Laboratorio Nro. 1
Recursión

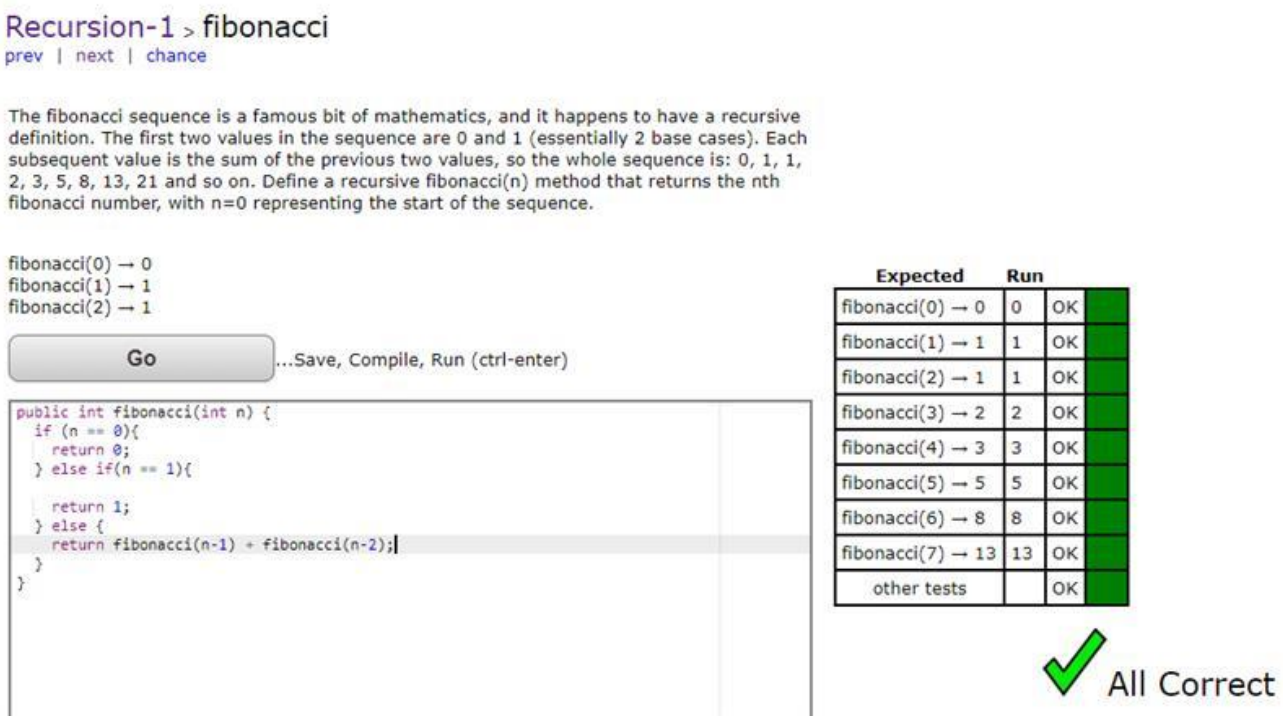
ESTRUCTURA DE DATOS 1
Código ST0245

Resumen de los ejercicios a resolver

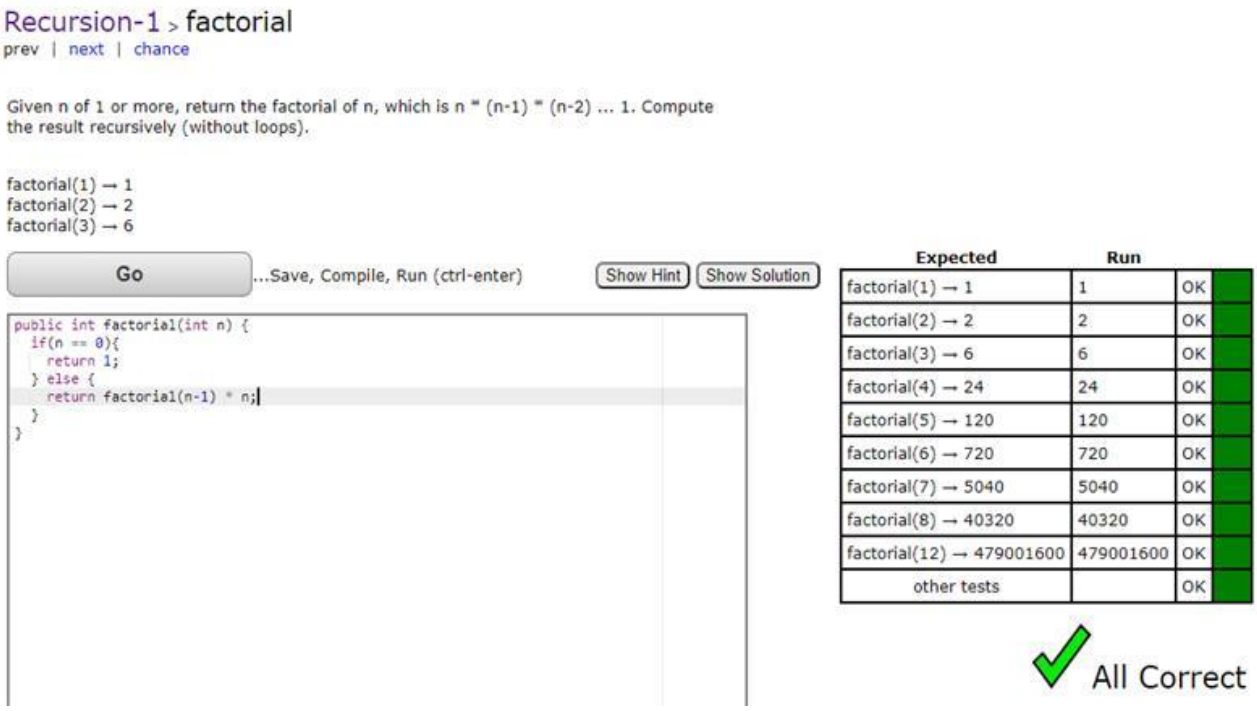
1.1 Implementen un algoritmo que calcule la longitud de la subsecuencia común (en el mismo orden relativo), más larga, entre dos cadenas de caracteres. Y probarla con los conjuntos de datos (en inglés, *datasets*) incluidos en github.
Se encuentra en github.

1.2 Implementen un algoritmo recursivo para encontrar de cuántas formas se puede llenar un rectángulo de 2xn cm^2 con rectángulos de 1x2 cm^2.
Se encuentra en github

2.1 Resuelvan --al menos-- 5 ejercicios del nivel *Recursion* 1 de CodingBat: http://codingbat.com/java/Recursion-1



$T(n) = C + T(n-1) + T(n-2)$



$T(n) = C + T(n-1)$

Recursion-1 > triangle

prev | next | chance

We have triangle made of blocks. The topmost row has 1 block, the next row down has 2 blocks, the next row has 3 blocks, and so on. Compute recursively (no loops or multiplication) the total number of blocks in such a triangle with the given number of rows.

triangle(0) → 0
triangle(1) → 1
triangle(2) → 3

Go ...Save, Compile, Run (ctrl-enter)

```
public int triangle(int rows) {
  if(rows == 0){
    return 0;
  } else {
    return rows + triangle(rows-1);
  }
}
```

Expected	Run		
triangle(0) → 0	0	OK	
triangle(1) → 1	1	OK	
triangle(2) → 3	3	OK	
triangle(3) → 6	6	OK	
triangle(4) → 10	10	OK	
triangle(5) → 15	15	OK	
triangle(6) → 21	21	OK	
triangle(7) → 28	28	OK	
other tests		OK	

All Correct

T(n) = C + T(n-1)

Recursion-1 > bunnyEars2

prev | next | chance

We have bunnies standing in a line, numbered 1, 2, ... The odd bunnies (1, 3, ..) have the normal 2 ears. The even bunnies (2, 4, ..) we'll say have 3 ears, because they each have a raised foot. Recursively return the number of "ears" in the bunny line 1, 2, ... n (without loops or multiplication).

