

Estructura de datos - UFM

David Gabriel Corzo Mcmath

2020-Jan-06 07:08:59

Índice general

1. Clase introductoria - 2020-01-06	4
1.1. Datos del auxiliar	4
1.2. Información preliminar	4
1.3. Dos formas de pasar parámetros a una función	4
1.4. Ver:	4
2. Clase introductoria - 2020-01-08	5
2.1. Charla de programas del curso	5
3. Clase de avance de tarea # 2	6
4. Clase	7
4.1. Unit testing	7
4.2. Postman	7
4.3. Jmeter	7
5. Clase - 2020-01-20	8
5.1. Data structure	8
5.1.1. SOLID	8
5.1.2. The 3 steps	8
5.2. Elementary data structure organization	9
5.2.1. Clasificaciones	9
5.3. Abstract data types	9
5.4. Tarea	9
5.4.1. Asert - es parte de la prueba de integración	9
6. Clase - 2020-10-22	10
6.1. Estructuras de datos no es igual Algoritmos	10
7. Clase - 2020-01-27	12
7.1. Array	12
7.1.1. Storing values in the array	12
7.1.2. Caché	13
7.2. Two-Dimensional Array	13
7.3. Multi-Dimensional array	13
8. Clase - 2020-01-29	16
8.1. Debugging tools	16
8.2. Ejemplos	16
9. Clase - 2020-02-03	17
9.1. Strings	17

ÍNDICE GENERAL

10. Clase - 2020-02-12	19
10.1. Estructuras de datos	21
11. Clase - 2020-02-17	22
12. Clase - 2020-02-19	23
12.1. Linked list implemented with Stack	23
13. Clase - 2020-03-18	24
13.1. Binary Search Tree	24
13.1.1. Requerimientos	24

Capítulo 1

Clase introductoria - 2020-01-06

1.1. Datos del auxiliar

- tortola@ufm.edu
- 56904805

1.2. Información preliminar

- Sugerencia de lenguaje: Java
- Se puede utilizar el lenguaje que quiera. Ideal Java por el aspecto de ser orientado a objetos.

1.3. Dos formas de pasar parámetros a una función

- ```
num = 0;
func SumOne(n){
 return n + 1;
}
num_2 = SumOne(num)
print(num)
print(num_2)
```

|                         |                       |
|-------------------------|-----------------------|
| Por valor<br>↳ 0<br>↳ 1 | Por referencia<br>↳ 1 |
|-------------------------|-----------------------|

Los de valor no modifican num, los de por referencia no lo modifican y otra lugar en memoria.

### 1.4. Ver:

- Postman para API's
- Jmeter
- Spring: para API's en Java

## Capítulo 2

# Clase introductoria - 2020-01-08

### 2.1. Charla de programas del curso

- IntelliJ Ultimate: IDE
- Jetbrains: conjunto de servicios para estudiantes
- Junit: para pruebas unitarias.

## Capítulo 3

# Clase de avance de tarea # 2

# Capítulo 4

## Clase

### 4.1. Unit testing

### 4.2. Postman

- Pasos para probar un API:

↓ New → Crear una colección.  
↓ Poner el URL al API.

### 4.3. Jmeter

- ↓ Test plan es mi route de los tests.
- ↓ Add → Thread → Thread groups
- ↓ Basic Test → Sampler → HTTP request
  - ↓ Server name or IP: nombre de dominio
  - ↓ Numero de puerto
- ↓ Agregamos un listener: Basic Test → Listener →

# Capítulo 5

## Clase - 2020-01-20

### 5.1. Data structure

- Es simplemente cómo en memoria tengo esa información disponible mientras estoy ejecutando esa información.
- *Ejemplo: Google, si google hubiera guardado una llave primaria por todo lo que existe estaríamos aún esperando, separan el software y las clasifican en diferentes estructuras de datos; es diferente transformar la data que manipularla.*

#### 5.1.1. SOLID

1. Single responsibility:
  - No hacer dos o tres métodos en una clase llamada “tareal”.
2. Open - Close
  -
3. L is for substitution
4. I
5. D

#### 5.1.2. The 3 steps

- Analyse the problem to determine basic operations that must be supported to interact with the data.
- Quantify the resource constraints for each operation.
- Select the data structure that best meets these requirements.

