Tecnológico de Monterrey

Act 6.2 - Reflexión Final de Actividades Integradoras de la Unidad de Formación TC1031 (Evidencia Competencia)

Mauricio Zavala Sánchez - A00837332

2 de diciembre del 2023

David Alonso Cantú Delgado

Programación de estructuras de datos y algoritmos fundamentales (Gpo 608)

1. Templates:
   - Importancia: Los templates en C + + son fundamentales para la programación genérica. Permiten escribir algoritmos y estructuras de datos que funcionan con diferentes tipos de datos sin perder eficiencia.
   - Eficiencia: Son eficientes en términos de reutilización de código y flexibilidad. Sin embargo, pueden resultar en código más extenso debido a la generación de instancias para cada tipo.

2. Recursión:
   - Importancia: La recursión simplifica el diseño de algoritmos y estructuras de datos, facilitando la resolución de problemas dividiéndolos en subproblemas más pequeños.
   - Eficiencia: Puede ser menos eficiente en términos de uso de memoria y velocidad de ejecución para ciertos problemas. La optimización de la recursión de cola mediante la eliminación de la recursividad final puede mejorar la eficiencia.

3. Search Algorithms (Algoritmos de Búsqueda):
   - Importancia: Algoritmos de búsqueda como binary search y linear search son cruciales para la recuperación eficiente de datos en conjuntos ordenados o no ordenados.
   - Eficiencia: La eficiencia varía según el contexto. Binary search es eficiente en conjuntos ordenados, mientras que linear search es apropiado para conjuntos no ordenados. Las mejoras pueden incluir el uso de búsqueda binaria en árboles de búsqueda binaria (BST).

4. Sort Algorithms (Algoritmos de Ordenamiento):
   - Importancia: Los algoritmos de ordenamiento, como quicksort y mergesort, son esenciales para organizar datos y mejorar el rendimiento en operaciones de búsqueda.
   - Eficiencia: La eficiencia depende del contexto y del tamaño de los datos. Quicksort es eficiente en la práctica debido a su implementación in-place y su rendimiento promedio rápido.

5. Linked List (Lista Enlazada):
   - Importancia: Las listas enlazadas son fundamentales para la gestión dinámica de datos y operaciones de inserción y eliminación eficientes.
   - Eficiencia: Eficientes para inserciones y eliminaciones, pero el acceso aleatorio es lineal. Las mejoras pueden incluir el uso de listas doblemente enlazadas para permitir un acceso más rápido hacia atrás.

6. Doubly Linked List (Lista Doblemente Enlazada):
   - Importancia: Similar a la lista enlazada, pero con la ventaja de acceso bidireccional, lo que mejora la eficiencia en ciertas operaciones.
   - Eficiencia: Mejora la eficiencia en operaciones que requieren acceso hacia atrás. Sin embargo, aumenta el consumo de memoria.

7. Queue:
   - Importancia: Las colas son cruciales para la implementación de estructuras como BFS y la gestión de tareas en un orden específico.
   - Eficiencia: Eficientes para operaciones FIFO. Mejoras podrían incluir implementaciones concurrentes para situaciones donde múltiples hilos acceden a la cola.

8. Stack:
   - Importancia: Las pilas son esenciales para implementar estructuras de datos como la evaluación de expresiones matemáticas y seguimiento de llamadas a funciones.
   - Eficiencia: Eficientes para operaciones LIFO. No necesitan operaciones de búsqueda. Mejoras podrían incluir la gestión eficiente de memoria en operaciones push/pop.

9. BST (Binary Search Tree):
   - Importancia: Árboles de búsqueda binaria son utilizados en la implementación de diccionarios y conjuntos ordenados.
   - Eficiencia: Eficientes para búsqueda, inserción y eliminación en promedio. Pueden ser ineficientes en casos degenerados. Mejoras incluirían equilibrio automático en árboles AVL o árboles rojo-negro.

10. AVL (Árbol AVL):
    - Importancia: Árboles AVL mantienen su equilibrio, mejorando la eficiencia en comparación con los árboles de búsqueda binaria estándar.
    - Eficiencia: Garantiza tiempos de búsqueda, inserción y eliminación logarítmicos. Requiere más operaciones de mantenimiento, pero las mejoras en la eficiencia son significativas.

11. Heap:
    - Importancia: Los montículos son cruciales para implementar colas de prioridad, necesarias para algoritmos como Dijkstra y heapsort.
    - Eficiencia: Operaciones de inserción y extracción eficientes en tiempo logarítmico. Mejoras podrían incluir la implementación de Fibonacci Heaps para ciertos algoritmos que requieren disminución de clave eficiente.

12. Graph (Grafo):
    - Importancia: Representa relaciones y conexiones entre entidades en problemas del mundo real.
    - Eficiencia: La eficiencia depende de la representación del grafo. Listas de adyacencia son eficientes para grafos dispersos, matrices de adyacencia para grafos densos.

13. Graph Traversal (Recorrido de Grafos):
    - Importancia: Algoritmos como BFS y DFS son fundamentales para explorar y analizar la estructura de los grafos.

- Eficiencia: Eficiente para diferentes escenarios. Las mejoras pueden incluir la optimización del espacio de almacenamiento en BFS mediante el uso de bit sets.

14. Dijkstra:
- Importancia: Encuentra el camino más corto en grafos ponderados, aplicaciones en redes y sistemas de navegación.
- Eficiencia: Eficiente con montículos de prioridad. Mejoras pueden incluir la implementación de algoritmos de búsqueda de caminos más cortos en paralelo para mejorar el rendimiento.

15. Hashtable:
- Importancia: Estructura eficiente para búsqueda, inserción y eliminación en tiempo constante en promedio.
- Eficiencia: La eficiencia depende de la función de dispersión y la resolución de colisiones. Las mejoras pueden incluir el uso de funciones de dispersión más avanzadas y estrategias de resolución de colisiones eficientes.

En general, la elección de la estructura de datos y el algoritmo adecuado depende del contexto y de los requisitos específicos del problema. Mejorar la eficiencia a menudo implica optimizar el uso de memoria, elegir algoritmos adecuados para el tamaño de los datos y, en algunos casos, implementar estructuras más avanzadas que se adapten mejor a situaciones específicas.