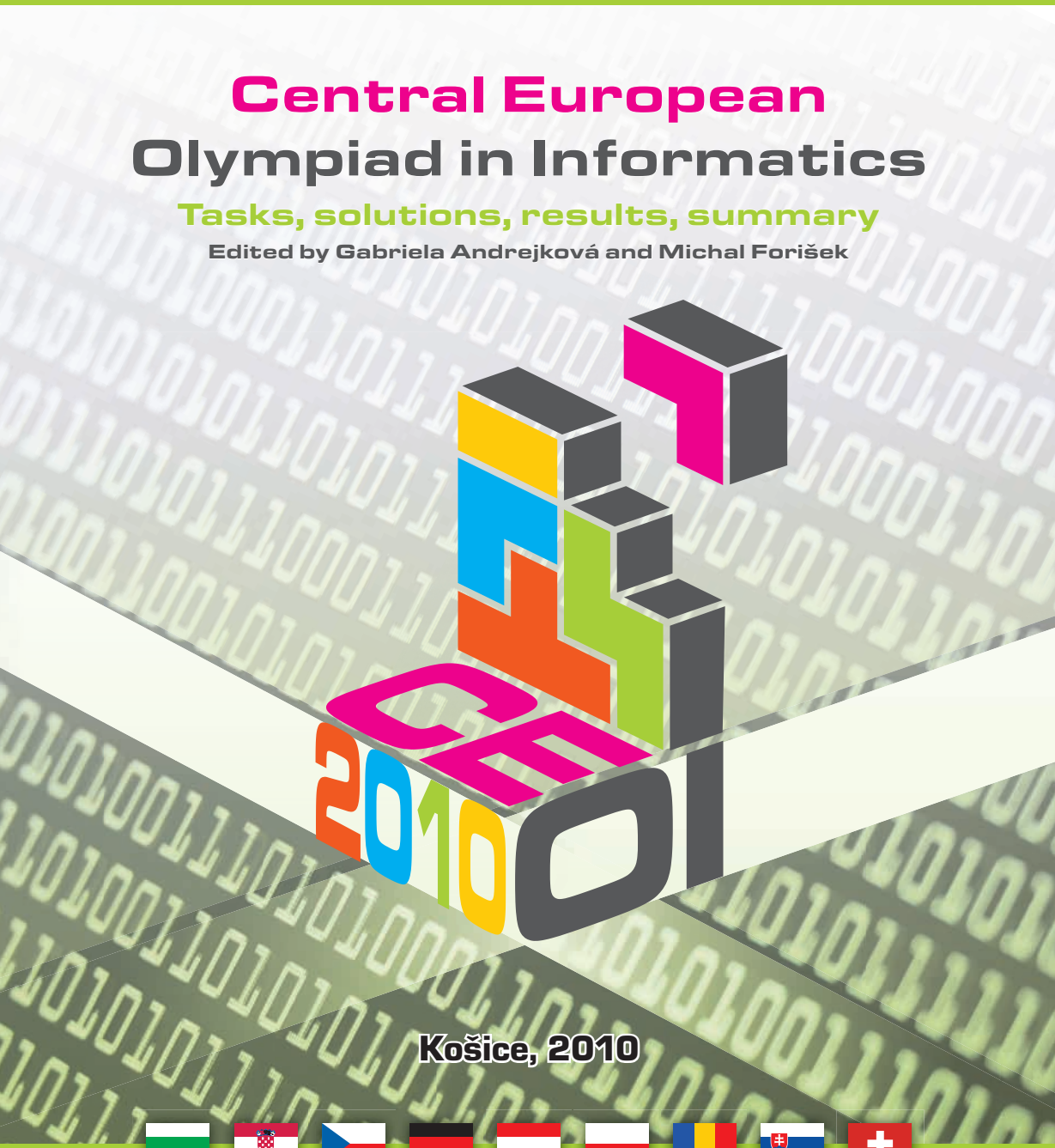
Ministry of Education, Science, Research and Sport of the Slovak Republic
Faculty of Science, P. J. Šafárik University in Košice, Slovakia
Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava, Slovakia
Slovak Society for Computer Science

Central European Olympiad in Informatics

Tasks, solutions, results, summary

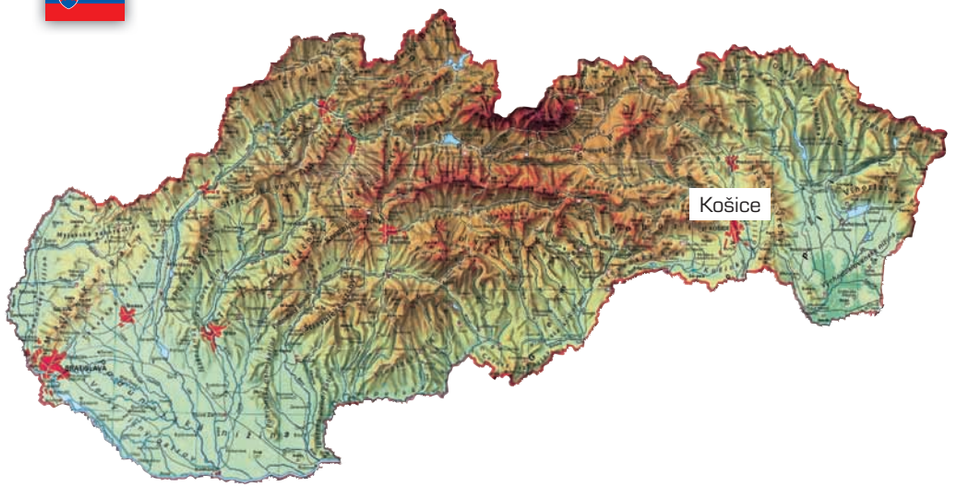
Edited by Gabriela Andrejková and Michal Forišek



Košice, 2010

http://ceoi2010.ics.upjs.sk

Slovakia



Authors:
Gabriela Andrejková, Michal Forišek

Central European Olympiad in Informatics - CEOI 2010

Tasks, solutions, results, summary

Publisher: Equilibria, s. r. o. Košice, www.equilibria.sk
Printing and graphic design: Equilibria, s. r. o., Košice, Poštová 13, 040 01 Košice
Edition: first, **Year:** 2010, **Pieces:** 120, **Number of Pages:** 36

ISBN 978-80-89284-65-8

Central European Olympiad in Informatics

Tasks, solutions, results, summary

July 12th - 19th, 2010, Košice, Slovakia

http://ceoi2010.ics.upjs.sk

Contents

Preface

Dear readers, dear friends,

17th annual of Central European Olympiad in Informatics (CEOI 2010) is over. Actually now everybody has gone home and we can only hope that those of you who are now reading this preface are also friends – if not personal friends of the contestants then at least of the idea of young people meeting to test their programming and problem solving skills and becoming friends in the process. In spite of everything we are glad to have participated in such an event and we would like to publish this summary volume to preserve and communicate some of the results – not only problems and their solutions and material ones like lists of contestants and awarded points, but also something of the atmosphere as well.

So what will you find in the volume?

The address of Prof. Branislav Rovan is devoted to sources and aims of Informatics. The regulations of CEOI contests are rarely published but might be of interest and were therefore included in the volume (General Assembly of CEOI prepared some modifications). Those interested in the actual problems and their solutions will find these in the next part of the volume. We hope that this booklet will help many contestants to prepare for their future programming contests. The list of participating teams and their individual members together with the table of results make up the "official" part of the volume; in addition, a detailed programme of the whole event should help the readers get some of the feeling for the atmosphere. In the last years, on-line contests on the same problems are made as parallel contests for contestants all over the world. We publish the results of the on-line contest for some comparison of achieved points.

Well, this seems like a good place to finish this introduction; thank you for reading it and perhaps some of us will meet at the coming CEOI's: next year in Poland, or the year after that in Hungary, or...

And at last but not least we would like to thank all organizers - Ministry of Education, Science, Research and Sport of the Slovak Republic, Faculty of Science, P. J. Šafárik University in Košice, Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava, Slovak Society for Computer Science, and all people behind them – for the great support of the event.

