IDI Open
Programming Contest
April 19th, 2008

The Problem set

Problemsetters

Nils Grimsmo, IDI/NTNU
Truls A. Bjørklund, IDI/NTNU
Erling Alf Ellingsen, Medallia
Erik Axel Nilesen, BearingPoint
Andreas Björklund, Logipard

Tips

- Tear the problem set apart and share the problems among you.
- Problems are not ordered by difficulty.
- Try solving the easy problems first. Two problems in this set are tagged with "easy" to help you getting started.
- If you get "incorrect answer" on a problem, you can print your program and debug it on paper while you let someone else work on a different problem on the computer.
- Contact Truls or Nils if you need help.

Rules

- Each team consists of one to three contestants.
- One computer is used per team.
- You may not cooperate with persons not on your team.
- You may print your programs on paper to debug them.
- What you may bring to the contest:
  - Any written material (Books, manuals, handwritten notes, printed notes, etc).
  - Pens, pencils, blank paper, stapler and other useful non-electronic office equipment.
  - NO material in electronic form (CDs, USB pen and so on).
  - NO electronic devices (PDAs and so on).
- The only electronic content you may consult during the content is that specified by the organiser (see the web-page). You may not copy source code from web pages, etc.
- Your programs should read from standard in and write to standard out. Writing to standard error will result in a failed submission. C programs should return 0 from `main()`.
- Your program may use at most 100MB of memory.
- Your programs may not:
  - access the network,
  - read or write files on the system,
  - talk to other processes,
  - fork,
  - or similar stuff.
  - If you try, your program will hang or crash. If it hangs, it will take a couple of minutes before others will be able to run their programs. And please do not crack somebody who uses their spare time trying to give you something valuable.
- Show common sense and good sportsmanship.

Problem A

Vampire

Vampire is a popular roleplaying game. As most roleplaying games, Vampire uses dice to determine random events. The most common use of the dice is to determine if you succeed or fail at a specific task. A task might be shooting another player, avoid falling out of a window, dodge a hit from an opponent etc.

The dice used in Vampire is 10-sided and the rules are as follows: You are allowed to roll x dice and you need y or more points to succeed. If a dice shows 8, 9 or 10 you get one point. This means that 1 through 7 gives no points. In addition, if you are lucky enough to roll a 10 you get one extra dice roll. This means that it is possible to get 2 or even more points even if you started out with only one dice.

An example: Truls tries to avoid falling into a trap. To see if she succeeds she has to roll 4 or more successes on 5 dice. The first roll gives her two 10s, one 4, one 6 and one 3. This means she only got two successes, but because she got two 10s she gets to roll two more dice. This time she rolls another 10 and a 2. She now has three successes and one more dice roll. The last roll lands on 5 and Truls falls into a big pit and dies.

But how big was Truls' chance of avoding death in the first place? Truls asks you to device a program that for a given number of dice and a point requirement tells her her chance of surviving.

Input specifications

The first line gives $1 \leq n \leq 100$, the number of cases. Then follow two numbers x and y on each line, where $0 \leq x \leq 100$ is the number of dice and $0 \leq y \leq 100$ is the number of points needed to succeed.

Output specifications

The output should consist of one line per case with the chance that the rolls succeed. It should be rounded to three decimal places.

Sample input

```
4
1 1
1 2
6 3
2 9
```

Output for sample input

```
0.300
0.030
0.320
0.000
```

Problem B

TV Battle

To your immense frustration, you only have one television set in your family. Deciding what TV show to watch at any given time is the source of a lot of argument. As an example, your little sister Anne always wants to watch ”Desperate Housewives” while you want to watch ”C.S.I.”. You believe it would be beneficial to use automated tools to settle these arguments.

You therefore decide to construct a system that will decide which show that should be on at any given point in time. During a season each show is typically sent on the exact same time each week. You therefore decide to solve this problem for one week, and let this solution decide for a whole season. Each member of the family is given y points which he/she can distribute among his/her favorite shows. We assume that each show starts at the same time and lasts for d consecutive minutes each week. Each TV show is then assigned the total number of points given to it by all family members in total. You should write a program that maximizes the total number of points of the TV shows shown on your television set.

