TD3: Algoritmos y Estructuras de Datos

Trabajo Práctico "Editor con Resaltado"

10 de Junio

- El TP debe realizarse en grupos de hasta 3 personas.
- La plazo de entrega es hasta el 28 de junio inclusive.
- Se evaluará no sólo la corrección técnica de la solución propuesta sino también la claridad del código escrito.

Descripción del problema

Se necesita proveer la implementación para el tipo de datos `EditorResaltado` que modela un editor de documentos de texto.

Complementando la funcionalidad básica de ingresar y eliminar texto, se requiere que este editor permita realizar *comentarios* sobre fragmentos del texto. Cada comentario aplica a un *rango* contiguo de palabras completas del texto. Los comentarios pueden solaparse en el texto, por lo que cada palabra puede estar alcanzada por cero o más de los comentarios realizados.

El editor debe resolver de manera eficiente la consulta de qué comentarios son pertinentes a una posición dada y las operaciones de edición del texto y de gestión de comentarios.

Consigna

1. Definir una estructura de representación en el archivo `EditorResaltado.h` que permita cumplir los Requerimientos de complejidad.
   - Utilizar clases provistas por la *Biblioteca Estándar de C++* (`std`) teniendo en cuenta sus órdenes de complejidad.
   - No es necesario (**y no se recomienda**) diseñar estructuras manejando memoria dinámica de manera manual (`new`, `delete`).
2. Escribir un comentario en `EditorResaltado.h` el **invariante de representación** de la clase (es decir, las condiciones que debe cumplir la estructura para ser válida) de **dos maneras**:
   - en castellano;
   - en lógica formal como predicado *Rep(e: estr)*[1]
3. Escribir en el archivo `EditorResaltado.cpp` la implementación de los métodos respetando los Requerimientos de complejidad y el **invariante de representación** especificado. No está permitido modificar la interfaz pública (sección `public`) de la clase.

[1] Se recomienda utilizar `forall`/`exists`/`&&`/`||` para denotar $\forall$/ $\exists$/$\wedge$/$\vee$, respectivamente. También, a modo de ejemplo, puede utilizar `sum{i=k}{n}{i}` para denotar $\sum_{i=k}^{n} i$.

4. Comentar claramente en el código las complejidades de **peor caso** de los métodos implementados, incluyendo los métodos que no tienen requisito de complejidad. Pueden hacerlo comentando la complejidad *O(...)* de cada línea y agregando al pie del algoritmo la cuenta de complejidad total (aplicando álgebra de órdenes) y cualquier justificación/aclaración extra que sea necesaria para las complejidades anotadas línea a línea. **No se pide ninguna justificación formal extra**.

Interfaz de la clase

```
typedef int id_comm;
class EditorResaltado {
  public:
  // Constructores
  EditorResaltado();
  // Método estático auxiliar para construir editores de forma conveniente
  static EditorResaltado con_texto(const string& texto);

  // Observadores
  unsigned longitud() const;
  const string& palabra_en(unsigned pos) const;
  const string& texto_comentario(id_comm id) const;
  const set<id_comm> & comentarios_palabra(unsigned pos) const;

  // Modificadores
  void cargar_texto(const string& archivo_texto, const string& archivo_comentarios);
  void insertar_palabra(const string& palabra, unsigned pos);
  void borrar_palabra(unsigned pos);
  id_comm comentar(const string& comentario, unsigned desde, unsigned hasta);
  void resolver_comentario(id_comm id);

    // Otras operaciones
  unsigned cantidad_comentarios() const;

  private:
  /* Completar... */
};
```

Notar:

- Se pueden agregar las **funciones auxiliares** que crea necesarias en la parte **privada** de la clase `EditorResaltado`.
- El método `comentarios_palabra` devuelve un contenedor *por referencia*. Esto significa que debe devolver una **referencia** a información previamente almacenada y, cuando la función devuelve el contenedor, no se computa el costo de copiarlo.

