OBSERVACIONES DE LA PRACTICA

Gabriel Villabon Cod 202013898
Natali Mercado Cod 202016282

Preguntas de análisis

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

Para cambiar el límite de recursión de Python se utiliza sys.setrecursionlimit(). Dicha función permite establecer un límite en la profundidad máxima del stack de Python con el fin de evitar un overflow del mismo por la recursividad infinita ("sys — System-specific parameters and functions — Python 3.9.5 documentation", 2001).

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

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

Este cambio se debe realizar porque el programa requiere una recursividad más profunda y, por ende, un límite más alto. Así, se evita que el intérprete muestre error (RecursionError exception) por exceder la recursión actual.

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

Se tiene como valor inicial mínimo de recursión para Python sys.setrecursionlimit(2 ** 4), sin embargo para la recursividad necesaria en el programa, este valor es muy pequeño.

```
RecursionError: maximum recursion depth exceeded while calling a Python object
```

El valor mínimo necesario es para correr la función 2 es:

```
156  if __name__ == "__main__":
157      threading.stack_size(67108864)  # 64MB stack
158      sys.setrecursionlimit(2 ** 5)
159      thread = threading.Thread(target=thread_cycle)
160      thread.start()
161
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Cargando información de transporte de singapur ....
Numero de vertices: 13535
Numero de arcos: 32270
El limite de recursion actual: 32
```

El valor máximo de recursion para Python sys.setrecursionlimit(2 ** 30), a partir de n>30 el valor supera el limite de recursión.

```
156  if __name__ == "__main__":
157      threading.stack_size(67108864)  # 64MB stack
158      sys.setrecursionlimit(2 ** 30)
159      thread = threading.Thread(target=thread_cycle)
160      thread.start()
161
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Cargando información de transporte de singapur ....
Numero de vertices: 13535
Numero de arcos: 32270
El limite de recursion actual: 1073741824
```

Análisis de opción 4 y 6

Máquina 1 - opción 4			
Archivo CSV	**Vértices**	**N° de arcos**	**Tiempo [ms]**
bus_routes_50	74	73	37,51
bus_routes_150	146	146	52,06
bus_routes_300	295	382	83,12
bus_routes_1000	984	1633	400,65
bus_routes_2000	1954	3560	1295,39
bus_routes_3000	2922	5773	2153,36
bus_routes_7000	6829	15334	4856,72
bus_routes_10000	9767	22758	21751,41
bus_routes_14000	13535	32270	29507,23

Máquina 2 - opción 4			
Archivo CSV	**Vértices**	**N° de arcos**	**Tiempo [ms]**
bus_routes_50	74	73	22,89
bus_routes_150	146	146	37,64
bus_routes_300	295	382	61,36
bus_routes_1000	984	1633	303,43
bus_routes_2000	1954	3560	843,76
bus_routes_3000	2922	5773	1853,86
bus_routes_7000	6829	15334	6276,94
bus_routes_10000	9767	22758	15199,05
bus_routes_14000	13535	32270	22900,1

Máquina 1 - opción 6			
Archivo CSV	**Vértices**	**N° de arcos**	**Tiempo [ms]**
bus_routes_50	74	73	0,8
bus_routes_150	146	146	1
bus_routes_300	295	382	1,02
bus_routes_1000	984	1633	1,04
bus_routes_2000	1954	3560	1,45
bus_routes_3000	2922	5773	1,68
bus_routes_7000	6829	15334	1,16
bus_routes_10000	9767	22758	1,35
bus_routes_14000	13535	32270	2,8

Máquina 2 - opción 6			
Archivo CSV	**Vértices**	**N° de arcos**	**Tiempo [ms]**
bus_routes_50	74	73	0,4
bus_routes_150	146	146	0,52
bus_routes_300	295	382	0,43
bus_routes_1000	984	1633	0,42
bus_routes_2000	1954	3560	0,73
bus_routes_3000	2922	5773	0,59
bus_routes_7000	6829	15334	0,62
bus_routes_10000	9767	22758	1,18
bus_routes_14000	13535	32270	0,84

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

Entre mayor sea la cantidad de datos diferentes mayor es la cantidad de vértices respecto a una cantidad inferior de datos. Esto a su vez aumenta la cantidad de arcos para

un mismo vértice debido a la conexión de una nueva ruta respecto a un arco ya inicialmente conectado. Por lo anterior, el tiempo que toma la función en establecer el camino aumenta.

e) ¿Qué características tiene el grafo definido?

Tiene como vértices las diferentes estaciones y como arcos las conexiones de los buses que pasan por esa estación.

f) ¿Cuál es el tamaño inicial del grafo?

El grafo tiene inicialmente 14000 datos.

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

g) ¿Cuál es la Estructura de datos utilizada?

Se tiene como estructura de datos una lista de adyacentes, la cual consiste en una matriz de listas separadas donde cada elemento de la matriz es una lista, que contiene todos los vértices adyacentes a un vértice i ("Graph Representation Tutorials & Notes | Algorithms | HackerEarth", n.d.).

```
e puede implementar sobre una lista de adyacencias (ADJ_LIST)
```

h) ¿Cuál es la función de comparación utilizada?

```
def compareStopIds(stop, keyvaluestop):
    """
    Compara dos estaciones
    """
    stopcode = keyvaluestop['key']
    if (stop == stopcode):
        return 0
    elif (stop > stopcode):
        return 1
    else:
        return -1
```

Compara si existe una estacion como vertice

Bibliografía

sys — System-specific parameters and functions — Python 3.9.5 documentation. (2001). Retrieved 17 May 2021, from https://docs.python.org/3/library/sys.html#sys.setrecursionlimit

Graph Representation Tutorials & Notes | Algorithms | HackerEarth. Retrieved 17 May 2021, from https://www.hackerearth.com/practice/algorithms/graphs/graph-representation/tutorial/#:~:text=An%20adjacency%20list%20is%20an,in%20the%20li