

**INSTITUTO TECNOLOGICO SUPERIOR
DE LA SIEERA NEGRA DE AJALPAN**

**PORTAFOLIO DE EVIDENCIAS
DE
ESTRUCTURA DE DATOS**

**INGENIERIA EN SISTEMAS COMPUTACIONALES
MATERIA: Estructura de Datos
DOCENTE: ARTURO BUSTAMANTE LAZCANO**

**YULI JOLETTE ORTIZ OSORIO
N.CONTROL: 24120167
TERCER SEMESTRE**

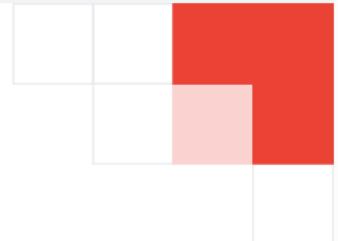
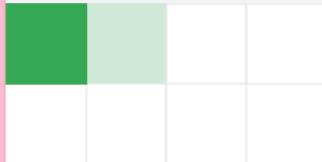
PERIODO AGOSTO-DICIEMBRE 2025



translated by Google

Se usó la [API de Cloud Translation](#) para traducir esta página.

[Switch to English](#)



1

Aprendizaje

Actividades de aprendizaje completadas en el ecosistema de desarrolladores de Google

Compartir

¡Ya tienes esta insignia!

Aprendizaje

[Learn JavaScript](#)

5 dic 2025

Tabla de contenido

Análisis - Semana 1: Introducción a JavaScript y Arreglos.....	5
Análisis - Semana 2: HTML5, Navegación y Árbol Binario de Búsqueda.....	11
Semana3/ Estructura de datos.....	21
Análisis - Semana 4: Tipos de Datos Primitivos y Grafos Dirigidos.....	31
Análisis - Semana 5: JavaScript, JSON y Listas Enlazadas.....	47
Análisis - Semana 6: map() funcional, Programación Orientada a Objetos y Colas (Queue).....	52
Análisis - Semana 7: Tablas Hash, Sets y Pilas (Stacks).....	58
Análisis - Semana 8: Listas Enlazadas (Linked Lists).....	64
Análisis - Semana 9: Repaso de Estructuras Fundamentales (Arrays, Listas, Pilas, Colas, Grafos, Tablas Hash, Árboles).....	69
Análisis - Semana 10: Árboles, Tablas y Estructuras Auxiliares.....	74
Análisis - Semana 11: Aplicación Web de Productos (HTML, CSS, JS, JSON).....	81
Análisis - Semana 12: Estructura de Proyecto Web (Archivos iniciales).....	87
Análisis - Semana 13: Recursión y Algoritmos de Ordenamiento.....	89
Análisis - Semana 14: Proyecto Demo — Objetos, Interactividad y Estilos.....	95
Análisis - Semana 15: Grafos Avanzados, Tablas Hash y Listas Enlazadas.....	99
Análisis - Semana 16: Pilas, Colas y Árboles — Aplicaciones y Práctica.....	104

Análisis - Semana 1: Introducción a JavaScript y Arreglos

Descripción General

Esta carpeta contiene el material introductorio de la materia **Estructuras de Datos**, enfocándose en los conceptos fundamentales de **JavaScript** y la introducción a **estructuras de datos lineales**, específicamente **arreglos (arrays)**.

Lenguaje

- **JavaScript (ES6+)**
- **HTML5**

El proyecto utiliza JavaScript vanilla sin frameworks externos, permitiendo aprender los conceptos fundamentales desde cero.

Estructura de Archivos

Semana1/

```
|── code.js           # Funciones básicas de JS (saludar, FizzBuzz)
|── index.html        # Página HTML principal
|── arrays/
|   └── index.js      # Clase custom de Array con métodos
└── array/
    ├── intro/         # Introducción a métodos de array
    ├── map/            # Método map() - transformación de datos
    ├── filter/          # Método filter() - filtrado de datos
    ├── reduce/          # Método reduce() - reducción de datos
    └── delete/          # Eliminación de elementos
```

```
|--- immutability/      # Conceptos de inmutabilidad  
└--- functionalProgramiming/  # Programación funcional con arrays
```

Objetivos

1. **Introducción a JavaScript:** Aprender la sintaxis básica y cómo integrar JS con HTML
 2. **Resolver problemas lógicos:** Implementar algoritmos clásicos como FizzBuzz
 3. **Entender estructuras de datos:** Crear una clase Array personalizada desde cero
 4. **Métodos de array:** Dominar los métodos funcionales (map, filter, reduce)
 5. **Programación Funcional:** Introducir conceptos de programación funcional con arrays
 6. **Inmutabilidad:** Comprender la importancia de la inmutabilidad en JavaScript
-

Descripción Detallada

Archivos Principales

`code.js`

Archivo introductorio que contiene tres funciones fundamentales:

1. **saludar():** Función básica que muestra un alert de bienvenida
2. **teSaludo(nombre):** Función con parámetro que personaliza el saludo
3. **fizzBuzz():** Implementación del problema clásico "FizzBuzz"
 - Itera números del 1 al 100
 - Imprime "Fizz" para múltiplos de 3
 - Imprime "Buzz" para múltiplos de 5

- Imprime "FizzBuzz" para múltiplos de 3 y 5
- Imprime el número si no cumple las condiciones anteriores

index.html

Página HTML que integra el código JavaScript, proporcionando:

- Interfaz visual con botones interactivos
- Referencias a funciones JavaScript mediante onclick
- Documentación en comentarios del desafío FizzBuzz

arrays/index.js

Implementación personalizada de una clase Array con métodos básicos:

```
class array {
    // Métodos implementados:
    - get(index)      // Obtener elemento por índice
    - push(item)      // Agregar elemento al final
    - pop()          // Eliminar último elemento
    - methodDelete(index) // Eliminar elemento en posición específica
    - shiftIndex(index) // Reorganizar índices después de eliminar
}
```

Carpetas Temáticas

Carpeta	Contenido
intro/	Conceptos básicos de arrays y acceso a elementos
map/	Transformación de elementos usando Array.prototype.map()

Carpeta	Contenido
filter/	Filtrado de elementos con Array.prototype.filter()
reduce/	Reducción de arrays a valores únicos con Array.prototype.reduce()
delete/	Técnicas para eliminar y reorganizar elementos
immutability/	Operaciones sin modificar el array original
functionalProgramiming/	Paradigmas de programación funcional

Análisis Técnico

Conceptos Clave Abordados

1. Fundamentos de JavaScript

- Declaración y llamada de funciones
- Parámetros y argumentos
- Integración HTML-JavaScript

2. Resolución de Problemas

El problema FizzBuzz es un clásico que enseña:

- Control de flujo (if/else)
- Operador módulo (%) para divisibilidad
- Iteración con bucles (for)
- Logging en consola

3. Estructura de Datos Personalizada

La clase Array personalizada demuestra:

- Uso de objetos como contenedores de datos
- Gestión de índices
- Organización de métodos en una clase
- Manipulación de datos mediante referencias

4. Métodos Funcionales de Array

- **map()**: Transformación de datos (1:1)
- **filter()**: Selección condicional de elementos
- **reduce()**: Agregación de datos a un único valor

5. Programación Funcional

- Funciones como ciudadanos de primera clase
- Callbacks y funciones de orden superior
- Evitar mutación de estado



Puntos de Aprendizaje Importantes

1. El problema **FizzBuzz** es un ejercicio fundamental para:

- Evaluar lógica de programación
- Practicar condicionales
- Entender iteración

2. Crear un **Array personalizado** muestra:

- Cómo funcionan internamente las estructuras de datos
- La importancia de la abstracción
- Complejidad de operaciones (push: O(1), delete: O(n))

3. **Métodos funcionales** permiten:

- Código más legible y declarativo

- Evitar mutación de datos
- Composición de operaciones

4. Inmutabilidad es crítica para:

- Predictibilidad del código
- Debugging más fácil
- Funciones puras y testables

Conclusión

Semana 1 establece las bases para el aprendizaje de estructuras de datos mediante:

- Introducción clara a JavaScript
- Implementación manual de estructuras comunes
- Exposición a paradigmas modernos (funcional, inmutable)
- Problemas prácticos y ejercicios interactivos

Este material es ideal para principiantes que desean comprender **cómo funcionan las estructuras de datos desde cero** antes de usar librerías y APIs de nivel superior.

```

File Edit Selection View Go Run Terminal Help
ANALIS SEMANA1.md
estru2025-mast... > ANALIS SEMANA1.md > # Análisis - Semana 1: Introducción a JavaScript y Arreglos > # Estructura de Archivos
1 # Análisis - Semana 1: Introducción a JavaScript y Arreglos "Análisis": Unknown word
2
3 ## ■ Descripción General "Descripción": Unknown word;
4
5 Esta carpeta contiene el material introductorio de la materia **estructuras de datos**, enfocándose en los conceptos fundamentales de **JavaScript** y la introducción a **estructuras de datos lineales**, específicamente **arreglos (arrays)**. "Título": Unknown word
6
7 ...
8
9 ## 🔍 Lenguaje "Lenguaje": Unknown word;
10
11 - **JavaScript (ES6+)** "JavaScript": Unknown word
12 - **HTML5** "HTML5": Unknown word
13
14 El proyecto utiliza JavaScript vanilla sin frameworks externos, permitiendo aprender los conceptos fundamentales desde cero. "Proyecto": Unknown word
15
16 ...
17
18 # Estructura de Archivos "Estructura": Unknown word;
19
20 ...
21 Semanal/ "Semana": Unknown word;
22   code.js Funciones básicas de JS (saludos, FizzBuzz) "Funciones": Unknown word,
23   index.html Página HTML principal "Página": Unknown word
24   arrays/ Implementación personalizada de arreglos "Implementación": Unknown word,
25     index.js Clase custom de Array con métodos "Clase": Unknown word,
26       array/ Temas avanzados de arreglos "Temas": Unknown word,
27         intro/ Introducción a métodos de array "Introducción": Unknown word,
28         filter/ Implementación personalizada de los métodos de array "Método": Unknown word,
29         filter/ Método filter() - filtrado de datos "Método": Unknown word,
30         reduce/ Método reduce() - reducción de datos "Método": Unknown word,
31         delete/ Eliminación de elementos "Eliminación": Unknown word,
32         immutability/ Conceptos de inmutabilidad "Conceptos": Misspelled word,
33           functionalProgramming/ Programación funcional con arrays "Programación": Unknown word,
34
35 ...
36
37 ## 📚 Objetivos "Objetivos": Unknown word;
38
39 1. **Introducción a JavaScript**: Aprender la sintaxis básica y cómo integrar JS con HTML "Introducción": Unknown word.
40 2. **Resolver problemas lógicos**: Implementar algoritmos clásicos como FizzBuzz "Problemas": Misspelled word.
41 3. **Entender estructuras de datos**: Crear una clase Array personalizada desde cero "Entender": Unknown word.
42 4. **Métodos de arrays**: Implementar los métodos de arrays (map, filter, reduce, etc.) "Método": Unknown word.
43 5. **Programación funcional**: Introducir conceptos de programación funcional con arrays "Programación": Unknown word.
44 6. **Inmutabilidad**: Comprende la importancia de la inmutabilidad en JavaScript "Inmutabilidad": Unknown word,
45
46
47 ...
48
49 ## ■ Descripción Detallada "Descripción": Unknown word;
50
51 ### Archivos Principales "Archivos": Unknown word;
52
53 #### 'code.js'
54 Archivo introductorio que contiene tres funciones fundamentales: "Archivo": Unknown word.
55
56 1. **saladar()** Función básica que muestra un alert de bienvenida

```

```

File Edit Selection View Go Run Terminal Help
ANALIS SEMANA1.md
estru2025-mast... > ANALIS SEMANA1.md > # Análisis - Semana 1: Introducción a JavaScript y Arreglos > # Estructura de Archivos
1 # Análisis - Semana 1: Introducción a JavaScript y Arreglos "Análisis": Unknown word
2
3 ## ■ Descripción Detallada "Descripción": Unknown word;
4
5 ### Archivos Principales "Archivos": Unknown word;
6
7 #### 'code.js'
8 Archivo introductorio que contiene tres funciones fundamentales: "Archivo": Unknown word.
9
10
11 #### 'index.html'
12 Página HTML que entrega el código JavaScript, proporcionando:
13   - Una interfaz visual con botones interactivos
14   - Referencia a funciones JavaScript mediante 'onClick'
15   - Documentación en comentarios del desafío FizzBuzz
16
17 #### 'arrays/index.js'
18 Implementación personalizada de una clase Array con métodos básicos:
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
39 #### 'arrays'
40 class array {
41   // Métodos implementados
42   - get(index) // Obtener elemento por índice
43   - push(item) // Agregar elemento al final
44   - pop() // Eliminar último elemento
45   - metodoDelete(index) // Eliminar elemento en posición específica
46   - shiftIndex(index) // Reorganizar índices después de eliminar
47
48
49
50
51
52
53
54
55
56
57
58
59
59 #### Carpeta Técnicas
60 [ Carpeta | Contenido ]
61 [-----]
62 | intro/** | Conceptos básicos de arrays y acceso a elementos |
63 | map/** | Transformación de elementos usando 'Array.prototype.map()' |
64 | filter/** | Filtrado de elementos con 'Array.prototype.filter()' |
65 | reduce/** | Reducción de arrays a valores totales con 'Array.prototype.reduce()' |
66 | delete/** | Técnicas para eliminar y reorganizar elementos |
67 | immutability/** | Operaciones sin modificar el array original |
68 | functionalProgramming/** | Paradigmas de programación funcional |
69
70
71
72
73
74
75
76
77
78
79
79 #### Análisis Técnico
80
81
82
83
84
85
86
87
88
89
89 #### Conceptos Clave Abordados
90
91
92
93
93 #### 1. **Fundamentos de JavaScript**
94 - Declaración y llamada de funciones
95 - Parámetros y argumentos
96 - Integración HTML-JavaScript
97
98 #### 2. **Resolución de Problemas**
99 El problema FizzBuzz es un clásico que enseña:
100 - Control de flujo (if/else)
101 - Operador módulo (%) para divisibilidad
102 - Iteración con bucles (for)
103 - Logging en consola
104
105
106
107
108
109
110
111
112
113

```

Análisis - Semana 2: HTML5, Navegación y Árbol Binario de Búsqueda

Descripción General

Esta carpeta contiene material sobre **HTML5 fundamental, navegación entre páginas web** y una introducción a **estructuras de datos no lineales** como el **Árbol Binario de Búsqueda (BST)**. Representa la transición del curso desde JavaScript puro hacia aplicaciones web más complejas con interfaces HTML y estructuras de datos avanzadas.

Lenguaje

- **HTML5**
- **JavaScript (ES6+)**
- **CSS3** (con Bootstrap 5.3.2)

El material combina HTML5 vanilla con JavaScript orientado a objetos, introduciendo también un framework CSS (Bootstrap) para diseño responsivo.

Estructura de Archivos

```
Semana2/
├── index.html          # Página principal - Tutorial HTML5
├── acerca.html         # Página "Acerca de" - Ejemplo de
  navegación
├── nosotros.html       # Página "Nosotros" - Ejemplo de
  navegación
├── HTML inicia con la etiqueta !DOCTYPE.txt # Documento de referencia
├── assets/              # Recursos estáticos (imágenes, etc.)
└── binarySearchTree/
    ├── index.html      # Página HTML para BST
```

```
|   └── main.js          # Clase Node y BinarySearchTree  
├── misitio/           # Proyecto de sitio web con Bootstrap  
|   ├── index.html      # Página principal con Bootstrap  
|   └── code.js         # Funciones JavaScript para interactividad  
└── tarea/             # Tarea práctica sobre Rock Ingles  
    ├── index.html      # Página principal del tema  
    ├── conciertos.html # Página de conciertos  
    └── fotos.html       # Página de fotos
```

Objetivos

1. **Dominar HTML5:** Entender la estructura semántica completa de un documento HTML
 2. **Navegación web:** Crear sitios multi-página con enlaces internos efectivos
 3. **Elementos HTML básicos:** Trabajar con headings, párrafos, listas y formatos
 4. **Introducción a Bootstrap:** Usar frameworks CSS para diseño responsive
 5. **Árbol Binario de Búsqueda (BST):** Implementar una estructura de datos no lineal fundamental
 6. **Operaciones en BST:** Insert, search y recorridos (in-order, pre-order, post-order)
-

Descripción Detallada

Archivos Principales

index.html (Tutorial HTML5)

Página completa que demuestra elementos HTML fundamentales:

- **Headings:** Uso de <h1> a <h6> para jerarquía de títulos
- **Párrafos y saltos:** Etiquetas <p>,
 para estructura de contenido
- **Listas:**
 - Ordenadas () para listas numeradas
 - Desordenadas () para listas con viñetas
- **Formato preformatado:** <pre> para preservar espacios y saltos
- **Otras etiquetas:** Ejemplos de enlaces, imágenes, tablas y formatos (bold, italic, etc.)

acerca.html y nosotros.html

Ejemplos de navegación básica con:

- Estructura completa de página HTML
- Menú de navegación con enlaces internos
- Meta tags para responsividad

HTML inicia con la etiqueta !DOCTYP.txt

Documento de referencia sobre:

- Declaración DOCTYPE para HTML5
- Importancia de meta tags (charset, viewport)
- Estructura básica de un documento HTML válido

Carpeta binarySearchTree/

main.js - Implementación Completa de BST

Clase Node:

```
class Node {
  constructor(value) {
    this.value = value;
    this.left = null;
  }
}
```

```
    this.right = null;  
}  
}  
}
```

Clase BinarySearchTree:

1. **insert(value)**: Inserta un nodo en la posición correcta
 - o Compara el valor con el nodo actual
 - o Navega hacia la izquierda si es menor
 - o Navega hacia la derecha si es mayor
 - o Complejidad: $O(\log n)$ promedio, $O(n)$ peor caso

2. **search(value)**: Busca un nodo por valor
 - o Retorna el nodo si lo encuentra
 - o Retorna false si no existe
 - o Complejidad: $O(\log n)$ promedio

3. Recorridos:

- o **showInOrder()**: Left \rightarrow Root \rightarrow Right (orden ascendente)
- o **showInPreOrder()**: Root \rightarrow Left \rightarrow Right
- o **showInPostOrder()**: Left \rightarrow Right \rightarrow Root (implícito en el código)

index.html

Estructura básica para conectar la lógica de JavaScript:

```
<script defer src='main.js'></script>
```

Carpeta misitio/

index.html - Proyecto Web con Bootstrap

Demuestra:

- Integración de Bootstrap 5.3.2 CDN

- Estructura responsive con grid system
- Componentes: Header, navbar colapsable, secciones
- Uso de clases Bootstrap para estilos predefinidos
- Colores temáticos (#212429 para fondo oscuro)

code.js

Contiene función botones() para interactividad (estructura base).

Carpeta tarea/

Tarea Práctica: Historia del Rock Ingles

Archivos de ejemplo mostrando:

- Contenido educativo sobre rock inglés
- **Listas no ordenadas:** Bandas iconicas (Beatles, Rolling Stones, Led Zeppelin, etc.)
- **Listas ordenadas:** Álbumes esenciales con ranking
- Estructura navegable entre index.html, conciertos.html, fotos.html
- Buena práctica de organización de contenido temático

Análisis Técnico

1. Conceptos de HTML5

Estructura Semántica

- DOCTYPE declaration: <!DOCTYPE html> (HTML5 simplificado)
- Meta tags críticos: charset, viewport para responsividad
- Headings para jerarquía de contenido (h1 > h6)

Elementos de Contenido

- <p>: Párrafos con espaciado automático
- <pre>: Preserva formato, útil para código
-
: Salto de línea explícito

- /: Listas ordenadas/desordenadas
- <a>: Enlaces internos y externos
- <nav>: Navegación semántica

2. Navegación Web Multi-página

El patrón implementado:

```
<nav>
  <ul>
    <li><a href="index.html">Inicio</a></li>
    <li><a href="nosotros.html">Nosotros</a></li>
    <li><a href="acerca.html">Acerca de</a></li>
  </ul>
</nav>
```

Ventajas:

- Navegación consistente entre páginas
- Rutas relativas facilitan mantenimiento
- Estructura escalable para sitios más grandes

3. Árbol Binario de Búsqueda (BST)

Propiedad Fundamental

Para cada nodo:

- Todos los nodos en el **subtree izquierdo** < valor del nodo
- Todos los nodos en el **subtree derecho** > valor del nodo

Complejidad de Operaciones

Operación	Mejor Caso	Peor Caso	Promedio
Insert	O(log n)	O(n)	O(log n)

Operación	Mejor Caso	Peor Caso	Promedio
Search	$O(\log n)$	$O(n)$	$O(\log n)$
Delete	$O(\log n)$	$O(n)$	$O(\log n)$

El peor caso ocurre cuando el árbol es degenerado (lista enlazada).

Recorridos

1. **In-Order:** Left → Root → Right
 - Produce elementos en orden ascendente
 - Útil para obtener datos ordenados
2. **Pre-Order:** Root → Left → Right
 - Procesa el nodo antes de sus hijos
 - Útil para copiar el árbol
3. **Post-Order:** Left → Right → Root
 - Procesa el nodo después de sus hijos
 - Útil para liberar memoria

4. Bootstrap 5.3.2

Framework CSS que proporciona:

- **Grid System:** Layouts responsivos con filas y columnas
- **Componentes:** Navbars, cards, buttons, etc.
- **Utilidades:** Clases para spacing, colores, tipografía
- **Responsividad:** Breakpoints para diferentes dispositivos

Ventajas:

- Desarrollo rápido sin CSS custom
- Consistencia visual
- Reducción de código redundante

5. Organización Pedagógica

La carpeta Semana 2 es un puente entre:

- **Conceptos básicos:** HTML5 y navegación
 - **Estructuras avanzadas:** BST
 - **Herramientas modernas:** Bootstrap
 - **Aplicaciones prácticas:** Proyectos temáticos (Rock, etc.)
-



Puntos de Aprendizaje Importantes

1. **HTML5 es la base:** Entender estructura semántica es crítico para accesibilidad y SEO
 2. **Navegación escalable:** El patrón multi-página facilita mantener sitios grandes
 3. **BST es fundamental:** Muchas estructuras avanzadas (AVL, Red-Black Trees) se construyen sobre BST
 4. **Recorridos recursivos:** Demuestran la elegancia de algoritmos recursivos vs iterativos
 5. **Frameworks CSS aceleren desarrollo:** Bootstrap ejemplifica cómo abstracciones facilitan la creación web
 6. **Proyectos prácticos refuerzan:** La tarea del Rock Inglés conecta teoría con aplicación real
-



Conclusión

Semana 2 es transicional y fundamental porque:

- ✓ **Consolida HTML5:** Desde estructura básica hasta navegación compleja
- ✓ **Introduce BST:** Primera estructura de datos no lineal, esencial para algoritmos
- ✓ **Integra herramientas modernas:** Bootstrap prepara para desarrollo web profesional

 **Aplica conocimiento:** Proyectos prácticos (Rock, sitios web) muestran utilidad real

 **Sienta bases:** Prepara para semanas posteriores sobre estructuras más complejas (Grafos, Árboles N-arios, etc.)

Este material es ideal para estudiantes que ya conocen HTML básico y quieren profundizar en aplicaciones web reales y estructuras de datos fundamentales.

Análisis - Semana 3: Bootstrap Avanzado, Listas Dblemente Enlazadas y Grafos

Descripción General

Semana 3 profundiza en **Bootstrap 5 avanzado** con énfasis en **sistemas de grid responsivo** y presenta dos estructuras de datos fundamentales: **Listas Dblemente Enlazadas (Doubly Linked Lists)** y **Grafos (Graphs)**. Este material marca un salto significativo en complejidad de estructuras de datos no lineales.

Lenguaje

- **HTML5**
- **CSS3 (Bootstrap 5 + Media Queries)**
- **JavaScript (ES6+)** con Programación Orientada a Objetos

Se introduce por primera vez el uso extensivo de **media queries** para crear diseños verdaderamente responsivos.

The screenshot shows a Microsoft Edge browser window with the following details:

- Title Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Address Bar:** estructura2025-master
- Left Sidebar (EXPLORER):**
 - ESTRUCTURA2025-MASTER
 - estructura2025-master
 - Semana 11
 - Semana 12
 - Semana1
 - Semana2
 - assets
 - binarySearchTree
 - mistido
 - tarea
 - acerca.html
 - ANALISIS_SEMANA2.md
 - E: HTML_inicia con la etiqueta (DOCTYP.txt
 - index.html
 - nosotros.html
 - Semanas
 - Semanario
 - Semana5
 - Semana6
 - Semana7
 - Semana8
 - Semana9
 - Semana10
 - Semana11
 - Semana12
 - Semana13
 - Semana14
 - Semanario15
 - graph
 - hashtable
 - linkedList
 - ANALYSIS_SEMANA15.md
 - js
 - code.js
 - Semanario16
 - queue
 - stack
 - tree
 - ANALYSIS_SEMANA16.md
 - index.css
 - index.html
 - index.js
 - notas.md
- Central Content Area:** A code editor showing the content of the file ANALISIS_SEMANA2.md. The file contains sections like "ANALISIS SEMANA2nd", "Descripción General", "Estructura de Archivos", "Objetivos", and "Descripción Detallada". It also includes several code snippets and explanatory text about HTML, CSS, and JavaScript concepts.
- Right Sidebar:** Shows a list of recent files and a "Markdown" section.
- Bottom Status Bar:** Ln 12, Col 24, Spaces: 3, UTT-B, CRLF, Markdown, Go Live, Go Live.

Estructura de Archivos

Semana3/

```
|── index.html          # Tutorial Bootstrap Grid System  
|── mistyle.css        # Estilos responsivos con media queries  
|── doublyLinkedList/  
|     enlazada          # Implementación de lista doblemente  
|         |── index.html    # Interfaz para navegación de películas  
|         |── main.js       # Clase DoublyLinkedList con métodos  
|         |── images/        # Recursos de imágenes  
|── graphs/             # Introducción a teoría de grafos  
|         |── graphs.js      # Representaciones de grafos (edge list,  
|         |     adjacency list/matrix)  
|         |── buildGraph.js   # Clase Graph con métodos básicos
```

Objetivos

1. **Dominar Bootstrap Grid System:** Crear layouts responsivos con contenedores, filas y columnas
 2. **Media Queries avanzadas:** Implementar diseños que se adapten a múltiples breakpoints
 3. **Listas Dblemente Enlazadas:** Entender nodos con referencias bidireccionales (next y prev)
 4. **Operaciones en DLL:** Implementar add, delete, reverse, show y clear
 5. **Introducción a Grafos:** Comprender vértices, aristas y diferentes representaciones
 6. **Estructuras de Grafos:** Implementar lista de adyacencia y matriz de adyacencia
-

Descripción Detallada

Archivos Principales

index.html - Tutorial Completo de Bootstrap Grid

Demuestra **6 tipos de contenedores Bootstrap**:

1. container:

- 100% wide con breakpoints
- Máximo ancho predefinido en cada breakpoint
- Uso: contenedores tradicionales

2. container-fluid:

- 100% del ancho siempre
- Sin restricción de max-width
- Uso: layouts de ancho completo

3. container-sm, container-md, container-lg, container-xl, container-xxl:

- 100% wide hasta el breakpoint especificado
- Ejemplos:
 - container-sm: $\geq 576\text{px}$ (small)
 - container-md: $\geq 768\text{px}$ (medium)
 - container-lg: $\geq 992\text{px}$ (large)
 - container-xl: $\geq 1200\text{px}$ (extra large)
 - container-xxl: $\geq 1400\text{px}$ (extra extra large)

Sistema Grid Row & Col:

```
<div class="container">  
  <div class="row">  
    <div class="col">Columna 1</div>  
    <div class="col">Columna 2</div>
```

```
</div>
```

```
</div>
```

Características:

- **Filas (row)**: Contenedores de 12 columnas
- **Columnas (col)**: Se distribuyen automáticamente
- **col-sm-6, col-md-4**: Especificar tamaño en breakpoints

mistyle.css - Media Queries Responsivas

Implementa cambios de color según el tamaño de pantalla:

```
/* xs (por defecto, <576px) - verde */
```

```
.responsive-bg { background-color: var(--bs-success); }
```

```
/* sm (≥576px) - azul primario */
```

```
@media (min-width: 576px) { ... }
```

```
/* md (≥768px) - gris secundario */
```

```
@media (min-width: 768px) { ... }
```

```
/* lg (≥992px) - rojo peligro */
```

```
@media (min-width: 992px) { ... }
```

```
/* xl (≥1200px) - amarillo alerta */
```

```
@media (min-width: 1200px) { ... }
```

```
/* xxl (≥1400px) - azul info */
```

```
@media (min-width: 1400px) { ... }
```

Breakpoints de Bootstrap 5:

Breakpoint	Dispositivo	Min-Width
xs	Móvil pequeño	< 576px
sm	Móvil	≥ 576px
md	Tablet	≥ 768px
lg	Desktop pequeño	≥ 992px
xl	Desktop	≥ 1200px
xxl	Desktop grande	≥ 1400px

Carpeta doublyLinkedList/

main.js - Implementación de Lista Dblemente Enlazada

Clase Node:

```
class Node {  
    this.value = value; // Dato almacenado  
    this.next = null; // Referencia al siguiente nodo  
    this.prev = null; // Referencia al nodo anterior  
}
```

Clase DoublyLinkedList:

Método	Descripción	Complejidad
add(value)	Agregar elemento al final	O(1)
show()	Mostrar lista hacia adelante	O(n)
reverse()	Mostrar lista hacia atrás	O(n)

Método	Descripción	Complejidad
clear()	Limpiar toda la lista	O(1)
delete(value)	Eliminar nodo por valor	O(n)

Características principales:

- Head y Tail:** Apuntadores al inicio y fin
- Bidireccionalidad:** Navegación adelante y atrás
- Búsqueda optimizada:** Puede iniciar desde cualquier extremo
- Eliminación eficiente:** No requiere reorganizar índices

Ventajas sobre Singly Linked List:

- Navegación en ambas direcciones
- Búsqueda desde ambos extremos
- Inserción/eliminación más eficiente
- Usado en navegadores (historial back/forward)

index.html - Interfaz de Navegación de Películas

Interfaz simple que demuestra uso práctico de DLL:

```
<button onclick="prevMovie();">Prev</button>
<button onclick="nextMovie();">Next</button>
<h4 id="title"></h4>
<img id="image" style="width: 100px;">
```

Caso de uso: Navegar entre películas con botones Anterior/Siguiente, típicamente usando DLL internamente.

Carpeta graphs/

graphs.js - Representaciones de Grafos

Muestra 4 formas de representar el mismo grafo:

1. Edge List (Lista de Aristas):

```
const graph = [  
    [0, 2],  
    [2, 3],  
    [2, 1],  
    [1, 2]  
];  
  
• Simple, pero ineficiente para búsquedas  
• Complejidad de búsqueda:  $O(E)$  donde  $E$  = aristas
```

2. Adjacency List (Lista de Adyacencia) - Array:

```
const graph = [[2], [2, 3], [0, 1, 3], [1, 2]];  
  
• Índice = vértice, valor = lista de vecinos  
• Complejidad de búsqueda:  $O(V + E)$ 
```

3. Adjacency List - Objeto:

```
const graph = {  
    0: [2],  
    1: [2, 3],  
    2: [0, 1, 3],  
    3: [1, 2]  
};  
  
• Más flexible, permite claves de cualquier tipo  
• Complejidad:  $O(1)$  acceso a vértice
```

4. Adjacency Matrix (Matriz de Adyacencia):

```
const graph = [  
    [0, 0, 1, 0],  
    [0, 0, 1, 0],  
    [0, 0, 0, 1],  
    [0, 0, 1, 0]
```

- ```

[0, 0, 1, 1],
[1, 1, 0, 1],
[0, 1, 1, 0]
];

```
- $\text{graph}[i][j] = 1$  si existe arista entre i y j
  - Complejidad de búsqueda:  $O(1)$
  - Usa más memoria para grafos dispersos

### Comparación de Representaciones:

| Representación     | Búsqueda de Arista | Espacio  | Uso                  |
|--------------------|--------------------|----------|----------------------|
| Edge List          | $O(E)$             | $O(E)$   | Algoritmos complejos |
| Adj. List (Array)  | $O(V+E)$           | $O(V+E)$ | Grafos típicos       |
| Adj. List (Objeto) | $O(1)$ promedio    | $O(V+E)$ | Grafos dinámicos     |
| Adj. Matrix        | $O(1)$             | $O(V^2)$ | Grafos densos        |

### buildGraph.js - Clase Graph

Implementa un grafo no dirigido usando lista de adyacencia:

```

class graph {
 constructor() {
 this.nodes = 0; // Contador de vértices
 this.adjacentList = {};// Lista de adyacencia
 }
}

```

```

addVertex(node) {
 this.adjacentList[node] = [];
 this.nodes++;
}

addEdge(node1, node2) {
 this.adjacentList[node1].push(node2);
 this.adjacentList[node2].push(node1); // No dirigido
}

```

**Ejemplo de uso:**

```

const myGraph = new graph();

myGraph.addVertex("1");
myGraph.addVertex("3");
myGraph.addVertex("4");
myGraph.addVertex("5");
myGraph.addVertex("6");
myGraph.addVertex("8");
myGraph.addEdge("1", "6");
myGraph.addEdge("6", "3");
myGraph.addEdge("3", "5");
myGraph.addEdge("4", "5");
// ... más aristas

```

## 1. Bootstrap Grid System Responsivo

**Concepto Core:** Bootstrap divide el ancho en 12 columnas. Las clases col-\* especifican cuántas columnas ocupa cada elemento.

