Estructuras de Datos y Algoritmos 1 - ST0245
Primer Parcial 002 - Martes

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

Marzo 23 de 2021

1 Mujeres en Ingeniería (2% extra)

(2 %) 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. Ada Lovelace	a. Creadora de los procesadores ARM
2. Rózsa Péter	b. Youtuber de algoritmos más vista
3. Gayle McDowell	c. Primera en ganar un premio Turing
4. Dorothy Vaughan	d. Primera persona en programar
5. Sophie Wilson	e. Teoría de las funciones recursivas
6. Barbara Liskov	f. Primera jefa de un equipo de computación en NASA

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

2 Recursión 30%

En la vida real, los algoritmos que trabajan sobre subsecuencias tienen una doble aplicación. Primero, se usan para analizar ADNs en bioinformática; por ejemplo, para establecer la presencia de un desorden genético hereditario. Segundo, se utilizan para analizar información de páginas web o redes sociales, como es el caso de Google, Facebook, Microsoft y Amazon. Te entregan dos cadenas de texto: s y t. Encuentra el tamaño de la cadena más grande que es subsecuencia tanto de s como t. En una subsecuencia se respeta el orden de aparición de los caracteres, pero estos no aparecen –necesariamente– de forma contigua. Como un ejemplo, para las cadenas "aa" y "xayaz", la respuesta es 2 porque la subsecuencia más larga es "aa".

Si trabajas en Java, considera el siguiente código:

```
1   int solve(String s, String t){
2     return solve(s, t, s.length(), t.length());
3   }
4   int solve(String s, String t, int i, int j){
5     if(i == 0 || j == 0)
6       return 0;
7     if(s.charAt(i - 1) == t.charAt(j - 1))
8       return ..........................;
9     //else
10    return ..........................;
11  }
```

Si trabajas en Python, considera el siguiente código:

```
1   def mainSolve( s,  t):
2     return solve(s, t, len(s), len(t))
3
4   def solve(s, t, i, j):
5     if i == 0 or j == 0:
6        return 0;
7     if s[i - 1] == t[j - 1]:
8       return ..........................
9     else:
10      return ..........................
11  }
```

(A) (10%) Completa la línea 8

_____

(B) (10%) Completa la línea 10

_____

(C) (10%) ¿Cuál es la complejidad asintótica para el peor de los casos?

T(n) = _____, donde n es la suma de la longitud de s y t.

3 Notación O 20%

En la vida real, hacemos filas de espera (*Waiting lines*) para entrar a un parqueadero, para ser atendido en un banco o para comprar un producto en un supermercado. Implementar filas de espera, en un programa informático, es importante para simular, por ejemplo, el impacto que tendría –sobre los tiempos de espera– el contratar un nuevo cajero. Por eso, estas implementaciones son de suma importancia para empresas como Bancolombia. La siguiente clase implementa una fila de espera para un máximo de n elementos. Tan sólo hace falta calcular la complejidad asintótica, para el peor de los casos, de las operaciones entrar a la fila de esperar (*enqueue*) y salir de la fila de espera (*dequeue*).

Si trabajas en Java, considera el siguiente código:

```
1  class WaitingLine {
2    int e[];
3    int index;
4    boolean started;
5
6    WaitingLine(int n) {
7      e = new int[n];
8      index = 0;
9      started = false;
10   }
11
12   public boolean isEmpty() {
13     return (index == 0 && !started) | index <= 0;
14   }
15
16   void enqueue(int x){
17     if (index == e.length) {
18       throw new RuntimeException("Full");
19     }
20     started = true;
21     e[index] = x;
22     index++;
23   }
24
25   int dequeue(){
26     if (isEmpty())
27       throw new RuntimeException("Empty");
28     int x = e[0];
29     for (int i = 0; i < index - 1; i++) {
30       e[i] = e[i + 1];
31     }
32     index--;
33     return x;
34   }
35 }
```

(A) (10%) Calcula la complejidad asintótica, para el peor de los casos, de `enqueue(x)`

O(_____)

(B) (10%) Calcula la complejidad asintótica, para el peor de los casos, de `dequeue(x)`

O(_____)

Si trabajas en Python, considera el siguiente código:

```
1  import numpy as np
2  class WaitingLine:
3
4    def __init__(self, n):
5      self.e = np.zeros(n)
6      self.index = 0;
7      self.started = False;
8
9    def isEmpty(self):
10     return (self.index == 0 and not self.started) or self.index <= 0
11
12   def enqueue(self,x):
13     if self.index == self.e.size:
14       raise Exception("Full")
15     self.started = True
16     self.e[self.index] = x
17     self.index = self.index + 1
18
19   def dequeue(self):
20     if self.isEmpty():
21       raise Exception("Empty")
22     x = self.e[0]
23     for i in range (0, self.index - 1):
24       self.e[i] = self.e[i + 1]
25     self.index = self.index - 1
26     return x
```

