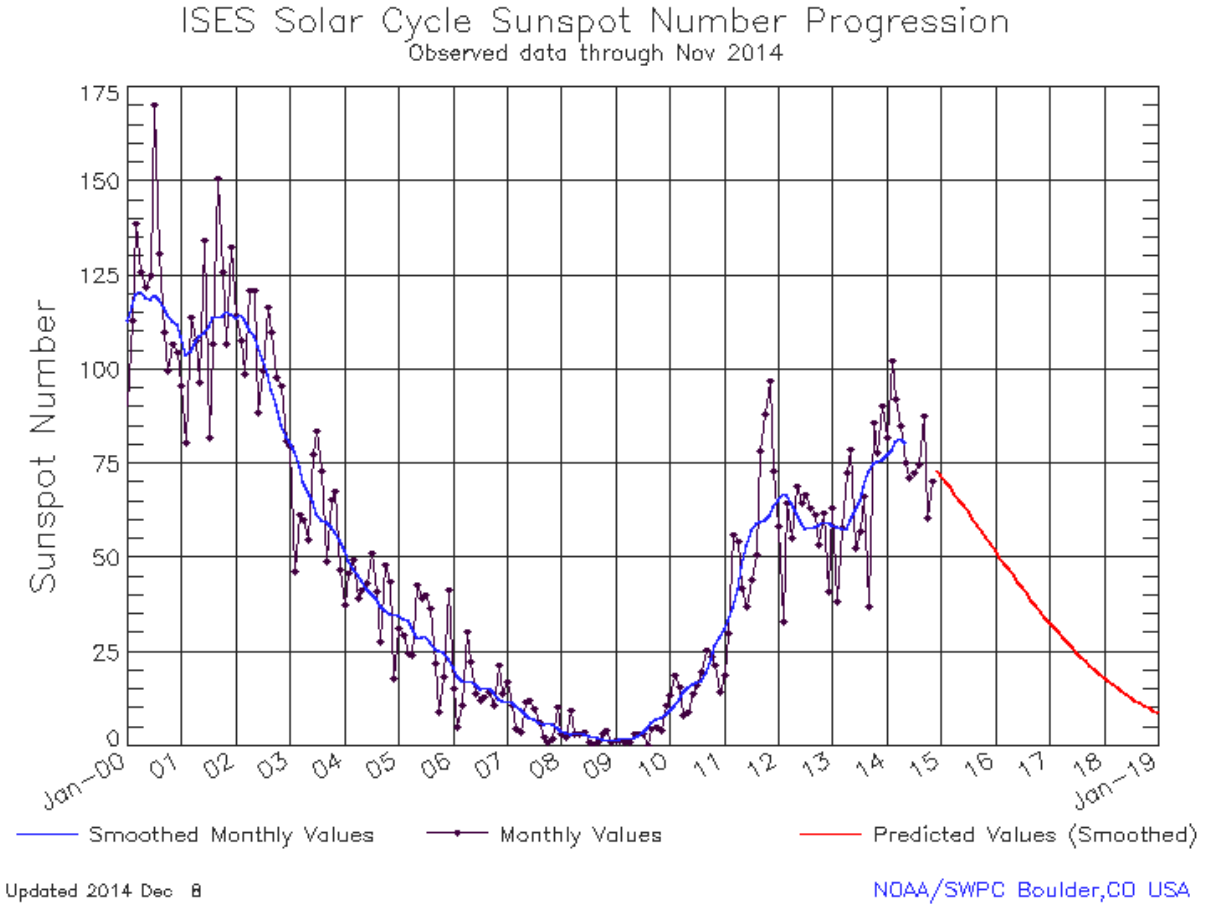
Problem R2 06: Periodic Function

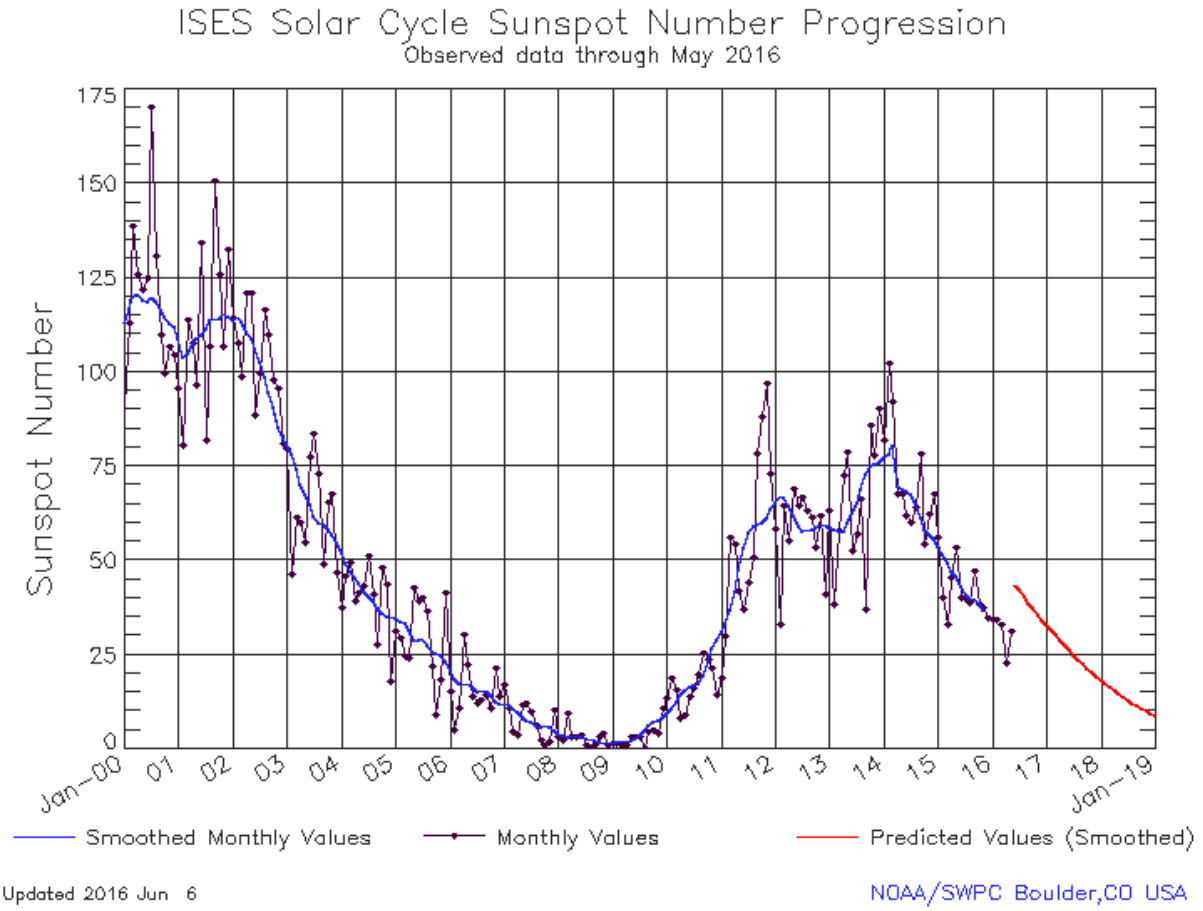
Solar cycle predictions are used by various agencies and many industry groups. The solar cycle is important for determining the lifetime of satellites in low-Earth orbit, as the drag on the satellites correlates with the solar cycle [...].(NOAA)






ISES Solar Cycle Sunspot Number Progression
Observed data through Dec 2012
Sunspot Number
175
150
125
100
75
50
25
0
Jan-00
01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
Jan-19
Smoothed Monthly Values
Monthly Values
Predicted Values (Smoothed)
Updated 2013 Jan 7
NOAA/SWPC Boulder,CO USA


ISES Solar Cycle Sunspot Number Progression
Observed data through Nov 2014
Sunspot Number
175
150
125
100
75
50
25
0
Jan-00
01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
Jan-19
Smoothed Monthly Values
Monthly Values
Predicted Values (Smoothed)
Updated 2014 Dec 8
NOAA/SWPC Boulder,CO USA



Sunspot Number Progression: Observed data through May 2008; Dec 2012; Nov 2014; Jun 2016.

The goal of the problem is to propose a perfect prediction center, with not so weak constraints.

Let us consider periodic functions from Z to R.

```
def f(x): return [4, -6, 7][x%3] # (with Python notations)
# 4, -6, 7, 4, -6, 7, 4, -6, 7, 4, -6, 7, 4, -6, 7, ...
```

For example, f is a 3-periodic function, with $f(0) = f(3) = f(6) = f(9) = 4$.

