Articulation Points

Andrés Valencia Oliveros[1,2]

Facultad de Ingeniería, Diseño e Innovación
Institución Universitaria Politécnico Grancolombiano
Bogotá, Colombia

Resumen

Definir si un grafo está conectado o desconectado, es un concepto fundamental de la teoría de grafos e importante en los modelos de redes, aplicables para gran variedad de problemas de decisión y que al ser analizados se pueden tomar decisiones más optimas al problema relacionado con puntos y caminos críticos.

Keywords: graph, articulation point, cut vertex, bridges.

1. Introducción

En las ciencias de la computación, los grafos son una estructura de datos que permiten modelar diferentes tipos de problemas como, mapas (rutas de un mapa), computación distribuida (red), redes sociales (sociedad), base de datos (*big data*, *business intelligence*, *NoSQL*) y redes neuronales. Los puntos de articulación muestran los vértices que son críticos para mantener el grafo conectado y se pueden usar para encontrar vulnerabilidades. El documento está organizado de la siguiente manera: Sección de la teoría de grafos, puntos de articulación, puentes y glosario de términos. Finalmente, se propone el trabajo a futuro.

2. Teoría de grafos

En matemáticas y en ciencias de la computación, la teoría de grafos estudia las propiedades de los grafos. Un grafo $G(V, E)$ es una colección de puntos, llamados vértices o nodos $V = \{v_1, v_2, \dots\}$, y segmentos de línea que conectan esos puntos, llamados aristas o arcos (en inglés *edges*) $E = \{e_1, e_2, \dots\}$; cada arista e tiene dos *puntos finales*, que son vértices. Se escribe $u \overset{e}{-} v$, y significa que la arista e incide sobre los vértices u y v; en este caso se puede decir que e conecta los vértices u y v, o que los vértices u y v son adyacentes [1].

[1] GitHub: anvalenciao

[2] Email: anvalenciao@poligran.edu.co

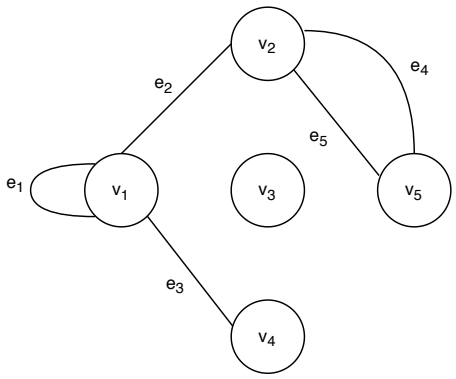

Figura 1. Ejemplo de un grafo. [1]

2.1. Grafo conexo

Un grafo G es conexo, si por cada dos vértices u y v, hay un camino (finito) que comienza en u y termina en v [1]. Para verificar si un grafo G es conexo, se puede aplicar un algoritmo determinista habitual, búsqueda en anchura en inglés *Breadth First Search* (BFS) o búsqueda en profundidad en inglés *Depth First Search* (DFS).

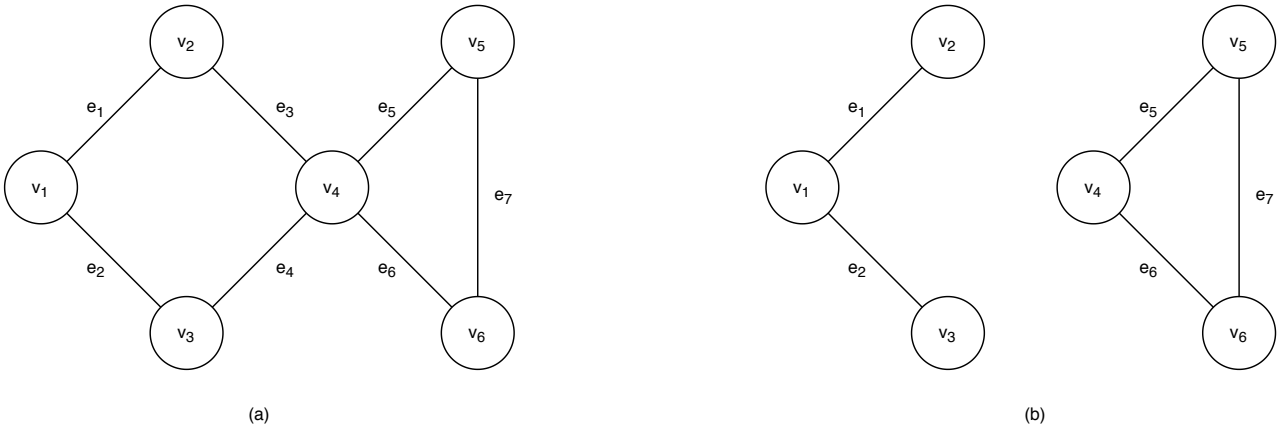

Figura 2. Tipos de grafos. (a) Conexo. (b) Disconexo.

