UNIVERSIDAD MAYOR DE SAN MARCOS

Universidad del Perú, DECANA DE AMÉRICA.

PROYECTO FINAL DE CURSO

Curso: Estructura de Datos

Profesor: Herminio Paucar Cuaresma

Grupo: 5

Integrantes:

Cortez Rosas, Ingrid Fiorella(18200311)-Participación [3]

Gonzales Julluni, Alexandra(18200081)-Participación [3]

Quinteros Peralta, Rodrigo Ervin(18200316)-Participación [3]

Tirado Julca, Juan Jose(18200117)-Participación [3]

Lima – 2020

Índice

1. Introducción:

Como estudiantes de Ingeniería de Software debemos estar preparados para enfrentar diariamente la resolución de problemas, además del análisis de soluciones para encontrar la más óptima. Si tuviésemos el caso de una red de comunicación, vemos que no es necesario que toda estación pueda comunicarse directamente con otra, puesto que hay estaciones que pueden actuar como medio para trasladar el mensaje entre una y otra estación. Si una estación dejase de funcionar, ¿todas las demás podrán seguir comunicándose entre sí?.

Para resolver esta y otras cuestiones surge la noción matemática del grafo, que no son más que unos nodos con algunas conexiones llamadas aristas.

De estos conceptos nace la teoría de grafos que tiene varias de aplicaciones en distintos campos por ejemplo resuelve la síntesis de circuitos secuenciales, es utilizada también para modelar trayectos de líneas de transporte.

En la informática se ha utilizado para la resolución de importantes y complejos algoritmos como ejemplos tenemos al Algoritmo de Dijkstra en la determinación del camino más corto en el recorrido de un grafo, o el Algoritmo de Kruskal que nos permite buscar un subconjunto de aristas que incluye todos los vértices, estableciendo como mínimo el valor de las aristas.

Tiene aplicaciones en las ciencias sociales, para desarrollar el concepto de red social representando de manera abstracta y gráfica las relaciones entre los actores.

Se usa para la solución de problemas de genética y problemas de automatización de la proyección (SAPR). Apoyo matemático de los sistemas modernos para el procesamiento de la información.

Allí radica la importancia del conocimiento de algoritmos de recorrido de nodos.

En el presente proyecto se realizó una aplicación web en la que se puede observar el funcionamiento de los dos principales algoritmos de búsquedas en nodos conectados, los cuales son búsqueda en profundidad DFS (Depth First Search) y búsqueda en anchura BFS (Breadth First Search).

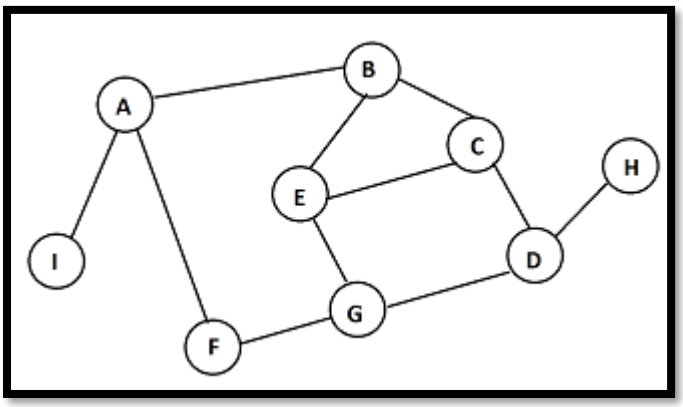
2. Marco Teórico:

2.1. Búsqueda en anchura (BFS)

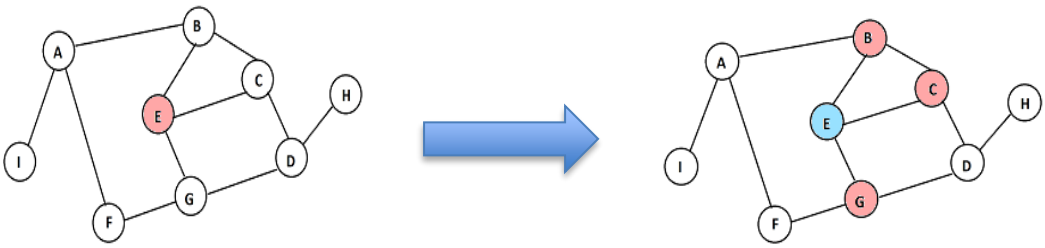
BFS es una técnica para recorrer un grafo, a partir de un nodo origen.Se trata de ir expandiendo la exploración de los nodos del grafo, en todas las direcciones a la vez. De esta forma, el grafo se irá recorriendo en orden de distancia al nodo origen, formándose una suerte de onda expansiva a su alrededor.

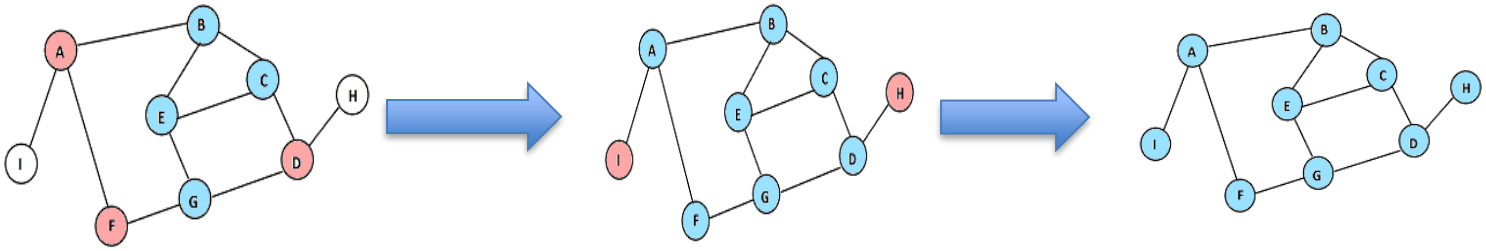
Ejemplo

Supongamos que tenemos el siguiente gráfico:



A continuación, se muestra el orden en que el algoritmo de BFS irá descubriendo los nodos del grafo, si se comienza una exploración desde el nodo E:





En cada paso, se muestran en rosa los nodos recién descubiertos, y en celeste los que ya habían sido descubiertos en pasos anteriores.

Observando estos ejemplos, es posible caracterizar el mecanismo fundamental del algoritmo de BFS de la siguiente manera:
Se comienza inicialmente con el nodo origen, como único nodo rosa recién descubierto.
La exploración se expande, desde los nodos rosa recién descubiertos, descubriendo nuevos nodos que no habían sido visitados antes. Es decir, todos los nodos blancos (que nunca habían sido vistos) que resultan ser vecinos de los nodos rosa, serán los rosa del próximo paso.
El paso anterior se repite una y otra vez descubriendo más y más nodos en orden, hasta que en algún paso ya no se descubran nodos nuevos. Es decir, se termina el algoritmo cuando ya no hay ningún nodo rosa.

Complejidad

Suponiendo que explorar los vecinos de un nodo tiene un costo proporcional a su grado (por ejemplo, como ocurre al usar listas de adyacencia), todas estas versiones del algoritmo tienen una complejidad(n+m).
Esto es porque cada nodo es procesado cuando es rosa, y un nodo es rosa durante únicamente un paso. Por lo tanto, se trabaja con un nodo explorando sus vecinos una única vez en todo el algoritmo. Si explorar los vecinos de un nodo cuesta proporcional a su grado, el costo total del algoritmo, además de un O(n) fijo para crear el arreglo de visitados y similares estructuras, tendrá un costo igual a la suma de todos los grados, que es 2m. Por lo tanto el total es O(n+m), proporcional al total de nodos y ejes del grafo, lo cual es muy bueno.

Si se utilizara matriz de adyacencia, el costo de buscar los vecinos de un nodo sería siempre O(n), sin importar su grado. En este caso, el costo de explorar cada uno de los n nodos sería siempre nn, y en total el costo sería O(n2). Muchas veces es perfectamente razonable un costo de O(n2), y en estos casos se puede utilizar perfectamente BFS con matriz de adyacencia.

2.2. Búsqueda en profundidad (DFS)

Es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo, recorrer un grafo consiste en visitar cada uno de los nodos a través de las aristas del mismo. Se trata de realizar recorridos de grafos de manera eficiente. Para ello, se pondrá una marca en un nodo en el momento en que es visitado, de tal manera que inicialmente no estaban marcados ninguno de los nodos del grafo.

El proceso para grafos no dirigidos es análogo al proceso para el caso de grafos dirigidos, solo cambia el significado del concepto de adyacencia. A continuación, describimos la idea de un procedimiento recursivo para realizar un recorrido en profundidad de un grafo no dirigido G. Para ello, en primer lugar a cada nodo v del grafo se le asocia un procedimiento, DFS(G, v), que se denomina recorrido en profundidad de G con origen v.

El algoritmo del procedimiento DFS(G, v) puede ser descrito de la siguiente manera:

- Se marca el nodo v.
- Si todos los nodos adyacentes a v están marcados, entonces TERMINAR, sino se elige un nodo w, adyacente a v que no esté marcado.
- Se ejecuta el proceso DFS(G, w).

Si G es conexo (es decir, si dos vértices cualesquiera de G siempre están conectados por un camino), entonces DFS(G, v) visita todos los nodos y aristas del grafo.

A la hora de describir un esquema algorítmico del recorrido en profundidad, asociaremos al procedimiento dos tareas, que denominaremos pre–trabajo y post–trabajo, con el fin de conseguir una serie de aplicaciones interesantes de los recorridos de grafos.

En donde pre–trabajo (v) es una tarea que se ejecutará inmediatamente después de marcar el nodo v; y post–trabajo (v, w) es una tarea que se ejecutará una vez que la llamada recursiva DFS(G, w) termina y devuelve el control a v.

```
procedimiento DFS(G, v)
    marcar (v)
    hacer pre-trabajo (v)
    para cada vertice w adyacente a v hacer
      si w no esta marcado entonces
        DFS(G, w)
        hacer post-trabajo (v, w)
```

En la ejecución de la búsqueda por profundidad se tienen tres teoremas:

Teorema 1: Sea G = (V, E) un grafo no dirigido y conexo. Para cada v ∈ V se verifica:

1. El procedimiento DFS(G, v) para.
2. Tras la ejecución del procedimiento DFS(G, v), todos los nodos de G están marcados.

