ESTRUCTURA DE DATOS
SECCIÓN A

MANUAL TÉCNICO

PROYECTO FASE 1

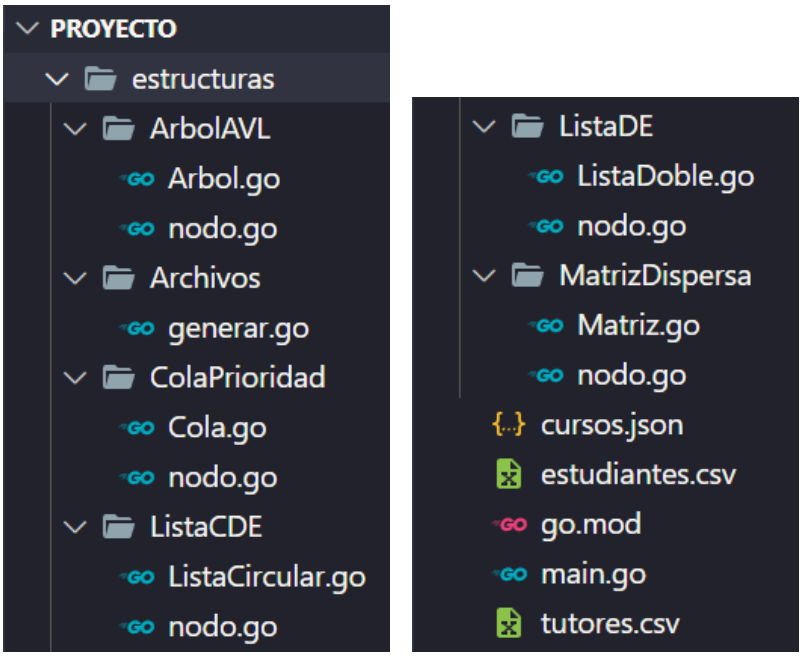
NOMBRE: ANA LUCIA FLETES ORDÓÑEZ
CARNÉ: 202010003

INTRODUCCIÓN

El siguiente manual técnico explica a detalle el funcionamiento del programa realizado en Go para la fase #1 correspondiente al Proyecto del curso de Estructura de Datos. Este consiste en la creación de una aplicación en consola que funcione como un portal para tutorías. En este podrán acceder el usuario administrador, que se encargará de cargar los datos de los tutores, estudiantes y cursos en una lista enlazada doble, una cola de prioridad y un árbol AVL correspondientemente, obteniendo la información de archivos '.csv' y '.json', igualmente podrá gestionar las solicitudes de los tutores existentes en la cola de prioridad y almacenar aquellos que han sido aceptados en una lista circular doblemente enlazada, y podrá generar reportes de las estructuras de datos utilizadas (elaborados con Graphviz); y los usuarios estudiantes, que tendrán la capacidad de visualizar a los tutores y asignarse cursos impartidos por los tutores disponibles, información que se guardará en una matriz dispersa.

ÍNDICE

ESTRUCTURA DEL PROYECTO



El proyecto consta de 5 carpetas correspondientes a las estructuras de datos a utilizar:

- ArbolAVL: para la estructura de datos “Árbol AVL”.
- ColaPrioridad: para la estructura de datos “Cola de Prioridad”.
- ListaCDE: para la estructura de datos “Lista Circular Doblemente Enlazada”.
- ListaDE: para la estructura de datos “Lista Doblemente Enlazada”.
- MatrizDispersa: para la estructura de datos “Matriz Dispersa”

Asimismo, cuenta con una carpeta denominada “Archivos”, donde se encuentra el archivo encargado de la generación de los reportes en ‘.dot’ y ‘.jpg’.

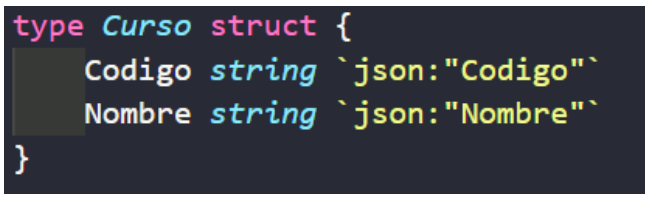
Y finalmente, tenemos el main y los archivos de prueba ‘.csv’ y ‘.json’.

