Estructuras de Datos Taller 2

Salomón Avila y Gabriel Jaramillo

February 2025

1 Resumen

El presente código ofrece una solución organizada y clara para procesar archivos de texto y realizar búsquedas específicas dentro de ellos, aplicando buenas prácticas de programación orientada a objetos y el uso de herramientas estándar de C++.

1.1 Planteamiento del Problema

El código en C++ tiene como objetivo analizar un archivo de texto para identificar palabras que cumplen ciertas condiciones. En particular, busca palabras que comienzan con una subcadena dada y palabras que contienen dicha subcadena o su versión invertida.

1.2 Propuesta de Solución

Para resolver el problema, el código implementa dos clases principales:

- **Palabra**: Representa una palabra individual, almacenando el texto y el número de línea donde aparece.
- **ArchivoTexto**: Contiene un vector de vectores de objetos `Palabra`, donde cada sub-vector representa una línea del archivo. Esta clase proporciona métodos para:
  - Agregar líneas de palabras.
  - Obtener el número total de líneas.
  - Buscar palabras que comiencen con una subcadena dada.
  - Buscar palabras que contengan la subcadena o su versión invertida.

El código lee un archivo de texto, extrae las palabras por línea y almacena la información en un objeto de la clase `ArchivoTexto`. Luego, realiza las búsquedas y muestra los resultados.

1.3 Método de Prueba

Para probar la funcionalidad del programa, se ejecuta con un archivo de entrada que contiene:

1. Un número que indica la cantidad de líneas.
2. Una subcadena que se usará como criterio de búsqueda.
3. Varias líneas de texto con palabras separadas por espacios.

El programa procesa este archivo, realiza las búsquedas y muestra los resultados en la consola.

1.4 Resultados Esperados

Se espera que el programa:

- Cuente correctamente las líneas del archivo.
- Identifique y muestre las palabras que comienzan con la subcadena.
- Identifique y muestre las palabras que contienen la subcadena y su versión invertida.
- Proporcione información clara y estructurada sobre las coincidencias encontradas.

2 TADs

TAD Palabra:
Datos mínimos:

- palabra, cadena de caracteres, almacena una palabra del archivo.
- numLinea, entero, guarda la línea en la cual se ubica la palabra.

Operaciones:

- FijarPalabra(nuevaPalabra), cambia el valor actual de palabra a nuevaPalabra.
- FijarNumLinea(nuevoNumLinea), cambia el valor actual de numLinea a nuevoNumLinea.
- ObtenerPalabra(), retorna el valor actual de la palabra.
- ObtenerNumLinea(), retorna el valor actual del número de la línea.

TAD ArchivoTexto:
Datos mínimos:

- líneasTexto, vector de vectores de Palabra, almacena palabras.

Operaciones:

- FijarListaLineas(nuevaLista), cambia el vector de vectores actual de lineasTexto a nuevaLista.
- ObtenerListaLineas(), retorna el vector de vectores "lineasTexto".
- ObtenerNumLineas(), retorna la cantidad de líneas en el texto.
- AgregarListaPals(nuevaLista), agrega un vector de Palabras al vector de vectores "lineasTexto".
- BuscarPrincipio(subcadena), busca las palabras que inician con la subcadena recibida.
- BuscarContiene(subcadena), busca las palabras que contienen la subcadena recibida.

3 Diseño

Con el objetivo de construir las funciones necesarias para la correcta implementación del código, se plantearon las siguientes instrucciones:

Función para `buscar el principio de una palabra`:

1. Se establece la cadena que se va a buscar.
2. Se iteran las líneas de texto del archivo.
3. Por cada línea, se extraen las palabras contenidas en ella.
4. Se verifica si la primera letra de cada palabra coincide con la primera letra de la cadena buscada.
5. En caso afirmativo, la palabra se almacena en un vector.
6. Se imprimen los resultados de la búsqueda.

Función para `buscar las cadenas que contienen la subcadena`:

1. Se define la subcadena y su versión invertida.
2. Se declaran vectores para almacenar las palabras coincidentes.
3. Se iteran las líneas de texto del archivo.
4. Se extraen las palabras de cada línea.
5. Se verifica si la subcadena está contenida dentro de cada palabra analizada.
6. Si la palabra contiene la subcadena, se agrega al vector correspondiente.
7. Se imprimen los resultados de la búsqueda.

