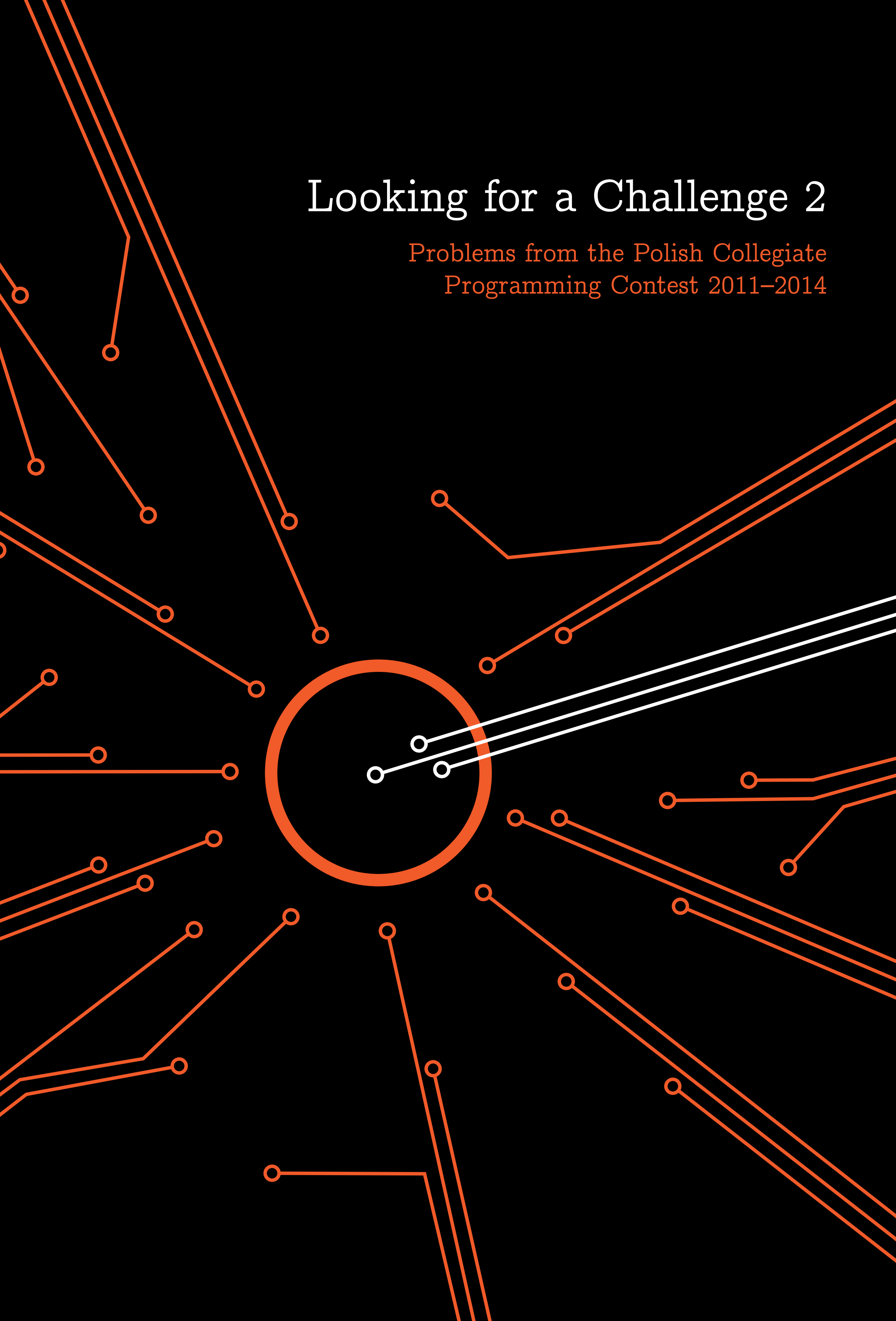
Looking for a Challenge 2

Problems from the Polish Collegiate Programming Contest 2011–2014



Original title: *W poszukiwaniu wyzwań 2*
Zadania z Akademickich Mistrzostw Polski
w Programowaniu Zespołowym 2011–2014
This translation based on second edition (version 2.1)

Editors
Krzysztof Diks, Tomasz Idziaszek, Jakub Łącki, Jakub Radoszewski

Translation
Tomasz Idziaszek (solution descriptions, additional materials),
Jakub Łącki, Jakub Radoszewski, Łukasz Warakomski (problem statements)

Authors of solution descriptions
Szymon Acedański, Tomasz Idziaszek, Adam Karczmarz, Tomasz Kociumaka,
Eryk Kopczyński, Jakub Łącki, Jakub Radoszewski

Authors of problems and model solutions
Szymon Acedański, Marcin Andrychowicz, Karol Cwalina, Marek Cygan,
Dawid Dąbrowski, Krzysztof Diks, Paweł Gawrychowski, Tomasz Idziaszek,
Adam Karczmarz, Tomasz Kociumaka, Eryk Kopczyński, Karol Kurach,
Jakub Łącki, Jakub Pachocki, Jakub Pawlewicz, Jakub Radoszewski,
Wojciech Rytter, Bartosz Szreder, Wojciech Śmietanka, Jacek Tomasiewicz,
Wojciech Tyczyński, Tomasz Waleń

Introduction

We present our Readers with descriptions of model solutions of problems from the Polish Collegiate Programming Contest that took place in years 2011–2014. These four editions of the contest (called in Polish *Akademickie Mistrzostwa Polski w Programowaniu Zespołowym* and abbreviated AMPPZ) were organized by the Faculty of Mathematics, Informatics and Mechanics and the Faculty of Management at the University of Warsaw. The main sponsor of the competition was PKO Bank Polski, and the PKO Bank Polski Foundation supported content-related preparations of the contest and funded scholarships for the best students.

The success of any programming competition depends primarily on proper selection of competition tasks and efficient conduct of the contest. The problem set should be chosen so as to give each competing team a chance of solving at least one task, and at the same time allow to select actual champions. The problems should be original, interesting, and stimulate to improve the skills of all, who did not solve them during the competition. The level of the problems in AMPPZ in years 2011–2014 was watched over by three young scientists from the Faculty of Mathematics, Informatics and Mechanics at the University of Warsaw — Tomasz Idziaszek, Jakub Łącki and Jakub Radoszewski — supported by a large number of students, PhD students and employees of the Faculty. They have prepared 44 original problems. Preparation of each of them required implementing model and alternative solutions, as well as the proper set of tests verifying the quality of solutions of the competitors.

Team programming competitions could not take place without proper technical facilities: computers connected with an efficient network and specialized competition software that evaluates live solutions sent by the competitors. The creators of this software, also used in many other programming competitions organized by the University of Warsaw, are Szymon Acedański and his team from the Faculty of Mathematics, Informatics and Mechanics at the University of Warsaw.

This book contains detailed discussions of solution to problems from AMPPZ in years 2011–2014, written by outstanding young scientists working in the field of algorithmics, who have also achieved significant successes (as competitors and organizers) in team programming competitions in the national and international level. The Polish descriptions of model solutions have been prepared as part of the Masters of Algorithmics project, which the aim was to prepare students of the Faculty of Mathematics, Informatics and Mechanics at the University of Warsaw for participation in programming competitions.

In 2015 we published these solutions in printed form, and now we share them online in electronic form. Moreover, we have decided to prepare English versions of these descriptions to the benefit of the whole international community. Anyone, who would like to solve the tasks, can use the portal `szkopul.edu.pl` (links to the individual problems are also given inside the book).

Tasks from AMPPZ are an excellent training material for anyone who would like to improve their algorithmic and programming skills, or explore the arcana of algorithms for their own satisfaction. We have marked the difficulty level of each problem with stars, so that every reader should be able to find something for themselves. One star denotes the simples tasks, whereas the most difficult are denoted by four stars.

We invite you to read the book, while wishing you a great intellectual experience.

Acknowledgements

The Polish Collegiate Programming Contest in years 2011–2014, organized in Warsaw by the University of Warsaw, were large organizational undertakings, requiring non-standard solutions (logistic, technical and content-related) in the field of computer science. Many institutions and people contributed to the realization of these undertakings. I am grateful to all who have made these competitions so successful. At the same time, I would like to thank in a special way to the most involved in the preparations of the Championships.

Warm words of thanks go to Mr. Zbigniew Jagiełło, the President of PKO Bank Polski, and Mrs. Urszula Kontowska, the President of PKO Bank Polski Foundation, for understanding the importance of such competitions in development of Polish IT and its promotion in the world. PKO Bank Polski as the main sponsor of the competition in Warsaw and teams of the University of Warsaw, contributed significantly to strengthen the rank of the Championships and the successes of Poles in international programming competitions.

The championships were held at the University of Warsaw with the use of university infrastructure. I thank the university authorities for kindness and support in carrying out such a complex organizational project. In particular, words of gratitude go to the Rectors of the University of Warsaw: Mrs. Prof. Katarzyna Chałasińska-Macukow, the Rector in years 2005–2012 and Mr. Prof. Marcin Pałys, the Rector in years 2012–2016, as well as Vice-Rectors: Prof. Tadeusz Tomaszewski and Prof. Alojzy Nowak. Great understanding and help was given by the deans of the faculties directly organizing the competitions: Prof. Stanisław Betley and Prof. Andrzej Tarlecki (from the Faculty of Mathematics, Informatics and Mechanics), as well as Prof. Alojzy Nowak and Prof. Jerzy Turyna (from the Faculty of Management).

The competition took place at the Faculty of Management at the University of Warsaw. Each time the Faculty gym was transformed into a great computer laboratory equipped with dozens of computer workstations connected into an efficient network, with servers and printers in the back room, and the ability to conduct television and internet broadcasts. The contests required specialized, dedicated software that enabled their carrying out in real time. It can be said without exaggeration that the technical side of the competition was secured by world-class professionals, who worked under the leadership of Marek Mossakowski and Jerzy Rolewicz from the Faculty of Management (room preparation, energy security, computer network) and Szymon Acedański from the Faculty of Mathematics, Informatics and Mechanics (dedicated competition software). I express the highest words of recognition to everyone working on the technical support of the competition.

Similarly to the technical, the content-related part of the competition was at the highest world level. I address words of admiration and gratitude for their work to Tomasz Idziaszek, Jakub Łącki and Jakub Radoszewski, who together with a large number of students, PhD students and employees of the Faculty of Mathematics, Informatics and Mechanics, have put a titanic effort into preparation of task competition and running a competition itself.

Finally, I would like to thank the people who directly helped me coordinate the preparation and conduct of the competition in the years 2011–2014. Working with them was a real pleasure. I thank Prof. Jan Madey for introducing me to the world of programming competitions and heading with me the competition in Warsaw. I thank Rafał Sikorski, a lawyer who liked computer science, for all daily organizational help. I am also grateful to Krzysztof Ciebiera, on whom I could always count, for his, often priceless, advice regarding organization of competitions and technical solutions.

Krzysztof Diks

About programming competitions

Team programming contests test the knowledge and skills needed at the work of every programmer. The work of algorithmists and programmers, thanks to whom we can store enormous amounts of information, so that we can search them very quickly and reach out to those just of interest to us, indirectly affects life of everyone of us. Like all practical IT problems, every competition task boils down to developing a method (algorithm) of rapid data processing to calculate the desired results. You need to be aware, that a participant of the competition not only has to arrange an appropriate algorithm, but also correctly program it in the programming language of their choice (usually C++, C, Java, or Python). It is known that even experienced programmers have to work hard to write a working program. On the way to this aim, they first fight with syntactic (compilation) errors, then they carry out a number of tests to eliminate logical errors, and finally they analyze the performance of their solutions.

This is the path followed by every participant of an algorithmic and programming competition. On a small scale, he or she passes the stages of implementation of real programming projects. Team programming contests also teach group work in a natural way. Typical rules of such competitions state that the contestants are divided into three-person teams, with only one computer available for each team. Competitors must skillfully divide the tasks to be solved depending on their competences and together look for solutions to the most difficult tasks. To successfully compete in team programming contests, you need the following skills:

- ability to precisely analyze algorithmic tasks (knowledge of mathematics and logical reasoning are extremely helpful here),
- efficiency in programming,
- knowledge of at least one development environment and ability to compile, debug and execute programs in this environment,
- knowledge of design techniques for algorithms and data structures,
- ability to work in a team.

The best competitors have mastered these skills at levels unmatched by an average professional IT specialist.

International Collegiate Programming Contest

The International Collegiate Programming Contest (in short ICPC) is the oldest and most prestigious IT competition in the world. The contest is intended for students. Each team participating in the competition consists of three people representing the same university. The championship has two stages. The first stage is the regional qualifying rounds, held on six inhabited continents.

Both regionals and final competitions are carried out in the same way. Each team of three has at its disposal one computer and from several to more than a dozen algorithmic-programming task to be solved within five hours. A solution to each task is a computer program that should correctly and efficiently calculate the results for the data prepared by the organizers of the competition. The solutions of the tasks are verified in real time and the teams are informed about the result of the verification: accepted, wrong answer, time-limit exceeded, run-time error. If the solution is not accepted, it is rejected and the team can send further solutions for this task. The competition is won by the team with the most accepted (solved) tasks. In the case of the same number of tasks accepted by more than one team, their order in the standings is determined by the shorter total time spent during solving all accepted tasks. In addition, a 20-minute penalty fee is charged for any previously rejected submission for a task that was ultimately accepted.

The roots of the ICPC date back to 1970, when in Texas a programming competition was held for the first time under the patronage of the Upsilon Pi Epsilon Computer Science Honor Society (UPE). The year 1977 is considered the first year of the International Collegiate Programming Contest. For the first time, the competition was a two-stage one, and the finals were held in Atlanta in conjunction with the ACM Computer Science Conference. Until the 1988/1989 academic year, the competition was mainly represented by universities from the USA and Canada. In 1989/1990 the competition gained global reach. In 1997, IBM started as the main sponsor of the competition and its rapid development has been observed ever since. In this year 2520 students took part in the qualifying rounds of the competition, while in the 2018/2019 academic year there were as much as 52 709. Among the 43 world champions to date, we will find teams representing the best IT universities in the world from four continents, including the University of Warsaw. The list on the next page provides a full list of all world champions.

Just like the ICPC is inseparably linked with Prof. Bill Poucher from Baylor University, who was the originator and animator of the competition, the development of team programming competitions in Poland and the successes of the University of Warsaw students are associated with Prof. Jan Madey, who in 1994 formed the first representation and sent it to the regional qualifications in Amsterdam. The team consisting of Jacek Chrząszcz, Piotr Krysiuk and Tomasz Śmigielski made a huge surprise by winning the qualifying round and advancing to the finals in Nashville. At the time, 38 teams took part in the finals and the Polish inexperienced team took eleventh place — the first not awarded with a medal*. Since then, teams of the University of Warsaw have been taking part in the competition finals without interruption. The University of Warsaw is the only university in the world that can boast such a number of uninterrupted participations in the finals. The greatest successes of the University of Warsaw (and Poland) were victories in the finals of the competition in 2003 and 2007. In 2003 won the team consisting of: Tomasz Czajka, Andrzej Gąsienica-Samek and Krzysztof Onak. In 2007 the world champions became Marek Cygan, Marcin Pilipczuk and Filip Wolski.

*For several years, 12 medals have been awarded in the finals: 4 gold, 4 silver, and 4 bronze. Previously, at most 10 medals were awarded. In very rare cases, when team achievements are similar, additional bronze medals may be awarded.

		Year and place of contest	*World Champions in Collegiate Programming*
1	1977	Atlanta (USA)	Michigan State University (USA)
2	1978	Detroit (USA)	Massachusetts Institute of Technology (USA)
3	1979	Dayton (USA)	Washington University (USA)
4	1980	Kansas City (USA)	Washington University (USA)
5	1981	St. Louis (USA)	University of Missouri-Rolla (USA)
6	1982	Indianapolis (USA)	Baylor University (USA)
7	1983	Melbourne (USA)	University of Nebraska (USA)
8	1984	Philadelphia (USA)	Johns Hopkins University (USA)
9	1985	New Orleans (USA)	Stanford University (USA)
10	1986	Cincinnati (USA)	California Institute of Technology (USA)
11	1987	St. Louis (USA)	Stanford University (USA)
12	1988	Atlanta (USA)	California Institute of Technology (USA)
13	1989	Louisville (USA)	University of California at Los Angeles (USA)
14	1990	Washington (USA)	University of Otago (New Zealand)
15	1991	San Antonio (USA)	Stanford University (USA)
16	1992	Kansas City (USA)	University of Melbourne (Australia)
17	1993	Indianapolis (USA)	Harvard University (USA)
18	1994	Phoenix (USA)	University of Waterloo (Canada)
19	1995	Nashville (USA)	Albert-Ludwigs-Universität Freiburg (Germany)
20	1996	Philadelphia (USA)	University of California at Berkeley (USA)
21	1997	San Jose (USA)	Harvey Mudd College (USA)
22	1998	Atlanta (USA)	Charles University in Prague (Czech Republic)
23	1999	Eindhoven (Netherlands)	University of Waterloo (Canada)
24	2000	Orlando (USA)	St. Petersburg State University (Russia)
25	2001	Vancouver (Canada)	St. Petersburg State University (Russia)
26	2002	Honolulu (USA)	Shanghai Jiao Tong University (China)
27	2003	Beverly Hills (USA)	University of Warsaw (Poland)
28	2004	Prague (Czech Republic)	St. Petersburg ITMO University (Russia)
29	2005	Shanghai (China)	Shanghai Jiao Tong University (China)
30	2006	San Antonio (USA)	Saratov State University (Russia)
31	2007	Tokyo (Japan)	University of Warsaw (Poland)
32	2008	Banff (Canada)	St. Petersburg ITMO University (Russia)
33	2009	Stockholm (Sweden)	St. Petersburg ITMO University (Russia)
34	2010	Harbin (China)	Shanghai Jiao Tong University (China)
35	2011	Orlando (USA)	Zhejiang University (China)
36	2012	Warsaw (Poland)	St. Petersburg ITMO University (Russia)
37	2013	St. Petersburg (Russia)	St. Petersburg ITMO University (Russia)
38	2014	Yekaterinburg (Russia)	St. Petersburg State University (Russia)
39	2015	Marrakech (Morocco)	St. Petersburg ITMO University (Russia)
40	2016	Phuket (Thailand)	St. Petersburg State University (Russia)
41	2017	Rapid City (USA)	St. Petersburg ITMO University (Russia)
42	2018	Beijing (China)	Moscow State University (Russia)
43	2019	Porto (Portugal)	Moscow State University (Russia)

The following list presents the full list of achievements of Polish students in the history of participation in International Collegiate Programming Contest (obtained places and medals). To date, teams from the following universities have advanced to the finals: University of Warsaw (UW), Jagiellonian University in Krakow (UJ), University of Wroclaw (UWr), AGH University of Science and Technology (AGH) and Poznan University of Technology (PP). In total, Polish teams have won six gold, eight silver and seven bronze medals.

ICPC 1995 (38 teams)

11.	UW	Jacek Chrząszcz, Piotr Krysiuk, Tomasz Śmigielski

ICPC 1996 (43 teams)

17.	UW	Marcin Mucha, Krzysztof Sobusiak, Tomasz Śmigielski

ICPC 1997 (50 teams)

11.	UW	Bartosz Klin, Marcin Mendelski-Guz, Marcin Sawicki

ICPC 1998 (54 teams)

9. (bronze)	UW	Adam Borowski, Jakub Pawlewicz, Krzysztof Sobusiak

ICPC 1999 (62 teams)

11.	UW	Bartosz Klin, Marcin Sawicki, Marcin Stefaniak

ICPC 2000 (60 teams)

22.	UW	Łukasz Anforowicz, Marek Futrega, Eryk Kopczyński

ICPC 2001 (64 teams)

6. (silver)	UW	Tomasz Czajka, Andrzej Gąsienica-Samek, Marcin Stefaniak

ICPC 2002 (64 teams)

11.	UW	Łukasz Kamiński, Eryk Kopczyński, Tomasz Malesiński

ICPC 2003 (70 teams)

1. (gold)	UW	Tomasz Czajka, Andrzej Gąsienica-Samek, Krzysztof Onak

ICPC 2004 (73 teams)

10. (bronze)	UW	Tomasz Malesiński, Krzysztof Onak, Paweł Parys
27.	UJ	Grzegorz Gutowski, Arkadiusz Pawlik, Paweł Walter

ICPC 2005 (78 teams)

5. (silver)	UWr	Paweł Gawrychowski, Jakub Łopuszański, Tomasz Wawrzyniak
17.	UW	Szymon Acedański, Tomasz Idziaszek, Jacek Jurewicz

ICPC 2006 (83 teams)

2. (gold)	UJ	Arkadiusz Pawlik, Bartosz Walczak, Paweł Walter
7. (silver)	UW	Marcin Michalski, Paweł Parys, Bartłomiej Romański

ICPC 2007 (88 teams)

1. (gold)	UW	Marek Cygan, Marcin Pilipczuk, Filip Wolski
26.	UWr	Paweł Gawrychowski, Jakub Łopuszański, Tomasz Wawrzyniak

ICPC 2008 (100 teams)

13.	UW	Marek Cygan, Marcin Pilipczuk, Filip Wolski
31.	AGH	Daniel Czajka, Andrzej Szombierski, Marcin Wielgus
31.	UJ	Rafał Józefowicz, Alan Meller, Bartosz Walczak

ICPC 2009 (100 teams)

9. (bronze)	UW	Marcin Andrychowicz, Maciej Klimek, Marcin Kościelnicki
34.	UJ	Kamil Kraszewski, Marek Wróbel, Paweł Zaborski

ICPC 2010 (103 teams)

8. (silver)	UW	Karol Kurach, Krzysztof Pawłowski, Michał Pilipczuk
14.	UWr	Władysław Kwaśnicki, Przemysław Pająk, Przemysław Uznański

ICPC 2011 (103 teams)

9. (bronze)	UJ	Robert Obryk, Adam Polak, Maciej Wawro
13.	UW	Tomasz Kulczyński, Jakub Pachocki, Wojciech Śmietanka
42.	UWr	Krzysztof Piecuch, Damian Rusak, Łukasz Zatorski

ICPC 2012 (112 teams)

2. (gold)	UW	Tomasz Kulczyński, Jakub Pachocki, Wojciech Śmietanka
18.	UJ	Robert Obryk, Adam Polak, Maciej Wawro
18.	UWr	Marcin Dublański, Jarosław Gomułka, Karol Pokorski
36.	PP	Konrad Baumgart, Piotr Żurkowski, Tomasz Żurkowski

ICPC 2013 (120 teams)

6. (silver)	UW	Marcin Andrychowicz, Maciej Klimek, Tomasz Kociumaka
9. (bronze)	UJ	Jakub Adamek, Grzegorz Guśpiel, Jonasz Pamuła
27.	UWr	Anna Piekarska, Damian Straszak, Jakub Tarnawski

ICPC 2014 (122 teams)

5. (silver)	UW	Jarosław Błasiok, Tomasz Kociumaka, Jakub Oćwieja
13.	UWr	Anna Piekarska, Tomasz Syposz, Jakub Tarnawski
45.	UJ	Jakub Adamek, Igor Adamski, Piotr Bejda

ICPC 2015 (128 teams)

12. (bronze)	UW	Kamil Dębowski, Błażej Magnowski, Marek Sommer
13.	UWr	Bartłomiej Dudek, Maciej Dulęba, Mateusz Gołębiewski
15.	UJ	Piotr Bejda, Grzegorz Guśpiel, Michał Seweryn

ICPC 2016 (128 teams)

5. (silver)	UW	Wojciech Nadara, Marcin Smulewicz, Marek Sokołowski
9. (bronze)	UWr	Barłomiej Dudek, Maciej Dulęba, Mateusz Gołębiewski
14.	UJ	Krzysztof Maziarz, Michał Zając, Szymon Łukasz

ICPC 2017 (133 teams)

2. (gold)	UW	Wojciech Nadara, Marcin Smulewicz, Marek Sokołowski
20.	UWr	Paweł Michalak, Tomasz Syposz, Michał Łowicki
34.	UJ	Vladyslav Hlembotskyi, Krzysztof Maziarz, Michał Zając

ICPC 2018 (140 teams)

14.	UW	Kamil Dębowski, Mateusz Radecki, Marek Sommer
31.	UJ	Vladyslav Hlembotskyi, Franciszek Stokowacki, Michał Zieliński

ICPC 2019 (135 teams)

4. (gold)	UW	Jakub Boguta, Konrad Paluszek, Mateusz Radecki
6. (silver)	UWr	Anadi Agrawal, Michał Górniak, Jarosław Kwiecień

Central European Regional Contest

Since the 1995/1996 academic year, Polish student teams have been fighting to advance to the finals in the regional competition — Central European Regional Contest (in short CERC). In CERC take part student teams from IT departments of leading universities from the following Central European countries: Austria, Croatia, Czech Republic, Hungary, Poland, Slovakia and Slovenia. The CERC region is very strong and often the advancement to the world finals from this region is a guarantee of success, as evidenced by three championship titles for teams from this region — the University of Warsaw won twice, and once the Charles University in Prague. Below is a list of the winners of the Central European Regional Contest.

