E1. Actividad Integradora 1

David Martínez Molina A01735425

Manual de Instalación

1. Descomprima el archivo .zip
2. En su IDE instale las siguientes librerías para python desde terminal:

- **pip install Unicode**
- **pip install memory-profiler**

3. Para probar el libro de su interés en c++ o python:

- Cambiar el contenido de la variable **inputBook**

En Python:

```
def main():

    #Inicio del tiempo de ejecución del algoritmo
    startTime = time.time()

    #Se crea la ruta donde se lee el libro
    inputBook = os.path.join("Libros", "four.txt")
```

En C++:

```
int main() {
    auto startTime = chrono::high_resolution_clock::now();
    string inputBook = "Libros/one.txt";
    string outputSA = "Output_CPlusPlus/outputSA_Cpp.txt";
```

Solo se debe cambiar la parte donde se encuentra *one.txt*, esta se puede cambiar actualmente a two.txt, three.txt y four.txt.

Nota: Para encontrar la variable más rápidamente se puede usar el comando CTRL + F.

4. Para buscar la cadena que desee, modifique el archivo “searchString.txt” y escriba la cadena de texto que busca encontrar.

5. Para la ejecución del archivo en Python o C++.

- Revisar las carpetas Output_CPlusPlus, OutputPython. Estas contienen los outputs del suffix array y de los indices de las ocurrencias del string indicado.

Liga repositorio de Github

https://github.com/David-Mtz-M/ActividadIntegradora1

Explicación del código

El código funciona a grandes rasgos de la siguiente manera:

getBuckets(T): Retorna el número de ocurrencias de cada elemento “T” y asigna un rango de valores a cada elemento único.

sais(T): Construye el arreglo de sufijos

search(string, pattern, SA): Busca un patron del string que se le pasa como parámetro usando el arreglo de sufijos SA. Esta función emplea una búsqueda binaria.

find_all_ocurrences(string, pattern, SA): Esta función usa la función search para encontrar la primer ocurrencia del patrón en el string dado. Después itera sobre las ocurrencias subsecuentes y regresa una lista de índices de dónde se encuentra cada patrón dentro del string.

read_file_content(filename): Regresa una cadena en formato unicode del libro completo.

print_memory_usage(): Imprime la cantidad de memoria en MiB que usa el programa durante su ejecución.

write_suffix_array(SA, outputSA): Escribe en un archivo el arreglo de sufijos en el PATH indicado.

write_total_ocurrences(ocurrences, outputOcurrences): Escribe en un archivo en donde se indican los indices de cada ocurrencia del string indicado previamente.

Diferencias entre paso de parámetros “por valor” y “por referencia”

Por valor

- Se pasa una copia del valor de la variable original a la función.
- Cualquier modificación realizada en el parámetro dentro de la función no afecta a la variable original fuera de la función.

Por referencia

- Se utiliza en lenguajes como C, C++ (mediante punteros).
- Cualquier modificación realizada en el parámetro dentro de la función afecta a la variable original fuera de la función.
- Se pasa una referencia o dirección de memoria a la variable original a la función

Comparación Python vs C++

Se usó la cadena “In”

Tiempo en segundos

Archivo	**Python**	**C++**
“one.txt”	5.88	8.54
“two.txt”	1.13	1.39
“three.txt”	14.053	15.81
“four.txt”	1.39	1.59

Memoria en MiB

Archivo	**Python**	**C++**
“one.txt”	107.86	96.59
“two.txt”	67.4	122.92
“three.txt”	139.05	148.23
“four.txt”	68.98	223.81

Comparación de tiempos (en segundos) con paso de parámetros por valor y referencia en C++

Por valor	Por referencia
11.17	8.54
4.21	1.39
20.35	15.81
4.63	1.59

Complejidad espacial y temporal

Debido a que la complejidad espacial y temporal se define a través de cuál es la función dentro de nuestro algoritmo de mayor complejidad. Se dice que tanto la complejidad espacial y temporal de los algoritmos es de O(n), siendo esta una complejidad lineal. Esto se debe a que el algoritmo SA-IS es uno de los algoritmos más eficientes con respecto a la construcción del arreglo de sufijos, cuyo tamaño es exactamente del mismo tamaño que la longitud de la cadena de texto usada.

Cabe recalcar que la complejidad espacial y temporal del programa es razonable para tamaños de entrada moderados. Sin embargo, es importante tener en cuenta que, para textos extremadamente grandes, el uso de memoria podría convertirse en un problema. Además, las estructuras de datos como mapas podrían ocupar más espacio del necesario y afectar el rendimiento en casos límite. En tales situaciones, se deberá recurrir a optimizaciones que permitan reducir el uso de memoria.

Reflexión personal

En esta tarea, he aprendido sobre la implementación de algoritmos avanzados relacionados con Suffix Arrays y la búsqueda de subcadenas en texto. Este código

en C++ representa una implementación de SA-IS, un algoritmo eficiente para construir el Suffix Array y usado para realizar búsquedas en texto, como las típicas que hacemos para buscar cadenas en páginas web usando CTRL + F.

Uno de los retos principales que enfrenté en esta tarea fue comprender la lógica y los detalles de la implementación del algoritmo SA-IS. Requiere un conocimiento profundo de conceptos como "buckets," "LMS" (Most Significant Suffix), y "induced sorting," que son fundamentales para su funcionamiento. Además, tuve que leer y buscar videos en Youtube para implementar la búsqueda binaria.

La importancia de aprender sobre algoritmos avanzados como SA-IS radica en su aplicación en problemas reales y de suma importancia. Estos algoritmos permiten realizar búsquedas eficientes en grandes conjuntos de datos, lo que es esencial en la actualidad en campos como la bioinformática, la recuperación de información y la compresión de datos. La construcción del Suffix Array es una técnica esencial para muchas aplicaciones, como la búsqueda de patrones, la comparación de genomas y la comprensión de texto. En mi caso, me sorprendió bastante saber que mi computadora que no tiene un hardware actual, pudo crear el suffix array de un libro de más de 200 páginas en menos de 12 segundos.

Esta tarea me ha enseñado la relevancia de comprender y aplicar algoritmos avanzados en la resolución de problemas prácticos. La eficiencia algorítmica es crucial en un mundo donde manejamos cantidades masivas de datos, y al aprender estos conceptos, he adquirido habilidades para abordar este tipo de problemas con menos miedo.