Análisis de Algoritmos y Estructuras de Datos
CI0116 – Grupo 4

Andrew Umaña Quiros B37091

Silvio Castillo C38910

Proyecto 2

Grafo de Recursos

Análisis Final de Proyecto

Como proyecto final se creó un juego donde el objetivo del jugador es adquirir cierta cantidad de recursos. Esa recolección se logra conectando la cuenta con la base del jugador, restando a los recursos la distancia de la ruta formada.

La implementación de este juego requirió la creación de un grafo, representado mediante una matriz de adyacencia, y la generación de rutas utilizando algoritmos de búsqueda para grafos.

Este proyecto sirvió para demostrar diferentes aspectos de los algoritmos de búsqueda para grafos.

Dicho esto, proseguimos a responder las siguientes preguntas propuestas por el enunciado del proyecto.

Analisis de Optimalidad:

1. Máquina BFS/DFS:

La primera opción para conectar nodos fue el algoritmo BFS. BFS (búsqueda por anchura) busca el camino que utiliza la menor cantidad de aristas entre un nodo A y un nodo B, revisando todos los nodos vecinos antes de pasar al siguiente nivel

a. ¿Por qué este algoritmo casi siempre genera la peor ganancia?

BFS no toma en cuenta el peso de las aristas, solo la cantidad de aristas. Esto terminaba dando resultados no óptimos. En muchas ocasiones, el algoritmo encontraba una ruta que incluso resultaba en una cantidad de recursos adquiridos negativa.

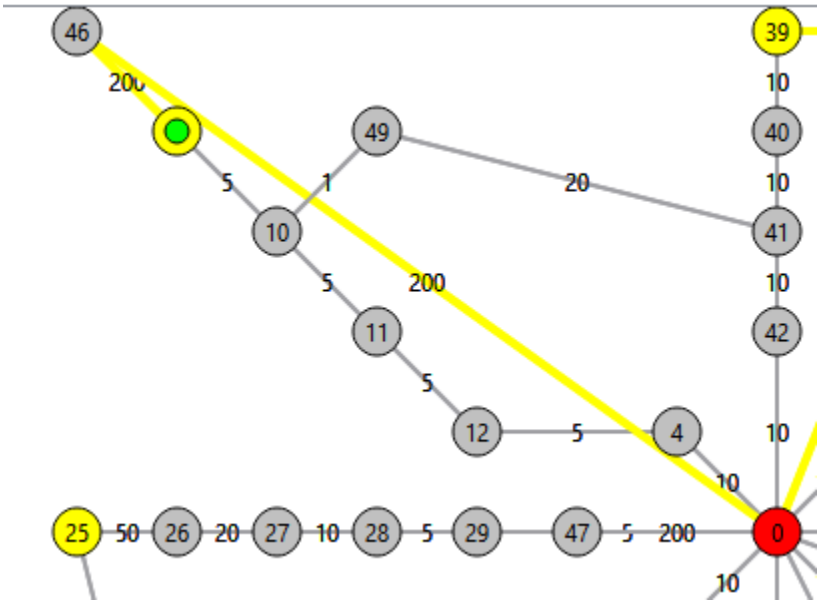
b. ¿Garantiza BFS o DFS encontrar algún camino? Sí/No y por qué

BFS siempre recorrerá todos los nodos de los vértices haya un objetivo o no. Teniendo en cuenta que este grafo es fuertemente conectado, BFS y DFS siempre habrían logrado encontrar un camino.

c. Explique cómo la ignorancia de los pesos lleva a rutas costosas.

BFS no toma en cuenta el peso de las aristas, solo la cantidad de aristas. Esto terminaba dando resultados no óptimos. En muchas ocasiones, el algoritmo encontraba una ruta que incluso resultaba en una cantidad de recursos adquiridos negativa.

d. Muestre una captura de pantalla de una tubería nivel 1, que tome un camino visiblemente absurdo y costoso en su mapa.



2. Máquina Greedy:

Para la máquina Greedy se eligió usar el algoritmo de Prim. Este algoritmo genera un árbol recubridor mínimo, es decir, un árbol cuya suma de pesos de las aristas es lo más baja posible.

a. ¿Por qué este algoritmo es mejor que el BFS/DFS, pero aun así no es perfecto?

Para la máquina Greedy se eligió usar el algoritmo de Prim. Este algoritmo genera un árbol recubridor mínimo, es decir, un árbol cuya suma de pesos de las aristas es lo más baja posible.

b. Describa y compare los conceptos de óptimo local y óptimo global.