bunnyEars2(0) → 0
bunnyEars2(1) → 2
bunnyEars2(2) → 5

Go ...Save, Compile, Run (ctrl-enter)

```
public int bunnyEars2(int bunnies) {
  if(bunnies == 0){
    return 0;
  } else if( bunnies % 2 == 1){
    return 2 + bunnyEars2(bunnies - 1);
  } else {
    return 3 + bunnyEars2(bunnies - 1);
  }
}
```

Expected	Run		
bunnyEars2(0) → 0	0	OK	
bunnyEars2(1) → 2	2	OK	
bunnyEars2(2) → 5	5	OK	
bunnyEars2(3) → 7	7	OK	
bunnyEars2(4) → 10	10	OK	
bunnyEars2(5) → 12	12	OK	
bunnyEars2(6) → 15	15	OK	
bunnyEars2(10) → 25	25	OK	
other tests		OK	

All Correct

T(n) = C + 2 * T(n-1)

2.2 Resuelvan --al menos-- 5 ejercicios del nivel *Recursion* 2 de la página CodingBat: http://codingbat.com/java/Recursion-2

Recursion-2 > groupSum

prev | next | chance

Given an array of ints, is it possible to choose a group of some of the ints, such that the group sums to the given target? This is a classic backtracking recursion problem. Once you understand the recursive backtracking strategy in this problem, you can use the same pattern for many problems to search a space of choices. Rather than looking at the whole array, our convention is to consider the part of the array starting at index **start** and continuing to the end of the array. The caller can specify the whole array simply by passing start as 0. No loops are needed -- the recursive calls progress down the array.

groupSum(0, [2, 4, 8], 10) → true
groupSum(0, [2, 4, 8], 14) → true
groupSum(0, [2, 4, 8], 9) → false

Go ...Save, Compile, Run (ctrl-enter) Show Hint Show Solution

```
public boolean groupSum(int start, int[] nums, int target) {
  if(start >= nums.length){
    return target == 0;
  } else if(groupSum(start + 1, nums, target - nums[start])) {
    return true;
  } else if(groupSum(start + 1, nums, target)) {
    return true;
  } else {
    return false;
  }
}
```

Expected	Run		
groupSum(0, [2, 4, 8], 10) → true	true	OK	
groupSum(0, [2, 4, 8], 14) → true	true	OK	
groupSum(0, [2, 4, 8], 9) → false	false	OK	
groupSum(0, [2, 4, 8], 8) → true	true	OK	
groupSum(1, [2, 4, 8], 8) → true	true	OK	
groupSum(1, [2, 4, 8], 2) → false	false	OK	
groupSum(0, [1], 1) → true	true	OK	
groupSum(0, [9], 1) → false	false	OK	
groupSum(1, [9], 0) → true	true	OK	
groupSum(0, [], 0) → true	true	OK	
groupSum(0, [10, 2, 2, 5], 17) → true	true	OK	
groupSum(0, [10, 2, 2, 5], 15) → true	true	OK	
groupSum(0, [10, 2, 2, 5], 9) → true	true	OK	
other tests		OK	

All Correct

$T(n) = C + 2 * T(n-1)$

Recursion-2 > groupSum6

prev | next | chance

Given an array of ints, is it possible to choose a group of some of the ints, beginning at the start index, such that the group sums to the given target? However, with the additional constraint that all 6's must be chosen. (No loops needed.)

groupSum6(0, [5, 6, 2], 8) → true
groupSum6(0, [5, 6, 2], 9) → false
groupSum6(0, [5, 6, 2], 7) → false

Go ...Save, Compile, Run (ctrl-enter)