	Year and place of contest		*European Champions in Collegiate Programming*
1	1995	Bratislava	University of Warsaw
2	1996	Bratislava	Comenius University in Bratislava
3	1997	Bratislava	Comenius University in Bratislava
4	1998	Prague	Comenius University in Bratislava
5	1999	Prague	Charles University in Prague
6	2000	Prague	University of Warsaw
7	2001	Warsaw	University of Warsaw
8	2002	Warsaw	University of Warsaw
9	2003	Warsaw	University of Warsaw
10	2004	Budapest	University of Warsaw
11	2005	Budapest	University of Warsaw
12	2006	Budapest	University of Warsaw
13	2007	Prague	University of Warsaw
14	2008	Wroclaw	University of Warsaw
15	2009	Wroclaw	University of Warsaw
16	2010	Wroclaw	University of Warsaw
17	2011	Prague	University of Warsaw
18	2012	Krakow	Comenius University in Bratislava
19	2013	Krakow	Comenius University in Bratislava
20	2014	Krakow	University of Zagreb
21	2015	Zagreb	University of Warsaw
22	2016	Zagreb	University of Warsaw
23	2017	Zagreb	University of Warsaw
24	2018	Prague	University of Warsaw

Polish Collegiate Programming Contest

Although the first starts of Polish teams in programming competitions looked like an ad-hoc mobilization, since the 1996/1997 academic year, preparations to participate in the championships and the selection of representative teams have been systematic. Usually, representation teams from individual universities are formed in local university qualifying rounds. The best teams compete in the Polish Collegiate Programming Contest (in Polish *Akademickie Mistrzostwa Polski w Programowaniu Zespołowym*; in short AMPPZ), which usually take place at the end of October. The first AMPPZ competition was held in 1996 at the Poznan University of Technology. The originators of the competition were Prof. Jerzy Nawrocki from the Poznan University of Technology and Prof. Jan Madey from the University of Warsaw. By 2018, 23 editions of AMMPZ were carried out, in which a team from the University of Warsaw won each time.

Year and place of contest			*Polish Champions in Collegiate Programming*
1	1996	Poznan (PP)	Marcin Mucha, Jakub Pawlewicz, Krzysztof Sobusiak
2	1997	Wroclaw (PWr)	Bartosz Klin, Marcin Mendelski-Guz, Marcin Sawicki
3	1998	Warsaw (UW)	Bartosz Klin, Marcin Sawicki, Marcin Stefaniak
4	1999	Warsaw (UW)	Jakub Pawlewicz, Marcin Stefaniak, Tomasz Waleń
5	2000	Warsaw (UW)	Tomasz Czajka, Andrzej Gąsienica-Samek, Marcin Stefaniak
6	2001	Wroclaw (UWr)	Łukasz Kamiński, Eryk Kopczyński, Tomasz Malesiński
7	2002	Wroclaw (UWr)	Łukasz Kamiński, Tomasz Malesiński, Paweł Parys
8	2003	Wroclaw (UWr)	Tomasz Malesiński, Krzysztof Onak, Paweł Parys
9	2004	Krakow (UJ)	Marcin Michalski, Paweł Parys, Bartłomiej Romański
10	2005	Krakow (UJ)	Marek Cygan, Marcin Pilipczuk, Piotr Stańczyk
11	2006	Krakow (UJ)	Marek Cygan, Marcin Pilipczuk, Filip Wolski
12	2007	Poznan (PP)	Marek Cygan, Marcin Pilipczuk, Filip Wolski
13	2008	Poznan (PP)	Tomasz Kulczyński, Jakub Łącki, Piotr Mikulski
14	2009	Poznan (UAM)	Jakub Łącki, Piotr Niedźwiedź, Wojciech Śmietanka
15	2010	Poznan (UAM)	Tomasz Kulczyński, Jakub Pachocki, Wojciech Śmietanka
16	2011	Warsaw (UW)	Tomasz Kulczyński, Jakub Pachocki, Wojciech Śmietanka
17	2012	Warsaw (UW)	Jarosław Błasiok, Mirosław Michalski, Jakub Oćwieja
18	2013	Warsaw (UW)	Jarosław Błasiok, Tomasz Kociumaka, Jakub Oćwieja
19	2014	Warsaw (UW)	Wojciech Nadara, Grzegorz Prusak, Marcin Smulewicz
20	2015	Wroclaw (UWr)	Kamil Dębowski, Błażej Magnowski, Marek Sommer
21	2016	Wroclaw (UWr)	Wojciech Nadara, Marcin Smulewicz, Marek Sokołowski
22	2017	Wroclaw (UWr)	Kamil Dębowski, Mateusz Radecki, Marek Sommer
23	2018	Wroclaw (UWr)	Jakub Boguta, Konrad Paluszek, Mateusz Radecki

2011

16th Polish Collegiate Programming Contest
Warsaw, October 28–30, 2011

A

Arithmetic Rectangle

★★★

Task author: Jakub Radoszewski
Solution description: Jakub Radoszewski
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2011/ary`

We are given a grid consisting of $n \times m$ unit squares. Each of the unit squares contains a single integer. In this task we are interested in *arithmetic rectangles* lying on the grid, i.e., rectangles composed of unit squares such that numbers in every row and every column form arithmetic sequences. Recall that an arithmetic sequence is a sequence in which any two consecutive terms differ by the same amount.

In addition, we aim to find the largest arithmetic rectangle, i.e., the one covering the most unit squares. For example, the largest arithmetic rectangle on the grid below consists of nine unit squares:

5	3	5	7
2	4	4	4
3	5	3	1
6	3	2	4

Input

In the first line of the input there is a single integer t ($1 \leqslant t \leqslant 10\,000$), specifying the number of test cases that follow. The description of each test case begins with a line with two integers n and m ($1 \leqslant n, m \leqslant 3\,000$). In each of the following n lines there are m integers from the range $[0, 10^9]$. These are numbers contained in subsequent unit squares of the grid. The size of each input file will not exceed 20 MB.

Output

Your program should output t lines with answers for the consecutive test cases. The answer for a single test case is one integer equal to the number of unit squares contained in the largest arithmetic rectangle that can be found on the grid described in the test case.

Example

For the input data:

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

the correct result is:

```
9
6
```

Solution

To describe problem *Arithmetic Rectangle* we use two helper problems, whose solutions are more or less known. In the solution we will reduce each subsequent problem to the previous one.

Air crashes

In this problem we are given an n-element sequence of integers $a_1, \ldots, a_n$. For each element of the sequence we would like to find the closest smaller element that lies to the left of it. Formally, for every $i \in \{1, \ldots, n\}$ we look for the greatest j, $j < i$, such that $a_j < a_i$. For the value to be always well-defined, we add to the sequence a sentinel element $a_0 = -\infty$ (see figure 1). The original problem of air crashes has rather drastic formulation. We are given numbers of casualties in air crashes which were happening in subsequent years. For each crash we would like to calculate, what is the period in the past that there was no equally severe crash. In this version we would find closest *larger* element to the left. Let's go back, however, to our initial formulation.

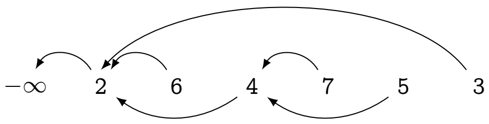


Figure 1. Solution to the air crash problem for the sequence $2, 6, 4, 7, 5, 3$.

The air crash problem is easy to solve in linear time, using a method of *following the arrows.* The idea is clear: we process the sequence from left to right and we assign to each element an arrow that leads to the closest smaller element. For given i we begin with checking if $a_{i-1} < a_i$. If yes, then we know that the arrow from a_i leads to a_{i-1}. If not, then we go to the first element smaller than a_{i-1}, that is exactly along the arrow from a_{i-1}. We continue this procedure until we find

$$\begin{tikzcd}
-\infty & 2 \arrow[l, bend right] & 6 \arrow[l, bend right] & 4 \arrow[ll, bend left] & 7 \arrow[l, bend right] & 5 \arrow[ll, bend left] & 3 \arrow[l, bend right]
\end{tikzcd}$$

Figure 2. Calculating result from $a_6 = 3$.

an element smaller than a_i. For example, figure 2 depicts determining the arrow from element $a_6 = 3$.

To justify that the algorithm works in linear time, is enough to show that we move along each arrow at most once. Suppose then that during determining the arrow for element a_i we move along arrow originating in some a_j $(j < i)$. That means that $a_j \geqslant a_i$, since otherwise a_j would be a candidate for result to a_i. Let's denote by a_k the resulting element for a_i. Later in the algorithm we would consider elements $a_{i+1}, a_{i+2}, \ldots$ We show that for any of them, moving along the arrows, we will not encounter element a_j. To reach element a_j we would have, in some moment, jump over element a_i (if we would encounter element a_i, then for sure we would not reach a_j in the same round). But the arrow over a_i enters an element smaller than a_i, and the earliest such element is a_k. Therefore, such arrow cannot go to a_j which is located to the right of a_k.

Plot

Now let's consider the second helper problem. We are given a board (array A) consisting of n rows and m columns, filled with zeros and ones. We need to find a rectangular fragment of this board filled with only ones, with the greatest area (see figure 3). The title of this section comes from the fact that such problem have appeared on the 9th Polish Olympiad in Informatics as a task *Plot*. For simplicity of language we will call our problem as finding the *optimal plot*.

1	0	0	1	1
0	1	1	1	1
1	1	1	1	0
1	1	1	1	0
0	1	1	0	1

Figure 3. The optimal plot in this case has area of 9.

The problem *Plot* can be solved in linear time complexity, namely $O(nm)$. We describe here one such solution, somewhat different than the model solution from the 9th Olympiad.

We denote by (i, j) a cell of the board which is located on the i-th row from the top and the j-th column from the left. At the beginning, for each cell (i, j) we calculate an auxiliary value $D[i, j]$ being the number of subsequent cells filled with ones located below this cell, including the cell (i, j). It is easy to calculate such values in time $O(nm)$, moving from the bottom to the top of the board. We use the fact that if $A[i,j] = 1$, then $D[i,j] = D[i+1,j]+1$, and otherwise of course $D[i,j] = 0$ (see figure 4).

1	0	0	4	2
0	4	4	3	1
2	3	3	2	0
1	2	2	1	0
0	1	1	0	1

Figure 4. Auxiliary array D.

Now it is time for the key observation. The optimal plot could be constructed as follows: we consider some cell (i,j) (in our solution we will test all possibilities for choosing (i,j)) and we calculate a rectangle containing this cell in its upper row, of height $D[i,j]$ and reaching to the right and to the left as far as possible, i.e. until reaching the board's border or a cell with a zero. Let's justify that for some cell (i,j) we will in fact obtain an optimal plot. The resulting plot must be maximal in the sense that each of its side either coincides with board's border or is adjacent to a cell containing a zero. In particular, it applies to the lower side of the plot. Let (i',j) denotes a cell located on the lower row of the plot which is adjacent to a cell with a zero (or any cell if the lower side if it coincides with board's border). Let i denotes the number of the top row of the plot. Then the plot has height $D[i,j]$, contains field (i,j) in its upper row and cannot be extended neither to the left, nor to the right, which is consistent with the observation. For example, on figures 3 and 4 cell (i,j) of the optimal plot of area 9 is its top-right corner.

The above observation can be alternatively formulated as follows: for every cell (i,j) with non-zero $D[i,j]$ we look for the closest cells in the same row for which values D are smaller than $D[i,j]$, i.e. such indices j' and j'' that

$$j' < j < j'', \quad D[i,j'], D[i,j''] < D[i,j], \quad j' \text{ is maximal}, \; j'' \text{ is minimal}.$$

The indices j' and j'' are numbers of columns which are located just behind the sides of our plot. We assume for simplicity that $D[i,0] = D[i,m+1] = 0$. Then the answer to the problem is maximum from products of form $(j'' - j' - 1) \cdot D[i,j]$ for all cells (i,j). Note, that to calculate j' and j'' we can simply use the solution to air crashes problem, applied to the i-th row of array D, only once from left to right (calculating j') and then from right to left (calculating j''). Using the previous algorithm row by row, we get an algorithm running in time $O(nm)$.

Arithmetic rectangle

Recall that in the problem from the contest, we have a board of size $n \times m$ consisting of non-negative integers. We are searching in it for an arithmetic rectangle of maximal area, where an arithmetic rectangle is a rectangle in which numbers in every row and in every column form arithmetic sequences. An example of such rectangle is depicted on figure 5.

1	3	7	11	15
2	4	6	8	10
5	5	5	5	5
8	6	4	2	0
6	3	0	4	8

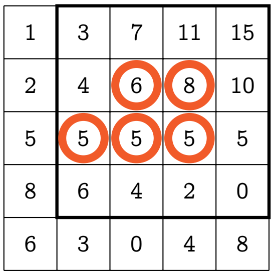
Figure 5. Maximal arithmetic rectangle has area of 16.

For a start, we handle rectangles of width 1 (i.e. contained in one row). We see, that every row of the board can be partitioned on maximal arithmetic sequences, such that every one has length at least two and every two consecutive ones have exactly one common element. For instance, the bottom row from the array on figure 5 can be partitioned into sequences $(6, 3, 0)$ and $(0, 4, 8)$, and the top row into sequences $(1, 3)$ and $(3, 7, 11, 15)$. Using such representation, in time $O(nm)$ we can easily find the longest arithmetic rectangle of width 1. Similarly we consider rectangles of length 1, width 2 (how?) and length 2.

From now on we are interested only in rectangles whose every side has length at least 3. We will reduce finding such rectangles to the problem *Plot*.

Let's mark with a circle every cell on the board such that a square of size 3, that contains this cell in the center, is an arithmetic rectangle (see figure 6). It turns out that the rectangle of both sizes no smaller than 3 is arithmetic if and only if all its cells contained in it (maybe except its interior border of width 1) are marked. Indeed, if the rectangle is arithmetic, then in particular all squares of size 3 contained in this rectangle are arithmetic. For the other direction: if in the rectangle two adjacent cells are marked, then the arithmetic sequences in their squares 3×3 will join in longer arithmetic sequences exactly as needed. Continuing this reasoning, we can show that every subrectangle of maximal length and width 3 or maximal width and length 3 is arithmetic. Thus we have that every row and every column of the rectangle forms an arithmetic sequence, thus the whole rectangle is arithmetic.

To conclude: we handled narrow arithmetic rectangles separately, and the finding thick enough arithmetic rectangle of maximal area was reduced to finding maximal rectangles consisting only of marked cells. This last problem corresponds almost to problem *Plot*, but there is a little catch here. Namely in that problem we look for exactly one plot, that with the greatest area. But after our transformation we no longer can guarantee that such plot, extended by a border of width 1, will give a rectangle of maximal area. It could happen that for some positive integers a, b, c, d we have $ab > cd$, but $(a + 2)(b + 2) < (c + 2)(d + 2)$ (can you show such example?). Fortunately, it is easy to get out of this situation. Maybe optimal arithmetic rectangle does not correspond to an optimal plot, but for sure it corresponds to some *maximal* plot, i.e. which cannot be extended in any direction. An now it



1	3	7	11	15
2	4	6	8	10
5	5	5	5	5
8	6	4	2	0
6	3	0	4	8

Figure 6. Marked cells are in centers of arithmetic squares 3×3.

is enough to recall that our solution to problem *Plot* in fact was considering all possible maximal plots (when finding this of the greatest area), then we can really use it for finding optimal arithmetic rectangle. Finally, we get a solution that runs in optimal time $O(nm)$.

B

Bytean Road Race ★★★★

Task author: Jakub Łącki
Solution description: Jakub Łącki
Available memory: 64 MB
`https://oi.edu.pl/en/archive/amppz/2011/baj`

The Bytean Road Race is to be held tomorrow in Bytetown city center. The streets of Bytetown form a regular grid: all of them go from the south to the north or from the west to the east. The participants of the race are only allowed to use some given parts of the roads.

Byteasar's task is to place the event's sponsors' banners on some of the crossings, and to do that he has to examine the route map of the race. The map depicts the segments of streets that the runners are allowed to use. There are n crossings and m vertical and horizontal road segments marked on it. Every segment starts and ends at some crossing and does not contain any other crossings. The road segments may only intersect at the crossings.

The crossings are numbered from 1 to n. The race will start at the crossing number 1 and finish at the crossing number n. The runners can pick their routes themselves, however they are required to run only south and east and only along the segments marked on the map. The road segments on the map are chosen in such a way that, by running in accordance with the rules, one can get to the finish from any place, and every place is reachable from the start crossing.

Byteasar wants to place the banners in a way that ensures that no runner will see a banner of the same sponsor twice. Therefore, for some pairs of crossings, Byteasar has to check whether it is possible for the route of some participant to go through both crossings. The race takes place tomorrow, so a program that will help him do the task is urgently needed.

Input

The first line of the input contains three integers n, m and k ($2 \leqslant n \leqslant 100\,000$, $1 \leqslant m \leqslant 200\,000$, $1 \leqslant k \leqslant 300\,000$). They specify the number of crossings on the race route, the number of marked segments on the map, and the number of pairs of crossings to be checked, respectively.

The following n lines describe the locations of the crossings. The i-th of these lines contains the coordinates of the i-th crossing represented by two integers x_i, y_i ($-10^9 \leqslant x_i, y_i \leqslant 10^9$). Moreover, $x_1 \leqslant x_n$ and $y_1 \geqslant y_n$. There will be at most one crossing at any given point. The axes of the coordinate system correspond to the real world cardinal directions in the natural way: the OX axis goes towards the east, and the OY axis — towards the north.

Each of the following m lines contains a description of a single segment on the map, consisting of a pair of integers a_i, b_i ($1 \leqslant a_i, b_i \leqslant n$, $a_i \neq b_i$), specifying the numbers of crossings connected by the segment. All of these segments are either vertical or horizontal and they may only intersect at common endpoints (the crossings).

The next k lines contain the descriptions of pairs of crossings to be checked. The i-th of these lines contains two integers p_i, q_i ($1 \leqslant p_i, q_i \leqslant n$, $p_i \neq q_i$).

Output

Your program should output k lines. The i-th of these lines should contain the word `TAK` (i.e., *yes* in Polish) if it is possible for the route of some participant to go through both crossings p_i and q_i (in any order). Otherwise the output should be `NIE` (*no* in Polish).

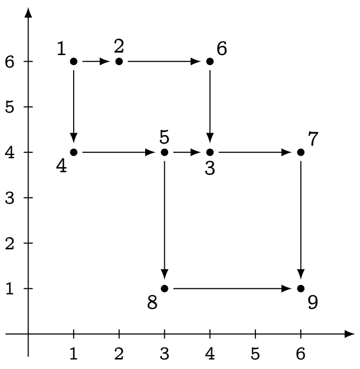
Example

For the input data:

```
9 10 4
1 6
2 6
4 4
1 4
3 4
4 6
6 4
3 1
6 1
1 2
4 1
2 6
3 6
5 4
5 3
5 8
3 7
7 9
9 8
4 8
2 5
8 7
7 6
```

the correct result is:

```
TAK
NIE
NIE
TAK
```



Solution

At the beginning let us rotate the Bytetown's map by 135° to the left. In effect the start crossing will be positioned at the very bottom of the map, the finish crossing at the top, and the streets will leave the crossing in two diagonal directions: to the top-right and to the top-left (see figure 1). The rotation does not change the answer anyhow, but it will be easier for us to work with such rotated map.

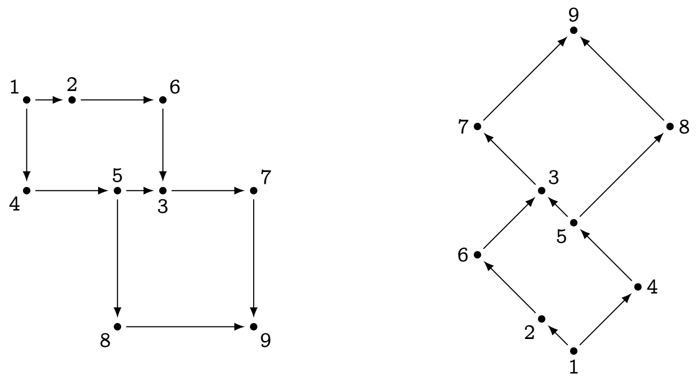


Figure 1. Bytetown's map from the task example (left) and the same map after rotating it by 135° (right).

We treat the Bytetown's map as a directed graph. In this graph we have n vertices representing the crossings and m edges corresponding to road segments. Let us fix some vertex v. The *left path* coming out from vertex v we call a path that starts in v and from every subsequent vertex, if it is possible, leaves to the left. If from some vertex there is only edge to the right, then the left path goes along this edge. The left path ends in the vertex corresponding to the finish. Analogously we define the *right path*. The notions of the left and right paths will be a foundation for our effective solution. Denote by L_v the left path from vertex v, and by R_v the right path from vertex v. In the example from figure 1 path L_4 goes through vertices $4, 5, 3, 7, 9$, and the path R_4 through vertices $4, 5, 8, 9$.

Suppose that we want to check whether we can go from vertex $p = (x_p, y_p)$ to vertex $q = (x_q, y_q)$. Let us draw a horizontal line h passing through vertex q. Assume that vertex p lies below the line h, otherwise a path from p to q obviously does not exist (unless $p = q$). Let us draw the paths L_p and R_p from vertex p. If we can go from vertex p to vertex q then the following condition is satisfied:

Condition 1. The path L_p intersects line h in vertex q or to the left from it, and the path R_p intersects line h in vertex q or to the right from it.

Why is that? We know that there exists a path from p to q (call it S_{pq}) that intersects line h in point q. The left path from p can never find itself to the right from S_{pq}. Analogously, the right path starting in p for sure will not be going to the left of S_{pq}. The example is on the left side of figure 2.

Moreover, the converse fact follows: if the condition 1 is satisfied, then there exists a path from p to q. To justify that, we consider a path S_{sq} from the starting

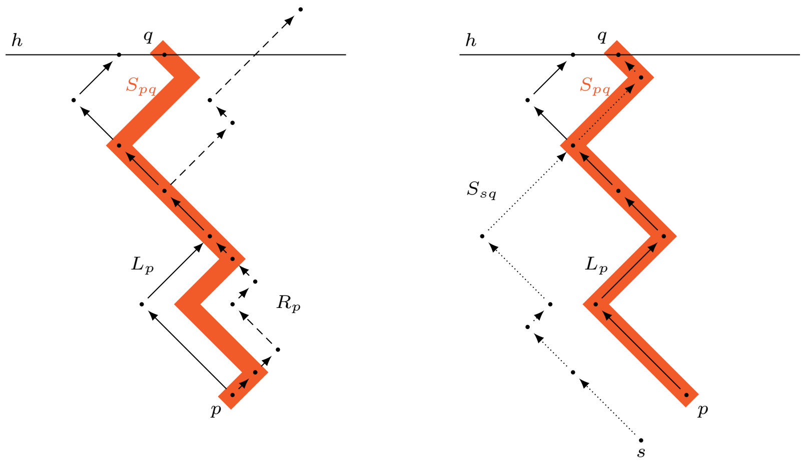


Figure 2. Left: paths from condition 1 and the line h. The path S_{pq} must lie in the area enclosed by paths L_p and R_p. Right: the path S_{sq} intersects L_p, thus we can construct a path from p to q.

vertex s to vertex q (the right part of figure 2). Such path exists for sure, since from the starting vertex s we can go to any other vertex. Moreover, each path going to vertex q which begins in vertex p or below, must intersect path L_p or R_p. Assume that path S_{sq} intersects path L_p. Then, to go from p to q, we follow along path L_p until it intersects the path S_{sq}, and then we change to path S_{sq} and follow along it to vertex q. All it means that to check whether from p we can go to q, it is enough to check the condition 1.

The structure of left and right paths

Note that left and right paths have a very strong structure. The left path coming out from vertex v can be constructed as follows. We go along the first edge (to the left, if it is possible, otherwise to the right), reaching vertex w, and from this moment the remaining part of the path L_v coincides with the path L_w.

Now we construct a two-dimensional array *left*, which will be used to quickly navigate left paths. We will use a method called *binary lifting*: for every vertex v and every i such that $0 \leqslant i \leqslant \lfloor \log_2 n \rfloor$, in the cell $\mathit{left}[v, i]$ we want to write the number of vertex that is reachable from vertex v if we make 2^i steps along the path L_v. Analogous array *right* is constructed for right paths. For convenience, in the finish vertex we add an edge that goes from this vertex to the same. Thanks to that all values in the arrays are well defined.

For every vertex v the value $\mathit{left}[v, 0]$ is calculated easily: it is enough to see where leads the first edge of path L_v. Next our algorithm runs in $\lfloor \log_2 n \rfloor$ phases. In the i-th phase we assume that we have already calculated values $\mathit{left}[\cdot, i-1]$, and on this basis we want to calculate $\mathit{left}[\cdot, i]$. We follow the idea: to check where we end up after 2^i steps along path L_v, first we make 2^{i-1} steps along it. The vertex w which we reach in such way was calculated in the previous phase. Now from vertex w we make another 2^{i-1} steps along L_w, which coincides with the rest of L_v. This way we will know where we find ourselves after 2^i steps from v. In other words, it is enough to make an assignment

$$\mathit{left}[v, i] := \mathit{left}[\mathit{left}[v, i-1], i-1].$$

Since arrays *left* and *right* have $O(n \log n)$ cells each, we can fill them in total time $O(n \log n)$.

