Tecnicatura en Procesamiento y Explotación de Datos Algoritmos y Estructura de Datos

Actividad: Trabajo práctico integrador 1

Título: Informe de análisis problema 1

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

Alumno:
Joaquin Frattin

Introducción.

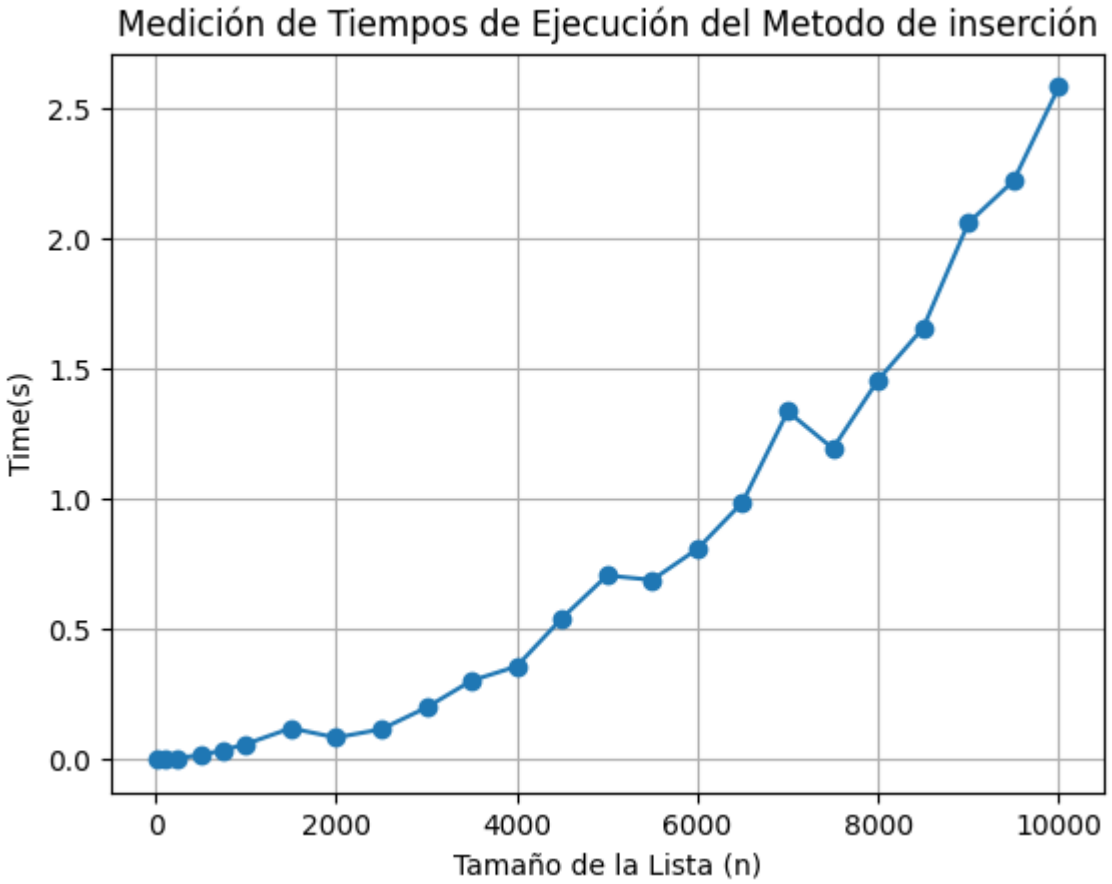
El objetivo de este informe es realizar la implementación de una Lista Doble Enlazada, la cual está conformada por la estructura de datos Nodos Enlazados. Además dicha estructura se utilizó para la implementación de un juego de guerra.
Finalizando se realizó un reordenamiento de un archivo de texto eligiendo el método de ordenamiento Quick sort

Problema 1: 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 objeto nodo posee una señalización en la cual brinda información sobre el nodo siguiente, y el nodo anterior.
Luego de definir la clase nodo, se definió una nueva clase “Lista Doble Enlazada” la cual contenía de atributos dos Nodos , y una enteró tamaño, los nodos apuntaban al objeto inicial y objeto final de la lista, y el tamaño representaba la cantidad de elementos enlazados que poseía la lista. Además se implementaron métodos de operaciones básicas entre ellas str, iterar, y adicionar, menor, también se implementaron métodos como la de retornar los valores atributos, el constructor , insertar un nodo al inicio, insertar un nodo al final, extraer un nodo al inicio, copiar, invertir lista, ordenar lista, entre otros.

Optamos por utilizar el método de ordenamiento de inserción, ya que se pedía un orden de complejidad igual o superior a este optamos por ello utilizamos el mismo método debido a su simplicidad de implementación. Sabíamos que este método podría ser lento para listas de gran tamaño, pero dado que la cantidad de datos que íbamos a manipular no era muy grande, consideramos que era una elección adecuada en función de la simplicidad y la claridad del algoritmo.

La figura a continuación denota como el tiempo de ejecución del algoritmo aumenta de manera cuadrática con respecto al tamaño de los datos de entrada (n).



