Laboratory practice No. 1: Recursion

Juan Sebastián Pérez Salazar
Universidad Eafit
Medellín, Colombia
jsperezs@eafit.edu.co

Yhoan Alenjandro Guzman García
Universidad Eafit
Medellín, Colombia
yaguzmang@eafit.edu.co

3) Practice for final project defense presentation

1. **The GroupSum5 exercise is as follows:**

```
public boolean groupSum5(int start, int[] nums, int target) {
      if(start >= nums.length) return target == 0;
      if(nums[start]%5==0){
        target -= nums[start];
        if(start+1<nums.length && nums[start+1]==1) start++;
        return groupSum5(start+1, nums, target);
      }else return groupSum5(start+1, nums, target - nums[start])|| groupSum5(start+1, nums, target);
  }
```

And it works in the following way, first, the base condition or stop condition is placed, in which, if the amount of the counter is greater than the length of the array, then the evaluation of 'target == 0', (this can be true or false). Then proceed to execute the following condition, in this condition it is evaluated if the element that is in the position 'start' in the array is a multiple of 5. If this is fulfilled, we proceed to eliminate this element from the target. As stated in the statement of the problem. As in the statement of the problem is posed that if the number following the 5 in a 1, this is not taken into account, you must first start to follow, and then ask if the element in the position start + 1 is 1, in If these two conditions are true, proceed to an update 1 at the beginning so that the next execution omits the 1, finally it has to do, if the condition of% 5 is false, proceed to make the two calls recursive. For these cases you can:
- The element is subtracted from the target
- The element is not subtracted from the target

2. **Recursion1BunnyEars**

- Code

```
public int bunnyEars(int bunnies) {
      if(bunnies==0) return bunnies;
      return 2+bunnyEars(bunnies-1);
 }
```

- **Tags**

```
public int bunnyEars(int bunnies) {
    if(bunnies==0) return bunnies;  //C1
    return 2+bunnyEars(bunnies-1); //C2 + T(n-1)
 }
```

- **Recurrence equation**

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ c_2 + T(n-1) & \text{if } n > 0 \end{cases}$$

- **Solution**

$T(n) = C_1 + C_2 n$

- **Notation Big O**

$T(n)$ is $O(C_1 + C_2 n)$
$T(n)$ is $O(C_2 n)$; sum rule
$T(n)$ is $O(n)$; product rule

Recursion1Count7

- **Code**

```
public int count7(int n) {
    if(n==0) return 0;
    return (n%10==7)? 1+count7(n/10) : count7(n/10);
 }
```

- **Tags**

```
public int count7(int n) {
    if(n==0) return 0; //C1
    return (n%10==7)? 1+count7(n/10) : count7(n/10); //C2 + T(n/10) + T(n/10)
 }
```

- **Recurrence equation**

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ c_2 + T(n/10) + T(n/10) & \text{if } n > 0 \end{cases}$$

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

- **Solution**

$$t(n) = (c_1 + 2\,C_2)\,2^{\frac{\log(n)}{\log(10)}-1} - C_2$$

- **Notation Big O**

T(n) is O($(c_1 + 2\,C_2)\,2^{\frac{\log(n)}{\log(10)}-1} - C_2$)
T(n) is O($(c_1 + 2\,C_2)\,2^{\frac{\log(n)}{\log(10)}-1}$); rest rule
T(n) is O($2^{\frac{\log(n)}{\log(10)}-1}$); product rule
T(n) is O($2^{\frac{\log(n)}{\log(10)}}$); product rule

Recursion1Factorial

- **Code**

```
public int factorial(int n) {
    if(n<=0) return 1;
    return n*factorial(n-1);
}
```

- **Tags**

```
public int factorial(int n) {
    if(n<=0) return 1; // C1
    return n*factorial(n-1); // n*T(n-1)
}
```

- **Recurrence equation**

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ C_2 * T(n-1) & \text{if } n > 0 \end{cases}$$

- **Solution**

$T(n) = C_1C_2^{n-1}$

- **Notation Big O**

T(n) is O($C_1C_2^{n-1}$)
T(n) is O(C_2^{n-1}); product rule
T(n) is O(C_2^{n}); product rule

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

Recursion1SumDigits

- Code

```
public int sumDigits(int n) {
    if(n==0)return n;
    return (n%10)+sumDigits(n/10);
}
```

- Tags

```
public int sumDigits(int n) {
    if(n==0)return n; // C1
    return (n%10)+sumDigits(n/10); // C2 + T(n/10)
}
```

- Recurrence equation

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ C_2 + T(n/10) & \text{if } n > 0 \end{cases}$$

- Solution

$$T(n) = c_1 + \frac{C_2 \log(n)}{\log(10)}$$

- Notation Big O

T(n) is O($c_1 + \frac{C_2 \log(n)}{\log(10)}$)