Sea G = (V, E) un grafo no dirigido arbitrario. Para realizar un recorrido en profundidad en G se procede como sigue:

1. Se elige un nodo, v, no marcado.
2. Se ejecuta el procedimiento D S(G, v).
3. Si no quedan nodos no marcados, el proceso TERMINA; si no, se vuelve al paso 1.

En consecuencia, el procedimiento de recorrido en profundidad de un grafo no dirigido arbitrario se puede describir como sigue:

```
procedimiento DFS(G)
    para cada nodo no marcado, v, hacer
        DFS(G, v)
```

Teorema 2: Sea G = (V, E) un grafo no dirigido. El coste del algoritmo DS(G) es del orden $\theta(m\acute{a}x\{|V|, |E|\})$ (bajo la hipótesis de que el pre–trabajo y el post–trabajo sean de coste constante).

La ejecución de un recorrido en profundidad de un grafo, G = (V, E), no dirigido proporciona de manera natural un bosque de ejecución. Las aristas del grafo, G, que no pertenecen a

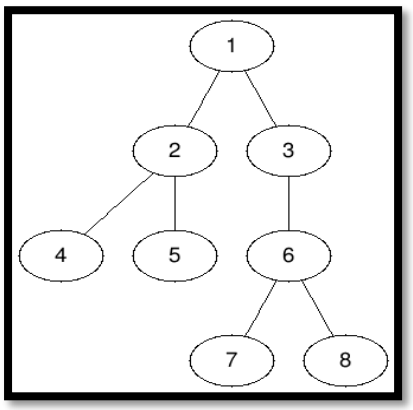
ningún árbol del bosque, se denomina arista de vuelta atrás. Si notamos el bosque del recorrido en profundidad por B = (V, F), entonces las aristas de vuelta atrás son las del conjunto E − F (que son las aristas no visitadas en dicho recorrido).

Teorema 3: Sea G = (V, E) un grafo no dirigido. Sea B un bosque de recorrido en profundidad de G. Cada arista de G, o bien es una arista de algún árbol del bosque, o bien conecta dos nodos de G tales que uno de ellos es un ascendiente del otro en B.

La búsqueda en profundidad se usa cuando queremos probar si una solución entre varias posibles cumple con ciertos requisitos.

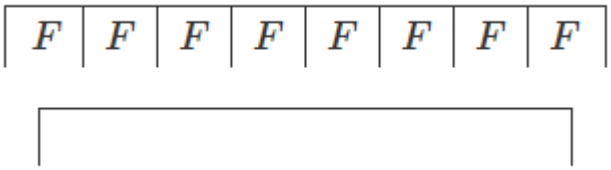
Ejemplo

Supongamos que tenemos el siguiente gráfico:

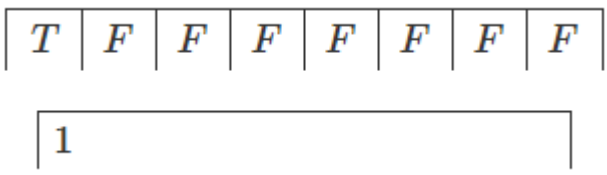


Y que lo vamos a empezar a recorrer desde el nodo 1.

Tenemos **false** en la variable visitado de cada nodo, y la lista de nodos para mirar está vacía:



Agregamos al nodo 1:



Sacamos al nodo 1 y agregamos a sus vecinos que no fueron visitados (en el orden en el que recorramos los vecinos, podría ser cualquier orden):

T	F	F	F	F	F	F	F

$2, 3$

Sacamos primero al nodo 3 y miramos sus vecinos:

T	F	T	F	F	F	F	F

$2, 6$

Sacamos al nodo 6 y miramos sus vecinos no visitados:

T	F	T	F	F	T	F	F

$2, 7, 8$

Notar que el 2 quedó, pero primero vamos a agotar los caminos hacia abajo del nodo 3

Sacamos al nodo 8 y como no tiene vecinos sin visitar, no hacemos nada (más que marcarlo como visitado al sacarlo, como a todos). Lo mismo luego con el nodo 7

T	F	T	F	F	T	T	T

2

Sacamos al nodo 2 y miramos sus vecinos no visitados:

T	T	T	F	F	T	T	T

$4, 5$

Sacamos al nodo 5 que no tiene nodos por visitar, y lo mismo con el 4 y terminamos.

Complejidad

Se ve que cada vértice se le visita a lo sumo una vez, y se hacen operaciones de tiempo constante, por lo que tenemos O(V)O(V) tiempo para los vértices. Además, cada arista se ve

también a lo sumo una vez, tomando O(E)O(E) tiempo. En total, la complejidad temporal es O(V+E)O(V+E), que en términos de grafos es óptima.

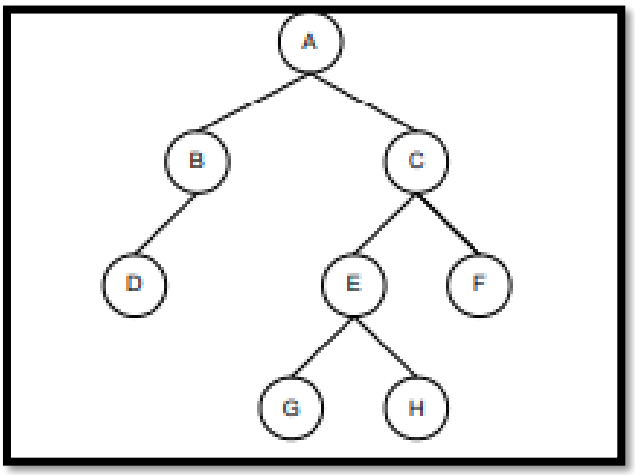
La complejidad espacial es sencilla de ver. Tenemos el grafo que guarda una entrada por arista (en realidad dos, para la arista (u,v)(u,v) guarda vv en el vector de uu y uu en el vector de vv). Luego, el arreglo de visitados tiene una variable por nodo, y en la pila (en el primer caso) a lo sumo están todos a la vez, una sola vez (si el nodo inicial es vecino de todos). Entonces la complejidad espacial también se ve que es O(V+E)O(V+E).

2.3. Grafos (Árboles)

Un árbol es una estructura de datos no lineal y homogénea en el que cada elemento puede tener varios elementos sucesores, pero sólo tiene un elemento anterior. Un árbol puede representarse de diferentes formas, como:

a) Diagrama de Venn
b) Grafos
c) Indentada

Nodo Padre: El nodo padre de un nodo N es aquel nodo que se apunta a sí mismo. Cada nodo sólo debe contener un padre.
Nodo Hijo: Un nodo puede tener varios hijos. Un nodo hijo es un nodo conectado directamente con otro cuando se aleja de la raíz.
Nodo Raíz: Es el único nodo del árbol que no tiene padre. En la representación que hemos usado, el nodo raíz es el que se encuentra en la parte superior del árbol.
Hojas: Las hojas vienen a ser todos los nodos que no tienen hijos.



2.4. Grafos (Bipartitas)

Un grafo bipartito, es denominado al grafo cuyos vértices se pueden separar en dos subconjuntos distintos, tal que no haya aristas entre los vértices del mismo conjunto. Se ve que un grafo es bipartito si no hay ciclo de longitud impar.

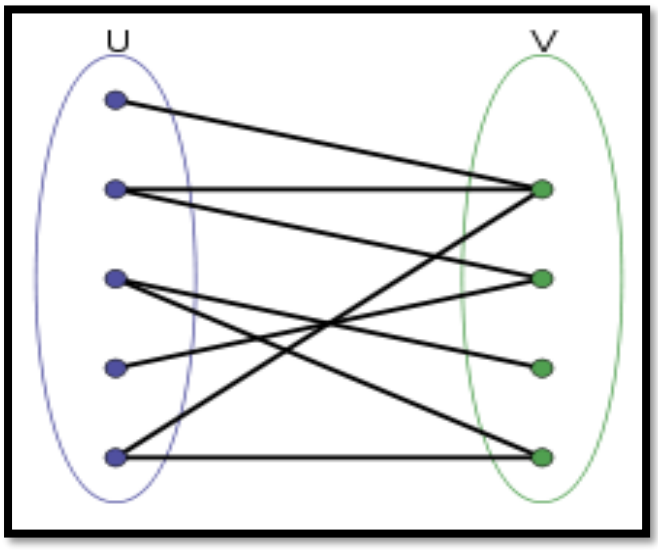
Los dos conjuntos pueden considerarse como una coloración del grafo con dos colores, los grafos bipartitos se caracterizan de diferentes maneras posibles:

a) Un grafo es bipartito si y solo si no contiene un ciclo impar.

b) Un grafo es bipartito si y sólo si es 2- color esto quiere decir que su número cromático es menor o igual a 2.

