CS 5200 Spring 2018 Analysis of Algorithms-Homework2

Xiongming Dai

Feburary 10,2018

1 Prove by the Principle of Recursion

The experimental code is as follows:

```
def f(i):
    if i<=1:
        return 1
    else:
        return f(i-1)+i*i


def g(n):
    return n*(n+1)*(2*n+1)/6


for i in range(50):
    print i+1, f(i+1),g(i+1)
```

The output is as follows:

```
1 1 1
2 5 5
3 14 14
4 30 30
5 55 55
6 91 91
7 140 140
8 204 204
9 285 285
10 385 385
11 506 506
12 650 650
13 819 819
14 1015 1015
15 1240 1240
16 1496 1496
17 1785 1785
```

18	2109	2109
19	2470	2470
20	2870	2870
21	3311	3311
22	3795	3795
23	4324	4324
24	4900	4900
25	5525	5525
26	6201	6201
27	6930	6930
28	7714	7714
29	8555	8555
30	9455	9455
31	10416	10416
32	11440	11440
33	12529	12529
34	13685	13685
35	14910	14910
36	16206	16206
37	17575	17575
38	19019	19019
39	20540	20540
40	22140	22140
41	23821	23821
42	25585	25585
43	27434	27434
44	29370	29370
45	31395	31395
46	33511	33511
47	35720	35720
48	38024	38024
49	40425	40425
50	42925	42925

Step1. Define the problem:
D is the set of natural numbers and includes the stopping value 1. We want to prove that for all n in D, the returns of $f(n) == g(n)$ is always true. when $f(i)$ and $g(n)$ are defined as above.
Step2. Check the stopping and 2 other values
We have done this with the progam, but can do it by hand $f(1) = 1; f(2) = 5; f(3) = 14; f(4) = 30$, $g(1) = 1; g(2) = 5; g(3) = 14; g(4) = 30$
Step3.Prove Recursion Stays in D
The recursive relation is: $f(i) = i^2 + f(i-1)$ Recursion is only called if $i > 1$ Thus, in all cases $i - 1 > 0$ thus,the call is made with a value in D
Step4.Prove that Recursion Halts.
Our method for doing this is the Halting Stratege. This means, that we first identify something that we can attach a natural number to ! Since n is a natural number, it seems logical to try n as the counter The recursive relation is : $f(i) = i^2 + f(i-1)$ It is easy to see that the counter decreases!
Step5. Prove that expression is inherited recursively
We want to prove that $f(n) == g(n)$ is true, assuming that $f(n-1) == g(n-1)$ is true;

$f(n) = n^2 + f(n-1)$, $f(n) = n^2 + g(n-1)$, $g(n-1) = (n-1)n(2n-1)/6$, Combine these equations, we have $f(n) = g(n)$

Since we have verified Steps1-5,we can conclude that it is true for all elements of D, this means that $f(n) = g(n)$ for all natural numbers.

2 Consecutive integers

The experimental code is as follows:

```
def f(n):
    if n==0:
        return 9
    else:
        return 9*n**2+9*n+9+f(n-1)




def g(n):
    return (n**3+(n+1)**3+(n+2)**3)


def P(n):
    return (f(n)==g(n)) and (0==(g(n) %9))

for n in range(16):
    print "n=%2d f(n)=%4d g(n)=%4d P(n)=%s"%(n,f(n),g(n),P(n))
```

The output is as follows:

```
n= 0 f(n)=    9 g(n)=    9 P(n)=True
n= 1 f(n)=   36 g(n)=   36 P(n)=True
n= 2 f(n)=   99 g(n)=   99 P(n)=True
n= 3 f(n)=  216 g(n)=  216 P(n)=True
n= 4 f(n)=  405 g(n)=  405 P(n)=True
n= 5 f(n)=  684 g(n)=  684 P(n)=True
n= 6 f(n)=1071 g(n)=1071 P(n)=True
n= 7 f(n)=1584 g(n)=1584 P(n)=True
n= 8 f(n)=2241 g(n)=2241 P(n)=True
n= 9 f(n)=3060 g(n)=3060 P(n)=True
n=10 f(n)=4059 g(n)=4059 P(n)=True
n=11 f(n)=5256 g(n)=5256 P(n)=True
n=12 f(n)=6669 g(n)=6669 P(n)=True
n=13 f(n)=8316 g(n)=8316 P(n)=True
n=14 f(n)=10215 g(n)=10215 P(n)=True
n=15 f(n)=12384 g(n)=12384 P(n)=True
```

Step1.Define the Problem
The domain D is the set of natural numbers, we want to prove that $P(n)$ is true for all natural numbers where these functions are defined as above.

