



# IEEEXTREME PROGRAMMING COMPETITION

2008

PROBLEM &  
INSTRUCTION BOOKLET

## Instructions

- Read all the problems **carefully**.
- Each of the problems has a problem number (shown on top), a title, an approximate "value rank" which gives an idea on how much points it can give, a descriptive text and some examples of inputs/outputs.
- You **don't have to do all** the problems and you **don't have to follow any specific order** for submissions. This allows you to do some strategy play deciding whether to go for the more valuable but more difficult problems first or solve the easier ones, which may give lower point counts.
- Should any of the problems have incomplete information or require more data (e.g: if you are required to code a dot multiplication for vectors and you didn't know what that is) please **feel free to use resources** like the Internet, books or extrasensory perception to learn about ways of solving your problem or for extra examples. The only thing the team **can't use is the help of other human beings** (this includes, but is not limited to, your proctor, members of other teams and your friends and family)
- Please make sure your program receives input and shows output in **EXACTLY the format** that is requested in the text and shown in the examples. This is necessary for the automated judging system to work properly. Also, make sure your solution complies with the procedures described in the "Submissions" section (see below).
- Updates and information will be sent to the teams **via the proctor's email** during the competition. Also, important **updates** will be published in the contest blog page
  - <http://ieeextreme.org>
- Should you have any issues or questions about general contest procedures please feel free to contact our **helpdesk team** during the competition writing email to [ieeextreme@ieee.org](mailto:ieeextreme@ieee.org). In case of issues, our backup mail address is [ieeextreme@gmail.com](mailto:ieeextreme@gmail.com). Please note the helpdesk team will **NOT** answer questions related to the programming problems nor provide extra examples or any other information.
- And most important... **Have fun! :)**

## Notes

Some problems have a very succinct description. This is deliberate. Filling in the gaps is part of the work in devising a solution.

Some problems accept the input at the console, others accept the input from a file whose name is given as an argument to the program in the command line.

Some problems give no hint of the dimension of the input data. This is deliberate: we want to withhold that information from the contestants.

When we say that something is “provided”, it means that you should download that piece of information from the contest site, where you will find it easily.

Most problems can be solved in any of the official languages, for any of the official platforms. Some have specific requirements, having to be solved in a particular language, or for a specific platform, or strictly observing a standard, so that in can be run in Windows and in Linux.

Some problems do not require that contestants submit a program, although contestants may wish to write a program that helps finding the solution.

## Submission procedure

- You must keep **all files** for the solution of each problem **in a separate directory**, identified by the problem code. This means all files of the solution from problem 1 should be inside a directory called "problem1"
- The programming languages allowed in the competition are C, C++ (using gcc), Java (using Sun's JDK) and C# (using the Mono platform). For more details about versions, check the section "Target machine setup" below.
- Inside every problem directory there should be at least one special file called a **buildfile**. This file will be used to build your program in the automated system. After being executed, the file should create an executable file, named "**program**", in the problem directory.
- For each of the problems, your buildfile can be **one of these 3 types**:
  - a **bash shell script**, in which case it should be named "build.sh" and will be executed by the system
  - a **GNU make makefile**, in which case it should be named "Makefile"
  - an **Apache Ant build file**, in which case it should be named "build.xml"
- Apart from the buildfile and the source code files (and libraries or other external resources you want to use) we would advise **NOT** to include any other files (like temporary files, object files or executables from previous compilations).
- You are allowed to use external libraries or other **freely available** third-party code in your solution as long as it can be used for your purpose under an **open source license**, you **include them** in the solution directory.
- Once you have all your source ready and your buildfile working properly to build your executable (remember, named "program"), you should **package the directory in a .ZIP, .RAR, .TAR or .TAR.GZ** file using any of the standard utilities available (like 7-Zip for Windows or any of the command-line utilities in Unix/Linux and MacOS X)
- Name that package file **using your team id and the problem number**, separated by dashes (-) and keeping the proper extension. For example, team 42 submission for problem 11, compressed with tar should be named 42-11.tar.
- Send the package via e-mail to:
  - [solutions.ieeextreme@gmail.com](mailto:solutions.ieeextreme@gmail.com)
  - or [solutions.ieeextreme@googlegmail.com](mailto:solutions.ieeextreme@googlegmail.com)
- Send the submission package file as an attachment, and use your team id and the problem number, separated by dashes (-), as the mail subject.
- If you want to be sure we have received your solutions please request a delivery report. Please do not send anything except solutions to this email address.
- In case there are problems with that mail, send it to our backup mail address: [ieeextreme@gmail.com](mailto:ieeextreme@gmail.com).

