Procedimientos de Búsqueda y Ordenamiento de Arreglos con MySQL

Este trabajo de investigación examina los procedimientos fundamentales de búsqueda y ordenamiento de arreglos, dos pilares esenciales de la informática moderna. Exploraremos la búsqueda secuencial como método introductorio, el algoritmo Push Down (Heapify) para ordenamiento eficiente, y su integración práctica con bases de datos MySQL. A través de pseudocódigo, implementaciones en Python y simulaciones paso a paso, comprenderemos cómo estos algoritmos transforman datos en información estructurada y accesible.

I PARTE. Investigación. ***Valor 15 puntos***

Temas:

Procedimiento de búsqueda y ordenamiento de un arreglo

Estudiantes: Jafet Loo (4-838-349)
Ihammal Medina (4-835-706)

Búsqueda Secuencial: Concepto Fundamental

La búsqueda secuencial, también conocida como búsqueda lineal, representa uno de los algoritmos más fundamentales en la informática. Este método examina cada elemento de un arreglo de manera ordenada, comenzando desde la primera posición, comparando sistemáticamente el valor buscado con cada elemento hasta encontrarlo o llegar al final de la estructura. Aunque su simplicidad es su mayor virtud, también es su limitación más evidente.

Su principal fortaleza radica en la flexibilidad: no requiere que el arreglo esté previamente ordenado, lo que lo hace ideal para estructuras de datos desorganizadas. La implementación es directa y comprensible, facilitando su enseñanza y mantenimiento. Sin embargo, la complejidad temporal de O(n) hace que sea ineficiente para arreglos de gran tamaño, donde su desempeño se degrada proporcionalmente con la cantidad de elementos.

Ventajas

- Fácil de implementar
- No requiere datos ordenados
- Memoria mínima

Desventajas

- Lento en arreglos grandes
- Complejidad O(n)
- Búsqueda exhaustiva

Aplicaciones

- Arreglos pequeños
- Datos no ordenados
- Búsquedas puntuales

Implementación de Búsqueda Secuencial

Pseudocódigo:

```
PROCEDIMIENTO BUSQUEDA_SECUENCIAL(A, n, clave)
    PARA i DESDE 0 HASTA n - 1 HACER
        SI A[i] = clave ENTONCES
            RETORNAR i
        FIN SI
    FIN PARA
    RETORNAR -1  // No encontrado
FIN PROCEDIMIENTO
```

Implementación en Python:

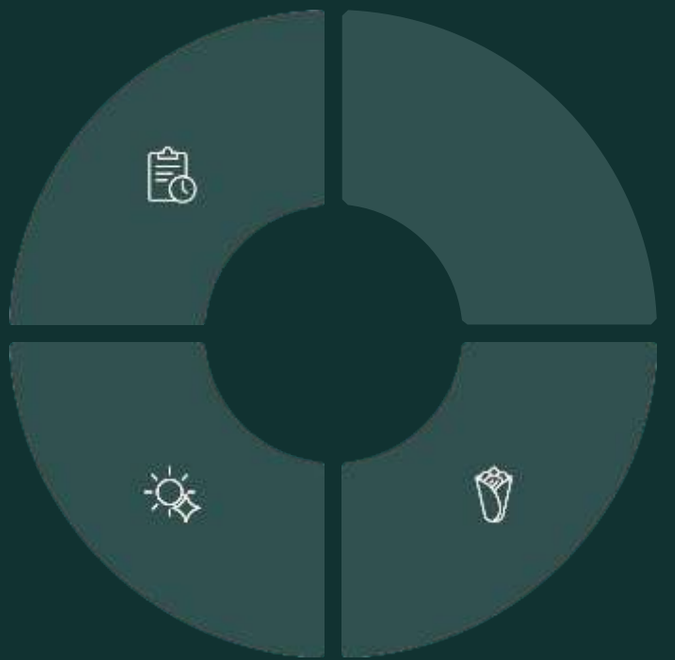
```
def busqueda_secuencial(A, clave):
    for i in range(len(A)):
        if A[i] == clave:
            return i
    return -1

# Ejemplo de uso
arreglo = [5, 3, 8, 1, 9]
clave = 1
resultado = busqueda_secuencial(arreglo, clave)
print("Elemento encontrado en la posición:", resultado)
```

El ejemplo demuestra cómo la función itera a través del arreglo [5, 3, 8, 1, 9] buscando el valor 1. Tras recorrer cuatro iteraciones, encuentra el elemento en la posición 3 (considerando indexación desde cero). La salida esperada confirma este resultado. Este algoritmo es particularmente valioso en situaciones donde los datos no están organizados previamente y la simplicidad del código es más importante que la eficiencia computacional.

