Universidad Nacional Mayor de San Marcos

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMATICA

Página Web: Orden Topológico y Puntos de Articulación

Curso : Estructura de datos

Profesor : Herminio Paucar Curasma

Ciclo: 5to

Escuela: Ingeniería de Software

Integrantes:

- Antunez Palomino ,Kori Xiomara(3)
- Gomez Cavero, Mishell(3)
- Santiago Arapa, Naysha Solange (3)
- Pérez Fonseca, Juan Diego (3)

Lima - Perú

septiembre , 2020

Índice

1. Introducción

Un grafo es una estructura de datos abstracta que se utiliza para implementar el concepto matemático de los grafos. Consiste en una colección de vértices (también llamados nodos) y aristas que conectan estos vértices. Un grafo se suele considerar como una generalización de la estructura del árbol, en la que en lugar de tener una relación puramente padre-hijo entre los nodos del árbol, puede existir cualquier tipo de relación compleja. Los gráficos se utilizan ampliamente para modelar cualquier situación en la que las entidades o cosas se relacionan entre sí por pares. Por ejemplo, árboles familiares (en los que los nodos miembros tienen una ventaja de padre a cada uno de sus hijos), redes de transporte (en las que los nodos son aeropuertos, intersecciones, puertos, etc, y las aristas pueden ser vuelos de aerolíneas, carreteras de un solo sentido, rutas de navegación, etc). (Reema, T. 2014).

El presente informe tiene como objetivo documentar el trabajo final del curso de Estructura de Datos, el cual consiste en la implementación de los algoritmos de Ordenamiento Topológico y de Puntos de Articulación, ambos aplicados a Grafos.

2. Marco Teórico

2.1. Algoritmos

Los algoritmos utilizados para el desarrollo del proyecto:

2.1.1. DFS

El algoritmo de búsqueda en profundidad avanza al expandir el nodo inicial de G y luego ir más y más profundamente hasta que se **encuentre el nodo objetivo**, o hasta que se encuentre un nodo que no tiene encontrado. Cuando se llega a un callejón sin salida, el algoritmo retrocede, volviendo al más reciente nodo que no ha sido completamente explorado.
En otras palabras, la búsqueda en profundidad comienza en un nodo inicial A que se convierte en el nodo actual.
Luego, examina cada nodo N a lo largo de una ruta P que comienza en A. Es decir, procesamos un vecino de A, luego un vecino de vecino de A, y así sucesivamente. Durante la ejecución del algoritmo, si llegar a una ruta que tiene un nodo N que ya ha sido procesado, luego retrocedemos a la actual nodo. De lo contrario, el nodo no visitado (no procesado) se convierte en el nodo actual.

El algoritmo procede así hasta que lleguemos a un callejón sin salida (final del camino P). Al llegar a los muertos final, retrocedemos para encontrar otro camino P ¢. El algoritmo termina cuando retroceder conduce de nuevo a la nodo inicial A. En este algoritmo, los bordes que conducen a un nuevo vértice son llamados bordes de descubrimiento y bordes que conducen a un vértice ya visitado son llamados bordes traseros.
Observe que este algoritmo es similar al recorrido en orden de un árbol binario. Su implementación es similar a la del algoritmo de búsqueda en amplitud, pero aquí use una pila en lugar de una cola. Nuevamente, usamos una variable STATUS para representar el estado actual de el nodo.

2.1.2. Ordenamiento Topológico

Un ordenamiento topológico (topological sort en inglés) de un grafo acíclico *G* dirigido es una ordenación lineal de todos los nodos de *G* que conserva la unión entre vértices del grafo

G original. La condición de que el grafo no contenga ciclos es importante, ya que no se puede obtener ordenación topológica de grafos que contengan ciclos.

Un ejemplo para entender el concepto es pensar en tareas que se preceden y se suceden. Este ordenamiento ordena el grafo de forma que nunca se realice una tarea que requiera de una predecesora para ocurrir (como las materias en una universidad, no se puede llevar cálculo 2 sin antes haber llevado cálculo 1).La ordenación topológica por tanto es una lista en orden lineal en que deben realizarse las tareas.

Para poder encontrar la ordenación topológica del grafo *G* deberemos aplicar una modificación del algoritmo de búsqueda en profundidad (DFS).

