

Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Estructuras de Datos

Manual de Técnico:
Proyecto 1

Hecho por:
Walter Daniel Jiménez Hernández
201901108
Fecha: 15/12/2022

Contenido

1. Introducción
2. Objetivos
3. Conceptos Previos
4. Especificación Técnica
 - a. Requisitos de Hardware
 - b. Requisitos de Software
 - i. Sistema Operativo
 - ii. Lenguaje de Programación
 - iii. Tecnologías usadas
5. Lógica del Programa
 - a. Clases Utilizadas
 - i. Usuario
 - ii. Artista
 - iii. Canción
 - iv. Programada
 - v. Podcast
 - b. Estructuras Usadas
 - i. Lista Simple
 - ii. Pila
 - iii. Cola
 - iv. Lista Doblemente Enlazada
 - v. Lista Circular
 - vi. Lista Circular Doblemente Enlazada
 - vii. Lista de Listas Dobles
 - viii. Matriz Dispersa
 - ix. Árbol Binario de Búsqueda
 - c. Funciones Utilizadas
 - i. Login
 - ii. Signin
 - iii. Carga Masiva
 - iv. Mostrar Usuarios
 - v. Mostrar Canciones
6. Link del Repositorio

Introducción

El programa cumple con la función de implementar diferentes estructuras de datos para poder organizar música de manera eficiente. Se pueden registrar usuarios, hacer carga masiva de los datos, visualizar el estado de las estructuras, tener amigos, bloquear usuarios y publicar canciones individualmente. Además el uso de encriptación para el manejo de las contraseñas.

Objetivo

El objetivo es poder aplicar el conocimiento sobre estructuras de datos para hacer de manera eficiente el manejo de los datos.

- Implementar estructuras lineales y no lineales.
- Utilizar herramientas de graficación para visualizar las estructuras.
- Utilizar algoritmos de ordenamiento.

Conocimientos Previos

- Programación en JavaScript
- Programación Orientada a Objetos
- Estructuras lineales
- HTML
- CSS
- Árboles Binarios
- Graphviz
- Encriptación Sha256
- Ordenamiento burbuja y Quicksort

Especificaciones Técnicas

Requisitos de Hardware

- Procesador Dual Core o Superior
- Mínimo 2 GB de RAM

Requisitos de Software

- Sistema Operativo
 - Windows 7
 - Windows 8
 - Windows 10
 - Windows 11
 - Distribuciones de Linux
 - MAC OS
- Navegador
 - Chrome
 - Edge
 - Mozilla Firefox
 - Safari
 - Opera
 - Brave
- Lenguaje de Programación
 - JavaScript
- Tecnologías Usadas
 - Visual Studio Code
 - Git
 - Github
 - Tailwind
 - SweetAlert
 - Graphviz
 - Github Pages

Lógica del Programa

Clases Utilizadas

- **Usuario:** Es un objeto creado para poder guardar la información de los usuarios de la aplicación así como los amigos que se agregan y los usuarios que se bloquean. Los atributos que tiene son los siguientes:

```
export class Usuario {  
  constructor(_dpi, _name, _username, _password, _phone, _admin) {  
    this.dpi = _dpi  
    this.name = _name  
    this.username = _username  
    this.password = _password  
    this.phone = _phone  
    this.admin = _admin  
    this.friends = new Pila()  
    this.bloqueados = new Cola()  
    this.playList = new ListaCircularDoble()  
  }  
}
```

- **Artista:** Es un objeto creado para poder guardar la información de los artistas de la aplicación. Los atributos que tiene son los siguientes:

```
export class Artista {  
  constructor(_name, _age, _country) {  
    this.name = _name  
    this.country = _country  
    this.age = _age  
  }  
}
```

- **Canción:** Es un objeto creado para poder guardar la información de las canciones de la aplicación. Los atributos que tiene son los siguientes:

```
export class Cancion {  
  constructor(_artista, _nombre, _duracion, _genero){  
    this.nombre = _nombre  
    this.artista = _artista  
    this.duracion = _duracion  
    this.genero = _genero  
  }  
}
```

- **Programada:** Es un objeto creado para poder guardar la información de las canciones que se programan para publicarse en una fecha específica. Los atributos que tiene son los siguientes:


```
export class Programada{  
  constructor(_month, _day, _song, _artist){  
    this.month = _month  
    this.day = _day  
    this.song = _song  
    this.artist = _artist  
  }  
}
```

