2005 Stanford Local Programming Contest

Saturday, October 8, 2005

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 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 (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).

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 50-100 judge test cases, which will be revealed after the contest. The time limit for a run is *10 seconds total for all 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`.

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, 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 the CS 193D page on using VNC, which can be found at `http://www.stanford.edu/class/cs193d/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 to David Arthur for contributing several of the problems used in this contest!

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 Scrabble (`scrabble.{c,cc,java}`)

1.1 Description

The game of Scrabble is played with tiles. A tile either has a single letter written on it, or it is blank. In the latter case, the tile may be used to represent a letter of your choice. On your turn, you arrange the tiles to form a word. Each tile may be used at most once, but not all tiles need to be used. Given several Scrabble tiles and a dictionary, determine how many words in the dictionary can be formed using the given Scrabble tiles.

1.2 Input

The input test file will contain multiple test cases. In each test case, the first line contains a positive integer $n \leq 1000$ indicating the number of words in the dictionary. The following n lines each contain a single string with between 1 and 7 uppercase letters, representing a word in the dictionary. No word will appear in the dictionary twice. The next line contains a single string giving the tiles you have available. It will contain only capital letters, representing tiles with that letter on it, and underscores, representing blank tiles. The string will contain between 1 and 7 characters, possibly including duplicate tiles. The end-of-file is marked by a test case with $n = 0$ and should not be processed. For example:

```
5
PROGRAM
CONTEST
PIZZA
ZA
PITA
_PIZA
3
BANANAS
CARROTS
FIGS
A__AA__
0
```

1.3 Output

For each test case, write a single line with the number of dictionary words that can be spelled with the given Scrabble tiles. For example:

```
3
2
```

In the first test case, `PIZZA`, `ZA` and `PITA` can be spelled as `PIZ_A`, `ZA` and `PI_A`. There are not enough letters to spell `PROGRAM` or `CONTEST`. In the second test case, `BANANAS` and `FIGS` can be spelled as `_A_A_A_` and `____`. On the other hand, `CARROTS` would require 6 blanks in addition to the `A`.

2 Convex area (`area.{c,cc,java}`)

2.1 Description

A 3-dimensional shape is said to be *convex* if the line segment joining any two points in the shape is entirely contained within the shape. Given a general set of points X in 3-dimensional space, the *convex hull* of X is the smallest convex shape containing all the points.

For example, consider $X = \{(0,0,0),(10,0,0),(0,10,0),(0,0,10)\}$. The convex hull of X is the tetrahedron with vertices given by X. Note that the tetrahedron contains the point $(1,1,1)$, so even if this point were added to X, the convex hull would not change.

Given X, your task is to find the surface area of the convex hull of X, rounded to the nearest integer.

NOTE: The convex hull of any point set will have polygonal faces. For this problem, you may assume there will be at most 3 points in X on any face of the convex hull.

2.2 Input

The input test file will contain multiple test cases, each of which begins with an integer n ($4 \leq n \leq 25$) indicating the number of points in X. This is followed by n lines, each containing 3 integers giving the x, y and z coordinate of a single point. All coordinates are between -100 and 100 inclusive. The end-of-file is marked by a test case with $n = 0$ and should not be processed. For example:

```
5
0 0 0
10 0 0
0 10 0
0 0 10
1 1 1
9
0 0 0
2 0 0
2 2 0
0 2 0
1 1 2
1 1 -2
1 1 -1
1 1 0
1 1 1
0
```

2.3 Output

For each test case, write a single line with the surface area of the convex hull of the given points. The answer should be rounded to the nearest integer (e.g., 2.499 rounds to 2, but 2.5 rounds to 3). For example:

```
237
18
```

To avoid ambiguities due to rounding errors, the judge tests have been constructed so that all answers are at least 0.001 away from a decision boundary (i.e., you can assume that the area is never 2.4997).

3 Nim (`nim.{c,cc,java}`)

3.1 Description

Nim is a 2-player game featuring several piles of stones. Players alternate turns, and on his/her turn, a player's move consists of removing *one or more stones* from any single pile. Play ends when all the stones have been removed, at which point the last player to have moved is declared the winner. Given a position in Nim, your task is to determine how many winning moves there are in that position.

3.2 Background

A position in Nim is called "losing" if the first player to move from that position would lose if both sides played perfectly. A "winning move," then, is a move that leaves the game in a losing position. There is a famous theorem that classifies all losing positions. Suppose a Nim position contains n piles having $k_1, k_2, \ldots, k_n$ stones respectively; in such a position, there are $k_1 + k_2 + \ldots + k_n$ possible moves. We write each k_i in binary (base 2). Then, the Nim position is losing if and only if, among all the k_i's, there are an even number of 1's in each digit position. In other words, the Nim position is losing if and only if the *xor* of the k_i's is 0.

