Universidad EAFIT

Estructura de datos y algoritmos I

Documentación Parcial 3 – Momento 2

Ismael García Ceballos

Carlos Alberto Álvarez Henao

Noviembre 02 de 2025

Introducción: Para este trabajo se clonó el repositorio

https://github.com/IsmaelGC24/Parcial3-EDA1.git

Durante el desarrollo de este trabajo, se utilizó ChatGPT (OpenAI) como herramienta de apoyo, la cual cumplió principalmente en la creación e implementación del algoritmo de Dijkstra y de la clase Graph definida en graph.hpp, indicando la estructura modular más adecuada en C++.
Además, propuso incluir casos de prueba adicionales para evaluar el manejo de ciclos e inalcanzables**,** lo que permitió validar la robustez del programa.

1. **Descripción del problema:**

El objetivo de esta práctica es implementar y comparar dos algoritmos para el problema de camino mínimo desde una fuente (SSSP) en grafos dirigidos:

- BFS (Breadth-First Search) para grafos no ponderados, donde la distancia se mide como el número mínimo de aristas.
- Dijkstra para grafos ponderados con pesos no negativos, donde la distancia es la suma mínima de los pesos.

El programa recibe un archivo de entrada con el número de vértices, aristas, vértice fuente y tipo de grafo. Según el valor del campo tipo, ejecuta BFS (tipo=0) o Dijkstra (tipo=1), generando como salida las distancias desde la fuente y el camino correspondiente hacia cada vértice alcanzable.

También se registran los tiempos de ejecución en milisegundos para comparar el rendimiento de ambos algoritmos en distintos tipos de grafos.

2. **Representación del Grafo:**

El grafo se implementó mediante una lista de adyacencia, ya que permite:

- Un uso de memoria proporcional a O(V + E).
- Recorrer fácilmente los vecinos de cada vértice.
- Mejor rendimiento que una matriz de adyacencia en grafos dispersos.

Cada entrada del vector principal almacena una lista de pares (v, peso) representando las aristas salientes desde un vértice u.

3. Seudocódigo de los algoritmos:

3.1 BFS (grafo no ponderado):

```
BFS(G, s):
    para cada vértice v en G:
        dist[v] = INF
        prev[v] = -1
    dist[s] = 0
    crear cola Q
    Q.encolar(s)

    mientras Q no esté vacía:
        u = Q.desencolar()
        para cada vecino v de u:
            si dist[v] == INF:
                dist[v] = dist[u] + 1
                prev[v] = u
                Q.encolar(v)
```

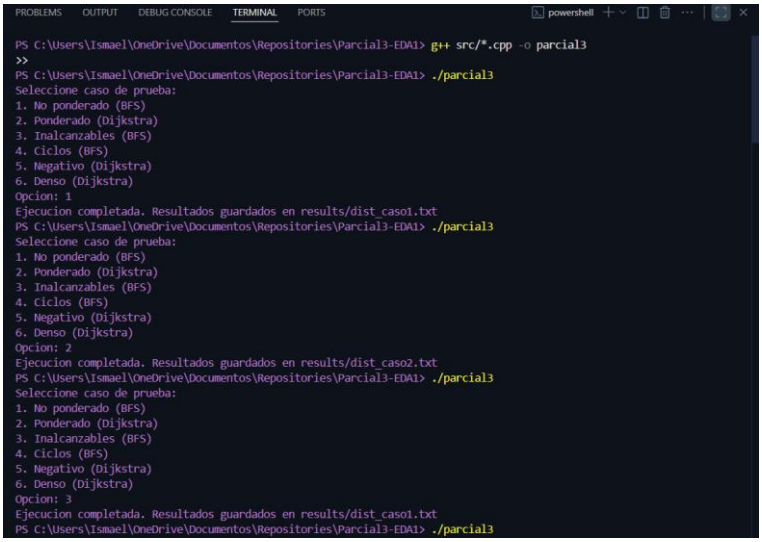
3.2 Dijkstra (grafo ponderado, sin negativos):

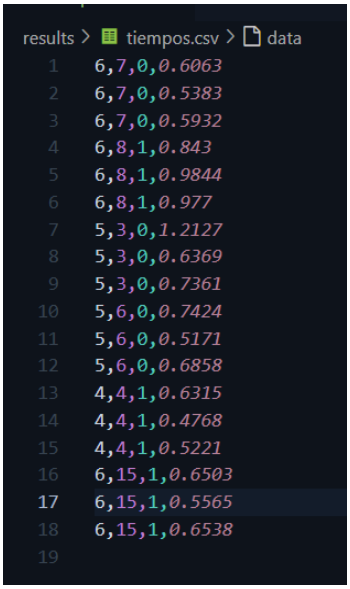
```
Dijkstra(G, s):
    para cada vértice v en G:
        dist[v] = INF
        prev[v] = -1
    dist[s] = 0

    crear min-heap PQ
    PQ.insertar((0, s))

    mientras PQ no esté vacío:
        (d, u) = PQ.extraerMin()
        si d > dist[u]:
            continuar
        para cada (v, w) en adyacentes[u]:
            si w < 0:
                reportar "no soportado (pesos negativos)"
                terminar
            si dist[v] > dist[u] + w:
                dist[v] = dist[u] + w
                prev[v] = u
                PQ.insertar((dist[v], v))
```

4. Resultados:





4.1 Formato del archivo tiempos.csv:

Cada línea del archivo results/tiempos.csv contiene los valores: n,m,tipo,t_ms

Donde:

- **n:** número de vértices del grafo.
- **m:** número de aristas del grafo.
- **tipo:** indica el algoritmo usado:
  - 0 → BFS (no ponderado)
  - 1 → Dijkstra (ponderado)
- **t_ms:** tiempo de ejecución en milisegundos para calcular los caminos mínimos.

5. **Comparación:**

Criterio	BFS	Dijkstra
Tipo de grafo	No ponderado	Ponderado (pesos ≥ 0)
Complejidad	O(V + E)	O((V + E) log V)
Estructura usada	Cola FIFO	Min-heap (priority queue)
Peso de aristas	Ignorado	Considerado
Casos con negativos	Soportado	No soportado
Uso de memoria	Bajo	Moderado
Ideal para	Caminos más cortos en grafos sin pesos	Rutas más baratas en grafos ponderados

6. **Conclusiones:**

- BFS es ideal cuando el grafo no tiene pesos o todas las aristas valen igual. Su comportamiento es lineal con respecto al número de vértices y aristas, y garantiza el menor número de saltos entre nodos.
- Dijkstra es más general, pero solo puede usarse si los pesos son no negativos. Su complejidad es mayor debido al uso de una cola de prioridad, pero proporciona caminos óptimos en términos de peso total.
- La lista de adyacencia resultó más eficiente en memoria y tiempo para los grafos dispersos, mientras que una matriz de adyacencia sería más costosa ($O(V^2)$) en estos casos.
- En los resultados experimentales, ambos algoritmos mostraron tiempos muy bajos, pero el caso denso (más aristas) evidenció un leve aumento en el costo de Dijkstra, coherente con su complejidad logarítmica.
- En presencia de aristas negativas, el programa detecta y explica la limitación del algoritmo de Dijkstra, garantizando robustez en la ejecución.

7. **Referencias:**

GitHub del profesor del curso:
https://github.com/carlosalvarezh/EstructuraDatosAlgoritmos1/blob/main/S08_EDNoLineales_Grafos.ipynb

ChatGPT, Open AI: https://chatgpt.com/