Universidad Diego Portales
Escuela de Informática &
Telecomunicaciones

Estructuras de Datos & Análisis de Algoritmos

Laboratorio n°1

Autores:
Allen Mora
Gabriel Varas

Profesor:
Marcos Fantóval

21-04-2025

Índice

1. Implementación

En este laboratorio se implementará una manera de agilizar las votaciones presidenciales para el Centro de Alumnos de la Escuela de Informática y Telecomunicaciones de la Universidad Diego Portales. Esto se logrará mediante la creación del sistema 'Electo', el cual permitirá gestionar a los votantes, los candidatos y los resultados de las elecciones.

2. Clases y Métodos

Las votaciones seguirán el siguiente flujo:

- 1. La persona va a votar.
- 2. El sistema verifica si la persona ya votó:
  - Si la persona ya votó, reportar el voto.
  - Si no ha votado, ingresa el voto a la urna.
- 3. Para ingresar un voto a la urna se siguen los siguientes pasos:
  - Crear nuevo voto.
  - Asignar el voto a la cola del candidato correspondiente.
  - Añadir el voto al historial.
  - Cambiar estado de voto del votante.

Para lograr todo esto se utilizarán las siguientes clases y métodos

- Clases:
  - Electo : Dentro de esta clases es donde ocurre todo, las demás clases y métodos se encuentran dentro de esta.

- 1. Voto: Esta clase contiene todos los parámetros que debe tener un voto, los cuales son:
  - id: Es el identificador único del voto.
  - votanteId: Contiene el identificador del votante.
  - cantidadoId: Contiene el identificador del candidato por el cual se votó.
  - timestamp : Este parámetro se utiliza para saber cuando se realizo el voto.

Figura 1: Clase Voto

- 2. Candidato: En esta clase se encuentran los parámetros de cada candidato, estos parametros son:
  - id: Identificador único del candidato.
  - nombre: Nombre del candidato.
  - partido: Partido del candidato.
  - votosRecibidos: Es una cola la cual contiene todos los votos que este candidato ha recibido.
  - agregarVoto(Voto voto): Método que inserta un elemento en la cola.
  - eliminarVoto(Voto voto): Método que elimina un voto especifico de la cola.

Figura 2: Clase Candidato

- 3. Votante: En esta clase se representa a cada una de las personas que votarán, esta clase contiene 3 parámetros los cuales son:
  - id: Identificador de cada votante.
  - nombre: Nombre del votante.
  - yaVoto: Este parámetro es un booleano el cual indica si el votante ya realizo su voto o si aun no lo realiza.
  - marcarComoVotado(): Método que cambia el valor de la variable yaVoto a true.

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

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

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

    public boolean getYaVoto() {
        return this.yaVoto;
    }

    public int getID() {
        return id;
    }

    public String getNombre() {
        return nombre;
    }
}
```

Figura 3: Clase Votante

- 4. UrnaElectoral: Esta clase funciona como su nombre indica como una Urna Electoral, esta contiene también 4 parametros para su funcionamiento los cuales son:
  - listaCandidatos: Esta es una LinkedList que contiene a todos los candidatos participantes de las elecciones.
  - historialVotos: Esta es una pila que contiene todos los votos emitidos.
  - votosReportados: Esta es una cola donde se guardan los votos que han sido reportados.
  - idCounter: Se utiliza este parámetro para asignar los ID únicos de cada voto. Va aumentando con cada voto asignado.
  - buscarCandidato(int id): Método que busca un candidato según el ID proporcionado. Utiliza un ciclo for para recorrer dinámicamente la lista enlazada de candidatos.

```java
public Candidato buscarCandidato(int id) {
    for (Candidato c : listaCandidatos) {
        if (c.getID() == id) {
            return c;
        }
    }
    return null;
}
```

Figura 4: Método buscarCandidato

  - verificarVotante(Votante votante): Método que verifica si alguien ya ha votado, utilizando el metodo getYaVoto de la clase Votante.
  - registrarVoto(Votante votante, int candidatoID): Método encargado de registrar un voto. Primero se verifica si el votante ya ha votado. En caso afirmativo, se reporta el voto correspondiente, buscando

```
123         public boolean verificarVotante(Votante votante) {
124             return votante.getYaVoto();
125         }
```

Figura 5: Clase Votante

el candidato por su ID y recorriendo sus votos para identificar cuál pertenece al votante, llamando luego al método reportarVoto. Si el votante aún no ha votado, se busca al candidato usando buscarCandidato, y si se encuentra un candidato válido, se crea un nuevo objeto Voto utilizando LocalTime.now() para registrar la hora actual. Luego, el voto se añade tanto al candidato como al historial de votos, se marca al votante como que ya ha votado, se incrementa el idCounter y se retorna true indicando que el proceso se completó exitosamente.

