Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

Estructuras Abstractas de Datos y Algoritmos para Ingeniería

Proyecto De Investigación:
“Algoritmo de Tarjan para Identificar los Componentes Fuertemente Conexos”

Profesor: Juan Carlos Coto Ulate

Elaborado por: Erick Muñoz Zamora

Carné: B44810

Introducción: **4**
Grafo: 4
Componentes Fuertemente Conectados: 5
Búsqueda en Profundidad: 5
Algoritmo de Tarjan: 6
Resultados: **6**
Implementación del Algoritmo: 6
Grafo: 6
Nodo: 7
Servidor: 7
Algoritmo de Tarjan: 8
Ejemplo: **10**
Discusión de los resultados: **12**
Conclusiones: **13**
Referencias: **13**

Introducción:

El presente proyecto de investigación busca realizar el estudio de lo que se llama el “Algoritmo de Tarjan”, el cual es usado para el procesamiento de grafos dirigidos, este algoritmo tiene como meta encontrar los componentes del grafo que están fuertemente conectados. Este algoritmo es presentado por Robert Tarjan en un artículo titulado “DEPTH-FIRST SEARCH AND LINEAR GRAPH ALGORITHMS”[1], que fue publicado en 1972, el cual en general se encuentra en un contexto donde se estaba buscando generar los algoritmos para responder preguntas teóricas sobre los grafos.

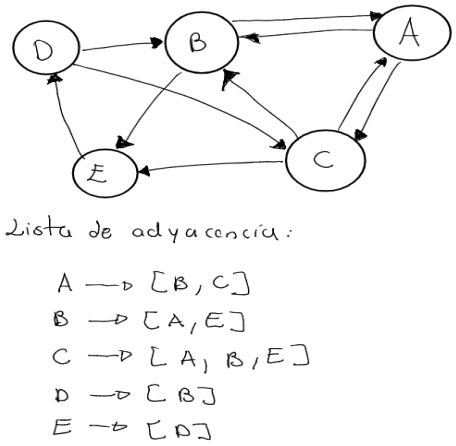
Una de las aplicaciones según Sedgewick[3], de este tipo de identificación de los componentes fuertemente conectados es para el uso en detectar la dependencia de servidores en la web, entonces por ejemplo si se genera un grafo sobre las conexiones que existen entre diferentes servidores, y hay algunos que están fuertemente conectados entre sí, entonces se puede pensar en una mejora del diseño de la arquitectura de redes para que las conexiones entre estos servidores sea más rápida, por ejemplo si estos servidores están separados, al juntarlos se podría disminuir la latencia y hacer que el proceso mejore en su performance.

A continuación se pretende ir dando definiciones básicas para el estudio del algoritmo central de la investigación, para así poder trabajar de una forma más clara cuando ya se vaya a mencionar formalmente dicho algoritmo.

Grafo:

Un grafo es un conjunto de vértices y arcos que forman conexiones entre los vértices, se puede definir un grafo de forma matemática como G = (V, E), donde V son los vértices y E los arcos. Además, cuando se habla de grafos dirigidos, los cuales tienen direcciones en sus arcos, lo cual representa la dirección en la que se conectan los vértices. [2]

Los grafos pueden ser representados por listas de adyacencia, como por ejemplo se tienen las siguientes listas de adyacencias, como se presenta en la siguiente figura:



En python se puede hacer una representación de un grafo a través de esta idea de la lista de adyacencias y de esta forma se pretende crear dicha estructura de datos.

Componentes Fuertemente Conectados:

Según Sedgewick[3], dos vértices **A** y **B**, están fuertemente conectados si **A** tiene un arco en dirección hacia **B** y **B** tiene un arco en dirección hacia **A**. Además se tienen que tener las siguientes consideraciones:

- **A** está fuertemente conectado a **A**, si:
    - **A** está fuertemente conectado a **B** y **B** está fuertemente conectado a **A**.
    - **A** está fuertemente conectado a **B** y **B** está fuertemente conectado a **X**, y además **A** está fuertemente conectado a **X**.

