INFORME TRABAJO PRÁCTICO FINAL COMPLEJIDAD TEMPORAL, ESTRUCTURA DE DATOS Y ALGORITMOS

Alumno: Agustín Cabeda

DNI: 44419749

Comisión: N°5

Este informe tiene el objetivo de explicar el código y funcionamiento del trabajo práctico cuyo enunciado es el siguiente:

“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 árbol, DatoDistancia dato): Este método agrega un nuevo dato a un árbol BK, donde tanto el dato como el árbol son enviados como parámetro.

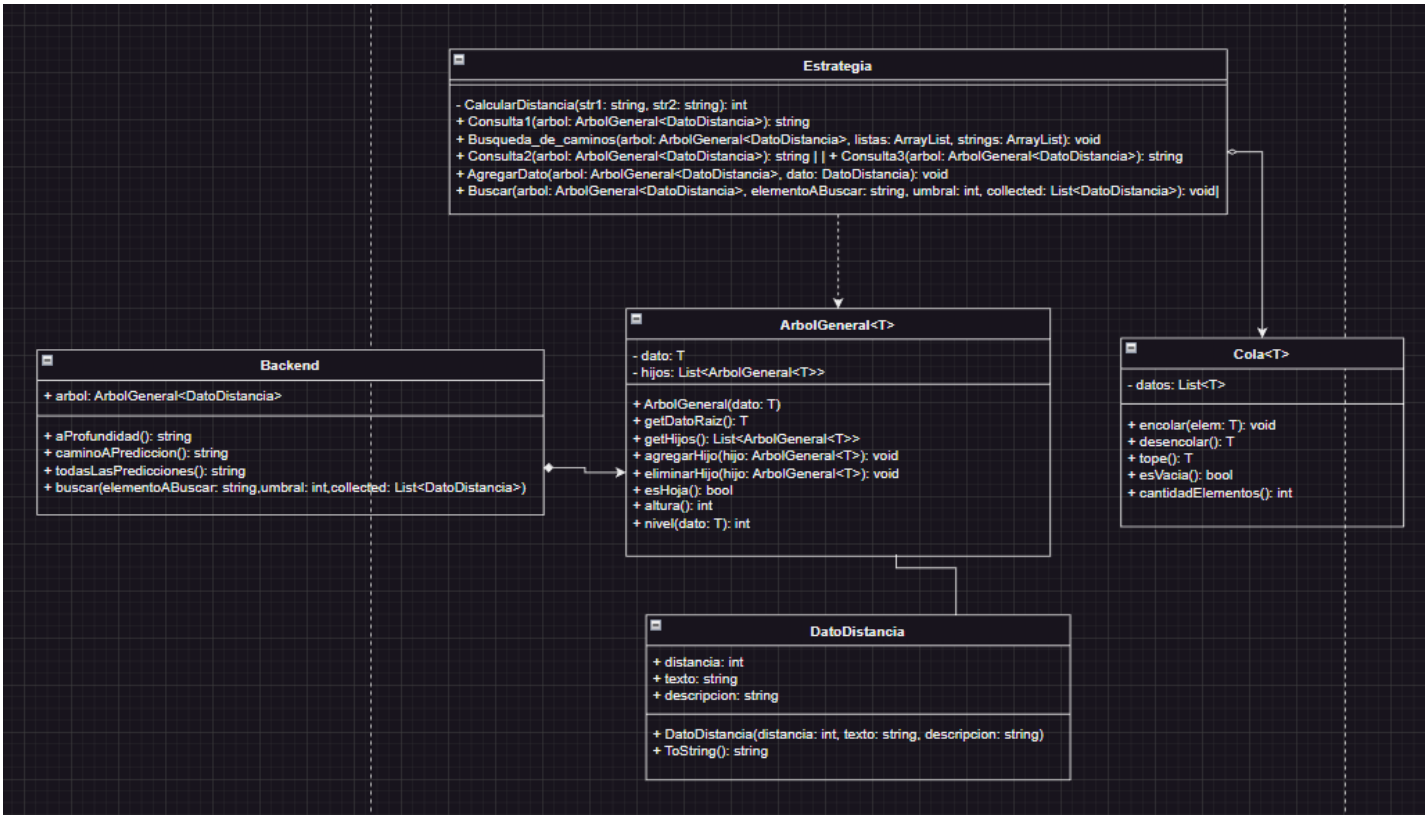
2. Buscar (ArbolGeneral árbol, string elementoABuscar, int umbral, List collected): Retorna en la lista llamada collected el resultado de realizar una búsqueda de la cadena almacenada en elementoABuscar sobre el árbol BK almacenado en el parámetro árbol y con un nivel de tolerancia indicado por el parámetro umbral.

3. Consulta1 (ArbolGeneral árbol): Retorna un texto con todas las hojas del árbol BK del sistema.

4. Consulta2 (ArbolGeneral árbol): Retorna un texto que contiene todos los caminos hasta cada hoja. “

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

Para hacer que el menú de la empresa funcione, se tuvieron que crear varias funciones. Para explicarlas, se realizó un diagrama UML relacionando todas las clases. A continuación, el diagrama UML:



En este trabajo final, se modificó y utilizó la clase llamada “ESTRATEGIA”.

