Estructuras de Datos Taller 2

Salomón Avila y Gabriel Jaramillo

February 2025

1 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 compilarlo. Para Linux es el siguiente:

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

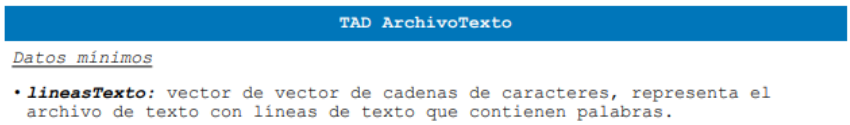
Y para Windows:

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

2 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



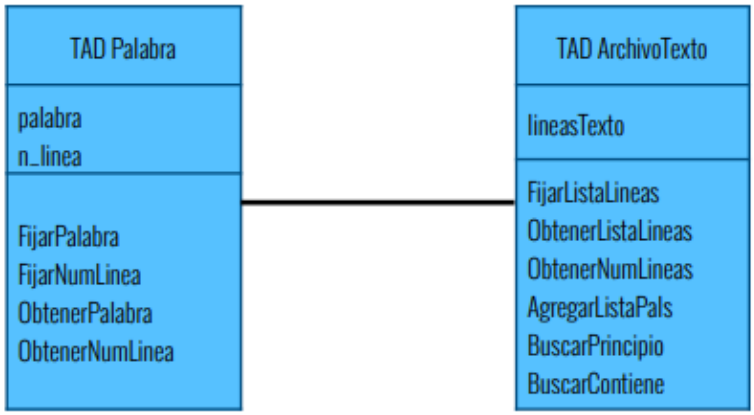
Y la implementación que proponemos es la expuesta en la Figura 2, donde se cumpliría con lo expuesto en el diagrama de clases de la Figura 3.

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

2.1 Palabras que inician con la subcadena

Figure 4: Algoritmo busqueda 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 palabra vamos a tener una consideración en la cual la palabra no debe ser de un menor tamaño que la subcadena. Si es de igual o mayor tamaño, entonces se itera por cada carácter tanto de la cadena como de la subcadena buscando un error, activando de esta manera un booleano de control presente en la función.

En caso de que la variable booleana sea verdadera, significa que efectivamente inicia por la subcadena, añadiendo la palabra a un vector temporal donde se almacenaran las respectivas palabras. Al final de esta función, se imprime cuantas palabras comienzan por la subcadena especificada, y por cada una de ellas se muestra su contenido y en que línea del texto se encuentran.

2.2 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 un mejor manejo, convertimos esta palabra en un string, y se creo la subcadena reversa usando la función reverse() incluida en la STL, donde al ingresar los iteradores del inicio y del final se encarga de reversar el texto.

Se usa la misma lógica del algoritmo pasado para acceder palabra por palabra al contenido del archivo, y se utiliza el método find propio de los strings. Este método consiste en buscar el substring dentro de un string, y si existe retorna la posición de la primera letra de dicha ocurrencia, en caso contrario retorna un npos, propio también del string.

Cada ocurrencia se va insertando a vectores temporales para su almacenamiento, Finalmente se imprime la cantidad de ocurrencias, y por cada una de estas la palabra y la línea en donde se encuentra alojada en el archivo.

3 Consideraciones adicionales

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