**INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY**

**CAMPUS PUEBLA**

Análisis y diseño de algoritmos avanzados (Gpo 602)

TC2038.602

E1. Actividad Integradora 1

**Profesor:**

Juan Manuel Ahuactzin Larios

Roberto Castro Gómez | A01425602

Santiago Rodríguez Gutiérrez | A01738097

Luis Antonio Salinas Gonzalez | A01735375

**20 de Octubre de 2025**

Para esta actividad se solicitó estudiar los algoritmos de SA-IS y Manber & Meyers para posteriormente traducir los códigos a C++ o Golang. Además, se solicitó agregar funcionalidades extra a los códigos como recibir como entrada el nombre de los archivos sobre los cuales se desea trabajar y reducir la carga en memoria.

El algoritmo de Manber & Meyers contaba con pequeños errores al inicio, por lo que se le preguntó a una IA generativa (Chat GPT) cual era ese error y este fue el resultado:

**PROMPT**

```
class SubstrRank:
    def __init__(self, left_rank=0, right_rank=0, index=0):
        self.left_rank = left_rank
        self.right_rank = right_rank
        self.index = index

def make_ranks(substr_rank, n):
    r = 1
    rank = [-1] * n
    rank[substr_rank[0].index] = r
    for i in range(n):
        if (substr_rank[i].left_rank != substr_rank[i-1].left_rank or
                    substr_rank[i].right_rank != substr_rank[i-1].right_rank):
            r += 1
        rank[substr_rank[i].index] = r
    return rank

def suffix_array(T):
    n = len(T)
    substr_rank = []

    for i in range(n):
        substr_rank.append(SubstrRank(ord(T[i]), ord(T[i + 1]) if i < n-1 else 0,
i))

    substr_rank.sort(key=lambda sr: (sr.left_rank, sr.right_rank))

    l = 2
    while l < n:
        rank = make_ranks(substr_rank, n)

        for i in range(n):
            substr_rank[i].left_rank = rank[i]
            substr_rank[i].right_rank = rank[i+l] if i+l < n else 0
```

```
            substr_rank[i].index = i
        l *= 2

        substr_rank.sort(key=lambda sr: (sr.left_rank, sr.right_rank))

    SA = [substr_rank[i].index for i in range(n)]

    return SA

SA = suffix_array("mississippi")
print(S

Explicame este código como a un niño y dime si hay algo mal
```

## Resultado

### ⚠️ Problemas en el código

Sí, **hay errores** (uno sutil y otro lógico):

---

#### ❌ Error #1: en `make_ranks`, el bucle empieza en `i = 0`

python Copy code

```python
for i in range(n):
    if (substr_rank[i].left_rank != substr_rank[i-1].left_rank or
        substr_rank[i].right_rank != substr_rank[i-1].right_rank):
```

👉 En la **primera vuelta** (`i = 0`), `substr_rank[i-1]` es `substr_rank[-1]`, o sea, **el último elemento**.
Eso es incorrecto y puede causar un resultado raro o un error lógico.

✅ **Solución:**
Empieza desde `i = 1`:

python Copy code

```python
for i in range(1, n):
```

#### ❌ Error #2: al actualizar `substr_rank` dentro del `while`

python Copy code

```python
substr_rank[i].left_rank = rank[i]
substr_rank[i].right_rank = rank[i+l] if i+l < n else 0
substr_rank[i].index = i
```

👉 Aquí `index = i` **rompe la lógica**.
`index` debería seguir siendo **el índice del sufijo original**, no la posición en el arreglo.

✅ **Solución:**
Debe usar el índice que tenía antes (el del sufijo real):

python Copy code

```python
substr_rank[i] = SubstrRank(rank[substr_rank[i].index],
                            rank[substr_rank[i].index + l] if substr_rank[i].index + l < n else 0,
                            substr_rank[i].index)
```

### Parámetros por valor y por referencia

Los parámetros por valor son aquellos que se pasan a una función para su uso y modificación con la finalidad de dicha función. Al ser pasados por valor, se envía directamente toda la información contenida y se genera un duplicado de dichos parámetros conforme se va ejecutando la función. Por otra parte, en los parámetros por referencia lo que se genera es una llamada a los parámetros creados previamente a la llamada de la función, permitiendo así las modificaciones y lecturas sobre los mismos en lugar de generar un nuevo espacio en la memoria.

