Universidad Diego Portales
Escuela de Informática &
Telecomunicaciones

Estructuras de Datos & Análisis de Algoritmos

Laboratorio 2: Votación

Autores:
Cinthya Fuentealba
Joaquín Roco

Profesor:
Marcos Fantoval

26 de Abril 2025

Índice

1. Introducción

En el mundo existen varias formas de gobierno, pero en este informe nos centraremos en la **democracia**. En la democracia el factor más importante son los ciudadanos, debido a que estos son lo que ejercen el poder para elegir a sus representantes políticos. Para que esto suceda se opta por un sistema de elecciones, donde se contabilizan los votos que recibe cada candidato previamente postulado y se genera un registro, en este registro se observa al candidato con la mayoría de votos, quien se convierte en el representante seleccionado finalmente. Este puede ser el representante (de los ciudadanos) para un puesto político como; diputado, senador, presidente, entre otros.
Este modelo tiene una serie de requisitos. Por ejemplo, en Chile: Todos los chilenos (as) y extranjeros que tengan cumplidos 18 años de edad y que se encuentren inscritos en el padrón electoral, el dia de la correspondiente a la votación tienen el derecho a voto.

En este segundo laboratorio, se busca dar solución a otra problemática relacionada con el sistema de votación, esta consiste en que se realizará una votación para la elección de presidente del Centro de Alumnos de la Escuela de Ingeniería e Informática de la Universidad Diego Portales. Por ello, se nos solicita poder crear el sistema de votación para esta elección mediante código de Java.

Este informe requiere del uso de un programa informático creado en IntelliJ IDEA, este se caracteriza por ser un editor, compilador y depurador de, entre otros lenguajes, Java.

Por ende, dentro del informe se detallan la realización del código que busca dar solución al problema propuesto, también sobre los componentes de este `código` y su funcionamiento. En el repositorio de GitHub [1] se encuentra el código `.Java` y el archivo LaTeX para su análisis.

2. Descripción de la implementación

2.1. Descripción del código

Lo primero a realizar es identificar partes importantes que contemplan un sistema de votación.

2.1.1. Clase Voto

Inicialmente, se debe comenzar sobre la estructura que compone el voto, para ello se realiza una `class Voto`, el cual contiene los siguientes atributos :

- **idVoto**
  Este corresponde al ID de identificación única del voto, que es un tipo de dato de tipo `int`.
- **idVotante**
  Este corresponde al ID único de la persona votante, este es de tipo `int`.
- **idCandidato**
  Este atributo hace rederencia al ID del candidato que la persona eligio como preferencia, este es un dato de tipo `int`.
- **timeStamp**
  Este corresponde a la hora en la que se realizó el voto, en formato "hh:mm:ss" (hora, minuto y segundo), este es de tipo `String`.

```
class Voto {
    private int idVoto;
    private int votanteId;
    private int candidatoId;
    private String timeStamp; //Hora del voto en formato "hh:mm:ss"
```

Código 1: Atributos de clase Voto

A esto le sigue el `Constructor` y los métodos que, para esta clase, no se necesitan más que los `getters` y `setters`.

- **Constructor:** inicializa los atributos de la clase.

```
    //Constructor
    public Voto(int idVoto, int votanteId, int candidatoId,
        String timeStamp) {
        this.idVoto = idVoto;
        this.votanteId = votanteId;
        this.candidatoId = candidatoId;
        this.timeStamp = timeStamp;
    }
```

Código 2: Constructor clase Voto

- **Getters y Setters:** son los que permiten acceder y modificar los valores de las variables privadas de la clase.

```
    //getters
    public int getidVoto() {return idVoto;}
    public int getVotanteId() {return votanteId;}
    public int getCandidatoId() {return candidatoId;}
    public String getTimeStamp() {return timeStamp;}

    //setters
    public void setId(int idVoto) {this.idVoto = idVoto;}
    public void setVotanteId(int votanteId) {this.votanteId =
        votanteId;}
    public void setCandidatoId(int candidatoId) {this.candidatoId
        = candidatoId;}
    public void setTimeStamp(String timeStamp) {this.timeStamp =
        timeStamp;}
```

Código 3: Métodos de clase Voto

2.1.2. Clase Candidato

Esta clase contiene los siguientes atributos:

- **id**
  Es el ID (único) del candidato, que esta en tipo `int`.
- **nombre**
  El nombre asociado al candidato y puede no ser único. Es de tipo `String`.
- **partido**
  El partido al cual pertenece el candidato. Es de tipo `String`.
- **votosRecibidos**
  Es una cola donde solo se almacenan los objetos de tipo `Voto`, que van a ser los votos asociados al candidato correspondiente. Es de tipo `Queue`.

```
    class Candidato {
    private int id;
    private String nombre;
    private String partido;
    private Queue<Voto> votosRecibidos; //cola de votos
        asociados al candidato de y acepta solo objetos tipo
        Voto
```

Código 4: Atributos de clase Candidato

- Ademas de sus correspondientes `Constructor`, `getters y setters`.

```
    //Constructor
    public Candidato(int id, String nombre, String partido) {
        this.id = id;
        this.nombre = nombre;
```

```
5           this.partido = partido;
6           votosRecibidos = new LinkedList<Voto>();
7       }
8
9       //getters
10      public int getId() {return id;}
11      public String getNombre() {return nombre;}
12      public String getPartido() {return partido;}
13
14      //setters
15      public void setId(int id) {this.id = id;}
16      public void setNombre(String nombre) {this.nombre =
            nombre;}
17      public void setPartido(String partido) {this.partido =
            partido;}
```

Código 5: Constructor, getters y setters de clase Candidato

- La clase `Candidato` debe llevar el metodo `agregarVoto()` para poder añadir los votos a la cola de `votosRecibidos` y asi contabilizar los votos del candidato correspondiente.

```
1       //Mete un voto a la cola de votos del candidato.
2       public void agregarVoto(Voto v) {votosRecibidos.add(v);}
```

Código 6: Método agregarVoto() clase Candidato

- Aparte de este metodo, se crean dos métodos auxiliares llamados `removerVotoPorId()` y `getTotalVotos()`.
- El método `removerVotoPorId()` esta enfocado en ser utlizado por el metodo `reportarVoto()`(15) de la clase`UrnaElectoral`(11). De esta forma `reportarVoto()` puede cumplir la función de mover un voto de la cola correspondiente al candidato a la cola de votos reportados.

```
1       /**
2        * Busca en votosRecibidos un voto con idVoto,
3        * lo elimina y lo devuelve; si no lo halla, devuelve null.
4        */
5       public Voto removerVotoPorId(int idVoto) { /** CREADO PARA
            USARLO EN reportarVoto() */
6           Iterator<Voto> it = votosRecibidos.iterator();
7           while (it.hasNext()) {
8               Voto v = it.next();
9               if (v.getidVoto() == idVoto) {
10                  it.remove();   // elimina seguro dentro del
                        iterator
11                  return v;      // devolvemos el voto para
                        reusar
12              }
```

```
13          }
14          return null;
15      }
```

Código 7: Método removerVotoPorId() clase Candidato

- El método `getTotalVotos()` esta enfocado en ser utilizado por el metodo `obtenerResultados()`(16) de la clase `UrnaElectoral` tenga la facilidad de retornar los resultados de las elecciones.

```
1           public int getTotalVotos() { /** CREADO PARA
                obtenerResultados() */
2           return votosRecibidos.size();
3       }
```

Código 8: Método getTotalVotos()

2.1.3. Clase Votante

Esta clase esta enfocada en tener un registro de las personas que van a votar y que ya han votado, para esto contiene los atributos:

- **id**
  Es el ID (único) de la persona que va a votar(votante), que esta en tipo `int`.
- **nombre**
  Es el nombre asociado al votante y puede no ser único. Es de tipo `String`.
- **yaVoto**
  Es de tipo `boolean` y se usa para verificar si la persona (el votante) voto previamente. Este método se usa para evitar votos duplicados y evitar fraudes.

```
1       class Votante {
2       private int id;
3       private String nombre;
4       private boolean yaVoto; //booleano para verificar si la
            persona voto previamente
```