## Target machine setup

- All your programs will be **automatically tested** in a target system, therefore it is very important that you follow the submission instructions above.
- Specific versions of programs are available in the target system. Please **DO NOT** use any extensions or features not available in these versions.
- All these programs are **freely available for download** on the Internet for diverse platforms (including Linux, MacOS X and Windows)
- For the **buildfiles**, our automated testing system uses **ant 1.6.5**, **make 3.8** and **bash 3.1-2**
- For **C/C++ submissions**, the system uses **gcc/g++ compilers version 4.0.3-1** (available from <http://gcc.gnu.org/>)
- For **Java submissions**, the system uses **Sun's JDK for the Java 2 Platform Standard edition 5.0** (available from <http://java.sun.com/j2se/1.5.0/systemconfigurations.html>)
- For **C# submissions**, the system uses **Mono SDK version 1.2.1** (available from <http://www.mono-project.com/Downloads>)
- All executables are **included in the system path**, and the usual **environment variables** pointing to the installation directories will also be set (e.g: ANT\_HOME, JAVA\_HOME or MONO\_HOME)
- The target machine system is based on a Linux distribution and will be tested for portability, therefore be careful **not to include any platform-specific code**.

## List of Problems

Problems 2, 3, 6, and 10 are worth 200 points. The others are worth 100 points.

- Problem 1: Merging IP ranges
- Problem 2: Dialogue among Civilizations
- Problem 3: S language interpreter
- Problem 4: Road Construction
- Problem 5: Escape Route
- Problem 6: Focus List
- Problem 7: Snapshot
- Problem 8: GPS Track Area
- Problem 9: Secret Agents
- Problem 10: Consistency of binary decision trees

## Problems

### Problem 1 – Merging IP ranges

Some specialized programs fight advertising and spyware by blocking incoming and outgoing connections using IP blocklist files (also known as blacklists.) A blocklist file contains a simple sequence of IP ranges that usually gets very long. Blocklists with more than 250,000 lines are not unheard of. Here is the initial segment of a blocklist file:

```
3.0.0.0-3.255.255.255
4.0.25.146-4.0.25.148
4.0.26.14-4.0.29.24
4.2.144.64-4.2.144.95
4.2.144.224-4.2.144.231
4.2.144.248-4.2.144.255
4.2.145.224-4.2.145.239
...
```

Some IP blocking programs offer the ability of merging multiple blocklists into optimized blocklists, which are as compact as possible. However this functionality tends to be buggy in the first releases of many of those programs because it requires some effort to come up with a merging algorithm that is both accurate and efficient. This will be one of your tasks for today.

#### **Task**

Please, write a program that takes a sequence of IP ranges (unsorted and coming from several sources) and produces an equivalent result sequence, sorted and with a minimal number of lines. Note that given the input sequence, the corresponding result sequence is unique.

For testing purposes, your program should also support the option of producing the length of the result sequence, a mere integer, instead of the entire result sequence.

#### **Input**

The program gets its input from the standard input stream. The first line of the input contains a single upper-case char, indicating the intended output:

- S requests the entire result sequence;
- L requests only the length of the result sequence.

A sequence of IP ranges with up to 300.000 lines follows, with one IP range per line.

#### **Output**

The program writes its output to the standard output stream. The output may be a sequence of IP ranges in lexicographical order or a single non-negative integer, depending on what has been requested in the input.

### **Sample Input**

S  
3.201.0.0-3.255.255.255  
4.0.25.146-4.0.25.148  
4.0.26.14-4.0.29.24  
4.2.144.64-4.2.144.95  
3.0.0.0-3.200.255.255  
4.2.144.224-4.2.144.231  
4.2.144.248-4.2.144.255  
4.2.145.224-4.2.145.239  
3.100.255.0-3.110.111.111

### **Sample Output**

3.0.0.0-3.255.255.255  
4.0.25.146-4.0.25.148  
4.0.26.14-4.0.29.24  
4.2.144.64-4.2.144.95  
4.2.144.224-4.2.144.231  
4.2.144.248-4.2.144.255  
4.2.145.224-4.2.145.239

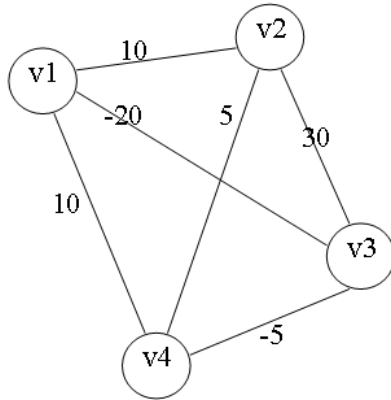
---

## **Problem 2 – Dialogue among Civilizations**

“Dialogue among Civilizations” is a theory in international relations. In its current form, it was first introduced by Mohammad Khatami, former President of Iran ([http://en.wikipedia.org/wiki/Dialogueue\\_Among\\_Civilizations](http://en.wikipedia.org/wiki/Dialogueue_Among_Civilizations)).

