Proyecto Final Análisis y Diseño de Algoritmos I

Sofia Castillo Giraldo

Merly Velásquez Cortez

Faculta de Ingeniera, Universidad del Valle

Análisis y Diseño de Algoritmos I

Mauricio López Benítez

17 de diciembre del 2024

1. Problema

Se necesita una aplicación que permita, a partir de árboles binarios de búsqueda: ordenar una sucesión de palabras en castellano.

- Requerimientos
    a) Lee la sucesión de palabras desde un archivo de texto plano, donde cada palabra estará en una línea diferente del archivo.
    b) Imprime en pantalla la sucesión ordenada de las palabras (recorrido in-orden del árbol).
    c) Comprobación de palabras en el árbol y mostrar comparaciones realizadas.
    d) Imprime en pantalla el árbol rojinegro.

2. Resolución del problema

Se crearon 4 archivos .py para la aplicación:

- arbol_rojinegro.py: Se encuentra la implementación completa del árbol rojinegro.
  Se creo una clase NodoRB para inicializar el nodo(palabra), su color, su derecha, su izquierda e inicializar a el padre.
  Luego, se creó una clase ArbolRojinegro, donde se iniciliza el nodo y la raíz como nill y el color de ese nodo como negro.
  En esta clase podemos encontrar 8 funciones que servirán para crear el árbol rojinegro mediante las palabras entrantes del txt.
  Funciones:
  - insertar: en esta función se asigna el nuevo nodo, su raíz y su padre, se hace las debidas comparaciones mediante un ciclo while y así determinar si el nodo es mayor o menor, igualmente si en el árbol tiene mas nodos, se hace la comparación para determinar el padre del nuevo nodo.
  - _balancear_insercion: en esta función entran los nuevos nodos y va configurando el árbol rojinegro, mediante un ciclo while que se encarga de hacer cumplir las propiedades de los arboles rojinegros y mediante eso cumplir los caso 1, caso 2 y caso 3. Además, realiza las debidas rotaciones (derecha e izquierda) llamando a sus correspondientes funciones.
  - _rotacion_izquierda y _rotacion_derecha: en estas funciones se hacen los debidos desplazamientos, realizando las nuevas asignaciones de padres a los nodos.

- Buscar: en esta función se busca la palabra en el archivo .txt y realiza las comparaciones al buscarlo y mediante la función buscar_palabra las muestra en la interfaz.
- Recorrido_inorden y Recorrido_inorden_recursivo: ordena las palabra inorden y las muestra en la interfaz mediante la función mostrar_palabras_ordenadas:.
- Cargar_y_llenar_arbol: en esta función se carga y se accede el archivo .txt para generar el árbol.
- Mostrar_arbol: indica si el árbol está vacío, si no llama la función genera_estructura_arbol que realiza el árbol y los muestra en la interfaz.

- árbol_BinarioBusqueda.py
  Se creo la clase del Nodo, donde se inicializa el nodo por la derecha e izquierda. Luego, la clase ArbolBinarioBusqueda contiene tiene 6 funciones:

  - insertar: inicializa el árbol para asignar la raíz con el primero nodo entrante y con else llama la función _insertar_nodo donde inserta los nodos al árbol binario de búsqueda.
  - Recorrido_inorden y _recorrido_inorden: en la primera función retorna la lista resultante con las palabras ordenadas y en la segunda realiza el recorrido inorden y los añade a las lista.
  - Buscar y _buscar: se encargan de buscar la palabra en el árbol binario de búsqueda y realiza las comparaciones.

- interfaz.py
  En este archivo se configuro la interfaz del árbol binario de búsqueda y muestra en la GUI lo que se retorna de las funciones del archivo árbol_BinarioBusqueda.py

- main.py
  inicializa la GUI del árbol binario de búsqueda y muestra la interfaz gráfica.

3. Estrategias de Programación

Las estrategias de programación que se utilizaron en este proyecto fueron las siguientes:

a) De estructura de datos se utilizó la estructura del Árbol rojinegro.
b) Se hizo encapsulamiento de las clases NodoBR y ArbolRojinegro.
c) Recursividad para realizar un tipo de recorrido in-orden en el árbol rojinegro.
d) Se hizo un manejo adecuado de el caso del nodo NIL y demás casos para validar los nodos y archivos.
e) Utilización de interfaz gráfica GUI mediante Tkinder, para realizar las operaciones necesarias.
f) Biblioteca filedialog, que permitió el manejo del archivo txt.
g) Estrategia de iteración mediante ciclos while y for.
h) Separación de funciones para mayor facilidad de manejo.
i) Generación de la estructura del árbol rojinegro, que representa de manera visual los nodos a la izquierda y a la derecha para su compresión.

4. Calculo Complejidad

En cada función del código se realizó el cálculo de su complejidad.

Operaciones del Árbol Rojinegro	
	Complejidad
Inserción	O (log n)
Búsqueda	O (log n)
Recorrido in-orden	O (n)

Operaciones de GUI	
Cargar y llenar el árbol	
Cargar archivo	O (m)
Insertar Palabras	O (m log n)
Complejidad total	O (m + m log n)
Mostrar palabras ordenadas	O(n)
Buscar palabras	O (log n)
Mostrar Árbol	O (n)

Complejidad total

En el peor caso (si se realiza todo):

O (m + m log n + n + k log n + n)

Donde:

- m: Número de palabras en el archivo.