Pseudocódigo DFS

```
DFS(grafo G)
    PARA CADA vértice u ∈ V[G] HACER
            estado[u] ← NO_VISITADO
            padre[u] ← NULO
    tiempo ← 0
    PARA CADA vértice u ∈ V[G] HACER
            SI estado[u] = NO_VISITADO ENTONCES
                    DFS_Visitar(u,tiempo)

DFS_Visitar(nodo u, int tiempo)
    estado[u] ← VISITADO
    tiempo ← tiempo + 1
    d[u] ← tiempo
    PARA CADA v ∈ Vecinos[u] HACER
            SI estado[v] = NO_VISITADO ENTONCES
                    padre[v] ← u
                    DFS_Visitar(v,tiempo)
    estado[u] ← TERMINADO
```

```
    tiempo ← tiempo + 1

    f[u] ← tiempo
```

2.1.3. Punto de articulación

En teoría de grafos, un **vértice de corte** o **punto de articulación** es un vértice de un grafo tal que al eliminarlo de éste se produce un incremento en el número de componentes conexos. Si el grafo estaba conectado antes de retirar el vértice, entonces pasará a desconectarse.
Encontrar puntos de articulación es muy útil porque, entre otros casos, permite hallar los puntos débiles en redes y así corregir y fortalecer. Si una red tiene un punto de articulación significa que si ese nodo se daña la red quedará separada en dos y no habrá comunicación
El algoritmo funciona haciendo uso de un DFS modificado.

Algoritmo DFS

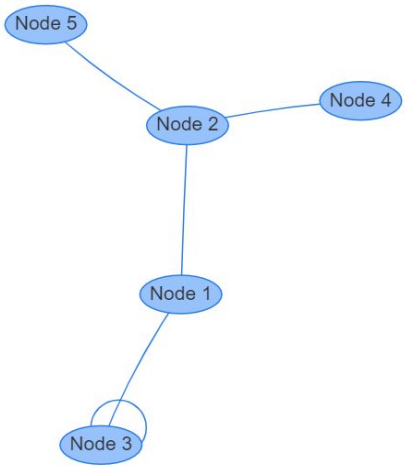
```
Método DFS (nodo)
        nodoVisitado[nodo] = true;
        Si (nodo tiene adyacentes) entonces
                tiempoDescubrimiento[nodo] = cont
                ordenDescubrimiento[cont] = nodo
                cont++
        Fin Si
        Para w desde 1 hasta cantidadAdyacentes hacer
                Si(adyacentes[w] no fue visitado) entonces
                        ady = adyacentes[w]
                        padre[ady] = nodo
                        DFS(ady)
                Si no
                        Si(ady != padre[nodo]) entonces
                                adyacente es arista de retorno de nodo
                        Fin Si
                Fin Si
        Fin Para
Fin DFS
```

```
Método recorridoDelArbloDFS ()
Para i desde 0 hasta numeroVertices hacer
        entero nActual = ordenDescubri[i]
        menorTiempoDescubri[nActual]= i
        Si (nActual tiene aristas de retorno) entonces
                entero j = 1
                Mientras (aristasRetorno[nActual][j] != vacio) hacer
                        entero nRtrn = aristasRetorno[nActual][j]
                         Si(menorTiempoDescubri[nActual] >  tiempoDescubri[nRtrn])
                                menorTiempoDescubri[nActual] = tiempoDescubri[nRtrn]
                                entero x = nActual
                                Mientras ( padre[x] != nRtrn)
                                   entero pa = padre[x]
                                   menorTiempoDescubri[pa] = menorTiempoDescubri[x]
                                x = pa
                                        Fin Mientras
                        Fin Si
                                j++
                Fin Mientras
        Fin Si
Fin Para
Fin Metodo

Método encontrarNodo()

        Para i desde 2 hasta  v-1  hacer

        hijo <- ordenDescubri[i]

        padre <- padre[hijo]


                Si menorTiempoDescubri[hijo] >=tiempoDescubri[padre] entonces

                        nodosDeArticulacion[padre] <- verdadero

                Fin Si

        Fin Para

Fin Metodo
```

