Microsoft

Development
Center
Serbia

bubble cup

challenge the future

Booklet

BUBBLE CUP 2016

Student programming contest

Microsoft Development Center Serbia

Problem set & Analysis from the Finals and Qualification rounds

Scientific committee
Aleksandar Damjanović
Ibragim Ismailov
Stefan Velja
Dimitrije Erdeljan
Marko Rakita
Borna Vukorepa
Slavko Ivanović
Daniel Silađi
Dimitrije Dimić
Rastko Suknjaja
Aleksandar Kiridžić

Qualification analyses
Ivan Stošić
Petar Veličković
Aleksa Milojević
Luka Vukelić
Daniel Paleka
Srđan Marković
Ivan Paljak
Nikola Jovanović
Nikola Herceg
Viktor Lučić
Domagoj Bradac
Adrian Beker
Roman Bilyi
Josip Kelava
Vsevolod Stepanov
Vladimir Milenković
Sergey Kulik
Miloš Suković

Cover
Sava Čajetinac

Typesetting
Ivana Tomić
Marijana Mikić
Ivana Čukanović

Volume editor
Dragan Tomić

Contents

Dear Finalist of Bubble Cup 9,

Thank you for participating in the ninth edition of the Bubble Cup. I hope you had a great time in Belgrade and enjoyed the event.

MDCS has a keen interest in organizing Bubble Cup. Many of our engineers participated in similar competitions in the past. One of the characteristics of our team culture in MDCS is passion for solving challenging technical problems.

Bubble Cup 9 has attracted the largest number of participants so far, with more than 600 students participating during qualifications. In addition to countries that have participated in Bubble Cup finals before (Serbia, Croatia, Belarus, Ukraine, Russia), we had several new countries this year (Romania, Bulgaria and Letonia). Also for the second time we organized an online mirror of the finals, with over 2000 teams registered for the competition.

Given that we live in the world where technological innovation will shape coming decades, your potential future impact on humankind will be great. Use these opportunities to advance your technical knowledge and to build relationships that could last you a lifetime.

Thanks,
Dragan Tomić
MDCS PARTNER Engineer manager/Director

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

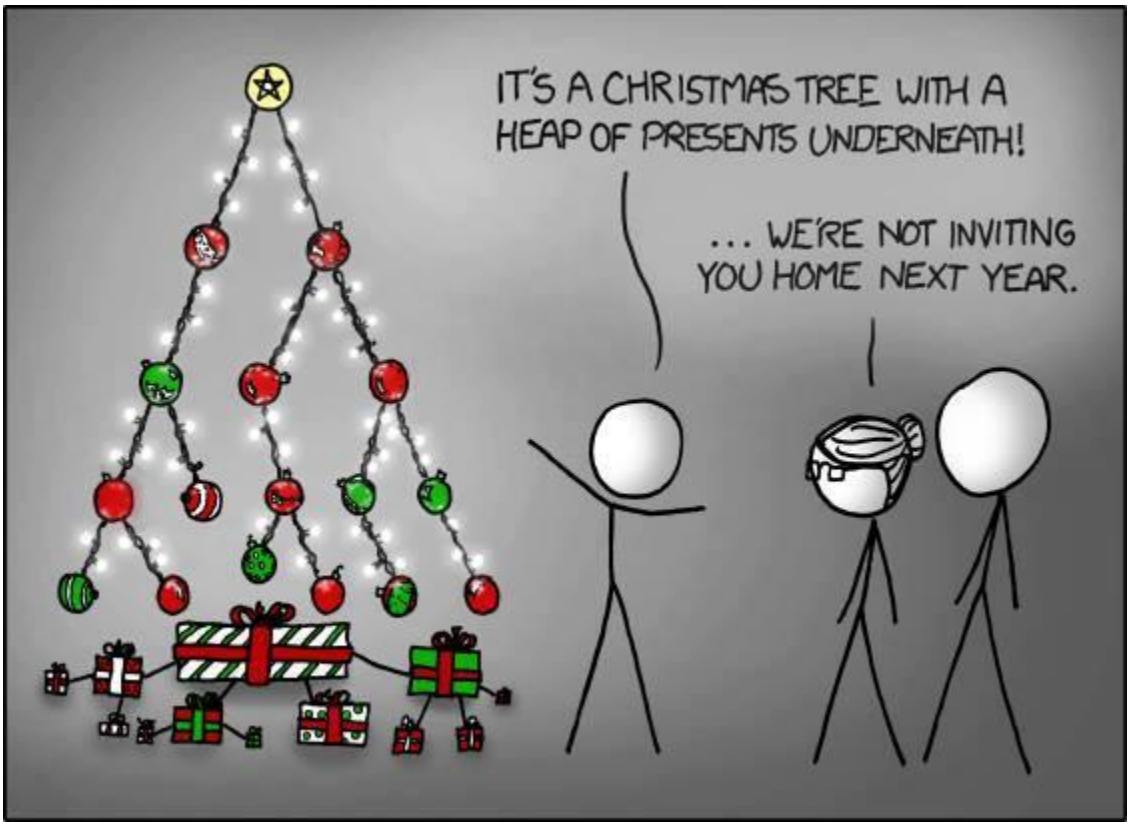
Format of the competition has remained the same this year. All competitors battled for the place in finals during two qualifications rounds. Top high school and university teams were invited to the finals in Belgrade, where they competed in the traditional five hours long contest. Finalists also had the opportunity to listen to experts in competitive programming during the Bubble Cup Conference, which was organized for the first time this year.

Microsoft Development Center Serbia (MDCS) was created with a mission to take an active part in conception of novel Microsoft technologies by hiring unique local talent from Serbia and the region. Our teams contribute components to some of Microsoft's premier and most innovative products such as SQL Server, Office, Bing. The whole effort started in 2005, and during the last 11 years a number of products came out as a result of 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 of 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



Finals 2016

Bubble Cup finals were held on September 9 and 10 in Startit Center. The event started with Bubble Cup opening ceremony, where **Tatjana Matić -** State secretary for Telecomunications, **Dražen Šumić –** Principal PM at MDCS**, Duško Obradović –** Long time mentor of Sombor teams and **Aca Ivanović** multiple BBC finalist.

The final competition remained in the traditional 5-hours format, similar to ACM ICPC. This year, university and high school students competed in the same category. In one of the most exciting Bubble Cup finals so far, 20 teams were presented with a total of 9 problems.

Final standings were unknown until the very end of the competition. Many successful submissions and changes on the scoreboard happened in the last 15 minutes of the finals. Ultimately, the team **LNU Penguins** (Ukraine, university) rose to the top, as the only team that solved 6 problems. The second place went to **koala lumpur** (Croatia, Gimnasium), followed by team **Elite Nicaraguan Eagles** (Russia, St. P. University). For Bubble Cup 9 we decided to give additional awards to teams that were bellow the overall top 3 bar but are in top three in their categories: 3rd best University - **Prvo u Drvo** (Croatia), 2nd Best High school - **-1/12 and still counting (Bulgaria),** 3rd Best High school – **HC++** (Croatia).

In addition to on-site finals in Belgrade, an online mirror of the finals was organized on **Codeforces** with team **1322 (tourist, VArtem)** emerging victorious as the only team that solved **all** the problems. Codeforces results: http://codeforces.com/contest/717/standings

Problem A: Cowboy Beblop at his computer

Authors

Stefan Velja

Implementation and analysis

Slavko Ivanović/Stefan Velja

Cowboy Beblop is a funny little boy who likes sitting at his computer. He somehow obtained **two** elastic hoops in the shape of $2D$ polygons, which are **not necessarily convex**. Since there's no gravity on his spaceship, the hoops are standing still in the air. Since the hoops are very elastic, Cowboy Beblop can stretch, rotate, translate or shorten their edges as much as he wants.

For both hoops, you are given the **number of their vertices**, as well as the **position of each vertex**, defined by its X, Y and Z coordinates. The vertices are given in the order they're connected: the 1st vertex is connected to the 2nd, which is connected to the 3rd, etc., and the last vertex is connected to the first one. The hoops are connected if it's impossible to pull them to infinity in different directions by manipulating their edges, **without** having their edges or vertices intersect at **any point** – just like when two links of a chain are connected. The polygons' edges do **not intersect** or **overlap**.

Cowboy Beblop is fascinated with the hoops he has obtained and he would like to know whether they are **connected or not**. Since he's busy playing with his dog, Zwei, he'd like you to figure it out for him. He promised you some sweets if you help him!

Input:

The first line of input contains an integer N, which denotes **the number of edges of the first polygon**.

The next N lines each contain the integers X, Y and Z - **coordinates of the vertices**, in the manner mentioned above.

The next line contains an integer M, denoting the **number of edges of the second polygon**, followed by M lines containing the **coordinates of the second polygon's vertices**.

Output:

Your output should contain only one line, with the words "YES" or "NO", depending on whether the two given polygons are connected.

Constraints:

- $3 \leq \mathrm{N} \leq 10^5$
- $3 \leq \mathrm{M} \leq 10^5$
- $-10^6 \leq \mathrm{X}, \mathrm{Y}, \mathrm{Z} \leq 10^6$
- It is guaranteed that both polygons are simple (no self-intersections), and in general that the obtained polygonal lines do not intersect each other. Also, you can assume that no 3 consecutive points lie on the same line.

Example input:

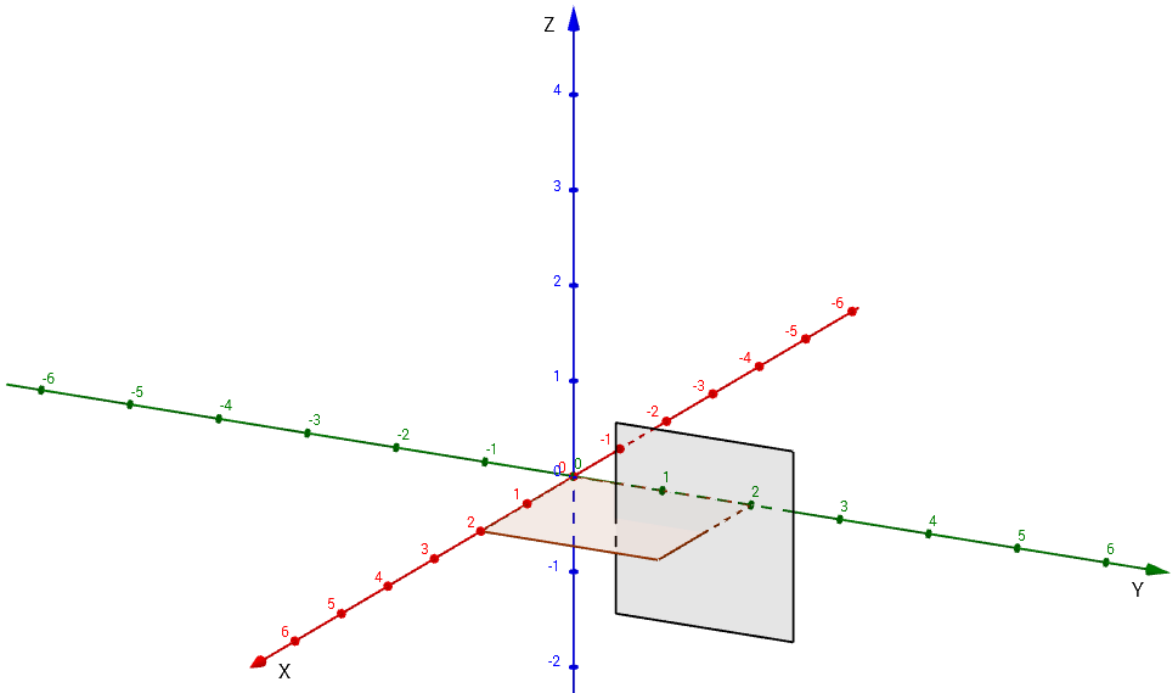
```
4
0 0 0
2 0 0
2 2 0
0 2 0
4
1 1 -1
1 1 1
1 3 1
1 3 -1
```

Example output:

YES

Explanation:

In the picture below, the two polygons are connected, as there is no way to pull them apart (they are shaped exactly like two square links in a chain). Note that the polygons do not have to be parallel to any of the $xy-, xz-, yz-$ planes in general.



Time and memory limit: 1s/64MB

Solution and analysis:

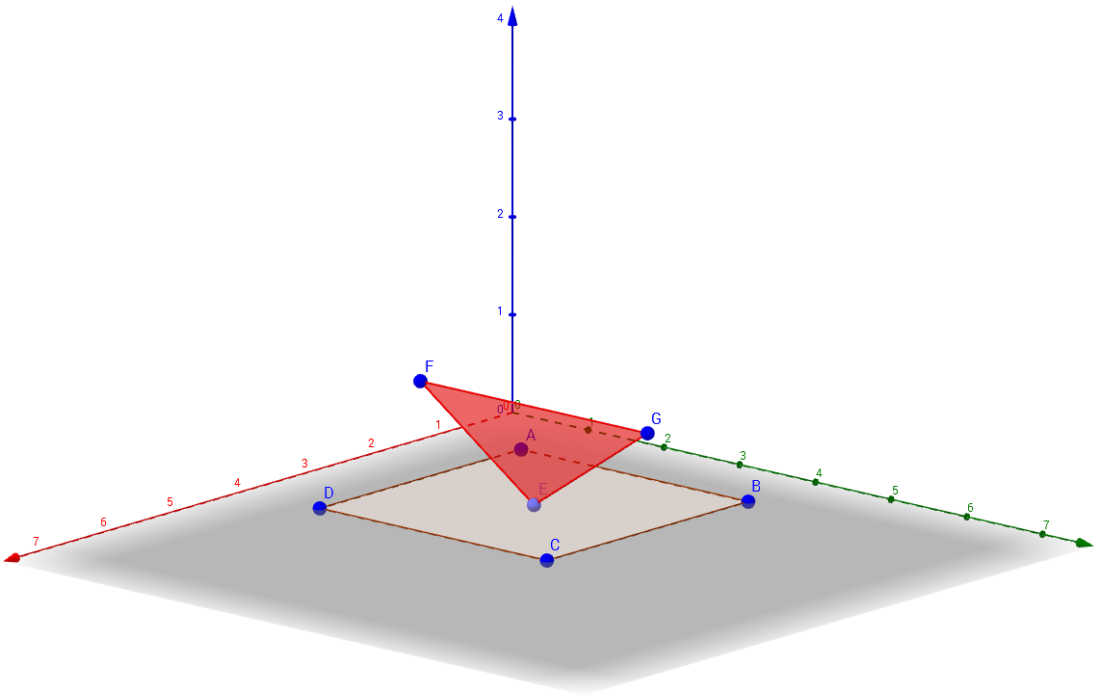
The solution to this task effectively consists of two parts: analyzing the geometry and the relations between the two polygons, and deriving whether they intersect or not.

For the second part, we need to find all intersections between each of the polygons and the common line of their two planes. Once that is done, and the intersection points along the line are sorted, we can simply go through them and count the number of intersection of, say, polygon P1 with the inside of polygon P2, as well as keep track of the directions in which it passes through P2. Whenever we reach a point belonging to polygon P2, our position (inside or outside of polygon P2) changes. Now we simply count points belonging to polygon P1, which we reach while being inside of the polygon P2. This solution has the complexity of $O(N * logN)$, where N is the sum of the edges of the two polygons.

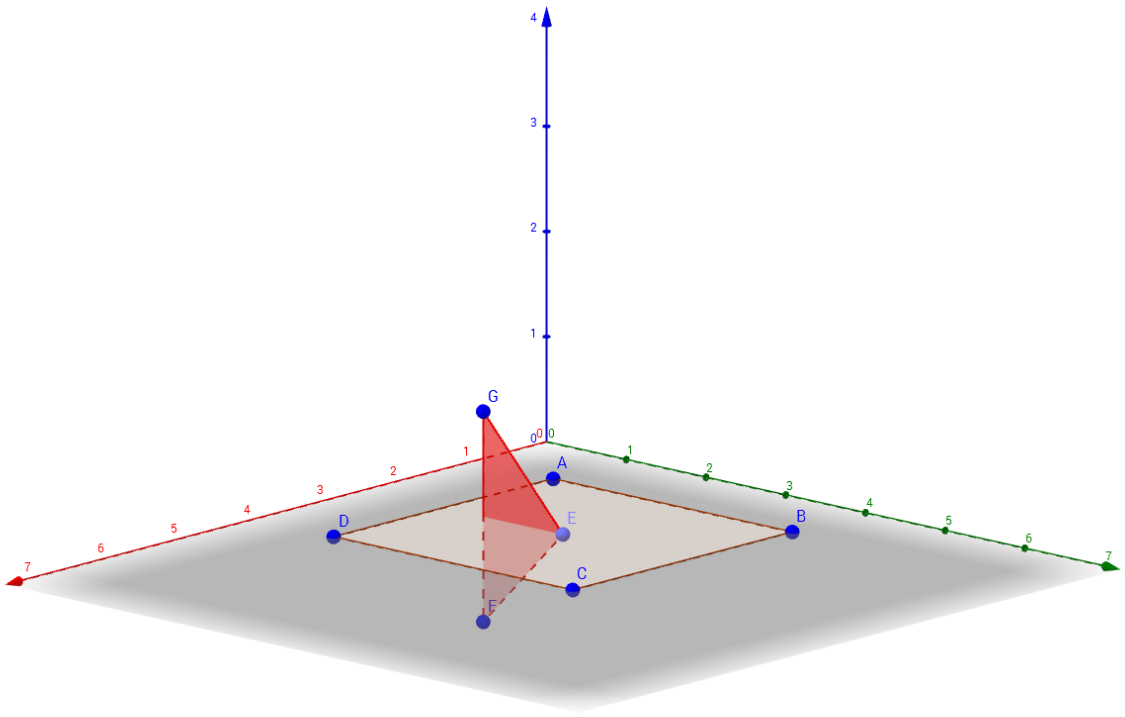
As for the geometry – it seems that an approach using vectors (the mathematical ones, not arrays with variable length) is much easier than the others. It relieves the coder from having to solve complicated equations, but instead uses relatively simple calculus, once all the vector operations have been defined. Thus, we first define the vectors of the two polygons' planes as the vector product of two consecutive edges' vectors. The two consecutive vectors will be used as the base of the $2D$ vector space defined by the plane. Then, for each edge of both polygons, we need to see whether it has intersections with the other polygon's plane or not. Let's observe the case when the edge's end points (let's label them A and B) are on the opposite sides of the plane (α). If the angle between α's direction vector and the vector from A to a random point in α is acute, then the angle between α's direction vector and the vector from B to the same point will be obtuse, and vice-versa. Thus, if the dot products of the two vectors for both A and B are of different sign, they are on the opposite side of α and there is an intersection. If either of the products is zero, then the corresponding point is inside α (this case will be discussed later on).

Finding the point of intersection (point X) between the line AB and α can be done in several ways. One solution is to pick a random point in α (point C) and observe the vector CA. It is easy to see that one can represent it as a linear combination of α's base vectors and the vector AB. Once the parameters of the linear combination have been found by solving the system of equations (determinants seem easiest), one can simply ignore the component associated with the vector AB. The rest will sum up to the vector CX, and thus X is found. It can also be done with even simpler expressions using vectors (left to the reader to find out), but this solution seemed very intuitive and easy to code, so I stuck with it.

The only special case happens when a polygon's vertex lies exactly on the other polygon's plane. In that case, one simply observes the previous and the next vertex and whether they are on the opposite sides of the plane or not. If they are – the intersection is taken as the "problematic" point and if they are not, we assume there is no intersection. See Picture 1 and Picture 2. Similar applies when two consecutive points are lying on the plane. Please note that simply ignoring the point lying on the other polygon's plane is wrong.



Picture 1: *Point E is on plane of A-B-C-D polygon. FG segment doesn't intersect A-B-C-D polygon, so E is not point of intersection.*



Picture 2*: Point E is on plane of A-B-C-D polygon. In this case FG segment intersect A-B-C-D polygon, so E is point of intersection.*

Problem B: Underfail

Authors — ***Aleksandar Damjanović/Stefan Velja***

Implementation and analysis — ***Marko Rakita/Rastko Suknjaja***

You have recently fallen through a hole and, after several hours of unconsciousness, you realized you are in an underground city. On one of your regular daily walks through the unknown, you have encountered **two** unusually looking skeletons called **Sanz and P'pairus**, who decided to accompany you and give you some puzzles for seemingly unknown reasons.

One day, Sanz has created a **crossword** for you. Not any kind of crossword, but a $1D$ crossword! You are given a string of length N and M words, none of which is longer than K. You are also given an array $P[\ ]$, which designates how much each word is worth – the i^{th} word is **worth** $P[i]$ points.

Whenever you find one of the M words in the string, you are given **the corresponding number of points**. Each letter in the crossword can be used at **most** X times. A certain word can be counted at different places, but you **cannot** count the same appearance of a word multiple times. If a word is a substring of another word, you can count them both (presuming you haven't used the letters more than X times).
In order to solve the puzzle, you need to tell Sanz what's the **maximum achievable number** of points in the crossword.

