Facultad de Ingeniería
UNER

Informe sobre la implementación y análisis de algoritmos de ordenamiento

Algoritmos y Estructura de Datos.

Integrantes:

- Zapata Mariana Gabriela.
- Weimer Valentin.
- Kerbs Javier.

2° Cuatrimestre, 2025

Problema 2: Juego de Cartas “Guerra”

1. Introducción.

El segundo ejercicio consistió en el desarrollo de estructuras de datos que se usaron al simular el juego de cartas (guerra) con el fin de implementar algoritmos (como por ejemplo la inserción, extracción y recorrido de una lista doblemente enlazada) en la clase mazo que debemos desarrollar, usando la lista doblemente enlazada creada en el proyecto 1 para almacenar el algoritmo “carta” presentado por la cátedra, y luego se ejecutó el algoritmo juegoguerra y verificar que este funcionaba correctamente. Además se comprobó la ejecución correcta de test mazo y testjuegoguerra. Para ello se aplicaron conceptos de estructuras de datos como el manejo de inserción, extracción en listas y el manejo de excepciones al intentar interactuar con mazos vacíos.

2. Solución implementada.

Clase Mazo:

Estructura interna:
Se utiliza la ListaDobleEnlazada del Problema 1 para almacenar el algoritmo Carta. Esto permite que se lleven a cabo las operaciones de inserción y extracción tanto al inicio como al final del mazo.
Operaciones implementadas:

- poner_carta_arriba: Inserta la carta al inicio del mazo
- poner_carta_abajo: Inserta la carta al final del mazo
- sacar_carta_arriba: Extrae la carta del inicio del mazo. Si el mazo está vacío, lanza la excepción DequeEmptyError.
- __len__(): Devuelve la cantidad de cartas en el mazo.
- __str__(): Representa el mazo como cadena de cartas visibles.

3. Resultados.

TestMazo: poner_carta_arriba y sacar_carta_arriba funcionan correctamente y poner_carta_abajo y extracción del inicio funcionan en orden esperado.
TestJuegoGuerra: las partidas devuelven el ganador correcto y el número de turnos esperado, aunque existen casos de empate.

4. Análisis de complejidad.

El análisis de complejidad es el siguiente:

- poner_carta_arriba: O(1), ya que solo se modifica el puntero al inicio del mazo.
- poner_carta_abajo: O(1), solo se actualiza el puntero al final del mazo.
- sacar_carta_arriba: O(1), se elimina el nodo del inicio sin recorrer la lista.
- __len__: O(1), porque la clase guarda en un contador la cantidad de cartas y devuelve dicho valor.
- __str__: O(n), donde n es el número de cartas en el mazo, y se recorre toda la lista para mostrar el contenido del mismo en texto.

Justificación: se utiliza una lista doblemente enlazada, por lo que insertar o eliminar al inicio o final es constante, mientras que recorrer todas las cartas para imprimir el mazo requiere de un tiempo lineal.

5. Conclusión general.

La clase mazo permite administrar cartas de manera eficiente usando una lista doble enlazada, las operaciones cumplen con los requerimientos de los tests provistos por la cátedra y de la clase JuegoGuerra, el manejo de errores con DequeEmptyError permite que no se produzcan fallas y la implementación es eficiente y escalable, manteniendo complejidad O(1).