Estructuras de Datos y Algoritmos 1 - ST0245
Segundo Parcial (001, 002 y 003)

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

Noviembre 10 de 2020

1. Árboles binarios 30 %

Según Andrés Mejía –egresado de Eafit; reconocido por haber trabajo en Google y Facebook; y, actualmente, ingeniero de software en Riot Games–, el principal reto que uno afronta al entrar a trabajar a una gran compañía –como Riot Games– es cómo entender el código que han hecho otras personas, entender su complejidad asintótica para el peor de los casos y poder realizar mejoras. El siguiente algoritmo es de vital importancia para empresas –como Oracle y Microsoft– porque se utiliza dentro de los compiladores de lenguajes orientados a objetos como Java y C#. Por si fuera poco, este ejercicio es común en entrevistas para grandes empresas, según el portal *LeetCode*. Imagina que llegas nuevo a una de estas empresas, y debes entender y modificar este código. Lo único que sabemos es que `left_mis` es el resultado de `mistery` del árbol izquierdo y `right_mis` es el resultado de `mistery` del árbol derecho.

Si trabajas en Java, revisa este código:

```
1  class Node {
2   int data;
3   Node left, right;
4   public Node(int item) {
5    data = item;
6    left = right = null;
7  }}
8
9  public class BinaryTree {
10  private Node root;
11  Node mistery(int n1, int n2) {
12   return mistery(root, n1, n2);
13  }
14
15  Node mistery(Node node, int n1, int n2) {
16   if (node == null) return null;
17
18   if (node.data == n1 || node.data == n2)
19    return node;
20
21   Node left_mis = mistery(node.left, n1, n2);
22   Node right_mis =mistery(node.right, n1, n2);
23
24   if (left_mis!=null && right_mis!=null)
25    return node;
26
27   return (left_mis != null) ? left_mis :
         right_mis;
28 }}
```

Si trabajas en Python, revisa este código:

```
1  class Node:
2   def __init__(self, key):
3    self.key = key
4    self.left = None
5    self.right = None
6
7  def mistery(root, n1, n2):
8   if root is None:
9    return None
10
11  if root.key == n1 or root.key == n2:
12   return root
13
14  left_mis = mistery(root.left, n1, n2)
15  right_mis = mistery(root.right, n1, n2)
16
17  if left_mis and right_mis:
18   return root
19
20  return left_mis if left_mis is not None else
        right_mis
```

A (10 %) ¿Qué retorna la función `mistery`? Retorna el nodo.....

B (10 %) ¿Cuál es la complejidad **asintótica**, en el peor de los casos, del algoritmo anterior o la ecuación de recurrencia para el peor caso? .....

C (10 %) En una entrevista de Oracle, nos dicen que es un *árbol binario de búsqueda (BST)*. ¿Cómo se puede hacer más eficiente el algoritmo `mistery` sabiendo que el árbol es un BST? Se puede.....

2. Pilas 20 %

El Internet de las cosas (en inglés, IoT) se refiere a una red de objetos físicos que están conectados con el propósito de intercambiar datos con otros objetos a través de Internet. La definición de la IoT ha evolucionado debido a la convergencia de análisis de datos en tiempo real, aprendizaje automático y sensores. IoT incluye objetos (como lámparas y cámaras de seguridad) que pueden controlarse mediante dispositivos como teléfonos y altavoces inteligentes.

Una de las empresas más importantes en el desarrollo de estas tecnologías es Cisco. Cisco tiene un problema y es que, en la programación de muchos de sus microcontroladores, no se permite el uso de recursión. Cisco necesita ordenar un arreglo de números enteros usando el algoritmo de Merge

Sort. Ayuda, por favor, a Cisco a escribir el algoritmo de Merge Sort utilizando dos pilas. Sólo le faltan algunas líneas a su algoritmo. ¡Ayúdanos a completarlas!

