# Estructuras de Datos 1 - ST0245
# Examen Parcial 2 - Jueves (033)

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

Octubre 24 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. Tablas de Hash 20%

En los videojuegos, las tablas de hash se usan para guardar la información de las armas; por ejemplo, su ataque, su velocidad y su duración. Considera la siguiente función de hash para cadenas de caracteres. La constante `TABLE_SIZE` representa el tamaño máximo del arreglo con el que se representa internamente la tabla de hash.

```
1  private int funcionHash(String k){
2    return ((int) k.charAt(0)) % TABLE_SIZE;
3  }
```

a (10%) ¿Qué problema presenta esa función hash? Las cadenas...

i que terminan con la misma letra colisionan

ii que inician con la misma letra colisionan

iii cuya suma de los caracteres es la misma colisionan

iv que tienen los mismos caracteres colisionan

b (10%) ¿Cuál es la complejidad asintótica, en el peor de los casos, de `funcionHash(k)`? Donde $n$ es la longitud de la cadena $k$.

i $O(n)$

ii $O(n^2)$

iii $O(n \log n)$

iv $O(1)$

## 2. Listas 20%

Liko escribió un nuevo método para una lista simplemente enlazada. Desafortunadamente, Liko olvidó qué hace el método. ¿Podrías ayudarnos a decifrar qué retorna el método `mistery` y su complejidad?

```
1  class Node {
2     int val;
3     Node next;
4     Node(int d) { val = d; }
5  }
6  class Algorithms {
7    static Node mistery(Node l1, Node l2) {
8      Node temp = new Node(0);
9      Node p=temp;
10     Node p1=l1;
11     Node p2=l2;
12     while(p1!=null && p2!=null){
13       if(p1.val < p2.val){
14         p.next = p1;
15         p1 = p1.next;
16       }else{
17         p.next = p2;
18         p2 = p2.next;  }
19       p=p.next;                    }
20     if(p1!=null){
21       p.next = p1;     }
22     if(p2!=null){
23       p.next = p2;     }
24     return temp.next;
25   }
26 }
```

a (10%) ¿Qué retorna el algoritmo anterior?

i Una nueva lista donde `l2` está al final de `l1`

ii Suponiendo que `l1` y `l2` están ordenadas, una nueva lista con los elementos de ambas listas ordenados

iii Una nueva lista donde `l2` está al final de `l1`

iv La lista `l1`

b (10%) ¿Cuál es la complejidad asintótica, en el peor de los casos, del algoritmo anterior? Donde $n$ es la longitud de la lista que inicia en el nodo `l1` y $m$ es la longitud de la lista que inicia en el nodo `l2`

i $O(n \times m)$

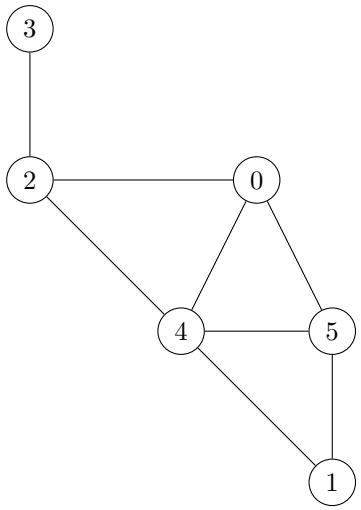
ii $O(n + m)$

iii $O(n \log m)$

iv $O(m \times (\log n)^2)$

## 3. Grafos 20%

Considera el siguiente grafo no dirigido:




a) (10%) ¿Cuál es un recorrido de *búsqueda primero en profundidad* del grafo anterior, si como nodo inicial se toma el nodo 1?

