

5 RUTA MÁS CORTA EN GRAFOS

Una importante aplicación de las técnicas de programación dinámica es la determinación de rutas de costo mínimo, en grafos. Este problema, así como otros relacionados, conforman una clase de problemas con muchas situaciones reales en las que el estudio teórico puede aplicarse. Para definir los términos del problema:

Sea $G(V, E, c)$ un grafo dirigido etiquetado, i.e.,

- V es un conjunto de *vértices* o *nodos*,
- $E \subseteq V \times V$ es un conjunto de *arcos*,
- $c: E \rightarrow \mathbf{R}^*$ es una función que *etiqueta* los arcos ¹, i.e., para $e \in E$, $c(e)$ es la *etiqueta* de e .

Es usual que el grafo $G(V, E, c)$ sea finito, es decir, que el conjunto de vértices V lo sea. Como convención para las discusiones siguientes, se supondrá que $V = \{1, 2, \dots, n\}$. La etiqueta de un arco representa una especie de *costo* de la transición entre los vértices que se conectan.

Un *camino de longitud* m , desde el vértice u_0 hasta el vértice u_m es una secuencia

$$r = \langle u_0, u_1, \dots, u_m \rangle \in V^+$$

tal que, para $i=0, \dots, m-1$, $(u_i, u_{i+1}) \in E$. En este caso, se define el *costo* del camino r como:

$$\text{costo}(r) := \langle +: 0 \leq i < m : c(u_i, u_{i+1}) \rangle.$$

El grafo $G(V, E, c)$ se puede representar con una *matriz de distancias (directas)* $D = (d_{ij}) \in M_n(\mathbf{R}^*)$, definida de modo que, para $1 \leq i, j \leq n$:

$$\begin{aligned} d(i, i) &:= 0, \\ d(i, j) &:= c(i, j) \quad , \text{ si } (i, j) \in E, \\ d(i, j) &:= \infty \quad , \text{ si } (i, j) \notin E \end{aligned}$$

La representación es tal que se olvida el costo de las "bucclas" o "lazos", i.e., si hay un arco con $c(i, i) > 0$, la representación lo "cambia" por otro de costo 0. La razón para esta decisión es el interés en determinar costos mínimos de caminos entre vértices; de esta forma, un camino que

¹ $\mathbf{R}^* := \mathbf{R}^+ \cup \{0, \infty\}$

utilice un lazo de costo positivo siempre tiene costo mayor que otro que no lo usa. Por otro lado, los vértices originalmente no conectados, se consideran ligados por un arco de costo infinito.

Ya en el Ejemplo 4 de 3.5.1 se mostró que, a partir de multiplicaciones de matrices $M_n(\mathbb{R}^*)$ se puede determinar la matriz $D^* = (d_{ij}^*)$, donde d_{ij}^* es, precisamente, el costo de un camino de costo mínimo del vértice i al vértice j . Entonces, D^* se llama la *matriz de distancias mínimas*.

Con esta notación, se plantean los siguientes problemas:

- Encontrar D^* .
- Dado un vértice u , encontrar $d_{u,\cdot}^*$, i.e., d_{uv}^* , para $v \in V$.
- Dados dos vértices u, v , encontrar d_{uv}^* .

Es claro que cualquier solución del problema de encontrar la matriz de distancias mínimas resuelve los otros dos problemas y que, resolver el problema de costos mínimos de un vértice a todos los demás conlleva una solución para encontrar el costo mínimo entre dos vértices. Por el contrario, es fácil construir soluciones de los problemas más complejos a partir de soluciones para los simples.

En cada problema planteado, resulta interesante, además de conocer el costo mínimo correspondiente, conocer la forma en que se consigue, es decir, determinar caminos de costo mínimo.