Mr. Khatami wants to start a universal conversation, but he does not know in which sequence he should do it, in order to attain the best possible results. Since dialogue with a nation can affect the relations with others, he asks you and a civilization analyst to assist him in solving this problem in the most effective way.

After long research, the analyst found a model for civilizations relationship. The model is a complete weighted graph, where the nodes represent the civilizations and the weights on the edges represents the *level* of relationship between the civilizations at the vertices. Negative weights indicate a conflict and positive weights show a friendly, peaceful relationship. Figure 1 displays such a graph, with four civilizations.



**Figure 1**

In this case, civilization v1 is in good terms with v2 and v4 but has a bad conflict with v3. Civilization v2 is in good terms with all the others, and v3 and v4 have a minor conflict.

At first, before any dialogue among civilizations is carried out, the weights on the edges are between -100 and 100. Note that the graph is complete. Therefore, a graph with  $n$  vertices has  $n(n-1)/2$  edges. In Figure 1, the number of vertices is 4 and the number of edges is 6.

When two civilizations  $i$  and  $j$  start a dialogue, the weights in the graph change according to the following formulas:

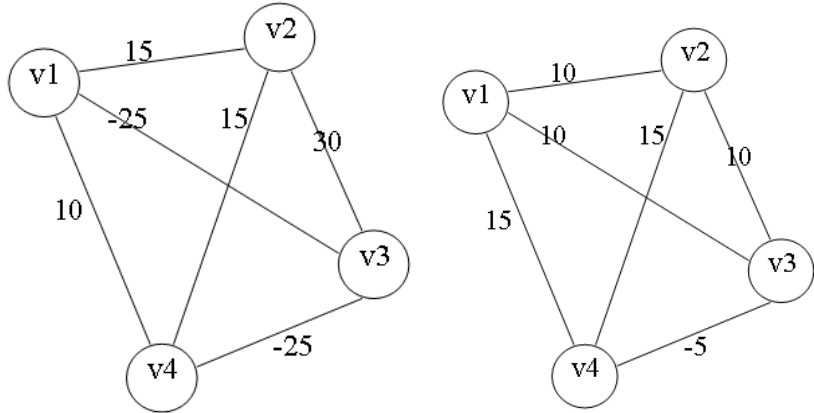
$n$  : number of vertices in graph

$$1 \leq i, j \leq n$$

$$\forall k : 1 \leq k \leq n \wedge k \neq i \wedge k \neq j \Rightarrow e'_{ik} = e'_{jk} = \min\{e_{ik} + e_{jk}, 1000\}$$

$$\text{for other edges : } e'_{ij} = e_{ij}$$

For instance, if nodes 1 and 4 start the dialogue, the graph in figure 1 will change to left graph in figure 2. As you see this condition is worse than the first graph. The negative relations have greater absolute values, and this means more conflict. On the other hand, if node 1 and 2 start a dialogue, the graph in figure 2 will change to right graph in figure 2. As you see this condition is better, since there are more friendly relations than in the first graph.

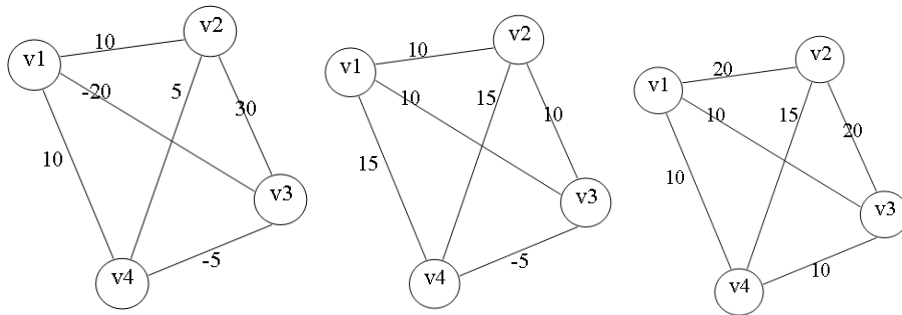


**Figure 2**

Now, let's assume that Mr Khatami is located at node 1. Can you guide him in choosing the nodes with which he should start a dialog, and in which order, so that all the relationships among civilizations become friendly, and the world reaches universal peace? Recall that friendly relationships are described by positive weights.