G. Andrejková, M. Forišek

Informatics – The Landscape

It is a rare and pleasant occasion to address on behalf of the Slovak Society for Computer Science a group of young people whose perception of the notion of informatics exceeds the common misconception. Much too often the ability to use some word-processing software, to send e-mails and to browse the web makes people believe they are informatics experts. You can appreciate the complexity of problems and the difficulty of finding efficient algorithms. All of you have proved to belong to the top few within your countries. Still, I hope, you do realize you have a long way to go to really understand informatics and to become true experts.

Informatics is most frequently defined to be the science about storing, organizing, manipulating and communicating information. What are the main ingredients of the information processing task?

First, one has to "create" information. This involves representing, encoding, ..., real world objects into some machine readable form, into a form that can be stored and manipulated in a computer. The importance of this step is not always appreciated. And it can make a difference. A frequent example cited is the representation of natural numbers. We could choose roman numerals or the decimal notation. It is clear which choice will let us multiply numbers easier.

Second, find efficient ways to manipulate information. This involves design of algorithms, data structures, ..., but also ways to present (visualize) information.

Third, make sure that what was done is what was intended to be done. One has to specify what a (software) system should do and prove it does, best in conjunction with the second step. Many formal and semiformal specification languages were designed and associated logics and proof systems studied.

Fourth, one has to really make it work. This means implementation and involves many of the engineering aspects of computer science. It involves writing up programs (transferring the above obtained design into machine executable code) and of course it needs hardware, machinery that executes programs.

Most of you have some understanding of efficiency in information processing. To fully understand the challenges in this and the other three areas you will of course need further study at the university. Rest assured there are enough hard problems to be solved to keep you busy for the rest of your life. And this is still not the whole story. One can clearly see shifting of the focus in informatics. In the early years it was the dominance of hardware and IBM was the IT company of the world. We are passing through the period of software dominance with Microsoft dominating the markets. We are clearly moving to a period where information is becoming dominant and you can witness growing importance of "information providers" like Google. Informatics is not well prepared for this period. The classical notion of information introduced by Shannon does not capture all aspects of information that are becoming relevant. Measuring the amount of information was useful when efficient transmission and storage of information was the main concern. Redefining the notion of information to capture its usefulness, relevance, security aspects, etc. is the new challenge for informatics. Let me express my hope that talented young people like you will help in advancing the understanding of information and information processing.

Address by Prof. RNDr. Branislav Rovan, PhD.

Regulations of the Central European Olympiad in Informatics

The following document contains the Regulations of the CEOI, as approved at the final IC meeting of CEOI 2010.

Introduction

The Olympiad is organized by the Ministry of Education or another appropriate institution of one of the eight Central European countries. According to the rules accepted by the initiators of the CEOI, teams of eight Central European country, i.e. Croatia, Czech Republic, Germany, Hungary, Poland, Romania, Slovak Republic and Slovenia (suspended), are invited as regular participants. Moreover, the host country may invite guest countries and a second team from the host country, sharing the same team leader. The International Committee (IC) of the CEOI consists of the eight team leaders, and a representative of the host country, who chairs the meetings of the IC. A host which is willing to organize a CEOI in a given year in its country, has to announce its intent at least one year before that CEOI (during the previous CEOI competition days). Selection of the next host is made by the IC by a majority vote. Revision of the Regulations of the CEOI is adopted by the IC by a 2/3 majority vote. Enlarging or decreasing the set of CEOI countries can only be adopted by consensus.

Goals

The CEOI aims at motivating secondary school students of Central Europe to: get more interested in informatics and information technology in general, test and prove their competence in solving problems with the help of computers, exchange knowledge and experience with other students of similar interest and qualification, establish personal contacts with young people of the Central European region. Additionally, the CEOI may: provide training for the students participating in the International Olympiad in Informatics (IOI), initiate discussion and cooperation in informatics education in the secondary schools of the Central European countries.

General Regulations

Each team is composed of up to four secondary school students, team leader and deputy team leader. Only the cost of travel to and from the place of the competition should be paid by teams; all local expenses are covered by the organizers. Accompanying persons and observers are welcome, but they should pay for their stay. Interested people are advised to contact the local organizers. The official language is English. Students may use their native languages. Programming problems will be formulated in English and then translated by the team leaders to the native language of their team. Both versions will be given to the students. Team leaders must be able to speak and write in English, as well as the language of their team. The computers will be IBM PC compatibles with selected software packages. Only the computers and software with built-in help facilities provided by the organizers may be used in the competition. In particular, the use of printed materials and electronic devices brought by the contestants will be forbidden. The programming languages of the contest are Pascal, C and C++; the precise versions of these languages will be updated each year. The compilers and programming environments for the above mentioned programming languages will be installed on the hard disk.

Team Composition

Students (contestants) have to be in school during the year when the contest is held and at most 19 years old. The team leader will be a member of the General Assembly. Observers and persons accompanying a delegation have to pay a fee.

General Assembly

General Assembly (GA) is composed of the team leaders of the participating countries and the president nominated by the host country. General Assembly selects problems to be solved in the competition from a set of problems prepared and proposed by the Scientific Committee. The selection procedure is the following: The chairperson of the Scientific Committee distributes the proposals for competition tasks and presents short descriptions of proposed model solutions. Their number equals the number of problems to be solved by the contestants. The GA members may either accept or, in case of a major ambiguity of formulation or other serious reasons, deny the proposals by voting. When and if a proposal is denied, another prepared proposal will be offered to the GA. For such cases, the Scientific Committee should prepare at least two extra proposals for each round. The text of the accepted proposals must not be changed by the GA, except for minor rephrasing that is needed to avoid smaller ambiguities. The selected problems will be translated by the team leaders into the native languages of the teams.

Scientific Committee

The Scientific Committee (SC) consists of a chairperson and a number of experts (SC members) from the host country. It becomes active well before the beginning of the Olympiad and has the task of selecting and preparing problem proposals. A further task of the Scientific Committee is to test and evaluate the solutions of the contestants.

Problems, Competition

The competition consists of two rounds in two days. In both rounds the working time is five hours and the contestants will be given three problems to solve. The selected problems will be translated by the team leaders into the native languages of the teams. Contestants may submit written questions to the Scientific Committee concerning the formulation and interpretation of the problems during the initial period of each competition round. Contestants may use their native language when asking questions; further details will be regulated by each year competition rules. The host country publishes competition rules that include the description of the competition environment and details on the course of competition at least one before CEOI starts. No special hardware requirement or software packages (e.g. graphic packages) will be needed to solve the problems. Generally, the contents and the form of the CEOI tasks is guided by the IOI syllabus. The whole communication between the CEOI authorities and contestants will be in a written form.

Evaluation

When the working time is over, the solutions of each of the contestant will be checked by an evaluator. The evaluation is based on the test data and the responses of the programs only. The evaluation procedure concludes with the meeting of the Scientific Committee, where the evaluation reports are discussed. Potential disagreements are dissolved by voting. Achieving a proper and balanced evaluation is the responsibility of the Scientific Committee. If a team leader does not accept the results of the evaluation, he/she may appeal to the General Assembly. Finally, the president of SC or IC presents the anonymous results to the General Assembly to take final decisions.

Results and Prizes

The General Assembly will determine the minimum scores for the gold, silver and bronze medals. The proportion of these gold, silver and bronze medals should be approximately 1:2:3. About 50% of the contestants should receive medals. Each contestant will receive a certificate of participation. The medals, certificates and other prizes will be given to the contestants at the official closing ceremony.

