OBSERVACIONES DEL LA PRACTICA

María Alejandra Estrada García Cod 202021060

Santiago Martínez Novoa Cod 202112020

Preguntas de análisis

a. ¿Qué instrucción se usa para cambiar el límite de recursión de Python?

R//

```
if __name__ == "__main__":
    threading.stack_size(67108864)  # 64MB stack
    sys.setrecursionlimit(2 ** 20)
    thread = threading.Thread(target=thread_cycle)
    thread.start()
```

Se utiliza la función que se observa arriba, la línea más importante es la de sys.setrecurionlimit() y dentro de los paréntesis se le asigna el valor al que se quiere cambiar el límite de recursión, en este caso es 2^20.

b. ¿Por qué considera que se debe hacer este cambio?

R// Lo que pasa es que Python limita las profundidades de la recursión para evitar que el programa caiga en recursividades infinitas, que resultan en desbordamientos de pila. Por lo que si se implementa un programa como el del laboratorio que tiene una profundidad de recursión elevada los límites deben ser cambiados para que el programa no sea terminado antes de tiempo.

c. ¿Cuál es el valor inicial que tiene Python cómo límite de recursión?

R// En Python, el límite a la cantidad de veces que una función recursiva puede llamarse a sí misma está fijado en 1000 llamadas recursivas.

d. ¿Qué relación creen que existe entre el número de vértices, arcos y el tiempo que toma la operación 4?

R//

Archivo	Vértices	Arcos	Tiempo
Bus_routes_50	74	73	20.46479999999792 2
Bus_routes_150	146	146	26.99889999999868 5
Bus_routes_300	295	382	52.92669999999816
Bus_routes_1000	984	1633	259.7426000000014
Bus_routes_2000	1954	3560	806.2506000000012
Bus_routes_3000	2922	5773	1698.177499999999 8
Bus_routes_7000	6829	15334	4355.7282

Bus_routes_10000	9767	22758	14524.620999999992
Bus_routes_14000	13535	32270	24402.7262

A medida que aumentan los vértices, también aumenta la cantidad de arcos. De hecho, se puede evidenciar de que a medida que se aumenta la cantidad de datos, también aumenta la cantidad de arcos que esta tiene, por ejemplo, con 74 vértices el promedio está justo por debajo de 1, mientras que con 13535 ya cada vértice tiene en promedio 2.38 arcos. Así mismo, a medida que se aumenta la cantidad de datos a cargar, mayor es el tiempo de operación.

e. ¿El grafo definido es denso o disperso?, ¿El grafo es dirigido o no dirigido?, ¿El grafo está fuertemente conectado?

R//

```
analyzer['connections'] = gr.newGraph(datastructure='ADJ_LIST',
                                      directed=True,
                                      size=14000,
                                      comparefunction=compareStopIds)
```

El grafo es disperso, directo y está fuertemente conectado.

f. ¿Cuál es el tamaño inicial del grafo?

R// El tamaño del grafo es 14000 (size=14000).

g. ¿Cuál es la Estructura de datos utilizada?

R// Lista de adyacencias (ADJ_LIST), esta estructura de datos se utiliza para asociar a cada vértice *i* del grafo una lista que contenga todos aquellos vértices *j* que sean adyacentes a él. De esta forma sólo reservará memoria para los arcos adyacentes a *i* y no para todos los posibles arcos que pudieran tener como origen *i*.

h. ¿Cuál es la función de comparación utilizada?

R// Se usa la función compareStopIds, la cual compara los identificadores de las paradas de autobús.