```
127     public boolean registrarVoto(Votante votante, int candidatoID) {
128         if (verificarVotante(votante)) {
129             System.out.println("El votante " + votante.getNombre() + " ya ha votado.");
130             Candidato candidato = buscarCandidato(candidatoID);
131             if (candidato != null) {
132                 for (Voto v : candidato.getVotosRecibidos()) {
133                     if (v.getVotanteID() == votante.getID()) {
134                         reportarVoto(candidato, v.getID());
135                         return true;
136                     }
137                 }
138             }
139             return false;
140         }
141 
142         Candidato candidato = buscarCandidato(candidatoID);
143 
144         if (candidato == null) {
145             System.out.println(x:"Candidato no encontrado.");
146             return false;
147         }
148 
149         String timestamp = java.time.LocalTime.now().toString();
150         timestamp = timestamp.substring(beginIndex:0, endIndex:8);
151 
152         Voto voto = new Voto(idCounter, votante.getID(), candidatoID, timestamp);
153 
154         candidato.agregarVoto(voto);
155         historialVotos.push(voto);
156         votante.marcarComoVotado();
157         idCounter++;
158 
159         return true;
160     }
```

Figura 6: Metodo registrarVoto

- reportarVoto(Candidato candidato, int idVoto): Método que permite eliminar un voto reportado. Se obtienen los votos del candidato indicado y se recorre la lista de votos usando un ciclo for. Si se encuentra un voto con el ID coincidente, se elimina utilizando el método eliminarVoto del candidato y se añade el voto anulado a la cola de votosReportados.
- obtenerResultados(): Método que retorna los resultados de las elecciones en un Map¡String, Integer¿. Se crea un nuevo mapa, se recorre la lista de candidatos, y se añade una entrada con el nombre del

```java
public boolean reportarVoto(Candidato candidato, int idVoto) {
    System.out.println("Reportando voto..." + idVoto);
    Queue<Voto> votos = candidato.getVotosRecibidos();

    for (Voto v : votos) {
        if (v.getID() == idVoto) {
            votosReportados.offer(v);
            candidato.eliminarVoto(v);
            System.out.println("Voto reportado: " + v.getID());
            return true;
        }
    }
    return false;
}
```

Figura 7: Metodo reportarVoto

candidato como llave y la cantidad de votos recibidos (obtenida del tamaño de la cola de votos) como valor.

```java
public Map<String, Integer> obtenerResultados() {
    Map<String, Integer> resultados = new HashMap<>();
    for (Candidato c : listaCandidatos) {
        resultados.put(c.getNombre(), c.getVotosRecibidos().size());
    }
    return resultados;
}
```

Figura 8: Metodo reportarVoto

- agregarCandidato(Candidato candidato): Método utilizado para la simulación. Agrega un candidato a la lista enlazada de listaCandidatos.

```java
public void agregarCandidato(Candidato candidato) {
    listaCandidatos.add(candidato);
}
```

Figura 9: Metodo reportarVoto

3. Experimentación y Resultados

3.1. Calculo del espacio requerido para almacenar los votos

Suponiendo que cada voto utiliza 64 Bytes, y el código admite hasta 10 millones de votantes, el espacio total a ocupar estaría dado por el numero de botos multiplicado por el tamaño de cada voto.

Por lo que se tendría 64 Bytes * 1.000.000 = 64.000.000 Bytes o lo que es igual 640 MB.

3.2. Propuesta de mejora

¿Cómo modificarías el sistema para soportar votaciones en múltiples facultades?

Agregar una nueva clase Facultad, que contenga su propia instancia de UrnaElectoral, lista de candidatos y votantes. De esta forma, cada facultad gestiona su propia elección de manera independiente

3.3. Complejidad Temporal

Método	**Función**	**Complejidad**
registrarVoto	- Buscar candidato en listaCandidato (ciclo for).	O(n)
	- Agregar voto a votosRecibidos.	O(1)
	- Actualizar estado de votante.	O(1)
	Total:	O(n)
obtenerResultados	- Iterar sobre listaCandidatos (ciclo for)	O(n)
reportarVoto	- Buscar voto en votosRecibidos (ciclo fo)	O(n)
	- Agregar el voto a votosReportados	O(1)
	- Eliminar voto de la cola de votosRecibidos	O(1)
	Total:	O(n)

Cuadro 1: Tabla notacion Big-O

4. Conclusiones

Como ventajas al utilizar listas enlazadas(LinkedList) sobre arreglos tenemos que al utilizar listas enlazadas resulta más fácil agregar nuevos candidatos o eliminar candidatos ya existentes. Otra ventaja de utilizar listas enlazadas, está al momento de agregar nuevos votos, puesto que este solo se enlaza a un nuevo nodo.

Algunas desventajas de utilizar las listas enlazadas, se encuentra al momento de querer buscar algún candidato por su ID, se vuelve lento debido a que para hacer esto se debe recorrer toda la lista, en cambio si se utilizaran arreglos, se obtendría el ID del candidato de manera directa, otra desventaja se encuentra en los votos puesto que no es posible acceder a cada voto por su indice.