- n: Número de palabras únicas (nodos del árbol).

- k: Número de búsquedas realizadas.

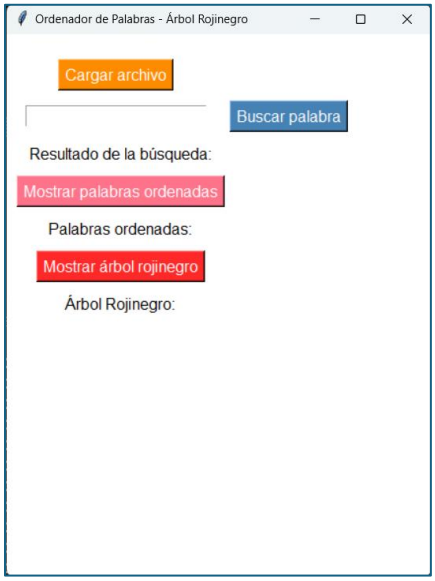
La complejidad dominante es:

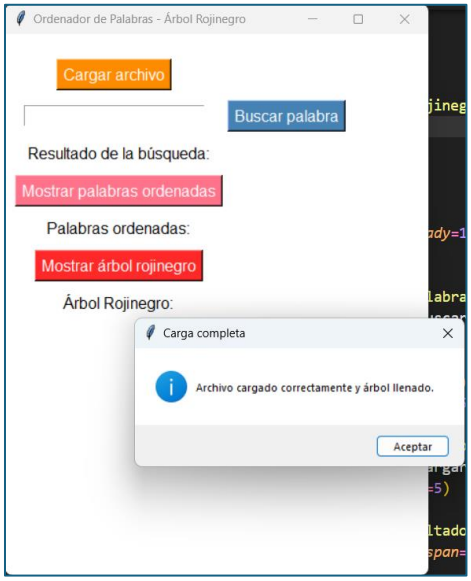
O (m log n + n)

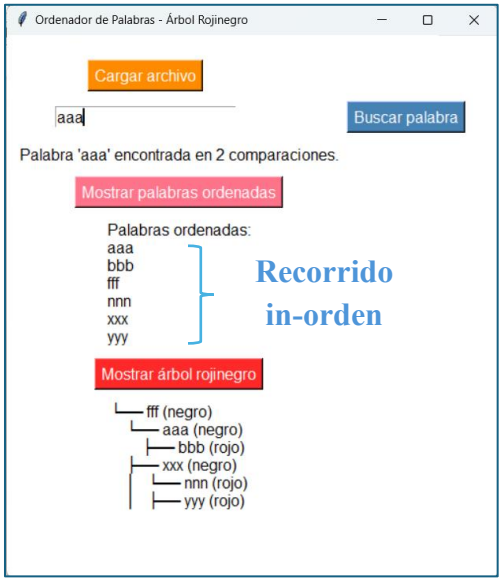
- Asumiendo que m (número de palabras en el archivo) es comparable o mayor que n (número de palabras únicas).