For instance, there are two ways to solve the graph in figure 1:

1. Mr Khatami (node 1) first speaks to node 2 and then speaks to node 3. This is illustrated in figure 3.

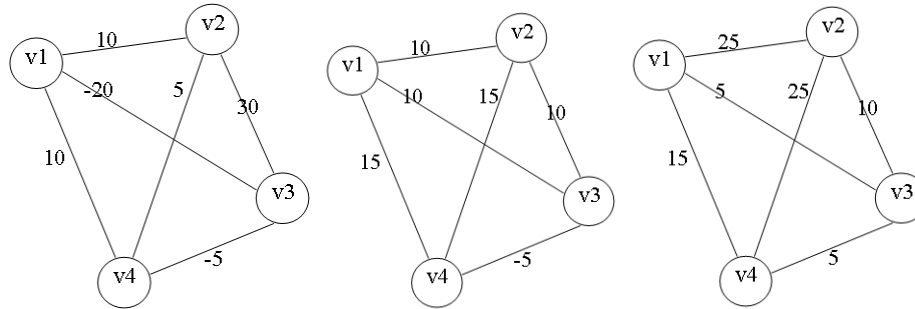


**Figure 3**

In figure 3, the left graph shows the initial graph, which is the same as figure 1. The graph in the middle shows the situation after dialogue between node 1 and node 2. The right graph shows the situation after the second dialogue, between node 1 and node 3. This graph displays universal peace, since all weights are positive.

So, in this case, peace is possible and the sequence (2, 3) is one solution.

2. Mr Khatami (node 1) first speaks to node 2 and then speaks to node 4. This is illustrated in figure 4.



**Figure 4**

This means that the sequence (2, 4) is also a solution.

As you can see, in this case we can reach universal peace in more than one way.

### Task

Please, write a program to solve the dialogue among civilizations. The program inputs a description of the graph, in the format described below, and outputs the message “yes” followed by a solution, if universal peach is possible, or the message “no” only, if it is impossible to reach universal peace.

Your program should solve a graph with number of nodes equal to countries in United Nations. (How many are there?!)

### Input

Your program is to get the file name as an argument in the command line. The file contains the description of the graph, as a square matrix. The first line of the file contains a positive integer,  $n$ , representing the number of nodes. Each of the remaining  $n$  lines contains a list  $n$  numbers, representing a row of the matrix.

### Output

If the graph is solvable, in the sense that there is a way to reach universal peace, the program output two line, on the console. The first line is the message “yes”, without the quotes, and the second line is the sequence of nodes with which to carry the dialogue, separated by single spaces. If universal peace is not possible, the program write “no”, without the quotes.

### Sample Input 1

The graph in figure1 is represented by the following file:

```
4
0 10 -20 10
10 0 30 5
-20 30 0 -5
10 5 -5 0
```

### **Sample Output 1**

The output corresponding to the above input example can be the following:

```
yes  
2 3
```

It can also be:

```
yes  
2 4
```

### **Sample Input 2**

Here are two more examples:

```
3  
0 -10 -10  
-10 0 -10  
-10 -10 0
```

### **Sample Output 2**

```
no
```

### **Sample Input 3**

```
3  
0 10 -20  
10 0 30  
-20 30 0
```

### **Sample Output 3**

```
yes  
2
```

### **Limits**

Your program should be able to handle a graph with all the countries in the world.

---

## **Problem 3 – S language interpreter**

The word “algorithm” has no general definition separated from a particular language (That is why famous Church’s thesis cannot be proved as a mathematical theorem).

What is then the simplest programming language powerful enough to express any algorithm? Such a language was introduced in “Computability, Complexity and Languages”, Martin Davis, Elain Weyuker, Academic Press, 1983, page 15, and is called language “S” (S for “Simple”, maybe). It is proven that “S” language is equivalent to Turing machine, i.e. every Turing’s machine algorithm can be expressed as an “S” program, and vice versa.

## Task

Your task is to write an interpreter for a variant of S language.

## Syntax

The syntax and semantics of the language will be stated using a non-formal description.

“S” program consists of sequence of statements separated by “;” (semicolon).

Statements have the form:

*[statement\_label] statement\_body*

*statement\_label* is a unique sequence of one or more alphanumeric characters of which the first is a letter. The *statement\_label* is optional; when present must be put in brackets, as shown above.