After filling both arrays, we can use them to check the condition 1. To this we use some kind of binary search. Suppose that we are interested in finding a point in which the path L_p intersects the line h.

Starting in vertex p we will follow the path L_p, making sure not to cross the line h. Subsequently for $i = \lfloor \log_2 n \rfloor, \lfloor \log_2 n \rfloor - 1, \ldots, 0$ we check whether after making 2^i steps along the left path from the current vertex, we end up above the line h. If yes, then we do nothing, otherwise, we perform these 2^i steps.

After this procedure, we will find the topmost vertex w on the path L_p that lies no higher than line h. Now it is enough to check, on which side of point q the first edge of path L_w intersects line h. We do similar thing for the path R_p, which allows us to check condition 1 and answer the query.

We got an algorithm, which after preprocessing in time $O(n \log n)$, answers each query in time $O(\log n)$.

Faster solution

Although the above solution is fast enough, we describe here an interesting algorithm working in time $O(n)$. At first we will deal with, it would seem, some different problem. Later we will show how it relates to our task.

Mutual positions of vertices in a tree

Consider a rooted binary tree in which every vertex has zero, one, or two children. If a vertex has two children, we assume that they are ordered: we call them left and right child. Let us distinguish two different vertices of the tree and call them a and b. How they can be positioned relative to each other?

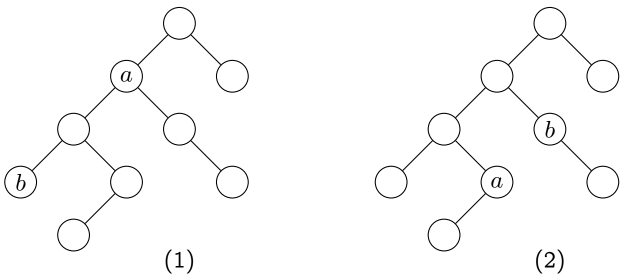


Figure 3. Two cases of mutual positions of vertices in a tree: (1) vertex a lies above vertex b, (2) vertex a lies to the left of vertex b. In two remaining cases we exchange the roles of vertices a and b.

Denote by S_a and S_b the paths from the root to vertices a and b, respectively. It turns out that there exist exactly four possible cases: vertex a can lie above vertex b (then it lies on the path S_b), below vertex b (then b lies on the path S_a), to the left of vertex b (when going from a to the root, we enter the path S_b from the left), or to the right of vertex b (when going from a to the root, we enter S_b from the right). It is easy to see that the first two cases are symmetrical, the same as the latter two cases. These cases are illustrated on figure 3. We show now, how we can quickly answer queries about mutual positions of the vertices.

Perform a depth-first search on the tree and for every vertex v write down time of enter I_v and time of leave O_v, i.e. the moment when we visit vertex v for the first time and the moment just after we traverse all vertices below vertex v. Thanks to that the mutual position of two different vertices a and b can be calculated as follows:

- a is above b if and only if $I_a < I_b$ and $O_b < O_a$,
- a is below b if and only if $I_b < I_a$ and $O_a < O_b$,
- a is to the left of b if and only if $O_a < I_b$,
- a is to the right of b if and only if $O_b < I_a$.

Thus after preprocessing in time $O(n)$ we can calculate relative positioning of vertices in the tree in constant time.

Linear-time solution

Now we are ready to describe a faster solution of our problem. We construct a *graph of left paths* which is a sum of left paths coming out from all vertices. It contains the same n vertices as Bytetown's map, but from each vertex (except finish) extends only one edge (it is the left edge if it exists, otherwise the right edge).

Note that such a graph is a binary tree whose root is a node with finish. That is why we have rotated the map in such a way that the finish is at the top — thanks to that the graph of left paths looks like every other decent tree in computer science: it has the root at the top and grows downwards. Denote this tree by T_L. Analogously we define a *graph of right paths* and we denote it by T_R.

Like before, we want to check, whether we can go from vertex p to vertex q. It is easy to guess, that our algorithm will check relative positions of these vertices in trees T_L and T_R. Once again we use help of a horizontal line h passing through vertex q.

Consider now possible mutual position of vertices p and q in tree T_L. First two cases are simple. If p lies below q, then the sough path exists. Also we know that p cannot lie above q in T_L, because it must have been above line h.

Assume now that p lies to the left of q in tree T_L. That means that if we look at the left paths L_p and L_q going out from p and q, then to a vertex in which these paths meet, the path L_p enters from the left, and L_q enters from the right. If we will backup both paths simultaneously, to cross the line h, then path L_q will go to q and on path L_p we will be on line h to the left of q. It is not hard to see that if p lies to the right of q in tree T_L, then the left path from p will intersect line h to the right of vertex q.

Sounds familiar, doesn't it? Indeed, that is one part of condition 1. If we add to this checking of relative positioning of vertices p and q in tree T_R, then we gen very effective way to checking condition 1.

Let us sum up our algorithm. First we construct trees T_L and T_R and perform preprocessing allowing us to check relative positioning of vertices in these trees. Both steps can be done in time $O(n)$. Next, to answer a query of existence of path from p to q, we check whether p does not lie above q and we check their relative positioning in trees T_L and T_R. All these checks can be done in constant time per query.

It turns out that the whole problem can be solve *optimally*. We do a linear-time preprocessing and later we answer each query in constant time. All of this is possible thanks to strict structure of our graph: it is planar and acyclic, moreover it contains exactly one source and one sink. In general graphs this problem is much harder: the best algorithm known, that answers queries in constant time, needs preprocessing in time $O(n^{2.38})$ and uses fast matrix exponentiation algorithm.

C

Will It Stop?

★

Task author: Jakub Łącki
Solution description: Jakub Łącki
Available memory: 64 MB
`https://oi.edu.pl/en/archive/amppz/2011/czy`

Byteasar was wandering around the library of the University of Warsaw and at one of its facades he noticed a piece of a program with an inscription “Will it stop?”. The question seemed interesting, so Byteasar tried to tackle it after returning home. Unfortunately, when he was writing down the piece of code he made a mistake and noted:

```
**while** $n > 1$ **do**
  **if** $n \bmod 2 = 0$ **then**
    $n := n/2$
  **else**
    $n := 3 \cdot n + 3$
```

Byteasar is now trying to figure out, for which initial values of the variable n the program he wrote down stops. We assume that the variable n has an unbounded size, i.e., it may attain arbitrarily large values.

Input

The first and only line of the input contains one integer n $(2 \leqslant n \leqslant 10^{14})$, for which we need to check whether the given program stops.

Output

In the first and only line of the output you should write a single word `TAK` (i.e., *yes* in Polish), if the program stops for the given value of n, or `NIE` (*no* in Polish) otherwise.

Example

For the input data:

```
4
```

the correct result is:

```
TAK
```

Solution

In the problem statement there is a fragment of a program, whose behaviour we need to analyze. Since we know the source code, we could just paste it to our solution and execute. If we run it for a given integer n, and it stops, then we immediately return an answer. The opposite case is slightly more difficult: how to be sure that the executed code will not stop? Let's start with a very brave (and not necessarily justifiable) assumption. If the code does not stop during, let's say 100 000 loop iterations, then we assume that it never won't stop.

Such an approach is easy to put into a solution, but it has two drawbacks. First of all, how do we know that it is enough to limit ourselves to 100 000 iterations? Secondly, if we represent number n as a 64-bit signed integer (e.g. we declare its type as `long long` in C++), then we will fall into a trap. It turns out, that even if the initial value of n, according to the problem statement, does not exceed 10^{14}, then after some iterations we could get a much bigger number. For instance, for initial value $n = 366\,713\,142\,269$ after around 160 iterations n reaches value $10\,010\,331\,589\,553\,303\,736$, that is above 10^{19}, whereas in a 64-bit signed integer we can represent numbers no larger than $2^{63} - 1 \approx 9{,}2234 \cdot 10^{18}$.

Let's approach that task differently. We see how the code behaves for every n from 1 to 20. It is convenient to get a help from the computer here. We will notice very quickly that the code stops for values n equal 1, 2, 4, 8 and 16 and for the remaining values it loops forever, since from some moment n assumes sequentially values $3, 12, 6, 3, 12, 6, 3, 12, 6, \ldots$ and so on forever. Thus it looks like the code stops for values being powers of two. Indeed, it requires a little thought to convince ourselves that if n is a power of two, then our code will divide variable n by 2 until it ends up with $n = 1$. Therefore, it n is a power of two, then the code will stop for sure. We could guess that for the remaining values of n the code will be running forever. This guess is correct, but why?

If the initial value of n is odd, the we perform the assignment $n := 3 \cdot n + 3$. It is easy to see that the new value of n will be divisible by 3. Moreover, from now on n will be always divisible by 3. Why is that? If n is divisible by 3, then no matter if we perform assignment $n := n/2$ or $n := 3 \cdot n + 3$ the new value will be still divisible by 3. Therefore in this case we will never reach $n = 1$! It remains to check what if n is even, but not a power of two. Then we will divide by 2 until it becomes odd. But we know that it will be an odd number greater than 1 (since we would end up at one only starting from a power of two). And since n became odd, then according to the previous analysis the code will not stop.

Therefore the whole problem boils down to testing whether n is a power of two. This condition can be tested quite easily: as long as n is divisible by 2, we perform this division. If after this procedure we get $n = 1$, then we had power of two at the beginning. Otherwise, given n was not a power of two.

Interestingly, this check can be performed even faster. We will use bitwise operators here. They perform operations on binary representations of integers and they are available in almost every popular programming language. The bitwise operator **and** (in C++ written using symbol `&`) works as follows: the result of a **and** b is an integer whose binary representation has 1s exactly on these positions on which both integer a and integer b have 1s. Meanwhile, bitwise operator **xor** (in C++ denoted by symbol `^`) calculates an integer in which 1s stand on these positions on which binary representations of two arguments differ. To test whether given n

is a power of two it suffices to check if

$$(n \textbf{ xor } (n-1)) \textbf{ and } n \text{ is equal to } n.$$

Why does this trick work? We left it as an exercise.

Collatz problem

On the facade of the library of the University of Warsaw indeed we can find a code similar to this presented in the problem statement. However, the problem statement reveals that Byteasar has made a mistake during copying the code at the piece of paper. The original code at the library looks as follows:

```
while $n > 1$ do
  if $n$ mod $2 = 0$ then
    $n := n/2$
  else
    $n := 3 \cdot n + 1$
```

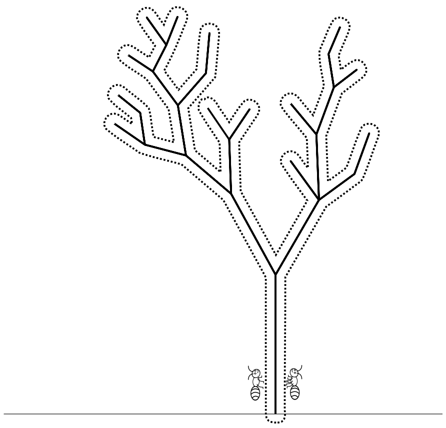
In the last row we have $n := 3 \cdot n + 1$ rather than $n := 3 \cdot n + 3$. Although the difference between these two codes looks small, it is in fact significant. In our problem we were able to show all numbers for which the code stops. Analogous problem for the code on the library wall is much harder. In 1937 German mathematician Lothar Collatz formulated a hypothesis that the above code stops for any positive n. Many mathematicians have worked on this hypothesis, but so far no one was able to settle it. It was validated using computers for all $n < 20 \cdot 2^{58} \approx 5{,}7646 \cdot 10^{18}$ and is generally believed true, but there is no formal proof that validates it. Thus if in our problem we would considering the original Collatz problem, the question *Will It Stop?* couldn't be answered for sure.

D

Ants

★★★★

Task author: Szymon Acedański
Solution description: Szymon Acedański, Tomasz Idziaszek
Available memory: 6 MB
`https://oi.edu.pl/en/archive/amppz/2011/drz`



Computer geeks like trees. Ants also like trees. Therefore we are given a tree with two ants walking on it — the Left Ant and the Right Ant — as shown in the figure above (the ants walk along the path depicted by a dotted line). They start their journey at the lower and of the trunk, on opposite sides. The Left Ant needs 2 seconds to walk along a single edge of the tree if walking from the root (upwards), and 1 second if walking towards the root (downwards). The Right Ant is two times faster. When the two ants meet, they both turn around and start walking in the opposite direction. If either of the ants steps from the tree to the ground, it immediately starts to climb up the opposite side of the trunk. Apart from that, the ants are so tiny that they would not be visible even under a microscope (they are intentionally depicted much larger in the figure). Your task is to write a program that computes the moment when the ants turn around for the second time.

Input

The first line of the input contains a single integer t ($1 \leqslant t \leqslant 1000$) representing the number of test cases described in the input.

The description of each test case consists of two lines. The first line contains an even integer n ($2 \leqslant n \leqslant 100\,000\,000$) specifying the number of edges in the tree. The second line holds a description of the tree. It is a string consisting of $\frac{n}{2}$ characters representing a $2n$-bit binary number written in a hexadecimal form (using digits and small letters from a to f). This number shows the Left Ant's path around the whole tree, assuming that the Right Ant stands still. The consecutive bits of this number (starting from the left) denote whether the Left Ant walks away from the root of the tree along the corresponding edge (bit 1) or towards the root along this edge (bit 0). The root has a trunk; that is, there is exactly one edge leading from the root of the tree.

The size of the input file does not exceed 50 MB, which is much more than the amount of memory available for your program.

Output

Your program should output t lines containing answers to the consecutive test cases. Each answer should represent the moment (in seconds) when the ants turn around for the second time, given as an irreducible fraction p/q (without any white space around /), where p and q are positive integers. If the answer is integer then, obviously, $q = 1$.

Example

For the input data:

```
1
28
fb1da30d1b7230
```

the correct result is:

```
282/5
```

The sample data corresponds to the figure above, and transforms to the following sequence of bits:

```
1111 1011 0001 1101 1010 0011 0000 1101 0001 1011 0111 0010 0011 0000
```

Solution

Then input specifications are written so as to make it impossible to store the description of the entire tree in memory. The algorithm from the model solution reads the description once without storing it. Here is a sketch of the solution:

- We read in the size of the tree; it gives us the number of "up" and "down" segments.

- While reading the description, we simulate the walk for the Left Ant until the first meeting with the Right Ant. For each edge, we can check if this is the place of the meeting since, knowing the distance traveled by the Left Ant, we know the distance the Right Ant would need to cover to the meeting point.

- After the meeting, both ants start to walk towards the root. We continue reading in the input, this time simulating the walk of the Right Ant. It turns out that the first to reach the root is. . . the Left Ant. This could sound surprising, since it is the Right Ant who moves faster. We will explain this phenomenon later.

- After the Left Ant reaches the root, we continue to simulate the walk of the Right Ant, until the second meeting point.

To see that above sketch can be developed into a complete algorithm, let us take a look at a more formal description of the model solution. A point on the tree can be described by a pair of real numbers (a, k), where a indicates the total number of edges traversed by the Left Ant from the root to this point (possibly traversing some edges in both directions, with those edges counted twice), and k indicates the height of the point (the number of edges from the root to the point). If it takes the Left Ant t_u seconds to climb "up" edge and t_d seconds to descend along a "down" edge, then this ant reaches (a, k) in

$$[a, k, t_u, t_d] := \frac{a+k}{2}\, t_u + \frac{a-k}{2}\, t_d$$

seconds. Therefore, the first meeting of the Left Ant with the Right Ant, (a_1, k_1), happens when

$$[a_1, k_1, 2, 1] = [2n - a_1, k_1, 1, \tfrac{1}{2}]. \tag{1}$$

Again, we can find this point by simulating the walk of the Left Ant while reading the tree description edge by edge. When the Left Ant is at point (a, k), which is the beginning of some edge, we can check if the meeting could happen on this edge itself. Let $b = 1$ if this is an "up" edge, and $b = -1$ otherwise. If the meeting is to happen after walking an ε piece of the edge then (1) is satisfied for

$$a_1 = a + \varepsilon, \quad k_1 = k + b\varepsilon,$$

therefore

$$\varepsilon = \frac{6n - 9a - k}{9 + b}.$$

Thus, if $0 \leqslant \varepsilon < 1$, then we have located the first meeting point. The point will be reached in

$$t_1 = [a_1, k_1, 2, 1] = \frac{3a_1 + k_1}{2}$$

seconds.

If we let the ants continue and return to the root, then the Left Ant would reach the root in $[a_1, -k_1, 2, 1] = t_1 - k_1$ seconds, and the Right Ant in

$$[2n - a_1, -k_1, 1, \tfrac{1}{2}] = t_1 - \frac{k_1}{2}$$

seconds. So we can see now that the Left Ant would be at the root before the Right Ant.

Now we can start looking for the second meeting point. We will do so by simulating the Right Ant's walk. After the Right Ant walks the distance ε' on the edge starting at (a', k'), it reaches the second meeting point (a_2, k_2), provided that

$$a_2 = a' + \varepsilon', \quad k_2 = k' + b'\varepsilon',$$
$$[a_1, -k_1, 2, 1] + [2n - a_2, k_2, 2, 1] = [a_2 - a_1, k_2 - k_1, 1, \tfrac{1}{2}],$$

thus

$$\varepsilon' = \frac{12n - 9(a' - a_1) + (k' - k_1)}{9 - b'}.$$

If $0 \leqslant \varepsilon' < 1$, then we have found the second meeting point. The time to reach this point is

$$t_2 = [a_2 - a_1, k_2 - k_1, 1, \tfrac{1}{2}] = \frac{3(a_2 - a_1) + (k_2 - k_1)}{4}$$

seconds. Therefore the total time from the first to the second meeting of the ants equals

$$t_1 + t_2 = \frac{3(a_1 + a_2) + k_1 + k_2}{4}.$$

Note that ε is a multiple of $\frac{1}{9+b}$, and simple algebraic manipulations reveal that ε' is a multiple of $\frac{9-b}{(9+b)(9-b')}$. This implies that the numbers a_1, a_2, k_1, k_2 are multiples of $\frac{1}{800}$. Therefore, all calculations can be carried out using 64-bit integers representing values multiplied by 800.

All of this results in an algorithm whose every step consists of reading one consecutive edge from the input and executing a constant number of operations. The running time of the algorithm is thus $O(n)$ with constant memory.

E

Gophers

★★

Task author: Jakub Radoszewski
Solution description: Jakub Radoszewski
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2011/eks`

Dick Dastardly wants to bedevil poor Bytean gophers. These nice little creatures live in holes in the upper parts of High Bytemountains.

Dick has found a mountain ridge with n gopher holes located along a straight line (for simplicity, we number the holes from 1 to n, from west to east). Dick plans to torture gophers using rock and roll music. He has bought m CD players, put a different Bytels' album in each of them and arranged the CD players along the ridge. It is known that played at full volume, the music from a CD player disturbs gophers located in holes distant by at most l meters from it.

Feeling troubled, the gophers asked you to check in which holes they will not be able to sleep well during this winter. They did not know that it was not yet the end of Dick's malice...

Dick Dastardly wants to make even more mess, and he will move the CD players from time to time. The gophers were able to steal Dick's secret plan and now they know precisely that on the morning of the i-th day Dick will take the CD player located p_i meters from the hole number 1 and will put it at a point located r_i meters from that hole. Help the gophers and count the number of holes in which they will not be able to fall asleep after each such operation.

Input

The first line of the input contains four integers n, m, d and l ($2 \leqslant n, m \leqslant 500\,000$, $1 \leqslant d \leqslant 500\,000$, $1 \leqslant l \leqslant 10^9$) representing the number of gophers' holes, the number of Dick's CD players, the number of days, and the range of a CD player, respectively.

The second line of the input contains $n-1$ integers $x_2, x_3, \ldots, x_n$ ($0 < x_2 < x_3 < \ldots < x_n \leqslant 10^9$) specifying the distances of the holes numbered $2, 3, \ldots, n$ from the hole number 1.

The third line contains m integers $z_1, z_2, \ldots, z_m$ ($0 \leqslant z_1 < z_2 < \ldots < z_m \leqslant 10^9$) specifying the distances of the consecutive CD players from the hole number 1. All the CD players are located to the east of this hole.

Next, d lines follow; the i-th of these lines contains two integers p_i and r_i ($0 \leqslant p_i, r_i \leqslant 10^9$, $p_i \neq r_i$) meaning that in the beginning of the i-th day Dick is going to move the CD player located p_i meters from the hole number 1 to the point located r_i meters to the east from that hole. You may assume that before such operation there is a CD player at the position p_i and there are no CD players at the position r_i.

Output

Your program should output $d + 1$ lines. The line number i (for $i = 1, 2, \ldots, d$) should contain one integer representing the number of holes in which no gopher would be able to sleep well during the night *before* the i-th Dick's operation. The last line should contain this number after the last Dick's operation.

Example

For the input data:

```
5 3 4 1
2 5 6 11
2 4 8
2 1
4 10
8 6
1 8
```

the correct result is:

```
2
3
3
5
3
```

Solution

The whole action in the problem happens on a line. The positions of gopher holes are described as points. At the same time, each CD player corresponds to some range. All ranges have the same length $2l$, thus to describe each range we also need only one point — in the problem statement it is the middle point of the range. The positions of gopher holes do not change, but Dick changes positions of his CD players. Our task is to simulate Dick's behaviour and calculate, after each of d movements, how many different holes are covered by ranges corresponding to CD players.

It turns out that the problem boils down to choosing suitable data structures, which will allow us to perform the simulation. Suitable, that is as simple as possible, but at the same time fast enough: we aim in time complexity of an operation that is logarithmical in terms of number of holes and CD players, that is n and m.

The key assumption is that we would like to maintain the number of holes covered by ranges during the whole time. Then printing the results is trivial. As it often comes in such problems, movement of a CD players can be divided into two stages: removal of a CD player and inserting it on a new position. Each of such stage is in some sense local: which holes will start or vanish to be covered depends only on ranges which are adjacent directly to the range of inserted or removed CD player. This is the case, because all ranges have the same length. We will use this property when constructing an efficient solution.

For keeping track of CD players we need a data structure that provides operations: insertion and removal of an element and finding the closest left and right neighbour of the element in the structure. For this aim perfectly fits a dictionary-like data structure implemented with balanced binary search trees, e.g. `set` from the C++ standard library (the problem statement guarantees that no two CD players will be placed in the same spot). The position of gopher holes do not change, thus we can simply keep them in an array sorted increasingly — exactly in the same way as they are given in the input. Data structures keeping CD players and holes will be denoted by M and N, respectively.

Let's follow what happens when we insert some CD player. Suppose that it will be placed in position x and its direct neighbours are on positions x' and x'' ($x' < x < x''$). The we must insert x into dictionary M, and add to the result all holes covered by range $[x-l, x+l]$ that were not covered by ranges $[x'-l, x'+l]$ and $[x''-l, x''+l]$. These are exactly holes in the range

$$[\max(x-l, x'+l+1), \min(x+l, x''-l-1)]. \tag{1}$$

Observe that the number of such holes can be found using two binary searches in array N. Each of this binary searches can be realized using `lower_bound` function from the standard library.

The operation of removal is analogous: this time we remove appropriate element from dictionary M and we subtract from the result all holes inside the range of form (1). In order for every CD player to have a predecessor and successor, we can add to dictionary M constant sentinels placed at positions $-l-1$ and 10^9+l+1 (coordinates of points from the problems are in range $[0, 10^9]$).

Each of d Dick's movements is simulated in time complexity of $O(\log m + \log n)$, where the first logarithm comes from operations on dictionary M, and the second one from binary searches in array N. For contestants fluent in using the C++ standard library, the algorithm is quite easy in implementation.

F

Laundry

★★

Task author: Szymon Acedański
Solution description: Jakub Radoszewski
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2011/fra`

A few friends have decided to do the laundry together. They are all very neat, so each day each of the friends wears one clean pair of socks and one clean shirt. The friends have put all dirty socks and shirts to their washing machine. Now they have started to plan how they will dry their laundry. To put this in order, they have decided that:

- every sock will be fastened to the string with a single clothespin,
- each shirt will be fastened with three clothespins,
- one person's socks should all be fastened with clothespins of the same color,
- one person's shirts should all be fastened with clothespins of the same color,
- clothes belonging to different persons may not be fastened with clothespins of the same color,
- apart from that, they wish to use the smallest possible number of clothespin's colors.

Now they have scattered all their clothespins on the floor and counted the number of clothespins of each color. Unfortunately, they were not able to figure out which colors should each of them use. Write a program that will help them with this problem.

Input

The first line of the input contains two integers n and k ($2 \leqslant n, k \leqslant 1\,000\,000$) specifying the number of friends and the number of clothespins' colors available. The second line contains n integers $d_1, d_2, \ldots, d_n$ representing the number of days each friend was collecting laundry ($1 \leqslant d_i \leqslant 1\,000\,000$). The third line contains k integers $l_1, l_2, \ldots, l_k$ representing the numbers of clothespins of respective colors ($1 \leqslant l_i \leqslant 4\,000\,000$).

Output

Your program should output the minimal number of clothespins needed to dry all the laundry. If it is not possible to dry all the laundry in the requested manner, your program should output a single word `NIE` (i.e., *no* in Polish).