5.1 RUTA MÁS CORTA ENTRE CADA PAR DE VÉRTICES

Se quiere solucionar el problema de calcular D^* , es decir, determinar, para $1 \leq i, j \leq n$, el valor de

$$d_{ij}^* = \langle \min: r \text{ es un camino de } i \text{ a } j: \text{costo}(r) \rangle$$

Una especificación del problema para una solución algorítmica:

Ctx: $m: \text{array}[1..n, 1..n] \text{ of } \mathbb{R}^*$
 Pre: $m = D$
 Pos: $m = D^*$

La búsqueda de una solución puede afrontarse con técnicas de programación dinámica. En primer lugar, conviene definir el siguiente léxico:

Para $w \subseteq V, i, j \in V$:

$R_{ij}^w := \{r \mid r = \langle i, w_1, w_2, \dots, w_m, j \rangle \text{ camino, para } k=1, \dots, m: w_k \in w\}$
 $C_{ij}^w := \langle \min: r \in R_{ij}^w : \text{costo}(r) \rangle$

De esta manera, para $1 \leq i, j \leq n$:

$$d_{ij}^* = C_{ij}^V.$$

Entonces se puede establecer una recurrencia para calcular los $C_{ij}^w, w \subseteq V, i, j \in V$:

$$C_{ij}^{\emptyset} = D_{ij}$$

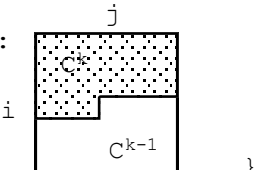
$$C_{ij}^{W \cup \{v\}} = \min\{C_{ij}^W, C_{iv}^W + C_{vj}^W\}, \text{ para } v \notin W$$

En particular, si ahora se denota $C_{ij}^{1..k}$ como C_{ij}^k , se tiene que:

$$C_{ij}^k = \begin{cases} D_{ij} & , \text{ si } k=0 \\ \min\{C_{ij}^{k-1}, C_{ik}^{k-1} + C_{kj}^{k-1}\} & , \text{ si } 0 < k \leq n \end{cases}$$

Esta recurrencia es la base fundamental para establecer la corrección del llamado algoritmo de Floyd-Warshall:

```

proc FW (var m:array[1..n,1..n] of R*)
{Pre:   m = D}
{Pos:   m = D*}
[ k:= 0;
  {Inv P: 0≤k≤n ∧ m = Ck }
  do k≠n → i,j,k:= 1,1,k+1;
    {Inv P1: 1≤i≤n+1 ∧ 1≤j≤n ∧ m:
      
    }
    do i≠n+1
      → m[i,j]:= min {m[i,j], m[i,k]+m[k,j]} ;
      if j≠n → j:= j+1
      [] j=n → i,j:= i+1,1
      fi
    od
  od
]

```

Para comprobar que los invariantes se conservan, debe notarse que:

$$C_{ik}^{1..k} = C_{ik}^{1..k-1}, \quad C_{kj}^{1..k} = C_{kj}^{1..k-1}$$

o bien:

$$C_{ik}^k = C_{ik}^{k-1}, \quad C_{kj}^k = C_{kj}^{k-1},$$

de manera que la instrucción

$$m[i,j] := \min\{m[i,j], m[i,k]+m[k,j]\};$$

sí efectúa el cálculo que se requiere.

Una versión abreviada, utilizando "for":

```

proc FW (var m:array[1..n,1..n] of R*)

```

```

{Pre:      m = D}
{Pos:      m = D*}
[  for k:=1 to n →
    for i:=1 to n →
        for j:=1 to n →  m[i,j] := min {m[i,j],m[i,k]+m[k,j]}  rof
    rof
  rof
]

```

Complejidades:

$$T_{FW}(n) = O(n^3)$$

$$S_{FW}(n) = O(n^2).$$

5.1.1 Determinación de rutas de costo mínimo

El algoritmo de Floyd-Warshall puede "decorarse" para resolver el problema relacionado de determinar una ruta de costo mínimo entre cada par de nodos.

Para esto se puede proceder, siguiendo con los métodos de la programación dinámica, recordando las decisiones que debieron tomarse en el cálculo de los costos mínimos. De este modo, se puede decidir, al construir un camino de costo mínimo, si se debe o no utilizar cada nodo intermedio considerado.

