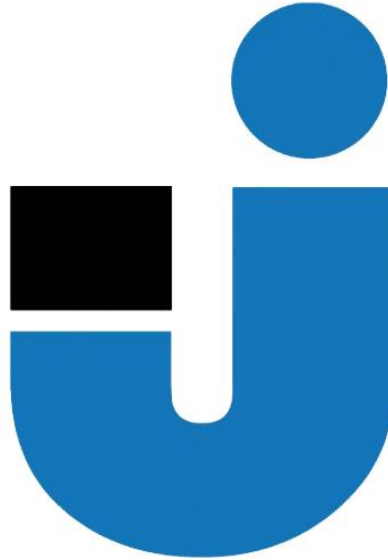


Universidad Nacional Arturo Jauretche



Universidad Nacional
ARTURO JAURETCHE

Trabajo Final Integrador

Materia: Complejidad Temporal de Estructura de Datos y Algoritmos

Comisión: 3

Autor: Emiliano Camer

Docente: Leonardo Javier Amet

Fecha: 14 de noviembre de 2022

Índice

| | |
|------------------------------|---|
| 1. Introducción | 3 |
| 2. Desarrollo..... | 3 |
| 2.1. Crear Árbol | 4 |
| 2.2. Consulta 1 | 5 |
| 2.3. Consulta 2 | 5 |
| 2.4 Consulta 3 | 7 |
| 3. Problemas y Mejoras | 8 |
| 4. Conclusiones | 9 |

1. Introducción

El siguiente informe consiste en la realización de un proyecto final de la materia “Complejidad Temporal de Estructuras de Datos y Algoritmos”.

Dicho proyecto se basa en la implementación del famoso juego “¿Quién es Quién?” de forma digital. Este consiste en elegir un personaje dentro de un set de distintas personas con diferentes atributos; para luego enfrentarnos contra una *Inteligencia Artificial* para ver quien logra adivinar el personaje que eligió el otro lo más rápido posible.



1. Captura del Juego

2. Desarrollo

Dicho juego fue desarrollado bajo un framework .NET, junto a una gran cantidad de clases y métodos, los cuales simulan tanto la inteligencia artificial de la máquina, como las preguntas, personaje, decisiones y formato del juego. Es un trabajo muy amplio, del cual una gran parte ya se nos otorga resuelta.

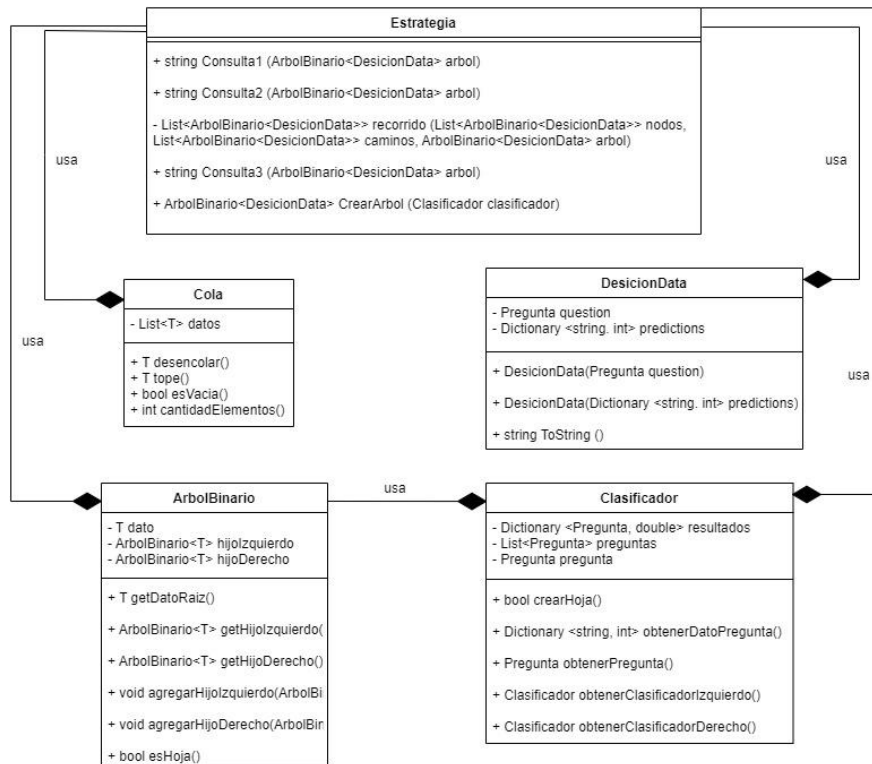
Nuestro desafío es poder implementar cuatro métodos dentro de la clase *Estrategia*, haciendo uso de algunas de las clases que tiene el proyecto.