Example

For the input data:

```
2 4
3 4
20 10 8 10
```

the correct result is:

```
3
```

whereas for the input data:

```
3 8
5 4 3
14 14 14 14 14 14 14 14
```

the correct result is:

```
NIE
```

Explanation of the first example: The first person needs 6 clothespins for her socks and 9 clothespins for her shirts. The second person needs 8 clothespins for her socks and 12 clothespins for her shirts. The second person should use the clothespins of the first color both for her socks and her shirts. The first person may then use, e.g., the clothespins of the second and the fourth color.

Solution

The task statement tells a story about n persons doing a laundry. For fastening his or her laundry the i-th person needs $5d_i$ clothespins. It could be clothespins of the same color, or it could be $3d_i$ clothespins of one color (shirts) and $2d_i$ clothespins of different color (socks). We have k colors of clothespins, and there are exactly l_j clothespins of color j. The colors assigned to different persons must be different. We want to assign the clothespins in such a way that we use the smallest possible number of different colors. It total we will use between n and $2n$ colors, and the more people get the only one color of clothespins, the better. We also need to check, weather the solution actually exists.

We will try to approach this problem in a greedy way, inviting the subsequent people to choose some clothespins from the set of ones unchosen by previous people. The people with more laundry are more picky, thus intuitively it would be profitable to invite them sooner, so they have more choice. It also seems sensible that every person, if it is possible, should choose one color of clothespins. Moreover, if in some moment there is more than one way of choosing the clothespins' colors, then of course it is better to choose a color which has as few clothespins as possible.

The following intuitions are a foundation to the following algorithm. We examine people in order of nonincreasing d_i and for each of them:

- if there is a color which has at least $5d_i$ clothespins, then among such colors we choose the one which has the smallest number of clothespins,

- otherwise, if there are two colors containing respectively at least $2d_i$ and $3d_i$ clothespins, then among these we choose two of the smallest numbers of clothespins,
- if none of above is true, then we end the algorithm with negative answer.

It turns out that the above algorithm correctly solves our problem.

Prove of correctness

We need to prove that if a solution exists, then our greedy algorithm will find it, and moreover, it will find a solution using the least possible number of clothespins.

Suppose then that a solution exists, and let OPT be some optimal solution, that is a solution that uses the least possible number of clothespins. On the other hand, by ALG we denote the assignment calculated by the greedy algorithm. (In theory it could be a partial assignment, if it was not possible to assign necessary number of clothespins for some person, and that was the reason of terminating the algorithm.) We will show that if OPT $\neq$ ALG, then we can transform OPT into ALG, keeping its correctness and not changing the number of used colors of clothespins. That will prove the optimality of solution ALG.

Assume, without loss of generality, that the persons are ordered according to the order of the greedy algorithm, in particular $d_1 \geqslant d_2 \geqslant \ldots \geqslant d_n$. Our prove will be done in steps; in each step we will modify the solution OPT, increasing the number of subsequent people who are treated the same in OPT and ALG.

Let i be the number of the first person, for whom OPT and ALG assigned clothespin's colors differently. We will show that we can change the solution OPT in such a way, that all people $1, \ldots, i$ are treated the same as in ALG.

Since OPT assigned some colors to person i, then it is not possible that ALG was not able to assign colors. Indeed, the greedy algorithm could choose the same colors that OPT chose. Similarly, it is not possible that OPT assigned only one color and ALG assigned two.

On the other hand, we know that these two solutions differ on the i-th person. First we will consider a case in which both solutions assign the same number of colors, but these colors are different. We assume that it is only one color in both solutions; a proof for two colors in analogous. Let in OPT it will be color p and in ALG color a. Since ALG always choose a suitable color of the smallest number of clothespins, then $l_a \leqslant l_p$. If solution OPT does not use color a, we can swap in this solution for the i-th person color p to a. If OPT uses clothespins of color a by assigning them to some person j, then we have $j > i$ (since for $j < i$ OPT assigns the same colors as ALG) and we can swap in OPT colors for people i and j. The number of clothespins of color p is sufficient for the j-th person, because from $j > i$ we have $d_j \leqslant d_i$. On the other hand, the number of clothespins of color a is sufficient for the i-th person, because ALG makes such assignment.

We are left with the last case in which solution OPT assigns to the i-th person two colors, let's say p and q (for shirts and socks, respectively), and in ALG it assigns only one color, let's say a. If in OPT color a was free, then we could assign it to the i-th person instead of colors p and q, reducing the number of used colors in effect — thus it is not possible, because it would contradict the optimality of OPT. That means that color a in solution OPT must have been used, let's say by person j (again $j > i$). Like before, we want to assign this color to person i. We have different cases, depending on how many clothespins of color a the j-th person has. If it is $2d_j$ clothespins, we can instead of color a give this person clothespins of color q (since $d_j \leqslant d_i$). If it is $3d_j$, then instead of color a, we give him or her color p. Note, that in both cases the number of used colors decreased, which contradicts the optimality of OPT. Therefore both of these cases cannot happen (but we could handle them anyway). In the only situation that could take place, the j-th person has in OPT $5d_j$ clothespins of color a. Then we assign to him or her $3d_j$ clothespins of color p and $2d_j$ of color q. In the effect the number of used colors does not increase and the i-th person has the same assignment as in ALG. That concludes the proof.

Implementation

At the end let's think a little bit, how to implement this solution efficiently. We need to sort people in the problem nonincreasingly by values d_i; we do it in time $O(n \log n)$. To represent clothespins of various colors, we need a data structure that provides two types of operations:

- find a color with smallest number of clothespins, but having at least x of them,
- remove all clothespins of certain color.

Both container `set` from C++ standard library and a static range tree (or counting tree, or Fenwick tree) allows performing these operations in time $O(\log k)$. Initializing the data structure takes time $O(k)$. Therefore we get an implementation of the greedy solution in total running time $O(k + n(\log k + \log n))$.

G

Bits Generator

★★★★

Task author: Tomasz Idziaszek
Solution description: Tomasz Idziaszek
Available memory: 64 MB
`https://oi.edu.pl/en/archive/amppz/2011/gen`

Byteasar likes to play with his random (well, actually pseudorandom) bits generator, which he has found on his computer. This generator works in a very simple way. The moment the computer is turned on, an integer in the range between 0 and $m-1$ is chosen automagically. This integer is called the *seed* of the generator; we will use variable z to represent it. Then, in order to generate a random bit, the following function is called. It computes a new value of the seed which is then used to generate a single bit:

```
$z := \lfloor (z \cdot a + c)/k \rfloor \bmod m$
**if** $z < \lfloor m/2 \rfloor$ **then**
  **return** 0
**else**
  **return** 1
```

The numbers a, c, k are some constants. Byteasar has called this function n times and has thus obtained a sequence of bits $b_1, b_2, \ldots, b_n$. Now he is wondering what is the number of different possible values of the initial seed.

Input

The first line of the input contains five integers a, c, k, m and n ($0 \leqslant a, c < m$, $1 \leqslant k < m$, $2 \leqslant m \leqslant 1\,000\,000$, $1 \leqslant n \leqslant 100\,000$). The second line contains an n-character string consisting of digits 0 and 1; the i-th digit of the string represents the bit b_i.

Output

You should output one integer representing the number of integers from the range between 0 and $m-1$ which could have been the initial seed of the generator.

Example

For the input data:

```
3 6 2 9 2
10
```

the correct result is:

```
4
```

Explanation of the example: The initial seed of the generator could have been equal to 1, 2, 7 or 8.

Solution

Although it could not be visible at the first sight, but in the solution we will use some knowledge from graph theory and text algorithms. We will use graphs to describe the state space of the generator, whereas for finding the initial seed of the generator we will use a dictionary of basic factors or (in a faster solution) an automaton for finding pattern in text.

The state of the bits generator is described by one integer z in range from 0 to $m-1$. Every time the function from the problem statement is called, it changes the state to $f(z) = \lfloor (z \cdot a + c)/k \rfloor \bmod m$ and generates a bit $b(z)$. The bit $b(z)$ is equal to 0 if $f(z) < \lfloor m/2 \rfloor$, or equal to 1 otherwise. Since numbers a, c and k which describe the function f are fixed, both values, the generated bit $b(z)$ as well as the new state $f(z)$, depend only on the current state z.

Therefore the bits generator can be represented as a directed graph (see figure 1). It has m vertices $v_0, v_1, \ldots, v_{m-1}$ corresponding to possible states of the generator, and from every vertex v_i there is exactly one directed edge to vertex $v_{f(i)}$. Additionally, each vertex v_i has a label $b(i)$.

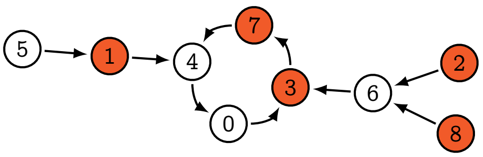


Figure 1. Directed graph with 9 vertices, which models the bits generator from the sample input. The vertices with labels 1 have been marked.

Let us denote by $p(v_i, n)$ a sequence of labels on a n-vertex path starting in vertex v_i. Note that the path is determined uniquely, since from every vertex we have exactly one outgoing edge. Our problem can be formulated as follows: we need to count the number of such vertices v_i that the sequence of labels $p(v_i, n)$ is identical with the sequence of bits $b_1, \ldots, b_n$ given in the input.

An algorithm of time complexity $O(nm)$ is obvious here (and it does not even need the above reduction for graph representation). For every vertex v_i it is enough to generate sequence $p(v_i, n)$ and test whether it is equal to $b_1, \ldots, b_n$.

Dictionary of basic factors

We will obtain a faster algorithm by using a data structure called *dictionary of basic factors*. For every power of two 2^j, where $0 \leqslant j \leqslant N$ and $N = \lfloor \log_2 n \rfloor$ we will be assigning *identifiers* to sequences of labels of length 2^j in such a way that two sequence will be assigned with the same identifier if and only if they are equal. The identifiers will be integers in range from 0 to $m-1$, therefore sequences of length 2^j could be compared in constant time. We denote by $ident[v_i, 2^j]$ the identifier corresponding to the sequence of labels $p(v_i, 2^j)$.

Since the basis for our data structure is not a single string, but a graph, we need an effective method of finding the further vertices on paths. We denote by $step[v_i, l]$ a vertex in which we end up after following l edges, starting from the vertex v_i. In the first phase we will calculate values $\mathit{step}[v_i, l]$ for l being powers of two, using a method called *binary lifting.*

We construct the dictionary of basic factors as follows. For $j = 0$ we simply assign $\mathit{ident}[v_i, 1] = b(i)$ and $\mathit{step}[v_i, 1] = f(v_i)$. For $j = 1, \ldots, N$ we assign identifiers for sequences of length 2^j based on identifiers for sequences two times shorter. We start by assigning a temporary identifier for every vertex v_i and sequence of labels $p(v_i, 2^j)$. The identifier is

$$\big(\mathit{ident}[v_i, 2^{j-1}],\ \mathit{ident}[\mathit{step}[v_i, 2^{j-1}], 2^{j-1}]\big),$$

which is a pair of integers in range from 0 to $m - 1$. Note that the second number in this pair is an identifier of a fragment of sequence $p(v_i, 2^j)$ which has length 2^{j-1} and starts exactly after 2^{j-1} elements of this sequence. In the second step we sort all these pairs lexicographically in time $O(m)$ (we can use radix sort here) and iterate over sorted list, assigning the final identifiers which are single integers — it is only important that to the same pairs we assign the same integer in range from 0 to $m - 1$. At the same time, the values $\mathit{step}[\cdot, 2^j]$ are assigned according to the following formula:

$$\mathit{step}[v_i, 2^j] = \mathit{step}[\mathit{step}[v_i, 2^{j-1}], 2^{j-1}].$$

The dictionary of basic factors allows us for efficient assignment of identifiers for sequences of labels of length n. The identifier for a sequence $p(v_i, n)$ is a pair of integers. The first number in this pair is the identifier of a sequence consisting of initial 2^N elements of $p(v_i, n)$. The second number is the identifier of a sequence consisting of trailing 2^N elements of $p(v_i, n)$. Since $2^N + 2^N > n$, such a pair uniquely identifies the sequence. Formally, the identifier for $p(v_i, n)$ is $(\mathit{ident}[v_i, 2^N], \mathit{ident}[\mathit{step}[v_i, n - 2^N], 2^N])$. In this case $n - 2^N$ does not have to be a power of two, therefore we calculate the value $\mathit{step}[v_i, n - 2^N]$ in time $O(\log n)$: we start from vertex $v = v_i$ and for each power of two 2^j in binary expansion of number $n - 2^N$ we perform $v := \mathit{step}[v, 2^j]$.

It remains to show how to use the dictionary of basic factors in finding the initial seeds of the bits generator. For simplicity of description, we add n vertices $u_1, \ldots, u_n$ to the graph, where vertex u_i has label b_i and an outgoing edge to vertex $u_{\min(i+1,n)}$. Calculating identifiers for all $m + n$ vertices in the graph can be performed in time complexity of $O((m + n) \log n)$. Next, it is enough to count vertices v_i for which the identifiers for sequences of labels $p(v_i, n)$ and $p(u_1, n)$ are equal.

The above algorithm has time complexity of $O((m + n) \log n)$. Unfortunately, the memory complexity of it is the same, which given the memory constraints in the problem, results in exceeding the memory. However, we can easily reduce the memory consumption to $O(m + n)$. For this purpose, during the construction of the dictionary of basic factors we will store in the memory only the last two rows of the table (corresponding to powers 2^{j-1} and 2^j) and calculate values $\mathit{step}[\cdot, n - 2^N]$ on the fly.

Automaton for finding pattern in trees

The problem could be solved in time complexity of $O(m+n)$. To this end we use some tools from the theory of text algorithms.

For a fixed string $x = x_1x_2\dots x_n$ (pattern) we could build a *pattern-matching automaton* for finding it in string $y = y_1y_2\dots y_m$ (text). The automaton reads subsequent letters from the text y and at every moment it is in one of $n+1$ states $s_0,\dots,s_n$. To be more specific: the automaton after reading sequence of letters $y_1\dots y_j$ is in state s_i, if, speaking informally, we managed to match exactly i letters of the pattern to the ending of the read sequence. In other words, the longest suffix of this sequence that is at the same time a prefix of pattern x has length i (i.e. $y_{j-i+1}\dots y_j = x_1\dots x_i$ and there is no larger i with such property). All the information needed by the automaton to work are stored in a transition table δ: the automaton which is in state s_i, after reading letter a from the input, goes to state $\delta(s_i,a)$. For instance for $0 \leqslant i < n$ we have $\delta(s_i, x_{i+1}) = s_{i+1}$.

Now, to find a pattern x in text y, it is enough to set the current state of the automaton to s_0, and then read subsequent letters of string y. Every time the automaton finds itself in state s_n, we know that we have found a new occurrence of word x (to be more precise: the last read letter is the last letter in this occurrence).

Such an automaton can be constructed in $O(nA)$ time, where A is the number of different letters which can appear in the pattern and the text. It should not pose any problem to anyone who knows the Knuth–Morris–Pratt algorithm. This pattern-matching algorithm maintains a state of the automaton during its runtime, however it does not explicitly construct the transition table of the automaton. The pattern matching in text of length m runs in time $O(m)$.

The pattern-matching automaton could be used to effectively finding a pattern in a tree. The problem is as follows: we have a rooted tree of m vertices, which are labeled using letters. We want to find every path that begins in any vertex and goes downward the tree, and whose subsequent labels spell the word x. The solution using the automaton is simple: we assign states of the automaton to the vertices. If the label of the root is a, then the root will have a state of $\delta(s_0,a)$. Next, a vertex labeled a, whose parent has assigned a state s_i, will be assigned a state $\delta(s_i,a)$. The number of occurrences of pattern x in the tree is the number of vertices with state s_n. The whole algorithm runs in time $O(nA+m)$ and the same memory complexity[⋆].

We are ready to return to our problem. Recall, that we need to find all occurrences of a string of length n in a labeled graph of m vertices, in which from every vertex there is exactly one outgoing edge. The possible letters are 0 and 1, then here $A = 2$.

Note that we can separately consider every weakly connected component of the graph, and at the end sum the obtained results. Since from every vertex there is exactly one outgoing edge, then a single connected component is a cycle with trees

[⋆]A tempting idea would be to use the Knuth–Morris–Pratt algorithm directly on the tree. After all, it also allows to find the states of the pattern-matching automaton and runs in linear time. However, there is a catch here: the total runtime of this algorithm is in fact linear, but calculating a *single* transition after matching k letters of the pattern can take $\Theta(k)$ time. For trees of specific shape such a slow step could take place at the same time for many vertices on the same level the tree. In such situation, although on every path downwards the tree the runtime is linear, the total runtime on the whole tree could be quadratic on the size of the tree. Thus the Knuth–Morris–Pratt algorithm alone is not sufficient and in fact we need to construct the pattern-matching automaton.

"attached" to it. Consider a single component and assume that its cycle consists of ℓ vertices, which have labels $y_1, y_2, \ldots, y_\ell$. Let us denote by T_i a tree attached to the i-th vertex on the cycle. We assume that the root of the tree is the vertex on the cycle, thus in every vertex of the cycle there is attached a tree (possibly consisting of this vertex only).

For a moment let us fix a tree T_i and say that we want to find all vertices in this tree that can be an initial seed of the bits generator. Since the edges of the tree are directed "upwards", in order to use the method described before, we will find in it a reversed pattern $x = b_n b_{n-1} \ldots b_1$. The method will work for all occurrences of pattern x which are completely contained in tree T_i. To also find occurrences that partially overlap with the cycle, we need to properly initialize the state of the automaton in the root of the tree.

For this purpose we find the longest suffix of sequence $\ldots y_\ell \ldots y_1 y_\ell \ldots y_1 y_\ell \ldots y_i$ (i.e. infinite sequence of labels on the cycle cut after the label y_i), that is at the same time a prefix of pattern x. Let us denote its length by j. Then the initial state of the automaton in the root of tree T_i is s_j.

It remains to show how to efficiently calculate initial states. Note that if the initial state for tree T_i is s_j, then for tree T_{i-1} the initial state is $\delta(s_j, y_{i-1})$ (except of special case $i = 1$, when instead of $i-1$ we have ℓ). Thus we only need to calculate the initial state for one tree.

We describe now how to do it effectively for tree T_1, i.e. how to find the length of the longest suffix of infinite sequence $\ldots y_\ell \ldots y_1 y_\ell \ldots y_1 y_\ell \ldots y_1$ that is at the same time a prefix of the pattern x. Similarly as before, we denote this value by j. The value j can be calculated, by running the pattern-matching automaton with pattern x on text $Y = (y_\ell \ldots y_1)^k$ (the cycle unrolled k times) for sufficiently large k. There is a small problem here: even if we know the smallest value of k for which we could correctly calculate j, this approach of simulating the automaton could be too time-consuming, since the calculations for a single cycle must be done in time proportional to its length.

We will tackle this problem as follows. First, we run the pattern-matching automaton for x on text $y_\ell \ldots y_1 y_\ell \ldots y_1$ resulted in unrolling the cycle twice and we examine the final state s_i of the automaton. If $i \leqslant \ell$, then we can assume that $j = i$.

For $i > \ell$ the matter is more difficult, since it means that the string x can go around the cycle multiple times. We know that a prefix of length ℓ of string x overlaps the cycle (possibly cyclically shifted). Let C be the greatest number for which a prefix of length $C\ell$ of string x overlaps with the cycle (i.e. x can be wound onto the cycle C times, but not $C+1$ times). That means that if we run the pattern-matching automaton for pattern x in text $Y = (y_\ell \ldots y_1)^{C+1}$, then the last state of the automaton would be s_j (the value we try to calculate). Suppose then than $C \geqslant 2$. We know that after reading initial 2ℓ letters of text Y the automaton will be in state s_i, thus from the periodicity of the prefix of pattern x, after reading $C\ell$ letters of text Y, the automaton will end up in state $s_{i+(C-2)\ell}$. It is enough to initiate the automaton with this state and finish the simulation, by reading final ℓ letters of text Y.

We need to effectively calculate the value C. Here we will find useful an array Pref (also know as *z-array*), in which Pref$[i]$ denotes the length of the longest prefix of string x that occurs in this word at position i. The array Pref can be calculated in linear time in respect to the length of the word. Having this array, the value C is obtained in constant time, since $C = \lfloor \mathrm{Pref}[\ell + 1]/\ell \rfloor + 1$.

To sum it up, the algorithm works as follows: first in time $O(n)$ we build an automaton for matching a pattern $x = b_n \dots b_1$ and we calculate the array Pref. Next we partition the graph into weakly connected components. In each component we find the pattern in time proportional to the sum of sizes of trees attached to the cycle and the size of the cycle, that is simply in time proportional to the size of the component. Thus the time complexity of the whole algorithm is $O(m + n)$.

H

Afternoon Tea

★

Task author: Tomasz Idziaszek
Solution description: Jakub Łącki
Available memory: 32 MB
`https://oi.edu.pl/en/archive/amppz/2011/her`

During his visit to the Bytic Islands, Byteasar really enjoyed the national beverage of Byteans: that is, tea with milk. This drink is always prepared in a strictly determined manner, which is as follows. Firstly the teacup is filled with tea mixed half and half with milk. Then an n-letter *ceremonial word* consisting of letters `H` and `M` is chosen. Now, for $i = 1, 2, \ldots, n$, the following action is performed: if the i-th letter of the ceremonial word is `H`, one should drink half of the teacup, add tea until the teacup is full, and stir. On the other hand, if the i-th letter of the word is `M`, one should perform a similar action, but milk should be added instead of tea. After such action is performed for each letter of the ceremonial word, the remaining liquid is disposed of.

Each time Byteasar performs the ceremony, he wonders which of the ingredients he has drunk more of: tea or milk. Help Byteasar to answer this question.

Input

The first line of the input holds an integer n ($1 \leqslant n \leqslant 100\,000$). The second line contains an n-letter word consisting of the letters `H` and `M`; this is the ceremonial word used by Byteasar.

Output

Your program should output a single letter `H` if Byteasar has drunk more tea than milk; a single letter `M` if he has drunk more milk than tea; or the word `HM` if he has drunk equal amounts of tea and milk.

Example

For the input data:

```
5
HMHHM
```

the correct result is:

```
H
```

Explanation of the example: Byteasar has drunk $1\frac{37}{64}$ teacups of tea and $\frac{59}{64}$ teacups of milk in total.

Solution

It gets a bit easier if we change our perspective a little. Instead of calculating the amount of tea and milk that were poured into the glass, let us consider the amount that Byteasar does not drink. In other words, we focus on calculating what remains in the cup after the ceremony. Since we know how many cups of each beverage were poured into the cup during the ceremony, this information is enough to answer the question. Moreover, in order to avoid working with fractions, we assume that the volume of the cup is 2^{n+1} milliliters and it is initially filled with 2^n milliliters of milk and the same amount of tea.

Let us focus on the case when, in the first turn, Byteasar pours in 2^n milliliters of tea. How much of this tea will remain in the cup until the end of the ceremony? In the second turn, Byteasar drinks half of this tea, so there are 2^{n-1} milliliters remaining. After the third turn, 2^{n-2} milliliters remain, and so on. After n turns (that is, $n-1$ dilutions) in the cup there will be 2 milliliters of tea that come from the first turn.

The liquid poured in the second turn is diluted $n-2$ times, so it determines 4 milliliters of the final contents of the cup. In general, from the i-th portion, 2^i milliliters last until the end. Apart from that, there is also 1 milliliter of tea and 1 milliliter of milk, which were both poured at the very beginning. For the sample ceremonial word `HMHHM`, at the end there are $1+2^1+2^3+2^4$ milliliters of tea and $1+2^2+2^5$ milliliters of milk in the cup.

This allows us to express the final contents of the cup as a sum of powers of 2. We then have to add all these numbers, which in some programming languages requires implementing big-integer arithmetic. Similarly, we can count the amount of tea and milk that Byteasar pours into the cup. At the end, it is sufficient to perform two subtractions to discover how much of each beverage Byteasar has drunk.

This allows us to reach the solution. However, it turns out that we do not have to perform complicated calculations. Recall that we only have to determine which beverage Byteasar has drunk more of, so we do not have to calculate the exact numbers.

A better solution

For the description, assume that Byteasar does not perform the last pouring, so the cup is half empty after the ceremony. This does not change anything, as Byteasar does not drink what he pours at the end. Therefore, let us remove the last letter from the ceremonial word, and assume from now on that it has length of $n-1$.

Let us denote by t_p and t_c the amount of tea poured into the cup an the amount that remained in the end. Similarly, let m_p and m_c denote equivalent values with regard to milk. We know that Byteasar has drunk $t_p - t_c$ milliliters of tea and $m_p - m_c$ milliliters of milk. At the end, the cup is half full, so $t_c + m_c = 2^n$.

We first focus on the case where there are more letters of one kind that the other in the ceremonial word. It turns out that in such a situation Byteasar drinks more of the beverage that was poured into the cup more often. Assume that he poured more tea, so t_p is larger than m_p by at least half of the cup (2^n milliliters), so $t_p \geqslant m_p + 2^n$. To compute the amount drunk by Byteasar, we have to subtract the final contents of the cup: that is, t_c and m_c. The cup, however, is half empty, and we know that it contains at least 1 milliliter of each beverage (coming from the first pouring). In particular, there are at most $2^n - 1$ milliliters of tea remaining in the cup ($t_c \leqslant 2^n - 1$). Hence, the amount of tea drunk by Byteasar is bounded as follows:

$$t_p - t_c \geqslant t_p - 2^n + 1 \geqslant m_p + 2^n - 2^n + 1 = m_p + 1.$$

This means that the amount of tea Byteasar has drunk is greater that the total amount of milk he has poured into the cup. Consequently, when the ceremonial word contains more `H`s that `M`s, Byteasar drinks more tea. Analogously, it the word contains more `M`s, Byteasar drinks more milk.