With a simplified notation, we will denote f as $[4, -6, 7]$.

For two periodic functions (with integral period), the quotient of periods will be rational, in that case it can be shown that the sum of the functions is also a periodic function. Thus, the set of all such functions is a vector space over R.

For that problem, we consider a function that is the sum of several periodic functions all with as period an integer N at the maximum. You will be given some starting values, and you'll have to find new ones.

Input

On the first line, you will be given an integer N.

On the second line, you will be given integers y : the first (0-indexed) $N \times N$ values of a periodic function f that is sum of periodic functions all with as period an integer N at the maximum.

On the third line, you will be given $N \times N$ integers x.

Output

Print $f(x)$ for all required x. See sample for details.

Examples

Input
3 15 3 17 2 16 4 15 3 17 10 100 1000 10000 100000 1000000 10000000 100000000 1000000000

Output
16 16 16 16 16 16 16 16 16

Explanation

For example, f can be seen as the sum of three periodic functions: $[10] + [5, -8] + [0, 1, 2]$ (with simplified notations; periods are 1,2 and 3). In that case $f(10) = [10][10\%1] + [5, -8][10\%2] + [0, 1, 2][10\%3] = 10 + 5 + 1 = 16$, and so on.

Constraints

$$N < 258$$

$$abs(y) < 10^9$$

$$0 \leq x < 10^9$$

Information

You can safely assume output fit in a signed 32bit container. There are 6 input files, with increasing value of N. Some details $(\#i, N)$:

$$(\#0, around\ 50)$$

$$(\#1, around\ 50)$$

$$(\#2, around\ 100)$$

$$(\#3, around\ 150)$$

$$(\#4, around\ 200)$$

$$(\#5, around\ 250)$$

Time and memory limit: 3s/ 256MB

Solution:

A lot of problems involve experimentation and often, when a solution is found, even when it is simple, it's hard to show its correctness. This problem is no exception.

First, let's rephrase the problem in terms of linear algebra. Let n be fixed throughout the solution. We are given a vector a of length n^2 as input and n^2 queries, asking us for the value of $y = f(x)$ for some integer x as described in the problem statement. The vector is such that it can be represented as a linear combination of vectors of the form $e_{k,j}$, where $e_{k,j}[i] = 1$ if $i \equiv_k j$ and 0 otherwise. It can be shown that every valid starting vector uniquely determines $f(x)$ for all x, although this is also implied by the problem statement.

The main idea is, if we can find some representation of a using a linear combination of vectors $e_{k,j}$, we can easily compute $f(x)$ in $O(n)$ per query: $f(x) = \sum_{k=1}^{n} c_{k,x \bmod k}$ where the coefficients $c_{k,j}$ are such that $\sum_{k=1}^{n} \sum_{j=0}^{k-1} c_{k,j} e_{k,j} = a$.

One approach is to choose values for k, j, calculate the arithmetic mean of $a_{k,j}, a_{k,k+j}, \ldots, a_{k,qk+j}$, where q is such that $(q+1)k + j \geq n$, and subtract this value from all these values, and add this value to the coefficient $c_{k,j}$. Note that this transformation is linear and can thus be represented by a matrix $B_{k,j}$, so we have $a' = B_{k,j} a$ after applying this transformation.

We repeat this process for some sequence of values (k, j) until a converges to the zero vector. At this point, the sum $\sum_{k=1}^{n} \sum_{j=0}^{k-1} c_{k,j} e_{k,j}$ is equal to the original vector.

The trick is to find a sequence of values (k, j) such that this process converges quickly. One such sequence is:

$$(n,0), (n,1), \ldots, (n,n-1), (n-1,0), (n-1,1), \ldots, (n-1,n-2), \ldots, (\lfloor n/2 \rfloor, 0), (\lfloor n/2 \rfloor, 1), \ldots, (\lfloor n/2 \rfloor, \lfloor n/2 \rfloor - 1)$$

Since this transformation is a sequence of linear transformations, it can be represented by a matrix, let's denote it with P. We can verify that every input vector will converge to zero when repeatedly multiplied with P, by verifying this property only for vectors $e_{k,j}$. By computing these matrix-vector products for every (k, j), it was shown using a computer program that all elements of $e'_{k,j} = P^d * e_{k,j}$ will be less than or equal to 10^{-14} for $d = 32$, meaning that the input vector will be close enough to zero after $d = 32$ steps, allowing us to accurately compute $f(x)$. In fact, we can use far smaller values of d, since input data is randomized – balancing between the time limit and correctness.

