TRABAJO PRÁCTICO: ALGORITMOS DE BÚSQUEDA Y ESTRUCTURAS HASH

Materia: Estructuras de Datos

Objetivo: Resolver problemas algorítmicos complejos utilizando punteros, gestión dinámica de memoria, Búsqueda Binaria y Tablas Hash.

PARTE I: BÚSQUEDA BINARIA Y VARIACIONES

Instrucciones: Para los siguientes ejercicios, se debe implementar una solución eficiente priorizando una complejidad temporal logarítmica (O(log n)) o cercana a ella, utilizando punteros para el manejo de arreglos y matrices.

Ejercicio 1: Búsqueda en Matriz Rotada

Descripción del Problema:

Se dispone de una matriz de dimensiones m*n. Originalmente, esta matriz tenía la propiedad de que cada fila estaba ordenada de izquierda a derecha y cada columna de arriba a abajo. Sin embargo, la matriz ha sufrido rotaciones que han alterado parcialmente este orden.

Tarea:

Desarrolle un algoritmo que encuentre si existe un número objetivo dentro de dicha matriz.

- **Requisito:** La búsqueda debe realizarse de forma eficiente.
- **Complejidad esperada:** O(log(m*n)) en el mejor caso o una aproximación eficiente que evite el recorrido completo (O(m*n)) si es posible aprovechar sub-estructuras ordenadas.

Ejercicio 2: Búsqueda con Frecuencia Específica

Descripción del Problema:

Dado un arreglo ordenado de números enteros que puede contener elementos repetidos, se busca identificar un elemento basándose en su frecuencia de aparición.

Tarea:

Implemente una función que encuentre la primera posición de un elemento cuya frecuencia en el arreglo sea exactamente k.

- Si el elemento con frecuencia k no existe, retornar -1.
- El algoritmo debe utilizar una modificación de la búsqueda binaria para encontrar los límites (izquierdo y derecho) del elemento y calcular su frecuencia sin recorrer el arreglo linealmente.

Ejercicio 3: Encontrar un "Pico" en un Arreglo

Descripción del Problema:

Un "pico" en un arreglo se define como un elemento que es estrictamente mayor que sus vecinos adyacentes (el anterior y el siguiente). El arreglo no está necesariamente ordenado y puede contener múltiples picos.

Tarea:

Escriba un algoritmo que encuentre el índice de cualquier pico existente en el arreglo.

- **Condición:** Se garantiza que siempre existe al menos un pico.
- **Restricción de Complejidad:** El algoritmo debe funcionar en tiempo O(log n), utilizando la comparación de vecinos para decidir hacia qué mitad del arreglo desplazarse.

Ejercicio 4: Búsqueda en Arreglo con Cambios de Sentido

Descripción del Problema:

Se tiene un arreglo de números que sigue un patrón de ordenamiento mixto: los números están ordenados de forma creciente, pero en ciertos puntos el orden cambia a decreciente y viceversa (oscilaciones), manteniendo una estructura monótona por tramos.

Tarea:

Implemente una búsqueda para encontrar un elemento objetivo navegando correctamente a través de estos cambios de sentido.

- **Estrategia:** El algoritmo debe determinar la monotonía local (creciente o decreciente) en cada paso para decidir en qué sub-rango continuar la búsqueda, manteniendo una eficiencia superior a la búsqueda lineal.

PARTE II: TABLAS HASH

Instrucciones: Implementar una estructura de Tabla Hash desde cero (sin utilizar librerías de contenedores estándar como std::map o std::unordered_map) manejando colisiones explícitamente.

Ejercicio 5: Análisis de Frecuencia de Palabras

Descripción del Problema:

Se tiene un texto fuente que contiene palabras separadas por espacios y diversos signos de puntuación (comas, puntos, etc.). Se requiere realizar un análisis estadístico de la frecuencia de aparición de cada palabra.

Tarea:

Diseñe e implemente una clase TablaHash que cumpla con los siguientes requerimientos:

1. **Estructura:** Utilizar un arreglo de punteros a nodos para manejar colisiones mediante **encadenamiento** (listas enlazadas).
2. **Procesamiento:** Tokenizar el texto de entrada ignorando la puntuación y normalizando las palabras (convertir todo a minúsculas) para que "Palabra" y "palabra" cuenten como la misma.
3. **Operaciones:**
    - Insertar palabras nuevas con frecuencia 1.
    - Actualizar la frecuencia de palabras ya existentes.
    - Buscar la frecuencia de una palabra específica.
4. **Reporte:** Al finalizar, el programa debe mostrar estadísticas de la tabla (factor de carga, colisiones detectadas) y listar todas las palabras con sus respectivas frecuencias.

Complejidad Promedio Esperada: O(n) para inserción y búsqueda, donde n es la cantidad de palabras únicas.