

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

Manual de Técnico:
Proyecto 2

Hecho por:
Walter Daniel Jiménez Hernández
201901108
Fecha: 01/01/2023

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. Actor
 - ii. Bloque
 - iii. Categoría
 - iv. Comentario
 - v. Película
 - vi. Usuario
 - b. Estructuras Usadas
 - i. Lista Simple
 - ii. ArbolBB
 - iii. ArbolAVL
 - iv. Tabla Hash
 - v. Blockchain
 - vi. Árbol de Merkle
 - c. Funciones Utilizadas
 - i. Login
 - ii. Carga Masiva
 - iii. Mostrar Películas
 - iv. Mostrar Actores
 - v. Mostrar Categorías
6. Link del Repositorio

Introducción

El programa cumple con la función de implementar diferentes estructuras de datos para poder organizar películas de manera eficiente. Se puede hacer carga masiva de los datos, visualizar el estado de las estructuras, comentar películas, puntuar películas y alquilar películas. Además el uso de encriptación para el manejo de los hash del blockchain.

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 no lineales
- HTML
- CSS
- Blockchain
- Graphviz
- Encriptación Sha256
- Ordenamiento burbuja

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. Los atributos que tiene son los siguientes:

```
export class Usuario{
  constructor(_dpi, _nombre_completo, _nombre_usuario, _correo, _contra, _telefono, _admin){
    this.admin = _admin
    this.dpi = _dpi
    this.mail = _correo
    this.name = _nombre_completo
    this.password = _contra
    this.phone = _telefono
    this.username = _nombre_usuario
  }
}
```

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

```
export class Actor{
  constructor(_dni, _nombre_actor, _correo, _descripcion){
    this.dni = _dni
    this.name = _nombre_actor
    this.mail = _correo
    this.description = _descripcion
  }
}
```


- **Bloque:** Es un objeto creado para poder guardar la información de cada bloque del blockchain . Los atributos que tiene son los siguientes:

```
export class Bloque{
  constructor(_index, _fecha, _dato, _nonce, _hashPrev, _merkle, _hash){
    this.index = _index
    this.date = _fecha
    this.transactions = _dato
    this.nonce = _nonce
    this.hashPrev = _hashPrev
    this.merkle = _merkle
    this.hash = _hash
  }
}
```

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

```
export class Categoria{
  constructor(_id, _company){
    this.id = _id
    this.company = _company
  }
}
```

- **Comentario:** Es un objeto creado para poder guardar la información de los comentarios de las películas. Los atributos que tiene son los siguientes:



```
export class Comentario{
    constructor(_username, _comentario){
        this.username = _username
        this.comentario = _comentario
    }
}
```


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



```
export class Pelicula{
    constructor(_id, _nombre, _descripcion, _star, _precio, _paginas, _categoria){
        this.id = _id
        this.name = _nombre
        this.description = _descripcion
        this.stars = _star
        this.price = _precio
        this.pages = _paginas
        this.category = _categoria
        this.comentarios = new ListaSimple()
    }
}
```

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
    }
}
```

- **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 actores.


```

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
  }
}

```

- **Árbol AVL:** Esta estructura ya no es lineal, cuenta con una raíz y esta con 2 nodos hijos, así se va ramificando con cada nodo pudiendo tener 2 hijos cada uno, esta se va auto-balanceando por su cuenta, en la aplicación se utiliza para poder almacenar las películas.

```

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

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

  insertarRecursivo(_rama, _dato){
    if(_rama == null){
      return new NodoArbolAVL(_dato)
    }else if(_rama.dato.id == _dato.id){
      Swal.fire('Oops...', 'La pelicula ya existe', 'error')
      return _rama
    }else if(_rama.dato.id < _dato.id){
      _rama.right = this.insertarRecursivo(_rama.right, _dato)
      if((this.altura(_rama.right) - this.altura(_rama.left)) == 2){
        if(_rama.right.dato.id < _dato.id){
          _rama = this.rotacionSimpleDer(_rama)
        }else{
          _rama = this.rotacionDobleDer(_rama)
        }
      }
    }else{
      _rama.left = this.insertarRecursivo(_rama.left, _dato)
      if((this.altura(_rama.left) - this.altura(_rama.right)) == 2){
        if(_rama.left.dato.id > _dato.id){
          _rama = this.rotacionSimpleIzq(_rama)
        }else{
          _rama = this.rotacionDobleIzq(_rama)
        }
      }
    }
  }
}

```

- **Tabla Hash:** Esta estructura es lineal, cuenta con un tamaño inicial y en cada nodo tiene una lista en donde se guardan las colisiones, el tamaño máximo para esta tabla es del 75%, en la aplicación se utiliza para poder almacenar las categorías.



```

export class TablaHash{
  constructor(){
    this.first = null
    this.size = 0
    this.used = 0
  }

  tamañoTabla(_size){
    for (let index = 0; index < _size; index++) {
      this.insertarIndex(index)
    }
  }

  insertarIndex(_index){
    let nuevo = new NodoTablaHash(_index)
    this.size++
    let temp = this.first
    if(temp != null){
      while(temp.next != null){
        temp = temp.next
      }
      temp.next = nuevo
    }else{
      this.first = nuevo
    }
  }
}

```

- **Blockchain:** Esta estructura es lineal, cuenta con diferentes hash de seguridad, tiene un registro del hash previo y el hash de la raíz del árbol de merkle, realiza una prueba de trabajo hasta que el hash inicia con 00, generado para cada

bloque en la aplicación se utiliza para poder almacenar las transacciones de alquiler.