Input:

The first line of input will contain one integer N– the length of the crossword, and the second line will contain the crossword string. The third line will contain the integer M – the number of given words, and the next M lines will contain descriptions of words: each line will have a word string and an integer p. The last line of the input will contain X – the maximal number of times a position in the crossword can be used.

Output:

Output a single integer – the maximal number of points you can get.

Constraints:

- $1 \leq \mathrm{N} \leq 500$
- $1 \leq \mathrm{M} \leq 100$
- $1 \leq \mathrm{X} \leq 100$
- $1 \leq \mathrm{K} \leq 500$
- $0 \leq \mathrm{p} \leq 100$

Example input:

6
abacba
2
aba 6
ba 3
3

Example output:

12

Explanation:

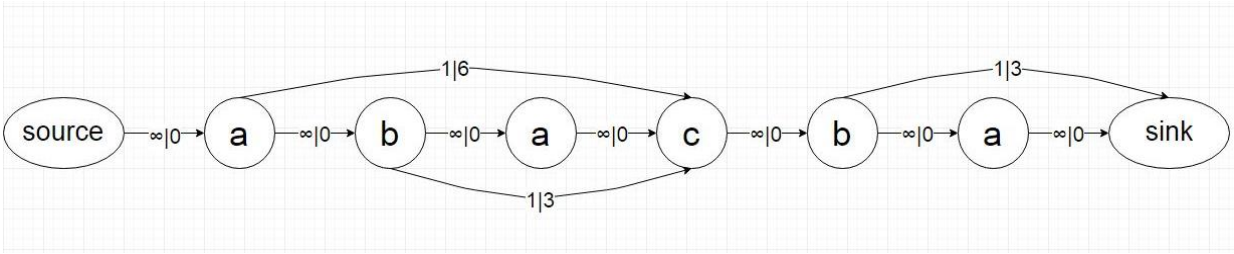
For example, with the string "$abacba$", words "aba" (6 points) and "ba" (3 points), and $X = 3$, you can get at most 12points - the word "aba" appears once ("$abacba$"), while "ba" appears two times ("$abacba$"). Note that for $X = 1$, you could get at most 9 points, since you wouldn't be able to count both "aba" and the first appearance of "ba".

Time and memory limit: 2s/256 MB

Solution and analysis:

In the basic case (when $X = 1$) we have a common DP problem. Unfortunately, for larger X it is much more complicated, so the basic case cannot be generalized.

Instead, we can look at this problem as a graph problem. We represent the crossword as a directed weighted graph, where edges have both a cost and a capacity. First, we represent every letter from the crossword as a vertex, and add an extra source and sink node. Then, we connect each letter's vertex with the next one with an edge of capacity ∞ and cost 0. Also, we connect the source node with the first letter and we connect the last letter with the sink node using the same kind of edge. Finally, for every position where one of the given words matches a substring, we connect the first letter of the match with a letter just after the end of the match with an edge of capacity 1 and a cost equal to the score we get for the word. By doing it for an example input we get this graph:



Any flow with total capacity X is equivalent to a set of selected words that satisfies the given constraint, since we can have at most X "overlapping" 1-capacity edges. The problem now reduces to maximizing the cost of that flow, which can be done by adapting an algorithm that

solves the min-cost flow problem (by using negative values). Using an adaptation of Bellman-Ford algorithm we can get a complexity of $O(E^2VloqV)$.

Problem C: Paint it really, really black

Authors

Borna Vukorepa/Stefan Velja

Implementation and analysis

Borna Vukorepa/Stefan Velja

I see a **pink** boar and I want it painted **black**. **Black** boars look much more awesome and mighty than the **pink** ones. Since Jaggy became the ruler of the forest, he has been trying his best to improve the diplomatic relations between the forest region and the ones nearby.

Some other rulers, however, have requested too much in return for peace between their two regions, so he realized he has to resort to intimidation. Once a delegate for diplomatic relations of a neighboring region visits *Jaggy's* forest, they might suddenly change their mind about attacking Jaggy, if they see a whole bunch of black boars. Black boars are really scary, after all.

Jaggy's forest can be represented as a tree (graph without cycles) with N vertices. Each vertex represents a boar and is colored either black or pink. Jaggy has sent a squirrel to travel through the forest and paint all the boars black. The squirrel, however, is quite unusually trained and while it traverses the graph, it changes the color of every vertex it visits, regardless of its initial color: **pink** vertices become **black** and **black** vertices become **pink**.

Since Jaggy is too busy to plan the squirrel's route, he needs your help. He wants you to construct a walk through the tree starting from vertex 1 such that in the end all vertices are **black**. A walk is any alternating sequence of vertices and edges, starting and ending with a vertex, such that every edge in the sequence connects the vertices before and after it.

Input:

The first line of input contains the integer N, denoting the number of nodes of the graph. The following line contains N integers, which represent the color of each node.

If the i^{th} integer is:

- 1, then the corresponding node is **black**
- -1 , then the node is **pink**.

Each of the next $N - 1$ lines contains two integers, which represent the indexes of the nodes which are connected (one-based).

Output:

Output the path of a squirrel: output a sequence of visited nodes' indexes in order of visiting. In case all of the nodes are initially black you should print 1.

Constraints:

- $2 \leq N \leq 2 \cdot 10^5$

Example input:

```
5
1 1 -1 1 -1
2 5
4 3
2 4
4 1
```

Example output:

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

Explanation:

At the beginning squirrel is at node 1 and its color is **black**. Next steps are as follows:

- From node 1 we walk to node 4 and change its color to **pink**
- From node 4 we walk to node 2 and change its color to **pink**
- From node 2 we walk to node 5 and change its color to **black**
- From node 5 we return to node 2 and change its color to **black**
- From node 2 we walk to node 4 and change its color to **black**
- Finally, we visit node 3 and change its color to **black**

Time and memory limit: 2s/256MB

Solution and analysis:

Root the tree at node 1. Now notice that if we had a function F such that $F(n)$ colors the whole subtree of node n black (except maybe n itself) and returns us to n in the process, we can easily solve the problem. Let p be the parent of n. Then after entering n do $F(n)$. If n is black, go to p and never return to that subtree again. Otherwise, go from n to p then to n then to p. Now n is black and we can continue to do the same for other children of p and so on.

Now it is easy to see that all that we need to do is to make DFS-like tour of the tree and upon returning from a node to a corresponding parent we check the color of the child node. If it is black, continue normally, otherwise visit it again and again return to the parent and then continue normally. The only exception is the root node as it has no parent. After finishing the tour and ending up in node 1, if it is black, we are done. If not, it is the only white one and we can select any child $\boldsymbol{c}$ of 1 and do $1 - c - 1 - c$. Now all nodes are black.

It is recommended to do the implementation with stack. Complexity is $O(N)$.

Problem D: Potions homework

Authors

Ibragim Ismailov
Stefan Velja

Implementation and analysis

Ibragim Ismailov
Aleksandar Damjanović

Harry Water, Ronaldo, Her-my-oh-knee and their friends have started a new school year at their ***MDCS School of Speechcraft and Misery***. At the time, they are very happy to have seen each other after a long time. The sun is shining, birds are singing, flowers are blooming, and their Potions class teacher, professor Snipe is sulky as usual. Due to his angst fueled by disappointment in his own life, he has given them a lot of homework in Potions class.

Each of the N students have been assigned a single task. Some students do certain tasks faster than others. Thus, they want to redistribute the tasks so that each student still does exactly one task, and that all tasks are finished.

Each student has their own laziness level, and each task has its own difficulty level. Professor Snipe is trying hard to improve their work ethics, so each student's laziness level is equal to their task's difficulty level.

Both sets of values are given in the array A, where $A[i]$ represents both the laziness level of the i^{th}student and the difficulty of their task. The time a student needs to finish a task is equal to the product of their laziness level and the task's difficulty.

They have asked you what is the shortest possible (total) time they must spend to finish all tasks.

Input:

The first line of input contains the integer N, which represents the total number of tasks. The next N lines contain exactly one integer each, which represents the difficulty of the task and the laziness of the student who initially received the task.

Output:

Your output should consist of only one line – the minimum time needed to finish all tasks, modulo 10007.

Constraints:

- $1 \leq \mathrm{N} \leq 100\,000$
- $1 \leq \mathrm{A[i]} \leq 100\,000$

Example input:	Example output:
2 1 3	6

Explanation:

If the students switch their tasks, they will be able to finish them in 3+3=6 time units.

Time and memory limit: 0.1s/64MB

Solution and analysis:

Everything is pretty simple here. What should be done here is to give the laziest student the easiest task, because it is always optimal to do so. If we do that continuously, it becomes obvious that solution is to sort the given array and get the following formula:

$$\sum_{i=1}^{N} A[i]^* A[N-i-1]$$

Problem E: Festival organization

Authors

Ibragim Ismailov/Stefan Velja

Implementation and analysis

Ibragim Ismailov/Dimitrije Erdeljan

The Prodiggers are quite a cool band and for this reason, they have been the surprise guest at the ENTER festival for the past 80 years. At the beginning of their careers, they weren't so successful, so they had to spend time digging channels to earn money; hence the name.

Anyway, they like to tour a lot and have surprising amounts of energy to do extremely long tours. However, they hate spending **two consecutive days** without having a concert, so they would very much like to **avoid** it.

The Prodiggers would like to hold K t**ours** of length of at least L **days**, and at most R **days**. Since they are quite superstitious, they want all their tours to have the **same length** and **different schedules** (regarding playing concerts and skipping days). Additionally, they would absolutely hate to skip two consecutive days in a **single** tour. Since their schedule is quite busy, they want you to tell them in **how many ways** can they hold the K tours, modulo M.

Input:

The first and only line of input will contain 3 numbers: *K*, *L*, *R*

Output:

Output a single number: in how many ways can they hold the *K* tours, modulo *M*=1000000007.

Constraints:

- $1 \leq K \leq 200$
- $1 \leq L \leq R \leq 10^{18}$

Example input:	**Example output:**
1 1 2	5

Time and memory limit: 1 s/256MB

Solution and analysis:

It is easy to prove that the answer for the query is $\sum_{n=l}^{r}\binom{F_{n+2}}{k} = \sum_{n=0}^{r}\binom{F_{n+2}}{k} - \sum_{n=0}^{l-1}\binom{F_{n+2}}{k}$, where F_n are Fibonacci numbers. Let's notice, that $\binom{x}{k}$ is a polynomial for x and can be expressed in the

form $c_0 + c_1 x + \ldots + c_k x^k$. Knowing this we can express the answer in a different way $\sum_{n=0}^{r} \binom{F_{n+2}}{k} = \sum_{n=0}^{r} \sum_{m=0}^{k} c_m F_{n+2}^m = \sum_{m=0}^{k} c_m \sum_{n=0}^{r} F_{n+2}^m$. Thus we reduced our problem to computing the sum of m^{th} powers of Fibonacci numbers. To do so we will refer to Binet's formula $F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$ or $F_n = \frac{\sqrt{5}}{5} \left(\phi^n - \psi^n \right)$.

$$F_n^m = \left(\frac{\sqrt{5}}{5} \right)^m \left(\phi^n - \psi^n \right)^m = \left(\frac{\sqrt{5}}{5} \right)^m \sum_{j=0}^{m} (-1)^{m-j} \binom{j}{m} \phi^{nj} \psi^{n(m-j)}$$

$$\sum_{n=0}^{r} F_n^m = \left(\frac{\sqrt{5}}{5} \right)^m \sum_{n=0}^{r} \sum_{j=0}^{m} (-1)^{m-j} \binom{j}{m} \left(\phi^j \psi^{(m-j)} \right)^n = \left(\frac{\sqrt{5}}{5} \right)^m \sum_{j=0}^{m} (-1)^{m-j} \binom{j}{m} \sum_{n=0}^{r} \left(\phi^j \psi^{(m-j)} \right)^n$$

The inner sum is almost always a geometric progression with $b_0 = 1$ and $q = \phi^j \psi^{(m-j)}$, except for the cases when $\phi^j \psi^{(m-j)} = 1$, but we may avoid any special cases by computing it in a way similar to binary exponentiation.

Indeed, in order to compute $\sum_{n=0}^{r} q^n$, we may start with computing $\sum_{n=0}^{2^k-1} q^n$ and q^{2^k}. Two sums with m1 and m2 elements can be merged together in the following way: $\sum_{n=0}^{m1+m2-1} q^n = \sum_{n=0}^{m1-1} q^n + q^{m1} \sum_{n=0}^{m1+m2-1} q^n$.

Thus we can compute the sum of m^{th} powers only with additions and multiplications. The only difficulty is that there is $\sqrt{5}$ present in these formulas (in ϕ and ψ) and there is no such number modulo $10^9 + 7$. The solution to this is simple: let's never specify an exact value for it. This way we will always work with numbers of the form $a + \sqrt{5} b$. The addition and multiplication of these numbers is fairly easy:

$$\left(a + \sqrt{5} b\right) + \left(c + \sqrt{5} d\right) = (a + c) + \sqrt{5}(b + d),$$

$$\left(a + \sqrt{5} b\right) \cdot \left(c + \sqrt{5} d\right) = (ac + 5bd) + \sqrt{5}(ad + bc).$$

These are the only operations that we require. The sum of Fibonacci numbers (which are integers) is also integer, so the final result will never contain any $\sqrt{5}$. Thus we have solved this problem.

Problem F: Pokermon League challenge

Authors

Borna Vukorepa/Stefan Velja

Implementation and analysis

Borna Vukorepa/Aleksandar Damjanović

Welcome to the world of **Pokermon**, yellow little mouse-like creatures, who absolutely love playing **poker**!

Yeah, right…

In the ensuing **Pokermon League**, there are N registered Pokermon **trainers**, and T existing trainer **teams**. Since there is a lot of jealousy between trainers, there are E **pairs of trainers** who **hate** each other. Their hate is **mutual**, there are **no identical pairs** among these, and no trainer hates himself (the world of Pokermon is a joyful place!). Each trainer has **a wish-list of length** L of teams he'd like to join. All the teams are divided into two conferences.

Your task is to divide players into teams and the teams into **two conferences**, so that:

- each trainer belongs to exactly one team
- no team is in both conferences
- total hate between conferences is at least $\frac{E}{2}$
- every trainer is in a team from his wish-list

Total hate between conferences is calculated as the number of pairs of trainers from teams from different conferences who hate each other.

Input:

The first line of input contains 2 non-negative integers:

- N -**total number** of Pokermon trainers
- E -**number of pairs** of trainers who hate each other

Each Pokermon trainer is represented by a number between $[1, N]$.
The next E lines contain 2 integers A and B indicating that Pokermon trainers A and B **hate each other**.
The next $2N$ lines are in a following format:
Starting with Pokermon trainer 1, for every trainer in consecutive order:

- $\text{first number } L$- a size of Pokermon trainers wish list
- in the next line are positive integers $t[i]$ - the teams Pokermon trainer would like to be on.

Each trainer's wish list **will not contain** repeating teams.

Teams on the wish lists are numbered in such a way that the set of all teams that appear **on at least 1 wish list** is set of consecutive positive integers $\{1, 2, 3, \dots, T\}$.

Output:

Contains 2 lines:

- The first line contains N numbers, **specifying the team every trainer is in**. First for trainer 1, then 2, etc.
- The second line contains T numbers where for **every team**, starting with team 1, there is an integer **specifying the conference** $(1\ or\ 2)$**) of that team**.

Constraints:

- $4 \leq N \leq 50\,000$
- $2 \leq E \leq 100\,000$
- $16 \leq L \leq 20$
- $1 \leq\ t[i]\ \leq\ T$
- $1 \leq\ T\ \leq\ 1\,000\,000$
- $1 \leq A, B\ \leq N$

Example input:

```
4 3
1 2
2 3
4 1
16
1 2 3 4 5 6 7 8 9 10 11 12 13 14 16 15
16
2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18
16
2 3 4 5 6 7 8 9 10 11 12 13 14 15 18 19
16
1 2 3 4 5 6 7 8 9 10 11 12 13 14 16 19
```

Example output:

```
1 2 2 3
1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Explanation:

Conference 1 contains only team 1, and conference 2 contains all other teams. Total hate between conferences is 2 which is greater than $\frac{E}{2} = \frac{3}{2} =\ 1.5$.

Pokermon trainer 1 belongs to team 1, trainers 2 and 3 to team 2 and trainer 4 to team 3. Other teams are empty but they have been assigned a conference.

Time and memory limit: 5s/256MB

Solution and analysis:

The key idea is to first put players into conferences and then assign them teams. We go through all of the players and put them into conferences so that the first condition is satisfied. We take them one by one and select a conference for each player such that he hates more (or equally many) players in the other conference that the one we put him in. That means that at any point of this process, there will be more hate edges connecting players in different conferences than those connecting players in the same conferences. Thus, in the end, we have satisfied first condition. Complexity is $O(N + E)$ for this part.

Now we will try to satisfy the second condition by assigning teams to the players. Let $\boldsymbol{S}$ be the set of all of the teams that appear on at least one wish-list. Let's select some subset $\boldsymbol{A}$ of $\boldsymbol{S}$. We select it in a way that every element from $\boldsymbol{S}$ will be thrown in it with 50% chance. Now, we will assign teams from $\boldsymbol{A}$ to players in conference 1 and teams from $\boldsymbol{Ac}$ to players in conference 2. If such assigning is possible while satisfying the second condition, we will be done. Let's see what is the chance that a particular player in conference 1 can't be assigned a valid team from $\boldsymbol{A}$. That means none of at least 16 teams on his wish-list are in $\boldsymbol{A}$. The chance for that is $\frac{1}{2^{16}}$. Same goes for all players in conference 1 and equivalently for conference 2. So, the chance that at least one player can't be assigned a team is at most $N \cdot \frac{1}{2^{16}} \leq 0.77$. This means that there is at least 23% chance that this method will give us the final solution which fulfills both problem conditions. This means that after a several steps, we will very likely solve the problem. Complexity is $O(NL)$ for this part. For example, if we do 30 steps, the probability that we will find a correct solution is $1 - 0.7730 = 0.9996$.

Problem G: Heroes of Making Magic III

Authors

Dimitrije Dimić/Stefan Velja

Implementation and analysis

Dimitrije Dimić/Daniel Silađi

I'm strolling on sunshine, yeah-ah! And don't it feel good!

Well, it certainly feels good for our **Heroes of Making Magic**, who are casually walking on a one-directional road, fighting imps. Imps are weak and feeble creatures and they are not good at much. However, Heroes enjoy fighting them. For fun, if nothing else.

Our Hero, **Ignatius**, simply adores imps. He is observing a line of imps, represented as a **zero-indexed array** of integers $A[]$ of **length** N, where $A[i]$ denotes the **number of imps at the** i^{th}position. Sometimes, imps can appear out of nowhere.

When heroes fight imps, they select a segment of the line, start at one end of the segment, and finish on the other end, **without** ever **exiting** the segment. They can move **exactly one** cell left or right from their current position and when they do so, they defeat one imp on the cell that they moved to, so, the number of imps on that cell decreases by one. This also applies when heroes appear at one end of the segment, at the beginning of their walk.

Their goal is to defeat all imps **on the segment**, without ever moving to an empty cell in it (without imps), since they would get bored. Since Ignatius loves imps, he doesn't really want to fight them, so **no** imps are harmed during the events of this task. However, he would like you to tell him whether it would be possible for him to clear a certain segment of imps in the abovementioned way if he wanted to.

You are given Q **queries**, which have two types:

1. **a b k** - denotes that k imps appear at **each** cell from the interval $[a, b]$
2. **a b** - asks whether Ignatius could defeat all imps in the interval $[a, b]$ in the way described above

Input:

The first line contains a single integer, N, the length of A. The following line contains N integers, $A[i]$, the initial number of imps in each cell. The third line contains a single integer Q, the number of queries. The remaining Q lines contain one query each, with a, b and k.

Output:

For each second type of query output $\mathbf{1}$ if it is possible to clear the segment, and $\mathbf{0}$ if it is not.

Constraints:

- $1 \leq \mathrm{N} \leq 200\,000$
- $1 \leq \mathrm{Q} \leq 300\,000$
- $0 \leq A[i] \leq 5000$

- $0 \leq a \leq b < N$
- $0 \leq k \leq 5000$

Example input:

```
3
2 2 2
3
2 0 2
1 1 1 1
2 0 2
```

Example output:

```
0
1
```

Explanation:

For the first query, one can easily check that it is indeed impossible to get from the first to the last cell while clearing everything. After we add 1 to the second position, we can clear the segment, for example by moving in the following way: $0 \to 1 \to 2 \to 1 \to 0 \to 1 \to 2$

Time and memory limit: 5s/64MB

Solution and analysis:

For queries of type 2, we are only interested in the elements A_k with n in the interval $[i, j]$, and nothing else. For the sake of convenience, label those elements as $a_0, a_1, \ldots, a_n$, where $n = j - i + 1$.

It can be easily seen that in order to clear a segment, we can just move back and forth between positions 0 and 1 until a_0 becomes 0, move to 1 and alternate between 1 and 2, etc – until we zero out the last element. If we want this procedure to succeed, several (in)equalities must hold:

- $a_0 \geq 1$ (since we have to decrease the first element of the segment in the first step)
- $a_1 - a_0 \geq 0$ (after following this procedure, a_1 is decreased exactly by a_0, and we end up on a_1)
- $a_2 - (a_1 - a_0 + 1) = a_2 - a_1 + a_0 - 1 \geq 0$, or, $a_2 - a_1 + a_0 \geq 1$
- $a_3 - a_2 + a_1 - a_0 \geq 0$, …
- In general, $a_m - a_{m-1} + a_{m-2} - \cdots + (-1)^m a_0$ has to be greater or equal than 0, if m is odd, and 1, if m is even

Equivalently, if we define a new sequence d', such that $d'_0 = a_0$, and $d'_m = a_m - d'_{m-1}$, these inequalities are equivalent to stating that $d'_0, d'_2, d'_4, \ldots \geq 1$ and $d'_1, d'_3, d'_5, \ldots \geq 0$.

Of course, we do not have to store the d array for each pair of indices i, j, but it is enough to calculate it once, for the entire initial array A. Then, we can calculate the appropriate values of d'_k for any interval $[i, j]$ (k is the zero-based relative position of an element inside the segment): Let $c = d_{i-1}$. Then,

- For even values of k, $d'_k = c + d_{i+k}$
- For odd values of k, $d'_k = d_{i+k} - c$

Now we only need a way to maintain the array d after updates of the form „$1\ i\ j\ v$“. Fortunately, we can see that this array is updated in a very regular way:

- Elements on even positions inside the interval $[i, j]$ are increased by v
- If $j - i$ is even, elements on positions $j + 1, j + 3, j + 5, \ldots$ are decreased by v, and elements on positions $j + 2, j + 4, j + 6, \ldots$ are increased by v

All of this can be derived from our definition of d.

Both of these queries can be efficiently implemented using a lazy-propagation segment tree, where each internal node stores two numbers, the minimal of its odd- and even-indexed leaves.

Using the approach described above, we arrive at a solution with the complexity of $O(q \log n)$.

Problem H: Dexterina's Lab

Authors

Dimitrije Erdeljan/Stefan Velja

Implementation and analysis

Rastko Suknjaja/Dimitrije Erdeljan

Dexterina and **Womandark** have been arch-rivals since they've known each other. Since both are super-intelligent teenage girls, they've always been trying to solve their disputes in a peaceful and nonviolent way. After god knows how many different challenges they've given to one another, their score is equal and they're both desperately trying to best the other in various games of wits. This time, Dexterina challenged Womandark to a game of **Nim**.

Nim is a two-player game in which players take turns removing objects from **distinct** heaps. On each turn, a player must remove **at least one** object, and may remove **any number** of objects from a single heap. The player to remove the **last** object wins.

By their agreement, the sizes of piles are selected randomly from the range $[0, X]$. Each pile's size is taken from the same probability distribution.

Womandark is coming up with a brand new and evil idea on how to thwart Dexterina's plans, so she hasn't got much spare time. She, however, offered you some tips on looking fabulous in exchange for helping her win in Nim. Your task is to tell her what is the **probability** that the **first** player to play wins, given the rules as above and assuming that both players play optimally.

Input:

The first line of input contains 2 integers: N and X. The second line contains $X + 1$ real numbers, given to 6 decimal places each: $P(0), \ldots, P(X)$.

Output:

Output a single real number, the probability that the first player wins. The answer will be judged as correct if it differs from the correct answer by at most 10^{-6}.

Constraints:

Constraints:

- $1 \leq \mathrm{N} \leq 10^9$
- $1 \leq \mathrm{X} \leq 100$

Example input:

```
2 2
0.5 0.25 0.25
```

Example output:

```
0.625
```

Explanation:

The correct answer is exactly 0.625. The checker will also accept, for example, outputs like 0.625000, 0.625001 and 0.625000000.

Time and memory limit: 0.5 s/256 MB

Solution and analysis:

Dynamic programming – $\mathcal{O}(xn^2)$

A well-known result from a game theory states that the winning player of a game of Nim is determined only by the bitwise XOR of the piles' sizes. The first player has a winning strategy if (and only if) this value is not zero.

Let P_i be the probability that a pile contains i objects. Define $d_{m,x}$ as the probability that the bitwise XOR of m random sizes is equal to x. The values of d can be expressed with the recurrence relation

$$d_{m,x} = \begin{cases} 1 \text{ if } m = 0 \wedge x = 0 \\ 0 \text{ if } m = 0 \wedge x \neq 0 \\ \sum_{i=0}^{N} d_{m-1,i} P_{i \oplus x} \quad \text{otherwise} \end{cases}$$

where N is the maximal possible value of the XOR (one less than the first power of 2 greater than the maximal pile size n) and $\oplus$ the bitwise XOR operator.

The probability that the second player wins is $d_{n,0}$. Because the first player wins if the second does not, we can calculate the values of d in $\mathcal{O}(xn^2)$ and output $1 - d_{n,0}$.

Matrix exponentiation – $\mathcal{O}(x^3 \log n)$

Let D_i be the vector $(d_{i,0} \quad d_{i,1} \quad \cdots \quad d_{i,N})^T$. The transformation that produces D_{i+1} from D_i is linear, and can therefore be expressed as $D_{i+1} = MD_i$, where M is a matrix given by $M_{i,j} = P_{i \oplus j}$.

Since the matrix multiplication is associative, $D_n = M^n D_0$ we can calculate M^n using $\mathcal{O}(\log n)$ matrix multiplications, for a total complexity to calculate D_n (which contains the solution $d_{n,0}$) of $\mathcal{O}(x^3 \log n)$.

This problem can also be solved in:

Vector exponentiation – $\boldsymbol{\mathcal{O}(x^2 \log n)}$

Faster multiplication – $\boldsymbol{\mathcal{O}(x \log x \log n)}$

Even faster multiplication – $\boldsymbol{\mathcal{O}(x \log x + x \log n)}$

But $\boldsymbol{\mathcal{O}(x^3 \log n)}$ is enough to get AC.

Problem I: R3D3's summer adventure

Authors — ***Ibragim Ismailov/Stefan Velja***

Implementation and analysis — ***Ibragim Ismailov/Aleksandar Kiridžić***

$R3D3$ spent some time on an **internship in MDCS**. After earning enough money, he decided to go on a holiday somewhere far, far away. He enjoyed sun tanning, drinking alcohol-free cocktails and going to concerts of popular local bands. While listening to "The White Buttons" and their hit song "Dacan the Baker", he met another robot for whom he was sure was the love of his life. Well, his summer, at least.

Anyway, $R3D3$ was too shy to approach his potential soulmate, so he decided to write her a love letter. However, he stumbled upon a problem. Due to a terrorist threat, the Intergalactic Space Police was monitoring all letters sent in the area. Thus, R3D3 decided to **invent his own alphabet**, for which he was sure his love would be able to decipher.

There are N letters in $R3D3$'s alphabet, and he wants to represent each **letter** as a sequence of $\mathbf{0}s$ **and** $\mathbf{1}s$, so that **no** letter's sequence is a **prefix of another one's sequence**. Since the Intergalactic Space Communications Service has lately introduced a tax for invented alphabets, R3D3 must pay **a certain amount of money** for **each bit** in his alphabet's code. He is too lovestruck to think clearly, so he asked you for help.

Given the costs C_0 and C_1 for each 0 and 1 in $R3D3$'s alphabet, respectively, you should come up with a coding for the alphabet (with properties as above) with **minimal total cost**.

Input:

The first line of input contains 3 integers:

- N - the number of letters in the alphabet
- C_0 - cost of 0s
- C_1 - cost of 1s

Output:

Output a single number - the minimal cost of the whole alphabet.

Constraints:

- $2 \leq \mathrm{N} \leq 10^8$
- $0 \leq C_0 \leq 10^8$
- $0 \leq C_1 \leq 10^8$

Example input:

```
4 1 2
```

Example output:

```
12
```

Explanation:

The alphabet is "00", "01", "10", "11". So minimal total cost is 12.

Time and memory limit: 1 sec/256 MB

Solution and analysis:

Basically the problem can be formulated in the following way: Let's build a binary tree with edges labeled '0' and '1'with N leaves so that the sum of costs of paths to all leaves is minimal. This tree is also called 'Varn code tree'*. Varn code tree is generated as follows. Start with a tree consisting of a root node from which descend 2 leaf nodes, the costs associated with the corresponding code symbols. Select the lowest cost node, let c be its cost, and let descend from it 2 leaf nodes $c + c(0)$ and $c + c(1)$. Continue, by selecting the lowest cost node from the new tree, until N leaf nodes have been created.

This greedy approach can be done in $O(NlogN)$ if we actually construct a tree. Basically we do the following thing N times: out of all codes we have selected the lowest, deleted it and created 2 new codes by adding '0' and '1' to the one we have deleted. If all of this was done with some standard data structure this is $O(NlogN)$.

If we actually don't need the tree and the coding itself, just the final cost, we can improve to $O(log^2(N))$. Let's define an array F to represent this tree, where $F[i]$ will be a number of paths to the leaves in this tree with cost equal to i. While sum of all values in F is lower than N we do the following procedure: find an element with lowest 'i' where $F[i] > 0$. Then, $F[i + c(0)] += F[i]$; $F[i + c(1)] += F[i]$; $F[i] = 0$. Basically we did the same thing as in the previous algorithm, just instead of adding 2 edges to the leaf with the lowest cost we did that to all of the leafs that have the lowest cost in the tree. Numbers in this array rise at least as fast as Fibonacci numbers, just the array will be sparse. So if instead of array we use some data structure like a map, we get $O(log^2(N))$ complexity.

Qualifications

Continuing the well-known format from previous years, the qualifications were split into two rounds, with ten problems in each round. Although the finals were in the traditional format with no open problems, the problem set in qualifications contained challenge problems (the problems that do not have a known "best" solution). Solved non-challenge problems were worth 1 point in the first round and 2 points in the second round. Challenge problem in the first round was worth maximum of 4 points, while challenge problem in the second round was worth maximum of 8 points.

The problems for both rounds were chosen from the publicly available archives at the SPOJ site (http://www.spoj.com/)

NUM	Problem name	Solved
1	Easy Password	29 times
2	Missing Side	54 times
3	Taklu Kuddus	60 times
4	Yossy, The King of IRYUDAT	34 times
5	Robot	42 times
6	St Bernard and Gravity	47 times
7	Help BTW	58 times
8	Cat and Mouse I	39 times
9	I Hate Parenthesis	57 times
10	[CH] Automatic Brainf##k Code Generator (Shortening AI)	55 times

Table 1. Statistic for Round 1

NUM	Problem name	Solved
1	Special Numbers (Reverse and Add)	13 times
2	World Record Lunch	16 times
3	Eclipse	7 times
4	Travelling Santa	35 times
5	When (You Believe)	15 times
6	Lights (Extreme)	31 times
7	K Edge-disjoint Branchings	3 times
8	Swap (Hard – Level 1000)	1 time
9	After Party Transfers	19 times
10	[CH] God Number is 20 (Rubik)	20 times

Table 2. Statistic for Round 2

The tradition of contestants providing solutions for the qualification problems continued this year as well. Please note, that these solutions are not official – and we cannot guarantee that all of them are accurate in general. (Still, a correct implementation of the solution should make all of the test cases pass on the SPOJ site at the time these problems were available to the contestants.)

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

Problem R1 01: Easy password

Time Limit: 0.5 second

Memory Limit: 1536 MB

This is the year 2100. Pranjali ma'am due to her intelligence survived this far, by her "Secret Potion" to freeze her age. She keeps her lotion in a very secret vault which is electronic password protected. This is a very special system designed by our great Sameer sir, which don't take any specific string as a password. Rather than it takes a "Tree", yes you heard that right a "Tree" of numbers as a password. Your password should be exact to the tree both in structure and value to unlock the vault. Its high time and many hackers are trying to hack the system. So Pranjali ma'am was bit worried and wanted to change the password. Some of the new passwords are very vulnerable so Sameer sir told not to use them. You have to make the **new password tree** for Pranjali ma'am to keep her "Secret Potion" safe. You are given **old password tree**, and the one **restricted password tree** provided by Sameer sir. The structure and value of both might not be same as Sameer sir don't know the exact password, he only knows the current hacking pattern of the hackers.

You have to make a **new password tree** with same structure as **old password tree**, with the nodes of **old password tree** permuted in some order and any new node should not be exactly equal in position and value to **restricted password tree**. The position of a node is defined by the sequence of moves (Left / Right) required to go down from the root to that node.
If there are multiple answers, print the one whose difference is minimum from the **old password tree**.
Calculating difference between trees:
$Diff = |sum((old[node[i]] - new[node[i]]) * 10^i)|$ where i is $n-1$ to 0. The ordering of nodes is based on level order traversal of trees. i.e. i will be $n-1$ for root node and so on.

Input
First line contains t, number of test cases. In each test case first line contain n and m, n is the number of nodes of **old password tree** and m is the number of nodes of **restricted password tree**. Next line contains n space separated numbers (value of nodes of **old password tree**). Next n lines contain expression like A B, in i^{th} line A and B is the left and right child of node with index i(0 based). If there is no right or left child of a node -1 will be provided in place of any index.
Next line contains m space separated numbers (value of nodes of **restricted password tree**). Then this tree is also described as the tree above.

Output
One integer. Minimum diff as per the above formula between **old tree** and **new tree** OR -1 if no such tree is possible.

Contraints
$1 <= T <= 50$
$0 <= node\_value <= 9$
$1 <= n <= 18$

Note:
1. New tree can be same as old tree.
2. If no such tree is possible print -1
3. All trees in the problem are Binary Trees

Example

Input	Output
1 3 3 5 4 8 1 2 -1 -1 -1 -1 5 1 8 1 2 -1 -1 -1 -1	63

Explanation:

According to the rule specified above the most feasible tree is rooted at node with value 4 and it's left child being 8 and right child being 5.

So the minimum diff is 63.

Solution:

Easy password is this year's Round 1 hardest problem according to the number of correct solutions.
Here we present a polynomial-time algorithm, running in $O(n^4)$.
We will begin by introducing the Hungarian algorithm (https://en.wikipedia.org/wiki/Hungarian_algorithm): Suppose we have a square matrix $n \times n$, and a number written in every cell. The set of cells is called good if there is exactly one chosen cell in every row and column. Hungarian algorithm finds good set of cells that minimizes the sum of numbers written in them in $O(n^3)$.
Equivalently, in weighted bipartite graph, Hungarian algorithm gives us the perfect matching with minimum total weight.
Detailed explanation of the algorithm and its implementation can be found at https://www.topcoder.com/community/data-science/data-science-tutorials/assignment-problem-and-hungarian-algorithm/ .
Now, we proceed to the actual problem. Generally speaking, our task is to permute the vertices of the

old tree and in that way make a new tree that minimizes the total cost of the operation. By the cost of sending vertex v to the vertex u we mean ∞, if that matching is forbidden by the restricted password tree, and $|(old[v] - new[v])| * weight[v]|$ if the matching is not forbidden, where $weight[v]$ represents the exponent 10^i in the statement.
So, firstly we form the $weight[]$ array, which equals 10^{n-1} for the root, and the exponent of 10 diminishes in the level order traversal of the tree. This can be done by elementary techniques and algorithms on the tree.
Next, "cost matrix" is computed, where $cost[i][j] = \begin{cases} \infty, & if\ old[i] = restricted[j] \\ (old[i] - old[j]) * weight[j], & else \end{cases}$
Before proceeding any further, it is necessary to check if there is any possible new tree, and this can be done by running the Hungarian algorithm on the matrix defined by $M[i][j] = |cost[i][j]|$. If the result is less than ∞, we have at least one matching and we will try to find the minimum one, else we output -1 and pass to the next test case. We need this check because in the rest of algorithm, we will assume that the total cost is less than ∞, so no cell filled with ∞ can ever be included.
At this point, we need to find the matching of the vertices of the old and the new tree that minimizes the absolute sum of costs.
As we can't just run Hungarian algorithm (we need a minimal absolute sum, not just minimal sum), we see that the cost of transferring any vertex to the root makes the biggest part in our sum, which is definitely greater than the sum of all others ($10^{n-1} > 9 * 10^{n-2} + \cdots + 9 * 10^0 = 10^{n-1} - 1$), so by our choice of the root, the sign of the sum is determined, and we can just proceed to minimizing the rest of the sum in the case that $cost[v][root] > 0$, and maximizing it otherwise. So, we will try out each possible new root, and among them choose our minimum. Hungarian algorithm does this minimization, and in case we want the maximization, we multiply whole cost matrix by -1, and minimize that (keeping in mind that every ∞ remains ∞ and does not become $-\infty$), multiplying the result at the end by -1, to get wanted maximum.

We have now almost solved the problem, but a small issue arises: what if, in the beginning, the cost of taking the vertex to the root was 0? Then, we would not have our sign determined yet. But, if we think about this for a while, we conclude that this is very good for us because in that way we minimized the sum (remember that the weight of the root was the biggest addend in the sum, much greater than the sum of all others).
So, we try again and again to match the next biggest vertex with the cost 0.
But, each time we transfer something to the root with the cost 0, we must check that we can still finish the tree(maybe the restricted tree is restricted in that measure that it doesn't allow us to match the remaining vertices in any way). So, we will run Hungarian algorithm on the matrix $M[i][j] = |cost[i][j]|$, and as long as the cost of the highest-weight vertex is 0, match it, and proceed, eliminating it from further consideration. Once we get to some vertex that can't be zero-matched, we will try out every possible (not restricted and not eliminated) vertex, and do the procedure described above.
Let's look for the proof of correctness of this approach, i.e. why does this procedure really give us minimal difference between the trees.
Obviously, for every possible result given by the algorithm, the new tree is legal, because no ∞ cells were included (we supposed that the sum is less than ∞). As explained above, each time we do a zero-match to the highest weight place, we make the number of digits of the final sum smaller, and so make the total possible sum smaller. After that, as we check each possibility for the next biggest weight vertex, and Hungarian algorithm minimizes the rest for us, the minimum is surely to be found.

The overall complexity is $O(n^4)$, where n is the number of nodes, because computing weight and cost arrays take us just $O(n^2)$, the first try to find any matching is $O(n^3)$ and matching vertices to the root at cost 0 takes us also $O(n^3)$, and the last part, trying out every root and doing Hungarian for each is $O(n * n^3) = O(n^4)$, which is the overall complexity of our algorithm.

Added by: dce coders
Solution by:

Name: ***Aleksa Milojević***
E-mail: akimil2000@gmail.com

Problem R1 02: Missing Side Solution

Time Limit: 0.5 second

Memory Limit: 1536 MB

It is well known that every triangle has a unique incircle and a unique circumcircle. Given the lengths of two sides of a triangle, determine the length of the missing side, such that the area between incircle and circumcircle is as small as possible.

Input

Input starts with a positive integer $t \approx 10000$ in a single line, then t lines follow. Each of the t lines contain the lengths of two sides of a triangle, separated by a single space. All values are given with four digits after the decimal point and are not larger than 100.

Output

For every test case print the length of the missing side that satisfies the condition given above, rounded to four digits after the decimal point. If there is no unique solution, print "ambiguous" (without quotes) instead.

Example

Input	Output
2 1.8491 3.5678 7.5000 5.1480	3.1416 6.0000

Solution:

Let A and B be the given side lengths, and let us assume without loss of generality that $A \geq B$. Because of triangle inequalities, the unknown side length X has to be in the interval $(A - B, A + B)$. To find areas of the inscribed and circumscribed circles we will use Heron's formula which says that the area of a triangle is

$$P = \sqrt{s(s-a)(s-b)(s-c)},$$

where a, b and c are the side lengths, and $s = (a + b + c)/2$. We will also use the two other well-known formulas for the area of a triangle given the radius R of the circumscribed circle, and the radius r of the inscribed circle, which go as follows:

$$P = rs,$$

and

$$P = \frac{abc}{4R}.$$

Using these formulas, the area between the circumcircle and the inscribed circle that we want to minimize is:

$$a(X) = \pi\left(\frac{A^2B^2X^2}{h(X)} - \frac{h(X)}{4(A+B+X)^2}\right),$$

where

$$h(x) = (A+B+X)(A+B-X)(A+X-B)(B+X-A).$$

The analysis of the function $a(X)$ is not easy, but through excessive testing with numerous different parameters A and B we see that this function is unimodal with one minimum in the interval $(A-B, A+B)$.

Using this property, we can use ternary search to find the desired minimum. Given that there are less than $N = 100000$ possible solutions, the final complexity of our algorithm is $O(t \log N)$ which easily fits in the given time limit.

Added by: numerix
Resource: *own problem*
Solution by:

Name: ***Luka Vukelić***
E-mail: lviskovitz@gmail.com

Problem R1 03: Taklu Kuddus

Time Limit: 0.5 second

Memory Limit: 1536 MB

Kuddus is a great programmer. He recently solved 100 problems in different OJ's. One day his girlfriend gave him a problem. But he failed to solve that problem and his girlfriend became very angry.
For this reason, his girlfriend doesn't talk to him.
He is losing his hair by thinking of how he can solve the problem. Now Kuddus came to you for help. As his friend, you have to help Kuddus and save his relation and hair also.

The problem is, you are given a string "S" and a pattern "P". You have to find $FS\ (x, y)$ that is defined as the maximum number of non-overlapping substring which is equal to the pattern "P" in the substring of S which starts at x and end at y (x , y are in 0 base indexes).
Suppose,

$S\ =\ "abcdef"$

$P\ =\ "cd"$

and the query is (1,5), so the substring will be "$bcdef$" and $FS(1,5)\ =\ 1$

Input

First line contains an integer T (number of test cases) ($1 <= T <= 10$)
Each case starts with a line containing string **S**, |S| <= 1000000. The next line contains string P**,** $|P|\ <= 1000000$**.**

Then an integer $q(1 <= q <= 100000)$ **,** Each of the next **q** lines will contain a query which is in the form $i\ j$

($0 \leq\ i\ \leq\ j\ <\ length(S)$).

Output

For each test case, print the case number in a single line.
Then for each query you have to print a line, value of $FS(i.j)$;
It is guaranteed that the summation of all the queries for each test case will not exceed 200000

Example

Input	Output
1 abababc ab 3 0 6 1 2 0 4	Case 1: 3 0 2

Solution:

Two strings S and P with up to 10^6 characters are given. We need to answer $Q <= 10^5$ queries of the form:
„For a given substring of S, at most how many non-overlapping instances of P can we find in it? “

Let's try to find a correct solution and then make it fast enough to fit under the time limit.

The greedy algorithm of taking occurrences of P from left to right is easily shown to produce the optimal answer for each query. Given a substring T, we check for the first $|T| - |P|$ positions of T whether an occurrence of P begins there, and then try to fit P's greedily, using $O(|T| * |P|)$ operations. This is nowhere near to passing the time limit, because we have to do that Q times, resulting in worst-case complexity $O(Q * |S| * |P|)$.

Occurrences of P in a substring of S are a subset of the occurrences of P in the whole S. As the number of queries is big, we will probably not be able to find occurences of P for each given substring. Instead, we will precompute them for the whole string S and then try to use the precomputed information to answer the queries.

We want to get an indicator function of whether a given position of S is the starting point of an occurrence of P. To accomplish that, we will use the Z algorithm on the string $P + $ „$“ $+ S$ in the usual fashion, which is $O(|S| + |P|)$.

Using now the precomputed occurrences of P in, the running time of the first solution becomes $O(Q * |S|)$, because checking whether an occurrence of P starts somewhere, can now be done in $O(1)$. Unfortunately, this is still not fast enough.

Noting that the answer to a query is binary searchable (if we can fit X occurrences of P, we can also fit $X - 1$), we will now consider questions of the following type:

„For a position i of S, where does the shortest string containing K non-overlapping instances of P beginning at i end? “

If we can binary search on K, we can answer every query comparing the endpoint of the substring with the answer to the questions. Unfortunately, we will have to modify our approach because answering to the questions cannot be done fast enough.

Denoting the answer to the question as (i, K), we can easily see that (bar the corner cases) the following relation holds:

$$(i, K + L) = ((i, K) + 1, L)$$

In other words, to take $K + L$ occurrences of P greedily, we can first take K and then take L from the first character not taken.

That enables us to mimic binary search: we will answer only the questions of the type $(i, 2^k)$ and jump indices using the relation. Now, there are $O(|S| * log|S|)$ possible questions $(i, 2^k)$, and we can easily precompute them using the relation and dynamic programming in $O(|S| * log|S|)$, where answers to $(i, 1)$ are computed using the Z algorithm.

Finally, only one maximal power of 2 fits in any number (because $2^k + 2^k = 2^{k+1}$), so we can traverse powers of 2 from the biggest possible to 2^0, jumping indices inside the loop and keeping track of how many occurrences of P we have put in the interval. The running time for each query is $O(log\ |S|)$, bringing the total running time to $O(\ (|S|\ +\ Q) * log\ |S|)$.

Added by: Raihat Zaman Neloy
Solution by:

Name: ***Daniel Paleka***
E-mail: danepale@gmail.com

Problem R1 04: Yossy, The King of IRYUDAT

Time Limit: 0.100s-0.150 second

Memory Limit: 1536 MB

Yossy is the king of IRYUDAT kingdom. The kingdom lives peacefully because Yossy is kind and wise. One day, the enemy wants to launch an attack on IRYUDAT. So, the enemy prepares a strong army to crush IRYUDAT kingdom.

Yossy knows about this. He must protect his kingdom. The most logical way to prevent this is to send an army that is at least as strong as the enemy's army. An army's power is the sum of each soldier's power. For example, if there are three soldiers with 10, 20, and 25 power respectively, the army's power is 55.

Yossy gathers all of his soldiers. To make a powerful army, he can simply choose soldiers with high power or just send them all. However, this can't be done because the soldiers need to eat. Each soldier has to eat a certain amount of food or they can't go to war. Unfortunately, the kingdom is currently lacking food supply. Yossy needs to choose soldiers wisely so he can make a powerful army with the food limitation.

Another problem occurs. Yossy doesn't know the enemy's power. So, he sends his underling to calculate the enemy's power. He wants to have a list of his possible army's powers, sorted from the best to the worst, so that after he knows the enemy's power, he can send an army that is at least as strong as the enemy's. Two armies are different if the combination of the soldiers is different.

Input

First line contains an integer $n(1 < n < 2000)$, the number of soldiers that Yossy has. The next n lines contain $p(0 < p < 9999)$ and $f(0 < f < 9999)$, the power and food portion of each soldier. The next line contains $s(0 < s < 999999)$, the maximum food supply. The last line contains $k(1 < k < 40)$, the length of Yossy's list. If no more combination of soldiers are available, output 0 until the end of the list.

For test data with $n > 15$, the p and f value will be randomly generated with no correlation.

Output

k lines containing the army's power from the first best to k-best army.

Example

Input	Output
4 45 3 30 5 45 9 10 5 15 4	90 85 75 75

Explanation

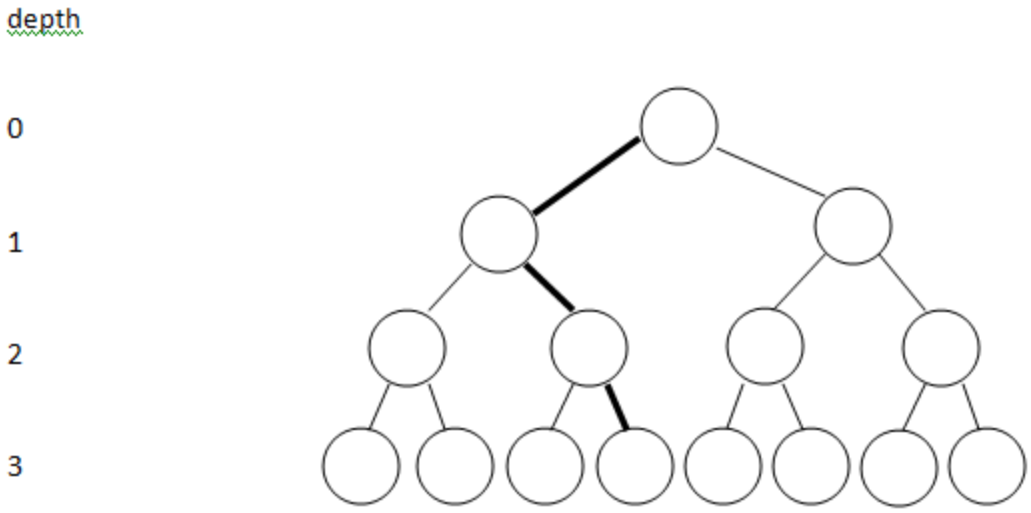
The 1st best army has power 90. It can be obtained from soldier 1 and 3, with sum of power 90 and sum of food portion 12.
The 2nd best army has power 85. It can be obtained from soldier 1, 2, and 4 with sum of power 85 and su m of food portion 13.
The 3rd best army has power 75. It can be obtained from soldier 1 and 2, with sum of power 75 and sum of food portion 8.
The 4th best army has power 75. It can be obtained from soldier 2 and 3, with sum of power 75 and sum of food portion 14.
Note that the 3rd and 4th best army have same power but the combination of soldiers is different.

Solution:

This problem is a variation of a problem known as $0/1$ knapsack problem. Our problem is different because we need to find not only the best solution but the k $(1 <= k <= 40)$ best solutions.