#### Recomendaciones

- “Todo trabajo tiene su estructura de datos perfecta.”

## 5.2. Elementary data structure organization

- Data
- Record
- File
- Key
- Values

### 5.2.1. Clasificaciones

- Fundamental data types: Linear, Non-linear.
- Primitive: int, char, float, double.
- Non-primitive: linked list, array.
- Data structures: **Nos preguntamos:** ¿una clase es una estructura de datos? depende, para el compilador sí; en memoria sí es una estructura de datos; Para nosotros no es una estructura de datos.

## 5.3. Abstract data types

- Solo la voy a usar.

## 5.4. Tarea

- Testing*
- Integration Unit
  - Prueba unitaria va directo a la clase. En este probas tu API
  - Prueba de integración va al medio de exposure.

### 5.4.1. Asert - es parte de la prueba de integración

- Es una prueba de confirmación, se puede colocar que algo está incluído.
- Es pasarle los asserts a Jmeter.

# Capítulo 6

## Clase - 2020-10-22

### 6.1. Estructuras de datos no es igual Algoritmos

- *Interesante: Algoritmos no implican código.*
- 20 % de esfuerzo va a construir el 80 % del trabajo.
- *Definición de “Modularización”:* proceso de dividir un algoritmo en varios módulos. Divide el problema en micro-problemitas.
- Top down approach: voy considerando las funcionalidades.
- Bottom up approach: decido hacer una funcionalidad específica.
- *Consultar el siguiente recurso:*
  - Behaviour driven design
  - Test driven design
- Algoritmo, tres bloques:
  - Recursos que va a necesitar
  - Tiempo que se va a tardar
  - Cantidad de memoria que se va a utilizar
- Time complexity:
  - Running times
  - Nos dice qué comportamiento va a tener
- Time-Space Trade-off:
  - Va ser útil tener una estructura de datos que pueda ocupar poco en memoria y al mismo tiempo tener una cantidad grande de operaciones.
- Expressing time and space complexity:
  - Tiempo es muy importante.
- Eficiencia de una algoritmo:
  - Un ciclo multiplica la complejidad del algoritmo.
  - Algunos algoritmos van a tener instrucciones ineficientes, cargar en memoria todos los datos y filtrarlos, esto es ineficiente.

- Algorithm efficiency:
  - Linear loops: la eficiencia es linal  $y = x$ .
  - Logarithmic loops: la eficiencia es logarítmicas  $f(n) = \log_{10}(n)$
  - Quadratic & doendent quadratic:  $f(n) = n^2$
- Notations:
  1. Big theta ( $\Theta$ ): en el peor de los escenarios, me lleva un poco a la realidad, enfocada en entender los dos límites, inferior o superior, que son el mínimo p máximo que me voy a tardar.
  2. Big O notation ( $O$ ): lo contrario que omega, se basa en el peor escenario, **Nos preguntamos:** ¿cuanto se va a tardar en el peor de los casos?; **Pésima condición**.
  3. Big omega notation ( $\Omega$ ): el mejor escenario, nos da una notación en la que se puede decir que en el mejor de los casos dado un input  $g(n)$  se va a tardar tal cantidad de tiempo. Si todo va bien → nos va dar una guía de escenarios óptimos en los que van a resultar. **Si todo va bien puedo llegar a tal punto.**
- Un algoritmo funcional va de la mano de algoritmos no funcionales.
- **Nos preguntamos:** ¿Cómo verifico si lo que cree es lo que me imaginé inicialmente?

|                         |                                                              |                               |
|-------------------------|--------------------------------------------------------------|-------------------------------|
| Behaviour Driven Design | Specification Testing<br>Integration testing<br>Init testing | ↓ Load testing<br>↑ Profiling |
| Test driven design      |                                                              |                               |