**Ejemplo práctico:**

```
<div class="row">
 <div class="col-md-4">33.33% en MD+</div>
 <div class="col-md-8">66.66% en MD+</div>
</div>
```

**Ventajas:**

- Mobile-first approach
- Breakpoints predefinidos
- Proporcional y flexible
- Facilita mantenimiento

## 2. Media Queries y Diseño Responsivo

Los media queries permiten aplicar estilos según condiciones:

```
/* Móvil (por defecto) */
.box { font-size: 12px; }

/* Tablet y superior */
 @media (min-width: 768px) {
 .box { font-size: 14px; }
}

/* Desktop y superior */
 @media (min-width: 992px) {
```

```
.box { font-size: 16px; }
}
```

### Filosofía Mobile-First:

1. Definir estilos base para móvil
2. Usar min-width para dispositivos más grandes
3. Progresiva mejora de experiencia

### 3. Listas Dblemente Enlazadas vs Simplemente Enlazadas

Característica	Singly	Doubly
Navegación	Adelante	Ambas direcciones
Memoria por nodo	1 puntero	2 punteros
Búsqueda	$O(n)$ desde inicio	$O(n/2)$ promedio
Eliminación	Requiere nodo anterior	Direct
Casos de uso	Pilas, colas	Navegadores, editores

### Complejidad de operaciones en DLL:

- Insert al inicio:  $O(1)$
- Insert al final:  $O(1)$
- Delete:  $O(n)$  búsqueda +  $O(1)$  eliminación
- Reverse iteration:  $O(n)$

### 4. Teoría de Grafos Fundamental

#### Terminología:

- **Vértice (Nodo):** Punto en el grafo

- **Arista (Edge)**: Conexión entre dos vértices
- **Grado**: Número de aristas conectadas a un vértice
- **Grafo Dirigido**: Las aristas tienen dirección
- **Grafo No Dirigido**: Las aristas son bidireccionales

### **Aplicaciones Reales:**

- Redes sociales (amigos = aristas)
- Navegación GPS (ciudades = vértices, caminos = aristas)
- Recomendaciones (usuarios → productos)
- Sistemas de transportes (estaciones = vértices)

## **5. Selección de Representación de Grafo**

### **Criterios:**

- **Grafo disperso** (pocas aristas): Lista de adyacencia
- **Grafo denso** (muchas aristas): Matriz de adyacencia
- **Búsquedas frecuentes**: Matriz de adyacencia  $O(1)$
- **Espacio limitado**: Lista de adyacencia  $O(V+E)$



### **Puntos de Aprendizaje Importantes**

1. **Bootstrap facilita diseño responsive**: Abstacta complejidad de media queries
2. **Mobile-first es crítico**: Los dispositivos móviles son mayoría de usuarios
3. **DLL es puente entre arrays y grafos**: Introduce referencias bidireccionales
4. **Grafos son ubicuos**: La mayoría de problemas reales involucran grafos
5. **Elección de representación importa**: Afecta complejidad y uso de memoria

## 6. Practicalidad de DLL: Aparecen en navegadores, editores de texto, reproductor multimedia

### 🎓 Conclusión

Semana 3 es **transicional y reveladora** porque:

- ✓ **Consolida Bootstrap:** Desde grid básico hasta responsividad completa
- ✓ **Introduce grafos:** Estructura fundamental para problemas complejos
- ✓ **Conecta conceptos:** DLL prepara para traversals de grafos
- ✓ **Expande horizonte:** Desde datos lineales a no lineales
- ✓ **Aplicaciones prácticas:** Media queries, navegación, sistemas de recomendación

Este material es ideal para estudiantes preparándose para **algoritmos avanzados** (BFS, DFS, Dijkstra) que dependen de estructuras de grafos sólidas. La combinación de Bootstrap responsive y estructuras de datos crea una base para desarrollo web moderno.

```
ANALISIS_SEMANA3.md
estructura2025-master > Semana3 > ANALISIS_SEMANA3.md > # Análisis - Semana 3: Bootstrap Avanzado, Listas Dblemente Enlazadas y Grafos > ## Estructura de Archivos
1. # Análisis - Semana 3: Bootstrap Avanzado, Listas Dblemente Enlazadas y Grafos "Análisis": Unknown word.
2.
3. ## Descripción General "Descripción": Unknown word.
4.
5. Semana 3 profundiza en **Bootstrap 5 avanzado** con énfasis en **sistemas de grid responsivos** y presenta dos estructuras de datos fundamentales: **Listas Dblemente Enlazadas (Doubly Linked Lists)** y **Grafos (Graphs)**. Este material marca un salto significativo en complejidad de estructuras de datos no lineales. "Semana": Unknown word.
6.
7. ...
8. # Lenguaje "Lenguaje": Unknown word.
9.
10. - **HTML5**#
11. - **CSS3**# (Bootstrap 5 + Media Queries)
12. - **JavaScript (ES6+)** con Programación Orientada a Objetos "Programación": Unknown word.
13.
14. Se introduce por primera vez el uso extensivo de **media queries** para crear diseños verdaderamente responsivos. "primera": Unknown word.
15.
16. ...
17. ...
18. ## Estructura de Archivos "Estructura": Unknown word.
19.
20. ...
21. ...
22. Semana3/ "Semana": Unknown word.
23. ---index.html # Tutorial Bootstrap Grid system
24. ---mystyle.css # Estilos responsive y media queries. "mystyle": Unknown word.
25. ---doublyLinkedList/ # Implementación de lista doblemente enlazada "implementación": Unknown word.
26. ---index.html # Interfaz para navegación de películas. "Interfaz": Unknown word.
27. ---main.js # Clase DoublyLinkedList con métodos. "Clase": Misspelled word.
28. ---graph/ # Recursos de imágenes "Recursos": Unknown word.
29. ---images/ # Introducción a teoría de grafos "Introducción": Unknown word.
30. ---graphs.js # Representaciones de grafos (edge list, adjacency list/matrix) "Representaciones": Misspelled word.
31. ---buildGraph.js # Clase Graph con métodos básicos "Clase": Misspelled word.
32. ...
33.
34. ...
35.
36. ## Objetivos "Objetivos": Unknown word.
37.
38. 1. "Desarrollar Bootstrap Grid System": Crear layouts responsivos con contenedores, filas y columnas. "Desarrollar": Unknown word.
39. 2. "Media Queries avanzadas": Implementar diseños que se adaptan a múltiples breakpoints "avanzadas": Unknown word.
40. 3. "Listas Dblemente Enlazadas": Implementar listas doblemente enlazadas (next y previous). "dblemente": Unknown word.
41. 4. "Operaciones en una DLL": Implementar add, delete, reverse, show y clear. "Operaciones": Unknown word.
42. 5. "Introducción a Grafos": Comprender vértices, aristas y diferentes representaciones "Introducción": Unknown word.
43. 6. "Estructuras de grafos": Implementar lista de adyacencia y matriz de adyacencia "Estructuras": Unknown word.
44.
45.
46.
47. ## Descripción Detallada "Descripción": Unknown word.
48.
49. ## Archivos Principales "Archivos": Unknown word.
50.
51. #### 'index.html' - Tutorial Completo de Bootstrap Grid
52.
53. Demuestra **tipos de contenedores Bootstrap**:
54.
55. 1. **container**:
56. 100% width con breakpoints
57.
58.
59.
60.
61.
62.
63.
64.
65.
66.
67.
68.
69.
69.
70.
71.
72.
73.
74.
75.
76.
77.
78.
79.
79.
80.
81.
82.
83.
84.
85.
86.
87.
88.
89.
89.
90.
91.
92.
93.
94.
95.
96.
97.
98.
99.
99.
100.
100.
101.
102.
103.
104.
105.
106.
107.
108.
109.
109.
110.
111.
112.
113.
114.
115.
116.
117.
118.
119.
119.
120.
121.
122.
123.
124.
125.
126.
127.
128.
129.
129.
130.
131.
132.
133.
134.
135.
136.
137.
138.
139.
140.
141.
142.
143.
144.
145.
146.
147.
148.
149.
150.
151.
152.
153.
154.
155.
156.
157.
158.
159.
159.
160.
161.
162.
163.
164.
165.
166.
167.
168.
169.
169.
170.
171.
172.
173.
174.
175.
176.
177.
178.
179.
179.
180.
181.
182.
183.
184.
185.
186.
187.
188.
189.
189.
190.
191.
192.
193.
194.
195.
196.
197.
198.
199.
199.
200.
201.
202.
203.
204.
205.
206.
207.
208.
209.
209.
210.
211.
212.
213.
214.
215.
216.
217.
218.
219.
219.
220.
221.
222.
223.
224.
225.
226.
227.
228.
229.
229.
230.
231.
232.
233.
234.
235.
236.
237.
237.
238.
239.
239.
240.
241.
242.
243.
244.
245.
246.
247.
247.
248.
249.
249.
250.
251.
252.
253.
254.
255.
256.
256.
257.
258.
258.
259.
259.
260.
261.
262.
263.
264.
265.
265.
266.
267.
267.
268.
268.
269.
269.
270.
271.
272.
273.
274.
275.
275.
276.
277.
277.
278.
278.
279.
279.
280.
281.
282.
283.
284.
285.
285.
286.
287.
287.
288.
288.
289.
289.
290.
291.
292.
293.
294.
295.
295.
296.
297.
297.
298.
298.
299.
299.
300.
301.
302.
303.
304.
305.
305.
306.
307.
307.
308.
308.
309.
309.
310.
311.
312.
313.
314.
315.
315.
316.
317.
317.
318.
318.
319.
319.
320.
321.
322.
323.
324.
325.
325.
326.
327.
327.
328.
328.
329.
329.
330.
331.
332.
333.
334.
335.
335.
336.
337.
337.
338.
338.
339.
339.
340.
341.
342.
343.
344.
344.
345.
346.
346.
347.
347.
348.
348.
349.
349.
350.
351.
352.
353.
353.
354.
355.
355.
356.
356.
357.
357.
358.
358.
359.
359.
360.
361.
362.
363.
364.
364.
365.
366.
366.
367.
367.
368.
368.
369.
369.
370.
371.
372.
373.
374.
374.
375.
376.
376.
377.
377.
378.
378.
379.
379.
380.
381.
382.
383.
384.
384.
385.
386.
386.
387.
387.
388.
388.
389.
389.
390.
391.
392.
393.
394.
394.
395.
396.
396.
397.
397.
398.
398.
399.
399.
400.
401.
402.
403.
403.
404.
405.
405.
406.
406.
407.
407.
408.
408.
409.
409.
410.
411.
412.
412.
413.
413.
414.
414.
415.
415.
416.
416.
417.
417.
418.
418.
419.
419.
420.
420.
421.
421.
422.
422.
423.
423.
424.
424.
425.
425.
426.
426.
427.
427.
428.
428.
429.
429.
430.
431.
432.
432.
433.
433.
434.
434.
435.
435.
436.
436.
437.
437.
438.
438.
439.
439.
440.
440.
441.
441.
442.
442.
443.
443.
444.
444.
445.
445.
446.
446.
447.
447.
448.
448.
449.
449.
450.
450.
451.
451.
452.
452.
453.
453.
454.
454.
455.
455.
456.
456.
457.
457.
458.
458.
459.
459.
460.
460.
461.
461.
462.
462.
463.
463.
464.
464.
465.
465.
466.
466.
467.
467.
468.
468.
469.
469.
470.
470.
471.
471.
472.
472.
473.
473.
474.
474.
475.
475.
476.
476.
477.
477.
478.
478.
479.
479.
480.
480.
481.
481.
482.
482.
483.
483.
484.
484.
485.
485.
486.
486.
487.
487.
488.
488.
489.
489.
490.
490.
491.
491.
492.
492.
493.
493.
494.
494.
495.
495.
496.
496.
497.
497.
498.
498.
499.
499.
500.
500.
501.
501.
502.
502.
503.
503.
504.
504.
505.
505.
506.
506.
507.
507.
508.
508.
509.
509.
510.
510.
511.
511.
512.
512.
513.
513.
514.
514.
515.
515.
516.
516.
517.
517.
518.
518.
519.
519.
520.
520.
521.
521.
522.
522.
523.
523.
524.
524.
525.
525.
526.
526.
527.
527.
528.
528.
529.
529.
530.
530.
531.
531.
532.
532.
533.
533.
534.
534.
535.
535.
536.
536.
537.
537.
538.
538.
539.
539.
540.
540.
541.
541.
542.
542.
543.
543.
544.
544.
545.
545.
546.
546.
547.
547.
548.
548.
549.
549.
550.
550.
551.
551.
552.
552.
553.
553.
554.
554.
555.
555.
556.
556.
557.
557.
558.
558.
559.
559.
560.
560.
561.
561.
562.
562.
563.
563.
564.
564.
565.
565.
566.
566.
567.
567.
568.
568.
569.
569.
570.
570.
571.
571.
572.
572.
573.
573.
574.
574.
575.
575.
576.
576.
577.
577.
578.
578.
579.
579.
580.
580.
581.
581.
582.
582.
583.
583.
584.
584.
585.
585.
586.
586.
587.
587.
588.
588.
589.
589.
590.
590.
591.
591.
592.
592.
593.
593.
594.
594.
595.
595.
596.
596.
597.
597.
598.
598.
599.
599.
600.
600.
601.
601.
602.
602.
603.
603.
604.
604.
605.
605.
606.
606.
607.
607.
608.
608.
609.
609.
610.
610.
611.
611.
612.
612.
613.
613.
614.
614.
615.
615.
616.
616.
617.
617.
618.
618.
619.
619.
620.
620.
621.
621.
622.
622.
623.
623.
624.
624.
625.
625.
626.
626.
627.
627.
628.
628.
629.
629.
630.
630.
631.
631.
632.
632.
633.
633.
634.
634.
635.
635.
636.
636.
637.
637.
638.
638.
639.
639.
640.
640.
641.
641.
642.
642.
643.
643.
644.
644.
645.
645.
646.
646.
647.
647.
648.
648.
649.
649.
650.
650.
651.
651.
652.
652.
653.
653.
654.
654.
655.
655.
656.
656.
657.
657.
658.
658.
659.
659.
660.
660.
661.
661.
662.
662.
663.
663.
664.
664.
665.
665.
666.
666.
667.
667.
668.
668.
669.
669.
670.
670.
671.
671.
672.
672.
673.
673.
674.
674.
675.
675.
676.
676.
677.
677.
678.
678.
679.
679.
680.
680.
681.
681.
682.
682.
683.
683.
684.
684.
685.
685.
686.
686.
687.
687.
688.
688.
689.
689.
690.
690.
691.
691.
692.
692.
693.
693.
694.
694.
695.
695.
696.
696.
697.
697.
698.
698.
699.
699.
700.
700.
701.
701.
702.
702.
703.
703.
704.
704.
705.
705.
706.
706.
707.
707.
708.
708.
709.
709.
710.
710.
711.
711.
712.
712.
713.
713.
714.
714.
715.
715.
716.
716.
717.
717.
718.
718.
719.
719.
720.
720.
721.
721.
722.
722.
723.
723.
724.
724.
725.
725.
726.
726.
727.
727.
728.
728.
729.
729.
730.
730.
731.
731.
732.
732.
733.
733.
734.
734.
735.
735.
736.
736.
737.
737.
738.
738.
739.
739.
740.
740.
741.
741.
742.
742.
743.
743.
744.
744.
745.
745.
746.
746.
747.
747.
748.
748.
749.
749.
750.
750.
751.
751.
752.
752.
753.
753.
754.
754.
755.
755.
756.
756.
757.
757.
758.
758.
759.
759.
760.
760.
761.
761.
762.
762.
763.
763.
764.
764.
765.
765.
766.
766.
767.
767.
768.
768.
769.
769.
770.
770.
771.
771.
772.
772.
773.
773.
774.
774.
775.
775.
776.
776.
777.
777.
778.
778.
779.
779.
780.
780.
781.
781.
782.
782.
783.
783.
784.
784.
785.
785.
786.
786.
787.
787.
788.
788.
789.
789.
790.
790.
791.
791.
792.
792.
793.
793.
794.
794.
795.
795.
796.
796.
797.
797.
798.
798.
799.
799.
800.
800.
801.
801.
802.
802.
803.
803.
804.
804.
805.
805.
806.
806.
807.
807.
808.
808.
809.
809.
810.
810.
811.
811.
812.
812.
813.
813.
814.
814.
815.
815.
816.
816.
817.
817.
818.
818.
819.
819.
820.
820.
821.
821.
822.
822.
823.
823.
824.
824.
825.
825.
826.
826.
827.
827.
828.
828.
829.
829.
830.
830.
831.
831.
832.
832.
833.
833.
834.
834.
835.
835.
836.
836.
837.
837.
838.
838.
839.
839.
840.
840.
841.
841.
842.
842.
843.
843.
844.
844.
845.
845.
846.
846.
847.
847.
848.
848.
849.
849.
850.
850.
```

The screenshot shows a terminal window with a file browser sidebar on the left and a code editor on the right.

**File Explorer (Left):**

- ESTRUCTURAS2025-MASTER
- estructuras2025-2025-master
- Semana 11
- Semana 12
- Semana1
- Semana2
- Semana3
- Semana4
- Semana5
- Semana6
- Semana7
- Semana8
- Semana9
- Semana10
- Semana11
- Semana12
- Semana13
- Semana14
- Semana15
- graph
- hashtable
- linkedList
- ANALISIS\_SEMANA15.md
- codejs
- Semana16
- queue
- tree
- ANALISIS\_SEMANA16.md
- index.css
- index.html
- indexjs
- notas.md

**Code Editor (Right):**

```
ANALISIS_SEMANA3nd .x
estructuras2025-2025-3> # ANALISIS_SEMANA3.md > ## Análisis - Semana 3: Bootstrap Avanzado, Listas Dblemente Enlazadas y Grafos > ## Estructura de Archivos
1 # Análisis - Semana 3: Bootstrap Avanzado, Listas Dblemente Enlazadas y Grafos "Análisis": Unknown word.
47 ## Descripción Detallada: "Descripción": Unknown word.
49 #### Archivos Principales: "Archivos": Unknown word.
51 ##### index.html": "Tutorial Completo de Bootstrap Grid
52
53 - Usar contenedores tradicionales
54
55 2. **container-fluid**:
56 - IWBW del ancho siempre
57 - Sin restricción de max-width
58 - Uso: layouts de ancho completo
59
60 3. **container-sm, `container-md, `container-lg, `container-xl, `container-xxl**:
61 - IWBW hasta el breakpoint especificado
62 Ejemplos:
63 - container-sm: 576px (small)
64 - container-md: 768px (medium)
65 - container-lg: 992px (large)
66 - container-xl: 1200px (extra large)
67 - container-xxl: 1400px (extra extra large)
68
69 **Sistemas Grid Row & Col:***
70 ***html
71 <div class="container">
72 <div class="row">
73 <div class="col" style="background-color: #f0f0f0; width: 100px;">
74 <div class="col" style="background-color: #e0e0e0; width: 100px;">
75 ...
76 </div>
77 </div>
78 </div>
79
80 ***Características:
81 - *Filas (row)*: Contenedores de 12 columnas
82 - *Columnas (col)*: Se distribuyen automáticamente
83 - **col-sm-6, col-md-4**: Especificar tamaño en breakpoints
84
85 ###### "mystyle.css" - Media Queries Responsivas
86
87 Implementa cambios de color según el tamaño de pantalla:
88
89 ***css
90 /* xs (por defecto, <576px) - verde */
91 @media (min-width: 576px) {
92 background-color: var(--bs-success);
93 }
94
95 /* sm (576px) - azul primario */
96 @media (min-width: 576px) {
97
98 /* md (768px) - gris secundario */
99 @media (min-width: 768px) {
100 background-color: var(--bs-secondary);
101 }
102
103 /* lg (992px) - rojo peligro */
104 @media (min-width: 992px) {
105 background-color: var(--bs-danger);
106 }
107
108 /* xl (1200px) - amarillo alerta */
109 @media (min-width: 1200px) {
110 background-color: var(--bs-warning);
111 }
112 }
113
114 /* xs (1400px) - azul info */
115 @media (min-width: 1400px) {
116 background-color: var(--bs-info);
117 }
118 }
```

## Análisis - Semana 4: Tipos de Datos Primitivos y Grafos Dirigidos

### Descripción General

Semana 4 combina **tipos de datos primitivos en JavaScript** (números, literales, constructores) con una **implementación avanzada de grafos dirigidos (Directed Graphs)**. Marca un punto de inflexión donde se profundiza en programación orientada a objetos aplicada a estructuras de datos complejas y se consolida la comprensión de números en JavaScript.

---

### Lenguaje

- **HTML5**
- **JavaScript (ES6+)** con enfoque en:
  - Tipos primitivos (number)
  - Clases modernas
  - Estructuras de datos (Map, Arrays)
  - Programación Orientada a Objetos (POO)

### Estructura de Archivos

Semana4/

```
|── index.html # Tutorial sobre tipos de datos numéricos
|── code.js # Funciones con números y operaciones
| básicas
└── graph/ # Implementación de grafo dirigido
 ├── index.html # Página HTML para grafo
 └── main.js # Clases Graph y Node para grafos dirigidos
```

---

### Objetivos

1. **Dominar tipos numéricos en JavaScript:** Entender números primitivos vs objetos Number
  2. **Usar notación legible de números:** Aplicar separadores visuales (5\_000\_000)
  3. **Operaciones aritméticas:** Crear funciones para sumas, multiplicaciones y tablas
  4. **Grafos dirigidos:** Implementar grafos donde las aristas tienen dirección
  5. **Clases Node y Graph:** Crear estructuras orientadas a objetos para grafos
  6. **Traversal de grafos:** Mostrar nodos y sus conexiones dirigidas
- 

## Descripción Detallada

### Archivos Principales

#### **index.html - Tutorial de Tipos de Datos Numéricos**

Página educativa que explica:

#### **Tipos de datos numéricos en JavaScript:**

1. **Números primitivos** (preferido):
  2. const number = 80;
  3. const decimal = 15.8;
    - o Creados literalmente
    - o Tipo primitivo eficiente
    - o Recomendado para uso general
4. **Objeto Number** (usando constructor):
  5. const num = new Number(42);
    - o Crea un objeto wrapper
    - o Menos eficiente

- Útil en casos específicos

### **Notación Legible:**

```
const legibleNumber = 5_000_000; // Equivalente a 5000000
```

- Separadores visuales (\_) para mejorar legibilidad
- No afecta el valor numérico
- Útil para números grandes

### **Tabla HTML de referencia:**

Constructor	Descripción
<code>new Number(number)</code>	Crea un objeto numérico a partir del número <code>number</code>
<code>number</code>	Simplemente, el número en cuestión. Notación preferida.

### **code.js - Funciones Numéricas Básicas**

#### **Variables globales:**

```
const number = 80; // Variable numérica constante
const decimal = 15.8; // Número decimal
```

```
const legibleNumber = 5_000_000; // Número con separador visual
```

### Funciones implementadas:

#### 1. **saldarNumber()**

- Itera desde 1,000,000 hasta 5,000,000 (de 1M en 1M)
- Imprime "Saludo número: X" para cada iteración
- Demuestra: loops, números grandes, legibilidad

#### 2. **entornoSuma(num)**

- Suma un parámetro con la variable global number (80)
- Retorna el resultado con console.log()
- Ejemplo: entornoSuma(20) → 100

#### 3. **tablaMultiplicar(num)**

- Genera tabla de multiplicación de 1 a 10
- Itera ascendente (i = 1 a 10)
- Formato: "num x i = resultado"

#### 4. **tablaMultiplicarM(num)**

- Similar a tablaMultiplicar() pero orden inverso
- Itera descendente (i = 10 a 1)
- Demuestra: loops invertidos

### Ejemplo de salida:

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

...

5 x 10 = 50

### Carpeta graph/

## **main.js - Implementación de Grafo Dirigido**

### **Clase Node:**

```
class Node {
 constructor(value) {
 this.value = value; // Valor del nodo
 this.edges = []; // Array de nodos conectados
 }

 addEdge(node) {
 this.edges.push(node); // Agregar conexión saliente
 }
}
```

### **Clase Graph:**

```
class Graph {
 constructor() {
 this.nodes = new Map(); // Map para almacenar nodos por valor
 }

 addNode(value) {
 const node = new Node(value);
 this.nodes.set(value, node); // Almacenar por clave (value)
 }

 addEdge(startValueNode, endValueNode) {
 const startNode = this.nodes.get(startValueNode);
 const endNode = this.nodes.get(endValueNode);
 startNode.addEdge(endNode);
 }
}
```

```

const endNode = this.nodes.get(endValueNode);

if (startNode && endNode) {
 startNode.addEdge(endNode); // DIRIGIDO: solo A→B, no B→A
}

}

show() {
 for(const node of this.nodes.values()) {
 const edges = node.edges.map(edge => edge.value).join(', ');
 console.log(` ${node.value} -> ${edges}`);
 }
}

```

**Ejemplo de uso:**

```

const graph = new Graph();

// Agregar nodos
graph.addNode('A');
graph.addNode('B');
graph.addNode('C');
graph.addNode('D');

// Agregar aristas dirigidas (unidireccionales)
graph.addEdge('A', 'B'); // A → B

```

```
graph.addEdge('A', 'D'); // A → D
graph.addEdge('B', 'C'); // B → C
graph.addEdge('B', 'D'); // B → D
graph.addEdge('C', 'D'); // C → D
graph.addEdge('D', 'A'); // D → A
```

```
graph.show();
```

**Salida esperada:**

A -> B, D

B -> C, D

C -> D

D -> A

**Visualización del grafo:**

A ----→ B

↑      ↓

|      C

|      ↓

└——→ D

---

 **Análisis Técnico**

**1. Tipos Numéricos en JavaScript**

**Primitivo vs Objeto**

Aspecto	Primitivo	Objeto
Creación	let n = 42;	let n = new Number(42);

Aspecto	Primitivo	Objeto
Tipo	number	object
Eficiencia	Alta	Baja
Memoria	Menos	Más
Métodos	Acceso automático	Directo
Comparación	$5 === 5 \rightarrow \text{true}$	$\text{new Number}(5) === \text{new Number}(5) \rightarrow \text{false}$

**Recomendación:** Usar primitivos siempre. Los wrappers Object son legado de JavaScript antiguo.

## Operaciones Numéricas

```
// Operaciones básicas

const a = 10;
const b = 3;

console.log(a + b); // 13 (suma)
console.log(a - b); // 7 (resta)
console.log(a * b); // 30 (multiplicación)
console.log(a / b); // 3.33... (división)
console.log(a % b); // 1 (módulo)
console.log(a ** b); // 1000 (exponenciación)
```

**Complejidad temporal:** O(1) para todas las operaciones aritméticas básicas.

## 2. Separador Visual de Números (Numeric Separator)

**Característica ES2021:**

```
// Legible

const million = 1_000_000;

const billion = 1_000_000_000;

const hex = 0xFF_FF_FF_FF;

const binary = 0b1111_0000;
```

// Es equivalente a:

```
const million = 1000000;

const billion = 1000000000;
```

### Ventajas:

- Mejora legibilidad en números grandes
- No afecta el valor
- Funciona en hex, binary, octal
- Ampliamente soportado en navegadores modernos

## 3. Grafos Dirigidos vs No Dirigidos

### Grafo No Dirigido (Semana 3)

```
addEdge(node1, node2) {

 this.adjacentList[node1].push(node2);

 this.adjacentList[node2].push(node1); // Bidireccional

}
```

Relación: A ↔ B (amistad en redes sociales)

### Grafo Dirigido (Semana 4)

```
addEdge(startValueNode, endValueNode) {

 const startNode = this.nodes.get(startValueNode);

 const endNode = this.nodes.get(endValueNode);
```

```
startNode.addEdge(endNode); // Unidireccional
}
```

Relación: A → B (seguir en Twitter)

### Aplicaciones de Grafos Dirigidos:

- Redes sociales (seguidor → seguido)
- Sistemas de recomendación
- Flujos de trabajo
- Análisis de dependencias
- Compiladores (DAG - Directed Acyclic Graph)

## 4. Uso de Map en lugar de Objetos

La clase Graph utiliza Map en lugar de objetos planos:

```
// Opción 1: Objeto plano
```

```
this.nodes = {};
```

```
// Opción 2: Map (utilizado)
```

```
this.nodes = new Map();
```

### Comparación:

Característica	Objeto {}	Map
Claves	Solo strings	Cualquier tipo
Iteración	for...in, Object.keys()	for...of, .values()
.size	Manual	Propiedad .size
Rendimiento	O(1)	O(1)
Métodos	Limitados	.has(), .delete(), .clear()

Característica	Objeto {}	Map
Prototipo	Hereda <b>proto</b>	Limpio

**Ventaja de Map aquí:**

- Métodos más intuitivos (.get(), .set())
- Mejor rendimiento en iteraciones
- Más seguro (sin colisiones con propiedades proto)

## 5. Complejidad de Operaciones en Grafo Dirigido

Operación	Complejidad	Descripción
addNode(value)	O(1)	Insertar en Map
addEdge(u, v)	O(1)	Agregar a array edges
show()	O(V + E)	Visitar cada nodo y arista
Buscar si existe arista	O(grado del nodo)	Buscar en array edges

Donde:

- V = número de vértices (nodos)
- E = número de aristas (conexiones)

## 6. Patrones de Diseño Aplicados

**Builder Pattern (implícito):**

```
const graph = new Graph();

graph.addNode('A');

graph.addNode('B');

graph.addEdge('A', 'B');
```

// Construcción paso a paso

**Observer Pattern (potencial):** La estructura permite fácilmente agregar métodos como:

- `onNodeAdded(callback)`
  - `onEdgeAdded(callback)`
- 



### Puntos de Aprendizaje Importantes

1. **Números primitivos son eficientes:** Evitar wrappers Object innecesarios
  2. **Legibilidad importa:** Los separadores visuales hacen código más mantenible
  3. **Grafos dirigidos son comunes:** La mayoría de aplicaciones reales usan direcciónalidad
  4. **Map > Objetos para estructuras:** Especialmente en estructura de datos complejas
  5. **POO facilita mantenimiento:** Clases Node y Graph encapsulan comportamiento
  6. **Diferencia conceptual crítica:** Dirigido vs No-dirigido cambia aplicaciones completamente
- 

### 🎓 Conclusión

Semana 4 es **fundamental en transición** porque:

- ✓ **Consolida JavaScript básico:** Domina números y sus operaciones
- ✓ **Introduce grafos dirigidos:** Extensión crucial de conceptos de Semana 3
- ✓ **Mejora POO:** Implementación más sofisticada de clases
- ✓ **Prepara para algoritmos:** BFS, DFS, topological sort dependen de grafos dirigidos

 **Aplicaciones reales:** Redes sociales, workflows, compiladores usan esto

 **Mejor performance:** Uso de Map vs objetos, números primitivos

Este material es ideal para estudiantes que dominan arrays y listas, y que están listos para estructuras no lineales más sofisticadas. La combinación de tipos primitivos fundamentales con grafos dirigidos avanzados crea un equilibrio entre lo básico y lo complejo, preparando para semanas posteriores sobre algoritmos de graph traversal (BFS, DFS) y problemas clásicos de grafos (camino más corto, componentes conectados, etc.).

Análisis - Semana 5: JavaScript, JSON y Listas Enlazadas

## Descripción General

La carpeta Semana5 reúne ejercicios y ejemplos prácticos sobre JavaScript básico aplicado al DOM, uso de archivos JSON para datos, y una introducción a **listas enlazadas** (linked lists). El material está orientado a consolidar conocimientos de programación imperativa y estructuras de datos lineales.

## Lenguaje

- **JavaScript (ES6+)**
  - **HTML5**
  - **CSS3 (básico)**
  - **JSON (para datos estáticos)**

## Estructura de Archivos

Semana 5/

```
├── index.html # Página principal con ejemplos HTML + scripts
├── code.js # Scripts de ejercicio (funciones y ejemplos)
├── inner.html # Página adicional / ejemplo
├── introduccion/ # Material introductorio
| ├── funtions/ # Ejemplos de funciones
| └── variable/ # Conceptos de variables
└── json/
 ├── index.html # Ejemplo de consumo de JSON en la página
 ├── main.js # Código que carga y procesa JSON
 └── people.json # Datos de ejemplo en formato JSON
└── linkedList/
 ├── index.html # Interfaz de demostración de linked list
 └── main.js # Implementación y métodos de la lista enlazada
```

└─ images/ # Recursos multimedia

---

## Objetivos

1. Entender integración básica entre HTML y JavaScript.
  2. Aprender a cargar y procesar datos en formato JSON desde el cliente.
  3. Implementar operaciones básicas en una Linked List (agregar, mostrar, eliminar).
  4. Practicar funciones, scopes y flujo de control en JavaScript.
  5. Presentar ejemplos interactivos que refuerzen la teoría.
- 

## Descripción Detallada

### **index.html y code.js**

- Contienen ejemplos de elementos HTML enlazados a funciones en code.js.
- Muestran uso de eventos (onclick) y manipulación simple del DOM.
- code.js incluye funciones para operaciones básicas (bucles, salidas en consola, demostraciones).

### **Carpeta json/**

- people.json provee datos de ejemplo (listas de objetos) para practicar lectura y renderizado dinámico.
- main.js ilustra cómo usar fetch() (o XMLHttpRequest) para cargar JSON y transformar los datos a HTML (tablas o listas).
- Útil para entender asincronía básica y promesas.

### **Carpeta linkedList/**

- main.js contiene la implementación de una lista enlazada simple (clase Node y clase LinkedList o funciones equivalentes).
- Métodos típicos: add, remove, show, clear y posiblemente find.

- index.html demuestra la interfaz para interactuar con la estructura (botones para agregar/eliminar y un área para mostrar la lista).
- 

## Breve Análisis Técnico

### 1. Integración HTML ↔ JS

- Patrón básico: incluir `<script src="code.js"></script>` y llamar funciones desde atributos onclick o añadir event listeners en JS.
- Buenas prácticas: preferir addEventListener y evitar lógica inline en HTML para separar estructura y comportamiento.

### 2. Consumo de JSON

- `fetch('people.json').then(res => res.json()).then(data => render(data))` es la forma moderna.
- Considerar manejo de errores (catch) y estados de carga.
- JSON es ideal para datos estáticos en ejemplos y pruebas locales.

### 3. Implementación de Linked List

- Nodos con value y next (singly linked list) o prev/next si es doble.
- Complejidades: add O(1) si se mantiene tail, remove O(n) en el peor caso.
- Ventajas pedagógicas: entender punteros y gestión dinámica de memoria conceptual en JS.

### 4. Calidad y mejoras sugeridas

- Modularizar código: extraer la lógica de estructuras a módulos (linkedList.js) y mantener main.js para la interacción.
  - Añadir pruebas unitarias simples (por ejemplo con Jest) para validar métodos de la lista.
  - Mejorar la UX: mostrar estados (vacío, elemento agregado) y mensajes de error.
-

## Puntos de Aprendizaje

- JSON + fetch() prepara para APIs reales.
  - Las linked lists permiten comprender por qué algunas operaciones en arrays son costosas.
  - Separación de responsabilidades (estructura vs UI) facilita mantenimiento del código.
- 

## Conclusión

Semana5 refuerza conceptos prácticos: manipulación del DOM con JavaScript, consumo de datos en JSON y la implementación manual de una Linked List. Es una semana orientada a traducción de teoría a práctica y sienta buenas bases para estudiar estructuras de datos más avanzadas y trabajar con APIs.

