Complejidad Temporal, Estructuras de Datos y Algoritmos

Enunciado Trabajo Final

Consideraciones Generales

El objetivo de este trabajo es integrar los contenidos vistos en la materia y el mismo se deberá realizar en forma individual.

El trabajo integrador deberá defenderse en dos exposiciones. En la primera, se presentará una versión limitada del alcance solicitado, según lo indicado en este documento. En la segunda exposición, se mostrará la versión que cumple con el alcance completo. Las fechas límite para la defensa de la primera parte y del trabajo en su totalidad están publicadas en el aula virtual, en el enlace “Plan de trabajo”.

El trabajo se presentará junto con una presentación que debe incluir:

- Datos del autor del trabajo final.
- Un diagrama de clases UML describiendo la estructura del sistema, mostrando las clases del sistema, sus atributos, métodos y las relaciones entre los objetos.
- Detalles de implementación, problemas encontrados y como fueron solucionados, condiciones de ejecución, etc.
- Las imágenes de todas las pantallas que componen el sistema codificado junto con una descripción completa de las mismas.
- Ideas o sugerencias para mejorarlo o realizar una versión avanzada del mismo.
- Una breve conclusión o reflexión de la experiencia adquirida a partir de la realización del trabajo final.

El trabajo deberá defenderse en una exposición presencial.

La presentación será evaluada tanto por su contenido como por la forma en que se exponga, y su calificación impactará en la nota final del trabajo