2.2. Grafo dirigido o digrafo

Un digrafo o grafo dirigido $G(V, E)$ se define de manera similar a un grafo, excepto que el par de *puntos finales* (u, v) de cada arista ahora está ordenado. Se escribe $u \xrightarrow{e} v$, dónde u es el vértice inicial de e; y v es el vértice final de e. Se dice que la arista e está dirigida de u a v [1].

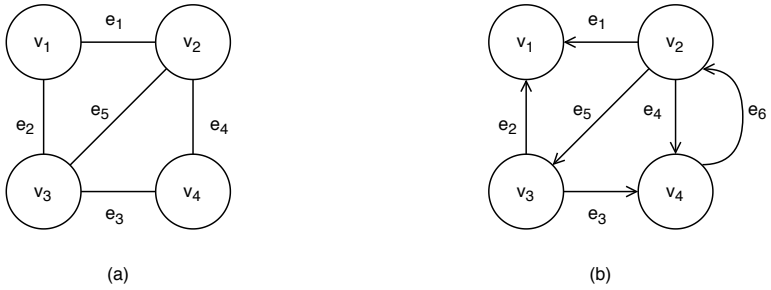

Figura 3. Tipos de grafos. (a) No dirigido. (b) Dirigido o digrafo.

3. Puntos de articulación

Un vértice v es un punto de articulación (o vértice de corte), si al eliminar el vértice v del grafo aumenta el número de componentes conectados. Es decir, genera algunos vértices inalcanzables para otros, se desconecta el grafo [2].

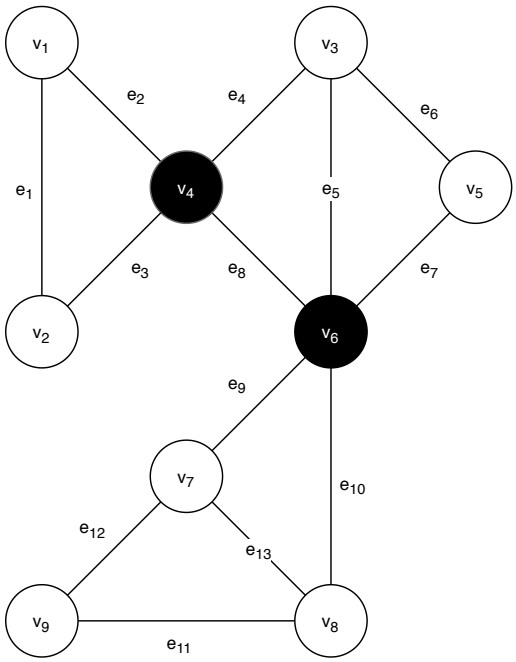

Figura 4. Ejemplo de grafo con dos puntos de articulación v_4 y v_6.

3.1. Algoritmo de Tarjan

El algoritmo de Tarjan se basa en una búsqueda en profundidad DFS y permite encontrar los puntos de articulación en un grafo, la complejidad del tiempo de ejecución para este algoritmo es lineal, la misma del DFS, $O(V + E)$. La complejidad espacial es igual al número total de vértices $O(V)$ [3].

3.1.1. *Pseudocódigo*

```
1: function GetArticulationPoints(i, d)
2:     visited[i] ← true
3:     disc[i] ← d
4:     low[i] ← d
5:     childCount ← 0
6:     isArticulation ← false
7:     for all ni in adj[i] do
8:         if not visited[ni] then
9:             parent[ni] ← i
10:            GetArticulationPoints(ni, d + 1)
11:            childCount ← childCount + 1
12:            if low[ni] ≥ disc[i] then
13:                isArticulation ← true
14:            end if
15:            low[i] ← Min(low[i], low[ni])
16:        else if ni ≠ parent[i] then
17:            low[i] ← Min(low[i], disc[ni])
18:        end if
19:    end for
20:    if (parent[i] ≠ null and isArticulation) or (parent[i] = null and childCount > 1) then
21:        return Output i as articulation point
22:    end if
23: end function
```

3.1.2. *Implementación*

En el grafo que se muestra en la Figura 5, se observa la representación del mismo en un árbol DFS, donde un vértice u es padre de otro vértice v, y u es un punto de articulación si satisface una de las siguientes dos condiciones [4]:

1. u es la raíz del árbol DFS y tiene al menos dos hijos independientes (no están conectados entre sí).
2. Si u no es la raíz comprueba si $low[v]$ es mayor o igual a $disc[u]$.

