LABORATORIO No. 11: Algoritmos en Grafos

Objetivos

Aprender a utilizar los algoritmos del Tipo Abstracto de Datos Grafo (graph) dentro de un problema definido. Al finalizar este laboratorio el estudiante estará en capacidad de:

1. Identificar los algoritmos principales implementados en el TAD graph.
2. Entender el funcionamiento de los algoritmos de grafos y como pueden solucionar requerimientos funcionales dentro de un problema definido.
3. Entender los órdenes de crecimiento espacial y temporal de los algoritmos.
4. Integrar los Grafos con las otras estructuras de datos vistas en el curso.

Fecha Límite de Entrega

Recuerde que para el final de la sesión del laboratorio los miembros del grupo deben por lo menos iniciar las actividades del **PASO 6: Actualizar el repositorio del laboratorio.**

La entrega completa del laboratorio hasta el **PASO 13: Compartir resultados con los evaluadores** será el martes 22 de noviembre antes de la media noche (11:59 p.m.).

Preparación del Laboratorio

Revisar el API del TAD Map ubicado en "**DISClib\ADT\graph.py"** y específicamente las estructuras de datos **graphstructure.py** y **adjlist.py** ubicado en "**DISClib\DataStructures".**

Por último, Revise los diferentes archivos ***.py** de los algoritmos en la carpeta **DISClib\Algorithms***

Trabajo Propuesto

PASO 1: Copiar el ejemplo en su organización

Copie/Haga **Fork** del repositorio del laboratorio en su organización con el procedimiento aprendido en las prácticas anteriores.

El repositorio del proyecto base que utiliza este laboratorio es el siguiente

- https://github.com/ISIS1225DEVS/ISIS1225-SampleAlgorithm.git

Antes de clonar el repositorio en su computador diríjase a su organización (Ej.: ***EDA2022-1-SEC02-G01*** para el primer grupo de la sección 2 del curso) y cambie el nombre del repositorio de acuerdo con el esquema **LabAlgorithm-S<<XX>>-G<<YY>>** donde **XX** es el número de la práctica de laboratorio y donde

YY es el número del grupo de trabajo. (Ej.: **LabGraph-S11-G01** para este **onceavo laboratorio** hecho por el **grupo 1** de la **sección 2).**

Recuerde que **NO necesita** agregar la sección o el semestre en este nombre porque ya está identificado en su organización.

PASO 2: Descargar el ejemplo

Después de renombrar el proyecto dentro de su organización ya puede clonar el proyecto. Descargue el código en su máquina local **(git clone)** siguiendo lo aprendido en las practicas anteriores.

Recuerde modificar el **README** del repositorio para incluir los nombres de los integrantes del grupo.

PASO 3: Entender la fuente de datos y la construcción del Grafo

Antes de iniciar a explorar y modificar el ejemplo, recuerde descargar los datos de trabajo **singapur_bus_routes** disponibles en el portal oficial del curso en BrightSpace. Descargue el **Zip,** descomprímalo y guarde los archivos CSV en la carpeta ***/Data/** de su copia local de código.

El conjunto de datos contiene información de rutas de buses, paraderos y distancias entre los paraderos del sistema de buses de la ciudad de Singapur del proyecto **Singapore Bus Data (Land transport authority)**[1]**.** Los archivos de rutas tienen la siguiente información:

- ServiceNo,
- Operator,
- Direction,
- StopSequence,
- BusStopCode,
- Distance,
- WD_FirstBus,
- WD_LastBus,
- SAT_FirstBus,
- SAT_LastBus,
- SUN_FirstBus,
- SUN_LastBus

Para crear el grafo se utilizan los archivos con las rutas y la secuencia de paraderos, así como la distancia entre paradas. Tenga en cuenta que una misma parada puede servir a más de una ruta, por lo cual los vértices del grafo tendrán la siguiente estructura: `<BusStopCode>-<ServiceNo>.`

Por ejemplo: `‘75009-10’` para indicar que esa parada 75009 es para la ruta 10 y `‘75009-101’` para indicar que esa misma parada también sirve a la ruta 101.

Los arcos, representan segmentos de ruta que comunican dos paradas: como peso de los arcos se tiene la distancia entre las dos estaciones.

