UNIVERSIDAD DE COSTA RICA

Facultad de Ingeniería

Escuela de Ciencias de la Computación e Informática

CI-0116 Análisis de Algoritmos y Estructuras de Datos

PROYECTO II: PROTOCOLO DE GRAFOS

Estudiantes

Brandon Alfaro Saborio - C4C251

Esteban Miranda Parajeles - C14801

Hermes Josue Rojas Sancho - C16882

Henry Josué Rodríguez Sánchez - C26700

Diciembre 2025

Análisis Final

1. Análisis de Optimalidad

Un camino de menor costo total genera más ganancias. Separando lo sucedido en el proyecto con las 3 máquinas de construcción:

Máquina BFS/DFS:

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

   R/ La máquina que usa BFS o DFS casi siempre termina generando la peor ganancia porque ambos algoritmos son búsquedas no informadas y no toman en cuenta los pesos de las aristas cuando deciden hacia dónde avanzar.

   BFS encuentra el camino con la menor cantidad de pasos, pero esa ruta “corta” puede estar compuesta por aristas muy caras. DFS, se mete profundo por un camino sin evaluar cuánto cuesta, lo que puede llevarlo por rutas largas o con aristas costosas.

   Como ninguno considera los pesos, los costos finales suelen ser muy altos y eso afecta directamente la ganancia obtenida.

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

   R/ Sí. En el contexto del proyecto, el grafo es no dirigido y conexo, por lo que BFS y DFS siempre van a encontrar un camino si este existe. Ambos algoritmos visitan todos los nodos alcanzables desde el punto de inicio, así que mientras el destino esté conectado, eventualmente lo alcanzan.

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

   R/ Esta ignorancia de pesos es justo lo que provoca rutas caras.

   BFS se guía únicamente por reducir la cantidad de pasos, sin importar si cada paso cuesta poco o muchísimo. DFS se va profundo siguiendo el primer camino disponible, incluso si ese camino tiene aristas muy caras o da rodeos innecesarios.

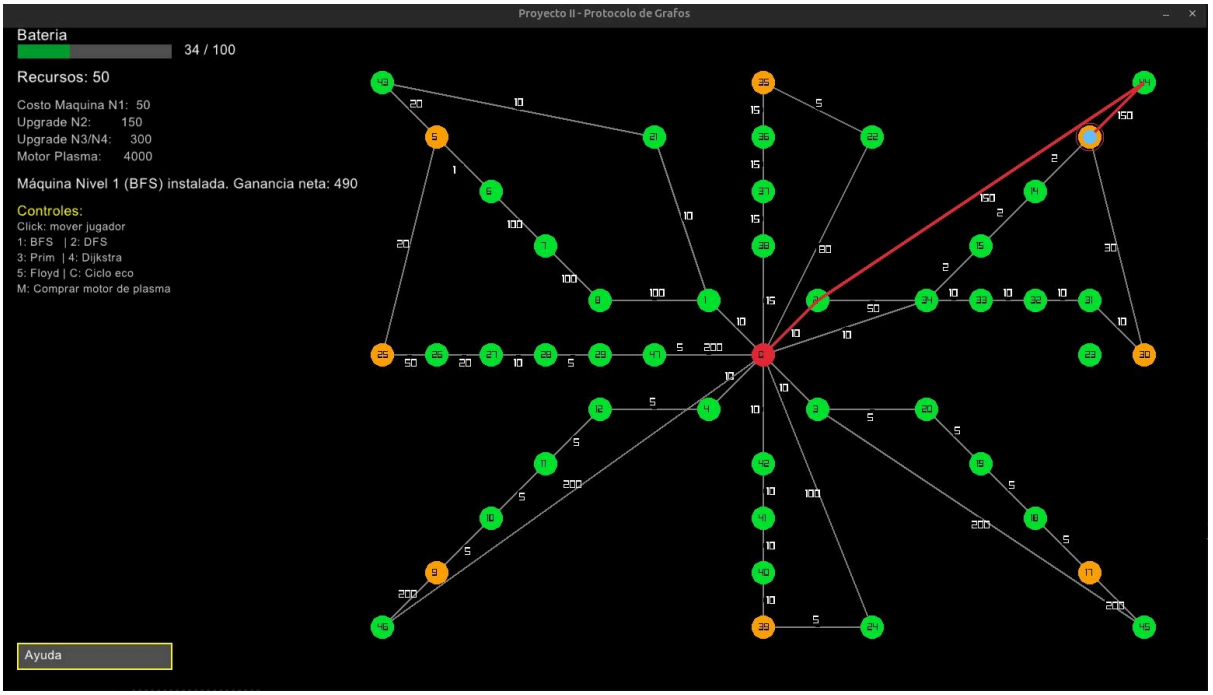
   En los dos casos, se cumplen los objetivos del algoritmo, pero sin evaluar el costo real, lo que lleva fácilmente a rutas que no son las más baratas.