Esta idea se plasma al mantener una matriz $intmd$, de manera que, para $1 \leq i, j \leq n$:

$$\begin{aligned}
 intmd_{ij} &= v, \text{ si existe una ruta que pasa por } v \in V, \text{ con costo mínimo} \\
 &= 0, \text{ si la ruta más corta de } i \text{ a } j \text{ es el arco } (i, j) \in E \\
 &= -1, \text{ si no hay camino de } i \text{ a } j.
 \end{aligned}$$

Más aun, se pueden definir los valores:

$$\begin{aligned}
 intmd_{ij}^k &= v, \text{ si existe una ruta que pasa por } v \in V, \text{ con costo mínimo, cuyos nodos intermedios} \\
 &\text{están en } \{1, \dots, k\} \\
 &= 0, \text{ si la ruta más corta de } i \text{ a } j, \text{ es el arco } (i, j) \in E \\
 &= -1, \text{ si no hay caminos de } i \text{ a } j, \text{ cuyos nodos intermedios están en } \{1, \dots, k\}.
 \end{aligned}$$

Naturalmente:

$$intmd_{ij}^n = intmd_{ij}.$$

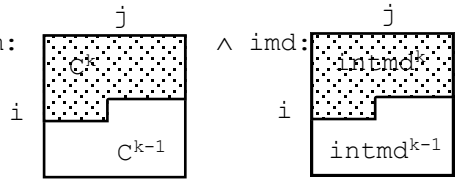
Entonces se pueden modificar los algoritmos para que calculen los nodos intermedios de las rutas óptimas:

```

proc FWropt (var  m:array[1..n,1..n] of R*,
               imd:array[1..n,1..n] of V ∪ {0,-1})
{Pre:      m = D}
{Pos:      m = D* ∧ imd = intmd }

```

```

[ k:= 0;
  for 1≤i,j≤n → if m[i,j]≠∞ → imd[i,j]:= 0
                [] m[i,j]=∞ → imd[i,j]:= -1
                fi
  rof;
  {Inv P: 0≤k≤n ∧ m = Ck ∧ imd = intmdk}
  do k≠n → i,j,k:= 1,1,k+1;
    {Inv P1: 1≤i≤n+1 ∧ 1≤j≤n ∧ m:
      
      }
    do i≠n+1
      → if m[i,j] > m[i,k]+m[k,j]
          then m[i,j]:= m[i,k]+m[k,j];
              imd[i,j]:= k
          fi
      if j≠n → j:= j+1
      [] j=n → i,j:= i+1,1
      fi
    od
  od
]

```

Las complejidades ahora son, de nuevo:

$$T_{FWopt}(n) = O(n^3)$$

$$S_{FWopt}(n) = O(n^2)$$

Sin embargo, el algoritmo anterior apenas calcula la estructura imd , con la que se pueden recuperar las rutas óptimas entre nodos, ya que, si se quiere calcular una ruta óptima entre i y j :

$$\begin{aligned}
 ropt_{ij} &= \langle j \rangle && , \text{ si } intmd_{ij} = 0 \\
 &= \perp && , \text{ si } intmd_{ij} = -1 \\
 &= ropt_{ik} \ \& \ ropt_{kj}, && \text{ si } intmd[i,j]=k>0
 \end{aligned}$$

Un algoritmo que calcule rutas óptimas utilizando esta idea, tendrá una complejidad, en el cálculo de la ruta óptima entre cualquier par de nodos:

$$T_{ropt}(n) = O(n)$$

$$S_{ropt}(n) = O(1).$$

5.2 EL ALGORITMO DE DIJKSTRA

Una variante del problema de determinar todas las distancias mínimas consiste en determinar, para un vértice fijo f , llamado la *fuentes*, las distancias a todos los demás vértices. En otras palabras, determinar d_{fv}^* , para $v \in V$, o bien, el vector D_f^* .

Una manera de solucionar el problema consiste en ejecutar el procedimiento de Floyd-Warshall y quedarse con la fila f de la matriz respuesta. Como se verá, este procedimiento ingenuo resulta en una complejidad temporal que puede ser mejorada sustancialmente.

