Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Estructuras de Datos

Vacaciones junio 2021

Proyecto: TytusDS

Índice

1. Competencias

1.1. Competencia general

- Que el estudiante aplique los conocimientos adquiridos para construir diferentes tipos de datos abstractos mediante una aplicación Web con lenguaje JavaScript.

1.2. Competencias específicas

- Que el estudiante implemente los diferentes tipos de datos abstractos vistos en clase.
- Que estudiante administre archivos JSON para carga y almacenamiento de datos.
- Que el estudiante pueda abstraer los diferentes tipos de datos para construir estructuras de datos compuestas.
- Que el estudiante ponga en práctica los conocimientos de BlockChain mediante árboles de Merkle.
- Que el estudiante utilice GitHub y Pages para hacer la publicación de su aplicación utilizando de manera opcional frameworks y bibliotecas de dibujo.

2. Descripción

2.1. Descripción general

TytusDS es una aplicación Web Open Source con licencia MIT que se utilizará para construir y mostrar las estructuras de datos más utilizadas y vista en la clase. Su finalidad es que sirva como herramienta para entender el funcionamiento de las estructuras de datos y algoritmos relacionados.

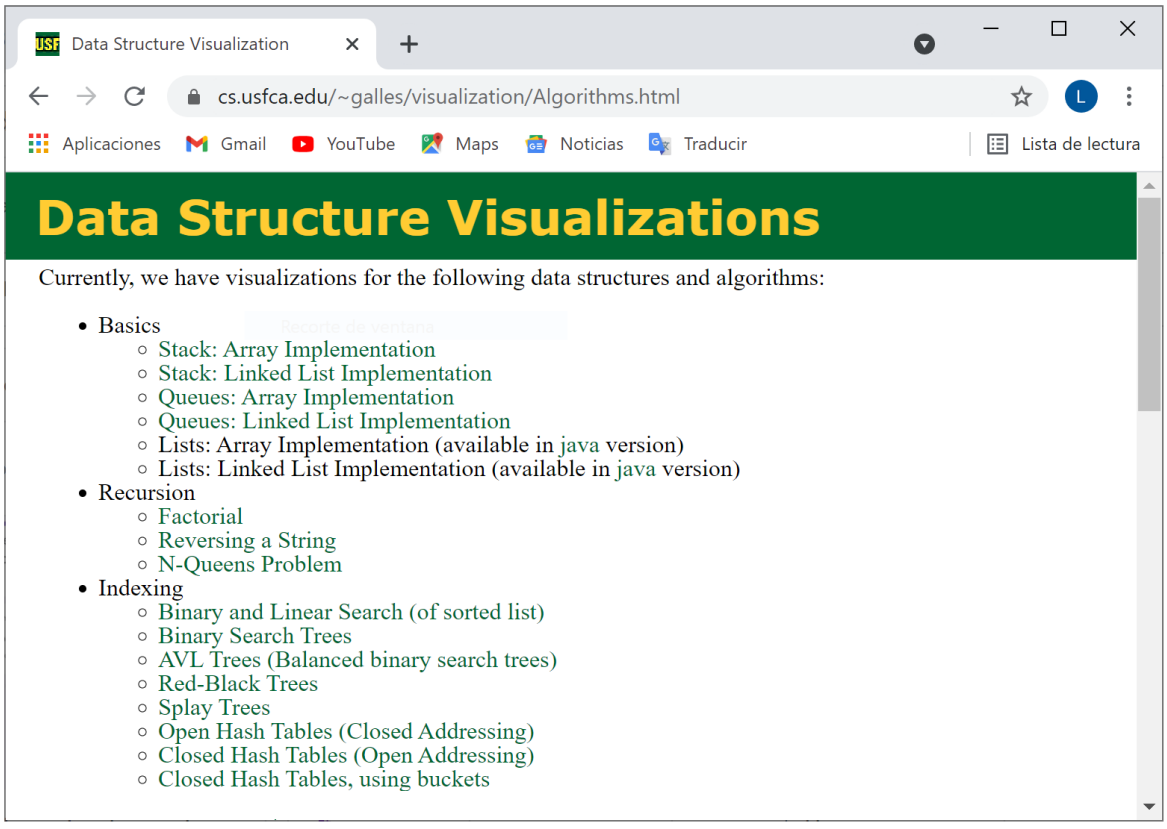
2.2. Fases

El proyecto consta de dos fases: la primera, consta de las estructuras lineales, ordenamientos y estructuras arbóreas; la segunda, consta de las estructuras no lineales, algoritmos y estructuras compuestas. Las estructuras deben soportar los diferentes tipos de datos que maneja JSON.

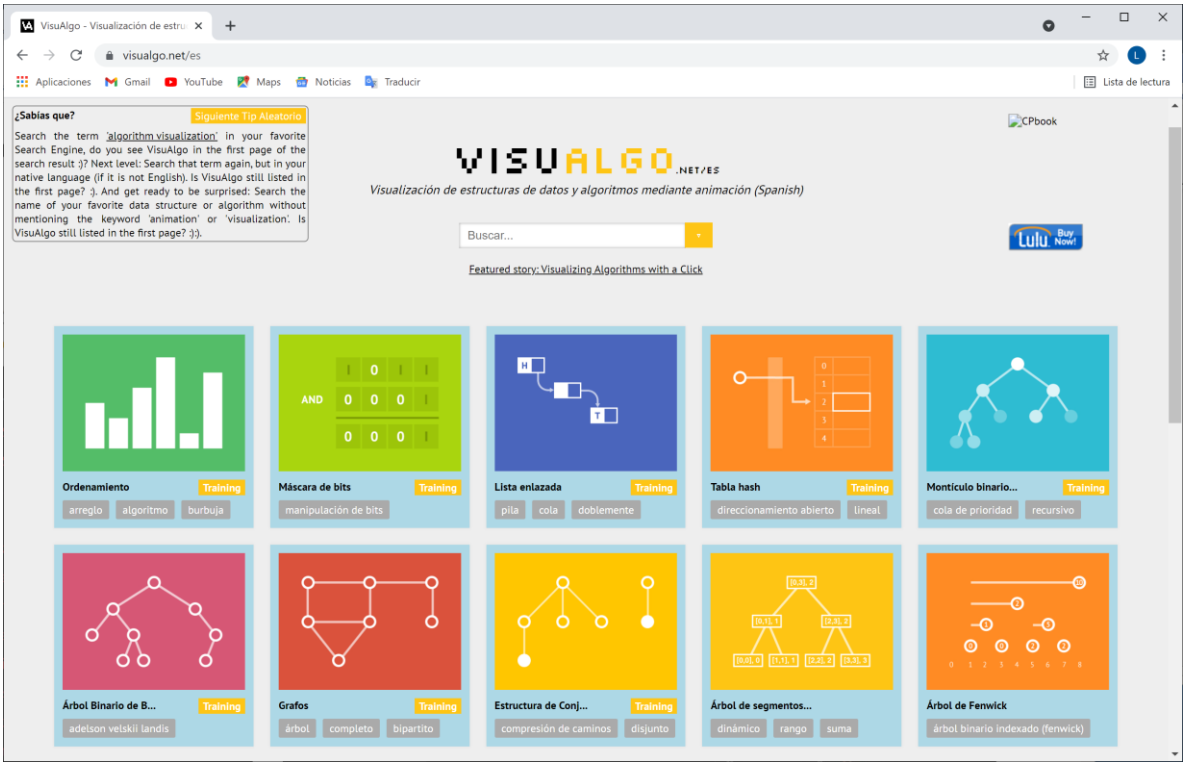
3. Componentes del sitio

En la página se debe mostrar el menú principal para ingresar a cada estructura o algoritmo de manera independiente, queda a discreción del grupo que se textual o gráfico. Tomar como ejemplo los siguientes sitios (considerando que no está permitida la copia de código):