Por último, el **grafo es dirigido**, dado que las rutas tienen una dirección específica entre las estaciones.

[1] Kaggle Singapore Bus Data, URL: https://www.kaggle.com/gowthamvarma/singapore-bus-data-land-transport-authority

PASO 4: Ejecutar el ejemplo y explorar el ejemplo

El proyecto **SampleAlgorithm** busca familiarizarlos con el TAD Grafos (graph) y su uso para solucionar problemas y una forma de probar su desempeño en aplicaciones MVC.

Diríjase al archivo **view.py** y ejecútelo, y seleccione secuencialmente la **opción 1** y **2** para iniciar el analizador y cargar información respectivamente.

```
****************************************
Bienvenido
1- Inicializar Analizador
2- Cargar información de buses de singapur
3- Calcular componentes conectados
4- Establecer estación base:
5- Establecer metodo de busqueda y estación base:
6- Hay camino entre estacion base y estación:
7- Ruta de costo mínimo desde la estación base y estación:
8- Estación que sirve a mas rutas:
9- Existe un camino de busqueda entre base y estación:
10- Ruta de busqueda entre la estación base y estación:
0- Salir
****************************************
Seleccione una opción para continuar
>
```

```
****************************************
Bienvenido
1- Inicializar Analizador
2- Cargar información de buses de singapur
3- Calcular componentes conectados
4- Establecer estación base:
5- Establecer metodo de busqueda y estación base:
6- Hay camino entre estacion base y estación:
7- Ruta de costo mínimo desde la estación base y estación:
8- Estación que sirve a mas rutas:
9- Existe un camino de busqueda entre base y estación:
10- Ruta de busqueda entre la estación base y estación:
0- Salir
****************************************
Seleccione una opción para continuar
>2

Cargando información de transporte de singapur ....
Numero de vertices: 13535
Numero de arcos: 32270
El limite de recursion actual: 1048576
```

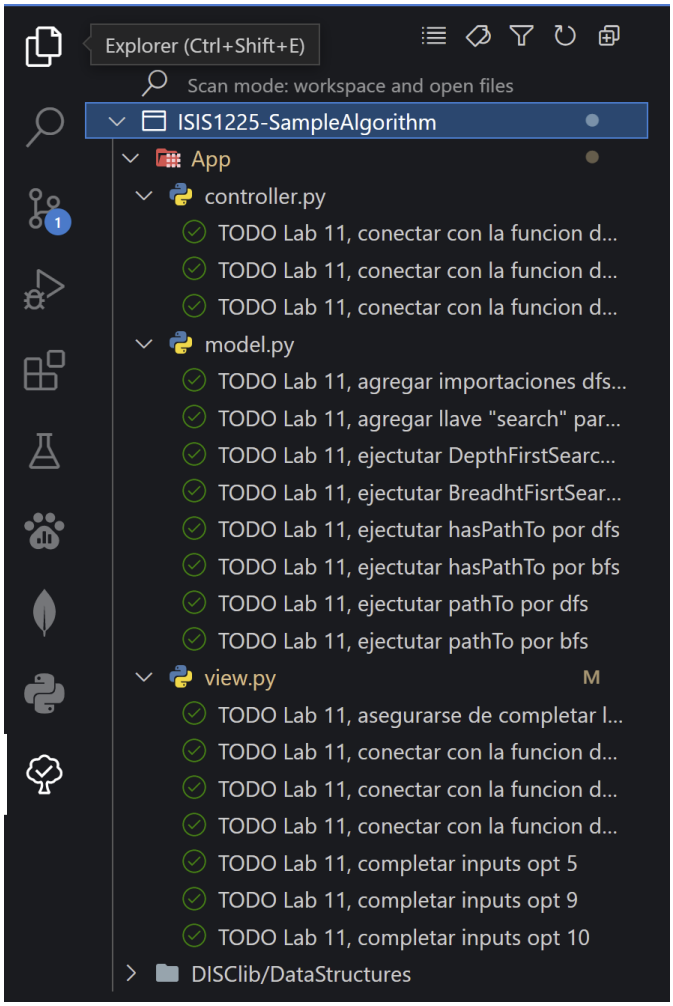
Al cargar la información, verá el número de vértices y arcos cargados en el grafo; así como el número de llamados recursivos de Python.

