2004 Stanford Local Programming Contest

Saturday, October 9, 2003

Read these guidelines carefully!

Rules

1. You may use resource materials such as books, manuals, and program listings. You may *not* search for solutions to problems on the Internet, though you are permitted to use online language references (e.g., the Java API documentation). You may *not* use any machine-readable versions of software, data, or existing electronic code. All programs submitted *must be typed during the contest*. No cutting and pasting of code is allowed.
2. Students may not collaborate in any way with each other or anybody else, including people contacted via Internet.
3. You are expected to adhere to the honor code. You are still expected to conduct yourself according to the rules, even if you are not in Gates B02.

Guidelines for submitted programs

1. All programs must be written in C, C++, or Java. For judging, I will compile the programs in the following way:

   - `.c`: using `gcc -lm`
   - `.cc`: using `g++ -lm`
   - `.java`: using `javac`

   All programs will be compiled and tested on a Leland `elaine` machine. Compilation errors or other errors due to incompatibility between your code and the `elaine` machines will result in a submission being counted incorrect. The base filename for each problem will be listed in the problem statement; please use only the listed filename extensions.
2. **Java users:** Please place your `public static void main()` function in a public class with the same name as the base filename for the problem. For example, a Java solution for the `test` program should be submitted in the file `test.java`, should contain a `main()` in `public class test`, and will be run using the command `java test`.
3. All solutions must be submitted as a single file.
4. All programs should accept their input on **stdin** and produce their output on **stdout**. They should be batch programs in the sense that they do not require human input other than what is piped into stdin.
5. Be careful to follow the output format described in the problem. I will be judging programs based on a `diff` of your output with the correct solution. As a note, each line of an output file must end in a newline character, and there should be no trailing whitespace at the ends of lines.

How will the contest work?

1. This year, the expected number of contestants actually exceeds the combined capacity of Gates B02 and the PUP cluster in the Gates basement! **To deal with this, we are allowing competitors to work remotely this year.**

2. If you chose to work remotely from a home computer, you may want to test out the submission script tonight (Friday) to see if it works for you. To do this, **submit a solution for the test problem** shown on the next page. You may not see a result from the grading program immediately. However, you should receive a grading response by midnight; if you do not, please let me know. I will also be grading problems from 12:00 to 12:50 on Saturday.

3. For those who choose to participate onsite, from 12:00 to 12:50, you will select a computer, set up your workspace and complete a test problem. Space in Gates B02 (14 Windows PCs) and the PUP cluster (14 Unix machines) is limited, and will be available on a first-come first-serve basis. If more computers are needed, any remaining contestants will be sent to Sweet Hall (many Solaris machines, including the `elaine` machines on which all testing will be done).

4. At 1:00, the problems will be posted on the main contest page in PDF and PS format, all registered participants will be sent an e-mail that the problems have been posted, and I will distribute paper copies of the problems to contestants who choose to compete in either Gates B02 or the PUP cluster.

5. For every run, your solution will either be accepted or rejected for one of the following reasons: *compile error*, *run-time error*, *time limit exceeded*, *wrong answer*, or *presentation error*. The time limit for a run is *ten seconds total for all test cases*. As a note, the judge solutions for each problem all take under a second to solve all test cases. The number of test cases for each problem will not be revealed in advance, so just write efficient code!

6. Watch your **e-mail on Leland**. I will be sending out any necessary messages as well as notification if your programs are accepted or rejected via e-mail, and the e-mails will go to `your_leland_id@stanford.edu`.

7. If you want to follow the progress of the contest, go to the **standings web page**. A link to this page exists on the main contest page, `http://ai.stanford.edu/~chuongdo/acm`.

8. At 5:00 the contest will end. No more submissions will be accepted. Contestants will be ranked by the number of solved problems. Ties will be broken based on total time, which is the sum of the times for correct solutions; the time for a correct solution is equal to the number of minutes elapsed since 1:00 plus 20 penalty minutes per rejected solution. No penalty minutes are charged for a problem unless a correct solution is submitted. After a correct submission for a problem is received, all subsequent incorrect submissions for that problem do not count towards the total time.

9. The top six contestants will advance to the regionals, subject to the constraint of a maximum of one graduate student per three person team. Full results will be posted as soon as possible after the competition.

Helpful hints

1. **Make sure your programs compile and run properly on the `elaine` machines.** If you choose not to develop on the Leland systems, you are responsible for making sure that your code is portable.

2. If you are using C++ and unable to get your programs to compile/run properly, try adding the following line to your `.cshrc` file

   ```
   setenv LD_LIBRARY_PATH /usr/pubsw/lib
   ```

   and re-login.

3. If you are a CS major and have a working `xenon` account, please work in the **PUP cluster** rather than Gates B02; the PUP cluster has UNIX machines which may be a more convenient programming environment if you intend to use Emacs, etc. If you don't know where the PUP cluster is, just ask!

