Práctica 8

Colas

Matú Hernández Diana

Rojo Mata Daniel

Justificación de los algoritmos

- **Método constructor.**

```
1    public Cola() {
2      inicio = null;
3      fin = null;
4      longitud = 0;
5    }
```

El algoritmo inicializa la estructura de datos `Cola` sin ningún elemento. El algoritmo es correcto porque cumple el propósito de crear una cola en donde el inicio es `null`, el fin es `null` y su longitud, por ende, es 0.

- **Método encolar**

```
1    public void encolar( T elemento ) {
2      NodoCola nuevoNodo = new NodoCola(elemento);
3      if (esVacia()== true) {
4        inicio = nuevoNodo;
5        fin = nuevoNodo;
6        longitud++;
7      }
```

```
8       else {
9         fin.siguiente = nuevoNodo;
10        fin = nuevoNodo;
11        longitud++;
12      }
13    }
```

Agregar un elemento en el inicio de la cola implica asignar un espacio de memoria para el elemento a apilar, en este caso, el nuevo elemento se denomina como `nodoNuevo` (línea 2). El bloque `if 3-7` toma como caso la cola vacía, asignando como `inico` y `fin` a `nuevoNodo` e incrementando la longitud en 1.

En el bloque `else 8-12` se ajusta el nodo siguiente del `fin` como *nodoNuevo* (`fin.siguiente = nuevoNodo`) y se aumenta la longitud de la cola en una unidad.

El algoritmo es correcto porque añade un elemento al final de la cola sin alterar los elementos ya existentes. La cola sigue manteniendo su estructura *FIFO (First In, First Out)* después de la operación, pues el fin anterior se actualiza, además de que se revisa el caso en que se tiene una cola vacía tratándolo de manera adecuada.

- **Método desencolar**

```
1   public void desencolar() throws Exception {
2     if (esVacia()== true) {
3       throw new Exception("La cola esta vacia");
4     }
5     else {
6       if(this.longitud == 1){
7         inicio = null;
8         fin = null;
9         longitud--;
10      }
11      else {
```

```
12          NodoCola actual = inicio;
13          inicio = inicio.siguiente;
14          actual.siguiente = null;
15          actual = null;
16          longitud--;
17        }
18      }
19    }
```

El bloque `if 2-4` revisa el caso en que la cola es vacia. En tal caso, se arroja una excepción. El bloque `else 5-10` revisa la situación en que la pila no es vacia y contiene exactamente un elemento.

Cuando lo anterior ocurre, a `inicio` y `fin` se les asigna el valor `null` y disminuye en 1 la longitud de cola. Por otra parte, dentro del bloque `else 11-17` se crea un nuevo nodo llamado `actual` y se establece la referencia como `incio` (`actual = inicio`). Se establecen las referencias *inicio* = *inicio.siguiente*; (línea 13) para asegurarse de que el nodo posterior al inicial ahora sea el `inicio` de la cola. Luego, se asigna `actual.siguiente` y `actual` como `null` para liberar la memoria ocupada por el nodo desencolado. Finalmente la longitud de la pila disminuye en una unidad.

El algoritmo es correcto porque realiza las operaciones de desencolado de manera correcta además de que gestiona la memoria de manera precisa al modificar como *null* las referencias en 14 y 15. Además proporciona mensajes de error significativos para el caso en que se introduzca una cola vacía. Así pues, cumple con las reglas fundamentales de las colas y asegura que la estructura de datos se maneje de forma coherente y segura en todas las situaciones posibles.

- **Método darElementoInicio**

```
1   public T darElementoInicio() throws Exception {
2     if(this.longitud == 0){
3       throw new Exception ("La cola esta vacia");
4     }
```

```
5       else
6         return this.inicio.elemento;
7     }
```

El bloque $if\ 2-4$ revisa el caso en que la cola es vacía En tal caso, se arroja una excepción. El método se encarga de regresar el elemento almacenado en el `inicio` cuando la cola no es vacía. El método es correcto pues se encarga de revisar las situaciones en donde la cola es vacía o no, y regresa de manera correcta (al acceder al campo) el elemento correspondiente.

- **Método esVacia**

```
1     public boolean esVacia() {
2         if(this.longitud == 0){
3           return true;
4         }
5         else
6           return false;
7     }
```

El bloque de código `if 2-4` revisa el caso en que la longitud de la cola sea cero, lo que equivale a decir que la cola es vacia. En este caso se retorna el booleano *true* y en su defecto, cuando la longitud de la laco es distinta de cero, se retorna *false* pues indica que la cola no es vacía.
El algoritmo es correcto pues valida los casos que se requieren accediendo de manera adecuada a los campos de la EDD y retornando los valores booleanos adecuados.