*statement\_body* is one of the following four types of statements:

*var=const*

*var++*

*var--*

*var?statement\_label*

where:

- *var* is a variable name - any alphanumeric sequence beginning with a letter. Variables may have non-negative integer values of any size. By “any size” we mean that integer values are not restricted to 32-bit or even 64-bit integers.
- *const* is a non-negative integer of any size,
- *++* is arithmetic operation of adding 1 to an integer,
- *--* is arithmetic operation of subtracting 1 from a non-negative integer. Note: subtracting 1 from 0 gives 0.
- *var?statement\_label* is a statement in which variable *var* is tested against 0. If value of *var* is not zero program continues execution with statement labeled *statement\_label*, otherwise the next statement is executed.

Space and whitespace characters are ignored.

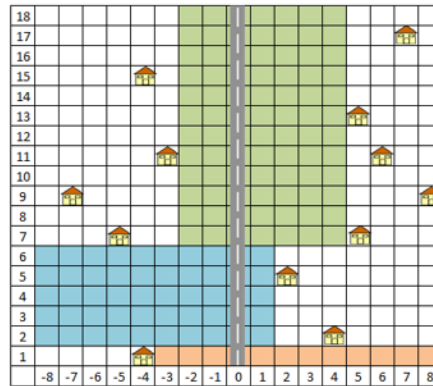
At the start of program execution, all the variables are initialized to 0.

Program starts the execution with the first statement. Program ends the execution when it reaches the end of the list of statements or when it is directed to a non-existing statement label. If, during the execution, the interpreter encounters a syntax error it will print the message “error in line n” and stop the execution.





Figure 1 gives an example terrain (with width 8 and height 18), and shows in color the optimal allocation if you are given exactly 3 parcels.



**Fig. 1** - Example terrain and optimal allocation for 3 rectangular parcels

### Task

Given a terrain (including the locations of the houses) and the number of rectangular parcels you are offered by the government, you must discover the way to allocate the parcels that maximizes the area of terrain you receive. The allocation of the parcels has the constraints defined above.

### Input

In the first line of input come two integers **H** and **W** representing respectively the height and width of the terrain to consider ( $5 \leq H \leq 1000$  and  $5 \leq W \leq 5000$ ).

The second line of input contains a single integer **R** indicating the number of rectangular parcels you must allocate ( $1 \leq R \leq 100$ ).

The third line of input contains a single integer **N** indicating the number of houses in the terrain ( $0 \leq N \leq 5000$ ). Then follow exactly **N** lines, each one with integers indicating the *Y* and *X* coordinates of a single house in the terrain. The houses can come in any order and there may be more than one house in each row (or column).

### Output

The output should consist of a line containing a single integer indicating the largest area you can achieve by allocating **N** rectangle parcels that obey the restrictions given above. The area is measured in number of grid cells, and it includes the road itself.

The sample input and output correspond to the example in figure 1.

### Sample Input

```
18 8
3
12
1 -4
2 4
7 5
9 -7
11 -3
11 6
5 2
13 5
15 -4
17 7
7 -5
9 8
```

### Sample Output

```
146
```

---

## Problem 5 – Escaping route

Miners are trained for the possibility of an accident. Part of their standard equipment is a PDA with a digital map of the mine. This map is given as a matrix of size  $n \times n$ , where each cell contains a number:

- 0, meaning that the cell is empty and the miner can pass without effort.
- $x > 0$ , meaning that there is a wall in that cell. In order to destroy this wall and pass through the cell,  $x$  sticks of dynamite are needed.

Miners also carry with them  $b$  sticks of dynamite, all the time.

In case of an accident, each miner has to escape as fast as possible from his present position to the exit of the mine. They have been instructed to look for the shortest route to the exit. For that, they may destroy some of the walls, using the sticks of dynamite.

The route may have only horizontal and vertical movements between the cells.

### Task

Please, write the program to guide a lone miner from his current position to the mine exit. More precisely, your program must compute the shortest path that leads from the current position by using the empty cells and by breaking wall, using at most the  $b$

### Input

The program should read the input file from the standard input. In the first line of the file there are 6 integers: the map size,  $n$ , the number of sticks of dynamite  $b$ , the coordinates of the current position,  $x_i$ ,  $y_i$ , and the coordinates of the exit position,  $x_f$ ,  $y_f$ . The input file has  $n$  more lines,

each one with  $n$  numbers. Each number represents the value of the corresponding cell in the map (zero if it is empty,  $x > 0$  if  $x$  sticks are needed to break the wall). In this map, the first row is row 0, and the first column is column zero. This means that the upper left corner has coordinates (0, 0) and the lower-right corner has coordinates ( $n-1$ ,  $n-1$ ).

