Problems for OpComp 2023

By Dr. Robert W. Hasker
Friday 17 November 2023
Copyright © 2023

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. Racing ASCII (10 points)

Two ASCII vehicles are racing each other, and your job is to see which wins. A bus and car are racing, dropping little pebbles behind. The race is to determine who travels the furthest in 1 millisecond. For example, the bus wins the following race:

```
..................bus
..............car
```

In this race, the two tie:

```
....................................car
....................................bus
```

If a vehicle crashes, it spills pebbles in front of it, and then that vehicle is disqualified and the other wins automatically. If they *both* crash, the result is a scratch.

Read the two lines from the console. You can assume the lines are all text with no spaces. One line contains “car”, the other “bus” (in either order), and the rest of each line is periods. No vehicle can drop more than a thousand pebbles in a millisecond. The output is “car wins”, “bus wins”, “tie”, or “scratch”.

2. Attractive Alliteration (10 points)

You heard on TikTok that due to the structure of English, it is impossible to have more than 4 words in a row that start with the same letter. You are curious, so you decide to write a program that finds and prints the longest alliteration in a book. For this problem, alliteration is defined as just the same letter on successive words rather than the more expansive version that allows some words. For example,

> Multiple Marvins in Indonesia say Sibyl sells sea shells by the sea shore. $$

Would result in printing

say Sibyl sells sea shells

Note that matching is case-insensitive. The end of the input is marked by $$, and the input may be on multiple lines. There is at least one word in the input (other than $$), all words are separated by one or more spaces or lines, the character of each word is a letter, $$ appears only at the end of the input, and there is always at least a space before the $$ (or it is on its own line). The output words are separated by a single space. If there are two sequences that are the same length, print the first.

3. Tubular (10 points)

Your friend sells posters in cardboard tubes and has asked you to build a machine that wraps the tubes (with posters in them) in paper for shipping as gifts. The machine selects a tube and cuts it to the proper length, then it wraps the tube in gift wrap. Tubes come in three diameters: 5 cm, 8 cm, and 12 cm. The 5 cm diameter works for posters up to 1 meter in the longest side, 8 cm for posters up to 2 meters, and 12 cm for all other posters. Posters sizes are specified in centimeters, and the machine rotates the poster so the tube covers the shortest side. The machine then cuts each tube to that length, adding an additional two cm (1 cm at each end) to allow for endcaps. The gift wrap is cut so it overlaps the tube by 10% around the circumference and leaves an extra section of wrapping paper at each end. That extra section is always the diameter of the tube being wrapped, so a 5 cm tube would require an extra 10 cm of wrapping paper.

Write a program that reads a list of paper sizes in the form

NxM

where N and M are both centimeters. The program is to write the minimum amount of wrapping paper needed in square meters. Note the actual amount of wrapping paper will likely be larger because papers come in preset widths, but that adjustment will be done by your supplier.

The first input is the number of posters, and the remaining input is a list of sizes, one for each poster, each on its own line. Note the x between the two numbers; there are no spaces in the line. For example, an input might be

5
23x41
80x12
240x100
10x10
900x650

You can assume the input is formatted properly (it is generated by another program) and that each order will have at least one poster. Print the result with at least 2 digits after the decimal. The above order would require 3.47 square meters of paper.

4. Digging for Lost Treasure (20 points)

A suspected pirate retired in northern Wisconsin in the town of Peshtigo. You have done careful research and found several hand-written maps giving possible locations of a small chest of Liberty gold coins buried deep in a field or under the house. Each map has a series of steps in eight different compass directions: N, NE, E, SE, S, SW, W, and NW. The directions assume you start at the front door; fortunately there are enough pictures and survey maps to determine that location. Where to dig is marked by an X at the end of the list. For example,

10N, 5E, 2Sw, 3e, X

These directions indicate a distance to walk from the pirate's front door; you determine the location of the door based on old pictures of the site and house and a town survey. These directions are as formatted above:

- All directions are on a single line, separated by commas.
- The last direction is X because it marks the spot.
- Each item in the list (other than the last) is a positive integer followed by a direction.
- Directions and the X can be upper or lower case.