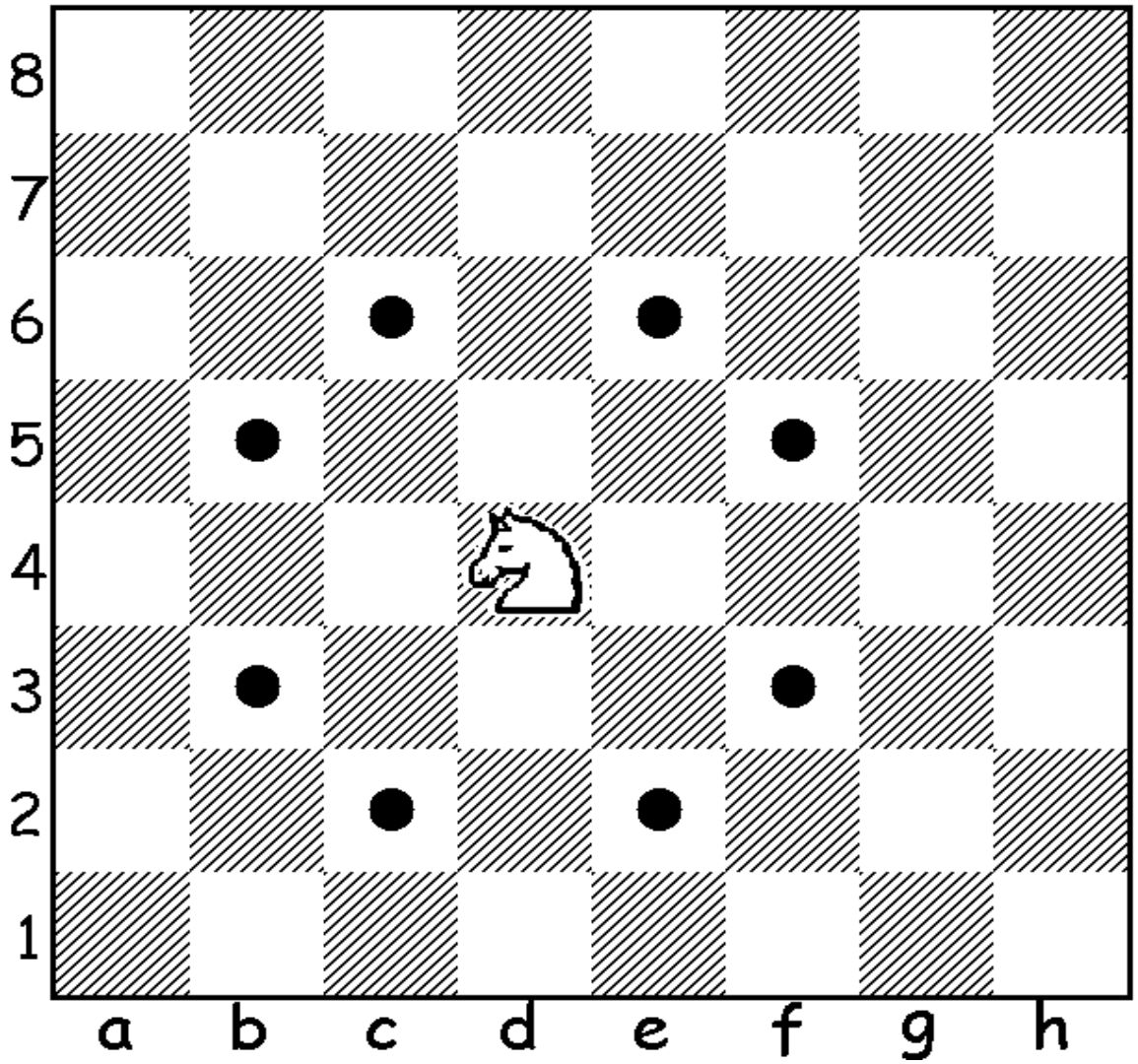
This solution has time complexity $O(d * n^3)$ and space complexity $O(n^2)$.

