

ND Programming Contest, Spring 2009

ND ACM Computer Club

Sunday, April 5, 2009

#1: Counting is Easy

The Problem

Fifty years in the future, you will be the first passenger on a space elevator built of carbon nanotubes. Unfortunately, in a horrible accident of failed unit conversion during the guidance system design, the elevator car will shoot off the end of the elevator shaft and continue for n years toward the nearest friendly galaxy. Assume that you have enough food and water to survive for these n years.

Upon landing, you encounter an alien race. They observe that you are obviously some form of intelligent life, and they attempt to communicate with you. Unfortunately, their concept of counting is quite different than ours. Whereas we have a base-10 system, your friendly hosts represent a number as a linear combination of 2 and 3, counting the twos on their fingers and the threes on their toes. Since they have no concept of “one” (they are a very social race) and all other positive integers can be represented thusly, this works well for them.

This system turns out to be redundant (that is, there are multiple ways to represent any given number). For our purposes, use as few threes as possible.

Your task is to write a program to allow you to convert ordinary decimal numbers into this odd notation. With any luck, you will be able to specify the galactic coordinates of Earth and arrive home safely. Who knew that number theory could be so life-saving?

Sample Input

Input will consist of a series of lines, each with one integer in decimal form. The last integer will be “1”, which indicates the end of the list, and your program should quit upon receiving this.

```
2
3
23
40
15
1
```

Sample Output

Output should consist of two integers per line, separated by one space. The first integer is the twos-count, and the second is the threes-count.

```
1 0
0 1
10 1
20 0
6 1
```

#2: As Quickly as Possible

The Problem

You are an ambulance driver with a passion for computer science.¹ In your free time, you are developing a program that, given a representation of the city streets in your area, will tell you the shortest possible route to every location. Why waste time driving to an emergency when lives are at stake?

For the purposes of this problem, let's focus on a single destination, and let's say that the street pattern is a regular grid (i.e., like Manhattan, or the complete opposite of Pittsburgh). The local streets department gives you a matrix of numbers representing the transit times for each segment of the grid. The hospital is located at the top-left of the grid and you are trying to reach the bottom-right corner of the grid. The grid consists of $n \times m$ intersections (horizontal by vertical).

For simplicity, all horizontal roads have transit times of 1 minute between intersections. Vertical roads have varying times; your program's input will consist of rows of n numbers giving the transit times for each vertical segment in a given row. Input will consist of $m - 1$ rows of numbers, since there is one fewer row of road segments than of intersections.

Your job, given the dimensions $n \times m$ and the matrix of segment transit times, is to report a single number that is the minimal transit time from the hospital to the emergency location.

Sample Input

There will be one line giving n and m , with a space between them, followed by $m - 1$ lines, each corresponding to one row. Multiple problems may appear; the end of the list is delimited by $n = m = 0$.

```
4 4
1 1 2 3
3 3 1 4
3 2 1 0

0 0      # end of sequence
```

The sample input matrix above is visualized in Figure 1.

Sample Output

Print exactly one decimal integer per input problem, giving the minimum transit time from the hospital to the opposite corner.

```
5
```

¹Somewhere out there, one of these exists.

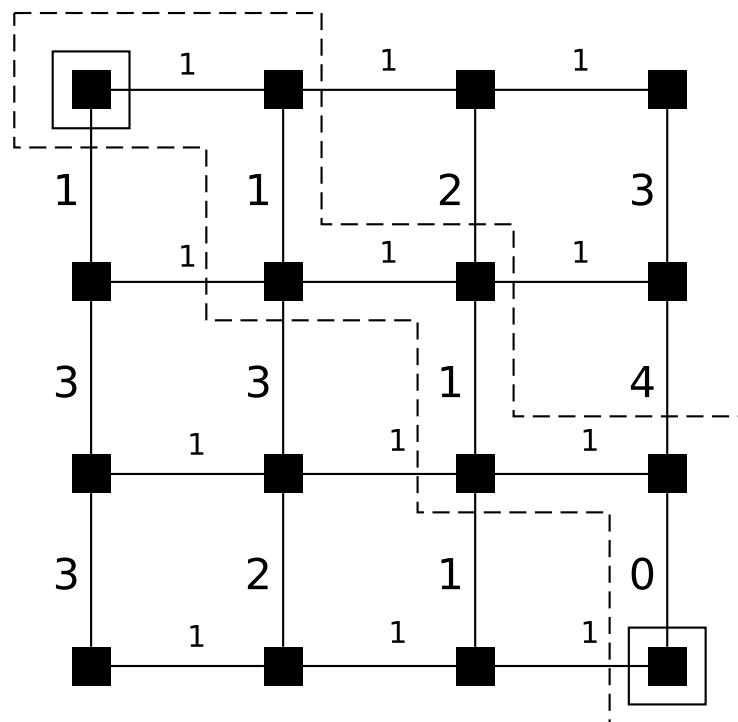


Figure 1: Street Grid and possible shortest path for sample input

#3: Strong Passwords for Everyone

The Problem

You are a system administrator at a midwestern Catholic University in northern Indiana. Due to recent security breaches by miscreants in a band of security crackers labeling themselves “ACM / Computer Club,” you have decided to enforce a strict password policy. All passwords must meet a set of criteria, and any that do not should be rightly rejected. You will write a program to verify passwords against these criteria.

A recent paper in the Journal of Interesting Security Solutions defines the strength of a password as the number of Security-Enhancing Characters – that is, any character that is not a-z or A-Z – divided by the number of alphabetic characters (a-z, A-Z) plus 1, multiplied by the square of the length of the password.

Given a list of passwords as input, compute the strength of each password and then print only the strongest password in the set.