Dichos métodos son:

1. CrearArbol (Clasificador clasificador): Este método construye un árbol de decisión el cual es la estructura del juego. Dicho árbol posee en sus *nodos internos*, las preguntas a realizar; y en sus *nodos hojas*; los personajes con sus probabilidades de ser o no tal personaje (que en este caso son del 100%).
2. Consulta1 (ArbolBinario< DecisionData > arbol): Retorna un texto con todas las posibles predicciones que puede calcular el árbol de decisión del sistema.
3. Consulta2 (ArbolBinario< DecisionData > arbol): Retorna un texto que contiene todos los caminos hasta cada predicción; es decir, todos los caminos existentes desde el *nodo raíz* hasta las hojas
4. Consulta3 (ArbolBinario< DecisionData > arbol): Retorna un texto que contiene los datos (sean preguntas o personajes) almacenados en los nodos del árbol diferenciados por el nivel en que se encuentran.

Para ello, debimos usar 3 de las clases que se encuentran en el programa:

- *DecisionData*: Crea el contenido de cada hoja, sean preguntas o personajes
- *Clasificador*: Se encargar de clasificar cada nodo, es decir, ubicarlos en el árbol de tal forma que sean nodos internos u hojas.
- *Cola*: Establece la estructura de almacenamiento de datos *Cola (Queue)*.
- *ArbolBinario*: Establece la estructura de almacenamiento de datos del tipo Árbol Binario.



2. UML de las clases utilizada, con sus métodos mas distintivos

2.1. Crear Árbol

Este método es el encargado de generar la estructura base del programa. Tal método recibe un clasificador como parámetro, y retorna un árbol binario de decisión.

Su implementación se hizo mediante la recursividad. Para el caso base, se busca saber si el nodo al que llegamos es una hoja. Esto lo hacemos mediante el método *“esHoja()”* de la clase Clasificador.

En caso de que el nodo sea una hoja, se crea el dato tipo *DecisiónData* que lleva almacenado. Luego se crea el nodo hoja, asignándole dicho valor, y retornándolo.

En caso contrario, se crea la estructura del árbol, el cual se va a ir completando con la recursividad. Primero se crea el dato *DecisionData* que va a almacenar el nodo. Luego se crea el árbol y se le asigna ese valor. Finalmente, a dicho nodo se le asigna un hijo izquierdo y derecho (por ser binario) de tipo árbol binario de decisión, los cuales se van a llamar a la función recursivamente, asignando como parámetro su clasificador.

```

//Creacion Árbol
public ArbolBinario<DecisionData> CrearArbol(Clasificador clasificador)
{
    if(clasificador.crearHoja()){
        DecisionData d = new DecisionData(clasificador.obtenerDatoHoja());
        ArbolBinario<DecisionData> nodo = new ArbolBinario<DecisionData>(d);
        return nodo;
    }
    else{
        DecisionData p = new DecisionData(clasificador.obtenerPregunta());
        ArbolBinario<DecisionData> arbolIzq = new ArbolBinario<DecisionData>(p);
        arbolIzq.agregarHijoIzquierdo(CrearArbol(clasificador.obtenerClasificadorIzquierdo()));
        arbolIzq.agregarHijoDerecho(CrearArbol(clasificador.obtenerClasificadorDerecho()));
        return arbolIzq;
    }
}
  
```

3. Método "CrearArbol"

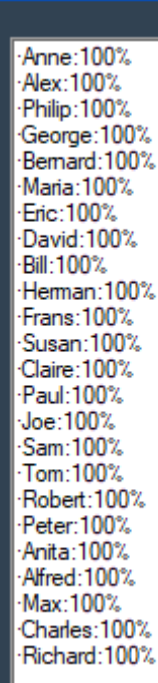
2.2. Consulta 1

Este método busca mostrar todos los nodos hoja del árbol binario en los cuales se encuentran los personajes junto con su probabilidad. Dicho método recibe un árbol binario de decisión como parámetro y retorna un string con todos los personajes.

