2006 Stanford Local Programming Contest

Saturday, October 7, 2006

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 -O2 -lm`
   - `.cc`: using `g++ -O2 -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. Make sure you `return 0;` in your `main()`; any non-zero return values will be interpreted by the automatic grader as a runtime error.
3. **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`.
4. All solutions must be submitted as a single file.
5. 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.
6. 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. If you chose to work remotely from a home computer, you may want to test out the submission script 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 Friday midnight; if you do not, please let me know. I will also be grading test problems from 12:00 to 12:50 on Saturday.

2. 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 and the PUP cluster is limited, and will be available on a first-come first-served 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).

3. 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.

4. For every run, your solution will be compiled, tested, and accepted or rejected for one of the following reasons: *compile error*, *run-time error*, *time limit exceeded*, *wrong answer*, or *presentation error*. In order to be accepted, your solution must match the judge output exactly (according to `diff`) on a set of approximately 20-100 judge test cases, which will be revealed after the contest. The time limit for a run is *10 seconds total for all judge test cases combined*, so write efficient code!

5. 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`.

6. 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/2006/`.

7. 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.

8. The top six contestants will advance to the regionals. 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. **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. Note that during the contest, the problems are ordered *alphabetically*, not in order of increasing difficulty!

3. 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.

4. 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!

5. 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 the CS 193D page on using VNC, which can be found at `http://www.stanford.edu/class/cs193d/vnc.html`.

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 to Samuel Gross and David Arthur for their contributions to this problem set.

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` 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 (make sure to run this from an elaine machine):

```
$ ~chuongdo/acm/submit 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.

NOTE: You do not need to submit this problem during the actual contest!

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 Saruman's army (`army.{c,cc,java}`)

1.1 Description

Saruman the White must lead his army along a straight path from Isengard to Helm's Deep. To keep track of his forces, Saruman distributes seeing stones, known as *palantirs*, among the troops. Each palantir has a maximum effective range of R units, and must be carried by some troop in the army (i.e., palantirs are not allowed to "free float" in mid-air). Help Saruman take control of Middle Earth by determining the minimum number of palantirs needed for Saruman to ensure that each of his minions is within R units of some palantir.

1.2 Input

The input test file will contain multiple cases. Each test case begins with a single line containing an integer R, the maximum effective range of all palantirs (where $0 \leq R \leq 1000$), and an integer n, the number of troops in Saruman's army (where $1 \leq n \leq 1000$). The next line contains n integers, indicating the positions $x_1, \ldots, x_n$ of each troop (where $0 \leq x_i \leq 1000$). The end-of-file is marked by a test case with $R = n = -1$.

```
0 3
10 20 20
10 7
70 30 1 7 15 20 50
-1 -1
```

1.3 Output

For each test case, print a single integer indicating the minimum number of palantirs needed.

```
2
4
```

In the first test case, Saruman may place a palantir at positions 10 and 20. Here, note that a single palantir with range 0 can cover both of the troops at position 20.

In the second test case, Saruman can place palantirs at position 7 (covering troops at 1, 7, and 15), position 20 (covering positions 20 and 30), position 50, and position 70. Here, note that palantirs must be distributed among troops and are not allowed to "free float." Thus, Saruman cannot place a palantir at position 60 to cover the troops at positions 50 and 70.

2 Fibonacci (`fib.{c,cc,java}`)

2.1 Description

In the Fibonacci integer sequence, $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$. For example, the first ten terms of the Fibonacci sequence are:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

An alternative formula[1] for the Fibonacci sequence is

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}}_{n \text{ times}}.$$

Given an integer n, your goal is to compute the last 4 digits of F_n.

2.2 Input

The input test file will contain multiple test cases. Each test case consists of a single line containing n (where $0 \leq n \leq$ 1,000,000,000). The end-of-file is denoted by a single line containing the number -1.

```
0
9
999999999
1000000000
-1
```

2.3 Output

For each test case, print the last four digits of F_n. If the last four digits of F_n are all zeros, print '0'; otherwise, omit any leading zeros (i.e., print F_n mod 10000).

```
0
34
626
6875
```

[1]As a reminder, matrix multiplication is associative, and the product of two 2×2 matrices is given by

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}.$$

Also, note that raising any 2×2 matrix to the 0th power gives the identity matrix:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}^0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

3 Football (`football.{c,cc,java}`)

3.1 Description

Consider a single-elimination football tournament involving 2^n teams, denoted $1, 2, \ldots, 2^n$. In each round of the tournament, all teams still in the tournament are placed in a list in order of increasing index. Then, the first team in the list plays the second team, the third team plays the fourth team, etc. The winners of these matches advance to the next round, and the losers are eliminated. After n rounds, only one team remains undefeated; this team is declared the winner.

Given a matrix $P = [p_{ij}]$ such that p_{ij} is the probability that team i will beat team j in a match determine which team is most likely to win the tournament.

3.2 Input

The input test file will contain multiple test cases. Each test case will begin with a single line containing n ($1 \leq n \leq 7$). The next 2^n lines each contain 2^n values; here, the jth value on the ith line represents p_{ij}. The matrix P will satisfy the constraints that $p_{ij} = 1.0 - p_{ji}$ for all $i \neq j$, and $p_{ii} = 0.0$ for all i. The end-of-file is denoted by a single line containing the number -1.

Note that each of the matrix entries in this problem is given as a floating-point value. To avoid precision problems, make sure that you use either the `double` data type instead of `float`.[2]

```
2
0.0 0.1 0.2 0.3
0.9 0.0 0.4 0.5
0.8 0.6 0.0 0.6
0.7 0.5 0.4 0.0
-1
```

3.3 Output

The output file should contain a single line for each test case indicating the number of the team most likely to win. To prevent floating-point precision issues, it is guaranteed that the difference in win probability for the top two teams will be at least 0.01.