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

   Captura de pantalla del programa:

   Tubería de nivel 1 creada utilizando algoritmo BFS o Búsqueda por Anchura: (aristas marcadas en color rojo)

   Toma un camino con los mínimos pasos posibles pero un costo total de 310, demostrando cómo la ignorancia de los pesos de los algoritmos de las máquinas nivel 1 puede llevar a rutas demasiado costosas en la mayoría de ocasiones.



Máquina Greedy:

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

   R/ Prim sí toma en cuenta los pesos al seleccionar aristas, lo que ya lo pone por encima de BFS y DFS. Sin embargo, su objetivo es construir un árbol recubridor mínimo y no encontrar el camino más barato entre dos nodos específicos.

   La ruta que sale del MST puede ser razonable, pero no necesariamente es la más barata posible. Por eso, aunque mejora respecto a nivel 1, el algoritmo no garantiza la mejor ruta entre origen y destino.

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

R/ El óptimo local es una solución que parece la mejor dentro de un conjunto pequeño de opciones inmediatas. Prim trabaja así: toma la arista más barata disponible en ese momento sin ver todo el panorama.

En cambio, el óptimo global es la mejor solución de todas las posibles. Dijkstra y Floyd-Warshall llegan a este tipo de solución porque comparan rutas completas y se quedan con la más barata de todas.

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

R/ Un escenario donde el algoritmo de Prim toma una mala decisión (o una decisión subóptima) ocurre en la ruta desde el nodo 13 (recurso) hasta el nodo 0 (base).

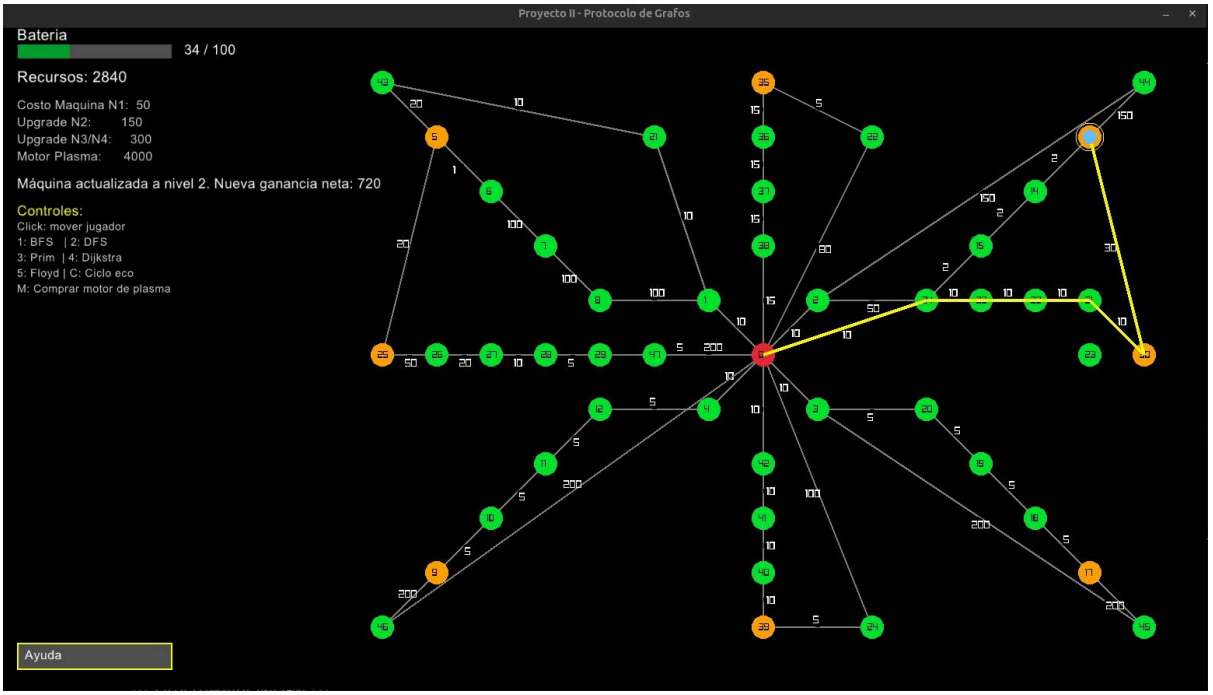
- Ruta de Prim (MST): Costo total = 80.
- Ruta Optima (Dijkstra y Floyd-Warshall): Costo total = 66.

