Universidad Diego Portales
Escuela de Informática &
Telecomunicaciones

Estructuras de Datos & Análisis de Algoritmos

Laboratorio 2: Votación

Autores:
Valentina Martínez
José Pablo Peña

Profesor:
Marcos Fantoval

26 de abril de 2025

Índice

1. Introducción

En el presente informe se hará estudio del código encontrado en: `https://github.com/valemiaz/Lab-2`, el cual tiene un sistema de votación para gestionar candidatos, votantes y resultados de elecciones para presidente del Centro de Alumnos de la Escuela de Informática y Telecomunicaciones de la UDP. Para realizar este código se utilizará Java, un lenguaje de programación orientado a objetos, en el cual se crearán diferentes clases y métodos. Una clase es una plantilla o plano para crear objetos, la cual a su vez contiene atributos (variables) y métodos (funciones). Este es el fundamento de la programación orientada a objetos en Java. Se estudiará también la aplicación de listas, pilas y colas, haciendo uso de estas estructuras para facilitar el flujo y funcionamiento de la votación y comparando las ventajas y desventajas de estas, buscando la gestión más eficiente de la memoria del sistema.

2. Descripción de la implementación

El código cuenta con cuatro clases. En primer lugar está la clase Voto, que representa un voto individual. Tiene 4 atributos: id (identificador único del voto), votanteId (ID del votante), candidatoId (ID del candidato por el cual se votó) y timeStamp (fecha y hora del voto).

Clase `Voto` en Java

```
public class Voto {
    private int id;
    private int votanteId;
    private int candidatoId;
    private String timeStamp;

    public Voto (int id, int votanteId, int candidatoId, String
        timeStamp){
         this.id=id;
         this.votanteId=votanteId;
         this.candidatoId=candidatoId;
         this.timeStamp=timeStamp;
    }

    public void setId(int id){this.id=id;}
    public void setVotanteId(int votanteId)
        {this.votanteId=votanteId;}
    public void setCandidatoId(int candidatoId)
        {this.candidatoId=candidatoId;}
    public void setTimeStamp(String timeStamp)
        {this.timeStamp=timeStamp;}

    public int getId(){return id;}
    public int getVotanteId(){return votanteId;}
    public int getCandidatoId(){return candidatoId;}
    public String getTimeStamp(){return timeStamp;}


}
```

En segundo lugar está la clase Candidato, la cual representa a cada candidato de la elección. Tiene 4 atributos: id (identificador del candidato), nombre (nombre del candidato), partido (partido político del candidato) y votosRecibidos (una cola de votos recibidos). También tiene 1 método llamado agregarVoto, el cual agrega un voto a la cola votosRecibidos y devuelve un error si es nulo.

Clase `Candidato` en Java

```
1  public class Candidato {
2      private int id;
3      private String nombre;
4      private String partido;
5      private Queue<Voto> votosRecibidos;
6
7      public Candidato (int id, String nombre, String partido,
           Queue<Voto> votosRecibidos){
8          this.id=id;
9          this.nombre=nombre;
10         this.partido=partido;
11         this.votosRecibidos=votosRecibidos;
12     }
13
14     public void setId(int id){this.id=id;}
15     public void setNombre(String nombre){this.nombre=nombre;}
16     public void setPartido(String partido){this.partido=partido;}
17     public void setVotosRecibidos(Queue<Voto>
           votosRecibidos){this.votosRecibidos=votosRecibidos;}
18
19     public int getId(){return id;}
20     public String getNombre(){return nombre;}
21     public String getPartido(){return partido;}
22     public Queue<Voto> getVotosRecibidos(){return votosRecibidos;}
23
24     public void agregarVoto(Voto v) {
25         if (v == null) {
26             throw new IllegalArgumentException("El voto no puede
                   ser nulo");
27         }
28         votosRecibidos.add(v);
29     }
30 }
```

En tercer lugar está la clase Votante, la cual representa a cada persona que puede votar. Tiene 3 atributos: id (identificador del votante), nombre (nombre del votante) y yaVoto (booleano que indica si ya votó). También tiene 1 método llamado marcarComoVotado, el cual marca a cada votante que ya votó.

Clase `Votante` en Java

```java
public class Votante {
    private int id;
    private String nombre;
    private boolean yaVoto;

    public Votante (int id, String nombre, boolean yaVoto){
        this.id=id;
        this.nombre=nombre;
        this.yaVoto=yaVoto;
    }

    public void setId(int id){this.id=id;}
    public void setNombre(String nombre){this.nombre=nombre;}
    public void setYaVoto(boolean yaVoto){this.yaVoto=yaVoto;}

    public int getId(){return id;}
    public String getNombre(){return nombre;}
    public boolean getYaVoto(){return yaVoto;}

    public void marcarComoVotado(){
        this.yaVoto=true;
    }
}
```

Finalmente está la clase UrnaElectoral, la cual simboliza la lógica interna final de todo el sistema de votación. Tiene 4 atributos: listaCandidatos (lista de todos los candidatos), historialVotos (pila con todos los votos emitidos), votosReportados (cola de votos que han sido reportados por posible fraude) y idCounter (contador para asignar ID únicos a los votos). A esta clase se le suman también 4 métodos: verificarVotante(Votante) devuelve true si el votante aún no ha votado, registrarVoto(Votante, int candidatoID) registra un voto si el votante no ha votado antes, reportarVoto(Candidato, int idVoto) marca un voto como reportado por fraude y obtenerResultados() devuelve un mapa con el nombre de cada candidato y la cantidad de votos recibidos.