Input specifications

The first line of input gives $1 \leq t \leq 30$, the number of test cases. The first line for each test case gives $1 \leq n \leq 100000$, the number of TV shows that have gotten any points from your family. Then follow n lines with 3 space separated integers s_i, d_i and p_i, where each line describes one TV show. s_i describes the minute at which show i starts, and d_i describes the number of minutes show i lasts ($0 \leq s_i < s_i + d_i \leq 10080$). If $s_i + d_i = k$, it means that you can start watching a new show at time k. p_i describes the total number of points show i has gotten in your family ($1 \leq p_i \leq 2000$).

Output specifications

There should be one line of output for each test case containing the maximum sum of points for the TV shows shown on the television set.

Sample input

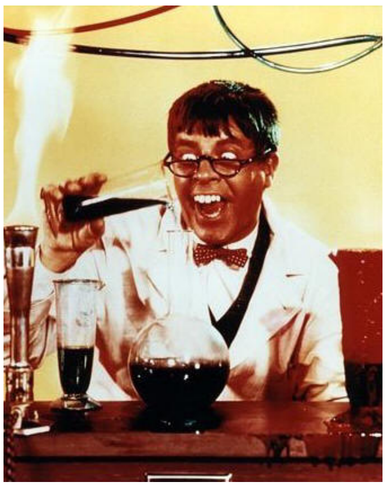
```
1
3
3 8 10
1 4 6
6 4 5
```

Output for sample input

```
11
```

Problem C

The nutty professor (Easy)



Lasse is a nutty professor. He believes strongly that in order to make computer programs for the future, one should make them parallel. In order to convince the rest of the world that he is correct he wants to carry out an experiment. The experiment proceeds as follows: For a number of problems where a serial implementation is available, he estimates the number of times this program will be executed within the next year (when the program will probably be rewritten). He orders his slave, Magnus, to develop a parallel version of the program and measures the amount of time Magnus uses developing it. At the end, they measure the execution time of both the parallel and the serial version.

Based on the collected data, Lasse wants to know in which cases the overall number of man-hours spent on this program is minimized by parallelizing. Man-hours are spent developing the parallel version and waiting while programs execute.

Input specifications

The input has $t \leq 1000$ cases, where t is given by the first line of input. Each test case is given by a line with four integers d, n, s and p separated by a single space. d is the time spent developing the parallel version ($0 \leq d \leq 1000000$), and n is the expected number of times this program will be executed during the next year ($0 \leq n \leq 100000$). s and p are the running times of the serial and parallel version of the program respectively ($0 \leq s, p \leq 1000$).

Output specifications

For each test case output "parallelize" on a single line if it is beneficial to parallelize. If it is not beneficial to parallelize, the outputted line should be "do not parallelize". If the expected total time spent with the program is similar regardless of whether it is parallelized or not, the outputted line should be "does not matter".

Sample input

```
3
10 2 3 2
20 5 8 2
0 2 1 1
```

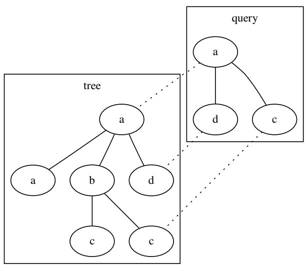
Output for sample input

```

do not parallelize
parallelize
does not matter
```

Problem D

Tree of Pain



Today Erik Axel's wife chose a green tie, so he knows all unit tests will pass. This is most fortunate, as his boss has a tough assignment for him. There was an article in PHB-weekly describing a study about organizational structures, performed by a group of Swedish psychologists. They had identified a set of patterns which they deemed dangerous to have in any organization tree, for example an engineer having an economist somewhere above him.

The boss has copied all the patterns from the article, and Erik Axel must now write a program to search for these in his consulting firm's organization tree. For each pattern, he must see if there is an injective mapping f from the rooted query pattern tree P to the rooted target tree T, maintaining labels and ancestorship both ways (liberal translation of his boss' words). This means $f(u) = f(v)$ if and only if $u = v$, label$(u) =$ label$(f(u))$, and u is an ancestor of v if and only if $f(u)$ is an ancestor of $f(v)$.

Input specifications

The first line of input defines the target tree, which has $1 \leq n \leq 10000$ nodes. A node is described by its label, which is a string of length $1 \leq p \leq 10$ of lower case letters, and then possibly its list of children. A list of children is enclosed in parentheses, and separated by commas. Then follows a line with $1 \leq q \leq 100$, the number of queries. Then follow q lines, each defining a query tree, with $1 \leq m \leq 16$ nodes.

