UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

ESCUELA ACADÉMICA PROFESIONAL DE INGENIERÍA DE SISTEMAS

ALGORÍTMICA 3

PROFESOR: Paucar Curasma, Herminio

TEMA DEL PROYECTO: Búsqueda de Puentes y Ciclos en un grafo de tamaño "n"

GRUPO 5

INTEGRANTES:

- Bayes Enriquez, Humberto Valentín 16200107
- Marquina Carihuasari, Alfonso Francisco 15200128
- Molina Soto, Lesli Lisbeth 15200132
- Muñoz Capcha, Alex Cristhian 16200137
- Parejas Fundar, Jorge Rubén 16200024
- Ramírez Martínez, Aldo Raúl 16200151

ÍNDICE

INTRODUCCIÓN 3
MARCO TEÓRICO 3
METODOLOGÍA DE DESARROLLO 7
Diagrama de Clases 7
Diagrama de Secuencia 8
Diagrama de Componentes 8
Prototipo: 9
APORTE 11
CONCLUSIONES Y RECOMENDACIONES. 12
REFERENCIAS 15

INTRODUCCIÓN

Nuestro objetivo fue desarrollar una aplicación web para entender cómo es que funciona el algoritmo Búsqueda de Puentes y Ciclos en un grafo de tamaño n, a su vez que esta aplicación resulte ser descriptiva al mostrarnos los algoritmos que implementa y siendo fácil de usar. Es así como dimos con vis la cual es una librería que nos ayudó con los grafos.

MARCO TEÓRICO

Java Script: es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

CSS: es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. CSS es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas.

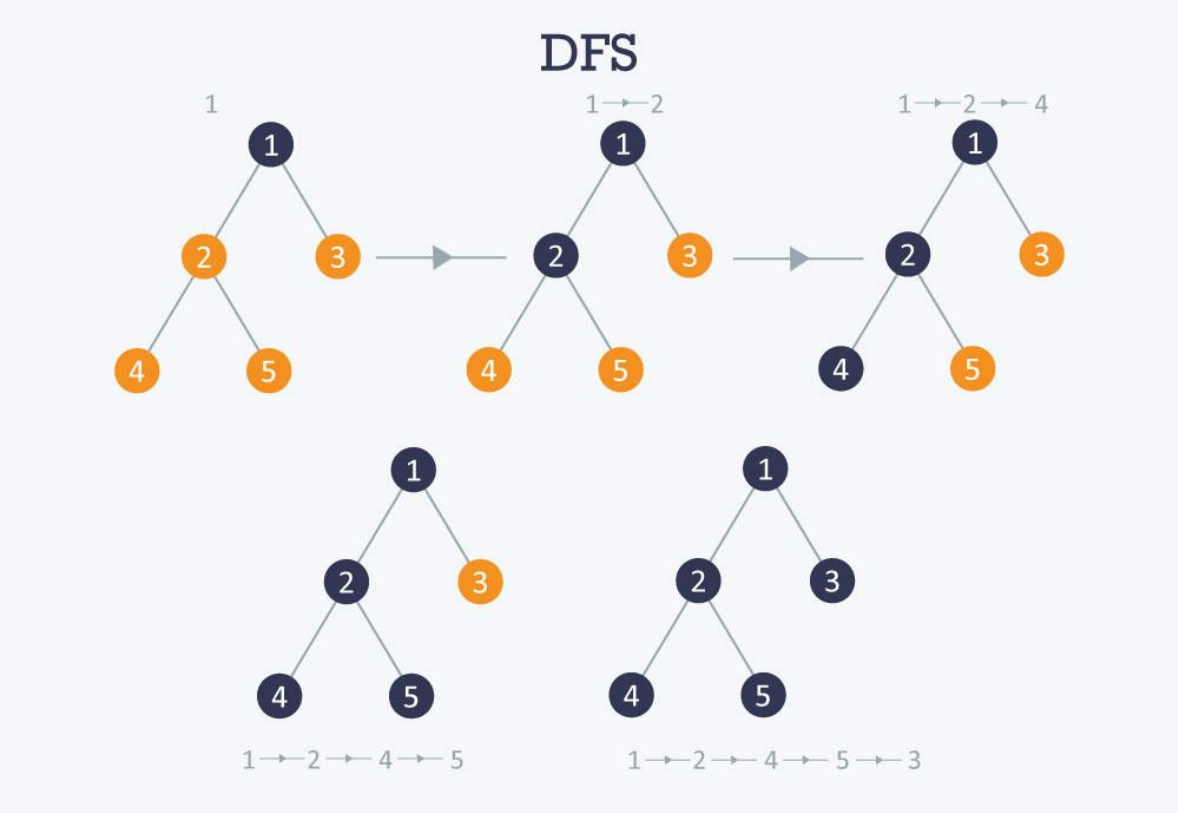
HTML5: es una colección de estándares que se usar en el diseño y desarrollo de páginas web. Es una colección cuya función es representar la forma en la que se presenta la información en un explorador de Internet, y al mismo tiempo la forma en la que se interactúa con ella.