El algoritmo de Dijkstra resuelve el problema de la siguiente manera (la semántica del procedimiento `Escoja_min` se explica con la aserción $Q1$):

```

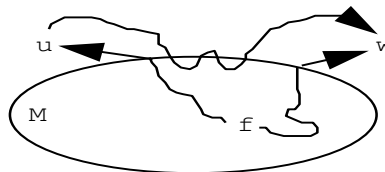
proc Dijkstra (G(V,E,c): grafo, f: V; var df:array[1..n] of R* )
  {Pos:      df = D_f^* }
  [ M:= {f};
    for v∈V    →  df[v] := c[f,v] rof;
    {Inv P:  ⟨∀v: v∈M : df[v]=D_{fv}^*⟩ ∧ ⟨∀v: v∈V\M : df[v]=C_{fv}^M⟩ }
    do M≠V →  w:= Escoja_min(df,V\M);
              {Q1: P ∧ df[w]≤df[V\M] ∧ w∈V\M }
              M:= M∪{w};
              for v ∈ V\M
                  →  df[v] := min{df[v],df[w]+c[w,v]}
              rof
    od
  ]
  
```

El invariante se conserva:

- Como el w elegido en cada iteración es un nuevo miembro del conjunto M , supóngase que no fuera cierto que

$$df[w] = D_{fw}^*$$

En este caso, debería existir un camino de costo óptimo de f a w , cuyos nodos intermedios no pueden estar todos en M (puesto que $df[w] = C_{fw}^M$). Sea u el primer nodo en este camino que no está en M . La situación se puede visualizar así:



Entonces:

$$\begin{aligned}
 D_{fw}^* &= D_{fu}^* + D_{uw}^* \\
 &= D_{fu}^M + D_{uw}^* && \text{, porque los antecesores de } u \text{ están en } M \\
 &= df[u] + D_{uw}^* && \text{, por el invariante}
 \end{aligned}$$

$< df[w]$, porque se supone que $df[w]$ no es óptimo
 $\leq df[u]$, porque w se escoge con $df[.]$ minimal.

Y se concluiría que $D_{uw}^* < 0$, lo cual es contradictorio.

- Por otra parte, para cada nodo en el nuevo $V \setminus M$, el último ciclo actualiza los valores para satisfacer el invariante.

El conjunto M se llama así porque "marca" los nodos para los que ya se conoce el óptimo. A $V \setminus M$ se le llama el conjunto de los "abiertos". Una forma de expresar el algoritmo con una estructura de datos que administre los abiertos es:

```

proc Dijkstra (G(V,E,c): grafo, f: V; var df:array[1..n] of R* )
{Pos:      df = Df*. }
[  A:= V\{f};
  for v∈V    →  df[v]:= c[f,v]    rof;
  do A≠∅    →  w:= Escoja_min(df,A);
                A:= A\{w};
                for v∈A ∧ (w,v)∈E
                    →  df[v]:= min{df[v],df[w]+c[w,v]}
                rof
  od
]
  
```

5.2.1 La complejidad del algoritmo de Dijkstra

El algoritmo de Dijkstra se puede considerar formado por la secuenciación de una inicialización

```

INIC:      A:= V\{f};
           for v∈V →  df[v]:= d[f,v]  rof;
  
```

y de un ciclo:

```

CICLO:    do A≠∅ →  w:= Escoja_min(df,A);
                A:= A\{w};
                for v∈A ∧ (w,v)∈E
                    →  df[v]:= min{df[v],df[w]+c[w,v]}
                rof
  od
  
```

El tamaño del problema es n , el número de vértices del grafo. Por otra parte, sea $e = |E|$ el número de arcos del grafo².

² Obsérvese que $0 \leq e \leq n^2$, de modo que $e = O(n^2)$.

Como operaciones básicas, considérense:

- # : añadir un elemento de un conjunto
- \ : eliminar un elemento de un conjunto
- Escoja_min : escoger el elemento de un conjunto que alcanza el mínimo de una medida
- min : calcular el mínimo de dos números.