- **Método darLongitud**

```
1       public int darLongitud() {
2           return this.longitud;
3       }
```

La línea 2 regresa el campo *longitud* accediendo a éste mediante el operador $\cdot$
El algoritmo es correcto pues solo se está accediendo a un campo de la EDD de forma adecuada.

Complejidad en tiempo y espacio

- **Método constructor.**

```
1    public Cola() {
2      inicio = null;  //1
3      fin = null;   //1
4      longitud = 0;  //1
5    }
```

$$T(n) = 1$$

$$M(n) = 0$$

- **Método encolar.**

```
1    public void encolar( T elemento ) {
2      NodoCola nuevoNodo = new NodoCola(elemento);
3      if (esVacia()== true) {  //1
4        inicio = nuevoNodo;  //1
5        fin = nuevoNodo;  //1
6        longitud++;  //2
7      }
8      else {
9        fin.siguiente = nuevoNodo;  //1
10       fin = nuevoNodo;  //1
11       longitud++;  //2
12     }
13   }
```

$$T(n) = 5$$

$$M(n) = 1$$

- **Método desencolar.**

```
1   public void desencolar() throws Exception {
2     if (esVacia()== true) {  //1
3       throw new Exception("La cola esta vacia");
4     }
5     else {
6       if(this.longitud == 1){  //1
7         inicio = null;  //1
8         fin = null;  //1
9         longitud--;  //2
10      }
11      else {
12        NodoCola actual = inicio;  //1
13        inicio = inicio.siguiente;  //1
14        actual.siguiente = null;  //1
15        actual = null;  //1
16        longitud--;  //2
17      }
18    }
19  }
```

$$T(n) = 6$$

$$M(n) = 0$$

- **Método darElementoInicio**

```
1   public T darElementoInicio() throws Exception {
2     if(this.longitud == 0){  //1
```

```
3         throw new Exception ("La cola esta vacia");
4       }
5       else
6         return this.inicio.elemento;
7     }
```

$$T(n) = 1$$

$$M(n) = 0$$

- **Método esVacia**

```
1     public boolean esVacia() {
2         if(this.longitud == 0){ //1
3           return true;
4         }
5         else
6           return false;
7     }
```

$$T(n) = 1$$

$$M(n) = 0$$

- **Método darLongitud.**

```
1       public int darLongitud() {
2           return this.longitud;
3       }
```

$$T(n) = 0$$

$$M(n) = 0$$

Observación

Se puede observar que en todos los métodos $T(n) = a$ y $M(n) = b$, donde $a, b \in \mathbb{N}$, esto sugiere que las funciones pertenecen a $O(1)$, es decir, las funciones anteriores son constantes en tiempo y en almacenamiento.

Pseudocódigos

- **Método Constructor**
- **Entrada:** Ninguna.
- **Salida:** Objeto de tipo Cola.

```
1    function Colla()
2        inicio <- null
3        fin <- null
4        longitud <- 0
5    end function
```

- **Método encolar**
- **Entrada:** Una cola C y un elemento e que se encuenta en T
- **Salida:** Cola C tal que C.fin.elemento = e

```
1    function encolar(C, e)
2        nuevoNodo <- *NodoCola(e)
3        if(esVacia(C)) then
4            inicio <- nuevoNodo
5            fin <- nuevoNodo
6            longitud++
7        else
8            fin.siguiente <- nuevoNodo
```

```
9               fin <- nuevoNodo
10              longitud <- longitud + 1
11          end if-else
12      end function
```

- **Método desencolar**
- **Entrada:** Cola C
- **Salida:** La cola C sin su primer elemento

```
1       function desencolar(C)
2           if(esVacia(C))
3               ERROR "Cola vacia"
4           else
5               if(longitud <- 1) then
6                   incio <- null
7                   fin <- null
8                   longitud <- 0
9               else
10                  actual <- inicio
11                  inicio <- inicio.siguiente
12                  actual.siguiente <- null
13                  actual <- null
14                  longitud <- longitud - 1
15          end if-else
16      end function
```

- **Método darElementoInicio.**
- **Entrada:** Cola
- **Salida:** Elemento de tipo T en el inicio de la cola.

- **Excepción:** Lanza una excepción si la cola está vacía.

```
1       function darElementoInicio(C)
2           if (C.longitud = 0)
3               entonces ERROR "Cola vacia"
4           else
5               return C.inicio.elemento
6           end if-else
7       end function
```

- **Método esVacia**
- **Entrada:** Cola C.
- **Salida:** `true` si la cola está vacía, `false` en caso contrario.

```
1       function esVacia(C)
2           if (P.longitud = 0)
3               entonces true
4           else
5             entonces false
6           end if-else
7       end function
```

- **Método darLongitud**
- **Entrada:** Cola
- **Salida:** Numero entero

```
1       function darLongitud(C)
2           return C.longitud
3       end function
```