```
2
```

In the test case above, teams 1 and 2 and teams 3 and 4 play against each other in the first round; the winners of each match then play to determine the winner of the tournament. The probability that team 2 wins the tournament in this case is:

$$\begin{aligned} P(2\text{ wins}) &= P(2\text{ beats }1)P(3\text{ beats }4)P(2\text{ beats }3) + P(2\text{ beats }1)P(4\text{ beats }3)P(2\text{ beats }4) \\ &= p_{21}p_{34}p_{23} + p_{21}p_{43}p_{24} = 0.9 \cdot 0.6 \cdot 0.4 + 0.9 \cdot 0.4 \cdot 0.5 \\ &= 0.396. \end{aligned}$$

The next most likely team to win is team 3, with a 0.372 probability of winning the tournament.

[2] The input files for this problem quickly get very large; in some cases, simply reading the matrix can be a speed bottleneck for your program. If you are using C++, using `scanf("%lf", &x);` instead of `cin >> x;` to read floating-point values from stdin may greatly speed up your program.

4 Robot (`robot.{c,cc,java}`)

4.1 Decription

Let $(x_1, y_1), \ldots, (x_n, y_n)$ be a collection of n points in a two-dimensional plane. Your goal is to navigate a robot from point (x_1, y_1) to point (x_n, y_n). From any point (x_i, y_i), the robot may travel to any other point (x_j, y_j) at most R units away at a speed of 1 unit per second. Before it does this, however, the robot must turn until it faces (x_j, y_j); this turning occurs at a rate of 1 degree per second.

Compute the shortest time needed for the robot to travel from point (x_1, y_1) to (x_n, y_n). Assume that the robot initially faces (x_n, y_n). To prevent floating-point precision issues, you should use the `double` data type instead of `float`. It is guaranteed that the unrounded shortest time will be no more than 0.4 away from the closest integer. Also, if you decide to use inverse trigonometric functions in your solution (hint, hint!), try `atan2()` rather than `acos()` or `asin()`.

4.2 Input

The input test file will contain multiple test cases. Each test case will begin with a single line containing an integer R, the maximum distance between points that the robot is allowed to travel (where $10 \leq R \leq 1000$), and an integer n, the number of points (where $2 \leq n \leq 20$). The next n lines each contain 2 integer values; here, the ith line contains x_i and y_i (where $-1000 \leq x_i, y_i \leq 1000$). Each of the points is guaranteed to be unique. The end-of-file is denoted by a test case with $R = n = -1$.

```
10 2
0 0
7 0
10 3
0 0
7 0
14 5
10 3
0 0
7 0
14 10
-1 -1
```

4.3 Output

The output test file should contain a single line per test case indicating the shortest possible time in second (rounded to the nearest integer) required for the robot to travel from (x_1, y_1) to (x_n, y_n). If no trip is possible, print "impossible" instead.

```
7
71
impossible
```

5 Spam (spam.{c,cc,java})

5.1 Description

To thwart content-based spam filters, spammers often modify the text of a spam email to prevent its recognition by automatic filtering programs. For any plain text string s (containing only upper-case letters), let $\Phi(s)$ denote the string obtained by substituting each letter with its "spam alphabet" equivalent:

<table>
<tr><td>A</td><td>4</td><td>(four)</td></tr>
<tr><td>B</td><td>|3</td><td>(pipe three)</td></tr>
<tr><td>C</td><td>(</td><td>(left-parenthesis)</td></tr>
<tr><td>D</td><td>|)</td><td>(pipe right-parenthesis)</td></tr>
<tr><td>E</td><td>3</td><td>(three)</td></tr>
<tr><td>F</td><td>|=</td><td>(pipe equals)</td></tr>
<tr><td>G</td><td>6</td><td>(six)</td></tr>
<tr><td>H</td><td>#</td><td>(pound)</td></tr>
<tr><td>I</td><td>|</td><td>(pipe)</td></tr>
<tr><td>J</td><td>_|</td><td>(underscore pipe)</td></tr>
<tr><td>K</td><td>|<</td><td>(pipe less-than)</td></tr>
<tr><td>L</td><td>|_</td><td>(pipe underscore)</td></tr>
<tr><td>M</td><td>|\/|</td><td>(pipe backslash forward-slash pipe)</td></tr>
<tr><td>N</td><td>|\|</td><td>(pipe backslash pipe)</td></tr>
<tr><td>O</td><td>0</td><td>(zero)</td></tr>
<tr><td>P</td><td>|0</td><td>(pipe zero)</td></tr>
<tr><td>Q</td><td>(,)</td><td>(left-parenthesis comma right-parenthesis)</td></tr>
<tr><td>R</td><td>|?</td><td>(pipe question-mark)</td></tr>
<tr><td>S</td><td>5</td><td>(five)</td></tr>
<tr><td>T</td><td>7</td><td>(seven)</td></tr>
<tr><td>U</td><td>|_|</td><td>(pipe underscore pipe)</td></tr>
<tr><td>V</td><td>\/</td><td>(backslash forward-slash)</td></tr>
<tr><td>W</td><td>\/\/</td><td>(backslash forward-slash backslash forward-slash)</td></tr>
<tr><td>X</td><td>><</td><td>(greater-than less-than)</td></tr>
<tr><td>Y</td><td>-/</td><td>(minus forward-slash)</td></tr>
<tr><td>Z</td><td>2</td><td>(two)</td></tr>
</table>

In this scheme, any plain text message s corresponds to exactly one spam-encoded message $\Phi(s)$. The reverse, however, is not necessarily true: a spam-encoded message may correspond to more than one plain text message.

Given a plain text message s, your goal is to determine the number of unique plain text messages whose spam encoding is $\Phi(s)$.

5.2 Input

The input test file will contain multiple test cases. Each test case consists of a single line containing a plain text string s containing from 1 to 100 upper-case letters. The end-of-file is denoted by a single line containing the word "end".

