Estructuras de Datos y Algoritmos 1 - ST0245
Primer Parcial 001 - Jueves

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

Marzo 25 de 2021

1 Mujeres en Ingeniería (2% extra)

Varios estudios argumentan que muchas mujeres deciden NO estudiar ingeniería porque creen en los estereotipos sobre el tipo de personas que trabajan en el campo, y no se ven a sí mismas encajando en esos estereotipos. De esta manera, incorrectas percepciones pueden dar forma a las trayectorias profesionales. Una forma de desmentir dichos estereotipos es reconociendo mujeres exitosas en ingeniería de sistemas y matemática –tanto a nivel nacional como mundial. Para lograr esto, relaciona las siguientes mujeres con su contribución:

1. Adriana Noreña	a. Primera en pasar a Práctica en Facebook
2. Hilda Geiringer	b. Primera en Eafit en Generación de Código
3. Paola Vallejo	c. Primera en Colombia en Transparencia de Software
4. Elizabeth Suescún	d. Primera en ir a una Conferencia Internacional en Semestre II
5. Luisa Toro	e. Primera en enseñar Matemáticas Aplicadas en Alemania
6. Luisa Vásquez	f. Primera Vicepresidente de Google para Latinoamérica

1..., 2..., 3..., 4..., 5..., 6...

2 Recursión 30%

Para las grandes empresas son de suma importancia los juegos; como un ejemplo, IBM desarrolló el sistema *Deep Blue* que venció al campeón mundial de Ajedrez en 1996 y Google desarrolló el sistema *Alpha Go* que venció al campeón mundial de Go en 2016. Considera el siguiente juego. Dos amigos de primaria están jugando con un arreglito de n enteros $a = a_1, a_2, \cdots, a_n$. El juego consiste en remover un elemento del principio o del fin del arreglito y sumarlo al conjunto de enteros que cada uno tiene en ese momento. El juego es alternado e –inicialmente– empieza a tomar el primer entero el amigo 1; después es el turno del amigo 2, y así sucesivamente. El juego finaliza cuando no quedan más enteros en el arreglito a. Inicialmente, ninguno de los dos amigos tiene un entero en su poder. Al final del juego, cada uno de los dos amigos tiene una suma total de X_1 y X_2, respectivamente. El objetivo del amigo 1 es hacer que $X_1 - X_2$ sea lo más grande posible; mientras que, para el amigo 2, es hacer que $X_1 - X_2$ sea lo más pequeño posible. Encuentra el valor de $X_1 - X_2$, al final del juego, si ambos amigos juegan de manera óptima.

Como un ejemplo, dada la entrada $a = [10, 80, 90, 30]$, la respuesta es 10 porque la secuencia sería la siguiente: El amigo 1 toma el 30, luego el amigo 2 toma el 10, luego el amigo 1 toma el 80 y, por último, el amigo 2 toma el 90. De esa forma, $X_1 = 30 + 80 = 110$ y $X_2 = 10 + 90 = 100$, luego $X_1 - X_2 = 10$.

Si trabajas en Java, revisa este código:

```
1   int solve(int[] a, int i, int j){
2     if(i == j)
3       return a[i];
4     return ..........................;
5   }
6   int solve(int[] a){
7     return solve(a, 0, a.length -1);
8   }
```

Si trabajas en Python, revisa este código:

```
1   def solveAUX(a, i, j):
2     if i == j:
3       return a[i]
4     return ..........................
5
6   def solve(a):
7     return solve(a, 0, len(a) - 1)
```

(A) (10%) Completa la línea 4

_____

(B) (10%) ¿Qué retorna al algoritmo `solve(a)` para $a = [10, 20, 30, 50]$

_____

(C) (10%) ¿Cuál es la ecuación de recurrencia que representa la complejidad para el peor de los casos?

T(n) = _____

Pista: En Java, el máximo de dos números se calcula con `Math.max(a,b)` y en Python con `max(a,b)`.

3 Notación O 20%

El problema de la subsecuencia común más larga es de importancia en biología informática; por ejemplo, para encontrar la explicación de los cánceres que son causados por desórdenes genéticos. Además, este problema es común en entrevistas de *Amazon*, según el portal *Geeks for Geeks*. El problema es el siguiente. Dadas dos secuencias –una llamada X de longitud m y otra llamada Y de longitud n–, el siguiente algoritmo encuentra la longitud de la subsecuencia más larga presente en ambas. Una subsecuencia es una secuencia que aparece en el mismo orden relativo, pero no necesariamente contiguo. Por ejemplo, "abc", "abg", "bdf", "aeg" y "acefg" son subsecuencias de "abcdefg". Como otro ejemplo, la cadena "ADH" es la subsecuencia común más larga (en inglés, *LCS*) de "ABCDGH" y "AEDFHR", y su longitud es 3.

