Taller Práctico

Estudiantes: Delanis Tejeira, Samir Salas **ID:** 4-829-1643, 4-833-843

Investigación

Procedimientos de búsqueda y ordenamiento de un arreglo

El procedimiento para buscar es un algoritmo que encuentra un elemento específico dentro de una estructura de datos, como un arreglo. El ordenamiento de un arreglo es el proceso de reorganizar sus elementos en un orden específico, como ascendente o descendente, utilizando algoritmos de ordenamiento. Estos algoritmos son fundamentales en programación para organizar y optimizar datos, y existen varios métodos como el de burbuja, selección, inserción y Quick Sort.

Se pueden mencionar unos métodos los cuales son:

- **Búsqueda secuencial**

Ejemplo

Si se desea encontrar la persona cuyo número de teléfono es 64-234567 en la guía telefónica de Osorno. Las guías de teléfonos están organizadas alfabéticamente por el nombre del abonado en lugar de por números de teléfono, de modo que deben explorarse todos los números, uno después de otro, esperando encontrar el número requerido. Dado que el arreglo no está en un orden prefijado, es probable que el elemento a buscar pueda ser el primer elemento, el último elemento o cualquier otro. De promedio, al menos el programa tendrá que comparar la clave de búsqueda con la mitad de los elementos del arreglo. El método de búsqueda secuencial funcionará bien con arreglos pequeños o no ordenados. La eficiencia de la búsqueda secuencial es pobre, tiene complejidad lineal, O(n).

Concepto

La búsqueda secuencial, también llamada búsqueda lineal busca un elemento de una lista utilizando un valor destino llamado clave es un método para encontrar un valor en una lista que consiste en revisar cada elemento de la lista uno por uno desde el principio. Continúa hasta que se encuentra el valor buscado o hasta que se han comprobado todos los elementos. La ventaja principal es que no requiere que la lista esté ordenada, pero su principal desventaja es que puede ser lenta si la lista es muy grande.

Sintaxis

La sintaxis de la búsqueda secuencial es un bucle que recorre un arreglo o lista elemento por elemento para comparar cada uno con el valor buscado. La forma más común es usar un ciclo for que itera desde el primer elemento hasta el final, verificando si el valor actual coincide con el objetivo. Si se encuentra, se puede devolver la posición; si el ciclo termina sin encontrarlo, el elemento no está en la lista.

- **Push Down**

Ejemplo

En lugar de cargar todos los datos en memoria y luego filtrarlos en la aplicación, la consulta "pushdown" filtra los datos en la base de datos antes de enviarlos, lo que resulta en un procesamiento mucho más rápido.

El concepto

"pushdown" en programación se refiere a mover la lógica de cálculo de una capa superior (como la aplicación) a una capa inferior (como la base de datos) para optimizar el rendimiento. Es una técnica para acercar los datos al punto de procesamiento, reduciendo la transferencia de datos y mejorando la eficiencia, especialmente en bases de datos que manejan grandes volúmenes de información.

Sintaxis

Q es el conjunto de estados.

∑es el conjunto de símbolos de entrada.

Γ es el conjunto de símbolos pushdown (que se pueden empujar y sacar de la pila).

q0 es el estado inicial.

Z es el símbolo de empuje inicial (que está presente inicialmente en la pila).

F es el conjunto de estados finales.

δ es una función de transición que asigna Q x {Σ ∪ ∈} x Γ a Q x Γ*. En un estado dado, el PDA leerá el símbolo de entrada y el símbolo de la pila (parte superior de la pila), se moverá a un nuevo estado y cambiará el símbolo de la pila.

Base de datos MYSQL:

Definición

MySQL es el sistema de gestión de bases de datos de código abierto más popular del mundo. Las bases de datos son los repositorios de información esencial para todas las aplicaciones de software. Por ejemplo, cada vez que alguien realiza una búsqueda en Internet, inicia sesión en una cuenta o completa una transacción, una base de datos almacena la información para poder acceder a ella en el futuro. MySQL sobresale en esta tarea.