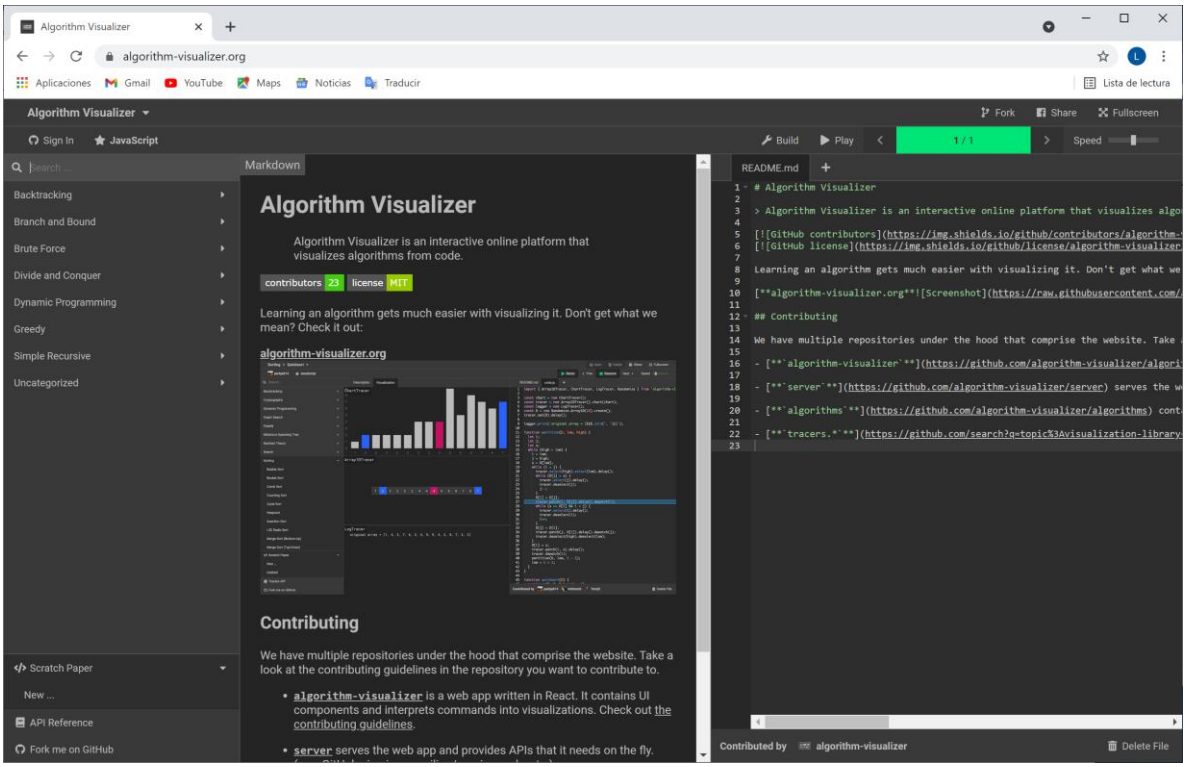
https://www.cs.usfca.edu/~galles/visualization/Algorithms.html



https://visualgo.net/es



https://algorithm-visualizer.org/



La diferencia de este proyecto de desarrollo radica en que el sitio utilizará idioma español y se utilizarán las estructuras vistas en el curso para que siguientes generaciones puedan hacer uso de la herramienta para aprender estructuras de datos, la estructuras que se deben implementar están listadas a continuación.

3.1. Menú principal

Para la primera fase (no se debe mostrar en el sitio el título de primera fase):

- Estructuras lineales
    - Lista simplemente enlazada
    - Lista doblemente enlazada
    - Lista circular simplemente enlazada
    - Lista circular doblemente enlazada
    - Pila
    - Cola
    - Cola de prioridad
- Ordenamientos
    - Burbuja
    - Selección
    - Inserción
    - Rápido
- Estructuras arbóreas
    - Árbol binario de búsqueda
    - AVL
    - Árbol B
    - Árbol B+
    - Árbol de Merkle

Para la segunda fase (no se debe mostrar en el sitio el título de segunda fase):

