UNIVERSIDAD DE COSTA RICA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERIA ELECTRICA

IE0217 – ESTRUCTURAS DE DATOS Y ALGORITMOS PARA INGENERIA

KOSARAJU-SHARIR'S ALGORITHM

POR:
KEVIN STWARD DELGADO ROJAS – B82566

PROFESOR:
JUAN CARLOS COTO ULATE

CIUDAD UNIVERSITARIA RODRIGO FACIO
II CICLO -NOVIEMBRE 2020

Índice

Introducción

En las décadas de 1950 y 1960, los matemáticos e informáticos comenzaron a estudiar los algoritmos de grafos en un contexto en el que el análisis de los algoritmos en sí estaba en desarrollo como campo de estudio. La amplia variedad de algoritmos de grafos a considerar, -junto con los desarrollos en curso en los sistemas informáticos, los lenguajes y nuestra comprensión de cómo realizar cálculos de manera eficiente-, dejó muchos problemas difíciles sin resolver. A medida que los informáticos comenzaron a comprender muchos de los principios básicos del análisis de algoritmos, comenzaron a comprender qué problemas de grafos podían resolverse de manera eficiente y cuáles no, y luego desarrollar algoritmos cada vez más eficientes para el primer conjunto de problemas. En la década de 1980, R. Kosaraju dio una nueva mirada al problema y desarrolló una nueva solución [1]. Se cuenta que Kosaraju iba a impartir unas clases sobre el algoritmo de Tarjan, pero olvido las notas de clase, y acabo descubriendo este.

El algoritmo de Kosaraju (también conocido como el algoritmo de Kosaraju-Sharir) es un algoritmo de tiempo lineal para encontrar los componentes fuertemente conectados de un grafo dirigido. El mismo algoritmo fue descubierto independientemente por Micha Sharir y publicado por él en 1981. Hace uso del hecho de que el grafo de transposición (el mismo grafo con la dirección de cada borde invertido) tiene exactamente los mismos componentes fuertemente conectados que el gráfico original [1].

Discusión

Funcionamiento del algoritmo Kosaraju-Sharir's

Primeramente, es importante conocer acerca de los componentes fuertemente conectados (SCC) de un grafo dirigido. Ya que el algoritmo Kosajaru-Sharir's se basa en esto.

Definición: Dos vértices v y w están fuertemente conectados si son mutuamente alcanzables: es decir, si hay una ruta dirigida de v a w y una ruta dirigida de w a v. Un dígrafo está fuertemente conectado si todos sus vértices están fuertemente conectados uno al otro [3]. Como se ilustra en la siguiente imagen:

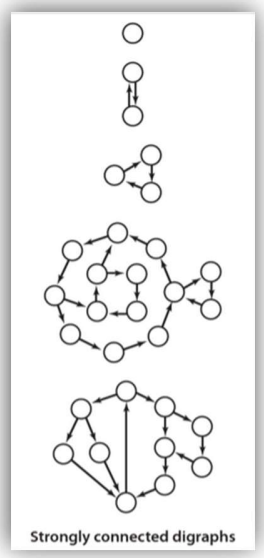

Ilustración 1

El funcionamiento del algoritmo Kosaraju se basa en tres simples pasos como se menciona en [3]:

1. Realice una primera búsqueda en profundidad (DFS) en todo el conjunto de grafos.
2. Invierta el grafo original.
3. Realice una búsqueda en profundidad (DFS) en el grafo invertido.

El algoritmo Kosaraju-Sharir's seria el siguiente:

1. Cree una pila
2. Para cada vértice u del grafo, marque "u" como no visitado.
3. Para cada vértice u del gráfico, haga Visit(u), donde Visit(u) es la subrutina recursiva:

   Si U se ha visitado a continuación:

   - Marcar u como visitado.
   - Para cada vecino externo v de u , visite(v).
   - Poner u a la pila .