SQL, que significa lenguaje de consulta estructurado (Structured Query Language), es un lenguaje de programación que se utiliza para recuperar, actualizar, suprimir y manipular datos en bases de datos relacionales. MySQL se pronuncia oficialmente "My es-kiu-el", pero "my sequel" es una variación común. Como sugiere el nombre, MySQL es una base de datos relacional basada en SQL diseñada para almacenar y gestionar datos estructurados.

¿Qué se requiere para ser instalado?

Hardware

- **Memoria RAM**: Al menos 512 MB de RAM, aunque se recomienda 1 GB o más para un mejor rendimiento.
- **Espacio en Disco**: Un mínimo de 500 MB de espacio en disco para la instalación básica. Es recomendable tener más espacio disponible para bases de datos grandes.
- **Procesador**: Un procesador moderno compatible con 64 bits es preferible para ejecutar MySQL de manera óptima.

Software

- **Sistema operativo:**
    - **Windows:** Windows 10 o posterior (32 o 64 bits), con versiones de Windows Server compatibles.
    - **macOS:** Compatible con la versión de MySQL que se instale.
    - **Linux:** Varias distribuciones son compatibles, como Ubuntu.
- **Protocolo de red:** Soporte para el protocolo TCP/IP.
- **Herramientas de instalación:** Un cliente FTP con reanudación es útil para descargas grandes, aunque se puede usar el instalador web.

¿Qué se necesita para hacer una conexión PHYTON-MYSQL? Explique

Para conectar Python con MySQL, necesitas instalar el conector de Python para MySQL usando pip, importar la biblioteca mysql.connector en tu código, y

usar mysql.connector.connect() con los detalles de conexión (host, usuario, contraseña, base de datos) en un bloque try...except para manejar errores. Es fundamental obtener un objeto cursor para ejecutar consultas SQL y, finalmente, cerrar la conexión y el cursor cuando hayas terminado.

Requisitos para la conexión:

1. Tener MySQL instalado y ejecutándose
2. Instalar el conector de Python: pip install mysql-connector-python
3. Conocer los datos de conexión: host, usuario, contraseña, nombre de base de datos

¿Qué es una replicación en una base de Datos? Sustente su respuesta

La replicación en MySQL es el proceso de copiar automáticamente los datos de un servidor de base de datos de origen (maestro) a uno o más servidores de réplica (esclavos). Esto se utiliza para mejorar la escalabilidad, la alta disponibilidad, la redundancia geográfica, la tolerancia a fallos y para descargar tareas como copias de seguridad y análisis del servidor maestro. La replicación es asíncrona por defecto, lo que significa que las réplicas no necesitan estar conectadas permanentemente para recibir actualizaciones.

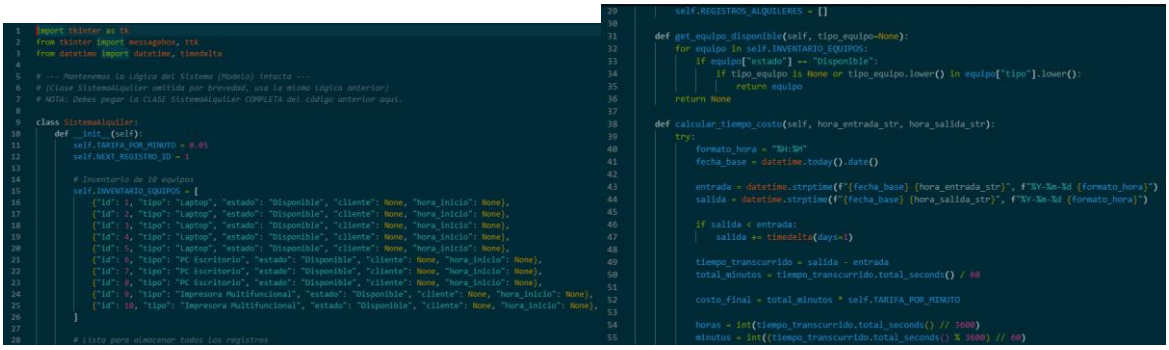
Tipos de replicación en MySQL:

