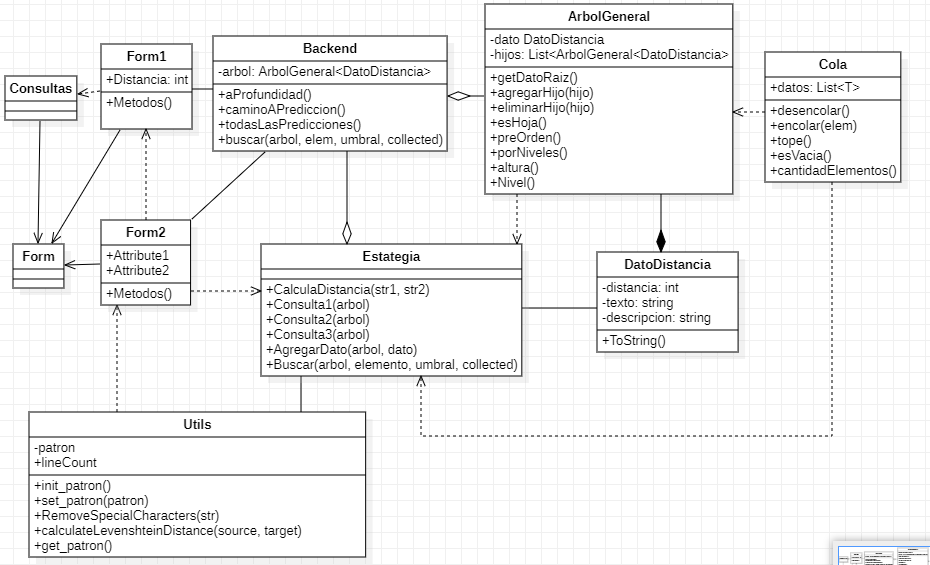
|  |  |  |
| --- | --- | --- |
|  | **Complejidad Temporal, Estructura de datos y Algoritmos**  Informe  **TRABAJO FINAL**  Buscador de coincidencias | **Comisión:** 5  **Profesor: Leonardo Amet**  **Alumno: Villalba Nicolas** |

**Introducción**

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.

**Metodología**

Para dicho programa, se utilizaron estrategias para la búsqueda de la palabra

****

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:

Claro, aquí te explico cómo funcionan cada uno de los métodos que has proporcionado:

1. Método `Consulta1(ArbolGeneral<DatoDistancia> arbol)`: Este método realiza una consulta en el árbol `arbol` y devuelve una cadena de texto con el resultado de la consulta. El método recorre el árbol en profundidad y, para cada nodo, verifica si es una hoja. Si es una hoja, se agrega su valor a la cadena de resultado. Luego, se procesan recursivamente todos los hijos del nodo actual. Finalmente, se devuelve la cadena de resultado que contiene los valores de las hojas.

2. Método `Consulta2(ArbolGeneral<DatoDistancia> arbol)`: Este método realiza otra consulta en el árbol `arbol` y devuelve una cadena de texto con el resultado. El método utiliza una cola para realizar un recorrido por niveles del árbol. Comienza encolando el nodo raíz y un camino vacío. Luego, en un bucle while, extrae un nodo de la cola y agrega su valor al camino actual. Si el nodo extraído es una hoja, agrega el camino actual a una lista de caminos. Si no es una hoja, agrega todos sus hijos a la cola junto con el camino actualizado. Al final, se construye la cadena de texto con los caminos encontrados y se devuelve como resultado.

El bucle `foreach` se utiliza en el método `Consulta2` para recorrer los hijos de un nodo en el árbol. Veamos su funcionamiento en detalle:

1. Se crea una copia del camino actual utilizando `var nuevoCamino = new List<DatoDistancia>(caminoActual)`. Esto se hace para tener un nuevo camino separado para cada hijo y evitar modificar el camino actual que ya se ha recorrido. Al crear una nueva lista y asignarla a `nuevoCamino`, se crea una copia independiente de `caminoActual`.

2. Se encola el hijo actual en la cola `c` utilizando `c.encolar(hijo)`. Esto permite que el hijo sea procesado en las iteraciones posteriores del bucle while.

3. Se encola la copia del camino actual actualizado en la cola `caminosActuales` utilizando `caminosActuales.encolar(nuevoCamino)`. Esto se hace para mantener el camino actualizado a medida que se avanza en el recorrido del árbol. Al encolar la copia del camino actual en `caminosActuales`, se garantiza que cada hijo tenga su propio camino individual.

En resumen, el bucle `foreach` se encarga de recorrer cada hijo del nodo actual en el árbol y realizar las operaciones necesarias para continuar el recorrido del árbol, encolando los hijos y actualizando los caminos correspondientes. Esto permite explorar todos los posibles caminos desde la raíz hasta las hojas del árbol.

3. Método `Consulta3(ArbolGeneral<DatoDistancia> arbol) `: Este método realiza una tercera consulta en el árbol `arbol` y devuelve una cadena de texto con el resultado. Utiliza una cola para recorrer el árbol por niveles. Comienza encolando el nodo raíz y luego, en un bucle while, extrae los nodos de la cola uno por uno. Por cada nodo extraído, se agrega su valor a la cadena de resultado y, si tiene hijos, se encolan todos los hijos. Al final de cada nivel, se incrementa el nivel actual y se continúa con el siguiente nivel hasta que la cola esté vacía. El resultado es una cadena de texto que muestra los valores de los nodos en cada nivel.

4. Método `AgregarDato(ArbolGeneral<DatoDistancia> arbol, DatoDistancia dato)`: Este método agrega un nuevo dato (`dato`) al árbol `arbol`. Si el nodo raíz del árbol está vacío, se crea un nuevo árbol con el dato proporcionado. En caso contrario, se verifica cada hijo del nodo raíz para determinar si el dato ya existe en el árbol. Si se encuentra un hijo con el mismo valor, se detiene la operación. Si el dato no existe en ninguno de los hijos, se calcula la distancia entre el valor del nodo raíz y el nuevo dato. Si la distancia es mayor que cero, se crea un nuevo nodo con el dato y se agrega como hijo del nodo raíz.

Si el árbol ya tiene una raíz, el método recorre los hijos existentes del árbol para verificar si alguno de ellos ya contiene el mismo dato. Esto se hace mediante el cálculo de la distancia entre el texto del dato de cada hijo y el texto del dato que se desea agregar. Si se encuentra un hijo con una distancia igual a cero, significa que el dato ya existe en el árbol y no se agrega nuevamente.

Si ningún hijo tiene una distancia igual a cero, el método calcula la distancia entre el texto del dato de la raíz del árbol y el texto del dato que se desea agregar. Si la distancia es mayor que cero, significa que el dato no existe en el árbol y se crea un nuevo nodo con el dato proporcionado y se agrega como hijo del nodo actual.

**Conclusión**