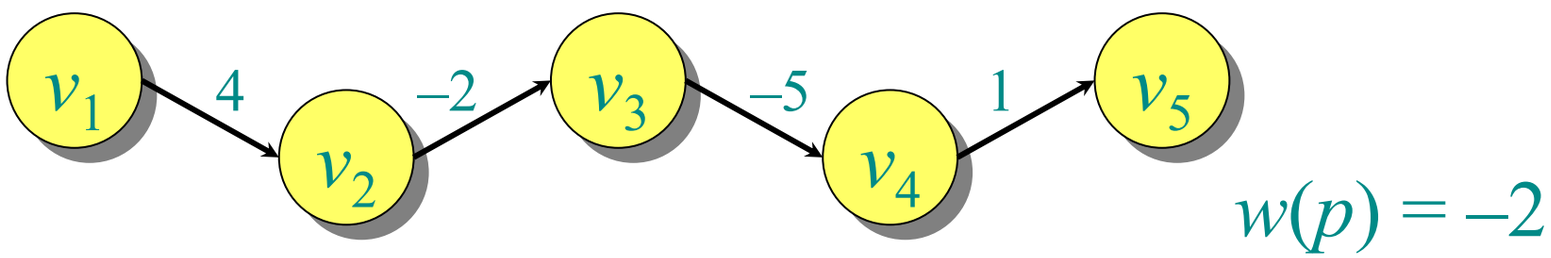
Caminos más cortos en gráficas

comp-420

Caminos más cortos en gráficas

- Dado un grafo con peso y dirigido G = (V, E) con una función de peso w : E→R que transforma aristas a valores reales de peso.
- El peso del camino p=<v_0, v_1, ..., v_k> es la suma de los pesos de los ejes que lo constituyen.

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$



Caminos más cortos en gráficas

- Un camino más corto de u a v es un camino de peso mínimo de u a v.
- El peso del camino más corto de u a v se define como:

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{si hay un camino de } u \text{ a } v, \\ \infty & \text{en cualquier otro caso.} \end{cases}$$

- Un camino más corto de u a v se define como cualquier camino p con peso:

$$w(p) = \delta(u,v)$$

Caminos más cortos en gráficas

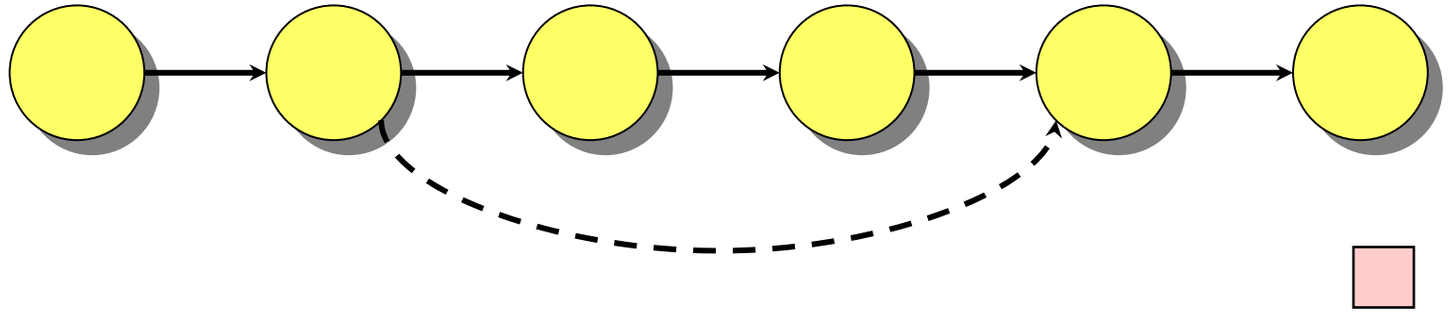
- ¿Qué interpretación puede darse al peso de las aristas?
  - distancia,
  - tiempo,
  - costo,
  - penalidad,
  - pérdida,
  - cualquier cantidad que se acumule linealmente a lo largo del camino y que se quiera minimizar.
- El algoritmo BFS es un algoritmo que encuentra el camino más corto en gráficas sin peso (con peso unitario). Es la base para los algoritmos para gráficas con peso.