Participants

BULGARIA	
Team Leader:	Emil Kelevedjiev
Deputy Leader:	Pavlin Peev
Contestants:	Anton Anastasov, Vladislav Haralampiev, Rumen Hristov, Mihail Kovachev

CROATIA	
Team Leader:	Ivo Šeparović
Deputy Leader:	Luka Kalinovčić
Contestants:	Stjepan Glavina, Ivan Katanić, Ivica Kičić, Adrian Satja Kurdija

CZECH REPUBLIC	
Team Leader:	Daniel Kráľ
Deputy Leader:	Pavel Töpfer
Contestants:	Lukáš Folwarczný, Filip Hlásek, Michal Mojzík, Štěpán Šimsa

GERMANY	
Team Leader:	Wolfgang Pohl
Contestants:	Simon Bürger, Patrick Klitzke, Aaron Montag, Klaas-Hendrik Poelstra

HUNGARY	
Team Leader:	Gyula Horváth
Deputy Leader:	László Zsakó
Contestants:	Patrik Adrián, Richárd Palincza, Zoltán Szenczi, Ágoston Weisz

POLAND	
Team Leader:	Jakub Radoszewski
Deputy Leader:	Tomasz Kulczyński
Contestants:	Igor Adamski, Adrian Jaskółka, Jan Milczek, Anna Piekarska

ROMANIA	
Team Leader:	Zoltan Szabo
Deputy Leader:	Andrei-Paul Puni
Contestants:	Vlad-Alexandru Gavrilă, Victor-Cristian Ionescu, Alexandru Tache, Bogdan-Cristian Tătăroiu

SLOVAKIA 1	
Team Leader:	Andrej Blaho
Deputy Leader:	Tomáš Vinař
Contestants:	Matej Balog, Tomáš Belan, Ján Hozza, Filip Sládek

SLOVAKIA 2	
Team Leader:	Andrej Blaho
Deputy Leader:	Tomáš Vinař
Contestants:	Michal Anderle, Martin Pitoňák, Ján Sebechlebský, Marek Špano

SWITZERLAND	
Team Leader:	Sandro Feuz
Deputy Leader:	Adrian Roos
Contestants:	Thomas Leu, Lazar Todorović, Josef Ziegler, Nikola Džokič

CEOI 2010 was organized

under the auspices of

Ministry of Education, Science, Research and Sport of the Slovak Republic

Organizers

Faculty of Science, P.J. Šafárik University in Košice, Slovakia
Slovak Society for Computer Science
Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava, Slovakia

Steering Committee

prof. RNDr. Pavol Sovák, CSc. (chair)
prof. RNDr. Andrej Bobák, DrSc.
doc. RNDr. Dana Pardubská, PhD.
prof. RNDr. Branislav Rovan, PhD.

Scientific Committee

RNDr. Michal Forišek, PhD. (chair)
RNDr. Ján Katrenič
Mgr. Monika Steinová
Mgr. Julka Šišková
Mgr. Lukáš Poláček
Mgr. Michal Nánási
Mgr. Marek Zeman

Organizing Committee

doc. RNDr. Gabriela Andrejková, CSc. (chair)
doc. RNDr. Gabriel Semanišin, PhD.
RNDr. František Galčík, PhD.
RNDr. Rastislav Krivoš-Belluš
doc. RNDr. Božena Mihalíková, CSc.
PhDr. Svetlana Libová
Doc. RNDr. Dušan Šveda, CSc.
Jana Boháčová
Mgr. Peter Mlynárčik
Mgr. Ladislav Mikeš
Veronika Vaneková

Technical Committee

Mgr. Martin Rejda (chair)
Ing. Marián Andrejko
Eduard Dvorný
Mgr. Ľudovít Hvizdoš
Michal Petrucha
Marek Bundala
Peter Fulla

Guides

Eva Maďarošová (Hungary)
Zuzana Bedécsová (Poland)
Mária Dolinská (Bulgaria)
Ivana Ďurčová (Germany)
Mária Harčarufková (Slovakia 1, 2)
Natália Iriasová (Croatia)
Matúš Jaraba (Switzerland)
Ján Jerguš (Romania)
Mária Piatnicová (Czech republic)

Reporters

Maroš Andrejko
Gabriela Hucovičová
Pavol Rajzák
Samuel Kupka

TASKS OF THE CEOI 2010

The Alliances

Author: **Marek Zeman**, FMFI UK Bratislava
Problem preparation: **Marek Zeman**, **Michal Forišek**, FMFI UK Bratislava

In a fantasy world, there is a large island of a rectangular shape. The sides of the island happen to be exactly R miles and C miles long, and the whole island is divided into a grid of areas. Some of the areas are uninhabited, and the rest are occupied by villages of fantasy beings: elves, humans, dwarves, and hobbits. Each area contains at most one village. Two villages are considered neighbours if their areas share a side.
Recently, the villages became terrified of the Great Evil. In order to feel safer, each village has decided to form military alliances with some of its neighbours. An alliance always involves two neighbouring villages, and it is a mutual and symmetric agreement.
Depending on the species living in the village, the inhabitants will not feel safe unless a specific configuration of alliances is formed:

- The elves feel confident, and therefore need exactly one alliance.
- Human villages require alliances with exactly two neighbours. Moreover, leaving two opposite sides of the village exposed is bad for tactical reasons. Therefore the two allied neighbours cannot be located on opposite sides of the village.
- Dwarves require alliances with exactly three neighbours.
- Hobbits are extremely scared, and therefore need to have alliances with all four of their neighbours.

In other words, except for humans each village requires a particular number of alliances, but does not care which neighbours will be its allies. For humans there is an additional restriction: the allied neighbours must not be on opposite sides of the village.
The conditions must be fulfilled irrespective of the position of the village on the map. For example, a dwarf village desires three alliances. If it is located on the coast, this means that it must have alliances with all three neighbours. If there is a dwarf village in a corner of the island, its inhabitants will never feel safe.

Task specification

For a given island description, your task is to decide whether it is possible to form alliances so that all inhabitants of the island will feel safe. In case of a positive answer, your task is also to suggest one viable configuration of alliances. In case of multiple solutions, choose an arbitrary one.

Input specification

The first line of the input contains two integers R and C specifying the size of the island. The following R lines encode a description of the island. Each line consists of C space-separated numbers between 0 and 4:
- 0 means uninhabited area,
- 1 means an elf village,
- 2 means a human village,

- 3 means a dwarf village.
- 4 means a hobbit village,

(Note that the number in the input always corresponds to the number of desired alliances for that village.)

Constraints

In all test cases assume that $1 \le R, C \le 70$.
In test cases worth a total of 55 points we have $min(R, C) \le 10$. Out of these, in test cases worth 15 points $R \cdot C \le 20$.

Another batch of test cases worth 10 points contains maps with only uninhabited areas and human villages.
(This batch is not included in the test cases worth 55 points.)

Output specification

If there is no solution, output a single line with the string "Impossible!" (quotes for clarity). Otherwise, output one valid map of alliances in the following format.

Each area should appear in the output as a matrix of characters. If the area is uninhabited, the corresponding section of the output will be filled with . (dot) symbols. If the area has a village there should be a symbol O (uppercase letter O) in the middle representing the village itself, and there should be symbols X (uppercase letter X) representing a configuration of its allies. The rest of the matrix should be filled with . (dot) symbols.

For each village type, all possible layouts of alliances are shown in the image below.

Examples

input:

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

output:

```
. . . . . . . . . . . .
. O X X O X X O . . . .
. X . . X . . X . . . .
. X . . X . . X . . . .
. O X X O X X O . . . .
. X . . X . . X . . . .
. X . . X . . X . . . .
. O X X O X X O X X O .
. . . . . . . . . . . .
```

input:

```
1 2
2 1
```

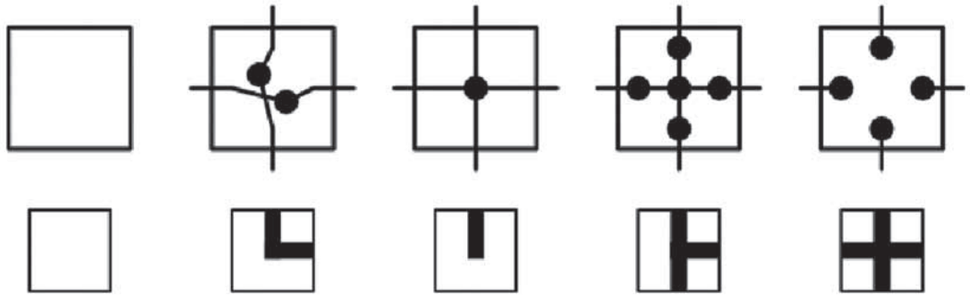
output:

Impossible!