Esta clase, se encarga de darle vida a un menú, que tiene de objetivo hacer una búsqueda y algunas consultas a un árbol que contiene varios títulos de películas. Sin duda esta es la clase más compleja, así que voy a explicar cada método “consulta” por separado.

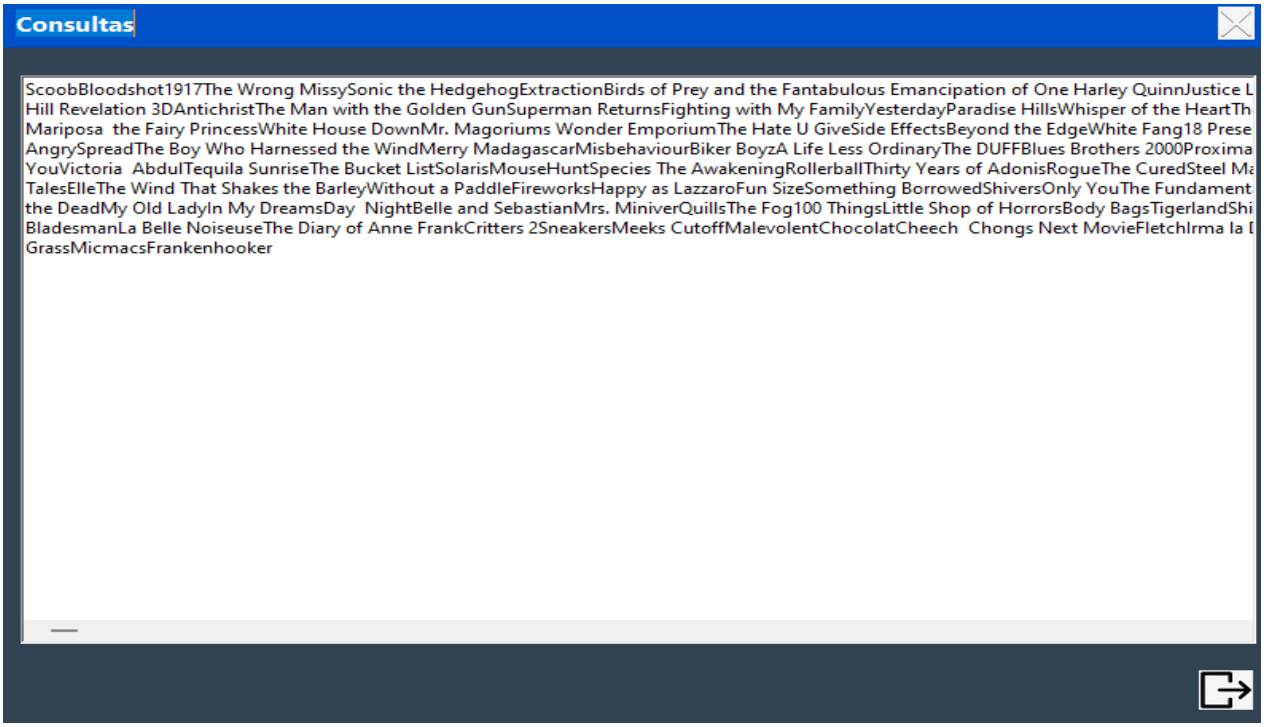
(Imágenes en la siguiente página)

```
public void AgregarDato(ArbolGeneral<DatoDistancia> arbol, DatoDistancia dato)
{
Cola<ArbolGeneral<DatoDistancia>> cola = new Cola<ArbolGeneral<DatoDistancia>>(); //creacion de la cola para recorrer el arbol general por niveles
ArbolGeneral<DatoDistancia> auxiliar; //creamos arbol auxiliar
cola.encolar(arbol); //encolamos el arbol que se paso de parametro
int distancia = CalcularDistancia(arbol.getDatoRaiz().texto, dato.texto); //creo variable que calcule la distancia entre la palabra de la raiz y el dato
if (distancia != arbol.getDatoRaiz().distancia) // si la distancia es diferente al dato raiz del arbol
{
    ArbolGeneral<DatoDistancia> nuevo = new ArbolGeneral<DatoDistancia>(dato); //se crea un arbol general con el DatoDistancia
    arbol.agregarHijo(nuevo); //se agrega ese arbol al arbol que se paso como parametro
}
while (!cola.esVacia()) //mientras la cola no este vacia se ejecuta lo de dentro
{
    auxiliar = cola.desencolar(); //se desencola un árbol de la cola y se asigna a el arbol aux
    foreach (var hijo in auxiliar.getHijos()) // hijos del arbol auxiliar
    {
        cola.encolar(hijo); //y los encolo
    }
    if (distancia == auxiliar.getDatoRaiz().distancia) // si la distancia calculada es igual a la distancia del dato raíz del árbol auxiliar
    {
        AgregarDato(auxiliar, dato); // Si son iguales, se llama recursivamente a la función AgregarDato,con parámetros el árbol aux y el objeto dato.
        //profundidad en el árbol para encontrar un nodo con la misma distancia, y se agregará el dato como hijo de ese nodo.
    }
}

}
```