It remains to consider the case where the number of `H`s and `M`s are equal, so the cup is filled with the same amount of each of the beverages. In particular, this happens when $n = 1$. This is a special case, in which Byteasar drinks only half of the cup containing tea mixed half an half with milk. So he drinks an equal amount of each beverage.

Let us now assume that $n > 1$. If the amounts of milk and tea poured into the cup are equal, Byteasar drinks more of the beverage whose remaining amount in the cup is *smaller*. We know that half of the final cup contents comes from the next-to-last pouring (the last that we have not ignored). Assume that Byteasar poured milk then. In addition, in the cup there is 1 milliliter of milk poured at the very beginning. We infer that there has to be more milk than tea in the cup, so Byteasar has drunk more tea.

Therefore, the entire algorithm boils down to three simple cases. If $n = 1$, Byteasar drinks an equal amount of both beverages. If one letter is more common than the other in the ceremonial word (not considering its last letter), then Byteasar drinks more of the beverage represented by the more common letter. If the two letters appear in the same quantities, then we know that Byteasar drinks less of the beverage that he poured in the next-to-last turn.

I

Intelligence Quotient ★★★

Task author: Marek Cygan
Solution description: Eryk Kopczyński
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2011/ilo`

At the University of Byteland one can only study maths and computer science. Currently there are n maths students and m computer science students. These majors are so hard to study that nobody studies both of them at the same time.

Byteasar is the rector of the university. He would like to form a team of students which will solve all the hardest problems of mankind. Since he knows the IQ of each student, he has decided to form a team with the largest possible sum of IQs of its members.

However, IQ is not everything. That is why the rector would like all members of the team to know each other. It is known that all maths students know each other. And similarly, each computer science student knows every other student majoring in computer science.

Help the rector by writing a program that will help him form a team of students with the largest possible sum of IQs in which all the members know each other.

Input

The first line of the input contains three integers n, m and k ($1 \leqslant n, m \leqslant 400$, $0 \leqslant k \leqslant n \cdot m$) which specify the number of maths students, the number of computer science students, and the number of pairs of students from different majors that know each other, respectively.

Each of the following k lines describes one pair of acquaintances: the i-th of these lines contains two integers a_i and b_i ($1 \leqslant a_i \leqslant n$, $1 \leqslant b_i \leqslant m$) specifying a number of a maths student and a number of a computer science student from the i-th pair. The maths students are numbered with integers starting from 1 and so are the computer science students.

The following line contains n integers in the range $[1, 10^9]$, which represent the IQs of the subsequent maths students. The next line contains m integers representing the IQs of the computer science students, in a similar format.

Output

The first line of the output should contain one integer equal to the maximum sum of IQs in a team satisfying Byteasar's requirements.

The second line should contain one integer — the number of maths students that Byteasar should choose. The third line should contain the numbers of these students, listed in any order. If there are no maths students in the team, an empty line should be printed.

The following two lines should describe the numbers of computer science students assigned to the team, in a similar format.

If there are multiple solutions, your program should output any one of them.

Example

For the input data:

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

the correct result is:

```
6
1
2
2
1 2
```

Solution

Let's formulate our problem in the language of graph theory. We are given a bipartite graph $G = (V_1, V_2, E)$, where V_1 denotes math students, V_2 denotes computer science students, and E is a relation of acquaintance between students of different majors. Each vertex v has a nonnegative weight $w(v)$ (IQ of the student). A *bipartite clique* is a bipartite graph in which every two vertices from different sets are connected with an edge. The problem from the statement boils down to finding in graph G a *heaviest* bipartite clique, i.e. bipartite clique which has greatest possible sum of vertices' weights. Formally, if we denote by W the set of vertices of the bipartite clique, it should be $(W \cap V_1) \times (W \cap V_2) \subseteq E$ and the sum of weights of vertices in V must be as big as possible.

Simplified variant

We begin with a simpler problem: we assume that all weights are equal to 1. In this case maximising the sum of vertices' weights of bipartite clique boils down to maximising the number of its vertices. Thus we are looking for a *largest* bipartite clique.

Let $E' = V_1 \times V_2 \setminus E$, i.e E' is the relation of non-acquaintance. We show now that the problems of finding the following sets of vertices are equivalent (see also figure 1):

- Set of vertices of a largest bipartite clique W.
- Largest independent set W' in graph $G' = (V_1, V_2, E')$. *Independent set* is a set of vertices, between which there are no edges:

$$((W' \cap V_1) \times (W' \cap V_2)) \cap E' = \emptyset.$$

- Smallest vertex cover P in graph G'. *Vertex cover* is a subset $P \subseteq V_1 \cup V_2$, such that for every edge $(v_1, v_2) \in E'$ either $v_1 \in P$ or $v_2 \in P$.
- Largest matching in graph G'. *Matching* is a subset of graph's edges such that no edges share a common vertex.

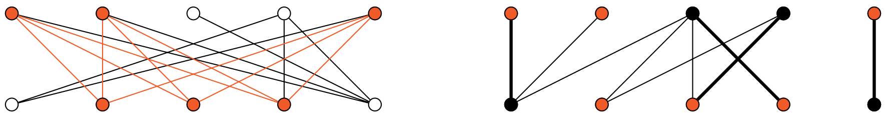


Figure 1. Left: sample bipartite graph G with a largest bipartite clique (colored vertices). Right: bipartite graph G' (complement of graph G) with a largest matching (thick edges), a minimal vertex cover (black vertices) and a largest independent set (colored vertices).

Equivalence of the first two problems follows directly from the definition of E'; in the second case we simply have $W = W'$. Equivalence of the second and the third comes from the fact that the set of vertices is independent if and only if its complement is a vertex cover. The last equivalence is König's Theorem.

Theorem 1 (König). In a bipartite graph the size of a largest matching is equal to the number of vertices in the smallest vertex cover.

The proof of König's Theorem is constructive and thanks to it we can find in linear time (on the size of the graph) a smallest vertex cover in G', given a largest matching in this graph. On the other hand, a maximal matching can be found in time complexity of $O(|V| \cdot |E'|)$ (where V denotes the set of all vertices in the graph) with a classic algorithm using alternating paths or in time $O(|E'|\sqrt{|V|})$ using Hopcroft–Karp algorithm. This means that this special variant of the problem can be solved in time $O(|E| + |E'|\sqrt{|V|})$.

The original problem

Now we get back to the general version of the problem, with different weights of vertices.

How can we generalize the previous reasoning? Observe, that we can do it easily, however not efficiently, as follows: we replace each vertex v with $w(v)$ copies of this vertex and we use the previous algorithm. It can be shown that in the optimal solution either we take all the copies of a given vertex or we take no copy at all. Thus in such way we get an optimal solution of the original problem. Experienced contestants will also easily restore the effectiveness in searching for the matching — it is known that the problem of finding a largest matching is a special case of the problem of finding a maximal flow. In a flow network we obtain here we can merge back copies of a given vertex and sum the capacities of merged edges, again obtaining a small graph.

It turns out that we can solve the whole problem differently, without having to reduce to the special case of unary weights. Some contestants applied this approach, thus we have decided to present it in this book. Algorithmically this solution is equivalent to the above (in particular, it hides a constructive proof of König's Theorem).

Algorithm

From the previous analysis we know that our problem is equivalent to finding a vertex cover of the smallest sum of vertices weights (we simply call it a *lightest* vertex cover). We reduce this problem to finding a maximal flow in a network.

We construct a flow network $H = (V_H, E_H)$ based on graph $G' = (V_1, V_2, E')$; see the left part of figure 2. Each edge in E' is given a capacity of ∞. Next we add source s and we connect it with every vertex $v_1 \in V_1$ with an edge of capacity $w(v_1)$. We also add sink t and connect every vertex $v_2 \in V_2$ with it with an edge of capacity $w(v_2)$. We show that the following values are equal:

- Weight of a lightest vertex cover in graph G'.
- Value of a minimal cut in flow network H. *Cut* in a flow network is a set of edges, that after removing it there is no path from s to t in the network; the *value* of a cut is the sum of capacities of removed edges.
- Value of a maximal flow in network H.

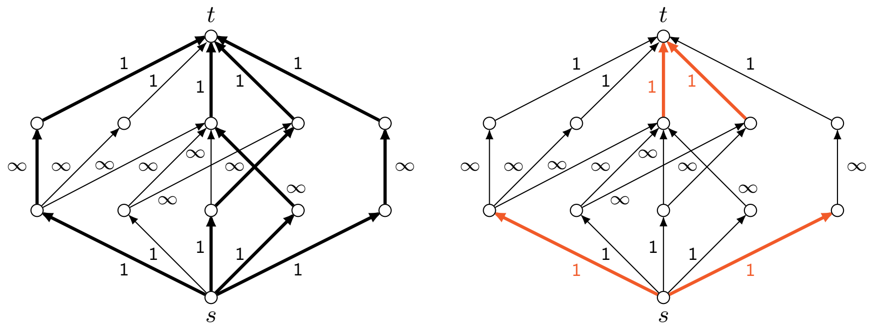


Figure 2. Left: flow network H with a maximal flow (thick edges) constructed for the bipartite graph G' from figure 1. Since vertex weights in this example are unary, the maximal flow corresponds to the maximal matching from this figure. Right: the minimal cut in network H (of value 4) corresponding to the minimal vertex cover from figure 1.

The equivalence of the last two follows from the fundamental theorem which characterizes flows in networks.

Theorem 2 (About maximal flow and minimum cut). In any flow network a minimal cut and a maximal flow have the same value.

We will show now equivalence of the first two values. It is enough to note that there is a one-to-one correspondence between cuts in H of weight less that ∞ and vertex covers in G'. Indeed, for any such cut we remove from network H some edges that connect the source with vertices from V_1 and some edges that connect vertices from V_2 with the sink, such that after removal there is no path coming from the source to the sink. Equivalently, for every edge $v_1v_2 \in E'$ in the cut must be present an edge sv_1 or edge v_2t. We interpret removal of edge sv_1 or v_2t as choosing vertex v_1 or v_2, respectively, to the vertex cover. In this way the condition of a cut means exactly that for every edge E' at least one of its endpoints must belong to the vertex cover, which essentially is the definition of such cover (see the right side of figure 2). Moreover, the weight of the cut is exactly the weight of the corresponding vertex cover. This shows the needed equality.

Implementation

To solve our problem, we need not only the weight of a lightest vertex cover of graph G', but also a vertex cover itself. We will obtain it immediately, if we know a minimal cut in the flow network H. As the last piece of the puzzle, we need the way to obtain a minimal cut from the structure of a maximal flow.

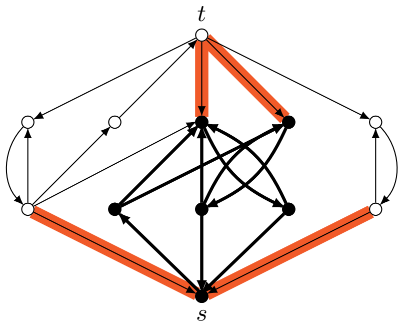


Figure 3. Residual network (without weights on edges) constructed for the maximal flow for the network from figure 2. Marked set X of vertices reachable from the source (black circles, thick edges) and set of edges whose counterparts in the original flow network consist the minimal cut (colored).

Fortunately, there is a simple algorithm for this, based on the residual network. Recall that the *residual network* is a weighted directed graph based on a flow network and some flow in this network. If in the flow network we have an edge uv with capacity c, through which flow f flows, then the residual network contains two edges: uv of weight $c - f$ and vu of weight f. We ignore edges of weight zero. Let X be the set of vertices that in the residual network constructed for a maximal flow are connected with the source with edges of positive weights. Then one of minimal cuts is the set of edges in the original network coming from X to $V_H \setminus X$; see figure 3.

Finally, it is worthwhile to think about the time complexity of our solution. The flow network has $|V_H| = |V| + 2$ vertices and $|E_H| = |E'| + 2 \cdot |V|$ edges. Edmonds–Karp algorithm of complexity $O(|V_H| \cdot |E_H|^2)$ will probably exceed the time limit. It is better to use Dinitz algorithm of complexity $O(|V_H|^2 \cdot |E_H|)$ (it is always useful to have this implementation ready for use on the contest) or quite simple in implementation scaling algorithm, whose outline we present below.

In the scaling algorithm we assume that all finite capacities are from the range $[0, 2^B)$. At the beginning we set all finite capacities to zeros. Next we perform B phases for i from $B - 1$ to 0. Before the i-th phase we increase the capacity of edge e by 2^i if and only if the i-th bit of the original capacity of this edge was equal to 1. In this approach in the i-th phase we will find at most $O(|V_H|)$ augmenting paths, each in time $O(|V_H|^2)$, therefore everything will work in time $O(B \cdot |V_H|^3)$.

J

Cave

★★★

Task author: Jakub Łącki
Solution description: Jakub Łącki
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2011/jas`

Byteasar has discovered a cave. It appears that the cave contains n chambers connected with passages in such a way that there exists a single way of getting from any chamber to any other chamber.

The cave should now be examined more thoroughly, so Byteasar has asked his friends for help. They have all arrived at the cave and they are willing to divide themselves into groups. Each group should examine the same number of chambers, and each chamber should be examined by exactly one group. Additionally, for the groups not to interfere with each others' work, the members of each group should be able to move between the assigned chambers without passing through chambers assigned to other groups.

How many groups can the explorers be divided into?

Input

The first line of the input contains one integer n ($2 \leqslant n \leqslant 3\,000\,000$) specifying the number of chambers in the cave. The chambers are numbered 1 through n.

The following $n - 1$ lines describe connections between the chambers. The i-th of these lines contains an integer a_i ($1 \leqslant a_i \leqslant i$) which represents a passage connecting chambers number $i + 1$ and a_i.

Output

Your program should output a single line containing all integers k, such that the chambers can be divided into k disjoint sets of equal size, and one can move between any two chambers belonging to the same set passing only through chambers from this set. The numbers should be written in an ascending order and separated with single spaces.

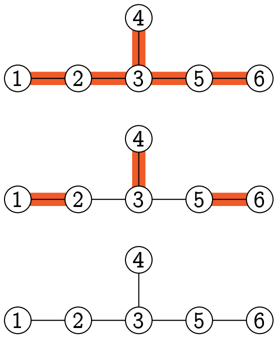
Example

For the input data:

```
6
1
2
3
3
5
```

the correct result is:

```
1 3 6
```



Solution

Let us use terminology of graph theory. The cave described in the problem statement is a connected graph without cycles, that is a tree. We begin with an obvious observation: in order to partition a tree into pieces of size k (i.e. containing k vertices each), then number k must be a divisor of n. Moreover, if we partition a tree into pieces of size k, then we will get exactly $\frac{n}{k}$ pieces. Finally, if we want to break a tree of n vertices into $\frac{n}{k}$ trees of size k, we must remove exactly $\frac{n}{k} - 1$ edges. These simple observations immediately guide us to a first solution. As it often is, this first algorithm will not be good enough, but we deal with it afterwards.

Assume that k is a divisor of n and we want to check, whether we can partition a tree into pieces of size k. On the first sight it could not be obvious, however, such a partition is uniquely determined (as long as it exists). A sketch of a proof of this fact could be as follows: consider any partition of the tree on pieces of size k, where $k < n$. In such a partition there is a piece P which is connected to the rest of the tree with exactly one edge e. In every partition of the tree for this value of k the edge e must connect two pieces, since otherwise we could get instead of P a piece of size less or greater than k. Now we can cut piece P from the rest of the tree and similarly prove that the rest of the tree can also be partitioned in exactly one way.

In our first solution we will use recursive approach. We run DFS on the input tree, and every time we return from recursive calls, we calculate the sizes of subtrees rooted in vertices. When we find a vertex in which a subtree of size k is rooted, we remove the edge that connects this subtree with the rest of the tree. Removed vertices no longer are taken into account when calculating sizes of subsequent subtrees. It should be clear, that if the whole tree can be partitioned into pieces of size k, then exactly $\frac{n}{k} - 1$ times we will claim than we need to remove an edge, and at the end only the piece of size k containing the root of the tree will remain. The converse implication is also true. If $\frac{n}{k} - 1$ times we cut a tree of size k, then we have found a partition of whole tree into $\frac{n}{k}$ pieces of size k.

This recursive procedure can be implemented in such a way that for a fixed k it will work in time $O(n)$. Therefore, the whole algorithm has the time complexity of $O(n \cdot d(n))$, where $d(n)$ is the number of divisors of n. Note that at the beginning of the algorithm we could iterate over all integers from 1 to n and test which of them divides n, since it only takes $O(n)$ time.

The question remains how big could $d(n)$ be. Among integers from 1 to $3 \cdot 10^6$ the largest number of divisors, namely 336, has number 2 882 880. Thus in the pessimistic case (which obviously was included in the tests to the problems) this solution will be to slow.

Faster solution

To speed up our solution, we consider all possible values of k during one DFS pass. Let us fix an edge e and answer the following question: how to test that edge e is removed during partitioning the tree into pieces of size k? For sure e must partition the tree into pieces of sizes divisible by k. In general the following fact holds:

Fact 1. A tree can be partitioned into pieces of size k if and only if there exist exactly $\frac{n}{k} - 1$ edges which connects subtrees of sizes being multiples of k.

Why is that? If we demand for an existence of a partition, then we remove exactly $\frac{n}{k} - 1$ edges, and at both sides of each removed edge there is some number of pieces of size k. Therefore, the number of vertices on both sides of each removed edge is divisible by k. For the opposite direction: suppose that there exist $\frac{n}{k} - 1$ edges satisfying the above condition. It is easy to see that if we remove from the tree these edges, then every subtree we obtain along will have size divisible by k. At the end, we get $\frac{n}{k}$ non-empty pieces, which sizes are divisible by k. Their total size is n, thus every piece must contain k vertices.

Fact 1 allows us to think about solution in simpler terms. It is sufficient to count how many edges satisfies certain properties. To be more precise, we iterate through all edges of the tree and for each k we calculate value $edg[k]$, that is the number of edges that connects subtrees of sizes being multiples of k.

Let us consider an edge that connects subtrees of sizes a and $n - a$. For which values of k should we increment the value $edg[k]$? It is easy to see that we should do this for all integers k that divides both a and $n - a$. In other words, we increment $edg[k]$ for all integers k that are divisors of $\mathrm{GCD}(a, n - a)$. This algorithm will be easier to realise "lazily". Instead immediately updating proper cells in array *edg*, we note on the side that we want to increment values in cells which indices are divisors of $\mathrm{GCD}(a, n - a)$, and final values in array *edg* will be calculated later.

For this purpose we will use yet another array which we denote by t. To note that we want to increment by 1 values $edg[k]$ for all integers k that are divisors of i, we will add 1 to cell $t[i]$. Thus when considering an edge connecting subtrees of sizes a and $n - a$ we will be incrementing by 1 the value $t[\mathrm{GCD}(a, n - a)]$. After going over all edges, we can calculate values $edg[k]$ by realizing changes from array t. Such a lazy approach will let us better estimate time complexity, since for every index i in array t we will go through all divisors of i only once. We also can calculate the final contents of array *edg* a little bit simpler. It is enough to note that we can obtain value $edg[k]$ by summing $t[i]$ over all i that are multiples of k.

In what time we could calculate array *edg* based on array t? By calculating $edg[k]$ we consider all multiples of k, which needs $\lfloor \frac{n}{k} \rfloor$ steps. Remember, that we need values in array *edg* only for divisors of n. The time complexity can be then expressed as $\sum_{k|n} \frac{n}{k}$. Observe, that we sum here all divisors of n, since if k iterates over divisors from *the smallest*, then $\frac{n}{k}$ also iterates over divisors of n, but from *the greatest*. Therefore, this step requires time $O(D(n))$, where $D(n)$ is the sum of divisors of n. Fortunately, $D(n)$ is a function which grows very slowly, since $D(n) = O(n \log \log n)$.

The most costly operation of our algorithm is therefore calculating the greatest common divisor of integers from range 1 to n. Using Euclid's algorithm, one such calculation can be done in time $O(\log n)$. Since we perform it n times, the the total running time of the algorithm is $O(n \log n)$.

Even faster solution

This is enough to solve the problem. However, we can show a little bit more efficient method. It turns out that all values $\mathrm{GCD}(i, n-i)$ for $i = 1, \ldots, n$ can be calculated beforehand in time complexity of $O(n \log \log n)$. Note that if integer d is divides both i and $n - i$, then it also divides n. To calculate $\mathrm{GCD}(i, n - i)$ it is sufficient to find the greatest divisor of n which divides i and $n - i$.

We will fill up an array *gcd*, in such a way that $gcd[i]$ at the end will be equal to GCD$(i, n-i)$. We iterate over all divisors of n in increasing order and for every divisor d we consider all its multiples $j \cdot d$. For each such multiple we update value $gcd[j \cdot d]$ for d. Thus, the algorithm goes over all multiples of all divisors of n, similarly like the algorithm calculating array *edg* based on array t. Using the former analysis, we conclude that the array *gcd* can be calculated in time $O(n \log \log n)$. Thanks to that we managed to reduce the time complexity of the whole solution to $O(n \log \log n)$. However, we should state that in practice such improvement will be rather unnoticeable.

How many divisors can integer have?

At the end let us go back to function $d(n)$, which counts divisors of n. We showed a limit for values of this function for $n \leqslant 3 \cdot 10^6$, but we did not say anything about asymptotical limit. It is easy to prove that $d(n) < 2\sqrt{n}$, since every number can have at most $\sqrt{n}$ divisors not greater that $\sqrt{n}$, and every divisor greater than $\sqrt{n}$ is paired up with a divisor smaller than $\sqrt{n}$. But there is even stronger estimation, namely $d(n) = n^{O(1/\log \log n)}$.

At the first sight it does not really tell us how much it is. Note that $\log \log n$ function grows very slowly. If we assume that $\log n$ is a binary logarithm, then $\log \log(2^{32}) = 5$, and $\log \log(2^{64}) = 6$. But for the needs of calculating running time of algorithmic problems, in which we usually deal with integers not bigger than 2^{64} it is convenient to assume that in pessimistic case value $d(n)$ is roughly $\sqrt[3]{n}$. From mathematical point of view this is a heresy, but such approximation works quite well in practice.

K

Cross Spider

★★

Task author: Szymon Acedański
Solution description: Tomasz Idziaszek
Available memory: 32 MB
`https://oi.edu.pl/en/archive/amppz/2011/krz`

The Bytean cross spider (*Araneida baitoida*) is known to have an amazing ability. Namely, it can instantly build an arbitrarily large spiderweb as long as all of its threads are contained in a single plane. This ability gives the spider an opportunity to use a fancy hunting strategy. It does not need to wait until a fly is caught in an already built spiderweb; if only the spider knows the current position of a fly, it can instantly build a spiderweb to catch the fly.

A cross spider has just spotted n flies in Byteasar's garden. Each fly is flying still in some point of a three-dimensional space. The spider is wondering if it can catch all the flies with a single spiderweb. Write a program that answers the spider's question.

Input

The first line of the input contains an integer n ($1 \leqslant n \leqslant 100\,000$). The following n lines contain a description of the flies in space: the i-th line contains three integers x_i, y_i, z_i ($-1\,000\,000 \leqslant x_i, y_i, z_i \leqslant 1\,000\,000$) giving the coordinates of the i-th fly (a point in the three-dimensional Euclidean space). No two flies are located in the same point.

Output

Your program should output a single word `TAK` (i.e., *yes* in Polish) if the spider can catch all the flies with a single spiderweb. Otherwise your program should output the word `NIE` (*no* in Polish).

Example

For the input data:

```
4
0 0 0
-1 0 -100
100 0 231
5 0 15
```

the correct result is:

```
TAK
```

whereas for the input data:

```
4
0 1 0
-1 0 -100
100 0 231
5 0 15
```

the correct result is:

```
NIE
```

Solution

The aim of the problem *Cross Spider* was to check, whether the contestants know the basic tools of computational geometry. For the given n points in the three-dimensional space, one has to check whether all of them lie on a single plane. To fix the notation, we denote the i-th point by $p_i = (x_i, y_i, z_i)$.

Any three non-collinear points (i.e. such points that do not lie on a single line) uniquely determine a plane in the space. The solution is divided into two parts: first we find three non-collinear points (if they exist), and later we check whether the remaining points lie on the plane given by these three points. Note that for $n \leqslant 3$ the answer is obviously positive, thus from now on we assume that we have at least four points.

We cannot simply take any three points (e.g. p_1, p_2 and p_3), since it could turn out that they lie on one line. But if it is not the case that all n points lie on a single line, then we can take two of them (e.g. p_1 and p_2) and search among remaining points some point which is non-collinear with them. To check whether point p_i (for $i = 3, \ldots, n$) lies on a line designated by points p_1 and p_2, we calculate the cross product $t_i := (p_2 - p_1) \times (p_i - p_1)$. Recall, that the cross product of vectors $v = (x_v, y_v, z_v)$ and $w = (x_w, y_w, z_w)$ is a vector defined as follows:

$$v \times w := (y_v \cdot z_w - y_w \cdot z_v,\ z_v \cdot x_w - z_w \cdot x_v,\ x_v \cdot y_w - x_w \cdot y_v).$$

Point p_i lies on a line containing points p_1 and p_2 if and only if all coordinates of vector t_i are equal 0.