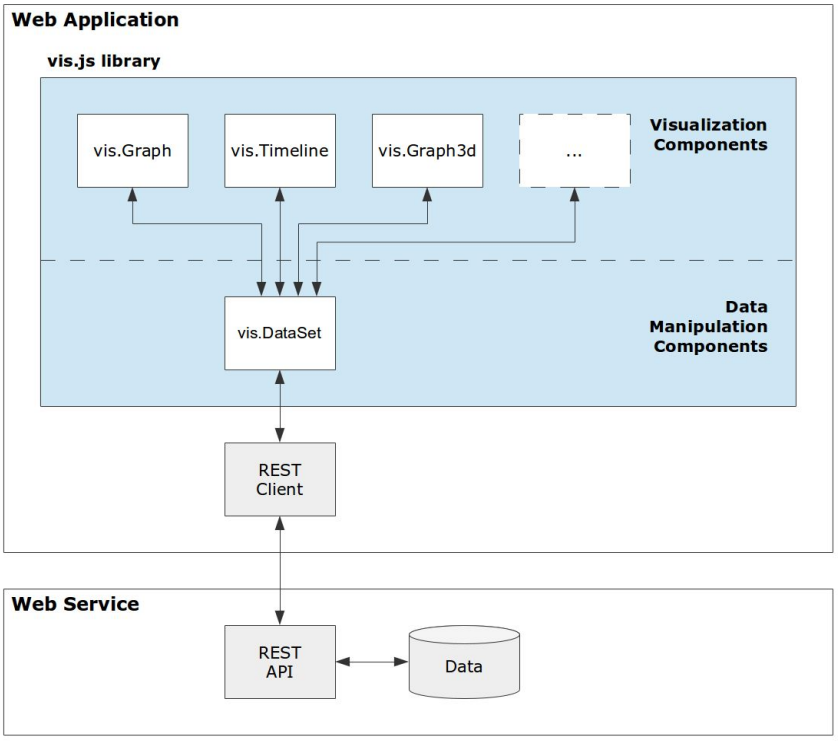
2.2. Librerías

2.2.1. Vis.Js

Vis.js es una librería diseñada para ser fácil de usar al mismo tiempo que permite la visualización de grandes conjuntos de datos dinámicos que se pueden manipular de forma interactiva. Las gráficas de esta librería se observan en la siguiente imagen.



Vis.js contiene los siguientes componentes:



- DataSet. Un conjunto de datos flexible basado en clave/valor. Agregar, actualizar y quitar elementos. Suscríbete a los cambios en el conjunto de datos. Un DataSet puede filtrar y ordenar elementos y convertir campos de elementos.
- DataView. Una vista filtrada o formateada en un Conjunto de datos.
- Red. Mostrar una red (gráfico dirigido por la fuerza) con nodos y bordes (anteriormente denominado gráfico).
- Graph2d. Trazar datos en una línea de tiempo con líneas o gráficos de barras.
- Graph3d. Mostrar datos en un gráfico tridimensional.
- Línea de tiempo. Mostrar diferentes tipos de datos en una línea de tiempo.

2.2.2. JavaScript

JavaScript, es uno de los más potentes e importantes lenguajes de programación en la actualidad, por tres enfoques claros: es útil, práctico y está disponible en cualquier navegador web.

JavaScript es creado por Brendan Eich y vio la luz en el año 1995 con el nombre de LiveScript, que luego fue nombrado JavaScript, nace como un lenguaje sencillo destinado a añadir algunas características interactivas a las páginas web. Sin embargo, hoy en día ha crecido de manera acelerada y es el lenguaje de programación que se utiliza en casi todos los sitios web en el mundo.

2.2.3. HT HTML

Es un lenguaje de marcado que se utiliza para el desarrollo de páginas de Internet. Se trata de la sigla que corresponde a HyperText Markup Language, es decir, Lenguaje de Marcas de Hipertexto, que podría ser traducido como Lenguaje de Formato de Documentos para Hipertexto.ML