- **Podcast:** Es un objeto creado para poder guardar la información de los podcasts de la aplicación. Los atributos que tiene son los siguientes:

```
export class Podcast {  
  constructor(_name, _topic, _guests, _duration){  
    this.name = _name  
    this.topic = _topic  
    this.guests = _guests  
    this.duration = _duration  
  }  
}
```

Estructuras Utilizadas

- **Lista Simple:** Esta lista contiene un único apuntador al siguiente elemento de la lista, en la aplicación se utiliza para poder almacenar los usuarios.



```
export class ListaSimple{

    constructor(){
        this.first = null
    }

    insertarInicio(_dato){
        let newNode = new NodoSimple(_dato)
        newNode.next = this.first
        this.first = newNode
    }
}
```

- **Pila:** Esta lista contiene un único apuntador al siguiente elemento de la lista y se pueden agregar y sacar elementos únicamente por un lado de la lista, en la aplicación se utiliza para poder almacenar los usuarios agregados como amigos.



```
export class Pila{

    constructor(){
        this.first = null
    }

    push(_dato){
        let newNode = new NodoSimple(_dato)
        newNode.next = this.first
        this.first = newNode
    }

    pop(){
        if(this.first != null){
            this.first = this.first.next
        }
    }
}
```

- **Cola:** Esta lista contiene un único apuntador al siguiente elemento de la lista y se pueden agregar únicamente por un lado de la lista y sacar solamente por el otro lado, en la aplicación se utiliza para poder almacenar los usuarios bloqueados.



```
export class Cola{

    constructor(){
        this.first = null
        this.tail = null
    }

    enqueue(_dato){
        let newNode = new NodoDoble(_dato)

        if(this.first == null){
            this.first = newNode
            this.tail = this.first
        }else{
            newNode.prev = this.tail
            this.tail.next = newNode
            this.tail = this.tail.next
        }
    }

    dequeue(){
        if(this.first != null){
            this.first = this.first.next
            this.first.prev = null
        }
    }
}
```

- **Lista Doblemente Enlazada:** Esta lista contiene dos apuntadores, uno apunta al siguiente elemento de la lista y el otro apunta al elemento anterior, en la aplicación se utiliza para poder almacenar las canciones de los usuarios en la lista de listas.

```
export class ListaDoble{


    constructor(){
        this.first = null
        this.tail = null
    }

    insertarInicio(_dato){
        let newNode = new NodoDoble(_dato)

        if(this.first == null){
            this.first = newNode
            this.tail = this.first
        }else{
            newNode.next = this.first
            this.first = newNode
            this.first.next.prev = this.first
        }
    }
}
```

- **Lista Circular Doblemente Enlazada:** Esta lista contiene dos apuntadores, uno apunta al siguiente elemento de la lista y el otro apunta al elemento anterior, además que el primer elemento apunta hacia el último y el último hacia el

primero, en la aplicación se utiliza para poder almacenar las canciones en la playlist.



```
export class ListaCircularDoble {
  constructor() {
    this.first = null
    this.tail = null
  }

  insertarInicio(_dato) {
    let newNode = new NodoDoble(_dato)

    if (this.first == null) {
      this.first = newNode
      this.tail = this.first
      this.first.prev = this.tail
      this.tail.next = this.first
    } else {
      newNode.next = this.first
      newNode.prev = this.tail
      this.first.prev = newNode
      this.first = newNode
      this.tail.next = this.first
    }
  }
}
```

- **Lista de listas dobles:** Esta lista contiene dos apuntadores, uno apunta al siguiente elemento de la lista y el otro apunta al elemento anterior, además que en cada nodo se encuentra otra lista la cual también tiene apuntadores dobles formando una lista dentro de otra, en la aplicación se utiliza para poder almacenar los artistas y sus canciones.

```

export class ListaListas{

  constructor(){
    this.first = null
  }

  insertarCabecera(_dato){
    let nuevoNodo = new NodoListas(_dato)

    if(this.first == null){
      this.first = nuevoNodo
      this.tail = this.first
    }else{
      let tmp = this.first
      while(tmp != null){
        if(tmp.dato.name == _dato.name){
          return false
        }
        tmp = tmp.next
      }
      nuevoNodo.next = this.first
      this.first = nuevoNodo
      this.first.next.prev = this.first
    }
  }

  insertarValor(_cabecera, _dato){
    let tmp = this.first

    while(tmp != null){
      if(tmp.dato.name == _cabecera){
        tmp.lista.insertarFinal(_dato)
        break
      }
      tmp = tmp.next
    }
  }
}