Resource: Sphere Online Judge

Solution by:
Name: Ivan Stošić
School: Faculty of Science, University of Nis
E-mail: ivan100sic@gmail.com

Problem R2 07: Traveling Knight

Your task is simple. A knight is placed at the top left corner of a chessboard having $2n$ rows and $2n$ columns. In how many ways can it move such that it ends up at a corner after at most k moves?



Input

The first line contains an integer T, the number of test cases. Each of the next T lines contains 2 integers: n, k.

Output

Output T lines, one for each test case, containing the required total number of configurations.

Since the answer can get very big, output it modulo 1000007.

Examples

Input	Output
3 2 1 2 2 3 3	1 5 7

Constraints

$$1 \leq T \leq 100$$

$$2 \leq n \leq 24$$

$$1 \leq k \leq 10^9$$

Time and memory limit: 1s/ 512MB

Solution and analysis

At its heart, this is a simple matrix exponentiation problem, but since there are only 23 different possible matrices it's possible to speed up the usual $O(n^3 \log k)$ algorithm with some precomputation.

1. Generating the matrix

The most straightforward approach is to just consider the chessboard as a graph with $4n^2$ nodes(which are chessboard squares) with an edge between two nodes whenever a knight can hop between corresponding squares in one move. Since the knight can stop at the corner before he made all k moves, an additional node is needed in our graph with an incoming edge from every corner and also an edge from this node to itself.

Because of chessboard and knight moves symmetry, we can reduce the size of our graph. Namely, instead of square's row/column we can only store its distance to the nearest horizontal/vertical side of the board (for example, $\min(r, 2n-1-r)$ instead of r). Another observation is that squares (x, y) and (y, x) are also symmetrical so, finally, the set of nodes in our graph is $\{(x, y) | 0 \leq x \leq y < n\}$ plus one additional node described earlier. This gives us a square matrix of order $\frac{n(n+1)}{2} + 1$, which is just 301 for $n = 24$.

2. Exponentiating the matrix

From now on let N denote the order of the square matrix A, and the task is to find $A^k v$ where v is some vector of order N(with A and v fixed and known beforehand). The usual "exponentiation by squaring" approach works in $O(N^3 \log k)$ which is too slow.

By Cayley–Hamilton theorem, $A^N = \sum_{i=0}^{N-1} c_i A^i$, where coefficients c_i can be found from characteristic polynomial of A. Of course finding characteristic polynomial is hard, but since there are only 23 possible matrices, we can precompute all these polynomials(using Mathematica, NumPy or writing

your own program to do it) and put them into our code directly. Of course, we are only interested in all there coefficients modulo $10^6 + 7$.

Now, let's use an idea similar to binary exponentiation to find $c_{k,0}, c_{k,1}, \ldots, c_{k,N-1}$ such that $A^k = \sum_{i=0}^{N-1} c_{k,i} A^i$ in $O(N^2 \log k)$ time. For $0 \le k < N$ this is trivial: $c_{k,k} = 1$ and the rest of them are zeroes. For $2k \ge N$ $A^{2k} = (A^k)^2 = \left(\sum_{i=0}^{N-1} c_{k,i} A^i\right)^2 = \sum_{i=0}^{2N-2} d_i A^i$ where d_i can be found with polynomial multiplication in $O(n^2)$. This is almost what we needed but there are $2N-1$ coefficients instead of N. Time to use our precomputed values! We can get rid of the leading coefficient $d_j A_j$ while $j \ge N$ by rewriting it as $d_j A^j = d_j \sum_{i=0}^{N-1} c_i A^{j-N+i}$. We repeat this process for $= 2N-2, 2N-3, \ldots, N$, each of $N-1$ steps takes $O(N)$ time so in total it also takes $O(N^2)$ time. For $2k+1 \ge N$ $A^{2k+1} = A^{2k} A = \sum_{i=0}^{N-1} c_{2k,i} A^{i+1}$ so we proceed similarly but we only need to get rid of A^N, so this case only takes $O(N)$ time.