```
export class Blockchain {
  constructor() {
    this.first = null
    this.last = null
    this.size = 0
    this.transaccionesTemp = new ArbolMerkle()
  }

  insertar(_dato) {
    let nuevo = new NodoBlockchain(_dato)
    this.size++
    if (this.first == null) {
      this.first = nuevo
      this.last = nuevo
    } else {
      nuevo.prev = this.last
      this.last.next = nuevo
      this.last = nuevo
    }
  }

  generarBloque(_merkle) {
    let date = this.obtenerDate()
    let hashprev = ""
    if (this.size == 0) {
      hashprev = "00"
    } else {
      hashprev = this.last.dato.hash
    }
  }
}
```

- **Árbol de Merkle:** Esta estructura no es lineal, cuenta con una cantidad de nodos hoja igual a una potencia de 2 para ser completo, el hash de cada hoja se transfiere al padre y cuando se unen 2 nodos hijos se suman los valores y se obtiene el hash, generado para cada bloque en la aplicación se utiliza para poder almacenar las transacciones de alquiler.

```
export class ArbolMerkle {
  constructor() {
    this.topHash = null
    this.dataBlock = new ListaSimple()
    this.size = 0
    this.index = 0
  }

  insertar(_dato) {
    this.dataBlock.insertarFinal2(_dato)
    this.size++
  }

  crearArbol(exp) {
    this.topHash = new NodoArbolMerkle(0)
    this.crearArbolRecursivo(this.topHash, exp)
  }

  crearArbolRecursivo(_rama, exp) {
    if (exp > 0) {
      _rama.left = new NodoArbolMerkle(0)
      _rama.right = new NodoArbolMerkle(0)
      this.crearArbolRecursivo(_rama.left, exp - 1)
      this.crearArbolRecursivo(_rama.right, exp - 1)
    }
  }
}
```

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
}
```

- **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 Películas:** Se recorre el árbol de películas y se va mostrando cada una de ellas, se agregan los botones para poder ver la información completa y también para poder alquilar cada una de ellas, además cuenta con botón para poder ordenar ascendentemente y descendentemente por nombre.

```

export function mostrarPelículas() {
  showUsersMovies()
  hideUsersActor()
  hideUsersCatego()
  hideUsersInfoMovies()
  const moviesList = $('#moviesList')
  moviesList.innerHTML = ""
  mostrarPelículasRecursivo(peliculas.root)

  const botonInfo = document.querySelectorAll('#infoMovies')
  botonInfo.forEach(element => {
    element.addEventListener("click", () => { infoPelículas(element.getAttribute("name")) })
  });

  const botonAlqui = document.querySelectorAll('#alquiMovies')
  botonAlqui.forEach(element => {
    element.addEventListener("click", () => { alquilar(peliculas.obtener(element.getAttribute("name"), peliculas.root)) })
  });
}

function mostrarPelículasRecursivo(root) {
  const moviesList = $('#moviesList')
  if (root != null) {
    mostrarPelículasRecursivo(root.left)
    moviesList.innerHTML += `
    <div class = "grid grid-cols-7 gap-4 items-center justify-center my-4 py-4 bg-[#0a283f] rounded-xl text-gray-200 opacity-90">
      <center><h2><b>${root.dato.name}</b></h2></center>
      <p class="col-span-3"><b>Descripción: </b>${root.dato.description}</p>
      <button id="infoMovies" name="${root.dato.id}" class="bg-[#496089] rounded-xl py-4 hover:scale-110 duration-300s">Información</button>
      <button id="alquiMovies" name="${root.dato.id}" class="bg-[#005a24] rounded-xl py-4 hover:scale-110 duration-300s">Alquilar</button>
      <h1><strong>Q ${root.dato.price}</strong></h1>
    </div>
  `
    mostrarPelículasRecursivo(root.right)
  }
}

```

- **Mostrar Actores:** Se recorre el árbol de actores y se va mostrando cada uno de ellas, además cuenta con botón para poder mostrar el contenido en Post-Orden, In-Orden y en Pre-Orden.

```
export function mostrarActores() {
  showUsersActor()
  hideUsersMovies()
  hideUsersCatego()
  hideUsersInfoMovies()
}

export function actoresInOrden() {
  const actorsList = $('#actorsList')
  actorsList.innerHTML = ""
  actoresInOrden2(actores.root)
}

function actoresInOrden2(raiz) {
  if (raiz != null) {
    actoresInOrden2(raiz.left)
    mostrarCardActores(raiz.dato)
    actoresInOrden2(raiz.right)
  }
}
```

- **Mostrar Categorías:** Se recorre la tabla hash y se muestra cada una de las categorías con todos sus datos.



```
export function mostrarCategorias() {  
    showUsersCatego()  
    hideUsersActor()  
    hideUsersMovies()  
    hideUsersInfoMovies()  
    categorias.mostrar()  
}
```

Link del Repositorio

https://github.com/DanielJH65/EDD_Proyecto2_201901108