Aaron Thompson
21 April 2019
CS-2252-01

Categorize 3: Fibonacci Words

For categorize 3, I have decided to complete the problem Fibonacci Words, which can be found at https://open.kattis.com/problems/fibonacci. I believe that this belongs in category 3 since it is a Dynamic Programming problem. For this problem, you only need 3 spots in the array as I have put in the attempted python program (I will go over that problem later). There is a lot going on in this problem, but this is the most important thing to note: this is not the actual Fibonacci sequence. Instead, you use the Fibonacci sequence equation (starting with 0 and 1), and then concatenate them together to create the numbers using the equation (i.e. your array with 7 values should look like [0,1,10,101, 1011010110101,1011010110110]). After you figure this out (which took me a while), you can truly begin to program.

So the biggest part of this problem is memory, and I hate it. When I tried to program the problem, I kept hitting that issue. However, this is code that, with unlimited memory, would work. To calculate the Fibonacci word, we only need 3 values, the previous 2 and the 1 that we are making, so we have an array of 3. This array does the formula based on starting on filling the 0th array index, and then going up from there. After it does this, it returns that string to the regular expression string, which then looks for matches and counts them. It is important to note that a match must have overlap, and we tackle this in the regular expression itself. In python, if you have the operator (?=…), it will only match if the pattern … is equal to what is after the number, but the regex will not go on to the next letter. And so by doing this, we ensure that everything is done properly. The memory issue comes into play when you hit the last sample test case given by the prompt, 96. Because we are concatenating strings, and strings are immutable, it is impossible to complete the problem when they are strings. However, frankly, I don't understand how python deals with binary numbers to know how to concatenate the actual 1's and 0's efficiently.

The following inputs and outputs are the values of which I would use to test the program. I would keep the 96-test case, because that is a valid edge case. I would also test on 1 with the string 10, and they should return 0, however it could throw a index out of bounds error. I would also throw a test in that is the full binary number to the full binary number that they get to ensure that the number that they are getting is the true one. If it was timed, I would also do a moderately high number (like 30), multiple times to see if they recompute it every time or store it somewhere smartly.

Input:

```
1
10
96
10110101101101
5
101101010
```

Output:

```
Case 1: 0
Case 2: 7540113804746346428
Case 3: 1
```

Aaron Thompson
21 April 2019
CS-2252-01

```
import re
repeatLibrary = [[],[]]

def fib(n):
        fibArray = ["0", "1", "10"]
        placeHolder = 0
        if n in repeatLibrary[0]:
                n = repeatLibrary[0].index(n)
                return repeatLibrary[1][n]
        else:
                repeatLibrary[0].append(n)
        for ii in range(3, n+1):
                if placeHolder == 0:
                        fibArray[0] = fibArray[2] + fibArray[1]
                        placeHolder = 1
                        prevHolder = 0
                elif placeHolder == 1:
                        fibArray [1] = fibArray[0] + fibArray[2]
                        placeHolder = 2
                        prevHolder = 1
                elif placeHolder == 2:
                        fibArray[2] = fibArray[1] + fibArray[0]
                        placeHolder = 0
                        prevHolder = 2
        repeatLibrary[1].append(fibArray[prevHolder])
        return fibArray[prevHolder]
def bitmatcher(testBits, calcBits):
        return len(re.findall(testBits[0]+"(?="+testBits[1:]+")", calcBits))

def main():
        caseNum = 1
        while True:
                try:
                        n = int(input())
                except:
                        break
                bitPattern = input()
                fib(n)
                timesFound = bitmatcher(bitPattern, fib(n))
                timesFound = int(timesFound)
                print(f"Case {caseNum}: {timesFound}")
                caseNum += 1
main()
```