Esto dicho de alguna forma un poco menos formal, se puede decir que **A** está fuertemente conectado a **B,** si existe un camino de **A** a **B** y existe otro camino de **B** a **A**, pasando por N nodos diferentes, incluso si solo es un camino de **B** en dirección a **A**.

Búsqueda en Profundidad:

Es un algoritmo para buscar en un grafo o un árbol. El algoritmo comienza en un nodo y avanza cuanto pueda en una dirección, si no puede avanzar más entonces se devuelve hasta que encuentra un arco que no ha sido explorado, y lo explora. Este algoritmo sigue este procedimiento hasta que ha recorrido todo el grafo.[4]

Algoritmo de Tarjan:

Según Kim[5], el algoritmo de Tarjan use una pasada del algoritmo de Búsqueda en profundidad, el descrito anteriormente, y usa otras estructuras de datos, dos listas para mantener información sobre los nodos visitados y además un valor de “Low link”, que es un valor que utiliza el algoritmo para identificar las conexiones que tienen los nodos, estas listas tienen el tamaño del número de nodos que tiene el grafo.

Según como lo presenta Fiset[6], los pasos que sigue el algoritmo son los siguientes:

1. Marcar todos los nodos del grafo como “No-visitados”.
2. Comenzar una búsqueda de profundidad.
3. Cuando se visita un nodo en la búsqueda, se le asigna un ID y un valor de “Low link”, este valor de “Low link” en este algoritmo va a ser el mismo ID. Además se agregan a un stack de nodos visitados.
4. Cuando termina el proceso de Búsqueda profunda, si el nodo previo se encuentra en el stack de nodos visitados, entonces hay que hacer una función mínima entre el valor de Low Link, del nodo previo y el del nodo actual, el resultado del valor de Low link va a ser el mismo para los dos.
5. Una vez que se han visitado todos los vecinos, si el nodo actual comenzó un conjunto de nodos fuertemente conectados, entonces se tienen que sacar los nodos de la pila hasta que se llegue al nodo actual.

Según el autor del algoritmo Tarjan[1], la complejidad en tiempo del algoritmo es una función O(V, E), donde V son la cantidad de nodos y E la cantidad de arcos, esta complejidad es lineal según él, es decir depende linealmente del tamaño de ambos.

Resultados:

Implementación del Algoritmo:

Se realizó la implementación del algoritmo siguiendo los pasos basados en el algoritmo de Tarjan, presentado por Fiset[6], descritos anteriormente. Para realizar la explicación un poco más adentro se explicarán las estructuras creadas que fueron usadas dentro de dicha implementación.

Grafo:

Esta clase lo que hace es contener la información pertinente para definir un grafo, su constructor que recibe los nodos que van a formar parte del grafo, y además se

le agrega la matriz de adyacencias como parámetro al mismo. Además de 2 métodos privados que son utilizados para hacer algunas operaciones cuando se setean la lista de Nodos y las Adyacencias en forma de matriz. import sys

```
from estructuras.grafo.nodo import Nodo

class Grafo:



   def __init__(self, listaNodos, adyacencias):
       self.nodos = []
       self.matrizAdyacencia = [[]]
       self.__agregarNodos(listaNodos)
         self.matrizAdyacencia = [ [ 0 for i in range(self.tamano) ] for j in
range(self.tamano) ]
       self.matrizAdyacencia = adyacencias

   def __agregarNodos(self, listaNodos):
       for nodo in listaNodos:
           self.nodos.append(nodo)

       self.tamano = len(self.nodos)

   def __agregarAdyacencias(self, nodo, listaNodos):
       for nodoEnlazado in listaNodos:
           self.matrizAdyacencia[nodo.id][nodoEnlazado.id] = 1
```