- Estructuras no lineales
    - Tabla Hash abierta
    - Tabla Hash cerrada
    - Recorrido y búsqueda por anchura de grafos
    - Recorrido y búsqueda por profundidad de grafos
    - Algoritmo de costo uniforme
    - Árbol de recubrimiento mínimo
- Algoritmo de codificación
    - Código de Hamming
    - Algoritmo de Huffman
    - Algoritmo LZW
    - Cifrado Feistel
    - Cifrado RSA
- Estructuras compuestas
    - Matrices dispersas
    - Row-major (2D a 1D)
    - Col-major (2D a 1D)
    - Construcción de estructuras compuestas

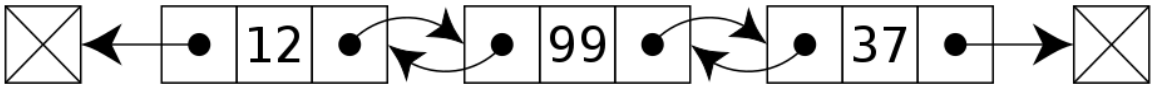
3.2. Estructuras lineales

Las opciones para las diferentes estructuras lineales son las siguientes:

- Agregar: agrega un elemento a la lista.
- Eliminar: elimina un elemento de la lista.
- Actualizar: actualiza un elemento de la lista.
- Buscar: busca un elemento de la lista.

- Cargar: carga desde un archivo JSON de un array de elementos, a excepción de la cola prioridad, esta debe tener un arreglo de objetos para indicar elemento y una prioridad numérica.
- Guardar: guarda la estructura actual en un archivo JSON. Considerar el array de elementos y para la cola de prioridad el array de objetos.

Entre la configuración de las estructuras lineales está el ingreso (al inicio, al final u ordenado), la repetición de elementos (activada o desactivada), y la velocidad de animación.



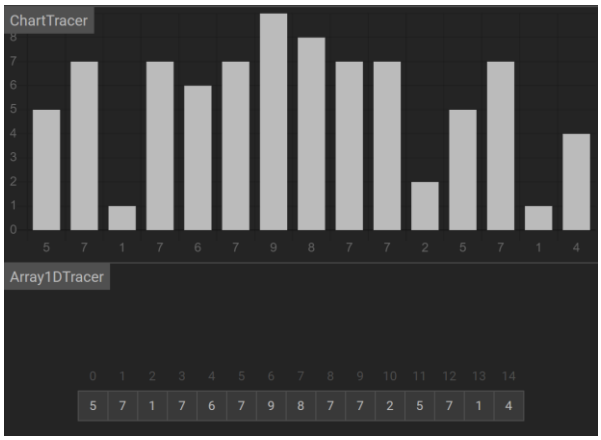
Ejemplo de una lista doblemente enlazada

3.3. Ordenamientos

Las opciones para los diferentes ordenamientos son las siguientes:

- Cargar: carga desde un archivo JSON un array de elementos.
- Guardar: guarda la estructura actual en un archivo JSON.

Entre la configuración de los ordenamientos solamente se encuentra la velocidad de animación.

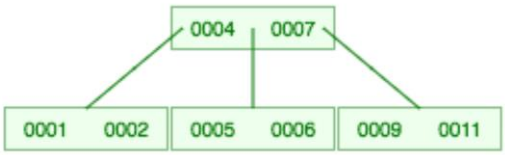


Ejemplo de ordenamiento antes de la animación

3.4. Estructuras arbóreas

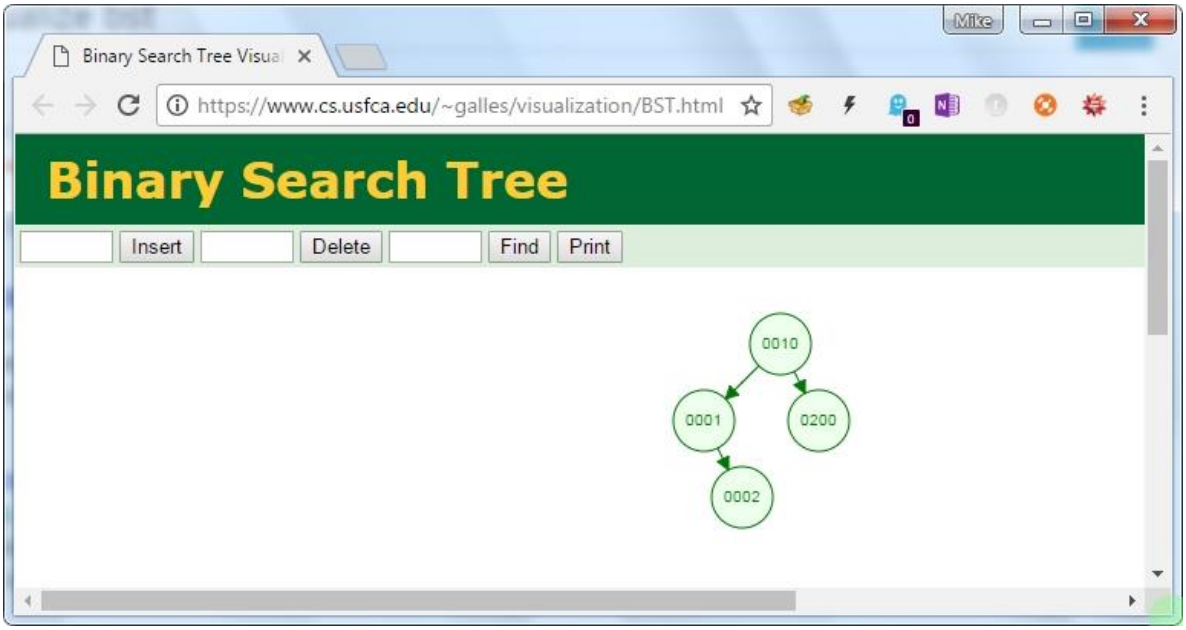
Las opciones para las diferentes estructuras arbóreas son las siguientes:

- Agregar: agrega un elemento al árbol.
- Eliminar: elimina un elemento al árbol.
- Actualizar: actualiza un elemento del árbol.
- Buscar: busca un elemento del árbol.
- Cargar: carga desde un archivo JSON de un array de elementos.
- Guardar: guarda la estructura actual en un archivo JSON.



Ejemplo de un árbol B de grado 3

Entre la configuración de las estructuras arbóreas está la repetición de elementos (activada o desactivada), y la velocidad de animación. Exclusivo para los árboles B y B+ está el grado.

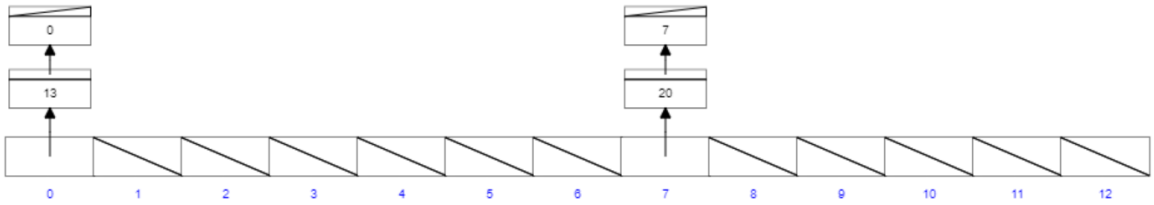


Ejemplo de operaciones de un árbol binario de búsqueda

3.5. Estructuras no lineales

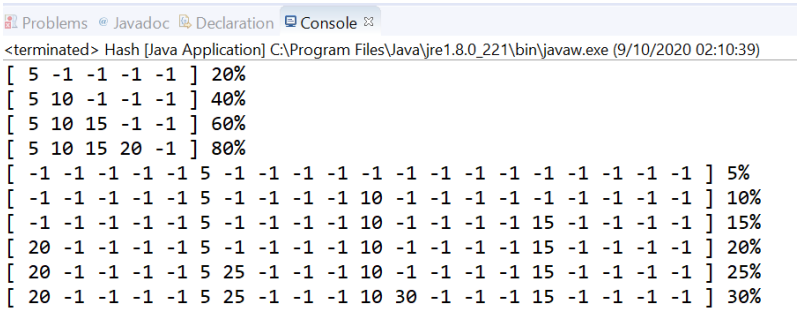
Las opciones para las tablas Hash son las siguientes:

- Agregar: agrega un elemento a la tabla.
- Eliminar: elimina un elemento de la tabla.
- Actualizar: actualiza un elemento de la tabla.
- Buscar: busca un elemento de la tabla.
- Cargar: carga desde un archivo JSON de un array de objetos.
- Guardar: guarda la estructura actual en un archivo JSON en un array de objetos.



Ejemplo de una tabla Hash abierta de direccionamiento cerrado de tamaño 13

Entre la configuración de las tablas hash está el tamaño (m), la función Hash (simple, división y multiplicación), el rango para Hash cerrado (minino y máximo) para hacer rehashing, la prueba para Hash cerrado (lineal, cuadrática y doble Hash), y la velocidad de animación.

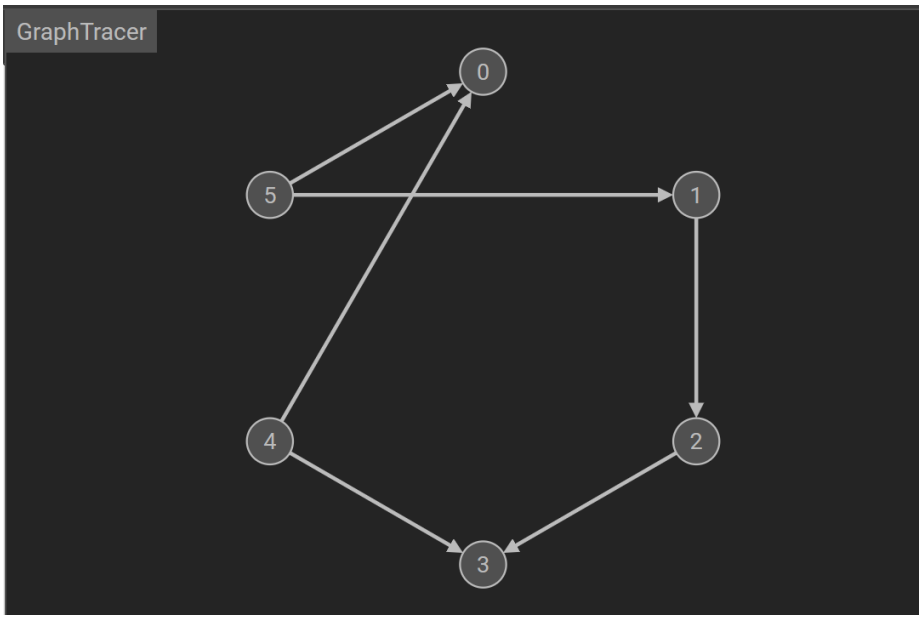


Ejemplo de rehashing

Las opciones para los grafos son las siguientes:

- Agregar: agrega un nodo al grafo.
- Eliminar: elimina un nodo al grafo.
- Actualizar: actualiza un nodo al grafo.
- Buscar: busca un nodo del grafo, con la opción de mostrar el camino.
- Recorrer: muestra el listado de nodos del árbol.
- Cargar: carga desde un archivo JSON de un array de objetos.
- Guardar: guarda la estructura actual en un archivo JSON en un array de objetos.

Entre la configuración de los grafos está el almacenamiento (matriz de adyacencia o lista de adyacencia), y la velocidad de animación. Hay que considerar que para el JSON se debe almacenar nodo inicio, nodo fin y distancia. Si fuera bidireccional es necesario dos líneas separadas de aristas.