Si trabajas en Java, considera este código:

```
1  /* Retorna la longitud de la LCS de X[0..m-1] y Y[0..n-1] */
2  int lcs( char[] X, char[] Y, int m, int n ) {
3      int L[][] = new int[m+1][n+1];
4
5      /* L[i][j] es la longitud de la LCS de X[0..i-1] y Y[0..j-1] */
6      for (int i=0; i<=m; i++) {
7        for (int j=0; j<=n; j++) {
8          if (i == 0 || j == 0)
9              L[i][j] = 0;
10         else if (X[i-1] == Y[j-1])
11             L[i][j] = L[i-1][j-1] + 1;
12         else
13             L[i][j] = max(L[i-1][j], L[i][j-1]);
14       }
15     }
16   return L[m][n];
17 }
```

Si trabajas en Python, considera este código:

```
1  # Retorna la longitud de la LCS de X[0..m-1] y Y[0..n-1]
2  def lcs(X , Y, m, n):
3      L = [[None]*(n+1) for i in range(m+1)]
4
5      #L[i][j] es la longitud de LCS de X[0..i-1] y Y[0..j-1]
6      for i in range(m+1):
7          for j in range(n+1):
8              if i == 0 or j == 0 :
9                  L[i][j] = 0
10             elif X[i-1] == Y[j-1]:
11                 L[i][j] = L[i-1][j-1]+1
12             else:
13                 L[i][j] = max(L[i-1][j]  , L[i][j-1])
14
15     return L[m][n]
```

(A) (10%) ¿Cuál es la complejidad asintótica, en **tiempo**, en el peor de los casos, en términos de n y de m? O(.....)

(B) (10%) ¿Cuál es la complejidad asintótica, en **memoria**, en el peor de los casos, en términos de n y de m? O(.....)

Pista: La complejidad en memoria quiere decir cuántos números nuevos crea el algoritmo, fuera de los que ya existían.

4 Listas enlazadas 20%

En la vida real, determinar si una cadena con paréntesis, llaves y corchetes es válida es de interés para grandes empresas como *Microsoft* (creador del lenguaje C#) u *Oracle* (dueño del lenguaje Java). Además, es un ejercicio muy común en entrevistas para estas empresas según el portal *Geeks for Geeks*. Veamos el problema. Dada una cadena que contiene únicamente los caractéres "(", "{", "[", ")", "}", "]", determina si ésta es válida. La cadena es válida si para cada caractér que abre, siempre hay uno que cierra en el orden correcto. Como un ejemplo "([ ])" es válida, pero "[(])" no es válida porque el paréntesis ")" debe cerrar antes que el corchete "]".

Si trabajas en Java, considera este código:

```
1  boolean solve(String exp) {
2    LinkedList<Character> s = new LinkedList();
3    for(int i=0;i<exp.length();++i){
4      char ei = exp.charAt(i);
5      if(ei == '(' | ei == '{' | ei == '[')
6        ..........................;
7      if(ei == ')' | ei == '}' | ei == ']'){
8        if(s.size() == 0 | !valid(s.get(s.size()-1), ei))
9          return false;
10       else{
11         ..........................;
12   } } }
13   return s.size() == 0;
14 }
15 boolean valid(char o, char c) {
16   return (o == '(' && c == ')') |
17          (o == '[' && c == ']') |
18          (o == '{' && c == '}');
19 }
```

En Java, la librería `LinkedList` es una lista doblemente enlazada. En esa librería, `add(e,i)` agrega e en la posición i, `add(e)` agrega a e al final, `remove(i)` elimina el elemento en la posición i y `get(i)` retorna el elemento en la posición i.