Si no lo recuerdas, el algoritmo de Merge Sort, escrito de forma recursiva, es el siguiente:

```
1  public void mergeSort(int [] a) {
2    mergeSort(a, 0, a.length);
3  }
4  void mergeSort(int a[], int beg, int end) {
5   int mid;
6   if(beg<end) {
7           mid = (beg+end)/2;
8           mergeSort(a,beg,mid);
9           mergeSort(a,mid+1,end);
10          merge(a,beg,mid,end);
11 }}
```

El algoritmo `merge(a, beg, mid, end)` recibe un arreglo ordenado entre la posición `beg` y `mid` y otro arreglo ordenado entre la posición `mid+1` y `end`, y lo organiza de tal forma que quede un arreglo ordenado entre la posición `beg` y `end`. Escrito de otra forma, el algoritmo `merge(arr1, arr2)` recibe dos arreglos ordenados y retorna un nuevo arreglo ordenado con los elementos de ambos arreglos.

Si trabajas en Java, revisa este código:

```
1  import java.util.Stack;
2  static int[] mergeSortStacks(int[] A) {
3   Stack<int[]> stack = new Stack<int[]>();
4   Stack<int[]> stack2 = new Stack<int[]>();
5   for (int i = 0; i < A.length; i++) {
6    stack.push(new int[]{A[i]});
7   }
8   ................
9    while (stack.size()>1) {
10    int[] r = stack.pop();
11    int[] l = stack.pop();
12    int[] merged = merge(l, r);
13    stack2.push(merged);
14   }
15   while (stack2.size()>1) {
16    int[] r = stack2.pop();
17    int[] l = stack2.pop();
18    int[] merged = merge(l, r);
19    stack.push(merged);
20   }
21  }
22  return ......... ? ........... :
        ............;
23 }
```

Si trabajas en Python, revisa este código:

```
1  from collections import deque
2  def mergeSortStacks(A):
3   stack = deque()
4   stack2 = deque()
5   for element in A:
6    stack.append([element])
7
8   ................
9    while stack.size()>1:
10    r = stack.pop()
11    l = stack.pop()
12    merged = merge(l, r)
13    stack2.append(merged)
14
15   while stack2.size()>1:
16    r = stack2.pop()
17    l = stack2.pop()
18    merged = merge(l, r)
19    stack.push(merged)
20
21
22  return .......... if ......... else
        ..............
```

A (10 %) Completa la línea 8 .....

B (10 %) Completa la línea 22 .....

3. Colas 20 %

Ahora ayuda, por favor, a Cisco a escribir el algoritmo de Merge Sort –sin recursión– utilizando una cola. Sólo le faltan algunas líneas a su algoritmo. El algoritmo `merge(arr1, arr2)` está explicado en el punto anterior.

Si trabajas en Java, revisa este código:

```
1  import java.util.LinkedList; import java.util.
       Queue;
2  static int[] mergeSortQueue(int[] A) {
3   Queue<int[]> queue = new LinkedList<int[]>();
4   for (int i = 0; i < A.length; i++) {
5    queue.add(new int[]{A[i]});
6   }
7   while (queue.size()>1) {
8    int[] r = queue.poll();
9    int[] l = queue.poll();
10   ..............
11   queue.add(merged);
12  }
13  return ...........
```

Si trabajas en Python, revisa este código:

```
1  from collections import deque
2  def mergeSortQueue(A):
3   queue = deque()
4   for element in A:
5    queue.appendLeft([element]);
6
7   while queue.size()>1:
8    r = queue.pop()
9    l = queue.pop()
10   ...........
11   queue.appendLeft(merged)
12
13  return ..........
```

A (10 %) Completa la línea 10 .....

B (10 %) Completa la línea 13 .....

4. Tablas de Hash 20 %

El siguiente problema es muy común en entrevistas de *Goldman Sachs* según el portal *Geeks for Geeks*. Dados dos arreglos *arr1* y *arr2*, de igual tamaño, la tarea es encontrar si los dos arreglos son iguales o no. Se dice que dos arreglos son iguales si ambos contienen el mismo conjunto de elementos, aunque el orden (o permutación) de los elementos puede ser diferente. Si hay repeticiones, entonces las ocurrencias de los elementos repetidos también deben ser iguales para que dos arreglos sean iguales. Las aplicaciones de este problema –en el sector bancario y financiero– son muy amplias, por la gran cantidad de transacciones.

