The 1st Problem Set

December 2, 2017

Pre-requisites

You should do the followings

1. On a separate and clean paper, you need to describe your own strategy to solve the problems below, and to justify why your strategy is effective while handling each problem
2. On a new clean paper, transform your strategy into an algorithm, using your own form to express algorithms.
3. Use the `Code ocean` (`https://codeocean.com`) platform; if necessary, you may invite me using my email address `lightsun.kim@gmail.com`.
4. Try to solve each problem within 3 hours. Thus you prepare two answer codes; one is a C code that you have made within 3 hours, and the other is a code augmented and fixed from the original code later.

1 Problem #1

Consider two polynomials $f(x)$ and $g(x)$ whose coefficients are integers. Thus, these polynomials can be written as

$$\begin{aligned} f(x) &= f_n x^n + f_{n-1} x^{n-1} + \cdots + f_1 x + f_0, \\ g(x) &= g_m x^m + g_{m-1} x^{m-1} + \cdots + g_1 x + g_0 \end{aligned}$$

where $f_n \neq 0, g_m \neq 0$ and for all $i \in \{0, 1, \ldots, n\}, j \in \{0, 1, \ldots, m\}, f_i$'s and g_i's are in $\mathbb{Z}$. Now given two integer polynomials f and g, we would like to compute the quotient polynomial $p(x) := \text{quo}(f, g) = p_u x^u + p_{u-1} x^{u-1} + \cdots + p_1 x + p_0$ and the remainder polynomial $r(x) := \text{rem}(f, g) = r_v x^v + r_{v-1} x^{v-1} + \cdots + r_1 x + r_0$. For example, let $f(x) = 3x^5 + 6$ and $g(x) = x^4 + x^3 - 3x + 9$. You can easily see that $0 \leq v < m$. Then their quotient polynomial $\text{quo}(3x^5 + 6, x^4 + x^3 - 3x + 9) = 3x$ and their remainder polynomial $\text{rem}(3x^5 + 6, x^4 + x^3 - 3x + 9) = -3x^4 + 9x^2 - 27x$. Thus one output is $p(x) = 3x - 3$ and the other output is $r(x) = 3x^3 + 9x^2 - 36x + 27$.

Language requirements. During tackling this problem, you should follow the programming rules:

- You should use an ANSI C programming language whose source code can run on `Code ocean` platform.
- Function naming: Begin with the lower character, and every parameters are strong-typed variables (i.e., do not use `void` typed variables). All functions should have a single return value; thus even if a function will return no values; you should provide `return` keyword.
- Variable naming: Begin with a type-discriminating prefix. For example, if a variable name is for an age and is with an integer type, you need to declare the variable as `int iAge;` Especially for string-type variables you are strongly recommended to use the prefix `sz`. For example, if a variable name is for a name, then `szName` is a preferable choice.

Input format.

`n m`
f_0 f_1 f_2 $\cdots$ f_n
g_0 g_1 g_2 $\cdots$ g_m

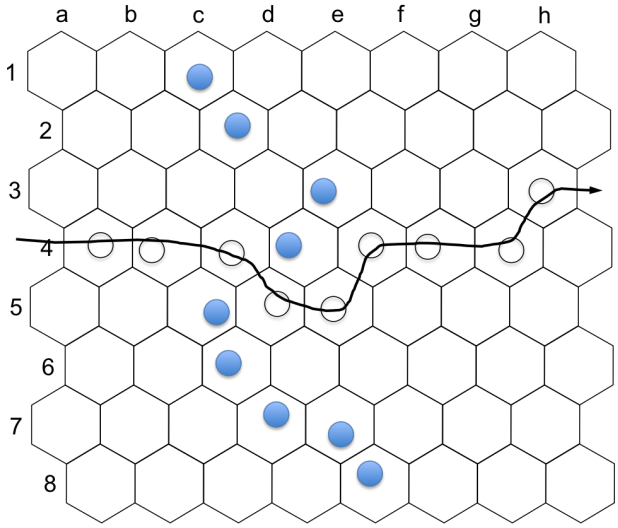
Here the input is given a text-format file, named `input.txt` and all strings are separated by the blank character.

Output format. The output should be given as a text-format file, named `output.txt` and mainly consists of two parts:

`[quotient]`
`p(x)=`p_u`x^u+`p_{u-1}`x^u-1+`$\cdots$`+`p_1`x+`p_0
`[remainder]`
`r(x)=`r_v`x^v+`r_{v-1}`x^v-1+`$\cdots$`+`r_1`x+`r_0

2 Problem #2

The game of Hex is a well-known game, invented by one of the most famous mathematician *John Nash*. In fact, he is the subject of the book as well as film "A Beautiful Mind". In this game, there are two players (say Alice and Bob). Alice plays the blue stone and Bob plays the white stone. Each player takes turns placing stones of their respective colors on an $n \times n$ hexagonal grid shown in the below. Note that one a



stone is placed, it cannot be moved. The goal of Alice is to connect the top and the bottom sides of the grid, and the goal of Bob is to connect the left and right sides of the grid, using stones of their respective colors. The top side of the grid is marked by English alphabet characters and the left side of the grid is marked by positive integers; thus any hex cell can be identified by a pair of marks. For example, the first blue stone is placed at $(1, \mathtt{c})$. We say that two cells are connected if they share an edge and both the same color stone. Therefore two blue stones placed at $(2, \mathtt{c})$ and $(3, \mathtt{e})$ are not connected.