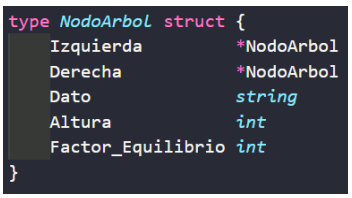
CARPETA ARBOLAVL

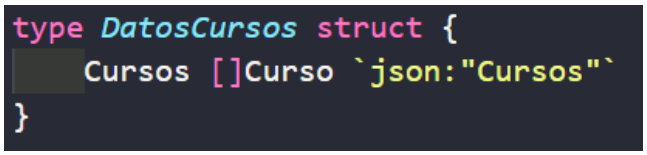
Contiene los archivos “nodo.go” y “Arbol.go” para la definición de la estructura de datos Árbol AVL.

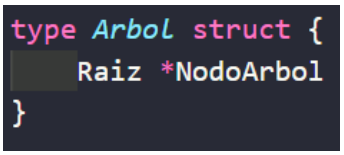
ATRIBUTOS

“nodo.go” posee 3 structs, uno para los atributos del curso (para guardar los valores del json), otro para el nodo del árbol y otro para un arreglo de cursos (para guardar los valores del json), y “Arbol.go” contiene un struct para el atributo del árbol AVL.









FUNCIONES

“Arbol.go” tiene definidas las siguientes funciones:

- **LeerJsonCursos(ruta string):** lee los datos del archivo de curso ‘.json’ de la ruta enviada y los agrega al árbol AVL.
- **InsertarCurso(dato string):** crea un nodo para almacenar el curso enviado y lo inserta en el árbol.
- **insertarNodo(raiz *NodoArbol, nuevoNodo *NodoArbol):** inserta un nodo, “nuevoNodo”, en el árbol AVL, tomando en cuenta el factor de desbalance del mismo, y si es necesario realizar rotaciones a la izquierda y/o derecha.
- **altura(raiz *NodoArbol):** retorna la altura del nodo enviado, que sería “raiz”.
- **equilibrio(raiz *NodoArbol):** determina el factor de equilibrio del nodo enviado “raiz” y retorna el dicho valor.

- **rotacionIzquierda(raiz *NodoArbol):** realiza la rotación a la izquierda, calcula nuevamente las alturas y factor de equilibrio, con el fin de balancear el árbol AVL.
- **rotacionDerecha(raiz *NodoArbol):** realiza la rotación a la derecha, calcula nuevamente las alturas y factor de equilibrio, con el fin de balancear el árbol AVL.
- **BuscarCurso(dato string):** busca si existe un curso determinado, es decir "dato", en el árbol AVL. Si existe, retorna true, caso contrario retorna false.
- **buscarNodo(dato string, raiz *NodoArbol):** busca un curso determinado, es decir "dato", dentro del árbol AVL. Si lo encuentra, retorna el nodo que lo contiene y de lo contrario retorna nil.
- **ReporteCursos():** genera el reporte en '.dot' y '.jpg' de los cursos existentes en el árbol AVL.
- **retornarValoresArbol(raiz *NodoArbol, indice int):** establece los nodos y sus conexiones, es un apoyo para la generación del reporte.

CARPETA ARCHIVOS

Contiene el archivo "generar.go" encargado de la generación de archivos '.dot' y '.jpg'.

FUNCIONES

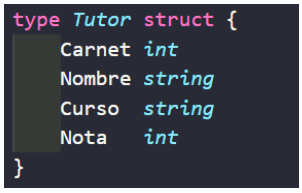
- **CrearArchivo(nombre_archivo string):** crea un archivo con el nombre enviado en el parámetro 'nombre_archivo' y extensión '.dot'.
- **EscribirArchivo(contenido string, nombre_archivo string):** escribe la cadena enviada en el parámetro 'contenido' en el archivo con nombre 'nombre_archivo'.
- **Ejecutar(nombre_imagen string, archivo string):** genera la imagen '.jpg' con nombre 'nombre_imagen' de acuerdo al archivo '.dot'

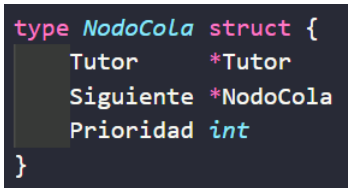
CARPETA COLA PRIORIDAD

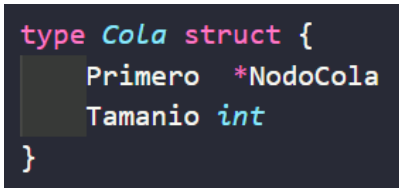
Contiene los archivos "nodo.go" y "Cola.go" para la definición de la estructura de datos Cola de Prioridad.