Step2.Check the Stopping and Two Other Values
$f(0) = 9, g(0) = 9$,9 mod 9 equal to zero. Therefor,$P(0)$ is true. $f(1) = 39, g(1) = 36$,36 mod 9 equal to zero. Therefor,$P(1)$ is true. $f(2) = 99, g(2) = 99$,99 mod 9 equal to zero. Therefor,$P(2)$ is true.
Step 3.Prove that Recursion Stays in D
f is the only recursively defined function and its formula is give above. Recursion is only called if n is larger or equal to 1. The value called is $n - 1$ $n - 1$ is larger or equal to zero, so the calling value remains in D.
Step4.Prove that Recursion Halts.
Our method for doing this is the Halting Strategy,this means that we first identify something what we can attach a natural number 0.Since n is a natural number, it seems logical to try n as the counter. The recursive relation is: $f(n) = 9n^2 + 9n + 9 + f(n-1)$,it is easy to see that the counter decreases!
Step5.Prove that P is inherited recursively.
We may assume that $P(n-1)$ is true.It means that $g(n-1) = f(n-1)$ and $g(n-1)$ mod 3 equal to 0. we begin with $f(n) = 9n^2 + 9n + 9 + f(n-1)$,$f(n) = 9n^2 + 9n + 9 + g(n-1) = g(n)$,$f(n) = 9(n^2 + n + 1) + g(n-1) = g(n)$ Therefore,$P(n)$ is true.
Step6.Conclude the Proof.
Since we have verified Steps1-5, we can conclude that $P(n)$ is true for all elements of D. This means that $f(n) = g(n)$ for all natural numbers, and that $g(n)$ mod 9=0 for all natural numbers.

The program works recursively as the above statement.

3 Divisible by 133

The experimental code is as follows:

```
def f(n):
    if n==0:
        return 11*11+12
    else:
        return 10*11**(n+1)+(12*12-1)*12**(2*n-1)+f(n-1)




def g(n):
    return (11**(n+2)+12**(2*n+1))


def P(n):
    return (f(n)==g(n)) and (0==(g(n) %133))

for n in range(10):
    print "n=%2d f(n)=%4d g(n)=%4d P(n)=%s"%(n,f(n),g(n),P(n))
```

The output is as follows:

```
n= 0 f(n)= 133 g(n)= 133 P(n)=True
n= 1 f(n)=3059 g(n)=3059 P(n)=True
n= 2 f(n)=263473 g(n)=263473 P(n)=True
```

```
n= 3 f(n)=35992859 g(n)=35992859 P(n)=True
n= 4 f(n)=5161551913 g(n)=5161551913 P(n)=True
n= 5 f(n)=743027857859 g(n)=743027857859 P(n)=True
n= 6 f(n)=106993419737953 g(n)=106993419737953 P(n)=True
n= 7 f(n)=15407023932534059 g(n)=15407023932534059 P(n)=True
n= 8 f(n)=2218611132677861593 g(n)=2218611132677861593 P(n)=True
n= 9 f(n)=319479999655934597459 g(n)=319479999655934597459 P(n)=True
```