1. **Replicación Maestro-Esclavo:**
    - Un servidor maestro maneja escrituras
    - Uno o más servidores esclavos reciben copias de los datos
    - Los esclavos pueden manejar lecturas
2. **Replicación Maestro-Maestro:**
    - Múltiples servidores pueden manejar escrituras
    - Los cambios se sincronizan entre todos los servidores

Ventajas de la replicación:

- **Alta disponibilidad:** Si un servidor falla, otros pueden continuar
- **Balance de carga:** Distribuir consultas entre múltiples servidores
- **Copia de seguridad:** Los esclavos pueden servir como backups
- **Análisis de datos:** Usar esclavos para reportes sin afectar producción

PARTE III



```
import tkinter as tk
from tkinter import messagebox, ttk
from datetime import datetime, timedelta

# --- Mantenemos la Lógica del Sistema (Modelo) intacta ---
# (Clase SistemaAlquiler omitida por brevedad, usa la misma lógica anterior)
# NOTA: Debes pegar la CLASE SistemaAlquiler COMPLETA del código anterior aquí.

class SistemaAlquiler:
    def __init__(self):
        self.TARIFA_POR_MINUTO = 0.05
        self.NEXT_REGISTRO_ID = 1

        # Inventario de 10 equipos
        self.INVENTARIO_EQUIPOS = [
            {"id": 1, "tipo": "Laptop", "estado": "Disponible", "cliente": None, "hora_inicio": None},
            {"id": 2, "tipo": "Laptop", "estado": "Disponible", "cliente": None, "hora_inicio": None},
            {"id": 3, "tipo": "Laptop", "estado": "Disponible", "cliente": None, "hora_inicio": None},
            {"id": 4, "tipo": "Laptop", "estado": "Disponible", "cliente": None, "hora_inicio": None},
            {"id": 5, "tipo": "Laptop", "estado": "Disponible", "cliente": None, "hora_inicio": None},
            {"id": 6, "tipo": "PC Escritorio", "estado": "Disponible", "cliente": None, "hora_inicio": None},
            {"id": 7, "tipo": "PC Escritorio", "estado": "Disponible", "cliente": None, "hora_inicio": None},
            {"id": 8, "tipo": "PC Escritorio", "estado": "Disponible", "cliente": None, "hora_inicio": None},
            {"id": 9, "tipo": "Impresora Multifuncional", "estado": "Disponible", "cliente": None, "hora_inicio": None},
            {"id": 10, "tipo": "Impresora Multifuncional", "estado": "Disponible", "cliente": None, "hora_inicio": None},
        ]

        # Lista para almacenar todos los registros
        self.REGISTROS_ALQUILERES = []

    def get_equipo_disponible(self, tipo_equipo=None):
        for equipo in self.INVENTARIO_EQUIPOS:
            if equipo["estado"] == "Disponible":
                if tipo_equipo is None or tipo_equipo.lower() in equipo["tipo"].lower():
                    return equipo
        return None

    def calcular_tiempo_costo(self, hora_entrada_str, hora_salida_str):
        try:
            formato_hora = "%H:%M"
            fecha_base = datetime.today().date()

            entrada = datetime.strptime(f"{fecha_base} {hora_entrada_str}", f"%Y-%m-%d {formato_hora}")
            salida = datetime.strptime(f"{fecha_base} {hora_salida_str}", f"%Y-%m-%d {formato_hora}")

            if salida < entrada:
                salida += timedelta(days=1)

            tiempo_transcurrido = salida - entrada
            total_minutos = tiempo_transcurrido.total_seconds() / 60