Now design an efficient program where you can determine after each move whether Alice or Bob just won a game of Hex. Considering the Hex game above, you can see that Bob wins the game.

Language requirements. During this problem, you should follow the programming rules:

- You should use an ANSI C programming language whose source code can run on `Code ocean` platform.
- Function naming: Begin with the lower character, and every parameters are strong-typed variables (i.e., do not use `void` typed variables). All functions should have a single return value; thus even if a function will return no values; you should provide `return` keyword.
- Variable naming: Begin with a type-discriminating prefix. For example, if a variable name is for an age and is with an integer type, you need to declare the variable as `int iAge;` Especially for string-type variables you are strongly recommended to use the prefix `sz`. For example, if a variable name is for a name, then `szName` is a preferable choice.

Input format. The input is given a text-format file, named `input.txt` and all strings are separated by the blank character.

```
n=8
0 0 b 0 0 0 0 0
0 0 b 0 0 0 0 0
```

```
0 0 0 b 0 0 0 0
w w 0 b 0 0 0 0
        ...
0 0 0 0 w 0 0 0
```

Here the size of a grid is always given in the form of n = size and the character '`0`' means a blank cell, and the character '`b`' means the blue stone was placed, and the character '`w`' means the white stone was placed.

Output format. The output is the running time in milliseconds of you program. You do not need to use an output file to show your result. Instead, your code should include the fragment of C code to measure the running time.

```c
#include <time.h>
...
int
function(void)
{
    struct timespec tBegin, tEnd;
    double dElapsed = 0.0;
    clock_gettime(CLOCK_MONOTONIC_RAW, &tBegin);
    perform(); /* perform something */
    lock_gettime(CLOCK_MONOTONIC_RAW, &tEnd);
    dElapsed = (tEnd.tv_nsec - tBegin.tv_nsec) / 1000000000.0 +
               (tEnd.tv_sec - tBegin.tv_sec));
    ...
}
```

Then the function `perform` takes `dElapsed` seconds to complete its task. Of course, you may a different code to measure the running time more precisely; in fact the more precise, the better.

3 Problem #3

This problem requires that you program a code for testing whether a given integer N is prime or not. Such program is called a primality test; Among them, the most popular algorithm has been invented by Miller and Rabin. The Miller-Rabin algorithm is given as follows.

Algorithm Miller-Rabin's primality test

INPUT: an odd positive integer $N \geq 3$ and a loop counter $t \geq 1$
OUTPUT: `true` if N is prime; `false` otherwise

1. Write $N - 1 = 2^{\alpha}\beta$ such that β is odd.
2. `for` $i = 1$ to t
3. Choose an integer $a \in \{2, 3, \ldots N - 2\}$, randomly.
4. Compute $y \leftarrow a^{\beta} \mod N$.
5. `if` $y \neq 1$ `and` $y \neq N - 1$ `then`
6. $j \leftarrow 1$.
7. `while` $j \leq \alpha - 1$ `and` $j \neq N - 1$
8. Compute $y \leftarrow y^2 \mod N$.
9. `if` $y = 1$ `then`
10. `return false`.
11. $j \leftarrow j + 1$.
12. `if` $y \neq N - 1$ `then`
13. `return false`.
14. `return true`.

Language requirements. During this problem, you should follow the programming rules:

- You should use an ANSI C programming language whose source code can run on `Code ocean` platform.
- Function naming: Begin with the lower character, and every parameters are strong-typed variables (i.e., do not use `void` typed variables). All functions should have a single return value; thus even if a function will return no values; you should provide `return` keyword.
- Variable naming: Begin with a type-discriminating prefix. For example, if a variable name is for an age and is with an integer type, you need to declare the variable as `int iAge;` Especially for string-type variables you are strongly recommended to use the prefix `sz`. For example, if a variable name is for a name, then `szName` is a preferable choice.

Input format. The input is given a text-format file, named `input.txt` and all strings are separated by the carriage-return character. The first input value `n` is the number of input integers, and all following numbers from N_1 to N_n are integers whose length is less than or equal to 64 bits.

`n`
N_1
N_2
$\vdots$
N_n

Output format. You should test whether all input values are prime or not, When an input N_j is prime, you only have to output `t`, but if the N_j is not prime, then you should find the nearest prime number N^* from N_j and should output `f`(N^*). The output should be saved at the file named `output.txt` as before.