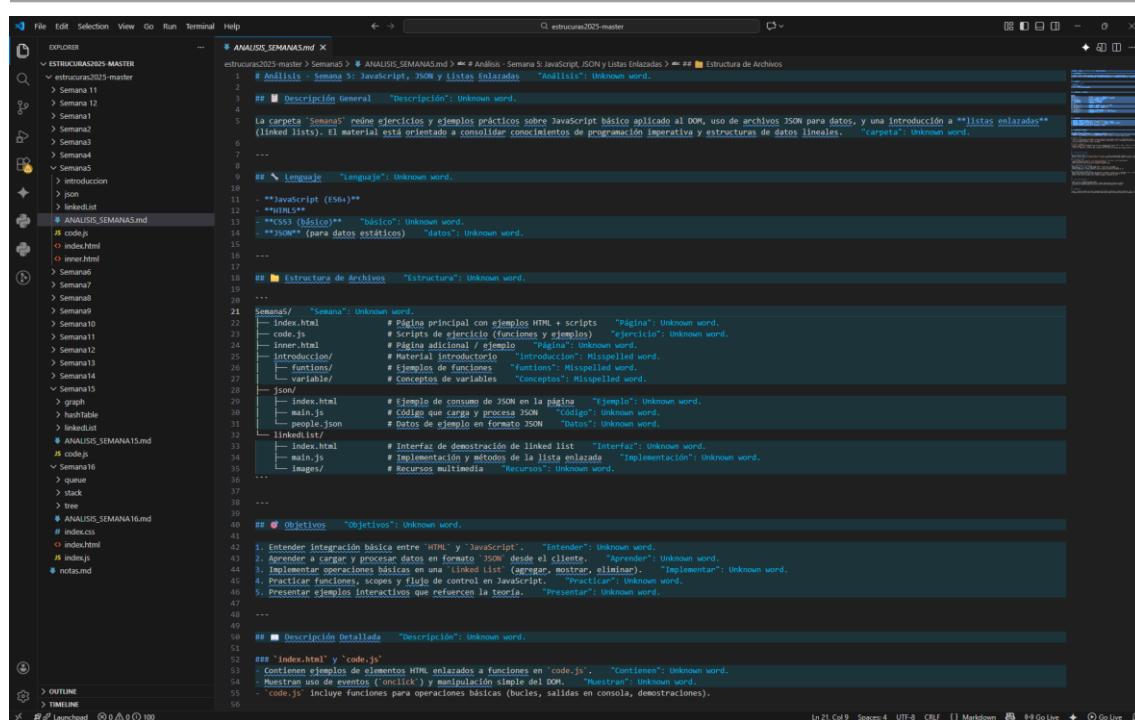
## Análisis - Semana 6: map() funcional, Programación Orientada a Objetos y Colas (Queue)

### Descripción General

La carpeta Semana6 integra conceptos de programación funcional aplicados a arrays (uso de map), fundamentos de **Programación Orientada a Objetos (POO)** en JavaScript y la implementación de **colas (queues)**. La semana busca consolidar la transición entre manipulación de colecciones y diseño de estructuras/abstracciones con clases.

### Lenguaje

- **JavaScript (ES6+)** — énfasis en funciones de orden superior, clases y módulos sencillos.
- **HTML5** — páginas de ejemplo que integran los scripts.
- **CSS3 (básico)** — estilos mínimos si aplica.



```
File Edit Selection View Go Run Terminal Help
EXPLORER ANALISIS_SEMANAS15.md
estrucuras2025-master
estrucuras2025-master
> Semana 11
> Semana 12
> Semana1
> Semana2
> Semana3
> Semana4
> Semana5
> Semana6
> Semana7
> Semana8
> Semana9
> Semana10
> Semana11
> Semana12
> Semana13
> Semana14
> Semana15
> graph
> hashTable
> linkedList
ANALISIS_SEMANAS15.md
JS code.js
index.html
inner.html
Semana6
Semana7
Semana8
Semana9
Semana10
Semana11
Semana12
Semana13
Semana14
Semana15
JS code.js
JS index.js
JS index.html
JS indexjs
JS notas.md
ANALISIS_SEMANAS16.md
JS code.js
JS index.html
JS indexjs
JS notas.md
OUTLINE
TIMELINE
File 21 Col 9 Spaces 4 - UTF-8 - CRLF { } Markdown ⌘ Go Live ⌘ Go Live ⌘
```

ANALISIS - Semana 5: Javascript, JSON y Listas Enlazadas "Analisis": Unknown word.

## ■ Descripción General "Descripción": Unknown word.

La carpeta "Semana" reúne ejercicios y ejemplos prácticos sobre JavaScript básico aplicado al DOM, uso de archivos JSON para datos, y una introducción a \*\*listas enlazadas\*\* (linked lists). El material está orientado a consolidar conocimientos de programación imperativa y estructuras de datos lineales. "carpeta": Unknown word.

---

## ■ Lenguaje "Lenguaje": Unknown word.

\*\*JavaScript (ES6+)\*\*  
\*\*HTML5\*\*  
\*\*CSS3 (básico)\*\*  
\*\*JSON\*\* (para datos estáticos) "datos": Unknown word.

---

## ■ Estructura de Archivos "Estructura": Unknown word.

---

Semana / "Semana": Unknown word.  
-- index.html # Página principal con ejemplos HTML + scripts "Página": Unknown word.  
-- code.js # Scripts de ejercicio (funciones y ejemplos) "ejercicio": Unknown word.  
-- inner.html # Página adicional / ejemplo "Página": Unknown word.  
-- introducción # Material introductorio "Introducción": Misspelled word.  
-- functions/ # Ejemplos de funciones "functions": Misspelled word.  
-- variables/ # Conceptos de variables "conceptos": Misspelled word.

-- json/  
-- index.html # Ejemplo de consumo de JSON en la página "Ejemplo": Unknown word.  
-- main.js # Código que carga y procesa JSON "Código": Unknown word.  
-- people.json # Datos de ejemplo en formato JSON "Datos": Unknown word.

-- linkedList/  
-- index.html # Interfaz de demostración de linked list "Interfaz": Unknown word.  
-- main.js # Implementación y métodos de la lista enlazada "Implementación": Unknown word.  
-- images/ # Recursos multimedia "Recursos": Unknown word.

---

---

## ■ Objetivos "Objetivos": Unknown word.

1. Entender integración básica entre "HTML" y "JavaScript". "Entender": Unknown word.
2. Aprender a cargar y procesar datos en formato JSON desde el cliente. "Aprender": Unknown word.
3. Implementar operaciones básicas en una "Linked List" (agregar, mostrar, eliminar). "Implementar": Unknown word.
4. Practicar funciones, scopes y flujo de control en Javascript. "Practicar": Unknown word.
5. Presentar ejemplos interactivos que refuerzen la teoría. "Presentar": Unknown word.

---

## ■ Descripción Detallada "Descripción": Unknown word.

---

## ■ index.html y "code.js"  
Contienen ejemplos de elementos HTML enlazados a funciones en "code.js". "Contienen": Unknown word.  
Muestran uso de eventos ("onClick") y manipulación simple del DOM. "Muestrar": Unknown word.  
"code.js" incluye funciones para operaciones básicas (bucles, salidas en consola, demostraciones).

```
File Edit Selection Go Run Terminal Help ← → C:\estructuras2025-master
ESTRUCTURAS2025-MASTER ... # ANALISIS_SEMANAS15.msd x
estructuras2025-master > Semana5 > # ANALISIS_SEMANAS15.msd > # Análisis - Semana 5: JavaScript, JSON y Listas Enlazadas > # Estructura de Archivos
1 # Análisis - Semana 5: JavaScript, JSON y Listas Enlazadas "Análisis": Unknown word.
2 50 ## Descripción Detallada "Descripción": Unknown word.
3 57 ## Carpeta "json"
4 - people.json provee datos de ejemplo (listas de objetos) para practicar lectura y renderizado dinámico.
5 - main.js ilustra como usar fetch() (o XMLHttpRequest) para cargar JSON y transformar los datos en HTML (tablas o listas).
6 - UI para entender asincronismo básicas y promesas.
7
8 ## Carpeta "linkedlist"
9 - main.js contiene la implementación de una lista enlazada simple (clase 'Node' y clase 'LinkedList' o funciones equivalentes).
10 - Métodos típicos: add(), remove(), show(), clear() y posiblemente find().
11 - index.html demuestra la interfaz para interactuar con la estructura (botones para agregar/eliminar y un área para mostrar la lista).
12
13 ## Breve Análisis Técnico
14
15 ## 1. Integración HTML + JS
16 - Patrón básico: Incluir <script src="code.js"></script> y llamar funciones desde atributos 'onclick' o añadir event listeners en JS.
17 - Buenas prácticas: preferir 'addEventListener' y evitar lógica inline en HTML para separar estructura y comportamiento.
18
19 ## 2. Consumo de JSON
20 - fetch('people.json').then(res => res.json()).then(data => render(data)) es la forma moderna.
21 - Considerar manejo de errores ('catch') y estados de carga.
22 - JSON es ideal para datos estáticos en ejemplos y pruebas locales.
23
24 ## 3. Implementación de linked List
25 - Nodos con 'value' y 'next' (singly linked list) o 'prev/next' si es doble.
26 - Complejidades: add (O(1)) si se mantiene 'tail', 'remove' (O(n)) en el peor caso.
27 - Ventajas pedagógicas: entender punteros y gestión dinámica de memoria conceptual en JS.
28
29 ## 4. Calidad y mejoras sugeridas
30 - Modularizar código: extraer la lógica de estructuras a módulos ("linkedlist.js") y mantener "main.js" para la interacción.
31 - Añadir pruebas unitarias simples (por ejemplo con 'jest') para validar métodos de la lista.
32 - Mejorar la UX: mostrar estados (vacío, elemento agregado) y mensajes de error.
33
34
35 ## Puntos de Aprendizaje
36
37 - JSON + 'fetch()' prepara para APIs reales.
38 - Las linked lists permiten comprender por qué algunas operaciones en arrays son costosas.
39 - Separación de responsabilidades (estructura vs UI) facilita mantenimiento del código.
40
41
42 ## Conclusiones
43
44 "Semana5" refuerza conceptos prácticos: manipulación del DOM con JavaScript, consumo de datos en 'JSON' y la implementación manual de una 'Linked List'. Es una semana orientada a traducción de teoría a práctica y sienta buenas bases para estudiar estructuras de datos más avanzadas y trabajar con APIs.
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
```

## Estructura de Archivos (estimada)

Semana 6/

```
|── index.html # Página principal de ejemplos
|
|── code.js # Ejemplos y utilidades JS
|
|── map/ # Ejemplos del método `map` y transformaciones
| | └── index.html / main.js
|
|── poo/ # Ejemplos de clases, constructores y herencia
| | └── clases/ referencia /
|
└── queue/ # Implementación de colas (enqueue, dequeue,
 peek)
 |
 └── index.html / main.js
```

Nota: la estructura refleja los temas principales presentes en la carpeta Semana6 (map, poo, queue).



1. Comprender y aplicar `Array.prototype.map()` para transformar colecciones de forma declarativa.
  2. Dominar conceptos básicos de POO en JavaScript: `class`, `constructor`, métodos, `this` y composición/herencia básica.
  3. Implementar una Queue (FIFO) con operaciones `enqueue`, `dequeue`, `peek` y `isEmpty`.
  4. Conectar programación funcional y POO: usar transformaciones antes/después de procesar estructuras lineales.
  5. Mejorar prácticas: separar lógica (clases/estructuras) de la interfaz (HTML), y usar `addEventListener` en lugar de `onclick` inline.
- 

## Descripción Detallada

### Carpeta map/

- Ejemplos de uso de `map()` para transformar arrays de datos (por ejemplo, arrays de objetos a arrays de strings o números).
- Comparación con `for` y `forEach`: `map` devuelve un nuevo array, es inmutable y encaja con programación funcional.
- Buenas prácticas: evitar efectos secundarios dentro del callback de `map`.

### Carpeta poo/

- Demostraciones de `class` y creación de instancias en ES6.
- Ejemplos típicos: clases `Person`, `Product`, o estructuras que encapsulan comportamiento y estado.
- Posible inclusión de herencia simple y métodos estáticos.
- Recomendaciones: mantener métodos puros cuando sea posible y evitar dependencias globales.

### Carpeta queue/

- Implementación clásica de Queue (puede ser con `Array` o con `LinkedList` internamente):

- enqueue(item) — agregar al final
  - dequeue() — eliminar del frente y retornar elemento
  - peek() — ver el frente sin eliminar
  - isEmpty() — true si la cola está vacía
  - Complejidad: enqueue y dequeue idealmente O(1) si se usa una estructura adecuada (por ejemplo, punteros head/tail en lista enlazada o un buffer circular).
  - Casos de uso: gestión de tareas, colas de impresión, breadth-first search (BFS) en grafos.
- 

## Breve Análisis Técnico

### 1. map() vs Mutación

- map() es apropiado para transformar datos sin mutar el original:
- const nums = [1, 2, 3];
- const squares = nums.map(x => x \* x); // [1,4,9]
- Evitar side-effects dentro del callback para preservar claridad y testabilidad.

### 2. POO en JavaScript (clases ES6)

- class es azúcar sintáctico sobre prototipos; entender this y binding es crucial.
- Ejemplo básico:
- class Person {
- constructor(name) { this.name = name; }
- greet() { return `Hola \${this.name}`; }
- }
- Considerar separar modelos (datos) de servicios (lógica que opera esos datos).

### 3. Implementación eficiente de Queue

- Usar Array.shift() es simple pero  $O(n)$  por reasignación de índices; para colas grandes preferir:
  - Linked list con head y tail ( $O(1)$  enqueue/dequeue).
  - Buffer circular con índice head y tail ( $O(1)$  y uso de espacio limitado).

### 4. Conexión entre paradigmas

- Flujo recomendado: recibir datos (JSON/array) → transformar con map/filter → encolar tareas con Queue → procesar con clases/servicios (POO).
- Esto muestra arquitectura simple: **ingest → transform → queue → process**.

### 5. Buenas prácticas y mejoras

- Modularizar: mover Queue y clases a módulos propios (queue.js, models/\*.js).
- Añadir tests unitarios mínimos (por ejemplo con un runner simple) para validar enqueue/dequeue.
- Documentar ejemplos en README.md dentro de la carpeta.

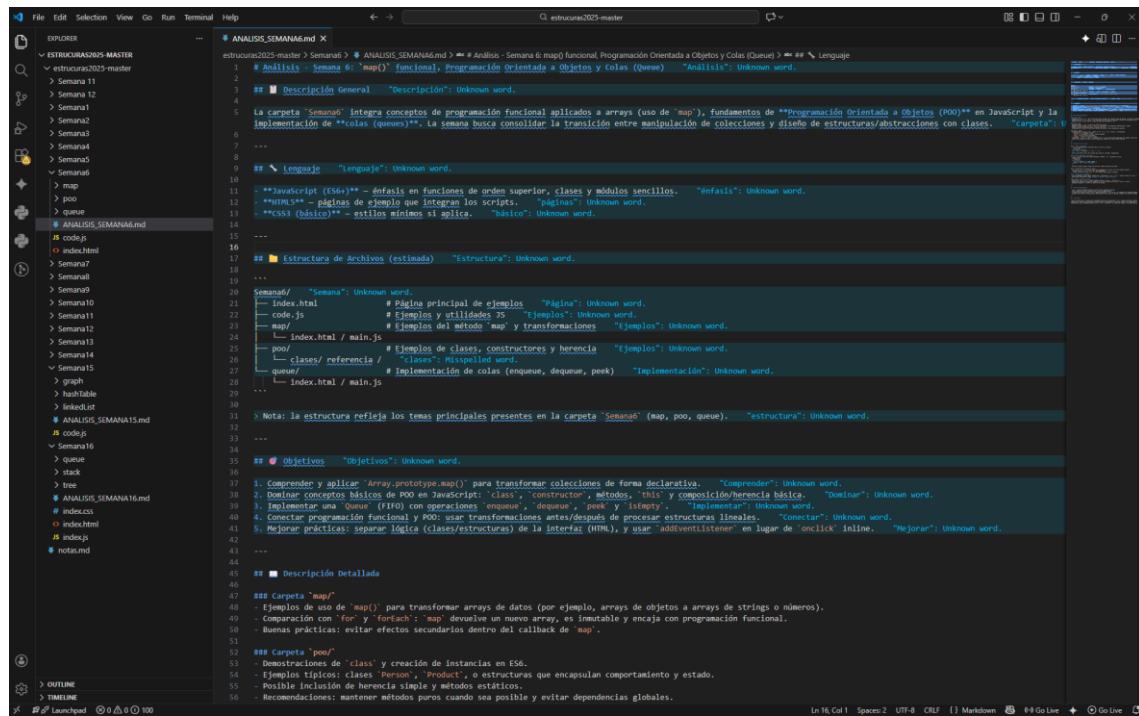


#### Puntos de Aprendizaje

- map() refuerza enfoque declarativo y evita errores comunes de mutación.
- POO en JavaScript facilita encapsulación y reuso de comportamiento.
- La elección de estructura para Queue impacta rendimiento; Array.shift() es aceptable para ejemplos pequeños, no para producción.
- Entender cómo combinar transformaciones funcionales con estructuras orientadas a objetos es clave para arquitecturas limpias.

## Conclusión

Semana6 busca cerrar la brecha entre tratamiento funcional de colecciones (map) y diseño orientado a objetos/estructuras (class, Queue). La semana prepara para algoritmos que requieren transformación previa de datos y procesamiento en cola (por ejemplo, BFS, pipelines de datos, procesamiento asíncrono). Aplicar las mejoras sugeridas (modularidad, tests, implementaciones O(1) para Queue) fortalecerá la calidad del código y la escalabilidad de los ejemplos.



```
File Edit Selection View Go Run Terminal Help
EXPLORER ANALISIS_SEMANA6.md
estruturas2025-master > ANALISIS_SEMANA6.md > # Análisis - Semana 6: "map" Funcional, Programación Orientada a Objetos y Colas (Queue) > # Descripción General
1 # Análisis - Semana 6: "map" Funcional, Programación Orientada a Objetos y Colas (Queue) "Análisis": Unknown word.
2
3 ## ■ Descripción General "Descripción": Unknown word.
4
5 La carpeta "Semana6" integra conceptos de programación funcional aplicados a arrays (uso de "map"), fundamentos de **Programación Orientada a Objetos (POO)** en Javascript y la implementación de **"celas (queues)"**. La semana busca consolidar la transición entre manipulación de colecciones y diseño de estructuras/abstracciones con clases. "carpeta": t
6
7 ...
8
9 ## ■ Lenguaje "Lenguaje": Unknown word.
10
11 : **JavaScript (ES6)** -- énfasis en funciones de orden superior, clases y módulos sencillos. "énfasis": Unknown word.
12 : **HTML5** -- páginas de ejemplo que integran los scripts. "páginas": Unknown word.
13 : **CSS3 (básico)** -- estilos mínimos si aplica. "básico": Unknown word.
14
15 ...
16
17 ## ■ Estructura de Archivos (estimada) "Estructura": Unknown word.
18
19 ...
20 Semana6/ "Semana": Unknown word.
21 index.html # Página principal de ejemplos "Página": Unknown word.
22 code.js # Ejemplos y utilidades JS "Ejemplos": Unknown word.
23 ...
24 code/ # Ejemplos del método "map" y transformaciones "Transformaciones": Unknown word.
25 ...
26 main.js # Ejemplos de clases, constructores y herencia "Ejemplos": Unknown word.
27 ...
28 queue/ # Ejemplos de colas (enqueue, dequeue, peek) "Implementación": Unknown word.
29 ...
30 ...
31 ...
32
33 Nota: la estructura refleja los temas principales presentes en la carpeta "Semana6" (map, poo, queue). "estructura": Unknown word.
34
35 ...
36
37 ## ■ Objetivos "Objetivos": Unknown word.
38
39 1. Comprender y aplicar "Array.prototype.map()" para transformar colecciones de forma declarativa. "Comprender": Unknown word.
40 2. Implementar operaciones básicas de POO en Javascript: "clases", "constructores", métodos, "herencia" y "función herencia básica". "Dominio": Unknown word.
41 3. Implementar una "Queue" (FIFO) con operaciones "enqueue", "dequeue", "peek" y "isEmpty". "Implementar": Unknown word.
42 4. Conectar programación funcional y POO: usar transformaciones antes/después de procesar estructuras lineales. "Conectar": Unknown word.
43 5. Mejorar prácticas: separar lógica (clases/estructuras) de la interfaz (HTML), y usar "addEventListener" en lugar de "onclick" inline. "Mejorar": Unknown word.
44
45 ...
46
47 ## ■ Descripción Detallada
48
49 ## ■ Carpeta "map"
50
51 Ejemplos de uso de "map()" para transformar arrays de datos (por ejemplo, arrays de objetos a arrays de strings o números).
52 - Comparación con "for" y "foreach": map devuelve un nuevo array, es immutable y encaja con programación funcional.
53 - Buenas prácticas: evitar efectos secundarios dentro del callback de map.
54
55 ## ■ Carpeta "poo"
56
57 Demos/ Ejemplos de "clases" y creación de instancias en ES6.
58 - Ejemplos típicos: clases "Person", "Product", o estructuras que encapsulan comportamiento y estado.
59 - Posible inclusión de herencia simple y métodos estáticos.
60 - Recomendaciones: mantener métodos puros cuando sea posible y evitar dependencias globales.
```

The screenshot shows a code editor interface with the following details:

- File Path:** estrucuras2025-master/ANALISIS\_SEMANA6.md
- Content:** A Markdown file containing code analysis comments. The comments are preceded by numbers (e.g., 1, 45, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105).
- Comments:** The comments provide insights into various programming concepts, such as:

  - Map:** Describes the 'map' function, mentioning its implementation in ES6, typical examples, and best practices.
  - Queue:** Discusses the 'queue' structure, its implementation in ES6, and its use in breadth-first search (BFS) algorithms.
  - Object-Oriented Programming:** Mentions the 'class' keyword and its use in prototypal inheritance.
  - Prototypal Inheritance:** Explains the 'Object.create()' method and its use in creating objects from prototypes.
  - Performance:** Notes on the performance of enqueue and dequeue operations, mentioning O(1) complexity.
  - Implementation:** Provides examples of implementing queues in JavaScript, including linked lists and buffers.
  - Best Practices:** Offers recommendations for maintaining code quality, such as avoiding side-effects in callbacks and using descriptive variable names.

- Editor Features:** The interface includes standard file navigation (File, Edit, Selection, View, Go, Run, Terminal, Help), a status bar at the bottom, and a right-hand sidebar showing a preview of the code.

## Análisis - Semana 7: Tablas Hash, Sets y Pilas (Stacks)

### Descripción General

La carpeta Semana7 agrupa ejercicios y ejemplos prácticos sobre **tablas hash (hash tables)**, **conjuntos (sets)** y **pilas (stacks)**. El material combina teoría (complejidad, colisiones) con implementaciones en JavaScript y ejemplos de uso para entender por qué estas estructuras son relevantes en problemas reales.

---

### Lenguaje

- **JavaScript (ES6+)**
- **HTML5**
- **CSS3 (básico)**

Las implementaciones usan JavaScript puro, con clases o funciones para ilustrar principios de diseño y rendimiento.

---

### Estructura de Archivos

Semana7/

```
|── index.html # Página de demostración y pruebas
|── code.js # Ejemplos y utilidades generales
|── hashTables/ # Implementación y ejemplos de tablas hash
| |── hashTables.js (o similar)
|── set/ # Ejemplos y uso de conjuntos (Set)
| |── index.html / main.js
└── stack/ # Implementación de pilas (Stack)
 |── stack.js / index.html
```

Nota: los nombres de archivos concretos pueden variar; la estructura refleja las subcarpetas observadas en el repositorio.

---

## Objetivos

1. Entender el concepto de **función hash** y cómo se usa para indexar datos.
  2. Implementar una **Hash Table** básica con manejo de colisiones (encadenamiento o open addressing).
  3. Aprender el uso y ventajas de Set en JavaScript (unión, intersección, diferencia).
  4. Implementar una **Stack** (LIFO) con operaciones push, pop, peek y isEmpty.
  5. Comparar complejidades y elegir la estructura adecuada según el problema.
- 

## Descripción Detallada

### Carpeta hashTables/

- Contiene una implementación educativa de una tabla hash.
- Elementos clave a revisar:
  - **Función hash:** conversión de claves a índices numéricos.
  - **Manejo de colisiones:** chaining (listas enlazadas por bucket) o open addressing.
  - **Operaciones:** set(key, value), get(key), has(key), delete(key).
- Recomendaciones: validar la calidad del hash y probar con colisiones intencionales.

### Carpeta set/

- Muestra el uso de Set nativo de ES6 y operaciones típicas:
  - new Set(iterable) — creación

- add, delete, has — manipulación básica
- Conversión a Array para aplicar map/filter
- Ejemplos típicos: eliminar duplicados, cálculos de unión/intersección.

## Carpeta stack/

- Implementación de pila para demostrar LIFO.
- Métodos esperados:
  - push(item) — agregar elemento
  - pop() — eliminar y retornar último elemento
  - peek() — ver el tope sin eliminar
  - isEmpty() — estado de la pila
- Casos de uso: evaluar expresiones (postfix), backtracking, navegación (historial)



## Breve Análisis Técnico

### 1. Tablas Hash

- Operaciones promedio: get, set, delete → O(1) promedio.
- Peor caso: O(n) cuando muchas claves colisionan o mal diseño del hash.
- Estrategias de colisión:
  - **Encadenamiento:** cada bucket mantiene una lista de entradas; sencillo y eficiente para cargas moderadas.
  - **Open addressing:** sondaje lineal/ cuadrático/ doble hashing; requiere manejo cuidadoso del factor de carga.
- Factor de carga (load factor) debe monitorearse y, si es necesario, redimensionar la tabla (rehashing).

### 2. Set (ES6)

- Implementación nativa optimizada; uso recomendado para operaciones de pertenencia y eliminación de duplicados.
- Operaciones: add, delete, has → O(1) promedio.
- Ejemplo: const unique = [...new Set(array)]; elimina duplicados rápidamente.

### 3. Stack (Pila)

- Implementación simple con Array (push/pop) ofrece O(1) para operaciones de tope.
- Alternativa con lista enlazada cuando se desea control explícito de nodos y O(1) garantiza sin reasignaciones.
- Uso en algoritmos: evaluación de expresiones, recorridos recursivos simulados, undo/redo.

### 4. Casos de uso y recomendaciones

- Usar **Hash Table** para índices rápidos por clave (caches, tablas de símbolos, conteo de frecuencia).
- Usar **Set** para limpiar datos y operaciones de teoría de conjuntos.
- Usar **Stack** donde el orden LIFO es necesario (paréntesis balanceados, DFS iterativo).

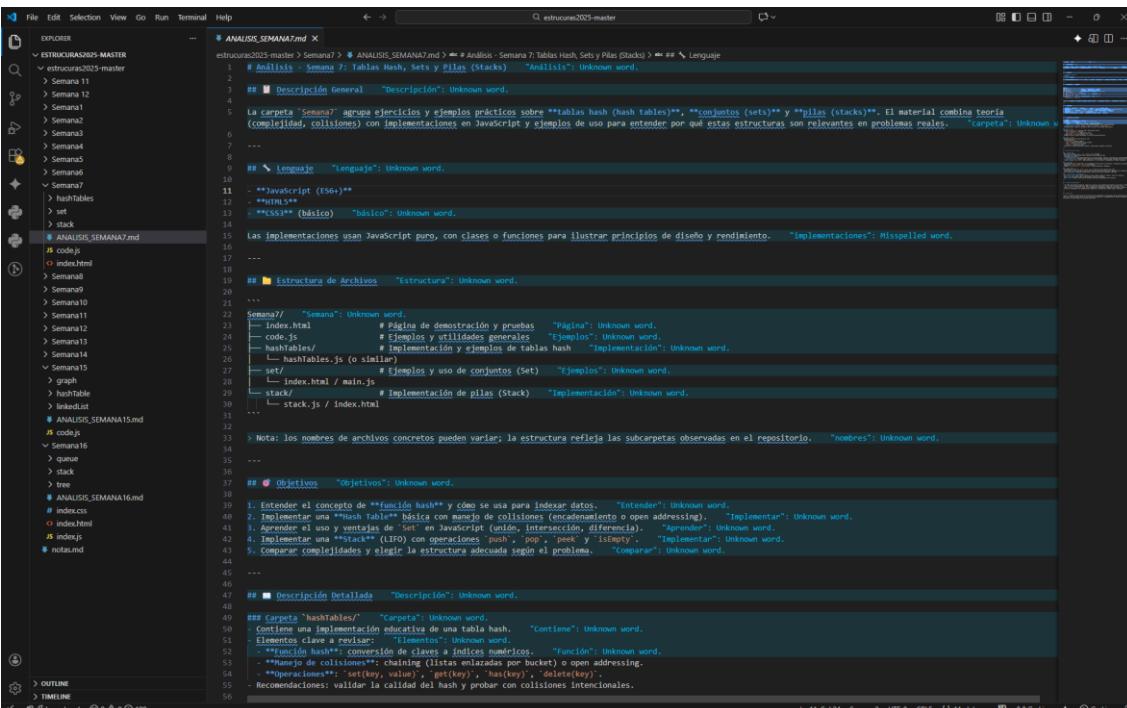


#### Puntos de Aprendizaje

- Las tablas hash ofrecen acceso muy rápido en la práctica, pero requieren buen manejo de colisiones y rehashing.
  - Set es una herramienta poderosa de ES6 para simplificar operaciones comunes con colecciones únicas.
  - Elegir Array vs LinkedList para Stack depende de la necesidad de rendimiento en escenarios extremos.
  - Es valioso complementar las implementaciones educativas con pruebas que simulen cargas y colisiones.
-