            costo_final = total_minutos * self.TARIFA_POR_MINUTO

            horas = int(tiempo_transcurrido.total_seconds() // 3600)
            minutos = int((tiempo_transcurrido.total_seconds() % 3600) // 60)

            tiempo_formato = f"{horas:01d}:{minutos:02d}:00"

            return tiempo_formato, round(costo_final, 2)
        except ValueError:
            return None, None

    def iniciar_alquiler(self, cliente, tipo_equipo, hora_entrada_str):
        equipo = self.get_equipo_disponible(tipo_equipo)

        if not equipo:
            return "ERROR: No hay equipos disponibles del tipo solicitado.", None

        equipo["estado"] = "Alquilado"
        equipo["cliente"] = cliente
        equipo["hora_inicio"] = hora_entrada_str

        registro = {
            "id_registro": self.NEXT_REGISTRO_ID,
            "id_equipo": equipo["id"],
            "tipo_equipo": equipo["tipo"],
            "cliente": cliente,
            "hora_entrada": hora_entrada_str,
            "hora_salida": None,
            "tiempo_total": None,
            "costo_final": None,
            "estado": "ACTIVO"
        }
        self.REGISTROS_ALQUILERES.append(registro)
        self.NEXT_REGISTRO_ID += 1

        mensaje = f"ÉXITO: Alquiler iniciado (Reg ID: {registro['id_registro']}). Equipo {equipo['id']} asignado."
        return mensaje, registro

    def finalizar_alquiler(self, registro_id, hora_salida_str):
        registro = next((r for r in self.REGISTROS_ALQUILERES if r["id_registro"] == registro_id and r["estado"] == "ACTIVO"), None)

        if not registro:
            return "ERROR: Registro activo no encontrado o ID inválido.", None

        tiempo, costo = self.calcular_tiempo_costo(registro["hora_entrada"], hora_salida_str)

        if tiempo is None:
            return "ERROR: Formato de hora de salida inválido (HH:MM).", None

        registro["hora_salida"] = hora_salida_str
        registro["tiempo_total"] = tiempo
        registro["costo_final"] = costo
        registro["estado"] = "FINALIZADO"

        equipo = next((e for e in self.INVENTARIO_EQUIPOS if e["id"] == registro["id_equipo"]), None)
        if equipo:
            equipo["estado"] = "Disponible"
            equipo["cliente"] = None
            equipo["hora_inicio"] = None

        mensaje = (
            f"ALQUILER FINALIZADO (Reg ID: {registro['id_registro']})\n"
            f"Tiempo: {tiempo}\n"
            f"Costo Total: ${costo:.2f}"
        )
        return mensaje, registro

# ----------------------------------------------------------

# --- 2. Interfaz Gráfica (Tkinter) con nuevo estilo ---

class AppGUI:
    # --- Definición de Estilos ---
    COLOR_FONDO = "#F0E6F7"      # Lavanda pálido
    COLOR_PRIMARIO = "#D8BFD8"   # Lila suave
    COLOR_SECUNDARIO = "#FFC0CB" # Rosa claro (Pink)
    COLOR_TITULO = "#800080"     # Púrpura
    FUENTE_TITULO = ('Georgia', 14, 'bold')
    FUENTE_NORMAL = ('Georgia', 11)

    def __init__(self, master, sistema):
        self.master = master
        self.sistema = sistema
        master.title("Gestión de Alquileres Tecnológicos")
        master.geometry("800x600")
        master.configure(bg=self.COLOR_FONDO)

        # Configurar estilo para ttk widgets (Notebook, Treeview)
        style = ttk.Style()
```

