Universidad Nacional ARTURO JAURETCHE	**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 varias estrategias para la búsqueda de coincidencias. Para ello se creó la clase 'Estrategia' en la cuenta con 6 métodos. El primer método es un mecanismo que se utiliza para medir la diferencia entre dos secuencias, a esta métrica se la conoce como 'la distancia de Levenshtein' creado por el matemático Vladimir Levenshtein. Lo que hace es recibir 2 string por parámetro y compararlos para determinar a cuantas modificaciones de distancia esta un string del otro.

```
private int CalcularDistancia(string str1, string str2)
{
    // using the method
    String[] strlist1 = str1.ToLower().Split(' ');
    String[] strlist2 = str2.ToLower().Split(' ');
    int distance = 1000;
    foreach (String s1 in strlist1)
    {
        foreach (String s2 in strlist2)
        {
            distance = Math.Min(distance, Utils.calculateLevenshteinDistance(s1, s2));
        }
    }

    return distance;
}
```

Consulta1

El segundo método realiza una consulta en el árbolBK y devuelve una cadena de texto con el resultado de todas sus hojas. Lo que hace es recorrer 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.

```
public String Consulta1(ArbolGeneral<DatoDistancia> arbol)
{
    //Se crea una variable para almacenar el resultado
    string resultado = "";
    //Si es una hoja se retorna el texto de la misma.
    if (arbol.esHoja())
    {
        return arbol.getDatoRaiz().ToString() + " \n";
    }
    // Si no es una hoja, se procesan los hijos.
    if (arbol.getHijos() != null)
    {
        foreach (var a in arbol.getHijos())
            //Y por medio de la recursion vamos concatenado todos los textos en la varible.
            resultado += Consulta1(a);
    }
    return resultado;
}
```

Consulta2

El siguiente método, es otra consulta en el árbolBK y devuelve una cadena de texto con el resultado de todos los caminos hacia cada hoja. Para dicho método, se utilizó la clase cola para realizar un recorrido por niveles del árbol. Comienza encolando el nodo raíz y un camino vacío. Luego, en un bucle while, se desencola el árbol en una variable auxiliar y se desencola el camino actual CaminoActual de la cola caminosActuales. Aux representa el arbol actual que se está procesando y caminoActual almacena el camino desde la raíz hasta el nodo actual.

Se crea una copia del camino actual utilizando la variable nuevoCamino. 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. Se encola el hijo actual en la cola c. Esto permite que el hijo sea procesado en las iteraciones posteriores del bucle while. Después, se encola la copia del camino actual actualizado en la cola caminosActuales. 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.

Por último, mediante un foreach se recorre cada camino encontrado en la lista caminosEncontrados y se agrega la información de cada elemento del camino a la variable resultado y se retorna.

```
public string Consulta2(ArbolGeneral<DatoDistancia> arbol)
{
    /*En primer lugar, se declaran las variables que nos van a ayudar a almacenar4
      los caminos encontrados, los nodos y los caminos actuales.*/
    List<List<DatoDistancia>> caminosEncontrados = new List<List<DatoDistancia>>();
    Cola<ArbolGeneral<DatoDistancia>> c = new Cola<ArbolGeneral<DatoDistancia>>();
    Cola<List<DatoDistancia>> caminosActuales = new Cola<List<DatoDistancia>>();

    //Inicializamos con la raíz del árbol y un camino vacío.
    c.encolar(arbol);
    caminosActuales.encolar(new List<DatoDistancia>());

    //Mientras haya elementos en la cola, iterar.
    while (c.cantidadElementos() > 0)
    {
        /* La variable aux se utiliza para almacenar el nodo actual que se está
           procesando en cada iteración del bucle while al igual que el camino actual
             que se ha recorrido desde la raíz hasta aux */
        ArbolGeneral<DatoDistancia> aux = c.desencolar();
        List<DatoDistancia> caminoActual = caminosActuales.desencolar();
        caminoActual.Add(aux.getDatoRaiz());
```