## 🎓 Conclusión

Semana7 consolida estructuras de datos esenciales para rendimiento en aplicaciones reales: hashing para acceso por clave, conjuntos para unicidad y pilas para control de flujo LIFO. Aplicar pruebas, medir el factor de carga y modularizar las implementaciones (módulos separados para hashTables, set, stack) mejorará la calidad del material y facilitará su reutilización en ejercicios posteriores.



```
File Edit Selection View Go Run Terminal Help
EXPLORER ANALISIS_SEMANA7.md
estrucuras2025-master > Semana7 > ANALISIS_SEMANA7.md > # Análisis - Semana 7: Tablas Hash, Sets y Pilas (Stacks) > # Lenguaje
1 # Análisis - Semana 7: Tablas Hash, Sets y Pilas (Stacks) "Analisis": Unknown word.
2
3 ## ■ Descripción General "Descripción": Unknown word.
4
5 La carpeta "Semana7" agrupa ejercicios y ejemplos prácticos sobre **tablas hash (hash tables)**, **conjuntos (sets)** y **pilas (stacks)**. El material combina teoría (complejidad, colisiones) con implementaciones en JavaScript y ejemplos de uso para entender por qué estas estructuras son relevantes en problemas reales. "carpeta": Unknown word.
6
7 ...
8
9 ## ■ Lenguaje "Lenguaje": Unknown word.
10
11 **JavaScript (ES6)**
12 **HTML5**
13 **CSS3** (básico): Unknown word.
14
15 Las implementaciones usan JavaScript puro, con clases o funciones para ilustrar principios de diseño y rendimiento. "Implementaciones": Misspelled word.
16
17 ...
18
19 ## ■ Estructura de Archivos "Estructura": Unknown word.
20
21 ...
22 Semana7/ "Semana": Unknown word.
23 Index.html # Página de demostración y pruebas "Página": Unknown word.
24 code.js # Ejemplos y utilidades generales "Ejemplos": Unknown word.
25 hashTables/ # Implementación y ejemplos de tablas hash "Implementación": Unknown word.
26 hashTables.js (o similar)
27 ...
28 ...
29 ...
30 ...
31 ...
32 ...
33 ...
34 ...
35 ...
36 ...
37 ## ■ Objetivos "Objetivos": Unknown word.
38
39 1. Entender el concepto de **función hash** y cómo se usa para indexar datos. "Entender": Unknown word.
40 2. Implementar una **Hash Table** básica con manejo de colisiones (encadenamiento o open addressing). "Implementar": Unknown word.
41 3. Aprender el uso y ventajas de Set en Javascript (única, intersección, diferencia).
42 4. Implementar una **stack** (LIFO) con operaciones push, pop, peek y isEmpty. "Aprender": Unknown word.
43 5. Comparar complejidades y elegir la estructura adecuada según el problema. "Comparar": Unknown word.
44
45 ...
46
47 ## ■ Descripción Detallada "Descripción": Unknown word.
48
49 ## Carpeta "hashTables/" "Carpeta": Unknown word.
50 Contiene una implementación educativa de una tabla hash. "Contiene": Unknown word.
51 Elementos clave a revisar: "Elementos": Unknown word.
52 - **Encadenamiento**: lista de enlaces entre los nodos que tienen la misma clave.
53 - **Open Addressing**: colisión. Claves almacenadas en un array.
54 - **Operaciones**: set(key, value), get(key), has(key), delete(key).
55 - Recomendaciones: validar la calidad del hash y probar con colisiones intencionales.
56
57 ...
58 ...
59 ...
60 ...
61 ...
62 ...
63 ...
64 ...
65 ...
66 ...
67 ...
68 ...
69 ...
70 ...
71 ...
72 ...
73 ...
74 ...
75 ...
76 ...
77 ...
78 ...
79 ...
80 ...
81 ...
82 ...
83 ...
84 ...
85 ...
86 ...
87 ...
88 ...
89 ...
90 ...
91 ...
92 ...
93 ...
94 ...
95 ...
96 ...
97 ...
98 ...
99 ...
100 ...
101 ...
102 ...
103 ...
104 ...
105 ...
106 ...
107 ...
108 ...
109 ...
110 ...
111 ...
112 ...
113 ...
114 ...
115 ...
116 ...
117 ...
118 ...
119 ...
119 ...
120 ...
121 ...
122 ...
123 ...
124 ...
125 ...
126 ...
127 ...
128 ...
129 ...
129 ...
130 ...
131 ...
132 ...
133 ...
134 ...
135 ...
136 ...
137 ...
138 ...
139 ...
139 ...
140 ...
141 ...
142 ...
143 ...
144 ...
145 ...
146 ...
147 ...
148 ...
149 ...
149 ...
150 ...
151 ...
152 ...
153 ...
154 ...
155 ...
156 ...
157 ...
158 ...
159 ...
159 ...
160 ...
161 ...
162 ...
163 ...
164 ...
165 ...
166 ...
167 ...
168 ...
169 ...
169 ...
170 ...
171 ...
172 ...
173 ...
174 ...
175 ...
176 ...
177 ...
178 ...
179 ...
179 ...
180 ...
181 ...
182 ...
183 ...
184 ...
185 ...
186 ...
187 ...
188 ...
189 ...
189 ...
190 ...
191 ...
192 ...
193 ...
194 ...
195 ...
196 ...
197 ...
198 ...
199 ...
199 ...
200 ...
201 ...
202 ...
203 ...
204 ...
205 ...
206 ...
207 ...
208 ...
209 ...
209 ...
210 ...
211 ...
212 ...
213 ...
214 ...
215 ...
216 ...
217 ...
218 ...
219 ...
219 ...
220 ...
221 ...
222 ...
223 ...
224 ...
225 ...
226 ...
227 ...
228 ...
229 ...
229 ...
230 ...
231 ...
232 ...
233 ...
234 ...
235 ...
236 ...
237 ...
238 ...
239 ...
239 ...
240 ...
241 ...
242 ...
243 ...
244 ...
245 ...
246 ...
247 ...
248 ...
249 ...
249 ...
250 ...
251 ...
252 ...
253 ...
254 ...
255 ...
256 ...
257 ...
258 ...
259 ...
259 ...
260 ...
261 ...
262 ...
263 ...
264 ...
265 ...
266 ...
267 ...
268 ...
269 ...
269 ...
270 ...
271 ...
272 ...
273 ...
274 ...
275 ...
276 ...
277 ...
278 ...
279 ...
279 ...
280 ...
281 ...
282 ...
283 ...
284 ...
285 ...
286 ...
287 ...
287 ...
288 ...
289 ...
289 ...
290 ...
291 ...
292 ...
293 ...
294 ...
295 ...
296 ...
297 ...
297 ...
298 ...
299 ...
299 ...
300 ...
301 ...
302 ...
303 ...
304 ...
305 ...
306 ...
307 ...
308 ...
309 ...
309 ...
310 ...
311 ...
312 ...
313 ...
314 ...
315 ...
316 ...
317 ...
318 ...
319 ...
319 ...
320 ...
321 ...
322 ...
323 ...
324 ...
325 ...
326 ...
327 ...
328 ...
329 ...
329 ...
330 ...
331 ...
332 ...
333 ...
334 ...
335 ...
336 ...
337 ...
338 ...
339 ...
339 ...
340 ...
341 ...
342 ...
343 ...
344 ...
345 ...
346 ...
347 ...
348 ...
349 ...
349 ...
350 ...
351 ...
352 ...
353 ...
354 ...
355 ...
356 ...
357 ...
358 ...
359 ...
359 ...
360 ...
361 ...
362 ...
363 ...
364 ...
365 ...
366 ...
367 ...
368 ...
369 ...
369 ...
370 ...
371 ...
372 ...
373 ...
374 ...
375 ...
376 ...
377 ...
378 ...
379 ...
379 ...
380 ...
381 ...
382 ...
383 ...
384 ...
385 ...
386 ...
387 ...
388 ...
389 ...
389 ...
390 ...
391 ...
392 ...
393 ...
394 ...
395 ...
396 ...
397 ...
398 ...
399 ...
399 ...
400 ...
401 ...
402 ...
403 ...
404 ...
405 ...
406 ...
407 ...
408 ...
409 ...
409 ...
410 ...
411 ...
412 ...
413 ...
414 ...
415 ...
416 ...
417 ...
418 ...
419 ...
419 ...
420 ...
421 ...
422 ...
423 ...
424 ...
425 ...
426 ...
427 ...
428 ...
429 ...
429 ...
430 ...
431 ...
432 ...
433 ...
434 ...
435 ...
436 ...
437 ...
438 ...
439 ...
439 ...
440 ...
441 ...
442 ...
443 ...
444 ...
445 ...
446 ...
447 ...
448 ...
449 ...
449 ...
450 ...
451 ...
452 ...
453 ...
454 ...
455 ...
456 ...
457 ...
458 ...
459 ...
459 ...
460 ...
461 ...
462 ...
463 ...
464 ...
465 ...
466 ...
467 ...
468 ...
469 ...
469 ...
470 ...
471 ...
472 ...
473 ...
474 ...
475 ...
476 ...
477 ...
478 ...
479 ...
479 ...
480 ...
481 ...
482 ...
483 ...
484 ...
485 ...
486 ...
487 ...
488 ...
489 ...
489 ...
490 ...
491 ...
492 ...
493 ...
494 ...
495 ...
496 ...
497 ...
498 ...
499 ...
499 ...
500 ...
501 ...
502 ...
503 ...
504 ...
505 ...
506 ...
507 ...
508 ...
509 ...
509 ...
510 ...
511 ...
512 ...
513 ...
514 ...
515 ...
516 ...
517 ...
518 ...
519 ...
519 ...
520 ...
521 ...
522 ...
523 ...
524 ...
525 ...
526 ...
527 ...
528 ...
529 ...
529 ...
530 ...
531 ...
532 ...
533 ...
534 ...
535 ...
536 ...
537 ...
538 ...
539 ...
539 ...
540 ...
541 ...
542 ...
543 ...
544 ...
545 ...
546 ...
547 ...
548 ...
549 ...
549 ...
550 ...
551 ...
552 ...
553 ...
554 ...
555 ...
556 ...
557 ...
558 ...
559 ...
559 ...
560 ...
561 ...
562 ...
563 ...
564 ...
565 ...
566 ...
567 ...
568 ...
569 ...
569 ...
570 ...
571 ...
572 ...
573 ...
574 ...
575 ...
576 ...
577 ...
578 ...
579 ...
579 ...
580 ...
581 ...
582 ...
583 ...
584 ...
585 ...
586 ...
587 ...
588 ...
589 ...
589 ...
590 ...
591 ...
592 ...
593 ...
594 ...
595 ...
596 ...
597 ...
598 ...
599 ...
599 ...
600 ...
601 ...
602 ...
603 ...
604 ...
605 ...
606 ...
607 ...
608 ...
609 ...
609 ...
610 ...
611 ...
612 ...
613 ...
614 ...
615 ...
616 ...
617 ...
618 ...
619 ...
619 ...
620 ...
621 ...
622 ...
623 ...
624 ...
625 ...
626 ...
627 ...
628 ...
629 ...
629 ...
630 ...
631 ...
632 ...
633 ...
634 ...
635 ...
636 ...
637 ...
638 ...
639 ...
639 ...
640 ...
641 ...
642 ...
643 ...
644 ...
645 ...
646 ...
647 ...
648 ...
649 ...
649 ...
650 ...
651 ...
652 ...
653 ...
654 ...
655 ...
656 ...
657 ...
658 ...
659 ...
659 ...
660 ...
661 ...
662 ...
663 ...
664 ...
665 ...
666 ...
667 ...
668 ...
669 ...
669 ...
670 ...
671 ...
672 ...
673 ...
674 ...
675 ...
676 ...
677 ...
678 ...
679 ...
679 ...
680 ...
681 ...
682 ...
683 ...
684 ...
685 ...
686 ...
687 ...
688 ...
689 ...
689 ...
690 ...
691 ...
692 ...
693 ...
694 ...
695 ...
696 ...
697 ...
698 ...
699 ...
699 ...
700 ...
701 ...
702 ...
703 ...
704 ...
705 ...
706 ...
707 ...
708 ...
709 ...
709 ...
710 ...
711 ...
712 ...
713 ...
714 ...
715 ...
716 ...
717 ...
718 ...
719 ...
719 ...
720 ...
721 ...
722 ...
723 ...
724 ...
725 ...
726 ...
727 ...
728 ...
729 ...
729 ...
730 ...
731 ...
732 ...
733 ...
734 ...
735 ...
736 ...
737 ...
738 ...
739 ...
739 ...
740 ...
741 ...
742 ...
743 ...
744 ...
745 ...
746 ...
747 ...
748 ...
749 ...
749 ...
750 ...
751 ...
752 ...
753 ...
754 ...
755 ...
756 ...
757 ...
758 ...
759 ...
759 ...
760 ...
761 ...
762 ...
763 ...
764 ...
765 ...
766 ...
767 ...
768 ...
769 ...
769 ...
770 ...
771 ...
772 ...
773 ...
774 ...
775 ...
776 ...
777 ...
778 ...
779 ...
779 ...
780 ...
781 ...
782 ...
783 ...
784 ...
785 ...
786 ...
787 ...
788 ...
789 ...
789 ...
790 ...
791 ...
792 ...
793 ...
794 ...
795 ...
796 ...
797 ...
798 ...
799 ...
799 ...
800 ...
801 ...
802 ...
803 ...
804 ...
805 ...
806 ...
807 ...
808 ...
809 ...
809 ...
810 ...
811 ...
812 ...
813 ...
814 ...
815 ...
816 ...
817 ...
818 ...
819 ...
819 ...
820 ...
821 ...
822 ...
823 ...
824 ...
825 ...
826 ...
827 ...
828 ...
829 ...
829 ...
830 ...
831 ...
832 ...
833 ...
834 ...
835 ...
836 ...
837 ...
838 ...
839 ...
839 ...
840 ...
841 ...
842 ...
843 ...
844 ...
845 ...
846 ...
847 ...
848 ...
849 ...
849 ...
850 ...
851 ...
852 ...
853 ...
854 ...
855 ...
856 ...
857 ...
858 ...
859 ...
859 ...
860 ...
861 ...
862 ...
863 ...
864 ...
865 ...
866 ...
867 ...
868 ...
869 ...
869 ...
870 ...
871 ...
872 ...
873 ...
874 ...
875 ...
876 ...
877 ...
878 ...
879 ...
879 ...
880 ...
881 ...
882 ...
883 ...
884 ...
885 ...
886 ...
887 ...
888 ...
889 ...
889 ...
890 ...
891 ...
892 ...
893 ...
894 ...
895 ...
896 ...
897 ...
898 ...
899 ...
899 ...
900 ...
901 ...
902 ...
903 ...
904 ...
905 ...
906 ...
907 ...
908 ...
909 ...
909 ...
910 ...
911 ...
912 ...
913 ...
914 ...
915 ...
916 ...
917 ...
918 ...
919 ...
919 ...
920 ...
921 ...
922 ...
923 ...
924 ...
925 ...
926 ...
927 ...
928 ...
929 ...
929 ...
930 ...
931 ...
932 ...
933 ...
934 ...
935 ...
936 ...
937 ...
938 ...
939 ...
939 ...
940 ...
941 ...
942 ...
943 ...
944 ...
945 ...
946 ...
947 ...
948 ...
949 ...
949 ...
950 ...
951 ...
952 ...
953 ...
954 ...
955 ...
956 ...
957 ...
958 ...
959 ...
959 ...
960 ...
961 ...
962 ...
963 ...
964 ...
965 ...
966 ...
967 ...
968 ...
969 ...
969 ...
970 ...
971 ...
972 ...
973 ...
974 ...
975 ...
976 ...
977 ...
978 ...
978 ...
979 ...
980 ...
981 ...
982 ...
983 ...
984 ...
985 ...
986 ...
987 ...
988 ...
989 ...
989 ...
990 ...
991 ...
992 ...
993 ...
994 ...
995 ...
996 ...
997 ...
998 ...
999 ...
999 ...
1000 ...
1001 ...
1002 ...
1003 ...
1004 ...
1005 ...
1006 ...
1007 ...
1008 ...
1009 ...
1009 ...
1010 ...
1011 ...
1012 ...
1013 ...
1014 ...
1015 ...
1016 ...
1017 ...
1018 ...
1019 ...
1019 ...
1020 ...
1021 ...
1022 ...
1023 ...
1024 ...
1025 ...
1026 ...
1027 ...
1028 ...
1029 ...
1029 ...
1030 ...
1031 ...
1032 ...
1033 ...
1034 ...
1035 ...
1036 ...
1037 ...
1038 ...
1039 ...
1039 ...
1040 ...
1041 ...
1042 ...
1043 ...
1044 ...
1045 ...
1046 ...
1047 ...
1048 ...
1049 ...
1049 ...
1050 ...
1051 ...
1052 ...
1053 ...
1054 ...
1055 ...
1056 ...
1057 ...
1058 ...
1059 ...
1059 ...
1060 ...
1061 ...
1062 ...
1063 ...
1064 ...
1065 ...
1066 ...
1067 ...
1068 ...
1069 ...
1069 ...
1070 ...
1071 ...
1072 ...
1073 ...
1074 ...
1075 ...
1076 ...
1077 ...
1078 ...
1078 ...
1079 ...
1080 ...
1081 ...
1082 ...
1083 ...
1084 ...
1085 ...
1086 ...
1087 ...
1088 ...
1089 ...
1089 ...
1090 ...
1091 ...
1092 ...
1093 ...
1094 ...
1095 ...
1095 ...
1096 ...
1097 ...
1098 ...
1099 ...
1099 ...
1100 ...
1101 ...
1102 ...
1103 ...
1104 ...
1105 ...
1106 ...
1107 ...
1108 ...
1109 ...
1109 ...
1110 ...
1111 ...
1112 ...
1113 ...
1114 ...
1115 ...
1116 ...
1117 ...
1118 ...
1119 ...
1119 ...
1120 ...
1121 ...
1122 ...
1123 ...
1124 ...
1125 ...
1126 ...
1127 ...
1128 ...
1129 ...
1129 ...
1130 ...
1131 ...
1132 ...
1133 ...
1134 ...
1135 ...
1136 ...
1137 ...
1138 ...
1139 ...
1139 ...
1140 ...
1141 ...
1142 ...
1143 ...
1144 ...
1145 ...
1146 ...
1147 ...
1148 ...
1149 ...
1149 ...
1150 ...
1151 ...
1152 ...
1153 ...
1154 ...
1155 ...
1156 ...
1157 ...
1158 ...
1159 ...
1159 ...
1160 ...
1161 ...
1162 ...
1163 ...
1164 ...
1165 ...
1166 ...
1167 ...
1168 ...
1169 ...
1169 ...
1170 ...
1171 ...
1172 ...
1173 ...
1174 ...
1175 ...
1176 ...
1177 ...
1178 ...
1178 ...
1179 ...
1180 ...
1181 ...
1182 ...
1183 ...
1184 ...
1185 ...
1186 ...
1187 ...
1188 ...
1189 ...
1189 ...
1190 ...
1191 ...
1192 ...
1193 ...
1194 ...
1195 ...
1195 ...
1196 ...
1197 ...
1198 ...
1199 ...
1199 ...
1200 ...
1201 ...
1202 ...
1203 ...
1204 ...
1205 ...
1206 ...
1207 ...
1208 ...
1209 ...
1209 ...
1210 ...
1211 ...
1212 ...
1213 ...
1214 ...
1215 ...
1216 ...
1217 ...
1218 ...
1219 ...
1219 ...
1220 ...
1221 ...
1222 ...
1223 ...
1224 ...
1225 ...
1226 ...
1227 ...
1228 ...
1229 ...
1229 ...
1230 ...
1231 ...
1232 ...
1233 ...
1234 ...
1235 ...
1236 ...
1237 ...
1238 ...
1239 ...
1239 ...
1240 ...
1241 ...
1242 ...
1243 ...
1244 ...
1245 ...
1246 ...
1247 ...
1248 ...
1249 ...
1249 ...
1250 ...
1251 ...
1252 ...
1253 ...
1254 ...
1255 ...
1256 ...
1257 ...
1258 ...
1259 ...
1259 ...
1260 ...
1261 ...
1262 ...
1263 ...
1264 ...
1265 ...
1266 ...
1267 ...
1268 ...
1269 ...
1269 ...
1270 ...
1271 ...
1272 ...
1273 ...
1274 ...
1275 ...
1276 ...
1277 ...
1278 ...
1278 ...
1279 ...
1280 ...
1281 ...
1282 ...
1283 ...
1284 ...
1285 ...
1286 ...
1287 ...
1288 ...
1289 ...
1289 ...
1290 ...
1291 ...
1292 ...
1293 ...
1294 ...
1295 ...
1295 ...
1296 ...
1297 ...
1298 ...
1299 ...
1299 ...
1300 ...
1301 ...
1302 ...
1303 ...
1304 ...
1305 ...
1306 ...
1307 ...
1308 ...
1309 ...
1309 ...
1310 ...
1311 ...
1312 ...
1313 ...
1314 ...
1315 ...
1316 ...
1317 ...
1318 ...
1319 ...
1319 ...
1320 ...
1321 ...
1322 ...
1323 ...
1324 ...
1325 ...
1326 ...
1327 ...
1328 ...
1329 ...
1329 ...
1330 ...
1331 ...
1332 ...
1333 ...
1334 ...
1335 ...
1336 ...
1337 ...
1338 ...
1339 ...
1339 ...
1340 ...
1341 ...
1342 ...
1343 ...
1344 ...
1345 ...
1346 ...
1347 ...
1348 ...
1349 ...
1349 ...
1350 ...
1351 ...
1352 ...
1353 ...
1354 ...
1355 ...
1356 ...
1357 ...
1358 ...
1359 ...
1359 ...
1360 ...
1361 ...
1362 ...
1363 ...
1364 ...
1365 ...
1366 ...
1367 ...
1368 ...
1369 ...
1369 ...
1370 ...
1371 ...
1372 ...
1373 ...
1374 ...
1375 ...
1376 ...
1377 ...
1378 ...
1378 ...
1379 ...
1380 ...
1381 ...
1382 ...
1383 ...
1384 ...
1385 ...
1386 ...
1387 ...
1388 ...
1389 ...
1389 ...
1390 ...
1391 ...
1392 ...
1393 ...
1394 ...
1395 ...
1396 ...
1397 ...
1398 ...
1399 ...
1399 ...
1400 ...
1401 ...
1402 ...
1403 ...
1404 ...
1405 ...
1406 ...
1407 ...
1408 ...
1409 ...
1409 ...
1410 ...
1411 ...
1412 ...
1413 ...
1414 ...
1415 ...
1416 ...
1417 ...
1418 ...
1419 ...
1419 ...
1420 ...
1421 ...
1422 ...
1423 ...
1424 ...
1425 ...
1426 ...
1427 ...
1428 ...
1429 ...
1429 ...
1430 ...
1431 ...
1432 ...
1433 ...
1434 ...
1435 ...
1436 ...
1437 ...
1438 ...
1439 ...
1439 ...
1440 ...
1441 ...
1442 ...
1443 ...
1444 ...
1445 ...
1446 ...
1447 ...
1448 ...
1449 ...
1449 ...
1450 ...
1451 ...
1452 ...
1453 ...
1454 ...
1455 ...
1456 ...
1457 ...
1458 ...
1459 ...
1459 ...
1460 ...
1461 ...
1462 ...
1463 ...
1464 ...
1465 ...
1466 ...
1467 ...
1468 ...
1469 ...
1469 ...
1470 ...
1471 ...
1472 ...
1473 ...
1474 ...
1475 ...
1476 ...
1477 ...
1478 ...
1478 ...
1479 ...
1480 ...
1481 ...
1482 ...
1483 ...
1484 ...
1485 ...
1486 ...
1487 ...
1488 ...
1489 ...
1489 ...
1490 ...
1491 ...
1492 ...
1493 ...
1494 ...
1495 ...
1496 ...
1497 ...
1498 ...
1499 ...
1499 ...
1500 ...
1501 ...
1502 ...
1503 ...
1504 ...
1505 ...
1506 ...
1507 ...
1508 ...
1509 ...
1509 ...
1510 ...
1511 ...
1512 ...
1513 ...
1514 ...
1515 ...
1516 ...
1517 ...
1518 ...
1519 ...
1519 ...
1520 ...
1521 ...
1522 ...
1523 ...
1524 ...
1525 ...
1526 ...
1527 ...
1528 ...
1529 ...
1529 ...
1530 ...
1531 ...
1532 ...
1533 ...
1534 ...
1535 ...
1536 ...
1537 ...
1538 ...
1539 ...
1539 ...
1540 ...
1541 ...
1542 ...
1543 ...
1544 ...
1545 ...
1546 ...
1547 ...
1548 ...
1549 ...
1549 ...
1550 ...
1551 ...
1552 ...
1553 ...
1554 ...
1555 ...
1556 ...
1557 ...
1558 ...
1559 ...
1559 ...
1560 ...
1561 ...
1562 ...
1563 ...
1564 ...
1565 ...
1566 ...
1567 ...
1568 ...
1569 ...
1569 ...
1570 ...
1571 ...
1572 ...
1573 ...
1574 ...
1575 ...
1576 ...
1577 ...
1578 ...
1579 ...
1579 ...
1580 ...
1581 ...
1582 ...
1583 ...
1584 ...
1585 ...
1586 ...
1587 ...
1588 ...
1589 ...
1589 ...
1590 ...
1591 ...
1592 ...
1593 ...
1594 ...
1595 ...
1596 ...
1597 ...
1598 ...
1599 ...
1599 ...
1600 ...
1601 ...
1602 ...
1603 ...
1604 ...
1605 ...
1606 ...
1607 ...
1608 ...
1609 ...
1609 ...
1610 ...
1611 ...
1612 ...
1613 ...
1614 ...
1615 ...
1616 ...
1617 ...
1618 ...
1619 ...
1619 ...
1620 ...
1621 ...
1622 ...
1623 ...
1624 ...
1625 ...
1626 ...
1627 ...
1628 ...
1629 ...
1629 ...
1630 ...
1631 ...
1632 ...
1633 ...
1634 ...
1635 ...
1636 ...
1637 ...
1638 ...
1639 ...
1639 ...
1640 ...
1641 ...
1642 ...
1643 ...
1644 ...
1645 ...
1646 ...
1647 ...
1648 ...
1649 ...
1649 ...
1650 ...
1651 ...
1652 ...
1653 ...
1654 ...
1655 ...
1656 ...
1657 ...
1658 ...
1659 ...
1659 ...
1660 ...
1661 ...
1662 ...
1663 ...
1664 ...
1665 ...
1666 ...
1667 ...
1668 ...
1669 ...
1669 ...
1670 ...
1671 ...
1672 ...
1673 ...
1674 ...
1675 ...
1676 ...
1677 ...
1678 ...
1679 ...
1679 ...
1680 ...
1681 ...
1682 ...
1683 ...
1684 ...
1685 ...
1686 ...
1687 ...
1688 ...
1689 ...
1689 ...
1690 ...
1691 ...
1692 ...
1693 ...
1694 ...
1695 ...
1696 ...
1697 ...
1698 ...
1699 ...
1699 ...
1700 ...
1701 ...
1702 ...
1703 ...
1704 ...
1705 ...
1706 ...
1707 ...
1708 ...
1709 ...
1709 ...
1710 ...
1711 ...
1712 ...
1713 ...
1714 ...
1715 ...
1716 ...
1717 ...
1718 ...
1719 ...
1719 ...
1720 ...
1721 ...
1722 ...
1723 ...
1724 ...
1725 ...
1726 ...
1727 ...
1728 ...
1729 ...
1729 ...
1730 ...
1731 ...
1732 ...
1733 ...
1734 ...
1735 ...
1736 ...
1737 ...
1738 ...
1739 ...
1739 ...
1740 ...
1741 ...
1742 ...
1743 ...
1744 ...
1745 ...
1746 ...
1747 ...
1748 ...
1749 ...
1749 ...
1750 ...
1751 ...
1752 ...
1753 ...
1754 ...
1755 ...
1756 ...
1757 ...
1758 ...
1759 ...
1759 ...
1760 ...
1761 ...
1762 ...
1763 ...
1764 ...
1765 ...
1766 ...
1767 ...
1768 ...
1769 ...
1769 ...
1770 ...
1771 ...
1772 ...
1773 ...
1774 ...
1775 ...
1776 ...
1777 ...
1778 ...
1779 ...
1779 ...
1780 ...
1781 ...
1782 ...
1783 ...
1784 ...
1785 ...
1786 ...
1787 ...
1788 ...
1789 ...
1789 ...
1790 ...
1791 ...
1792 ...
1793 ...
1794 ...
1795 ...
1796 ...
1797 ...
1798 ...
1799 ...
1799 ...
1800 ...
1801 ...
1802 ...
1803 ...
1804 ...
1805 ...
1806 ...
1807 ...
1808 ...
1809 ...
1809 ...
1810 ...
1811 ...
1812 ...
1813 ...
1814 ...
1815 ...
1816 ...
1817 ...
1818 ...
1819 ...
1819 ...
1820 ...
1821 ...
1822 ...
1823 ...
1824 ...
1825 ...
1826 ...
1827 ...
1828 ...
1829 ...
1829 ...
1830 ...
1831 ...
1832 ...
1833 ...
1834 ...
1835 ...
1836 ...
1837 ...
1838 ...
1839 ...
1839 ...
1840 ...
1841 ...
1842 ...
1843 ...
1844 ...
1845 ...
1846 ...
1847 ...
1848 ...
1849 ...
1849 ...
1850 ...
1851 ...
1852 ...
1853 ...
1854 ...
1855 ...
1856 ...
1857 ...
1858 ...
1859 ...
1859 ...
1860 ...
1861 ...
1862 ...
1863 ...
1864 ...
1865 ...
1866 ...
1867 ...
1868 ...
1869 ...
1869 ...
1870 ...
1871 ...
1872 ...
1873 ...
1874 ...
1875 ...
1876 ...
1877 ...
1878 ...
1879 ...
1879 ...
1880 ...
1881 ...
1882 ...
1883 ...
1884 ...
1885 ...
1886 ...
1887 ...
1888 ...
1889 ...
1889 ...
1890 ...
1891 ...
1892 ...
1893 ...
1894 ...
1895 ...
1896 ...
1897 ...
1898 ...
1899 ...
1899 ...
1900 ...
1901 ...
1902 ...
1903 ...
1904 ...
1905 ...
1906 ...
1907 ...
1908 ...
1909 ...
1909 ...
1910 ...
1911 ...
1912 ...
1913 ...
1914 ...
1915 ...
1916 ...
1917 ...
1918 ...
1919 ...
1919 ...
1920 ...
1921 ...
1922 ...
1923 ...
1924 ...
1925 ...
1926 ...
1927 ...
1928 ...
1929 ...
1929 ...
1930 ...
1931 ...
1932 ...
1933 ...
1934 ...
1935 ...
1936 ...
1937 ...
1938 ...
1939 ...
1939 ...
1940 ...
1941 ...
1942 ...
1943 ...
1944 ...
1945 ...
1946 ...
1947 ...
1948 ...
1949 ...
1949 ...
1950 ...
1951 ...
1952 ...
1953 ...
1954 ...
1955 ...
1956 ...
1957 ...
1958 ...
1959 ...
1959 ...
1960 ...
1961 ...
1962 ...
1963 ...
1964 ...
1965 ...
1966 ...
1967 ...
1968 ...
1969 ...
1969 ...
1970 ...
1971 ...
1972 ...
1973 ...
1974 ...
1975 ...
1976 ...
1977 ...
1978 ...
1979 ...
1979 ...
1980 ...
1981 ...
1982 ...
1983 ...
1984 ...
1985 ...
1986 ...
1987 ...
1988 ...
1989 ...
1989 ...
1990 ...
1991 ...
1992 ...
1993 ...
1994 ...
1995 ...
1996 ...
1997 ...
1998 ...
1999 ...
1999 ...
2000 ...
2001 ...
2002 ...
2003 ...
2004 ...
2005 ...
2006 ...
2007 ...
2008 ...
2009 ...
2010 ...
2011 ...
2012 ...
2013 ...
2014 ...
2015 ...
2016 ...
2017 ...
2018 ...
2019 ...
2020 ...
2021 ...
2022 ...
2023 ...
2024 ...
2025 ...
2026 ...
2027 ...
2028 ...
2029 ...
2030 ...
2031 ...
2032 ...
2033 ...
2034 ...
2035 ...
2036 ...
2037 ...
2038 ...
2039 ...
2040 ...
2041 ...
2042 ...
2043 ...
2044 ...
2045 ...
2046 ...
2047 ...
2048 ...
2049 ...
2050 ...
2051 ...
2052 ...
2053 ...
2054 ...
2055 ...
2056 ...
2057 ...
2058 ...
2059 ...
2059 ...
2060 ...
2061 ...
2062 ...
2063 ...
2064 ...
2065 ...
2066 ...
2067 ...
2068 ...
2069 ...
2069 ...
2070 ...
2071 ...
2072 ...
2073 ...
2074 ...
2075 ...
2076 ...
2077 ...
2078 ...
2079 ...
2079 ...
2080 ...
2081 ...
2082 ...
2083 ...
2084 ...
2085 ...
2086 ...
2087 ...
2088 ...
2089 ...
2089 ...
2090 ...
2091 ...
209
```

## Análisis - Semana 8: Listas Enlazadas (Linked Lists)

### Descripción General

La carpeta Semana8 está centrada en **listas enlazadas** (principalmente singly linked lists) y ejemplos prácticos para entender nodos, punteros y operaciones dinámicas sobre colecciones. Es una semana orientada a manipulación de estructuras lineales sin índices numéricos fijos.

---

### Lenguaje

- **JavaScript (ES6+)**
  - **HTML5** (páginas de demostración)
  - **CSS3** (estilos mínimos para la interfaz)
- 

### Estructura de Archivos (estimada)

Semana8/

```
|── linkedlist/ # Implementaciones y demos
| ├── index.html # Interfaz de prueba
| ├── main.js / linkedlist.js # Clases Node y LinkedList
| └── images/ # Recursos (opcional)
```

---

### Objetivos

1. Entender la representación de una Node con value y next.
2. Implementar operaciones básicas: add/append, prepend, remove, find, traverse.
3. Analizar complejidad temporal de cada operación ( $O(1)$ ,  $O(n)$ ).

4. Comparar LinkedList vs Array para seleccionar la estructura adecuada.
  5. Usar la lista en ejemplos prácticos (colas, recorrido, edición dinámica).
- 

## Descripción Detallada

### Implementación típica

- **Clase Node:** almacena value y next.
- **Clase LinkedList:** mantiene head, opcionalmente tail y length.
- **Métodos comunes:**
  - append(value) — agregar al final ( $O(1)$  si existe tail,  $O(n)$  si no)
  - prepend(value) — agregar al inicio ( $O(1)$ )
  - remove(value) — eliminar primer nodo con ese valor ( $O(n)$ )
  - find(predicate) — buscar por condición ( $O(n)$ )
  - toArray() / fromArray() — conversión para interoperabilidad

### Interfaz (index.html)

- Botones para agregar/eliminar/recorrer.
  - Área de visualización que representa nodos y flechas (ej. listas en DOM).
  - Ejemplo didáctico para mostrar cómo cambian head y next.
- 

## Breve Análisis Técnico

### Ventajas de LinkedList

- Inserción/eliminación en extremos:  $O(1)$  con referencias adecuadas.
- Útil cuando las operaciones frecuentes son adiciones/eliminaciones dinámicas.
- No requiere reasignar memoria contigua.

## Desventajas

- Acceso aleatorio por índice:  $O(n)$  (no es adecuado cuando se necesita acceso por índice frecuente).
- Overhead en memoria por referencias next.
- Implementaciones simples con Array pueden ser más rápidas para tamaños pequeños por optimizaciones internas.

## Complejidades clave

- append con tail:  $O(1)$
- append sin tail:  $O(n)$
- prepend:  $O(1)$
- remove/find:  $O(n)$
- toArray:  $O(n)$

## Buenas prácticas y mejoras

- Mantener tail y length para optimizar operaciones y simplificar tests.
- Separar lógica de estructura (ej. linkedlist.js) de la UI (index.html, main.js).
- Añadir pruebas unitarias simples para validar invariantes (p. ej. length, head nulo en lista vacía).



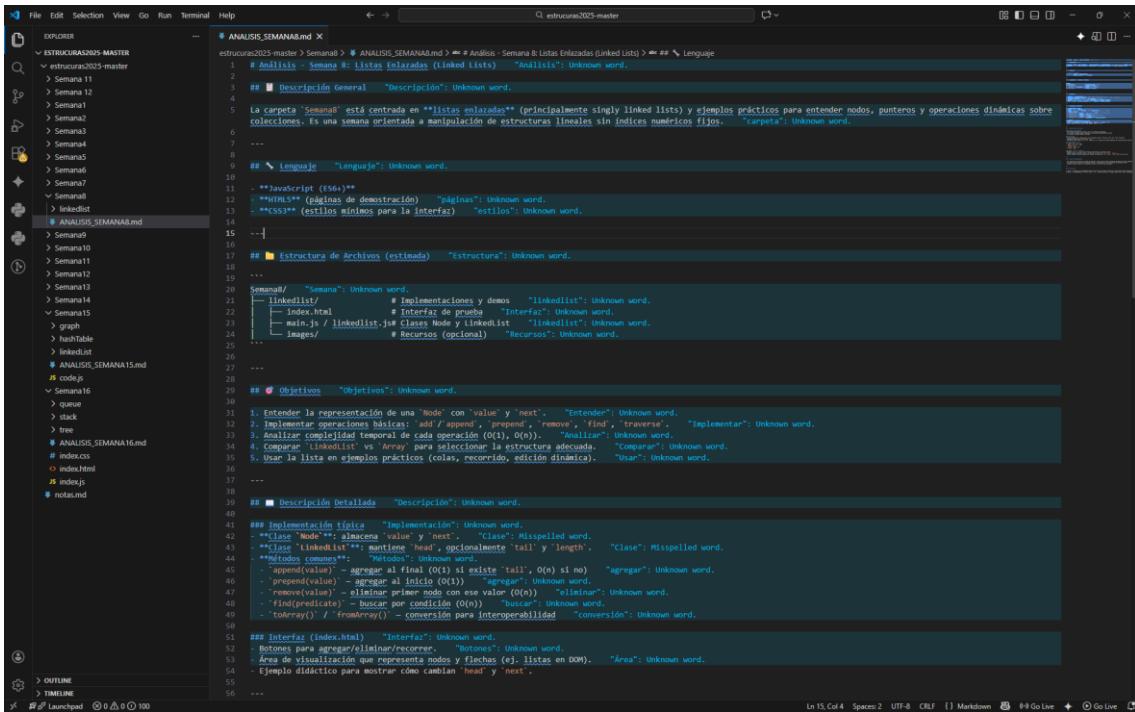
### Puntos de Aprendizaje

- Las linked lists enseñan el concepto de punteros y cómo las estructuras dinámicas difieren de arrays indexados.
- Son base para otras estructuras (stacks, queues, listas dobles) y para algoritmos que manipulan nodos.
- Comparar siempre trade-offs: rendimiento práctico vs teórico según escenario.