- La función “AgregarDato” tiene como objetivo agrega un nuevo dato al árbol BK, para poder formarlo con la raíz y sus hojas.
  Su mecanismo consiste en crear una cola para recorrer el árbol con el método “Por niveles”. Esta función obtiene la distancia entre la raíz y el dato que se da como parámetro, y se fija la diferencia de distancia con la raíz.
  Si esta es diferente, se crea un nuevo árbol con el dato y se agrega como hijo del árbol original. Luego se inicia un bucle while que se ejecute mientras no este vacía la cola. Se desencola un árbol de la cola y se agarra la lista de sus hijos y se encola. Luego revisa si la distancia calculada previamente es igual a la raíz del árbol auxiliar, y si eso pasa, se llama recursivamente a la función “AgregarDato”.
  La verdad me costó bastante tiempo en entender como lograr hacer que la función funcione, y tuve que investigar bastante para lograr poder hacerlo

```
public String Consulta1(ArbolGeneral<DatoDistancia> arbol)
{
string palabra = ""; // Creo una oración donde voy a agrupar los textos de cada nodo
if (arbol.esHoja()) //si el arbol es una hoja
{
    palabra = palabra + "" + arbol.getDatoRaiz().texto; // Asigna el texto de la raíz a palabra_2
}
else //si no es hoja
{
    foreach (ArbolGeneral<DatoDistancia> hijo in arbol.getHijos()) //se itera sobre cada hijo del arbol
    {
        palabra = palabra + Consulta1(hijo); // Concatena los resultados de cada hijo recursivamente
    }
}
return palabra; // por ultimo retorno la oración
}
```



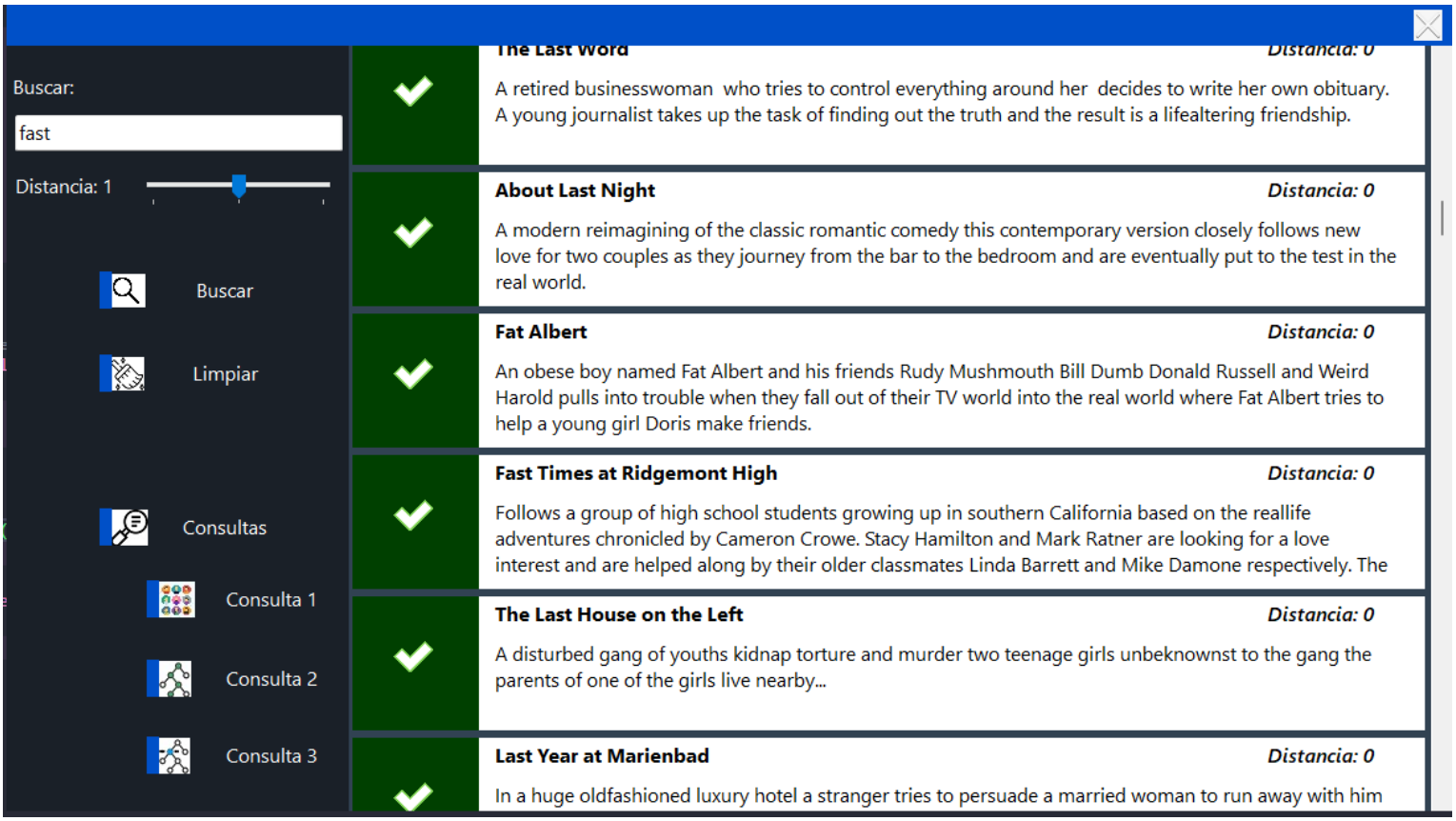
- La función “Consulta1” tiene como objetivo retornar un texto con todas las hojas.
  Esta misma inicia creando un string y verificando si el árbol pasado como parámetro es una hoja. Si no tiene hijos, se guarda la raíz simplemente. Si tiene hijos itera sobre cada uno y los agrega a la palabra mediante la recursividad.
  Esta función fue una de las mas simples en hacer y no me dio mayores contratiempos, lo mas costoso fue la parte de la recursividad, pero después de ver algunas clases grabadas por el profesor de la comisión, pude realizarlo.

