Data Structures (CS 102), Professor Yap – Fall 2019

HOMEWORK

October 10, 2019

Homework 4: Due on Thur Oct 17, 2019

- Carefully read the Homework Policy in the class wiki. Pay attention to the rules for submitting programs in a zip file (not rar or other formats).

PART A: WRITTEN ASSIGNMENT

A.1 (3+3+3 Points)

(a) R-4.11, page 182 of Text.

(b) R-4.12

(c) R-4.13

A.2 (3+3 Points) Let F_n ($n = 0, 1, 2, \ldots$) be the n-th Fibonacci number where $F_n = n$ for $n = 0, 1$. If $P_n = \begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix}$ be the n-th pair. Let $N = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ be the "next matrix" because $N \cdot P_n = P_{n+1}$ is the next pair (please verify).

(a) Please compute N^n for $n = 1, 2, 4, 8$.

(b) Prove that $N^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$ for all natural numbers $n \geq 1$. Be sure to state the base case.

Review Lecture 6 and read Section 4.4.3 of Text.

A.3 (3+5+3 Points) In our `List.java` design, we had the methods `addHead`, `addTail` and `removeHead`.

(a) Why did we not have a method called `removeTail`?
NOTE: it is important to understand the reason because similar reasons are at work in the methods provided by Java libraries!

(b) Please add the method `removeTail` to `List.java` (use our solution to hw3). Like `removeHead`, it takes no arguments, but returns an `int` from the removed node.

(c) Moreover add a test to the existing `main` method in which you further apply `removeTail` 3 times to our two lists in the main method, and print the result. Your output should resemble this sample:

```
> make p=List r
tailList is:
   1 5 1 2 5 4 0 8 7
headList is:
   7 8 0 4 5 2 1 5 1
tailList after 3 removeHead() is:
```

```
   2 5 4 0 8 7
headList after 3 removeHead() is:
   4 5 2 1 5 1
tailList after 3 removeTail() is:
   2 5 4
headList after 3 removeTail() is:
   4 5 2
```

A.4 (2+2+2+2 Points) Our standing rule for submitting your programming homework is to put the entire folder inside a zip file named `hwXXX_YYY.zip` where you should replace `XXX` by the homework number, and `YYY` by your last name (with initial cap). We want to detect this using Java patterns. Consider the following instance of the MatchAB Problem for the following AA-list and BB-list:

```
AA: hw1_Yap.zip, hw9_Konkimalla.zip, hw3_Abcd.zip, hw1_Xyz.zip
BB: hw1_Yap.rar, hw01_Yap.zip, Hw1_Abcd.zip, hw-Xyz.zip, hwk1_Konkimalla.zip
```

As in homework 3, please write four Java patterns `Pat1`, . . . ,`Pat4` which are (respectively) a solution, has only false negative, has only false positive, and has both kinds of falsity. To avoid the trivial solution, the solution pattern `Pat1` should have at most 25 characters. Please these patterns in a text file called `ppp.txt` under the `src` folder of your programming folder. This allows us to test your solution.

PART B: PROGRAMMING (60 Points)

Note that we have two programs to write! You may use the targets t1, t2, t3 in our Makefile to test your programs (please do not change them).

B.1 Start with the Java source `List.java` found in `hw4_Yap.zip`. We slightly modified this class from hw3. Please modify the file `List.java` by implementing the method `removeTail`, and to test it, as described in our written part. Please do not modify other parts of `List.java`. The command

```
> make run p=List
```

should produce an output similar to the example in the written part.

B.2 We provide you with two classes for this problem: `MatrixBig` and `FibBig`. The "Big" in these names refers to the fact that we are using `BigInteger` instead of `int`. Class `FibBig` contains a method `fibBig(n)` to compute the n-th Fibonacci number using the method of memo-ization (so it takes time $O(n)$ to compute F_n). Class `MatrixBig` contains an implementation of adding and multiplying 2×2 matrices. Your goal is to modify `MatrixBig` in order to to implement and test a method called `fibLog(n)`. You should not modify `FibBig`, which is provided so that its method can be called and compared against `fibLog(n)`. The name "fibLog" comes from the fact that it should compute F_n in only $O(\log n)$ steps. Note that the argument `n` for `fibLog(n)` is `int` but[1] the output is `BigInteger`.

We now sketch the method of `fibLog(n)`. Let `bin(n)` denote the binary representation of a integer `n`. You can use the Java method `Integer.toBinaryString(int n)` to compute `bin(n)`. Here are some examples:

n:	1	2	3	4	5	10	11	12	13	14	15
bin(n):	1	10	11	100	101	1010	1011	1100	1101	1110	1111

[1] Reason? Because `int` is much easier to handle, and there is no reason to compute F_n for n that is larger than what `int` can represent. But the value of F_n needs `BigInteger` to represent it.

Using `bin(n)`, we can compute N^n very efficiently, where N is the "next matrix" from the written part. First compute these matrices:

$$\begin{aligned} N_0 &:= N^1 = N^{2^0} \\ N_1 &:= N^2 = N^{2^1} \\ N_2 &:= N^4 = N^{2^2} \\ &\vdots \\ N_k &:= N^{2^k} \end{aligned}$$

where 2^k is the largest power of 2 that is not larger than n. It follows that $k \le \log_2 n$. Note that $N_{i+1} = N_i N_i = (N_i)^2$. Then we compute N^n using at most k additional multiplications:

```
M = I (identity matrix);
for (i=0; i<=k; i++)
    if (b_i=1) M = M · N_i
return M
```

Implement a method called `power(int n)` a `MatrixBig` object that represents N^n; this takes at most $2k \le 2 \log n$ matrix operations. Using `power(n)` you can write the method called `fibLog(int n)` that returns F_n as a `BigInteger`.

Finally, write a method `compare(int n)` that returns an array of three doubles:

`[TT, tt, rr]`

where TT is the average time in seconds (not milliseconds) to run `fibBig(n)` 3 times, and tt is the corresponding time for `fibLog(n)`, and `rr=TT/tt` is their ratio. Please run the test for these 5 values:

$$n = 100, 1000, 10000, 100000, 1000000$$

Report your results in a file `timeRatio.txt` of your folder (at the same level as `src`). For your reference, here is my output of such a test:

```
nn        TT(secs)     tt(secs)     TT/rr
======    =========    =========    ========
   100     0.000        0.000       Infinity
  1000     0.001        0.000       3.0000
 10000     0.009        0.003       3.0000
100000     0.240        0.024       9.9861
1000000    18.488       0.370       49.9217
```

Don't worry if you get different numbers because they depend on your computer configuration. In fact, if you have a slow computer, you may kill a job if it takes more than 10 minutes and report "time-out".

We provide a class `BigUtil` with a method `show35(BigInteger N)` that will print only the first 3 digits and last 5 digits of `N` (and print the total number of digits in `N`). E.g. `N=1234567890` will print as `123...67890(10)`. Feel free to use this routine!