- Observaciones:
  - Cosas difíciles de determinar es qué estructura de datos usar.
  - **Definición de “specification testing”:** basarse en lo más alto de especificación de un user story.
  - **Definición de “integration testing”:** CI, continuous integration.
  - **Definición de “load testing”:** simular muchas interacciones con tu sistema, mecanismo que genera de manera dinámica simulación de uso en exceso.
    - Genero volumen
    - Genero Uso
    - Pero no me dice qué pasó solo en  $n$  cantidad de usuarios en un determinado momento con mi aplicación.
  - **Definición de “profiling”:** enfocado a entender qué está pasando en tiempo de ejecución.
    - Call tree view
    - CPU usage
    - Memory usage
    - Time spent
    - Networking
- Transformaciones:
  - Las manipulaciones en transformaciones

# Capítulo 7

## Clase - 2020-01-27

### 7.1. Array

- Colección de datos del mismo tipo.
- En memoria se hace de manera consecutiva.
- Con arraylists no se almacena de manera consecutiva.
- Elementos:
  - Data type: Any valid data type.
  - Name: How we'll identify.
  - Size: Maximum numbers of values to hold.
- En el procesador:
  - Tiene una eficiencia mayor.
  - Como se guarda consecutivamente solo se tiene que buscar una vez en memoria por ser consecutiva.
  - Por eso se necesita declarar con cuántos elementos inicializar el array.
- Tener presente cuánto ocupa en memoria cada tipo de elementos.
- La función .length recorre todo el array y retorna el número de elementos que tiene.

#### 7.1.1. Storing values in the array

- Cuando ingresamos información a un array lo hacemos por orden de índices.
- Accesar: por el índice [i].
  - System.out.println(A[i]); // Prints the element stored in that index in memory.
- Operations on arrays:
  - Deleting elements: cuando se elimina un elemento, se tiene que hacer un corrimiento de las cantidades correspondientes; si por ejemplo tiene que un array de n elementos, se desplaza indices  $n$  por los  $n + 1$ .
    - $A[i] = A[i + 1];$
- Merging arrays:
  - array1;

- array2;
  - merged = array1 + array2;
- Passing arrays to functions:
1. Passing entire arrays:
    - Le pasa todo el arreglo, con todo y la posición de memoria.
  2. Passing individual elements: esto permite trabajar el contenido del array por aparte, creo una copia y trabajo con la información, entonces no modiflico el array original, si paso la dirección de memoria del índice o de la posición de memoria si voy a alterar el arreglo original.
    - Passing data values
    - Passing addresses

### 7.1.2. Caché

1. Almacenamiento
2. Temporal
3. Fácil Acceso

#### Beneficios

- Tener más efectividad.
- Reduce el impacto cuando tenemos que interactuar .
- Está enfocado en leer.

## 7.2. Two-Dimensional Array

- Almacena matrices en memoria, tiene varias implicaciones:
  - Tener contabilidad de dos indices.
  - Agregar una línea implica agregar n cantidad de el array anidado.
- Se almacena de la misma manera que un array de una dimensión.
- Cuando se almacena un array bidimensional, se hacen operaciones matemáticas para ubicar en memoria los indices.
- Inicialización:
  - Se declaran valores al momento de declarar.
  - Se declaran valores parametrados.

## 7.3. Multi-Dimensional array

- Multidimensional array:
  - *Ejemplo: Graficador de imágenes*

$$\begin{array}{c}
 \text{Temporalidad} \\
 \overbrace{\quad\quad\quad}^{\text{Temporalidad}} \\
 grafo \quad \overbrace{[t_1]}^{\text{Temporalidad}} \quad \underbrace{[x_1][y_1]}_{\text{Cuadrícula}} = \underbrace{\text{Color}}_{255}
 \end{array}$$