Así:

$$\begin{aligned}T_D(n) &= T(\text{INIC}) + T(\text{CICLO}) \\T(\text{INIC}) &= O(n) T(\#) \\T(\text{CICLO}) &= O(n) [T(\text{Escoja_min}) + T(\backslash)] + O(e) T(\text{min})\end{aligned}$$

Es decir:

$$T_D(n) = O(n) [T(\#) + T(\text{Escoja_min}) + T(\backslash)] + O(e) T(\text{min})$$

La complejidad definitiva depende de las estructuras de datos que se utilicen. En otras palabras, ya que es concebible representar tanto el grafo $G(V, E, c)$ como el conjunto de abiertos A con diversas estructuras de datos, el uso de recursos dependerá de la conveniencia de estas estructuras para realizar el algoritmo.

A continuación se muestran tres variantes de representación que arrojan complejidades diferentes para las ejecuciones del algoritmo de Dijkstra:

Variante 1:

- $G \approx$ matriz de distancias directas (adyacencias etiquetadas con costos): $O(n^2)$.
- $A \approx$ arreglo booleano: $O(n)$.

Como costo de las operaciones básicas, se supone:

- $T(\#) = O(1)$
- $T(\backslash) = O(1)$
- $T(\text{Escoja_min}) = O(n)$
- $T(\text{min}) = O(1)$

Entonces:

$$\begin{aligned}T_D(n) &= O(n) [O(1) + O(n) + O(1)] + O(e) O(1) \\&= O(n^2 + e) \\&= O(n^2).\end{aligned}$$

Variante 2:

- $G \approx$ lista de distancias directas: $O(n^2)$.
- $A \approx$ arreglo booleano: $O(n)$.

Como costo de las operaciones básicas, se supone:

- $T(\#) = O(1)$

- $T(\backslash)$ = $O(1)$
- $T(\text{Escoja_min})$ = $O(n)$
- $T(\text{min})$ = $O(1)$

El análisis es idéntico al del caso anterior:

$$T_D(n) = O(n^2)$$

Variante 3:

- $G \approx$ lista de distancias directas: $O(n^2)$.
- $A \approx$ montón (inglés: *heap*): $O(n)$.

Como costo de las operaciones básicas, se supone:

- $T(\#)$ = $O(\log n)$
- $T(\backslash)$ = $O(\log n)$
- $T(\text{Escoja_min})$ = $O(1)$
- $T(\text{min})$ = $O(1)$

$$\begin{aligned} T_D(n) &= O(n) [O(\log n) + O(1) + O(\log n)] + O(e) O(1) \\ &= O(n \log n + e) \end{aligned}$$

Ahora, si el grafo $G(V, E, c)$ no tiene "demasiados arcos", i.e., si $e \leq n \log n$, se tendrá que:

$$T_D(n) = O(n \log n).$$

Este caso, de grafos con pocos arcos, es frecuente en la práctica. Es decir, se tiene una variante "práctica" del algoritmo de Dijkstra, mejor que $O(n^2)$.

5.2.2 La implantación con montones

La Variante 3 de implantación del algoritmo de Dijkstra que se analizó en la sección anterior supone la utilización de una estructura de datos llamada *montón* (inglés: *heap*) para manejar el conjunto de abiertos. En la literatura también se referencia esta estructura como una *cola de prioridad*.

Un montón sirve para representar conjuntos ordenados totalmente, de suerte que es posible realizar las siguientes operaciones y a los costos anotados:

- $T(\#)$ = $O(\log n)$: agregar un elemento
- $T(\backslash)$ = $O(\log n)$: eliminar un elemento
- $T(\text{min})$ = $O(1)$: calcular el elemento mínimo

Para lograr este desempeño se emplea un árbol binario cuyos vértices corresponden a los elementos del conjunto que se representa y que, además, tiene las propiedades de *forma* y *orden*.

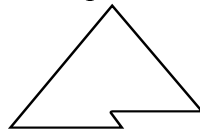
La propiedad de *forma*, consiste en que

- la diferencia entre las alturas de dos hojas del árbol es 0 ó 1.
- las hojas más profundas están en las ramas más a la izquierda.
- todo nodo tiene 0 ó 2 hijos, excepto quizás el padre de la hoja más profunda más a la derecha, que puede tener sólo un hijo.