TÉCNICAS DE PROGRAMACIÓN

DFS: El algoritmo DFS posee varias aplicaciones la más importante es para problemas de conectividad, si un grafo es conexo, detectar ciclos en un grafo, número de componentes conexas, etc y es bastante útil en otro algoritmos como para hallar las componentes fuertemente conexas en un grafo (Algoritmo de Kosaraju, Algoritmo de Tarjan), para hallar puntos de articulación o componentes biconexas (puentes), para recorrido en un circuito o camino euleriano, topological sort, flood fill y otras aplicaciones.

Como trabaja

DFS va formando un árbol al igual que BFS pero lo hace a profundidad. Existen dos formas de hacer el recorrido una es usando una Pila y otra de manera recursiva.



LIBRERÍA USADA

VIS: Una librería de visualización para proyectos Web que ha sido diseñada para que sea fácil de utilizar y gestionar grandes cantidades de datos dinámicamente, además de posibilitar la interacción del usuario con dichos datos.

Vis.js nos permite realizar distintos tipos de gráficos:

- ✓ Redes: conexiones y relaciones entre diferentes nodos.
- ✓ Líneas temporales: ver qué cambios ha habido a lo largo del tiempo en un determinado contexto.
- ✓ Gráficas 2D: gráficas en dos dimensiones, con ejes 'x' e 'y'.
- ✓ Gráficas 3D: gráficas tridimensionales para representar 3 coordenadas.

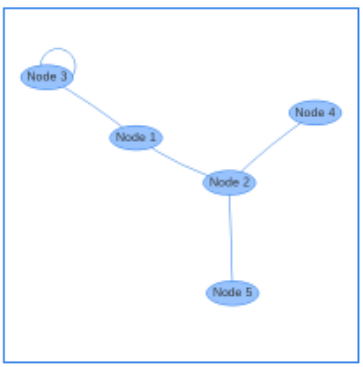
```
 * @constructor DataSet
 */
function DataSet(data, options) {
  // correctly read optional arguments
  if (data && !Array.isArray(data)) {
    options = data;
    data = null;
  }

  this._options = options || {};
  this._data = {}; // map with data indexed by id
  this.length = 0; // number of items in the DataSet
  this._fieldId = this._options.fieldId || 'id'; // name of the field containing id
  this._type = {}; // internal field types (NOTE: this can differ from this._options.type)

  // all variants of a Date are internally stored as Date, so we can convert
  // from everything to everything (also from ISODate to Number for example)
  if (this._options.type) {
    var fields = (0, _keys2['default'])(this._options.type);
    for (var i = 0, len = fields.length; i < len; i++) {
      var field = fields[i];
      var value = this._options.type[field];
      if (value == 'Date' || value == 'ISODate' || value == 'ASPDate') {
        this._type[field] = 'Date';
      } else {
        this._type[field] = value;
      }
    }
  }

  this._subscribers = {}; // event subscribers

  // add initial data when provided
  if (data) {
    this.add(data);
  }

  this.setOptions(options);
```

VIS.CSS

Librería utilizada para crear físicas a grafos ya creados de la misma.

Vis Network

Basic usage




```
var locales = {
  en: {
    edit: 'Edit',
    del: 'Delete selected',
    back: 'Back',
    addNode: 'Add Node',
    addEdge: 'Add Edge',
    editNode: 'Edit Node',
    editEdge: 'Edit Edge',
    addDescription: 'Click in an empty space to place a new node.',
    edgeDescription: 'Click on a node and drag the edge to another node to connect them.',
    editEdgeDescription: 'Click on the control points and drag them to a node to connect to it.',
    createEdgeError: 'Cannot link edges to a cluster.',
    deleteClusterError: 'Clusters cannot be deleted.',
    editClusterError: 'Clusters cannot be edited.'
  }
}
```

```
div.vis-network div.vis-manipulation {
  box-sizing: content-box;

  border-width: 0;
  border-bottom: 1px;
  border-style:solid;
  border-color: #d6d9d8;
  background: #ffffff; /* Old browsers */
  background: -moz-linear-gradient(top,  #ffffff 0%, #fcfcfc 48%, #fafafa 50%, #fcfcfc 100%); /* FF3.6+ */
  background: -webkit-gradient(linear, left top, left bottom, color-stop(0%,#ffffff), color-stop(48%,#fcfcfc), color-stop(50%,
  background: -webkit-linear-gradient(top,  #ffffff 0%,#fcfcfc 48%,#fafafa 50%,#fcfcfc 100%); /* Chrome10+,Safari5.1+ */
  background: -o-linear-gradient(top,  #ffffff 0%,#fcfcfc 48%,#fafafa 50%,#fcfcfc 100%); /* Opera 11.10+ */
  background: -ms-linear-gradient(top,  #ffffff 0%,#fcfcfc 48%,#fafafa 50%,#fcfcfc 100%); /* IE10+ */
  background: linear-gradient(to bottom,  #ffffff 0%,#fcfcfc 48%,#fafafa 50%,#fcfcfc 100%); /* W3C */
  filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#ffffff', endColorstr='#fcfcfc',GradientType=0 ); /* IE6-9 */

  padding-top:4px;
  position: absolute;
  left: 0;
  top: 0;
  width: 100%;
  height: 28px;
}

div.vis-network div.vis-edit-mode {
  position:absolute;
  left: 0;
  top: 5px;
  height: 30px;
}

/* FIXME: shouldn't the vis-close button be a child of the vis-manipulation div? */

div.vis-network div.vis-close {
  position:absolute;
```

