MANUAL TÉCNICO

UNIVERSIDAD
NACIONAL
DE COLOMBIA

PROYECTO FINAL

ANÁLISIS Y DISEÑO DE ALGORITMOS

JUAN GONZÁLEZ NAVARRO

SANTIAGO LÓPEZ FLOR

UNIVERSIDAD NACIONAL DE COLOMBIA

FACULTAD DE ADMINISTRACION

ADMINISTRACION DE SISTEMAS INORMATICOS

MANIZALES 2022

CONTENIDO

I. Introducción 3
II. Objetivo del manual 4
III. Propósito del manual 4
IV. Alcance del manual 4
V. Requerimientos para el desarrollo 4
VI. Herramientas para el desarrollo 4
VII. HTML 5
- Conexiones 5
- Nav-Bar 6
- Canvas 7
- Barra de botones (btn-group) 8

VIII. JAVASCRIPT 9
- Pantalla Interactiva 9
- Funciones 12
- Pantalla Estática 14

IX. CSS 15

INTRODUCCIÓN

Con este documento se planea conocer el manejo de los grafos no dirigidos por medio de un programa diseñado con HTML, JS, y CSS, pero para ello, debemos entender que es un grafo.

En matemáticas y ciencias de la computación, un grafo (del griego grafos: dibujo, imagen)1 es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto.2Son objeto de estudio de la teoría de grafos.

Típicamente, un grafo se representa gráficamente como un conjunto de puntos (vértices o nodos) unidos por líneas (aristas o arcos).

Desde un punto de vista práctico, los grafos permiten estudiar las interrelaciones entre unidades que interactúan unas con otras. Por ejemplo, una red de computadoras puede representarse y estudiarse mediante un grafo, en el cual los vértices representan terminales y las aristas representan conexiones (las cuales, a su vez, pueden ser cables o conexiones inalámbricas).

Prácticamente cualquier problema puede representarse mediante un grafo, y su estudio trasciende a las diversas áreas de las ciencias exactas y las ciencias sociales.

Dicho esto, es importante resaltar que en este proyecto no se trabajó sobre alguna teoría de grafos en específico, ni un algoritmo especifico, se recolectó la mayor información posible por ambas partes y se intentó definir un medio por el cual un usuario cualquiera pudiese plasmar o entender el funcionamiento de las particiones sobre los grafos.

Para este proyecto, se necesitaron alrededor de unas 2 semanas para la construcción de interfaz, y al menos otras dos semanas para la labor investigativa donde se recopiló la información necesaria para proceder con la realización, diseño, y esquematización de los grafos y teorías aplicadas en el curso “Análisis y diseño de algoritmos” orientado por la docente Luz Enith Guerrero Mendietai.

Objetivo del manual

Instruir a un usuario avanzado sobre el uso general y correcto del sistema presentado en el documento y dar pie para posibles mejores o correcciones sobre el mismo.

Propósito del manual

Dar a conocer a los próximos desarrolladores los procedimientos para configurar/tratar/programar y manipular el sistema por el cual se pueden desarrollar los grafos no dirigidos

Alcance del manual

Con este manual se pretende brindar una herramienta para instruir a futuros desarrolladores e incentivarlos en su proceso de conocimiento con una pequeña página la cual les permita poner en práctica sus conocimientos sobre diseño y creación de paginas web, asociado a su capacidad de lógica y seguimiento de algoritmos.

Requerimientos para el desarrollo

1. Conexión a internet o red WIFI
2. Contar con algún sistema operativo de los que se enuncian en el siguiente documento
    - Link de revisión: https://docs.citrix.com/es-es/citrix-workspace-app-for-html5/system-requirements.html

Herramientas para el desarrollo

1. Visual studio code VERSIÓN 1.68
    - Link de descarga: https://code.visualstudio.com/download
    - Link de apoyo para instalación: https://www.mclibre.org/consultar/informatica/lecciones/vsc-instalacion.html

HTML

En este caso, se evidenciará cada elemento que compone a la parte visual o HTML del código, con una leve descripción de donde se ubica este y para que sirve. Empecemos:

- **Conexiones:** Es importante conectar nuestro HTML con nuestra hoja de estilos css y nuestro java script, lo que nos permitirá posteriormente realizar tareas de forma mas ordenada y mejor detallada

  Para comunicar HTML con CSS usamos:

```
<link rel="stylesheet" href="estilos.css">
```

  Es importante que en el href, “estilos.css”, estilos si sea el nombre del archivo css que creamos, por ejemplo, “hojitadeestilos.css”.