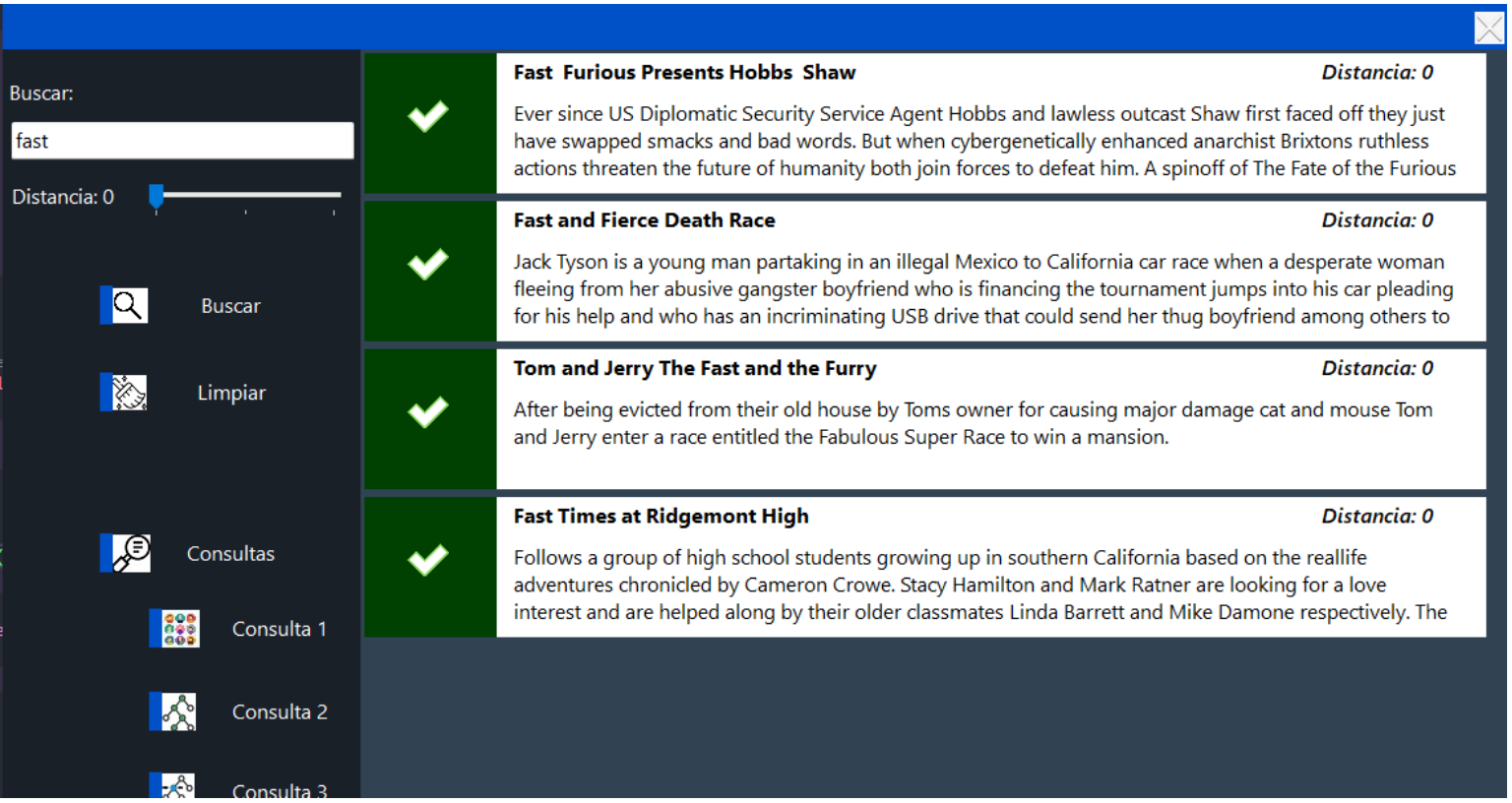
```
    2 references
    public void Buscar(ArbolGeneral<DatoDistancia> arbol, string elementoABuscar, int umbral, List<DatoDistancia> collected)
    {
        if (arbol == null) //si no hay arbol
        return; //retorna

    int distancia = CalcularDistancia(arbol.getDatoRaiz().texto, elementoABuscar); //calculamos la distancia entre la raiz y el dato que buscamos

    if (distancia <= umbral) //si el umbras es igual o mas grande que la distancia, se agrega a la lista
        collected.Add(arbol.getDatoRaiz());

    foreach (var hijo in arbol.getHijos()) //si no es asi, se busca a los hijos
    {
        Buscar(hijo, elementoABuscar, umbral, collected); //se llama recursivamente a la función Buscar con hijo, para agregarlos todos
    }
    }
}
```