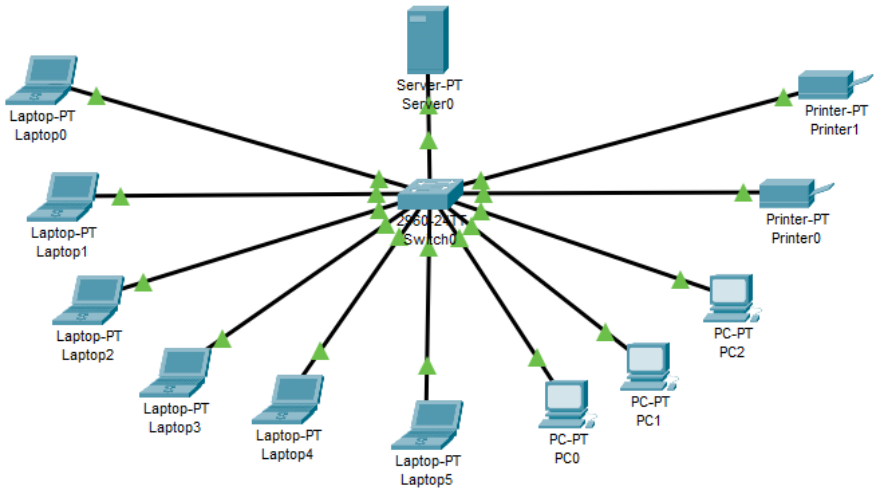
```
        style.theme_create('girly_theme', parent='alt', settings={
            "TNotebook": {"configure": {"tabmargins": [2, 5, 2, 0], "background": self.COLOR_FONDO}},
            "TNotebook.Tab": {"configure": {"padding": [10, 5], "font": self.FUENTE_NORMAL, "background": self.COLOR_PRIMARIO},
                              "map": {"background": [("selected", self.COLOR_SECUNDARIO)]}},
            "TButton": {"configure": {"font": self.FUENTE_NORMAL, "background": self.COLOR_PRIMARIO, "foreground": self.COLOR_TITULO}},
            "Treeview": {"configure": {"font": self.FUENTE_NORMAL, "background": "white", "foreground": "black", "rowheight": 25}},
            "Treeview.Heading": {"configure": {"font": self.FUENTE_NORMAL, "background": self.COLOR_PRIMARIO, "foreground": self.COLOR_TITULO}},
        })
        style.theme_use('girly_theme')

        # Configurar pestañas (Notebook)
        self.notebook = ttk.Notebook(master)
        self.notebook.pack(pady=10, padx=10, fill="both", expand=True)

        self.tab_alquilar = tk.Frame(self.notebook, bg=self.COLOR_FONDO)
        self.tab_finalizar = tk.Frame(self.notebook, bg=self.COLOR_FONDO)
        self.tab_inventario = tk.Frame(self.notebook, bg=self.COLOR_FONDO)

        self.notebook.add(self.tab_alquilar, text="📄 Iniciar Alquiler")
        self.notebook.add(self.tab_finalizar, text="✅ Finalizar Alquiler")
        self.notebook.add(self.tab_inventario, text="📦 Control de Activos")

        self._crear_tab_alquilar()
        self._crear_tab_finalizar()
        self._crear_tab_inventario()

        self.notebook.bind("<<NotebookTabChanged>>", self._actualizar_tabs)
```

```
    def _crear_tab_alquilar(self):
        """Diseña la interfaz para INICIAR un alquiler."""

        # Variables de control
        self.cliente_var = tk.StringVar()
        self.tipo_equipo_var = tk.StringVar(value="Laptop")
        self.hora_entrada_var = tk.StringVar(value=datetime.now().strftime("%H:%M"))

        tk.Label(self.tab_alquilar, text="📄 INICIAR NUEVO ALQUILER 📄", font=self.FUENTE_TITULO, bg=self.COLOR_FONDO, fg=self.COLOR_TITULO).pack(pady=10)

        # Cliente
        tk.Label(self.tab_alquilar, text="Nombre del Cliente:", font=self.FUENTE_NORMAL, bg=self.COLOR_FONDO).pack(pady=5)
        tk.Entry(self.tab_alquilar, textvariable=self.cliente_var, width=30, font=self.FUENTE_NORMAL, bg='white').pack(pady=5)

        # Tipo de Equipo
        tk.Label(self.tab_alquilar, text="Tipo de Equipo:", font=self.FUENTE_NORMAL, bg=self.COLOR_FONDO).pack(pady=5)
        tipos = ["Laptop", "PC Escritorio", "Impresora Multifuncional"]
        ttk.Combobox(self.tab_alquilar, textvariable=self.tipo_equipo_var, values=tipos, state="readonly", width=28, font=self.FUENTE_NORMAL).pack(pady=5)

        # Hora de Entrada
        tk.Label(self.tab_alquilar, text="Hora de Entrada (HH:MM):", font=self.FUENTE_NORMAL, bg=self.COLOR_FONDO).pack(pady=5)
        tk.Entry(self.tab_alquilar, textvariable=self.hora_entrada_var, width=10, font=self.FUENTE_NORMAL, bg='white').pack(pady=5)