Algunos árboles con la propiedad de forma:



Un árbol con la propiedad de forma es casi completo, excepto que, en el último nivel hay hojas de la altura del árbol -aquellas que están más a la izquierda- o de altura inferior en uno a la del árbol. Esquemáticamente, la forma de estos árboles se puede visualizar así:



Nótese que si el montón representa un conjunto de n elementos, su altura es $O(\log n)$.

La propiedad de *orden* consiste en que, o bien el árbol es vacío, o su raíz es mayor que todos los elementos que están bajo ella y, además, tanto el hijo izquierdo como el derecho también satisfacen la propiedad de orden.

Ahora se puede justificar el hecho de que las operaciones de añadir, eliminar y calcular mínimo tengan los costos ya anotados:

- Para calcular el mínimo, basta mirar la raíz del montón. Así:

$$T(\min) = O(1).$$

- Para añadir un elemento, se coloca en el puesto que le correspondería si se quisiera mantener la propiedad de forma. Después, el elemento se "aspira" hacia arriba, intercambiándolo sucesivamente con su padre, manteniendo la propiedad de forma, hasta que se cumpla la propiedad de orden. Gráficamente:

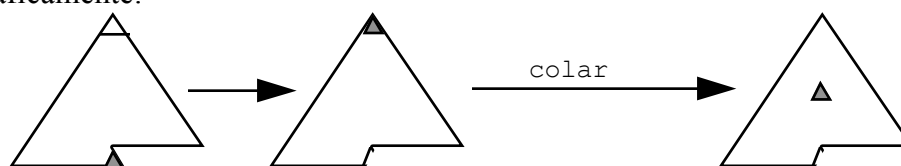


Así:

$$T(\#) = O(\log n)$$

ya que el número de intercambios del nuevo valor con sus padres interinos puede ser, a lo sumo, la altura del árbol, i.e., $\log n$.

- Para eliminar el elemento mínimo, se quita la raíz, y se coloca en su puesto el elemento más a la derecha de los que están en el nivel más profundo. Esto mantiene la propiedad de forma. Entonces se "cuela" hacia abajo la raíz, intercambiándola sucesivamente con el mayor de sus dos hijos si uno de ellos fuera mayor. Cuando esto deje de suceder se tendrá la propiedad de orden. Gráficamente:

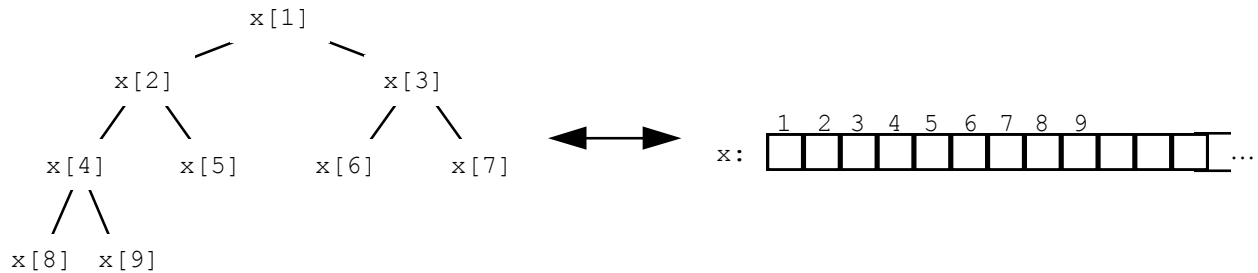


Por tanto, eliminar el elemento mínimo demanda $O(\log n)$ pasos, puesto que la labor de colar puede tomar, a lo sumo, $O(\log n)$ pasos. Un argumento similar sirve para mostrar que cualquier elemento se puede eliminar en un número de pasos del mismo orden, es decir,

$$T(\lambda) = O(\log n).$$

5.2.3 Implantación de los montones

Un montón puede ser representado muy eficientemente con un arreglo lineal, con la correspondencia sugerida por la figura siguiente:



Más formalmente:

Valor nodo i	:	$x[i]$
Raíz	:	1
Hijo izquierdo de i	:	$2i$
Hijo derecho de i	:	$2i+1$
Padre de i	:	$\lfloor \frac{i}{2} \rfloor$
i no tiene hijos	:	$2i > n$
i tiene un hijo	:	$2i = n$
i tiene dos hijos	:	$2i < n$

Con esta representación se alcanzan los costos de aspirar y colar ya analizados.

