Articulation Points

Andrés Valencia Oliveros[1,2]

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

Resumen

...

Keywords: articulation point, cut vertex

1 Introducción

...

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].

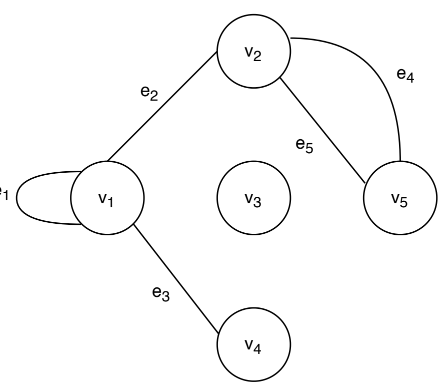

Fig. 1. Ejemplo de un grafo. [1]

[1] GitHub: anvalenciao

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

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).

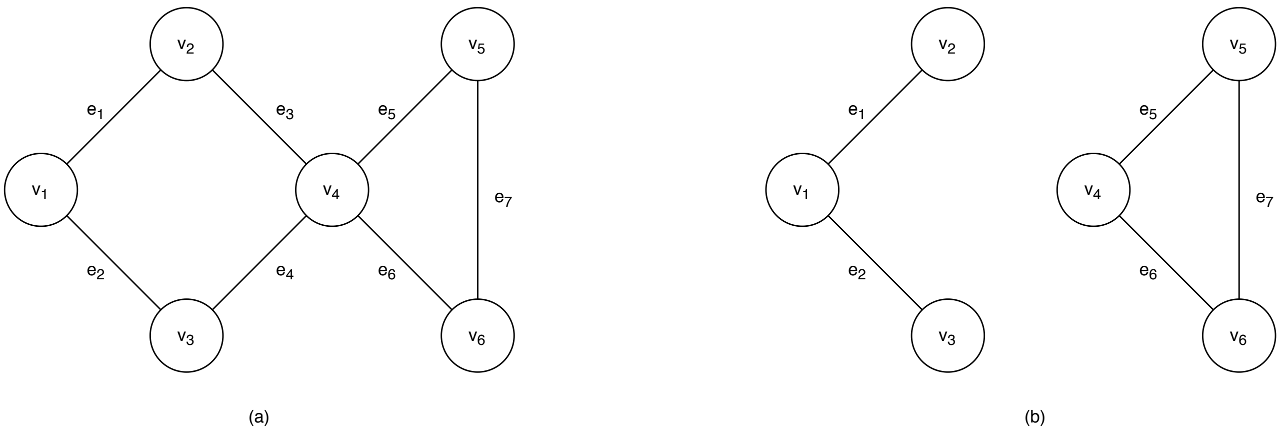

Fig. 2. Tipos de grafos. (a) Conexo. (b) Disconexo.

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].

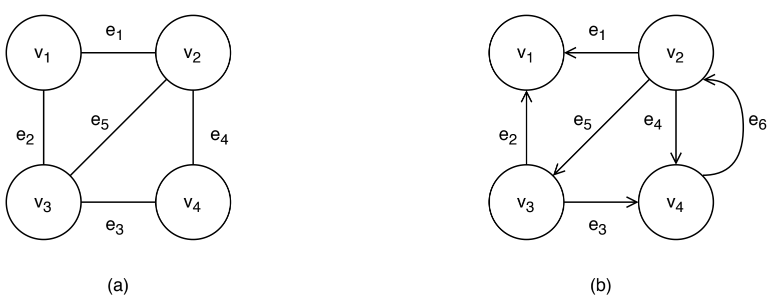

Fig. 3. Tipos de grafos. (a) No dirigido. (b) Dirigido o digrafo.

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.

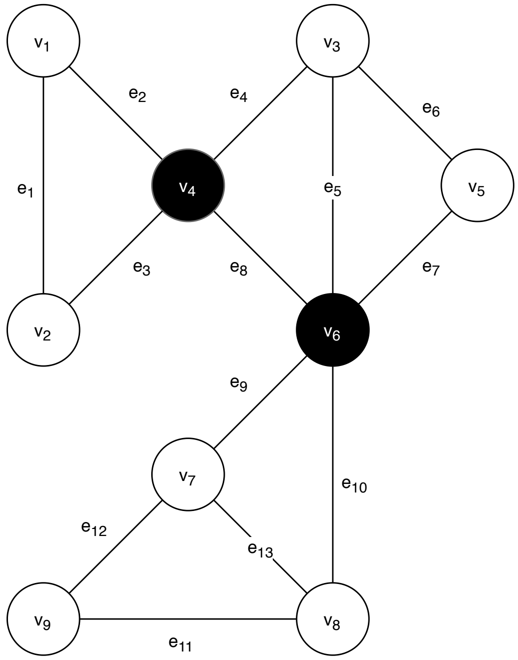

Fig. 4. Ejemplo de grafo con dos puntos de articulación v_4 y v_6.