ATRIBUTOS

"nodo.go" posee 2 structs, uno para los atributos del tutor y otro para el nodo de la cola, y "Cola.go" contiene un struct para los atributos de la cola de prioridad.







FUNCIONES

"Cola.go" tiene definidas las siguientes funciones:

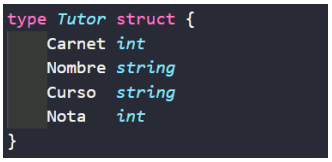
- **LeerCSVTutores(ruta string):** lee los datos del archivo de tutores '.csv' de la ruta enviada y los agrega a la cola.
- **Encolar(carnet int, nombre string, curso string, nota int):** crea al tutor con los datos de carnet, nombre, curso y nota enviados, define su prioridad y los inserta en la cola según corresponda.
- **Mostrar_Primero_Cola():** muestra los datos del primer tutor de la cola y el carnet del siguiente tutor en la cola.
- **Descolar():** saca al primer tutor de la cola.

CARPETA LISTACDE

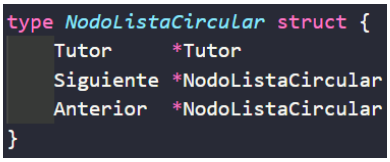
Contiene los archivos "nodo.go" y "ListaCircular.go" para la definición de la estructura de datos Lista Circular Doblemente Enlazada.

ATRIBUTOS

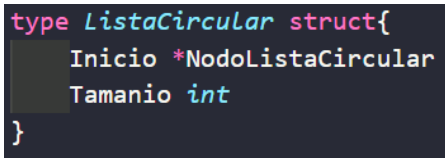
"nodo.go" posee 2 structs, uno para los atributos del tutor y otro para el nodo de la lista circular doble, y "ListaCircular.go" contiene un struct para los atributos de la lista circular doblemente enlazada.



```
type Tutor struct {
    Carnet int
    Nombre string
    Curso  string
    Nota   int
}
```



```
type NodoListaCircular struct {
    Tutor     *Tutor
    Siguiente *NodoListaCircular
    Anterior  *NodoListaCircular
}
```



```
type ListaCircular struct{
    Inicio *NodoListaCircular
    Tamanio int
}
```

FUNCIONES

"ListaCircular.go" tiene definidas las siguientes funciones:

- **InsertarTutor(carnet int, nombre string, curso string, nota int):** crea al tutor con los datos de carnet, nombre, curso y nota enviados, y lo inserta en la lista circular doblemente enlazada en orden según el número de carnet.
- **MostrarTutores():** muestra los datos de curso y nombre de los tutores presentes en la lista circular doble.
- **BuscarTutor(curso string):** busca al tutor del curso enviado y retorna el nodo correspondiente a este, o nil si no lo encuentra.
- **BuscarCurso(curso string):** busca el curso enviado en la lista, retornando true si lo encuentra o false en el caso contrario.
- **ReporteTutores():** genera el reporte en '.dot' y '.jpg' de los tutores existentes en la lista circular doblemente enlazada.

CARPETA LISTADE

Contiene los archivos "nodo.go" y "ListaDoble.go" para la definición de la estructura de datos Lista Doblemente Enlazada.

ATRIBUTOS

"nodo.go" posee 2 structs, uno para los atributos del estudiante y otro para el nodo de la lista doble, y "ListaDoble.go" contiene un struct para los atributos de la lista doblemente enlazada.

```
type Estudiante struct {
    Carnet int
    Nombre string
}
```

```
type NodoListaDoble struct {
    Estudiante *Estudiante
    Siguiente  *NodoListaDoble
    Anterior   *NodoListaDoble
}
```

```
type ListaDoble struct {
    Inicio  *NodoListaDoble
    Tamanio int
}
```

FUNCIONES

"ListaDoble.go" tiene definidas las siguientes funciones:

- **LeerCSVEstudiantes(ruta string):** lee los datos del archivo de estudiantes '.csv' de la ruta enviada y los agrega a la lista doblemente enlazada.
- **InsertarEstudiante(carnet int, nombre string):** crea al estudiante con los datos carnet y nombre enviados, y lo inserta en la lista doblemente enlazada.
- **BuscarEstudiante(carnet string, contraseña string):** busca al estudiante con el número de carnet enviado, compara que carnet sea igual a contasena y retorna true si lo encuentra y carnet es igual a contrasena, de lo contrario, retorna false.
- **ReporteEstudiantes():** genera el reporte en '.dot' y '.jpg' de los estudiantes existentes en la lista doblemente enlazada.

