Estructuras de Datos 1 - ST0245
Examen Parcial 2 - Martes (032)

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

Octubre 22 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 representamente 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 esta 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. Desafortunamente, Liko olvidó qué hace el método. ¿Podrías ayudarnos a decifrar qué retorna el método `mistery` y su complejidad?

```
1  class Node {
2     int data;
3     Node next;
4     Node(int d) { data = d; }
5  }
6  class AlgoritmosDeListas {
7   public static Node mistery(Node node) {
8     Node prev = null;
9     Node current = node;
10    Node next = null;
11    while (current != null) {
12      next = current.next;
13      current.next = prev;
14      prev = current;
15      current = next;
16    }
17    node = prev;
18    return node;
19  }
20 }
```

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

i La lista ordenada de mayor a menor

ii La lista con los elementos en orden inverso

iii La lista ordenada de menor a mayor

iv La misma lista

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 con el nodo `node`.

i $O(n)$

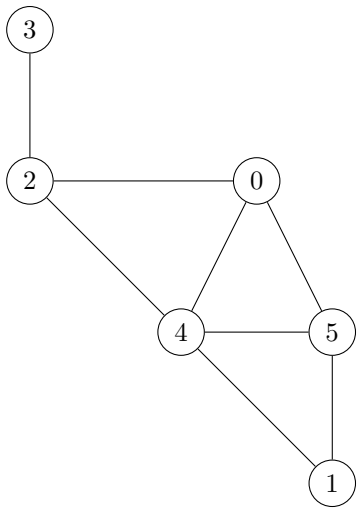
ii $O(n^2)$

iii $O(n \log n)$

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

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. Pilas 20%

Kefo escribió un nuevo método para evaluar una expresión aritmética 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:

- Esta entrada [“2”, “1”, “+”, “3”, “*”], es equivalente a esta expresión $((2+1)*3)$ y la respuesta es 9
- Esta entrada [“4”, “13”, “5”, “/”, “+”], es equivalente a $(4+(13/5))$ y la respuesta es 6

```
1  int evalRPN(String[] expression) {
2    int returnValue = 0;
3    String operators = "+-*/";
4    Stack<String> stack = new Stack<String>();
5    for(String t : expression){
6      if(!operators.contains(t)){
7          .............;
8      }else{
9        int a = Integer.valueOf(stack.pop());
10       int b = Integer.valueOf(stack.pop());
11       int index = operators.indexOf(t);
12       switch(index){
13         case 0:
14          stack.push(String.valueOf(a+b));
15          break;
16         case 1:
17          stack.push(String.valueOf(b-a));
18          break;
19         case 2:
20          stack.push(String.valueOf(a*b));
21          break;
22         case 3:
23          stack.push(String.valueOf(b/a));
24          break;
25        }
26      }
27    }
28    returnValue = Integer.valueOf(.............);
29    return returnValue;
30  }
```

a (10%) Completa, por favor, la línea 7

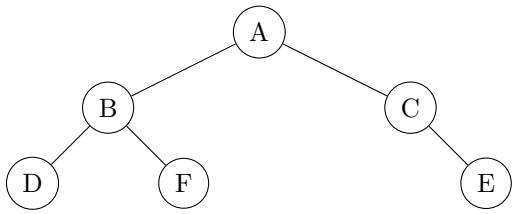
---

b (10%) Completa, por favor, la línea 28

El método `String.valueOf(a)` convierte el entero a en un String. El ciclo `for(String t : expression)` se lee como “para cada `String t` en el arreglo `expression`, de izquierda a derecha, haga...” El método `A.contains(b)` retorna verdadero si b está en A; de lo contrario, falso. El método `operators.indexOf(t)` retorna la posición en la cadena `operators` donde aparece el elemento t.

5. 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 sea un *árbol binario de búsqueda*?

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 NO se autobalancea?

i $O(n)$

ii $O(1)$

iii $O(\log n)$

iv $O(n \log n)$