Output specifications

For each test case, output a line with "`disaster`" if there is a match for the query, or "`great success`" otherwise.

Sample input

```
a(a,b(c,c),d)
2
a(c,d)
a(b,c)
```

Output for sample input

```
disaster
great success
```

Problem E

Eight puzzle

You just got your sweet little brother Erling an entertaining puzzle. It is a 3 x 3 board with eight quadratic pieces, where you can slide a piece into the open slot. After rearranging the pieces randomly, the goal of the game is to get the board into the configuration

1	2	3
4	5	6
7	8	

by sliding pieces one by one.

After playing with a puzzle for a while, Erling claims that he can solve any instance in a minimal number of steps. Since you don't believe him, you write a program to solve the puzzles optimally.

8	5	3
	1	7
6	2	4

Input specifications

The first line of input gives $1 \leq n \leq 100$, the number of test cases, followed by a blank line. Each test case is given by three lines giving the start configuration of the board, each consisting of three symbols, followed by a blank line. The cases all contain the symbols $1 \ldots 8$ and `#` exactly once, where the latter represents an open space.

Output specifications

For each test case output the minimum number of moves to solve the puzzle, or `impossible` if it cannot be done.

Sample input

```
2

123
4#5
786

123
456
87#
```

Output for sample input

```
2
impossible
```

Problem F

The Traveling Orienteerer (Easy)

Lasse is organizing the orienteering world championship in Trondheim in 2008. Since he has been running in Bymarka for decades, he has a lot of route proposals. He must find a route with just the right length, but he doesn't have time to measure them all by running through the control points in order, as he is too busy parallelizing code. The job is therefore given to Ola, who he thinks spends too much time with schoolwork.

As Ola is slightly less interested in orienteering, he figures out a more time-saving way of measuring the length of all the routes. He brings his GPS, visits all the points of all the routes in the most convenient order, and goes home early to do the rest of the job on his computer.

Input specifications

The first line of input gives $1 \leq n \leq 1000$, the total number of control points. Then follow n lines giving their coordinates, with two floating-point numbers x_i and y_i, with $0.0 \leq x_i, y_i \leq 10000.0$. The number of routes is then given by $1 \leq m \leq 100$. Each route is defined by first a line with $2 \leq p \leq 17$, the number of control points (including start and goal), and then a line with p indexes $0 \leq i < n$, identifying them.

Output specifications

For each test case output the total track distance, rounded, with no decimals.

Sample input

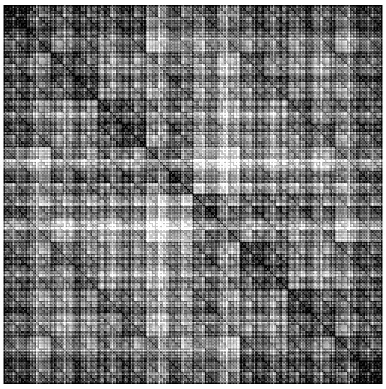
```
5
0.0 0.0
1000.0 1000.0
123.45 0.0
3475.43 7765.4
4325.9865 13.0
2
2
0 1
4
3 1 4 0
```

Output for sample input

```
1414
14999
```

Problem G

Paper



Nils is preparing his Master's thesis, and wants to include the source code for his new logarithmic-time halting problem solver. To minimize the environmental impact of the inevitable millions of printouts of his revolutionary paper around the world, he wants to make his program as short as possible.

You will help by designing an expression optimizer for boolean expressions. The input will be a fully parenthesized expression, using the standard one-character boolean operators (`&`, `|` and `!`).

For each expression, print the length of the shortest equivalent expression (not including spaces).

```
expression ::= variable
             | '(' expression ' ' operator ' ' expression ')'
             | '!' expression
operator ::= '&' | '|'
variable ::= 'x' | 'y' | 'z'
```

Note in particular that parentheses are required for all operators. This means that each operator will contribute three characters to the length, whereas NOTs and variable references contribute one character each.

Input specifications

The input has $1 \leq n \leq 50$ cases, where n is given by the first line of input. Each test case is given by a line with an expression as decribed earlier. Each expression will be no more than 500 characters long (including spaces).

Output specifications

For each test case output the length of the shortest equivalent expression.