4 Compilación

Tanto para Windows como para Linux, el comando de compilación es el mismo:

```
g++ -o repeticiones repeticiones.cpp
```

Sin embargo, la variación se encuentra al momento de ejecutar el programa. Para Linux es el siguiente:

```
./repeticiones [Nombre_Del_Archivo.txt]
```

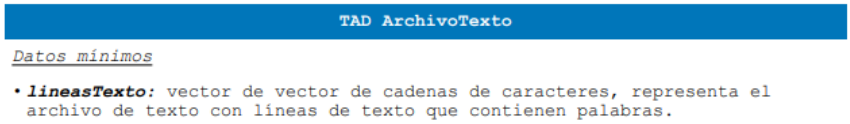
Y para Windows:

```
repeticiones.exe [Nombre_Del_Archivo.txt]
```

5 Análisis Códigos

En materia de los TADS, se siguieron en su mayoría los diseños sugeridos para el desarrollo del taller; sin embargo, se hizo una modificación a la clase `ArchivoTexto`.

Figure 1: Sugerencia TAD



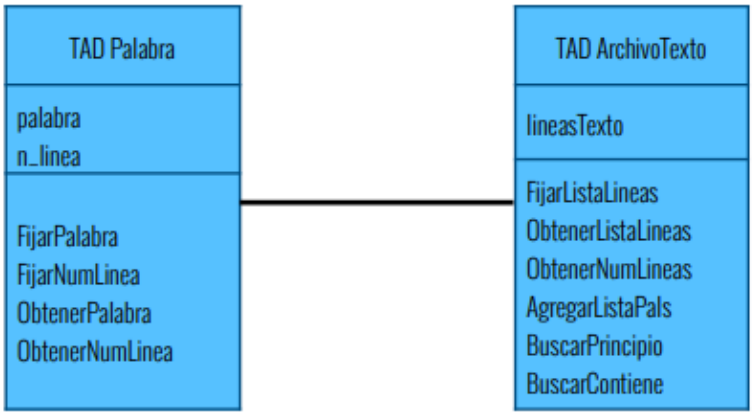
La implementación que proponemos es la expuesta en la Figura 2, donde se cumpliría con lo expuesto en el diagrama de clases de la Figura 3.

Figure 2: Propuesta

```
/**
 * Creacion de la clase palabra con sus metodos y atributos
 */
class ArchivoTexto{
    private:

        vector<vector<Palabra>> lineasTexto;
```

Figure 3: Diagrama de clases



A continuación se presentará la lógica detrás del desarrollo.

5.1 Palabras que inician con la subcadena

Figure 4: Algoritmo búsqueda al inicio

```
void buscarPrincipio(Palabra subcadena){
    string subcadena_extraida = subcadena.obtenerPalabra();
    vector<Palabra> comienzan;
    for(auto l : lineasTexto){
        for(auto p : l){
            string palabra_extraida = p.obtenerPalabra();
            if(palabra_extraida.size()<subcadena_extraida.size()) continue;
            bool flag = true;
            for(int i = 0; i<subcadena_extraida.size(); i++){
                if(palabra_extraida[i] != subcadena_extraida[i]) flag = false;
            }
            if(flag){
                comienzan.push_back(p);
            }
        }
    }
    cout<<"\nHay "<<comienzan.size()<<" palabras que comienzan con: "<<subcadena_extraida<<"
    for(auto p : comienzan){
        cout<<"Linea "<<p.obtenerNumLinea()<<": "<<p.obtenerPalabra()<<"\n";
    }
}
```

En el código presentado en la Figura 4, por cada palabra dentro del vector de vectores de palabras se evalúa si su tamaño es mayor o igual al de la subcadena. Si es así, se itera carácter por carácter de la palabra y la subcadena para verificar si la palabra comienza efectivamente con la subcadena. Si la verificación es exitosa (el booleano de control es verdadero), se añade la palabra a un vector temporal para su posterior impresión, indicando cuántas palabras cumplen la

condición y mostrando, para cada una, su contenido y la línea en la que se encuentran.