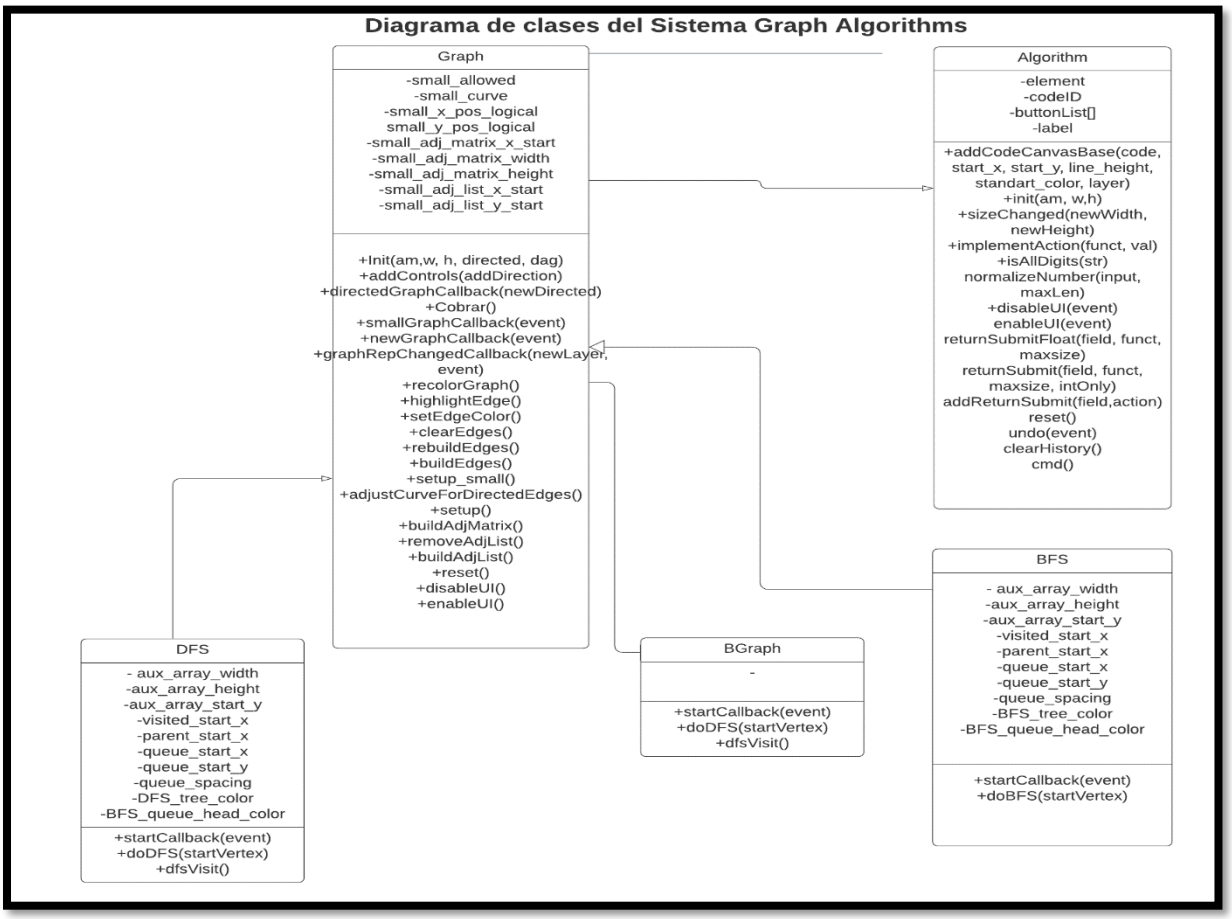
c) El espectro de un grafo es simétrico si y sólo si es un grafo bipartito.

Bipartite Graph



3. Metodología de Desarrollo:

3.1. Diagrama de Clases

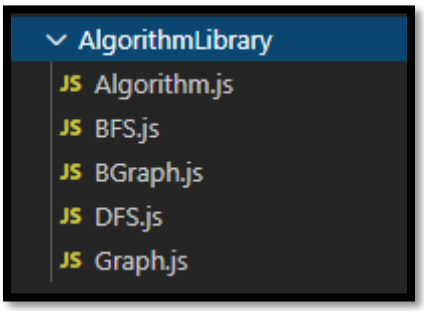


4. Librerías y otras herramientas:

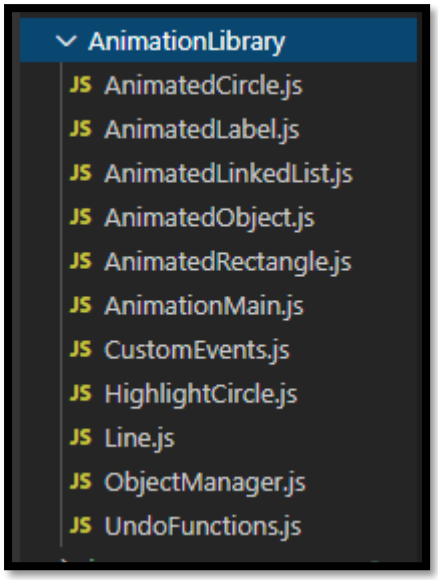
4.1. Librerías:

Para el desarrollo de este proyecto se usaron dos librerías que podemos encontrar en la página que recomendo el profesor, https://www.cs.usfca.edu/~galles/visualization/source.html , las cuales son AlgorithmLibrary y AnimationLibrary. Estas librerías están publicadas bajo una licencia FreeBSD. Copyright 2011 David Galles, Universidad de San Francisco. Todos los derechos reservados.

4.2. AlgorithmLibrary:



4.3. AnimationLibrary:



4.4. Archivo BFS.js:

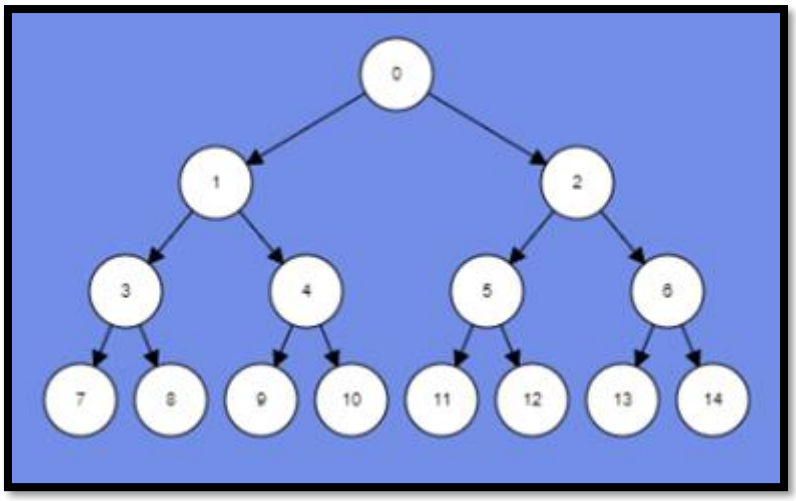
En este archivo javascript podemos encontrar el algoritmo (Búsqueda de anchura) para desarrollar eficientemente el tipo de dato árbol o grafo bipartita, es llamado en los archivos *BBFS.html* y *TBFS.html*

4.5. Archivo DFS.js:

En este archivo javascript podemos encontrar el algoritmo (Búsqueda de altura) para desarrollar eficientemente el tipo de dato árbol o grafo bipartita, es llamado en los archivos *BDFS.html* y *TDFS.html*

4.6. Archivo Graph.js:

En este archivo javascript podemos encontrar el algoritmo para graficar óptimamente un árbol.



Como podemos observar el árbol posee 15 nodos, que fueron creados y unidos con las siguientes líneas de código.

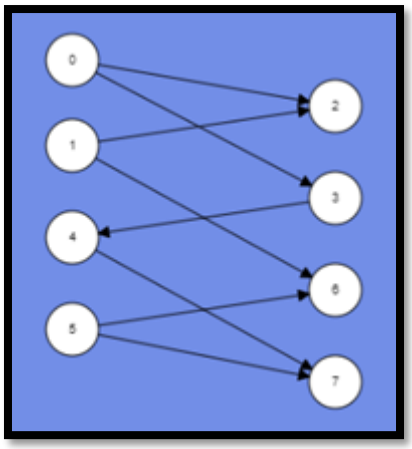
```
var SMALL_ALLOWED = [[true, false,  false,  true,  true,  true, true, true, true, true, true,  true,  true,  true,  true],
                     [true,  true, true,  false,  false, true,  true,  true, true, true, true, true,  true,  true,  true],
                     [true,  true,  true, true, true,  false,  false,  true, true, true, true, true,  true,  true,  true],
                     [true,  true,  true, true, true, true,  true, false, false, true, true, true,  true,  true,  true],
                     [true,  true, true,  true, true,  true, true,  true, true, false, false, true,  true,  true,  true],
                     [true, true,  true,  true,  true, true, true,  true, true, true, true, false,  false,  true,  true],
                     [true, true,  true,  true, true,  true,  true,  true, true, true, true, true,  false,  false,  true],
                     [true, true, true, true,  true,  true,  true,  true, true, true, true, true,  true,  true,  true],
                     [true, true, true, true,  true,  true,  true,  true, true, true, true, true,  true,  true,  true],
                     [true, true, true, true,  true,  true,  true,  true, true, true, true, true,  true,  true,  true],
                     [true, true, true, true,  true,  true,  true,  true, true, true, true, true,  true,  true,  true],
                     [true, true, true, true,  true,  true,  true,  true, true, true, true, true,  true,  true,  true],
                     [true, true, true, true,  true,  true,  true,  true, true, true, true, true,  true,  true,  true],
                     [true, true, true, true,  true,  true,  true,  true, true, true, true, true,  true,  true,  true]];
```

Los nodos fueron ubicados con las siguientes líneas de Código. La cual uno representa el eje X y el otro el eje Y

```
var SMALL_X_POS_LOGICAL = [775, 675, 875, 625, 725, 825, 925, 600, 650, 700, 750, 800, 850, 900, 950];
var SMALL_Y_POS_LOGICAL = [40, 100, 100, 160, 160, 160, 160, 220, 220, 220, 220, 220,220, 220, 220];
```

4.7. Archivo BGraph.js:

En este archivo javascript podemos encontrar el algoritmo para graficar óptimamente un grafo bipartita.