T(n) is O($\frac{C_2 \log(n)}{\log(10)}$); sum rule

T(n) is O($C_2 \log(n)$); product rule

T(n) is O($\log(n)$); product rule

Recursion1Triangle

- Code

```
public int triangle(int rows) {
    if(rows==0) return 0;
```

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

```
        return rows+triangle(rows-1);
    }
```

- **Tags**

```
    public int triangle(int rows) {
        if(rows==0) return 0; // C1
        return rows+triangle(rows-1); // C2 + T(n-1)
    }
```

- **Recurrence equation**

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ c_2 + T(n-1) & \text{if } n > 0 \end{cases}$$

- **Solution**

$T(n) = C_1 + C_2 n$

- **Notation Big O**

$T(n)$ is $O(C_1 + C_2 n)$
$T(n)$ is $O(C_2 n)$; sum rule
$T(n)$ is $O(n)$; product rule

Recursion2GroupNoAdj

- **Code**

```
    public boolean groupNoAdj(int start, int[] nums, int target) {
        if (start >= nums.length) {
            return target == 0;
        }
        if (start + 1 < nums.length && nums[start + 1] == 1) {
            start++;
        }
        return groupNoAdj(start + 2, nums, target - nums[start]) || groupNoAdj(start + 1, nums,
target);

    }
```

- **Tags**

```
    public boolean groupNoAdj(int start, int[] nums, int target) {
        if (start >= nums.length) { // C1
            return target == 0;
        }
```

```
        if (start + 1 < nums.length && nums[start + 1] == 1) {
            start++;
        }
        return groupNoAdj(start + 2, nums, target - nums[start]) || groupNoAdj(start + 1, nums, target); // T(n-1) + T(n-2)

    }
```

- **Recurrence equation**

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ C_2 + T(n-1) + T(n-2) & \text{if } n > 0 \end{cases}$$

- **Solution**

$T(n) = C_1 * 2^{n-1}$

- **Notation Big O**

$T(n)$ is $O(C_1 * 2^{n-1})$
$T(n)$ is $O(2^{n-1})$; product rule
$T(n)$ is $O(2^n)$; product rule

Recursion2GroupSum5

- **Code**

```
public boolean groupSum5(int start, int[] nums, int target) {
        if(start >= nums.length) return target == 0;
        if(nums[start]%5==0){
          target -= nums[start];
          if(start+1<nums.length && nums[start+1]==1) start++;
          return groupSum5(start+1, nums, target);
        }else return groupSum5(start+1, nums, target - nums[start])|| groupSum5(start+1, nums, target);
    }
```

- **Tags**

```
public boolean groupSum5(int start, int[] nums, int target) {
        if(start >= nums.length) return target == 0; // C1
        if(nums[start]%5==0){
          target -= nums[start];
          if(start+1<nums.length && nums[start+1]==1) start++;
          return groupSum5(start+1, nums, target);
        }else return groupSum5(start+1, nums, target - nums[start])|| groupSum5(start+1, nums, target); //C2 + T(n-1) + T(n-1)
```

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

}

- Recurrence equation

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ C_2 + T(n-1) + T(n-1) & \text{if } n > 0 \end{cases}$$

- Solution

$T(n) = C_1 * 2^{n-1}$

- Notation Big O

$T(n)$ is $O(C_1 * 2^{n-1})$
$T(n)$ is $O(2^{n-1})$; product rule
$T(n)$ is $O(2^n)$; product rule

Recursion2GroupSum6

- Code

```
public boolean groupSum6(int start, int[] nums, int target) {
      if (start >= nums.length) {
         return target == 0;
      }
      if (nums[start] == 6) {
         target -= nums[start];
         if (start + 1 < nums.length && nums[start + 1] == 1) {
            start++;
         }
         return groupSum6(start + 1, nums, target);
      } else {
         return groupSum6(start + 1, nums, target - nums[start]) || groupSum6(start + 1, nums,
target);
      }
}
```

- Tags

```
public boolean groupSum6(int start, int[] nums, int target) {
      if (start >= nums.length) { // C1
         return target == 0;
      }
      if (nums[start] == 6) {
         target -= nums[start];
```

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

```
            if (start + 1 < nums.length && nums[start + 1] == 1) {
                start++;
            }
            return groupSum6(start + 1, nums, target);
        } else {
            return groupSum6(start + 1, nums, target - nums[start]) || groupSum6(start + 1, nums,
target); // C2 + T(n-1) + T(n-1)
        }
}
```

- **Recurrence equation**

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ C_2 + T(n-1) + T(n-1) & \text{if } n > 0 \end{cases}$$

- **Solution**

$T(n) = C_1 * 2^{n-1}$

- **Notation Big O**

$T(n)$ is $O(C_1 * 2^{n-1})$
$T(n)$ is $O(2^{n-1})$; product rule
$T(n)$ is $O(2^n)$; product rule

