ESTRUCTURA DE DATOS 1
Código ST0245

Laboratorio Nro. 1
Recursión

Esteban Gonzalez Tamayo
Universidad Eafit
Medellín, Colombia
egonzalezt@eafit.edu.co

David Felipe Garcia Contreras
Universidad Eafit
Medellín, Colombia
dfgarciac1@eafit.edu.co

2) Simulacro de Maratón de Programación sin documentación HTML, en GitHub

2.1
2.1.1

El ejercicio factorial funciona recibiendo un valor a calcular, el factorial funciona de la siguiente manera se necesita multiplicar el valor deseado "3" con el mismo pero un valor menos "3-1" hasta que se llegue a 1 y así obtener el valor deseado por lo cual si se desea aplicar en código se puede realizar recursivamente usando n*factorial(n-1), aplicando el ejemplo de 3 el código funcionaria de la siguiente manera 3*factorial(2) → (3*2)*factorial(1) → (3*2*1)*factorial(0) → como n vale cero retorna 1 y todos los valores acumulados → 3*2*1=6.

2.1.2

El ejercicio bunnyEars nos indica que por cada conejo que hay cada uno tiene 2 oídos por lo cual en este ejercicio si hay dos conejos significa que hay 4 oídos, el código lo que realiza es que por cada llamado recursivo que se le haga al código, este retornara el número de oídos del conejo y también disminuirá la cantidad de conejos, si nos entregan 3 conejos el código realizara lo siguiente 2+bunnyEars(3) → 2+bunnyEars(2) → 2+bunnyEars(1) → cuando el código llega a 0 finaliza el proceso y luego retorna el número de oídos = 6.

2.1.3

Este ejercicio bunnyEars2, los conejos están en fila y cada uno tiene un número, cuando el número del conejo es impar es porque el conejo posee dos oído y si es par el conejo posee 3 oídos, el este código se rige por lo siguiente: cuando el número del conejo no es par, este tiene 2 oídos, de lo contrario posee 3 oídos, el método recursivo se detiene y retorna el número de oídos contados, si hay 3 conejos enumerados el código realiza lo siguiente 2+bunnyEars2(3) → 3+bunnyEars2(2) → 2+bunnyEars2(1) → 2+3+2 = 7, fin del método.

2.1.4

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

El ejercicio sumDigits tiene como objetivo sumar cada digito de un valor recibido, para esto se necesitará realizar el módulo(%) para obtener el ultimo digito, luego al dividirlo por 10 lo que está haciendo es convertir el penúltimo en el último y el ultimo omite, si se realiza la suma de 123 se realiza de la siguiente manera 123 % 10 + sumDigits(123 / 10) → 3 + sumDigits(12) → 2 + sumDigits(1) → 1 + sumDigits(0), luego suma los valores dando como resultado 6 y al llegar a cero el método finaliza.

2.1.5

El ejercicio powerN calcula la potencia de $base^n$, para esto la base no se puede modificar dentro del método powerN. La recursión debe disminuir a n para que así cuando llegue a 1 este retorne el valor de la base, la base se debe multiplicar por fuera del método como: base*powerN(base, n-1) para que en cada llamado recursivo esta se multiplique por misma. Si calculamos la potencia de 3 elevado a la 3 seria 3*powerN(3,3) → 3*powerN(3,2) → 3*powerN(3,1) y cuando llegue a 1 el proceso termina y luego calcula la potencia 3*3*3 =27.

2.2.1

El ejercicio Groupsum5 se trata de sumar los números contenidos en un arreglo hasta lograr un objetivo, este tiene una condición que se debe cumplir, para este algoritmo se realizó varias comparaciones para verificar si hay múltiplos de 5 y si el número que le sigue no es uno, cuando esto sucede se llama recursivamente de la siguiente forma return groupSum5(start+1, nums, target - nums[start]) donde aquí start es +1 para incluir el número que sigue, de lo contrario se realizara lo mismo con la diferencia de que se saltara el 1

2.2.2

El ejercicio groupNoAdj se trata acerca de avanzar en el arreglo de a dos números mirando si cumple con el objetivo(target) y devuelve verdadero si no cumple avanza de a uno sin cambiar el target devolviendo true si se cumple de lo contrario si ambas cosas no se cumplen devuelve falso

Tomado de:
Alfredo Miranda (2013) groupNoAdj.java [Source code]
https://github.com/mirandaio/codingbat/blob/master/java/recursion-2/groupNoAdj.java

2.2.3

El ejercicio groupSum6 consiste en alcanzar un objetivo dado, pero con la condicion de que si algún valor del arreglo es un 6 este debe ser incluido para luego ser operado y verificar si el valor obtenido es el deseado. Por lo cual este debe realizar dos decisiónes: 1) cuando es 6 y 2) para cualquier valor diferente de 6.