```

- **Matriz dispersa:** Esta lista contiene dos apuntadores, uno apunta al siguiente elemento de la lista y el otro apunta al elemento anterior, contiene una sección de cabeceras la cual es una lista de 2 dimensiones apuntando al siguiente elemento, los valores están compuestos de nodos de 4 direcciones los cuales

apuntan hacia arriba, abajo, izquierda y derecha para poder conectarse con los demás de la fila y columna, en la aplicación se utiliza para poder almacenar las canciones programadas teniendo como columnas los días y como filas los meses del año.



```
export class MatrizDispersa{
  constructor() {
    this.colsList = new Header();
    this.rowsList = new Header();
  }

  obtener(x, y){

    let columna = this.colsList.getHeader(y);
    let row = this.rowsList.getHeader(x);

    if(columna != null && row != null){
      let aux = columna.access

      while(aux != null){
        if(aux.x == x && aux.y == y){
          return aux.dato
        }
        aux = aux.down
      }
    }
    return null
  }
}
```

- **Árbol Binario de Búsqueda:** Esta estructura ya no es lineal, cuenta con una raíz y esta con 2 nodos hijos, así se va ramificando con cada nodo pudiendo 2 hijos cada uno, en la aplicación se utiliza para poder almacenar los podcasts.

```
export class ArbolBB{
  constructor(){
    this.root = null
  }

  insertar(_dato){
    this.root = this.insertarRecursivo(this.root, _dato)
  }

  insertarRecursivo(_root, _dato){
    if(_root == null){
      _root = new NodoArbolBB(_dato)
    }else if(_root.dato.name == _dato.name){
      _root.dato = _dato
    }else if(_root.dato.name < _dato.name){
      _root.right = this.insertarRecursivo(_root.right, _dato)
    }else{
      _root.left = this.insertarRecursivo(_root.left, _dato)
    }
    return _root
  }
}
```

Funciones Utilizadas

- **Login:** Se obtienen los datos del formulario de login y se utiliza la función de buscarUsuario de la lista simple a esta se le envía un username y una contraseña encriptada y si hay alguna coincidencia devuelve el objeto y se muestra la página del login.

```

export function login(e) {
  const username = document.getElementById("usernamelogin")
  const password = document.getElementById("passwordlogin")
  const admin = document.getElementById("adminlogin")

  usuarioActual = usuarios.buscarUser(username.value, sha256(password.value), admin.checked)

  if (usuarioActual != null) {
    hideLogin()
    if (usuarioActual.dato.admin) {
      showAdmin()
    } else {
      showUser()
    }
  } else {
    Swal.fire('Oops...', 'Usuario o contraseña incorrectos', 'warning')
  }

  e.preventDefault()

  username.value = ""
  password.value = ""
  admin.checked = false
}

```

- **Singin:** Se obtienen los datos del formulario de registro y se utiliza la función de insertarFinal de la lista simple y si todo salió se muestra una alerta de que el usuario fue registrado.

```

export function signin(e) {
  const username = document.getElementById("usernamesignin")
  const password = document.getElementById("passwordsignin")
  const name = document.getElementById("namesignin")
  const dpi = document.getElementById("dpisignin")
  const tel = document.getElementById("telsignin")

  if (usuarios.insertarFinal(new Usuario(dpi.value, name.value, username.value, sha256(password.value), tel.value, false))) {
    Swal.fire('Perfecto...', 'Usuario registrado', 'success')
  } else {
    Swal.fire('Oops...', 'Dpi o nombre de usuario ya registrados', 'error')
  }

  e.preventDefault();

  username.value = ""
  password.value = ""
  name.value = ""
  dpi.value = ""
  tel.value = ""
}

```

- **Carga Masiva:** En cada una de las cargas se muestra una alerta con un input tipo file para que se pueda cargar el archivo json, al momento de cargarse este

se pasa por un FileReader y luego se recorren todos los elementos que se leyeron y se agregan a la estructura correspondiente.

```
export function ingresarUsuarios() {  
  
  Swal.fire({  
    title: 'Carga masiva de usuarios',  
    html: '<input type="file" id="fileUser" class="swal2-input">',  
    confirmButtonText: 'Cargar',  
    focusConfirm: false,  
    preConfirm: () => {  
      const fileuser = Swal.getPopup().querySelector('#fileUser').files[0]  
      return fileuser  
    }  
  }).then((result) => {  
    const reader = new FileReader()  
  
    reader.addEventListener("load", () => {  
      let datos = JSON.parse(reader.result)  
      datos.forEach(user => {  
        usuarios.insertarFinal(new Usuario(user.dpi, user.name, user.username, sha256(user.password), user.phone, user.admin))  
        Swal.fire("Registrados...", 'Carga masiva realizada', 'success')  
      });  
    })  
  
    reader.readAsText(result.value)  
  })  
}
```