  Para comunicar HTML con Java Script usamos:

```
<script src="codigo.js"></script>
```

  Al final de nuestro script, justo en la línea anterior a nuestra etiqueta de cierre body, se recomienda invocar al script de js con el código anterior, donde src hace referencia al documento js llamado “código.js”, igual que con css, es importante respetar el nombre entre las comillas y la extensión.

  Para importar los estilos de Bootstrap:

```
<!-- CSS only -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
0evHe/X+R7YkIZDRvuzKMRqM+OrBnVFBL6DOitfPri4tjfHxaWutUpFmBp4vmVor"
crossorigin="anonymous">
    <!-- JavaScript Bundle with Popper -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/js/bootstrap.bundle.min.js" integrity="sha384-
pprn3073KE6tl6bjs2QrFaJGz5/SUsLqktiwsUTF55Jfv3qYSDhgCecCxMW52nD2"
crossorigin="anonymous"></script>
```

  En este caso, Bootstrap proporciona estas dos líneas de código para poder trabajar con sus estilos y herramientas de manera óptima. Estos se obtienen de la página principal de Bootstrap en su documentación inicial.

- **Nav-bar:**

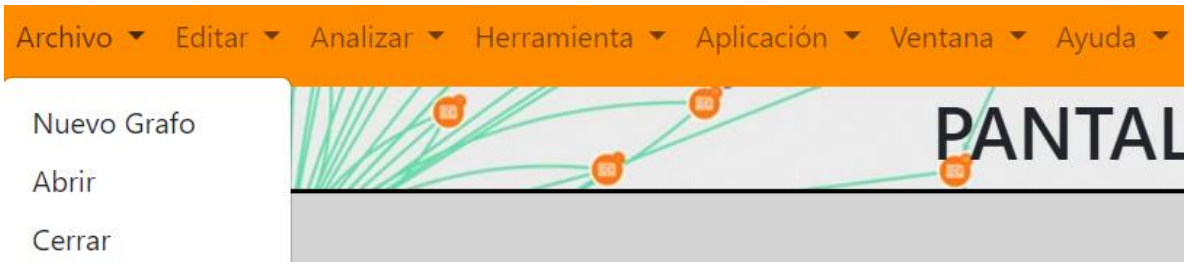
```
<div class="collapse navbar-collapse" id="navbarSupportedContent">
  <ul class="navbar-nav me-auto mb-2 mb-lg-0">
      <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
            Archivo
          </a>
          <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
            <li><a class="dropdown-item" href="#">Nuevo Grafo</a></li>
            <li><a class="dropdown-item" href="#">Abrir</a></li>
            <!-- <li><hr class="dropdown-divider"></li> -->
            <li><a class="dropdown-item" href="#">Cerrar</a></li>
            <li><a class="dropdown-item" href="#">Guardar</a></li>
            <li><a class="dropdown-item" href="#">Guardar como</a></li>
            <li><a class="dropdown-item" id="Exportarbtn">Exporttar datos</a></li>
            <li><a class="dropdown-item" href="#">Importar Datos</a></li>
            <li><a class="dropdown-item" href="#">Inicio</a></li>
            <li><a class="dropdown-item" href="#">Imprimir</a></li>
          </ul>
      </li>
```

Podemos ver que el navbar se importa de Bootstrap ya que muchas de sus clases vienen de este framework, sin embargo, este fue editado para agregar campos dentro de su botón “dropdown-menu”, el cual contiene varias funcionalidades, las cuales se renombran en su clase como dropdown-item y se les asigna un href para que cumplan la función de redireccionarnos a algún otro index o en general, cumplan alguna función dentro del ejercicio propuesto.

