El Poder de BFS y DFS en los Grafos

Christian Robles
6 de noviembre de 2022

Descripción

Qué hace la aplicación

Esta aplicación fue creada como parte del curso graduado Análisis de Algoritmos (COMP6785) de la Universidad de Puerto Rico en Mayagüez, con el Prof. Alcibiades Bustillo. La aplicación no tiene una interfaz gráfica, sino que es una aplicación de terminal o consola. Dicha aplicación le permite al usuario conseguir el orden topológico de un grafo, determinar si un grafo tiene ciclos, o calcular cuantos componentes conexos tiene un grafo.

El grafo debe ser provisto por el usuario en un archivo de texto sin formato (TXT) nombrado de la siguiente manera: <*nombre*>DG.txt si es un grafo dirigido o <*nombre*>G.txt si es un grafo no dirigido, donde <*nombre*> puede ser cualquier concatenación de caracteres. El archivo en su interior debe seguir la siguiente estructura: la primera línea contiene un entero único *n* que indica la cantidad de vértices, la segunda línea contiene un entero único *e* que indica cuantas aristas hay en el grafo, y las líneas restantes contienen dos enteros separados por espacios en blanco, digamos *u* y *v*, los cuales representan la arista que los conecta (*u*, *v*). El archivo debe estar ubicado dentro de la carpeta nombrada como data, ya que la aplicación busca archivo dentro de esta carpeta.

El nombre del archivo lo pasa el usuario a la aplicación mediante la consola. Al momento de ingresar el nombre, la extensión del archivo (.txt) puede ser obviada porque la aplicación la añade implícitamente. Una vez se pasa el nombre del archivo, la aplicación procede a buscar el archivo en la carpeta de data. En el caso de que no se encuentre el archivo, la aplicación se lo indicará al usuario y le permitirá entrar nuevamente en nombre del archivo. En este paso del algoritmo hay un ciclo que se ejecutara infinitamente. Se puede salir del ciclo de una de dos formas: encontrando un archivo con el nombre provisto por el usuario o escribiendo la palabra “salir” lo que ocasionará la terminación completa de la ejecución de la aplicación. Una vez se provea un nombre que coincida con el de un archivo de la carpeta de data, la aplicación continua su ejecución determinando si el grafo va a ser dirigido o no, y creando el grafo eventualmente.

Lo primero que se hace en esta parte es leer las dos primeras líneas del archivo, que contienen cantidad de vértices y de aristas del grafo, respectivamente. Luego crea una instancia de un grafo, utilizando la clase *Graph*, y se van leyendo una a una las líneas

restantes del archivo, las cuales nos indican cuáles son las aristas del grafo. Las líneas restantes del archivo contienen dos valores *u* y *v* que corresponden a los vértices conectados por la arista (*u*, *v*). Entonces, a medida que se van leyendo las líneas, se van añadiendo los vértices *u* y *v* al grafo, seguido por la adición de la arista (*u*, *v*) que los conecta. Una vez se termina de leer las líneas del archivo, el archivo se cierra y se continúa trabajando con el grafo. En este punto ya el grafo está completado. Después lo que sigue es presentar al usuario las tres distintas operaciones que puede realizar con el grafo.

La primera de ellas es obtener el orden topológico del grafo. Para esto se necesita que el grafo sea dirigido y que no tenga ciclos. Por tanto, si el grafo no es dirigido, la aplicación imprime un mensaje indicando lo antes mencionado. Por otro lado, si el grafo es dirigido, entonces se le aplica "Depth First Search" (DFS) al grafo mediante una función que devuelve si el grafo tiene ciclos o no, y el orden topológico en una lista enlazada (instancia de la clase *LinkedList*). Si el grafo tiene ciclos, la aplicación imprime un mensaje en la consola indicando, si no tiene ciclos, entonces se imprime el orden topológico obtenido. La segunda operación que se puede realizar con el grafo es determinar si este tiene ciclos. En este caso nuevamente se le aplica DFS al grafo, que como se mencionó previamente devuelve si el grafo tiene ciclos y el orden topológico (que no es de interés en este caso), y basado en esto se devuelve un mensaje indicando si el grafo tiene o no ciclos. Por último, la tercera operación que se puede realizar con el grafo es contar sus componentes conexos. Para esto se ejecuta un método llamado *ConnectedComponents* el cual utilizando "Breath First Search" determina cuantos componentes conexos hay en el grafo y devuelve dicho valor. Luego la aplicación imprime un mensaje en la consola indicando la cantidad de componentes conexos encontrados. Estas operaciones se presentan al usuario en forma de lista enumerada y el usuario tiene que entrar el número de la opción que desea. Esta parte también es implementada con un ciclo infinito del cual se puede salir si se provee un número pertinente a una de las opciones, o si se escribe la palabra "salir" lo que terminara la ejecución del programa sin haber realizado ninguna operación. Una vez se entra un número permitido, se realiza la operación pertinente a la opción identificada por el número ingresado, se presenta el resultado, y se termina la ejecución del programa.