Tomado de:
Alfredo Miranda (2013) groupSum6.java[Source code]
https://github.com/mirandaio/codingbat/blob/master/java/recursion-2/groupSum6.java

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

3) Simulacro de preguntas de sustentación de Proyectos

3.1

//T(n)=C1+C2
//T(n)=C3+T(n-2)+(n+1)
R: T(n)es O(2^n)

3.2

Operación	Tiempo(ms)
1	1
2	1
3	0
4	1
5	2
6	3
7	5
8	8
9	13
10	23
11	35
12	57
13	101
14	162
15	246
16	408
17	635
18	1028
19	1637
20	2692

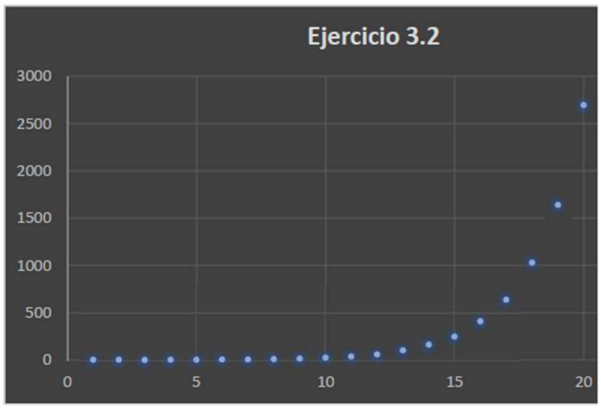


Imagen 1

A medida que n aumenta la recursión vuelve y se repite para hacer cumplir el caso base donde si se toma en cuenta con 20 datos obtienen los valores de la imagen 1 por lo cual con n = 50 este puede demorarse el doble de lo que tardo con 20 datos ya que como se observa en la imagen 1 este crece exponencialmente

3.3

No ya que su Notación asintótica es exponencial la cual implica que por cada vez que n aumenta más tiempo se va a demorar el código, tomando esto como referencia para un puerto donde n es un número muy grande se tornaría lenta la ejecución del código.

3.4

Este ejercicio funciona realizado varias condiciones, ya que este necesita comprobar primero si el numero 5 no está de ultimo, luego como lo indica el ejercicio si después del 5 hay un 1 este se deberá saltar al número que le sigue para poder restarse o sumarse hasta alcanzar

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

el target. Por lo cual este necesitará de dos condiciones la cual verificara si hay un múltiplo de 5 para restarle al target ya que el 5 se debe incluir en la operación, por lo cual deberá verificar si después del 5 hay un 1 para realizar el salto, de lo contrario no realizara el salto.

3.5

Ejercicios Numeral 2.1:

1)

```
   public int factorial (int n)
{
       if(n==0)
       {
          return 1;// T(n) = c1
       } else return n*factorial(n-1);
}
             T(n) = c2 + T(n-1)
```

n es el número del factorial
Ejemplo: n=4 entonces seria calcular Factorial de 4
Por Wólfram:
Resolviendo en Wólfram: T(n) = c2*n + c1
T(n) es O (c2*n + c1), por definición de O
T(n) es O(c2*n), por regla de la suma
T(n) es O(n)

2)

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

n es el número de conejos (Veces que se va a repetir el algoritmo)

```
   }
      T(n)= C1
      T(n)=C2+T(n-1)
```

Por Wolfram: T(n) = c2*n + c1
// T(n) es O (c2*n + c1), por definición de O
// T(n) es O(c2*n), por regla de la suma

3)

```
   public int bunnyEars2(int bunnies) {
       if(bunnies==0)
       {
```

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
            return 0;//C1
        }
        if (bunnies%2! =0)
        {
            return 2+bunnyEars2(bunnies-1);
  //C2+T(n-1)
        }
        //C3+T(n-1)
        else return 3+bunnyEars2(bunnies-1);
    }
       T(n)=C1
       T(n)=C2+T(n-1)
       T(n)=C3+T(n-1)
```

Por Wolfram:

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

```
Wolfram= c2*log10 n
 // T(n) es O (c2*log10 n), Definición O
 // T(n) es O (log n), Regla del producto
```