   De lo contrario, no haga nada.

4. Para cada elemento u de la pila en orden, haga Assign(u , u) donde Assign(u , root) es la subrutina recursiva:

   Si u no se ha asignado a un componente, entonces:

   - Asigne u como perteneciente al componente cuya raíz es root .
   - Para cada vecino v de u , haga Asignar(v , root).

   De lo contrario, no haga nada.

Por ejemplo, tomemos el siguiente grafo:

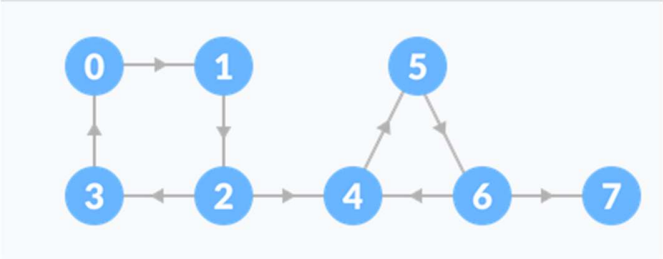

Ilustración 2. Grafo Inicial

Los componentes fuertemente conectados del gráfico anterior son:

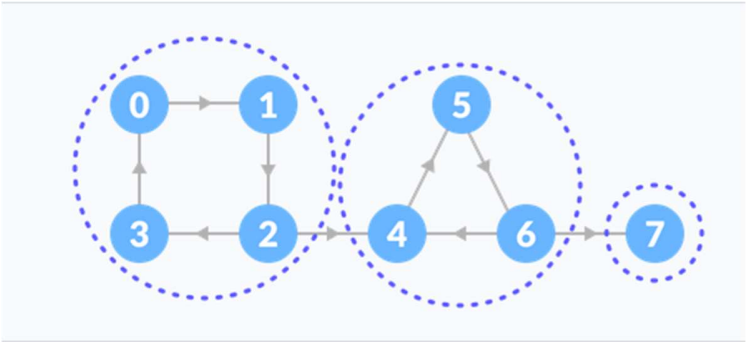

Ilustración 3. Componentes fuertemente conectados

Puede observar que, en el primer componente fuertemente conectado, cada vértice puede alcanzar el otro vértice a través del camino dirigido.

Entonces, comencemos desde el vértice-0, visitemos todos sus vértices secundarios y marquemos los vértices visitados como hechos. Si un vértice conduce a un vértice ya visitado, empuje este vértice a la pila.

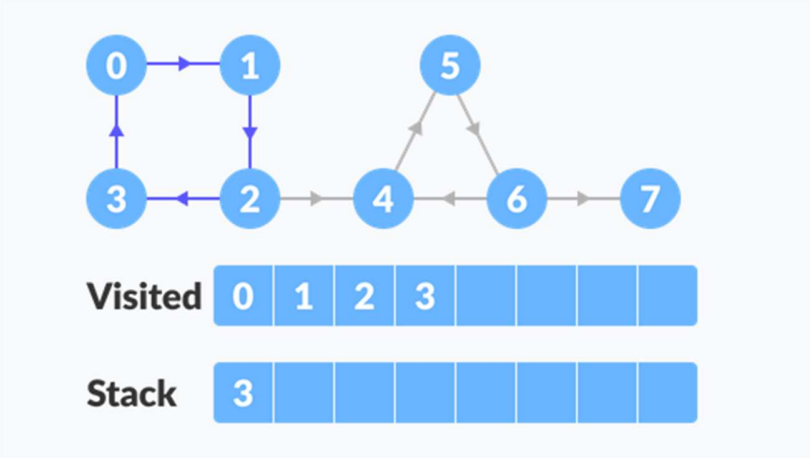

Ilustración 4. DFS en el grafo

Vaya al vértice anterior (vértice-2) y visite sus vértices secundarios, es decir, vértice-4, vértice-5, vértice-6 y vértice-7 secuencialmente. Como no hay ningún lugar adonde ir desde el vértice-7, empújelo en la pila.