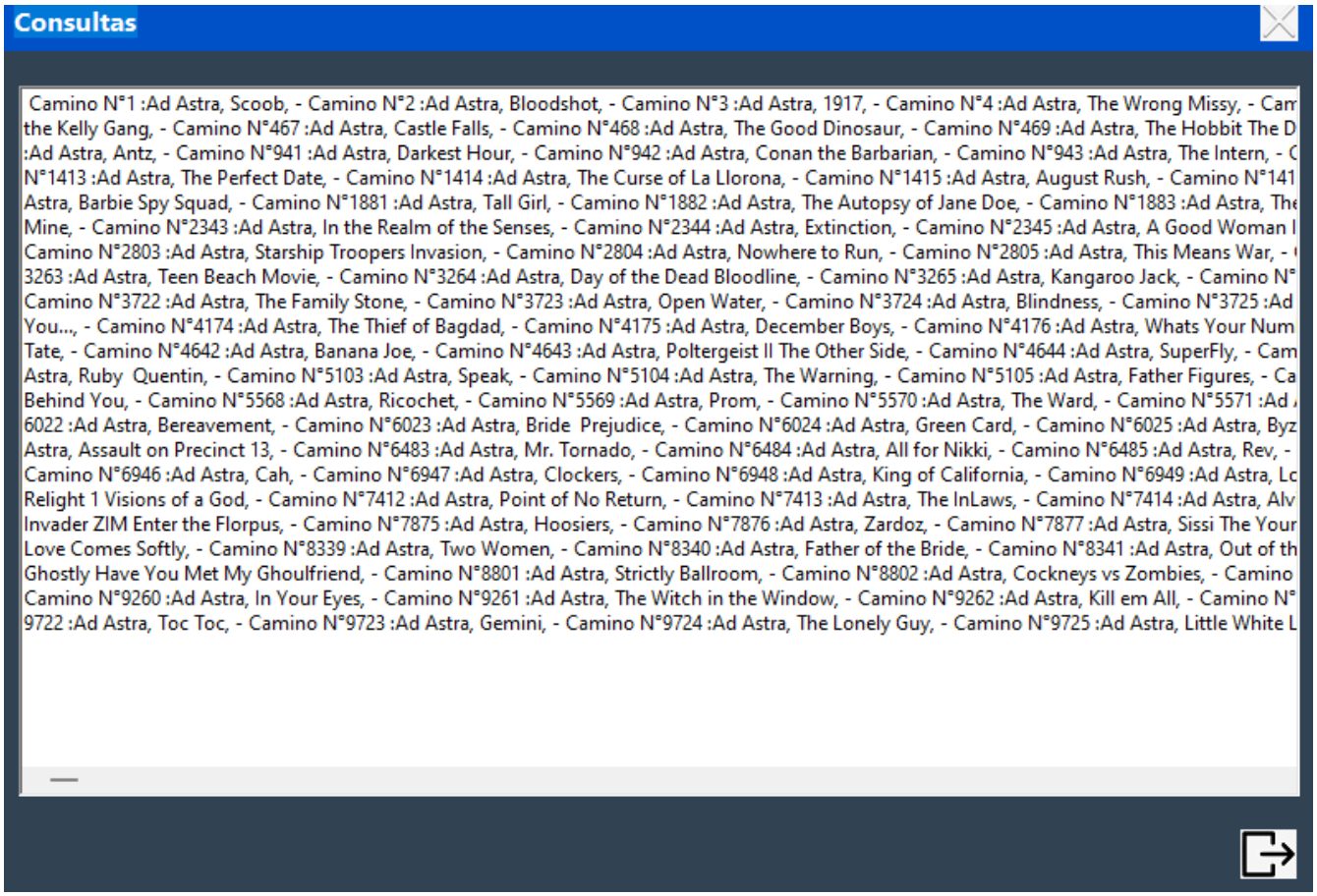

Buscar:
fast
Distancia: 1
Buscar
Limpiar
Consultas
Consulta 1
Consulta 2
Consulta 3
The Last Word
Distancia: 0
A retired businesswoman who tries to control everything around her decides to write her own obituary. A young journalist takes up the task of finding out the truth and the result is a lifealtering friendship.
About Last Night
Distancia: 0
A modern reimagining of the classic romantic comedy this contemporary version closely follows new love for two couples as they journey from the bar to the bedroom and are eventually put to the test in the real world.
Fat Albert
Distancia: 0
An obese boy named Fat Albert and his friends Rudy Mushmouth Bill Dumb Donald Russell and Weird Harold pulls into trouble when they fall out of their TV world into the real world where Fat Albert tries to help a young girl Doris make friends.
Fast Times at Ridgemont High
Distancia: 0
Follows a group of high school students growing up in southern California based on the reallife adventures chronicled by Cameron Crowe. Stacy Hamilton and Mark Ratner are looking for a love interest and are helped along by their older classmates Linda Barrett and Mike Damone respectively. The
The Last House on the Left
Distancia: 0
A disturbed gang of youths kidnap torture and murder two teenage girls unbeknownst to the gang the parents of one of the girls live nearby...
Last Year at Marienbad
Distancia: 0
In a huge oldfashioned luxury hotel a stranger tries to persuade a married woman to run away with him


Buscar:
fast
Distancia: 0
Buscar
Limpiar
Consultas
Consulta 1
Consulta 2
Consulta 3
Fast Furious Presents Hobbs Shaw
Distancia: 0
