Informe TP-3

Alumnos: Cisneros Franco

DNI: 20-41937955-8

Introducción

En el Trabajo Practico número 3 de Programación 3, se pide Implementar una aplicación visual, que dado un grafo simple con pesos en sus vértices se diseñe é implemente un algoritmo Heurístico Goloso para conseguir la Clique de Peso Máximo en el Grafo.

Para obtener el Grafo se puede optar por tomar datos; vértices, Peso de vértice y aristas para crear el grafo, ó tomar el grafo ya existente desde un archivo de extensión ".txt"

Desarrollo

Yo é elegido trabajar con un grafo con estructura de listas, donde cada vértice v de V(G), guarda una lista de sus vecinos. para obtener dicho grafo opte por leerlo desde un archivo de extensión ".txt" haciendo uso de la librería Json.

A continuación describiré cada una de las clases que considere necesarias para el desarrollo.

Las Clases de Interfaz Visual;

Main

Es la interfaz visual con la que primero se encuentra el usuario al ejecutar la aplicación, tiene la tarea de tomar la dirección (path) del archivo de donde se debe leer el grafo, realizar el pedido para obtener la Clique de Peso Máximo a orden del usuario, y luego presentarle al usuario el resultado de la búsqueda.

Quiero aclarar que antes de realizar los cálculos de la búsqueda, se realiza una verificación de que el archivo.txt exista en la dirección que se dio.

Resultado

Tiene como única tarea presentar los datos finales de los cálculos, a orden de la interfaz "Main", en el constructor toma 3 parámetros (Strings).

El primer parámetro es el conjunto de vértices del grafo "V", que solo se muestra el número y el Peso. El segundo parámetro es el conjunto de Aristas del Grafo "E".

Nota: Elegí mostrar el conjunto de Aristas del Grafo y no el conjunto de vecinos de cada vértice, porque considere que sería más fácil de entender para el usuario y además por la Invariante de Representación de la Aplicación. El usuario no necesita saber cómo está diseñado el Software, solo necesita poder comprender el resultado de la búsqueda.

Clases del Código de Negocio;

Vértice

Tiene como atributos:

- Número (su numeración en el Grafo)
- Peso
- Vecinos (una lista con el número de cada uno de sus vecinos)

Operaciones:

- Un constructor que toma como parámetros número y peso
- **argregarVecino:** toma como parámetros un Vértice, y agrega a su lista de vecinos solo el numero de dicho vértice (no el objeto)
- **eliminarVecino:** toma un vértice y lo quita de sus lista de vecinos
- **setPeso:** toma un valor doublé.

Nota: Decidí agregar esta última operación porque cuando se crea el Grafo toma como parámetro la cantidad de vértices y los crea en una numeración 0 a n-1, y en el grafo se los crea por defecto con un peso igual a Cero.

Grafo

Atributos

- **Vértices**: una lista de Objetos tipo vértice.

Operaciones

- **Constructor:** toma como parámetro la cantidad de vértices a crear y los crea automáticamente con una números entre 0 y n-1, con un Peso por defecto de Cero.
- **setearPesoVertice:** toma como parámetro el número del vértice y el valor a setearlo.
- **agregarArista:**
- **eliminarArista:**
- **getVecinos:** toma un vertice y devuelve una lista de Vertices que son sus Vecinos.
- **getAristas:** retorna un String con las aristas del Grafo sin repetir.
- **mostrarVertices:** retorna un String con el numero y peso de cada vertice.

MaxClique:

Cuenta con un conjunto de Operaciones donde solo una es publica;

- **Solver:** toma como parámetro un Grafo, y retorna una lista de vértices que forman la Clique de Peso Máximo en el grafo.

Esta clase implementa Algoritmo Heurístico Goloso, para conseguir la clique de peso máximo.

Para ello primero busca en el grafo el vertice de peso máximo y lo agrega a la lista de vértices que forman la clique de peso máximo, luego entre sus vecinos busca vecinos en común entre este vértice y todos los demás y los agrega a la lista.

Asegurándose de no repetir vertice, esto continua hasta que ya no haya vecinos en comun entre el primer vertice y todos sus vecinos.

Auxiliares:

Esta clase esta diseñada para contener los métodos auxiliares que nesesiten las clases interfaz. Solo contiene un método:

- **existeGrafo:** toma como parámetro un String, que debe ser la direccion del archivo ".txt" donde se debe buscar grafo. Tiene como tarea indicar si existe ó no el archivo indicado, retorna un booleano.

a partir de este método se puede advertir al usuario si el archivo indicado existe ó no.

GrafoJson:

Atributos

- **grafo:** es un único grafo.

Operaciones

- **contructor:** no toma parametros.
- **setGrafo:** toma como parámetro un Grafo y lo setea en su variable.
- **getGrafo:** retorna una copia de su Grafo.
- **generarJson:** que retorna un archivo (string) Json a partir de los atributos del mismo GrafoJson.
- **generarJsonPretty:** retorna un archivo Json versión Pretty
- **guardarGson:** toma como parámetro un String que es el archivo a guardar en disco y la direccion del Archivo de extencion ".txt" donde se guardara.
  si el archi.txt no existe en la direccion dada se lo crea automáticamente.
- Toma como parámetro la direccion de una archivo.txt donde se leera buscara el grafo, y retorna un objeto tipo GrafoJson.

Conclusión:

Entonces según lo pedido se implemento una aplicación que toma la direccion de una archivo de extencion ".txt" que se supone que debe contener un Grafo, y si este archivo existe (ó es encontrado) muestra por interfaz visual la describcion de la clique de peso máximo encontrada. De lo contrario informa que el archivo indicado no fue encontrado en la dirección indicada.

Esta aplicación implementa algoritmo Heurístico Goloso para conseguir la clique de peso máximo.

Nota

ya guarde un archivo.txt con un grafo, pero si quiere guardar otro recuerde que tiene que hacer uso de la clase GrafoJson.

Además como ya dije el Construtor del Grafo toma la cantidad de vértices a contener y los crea automáticamente con un peso igual a cero. Por lo que los pesos de cada vertice pueden ser cambiados.

Eleji esta forma para que los vértices puedan tener una numeración continua, como lo hicimos todo este tiempo.