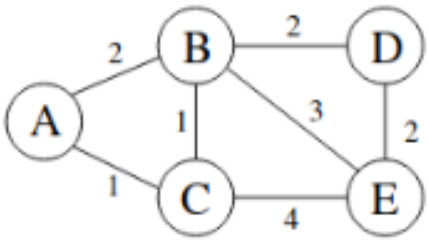
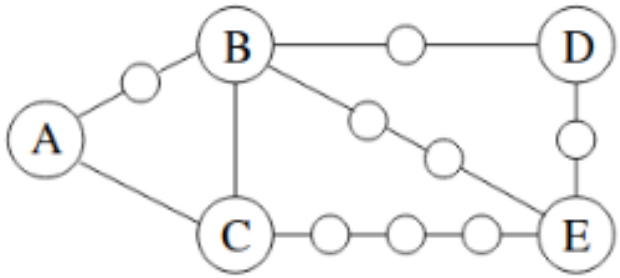
Analisis y Diseño de Algoritmos

Juan Gutiérrez

June 30, 2022

Caminos minimos





Caminos minimos

Recibe: Un grafo G con longitudes ℓ no negativas en las aristas y un vértice $s \in V(G)$. Modifica *dist* de manera que $dist(v)$ guarda la distancia de s a v en G.

DIJKSTRA(G, s)

1: **for** $v \in V(G)$
2: $\quad dist(v) = \infty$
3: $dist(s) = 0$
4: $R = \emptyset$
5: **while** $R \neq V(G)$
6: $\quad$ Sea $v \in V(G) \setminus R$ tal que $dist(v) = \min\{dist(w) : w \in V(G) \setminus R\}$
7: $\quad R = R \cup \{v\}$
8: $\quad$ **for** $vz \in E(G)$
9: $\quad\quad$ **if** $dist(v) + \ell_{vz} < dist(z)$
10: $\quad\quad\quad dist(z) = dist(v) + \ell_{vz}$

Caminos minimos

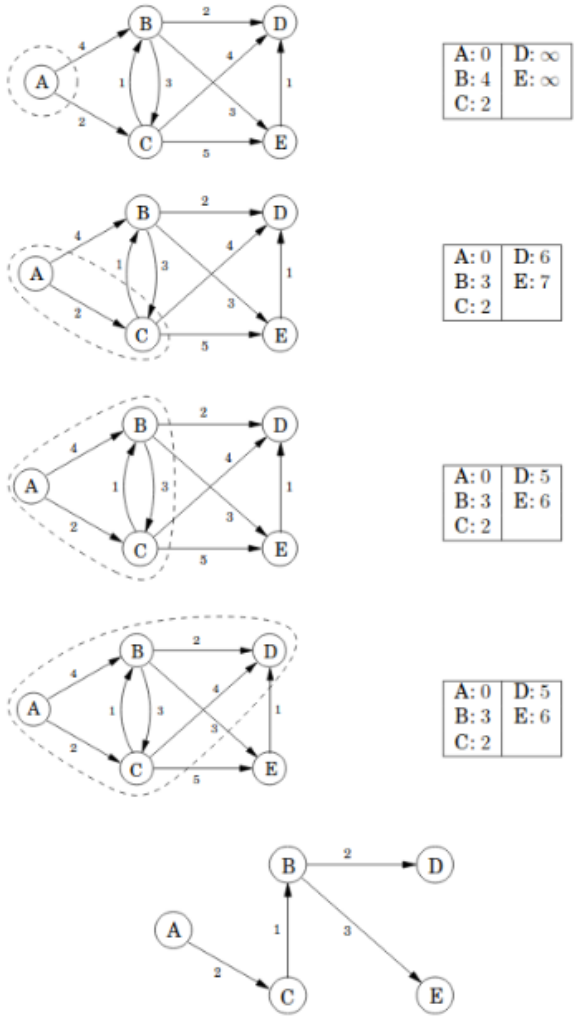


Figure 7: Tomada del libro Dasgupta et al, Algorithms

Caminos minimos

Recibe: Un grafo G con longitudes ℓ no negativas en las aristas y un vértice $s \in V(G)$. Modifica *dist* de manera que $dist(v)$ guarda la distancia de s a v en G.