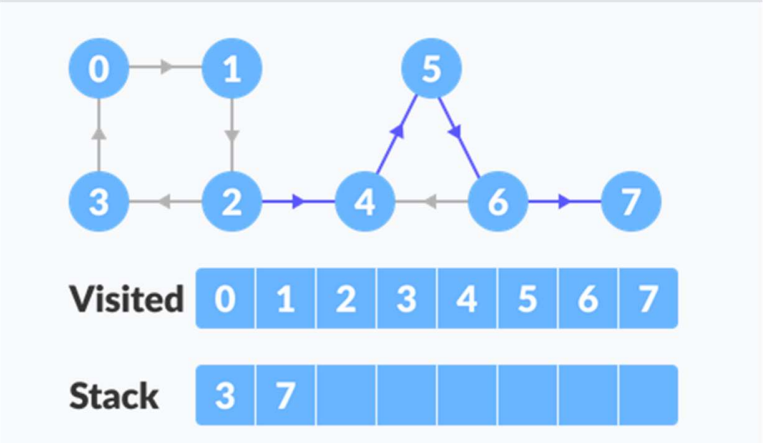

Ilustración 6. DFS en el grafo

Vaya al vértice anterior (vértice-6) y visite sus vértices secundarios. Pero, se visitan todos sus vértices secundarios, así que empújelo en la pila.

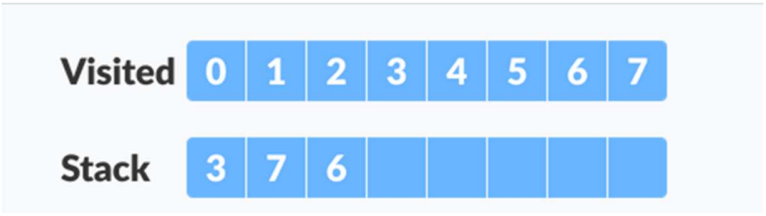

Ilustración 5. Apilado

Del mismo modo, se crea una pila final.

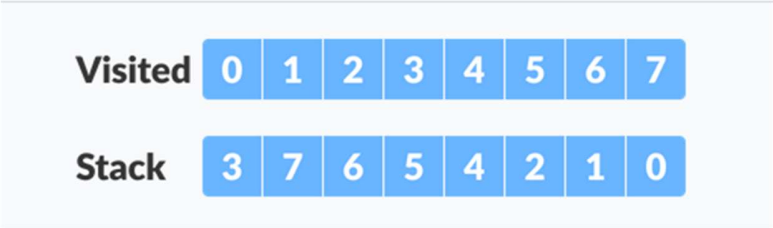

Ilustración 7. Pila final

Ahora, invierta el grafo original.

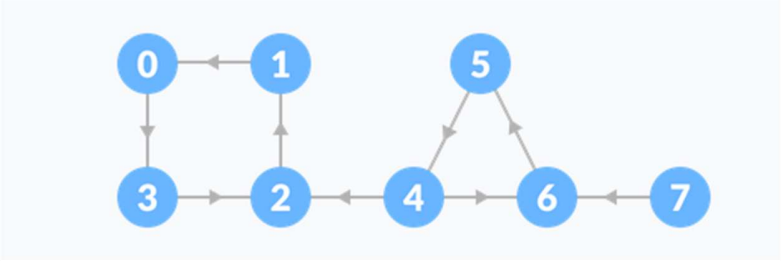

Ilustración 8. DFS grafo invertido

Extraiga el vértice-0 de la pila. Comenzando desde el vértice-0, recorra sus vértices secundarios (vértice-0, vértice-1, vértice-2, vértice-3 en secuencia) y márquelos como visitados. El hijo del vértice-3 ya está visitado, por lo que estos vértices visitados forman un componente fuertemente conectado.

