**Factor de carga de cada elemento**

**Reto 4**

Felipe Rueda Rivera 202010903

Julian Rivera 202013033

Texto

Descripción generada automáticamente**Carga de datos:**

Dentro del analyzer cargamos lo siguiente:

* Vertices
* Arcos
* Components
* Paths
* landing\_points
* paises\_nombre
* países\_codigo

Se puede evidenciar que se guardo un grafo los cuales son “Vertices”,” Arcos”, “components”, “paths”, los vértices se guardo en una tabla de hash tipo probing, por lo que como vimos antes, si hay colisiones no guarda en un bucket sino que encuentra la siguiente posición vacía. Los arcos son guardados en una lista de adyacencia y se observa que este grafo es dirigido. Tenemos pensado guardar en este grafo como arcos los códigos de los landing points y los pesos de los arcos es la distancia en kilómetros que unen ambos landing points.

Adicionalmente, como el reto nos pide que como la entrada como el resultado piden nombre de los landing points o el nombre de los países y en un requerimiento nos pide otra información de cada landing point. Entonces, en estas tres tablas de hash “landing\_points” donde tenemos pensado poner como llave el nombre del landing point y como valor la información del cada landing point,”países\_nombre” donde vamos a poner como llave el nombre del país y como valor va a ser el código de dicho país. Finalmente, la tabla de hash llamada ”países\_codigo” tenemos como llave el código de los países y como valor una lista con la información de cada país.

Texto

Descripción generada automáticamente**Grafo (llave 1,2,3,4 de analyzer):**

Como se dijo antes en el grafo se va a guardar como vértices son los códigos de los landing points y como pesos de los arcos es la distancia que esta entre los landing points. Para obtener esta información se extrajo del archivo llamado “connections.csv”

**Tablas de hash (llave 5,6,7 de analyzer):**

Texto

Descripción generada automáticamente

Como se dijo antes cada de estas tablas tienen diferentes llaves, pues en la tabla de hash llamada “landing\_points” la llave es el nombre del landing y como valor como se dijo antes es una lista con la información del landing point. Así mismo se guarda información en las demás tablas de hash sino con diferente llave o diferente valor.

**Requerimiento 1:**

Texto

Descripción generada automáticamente

En el requerimiento uno se implementa el método de kosaraju pues en este requerimiento se pide el número total de clústeres presentes en la red, además de informar que si los dos landings points que inserto el usuario están en el mismo clúster. Se utilizo el método de Kosaraju pues este método pues esta marca cada vértice que está en el mismo clúster pues este utiliza Depth First Order (DFO) y tiene complejidad de O(EV).

En el caso para informar si los landing points están en el mismo clúster, esto quiere decir que ambos vértices deben estar fuertemente conectados, por lo que se utilizó una función que se puede encontrar en SCC, esta función tiene tres parámetros uno el grafo y los otros dos son los vértices que se quieren buscar, y por un medio de un booleano informa si efectivamente ambos vértices están fuertemente conectados o están en el mismo clúster.

Texto

Descripción generada automáticamentePrueba requerimiento 1:

**Requerimiento 2:**

****

En el requerimiento dos tiene que dar de respuesta el total de landing points y el total de cables conectados a dichos landing points, para completar este requerimiento se utilizó una función básica de TAD Graph llamada vertices, esta función retorna una lista con todos los vértices del grafo, esta función tiene como parámetro el grafo. Para sacar el total de cables conectados se utiliza otras dos funciones básicas de TAD Graph las cuales son degree e indegree estas dos retornan el numero de arcos que llegan al vertice y que salen del vertice por lo que se sumo con un total, así se observa el total de cables conectados a los landing points.

Prueba:

Texto

Descripción generada automáticamente

**Requerimiento 3:**

**Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente**En el requerimiento tres tiene que dar como respuesta la ruta en esta se tenía que incluir la distancia entre los landing points y la distancia total de la ruta. Para este requerimiento se utilizó el algoritmo de Dijkstra y su complejidad es de O(E logV) pues esta es el algoritmo más eficaz en el peor caso para encontrar el mínimo camino. En este caso, se implementó el algoritmo de Dijkstra con la función “minimunCostPaths”, después con la función “pathTo” se encontró el camino desde el país a al país b con el mínimo camino y con la función “distTo” se calcula la distancia total de la ruta.

Texto

Descripción generada automáticamentePrueba: