OBSERVACIONES DEL LA PRACTICA

Juan David Salguero, 201923136, J.salguero@uniandes.edu.co
David Molina, 202125176, d.molinad@uniandes.edu.co

Preguntas de análisis

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

Se encuentra en el archivo view.py del repositorio como sys.setRecursionlimit(), la instrucción es la siguiente:

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

Este cambio es necesario para iterar archivos de gran tamaño, de otra manera, python pensara que cualquier función que se llame así misma está en un loop infinito y cancelará la operación.

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

Según las librerías de python, el valor inicial es de 1000 recursiones.

Observaciones opción 4

Tamaño de datos	Vértices	Arcos	Tiempo (ms)
50 Routes	74	73	36.238
150 Routes	146	146	49.802
300 Routes	295	382	77.122
1000 Routes	984	1633	411.923
2000 Routes	1954	3560	1334.391
3000 Routes	2922	5773	2201.768
7000 Routes	6829	15334	9394.412
10000 Routes	9767	22758	20788.228
14000 Routes	13535	32270	43859.660

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

En general se observa una relación de crecimiento en el tiempo de ejecución entre mayor sea la cantidad de vértices y arcos. Entre menos vértices se creen en el grafo, menos arcos son

necesarios para conectarlos; en consecuencia, menos iteraciones debe hacer el algoritmo para encontrar la ubicación de la estación dada por input.

Observaciones opción 6

Tamaño de datos	Vértices	Arcos	Tiempo (ms)
50 Routes	74	73	11.047
150 Routes	146	146	2.435
300 Routes	295	382	2.386
1000 Routes	984	1633	310.119
2000 Routes	1954	3560	4.512
3000 Routes	2922	5773	2.240
7000 Routes	6829	15334	32.121
10000 Routes	9767	22758	17.103
14000 Routes	13535	32270	17.381

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

En general la mayoría de los grafos probados tienen un número de arcos cercano o mayor al número de vértices, como se ilustra en la tabla anterior. Por ende, se piensa que el grafo es denso.

Por otra parte, el grafo es dirigido, pues se crean conexiones entre los vértices. No obstante, no está fuertemente conectado, pues cada estación crea un arco con sus estaciones adyacentes, pero no vice versa.

```
def addStopConnection(analyzer, lastservice, service):
    """
    Adiciona las estaciones al grafo como vertices y arcos entre las
    estaciones adyacentes.

    Los vertices tienen por nombre el identificador de la estacion
    seguido de la ruta que sirve.  Por ejemplo:

    75009-10

    Si la estacion sirve otra ruta, se tiene: 75009-101
    """
    try:
        origin = formatVertex(lastservice)
        destination = formatVertex(service)
        cleanServiceDistance(lastservice, service)
        distance = float(service['Distance']) - float(lastservice['Distance'])
        distance = abs(distance)
        addStop(analyzer, origin)
        addStop(analyzer, destination)
        addConnection(analyzer, origin, destination, distance)
        addRouteStop(analyzer, service)
        addRouteStop(analyzer, lastservice)
        return analyzer
```

f) ¿Cuál es el tamaño inicial del grafo?

14000, como se muestra en la creación del grafo.

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

g) ¿Cuál es la estructura de datos utilizada?

De la librería DISC, se importan y utilizan las estructuras de tipo grafo.

```
from DISClib.ADT import graph as gr
```

h) ¿Cuál es la función de comparación utilizada?

Se utiliza compareStopIds, que compara dos estaciones al comparar sus llaves, como se muestra a continuación.

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