Tecnologías, y por qué se utilizaron

Para esta aplicación se utilizó el lenguaje de programación Python por su baja complejidad al momento de programar, la alta capacidad de entendimiento al momento de leer el código, y la alta comunidad que lo respalda. Adicional a esto se creó una clase para grafos, una clase para listas enlazadas, un método para aplicar DFS, y un método para contar los componentes conexos de un grafo utilizando BFS. A continuación, se explica el porqué se escogieron y/o como funcionan.

La clase de grafos, *Graph*, se utilizó para implementar una estructura de datos personalizada en la cual se puedan representar y manipular los grafos de una forma conveniente. Esta clase te permite crear un grafo vacío, sin vértices ni aristas, pasándole al constructor solamente una variable booleana que indica si el grafo va a ser dirigido o no. Esta clase también contiene las listas de adyacencia (en forma de listas enlazadas) respectivas a sus vértices. Con el grafo una vez creado se pueden añadir vértices individualmente, pasando el valor del vértice que se desea añadir a una función llamada *add_vertex* que se encarga de crear una instancia de la clase *Vertex* pasando el valor dado al constructor de dicha clase. Esta función también crea una lista de adyacencia vacía respectiva al valor dado y consecuentemente al vértice previamente creado. También está la función *add_edge* que recibe dos valores que representan una arista, el primero siendo el valor del vértice de donde sale la arista y el segundo siendo el valor del vértice donde incide la arista. Esta función se encarga añadir la arista al grafo y los vértices en las listas de adyacencia correspondiente. Esta clase posee muchas otras funciones, de las cuales muchas no se utilizan en la implementación de este proyecto, pero fueron creadas con el propósito hacer la clase lo más completa posible. Otros métodos de esta clase se presentarán más adelante cuando se hable de otras implementaciones que los utilizan.

La clase de listas enlazadas, *LinkedList*, se implementó más como un reto para el creador de la aplicación porque uno de sus temas favoritos del campo de las ciencias en computadoras son las estructuras de datos. Aparte de esto, por la estructura secuencial que tiene las listas enlazadas, estas son conveniente para representar listas de adyacencia en un grafo y el orden topológico de un grafo, si son utilizadas correctamente. De esta clase solo dos funciones que se utiliza en términos de la implementación de la aplicación pertinente a este proyecto, las demás fueron añadidas para tener una estructura de datos lo más completa posible. El primer método es *add_element* que recibe el elemento que se desea añadir y crea un nodo (instancia de la clase *_Node*) que va a ser constituido por el elemento y una referencia al nodo del próximo valor que se encuentra en la lista, en este caso ninguno, ya que el elemento se va a ubicar al final de la lista. La otra función es utilizada se llama *add_at*, y es utilizada cuando se esta buscando el orden topológico del grafo. Esta función recibe un elemento y una posición, creando así un nodo para el elemento dado y ubicándolo en la posición dada.

El método *DFS*, realiza una búsqueda primero por profundidad, y está a cargo de determinar si existen ciclos o no en el grafo, y el orden topológico correspondiente. Este método recibe un grafo G (instancia de *Graph*) y utiliza la función *V* de *Graph* que devuelve una lista de los vértices del grafo, para iterar a través de ellos e ir visitándolos. Inicialmente, se le asigna un color (blanco) y un predecesor (*None*) a todos los vértices del grafo, se inicializa una variable time a cero y el orden topológico con una lista enlazada vacía. Luego itera por los vértices del grafo y si el color del vértice actual es blanco, entonces de manera recursiva mediante la función *_DFS_visit* que recibe como parámetros el grafo y un vértice

del grafo. Esta función *_DFS_visit* lo que hace es que actualiza la variable time, le cambia el color de blanco a gris y le asigna un tiempo de descubrimiento al vértice ingresado (*v*) y luego itera por la lista de adyacencia correspondiente al vértice, la cual se obtiene mediante la función *Adj* de *Graph* pasándole *v* como parámetro. Si el vértice actual adyacente (*u*) es de color blanco, entonces se le asigna *v* como predecesor a *u*, y se llama *_DFS_visit* nuevamente, pero esta vez para *u*. Por otro lado, si *u* tiene el color gris asignado y el grafo ingresado es dirigido, o si *u* tiene el color gris asignado y el grafo no es dirigido y *u* no es predecesor de *v*, entonces el grafo tiene un ciclo y se termina la ejecución de la función. Si ninguna de las situaciones antes mencionadas se cumple, entonces se continúa la iteración por la lista de adyacencia correspondiente a *v*. Luego de que se termina la iteración recursiva de los vértices adyacentes de *v*, se le cambia el color de *v* a negro, se le actualiza la variable tiempo nuevamente, se le asigna un tiempo de finalización a *v*, y por último se añade *v* a la lista enlazada del orden topológico en la primera posición. Este proceso se repite recursivamente hasta que no queden vértices de color blanco.