Como podemos observar el árbol posee 8 nodos, que fueron creados y unidos en la siguiente línea de código.

```
var SMALL_ALLLOWED = [[false, true,  false,  false,  true,  true, true, true],
                      [true,  true, false,  true,  true, true,  false,  true],
                      [true,  true,  true, true, true,  true,  true,  true],
                      [true,  true,  true, true, false, true,  true, true],
                      [true,  true, true,  true, true,  true, true,  false],
                      [true, true,  true,  true,  true, true, false,  false],
                      [true, true,  true,  true, true,  true,  true, true],
                      [true, true, true, true,  true,  true,  true,  true]];
```

Los nodos fueron ubicados con las siguientes líneas de Código. La cual uno representa el eje X y el otro el eje Y.

```
var SMALL_X_POS_LOGICAL = [700, 700, 900, 900, 700, 700, 900, 900];
var SMALL_Y_POS_LOGICAL = [25, 90, 60, 130, 160, 230, 200, 270];
```

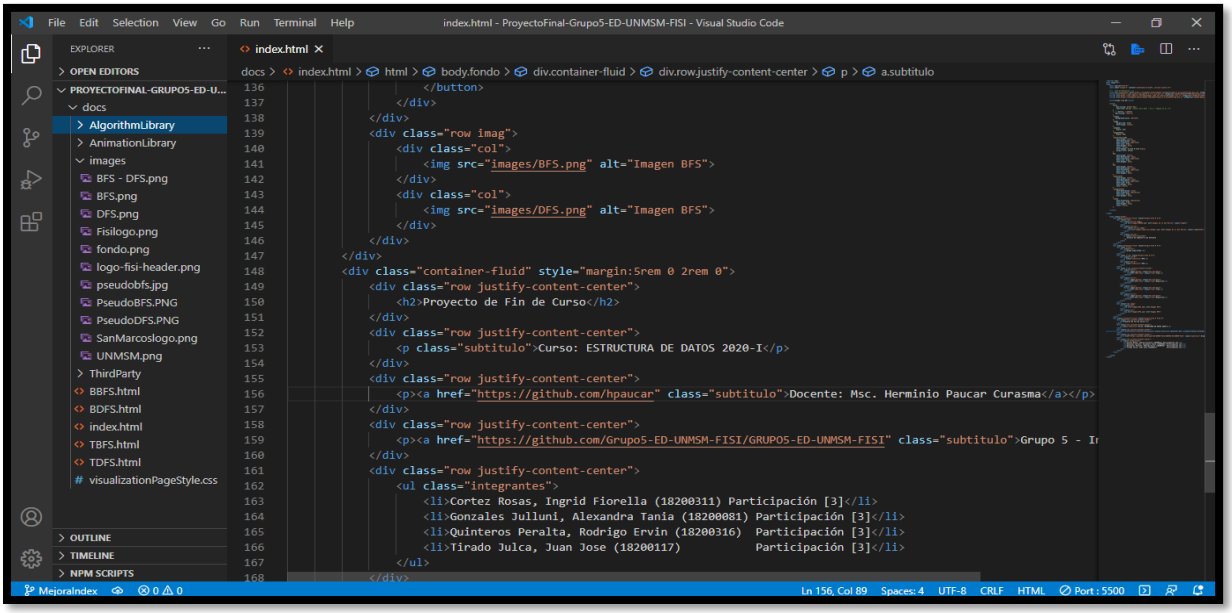
4.8. Otras herramientas:

4.8.1. Visual:

VS Code es un editor de código fuente sofisticado que admite muchas funcionalidades prácticas al momento de trabajar con el código. Estas son algunas de ellas:

- **Multiplataforma:** Fue creado y diseñado para que funcione en los tres sistemas operativos mayormente utilizados: Windows, Linux y Mac OS.
- **Plugins:** VS Code es una herramienta que se actualiza constantemente, tiene la posibilidad de adaptar plugins para trabajar.

- **Open Source:** VS Code es un editor de código abierto.



4.8.2. Git:

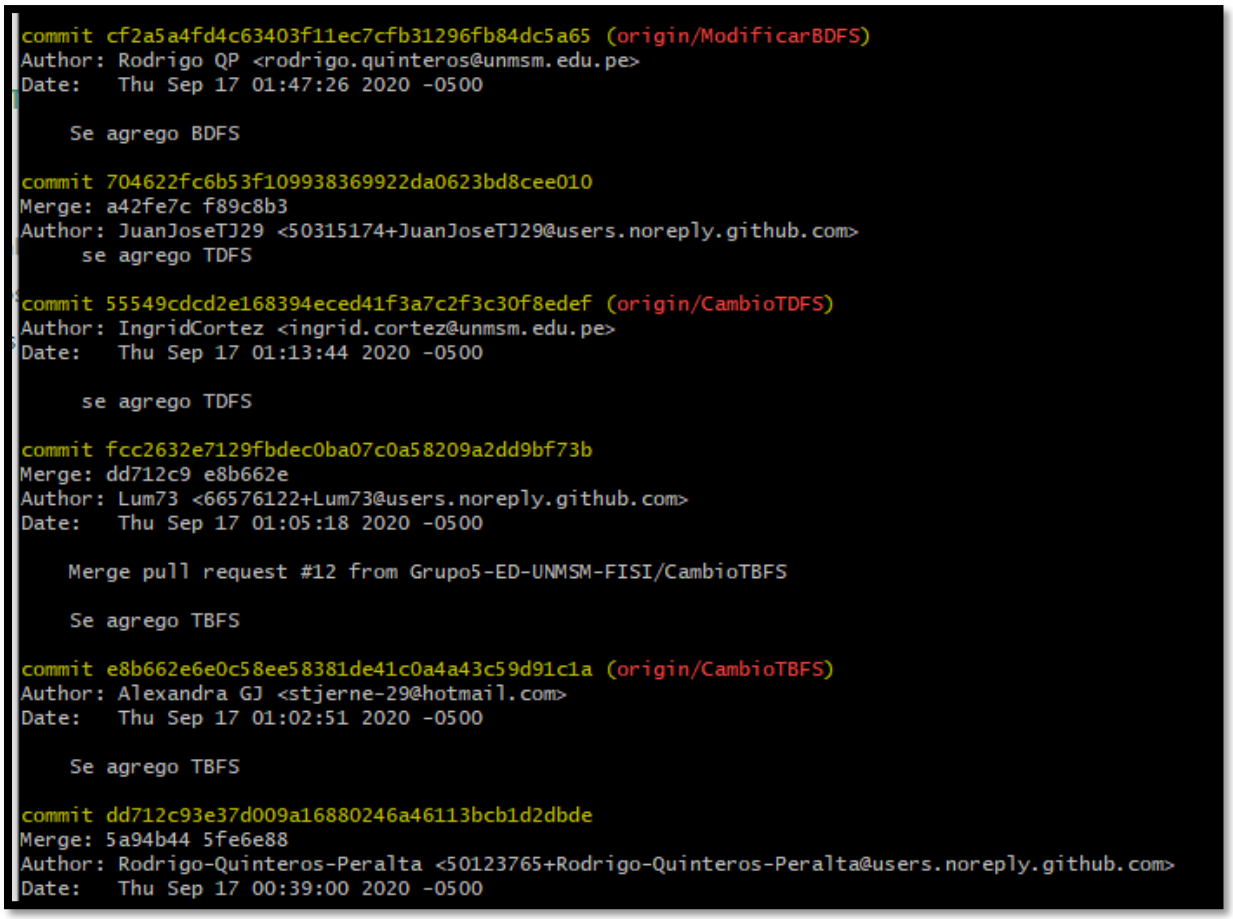
Utilizamos Git para un mejor control de versiones de nuestro código al momento de programar en equipo.
Ya que con él podemos trabajar en simultáneo sin la necesidad de estar solicitando cambios al resto de integrantes.

Lo que nos facilita al momento de programar, más si contamos la coyuntura actual en la que nos encontramos hoy en día.

Se usaron los siguientes comandos para el desarrollo óptimo del proyecto.

- Para revisar en qué rama estas → *git branch*
- Para moverse a la rama master→ *git checkout master*
- Para bajar los cambios → *git pull*
- Para crear y moverse a la rama nueva → *git switch -c nombrederama*
- Agregar todos los archivos para que esté pendiente de los cambios → *git add .*

- Crear commit (fotografía del proyecto en ese momento) → *git commit -m " Mensaje...."*
- Para subir los cambios locales al servidor remoto → *git push origin --all*



4.8.3. Bootstrap:

Bootstrap es un framework CSS3 y HTLM5 utilizado en aplicaciones front-end que nos facilita distribuir texto, imágenes y botones de la página.

Veamos un ejemplo en el archivo *index.html*, en la cual se utilizó el concepto grid que nos permite crear columnas invisibles en nuestra página.

Un ejemplo:

```
<div class="container">
  <div class="row">
    <div class="col-4">
          One of three columns
    </div>
    <div class="col-4">
          One of three columns
    </div>
    <div class="col-4">
          One of three columns
    </div>
  </div>
</div>
```

Resultado:

One of three columns	One of three columns	One of three columns

Nota: Solo puede ver haber como máximo col-12 en total en un row(fila); si sobrepasa, la siguiente col se dirige hacia abajo

Index.html

- **Código:**

```
<div class="container-fluid" style="margin:2rem 0 0 0">
    <div class="row">
        <div class="col-12 imag">
            <img src="images/UNMSM.png" alt="Imagen de la San Marcos" class="imagen">
        </div>
        <div class="col-12">
            <div class="col-12 imag">
                <img src="images/logo-fisi-header.png" alt="Imagen de la San Marcos" class="imagenFISI">
            </div>
        </div>
        <div class="col-12">
            <p class="texto-facultad">
                Escuela de Ingeniería de Software
            </p>
        </div>
    </div>
</div>
```

Ya que se está usando 3 col-12, lo que Bootstrap hará será crear 3 filas dentro de un row.

- **Resultado:**

Podemos observar que se crearon 3 col-12.

1. Imagen de San Marcos
2. Imagen de la Fisi
3. El texto de “ESCUELA DE INGENIERÍA DE SOFTWARE”.

- **Código:**

```
<div class ="row">
    <div class="col">
        <h1>graph algorithms</h1>
    </div>
</div>
```

Ya que solo hay un “col”, por defecto ocupará todo el espacio

- **Resultado:**

- **Código:**

```
<div class ="row" style="margin:4rem 0 0 0">
    <div class="col-6">
        <p class="subtitulo">BFS</p>
    </div>
    <div class="col-6">
        <p class="subtitulo">DFS</p>
    </div>
</div>
```

Hay col-6 y col-6, cada uno ocupa seis columnas de 12.

