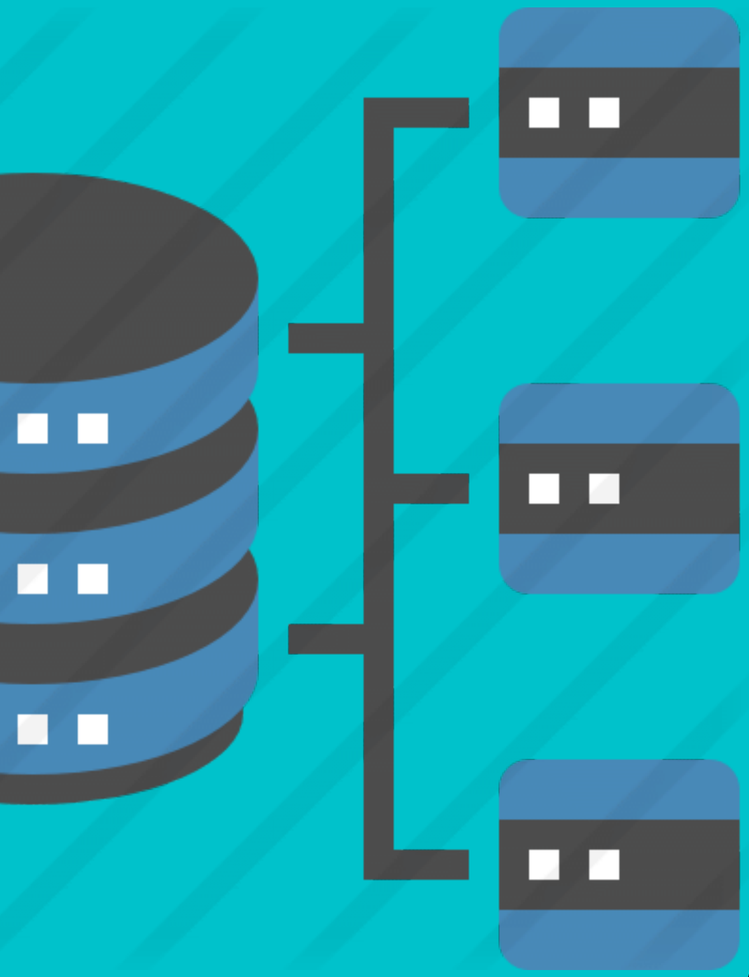


UNIVERSIDAD NACIONAL DE COLOMBIA

Gestión Integral del Parque Automotor

Primera entrega - Estructuras de Datos

Andrés Camilo Cardona Carrasquilla
María Paula Calderón Jaimes
Juan Sebastián Pachón Carvajal

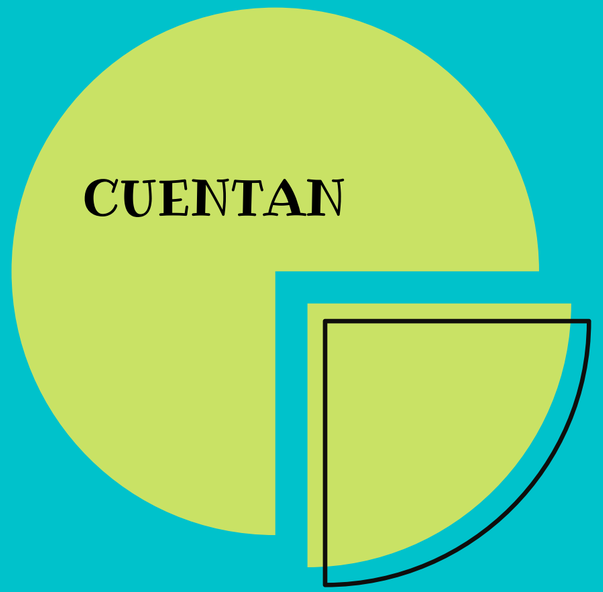
Problema a solucionar

EN COLOMBIA:

SEGUN RUNT

El 56% de los conductores no cumple con la revisión técnico- mecánica

No se cuenta con un Departamento de gestión de flotas



SEGUN RUNT

SEGURO OBLIGATORIO DE ACCIDENTES DE TRÁNSITO

- No se conocen las fechas de las siguientes revisiones
- Lo que genera retrasos en operaciones de empresas