### Conclusión

Semana8 es esencial para comprender manejo dinámico de colecciones en memoria y para preparar temas posteriores (listas dobles, colas, y algoritmos que requieren manipulación de nodos). La implementación correcta (uso de tail, manejo de bordes) y pruebas simples harán que los ejemplos sean robustos y reutilizables.



The screenshot shows a code editor window with the title bar "estrucuras2025-master". The left sidebar shows a file tree with several folders and files, including "ESTRUCTURAS2025-MASTER", "ANALISIS\_SEMANA8.md", "ANALISIS\_SEMANA15.md", and "ANALISIS\_SEMANA16.md". The main pane displays a large block of text, which is the output of a code analysis tool. The text is a mix of English and Spanish, with many "Unknown word" errors highlighted in red. The analysis covers topics like singly linked lists, array-like structures, and various methods and interfaces. The code editor interface includes tabs for "OUTLINE" and "TIMELINE" at the bottom.

```
estrucuras2025-master > Semana8 > ANALISIS_SEMANA8.md > Análisis - Semana8: Listas Enlazadas (Linked Lists) > # Análisis - Semana8: Listas Enlazadas (Linked Lists) > # Lenguaje
1 # Análisis - Semana8: Listas Enlazadas (Linked Lists) "Análisis": Unknown word.
2
3 ## ■ Descripción General "Descripción": Unknown word.
4
5 La carpeta "Semana8" está centrada en **listas enlazadas** (principalmente singly linked lists) y ejemplos prácticos para entender nodos, punteros y operaciones dinámicas sobre colecciones. Es una semana orientada a manipulación de estructuras lineales sin índices numéricos fijos. "Carpeta": Unknown word.
6
7 ...
8
9 ## ■ Lenguaje "Lenguaje": Unknown word.
10
11 **JavaScript (ES6)**
12 **HTML5** (páginas de demostración) "páginas": Unknown word.
13 **CSS3** (estilos mínimos para la interfaz) "estilos": Unknown word.
14
15 ...
16
17 ## ■ Estructura de Archivos (estimada) "Estructura": Unknown word.
18
19 ...
20 Semana8/ "Semana8": Unknown word.
21 linkedList/ # Implementaciones y demos "linkedList": Unknown word.
22 index.html # Interfaz de prueba "Interfaz": Unknown word.
23 main.js / linkedList.js Clases Nodo y linkedList "Nodo": Unknown word.
24 images/ Recursos (optional) "Recursos": Unknown word.
25 ...
26
27 ...
28
29 ## ■ Objetivos "Objetivos": Unknown word.
30
31 1. Entender la representación de una "Node" con "value" y "next". "Entender": Unknown word.
32 2. Implementar operaciones básicas: agregar, eliminar, recorrer, "Implementar": Unknown word.
33 3. Implementar el protocolo de cada operación (O(n), O(1)).
34 4. Comparar linkedList vs Array para seleccionar la estructura adecuada. "Comparar": Unknown word.
35 5. Usar la lista en ejemplos prácticos (colas, recorrido, edición dinámica). "Usar": Unknown word.
36
37 ...
38
39 ## ■ Descripción Detallada "Descripción": Unknown word.
40
41 ## Implementación típica "Implementación": Unknown word.
42 **Clase Nodo** almacena "value" next. "Clase": Misspelled word.
43 **Clase linkedList** almacena "value" next, opcionalmente tail y length. "Clase": Misspelled word.
44
45 **métodos comunes** "Métodos": Unknown word.
46 - append(value) - agregar al final (O(1)) si existe 'tail', O(n) si no "agregar": Unknown word.
47 - prepend(value) - agregar al inicio (O(1)) "agregar": Unknown word.
48 - remove(value) - eliminar primer nodo que sea 'value' (O(n)) "eliminar": Unknown word.
49 - find(element) - buscar por condición (O(n)) "encontrar": Unknown word.
50 - toArrayList() / fromArrayList() - conversión para interoperabilidad "conversion": Unknown word.
51
52 ## Interfaz (index.html) "Interfaz": Unknown word.
53 Botones para agregar/eliminar/recorrer "agregar": Unknown word.
54 Área de visualización que representa nodos y flechas (→) listas en DOM. "Área": Unknown word.
55 Ejemplo didáctico para mostrar cómo cambian 'head' y 'next'.
56 ...
57
```

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar displaying a file tree for a project named 'ESTRUCTURAS2023-MASTER'. The current file open in the main editor is 'ANALISIS\_SEMANA8.md'. The content of the file is a series of numbered code comments in English, likely from a GitHub repository. The comments discuss various aspects of linked lists, such as analysis, implementation details, advantages, disadvantages, complexities, and practical applications. The code editor includes standard navigation bars at the top and bottom status bars indicating file size, encoding, and other metadata.

```
1 # Analisis - Semana 8: Listas Enlazadas (Linked Lists) > == ## Lenguaje
2 ## Descripción Detallada "Descripción": Unknown word.
3 ## Interfaz (index.html) "Interfaz": Unknown word.
4 Botones para agregar/eliminar/recuperar "Botones": Unknown word.
5 Áreas de visualización que representan nodos y flechas (ej. listas en DOM). "Área": Unknown word.
6 Ejemplo didáctico para mostrar como cambian head y next.
7 ...
8 ## Breve Análisis Técnico
9 ...
10 ## Ventajas de Linkedlist
11 Inserción/Eliminación en extremos: O(1) con referencias adecuadas.
12 Util cuando las operaciones frecuentes son adiciones/eliminaciones dinámicas.
13 No requiere reasignar memoria contigua.
14 ...
15 ## Desventajas
16 - Acceso aleatorio por índice: O(n) (no es adecuado cuando se necesita acceso por índice frecuente).
17 - Overhead en memoria por referencias next.
18 - Implementaciones simples con Array pueden ser más rápidas para tamaños pequeños por optimizaciones internas.
19 ...
20 ## Complejidades Clave
21 - 'append' con tail: O(1)
22 - 'append' sin tail: O(n)
23 - 'remove': O(1)
24 - 'remove' / 'find': O(n)
25 - 'toarray': O(n)
26 ...
27 ## Buenas prácticas y errores
28 - Mantener 'tail' y 'length' para optimizar operaciones y simplificar tests.
29 - Separar lógica de estructura (ej. linkedlist.js) de la UI ('index.html', 'main.js').
30 - Añadir pruebas unitarias simples para validar invariantes (p. ej. 'length', 'head' nulo en lista vacía).
31 ...
32 ...
33 ## Puntos de Aprendizaje
34 ...
35 - Las linked lists enseñan el concepto de punteros y cómo las estructuras dinámicas difieren de arrays indexados.
36 - Son base para otras estructuras (stacks, queues, listas dobles) y para algoritmos que manipulan nodos.
37 - Comparar siempre trade-offs: rendimiento práctico vs teórico según escenario.
38 ...
39 ...
40 ## Conclusiones
41 ...
42 ...
43 - "Semana 8" es esencial para comprender manejo dinámico de colecciones en memoria y para preparar temas posteriores (listas dobles, colas, y algoritmos que requieren manipulación de nodos). La implementación correcta (uso de 'tail', manejo de bordes) y pruebas simples harán que los ejemplos sean robustos y reutilizables.
44 ...
45 ...
46 ...
47 ...
48 ...
49 ...
50 ...
51 ...
52 ...
53 ...
54 ...
55 ...
56 ...
57 ...
58 ...
59 ...
60 ...
61 ...
62 ...
63 ...
64 ...
65 ...
66 ...
67 ...
68 ...
69 ...
70 ...
71 ...
72 ...
73 ...
74 ...
75 ...
76 ...
77 ...
78 ...
79 ...
80 ...
81 ...
82 ...
83 ...
84 ...
85 ...
86 ...
87 ...
88 ...
89 ...
90 ...
91 ...
92 ...
93 ...
94 ...
95 ...
```

## Análisis - Semana 9: Repaso de Estructuras Fundamentales (Arrays, Listas, Pilas, Colas, Grafos, Tablas Hash, Árboles)

### Descripción General

Semana9 reúne implementaciones y ejemplos de las estructuras de datos fundamentales: **arrays**, **listas enlazadas (singly/doubly)**, **pilas (stacks)**, **colas (queues)**, **grafos (graphs)**, **tablas hash (hash tables)** y **árboles (trees)**. Es una sesión de repaso y consolidación que compara implementaciones nativas de JavaScript con versiones educativas escritas desde cero.

---

### Lenguaje

- **JavaScript (ES6+)** — implementaciones con clases y funciones
  - **HTML5** — páginas de demostración y pruebas
  - **CSS3** — estilos mínimos (si aplica)
- 

### Estructura de Archivos (observada)

Semana9/

```
|── arrays.js # Ejemplos y utilidades con arrays
|── singly_linkedList.js # Implementación de lista simplemente
| enlazada
|── doubly_linkedList.js # Implementación de lista doblemente
| enlazada
|── stack.js # Implementación de pila (LIFO)
|── queue.js # Implementación de cola (FIFO)
|── queues/ # Ejemplos adicionales de colas
|── hash_table.js # Implementación educativa de tabla hash
|── graph.js # Implementación y ejemplos de grafos
|── tree.js # Implementación de árbol (posible BST)
```

---

└── README.md	# Notas y ejemplos de la semana
└── LICENSE	# Licencia del repositorio

---

## Objetivos

1. Repasar la funcionalidad y coste de las estructuras de datos básicas.
  2. Comparar implementaciones nativas (Array, Set, Map) con versiones manuales.
  3. Practicar operaciones clave: inserción, eliminación, búsqueda y recorridos.
  4. Entender trade-offs de memoria y tiempo entre implementaciones.
  5. Conectar estructuras para resolver problemas compuestos (p. ej. BFS usando Queue).
- 

## Descripción Detallada

### **arrays.js**

- Ejemplos de manipulación de arrays: push, pop, shift, unshift, map, filter, reduce.
- Discusión sobre complejidad y cuándo emplear estructuras alternativas.

### **singly\_linkedList.js y doubly\_linkedList.js**

- Implementaciones educativas de Node y LinkedList.
- Métodos típicos: append/prepend, remove, find, toArray.
- doubly\_linkedList añade prev para navegación bidireccional y operaciones más eficientes en extremos.

### **stack.js y queue.js (+ queues/)**

- Stack: push, pop, peek, isEmpty — LIFO; uso con Array o lista enlazada.

- Queue: enqueue, dequeue, peek, isEmpty — FIFO; recomendaciones para O(1) (linked list o buffer circular).
- Ejemplos de uso: evaluación de expresiones (stack), BFS (queue), undo/redo.

### **hash\_table.js**

- Implementación de tabla hash con función hash básica y manejo de colisiones (encadenamiento o probing).
- Métodos: set, get, has, delete y consideraciones sobre rehashing/redimensionado.

### **graph.js**

- Representaciones y ejemplos: lista de adyacencia y posiblemente matrix.
- Algoritmos básicos incluidos o fácilmente añadibles: BFS, DFS.
- Uso práctico: modelar relaciones y traversals.

### **tree.js**

- Implementación de árbol, probablemente Binary Search Tree (BST).
- Operaciones: insert, search, remove y recorridos (in-order, pre-order, post-order).

## Breve Análisis Técnico

### **Comparativa rápida**

- Array (nativo): ideal para acceso por índice O(1), menos eficiente para inserciones al frente (O(n)).
- LinkedList: inserciones/eliminaciones O(1) en extremos, acceso aleatorio O(n).
- Stack/Queue: patrones de acceso simples que se pueden implementar sobre Array o LinkedList según necesidades.

- Hash Table: acceso promedio  $O(1)$ , sensible a colisiones y factor de carga.
- Graph/Tree: estructuras no lineales; elección de representación afecta memoria y eficiencia de algoritmos.

### Complejidades (resumen)

- Acceso por índice (Array):  $O(1)$
- Insert/Remove en medio (Array):  $O(n)$
- Insert/Remove en extremos (LinkedList con tail/head):  $O(1)$
- Stack/Queue operations:  $O(1)$  (implementadas adecuadamente)
- HashTable get/set:  $O(1)$  promedio,  $O(n)$  peor caso
- BST operations:  $O(\log n)$  promedio,  $O(n)$  peor caso (degeneración)
- Graph traversals (BFS/DFS):  $O(V + E)$

### Recomendaciones pedagógicas

- Mantener implementaciones sencillas y bien comentadas para ilustrar comportamiento interno.
- Añadir tests simples y ejemplos de uso (ej. usar queue en BFS) para reforzar aprendizaje.
- Documentar limitaciones (por ejemplo, cuando usar Array vs LinkedList).

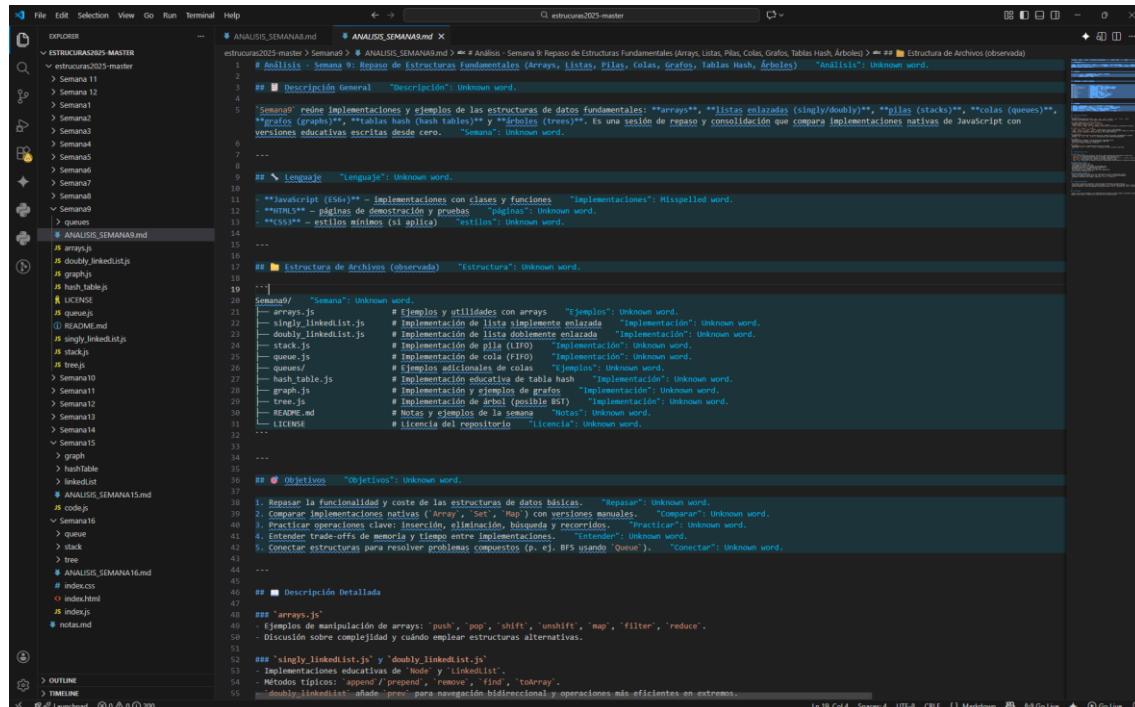


### Puntos de Aprendizaje

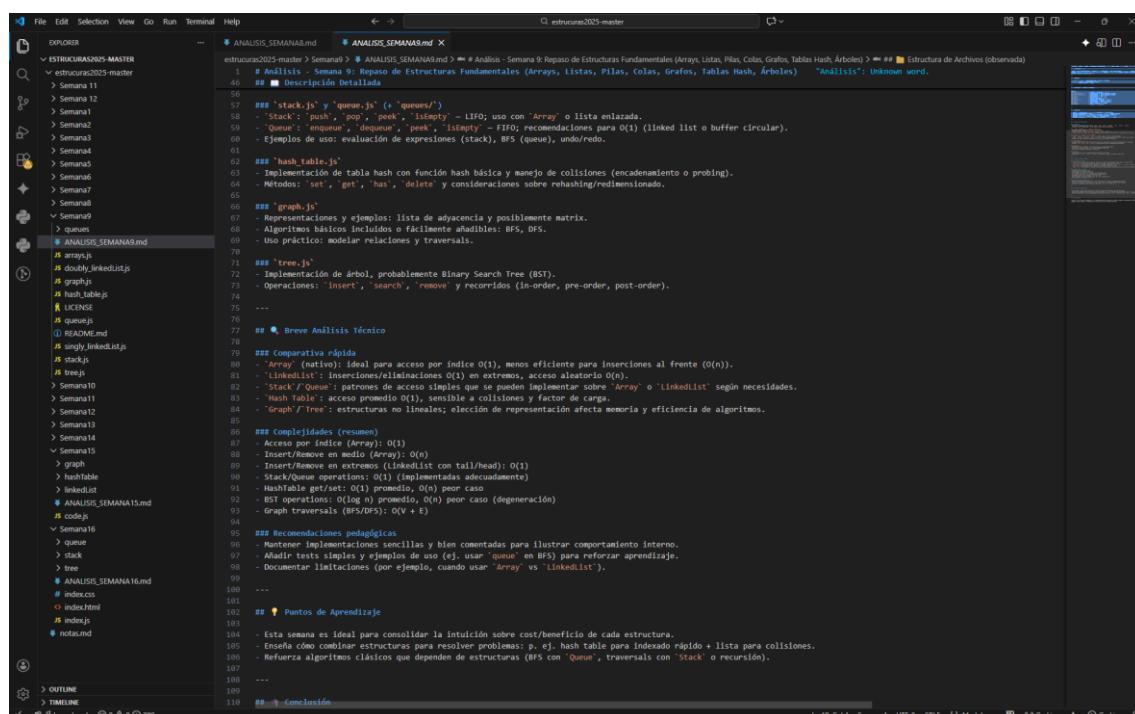
- Esta semana es ideal para consolidar la intuición sobre cost/beneficio de cada estructura.
- Enseña cómo combinar estructuras para resolver problemas: p. ej. hash table para indexado rápido + lista para colisiones.
- Refuerza algoritmos clásicos que dependen de estructuras (BFS con Queue, traversals con Stack o recursión).

## 🎓 Conclusión

Semana9 actúa como un checkpoint: reúne implementaciones esenciales y permite comparar directamente estrategias y complejidades. Reforzar con pruebas y ejemplos aplicados (pequeños retos) ayudará a internalizar cuándo y por qué usar cada estructura.



```
ANALISIS_SEMANA8.md
ANALISIS_SEMANA8.md
estrucutras2025 master > Semana9 > ANALISIS_SEMANA8.md > # Análisis - Semana 9: Repaso de Estructuras Fundamentales (Arrays, Listas, Pilas, Colas, Grafos, Tablas Hash, Árboles) > == ## Estructura de Archivos (observado)
1 # Análisis - Semana 9: Repaso de Estructuras Fundamentales (Arrays, Listas, Pilas, Colas, Grafos, Tablas Hash, Árboles) "Análisis": Unknown word
2
3 ## ■ Descripción General "Descripción": Unknown word.
4
5 ## Semana9 "Semana": Unknown word.
6
7 ## ■ Lenguaje "Lenguaje": Unknown word.
8
9 ## ■ Implementaciones y ejemplos de las estructuras de datos fundamentales: "arrays**", **"listas enlazadas (singly/doubly)**", **"piles (stacks)**", **"colas (queues)**", **"grafos (graphs)**", **"tablas hash (hash tables)**" y **"árboles (trees)**". Es una sesión de repaso y consolidación que compara implementaciones nativas de Javascript con versiones educativas escritas desde cero. "Semana": Unknown word.
10
11
12 ## ■ Lenguaje "Lenguaje": Unknown word.
13
14 ## ■ Implementaciones con clases y funciones "Implementaciones": Misspelled word.
15
16 ## ■ HTML5** - páginas de demostración y pruebas "páginas": Unknown word.
17
18 ## ■ CSS** - estilos mínimos (sí aplica) "estilos": Unknown word.
19
20
21 ## ■ Estructura de Archivos (observado) "Estructura": Unknown word.
22
23
24
25 ## ■ Descripción General "Descripción": Unknown word.
26
27 ## ■ Ejemplos adicionales de colas "Ejemplos": Unknown word.
28
29 ## ■ Implementación educativa de tabla hash "Implementación": Unknown word.
30
31 ## ■ Ejemplos adicionales de grafos "Ejemplos": Unknown word.
32
33 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
34
35 ## ■ Ejemplos adicionales de listas enlazadas "Ejemplos": Unknown word.
36
37 ## ■ Objetivos "Objetivos": Unknown word.
38
39 1. Repasar la funcionalidad y coste de las estructuras de datos básicas. "Repasar": Unknown word.
40 2. Comparar implementaciones nativas ('Array', 'Set', 'Map') con versiones manuales. "Comparar": Unknown word.
41 3. Practicar operaciones clave: inserción, eliminación, búsqueda y recorridos. "Practicar": Unknown word.
42 4. Entender trade-offs de memoria y tiempo entre implementaciones. "Entender": Unknown word.
43 5. Conectar estructuras para resolver problemas compuestos (p. ej. BFS usando 'queue'). "Conectar": Unknown word.
44
45
46 ## ■ Descripción Detallada "Descripción": Unknown word.
47
48 ## ■ arrays.js "arrays": Unknown word.
49 Ejemplos de manipulación de arrays: "push", "pop", "shift", "unshift", "map", "filter", "reduce".
50 - Discusión sobre complejidad y cuándo emplear estructuras alternativas.
51
52 ## ■ singly_LinkedList.js y 'doubly_LinkedList.js'
53 Implementaciones educativas de linkedList".
54 - Métodos típicos: 'append' / 'prepend', 'remove', 'find', 'isEmpty'.
55 - 'doubly_LinkedList': añade 'prev' para navegación bidireccional y operaciones más eficientes en extremos.
56
57 ## ■ stack.js y 'queue.js' "queue": Unknown word.
58 - Stack: 'push', 'pop', 'peek', 'isEmpty' - LIFO; uso con 'Array' o lista enlazada.
59 - Queue: 'enqueue', 'dequeue', 'peek', 'isEmpty' - FIFO; recomendaciones para O(1) (linked list o buffer circular).
60 - Ejemplos de uso: evaluación de expresiones (stack), BFS (queue), undo/redo.
61
62 ## ■ hash_table.js "hash_table": Unknown word.
63 - Implementación de tabla hash con función hash básica y manejo de colisiones (encadenamiento o probing).
64 - Métodos: 'set', 'get', 'has', 'delete' y consideraciones sobre rehashing/redimensionado.
65
66 ## ■ graphs.js "graphs": Unknown word.
67 - Representaciones y ejemplos: lista de adyacencia y posiblemente matriz.
68 - Algoritmos básicos incluidos o fácilmente仿写: BFS, DFS.
69 - Uso práctico: modelar relaciones y tráves.
70
71 ## ■ tree.js "tree": Unknown word.
72 - Implementación de árbol, probablemente Binary Search Tree (BST).
73 - Operaciones: 'insert', 'search', 'remove' y recorridos (in-order, pre-order, post-order).
74
75
76
77 ## ■ Breve Análisis Técnico "Breve Análisis Técnico": Unknown word.
78
79 ## ■ Comparativa rápida "Comparativa": Unknown word.
80 - Array (nativo): ideal para acceso por índice O(1), menos eficiente para inserciones al frente O(n).
81 - Listas enlazadas: inserciones/eliminaciones O(1) en extremos, acceso aleatorio O(n).
82 - Stack/Queue: patrones de acceso simples que se pueden implementar sobre 'Array' o 'LinkedList' según necesidades.
83 - Hash Table: acceso promedio O(1), sensible a colisiones y factor de carga.
84 - Graph/Tree: estructuras no lineales; elección de representación afecta memoria y eficiencia de algoritmos.
85
86 ## ■ Complejidades (resumen) "Complejidades": Unknown word.
87 - Acceso por índice (Array): O(1)
88 - Insert/Remove en medio (Array): O(n)
89 - Insert/Remove en extremos (LinkedList con tail/head): O(1)
90 - Stack/Queue operaciones: O(1) (implementadas adecuadamente)
91 - Hash Table promedio: O(1) (ideal para caso promedio)
92 - BST operaciones: O(log n) promedio, O(n) peor caso (degeneración)
93 - Graph traversals (BFS/DFS): O(V + E)
94
95 ## ■ Recomendaciones pedagógicas "Recomendaciones": Unknown word.
96 - Mantener implementaciones sencillas y bien comentadas para ilustrar comportamiento interno.
97 - Añadir tests simples y ejemplos de uso (ej. usar 'queue' en BFS) para reforzar aprendizaje.
98 - Documentar limitaciones (por ejemplo, cuando usar 'Array' vs 'LinkedList').
99
100
101 ## ■ Puntos de Aprendizaje "Puntos de Aprendizaje": Unknown word.
102
103
104 - Esta semana es ideal para consolidar la intuición sobre cost/beneficio de cada estructura.
105 - Enseña cómo combinar estructuras para resolver problemas: p. ej. hash table para indexado rápido + lista para colisiones.
106 - Reforzar algoritmos clásicos que dependen de estructuras (BFS con 'Queue', traversals con 'Stack' o recursión).
107
108
109 ## ■ Conclusion "Conclusion": Unknown word.
110
```



```
ANALISIS_SEMANA9.md
ANALISIS_SEMANA9.md
estrucutras2025 master > Semana9 > ANALISIS_SEMANA9.md > # Análisis - Semana 9: Repaso de Estructuras Fundamentales (Arrays, Listas, Pilas, Colas, Grafos, Tablas Hash, Árboles) > == ## Estructura de Archivos (observado)
1 # Análisis - Semana 9: Repaso de Estructuras Fundamentales (Arrays, Listas, Pilas, Colas, Grafos, Tablas Hash, Árboles) "Análisis": Unknown word.
2
3 ## ■ Descripción Detallada "Descripción": Unknown word.
4
5 ## ■ Ejemplos adicionales de colas "Ejemplos": Unknown word.
6
7 ## ■ Implementación educativa de tabla hash "Implementación": Unknown word.
8
9 ## ■ Ejemplos adicionales de grafos "Ejemplos": Unknown word.
10
11 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
12
13 ## ■ Ejemplos adicionales de listas enlazadas "Ejemplos": Unknown word.
14
15 ## ■ Descripción General "Descripción": Unknown word.
16
17 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
18
19 ## ■ Implementación educativa de tabla hash "Implementación": Unknown word.
20
21 ## ■ Ejemplos adicionales de grafos "Ejemplos": Unknown word.
22
23 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
24
25 ## ■ Descripción General "Descripción": Unknown word.
26
27 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
28
29 ## ■ Implementación educativa de tabla hash "Implementación": Unknown word.
30
31 ## ■ Ejemplos adicionales de grafos "Ejemplos": Unknown word.
32
33 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
34
35 ## ■ Descripción General "Descripción": Unknown word.
36
37 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
38
39 ## ■ Implementación educativa de tabla hash "Implementación": Unknown word.
40
41 ## ■ Ejemplos adicionales de grafos "Ejemplos": Unknown word.
42
43 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
44
45 ## ■ Descripción General "Descripción": Unknown word.
46
47 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
48
49 ## ■ Implementación educativa de tabla hash "Implementación": Unknown word.
50
51 ## ■ Ejemplos adicionales de grafos "Ejemplos": Unknown word.
52
53 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
54
55 ## ■ Descripción General "Descripción": Unknown word.
56
57 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
58
59 ## ■ Implementación educativa de tabla hash "Implementación": Unknown word.
60
61 ## ■ Ejemplos adicionales de grafos "Ejemplos": Unknown word.
62
63 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
64
65 ## ■ Descripción General "Descripción": Unknown word.
66
67 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
68
69 ## ■ Implementación educativa de tabla hash "Implementación": Unknown word.
70
71 ## ■ Ejemplos adicionales de grafos "Ejemplos": Unknown word.
72
73 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
74
75 ## ■ Descripción General "Descripción": Unknown word.
76
77 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
78
79 ## ■ Implementación educativa de tabla hash "Implementación": Unknown word.
80
81 ## ■ Ejemplos adicionales de grafos "Ejemplos": Unknown word.
82
83 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
84
85 ## ■ Descripción General "Descripción": Unknown word.
86
87 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
88
89 ## ■ Implementación educativa de tabla hash "Implementación": Unknown word.
90
91 ## ■ Ejemplos adicionales de grafos "Ejemplos": Unknown word.
92
93 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
94
95 ## ■ Descripción General "Descripción": Unknown word.
96
97 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
98
99 ## ■ Implementación educativa de tabla hash "Implementación": Unknown word.
100
101 ## ■ Ejemplos adicionales de grafos "Ejemplos": Unknown word.
102
103 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
104
105 ## ■ Descripción General "Descripción": Unknown word.
106
107 ## ■ Ejemplos adicionales de pilas "Ejemplos": Unknown word.
108
109 ## ■ Implementación educativa de tabla hash "Implementación": Unknown word.
110
```

## Análisis - Semana 10: Árboles, Tablas y Estructuras Auxiliares

### Descripción General

La carpeta Semana10 agrupa implementaciones y ejercicios sobre **árboles (binary trees, tree)**, **tablas (hash/dictionary)** y estructuras auxiliares como **listas enlazadas y pilas**. El material mezcla implementaciones educativas (desde cero) con ejemplos prácticos para entender algoritmos de búsqueda, inserción y recorridos.

---

### Lenguaje

- **JavaScript (ES6+)**
- **HTML5** (páginas de ejemplo)
- **CSS3** (estilos básicos)

Las implementaciones usan JS puro, clases y estructuras nativas (Map, Set, Array) cuando conviene.

---

### Estructura de Archivos (observada)

```
Semana10/
├── alternative code.txt # Notas / código alternativo
├── binaryTree.js # Implementación de árbol binario (insert, search,
 # recorridos)
├── code.js # Ejemplos y utilidades generales
├── dictionary.js # Implementación/uso de diccionarios (key-value)
├── hastTable.js # (typo) implementación de hash table educativa
├── index.html # Página de demostración
├── linkedList.js # Implementación de lista enlazada
└── Set.js # Ejemplo/implementación de Set personalizado
```

```
├── style.css # Estilos para las páginas
├── stack/ # Implementación de pilas (stack.js)
| └── stack.js
└── tree/ # Implementación de árboles (posible BST / utilities)
 └── tree.js
```

---

## Objetivos

1. Comprender y manipular **árboles binarios**: inserción, búsqueda y recorridos (in-, pre-, post-order).
2. Implementar y comparar **tablas hash / diccionarios** con estructuras nativas (Map) y manuales.
3. Repasar y aplicar **listas enlazadas y pilas** como apoyo en algoritmos de árboles.
4. Analizar complejidad y casos degenerados (árboles no balanceados, colisiones en hash).
5. Entender cuándo usar estructuras nativas vs implementaciones educativas.

---

## Descripción Detallada

### **binaryTree.js / tree/tree.js**

- Implementación de un árbol binario (posiblemente BST).
- Operaciones típicas:
  - `insert(value)` — insertar valor en posición correcta
  - `search(value)` — buscar un nodo
  - `remove(value)` — (si está implementado) eliminación con reasignación adecuada
  - Recorridos: `inOrder`, `preOrder`, `postOrder`

- Uso pedagógico: entender recursión y propiedades de árboles.

### **hashTable.js (hash table)**

- Implementación educativa de una hash table (nota: nombre del archivo tiene un typo).
- Elementos a revisar:
  - Función hash simple para convertir keys en índices
  - Manejo de colisiones (encadenamiento o probing)
  - Métodos: set, get, has, delete
- Comparación con dictionary.js (puede mostrar uso de Object o Map).

### **dictionary.js**

- Ejemplos de uso de diccionarios en JS: objetos planos o Map para key-value stores.
- Buenas prácticas: evitar usar objetos para claves no-string; preferir Map para claves arbitrarias.

### **linkedList.js y stack/stack.js**

- Soporte para operaciones auxiliares en árboles (p. ej. traversal iterativo requiere stack o queue).
- linkedList.js ofrece nodos y métodos (append, remove, find) útiles para colas/pilas personalizadas.
- stack.js implementa LIFO; útil para DFS iterativo o evaluaciones.

### **index.html y style.css**

- Interfaz para probar las implementaciones y visualizar recorridos o resultados.
- Estilos mínimos para presentar nodos y estructuras.



## Árboles (BST)

- Operaciones promedio: insert, search →  $O(\log n)$  si el árbol está balanceado.
- Peor caso:  $O(n)$  cuando el árbol está degenerado (inserciones ordenadas).
- Recomendación pedagógica: mostrar primero BST simple y luego comentar sobre balanceo (AVL/Red-Black).

## Tablas Hash / Diccionarios

- get/set promedio  $O(1)$ ; peor caso  $O(n)$  por colisiones.
- Importante: elegir o diseñar una función hash adecuada y controlar el factor de carga con rehashing.
- Para JS real, Map ofrece comportamiento robusto y manejo de claves no-string.

## Estructuras auxiliares

- Stack y Queue son esenciales para versiones iterativas de traversals (DFS/BFS).
- LinkedList puede ofrecer  $O(1)$  en enqueue/dequeue si se mantiene head y tail.

## Calidad del código y mejoras

- Corregir el nombre hastTable.js a hashTable.js para evitar confusión.
- Añadir comentarios y ejemplos de uso en README.md para cada implementación.
- Incluir tests simples que demuestren invariantes (por ejemplo, insert seguido de search debe encontrar el valor).