Código 9: Atributos clase Votante

- Esta clase posee un constructor, getters, setters y el método `marcarComoVotado()` el cual cambia el estado del boolean `yaVoto`. Por ejemplo, si `yaVoto` es *"false"*, con este método cambia a *"true"*.

```
1       //Constructor
2       public Votante(int id, String nombre, boolean yaVoto) {
3           this.id = id;
4           this.nombre = nombre;
5           this.yaVoto = yaVoto;
6       }
7
8       //getters
```

```
9       public int getId() {return id;}
10      public String getNombre() {return nombre;}
11      public boolean getYaVoto() {return yaVoto;}
12
13      //setters
14      public void setId(int id) {this.id = id;}
15      public void setNombre(String nombre) {this.nombre =
            nombre;}
16      public void setYaVoto(boolean yaVoto) {this.yaVoto =
            yaVoto;}
17
18      //Cambia yaVoto a true
19      public void marcarComoVotado() {yaVoto = true;}
20  }
```

Código 10: Contructor y métodos clase Votante

2.1.4. Clase UrnaElectoral

La UrnaElectoral sirve como un medio para que los votantes puedan votar por un candidato. Tiene los siguientes atributos:

- **listaCandidatos**
  Es una lista enlazada donde se almacenan los objetos de tipo `Candidato`, es decir los candidatos. Es de tipo `LinkedList`.
- **historialVotos**
  Es una pila que almacena objetos de tipo `Voto` en orden cronologico inverso. Es de tipo `stack`.
- **votosReportados**
  Es una cola de objetos tipo `Voto` para votos anulados o impugnados. Es de tipo `Queue`.
- **idCounter**
  Es un contador de IDs para votos (le asigna un ID único a cada voto). Es de tipo `int`.

```
1       class UrnaElectoral {
2       private LinkedList<Candidato> listaCandidatos;
3       private Stack<Voto> historialVotos;
4       private Queue<Voto> votosReportados;
5       private int idCounter;
```

Código 11: Atributos de clase UrnaElectoral

- Esta clase posee un contructor que inicializa todos los atributos y el `idCounter` en 0.

```
1
2     public UrnaElectoral() {
3         listaCandidatos = new LinkedList();
```

```
4           historialVotos = new Stack(); //pila para almacenar votos
                en orden cronologico inverso (la pila recibira solo
                Votos)
5           votosReportados = new LinkedList<Voto>(); //cola para
                votos anulados o impugnados (La cola recibira solo
                Votos)
6           idCounter = 0; //contador de IDs para votos
7       }
```

Código 12: Contructor de clase UrnaElectoral

- El método `verificarVotante()`, como su nombre lo indica, verifica si el votante voto previamente o esta es la primera vez que lo hará. Para ello hace uso del método `getyaVoto()`(10) perteneciente a la clase `Votante`(9) que retorna el valor booleano *"true"* en caso de que ya haya votado o *"false"* en caso contrario.

```
1
2       //Verifica si el votante ya ha votado.
3       public boolean verificarVotante(Votante votante) {
4           return votante.getYaVoto();
5       }
```

Código 13: Método verificarVotante

- El método `registrarVoto()` utiliza el método `verificarVotante()` para eliminar la posibilidad de que votara anteriormente. Si no ha votado entonces se usa el *guard clause* para facilitar la legibilidad, evitando un *else* o *else if*. El siguiente paso es buscar al candidato por el cual el `votante` va a votar, para esto se emplea un *for-each* para recorrer la `listaCandidatos`.
- Un *for-each* es un"for mejorado"que usa internamente un Iterator, que avanza en O(1) cada vez, resultando en O(n) total. Esto simplifica el recorrido de colecciones o arreglos.
- Luego comprueba que el ID del candidato que estamos buscando se encuentre en esta lista para crear el voto (si no esta, es porque no hay candidato con ese ID). Se utiliza `LocalTime.now()` para asignarle un formato de tiempo al voto.
- Después se agrega el voto a la cola `votosRecibidos` asociada al candidato con el metodo `agregarVoto()`(6). También se agrega el voto a la pila `historialVotos`(11). Y por último se cambia el estado del votante, para saber que ya ha votado, con el metodo `marcarComoVotado()`(10).