DIJKSTRA-FP(G, s)

1: **for** $v \in V(G)$
2: $\quad dist(v) = \infty$
3: $dist(s) = 0$
4: $Q = V(G)$ //fila de prioidades inicializada
5: **while** $Q \neq \emptyset$
6: $\quad v =$ EXTRACT-MIN(Q)
7: $\quad$ **for** $vz \in E(G)$
8: $\quad\quad$ **if** $dist(v) + \ell_{vz} < dist(z)$
9: $\quad\quad\quad dist(z) = dist(v) + \ell_{vz}$
10: $\quad\quad\quad$ DECREASE-KEY(Q,z)

Caminos minimos

Implementation	`deletemin`	`insert`/ `decreasekey`	$\|V\|$ × `deletemin` + ($\|V\| + \|E\|$) × `insert`
Array	$O(\|V\|)$	$O(1)$	$O(\|V\|^2)$
Binary heap	$O(\log \|V\|)$	$O(\log \|V\|)$	$O((\|V\| + \|E\|) \log \|V\|)$
d-**ary heap**	$O(\frac{d \log \|V\|}{\log d})$	$O(\frac{\log \|V\|}{\log d})$	$O((\|V\| \cdot d + \|E\|) \frac{\log \|V\|}{\log d})$
Fibonacci heap	$O(\log \|V\|)$	$O(1)$ **(amortized)**	$O(\|V\| \log \|V\| + \|E\|)$

Caminos minimos

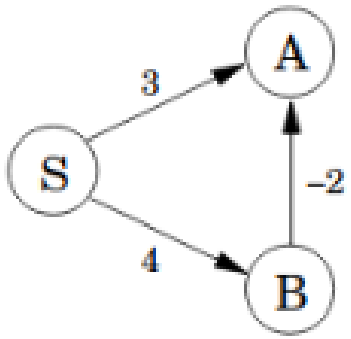


Figure 9: Tomada del libro Dasgupta et al, Algorithms

Caminos minimos

Recibe: Un grafo G con longitudes ℓ (positivas o negativas) en las aristas, **y sin ciclos negativos** y un vértice $s \in V(G)$. Modifica *dist* de manera que $dist(v)$ guarda la distancia de s a v en G.

BELLMAN-FORD(G, s)

1: **for** $v \in V(G)$
2: $\quad dist(v) = \infty$
3: $dist(s) = 0$
4: **for** $i = 1$ hasta $V|(G)| - 1$
5: $\quad$ **for** $uv \in E(G)$
6: $\quad\quad$ UPDATE(u, v)