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

Go

Editor font size %: 100

Shorter output

Expected	Run		
groupSum6(0, [5, 6, 2], 8) → true	true	OK	
groupSum6(0, [5, 6, 2], 9) → false	false	OK	
groupSum6(0, [5, 6, 2], 7) → false	false	OK	
groupSum6(0, [1], 1) → true	true	OK	
groupSum6(0, [9], 1) → false	false	OK	
groupSum6(0, [], 0) → true	true	OK	
groupSum6(0, [3, 2, 4, 6], 8) → true	true	OK	
groupSum6(0, [6, 2, 4, 3], 8) → true	true	OK	
groupSum6(0, [5, 2, 4, 6], 9) → false	false	OK	
groupSum6(0, [6, 2, 4, 5], 9) → false	false	OK	
groupSum6(0, [3, 2, 4, 6], 3) → false	false	OK	
groupSum6(0, [1, 6, 2, 6, 4], 12) → true	true	OK	
groupSum6(0, [1, 6, 2, 6, 4], 13) → true	true	OK	
groupSum6(0, [1, 6, 2, 6, 4], 4) → false	false	OK	
groupSum6(0, [1, 6, 2, 6, 4], 9) → false	false	OK	
groupSum6(0, [1, 6, 2, 6, 5], 14) → true	true	OK	
groupSum6(0, [1, 6, 2, 6, 5], 15) → true	true	OK	
groupSum6(0, [1, 6, 2, 6, 5], 16) → false	false	OK	
other tests		OK	

All Correct

$T(n) = C + 2 * T(n-1)$

Recursion-2 > groupNoAdj

prev | next | chance

Given an array of ints, is it possible to choose a group of some of the ints, such that the group sums to the given target with this additional constraint: If a value in the array is chosen to be in the group, the value immediately following it in the array must not be chosen. (No loops needed.)

groupNoAdj(0, [2, 5, 10, 4], 12) → true
groupNoAdj(0, [2, 5, 10, 4], 14) → false
groupNoAdj(0, [2, 5, 10, 4], 7) → false

Go ...Save, Compile, Run (ctrl-enter)

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

	Expected	Run	
groupNoAdj(0, [2, 5, 10, 4], 12) → true	true	OK	
groupNoAdj(0, [2, 5, 10, 4], 14) → false	false	OK	
groupNoAdj(0, [2, 5, 10, 4], 7) → false	false	OK	
groupNoAdj(0, [2, 5, 10, 4, 2], 7) → true	true	OK	
groupNoAdj(0, [2, 5, 10, 4], 9) → true	true	OK	
groupNoAdj(0, [10, 2, 2, 3, 3], 15) → true	true	OK	
groupNoAdj(0, [10, 2, 2, 3, 3], 7) → false	false	OK	
groupNoAdj(0, [], 0) → true	true	OK	
groupNoAdj(0, [1], 1) → true	true	OK	
groupNoAdj(0, [9], 1) → false	false	OK	
groupNoAdj(0, [9], 0) → true	true	OK	
groupNoAdj(0, [5, 10, 4, 1], 11) → true	true	OK	
other tests		OK	

All Correct

$T(n) = C + T(n-1)$

Recursion-2 > splitArray

prev | next | chance

Given an array of ints, is it possible to divide the ints into two groups, so that the sums of the two groups are the same. Every int must be in one group or the other. Write a recursive helper method that takes whatever arguments you like, and make the initial call to your recursive helper from splitArray(). (No loops needed.)

splitArray([2, 2]) → true
splitArray([2, 3]) → false
splitArray([5, 2, 3]) → true

Go ...Save, Compile, Run (ctrl-enter)

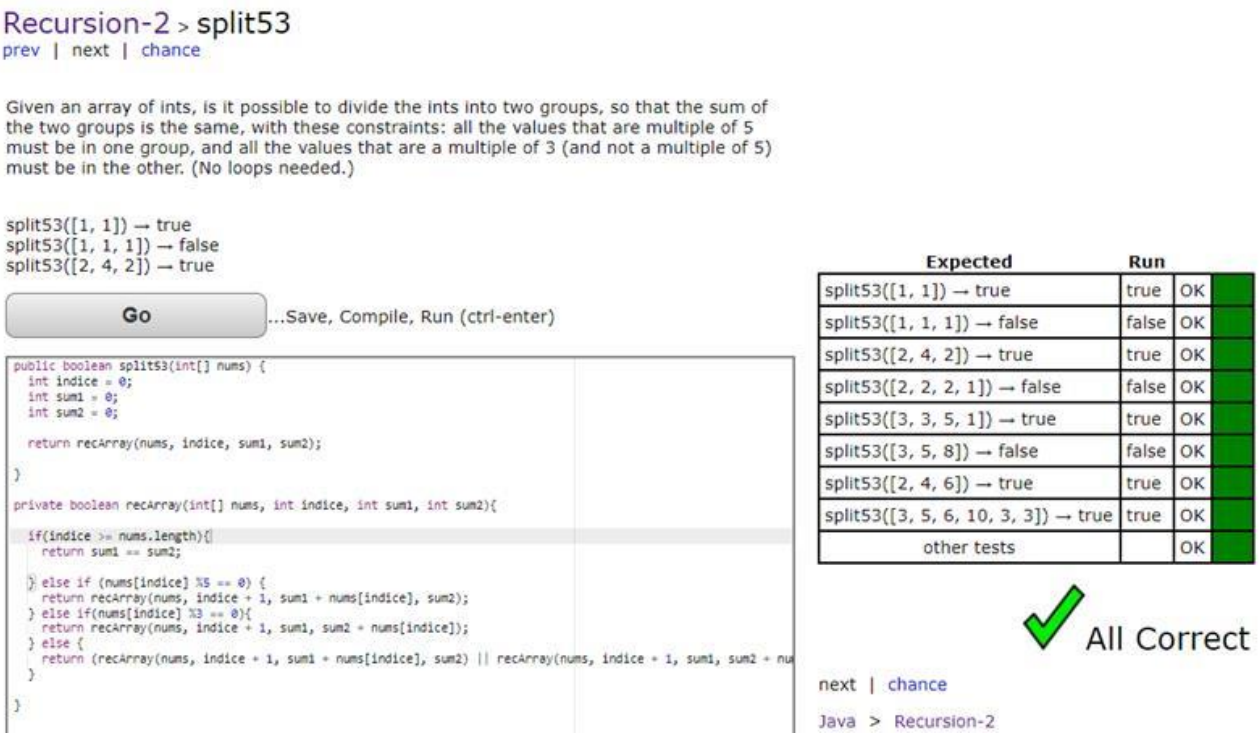
```
public boolean splitArray(int[] nums) {
  int indice = 0;
  int sum1 = 0;
  int sum2 = 0;

  return recArray(nums, indice, sum1, sum2);



}

private boolean recArray(int[] nums, int indice, int sum1, int sum2){
  if(indice >= nums.length){
    return sum1 == sum2;
  } else {
    int valor = nums[indice];
    return recArray(nums, indice + 1, sum1 + valor, sum2) || recArray(nums, indice + 1, sum1, sum2 + valor);
  }
}
```

Expected	Run		
splitArray([2, 2]) → true	true	OK	
splitArray([2, 3]) → false	false	OK	
splitArray([5, 2, 3]) → true	true	OK	
splitArray([5, 2, 2]) → false	false	OK	
splitArray([1, 1, 1, 1, 1, 1]) → true	true	OK	
splitArray([1, 1, 1, 1, 1]) → false	false	OK	
splitArray([]) → true	true	OK	
splitArray([1]) → false	false	OK	
splitArray([3, 5]) → false	false	OK	
splitArray([5, 3, 2]) → true	true	OK	
splitArray([2, 2, 10, 10, 1, 1]) → true	true	OK	
splitArray([1, 2, 2, 10, 10, 1, 1]) → false	false	OK	
splitArray([1, 2, 3, 10, 10, 1, 1]) → true	true	OK	
other tests		OK	

All Correct

next | chance

$T(n) = C + T(n-1)$



Recursion-2 > split53

prev | next | chance

Given an array of ints, is it possible to divide the ints into two groups, so that the sum of the two groups is the same, with these constraints: all the values that are multiple of 5 must be in one group, and all the values that are a multiple of 3 (and not a multiple of 5) must be in the other. (No loops needed.)

split53([1, 1]) → true
split53([1, 1, 1]) → false
split53([2, 4, 2]) → true

Go ...Save, Compile, Run (ctrl-enter)

```
public boolean split53(int[] nums) {
  int indice = 0;
  int sum1 = 0;
  int sum2 = 0;

  return recArray(nums, indice, sum1, sum2);

}

private boolean recArray(int[] nums, int indice, int sum1, int sum2){

  if(indice >= nums.length){
    return sum1 == sum2;

  } else if (nums[indice] %5 == 0) {
    return recArray(nums, indice + 1, sum1 + nums[indice], sum2);
  } else if(nums[indice] %3 == 0){
    return recArray(nums, indice + 1, sum1, sum2 + nums[indice]);
  } else {
    return (recArray(nums, indice + 1, sum1 + nums[indice], sum2) || recArray(nums, indice + 1, sum1, sum2 + nu
  }

}
```

Expected	Run		
split53([1, 1]) → true	true	OK	
split53([1, 1, 1]) → false	false	OK	
split53([2, 4, 2]) → true	true	OK	
split53([2, 2, 2, 1]) → false	false	OK	
split53([3, 3, 5, 1]) → true	true	OK	
split53([3, 5, 8]) → false	false	OK	
split53([2, 4, 6]) → true	true	OK	
split53([3, 5, 6, 10, 3, 3]) → true	true	OK	
other tests		OK	

All Correct

next | chance

Java > Recursion-2

$T(n) = C + T(n-1)$

3.1. Calculen la complejidad, para el peor de los casos, del ejercicio 1.1

$T(n) = C + 2*T(n-1) + T(n-2)$

3.3. ¿La complejidad del algoritmo del ejercicio 1.1 es apropiada para encontrar la subsecuencia común más larga entre ADNs mitocondriales como los de los *datasets*?
No es apropiado para grandes cadenas por que la complejidad es lineal.

3.5. Calculen la complejidad de los **Ejercicios en Línea** de los numerales 2.1 y 2.2
Ya está hecho arriba.

4. Simulacro de Parcial

4.1 Una de las tecnologías cruciales para que la Cuarta Revolución Industrial tenga éxito es la ciberseguridad. Una de las áreas más importantes de la ciberseguridad es la criptografía. Criptografía es el arte y técnica de escribir claves secretas de tal forma que lo escrito solamente sea inteligible para quien sepa descifrarlo. Consideremos una aplicación de criptografía. Algunas palabras del alfabeto español se pueden escribir como la unión consecutiva de los símbolos de la tabla periódica universal.

Por ejemplo, la palabra “Población” se puede escribir como la unión de los símbolos {Po, B, La, C, I, O, N}, PoBLaCION. Te entregan una cadena de caracteres S y un conjunto de símbolos T. El objetivo es determinar si S se puede escribir como la unión consecutiva de cero o más símbolos de T. Por ejemplo, sea T={Ti,B,I,O,C} y S= “Biótico”, la respuesta es verdadero; para S= “”, la respuesta es verdadero; y para S=“Titan”, la respuesta es falso. El siguiente código resuelve el problema, pero le faltan algunas líneas; por favor, complétalas. ¡Gracias!

Puedes asumir que T contiene todas las posibles combinaciones de acentos, mayúsculas y minúsculas de cada símbolo de la tabla periódica. En Java, el método s.substring(a,b) retorna la subcadena de s entre los índices a y b−1, incluídos. El método t.contains(s) retorna verdadero si la cadena s se encuentra dentro de t; de lo contrario, falso.

```
1    boolean solve(String s,List<String> t){
2       return solve(t, s, s.length());
3    }
4    boolean solve(List<String> t, String s,
         int n){
5       for(int i = 1; i <= n; ++i){
6         String pfx = ..................;
7         if(t.contains(pfx)){
8           if(i == n){
9             return ............;
10          }
11          return ......................;
12        }
13      }
14      return false;
15   }
```

1. Completa la línea 6 .....
   a. `s.substring(0, i)`
   b. `s.substring(0, n)`
   c. `s.substring(i, n)`
2. Completa la línea 9 .....
   a. `false`
   b. `s.substring(0, i)`
   c. `true`
3. Completa la línea 11 .....
   a. `solve(t, s.substring(i),  n - i)`
   b. `solve(pfx, t),  n - i)`
   c. `solve(t, s.substring(n),  I - n)`

4.2 Rellenar por difusión se utiliza para implementar el tarrito de *Microsoft Paint*. Además, según el portal *Geeks for Geeks*, es un problema –muy frecuente– en entrevistas de Amazon, Microsoft y Adobe. El problema es el siguiente. Dado una matriz de dimensión *N*×*M* (`screen`), la ubicación de un pixel en la matriz (*x*,*y*) y un color (`newC` ∈*N*), nuestra tarea es reemplazar el color (`prevC` ∈*N*) del pixel (*x*,*y*) –y el de todos sus vecinos que tengan el color `prevC`– con el color `newC`. **¡Incluyamos los vecinos diagonales!** Como un ejemplo, si tenemos *x*=4,*y*=4,*prevC*=2,*newC*=3 y la matriz (`screen`) en el lado izquierdo, después de aplicar el algoritmo, debe lucir como la del lado derecho. Faltan unas líneas, complétalas, por favor.

```
{{1, 1, 1, 1, 1, 1, 1, 1},    {{1, 1, 1, 1, 1, 1, 1, 1},
{1, 1, 1, 1, 1, 1, 0, 0},     {1, 1, 1, 1, 1, 1, 0, 0},
{1, 0, 0, 1, 1, 0, 1, 1},     {1, 0, 0, 1, 1, 0, 1, 1},
{1, 2, 2, 2, 2, 0, 1, 0},     {1, 3, 3, 3, 3, 0, 1, 0},
{1, 1, 1, 2, 2, 0, 1, 0},     {1, 1, 1, 3, 3, 0, 1, 0},
{1, 1, 1, 2, 2, 2, 2, 0},     {1, 1, 1, 3, 3, 3, 3, 0},
{1, 1, 1, 1, 1, 2, 1, 1},     {1, 1, 1, 1, 1, 3, 1, 1},
{1, 1, 1, 1, 1, 2, 2, 1},};   {1, 1, 1, 1, 1, 3, 3, 1},};
```

Si trabajas en Java, revisa este código:

```
1  void floodFillUtil(int screen[][], int x, int
       y, int prevC, int newC, int N, int M) {
2   if (x < 0 || x >= M || y < 0 || y >= N)
3         return;
4   if (screen[x][y] != prevC)
5         return;
6
7   screen[x][y] = newC;
8
9   ....
10  ....
11  ....
12  ....
13  floodFillUtil(screen, x+1, y, prevC, newC, N,
        M);
14  floodFillUtil(screen, x-1, y, prevC, newC, N,
        M);
15  floodFillUtil(screen, x, y+1, prevC, newC, N,
        M);
16  floodFillUtil(screen, x, y-1, prevC, newC, N,
        M); }
17
18 void floodFill(int screen[][], int x, int y,
      int newC, int N, int M) {
19  int prevC = screen[x][y];
20  floodFillUtil(screen, x, y, prevC, newC, N, M
        ); }
```

Si trabajas en Python, revisa este código:

```
1   def floodFillUtil(screen, x, y, prevC, newC, N
        , M):
2     if x < 0 or x >= M or y < 0 or y >= N or
          screen[x][y] != prevC:
3           return
4     if screen[x][y] == newC:
5           return
6
7     screen[x][y] = newC
8
9     ....
10    ....
11    ....
12    ....
13    floodFillUtil(screen, x + 1, y, prevC, newC,
          N, M)
14    floodFillUtil(screen, x - 1, y, prevC, newC,
          N, M)
15    floodFillUtil(screen, x, y + 1, prevC, newC,
          N, M)
16    floodFillUtil(screen, x, y - 1, prevC, newC,
          N, M)
17
18  def floodFill(screen, x, y, newC, N, M):
19    prevC = screen[x][y]
20    floodFillUtil(screen, x, y, prevC, newC, N, M
          )
```

1. El código esta completo no le falta líneas.

2. ¿Cuál es la ecuación de recurrencia que representa la complejidad, en el tiempo, para el peor de los casos, en términos de $p=n+m$?
   T(p) = ...