GITHUB: (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de ordenador.

VISUAL CODE: Es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias.

METODOLOGÍA DE DESARROLLO

Estudiamos "como programar en html " desde 0 luego "como programar javascript" desde 0 y al final "como programar css desde cero". Unimos esos 3 conocimientos pero luego la duda era como generar algo visual, encontramos que utilizaban la biblioteca vis, revisamos su uso y probando que funciones nos servirían para crear un grafo luego creamos un grafo interno en el mismo javascript, donde haria el proceso que nos pedía "puentes y ciclos"

Y el vis solo sirvió para mostrar el visualizado y pintar lo que convenía para mostrar los puentes y ciclos.

Diagrama de Clases

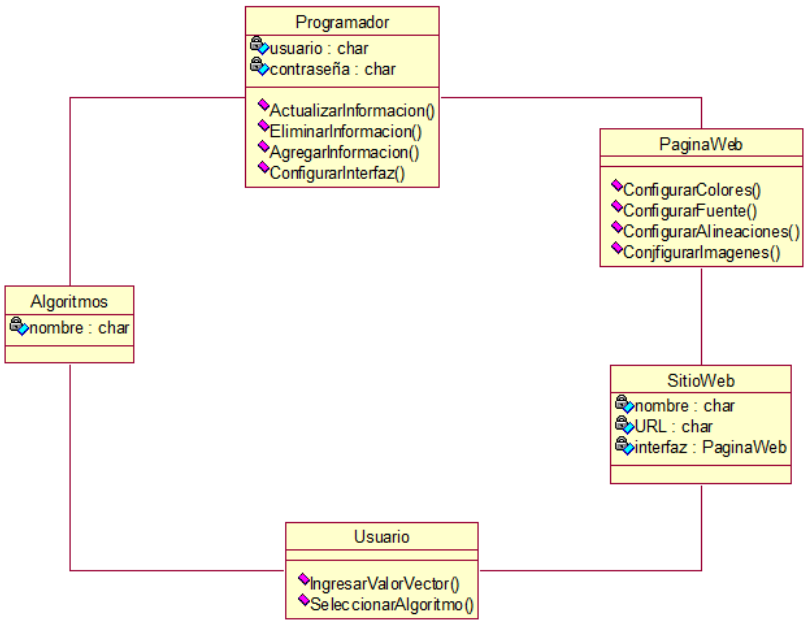


Diagrama de Secuencia

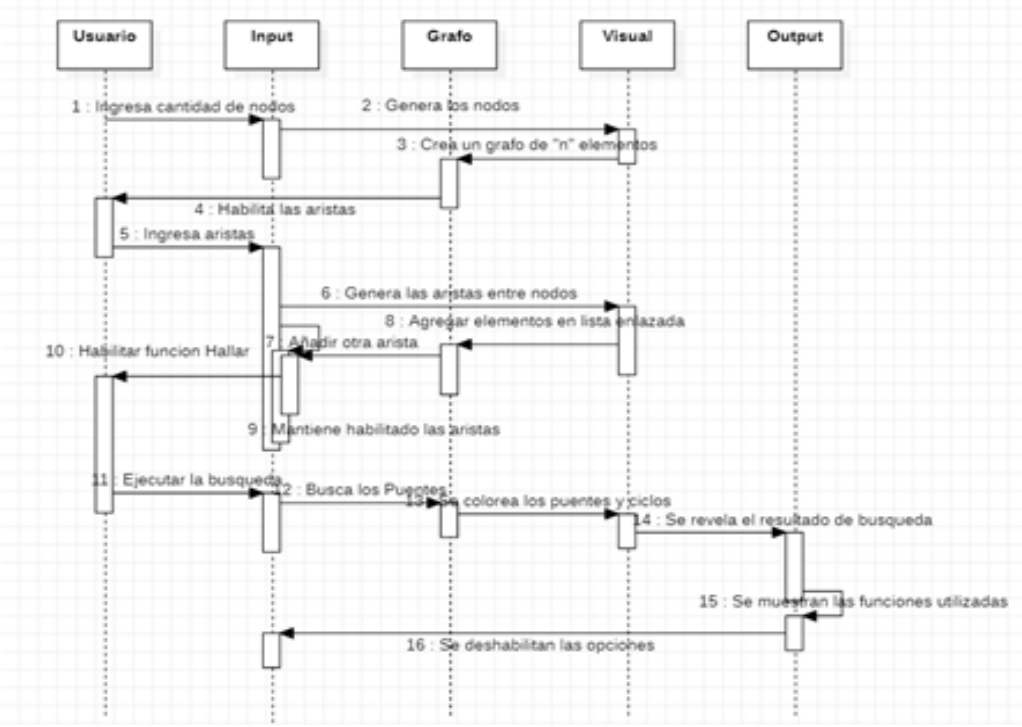
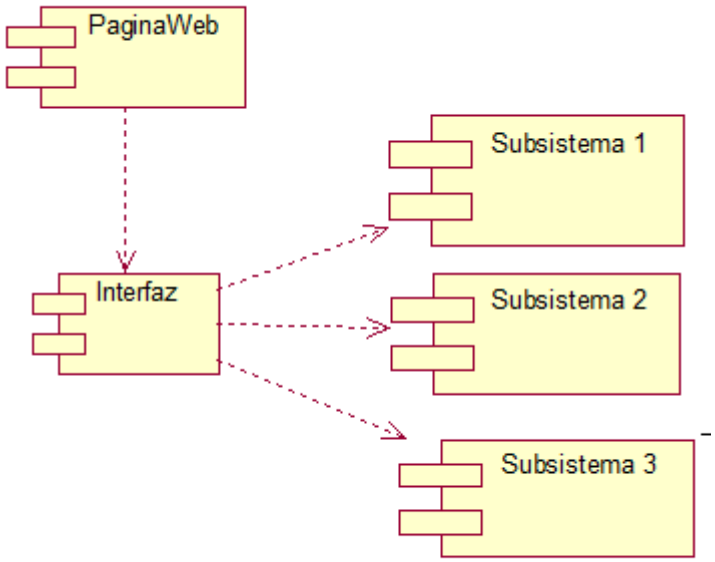
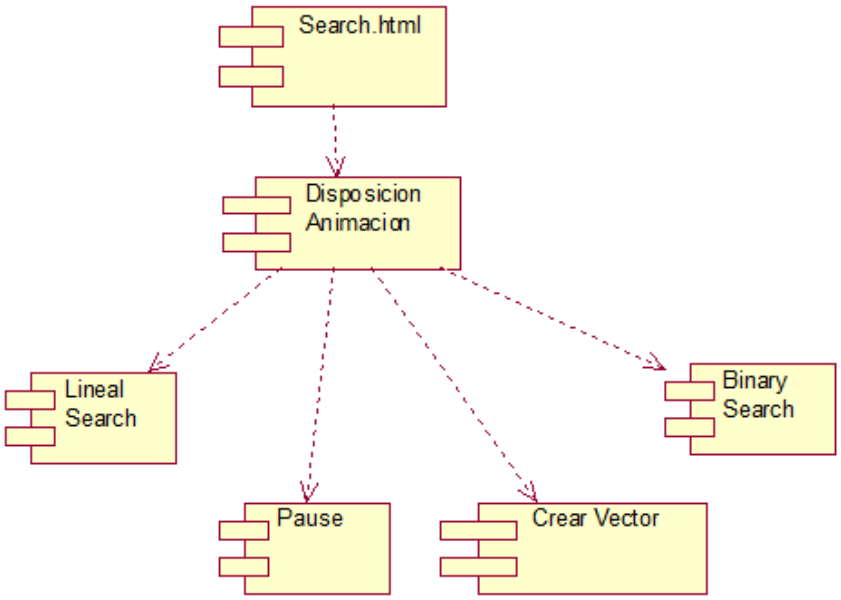