El sobrecosto es de 21.2%. Esto muestra que confiar completamente en el nivel 2 no es ideal cuando se busca el camino más rentable.

Capturas de pantalla del programa:

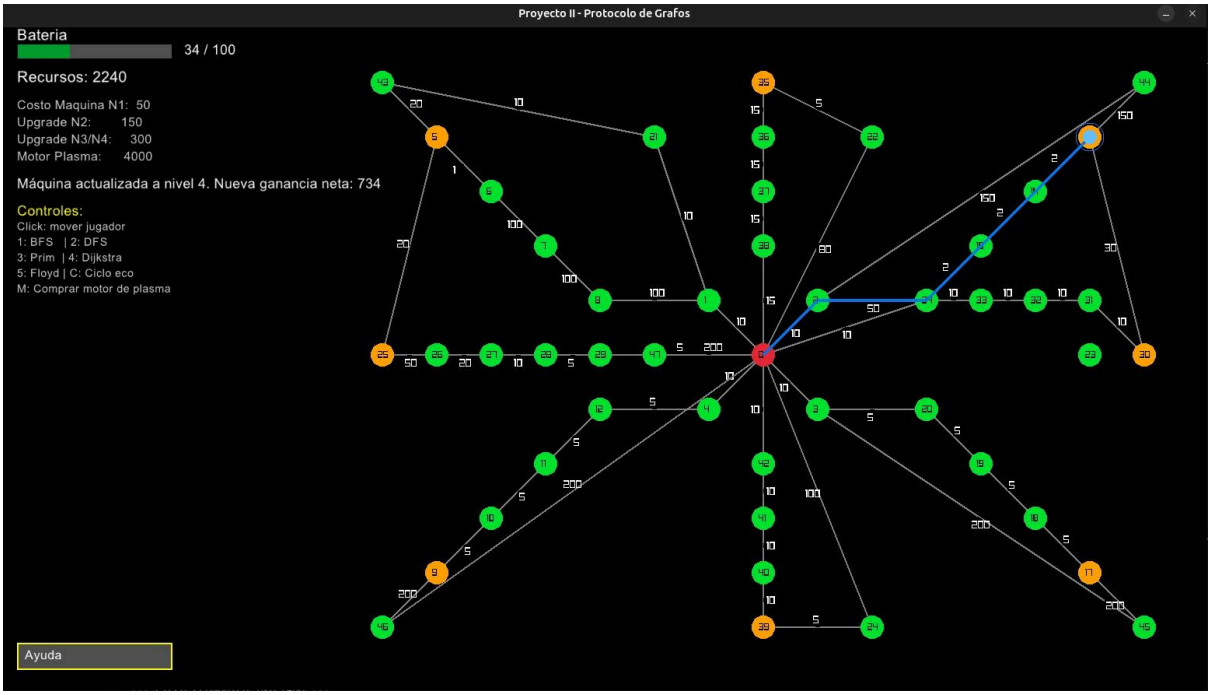
Tubería de nivel 2 creada utilizando algoritmo de Prim: (aristas marcadas en color amarillo)

Toma un camino con un costo total de 80.



Tubería de nivel 4 creada utilizando algoritmo de Floyd-Warshall: (aristas marcadas en color azul)

Toma un camino con un costo total de 66.



Máquina Dijkstra/A*:

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

   R/ En las pruebas del proyecto, los únicos algoritmos que siempre dieron el camino más barato fueron Dijkstra y Floyd-Warshall. Esto se debe a que ambos están diseñados específicamente para resolver el problema del camino mínimo.