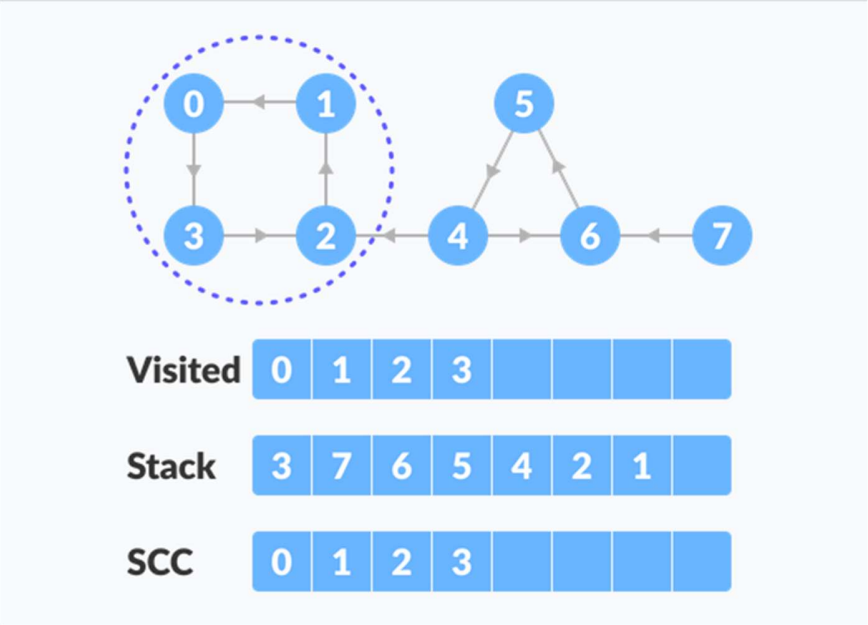

Ilustración 9. Empieza desde arriba y recorre todos los vértices

Ve a la pila y abre el vértice superior si ya lo has visitado. De lo contrario, elija el vértice superior de la pila y atraviese sus vértices secundarios como se muestra arriba.