2.2.4. CSS

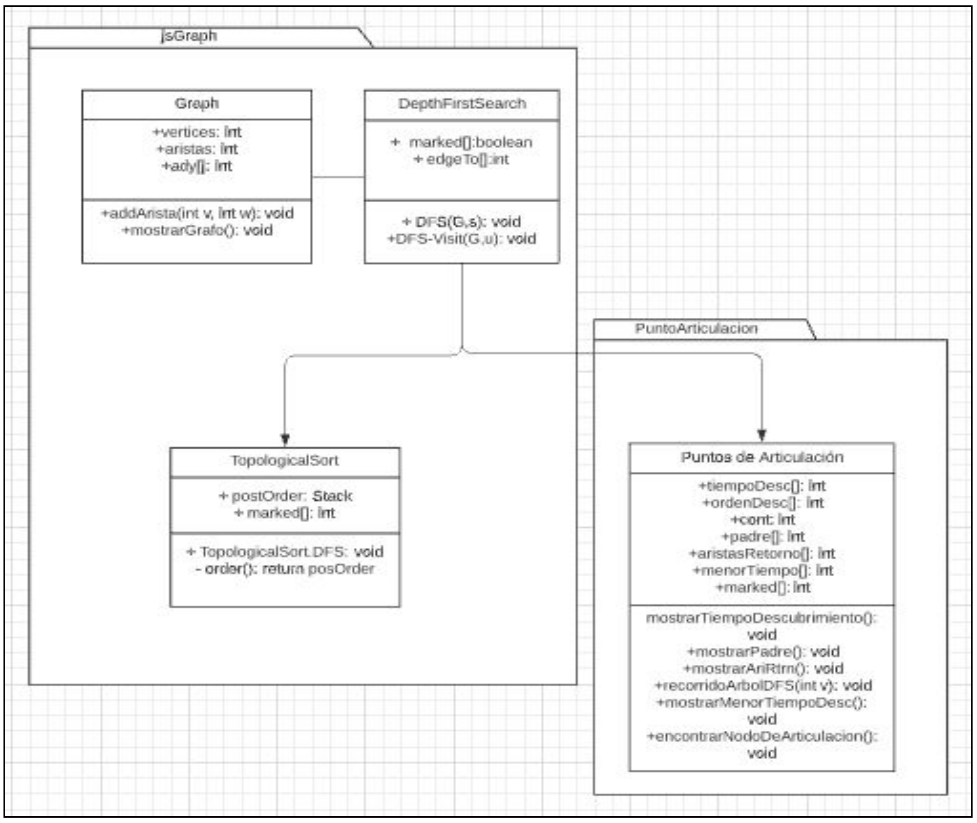
El lenguaje CSS es un lenguaje que determina el estilo de los documentos HTML. Abarca opciones relativas a fuentes, colores, márgenes, líneas, altura, anchura, imágenes de fondo, entre otros.

En la actualidad es posible utilizar lenguaje HTML para desarrollar el formato de páginas web. Sin embargo, el lenguaje CSS ofrece más opciones y es más preciso, además de que es compatible con todos los navegadores actuales.

