PROYECTO EDD

FASE 1

Manual de Técnico

Oswaldo Antonio Choc Cuters
Estructuras de datos
Sección A
201901844

Descripción

Se realizo mediante Visual Studio Code (IDE) utilizando Golang como único lenguaje de programación se realizó un programa que aplicó el uso de estructuras para poder organizar de mejor forma la información ingresada desde la consola y así mismo ir accediendo a los métodos y funciones de cada estructura para poder trabajar de una forma más ordenada.

Es importante tener en cuenta que en el presente manual se hace mención del funcionamiento de cada clase, objeto o variable y va dirigido para cualquier persona con cierto conocimiento básico de programación.

Objetivos

Objetivos generales:

- Explicar el proceso por el cual se diseñó el programa.
- Rectificar el proceso realizado.

Objetivos específicos:

- Describir las herramientas utilizadas para el diseño y desarrollo del programa.
- Definir cada clase, método y funciones que presenta el programa.
- Definir la lógica que se utilizó para la realización del programa.
- Explicar los conceptos de estructuras de datos que se implementaron

Especificación técnica:

Se solicitó a la creación como estudiante de estructuras de datos, una aplicación para llevar control de todos los estudiantes tutores como los que desean llevar las tutorías.

Lógica del programa:

Tomando él cuenta el tema de estructura de datos se implementó varias de ellas para almacenar la información ingresada.

Estructuras utilizadas:

- **Lista Enlazada Doble:** Registro de estudiantes que buscan tutoría.
- **Lista Circular Doble Enlazada:** Estudiantes disponibles como tutores.
- **Cola de Prioridad:** Gestión de solicitudes de tutoría, priorización y asignación.
- **Matriz dispersa:** Representación de la asignación entre estudiantes y tutores.
- **Árbol AVL:** Organización de cursos que existen dentro de la escuela de sistemas.

Menú principal

En el menú principal se valido para que ingresen los alumnos o el administrador. Tomando en cuenta que el administrador entra con las siguientes credenciales.

Administrador:

- Usuario: ADMIN_201910844
- Contraseña: admin

Estudiantes:

- Usuario: #Carnet
- Contraseña: #Carnet

Una vez que ingrese el administrador se tiene que mostrar el siguiente menú tomando como formato el siguiente:

```go
func MenuAdmin() {
    limpiar()
    opcion := 0
    salir := false
    for !salir {

        fmt.Println("╔═══════════════════════════════════╗")
        fmt.Println("║ 1. Carga de Estudiantes Tutores ║")
        fmt.Println("║ 2. Carga de Estudiantes          ║")
        fmt.Println("║ 3. Cargar de Cursos              ║")
        fmt.Println("║ 4. Control de Estudiantes        ║")
        fmt.Println("║ 5. Reportes                      ║")
        fmt.Println("║ 6. Salir                         ║")
        fmt.Println("╚═══════════════════════════════════╝")
        fmt.Scanln(&opcion)
        switch opcion {
        case 1:
            CargaTutores()
        case 2:
            CargaEstudiantes()
        case 3:
            CargaCursos()
        case 4:
            ControlEstudiantes()
            limpiar()
        case 5:
            generarReportes()

        case 6:
            salir = true
            limpiar()
        }
```

En este menú se desarrollo cada funcionalidad que puede realizar el administrador.

1) Carga de estudiantes tutores:

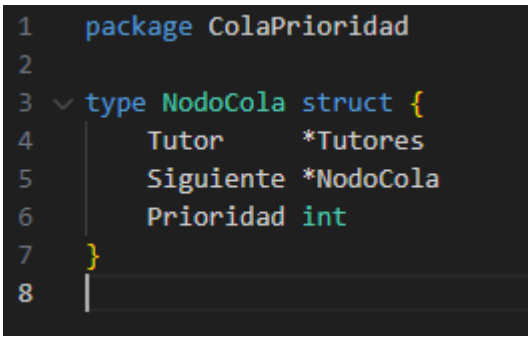
Como se indico anteriormente la carga de tutores se realizo mediante un archivo (.csv) el archivo de tutores académicos tendrá 4 parámetros, carnet, nombre, curso, y nota, para este caso, primero debe ir a la cola

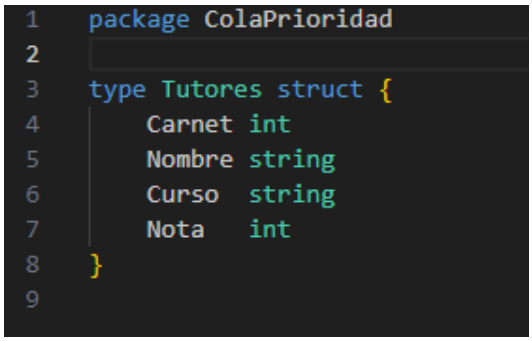
de prioridad, para ello se debe analizar la nota por lo que se tendrá 4 prioridades, descritas a continuación:

Prioridad 1: Alumnos tutores que tengan una nota entre 90-100
Prioridad 2: Alumnos tutores que tengan una nota entre 75-89
Prioridad 3: Alumnos tutores que tengan una nota entre 65-74
Prioridad 4: Alumnos tutores que tengan una nota entre 64-61

Se tomará como prioridad más alta la 1 y la más baja la 4. Aquellos que tengan una nota menor de 61, serán automáticamente eliminados y no se tomarán en cuenta.

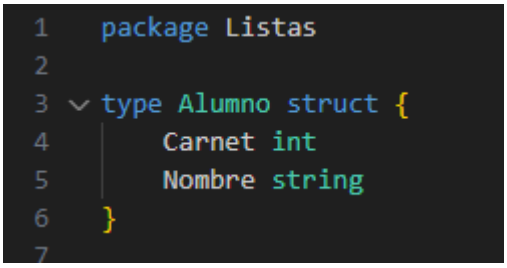
Una vez validado toda esa información se desarrolló el método para leer el archivo (.csv) y cada información se manda a ingresar en la cola de prioridad que tiene las siguientes estructuras según lo pedido:

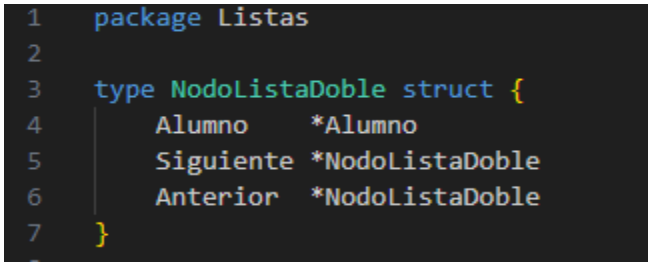




2) Carga de estudiantes:

Como se indicó anteriormente la carga de estudiantes se realizó mediante un archivo (.csv) el archivo de alumnos tendrá sólo 2 parámetros, carnet y nombre, con la siguiente estructura, para este caso se debe guardar directamente a la Lista Doblemente Enlazada la cual tiene la siguiente estructura.





3) Carga de cursos:

Como se indicó anteriormente la carga de cursos es mediante la estructura de árbol AVL donde se deberá ingresar desde un archivo (.json) el archivo json tendrá un arreglo llamado Cursos y cada posición de dicho arreglo, tendrá 2 atributos más siendo Código y Nombre. Toda esta información se irá guardando en el árbol AVL, tomando como parámetro de balanceo el código del curso, el árbol AVL tiene las siguientes estructuras:

```go
package ArbolAVL

type NodoArbol struct {
    Izquierdo         *NodoArbol
    Derecho           *NodoArbol
    Valor             string
    Altura            int
    Factor_Equilibrio int
}

```

```go
type ArbolAVL struct {
    Raiz *NodoArbol
}

/****************************************/
type Curso struct {
    Codigo string `json:"Codigo"`
    Nombre string `json:"Nombre"`
}

type DatosCursos struct {
    Cursos []Curso `json:"Cursos"`
}
```

Tomando en cuenta la estructura de un árbol AVL se debe de tener en cuenta el tema de balanceo y las categorías básicas de un árbol binario de búsqueda ABB lo cual se crearon las funciones de inserción y balanceo según corresponda tomando en cuenta el código del curso como guía de los nodos y así mismo tener una referencia para poder ordenarlos correctamente, con los valores menores a la raíz a la izquierda y los valores mayores a la derecha de la raíz.

4) Control de estudiantes:

Para el control de estudiantes debemos tener en cuenta la información almacenada en la cola de prioridad que se tuvo que haber creado a la hora de ingresar los estudiantes tutores y haberse almacenado en la dicha cola, una vez ingresado al control de estudiantes el administrador tiene la opción de poder aceptar o rechazar a los tutores que fueron ingresados por lo tanto se desarrolló el siguiente menú.

```
func ControlEstudiantes() {
    limpiar()
    opcion := 0
    salir := false

    for !salir {

        colaPrioridad.Primero_Cola()
        fmt.Println("    1. Aceptar")
        fmt.Println("    2. Rechazar")
        fmt.Println("    3. Salir")
        fmt.Scanln(&opcion)

        if opcion == 1 {
            if colaPrioridad.Longitud != 0 {
                limpiar()
                listaDobleCircular.Agregar(colaPrioridad.Primero.Tutor.Carnet, colaPrioridad.Primero.Tutor.Nombre, cc
                colaPrioridad.Descolar()
            }
        } else if opcion == 2 {
            colaPrioridad.Descolar()
        } else if opcion == 3 {
            salir = true
        } else {
            fmt.Println("Opcion invalida")
        }
    }
}
```

Al momento de aceptar al tutor se almacena en la lista circular doblemente enlazada por lo tanto se eliminar de la cola de prioridad y se almacena en dicha lista, así mismo si se elimina siempre se elimina de la cola de prioridad.