Nodo:

Este se implementó para darle cierta información importante sobre lo que podría ser un nodo, en este caso se designa un nodo, como un objeto que contiene un id, además contiene un objeto de la clase Servidor, el cual contiene información sobre lo que podría ser el mismo, por ejemplo el nombre del servidor, su ubicación y su sistema operativo.

```
import sys
class Nodo:
   def __init__(self, id, servidor):
       self.id = id
       self.servidor =  servidor
```

Servidor:

Como se dijo anteriormente, este solo se usó de una forma ilustrativa.

```
class Servidor:
   def __init__(self, name, location, os):
```

```
        self.name = name
        self.location = location
        self.os = os
```

Algoritmo de Tarjan:

Finalmente se presenta el código que se hizo para la implementación del algoritmo de Tarjan,

```
from estructuras.grafo.grafo import Grafo
from estructuras.grafo.nodo import Nodo
from estructuras.servidor.servidor import Servidor
class AlgoritmoTarjan:
   def __init__(self, grafo):
       self.__NODESCUBIERTO = -1

       self.__tamano = grafo.tamano
       self.__matrizAdyacencia = grafo.matrizAdyacencia
       self.__idNodo = 0
       self.NComponentesFuertes = 0
       self.__idsAlg = [0]*self.__tamano
       self.__valoresLowLink = [0]*self.__tamano
       self.__enPila = [False]*self.__tamano
       self.__pila = []
       self.resultadoLowLink = self.__componentesFuertementeConexos()

   def __componentesFuertementeConexos(self):

       for i in range(0, self.__tamano):
           self.__idsAlg[i] = self.__NODESCUBIERTO

       for i in range(0, self.__tamano):
           if self.__idsAlg[i] == self.__NODESCUBIERTO:
               self.__busquedaEnProfundidad(i)
       return self.__valoresLowLink

   def __busquedaEnProfundidad(self,nodo):
       self.__pila.append(nodo)
       self.__enPila[nodo] = True
       self.__idsAlg[nodo] = self.__idNodo
       self.__valoresLowLink[nodo] = self.__idNodo
       self.__idNodo += 1
       listaAdyacencia = []

       for i in range(0, self.__tamano):
           if self.__matrizAdyacencia[nodo][i] == 1:
               listaAdyacencia.append(i)

       for nodoAdyacente in listaAdyacencia:

           if self.__idsAlg[nodoAdyacente] == self.__NODESCUBIERTO:
               self.__busquedaEnProfundidad(nodoAdyacente)

           if self.__enPila[nodoAdyacente]:
```

```
            self.__valoresLowLink[nodo]       =       min(self.__valoresLowLink[nodo],
self.__valoresLowLink[nodoAdyacente])
      if self.__idsAlg[nodo] == self.__valoresLowLink[nodo]:
          for i in (0,len(self.__pila)):
              nodoPop = self.__pila.pop()
              self.__enPila[nodoPop] = False
              self.__valoresLowLink[nodoPop] = self.__idsAlg[nodo]
              if nodoPop == nodo:
                  break
          self.NComponentesFuertes += 1
```

Esta clase lo que hace es tomar el grafo que se le de en el constructor, y crear todas las diferentes variables privadas que necesita para desarrollarse, en este caso un flag para definir si un nodo ha sido descubierto o no en el algoritmo de busqueda en profundidad, una lista para almacenar los IDs de los nodos que se van asignando en el curso del algoritmo, inicialmente seteados a -1, ya que no han sido visitados, luego una lista para los valores de Low-link pertenecientes a cada nodo, de igual forma cambian conforme avanza el algoritmo, por ùltimo una lista de booleanos para definir si los nodos han sido visitados y otra lista que funciona como pila que guarda los IDs de los nodos visitados.

