# Estructuras de Datos 2 - ST0247
# Examen Parcial 2 - Martes (033)

Nombre .....
Departamento de Informática y Sistemas
Universidad EAFIT

Mayo 9 de 2019

**En las preguntas de selección múltiple, una respuesta incorrecta tendrá una deducción de 0.2 puntos en la nota final. Si dejas la pregunta sin responder, la nota será de 0.0. Si no conoces la respuesta, no adivines.**

## 1 Programación Dinámica (30%)

Existe una *rana* sobre una roca en el punto 1 de un río con puntos $1, 2, \cdots, n-1, n$. La rana desea llegar al punto $n$ haciendo el menor número de saltos posibles. La rana puede realizar saltos de a lo sumo $d$ puestos siempre y cuando al punto que salte contenga una roca. Ayúdanos a escribir un programa que nos determine el **mínimo número de saltos** que tiene que realizar la rana para llegar al punto $n$.
Por ejemplo, si existen 8 puntos en el río y la configuración (`config`) de los puntos es $\{1, 0, 0, 1, 1, 0, 1, 1\}$ (donde el valor 1 indica la presencia de una roca), para un valor $d = 3$, la rana puede llegar al punto 8 en tres saltos de la siguiente forma: primero salta al punto 4, luego al punto 7 y, finalmente, al punto 8.

- La constante Integer.MAX_VALUE devuelve el valor $2^{31} - 1$.
- El método Arrays.fill(int[] *a*, int *value*) ubica en cada posición del arreglo *a*, el valor *value*.

```
1   int rana(int[]config, int d, int[]f){
2     int n = config.length;
3     asssert n == f.length;
4     Arrays.fill(f, Integer.MAX_VALUE);
5     //f[i]: minimo numero de saltos
6     //hasta llegar al punto i.
7     f[0] = 0;
8     for(int i=0; i<n; ++i){
9       for(int j=1; j<=d; ++j){
10        if(i-j>=0 && config[i]==1){
11          f[i]=.....;
12        }
13      }
14    }
15    return .....;
16  }
```

a (10%) Determina la complejidad asintótica, en el peor de los casos, del algoritmo anterior: _____

b (10%) Completa la línea 9 _____

c (10%) Completa la línea 13 _____

## 2 Voraces 30%

El coloramiento de grafos es un problema principal en la ingeniería de sistemas, de allí se pueden resolver muchas tareas, como lo son asignación de salones, semáforos, entre otras. Por eso, hoy vamos a desarrollar un algoritmo voraz para colorear un grafo con la mínima cantidad de colores. Ayúdanos a resolver el problema de encontrar la mínima cantidad de colores para pintar un grafo cualquiera. El algoritmo es el siguiente:

- Colorear el vértice 0 con el color 0.
- Por cada uno de los restantes $n$–1 vertices del grafo, tomar un vértice $u$. Colorear $u$ con el menor color que aún no ha sido utilizado para colorear sus vértices adyacentes. Si todos los colores previos ya han sido utilizados, colorear $u$ con un nuevo color.

```
1   int colorear(boolean[][] grafo){
2     int n = grafo.length;
3     int[] color = new int[n + 1];
4     for(int i = 0; i < n; ++i) color[i]
        = -1;
5     color[0] = 0;
6     boolean[] colordisponible = new
        boolean[n + 1];
7     for(int i = 0; i < n; ++i)
        colordisponible[i] = true;
8     for(int u = 1; u < n; ++u){
9       for(int i = 0; i < n; ++i)
10        if(grafo[u][i])
11          if(color[i] != -1)
12            colordisponible[color[i]] =
                false;
13    int aval;
14    for(aval = 0; aval < n; ++aval)
15      if(colordisponible[aval])
16        break;
17    color[u] = .....;
```

```
18      for(int i = 0; i < n; ++i)
            colordisponible[i] = true;
19      boolean[] usado = new boolean[n +
            1];
20      int res = 0;
21      for(int i = 0; i < n; ++i) if(!usado
            [color[i]]){
22        usado[..............] = true;
23        res++;
24      }
25      return res;
26    }
```

a) (10%) Completa la línea 22 .....

b) (10%) Completa la línea 17 .....

c) (10%) ¿Cuál es la complejidad asintótica, en el peor de los casos, del algoritmo anterior?

i) $O(n)$

ii) $O(n \times \log(n))$

iii) $O(n^2)$

iv) $O(n^3)$

# 3 Voraces 20%

Dados dos arreglos de enteros $a$, $b$ de tamaño $n$ cada uno, encuentre dos permutaciones $a'$, $b'$ tales que $\sum_{i=0}^{n-1} |a'_i - b'_i|$ sea tan pequeño como sea posible.

**Nota:** En Java, `Math.abs(n)` calcula el valor absoluto de $n$.

```
1   int solve(int[] a, int[] b){
2     int n = a.length;
3     //Paso 1: Procesamiento inicial
4     ..............;
5     ..............;
6     int res = -1;
7     for(int i = 0; i < n; ++i){
8       //Desicion voraz
9       res += Math.abs(a[i] - b[i]);
10    }
11    return res;
12  }
```

a) Completa las líneas 4, 5 ....., .....

b) Completa la línea 9 .....

# 4 Programación Dinámica (20%)

En áreas como la criptografía es usual la descomposición de números para encontrar una contraseña. Dado un entero positivo $S$, necesitamos descomponerlo en la suma de al menos 2 números enteros positivos, de tal forma que la multiplicación de tales enteros sea tan grande como sea posible. Vea los siguientes ejemplos:

- $S = 5$. $S = 2 + 3$. Respuesta: $2 \times 3 = 6$.
- $S = 7$. $S = 3 + 4$. Respuesta: $3 \times 4 = 12$.
- $S = 10$. $S = 3 + 3 + 4$. Respuesta: $3 \times 3 \times 4 = 36$.
- $S = 2$. $S = 1 + 1$. Respuesta: $1 \times 1 = 1$.

```
1   int dp(int S, int[] f) {
2     if (S <= 1) return ..............;
3     if (S == 2) return 1;
4     if (f[S] != -1) {
5       return f[S];
6     }
7     int res = 0;
8     for (int i = 1; i <= S; i++) {
9       res=Math.max(res, (S - i) * i);
10      res=Math.max(res, ...........);
11    }
12    f[S] = res;
13    return res;
14  }
15  int solve(int S){
16    int[] f = new int[S + 1];
17    Arrays.fill(f, -1);
18    return dp(S, f);
19  }
```

a Completa la línea 10 _____

b Completa la línea 2 _____