Sebastián Rosas Maciel - A01233989

Reflexión: Actividad Integradora No. 1

Algoritmo de Manacher - Complejidad: O(n)

Mi mayor contribución dentro de esta actividad fue la segunda parte del reto, la cual consiste en buscar palindromos dentro de el archivo. Al final termine optando por utilizar el algoritmo de Manacher ya que este busca encontrar el palindromo mas largo posible.

```
std::pair<int, int> Algorithm::manacher(const std::string &s) {
    int n = s.size();
    if (n == 0) return {0, 0};

    // Crear la cadena extendida
    std::string extended = "#";
    for (char c : s) {
        extended += c;
        extended += '#';
    }

    n = extended.size();
    std::vector<int> LPS(n, 0);
    int center = 0, right = 0;
    int maxLen = 0, start = 0;

    for (int i = 0; i < n; i++) {
        int mirror = 2 * center - i;
        if (i < right) {
            LPS[i] = std::min(right - i, LPS[mirror]);
        }

        int a = i + (1 + LPS[i]);
        int b = i - (1 + LPS[i]);
        while (a < n && b >= 0 && extended[a] == extended[b]) {
            LPS[i]++;
            a++;
            b--;
        }

        if (i + LPS[i] > right) {
            center = i;
            right = i + LPS[i];
        }

        if (LPS[i] > maxLen) {
            maxLen = LPS[i];
            start = (i - LPS[i]) / 2; // Ajusta el índice de inicio para
la cadena original
        }
```

```
    }

    // Ajusta el índice de fin para la cadena original
    int end = start + maxLen - 1;

    return {start, end};
}
```

Funcionamiento

El algoritmo toma a un solo string como entrada, y regresa a un par de **dos numeros enteros** como salida, donde el primer numero representa al **indice donde empieza** el palindromo, y el segundo numero marca al indice donde **finaliza**.

Primero, se verifica si la cadena de entrada está vacía. Si es así, se devuelve un par de ceros.

```
int n = s.size();
if (n == 0) return {0, 0};
```

Luego, se crea una cadena extendida donde se inserta un carácter especial ('#') entre cada par de caracteres en la cadena original. Esto se hace para manejar palíndromos de longitud par e impar de manera uniforme.

```
std::string extended = "#";
for (char c : s) {
    extended += c;
    extended += '#';
}
```

Se inicializa un vector LPS para almacenar la longitud del palíndromo más largo hasta el índice i. También se inicializan las variables center y right para rastrear el centro y el extremo derecho del palíndromo más largo encontrado hasta ahora.

```
n = extended.size();
std::vector<int> LPS(n, 0);
int center = 0, right = 0;
int maxLen = 0, start = 0;
```

El bucle principal itera a través de cada carácter en la cadena extendida. Para cada carácter, calcula su "espejo" con respecto al centro del palíndromo actual. Si el carácter está dentro del palíndromo actual, su longitud de palíndromo se inicializa con el mínimo de la longitud del espejo y la distancia al extremo derecho del palíndromo.

```
for (int i = 0; i < n; i++) {
    int mirror = 2 * center - i;
    if (i < right) {
        LPS[i] = std::min(right - i, LPS[mirror]);
    }
```

Luego, intenta expandir el palíndromo en torno al carácter actual y actualiza la longitud del palíndromo en LPS[i] en consecuencia.

```
int a = i + (1 + LPS[i]);
int b = i - (1 + LPS[i]);
while (a < n && b >= 0 && extended[a] == extended[b]) {
    LPS[i]++;
    a++;
    b--;
}
```

Si el palíndromo expandido se extiende más allá del extremo derecho del palíndromo actual, se actualizan las variables center y right.

```
if (i + LPS[i] > right) {
    center = i;
    right = i + LPS[i];
}
```

Si la longitud del palíndromo en el índice actual es mayor que maxLen, se actualizan maxLen y start.

```
if (LPS[i] > maxLen) {
    maxLen = LPS[i];
    start = (i - LPS[i]) / 2; // Ajusta el índice de inicio para la cadena
original
}
```

Finalmente, se calcula el índice de fin del palíndromo más largo en la cadena original y se devuelve el par de índices de inicio y fin.

```
int end = start + maxLen - 1;
return {start, end};
```

Relfexión

Durante la resolucion de este reto, aprendi que homologar procesos no es necearíamente la accion correcta a tomar al momento de resolver varios problemas, ya que encomendar multiples acciones a una sola estructura no siempre es la tarea mas sencilla, mucho menos la mas eficiente.

Es por esto que idealmente, deberiamos de buscar cual es la solucion mas eficiente a un problema, y de ser posible, intentar complermentarla con la estructura que tengamos como base (si aplica).