Laboratory practice No. 1: Recursions

Kevin Gutierrez Gomez
Universidad Eafit
Medellín, Colombia
kgutierreg@eafit.edu.co

Jose Alejandro Villamizar Peñaranda
Universidad Eafit
Medellín, Colombia
javillamip@eafit.edu.co

2.3)GroupSum5 explanation

The function begins with a conditional that will return true if the target value is equals to zero; then another conditional is defined which will determine if the value in the "start" position in the array is multiples of 5. Then if the value is multiple of 5 a conditional will be defined that will determine if the index of the value is not the last index and the value is followed by the number 1, if that is true then it will advance without counting the following number and subtracting the multiple of 5 from the target; otherwise, the following value will be counted.

Last but not least the recursive call that will evaluate what happens when the value from the start position that is not multiple of 5 is subtracted from target, if this action result in target equals to zero the it will return true.

2.4) Complexity of 2.1 and 2.2 (CodingBat)

Recursion 1:

1.
```
public int factorial(int n) {
return (n != 0)? //C0
n * factorial(n-1): //C1 + T(n-1)
1; //C2
}
 //T(n)=Cn + C = O(n)
```
2.
```
public int bunnyEars(int bunnies) {
 return (bunnies == 0) ?//C0
 bunnies //C1
 :2 + bunnyEars(bunnies - 1);/ / C2 + T(n-1)
 }
 //T(n)= Cn+C = O(n)
```

3.
```
Public int bunnyEars(int bunnies){
If(bunnies <= 0) return 0; // C0
Return (bunnies%2 == 0) // C1
?3 + bunnyEars2(bunnies - 1) // C2 + T(n-1)
:2 + bunnyEars2(bunnies - 1); C3 + T(n-1)
```

}// T(n) = Cn + c = O(n)

4. public int triangle(int rows) {
return (rows != 0) ?// C0

rows + triangle(rows - 1) : // C1 + T(n-1)
0; C2
} // T(n) = Cn + c = O(n)

5. public int sumDigits(int n) {
return (n == 0) ? C0
0 : C1
(n % 10) + sumDigits(n/10); C2 + T(n/10)
}// T(n) = Cn + c = O(n)

Recursion 2

1. public boolean groupSum6(int start, int[] nums, int target) {
if (start >= nums.length) return target == 0; // C0
if (nums[start] == 6) return groupSum6(start +1, nums, target - nums[start]); // C1 + T(n-1)

return groupSum6(start + 1, nums, target)
|| groupSum6(start + 1, nums, target - nums[start]); T(n-1)_+ T(n-1)
}
//T(n) = C2^n + c = O(2^n)
2. public boolean groupNoAdj(int start, int[] nums, int target) {
if (start >= nums.length) return target == 0; // C0
return groupNoAdj(start + 2, nums, target - nums[start])
|| groupNoAdj(start + 1, nums, target); T(n-1) + T(n-1)
} // T(n) = C2^n + c = O(2^n)
3. public boolean groupSum5(int start, int[] nums, int target) {
if (start >= nums.length) return target == 0; //C0
if (nums[start] % 5 == 0) // C1
if (start < nums.length - 1 && nums[start + 1] == 1) C2
return groupSum5(start + 2, nums, target - nums[start]); // T(n-1)

else return groupSum5(start + 1, nums, target - nums[start]); //T(n-1)
return groupSum5(start + 1, nums, target)
|| groupSum5(start + 1, nums, target - nums[start]); //T(n-1) + T(n-1)
} // T(n) = C2^n + c = O(2^n)
4. private boolean splitArray_aux(int start, int[] nums, int sum1, int sum2) {
if (start >= nums.length) return sum1 == sum2; // C0

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

```
        return splitArray_aux(start + 1, nums, sum1 + nums[start], sum2)
        || splitArray_aux(start + 1, nums, sum1, sum2 + nums[start]); // T(n-1) + T(n-1)
    } // T(n) = C2^n + c = O(2^n)
```

2.5)

N or M represents the value that the user passes to the program, so it represents a variable of any value

3) Practice for final project defense presentation

1. The stack saves the recursive calls made by a function, in case it exceeds the maximum number of calls in the stack either by a very large problem size or by an error in the control condition the stackOverflowError will be generated.
2. It is not possible to exceed a certain iteration number because the complexity of the recursive algorithm is quadratic and the time is increased exponentially in each iteration, so it is advisable for this problem to use a linear complexity with the use of a for cycle.
3. To calculate large values of Fibonacci you can use a cycle because its complexity is linear.
4. *The complexity of the recursion-1 problems is linear, and the complexity of the recursion-2 problems is in the form of 2^n*

4) Practice for midterms

1. *Start + 1, nums, target*
2. *D*
3. *1=n-a,a,b,c / 2=solucionar(n-b,a,b,c)+1 / 3=solucionar(n-c,a,b,c)+1*
4. *E*
5. *Linea 2 = return n / Linea 3 = return n – 1 / linea 4 = return n - 2 / 5.2) B*
6. *Linea 10 = sumaAux(n, i+ 2) / Linea 12 = sumaAux(n, i + 1)*
7. *Linea 9 = S, i + 1, t – S[i] / Linea 10 = S, i+ 1, t*

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co