**Preguntas lab-09**

1. ¿Qué instrucción se usa para cambiar el límite de recursión de Python?
2. ¿Por qué considera que se debe hacer este cambio?
3. ¿Cuál es el valor inicial que tiene Python cómo límite de recursión?
4. ¿Qué relación creen que existe entre el número de vértices, arcos y el tiempo que toma la operación 4?
5. ¿El grafo definido es denso o disperso?, ¿El grafo es dirigido o no dirigido?, ¿El grafo está fuertemente conectado?
6. ¿Cuál es el tamaño inicial del grafo?
7. ¿Cuál es la Estructura de datos utilizada?
8. ¿Cuál es la función de comparación utilizada?

**Respuestas:**

1. sys.setrecursionlimit(), esta función permite cambiar el limite recursión máximo de Python, es decir permite cambiar la cantidad máxima de veces que un algoritmo pueda ser llamado así mismo, o utilizar recursión en él, el limite por defecto es 1000 llamado recursivos. Esto permite que si un ciclo es infinito, Python sea capaz de pararlo y dejar de consumir memoria.
2. Se puede dar la ocasión, que nos toque recorrer todo el grafo, por lo que sí es un grafo de tamaño 14000, Python detendría su ejecución y no nos daría ningún resultado.
3. 1048576, en mi maquina ese es el límite de recursión, lo obtuve por la función sys.getrecursionlimit()
4. Lo que tarda la función djk.Dijkstra() es igual a o(|vértices|+|arcos|\*ln(vértices)), por lo cual se relacionan directamente el número de vértices y arcos con el tiempo que tardara el algoritmo en ejecutarse, esto también depende de la máquina y la velocidad de reloj que maneje.
5. El grafo so está fuertemente conectado, pues todos sus vértices están conectados con otros, es decir se puede llegar siempre a todos los vértices desde un vértice A, es un grafo dirigido ya que tiene un peso asociado desde un vértice A, a un vértice B, es por eso que se puede hallar y usar el algoritmo de costo mínimo, el grafo es denso, debido a que creamos 14 mil espacios y usamos 13535 espacios, por lo cual es denso ya que no tiene muchos espacios libres.
6. En el caso del archivo de 14000 tenemos esta cantidad, Numero de vértices: 13535, Numero de arcos: 32270.
7. Se utiliza la estructura de datos Graph, implementada en el TAD DISC.LIB, además se crea un mapa donde se aloja la información de cada vértice.
8. def compareroutes(route1, route2):

    """

    Compara dos rutas

    """

    if (route1 == route2):

        return 0

    elif (route1 > route2):

        return 1

    else:

        return -1

Esta función compara el peso de dos rutas ingresadas por parámetro, a diferencia de la función utilizada por un mapa, en esta no hay necesidad de obtener la llave.