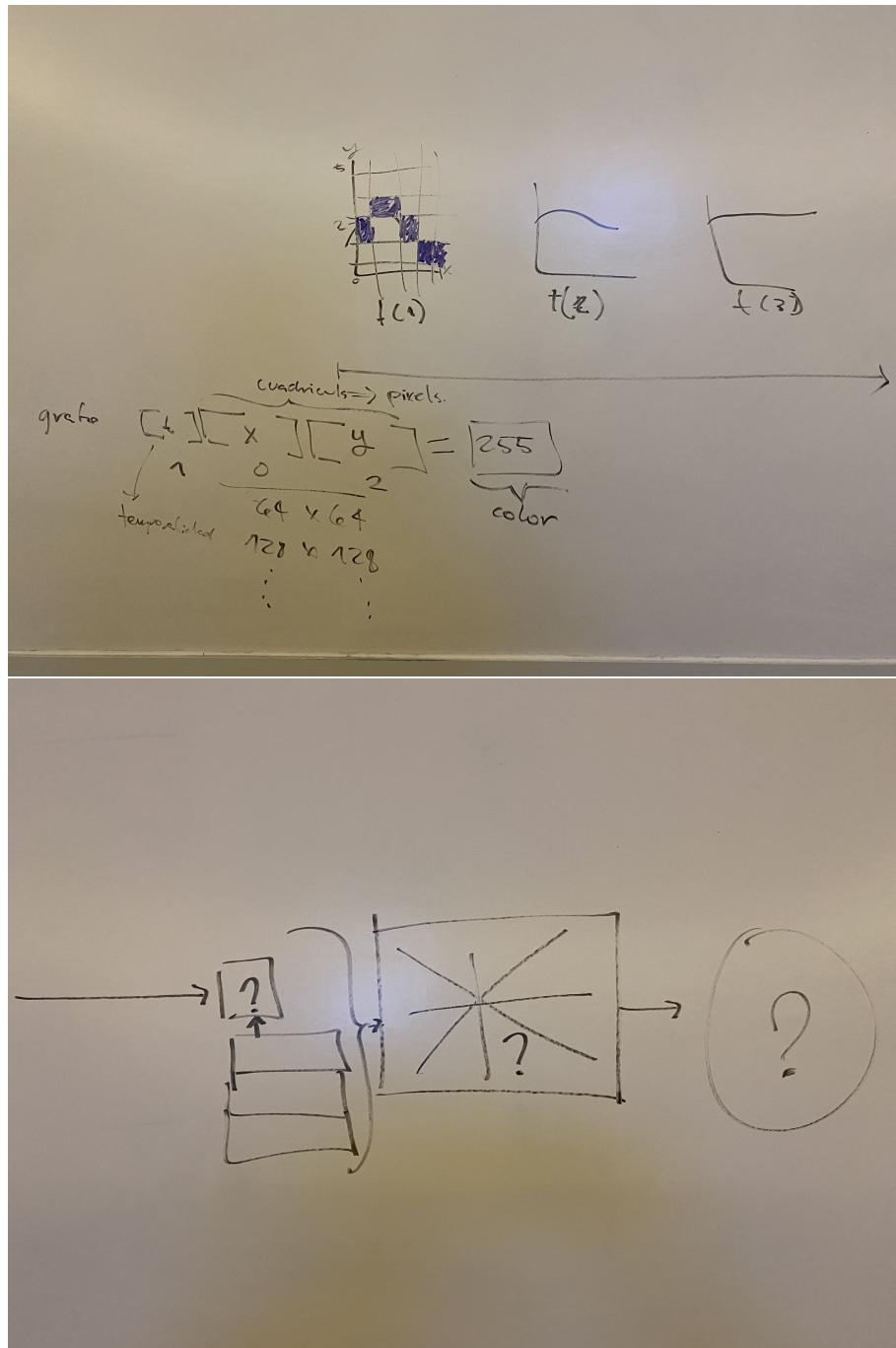


Figura 7.1: Ejemplo de multidimensional arrays & el black box exercise

- Entonces el tiempo puede tener manifestación en un gif, las imágenes que resultan se pintan en la pantalla. Mientras más pixeles tenemos mejor la calidad.
- Los arrays multidimensionales podemos combinarlos con objetos :
  - Nos va permitir más que un solo valor.
  - Se puede almacenar objetos en el array, por lo general se va a tener que tener el mismo tamaño.
  - En Java se puede declarar un array de tipos de datos objetos.