Requerimientos

FUNCIONALES DE LA APP

Garantizar que los vehículos operen en óptimas condiciones

Facilitar la gestión y administración de flotas

Señalar las fechas y periodicidad de mantenimientos

Optimizar el uso de vehículos en empresas

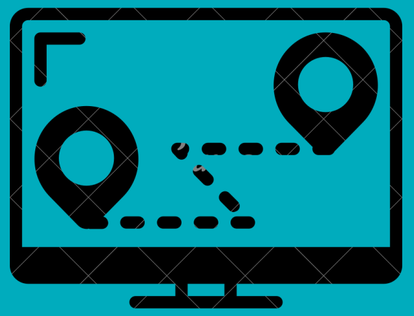
Registro de propiedades y elementos de vehículos

Obtención de información de la flota o vehículos en específico

Nuestros objetivos

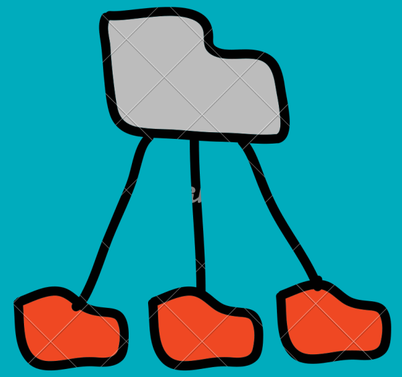
AL USAR ESTRUCTURAS DE DATOS

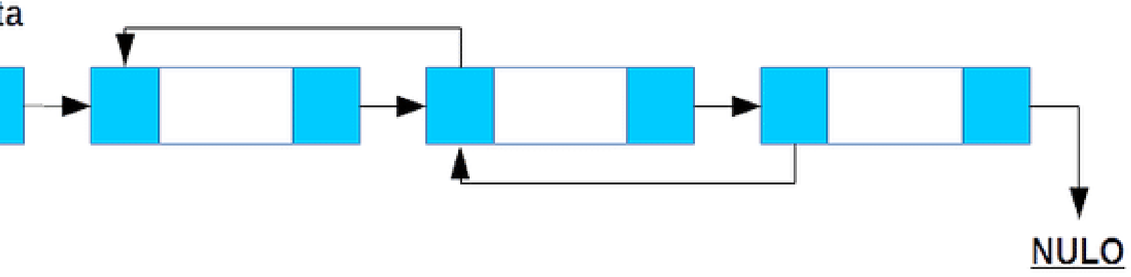
Definir
acciones a realizar mediante rutas.



Establecer
relaciones y ordenar por tipología y jerarquía

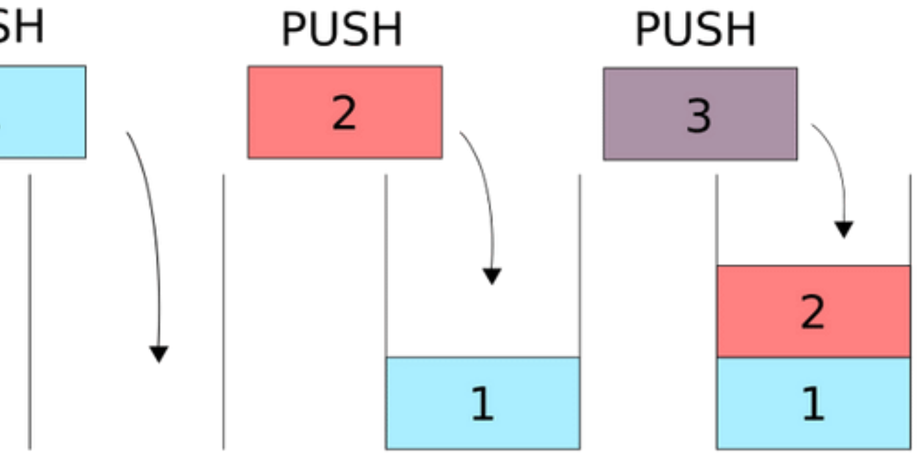
Ordenar
e instanciar de forma cronológica.

