Guía Detallada: Rutas Óptimas en Grafos

Melissa Robles

10 de marzo de 2025

Índice

1. Introducción y Definiciones Básicas **2**
1.1. Grafos, Vértices y Arcos . . . 2
1.2. Caminos y Costos . . . 2
1.3. Matriz de Distancias . . . 2

2. Problema de Rutas de Costo Mínimo **2**

3. Algoritmo de Floyd-Warshall **3**
3.1. Conceptos y Recurrencia . . . 3
3.2. Pseudocódigo . . . 3
3.3. Complejidad . . . 3
3.4. Recuperación de Rutas . . . 3

4. Algoritmo de Dijkstra **4**
4.1. Idea General . . . 4
4.2. Pseudocódigo . . . 4
4.3. Complejidad y Variantes . . . 4
4.4. Recuperación de Rutas . . . 4

5. Generalización: Problema de Caminos en Semianillos **5**
5.1. Definición y Ejemplos . . . 5
5.2. Clausura Matricial . . . 5
5.3. Adaptación del Algoritmo de Floyd-Warshall . . . 5

6. Ejercicios **5**

7. Conclusión **7**

1. Introducción y Definiciones Básicas

1.1. Grafos, Vértices y Arcos

Un **grafo dirigido etiquetado** se representa como:

$$G(V, E, c)$$

donde:

- **V**: Conjunto de vértices o nodos (por ejemplo, $\{1, 2, \ldots, n\}$).
- **E**: Conjunto de arcos, definidos como pares ordenados (i, j).
- **c**: Función de costos que asigna a cada arco un valor (distancia, tiempo, etc.).

1.2. Caminos y Costos

- **Camino:** Una secuencia de vértices

$$r = \langle u_0, u_1, \ldots, u_m \rangle$$

  tal que para cada i, $(u_i, u_{i+1}) \in E$.
- **Costo del camino:** Se define como

$$\text{costo}(r) = \sum_{i=0}^{m-1} c(u_i, u_{i+1}).$$

1.3. Matriz de Distancias

El grafo se puede representar mediante una matriz $D = (d_{ij})$ definida por:

- $d(i, i) = 0$.
- $d(i, j) = c(i, j)$ si existe un arco de i a j.
- $d(i, j) = \infty$ si no existe conexión directa.

2. Problema de Rutas de Costo Mínimo

El objetivo es determinar el camino de menor costo entre cada par de nodos en el grafo. Para ello, se define la **matriz de distancias mínimas** D^*:

$$d^*_{ij} = \text{mín}\{\text{costo}(r) : r \text{ es un camino de } i \text{ a } j\}.$$

Este planteamiento abarca las siguientes variantes:

- **Single source shortest path:** Encontrar los caminos mínimos desde un nodo fuente s a todos los demás.
- **Single pair shortest path:** Encontrar el camino mínimo entre dos nodos específicos.
- **All-pairs shortest path:** Calcular d^*_{ij} para todo i, j.

Además, es importante recuperar la secuencia de vértices que conforma cada ruta óptima.

3. Algoritmo de Floyd-Warshall

El algoritmo de Floyd-Warshall resuelve el problema **all-pairs shortest path** usando programación dinámica.

3.1. Conceptos y Recurrencia

Definimos $C^k(i,j)$ como el costo mínimo para ir de i a j utilizando únicamente los nodos $\{1,2,\ldots,k\}$ como intermedios:

- **Caso base** : $C^0(i,j) = d(i,j)$.
- **Recurrencia** : Para $k \geq 1$,

$$C^k(i,j) = \text{mín}\{C^{k-1}(i,j),\ C^{k-1}(i,k) + C^{k-1}(k,j)\}.$$

3.2. Pseudocódigo

A continuación se muestra una versión con bucles `for`:

```
proc FW(var m: array[1..n, 1..n] of R*)
   // Pre: m contiene la matriz de costos directos D.
   // Pos: m se convierte en la matriz de distancias mínimas D*.
   for k := 1 to n do
      for i := 1 to n do
         for j := 1 to n do
            m[i,j] := min{ m[i,j], m[i,k] + m[k,j] }
         end for
      end for
   end for
end proc
```

3.3. Complejidad

- **Tiempo:** $O(n^3)$ (debido a los tres bucles anidados).
- **Espacio:** $O(n^2)$ (por la matriz m).