Problema 2: Implementación Lista Doble Enlazada

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 Carta y métodos adaptados:

Creamos una clase llamada "Carta" para representar Cartas del juego. Es un objeto el cual posee un diccionario para poder adaptar los valores de la baraja francesa a valores numéricos. Esta clase posee 3 atributos, valor , palo , y estado, valor y palo son valores y los palos que tiene un mazo de cartas, y estado es un booleano el cual es true si la carta se muestra boca arriba y false si está boca abajo. Luego se implementaron métodos de igualación, comparación, voltear, métodos para devolver atributos,

Clase Mazo y métodos adaptados

Creamos una clase llamada "Mazo" para representar el mazo de cartas del juego. Esta clase puede representarse como una lista doblemente enlazada, por lo cual de atributo posee una lista doble enlazada los cuales los datos de los nodos son objetos de la clase cartas. Se implementaron métodos de sacar carta arriba, poner carta arriba, poner carta abajo, mazo está vacío, invertir todas las cartas, y métodos esenciales como de iteración, construcción, str.

Clase JuegoGuerra y sus Funcionalidades

La clase"JuegoGuerra" que contiene los métodos y funciones necesarias para el desarrollo del juego, posee como atributos mazo mesa que son son las cartas a repartir, mazo jugador 1, mazo jugador 2, que son los mazos de ambos jugadores, un objeto semilla que es un valor para darle aleatoriedad al juego el cual modifica el orden de reparto del mazo en los mazos de los jugadores, un diccionario llamado resultado que imprime el resultado del juego, y un objeto entero llamado turnos que cuenta la cantidad de turnos.
Las operaciones que implementamos en esta clase incluye:
Armar mazo: este método crea el mazo de juego, el cual crea objetos tipo carta con palo y valor y lo introduce a un objeto mazo.
Repartir: este método divide este mazo de cartas en dos mazos inferiores que representan los mazos de los jugadores.
iniciarjuego: Este método inicia el juego y compara las cartas del tablero y ejecuta las reglas del mismo. En el caso de que ambas cartas sean iguales se van a llamar el método Jugargurrar.
Jugar Guerra: este método es una función recursiva la cual ejecuta las reglas del evento de “guerra” del juego el cual posee otras reglas y mecanismos para su ejecución. Se consideró que esta función se recursiva ya que si la guerra entre dos cartas vuelven a ser iguales se desencadena nuevamente un evento de “guerra”, por lo que se vuelven a usar las reglas y mecanismos del método mismo.

Problema 3 Implementación del Ordenamiento Externo

Este código en Python contiene funcioneslas llama para crear y ordenar un archivo de datos. A continuación, un resumen de cada función:
crear_archivo_de_datos(nombre, mb`: Esta función se encarga de crear un archivo de datos con un tamaño especificado en megabytes (mb). La función toma dos argumentos: `nombre`, que es el nombre del archivo a crear, y `mb`, que es el tamaño en megabytes. La función genera números enteros aleatorios de 20 dígitos y los escribe en el archivo en bloques de 100,000 valores hasta que se alcance el tamaño deseado en megabytes.

merge_sort_externo(archivo_entrada, archivo_salida, tamano_bloque=1000): Esta función implementa el algoritmo de ordenamiento externo utilizando el método de Mezcla Directa (Merge Sort). Toma tres argumentos: `archivo_entrada`, que es el nombre del archivo de entrada con los datos desordenados, `archivo_salida`, que es el nombre del archivo en el que se escribirán los datos ordenados, y `tamano_bloque` (por defecto, 1000), que es el

tamaño de bloque utilizado para la división y ordenamiento interno del archivo de entrada. La función divide los datos en bloques, ordena cada bloque internamente y luego fusiona estos bloques en el archivo de salida, manteniendo el orden ascendente.

3. `merge_blocks(sorted_blocks, file_salida)`: Esta función se utiliza dentro de `merge_sort_externo` para fusionar bloques ordenados. Toma dos argumentos: `sorted_blocks`, que es una lista de bloques ordenados, y `file_salida`, que es el archivo en el que se escribirán los datos fusionados en orden ascendente.

4. `crearyordenar(archivo_entrada, archivo_salida, mb)`: Esta función es la que coordina todo el proceso. Se encarga de crear un archivo de datos utilizando `crear_archivo_de_datos` con un tamaño especificado en megabytes (mb) y luego llama a `merge_sort_externo` para ordenar los datos y escribirlos en un archivo de salida.

El bloque de código en la parte inferior del script inicia el proceso al llamar a `crearyordenar` con los nombres de los archivos de entrada y salida, así como el tamaño en megabytes. Después de que se complete el ordenamiento, se imprime un mensaje indicando que el archivo se ha creado y ordenado correctamente.

Para luego verificar si el archivo está ordenado correctamente se utiliza el script verificarorden.py, el cual verifica de la forma enunciada en el trabajo práctico si el archivo ordenado cumple que los datos están ordenados y que pesan el mismo tamaño.