Estructuras de Datos 2022-1
Práctica 2: Listas

Pedro Ulises Cervantes González
confundeme@ciencias.unam.mx

Emmanuel Cruz Hernández
emmanuel_cruzh@ciencias.unam.mx

Yessica Janeth Pablo Martínez
yessica_j_pablo@ciencias.unam.mx

América Montserrat García Coronado
ame_coronado@ciencias.unam.mx

Adrián Felipe Vélez Rivera
adrianf_velez@ciencias.unam.mx

Fecha límite de entrega: 15 de octubre de 2021
Hora límite de entrega: 23:59:59 hrs

1. Objetivo

Una lista es una estructura de datos que almacena datos en forma ordenada, uno después de otro. La estructura tiene variedad de aplicaciones reales como una lista de compras, una lista de reproducción, una lista de productos, entre otras.

Se espera explotar el mayor potencial posible de una lista dentro de una aplicación, tomando en cuenta que la complejidad en tiempo y espacio se aproveche de mejor forma en la aplicación de la estructura.

2. Actividad

Dada la interfaz *TDAList* implementa sus operaciones basado en una lista con dobles referencias. Estas operaciones deben estar implementadas en un clase llamada *DoubleLinkedList*.

Sea n la cantidad de elementos de una lista.

2.1. Agregar elementos *(1.5 puntos)*

Implementa el método `add(int i, T e)` con a lo más *(n+1)/2* iteraciones.

2.2. Limpiar la lista *(0.5 puntos)*

Implementa el método `clear()` en tiempo O(1).

2.3. ¿Está contenido? *(0.5 puntos)*

Implementa el método `contains(T e)` con a lo más *(n+1)/2* iteraciones.

2.4. Obtener *(1 punto)*

Implementa el método `get(int i)` con a lo más *(n+1)/2* iteraciones.

2.5. ¿Es vacía? *(0.5 puntos)*

Implementa el método `isEmpty()` en tiempo O(1).

2.6. Remover *(1.5 puntos)*

Implementa el método `remove(int i)` con a lo más *(n+1)/2* iteraciones.

2.7. Longitud *(0.5 puntos)*

Implementa el método `size()` en tiempo O(1).

2.8. Reversa *(1 punto)*

Implementa el método `revert()` en tiempo O(n).

2.9. Corte *(1 punto)*

Implementa el método `cut(boolean side)` con a lo más *(n+1)/2* iteraciones.

2.10. toString *(0.5 puntos)*

Implementa el método `toString()`, dando una representación de la lista. Puedes representarla como mejor te guste.

2.11. Iterador *(1 punto)*

Implementa un iterador para la lista. Basta con la implementación de *next()* y *hasNext()*.
Este iterador debe ser regresado con el método llamado `listIterador()`.

2.12. Main *(0.5 puntos)*

Crea un menú en el método *main* de la clase *DoubleLinkedList* donde pongas a prueba cada uno de los métodos implementados. El menú debe tener las siguientes opciones:

1. Agregar una cadena a la lista
2. Eliminar una cadena a la lista
3. Limpiar la lista
4. Verificar si un elemento está en la lista
5. Obtener un elemento de la lista
6. Verificar si la lista está vacía
7. Obtener la longitud de la lista
8. Obtener la reversa de la lista
9. Cortar la lista
10. Mostrar la lista
11. Salir del menú

3. Recursos de apoyo

- Operaciones sobre listas (representación gráfica): `https://docs.google.com/presentation/d/1ABwryIv4IwNPE3EqD7zDlbkDlVKzLb6jrXBmH0cYB1E/edit?usp=sharing`
- Explicación de las operaciones sobre listas: `https://youtu.be/lyqRbcxhXY4`
- Eliminación de un elemento de una lista simple: `https://www.youtube.com/watch?v=VoPRnfn06VI`
- Insertar elementos en una lista simple: `https://youtu.be/hIpW1Djw1Qk`

4. Punto Extra

Implementa sólo una de las siguientes opciones:

- ArrayList: una lista basada en arreglos.
- OrderdList: una lista ordenada de forma ascendente.

Cualquiera de las dos opciones debe implementar la interfaz *TDAList*.
NOTA: Para el caso de *OrderdList*, el método `add(int i, T e)` no utilizará el parámetro i, ya que no es necesario, así que cualquier valor que reciba en i será ignorado por la implementación.

5. Reglas Importantes

- No se recibirán prácticas en las que estén involucrados más de dos integrantes.
- Cumple con los lineamientos de entrega.
- Todos los archivos deberán contener nombre y número de cuenta.
- Tu código debe estar comentado. Esto abarca clases, atributos, métodos y comentarios extra que consideres necesarios.
- Utiliza correctamente las convenciones para nombrar variables, constantes, clases y métodos.
- El programa debe ser robusto.
- En caso de no cumplirse alguna de las reglas especificadas, se restará 0.5 puntos en tu calificación obtenida.
- **Queda estrictamente prohibido usar una implementación de listas hecha por Java, de lo contrario la actividad se evaluará sobre 0**.