A challenge is that while the pirate's height is documented in arrest records, they are inconsistent and so the stride length must be estimated. Given the range, we know each step is between 62 and 68 centimeters. Because the pirate is a practiced navigator with a good compass, you can trust that he is precise about what direction he is facing when stepping, and you can assume he steps the same distance on each pace. However, your dig point must account for the unknown step distance by computing a bounding box around it.

Using the list of directions, compute the distance and direction to the digging location relative to the location of the front door. Express the distance as a number of meters north (with negative meaning move south instead) and a number of meters east (with negative meaning move west instead). For each supplied map, report the area to be dug. One corner will be determined by using the minimum stride distance, the other using the maximum stride distance. For example, the report for the above example would be

Dig in the area from (5.32 N, 4.08 E) to (5.84 N, 4.48 E)

The first point in this report assumes the shortest stride length, the second the longest stride length.

5. 1337-plate (20 points)

A state has a requirement that license plates consist of 3 letters followed by 3 digits. You decide to build a website to appeal to the tech crowd by identifying words that can be written using the Leet alphabet in which letters are replaced by digits. For example, 1337 spells "leet" using this

alphabet. Write a program which takes a list of words, identify which have three-character Leet endings, and print that list (with the Leet replacements for the last three letters) using upper case letters and in alphabetical order. Use the following table for substitutions:

- 0 = O
- 1 = I or L
- 2 = Z
- 3 = E, M, or W
- 4 = A
- 5 = S
- 6 = G
- 7 = T
- 8 = B or X
- 9 = J or P

There are additional substitutions in Leet, but those substitutions often change the word size. For this problem, the output word size is always the same as the input word size, so words that are not 6 letters in length are ignored.

As an example, if the input is

dazzle
cinemas
circle
days
impose
cipher
cinema

then the output would be

CIN334
DAZ213
IMP053

6. Rule 184 (20 points)

Rule 184 is a cellular automaton rule often applied to one-dimensional models. For this problem, we will be modelling traffic flow, and for purposes of this explanation, we will assume traffic flows from left to right. The “road” is simply an array, with each element holding a ‘c’ if it is occupied by a car and a ‘.’ if it is empty. The cars will move to the right if there is an empty spot but stay in place if there is already a car at that place. A car at the end of the road (the rightmost position) is always carried away, possibly to that great parking lot in the sky.

Implement this Rule 184 model of traffic flow. Your program should prompt the user for how long the road is (total number of cells) and a string of '.'s and 'c's representing the initial condition of the road. Run the model until all cars have left the road, counting each step as a single tick. Report the number of ticks required to clear the road. You must support a road with at least 1000 cells, and a string of initial conditions at least 50 characters long. If the user enters a string that contains any character but '.' or 'c', indicate the error to the user and exit.

For example, if the input is

```
8
c.cc..c
```

then it takes 9 ticks to clear the traffic.

7. Stuffing Recipes (40 points)

You have a friend who is cooking for a large number of people, and they need your help planning a filling meal. Your program reads a list of recipes and items in the pantry and prints a list of dishes to make. A sample recipe is

```
omelet:
  3 count egg
  0.03 l butter
  40 g cheese
  5 g salt
  435 calories
```

The first line is a name (as a single word) followed by a colon. The next several lines are ingredients, each with a quantity, a unit of measurement (counts, liters, or grams), and a name. For example, this recipe needs 3 eggs, 0.03 liters of butter, 40 grams of cheese, and 5 grams of salt. The last line is always a number of calories. You can assume the recipe is always in this format; there is always a name, the items are indented, and the end of the recipe is a number of calories. The list of recipes is terminated by the word "END", alone on a line. This is followed by a list of pantry items; for example:

```
bread_slice 30 count
salt 500 g
butter 1 l
cheese 500 g
egg 11 count
sugar 1000 g
corn 2000 g
```

A file containing this recipe, one other, and this list of pantry items is available as `breakfast.txt`.