# Capítulo 8

## Clase - 2020-01-29

### 8.1. Debugging tools

- *Definición de “Step over”:* usa el debugger para ir a la siguiente línea.
- *Definición de “Step into”:* va directamente al scope del método.
- *Definición de “Step out”:* sale de “Step into”

### 8.2. Ejemplos

```
1 public class Debugging {
2 public static int factorial(int n) {
3 if (n == 0){
4 return 1;
5 } else {
6 return n * factorial(n-1);
7 }
8 }
9 public static void main(String[] args) {
10 int res = factorial(5);
11 System.out.println("factorial(n): " + res);
12 }
13 }
```

# Capítulo 9

## Clase - 2020-02-03

### 9.1. Strings

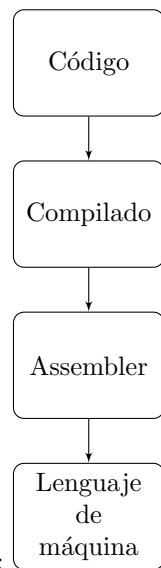
Los strings se almacenan de manera continua en memoria, se separan por medio de un **null byte** o 0:



\0 = NULL = NIL

- Hay varias maneras de inicializar un strings:

- char str[10]; // Lo declaro como un array y pide 10 bytes en memoria.
- char str[]; // Lo declaro como una array de indefinida cantidad de posiciones.
- char str[] = {'h','e','l','l','o',\0}



- Cómo el compilador funciona:

- Los strings son considerados como un array:

- Para sacar el lenght de un array:

```
1 function len(){
2 str = "hola";
3 i = 0;
```

```
4 while (str[i] != NIL){
5 i++;
6 }
7 return i;
8 }
```

- Estas operaciones de los strings tienen sus abstracciones en código de asembler.
- <https://repl.it/languages/c> Para probar código de c en línea.

```
1 #include <stdio.h>
2
3 int main(void) {
4 char second_string[6] = {'h','e','l','l','o'};
5 printf("Second string: %s\n", second_string);
6
7 char third_string[6] = "Hello";
8 printf("Third string: %s\n", third_string);
9 printf("This is an element is one before NIL =>
10 \"%c\"\n",third_string[5]);
11
12 char string[] = "hello this doesn't delimit the memory size";
13 printf("String: %s\n",string);
14
15 return 0;
 }
```

# **Capítulo 10**

**Clase - 2020-02-12**

Strings:

1. Strings are “arrays of characters”. \*\*\*
2. Every string is terminated with a “null byte \0” \*\*\*
3. If a string is given as “AB CD”, THE LENGTH IS “5”. \*\*\*
4. Char mesg[100]; can store a maximum “100” of characters. \*\*\*
5. S1>S2 means “it evaluates the lengths of the strings and returns true if S1 is bigger than S2”. \*\*\*

## 10.1. Estructuras de datos

- La diferencia entre `typedef struct` y sólo `struct` es que el `typedef` define un tipo de dato.
- Intento de crear un struct anidada:

```
1 typedef struct student {
2 typedef struct name {
3 String first_name;
4 String mid_name;
5 String last_name;
6 }
7 int r_10;
8 String course;
9
10 typedef struct DOB {
11 int dd;
12 int mm;
13 int yy;
14 }
15 double fees;
16 }
```

- Close example to using unions in real life:

```
1 class Animal {
2 int sound() {...};
3 }
4
5 class Dog inherits Animal {
6 String sound() {...};
7 int sound(){...};
8 }
```

# **Capítulo 11**

**Clase - 2020-02-17**

# Capítulo 12

## Clase - 2020-02-19

### 12.1. Linked list implemented with Stack

- List:
  - Insert
  - Search/Traverse
  - Delete
- Stack:
  - Push: insert on top
  - Pop: extract last inserted item
  - Peek: who is on top
  - Clear: empty stack
- Una interfaz lo que define que contrato de implementación se tiene.
- Opcional: por que hace sentido que la clase stack extienda de vector.