3.4. Recuperación de Rutas

Para reconstruir la secuencia de vértices del camino óptimo, se introduce una matriz auxiliar (por ejemplo, `imd`):

- **Inicialización:** Si $(i,j) \in E$, se asigna un valor (por ejemplo, 0) indicando que no se usó nodo intermedio; si no existe arco, se asigna -1.
- **Actualización:** Cada vez que se mejora $m[i,j]$ mediante k, se registra k en la matriz.
- **Reconstrucción:** Si `imd[i,j]` es 0, el camino es directo; si es k, el camino se reconstruye dividiéndolo en los subcaminos de i a k y de k a j.

4. Algoritmo de Dijkstra

El algoritmo de Dijkstra resuelve el problema **single source shortest path** en grafos con pesos positivos.

4.1. Idea General

- **Inicialización:** Se define un arreglo d con $d[s] = 0$ y $d[v] = \infty$ para $v \neq s$.
- **Conjunto de nodos procesados:** Se utiliza un conjunto M para marcar los nodos cuya distancia mínima ya se ha determinado.
- **Iteración:** Mientras existan nodos no procesados, se selecciona el nodo w con la mínima distancia $d[w]$ y se actualizan los costos de sus vecinos.

4.2. Pseudocódigo

```
proc Dijkstra(G(V,E,c): grafo, s: V; var d: array[1..n] of R*)
   // Inicialización: d[s] = 0, d[v] = c(s,v) para cada v  s.
   M := {s}
   for each v in V do
      d[v] := c(s,v)
   end for

   while M  V do
      w := Escoja_min(d, V \ M)
      M := M  {w}
      for each v in V \ M such that (w,v) in E do
         d[v] := min{ d[v], d[w] + c(w,v) }
      end for
   end while
end proc
```

4.3. Complejidad y Variantes

- **Implementación básica:** Con un arreglo, la operación de `Escoja_min` cuesta $O(n)$, dando una complejidad total de $O(n^2)$.
- **Implementación con montículos (heap):** Al usar una cola de prioridad, las operaciones de inserción y extracción se realizan en $O(\log n)$ y la selección del mínimo en $O(1)$, reduciendo la complejidad a $O(n \log n + e)$ en grafos poco densos.

4.4. Recuperación de Rutas

Para obtener la ruta completa además del costo mínimo, se utiliza un arreglo de predecesores:

$$pred[v] = \text{nodo que actualizó } d[v]$$

Posteriormente, la ruta de s a v se reconstruye retrocediendo desde v hasta s.

5. Generalización: Problema de Caminos en Semianillos

El problema de rutas se puede generalizar a contextos en los que los costos pertenecen a un **semianillo** $(E, \oplus, \otimes, 0, 1)$.

5.1. Definición y Ejemplos

- Un **semianillo** es una estructura algebraica donde:
  - $\oplus$ es una operación (por ejemplo, mín o máx).
  - $\otimes$ es la operación que combina los costos (por ejemplo, suma o mínimo).
  - 0 y 1 son los elementos neutros para $\oplus$ y $\otimes$, respectivamente.
- Ejemplo clásico: $(\mathbb{R}^*, \text{mín}, +, \infty, 0)$ para rutas de costo mínimo.
- Ejemplo de capacidad en redes: $(\mathbb{R}^*, \text{máx}, \text{mín}, 0, \infty)$, donde el "costo" de un camino es el mínimo de las capacidades a lo largo del mismo.

5.2. Clausura Matricial

Se define la **clausura** de la matriz C como:

$$C^*_{ij} = \bigoplus_{k \geq 0} C^k_{ij},$$

donde C^k es la k-ésima potencia de C en el semianillo, generalizando así el concepto de caminos óptimos.

