TPO

Programación II

Integrantes del equipo:

Borras Juan Cruz

Davidson Victoria

Gomez Gonzalo Martín

López Agostina Sol

Villaverde Pilar Rocío

Docente:

Nicolás Pérez

Carrera:

Ingeniería en Informática

Institución:

Universidad Argentina de la Empresa

Consigna:

- Buscar un problema real que se resuelva con alguno de los 3 algoritmos de Grafos Pesados que vimos en clase.

- Crear o conseguir un programa que muestre este problema resuelto con alguno de estos algoritmos, bajo TDA.

- En el repositorio del proyecto crear un PDF breve (¾) paginas donde se explique el problema, la solución y las diferencias si las hay, con el algoritmo mostrado en clase. Además si en la investigación apareció algún otro algoritmo comentarlo y buscar un testeo bajo TDA.

Diagnóstico del problema

La rapidez con la que los servicios de emergencia, especialmente los cuerpos de bomberos, responden ante una llamada puede ser la diferencia entre un susto y una tragedia. Desde los primeros cuerpos organizados de bomberos en el siglo XVII hasta las unidades modernas equipadas con tecnología de punta, uno de los objetivos fundamentales ha sido siempre el mismo: llegar lo antes posible al lugar donde se los necesita. Esta meta, sin embargo, se enfrenta hoy a nuevos desafíos urbanos: el crecimiento acelerado de las ciudades, los embotellamientos, las obras públicas permanentes, las manifestaciones, y las condiciones climáticas variables hacen que elegir una ruta eficiente no sea una tarea trivial.

Durante años, las rutas eran decididas por la experiencia del conductor o por protocolos preestablecidos, sin considerar en tiempo real el estado actual del tránsito. En muchos casos, la falta de una planificación dinámica ha provocado demoras innecesarias que podrían haberse evitado con herramientas más avanzadas. Hoy, gracias a los avances en informática, inteligencia artificial y modelado de sistemas complejos, es posible optimizar los recorridos con algoritmos diseñados específicamente para encontrar caminos más cortos o rápidos en estructuras de red. En particular, el **algoritmo A*** se presenta como una herramienta ideal para este tipo de situaciones.

Para poder aplicar un algoritmo como A*, es necesario primero transformar el espacio físico, es decir, una ciudad, en una estructura computacional que se pueda procesar. Para eso se recurre a la teoría de grafos. Un grafo es un conjunto de nodos (también llamados vértices) y aristas (o enlaces) que representan conexiones entre los primeros. En este contexto:

- Los nodos pueden representar intersecciones, estaciones de bomberos, puntos de referencia o incluso entradas a edificios importantes.

- Las aristas representan tramos de calles, y cada una tiene un peso asignado. Este peso puede ser la distancia física, pero lo más útil en nuestro caso es utilizar el tiempo estimado de tránsito.

Este modelo permite que una ciudad entera, incluso con miles de calles, pueda ser analizada matemáticamente. A partir de ahí, se pueden aplicar algoritmos de búsqueda que exploren el grafo en busca de la mejor ruta.

Funcionamiento del Algoritmo A*

El algoritmo A* (A estrella) es una extensión del algoritmo de Dijkstra, con la diferencia clave de que A* utiliza una **heurística** para estimar la distancia restante al destino. Su funcionamiento se basa en evaluar, para cada nodo, la suma de dos valores:

- **g(n)**: el costo real acumulado desde el nodo de inicio hasta el nodo actual n.
- **h(n)**: una estimación (heurística) del costo desde el nodo n hasta el nodo destino.

Por lo tanto, el costo que implica llegar de un nodo a otro se representa con la siguiente fórmula:

f(n) = g(n) + h(n)

La clave del buen funcionamiento de A* está en definir correctamente esa heurística. En el caso del tránsito urbano, podría basarse en la distancia en línea recta (usando, por ejemplo, la fórmula de Haversine si se trabaja con coordenadas geográficas) o en un tiempo promedio calculado a partir de datos históricos.

Al priorizar la exploración de los nodos con menor costo/peso total, A* evita perder tiempo en caminos que, aunque parezcan prometedores en principio, se alejan del objetivo final. Así, encuentra la ruta más corta en el menor tiempo posible, considerando tanto lo ya recorrido como lo que falta.