Caminos más cortos en gráficas

- Nos enfocaremos a encontrar el camino más corto a partir de una sola fuente:
  - Dada una gráfica $G = (V, E)$, ¿cómo ir de un vértice fuente $s \in V$ a cáda vértice $v \in V$?
- ¿Qué otros problemas se podrían resolver?
  - camino más corto a un solo destino (a partir de todos los vértices).
  - camino más corto entre un solo par de vértices.
  - camino más corto entre todos los pares de vértices de una gráfica (hay métodos más eficientes para resolver este problema).

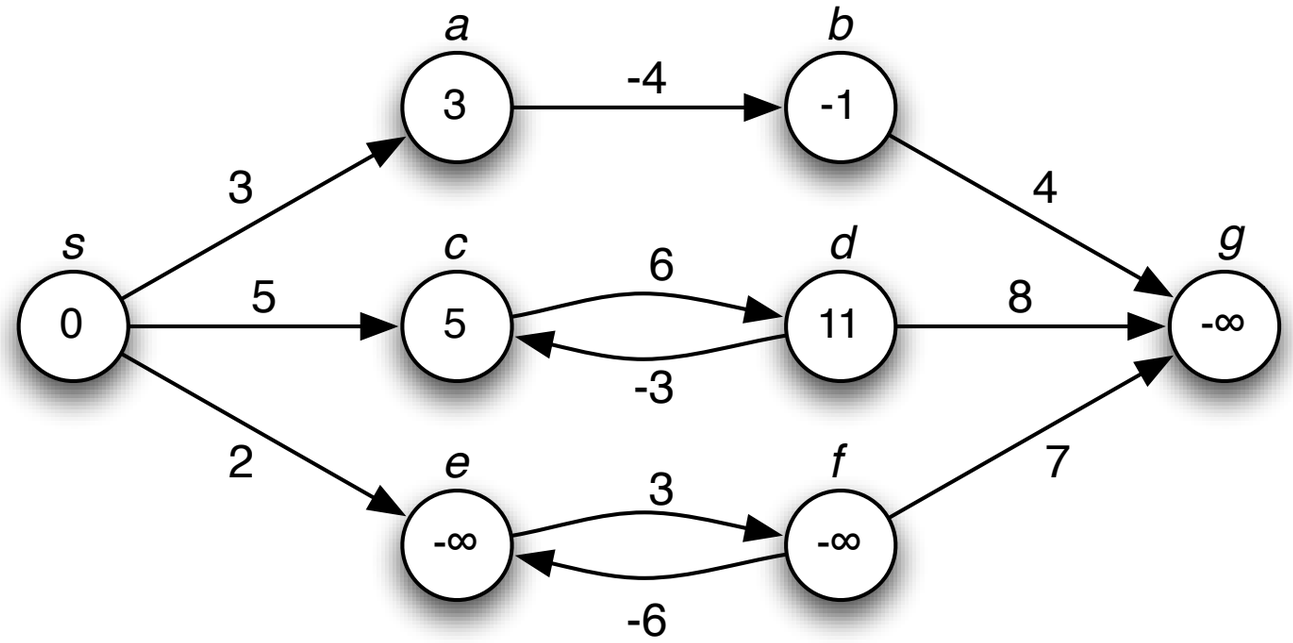
Subestructura óptima

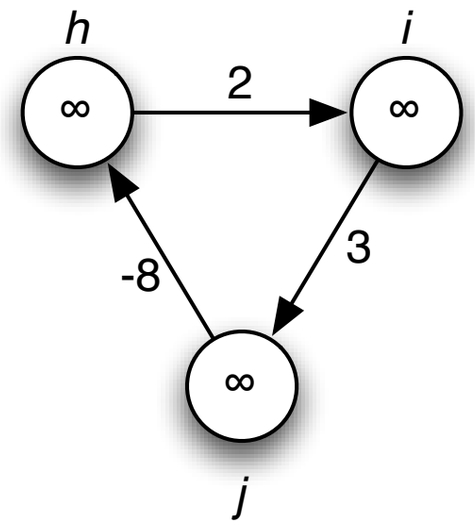
- Un subcamino de un camino más corto (óptimo) entre dos vértices es un camino más corto.
- Cortando y pegando ...