The program is to find all recipes that can be cooked with the products available in the pantry (which includes some items that would be in a fridge – we are treating it all as the pantry for the purposes of this program) and pick the one that generates the most total calories. For example, the above pantry can support 3 omelets. Then remove the ingredients used for that recipe,

remembering to factor in the quantity. For example, cooking three omelets removes 9 eggs from the pantry. Then find the next recipe that delivers the largest total calories, and remove its items. Repeat this until there are no recipes that can be cooked. When you are done, print the recipe names, the quantities, and the total calories for each recipe:

```
omelet: 3 servings for 1305 calories
```

If you add the line

```
syrup 0.2 l
```

to the pantry in breakfast.txt, your program should print

```
omelet: 3 servings for 1305 calories
french_toast: 2 servings for 1100 calories
```

Some details:

- All recipe names and ingredients are in lower case.
- You can assume well-formed input; recipes and pantry items follow the specified format.
- Print the items in order by total calories.

This way of solving the problem does not always maximize the total calories in the meal, but it will generally do well. The basic strategy is to find the recipe that, when made at the maximum quantity possible, generates the greatest total number of calories. The ingredients are removed from the pantry, and the process is repeated. It will be a filling meal!

8. Blocked Out (40 points)

A challenge to theater goers is that travel companies will purchase large blocks of tickets close to the stage so they can bundle them with hotel rooms and air fares for people who are interested in travelling to an event. Your city has passed ordinances against this, so a theater has employed you to identify possible offenders.

The input to your program is a chart showing seats. Each seat is a letter followed by a number, where the letter indicates the row (starting from closest to the stage, the bottom) and the number represents the column (starting from the leftmost seat in the row; the position of the leftmost seat varies by row). Seats that are open are marked with a dot (‘`.`’), and seats that don’t exist are marked with a dash (‘`-`’). Seats might not exist because of columns, walls, or other equipment. Any other (printable) character indicates who purchased the seat. Different purchasers have different identifying characters; this allows the theater to anonymize the purchasers so who is purchasing does not influence the algorithm.

The ordinance identifies a block purchaser as the same purchaser buying seats in an area with 8 or more adjacent seats, where a two seats are adjacent if they are purchased by the same person and next to each other in either the same row or column. Note this says nothing about purchasing multiple tickets at non-adjacent locations. Your program is to identify all potential blocks and list them in order by the number of seats in each block. If two blocks are the same size, it does not matter which is listed first. For each offender, write the message

```
Purchaser x bought y seats starting at uv
```

where x is the identifier for the purchaser, y is a count, u is the row closest to the stage, and v is the seat number (counting from 1) in row v that is furthest left. For example, if the input is

```
. 1 1 - . . a a
0 1 1 1 1 c a a
1 1 1 - . 2 b b
- b b b b b b -
c c c c c c c -
2 3 a a b a b c
```

Then the program would print

```
Purchaser 1 bought 9 seats starting at D1
Purchaser b bought 8 seats starting at C2
```

You can assume that each row is formatted as shown here: seats are separated by a single space and each row has the same number of characters. If someone purchases multiple, separate blocks, each of which is 8 or more seats, then there will be multiple reports for that person.

9. Scrabble Scramble (40 points)

Scrabble is a game in which players place words horizontally and vertically on a board so that each new word shares at least one letter with any existing words. You are being given a target word and a board and are to identify all places the word can be placed on the board. The placement must be legal: it must cross at least one other word, any time it crosses a word the letters must match, and there cannot be any letters before, after, or beside the new word. (This is a simplification of the real rules: new letters can be placed next to existing letters in the game as long as the new letters always create valid words. We are ignoring such exceptions.)

The board is given as N by M grid with '`.`' used to indicate an empty square and a lower case letter for a square that is filled. The target word is all lower case. For each position found, indicate the starting point; the position of the first character (where the upper left corner is (1, 1)) and whether the word would be placed horizontally or vertically at that point. You can list the possible places in any order. For example, given the input

```
cape
........
........
...a....
...p....
...p....
..blame.
...e....
........
```

the program would print something like

```
Possible placements for cape:
 - horizontally at row 3, col 3
 - horizontally at row 4, col 2
 - vertically at row 3, col 7
```

If there is no placement, write “No placement found for [word]”.

Notes:

- You may assume the target word never has repeated letters. This simplifies the problem a bit since each letter on the board can match just one position in the target.
- There is no requirement to check words against a dictionary; you can assume all occupied spaces contain valid words and the input word is valid.
- The board is rectangular with the same number of spaces in each row.
- Boards will be between 5 by 5 and 25 by 25, but there is no requirement that both dimensions have the same size.