   Dijkstra calcula el costo mínimo desde un nodo inicial hacia cualquier destino, mientras que Floyd-Warshall calcula las mejores rutas entre todos los pares de nodos. En ambos casos, se comparan todas las opciones relevantes y el resultado corresponde al óptimo global.

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

- **Dijkstra:**
  Explora el grafo usando una cola de prioridad. Cada vez que encuentra nuevas

rutas, las guarda ahí y siempre trabaja primero con la de menor costo. Desde esa ruta barata va expandiendo hasta llegar al destino.

- **Floyd-Warshall:**
  No recorre el grafo paso a paso. Usa una matriz donde guarda las mejores distancias entre todos los nodos y la va actualizando cuando encuentra una combinación que resulta más barata. También utiliza una matriz de predecesores para reconstruir las rutas cuando se necesitan.

  Comparación con niveles anteriores:
  Mientras BFS/DFS avanzan sin fijarse en pesos y Prim construye un MST sin optimizar rutas específicas, Dijkstra sí se centra en la ruta más barata desde el inicio hasta el destino.
  Floyd-Warshall evalúa todas las combinaciones posibles, quedándose con la mejor entre cada par de nodos.

2. Análisis de Complejidad Temporal

En esta sección analizarán cuán la velocidad en la que se calcula la ruta, óptima para los algoritmos desarrollados:

Para los análisis solicitados tómese **V** como los “vértices” y **N** como el número de aristas o conexiones de un nodo.

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

- **BFS (Búsqueda en Anchura):** Se utiliza una cola, se recorre el grafo utilizando la función grafo.obtenerAdyacente(), el algoritmo visita cada vértice y arista una sola vez, por ende, su complejidad es de O(V+N).
- **DFS (Búsqueda en Profundidad):** se utiliza una pila, la lógica del recorrido del grafo en cuanto a visitas es igual que la del BFS, visita cada vértice y arista una sola vez, su complejidad también es O(V+N)
- **Dijkstra:** Se utiliza una cola de prioridad, cada vez que se encuentra un camino más corto se hace un “push” a la cola, que se encarga de decir cual es el siguiente camino a tomar, esta inserción tiene un costo de O(LogV), en el peor de los casos esto podría ocurrir para cada una de las aristas, por lo que su complejidad final es O(N*LogV).
- **Floyd-Warshall:** Este algoritmo cuenta con tres bucles anidados que recorren todos los vértices, su complejidad temporal es de O(V^3).

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

- La cola de prioridad se ve como una mejor opción siempre que se trabaja con grafos dispersos, es decir, grafos donde hayan menos aristas del máximo posible que podrían haber, en este caso, gracias a la cola de prioridad es que el algoritmo de Dikstra tiene la complejidad mostrada anteriormente, si se hubiera utilizado un simple vector, se tendría que recorrer por completo el mismo en cada paso hasta encontrar la mejor opción, la complejidad escala a O (V^2), ya que el primer bucle iterar sobre el vértice actual y el segundo bucle interno busca linealmente el camino más corto.

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

- El motivo del uso de este algoritmo es su eficiencia y correctitud en grafos ponderados (Siempre que no existan pesos negativos), su análisis permite encontrar de manera verídica el camino más corto con menos peso posible, si lo comparamos con BFS y DFS, vemos que ellos asumen que los pesos de todas las aristas son iguales, lo que hace que no siempre nos devuelvan el camino con menor costo posible, Dijkstra si lo hace.

3. Análisis de Estructuras de Datos

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

- La densidad del mapa influye de gran manera a la hora de elegir si usar matriz o lista de adyacencia, ya que si es un grafo disperso, la diferencia de iteraciones entre uno y otro es gigantesca, siendo la complejidad de la lista O(V+N) y de la matriz O(V^2), sin embargo, si es grafo es denso, la mejor opción sería la matriz de adyacencia.

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

- Con BFS, DFS y Dijkstra el rendimiento sería más lento, ya que actualmente los algoritmos iteran solamente sobre las conexiones existentes, con una matriz de adyacencia, los algoritmos deberían iterar sobre toda la fila de la matriz para comprobar si existe alguna conexión, lo que genera un considerable aumento uso de memoria.

  Por otro lado, con Floyd-Warshall sería un poco mejor el rendimiento ya que el algoritmo por sí solo requiere una matriz, entonces se evitaría el paso que

tenemos actualmente que construye la matriz utilizando la lista de adyacencia.