Step1.Define the Problem
The domain D is the set of natural numbers, we want to prove that $P(n)$ is true for all natural numbers where these functions are defined as above.
Step2.Check the Stopping and Two Other Values
$f(0) = 133, g(0) = 133$,133 mod 133 equal to zero. Therefor,$P(0)$ is true. $f(1) = 3059, g(1) = 3059$,3059 mod 133 equal to zero. Therefor,$P(1)$ is true. $f(2) = 263473, g(2) = 263473$,263473 mod 133 equal to zero. Therefor,$P(2)$ is true.
Step 3.Prove that Recursion Stays in D
f is the only recursively defined function and its formula is give above. Recursion is only called if n is larger or equal to 1. The value called is $n - 1$ $n - 1$ is larger or equal to zero, so the calling value remains in D.
Step4.Prove that Recursion Halts.
Our method for doing this is the Halting Strategy,this means that we first identify something what we can attach a natural number 0.Since n is a natural number, it seems logical to try n as the counter. The recursive relation is: $f(n) = 10 * 11^{n+1} + 143 * 12^{2n-1} + f(n-1)$,it is easy to see that the counter decreases!
Step5.Prove that P is inherited recursively.
We may assume that $P(n-1)$ is true.It means that $g(n-1) = f(n-1)$ and $g(n-1)$ mod 3 equal to 0. we begin with $f(n) = 10 * 11^{n+1} + 143 * 12^{2n-1} + f(n-1)$,$f(n) = 10 * 11^{n+1} + 143 * 12^{2n-1} + g(n-1) = g(n)$,. Therefore,$P(n)$ is true.
Step6.Conclude the Proof.
Since we have verified Steps1-5, we can conclude that $P(n)$ is true for all elements of D. This means that $f(n) = g(n)$ for all natural numbers, and that $g(n)$ mod 133=0 for all natural numbers.

The program works recursively as the above statement.

4 Fib Proof

In order to prove that for
$$2^n \geq fib(n) \geq 2^{n/2}, (n \geq 2)$$
, we just need to prove
$$2^{n+1} \geq fib(n+1) \geq 2^{(n+1)/2}, (n \geq 2)$$
, assume $2^n \geq fib(n) \geq 2^{n/2}, (n \geq 2)$. If $n = 2, 2^2 \geq fib(2) = 2 \geq 2^1$,

$$2^n \geq fib(n) \geq 2^{n/2}, (n \geq 2) \quad (1)$$
$$2^{n-1} \geq fib(n-1) \geq 2^{(n-1)/2}, (n \geq 2) \quad (2)$$

$\Rightarrow 2^n + 2^{n-1} \geq fib(n) + fib(n-1) \geq 2^{n/2} + 2^{(n-1)/2}, (n \geq 2)$

$\Rightarrow fib(n) + fib(n-1) \leq 2^n + 2^{n-1} < 2^{n+1}$ $2^{n/2} + 2^{(n-1)/2} = 2^{n/2}(\frac{\sqrt{2}}{2} + 1) > 2^{n/2} \bullet \sqrt{2} = 2^{(n+1)/2} \Rightarrow 2^{n+1} \geq fib(n+1) \geq 2^{(n+1)/2}$

Therefore, it has been proved!

5 Fibcount

The experimental code is as follows:

```
def fibcount(n):
    if n<2:
        return (1,1)
    f1,c1=fibcount(n-1)
    f2,c2=fibcount(n-2)
    return (f1+f2,c1+c2+1)


def Rec(n):
    if n<2:
        return (1)
    else:
        return Rec(n-1)+Rec(n-2)+1

def fibCC(n):
    d1,d2=fibcount(n)
    return d2

def P(n):
   return (Rec(n)==fibCC(n))
for n in range(30):
    print "n=%2d Rec(n)=%4d fibCC(n)=%4d P(n)=%s"%(n,Rec(n),fibCC(n),P(n))
```

The output is as follows:

```
n= 0 Rec(n)=    1 fibCC(n)=    1 P(n)=True
n= 1 Rec(n)=    1 fibCC(n)=    1 P(n)=True
n= 2 Rec(n)=    3 fibCC(n)=    3 P(n)=True
n= 3 Rec(n)=    5 fibCC(n)=    5 P(n)=True
n= 4 Rec(n)=    9 fibCC(n)=    9 P(n)=True
n= 5 Rec(n)=   15 fibCC(n)=   15 P(n)=True
n= 6 Rec(n)=   25 fibCC(n)=   25 P(n)=True
n= 7 Rec(n)=   41 fibCC(n)=   41 P(n)=True
n= 8 Rec(n)=   67 fibCC(n)=   67 P(n)=True
n= 9 Rec(n)=  109 fibCC(n)=  109 P(n)=True
n=10 Rec(n)=  177 fibCC(n)=  177 P(n)=True
n=11 Rec(n)=  287 fibCC(n)=  287 P(n)=True
n=12 Rec(n)=  465 fibCC(n)=  465 P(n)=True
n=13 Rec(n)=  753 fibCC(n)=  753 P(n)=True
n=14 Rec(n)=1219 fibCC(n)=1219 P(n)=True
n=15 Rec(n)=1973 fibCC(n)=1973 P(n)=True
n=16 Rec(n)=3193 fibCC(n)=3193 P(n)=True
n=17 Rec(n)=5167 fibCC(n)=5167 P(n)=True
n=18 Rec(n)=8361 fibCC(n)=8361 P(n)=True
n=19 Rec(n)=13529 fibCC(n)=13529 P(n)=True
n=20 Rec(n)=21891 fibCC(n)=21891 P(n)=True
```

```
n=21 Rec(n)=35421 fibCC(n)=35421 P(n)=True
n=22 Rec(n)=57313 fibCC(n)=57313 P(n)=True
n=23 Rec(n)=92735 fibCC(n)=92735 P(n)=True
n=24 Rec(n)=150049 fibCC(n)=150049 P(n)=True
n=25 Rec(n)=242785 fibCC(n)=242785 P(n)=True
n=26 Rec(n)=392835 fibCC(n)=392835 P(n)=True
n=27 Rec(n)=635621 fibCC(n)=635621 P(n)=True
n=28 Rec(n)=1028457 fibCC(n)=1028457 P(n)=True
n=29 Rec(n)=1664079 fibCC(n)=1664079 P(n)=True
```