Solution

For some easy points to start with, note that a human village is uniquely determined if we know whether it has an alliance with its top and its left neighbour. Using this observation, inputs that contain human villages only can be solved simply: just iterate row by row, each cell will be uniquely determined at the time when you visit it. (Of course, in the optimal solution it is not necessary to implement this separately.)
Inputs with a small number of columns can be solved using dynamic programming or recursion with memoization: Process the cells row by row, each row left to right. At any moment we will ask the same question: it is possible to create alliances in the remaining cells in such a way that everything matches? Note that the answer does not depend on how the alliances look like in the already processed part of the map. It only depends on the following variables:
- what is the next cell to be processed,
- for each column, is there an alliance (edge) going down from the last processed cell?

This gives us $O(RC2^C)$ states. Each time we find out the answer for any of the state, we store it in order to avoid computing the same thing again in the future. The time complexity of this solution is exponential in the length of the shorter edge of the map, and polynomial in the length of the longer edge.
The optimal solution solves the task using maximum matching in a bipartite graph. We will replace each village by a cluster of nodes as shown below (in order: empty cell, humans, elves, dwarves, hobbits):



Note that a human village is replaced by two separate nodes: one connected to the top and to the bottom, the other to the left and to the right. Edges which cross cell boundary represent potential alliances. It is possible to design the alliances if and only if the resulting graph has a perfect matching, i.e., a matching that covers all vertices. This should be obvious for all village types except for dwarves. For a dwarf village, note that we must match the central node to one of its four neighbours. From each of the remaining three neighbours of the central node the matching edge will then lead across the cell boundary. Also note that the graph is bipartite: to obtain one possible partition, colour the cells in a chessboard fashion, and colour all nodes by their cell colour, except for central nodes for dwarf villages, which get the opposite colour.

An Arithmetic Rectangle

Author: **Michal Forišek**, FMFI UK Bratislava
Problem preparation: **Monika Steinová**, **Michal Forišek**, **Juliana Šišková**, FMFI UK Bratislava

After the lesson on arithmetic progressions, the teacher gave Peter homework: a sheet of paper with numbers written in some cells of a grid. Some of the cells were empty. Peter's task is to create an arithmetic rectangle by filling in the missing numbers. In an arithmetic rectangle, all the numbers in each row and in each column have to form an arithmetic progression in the order in which they appear.

For example, this is an arithmetic rectangle of type 3 x 5:

```
1  3  5  7  9
2  2  2  2  2
3  1 -1 -3 -5
```

Peter is lazy to do such tasks manually. He wants you to write a program that will do it for him.

Task specification

You are given a grid of **integers**, some of them substituted by dots. Find out whether it is possible to replace the dots by some **rational** numbers in order to obtain an arithmetic rectangle. If there is a solution, find one.

Note: An arithmetic progression is a sequence of numbers such that the difference of any two successive elements of the sequence is a constant.

Input specification

The first line of the input contains two positive integers R and C : the dimensions of the grid. R lines follow, each of them containing C tokens separated by single spaces. Each of the tokens is either a dot (.), or an integer.

Constraints

Each number given in the grid is between -100 and 100, inclusive. There are 10 batches of test cases, worth 10 points each. In batches 1 through 9 we have . The properties of the individual batches are given below.

- **Batch 1.** All numbers are already filled in.
- **Batch 2.** Either $R = 1$ or $C = 1$ in each test case.
- **Batch 3.** $R = C = 2$ in each test case.
- **Batch 4.** Each test case has a unique solution that can be found using the approach suggested in the first example.
- **Batch 5.** Each test case has a unique solution, and the solution contains integers only.
- **Batch 6.** Each test case has a unique solution.
- **Batch 7.** Each test case either has a unique solution that contains integers only, or has no solution at all.
- **Batch 8.** Each test case either has a unique solution, or has no solution at all.
- **Batch 9.** Arbitrary test cases.
- **Batch 10.** Arbitrary test cases with $1 \le R, C \le 50$.

Output specification

If there is no solution, output a single line with the string "No solution." (quotes for clarity). If there are multiple solutions, pick and output any single solution.

When outputting a solution, output R rows, each with C space-separated rational numbers.

Each rational number shall be printed as "N/D", where D is positive and N and D are relatively prime. If D is 1, omit the "/D" part.

No number in your output may have more than 20 digits. (It should be easy to satisfy this restriction, its only intent is to simplify checking your outputs.)

Examples

input:

```
3 5
. . 3 . 5
. . . 5 .
. . . . 7
```

output:

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

The above example can be solved as follows: First, note that the second number in the last column must be 6. Then fill in row 1, row 2, and finally fill in columns 1 through 4.

input:

```
1 6
4 . . 0 . .
```

output:

```
4 8/3 4/3 0 -4/3 -8/3
```

input:

```
1 4
1 2 . 2
```

output:

No solution.

input:

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

output:

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

This is one of many possible solutions for this input.

Solution

Several of the test batches were simple to solve even if one has no idea about how to solve the task in general. For example, in the first batch the entire grid was already filled in, and all we had to do was to check whether it is filled correctly. This can be done using two nested cycles. In the third batch each input had two rows and two columns only. In this case, any rectangle of numbers is an arithmetic rectangle, because every sequence with only two elements is arithmetic. Hence it was sufficient to replace all dots by arbitrary values, e.g., zeroes.

In the fourth batch we were guaranteed that a greedy approach works: whenever you have a row or a column with two known values, use them to compute all other values of the unique arithmetic sequence that matches the two known values. It is easily shown that it is enough to iterate twice over all rows and columns: if it is possible to solve the instance using the greedy approach, we will first fill in at least one line (a row or a column), then at least two orthogonal lines (two columns or two rows), and finally all lines parallel to the first one (all remaining rows or columns). Some additional points could be obtained by guessing: if the greedy algorithm stopped and we still don't know all cells, guess that the first unknown one contains a zero.

To solve the general case, we need to start thinking in a more systematic way. Intuitively, it seems obvious that we cannot have too many values given before all other cells are determined uniquely. For example, we may note that once we fix the four values in the corners, the rest of the grid is determined uniquely. But how can we to formalize this intuition?

There are many possibilities that only differ in details: some produce slightly uglier equations than the others. We will pick one of the nicer possibilities. Let A be the value in the upper left corner of the grid, let B and C be its horizontal and vertical neighbours, and let D be the other common neighbour of B and C. It is easy to see that the entire grid is determined by these four values. For example, the first row contains the values A, B=A+(B-A), A+2*(B-A), A+3*(B-A), ...

For an empty grid, any combination of values for A, B, C, and D would give us a valid solution. If the given grid is not empty, each of the given values gives us a constraint on A, B, C, D. It can easily be seen that this constraint can be expressed as a linear equation in the variables A, B, C, D. Hence we reduced the general task to the following standard problem: we are given a system of linear equations in four variables, we need to check whether it is solvable, and find one solution if it is. This can be done using the Gaussian elimination process: we manipulate the equations in such a way that we obtain an equivalent system in which all coefficients below the main diagonal are zero.

Note that the number of variables is very low, so it is possible to solve the system in integer variables without overflow, and only then to evaluate the cells as fractions. If we have two equations E_1 and E_2 such that the variable A has coefficient c_1 in E_1 and c_2 in E_2, we can replace the second equation by the equation $c_2*E_1-c_1*E_2$. In this equation A has coefficient zero. Note that if $\gcd(c_1,c_2)>1$, we can then reduce the coefficients in this new equation, dividing them all by $\gcd(c_1,c_2)$. This is necessary to prevent overflow. In this way we get rid of most of the fractional arithmetic.

Another way how to solve all cases (possibly except for the largest one) without fractions: one can realize that as the input contains integers only, all rational numbers in the answer must necessarily have small denominators. It is then possible to solve the system using floating point variables, and only in the end to convert the floating point numbers into matching fractions.