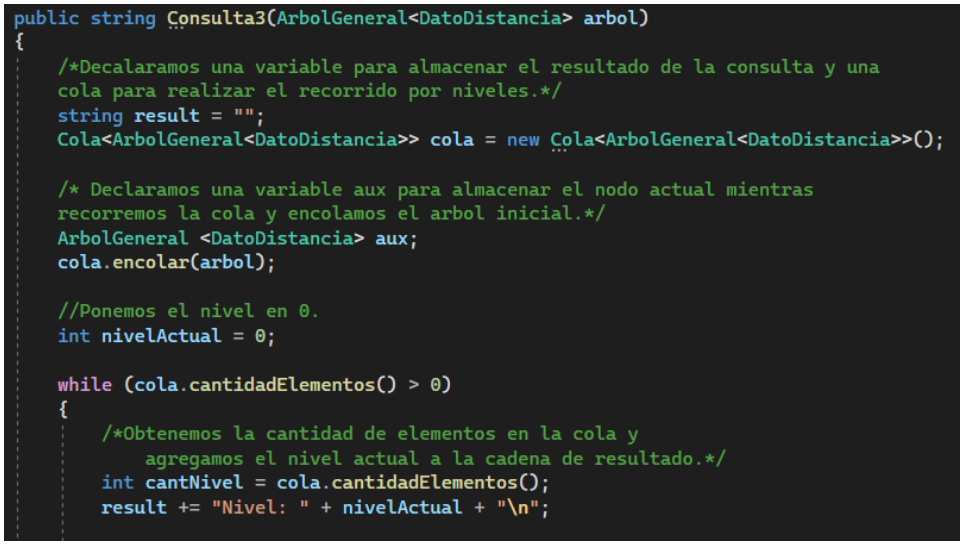
```
        if (aux.esHoja())
        {
            //Si llego a una hoja, agrega el camino actual a la lista de caminos
            caminosEncontrados.Add(caminoActual);
        }
        else
        {
            //Sino se recorre cada hijo del nodo actual y se actualizan los caminos
            foreach (var hijo in aux.getHijos())
            {
                /*Se crea una copia del camino actual para tener un nuevo camino
                 por separado para cada hijo y evitar modificar el camino
                actual que ya se ha recorrido.*/
                var nuevoCamino = new List<DatoDistancia>(caminoActual);
                c.encolar(hijo);
                caminosActuales.encolar(nuevoCamino);
            }
        }
    }
    //Por ultimo, devolvemos la cadena de texto con los caminos encontrados
    string resultado = "";
    //Agreamos un contador para enumerar los caminos.
    int n = 0;
```

```
    foreach (var camino in caminosEncontrados)
    {
        n++;
        resultado += "Camino n°"+ n +": \n";

        foreach (var ca in camino)
        {
            resultado += ca.ToString() + "\n";
        }
        resultado+= "\n";
    }
        return resultado;
}
```

Consulta3

Este método realiza un recorrido por niveles en un árbol general y devuelve una cadena de texto que muestra los nodos visitados agrupados por nivel. Para el recorrido por niveles dentro de árbol se utiliza una cola. En cada nivel, se desencolan los nodos y se agrega su dato raíz a la cadena de resultado. Luego, se encolan los hijos de cada nodo desencolado. El proceso se repite hasta que se hayan visitado todos los nodos del árbol. Al final, se devuelve todos los textos agrupados por niveles.