5.3 EL PROBLEMA GENERAL DE CAMINOS

El problema de la ruta más corta se puede generalizar cuando se considera que las etiquetas del grafo $G(V, E, c)$ toman valores en un semianillo $(E, \oplus, \otimes, 0, 1)$. Más aun, se supondrá que el semianillo E es cerrado³.

Para el planteo del problema original, las etiquetas toman valores en el semianillo $(\mathbf{R}^*, \min, +, \infty, 0)$. Cuando un arco entre dos nodos u, v no existe, se entiende que $c(u, v) = \infty$. Además, se espera que $c(u, u) = 0$.

En la generalización debe pensarse que no todos los arcos deben existir. Naturalmente, la generalización induce la necesidad de considerar que se tiene $c(u, v) = 0$, el módulo de \oplus , cuando $(u, v) \notin E$. Adicionalmente, debe tenerse que $c(u, u) = 1$, el módulo de \otimes .

Para un camino r de la forma

$$r = \langle u_0, u_1, \dots, u_{m-1} \rangle$$

se define el *costo* como:

$$\text{costo}(r) := \langle \otimes : 0 \leq i < m : c(u_i, u_{i+1}) \rangle.$$

Para $i, j \in V$, se define P_{ij} como el conjunto de todos los caminos de i a j .

El *problema general de caminos* se define como el de determinar, para $i, j \in V$, el valor

$$c_{ij}^* = \langle \oplus : p \in P_{ij} : \text{costo}(p) \rangle.$$

Ejemplo: capacidad de una red de transporte

Supóngase un grafo $G(V, E, c)$ en el que los arcos representan capacidades de transporte entre los nodos que unen. Por ejemplo, este es el caso de una red de carreteras, en la que las etiquetas corresponden al máximo peso que los puentes entre dos nodos pueden soportar. El problema es determinar la máxima carga que se puede enviar de un nodo fuente a un nodo sumidero.

Las etiquetas pueden considerarse en el semianillo⁴ $(\mathbf{R}^*, \max, \min, 0, \infty)$. Para un par de nodos u, v , que no tengan un arco que los conecte directamente, se debe pensar que $c(u, v) = 0$, i.e., la capacidad de transporte es 0.

Para un camino r de la forma

$$r = \langle u_0, u_1, \dots, u_{m-1} \rangle$$

se define el costo o *capacidad* como:

$$\text{cap}(r) := \langle \min : 0 \leq i < m : c(u_i, u_{i+1}) \rangle.$$

i.e., la capacidad de un camino está definida por la del menor de los tramos que lo componen.

³ Es decir, las sumas de un número enumerable de elementos están bien definidas, su valor no depende del orden de los sumandos y la multiplicación distribuye sobre estas sumas infinitas.

⁴ cf. Ejercicio 3.6.4.a.

Para este caso, la interpretación del problema general de caminos corresponde a encontrar, para $i, j \in V$, el valor

$$c_{ij}^* = \langle \max: p \in P_{ij} : \text{cap}(p) \rangle.$$

es decir, la máxima capacidad posible, considerando todas las maneras de conectar los nodos en cuestión.

||

5.3.1 El problema de caminos como una clausura matricial

Ya en el Capítulo 3 se mostró la fuerte relación que existe entre el problema de la multiplicación de matrices en el semianillo $(\mathbb{R}^*, \min, +, \infty, 0)$ y el problema original de encontrar costos de rutas mínimas. No es difícil mostrar que estos resultados se pueden generalizar fácilmente al problema de caminos definido en la sección anterior. Más exactamente, valen:

Teorema A

Para $k \geq 0$, sea C^k la k -ésima potencia de la matriz $C = (c_{ij}) \in M_n(\mathbb{E})$, definida de modo que, para $i, j \in V$:

$$\begin{aligned} c_{ij} &:= c(i, j), & \text{si } (i, j) \in E \\ &:= 0, & \text{si } (i, j) \notin E \end{aligned}$$

Entonces, para $i, j \in V$:

$$C_{ij}^k = \langle \oplus: p \in P_{ij}, |p|=k : \text{costo}(p) \rangle$$

||

Corolario B

Para $i, j \in V$:

$$C_{ij}^* = \langle \oplus: k \geq 0 : C_{ij}^k \rangle$$

||

El Corolario B muestra cómo la pregunta del problema generalizado de caminos corresponde, precisamente, al cálculo de la clausura de la matriz C en el semianillo $M_n(\mathbb{E})$.