4. If you are working on a PC in Gates B02, it may be helpful to run a VNC session if you don't like coding from a terminal. Check out Jerry Cain's CS 107 page on using VNC, which can be found at `http://www.stanford.edu/class/cs107/vnc.html`.

5. Read (or skim) through all of the problems at the beginning to find the ones that you can code quickly. Finishing easy problems at the beginning of the contest is especially important as the time for each solved problem is measured from the beginning of the contest.

6. If you need a clarification on a problem or have any other questions, come talk to me in **Gates B02** or the **PUP cluster**, or send an e-mail to `chuongdo@cs`.

The directions given here are based on those taken from Brian Cooper's 2001 Stanford Local Programming Contest problem set. Many thanks also to Daniel Wright and Cristian Cadar for correcting several ambiguities in the problem set and testing out the problems.

0 Test Problem (`test.{c,cc,java}`)

0.1 Description

This is a test problem to make sure you can compile and submit programs. Your program for this section will take as input a single number N and return the sum of all integers from 1 to N, inclusive.

You must submit your solutions via the Leland system and the submission script I have created. **You may develop your programs on whatever platform you want, but submission must be via Leland**. To submit a solution, run the `~chuongdo/acm/submit.pl` script with a single argument: the name of the file you are submitting. For each problem, your solution must adhere to the naming convention specified next to the problem name! For example, to submit a C++ solution to this problem, type:

```
$ ~chuongdo/acm/submit.pl test.cc
```

When you submit your solution, the judge will receive an e-mail and will judge your solution as soon as possible (please be patient!). Once this is finished, you will receive an e-mail stating that you have completed the problem or that you have not, and a reason why not. You can resubmit rejected solutions as many times as you like (though incurring a 20 minute penalty for each rejected run of a problem you eventually get right). Once you have submitted a correct solution, future submissions of that problem will still be graded but will not count towards your final score or total time.

0.2 Input

The input test file will contain multiple test cases. Each test case is specified on a single line containing an integer N, where $-100 \leq N \leq 100$. The end-of-file is marked by a test case with $N = -999$ and should not be processed. For example:

```
5
-5
-999
```

0.3 Output

The program should output a single line for each test case containing the sum of the integers from 1 to N, inclusive. For example:

```
15
-14
```

0.4 Sample C Solution

```
#include <stdio.h>

int main(){
  int n;
  while (1){
    scanf ("%d", &n);
    if (n == -999) break;
    if (n > 0) printf ("%d\n", n * (n + 1) / 2);
    else printf ("%d\n", 1 + n * (1 - n) / 2);
  }
  return 0;
}
```

0.5 Sample C++ Solution

```
#include <iostream>
using namespace std;

int main(){
  while (true){
    int n;
    cin >> n;
    if (n == -999) break;
    if (n > 0) cout << n * (n + 1) / 2 << endl;
    else cout << 1 + n * (1 - n) / 2 << endl;
  }
  return 0;
}
```

0.6 Sample Java Solution

```
import java.io.*;

public class test {
    public static void main (String args[]){
        try {
            BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
            while (true){
                int n = Integer.parseInt (br.readLine());
                if (n == -999) break;
                if (n > 0) System.out.println (n * (n + 1) / 2);
                else System.out.println (1 + n * (1 - n) / 2);
            }
        } catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

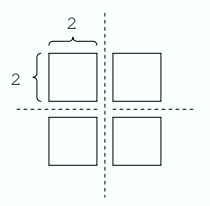
1 Cake Cutting (`cake.{c,cc,java}`)

1.1 Description

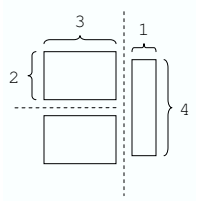
You are given a rectangular cake of integral dimensions $w \times h$. Your goal is to divide this cake into m rectangular pieces of integral dimensions such that the area of the largest piece is minimal. Each cut must be a straight line parallel to one of the sides of the original cake and must divide a piece of cake into two new pieces of positive area. Note that since a cut divides only a single piece, exactly $m - 1$ cuts are needed.

1.2 Examples

If $w = 4$, $h = 4$, and $m = 4$, then the following cuts minimize the area of the largest piece:



However, if $w = 4$, $h = 4$, and $m = 3$, then the following cuts are optimal:



1.3 Input

The input test file will contain multiple test cases, each of which consists of three integers w, h, m separated by a single space, with $1 \leq w, h, m \leq 20$ and $m \leq wh$. The end-of-file is marked by a test case with $w = h = m = 0$ and should not be processed. For example:

```
4 4 4
4 4 3
0 0 0
```

1.4 Output

For each test case, write a single line with a positive integer indicating the area of the largest piece. For example:

```
4
6
```

2 Election (`election.{c,cc,java}`)

2.1 Description

Given the voting preferences of a population of M people, you are to determine the winner of an election among N candidates, numbered $1, \ldots, N$. For this problem, the M people are partitioned into G "groups" where all members within a group have the same voting preferences. The candidate preferences for a group are specified by listing candidates from most preferred to least preferred. Election results are determined by an *instant-runoff voting* procedure.

In this method, the first choices of the M people in the population are counted and the least popular candidate is eliminated. In the event of a tie, the highest-numbered candidate is eliminated. Then, the eliminated candidate is removed from the preference list of all M individuals in the population, and again the least popular candidate is eliminated. This process repeats until only a single candidate is left.

2.2 Input

The input test file will contain multiple test cases. Each input test case begins with a single line containing the integers G and N where $2 \leq N \leq 5$ and $1 \leq G \leq 20$. The next G lines are of the format "M_i a_{i1} a_{i2} $\ldots$ a_{iN}" where $1 \leq M_i \leq 20$ and $a_{i1}, \ldots, a_{iN}$ is a permutation of the integers $1, \ldots, N$. M_i is the number of individuals in the ith group, and $a_{i1}, \ldots, a_{iN}$ is the ordering of the N candidates from most preferred to least preferred for the ith group. The end-of-file is marked by a test case with $G = N = 0$ and should not be processed. For example:

```
3 4
10 1 4 2 3
15 3 2 1 4
12 4 3 2 1
3 2
10 1 2
10 1 2
20 2 1
0 0
```

2.3 Output

For each input case, the program should print the winner of the election on a single line. For example:

```
4
1
```

3 Nash Equilibrium (`nash.{c,cc,java}`)

3.1 Description

A two-player normal form game between two individuals A and B is completely specified by

- $\{a_1, \ldots, a_m\}$, a set of actions for player A,
- $\{b_1, \ldots, b_n\}$, a set of actions for player B,
- PA, an $m \times n$ payoff matrix for player A, and
- PB, an $m \times n$ payoff matrix for player B.

In such a game, both players simultaneously select actions to be played (say a_i and b_j for players A and B, respectively). Then payoffs for each player are determined according to the payoff matrices ($PA[i, j]$ and $PB[i, j]$ for players A and B, respectively). The goal of each player is to maximize his or her payoff.

For player A, the set of *best responses* to a particular action b_j by player B consists of any action a_i which maximizes A's payoff, that is, whose payoff is $max_{i'} PA[i', j]$. Similarly, for player B, the set of best responses to a particular action a_i by player A is any action b_j whose payoff is $max_{j'} PB[i, j']$. A pair of strategies (a_i, b_j) is said to be a *pure strategy Nash equilibrium* if a_i is a best response to b_j and b_j is a best response to a_i.

In this problem, you are given the payoff matrices for two players A and B, and your task is to find and list all pure strategy Nash equilibria.

3.2 Example

Consider the following two-player game in which A and B each have two possible actions, and the payoff matrices are:

PA	b_1	b_2
a_1	1	5
a_2	0	3

PB	b_1	b_2
a_1	1	0
a_2	5	3

Here, if player A chooses a_1, then choosing b_1 allows player B to maximize his payoff ($PB[1, 1] = 1 > 0 = PB[1, 2]$). Similarly, if player B choose b_1, then choosing a_1 allows player A to maximize his payoff ($PA[1, 1] = 1 > 0 = PA[2, 1]$). Thus, a_1 is the best response for b_1 and vice versa, so (a_1, b_1) is a pure strategy Nash equilibrium of this game. However, note that (a_2, b_2) is not a Nash equilibrium; if player A chooses action a_2, b_1 is the best response since $PB[2, 1] = 5 > 3 = PB[2, 2]$.

3.3 Input

The input test file will contain multiple test cases. Each test case begins with a single line containing two integers, m and n, where $1 \leq m, n \leq 20$. The next m lines specify the m rows of payoff matrix PA. The m lines after that specify the m rows of payoff matrix PB. All payoff matrix values will be integers between -100 and 100, inclusive. The end-of-file is marked by a test case with $m = n = 0$ and should not be processed. For example:

```
3 4
1 5 -3 2
4 2 -1 -3
3 -2 5 9
0 -4 -1 -5
-9 2 8 3
```

```
-3 4 -2 3
2 2
1 10
0 5
1 0
10 5
2 2
1 1
1 1
1 1
1 1
0 0
```

3.4 Output

For each input case, suppose that N is the number of Nash equilibria for the described normal form game. Then, the output of the program consists of (1) a line containing the single integer N, and (2) N lines containing two integers i and j, where (a_i, b_j) is the corresponding Nash equilibrium. Note that the program must list all Nash equilibria in lexicographical order, i.e., (a_{i_1}, b_{j_1}) is listed before (a_{i_2}, b_{j_2}) if $i_1 < i_2$ or if $i_1 = i_2$ and $j_1 < j_2$. For example:

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

4 Triangle (`triangle.{c,cc,java}`)

4.1 Description

A *lattice point* is an ordered pair (x, y) where x and y are both integers. Given the coordinates of the vertices of a triangle (which happen to be lattice points), you are to count the number of lattice points which lie completely inside of the triangle (points on the edges or vertices of the triangle do not count).

4.2 Input

The input test file will contain multiple test cases. Each input test case consists of six integers x_1, y_1, x_2, y_2, x_3, and y_3, where (x_1, y_1), (x_2, y_2), and (x_3, y_3) are the coordinates of vertices of the triangle. All triangles in the input will be non-degenerate (will have positive area), and $-15000 \leq x_1, y_1, x_2, y_2, x_3, y_3 \leq 15000$. The end-of-file is marked by a test case with $x_1 = y_1 = x_2 = y_2 = x_3 = y_3 = 0$ and should not be processed. For example:

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

4.3 Output

For each input case, the program should print the number of internal lattice points on a single line. For example:

```
0
6
```

5 Brackets (`brackets.{c,cc,java}`)

5.1 Description

We give the following inductive definition of a "regular brackets" sequence:

- the empty sequence is a regular brackets sequence,
- if s is a regular brackets sequence, then (s) and $[s]$ are regular brackets sequences, and
- if a and b are regular brackets sequences, then ab is a regular brackets sequence.
- no other sequence is a regular brackets sequence

For instance, all of the following character sequences are regular brackets sequences:

```
(), [], (()), ()[], ()[()]
```

while the following character sequences are not:

```
(, ], )(, ([)], ([(]
```

Given a brackets sequence of characters $a_1 a_2 \ldots a_n$, your goal is to find the length of the longest regular brackets sequence that is a subsequence of s. That is, you wish to find the largest m such that for indices $i_1, i_2, \ldots, i_m$ where $1 \leq i_1 < i_2 < \ldots < i_m \leq n$, $a_{i_1} a_{i_2} \ldots a_{i_m}$ is a regular brackets sequence.

5.2 Example

Given the initial sequence `([([]])]`, the longest regular brackets subsequence is `[([])]`.

5.3 Input

The input test file will contain multiple test cases. Each input test case consists of a single line containing only the characters `(`, `)`, `[`, and `]`; each input test will have length between 1 and 100, inclusive. The end-of-file is marked by a line containing the word "end" and should not be processed. For example:

```
((()))
()()()
([]])
)[)(
([][][)
end
```

5.4 Output

For each input case, the program should print the length of the longest possible regular brackets subsequence on a single line. For example:

```
6
6
4
0
6
```

6 Repeatless Numbers (`repeatless.{c,cc,java}`)

6.1 Description

A *repeatless number* is a positive integer containing no repeated digits. For instance, the first 25 repeatless numbers are

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 23, 24, 25, 26, 27, ...

Given an integer n, your goal is to compute the nth repeatless number.

6.2 Input

The input test file will contain multiple test cases, each consisting of a single line containing the integer n, where $1 \leq n \leq 1000000$. The end-of-file is marked by a test case with $n = 0$ and should not be processed. For example:

```
25
10000
0
```

6.3 Output

For each input case, the program should print the nth repeatless number on a single line. For example:

```
27
26057
```

7 Planet Hunting (`planet.{c,cc,java}`)

7.1 Description

In a fictitious solar system consisting of star S, a planet P, and its moon M, planet P orbits in a perfect circle around star S with a revolution period of exactly T Earth days, and moon M orbits in a perfect circle around planet P with an unknown revolution period. Given the position of moon M relative to star S at three different time points, your goal is to compute the distance of planet P from star S.

To do this, consider a two-dimensional Cartesian coordinate system with its origin centered at star S. You may assume that P's counterclockwise orbit around S and M's counterclockwise orbit around P both lie completely within the $x - y$ coordinate plane. Let (x_1, y_1) denote the position of the moon M on the first observation, let (x_2, y_2) denote its position k_1 Earth days later, and let (x_3, y_3) denote its position k_2 Earth days after the second observation.

7.2 Input

The input test file will contain multiple test cases. Each test case consists of two lines. The first line contains the integers T, k_1, and k_2, where $1 \leq T, k_1, k_2 \leq 1000$. The second line contains six floating-point values x_1, y_1, x_2, y_2, x_3, and y_3. Input points have been selected so as to guarantee a unique solution; the final distance from planet P to star S will always be within 0.1 of the nearest integer. The end-of-file is denoted with a single line containing "0 0 0".

```
360 90 90
5.0 1.0 0.0 6.0 -5.0 1.0
0 0 0
```

7.3 Output

For each input case, the program should print the distance from planet P to star S, rounded to the nearest integer.

```
5
```