Now we can write $A^k v = \sum_{i=0}^{N-1} c_{k,i} A^i v$. So the only thing left to do is precompute $A^i v$ for $i = 0, 1, \ldots, N-1$, which can be done in $O(N^3)$(by multiplying matrix by vector N times). Unlike characteristic polynomials, this precomputation can be done during your program execution.

All in all, this solution takes $O(N^3)$ preprocessing time for each matrix and $O(N^2 \log k)$ time for each test.

Resource: Sphere Online Judge
Solution by:
Name: Mikhail Mayorov
School: Perm State University, Russia
E-mail: mmaxio@gmail.com

Problem R2 08: Delivery Game

It's time to play. A group of students are about to play a game, this particular game consists of a group of students disposed on a line and a delivery point at the end of this line. The first student of the line has a package, so the goal of this game is simple: the package must be delivered to the delivery point (easy, isn't it?). The game starts with the first student (the one with the package) running to the next student on the line; once he reaches the next student, he can either keep going to the next student on the line or delivery the package to that student, and let him continue the way. This process repeats all along the way. Now the interesting part: every student has preparation time Pi (minutes required to take the package, put it in his knapsack, and get ready to leave) and a travel time Ti (minutes required to travel a single meter). The students want to know the minimum time needed to delivery the package.

Input

The datasets will consist of a number $(1 \leq N \leq 1\,000\,000)$ the number of students in the line. Then follow N lines containing three integers Di, Pi and $Ti, (1 \leq Di, Pi, Ti \leq 1\,000\,000)$ denoting the distance in meters to the delivery point, the preparation time in minutes and the travel time in minutes of the i-th student. Students are given in order (i.e. the first student in the dataset is the one with the package and the farthest to the delivery point).

Output

Print the minimum number of minutes needed to complete the delivery.

Examples

Input	Output
3 5 3 2 4 3 1 1 2 1	12

Time and memory limit: 3s/ 64MB

Solution:

We will use dynamic programming to solve this problem. Let $dp(i)$ denote the minimum time required to reach the i-*th* student.

$$dp(i) = min\{\, dp(k) + P(k) + [D(k) - D(i)] * T(k) \,\}, k < i$$

Now, we can easily compute these values in $O(n^2)$ time complexity, however, that is far too slow. Let's consider a different approach and write the previous relation a little differently:

$dp(i) = min\{ a(k) * D(i) + b(k) \}, k < i$, where

$a(k) = -T(k), b(k) = dp(k) + P(k) + D(k) * T(k)$.

This is a well-known problem of maximizing linear equations, so we will solve it using an envelope.

However, the problem is that we cannot construct it all at once. There are several different approaches to this, and I will describe two of them.

First, we will keep these lines stored by their increasing coefficient $a(k)$ in a balanced binary tree structure. We need to support the following operations: insert a line in the structure while maintaining the envelope and querying for the minimum. Because we only insert lines, once a line is no longer contained in the envelope, it will never again be contained. Thus, after inserting a line, we only need to remove some of the neighboring lines in the tree. Querying can be done with a simple binary search on the tree. This gives an overall time complexity of $O(n \, lg \, n)$

The second approach is somewhat more complex, but more beautiful in my opinion. We will use divide and conquer in the following way: recursively solve the left half, make all transitions from the left to the right half, recursively solve the right half. Now in each stage of the algorithm, we need to make a static envelope from the lines on the left and query it with all the values on the right. If we use merge sort to avoid slow sorting in every step and make all the queries using two pointers, we can achieve the same time complexity of $O(n \, lg \, n)$.

Resource: Caribbean Online Judge
Solution by:
Name: Domogoj Bradac
School: University of Zagreb
E-mail: domagoj.bradac@gmail.com

Problem R2 09: [Challenge] Lawnmower

Dorian the Caretaker hates mowing the lawn. There is nothing worse than listening to this monotonous purr of the engine and inhaling its fumes for a few hours solid. Unfortunately, it just so happens that cutting the grass on the local millionaire's golf course is his only duty.

But, fortunately for Dorian, salvation is here! His employer bought the newest miracle of engineering - Super Mower 3000, which can be remotely controlled. No more monotony, no more sweat and toil! Dorian will be finally fired, so he can start something he wanted to start for a long time - the search for a new job. No proper education and lack of prospects will only make it more fun.