Ever since US Diplomatic Security Service Agent Hobbs and lawless outcast Shaw first faced off they just have swapped smacks and bad words. But when cybergenetically enhanced anarchist Brixtons ruthless actions threaten the future of humanity both join forces to defeat him. A spinoff of The Fate of the Furious
Fast and Fierce Death Race
Distancia: 0
Jack Tyson is a young man partaking in an illegal Mexico to California car race when a desperate woman fleeing from her abusive gangster boyfriend who is financing the tournament jumps into his car pleading for his help and who has an incriminating USB drive that could send her thug boyfriend among others to
Tom and Jerry The Fast and the Furry
Distancia: 0
After being evicted from their old house by Toms owner for causing major damage cat and mouse Tom and Jerry enter a race entitled the Fabulous Super Race to win a mansion.
Fast Times at Ridgemont High
Distancia: 0
Follows a group of high school students growing up in southern California based on the reallife adventures chronicled by Cameron Crowe. Stacy Hamilton and Mark Ratner are looking for a love interest and are helped along by their older classmates Linda Barrett and Mike Damone respectively. The

- La función “Buscar” tiene el objetivo de poder buscar las distintas películas que hay ubicadas en el árbol, dependiendo de la distancia que desee. En los ejemplos que puse, se busca todas las películas que tengan distancia 0 y 1 a la palabra “fast”.
  El método primero verifica si existe un árbol, y en caso de no existir retorna un null. Luego calcula la distancia entre la raíz y la palabra. Si la distancia esta dentro de lo permitido del umbras, se agrega a la lista. Si no, se usa la función recursivamente en los hijos.