  Se hace un `return true` para dar a entender que el voto se registró con éxito.

```
1
2       //Utiliza el metodo verificarVotante
3       //luego registra el voto (se tiene que crear un nuevo Voto
            para agregarlo al Candidato)
```

```
4       //en el Candidato correspondiente y lo aniade al historial.
            Posteriormente tiene que cambiar el estado del Votante
5       public boolean registrarVoto(Votante votante, int candidatoId)
            {
6           // 1) Si ya voto, salimos con false
7           if (verificarVotante(votante)) {
8               return false;
9           } // guard clause o early return
10
11          // 2) Buscamos el candidato en la lista
12          for (Candidato candidato : listaCandidatos) { //for-each o
                "for mejorado", for (Tipo elemento : coleccion)
13              if (candidato.getId() == candidatoId) {
14
15                  // 3) Creamos un nuevo Voto
16                  String timeStamp = LocalTime.now().format
17                  (DateTimeFormatter.ofPattern("HH:mm:ss"));
18                  idCounter++;
19                  Voto nuevoVoto = new Voto(idCounter,
                        votante.getId(), candidatoId, timeStamp);
20                  // auto incremental, id del elector, // id del
                        candidato, // hora en hh:mm:ss
21
22                  // 4) Lo agregamos al candidato
23                  candidato.agregarVoto(nuevoVoto);
24
25                  //y al historial
26                  historialVotos.push(nuevoVoto);
27
28                  // 5) Marcamos al votante como que ya voto
29                  votante.marcarComoVotado();
30
31                  return true;  // voto registrado con exito
32              }
33          }
34
35          // Si llegamos aqui, no existe ningun candidato con ese ID
36          return false;
37      }
```

Código 14: Método registrarVoto

- El método `reportarVoto()` cumple la función de mover un voto de la cola `votoRecibidos`(4), del candidato correspondiente, a la cola de `votosReportados`(11).
- Para esto, primero recorre la cola `votosReportados` en busca del ID del voto, verificando si estaba reportado anteriormente y en caso de estarlo entregar un mensaje avisando.
- Si no se encuentra, le pedimos al candidato que quite el voto de su cola. El método `removerVotoPorId()`(7) es clave debido a que simplifica la operación. Si el método retorna *"null"* no existe el voto en la cola de del candidato.

- Por último, si se hizo efectivo el hallazgo del voto en la cola del candidato, se añade a la cola `votosReportados`(11) y retorna *"true"*.

```
1         //Mueve un voto de la cola correspondiente al candidato a
              la cola de votos reportados
2         //(Entrega un mensaje en caso de existir ya el voto en la
              cola por ejemplo, en caso de fraude)
3         public boolean reportarVoto(Candidato candidato, int
              idVoto) {
4
5             for (Voto yaReportado : votosReportados) { /**
                  for-each (for mejorado) para recorrer la cola */
6                 if (yaReportado.getidVoto() == idVoto) {
7                     System.out.println("El voto (ID) " + idVoto +
                          " ya esta reportado.");
8                     return false;
9                 }
10            }
11
12            // 2) Pedimos al candidato que quite el voto de su cola
13            Voto v = candidato.removerVotoPorId(idVoto);
14            if (v == null) {
15                System.out.println("No existe el voto (ID) " +
                      idVoto +
16                        " en la cola del candidato " +
17                        candidato.getId());
18                return false;
19            }
20
21            // 3) Lo agregamos a votosReportados
22            votosReportados.add(v);
23            System.out.println("Voto (ID) " + idVoto + " reportado
                  exitosamente.");
24            return true;
25        }
```

Código 15: Método reportarVoto

- El método `obtenerResultados()` retorna un mapa con el conteo de votos de cada candidato. Usa un `StringBuilder` que es una clase de Java diseñada para construir cadenas de texto de manera eficiente y esto permite concatenar el nombre de candidato con su partido y los votos que logró, evitando así la sobrecarga de crear múltiples cadenas intermedias al concatenar.
- Recorre la `listaCandidatos`(11) con un *for-each* y encadena cada método `append` que añade la representación en texto de su argumento al final de lo que ya contiene el buffer.
- Por último retorna un `sb.toString()` que convierte todo el contenido acumulado en el `StringBuilder` en un objeto `String` inmutable.

```
1         //Retorna un mapa con el conteo de votos por Candidato.
```

```
2         public String obtenerResultados() {
3             //StringBuilder es una clase de Java diseniada para
                  construir cadenas de texto de manera eficiente
4
5             StringBuilder sb = new StringBuilder("=== Resultados
                  de la Eleccion ===\n");
6             for (Candidato c : listaCandidatos) {
7                 //El metodo append aniade la representacion en
                      texto de su argumento
8                 //(puede ser otro String, un numero, un objeto,
                      etc.) al final de lo que ya contiene el buffer.
9                 sb.append("Candidato ")
10                        .append(c.getNombre())
11                        .append(" (")
12                        .append(c.getPartido())
13                        .append("): ")
14                        .append(c.getTotalVotos());
15                        if(c.getTotalVotos() == 1){
16                            sb.append(" voto\n");
17                        } else{
18                            sb.append(" votos\n");
19                        }
20            }
21            //sb.toString convierte todo el contenido acumulado en
                  el StringBuilder en un objeto String inmutable
22            return sb.toString();
23        }
24    }
```