CARPETA MATRIZDISPERSA

Contiene los archivos "nodo.go" y "Matriz.go" para la definición de la estructura de datos Matriz Dispersa.

ATRIBUTOS

"nodo.go" posee 2 structs, uno para los atributos del dato a almacenar (que son los carnets tanto del estudiante, como del tutor, y el curso) y otro para el nodo de la matriz dispersa, y "Matriz.go" contiene un struct para los atributos de la matriz dispersa.

```
type Dato struct {
    Carnet_Tutor       int
    Carnet_Estudiante  int
    Curso              string
}
```

```
type NodoMatriz struct {
    Siguiente  *NodoMatriz
    Anterior   *NodoMatriz
    Abajo      *NodoMatriz
    Arriba     *NodoMatriz
    PosX       int
    PosY       int
    Dato       *Dato
}
```

```
type Matriz struct {
    Raiz                 *NodoMatriz
    Cantidad_Estudiantes int
    Cantidad_Tutores     int
}
```

FUNCIONES

"Matriz.go" tiene definidas las siguientes funciones:

- **InsertarElemento(carnet_estudiante int, carnet_tutor int, curso string):** inserta el elemento correspondiente a la asignación en las fila y columna determinadas por carnet_estudiante y carnet_tutor de la matriz dispersa.
- **buscarColumna(carnet_tutor int, curso string):** recorre el encabezado de las columnas desde la raíz hacia la derecha, busca un nodo con carnet_tutor y curso iguales a los enviados como parámetros y retorna el nodo con datos iguales si es encontrado, sino retorna nil.
- **buscarFila(carnet_estudiante int):** recorre el encabezado de las filas desde la raíz hacia abajo, busca un nodo con carnet_estudiante igual al enviado como parámetro y retorna el nodo con dato igual si es encontrado, sino retorna nil.
- **nuevaColumna(x int, carnet_tutor int, curso string):** crea el nodo, determinando la posición en "x" como el parámetro "x" recibido y la posición en "y" como -1 por formar parte del encabezado de las columnas. Luego inserta el nodo en la columna de la matriz y se retorna el mismo nodo.
- **nuevaFila(y int, carnet_estudiante int, curso string):** crea el nodo, determinando la posición en "y" como el parámetro "y" recibido y la posición en "x" como -1 por formar parte del encabezado de las filas. Luego inserta el nodo en la fila de la matriz y se retorna el mismo nodo.
- **insertarColumna(nuevoNodo *NodoMatriz, nodoRaiz *NodoMatriz):** inserta un nodo en la matriz dispersa, asignando sus apuntadores para las columnas, es decir, "Siguiente" y "Anterior".
- **insertarFila(nuevoNodo *NodoMatriz, nodoRaiz *NodoMatriz):** inserta un nodo en la matriz dispersa, asignando sus apuntadores para las filas, es decir, "Arriba" y "Abajo".

MAIN

FUNCIONES

"main.go" tiene definidas las siguientes funciones:

- **Login():** recibe los datos de usuario y contraseña del usuario, comprueba si las credenciales pertenecen a un usuario administrador o estudiante, y lo dirige al menú correspondiente.
- **MenuAdministrador():** muestra las opciones disponibles para el administrador y recibe el valor correspondiente a la opción seleccionada para ejecutarla.
- **CargarTutores():** requiere el ingreso de la ruta donde se encuentra el archivo '.csv' con la información de los tutores para su futura lectura.
- **CargarEstudiantes():** requiere el ingreso de la ruta donde se encuentra el archivo '.csv' con la información de los estudiantes para su próxima lectura.
- **CargarCursos():** requiere el ingreso de la ruta donde se encuentra el archivo '.json' con la información de los cursos para a continuación leerlo.
- **ControlTutores():** muestra la información de los tutores en la cola de prioridad, permitiendo el ingreso de una opción, ya sea para aceptar o rechazar al tutor.
- **Reportes():** presenta un menú con los distintos reportes existentes, permitiendo que el usuario ingrese la opción correspondiente al reporte a generar.
- **MenuEstudiantes():** muestra las opciones disponibles para el estudiante y recibe el valor correspondiente a la opción seleccionada para ejecutarla.
- **AsignarCurso():** recibe el código del curso proporcionado por el usuario, manda a buscar este al árbol AVL y a la lista circular doblemente enlazada para verificar que el curso existe y tiene tutores disponibles para asignarse.