Bodyguards

Author: **František Simančík**,
Problem preparation: **Daniel Bundala**, FMFI UK Bratislava, **Ján Katrenič**, PF UPJŠ Košice

Have you ever met the members of the ten European royal houses, the Argentinian football team including their coach Diego Maradona, or all of the Turing and Fields medal prize winners? We have invited many celebrities from all over the world to the CEOI 2010 closing ceremony. Unfortunately, very few of them responded to our invitation, and those who did, politely rejected. Nevertheless, do not forget to bring your camera to the ceremony, one never knows who might turn up!

As you can imagine, the security of the guests is of utmost importance. The problem is how to seat their bodyguards in the audience so that maximal security is guaranteed.

The auditorium contains many seats, arranged into a large grid. Based on the safety regulations a security expert has determined necessary bodyguard counts for each row and each column of the auditorium.

Task specification

You are given the required number of bodyguards for each row and column of the auditorium. This information is given in a compressed form as explained below. Determine whether it is possible to place the bodyguards in such a way that in each row and each column we would have exactly the required number of bodyguards.

Assume that the auditorium is initially empty, i.e., you may seat bodyguards wherever you wish. Each seat may only be occupied by a single bodyguard.

Input specification

The input begins with the description of the rows. The first line of the input contains one positive integer R : the number of groups of rows. R lines follow. Each of these lines contains 2 positive integers: the required number of bodyguards in each row of the group and the number of rows that form the group.

This is followed by the description of column groups. The next line contains one positive integer C : the number of groups of columns. C lines follow. Each of these lines contains 2 positive integers: the required number of bodyguards in each column of the group and the number of columns that form the group.

Constraints

You may assume that the total number of bodyguards required by row constrains is the same as the total number of bodyguards required by column constraints. You may assume that this total number of bodyguards is at most 10^{18}.

You may assume that all numbers are positive integers that do not exceed 1 000 000 000.

- You may assume that $1 \le R, C \le 200\,000$.

Several batches of test cases, worth a total of 50 points, satisfy the following criteria:
the total number of rows in the auditorium will be at most 2 000
the total number of columns in the auditorium will be at most 2 000
the total number of bodyguards will be at most 1 000 000.

In a test batch worth another 10 points we have $R, C \le 100$ in each test case.

Output specification

Output a single line with the number "1" if the constraints are satisfiable and the number "0" otherwise (quotes for clarity).

Example

input:

```
2
2 1
1 2
1
2 2
```

output:

```
1
```

There are two groups of rows: the first one has one row that must contain two bodyguards, the second one has two rows that must contain one bodyguard each. There is one group of columns: each of the two columns must contain two bodyguards. One possible placement of bodyguards:

```
X X
X .
. X
```

input:

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

output:

```
0
```

Two of the rows are required to be full of bodyguards. Hence there must be at least two bodyguards in each column. However, the last column must only contain one bodyguard, which is a contradiction.

Solution

The following simple greedy strategy works: pick the row with most bodyguards required. Place the bodyguards into columns that currently require the most bodyguards. To prove that this strategy works, assume that a valid seating exists and consider the current row. Mark the seats where we want to seat the bodyguards. If one of these seats (in column C_1) is empty, our row must contain a bodyguard B sitting in an unmarked seat (in column C_2). As C_1 contains at least as many bodyguards as C_2, there must be some other row such that in C_1 we have a bodyguard B' and in C_2 we don't. We can then move B' to column C_2 and B to column C_1. After finitely many repetitions of the above process we will get a valid seating in which our row contains bodyguards precisely in the required places.
This strategy, if implemented using an efficient priority queue (such as a heap) scored 50 points.
For the optimal solution we need a faster strategy. First of all, note that the relative order of rows and columns does not matter. So the first step is to sort the groups according to the number of bodyguards in each row/column, starting with the ones containing most bodyguards.
Consider the seating of the bodyguards in the auditorium in which there is the correct number of bodyguards in each row, and these occupy the first possible columns in each row. This seating is possibly invalid but it shows us a necessary condition for a solution to exist: For any K, sum the required number of bodyguards in the first K columns, and also sum the current number of bodyguards in those columns. Clearly, the current number of bodyguards is the largest possible. Hence if for some K the required number exceeds the current number, we know that no solution exists.
On the other hand, we claim that the above set of conditions is also sufficient for a solution to exist. To prove that, we will describe a process how to construct one valid solution, given that all conditions hold. Start with the possibly invalid seating defined above, then process the columns one by one. For each column: when you start processing it, it contains at least as many bodyguards as it should. If it contains more, pick the lowest rows (the ones containing fewest bodyguards) and send their bodyguards from the current column into the first available column on the right. It is easily verified that this change is always possible, and after this change the conditions still hold for all K.
The final step is to note that it is not necessary to actually check the conditions for all possible K, we just have to do this for the places where the required number of bodyguards changes.

The MP3 Player

Author: **Peter Fulla**, FMFI UK Bratislava
Problem preparation: **Peter Fulla, Daniel Bundala**, FMFI UK Bratislava

Georg's new MP3 player has many interesting features, one of them being the key lock. All the keys are locked after more than T seconds of inactivity. After the key lock is engaged, no key performs its original function, but if any key is pressed, the key lock is disengaged.

For example, assume that $T = 5$ and the player is currently locked. Georg presses the key A, waits for 3 seconds, presses the key B, waits for 5 seconds, presses C, waits for 6 seconds, and presses D. In this case only the keys B and C perform their regular functions. Note that the keys became locked between C and D was pressed.

Sound level of the MP3 player is controlled by the + and - keys, increasing and decreasing volume by 1 unit respectively. The sound level is an integer between 0 and V_{max}. Pressing the + key at volume V_{max} or pressing the - key at volume 0 leaves the volume unchanged.

Task specification

Georg does not know the value of T. He wanted to find it by an experiment. Starting with a locked keyboard, he pressed a sequence of N + and - keys. At the end of the experiment Georg read the final volume from the player's display. Unfortunately, he forgot to note the volume before his first key press. For the purpose of this task, the unknown initial volume will be denoted V_1 and the known final volume will be denoted V_2.

You are given the value V_2 and a list of keystrokes in the order in which Georg made them. For each key, you are given the type of the key (+ or -) and the number of seconds from the beginning of the experiment to the moment when the key was pressed. The task is to find the largest possible **integer** value of T which is consistent with the outcome of the experiment.

Input specification

The first line of the input contains three space-separated integers N, V_{max} and V_2 ($0 \le V_2 \le V_{max}$). Each of the next N lines contains a description of one key in the sequence: a character + or -, a space and an integer C_i ($0 \le C_i \le 2 \cdot 10^9$), the number of seconds from the beginning of the experiment. You may assume that the key presses are in sorted order and that all times are distinct (i.e., $C_i < C_i + 1$ for all $1 \le i < N$).

Constraints

You may assume that $2 \le N \le 100\,000$ and $2 \le V_{max} \le 5000$.
In test cases worth 40 points $N \le 4000$.
In test cases worth 70 points $N \cdot V_{max} \le 400\,000$.

Output specification

If T can be arbitrarily large, output a single line containing the word *"infinity"* (quotes for clarity).

Otherwise, output a single line containing two integers T and V_1 separated by a single space.

The values must be such that carrying out the experiment with locking time T starting at volume V_1 gives the final volume V_2. If there are multiple possible answers, output the one with the largest T; if there are still multiple possible answers, output the one with the largest V_1. *(Note that at least one solution always exists: for $T = 0$ none of the keys performs its action, so it suffices to take $V_1 = V_2$.)*

Examples

input:

```
6 4 3
- 0
+ 8
+ 9
+ 13
- 19
- 24
```

output:

```
5 4
```

For $T = 5$ the keys perform the following actions: unlock, unlock, +, +, unlock, -.
For any we would get $V_2 = 3$. Note that the output contains the largest possible V_1.
For the last two keystrokes will both be active, hence it will be impossible to have $V_2 = 3$.

input:

```
3 10 10
+ 1
+ 2
- 47
```

output:

infinity

If $V_1 = 10$ then for any T we'll have $V_2 = 10$.

