OBSERVACIONES DEL LA PRACTICA

Nicolas Merchan Cuestas - 202112109

Preguntas de análisis

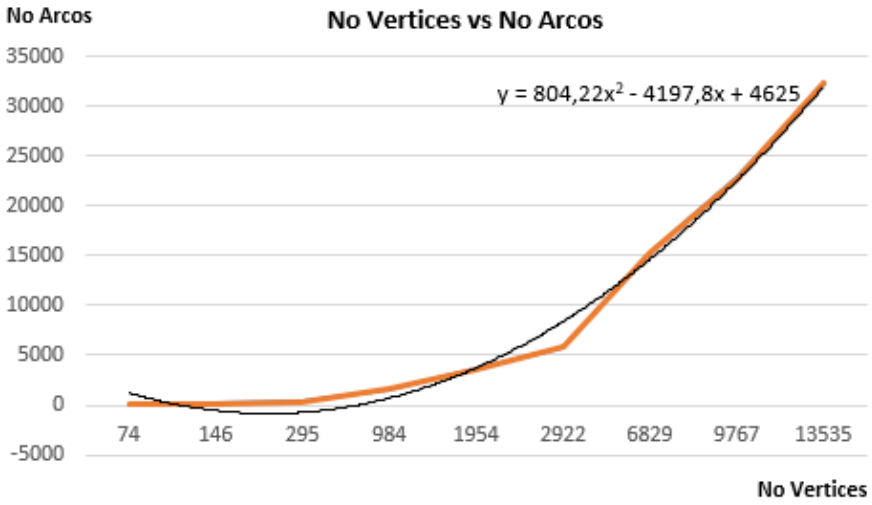
a) ¿Qué instrucción se usa para cambiar el límite de recursión de Python?
El límite de recursión de Python puede ser cambiado mediante la instrucción `sys.setrecursionlimit(recursión_limit)`, donde la variable `recursión_limit` representa el límite de recursión deseado. Se importó la librería `sys`.

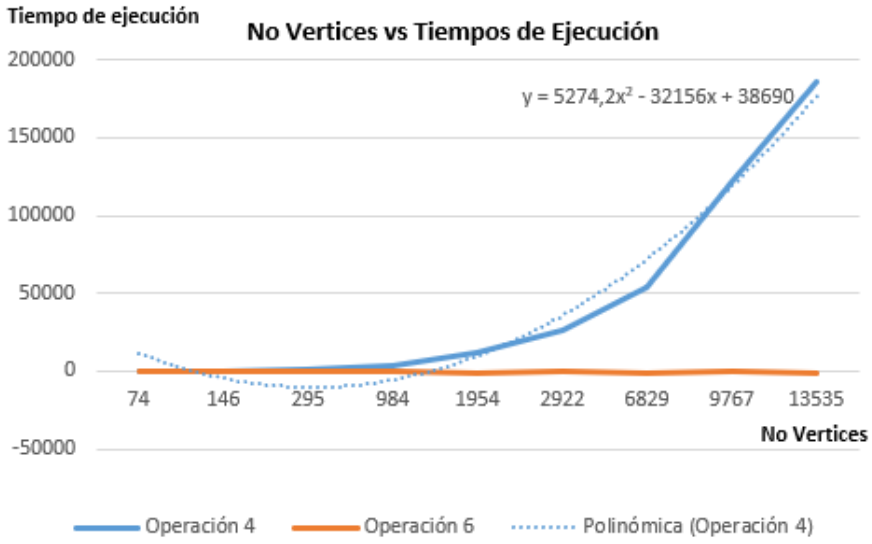
b) ¿Por qué considera que se debe hacer este cambio?
El límite de recursión debe ser cambiado, porque la construcción y análisis de los gráfos que representan el sistema considerado requieren de muchos más procesos recursivos que el límite establecido por defecto en python.

c) ¿Cuál es el valor inicial que tiene Python cómo límite de recursión?
El valor inicial del límite de recursión en Python puede ser obtenido mediante la instrucción `sys.getrecursionlimit()`, he importado la librería `sys`. El resultado por defecto de la instrucción son 1000 recursiones.

d) ¿Qué relación creen que existe entre el número de vértices, arcos y el tiempo que toma la operación 4?
Considero que el número de arcos crece exponencialmente con el incremente en el número de vértices. Así mismo, el tiempo de ejecución de la opción 4 incrementa exponencialmente tanto con el número de vértices como de arcos.

No. Paradas	Vertices	Arcos	Tiempo - Operación 4 (mseg)	Tiempo – Operación 6 (mseg)
50	74	73	218,75	16
150	146	146	359	31,25
300	295	382	672	16
1000	984	1633	3343,75	16
2000	1954	3560	11968,75	0
3000	2922	5773	26843,75	16
7000	6829	15334	53468,75	0
10000	9767	22758	121843,625	31,25
14000	13535	32270	185625	0


No Arcos
No Vertices vs No Arcos
35000
30000
25000
20000
15000
10000
5000
0
-5000
y = 804,22x² - 4197,8x + 4625
74
146
295
984
1954
2922
6829
9767
13535
No Vertices


Tiempo de ejecución
No Vertices vs Tiempos de Ejecución
200000
150000
100000
50000
0
-50000
y = 5274,2x² - 32156x + 38690
74
146
295
984
1954
2922
6829
9767
13535
No Vertices
Operación 4
Operación 6
Polinómica (Operación 4)


Tiempo de ejecución
No Arcos vs Tiempos de Ejecución
200000
150000
100000
50000
0
-50000
y = 5274,2x² - 32156x + 38690
73
146
382
1633
3560
5773
15334
22758
32270
No Arcos
Operación 4
Operación 6
Polinómica (Operación 4)

e) ¿El grafo definido es denso o disperso?, ¿El grafo es dirigido o no dirigido?, ¿El grafo está fuertemente conectado?
El grafo definido es disperso, dado que para los 9 archivos analizados se da que

$$\frac{A}{V(V-1)} < 0.3,$$

para A igual al número de arcos, V igual al número de vértices y V(V-1) igual al máximo número de arcos que pueden existir entre V vértices. El grafo es dirigido, dado que las rutas tienen una dirección específica entre las estaciones (enunciado). El grafo está fuertemente conectado, dado que siempre existe una ruta para llegar de una estación a otra y viceversa. Dicha ruta no tiene que ser igual cuando se busca retornar a la estación de origen desde la estación de destino (enunciado).

f) ¿Cuál es el tamaño inicial del grafo?
El tamaño inicial del grafo es de 14000 elementos. Ello se evidencia en el parámetro `size=1400` (línea 68) de la función `gr.newGraph()` en `model.py`.

g) ¿Cuál es la Estructura de datos utilizada?
La estructura de datos utilizada es una lista de adyacencias. Ello se evidencia en el parámetro `datastructure='ADJ_LIST'` (línea 66) de la función `gr.newGraph()` en `model.py`.

h) ¿Cuál es la función de comparación utilizada?
La función `compareStopsIds()` es utilizada en la creación del grafo definido y comprara los códigos referentes a las estaciones ingresadas por parámetro y retorna 0 cuando los códigos de las estaciones son iguales, 1 cuando el primer código es superior al código posterior ingresado por parámetro, y -1 cuando el primer código es inferior al código posterior ingresado por parámetro. Ello se evidencia en el parámetro `comparefunction=compareStopsIds` (línea 69) de la función `gr.newGraph()` en `model.py`