Super Mower 3000 has to be programmed beforehand, though. The golf course is a rectangle with n rows and m columns, which consists of square-shaped fields 1-meter wide. Some fields contain just grass, other fields have obstacles on them. Super Mower can instantly cut the grass just by being on the field. It cannot enter the field with an obstacle. Every grass-covered field is reachable from every other grass-covered field.

Initially, Super Mower stands on the field in the first column of the first row, facing right. It executes a sequence of commands afterwards. There are four different commands:

- N - move one field forwards
- W - move one field backwards
- L - rotate 90 degrees left
- P - rotate 90 degrees right

Moving one field forwards or backwards takes one second. Rotations are slower and take three seconds each.

Given a description of the golf course, devise the sequence of commands that will make Super Mower mow the whole lawn (that is, it will visit every grass-covered field), while taking the least amount of time.

Input

The first line contains a single integer t, denoting the number of testcases. ($t \leq 10$). The descriptions of the testcases follow.

First line of the description consists of two integers n and m ($2 \leq n, m \leq 100$), denoting the size of the golf course. Then n lines follow, each containg m characters. The j-th character in the i-th line describes the field in the i-th row and j-th column. "." (a dot) indicates a grass covered field, "#" indicates an obstacle. It is guaranteed that the first field is grass-covered.

Output

For each test case print one line with a string containing only letters "N","L","W" or "P", denoting the sequence of commands for Super Mower 3000 to execute. The sequence should make Super Mower

visit every grass-covered field, without entering a field with an obstacle or going outside the course, and it cannot be longer than $16 * n * m$ commands.

Examples

Input	Output
2 4 7 ..... .##.##. .##.##. ..... 4 8 ....#.### .#.#.... .#.#....	NNNNNNPNNNPNNNPNNWWLNNNPNN NNNNNNNWWWPNNNLNNNLNLNNNPNNLNNLNNNWWPNNLNN

Scoring

If the sequence of commands satisfies all the conditions given in the problem statement, and it takes x seconds to execute, it is worth $\frac{x}{n*m}$ points. Overall score is equal to the sum of individual scores.

Explanation

The first sequence takes 36 seconds to execute, the second sequence takes 60 seconds. Then, your score is $36/(4 * 7) + 60/(4 * 8)$, so about 3.16.

Time and memory limit: 5s/ 512MB

Solution:

As a first solution, we can use a *dfs* (Depth-first search) algorithm to find path and visit all grass-covered fields.

In *dfs*-tree all nodes except one have a father. So, when Super Mario is located on the grass-covered field which doesn't have unvisited grass-covered neighbor, *dfs* breaks and returns to his father etc., until we can move to the unvisited grass-covered field.

Why is it important to know what is the father of every field? Because we have to reconstruct the path from *dfs* algorithm.

We have 4 sides: right, down, left and up. Let's assign numbers to them.

- Right – 0;
- Down – 1;
- Left – 2;
- Up – 3.

When we arrive on the field (i, j) from $(i, j-1)$. We know that we came to current field from the left side, and then father $[i][j]$ is on the left side, so we can assign father $[i][j] = 2\ (left)$.

Of course, these could be any other numbers.

Now that we have a skeleton of our solution we can optimize it:

Cut the road:

When we get stuck in *dfs* on a grass-covered filed which has no unvisited grass-covered neighbors, instead of returning through ancestors in *dfs* we can use *Dijkstra's* algorithm to find the cheapest path for the field which has unvisited neighbor. We save a lot of commands this way.

Combine:

In *dfs* algorithm we usually ask:

If the left is free dfs(left)

If the right is free dfs(right) etc.

For some test cases, it is better to ask first for right field and then for left. So, we can combine a lot of *dfs* searches and use the best for our solution. Also, we can choose the order of fields by comparing the size of their *dfs* trees. When we start a *dfs* algorithm from some field, we can define the size of the *dfs* tree for current cell as a number of fields that are visited.

Cut the string:

This is a small optimization, but in challenge problems, every optimization is welcome.

At small solutions, we can cut some substring in the solution and check whether we'll visit all the fields, then update the solution.

Resource: Sphere Online Judge/PIZZA 2015

Solution by:
Name: Balsa Knežević
School: School of Electrical Engineering
E-mail: balsak97@gmail.com

Problem R2 10: Terminus Est

Est is a super cute sword spirit that belongs to Kamito. One day, she goes for a walk with him in a spirit forest in Astral Zero. Est quickly realizes that this forest has many clearings and paths, and the clearings and paths actually form a tree structure.