### Output

The program should write the result of the computation on the standard output. The first line will contain the length of the route in number of moves,  $m$ . The following  $m+1$  lines contain two numbers each, separated by a space, representing the coordinates of the cells through which the route passes, one cell per line. The starting and the final points are part of the route, and should both be displayed, if different. If there is no route, the length will be displayed as  $-1$ .

### Sample Input

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

### Sample Output

```
7
0 1
1 1
2 1
2 2
3 2
4 2
4 3
4 4
```

---

## Problem 6 – Focus List

The *focus list* is a list of financial instruments (shares and indexes) the trader must pay special attention to: the trader has to mainly monitor the volatility and dividend table changes.

Years ago, we developed under Windows a small C++ command prompt application that lists the focus instruments. At that time, the number of focus elements was less than 30. Currently the list is more than 2,000 elements and the pressure from the trader to improve the application is everyday more insistent.

After talking with the traders, we extracted the following function to implement: the trader would like to dynamically filter the list on the instrument name, the Reuters and/or Bloomberg code with characters and some wildcards: '\*' for multiple characters and '?' for single character). Furthermore, the trader wants to use three other special characters, as follows:

- '~' to display the previous page;



- '#' to display the next page;
- '^' to reset the filter string.

### **Task**

Your task is to adapt the current software with the above demand. This software is provided to you, “as is”.

### **Input**

A list of 2,000 focus instruments is also provided.

### **Further Requirements**

Your program must compile under UNIX and under Windows using the ANSI Standard.

Please note that you are not to write a new application from scratch. On the contrary you should reuse the available code whenever possible and reasonable.

As the windows workstations of the traders have from one dual core up to 2 quad cores processors, we expect an ad hoc use of open mp in the solution.

We would also that you supply the following documents together with your solution:

- A small analysis of the implementation.
  - A few examples illustrating to the trader the functionality of your solution.
  - A small report showing the advantage of a multi threaded solution *versus* a single threaded one.
- 

## **Problem 7 – Snapshot**

Usual search engines receive a set of keywords and look for all the documents that contain these keywords. The documents are listed in the order of document significance. In this problem we consider the significance of a document for a set of keywords is given by the minimum number of words of the continuous piece of text that contains all the searched keywords.

For instance: consider the keywords “2008” and “IEEEExtreme”, and the following two texts: “The registration for the 2008 edition of IEEEExtreme is now open” and “IEEEExtreme 2008 edition is going to take place on March 8th 2008”. The significance of the first text is 4, and of the second one is 2. If any of the given words is not present in the text, the significance is zero.

### **Task**

Please write a program that reads from the standard input a text in which the words are separated only by spaces, and finds the significance of text against the keywords given as the parameters to your program.

### **Example**

For the input text:

The registration for the 2008 edition of IEEEExtreme is now open  
your program executed as:

```
> snapshot 2008 IEEEExtreme
```

should write 4 on the standard output.

Note: if not all the words are found, the program should return 0.

---

## Problem 8 – GPS Track Area

GPS is a satellite-based system for determining your position on earth. Apart from that, it is also used for tracking. This means some software logs the GPS data and the user can afterwards analyse the data. One application is to compute the area, which is covered by the track.

### **Task**

Please, write a program that computes the area of a given GPS track. The data is in the NMEA 0183 format and is fed to the program using standard input.

You should assume the following:

- the start point and the end point of the track are connected;
- the altitude as irrelevant;
- the track as one single polygon (loops and interruptions do not need to be handled);
- the sentence GPGGA is in the tracking data;
- the earth is a sphere with radius 6,378,137 meters;

The result is the area in square meters, which has to be printed to the standard output. If an error happens during the computation, your program should print "error".

### **Examples**

For example, if goodlog is valid logfile, describing a track whose area is 324 square meters, your program should run as follows:

```
gps2area < goodlog  
324
```

If badlog is an invalid logfile, we should get "error":

```
gps2area < badlog  
error
```

---

## Problem 9 – Secret Agents

Recently Qubitland made big progress in its war against Byteland. Using spy satellites it managed to create a map of all Byteland's bases. Now it wants to destroy all of them as fast as possible. Each base has to be destroyed by a Qubitland agent. Since, in general, there are more bases than agents, some agents will have to destroy more than one base. Qubitland sends an aircraft with the agents to the enemy territory and at any moment one or more agents can jump out of the plane. After landing with parachutes on the ground, the agent starts his mission. He goes to the first base that has been assigned to him and destroys it. After destroying it he can go to another base and destroy it too and so on.

### Task

Please, write a program to schedule the mission for the plane and agents. Each agent has its own moving speed and time he needs to destroy the base. You can assume that the airplane cannot be destroyed and it is flying with constant speed. It can take off and land only once but after starting it has enough fuel to fly as long as it is needed. You can also assume that the terrain is flat and has no obstacles (like rivers, fences or precipices) so agents can go everywhere. Qubitland wants to complete the mission as fast as possible so the shorter the schedule you create, the higher the score you will receive.