Sample Input

The program input will consist of an integer n on its own line, followed by n passwords each on their own line. Multiple of these sets may be given; process each independently as described above. The sequence will terminate with a set of zero passwords.

```
3
myeasy password
helloworld!!123
aZ1#12)(*crackthis!!
5
asdf
myeasy password
easy passwordaa
!@#a
!@#!a1
0
```

Sample Output

```
aZ1#12)(*crackthis!!
!@#!a1
```

#4: Programming Contest Scoring – A Little Help?

The Problem

The organizers of this contest ran out of time and could not complete the contest-administration software for today, despite pulling a series of all-nighters. We have all of the problem-judging code working perfectly, so we can tell you whether you have solved a problem correctly and exactly when you solved it. However, we cannot compute your score and rank you among all the teams. If we are to award any prizes at all this afternoon, we need your help.

The rules of scoring are simple: first, teams are sorted by number of problems solved correctly. Then, to break ties between teams with equal numbers of problems, we sum up the time elapsed since the start of contest (in minutes) at which each completed problem was submitted. So, for example, if you submit problem one at 45 minutes into the contest, and problem two at 50 minutes into the contest, your time-sum is 95; if another team with two problems correct has a time-sum of 93, they beat you. Incorrect problem submissions impose a 10-minute penalty per submission, but only if the problem is eventually solved correctly. So you can submit incorrect solutions for problem 3 ten times if you wish; this does not hurt you until you submit a correct solution, at which point you have 100 penalty minutes plus the submission time added to your time-sum. (But you still jump ahead because you are now among the teams with three problems completed rather than only two.)

Assume that a contest has exactly six problems.

Given a log of the contest that records all problem submissions, your task is to produce the final contest ranking. And no cheating, please – an unfair scoring program is an incorrect scoring program!

Sample Input

The input consists of multiple contest logs. Each log starts with an integer n on its own line, indicating the total number of submissions, and then t on its own line, indicating the number of teams (labeled 0 through $t - 1$). Each following line has the format `team <space> problem <space> time <space> correct?` where “team” is the team number, “problem” is the problem number (0 through 5), “time” is time elapsed since start, in minutes, and “correct?” is 0 or 1 for incorrect or correct. The input will end with $n = 0$ and $t = 0$.

```
10
3
0 0 10 1
1 0 15 1
2 1 15 0
2 1 20 1
1 5 32 1
2 5 33 1
```

```
0 2 33 1
2 4 40 1
0 5 45 1
0 3 52 0
```

```
0
0
```

Sample Output

Produce t lines, one per team. Each line should consist of: team number; number of problems solved correctly; and time total including penalties. These three numbers per line should be separated by single spaces.

The output lines should be sorted by team ranking, highest first (where ranking is determined first by number of problems solved, then by time total (lowest first), as specified above.

```
0 3 88
2 3 103
1 2 47
```

#5: Optimal Haiku

The Problem

*optimal haiku
produced by the computer
are quite terrible*

Despite the sad truth – that computers cannot yet write good poetry – we will try. Your friends, trying to develop a literary side in you, their token computer-nerd friend, have encouraged you to write some poetry. Haiku is the easiest poetry of all: the only constraint is that there are three lines, the first with 5 syllables, the second with 7, and the third with 5. Many haiku are creative and make deep and insightful statements despite their brevity. Our haiku will do no such thing.

You will receive a list of words and you must arrange them into haiku form. You can count syllables by looking for consonants and vowels; we define a syllable, for simplicity, to be a consonant followed by one or more vowels, OR one or more vowels starting a word. So “hello” has two syllables, since the consonant ‘h’ is followed by vowel ‘e’ and the second ‘l’ is followed by ‘o’, and “eagle” also has two syllables since two vowels start the word and the word ends with a consonant ‘l’ followed by a vowel ‘e’. Not all English words can be dissected by these simple rules, of course, but we will only provide words that can be.

Given a stream of words, your only job is to determine where the line splits will be. We guarantee that there will be no need to split multisyllabic words across lines. So we may provide you with two 2-syllable words and a one-syllable word, to form the first line, and then a seven-syllable word, for the second line. But we will never start you off with three 2-syllable words, since you can’t make any split that will give you a 5-syllable first line.

Sample Input

Input will consist of an integer n followed by n words.

```
18  
hello world program blinking cursor on the screen waiting for  
input the loop keeps looping forever and forever
```

Sample Output

Output the words given you, separated into lines in the haiku pattern of 5-7-5. We want a continuous stream of haiku, so after the first 5-7-5, start another 5-7-5, etc. It is alright to end in the middle of a haiku when the words run out. Separate haiku stanzas by a blank line.

```
hello world program  
blinking cursor on the screen
```

waiting for input

the loop keeps looping
forever and forever

#6: Binary, by Nary an Engineer

The Problem

It turns out that very few engineers are competent at converting numbers to binary any more. Aside from being an excellent setup for the title pun, this situation is extremely troubling and necessitates immediate action. All Notre Dame engineers will now have four hours per day of practice in binary arithmetic of all sorts: conversion to and from decimal, addition, multiplication, taking square and cube roots... it will be brutal but it will be good for the souls of all.

As part of the training system, you are to write a program that converts decimal integers to binary. This may seem a simple task (and really, it is), but do not dare to underestimate the importance of your task. A job well done could mean the difference between a failed generation of engineers (and thus, the true leaders of the world), and a stream of well-trained engineers so sharp that all the world's problems will be solved adequately in fifty years. Imagine the possibilities.

Sample Input

Input will consist of a number n , followed by n numbers which you are to print out in binary.

```
5
2
16
256
63
30
```

Sample Output

Print the binary numbers most-significant-bit first, one per line.

```
10
10000
100000000
1111111
11110
```