Note que a diferencia de **SampleGraph** el menú de **SampleAlgorithm** tiene diferencias en sus opciones. En específico las **opciones 5, 9 y 10 son nuevas.** Sin embargo, estas opciones son simplemente un esqueleto y su implementación estará a cargo de los estudiantes para familiarizarse con el uso de los algoritmos del **ADT Graph.**

PASO 5: Implementar cambios en el proyecto

Lo primero que deben buscar son las modificaciones necesarias para implementa las **opciones 5, 9 y 10** del programa. Para ello usen el Plug-in de **VS Code** [TODO Tree.](#) Al desplegar las anotaciones ***#TODO*** sugeridas para esta práctica, encontrara una lista de cambios como se muestra a continuación.

Todas estas modificaciones tienen como objetivo implementar los algoritmos de BFS y DFS dentro del **SampleAlgorithm** e y permitirle al usuario hacer una búsqueda de caminos con los algoritmos disponibles para el ADT Graph.



Divídanse el trabajo equitativamente y utilizando lo aprendido en las practicas anteriores implementen los cambios sugeridos en los comentarios dentro de los tres archivos del MVC (**model.py**, **view.py** y **controller.py**).

A continuación, encontrará un resumen de los cambios sugeridos que debe encontrar en el código.

En el modelo (**model.py**) encontrará ocho (8) cambios sugeridos para implementar, estos van desde agregar las importaciones necesarias desde DISCLib, agregar nuevas llaves al diccionario del analyzer y la implementación de la lógica para utiliza DFS y BDS.

En el código fuente algunos ejemplos de las marcas **#TODO** para los cambios sugeridos se ven de la siguiente manera:

```
import config
from DISClib.ADT import graph as gr
from DISClib.ADT import map as m
from DISClib.ADT import list as lt
from DISClib.Algorithms.Graphs import scc
from DISClib.Algorithms.Graphs import dijsktra as djk
# TODO Lab 11, agregar importaciones dfs y bfs
from DISClib.Utils import error as error
assert config
```

```
def searchPaths(analyzer, initialStation, method):
    """
    searchPaths Calcula los caminos posibles desde una estacion de origen
    y puede utilizar los algoritmos "dfs" o "bfs"

    Args:
        analyzer (dict): diccionario con las estructuras de datos del modelo
        originStation (vertice): estacion de origen del recorrido
        method (str, optional): algoritmo de busqueda. Por defecto es "dfs"

    Returns:
        dict: devuelve el analyzer del modelo
    """
    # TODO Lab 11, ejectutar DepthFirstSearch de dfs
    if method == "dfs":
        pass
    # TODO Lab 11, ejectutar BreadhtFisrtSearch de bfs
    elif method == "bfs":
        pass
    return analyzer
```

En la vista (**view.py**) encontrará 7 cambios que constituyen la implementación de las opciones del menú con manejo de entradas (Inputs) por medio de la consola de usuario.

En el código fuente algunos ejemplos de las marcas #TODO para los cambios sugeridos se ven de la siguiente manera:

```
def optionFive(cont, initialStation, searchMethod):
    # TODO Lab 11, conectar con la funcion del controller searchPaths
    pass
```

```
        elif int(inputs) == 5:
            # TODO Lab 11, completar inputs opt 5
            searchMethod = input("Seleccione 'dfs' o 'bfs' como algoritmo: ")
            msg = "Estación Base: BusStopCode-ServiceNo (Ej: 75009-10): "
            initialStation = input(msg)
            pass
```

En el controlador (**controller.py**) encontrará tres cambios, que cumplen el objetivo de conectar las funciones creadas en el modelo y conectarlas con las opciones invocadas en la vista.