5.2 Búsqueda en cualquier posición

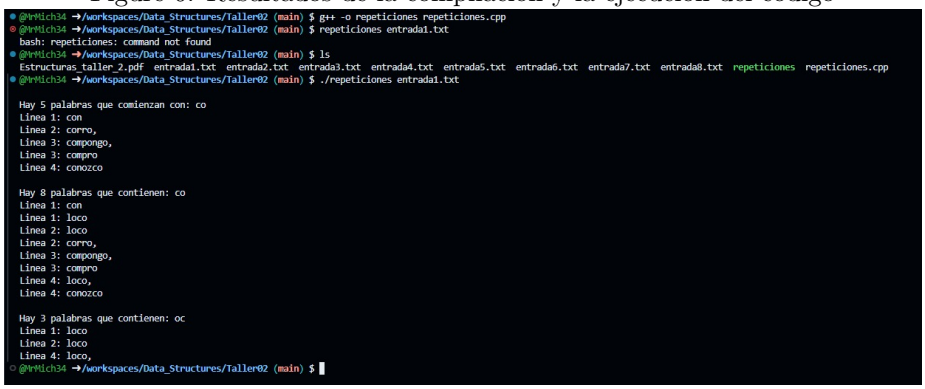
Figure 5: Algoritmo búsqueda en cualquier posición

```
void buscarContienen(Palabra subcadena){
    string subcadena_extraida = subcadena.obtenerPalabra();
    string subcadena_reversa = subcadena_extraida;
    reverse(subcadena_reversa.begin(),subcadena_reversa.end());
    vector<Palabra> contienen,contienen_invertida;
    for(auto l : lineasTexto){
        for(auto p : l){
            string palabra_extraida = p.obtenerPalabra();
            if(palabra_extraida.find(subcadena_extraida) != string::npos){
                contienen.push_back(p);
            }
            if(palabra_extraida.find(subcadena_reversa) != string::npos){
                contienen_invertida.push_back(p);
            }
        }
    }
    cout<<"\nHay "<<contienen.size()<<" palabras que contienen: "<<subcadena_extraida<<"\n";
    for(auto p : contienen){
        cout<<"Linea "<<p.obtenerNumLinea()<<": "<<p.obtenerPalabra()<<"\n";
    }
    cout<<"\nHay "<<contienen_invertida.size()<<" palabras que contienen: "<<subcadena_reversa<<"\n";
    for(auto p : contienen_invertida){
        cout<<"Linea "<<p.obtenerNumLinea()<<": "<<p.obtenerPalabra()<<"\n";
    }
}
```

Para facilitar el manejo, se convierte la palabra a un `string` y se crea la versión reversa de la subcadena utilizando la función `reverse()` de la STL. La lógica de búsqueda es similar a la anterior: se recorre palabra por palabra el contenido del archivo, y se utiliza el método `find` propio de los `string`. Este método busca la subcadena dentro de la palabra, retornando la posición de la primera ocurrencia o `npos` si no la encuentra. Cada ocurrencia se almacena en vectores temporales y, al final, se imprime la cantidad de coincidencias junto con la palabra y la línea donde se encuentra.

6 Resultados

Figure 6: Resultados de la compilación y la ejecución del código



7 Consideraciones adicionales

El resto del código presente en el archivo `repeticiones.cpp` contiene la lógica simple para la lectura de archivos y el desglose de un `string` completo en palabras individuales, utilizando los TADS propuestos.

8 Conclusiones

El análisis del código en C++ permite extraer varias conclusiones interesantes sobre su diseño y funcionalidad:

- **Modularidad y abstracción:** El uso de clases como `Palabra` y `ArchivoTexto` demuestra una clara separación de responsabilidades. Cada clase se encarga de manejar sus propios datos y operaciones, lo que favorece la reutilización y el mantenimiento del código.
- **Utilización de TADS y STL:** Se hace uso extensivo de contenedores como `vector` y de funciones de la STL, lo que mejora la eficiencia y simplifica el manejo de colecciones de datos, como la lectura y procesamiento de líneas de un archivo.
- **Búsqueda y manipulación de cadenas:** El código implementa algoritmos para buscar palabras que comienzan con una subcadena y para encontrar palabras que contienen una subcadena (o su versión invertida). Esto evidencia la importancia de entender y manejar correctamente las funciones de manipulación de cadenas, como `find` y `reverse`, para resolver problemas comunes de procesamiento de texto.

- **Comentarios y documentación:** La inclusión de comentarios detallados y la correcta documentación del código facilitan la comprensión del mismo, tanto para el autor como para otros desarrolladores que puedan trabajar en el proyecto en el futuro.
- **Lectura y desglose de archivos:** La lógica implementada para la lectura de archivos y el posterior desglose de cada línea en palabras individuales demuestra cómo se pueden aplicar técnicas básicas de entrada/salida en C++ para procesar datos de forma eficiente.