4 Puentes

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

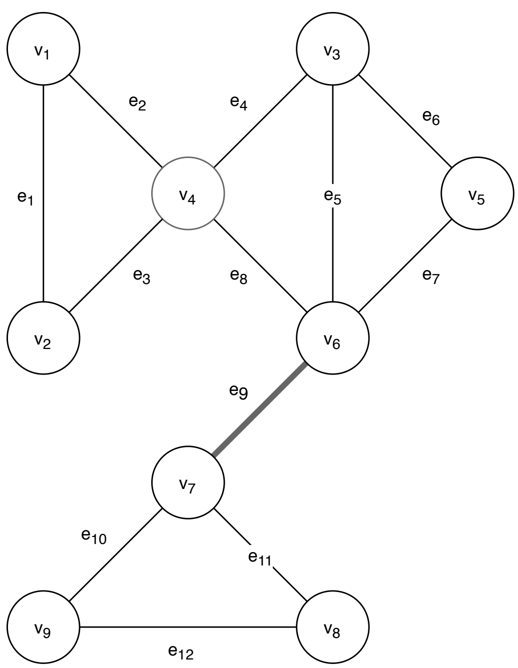

Fig. 5. Ejemplo de grafo con una arista puente e_9.

5 Algoritmos

lorem ipsum dolor sit amet. /home/andres/Documentos/ArticulationPoints/document/Algorithms/code

5.1 *Algoritmo de Tarjan*

El algoritmo de Tarjan para encontrar puntos de articulación

5.1.1 *Pseudocódigo*

Algorithm 1 Linear time depth first search

```
function GetArticulationPoints(i, d)
    visited[i] ← true
    depth[i] ← d
    low[i] ← d
    childCount ← 0
    isArticulation ← false
    for all ni in adj[i] do
        if not visited[ni] then
            parent[ni] ← i
            GetArticulationPoints(ni, d + 1)
            childCount ← childCount + 1
            if low[ni] ≥ depth[i] then
                isArticulation ← true
            end if
            low[i] ← Min(low[i], low[ni])
        else if ni ≠ parent[i] then
            low[i] ← Min(low[i], depth[ni])
        end if
    end for
    if (parent[i] ≠ null and isArticulation) or (parent[i] = null and childCount > 1) then
        return Output i as articulation point
    end if
end function
```

Algoritmo 1: sample code

```
// https://medium.com/@ziyoshams/graphs-in-javascript-cc0ed170b156
// https://github.com/mission-peace/interview/blob/master/src/com/interview/graph/ArticulationPoint.java
// https://en.wikipedia.org/wiki/Biconnected_component

class Graph {
  constructor(vertices) {
    this.V = vertices;
    this.graph = new Object();
    this.Time = 0;
  }

  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);
  }

  APUtil(u, visited, ap, parent, low, disc) {
    let children = 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;
        children += 1;

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

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

  AP() {
    var visited = new Array(this.V),
        disc =  new Array(this.V),
        low =  new Array(this.V),
        parent =  new Array(this.V),
        ap = new Array(this.V);

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

    for (let i = 0; i < this.V; i++) {
      if (visited[i] == false) {
        this.APUtil(i, visited, ap, parent, low, disc);
      }
    }

    for (let i = 0; i < this.V; i++) {
      if (ap[i] == true) {
        console.log(i + " ");
      }
    }

  }
}

// g1 = {0: [1], 1: [0, 2], 2: [1, 3], 3: [2]};
g1 = new Graph(4);
g1.addEdge(0, 1);
g1.addEdge(1, 2);
g1.addEdge(2, 3);

/*g1 = new Graph(5)
g1.addEdge(1, 0);
g1.addEdge(0, 2);
g1.addEdge(2, 1);
g1.addEdge(0, 3);
g1.addEdge(3, 4);*/

/*g1 = new Graph(7);
g1.addEdge(0, 1);
g1.addEdge(1, 2);
g1.addEdge(2, 0);
g1.addEdge(1, 3);
```

```
g1.addEdge(1, 4);
g1.addEdge(1, 6);
g1.addEdge(3, 5);
g1.addEdge(4, 5);*/
g1.AP();
//console.log(g1);
```

5.1.2 Complejidad

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[2]. 2

BFS *Breadth First Search*. 2

DFS *Depth First Search*. 2

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

Referencias

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

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