En el código fuente algunos ejemplos de las marcas #TODO para los cambios sugeridos se ven de la siguiente manera:

```python
def searchPaths(analyzer, initialStation, searchMethod):
    """
    Calcula todos los recorridos por "dfs" o "bfs" de initialStation a
    todas las otras estaciones del sistemas
    """
    # TODO Lab 11, conectar con la funcion del model searchPaths
    return None
```

Al finalizar los cambios sobre el MVC podrá ejecutar las opciones 4 con un algoritmo de búsqueda (**'bfs'** o **'dfs'**) a una estación de origen (ej.: **'75009-10'**).

```
*****************************************
Bienvenido
1- Inicializar Analizador
2- Cargar información de buses de singapur
3- Calcular componentes conectados
4- Establecer estación base:
5- Establecer metodo de busqueda y estación base:
6- Hay camino entre estacion base y estación:
7- Ruta de costo mínimo desde la estación base y estación:
8- Estación que sirve a mas rutas:
9- Existe un camino de busqueda entre base y estación:
10- Ruta de busqueda entre la estación base y estación:
0- Salir
*****************************************
Seleccione una opción para continuar
>5
Seleccione 'dfs' o 'bfs' como algoritmo: bfs
Estación Base: BusStopCode-ServiceNo (Ej: 75009-10): 75009-10
Calculando caminos de busqueda con bfs
FIN!
```

Luego podrá comprobar si la ruta existe a una estación de destino (ej,: **'15151-10'**) utilizando la opción 9 y detallando los vértices con la opción 10 como se muestran a continuación.

```
*****************************************
Bienvenido
1- Inicializar Analizador
2- Cargar información de buses de singapur
3- Calcular componentes conectados
4- Establecer estación base:
5- Establecer metodo de busqueda y estación base:
6- Hay camino entre estacion base y estación:
7- Ruta de costo mínimo desde la estación base y estación:
8- Estación que sirve a mas rutas:
9- Existe un camino de busqueda entre base y estación:
10- Ruta de busqueda entre la estación base y estación:
0- Salir
*****************************************
Seleccione una opción para continuar
>9
Estación destino (Ej: 15151-10): 15151-10
Hay camino de busqueda entre la estación base : y la estación: 15151-10:
True
```

```
****************************************
Bienvenido
1- Inicializar Analizador
2- Cargar información de buses de singapur
3- Calcular componentes conectados
4- Establecer estación base:
5- Establecer metodo de busqueda y estación base:
6- Hay camino entre estacion base y estación:
7- Ruta de costo mínimo desde la estación base y estación:
8- Estación que sirve a mas rutas:
9- Existe un camino de busqueda entre base y estación:
10- Ruta de busqueda entre la estación base y estación:
0- Salir
****************************************
Seleccione una opción para continuar
>10
Estación destino (Ej: 15151-10): 15151-10
El camino de busqueda es de longitud: 36
75009-10
76059-10
76059-20
76069-20
76069-168
75059-168
75069-168
75349-168
84209-168
84529-168
84529-228
84219-228
84351-228
84009-228
84009-229
84009-26
84009-30
84029-30
84019-30
84019-30e
82069-30e
81199-30e
81189-30e
81179-30e
91099-30e
91089-30e
14141-30e
14121-30e
15141-30e
15151-30e
15151-30
15151-188
15151-176
15151-175
15151-143
15151-10
```

NOTA: si encuentra dificultad implementando estas modificaciones les sugerimos leer el apéndice del documento que contiene la solución sugerida a la práctica.

Al finalizar las modificaciones y hacer pruebas funcionales ejecutando ambos algoritmos de BFS y DFS considere las siguientes preguntas:

a) ¿Existe alguna diferencia entre los resultados encontrados por BFS y DFS?
b) ¿Por qué existen diferencias entre los dos algoritmos?

PASO 6: Actualizar el repositorio del laboratorio

Para el repositorio del laboratorio inicie los cambios solicitados en los TODOs y confirme los cambios con los comandos **Commit** y **Push** en la rama **main** local y de GitHub con el comentario *"laboratorio 11 – inicio de modificaciones"* antes de la fecha límite de entrega.

PASO 7: Copiar la plantilla del Reto No. 4 en su organización

Copie/Haga **Fork** del repositorio del reto en su organización con el procedimiento aprendido previamente. El repositorio del proyecto base que utiliza este módulo es:

- https://github.com/ISIS1225DEVS/Reto4-Template.git

Antes de clonar el repositorio diríjase a su organización (Ej.: ***EDA2022-1-SEC02-G01*** para el primer grupo de la sección 2 del curso) y cambie el nombre del repositorio según el esquema **Reto4-G<<XX>>** donde **XX** es el número del grupo de trabajo. (Ej.: **Reto4-G01** para el **grupo 1** de la **sección 2).**

Recuerde que el repositorio **debe ser privado** y **NO necesita** agregar la sección o el semestre en este nombre.