# Capítulo 13

## Clase - 2020-03-18

### 13.1. Binary Search Tree

- Busqueda rápida
- Valir mínimo y máximo de forma directa
- Hijo izquierdo menor al padre
- Hio derecho mayor al padre
- Operaciones con árbol no balanceado  $O(n)$
- Operaciones con árbol balanceado:  $O(\log(n))$

#### 13.1.1. Requerimientos

- Instancear nuevo BST.
- Insertar valores.
- Traverse
- Obtener valor mínimo y valor máximo.

```
1 import gc
2 from memory_profiler import profile
3
4
5 class Node:
6 def __init__(self, val):
7 self.value = val
8 self.left_child = None
9 self.right_child = None
10
11 def insert(self, data):
12 if (self.value == data):
13 return False # No se pueden valores duplicados
14 elif self.value > data:
15 if (self.left_child):
16 return self.left_child.insert(data)
17 else:
18 self.left_child = Node(data)
```

```
19 return True
20 else:
21 if (self.right_child):
22 return self.right_child.insert(data)
23 else:
24 self.right_child = Node(data)
25 return True
26
27 def find(self, key):
28 if (self.value == key):
29 return True
30 elif self.value > key:
31 if (self.left_child):
32 return self.left_child.find(key)
33 else:
34 return False
35 else:
36 if (self.right_child):
37 return self.right_child.find(key)
38 else:
39 return False
40
41 def traverse(self):
42 print(self.value)
43 if (self.left_child):
44 return self.left_child.traverse()
45 if (self.right_child):
46 return self.right_child.traverse()
47
48 def find_min(self):
49 if (self.left_child):
50 return self.left_child.find_min()
51 else:
52 return self.value # Retorna el valor que tenga en ese momento
53
54 def find_max(self):
55 if (self.right_child):
56 return self.right_child.find_max()
57 else:
58 return self.value # Retorna el valor que tenga en ese momento
59
60
61 class Tree:
62 def __init__(self):
63 self.root = None
64
65 def insert(self, data):
66 if (self.root):
67 return self.root.insert(data)
68 else:
69 self.root = Node(data)
70 return True
71
72 def find(self, key):
```

```
73 if (self.root):
74 return self.root.find(key)
75 else:
76 return False # No existe
77
78 def traverse(self):
79 if (self.root):
80 self.root.traverse()
81 else:
82 return False
83
84 def find_min(self):
85 if (self.root):
86 return self.root.find_min()
87 else:
88 return False
89
90 def find_max(self):
91 if (self.root):
92 return self.root.find_max()
93 else:
94 return False
95
96 @profile
97 def main():
98 bst = Tree()
99 # print(bst.traverse())
100 # print(bts.insert(10)) # True
101 # print(bts.insert(15)) # True
102 # print(bts.find(15)) # True
103 # print(bts.find(1)) # False
104 bst.insert(5)
105 bst.insert(15)
106 bst.insert(20)
107 bst.insert(21)
108 bst.insert(18)
109 bst.insert(30)
110 bst.insert(120)
111 bst.insert(-15)
112 bst.insert(-100)
113 # bst.traverse()
114 print("Min val: ",bst.find_min())
115 print("Max val: ",bst.find_max())
116
117 for obj in gc.get_objects():
118 if isinstance(obj,Tree):
119 print("Tree:", obj)
120 for obj in gc.get_objects():
121 if isinstance(obj, Node):
122 print("Node: ", obj.value, obj)
123 print("Right child: ", obj.right_child)
124
125
126
```

```
127 if __name__ == "__main__":
128 main()
```

Insertar: -15,-18,5,15,20,18,21,30 al BST