- **Resultado:**

- **Código:**

```
<div class ="row justify-content-around">
    <div class="col-1">
        <button type="button" class="btn btn-dark">
            <a href="TBFS.html" class="link">Tree</a>
        </button>
    </div>
    <div class="col-1">
        <button type="button" class="btn btn-dark">
            <a href="BBFS.html" class="link">Bipartita</a>
        </button>
    </div>
    <div class="col-1">
        <button type="button" class="btn btn-dark">
            <a href="TDFS.html" class="link">Tree</a>
        </button>
    </div>
    <div class="col-1">
        <button type="button" class="btn btn-dark">
            <a href="BDFS.html" class="link">Bipartita</a>
        </button>
    </div>
</div>
```

- Hay 4 col-1, cada uno ocupa una col de 12. En total se ocupan 4 columnas de 12.
- La class= “justify-content-around” separa cada col en partes iguales
- Bootstrap nos permite crear botones oscuros con el siguiente código:

➢ <button type="button" class="btn btn-dark"> </button>

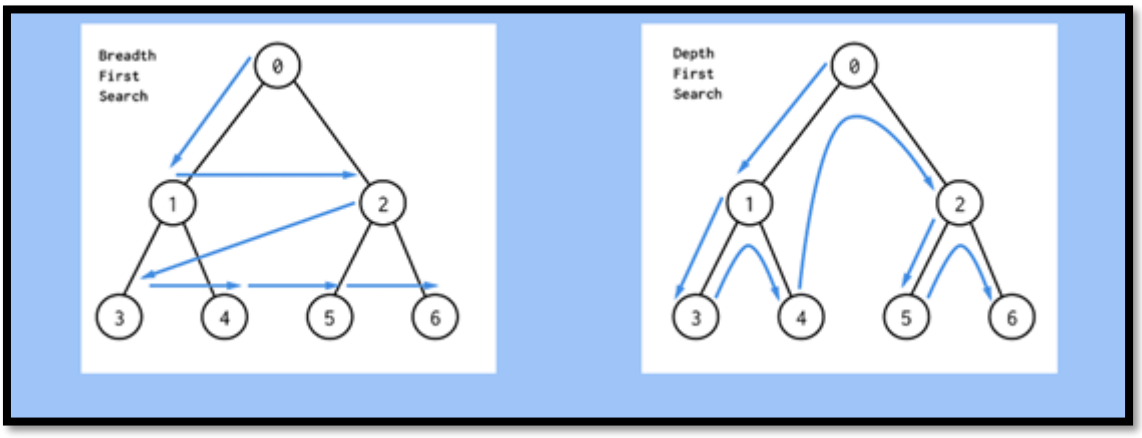
- **Resultado:**

- **Código:**

```
<div class="row imag">
    <div class="col">
        <img src="images/BFS.png" alt="Imagen BFS">
    </div>
    <div class="col">
        <img src="images/DFS.png" alt="Imagen BFS">
    </div>
</div>
```

Como hay 2 col por defecto, Bootstrap pone como valor a cada col como col-6 ya que 12/2 = 6

- **Resultado:**



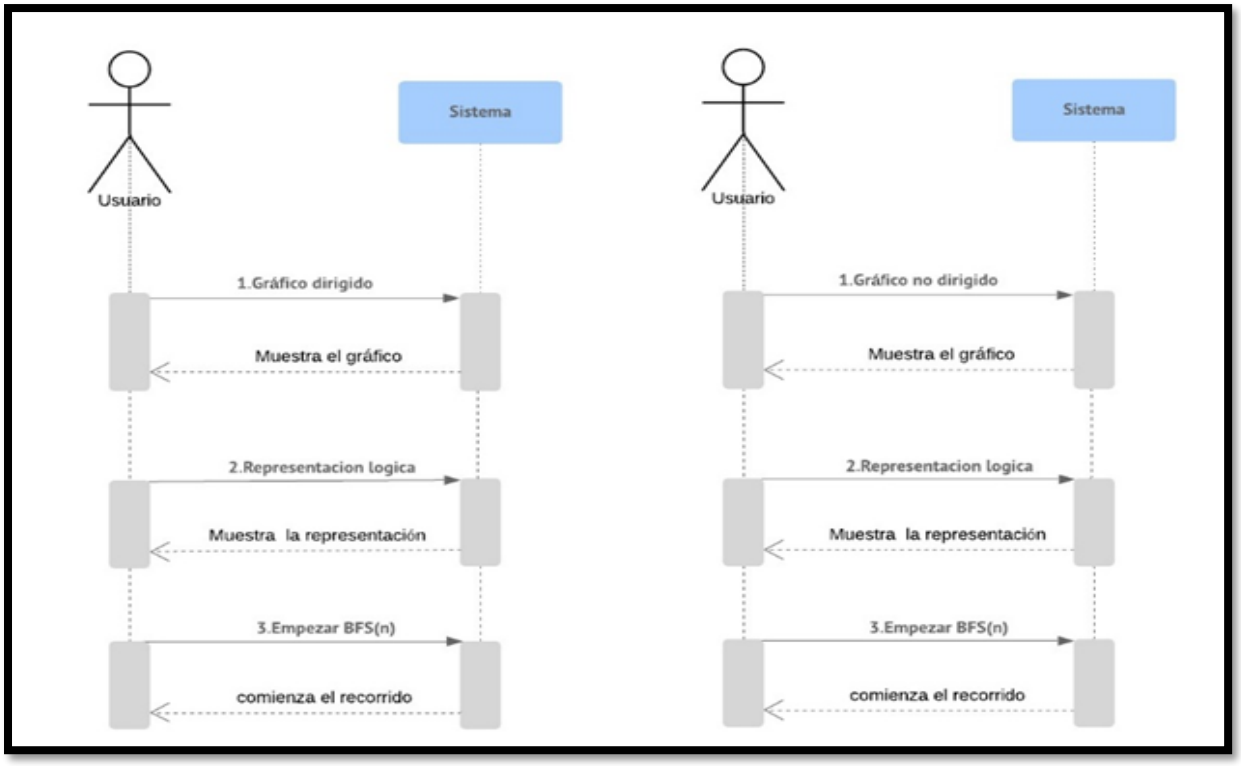
- **Código:**

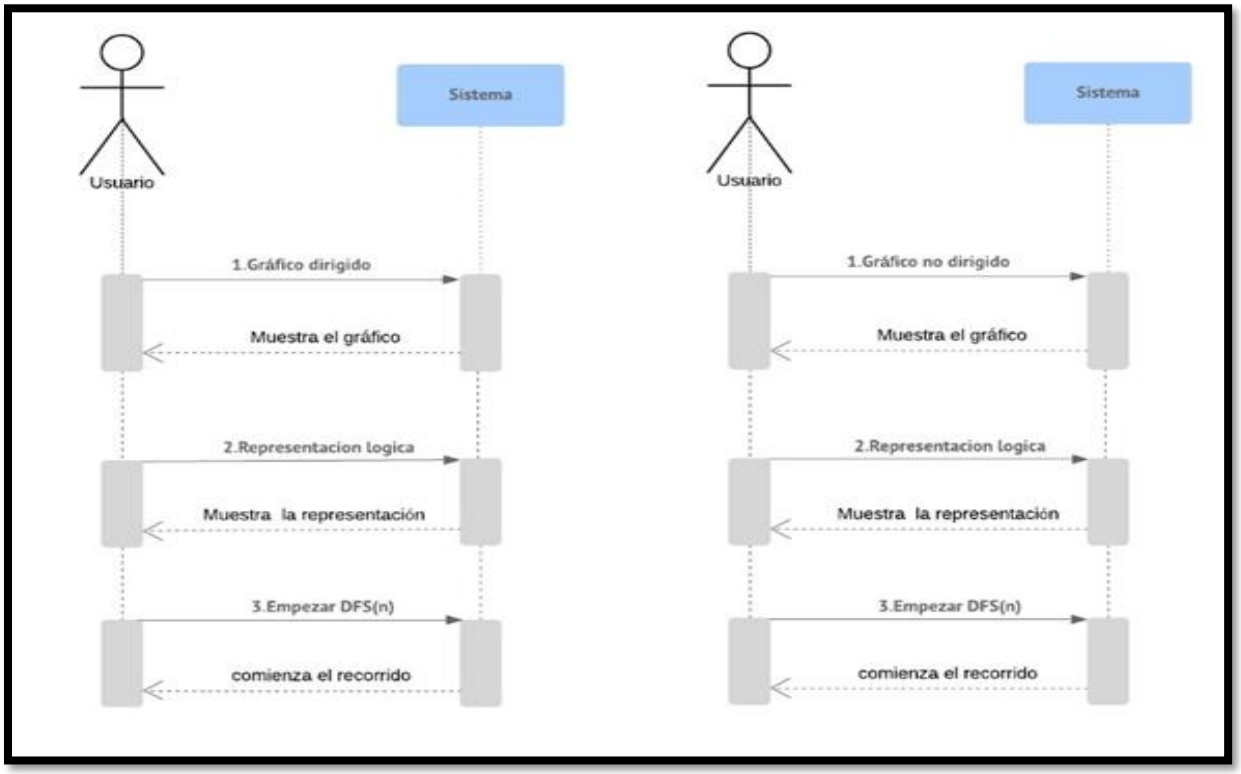
```
</div>
<div class="container-fluid" style="margin:5rem 0 2rem 0">
    <div class="row justify-content-center">
        <h2>Proyecto de Fin de Curso</h2>
    </div>
    <div class="row justify-content-center">
        <p class="subtitulo">Curso: ESTRUCTURA DE DATOS 2020-I</p>
    </div>
    <div class="row justify-content-center">
        <p><a href="https://github.com/hpaucar" class="subtitulo">Docente: Herminio Paucar Carasma Msc.Computer Scie</a></p>
    </div>
    <div class="row justify-content-center">
        <p><a href="https://github.com/Grupo5-ED-UNMSM-FISI/GRUPOS-ED-UNMSM-FISI" class="subtitulo">Grupo 5 - Integrantes</a></p>
    </div>
    <div class="row justify-content-center">
        <ul class="integrantes">
            <li>Cortez Rosas, Ingrid Fiorella (18200311) Participación [3]</li>
            <li>Gonzales Julluni, Alexandra Tania (18200081) Participación [3]</li>
            <li>Quinteros Peralta, Rodrigo Ervin (18200316)  Participación [3]</li>
            <li>Tirado Julca, Juan Jose (18200117)          Participación [3]</li>
        </ul>
    </div>
</div>
```