Por otro lado tenemos los métodos privados pertenecientes a la clase que son el de ***componentesfuertementeconexos()*** y ***busquedadEnProfundidad()***, el primero lo que hace es llamar a ***búsquedaEnProfundidad()*** por cada nodo nodo no descubierto, que en un inicio son todos. dentro de la ***busquedaEnProfundidad()*** se usa la recursión para ir encontrando caminos entre los nodos, primero se transforma la matriz de adyacencias en listas de adyacencias para el nodo que se le está buscando los vecinos hacia los que tiene arcos dirigidos desde el nodo hasta otro, se comienza a iterar por cada vecino de estos y se revisa si cada uno ya ha sido visitado o no, si no entonces se llama a ***busquedaEnProfundidad()*** para el vecino que se tiene, cuando se llega a algún nodo en el cual todos sus nodos fueron visitados, entonces se revisa si alguno de ellos fue uno de los visitados en esta iteración, si sí entonces se cierra el camino o path y por ende estos tienen una relación fuerte, es decir son parte de un mismo cúmulo de nodos fuertemente conexos y así se les asigna un mismo valor de Low-Link. el algoritmo regresa a ***componentesfuertementeconexos(),*** pero ya un grupo de estos fuertemente

conexos ha sido encontrado, entonces ya al revisar si han sido descubiertos, se va a encontrar otro nodo y se va a intentar hacer el mismo procedimiento dentro de este.

Ejemplo:

```
listaNodos = [Nodo(0, Servidor("server1", "Alajuela", "Ubuntu")),
Nodo(1, Servidor("server2", "SanJosè", "Windows")),
Nodo(2, Servidor("Server3", "Cartago", "Fedora"))]
matriz = [[0, 1, 1],[0, 0, 0], [1, 0 ,0]]
grafo = Grafo(listaNodos, matriz)
print("Grafo 1")
tarjanGrafo1 = AlgoritmoTarjan(grafo)
print("Para    el    grafo    con    la    lista    de    adyacencia    \n"    +
str(grafo.matrizAdyacencia))
print("El número de cúmulos de nodos conexos entre sí es " +
            str(tarjanGrafo1.NComponentesFuertes) + ",\ny lista de valores de
low link es \n"
          + str(tarjanGrafo1.resultadoLowLink))
print("\nGrafo 2")
listaNodos = [Nodo(0, Servidor("server1", "Alajuela", "Ubuntu")),
Nodo(1, Servidor("server2", "SanJosè", "Windows")),
Nodo(2, Servidor("Server3", "Cartago", "Fedora")), Nodo(3, Servidor("server2",
"SanJosè", "Windows")),
Nodo(4, Servidor("Server3", "Cartago", "Fedora"))]
matriz = [[0, 1, 1,0,0],[0, 0, 0,0,0], [1, 0 ,0,1,0],[0, 0 ,0,0,1], [0, 0
,0,1,0]]
grafo2 = Grafo(listaNodos, matriz)
tarjanGrafo2 = AlgoritmoTarjan(grafo2)
print("Para    el    grafo    con    la    lista    de    adyacencia    \n"    +
str(grafo2.matrizAdyacencia))
print("El número de cúmulos de nodos conexos entre sí es " +
            str(tarjanGrafo2.NComponentesFuertes) + ",\ny lista de valores de
low link es "
          + str(tarjanGrafo2.resultadoLowLink))
print("\ngrafo 3")
matriz = [[0, 1, 1,0,0],[1, 0, 0,0,0], [1, 0 ,0,1,0],[0, 0 ,0,0,1], [0, 0
,0,1,0]]
grafo3 = Grafo(listaNodos, matriz)
tarjanGrafo3 = AlgoritmoTarjan(grafo3)
print("Para    el    grafo    con    la    lista    de    adyacencia    \n"    +
str(grafo3.matrizAdyacencia))
print("El número de cúmulos de nodos conexos entre sí es " +
            str(tarjanGrafo3.NComponentesFuertes) + ",\ny lista de valores de
low link es "
          + str(tarjanGrafo3.resultadoLowLink))
```