There are N clearings numbered from 1 to N and $N-1$ paths in the spirit forest, and between every pair of clearings there is a unique simple path. In every clearing, there may be a demon spirit, which Est will immediately defeat as she is far superior than these lowly demon spirits. Est is eager to defeat some demon spirits, but there is a problem: she doesn't know which clearing she is in right now (although she memorized the layout of the forest). For lack of a better option, Est decides to just keep moving from her current location without walking over the same path more than once and fight every demon spirit she meets along the way. Est may decide to stop at a clearing at any time during this journey. The path will visit at least two clearings, including the one Est starts at.

A path between clearings i and j $(i < j)$ is considered good if for two parameters a and b

$(0 \leq a \leq b \leq N)$ there are at least a demon spirits and at most b demon spirits on the simple path between i and j. Est will enjoy herself the most if the path she chooses is a good path. Thus, she has Q questions: given parameters a and b, what is the probability that the path she takes is a good path?

Est is quite kind, and as such, she does not want you to deal with incredibly small real numbers. Therefore, if p is the probability, you should output $p * N * \frac{N-1}{2}$. This comes from the fact that the probability of choosing a good path is the number of good paths divided by the total number of paths. Since Est does not know where she is initially, we should assume each clearing has a $\frac{1}{N}$ chance of being Est's initial clearing. Since Est's will cannot be predicted by mere humans, we should also assume each clearing that is not the initial clearing has a $\frac{1}{N-1}$ chance of being chosen as the final clearing where Est stops. In other words, you will just need to output the number of distinct good paths in the spirit forest for every a and b Est asks you to. In particular, a path is considered distinct from another path if one path visits a clearing that the other path doesn't. Therefore, there are $N * \frac{(N-1)}{2}$ distinct paths in total.

Note: Demon spirits don't move from their initial clearings.

Input

The first line of input will have N.

The second line of input will have N space-separated digits, either 0 or 1. If and only if the i-th number is 1, the i-th clearing has a demon spirit.

The next $N-1$ lines describe the spirit forest. Each line in the form u v, which means the clearings u and v are directly connected.

The $(N+2)$-th line with have Q.

The next Q lines each have a and b, separated by a single space.

Output

There should be Q lines of output, the answers to Est's questions. You should output the answers to Est's questions in the order that they are given.

Constraints

$$2 \leq N \leq 100000$$

$$1 \leq Q \leq 200000$$

Examples

Input	Output
8 0 1 1 1 1 0 0 1 2 1 3 1 4 1 5 4 6 5 7 4 8 4 3 0 8 1 2 3 3	28 20 8

Time and memory limit: 8s/ 256MB

Solution:

First of all, let's find the answer for each i from 0 to N, and find prefix sums on these answers, to reply the queries on $O(1)$.

We will use centroid decomposition and *FFT* to solve the new problem. You can find information about this algorithms in the net.

We fix some centroid and its subtree and let's consider all the paths going through this centroid. Path has three parts: starting from some vertex v from one subtree, it is going to centroid, and then going to some vertex u in another subtree. If the first part has sum equal to $s1$, the second part has sum equal to $s2$, and the third — $s3$, then we have to add 1 path to answer $[s1 + s2 + s3]$. $s2$ in this sum is constant, equal to $0/1$, so we will consider it manually. Okay, now, let's *DFS* each subtree and calculate the number of paths to centroid with each sum. $Cnt[i]$ will be the number of such paths from centroid to one subtree, which have sum i. Now, it's obvious, to add to the answer array all the paths quckly, we have to multiply this arrays as polynoms. We iterate the subtree, multiply it's Cnt array and

all the sum of all the previous subtrees, and add the product to the answer array. Sum of the two arrays A and B is such array C, where $C[i] = A[i] + B[i]$ for each i. To reach good complexity, we have to iterate the subtrees in an increasing order (because *FFT* multiplies two polynoms of size N and M in

$$(N + M) * log(N + M)).$$

Complexity: $N * log^2(N)$

Resource: DMOJ
Solution by:

Name: Yuri Shilayaev
School: Gymnasium 1, Minsk
E-mail: sda2000.yura@gmail.com
