COL100 Assignment 1

Date of Submission: 4 December, 2020

1 Stair climbing problem

1. If $n = 1$ there's only one way to reach the top stair- by climbing a single stair... If $n = 2$, there are two ways: by climbing 2 stairs at once, or by climbing 1 stair twice. For $n > 2$, there are two methods of reaching the nth stair:

   (a) By climbing two stairs from the $(n-2)$th stair

   (b) By climbing one stair from the $(n-1)$th stair

   So, in total the number of ways of reaching the nth stair is

   $$f(n) = f(n-1) + f(n-2)$$

2. (Code in SML File)[1]

```
1  fun climbStair(n) = if n < 1 then raise
   Fail("Stairs can't be nonpositive") else if n = 1
   then 1
2                      else if n = 2 then 2
3    else climbStair(n-1) + climbStair(n-2);
4  climbStair(9);
```

```
> val climbStair = fn: int → int;
> val it = 55: int;
```

Figure 1: Code for Problem 1

3. For $n = 3$, the given equation is true.

   Assume that the equation

   $$f(n) = 2 + \sum_{i=1}^{n-2} f(i)$$

   holds for some $n > 3$. Then for $n + 1$, we have

   $$f(n+1) = 2 + \sum_{i=1}^{n-1} f(i)$$

   $$f(n+1) = f(n) + f(n-1) = 2 + \sum_{i=1}^{n-1} f(i)$$

[1]The function climbStair is the same as the fibonacci function. This process can also be done using fastfib function as seen on Piazza - much faster.

$$f(n) = 2 + \sum_{i=1}^{n-2} f(i)$$

Which is true based on our assumption. Therefore by the Principle of Mathematical Induction, the given equation holds for all $n > 2$.

2 Playing with digits

1. Let n be a positive integer with digits $d_k...d_1d_0$ with d_k being the most significant digit. $f(n)$ has been defined as

$$f(n) = 2^k d_k + ... + 2^1 d_1 + 2^0 d_0$$

$$Claim : f(n) = 2f(\lfloor n/10 \rfloor) + d_0$$

$$Proof : f(n) = 2^k d_k + ... + 2^1 d_1 + 2^0 d_0 \; and \; (\lfloor n/10 \rfloor = d_k...d_2d_1)^2$$

$$f(\lfloor n/10 \rfloor) = 2^{k-1} d_k + ... + 2^0 d_1$$

$$2f(\lfloor n/10 \rfloor) = 2^k d_k + ... + 2^1 d_1 = f(n) - d_0$$

which implies that our claim is true.

2. Since now we have a recursive relation between $f(n)$ and $f(\lfloor n/10 \rfloor)$, we can use it to write a code for $f(n)$ by setting the base case as $f(0) = 0$.

```
1  fun modifiedDigitSum(x) = if x < 0 then raise
   Fail("Integer entered must be non-negative") else
   if x = 0 then 0 else modifiedDigitSum(x div 10)*2 +
   x mod 10;
2  modifiedDigitSum(132);
```

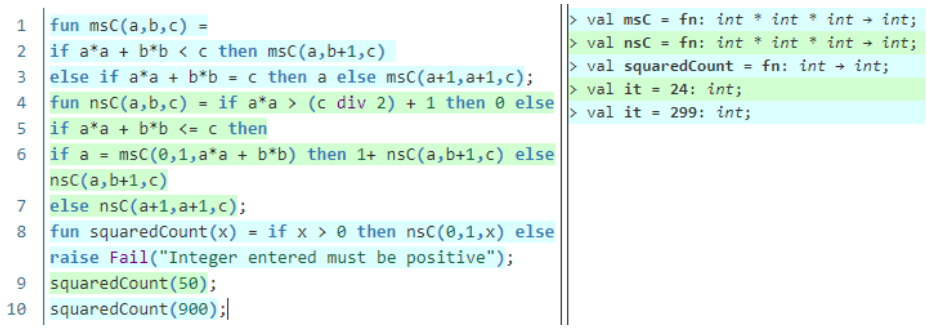
```
> val modifiedDigitSum = fn: int → int;
> val it = 12: int;
```

Figure 2: Code for Problem 2

3 Sum of squares

(Code in SML File)[3]

[2] $\lfloor x \rfloor$ represents the greatest integer smaller than or equal to x.
[3] Expect a time violation for values bigger than or around 900.



```
fun msC(a,b,c) =
if a*a + b*b < c then msC(a,b+1,c)
else if a*a + b*b = c then a else msC(a+1,a+1,c);
fun nsC(a,b,c) = if a*a > (c div 2) + 1 then 0 else
if a*a + b*b <= c then
if a = msC(0,1,a*a + b*b) then 1+ nsC(a,b+1,c) else
nsC(a,b+1,c)
else nsC(a+1,a+1,c);
fun squaredCount(x) = if x > 0 then nsC(0,1,x) else
raise Fail("Integer entered must be positive");
squaredCount(50);
squaredCount(900);
```

```
> val msC = fn: int * int * int → int;
> val nsC = fn: int * int * int → int;
> val squaredCount = fn: int → int;
> val it = 24: int;
> val it = 299: int;
```

Figure 3: Code for Problem 3

3.1 Algorithm

The helper function $msC(a, b, c)$ has been defined as:

$$msC(a,b,c) = \begin{cases} a & a^2 + b^2 = c \\ msC(a, b+1, c) & a^2 + b^2 < c \\ msC(a+1, a+1, c) & \text{otherwise} \end{cases}$$

Another helper function $nsC(a, b, c)$ has been defined as:

$$nsC(a,b,c) = \begin{cases} 0 & a^2 > [c/2] + 1 \\ nsC(a, b+1, c) & a^2 + b^2 \leq c \text{ and } a \neq msC(0, 1, a^2 + b^2) \\ 1 + nsC(a, b+1, c) & a^2 + b^2 \leq c \text{ and } a = msC(0, 1, a^2 + b^2) \\ nsC(a+1, a+1, c) & \text{otherwise} \end{cases}$$

Our main function is simply $squaredCount(x) = nsC(0, 1, x)$.

3.2 Proof of Correctness

The function $squaredCount(x)$ takes as input the integer for which we want to calculate the number of integers that can be expressed as the sum of squares. Now there can be multiple ways of expressing an integer as a sum of squares. For example: $25 = 0^2 + 5^2 = 3^2 + 4^2$ are two ways of writing 25 as a sum of two squares. We need to ensure that we don't double count these integers. The helper function $msC(a, b, c)$ is made to ensure that double counting doesn't take place.

First, $nsC(a, b, c)$ starts with $(0, 1, x)$ as input and then checks whether $a^2 + b^2 < x$. Now for sufficiently large x, this will be the case for initial values of (a, b). When this happens, the function $msC(0, 1, a^2 + b^2)$ calculates a pair of integers (p, q) such that $p \leq q$ and $p^2 + q^2 = a^2 + b^2$ and for every such pair (p_i, q_i) for which this property holds, $msC(0, 1, a^2 + b^2)$ selects the pair of integers such that p is the smallest one among all p_i and then returns p as the

output. The way it is defined, it is set to output an integer $p \leq a$ and thus it will always terminate at some point.

Now, $nsC(a, b, x)$ checks whether $a = p$ or not. In case $a = p$, that implies that a^2+b^2 has *not* been counted before. So, the function returns $1+nsC(a, b+1, x)$ as output i.e. it continues for bigger values of (a, b). In case $a \neq p$, that implies that $a^2 + b^2$ has been counted before and the function outputs $nsC(a, b+1, x)$. This way, the function continues to increment b by *one* till a^2+b^2 crosses x and then the function starts this process again for $(a', b') = (a+1, a+1)$. This ensures that we don't double check for (a, b) *and* (b, a) for $a < b$.

Since we are starting with $(a, b) = (0, 1)$ and computing till $a^2 > [x/2] + 1$ which is the same as the condition

$$a^2 + b^2 \geq a^2 + a^2 (\text{as } b \geq a) = 2a^2 > 2[x/2] + 2 > x \text{ or simply } a^2 + b^2 > x$$

So, we are counting all possible cases starting from 1 till x itself. Hence, this algorithm only considers the integers of the required form and counts them if and only if they haven't been counted before.

4 Summing a series

```
1  fun t(n) = let
2  val x = real(2*(n-1)) in
3    if n = 1 then 3.0
4      else if n mod 2 = 0 then 4.0/((x)*(x+1.0)*(x+2.0))
5  else (~4.0)/((x)*(x+1.0)*(x+2.0)) end;
6  fun Mod(x) = if x > 0.0 then x else 0.0-x;
7  fun f(x,n) = if Mod(x) > Mod(t(n)) then 0.0 +t(n) else
   f(x,n+1)+t(n);
8  fun nilakanthaSum(x) = if x > 0.0 then f(x,1)
9  else raise Fail("Enter positive real as input");
10 nilakanthaSum(0.000001);
```

```
> val t = fn: int → real;
> val Mod = fn: real → real;
> val f = fn: real * int → real;
> val nilakanthaSum = fn: real → real;
> val it = 3.1415931417757212: real;
```

Figure 4: Code for Problem 4

The helper function $t(n)$ represents the nth term of the Nilakantha sequence. Function $Mod(x)$[4] is simply the mathematical function $Mod(x) = |x|$ and gives the absolute value of x.

Function $nilakanthaSum(x) = f(x, 1)$ is the required function.

Function $f(x, n)$ compares whether the absolute value of the nth term i.e. $Mod(t(n))$ is less than $Mod(x)$. If $Mod(t(n)) > Mod(x)$ then the function returns $t(n)$ as output. Else it returns $t(n) + f(x, n + 1)$ as output. This way it continues to check for a term whose absolute value is less than the absolute value of x and since the terms of the series get smaller as n increases, we will find a term for some sufficiently large n whose absolute value is less than the absolute value of x, and the algorithm will terminate at that step.

[4]Note that $Mod(x)$ is not the same as the *mod* operator.

5 Notes

1. According to the requirements of the question, "raise Fail" codes have been put in place to ensure that the input is correct.

2. I was not entirely sure whether using the *fibloop* code as seen on Piazza COL100 question 160 would be considered as Plagiarism or not so I have opted not to use it. But that code would be much more efficient to calculate $climbStair(n)$ for bigger values of n. (The "normal" code *infamously* runs out of time for $n = 23$.)

3. No references other than SML editor have been used for writing the code.

4. I have taken the help of mutiple websites to write this document as this my first time using LaTeX and the only help I have looked out for is regarding LaTeX itself. I have designed the algorithms and the corresponding codes by myself.

The End