Recursion2SplitArray

- **Code**

```
public boolean aux(int start, int[] nums, int cont1, int cont2){
        return start >= nums.length ? cont1==cont2 : aux(start+1, nums, cont1+nums[start], cont2)||
                aux(start+1, nums, cont1, cont2+nums[start]);
}
```

- **Tags**

```
public boolean aux(int start, int[] nums, int cont1, int cont2){
        return start >= nums.length ? cont1==cont2 : aux(start+1, nums, cont1+nums[start], cont2)||
                aux(start+1, nums, cont1, cont2+nums[start]); // C1 + C2 T(n-1) + T(n-1)
}
```

- **Recurrence equation**

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ C_2 + T(n-1) + T(n-1) & \text{if } n > 0 \end{cases}$$

- Solution

$T(n) = C_1 * 2^{n-1}$

- Notation Big O

$T(n)$ is $O(C_1 * 2^{n-1})$
$T(n)$ is $O(2^{n-1})$; product rule
$T(n)$ is $O(2^n)$; product rule

Recursion2SplitOdd10

- Code

```
public boolean aux(int start, int[] nums, int cont1, int cont2){
    return start >= nums.length ? cont1%10==0 && cont2%2==1 : aux(start+1, nums,
            cont1+nums[start], cont2)|| aux(start+1, nums, cont1, cont2+nums[start]);
}
```

- Tags

```
public boolean aux(int start, int[] nums, int cont1, int cont2){
    return start >= nums.length ? cont1%10==0 && cont2%2==1 : aux(start+1, nums,
            cont1+nums[start], cont2)|| aux(start+1, nums, cont1, cont2+nums[start]); //C1+C2+
                                                                                      T(n-1)+T(n-1)
}
```

- Recurrence equation

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ C_2 + T(n-1) + T(n-1) & \text{if } n > 0 \end{cases}$$

- Solution

$T(n) = C_1 * 2^{n-1}$

- Notation Big O

$T(n)$ is $O(C_1 * 2^{n-1})$

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

T(n) is $O(2^{n-1})$; product rule
T(n) is $O(2^n)$; product rule

3. **Meaning of 'n' and 'm'**

In the previous exercises you can notice that 'n' is the number of iterations that a program has during its process. Each time a recursive call is made, this 'n' is decreased or increased, depending on the case, until the fulfillment of the stop condition is reached. On the other hand, 'm' is an entry that also refers to the number of iterations and the execution of the program, however the most common use of this entry is to set a limit or interval with 'n', which is the main entrance of the program, in these cases you can get to work with loops and establish that the loop will start at 'n' and end at 'm' or vice versa.

4. What was learned about stack overflow is that this is an error that can occur in the programs because the memory that is established for this program is exceeded. The stack is a buffer that stores the requests that need to be handled and when the memory limit is exceeded, there is not enough space for these instructions. This problem can happen due to multiple events, for example:
The architecture of the computer on which the program is running, the language in which the program is written, and the total amount of memory available in the system.
This problem can be solved by increasing the capacity of the memory that the program has by means of an instruction. [11]

5. The largest value that could be calculated for the Fibonacii sequence was 100 because the program did not calculate more with quite large values, and this is due to some main aspects:
- The memory capacity that is defined for the program cannot cope and this generates the error, StackOverFlow.
- The processor of the program does not have the capacity to calculate high values and this generates that sometimes the process of the program becomes slow.
- The complexity of the Fibonacci program is quite large and it is n^2.

6. One of the options that can be used to calculate Fibonacci for large values, in addition to increasing the memory space that is given to the program, is to try to execute the process in parts, in other words, the process is done up to a certain part of the program. n and then keep saving the total amount of the sum that is carried at that moment, except for the last two calculations, so that the series can be continued. With this option, by constantly keeping concise data elsewhere outside the execution, you can get the execution to begin its process again, but with different initial values, this would avoid the high accumulation of instructions in the stacks.

4) Practice for midterms

1. Start + 1, nums, target
2. B) 2T(n/2)+C
3. Line 4: Solucionar(n-a, a, b, c) + 1
   Line 5: Math.max(res, Solucionar(n-b, a, b, c) +1)
   Line 6: Math.max(res, Solucionar(n-c, a, b, c) +1)

[1] Based on: https://whatis.techtarget.com/definition/stack-overflow

4. E) La suma de los elementos del arreglo a y es O(n)
5. Line 2: return n;
   Line 3: n-1
   Line 4: n-2
   T(n) = T(n-1) + T(n-2) + C
6. Line 10: sumaAux(n, i+2)
   Line 12: sumaAux(n, i+1)
7. Line 9: comb(S, i+1, t-S[i])
   Line 10: comb(S, i+1, t)
8. Line 9: return 0;
   Line 13: ni + nj;

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