Universidad Nacional Autónoma de México

Facultad de Ciencias

Análisis de Algoritmos

Práctica 4 - Componentes conexas

Jorge Angel Sánchez Sánchez

Fecha de entrega: 1 de Diciembre de 2024

Pasos para ejecutar el programa

1. Requisitos previos

- Asegurarse de tener instalado **Python 3** en el equipo.
  - Puede verificarse con el comando: `python3 --version`.
- Contar con un editor de texto (como Visual Studio Code) para visualizar o modificar el código.

1. Representación de la Solución

En este programa, implementé un algoritmo para encontrar las **componentes conexas** de un grafo utilizando el enfoque de **búsqueda en profundidad (DFS)**. El grafo se representa mediante un **diccionario de adyacencia**, donde cada vértice tiene asociada una lista de sus vértices adyacentes.

Para organizar el código, decidí utilizar una **clase llamada 'Grafo'**, que agrupa los métodos necesarios para cargar el grafo desde un archivo, realizar la búsqueda en profundidad y encontrar las componentes conexas. De esta manera, la implementación es más modular y fácil de reutilizar.

Además, la clase incluye una función que imprime las componentes conexas de forma legible, tal como se solicita en la práctica, permitiendo que cada componente se muestre como una lista de nodos ordenada.

2. Pasos para Ejecutar el Programa

Para ejecutar el programa, seguí estos pasos:

1. **Preparar el archivo de entrada**: El archivo debe contener los vértices del grafo en la primera línea, separados por comas. Luego, las líneas siguientes deben contener pares de vértices que representan las aristas del grafo, también separados por comas. El formato del archivo es el siguiente:

```
1,2,3,4,5,6,7,8,9,10,11,12,13
1,7
5,3
3,1
10,2
9,2
4,9
6,11
13,12
8,13
8,12
```

2. **Ejecutar el programa**: Después de preparar el archivo `grafo.txt`, se debe ejecutar el programa desde la terminal o un entorno de desarrollo que soporte Python. Para hacerlo, hay que navegar al directorio donde se encuentra el archivo y ejecutar el siguiente comando:

```
python3 practica4.py
```

3. **Verificar la salida**: El programa mostrará en consola las componentes conexas encontradas en el grafo, cada una en una línea separada. Por ejemplo:

```
[1, 3, 5, 7]
[2, 4, 9, 10]
[6, 8, 11, 13, 12]
```

3. Representación de la Salida

La salida del programa consiste en las **componentes conexas** del grafo. Cada componente es un conjunto de vértices que están conectados de alguna forma. Para cada componente, el programa imprime una lista de nodos ordenados de menor a mayor, siguiendo el formato solicitado.

Ejemplo de salida:

```
[1, 3, 5, 7]
[2, 4, 9, 10]
[6, 8, 11, 13, 12]
```

Esto significa que:

- Los vértices 1, 3, 5 y 7 forman una componente conexa.
- Los vértices 2, 4, 9 y 10 forman otra componente conexa.
- Los vértices 6, 8, 11, 13 y 12 forman una tercera componente conexa.

4. Explicación de la Lógica del Programa

Para implementar la solución, utilicé la siguiente estructura:

4.1. Estructura de la clase `Grafo`:

La clase `Grafo` maneja los vértices y las conexiones entre ellos en un **diccionario de adyacencia**, que es una estructura eficiente para representar grafos. Este diccionario permite guardar rápidamente las relaciones entre vértices.

En el método `cargar_desde_archivo`, se lee el archivo de entrada, se extraen los vértices y las conexiones, y se almacenan en el diccionario.

4.2. Algoritmo DFS (Depth-First Search):

La búsqueda en profundidad se realiza mediante el método `dfs_recursivo`. Este método comienza en un vértice y explora todos los vértices adyacentes no visitados de forma recursiva. Cada vez que se visita un nuevo vértice, se marca como visitado y se agrega a la lista de la componente conexa.

4.3. Detección de Componentes Conexas:

El método `obtener_componentes` recorre todos los vértices del grafo. Si un vértice aún no ha sido visitado, ejecuta DFS para encontrar todos los vértices conectados, agrupándolos en componentes conexas.

4.4. Impresión de Resultados:

Finalmente, el método `imprimir_componentes` toma las componentes conexas y las imprime en el formato adecuado, con los vértices ordenados de menor a mayor.