ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES

FACULTAD DE INGENIERÍA Y CIENCIAS

# Estructura de Datos - Tarea

Libreta de Contactos empleando Estructuras de Datos

Nicolás Reyes Gòmez[1].
Ingeniería civil en informática y telecomunicaciones.
nicolas.reyes@mail.udp.cl

---

Profesor: Nicolás Rosso Chamorro
Ayudante: John Bidwell Boitano

# Índice

# 1. Introducción

A continuación se procede a la implementación de una libreta de contactos en python, empleando 5 tipos de estructuras de datos:

- Lista Simple
- Árbol Binario de Búsqueda
- Árbol AVL
- Árbol 2-3
- Hash

Los respectivos ordenes de convergencia respecto al tiempo de ejecución al momento de insertar, buscar y eliminar un elemento se detallan más adelante

# 2. Análisis

## 2.1. Lista Simple

### 2.1.1. Orden De Convergencia

Si consideramos n nodos tenemos que:

- insertar: Su orden es O(1) en todos los casos
- buscar: El peor caso es O(n)
  Para n = 10 : O(n)= 10
  Para n = 20 : O(n)= 20
  Para n = 100 : O(n)= 100
  Para n = 1000: O(n)= 1000

- eliminar: El peor caso es O(n)
  Para n = 10 : O(n)= 10
  Para n = 20 : O(n)= 20
  Para n = 100 : O(n)= 100
  Para n = 1000: O(n)= 1000

## 2.2. Arbol Binario de Busqueda

### 2.2.1. Orden De Convergencia

Si consideramos n nodos tenemos que:

- insertar: El peor caso es O(n)
  Para n = 10 : O(n)= 10
  Para n = 20 : O(n)= 20
  Para n = 100 : O(n)= 100
  Para n = 1000: O(n)= 1000

- buscar: El peor caso es O(n), el mejor caso es O(logn)
  Para n = 10 : O(n)= 10 u O(logn) = 3.32193
  Para n = 20 : O(n)= 20 u O(logn) = 4.32193
  Para n = 100 : O(n)= 100 u O(logn) = 6.64386
  Para n = 1000: O(n)= 1000 u O(logn) = 7.64386

- eliminar: El peor caso es log(n)
  Para n = 10 : O(logn) = 3.32193
  Para n = 20 : O(logn) = 4.32193
  Para n = 100 : O(logn) = 6.64386
  Para n = 1000: O(logn) = 7.64386

## 2.3. Arbol AVL

### 2.3.1. Orden De Convergencia

Si consideramos n nodos tenemos que:

- insertar: El peor caso es O(logn)
  Para n = 10 : O(logn) = 3.32193
  Para n = 20 : O(logn) = 4.32193
  Para n = 100 : O(logn) = 6.64386
  Para n = 1000: O(logn) = 7.64386

- buscar: El peor caso es O(logn)
  Para n = 10 : O(logn) = 3.32193
  Para n = 20 : O(logn) = 4.32193
  Para n = 100 : O(logn) = 6.64386

Para n = 1000: O(logn) = 7.64386

- eliminar: El peor caso es O(logn)
  Para n = 10 : O(logn) = 3.32193
  Para n = 20 : O(logn) = 4.32193
  Para n = 100 : O(logn) = 6.64386
  Para n = 1000: O(logn) = 7.64386

## 2.4. Arbol 2-3

### 2.4.1. Orden De Convergencia

Si consideramos n nodos tenemos que:

- insertar: El peor caso es O(n)
  Para n = 10 : O(n)= 10
  Para n = 20 : O(n)= 20
  Para n = 100 : O(n)= 100
  Para n = 1000: O(n)= 1000

- buscar: El peor caso es O(logn)
  Para n = 10 : O(logn) = 3.32193
  Para n = 20 : O(logn) = 4.32193
  Para n = 100 : O(logn) = 6.64386
  Para n = 1000: O(logn) = 7.64386

- eliminar: El peor caso es log(n)
  Para n = 10 : O(logn) = 3.32193
  Para n = 20 : O(logn) = 4.32193
  Para n = 100 : O(logn) = 6.64386
  Para n = 1000: O(logn) = 7.64386

## 2.5. Hash

### 2.5.1. Orden De Convergencia

Si consideramos n nodos tenemos que:

- insertar: El peor caso es O(1)
  Para n = 10 : O(1)= 1
  Para n = 20 : O(1)= 1
  Para n = 100 : O(1)= 1
  Para n = 1000: O(1)= 1

- buscar: El peor caso es O(n), el mejor es O(1)
  Para n = 10 : O(n)= 10 u O(1) = 1
  Para n = 20 : O(n)= 20 u O(1) = 1
  Para n = 100 : O(n)= 100 u O(1) = 1
  Para n = 1000: O(n)= 1000 u O(1) = 1

- eliminar: El peor caso es O(n), el mejor es O(1)
  Para n = 10 : O(n)= 10 u O(1)= 1
  Para n = 20 : O(n)= 20 u O(1) = 1
  Para n = 100 : O(n)= 100 u O(1) = 1
  Para n = 1000: O(n)= 1000 u O(1) = 1

# 3. Conclusión

Podemos notar que las mejores según las operaciones que queramos realizar y el tipo de archivo que estemos almacenando debemos elegir un tipo de estructura por sobre las demás, por ejemplo si queremos insertar, la mejor opción es una lista o un hash, en el caso de las búsquedas o de eliminar, si tenemos muchas colisiones en nuestros archivos las mejores opciones resultan ser los árboles, sobre todo el AVL o el 2-3, sim embargo si por el contrario tenemos bajas colisiones la mejor opción es emplear un hash, por lo tanto resulta fundamental realizar este análisis previo al momento de implementar una estrucutra de datos.