3.3 Example

Consider the position with three piles given by $k_1 = 7, k_2 = 11$, and $k_3 = 13$. In binary, these values are as follows:

$$\begin{array}{r} 111 \\ 1011 \\ 1101 \end{array}$$

There are an odd number of 1's among the rightmost digits, so this position is not losing. However, suppose k_3 were changed to be 12. Then, there would be exactly two 1's in each digit position, and thus, the Nim position would become losing. Since a winning move is any move that leaves the game in a losing position, it follows that removing one stone from the third pile is a winning move when $k_1 = 7, k_2 = 11$, and $k_3 = 13$. In fact, there are exactly three winning moves from this position: namely removing one stone from any of the three piles.

3.4 Input

The input test file will contain multiple test cases, each of which begins with a line indicating the number of piles, $1 \leq n \leq 1000$. On the next line, there are n positive integers, $1 \leq k_i \leq 1,000,000,000$, indicating the number of stones in each pile. The end-of-file is marked by a test case with $n = 0$ and should not be processed. For example:

```
3
7 11 13
2
1000000000 1000000000
0
```

3.5 Output

For each test case, write a single line with an integer indicating the number of winning moves from the given Nim position. For example:

```
3
0
```

4 Dropping tests (`drop.{c,cc,java}`)

4.1 Description

In a certain course, you take n tests. If you get a_i out of b_i questions correct on test i, your cumulative average is defined to be

$$100 \cdot \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} b_i}.$$

Given your test scores and a positive integer k, determine how high you can make your cumulative average if you are allowed to drop any k of your test scores.

4.2 Example

Suppose you take 3 tests with scores of 5/5, 0/1, and 2/6. Without dropping any tests, your cumulative average is $100 \cdot \frac{5+0+2}{5+1+6} = 50$. However, if you drop the third test, your cumulative average becomes $100 \cdot \frac{5+0}{5+1} \approx 83.33 \approx 83$.

4.3 Input

The input test file will contain multiple test cases, each containing exactly three lines. The first line contains two integers, $1 \leq n \leq 1000$ and $0 \leq k < n$. The second line contains n integers indicating a_i for all i. The third line contains n positive integers indicating b_i for all i. It is guaranteed that $0 \leq a_i \leq b_i \leq 1,000,000,000$. The end-of-file is marked by a test case with $n = k = 0$ and should not be processed. For example:

```
3 1
5 0 2
5 1 6
4 2
1 2 7 9
5 6 7 9
0 0
```

4.4 Output

For each test case, write a single line with the highest cumulative average possible after dropping k of the given test scores. The average should be rounded to the nearest integer. For example:

```
83
100
```

To avoid ambiguities due to rounding errors, the judge tests have been constructed so that all answers are at least 0.001 away from a decision boundary (i.e., you can assume that the average is never 83.4997).

5 Box walking (`box.{c,cc,java}`)

5.1 Description

You are given a three-dimensional box of integral dimensions $\ell_x \times \ell_y \times \ell_z$ The edges of the box are axis-aligned, and one corner of the box is located at position $(0, 0, 0)$. Given the coordinates (x, y, z) of some arbitrary position on the surface of the box, your goal is to return the square of the length of the shortest path along the box's surface from $(0, 0, 0)$ to (x, y, z).

5.2 Examples

If $\ell_x = 1$, $\ell_y = 2$, and $\ell_z = 1$, then the shortest path from $(0, 0, 0)$ to $(1, 2, 1)$ is found by walking from $(0, 0, 0)$ to $(1, 1, 0)$, followed by walking from $(1, 1, 0)$ to $(1, 2, 1)$. The total path length is $\sqrt{8}$.

5.3 Input

The input test file will contain multiple test cases, each of which consists of six integers ℓ_x, ℓ_y, ℓ_z, x, y, z where $1 \leq \ell_x, \ell_y, \ell_z \leq 1000$. Note that the box may have zero volume, but the point (x, y, z) is always guaranteed to be on one of the six sides of the box. The end-of-file is marked by a test case with $\ell_x = \ell_y = \ell_z = x = y = z = 0$ and should not be processed. For example:

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

5.4 Output

For each test case, write a single line with a positive integer indicating the square of the shortest path length. (Note: The square of the path length is *always* an integer; thus, your answer is exact, not approximate.) For example:

```
8
5
```

6 Colored stones (`stones.{c,cc,java}`)

6.1 Description

You are given a row of m stones each of which has one of k different colors. What is the minimum number of stones you must remove so that no two stones of one color are separated by a stone of a different color?