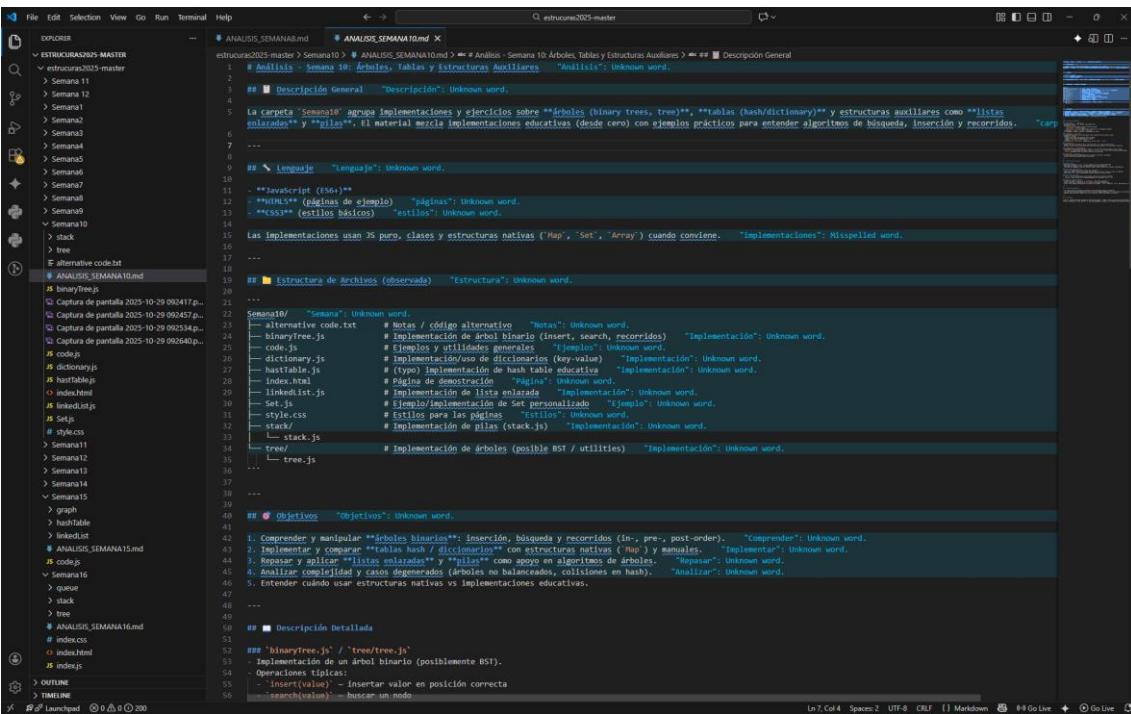
### Puntos de Aprendizaje

- Los árboles muestran claramente la diferencia entre rendimiento promedio y degenerado.

- Las hash tables requieren pensamiento sobre colisiones y redimensionamiento; Map es preferible en producción.
- Combinar estructuras (por ejemplo, usar Stack para DFS) ilustra cómo componer soluciones.

## Conclusión

Semana10 consolida conceptos avanzados de estructuras de datos: árboles y tablas son pilares para algoritmos eficientes. Fortalecer las implementaciones con limpieza de nombres, documentación y tests mejorará la usabilidad pedagógica y preparará al estudiante para estudiar balanceo de árboles, hashing robusto y algoritmos de búsqueda avanzados.



```

File Edit Selection View Go Run Terminal Help
ANALISIS_SEMANA10.md ANALISIS_SEMANA10.md
estrucuras2025-master > ANALISIS_SEMANA10.md > # Análisis - Semana 10: Árboles, Tablas y Estructuras Auxiliares > ## Descripción General
1 # Análisis - Semana 10: Árboles, Tablas y Estructuras Auxiliares "Análisis": Unknown word.
2
3 ## Descripción General "Descripción": Unknown word.
4
5 La carpeta "Semana10" agrupa implementaciones y ejercicios sobre **árboles (binary trees, tree)**, **tablas (hash/dictionary)** y estructuras auxiliares como **listas enlazadas** y **pilas**. El material mezcla implementaciones educativas (desde cero) con ejemplos prácticos para entender algoritmos de búsqueda, inserción y recorridos.
6
7 ...
8
9 ## Lenguaje "Lenguaje": Unknown word.
10
11 - **JavaScript (Ejemplos)**
12 - **HTML5** (páginas de ejemplo) "páginas": Unknown word.
13 - **CSS3** (estilos básicos) "estilos": Unknown word.
14
15 Las implementaciones usan JS puro, clases y estructuras nativas ("Map", "Set", "Array") cuando conviene. "Implementaciones": Misspelled word.
16
17 ...
18
19 ## Estructura de Archivos (observada) "Estructura": Unknown word.
20
21 ...
22
23 Semana10/ "Semana10": Unknown word.
24 - alternative code.txt # Notas / código alternativo "Notas": Unknown word.
25 - binarytree.js # Implementación de árbol binario (insert, search, recorridos) "Implementación": Unknown word.
26 - code.js # Ejemplos y utilidades generales "Ejemplos": Unknown word.
27 - dictionary.js # Implementación/uso de diccionarios (key-value) "Implementación": Unknown word.
28 - index.html # Página de inicio "Página": Unknown word.
29 - linkedlist.js # Implementación de lista enlazada "Implementación": Unknown word.
30 - Set.js # Ejemplo/implementación de Set personalizado "Ejemplo": Unknown word.
31 - style.css # Estilos para las páginas "Estilos": Unknown word.
32 - stack.js # Implementación de pilas (stack.js) "Implementación": Unknown word.
33
34 tree/ # Implementación de árboles (posible BST / utilities) "Implementación": Unknown word.
35 - tree.js ...
36
37 ...
38
39 ## Objetivos "Objetivos": Unknown word.
40
41 1. Comprender y manipular **árboles binarios** (inserción, búsqueda y recorridos (In-, pre-, post-order)). "Comprender": Unknown word.
42 2. Implementar y comparar **tablas hash / diccionarios** con estructuras nativas ("Map") y arrays. "Implementar": Unknown word.
43 3. Repasar y aplicar **listas enlazadas** y **pilas** como apoyo en algoritmos de árboles. "Repasar": Unknown word.
44 4. Analizar complejidad y casos degenerados (árboles no balanceados, colisiones en hash). "Analizar": Unknown word.
45 5. Entender cuándo usar estructuras nativas vs implementaciones educativas.
46
47 ...
48
49 ## Descripción Detallada
50
51 #### binarytree.js / tree/tree.js
52 - Implementación de un árbol binario (posiblemente BST).
53 - Operaciones típicas:
54 - - insert(value) - Insertar valor en posición correcta
55 - - search(value) - buscar un nodo
56

```

File Edit Selection View Go Run Terminal Help

EXPLORER ANALISIS\_SEMANA10.md

ESTRUCTURAS2025-MASTER ANALISIS\_SEMANA10.md # Análisis - Semana 10. Árboles, Tablas y Estructuras Auxiliares > ## Descripción General

```

ANALISIS_SEMANA10.md X
estrucuras2025-master> Semana10> ANALISIS_SEMANA10.md # Análisis - Semana 10. Árboles, Tablas y Estructuras Auxiliares > ## Descripción General
1 ## Descripción Detallada
2 #### 'binarytree.js' / 'tree/tree.js'
3 - 'search(value)' - buscar un nodo
4 - 'remove(value)' - (si está implementado) eliminación con reasignación adecuada
5 - 'removeValue()' - (si está implementado) eliminación con reasignación adecuada
6 - 'removeValue(value)' - (si está implementado) eliminación con reasignación adecuada
7 - 'removeValue(value, postorder)'
8 - Uso pedagógico: entender recursión y propiedades de árboles.
9
10 #### 'hashTable.js' (Hash table)
11 - Implementación intuitiva de una hash table (nota: nombre del archivo tiene un tipo).
12 - Ejemplos de uso: inserción y eliminación.
13 - Función hash simple para convertir keys en índices
14 - Manejo de colisiones (encadenamiento o probing)
15 - Métodos: 'set', 'get', 'has', 'delete'
16 - Comparación con 'dictionary.js' (puede mostrar uso de 'Object' o 'Map').
17
18 - Ejemplos de uso de diccionarios en JS: objetos planos o 'Map' para key-value stores.
19 - Buenas prácticas: evitar usar objetos para claves no-string; preferir 'Map' para claves arbitrarias.
20
21 #### 'linkedlist.js' y 'stack/stack.js'
22 - Soporte para operaciones auxiliares en árboles (p. ej. traversal iterativo requiere stack o queue).
23 - 'linkedlist.js' ofrece nodos y métodos ('append', 'remove', 'find') útiles para colas/pilas personalizadas.
24 - 'stack.js' implementa LIFO; útil para DFS iterativo o evaluaciones.
25
26 #### 'index.html' y 'style.css'
27 - Interfaz para probar las implementaciones y visualizar recorridos o resultados.
28 - Estilos mínimos para presentar nodos y estructuras.
29
30 #### Breve Análisis Técnico
31
32 #### Árboles (BST)
33 - Operaciones promedio: 'insert', 'search' = O(log n) si el árbol está balanceado.
34 - Peor caso: O(n) cuando el árbol está degenerado (inserciones ordenadas).
35 - Recomendación pedagógica: mostrar primero BST simple y luego comentar sobre balanceo (AVL/Red-Black).
36
37 #### Tablas Hash / Diccionarios
38 - 'get' / 'set' promedio O(1); peor caso O(n) por colisiones.
39 - Importante: elegir o diseñar una función hash adecuada y controlar el factor de carga con rehashing.
40 - Para JS real, 'Map' ofrece comportamiento robusto y manejo de claves no-string.
41
42 #### Estructuras auxiliares
43 - 'Stack' y 'Queue' son esenciales para versiones iterativas de traversals (DFS/BFS).
44 - 'LinkedList' puede ofrecer O(1) en enqueues/dequeues si se mantiene 'head' y 'tail'.
45
46 #### Calidad del código y mejoras
47 - Corregir el nombre 'hashTable.js' a 'hashTable.js' para evitar confusión.
48 - Añadir comentarios y ejemplos de uso en 'README.md' para cada implementación.
49 - Incluir tests simples que demuestren invariantes (por ejemplo, 'insert' seguido de 'search' debe encontrar el valor).
50
51 #### Puntos de Aprendizaje
52
53 - Los árboles muestran claramente la diferencia entre rendimiento promedio y degenerado.

```

IntelliJ IDEA 2023.2.3 | Space2 | UTF-8 | CMF | Markdown | 445 Scans | 4 | 0 Scans | 200

## Análisis - Semana 11: Aplicación Web de Productos (HTML, CSS, JS, JSON)

### Descripción General

La carpeta Semana11 contiene una pequeña aplicación web que muestra un catálogo de productos estático. Se centra en integrar **HTML5**, **CSS3** y **JavaScript** para leer datos en formato **JSON** (products.json) y renderizar una vista interactiva en el navegador, con recursos en la carpeta assets.

---

### Lenguaje

- **HTML5** — estructura de la página (index.html)
  - **CSS3** — estilos en style.css ( posible uso de variables y clases)
  - **JavaScript (ES6+)** — lógica en app.js para cargar y renderizar products.json
  - **JSON** — datos de producto (products.json)
- 

### Estructura de Archivos

Semana11/

```
|── index.html # Página principal de la tienda / demo
|── app.js # Lógica de la aplicación: carga y renderizado de
| productos
|── products.json # Datos de ejemplo (array de objetos producto)
|── style.css # Estilos de la UI
└── assets/ # Imágenes, íconos y otros recursos estáticos
```

---

### Objetivos

1. Aprender a consumir datos locales en JSON y renderizarlos dinámicamente en el DOM.

2. Practicar manipulación del DOM y creación de componentes HTML vía JS (cards, listas).
  3. Implementar interactividad básica: filtrado, búsqueda, botones o handlers.
  4. Entender limitaciones de cargar JSON localmente (CORS / servidor vs file://).
  5. Mejorar estilos y disposición visual con style.css.
- 

## Descripción Detallada

### **index.html**

- Plantilla principal que incluye estructura semántica (header, main, footer) y un contenedor donde app.js inyecta productos.
- Incluye referencia a style.css y app.js.

### **app.js**

- Funciones clave esperadas:
  - Cargar products.json mediante fetch() o XMLHttpRequest.
  - Parsear y transformar datos (map/filter) para preparar la representación.
  - Renderizar elementos (cards, listas) en el DOM.
  - Añadir listeners para interactividad (p. ej. filtros, botones "añadir al carrito").
- Posibles consideraciones: manejo de errores en fetch, estados de carga (skeleton/loading), y separación de responsabilidades (render vs data).

### **products.json**

- Array de objetos con campos típicos: id, title, description, price, image, category.
- Útil para practicar paginación, filtros por categoría y ordenamientos.

## **style.css y assets/**

- style.css define la estética: grid/flex layout para cards, tipografías, colores y responsive.
  - assets/ contiene imágenes y recursos referenciados por products.json o index.html.
- 

### Breve Análisis Técnico

#### **1. Consumo de JSON local**

- fetch('products.json') funciona correctamente si la página se sirve por HTTP(S) (p. ej. localhost).
- Si se abre desde file://, algunos navegadores bloquean fetch por políticas CORS/archivo; usar un servidor local (p. ej. npx http-server o python -m http.server) para pruebas.

#### **2. Renderizado y rendimiento**

- Renderizar muchos productos puede impactar el DOM; usar fragmentos de documento (DocumentFragment) o render por lotes mejora rendimiento.
- Para render dinámico: crear plantillas (template literals) o <template> en HTML y clonarlo.

#### **3. UX e interactividad**

- Añadir estado de carga y manejo de errores mejora la experiencia.
- Funcionalidades recomendadas: búsqueda en vivo, filtros por categoría, ordenamiento por precio, paginación, y un pequeño carrito local (localStorage).

#### **4. Accesibilidad y SEO**

- Usar alt en imágenes, roles ARIA para componentes dinámicos y botones accesibles.
- Si el sitio se despliega estáticamente, considerar prerendering para SEO (no obligatorio para ejercicios).

## 5. Mejora del código y modularidad

- Separar la lógica en módulos: api.js (fetch), ui.js (render), utils.js.
  - Evitar lógica inline en index.html; usar addEventListener en app.js.
  - Añadir pruebas básicas (p. ej. comprobaciones unitarias de funciones puras).
- 



### Puntos de Aprendizaje

- Integración práctica de fetch() + JSON con DOM es una habilidad central en desarrollo web.
  - Diferencias entre desarrollo local y servido: siempre probar con un servidor local.
  - Mejoras incrementales (filtrado, paginación, caching) transforman una demo en una mini-app real.
- 



### Conclusión

Semana11 ofrece un ejercicio práctico y completo para consolidar HTML/CSS/JS con datos JSON. Es excelente para practicar el flujo data → transform → render y para introducir buenas prácticas (modularidad, manejo de errores y accesibilidad). Para preparar despliegue o pruebas más reales, ejecutar la app en un servidor local y añadir controles de interacción avanzados.

The terminal window displays the output of a code analysis tool (likely ESLint or similar) on the file ANALISIS\_SEMANA11.md. The output is color-coded with red for errors and yellow for warnings. The code itself is a mix of Markdown and JSON-like configuration files.

```
File Edit Selection View Go Run Terminal Help <- > ANALISIS_SEMANA11.md & ANALISIS_SEMANA11.md & # Análisis - Semana 11: Aplicación Web de Productos (HTML, CSS, JS, JSON) & ## 🔍 Lenguaje
estructura2025-master > ANALISIS_SEMANA11.md > # Análisis - Semana 11: Aplicación Web de Productos (HTML, CSS, JS, JSON) > ## 🔍 Lenguaje
1 2
3 ## 📄 Descripción General "Descripción": Unknown word.
4
5 La carpeta "Semana11" contiene una pequeña aplicación web que muestra un catálogo de productos estático. Se centra en integrar **"HTML5**", **"CSS3** y **"JavaScript** para leer datos en formato **"JSON** (products.json) y renderizar una vista interactiva en el navegador, con recursos en la carpeta "assets". "Carpeta": Unknown word.
6
7 ...
8 ## 🔍 Lenguaje "Lenguaje": Unknown word.
9
10 ...
11 ## **HTML5** - estructura de la página ("index.html") "estructura": Unknown word.
12 ...
13 ## **CSS3** - estilos en style.css (posible uso de variables y clases) "estilos": Unknown word.
14 ...
15 ## **JavaScript (ES6+)** - lógica en app.js para cargar y renderizar products.json "lógica": Unknown word.
16 ...
17 ## 📁 Estructura de Archivos "Estructura": Unknown word.
18 ...
19 ...
20 ...
21 Semana11/ "Semana": Unknown word.
22 ...
23 ...
24 ...
25 ...
26 ...
27 ...
28 ...
29 ...
30 ...
31 ## 🌟 Objetivos "Objetivos": Unknown word.
32
33 1. Aprender a consumir datos locales en "JSON" y renderizarlos dinámicamente en el DOM. "Aprender": Unknown word.
34 2. Practicar manipulación del DOM y creación de componentes HTML vía JS (cards, listas). "Practicar": Unknown word.
35 3. Implementar interactividad básica: filtrado, búsqueda, botones o handlers. "Implementar": Unknown word.
36 4. Entender limitaciones de cargar "JSON" localmente ("JSON" vs "fetch"). "Entender": Unknown word.
37 5. Mejorar estilos y disposición visual con "style.css". "Mejorar": Unknown word.
38
39 ...
40 ...
41 ## 📄 Descripción Detallada "Descripción": Unknown word.
42
43 ## 'index.html'
44 plantilla principal que incluye estructura semántica (header, main, footer) y un contenedor donde "app.js" inyecta productos. "Plantilla": Unknown word.
45 incluye referencia a "style.css" y "app.js". "Incluir": Unknown word.
46
47 ## 'app.js'
48 Funciones clave esperadas: "funciones": Unknown word.
49 - Crear API para obtener datos (pedirlos a través de fetch) o XMLHttpRequest. "Cargar": Unknown word.
50 - Parsear y transformar datos (map/filter) para preparar la representación. "Parsear": Unknown word.
51 - Renderizar elementos (cards, listas) en el DOM. "Renderizar": Unknown word.
52 - Agregar listeners para interactividad (p. ej. filtros, botones "añadir al carrito").
53 Posibles consideraciones: manejo de errores en fetch, estados de carga (skeleton/loading), y separación de responsabilidades (render vs data).
54
55 ## 'products.json'
56 - Array de objetos con campos típicos: "id", "title", "description", "price", "image", "category".
57
58
59 ...
60 ...
61 ...
62 ...
63 ...
64 ...
65 ...
66 ...
67 ...
68 ...
69 ...
70 ...
71 ...
72 ...
73 ...
74 ...
75 ...
76 ...
77 ...
78 ...
79 ...
80 ...
81 ...
82 ...
83 ...
84 ...
85 ...
86 ...
87 ...
88 ...
89 ...
90 ...
91 ...
92 ...
93 ...
94 ...
95 ...
96 ...
97 ...
98 ...
99 ...
100 ...
101 ...
102 ...
103 ...
104 ...
105 ...
106 ...
107 ...
108 ...
109 ...
110 ...
111 ...
112 ...
113 ...
114 ...
115 ...
116 ...
117 ...
118 ...
119 ...
120 ...
121 ...
122 ...
123 ...
124 ...
125 ...
126 ...
127 ...
128 ...
129 ...
130 ...
131 ...
132 ...
133 ...
134 ...
135 ...
136 ...
137 ...
138 ...
139 ...
140 ...
141 ...
142 ...
143 ...
144 ...
145 ...
146 ...
147 ...
148 ...
149 ...
150 ...
151 ...
152 ...
153 ...
154 ...
155 ...
156 ...
157 ...
158 ...
159 ...
160 ...
161 ...
162 ...
163 ...
164 ...
165 ...
166 ...
167 ...
168 ...
169 ...
170 ...
171 ...
172 ...
173 ...
174 ...
175 ...
176 ...
177 ...
178 ...
179 ...
180 ...
181 ...
182 ...
183 ...
184 ...
185 ...
186 ...
187 ...
188 ...
189 ...
190 ...
191 ...
192 ...
193 ...
194 ...
195 ...
196 ...
197 ...
198 ...
199 ...
200 ...
201 ...
202 ...
203 ...
204 ...
205 ...
206 ...
207 ...
208 ...
209 ...
210 ...
211 ...
212 ...
213 ...
214 ...
215 ...
216 ...
217 ...
218 ...
219 ...
219 ...
220 ...
221 ...
222 ...
223 ...
224 ...
225 ...
226 ...
227 ...
228 ...
229 ...
229 ...
230 ...
231 ...
232 ...
233 ...
234 ...
235 ...
236 ...
237 ...
238 ...
239 ...
239 ...
240 ...
241 ...
242 ...
243 ...
244 ...
245 ...
246 ...
247 ...
248 ...
249 ...
249 ...
250 ...
251 ...
252 ...
253 ...
254 ...
255 ...
256 ...
257 ...
258 ...
259 ...
259 ...
260 ...
261 ...
262 ...
263 ...
264 ...
265 ...
266 ...
267 ...
268 ...
269 ...
269 ...
270 ...
271 ...
272 ...
273 ...
274 ...
275 ...
276 ...
277 ...
278 ...
279 ...
279 ...
280 ...
281 ...
282 ...
283 ...
284 ...
285 ...
286 ...
287 ...
288 ...
289 ...
289 ...
290 ...
291 ...
292 ...
293 ...
294 ...
295 ...
296 ...
297 ...
298 ...
299 ...
299 ...
300 ...
301 ...
302 ...
303 ...
304 ...
305 ...
306 ...
307 ...
308 ...
309 ...
309 ...
310 ...
311 ...
312 ...
313 ...
314 ...
315 ...
316 ...
317 ...
318 ...
319 ...
319 ...
320 ...
321 ...
322 ...
323 ...
324 ...
325 ...
326 ...
327 ...
328 ...
329 ...
329 ...
330 ...
331 ...
332 ...
333 ...
334 ...
335 ...
336 ...
337 ...
338 ...
339 ...
339 ...
340 ...
341 ...
342 ...
343 ...
344 ...
345 ...
346 ...
347 ...
348 ...
349 ...
349 ...
350 ...
351 ...
352 ...
353 ...
354 ...
355 ...
356 ...
357 ...
358 ...
359 ...
359 ...
360 ...
361 ...
362 ...
363 ...
364 ...
365 ...
366 ...
367 ...
368 ...
369 ...
369 ...
370 ...
371 ...
372 ...
373 ...
374 ...
375 ...
376 ...
377 ...
378 ...
379 ...
379 ...
380 ...
381 ...
382 ...
383 ...
384 ...
385 ...
386 ...
387 ...
388 ...
389 ...
389 ...
390 ...
391 ...
392 ...
393 ...
394 ...
395 ...
396 ...
397 ...
398 ...
399 ...
399 ...
400 ...
401 ...
402 ...
403 ...
404 ...
405 ...
406 ...
407 ...
408 ...
409 ...
409 ...
410 ...
411 ...
412 ...
413 ...
414 ...
415 ...
416 ...
417 ...
418 ...
419 ...
419 ...
420 ...
421 ...
422 ...
423 ...
424 ...
425 ...
426 ...
427 ...
428 ...
429 ...
429 ...
430 ...
431 ...
432 ...
433 ...
434 ...
435 ...
436 ...
437 ...
438 ...
439 ...
439 ...
440 ...
441 ...
442 ...
443 ...
444 ...
445 ...
446 ...
447 ...
448 ...
449 ...
449 ...
450 ...
451 ...
452 ...
453 ...
454 ...
455 ...
456 ...
457 ...
458 ...
459 ...
459 ...
460 ...
461 ...
462 ...
463 ...
464 ...
465 ...
466 ...
467 ...
468 ...
469 ...
469 ...
470 ...
471 ...
472 ...
473 ...
474 ...
475 ...
476 ...
477 ...
478 ...
479 ...
479 ...
480 ...
481 ...
482 ...
483 ...
484 ...
485 ...
486 ...
487 ...
488 ...
489 ...
489 ...
490 ...
491 ...
492 ...
493 ...
494 ...
495 ...
496 ...
497 ...
498 ...
499 ...
499 ...
500 ...
501 ...
502 ...
503 ...
504 ...
505 ...
506 ...
507 ...
508 ...
509 ...
509 ...
510 ...
511 ...
512 ...
513 ...
514 ...
515 ...
516 ...
517 ...
518 ...
519 ...
519 ...
520 ...
521 ...
522 ...
523 ...
524 ...
525 ...
526 ...
527 ...
528 ...
529 ...
529 ...
530 ...
531 ...
532 ...
533 ...
534 ...
535 ...
536 ...
537 ...
538 ...
539 ...
539 ...
540 ...
541 ...
542 ...
543 ...
544 ...
545 ...
546 ...
547 ...
548 ...
549 ...
549 ...
550 ...
551 ...
552 ...
553 ...
554 ...
555 ...
556 ...
557 ...
558 ...
559 ...
559 ...
560 ...
561 ...
562 ...
563 ...
564 ...
565 ...
566 ...
567 ...
568 ...
569 ...
569 ...
570 ...
571 ...
572 ...
573 ...
574 ...
575 ...
576 ...
577 ...
578 ...
579 ...
579 ...
580 ...
581 ...
582 ...
583 ...
584 ...
585 ...
586 ...
587 ...
588 ...
589 ...
589 ...
590 ...
591 ...
592 ...
593 ...
594 ...
595 ...
596 ...
597 ...
598 ...
599 ...
599 ...
600 ...
601 ...
602 ...
603 ...
604 ...
605 ...
606 ...
607 ...
608 ...
609 ...
609 ...
610 ...
611 ...
612 ...
613 ...
614 ...
615 ...
616 ...
617 ...
618 ...
619 ...
619 ...
620 ...
621 ...
622 ...
623 ...
624 ...
625 ...
626 ...
627 ...
628 ...
629 ...
629 ...
630 ...
631 ...
632 ...
633 ...
634 ...
635 ...
636 ...
637 ...
638 ...
639 ...
639 ...
640 ...
641 ...
642 ...
643 ...
644 ...
645 ...
646 ...
647 ...
648 ...
649 ...
649 ...
650 ...
651 ...
652 ...
653 ...
654 ...
655 ...
656 ...
657 ...
658 ...
659 ...
659 ...
660 ...
661 ...
662 ...
663 ...
664 ...
665 ...
666 ...
667 ...
668 ...
669 ...
669 ...
670 ...
671 ...
672 ...
673 ...
674 ...
675 ...
676 ...
677 ...
678 ...
679 ...
679 ...
680 ...
681 ...
682 ...
683 ...
684 ...
685 ...
686 ...
687 ...
688 ...
689 ...
689 ...
690 ...
691 ...
692 ...
693 ...
694 ...
695 ...
696 ...
697 ...
698 ...
699 ...
699 ...
700 ...
701 ...
702 ...
703 ...
704 ...
705 ...
706 ...
707 ...
708 ...
709 ...
709 ...
710 ...
711 ...
712 ...
713 ...
714 ...
715 ...
716 ...
717 ...
718 ...
719 ...
719 ...
720 ...
721 ...
722 ...
723 ...
724 ...
725 ...
726 ...
727 ...
728 ...
729 ...
729 ...
730 ...
731 ...
732 ...
733 ...
734 ...
735 ...
736 ...
737 ...
738 ...
739 ...
739 ...
740 ...
741 ...
742 ...
743 ...
744 ...
745 ...
746 ...
747 ...
748 ...
749 ...
749 ...
750 ...
751 ...
752 ...
753 ...
754 ...
755 ...
756 ...
757 ...
758 ...
759 ...
759 ...
760 ...
761 ...
762 ...
763 ...
764 ...
765 ...
766 ...
767 ...
768 ...
769 ...
769 ...
770 ...
771 ...
772 ...
773 ...
774 ...
775 ...
776 ...
777 ...
778 ...
778 ...
779 ...
779 ...
780 ...
781 ...
782 ...
783 ...
784 ...
785 ...
786 ...
787 ...
788 ...
789 ...
789 ...
790 ...
791 ...
792 ...
793 ...
794 ...
795 ...
796 ...
797 ...
798 ...
799 ...
799 ...
800 ...
801 ...
802 ...
803 ...
804 ...
805 ...
806 ...
807 ...
808 ...
809 ...
809 ...
810 ...
811 ...
812 ...
813 ...
814 ...
815 ...
816 ...
817 ...
818 ...
819 ...
819 ...
820 ...
821 ...
822 ...
823 ...
824 ...
825 ...
826 ...
827 ...
828 ...
829 ...
829 ...
830 ...
831 ...
832 ...
833 ...
834 ...
835 ...
836 ...
837 ...
838 ...
839 ...
839 ...
840 ...
841 ...
842 ...
843 ...
844 ...
845 ...
846 ...
847 ...
848 ...
849 ...
849 ...
850 ...
851 ...
852 ...
853 ...
854 ...
855 ...
856 ...
857 ...
858 ...
859 ...
859 ...
860 ...
861 ...
862 ...
863 ...
864 ...
865 ...
866 ...
867 ...
868 ...
869 ...
869 ...
870 ...
871 ...
872 ...
873 ...
874 ...
875 ...
876 ...
877 ...
878 ...
878 ...
879 ...
879 ...
880 ...
881 ...
882 ...
883 ...
884 ...
885 ...
886 ...
887 ...
888 ...
889 ...
889 ...
890 ...
891 ...
892 ...
893 ...
894 ...
895 ...
896 ...
897 ...
898 ...
899 ...
899 ...
900 ...
901 ...
902 ...
903 ...
904 ...
905 ...
906 ...
907 ...
908 ...
909 ...
909 ...
910 ...
911 ...
912 ...
913 ...
914 ...
915 ...
916 ...
917 ...
918 ...
919 ...
919 ...
920 ...
921 ...
922 ...
923 ...
924 ...
925 ...
926 ...
927 ...
928 ...
929 ...
929 ...
930 ...
931 ...
932 ...
933 ...
934 ...
935 ...
936 ...
937 ...
938 ...
939 ...
939 ...
940 ...
941 ...
942 ...
943 ...
944 ...
945 ...
946 ...
947 ...
948 ...
949 ...
949 ...
950 ...
951 ...
952 ...
953 ...
954 ...
955 ...
956 ...
957 ...
958 ...
959 ...
959 ...
960 ...
961 ...
962 ...
963 ...
964 ...
965 ...
966 ...
967 ...
968 ...
969 ...
969 ...
970 ...
971 ...
972 ...
973 ...
974 ...
975 ...
976 ...
977 ...
978 ...
978 ...
979 ...
979 ...
980 ...
981 ...
982 ...
983 ...
984 ...
985 ...
986 ...
987 ...
988 ...
988 ...
989 ...
989 ...
990 ...
991 ...
992 ...
993 ...
994 ...
995 ...
995 ...
996 ...
996 ...
997 ...
997 ...
998 ...
998 ...
999 ...
999 ...
1000 ...
1000 ...
1001 ...
1001 ...
1002 ...
1002 ...
1003 ...
1003 ...
1004 ...
1004 ...
1005 ...
1005 ...
1006 ...
1006 ...
1007 ...
1007 ...
1008 ...
1008 ...
1009 ...
1009 ...
1010 ...
1010 ...
1011 ...
1011 ...
1012 ...
1012 ...
1013 ...
1013 ...
1014 ...
1014 ...
1015 ...
1015 ...
1016 ...
1016 ...
1017 ...
1017 ...
1018 ...
1018 ...
1019 ...
1019 ...
1020 ...
1020 ...
1021 ...
1021 ...
1022 ...
1022 ...
1023 ...
1023 ...
1024 ...
1024 ...
1025 ...
1025 ...
1026 ...
1026 ...
1027 ...
1027 ...
1028 ...
1028 ...
1029 ...
1029 ...
1030 ...
1030 ...
1031 ...
1031 ...
1032 ...
1032 ...
1033 ...
1033 ...
1034 ...
1034 ...
1035 ...
1035 ...
1036 ...
1036 ...
1037 ...
1037 ...
1038 ...
1038 ...
1039 ...
1039 ...
1040 ...
1040 ...
1041 ...
1041 ...
1042 ...
1042 ...
1043 ...
1043 ...
1044 ...
1044 ...
1045 ...
1045 ...
1046 ...
1046 ...
1047 ...
1047 ...
1048 ...
1048 ...
1049 ...
1049 ...
1050 ...
1050 ...
1051 ...
1051 ...
1052 ...
1052 ...
1053 ...
1053 ...
1054 ...
1054 ...
1055 ...
1055 ...
1056 ...
1056 ...
1057 ...
1057 ...
1058 ...
1058 ...
1059 ...
1059 ...
1060 ...
1060 ...
1061 ...
1061 ...
1062 ...
1062 ...
1063 ...
1063 ...
1064 ...
1064 ...
1065 ...
1065 ...
1066 ...
1066 ...
1067 ...
1067 ...
1068 ...
1068 ...
1069 ...
1069 ...
1070 ...
1070 ...
1071 ...
1071 ...
1072 ...
1072 ...
1073 ...
1073 ...
1074 ...
1074 ...
1075 ...
1075 ...
1076 ...
1076 ...
1077 ...
1077 ...
1078 ...
1078 ...
1079 ...
1079 ...
1080 ...
1080 ...
1081 ...
1081 ...
1082 ...
1082 ...
1083 ...
1083 ...
1084 ...
1084 ...
1085 ...
1085 ...
1086 ...
1086 ...
1087 ...
1087 ...
1088 ...
1088 ...
1089 ...
1089 ...
1090 ...
1090 ...
1091 ...
1091 ...
1092 ...
1092 ...
1093 ...
1093 ...
1094 ...
1094 ...
1095 ...
1095 ...
1096 ...
1096 ...
1097 ...
1097 ...
1098 ...
1098 ...
1099 ...
1099 ...
1100 ...
1100 ...
1101 ...
1101 ...
1102 ...
1102 ...
1103 ...
1103 ...
1104 ...
1104 ...
1105 ...
1105 ...
1106 ...
1106 ...
1107 ...
1107 ...
1108 ...
1108 ...
1109 ...
1109 ...
1110 ...
1110 ...
1111 ...
1111 ...
1112 ...
1112 ...
1113 ...
1113 ...
1114 ...
1114 ...
1115 ...
1115 ...
1116 ...
1116 ...
1117 ...
1117 ...
1118 ...
1118 ...
1119 ...
1119 ...
1120 ...
1120 ...
1121 ...
1121 ...
1122 ...
1122 ...
1123 ...
1123 ...
1124 ...
1124 ...
1125 ...
1125 ...
1126 ...
1126 ...
1127 ...
1127 ...
1128 ...
1128 ...
1129 ...
1129 ...
1130 ...
1130 ...
1131 ...
1131 ...
1132 ...
1132 ...
1133 ...
1133 ...
1134 ...
1134 ...
1135 ...
1135 ...
1136 ...
1136 ...
1137 ...
1137 ...
1138 ...
1138 ...
1139 ...
1139 ...
1140 ...
1140 ...
1141 ...
1141 ...
1142 ...
1142 ...
1143 ...
1143 ...
1144 ...
1144 ...
1145 ...
1145 ...
1146 ...
1146 ...
1147 ...
1147 ...
1148 ...
1148 ...
1149 ...
1149 ...
1150 ...
1150 ...
1151 ...
1151 ...
1152 ...
1152 ...
1153 ...
1153 ...
1154 ...
1154 ...
1155 ...
1155 ...
1156 ...
1156 ...
1157 ...
1157 ...
1158 ...
1158 ...
1159 ...
1159 ...
1160 ...
1160 ...
1161 ...
1161 ...
1162 ...
1162 ...
1163 ...
1163 ...
1164 ...
1164 ...
1165 ...
1165 ...
1166 ...
1166 ...
1167 ...
1167 ...
1168 ...
1168 ...
1169 ...
1169 ...
1170 ...
1170 ...
1171 ...
1171 ...
1172 ...
1172 ...
1173 ...
1173 ...
1174 ...
1174 ...
1175 ...
1175 ...
1176 ...
1176 ...
1177 ...
1177 ...
1178 ...
1178 ...
1179 ...
1179 ...
1180 ...
1180 ...
1181 ...
1181 ...
1182 ...
1182 ...
1183 ...
1183 ...
1184 ...
1184 ...
1185 ...
1185 ...
1186 ...
1186 ...
1187 ...
1187 ...
1188 ...
1188 ...
1189 ...
1189 ...
1190 ...
1190 ...
1191 ...
1191 ...
1192 ...
1192 ...
1193 ...
1193 ...
1194 ...
1194 ...
1195 ...
1195 ...
1196 ...
1196 ...
1197 ...
1197 ...
1198 ...
1198 ...
1199 ...
1199 ...
1200 ...
1200 ...
1201 ...
1201 ...
1202 ...
1202 ...
1203 ...
1203 ...
1204 ...
1204 ...
1205 ...
1205 ...
1206 ...
1206 ...
1207 ...
1207 ...
1208 ...
1208 ...
1209 ...
1209 ...
1210 ...
1210 ...
1211 ...
1211 ...
1212 ...
1212 ...
1213 ...
1213 ...
1214 ...
1214 ...
1215 ...
1215 ...
1216 ...
1216 ...
1217 ...
1217 ...
1218 ...
1218 ...
1219 ...
1219 ...
1220 ...
1220 ...
1221 ...
1221 ...
1222 ...
1222 ...
1223 ...
1223 ...
1224 ...
1224 ...
1225 ...
1225 ...
1226 ...
1226 ...
1227 ...
1227 ...
1228 ...
1228 ...
1229 ...
1229 ...
1230 ...
1230 ...
1231 ...
1231 ...
1232 ...
1232 ...
1233 ...
1233 ...
1234 ...
1234 ...
1235 ...
1235 ...
1236 ...
1236 ...
1237 ...
1237 ...
1238 ...
1238 ...
1239 ...
1239 ...
1240 ...
1240 ...
1241 ...
1241 ...
1242 ...
1242 ...
1243 ...
1243 ...
1244 ...
1244 ...
1245 ...
1245 ...
1246 ...
1246 ...
1247 ...
1247 ...
1248 ...
1248 ...
1249 ...
1249 ...
1250 ...
1250 ...
1251 ...
1251 ...
1252 ...
1252 ...
1253 ...
1253 ...
1254 ...
1254 ...
1255 ...
1255 ...
1256 ...
1256 ...
1257 ...
1257 ...
1258 ...
1258 ...
1259 ...
1259 ...
1260 ...
1260 ...
1261 ...
1261 ...
1262 ...
1262 ...
1263 ...
1263 ...
1264 ...
1264 ...
1265 ...
1265 ...
1266 ...
1266 ...
1267 ...
1267 ...
1268 ...
1268 ...
1269 ...
1269 ...
1270 ...
1270 ...
1271 ...
1271 ...
1272 ...
1272 ...
1273 ...
1273 ...
1274 ...
1274 ...
1275 ...
1275 ...
1276 ...
1276 ...
1277 ...
1277 ...
1278 ...
1278 ...
1279 ...
1279 ...
1280 ...
1280 ...
1281 ...
1281 ...
1282 ...
1282 ...
1283 ...
1283 ...
1284 ...
1284 ...
1285 ...
1285 ...
1286 ...
1286 ...
1287 ...
1287 ...
1288 ...
1288 ...
1289 ...
1289 ...
1290 ...
1290 ...
1291 ...
1291 ...
1292 ...
1292 ...
1293 ...
1293 ...
1294 ...
1294 ...
1295 ...
1295 ...
1296 ...
1296 ...
1297 ...
1297 ...
1298 ...
1298 ...
1299 ...
1299 ...
1300 ...
1300 ...
1301 ...
1301 ...
1302 ...
1302 ...
1303 ...
1303 ...
1304 ...
1304 ...
1305 ...
1305 ...
1306 ...
1306 ...
1307 ...
1307 ...
1308 ...
1308 ...
1309 ...
1309 ...
1310 ...
1310 ...
1311 ...
1311 ...
1312 ...
1312 ...
1313 ...
1313 ...
1314 ...
1314 ...
1315 ...
1315 ...
1316 ...
1316 ...
1317 ...
1317 ...
1318 ...
1318 ...
1319 ...
1319 ...
1320 ...
1320 ...
1321 ...
1321 ...
1322 ...
1322 ...
1323 ...
1323 ...
1324 ...
1324 ...
1325 ...
1325 ...
1326 ...
1326 ...
1327 ...
1327 ...
1328 ...
1328 ...
1329 ...
1329 ...
1330 ...
1330 ...
1331 ...
1331 ...
1332 ...
1332 ...
1333 ...
1333 ...
1334 ...
1334 ...
1335 ...
1335 ...
1336 ...
1336 ...
1337 ...
1337 ...
1338 ...
1338 ...
1339 ...
1339 ...
1340 ...
1340 ...
1341 ...
1341 ...
1342 ...
1342 ...
1343 ...
1343 ...
1344 ...
1344 ...
1345 ...
1345 ...
1346 ...
1346 ...
1347 ...
1347 ...
1348 ...
1348 ...
1349 ...
1349 ...
1350 ...
1350 ...
1351 ...
1351 ...
1352 ...
1352 ...
1353 ...
1353 ...
1354 ...
1354 ...
1355 ...
1355 ...
1356 ...
1356 ...
1357 ...
1357 ...
1358 ...
1358 ...
1359 ...
1359 ...
1360 ...
1360 ...
1361 ...
1361 ...
1362 ...
1362 ...
1363 ...
1363 ...
1364 ...
1364 ...
1365 ...
1365 ...
1366 ...
1366 ...
1367 ...
1367 ...
1368 ...
1368 ...
1369 ...
1369 ...
1370 ...
1370 ...
1371 ...
1371 ...
1372 ...
1372 ...
1373 ...
1373 ...
1374 ...
1374 ...
1375 ...
1375 ...
1376 ...
1376 ...
1377 ...
1377 ...
1378 ...
1378 ...
1379 ...
1379 ...
1380 ...
1380 ...
1381 ...
1381 ...
1382 ...
1382 ...
1383 ...
1383 ...
1384 ...
1384 ...
1385 ...
1385 ...
1386 ...
1386 ...
1387 ...
1387 ...
1388 ...
1388 ...
1389 ...
1389 ...
1390 ...
1390 ...
1391 ...
1391 ...
1392 ...
1392 ...
1393 ...
1393 ...
1394 ...
1394 ...
1395 ...
1395 ...
1396 ...
1396 ...
1397 ...
1397 ...
1398 ...
1398 ...
1399 ...
1399 ...
1400 ...
1400 ...
1401 ...
1401 ...
1402 ...
1402 ...
1403 ...
1403 ...
1404 ...
1404 ...
1405 ...
1405 ...
1406 ...
1406 ...
1407 ...
1407 ...
1408 ...
1408 ...
1409 ...
1409 ...
1410 ...
1410 ...
1411 ...
1411 ...
1412 ...
1412 ...
1413 ...
1413 ...
1414 ...
1414 ...
1415 ...
1415 ...
1416 ...
1416 ...
1417 ...
1417 ...
1418 ...
1418 ...
1419 ...
1419 ...
1420 ...
1420 ...
1421 ...
1421 ...
1422 ...
1422 ...
1423 ...
1423 ...
1424 ...
1424 ...
1425 ...
1425 ...
1426 ...
1426 ...
1427 ...
1427 ...
1428 ...
1428 ...
1429 ...
1429 ...
1430 ...
1430 ...
1431 ...
1431 ...
1432 ...
1432 ...
1433 ...
1433 ...
1434 ...
1434 ...
1435 ...
1435 ...
1436 ...
1436 ...
1437 ...
1437 ...
1438 ...
1438 ...
1439 ...
1439 ...
1440 ...
1440 ...
1441 ...
1441 ...
1442 ...
1442 ...
1443 ...
1443 ...
1444 ...
1444 ...
1445 ...
1445 ...
1446 ...
1446 ...
1447 ...
1447 ...
1448 ...
1448 ...
1449 ...
1449 ...
1450 ...
1450 ...
1451 ...
1451 ...
1452 ...
1452 ...
1453 ...
1453 ...
1454 ...
1454 ...
1455 ...
1455 ...
1456 ...
1456 ...
1457 ...
1457 ...
1458 ...
1458 ...
1459 ...
1459 ...
1460 ...
1460 ...
1461 ...
1461 ...
1462 ...
1462 ...
1463 ...
1463 ...
1464 ...
1464 ...
1465 ...
1465 ...
1466 ...
1466 ...
1467 ...
1467 ...
1468 ...
1468 ...
1469 ...
1469 ...
1470 ...
1470 ...
1471 ...
1471 ...
1472 ...
1472 ...
1473 ...
1473 ...
1474 ...
1474 ...
1475 ...
1475 ...
1476 ...
1476 ...
1477 ...
1477 ...
1478 ...
1478 ...
1479 ...
1479 ...
1480 ...
1480 ...
1481 ...
1481 ...
1482 ...
1482 ...
1483 ...
1483 ...
1484 ...
1484 ...
1485 ...
1485 ...
1486 ...
1486 ...
1487 ...
1487 ...
1488 ...
1488 ...
1489 ...
1489 ...
1490 ...
1490 ...
1491 ...
1491 ...
1492 ...
1492 ...
1493 ...
1493 ...
1494 ...
1494 ...
1495 ...
1495 ...
1496 ...
1496 ...
1497 ...
1497 ...
1498 ...
1498 ...
1499 ...
1499 ...
1500 ...
1500 ...
1501 ...
1501 ...
1502 ...
1502 ...
1503 ...
1503 ...
1504 ...
1504 ...
1505 ...
1505 ...
1506 ...
1506 ...
1507 ...
1507 ...
1508 ...
1508 ...
1509 ...
1509 ...
1510 ...
1510 ...
1511 ...
1511 ...
1512 ...
1512 ...
1513 ...
1513 ...
1514 ...
1514 ...
1515 ...
1515 ...
1516 ...
1516 ...
1517 ...
1517 ...
1518 ...
1518 ...
1519 ...
1519 ...
1520 ...
1520 ...
1521 ...
1521 ...
1522 ...
1522 ...
1523 ...
1523 ...
1524 ...
1524 ...
1525 ...
1525 ...
1526 ...
1526 ...
1527 ...
1527 ...
1528 ...
1528 ...
1529 ...
1529 ...
1530 ...
1530 ...
1531 ...
1531 ...
1532 ...
1532 ...
1533 ...
1533 ...
1534 ...
1534 ...
1535 ...
1535 ...
1536 ...
1536 ...
1537 ...
1537 ...
1538 ...
1538 ...
1539 ...
1539 ...
1540 ...
1540 ...
1541 ...
1541 ...
1542 ...
1542 ...
1543 ...
1543 ...
1544 ...
1544 ...
1545 ...
1545 ...
1546 ...
1546 ...
1547 ...
1547 ...
1548 ...
1548 ...
1549 ...
1549 ...
1550 ...
1550 ...
1551 ...
1551 ...
1552 ...
1552 ...
1553 ...
1553 ...
1554 ...
1554 ...
1555 ...
1555 ...
1556 ...
1556 ...
1557 ...
1557 ...
1558 ...
1558 ...
1559 ...
1559 ...
1560 ...
1560 ...
1561 ...
1561 ...
1562 ...
1562 ...
1563 ...
1563 ...
1564 ...
1564 ...
1565 ...
1565 ...
1566 ...
1566 ...
1567 ...
1567 ...
1568 ...
1568 ...
1569 ...
1569 ...
1570 ...
1570 ...
1571 ...
1571 ...
1572 ...
1572 ...
1573 ...
1573 ...
1574 ...
1574 ...
1575 ...
1575 ...
1576 ...
1576 ...
1577 ...
1577 ...
1578 ...
1578 ...
1579 ...
1579 ...
1580 ...
1580 ...
1581 ...
1581 ...
1582 ...
1582 ...
1583 ...
1583 ...
1584 ...
1584 ...
1585 ...
1585 ...
1586 ...
1586 ...
1587 ...
1587 ...
1588 ...
1588 ...
1589 ...
1589 ...
1590 ...
1590 ...
1591 ...
1591 ...
1592 ...
1592 ...
1593 ...
1593 ...
1594 ...
1594 ...
1595 ...
1595 ...
1596 ...
1596 ...
1597 ...
1597 ...
1598 ...
1598 ...
1599 ...
1599 ...
1600 ...
1600 ...
1601 ...
1601 ...
1602 ...
1602 ...
1603 ...
1603 ...
1604 ...
1604 ...
1605 ...
1605 ...
1606 ...
1606 ...
1607 ...
1
```

