Problems for Op 2021

By Dr. Robert W. Hasker
with the assistance of the MSOE Competitive Programming Teams
Friday 19 November 2021

Notes: Unless a problem specifies otherwise, you may assume that all user inputs are valid. The precise wording of your output is not critical as long as it is completely clear. There is generally no time limit on solutions, but if a solution runs far longer than deemed reasonable it will be terminated and marked as incorrect.

1. The End of the Word (10 points)

Prompt the user for a letter and a sentence, and print the first word in the sentence that ends with that letter. For this problem, words are separated by spaces. All input will be lower case letters and there will be *no* punctuation (including periods). If no word ends with the target letter, print "not found". You can assume the sentence is on a single line. For example, if the target letter is 'g' and the sentence is the Terry Pratchett quote

> coffee is a way of stealing time that should by rights belong to your older self

then the program should print

> stealing

2. Cup by Cup (10 points)

When a user puts a pod into a particular coffee maker, the machine then takes water from a reservoir to fill 4 oz, 6 oz, or 8 oz cups of coffee. Because not all of the water goes through the pod, one ounce of water is left in the pod when done. For example, a 6 oz cup of coffee requires 7 ounces of water. In this problem you will figure out how many cups of coffee can be made given a particular sequence of requests. Assume the reservoir must be refilled whenever there is (strictly) less than 10 ounces of water remaining. This ensures there is enough for one full cup with a bit left over. Your program will predict when the reservoir must be refilled given a particular size and a particular sequence of mugs of different sizes. The output is the number of cups made before the reservoir must be refilled. For example, if the input is

> 34
> 4 6 6 4 4 8 6

the output should be

> Filled 5 cups.

since 5 + 7 + 7 + 5 + 5 = 29, leaving just 5 ounces left. You can assume the user enters an integer capacity followed by enough cups (in legal sizes) to ensure at least one refill.

3. Twisted Pairs (10 points)

Read a sentence and display the positions of the first pair of letters that occurs later in the sentence in the reverse order. The pair of letters can be in the same word or any other word, but they must be adjacent, they cannot overlap, and they cannot be the same letter. The output is the positions (starting at 1) of the two pairs. In this problem, sentences have capital letters and end with a period. Ignore case when looking for a match. For example, if the sentence is

Yay, let's play.

then the output would be

Twisted pairs are at 1 and 14.

since "Ya" at the start matches "ay" in play. Another example is

All walls are level.

which has twisted pairs at positions 15 and 18 (le and el).

The sentence will be on a single line. When there is more than one twisted pair in the sentence, display the first complete pair found. If no twisted pair is found, print "No twisted pairs."

4. Lost Lumens (20 points)

In science you learn that light follows the "inverse square law": if the distance to an object is doubled, then the amount of light that reaches it (say, from a bulb) is 25%. That is, if you ignore loss to the atmosphere and assume the light is a point source, then the intensity at distance d will be $1/d^2$ of the intensity at the light. For various screen sizes and distances from the lens, use this formula to compute how much less light reaches the audience when they look at the corner of a movie screen rather than its center. Your program is to prompt for and read the following data (in the following order): the distance to the center of the screen, the height of the screen, and the width of the screen. Note that no screen reflects 100% of the light that falls on it; assume that only 90% of the light is reflected back to the viewer, that the viewer is at the same location as the projector lens, and the viewer's eye is perpendicular to the center of the screen. All distances are in meters. Print the resulting loss at the corners as a percentage rounded to four digits after the decimal. (Generally one would not use this precision, but this program could be used to determine changes in projector lenses.) For example, if the distance is 25 meters and the screen is 5 meters high and 8 meters wide, then the loss is 6.7571%.

5. Balanced Lists (20 points)

Define a list of numbers as "balanced" if the mean value (the average) is some number n. Write a program which reads a list of numbers, where each number is between 0 and 100 (inclusive), and prints the maximum number of disjoint sublists which are balanced about a target n. The program will read how many elements are in the list followed by the target mean and the list of values. It will then print the balanced sublists followed by the (maximum) count of sublists which are balanced. If there is more than one way to achieve the maximum number of sublists, just print any one of the ways.

For example, if the input is

11 5

1 3 1 15 8 5 9 1 14 7 3

then the output is

Balanced sublists:
1 3 1 15
5
9 1
7 3
Count: 4

You can assume the maximum list length is 1000 and that both the target value and the values are all integers. Note that you need consider only consecutive numbers when finding sublists, that there is no overlap between sublists, and that the mean of each sublist must match the target mean without rounding.