Cada item está debidamente contenido en una etiqueta li, la cual hace referencia a lista, lo que nos permite lograr que cuando al botón de nombre “Archivo” el cual se clasifica como toggle (item que saldrá como titular en la barra de contenidos), este, nos muestre una lista de funcionalidades desplegada con elementos como “Nuevo grafo”, “Abrir”, ”Cerrar”, ”Guardar”, etc…

Es importante resaltar que a cada uno de estos botones puede asignárseles un id para poder trabajar con ellos posteriormente con javascript y sus eventos, o bien para cambiar su diseño especifico desde la hoja de estilos en css, muestra de esto es el botón “Exportar” el cual tiene id propio.

En la interfaz se visualizaría así:



Es de resaltar que en el código fuente, estos botones aún se conservan vírgenes, pues ninguno de estos se encuentra asociado a una rutina en particular, por ende, lo único que está cambiando son los nombres y los campos dentro de los mismos, como en este caso, editar:

```
<!-- EDITAR -->
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
    Editar
  </a>
  <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
    <li><a class="dropdown-item" href="#">Deshacer</a></li>
    <li><a class="dropdown-item" href="#">Nodo</a></li>
    <li><a class="dropdown-item" href="#">Arco</a></li>
  </ul>
</li>
```

Así sucesivamente hasta el botón de “Ayuda” se sigue el mismo procedimiento para la creación de los campos.

Por último, el botón buscar el cual se puede asociar a una base de datos la cual por medio del botón permita hacer consultas sql para la búsqueda interna de posibles grafos predefinidos los cuales puedan ser usados dentro del algoritmo

```
<!-- BUSCAR -->
<form class="d-flex" role="search">
  <input class="form-control me-2" type="search" placeholder="Buscar" aria-label="Search">
  <button class="btn btn-outline-success" type="submit">Buscar</button>
</form>
<!-- FIN BUSCAR -->
```

Este se encuentra aún sin subrutinas asignadas y se importó tal cual de Bootstrap.

- **Canvas (Pantallas):**

```
<!-- PANTALLA DE GRAFOS INTERACTIVA -->
    <center><h1>PANTALLA PARA DIBUJAR</h1></center>
    <center><canvas id="PantallaGrafos" width="1200" height="600" ></canvas></center>
<!-- FIN INTERACTIVA -->
```

La etiqueta canvas que nos proporciona HTML la usamos en este proyecto para trabajar sobre ella los grafos diseñados por el usuario he incluso los que predefinimos nosotros para el fácil entendimiento del algoritmo para el usuario.

En la línea 114 contamos con una etiqueta center la cual nos permite ubicar el elemento que tengamos entre esta etiqueta, justo en el centro de la pantalla en el eje X, posteriormente vemos una etiqueta h1 la cual se utiliza para encabezados.

En la línea 115 invocamos a la etiqueta canvas para definir una pantalla la cual cuenta con un id con el que posteriormente nos referimos a la pantalla para poder

dibujar sobre ella desde java script (js). Seguido a esto, encontramos que definimos las dimensiones que queremos que tenga nuestra pantalla en cuestión, siendo “widht” el ancho y “height” el alto de la pantalla. Los números a los cuales apuntan estas condiciones son el ancho y alto en pixeles. El canvas en pantalla:

- **Barra de botones:**

```
<!-- TABLA UTIL -->
<div class="btn-group" role="group" aria-label="Basic example">

  <button id="refresh" type="button" class="btn btn-primary" href="/index.html">Limpiar</button>
  <button id="particionbtn" type="button" class="btn btn-primary">Partición</button>
  <button id="dibujarbtn" type="button" class="btn btn-primary">Dibujar</button>
</div>
<!-- FIN TABLA UTIL -->
```

La barra de botones o en el código llamada “Tabla útil”, hace referencia a nuestros tres botones principales para nuestra pantalla interactiva, Limpiar, Partición, y Dibujar.

Estos botones se encuentran contenidos en un div (etiqueta que nos sirve para contener varios botones de la misma clase si asi lo deseamos). Cada botón cuenta con un id único, pues esto nos permite asignar tareas diferentes a cada botón desde el java script o darles un diseño diferente desde nuestra hoja de estilos, y , en el caso de Limpiar, vemos que este cuenta con un atributo href el cual está apuntando hacia una ruta index, esto es para que cuando el usuario toque el botón de limpiar, href, llame a la página misma index, haciendo que esta se recargue para borrar los datos de la pantalla.