Diagrama de Componentes

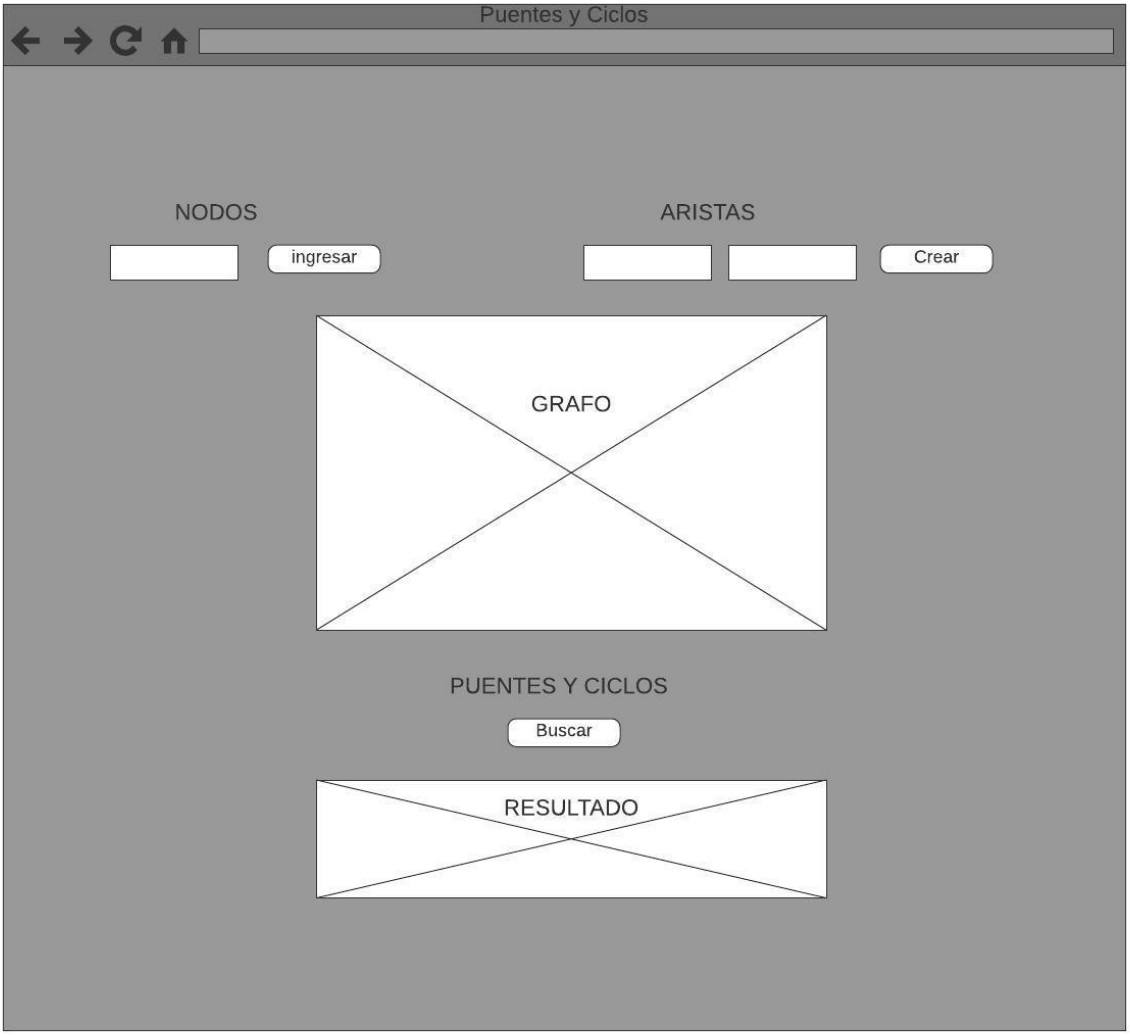




Prototipo:

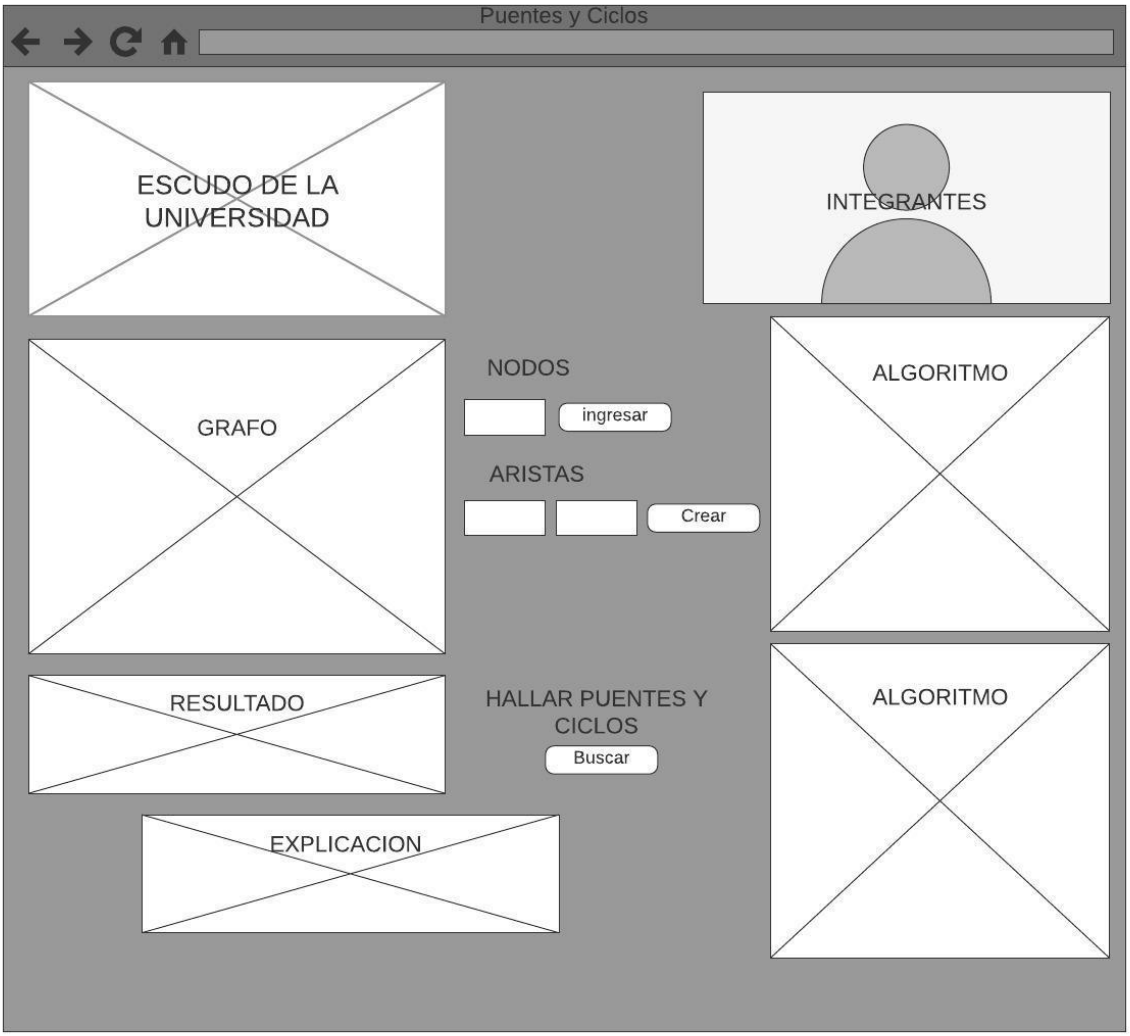
Después de encontrar la librería de la cual nos apoyaremos para hacer los grafos, decidimos optar una por contar con una caja donde se mostrará el grafo, cajas para ingresar el número de nodos y otras dos para hacer las conexiones entre estos (puentes o ciclos).

Primer Prototipo



Si bien era funcional queríamos que el usuario pudiese ver lo que ocurre detrás de nuestra aplicación mostrando los algoritmos y a su vez decidimos agregar el escudo de la universidad y a los desarrolladores (nosotros).