```
BU
UJ
THEQUICKBROWNFOXJUMPEDOVERTHELAZYDOGS
end
```

5.3 Output

For each test case, print the number of unique plain text messages (including the original message) whose spam encoding is $\Phi(s)$. The number of unique plain text messages is guaranteed to be no greater than 1,000,000,000.

```
6
5
144
```

In the first test case, the spam encoding of 'BU' is '|3|_|'. The 6 plain text messages with this spam encoding are 'BU', 'IEU', 'BIJ', 'IEIJ', 'BLI', and 'IELI'.

In the second test case, the spam encoding of 'UJ' is '|_|_|'. The 5 plain text messages with this spam encoding are 'UJ', 'LU', 'IJJ', 'LLI', and 'LIJ'.

6 Sudoku (`sudoku.{c,cc,java}`)

6.1 Description

In the game of Sudoku, you are given a large 9×9 grid divided into smaller 3×3 subgrids. For example,

.	2	7	3	8	.	.	1	.
.	1	.	.	.	6	7	3	5
.	.	.	.	.	.	.	2	9
3	.	5	6	9	2	.	8	.
.	.	.	.	.	.	.	.	.
.	6	.	1	7	4	5	.	3
6	4	.	.	.	.	.	.	.
9	5	1	8	.	.	.	7	.
.	8	.	.	6	5	3	4	.

Given some of the numbers in the grid, your goal is to determine the remaining numbers such that the numbers 1 through 9 appear exactly once in (1) each of nine 3×3 subgrids, (2) each of the nine rows, and (3) each of the nine columns.

6.2 Input

The input test file will contain multiple cases. Each test case consists of a single line containing 81 characters, which represent the 81 squares of the Sudoku grid, given one row at a time. Each character is either a digit (from 1 to 9) or a period (used to indicate an unfilled square). You may assume that each puzzle in the input will have exactly one solution. The end-of-file is denoted by a single line containing the word "end".

```
.2738..1..1...6735.......293.5692.8...........6.1745.364.......9518...7..8..6534.
......52..8.4......3...9...5.1...6..2..7........3.....6...1..........7.4.......3.
end
```

6.3 Output

For each test case, print a line representing the completed Sudoku puzzle.

```
527389416819426735436751829375692184194538267268174593643217958951843672782965341
416837529982465371735129468571298643293746185864351297647913852359682714128574936
```

7 Tic-Tac-Toe (`ttt.{c,cc,java}`)

7.1 Description

In the game of tic-tac-toe, two players take turns marking squares of an initially empty 3×3 grid with either X's or O's. The first player always marks squares using X's, whereas the second player always marks squares using O's. If at any point during the game either player manages to mark three consecutive squares in a row, column, or diagonal with his/her symbol, the game terminates.

Given a board configuration, your goal is to determine whether the board configuration represents the possible final state of a valid tic-tac-toe game.

7.2 Input

The input test file will contain multiple cases. Each test case consists of a single line containing 9 characters, which represent the 9 squares of a tic-tac-toe grid, given one row at a time. Each character on the line will either be 'X', 'O' (the letter O), or '.' (indicating an unfilled square). The end-of-file is marked by a single line containing the word "end".

```
XXXOO.XXX
XOXOXOXOX
OXOXOXOXO
XXOOOXXOX
XO.OX...X
.XXX.XOOO
X.OO..X..
OOXXXOOXO
end
```

7.3 Output

For each input test case, write a single line containing either the word "valid" or "invalid" indicating whether the given board configuration is the final state of some possible tic-tac-toe game.

```
invalid
valid
invalid
valid
valid
invalid
invalid
invalid
```