

IEEEXTREME PROGRAMMING COMPETITION

2008

**PROBLEM &
INSTRUCTION BOOKLET
#3**

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. In case of issues, our backup mail address is 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 it 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
 - or 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.

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

Problem 15: Coins

Problem 16: Soccer AI

Problems

Problem 15 – Coins

Alice and Bob were sitting in the sun; drinking orange juice; and watching some migrating ducks fly to Africa.

"Look", noted Alice, "one of the ducks left a trail of golden coins on the floor".

"Great!" exclaimed Bob, "let's play a game with this line of coins. We will take turns, where each one of us will flip one coin from 'head' into 'tail' state".

"Ok", agreed Alice and added, "but when we flip a coin, we can also opt to flip the coin immediately after it, even if that coin is a 'tail', in which case it becomes a 'head'".

"And whoever can not play - loses" cried both of them simultaneously.

Cunning Bob knew that he could count on witty IEEEExtreme contestants to help him win. Can you help him do that?

Task

Your task is to write a program that given a string of H/T letters, computes a winning move for the flip-coin game, if there is one, or reports that there is no winning move, if this is the case. A winning move is a legal move such that either the player wins immediately (because there are no more coins to flip), or else, after any subsequent move by the opponent there is a winning move for the player.

For example, if the input is TTTT then Bob lost the game (there is no "head" so he can not play and thus he lost). For the input TTTHTTTT, Bob wins by flipping the fifth coin; for the input TTHHT, Bob wins by flipping both "Heads" (third and fourth coins); for the input THHTHTHT, Bob wins if he flips coins 2 and 3.

Input

The input file to be read from the console contains one line in which there is a string entirely composed of the letters H and T, representing the state of the coins, as explained.

Output

The output file, to be written at the console, contains one line, with one number. A positive number N means that flipping the N th coin is a winning move. A negative number, written $-N$, means that flipping the N th and the $N+1$ th coins is a winning move. Zero, written 0, means that there is no winning move. Note that, in general, there can be several winning moves, for a given list of coins. Your program can output any of them.

Sample Input 1

TTTT

Sample Output 1

0

Sample Input 2

TTTTHTTTT

Sample Output 2

5

Sample Input 3

TTHHT

Sample Output 3

-3

Sample Input 4

THHHTHTTHT

Sample Output 4

6

Sample Input 5

THHHTTTTHT

Sample Output 5

0

Problem 16 – Soccer AI

Soccer is considered to be the “King of Sports” in most of the world. The game is quite simple: each team wants to send the ball into the opponent team’s goal, the more times, the better. Still, there are some complicated rules, namely the offside rules and rules defining what goalkeepers can do, cannot do and what can be done to them. So, in this problem, we will simplify...

First, suppose that there are no goalkeepers. That’s a huge simplification, but understandable.

The simplified rules for offside are the following: a player is considered to be offside when all the following four conditions are true:

1. The high pass was made by a co-player of the player examined for offside
2. The ball is heading towards the goalpost of the defending team

3. The player, examined for offside, lied behind all the players of the defending team at the moment that the high pass was made.
4. The player, examined for offside, is not the one who initially had the ball.

If a player is offside, he should not move towards the end position of the ball after an attacking pass is made, because that would result in cancelling the attack effort of his team.

Under these simplified rules, wouldn't it be interesting to estimate which player or players – not being offside – are in best position to get the ball, after a high pass is made?

Consider that for each high pass we know the exact location (in x - y coordinates) where the ball will finally land after the pass is made. Moreover, we suppose that all players move at a constant speed and that the first team attacks from left-to-right whereas the second team attacks from right-to-left. The attacking team is the one who initially has the ball; the other team is the defending team.

As an example, consider the following figure which corresponds to the first input in the samples below. In this figure, the players of the first team (attacking from left-to-right) are represented by white circles and the players of the second team (attacking from right-to-left) by cyan circles. Below every player, the coordinates (in $[x, y]$ format) and the speed (in meters/sec) of the player are shown. The player initially having the ball has a thick red line around its circle and the final position of the ball is marked by an orange X.



In this figure, we observe that the player initially having the ball belongs to the first team and makes a high pass towards the goalpost of the defending team. The player with coordinates $[89, 41]$ is closer to the end position of the ball, but he should not move towards that point because he is offside. At the moment when the high pass is made, the non-offside players closest to the end position of the ball are the ones with coordinates $[75, 44]$ and $[75, 32]$. Since they move at the

same speed, they reach the landing position of the ball at exactly the same time. (But note that this particular case: in general, not all speeds are the same, as the example also shows.)

Task

Please, write a program that, given a description of the playing field, with the positions and speed of all the players, the player who has the ball and the end point of the ball after the player who has the ball makes a high pass, finds the non-offside players that are closest, in terms of time, to the end position of the ball at the exact moment the high pass is made.

Input

The input file is given as an argument to the program in the command line. It is formatted as follows:

- The first line has two comma-separated positive integer numbers. The first one, A, represents the number of players of the first team, and the second one, B, represents the number of players of the second team.
- In each of the following A lines, there are three comma-separated positive integer numbers, representing the X and Y coordinates and the speed in meters/second of each of the players of the first team.
- After that, in each of the following B lines we have the data for the players of the second team.
- There is one more line, with three comma-separated positive integer numbers. The first number ranges from 1 to (A+B) and represents the player (in order of appearance in the input file) that currently has the ball and makes the high pass. The other two numbers represent the X and Y coordinates that map the location of the end position of the ball.

Output

Your program writes at the console a list of comma-separated pairs representing the coordinates of the players who can reach the end position of the ball faster, after the pass is made. There will be one pair per line, write in the order they appear in the input file.

Sample Input 1

```
5, 5
25, 15, 3
28, 42, 4
75, 44, 3
68, 50, 4
89, 42, 6
75, 32, 3
80, 22, 3
50, 51, 5
25, 49, 6
11, 26, 6
4, 88, 38
```

Sample Output 1

```
75, 44
```

75, 32

Sample Input 2

3, 2

94, 22, 5

95, 44, 8

89, 45, 5

88, 50, 5

85, 40, 6

2, 94, 21

Sample Output 2

94, 22

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 to request permission.

No computers were harmed during the creation of this booklet.