Prototipo Final



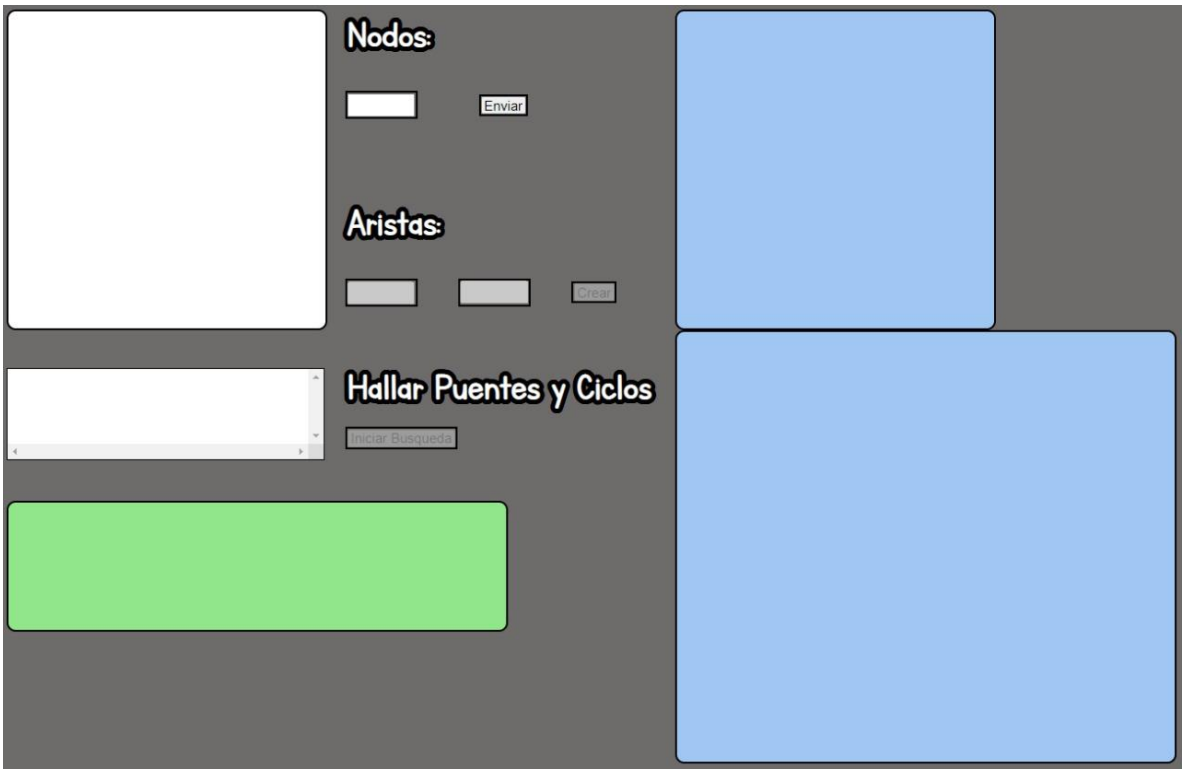
APORTE

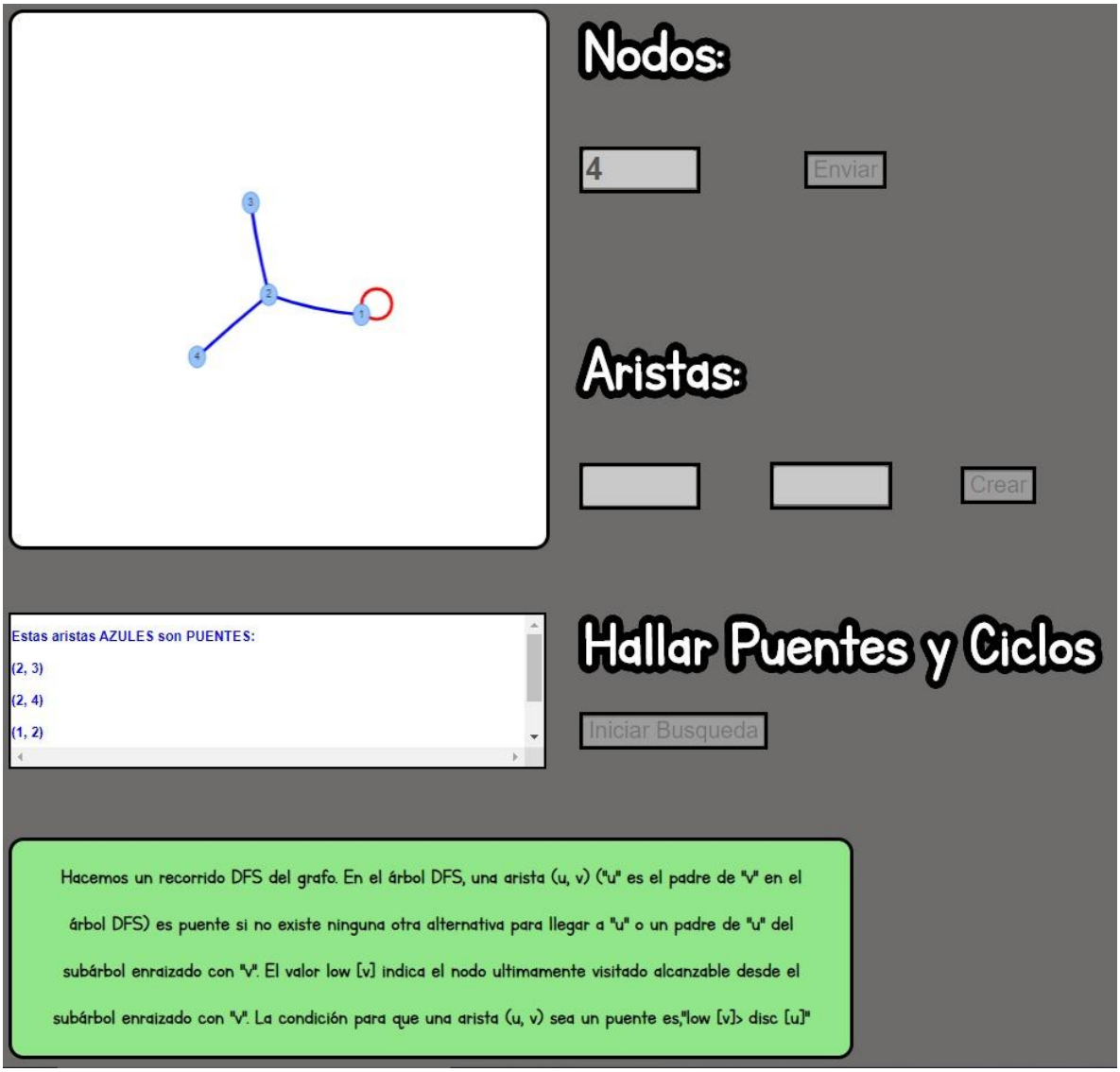
Todo el grupo participo en la elaboración del código. Tuvimos que aprender el uso de los programas y librerías.