Para la implementación, se decidió utilizar una variación del recorrido “*Post-Orden*” que también se resuelve recursivamente. Para eso, el método pregunta si el nodo en el que está parado es una hoja. En caso afirmativo, extrae su contenido y lo concatena en la variable “*texto*” la cual guarda todos los personajes. Antes que nada, dicho dato debe ser convertido a tipo string mediante la función “*.ToString()*” de la clase *DecisionData*. Finalmente se retorna ese texto. En caso de que no sea una hoja, se recorre sus hijos de izquierda a derecha llamando recursivamente a la función, reasignando la variable texto concatenándole todos los datos que vaya adquiriendo en el camino. Finalmente, se retorna el texto con todos los personajes.

```
//Consulta - 1
public String Consulta1(ArbolBinario<DecisionData> arbol)
{
    String texto = "";
    if(arbol.esHoja()){
        DecisionData result = arbol.getDatosRaiz();
        texto = texto + "." + result.ToString() + "\n";
        return texto;
    }
    if(arbol.getHijoIzquierdo() != null){
        texto = texto + this.Consulta1(arbol.getHijoIzquierdo());
    }

    if(arbol.getHijoDerecho() != null){
        texto = texto + this.Consulta1(arbol.getHijoDerecho());
    }
    return texto;
}
```



4. Método "Consulta 1" junto a su Mensaje en Pantalla

2.3. Consulta 2

Para este método se busca mostrar en pantalla todos los caminos del árbol, desde el nodo raíz hasta todas las hojas. Tal método recibe un árbol binario de decisión como parámetro, y retorna un string con todos los caminos.

Con el fin de llevar a cabo su implementación, se crearon de antemano dos listas *List<>* de tipo *ArbolBinario<DecisionData>*, siendo la llamada *nodos* la que almacena momentáneamente cada camino, y *caminos* la que almacena cada uno de los caminos.

Para facilitar la implementación, se creó un método privado auxiliar llamado dado “*recorrido*” el cual recibe como parámetro una lista de nodos, de caminos y un árbol de decisión. Dicha función va a ir guardando los distintos caminos en cada una de las variables. Para ello, primero guarda el dato del nodo árbol que está parado en *nodos*. Luego, pregunta si dicho nodo es una hoja o no. En caso que sí, llegamos al caso base; ósea, encontramos uno de los caminos. Por ende, realizamos una copia de lo que tiene la lista *nodos* en la lista *caminos* mediante la función “*.AddRange()*” de *List<>*, la cual copia dato por dato de una lista a otra.

Por otra parte, en caso de que no sea hoja, se recorren los hijos del nodo hoja de izquierda a derecha, llamando

recursivamente a la función, enviándole por parámetro la misma listas *nodos* y *caminos*, pero como árbol le manda tanto su hijo izquierdo, como el derecho. Una vez que la recursividad llega al caso base, a la vuelta se va quitando el último elemento (nodo árbol) de las lista *nodos*; sin afectar a *caminos*. Finalmente, la función retorna los *caminos*.

Ya en el método principal (*Consulta2*), lo que retorna la función *recorrido* se le asigna a la variable *caminos*. Ya con todos los caminos, mediante un *foreach* se recorre cada elemento de la lista, concatenándolos a la variable “*texto*”. Aquí, al igual que en la *Consulta1*, se utiliza el método *.ToString()* para convertir el dato a string.

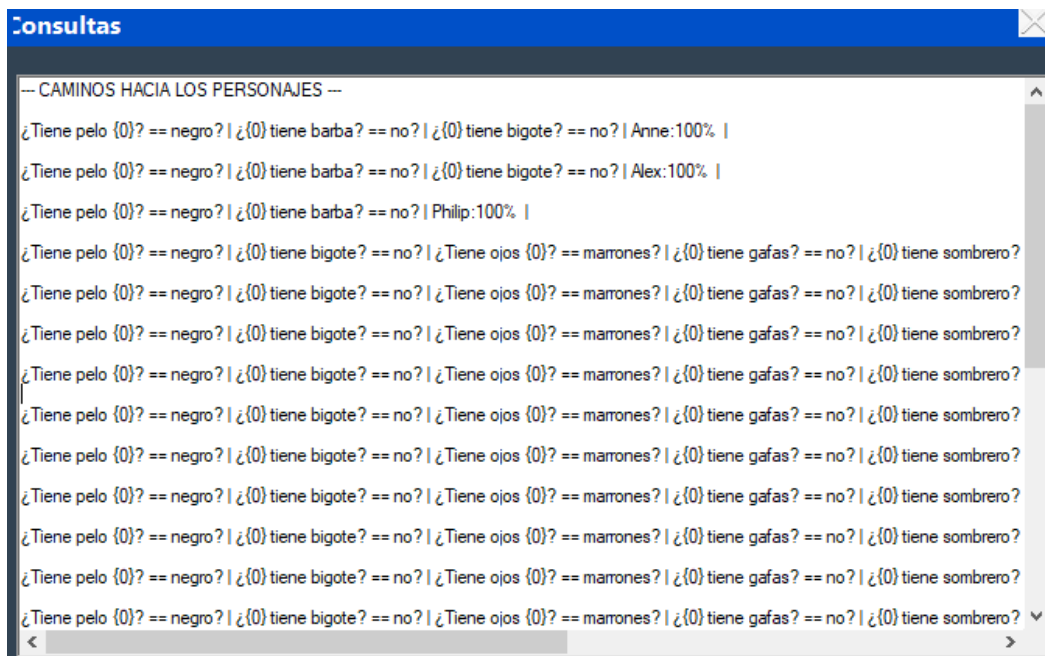
Por último, en cada iteración se pregunta si el dato es un dato hoja, ya que de ser así, se agrega un “espaciado” para diferenciar cada camino en la pantalla. Finalmente, se retorna el *texto* con todos los caminos.

```
//Consulta - 2
public String Consulta2(ArbolBinario<DecisionData> arbol)
{
    String texto = "--- CAMINOS HACIA LOS PERSONAJES --- \n\n";
    List<ArbolBinario<DecisionData>> nodos = new List<ArbolBinario<DecisionData>>();
    List<ArbolBinario<DecisionData>> caminos = new List<ArbolBinario<DecisionData>>();
    caminos = recorrido(nodos, caminos, arbol);
    foreach(var elem in caminos){
        texto = texto + elem.getDatoRaiz().ToString() + " | ";
        if(elem.esHoja()){
            texto = texto + "\n\n";
        }
    }

    return texto;
}

//Metodo Auxiliar
private List<ArbolBinario<DecisionData>> recorrido(List<ArbolBinario<DecisionData>> nodos, List<ArbolBinario<DecisionData>> caminos, ArbolBinario<DecisionData> arbol){
    nodos.Add(arbol);
    if(arbol.esHoja()){
        caminos.AddRange(nodos);
    }
    if(arbol.getHijoIzquierdo() != null){
        recorrido(nodos, caminos, arbol.getHijoIzquierdo());
        nodos.RemoveAt(nodos.Count - 1);
    }
    if(arbol.getHijoDerecho() != null){
        recorrido(nodos, caminos, arbol.getHijoDerecho());
        nodos.RemoveAt(nodos.Count - 1);
    }
    return caminos;
}
```

5. Método "Consulta2" y método auxiliar "Recorrido"



6. Consulta 2 en pantalla del juego

2.4 Consulta 3

Como último método, se busca mostrar la estructura del árbol binario dividido por niveles. Como venimos viendo, el método recibe un árbol binario de decisión como parámetro, y retorna un string con todos los caminos.

Para su implementación se utiliza una variación del “*Recorrido por Niveles*” de los arboles binarios. Por ende, primero se crean tres estructuras de datos:

1. Una *Cola*<> de tipo árbol binario de decisión, en la cual se van a ir encolando cada nodo del árbol iterativamente
2. Una variable *auxiliar* de tipo árbol binario de decisión, en la cual se va a ir desencolando los elementos de la cola.
3. Las variables de tipo string *texto* donde se van a ir guardando los datos tal como en los anteriores métodos, así como también la variable entera *nivel* la cual se incrementará cada vez que se pase de nivel en el árbol.

Con todo listo, lo primero en hacerse es encolar el nodo raíz del árbol, seguido de un *null* el cual será nuestro comodín divisor de niveles.

Tras esto, se entra a un ciclo *while* el cual iterara siempre y cuando la cola no se vacié. Adentro del mismo, se guarda en *auxiliar* el dato que se desencolo (un nodo árbol). Luego se pregunta si dicho dato desencolado es igual a *null* (comodín). En caso de que si, se verifica nuevamente si la cola se encuentra vacía, para evitar posibles *loops infinitos* debido al *null*. Si esto se cumple, hemos pasado de un nivel a otro. Por tal, concatenamos en *texto* el nivel en el que estamos, incrementamos la variable *nivel*, y volvemos a encolar un *null*, lo cual es una forma de “mover el comodín” dentro de la cola para ir dividiendo los niveles.

Por otra parte, si la variable *auxiliar* no es *null*, significa que es un nodo árbol. Por consiguiente, se procede a procesar dicho dato, convirtiéndolo a string con “*.ToString()*” y concatenándolo a *texto*. Luego de ello, se encola tanto el hijo izquierdo como el derecho del nodo.