Óptimo global es cuando se genera un resultado óptimo para todo el sistema; un MST es un ejemplo de esto. Óptimo local se refiere a un resultado óptimo para un componente, como la ruta más eficiente para un solo nodo.

c. Diseñe o encuentre un escenario en su mapa donde el algoritmo greedy tome una mala decisión.

Un ejemplo de esto sería usar prim en un grafo altamente conectado; un MST tendrá la suma más baja para todas las aristas. Pero si queremos buscar la ruta entre nodos A y B, prim no

garantiza que tenga la mejor ruta para estos dos. Si hay una ruta donde A-B que tenga valor de 10, pero Prim decidió omitir esa arista porque era muy pesada comparada a las demás, entonces que MST permitió fue A-6-C-6-B la suma va a dar una ruta de peso 12, lo cual no es óptimo.

3. Máquina Dijkstra/A*:

El tercer método utilizado fue el algoritmo de Dijkstra.

a. ¿Por qué este algoritmo siempre encuentra el camino más rentable?

Dijkstra es similar a Prim; utiliza una cola de prioridad para decidir cuál es el siguiente camino a evaluar, pero además almacena las rutas ya encontradas para compararlas con nuevas rutas. Esto garantiza que el algoritmo encuentre las mejores rutas desde un nodo A hacia todos los demás. Si se tiene un nodo objetivo B, este algoritmo definitivamente encuentra la ruta más corta.

b. Explique cómo explora el grafo en comparación con los otros dos.

El algoritmo simplifica la búsqueda, pues el peso de cada ruta ya está almacenado en un arreglo de distancias. Esto lo hace más eficiente que Prim, que requiere un algoritmo auxiliar para buscar la ruta resultante.

Análisis temporal

1. Declare la complejidad temporal (Big O) de BFS, DFS y Dijkstra en función de los vértices y las aristas.

Con V siendo el número de vértices (nodos) y E el número de aristas, las complejidades temporales son:

BFS: O(V + E)

BFS es relativamente eficiente. El uso de la cola es O(1), por lo que el número de vértices termina siendo el valor dominante. Para cada vértice, se revisan todas sus aristas, evitando repeticiones.

Prim: O(E log V)

En su versión estándar, la creación de la cola de prioridad toma O(V). La cola de prioridad hace que la revisión de la conexión sea más rápida dando O(log V). Luego, cada vértice revisa sus conexiones, realizando E para extracción de la cola de prioridad.

Dijkstra: O(E log V)

Dijkstra funciona de forma muy similar a Prim. Utiliza una cola de prioridad mínima para decidir cuál es la siguiente ruta a revisar, la diferencia es que esta cola guarda la suma de rutas encontradas desde un nodo en particular, no solo de aristas.

2. Si usó una cola de prioridad (Dijkstra), ¿Cómo afecta esto la complejidad en comparación con una búsqueda simple en un arreglo?

Una implementación simple basada en arreglos tendría $O(V^2)$, pues para cada vértice se revisa un arreglo entero. Sin embargo, al usar una cola de prioridad mínima, el tiempo se reduce a O(E log V).

3. Si Dijkstra es más lento de calcular, ¿por qué se utiliza

De los tres algoritmos, Dijkstra no es el más rápido en su versión básica, pero sí es el único que resuelve específicamente el problema de encontrar la mejor distancia de un nodo a todos los demás. Para el juego, este algoritmo da los mejores resultados, pues define su ruta con base en un nodo específico en lugar de crear un árbol con la suma mínima de pesos.

Análisis de estructura de datos

Este grafo en particular tenía una gran cantidad de nodos, pero no estaban fuertemente conectados. Muchos nodos solo tenían a lo más dos conexiones. Aunque todos los nodos eran accesibles, se requerían muchos nodos intermedios.

1. ¿Cómo influye la densidad del mapa (cantidad de aristas) en la elección entre una matriz y una lista de adyacencia?

Para este grafo se decidió usar una lista de adyacencia debido a su manejo más eficiente de memoria. Si se hubiera usado una matriz, muchas celdas contendrían información inútil. La lista solo guarda las conexiones necesarias.

2. ¿Cómo habría cambiado el rendimiento (memoria/velocidad) de los algoritmos si hubieran elegido la otra estructura?

Es posible también que el rendimiento de los algoritmos hubiera empeorado usando una matriz de adyacencia. Muchos de estos algoritmos requieren recorrer las conexiones de cada nodo. En este grafo, al ser esparzo, la matriz tendría muchas celdas con 0, y el recorrido incluiría muchas iteraciones sin utilidad.