### Input

In the first line of the input file there are three numbers  $x$ ,  $y$ ,  $v$  – coordinates of the airport where the plane is at the beginning of the mission and speed of the plane. In the second line there are two numbers  $m$ ,  $n$  – the number of bases and number of agents respectively. In the third line there are  $m$  pairs of numbers – coordinates of bases. After this line there are  $n$  lines. Each line contains one string consisting of up to 10 alphanumeric characters – the name of the agent and two numbers – the speed of the agent and time that he needs to destroy the base. All coordinates are double numbers greater or equal 0.0 and less or equal 10000.0. All speeds and times are double numbers greater or equal 1.0 and less or equal 1000.0. To travel from point  $x_1$ ,  $y_1$  to point  $x_2$ ,  $y_2$  with the speed  $v$  the agent spends the time computed by the following formula:

$$t_{12} = \frac{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}{v}$$

Time is measured in hours. Distance is measured in some units. You can assume that time between jumping out from the plane and landing is 0.0.

### Output

The output should consist of two parts, separated by an empty line. The first part describes the mission of the plane and consists of two kinds of lines:

**FLY**  $x$   $y$  –  $x$  and  $y$  are coordinates of the next target of the plane. They should be double numbers and have no more than 6 digits after decimal point. Note that the target does not have to be directly above one of the bases.

**OUT name** – name is the name of the agent who should jump out at the coordinates given by the last **FLY** line before this line. Note that more than one agent can jump out at the same coordinates.

The second part describes the missions of the agents. There should be one line for each agent who jumped out from the plane. Some agents can stay in the plane to the end of the mission – then there should be no lines for them. Lines can be in arbitrary order. For each agent you should output his name and the list of bases that this agent should destroy. The agent will first go to the first base on the list, destroy it, then go to the second, destroy it and so on. The number of the base is defined by the position of its coordinates in the input and starts with 1. At the end of the list there should be -1.

### Sample Input

```
10.0 0.0 20.0
3 2
10.0 20.0 20.0 20.0 50.0 50.0
A005 2.0 1.0
A007 1.0 5.0
```

### Sample Output

```
FLY 10.0 20.0
OUT A005
FLY 50.0 50.0
OUT A007

A005 1 2 -1
A007 3 -1
```

According to this output the plane should first fly to (10, 20). It takes 1.0 hour and then agent A005 should jump out and plane should fly to (50, 50). The flight takes 2.5 hours so agent A007 will leave the plane in 3.5 hours after start. A005 starts his mission after 1.0 hour from start, spends 1.0 hour on destroying the base 1, then 5.0 hours walking to the base 2 and again 1.0 hour on destroying it so he ends his mission after 8.0 hours. A007 starts his mission 3.5 hours from start and spends 5.0 hours on destroying base 3 so he ends it after 8.5 hours. The time when the last base is destroyed is 8.5 so this is the score for the solution. For this particular case the better solution does not exist.

### Judging

With this task we provide 10 test cases. You should write a program that for each test generates the mission schedule which is as short as possible. The length of the mission is the time from the taking off of the airplane after which the last base will be destroyed. You do not have to care about what happens with agents after completing their missions. This problem is very hard so you do not have to find the best solution but you should find as good solution as you can. After the end of the contest for each test for which you generated a correct output the number of points will be calculated in the following way:

$$p = \left( 1 - 0.8 * \frac{S - S_{min}}{S_{max} - S_{min}} \right) * M$$

Where  $p$  is number of points,  $S$  is your score,  $S_{min}$  is score of the best participant,  $S_{max}$  is score of the worst participant and  $M$  is maximum number of points for the test. The score is equal to the length of the schedule in hours.

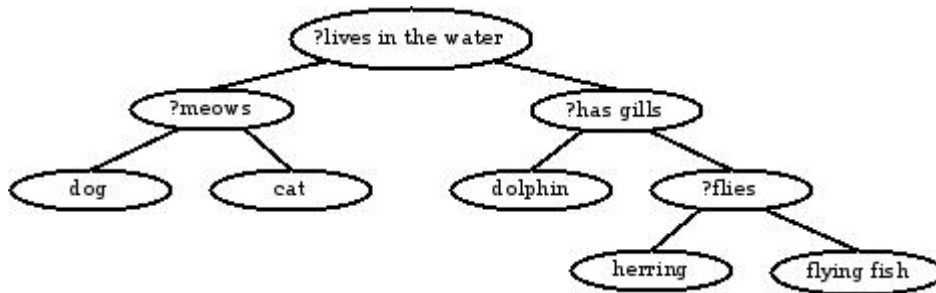
For each input file you should generate the output file with the same name but its extension should be 'out' instead of 'in'. You can use provided program (binaries for Windows, MacOS and Linux) to check if your output file is correct (but not to score it).

