UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ciencias

Manejo de Datos

Prof. Jessica Santizo Galicia

Ayud: Fernando Ortega Toraya

Tarea02

Fecha de salida: Octubre 6, 2023
Fecha de entrega: Octubre 18, 2023 22:00

Objetivo: El alumno practicará la creación de tipos abstractos de datos y la implementación de algunos métodos que le permitirán desarrollar su lógica de programación.

1. **Ejercicio01 (1pto):**
   Justifica con tus propias palabras por qué la complejidad de Mergesort es de 0(nlog n) para el mejor caso, caso promedio y peor caso.

2. **Ejercicio02 (2pts):**
   Investigar cuál es el algoritmo de ordenamiento que usa el método sort de Python. Explicar cuál es su complejidad computacional.

3. **Ejercicio03 (2pts):**
   Escribir un programa de consulta de teléfonos. Leer un conjunto de datos de 1000 nombres y números de teléfono de un archivo que contiene los números en orden aleatorio. Las consultas han de poder realizarse por nombre y por número de teléfono. Para realizar la consulta del número, primero ordena los datos haciendo uso de un algoritmo de ordenamiento recursivo (debes de utilizar los códigos vistos en clase) y posteriormente realiza la búsqueda binaria para encontrar el elemento.

4. **Ejercicio04 (5pts):**
   Implementar un tipo abstracto de datos (TDA) el cuál deberá ser una colección de datos ordenados. Utiliza una lista [] para esta implementación. La clase abstracta que se debe de implementar deberá contener los siguientes métodos:

   - **esta_vacia**: Devuelve True si no hay elementos en la colección y False en otro caso
   - **limpiar**: Limpia la colección, es decir la deja sin elementos. El método debe de ser de orden O(1)
   - **tamanio**: Devuelve el número total de elementos en la colección. El método debe de ser de orden O(1)
   - **eliminar_min**: Elimina de la colección al elemento más pequeño. (Todas las presencias de ese elemento).
   - **eliminar_max**: Elimina de la colección al elemento más grande. (Todas las presencias de ese elemento).
   - **agregar(elemento)**: Se debe de agregar un elemento a la colección, y se debe de mantener la colección ordenada de menor a mayor.
     - *Ejemplo*: Si la colección es: coleccion = [2,3,5,5,7,10,13,15,17,17, 30, 30]
       coleccion.agregar(4) deberá agregar el número 4 en el lugar correspondiente, manteniendo el orden.
       La colección deberá quedar de la siguiente forma:
       coleccion = [2,3,**4**,5,5,7,10,13,15,17,17,30,30]

- **buscar_elemento(elemento)**: Devuelve el índice en dónde se encuentra **la primera presencia** del elemento en la colección ó -1 si no se encuentra.
  La colección no se modifica y permanece intacta. (Usar la búsqueda binaria para este método).

- **cuantos(elemento)**: Devuelve la cantidad de veces que aparece un elemento dentro de la colección.

  - *Ejemplo*: Si la colección es: coleccion = [2,3,5,5,7,10,13,15,17,17,30,30]
    coleccion.cuantos(17) deberá regresar: 2
    coleccion.cuantos(-1) deberá regresar: 0
    coleccion.cuantos(3) deberá regresar: 1

- **obtener_subcoleccion(elemento)**: Regresa una subcolección que contiene a los elementos a partir del primer_elemento que se pasa como parámetro, si el elemento no se encuentra el método lanza una excepción indicando que no existe dicha subcolección.

  - *Ejemplo*: Si la colección es: coleccion = [2,3,5,5,7,10,13,15,17,17,30,30]
    coleccion.obtener_subcoleccion(17) deberá regresar: [17,17,30,30]

- **obtener_subcoleccion_rango(primer_elemento, segundo_elemento)**: Regresa una subcolección que empiece en la primera posición de primer elemento y finalice en la última posición de segundo_elemento. Si primer_elemento o segundo_elemento no se encuentran, debes lanzar una exepción indicando con un mensaje que el elemento no se encuentra dentro de la colección.

