Eugenio Pedraza - A00834064

Reflexión

Durante la actividad logramos utilizar las diversas herramientas (algoritmos) aprendidos durante la clase para poder resolver la problemática de encontrar substrings, palindromos, y substring mas largo. Uno de mis mayores aportaciones del trabajo fue en poder leer los archivos de texto y convertirlos en string para poder utilizar los algoritmos en ellos, y la busqueda del substring utilizando el algoritmo KMP dado que fue el que proporciono una complejidad optima para busqueda en archivos de tamaño grande. Durante la actividad intentamos implementar diferentes soluciones a la problemática, un ejemplo de ellos fue intentar probar con un arbol trie, pero al hacer pruebas nos dimos cuenta de lo ineficiente que era y por eso aprendi que es necesario poder tener diferentes soluciones a la misma problemática para asi descartar los menos apropiados.

Algoritmo KMP - Complejidad O(m) en creación de la tabla y búsqueda, lo que al correr la búsqueda y creación tiene una complejidad de O(m+n).

El algoritmo se divide en dos partes, esta primera parte se encarga de recibir el archivo mcodeX para computar la tabla KMP como vector que será utilizado para poder encontrar el substring, ya que permite en la búsqueda saltarse las partes que no se parezcan.

Para ejemplificar si la palabra para crear la tabla es “adidas” se hará de esta manera:

1. *El primer índice de la tabla KMP siempre equivale a 0.*
2. *Compara el índice 0 (a) con el 1 (d), como no son iguales y len == 0 ingresa en el índice 2 un 0.*
3. *Sube un dígito el valor i ahora se compara el índice 0 (a) con 2 (i), como no son iguales y len == 0 ingresa en el índice 3 un 0.*
4. *Sube otro dígito el valor i y ahora se compara el índice 0 (a) con 3 (d), como no son iguales y len == 0 ingresa en el índice 4 un 0.*
5. *Sube otro dígito el valor i y ahora se compara el índice 0 (a) con 4 (a), como si son iguales sube un dígito a len y i, y se agrega 1 (el valor de len).*
6. *Ahora se hace la comparación del índice 1 (d) con 5 (s) como no son iguales y len > 0 se cambia el valor de len con la siguiente fórmula len = kmpTable[len-1] convirtiendo el último valor en 0.*
7. *La tabla kmp se vería algo así [0,0,0,0,1,0].*

```
std::vector<int> Algorithm::computeKMPTable(const std::string& word) {
    int m = word.length();
    std::vector<int> kmpTable(m, 0);
```

```
    int len = 0;
    int i = 1;

    while (i < m) {
        if (word[i] == word[len]) {
            len++;
            kmpTable[i] = len;
            i++;
        } else {
            if (len != 0) {
                len = kmpTable[len - 1];
            } else {
                  kmpTable[i] = 0;
                   i++;
            }
        }
    }

    return kmpTable;
}
```

Una vez que la tabla está creada se continua en esta función de búsqueda en el cual recibe de parametros la palabra de búsqueda y el texto completo. Recorriendo todo el string del archivo para ver si existen coincidencias.

Vamos a ejemplificar:
Texto: “zzz adiz adidas zzz adias”
Patrón: “adidas”

1. *Se computa la tabla KMP dándonos de resultado un vector con los valores [0, 0, 0, 0, 1, 0]*
2. *Se inicializa con i = 0 y j = 0*
3. *El algoritmo va corriendo el texto hasta que encuentra coincidencias con el patrón.*
4. *Encuentra coincidencias de “a” con “a” (i = 4, j =0) luego “d” con “d” (i=5, j =1), y así va hasta que se complete la coincidencia de “adidas”.*
5. *Si se encuentra el patron y j == pattern.size() se devuelve un tuple con true, i los índices de inicio y final del substring.*
6. *En el dado caso que no exista un patrón se regresa false con dos enteros con un valor de -1.*

```
std::tuple<bool,int, int> Algorithm::KMP_Search(const std::string& text, const std::string&
pattern) {
    std::vector<int> kmpTable = computeKMPTable(pattern);
    int i = 0;
    int j = 0;

    while (i < text.size()) {
```

```
        if (pattern[j] == text[i]) {
            j++;
            i++;
        }

        if (j == pattern.size()) {

            return std::make_tuple(true, i-j, i-1);

            j = kmpTable[j - 1];
        } else if (i < text.size() && pattern[j] != text[i]) {
            if (j != 0) {
                j = kmpTable[j - 1];
            } else {
                i = i + 1;
            }
        }
    }

    return std::make_tuple(false, -1, -1);
}
```

Esta es la función encargada de abrir los archivos y leerlos para poder meterlos dentro de una variable como strings.

```
std::string Algorithm::readFileIntoString(const std::string& path) {
    std::ifstream inputFile(path);
    if (!inputFile.is_open()) {
        std::cerr << "Error opening file " << path << std::endl;
        return "";
    }
    std::stringstream buffer;
    buffer << inputFile.rdbuf();
    inputFile.close();
    return buffer.str();
}
```

El main encargado de llamar las funciones y crear las variables necesarias para poder hacer la resolución de la problemática.

```
int main() {
    Algorithm buscador;
```

```
    std::string transmission1 = buscador.readFileIntoString("transmission1V2.txt");
    std::string transmission2 = buscador.readFileIntoString("transmission2V2.txt");
    std::string mcode1 = buscador.readFileIntoString("mcode1V2.txt");

    std::cout << "\nParte 1: Busqueda de Mcode dentro de los ashivos de transmicion" <<
std::endl;
    std::tuple<bool,int, int> tr1_mc1 = buscador.KMP_Search(transmission1, mcode1);
    std::tuple<bool, int, int> tr2_mc1 = buscador.KMP_Search(transmission2, mcode1);

    if (std::get<0>(tr1_mc1)) {
        std::cout << "Mcode1 se encuentra dentro de Transmision1 en la posicion: "
                << std::get<1>(tr1_mc1) << " a " << std::get<2>(tr1_mc1) << std::endl;
    } else {
        std::cout << "Mcode1 no se encuentra dentro de Transmision1" << std::endl;
    }

    if (std::get<0>(tr2_mc1)) {
        std::cout << "Mcode1 se encuentra dentro de Transmision2 en la posicion: "
                << std::get<1>(tr2_mc1) << " a " << std::get<2>(tr2_mc1) << std::endl;
    } else {
        std::cout << "Mcode1 no se encuentra dentro de Transmision2" << std::endl;
    }

    std::cout << "\nParte 2: Busqueda de codigo espejeado" << std::endl;
    std::pair<int, int> result = buscador.manacher(transmission1);

    if (result.first == -1 && result.second == -1) {
        std::cout << "No se ha encontrado codigo espejeado" << std::endl;
    } else {
        printf("Se encontro codigo espejeado desde el indice %d hasta %d", result.first,
result.second);
        std::cout << "\n";
    }

    std::pair<int, int> result2 = buscador.manacher(transmission2);

    if (result2.first == -1 && result2.second == -1) {
        std::cout << "No se ha encontrado codigo espejeado" << std::endl;
    } else {
        printf("Se encontro codigo espejeado desde el indice %d hasta %d", result2.first,
result2.second);
        std::cout << "\n";
    }

    std::cout << "\n";

    std::cout << "\nParte 3: Coincidencia en transmiciones" << std::endl;
```

```
    buscador.lcs(transmission1, transmission2);

    return 0;
}
```