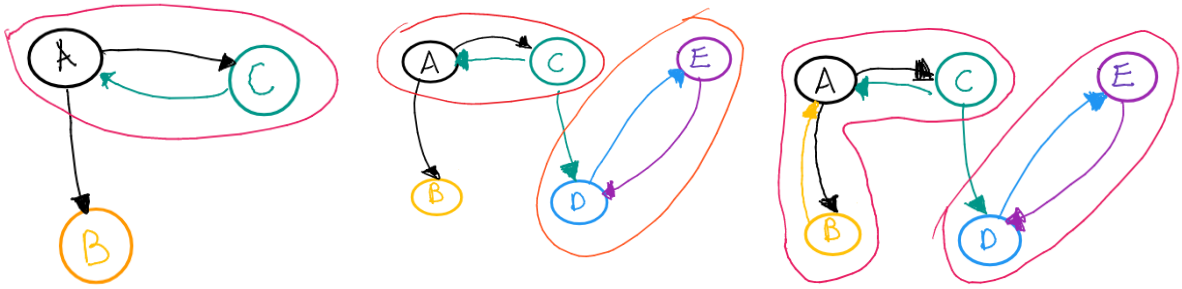
A partir del código anterior utilizado para hacer dos corridas del algoritmo, se obtiene el siguiente output obtenido:

```
erjomuza@Erjomuza:~/proyecto-estructuras/proyecto-tarjan/bin$ poetry run python algoritmo_tarjan.py
Grafo 1
Para el grafo con la lista de adyacencia
[[0, 1, 1], [0, 0, 0], [1, 0, 0]]
El número de cúmulos de nodos conexos entre sí es 2,
y lista de valores de low link es
[0, 1, 0]

Grafo 2
Para el grafo con la lista de adyacencia
[[0, 1, 1, 0, 0], [0, 0, 0, 0, 0], [1, 0, 0, 1, 0], [0, 0, 0, 0, 1], [0, 0, 0, 1, 0]]
El número de cúmulos de nodos conexos entre sí es 3,
y lista de valores de low link es [0, 1, 0, 3, 3]

grafo 3
Para el grafo con la lista de adyacencia
[[0, 1, 1, 0, 0], [1, 0, 0, 0, 0], [1, 0, 0, 1, 0], [0, 0, 0, 0, 1], [0, 0, 0, 1, 0]]
El número de cúmulos de nodos conexos entre sí es 2,
y lista de valores de low link es [0, 0, 0, 3, 3]
```

Figuras representativas de los grafos que se probaron dentro del ejemplo:



Revisando los resultados de Low Link para cada grafo diferente, luego de aplicar el algoritmo, se puede ver que en el caso del grafo 1, se tiene que el nodo 1 y el nodo 3 tienen un mismo valor de Low Link, por lo tanto están fuertemente conectados, haciendo la revisión visual de la representación gráfica se puede ver como el A y el C, es decir el 1ero y el 3ero. Haciendo el mismo análisis en el segundo grafo, se tiene que se puede ver que en la lista de valores de Low Link, los que tienen un mismo valor son el 1ero y el 3er nodo, es decir A y C, como se ve en la imagen se confirma que son fuertemente conectados, además el 4to elemento y el 5to tienen una conexión fuerte, de igual forma se puede ver en los nodos D y E de la segunda representación. Para el tercer ejemplo, ahora se tiene que los primeros 3 son un cúmulo de nodos fuertemente conectados y los últimos 2 también, se nota como en representación gráfica esto se cumple.

