INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY

SEMESTRE AGO-DIC 2021

ANÁLISIS Y DISEÑO DE ALGORITMOS AVANZADOS

RELFEXIÓN DE LA ACTIVIDAD INTEGRADORA 1
12/SEPTIEMBRE/2021

PABLO YAMAMOTO MAGAÑA - A01022382

Para dar solución a la actividad integradora 1, decidimos crear un fichero llamado data en donde tuvieramos una clase Transmission, cada instancia de esta clase tenía una propiedad (entre otras propiedades que se explicarán más adelante) llamada transmission que era un vector de strings en donde almacenamos cada línea de los archivos transmission1.txt y transmission2.txt.

```
class Transmission
{
public:
    vector<string> transmission;

    Transmission(string fileName)
    {
        ifstream MyFile;
        MyFile.open(fileName);
        string line;

        while (!MyFile.eof())
        {
            getline(MyFile, line, '\n');
            transmission.push_back(line);
        }
    }
```

Al construir cada objeto, se recibía el nombre de los archivos, se leía cada linea y se guardaba en el vector ya mencionado. Esto resulta conveniente para poder acceder a cada línea como veremos más adelante.

```
int main()
{
    Transmission transmission1("transmission1.txt");
    Transmission transmission2("transmission2.txt");

    //vector<string> codeFiles ();
    vector<string> codes(readCodes({"mcode1.txt", "mcode2.txt", "mcode3.txt"}));
```

En la función main en nuestro archivo ActIn1.cpp creamos nuestros objetos mandando los archivos de transmisión respectivamente y también tenemos una función llamada readCodes que devuelve un vector de strings conteniendo cada linea de código malicioso.

```
vector<string> readCodes(vector<string> codeFiles)
{
    vector<string> codes;
    string line;

    for (string code : codeFiles)
    {
        ifstream MyFile;
        MyFile.open(code);
        getline(MyFile, line, '\n');
        codes.push_back(line);
    }

    return codes;
}
```

En nuestro archivo data.h tenemos otras funciones que sirven para llamar los algoritmos necesarios para la solución del proyecto. Creamos ficheros con los algos de Boyer-Moore, Longest Common Substring y Longest Palindrome.

Dentro de nuestro archivo ActIn1.cpp tenemos 3 funciones para imprimir cada parte del proyecto. La parte 1 verifica si existe codigo malicioso en cualquier archivo de transmisión, es decir comparamos el archivo transmission1.txt y transmission2.txt con mcode1, 2 y 3.txt.
La primera parte utiliza el algoritmo de Boyer-Moore (en el peor de los casos tiene una complejidad de O(mn) y O(n/m) en el mejor) que implementa las heurísticas del carácter malo y el sufijo bueno. La implementación de ambas permite escoger el salto de palabra más grande y ahorrar tiempo. Esta parte del proyecto es importante porque nos da una idea de como podemos analizar archivos en busca de ciertas cosas y definitivamente podemos ver su aplicación dentro del área de ciberseguridad.

Nuestra función utiliza un vector para cada archivo de transmisión, si encontró código malicioso, si el primer valor de nuestro vector tiene un 1, es porque si hay una coincidencia, después imprimos el true, la línea en donde se encuentra seguido de la posición.

```
void printPartOne(Transmission T, vector<string> codes)
{
    for (int i = 0; i < codes.size(); i++)
    {
        vector<int> partOne(T.searchCode(codes[i]));
        if (partOne[0])
        {
            cout << "true"
                 << " " << partOne[1] << " " << partOne[2] << endl;
        }
        else
        {
            cout << "false" << endl;
        }
    }
}
```

Para la parte 2 teníamos que encontrar si había código espejado dentro de los archivos de transmisión. Implementamos el algoritmo de Manacher (tiene una complejidad de O(n)). Las condiciones del proyecto nos aseguran que hay un palindromo en cada archivo.

```
void printPartTwo(Transmission T)
{
    vector<int> partTwo(T.findLongestPalindrome());
    for (int i = 0; i < partTwo.size(); i++)
    {
        cout << partTwo[i] << " ";
    }
    cout << endl;
}
```

Para la parte 3, encontramos el substring más largo y también imprimos la línea donde se encuentra, la longitud y la posición. Para la posición hicimos una pequeña modificacón porque regresamos el indice donde empieza el substring en lugar en donde termina. El algo implementado tiene una complejidad de O(mn) pero como tenemos que comparar cada línea del archivo transmission1.txt con cada línea del archivo transmission2.txt entonces realmente la complejidad es mucho mayor.

```
vector<int> findLongestCommonSubstring(Transmission t2)
{
    LCS longesCSubstring;
    vector<int> ans(3, 0);

    for (int i = 0; i < transmission.size(); i++)
    {
        for (int j = 0; j < t2.transmission.size(); j++)
        {
            vector<int> a(longesCSubstring.longestCommonSubstring(transmission[i],
            t2.transmission[j]));
            if (a.at(0) > ans.at(2))
            {
                ans.at(0) = i;
                ans.at(1) = a.at(1);
                ans.at(2) = a.at(0);
            }
        }
    }

    return ans;
}
```