En ambos lenguajes (Python y C++) se utilizan los parámetros por valor cuando se quiere mantener la estructura y el contenido de los mismos al momento de ejecutar una función. Por otra parte, los parámetros por referencia se utilizan para optimizar el uso de memoria cuando no se desea mantener la estructura previa a la función.

Dentro de cada función, se modificaron los parámetros para ser valores referenciados y poder utilizar un poco menos de memoria.

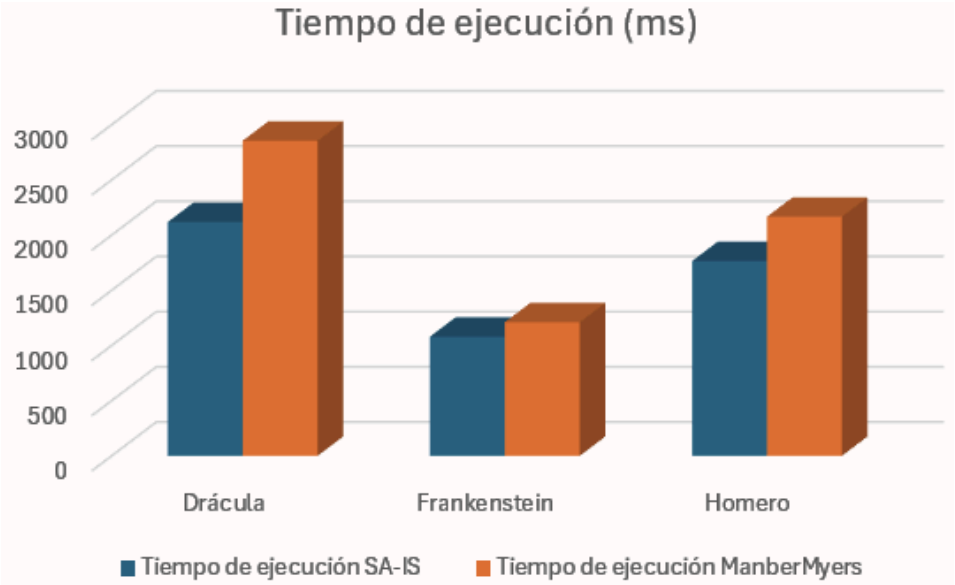
```
C/C++
map<int, pair<int,int>> getBuckets(vector<int>& T){}
vector<int> sais(vector<int>& T){}
string leerArchivo(string& nombre){}
```

De la misma manera, una de las optimizaciones realizadas fue el intercambio del diccionario para los caracteres “L” y “S” a un vector booleano, dentro de este enfoque, el valor positivo representa el carácter “S”, mientras que el carácter “L” es representado por el valor negativo.