4.3 En la vida real, las grandes empresas de tecnología, solicitan calcular la complejidad de los algoritmos que preguntan en sus entrevistas técnicas; por ejemplo, Google, Facebook y Riot Games. Estas entrevistas se realizan para acceder a una práctica laboral como lo hizo Santiago Zubieta, en 2015, y Juan M. Ciro, en 2019, para hacer sus prácticas en Google y Facebook, respectivamente. ¡Ambos eran estudiantes de la Universidad Eafit! Para prepararnos para esas entrevistas, para cada uno de los siguientes códigos, determina la ecuación que representa su complejidad para el peor caso.

Para el siguiente código, determina la ecuación que representa su complejidad para el peor caso. Considera que C es una constante.

```
1    void mistery(int n, int m){
2      int res = 0;
3      for(int i = 0; i < n; ++i){
4        for(int j = 1; j < m; ++j){
5          for(int k = 1; k < m; ++k){
6            res = res + 1;
7    } } } }
```

La complejidad de la función *mistery* es

a. $T(n,m) = C \times n \times m$
b. $T(n,m) = C \times n \times m^2$
c. $T(n,m) = C \times n \times m.\log m$
d. $T(n,m) = C \times n \times m^3$

4.4 Lika y Kefo están estudiando para el examen de matemáticas en su colegio. Hoy, ellos encontraron muchas secuencias de números interesantes, una de ellas los números Fibonacci. Para Lika –que ya conocía estos números– se le hace aburrido escribirlos todos; sin embargo, ellos encontraron otra secuencia de números llamados los números de Lucas. En el libro donde ellos encontraron esta secuencia de números, sólo hay 7 números

pertenecientes a la secuencia y al final de la hoja dice: “¿Podrás definir una relación de recurrencia que nos permita deducir el n-ésimo número de Lucas?” “Por supuesto” –dijeron ambos. Ahora, ellos quieren escribir un algoritmo que nos permita retornar el n-ésimo número de Lucas. ¿Puedes ayudarlos a escribir el algoritmo? La secuencia encontrada fue la siguiente: 2, 1, 3, 4, 7, 11, 18, ... Se sabe que el número 2 y el número 1 son el primer y segundo término, respectivamente, de la secuencia de los números de Lucas. Al código le faltan algunas líneas, para completar el ejercicio deberás completarlas.

```
1   int lucas(int n){
2   if(n == 0) return 2; C1 = 3
3   if(n == 1) return 1;
4   return lucas(n-1) + lucas(n-2);
5   }
```

4.4.1 La complejidad del algoritmo anterior, para el peor de los casos, en términos de *n* y donde c es una constante, es:

a. $T(n)=T(n-1)+c$, que es $O(n)$
b. $T(n)=4T(n/2)+c$, que es $O(n^2)$
c. $T(n)=T(n-1)+T(n-2)+c$, que es $O(2^n)$
d. $T(n)=c$, que es $O(1)$

4.5 En la vida real, los palíndromes se utilizan para desarrollar algoritmos de compresión de cadenas de ADN. Para este parcial, considera un algoritmo capaz de decir si una cadena de caracteres es un palíndrome o no. Un palíndrome es una cadena que se lee igual de izquierda a derecha que de derecha a izquierda. A continuación, algunos ejemplos:

- Para “amor a roma”, la respuesta es true
- Para “mamita”, la respuesta es false
- Para “cocoococ”, la respuesta es true

El algoritmo `isPal` soluciona el problema, pero le faltan unas líneas. Complétalas, por favor.

```
01      static boolean isPal(String s) {
02      if(s.length() == 0 || s.length() == 1)
03      return .......;
04      if(.........)
05      return isPal(s.substring(1, s.length()-1));
06      //else
07      return false;
08      }
```

En Java, el método `s.charAt(i)` permite saber qué caracter hay en la posición *i* de la cadena *s* y `s.substring(a,b)` retorna una subcadena de *s* entre los índices *a* y *b*−1.

Completen, por favor, las líneas faltantes:

1. Completa la línea 3 .....
   a. `true`
   b. `false`
   c. `s`
2. Completa la línea 4 .....
   a. `s.substring(0,s.length()-1).equals(s.substring(s.length()-1, 0))`
   b. `s.charAt(0) == (s.charAt(s.length()-1))`
   c. `true`