Ejemplo de un grafo dirigido

3.6. Algoritmos de codificación

Para todos los algoritmos se puede cargar el contenido mediante un archivo de texto. Al cargar el contenido debe mostrarse en un campo de texto y puede ser editado para su posterior almacenamiento. En otro cuadro de texto ya sea a la derecha o abajo se debe mostrar el resultado del algoritmo con posibilidad de guardar en un archivo de texto.

Palabra			0		1	1	0		1	0	1
Dato	p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7
p1	1		0		1		0		1		1
p2		0	0			1	0			0	1
p4				0	1	1	0				
p8								0	1	0	1

Ejemplo de codificación Hamming

Los algoritmos de Hamming, Huffman, LZW y RSA, se deben implementar como se enseñe en clase sin ninguna configuración adicional.

Para el cifrado feistel también se implementará como en clase, pero se tendrá la opción de ingresar el número de rondas a ejecutar.

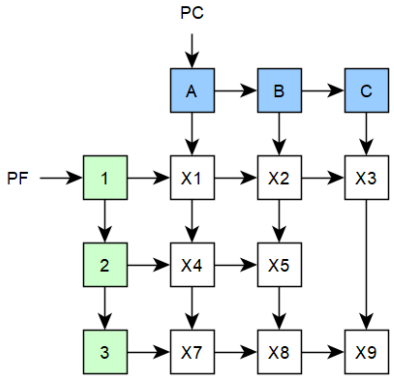
	L								R								
W0	0	1	0	0	0	0	1	1	0	1	0	0	0	1	1	0	
									0	0	1	1	0	0	0	1	K0
									0	1	1	1	0	1	1	1	F0
									0	1	0	0	0	0	1	1	
W1	0	1	0	0	0	1	1	0	0	0	1	1	0	1	0	0	
									0	1	1	0	0	0	1	0	K1
									0	1	0	1	0	1	1	0	F1
									0	1	0	0	0	1	1	0	
W2	0	0	1	1	0	1	0	0	0	0	0	1	0	0	0	0	
									1	1	0	0	0	1	0	0	K2
									1	1	0	1	0	1	0	0	F2
									0	0	1	1	0	1	0	0	
W3	0	0	0	1	0	0	0	0	1	1	1	0	0	0	0	0	
									1	0	0	0	1	0	0	1	K3
									0	1	1	0	1	0	0	1	F3
									0	0	0	1	0	0	0	0	
W4	1	1	1	0	0	0	0	0	0	1	1	1	1	0	0	1	

Ejemplo de Feistel de 4 rondas

Para todos los algoritmos se debe tener la configuración de la velocidad de animación.

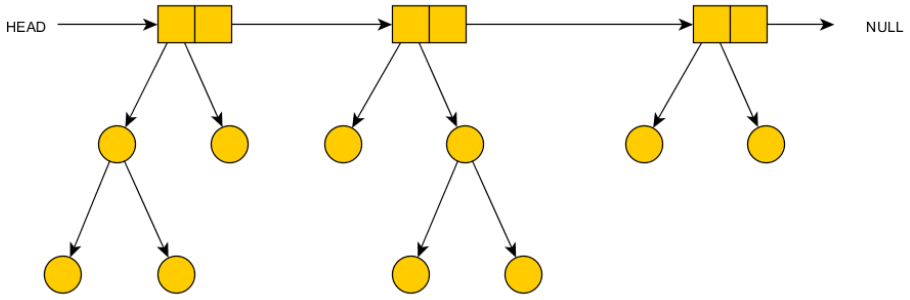
3.7. Estructuras compuestas

Para este apartado, se podrá seleccionar dos tipos de estructuras de datos, de las diferentes estructuras implementadas en el sitio. Por ejemplo, podría ser una lista de listas, un árbol de lista, una lista de árboles, etc.



Ejemplo de una lista de listas con casillas vacías representado mediante una matriz dispersa con encabezados

Las operaciones y la configuración serán las mismas que se utilizaron para cada una de las dos estructuras que fueron seleccionadas. Con la matriz dispersa se debe considerar la activación o desactivación de encabezados y el número de enlaces entre nodos.