Aplicación en los Servicios de Bomberos

Implementar este algoritmo en los servicios de bomberos puede realizarse a través de un sistema inteligente de despacho. Dicho sistema, conectado a una base de datos en tiempo real sobre el estado del tráfico, puede calcular automáticamente qué unidad debe responder a cada emergencia y cuál es la ruta óptima para hacerlo. Por ejemplo:

- Si hay dos estaciones de bomberos equidistantes a un incendio, el sistema puede determinar cuál tiene mejor acceso en ese momento, según cortes de calles, embotellamientos, semáforos sincronizados, etc.
- En caso de un incendio en un edificio de difícil acceso, puede calcularse una ruta alternativa que llegue más cerca del acceso principal o permita el despliegue más rápido de escaleras y mangueras.

Estas decisiones, tomadas en milisegundos por un sistema computacional, pueden ahorrar minutos vitales en una situación de vida o muerte.

Además, la implementación de A* puede complementarse con mapas digitales interactivos, pantallas en los camiones y sistemas de navegación con voz, que guíen al conductor paso a paso. En una segunda etapa, se podrían incluir incluso datos del clima, horarios escolares (para evitar zonas con tráfico de entrada/salida), eventos especiales, protestas o manifestaciones, entre otros.

Desafíos y Recomendaciones

Aplicar este tipo de soluciones no está exento de dificultades. Algunos de los principales desafíos incluyen:

- **Precisión de la heurística**: si la estimación es muy baja o muy alta, puede afectar el rendimiento del algoritmo.

- **Calidad de los datos**: la información del tráfico debe estar actualizada en tiempo real. Para ello, puede integrarse con sensores, cámaras de tránsito y aplicaciones de movilidad urbana como Waze o Google Maps (mediante APIs).

- **Capacitación del personal**: los operadores de los servicios deben saber interpretar y confiar en las rutas sugeridas por el sistema.

- **Robustez del sistema**: debe funcionar correctamente incluso bajo presión, ante cortes de energía, caídas del sistema, etc.

Pese a estos retos, los beneficios que se pueden obtener son enormes. Un solo minuto de diferencia en la llegada puede reducir drásticamente la extensión del fuego o evitar que una estructura colapse.

Conclusión

La combinación entre teoría matemática, ciencia computacional y necesidades operativas concretas demuestra cómo la tecnología puede mejorar servicios públicos críticos. El algoritmo A* se presenta como una solución robusta, eficiente y adaptable para mejorar la toma de decisiones en situaciones de emergencia.

Al integrar esta herramienta en los servicios de bomberos, no solo se optimiza la logística, sino que se construye un sistema más resiliente, más ágil y con mayor capacidad de respuesta ante imprevistos. En última instancia, esta implementación representa una mejora en la calidad del servicio y en la protección de la comunidad.

En el futuro, la incorporación de inteligencia artificial, aprendizaje automático y análisis predictivo podría llevar este tipo de sistemas a un nuevo nivel, anticipándose incluso a los

focos de emergencia mediante análisis de riesgo territorial. Así, la tecnología no solo ayudaría a responder mejor, sino a prevenir más.