La función para contar componentes conexos, *ConnectedComponents*, recibe un grafo como parámetro de entrada. Esta función inicialmente le asigna el color blanco, una distancia infinita, y un predecesor *None* a todos los vértices del grafo, inicializa la variable que va a llevar cuenta de los componentes conexos del grafo a cero, y finalmente itera por todos los vértices del grafo. Si el vértice actual es de color blanco, entonces se incrementa por una unidad la variable que lleva el conteo, y se le aplica una búsqueda primero por profundidad (*_BFS*). La función *_BFS* recibe como parámetro el grafo y un vértice inicial *s*. Esta función procede a cambiar el color del vértice a gris, actualizar la distancia a cero, y añadir *s* a un arreglo *Q* el cual es manipulado con un "Queue". Luego mientras *Q* no este vacío, se le hace "dequeue" a *Q* utilizando la función *pop* de los arreglos de Python para extraer el primer elemento de la lista, y se itera por los vértices adyacentes al vértice extraído de *Q*. Si decimos que, *u* es el vértice extraído de *Q* y que *v* es el vértice adyacente a *u* de la iteración actual, entonces si el color de *v* es blanco, se le cambia el color de *v* a gris, se le actualiza la distancia a *v* sumándole una unidad a la distancia de *u*, se asigna *u* como el predecesor de *v*, y por último se le hace "enqueue" a *v* en *Q* mediante la función *append* de los arreglos de Python. Por otro lado, si el color de *v* no es blanco, se continúa la iteración a través de los vértices adyacentes a *u*, repitiendo el mismo proceso. Finalmente, luego que la iteración por los vértices adyacentes a *u* termina, se le actualiza el color de *u* a negro.

<u>Desafíos enfrentados</u>

De las cosas que más difícil le resultaron al creador de la aplicación fue entender como presentar un grafo que fuera conveniente para lo que se quería realizar, la lógica y organización que conllevo crear la aplicación, como implementar las clases y métodos utilizados, entre otras cosas.

Instalar y Ejecutar el Proyecto

Para ejecutar el proyecto solo se debe tener una versión estable de Python instalada en el sistema completo o en la ubicación del proyecto. Desde el terminal, dentro de la carpeta que contiene la aplicación, se debe ejecutar el comando de Python seguido del main.py (python3 main.py). Esto sería todo lo pertinente a la ejecución del programa, no es necesario la instalación o ejecución de ninguna dependencia externa.

Como Usar Proyecto

Primero tiene que descargar la carpeta del proyecto si ya no la tiene descargada. Luego, para utilizar el proyecto, debe abrir la aplicación del terminal o consola en su computadora, y abrir la carpeta del proyecto desde ahí. Una vez dentro de la carpeta se va a ejecutar el proyecto como se indica en la sección de Instalar y Ejecutar el Proyecto. Una vez se ejecute el proyecto va a aparecer información en la consola sobre el proyecto que se puede encontrar en la sección de Descripción. La información habla sobre el proyecto en general, el formato del nombre del archivo, la estructura del archivo, entre otros. Seguido de esta información, la aplicación va a pedirle al usuario que ingrese un nombre para el archivo donde se encuentra el grafo que se desea utilizar, sin la extensión del archivo, de la siguiente manera:

Nombre del archivo (sin .txt): grafoDG

Si el grafo se encuentra, por ejemplo, en el archivo “grafoDG.txt”, entonces el usuario de escribir “grafoDG” y presionar la tecla Enter. Note que la “D” en el nombre del archivo se utiliza para identificar el grafo como grafo dirigido. Si el usuario desea que el grafo sea no dirigido, entonces debe cambiar el nombre del archivo a algo como “grafoG.txt”, consecuentemente escribiría “grafoG” en vez de “grafoDG”. Los archivos tienen que ser de tipo texto sin formato y su nombre debe terminar en “DG” o “G” dependiendo del tipo de grafo. Si el usuario no desea seguir con la ejecución del programa, puede ingresar alternativamente “salir” para terminar el programa. Si el usuario ingresa un nombre de un archivo existente, la aplicación va a cargar el grafo, y luego va a presentarla al usuario una serie de opciones que se muestran a continuación.

1. *Conseguir el orden topologico del Grafo.*
2. *Determinar si el Grafo tiene al menos un ciclo.*
3. *Contar los componentes conexos del Grafo.*

Ahora el usuario tiene que elegir de las opciones provistas por la aplicación ingresando el número pertinente a la operación que desea realizar. La aplicación va a pedirle al usuario que entre la opción que desea mostrándole al usuario la siguiente línea:

Indique el numero de la opción pertinente: 2

En este punto el usuario también puede ingresar la palabra “salir” para terminar por completo la ejecución del programa. Por el contrario, si el usuario ingresa un número correspondiente a las opciones antes mencionadas, el programa prosigue a ejecutar la operación deseada y devolver el resultado correspondiente a la operación sobre el grafo ingresado. El resultado de la operación va a estar precedido por “>>” en la consola, como se muestra abajo.

>> *El Grafo contiene al menos un ciclo.*

Luego de haber ejecutado la operación deseada sobre el grafo, y haber devuelto el resultado pertinente, se da por terminado la ejecución del programa.