OBSERVACIONES DEL LA PRACTICA

Ana Sofía Castellanos Mosquera202114167

Martín Santiago Galván Castro 201911013

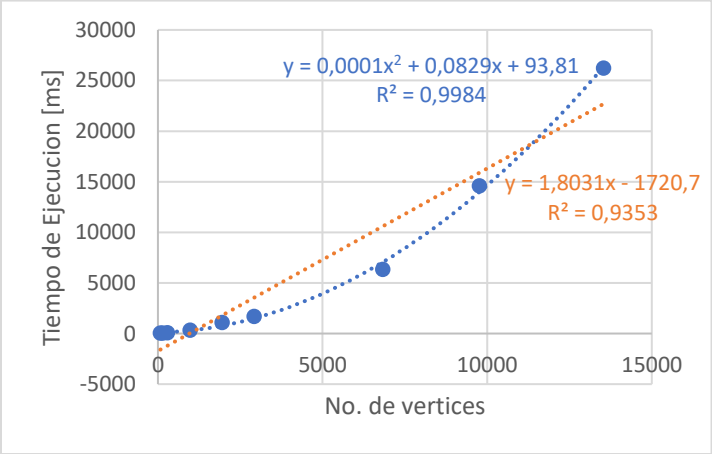
Preguntas de análisis

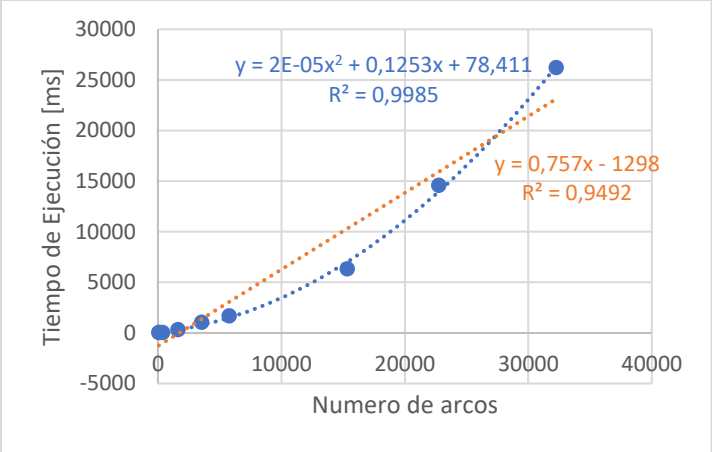
a) ¿Qué instrucción se usa para cambiar el límite de recursión de Python?
Al final del código en view, se encuentra un bloque de código el cual es el que se encarga de cambiar el limite de recursión de Python. Más específicamente, el comando que realiza este cambio se realiza con la librería "sys", con el comando sys.setrecursionlimit(n). Donde n es el limite de recursión que se desea hacer.

b) ¿Por qué considera que se debe hacer este cambio?
Es necesario realizar el cambio del límite de recursión para poder llevar a cabo las operaciones del programa. Esto se puede comprobar experimentalmente cambiando el valor que se puso en el limite de recursión del programa. Originalmente, este valor es igual a 2^{20}. Al cambiar el valor de la potencia a un numero menor, como un 10, y se intenta ejecutar las opciones del programa, Python mostrara un error de límite de recursión. Esto se debe a que, por ejemplo, en el algoritmo de la opción 4 que busca los caminos mas cortos, se utiliza el algoritmo de dijsktra, el cual se basa en la recursión para llevar a cabo el algoritmo. Dado a que se importan muchos vértices, o al menos una cantidad importante, si no se cambia el limite de recursión, puede que el código no pueda llevarse a cabo.

c) ¿Cuál es el valor inicial que tiene Python cómo límite de recursión?
Python posee por default un limite de recursión de 1000.

d) ¿Qué relación creen que existe entre el número de vértices, arcos y el tiempo que toma la operación 4?
Para poder aproximarse a encontrar la relación entre el número de vértices y arcos con el tiempo de ejecución de la operación 4, se realizaron 2 graficas. Estas muestran la relación entre las dos variables y el tiempo de ejecución. En las variables se incluyen dos líneas de tendencia. Una lineal y otra cuadrática. Se determina cual es la verdadera dependencia analizando el valor de la variable R^2.





Analizando ambas graficas, se puede encontrar de manera experimental que, para el numero de vertices y el numero de arcos, la relación con el tiempo de ejecución aparenta ser cuadratico.

Analizando mas a fondo la funcion del codigo, podemos encontrar que la opcion 4, que busca el camino mas corto a partir de una base, utiliza el algoritmo de dijkstra. Buscando información de este algoritmo, se puede encontrar que este puede poseer dos complejidades dependiendo de como se implemente.

Si el algoritmo se implementa sin el uso de colas de prioridad, se encuentra que la complejidad del algoritmo es de $O(V^2)$. En donde V es el numero de vertices. Pero, si se utiliza la cola de prioridad, la complejidad del algoritmo se vuelve $O(A \log(V))$, donde A es el numero de arcos. Analizando el codigo que se utiliza, se puede revisar que la implementacion el el laboratorio utiliza colas de prioridad. Sin embargo, por falta de una metodologia para determinar si lo anterior es cierto, se considera por el momento que la relación entre las variables es cuadratica. Pero se podria argumentar que por el bajo valor del coeficiente de la expresión cuadratica, se tiende una relación linearitmica

e) ¿Qué características tiene el grafo definido?

   El grafo que se define en analyzer[‘connections’] es un grafo dirigido lo cual implica que no se puede recorrer de forma bidireccional dos vértices (estaciones), a menos que existan dos arcos uno de A - B y otro de B - A que los conecte. De esta manera, puede existir una conexión en una dirección, pero no necesariamente en la otra, es decir puede existir una ruta entre una estación X a una Y pero puede no haber una de Y a X.

f) ¿Cuál es el tamaño inicial del grafo?

   El grafo se construye inicialmente con un tamaño de 14000 elementos.

g) ¿Cuál es la Estructura de datos utilizada?

Dentro del analizador se hace uso de diversas estructuras de datos. En primer lugar, analyzer['stops'] es un Map que guarda los vértices del grafo y que implementa una estructura de datos de tabla de Hash con manejo de colisiones Linear Probing.

En segundo lugar, analyzer['connections'] es un grafo que representa las rutas de las estaciones y está implementado mediante una estructura de datos de Lista de Adyacencia.

En tercer lugar, analyzer['components'] llama a la función KosarajuSS(graph)que se encuentra en la carpeta DISClib, Algorithms, scc, la cual saca los componentes fuertemente conectados y devuelve diccionario que a su vez implementa tres Maps con una estructura de datos del tipo Linear Probing, en el cual se relacionan el número del componente en el que está cada vértice y se marcan los vértices.

Finalmente, analyzer['paths'] llama a la función Dijkstra, que se encuentra en la carpeta DISClib, Algotirhms, dijsktra, la cual retorna un diccionario que tiene en source el nombre del vértice fuente, en iminpq una cola de prioridad y en visited un Map implementado con el sistema de manejo de colisiones Linear Probing.

h) ¿Cuál es la función de comparación utilizada?

Se hace uso de la función de comparación compareStopIds, esta función recibe como parámetros un valor stop y keyvaluestop, a partir de esto compara dos estaciones y retorna 0 si son iguales, 1 si stop es mayor que la llave de keyvaluestop y –1 si stop es menor que la llave de keyvaluestop.