First approach that comes from the top of our head is brute-force approach. It means that we should try all possible combinations of soldiers, so if there are n soldiers, then there are 2^n possibilities which is too many because n can go up to 2000.

So we have to come up with a different approach. My solution uses branch and bound algorithm (https://en.wikipedia.org/wiki/Branch_and_bound) which is mostly used with optimization problems, as the one we are dealing with. It uses a state space tree, in this case a complete binary tree with a depth n (root having depth 0, leaves having depth n). Each path leading from the root to one of the leaves represents one subset of soldiers. If we are currently at one of the nodes with depth i and we go to its left child that means we choose soldier $i + 1$, or if we go to its right child it means that we didn't chose soldier $i + 1$.



Marked path means that we choose soldier 1 but we didn't chose soldiers 2 and 3.

So we make a state space tree (power of each soldier will be its value and food portion will be its weight). In each node we store current value, current weight and upper bound. What's an upper bound? It is a value for which we can be sure that it will be higher than the total power of soldiers if we go through this node. How we determine this value? We sort all soldiers by value f (from the largest to the smallest) which is value/weight. If we pay attention to our sample test case, we have soldiers with next attributes:

Soldier	Value	Weight	f
1	45	3	15
2	30	5	6
3	45	9	5
4	10	5	2

so they are sorted by f.

So if we want to determine the upper bound for some node we use the following algorithm:

1. $upper\ bound = current\ value$

2. $temporary\ weight = current\ weight$

3. if we can take the next soldier without going over maximum food limit $upper\ bound = upper\ bound\ +\ next\ soldier's\ value$

$temporary\ weight = temporary\ weight\ +\ next\ soldier's\ weight$

Repeat 3.

4. if we can't take the next soldier, the best we can do is to take a fraction of that soldier (we cannot do this in the task, just to determine the upper bound)

$$upper\ bound = upper\ bound\ +\ ((maximum\_food\ \_limit - temporary\_weight)/next\_soldier's\_weight) * next\_soldier's\_value$$

For example, upper bound for the root is $45 + 30 + 45 * 7/9 = 110$.

Now it gets simpler. We start building the state space tree from the root. Then we determine upper bounds for its children and greedily go to the child that has higher upper bound hoping it will produce better result. Now we continue building a tree always going to the node which has the highest upper bound. When we come to one of the leaves we check if it's better than one of our current k solutions and if it is, we update our k-best solutions. The moment when we are left only with nodes which have their upper bound values lower than our current k-best solution is the moment when we stop building a tree and write current k solutions.

Added by: Andy
Solution by:

Name: ***Srđan Marković***
E-mail: srdjnmrkvc@gmail.com

Problem R1 05: Robot

Time Limit: 1 second

Memory Limit: 1536 MB

There is a robot on the $2D$ plane. Robot initially standing on the position $(0, 0)$. Robot can make four different moves:

1. Up (from (x, y) to $(x, y + 1)$) with probability U.
2. Right (from (x, y) to $(x + 1, y)$) with probability R.
3. Down (from (x, y) to $(x, y - 1)$) with probability D.
4. Left (from (x, y) to $(x - 1, y)$) with probability L.

After moving N times Robot gets points.

- Let x_1 be the smallest coordinate in X-axis, that Robot reached in some moment.
- Let x_2 be the largest coordinate in X-axis, that Robot reached in some moment.
- Let y_1 be the smallest coordinate in Y-axis, that Robot reached in some moment.
- Let y_2 be the largest coordinate in Y-axis, that Robot reached in some moment.

Points achieved by Robot equals to $x_2 - x_1 + y_2 - y_1$.

Given N, U, R, D, L. Calculate expected value of points that Robot achieved after N moves.

Input

First line: One integer N $(1 \leq N \leq 200)$.

Second line: Four real numbers U, R, D, L $(U + R + D + L = 1, 0 \leq U, R, D, L \leq 1)$ with maximum of 6 numbers after dot.

Output

One number: expected value of points achieved by Robot. The answer will be considered correct if its relative or absolute error does not exceed 10^{-6}.

Example 1

Input	Output
2 0.100000 0.200000 0.300000 0.400000	1.780000

Example 2

Input	Output
3 0.25 0.25 0.25 0.25	2.375000

Solution:

Let $X = x_2 - x_1 + y_2 - y_1$ denote the number of points the robot achieved after N moves. Due to linearity of the expectation operator, we can write $E[X] = E[x_2] - E[x_1] + E[y_2] - E[y_1]$. Let's see how to compute $E[x_2]$, i.e., the expected largest x-coordinate that the robot has reached in some moment.

To do so, we will use dynamic programming. More precisely, let's define $dp(n, x, best_x)$ as the expected largest x-coordinate given that the robot has thus far made n steps, stands on position x and the largest x-coordinate it has been on is $best_x$. From this state, the robot can move to position $(x + 1)$ with probability R, to $(x - 1)$ with probability L or it can move vertically with probability $(U + D)$. Therefore, the following relation holds:

$$dp(n, x, best_x) = R * dp(n + 1, x + 1, \max(best_x, x + 1))$$
$$+L * dp(n + 1, x - 1, best_x)$$
$$+(U + D) * dp(n + 1, x, best_x)$$

Naturally, $dp(N, x, best_x) = best_x$ and $E[x_2] = dp(0,0,0)$. Computation of $E[x_1], E[y_2]$ and $E[y_1]$ is analogous. The time complexity of the algorithm is therefore $O(N^3)$ which is sufficient to pass the test data.

Added by: Bartek
Solution by:

Name: ***Ivan Paljak***
E-mail: ipaljak@gmail.com

Problem R1 06: St Bernard and Gravity

Time Limit: 1 second

Memory Limit: 1536 MB

St. Bernard's new game is played on an $R \times C$ board. Initially every square is either empty or blocked by a wall. Bernard throws a rock into the board by putting it in the topmost row of a column and then letting gravity do the rest.

Gravity works as follows:
=> If the square under the rock is a wall or if the rock is in the bottom row of a column, then the rock remains there.
=> If the square under the rock is empty, then the rock moves to that square.
=> If the square under the rock contains another rock, then the falling rock may slide sideways :
- o If the squares left and left-down of the rock are empty, then the rock slides one square left.
- o If the rock doesn't slide left and the squares to the right and right-down are empty, then the rock slides one square right.
- o Otherwise, the rock remains there and never moves again.

Bernard will never throw another rock while the previous rock hasn't settled down.
Write a program that draws the board after Bernard throws all his rocks into the board, if we know the columns that Bernard threw his rocks into, in order.
Note: Bernard will never throw a rock into column in which the top row isn't empty.

INPUT

The first line contains integers R and C $(1 <= R <= 30000, 1 <= C <= 30)$, the size of the board.
Each of the following R lines contains C characters, the initial layout of the board. A '.' represents an empty field, while the uppercase letter 'X' is a square blocked by a wall.
The next line contains an integer N $(1 <= N <= 100000)$, the number of rocks Bernard throws.
Each of the following N lines contain an integer between 1 and C, the column into which Bernard throws a rock (the leftmost column is column 1).

OUTPUT

Output R lines, each containing C characters, the final layout of the board. Rocks should be presented with an uppercase letter 'O'.

Sample

Input	Output
5 4 .... X... .... 4 1 1 1 1	.... O... X... .... OOO.
Input	**Output**
7 6 ....XX. .... .XX... ...... 6 1 4 4 6 4 4	....O.. ...XX. ...... .OO... .XX... O..O.O

In the first example, all rocks are thrown in the first column. The first rock stops on the wall. The second rock falls on the first, slides right and stops at the bottom of the second column. The third rock falls on the first then on the second rock, slides left and rests at the bottom of the first column. The fourth rock falls on the first then on the second, then slides right.

Solution:

Let's devise some conclusions directly from the statement: there are three possible moves a rock can make at any point during its journey: down, right-down and left-down. Even though left-down is defined as left+down**,** we will treat it as one move for convenience. This is valid, since if the rock ever moves to the side it must go down in the next turn. The same applies to right-down, by analogy. Using this formulation of possible moves, we can, for a given initial layout, precompute the full paths for each

column in $O(RC)$, by simulating gravity. Using this, the terminal position of the first rock which Bernard throws can be found in $O(1)$; it's simply the last cell on the precomputed path for the corresponding column. Obviously, precomputing all the paths again for the next rock is too costly, so we need a fast way to update the paths. We can derive the update algorithm by noticing that the only paths that actually change after a new rock settles at cell X, are the ones that had X as their last cell. If X is not in the path then it can't change it, and if X is in the path it has to be the last cell (otherwise the rock wouldn't stop there). Now that we've noticed this, we can update all the affected paths by doing a rollback: deleting X from each path and then simulating the gravity from the penultimate cell. The total amount of work we will do per column is at most $O(R + N)$ ($R\ rows\ +\ N\ rollbacks$), which yields the total time complexity of $O((R + N) * C)$.

Added by: BLANKRK
Solution by:

Name: ***Nikola Jovanović***
School: Faculty of Computing, Union University
E-mail: njov996@gmail.com

Problem R1 07: Help BTW

Time Limit: 2 second

Memory Limit: 1536 MB

BTW wants to buy a gift for her BF and plans to buy an integer array. Generally, Integer arrays are costly and hence bought the cheapest one from the market. Her BF is very judgmental and assess the quality of the array by just looking at the **smallest element** in the array. Hence, she decided to improve the quality of the array. Increment operation on array elements are very costly and would take **ONE FULL DAY** to complete. Also, Increment operations can be done in parallel on **at most** M **consecutive** array elements. She only has K days left. Help BTW calculate the **maximum possible "quality"** of the array she can attain.

(BTW BTW is the name of the character)

Input

Very first line contains T – (number of test cases)

First line in each test case contains N – (size of the array BTW has bought from the market), M, K (days left)

Second line in each test case contains N integers (values of the initial array BTW bought from the market)

Output

Print the maximum possible "quality" she can attain.

Constraints

$1 <= T <= 100$

$1 <= N <= 10^5$

$0 <= M <= N$

$0 <= K <= 10^9$

$0 <= Values\ of\ array <= 10^9$

Example

Sample test case 1

Input	Output
3 2 1 2 2 3	3

Sample test case 2

Input	Output
3 2 1 2 3 2	2

Solution:

This was one of the easiest problems in this year's Bubble Cup qualification rounds.

The problem solution (the maximum smallest element in the array) can be binary searched, because if the smallest element can be incremented to x, it can be increased to $x - 1$ in less days.

We can determine if the smallest element can be increased to x by the following greedy algorithm:

it is obvious that we will always want to increment the maximum number of elements (at most m).

Therefore, if we iterate from left to right, only one interval can be chosen for incrementing the leftmost element of the array. If the first element is smaller than x, we will increment the first m elements of the array by x – (first element), otherwise we do nothing.

Now we can move on to the element to its right and repeat the described procedure until we get to the right end of the array. If the total number of days needed is less than k or equal to it, the

smallest element in the array can be at least x, otherwise it must be less than x.

The final complexity is $O(n * log(10^9))$.

Added by: Noob
Solution by:

Name: ***Nikola Herceg***
E-mail : nikolaherceghr@gmail.com

Problem R1 08: Cat and Mouse I

Time Limit: 1 second

Memory Limit: 1536 MB

A cat is chasing a mouse in a rectangular room which contains some obstacles. The cat and mouse move according to the following rules:

1. The cat and mouse move in turns, in alternating fashion. The cat starts.
2. Both the cat and the mouse can only move to a location that does not contain an obstacle, and that is horizontally, vertically, or diagonally adjacent to their current location. Staying in the same location is always a valid move.
3. Neither the cat nor the mouse can move beyond the border of the room.
4. The cat is always aware of the location of the mouse, and vice versa.

Given the description of the room, and the starting positions of the cat and of the mouse, your task is to decide whether the mouse can avoid the cat forever or not.

Input

The first line of the input contains T $(1 \leq \mathrm{T} \leq 10)$, the number of test cases. Then T test cases follow, separated by blank lines. Each test case consists of a line describing the length of the two sides of the room, R and C $(1 \leq \mathrm{R} \leq 10,\ 1 \leq \mathrm{C} \leq 10)$, followed by a description of the room. The room is given as a matrix of R rows and C columns, where each entry of the matrix is either:

- a dot (.), if the corresponding location is free of obstacles,
- a hash sign (#), if the corresponding location contains an obstacle,
- an m, if the corresponding location is the starting position of the mouse,
- a c, if the corresponding location is the starting position of the cat.

The starting position of the cat and the starting position of the mouse are always different. Clearly, the starting positions are free of obstacles.

Output

The output consists of one line for each test case, containing the string mouse if the mouse can avoid the cat forever, and the string cat otherwise.

Example

Input	Output
2 3 3 #c# .#m #.# 3 3 #c# .#. #m#	cat mouse

Solution:

The problem in this task is to figure out whether the mouse wins. If the cat and the mouse are not connected, then the mouse wins, otherwise there are 2 scenarios.

First scenario: when in the area where the mouse and the cat are doesn't contain at least one island of obstacles.

a. if the distance from the mouse to any island is two steps shorter than the cat's distance from the same island (because the cat starts first), the mouse wins;

b. otherwise the cat wins.

Second scenario: when there are no islands in the area where the cat and the mouse are. In that case, the cat wins.

Algorithm can be done using $3\ BFS$.

1. finding islands
2. finding out whether the cat and the mouse are connected
3. finding distances from the cat and the mouse to any island.

Added by: Vincenzo Bonifaci
Solution by:

Name: ***Viktor Lucic***
E-mail : stefanbalaz2@gmail.com

Problem R1 09: I Hate Parenthesis

Time Limit: 1s-2 second

Memory Limit: 1536 MB

"I Hate Parenthesis", it was the phrase that Ana said during the class of arithmetic. She wants a simple code made for yourself (of course) to "kick out" the parenthesis from particular expressions without break maths.

Input

The first line contains the number of test cases $\boldsymbol{T}$. One expression per line from second to $(\boldsymbol{T}+\mathbf{1})^{\text{th}}$ line. Each expression contains only uppercase letters (each letter can appear once), operators $(+,*)$ and parentheses $(())$. There are no spaces inside the expressions and no input line contains more than 80 characters.

Output

A single line per case. The line must contain the expression without parenthesis. If there is more than one addend, then you must sort first the one with more factors. Addends with the same amount of factors must be sorted in alphabetic order. See the input/output for more details.

Example

Input	Output
2 C*(A+B) (A+B+D*E)*C	A*C+B*C C*D*E+A*C+B*C

Solution:

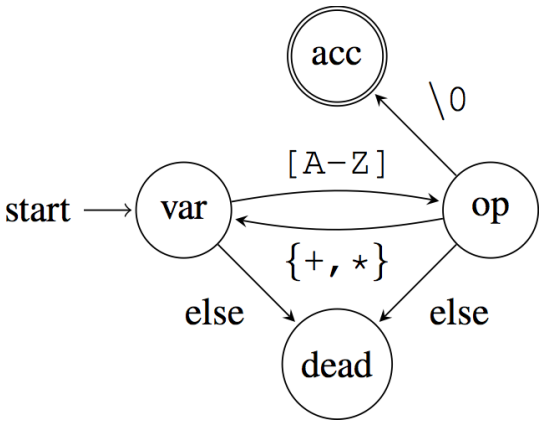
As its name suggests, this problem requires you to expand out all the parentheses within a mathematical expression containing variables, addition, multiplication and parentheses, without changing the semantics of the expression. This requires us to do two tasks:

1. Representing the expression as a data structure which maintains all the necessary semantics and is simple to work with for expansions;
2. Actually performing all the expansions on the constructed structure.

The first task represents a simple problem if you have encountered similar issues before (http://www.spoj.com/problems/ONP/ is a very popular example), however, I believe that typical implementations of such algorithms tend to be very hand wavy in their description. Therefore, instead of directly discussing the algorithm implementation, I will make a more formal approach to the topic.

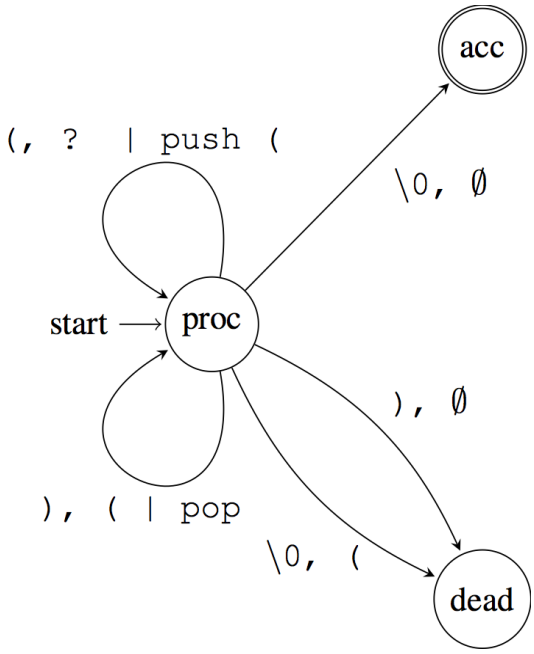
Namely, let's first discuss a simpler problem: the problem of **verifying** whether a given string is a mathematically valid statement with parentheses. If we're unable to verify that the string is a valid input, then we're also unable to properly process it - therefore our main task is at least difficult. In fact, the product of our approach to the verification problem will give us a desirable data structure for representing these expressions as a consequence.

First of all, what if there were no parentheses? The verification then becomes really simple, especially considering that all variables have to be single-character---we can simply make a single pass over the string, and make sure that we get an alternating sequence of letters and operators. We can represent this process as a **finite automaton** (as depicted by the figure on the RHS); at each step it consumes a single character and changes state; if we're able to make it to the end of the string without reaching the dead state, then the string is a valid expression without parentheses. Note that such a construction does not accept an empty string or a string starting with a + (which is not allowed by the task). An important thing to note about finite automata is that at any stage they're only able to "see" their current state and the next character in the string to consume; i.e. they are **memoryless**.



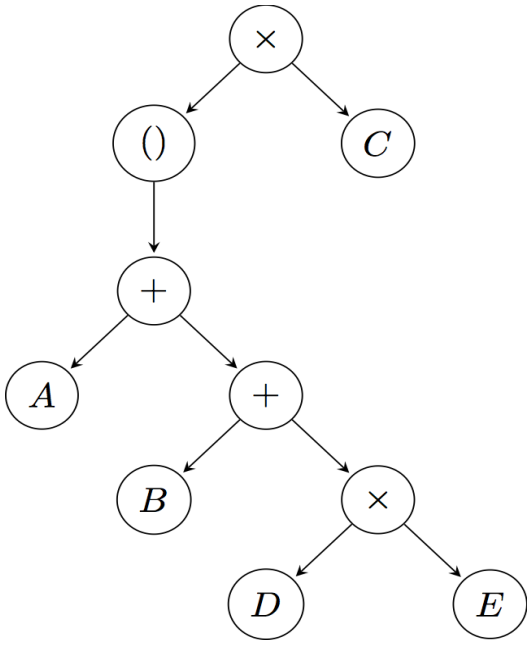
Adding parentheses into the mix proves a slightly harder task; let's consider just the issue of verifying whether the string consisting solely of parentheses is valid (e.g. "()(())((()()))" is a valid string while "(()" is not). This actually requires a more powerful model than a finite automaton, because we have to be able to keep track of how many parentheses we have left to close as we go through the string---that is, we need a way to store additional memory. A simplest way to do so is to have an auxiliary stack of symbols, and be able to make decisions based on not just the current character in the string, but also the topmost symbol of the stack---keeping in mind that now transitions can change both the state of the system and the state of the stack. This construction is known as a **pushdown automaton**, and is illustrated by the figure on the RHS. Now note how our transitions are augmented; they can now depend on two characters (using the empty set symbol to denote an empty stack), and they may be followed by a push/pop operation on the stack. Fun fact: if we were to augment this structure so that it has two stacks, we would obtain a computational power equivalent to a **Turing machine**, and therefore capable of doing any and all processing any computer can do.

Apart from verification, a pushdown automaton is capable of parsing a particular (context-free) **grammar** of expressions, which means that, given an expression belonging to the grammar, the automaton can produce a **syntax tree** of the expression, a structure explaining exactly how this sentence was generated by the grammar. This structure arises commonly in compiler construction and natural language processing, and is therefore quite valuable. For the purposes of our problem, an appropriate initial grammar to consider is as follows (there are some issues with it, which will be discussed later):



$$E ::= [\text{'}A\text{'} - \text{'}Z\text{'}] \mid E + E \mid E{*}E \mid (E)$$

Essentially, what this means is that an expression is either a variable, a sum of two expressions, a product of two expressions, or a bracketed expression. An example syntax tree for the expression "$(A + B + D{*}E){*}C$" is then shown by the figure on the right.



Once we have this structure in our possession, we can easily expand out the parentheses, by maintaining for every considered sub-expression a vector of individual products of variables (representing the sum-of-products form), and appropriately combining these vectors of a syntax tree node's children, based on its operation, in a bottom-up fashion (addition refers to concatenating the children vectors, multiplication to appending all pairs of products between them to one another, and parenthesis nodes can simply be ignored).

The only remaining issue is how to construct the syntax tree, given the grammar and input string. The simplest approach is to make a recursive descent parser, one that will have separate methods for parsing different detected components of the string, and these methods will be mutually recursive as required. Note that the grammar given before is not amenable to such an approach; it is left-recursive, meaning that a call to $parseE()$ would recursively call $parseE()$ in a manner that causes an infinite loop. Furthermore, we haven't properly encoded the fact that there exists operator precedence of * over $+$. As a grammar that fixes both of these issues, we will use the following grammar with three symbols, $E/T/F$, with E being the root symbol to parse from:

$E ::= T + E \mid T$

$T ::= F * T \mid F$

$F ::= (E) \mid ['A' - 'Z']$

You may note that this grammar no longer has left-recursion, and also it will aim to perform multiplications "deeper" than additions in the generated tree, properly encoding operator precedence. Should we maintain a structure ast_node defined as follows:

```
enum nonterm {E, T, F};

struct ast_node
{

  nonterm kind; // is this an E, T or F symbol

  char leaf; // if it is a variable, what is its character ID

  vector<ast_node*> chds; // children in the syntax tree
};
```

Then the C++ implementations of mutually recursive functions for parsing each of the three symbols are provided in tabular form below.

```
ast_node *parseE()
{
  ast_node *ret = new ast_node();
  ret -> kind = E;
  ast_node *l = parseT();
  ret -> chds.push_back(l);
  if (ind < len)
  {
    if (line[ind] == '+')
    {
      ind++;
      ast_node *r = parseE();
      ret -> chds.push_back(r);
    }
  }
  return ret;
}
```

```
ast_node *parseT()
{
  ast_node *ret = new ast_node();
  ret -> kind = T;
  ast_node *l = parseF();
  ret -> chds.push_back(l);
  if (ind < len)
  {
    if (line[ind] == '*')
    {
      ind++;
      ast_node *r = parseT();
      ret -> chds.push_back(r);
    }
  }
  return ret;
}
```

```
ast_node *parseF()
{
  ast_node *ret = new ast_node();
  ret -> kind = F;
  if (line[ind] == '(')
  {
    ind++;
    ast_node *chd = parseE();
    ret -> chds.push_back(chd);
    ind++;
  }
  else
  {
    ret -> leaf = line[ind++];
  }
  return ret;
}
```

It should be clear that the complexity of all algorithms involved is linear, in both time and space. Furthermore, note that whenever recursion happens, the position of the index (`ind`) is advanced, and therefore no infinite loop is possible.

Added by: leandro
Solution by:

Name: ***Petar Veličković***
E-mail : pv273@cam.ac.uk

Problem R1 10: (CH) Automatic Brainf

Time Limit: 22 second

Memory Limit: 1536 MB

In this problem, you'll make an automatic Brainf**k code generator. Given a text file (input data), your program should output brainf**k code that if executed it will print the text file (exactly same with given input data). The text file contains printable $ASCII$ character only: $\{ '<line\ feed>'(ASCII(10)) , '<space>'(ASCII(32)) , \ldots , '\sim'(ASCII(126)) \}$. It's guaranteed that the filesize of text file is less than 1MB.

Input

Text file containing printable $ASCII$ only with size<1MB.

Output

Brainf**k code that print that text file.

Score

Your score is: Sum of all BF code length in all test data

Example

Input	Output
Hello World!	++++++++++[>+++++++>++++++++++>+++>+<<<<-] >++.>+.+++++++..+++.>++.<<+++++++++++++++.>.+++.- ------.--------.>+.>. **Your Score (BF Code Length, valid command only):** 111

Solution:

As with any other challenge problem, this problem requires trying out many different ideas and one idea performed particularly well.

First we limit the instructions to $+, -, <, >$ and of course the print instruction. Next, we limit the memory region of the tape to exactly K consecutive fields. We now optionally populate[1] the region with some initial values. Now we define the following greedy algorithm (Algorithm 1):

[1] We can populate the tape faster by using "[" and "]" instructions.

Every time you want to print a symbol x, for each field, calculate the cost (number of instructions) it would take to go to that field, adjust the value in that field and print it. Pick the field with the lowest cost.

Even on its own this algorithm performed reasonably well. However, in our final solution, this algorithm is just the first phase of another greedy algorithm (Algorithm 2):

Every time you want to print a symbol x, for each field, calculate the cost (number of instructions) it would take to go to that field, adjust the value in that field and print it + the cost of running additional M steps of Algorithm 1.

Again as with any other challenge problem, hand-pick the best values for K, M. The problem setter was kind enough to provide detailed feedback for each test case.

Added by: Tjandra Satria Gunawan
Solution by:

Name: ***Ivan Stošić***
School: Faculty of Sciences and Mathematics, University of Niš
E-mail : ivan100sic@gmail.com

Problem R2 01: Special Numbers (Reverse and Add)

Time Limit: 3 second

Memory Limit: 1536 MB

Michael likes to *reverse and add* numbers. When he finds a number, he reverses its digits and adds it to the original number.

One day, when he was doing "reverse and add", he found that some numbers (*special numbers*) can be obtained from two distinct digit numbers. The number 121 is one of them. He can get it from 47 (2-digit) and 110 (3-digit). He tried to find many of these special numbers, but he could find only two numbers under 10000.

He wants to know the first 5,000 special numbers in ascending order. Can you help him?

Input

This problem has no input data.

Output

Output the first 5,000 special numbers in ascending order. (One special number per one line.)

Example

Output
121 1111 ... [4998 lines] ...

Information

Source Limit is 10 KB.

Solution:

Let's represent the digits of a number A with $a_1, a_2, a_3, \ldots a_n$. Let $f(A)$ be the number we get after applying the reverse and add operation on A. $f(A)$ will be of the form $a_1 + a_n,\ a_2 + a_{n-1},\ \ldots\ a_2 + a_1 + a_n$ (ignoring the carrying digits). Let's say that the number A represented as $a_1, a_2, a_3, \ldots a_n$ and the number B represented as $b_1, b_2, b_3 \ldots b_{n-1}$ are such that $f(A) = f(B) = X$. Let X be represented as $x_1, x_2, x_3 \ldots x_n$. It must have exactly n digits, since after doing the reverse and add operation on A, the result will have at least n digits, and after doing it on B, the result will have no more than n digits.

Already, we can make a couple more observations: $x_1 = 1$, since you cannot carry more than 1 when adding two numbers. This implies that $a_1 + a_n = 1$. Furthermore, $b_1 + b_{n-1} = 11$, because $x_1 = x_n = 1$. This kind of reasoning motivates us to choose the values of $a_1 + a_n$, $b_1 + b_n$, $a_2 + a_{n-1}$, $b_2 + b_{n-1}$... in this one, or in a very similar order.

We can iterate over n, and find all the special numbers with n digits. We will recursively choose the digits of a and b always making sure that we prune the recursion if we are sure it will not lead to a special number. Let's say we have chosen several digits of A and of B. It is easy to check what are the maximum and minimum numbers we can achieve as $f(A)$ and $f(B)$. If these intervals do not intersect, we can prune the recursion because we know it doesn't lead to a special number. Also, by choosing several digits of A, we have set some numbers of last digits of $f(A)$. Thus, we check whether the last digits of $f(A)$ and $f(B)$ can possibly match.

It is difficult to analyze the exact complexity of this solution, but the recursion will be pruned very often and the program fits within the time limit.

Added by: Min_25
Solution by:

Name: ***Domagoj Bradac***
E-mail: domagoj.bradac@gmail.com

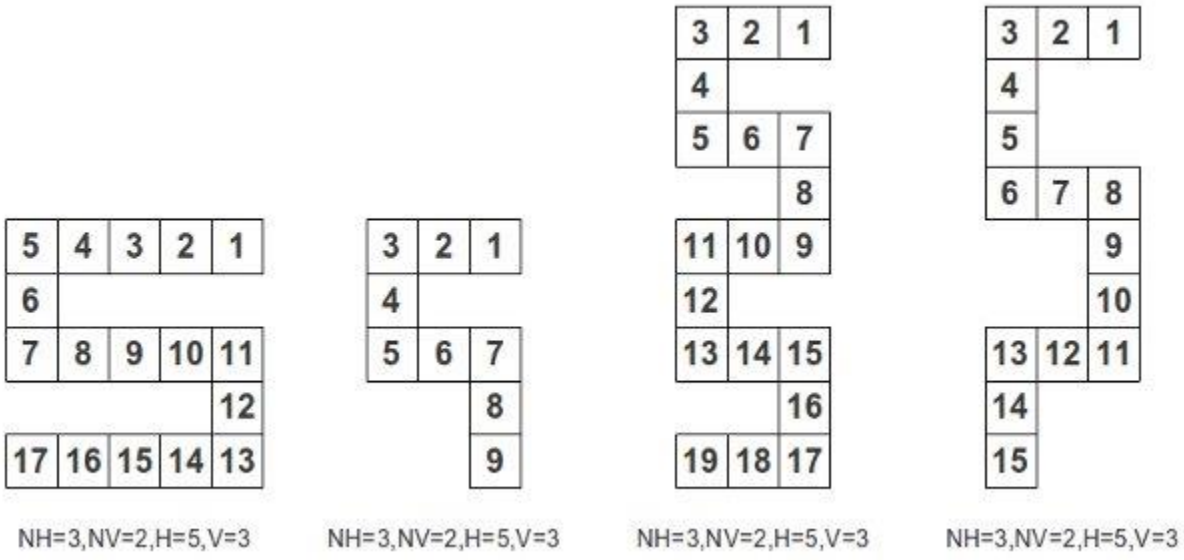
Problem R2 02: World Record Lunch

Time Limit: 0.111s-0.620 second

Memory Limit: 1536 MB

A group of people is trying to beat the world record for the largest number of people having lunch at the same time. In order achieve this goal, they are using the country's largest bridge and they have decided to arrange the tables following the shape of the letter 'S'.

The table layout can be described by 4 integers: NH, NV, H and V. The two first integers, NH and NV, represent respectively the number or rows and number of columns in the layout. The last two integers represent respectively the number of tables in each row and column. For a given layout, the tables are numbered consecutively, starting with table #1 in the top-right corner. The following figure illustrates several possible layouts:



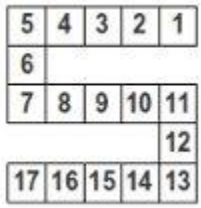
Thousands of groups of people are expected to come, and the organizers have to define where to seat everyone. Each group needs a certain number of tables and they do not share tables with other groups. Furthermore, a group wants their tables to be together and not split among rows and columns, that is, they want a set of consecutive tables either on the same row or on the same column. If this condition cannot be met, the group prefers to go away and have lunch at another place. The groups also enjoy having some privacy and prefer unoccupied adjacent tables, that is, no one at the table exactly before the first table of the group, and no one at the table exactly after the last table of the group. If this happens, we say that the group found a **private** place.

Whenever a group arrives, the organization must decide where to seat that group based solely on the table occupation at the moment, without taking into account further groups that may be coming. Since a group will always be seated on consecutive tables, the position of a group can be defined by the index of the first table of the group. So, when a group arrive, the organization wants the following:

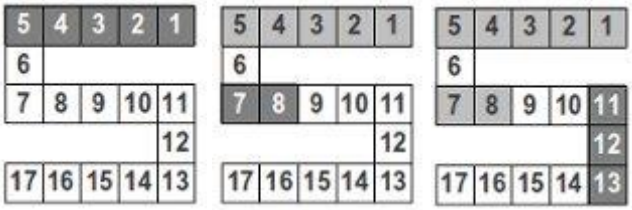
- If there is a suitable private place for the group, then choose the lowest possible index guaranteeing privacy;

- If no privacy can be ensured, but there is space for the group, then choose the lowest possible index with space for the entire group;
- If there is no space available on a single row or column where the group would fit, then the organizers must send the group away.

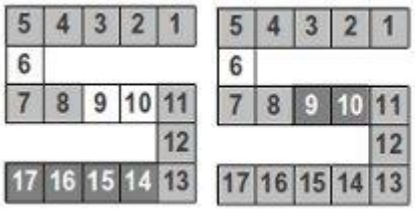
An example would be the following. Imagine a layout with $NH = 3$, $NV = 2$, $H = 5$ and $V = 3$ (the first layout give on the figure above). And now imagine that groups needing 5, 2, 3, 5, 4 and 2 tables arrive, in that order. This is what happens:



In the beginning all tables are empty



Group 1 needs 5 tables. It is put on index 1.
Group 2 needs 2 tables. It is put on index 7.
Group 3 needs 3 tables. It is put on index 11.
Group 4 needs 5 tables. No position is available.



Group 5 needs 4 tables. It is put on index 14 (with no privacy).
Group 6 needs 2 tables. It is put on index 9 (with no privacy).

Can you help the organizers on this task?

The Problem

Given a table layout (number of rows, columns and number of tables per row and column) and the description of groups arriving (specifying number of tables needed per group), your task is to calculate where each group should be seated following the rules described above.

Input

The first line contains 5 integers $NH\ \ NV\ \ H\ \ V\ \ N$, separated by single spaces. NH and NV indicate respectively the number of rows and columns of the table layout. H and V indicate respectively the number of tables per row and column. N indicates the number of groups arriving.

Then come N lines, each one indicating G_i, the number of tables the i^{th} group needs. These lines come in order of arrival of the groups.

Output

The output should have exactly N lines, each one indicating where the respective group should seat. If there is a suitable position, the line should contain an integer indicating the index of the first table of the group. If no position is available, the line should contain the string "no", without the quotes.

Restrictions

The following limits are guaranteed for all the test cases that will be used for evaluating your program:

$1 \leq NH \leq 10\,000$ Number of rows

$NH - 1 \leq NV \leq NH$ Number of columns

$3 \leq H, V \leq 1\,000$ Number of tables in each row/column

$1 \leq N \leq 50\,000$ Number of groups

$1 \leq G_i \leq 1\,000$ Number of tables each group needs

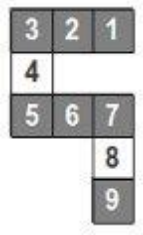
Example Input 1	Example Output 1
3 2 5 3 6 5 2 3 5 4 2	1 7 11 no 14 9
Example Input 2	**Example Output 1**
2 2 3 3 3 3 3 1	1 5 9

Input Explanation 1

It is the example given before in the problem statement.

Input Explanation 2

In the end the tables would look like this:



Solution:

The difficulty of this problem lies mostly in the implementation of its solution. To begin with, let's introduce some notation in order to simplify the explanation of the algorithm. We will call a table *special* if it is unoccupied and if it precedes the first table or succeeds the last table in a row/column. Also, we will imagine that there is a special table immediately before and after the whole table layout. Call a segment of unoccupied tables *alive* if all its tables lie in the same row/column and it can't be extended left/right to obtain a segment of the same kind. We will split such segments into four categories, depending on whether the tables that precede/succeed them are special or not. Throughout the solution, we will keep track of the following information:

- the set of all special tables
- the set of all alive segments
- for each category and length, the set of starting indices of all alive segments corresponding to this category and length
- for each category, a segment tree which can be used to obtain the following information: for a certain length len, what is the smallest starting index of an alive segment from this category whose length is at least len?

At the beginning, we initialize all of these data structures. When a new group arrives, we do the following. We first check if there is a private place that suits the group by querying the segment tree for each category of segments. More precisely, if the group size is sz and a category is described by an ordered pair (i, j), where $i = 1$ if the preceding table is special and $i = 0$ otherwise, similarly for j, then we need a segment belonging to the category of length at least $sz + 2 - i - j$. If we fail to find a private place for the group, we continue searching for any kind of space that suits the group, but this time the required length of a segment is at least sz.

After finding an appropriate place for the group (or finding out that there's no place for it), we have to update the data structures, which is the trickiest part of the solution.

While updating, we are interested in the following information:

- $todo$ - the set of all special tables that cease to be special after the group arrival
- $trans$ - the set of all alive segments whose category changes after the group arrival

- $cand$ - the set of all alive segments that cease to be alive after the group arrival

The set $todo$ can be found by considering the special tables that lie in the segment $[l, r]$ occupied by the group. The set $cand$ can be found as the set of all segments that are intersected by $[l, r]$. The set $trans$ can be found by considering alive segments that are adjacent to at least one table from $todo$ and aren't in $cand$. For the purpose of finding the required sets we can perform a binary search on our data structures (using built-in functions, for example lower bound and upper bound in C++ set). Finally, we have to erase these objects from the corresponding data structures (and possibly insert them back, in case of the elements of $trans$). While inserting/erasing segments, we also need to update the segment tree.

Despite a relatively high constant factor, the overall complexity of the solution is $O(NlogN)$, because each operation (segment tree query/update, built-in binary search) can be done in $O(logN)$.

Added by: Miguel Oliveira
Solution by:

Name: ***Adrian Beker***
E-mail: adro.beker@gmail.com

Problem R2 03: Eclipse

Time Limit: 30 second

Memory Limit: 1536 MB

Every so often we hear on the news that there is going to be either a solar or lunar eclipse. Eclipses have a long history dating back well into the BC's. Astronomers study total solar eclipses very closely as they provide the rare opportunity to observe the corona.

An eclipse occurs when two celestial bodies and a star are (nearly) linearly aligned and the shadow cast by the one body intersects the other body, creating darkness on the latter body.

We are interested in determining when a solar eclipse will next occur. In Figure 1 you can see two labelled regions. The umbra is the area of total darkness — a body in this region will experience a total solar eclipse. The penumbra is the area of partial darkness — a body in this region will experience a partial solar eclipse.

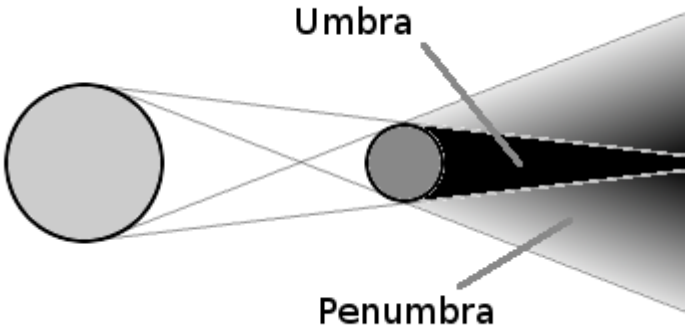
You will be given the size and location of a star and two celestial bodies. Your task is to determine if the first celestial body creates a solar eclipse on the second celestial body. If it does then you are to determine whether it is a total or partial eclipse and whether the entire body is in eclipse. If part of the body is experiencing total eclipse while the entire body is experiencing at least a partial eclipse, we are only interested in the part that is in total eclipse.

Consider a scaled model of our solar system with the sun at the origin (0, 0, 0) with radius 700, the moon at position (49900, 1000, 149700) with radius 2 and Earth at position (50000, 1000, 150000) with radius 7. In Figure 1, the sun would be the star on the left and the moon would be the smaller body on the right. Part of Earth would then fall in the black umbra region and hence partly experience a total solar eclipse.

For any body:

- $1 \leq r \leq 10^6$
- $0 \leq x, y, z \leq 10^9$

It is guaranteed that any two bodies will be at least 1 unit apart, and that moving any one of the bodies by 1 unit (in any direction) will not change the answer.



Input

A test case is described by three lines, each describing the size and location of a single body. The first line contains four space-separated integers xs, ys, zs and rs, describing the center (xs, ys, zs) and radius rs of the star. The following two lines define the two celestial bodies in the same manner.

Test cases follow directly after one another with $a\ -1$ representing the end of the test cases.

Output

Each test case has a single line of output describing the type of eclipse for that case. If the second celestial body listed in the test case is experiencing an eclipse, then one of the following lines must be output:

- Entire total solar eclipse
- Part total solar eclipse
- Entire partial solar eclipse
- Part partial solar eclipse

If there is no solar eclipse, the line "No solar eclipse" must be output.

Example

Input	Output
0 0 0 700 49900 1000 149700 2 50000 1000 150000 7 0 0 0 10 50 0 100 40 60 0 200 1 -1	Part total solar eclipse Entire total solar eclipse

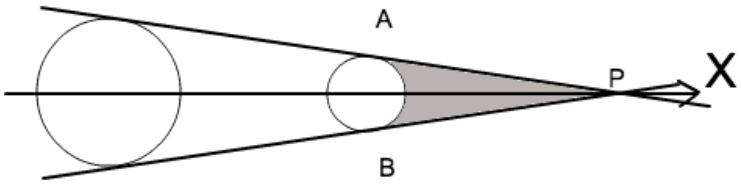
Solution:

The first part of the solution is to understand that the problem is actually 2-dimensional. Let's look at the plane which contains centers of all three bodies. It's enough to check whether the projection of the second celestial body on that plane lies inside the projection of umbra or penumbra. Rather than some formal proof, it would be more helpful if you imagine how all this bodies look like.

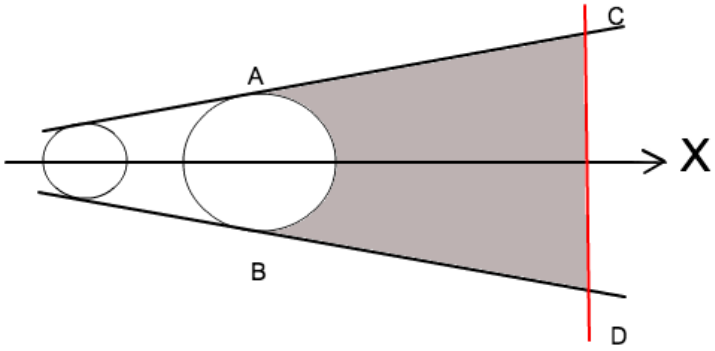
Next step is to rotate this bodies to make them 2-dimensional. We want to make $z = 0$ for all three bodies. It is possible to create rotation matrix but there exists another way which will be explained. Let's move star to point (0,0). Let's define $d_1 = dist(star, first\ celestial\ body)$, $d_2 = dist(star\ , second\ celestial\ body)$ and $\alpha =$ *angle (vector from star to first celestial body, vector from star to second celestial body)*. Let's move first celestial body to $(d_1, 0)$. To find the correct placement of the second celestial body, we need to take vector $(1,0)$, rotate by α and multiply by d_2. As a triangle can be uniquely determined by 2 sides and an angle between them, such placement of bodies is the same as after rotating the plane.

Let's check if the total solar eclipse happens. There are 2 cases when the first celestial body is bigger or smaller than the star. This 2 cases are shown on the following drawings:

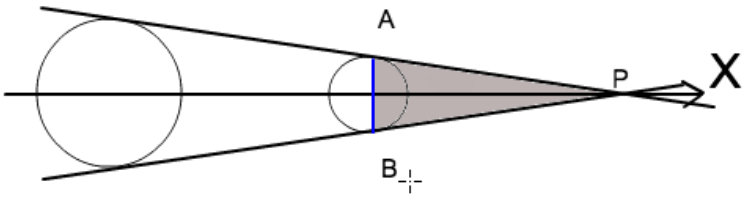
1)



2)



In order to find the edges of the umbra region we need to find the 2 tangents of circles so that points of tangent have the same sign of Y coordinate. For finding the tangents we use the standard algorithm which won't be described in this editorial. Also in the second case, the region is infinite, but it is possible to make it finite adding, for example, constraint $X \leq 10^{10}$ (red line on the second drawing). To make the problem easier let's change the gray region to the following (the drawing corresponds only to case 1, the second case is the same):



The answer for this region will be the same because celestial bodies can't intersect. The umbra region becomes polygon. Now we need to check if the circle lies inside the polygon or intersects the polygon.

Firstly, in order to check that the circle lies inside the polygon we must confirm that the circle center is inside the polygon and that the distances from the center to all sides should be greater or equal to radius.

Then, to check if the circle intersects with the polygon we need to confirm that the distance from the center to one of the sides is less than radius.

For penumbra we have only one case which can be solved in a similar way: find 2 other tangents (where points of the tangent have different signs of Y coordinate), then make them finite ($X \leq 10^{10}$) , add part of the first celestial body so that penumbra becomes polygon and check if the second celestial body lies inside this region or intersects it.

All of this requires careful implementation but you shouldn't have any precision problem as it is guaranteed in the statement.

Added by: Marco Gallotta
Resource: *ACM Southern African Regionals 2007*
Solution by:
Name: ***Roman Bilyi***
E-mail: roman.bilyi@gmail.com

Problem R2 04: Travelling Santa

Time Limit: 2 second

Memory Limit: 1536 MB

At last, Santa is on his way! He's got a number of presents to deliver to various households, and he won't stop until he's disposed of them all. However, each present is only suitable for one gender - for example, men might enjoy such things as baseball bats, football helmets, and certain adult reading material, while women would prefer make-up, knitting needles, and certain other adult reading material.

At the moment, Santa is in a neighborhood with H $(1 \leq H \leq 50)$ houses (numbered $1..H$), connected by R $(1 \leq R \leq 10^4)$ roads. The family living in house i includes M_i $(0 \leq M_i \leq 10)$ males and F_i $(0 \leq F_i \leq 10)$ females. The i^{th} road runs between distinct houses A_i and B_i, and can be travelled in either direction. No pair of houses is directly connected by more than one road.
Santa starts at house 1, carrying M_S $(0 \leq M_S \leq 50)$ male-appropriate and F_S $(0 \leq F_S \leq 50)$ female-appropriate presents. He then repeats the following process until he's out of gifts. First, he moves randomly to an adjacent house (a house reachable by taking one road) - it's guaranteed that there will be at least one such house. If he currently has m male-appropriate and f female-appropriate presents, his probability of moving to adjacent house i is proportional to the value of $M_i m + F_i f$. Of course, the probabilities for all adjacent houses must add up to 1. Note that if this value is 0 for all adjacent houses, then Santa will move to any of them with equal probability. After reaching the new house, he delivers a male present to it with probability $\frac{m}{m+f}$, and a female present otherwise.
Wanting to plan ahead to leaving this neighborhood, Santa is curious as to where he'll end up. He'd like you to calculate the probability of him being at each of the H houses when he runs out of presents.

Input

First line: H and R
Next H lines: M_i and F_i, for $i = 1..H$
Next R lines: A_i and B_i, for $i = 1..R$
Next line: M_S and F_S

Output

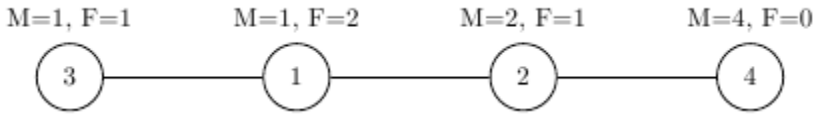
H lines: The probability of Santa finishing his present-delivering process at house i, rounded off to 6 decimal places, for $i = 1..H$

Example

Input	Output
4 3 1 2 2 1 1 1 4 0 1 2 2 4 3 1 1 1	0.760000 0.000000 0.000000 0.240000

Explanation of Sample:

The neighborhood looks as follows:



House 1 is adjacent to houses 2 and 3. The probability of Santa moving to house 2 is proportional to $2 \cdot 1 + 1 \cdot 1 = 3$, while the probability of him moving to house 3 is proportional to $1 \cdot 1 + 1 \cdot 1 = 2$. Therefore, he will move to house 2 with probability $\frac{3}{5}$, and to house 3 with probability $\frac{2}{5}$. If he moves to house 3, then, regardless of which present he delivers, he will be guaranteed to move back to house 1 next, where he will deliver the remaining present.
Otherwise, if he moves to house 2 at first, then he will proceed to deliver a male-appropriate present there with probability $\frac{1}{1+1} = \frac{1}{2}$, and a female-appropriate one also with probability $\frac{1}{2}$. In the former case, he will then have just 1 female present remaining, so he will move on to house 4 with probability 0, and back to house 1 with probability 1. Otherwise, he will have 1 male present remaining, and will move on to house 4 with probability $\frac{4}{5}$, and back to house 1 with probability $\frac{1}{5}$. In any case, he will then deliver his second present and be done.

Editor's note:
You can deliver more male or female presents to a house, than the number of given gender inhabitants. Even if that number would be zero, you could still drop given type of presents. You only need to follow the formulas in the problem description.

Solution:

This was one of the conceptually simpler problems in the second round. The first solution described here will go through all possible paths Santa can take. If there is a probability p that we will end in a house $\boldsymbol{x}$ with the set of gifts S, we can calculate the probability q of Santa going to the house $\boldsymbol{y}$ next, and being left with a set of gifts S', by using the formula described in the task ($M_i m + F_i f$ divided by

sum of these. If the sum is zero, take $1/m$ as probability, with m being the number of cities linked to x). Therefore, we can write a recursive function $dfs(\ x, p, S\ )$ which calls itself as $dfs(\ y, p * q, S'\ )$ for each (y, S') possible. Or if S is empty, increase the probability of ending the jorney in a house x by p:

$$prob[x] \mathrel{+}= p;\ return;$$

Note that S can be easily described by two numbers, and calling the recursion as $dfs(\ 1, 1, (M_s, F_s)\ )$ solves the problem.

However, this is too slow – time complexity can be up to exponential. The way we are going to solve this problem is actually dynamic programming. Instead of recursively checking all possible paths, and visiting the same positions (defined by (x, S)) multiples the time by coming there in many ways, we are going to iterate through each position once, having probabilities of all possible ways coming there already summed up. Iteration should go in this order: for all S, by descending sum M_s+ F_s, for all x, by ascending x. All positions should be initialized as 0, except (1, (M_s, F_s)), which should be set to 1. The relation we are going to use is:

$$dp[S'][y] \mathrel{+}= dp[S][x] * q;$$

Note that, if it is possible to get from the position P to position Q, you will always visit P first, because whenever Santa moves, he loses a gift and the number of gifts he still has decreases. Time complexity is $M_S * F_s * H^2$, because there is $M_S * F_s * H$ states, and from each house Santa can go up to at most H others. For additional information, check a C++ solution at: http://pastebin.com/GNAjZAC2

Added by: SourSpinach
Solution by:

Name: ***Josip Kelava***
E-mail: jkelava6@gmail.com

Problem R2 05: When (you believe)

Time Limit: 0.5 second

Memory Limit: 1536 MB

It's said: there can be miracles, when you believe. The following programming language shows the power of "when". It has a very simple (case insensitive) grammar, shown below:

```
PROGRAM := WHEN | PROGRAM WHEN

WHEN := 'when ' EXPRESSION <ENTER> STATEMENTS 'end when' <ENTER>

STATEMENTS := STATEMENT | STATEMENTS STATEMENT

STATEMENT := PRINT | SET

PRINT := 'print '  EXPRESSION_LIST <ENTER>

SET := 'set ' ASSIGNMENT_LIST <ENTER>

EXPRESSION_LIST := EXPRESSION | EXPRESSION_LIST ',' EXPRESSION

ASSIGNMENT_LIST := ASSIGNMENT | ASSIGNMENT_LIST ',' ASSIGNMENT

ASSIGNMENT := VARIABLE '=' EXPRESSION

EXPRESSION := '(' EXPRESSION OP EXPRESSION ')' | VARIABLE | NUMBER

OP :=  '<' | '+' | '-' | 'and' | 'or' | 'xor'

VARIABLE := '$' NOT_DOLLAR_STRING '$'

NUMBER := DIGIT | NUMBER DIGIT

DIGIT := '0' | .. | '9'

NOT_DOLLAR_STRING := any sequence of printable characters (including blanks)
                     that does not contain a $ symbol.
```

In the above, any string enclosed in single quotes are to be treated literally. <ENTER> is the end of line.

In words, Spaces are allowed before or after any literal except inside a number. Spaces are allowed in variable names, and each non-empty sequence of spaces is treated as a single underscore, so the following refer to the same variable:

```
$Remote Switch#1$

$Remote_Switch#1$

$Remote   switch#1$
```

All numbers appearing in the program will be integers between 0 and 1000000000, inclusive. All variable and literal values are integers between -1000000000 and 1000000000, inclusive. All variables are global and initially zero. The programs you will be tested on will never have an EXPRESSION that evaluates to a value outside of this range. The logical operators evaluate to 0 for false and 1 for true, and treat any nonzero value as true.

Running the program amounts to executing all the active when clauses until none are active. More specifically, the active list of when clauses are initially empty, then the following steps are repeated:

In the order they appear in the program, the conditions of all when clauses that are not currently active are evaluated. If true, the clause is added to the end of the active list, with its first statement marked as "ready". Each active when clause has one "ready" statement.

If the active list is empty after this step, the program terminates.

The "ready" statement from the "current" when clause (initially the first clause in the active list) is executed.

The statement marked as "ready" is advanced, removing the when clause from the active list if this is the last statement in the "current" when clause.

The when clause marked as "current" is advanced, cycling to the beginning of the active list if the end is reached.

In other words, inactive when conditions are evaluated to determine if these clauses are added to the active list. Then one statement (set or print) is executed from the current active when clause. If this is the last statement in that clause, it is removed from the active list. One the next iteration, one statement is executed from the next active when clause, etc.

A set statement executes all the assignments concurrently, so that