Push Down (Heapify): Ordenamiento Avanzado

El método Push Down, también denominado Sift Down o Heapify, constituye el corazón del algoritmo Heap Sort. Este procedimiento mantiene la propiedad fundamental de un heap: una estructura de árbol binario donde cada nodo padre es mayor (en un max-heap) o menor (en un min-heap) que sus hijos. El proceso de Push Down desciende un elemento dentro de la estructura jerárquica hasta posicionarlo correctamente, garantizando que la propiedad del heap se preserve.

La complejidad temporal de O(n log n) lo convierte en una opción superior a algoritmos simples como ordenamiento de burbuja para conjuntos de datos grandes. Aunque requiere una comprensión más profunda de estructuras de árbol y manipulación de índices, su eficiencia justifica la inversión educativa. El algoritmo no necesita memoria adicional significativa, lo que lo hace atractivo para ambientes con restricciones de recursos.



Complejidad

O(n log n)

Memoria

O(1) adicional

Optimización

Muy eficiente

Estabilidad

No estable

Código y Simulación de Push Down

Implementación en Python:

```
def sift_down(A, start, end):
    root = start
    while root * 2 + 1 <= end:
        child = root * 2 + 1
        swap_idx = root
        if A[swap_idx] < A[child]:
            swap_idx = child
        if child + 1 <= end and A[swap_idx] < A[child + 1]:
            swap_idx = child + 1
        if swap_idx == root:
            return
        A[root], A[swap_idx] = A[swap_idx], A[root]
        root = swap_idx

def heapify(A):
    n = len(A)
    start = (n - 2) // 2
    while start >= 0:
        sift_down(A, start, n - 1)
        start -= 1

def heap_sort(A):
    n = len(A)
    heapify(A)
    end = n - 1
    while end > 0:
        A[0], A[end] = A[end], A[0]
        end -= 1
        sift_down(A, 0, end)

# Ejemplo
arr = [5, 3, 8, 1, 9, 2]
heap_sort(arr)
print("Arreglo ordenado:", arr)
```

La salida esperada produce: [1, 2, 3, 5, 8, 9]. Este código implementa el proceso completo: primero convierte el arreglo en un heap (heapify), luego extrae repetidamente el elemento máximo, posicionándolo al final. El Push Down garantiza que tras cada intercambio, la estructura de heap se mantenga válida, permitiendo un ordenamiento eficiente.

MySQL: Sistema de Gestión de Bases de Datos

MySQL es un Sistema de Gestión de Bases de Datos Relacional (RDBMS) de código abierto que utiliza Structured Query Language (SQL) para manipular información. Su arquitectura cliente-servidor permite que múltiples usuarios accedan simultáneamente a los datos, ofreciendo confiabilidad, seguridad y eficiencia. Desde su creación, MySQL se ha convertido en el estándar de facto para aplicaciones web, siendo especialmente popular en stacks tecnológicos como LAMP (Linux, Apache, MySQL, PHP).

La naturaleza relacional de MySQL significa que los datos se organizan en tablas con filas y columnas, donde las relaciones entre tablas se establecen mediante claves foráneas. Esta estructura garantiza la integridad referencial y facilita consultas complejas. Su compatibilidad con múltiples lenguajes de programación, incluyendo Python, Java y PHP, lo hace versátil para diversos proyectos. La capacidad de replicación y su robusto sistema de transacciones lo hacen confiable para aplicaciones empresariales.

Requisito	Descripción	Mínimo
S.O.	Windows, Linux, macOS	Cualquiera
Disco	Espacio para datos	500 MB
RAM	Memoria operativa	1 GB
Puerto	Conexión por defecto	3306

Conexión Python-MySQL

Instalación del Conector:

```
pip   install   mysql-connector-python
```

Código de Conexión Básica:

```
import mysql.connector

# Conexión a la base de datos
conexion  =  mysql.connector.connect(
   host="localhost",
   user="root",
   password="1234",
   database="ejemplo_db"
)

cursor = conexion.cursor()

# Crear una tabla
cursor.execute("""
   CREATE TABLE IF NOT EXISTS usuarios (
      id INT AUTO_INCREMENT PRIMARY KEY,
      nombre VARCHAR(50),
      edad INT
   )
""")

# Insertar datos
cursor.execute("INSERT INTO usuarios (nombre, edad) VALUES ('Ana', 22)")
conexion.commit()

# Consultar datos
cursor.execute("SELECT * FROM usuarios")
for fila in cursor.fetchall():
   print(fila)

cursor.close()
conexion.close()
```

Este código establece una conexión con un servidor MySQL local, crea una tabla de usuarios, inserta un registro y lo recupera. La salida esperada es: (1, 'Ana', 22). La integración entre Python y MySQL permite automatizar operaciones de base de datos, ideal para aplicaciones que requieren persistencia de datos y consultas dinámicas.