Caminos minimos

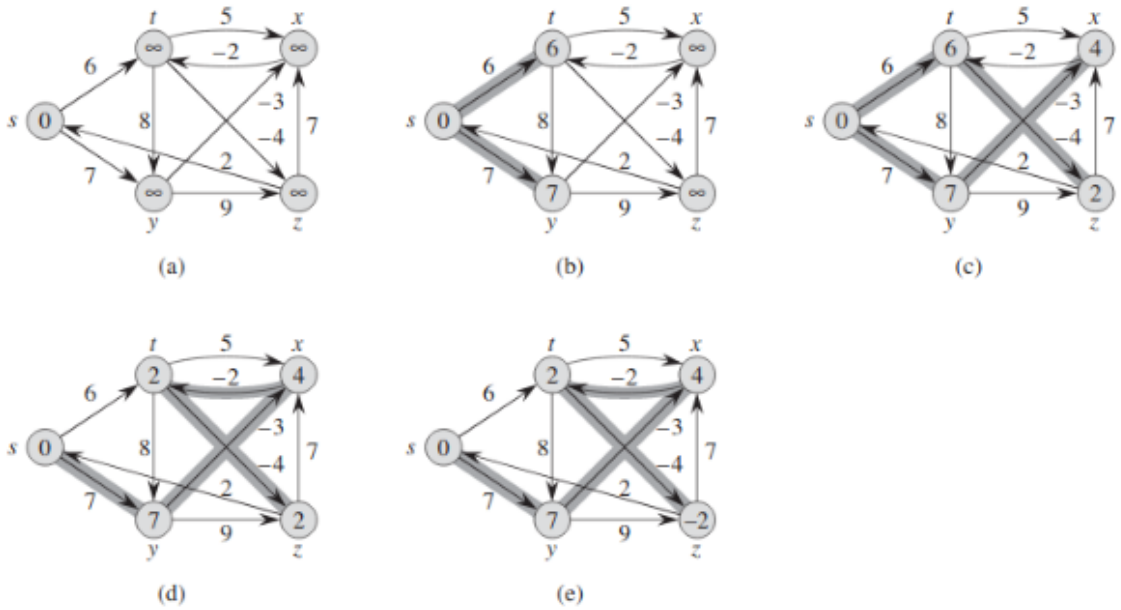


Figure 10: Tomada del libro Cormen et al, Introduction to Algorithms. El orden de procesamiento es $tx, ty, tz, xt, yx, yz, zx, zs, st, sy$.

Caminos minimos entre todos los pares

Consideremos el siguiente problema.

- Entrada: Grafo dirigido G con pesos en los arcos y **sin ciclos negativos**.
- Salida: Matriz M tal que $M[u, v]$ guarda el peso de un camino mínimo de u a v en G.

Caminos minimos entre todos los pares

- Si los pesos son no negativos, podemos usar Dijkstra y tendríamos un algoritmo $O(n(n+m)\lg n) = O(nm \lg n)$.
- Si los pesos pueden ser negativos, usamos Bellman-Fordy y tendríamos un algoritmo $O(n^2 m)$.

Caminos minimos entre todos los pares

Convenciones:

- El conjunto de vértices del grafo en cuestión, $V(G)$, es $\{1, 2, \ldots, n\}$.
- Los pesos de las aristas son representados por una matriz $W = (w_{ij})$, donde

$$\mathrm{OPT}(i,j,\ell) = \begin{cases} 0 & : \text{si } \ell = 0 \text{ e } i = j \\ \infty & : \text{si } \ell = 0 \text{ e } i \neq j \\ \min_{1 \leq k \leq n} \{\mathrm{OPT}(i,k,\ell-1) + w_{kj}\} & : \text{si } \ell > 0 \end{cases}$$

Primer algoritmo

Se basa en la siguiente propiedad.

Lemma

Todo subcamino de un camino mínimo es mínimo.

Primer algoritmo

Para cada par (i, j), sea $\mathrm{OPT}(i, j, \ell)$ el peso de un camino mínimo desde i hacia j que usa como máximo ℓ aristas. Tenemos la sgte recurrencia:

$$\mathrm{OPT}(i,j,\ell) = \begin{cases} 0 & : \text{si } \ell = 0 \text{ e } i = j \\ \infty & : \text{si } \ell = 0 \text{ e } i \neq j \\ \min_{1 \leq k \leq n} \{\mathrm{OPT}(i,k,\ell-1) + w_{kj}\} & : \text{si } \ell > 0 \end{cases}$$

Primer algoritmo

Require: Una matriz W asociada a un grafo G con pesos en las aristas, con n vertices, **y sin ciclos negativos**.

Ensure: Una matriz que guarda las distancias entre todos los pares de vértices de G.

ALGO-1(W)

1: $M^{(1)} \leftarrow W$
2: **for** $\ell = 2$ hast $n - 1$
3: $\quad M^{(\ell)} \leftarrow$ CALCULAR $(M^{(\ell-1)}, W)$
4: **return** $M^{(n-1)}$

Primer algoritmo

CALCULAR($M^{(\ell-1)}, W$)

1: **for** $i = 1$ hast n
2: **for** $j = 1$ hast n
3: $M_{ij}^{(\ell)} \leftarrow \infty$
4: **for** $k = 1$ hast n
5: $M_{ij}^{(\ell)} \leftarrow \min\{M_{ij}^{(\ell-1)}, M_{ij}^{(\ell-1)} + W_{kj}\}$
6: **return** $M^{(\ell)}$

Segundo algoritmo

Require: Una matriz W asociada a un grafo G con pesos en las aristas, con n vertices, **y sin ciclos negativos**.