Código 16: Método obtenerResultados

2.1.5. Main

En el Main probamos el código creando objetos de tipo `Candidato` y `Votante`. Se añaden los candidatos a la `listaCandidatos` y registramos lo votos. Al momento de registrar los votos necesitamos crear una variable llamada `contadorDeVotos` para asignar un ID único a cada voto.

Imprimimos los resultados usando `urna.obtenerResultados()` para compararlos más tarde.

Probamos a reportar un voto, para esto intentamos duplicar un voto tratando de votar nuevamente con el mismo votante y lo reportamos con `reportarVoto()`(15).

```
1         public static void main(String[] args) {
2         int contadorDeVotos = 0; // auxiliar para saber el numero
              correspondiente del voto registrado
3
4         System.out.println("\n"+"---------------------------------------");
5         Electo electo = new Electo();
6         // 1) Creamos la urna
7         UrnaElectoral urna = electo.new UrnaElectoral();
```

```
8
9           // 2) Aniadimos algunos candidatos
10          Candidato c1 = electo.new Candidato(1, "Chamelo Gas",
                "Partido A");
11          Candidato c2 = electo.new Candidato(2, "Bob Patinho",
                "Partido B");
12          Candidato c3 = electo.new Candidato(3, "Kanye West",
                "Partido C");
13          // Como estamos dentro de la clase Electo, podemos acceder
                a la lista privada:
14          urna.listaCandidatos.add(c1);
15          urna.listaCandidatos.add(c2);
16          urna.listaCandidatos.add(c3);
17
18          // 3) Creamos votantes
19          Votante v1 = electo.new Votante(1, "Juan",  false);
20          Votante v2 = electo.new Votante(2, "Maria", false);
21          Votante v3 = electo.new Votante(3, "Jose", false);
22
23          // 4) Registramos votos
24          contadorDeVotos++;
25          System.out.println("Voto (ID) " + contadorDeVotos +  " del
                votante " + v1.getNombre()
26                  + " registrado: " + urna.registrarVoto(v1, 1)); //
                        true
27          contadorDeVotos++;
28          System.out.println("Voto (ID) " + contadorDeVotos +  " del
                votante " + v2.getNombre()
29                  + " registrado: " + urna.registrarVoto(v2, 2)); //
                        true
30          contadorDeVotos++;
31          System.out.println("Voto (ID) " + contadorDeVotos +  " del
                votante " + v3.getNombre()
32                  + " registrado: " + urna.registrarVoto(v3, 2));
33
34          //Imprimimos para ver los resultados antes de reportar y
                anular votos
35          System.out.println("---------------------------------------"+"\n"
                + urna.obtenerResultados()
36                  + "---------------------------------------"+"\n" );
37
38          /** Intento de voto duplicado por parte de Juan (v1),
                candidato 1 (c1), id (1) */
39          System.out.println(v1.getNombre() + " esta votando otra
                vez: " + urna.registrarVoto(v1, 1) + "\n"); // false
40
41          // 5) Reportamos (anulamos) un voto
42          urna.reportarVoto(c1, 1); // mueve el voto con idVoto=1 de
                c1 a votosReportados
43          urna.reportarVoto(c1,1); //id del voto de Juan es 1, pero
                ya se habia reportado anteriormente
44          System.out.println("\n" + v3.getNombre() + " esta votando
```

```
            otra vez: " + urna.registrarVoto(v3, 3));
45          urna.reportarVoto(c2, 3); //id del voto de Jose es 3
46
47          // 6) Imprimimos resultados
48          System.out.println("----------------------------------------"+"\n"
               + urna.obtenerResultados()
49                  + "----------------------------------------"+"\n" );
50
51          /** Tanto 'Juan' como 'Jose' intentaron votar de nuevo, lo
               que ocasiona que se anulen ambos votos
52           * Por lo tanto, el partido A: 1 voto -> 0 votos y Partido
                B: 2 votos -> 1 voto
53           */
54      }
```

Código 17: Main

3. Análisis

3.1. Complejidad Temporal

A continuación, se presenta una tabla en la cual se representa la complejidad temporal en notación Big-O, esta se realizara para los principales métodos del código, estos son: `registrarVoto()`(14), `obtenerResultados()`(16), `reportarVoto()`(15).