```
5)   public int powerN (int base, int n) {
        if(n<=1)
        {
            return base;//C1
        }
        else return base*powerN(base, n-1);
        //C2+T(n-1)
    }
}
T(n)=C1
T(n)=C2+T(n-1)
Por Wolfram: T(n) = c2*n + c1
 // T(n) es O (c2*n + c1), por definición de O
 // T(n) es O(c2*n), por regla de la suma
```

Ejercicios Numeral 2.2:

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

1)

```
public boolean groupSum5(int start, int [] nums, int target) {
    if (start >= nums.length)
        return target == 0;//T(n,m,z)=C1

    if(start<nums.length-1)
    {
    if(nums[start]%5==0&&nums[start+1]! =1)
        return groupSum5(start+1, nums, target - nums[start]);
/ T (n, m, z) =C2+T (n+1+m+ (z-m * n))
    }
    if(nums[start]%5==0)
        return groupSum5(start+2, nums, target - nums[start]);
T (n, m, z) = C3+T (n+2+m+ (z-m * n))
    If (groupSum5(start+1, nums, target - nums[start]))
        return true ;// T (n, m, z) = C4

    if (groupSum5(start+1, nums, target))
        return true;// T (n, m, z) = C5
    return false   ;// T (n, m, z) = C6
}
T (n, m, z) =C1
T (n, m, z) =C2+T (n+1+m+ (z-m * n))
T (n, m, z) = C3+T (n+2+m+ (z-m * n))
T (n, m, z) = C4+C5+C6
```

2) **p**ublic boolean groupSum6(int start, int [] nums, int target) {

```
    If (start >= nums.length)
        return target == 0; T (n, m, z) =C1

    if(nums[start] == 6)
        return groupSum6(start+1, nums, target - 6);
         T (n, m, z) =C2+T (N+1+M+ (Z-6))

    if (groupSum6(start+1, nums, target - nums[start]))
        return true;
T (n, m, z) =C3


    If (groupSum6(start+1, nums, target))
        return true; T (n, m, z) =C4
```

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
    return false; T (n, m, z) =C5

}
```

Alfredo Miranda(2013) groupSum6.java[Source code]
https://github.com/mirandaio/codingbat/blob/master/java/recursion-2/groupSum6.java

```
3) public boolean groupNoAdj(int start, int[] nums, int target) {

  if (start >= nums.length)

    return (target == 0); T(n)=C1


  if (groupNoAdj(start+1, nums, target)) return true;
T(n)=C2

  if (groupNoAdj(start+2, nums, target-nums[start])) return true;
T(n)=C3

  return false;
T(n)=C4

T(n)=C1+C2+C3+C3+C4

}
```

Alfredo Miranda(2013) groupNoAdj.java [Source code]
https://github.com/mirandaio/codingbat/blob/master/java/recursion-2/groupNoAdj.java

3.6
3.6.1
Factorial: n es el número de la factorial
3.6.2
bunnyEars: n es el número de conejos (Veces que se va a repetir el algoritmo)
3.6.3
bunnyEars2: n es el número de conejos (Veces que se va a repetir el algoritmo).
3.6.4
sumDigits: n es el numero que recibe la función donde esta se busca separarlos del ultimo hasta el primero para hacer la suma de cada digito.
3.6.5
n va a ser considerado como la potencia la que va a hacer repetir tantas veces la recursión

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

3.6.6
N = número con el cual se empieza
M= Arreglo que contiene los números que se buscan sumar
Z= Objetivo al cual se busca llegar

3.6.7
N = número con el cual se empieza
M= Arreglo que contiene los números que se buscan sumar
Z= Objetivo al cual se busca llegar

4) Simulacro de Parcial

4.1 *a*
4.2 *b*

4.2.1 *length-1*
4.3

4.5.1
Linea1: if(n<=2) n;
Linea2: return formas(n-1) +
Linea3: formas(n-2);
4.5.2
b. T(n)=T(n-1) +T(n-2) + C

4.6
4.6.1
return suma (numero, i+2);
4.6.2
suma (numero, i+1);

4.8
4.8.1
return 2;
4.8.2
int suma= ni +nj;

4.10
b.6

4.11
n - 1 + lucas (n - 2);