Ensure: Una matriz que guarda las distancias entre todos los pares de vértices de G.

ALGO-2(W)

1: $M^{(1)} \leftarrow W$
2: $\ell \leftarrow 1$
3: **while** $\ell < n - 1$
4: $\quad M^{(2\ell)} \leftarrow$ CALCULAR $(M^{(\ell)}, M^{(\ell)})$
5: $\quad \ell \leftarrow 2\ell$
6: **return** M^{ℓ}

Tercer algoritmo: Floyd-Warshall

- $\mathrm{OPT}(i, j, k)$ = costo de un camino mínimo de i hacia j cuyos vértices internos están en el conjunto $\{1, 2, \ldots, k\}$.
- k es un vértice interno en P
  En ese caso, existen dos caminos P_1 y P_2 tales que P_1 es un camino de i hacia k y P_2 es un camino de k hacia j. Note que todos los vértices internos de P_1 y P_2 están en $\{1, 2, \ldots, k-1\}$.
- k no es un vértice interno en P.
  En ese caso, P es un camino mínimo entre i y j cuyos vértices internos están en $\{1, \ldots, k-1\}$.

Tercer algoritmo: Floyd-Warshall

$$\mathrm{OPT}(i,j,k) = \begin{cases} w_{ij} & : \text{si } k = 0 \\ \min\{OPT(i,j,k-1), OPT(i,k,k-1) + OPT(k,j,k-1)\} & : \text{si } k \geq 1 \end{cases}$$

Tercer algoritmo: Floyd-Warshall

Require: Una matriz W asociada a un grafo G con pesos en las aristas, con n vertices, **y sin ciclos negativos**.

Ensure: Una matriz que guarda las distancias entre todos los pares de vértices de G.

Floyd-Warshall(W)

1: $M^{(0)} \leftarrow W$
2: **for** $k = 1$ hasta n
3: **for** $i = 1$ hasta n
4: **for** $j = 1$ hasta n
5: $M_{ij}^{(k)} \leftarrow \min\{M_{ij}^{(k-1)}, M_{ik}^{(k-1)} + M_{kj}^{(k-1)}\}$
6: **return** $M^{(n)}$

Cuarto algoritmo: Johnson

- Tiempo de ejecución $O(n^2 \lg n + nm)$. Si el grafo es esparso, este tiempo de ejecución es $O(n^2 \lg n)$: mejor que todos los algoritmos anteriores.
- Idea: cambiar los pesos del grafo de manera tal que los nuevos pesos sean no negativos, a la vez que los caminos mínimos se siguen manteniendo. Luego corre n veces dijkstra.
- No es necesario construir una matriz de pesos para el input.

Cuarto algoritmo: Johnson

Require: Un grafo G con pesos w en las aristas, con n vertices, **y sin ciclos negativos**.

Ensure: Una matriz que guarda las distancias entre todos los pares de vértices de G.

Johnson(G, w)

1: $h \leftarrow$ Calcular-h(G, w)
2: **for** cada arista uv
3: $\quad \hat{w}_{uv} = w_{uv} + h(u) - h(v)$
4: **for** cada vértice u
5: $\quad dist \leftarrow$ Dijkstra$(G, \hat{w}, u)$
6: $\quad$ **for** cada vértice v
7: $\quad\quad M_{uv} \leftarrow dist(v) + h(v) - h(u)$
8: **return** M

Cuarto algoritmo: Johnson

Require: Un grafo G con pesos w en las aristas, con n vertices, **y sin ciclos negativos**.

Ensure: Un arreglo h, indexado por $V(G)$, tal que $h(v) \leq h(u) + w_{uv}$ para cada arista uv.

Calcular-h(G, w)

1: Sea G' el grafo auxiliar resultante de adicionar un vértice s a G y dar pesos 0 a todas las aristas que salen de s. Sea w' la función de pesos resultante.
2: $h \leftarrow$ Bellman-Ford(G', w', s)
3: **return** h

Gracias