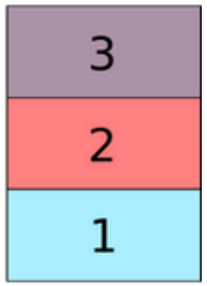




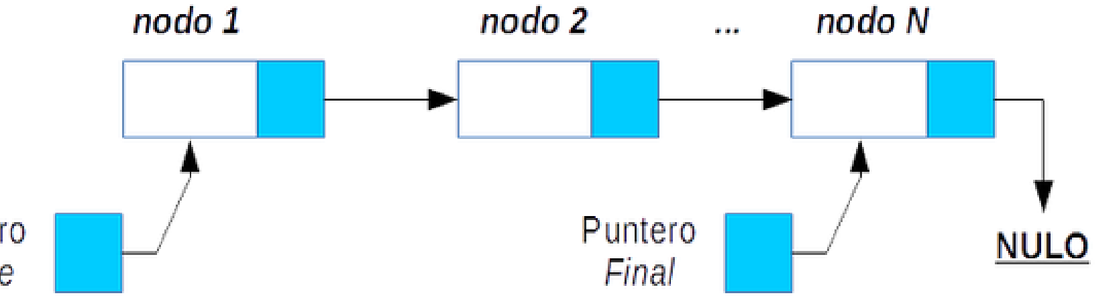
LISTAS

(Imagen de creación propia)





PILAS



(Imagen de creación propia)

COLAS

ESTRUCTURAS DE DATOS

Utilizadas en la app

LISTAS

1

SUCESIÓN DE NODOS SECUENCIALES

Cada uno con un puntero

2

EL TAMAÑO DE LA INFORMACIÒN

Puede variar

3

CADA NODO PERMITE

Acceder a los otros nodos

4

NOS PERMITEN AGREGAR INFORMACIÒN

O al principio, a nivel intermedio o al final

```java
public class Lista<T extends Comparable<T>> {

    private DoubleNode<T> head;

    public Lista() { head = null; }

    public boolean insert(T item){

        boolean inserted;
        DoubleNode<T> ptr, NodeI;
        inserted = false;
        ptr = head;
        while(ptr != null && ptr.getNext() != null && !ptr.getData().equals(item)){
            ptr = ptr.getNext();
        }
        DoubleNode newi = new DoubleNode(item);
        if(ptr == null){
            inserted = true;
            head = newi;
        }else{
            if(!ptr.getData().equals(item)){
                inserted = true;
                ptr.setNext(newi);
                newi.setBack(ptr);
            }
        }

        return  inserted;
```

LISTAS

PILAS

ESTRUCTURAS LIFO:

Último en entrar, primero en salir

SON MUCHO MÁS ÚTILES

En procedimientos recursivos

CENTRADAS EN:

Insertar y eliminar elementos

PERMITE TRANSFERIR DATOS

De forma mucho más rápida

PILAS

```java
public class Pila<T extends Comparable> {
    private DoubleNode<T> top;

    public Pila() { top = null; }

    public boolean empty() { return (top == null); }

    public void push(T data){
        DoubleNode next = new DoubleNode<T>(data);
        if (top != null) {
            top.setNext(next);
            next.setBack(top);
        }
        top = next;

    }
    public T pop(){
        if(empty()) throw new RuntimeException("Stack is empty");
        T data = top.getData();
        top = top.getBack();
        return data;
```

COLAS

1

ESTRUCTURAS FIFO:

Primero en entrar, primero en salir

2

POLÍTICA ESPECIAL DE:

Inserción y eliminación de elementos

3

NODOS ENLAZADOS QUE

Permiten el justo tratamiento de datos

4

TIENE DOS PUNTOS DE ACCESOS

Front (para eliminación)
Rear (para ingresar datos)

```java
public class Cola<T extends Comparable> {

private NodeGeneric<T> front, rear;

    public Cola(){
        front = null;
        rear = null;
    }

    public void enqueue(T item ){
        NodeGeneric<T> newp= new NodeGeneric<T>(item);
        if(rear != null){
            rear.setNext(newp);
        }
        else{
            front = newp;
        }
        rear = newp;
    }

    public boolean empty() { return (front == null); }

    public T dequeue(){
        T item = null;
        if(!empty()){
            item = front.getData();
            front = front.getNext();
        }
```

COLAS