5.3.2 El algoritmo de Floyd-Warshall generalizado

El algoritmo de Floyd-Warshall puede extenderse para solucionar el problema general de caminos. En el caso general debe considerarse la posibilidad de utilizar caminos que incluyan ciclos. El resultado se puede presentar de la siguiente manera:

```

proc FWg (var m: array[1..n, 1..n] of E)
  {Pre:      m = C}

```

```

{Pos:    m = C*}
[ for k:=1 to n →
  for i:=1 to n →
    for j:=1 to n
      → m[i,j] := m[i,j] ⊕ (m[i,k] ⊗ Ckk* ⊗ m[k,j]);
        m[k,j] := Ckk* ⊗ m[k,j]
    rof;
  m[i,k] := m[i,k] ⊗ Ckk*
rof
rof
]

```

Complejidades:

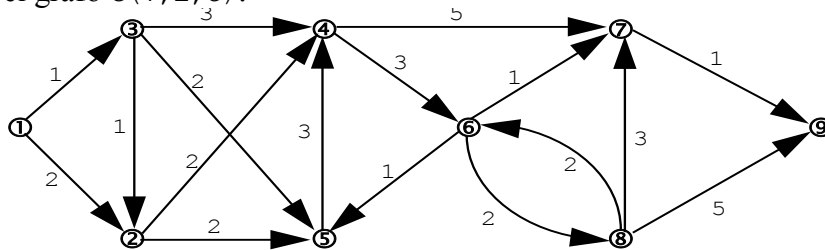
$$T_{FWG}(n) = O(n^3)$$

$$S_{FWG}(n) = O(n^2).$$

En muchas ocasiones se tiene que $C_{kk}^* = 1$, como en el caso del algoritmo original, el problema de conectividad o el problema de capacidad antes mencionado. Esta situación corresponde al hecho de que los ciclos no contribuyan a "mejorar los caminos".

5.4 EJERCICIOS

1 Considere el grafo $G(V, E, c)$:



- a Construya una matriz de distancias directas que represente a G .
 - b Determine el costo mínimo de un camino del nodo 1 al nodo 9.
- 2 Compruebe la corrección del algoritmo de Floyd-Warshall generalizado.
 - 3 Explique por qué el algoritmo de Dijkstra no resuelve el problema de la ruta mínima si se consideran arcos etiquetados con números negativos. Dé condiciones suficientes para que el problema tenga sentido en un grafo con costos tanto positivos como negativos y explique cómo debe modificarse el algoritmo para considerar este caso.
 - 4 Modifique el algoritmo de Dijkstra para que resuelva el problema -más simple- de averiguar el costo mínimo de un camino entre dos vértices.

- 5 Complete el algoritmo de Dijkstra para que, adicionalmente, calcule información necesaria para resolver el problema de encontrar rutas mínimas entre la fuente y los demás vértices. ¿Cuánto encarece las complejidades de espacio y tiempo el realizar este trabajo adicional?
- 6 Una agencia de turismo tiene información sobre posibilidades de viaje directo entre cada pareja de n ciudades. Para ir de manera directa de una ciudad A a una ciudad B , se conoce la distancia por recorrer, el precio del pasaje y la duración del viaje. Dadas dos ciudades, describa un algoritmo para calcular costos de rutas óptimas en cuanto a distancia recorrida, precio de pasajes y duración. Estime la complejidad temporal y espacial de su algoritmo.
- 7 Siga las ideas del algoritmo de Dijkstra para construir un algoritmo que determine qué nodos son alcanzables desde un vértice distinguido en un grafo dirigido $G(V, E)$.