In the example, the first expression is never true. It can be written as `(x & !x)`, which has a length of 6. The third example can be simplified by applying De Morgan's law twice (after which it becomes equivalent to the second example).

Sample input

```
3
(((x & !y) & (y & !z)) & (z & !x))
((x & z) | (y & z))
!((!x & !y) | !z)
```

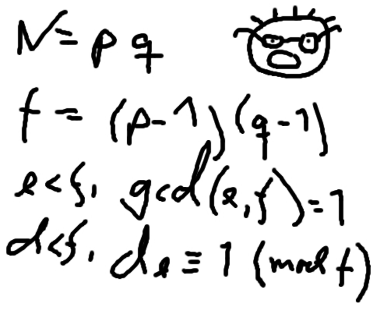
Output for sample input

```

6
9
9
```

Problem H

The Still Embarrassed Cryptographer



The not so young any more, but still very promising cryptographer Børge is implementing a new security module for his company. There were a lot of problems with the old module last time Børge was on holiday, as nobody could understand his code. So his boss have ordered him to keep this new module much simpler.

In this system the secret encryption key is an injective function c from the alphabet onto itself, such that for a string $S = s_1 \ldots s_m$, we have $\mathrm{crypt}(S) = \mathrm{c}(s_1) \ldots \mathrm{c}(s_m)$. The secret decryption key c^{-1} has the property $\mathrm{c}^{-1}(\mathrm{c}(s)) = s$, and is used in the decryption $\mathrm{crypt}^{-1}(T) = \mathrm{c}^{-1}(t_1) \ldots \mathrm{c}^{-1}(t_m)$. Børge's functions crypt() and $\mathrm{crypt}^{-1}()$ send each symbol with a Remote Procedure Call to where the secret keys are stored, deep inside a mountain.

A problem is that $\mathrm{crypt}^q(\mathrm{crypt}(S)) = S$ for some q, and the eager cracker can just keep on applying crypt() on the encrypted message until he gets a readable message. To make the system totally safe, Børge wants to make crypt() throw a `SecurityExceptionInAmundsCodeReally` if q is a small number. Help Børge implement this function.

Input specifications

The first line of input gives $1 \leq n \leq 100$, the number of test cases. Each test case consists of two lines, containing original string S and the encrypted string T respectively, such that $\mathrm{crypt}(S) = T$. You have $1 \leq |S| = |T| \leq 1000$. The strings are from the alphabet "`A`"… "`Z`". The encryption function c is different in each test case.

Output specifications

For each test case, output a line with the number q, or "`mjau`", if this cannot be decided just from S and T.

Sample input

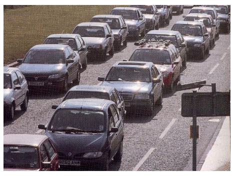
```
3
CRYPTO
CPTOYR
A
A
A
B
```

Output for sample input

```
5
0
mjau
```

Problem I

Traffic load



The National Institute of Traffic frequently measures how much traffic there is on our roads. Their means for doing so is to put two pressure sensitive cords across the road a few meters apart. When a car drives over a cord, a little box at the end of the cord registers the time, once for each wheel-pair of course. A car coming from the left therefore gives rise to four time stamps:

- One at time t ms at the left cord for the front wheel-pair.
- One at time $t + 500$ ms at the left cord for the rear wheel-pair.
- One at time $t + 1000$ ms at the right cord for the front wheel-pair.
- One at time $t + 1500$ ms at the right cord for the rear wheel-pair.

Of course, with a car coming from the right the situation is the same with the roles of left and right swapped. Given the two lists of time stamps, you are to tell how many cars came from the left. At most one car is passing over a cord at a time.

Input specifications

On the first line of input is a single positive integer $1 \leq n \leq 100$ specifying the number of test cases to follow. Each test case begins with a positive even integer $m \leq 200$ on a line of itself telling the number of time stamps in each of the two cord boxes. Next follow m strictly increasing positive integers less than 10^9 telling the time stamps of the left cord box. Then follow m strictly increasing positive integers less than 10^9 telling the time stamps of the right cord box.

Output specifications

For each test case there should be one line of output containing the number of cars which came from the left.

Sample input

```
2
4
17 517 1432 1932
432 932 1017 1517
6
235 451 735 951 2351 2851
1235 1351 1451 1735 1851 1951
```

Output for sample input

```
1
2
```