  - *Ejemplo*: Si la colección es: coleccion = [2,3,5,5,7,10,13,15,17,17,30,30]
    coleccion.obtener_subcoleccion_rango(5,17) deberá regresar: [5,5,7,10,13,15,17,17]

  - *Ejemplo*: Si la colección es: coleccion = [2,3,5,5,7,10,13,15,17,17,30,30]
    coleccion.obtener_subcoleccion_rango(-5,40) el programa deberá mostrar un mensaje similar a: "No se puede obtener la subcolección por que no se encuentra el -5 y 40"

- **subcoleccion_sin_repeticion()**: Regresa una subcolección de la colección original con los elementos sin repetición.
  - *Ejemplo*: Si la colección es: coleccion = [2,3,5,5,7,10,13,15,17,17,30,30]
    coleccion.subcoleccion_sin_repeticion() deberá regresar [2,3,5,7,10,13,15,17,30]

- **reemplazar(elemento, nuevo)**: Remplaza todas las presencias de elemento por nuevo.

  - *Ejemplo*: Si la colección es: coleccion= [2,3,5,5,7,10,13,15,17,17,30,30]
    coleccion.remplazar(elemento,30,40) deberá regresar coleccion = [2,3,5,5,7,10,13,15,17,17,40,40]
    Es importante tomar en cuenta que, al mandar a llamar al método remplazar, la colección debe seguir ordenada.
    Si la colección es: coleccion = [2,3,5,5,7,10,13,15,17,17,30,30]
    coleccion.remplazar(elemento,5,8) deberá regresar coleccion = [2,3,7,8,8,10,13,15,17,17,30,30]

- **__eq__(otra)**: Devuelve true si dos colecciones son iguales, dos colecciones son iguales si contienen los mismos elementos y la misma cantidad de ellos: Se compara la colección que manda a llamar al método con la que se pasa como parámetro(otra). Usar el método isinstance() para validar que se están comparando dos colecciones.

  - *Ejemplo*:
    Si coleccion_a = [1,2,3] y coleccion_b = [1,2,3] el método regresa True
    Si coleccion_a = [1,8,6,3] y coleccion_b = [1,9,3] el método regresa False

- **imprimir_ascendente()**: Este método deberá de imprimir a los elementos de la colección de menor a mayor, sin incluir los None

- **imprimir_descendente()**: Este método deberá de imprimir los elementos de la colección de mayor a menor, sin incluir los None

***NOTA*: Sólo puedes usar los métodos list.insert(i, x) y list.append(x). NO puedes usar los demás métodos existentes de esta estructura de datos. Tampoco la estructura de datos Set.**

Crea una clase de prueba que permita probar la implementación de cada uno de los métodos anteriores. Asume que siempre serán listas de números.

Forma de entrega:

- El programa se encuentra en Git.
- El líder del proyecto deberá crear un repositorio privado en Github y subir su proyecto ahí, agregando a los colaboradores que son los integrantes del equipo, a Fernando, (enviar invitación al repositorio hasta la fecha de etrega). Tu rama **main** deberá contener los **4** ejercicios, el archivo .gitignore y tu README.md el cuál deberá contener a los integrantes de tu equipo por orden alfabético, **número de equipo** y funcionalidad de sus programas.
- Deberás compartir el link de tu repositorio a través de classroom para poder hacer la revisión de la práctica.
- La tarea se entrega en equipos de 4 personas. No se aceptan tareas individuales.
- Sigue los lineamientos que se encuentran en la página del curso.
- No olvides agregar comentarios a tus programas y seguir las buenas prácticas de Programación.

Esta es la estructura que deberá tener tu proyecto en Git:

Tarea02
- **Ejercicios**
  - **Ejercicio01**
    - ⋮
  - **Ejercicio02**
    - ⋮
  - **Ejercicio03**
    - ⋮
  - **Ejercicio04**
    - ⋮
  - **.gitignore**
  - **README.md**