VISUALIZADOR INTERACTIVO DE GRAFOS

Graficas por Computadora

NOVEMBER 28, 2025

GASPAR ALONSO CARDOS UC
LCC

Indice

Visualizador Interactivo de Grafos

Resumen Técnico

El proyecto consiste en una **Single Page Application** (**SPA**) capaz de renderizar grafos y simular algoritmos de búsqueda paso a paso. Se diferencia de visualizadores tradicionales basados en DOM/SVG al utilizar **WebGL**, lo que permite aprovechar la aceleración por hardware (**GPU**) para dibujar y animar grafos complejos con alto rendimiento y fluidez.

Arquitectura del Sistema

Estructura de Módulos

El código fuente se organiza en módulos especializados para facilitar la mantenibilidad:

- **App.jsx (Controlador):** Orquestador principal. Gestiona el ciclo de vida de la aplicación, los estados globales (graphData, history, isPlaying), los temporizadores de animación (setInterval) y la interacción con el usuario.
- **Algorithms.js (Lógica de Negocio):** Contiene la implementación pura de los algoritmos (BFS, DFS, Dijkstra, IDA*). Devuelve un historial de pasos (snapshots) en lugar de manipular el DOM directamente.
- **GraphGenerator.js (Generación de Datos):** Módulo matemático encargado de crear grafos procedimentales (Aleatorio, Preferencial, Completo).
- **Metrics.js (Análisis):** Funciones puras para el cálculo de propiedades topológicas (Densidad, Grado Promedio).
- **PseudocodeData.js (Datos Estáticos):** Diccionario que contiene la representación textual de los algoritmos para el panel didáctico.

Frontend

- **Framework:** React 18 (Manejo de estado y ciclo de vida de componentes).
- **Build Tool:** Vite (Empaquetado optimizado y entorno de desarrollo rápido).
- **Motor Gráfico:** reagraph. Una librería de alto nivel sobre WebGL que implementa renderizado de grafos y algoritmos de distribución espacial (Force-Directed Layout).

Estructuras de Datos

Internamente, el grafo se gestiona mediante dos estructuras principales:

1. Modelo JSON (Vista y Persistencia): Arrays de objetos serializables compatibles con el motor de renderizado y la exportación de archivos.

```
{
  "nodes": [{ "id": "1", "label": "N1", "fill": "#..." }],
  "edges": [{ "source": "1", "target": "2", "weight": 5 }]
}
```

2. Listas de Adyacencia (Lógica Algorítmica): Durante la ejecución, el grafo se convierte a un Map<String, Array> para permitir un acceso de complejidad O(1) a los vecinos de cualquier nodo.

Algoritmos Implementados

La lógica algorítmica está completamente desacoplada de la visualización. Los algoritmos no manipulan el DOM directamente; en su lugar, generan un **Historial de Estados** (**Snapshot History**) que la interfaz reproduce posteriorment.

Estrategia de Animación

Cada algoritmo retorna un array de objetos Step, representando un fotograma de la animación:

```
{
  active: "ID_Nodo",         // Nodo siendo procesado actualmente
  visited: ["ID_1", ...],    // Lista de nodos ya explorados
  openList: [...],           // Estado actual de la Cola/Pila/PQ (para el panel didáctico)
  line: 3,                   // Línea de pseudocódigo a resaltar
  description: "Texto...",   // Mensaje narrativo para el log
  distances: { ... },        // (Solo Dijkstra) Mapa de costos acumulados
  finalPath: [...]           // (Solo último paso) Ruta reconstruida
}
```

El componente App.jsx utiliza un temporizador (setInterval) para recorrer este array y actualizar las propiedades visuales (color, tamaño) del grafo en cada frame.

Detalle de Algoritmos

- **BFS (Anchura):** Utiliza una **Cola** (**FIFO**). Garantiza el camino más corto en grafos no ponderados.
- **DFS (Profundidad):** Utiliza una **Pila** (**LIFO**) implícita o explícita. Prioriza la exploración de ramas profundas.
- **Dijkstra:** Implementado con una **Cola de Prioridad** lógica. Maneja pesos en las aristas.
    - **Manejo de Errores:** Incluye sanitización de datos. Si una arista no tiene peso definido en el JSON, se le asigna valor **1** automáticamente para evitar fallos matemáticos (**NaN**).
- **IDA* (Iterative Deepening A):** Implementación recursiva acotada por un umbral de costo (**f = g + h**). Al no tener coordenadas espaciales garantizadas, se utiliza una heurística **h(n)=0**. Visualiza cómo el algoritmo reinicia la búsqueda expandiendo su límite ("threshold") iterativamente.

Reconstrucción de Ruta

Se implementa una función de backtracking (reconstructPath). Al encontrar el objetivo, se recorre el mapa de padres desde el destino hasta el inicio. Esta ruta se renderiza visualmente pintando las aristas y nodos de color dorado (#F1C40F).

Algoritmo de Dibujo (Force-Directed Layout)

La distribución espacial de los nodos no es estática, sino una simulación física en tiempo real:

1. **Nodos:** Actúan como partículas cargadas que ejercen fuerzas de repulsión (evitan superposición).
2. **Aristas:** Actúan como resortes que ejercen fuerzas de atracción (mantienen la cohesión).
3. **Centro de Gravedad:** Una fuerza de gravedad mantiene el grafo centrado en el lienzo.

Esto permite que, sin importar la topología del grafo cargado, el visualizador encuentre automáticamente una disposición estéticamente agradable y legible.

Lógica de Coloreado Dinámico

La función getVisualGraph intercepta el estado actual de la animación y aplica estilos condicionales.

- **Normalización de Tipos:** Se implementó una conversión explícita String(id) para comparar IDs. Esto soluciona inconsistencias cuando los grafos importados usan enteros y los algoritmos usan cadenas.
- **Optimización:** Para resaltar la ruta final, se utiliza un Set con claves bidireccionales ("A->B" y "B->A") para identificar las aristas de la solución en tiempo constante.

Interfaz de Usuario (UI/UX)

La interfaz se diseñó con un layout de tres paneles para maximizar la comprensión didáctica:

1. **Panel de Control:** Configuración de grafos, selección de algoritmos y gestión de archivos (Carga/Exportación JSON).
2. **Controles de Tiempo:** Se implementó un control de reproducción no lineal que permite al usuario pausar y seguir con el paso a paso y ajustar la velocidad para analizar decisiones específicas del algoritmo.
3. **Lienzo WebGL:** Visualización interactiva con soporte para Zoom, Paneo, Tooltips (información al pasar el mouse) y Leyenda de colores.
4. **Panel Didáctico:**
    a. **Pseudocódigo:** Resalta la línea de código exacta que se ejecuta en cada paso.
    b. **Frontera:** Muestra en tiempo real el contenido de la estructura de datos (Cola/Pila).
5. **Reportes:** El sistema permite generar y descargar un archivo .txt con el registro completo de la ejecución ("Log"), detallando cada paso tomado por el algoritmo.

Cálculo de Métricas

El sistema analiza la topología del grafo en tiempo real:

- **Densidad:** $D = \frac{2|E|}{|V|(|V|-1)}$ (para grafos no dirigidos). Mide qué tan cerca está el grafo de ser completo.
- **Grado Promedio:** Indica la conectividad media de la red.

Despliegue

El proyecto está configurado para integración continua (CI/CD) mediante GitHub Pages. El script de construcción (npm run build) genera archivos estáticos optimizados en la carpeta /dist, asegurando que la aplicación sea accesible públicamente vía web.