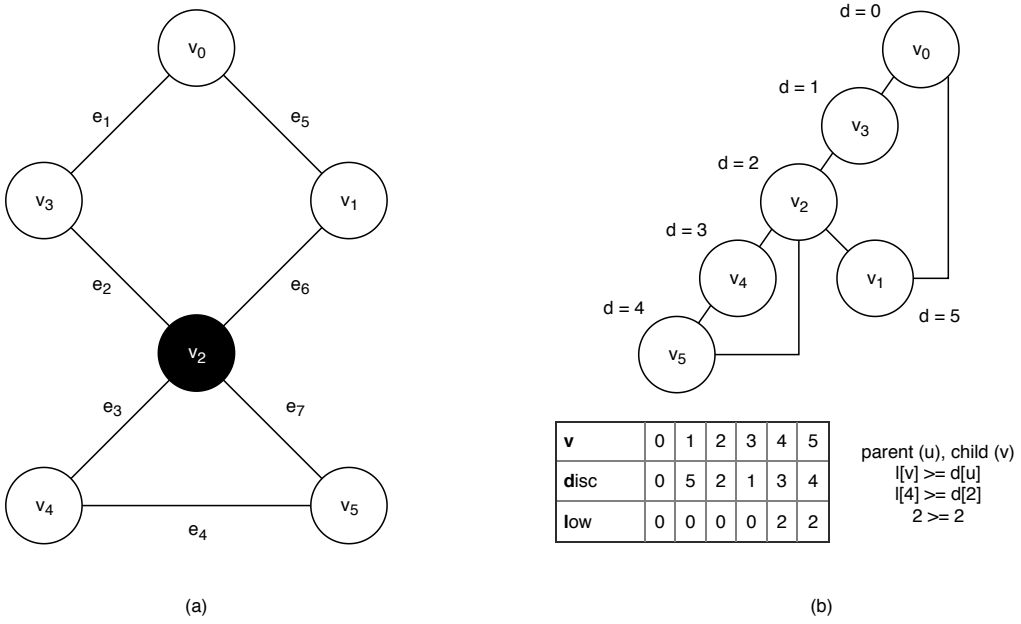

Figura 5. Punto de articulación. (a) Grafo con punto de articulación en el vértice v_2. (b) Árbol DFS del grafo y *Back Edge*.

El siguiente código en JavaScript, es la implementación del algoritmo de Tarjan. No sé recomienda ejecutarlo en ambientes de producción, se desarrollo en este lenguaje para ser probado en el lado del servidor (Node.js) y en el lado del cliente (Navegador Web Moderno).