Sintaxis SQL y Operaciones Fundamentales

SQL (Structured Query Language) es el lenguaje estándar para interactuar con bases de datos relacionales. Sus comandos se dividen en categorías: DDL (Data Definition Language) para crear estructuras, DML (Data Manipulation Language) para manipular datos, y DCL (Data Control Language) para gestionar permisos.

```
-- Crear base de datos
CREATE DATABASE ejemplo_db;
USE ejemplo_db;

-- Crear tabla
CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50),
    edad INT
);

-- Insertar datos
INSERT INTO usuarios (nombre, edad) VALUES ('Ana', 22);
INSERT INTO usuarios (nombre, edad) VALUES ('Carlos', 25);

-- Consultar datos
SELECT * FROM usuarios;
SELECT nombre FROM usuarios WHERE edad > 23;

-- Actualizar datos
UPDATE usuarios SET edad = 23 WHERE nombre = 'Ana';

-- Eliminar datos
DELETE FROM usuarios WHERE id = 1;
```

1 **CREATE**

Establece nuevas tablas y bases de datos con esquemas definidos

2 **INSERT**

Agrega registros nuevos a las tablas existentes

3 **SELECT**

Recupera datos mediante consultas flexibles

4 **UPDATE**

Modifica registros existentes

Replicación de Bases de Datos

La replicación de bases de datos constituye un mecanismo crítico para garantizar alta disponibilidad, confiabilidad y distribución eficiente de cargas de trabajo. Este proceso copia datos de una base principal (maestro) hacia una o varias bases secundarias (réplicas), manteniéndolas sincronizadas. En ambientes empresariales, la replicación es fundamental para minimizar tiempos de inactividad y proporcionar redundancia ante fallos del sistema principal.

Existen dos paradigmas principales: replicación síncrona, donde la réplica se actualiza simultáneamente con cambios en el maestro, garantizando consistencia absoluta pero con latencia aumentada; y replicación asíncrona, donde existe un pequeño retraso pero con mejor rendimiento. Las ventajas incluyen copias de seguridad en tiempo real sin detener operaciones, balanceo de carga distribuido entre múltiples servidores, y recuperación rápida ante desastres. Las desventajas abarcan mayor complejidad de configuración y posibles inconsistencias temporales en replicación asíncrona.

1 Replicación Síncrona

- Mayor consistencia
- Actualización simultánea
- Mayor latencia
- Más recursos

2 Replicación Asíncrona

- Mejor velocidad
- Pequeño retraso
- Menor latencia
- Menos recursos

Alta Disponibilidad
Continuidad operativa

Balanceo de Carga
Distribución eficiente

Seguridad de Datos
Respaldos actualizados

Conclusiones e Integración de Conceptos

Este trabajo ha explorado tres pilares fundamentales de la informática moderna: búsqueda, ordenamiento y gestión de datos. La búsqueda secuencial proporciona la base conceptual sobre algoritmos de búsqueda simples, ideal para introducir conceptos de complejidad temporal. El método Push Down, aunque más complejo, demuestra cómo estructuras avanzadas como heaps permiten algoritmos de ordenamiento altamente eficientes con complejidad O(n log n), superando significativamente a métodos elementales.

MySQL materializa estos conceptos teóricos en sistemas reales, permitiendo almacenar, consultar y manipular millones de registros de manera estructurada y confiable. La integración con Python demuestra cómo los algoritmos y bases de datos se combinan en aplicaciones prácticas. La replicación subraya la importancia de la confiabilidad y disponibilidad en sistemas empresariales modernos, donde la continuidad operativa es crítica.

01

Conceptos Algorítmicos

Búsqueda secuencial y Push Down establecen fundamentos de complejidad temporal

02

Implementación Práctica

Python proporciona sintaxis clara para traducir algoritmos a código ejecutable

03

Persistencia de Datos

MySQL gestiona datos estructurados con ACID y escalabilidad empresarial

04

Confiabilidad Operativa

Replicación asegura continuidad y recuperación ante fallos

05

Aplicación Integral

La combinación de estos elementos crea sistemas robustos y eficientes

Referencias Académicas:

- Documentación oficial de Python: https://docs.python.org
- Documentación de MySQL: https://dev.mysql.com/doc
- Tutorial de Heap Sort: https://www.geeksforgeeks.org/heap-sort/
- Conector Python-MySQL: https://pypi.org/project/mysql-connector-python/