En este menú se validó de que existirá ya un tutor aceptado para el mismo curso, la lógica para poder seguir ese caso es de poder ver el estudiante tutor con mayor nota ese será aceptado y el otro deberá de eliminarse de la lista circular doblemente enlazada, es importante el orden en esta lista ya que, así como la cola de prioridad se daba más énfasis según la nota en este caso se ordena por el número de carnet.

5) Reportes:

En el área de reportes se debe de poder graficar la información almacenada en cada estructura de datos que se utilizó. Para eso se utilizaron además de las funciones principales en cada estructura de datos los siguientes funciones para la creación de la graficas a través de Graphviz.

```go
func CrearArchivo(nombre_archivo string, nombreReporte string) {
    var _, err = os.Stat(nombre_archivo)

    if os.IsNotExist(err) {
        var file, err = os.Create(nombre_archivo)
        if err != nil {
            return
        }
        defer file.Close()
    }
    fmt.Println("Reporte " + nombreReporte + " generado exitosamente")
}
```

```go
func EscribirArchivo(contenido string, nombre_archivo string) {
    var file, err = os.OpenFile(nombre_archivo, os.O_RDWR, 0644)
    if err != nil {
        return
    }
    defer file.Close()
    _, err = file.WriteString(contenido)
    if err != nil {
        return
    }
    err = file.Sync()
    if err != nil {
        return
    }
    //fmt.Println("-► -► Archivo generado exitosamente")
}
```

```go
func Ejecutar(nombre_imagen string, archivo string) {
    path, _ := exec.LookPath("dot")
    cmd, _ := exec.Command(path, "-Tjpg", archivo).Output()
    mode := 0777
    _ = os.WriteFile(nombre_imagen, cmd, os.FileMode(mode))
    //dot lista.dot -Tjpg lista.jpg -o
}
```

Inicio de sesión Estudiante

Al inicial el estudiante con las credenciales correctas, se diseñó el siguiente menú.

```
func MenuEstudiantes() {
    limpiar()
    opcion := 0
    salir := false
    for !salir {
        fmt.Println("\n1. Ver Tutores Disponibles")
        fmt.Println("2. Asignarse Tutores")
        fmt.Println("3. Salir")
        fmt.Scanln(&opcion)
        switch opcion {
        case 1:
            fmt.Print("\033[H\033[2J")
            listaDobleCircular.Imprimir()
        case 2:
            fmt.Print("\033[H\033[2J")
            AsignarCurso()
        case 3:
            fmt.Print("\033[H\033[2J")
            salir = true
        }
    }
}
```

1) Ver tutores disponibles.

En esta opción no es más que recorrer e imprimir los tutores que fueron aceptados por el administrador para dar los cursos correspondientes.

```
func (c *ListaDobleCircular) Imprimir() {
    aux := c.Inicio
    contador := 1
    for contador <= c.Longitud {
        fmt.Println("-----------------------------------------------")
        fmt.Println(aux.Tutor.Curso, " -> ", aux.Tutor.Nombre)
        fmt.Println("-----------------------------------------------")
        aux = aux.Siguiente
        contador++
    }
}
```

2) Asignarse a cursos.

En esta opción si se tuvieron que validar varias posibles casos.

- Que el cursos que desea asignarse no exista.
- Que el cursos no tenga tutor
- Que el curso exista y tenga tutor

Para almacenar todas las asignaciones se implemento la matriz dispersa la cual como cabeceras tiene los carnets de los tutores y en las filas los carnets de los alumnos que se asignaron a los cursos.

En la primera situación donde el estudiante deseé asignarse a un curso que no exista ser realizo la búsqueda en el árbol AVL para ver si el curso existe o no eso es lo principal.

En la segunda situación donde el curso no tenga tutor dependerá si en al lista doblemente circular aparece o no un tutor aceptado por el administrador, si en todo caso no hay ningún tutor no se podrá continuar con la asignación.

En la ultima situación donde si exista el curso y si exista el tutor se deberá de almacenar la relación en la matriz dispersa.