CSC 495 – Problem Set 4 – Due Tuesday, February 7

Problem 4.R1: Best Range

Required Problem
Points: 50 points

Background

Consider a list of integers (positive and negative), and you are asked to find the part of that list that has the largest sum. More specifically, given n integers $x_1, x_2, \ldots, x_n$, find s (start) and e (end) such that

$$\sum_{i=s}^{e} x_i$$

is as large as possible. Note that it's possible for individual terms in this sum to be negative, as long as the overall sum is as large as possible. For example, in the sequence of numbers below, the sublist with the largest sum is marked with the box (the sum of that sublist is 23).

-4 2 -3 | 3 6 -4 -2 4 7 9 | -1

The Problem to Solve

You will be given n integers, and must find the maximum sum, and the start and end positions of this maximizing range. You are guaranteed that n is at most 1,000,000, and the sum of all sublists fits in a regular 32-bit signed int.

Hints, Techniques, and Elegance Considerations

There are multiple ways of solving this problem, and there is a reasonable solution that very easily solves this problem in $\Theta(n^2)$ time. Given the simplicity of writing a correct solution, the goal here is elegance and demonstration of the theme for this week: loop invariants and reasoning about correctness of programs. Elegance will have a higher weight than normal for this problem/assignment. There is an elegant solution that uses only a constant amount of memory, meaning a few primitive-type variables — no arrays, vectors, or anything that takes more than a few bytes of memory. The easiest way to develop this solution is to think about maintaining a loop invariant, and use that to guide development of your algorithm. To get all of the elegance points, you must include a specifically stated and justified loop invariant in your code, and explain how that relates to the strategy and winning the game.

Input and Output

The input will consist of an integer `n`, followed by `n` lines containing a single integer each. You should print out three numbers on one line, separated by spaces: the maximum possible sum, the starting and ending position of the range (where positions range from 1 to `n`). The sample below is the same problem illustrated in the "Background" section.

Sample Input

```
11
-4
2
-3
3
6
-4
-2
4
7
9
-1
```

Sample Output

```
23 4 10
```

Problem 4.R2: That's the Last Draw

Required Problem
Points: 50 points

Background

Consider the following two-player game: The game starts with a pile of n cards, and the players take turns drawing anywhere from 1 to m cards. The player to draw the last card wins. So in a game between Alice and Bob with $n = 8$ cards and $m = 3$, it could proceed like this: Alice draws 3 cards, Bob draws 1 card, Alice draws 1 card, and Bob draws all 3 remaining cards to win the game. It turns out that the initial setup determines who wins, if both players play optimally. In other words, given m and n there is either a strategy where the first player is guaranteed to win, or it is impossible for the first player to win if the second player plays optimally.

The Problem to Solve

You are to write a program that plays this game, where your program always goes first. This program is a little different from the other ones in this class in that not all input is given up front. It is interactive, and your program must respond to inputs as they arrive. Your program must guarantee a win if the initial configuration is such that this is possible. If the initial configuration doesn't guarantee a winning strategy your program should play the game anyway — if the opposing player ever plays sub-optimally, leaving an opportunity for you to win, then you must play to guarantee a win. When you submit your program, it will be tested in various scenarios with an "opponent program," and your program is expected to respond appropriately. You are guaranteed that n and m will be no more than 1000.

Hints, Techniques, and Elegance Considerations

Once you see the strategy, this is a trivial program to write. The purpose of this problem is to stress the theme for this week: loop invariants and reasoning about correctness of programs. Elegance will play a larger role than normal for this problem/assignment, and you are required to include comments in your code that justify why your program works. In other words, include a specifically stated and justified loop invariant in your code, and explain how that relates to the strategy and winning the game.

Input and Output

Your program should start by reading the two integers `n` (the number of cards) and `m` (the maximum number of cards that a player can draw). It should then output a line of the form

`I draw 1 card, leaving` x

or

`I draw` k `cards, leaving` x

depending on which is grammatically correct (meaning $k > 1$ in the second case). When a draw (either yours or the opponents) leaves zero cards, print either "`I win`" or "`You win`", depending on who made the last draw. A sample interaction is shown below, where the opponent-supplied input is underlined.

Sample Interaction

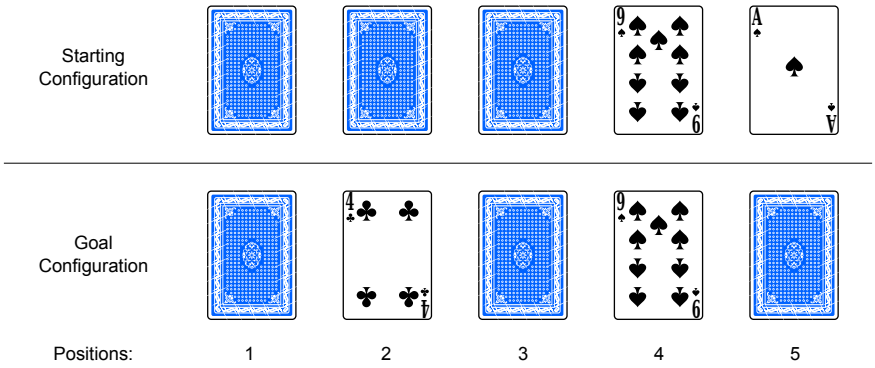
10
3
I draw 2 cards, leaving 8
3
I draw 1 card, leaving 4
1
I draw 3 cards, leaving 0
I win

Problem 4.C1: Flippin' Pairs

Challenge Problem
Ninja Points: This challenge problem is worth up to 20 base ninja points

Consider a long line of playing cards, some face-up and some face-down, that we want to turn into a given pattern of up/down cards (which we'll call the "goal pattern") by flipping cards over. If we can flip each card individually then the way to do this is obvious: every card that is in the wrong orientation (face-up when it should be face-down, or vice-versa) should be flipped over. But what if you are required to turn cards over in pairs, so that if you flip over a card at least one of its neighbors must also be flipped over. Is it always possible to turn an initial configuration into the goal pattern? And if it is possible, can you find the minimum number of flips required?