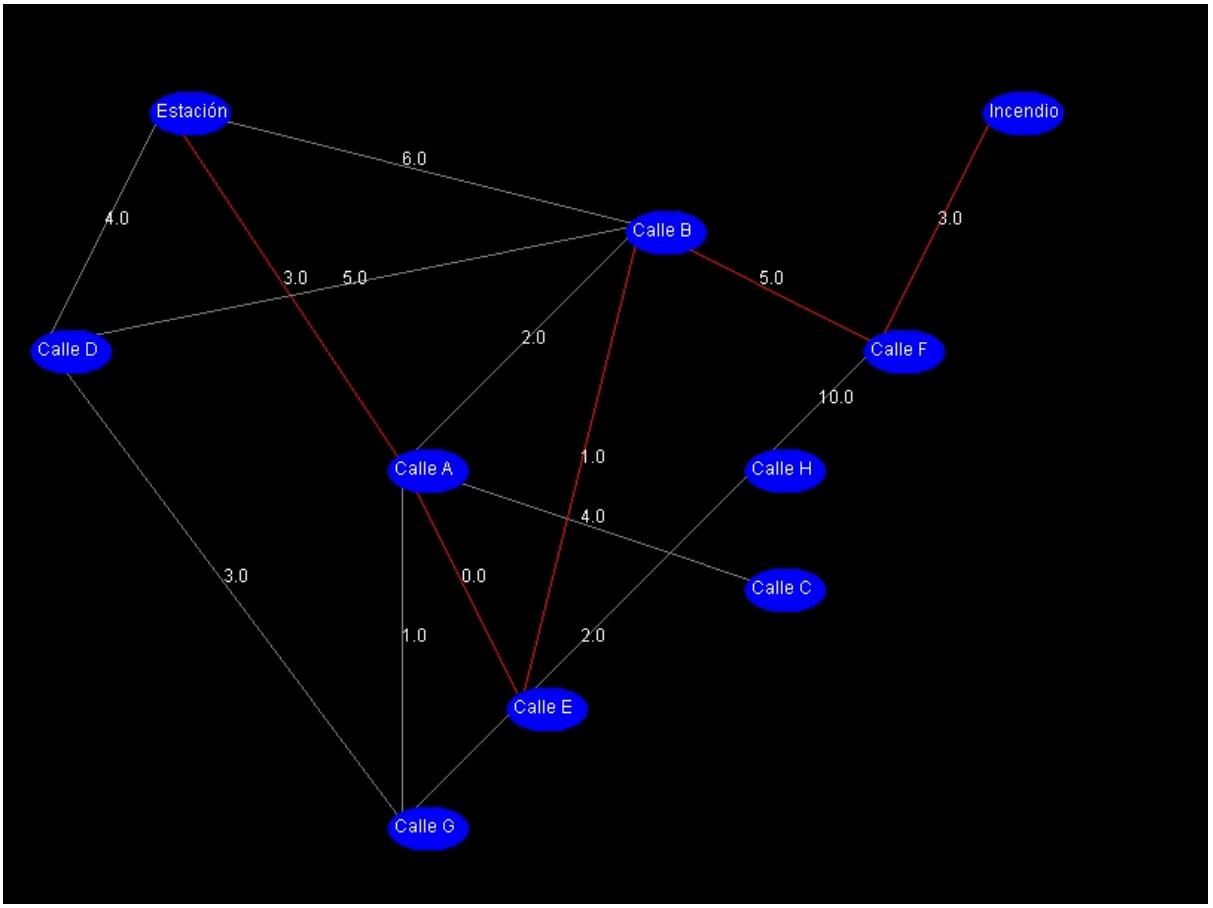
Ejemplo de ejecución del programa:

Input:

```
Nodo estacion = new Nodo(nombre:"Estación", x:1, y:1);
Nodo A = new Nodo(nombre:"Calle A", x:3, y:4);
Nodo B = new Nodo(nombre:"Calle B", x:5, y:2);
Nodo C = new Nodo(nombre:"Calle C", x:6, y:5);
Nodo D = new Nodo(nombre:"Calle D", x:0, y:3);
Nodo E = new Nodo(nombre:"Calle E", x:4, y:6);
Nodo F = new Nodo(nombre:"Calle F", x:7, y:3);
Nodo G = new Nodo(nombre:"Calle G", x:3, y:7);
Nodo H = new Nodo(nombre:"Calle H", x:6, y:4);
```

```
ciudad.conectar(estacion, A, peso:3);
ciudad.conectar(estacion, B, peso:6);
ciudad.conectar(estacion, D, peso:4);
ciudad.conectar(A, C, peso:4);
ciudad.conectar(B, A, peso:2);
ciudad.conectar(G, A, peso:1);
ciudad.conectar(D, G, peso:3);
ciudad.conectar(G, H, peso:2);
ciudad.conectar(H, F, peso:10);
ciudad.conectar(B, D, peso:5);
ciudad.conectar(F, incendio, peso:3);
ciudad.conectar(B, F, peso:5);
ciudad.conectar(E, B, peso:1);
ciudad.conectar(A, E, peso:0);
```

Output:


Estación
Incendio
6.0
4.0
Calle B
3.0
3.0
5.0
5.0
Calle D
2.0
Calle F
10.0
Calle A
1.0
Calle H
4.0
3.0
0.0
Calle C
1.0
2.0
Calle E
Calle G