## Problem 10 – Consistency of binary decision trees

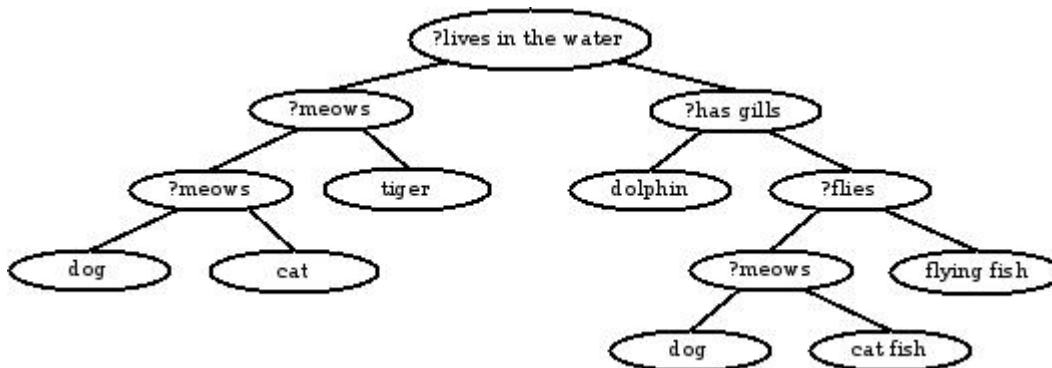
Decision trees are an important technique used in knowledge representation, classification systems and AI computer games. In a decision tree, each branch node is called a decision node and represents a choice between some alternatives. As for the terminal nodes, each one represents a particular classification or a particular entity.

A decision tree is said to be binary if each decision node signifies a choice between exactly two alternatives, i.e., a YES/NO question. In this problem, it is assumed that each decision node causes an actual split in the domain of entities, meaning that every decision node must have exactly two children.

To expound, the following binary decision tree represents some knowledge about animals. At each decision node, the left sub-tree represents the NO answer, and the right sub-tree represents the YES answer.



A decision tree is said to be inconsistent if some contradiction could be derived from it. Here is an example of an inconsistent decision tree:



There are two contradictions in this tree: (1) the cat meows and does not meow; (2) the dog lives in the water and does not live in the water.

Decision trees are usually built in a way that ensures consistency. However, checking the consistency of a binary decision tree is an interesting problem by itself. The definition of inconsistency can be directly applied; however the careful analysis of the circumstances that make a tree inconsistent may provide a simpler solution to program.

### **Task**

Please, write a program to check the consistency of a binary decision tree.

### **Input**

The program gets its input from the standard input stream. The input represents a binary decision, defined using the following recursive rules:

- The root of the tree is the first line;
- The left sub-tree is in the following lines;
- The right sub-tree is in the following lines.

### **Output**

The program writes its output to the standard output stream. The output is a line with a single word. The possible results are: CONSISTENT, INCONSISTENT.

### **Sample Input**

```
?lives in the water
?meows
dog
cat
?has gills
dolphin
?flies
herring
flying fish
```

### **Sample Output**

```
CONSISTENT
```

### **Further requirements**

Your program in this problem is to be written in Algol60, using the RHA Algol60 V5.5 compiler for MSDOS, available at <http://www.angelfire.com/biz/rhaminisys/algol60.html>. This compiler runs on Windows.

Algol60 is a fundamental historical language we all have heard of, but in which most of us have never programmed. Well, it's time to fill that gap in our resumés: from now on, we may add Algol60 in the list of the languages that we have mastered. This will certainly create a deep impression on our prospective employers.

## Disclaimer

All the brands, names and registered trademarks that may appear on this document are marks (trademarks, service marks, registered trademarks, or registered service marks) of their respective owners in the USA and/or other territories.

Some of the materials contained in this document are included from articles in the Wikipedia project (<http://wikipedia.org>) or other sources covered by open licenses (like Creative Commons) and are used here for non-profit purposes. Other legal terms may apply to this document. Before using the content in the booklet for any other purpose than participating in the 2008 edition of IEEEExtreme contest, please contact the team at [ieeextreme@ieee.org](mailto:ieeextreme@ieee.org) to request permission.

No computers were harmed during the creation of this booklet.

---