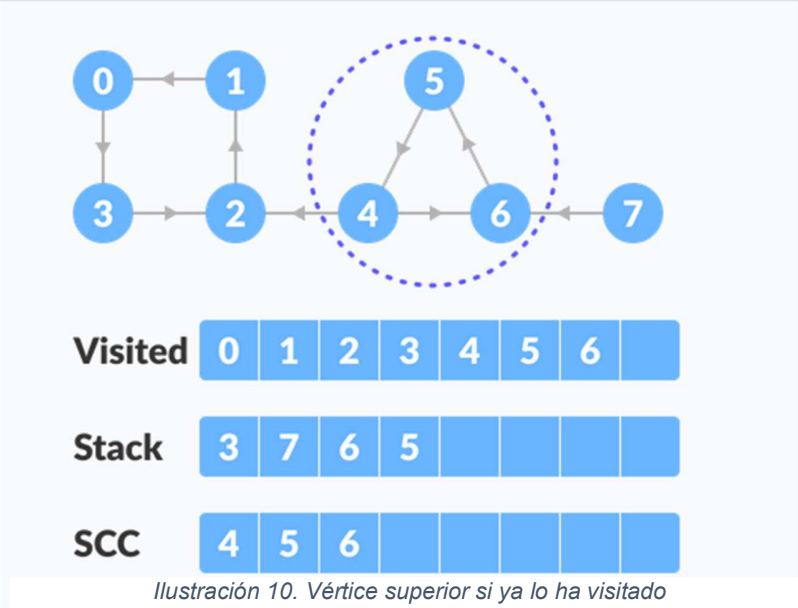

Ilustración 10. Vértice superior si ya lo ha visitado

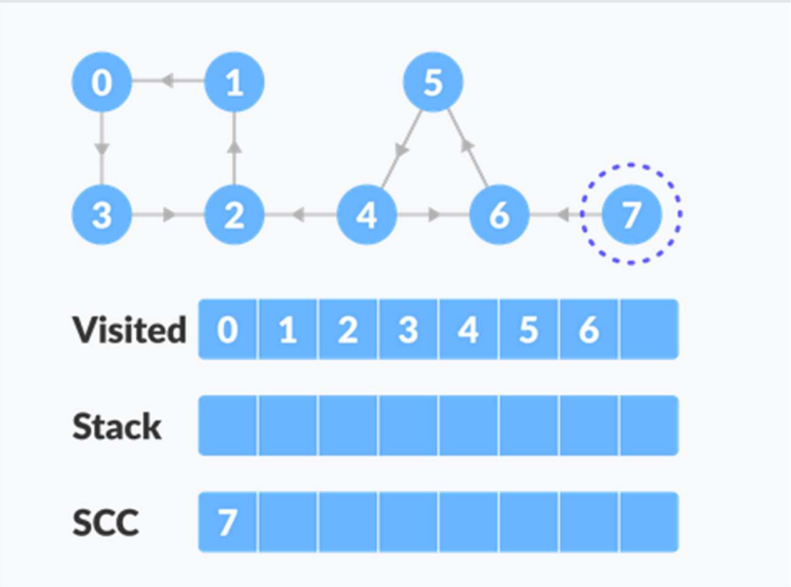

Ilustración 11. Componente fuertemente conectado

Por tanto, los componentes fuertemente conectados son:

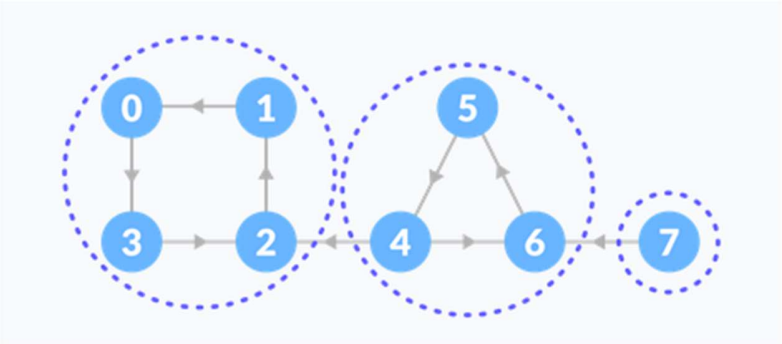

Ilustración 12. Todos los componentes fuertemente conectados

Utilización del algoritmo Kosaraju-Sharir's

La utilización de los algoritmos que se basen en los SSC como el Kosaraju-Sharir's tiene un amplio campo de aplicación como lo menciona [3], por ejemplo, los componentes fuertemente conectados pueden ayudar a los autores de libros de texto a decidir qué temas deben agruparse y a los desarrolladores de software a decidir cómo organizar los módulos del programa.

La siguiente ilustración 13, muestra un modelo de dígrafo de contenido web, donde los vértices representan páginas y los bordes representan hipervínculos de una página a otra. Los componentes fuertemente conectados de un dígrafo de este tipo pueden ayudar a los ingenieros de redes a dividir la gran cantidad de páginas de la web en tamaños más manejables para su procesamiento. También para la creación de mapas, aplicaciones de generador de rutas para vehículos.

application	vertex	edge
web	page	hyperlink
textbook	topic	reference
software	module	call
food web	organism	predator-prey relationship

Typical strong-component applications

Ilustración 13

Complejidad del algoritmo

La complejidad de un algoritmo mide la utilización de los recursos disponibles. Los recursos más importantes con los que cuenta una computadora a la hora de ejecutar un programa son dos [4]:

- Memoria: La máxima cantidad de memoria que usa el programa al mismo tiempo. Si usa varias veces la misma memoria se cuenta una sola vez.
- Tiempo: Cuánto tarda en correr el algoritmo. Se mide en cantidad de operaciones básicas (sumas, restas, asignaciones, etc)

El algoritmo Kosaraju-Sharir's se ejecuta en tiempo lineal, es decir O(V+E) [3], eso debido a que realiza el recorrido del grafo dos veces. Por lo cual tarda un poco mas que otros algoritmos que solo recorren una vez el grafo.