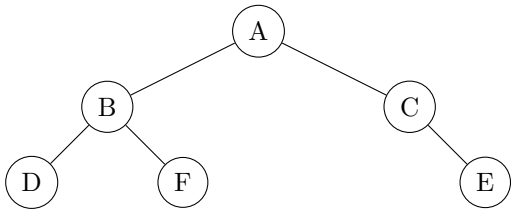
i) 1, 5, 0, 3, 2, 4
ii) 1, 4, 5, 0, 2, 3
iii) 1, 4, 0, 3, 5, 2
iv) 1, 5, 4, 0, 3, 2

b) (10%) ¿Cuál es un recorrido de *búsqueda primero en amplitud* del grafo anterior, si se toma como nodo inicial el nodo 1?

i) 1, 4, 5, 0, 2, 3
ii) 1, 5, 0, 2, 3, 4
iii) 1, 4, 2, 0, 3, 5
iv) 1, 3, 0, 4, 5, 2

## 4. Árboles binarios de búsqueda 20%

Considera el siguiente árbol:




a (10%) ¿Qué configuración de vértices $[A, B, C, D, E, F]$ hacen el árbol anterior un árbol de búsqueda binaria?

i $[5, 7, 1, 4, 9, 8]$
ii $[-1, 4, 9, 8, 6, 5]$
iii $[11, 4, 7, 9, 3, 1]$
iv $[8, 6, 9, 4, 7, 10]$

b (10%) ¿Cuál es la complejidad asintótica, en el peor de los casos, de encontrar un elemento en un árbol de búsqueda binaria que se autobalancea como, por ejemplo, el árbol rojo negro o el árbol AVL?

i $O(n)$
ii $O(1)$
iii $O(\log n)$
iv $O(n \log n)$

## 5. Pilas 20%

Dijkstra escribió un algoritmo para convertir una *expresión matemática infija* en *notación polaca inversa*, pero se le borraron algunas líneas y no había hecho *commit* en git. La notación polaca inversa evita el uso de paréntesis; para lograrlo primero se colocan los operandos y luego los operadores. Considera los siguientes ejemplos:

- `postfix(``( 5 + 7 ) * 2'')` retorna ``5 7 + 2 *''
- `postfix(``5 + 7 / 2'')` retorna ``5 7 2 / +''

```
1  String postfix(String infix) {
2   String ops = "+-*/";
3   StringBuilder output = new StringBuilder();
4   Deque<String> stack  = new LinkedList<>();
5   for (String token : infix.split(" ")) {
6    if (ops.contains(token)){//es un operador
7       while ( ! stack.isEmpty() && isHigherPrec
            (token, stack.peek()))
8         output.append(stack.pop()).append(' ');
9       stack.push(token);
10   } else if (token.equals("(")) { // es un (
11      stack.push(token);
12   } else if (token.equals(")")) { // es un )
13      while ( ! stack.peek().equals("("))
14        output.append(stack.pop()).append(' ');
15      stack.pop();
16   } else {   // es un digito
17      output.append(.....).append(' ');
18   }
19  }
20  while ( ! stack.isEmpty())
21     output.append(stack.pop()).append(' ');
22  return output.toString();
23 }
```

a (10%) Completa, por favor, la línea 21

---

b (10%) ¿Cuál es la complejidad asintótica, en el peor de los casos, de la operación `pop()` de una pila?

i $O(n^2)$
ii $O(n)$
iii $O(1)$
iv $O(n^2 \times \log n)$

La clase `StringBuilder` es una implemención de cadenas de caracteres con listas enlazadas. La clase `Deque` es una implementación de una pila con listas enlazadas. El método `isHigherPrec(a,b)` retorna verdadero si $a$ tiene mayor precedencia que $b$; de lo contrario, falso. El método `S.append(B)` agrega los caracteres de un `String` $B$ al final del `StringBuilder` $S$. El método `peek()` permite ver el elemento en el tope de una pila sin retirarlo. El método `pop()` retira un elemento de una pila y lo retorna. El método `push(e)` ingresa el elemento $e$ a una pila. El método `S.split(`` '')` genera un arreglo de cadenas separando la cadena $S$ por espacios. El ciclo `for(String token : infix.split(" "))` se lee como "para cada `String token` en el arreglo `infix.split(" ")`, de izquierda a derecha, haga..." El método `A.contains(b)` retorna verdadero si $b$ está en A; de lo contrario, falso.