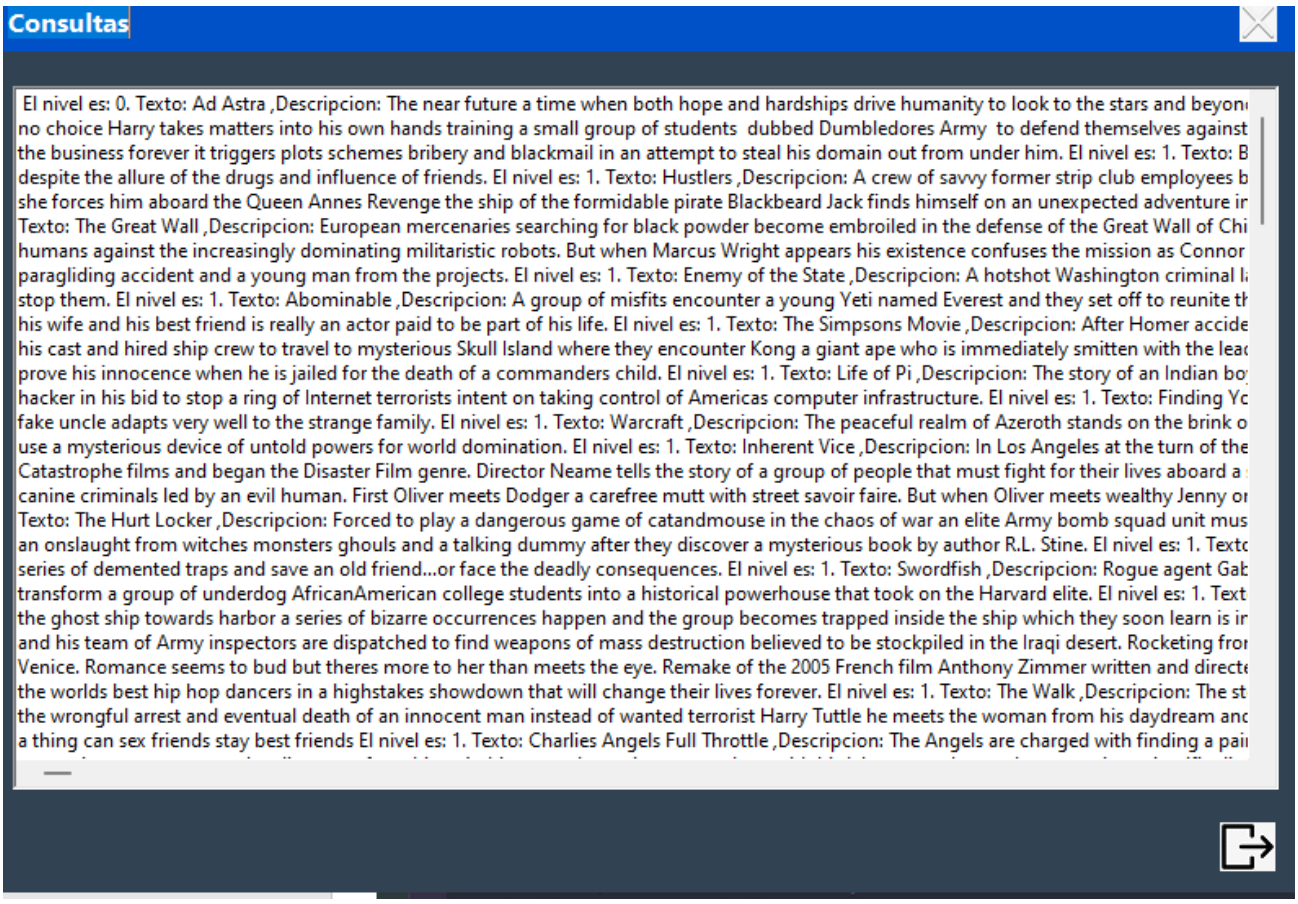
```
2 references
public void Busqueda_de_caminos(ArbolGeneral<DatoDistancia> arbol, ArrayList listas, ArrayList strings) //creamos función para los caminos
{
if (!arbol.esHoja()) //si no es una hoja
{
    strings.Add(arbol.getDatoRaiz().texto); //agregamos el texto del nodo raiz a la lista de strings
    foreach (ArbolGeneral<DatoDistancia> hijo in arbol.getHijos()) //hacemos una busqueda sobre cada hijo
    {
        Busqueda_de_caminos(hijo, listas, strings);  //se llama recursivamente a la función con el hijo
        strings.RemoveAt(strings.Count - 1); //se elimina el último elemento de lista1. Esto se hace para retroceder
        //y explorar otros caminos posibles desde el mismo nivel del árbol.
    }
}
else  //si es una hoja
{
    strings.Add(arbol.getDatoRaiz().texto);  //agregamos el texto del nodo a la lista de strings
    ArrayList lista_auxiliar = new ArrayList();  //creamos una lista auxiliar
    foreach (string ele in strings) //iteramos en la lista de los strings
    {
        lista_auxiliar.Add(ele); //se agrega a la lista auxiliar los strings de lista de strings
    }
    listas.Add(lista_auxiliar); //se agrega a la lista2 toda la lista auxiliar
}
}
```

```
public String Consulta2(ArbolGeneral<DatoDistancia> arbol)
{
    int contador = 1; //utilizamos un contador y lo inicializamos en 1
    ArrayList lista1 = new ArrayList(); //creamos una lista llamada lista1
    ArrayList lista2 = new ArrayList(); //creamos una lista llamada lista2
    string palabra2 = ""; //creo una variable donde se van a guardar el texto de cada nodo hacia las hojas
    Busqueda_de_caminos(arbol, lista1, lista2); //utilizo la función de busqueda de caminos
    foreach (ArrayList lista in lista1) //recorremos todas las listas dentro de lista2
    {
        palabra2 = palabra2 + " Camino N°" + contador + " :"; //agregamos el numero de contador a la palabra2
        foreach (string nodo in lista)  //recorremos cada nodo de la lista
        {
            palabra2 = palabra2 + nodo + ", "; //agregamos el texto del nodo seguido de una coma y un espacio
        }
        palabra2 = palabra2 + "-"; //agregamos separadores
        contador++; //aumentamos contador
    }
    return palabra2; //retorna oracion
}
1 reference
```