Discusión de los resultados:

La implementación del algoritmo de Tarjan en Python requirió de una investigación por sí sola la cual llevó a un entendimiento mayor del lenguaje y de sus diferentes formas de ser utilizado. Por ejemplo la creación y uso de un programa más estructurado en el cual ciertas clases pertenecían a un módulo, como se hizo con la clase **Grafo** y la clase **Nodo**, ambas pertenecientes al módulo grafo dentro del programa. Además el uso de una clase para la declaración del **AlgoritmoTarjan,** la cual fue implementada debido a que se considera que usar una clase es importante ya que el algoritmo como tal contiene información que es importante para sí, como sus listas utilizadas para tener indicadores, sus listas para manter un tracking de los valores que han sido visitados y los que no, por otro lado tiene sus métodos propios, privados para encontrar los valores de Low-Link, el cual además llama al otro método para hacer la búsqueda en profundidad que está encargado de realizar la búsqueda siguiendo el algoritmo de Tarjan y en esta estructura se tiene acceso a su propiedad llamada “resultadoLowLink”, entonces luego de crear una nueva instancia de **AlgoritmoTarjan** con un **Grafo** como parámetro como es indicado en el código**,** se puede acceder a la lista que contiene los valores de Low-link para los nodos del grafo que entra de parámetro, además del número de cúmulos de nodos fuertemente conectados.

Por otro lado es importante recalcar la capacidad del lenguaje de trabajar en los paradigmas de programación, ya que para hacer estas pruebas de los ejemplos se puede utilizar un scripting meramente funcional, y por otro lado se tiene la parte orientada a objetos dentro de las estructuras mencionadas.

Con respecto a los resultados de los corridas de ejemplo del algoritmo, se puede resaltar que se pudo implementar una versión del algoritmo capaz de obtener resultados correctos acerca de los valores de low-link obtenidos para los nodos de un grafo, es decir se puede identificar cuáles nodos pertenecen a un cúmulo fuertemente conexo y se pudo realizar la verificación, dibujando el grafo y analizando el resultado obtenido de los valores de Low Link obtenidos a través del uso del algoritmo de Tarjan implementado.

Conclusiones:

A través del trabajo de investigación realizado se pueden llegar a las siguientes conclusiones:

1. El lenguaje de programación de Python se puede adaptar para ser usado de diferentes formas, lo cual permitió el desarrollo del proyecto de forma relativamente sencilla, y muy intuitiva.
2. El algoritmo de Tarjan se puede utilizar para encontrar relaciones fuertes entre diferentes los miembros de un sistema compuesto en el cual las relaciones de esos miembros se pueda representar como un grafo dirigido, como el ejemplo de redes de computadoras, o servidores, incluso relaciones entre seres vivos, y así analizar cuáles interactúan más entre sí.
3. La implementación del algoritmo de Tarjan en Python se pudo hacer de forma satisfactoria, y se pudo comprobar su eficacia.
4. El uso de la programación orientada a objetos permite la solución de problemas más estructurada e intuitiva, ya que se puede observar qué variables son importantes para cada parte del programa.

Referencias:

[1] R. Tarjan, “DEPTH-FIRST SEARCH AND LINEAR GRAPH ALGORITHMS*” SIAM, SIAM J. Comput., 1972.

[2]B. Baka, “Python Data Structures and Algorithms”, Packt Publishing, Birmingham -Mumbai, 2018

[3] R. Sedwick, “Strong components - directed graphs,” Princeton University, Coursera.org. Disponible en: https://www.coursera.org/lecture/algorithms-part2/strong-components-fC5Yw.

[4] D. Heap, “Depth-first search (DFS).” Toronto University, 2002, Disponible en: http://www.cs.toronto.edu/~heap/270F02/node36.html

[5] B. Kim, “TIL - Tarjan's Algorithm”, Dev.to, 2019. Disponible en: https://dev.to/bkbranden/til-tarjan-s-algorithm-2p42

[6] W. Fiset, “Tarjans Strongly Connected Components algorithm | Graph Theory”, Youtube, 21/2/2019, Disponible en: https://www.youtube.com/watch?v=TyWtx7q2D7Y