5. Resultados

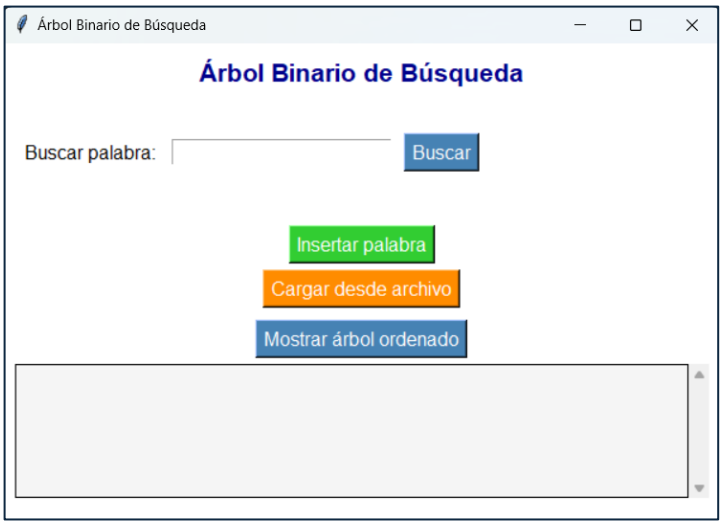
a) Árbol Rojinegro

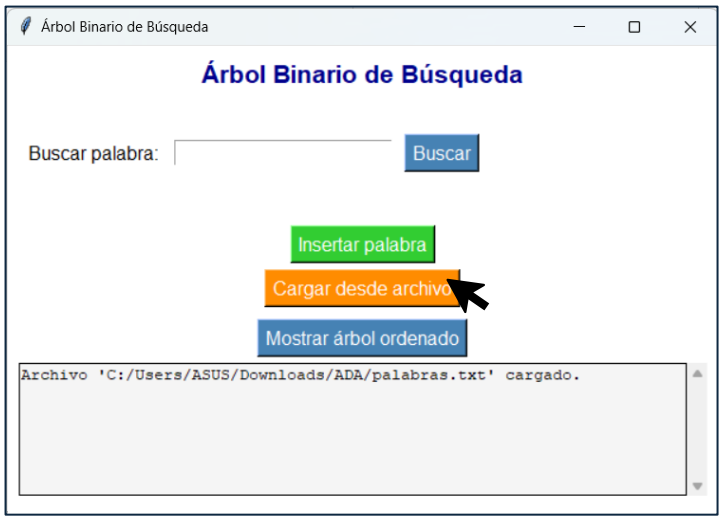


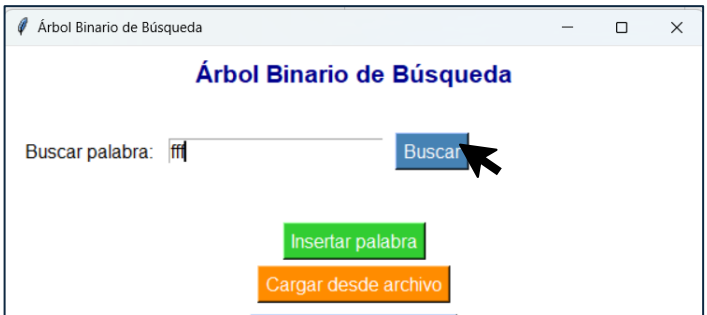


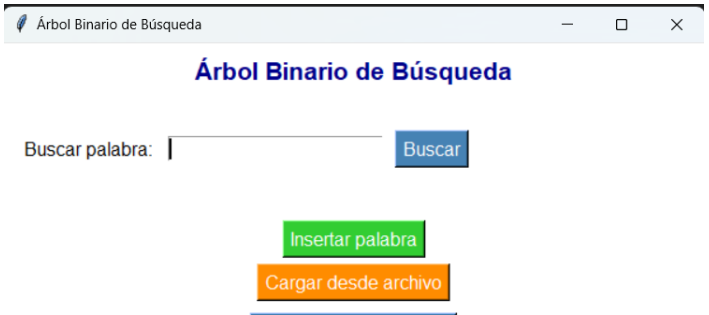


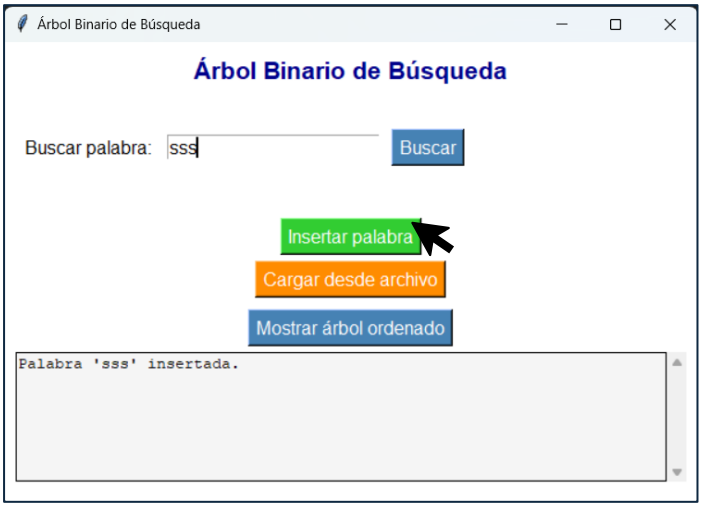
b) Árbol Binario de Búsqueda

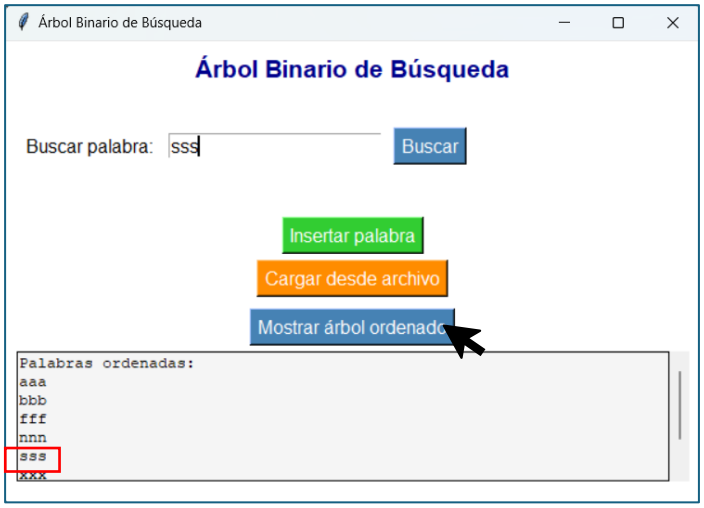












6. Conclusiones

Realizar este trabajo, el haber implementado las estructuras de datos vistas en clases y calcular la complejidad de cada una de las funciones, logro un mejor apropiamiento de los temas visto en clase. Además, la utilización de las estrategias de programación logró una optimización no solo en la complejidad sino también en el tiempo de desarrollo.