JAVA SCRIPT (JS)

En este apartado, se evidenciará el funcionamiento de la página por medio de Java Script con sus correspondientes eventos y funciones para el tratamiento de información.

```
//INICIALIZAR VARIABLES Y ARREGLOS
let nodes = [];
let selectedNode = null;
let arcos = [];
```

Lo mas importante para un programa es definir las variables de entorno con las cuales se trabajará, en este caso, eso es lo que hacemos con estas líneas de código, inicializamos dos arreglos llamados nodes y arcos y una variable selectedNode en null la cual nos permitirá mas adelante hacer ciertas verificaciones

- ***Pantalla interactiva:***

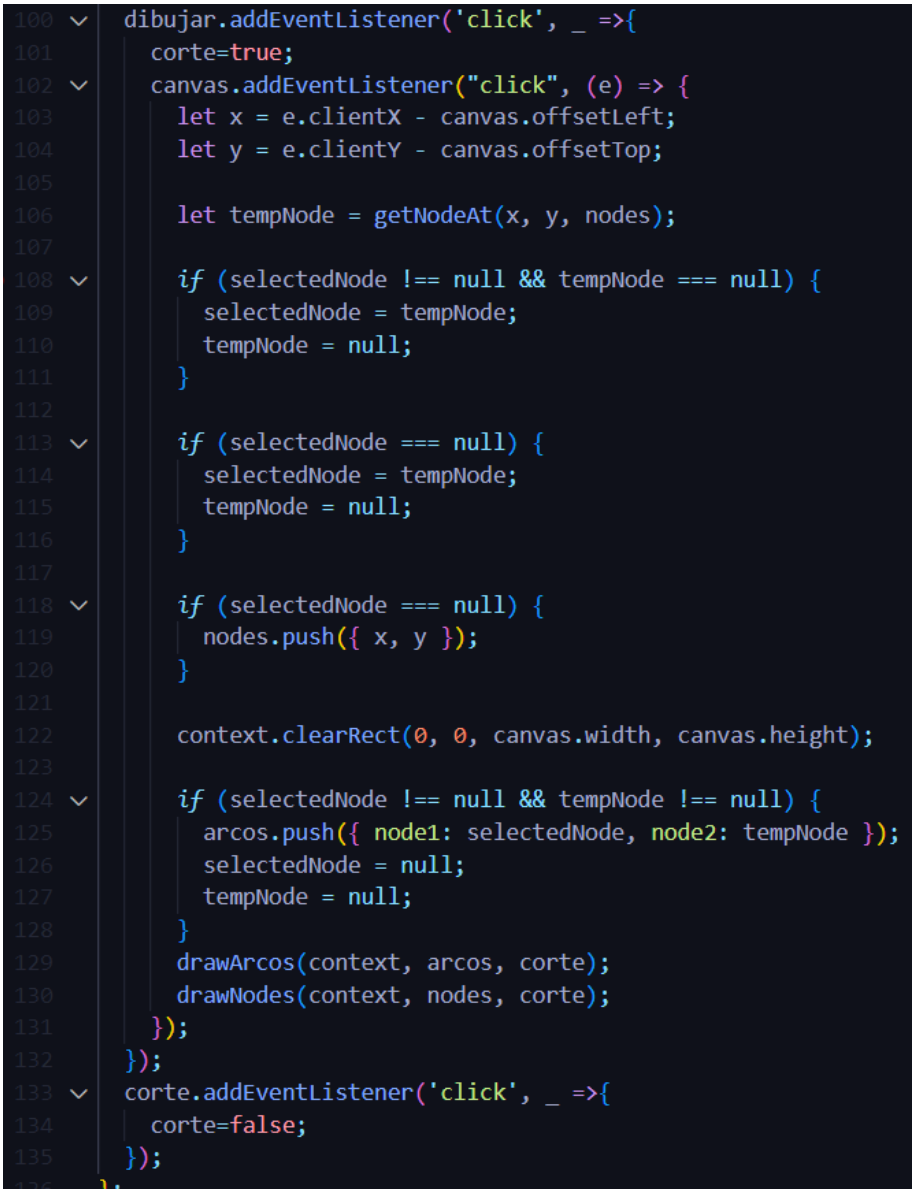
```
// PANTALLA INTERACTIVA
window.onload = async () => {
  var canvas = document.getElementById("PantallaGrafos");
  var context = canvas.getContext("2d");
  var corte = document.getElementById("particionbtn");
  var dibujar = document.getElementById('dibujarbtn');

  dibujar.addEventListener('click', _ =>{ ···
  });
  corte.addEventListener('click', _ =>{
    corte=false;
  });
};
```