(A) (10%) Calcula la complejidad asintótica, para el peor de los casos, de `enqueue(x)`

O(_____)

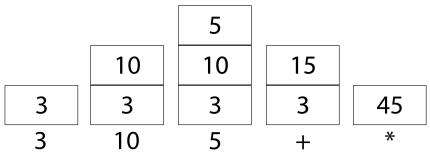
(B) (10%) Calcula la complejidad asintótica, para el peor de los casos, de `dequeue(x)`

O(_____)

4 Listas enlazadas 20%

La *Notación polaca inversa* es una notación matemática en la cual los operadores siguen sus operandos. En la vida real, la notación polaca inversa fue ampliamente utilizada, en los años 70s y 80s, por las calculadoras científicas Hewlett-Packaar. Por ejemplo, la expresión en notación polaca $5\ 31\ 4 + \times$, equivale a la expresión $(31 + 4) \times 5$. Dado un arreglo de operadores y operandos (*tokens*) que representan una expresión en notación polaca, considerando sólo los operadores "(+, *, -, /)", determina el resultado de la expresión. Utilizando una lista enlazada, es posible operar una expresión en notación, así:

Equation: 3 10 5 + *



Ya hemos adelantado algo de trabajo, pero falta completar algunas líneas, por favor.

Si trabajas en Java, considera el siguiente código:

```
1  int solve(String[] tokens){
2    String operators = "+-*/";
3    LinkedList<String> aList = new LinkedList<String>();
4    for(String t : tokens){
5      if(!operators.contains(t)){
6        ..............;
7      }else{
8        int a = ..................;
9        aList.remove(0);
10       int b = ..................;
11       aList.remove(0);
12       int index = operators.indexOf(t);
13       if(index == 0)
14        aList.add(0,String.valueOf(a+b));
15       else if(index == 1)
16        aList.add(0,String.valueOf(b-a));
17       else if(index == 2)
18        aList.add(0,String.valueOf(a*b));
19       else // index == 3
20        aList.add(0,String.valueOf(b/a));
21     }
22   }
23   String last = aList.remove(0);
24   return ..............;
25 }
```

(A) (10%) Completa la línea 6

_____

(B) (10%) Completa las líneas 8 y 10

_____

_____

Y completa la línea 24

_____

En Java, el método `String.valueOf(a)` convierte el entero a en una cadena de caracteres y el método `Integer.valueOf(b)` convierte la cadena b en un entero.

Si trabajas en Python, considera el siguiente código:

```
1  def solve(tokens):
2    operators = "+-*/"
3    aList = []
4    for t in tokens:
5      if not t in operators:
6        ..................
7      else:
8        a = .........
9        aList = aList[1:]
10       b = ..........
11       aList = aList[1:]
12       index = operators.index(t)
13       if index == 0:
14        aList = [str(a+b)] + aList
15       elif index == 1:
16        aList = [str(a-b)] + aList
17       elif index == 2:
18        aList = [str(a*b)] + aList
19       else:
20        aList = [str(b/a)] + aList
21   last = aList[0]
22   return ...............
```

(A) (10%) Completa la línea 6

_____

(B) (10%) Completa las líneas 8 y 10

_____

_____

Y completa la línea 22

_____

En Python, la función `str(a)` convierte el entero a en una cadena de caracteres y la función `int(b)` convierte la cadena b en un entero.

5 Vectores dinámicos 10%

Empresas como Microsoft se benefician muchísimo de los algoritmos basados en subarreglos, por ejemplo, para implementar el algoritmo de *buscar* o *reemplazar* en sus programas ofimáticos. Pensemos un problema con subarreglos. Dado un arreglo `arr` de `n` enteros, encuentra, por favor, el subarreglo contiguo con la máxima suma posible. Este problema, coincidencialmente, es muy común en entrevistas de Microsoft, Oracle y Amazon según el portal *Geeks for Geeks*. Como un ejemplo, si tenemos el arreglo $arr = \{1, 2, 3, -2, 5\}$, la respuesta es 9 porque todo el arreglo es el subarreglo con la máxima suma. Como otro ejemplo, si tenemos el arreglo $arr = \{-1, -2, -3, -4\}$, la respuesta es -1 porque el subarreglo con máxima suma es $\{-1\}$. Hemos avanzado un poco, pero falta una línea. Complétala, por favor. Gracias.

Si trabajas en Java, considera el siguiente código:

```
1  int maxSubArraySum(ArrayList<Integer> a)
2  {
3          int size = a.size();
4          int max_so_far = Integer.MIN_VALUE, max_ending_here = 0;
5
6          for (int i = 0; i < size; i++)
7          {
8              max_ending_here = max_ending_here + a.get(i);
9              if (.....................)
10                 max_so_far = max_ending_here;
11             if (max_ending_here < 0)
12                 max_ending_here = 0;
13         }
14         return max_so_far;
15     }
16 }
```

En Java, `Integer.MIN_VALUE` representa el entero más pequeño.

Si trabajas en Python, considera el siguiente código:

```
1  import math
2  def maxSubArraySum(a,size):
3
4      max_so_far = -1*math.inf - 1
5      max_ending_here = 0
6
7      for i in range(0, size):
8          max_ending_here = max_ending_here + a[i]
9          if (.....................):
10             max_so_far = max_ending_here
11
12         if max_ending_here < 0:
13             max_ending_here = 0
14     return max_so_far
```

En Python, `math.inf` representa el infinito (∞) y `-1*math.inf` el menos infinito ($-\infty$).

(A) (10%) Completa la línea 9

---

6 Complejidad 20%

En empresas como Riot Games, Electronic Arts y Microsoft, para entrar a trabajar como desarrollador de videojuegos, es imperativo conocer sobre complejidad de algoritmos. Considera el siguiente algoritmo:

Si trabajas en Java, considera el siguiente código:

```
1  void doSomething(int n){
2    int sum = 0;
3    for(int i = 1; i <= n; ++i){
4      for(int j = i + 1; j <= n; j = j * 2){
5        sum ++;
6      }
7    }
8  }
```

Si trabajas en Python, considera el siguiente código:

```
1  def doSomething(n):
2    sum = 0
3    for i in range(1, n+1):
4      j = i+1
5      while j <= n:
6        sum = sum + 1;
7        j = j * 2
```

a (10%) Calcula la complejidad asintótica, para el peor de los casos, es decir, $O(T(n))$: O(.....)

b (10%) Calcula la complejidad asintótica, para el peor de los casos, si cambiamos `j = j * 2` por `j = j + 2`: O(.....)

7 Complejidad (10% EXTRA)

El desarrollo de juegos es de suma importancia para empresas como *Riot Games* y *Microsoft*. Consideremos el siguiente juego. Hay n muros donde María Isabel va a empezar a saltar. María se encuentra en el muro 1 y quiere llegar al muro n. Sí María se encuentra en el muro i, solo puede saltar a los muros $i+1, i+2, \cdots, i+k$. Al realizar el salto –desde el muro i hasta el muro j–, ella gasta $\mid a_i - a_j \mid$ unidades de energía. María solo dispone de M unidades de energía. El siguiente algoritmo determina si existe alguna manera de que ella pueda llegar al muro n.

Como un ejemplo, si $a = \{1, 2, 2, 1, 3, 2, 1\}$, $k = 2$ y $M = 4$, la respuesta es verdadero, mientras que si $a = \{1, 2, 2, 1, 3, 2, 1\}$, $k = 2$ y $M = 1$, la respuesta es falso.

Si trabajas en Java, considera el siguiente código:

```
1   int solve(int[] a, int k, int n, int i){
2     if(i == n - 1){
3       return 0;
4     }
5     int res = Integer.MAX_VALUE;
6     for (int j = 1; j <= k; ++j){
7       if(j + i >= n) break;
8       res = Math.min(res, solve(a, k, n, j + i) + Math.abs(a[j + i] - a[i]));
9     }
10    return res;
11  }
12  boolean mariaCanGoToN(int[] a, int k, int M){
13    return solve(a, k, a.length, 0) <= M;
14  }
```

Si trabajas en Python, considera el siguiente código:

```
1   def solve(int[] a, int k, int n, int i){
2     if(i == n - 1){
3       return 0;
4     }
5     int res = Integer.MAX_VALUE;
6     for (int j = 1; j <= k; ++j){
7       if(j + i >= n) break;
8       res = Math.min(res, solve(a, k, n, j + i) + Math.abs(a[j + i] - a[i]));
9     }
10    return res;
11  }
12  boolean mariaCanGoToN(int[] a, int k, int M){
13    return solve(a, k, a.length, 0) <= M;
14  }
```

a (10 %) Calcula la complejidad asintótica, para el peor de los casos, del algoritmo `mariaCAnGoToN`, donde n es el número de elementos del arreglo y k es el máximo que ella puede saltar.

(A) $T(n,k) = k * T(n-k) + c$, que es $O(k^n)$

(B) $T(n,k) = T(n-k) + c$, que es $O(n)$

(C) $T(n,k) = 2 * T(n-1) + c$, que es $O(2^n)$

(D) $T(n,k) = T(n/k) + c$, que es $O(log_k\ n)$

b ** Para lo más curiosos: ¿Es posible mejorar la complejidad en tiempo de este algoritmo utilizando programación dinámica? Justifica tu respuesta mencionando, por ejemplo, el tamaño de la tabla y la nueva complejidad en tiempo.

.....