Implementación del algoritmo Kosaraju-Sharir’s

El siguiente algoritmo fue tomado como base de [3], para realizar a cabo los ejemplos.

```python
# El algoritmo de Kosaraju para encontrar componentes fuertemente conectados en Python

from collections import defaultdict

class Graph:

    def __init__(self, vertex):
        self.V = vertex
        self.graph = defaultdict(list)

    # Add edge into the graph
    def add_edge(self, s, d):
        self.graph[s].append(d)

    # dfs (Depth First Search)
    def dfs(self, d, visited_vertex):
        visited_vertex[d] = True
        print(d, end='')
        for i in self.graph[d]:
            if not visited_vertex[i]:
                self.dfs(i, visited_vertex)
```

```python
    # ordenamiento
    def fill_order(self, d, visited_vertex, stack):
        visited_vertex[d] = True
        for i in self.graph[d]:
            if not visited_vertex[i]:
                self.fill_order(i, visited_vertex, stack)
        stack = stack.append(d)

    # transpose the matrix
    def transpose(self):
        g = Graph(self.V)

        for i in self.graph:
            for j in self.graph[i]:
                g.add_edge(j, i)
        return g
```

```
    # Print stongly connected components
    def print_scc(self):
        stack = []
        visited_vertex = [False] * (self.V)

        for i in range(self.V):
            if not visited_vertex[i]:
                self.fill_order(i, visited_vertex, stack)

        gr = self.transpose()

        visited_vertex = [False] * (self.V)

        while stack:
            i = stack.pop()
            if not visited_vertex[i]:
                gr.dfs(i, visited_vertex)
                print("")
```

Los grafos a probar serían los siguientes:

```
g = Graph(8)
g.add_edge(0, 1)
g.add_edge(1, 2)
g.add_edge(2, 3)
g.add_edge(2, 4)
g.add_edge(3, 0)
g.add_edge(4, 5)
g.add_edge(5, 6)
g.add_edge(6, 4)
g.add_edge(6, 7)

print("Strongly Connected Components:")
g.print_scc()
print("\n")

g1 = Graph(5)
g1.add_edge(1, 0)
g1.add_edge(0, 2)
g1.add_edge(2, 1)
g1.add_edge(0, 3)
g1.add_edge(3, 4)

print("Strongly Connected Components:")
g1.print_scc()
```

Conclusiones

- El algoritmo de Kosaraju-Sharir es uno de los cuales conceptualmente es más simple, pero no es tan eficiente. Debido a que es un algoritmo de tiempo lineal que hace un recorrido DFS dos veces en el grafo.

- Es un algoritmo que encuentra aquellos componentes fuertemente conectados en conjuntos de grafo dirigidos.

- Es un algoritmo que se puede implementar en distintas aplicaciones en diferentes áreas que lo requieran. Entre las mas populares son en mapas, generadores de rutas para vehículos, aplicaciones web, software y todo lo que se pueda basar en componentes fuertemente conectados.

Referencias

[1]. R. Sedgewick, K. Wayne. Algorithms. Addison-Wesley Professional, 2011. [E book] Available: Google Books.

[2]. Rosseta Code. “Kosaraju”. 2007. [Online]. Available: https://rosettacode.org/wiki/Kosaraju

[3] Parewa Labs Pvt. Ltd. “Strongly Connected Components”. 2020. [Online]. Available: https://www.programiz.com/dsa/strongly-connected-components

[4] L. Taravilse. “Training Camp 2012”. [Online]. Available: https://tc-arg.tk/pdfs/2012/01-Grafos.pdf