Si trabajas en Python, considera este código:

```
1  def solve(exp):
2    s = deque()
3    for i in range(0, len(exp)):
4      ei = exp[i]
5      if ei == "(" or ei == "{" or ei == "[":
6        ........................
7      if ei == ")" or ei == "}" or ei == "]":
8        if len(s) == 0 or not valid(s[len(s)-1], ei):
9          return False
10       else:
11         ........................
12   return len(s) == 0
13
14 def valid(o, c):
15   return (o == "(" and c == ")") or \
16          (o == "[" and c == "]") or \
17          (o == "{" and c == "}")
```

En Python, la librería `deque` es una lista doblemente enlazada. En deque, `append()` agrega al final, `appendLeft()` agrega al inicio, `pop()` elimina el último elemento y `popleft()` elimina el primer elemento.

(A) (10%) Completa la línea 6 .....

Y Completa la línea 11 .....

(B) (10%) Cuál es la complejidad asintótica, en el peor de los casos, del algoritmo anterior.

O(_____) , donde e es el número de caracteres de `exp`.

5 Vectores dinámicos 10%

En seguridad informática, las permutaciones de una cadena se utilizan para realizar un ataque por fuerza bruta a un sistema informático. Consideremos el siguiente problema de permutaciones. Dada una cadena de letras y números, es necesario encontrar todas las posibles permutaciones de ésta cadena al usar solo letras minúsculas y mayúsculas. Por ejemplo,

- "as22b". **Respuesta:** As22b, aS22b, as22b, AS22b, As22B, aS22B, as22BB, AS22B.
- "123". **Respuesta:** 123
- "1B4". **Respuesta:** 1b4, 1b4

El siguiente código resuelve el problema, pero es necesario completar algunas lineas.

Si trabajas en Java, considera este código:

```
1  ArrayList<String> solve(String s) {
2    ArrayList<String> sol = new ArrayList();
3    tmp("", s, 0, s.length() - 1, sol);
4    return sol;
5  }
6  void tmp(String l,String s,int i,int j, ArrayList<String> sol)
7  {
8    if (i > j) {
9      ..........................
10     return;
11   }
12   String c = "" + s.charAt(i);
13   c = c.toLowerCase(); // Convierte c a minusculas
14   tmp(l + c, s, i + 1, j, sol);
15   String k = "" + s.charAt(i);
16   k = k.toUpperCase(); // Convierte k a mayusculas
17   if (!c.equals(k)) {
18     ..........................
19   }
20 }
```

Si trabajas en Python, considera este código:

```
1  def solve(s):
2    sol = [];
3    tmp("", s, 0, len(s) - 1, sol)
4    return sol
5
6  def tmp( l, s, i, j, sol):
7
8    if i > j:
9      ..........................
10     return
11
12   c = "" + s[i]
13   c = c.lower() # Convierte c a minusculas
14   tmp(l + c, s, i + 1, j, sol)
15   k = "" + s[i]
16   k = k.upper() # Convierte k a mayusculas
17   if not c == k:
18     ..........................
```

a (10%) Completa la línea 9 .....

b (10%) Completa la línea 17 .....

6 Complejidad 20%

En la vida real las sub-cadenas son súper importantes para empresas como *Google* y *Microsoft* para mejorar sus motores de búsqueda *Google* y *Bing*, respectivamente. Dado una cadena de caracteres en minúsculas s de tamaño n y un entero k, es necesario contar el numero de sub-cadenas de s de tamaño $k \leq n$, tal que cada cada una de estas sub-cadenas no tiene caracteres repetidos. **Ejemplo:** *"andamasqueyo"*, $k = 5$. **Respuesta**: 4 (*masqu, asque, squey, queyo*). **Ejemplo:** *"casa"*, $k = 5$. **Respuesta**: 0.
Lo que tienes que hacer es construir un algoritmo para resolver este problema. Ya algunas líneas se han implementado, entonces sólo debes implementar el resto. Para lograrlo se crea un arreglo `count` que contiene las ocurrencias de cada letra, enumeradas de la 0 a la 26. Cuando se hace la operación $c - 97$ –donde c es un caracter– lo que se hace es asignar a la letra a la posición 0, a la letra b la posición 1, y así sucesivamente. Esto sucede porque los números del 97 al 122 se usan para representar las letras de la a a la z. Para lograr eso en Python, se debe convertir el caracter con `ord()`.