If all vectors t_i corresponding to points $p_3, \ldots, p_n$ are zeros, then all points lie on a single line (thus on a common plane) and the answer is positive. On the other hand, if for some i we got a non-zero vector t_i, then the triplet of points p_1, p_2 and p_i uniquely determines a plane. In such a situation the cross product t_i is a vector perpendicular to the plane containing points p_1, p_2 and p_i. To check whether some other point p_j lies in this plane, we only have to check whether vector $p_j - p_1$ is perpendicular to t_i. This can be done by calculating a dot product $t_i \cdot (p_j - p_1)$. Recall that the dot product of two vectors $v = (x_v, y_v, z_v)$ and $w = (x_w, y_w, z_w)$ is a number defined as

$$v \cdot w := x_v \cdot x_w + y_v \cdot y_w + z_v \cdot z_w.$$

Two non-zero vectors are perpendicular if and only if their dot product is equal to 0. Thus point p_j lies on a plane determined by points p_1, p_2 and p_i if the dot product of t_i and $p_j - p_1$ is equal to 0.

The above solution can be implemented in such a way that each point is checked only once. The time complexity of the whole solution is $O(n)$.

Watch out for constraints

One of decisions that should be made during the implementation of the above calculations, is choice of suitable data type. To avoid subtle troubles with operations on floating-point numbers, it is good to use integer types (fortunately it is possible here, since the input data are integers and we only perform on them operations of addition, subtraction and multiplication).

If we write down the dot product from our solution, it will turn out that it is a sum of six addends. Each addend is a product of three values, which are differences of point coordinates. The absolute value of each coordinate is limited by 10^6, thus the absolute value of each addend does not exceed $(2 \cdot 10^6)^3 = 8 \cdot 10^{18}$. This is just enough, so that each addend fit in a 64-bit signed variable.

Unfortunately, estimation $48 \cdot 10^{18}$ on the sum of these addends exceeds the range of 64-bit integer. We can cope with this in several ways. The easiest way is to use 128-bit integers, as long as our compiler supports them (e.g. the GCC compiler used during the contest provides `__int128` extension). One could also implement such integers, especially that it only needs to implement operations of addition and comparing with zero. The third way is to calculate this sum twice: once modulo m_1 and then modulo m_2, where m_1 and m_2 are relatively prime numbers whose product exceeds $48 \cdot 10^{18}$. From the Chinese remainder theorem we know that both results will be zero if and only if the sum of addends is equal to 0.

2012

17th Polish Collegiate Programming Contest
Warsaw, October 26–28, 2012

A

Vending Machine

★★★

Task author: Jakub Pachocki
Solution description: Eryk Kopczyński
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2012/aut`

Byteasar studies computer science at the University of Bytetown. There is a snack vending machine at his faculty that sells n types of snacks, numbered 1 through n. Snacks of different types may have different price, since they differ in size and flavor.

Recently Byteasar discovered that the vending machine is broken. If one buys a snack of type i, the vending machine additionally dispenses one snack of each of the types 1, 2, ..., $i-1$, provided that snacks of these types are available (if there are no snacks of some of the types $1, 2, \ldots, i-1$, simply no snack of this type is dispensed). Buying snack of type i is possible only if at least one snack of this type is available.

Byteasar decided to take advantage of the fault he discovered. He would like to find out what is the maximum total value (that is, the sum of prices) of snacks that he can obtain in the vending machine using a given amount of money. He does not have to use all the money.

Input

The first line of the input contains two integers n and k ($1 \leqslant n \leqslant 40$, $1 \leqslant k \leqslant 64\,000$) specifying the number of types of snacks and the amount of money that Byteasar has at his disposal. The second line holds n integers $c_1, \ldots, c_n$ ($1 \leqslant c_i \leqslant 40$), the prices of snacks of respective types. The third line holds n integers $l_1, \ldots, l_n$ ($0 \leqslant l_i \leqslant 40$), the quantities of snacks of respective types that are available in the vending machine.

Output

The only line of the output should contain one integer: the total price of snacks that Byteasar can obtain in the vending machine using at most k units of money.

Example

For the input data:

```
6 8
7 2 3 5 7 2
1 3 0 3 2 1
```

the correct result is:

```
30
```

Explanation of the example: We buy a snack of type 6; the vending machine also dispenses one snack of each of the types 1, 2, 4 and 5. We buy a snack of type 4; in addition to this snack, the vending machine dispenses one snack of type 2.

Solution

Translation in progress...

B

Bus Trip

★★★

Task author: Jakub Radoszewski
Solution description: Jakub Łącki
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2012/biu`

Byteasar works as a teacher in a primary school in Byteburg. The weather is currently really nice, so Byteasar would like to take his class for a bus trip to Bytetown — the capital of Byteland. Byteasar has decided to hire a travel agency to plan the trip.

The streets in Bytetown form a regular grid of east–west and north–south streets. The distance between any pair of adjacent parallel streets is equal to 1 kilometer. There are tourist attractions located at several junctions. Bytean guides have assigned *attractiveness* coefficient to each tourist attraction: the greater the attractiveness, the more interesting is the attraction for the visitors. Byteasar knows that the pupils in the class he teaches get bored easily, thus he requires that the attractions visited on the trip have increasing attractiveness.

The travel agency has agreed to Byteasar's requirements, but at the same time the agency would like to earn as much money as possible for organizing the trip. The agency gets a fixed rate of 1 bythaler for each kilometer of the trip. The bus always chooses the shortest route along the streets of Bytetown when driving between the attractions on the trip. Moreover the agency gets additional money from the managers of attractions that are visited along the trip.

Help the agency plan a trip that satisfies Byteasar's requirements and grants the agency the highest profit. Please note that driving next to a tourist attraction (without stopping) does not count as visiting the attraction.

Input

The first line of the input contains two integers n and m ($2 \leqslant n, m \leqslant 1000$), the number of east–west streets and the number of north–south streets.

Following n lines describe the tourist attractions in Bytetown. The i-th of these lines holds m integers $w_{i,j}$ ($0 \leqslant w_{i,j} \leqslant 10^6$) specifying the attractiveness of each of the attractions located at the junctions of the i-th east–west street with the respective north–south streets. Attractiveness 0 means that there is no tourist attraction at the respective junction. You can assume that there is at least one tourist attraction in Bytetown.

Each of the following n lines holds n integers $c_{i,j}$ ($0 \leqslant c_{i,j} \leqslant 10^9$). The number $c_{i,j}$, that is, the j-th number in the i-th of the considered lines, represents the amount of money (in bythalers) that the agency receives for making a trip lead through the tourist attraction described by the attractiveness $w_{i,j}$. If there is no tourist attraction at a junction, the corresponding number $c_{i,j}$ equals 0.

Output

The only line of the output should contain one integer: the maximum profit (in bythalers) that the agency can make for organizing a trip leading through a number of attractions with strictly increasing attractiveness.

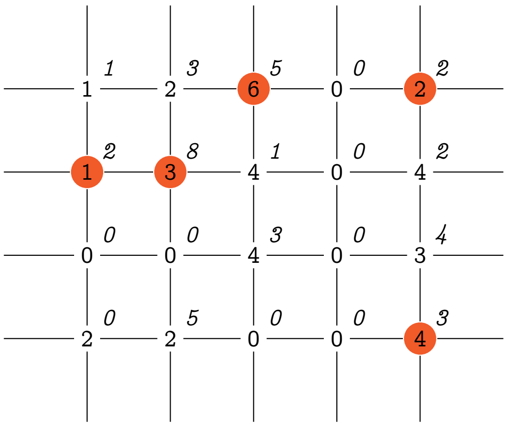
Example

For the input data:

```
4 5
1 2 6 0 2
1 3 4 0 4
0 0 4 0 3
2 2 0 0 4
1 3 5 0 2
2 8 1 0 2
0 0 3 0 4
0 5 0 0 3
```

the correct result is:

```
39
```



Explanation of the example: The numbers in the junctions represent the attractiveness of the respective attractions. The numbers written in italic represent the profit of the travel agency for making a trip lead through the respective attractions. The most profitable trip for the agency is highlighted with circles: the agency receives 2, 2, 8, 3 and 5 bythalers for these attractions respectively, and moreover it receives 19 bythalers as the bus fare.

Solution

Translation in progress. . .

C

Sequence

★★

Task author: Jakub Radoszewski
Solution description: Jakub Radoszewski
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2012/cia`

We say that an integer sequence $a_1, a_2, \ldots, a_n$ is k-*even* if the sum of any k consecutive terms of the sequence is even.

For a given sequence we would like to find out how many of its terms need to be changed so that the sequence becomes k-even.

Input

The first line of the input contains two integers n and k $(1 \leqslant k \leqslant n \leqslant 1\,000\,000)$ specifying the length of the sequence and the "evenness" parameter. The second line contains a sequence composed of n integers $a_1, a_2, \ldots, a_n$; each of them satisfies $0 \leqslant a_i \leqslant 1\,000\,000\,000$.

Output

The only line of the output should hold one integer: the minimum number of terms of the sequence that need to be changed so that it becomes k-even.

Example

For the input data:

```
8 3
1 2 3 4 5 6 7 8
```

the correct result is:

```
3
```

whereas for the input data:

```
8 3
2 4 2 4 2 4 2 4
```

the correct result is:

```
0
```

Solution

Translation in progress...

D

DNA

★★

Task author: Jakub Łącki
Solution description: Jakub Łącki
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2012/dna`

The mad scientist Byteasar would like to give birth to a new kind of creatures. For this, he has decided to modify the genome of a Bytean mouse.

A DNA can be described as a sequence of letters `A`, `C`, `G` and `T`. Byteasar's master plan is quite simple: using the DNA of a mouse, he is going to create a new DNA of the same length that is *as little* similar to the mouse's DNA as possible. The similarity of two DNAs is the length of their longest common subsequence. The longest common subsequence of two words x, y is defined as the longest word that can be obtained from each of x, y by removing some (possibly none) letters from both words. (Note that two words may have several longest common subsequences. For example, the longest common subsequences of the words `CACCA` and `CAAC` are `CAA` and `CAC`.) Write a program that computes the requested DNA.

Input

The first line of the input contains one integer n ($1 \leqslant n \leqslant 10\,000$) specifying the length of the DNA of a Bytean mouse. The second line contains mouse's DNA as a sequence of n uppercase letters from the set $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$.

Output

The first line of the output should contain one integer: the similarity of the Bytean mouse's DNA and the DNA computed by your program. The second line should hold a sequence of n letters `A`, `C`, `G`, `T`. This should be a DNA that is as little similar to the DNA from the input as possible. Should there be many correct answers, your program may output any one of them.

Example

For the input data:

```
4
GACT
```

one of the correct results is:

```
1
TCAG
```

Solution

Translation in progress...

E

Evaluation of an Expression ★★★★

Task author: Jakub Radoszewski
Solution description: Jakub Radoszewski
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2012/ewa`

Consider an expression E, containing integer constants from 0 to 9, variables from a to z, and the following operations: addition, multiplication, and exponentiation with a constant exponent. Quite surprisingly, each of the variables $a, b, \ldots, z$ appears in the expression E *at most once*. For a given prime number p, we would like to know how many roots modulo p the polynomial represented by this expression has. In other words, we want to count the number of ways in which integers from 0 to $p-1$ can be assigned to the variables in E, so that the value of E is divisible by p. Since the number of such roots can turn out large, it suffices to output it modulo 30 011.

For example, the polynomial represented by the expression

$$E = ((a+y)\cdot(z+8))^2$$

has 15 roots modulo $p = 3$, among which the roots:

$$(a=0,\ y=0,\ z=0), \qquad (a=1,\ y=2,\ z=0), \qquad (a=2,\ y=0,\ z=1)$$

can be found.

More formally, an *expression* is defined as follows:

- Each integer constant `0`, `1`, ..., `9` is an expression.
- Each variable `a`, `b`, ..., `z` is an expression.
- If `A` and `B` are any expressions, then each of `(A+B)` and `(A*B)` is also an expression: the first is the sum of expressions `A` and `B`, and the second is their product.
- If `A` is any expression, and `B` is an integer constant from `2`, `3`, ..., `9`, then `(A^B)` is also an expression: the expression `A` raised to the power of `B`.

Input

The first line of the input contains one prime number p ($2 \leqslant p < 15\,000$). The second line contains an expression E as specified above, described by a sequence of at most 300 characters `0`, `1`, ..., `9`, `a`, `b`, ..., `z`, `+`, `*`, `^`, `(`, `)`, without any white space.

Output

Let k denote the number of roots modulo p of the polynomial E. Your program should output one non-negative integer, remainder of dividing k by 30 011.

Example

For the input data:

```
3
(((a+y)*(z+8))^2)
```

the correct result is:

```
15
```

Solution

Translation in progress...

F

Formula One

★★★★

Task author: Tomasz Idziaszek
Solution description: Tomasz Idziaszek
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2012/for`

Little Bytie really enjoys watching Formula One races which are held annually on a track between Bytetown and Byteburg. The most exciting moments for him are overtakes. He would like to see as many of them as possible.

Bytie is dreaming of a race in which n Formula One cars compete and the car that started the race at the i-th position (for each $1 \leqslant i \leqslant n$) performs a_i overtakes during the race. We assume for simplicity that at each moment of time at most one overtaking takes place, in which exactly two cars participate (that is, one car goes past another car).

Bytie is wondering whether such a race is possible at all. Could you help him figure this out?

Input

The first line of the input contains one integer t that represents the number of test cases that follow.

Each test case is described in two lines. The first line contains one integer n ($1 \leqslant n \leqslant 1\,000\,000$) specifying the number of cars that participate in the race. The second line holds a sequence of n integers $a_1, a_2, \ldots, a_n$ ($0 \leqslant a_i \leqslant 10^9$) that gives the number of overtakes performed by the respective cars.

The size of a single input file does not exceed 20 MB.

Output

Your program should output t lines containing answers to the respective test cases. Each line should hold a single word `TAK` (i.e., *yes* in Polish) or `NIE` (*no* in Polish) depending on whether the race described by Bytie is possible or not.

Example

For the input data:

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

the correct result is:

```
TAK
NIE
TAK
```

Solution

Translation in progress...

G

Save the Dinosaurs ★★★★

Task author: Tomasz Idziaszek
Solution description: Tomasz Idziaszek
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2012/gen`

Breaking news from Bytetown: the archaeologists have discovered fossil remains of dinosaurs near the city! After hearing the news, several citizens of Bytetown wanted to pick up a bone or two for themselves. To save the priceless remains of dinosaurs, the mayor of Bytetown has decided to protect the excavation area and hired army for this purpose.

General Byteasar has located n soldiers in several *strategic positions* of the excavation area. (The soldiers can not stand just anywhere and hinder archaeologists' work. In addition, they must have good visibility to protect the excavation site.) We say that a point of the area is *protected* if moving from that point in any direction one unavoidably reduces the distance to at least one of the soldiers.

Byteasar has just been assigned a new rookie soldier. The general has decided to place the soldier in one of the m remaining vacant strategic positions. For each of the possible placements he would like to know what is the total area of the protected part of the excavations.

Input

The first line of the input contains two integers n and m ($3 \leqslant n \leqslant 100\,000$, $1 \leqslant m \leqslant 100\,000$) specifying the number of soldiers that are already stationed in the excavation area and the number of vacant strategic positions. The following n lines provide a description of the soldier's positions: the i-th of those lines contains two integers x_i, y_i ($-10^8 \leqslant x_i, y_i \leqslant 10^8$) that represent the coordinates of the position occupied by the i-th soldier (in a Cartesian coordinate system). The following m lines provide a description (in the same format) of the vacant strategic positions. All the points listed in the input are distinct.

You may assume that the area of the part of the excavations protected by the n soldiers is positive.

Output

Your program should output exactly m lines. The i-th line should contain the total area of the protected part of the excavations after the rookie soldier is located in the i-th previously vacant strategic position. All numbers should be written with a single digit after the decimal dot.

Example

For the input data:

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

the correct result is:

```
5.0
2.5
```

Solution

Translation in progress...

H

Hydra

★★

Task author: Tomasz Idziaszek
Solution description: Tomasz Idziaszek
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2012/hyd`

Little Bytie got a gift for his birthday. The gift contained a computer game called *The Amazing Adventures of Knight Byteasar*. The purpose of this game is to lead the knight through numerous challenges to defeat villains and evil witches and rescue damsels in distress. Bytie managed to complete almost all levels of the game. Now he is stuck at the last level in which Byteasar needs to fight a giant serpent, the Bytean Hydra.

Byteasar will use his sword to fight the monster. Two sword strokes are available in the game: Byteasar can either cut off the serpent's head or slaughter the head (the latter stroke, obviously, requires more effort). Cutting the head off is simpler, however, it results in new heads growing back from the serpent's neck. Hydra is defeated only when it has no more heads and no new heads can grow back from its neck.

The Bytean Hydra may have n types of heads that we number from 1 to n. In the beginning the serpent has one head of type 1. A head of type i (for $1 \leqslant i \leqslant n$) has the following characteristics: the number of sword swipes necessary to cut a head of this type off, u_i, the number of sword swipes necessary to slaughter a head of this type, z_i, and a list of r_i types of heads that grow back in place of a head of this type if it is cut off, $g_{i,1}, \ldots, g_{i,r_i}$.

Help Bytie compute the minimum number of sword swipes that are necessary to defeat the Hydra.

Input

The first line of the input contains one integer n ($1 \leqslant n \leqslant 200\,000$), specifying the number of types of heads of the Hydra. The following n lines hold a description of the respective types of heads; the i-th of those lines describes heads of type i. It starts with three integers u_i, z_i, r_i ($1 \leqslant u_i < z_i \leqslant 10^9$, $1 \leqslant r_i$) followed by a list of integers $g_{i,1}, \ldots, g_{i,r_i}$ ($1 \leqslant g_{i,j} \leqslant n$). The sum of all integers r_i does not exceed $1\,000\,000$.

Output

The only line of the output should contain one integer: the minimum number of sword swipes that are necessary to complete the game.

Example

For the input data:

```
4
4 27 3 2 3 2
3 5 1 2
1 13 2 4 2
5 6 1 2
```

the correct result is:

```
26
```

Solution

Translation in progress...

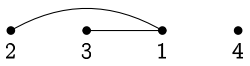
I

Inversions ★

Task author: Krzysztof Diks
Solution description: Jakub Łącki
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2012/inw`

Byteasar discovered a new family of undirected graphs that can be represented using inversions. Let $V = \{1, 2, \ldots, n\}$ be the set of vertices, and $a_1, a_2, \ldots, a_n$ some sequence of pairwise distinct numbers from set V. The vertices a_i and a_j are connected by an edge if the pair (i, j) forms an *inversion* in this sequence, that is, $i < j$ and $a_i > a_j$.

For example, let $n = 4$ and consider the sequence $2, 3, 1, 4$. We obtain the following graph:



Byteasar would like to check if the representation that he invented is useful indeed. He has decided to write a program that finds all the *connected components* of the graph. Recall that two vertices $u, v \in V$ belong to the same connected component if there exists a sequence of vertices starting with u and ending with v such that every two subsequent vertices in the sequence are connected by an edge. In our example we have two connected components: $\{1, 2, 3\}$ and $\{4\}$.

Help Byteasar!

Input

The first line of the input contains one integer n ($1 \leqslant n \leqslant 1\,000\,000$) specifying the number of vertices of the graph. The second line contains n integers $a_1, a_2, \ldots, a_n$.

Output

The first line of the output should contain the number of connected components in the graph; denote this number by m. Each of the following m lines should hold a description of one connected component. First a number k should be written, the size of the component, and then an *increasing* sequence of k vertex numbers of the component. The components should be listed in such order that the numbers of the first vertices of the components form an increasing sequence. In other words, if S and S' are two connected components, $u \in S$, $v \in S'$ are their vertices with the smallest number and $u < v$, then the component S should be listed earlier than S'.

Example

For the input data:

```
4
2 3 1 4
```

the correct result is:

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

Solution

Translation in progress...

J

Do It Tomorrow ★

Task author: Tomasz Idziaszek
Solution description: Jakub Radoszewski
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2012/jut`

procrastination (Latin *procrastinatio*, from *pro cras* — for tomorrow)
— a perpetual habit of putting off important tasks to a later time

Byteasar tends to postpone all the tasks he needs to perform. You could say that procrastination is his middle name. However, if he promises to do something, you can certainly count on him.

Byteasar woke up early today and prepared a list of n tasks that he needs to perform in near future. The i-th task will take him d_i consecutive days to perform and has to be completed within the next t_i days, starting from today. Byteasar would like to know how much time he can spend doing nothing until he really has to start performing some tasks. Could you write a program that will help him find that out? Byteasar could also write such a program himself, however this would be against his nature.

Input

The first line of the input contains one integer n ($1 \leqslant n \leqslant 1\,000\,000$), specifying the number of tasks that Byteasar has to perform. The following n lines hold a description of the tasks: the i-th of those lines contains two integers d_i and t_i ($1 \leqslant d_i, t_i \leqslant 10^9$). We assume that Byteasar is able to perform all the tasks on time.

Output

Your program should output one integer k: the maximum number of days during which Byteasar can avoid working. In other words, on the day number $k + 1$ at latest Byteasar must start performing one of the tasks in order to be able to eventually complete all the tasks on time.

Example

For the input data:

```
3
2 8
1 13
3 10
```

the correct result is:

```
5
```

Explanation of the example: For the first five days Byteasar rests. On the following five days he performs the first and the third task (in that order). Afterwards he uses one of next three days to perform the second task.

Solution

Translation in progress…

K

Rabbits

★★★

Task author: Tomasz Idziaszek
Solution description: Jakub Radoszewski
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2012/kro`

Byteasar has decided to go green and grow lettuce in his garden. As you can imagine, Bytean rabbits simply love lettuce, thus it's not surprising at all that they instantly arrived in Byteasar's garden.

In the garden there are n beds of lettuce numbered 1 through n. Every two subsequent beds are adjacent, that is, for each $i = 1, 2, \ldots, n-1$ the beds number i and $i+1$ are adjacent and, moreover, the bed number n is adjacent to the bed number 1. Right now there are a_i rabbits staying at the bed number i and eating away Byteasar's lettuce.

Byteasar wants to chase out of the garden as many rabbits as possible. For this he is going to use his good old gun. The gun has k bullets inside. Rabbits are extremely timid, so whenever Byteasar shoots towards the bed number i, all the rabbits from that bed leave Byteasar's garden for good. What is more, the rabbits from both adjacent beds are so frightened that they all move to the adjacent bed (obviously, we mean the adjacent bed different from the one towards which was the shot).

Help Byteasar to find the maximum number of rabbits that he can chase out of his garden with at most k shots.

Input

The first line of the input contains two integers n and k ($5 \leqslant n \leqslant 2000$, $1 \leqslant k \leqslant n$) specifying the number of beds of lettuce in the garden and the number of bullets Byteasar has in his gun. The second line holds n integers $a_1, a_2, \ldots, a_n$ ($0 \leqslant a_i \leqslant 1\,000\,000$) representing the number of rabbits staying at the subsequent beds.

Output

Your program should output one integer: the maximum number of rabbits that can be chased out of Byteasar's garden using at most k shots.

Example

For the input data:

```
5 2
6 1 5 3 4
```

the correct result is:

```
13
```

Explanation of the example: First, Byteasar chases out the 6 rabbits from the bed number 1 (as a result, the rabbits from the bed number 5 move to the bed number 4, whereas the rabbits from the bed number 2 move to the bed number 3). Next, Byteasar chases out the 7 rabbits from the bed number 4.

Solution

Translation in progress. . .

2013

18th Polish Collegiate Programming Contest
Warsaw, October 25–27, 2013

A

The Motorway

★★★

Task author: Jakub Radoszewski
Solution description: Jakub Łącki
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2013/aut`

Autobyte Company is involved in the construction of one of Byteland motorways. Until recently the company collected toll charges only at the starting point of the motorway. However, Byteasar, the new chairman in charge of the company, noticed that in such a case the charged amount does not depend on the distance covered by customers in bytemiles. Therefore, the company plans to build toll-collecting points along the entire length of the motorway.

Byteasar, during his motorway trip, with the help of the odometer in his car, put down the location of all the n entry points (the position of an entry point is its distance from the start of the motorway). The company decided to locate $n + 1$ toll-collecting points evenly along the motorway. That means the distance between each two subsequent toll-collecting points would be the same. At the same time between each two such points there should be a motorway entry point and there should be a toll-collecting point between each two subsequent motorway entries. Luckily, it turned out that the existing location of entry points makes such arrangement possible.

Your task would be to calculate the minimum and maximum distance between toll-collecting points. Formally speaking, we are seeking the lowest and highest value for l, for which there exists a position b_0 of first toll-collecting point, such that the consecutive points should be located in $b_0 + l, b_0 + 2l, \ldots, b_0 + nl$ positions. It may be so that the location of a given toll-collecting point, determined by the above procedure, falls in exactly the same position, as the location of an entry point. In this case the toll booth would be positioned in close vicinity of an entry point, either just before or just after it. In other words, the position of the j-th entry point should be included in the following interval $[b_0 + (j-1)l, b_0 + jl]$.