## Análisis - Semana 12: Estructura de Proyecto Web (Archivos iniciales)

### Descripción General

La carpeta Semana12 contiene una plantilla/estructura inicial para proyectos web: HTML base, recursos estáticos (CSS, imágenes), scripts JavaScript y configuración de editor. Es útil como punto de partida para prácticas y despliegues sencillos.

---

### Lenguaje

- **HTML5** — estructura del proyecto
  - **CSS3** — estilos (carpeta css/)
  - **JavaScript (ES6+)** — lógica de cliente en js/
  - **JSON / configuración** — opcional en .vscode (settings)
- 

### Estructura de Archivos (observada)

Semana12/

```
└── archivos-iniciales/
 ├── index.html # Plantilla HTML base
 ├── .vscode/ # Configuración de espacio de trabajo (opcional)
 └── css/ # Estilos (main.css, reset, etc.)
 └── img/ # Imágenes y assets
 └── js/ # Scripts de interacción
```

---

### Objetivos

1. Proveer una plantilla mínima para comenzar proyectos web.
2. Enseñar buenas prácticas de organización (separar css, js, img).
3. Facilitar pruebas locales y despliegue estático (servir index.html).
4. Permitir personalización rápida para ejercicios y demos.

---

## Descripción Detallada

- index.html: Documento HTML5 básico con meta charset, viewport y enlaces a css/main.css y js/main.js.
- css/: Hoja(s) de estilo para layout y componentes.
- img/: Recursos gráficos organizados por tema o componente.
- js/: Código de interacción (event listeners, manipulación DOM, pequeñas utilidades).
- .vscode/ (si existe): Configs útiles para desarrollar (formatters, liveServer settings, snippets).

---

## Breve Análisis Técnico

- Estructura clásica y apropiada para proyectos estáticos y prácticas didácticas.
- Recomendación para desarrollo local: usar un servidor estático (por ejemplo python -m http.server o npx http-server) para evitar problemas con fetch y rutas relativas.
- Buenas prácticas: usar index.html como punto de entrada, agrupar assets y mantener código modular (js/main.js, js/modules/).
- Considerar añadir un archivo README.md con instrucciones de arranque y dependencias (si las hubiera) y .gitignore para controlar versiones.

---

## Puntos de Aprendizaje

- Separación de responsabilidades (html/css/js) facilita colaboración y pruebas.
- Empezar desde una plantilla reduce fricción y acelera experimentación.

- La configuración del editor en .vscode mejora consistencia del equipo (line endings, formato).
- 

## Conclusión

Semana12 proporciona una plantilla sólida para comenzar proyectos web prácticos. Para avanzar, se recomienda añadir ejemplos concretos (componentes, pequeños scripts de interacción), documentación (README.md) y scripts de desarrollo (package.json) si se incorporarán herramientas de build.

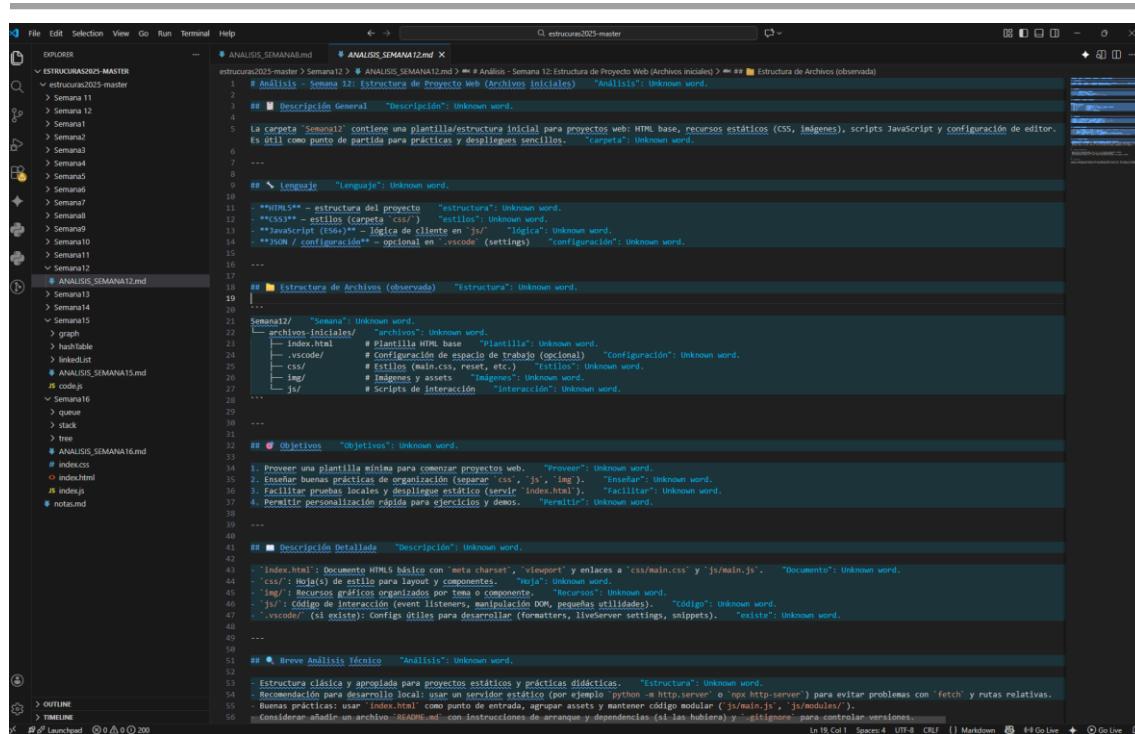
# Análisis - Semana 13: Recursión y Algoritmos de Ordenamiento

## Descripción General

La carpeta Semana13 se enfoca en dos temas clave de algoritmos: **recursión** y **ordenamiento**. Contiene ejemplos y utilidades para entender la pila de llamadas, distintos estilos de implementación (iterativa vs recursiva) y algoritmos de ordenamiento clásicos (mergesort, quicksort, selection sort).

## Lenguaje

- **JavaScript (ES6+)** — implementaciones de funciones recursivas y algoritmos de ordenamiento
- **HTML5** — interfaz de demostración (index.html)
- **Markdown** — documentación (README.md, database.md, processes.md, design.md)



ANALISIS\_SEMANA12.md

```
1 # Análisis - Semana 12: Estructura de Proyecto Web (Archivos iniciales) > ## Estructura de Archivos (observada)
2
3 ## ■ Descripción General "Descripción": Unknown word.
4
5 La carpeta "Semana12" contiene una plantilla/estructura inicial para proyectos web: HTML base, recursos estáticos (CSS, imágenes), scripts JavaScript y configuración de editor.
6 Es útil como punto de partida para prácticas y despliegues sencillos. "Carpeta": Unknown word.
7 ...
8
9 ## ■ Lenguaje "Lenguaje": Unknown word.
10
11 - **HTML** - estructura del proyecto "estructura": Unknown word.
12 - **CSS** - estilos (carpeta "css") "estilos": Unknown word.
13 - **JavaScript (ts)** - lógica de cliente en "js/" "lógica": Unknown word.
14 - **JSON / configuración** - opcional en ".vscode" (settings) "configuración": Unknown word.
15
16 ...
17
18 ## ■ Estructura de Archivos (observada) "estructura": Unknown word.
19
20 ...
21 Semana12/ "Semana": Unknown word.
22 archivos-iniciales/ "archivos": Unknown word.
23 index.html # Plantilla HTML base
24 configuracion/ # Configuración de espacio de trabajo (opcional)
25 .vscode/ # Estilos (main.css, reset, etc.) "files": Unknown word.
26 img/ # Imágenes y assets "Imágenes": Unknown word.
27 js/ # Scripts de Interacción "Interacción": Unknown word.
28 ...
29
30 ...
31
32 ## ■ Objetivos "Objetivos": Unknown word.
33
34 1. Proveer una plantilla mínima para comenzar proyectos web. "Proveer": Unknown word.
35 2. Enseñar buenas prácticas de organización (carpeta "css", "js", "img"). "Enseñar": Unknown word.
36 3. Facilitar pruebas locales y despliegue estático (servir "index.html"). "Facilitar": Unknown word.
37 4. Permitir personalización rápida para ejercicios y demos. "Permitir": Unknown word.
38
39 ...
40
41 ## ■ Descripción Detallada "Descripción": Unknown word.
42
43 - "index.html": Documento HTML básico con "meta charset", "viewport" y enlaces a "css/main.css" y "js/main.js". "Documento": Unknown word.
44 - "css/main.css" de los componentes "Componentes": Unknown word.
45 - "img": Recursos gráficos organizados por componentes. "Recursos": Unknown word.
46 - "js/": Código de Interacción (event listeners, manipulación DOM, pequeñas utilidades). "Código": Unknown word.
47 - ".vscode/": Config útiles para desarrollar (formatters, liveServer settings, snippets). "existe": Unknown word.
48
49 ...
50
51 ## ■ Breve Análisis Técnico "Análisis": Unknown word.
52
53 - Estructura clásica y apropiada para proyectos estáticos y prácticas didácticas. "Estructura": Unknown word.
54 - Recomendación para desarrollo local: usar un servidor estático (por ejemplo python -m http.server o npx http-server) para evitar problemas con 'Fetch' y rutas relativas.
55 - Buenas prácticas: usar "index.html" como punto de entrada, agrupar assets y mantener módulos modulares ("js/main.js", "js/modules/").
56 - Consideraciones para el archivo README.md: usar una estructura jerárquica y organizada (el archivo hubiera "x": "y" en su lugar).
```

```

File Edit Sélection View Go Run Terminal Help
File Explorer ANALISIS_SEMANA12.md ANALISIS_SEMANA12.md
estruca2025-master > Semana12 > ANALISIS_SEMANA12.md > # Análisis - Semana 12: Estructura de Proyecto Web (Archivos iniciales) > # Estructura de Archivos (observada)
1 # Análisis - Semana 12: Estructura de Proyecto Web (Archivos iniciales) "Análisis": Unknown word.
2 ## Descripción Detallada "Descripción": Unknown word.
3 ...
4 ...
5 ...
6 ...
7 ...
8 ...
9 ...
10 ...
11 ...
12 ...
13 ...
14 ...
15 ...
16 ...
17 ...
18 ...
19 ...
20 ...
21 ...
22 ...
23 ...
24 ...
25 ...
26 ...
27 ...
28 ...
29 ...
30 ...
31 ...
32 ...
33 ...
34 ...
35 ...
36 ...
37 ...
38 ...
39 ...
40 ...
41 ...
42 ...
43 ...
44 ...
45 ...
46 ...
47 ...
48 ...
49 ...
50 ...
51 ...
52 ...
53 ...
54 ...
55 ...
56 ...
57 ...
58 ...
59 ...
60 ...
61 ...
62 ...
63 ...
64 ...
65 ...
66 ...
67 ...
68 ...
69 ...
70 ...
71 ...

```

ANALISIS\_SEMANA12.md

ANALISIS\_SEMANA15.md

ANALISIS\_SEMANA16.md

code.js

code16

queue

stack

tree

graph

hashTable

linkedList

notes.md

outline

timeline

Launchpad

1 In 19, Col 1 Spaces: 4 - UTF-8 - CR/LF | Markdown | Go Live | Go Live

## Estructura de Archivos

Semana13/

- |── index.html # Página de demostración / ejercicios
- |── main.js # Código de interacción o ejemplos globales
- |── README.md # Notas generales de la semana
- |── database.md # Notas relacionadas (si aplica)
- |── processes.md # Descripción de procesos/algoritmos
- |── design.md # Diseño de soluciones o diagramas
- |── Recursion/
  - | ├── callStackExample.js
  - | ├── countToZero.js
  - | ├── fibonacciliterative.js
  - | ├── fibonacciRecursive.js
  - | └── fibonacciRecursiveBetter.js
- └── Sorting/ # Implementaciones de ordenamiento

```
└── mergesort.js
└── quicksort.js
└── selectionSort.js
```

---

## Objetivos

1. Comprender la recursión: cómo funciona la pila de llamadas y cuándo usarla.
  2. Comparar implementaciones recursivas e iterativas (ej.: Fibonacci recursivo vs iterativo).
  3. Implementar y analizar algoritmos de ordenamiento clásicos: mergesort, quicksort y selection sort.
  4. Calcular y comparar complejidades temporales y espaciales de cada algoritmo.
  5. Practicar la transformación de una solución recursiva a una iterativa y viceversa.
- 

## Descripción Detallada

### Carpeta Recursion/

- Ejemplos que ilustran conceptos:
  - callStackExample.js: visualiza la pila de llamadas y la entrada/salida de funciones.
  - countToZero.js: ejemplo simple de recursión con condición base clara.
  - fibonacciRecursive.js vs fibonacciIterative.js: comparación directa de rendimiento y complejidad.
  - fibonacciRecursiveBetter.js: optimizaciones como memoización para mejorar recursión.

### Carpeta Sorting/

- mergesort.js: algoritmo divide y vencerás con complejidad  $O(n \log n)$  y espacio adicional  $O(n)$ .
  - quicksort.js: algoritmo promedio  $O(n \log n)$ , peor caso  $O(n^2)$ ; discusión sobre pivot y particionamiento.
  - selectionSort.js: algoritmo sencillo con complejidad  $O(n^2)$ , útil para entender principios básicos.
- 

## Breve Análisis Técnico

### Recursión

- Conceptos clave:
  - **Caso base**: condición que detiene la recursión.
  - **Llamada recursiva**: descomposición del problema en subproblemas.
  - **Pila de llamadas**: cada llamada ocupa espacio en la pila; riesgo de stack overflow si la profundidad es grande.
- Optimización: **memoización** para evitar recalcular subproblemas (ej. Fibonacci), y **tail recursion** (cuando el motor JS lo optimiza).
- Trade-offs: la recursión suele ser más legible, pero puede usar más memoria; la versión iterativa suele ser más eficiente en espacio.

### Ordenamiento

- Complejidades y características:
  - **Mergesort**:  $O(n \log n)$  tiempo,  $O(n)$  espacio; estable; buen rendimiento en datos grandes.
  - **Quicksort**:  $O(n \log n)$  promedio,  $O(n^2)$  peor caso; in-place posible; elegir buen pivot (randomizado o median-of-three) reduce probabilidad de peor caso.
  - **Selection Sort**:  $O(n^2)$  tiempo,  $O(1)$  espacio; sencillo pero ineficiente para grandes  $n$ ; útil en contextos con memoria limitada y tamaños pequeños.

- Consideraciones prácticas: para datos parcialmente ordenados o datos grandes en memoria limitada, elegir el algoritmo adecuado según caso.

### Recomendaciones de práctica

- Añadir pruebas de rendimiento simples (timings) comparando implementaciones para distintos tamaños de entrada.
- Visualizar recursión y ordenamiento (pequeñas animaciones) ayuda a la comprensión.
- Implementar memoización en problemas recursivos con subproblemas repetidos.



### Puntos de Aprendizaje

- Entender la recursión conceptualmente es esencial para muchos algoritmos; la memoización convierte soluciones recursivas exponenciales en polinomiales.
- Los algoritmos de ordenamiento enseñan estrategias (divide & conquer, in-place, stable vs unstable) y son la base para algoritmos más avanzados.
- Comparar implementaciones y medir con casos de prueba reales afianza la elección de algoritmo.



### Conclusión

Semana 13 es una semana clave para interiorizar principios algorítmicos: recursión y ordenamiento. Practicar con ejemplos concretos (Fibonacci, mergesort, quicksort) y medir su comportamiento hará que los conceptos teóricos se vuelvan aplicables en problemas reales de programación y optimización.