Clase `UrnaElectoral` en Java

```
public class UrnaElectoral {
    private LinkedList<Candidato> listaCandidatos;
    private Stack<Voto> historialVotos;
    private Queue<Voto> votosReportados;
    private int idCounter;

    public UrnaElectoral (){
        listaCandidatos = new LinkedList<>();
        historialVotos = new Stack<>();
        votosReportados = new LinkedList<>();
        idCounter = 1;
    }

    public boolean verificarVotante(Votante votante){
        return !votante.getYaVoto();
    }

    public void registrarVoto(Votante votante, int candidatoID) {
        if (votante.getYaVoto()) return;

        for (Candidato candidato : listaCandidatos) {
            if (candidato.getId() == candidatoID){
                Voto nuevoVoto = new Voto(idCounter++,
                    votante.getId(), candidatoID, new
                    Date().toString());
                candidato.agregarVoto(nuevoVoto);
                historialVotos.push(nuevoVoto);
                votante.marcarComoVotado();
                break;
            }
        }
    }

    public void reportarVoto (Candidato candidato, int idVoto){
        for (Voto voto : candidato.getVotosRecibidos()){
            if (voto.getId() == idVoto){
```

```
35                  if (votosReportados.contains(voto)){
36                      System.out.println("Fraude detectado. El voto
                            ya ha sido reportado.");
37                      return;
38                  }
39                  votosReportados.add(voto);
40                  System.out.println("Voto reportado exitosamente.");
41                  return;
42              }
43          }
44
45          System.out.println("Voto no encontrado para el candidato
                con ID: " + candidato.getId());
46      }
47
48      public Map<String, Integer> obtenerResultados() {
49          Map<String, Integer> resultados = new HashMap<>();
50
51          for (Candidato candidato : listaCandidatos) {
52              String nombre = candidato.getNombre();
53              int cantidadVotos =
                    candidato.getVotosRecibidos().size();
54              resultados.put(nombre, cantidadVotos);
55          }
56
57      return resultados;
58      }
59
60
61  }
```

En este programa, las tres primeras clases (Voto, Candidato y Votante) son clases auxiliares que sirven como apoyo de la clase principal, UrnaElectoral, la cual modela el funcionamiento de todo el sistema.

3. Análisis de complejidad y uso de memoria

Se hace análisis de complejidad a tres métodos clave los cuales pertenecen a la última clase, Urnaelectoral:

Complejidad: Clase `UrnaElectoral`

registrarVoto()

```
public void registrarVoto(Votante votante, int candidatoID) {
    if (votante.getYaVoto()) return;

    for (Candidato candidato : listaCandidatos) {
        if (candidato.getId() == candidatoID){
            Voto nuevoVoto = new Voto(idCounter++,
                votante.getId(), candidatoID, new
                Date().toString());
            candidato.agregarVoto(nuevoVoto);
            historialVotos.push(nuevoVoto);
            votante.marcarComoVotado();
            break;
        }
    }
}
```

La complejidad de este método es $\mathcal{O}(n)$ ya que tiene que recorrer (ciclo for) toda la lista de candidatos hasta encontrar el que tiene el ID que busca. Si el candidato está al final de la lista, se revisan todos, haciendo que se tenga complejidad $\mathcal{O}(n)$.

reportarVoto()

```
public void reportarVoto(Candidato candidato, int idVoto) {
    for (Voto voto : candidato.getVotosRecibidos()) {
        if (voto.getId() == idVoto) {
            if (votosReportados.contains(voto)) {
                System.out.println("Fraude detectado. El voto ya
                    ha sido reportado.");
                return;
            }
            votosReportados.add(voto);
            System.out.println("Voto reportado exitosamente.");
            return;
        }
    }

    System.out.println("Voto no encontrado para el candidato con
        ID: " + candidato.getId());
}
```

La complejidad de este método es $\mathcal{O}(n)$ ya que hay un ciclo for que recorre todos los votos recibidos por el candidato. En el peor de los casos, el voto buscado puede estar al final o directamente no estar, por lo que se revisa toda la lista.

obtenerResultados()

```
public Map<String, Integer> obtenerResultados() {
    Map<String, Integer> resultados = new HashMap<>();

    for (Candidato candidato : listaCandidatos) {
        String nombre = candidato.getNombre();
        int cantidadVotos = candidato.getVotosRecibidos().size();
        resultados.put(nombre, cantidadVotos);
    }

    return resultados;
}
```

La complejidad de este método es $\mathcal{O}(n)$ ya que el for recorre todos los candidatos en la lista y agrega sus nombres y la cantidad de votos que recibieron al Map de resultados.