Finalmente, cuando la cola se vacié (no haya más nodos por procesar), se retorna el *texto* con los datos divididos entre niveles.

```
//Consulta - 3
public String Consulta3(ArbolBinario<DecisionData> arbol)
{
    Cola<ArbolBinario<DecisionData>> c = new Cola<ArbolBinario<DecisionData>>();
    ArbolBinario<DecisionData> aux;
    String texto = "--- NIVELES DEL ARBOL--- \n\n Nivel 0: \n";
    int nivel = 1;

    c.encolar(arbol);
    c.encolar(null);

    while(!c.esVacia()){
        aux = c.desencolar();
        if(aux == null){
            if(!c.esVacia()){
                texto = texto + "Nivel " + nivel + ": \n";
                nivel++;
                c.encolar(null);
            }
        }
        else{
            DecisionData d = aux.getDatosRaiz();
            texto = texto + "\t " + d.ToString() + " \n ";

            if(aux.getHijoIzquierdo() != null){
                c.encolar(aux.getHijoIzquierdo());
            }

            if(aux.getHijoDerecho() != null){
                c.encolar(aux.getHijoDerecho());
            }
        }
    }
    return texto;
}
```

7. Método "Consulta3"

```
Consultas

-- NIVELES DEL ARBOL--

Nivel 0:
    ¿Tiene pelo {0}? == negro?
Nivel 1:
    ¿{0} tiene barba? == no?
    ¿{0} tiene bigote? == no?
Nivel 2:
    ¿{0} tiene bigote? == no?
    Philip:100%
    ¿Tiene ojos {0}? == marrones?
    ¿Tiene ojos {0}? == azules?
Nivel 3:
    Anne:100%
    Alex:100%
    ¿{0} tiene gafas? == no?
    ¿{0} tiene gafas? == si?
    Alfred:100%
    ¿{0} tiene barba? == no?
Nivel 4:
    ¿{0} tiene sombrero? == si?
    ¿{0} tiene sombrero? == si?
    Tom:100%
    ¿Tiene pelo {0}? == marrones?
    ¿Tiene pelo {0}? == marrones?
    Richard:100%
Nivel 5:
    ¿Tiene pelo {0}? == blanco?
```

8. Respuesta de la consulta en el juego

3. Problemas y Mejoras

A la hora de encargar el trabajo, el principal problema surgió con encontrarme tantas clases con métodos que no comprendo, haciendo que no sepa por donde comenzar. Luego me detuve a leer atentamente la consigna del proyecto y junto a una explicación en clase sobre donde debíamos trabajar; pude ubicar en qué lugar debía realizar mi implementación y comencé con ello.

El segundo inconveniente fue familiarizarme con cuál era el objetivo que cumplía cada clase de las que teníamos que usar. Tras prueba y error pude entenderlas y comenzar a trabajar.

Otro inconveniente que encontré fue no saber cómo concatenar los datos en un String para retornarlo, ya que como primera instancia mi intención era imprimirlos por consola. Tras una charla con el profe, pude solucionarlo de inmediato.

Cabo a los métodos, tuve un problema tanto en la Consulta2 como en la Consulta3. En la 2, la complicación se dio debido a que en la lista *camino*s se iban guardando los nodos "procesados" de forma temporal. Dicho error era generado ya que estaba haciendo una referencia de *nodos* a *camino*s. Una vez ubicado el error, fue cuestión de utilizar el *AddRange()* y problema resuelto.

Por parte de la Consulta3, el principal problema era lograr ubicar el *null* de forma que no se generaran ciclos, ni que los niveles quedaran mal implementados. Tras prueba, error, e intercambiar ideas con compañeros, logre solucionar tal problema.

En tanto a las mejoras, como idea se me ocurre que la inteligencia artificial vaya recolectando datos tanto de sus preguntas, como de las que yo hago, con el fin de que, en caso de que el usuario este por ganar, se arriesgue a elegir uno de los posibles personajes para poder ganar. Dicha implementación se me hace muy dificultosa para el nivel en el que me encuentro actualmente.

4. Conclusiones

Como conclusiones, obtenemos un trabajo interesante, el cual logro reforzar los conocimientos aprendidos en la cursada, más que nada sobre cómo funciona una estructura de tipo “Árbol Binario”.

Por otra parte, durante el desarrollo tuve que aprendiera a utilizar y entender código “ajeno” a mí a la hora de crear los métodos mediante otras clases, cosa que en el día a día del programador es mi habitual.

Como opinión final, creo que es un buen cierre a la materia, logrando reforzar los conceptos de estructuras de datos, así como también la idea de trabajar con otras personas y, sobre todo, otro código desconocido a uno.