Método	Análisis	Complejidad Big-O
`registrarVoto()`	-Verificar si el votante ya voto O(1). -Recorrer la lista de candidatos O(n). -Insertar en la cola de votos del candidato O(1). -Insertar en la pila de historial O(1). -Indicar si el votante ya voto o no O(n).	O(n)
`reportarVoto()`	-Buscar en la cola de votos reportados O(m) -Remover voto de la cola de votos del candidato -Insertar en la cola de votos reportados O(1)	O(m+k) O(k)
`obtenerResultados()`	-Recorrer la lista de candidatos O(n) -Por candidato, se tiene el tamaño de su cola O(n) Construir un StringBuilder O(n)	O(n)

Tabla 1: Tabla Complejidad Temporal

Como acotación debemos indicar que:

- n: Cantidad de candidatos.

- m: Cantidad de votos que fueron reportados.
- k: Cantidad de votos de candidato individualmente.

Lo que podemos apreciar de la Tabla 1 es que el método `registrarVoto()` es lineal, ya que debe recorrer las lista completa para buscar al candidato elegido por el votante, es decir O(n). El método `reportarVoto()` es proporcional a la cantidad de votos que fue reportado mas la cantidad de votos que tiene el candidato, por ende su complejidad es O(m+n). Y por último el método `obetenerResultados()`, de igual forma recorre todos los candidatos para mostrar el conteo de los votos, O(n).

3.2. Gestión de Memoria

Se indica en el problema que:

- Cada voto ejercido ocupa un total de 64 bytes.
- El sistema admite hasta 10 millones de votantes.

Por ende, se puede considerar que cada votante emite un único voto; también se debe tener en cuenta que el voto se almacena dos veces, por ende se duplica su cantidad: una de ellas en la cola de votos recibidos del candidato (`votosRecibidos()`), mientras que la otra es cuando se almacenan en la pila del historial `historialVotos()`.

Por ende, se tiene que:

1. Un voto ocupa una memoria de 64 bytes.
2. Existen dos votos almacenados por voto, es decir:

   $$64 \times 2 = 128$$

   Por lo tanto son 128 bytes por cada votante.
3. Hay 10 millones de votantes, por lo tanto se deben multiplicar por la cantidad de votantes para obtener la memoria total que es utilizada, de la siguiente manera:

   $$10,000,000 \times 128 = 1,280,000,000$$

   En consecuencia, se tiene que la memoria total es de 1.280.000.000 bytes, lo que equivale a 1.280 MB, a lo que a su vez equivale a 1.28 GB aproximadamente.

3.3. Propuesta de Mejora

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

En el problema solucionado actual, los votantes y los candidatos pertenecen a una única urna, la cual no identifica a los votantes por facultades.

Por lo tanto, se propone que para el manejo de una votación donde participen varias facultades, es organizar los votos ejercidos y los candidatos a elegir por facultad, es decir, en vez de tener una única urna que abarque a todas las facultades, dividir esta y hacer que cada facultad tenga su propia urna electoral.

Para ello se debe crear una nueva clase a la que se llamaría como "`class Facultad`", la cual debe contener dos atributos, uno de ellos con el nombre de la facultad de tipo `String`, y el otro que represente una urna electoral de tipo `UrnaElectoral`. De esta manera, cada facultad obtendría su propia urna, sus propios candidatos y también su propio historial de votos.

De esta forma, el sistema de las elecciones sería más ordenado, ya que cada facultad tendría su propio sistema independiente, evitando equivocaciones o confusiones, entre los candidatos que existen y los votantes de las distintas facultades, también permitiría que los resultados sean más expeditos de entregar.

4. ¿Listas Enlazadas o Arreglos?