6. Stick Stacks (20 points)

In a particular game, you start with a non-empty pile of n sticks with a goal of getting to x sticks in a minimum number of moves. Each move has two choices: add a single stick or double the number of sticks you already have. The output of your program will be the number of moves it takes to get to the target number. You must reach the target exactly; having too many sticks is as wrong as not having enough. Prompt for the number of sticks in the initial pile first, then for the target number of sticks.

7. Scrambled Words (40 points)

For this problem you are given a non-empty list of words, one word per line. A word "covers" another word in the list if all of the letters in the second word appear in the first. If a word has duplicate letters, that letter must appear at least that many times in the covering word. For example, "correct" covers "core" and "rector" but not "error". The program read the list of words from standard input and then must print the word that covers the most other words in the input list along with how many words it covers. If two words cover the same number of words, then report the first on the list. Note that all words cover at least one word, itself. There will be at most 2000 words. For example, if the word list is

```
art
car
cart
ear
earn
trac
near
rare
rat
tar
```

then the output is

```
cart covers 6 words
```

A dictionary is available so you can create your own 2000-word file. Expect your solution to take at least 5-10 seconds on 2000-word files, and we will allow up to a minute.

8. Busy Builder (40 points)

In this problem you will determine the maximum profit from building various products using an inventory of supplies. For example, suppose your inventory consists of

- 10 legs
- 20 screws
- 2 tabletops
- 6 castors

and a recipe for a dining table requires 4 legs, 8 screws, 1 tabletop, and 4 castors; you could then use this supply inventory to build one dining table.

The program reads from standard input. The input is in two sections: inventory items (product on hand) and recipes. Each inventory item is on a separate line with the quantity of the item first followed by its name. The name can be one or more words. Each recipe starts with the word 'recipe' followed by the value of the item (once built) and a name. The items needed to complete that recipe are then listed, one per line, after the recipe header. Each line consists of a number followed by the name of the item; this name will be matched against the items in the inventory. Generally, the item names will be singular, but there is no need to check this.

The program must determine how many of which recipes can be built, maximizing the total value of all of the items built. The output will be a list of recipe names in alphabetical order with the quantity of each. If no recipe can be built, print "Cannot make any recipes." For example, if the input is

```
10 long wooden leg
4 short wooden leg
8 dowl rod
5 chair back
3 chair seat
20 long screw
2 table top
16 castor
recipe 300 dining table
  4 long wooden leg
  8 long screw
  1 table top
  4 castor
recipe 700 fancy chair
  2 long wooden leg
  2 short wooden leg
  4 dowl rod
  1 chair back
  1 chair seat
```

then the output should be

```
Maximum profit: 1700
Recipes constructed:
 1 dining table
 2 fancy chair
```

This case is available in the file problem8.txt. Note an alternative production is two tables and one chair, but the chairs provide more profit. Your solution must complete each problem within a few minutes or it will be judged as incorrect. However, we will not provide inventories that can build more than a dozen products.

9. Alien Takeover (40 points)

You are working for a fleet from outside our solar system that is attempting to subdue a part of Earth. It has been determined that the most effective way to subdue the population is to seed it with a highly infectious virus. This virus will infect anyone else who passes within 3 feet over the course of one week. At that time the person becomes bed-ridden for a month. Resources are scarce, even for virus-toting aliens, so they must determine a minimum number of people to infect (within a population) to ensure all are bed-ridden when they land. As a member of the invasion fleet, you have been asked to write a program which computes the number to infect. Your program will read the number of people in the population (from standard input) followed by lists of frequent contacts for each person. Note that if person A contacts person B, then B also contacts A, but there is no guarantee that B's list will include A. It is also possible that if A contacts person C, there is no list of contacts for C. The contact lists define groups of people who all are in frequent contact with each other, so only one person in each group needs to be infected. The output of the program is the number of people that must be infected to ensure the entire population becomes infected.

Your implementation will be used on small trial problems only. For this problem, you can assume there will be no more than 200 distinct names.

For example, if the input is

```
6
Ann Doug
Brenda Zoe
Charles
Doug Charles
Xander
Zoe Brenda
```

then the output would be

```
3
```

since Ann, Charles, and Doug form one group, Brenda and Zoe form another, and Xander meets no one. Case is not significant in names.