- **Mostrar Usuario:** Se recorre la lista de usuarios y se van agregando uno por uno al html para que puedan ser vistos por el usuario y los pueda agregar como amigos o bloquearlos.

```

let usersList = document.getElementById("usersListFriends")
let friendsList = document.getElementById("friendsListFriends")
usersList.innerHTML = ""
friendsList.innerHTML = ""
for (let index = 0; index < usuarios.size(); index++) {

    let user = usuarios.obtenern(index)
    let icon = document.createElement("ion-icon")
    icon.setAttribute("name", "person-circle-outline")
    let userIcon = document.createElement("span")
    userIcon.classList.add("text-9xl")
    userIcon.classList.add("mx-5")
    userIcon.appendChild(icon)
    let userName = document.createElement("p")
    userName.setAttribute("class", "text-xl font-bold text-gray-800 mx-5")
    userName.innerHTML = user.username
    let botonAgregar = document.createElement("button")
    botonAgregar.setAttribute("id", "agregarAmigoUser")
    botonAgregar.setAttribute("name", user.username)
    botonAgregar.setAttribute("class", "mt-5 py-2 px-5 bg-green-700 border-green-800 rounded-xl hover:scale-110 duration-300 text-slate-100")
    botonAgregar.innerHTML = "Agregar"
    let botonBloquear = document.createElement("button")
    botonBloquear.setAttribute("id", "bloquearUsuarioUser")
    botonBloquear.setAttribute("name", user.username)
    botonBloquear.setAttribute("class", "my-5 py-2 px-5 bg-red-700 border-red-800 rounded-xl hover:scale-110 duration-300 text-slate-100")
    botonBloquear.innerHTML = "Bloquear"

    let userBox = document.createElement("div")
    userBox.setAttribute("class", "flex flex-col")
    userBox.appendChild(userIcon)
    userBox.appendChild(userName)

    if (usuarioActual.dato.friends.buscarUser(user.username) != null) {
        if (usuarioActual.dato.bloqueados.buscarUser(user.username) == null) {
            userBox.appendChild(botonBloquear)
        }
        friendsList.appendChild(userBox)
    } else {
        if (usuarioActual.dato.bloqueados.buscarUser(user.username) == null && usuarioActual.dato.username != user.username) {
            userBox.appendChild(botonAgregar)
            userBox.appendChild(botonBloquear)
            usersList.appendChild(userBox)
        }
    }
}
}

```

- **Mostrar Canciones:** Se recorre la lista de canciones y se van agregando uno por uno al html para que puedan ser vistos por el usuario y los pueda agregar a la playlist.



```

let cancionActualindex = 0
if (artistasCanciones.size() > 0) {
  let cancionActual = artistasCanciones.obtenern(cancionActualindex)
  const labelCancion = document.getElementById("nombreCancionMusic")
  const labelArtista = document.getElementById("artistaCancionMusic")

  labelCancion.innerHTML = cancionActual.nombre
  labelArtista.innerHTML = cancionActual.artista

  const agregarPlaylistMusic = document.getElementById("agregarPlaylistMusic")
  agregarPlaylistMusic.setAttribute("name", cancionActualindex)

  const botonSiguienteMusic = document.getElementById("nextCancionMusic")
  const botonAnteriorMusic = document.getElementById("anteriorCancionMusic")

  botonSiguienteMusic.addEventListener("click", () => {
    cancionActualindex++
    if (artistasCanciones.size() > cancionActualindex) {
      cancionActual = artistasCanciones.obtenern(cancionActualindex)
      labelCancion.innerHTML = cancionActual.nombre
      labelArtista.innerHTML = cancionActual.artista
      agregarPlaylistMusic.setAttribute("name", cancionActualindex)
    }
  })
  botonAnteriorMusic.addEventListener("click", () => {
    cancionActualindex--
    if (cancionActualindex >= 0) {
      cancionActual = artistasCanciones.obtenern(cancionActualindex)
      labelCancion.innerHTML = cancionActual.nombre
      labelArtista.innerHTML = cancionActual.artista
      agregarPlaylistMusic.setAttribute("name", cancionActualindex)
    }
  })
}

```

Link del Repositorio

https://github.com/DanielJH65/EDD_VD_Proyecto1_201901108