4.1. Ventajas

- ***Listas Enlazadas***

  1. **Insertar y eliminar:** Realizar inserciones o eliminar elementos en las listas enlazadas es muchísimo mas rápido siendo en notación Big-O de O(1).
  2. **Redimensionamiento:** En estas, no se requiere redimensionar memoria, en cambio los arreglos la deben duplicar al momento de llenarse al máximo.
  3. **Eficiencia:**Al requerir una variabilidad en la cantidad de elementos, estas son más eficientes en memoria.

- ***Arreglos***

  1. **Acceso a elementos:** Para acceder a un elemento en especifico es muchísimo más rápido, ya que estos están almacenados contiguamente, siendo en notación Big-O de O(1).

2. **Rendimiento:** En estos, los elementos están guardados de manera contigua por lo que permite que los procesadores más modernos puedan obtener la información mucho mas rápidamente.

4.2. Desventajas

- ***Listas Enlazadas***

  1. **Acceso a elementos:** Para acceder a un elemento en especifico se va a demorar O(n), ya que debe recorrer nodo por nodo.
  2. **Uso de memoria:** En una lista enlazada cada nodo almacena un dato y un puntero al siguiente nodo, por lo que a comparación de un arreglo, sí ocupa más memoria.
  3. **Rendimiento:** Este tipo de estructura de datos, los elementos no están guardados de manera continua por lo que utilizan mayor memoria cache, y su rendimiento es menor.

- ***Arreglos***

  1. **Insertar y eliminar:** Realizar inserciones o eliminar elementos en los arreglos es muchísimo mas complejo y lento, ya que se requiere realizar muchos movimientos de estos, siendo en notación Big-O de O(n).
  2. **Redimensionamiento:** Cuando un arreglo se completa a su máxima capacidad este crea un nuevo arreglo más grande, y los elementos se copian, esto hace que sea todo mas lento.
  3. **Flexibilidad:** En situaciones donde varían mucho la cantidad de elementos no es muy aconsejable el uso de arreglos, además de tener que predecir un tamaño aproximado, para evitar que se complete a su totalidad.

En consecuencia, dadas las ventajas y desventajas de listas enlazadas y de arreglos, en este problema en especifico donde se realizará una votación, es mas aconsejable el uso de listas enlazadas, ya que se requiere de un sistema mucho mas dinámico que un arreglo, ya que también es dinámico pero de manera limitada al necesitar de una copia de sus elementos, por ende sería menos eficiente el uso de estos últimos. Si bien en un arreglo acceder a posiciones especificas es mas sencillo, las inserciones y eliminaciones se necesitaría de una mayor cantidad de datos en la memoria.

5. Conclusión

Este laboratorio se permitió principalmente conocer más sobre el funcionamiento y el uso de estructuras de datos fundamentales como lo son: listas enlazadas, pilas y colas. También ejercitar en un contexto más realista, el diseño de clases y eficiencia de la memoria, escogiendo la estructura de datos más adecuada.

Se procedió a crear un código mediante el cual se puede dar solución al problema planteado, el cual garantiza el desarrollo de este de manera eficiente, permitiendo que los resultados sean claros y precisos.

Durante el desarrollo de este laboratorio se logro comprobar que el uso de listas enlazadas pueden aportar con una mayor flexibilidad para manejar datos tan variables como lo son en una votación, por ende también se comprendió la razón de por qué en este caso era más eficiente el uso de estas.

No solo se logró comprender sobre el uso de listas enlazadas, sino que también se logró entender y aprender sobre la ocupación correcta de pilas y colas, las cuales entregan de manera más sencilla la gestión del historial de votos y los votos reportados.

Otro punto importante a considerar es que se logró entender sobre la complejidad temporal en notación Big-O, el cual manejea una complejidad de O(n), adecuado para el problema planteado.

Además se abarco el problema, desde un punto de vista mucho más amplio y suponiendo un escenario mucho más robusto, en cual abarcaría muchos más componentes, y por lo tanto cambios en el código propuesto originalmente, realizando modificaciones en él para poder adaptarse a una situación en donde se abarcaría una mayor cantidad de datos.

Para finalizar, como posible mejora futura, se propone el uso de una votación electrónica, de esta manera se podrían generar nuevos procesos que generen una votación mucho más segura y eficiente, manteniendo la confidencialidad de los votantes y una transparencia en sus resultados.

Referencias

[1] *Repositorio en Github*. 2025. URL: https://github.com/Jrgitx/Informe_Lab2_S2.git.