Solution

To start attacking this task, there are many slow correct solutions which are reasonably easy to implement. The simplest possible strategy is to try all possible values of T and V_1. Once we fix T and V_1, we can simply simulate the process and check whether the final volume is V_2 or not.

The first possible improvement: we do not have to consider all values of T, only those where something changes – i.e., such that for delay T+1 some new keystroke will become active. There are at most N-1 such values of T.

Another place where the trivial solution can be improved is the value V_1. Note that for a fixed T the function f_T that returns V_2 for a given V_1 is non-decreasing: if we start the experiment at a higher volume, we will end it at least at the same volume as before. Additionally, if $f_T(v+1) > f_T(v)$, then clearly $f_T(v+1) = f_T(v)+1$. This makes our situation much easier. For a given T, if we want to check whether V_2 can be obtained, we can simply check whether $f_T(0) <= V_2 <= f_T(V_{max})$. Once we know the optimal T, we can find the largest valid V_1 using binary search.

There are several solutions with slightly different time complexities that score 100 points. We will now describe one of them. For this optimal solution we need to take a closer look at f_T. Note that it always has the following form (for some constants A,B,C): for V_1 from 0 to A-1 the function is constant and equal to C, for V_1 from A to B-1 it increases by 1, and then from B to V_{max} it is constant again. Additionally, this is not only true for the entire sequence: For any segment of keystrokes and any T the corresponding function can be described in this form. Also note that if we have two segments of keystrokes for which we know their functions and concatenate them, the new function for the longer segment can be computed in O(1).

We will now start at T=0 and keep on increasing T until we try all interesting values of T. We will use an interval tree to represent the current function for various segments of the input. Each time we increase T to the new interesting value, some keys will become active. For each of them, update the interval tree: change the function of the corresponding 1-key segment and recompute all O(log N) segments that contain it. After each such change, the root of the interval tree contains the function for the entire sequence of keystrokes, hence we can easily verify whether V_2 can be obtained for the current T.

PIN

Author: **Lukáš Poláček**,
Problem preparation: **Lukáš Poláček**, **Daniel Bundala, Michal Forišek**, **Ján Katrenič**

Martin has just been hired as a computer administrator in a big company. The company did not change its authorization system since 1980s. Every person has a four-digit personal identification number (PIN). Nobody uses usernames or passwords, you can login just by typing your PIN. As the company grew, they added the possibility to use letters as well, but the length of the PIN remained the same.

Martin is not happy with the situation. Suppose there are people whose PINs differ only at a single place, for example 61ab and 62ab. If the first person accidentally presses 2 instead of 1, the system would still let him in. Martin would like to make the statistics about the PINs currently in use, in particular, compute the number of pairs of PINs that differ at 1, 2, 3 or 4 positions. He hopes that these numbers will be alarming enough to convince his boss to invest in a better system.

Task specification

Given the list of PINs and an integer *D*, find the number of pairs of PINs that differ at exactly *D* positions.

Input specification

The first line of the input contains two space-separated positive integers *N* and *D*, where *N* is the number of PINs and D is the chosen number of differences. Each of the following *N* lines contains a single PIN.

Constraints

You may assume that in all test cases and .
Each PIN is of length 4 and each character is either a digit or a lowercase letter between 'a' and 'z', inclusive.
You may assume that all PINs in the input are different.
In test cases worth 15 points, .
In test cases worth 60 points, . Out of those, in test cases worth 30 points, $D = 1$.
In test cases worth 75 points, every PIN will only consist of digits or lowercase letters between 'a' and 'f',
inclusive. Thus it can be viewed as a hexadecimal number.

Output specification

Output a single line with a single number: the number of pairs of PINs that differ at **exactly** *D* positions.

Examples

input:

```
4 1
0 0 0 0
a 0 1 0
0 2 0 2
a 0 e 2
```

output:

```
0
```

For these PINs each pair of PINs differs at more than one position.

input:

```
4 1
0 0 0 0
a 0 1 0
0 2 0 2
a 0 e 2
```

output:

```
3
```

There are three pairs that differ at exactly 2 positions: (0000,a010), (0000,0202), and (a010,a0e2).

Solution

The first step towards solving this task is to write a simple program that will use brute force to compute the answer: it should compare all pairs of PINs and for each pair of PINs compute the number of differences. This program scores 15 points and additionally you can use it later to test your implementation of a more efficient solution.

For partial score you could decide to focus on the simpler cases: D=1 and possibly D=2. For D=1 there is a pretty simple solution: Start by allocating an array with 36^4 Booleans, each representing some 4-character string. Set those that correspond to the given PINs to true. Now, for each PIN generate all 35*4 strings that differ from it in a single character. Using the above array, we can easily check whether the new string is also a PIN. In this way, we will count each good pair of PINs twice (once in each direction).

The above approach can also be modified to handle D=2 in approximately the same number of steps. To do so, we will do the same as for D=1, and we will use an additional array of 36^4 integers, initialized to zeroes. Once again, for each PIN we will generate all strings that only differ from that PIN in a single character. For each such PIN we increment the counter at the appropriate position in the array. Once this phase is done, for each of the 36^4 possible strings we know the number of PINs that differ from this string in a single character. And this additional information is sufficient to compute the answer for D=2. How?

Consider a situation in which we generated the same string for two different PINs. We will call this situation a collision. For example, if the string 'ab3d' was generated 4 times, there were (4*3)/2=6 collisions for this string. For the given test case, let A be the answer for D=1 and let B be the answer for D=2. We already know A and we want to compute B. To do so, note that for each of the A pairs of PINs in distance 1 we get 36-2=34 different collisions, and for each of the B pairs of PINs in distance 2 we get 2 different collisions. Hence it is sufficient to compute the total number of collisions, the value B is then obtained by solving a simple linear equation.

Note that in the above solution we obtained the correct answer for D=2 by computing the answer for D=1 first, and then subtracting a factor based on this answer at an appropriate point in the computation for D=2. This observation can suggest the right tool that can be used to solve the remaining cases as well: the principle of inclusion and exclusion.

Instead of computing how many pairs of PINs differ on some places, we will compute the complementary information: how many pairs of PINs match? More precisely, for each of the 2^4 sets of places we will compute the number of pairs of PINs that match on those places (and either match or differ at all other places, we don't care about those). For example, one of the 16 numbers we will compute will be the number of pairs of PINs that match on the first place and the third place.

And this information is all we need to compute the answers for all distances. For example, the answer for D=4 (i.e., the number of pairs that differ in all places) can be computed as (the number of all pairs) – (for each place, the number of pairs that match on that place) + (for each pair of places, the number of pairs that match on those places) – (for each three places, the number of pairs that match on those places).

A Huge Tower

Author: **Michal Forišek**,
Problem preparation: **Monika Steinová**, **Michal Forišek**, **Michal Nánási**

The ancient Babylonians decided to build a huge tower. The tower consists of *N* cubic building blocks that are stacked one onto another. The Babylonians gathered many building blocks of various sizes from all over the country. From their last unsuccessful attempt they have learned that if they put a large block directly onto a much smaller block, the tower will fall.

Task specification

Each two building blocks are different, even if they have the same size. For each building block you are given its side length. You are also given an integer *D* with the following meaning: you are not allowed to put block A directly onto block B if the side length of A is strictly larger than *D* plus the side length of B.
Compute the number of different ways in which it is possible to build the tower using **all** the building blocks.
Since this number can be very large, output the result modulo $10^9 + 9$.

Input specification

The first line of the input contains two positive integers *N* and *D*: the number of building blocks and the tolerance respectively. The second line contains *N* space-separated integers; each represents the size of one building block.

Constraints

All numbers in the input files are positive integers not exceeding 10^9.
N is always at least 2.
In test cases worth 70 points *N* will be at most 70.
Out of those, in test cases worth 45 points, *N* will be at most 20.
Out of those, in test cases worth 10 points, *N* will be at most 10.
For some of the test cases the total number of valid towers will not exceed 1 000 000. These test cases are worth 30 points in total.
For the last six test cases (worth 30 points) the value of *N* is larger than 70. No upper bound on *N* is given for these test cases.

Output specification

Output a single line containing a single integer: the number of towers that can be built, modulo 1 000 000 009.