Si trabajas en Java, revisa este código:

```java
import java.util.*;
static boolean areEqual(int arr1[], int arr2
    []) {
 int n = arr1.length;
 int m = arr2.length;
 if (n != m)
  return false;
 Map<Integer, Integer> map = new HashMap<
     Integer, Integer>();
 int count = 0;
 for (int i = 0; i < n; i++) {
  if (map.get(arr1[i]) == null)
   map.put(arr1[i], 1);
  else {
   count = map.get(arr1[i]);
   count++;
   map.put(arr1[i], count);
  }
 }
 for (int i = 0; i < n; i++) {
  if (!map.containsKey(arr2[i]))
   return false;
  if (map.get(arr2[i]) == 0)
   return false;
  count = map.get(arr2[i]);
  --count;
  map.put(arr2[i], count);
 }
 ..............
}
```

Si trabajas en Python, revisa este código:

```python
from collections import defaultdict
def areEqual(arr1, arr2, n, m):
 if (n != m):
  return False;
 count = defaultdict(int)
 for i in arr1:
  count[i] += 1
 for i in arr2:
  if (count[i] == 0):
   return False
  else:
   count[i] -= 1
 ............
```

A (10 %) Completa la última línea.....

B (10 %) ¿Cuál es la complejidad asintótica, en el peor de los casos, de `areEqual`? Donde n es el número de elementos del primer arreglo y m el número de elementos del segundo arreglo. Es O(.....)

5. Grafos 10 %

Microsoft Game Studios es, hoy en día, uno de las empresas que controlan la saga de videojuegos de *Age of Empires*. Una de la peculiaridades que tienen los videojuegos de esta saga es que los participantes se pueden ubicar en islas, distribuídas sobre el mar, que deben controlar para poder lograr la victoria. Dada una matriz booleana 2D, encuentra el número de islas. Un grupo de 1s conectados forma una isla. Por ejemplo, la siguiente matriz contiene 5 islas.

```
Entrada : mat[][] = {{1, 1, 0, 0, 0},
                     {0, 1, 0, 0, 1},
                     {1, 0, 0, 1, 1},
                     {0, 0, 0, 0, 0},
                     {1, 0, 1, 0, 1}
Salida  : 5
```

El problema se puede resolver fácilmente aplicando *recorrido primero en profundidad (en inglés, DFS)* en cada punto. En cada llamado a DFS se visita una isla. Llamaremos a DFS en el siguiente nodo no visitado. El número de llamados a DFS da el número de islas.

Si trabajas en Java, revisa este código:

```java
class Islands {
 static final int ROW = 5, COL = 5;

 // Verificar si una celda (fila, columna)
 // se le puede llamar un DFS
 boolean isSafe(int M[][], int row, int col,
     boolean visited[][]) {
  return (row >= 0) && (row < ROW) && (col >=
      0) && (col < COL) && (M[row][col] == 1 &&
       !visited[row][col]);
 }

 // Un DFS en una booleana matriz. Este
 // considera como vecinos a los 8 adyacentes
 void DFS(int M[][], int row, int col, boolean
      visited[][]) {
  int rowNbr[] = new int[] { -1, -1, -1, 0, 0,
       1, 1, 1 };
  int colNbr[] = new int[] { -1, 0, 1, -1, 1,
      -1, 0, 1 };
  visited[row][col] = true;
  for (int k = 0; k < 8; ++k)
   if (isSafe(M, row + rowNbr[k], col + colNbr
       [k], visited))
    ..............
 }

 // Contar el numero de islas
 int countIslands(int M[][]) {
  boolean visited[][] = new boolean[ROW][COL];
  int count = 0;
  for (int i = 0; i < ROW; ++i)
   for (int j = 0; j < COL; ++j)
    if (M[i][j] == 1 && !visited[i][j]) {
     DFS(M, i, j, visited);
     ++count;
    }
  return count;
}}
```