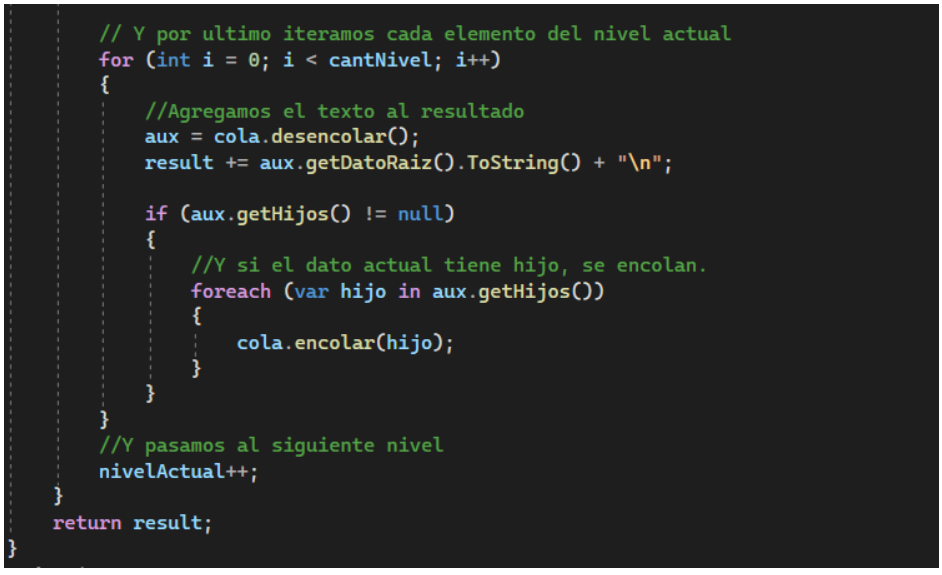


```
public string Consulta3(ArbolGeneral<DatoDistancia> arbol)
{
    /*Decalaramos una variable para almacenar el resultado de la consulta y una
    cola para realizar el recorrido por niveles.*/
    string result = "";
    Cola<ArbolGeneral<DatoDistancia>> cola = new Cola<ArbolGeneral<DatoDistancia>>();

    /* Declaramos una variable aux para almacenar el nodo actual mientras
    recorremos la cola y encolamos el arbol inicial.*/
    ArbolGeneral <DatoDistancia> aux;
    cola.encolar(arbol);

    //Ponemos el nivel en 0.
    int nivelActual = 0;

    while (cola.cantidadElementos() > 0)
    {
        /*Obtenemos la cantidad de elementos en la cola y
            agregamos el nivel actual a la cadena de resultado.*/
        int cantNivel = cola.cantidadElementos();
        result += "Nivel: " + nivelActual + "\n";
```



```
        // Y por ultimo iteramos cada elemento del nivel actual
        for (int i = 0; i < cantNivel; i++)
        {
            //Agregamos el texto al resultado
            aux = cola.desencolar();
            result += aux.getDatoRaiz().ToString() + "\n";

            if (aux.getHijos() != null)
            {
                //Y si el dato actual tiene hijo, se encolan.
                foreach (var hijo in aux.getHijos())
                {
                    cola.encolar(hijo);
                }
            }
        }
        //Y pasamos al siguiente nivel
        nivelActual++;
    }
    return result;
}
```

AgregarDato

El cuarto método agrega un nuevo dato al árbolBK que funciona de la siguiente manera:

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. Para ello, se calcula la distancia con el método de la presente clase “CalcularDistancia” para verificar si el dato ingresado tiene una distancia igual a 0, eso quiere decir que el dato es el mismo que el ingresado, entonces no lo agrega. 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.

```
public void AgregarDato(ArbolGeneral<DatoDistancia> arbol, DatoDistancia dato)
{
    //Primero se agrega el dato si el arbol esta vacio
    if (arbol.getDatoRaiz() == null){
        arbol = new ArbolGeneral<DatoDistancia>(dato);
    }else{
        //sino recorro los hijos para ver si hay algun dato igual al ingresado
        foreach (ArbolGeneral<DatoDistancia> hijo in arbol.getHijos()){
            //se calcula la distancia, si es igual a 0
            //quiere decir que ya hay un dato igual al infresado enotnces no agrega nada

            int distancia = CalcularDistancia(hijo.getDatoRaiz().texto, dato.texto);
            if (distancia == 0){
                return;
            }
        }
        //Y si hay distancia, se agrega el dato
        int distanciaRaiz = CalcularDistancia(arbol.getDatoRaiz().texto, dato.texto);
        if (distanciaRaiz > 0){
            // Crear un nuevo nodo para el dato y agregarlo como hijo del nodo actual
            arbol.getHijos().Add(new ArbolGeneral<DatoDistancia>(dato));
        }
    }
}
```