Input

The first line of the input contains one integer n ($3 \leqslant n \leqslant 1\,000\,000$) specifying the number of motorway entry points. The second line of the input contains an increasing sequence of n integers $a_1, a_2, \ldots, a_n$ ($0 \leqslant a_i \leqslant 10^9$). The following sequence elements are the positions of subsequent motorway entry points.

Output

Your program should produce two real numbers presenting smallest and largest possible distance between two subsequent toll collecting points in bytemiles. You can assume that the difference between these values is not less than 10^{-9}.

Your result will be considered as being correct in case it is included in the interval $[x(1-\varepsilon) - \varepsilon, x(1+\varepsilon) + \varepsilon]$, where x is the correct answer and $\varepsilon = 10^{-8}$. Therefore both relative error and absolute error of the answer equal to ε will be accepted.

Example

For the input data:

```
6
2 3 4 5 6 7
```

the correct result is:

```
0.833333333333 1.250000000000
```

Solution

Translation in progress. . .

B

Bytehattan

★★★

Task author: Jakub Łącki
Solution description: Jakub Łącki
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2013/baj`

Bytehattan is one of the islands in the capital of Byteland. The island is a witness to parades, outings and processions organized frequently. In fact these are so persistent, that street closing and serious traffic congestion result. Byteasar, employed at the town hall, has been appointed to monitor the city traffic.

The streets of Bytehattan form a regular $n \times n$ grid. Let us look at the map of Bytehattan as if it were grid coordinates: for each pair of integers x, y, such that $1 \leqslant x, y \leqslant n$, at point defined by its coordinates (x, y) there is an intersection. Each two street crossings 1 unit away are joined by a street measuring 1 unit.

Byteasar receives messages concerning street closures. Each message informs that a particular street will be closed from now on. After receiving such information concerning the closure of a given street, Byteasar should determine whether it would be still possible to commute between two intersections which are located at the ends of such a closed street, using roads which have not yet been closed. Help him and create a program helping him with his job.

Input

The first line of the input contains two integers n and k ($2 \leqslant n \leqslant 1500$, $1 \leqslant k \leqslant 2n(n-1)$). They specify number of streets in Bytehattan and the number of messages concerning closed streets, respectively. Each of the following k lines contains information concerning the closure of one of the streets; information is provided in chronological order. Each of these lines consists of *two* streets described, one after the other. In practice exactly one of the streets becomes closed*. In case it is still possible to commute between two intersections which are located at the ends of such a closed street, described in the *previous line*, the first of these streets becomes closed. In case it is not possible, the second one is closed. The first closure, out of those k closures described in the input, applies to the first one out of the two streets listed. Each street can only be closed once.

The description of a given street is a pair of integers a_i, b_i ($1 \leqslant a_i, b_i \leqslant n$) followed by a letter c_i ($c_i \in \{\mathtt{N}, \mathtt{E}\}$). Such a triple determines a street, with one of its ends is positioned at an intersection described by coordinates (a_i, b_i). In case $c_i = \mathtt{N}$, the other end of the street is positioned at an intersection described by coordinates $(a_i, b_i + 1)$. In case $c_i = \mathtt{E}$, the other end of the street is positioned at an intersection described by coordinates $(a_i + 1, b_i)$. If $c_i = \mathtt{N}$, then $b_i < n$, similarly if $c_i = \mathtt{E}$, then $a_i < n$.

*The intention of the jury is that such an atypical input format would create the need for processing each street closure before starting the processing of subsequent street closures.

Output

Exactly k lines should be contained in the output. In case after the i-th street closure it is still possible to commute between the intersections on a closed street from the input, in the i-th line of output there should be the word `TAK` (i.e., *yes* in Polish). Otherwise the i-th line should contain word `NIE` (*no* in Polish).

Example

For the input data:

```
3 4
2 1 E 1 2 N
2 1 N 1 1 N
3 1 N 2 1 N
2 2 N 1 1 N
```

the correct result is:

```
TAK
TAK
NIE
NIE
```

Solution

Translation in progress...

C

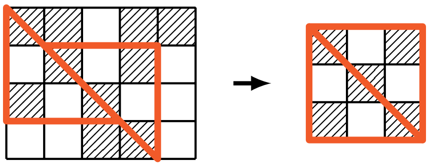
The Carpenter

★★★★

Task author: Tomasz Idziaszek
Solution description: Tomasz Idziaszek
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2013/cie`

Byteasar would fancy a game of checkers, however his chessboard got lost somewhere. He only managed to find a wooden board, sized $n \times m$, divided into nm equal in size, square fields. Each field is painted either white or black, however the arrangement of the colors on this board not necessarily matches the proper chessboard pattern. In such a case Byteasar has decided to utilise his carpentry experience and with the help of a saw he plans to cut out a chessboard, which is a square consisting of certain number of fields, where two fields sharing sides have alternate colors.

It is not clear whether Byteasar manages to find out a properly sized square on the board. So, he decided to cut out two triangular pieces from the board in order to glue them together in such a way that a chessboard would be created. (The pieces must be separable, however they may be turned around in any way after cutting out.) Help Byteasar and calculate the largest chessboard size that he is able to obtain by using this method. The figure below presents the board sized 4×5 and the two triangles, which could be glued together in order to form chessboard sized 3×3:



Input

The first line of the input contains two integers n and m ($1 \leqslant n, m \leqslant 1000$) specifying the board size. The following n lines contains m integers each: the j-th number from the i-th line ($1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$) describes the color positioned on the intersection of the j-th column and the i-th row of the board. Digit 0 describes white field and digit 1 — black field.

Output

The first and only line of the output should contain one integer, representing the largest chessboard size, which is obtainable by cutting two triangular pieces from the board and pasting them together.

Example

For the input data:

```
4 5
1 1 0 1 1
0 1 0 1 0
1 0 1 0 0
0 0 1 1 0
```

the correct result is:

```
3
```

whereas for the input data:

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

the correct result is:

```
2
```

Solution

Translation in progress...

D

Demonstrations ★★

Task author: Jakub Łącki
Solution description: Jakub Łącki
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2013/dem`

This Sunday, The Byte Day will be celebrated in Bytetown, one of the most important annual Bytelandian celebrations. However, everything indicates that this year jubilation will not only be an idyllic family fete.

Well, Bytetown citizens are strongly divided concerning one crucial matter. Some believe that in line with tradition, the byte should always be equal to eight bits. However there are progress supporters who would rather go for more capacious, 16-bit bytes. Others see the whole matter much more rigidly and would eagerly like to declare that the byte should always have only four bits. Finally there are less significant subversive movements in Bytetown, whose members advocate that the count of bits in the byte should not be the power of two, or yet it must not necessarily be an even number! All of these societies plan to hold their own manifestation in order to convince Bytetown citizens to their cases.

Many Bytetown citizens are afraid that such a number of demonstrations might interfere with the The Byte Day celebrations. The Lord Major of Bytetown sensed a significant public support could be gained, by forbidding some of the demonstrations. Due to the fact that such decisions raise controversy, the Lord Mayor decided he would only cancel two demonstrations. But at the same time, he would like to be able to choose such demonstrations for cancellation, that would result in the total time taken by any other possible demonstrations taking place in the city after the cancellation, to possibly be shortest. Help the Lord Mayor and give him a clue how much time in the city without a demonstration he can achieve.

Input

The first line of the input contains one integer n ($2 \leqslant n \leqslant 500\,000$) specifying the number of planned demonstrations. Each of the subsequent n lines describes one demonstration: the i-th of those lines contains two integers a_i and b_i ($0 \leqslant a_i < b_i \leqslant 10^9$), which mean that the i-th demonstration begins a_i byteminutes after sunrise and ends b_i byteminutes after sunrise.

Output

Your program should produce exactly one non-negative integer, describing by how much time demonstrations taking place could possibly be shortened, in case the Lord Mayor of Bytetown cancels maximum two demonstrations.

Example

For the input data:

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

the correct result is:

```
4
```

Explanation of the example: Lord Mayor of Bytetown should not issue permits for the first and the fourth demonstration.

Solution

Translation in progress...

E

The Exam ★

Task author: Jakub Łącki
Solution description: Tomasz Idziaszek
Available memory: 128 MB
https://oi.edu.pl/en/archive/amppz/2013/egz

Professor Byteoni is preparing *Bit & Byte Theory* exam. He has already prepared n questions. Each of these questions has been ranked by the professor with an expected difficulty coefficient, that is a natural number ranging from 1 to n. Each of the questions holds a different coefficient.

Now the professor is considering the exam questions sequence. Professor wishes to determine whether his students are able to judge the question difficulty by themselves. For this purpose he plans to line up his questions in such a way, that coefficients of subsequent questions differ at least by k. Help the professor to find such a sequence.

Input

The first and only line of the input contains two integers n and k ($2 \leqslant n \leqslant 1\,000\,000$, $1 \leqslant k \leqslant n$) specifying the number of questions prepared by the professor and the lower limit of the difficulty difference of subsequent exam questions.

Output

Your program should output one line containing sought question difficulty coefficients sequence, in other words a sequence of n pairwise distinct natural numbers ranging from 1 to n, where each two subsequent numbers differ at least by k. If there are multiple correct answers, your program should write any one of these. In case the sought sequence does not exist, your program should write only one word NIE (i.e., *no* in Polish).

Example

For the input data:

```
5 2
```

one of the correct results is:

```
2 4 1 5 3
```

whereas for the input data:

```
5 4
```

the correct result is:

```
NIE
```

Solution

Translation in progress...

F

Speed Cameras ★★

Task author: Tomasz Idziaszek
Solution description: Tomasz Idziaszek
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2013/fot`

The Lord Mayor of Bytetown plans to locate a number of radar speed cameras in the city. There are n intersections in Bytetown numbered from 1 to n, and $n-1$ two-way street segments. Each of these street segments stretches between two intersections. The street network allows getting from each intersection to any other.

The speed cameras are to be located at the intersections (maximum one per intersection), wherein the Lord Mayor wants to maximise the number of speed cameras. However, in order not to aggravate Byteland motorists too much, he decided that on every route running across Bytetown roads that does not pass through any intersection twice there can be maximum k speed cameras (including those on endpoints of the route). Your task is to write a program which will determine where the speed cameras should be located.

Input

The first line of the input contains two integers n and k ($1 \leqslant n, k \leqslant 1\,000\,000$) specifying the number of intersections in Bytetown and maximum number of speed cameras which can be set up on an individual route. The $n-1$ lines that follow describe Bytetown street network: the i-th line contains two integers a_i and b_i ($1 \leqslant a_i, b_i \leqslant n$), meaning that there is a two-way street segment which joins two intersections numbered a_i and b_i.

Output

The first line of the output should produce m: the number describing the maximum number of speed cameras, that can be set up in Bytetown. The second line should produce a sequence of m numbers describing the intersections where the speed cameras should be located. Should there be many solutions, your program may output any one of them.

Example

For the input data:

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

one of the correct results is:

```
3
1 2 4
```

Solution

Translation in progress...

G

Marbles

★★★★

Task author: Jakub Radoszewski
Solution description: Jakub Radoszewski
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2013/gra`

Byteted and Bited decided to play marbles. There is an ever number of marbles in the urn. Each marble has been marked by exactly one digit. The rules of the game are very simple: the players take on random one marble each from the urn in turns. The game ends when the urn is empty. The player who has accumulated a set of marbles with a larger product of digits, wins.

The boys very much got to like this game. They are both very ambitious and they really like to win, so a draw makes nobody happy. Byteted and Bited are determined to avoid such an ending situation at all costs. Write a program which will check if for a given initial set of marbles in the urn, the game can end up drawn.

Input

The first line of the input contains one integer t ($1 \leqslant t \leqslant 1000$), specifying the number of test cases to be considered.

Each of the following t lines contains ten non-negative integers $k_0, \ldots, k_9$ ($0 \leqslant k_i \leqslant 10^{15}$), where k_i specifies the number of marbles marked with digit i. The sum of the numbers k_i is even and positive in each test case.

Output

Your program should produce t lines containing answers to respective test cases. The result for each test case that can end with a draw is word `TAK` (i.e., *yes* in Polish). In the opposite case the result should be `NIE` (*no* in Polish).

Example

For the input data:

```
5
0 1 0 1 1 4 1 0 5 1
0 1 1 0 3 0 0 0 0 3
1 1 0 4 0 0 2 0 0 2
100000 100000 100000 100000 100000 100000 100000 100000 100000 100000
0 99999 99999 100000 100000 100000 100000 100000 100000 100000
```

the correct result is:

```
TAK
NIE
NIE
TAK
NIE
```

Solution

Translation in progress...

H

The Hero

★★★

Task author: Tomasz Idziaszek
Solution description: Jakub Łącki
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2013/her`

Byteotheus, most famous Byteotian hero, once again emerged victorious from the battle. While his crew are loading the ship up with the acquired valuables, in his cabin, Byteotheus plans his way back to his homeland island — the Bitaca. It is not an easy task. Many gods envy Byteotheus popularity among the people and gladly would take him down a peg or two. Fortunately, some of them look favourably on him, especially goddess Bythena. It was none other but her that sent Byteotheus a dream last night, warning him of the dangers that he could encounter.

There are n islands on the Byteonian Sea. It will be convenient to number those from 1 to n. Presently Byteotheus's ship is at island 1, and its destination is the Bitaca — island n. Some pairs of islands are joined by *one-way* sea routes, although each of those islands is a start point for maximum of 10 sea routes. We are numbering the sea routes from 1 to m; the i-th route leads from island a_i to island b_i, and it takes exactly d_i days to cover it. In case the ship set sail on the i-th route, starting from island a_i at dawn on day j, it will reach its destination island b_i at dawn, at day $j + d_i$. The ship can stop at any island for an indefinite period before moving on again. However, before reaching a successive island, it cannot deviate off the set path, and sail no longer that is required to cover the particular route. Byteotheus can start his voyage from island 1 at dawn on the first day, at the earliest.

The goddess Bythena warning has been very precise. She provided Byteotheus an exact list of p traps, prepared by the gods. Every trap is situated on a certain island and is active for a certain time period. To be more precise, the i-th trap is located on island w_i and is active from day s_i until day k_i (inclusive). The traps are really dangerous — in case Byteotheus's ship finds itself on an island with an active trap, no one will survive. Luckily his homeland Bitaca is free from traps, and no traps on the island 1 are active on the first day.

Obviously Byteotheus wants to plan his way home, to avoid all traps. He wonders, however, how much longer he would need for his voyage because of them. Help him and indicate the minimum number of days necessary to safely return to Bitaca.

Input

The first line of the input contains two integers n and m ($2 \leqslant n \leqslant 100\,000$, $1 \leqslant m \leqslant 1\,000\,000$) specifying the number of islands and the number of sea routes. Subsequent m lines describe the sea routes: the i-th line contains three integers a_i, b_i, d_i ($1 \leqslant a_i, b_i \leqslant n$, $a_i \neq b_i$, $1 \leqslant d_i \leqslant 10^9$), indicating that the i-th route leads from island a_i to island b_i and it takes d_i days. All routes are one way. Every island is a start point for maximum of 10 sea routes.

The next line contains integer p ($0 \leqslant p \leqslant 100\,000$), describing the number of the traps. Next p lines hold the description of the traps: in the i-th line there are three integers w_i, s_i, k_i ($1 \leqslant w_i < n$, $1 \leqslant s_i \leqslant k_i \leqslant 10^9$), indicating that the i-th trap is located on the island w_i and is active from day s_i until and including day k_i. If $w_i = 1$, then $s_i > 1$.

Output

In case it is not possible to plan the route avoiding all the traps, the one and only line should output word NIE (i.e., *no* in Polish). In the opposite case, an integer d should be output, describing the minimum number of days required to finalise the voyage (the ship reaches Bitaca on day $d + 1$ at sunrise).

Example

For the input data:

```
5 6
1 2 3
1 4 13
2 3 1
2 4 2
3 2 2
4 5 1
5
1 2 4
1 8 8
2 6 7
2 10 11
4 6 7
```

the correct result is:

```
10
```

Explanation of the example: Byteotheus set sail from island 1 on the first day, at sunrise. He arrives on island 2 on the fourth day. There he waits one day and starts off for island 3. After getting there on the sixth day, he immediately turns back to island 2, where from he travels in the direction of island 4 on the eighth day. He arrives there on the tenth day and finally reaches Bitaca on the eleventh day.

Solution

Translation in progress...

I

Genetic Engineering ★★

Task author: Tomasz Idziaszek
Solution description: Jakub Łącki
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2013/inz`

Byteotian paleoarchaeologists recently unearthed a few pieces of amber, which had trapped ancient mosquitoes inside. After analysing the samples of insects it turned out that they come from the Jurassic period, and therefore likely to have been in contact with large reptiles that dominated the Byteotian lands. This gave geneticists a quaint idea: to try to recover byteoraptor genetic material from the blood of mosquitoes.

Byteoraptor genome, as in all Bytean organisms, is a chain consisting of a number of byteo-aminoacids. For simplicity we denote the types of byteo-aminoacids by natural numbers. Redundancy occurs in a genome — every type of byteo-aminoacid is repeated k times (specifically, the length of each valid genome is a multiple of k). In other words, if we divide the genome into blocks consisting of subsequent k byteo-aminoacids, each block will contain byteo-aminoacids of the same kind.

Geneticists were able to isolate a suspected chain consisting of byteo-aminoacids, from the blood of a mosquito, being n in length. Unfortunately, the chain may not be a valid genome — scientists suspect that it may have been contaminated by foreign byteo-aminoacids. Presently they want to test their hypothesis and remove the least byteo-aminoacids from that chain, such that a valid genome emerges. In case of many equally good possibilities, the researchers are interested in the genome that is the earliest in lexicographical order*. Your task is to help them to make a breakthrough discovery.

Input

The first line of the input contains two integers n and k ($1 \leqslant n \leqslant 1\,000\,000$, $2 \leqslant k \leqslant 1\,000\,000$) specifying the length of extracted chain of byteo-aminoacids and redundancy degree of a valid genome. The second line contains a sequence of n integers $g_1, \ldots, g_n$ ($1 \leqslant g_i \leqslant 1\,000\,000$), representing the types of subsequent byteo-aminoacids in the chain.

Output

The output should contain two lines. The first one should contain the number m ($0 \leqslant m \leqslant n$) specifying the length of the longest valid genome, which may arise by removing some byteo-aminoacids from the specified chain.

The second line should contain a sequence of m numbers describing the types of subsequent byteo-aminoacids in the valid genome. In case there are multiple

*Let l_1 and l_2 be two different chains of the same length, consisting of byteo-aminoacids. To determine which one is earlier in lexicographical order, it is necessary to find the first position where the chains differ. The chain earlier in the lexicographical order is the one which has byteo-aminoacid marked with a lower number in this position.

solutions, your program should output the smallest lexicographically. If $m = 0$ (i.e. geneticists have failed to isolate any nonempty valid genome), the second line of output should be empty.

Example

For the input data:

```
16 3
3 2 3 1 3 1 1 2 4 2 1 1 2 2 2 2
```

the correct result is:

```
9
1 1 1 2 2 2 2 2 2
```

Solution

Translation in progress...

J

Jánošík

★

Task author: Tomasz Idziaszek
Solution description: Tomasz Idziaszek
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2013/jan`

Jánošík, otherwise known as Robin Hood, takes from the rich to give to the poor. Together with his gang they robbed a convoy carrying gold to the counts' castle and n caskets fell prey to robbers. After transporting their loot to the cave it turned out that the i-th casket (for $i = 1, 2, \ldots, n$) contains exactly i money-bags full of gold.

In case a poor man comes to Jánošík asking for a few gold ducats, Jánošík utilises the following procedure. First he chooses a nonempty casket that contains the smallest number of money-bags with gold. In case the casket contains exactly one money-bag, Jánošík hands it to the man in need, and sees him go away happily. Otherwise, if the casket contains an odd number of money-bags, Jánošík puts one of the money-bags in his pocket, and starts the whole process again. However, in case there is an even number of money-bags, Jánošík takes exactly half of them out and puts them in an empty casket (luckily the empty caskets are plentiful in the cave) and begins the whole procedure anew. Therefore if a penniless man comes to Jánošík, and in case he still will be in a possession of at least one nonempty casket, as a result of (possibly multiple) employment of Jánošík's procedure, the poor man is sure to get a money-bag full of gold. The poor would come to the Jánošík's cave until all the caskets are empty.

Fellow robbers from Jánošík's gang wonder if their leader does not ruin the good name of thugs with his behaviour. They want to know how many looted money-bags remain in Jánošík's pocket when all the caskets are empty.

Input

The first and only line of the input contains one integer n ($1 \leqslant n \leqslant 10^9$) specifying the number of caskets robbed by Jánošík's gang.

Output

The first and only line of output should contain an integer representing the number of money-bags, which will remain in Jánošík's pocket after emptying all the caskets.

Example

For the input data:

```
5
```

the correct result is:

```
2
```

Solution

Translation in progress...

K

Blankets

★★★

Task author: Jakub Łącki
Solution description: Eryk Kopczyński
Available memory: 128 MB
`https://oi.edu.pl/en/archive/amppz/2013/koc`

This summer, Byteburg citizens are turning out in droves at the city beach down by the Byteotian Lake to experience the joy of sunbathing. Every Byteburg citizen arrives at the beach equipped with the blanket manufactured by *Byteasar & Son*, the trendiest this season. All blankets are of equal size $a \times b$ (although of different patterns), and each sunbather sets out his or her blanket in such a way that its longer side is always perpendicular to the lake.

One of this year's sunbathers is professor Byteoni. After a few days of sunbathing professor noticed, that all the people who come to the beach always set out their blankets in their own favourite individual places. Although people come to the beach and leave it at different times, the professor never heard that any sunbather had taken over somebody else's favourite place by putting the blanket there. This observation made the professor so curious, he decided to study this phenomenon.

For that purpose he set a coordinate system on the beach, and for every of the n Byteburgians noted down the coordinates of each of the spots where individual citizens always put their blankets. The system is devised in such a way that the OX-axis is parallel to a sides, and the OY-axis to b sides of all of the blankets. The professor initially wanted to calculate the area of intersection of the areas occupied by the blankets for each pair of them. But then he realized that it is enough for further research that he has only the *average* of these values. In other words, he is interested in the expected value of the area of intersection of the fields occupied by blankets belonging to two different random people of Byteburg. Using the data provided by the professor, help him do the calculation.

Input

The first line of the input contains three integers n, a and b ($2 \leqslant n \leqslant 200\,000$, $1 \leqslant a, b \leqslant 1\,000\,000$) specifying the number of Byteburg inhabitants and the sizes of the blankets, respectively. Each of the subsequent n lines contains two integers x_i and y_i ($0 \leqslant x_i, y_i \leqslant 1\,000\,000$) indicating the coordinates of the point where the i-th Byteburg citizen always puts the lower left corner of his or her blanket.

Output

Your program should print one real number: the average area of intersection of the areas occupied by the blankets belonging to pairs of Byteburg inhabitants. Your result will be deemed valid if it is in the range $[x - \varepsilon, x + \varepsilon]$, where x is the correct answer and $\varepsilon = 10^{-2}$.

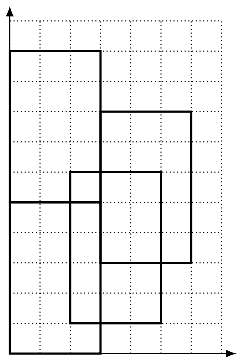
Example

For the input data:

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

the correct result is:

```
1.833333333
```



Explanation of the example: The exact result is $\frac{4+0+0+1+6+0}{6} = 1\,\frac{5}{6}$.

Solution

Translation in progress...

2014

19th Polish Collegiate Programming Contest
Warsaw, October 24–26, 2014

A

The Lawyer ★

Task author: Jakub Łącki
Solution description: Jakub Łącki
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2014/adw`

Byteasar is an advocate, co-owner of the law firm *Byteasar & Associates*. He is one of the most sought-after members of the Byteotian Bar. Not surprisingly, he is always extremely busy. Every day he is involved in a number of meetings and has long since ceased to control whether he will be able to participate in all of them. Therefore he has hired a secretary, whose job is to help him to control this chaos. Byteasar decided that every day he will be taking part in two meetings only, however his participation would be complete, that is, from the very beginning to the very end. The remaining meetings will be taken care of by assistants, who are quite plentiful at Byteasar's office.

Unfortunately, it is sometimes difficult even to find two meetings which do not overlap in Byteasar's busy schedule. We assume that two meetings do not overlap, if one of them starts *strictly after* the previous has finished. Help Byteasar's secretary and write a program that can deal with this problem.

Input

The first line of the input contains two integers n and m ($2 \leqslant n \leqslant 500\,000$, $1 \leqslant m \leqslant 20$) specifying the number of meetings in Byteasar's schedule and the number of days included in it.

Each of the next n lines describes one meeting. Description of the meeting consists of three integers a_i, b_i, d_i ($1 \leqslant a_i < b_i \leqslant 80\,000\,000$, $1 \leqslant d_i \leqslant m$) which indicate that on the day d_i Byteasar has a meeting scheduled that starts exactly a_i milliseconds after midnight and ends b_i milliseconds after midnight.