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

4.11.2
c.O(2^n)
4.12
4.12.1
sat
4.12.2
Math.masx(fi+fj);
4.12.3
sat

5) Recursión and Back tracking

5.1

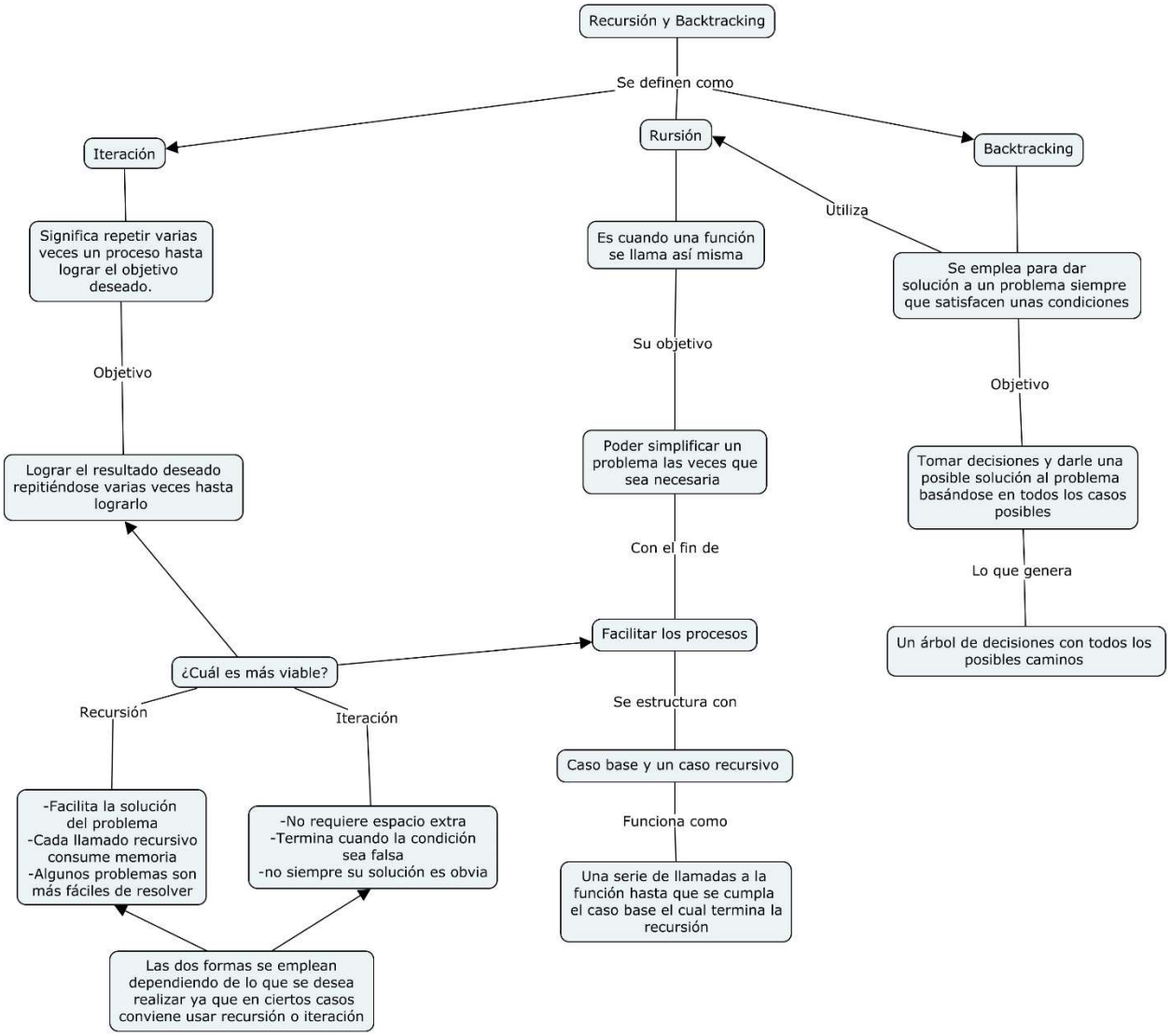
Resumen: La recursión se trata acerca de toda función que se puede llamar a si misma la cual puede tener mas de un llamado recursivo. Cada vez que se llama a la recursión se esa dividiendo en problemas el cual el objetivo es hacer que se cumpla se cumpla. Es importante mirar que esta no es la única manera como puede ser la iteración es una alternativa por sus ventajas sobre la recursión una de ellas siendo no requerir de un espacio extra en memoria.

Si se profundiza un poco en la recursión se puede encontrar un método que es el Back tracking la cual genera posibles opciones (distintas rutas de complejidad, una cosa característica de esta misma es como significa que se puede devolver si la solución no resulto se escogerá una alternativa.

5.2 Mapa conceptual

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

ESTRUCTURA DE DATOS 1
Código ST0245



PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473