Buscar

El último método realiza una búsqueda en profundidad sobre el árbolBK para encontrar los elementos que cumplan con el rango de distancia ingresado por el usuario. Si la distancia es menor o igual al umbral, se agrega el elemento a una lista. Luego, se repite el proceso con cada uno de los hijos del árbol de forma recursiva. Al finalizar, se devuelve la lista con los elementos encontrados.

```
public List<DatoDistancia> Buscar(ArbolGeneral<DatoDistancia> arbol, string elementoABuscar, int umbral, List<DatoDista
    {

    if (arbol.getDatoRaiz() == null)
    {
        return collected;
    }
    else
    {
        //calculamos la distancia, si el dato esta entre el umbral se agrega el dato a la coleccion
        int distancia = CalcularDistancia(arbol.getDatoRaiz().texto, elementoABuscar);
        if (distancia <= umbral)
        {
            collected.Add(arbol.getDatoRaiz());
        }
        //y por medio de recusion hacemos lo mismo con los hijos
        foreach (ArbolGeneral<DatoDistancia> hijo in arbol.getHijos())
        {
            Buscar(hijo, elementoABuscar, umbral, collected);
        }
    }
    return collected;
}
```

Diagrama UML

Para visualizar un poco todo el proyecto creado, mostramos como está conformado y que relaciones tienen todas las clases mediante el siguiente diagrama UML:

Empezamos con la clase Cola la cual tiene una lista de atributo de manera pública y se relaciona con la clase ArbolGeneral<DatoDistancia> mediante una relación de dependencia Esto significa que en la clase ArbolGeneral<DatoDistancia> se crea una instancia y se utilizan métodos encolar y desencolar de la clase Cola<T>.Del mismo modo, con la clase estrategia.

La clase ArbolGeneral<DatoDistancia> tiene una relación de composición con la clase DatoDistancia. Esto indica que la clase ArbolGeneral<DatoDistancia> contiene objetos de la clase DatoDistancia como parte de su estructura interna. También, tiene una relación de agregación con la clase Backend. Esto indica que la clase Backend utiliza objetos de la clase ArbolGeneral<DatoDistancia> en su implementación, pero los objetos de ArbolGeneral<DatoDistancia> pueden existir de forma independiente al Backend. Otra relación es con la clase Estrategia, la cual tiene una relación de dependencia. Esto indica que la clase Estrategia depende de la clase ArbolGeneral<DatoDistancia> en su implementación o en algún método específico.