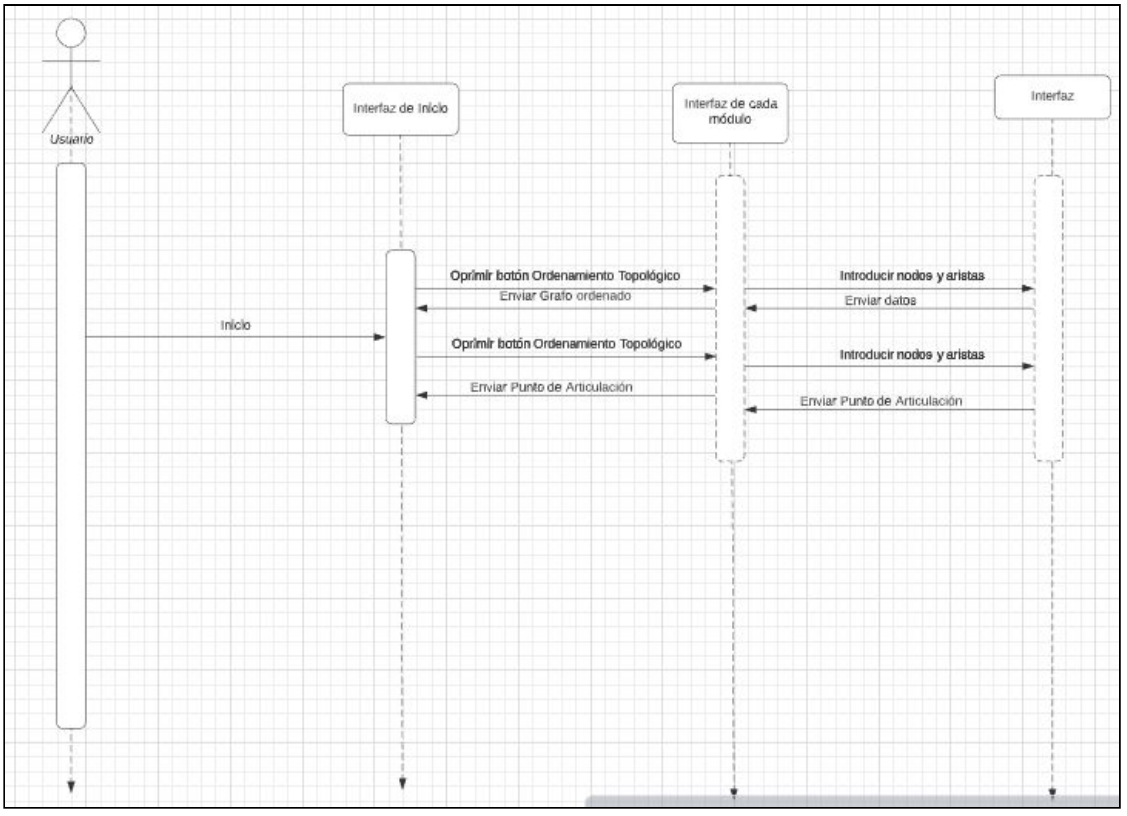
3. Metodología de Desarrollo

Para desarrollar la página web, primero se hizo un diagrama de clases; el cual, se tomó como guía para el desarrollo. En la fase de desarrollo primero se programó los algoritmos de los dos grafos, orden topológico dfs y puntos de articulación, en los archivos tipo javascript. Como segundo paso se procedió a codificar la estructura de la página web en los archivos tipo html donde se importaron los archivos javascript, las librerías usadas para realizar los grafos. Por último se codificaron los archivos de extensión css, donde se encuentran los diseños usados en la página. Obteniendo de esta manera la página web.