En esta primera parte, nos encontramos con una instrucción del js la cual nos permite reconocer cuando nuestra pantalla está cargada y encendida, seguido a esto, en cada variable (var), almacenaremos una serie de datos, sin embargo, para

trabajar en nuestra pantalla ya interactiva canvas, es importante que obtengamos su información por medio de su id, como lo hace el método getElementById, y lo mostrará en el document, esto se almacenará en una variable canvas de la cual obtendremos su contexto en un formato 2d, esto es importante tenerlo claro pues es el método que usaremos para trabajar nuestros gracios. Entendido esto, obtenemos los valores de los botones partición y dibujar para poder operar y añadir funciones sobre los mismos.

En la línea 100, lo que se está haciendo es agregar un evento al botón dibujar, por eso, se coloca el addEventListener, para que este comando nos permita entender que cuando al botón dibujar, lo pongan en estado 'click', este mismo ejecutará la serie de operaciones que se le asignen dentro de su listener, como las siguientes.



```
100  dibujar.addEventListener('click', _ =>{
101    corte=true;
102    canvas.addEventListener("click", (e) => {
103      let x = e.clientX - canvas.offsetLeft;
104      let y = e.clientY - canvas.offsetTop;
105
106      let tempNode = getNodeAt(x, y, nodes);
107
108      if (selectedNode !== null && tempNode === null) {
109        selectedNode = tempNode;
110        tempNode = null;
111      }
112
113      if (selectedNode === null) {
114        selectedNode = tempNode;
115        tempNode = null;
116      }
117
118      if (selectedNode === null) {
119        nodes.push({ x, y });
120      }
121
122      context.clearRect(0, 0, canvas.width, canvas.height);
123
124      if (selectedNode !== null && tempNode !== null) {
125        arcos.push({ node1: selectedNode, node2: tempNode });
126        selectedNode = null;
127        tempNode = null;
128      }
129      drawArcos(context, arcos, corte);
130      drawNodes(context, nodes, corte);
131    });
132  });
133  corte.addEventListener('click', _ =>{
134    corte=false;
135  });
```

Lo primero con lo que nos encontramos en esta función es con una nueva variable llamada corte la cual definimos en true. Esto lo hacemos para que cuando se llame al evento dibujar, esta variable booleana se considere verdadera y nos permita hacer una serie de pasos, pero, si ese estado de la variable pasa a ser un estado en false, esto nos indicará una serie de pasos diferentes los cuales veremos mas adelante.

Nos encontramos con otro listener, lo que nos indica que aquí se llevará a cabo una función o serie de funciones sobre la pantalla canvas, iniciando por definir los puntos X y Y de los nodos a trabajar, y como se van a mover, pues recordemos que en el protocolo para dibujar sobre HTML, si queremos desplazar un dibujo, este se mueve con las coordenadas left y top

Cuando en la línea 106 llamamos a getNodeAt, lo que buscamos hacer es una validación de los nodos mediante la siguiente función getNodeAt la cual explicaremos al final.

Posterior a ello encontramos una serie de validaciones para los vértices, las cuales consisten en verificar diferentes casos sobre si nuestro nodo seleccionado (selectedNode), y nuestro segundo nodo seleccionado ósea el de llegada (tempNode), no se encuentren ya previamente seleccionados o no se sobre escriban llamándose a si mismos y generando aristas sobre ellos.

El siguiente if de la línea 124 es para hacer referencia al caso de cuando ya seleccionamos un nodo de salida y un nodo de llegada, es decir, que selectedNode y tempNode son diferentes de null, entonces trazaremos una arista.

En las líneas 129 y 130 lo que hacemos es, que como ya verificamos que ambos nodos estén seleccionados, se deba trazar una línea entre los ejes de estos mismos, para esto llamamos a la función drawArcos y drawNodes las cuales se explicarán al final

Por ultimo en esta ventana de la pantalla interactiva, lo que hacemos es asignar una acción al botón partición el cual fue renombrado por la variable corte, la cual, si llamamos, vemos que nos cambia el estado de corte, a false, lo cual le hará saber al algoritmo que como el estado de corte es false, entonces se encuentra en partición.