6.2 Input

The input test file will contain multiple test cases. Each input test case begins with a single line containing the integers m and k where $1 \leq m \leq 100$ and $1 \leq k \leq 5$. The next line contains m integers $x_1, \ldots, x_m$ each of which takes on values from the set $\{1, \ldots, k\}$, representing the k different stone colors. The end-of-file is marked by a test case with $m = k = 0$ and should not be processed. For example:

```
10 3
2 1 2 2 1 1 3 1 3 3
0 0
```

6.3 Output

For each input case, the program should the minimum number of stones removed to satisfy the condition given in the problem. For example:

```
2
```

In the above example, an optimal solution is achieved by removing the 2nd stone and the 7th stone, leaving three "2" stones, three "1" stones, and two "3" stones. Other solutions may be possible.

7 RSI (`rsi.{c,cc,java}`)

7.1 Description

You have the goal of becoming the world's fastest two-fingered typist. In this problem, your goal is to optimize the movement of your fingers when typing numeric values in order to ensure that you finish typing a number in the shortest amount of time possible. Your numeric keyboard has the following layout:

7	8	9
4	5	6
1	2	3
0		

For convenience, we refer to the cells above according to their row and column; hence, the "5" key is at position $(2, 2)$, and the "0" key takes up both positions $(4, 1)$ and $(4, 2)$. At time 0, your left pointer finger is on the "4" key and your right pointer finger is on the "5" key. During each time interval, each finger may press the key underneath it, move vertically one position, or move horizontally one position. Although both fingers may move simultaneously within a single time interval,

- at most one key may be pressed during any given time interval,
- the left pointer finger's column must always be less than the right pointer finger's column at the end of each time interval, and
- both fingers must always be above one of the 10 keys in the diagram at the end of each time interval (e.g., neither finger cannot hover over position $(4, 3)$).

The "0" key may be pressed by a finger at either positions $(4, 1)$ or $(4, 2)$.

7.2 Examples

- Typing "56" takes three time units. At time 1, both left and right fingers have moved one position to the right and are on keys "5" and "6" respectively. Then, each key is pressed sequentially.
- Typing "71" takes five time units. During the first two time units, the left finger moves up to the "7" and presses the key. However, the right finger is not allowed to be in the same column as the left finger, and hence the left finger takes two time units to get to the "1" key and one time unit to press it.

7.3 Input

The input test file will contain multiple test cases. Each test case consists of a single line containing a string of between 1 and 100 digits. The end-of-file is marked by a line containing the word "eof" and should not be processed. For example:

```
56
71
902
eof
```

7.4 Output

For each input case, output the minimum number of time units required to type the given digits. For example:

```
3
5
6
```

8 Do it! (`doit.{c,cc,java}`)

8.1 Description

You are the boss of a small lighting fixture company which has n employees. Inspired by Ben Stiller's character in *Starsky and Hutch*, you have recently taken on the habit of telling your employees to "do it!" when you want things done. While n_+ of the n employees respond positively to your "do it!"s, n_- employees respond negatively and n_0 are neutral to your words.

At time 0, each of your employees begins working alone on building a lighting fixture. Each lighting fixture takes 100 units of labor to finish. Normally, each of your employees contributes r units of labor towards finishing his/her lighting fixture during each time interval (or the amount of labor units remaining for the fixture, whichever is smaller). Thus, an employee would normally take $\lceil 100/r \rceil$ time intervals to finish his or her lighting fixture. During an interval, however, if you yell "do it!" over the company intercom, employees who respond positively to your command will do $r + 2$ units of labor during that time interval, whereas employees who respond negatively will do $r - 1$ units of labor during the time interval.

Assuming that each employee works on only his or her lighting fixture, and assuming that you yell "do it!" at most once during each time interval, your goal is to plan a sequence of "do it!"s so as to ensure that the sum of the times needed to finish all n lighting fixtures is minimized.

8.2 Input

The input test file will contain multiple test cases. Each input test case will be given as a line containing four integers, n_+, n_-, n_0, and r (where $0 \leq n_+, n_-, n_0 \leq 1000$ and $1 \leq r \leq 100$). The end-of-file is marked by a test case with $n_+ = n_- = n_0 = r = 0$ and should not be processed. For example:

```
3 1 1 2
1 3 0 2
0 0 0 0
```

8.3 Output

For each input case, the program should print the minimum sum of times needed to finish all n lighting fixtures. For example:

```
188
200
```

In first test case, one optimal strategy is to yell "do it!" in each of the first 25 time intervals. As a result, the 3 positively-responding employees provide 4 units of labor per time interval and thus finish their fixtures in 25 time units. The 1 negatively-responding employee will provide 1 unit of labor per time interval for the first 25 time intervals, 2 units of labor per time interval afterwards, and thus will finish in $25 + 38 = 63$ time units. Finally, the neutral employee will always provide 2 units of labor per time interval and hence will finish in 50 time units. This gives a total of $3(25) + 63 + 50 = 188$ time units.

In the second test case, an optimal strategy is to never yell "do it!". Thus all four employees finish in 50 time units so the total amount of time taken is 200 time units.