Estructuras de Datos y Algoritmos

Jose Luis Tavera - 201821999
Juan Diego Yepes - 202022391

Laboratorio X

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

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

Esta función crea un thread, es decir un mini-programa que ejecuta la función thread_cycle, que es el programa principal. El límite de recursión se establece con llamando a la librería sys y ejecutando la función setrecursionlimit(), donde se inserta el valor de 2^{20}. Al crear el hilo o thread, la ejecución se realiza con una pila de 64MB, y el límite de recursión le indica al sistema que se pueden realizar más de 1000 recursiones que es el valor por defecto.

1.1 ¿Por qué considera que se debe hacer este cambio?

Este cambio es importante dado que permite la ejecución del programa; puesto que si no se actualizara el límite de recursión, el programa simplemente no se ejecutaría porque los grafos no se procesarían adecuadamente.

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

Como se menciona anteriormente, el límite por defecto de recursión es de 1000

2 ¿Qué relación creen que existe entre el número de vértices, arcos y el tiempo que toma la operación cuatro?

Para la opción cuatro en la ejecución del punto de partida en cada uno de los archivos obtuvimos una relación entre el número de vértices, arcos y el tiempo que podemos apreciar en la siguiente tabla:

Tamaño	Tiempo[ms]	Vertices	Arcos	Límite de Recursión
50	45,34	74	73	1048576
150	53,115	146	146	1048576
300	83,925	295	382	1048576
1000	272,652	984	1633	1048576
2000	668,969	1954	3560	1048576
3000	1317,486	2922	5773	1048576
7000	4276,260	6829	15334	1048576
10000	10904,995	9767	22758	1048576
14000	18217,163	13535	32270	1048576

Tabla 1: Tiempo, Vértices, Arcos y Límite de Recursión para todos los archivos con la opción cuatro

La opción cuatro del ejemplo, referencia la función homónima de "optionFour", con la que busca el los caminos de costo mínimo desde la estación inicial a todas.

Funciones del View

```
elif int(inputs[0]) == 4:
            msg = "Estacion Base: BusStopCode-ServiceNo (Ej: 75009-10): "
            initialStation = input(msg)
            optionFour(cont, initialStation)
```

```
def optionFour(cont, initialStation):
    controller.minimumCostPaths(cont, initialStation)
```

Funciones del Controller

```
def minimumCostPaths(analyzer, initialStation):
    """
    Calcula todos los caminos de costo minimo de initialStation a todas
    las otras estaciones del sistema
    """
    return model.minimumCostPaths(analyzer, initialStation)
```

Funciones del Model

```
def minimumCostPaths(analyzer, initialStation):
    """
    Calcula los caminos de costo m nimo desde la estacion initialStation
    a todos los demas vertices del grafo
    """
    analyzer['paths'] = djk.Dijkstra(analyzer['connections'], initialStation)
    return analyzer
```

En la función del model se evidencia que se está usando de la librería de algoritmos el algoritmo de Dijkstra (hecho congruente con la teoría ya que se trata de un problema de caminos mínimos). Por lo que la pregunta se reduce a la relación del número de arcos y vértices en el órden de crecimiento de Dijkstra.

Tamaño	Tiempo[ms]	Vertices	Arcos
50	1,632	60	59
150	1,772	60	59
300	1,740	66	65
1000	1,365	64	63
2000	3,618	81	80
3000	2,004	82	81
7000	3,804	96	95
10000	2,995	113	112
14000	3,760	137	136

Tabla 1: Tiempo, Vértices, Arcos y Límite de Recursión para todos los archivos con la opción seis

En esta tabla se evidencia el tiempo que tarda en encontrar el camino más corto entre las estaciones dadas en el ejemplo (estación base: 75009-10 a la estación destino: 15151-10), y cuántos arcos y vértices debe tener.

3 ¿Qué características tiene el grafo definido?

La definición del grafo la podemos encontrar en el model, específicamente en la creación del analyzer en donde en la función de newGraph se especifícan los siguientes parámetros:

- **datastructure:** ADJ_LIST
- **directed:** True

- **size:** 14000
- **comparefunction:** compareStopIds

Por lo tanto, podemos afirmar que el grafo creado es un grafo dirigido, con un estructura de datos de lista adyacente, de tamaño igual a 14000 vértices y con una función de comparación preestablecida.

```
def newAnalyzer():
    """ Inicializa el analizador

   stops: Tabla de hash para guardar los vertices del grafo
   connections: Grafo para representar las rutas entre estaciones
   components: Almacena la informacion de los componentes conectados
   paths: Estructura que almancena los caminos de costo minimo desde un
           vertice determinado a todos los otros v rtices del grafo
    """
    try:
        analyzer = {
                    'stops': None,
                    'connections': None,
                    'components': None,
                    'paths': None
                    }

        analyzer['stops'] = m.newMap(numelements=14000,
                                     maptype='PROBING',
                                     comparefunction=compareStopIds)

        analyzer['connections'] = gr.newGraph(datastructure='ADJ_LIST',
                                              directed=True,
                                              size=14000,
                                              comparefunction=compareStopIds)
        return analyzer
    except Exception as exp:
        error.reraise(exp, 'model:newAnalyzer')
```

4 ¿Cuál es el tamaño inicial del grafo?

Como se evidencia en la creación del grafo la el tamaño inicial del gráfo corresponde a 14000 vértices que están definidos en el parámetro de size.

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

5 ¿Cuál es la estructura de datos utilizada?

Como se evidencia en la creación del grafo la estructura de datos utilizada es una lista adyacente definida en el parámetro de datastructure

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

6 ¿Cuál es la función de comparación utilizada?

En la creación del grafo se estipula dentro de sus parámetros como compare function la función de compareStopIds, que se muestra a continuación:

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

La cual haciendo uso del mapa definido en el analyzer busca el valor correspondiente a cada parada que es igual a un código único con el que compara si son iguales o no para devolver 0 o -1 dependiendo del caso.