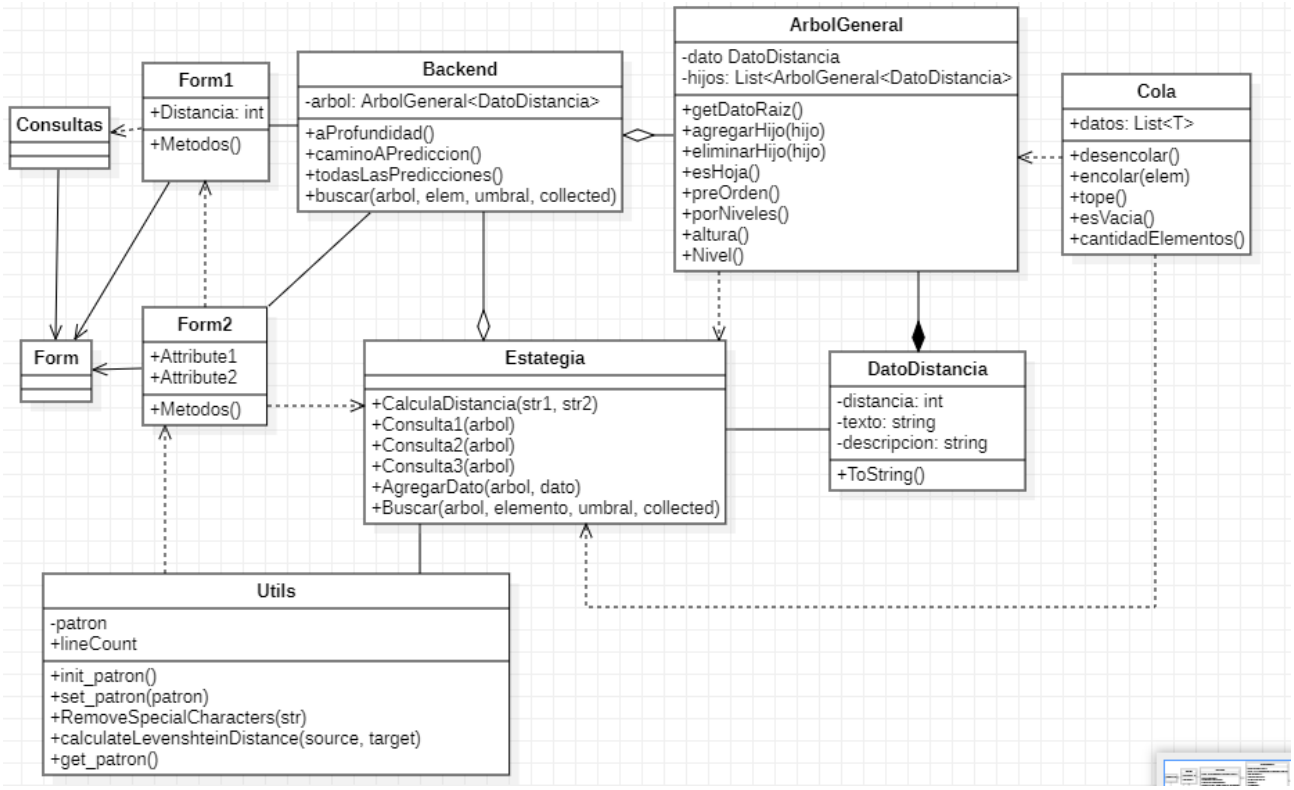
La clase Utils es una clase de utilidad que contiene métodos estáticos y no tiene una relación directacon las otras clases. Es utilizada por la clase Estrategia en el método CalcularDistancia.

La clase DatoDistancia tiene relación de asociación con Estrategia, ya que se pasa un dato distancia por uno de sus métodos.

La clase Backend utiliza la clase ArbolGeneral<DatoDistancia> en las propiedades estáticas árbol. También utiliza la clase Estrategia en los métodos aProfundidad, caminoAPrediccion, todasLasPredicciones y buscar.

La clase Backend tiene una relación de dependencia con la clase Estrategia. Esto indica que el backend utiliza la estrategia para realizar consultas. La relación se representa con una flecha punteada hacia Estrategia.

La clase Backend tiene una relación de dependencia con la clase Utils. Esto indica que el backend utiliza la clase Utils para ciertas operaciones auxiliares. La relación se representa con una flecha punteada hacia Utils.



Problemas encontrados

No encontraba la forma para que se guarden los caminos que iba encontrando en el método de colsulta2 y se resolvió agregando una cola y listas auxiliares de más para poder guardar esos caminos.

Ideas o sugerencias

Una idea para implementar en este buscador podría ser la opción de realizar búsquedas basadas en sinónimos o términos relacionados. Con esta implementación, los usuarios podrán obtener resultados más completos al realizar búsquedas, ya que se tendrán en cuenta los sinónimos o términos relacionados de los datos.

Conclusión

La realización de este trabajo final ha brindado la oportunidad de aplicar los conocimientos adquiridos durante la cursada, por el cual este proyecto ha sido muy beneficioso para entender por medio de la práctica cómo funcionan las estructuras de datos, algoritmos de búsqueda y manipulación de datos en un contexto real de un buscador de coincidencias aproximadas. Se ha trabajado en la implementación de diferentes estrategias de búsqueda y en la interacción entre diversas clases para lograr el funcionamiento deseado.