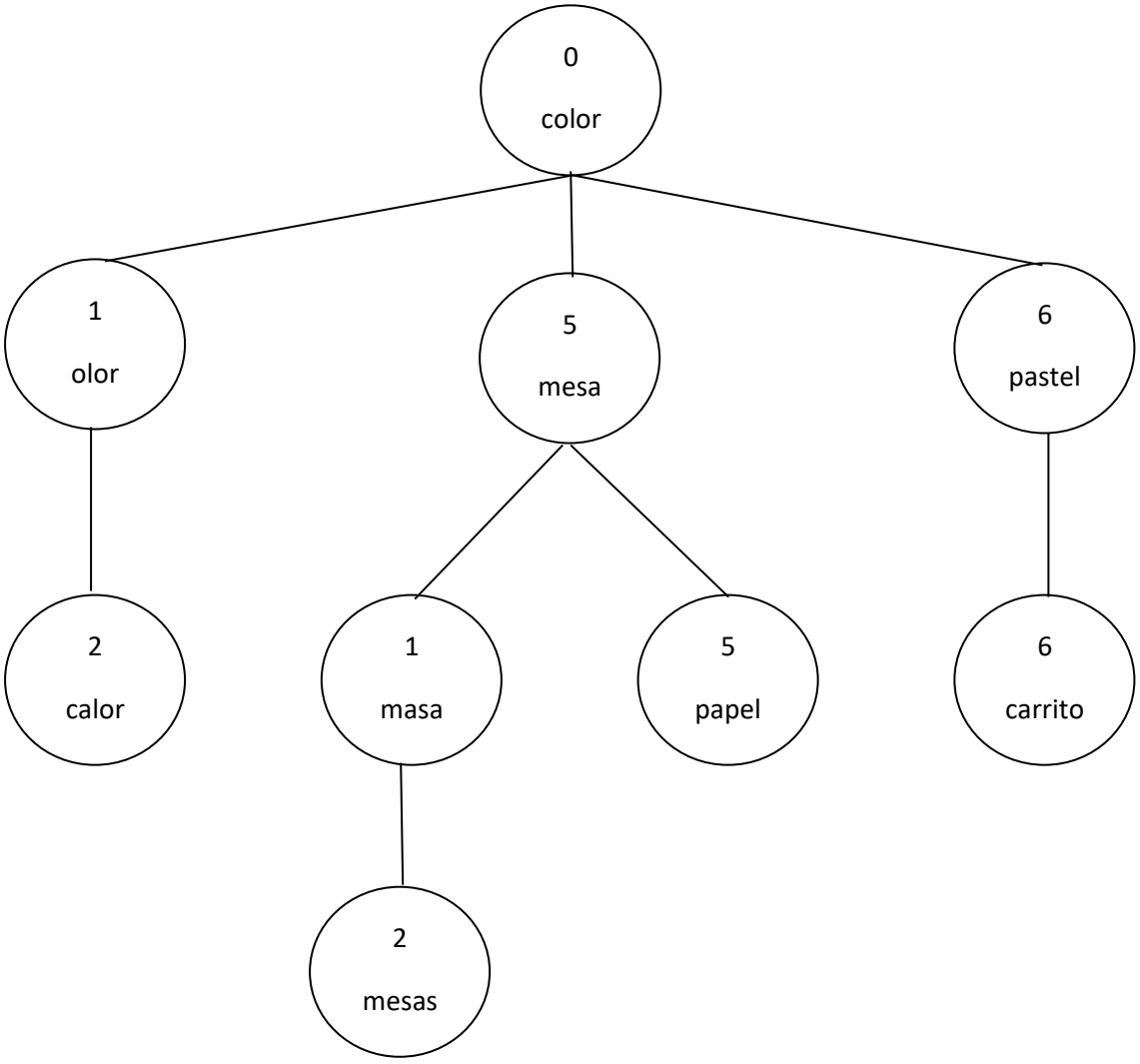
Enunciado

Un ***árbol BK*** (*BK-tree*) es un árbol general métrico presentado por Walter Austin Burkhard y Robert M. Keller. Así, un árbol métrico es una estructura de datos que esta específicamente adaptado obtener búsquedas e indexación de datos en forma eficiente. Además este tipo de estructura de datos utiliza una función de medida, d(x,y), que generalmente y por simplicidad calcula valore enteros como métricas. Entonces, BK-tree se define de la siguiente manera:

- Un elemento cualquiera es tomado para la raíz del árbol y este puede tener cero o más subárboles.
- El subárbol k-ésimo es construido recursivamente con todos los elementos b cuya distancia a la raíz es k, dicho de otra manera d(dato_raiz, b) = k.

Los árboles BK son ampliamente utilizados para realizar búsquedas de coincidencia aproximada entre cadenas de texto.

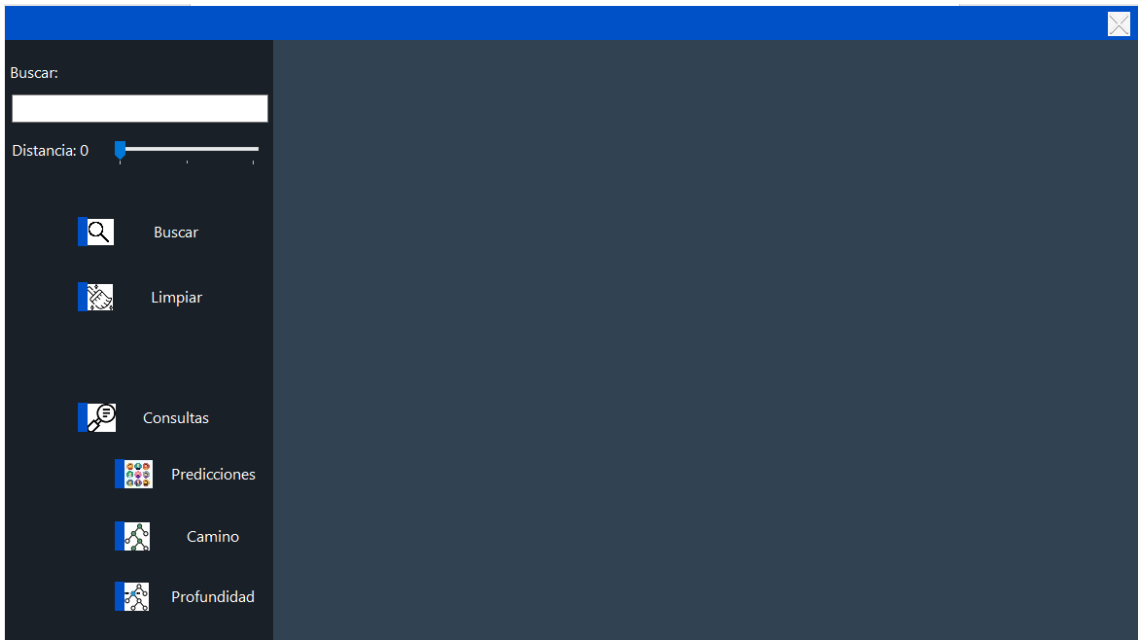
En la siguiente figura, se muestra un ejemplo en el cual se construye el árbol BK a partir de la siguiente entrada: {"color", "olor", "calor", "mesa", "masa", "mesas", "pastel", "papel", "carrito"} y como función de medida se utiliza la distancia de ***Levenshtein*** :



