COMP 1405B
Fall 2019 – Practice Problems #1

Objectives

- Practice problem solving
- Practice applying computational thinking principles to problems
- Practice developing algorithmic solutions to problems

The following problems deal with computational thinking, problem solving, and algorithm development. Many of these problems are challenging and involve concepts we will not cover from a programming perspective for quite some time in the course. You are not expected to be able to solve all these problems at this point (or even by the end of the course!). However, it is possible to design algorithms to solve problems that you are incapable of programming. Remember, an algorithm is different from an implementation/program. If you are unsure about how you are phrasing steps or formatting your algorithms, which is a very common problem when getting started, share your solution(s) to get feedback.

If you cannot solve a problem completely, a good start is to think about what information/data you need to solve the problem, as well as what steps you might take to process that data in order to produce a result. Problems that you struggle with will be good to come back to later in the course, once we have covered more material. Many of them will make a lot more sense once you know more about programming in general.

Many of these problems do not have a single correct answer. Sharing your ideas with others in the class will give us all a better idea of the different aspects that can be considered and the many different approaches that could be applied. We can also discuss the different pros and cons of different solutions.

Several of these problems involve lists of data, which are a more complex programming concept that we won't discuss until after the Fall break. If you want to start working on problems involving lists now, the following information may be helpful. Think of a list as an ordered collection. You can refer to specific items in the list by their index (e.g., 1st, 2nd, x^{th}) and can also refer to the length of the list. You can add and remove items from a list. You can check if an item is in a list. These are ideas that you can include in your algorithms using regular English that will allow you to work with lists of data.

Problem 1 (Finding the Right Numbers)

The sum of 2 consecutive odd numbers is 44. What are the two numbers? Assuming that you did not just guess the numbers exactly correct immediately, write out the steps that you used to solve the problem. Now assume you are given some number *x*. Write out a general algorithm for finding the 2 consecutive odd numbers that sum to *x* or to determine if no solution exists.

Problem 2 (Finding the Largest Number(s))

Assume you are given a list of numbers, such as [8, 3, 5, 1, 2, 9, 7, 6, 4]. Explain in detail the steps you would take to find the largest number. Remember to be specific about things like what numbers you are considering at each step and what actions you are taking (i.e., be much more specific than 'look through the list and find the largest number'). If your solution is not already in the form of an algorithm, re-write it to use proper pseudocode or create a flowchart. What about the related problem of finding the largest and second largest numbers? What about a general solution that allows you to find the X largest numbers, where X is some number?

Problem 3 (Rock Paper Scissors)

Create pseudocode or a flowchart to model a single round of a game of rock/paper/scissors. You should get a choice from each player and then compare the values to determine who is the winner of the round. Note that when creating an if ___ then ___ step in your algorithm, the first blank can be anything that could be true or false. For example, "if player1 selects rock and player2 selects paper".

For an extra challenge, expand your solution to repeat the game for a number of rounds. The game can stop after a set number of rounds or once one of the players has won a specified amount of games (i.e., a 'best out of seven' match). Approaching the problem this way is a good example of decomposition. You solve the easier problem of a single round first, and that solution then makes up a part of the more complex final solution.

Problem 4 (Classifying Survey Data)

We are collecting survey data for some research. The participants are asked to enter their level of interest about some topic on a scale from 1-10. We want to classify each participant as either very interested, moderately interested, or disinterested. Create an algorithm or flowchart to do so. Does this question give you all of the information you need to solve the problem? Sometimes you must make assumptions or decisions about how you will handle things.

Problem 5 (Highest/Lowest Grade)

Create an algorithm or flowchart to read numbers representing grades from a user and print out the highest and lowest grade when they are finished.

Problem 6 (Autonomous Vehicles)

You are creating a controller for a self-driving car. The controller must decide if the car should proceed through an intersection or not. Assume that the information you are given includes: the current speed of the car, the distance of the car to the intersection, and the current color of the light. You can use the decision rules below if you want, or create your own rules for the car. You can also add more information into the problem to create a more complex solution.

Basic rules to govern a car's decision:

1. Go when the light is green.
2. When the light is yellow, Go if the car can reach the intersection within 5 seconds. Otherwise, the car must Stop.
3. When the light is red, Go if the car can reach the intersection in 2 seconds. Otherwise, the car must Stop.
4. If the light is any color other than green, red, or yellow, the car must Stop.

Problem 7 (Bank Machine)

Write an algorithm for a bank machine. What information do you need to get from the person using the machine? How will you process this information? How will you decide what to output to the person using the machine (e.g., was their transaction successful?). You will likely have to make some assumptions about what data would need to exist within the system (e.g., account numbers, PIN numbers, balances, etc.). Applying decomposition to this problem would also be a good start – consider a bank machine that can only perform a single operation (e.g., take a single deposit from a single person), then build up from there.

Problem 8 (Card Games)

Create an algorithm or flowchart for a basic 2-player card game. Some good examples of games could be: war, go fish, crazy 8s. Again, decomposition will be important here.

Start by thinking about some basic parts of the game. Also, we do not have to duplicate the EXACT rules of the game. If you are finding some of the rules hard to deal with, you can ignore them and possible consider them later.

Problem 9 (Getting Rich)

You are given an ordered list, from earliest to latest of stock prices that occurred throughout the day. You want to create a program that determines the largest profit you could have attained by buying a single stock at one price and selling it at a later price. Create an algorithm or flowchart to find this value. This one may be difficult – think about how you determine profit, as well as all of the possible buy/sell combinations. It may also be beneficial to create an example problem on paper and work through how you would solve the problem by hand (this is a good idea for many problems!).

Problem 10 (Games)

Think about a game you play - it can be a computer/mobile game, board game, etc.. Depending on how complex the game is, you may need to select one aspect of the game (e.g., your health bar, the game board, player movement, etc.). Think about the information that you would need to represent this part of the game. What sorts of rules would be applied to manipulate this information? This would constitute a part of the algorithm for the entire game (decomposition!). Try to create an algorithm or flowchart to mimic part of the game.

Problem 11 (Finding Pictures)

Assume a user’s social network profile has a list of pictures. Each picture has a list of tags. Given a search term (e.g., ‘puppy’), we need to retrieve a list of pictures that are tagged with that term. How could you do this? If you have created an algorithm to solve the single keyword search problem, consider writing an algorithm that takes a list of keywords. Could you order the pictures based on how many keywords they match? If this one is too hard, you can also consider finding only one picture that has the most keyword matches. You can create an artificial social network profile on paper and apply your algorithm with different keywords to see if it works.

Problem 12 (Finding Friends)

Many social networks will suggest new friends to you by trying to find people that you may know. Think about how these suggestions could be made and try to create a basic algorithm. You can make assumptions about what information is available to you. For example, you could assume that each user has a list of friends, a list of interests, etc..

Problem 13 (Movie Recommendations)

Netflix, YouTube, and other streaming sites recommend videos that they think you will like. Can you think of a way this could be accomplished? Think about what information they need to remember about you and about each video. Can you think of other information they might use? Can you come up with a recommendation algorithm?

Problem 14 (Create Your Own Problem)

Create your own problem! Think about something you do or interact with on a regular basis. If you were going to create an algorithm, how would you do so? You don't need to come up with an exact solution. It is beneficial just to think about what types of data, rules, etc., that you might need. You can also apply decomposition to identify more basic elements of the problem that are easier to solve. Discuss it with others and they may have more ideas. How complex of a problem can we all solve together?