Ejemplo de una lista de árboles

Tomar en cuenta que la carga y almacenamiento de datos en JSON debe tener la estructura adecuada para su lectura y escritura.

4. Términos del proyecto, entregables y calificación

4.1. Términos del proyecto

- Se formarán equipos de hasta 3 estudiantes por afinidad.
- Cada equipo deberá decidir quién será el coordinador para agregarlo como colaborador del repositorio para aceptar los commits, considerar también que el coordinador proporcionará una nota a cada estudiante donde no habrá empates.
- También cada estudiante debe agregar su Username de GitHub a la hoja de cálculo y modificar su Name para que aparezca su nombre completo para lograr identificar a cada estudiante.
- El proyecto está diseñado por el catedrático bajo una licencia Open Source, específicamente MIT. Por convenio, los estudiantes aparecerán como contribuidores junto con el copyright. Además, cualquier biblioteca autorizada también se debe colocar la licencia y el copyright en el archivo LICENSE.md en su carpeta respectiva.
- El lenguaje de programación es JavaScript. Se permite el uso de frameworks y bibliotecas media vez cumplan con tener licencia MIT y que se ejecuten adecuadamente en GitHub Pages. Para la construcción de estructuras o algoritmos NO ESTÁ PERMITIDO EL USO DE BIBLIOTECAS.
- Cada integrante de los equipos debe hacer sus propuestas de cambio mediante pull request directamente al master de este repositorio (no hacer pull request de la rama de cada uno para evitar conflictos), queda a discreción de cada equipo utilizar de manera independiente una rama u otro repositorio para pruebas.

- Para el desarrollo del proyecto se asignará una carpeta del repositorio de TytusDS a cada grupo.
- Copias de proyectos tendrán de manera automática una nota de 0 puntos y serán reportados a la Escuela de Ciencias y Sistemas los involucrados.
- Durante la calificación se realizarán preguntas sobre el código para verificar la autoría de este, de no responder correctamente la mayoría de las preguntas se reportará la copia.
- La entrega se realizará mediante commits al repositorio principal, para cada equipo se le indicará su carpeta específica.
- La calificación se realizará de manera virtual (ya sea en meet o zoom) con las cámaras activadas, cada calificación será almacenada.
- No se recibe ningún proyecto después de finalizada la entrega.
- Copias de proyectos obtendrán una nota 0, por lo que pierde automáticamente el laboratorio, se utilizará la herramienta JPlag https://jplag.ipd.kit.edu/
- Durante la calificación se verificará la autoría mediante preguntas, si no la responde se procederá a anular su nota del proyecto.
- Los estudiantes al hacer un commit aceptan las condiciones, licencias y convenios relacionados con el fin del proyecto.
- En cuanto al almacenamiento y extracción de datos, se debe considerar la codificación UTF8, ASCII e ISO 8859-1.

4.2. Calificación

La nota tendrá la siguiente ponderación:

- Un porcentaje será evaluado mediante una hoja de calificación (nota grupal de 50 puntos).
- Un porcentaje será evaluado mediante Stack Ranking de grupos. El proyecto del equipo ganador será utilizado como base de TytusDS (nota grupal de 20 puntos).

- Un porcentaje será evaluado mediante el total de commits aceptados y la calidad de estos mediante Stack Ranking por equipo (nota individual de 10 puntos).
- Un porcentaje será por calificación interna del equipo también por Stack Ranking por equipo (nota individual de 10 puntos).
- Un porcentaje será evaluado mediante una pregunta y modificación de código no necesariamente en la parte que haya trabajado (nota individual de 10 puntos).

La metodología Stack Ranking proporciona el 100% de la ponderación al mejor equipo o estudiante, mientras que los últimos lugares media vez sean totalmente funcionales obtendrán el 60%.

La fase 1 tiene una ponderación de 40 puntos y la fase 2 tiene una ponderación de 60 puntos.

Fecha de entrega fase 1: sábado 19 de junio de 2021

Fecha de entrega fase 2: sábado 3 de julio de 2021