hw0 problem 3 (PSOFT)

Brian Wang-Chen

January 2024

Question 1

Why did Fibonacci fail the testThrowsIllegalArgumentException test? What did you have to do to fix it?

The Fibonacci failed the testThrowsIllegalArgumentException test because 0 is not a negative number however Fibonacci considered it as a negative number. So when 0 is passed in as n, it will throw the error when it should not because zero is not negative, instead it should return 0. I changed the conditional to correctly consider negative numbers, not including 0.

Old : if (n <= 0) throw new IllegalArgumentException(n + " is negative");
New: if (n < 0) throw new IllegalArgumentException(n + " is negative");

Question 2

Why did Fibonacci fail the testBaseCase test? What (if anything) did you have to do to fix it?

The Fibonacci failed the testBaseCase test because it does not correctly calculate the base cases because the conditional has a flaw which is considering the 2ND element as a base case. The Fibonacci sequence goes like this: 0, 1, 1 meaning the 2ND element should not just return n but the previous 2 elements added together.

Old : else if (n <= 2) return n;
New : else if (n < 2) return n;

Question 3

Why did Fibonacci fail the testInductiveCase test? What (if anything) did you have to do to fix it?

Fibonacci failed the testInductiveCase test because it was not correctly calculating the term based off n. The Fibonacci formula for the Nth term is the 2 previous terms added together, fib(n-1) + fib(n-2). Fibonacci was using the wrong formula of fib(n+1) - fib(n-2) which will give the incorrect number

Old : getFibTerm(n + 1) - getFibTerm(n - 2)
New : getFibTerm(n - 1) + getFibTerm(n - 2)

Question 4

Why did Fibonacci fail the testLargeN test? What (if anything) did you have to do to fix it?

Fibonacci failed the testLargeN test because it was taking way too long to run because it was not optimized to calculate such large terms with the N the testLargN test case was giving. I had to rewrite the code to be optimized enough to run in reasonable time and output the correct number. I also had to change the type data Fibonacci was using from int to long because the numbers being calculated by testLargeN were simply too big therefore it would cause overflow issues if it was a integer, so I changed it to a long.

Changes :
private static Map<Integer, Long> memory = new HashMap<>();
if (memory.containsKey(n)) return memory.get(n);
memory.put(n, (long)n);
long term = getFibTerm(n - 1) + getFibTerm(n - 2);
memory.put(n, term);
return term;

```
private static Map<Integer, Long> memory = new HashMap<>();

public long getFibTerm(int n) {
    if (n < 0) {
        throw new IllegalArgumentException(n + " is negative");
    } else if (memory.containsKey(n)){
        return memory.get(n);
    } else if (n < 2) {
        memory.put(n, (long)n);
        return n;
    } else {
        long term = getFibTerm(n - 1) + getFibTerm(n - 2);
        memory.put(n, term);
        return term;
    }
}
```

Question 5

What was causing Fibonacci to be so slow on testLargeN test? What did you do to make Fibonacci faster while still preserving the recursive nature of your implementa- tion?

The reason why Fibonacci was so slow with testLargN was because it was calculating the 60Th term of the sequence with only the starting base cases of 0, 1. It would need to recursively calculate every term before the 60Th term to calculate the 60th term which would simply take too long. To make it faster while preserving the recursive nature, implemented a hash map that stores Fibonacci terms that have already been calculated, which will be used by a conditional to see that if the calculated term has already been calculated before, if yes, it will just pull it from memory, if not in memory it will put it in. Also had to change the return types to longs instead because for the large test case, we had to deal with extremely large numbers which the int cannot represent.

Changes can be seen in question 4 picture