- Esta propiedad de subestructura óptima...¿de qué tipo de algoritmos es característica?
  - Algoritmo de Dijkstra: algoritmo glotón.
  - Algoritmo de Floyd-Warshall: programación dinámica.

Gráficas con aristas de peso negativo





$< s, a >, \delta(s, a) = w(s, a) = 3$

$< s, b >, \delta(s, b) = w(s, a) + w(a, b) = 3 + (-4) = -1$

$< s, c >, < s, c, d, c >, < s, c, d, c, d, c > \ldots \;\; w < c, d, c >= 6 + (-3) = 3 > 0$

$< s, c >, \delta(s, c) = w(s, c) = 5$

$< s, d >, \delta(s, d) = w(s, c) + w(c, d) = 11$

$< s, e >, < s, e, f, e >, < s, e, f, e, f, e > \ldots w(e, f, e) = 3 + (-6) = -3 < 0$

$< s, e >= \delta(s, e) = -\infty$

Gráficas con aristas de peso negativo

- Si la gráfica G=(V,E) no contiene cíclos de peso negativo alcanzables desde el vértice fuente s, entonces para todo v∈V, el peso del camino más corto $\delta(s,v)$ sigue bien definido, incluso si algunas aristas tienen peso negativo.

- Si la gráfica contiene cíclos de peso negativo alcanzables desde s, no existen caminos más cortos a esos vértices y decimos que $\delta(s,v)=-\infty$.

- Los vértices no alcanzables desde s tienen un peso de camino más corto de $\delta(s,v)=\infty$.

- Normalmente se verifica la existencia de cíclos con pesos negativos al iniciar un algoritmo de caminos más cortos.

- ¿Puede tener cíclos un camino más corto?¿Por qué?

Representación de los caminos más cortos

- La mayoría de las veces no sólo queremos el peso del camino sino también queremos tener registro de las aristas y vértices por los que este camino pasa.

- Dada una gráfica $G=(V,E)$, mantenemos para cada vértice $v \in V$, un predecesor $v.\pi$ que será otro vértice en la gráfica o NIL.

- Los algoritmos de caminos más cortos ponen el atributo π de tal manera que siguiendo la cadena de predecesores que se originan en v se pueda reconstruir el camino hasta s.

```
PRINT-SHORTEST-PATH (G, s, v)
1  if v = s
2      then print s
3      else if π[v] = NIL
4          then print "no path from" s "to" v exists
5          else PRINT-SHORTEST-PATH(G, s, π[v])
6               print v
```

Representación de los caminos más cortos

- Durante la ejecución de un algoritmo de caminos más cortos, los valores de π no representan los caminos más cortos sino la subgráfica de predecesores $G_\pi=(V_\pi,E_\pi)$.

- Aquí se define a V_π como el conjunto de vértices de G con predecesores diferentes a NIL, más el vértice fuente:

$$V_\pi = \{v \in V : \pi[v] \neq \text{NIL } \} \cup \{s\}$$

- E_π es el conjunto de aristas dirigidas inducido por valores de π para los vértices V_π.

$$E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}.$$

- Los valores de π producidos por los algoritmos que veremos tienen la propiedad que al terminar, G_π es un árbol de caminos más cortos.