Output

Your program should output m lines. The i-th of these lines should contain information, whether Byteasar is able to attend two meetings on the i-th day. In case it is not possible, one word `NIE` (i.e., *no* in Polish) should be produced. Otherwise, the word `TAK` (*yes* in Polish) should be output, followed by the numbers of the two meetings in which Byteasar can participate. Meetings are numbered from 1 to n, in accordance to their order at the input. The first of these two meetings should start earlier. The second meeting should start at least a millisecond after the first one is finished.

If there are multiple correct answers, your program should output any one of them.

Example

For the input data:

```
6 3
3 5 1
2 4 2
1 8 1
6 7 3
3 5 2
7 12 1
```

the correct result is:

```
TAK 1 6
NIE
NIE
```

Solution

Translation in progress...

B Petrol ★★★

Task author: Jakub Łącki
Solution description: Adam Karczmarz
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2014/ben`

Byteasar is employed in the logistics department of Byteonian petroleum giant Byteoil. His job is to plan fuel deliveries to petrol stations.

There are n intersections in Byteotia (marked by numbers from 1 to n) and m two-way roads connecting certain pairs of intersections. There are Byteoil petrol stations located at some intersections.

Byteoil transport fleet consists of petrol tankers with various capacities of fuel tanks. Each tanker consumes 1 litre of petrol per kilometre of distance travelled. It can therefore be assumed that a tanker having tank capacity of b litres can cover a maximum distance of b kilometres without the need to refuel. Drivers cannot use the fuel carried by the tanker in the main tank, however they can refuel at any Byteoil petrol stations free of charge.

Byteasar's work consists of repeatedly answering the following queries: Is a petrol tanker with a fuel tank of capacity of b litres capable of covering the distance from a petrol station located at the intersection x to a petrol station located at the intersection y? Tanker with a fuel tank of capacity of b litres cannot cover a distance of more than b kilometres, during which there will be no Byteoil petrol station. Tankers starting point is always located at an intersection where Byteoil petrol station is present, and also all the trips end at an intersection with Byteoil petrol station.

Help Byteasar to provide an automated reply to his logistic queries.

Input

The first line of the input contains three integers n, s and m ($2 \leqslant s \leqslant n \leqslant 200\,000$, $1 \leqslant m \leqslant 200\,000$), specifying the number of intersections, the number of petrol stations and the number of roads in Byteotia, respectively. The second line contains a sequence of s pairwise distinct integers $c_1, c_2, \ldots, c_s$ ($1 \leqslant c_i \leqslant n$), specifying the intersections where Byteoil stations are located.

The next m lines describe the roads in Byteotia; the i-th of these lines contains three integers u_i, v_i and d_i ($1 \leqslant u_i, v_i \leqslant n$, $u_i \neq v_i$, $1 \leqslant d_i \leqslant 10\,000$), indicating that the i-th road has a length of d_i kilometres and connects the intersection u_i with the intersection v_i. Each pair of intersections is connected by at most one road.

The next line contains one integer q ($1 \leqslant q \leqslant 200\,000$), specifying the number of queries. The consecutive q lines hold the descriptions of queries; the i-th of these lines contains three integers x_i, y_i and b_i ($1 \leqslant x_i, y_i \leqslant n$, $x_i \neq y_i$, $1 \leqslant b_i \leqslant 2 \cdot 10^9$) indicating query concerning the possibility of a tanker with a capacity of b_i litres to cover the distance from a petrol station at the intersection x_i to the station at the intersection of y_i. It can be assumed that at both intersections x_i, y_i Byteoil petrol stations are located.

Output

Your program should output exactly q lines. The i-th of these lines should contain one word `TAK` (i.e., *yes*) or `NIE` (i.e., *no*), depending on whether the tanker with a fuel tank with a capacity of b_i is able to travel from the intersection x_i to the intersection y_i.

Example

For the input data:

```
6 4 5
1 5 2 6
1 3 1
2 3 2
3 4 3
4 5 5
6 4 5
4
1 2 4
2 6 9
1 5 9
6 5 8
```

the correct result is:

```
TAK
TAK
TAK
NIE
```

Solution

Translation in progress...

C

The Prices

★★

Task author: Jakub Radoszewski
Solution description: Jakub Radoszewski
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2014/cen`

Byteasar works as a purchasing manager at one of Byteotian restaurants. Every evening he receives a shopping list from his manager. Food products must be purchased the next day, in the morning. Byteasar should buy exactly one piece of each product from the list. Manager is always pressing that the total costs are as little as possible.

Byteasar sits down in the evening with his computer and checks the prices of all needed products at local grocery wholesalers. He also knows the cost of the trips from the restaurant to each wholesale and back. Now Byteasar must decide which products are to be purchased at each of the warehouses.

For each wholesale, where Byteasar decided to buy some products, he does as follows. He goes from the restaurant to the warehouse, does the shopping, and immediately brings the purchased products back to the restaurant. Luckily, the boot of his car is big enough that it eliminates the need to visit any of the warehouses more than once, as all the purchased goods can be delivered in one go. Food products are highly perishable, so during one trip Byteasar can make purchases only at one warehouse.

Write a program that will help Byteasar to calculate the cheapest way of making all the purchases.

Input

The first line of the input contains two integers n and m ($1 \leqslant n \leqslant 100$, $1 \leqslant m \leqslant 16$) specifying the number of wholesalers and the number of products that Byteasar should buy. Next n lines contain descriptions of the prices at individual wholesalers.

The first number in the i-th of these lines, d_i ($1 \leqslant d_i \leqslant 1\,000\,000$), describes the trip cost from the restaurant to the i-th warehouse (including the return cost). It is followed by a sequence of m integers $c_{i,1}, c_{i,2}, \ldots, c_{i,m}$ ($1 \leqslant c_{i,j} \leqslant 1\,000\,000$): number $c_{i,j}$ specifies the price of j-th product in the i-th warehouse.

Output

Your program should output a single line containing a single integer: the sum of the product prices and the cost of the trips to warehouses selected by Byteasar in the cheapest possible purchase plan.

Example

For the input data:

```
3 4
5 7 3 7 9
2 1 20 3 2
8 1 20 1 1
```

the correct result is:

```
16
```

Explanation of the example: Byteasar buys the product number 2 in the first warehouse, and all the other products in the second one. So he does not have to visit the third warehouse.

Solution

Translation in progress. . .

D

Divisors ★

Task author: Jakub Łącki
Solution description: Jakub Łącki
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2014/dzi`

You are given a sequence of n integers $a_1, a_2, \ldots, a_n$. You should determine the number of such ordered pairs (i, j), that $i, j \in \{1, \ldots, n\}$, $i \neq j$ and a_i is a divisor of a_j.

Input

The first line of the input contains one integer n ($1 \leqslant n \leqslant 2\,000\,000$). The second line contains a sequence of n integers $a_1, a_2, \ldots, a_n$ ($1 \leqslant a_i \leqslant 2\,000\,000$).

Output

The first and only line of the output should contain one integer: the number of pairs sought.

Example

For the input data:

```
5
2 4 5 2 6
```

the correct result is:

```
6
```

Explanation of the example: There are 6 pairs with the specified properties: $(1, 2)$, $(1, 4)$, $(1, 5)$, $(4, 1)$, $(4, 2)$, $(4, 5)$.

Solution

Translation in progress...

E

Euclidean Nim

★★

Task author: Tomasz Idziaszek
Solution description: Adam Karczmarz
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2014/euk`

Euclid and Pythagoras are pseudonyms of two Byteotians known for their love of mathematical puzzles. Lately, they spend evenings playing the following game. There is a heap of n stones on the table. Friends perform alternating moves. Euclid's move consists of taking any positive multiple of p stones from the heap (providing the heap contains at least p stones) or adding exactly p stones to the heap — adding the stones is possible, however, only in case the heap contains less than p stones. Pythagoras' move is analogous, except that either he takes a multiple of q stones, or adds exactly q stones. The winner is the player who empties the heap. Euclid begins the game.

Friends wonder whether they have worked out this game perfectly. Help them and write a program that will state what should be the result of the game, providing both players are making optimal moves.

Input

The first line of the input contains one integer t ($1 \leqslant t \leqslant 1000$) specifying the number of test cases described in the following part of the input. Description of one test case consists of one line containing three integers p, q and n ($1 \leqslant p, q, n \leqslant 10^9$).

Output

The output should include exactly t lines containing answers to the subsequent test cases from the input. The answer should be one letter `E` (if Euclid could bring about his victory, regardless of the Pythagoras' movements), `P` (if Pythagoras could bring about his victory, regardless of Euclid's moves) or `R` (for *remis*, i.e. *draw* in Polish, if the game will be played infinitely).

Example

For the input data:

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

the correct result is:

```
P
P
E
R
```

Explanation of the example: In the game from the first test case Euclid must add three stones to the heap in his move. Thanks to that Pythagoras can collect all 4 stones in his move and thus win.

Solution

Translation in progress...

F

Pillars

★★★★

Task authors: Jakub Radoszewski, Tomasz Idziaszek
Solution description: Tomasz Idziaszek
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2014/fil`

Byteasar is the administrating manager at a large warehouse. Anticipating a severe winter, he decided to install underfloor heating system in the warehouse.

The plan of the warehouse is a rectangle of even dimensions $n \times m$ divided into unit squares. Most of the unit squares comprise warehouse space, but some of them are occupied by massive pillars providing additional constructional support for the warehouse structure. Each pillar occupies a square 2×2 on the warehouse plan, composed of unit squares. Pillars are not arranged too densely — centers of any two of them are located at least 6 units away (in Euclidean metric). Additionally, the centre of each pillar is located at least 3 units away from each of the outer warehouse walls.

Heating will be accomplished by using one heating pipe installed under the floor of the warehouse. The pipe is to run through the centres of all unit squares except the unit squares occupied by pillars. Each pipe section must run parallel to one of the walls of the hall and the turns could be located only at the centres of the unit squares. The pipe must begin and end in the same place. At this point the cold water would be discharged outside and hot water fed into the pipe.

Byteasar has asked you to plan the pipe layout in the warehouse. To help you, he introduced Cartesian coordinate system onto the warehouse plan, where the abscissae belong to the interval $[0, n]$, and ordinates to the interval $[0, m]$. The coordinates of the centres of all unit squares are numbers in the form $k + \frac{1}{2}$ for $k \in \mathbb{N}$.

Input

The first line of the input contains three integers n, m and f ($1 \leqslant n, m \leqslant 1000$ and n and m are even) indicating the warehouse dimensions and the number of the pillars. Each of the next f lines contains two integers x_i and y_i ($0 \leqslant x_i \leqslant n$, $0 \leqslant y_i \leqslant m$) specifying the coordinates of the centre of the i-th pillar.

Output

In the first line of the output your program should produce one word `TAK` (i.e., *yes*) or `NIE` (i.e., *no*) depending on whether the implementation of floor heating in line with Byteasar's requirements is achievable, or not. In case the answer is `TAK`, the second line should contain a description of the exemplary plan of the pipe layout in the form of a string of $nm - 4f$ letters. We agree to assume that the beginning of the pipe is located at the point with the coordinates $(\frac{1}{2}, \frac{1}{2})$. Following parts of the pipe are marked as follows: transition by the vector $[0, 1]$ is denoted by a letter `G`, by the vector $[0, -1]$ is denoted by `D`, by the vector $[1, 0]$ is denoted by `P`, and by the vector $[-1, 0]$ is denoted by `L`. In case there are multiple correct answers, your program should output any of them.

Example

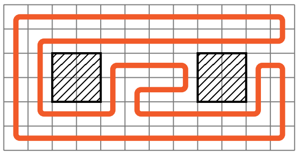
For the input data:

```
12 6 2
3 3
9 3
```

the correct result is:

```
TAK
PPPPPPPPPPPGGGLDDLLLLLGPPGLLLDDLLLGGGPPPPPPPPPPGLLLLLLLLLLLDDDDD
```

Example output corresponds to the following figure:



Solution

Translation in progress. . .

G

Global Warming

★★★

Task authors: Jacek Tomasiewicz, Tomasz Idziaszek
Solution description: Tomasz Idziaszek
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2014/glo`

Professor Byteoni prepares a report for Interbyteotian Commission for Climate Change. This report should demonstrate clearly, unambiguously, and without any doubt, the impact of Byteotian population on climate changes in the region. Although the professor has a lot of empirical data, but to pervade the mainstream media, it not enough to present substantive arguments. Equally important is to present data clearly and correctly targeted. To this end, he wants to select the data, to be presented on the main chart in the report, thoughtfully and in a deliberate way.

The key graph will contain information about the average air temperature throughout the years. The professor has data available concerning the mean annual temperatures for the last n years. He wants to describe this graph using the following comment: “in the year r_{min} the temperature was the lowest, and in the year r_{max} it was the highest, therefore, it is clear that…”. Unfortunately, he fears that the same minimum or maximum temperature can occur several times throughout this period, and therefore such information could undermine the strength of his statement.

The professor has therefore decided to present only part of the data on his chart. His idea is to select the year range in such a way that this interval will include exactly one year with a minimum temperature *within that range* and exactly one year with a maximum temperature also *within that range*. The selected range may not include the year with the globally maximum or minimum average temperature in the last n years (or none of them). Of course, the professor would like to put as much data as possible on this chart, so he is interested in the longest year span.

Input

The first line of the input contains one integer n ($1 \leqslant n \leqslant 500\,000$), specifying the number of years for which the professor knows the average temperatures. The second line contains a sequence of n integers $t_1, t_2, \ldots, t_n$ ($-10^9 \leqslant t_i \leqslant 10^9$). The number t_i specifies the average temperature in the i-th year.

Output

The output should contain two integers l and k. They indicate that the longest interval satisfying the professor’s conditions is of length of l years, and the earliest year in which such an interval may begin is k.

Example

For the input data:

```
10
8 3 2 5 2 3 4 6 3 6
```

the correct result is:

```
6 4
```

Explanation of the example: The chart will present temperatures 5, 2, 3, 4, 6, 3. This interval contains exactly one year with a minimum temperature of 2 and one year with a maximum temperature of 6.

Solution

Translation in progress. . .

H

Hit of the Season ★★★★

Authors: Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń
Solution description: Tomasz Kociumaka
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2014/hit`

Byteotian Printing Factory (BPF) has received a large production order for striped wallpaper. Striped wallpaper is the hit of the season in interior design. Each wallpaper has n equal width vertical stripes colored red, green and blue. BPF is to take care of the design and printing of wallpapers. The customer has described the color of certain wallpaper stripes; for the remaining stripes, the customer allowed BPF a complete freedom.

BPF uses matrices to print a certain number of consecutive stripes on the wallpaper. The matrix has a specific color of each of the printed stripes. The matrix may be shorter than the entire wallpaper. With each application, the matrix stripes must match and overlay the wallpaper stripes; then printing of all the matrix stripes takes place. In this way, a single wallpaper stripe can be printed over more than once. In case a given stripe is printed over using different colors, the final color is a blend of these colors. The matrix prints only in one orientation and must not in any way be rotated.

BPF employees, irrespective of their sense of aesthetics, would primarily like to design the shortest possible matrix that will allow printing the entire wallpaper. They must bear in mind that in the case of stripes defined by the customer they must use pure color, without any addition of any other color. In other words, for each matrix application printing over such a single-color stripe, the matrix stripe color must be exactly as defined by the client. No stripe on the wallpaper can remain colorless.

Input

The first line of the input contains one integer t ($1 \leqslant t \leqslant 10$) specifying the number of test cases. Each of the next t lines describes a single test case and contains a string of upper-case letters `R`, `G`, `B` and asterisks (`*`), specifying the desired wallpaper appearance. The letters specify the color of the stripes, and the asterisks mark the stripes, the color of which has not been specified by the client. A string is not empty, it consists of a maximum of 3000 characters and contains at most 19 asterisks.

Output

For each of the test cases your program should output one line containing a string of characters `R`, `G`, `B`: minimum length matrix that allows printing the desired wallpaper. If for a given test case, there are many correct solutions, your program should output any of them.

Example

For the input data:

```
1
RRG*R*BRR**B
```

the correct result is:

```
RRGB
```

Solution

Translation in progress...

I

The Staging ★★★★

Task author: Adam Karczmarz
Solution description: Adam Karczmarz
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2014/ins`

Steven Byteberg is a movie director, specialising in action movies. Presently he is working on a new movie, whose theme is Byteonian Mafia wars. Byteberg wonders what should be the final shape of the movie climax scenes, which will be a spectacular gunfire exchange.

There are n gangsters participating in this scene, numbered, for simplicity, using consecutive numbers from 1 to n. When the tension reaches its critical point, each gangster pulls out his weapon and takes aim at another gangster. Nobody gets aimed at by more than one gangster. Gangsters are poor, but well trained — each can shoot only once, however this shot is always accurate and always deadly.

At some point, one of the thugs cannot withstand the tension any longer and the shooting starts.

The director has determined the initial order in which gangsters have to pull the triggers. Namely, the gangster i shoots towards the gangster p_i at the precise moment t_i, unless gangster i already has been killed by that time. The gangster is killed exactly at the same moment when someone shoots in his direction.

The director would like to know how many gangsters will be alive at the end of this scene. Byteberg is not yet completely certain concerning the order in which the gangsters have to shoot. From time to time he commands to change one of the values t_i. After every such change he would like to know the number of gangsters who would survive, referring to the new order in which gangsters shoot (taking into account all changes made so far).

Input

The first line of the input contains one integer n ($2 \leqslant n \leqslant 200\,000$), specifying the number of gangsters involved in the scene. The second line contains n integers $p_1, p_2, \ldots, p_n$ ($1 \leqslant p_i \leqslant n$, $p_i \neq i$, $p_i \neq p_j$ for $i \neq j$) describing whom successive gangsters intend to shoot at.

The third line contains n integers $u_1, u_2, \ldots, u_n$ ($1 \leqslant u_i \leqslant 10^9$), describing the initial order in which the gangsters are shooting: the initial value t_i is equal to u_i.

The fourth line contains one integer q ($0 \leqslant q \leqslant 200\,000$), specifying the number of the changes of the values of $t_1, \ldots, t_n$ as planned by Byteberg. Next q lines contain a description of these changes. The i-th line contains two integers k_i and v_i ($1 \leqslant k_i \leqslant n, 1 \leqslant v_i \leqslant 10^9$), describing that the i-th change consists in setting the value of t_{k_i} to v_i. Numbers $u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_q$ are pairwise distinct.

Output

Your program should produce exactly $q + 1$ lines. The first line should contain the number of gangsters who survive the shooting, assuming the initial order of shooting. The i-th of the following q lines should present the number of survived gangsters, assuming that the order of shooting is determined by the sequence $t_1, \ldots, t_n$ after performing all the changes from the first to the i-th.

Example

For the input data:

```
4
2 3 4 1
1 2 3 4
3
1 8
2 7
3 6
```

the correct result is:

```
2
2
1
1
```

Solution

Translation in progress...

J

The Cave

★★★

Task author: Tomasz Idziaszek
Solution description: Jakub Łącki
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2014/jas`

A group of speleologists plans to explore a recently discovered cave. The cave consists of n chambers numbered from 1 to n. The chambers are connected by $n-1$ corridors in such a way that any chamber can be reached from any other. Each corridor connects exactly two chambers.

The cave will be explored by a group of m speleologists; for simplicity we number them by integers from 1 to m. Each speleologist presented some requirements concerning the area of the cave which he or she would like to explore. Speleologist i would like to begin the exploration in chamber a_i, finish exploring in chamber b_i and traverse a maximum of d_i corridors on the way (passing the same corridor each time is to be counted separately). Byteasar, head of the expedition, would like all the researchers to be able to meet at a certain point in time in order to exchange their observations. For this reason, he is wondering whether he could choose one of the chambers of the cave, and plan the routes of all the speleologists in such a way, that they all pass through the selected chamber. Of course planned routes must meet the requirements described initially by the researchers.

Input

The first line of the input contains one integer t ($1 \leqslant t \leqslant 1000$) that specifies the number of test cases. It is followed by the descriptions of the individual cases. Description of a single case begins with a line including two integers n and m ($2 \leqslant n, m \leqslant 300\,000$), which describe the number of chambers in the cave and the number of speleologists, respectively. The next $n-1$ lines describe the cave corridors. Each of them contains two integers u_i, w_i ($1 \leqslant u_i, w_i \leqslant n$), which indicate that chambers u_i and w_i are connected by a direct corridor.

Next m lines describe the speleologists' requirements. The i-th of these lines contains three integers a_i, b_i, d_i ($1 \leqslant a_i, b_i \leqslant n$, $1 \leqslant d_i \leqslant 600\,000$). They indicate that the speleologist i will begin by starting to explore chamber a_i, finish in chamber b_i, and moving between chambers passes through at most d_i corridors. You may always assume that it is possible to move from chamber a_i to b_i traversing not more than d_i corridors. Both the sum of values n for all the test cases, as well as the total value of m does not exceed 300 000.

Output

Your program should output exactly t lines. The i-th line should contain the answer to the i-th test case from the input. In case it is possible to set speleologists' routes is such a manner, so that they all run through one common chamber, one word `TAK` (i.e., *yes* in Polish) should be produced, followed by the number of the chamber where the meeting is to take place. Otherwise, your program should output only one word NIE (*no* in Polish). If there are multiple correct answers, your program should output any of them.

Example

For the input data:

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

the correct result is:

```
TAK 2
NIE
```

Solution

Translation in progress...

K The Captain ★★

Task author: Jakub Łącki
Solution description: Jakub Łącki
Available memory: 256 MB
`https://oi.edu.pl/en/archive/amppz/2014/kap`

Captain Byteasar sails the waters of Byteic Sea together with his irreplaceable First Officer Bytec. There are n islands in the Byteic Sea, which we numbered from 1 to n. Captain's ship has docked at the island number 1. Captain's expedition plan is to sail to the island number n.

During the voyage, the ship always moves in one of four directions of the world: north, south, east, or west. At any time it is either the Captain or the First Officer standing at the helm. Every time the ship will perform 90° turn, they would change at the helm.

Along its way, the vessel may stop at other islands. After each stop, the Captain can decide whether he takes control of the helm first, or not. In other words, for each route leg, leading from an island to another one, one of the sailors stands at the helm while the ship is travelling north or south, and the other controls it while it is moving east or west. In particular, if a given fragment of the route runs exactly in one of the four directions of the world, the ship is controlled by only one of the sailors.

The captain is now considering how to plan a route of the forthcoming voyage and how to divide labour in such a way to spend as little time at the helm. At the same time the Captain does not care how long the calculated route would be. It is assumed that the vessel is sailing at a constant rate of one unit per hour.

Input

The first line of the input contains a single integer n ($2 \leqslant n \leqslant 200\,000$) specifying the number of islands in the sea. For simplicity, we introduce a coordinate system onto Byteic Sea with axes parallel to the directions of the world. Every island is represented as a single point. Subsequent n lines contain descriptions of the islands: the i-th line contains two integers x_i, y_i ($0 \leqslant x_i, y_i \leqslant 1\,000\,000\,000$) specifying the coordinates of the i-th island in the sea. Each island bears different coordinates.

Output

Your program should output a single integer: the least number of hours the Captain will have to steer the ship on the route from the island number 1 to the island number n.

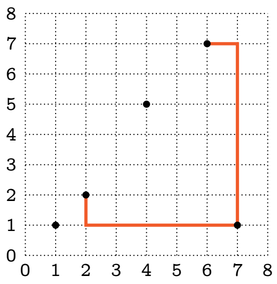
Example

For the input data:

```
5
2 2
1 1
4 5
7 1
6 7
```

the correct result is:

```
2
```



Explanation of the example: The Captain may designate a route that is indicated in the figure. During the voyage from island 1 (of coordinates $(2, 2)$) to island 4 (of coordinates $(7, 1)$) the Captain controls the ship only for an hour, while the ship is sailing south. During the second leg of the trip the Captain controls the vessel only when it is moving east.

Solution

Translation in progress. . .

Literature

L. Banachowski, K. Diks, W. Rytter, *Algorytmy i struktury danych.* WN PWN, Warsaw, 2019

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, *Introduction to Algorithms.* The MIT Press, 2009

R.L. Graham, D.E. Knuth, O. Patashnik, *Concrete Mathematics.* Addison-Wesley Professional, 1994

P. Stańczyk, *Algorytmika praktyczna. Nie tylko dla mistrzów.* WN PWN, Warsaw, 2009

Looking for a Challenge? The Ultimate Problem Set from the University of Warsaw Programming Competitions. K. Diks, T. Idziaszek, J. Łącki, J. Radoszewski (red.). WN PWN, Warsaw, 2018

Przygody Bajtazara. 25 lat Olimpiady Informatycznej – wybór zadań. K. Diks, T. Idziaszek, J. Łącki, J. Radoszewski (red.). WN PWN, Warsaw, 2018

Netography

`deltami.edu.pl` — web page of the popular science monthly *Delta*

`oi.edu.pl` — “blue books” from the Polish Olympiad in Informatics

`was.zaa.mimuw.edu.pl` — practical algorithms video courses

The descriptions of problems *Arithmetic Rectangle*, *Speed Cameras* and *Pillars* were based on articles in the popular science monthly *Delta* (from issues 3/2012, 1/2015 and 3/2015, respectively). The descriptions of problems *Ants* and *Afternoon Tea* were taken from the book *Looking for a Challenge?*

Contents