UNIVERSIDAD AUTÓNOMA "TOMAS FRÍAS" INGENIERÍA DE SISTEMAS	
NOMBRE: Univ. Alvaro Rene Condori Quispe	MATERIA: Investigación operativa II
DOCENTE: Ing. Ditmar Castro Angulo	SIGLA: SIS-544
FECHA: 27/09/2024	INFORME

1) BFS

El algoritmo de búsqueda en anchura BFS es un método fundamental en teoría de grafos para explorar nodos y sus conexiones. Este algoritmo opera utilizando una estructura de datos de cola, lo que le permite visitar todos los vértices en un nivel antes de pasar al siguiente. En el contexto de su implementación en una interfaz gráfica, como se muestra en el código proporcionado, BFS resulta ser eficiente para la búsqueda de rutas en grafos densos y no dirigidos, donde se busca la conexión más corta entre un nodo fuente y todos los demás nodos.

La velocidad del BFS es principalmente el número de aristas. Esta complejidad se deriva del hecho de que el algoritmo visita cada vértice una vez y recorre cada arista una vez. Sin embargo, su rendimiento puede verse afectado por factores externos como el diseño del grafo, la implementación del algoritmo, y las características de las vías en un contexto de optimización de rutas.

La optimización de rutas se puede ver beneficiada al considerar características como la superficie de la vía. Por ejemplo, una vía pavimentada puede tener un menor "costo" de tránsito que una vía de tierra. Incorporar estos atributos en la estructura del grafo, como pesos en las aristas, puede resultar en una mejora significativa en la selección de rutas más eficientes. Esto significa que, aunque BFS puede ser rápido para la exploración inicial, la incorporación de características de la superficie de la vía permite crear un algoritmo de búsqueda de caminos más inteligente y adaptable, que prioriza rutas con mejores condiciones.

2) DFS

El algoritmo de búsqueda en profundidad DFS es una técnica fundamental en la exploración de grafos que sigue un enfoque recursivo o iterativo. A diferencia del algoritmo BFS, que utiliza una cola, DFS emplea una pila para explorar un camino hasta que no hay más nodos que visitar, retrocediendo posteriormente para explorar otros caminos. Su complejidad temporal es el número de aristas, similar a BFS. Sin embargo, DFS puede ser menos eficiente en grafos densos debido a su tendencia a profundizar en un camino antes de explorar otras opciones.
La implementación de DFS en una interfaz gráfica, como se evidencia en el código proporcionado, permite observar visualmente el proceso de exploración. Este enfoque es particularmente útil en aplicaciones donde se requiere una búsqueda exhaustiva, como en la resolución de laberintos o en la identificación de componentes conexos en un grafo. Sin embargo, la naturaleza recursiva de DFS puede provocar problemas de rendimiento en grafos muy profundos.
En el contexto de optimización de rutas, como en el caso de elegir caminos en función de la superficie de la vía, DFS puede no ser la mejor opción. Este algoritmo podría encontrar un camino que parece óptimo en la primera exploración, pero no garantiza que sea el más eficiente en términos de condiciones de la vía. Por ejemplo, una ruta que sigue una carretera pavimentada puede ser más

rápida que una que se adentra en caminos de tierra, a pesar de que el DFS podría seleccionar inicialmente un camino diferente.
En conclusión, aunque DFS es eficaz para explorar grafos y encontrar soluciones en problemas específicos, su uso en la optimización de rutas debe ser complementado con estrategias que consideren las características de las vías, asegurando que la selección final de rutas sea no solo válida, sino también óptima en términos de eficiencia y condiciones de tránsito.

3) PRIM

El algoritmo de Prim es una técnica fundamental en la teoría de grafos que se utiliza para encontrar el árbol de expansión mínima de un grafo conectado y ponderado. Este algoritmo funciona construyendo el MST de forma incremental, comenzando desde un vértice inicial y agregando aristas de menor peso que conectan los vértices visitados con los no visitados. La complejidad temporal de Prim depende de la implementación de la estructura de datos utilizada para manejar las aristas; en el caso de utilizar una cola de prioridad.

La implementación presentada del algoritmo de Prim utiliza una cola de prioridad para seleccionar eficientemente la arista de menor peso en cada paso. A medida que se agregan vértices al MST, el algoritmo garantiza que se están considerando las mejores opciones disponibles, lo que resulta en un árbol que minimiza el costo total de las conexiones. Esta propiedad lo convierte en una herramienta eficaz para la optimización de rutas, especialmente en aplicaciones como el diseño de redes de transporte o telecomunicaciones.

En el contexto de optimización de rutas, como en la planificación de carreteras, considerar las características de la superficie de la vía es crucial. Por ejemplo, si se priorizan caminos pavimentados sobre caminos de tierra en la implementación del algoritmo, se puede modificar la estructura de costos en las aristas. Esto permite que el algoritmo de Prim no solo busque el menor costo de distancia, sino que también tenga en cuenta factores como la calidad de la superficie y las condiciones de tránsito.

En conclusión, el algoritmo de Prim es eficiente para encontrar el MST en grafos ponderados, y su aplicación puede optimizar rutas al integrar características relevantes de las vías. Sin embargo, para maximizar su eficacia, es vital adaptar el modelo de costos en función de la superficie de la vía y otros factores operativos, asegurando así que el resultado final no solo sea económico, sino también práctico y seguro para el tránsito.

4) DIJKSTRA

El algoritmo de Dijkstra es un método ampliamente utilizado para encontrar la ruta más corta entre un nodo de inicio y todos los demás nodos en un grafo ponderado sin aristas de peso negativo. Su eficiencia se basa en el uso de una cola de prioridad, lo que permite seleccionar de manera óptima el siguiente vértice a procesar. La complejidad temporal del algoritmo, cuando se utiliza una cola de prioridad basada en un montículo binario, es el número de aristas.

La implementación del algoritmo de Dijkstra en el código presentado inicia estableciendo la distancia al vértice de inicio como cero y asignando un valor infinito a las distancias de los demás vértices. A medida que se procesan los vértices, el algoritmo actualiza las distancias y el camino más corto hacia cada nodo vecino. Este enfoque garantiza que, al final del proceso, se obtenga la ruta más corta al vértice de destino especificado.

En el contexto de la optimización de rutas, es crucial considerar características adicionales como la superficie de la vía. La calidad de las carreteras puede influir significativamente en el tiempo y el costo de viajar a través de ellas. Por ejemplo, caminos pavimentados suelen ser más rápidos y seguros que los caminos de tierra, lo que debería reflejarse en el peso de las aristas. Al ajustar los costos en las aristas del grafo según la superficie de la vía, se puede garantizar que el algoritmo de Dijkstra no solo encuentre la ruta más corta en términos de distancia, sino también la más eficiente en términos de condiciones de tránsito.

En conclusión, el algoritmo de Dijkstra es eficiente para determinar la ruta más corta en grafos ponderados. Sin embargo, su eficacia puede mejorarse al integrar características de la superficie de la vía en la modelación de costos, asegurando que las rutas seleccionadas no solo sean óptimas en términos de distancia, sino también en eficiencia y seguridad para el usuario final.