Se utiliza el row justify-content-center para que el texto de cada row se ubique en el centro

- **Resultado:**

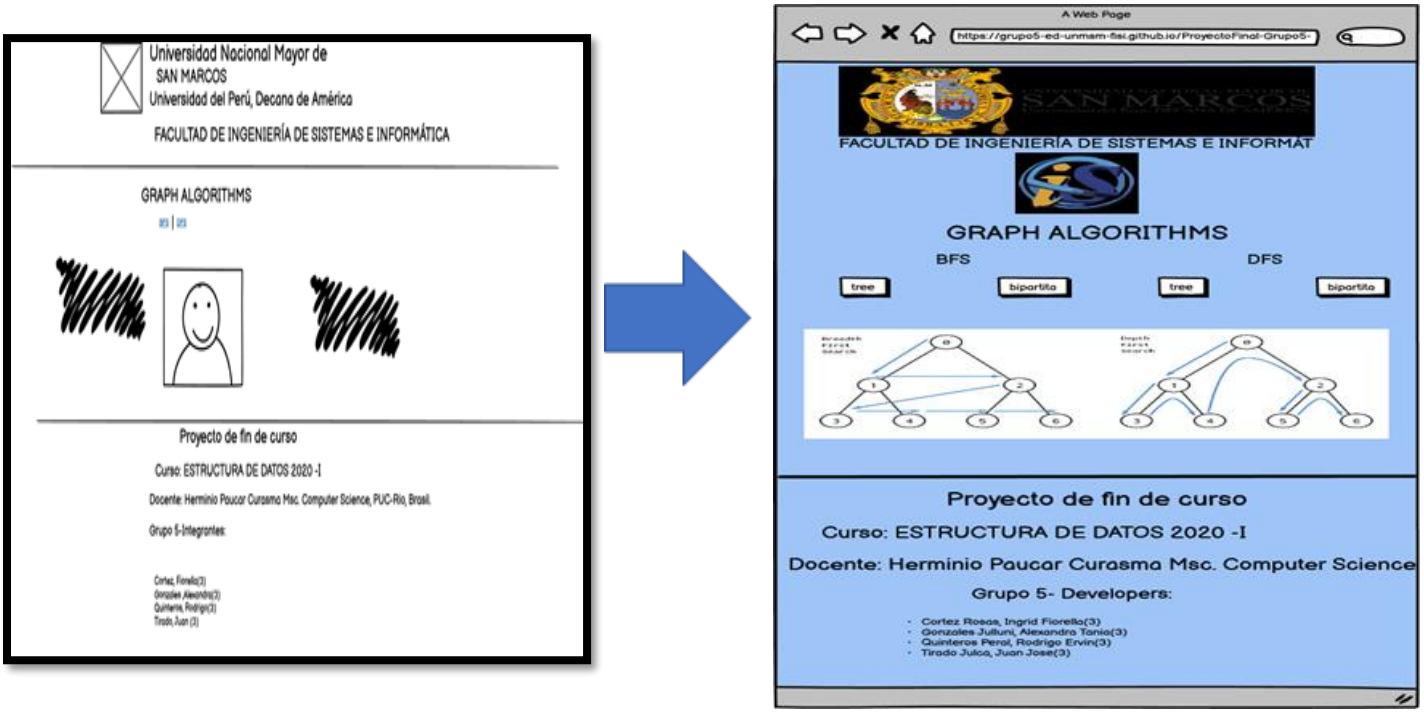
5. Diagramas de secuencia



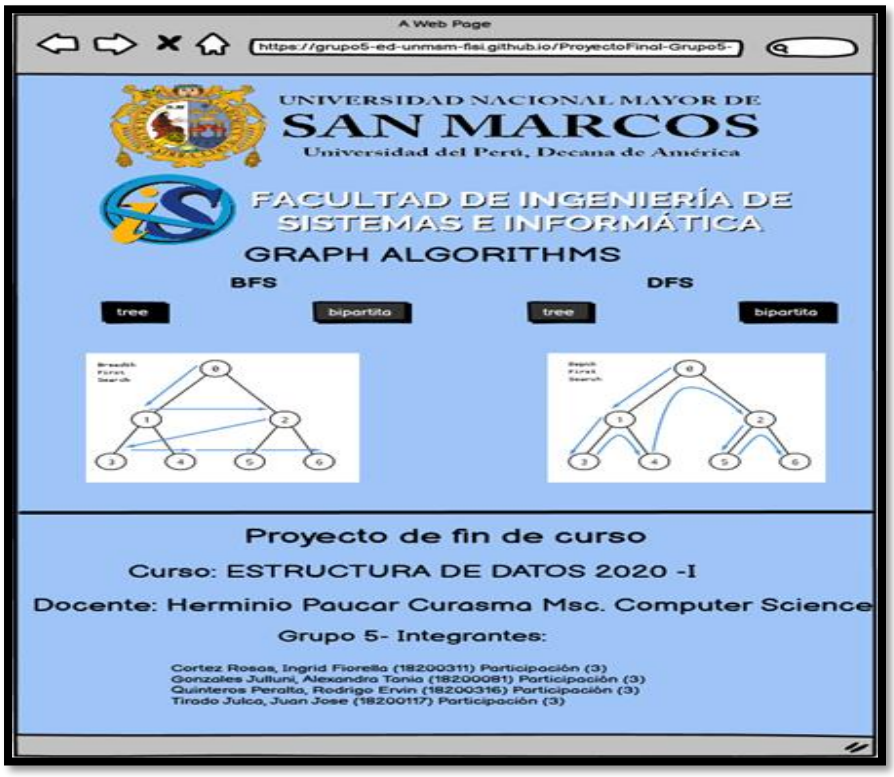


6. Prototipo mockups de la aplicación:

6.1. Interfaz principal :

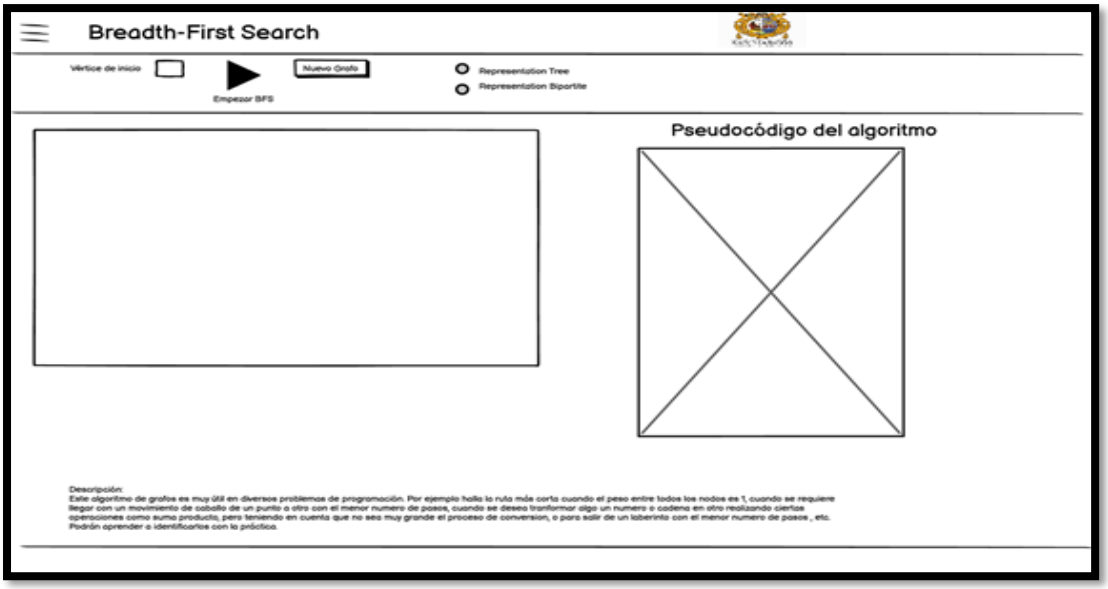


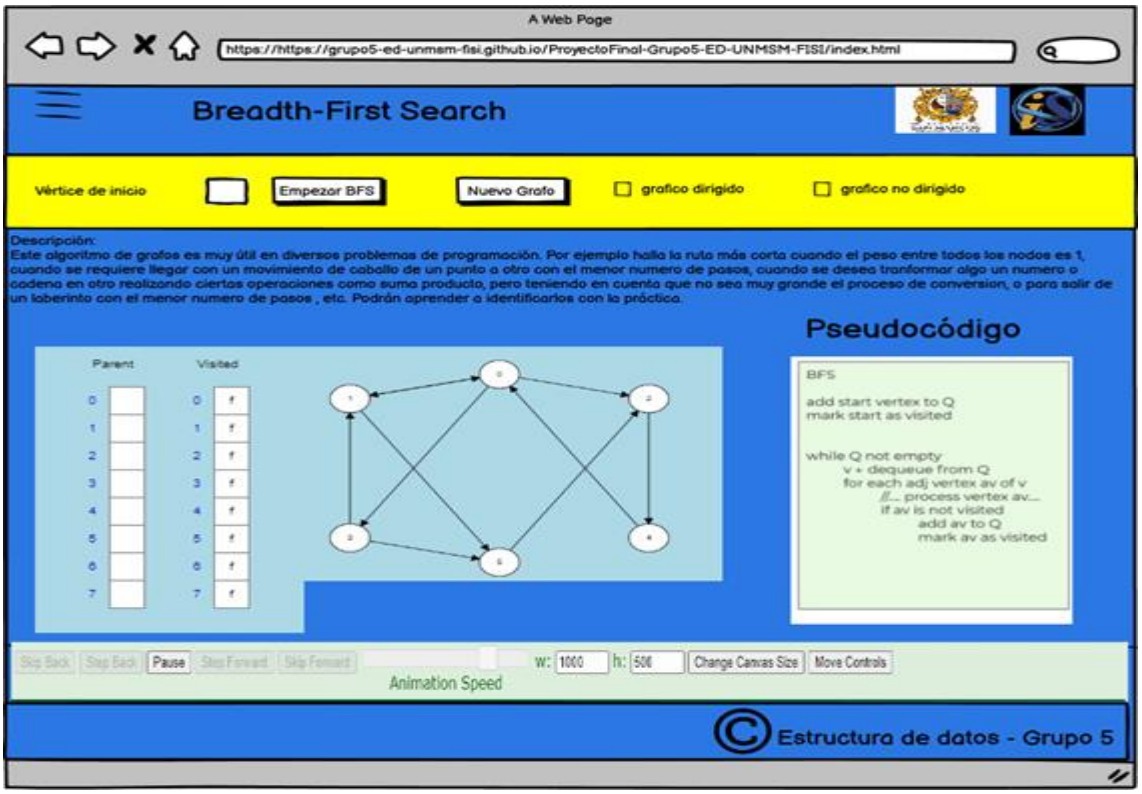
Versión final



6.2. Interfaz de aplicaciones:

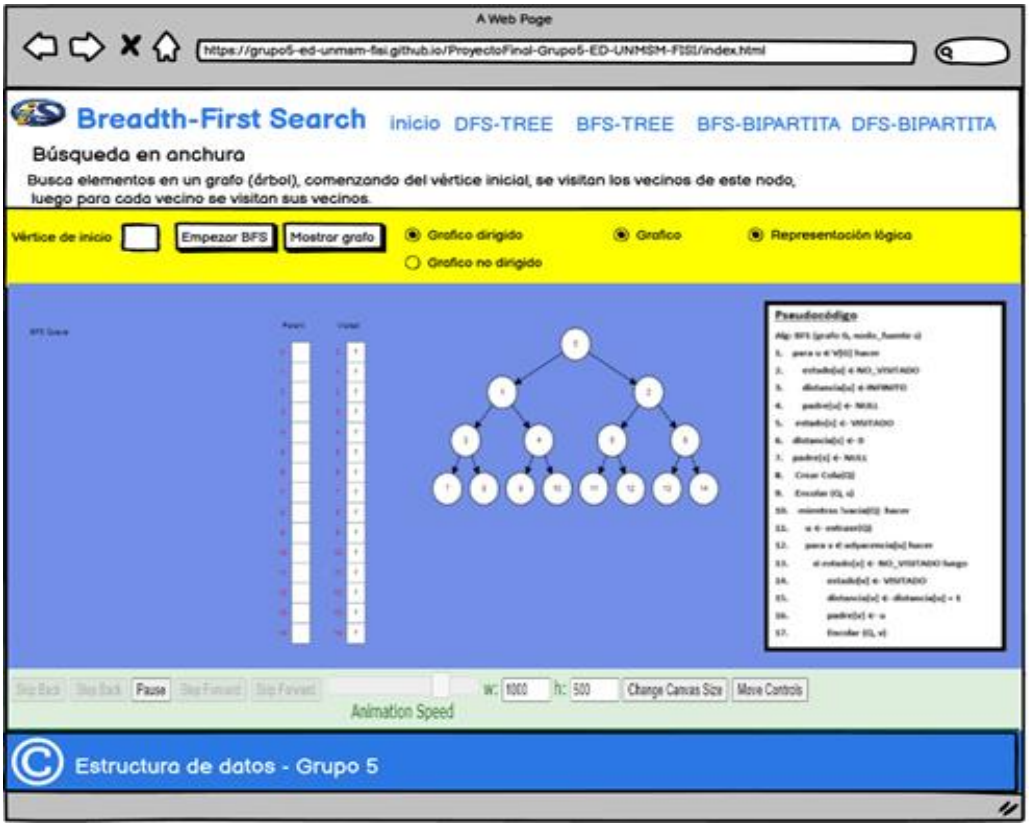
6.2.1. Primeras versiones



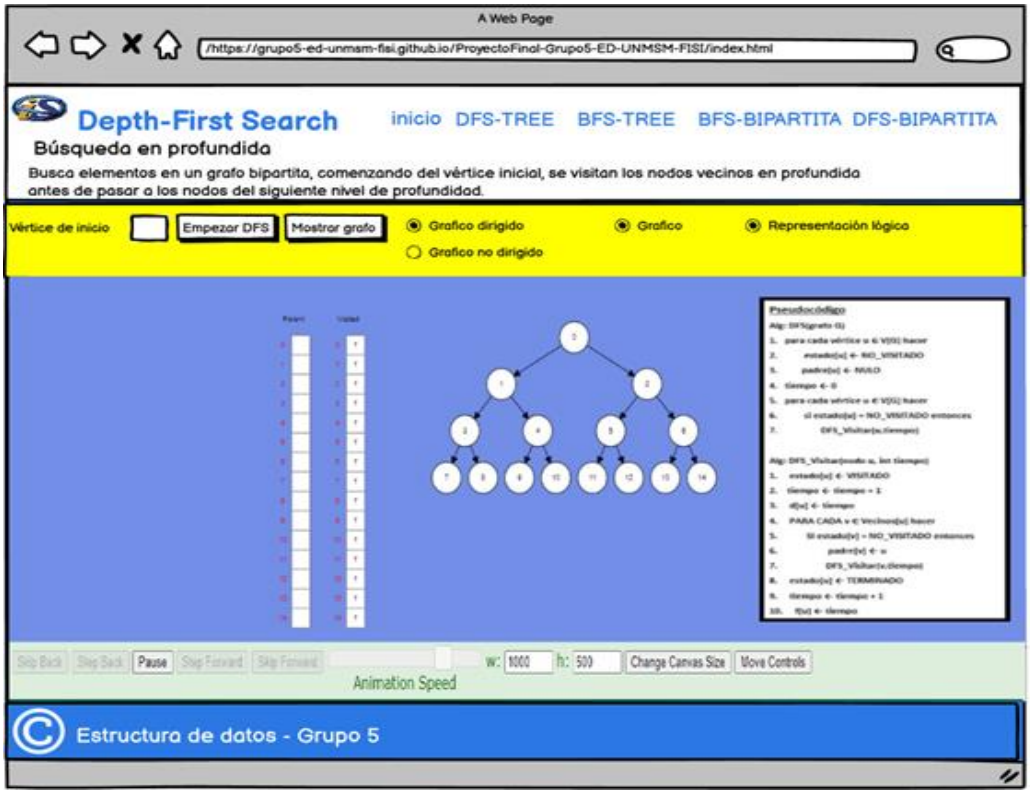


Versiones finales:

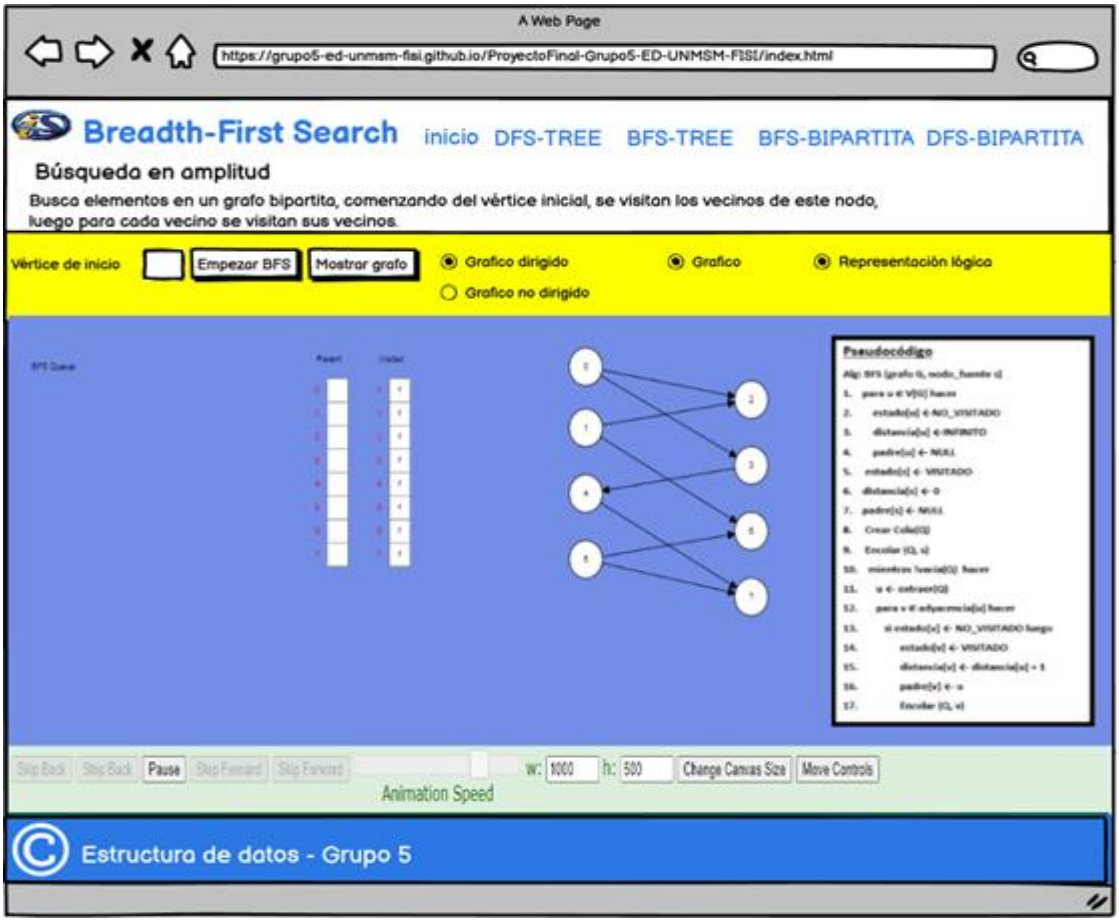
BFS - TREE



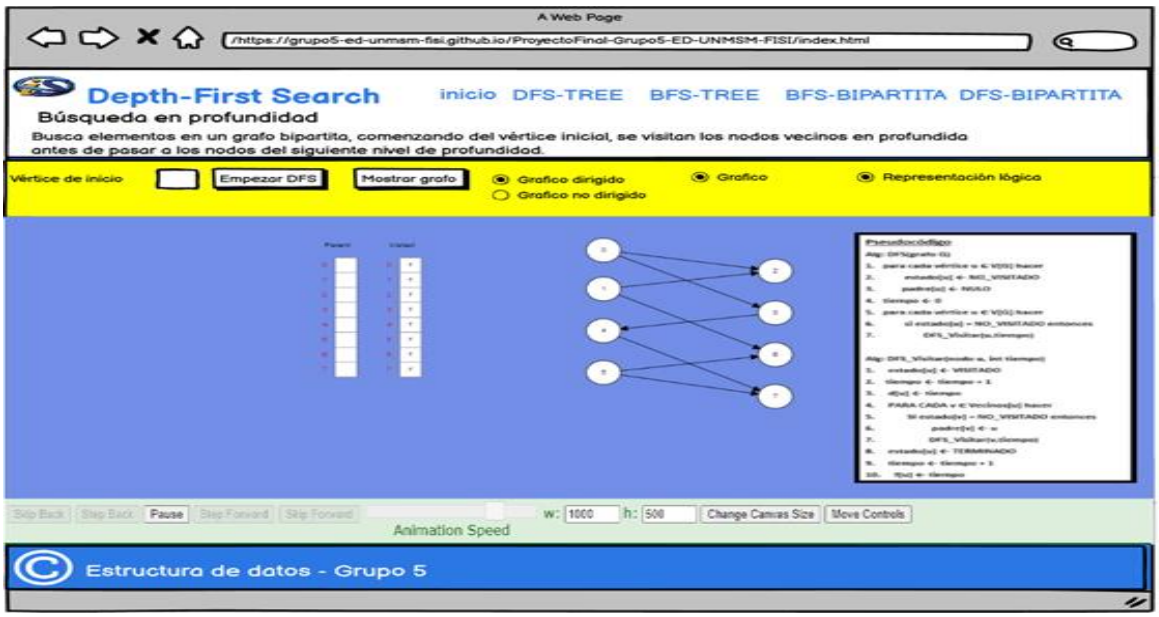
DFS - TREE



BFS - BIPARITITA



DFS - BIPARITITA



7. Conclusiones:

Con los algoritmos empleados en esta aplicación, se puede encontrar caminos o recorridos entre dos o más nodos dependiendo de la naturaleza del problema, logrando un desarrollo más eficiente. La utilización de estos algoritmos nos brindarán múltiples ventajas, en el caso de BFS nos permitirá encontrar el camino más corto entre 2 nodos, medido por el número de nodos conectados, podremos probar si un grafo de nodos es bipartito es decir si se puede dividir en 2 conjuntos, encontrar el árbol de expansión mínima en un grafo no ponderado y el algoritmo DFS nos permitirá encontrar nodos conectados en un grafo ,ordenamiento topológico en un grafo acíclico dirigido ,encontrar puentes en un grafo de nodos y encontrar nodos fuertemente conectados. Estos algoritmos, por ejemplo, en el caso de se puede usar en mapas de Google para encontrar rutas óptimas entre puntos turísticos de interés o para hacer búsquedas de textos en una página web por medio de un Web Crawler y en caso de por medio del algoritmo DFS, se puede resolver o simular juegos como laberintos o ajedrez.

8. Recomendaciones:

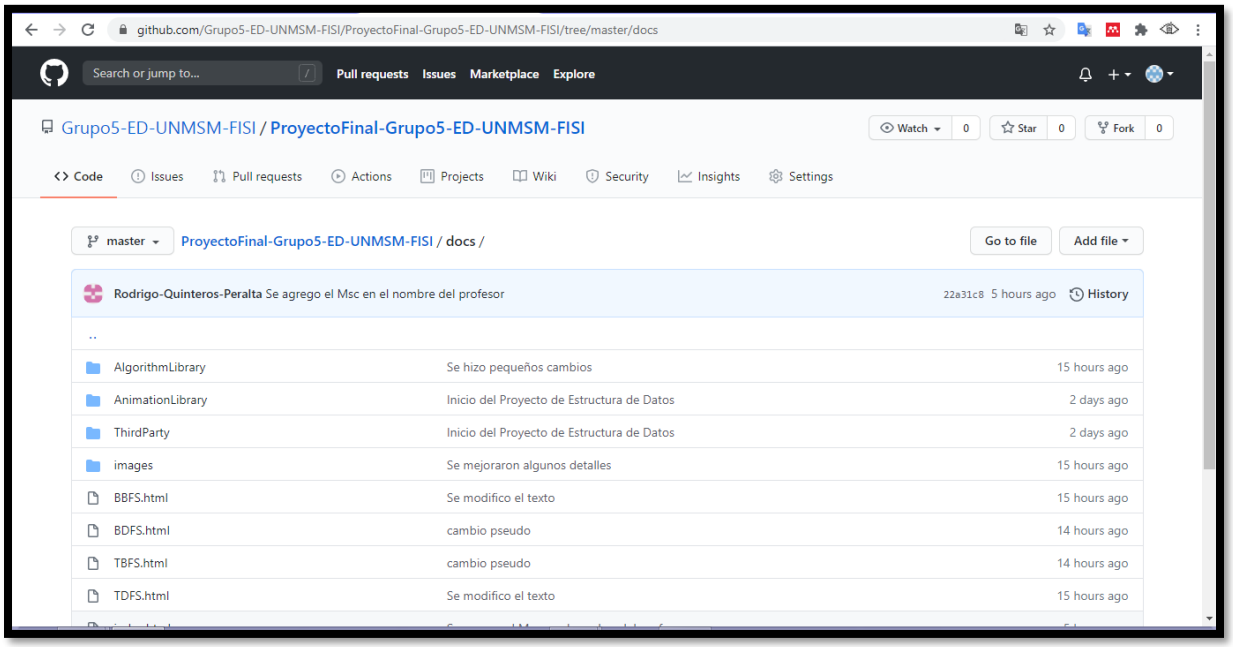
- Respecto al uso de la página web con GitHub se recomienda que el nombre de la carpeta que estará en tu repositorio sea docs, ya que de esta manera será más fácil enlazar la página web con tu código.

- Respecto al trabajo en equipo es de mucha ayuda hacerlo colaborativamente a través de GitHub donde cada miembro creará una rama, realizará cambios, se hará un commit que después será examinado y si es aprobado se hará un merge request con el master.

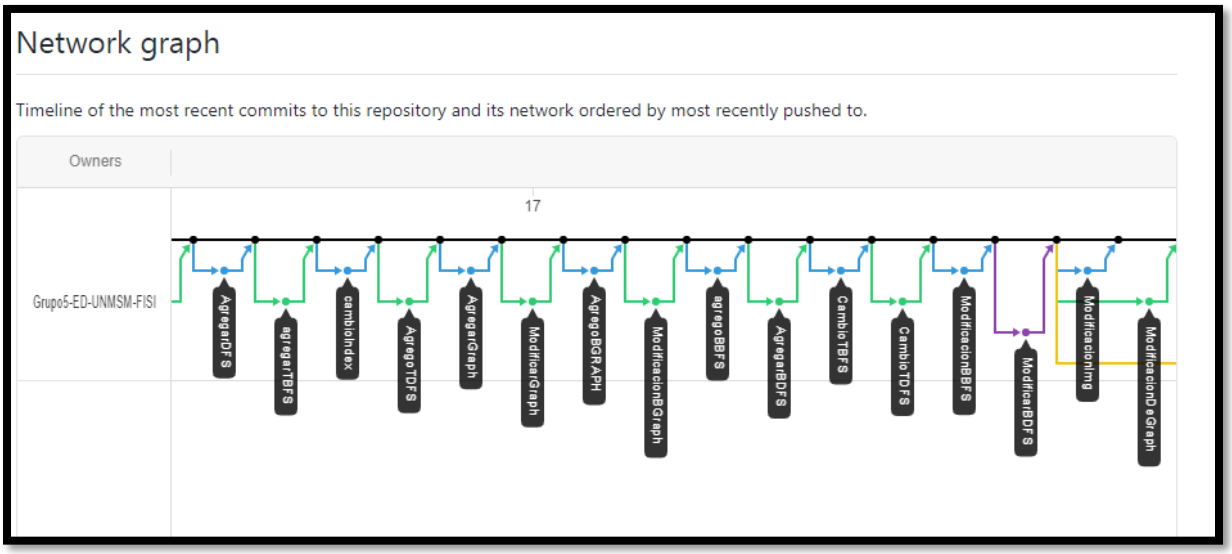
- Es recomendable usar Visual Code Studio ya que es un editor de código muy útil, de fácil instalación y uso.

9. Anexos:

Repositorio en GitHub:



Trabajo colaborativo en GitHub:



Despliegue del proyecto con GitHub page:

https://grupo5-ed-unmsm-fisi.github.io/ProyectoFinal-Grupo5-ED-UNMSM-FISI/index.html

10. Referencias:

- Badia, M., Martinez B., Morales, A., *Árboles-Estructura de datos de la Información* , Tema 10
- Twitter. (s. f.). *Bootstrap*. https://getbootstrap.com/docs/4.5/layout/overview/. Recuperado 18 de septiembre de 2020, de https://getbootstrap.com/.
- *Visual Studio Code*. (s. f.). https://visualstudio.microsoft.com/es/. Recuperado 18 de septiembre de 2020, de https://visualstudio.microsoft.com/es/
- *Git*. (s. f.). https://git-scm.com/. Recuperado 18 de septiembre de 2020, de https://git-scm.com/
- Galles, D. (s. f.). *Data Structure Visualizations*. Recuperado 18 de septiembre de 2020, de https://myusf.usfca.edu/arts-sciences/computer-science