En resumen:

Operación	**Complejidad**
`registrarVoto()`	$\mathcal{O}(n)$
`reportarVoto()`	$\mathcal{O}(n)$
`obtenerResultados()`	$\mathcal{O}(n)$

Gestión de memoria

Se calcula el espacio requerido para almacenar todos los votos con dos características: en primer lugar, que cada voto utiliza 64 bytes, y en segundo lugar, que el sistema admite hasta 10 millones de votantes. Esto se calcula de la siguiente forma:

$$\text{Espacio requerido} = 10\,000\,000 \times 64 = 640\,000\,000\,\text{bytes}$$

$$\text{Espacio en megabytes} = \frac{640\,000\,000}{1\,048\,576} \approx 610\,\text{MB}$$

Según el resultado, el sistema necesita aproximadamente 610 MB de memoria.

Propuesta de mejora

Una mejora del programa sería modificar el sistema para soportar votaciones en múltiples facultades. Para esto se podría agregar una nueva clase llamada Facultad que a su vez contenta su propia clase UrnaElectoral, lista de candidatos y lista de votantes, y crear una clase principal que administre todas las facultades.
Otra mejora sería mantener el historial de votos en orden cronológico inverso. Esto se puede hacer a través del uso de una lista enlazada de los votos, la cual permite recorrer el historial de forma directa en orden inverso sin modificarlo.

4. Ventajas y desventajas de utilizar listas enlazadas

En este caso, la ventaja principal de las listas enlazadas es que permiten insertar y eliminar elementos con flexibilidad sin necesidad de mover otros, sobre todo considerando que el tamaño de la lista es dinámico debido a la cantidad de votantes.

Por otro lado, una desventaja sería su mayor complejidad con respecto a los arreglos: La complejidad de las listas enlazadas es O(n), ya que se necesita recorrer la lista completa para encontrar un elemento, mientras que la complejidad de los arreglos es O(1) por su búsqueda inmediata. Otra desventaja en relación a lo hablado en el informe, es que las listas enlazadas consumen más memoria ya que cada nodo debe guardar información extra para referenciar el siguiente nodo y así sucesivamente, en cambio los arreglos solo contienen los datos.

5. Conclusión

En el desarrollo de este laboratorio, se logró implementar un sistema de votación utilizando Java y estructuras de datos en variadas clases, lo que nos permitió lograr los objetivos planteados.

¿Qué se hizo? El laboratorio consistió en crear un programa que simulara una votación, se debía trabajar para organizar candidatos, votantes y los votos emitidos. Para esto se utilizan las clases Voto, Candidato, Votante y UrnaElectoral, aplicando conceptos de programación orientada a objetos.

¿Con qué se hizo? Se utilizó el lenguaje de programación Java, por sus características de orientación a objetos, como encapsulación, métodos y atributos. Además, utilizamos estructuras de datos de Java, como LinkedList, Stack y Queue, para optimizar el manejo de la información.

¿Cómo se hizo? Desarrollamos el código usando estructuras de datos las cuales son listas enlazadas, pilas y colas para organizar y manipular la información. Por ejemplo, la clase UrnaElectoral empleó una lista enlazada para almacenar candidatos, una pila para el historial de votos y una cola para los votos. Esto es para operaciones como el registro de votos, la verificación de votantes y los resultados.

¿Por qué se hizo? El objetivo principal era reforzar nuestros conocimientos en programación orientada a objetos y estructuras de datos, aplicándolos a un problema realista. Esto personalmente nos permitió entender como seleccionar la estructura adecuada para cada necesidad, evaluando ventajas y desventajas en términos de complejidad y aprender sobre el uso de memoria.

Objetivos: A través del desarrollo de este laboratorio, se logró evaluar el dominio de listas enlazadas, pilas y colas aplicadas en un contexto realista, siendo estas dirigidas a la creación de un sistema de votaciones. Se diseñaron múltiples clases orientadas a objetos, practicándose así la creación de clases, implementando relaciones entre Candidato, Votante, Voto y UrnaElectoral. También aprendiendo sobre el uso de memoria en Java y la diferencia según la estructura de datos que se utilice ya que se realizó una comparación entre distintas estructuras de datos, identificando ventajas y desventajas relevantes, siendo estas características como la flexibilidad de las listas enlazadas frente a los arreglos, así como su impacto en el rendimiento y el consumo de memoria.

En conclusión, el laboratorio cumplió plenamente los objetivos propuestos.