Step1.Define the Problem
The domain D is the set of natural numbers, we want to prove that $P(n)$ is true for all natural numbers where these functions are defined as above.
Step2.Check the Stopping and Two Other Values
$Rec(0) = 1, fibCC(0) = 1$.Therefor,$P(0)$ is true. $Rec(1) = 1, fibCC(1) = 1$. Therefor,$P(1)$ is true. $Rec(2) = 3, fibCC(2) = 3$. Therefor,$P(2)$ is true.
Step 3.Prove that Recursion Stays in D
Rec is the only recursively defined function and its formula is give above. Recursion is only called if n is larger or equal to 2. The value called is $n-1$, $n-1$ is larger or equal to one, so the calling value remains in D.
Step4.Prove that Recursion Halts.
Our method for doing this is the Halting Strategy,this means that we first identify something what we can attach a natural number 0.Since n is a natural number, it seems logical to try n as the counter. The recursive relation is:

$Rec(n) = Rec(n-1) + Rec(n-2) + 1$,it is easy to see that the counter decreases!
Step5.Prove that P is inherited recursively.

From $fibCC(n)$,we find that $fibCC(n) = 2 * fib(n-1) + 2 * fib(n-2) - 1$, if $Rec(n-1) = Rec(n-2) + Rec(n-3) + 1$, We can easily find that $Rec(n) = 2 * fib(n-1) + 2 * fib(n-2) - 1 = fibCC(n)$. Therefore,$P(n)$ is true.
Step6.Conclude the Proof.
Since we have verified Steps1-5, we can conclude that $P(n)$ is true for all elements of D. This means that $fibCC(n) = Rec(n)$ for all natural numbers.

The program works recursively as the above statement.

For the second part, if $n = 0, \operatorname{Re}c(0) = 2fib(0) - 1 = 1$, in order to prove $\operatorname{Re}c(n) = 2fib(n) - 1$,, we just need to prove $\operatorname{Re}c(n+1) = 2fib(n+1) - 1$,, given $\operatorname{Re}c(n) = 2fib(n) - 1$,.From $\operatorname{Re}c(n) = \operatorname{Re}c(n-1) + \operatorname{Re}c(n-2) + 1$, $\operatorname{Re}c(n+1) = \operatorname{Re}c(n) + \operatorname{Re}c(n-1) + 1$,$= 2fib(n) - 1 + 2fib(n-1) - 1 + 1 = 2fib(n) + 2fib(n-1) - 1 = 2fib(n) - 1$

6 Konig's Tree

The first Method:
According to the hint of the problem, we can choose an infinite sequence of nodes $F(0), F(1), F(2), ...F(n)$ of T such that:1.$F(0)$ is the root node;2.$F(n)$ is a child of $F(n-1)$;3.Each $F(n-1)$ has infinitely many descendants.It is very easy to find that the sequence $F(0), F(1), F(2), ...$ is the branch of infinite length.

First,the root node $F[0]$,from the definition, it has a finite number of children. Assume that all of these children had a finite number of descendants. it means that $F(0)$ had a finite number of descents,consequently $F(0)$ has at least one child with descendants.Thus, we might pick $F(1)$ as any one of those children.