3.1. Diagrama de Clases



3.2. Diagrama de secuencia



3.3. Prototipos Mockups

Imagen .- Interfaz de Inicio

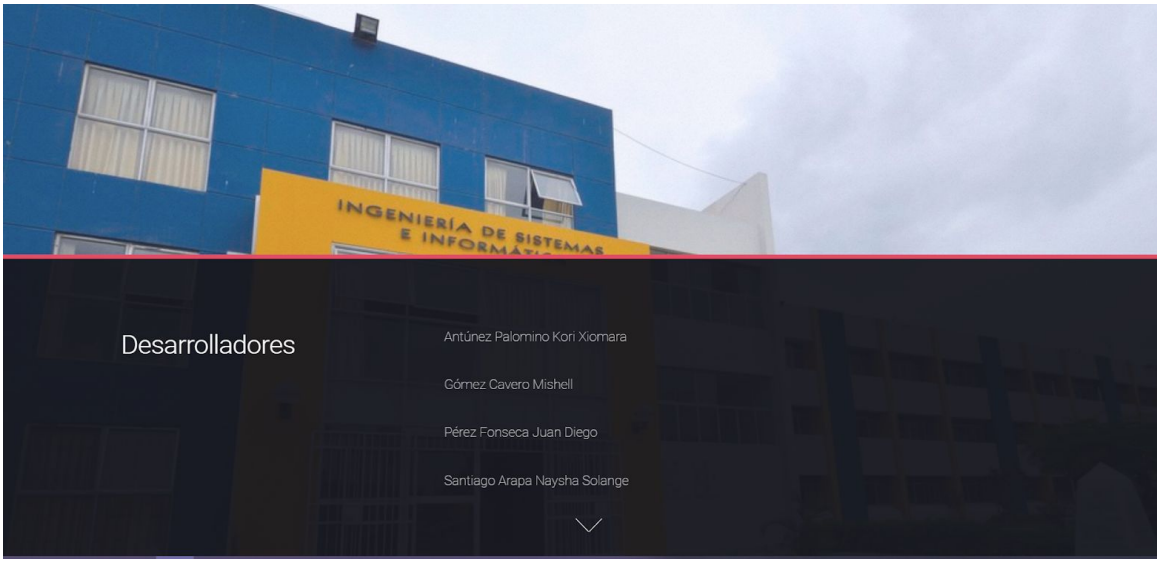


Imagen .- Interfaz de Inicio

Imagen .- Interfaz de Orden Topológico

Imagen .- Interfaz de Puntos Articulación

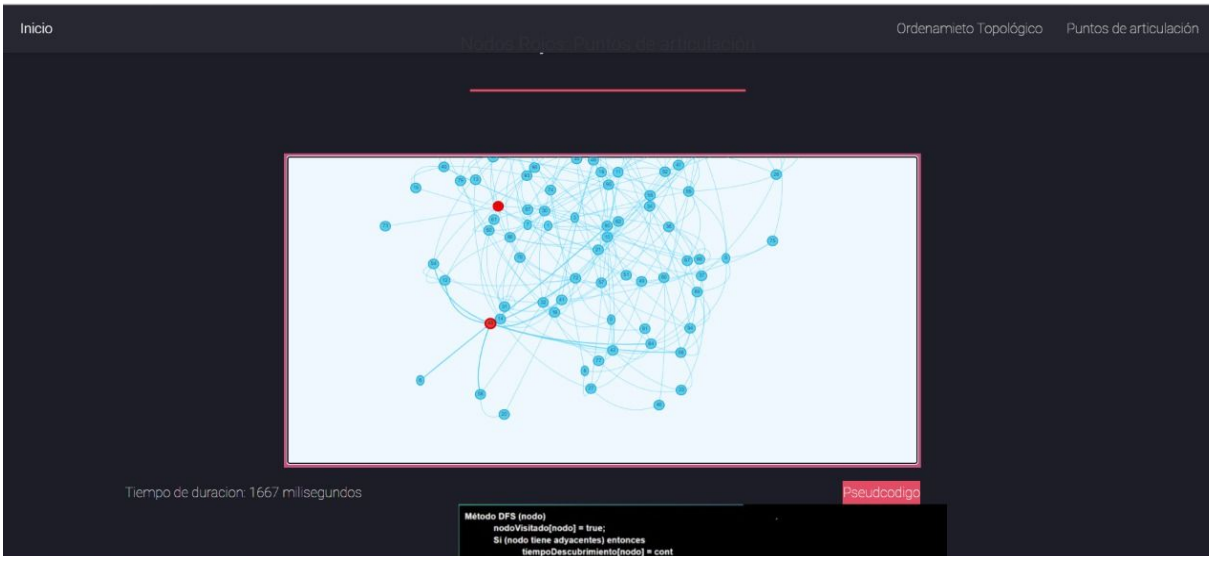


Imagen .- Interfaz de 100 nodos (Punto de articulación)

4. Aporte

Hemos realizado dos aportes principales a la página web como grupo, el primer aporte fue realizar el contenedor de la gráfica de la página con funcionalidad zoom in zoom out, la segunda funcionalidad es sobre los valores de entrada, se logró realizar que la entrada tenga la capacidad de recibir hasta los 100 nodos.

- Kori Antúnez Palomino: Desarrollo en parte del código de orden topológico usando programación estructurada.
- Naysha Santiago Arapa: Implementación lógica del algoritmo de Ordenamiento Topológica usando programación estructurada. Implementación de tiempo de ejecución de los algoritmos y pintado de grafo en pantalla.
- Mishell Gomez Cavero: Implementación lógica del algoritmo de Ordenamiento topológico usando programación orientada a objetos.
- Juan Diego Pérez Fonseca: Implementación lógica del algoritmo de Punto de Intersección usando programación estructurada. Implementación de tiempo de ejecución de los algoritmos y pintado de grafo en pantalla.

5. Conclusiones y Recomendaciones

En conclusión se logró implementar el desarrollo de los grafos y el coloreado tanto de los vértices como las aristas , gracias al uso de la librería Vis.js que nos proporciona JavaScript.

Como recomendaciones sugerimos tener conocimientos previos de desarrollo web para facilitar la migración de los diferentes tópicos de la Estructura de Datos desarrollados en lenguajes convencionales a lenguajes de programación actualmente demandados.

6. Referencias

- Reema, T. (2014). *Data Structures Using C* (segunda ed., Vol. 1). Oxford, Reino Unido: Oxford University Press.
- Trudeau, R. J. (1994). *Introduction to Graph Theory* (2nd Revised ed. ed.). New York, Estados Unidos: Dover Publications.
- West, D. B. (2000). *Introduction to Graph Theory* (2nd Revised ed. ed., Vol. 1). Illinois, Estados Unidos: Pearson.
- Gel'Fand, I. M., Glagoleva, E. G., & Shnol, E. E. (2002). Functions and Graphs;Dover Books on Mathematics (Illustrated ed., Vol. 1). New York, Estados Unidos: Dover Publications.
- Elmasry, A., Mehlhorn, K., & Schmidt, J. M. (2012). Every DFS Tree of a 3-Connected Graph Contains a Contractible Edge. Journal of Graph Theory, 72(1), 112-121. https://doi.org/10.1002/jgt.21635
- Tutorials Point. (2020). JavaScript - Date getTime() Method - Tutorialspoint. Recuperado de https://www.tutorialspoint.com/javascript/
- Librería Vis.js. (2019). Recuperado de https://visjs.org/
- Jacomy, A. (2019). Sigma.js. Recuperado 18 de septiembre de 2020, de http://sigmajs.org/