For example, consider the instance of this problem illustrated below:



To turn the starting configuration into the goal configuration, flip the cards at positions 2 and 3, then flip cards at positions 4 and 5, and finally, flip 3 and 4. This takes 3 flips, which is optimal.

The Problem to Solve

You are given n cards, where each card has a current orientation (face-up or face-down) and a goal orientation. You are to write a program that will either determine that it is impossible to reach a configuration in which all cards are in their goal orientation, or (if it is possible to reach the goal) give the minimum number of flips required. While this problem is stated as "playing cards" the input may be much larger than that toy statement — your program should be able to handle lines of up to a million cards in less than 10 seconds.

Hints and Techniques

Hint 1: Here's a powerful problem-solving technique: In problems where you are trying to turn one arbitrary configuration into another arbitrary configuration, try removing the "arbitrary" part from one side by transforming the problem. Trying to reach a particular, fixed output configuration is often easier to visualize, and if the transformation keeps the basic characteristics of the original problem then the insight you gain through this can be very valuable. In this problem, what if the problem were to turn all of the cards face down? Try to solve that problem

first, and then think about how it relates to the more difficult problem stated here. It also might help if you think of this problem in terms of flipping bits rather than cards.

Hint 2: There's a reason I'm giving this problem during our discussion of loop invariants. Try to come up with a loop invariant that is preserved by any pair-flip (it's easier to see this in the simplified problem described in "Hint 1"). Problem 6 in Column 4 of *Programming Pearls* might get you thinking about the right thing for a loop invariant. Does that loop invariant give you any clues on whether making the necessary transformation is possible with a sequence of pair flips?

Elegance Considerations

It's possible to do this with a very efficient $O(n)$ time algorithm. Once you figure that out, with a simple loop invariant, it should lead to very simple, clean, and elegant code (but document your loop invariant in a comment). Once you have the basic algorithm down, think about space usage — how can you implement this using as little memory as possible?

Input and Output

The input starts with an integer `n` that gives the number of cards. This is followed by `n` lines, one for each card, where each line contains two numbers indicating the current orientation of the card and the desired orientation of the card. Orientations are given with numbers 0 (meaning face-down) and 1 (meaning face-up). The output should be either the word "`Impossible`" or the minimum number of pair flips required to turn the current configuration into the goal configuration. Two different samples are given below — the first one is exactly the problem from the "Background" section.

Sample Input 1

```
5
0 0
0 1
0 0
1 1
1 0
```

Sample Output 1

```
3
```

Sample Input 2

```
5
0 0
0 1
0 1
1 1
1 0
```

Sample Output 2

```
Impossible
```

Problem 4.C2: Justify Yourself!

Challenge Problem
Ninja Points: This challenge problem is worth up to 20 base ninja points

This is not a programming problem — it might be the only non-programming problem we have all semester, although the thought process that this illustrates is something that should be part of your mindset whenever you write programs for non-trivial problems.

The Problem to Solve

Write up a clear and rigorous argument that the algorithm you used for Problem 4.C1 is correct. In particular, prove two things: First, if your algorithm says "`Impossible`", then it really is impossible to turn the current configuration into the goal configuration. If you found the proper loop invariant when deriving the algorithm, then this part should be pretty easy (but you have to write it up clearly!). Secondly, prove that if it is possible to turn the current configuration into the goal configuration, then the number of moves that you print is actually the minimal number of moves. Optimality is hard, if not impossible, to prove with a loop invariant, so think about other ways to reason about this. You can either write this by hand (if you're neat!) or type it up, but you should turn this in on paper in class on the due date.

Other Musings....

There are a lot of fun problems that are of the general form "find a sequence of moves to turn a starting configuration into a goal configuration," and in fact this is a common form of puzzles — that's all that Rubik's cube or most of the "Thinkfun" games are, after all. Some surprisingly simple-to-state problems like this have unsolved issues.

For example: Imagine playing cards, like we have here, but instead of laid out in a line they are stacked up as normal in a deck — but can be in any order and in any orientation (face-up or face-down). You want to put the cards in order, and all face-down, where the only operation you're allowed to use is to take some number of cards off the top of the deck, flip that stack over, and put it back on top of the deck. What is the minimum number of flips required to get the deck in the desired configuration? It turns out that this is a very hard problem, sometimes referred to as the "burnt pancake problem" (think about it... and Google it if that name doesn't make sense). Several things are unknown about this problem, and the related (non-burnt) pancake problem. An improved upper-bound on the number of flips required for the non-burnt pancake problem was proved as recently as 2007, and there have been other published results in the past 4–5 years. As an interesting historical note, this was a computer science problem that was actually studied, with some results written up and published, by the young Bill Gates[1] before he went off and started some little company or something.

[1] W. Gates and C. Papadimitriou. "Bounds for Sorting by Prefix Reversal," *Discrete Mathematics*, Vol. 27, pp. 47–57, 1979.