Algoritmo 1: Código en JavaScript

```
/**
 * Test: Ubuntu 19.04 / node v10.15.2
 * Command: node /{path}/Tarjan.js
 * Based on: https://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/
 */
class Graph {
  constructor() {
    this.graph = new Object();
    this.time = 0; // Valor actual del tiempo de descubrimiento.
  }

  addEdge(u, v) {
    if (this.graph[u] === undefined) {
      this.graph[u] = new Array();
    }

    if (this.graph[v] === undefined) {
      this.graph[v] = new Array();
    }

    this.graph[u].push(v);
    this.graph[v].push(u);
  }

  DFS(u = 0, visited = [], ap = [], parent = [], low = [], disc = []) {
    let childCount = 0;
    visited[u] = true;

    disc[u] = this.time;
    low[u] = this.time;
    this.time += 1;

    for (const v of this.graph[u]) {
      if (visited[v] == false) {
        parent[v] = u;
        childCount += 1;

        this.DFS(v, visited, ap, parent, low, disc);

        low[u] = Math.min(low[u], low[v]);
        if (parent[u] == -1 && childCount > 1) {
          ap[u] = true;
        }

        if (parent[u] != -1 && low[v] >= disc[u]) {
          ap[u] = true;
        }

      } else if (v != parent[u]) {
        low[u] = Math.min(low[u], disc[v]);
      }
    }
  }

  ArticulationPoint() {
    let V =  Object.keys(this.graph).length; // Numero de vertices.
    let visited = new Array(V), // Denota si se visita o no un vertice durante el DFS.
        disc =  new Array(V), // Almacenan el tiempo de descubrimiento de cada vertice.
        low =  new Array(V), // Almacena, para cada vertice, el tiempo descubierto del vertice mas antiguo.
        parent =  new Array(V), // Almacena el padre de cada vertice en el arbol DFS.
        ap = new Array(V); // Almacena los puntos de articulacion.

    for (let i = 0; i < V; i++)
    {
        parent[i] = -1;
        visited[i] = false;
        ap[i] = false;
```

```
68      }
69
70      for (let i = 0; i < V; i++) {
71        if (visited[i] == false) {
72          this.DFS(i, visited, ap, parent, low, disc);
73        }
74      }
75
76      for (let i = 0; i < V; i++) {
77        if (ap[i] == true) {
78          // Imprime los puntos de articulacion.
79          console.log(i);
80        }
81      }
82
83    }
84  }
85
86  // { 0: [ 1, 3 ], 1: [ 0, 2 ], 2: [ 1, 3, 4, 5 ], 3: [ 0, 2 ], 4: [ 2 ], 5: [ 2 ] }
87  g = new Graph();
88  g.addEdge(0, 1);
89  g.addEdge(0, 3);
90  g.addEdge(1, 2);
91  g.addEdge(3, 2);
92  g.addEdge(2, 4);
93  g.addEdge(2, 5);
94  g.ArticulationPoint();
```

4. Puentes

Una arista se llama puente si al eliminarla del grafo (manteniendo los vértices) aumenta el número de componentes conectados [2].

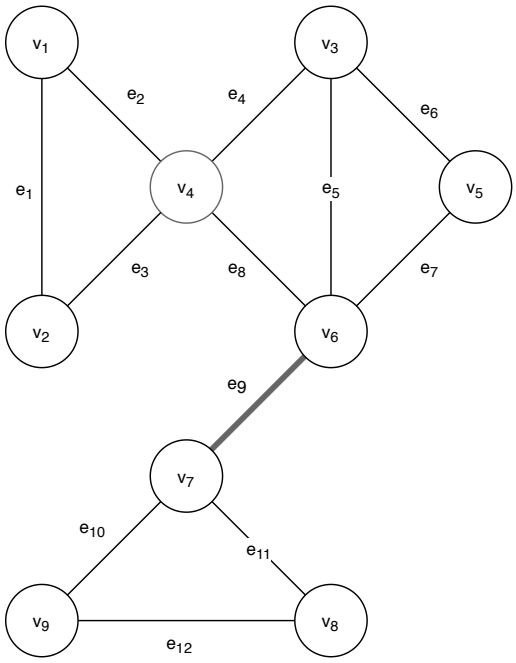

Figura 6. Ejemplo de grafo con una arista puente e_9.

5. Trabajo futuro

Como continuación de este trabajo, se presentan problemas propuestos en *UVa Online Judge* relacionados a la categoría de puntos de articulación. Se propone la siguiente colección de problemas:

- UVa 315 - Network
- UVa 610 - Street Directions
- UVa 796 - Critical Links
- UVa 10199 - Tourist Guide
- UVa 10765 - Doves and bombs

Glosario de términos

adyacentes Si una arista conecta dos vértices, se dice que son adyacentes. 1

algoritmo determinista Su comportamiento se puede predecir completamente a partir de la entrada, el algoritmo realiza los mismos cálculos y ofrece los mismos resultados[5]. 2

BFS *Breadth First Search*. 2

DFS *Depth First Search*. 2, 3

puntos finales Dos vértices conectados por una arista. 1, 2

Referencias

[1] S. Even, *Graph algorithms*. Cambridge University Press, 2011.

[2] V. Jaimini, "Articulation Points and Bridges Tutorials & Notes — Algorithms — HackerEarth," 2017.

[3] "Biconnected component."

[4] A. Bari, "5.2 articulation point and biconnected components," 2018.

[5] P. E. Black, "deterministic algorithm," 2009.