Universidad Nacional
de Entre Ríos

Facultad de ingeniería

Tecnicatura en Procesamiento y Explotación de Datos

Algoritmos y Estructura de Datos

Docentes:

Dr. Javier Eduardo Diaz Zamboni
Bioing. Jordán F. Insfrán
Bioing. Diana Vertiz del Valle
Bruno M. Breggia

Actividad: Informe de análisis TP 1.

Alumnos: Starchevich Adriel , Escalante Axel

18 Septiembre 2023

Introducción.

En el siguiente informe se detallarán las implementaciones TAD de Lista Doble Enlazada, con la estructura de datos Nodos enlazados, y funcionalidades básicas. Además de recrear el JuegoGuerra simulando el mazo con esta Lista Doble Enlazada. Por último se explicará el funcionamiento del ordenamiento externo.

Desarrollo

Implementación Lista Doble Enlazada

Para la primera consigna hemos implementado un TAD Lista Doble Enlazada utilizando Estructuras de Datos de nodos doblemente enlazados.

Implementación de la Clase Nodo

Para lograr nuestra implementación, creamos una clase llamada "Nodo" que toma como atributo el dato que se desea almacenar. Cada instancia de la clase Nodo tiene dos enlaces de inicialización que apuntan al nodo anterior y al nodo siguiente, y estos valores se establecen inicialmente como "None".

Operaciones de la Lista Doblemente Enlazada

La implementación de nuestra Lista Doblemente Enlazada incluye las operaciones básicas necesarias, como la obtención del tamaño de la lista, la extracción de elementos, la inserción de nuevos elementos al inicio y al final, entre otras.

Elección del Algoritmo de Ordenamiento

Optamos por implementar el algoritmo de ordenamiento rápido (Quicksort) para ordenar los nodos de la lista doblemente enlazada. Esta elección se basó en su eficiencia, ya que tiene una complejidad de O(n log n) en la mayoría de los casos. Además, Quicksort es uno de los algoritmos más rápidos y eficientes para la ordenación de datos, lo que lo hace adecuado para nuestras necesidades.

En términos simples, la figura 1 denota como el tiempo de ejecución del algoritmo aumenta de manera logarítmica con respecto al tamaño de los datos de entrada (n).

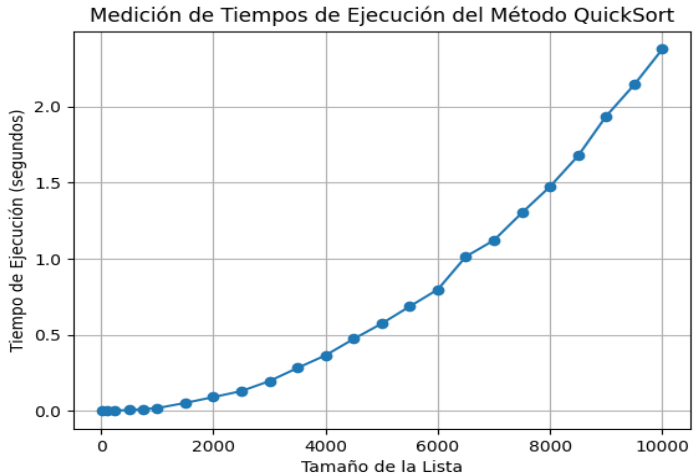

Figura 1. Análisis de medición de tiempo(s) para el ordenamiento QuickSort

Implementación del Juego de Cartas "Guerra"

Para la segunda consigna de nuestro trabajo práctico, recreamos juego de cartas "Guerra". En esta implementación, utilizamos una estructura de lista doblemente enlazada, como se mencionó anteriormente, para el mazo de cartas.

Clase Mazo y métodos adaptados

Creamos una clase llamada "Mazo" para representar el mazo de cartas del juego. Los métodos básicos que ya habíamos implementado en nuestra Lista Doble Enlazada, como extraer, agregar al inicio y agregar al final, se adaptaron al juego "Guerra". Ahora se llaman "sacar arriba," "poner arriba," y "poner abajo," respectivamente. Estos métodos se usan para manipular las cartas en el mazo durante el juego.

Clase JuegoGuerra y sus Funcionalidades

Además de la clase "Mazo", creamos una clase llamada "JuegoGuerra" que contiene los métodos y funciones necesarias para jugar al juego de "Guerra".

Las operaciones que implementamos en esta clase incluye:

Repartir las Cartas: Implementamos un método para repartir las cartas entre dos jugadores, dividiendo el mazo en partes iguales.

Comparación de Valores de Cartas: Creamos métodos para comparar los valores de las cartas jugadas por cada jugador y determinar el ganador de una ronda.

Iniciar una Guerra: En caso de que los valores de las cartas sean iguales, implementamos un mecanismo para iniciar una "guerra," donde se juegan múltiples cartas boca abajo antes de comparar nuevamente los valores. El ganador de la guerra se lleva todas las cartas en juego.

Implementación del Ordenamiento Externo

La finalidad de esta consigna es abordar la problemática de ordenar archivos de datos extremadamente grandes, los cuales no pueden ser procesados en su totalidad en la memoria principal debido a sus dimensiones. Para resolver este desafío, hemos implementado un algoritmo de ordenamiento externo que divide el archivo original en bloques de un tamaño especificado por el usuario. A continuación, detallamos el proceso implementado:

División en Bloques y Ordenamiento

- División en Bloques: El primer paso consiste en dividir el archivo original en bloques de un tamaño predeterminado, el cual es ingresado por el usuario. Esta división se realiza de manera secuencial y pasarán a dos archivos auxiliares.

- Ordenamiento Interno: Cada bloque creado en el paso anterior es ordenado internamente, utilizando el algoritmo de ordenamiento sorted. El objetivo es garantizar que los datos dentro de cada bloque estén ordenados de menor a mayor.

Fusión de Bloques y Repetición del Proceso

- Fusión de Bloques: Una vez que los bloques están ordenados, se procede a fusionarlos(Mezcla directa) y pasarlos al archivo original.
- Repetición de la Partición: Se repite el proceso de partición, pero esta vez con un tamaño de bloque multiplicado por 2. Es decir, los bloques resultantes serán el doble de grandes que los anteriores. Este proceso de partición y fusión se repite iterativamente hasta que el archivo completo quede completamente ordenado.

Verificación de Ordenamiento:
Para verificar que el ordenamiento se implementó una función que toma el archivo completo ordenado, luego de aplicarle un sort se comprueba si el archivo es igual al ordenado, si es así la función retorna un “True” de lo contrario un “False”.

Conclusión:
Implementamos un Tipo Abstracto de Datos (TAD) de Lista Doble Enlazada utilizando nodos, recreamos el juego de cartas "Guerra" y desarrollamos un algoritmo de ordenamiento externo para manejar archivos de datos masivos. En cada caso, logramos implementaciones que demuestran la utilidad de estructuras de datos y algoritmos en situaciones prácticas. Estas implementaciones destacan la importancia de la planificación y el diseño para resolver problemas de manera eficiente.