Si trabajas en Java, considera este código:

```
1  int solve(String s, int k){
2    if(s.length() < k) return 0;
3    int[] count = new int[26];
4    for(int i = 0; i < k; ++i){
5      count[s.charAt(i) - 97]++;
6    }
7    int ans = check(count) ? 1: 0;
8    for(int i = k; i < s.length(); ++i){
9      count[s.charAt(i - k) - 97] --;
10     count[s.charAt(i) - 97] = count[s.charAt(i) - 97] + 1;
11     if(check(count)){
12       ans++;
13     }
14   }
15   return ans;
16 }
17 boolean check(int[] count){
18   for(int i = 0; i < 26; ++i){
19     if(count[i] > 1) return false;
20   }
21   return true;
22 }
```

Si trabajas en Python, considera este código:

```
1  def solve(s, k):
2    if(len(s) < k):
3      return 0
4    count = np.zeros(26) # arreglo de 26 elementos
5    for i in range (0,k):
6      count[ord(s[i]) - 97] += 1
7    ans = 1 if check(count) else 0
8    for i in range (k, len(s)):
9      count[ord(s[i - k]) - 97] -= 1
10     count[ord(s[i]) - 97] = count[ord(s[i]) - 97] + 1
11     if check(count):
12       ans += 1
13   return ans
14
15 def check(count):
16   for i in range(0,26):
17     if count[i] > 1:
18      return False
19   return True
```

(A) (10%) ¿Cuál es la complejidad asintótica **en tiempo** para el peor de los casos?

(B) (10%) ¿Cuál es la complejidad asintótica **en memoria** para el peor de los casos?

7 Complejidad (10% EXTRA)

Te entregan un entero $Q \geq 1$. Luego, te entregan Q consultas, cada una de las cuales se compone de una tripleta (a, n, k); donde a es un arreglo de enteros, n es el tamaño de a y k es un entero $1 \leq k \leq n$. La idea es responder en el menor tiempo posible las Q consultas: Al ordenar los elementos de a de forma creciente, ¿cuál es el elemento que queda en la posición k?

Si trabajas en Java, considera este código:
Se propone un algoritmo usando el algoritmo de ordenamiento *Merge Sort*. El siguiente es el algoritmo que se está usando:

```
1    int[] solve(int Q, int[][] aq, int[] k){
2      int[] result = new int[Q];
3      for(int i = 0; i < Q; ++i){
4        Arrays.sort(aq[i]); // Este usa Merge Sort
5        result[i] = aq[i][k[i]];
6      }
7      return result;
8    }
```

Se cree que el algoritmo anterior se puede mejorar. Entonces, se propone el siguiente algoritmo.

```
1   int[] solve(int Q, int[][] aq, int[] k){
2      int[] result = new int[Q];
3      for(int i = 0; i < Q; ++i){
4        insertionSort(aq[i]); //Este es insertion sort
5        result[i] = aq[i][k[i]];
6      }
7      return result;
8    }
9
10  void insertionSort(int arr[]) {
11          int n = arr.length;
12          for (int i = 1; i < n; ++i) {
13              int key = arr[i];
14              int j = i - 1;
15              while (j >= 0 && arr[j] > key) {
16                  arr[j + 1] = arr[j];
17                  j = j - 1;
18              }
19              arr[j + 1] = key;
20          }
21    }
```

Si trabajas en Python, considera este código:
Se propone un algoritmo usando el algoritmo de ordenamiento *Merge Sort*. El siguiente es el algoritmo que se está usando:

```
1    def solve( Q, aq,  k):
2      result = np.zeros(Q)
3      for i in range(0, Q):
4        aq[i].sort() # Este usa Merge Sort
5        result[i] = aq[i][k[i]]
6      return result
```

Se cree que el algoritmo anterior se puede mejorar. Entonces, se propone el siguiente algoritmo.

```
1   def solve(Q, aq, k):
2     result = np.zeros(Q)
3     for i in range(0, Q):
4       insertionSort(aq[i]) # Este es Insertion Sort
5       result[i] = aq[i][k[i]]
6     return result
7
8   def insertionSort(arr):
9     for i in range(1, len(arr)):
10        key = arr[i]
11        j = i-1
12        while j >= 0 and key < arr[j]:
13                arr[j + 1] = arr[j]
14                j -= 1
15        arr[j + 1] = key
```

(A) (10%) En el mejor caso, ¿cuál de los dos algoritmos usaría y por qué? Justifica tu respuesta usando la notación **O**.

_____

_____

_____

Y En el peor caso, ¿cuál de los dos algoritmos usaría y por qué? Justifica tu respuesta usando la notación **O**.

_____

_____

_____