Now, suppose node $F(n-1)$ has infinitely many descendants. As $F(n-1)$ has a finite number of children, by the same argument as above,$F(n-1)$ has at least one child with infinitely many descendants.Thus we may pick $F(n)$ which has inifitely many descendants. Similary, it has at least one children with infinitely descendants, that means the function F gives an infinite path through T. The problem is being proofed.

The second Method:
$F(0)$ is the root of T, we can establish the path begining with a. Assume $t_0 = a, t_1, t_2, ...$ be the levels of T. Next we separate all vertices in leavels higher then t_0 into as many finite parts as is the cardinality of t_1. That is to say, if $t_1 = b_1, b_2, ..., b_n$ then we separate the vertices in n parts:$A_1, A_2, ..., A_n$. This is conducted as follows in detail: All the vertex l will be connected to the root a via some point b_i. We can take l in A_i. Therefore, we can separate infinitely many vertices in finitely many parts, one of them will contain infinitely many vertices.Assume that part be A_j. We can choose $l_1 = b_j$ as the next vertex in current under-constructed path.Next we can remove the vertex a and consider some containing l_1. While it is easy to find that it is also an infinite tree with each level finite and having l_1 as its root. Similarly, we get a vertex l_2 that is near tol_1 and has inifitely many vertices corresponding to it. We can continue establishing the remaining path by selecting l_2 as a part. By induction, we can obtain an infinite path based on these roots.Therefore, T has an infinite path.

7 Full Binary Tree

(a)Python functions that calculate L and I given a full binary tree as input:

```
def leaves_and_internals(tree):

    leaves = []
    internal = []

    if tree.l is None and tree.r is None:
        leaves.append(tree.v)
    else:
        internal.append(tree.v)
    if tree.l:
        subleaf, subinternal = tree.l.leaves_and_internals()
        leaves.extend(subleaf)
        internal.extend(subinternal)
    if tree.r:
        subleaf, subinternal = tree.r.leaves_and_internals()
        leaves.extend(subleaf)
        internal.extend(subinternal)

    return (leaves, internal)
```

(b)Run some samples casess of this function, I find that if the number of internal nodes is n, the number of leaves is $n+1$. Some part of the main code is as follows:

```
#!/usr/bin/python
```

```python
class Node:
    def __init__(self, val):
        self.l = None
        self.r = None
        self.v = val

class Tree:
    def __init__(self):
        self.root = None

    def getRoot(self):
        return self.root

    def add(self, val):
        if(self.root == None):
            self.root = Node(val)
        else:
            self._add(val, self.root)

    def _add(self, val, node):
        if(val < node.v):
            if(node.l != None):
                self._add(val, node.l)
            else:
                node.l = Node(val)
        else:
            if(node.r != None):
                self._add(val, node.r)
            else:
                node.r = Node(val)

    def find(self, val):
        if(self.root != None):
            return self._find(val, self.root)
        else:
            return None

    def _find(self, val, node):
        if(val == node.v):
            return node
        elif(val < node.v and node.l != None):
            self._find(val, node.l)
        elif(val > node.v and node.r != None):
            self._find(val, node.r)

    def deleteTree(self):
        # garbage collector will do this for us.
```

```
        self.root = None

    def printTree(self):
        if(self.root != None):
            self._printTree(self.root)

    def _printTree(self, node):
        if(node != None):
            self._printTree(node.l)
            print str(node.v) + ' '
            self._printTree(node.r)

#       3
# 0       4
#   2        8
tree = Tree()
tree.add(3)
tree.add(4)
tree.add(0)
tree.add(8)
tree.add(2)
tree.printTree()
```

(c)prove that the relationship I conjectured in (b) is correct. As the full binary tree's all the node have the largest node is less or equal to 2, we consider the number of leaves as n_0, the number of some nodes that have only one descendant is n_1,the number of some nodes that have two descendant is n_2: we have the total nodes:$n = n_0 + n_1 + n_2$. On the other hand, the total number of decendant is $n_1 + 2 * n_2$. In this full binary tree, only the root is not the decendant of the other nodes, therefore, the total number of nodes in the full binary tree can also be denoted as $n = n_1 + 2 * n_2 + 1$. From thess two equations, we have $n_0 = n_2 + 1$. Therefore,(b) have been proved.