The screenshot shows a terminal window with multiple tabs open. The current tab displays a file named 'ANALISIS\_SEMANA13.md' which contains a detailed analysis of recursion and sorting algorithms. The file includes sections on 'Descripción General', 'Estructura de Archivos', 'Semana13', 'Objetivos', and 'Descripción Detallada'. The code editor on the right shows the same content, with syntax highlighting for various programming constructs like 'JavaScript (ES6)', 'HTML', and 'CSS'. The file browser on the left lists other files and folders related to the course, such as 'ESTRUCTURAS2025-MASTER', 'ANALISIS\_SEMANA13', 'ANALISIS\_SEMANA14', 'ANALISIS\_SEMANA15', 'ANALISIS\_SEMANA16', and 'OUTLINE'.

```
File Edit Selection View Go Run Terminal Help
ESTRUCTURAS2025-MASTER
explorer
ESTRUCTURAS2025-MASTER
Semana 11
Semana 12
Semana1
Semana2
Semana3
Semana4
Semana5
Semana6
Semana7
Semana8
Semana9
Semana10
Semana11
Semana12
Semana13
> Recursion
> Sorting
ANALISIS_SEMANA13.md
database.md
design.md
index.html
main.js
OnComputer, Deployment, Sistema de Asis...
processes.md
README.md
Semana14
Semana15
graph
hashable
linkedlist
ANALISIS_SEMANA15.md
code.js
Semana16
queue
stack
tree
ANALISIS_SEMANA16.md
index.css
index.html
index.js
notes.md

ANALISIS_SEMANA13.md
estructuras2025-master > Semana13 > ANALISIS_SEMANA13.md => Análisis - Semana 13: Recursión y Algoritmos de Ordenamiento > #> #> Lenguaje
1 # Análisis - Semana 13: Recursión y Algoritmos de Ordenamiento "Análisis": Unknown word.
2
3 ## ■ Descripción General "Descripción": Unknown word.
4
5 La carpeta "Semana13" se enfoca en dos temas clave de algoritmos: **"recursión** y **"ordenamiento**". Contiene ejemplos y utilidades para entender la pila de llamadas, distintos estilos de implementación (iterativa vs recursiva) y algoritmos de ordenamiento clásicos (mergesort, quicksort, selection sort). "carpeta": Unknown word.
6
7 ...
8
9 ## Lenguaje "Lenguaje": Unknown word.
10
11 **JavaScript (ES6)** = implementaciones de funciones recursivas y algoritmos de ordenamiento "implementaciones": Misspelled word.
12 **HTML** = interfaz de demostración ("Index.html") "Interfaz": Unknown word.
13 **Markdown** = documentación ("README.md", "database.md", "processes.md", "design.md") "documentación": Unknown word.
14
15 ...
16
17 ## ■ Estructura de Archivos "Estructura": Unknown word.
18
19 ...
20 Semana13 "Semana": Unknown word.
21 index.html # página de demostración / ejercicios "Página": Unknown word.
22 main.js # Código de interacción o ejemplos globales "Código": Unknown word.
23 README.md # Notas generales de la semana "Notas": Unknown word.
24 database.md # Notas relacionadas (sí aplica) "Notas": Unknown word.
25 processes.md # Descripción de procesos/algoritmos "Descripción": Unknown word.
26 design.md # Diseño de soluciones o diagramas "Diseño": Unknown word.
27 recursion/
28 callStackExample.js # Implementaciones de ordenamiento "Implementaciones": Misspelled word.
29 countToZero.js
30 fibonacciIterative.js
31 fibonacciRecursive.js
32 fibonacciRecursiveBetter.js
33 Sorting/
34 mergesort.js "mergesort": Unknown word.
35 quicksort.js
36 selectionsort.js
37 ...
38
39 ...
40
41 ## ■ Objetivos "Objetivos": Unknown word.
42
43 1. Comprender la recursión: cómo funciona la pila de llamadas y cuándo usarla. "Comprender": Unknown word.
44 2. Comparar implementaciones recursivas e iterativas (ej.: Fibonacci recursivo vs iterativo). "Comparar": Unknown word.
45 3. Analizar la complejidad temporal y espacial de los algoritmos. "Analizar": Unknown word.
46 4. Calcular y comparar complejidades temporales y espaciales de cada algoritmo. "Calcular": Misspelled word.
47 5. Practicar la transformación de una solución recursiva a una iterativa y viceversa. "Practicar": Unknown word.
48
49 ...
50
51 ## ■ Descripción Detallada
52
53 ### Carpeta 'Recursion'
54 - Ejemplos y conceptos:
55 - callStackExample.js: visualiza la pila de llamadas y la entrada/salida de funciones.
56 - countToZero.js: ejemplo simple de recursión con condición base clara.
57
58
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
217
218
219
219
220
221
222
223
224
225
226
227
227
228
229
229
230
231
232
233
234
235
236
237
237
238
239
239
240
241
242
243
244
245
246
247
247
248
249
249
250
251
252
253
254
255
256
257
257
258
259
259
260
261
262
263
264
265
266
267
267
268
269
269
270
271
272
273
274
275
275
276
277
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
307
308
309
309
310
311
312
313
314
315
315
316
317
317
318
319
319
320
321
322
323
324
325
325
326
327
327
328
329
329
330
331
332
333
334
335
335
336
337
337
338
339
339
340
341
342
343
344
345
345
346
347
347
348
349
349
350
351
352
353
354
355
355
356
357
357
358
359
359
360
361
362
363
364
365
365
366
367
367
368
369
369
370
371
372
373
374
375
375
376
377
377
378
379
379
380
381
382
383
384
385
385
386
387
387
388
389
389
390
391
392
393
394
395
395
396
397
397
398
399
399
400
401
402
403
404
405
405
406
407
407
408
409
409
410
411
412
413
413
414
415
415
416
417
417
418
419
419
420
421
422
423
423
424
425
425
426
427
427
428
429
429
430
431
432
433
433
434
435
435
436
437
437
438
439
439
440
441
442
443
443
444
445
445
446
447
447
448
449
449
450
451
452
453
453
454
455
455
456
457
457
458
459
459
460
461
462
463
463
464
465
465
466
467
467
468
469
469
470
471
472
472
473
474
474
475
476
476
477
478
478
479
479
480
481
482
482
483
484
484
485
486
486
487
488
488
489
489
490
491
492
492
493
494
494
495
496
496
497
498
498
499
499
500
501
502
502
503
504
504
505
506
506
507
508
508
509
509
510
511
511
512
513
513
514
515
515
516
517
517
518
519
519
520
521
521
522
523
523
524
525
525
526
527
527
528
529
529
530
531
531
532
533
533
534
535
535
536
537
537
538
539
539
540
541
541
542
543
543
544
545
545
546
547
547
548
549
549
550
551
551
552
553
553
554
555
555
556
557
557
558
559
559
560
561
561
562
563
563
564
565
565
566
567
567
568
569
569
570
571
571
572
573
573
574
575
575
576
577
577
578
579
579
580
581
581
582
583
583
584
585
585
586
587
587
588
589
589
590
591
591
592
593
593
594
595
595
596
597
597
598
599
599
600
601
601
602
603
603
604
605
605
606
607
607
608
609
609
610
611
611
612
613
613
614
615
615
616
617
617
618
619
619
620
621
621
622
623
623
624
625
625
626
627
627
628
629
629
630
631
631
632
633
633
634
635
635
636
637
637
638
639
639
640
641
641
642
643
643
644
645
645
646
647
647
648
649
649
650
651
651
652
653
653
654
655
655
656
657
657
658
659
659
660
661
661
662
663
663
664
665
665
666
667
667
668
669
669
670
671
671
672
673
673
674
675
675
676
677
677
678
679
679
680
681
681
682
683
683
684
685
685
686
687
687
688
689
689
690
691
691
692
693
693
694
695
695
696
697
697
698
699
699
700
701
701
702
703
703
704
705
705
706
707
707
708
709
709
710
711
711
712
713
713
714
715
715
716
717
717
718
719
719
720
721
721
722
723
723
724
725
725
726
727
727
728
729
729
730
731
731
732
733
733
734
735
735
736
737
737
738
739
739
740
741
741
742
743
743
744
745
745
746
747
747
748
749
749
750
751
751
752
753
753
754
755
755
756
757
757
758
759
759
760
761
761
762
763
763
764
765
765
766
767
767
768
769
769
770
771
771
772
773
773
774
775
775
776
777
777
778
779
779
780
781
781
782
783
783
784
785
785
786
787
787
788
789
789
790
791
791
792
793
793
794
795
795
796
797
797
798
799
799
800
801
801
802
803
803
804
805
805
806
807
807
808
809
809
810
811
811
812
813
813
814
815
815
816
817
817
818
819
819
820
821
821
822
823
823
824
825
825
826
827
827
828
829
829
830
831
831
832
833
833
834
835
835
836
837
837
838
839
839
840
841
841
842
843
843
844
845
845
846
847
847
848
849
849
850
851
851
852
853
853
854
855
855
856
857
857
858
859
859
860
861
861
862
863
863
864
865
865
866
867
867
868
869
869
870
871
871
872
873
873
874
875
875
876
877
877
878
879
879
880
881
881
882
883
883
884
885
885
886
887
887
888
889
889
890
891
891
892
893
893
894
895
895
896
897
897
898
899
899
900
901
901
902
903
903
904
905
905
906
907
907
908
909
909
910
911
911
912
913
913
914
915
915
916
917
917
918
919
919
920
921
921
922
923
923
924
925
925
926
927
927
928
929
929
930
931
931
932
933
933
934
935
935
936
937
937
938
939
939
940
941
941
942
943
943
944
945
945
946
947
947
948
949
949
950
951
951
952
953
953
954
955
955
956
957
957
958
959
959
960
961
961
962
963
963
964
965
965
966
967
967
968
969
969
970
971
971
972
973
973
974
975
975
976
977
977
978
979
979
980
981
981
982
983
983
984
985
985
986
987
987
988
989
989
990
991
991
992
993
993
994
995
995
996
997
997
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
158
```

```
File Edit Selection View Go Run Terminal Help < > Q estructuras2025-master
ANALISIS_SEMANA8.md ANALISIS_SEMANA13.jsmd X
estructuras2025-master > Semana13 > ANALISIS_SEMANA13.jsmd >> # Análisis - Semana 13: Recursión y Algoritmos de Ordenamiento "Analisis": Unknown word
 51 - ## Breve descripción del tema
 52 - ## Carpeta "Recursión"
 53 - - `callStackExample.js`: visualiza la pila de llamadas y la entrada/salida de funciones.
 54 - - `countToZero.js`: ejemplo simple de recursión con condición base clara.
 55 - - `fibonacciRecursive.js` vs `fibonacciIterative.js`: comparación directa de rendimiento y complejidad.
 56 - - `fibonacciRecursiveBetter.js`: optimizaciones como memoización para mejorar recursión.
 57 -
 58 - ## Carpeta "Sorting"
 59 - - `mergesort.js`: algoritmo divide y vencerás con complejidad $O(n \log n)$ y espacio adicional $O(n)$.
 60 - - `quicksort.js`: algoritmo promedio $O(n \log n)$, peor caso $O(n^2)$; discusión sobre pivot y particionamiento.
 61 - - `selectionSort.js`: algoritmo sencillo con complejidad $O(n^2)$, útil para entender principios básicos.
 62 -
 63 - ## Carpeta "Sorting"
 64 - - `mergesort.js`: algoritmo divide y vencerás con complejidad $O(n \log n)$ y espacio adicional $O(n)$.
 65 - - `quicksort.js`: algoritmo promedio $O(n \log n)$, peor caso $O(n^2)$; discusión sobre pivot y particionamiento.
 66 - - `selectionSort.js`: algoritmo sencillo con complejidad $O(n^2)$, útil para entender principios básicos.
 67 -
 68 - ## Breve Análisis Técnico
 69 -
 70 - ## Recursión
 71 - - Conceptos clave:
 72 - - **Caso base**: condición que detiene la recursión.
 73 - - **Llamada recursiva**: descomposición del problema en subproblemas.
 74 - - **Pila de llamadas**: cada llamada ocupa espacio en la pila; riesgo de "stack overflow" si la profundidad es grande.
 75 - - Optimización: "memoización" para evitar recalcular subproblemas (ej. Fibonacci), y "tail recursion" (cuando el return es el último).
 76 - - Trade-offs: la recursión suele ser más legible, pero puede usar más memoria; la versión iterativa suele ser más eficiente en espacio.
 77 -
 78 - ## Ordenamiento
 79 - - Complejidades y características:
 80 - - `Mergesort.js`: $O(n \log n)$ tiempo, $O(n)$ espacio; estable; buen rendimiento en datos grandes.
 81 - - `Quicksort.js`: $O(n \log n)$ tiempo, $O(n^2)$ espacio; in-place posible; elegir buen pivote (randomizado o median-of-three) reduce probabilidad de peor caso.
 82 - - `Selection Sort.js`: $O(n^2)$ tiempo, $O(1)$ espacio; sencillo pero insuficiente para grandes n ; útil en contextos con memoria limitada y tamaños pequeños.
 83 - - Consideraciones prácticas: para datos parcialmente ordenados o datos grandes en memoria limitada, elegir el algoritmo adecuado según caso.
 84 -
 85 - ## Recomendaciones de práctica
 86 - - Adadir pruebas de rendimiento simples (timing) comparando implementaciones para distintos tamaños de entrada.
 87 - - Visualizar recursión y ordenamiento (pequeñas animaciones) ayuda a la comprensión.
 88 - - Implementar memoización en problemas recursivos con subproblemas repetidos.
 89 -
 90 - ## Puntos de Aprendizaje
 91 -
 92 - Entender la recursión conceptualmente es esencial para muchos algoritmos; la memoización convierte soluciones recursivas exponenciales en polinómicas.
 93 - Los algoritmos de ordenamiento enseñan estrategias (divide & conquer, in-place, stable vs unstable) y son la base para algoritmos más avanzados.
 94 - Comparar implementaciones y medir con casos de prueba reales afianza la elección del algoritmo.
 95 -
 96 -
 97 -
 98 -
 99 - ## Conclusiones
100 -
101 - "Semana13" es una semana clave para interiorizar principios algorítmicos: recursión y ordenamiento. Practicar con ejemplos concretos (Fibonacci, mergesort, quicksort) y medir su comportamiento hará que los conceptos teóricos se vuelvan aplicables en problemas reales de programación y optimización.
```

## Análisis - Semana 14: Proyecto Demo — Objetos, Interactividad y Estilos

### Descripción General

La carpeta Semana14 contiene un pequeño proyecto de demostración que integra **HTML**, **CSS** y **JavaScript** para practicar conceptos de objetos en JavaScript, manipulación del DOM y estilos. Incluye ejemplos y notas que sirven como referencia para aplicar patrones básicos de organización y diseño de UI.

---

### Lenguaje

- **HTML5** — estructura de la página (index.html).
  - **CSS3** — estilos en styles.css.
  - **JavaScript (ES6+)** — lógica en app.js y ejemplos en demo/objetos.js.
  - **Markdown** — notas.md con apuntes o recordatorios.
- 

### Estructura de Archivos

Semana14/

```
|── index.html # Página principal del demo
|── app.js # Lógica principal / handlers de UI
|── notas.md # Apuntes y notas de la semana
|── styles.css # Estilos y layout del demo
|── .vscode/ # Configuración del espacio (opcional)
└── demo/
 └── objetos.js # Ejemplos prácticos con objetos JS (clases, literales)
```

---

### Objetivos

1. Practicar la creación y uso de **objetos** en JavaScript (literales y/o clases).

2. Manipular el DOM para actualizar la interfaz desde app.js.
  3. Aplicar estilos responsivos y coherentes con styles.css.
  4. Separar responsabilidades: lógica (app.js) vs presentación (index.html/styles.css).
  5. Documentar aprendizajes y observaciones en notas.md.
- 

### Descripción Detallada

- index.html sirve como plantilla de la demo: contiene elementos interactivos (botones, formularios o tarjetas) que conectan con app.js.
  - app.js implementa la lógica de interacción: event listeners, manipulación del DOM (crear/actualizar nodos), y llamadas a funciones del demo.
  - demo/objetos.js contiene ejemplos didácticos sobre objetos: creación de objetos literales, uso de this, constructores o clases, métodos y ejemplos de mutación vs inmutabilidad.
  - styles.css define la apariencia: layout (grid/flex), variables de color, y estilos responsivos.
  - notas.md recopila observaciones, tareas pendientes o recordatorios del autor.
- 

### Breve Análisis Técnico

#### Organización y buenas prácticas

- La separación index.html (markup) / styles.css (presentación) / app.js (comportamiento) es correcta y facilita mantenimiento.
- demo/objetos.js es útil como módulo de ejemplos; sería ideal importarlo explícitamente desde app.js (ES Modules) para mejorar modularidad.

#### Interactividad y rendimiento

- Para muchas operaciones DOM, usar DocumentFragment o actualizaciones por lotes reduce repaints.
- Evitar lógica inline (onclick) y preferir addEventListener en app.js mejora testabilidad.

## Mantenimiento y escalabilidad

- Convertir ejemplos en módulos (/demo, /lib) y usar type="module" en <script> prepara el proyecto para crecimiento.
- Añadir un README.md en la carpeta con instrucciones de ejecución (usar servidor local) ayuda a reproducir la demo.

## Accesibilidad

- Verificar alt en imágenes, foco en inputs, y atributos ARIA donde aplique.
- Asegurar contraste adecuado en styles.css para cumplir con WCAG básicos.



## Recomendaciones rápidas

- Usar módulos ES (import / export) para demo/objetos.js y app.js.
- Añadir un pequeño script serve (p. ej. npx http-server) en README.md para pruebas locales.
- Documentar ejemplos en notas.md con ejemplos de entrada/salida para cada función demostrada.



## Conclusión

Semana14 es una práctica compacta que integra objetos en JavaScript con manipulación del DOM y estilos CSS. Con pequeñas mejoras (modularización, documentación y accesibilidad) la demo puede evolucionar a una colección de componentes reutilizables y un material de referencia útil para estudiantes.

**File Edit Sélection View Go Run Terminal Help**

**EXPLORER**

- ESTRUCTURAS2025-MASTER
  - estructuras2025-master
    - > Semana 11
    - > Semana1
    - > Semana2
    - > Semana3
    - > Semana4
    - > Semana5
    - > Semana6
    - > Semana7
    - > Semana8
    - > Semana9
    - > Semana10
    - > Semana11
    - > Semana12
    - > Semana13
    - > Semana14
      - > vscode
      - > demo
      - ANALISIS\_SEMANA14.md
      - JS app.js
      - index.html
      - notas.md
      - # styles.css
    - > Semana15
      - > graph
      - > hashtable
      - > linkedlist
      - ANALYSIS\_SEMANA15.md
      - # index.css
      - index.html
      - JS index.js
      - notas.md

**OUTLINE**

**TIMELINE**

**File Edit Sélection View Go Run Terminal Help**

**EXPLORER**

- ESTRUCTURAS2025-MASTER
  - estructuras2025-master > ANALISIS\_SEMANA14.md > # Análisis - Semana 14: Proyecto Demo – Objetos, Interactividad y Estilos > # Lenguaje
  - 1 # Análisis - Semana 14: Proyecto Demo – Objetos, Interactividad y Estilos "Análisis": Unknown word.
  - 2
  - 3 ## ■ Descripción General "Descripción": Unknown word.
  - 4
  - 5 La carpeta "Semana14" contiene un pequeño proyecto de demostración que integra \*\*HTML\*\*, \*\*CSS\*\* y \*\*JavaScript\*\* para practicar conceptos de objetos en JavaScript, manipulación del DOM y estilos. Incluye ejemplos y notas que sirven como referencia para aplicar patrones básicos de organización y diseño de UI. "carpeta": Unknown word.
  - 6
  - 7 ...
  - 8 ## Lenguaje "Lenguaje": Unknown word.
  - 9
  - 10 - \*\*HTML\*\* – estructura de la página (index.html). "estructura": Unknown word.
  - 11 - \*\*CSS\*\* – estilos en styles.css. "estilos": Unknown word.
  - 12 - \*\*JavaScript (ES6+)\*\* – lógica en app.js y ejemplos en demo/objetos.js. "lógica": Unknown word.
  - 13 - \*\*Markdown\*\* – notas.md con apuntes o recordatorios. "notas": Unknown word.
  - 14
  - 15 ...
  - 16
  - 17 ## ■ Estructura de archivos "Estructura": Unknown word.
  - 18
  - 19 ...
  - 20
  - 21 Semana14/ "Semana": Unknown word.
    - index.html # Página principal del demo "Página": Unknown word.
    - app.js # Lógica principal / handlers de UI "lógica": Unknown word.
    - notas.md # Apuntes y notas de la semana "notas": Unknown word.
    - styles.css # Estilos y layout del demo "estilos": Unknown word.
    - vscode/ # Configuración del espacio (opcional) "Configuración": Unknown word.
    - demo/
      - objetos.js # Ejemplos prácticos con objetos JS (clases, literales) "objetos": Unknown word.
  - 22 ...
  - 23
  - 24
  - 25
  - 26
  - 27
  - 28
  - 29
  - 30
  - 31
  - 32
  - 33 ## Objetivos "Objetivos": Unknown word.
  - 34
  - 35 1. Practicar la creación y uso de \*\*objetos\*\* en JavaScript (literales y/o clases). "Practicar": Unknown word.
  - 36 2. Manipular el DOM para actualizar la interfaz desde app.js. "actualizar": Unknown word.
  - 37 3. Aplicar estilos responsivos y coherentes con styles.css. "Aplicar": Unknown word.
  - 38 4. Separar responsabilidades: lógica (app.js) vs presentación (index.html / styles.css). "Separar": Unknown word.
  - 39 5. Documentar aprendizajes y observaciones en notas.md. "Documentar": Unknown word.
  - 40
  - 41
  - 42
  - 43
  - 44
  - 45 ## ■ Descripción Detallada "Descripción": Unknown word.
  - 46
  - 47 - index.html: Sirve como plantilla de la demo; contiene elementos interactivos (botones, formularios o tarjetas) que conectan con app.js. "Sirve": Unknown word.
  - 48 - app.js: Implementa la lógica de interacción: event listeners, manipulación del DOM (crear/actualizar nodos), y llamadas a funciones del demo. "Implementa": Unknown word.
  - 49 - demo/objetos.js: Contiene ejemplos didácticos sobre objetos: creación de objetos literales, uso de this, constructores o clases, métodos y ejemplos de inmutabilidad de "objetos". "objetos": Unknown word.
  - 50 - styles.css: Define la apariencia: layout (grid/flex), variables de color, y estilos responsivos.
  - 51 - notas.md: Recopila observaciones, tareas pendientes o recordatorios del autor.
  - 52
  - 53
  - 54
  - 55 ## Breve Análisis Técnico
  - 56
  - 57
  - 58
  - 59
  - 60
  - 61
  - 62
  - 63
  - 64
  - 65
  - 66
  - 67
  - 68
  - 69
  - 70
  - 71
  - 72
  - 73
  - 74
  - 75
  - 76
  - 77
  - 78
  - 79
  - 80
  - 81
  - 82
  - 83

**OUTLINE**

**TIMELINE**

## Análisis - Semana 15: Grafos Avanzados, Tablas Hash y Listas Enlazadas

### Descripción General

Semana15 aborda implementaciones y ejemplos prácticos sobre **grafos, tablas hash y listas enlazadas** (dobles y simples). El material combina teoría y código para comprender representaciones de grafos, manejo de colisiones en tablas hash y operaciones en listas enlazadas.

---

### Lenguaje

- **JavaScript (ES6+)** — clases, módulos simples y manipulación de estructuras.
  - **HTML5** — páginas de demostración (si existen) para probar algoritmos.
  - **CSS3** — estilos mínimos para visualizar resultados (opcional).
- 

### Estructura de Archivos (observada)

Semana15/

```
| └── code.js # Ejemplos generales / utilidades
|
| └── graph/
| | └── graph.js
| | └── graph_2.js
|
| └── hashTable/ # Implementación educativa de hash table
| | └── hashTable.js

└── linkedList/ # Listas enlazadas (doble y simple)
 |
 └── dobly.js
 └── singly.js
```

---

### Objetivos

1. Comprender representaciones de grafos (lista de adyacencia, matriz) y operaciones básicas.
  2. Implementar y probar algoritmos sobre grafos (BFS/DFS o utilidades similares).
  3. Revisar la implementación de una **hash table** y técnicas de colisión.
  4. Practicar operaciones en linkedList (append, remove, find) y en doubly (prev/next).
  5. Comparar rendimiento y casos de uso de cada estructura.
- 

### Descripción Detallada

#### Carpeta graph/

- graph.js y graph\_2.js contienen implementaciones y ejemplos de grafos.
- Deben mostrar: creación de vértices, adición de aristas, impresión de la lista de adyacencia y ejemplos de traversals.
- Utilidad didáctica: modelar relaciones y aplicar BFS/DFS para búsquedas y componentes conectados.

#### Carpeta hashTable/

- hashTable.js implementa una tabla hash educativa con funciones básicas:
  - set(key, value), get(key), delete(key), has(key).
- Revisar manejo de colisiones (encadenamiento vs open addressing) y rehashing si aplica.

#### Carpeta linkedList/

- singly.js: lista simplemente enlazada con head, append, remove, find.
- doubly.js: lista doblemente enlazada con prev y next, métodos para insertar/eliminar en ambos extremos, y recorrido inverso.

- Importante mantener `length`, `head` y `tail` para eficiencia  $O(1)$  en operaciones de extremos.
- 

## Breve Análisis Técnico

### Grafos

- Representación recomendada: **lista de adyacencia** para grafos dispersos ( $E \ll V^2$ ).
- Operaciones importantes: agregar vértices/aristas  $O(1)$ , BFS/DFS  $O(V + E)$ .
- Verificar mutabilidad de la estructura al agregar/eliminar nodos.

### Tablas Hash

- Operaciones promedio  $O(1)$ ; el rendimiento depende de la calidad del hash y del factor de carga.
- Para enseñanza, el encadenamiento es más sencillo de explicar e implementar.
- Añadir pruebas con claves que provoquen colisiones para validar la robustez.

### Listas Enlazadas

- `singly`: insert/append  $O(1)$  con `tail`, búsqueda/removal  $O(n)$ .
  - `dobly`: soporta recorrido inverso y eliminación de un nodo en  $O(1)$  si se tiene referencia al nodo.
  - Casos de uso: implementación de colas, pilas, y estructuras intermedias.
- 

## Recomendaciones y Buenas Prácticas

- Modularizar código: exportar `Graph`, `HashTable`, `LinkedList` como módulos reutilizables.

- Añadir tests unitarios básicos para cada estructura (operaciones invariantes).
  - Documentar limitaciones y complejidades en un README.md dentro de cada carpeta.

## Conclusión

Semana 15 integra estructuras críticas para problemas reales: grafos para relaciones complejas, hash tables para acceso rápido por clave y listas enlazadas para manipulación dinámica. Fortalecer las implementaciones con tests, ejemplos de uso (p. ej. BFS sobre datos reales) y modularidad mejorará la enseñanza y la reutilización del código.

```
File Edit Sélection View Go Run Terminal Help C:\estrucuras2025-master ANALISIS_SEMANAS15.md ANALISIS_SEMANAS15.md C:\estrucuras2025-master

explorer
estrucuras2025-master
estrucuras2025-master
Semana11
Semana12
Semana1
Semana2
Semana3
Semana4
Semana5
Semana6
Semana7
Semana8
Semana9
Semana10
Semana11
Semana12
Semana13
Semana14
Semana15
graph
hashTable
linkedlist
ANALISIS_SEMANAS15.md
code.js
Semana16
notes.md

ANALISIS_SEMANAS15.md
Analisis - Semana 15: Grafos Avanzados, Tablas Hash y Listas Enlazadas > ## 🔮 Lenguaje
1 # Analisis - Semana 15: Grafos Avanzados, Tablas Hash y Listas Enlazadas "Analisis": Unknown word,
44 ## 📄 Descripción Detallada "Descripción": Unknown word,
46 ### Carpeta "graph.js" "Carpeta": Unknown word,
47 - Utilidad didáctica: modelar relaciones y aplicar BFS/DFS para búsquedas y componentes conectados.
49
50 #### Carpeta "hashTable"
51 "hashTable.js" implementa una tabla hash educativa con funciones básicas:
52 | - `set(key, value)`, `get(key)`, `delete(key)`, `has(key)` .
53 | - Revisar manejo de colisiones (encadenamiento o open addressing) y rehashing si aplica.
54
55 #### Carpeta "LinkedList"
56 "singly.js": lista simplemente enlazada con 'head', 'append', 'remove', 'find'.
57 - `doubly.js`: lista doblemente enlazada con 'prev' y 'next', métodos para insertar/eliminar en ambos extremos, y recorrido inverso.
58 - Importante mantener 'length', 'head' y 'tail' para eficiencia O(1) en operaciones de extremos.
59
60 ...
61
62 ## 🌐 Breve Análisis Técnico
63
64 #### Grafos
65 - Representación recomendada: "lista de adyacencia" para grafos dispersos ($E \ll V^2$).
66 - Operaciones importantes: agregar vértices/aristas ($O(1)$), BFS/DFS: $O(V + E)$.
67 - Verificar mutabilidad de la estructura al agregar/eliminar nodos.
68
69 #### Tablas Hash
70 - Operación típica: $O(1)$; el rendimiento depende de la calidad del hash y del factor de carga.
71 - Para eficiencia, el encadenamiento es más sencillo de explicar e implementar.
72 - Aadir pruebas con claves que provocuen colisiones para validar la robustez.
73
74 #### Listas Enlazadas
75 - `singly` : Insert/Append $O(1)$ con 'tail', búsqueda/removal $O(n)$.
76 - "doubly" : soporta recorrido inverso y eliminación de un nodo en $O(1)$ si se tiene referencia al nodo.
77 - Casos de uso: implementación de colas, pilas, y estructuras intermedias.
78
79 ...
80
81 ## 💡 Recomendaciones y Buenas Prácticas
82
83 - Modularizar código: exportar 'Graph', 'HashTable', 'LinkedList' como módulos reutilizables.
84 - Aadir tests unitarios básicos para cada estructura (operaciones invariantes).
85 - Documentar limitaciones y complejidades en un 'README.md' dentro de cada carpeta.
86
87 ...
88
89 ## 🎉 Conclusión
90
91 "Semana15": Integra estructuras críticas para problemas reales: grafos para relaciones complejas; hash tables para acceso rápido por clave y listas enlazadas para manipulación dinámica. Fortalecer las implementaciones con tests, ejemplos de uso (p. ej. BFS sobre datos reales) y modularidad mejorará la enseñanza y la reutilización del código.
```

## Análisis - Semana 16: Pilas, Colas y Árboles — Aplicaciones y Prácticas

### Descripción General

La carpeta Semana16 contiene implementaciones y ejemplos prácticos de **colas (queues)**, **pilas (stacks)** y **árboles (trees)**, además de una página de demostración index.html, estilos index.css y lógica front-end en index.js. Esta semana integra estructuras lineales y no lineales enfocadas en su uso práctico y aplicaciones simples.

---

### Lenguaje

- **JavaScript (ES6+)** — implementaciones y demo interactivas (index.js, queue/, stack/, tree/).
  - **HTML5** — interfaz de demostración (index.html).
  - **CSS3** — estilos en index.css.
- 

### Estructura de Archivos (observada)

Semana16/

```
|── index.html # Página demo para interactuar con estructuras
|── index.css # Estilos de la demo
|── index.js # Lógica de la demo (conexión UI ↔ estructuras)
|── queue/
| |── queue.js # Implementación de cola (enqueue, dequeue,
| |── peek)
|── stack/
| |── stack.js # Implementación de pila (push, pop, peek)
└── tree/
 └── tree.js # Implementación de árbol (posible BST o utilidades)
```

---

## Objetivos

1. Implementar y usar Queue (FIFO) y Stack (LIFO) correctamente en ejemplos prácticos.
  2. Entender cómo usar una Queue para BFS y una Stack para DFS (iterativo).
  3. Manejar operaciones básicas en Tree (inserción, búsqueda, recorridos) y conectar con la UI.
  4. Mostrar interactividad con index.html para experimentar con las estructuras.
  5. Aplicar buenas prácticas: modularidad, separación UI/logic y manejo de errores.
- 

## Descripción Detallada

### **queue/queue.js**

- Implementa los métodos básicos:
  - enqueue(item) — agregar al final
  - dequeue() — eliminar del frente
  - peek() — ver el elemento frontal
  - isEmpty() — comprobar si está vacía
- Recomendación: implementar con head/tail (lista ligada) o buffer circular para O(1) en dequeue.

### **stack/stack.js**

- Implementa push, pop, peek, isEmpty.
- Implementación simple con Array (push/pop) es aceptable para demos.

### **tree/tree.js**

- Implementación de árbol (probablemente BST) con insert, search y recorridos (inOrder, preOrder, postOrder).
- Usos prácticos: mostrar nodos en la UI, ordenar datos mediante inOrder si es BST.

### **index.html + index.js + index.css**

- index.html contiene controles (formularios/botones) para interactuar con las estructuras.
  - index.js conecta eventos de la UI con llamadas a los métodos de Queue/Stack/Tree y actualiza la vista.
  - index.css estiliza la visualización de nodos, colas o pilas para facilitar la comprensión.
- 



### **Breve Análisis Técnico**

#### **Complejidades y recomendaciones**

- Queue y Stack bien implementadas tienen operaciones  $O(1)$  para encolar/desencolar y push/pop.
- Tree (BST) operaciones: insert/search  $O(\log n)$  promedio,  $O(n)$  peor caso; documentar limitaciones.
- Para demos con muchos elementos, optimizar renderizado usando DocumentFragment y actualizaciones parciales del DOM.

#### **Robustez y UX**

- Validar entradas en la UI y mostrar mensajes de error cuando las operaciones no sean válidas (por ejemplo, dequeue en cola vacía).
- Añadir visualizaciones simples (lista horizontal para Queue, vertical para Stack, diagrama básico para Tree) ayuda al aprendizaje.

#### **Pruebas y mantenibilidad**

- Separar las implementaciones (queue/queue.js, stack/stack.js, tree/tree.js) y exportarlas como módulos facilita pruebas unitarias.

- Añadir pruebas simples (p. ej. scripts que verifican invariantes) mejora la confiabilidad del material didáctico.
- 

### Puntos de Aprendizaje

- Entender cuándo usar Queue vs Stack según patrón de acceso (FIFO vs LIFO).
  - Relacionar Queue con algoritmos de recorrido por niveles (BFS) y Stack con DFS iterativo.
  - Conocer las limitaciones de un BST no balanceado y la necesidad de balanceo en escenarios reales.
  - La integración UI ↔ estructuras transforma conceptos teóricos en experiencia práctica.
- 

### Conclusión

Semana 16 cierra el ciclo práctico mostrando estructuras de acceso simple y árboles, y cómo exponerlas en una UI interactiva. Mejoras sugeridas: modularizar como ES modules, añadir tests y optimizar la visualización para datasets más grandes.

The screenshot shows a Microsoft Visual Studio Code window with the following details:

- File Explorer:** Shows a tree view of files and folders. The root folder is "ESTRUCTURAS2025-MASTER". Inside it are "estructuras2025-master" and "ANALISIS SEMANA16". "ANALISIS SEMANA16" contains subfolders "Semana 11" through "Semana 16", and files "index.css", "index.html", "indexjs", and "notas.md".
- Editor:** The main editor area displays the content of the file "ANALISIS SEMANA16.html". The code includes various comments (e.g., "# Analisis", "## Descripción General", "## Estructura de Archivos (observada)", "## Objetivos", "## Descripción Detallada") and sections of code, many of which are flagged as "Unknown word" by the linter.
- Status Bar:** At the bottom, it shows "Ln 15 Col 4" and icons for "Spaces 2", "UTF-8", "CRLF", "Markdown", "Go Live", and "Go Live".

File Edit Selection View Go Run Terminal Help

ESTRUCTURAS-MASTER

- estructuras2025-master
  - Semana 11
  - Semana 12
  - Semana 1
  - Semana 2
  - Semana 3
  - Semana 4
  - Semana 5
  - Semana 6
  - Semana 7
  - Semana 8
  - Semana 9
  - Semana 10
  - Semana11
  - Semana12
  - Semana13
  - Semana14
  - Semana15
  - Semana16
    - > queue
    - > stack
    - > tree
- ANALISIS\_SEMANA16.md
- index.css
- index.html
- index.js
- notas.md

ANALISIS\_SEMANA16.md X

estructuras2025-master > Semana16 > ANALISIS\_SEMANA16.md => Análisis : Semana 16: Pilas, Colas y Árboles - Aplicaciones y Prácticas > => # Lenguaje

## Análisis : Semana 16: Pilas, Colas y Árboles - Aplicaciones y Prácticas "Análisis" Unknown word.

44 ## ■ Descripción Detallada: "Descripción", Unknown word.

54 ### "stack/stack.js"

57

58 ## 'tree/tree.js'

59 ## Implementación de árbol (probablemente BST) con "insert", "search" y recorridos ("inorder", "preOrder", "postorder"). "Implementación": Unknown word.

60 - Buena práctica: mostrar nodos en la UI, ordenar datos mediante "inorder" si es BST.

61

62 ## "index.html" <- "index.js" + "index.css"

63 - Index.html contiene controles (formularios/inputs) para interactuar con las estructuras.

64 - Index.js conecta eventos de la UI con llamadas a los métodos de "queue"/"stack"/"Tree" y actualiza la vista.

65 - Index.css utiliza la visualización de nodos, colas o pilas para facilitar la comprensión.

66

67 ...

68 ## 🔍 Breve Análisis Técnico

69

70 ## Complejidades y recomendaciones

71 - Queue y "stack" bien implementados tienen operaciones O(1) para encolar/desencolar y push/pop.

72 - Tree (BST) operaciones: "insert" / "search" O(log n) promedio, O(n) peor caso; documentar limitaciones.

73 - Para darles más elementos, optimizar renderizado usando DocumentFragment y actualizaciones parciales del DOM.

74

75 ## Robustez y UX

76 - Validar entradas en la UI y mostrar mensajes de error cuando las operaciones no sean válidas (por ejemplo, "dequeue" en cola vacía).

77 - Añadir visualizaciones simples (lista horizontal para "Queue", vertical para "Stack", diagrama básico para "Tree") ayuda al aprendizaje.

78

79 ## Pruebas y mantenibilidad

80 - Separar las implementaciones ("queue/queue.js", "stack/stack.js", "tree/tree.js") y exportarlas como módulos facilita pruebas unitarias.

81 - Añadir pruebas simples (p. ej. scripts que verifican invariantes) mejora la confiabilidad del material didáctico.

82

83

84

85

86 ## 🌟 Puntos de Aprendizaje

87

88 - Entender cuál es "Queue" vs "Stack" según patrón de acceso (FIFO vs LIFO)

89 - Relacionar "Queue" con algoritmos de recorrido por niveles (BFS) y "Stack" con DFS iterativo.

90 - Conocer las limitaciones de un BST no balanceado y la necesidad de balanceo en escenarios reales.

91 - La integración UI + estructuras transforma conceptos teóricos en experiencia práctica.

92

93

94

95 ## 📚 Conclusiones

96

97 - "Semana16" cierra el ciclo práctico mostrando estructuras de acceso simple y árboles, y cómo exponerlas en una UI interactiva. Mejoras sugeridas: modularizar como ES modules, añadir tests y optimizar la visualización para datasets más grandes.

98

> OUTLINE

> TIMELINE

↳ Launched ⏱ 0 ⏳ 0 ⏴ 0 ⏵ 0

Ln 15 Col 4 Spaces 2 UFT-8 CR/LF | Markdown E/E Go Live