- La función “Consulta2” retorna un texto que contiene todos los caminos hasta cada hoja. Para hacerla, primero implemente otra función, llamada “Busqueda_de_caminos”.
  Busqueda_de_caminos guarda en una lista, otra lista con todos los strings que formando el camino. Primero verifica si no es una hoja, agrega el texto raíz a la lista strings y itera sobre los hijos y llama recursivamente a la función. Si es hoja, crea una lista auxiliar, y dentro guarda todos los strings.
  Luego Consulta2 funciona de la siguiente manera: Crea un contador y dos listas vacías, para luego llamar a la función Busqueda_de_caminos. Recorre dentro de lista1, y forma la oración pedida.
  Esta consulta me costo bastante tiempo de entender y tuve que pedir ayuda, ya que no tomaba en cuenta la idea de utilizar otra función, pero una vez que entendí el concepto pude hacerlo.

```
public String Consulta3(ArbolGeneral<DatoDistancia> arbol)
{
string palabras = ""; //creo una variable donde se va a guardar el texto de cada nodo hacia las hojas
DatoDistancia auxiliar = new DatoDistancia(1, "Texto", "Descripción"); //creo un objeto DatoDistancia
ArbolGeneral<DatoDistancia> contador = new ArbolGeneral<DatoDistancia>(auxiliar); //arbolgeneral que contiene auxiliar
int nivel = -1; //contador de niveles, inicializado en -1 para que al menos haga la iteracion en el nivel 0
if (arbol.esHoja()) //si es una hoja
{
    nivel= nivel+1; //subo el nivel para que quede en 0
    palabras = palabras + "El nivel es: " + nivel +". Texto: " + arbol.getDatoRaiz().texto + " ,Descripcion: " + arbol.getDatoRaiz().descripcion; //agregamos la información
}
else//si no es
{
    nivel=nivel+1; //subo el nivel
    Cola<ArbolGeneral<DatoDistancia>> cola = new Cola<ArbolGeneral<DatoDistancia>>();  //creo una cola para almacenar nodo
    ArbolGeneral<DatoDistancia> arbol_auxiliar;  // creo un arbol auxiliar que servira de puntero
    cola.encolar(arbol); //encolo el arbol pasado como parametro
    cola.encolar(contador); //encolo el contador
    while (!cola.esVacia()) //mientras la cola no este vacia
    {
        arbol_auxiliar = cola.desencolar(); //desencolo el primer elemento de la cola en el arbol
        if (arbol_auxiliar != contador) //mientras sea distinto a contador
        {
            palabras = palabras + " El nivel es: " + nivel +". Texto: " + arbol_auxiliar.getDatoRaiz().texto + " ,Descripcion: " + arbol_auxiliar.getDatoRaiz().descripcion;//a
            foreach (var hijo in arbol_auxiliar.getHijos()) //recorro todos los hijos del nodo utilizando un foreach
            {
                cola.encolar(hijo); //los encolo
            }
        }
        else //si es igual al contador
        {
            nivel =nivel+1; //subo el nivel
            if (!cola.esVacia())//si la cola esta vacia
            {
                cola.encolar(contador); //encolo el contador
            }
        }
    }
}
return palabras; //retorno las oraciones
}
```



- La función “Consulta3” retorna un texto que contiene los datos almacenados en los nodos del árbol diferenciados por su nivel.
  Lo primero que hace es crear un objeto de DatoDistancia auxiliar y un árbol que se utiliza de contador, seguido de una variable iniciada en -1 para luego al aumentarla que este en 0. Verifica si es una hoja, es decir, si no tiene hijos, y agrega el nivel y el texto a la oración.
  Si tiene hijos, crea una cola y un árbol auxiliar, y luego encola el árbol y el contador. Verifica que la cola no este vacía, es decir que haya árbol, desencolamos en el árbol auxiliar y agregamos al texto. Luego encolamos a todos los hijos, Si árbol auxiliar es igual al árbol contador, subimos el nivel y vemos que no este vacía, para encolar a contador.
  Esta consulta fue la que más me costó sin duda, tuve que buscar mucho en internet y aun así no lograba hacerla, ya que no tenía en cuenta crear otra función para ayudarme. Tuve la suerte, de que un compañero que ya curso esta materia, pudo ayudarme y explicarme cómo hacerlo, y así pude terminarlo.

Conclusiones

En conclusión, me parece un trabajo bastante complejo y difícil de hacer sin investigar o pedir ayuda (como en mi caso, a un estudiante que ya curso esta materia anteriormente), pero una vez que logras hacerlo sientes que valió la pena las horas sentado investigando. Personalmente no es un trabajo que me haya gustado mucho, ya que el lenguaje c# no es de mis favoritos, y cambiar de Python (mi lenguaje más utilizado a día de hoy) a un trabajo tan grande y que requiere mucho tiempo de otro lenguaje fue todo un desafío. Aun así, gracias al Previo ejercicio que se dio en la materia, pude adaptarme. Es una gran herramienta para entender más acerca de árboles y recursividad.
En cuanto a como mejorarlo, la verdad no hay mucho, si podría decir que podrían achicar la lista de películas, ya que si tu computadora no es tan buena podría tardarte mucho. Podría mejorarse la interfaz a una más amigable a la vista, y agregar algún método de consulta nuevo. También, podría agregarse al cronograma mas clases para ayudar con el trabajo.