Nota: Observar que la distancia que existe entre el texto del nodo padre y el texto de cada uno de los hijos esta almacenada en los nodos hijo para esta implementación del árbol BK.

Una empresa informática está llevando a cabo un buscador de coincidencias aproximadas que tiene por objetivo indexar datos almacenados en un archivo csv y realizar búsquedas sobre los mismos.

El sistema inicia solicitando al usuario que seleccione el archivo csv donde se encuentran los recursos a indexar. A continuación, el buscador construirá el ***árbol BK*** con los datos provistos para luego proveer la posibilidad de realizar búsquedas desde la siguiente vista:



Para realizar una búsqueda es necesario introducir:

1. El término a buscar dentro del cuadro de texto indicado bajo la etiqueta "*Buscar*".
2. El nivel de coincidencia a través de la barra deslizante denominada "*Distancia*".

Para luego hacer "*click*" sobre el botón buscar, mostrándose los resultados obtenidos en el panel de la derecha.

La empresa informática lo ha contratado a Ud. para implementar la estrategia que utiliza el buscador y esta debe estar basada en el uso de un ***árbol BK***. En el sistema la codificación de esta última se realizará en la clase **Estrategia** a través de los siguientes métodos que Ud. debe implementar:

1. **AgregarDato** (ArbolGeneral<DatoDistancia> arbol, DatoDistancia dato): Este método agrega un nuevo dato a un árbol BK, donde tanto el dato como el árbol son enviados como parámetro.
2. **Buscar**(ArbolGeneral<DatoDistancia> arbol, string elementoABuscar, int umbral, List<DatoDistancia> collected): Retorna en la lista llamada *collected* el resultado de realizar una búsqueda de la cadena almacenada en *elementoABuscar* sobre el árbol BK almacenado en el parámetro *arbol* y con un nivel de tolerancia indicado por el parámetro *umbral*.
3. **Consulta1** (ArbolGeneral<DatoDistancia> arbol): Retorna un texto con todas las hojas del árbol BK del sistema.

4. **Consulta2** (ArbolGeneral<DatoDistancia> arbol): Retorna un texto que contiene todos los caminos hasta cada hoja.

5. **Consulta3** (ArbolGeneral<DatoDistancia> arbol): Retorna un texto que contiene los datos almacenados en los nodos del árbol diferenciados por el nivel en que se encuentran.

En **la primera** de las dos exposiciones, **se deberá tener implementado y funcionando los puntos 1 y 2 del listado anterior de funcionalidades** para ser presentadas bajo los lineamientos antes descritos. En la segunda exposición, se deberá contar con el listado completo funcionando para ser presentado de manera análoga a lo realizado en la primera exposición.

La catedra proveerá clases utilitarias a fin de simplificar la construcción y prueba del buscador. A continuación, se describen los métodos más importantes:

Clase	**Métodos**
Estrategia	– **CalcularDistancia**(string str1, string str2): Calcula la distancia de ***Levenshtein*** que existe entre str1 y str2.
DatoDistancia	– **DatoDistancia**(int distancia, string texto, string descripcion): Construye un objeto que almacena un texto, una descripción detallada sobre el texto anterior y una distancia de ***Levenshtein***. – **ToString():** Devuelve un texto que contiene el dato almacenado en formato string .