        # Botón
        tk.Button(self.tab_alquilar, text="INICIAR ALQUILER (Insertar)", command=self._handle_iniciar_alquiler, bg=self.COLOR_SECUNDARIO, fg='black', font=self.FUENTE_N

    def _crear_tab_finalizar(self):
        """Diseña la interfaz para FINALIZAR un alquiler."""
```

```
        self.id_registro_var = tk.IntVar()
        self.hora_salida_var = tk.StringVar(value=datetime.now().strftime("%H:%M"))
        self.tiempo_var = tk.StringVar(value="0:00:00")
        self.costo_var = tk.StringVar(value="0.00")

        tk.Label(self.tab_finalizar, text="✅ FINALIZAR Y CALCULAR COSTO ✅", font=self.FUENTE_TITULO, bg=self.COLOR_FONDO, fg=self.COLOR_TITULO).pack(pady=10)

        # Treeview para mostrar registros activos
        tk.Label(self.tab_finalizar, text="Selecciona el Registro Activo:", font=self.FUENTE_NORMAL, bg=self.COLOR_FONDO).pack(pady=5)
        self.tree_activos = ttk.Treeview(self.tab_finalizar, columns=("ID", "Cliente", "Equipo", "H. Entrada"), show="headings", height=5)
        self.tree_activos.heading("ID", text="Reg. ID", anchor=tk.CENTER)
        self.tree_activos.heading("Cliente", text="Cliente", anchor=tk.W)
        self.tree_activos.heading("Equipo", text="Equipo", anchor=tk.W)
        self.tree_activos.heading("H. Entrada", text="Hora Entrada", anchor=tk.CENTER)

        self.tree_activos.column("ID", width=60, anchor=tk.CENTER)
        self.tree_activos.column("Cliente", width=150)
        self.tree_activos.column("Equipo", width=150)
        self.tree_activos.column("H. Entrada", width=100, anchor=tk.CENTER)

        self.tree_activos.pack(pady=5)
        self.tree_activos.bind("<<TreeviewSelect>>", self._select_registro)

        # Inputs para Finalizar
        input_frame = tk.Frame(self.tab_finalizar, bg=self.COLOR_FONDO)
        input_frame.pack(pady=10)

        tk.Label(input_frame, text="ID de Registro:", font=self.FUENTE_NORMAL, bg=self.COLOR_FONDO).grid(row=0, column=0, pady=5)
```

```
                        r["hora_salida"] if r["hora_salida"] else "N/A",
                        r["tiempo_total"] if r["tiempo_total"] else "N/A",
                        f"${r['costo_final']:.2f}" if r["costo_final"] is not None else "N/A",
                        r["estado"]
                    ), tags=(tag,))

        # Colores para el historial
        tree.tag_configure('activo_reg', background='#FFFACD', foreground='black') # Amarillo pálido para activo
        tree.tag_configure('finalizado', background='#E0FFFF', foreground='black') # Cian pálido para finalizado

        tree.pack(pady=10, padx=10, fill="both", expand=True)

# --- 4. Inicialización de la Aplicación ---

if __name__ == "__main__":
    root = tk.Tk()
    sistema = SistemaAlquiler()
    gui = AppGUI(root, sistema)
    root.mainloop()
```

PARTE IV


Server-PT
Server0
Laptop-PT
Laptop0
Printer-PT
Printer1
Laptop-PT
Laptop1
2960-24TT
Switch0
Printer-PT
Printer0
Laptop-PT
Laptop2
PC-PT
PC2
Laptop-PT
Laptop3
Laptop-PT
Laptop4
PC-PT
PC1
Laptop-PT
Laptop5
PC-PT
PC0