PASO 8: Descargue el Reto No. 4

Después de renombrar el proyecto dentro de su organización ya puede clonar el proyecto. Descargue el código en su máquina local siguiendo lo aprendido en las practicas anteriores.

Recuerde modificar el archivo **README** del repositorio para incluir los nombres de los integrantes del grupo e identificar claramente el miembro quien implementara cada requerimiento individual. Un ejemplo es:

- *Req. 3 - Santiago Arteaga, 200411086, sa-artea@uniandes.edu.co*
- *Req. 4 - Carlos Lozano, 200211089, calozanog@uniandes.edu.co*
- *Req 5 – Fernando De la Rosa, 200015053, fde@uniandes.edu.co*

PASO 9: Crear el menú del Reto No. 4

Tomando inspiración del código estudiado en el ejemplo, implemente en el **view.py** del reto el menú de opciones para la carga de archivos y los cinco requerimientos correspondientes.

PASO 10: Analizar Datos del Reto

Descargue los datos oficiales del reto de la sección unificada de la clase de la carpeta de contenido **RETOS/Reto 4/Datos** el grupo de archivos ZIP de la página contienen los CSV necesarios para el desarrollo del reto.

Por último, examine los datos provistos para el reto y para cada uno de los requerimientos responder las siguientes preguntas de análisis y observación:

a) ¿Cuántos grafos se necesitan definir para solucionar los requerimientos del reto? y ¿Por qué?
b) ¿Cuáles son las características específicas de cada uno de los grafos definidos? (vértices, arcos, denso o disperso, dirigido o no dirigido).
c) Además de los grafos, ¿Qué otras estructuras de datos adicionales se necesitan para resolver los requerimientos? Y ¿Por qué?

PASO 11: Actualizar los repositorios

Para el repositorio del laboratorio confirme los cambios con los comandos **`Commit`** y **`Push`** en la rama **main** local y de GitHub con el comentario *"laboratorio 11 - Entrega final"* antes de la fecha límite de entrega.

Para el repositorio del Reto confirme los cambios con los comandos **`Commit`** y **`Push`** en la rama **main** local y de GitHub con el comentario *"Primer avance – Reto 4"* antes de la fecha límite de entrega.

PASO 12: Revisar entregables de la practica

Finalmente, para realizar la entrega del laboratorio revise que sus entregables de la practica estén completos. Para ello, siga las siguientes indicaciones:

1) Acceso al profesor de laboratorio y los monitores de su sección a la organización del grupo.
2) **README** de los repositorios con los datos completos de los integrantes del grupo (nombre completo, correo Uniandes y código de estudiante).
3) Enlace al repositorio GitHub **LabGraph-S<<XX>>-G<<YY>>** con rama ***Main*** actualizada con el comentario *"Laboratorio 11 – Entrega final"* antes del límite de entrega.
4) Incluir en repositorio del laboratorio en la carpeta **Docs** el documento **observaciones-lab 11.pdf** con las respuestas a las preguntas de observación.
    a) ¿Existe alguna diferencia entre los resultados encontrados por BFS y DFS?
    b) ¿Por qué existen diferencias entre los dos algoritmos?
    i) ¿Cuántos grafos se necesitan definir para solucionar los requerimientos del reto? y ¿Por qué?
    ii) ¿Cuáles son las características específicas de cada uno de los grafos definidos? (vértices, arcos, denso o disperso, dirigido o no dirigido).
    iii) Además de los grafos, ¿Qué otras estructuras de datos adicionales se necesitan para resolver los requerimientos? Y ¿Por qué?
5) Enlace al repositorio GitHub Reto4-G<<XX>> con rama **`main`** actualizada con el comentario "Primera entrega – Reto 4" antes de la fecha límite de entrega.

PASO 13: Compartir resultados con los evaluadores

Envíe los **enlaces (URL)** de los dos repositorios por **BrightSpace** antes de la fecha límite de entrega.

Recuerden que cualquier documento solicitado durante en la práctica debe incluirse dentro del repositorio GIT y que solo se calificaran los entregables hasta el último ***COMMIT*** realizado antes de la media noche **(11:59 PM) del 22 de noviembre de 2022.**