Si trabajas en Python, revisa este código:

```python
class Graph:
 def __init__(self, row, col, g):
  self.ROW = row
  self.COL = col
  self.graph = g

 # Verificar si una celda (fila, columna)
 # puede ser incluida en el llamado al DFS
 def isSafe(self, i, j, visited):
  return (i >= 0 and i < self.ROW and
          j >= 0 and j < self.COL and
   not visited[i][j] and self.graph[i][j])

 # Hacer DFS en una matriz 2D. Este
 # considera como vecinos los 8 adyacentes
 def DFS(self, i, j, visited):
  rowNbr = [-1, -1, -1,  0, 0,  1, 1, 1];
  colNbr = [-1,  0,  1, -1, 1, -1, 0, 1];
  visited[i][j] = True
  for k in range(8):
   if self.isSafe(i + rowNbr[k], j + colNbr[k
       ], visited):
    ..............

 # Retorna el
 # numero de islas que hay en la matriz
```

```
26    def countIslands(self):
27     visited = [[False for j in range(self.COL)]
           for i in range(self.ROW)]
28     count = 0
29     for i in range(self.ROW):
30      for j in range(self.COL):
31       if visited[i][j] == False and self.graph[i
             ][j] == 1:
32        self.DFS(i, j, visited)
33        count += 1
34     return count
```

A (10 %) Completa la línea 18 en Java o la 22 en Python .....

6. (Opcional) Tablas de Hash 10 %

Según el portal de *LeetCode*, este problema es común en entrevistas y es de alta dificultad. Dada un arreglo de números enteros `arr` con longitud n, encuentre la longitud de la subsecuencia más larga de manera que los elementos de la subsecuencia sean números enteros consecutivos. Los números consecutivos pueden estar en cualquier orden en el arreglo. Existen al menos dos formas de hacer este problema. ¿Cuál es la complejidad asintótica, para el peor de los casos, de cada una de ellas?

6.1. Primera solución

Si trabajas en Java, revisa este código:

```
1  int findLongestConseqSubseq(int arr[], int n){
2   HashSet<Integer> S = new HashSet<Integer>();
3   int ans = 0;
4   // Agregar a la tabla cada elemento de arr
5   for (int i = 0; i < n; ++i)
6    S.add(arr[i]);
7   // Probar cada posible inicio de la secuencia
8   // y su longitud optima
9   for (int i = 0; i < n; ++i) {
10   // si el elemento actual es el inicio
11   if (!S.contains(arr[i] - 1)) {
12    // Probemos los siguientes elementos
13    int j = arr[i];
14    while (S.contains(j))
15     j++;
16     // y actualizamos la longitud optima
17     if (ans < j - arr[i])
18      ans = j - arr[i];
19    }
20   }
21   return ans;
22  }
```

Si trabajas en Python, revisa este código:

```
1  def findLongestConseqSubseq(arr, n):
2   s = Set() #Conjunto hecho con tablas de hash
3   ans = 0
4   # Agregar a la tabla cada elemento de arr
5   for ele in arr:
6    s.add(ele)
7   # Probar cada posible inicio de la secuencia
8   # y su longitud optima
9   for i in range(n):
10   # si el elemento actual es el inicio
11   if (arr[i]-1) not in s:
12    # Probemos los siguientes elementos
13    j = arr[i]
14    while(j in s):
15     j+= 1
16     # y actualizamos la longitud optima
17     ans = max(ans, j-arr[i])
18   return ans
```

6.2. Segunda solución

Si trabajas en Java, revisa este código:

```
1  int findLongestConseqSubseq(int arr[], int n)
       {
2   Arrays.sort(arr);
3   int ans = 0, count = 1;
4   // encontrar la longitud maxima
5   // recorriendo el arreglo
6   for(int i = 1; i < n; i++) {
7    // Si el elemento actual es
8    // igual al anterior mas 1
9    if (arr[i] == arr[i - 1] + 1)
10    count++;
11   else
12    count = 1;
13   // Actualizar la respuesta
14   ans = Math.max(ans, count);
15  }
16  return ans;
17 }
```

Si trabajas en Python, revisa este código:

```
1  def findLongestConseqSubseq(arr, n):
2   arr.sort()
3   ans = 0
4   count = 1
5   # encontrar la longitud maxima
6   # recorriendo el arreglo
7   for i in range (1, n):
8    # Si el elemento actual es
9    # igual al anterior mas 1
10   if arr[i] == arr[i - 1] + 1:
11    count += 1;
12   else:
13    count = 1
14   # Actualizar la respuesta
15   ans = max(ans, count)
16  }
17  return ans
18 }
```

6.3. Pregunta

A (10 % Extra) ¿Cuál es la complejidad **asintótica**, en el peor de los casos, del los algoritmos anteriores? El primero es O(.....) y el segundo es O(.....). Por esta razón, yo elejiría la ..... solución.