FUNCIONES

- **drawNodes:** Esta función se encarta de recibir los nodos y la variable corte para verificar y señalar en que método está, si corte es true, entonces procederá a resaltar los vértices de color verde indicando que no se está haciendo ninguna partición, sin embargo, si el estado es false, los vértices se resaltarán en color rojo para decirle al usuario que se encuentra haciendo una partición

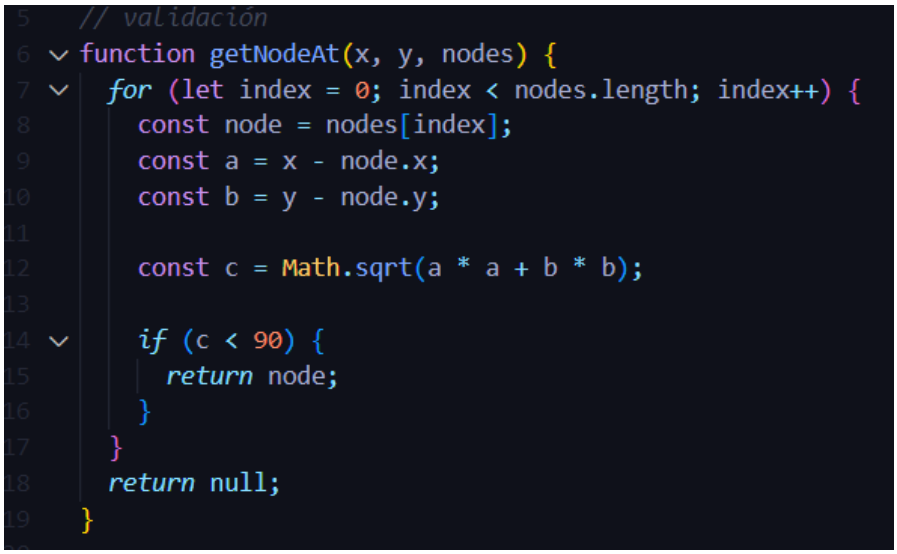
```
21  //función para dibujar los vertices
22  function drawNodes(ctx, nodes, corte) {
23    for (let index = 1; index < nodes.length; index++) {
24      const node = nodes[index];
25    //condición de if para que cuando partición sea invo
26    if (corte == false){
27        if (node === selectedNode) {
28          ctx.strokeStyle = "#FF0000";
29        } else {
30          ctx.strokeStyle = "#000000";
31        }
32
33        ctx.beginPath();
34        ctx.lineWidth = 4;
35        ctx.fillStyle = "#FFFFFF";
36        ctx.arc(node.x, node.y, 35, 0, 2 * Math.PI);
37        ctx.stroke();
38        ctx.fill();
39
40        if (node === selectedNode) {
41          ctx.fillStyle = "#FF0000";
42        } else {
43          ctx.fillStyle = "#000000";
44        }
45
46        ctx.font = "30px Arial";
47        ctx.fillText(index, node.x - 5, node.y + 5);
48    // de lo contrario, seguirá su proceso normal de dir
```

Corte == False

```
49        }else{
50          if (node === selectedNode) {
51            ctx.strokeStyle = "#008000";
52          } else {
53            ctx.strokeStyle = "#000000";
54          }
55
56          ctx.beginPath();
57          ctx.lineWidth = 4;
58          ctx.fillStyle = "#FFFFFF";
59          ctx.arc(node.x, node.y, 35, 0, 2 * Math.PI);
60          ctx.stroke();
61          ctx.fill();
62
63          if (node === selectedNode) {
64            ctx.fillStyle = "#008000";
65          } else {
66            ctx.fillStyle = "#000000";
67          }
68
69          ctx.font = "30px Arial";
70          ctx.fillText(index, node.x - 5, node.y + 5);
71        }
72      }
73    }
```