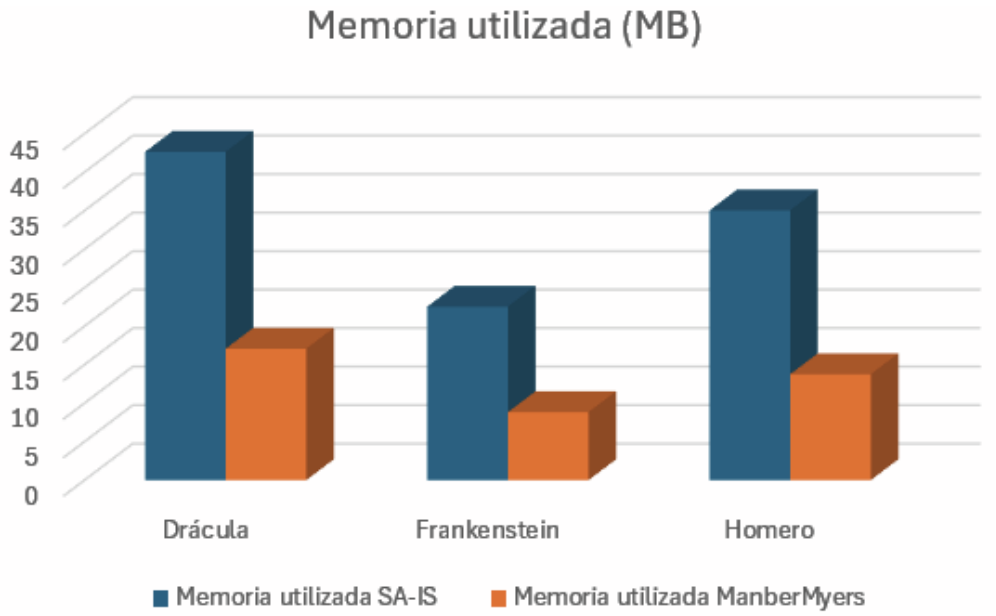
```
C/C++
vector<bool> t(n);//Vector que identifica los tipos de posición: 'S' o 'L'.
//Último carácter de tipo S.
t[n-1] = true;  // true = 'S', false = 'L'

// Clasificación de cada posición
for (int i = n - 1; i > 0; i--) {
	if (T[i-1] == T[i])
		t[i-1] = t[i];
	else
		t[i-1] = (T[i-1] < T[i]);  // true si S, false si L
```

### Gráficas comparativas entre algoritmos

| Tiempo de ejecución (ms) | | |
|---|---|---|
| Libro | SA-IS | Manber Myers |
| Drácula | 2119 | 2855 |
| Frankenstein | 1080 | 1211 |
| Homero | 1765 | 2167 |




| Memoria utilizada (MB) | | |
|---|---|---|
| Libro | SA-IS | Manber Myers |
| Drácula | 42.63 | 17.07 |
| Frankenstein | 22.53 | 8.83 |
| Homero | 35.03 | 13.75 |




**Complejidad temporal**

- **Manber & Myers:**

  La complejidad temporal de este algoritmo es de O (n log n), esto debido a que en cada iteración se duplica el tamaño de las subcadenas que se están considerando y se realiza un ordenamiento basado en estos rangos calculados.

- **SA-IS:**

  El algoritmo tiene una complejidad temporal de O(n) donde n representa el número de líneas de texto que contiene el archivo ya que siempre se analizan las mismas líneas en base a la cantidad de caracteres disponibles.

**Complejidad espacial**

- **Manber & Myers:**

  Este algoritmo usa algunos arreglos auxiliares, su complejidad aproximada es de O (n), sin embargo, la constante de este algoritmo es menor que la de SA-IS. eeeeeee

- **SA-IS:**

  Debido a que las funciones utilizan valores referenciados en lugar de generar nuevos arreglos en cada llamada, la complejidad temporal termina siendo de O(n).

**Reflexiones individuales**

**Santiago:** Esta actividad me pareció demasiado interesante. Primero, me permitió hacer un repaso acerca del lenguaje C++ y las distintas librerías que pueden ser utilizadas, así como sus tipos de datos únicos del lenguaje. También, me ayudó a descubrir la estructura de datos bit vector, que es, como se programa y cual es su uso; esta estructura de datos es muy importante si es primordial reducir el uso de memoria del programa. Por último, realice una investigación y amplíe mis conocimientos con respecto a los algoritmos SA-IS y Manber Myers los cuales son de gran ayuda en reconocedores léxicos.

**Roberto:** La implementación de esta actividad me permitió reforzar los conocimientos adquiridos acerca de los algoritmos de ordenamiento y búsqueda, así como de la construcción de arreglos para almacenar eficientemente el contenido. También reforcé la parte del consumo de tiempo y memoria y cómo analizarlos dentro de las ejecuciones, así como maneras de optimizarlos y reducir la carga generada por los algoritmos.

**Luis Antonio:** Esta actividad me permitió entender cómo es que distintos algoritmos de construcción de arreglos de sufijos pueden variar en eficiencia. Al traducir ambos algoritmos, pude entender que tan importantes son los detalles al implementarlos, como el manejo de la memoria o el paso de variables por referencia y cómo esto afecta a la eficiencia real de los distintos algoritmos.

**Anexos:**

**Repositorio de Github**