Reporte Semana 6: Algoritmo de Dijkstra

Caminos Más Cortos en Grafos Ponderados

📋 Resumen Ejecutivo

Semana: 6
Tema: Algoritmo de Dijkstra
Fecha: Diciembre 2025

🎯 Objetivos Cumplidos

- ✅ Implementar el algoritmo de Dijkstra
- ✅ Encontrar caminos más cortos ponderados
- ✅ Usar cola de prioridad para optimización
- ✅ Reconstruir caminos óptimos
- ✅ Análisis de complejidad

💻 Implementación

Algoritmo de Dijkstra

Características:

- Encuentra caminos más cortos desde un nodo origen
- Funciona con pesos positivos
- Usa cola de prioridad (min-heap)
- Complejidad: O((V + E) log V)

Componentes:

1. Cola de prioridad (SortedSet)
2. Diccionario de distancias
3. Diccionario de predecesores
4. Relajación de aristas

📊 Resultados de Ejecución

Red de Ciudades (Grafo de Prueba)

```
A --4-- B
|       |
2       1
|       |
```

```
C --8-- D --2-- E --3-- F
|       |
10      5
|       |
E ------+
```

Salida del Programa

```
🗺 Semana 6 - Algoritmo de Dijkstra

📊 Red de ciudades:
A-B(4), A-C(2), B-C(1), B-D(5), C-D(8)
C-E(10), D-E(2), D-F(6), E-F(3)

🎯 Distancias más cortas desde A:
  A → A: 0.0
  A → B: 3.0
  A → C: 2.0
  A → D: 8.0
  A → E: 10.0
  A → F: 13.0

🛣 Caminos más cortos:
  A → B: A → C → B (distancia: 3.0)
  A → C: A → C (distancia: 2.0)
  A → D: A → C → B → D (distancia: 8.0)
  A → E: A → C → B → D → E (distancia: 10.0)
  A → F: A → C → B → D → E → F (distancia: 13.0)

☑ Programa completado!
```

📈 Análisis de Complejidad

Dijkstra con Cola de Prioridad

Aspecto	Complejidad
Temporal	O((V + E) log V)
Espacial	O(V)
Estructura	SortedSet (min-heap)

Desglose:

- Inicialización: O(V)
- Bucle principal: O(V) iteraciones
- Cada extracción de cola: O(log V)
- Cada relajación: O(log V)

- Total: O((V + E) log V)

Comparación con Otros Algoritmos

Algoritmo	Complejidad	Pesos Negativos	Uso
Dijkstra	O((V+E)logV)	✖ No	Grafos con pesos positivos
Bellman-Ford	O(VE)	☑ Sí	Grafos con pesos negativos
Floyd-Warshall	O(V^3)	☑ Sí	Todos los pares
BFS	O(V+E)	N/A	Sin pesos (o peso=1)

🔍 Funcionamiento del Algoritmo

Paso a Paso

1. **Inicialización:**
    - Distancia al origen = 0
    - Distancia a todos los demás = ∞
    - Agregar origen a cola de prioridad
2. **Bucle Principal:**
    - Extraer nodo con menor distancia
    - Para cada vecino:
        - Calcular nueva distancia
        - Si es menor que la actual:
            - Actualizar distancia
            - Actualizar predecesor
            - Agregar a cola de prioridad
3. **Finalización:**
    - Cuando la cola está vacía
    - Todas las distancias mínimas calculadas

Ejemplo Visual

```
Iteración 1: Procesar A (dist=0)
  Actualizar B: ∞ → 4
  Actualizar C: ∞ → 2
  Cola: [(2,C), (4,B)]

Iteración 2: Procesar C (dist=2)
  Actualizar B: 4 → 3 (mejor: A→C→B)
  Actualizar D: ∞ → 10
  Actualizar E: ∞ → 12
  Cola: [(3,B), (4,B), (10,D), (12,E)]
```

```
Iteración 3: Procesar B (dist=3)
  Actualizar D: 10 → 8 (mejor: A→C→B→D)
  Cola: [(4,B), (8,D), (10,D), (12,E)]

... y así sucesivamente
```

🎓 Aplicaciones Prácticas

1. Navegación GPS

- Encontrar ruta más corta entre dos puntos
- Considerar distancia, tiempo o costo
- Actualización en tiempo real

2. Redes de Computadoras

- Routing de paquetes
- Protocolo OSPF (Open Shortest Path First)
- Optimización de latencia

3. Logística

- Rutas de entrega óptimas
- Minimización de costos
- Planificación de transporte

4. Juegos

- Pathfinding para NPCs
- Navegación de personajes
- Optimización de movimientos

💡 Conceptos Clave Aprendidos

1. Relajación de Aristas

```
Si distancia[u] + peso(u,v) < distancia[v]:
    distancia[v] = distancia[u] + peso(u,v)
    predecesor[v] = u
```

2. Cola de Prioridad

- Siempre procesar nodo con menor distancia
- Garantiza optimalidad
- Evita reprocesar nodos

3. Reconstrucción de Caminos

- Guardar predecesores durante exploración
- Reconstruir desde destino a origen
- Invertir para obtener camino correcto

4. Limitaciones

- ❌ No funciona con pesos negativos
- ❌ Puede dar resultados incorrectos si hay ciclos negativos
- ✅ Usar Bellman-Ford para pesos negativos

🔬 Casos de Prueba

Caso 1: Red Simple

```
Entrada: A-B(4), A-C(2), B-C(1)
Resultado: A→B = 3 (vía C)
Verificación: ✅ Correcto
```

Caso 2: Múltiples Caminos

```
Entrada: A→F (6 nodos, 9 aristas)
Resultado: A→C→B→D→E→F (13 km)
Camino directo: No existe
Verificación: ✅ Óptimo
```

Caso 3: Nodo Aislado

```
Entrada: Grafo con nodo G aislado
Resultado: Distancia a G = ∞
Verificación: ✅ Correcto
```

🎯 Conclusiones

Logros

1. ✅ Implementación correcta de Dijkstra
2. ✅ Uso eficiente de cola de prioridad
3. ✅ Reconstrucción de caminos óptimos
4. ✅ Complejidad O((V+E) log V)

Ventajas del Algoritmo

- Encuentra caminos óptimos garantizados
- Eficiente para grafos con pesos positivos
- Ampliamente usado en aplicaciones reales
- Base para algoritmos más avanzados (A*)

Limitaciones

- No funciona con pesos negativos
- Requiere cola de prioridad eficiente
- Calcula desde un solo origen

Próximos Pasos

- Semana 7: Árboles binarios de búsqueda
- Algoritmo A* (Dijkstra con heurística)
- Bellman-Ford (pesos negativos)

📚 Referencias

- Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs"
- Cormen, T. H., et al. (2009). *Introduction to Algorithms* (3rd ed.)
- Material del curso - Semana 6

Fecha: Diciembre 2025
Curso: Estructuras de Datos Avanzadas
Semana: 6 - Dijkstra
Estado: ✅ Completado