Examples

input:

```
4 1
1 2 3 100
```

output:

```
4
```

We can arrange the first three blocks in any order, except for 2,1,3 or 1,3,2. The last block has to be at the bottom.

input:

```
6 9
10 20 20 10 10 20
```

output:

```
36
```

We are not allowed to put a cube of size 20 onto a cube of size 10. There are six ways to order the cubes of size 10, and six ways to order the cubes of size 20.

Solution

A simple cycle over all permutations of cubes was sufficient to score 10 points for this problem. The first possible improvement is to build the tower from the bottom to the top, using a recursive function that backtracks as soon as the current tower becomes invalid. Such solution will skip some of the permutations that do not represent valid towers, hence for some test cases it will be faster. A simple backtracking solution would score 20 points.
There is a simple algorithmic improvement which we can add to the backtracking: at any moment, we can check whether there is at least one possible way to build the rest of the tower using the remaining cubes. This is pretty simple: consider the largest of the remaining cubes. If we want to create a valid tower, we have to place this cube on the top sooner or later. So let's try to do it using a simple greedy approach: if we have an unused cube larger than the one currently on the top that can be added to the top, do it. Clearly, if this approach fails to place the largest unused cube into the tower, it is not possible at all. On the other hand, once we manage to place the largest unused cube, the remaining unused cubes can be stacked on its top in sorted order. If we sort the cubes in the beginning, this greedy check can be implemented in linear time using a simple cycle.
How does this help? Each time we make a recursive call, we will run the above check, and if there are no valid towers for the current branch, we will backtrack immediately. It can easily be seen that the runtime of this improved backtracking is proportional to the actual number of valid towers. Such a solution would score 30 points.
Another algorithmic way to improve the search is to use memoization. Consider any state somewhere in the middle of the search: we have a partially built tower, and some set of unused cubes. What we need to compute is the number of ways in which the unused cubes may be placed in order to create a valid tower. The important thing to note at this moment is that this count does not depend on the order in which the used cubes are placed. The only two pieces of information we need are the set of unused cubes, and the index of the cube on the top of the tower so far.
There are 2^N possible sets of unused cubes and N possible cubes on the top of the partially built tower, hence there are $O(N\,2^N)$ possible states of the search. Note that this is a significant improvement over the initial N! possible states. For each of the states, we can compute the

answer in O(N) time by summing the answers of O(N) recursive calls: each corresponding to adding one of the unused cubes on the top of the tower. This approach scored 45 points. It could either be implemented as recursion with memoization, or as dynamic programming (In the dynamic programming solution, we would process the possible states ordered by the number of unused cubes in decreasing order. Note that whenever we process a state, we already processed all states that can be obtained by adding a cube to its current tower, and therefore we know the answers for these new states.) The recursive implementation could also be improved using the greedy check for no solutions to score at least 55 points.

The optimal solution is surprisingly simple and fast. It is based on the following observation: Remove one largest cube A and consider any valid tower built from the other N-1 cubes. Suppose we now want to insert the largest cube somewhere into the tower. Regardless of how the tower looks like, the number of ways in which we can do it is always the same: we can place A on top of any of the cubes that are within D of its size.

Let C(N-1) be the count of valid towers using the first N-1 cubes, and let X(N) be the number of ways how to insert the largest cube into any such tower. It is easy to see that all X(N)*C(N-1) towers obtained in this way are distinct.

On the other hand, if we take any valid tower with N cubes and remove the largest cube, it can easily be seen that we will always obtain a valid tower with N-1 cubes. Hence there are exactly X(N)*C(N-1) towers built using all N cubes.

This solution can be easily implemented in O(N log N), for example by sorting and then traversing the sorted array using two indices to compute all values X(i).

Results

Rank	Contestant	Country	AL	AR	BO	MP	PI	TO	Score	Medal
GOLD MEDAL										
1	Anna Piekarska	Poland	55	90	60	60	95	100	460	GOLD
2-3	Anton Anastasov	Bulgaria	55	60	50	40	100	100	405	GOLD
2-3	Stjepan Glavina	Croatia	25	40	40	100	100	100	405	GOLD
SILVER MEDAL										
4-5	Ivan Katanić	Croatia	55	90	50	40	100	55	390	SILVER
4-5	Ivica Kičić	Croatia	55	90	50	0	95	100	390	SILVER
6-7	Bogdan-Cristian Tătăroiu	Romania	35	80	50	10	100	100	375	SILVER
6-7	Tomáš Belan	Slovakia	55	100	0	100	100	20	375	SILVER
8	Vladislav Haralampiev	Bulgaria	100	10	50	10	100	100	370	SILVER
9	Jan Milczek	Poland	100	20	20	20	100	100	360	SILVER
10	Vlad-Alexandru Gavrilă	Romania	15	60	40	20	100	45	280	SILVER
BRONZE MEDAL										
11	Rumen Hristov	Bulgaria	5	10	50	70	100	40	275	BRONZE
12	Adrian Satja Kurdija	Croatia	35	30	50	10	100	45	270	BRONZE
13	Adrian Jaskółka	Poland	100	0	-	10	100	55	265	BRONZE
14	Simon Bürger	Germany	65	-	0	0	90	100	255	BRONZE
15-16	Igor Adamski	Poland	25	10	50	10	100	45	240	BRONZE
15-16	Victor-Cristian Ionescu	Romania	-	40	50	40	15	95	240	BRONZE
17	Filip Hlásek	Czech Rep.	15	10	0	40	100	45	210	BRONZE
18	Ján Hozza	Slovakia	25	40	0	40	70	20	195	BRONZE
19	Mihail Kovachev	Bulgaria	55	0	50	0	30	55	190	BRONZE
20	Richárd Palincza	Hungary	0	40	40	0	90	0	170	BRONZE

21			25	-	0	40	15	45	125	
22			15	10	0	10	65	20	120	
23			-	0	-	10	0	100	110	
24			-	20	0	0	65	20	105	
25			-	30	0	0	50	20	100	
26-28			10	0	0	0	70	15	95	
26-28			0	20	0	0	55	20	95	
26-28			15	0	10	10	40	20	95	
29			15	20	50	0	5	0	90	
30			0	20	50	10	5	-	85	
31			-	40	-	0	40	0	80	
32			15	40	0	-	15	-	70	
33			-	0	0	40	15	10	65	
34			0	0	-	0	15	40	55	
35			-	-	0	-	30	20	50	
36			-	20	0	0	15	10	45	
37			-	10	0	0	25	0	35	
38			-	0	-	-	30	-	30	
39			-	0	-	-	15	10	25	
40			-	10	0	0	0	0	10	

Results of the Online Contest (gold & silver medal):

Rank	Contestant	Country	AL	AR	BO	MP	PI	TO	Score	Medal
GOLD MEDAL										
1	Tiancheng Lou	China	100	100	100	100	100	60	560	GOLD
2	Tomasz Kulczyński	Poland	100	90	100	20	100	100	510	GOLD
3-4	Jiang Zhongtian	China	100	80	100	20	100	45	445	GOLD
3-4	Feng Qiwei	China	55	100	50	40	100	100	445	GOLD
5	Bai Chi		100	20	50	70	100	100	440	GOLD
6	Gao Xin	China	100	30	100	0	100	100	430	GOLD
7	rpb		100	100	0	80	100	45	425	GOLD
SILVER MEDAL										
8	楚雨荨	China	100	40	30	20	100	100	390	SILVER
	Lan Zhou	China	100	90	20	40	100	40	390	SILVER
	Yuchao Pan	China	100	40	50	0	100	100	390	SILVER
	Jingshu Mao	China	100	20	30	40	100	100	390	SILVER
12	Balrog	China	15	80	50	40	100	100	385	SILVER
13	Zou Leqi	China	100	40	–	40	100	100	380	SILVER
14	HuangYicheng	China	100	10	50	10	100	100	370	SILVER
15	Ali Babaee Cheshme Ahmad Rezaee	Iran	25	100	0	100	90	50	365	SILVER
16	Shang Wu	China	100	50	50	30	100	30	360	SILVER
17	taobingxue	China	100	10	0	40	100	100	350	SILVER
	Nicholas Jimsheleishvili	Georgia	100	10	40	0	100	100	350	SILVER
	陈许旻	China	100	40	10	0	100	100	350	SILVER
20	Hanjun Dai	China	–	90	50	–	90	100	330	SILVER
21	Yuzhou Gu	China	90	–	30	0	100	100	320	SILVER
22	Shi Chen	China	55	30	50	40	100	40	315	SILVER
23	Jiezhong Qiu	China	100	10	–	0	100	100	310	SILVER
	jiazhipeng	China	100	20	50	20	100	20	310	SILVER
25	Zhuojie	China	100	90	10	–	100	–	300	SILVER
	张伟奇	China	100	0	0	0	100	100	300	SILVER
	Luka Kalinovcic	Croatia	–	–	–	100	100	100	300	SILVER
28	Yuan Deng	China	100	–	50	0	100	45	295	SILVER
	Gritskevich Eugene	Belarus	15	20	50	10	100	100	295	SILVER
30	Shunning Jiang	China	100	10	0	–	70	100	280	SILVER
	章彦恺	China	100	0	0	0	90	90	280	SILVER