```
set $x$ = $y$, $y$ = $x$
```

swaps the values of $\$x\$$ and $\$y\$$. The same variable cannot appear twice on the left hand part of the same set statement (so set $\$x\$ = 1, \$x\$ = 2$ is illegal).

A print statement evaluates and prints the given expressions in the output, separated by commas and followed by a new line. So

```
print 1, (2+3)
```

results in the line

1,5

in the output.

Input

The input consists of a single syntactically correct program. You may assume that the program will not execute more than 100000 set statements and 100000 print statements.

Output

Print the output produced by executing the given program. Both the input and output file will not exceed 100KB.

Example

Input	Output

When ($Mr. Bill$<5)
  Set $mr._bill$=($mr. bill$+1),Y=(Y+10)
End When
When ($mr. Bill$<10)
  Set $MR. BILL$=($mr. bill$+1)
  Print $mr. bill$,Y
End When
	3,20
	6,40
	7,40
	8,40
	9,40
	10,40

Solution:

In this problem you had to parse a program written in a pseudo programming language and execute it.

The solution for this problem is pretty straightforward. You just have to parse the given text, build a program tree with some recursive functions while parsing expressions and then just emulate the program as described in the statement.

Before emulating the program, you should preprocess its text. You should convert all the characters to lowercase and replace all whitespaces inside the variables to underscores ("_"). Also it could be useful to remove the extra whitespaces between words or even split program to tokens before emulating it.

To emulate the program, firstly you should split given lines to "when" blocks. For each "when" line you should find the matching "end when" line and parse every line between them. Each line would be a new statement in a list of statements for given "when" block.

If it's a "print" statement, you split the part of line by "," character and build an expression tree for each string.

A "set" statement is treated the same way, you just build an array of pairs of (variable, expression) instead of just an array of expressions.

Parsing expressions could be done recursively. In this problem it's easier than usually since there are no priorities of operators.

There are two cases:

1. A string begins with an open bracket. In this case you should find the matching closing bracket and evaluate an expression inside it.
2. Otherwise the line begins with a number or a variable. You just evaluate it (convert string to a number or substitute a variable with its value).

If there are no more characters in the string, then you just return the result of evaluation.

Otherwise, the next character(s) denote an operator which is applied to already calculated value and a remaining part of a string. Then you should recursively calculate the value of a right side of a string and then apply an operator to both values and return resulting value.

The values of variables can be stored in a map.

When you want to evaluate a "print" statement you just evaluate each expression in its expression list and then print results of all of them.

To evaluate a "set" statement you should firstly evaluate each expression and then change values of corresponding variables.

To evaluate a program, you should support a deque of pairs of ("when" block id, position in "when" block) and a boolean array, i^{th} value of which is true if i^{th} "when" block is currently in deque.

Then you just perform the following cycle:

1. Iterate over all "when" blocks. Evaluate a condition of each block, if it's true and block is not in a deque then add (id of this block, 0) to the end of deque.
2. If deque is empty, then exit from cycle.
3. Remove the first element from deque and evaluate a statement in a corresponding "when" block. If it wasn't a last statement in a block, then you should increment a second value in this pair and add it to the end of deque.

Added by: Fudan University Problem Setters
Resource: *Whitney Houston: When You Believe*
Solution by:

Name: ***Vsevolod Stepanov***
E-mail: tehnar5@gmail.com

Problem R2 06: LIGHTS3 - Lights (Extreme)

Time Limit: 0.535 second

Memory Limit: 1536 MB

Note: it is advisable to solve LIGHTS2 before attempting this problem.

John has n light bulbs and a switchboard with n switches; each bulb can be either on or off, and pressing the i^{th} switch changes the state of bulb i from on to off, and viceversa. He is using them to play a game he has made up. In each move, John selects a (possibly empty) set of switches and presses them, thus inverting the states of the corresponding bulbs. Initially all lights are off, and after exactly m moves John would like to have the first v bulbs on and the rest off; otherwise he loses the game. There is only one restriction: he is not allowed to press the same *set* of switches in two different moves.

This is quite an easy game, as there are lots of ways of winning. This has encouraged him to keep playing different winning games, and now he is intent on trying them all. Help him count how many ways of winning there are. Two games are considered the same if, after a reordering of the moves in one of them, at every step the same set of switches is pressed in both of them.

For example, if $n = 4, m = 3$, and $v = 2$, one possible winning game is obtained by pressing switches 1, 2 and 4 in the first move, 1 and 3 in the second one, and 1, 3 and 4 in the last one. This is considered equivalent to, say, first pressing 1 and 3; then 1, 2, 4; and then 1, 3, 4.

Input

The input has at most 1500 lines, one for each test case. Each line contains three integers n $(1 \leq n \leq 10\,000\,000)$, m $(1 \leq m \leq 10\,000\,000)$, and v $(0 \leq v \leq n)$. The last line of input will hold the values 0 0 0 and must not be processed.

Output

Print one line for each test case containing the number of ways John can play the game, modulo the prime 10 567 201.

Sample Input	Sample Output
3 3 1 6 4 0 6 4 3 0 0 0	7 10416 9920

Solution:

First of all, note that the problem is equivalent to the following one:

Let S be $\{0,1\}^N$. Given $v \in S$, how many sets of m distinct vectors from S satisfy the condition: $v_1 \oplus v_2 \oplus v_3 \oplus \ldots \oplus v_m = v$, where $\oplus$ is bitwise xor operator.

Intuition led me to the fact that for every v not equal to $\{0,0,0,\ldots,0\}$, the answer will be the same, so I've checked it with brute-force program for small inputs, and it confirmed my assumption.

First, take a look when M is odd.

$v_1 \oplus v_2 \oplus v_3 \oplus \ldots \oplus v_m = \vec{0}$ is equivalent to $(v_1 \oplus v) \oplus (v_2 \oplus v) \oplus (v_3 \oplus v) \oplus \ldots \oplus (v_m \oplus v) = v$, so number of solutions for $v = \vec{0}$ is same for any other v.

There are $\frac{2^N}{M}$ different combinations (sum of solutions for every vector v) and 2^N different possibilities of picking vector v, and using the fact that all the choices are equivalent, solution for every vector v will be $\frac{2^N}{M}/2^N$.

Now, there remains a case with M even.

Let's try to find a connection between the answer for $v = \vec{0}$ (let's call it x) given N and M, and the answer for v equal to anything else (let's call it y).

Let's try to transform a solution for v not equal to $\vec{0}$ to a solution with v equal to $\vec{0}$.

$v_1 \oplus v_2 \oplus v_3 \oplus \ldots \oplus v_m = v$

$v_1 \oplus \ldots \oplus (v_i \oplus v) \oplus \ldots \oplus v_m = \vec{0}$

It's possible when there is no j that satisfies $(v_i \oplus v) = v_j$, or $v_i \oplus v_j = v$

Let's count the number of solutions where that is the case. It is true if and only if we can divide our m vectors in $m/2$ pairs of two vectors, such that xor of every pair of vectors is equal to v.

Now, we have the following system of two equations:

$x - y = (-1)^{M/2} \frac{2^{N-1}}{M/2}$ If $M/2$ is even, $x > y$, otherwise $x < y$, and the difference is number of ways to pick $m/2$ pairs.

$x + (2^{N-1})\, y = \frac{2^N}{M}$ On the left, we have a sum of solutions for every possible vector v, and on the right side we have all the possible combinations.

Solving this system yields the following solutions:

$y = \frac{\frac{2^N}{M} - (-1)^{M/2} \frac{2^{N-1}}{M/2}}{2^{N-1}+1}$ and $x = \frac{2^N}{M} - (2^{N-1})\, y$

So, we will calculate all of the powers of 2.

Now, the only thing left to be done is to calculate x and y, and it can be done by a well-known exponentiation by squaring using modulo. Dividing using modulo can be done using modular inverse.

https://en.wikipedia.org/wiki/Modular_multiplicative_inverse

Added by: David García Soriano

Solution by:

Name: ***Vladimir Milenković***

E-mail: vladimirm98@gmail.com

Problem R2 07: K Edge-disjoint Branchings

Time Limit: 0.5 second

Memory Limit: 1536 MB

Given a directed graph, may contains repeated edges. We assume that the graph contains and only contains K edge-disjoint branchings rooted by node 0.
A branching for a graph is a set of directed edges that from a certain root (root, in this problem, is node 0) we can find one path to every other node in the graph by only the edges in the branching.
K edge-disjoint branching is K branchings that share no common edges.
Your task which is easy and funny is to find out the K branchings.

Input

The first line of input contains a single integer T, ($T <= 20$), denoting the number of test cases.
For each test case:
The first line contains two integers N and K, ($2 <= N <= 500, 2 <= K <= 6$), which is the number of the nodes in the graph and the number of edge-disjoint branchings.
Then next $(N-1) * K$ lines contains the information about the edges. There are 2 integers X and Y in every line, meaning there exist an edge from X to Y in the graph.

Output

You should output the branchings you have found.
For every test cases, print the number of test case at the start of output, then you should output K lines.
Each line is about a branching which contains $N-1$ integers that the ID of the edges in this branching.
The ID of edges starts with 0. Every edge will appear and only appear once in the output.
See samples for further details.

Example

Input	Output
2 2 2 0 1 0 1 3 2 0 1 0 2 2 1 1 2	Case 1: 0 1 Case 2: 0 3 1 2

Solution:

Let's name the initially given graph G.

The solution of the problem consists of $(K-1)$ steps. On the i^{th} step, we'll do the following:

- First, let's find the branching B, such that the graph $(G-B)$ can be represented as the union of $(K-i)$ disjoint branchings. Let's call such branching B *appropriate*.
- Then, we remove this branching B from the graph G.

So, the essential thing that we are going to do is to decrease the value of K, given in the problem, by one during each step obtaining the smaller-size problem. Now we need to figure out the way to find the appropriate branching B.

If we wanted to find *any* branching B, not necessarily the *appropriate* one, it would be quite simple: we would just do the depth-first traversal of the graph G and then just pick the parent edge for each of the nodes. However, it won't be enough for this problem, because by taking any branching we can destroy the edges of two or more further branchings, so therefore, it will not be possible to build the rest.

Still, finding the *appropriate* branching has much in common with finding *any* branching. In particular, we can still do the depth-first traversal of the graph G that starts at the node 0. Then, when trying to use an edge, we just need to add one more additional check of whether it is possible to take this edge to B and to have the sought number of branchings in $(G-B)$. In the pseudocode, it would look as the following:

```
routine buildBranching(node v):
        visited[v] = true
        for any edge e adjacent to v:
                if not visited[endpoint of e] and G-(B+e) has enough branchings:
                        add e to B
                        call buildBranching(endpoint of e)
```

Now the problem clearly boils down to checking whether the graph $G-(B+e)$ has at least $(K-i)$ disjoint branchings or not. There are two ways to solve it.

The slower way. Let's think about necessary conditions for the arbitrary graph H to have M disjoint branchings.

Claim: If the graph H has M branchings, rooted at the node 0, then the value of the maximal flow/minimum cut between the node 0 and each node V, different from 0 is at least M units.

The proof is quite simple: given that the graph G has M disjoint branchings, we can construct M disjoint ways from the node 0 to the node V, namely, one way in each branching. The existence of the ways

follows from the definition of the branching and the ways will clearly be disjoint, because they belong to different branchings.

Now, it appears that it is a sufficient condition as well, known as **Edmonds' theorem**.

We can use the obtained result. Now, when we need to check, whether the graph H has M disjoint branchings or not, we can just check that the maximal flow from the node 0 to any other node is at least M.

Let's estimate the complexity of such approach. Checking that the maximal flow from the node 0 to the fixed node X doesn't exceed K can be done with K iterations of Ford-Fulkerson's algorithm. One pass of the algorithm will take $O(N + M)$, where M is the number of edges. So, checking a single node results in $O(K(N + M))$. In our problem we have $M \sim KN$, so it is equal to $O(K(N + KN))$, or simply $O(K^2N)$. Having in mind that we will need to check every of $N - 1$ nodes, we obtain the complexity of $O(K^2N^2)$. We can also note that such check will be called $O(K^2N)$ times, so it is quite slow for getting an AC. Probably it may pass after doing a lot of optimizations.

The faster way. We can obtain the faster solution from the slower solution.

Consider, we're having a graph H, such that it contains M edge-disjoint branchings and an edge e in H. The question is, will $(H - e)$ still contain M edge-disjoint branchings or not.

Let's think, what if the disjoint paths could be destroyed when removing the edge $e = (x, y)$ from the graph G. Clearly, those containing this edge. But if the path to some node v, having the edge e is destroyed, and v is not an endpoint of the edge e, then we can find the path that is the «prefix» of the given path that will end at v and will be destroyed in $(H - e)$ as well. That gives us the thought that it is only necessary to check that the amount of the disjoint paths from the node 0 to the node y. So we can get rid of calling the maximal flow $(N - 2)$ times.

That is, we've got a way to check the amount of branchings by calling the maximal flow only once, thus, speeding up the algorithm N times. So, we get $O(K^4N^2)$ solution. Even though it may look like it's «on the verge» of the time limit, it is quite fast. In particular, it is because on each step we decrease K by one and do eliminate $N - 1$ edges. There is room for the further optimizations, such as taking all the edges which were not used in any of the disjoint paths without checking, but such solution passes even without these constant optimizations.

Added by: Fudan University Problem Setters
Resource: *g201513*
Solution by:

Name: ***Sergey Kulik***
E-mail: xcwgf666@gmail.com

Problem R2 08: Swap (Hard - Level 1000)

Time Limit: 100 second

Memory Limit: 1536 MB

Let's play with sequence of non-negative integers. Given two sequence of n non negative integers $(a_1, a_2, \ldots, a_n)$ and $(b_1, b_2, \ldots, b_n)$. Both sequence has maximum element less than k, $max(a_1, a_2, \ldots, a_n) < k$ and $max(b_1, b_2, \ldots, b_n) < k$. The game rule is you can edit both sequence with this operaton: swap a_i and b_i with $1 \leq i \leq n$, and the goal is to make sequence a and b become increasing sequence: $a_i \leq a_j$ if and only if $i \leq j$ and $b_i \leq b_j$ if and only if $i \leq j$. But not all initial sequence a and b can be solved.

For example, (2,0) and (0,1) is a pair of sequence that can't be solved:

- If you don't swap any element, you have (2,0) and (0,1), but sequence (2,0) is not increasing.
- If you swap first element only, then the pair become like this (0,0) and (2,1), sequence (2,1) is not increasing.
- If you swap second element only, then the pair become like this (2,1) and (0,0), again (2,1) is not increasing.
- If you swap both element, then the pair become like this (0,1) and (2,0), again (2,0) is not increasing

So it's impossible to solve if initial sequence is (2,0) and (0,1), because all possible move can't make both sequence become increasing.

Now given n and k, your task is to compute number of different pair of initial sequence (a, b) that can be solved with game described above.

Input

First line there is an integer T denoting number of test case, then T test cases follow.

For each case, there are two integers n and k writen in one line, separated by a space.

Output

For each case, output number of different pair of initial sequence (a, b), since the answer can be large, output the answer modulo 10^9+7.

Constraints

$0 < T \leq 10^4$

$0 < min(n, k) \leq 1000$

$0 < max(n, k) < 10^{1000}$

Example

Input	Output
6 2 1 1 2 1 3 2 2 3 2 2 3	1 4 9 11 26 46

Explanation

Here is list of all possible pair of initial sequence (a, b) on each case:

Case 1: $\{[(0,0),(0,0)]\}$

Case 2: $\{[(0),(0)],[(0),(1)],[(1),(0)],[(1),(1)]\}$

Case 3: $\{[(0),(0)],[(0),(1)],[(0),(2)],[(1),(0)],[(1),(1)],[(1),(2)],[(2),(0)],[(2),(1)],[(2),(2)]\}$

Case 4:
{[(0,0),(0,0)],[(0,0),(0,1)],[(0,0),(1,1)],[(0,1),(0,0)],[(0,1),(0,1)],[(0,1),(1,0)],[(0,1),(1,1)],[(1,0),(0,1)],[(1,1),(0,0)],[(1,1),(0,1)],[(1,1),(1,1)]}

Case 5:
{[(0,0,0),(0,0,0)],[(0,0,0),(0,0,1)],[(0,0,0),(0,1,1)],[(0,0,0),(1,1,1)],[(0,0,1),(0,0,0)],[(0,0,1),(0,0,1)],[(0,0,1),(0,1,0)],[(0,0,1),(0,1,1)],[(0,0,1),(1,1,0)],[(0,0,1),(1,1,1)],[(0,1,0),(0,0,1)],[(0,1,0),(1,0,1)],[(0,1,1),(0,0,0)],[(0,1,1),(0,0,1)],[(0,1,1),(0,1,1)],[(0,1,1),(1,0,0)],[(0,1,1),(1,0,1)],[(0,1,1),(1,1,1)],[(1,0,0),(0,1,1)],[(1,0,1),(0,1,0)],[(1,0,1),(0,1,1)],[(1,1,0),(0,0,1)],[(1,1,1),(0,0,0)],[(1,1,1),(0,0,1)],[(1,1,1),(0,1,1)],[(1,1,1),(1,1,1)]}

Case 6:
{[(0,0),(0,0)],[(0,0),(0,1)],[(0,0),(0,2)],[(0,0),(1,1)],[(0,0),(1,2)],[(0,0),(2,2)],[(0,1),(0,0)],[(0,1),(0,1)],[(0,1),(0,2)],[(0,1),(1,0)],[(0,1),(1,1)],[(0,1),(1,2)],[(0,1),(2,2)],[(0,2),(0,0)],[(0,2),(0,1)],[(0,2),(0,2)],[(0,2),(1,0)],[(0,2),(1,1)],[(0,2),(1,2)],[(0,2),(2,0)],[(0,2),(2,1)],[(0,2),(2,2)],[(1,0),(0,1)],[(1,0),(0,2)],[(1,1),(0,0)],[(1,1),(0,1)],[(1,1),(0,2)],[(1,1),(1,1)],[(1,1),(1,2)],[(1,1),(2,2)],[(1,2),(0,0)],[(1,2),(0,1)],[(1,2),(0,2)],[(1,2),(1,1)],[(1,2),(1,2)],[(1,2),(2,1)],[(1,2),(2,2)],[(2,0),(0,2)],[(2,1),(0,2)],[(2,1),(1,2)],[(2,2),(0,0)],[(2,2),(0,1)],[(2,2),(0,2)],[(2,2),(1,1)],[(2,2),(1,2)],[(2,2),(2,2)]}

Other Info

Test case (n and k) is generated randomly using this rule:

- Probability that $n > k$ *or* $n <= k$ is ~50% each.
- Maximum n and k is random log-uniform.
- Minimum n and k is random uniform.

Click here if you want to know my program speed and other detail.

Explanation about my Algorithm complexity:

My 3.8KB of C code with $O(min(n,k)^3)$ complexity got AC in 32.17s.

Other submission like $O(min(n,k)^4)$ in fast language, and $O(min(n,k)^3)$ in slow language is all TLE. That's why this problem has "Hard" label.
Sorry for slow language user, I think it's impossible to solve this problem unless $O(min(n,k)^2)$ exists. I recommend to try Medium version first, or learn fast language.

About complexity, I've proved using math that no algo with complexity better than $O(min(n,k)^2)$, this is the lower bound. My best algo for now is $O(min(n,k)^3)$, this is the upper bound. So the optimal algo lies between that lower and upper bound. I still don't have proof that my algo is optimal, so there is possibility that there is an algorithm that better than $O(min(n,k)^3)$.

Btw, if I found around $O(min(n,k)^2)$ by myself, I'll set "Extreme" version (level 10000+) of this problem. But if there is someone who solve this problem in around $O(min(n,k)^2)$, of course he/she has honor to set "Extreme" version of this problem.

Time limit ~3× my program top speed.

Solution:

Part 0: The basics

The problem is to find a number of pairs of sequences satisfying a certain condition. Instead, we'll "rotate the problem by 90 degrees" and count the number of sequences of pairs. It is not hard to see that the condition for the sequence of pairs (a_i, b_i) is:

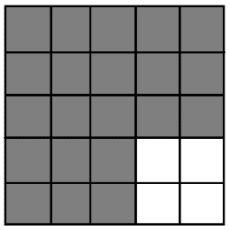
$$((a_{i-1} \le a_i) and (b_{i-1} \le b_i)) or ((b_{i-1} \le a_i) and (a_{i-1} \le b_i)), 2 \le i \le n$$

Let's denote the above relation by $(a_{i-1}, b_{i-1}) \preccurlyeq (a_i, b_i)$.

If this condition is true then there is a way to swap some of the pairs and get a valid sequence, otherwise it is impossible.

This gives us a straightforward dynamic programming solution with time complexity $O(nk^4)$ – a state is described by its length and the last pair, and the solution for each state can be computed in $O(k^2)$:

$$D[l][a][b] = \sum_{(a',b') \preccurlyeq (a,b)} D[l-1][a'][b']$$



This can be improved to $O(nk^2)$ by precomputing the sum of all prefix submatrices of $D[l]$. For example, the picture on the right shows all the pairs that would be taken into consideration by the above sum for the field $(4, 2)$. Our dynamic programming solution actually produces a sequence of matrices. Also note that the value of $D[l][a][b]$ **does not** depend on neither N nor K. Our task is to compute $D[N][K][K]$ for either a small value of N and a huge value of K or for a small value of K and a huge value of N. We will use completely different techniques for these two parts. We'll also set $D[0][1][1] = 1$ and $D[0][a][b] = 0$ for $(a, b) \neq (1,1)$, so the relationship above holds for $l = 1$.

Part 1: A distant field in one of the first 1000 matrices

This is the easier of the two sub problems. It was empirically determined that the values

$$Q_{n,k} = D[n][k][k]$$

are generated by a polynomial of degree $2n-2$. Formally:

$$Q_{n,k} = P_n(k)$$

for some P_n whose degree is $2n-2$. The polynomial's coefficients may not be integers but that is not a problem since we're doing arithmetic modulo p. To compute $P_n(k)$ for arbitrarily large k first we need to find the coefficients of P_n. Since the number of queries is large, we'll precompute the coefficients for all the polynomials $P_n, 1 \leq n \leq 1000$. This can be done by computing the first $2n-1$ values of $P_n(x)$, for example by using the dynamic programming solution above in $O(N_{MAX}^3)$. Now we can use Lagrange polynomials to calculate the coefficients in quadratic time per polynomial. To answer a query, simply compute $x = K \ mod \ (10^9+7)$ and calculate $P_n(x)$. The complexity is $O(N)$ per query.

Part 2: A nearby field in a distant matrix

Observe the infinite sequence of integers $R_{a,b}(n) = D[n][a][b]$. Every such sequence corresponds to one generating function

$$f_{a,b}(x) = R_{a,b}(0) + \ R_{a,b}(1)x + R_{a,b}(2)x^2 + \cdots$$

Note that $R_{a,b}(0) = 0$ everywhere except for $(a,b) = (1,1)$. More often than not, these functions have "simple" closed forms. For example:

$$f_{1,1}(x) = 1 + x + x^2 + \cdots = \frac{1}{1-x}$$

The following is a derivation of $f_{a,b}$ for $(a,b) \neq (1,1)$. Now comes the most interesting part. Observe that, by definition:

$$R_{a,b}(n) = \sum_{(a',b') \preccurlyeq (a,b)} R_{a',b'}(n-1)$$

so,

$$f_{a,b}(x) = \sum_{n=1}^{\infty} (\sum_{(a',b') \preccurlyeq (a,b)} R_{a',b'}(n-1))x^n$$

The sum starts from 1 because the constant term is 0. We can exchange the order of the sums:

$$f_{a,b}(x) = \sum_{(a',b') \preccurlyeq (a,b)} (\sum_{n=1}^{\infty} R_{a',b'}(n-1)\, x^n)$$

$$f_{a,b}(x) = \sum_{(a',b') \preccurlyeq (a,b)} (\sum_{n=0}^{\infty} R_{a',b'}(n)\, x^{n+1})$$

$$f_{a,b}(x) = \sum_{(a',b') \preccurlyeq (a,b)} x f_{a',b'}(x)$$

$$f_{a,b}(x) = x(\sum_{(a',b') \preccurlyeq (a,b)} f_{a',b'}(x))$$

Denote $(a', b') \prec (a, b)$ iff $(a', b') \preccurlyeq (a, b)$ and $(a', b') \neq (a, b)$ and $(a', b') \neq (b, a)$. Note that this relation is a strict partial ordering.

The problem here is that (a, b) appears on the right side of the equation, since $(a, b) \preccurlyeq (a, b)$. Actually, it appears exactly once if $a = b$ and exactly twice when $a \neq b$, since $f_{a,b} = f_{b,a}$. So, for $a = b$ we have:

$$f_{a,b}(x) - 2x f_{a,b}(x) = x(\sum_{(a',b') \prec (a,b)} f_{a',b'}(x))$$

$$f_{a,b}(x) - 2x f_{a,b}(x) = x(\sum_{(a',b') \prec (a,b)} f_{a',b'}(x))$$

$$f_{a,b}(x) = \frac{1}{1-2x} x(\sum_{(a',b') \prec (a,b)} f_{a',b'}(x))$$

Similarly, when $a = b$ we have:

$$f_{a,b}(x) = \frac{1}{1-x} x(\sum_{(a',b') \prec (a,b)} f_{a',b'}(x))$$

We will only search for $f_{a,b}$ where $a \leq b$ since $f_{a,b} = f_{b,a}$.

Although far from obvious, all these generating functions have the following form:

$$f(x) = e + \frac{c_1}{(1-x)^1} + \cdots + \frac{c_u}{(1-x)^u} + \frac{d_1}{1-2x} + \cdots + \frac{d_v}{(1-2x)^v}$$

Here c_i, d_i are constants and the number of terms grows linearly with a, b. We won't prove it, but here's the idea:

$$\frac{1}{1-x} * \frac{1}{1-2x} = \frac{-1}{1-x} + \frac{2}{1-2x}$$

$$\frac{x}{1-wx} = -\frac{1}{w} + \frac{1/w}{1-wx}$$

We can add two generating functions and multiply them with a simple term, such as x or $\frac{1}{1-2x}$ in linear time – this is not trivial but the details are left to the reader as an exercise. To efficiently compute the coefficients in $f_{a,b}(x)$, we'll need to improve the formula for $f_{a,b}$. Based on the properties of the relation $\prec$, the following holds:

$$f_{a,a}(x) = x\, f_{a,a}(x) + f_{a-1,a}(x)$$

$$f_{a,a}(x) = \frac{1}{1-x} f_{a-1,a}(x)$$

For $a < b$:

$$f_{a,b}(x) = f_{a-1,b}(x) + f_{a,b-1}(x) - f_{a-1,b-1}(x) + 2x f_{a,b}(x)$$

$$f_{a,b}(x) = \frac{1}{1-2x}\left(f_{a-1,b}(x) + f_{a,b-1}(x) - f_{a-1,b-1}(x)\right)$$

Now we can compute the coefficients in linear time per generating function. Overall the time needed to precompute all the functions is $O(K_{MAX}^3)$.

Finally, to answer a query N, K, use the function $f_{K,K}(x)$ and calculate the coefficient next to x^N. We can use the following identity:

$$\sum_{n=0}^{\infty} \binom{n+k-1}{k} a^n x^n = \frac{1}{(1-ax)^k}$$

The values for the binomial coefficients can be computed by starting from $k = 1$ where $\binom{n+k-1}{k} = n$ and using $\binom{n+k-1}{k} = \binom{n+k-2}{k-1}\frac{n+k-1}{k}$. Nothing goes to waste as the values for all k from 1 to K may be needed.

The only difficulty here could be computing a^n modulo p where n is huge. The idea is to use Fermat's little theorem: $a^{p-1} \equiv 1 (mod\ p)$, calculate $r = n\ mod\ (p-1)$ and then calculate $a^n = a^{r+(p-1)q} = a^r$ modulo p. The complexity is $O(K)$ per query.

Summary:

Overall, the time complexity is $O(M^3)$ where M is the greatest possible value of $\min(N, K)$ plus an additional $O(\min(K, N) + \log(\max(K, N)))$ per query.

Added by: Tjandra Satria Gunawan
Resource: *Modified Swap (Original)problem*
Solution by:

Name: ***Ivan Stošić***
School: Faculty of Sciences and Mathematics, University of Niš
E-mail: ivan100sic@gmail.com

Problem R2 09: After Party Transfers

Time Limit: 0.346 second-1.038 second

Memory Limit: 1536 MB

You've been to a potluck party (party where each person contributes food), but some people are angry that they spent more money preparing their dish than the others. Now they want you (they span the bottle) to figure out how the participants of the party can, in the *fewest* amount of transfers, become even.

To clarify, you are free to transfer money between them in any way you like, but eventually the money they spent on their dishes, plus the money they received, minus the money they sent, must be equal for all participants.

Input

The first line contains $C \in [0..10]$ - the number of test cases.

For each test case, the first line contains the number $N \in [0..20]$ - the number of people at the party.

The following N lines contains a interger $x_i \in [0,10^6]$ representing the amount of money the i^{th} persons dish cost.

Output

For each testcase:

The first line should be the minimal amount of transfers needed to even out the party budget.

The following lines should be on the form "$a -> b: t$", where $a, b \in [0..N)$ and represent that person 'a' must transfer 't' money to person 'b'. t is here a floating point number of at least 6 digits' precision. (Notice: that is 6 digits' precision after adding together your in and out flows)

In case of multiple best solutions exists, print any.

Example

Input	Output
2 **6** 2 4 1 5 0 6 **3** 9 16 25	3 0 -> 1: 1.0 2 -> 3: 2.0 4 -> 5: 3.0 2 0 -> 1: 7.6666666666667 1 -> 2: 8.33333333333

Explanation:

In the first case, 3 is the minimal amount of transfers. An alternative transfer pattern would be "$0 -> 3:1 \quad 1 -> 3:2 \quad 2 -> 4:2 \quad 2 -> 5:1$". That would have made everyone even, but would have taken one more transfer (4).

In the second case, after the transfers, each person is down 16.666.

Solution:

This problem is a variation of the well-known Subset sum problem. More about the problem can be found on the Wikipedia article – https://en.wikipedia.org/wiki/Subset_sum_problem.

First let's look at the problem naively. We can easily calculate the amount of money each person needs to give or receive (given minus average). We look at two people in each step, the one with the most debt and the one who gave the most money. By transferring money between them we can assure that at least one of them have given appropriate amount of money after the transfer. In a similar manner we can finish the process in $N - 1$ transfers at most.

For our convenience let's subtract the average amount given from each person, and make a new average 0.

Let's now look at one example where we show that the naive solution is not optimal. Suppose that 6 people gave the following amounts: $18, 10, 11, 30, 9, 0$ $(5, -3, -2, 17, -4, -13$ normalized). Naive solution leads to 5 transfers, but the optimal solution is $4: 0 -> 1: 3.0;\ 0 -> 2: 2.0;\ 3 -> 5: 13.0;\ 3 -> 4: 4.0.$ We divided people in two groups of 3 ($\{0,1,2\}, \{3,4,5\}$) and applied the naive solution on them. Notice that the sum of normalized values for each group is 0.

Every time we find two groups that have sums of normalized values add up to 0, we can solve each group separately. This is helpful because it eliminates at least one transfer. We can solve group of N in at most $N - 1$ transfers, so if we divide a group into two of N_1 and N_2 people, we have a solution in at most $(N_1 - 1) + (N_2 - 1) = N_1 + N_2 - 2 = N - 2$. This leads us to the conclusion that separating people in most disjoint subgroups possible will give us an optimal solution.

In light of the previous discussion, we will try to find the solution in a following way: by finding the smallest subgroup and applying our naive solution to it. If that subgroup is not the whole group, we repeat the first step in the rest of the group, otherwise we are finished.

To apply this algorithm, we first find all subgroups that have the sum of normalized values equal to 0 using a modified subset sum algorithm in $O(2^{N/2})$. After that we sort all of the found subgroups by size. With them we construct our solution by picking the disjoint ones.

Advice: Try to avoid float and double since numerical error in calculation can lead to wrong answer. Use rational numbers instead.

Added by: Thomas Dybdahl Ahle
Solution by:

Name: ***Milos Suković***
E-mail: milos93nbgd@gmail.com

Problem R2 10: God Number is 20 (Rubik)

Time Limit: 60 second

Memory Limit: 1536 MB

The Rubik's cube is the most popular toy in the world. Some even consider it as sport, since it requires dedication and training to master it. Also, there are competitions all over the world. There are many ways to compete: $3x3x3$ (Rubik's cube), $2x2x2$, $3x3x3$ blindfolded, $4x4x4$, etc. Usually, a speed cuber takes between 50 and 60 move to solve it, using Fridrich method. Fridrich (or CFOP) is a fast method, but requires a big effort for memorization. There are other methods like Roux (~48 moves), Petrus (~45 moves), ZZ (~50 moves) which requires fewer moves, but CFOP is still faster. There are other popular methods like Old Pochmann (~350 moves) and M2/R2 (~150 moves). They are intended for blind solving, since you can solve one piece at time.

A group of researchers showed, by brute force, that any of the $43{,}252{,}003{,}274{,}489{,}856{,}000$ positions of the Cube can be solved with at most 20 moves (half turn metric), now, "God Number is 20". The algorithm that always find the optimal solution for the cube is known as God's Algorithm, and it is just theoretical by these days.

Note: due to some abuses, in June 10th, 2016, a new judge was added. This one generates random cubes every time, although the n is fixed for every user. You can see the scrambles that generated the cubes you've got by clicking your score.

Input

First line, you have $n < 100$, the number of cubes to be solved. You will be given n valid scrambled cube position, as in the sample, with default orientation (white on top, green on front).

The cube's faces are represented like:

U

LFRB

D

Output

You have to output the length of your solution, then the solution (at most 1000 face turns). Any valid sequence that solves the cube will be accepted. If you want to skip a particular cube, you have to print "-1" instead of the solution length. The penalty for that is 1000 moves, although if you skip all the cases, you will get WA. Every move must be in HTM, without cube rotations or double rotations, i. e., the only accepted moves are $\{U, F, R, \dots\}$ and $\{U2, U', F2, F', \dots\}$.

Score

Score is the sum of the moves plus penalties.

Sample Input	Sample Output
3 W W W W W W O O O O O Y G G G W R R B B B O O Y G G G W R R B B B O O Y G G G W R R B B B R R R Y Y Y Y Y Y R W G W W W W W O G R R B G W B B W O O W O O O G G G R R R B B B O O O G G G R R R B B B Y Y Y Y Y Y Y Y Y G R B O W G O O G W G Y G Y W O R O W W R Y O O B G G Y R W O B B Y G R B Y B R W Y B R R Y R W W Y B G B O	1 F' 7 R U R' U R U2 R' -1

Sample Score

1008

(1+7+1000)

Solution:

The highest scoring solution is surprisingly simple due to the original problem's author's miscalculation. Simply extract the test cases from the SPOJ system and solve them offline. The strategy described below worked during the 2nd qualifying round and may not be successful in the future.

Part 1 - How test extraction works in general

You can use the SPOJ system's feedback (time and memory consumed, reason why your code fails) to extract information from the test case your program is being run on.

When you submit a simple brute-force solution with a depth-limiter, you get a score of 50002 - meaning that it's likely there is a single test case where the solution is only 2 and 50 more complex test cases. This can be verified by submitting a code which, for example times out when $n = 51$.

This number allows us to come up with a more sophisticated extraction scheme. In one submission you can extract one even integer from 2 to 998, for example by adding redundant pairs of moves to the test case where the solution has two steps. This can be extended to odd integers by combining 90 degree twists with 180 degree twists.

Part 2 - Reducing the number of submissions

The problem now is to minimize the number of submissions needed to extract 50 cubes. For one cube, we could extract one facelet at a time. Since there are 54 facelets on a cube, that would take too long. Fortunately, there are far fewer cube configurations, around $s \approx 43 * 10^{18}$. If we could find a bijection f from the set of cube configurations to integers $[0, s-1]$ we could extract one cube in only seven submissions by representing the number $x = f(cube)$ in base 990 and then extracting the digits.

This can be done by separately analyzing corner cubies and edge cubies. It is not hard to see that the number of permutations of corner cubies is $8!$ and the number of orientations is 3^8, the number of permutations of edge cubies is $12!$ and the number of orientations is 2^{12}. In total, this number is $12!\, 2^{12} 8!\, 3^8 = 12s$. The reason for the factor of 12 is that there are some invariants which are maintained after every cube move. For example, the orientation of the 8th corner can be deduced from the other 7 in a valid cube. The same is true for the orientation of the 12th edge cubie. Also, the permutation of the edges will always be even, hence the factor $2 * 3 * 2 = 12$.

Finally, after analyzing the cube and calculating $x = f(cube)$, send 7 submissions to SPOJ to retrieve x and ultimately $cube = f^{-1}(x)$.

Part 3 - Offline solving

For this part there are online solvers which can quickly solve the cube in around 22 moves, but they are not optimal. There is one amazing free program created by Herbert Kociemba which can solve any cube optimally, often within a few minutes. See the link below for more details.

After solving all the cubes offline simply store the solutions into your final SPOJ submission.

Link: http://kociemba.org/cube.htm

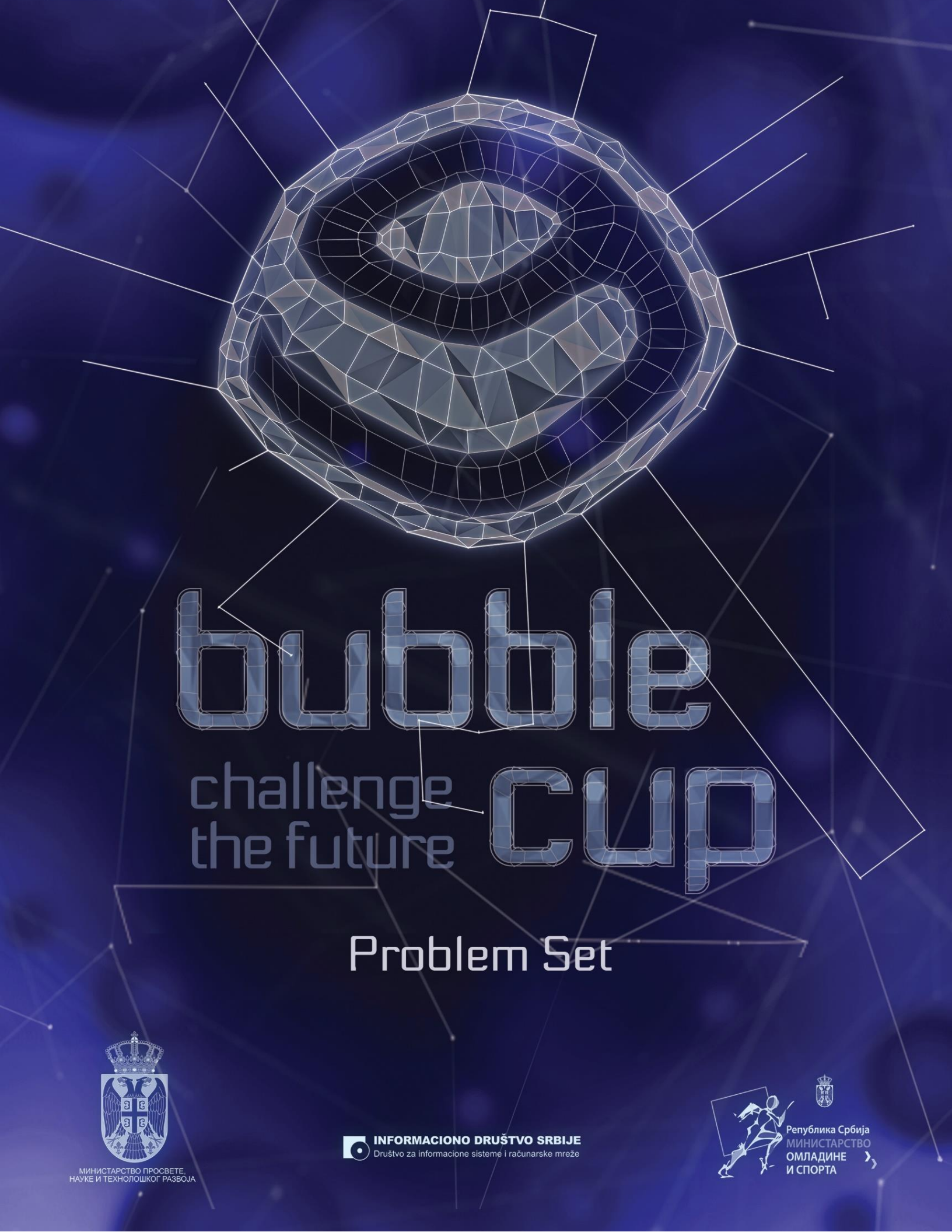
Added by: campos20
Solution by:

Name: ***Ivan Stošić***
School: Faculty of Science and Mathematics, University of Niš
E-mail: ivan100sic@gmail.com

The scientific committee would like to thank everyone who did important behind-the-scenes work.
We couldn't have done it without you!

We will prepare a lot of surprises for you again next year!
Stay with us!

Bubble Cup Crew



Problem Set

МИНИСТАРСТВО ПРОСВЕТЕ,
НАУКЕ И ТЕХНОЛОШКОГ РАЗВОЈА

INFORMACIONO DRUŠTVO SRBIJE
Društvo za informacione sisteme i računarske mreže

Република Србија
МИНИСТАРСТВО
ОМЛАДИНЕ
И СПОРТА