Requerimientos de complejidad

Dadas las siguientes magnitudes asociadas al editor:

P	cantidad de palabras totales del texto
C	cantidad de comentarios totales
M	cantidad de comentarios de la palabra *más comentada*
Ri	cantidad de palabras del rango del comentario i

Se pide respetar las siguientes cotas de complejidad en el peor caso:

`EditorResaltado()`	*O(1)*
`con_texto(const string& texto)`	*sin requerimiento*
`longitud()`	*O(1)*
`palabra_en(unsigned pos)`	*O(1)*
`texto_comentario(id_comm id)`	*O(log C)*
`comentarios_palabra(unsigned pos)`	*O(1)*
`cargar_texto(...)`	*sin requerimiento*
`insertar_palabra(...)`	*O(P+C)*
`borrar_palabra(...)`	*O(P+C)*
`comentar(...)`	*O(P+C)*
`resolver_comentario(id_comm id)`	*O(log C + Ri.(log M))*
`cantidad_comentarios()`	*O(1)*

Descripción detallada de las operaciones

- `EditorResaltado()`
  - Pre: Verdadero
  - Post: Construye un editor vacío.
- `static EditorResaltado con_texto(const string & texto)`
  - Pre: `texto` es una cadena de palabras separadas por espacios
  - Post: devuelve un editor con las palabras de `texto` insertadas en su orden original
- `unsigned longitud() const`
  - Pre: Verdadero
  - Post: Devuelve la cantidad de palabras del texto.
- `const string& palabra_en(unsigned pos) const`
  - Pre: 0 <= pos < longitud()
  - Post: devuelve la palabra en esa posición.
- `const string& texto_comentario(id_comm id) const`
  - Pre: El comentario `id` existe para alguna posición del texto.
  - Post: Devuelve el texto del comentario `id`
- `const set<id_comm>& comentarios_palabra(unsigned pos) const`
  - Pre: 0 <= pos < longitud()
  - Post: Devuelve el conjunto de comentarios que aplican a la posición `pos`.
- `void cargar_texto(const string& archivo_texto, const string& archivo_comentarios)`
  - Pre: El archivo `archivo_comentarios` tiene comentarios válidos para el texto en el archivo `archivo_texto`. Se espera que cada línea `archivo_comentarios` siga el formato "`i j Texto del comentario[enter]`", donde i y j son las posiciones entre las que se realiza el comentario.
  - Post: Borra el contenido viejo y carga el texto del archivo `archivo_texto` y posteriormente le aplica los comentarios de `archivo_comentarios`.
- `void insertar_palabra(const string& palabra, unsigned pos)`
  - Pre: 0 <= pos <= longitud()
  - Post: inserta `palabra` en la posición `pos`. Si `pos = longitud()` se inserta al final. Si `pos` era previamente afectada por uno o más comentarios, los mismos se expanderán para contener a la palabra insertada también.
- `void borrar_palabra(unsigned pos)`
  - Pre: 0 <= pos < longitud()
  - Post: elimina la palabra ubicada en la posición `pos`. Elimina cualquier comentario cuya única posición afectada fuera `pos`.
- `id_comm comentar(const string& texto, unsigned desde, unsigned hasta)`
  - Pre: 0 <= desde < hasta <= longitud()

  - Post: Devuelve el ID de un nuevo comentario que aplica a *todas* las palabras entre `desde` y `hasta` (sin incluir esta última posición). **Nota**: Los IDs deben asignarse de manera creciente empezando por 1. **No se reutilizan IDs de comentarios eliminados.**
- `void resolver(id_comm id)`
  - Pre: El comentario `id` existe para alguna posición del texto.
  - Post: Se elimina `id` de todas las posiciones afectadas por éste.
- `unsigned cantidad_comentarios() const`
  - Pre: Verdadero
  - Post: Devuelve la cantidad de comentarios actuales en el editor.

Entorno de desarrollo

El código base contiene la configuración necesaria para cargar el proyecto en VSCode dentro de un container. Al seleccionar el directorio mediante *Open Folder* asegurarse de elegir `tp-codigo`, ya que en caso contrario el paso de configuración de CMake podría fallar de forma poco clara.

Una vez elegido el directorio y abierto el container, se debe configurar el proyecto con CMake tras lo cual el editor detectará dos *targets*:

- `editor-resaltado`: el ejecutable del editor, con interfaz gráfica interactiva (experimental)
- `editor-tests`: tests para verificar el comportamiento del editor

Sugerencias:

- Enfocarse inicialmente en tests, ya que la interfaz gráfica no funcionará hasta que la clase `EditorResaltado` esté 100 % implementada.
- Escribir tests adicionales si desean probar otros casos bordes no cubiertos o hacer pruebas.

Entrega

Este trabajo debe resolverse modificando únicamente los archivos `EditorResaltado.h` y `EditorResaltado.cpp`, los cuales deberán entregarse a través del campus virtual antes de que finalize el plazo de entrega.

Versiones de este documento

- 10/6: Versión inicial.