5.3. Adaptación del Algoritmo de Floyd-Warshall

El algoritmo se adapta sustituyendo:

- mín $\rightarrow \oplus$.
- $+ \rightarrow \otimes$.

La estructura del algoritmo se mantiene, pero se opera sobre los elementos del semianillo.

6. Ejercicios

A continuación, se proponen ejercicios para poner en práctica los conceptos y algoritmos vistos:

Ejercicio 1: Matriz de Distancias y Costo Mínimo

a. Dado un grafo $G(V, E, c)$ con nodos y arcos definidos, construye la matriz de distancias directas D.

b. Aplica la recursión de Floyd-Warshall para calcular la matriz de distancias mínimas D^*.

c. Determina el costo mínimo del camino desde el nodo 1 hasta el nodo 9.

Ejercicio 2: Corrección del Algoritmo de Floyd-Warshall

Demuestra, usando inducción o argumentos por contradicción, que la recursión:

$$C^k(i,j) = \text{mín}\{C^{k-1}(i,j),\ C^{k-1}(i,k) + C^{k-1}(k,j)\}$$

calcula correctamente el costo mínimo entre cada par de nodos i y j.

Ejercicio 3: Limitaciones de Dijkstra con Pesos Negativos

- Explica por qué Dijkstra no es adecuado cuando existen arcos con pesos negativos.
- Describe las condiciones en las que se puede resolver el problema (por ejemplo, ausencia de ciclos negativos) y menciona alternativas como el algoritmo de Bellman-Ford.

Ejercicio 4: Dijkstra para Camino Único

Modifica el algoritmo de Dijkstra para que encuentre únicamente el costo mínimo entre dos vértices específicos, en lugar de calcular los caminos mínimos desde la fuente a todos los nodos. Analiza cómo afecta esto a la complejidad.

Ejercicio 5: Recuperación de Rutas en Dijkstra

Completa la implementación de Dijkstra para que, además de los costos mínimos, se almacene la información (por ejemplo, arreglo de predecesores) necesaria para reconstruir la ruta óptima desde la fuente. Discute el impacto en la complejidad de espacio y tiempo.

Ejercicio 6: Optimización Multiobjetivo en Redes de Transporte

Una agencia de turismo dispone de datos de viajes directos entre n ciudades, incluyendo:

- Distancia a recorrer.
- Precio del pasaje.
- Duración del viaje.

Diseña un algoritmo para determinar rutas óptimas considerando cada uno de estos criterios (por ejemplo, de forma independiente o mediante técnicas de optimización multiobjetivo). Estima la complejidad temporal y espacial de la solución.

Ejercicio 7: Determinación de Nodos Alcanzables

Desarrolla un algoritmo basado en las ideas de Dijkstra que determine los nodos alcanzables desde un nodo dado en un grafo dirigido. Justifica la corrección y analiza la complejidad.

7. Conclusión

En este documento se han abordado los siguientes aspectos:

- **Fundamentos de grafos:** Definición de nodos, arcos, caminos y representación mediante matrices.
- **Problema de rutas de costo mínimo:** Modelado y variantes (single source, single pair, all-pairs).
- **Algoritmo de Floyd-Warshall:** Programación dinámica para calcular distancias mínimas, con recursión, pseudocódigo, complejidad y recuperación de rutas.
- **Algoritmo de Dijkstra:** Resolución del problema de fuente única en grafos con pesos positivos, variantes y recuperación de rutas.
- **Generalización a semianillos:** Extensión del problema para otros contextos algebraicos.
- **Ejercicios:** Problemas prácticos y teóricos para consolidar el aprendizaje.

Esta guía integral y detallada ofrece una base sólida para comprender y aplicar algoritmos de rutas óptimas en grafos, tanto en su forma clásica como en sus variantes y generalizaciones, facilitando su aplicación en áreas como transporte, redes de comunicación y optimización en general.