Statistics for the tasks (real contest):

 # contestants = 40

	alliances	arithmetic	bodyguards	mp3player	pin	tower
Average number of points	34.46	28.65	23.82	20.56	58.88	45.14
# >=90 pts solutions	3	4	0	2	17	10
# >=50 pts solutions	10	7	13	4	23	13

The problem Bodyguard was not solved by full point, but it was not the last one from the average point of view.

Statistics for the tasks (online contest):

 # contestants = 210 (with non zero results)

	alliances	arithmetic	bodyguards	mp3player	pin	tower
Average number of points	42.19	24.25	23.13	16.51	58.60	53.20
# >=90 pts solutions	39	18	9	4	76	44
# >=50 pts solutions	57	25	47	6	96	50

Photo Gallery

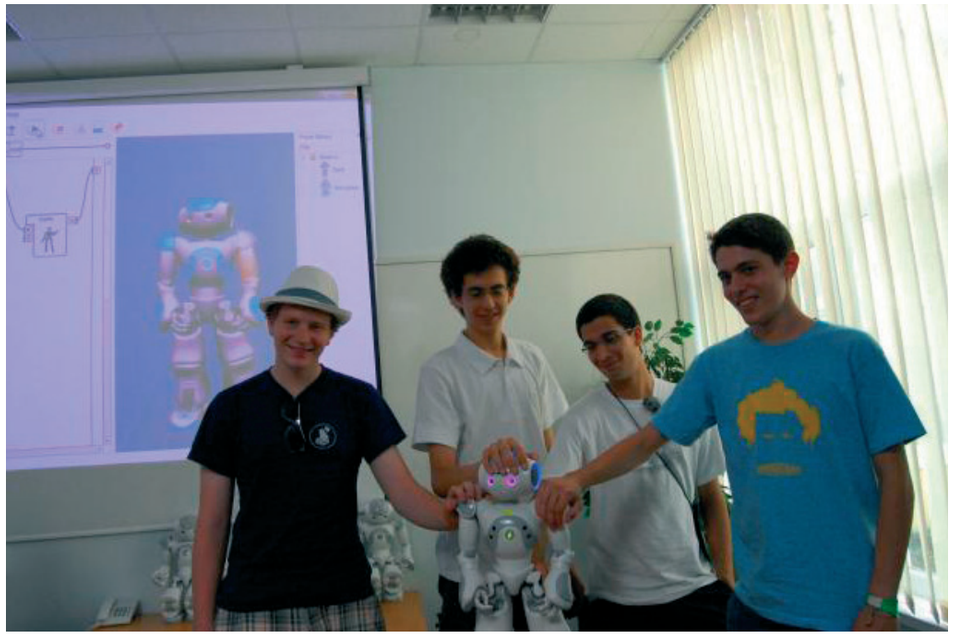
The winner Anna Piekarska from Poland

Two gold medallists Anton Atanasov from Bulgaria and Stjepan Glavina from Croatia

Silver medallist Tomáš Belan from Slovakia

Photo Gallery

Second competition day



Humanoid Nao in the team of Switzerland

Programme

Monday, July 12

Time	Contestants	Leaders
-	**Arrival of Delegations**	
19:00	**Dinner**	
20:00	**Free Time**	**Jury Meeting**

Tuesday, July 13

Time	Contestants	Leaders
08:00	**Breakfast**	
09:30	**Opening Ceremony** (UPJŠ, Moyzesova 11, lecture room M5) Scientific Lecture (prof. V. Geffert: Multiway in-place merging)	
11:30	**Practice Session** (Faculty of Science, Jesenná 5)	
13:30	**Lunch** (Faculty of Science, Jesenná 5)	
15:00	**Sightseeing**	
19:00	**Dinner**	
20:00	**Free Time**	**Jury Meeting** (Faculty of Science)

Wednesday, July 14

Time	Contestants	Leaders
07:15 - 07:45	**Breakfast**	-
08:00 - 08:30	-	**Breakfast**
08:30 - 10:30	**1st Competition**	**Questions**
10:30 - 13:30	**1st Competition**	**Free Time**
13:30	**Lunch** (Faculty of Science)	
14.30	**Results of Evaluation** (Faculty of Science)	
15:00 - 18:00	**Free Time** (Gym, Medická 5)	
18:30	**Free Time**	**Jury Meeting** (Faculty of Science)
19:00	**Dinner**	
20:30	**Bowling - Spot**	

Thursday, July 15

Time	Contestants	Leaders
07:30	**Breakfast**	
08:30	**Trip to High Tatras:** A choice: Štrbské and Popradské pleso (pleso = a glacier lake) B choice: Téry Chalet	
19:00	**Dinner**	
20:00	**Free Time**	**Jury Meeting** (Faculty of Science)

Programme

Friday, July 16

Time	Contestants	Leaders
07:15 - 07:45	**Breakfast**	-
08:00 - 08:30	-	**Breakfast**
08:30 - 10:30	**2nd Competition**	**Questions**
10:30 - 13:30	**2nd Competition**	**Free Time**
13:30	**Lunch** (Faculty of Science)	
14.30	**Results of Evaluation** (Faculty of Science)	
15:30	Popular Lecture (prof. Ing. P. Sinčák: Humanoid robot Nao in CIT TU Košice)	
17:00 - 18:00	**Free Time** (Gym, Medická 5)	
18:00	**Free Time**	**Jury Meeting** (Faculty of Science)
19:00	**Dinner**	
20:00	**Sport, Activities** Prepared by Guides	

Saturday, July 17

Time	Contestants	Leaders
07:30	**Breakfast**	
08:30	**Trip** to Krásna Hôrka Castle and Zádiel gorge	
19:00	**Dinner**	
20:00	**Sport, Activities** Prepared by Guides	**Jury and Organizing Committee Meeting**

Sunday, July 18

Time	Contestants	Leaders
08:00	**Breakfast**	
10:00 - 13:30	**Excursion**: Museum of Aviation in Košice	
13:30	**Lunch** (Student Dormitory, Medická 4)	
16:00	**Awarding and Closing Ceremony** (Historical hall, Šrobárova 2)	
18:00	**Dinner**	

Monday, July 19

Time	Contestants	Leaders
07:30	**Breakfast**	
-	**Departure of Delegations**	

ISBN 978-80-89284-65-8

SPONZORS:

Košice IT Valley
http://www.kosiceitvalley.sk/
Technická univerzita v Košiciach
Letná 9
040 01 Košice

Dopravný podnik mesta Košice, a.s.
http://www.dpmk.sk/
Bardejovská 6
043 29 Košice

EDUXE
http://www.eduxe.sk/
M.C. Sklodowskej 2
851 04 Bratislava

CONTACT

P.J. Šafárik University in Košice
Faculty of Science
Institute of Computer Sciences
WWW: http://ceoi2010.ics.upjs.sk

Address: Jesenná 5,
041 54 Košice, Slovakia

ISBN 978-80-89284-65-8