CONCLUSIONES Y RECOMENDACIONES.

Logramos apreciar de forma visual el mecanismo, funcionamiento y eficiencia del algoritmo planteado en la cual pudimos aplicar nuestros conocimientos aprendidos en esta materia así pudimos comprender la aplicación de dicho algoritmo y ver su complejidad respectiva.

Al crear una página web mostramos el funcionamiento del algoritmo de una forma muy didáctica y al ser esta representación visual logramos que sea más comprensible. También en la creación de la página web pudimos aprender y usar herramientas como el html5, JavaScript y Css que nos ayudó para poder recrear las animaciones usadas para la comprensión del algoritmo.




Nodos:
4
Enviar
Aristas:
Crear
Estas aristas AZULES son PUENTES:
(2, 3)
(2, 4)
(1, 2)
Hallar Puentes y Ciclos
Iniciar Busqueda
Hacemos un recorrido DFS del grafo. En el árbol DFS, una arista (u, v) ("u" es el padre de "v" en el árbol DFS) es puente si no existe ninguna otra alternativa para llegar a "u" o un padre de "u" del subárbol enraizado con "v". El valor low [v] indica el nodo ultimamente visitado alcanzable desde el subárbol enraizado con "v". La condición para que una arista (u, v) sea un puente es,"low [v]> disc [u]"

```
funcion puente()
{
 // Marca todos los nodos como no visitados
 boolean visited[] = new boolean[V];
 int disc[] = new int[V];
 int disc[] = new int[V];
 int low[] = new int[V];
 int parent[] = new int[V];
 // Inicializa los padres y visitados
 for (int i = 0; i < V; i++)
 {
     parent[i] = -1;
     visited[i] = false;
 }
 // Llama la funcion recursiva para encontrar puentes
 // en el arbol DFS enraizado en el nodo "i"
 for (int i = 0; i < V; i++)
     if (visited[i] == false)
         puenteUtil(i, visited, disc, low, parent);
}
```

```
funcion puenteUtil(int u, boolean visited[], int disc[], int low[], int parent[])
{
 visited[u] = true;  //Marca el nodo actual como visitado
 disc[u] = low[u] = ++time;  //Inicializa las veces visitadas y el valor de low (time comienza como cero)
 Iterator i = adj[u].iterator();
 //Va a traves de todos los nodos adyacentes
 while (i.hasNext())
 {
     int v = i.next();  //v es el siguiente de u
     // Si v no ha sido visitado aun, se convierte en nodo hijo
     //de u en el arbol DFS y se hace la funcion recursiva para este mismo
     //Si v no es visitado aun, se hace la funcion recursiva asi mismo automaticamente.
     if (!visited[v])
     {
         parent[v] = u;
         puenteUtil(v, visited, disc, low, parent);
         //Verifica si el subarbol enraizado con v tiene una
         //conexion con uno de los antecesores de u
         low[u] = Math.min(low[u], low[v]);
         //Si el no más bajo accesible desde el subárbol
         //debajo de v está debajo de u en el árbol DFS
         //entonces u-v es un puente
         if (low[v] > disc[u])
             Print u; Print v  //Arista (u,v) como puente
     {
     // Actualiza el valor de low en u para que la llame a la funcion del padre
     else if (v != parent[u])
         low[u] = Math.min(low[u], disc[v]);
 } }
```

REFERENCIAS

Béla Bollobás, Modern graph theory, GTM 184, Springer Verlag, 1998. Página 6.

Steven Halim and Felix Halim. Competitive Programming in National University ofSingapore. InA new learning paradigm: competition supported by technology. EdicionesSello Editorial S.L., 2010.

Varios. (2012). Algorithms and More. 2012, de Blog about programming Sitio web: https://jariasf.wordpress.com/2012/03/02/algoritmo-de-busqueda-depth-first-search-parte-1/