Resumen Laboratorio 1

Grupo 7: José Cristóbal Arroyo, Camilo Morillo, Nelson Rojas

August 2020

1 Toma de Datos

Los datos registrados a continuación fueron tomados con una laptop con 12GB de memoria RAM, y un procesador AMD Ryzen 3 3300U

1.1 Datos temporales

Archivo	**Tiempo Opción 1 (s)**	**Tiempo Opción 3 (s)**	**Columna Filtrada - Criterio**
test.csv	0,015625s	0s	("nombre") - Pedro
AllMoviesCastingRaw.csv	4,375s	0,321825s	("director_name") - Spielberg
MoviesCastingRaw-Small.csv	0,03125s	0s	("director_name") - Spielberg
AllMoviesDetailsCleaned.csv	8,40625s	0,25s	("genre") - Drama
SmallMoviesDetailsCleaned	0,0625s	0s	("genre") - Drama

En estos datos podemos observar que el tiempo de ejecución de los archivos con los registros completos (Los archivos AllMovies) Tuvieron una diferencia exorbitante en el tiempo de ejecución con respecto a los archivos small.

Cabe aclarar que estos datos fueron tomados de el promedio de 5 ejecuciones para cada uno de los archivos.

1.2 Cantidad de datos

Archivo	**Elementos Cargados**	**Cantidad de Columnas**	**Datos Totales Cargados**
test.csv	4	4	16
AllMoviesCastingRaw.csv	329044	19	6251836
MoviesCastingRaw-Small.csv	2000	19	38000
AllMoviesDetailsCleaned.csv	329044	22	7238968
SmallMoviesDetailsCleaned	2000	22	44000

La columna titulada "Datos Totales Cargados" Equivale a los "elementos cargados" (según la opción 2 del programa) Multiplicado por la cantidad de columnas, o la cantidad de llaves (keys) de los diccionarios.

La cantidad de datos cargados por cada archivo es muy importante a la hora de analizar los resultados de los datos de la sección 1.1 Puesto que podemos deducir que entre más datos se hayan cargado, más será el tiempo de ejecución.

1.2.1 ¿Qué orden de complejidad tendría las funciones (consulta y lectura de archivo)?

Como ya fue explicado anteriormente, estas funciones poseen una Complejidad Lineal O(N)

2 Análisis de Complejidad

2.1 Implementación requerimiento 1 Reto 1

2.1.1 ¿Cómo implementaría la función?

La función fue implementada de la siguiente forma:

```
def countElementsByCriteria(director_Name, lst_average, lst_director):
    print()
    print()
    t1_start = process_time()
    count=0
    sum=0
    for i in range(len(lst_average)):
        if float(lst_average[i]["vote_average"])>=6 and director_Name in lst_director[i]["director_name"]:
            count+=1
            sum+=float(lst_average[i]["vote_average"])
    promedio=sum/count
    t1_stop = process_time() #tiempo final
    print("Tiempo de ejecución ",t1_stop-t1_start," segundos")
    return count, promedio
```

Figure 1: Código de la función

2.1.2 ¿Qué orden de complejidad tendría la función implementada?

Esta función tiene una complejidad Lineal O(N)

2.1.3 ¿Cuál es el ciclo regular para actualizar código en un repositorio GIT?

Para cada branch (rama) en el repositorio, los usuarios hacen commits para actualizarlas, por último, debe hacerce un merge de ramas para actualizar el repositorio en la rama master

2.1.4 ¿Qué ventajas y limitantes tiene el uso de Ramas/Branches?

Ventajas: La posibilidad de trabajar en paralelo distintas versiones de un código
Desventajas: Los conflictos que pueden ocurrir entre las distintas branches

2.1.5 ¿Cuáles serían las buenas prácticas para solucionar conflictos?

Analizar la complejidad temporal y espacial del código para encontrar el más óptimo

2.1.6 ¿Cómo podría reducir o aumentar la complejidad de la consulta?

Aumentando o reduciendo la cantidad de datos

2.1.7 ¿Cómo afecta un TAD en la complejidad?, ¿Qué alternativas existen?

El TAD afecta en la complejidad espacial, puesto que es posible utilizar diccionarios o listas de tuplas o listas de diccionarios.