Corte == True

- **getNodeAt:** Esta función de lo que se encarga es de verificar que un vértice no se dibuje sobre otro, y que, por el contrario, guarden cierta mínima distancia entre ellos para poder ver mejor el resultado de las pruebas de la pantalla interactiva



```
// validación
function getNodeAt(x, y, nodes) {
  for (let index = 0; index < nodes.length; index++) {
    const node = nodes[index];
    const a = x - node.x;
    const b = y - node.y;

    const c = Math.sqrt(a * a + b * b);

    if (c < 90) {
      return node;
    }
  }
  return null;
}
```

- **drawArcos:** Esta función se encarga de dibujar las aristas para la pantalla interactiva, como se puede ver, hay una parte comentada, esto es debido a que en esta parte se intentó validar que cuando se invocara a partición, el programa debería permitir rayar sobre una arista la cual escoja el usuario, y eleminarla.

```
//función para dibujar las aristas
function drawArcos(ctx, arcos,corte) {
  // if(corte == false){//Se busca que con esta
  //   ctx.moveTo(node1.x, node1.y);
  //   ctx.lineTo(node2.x, node2.y);
  //   ctx.strokeStyle = "#D3D3D3";
  //   ctx.stroke();
  // }else{ // de lo contrario, seguirá su proc
    for (let i = 0; i < arcos.length; i++) {
      const arco = arcos[i];
      ctx.moveTo(arco.node1.x, arco.node1.y);
      ctx.lineTo(arco.node2.x, arco.node2.y);
      ctx.strokeStyle = "#000000";
      ctx.stroke();
    }
  }
```

- ***Pantalla Estática:*** En esta pantalla, lo que se hizo fue pre definir un grafo por medio de las herramientas que se brindaban en la etiqueta canvas para el dibujo de círculos, líneas y texto.

```
// PANTALLA INTERACTIVA
window.onload = async () => {···
};
// PANTALLA ESTÁTICA
window.addEventListener('load', () => {
  var canvas2 = document.getElementById("PantallaGrafos2");
  var ctx = canvas2.getContext("2d");
  var aplicar = document.getElementById("Aplicarbtn");
  aplicar.addEventListener('click', _ =>{
      ctx.font = "30px Arial";
      ctx.fillText("Grafo Total", 500, 80);
      ////////////////////////////////////// LINEAS
    // LINEA 1 A 2
      ctx.moveTo(96, 95);
      ctx.lineTo(96, 450);
      ctx.stroke();
    //
```

Para este acceder a este evento de la segunda pantalla, debemos clickear el botón de aplicar, y este por medio de un listener, procederá a dibujar un grafo ya pre definido

Para dibujar líneas: El moveTo hace referencia al punto inicial de la línea y el lineTo hacia el punto final, ambos tienen dos coordenadas de puntos que son X y Y respectivamente	`ctx.moveTo(96, 95);` `ctx.lineTo(96, 450);`

Para dibujar Círculos: El arc hace referencia a la dimensión del círculo donde sus primeros dos números son sus coordenadas en X y Y, y su tercer valor es su diámetro.	`ctx.beginPath();`
	`ctx.arc(95, 450, 35, 0, 2 * Math.PI);`
	`ctx.stroke();`

Para Escribir: Se selecciona el tamaño de la fuente, y posterior a ello, que se quiere escribir. Por último, sus coordenadas en X y Y.	`ctx.font = "30px Arial";` `ctx.fillText("Grafo Total", 500, 80);`

CSS

En esta hoja de estilos solo se hizo referencia a los elementos del HTML obteniéndolos por su id y modificando algunos colores de si mismos, también, se agregó un fondo para el proyecto como se ve al inicio de la hoja

```
You, hace 1 segundo | 1 author (You)
body {
    background: url("fondo.jpg");
  }
#PantallaGrafos{
    background-color: lightgray;
    border: 3px solid black;
}
#PantallaGrafos2{
    background-color: lightgray;
    border: 3px solid black;
}
#Nav-Bar{
    background-color: darkorange;
}
#barra-botones{
    background-color: darkorange;
}
#refresh{
    background-color: darkorange;
}
#particionbtn{
    background-color: darkorange;
}
#dibujarbtn{
    background-color: darkorange;
}
#Aplicarbtn{
    background-color: darkorange;
}
#particionbtn2{
    background-color: darkorange;
}
#refresh2{
    background-color: darkorange;
}
```