Árboles de caminos más cortos

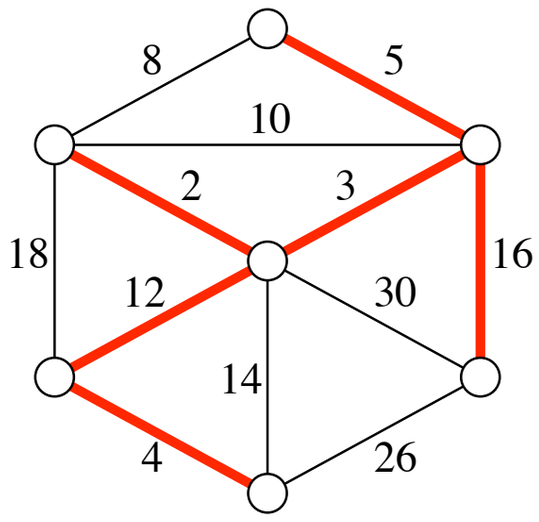
- Un árbol de caminos más cortos (shortest-paths tree), es un árbol con raíz que contiene un camino más corto desde la fuente s a cada vértice alcanzable desde s.

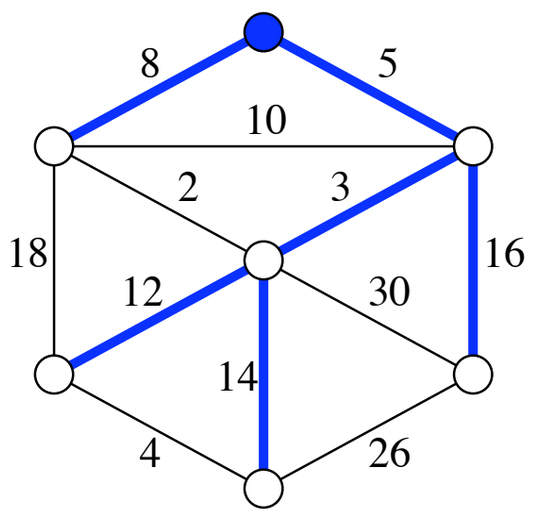
- Los árboles de caminos más cortos son generalmente muy diferentes a los árboles mínimos generadores.

- Los árboles mínimos generadores son únicos (cuando las aristas no tienen el mismo valor).

- Los árboles de caminos más cortos son diferentes para cada vértice fuente.

MST





SPT

Algoritmo genérico de caminos más cortos

- Problema: a partir de un vértice fuente $s \in V$, encontrar los pesos de los caminos más cortos $\delta(s,v)$ para todo $v \in V$.
- Si todos los pesos de aristas $w(u,v)$ son positivos, todos los pesos de camino más corto deben existir.

Algoritmo genérico de caminos más cortos

- Para cada vértice $v \in V$, se mantienen dos atributos:
  - $d[v]$ es la longitud del camino más corto tentativo de s a v.
  - $\pi[v]$ es el predecesor de v en el camino más corto tentativo.
- Se inicializa el algoritmo de la siguiente manera:

```
INITIALIZE-SINGLE-SOURCE (G, s)
1  for each vertex v ∈ V[G]
2      do d[v] ← ∞
3         π[v] ← NIL
4  d[s] ← 0
```

- Con un tiempo de ejecución de $\Theta(V)$.

Relajación de aristas en caminos más cortos

- Después de la inicialización π[v] = NIL para todo v∈V, d[s]=0 y d[v]=∞ para v∈V-{s}.
- Se dice que una arista (u,v) está tensa si d[u] + w(u,v) < d[v].
- Si (u,v) es una arista tensa entonces el camino tentativo más corto $s \rightsquigarrow v$ es incorrecto ya que el camino $s \rightsquigarrow u \longrightarrow v$ es más corto.
- El algoritmo genérico encuentra repetidamente aristas tensas en una gráfica y las relaja:

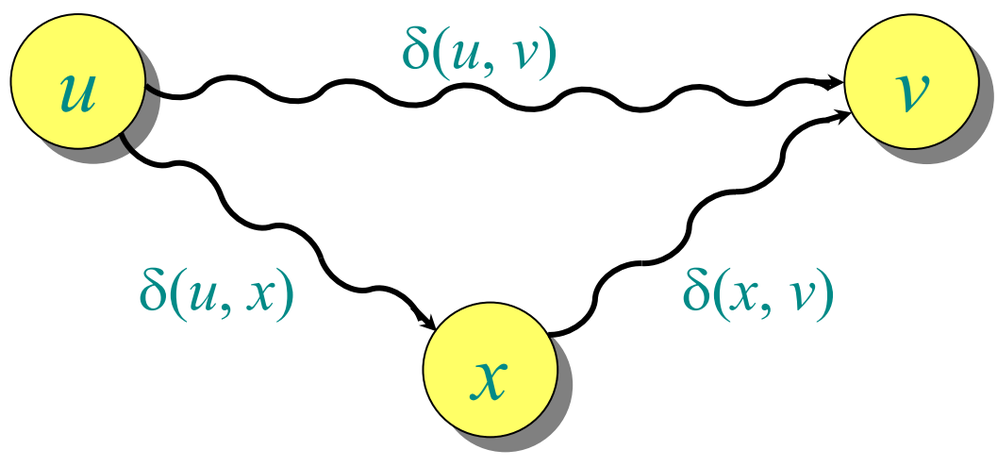
```
RELAX (u, v, w)
1  if d[v] > d[u] + w(u, v)
2     then d[v] ← d[u] + w(u, v)
3          π[v] ← u
```

Algoritmo genérico de caminos más cortos

- El algoritmo termina cuando ya no hay aristas tensas obteniendo el camino más corto a partir del vértice s.
- Para probar que los algoritmos de caminos más cortos son correctos, se necesitan enumerar varias propiedades de los caminos más cortos y del proceso de relajación:

Desigualdad triangular:

Para todo $u,v,x \in V$, tenemos $\delta(u,v) \leq \delta(u,x) + \delta(x,v)$.



Propiedades de los caminos más cortos y del proceso de relajación

- **Propiedad de la cota superior:**

  Siempre tenemos $d[v] \geq \delta(s,v)$ para todos los vértices $v \in V$ y una vez que $d[v]$ alcanza el valor $\delta(s,v)$ nunca cambia.

- **Propiedad de no existencia de camino:**

  Si no existe un camino de s a v, entonces tenemos $d[v] = \delta(s,v) = \infty$.

- **Propiedad de convergencia:**

  Si $s \rightsquigarrow u \rightarrow v$ es un camino más corto en G para algún $u,v \in V$, y si $d[u]=\delta(s,u)$ en cualquier momento anterior a la relajación de la arista (u,v), entonces $d[v]=\delta(s,v)$ en todo momento ulterior.

Propiedades de los caminos más cortos y del proceso de relajación

- Propiedad de la relajación del camino:

  Si $p=\langle v_0,v_1,\ldots,v_k\rangle$ es el camino más corto de $s=v_0$ a v_k, y las aristas de p están relajadas en el orden (v_0,v_1), (v_1,v_2), ... , (v_{k-1}, v_k), entonces $d[v_k]=\delta(s,v_k)$.

- Propiedad del subgrafo predecesor:

  Una vez que $d[v]=\delta(s,v)$ para todo $v\in V$, el subgrafo predecesor es un árbol de caminos más cortos con raíz en s.

Algoritmo genérico de caminos más cortos

- ¿En qué orden se deben relajar las aristas?
- ¿Cuáles aristas deben relajarse?
- Parecido al algoritmo de recorrido en gráficas:
    - mantenemos una bolsa de vértices, que inicialmente contiene el vértice fuente s.
    - cuando sacamos un vértice u de la bolsa, revisamos sus aristas salientes, buscando una para relajar.
    - cuando encontramos una, relajamos la arista (u,v) y ponemos v en la bolsa.

Algoritmo genérico de caminos más cortos

```
GenericSSSP(G, s)
1  Initialize-Single-Source(G, s)
2  put s in the bag
3  while the bag is not empty
4      take u from the bag
5      for all edges (u, v)
6          if (u, v) is tense
7              Relax(u, v, w)
8              put v in the bag
```

Algoritmo genérico de caminos más cortos

- Al igual que los algoritmos de recorrido en gráficas usar diferentes estructuras como bolsa resulta en diferentes algoritmos:
  - pila (stack)
  - cola (queue)
  - montículo (heap)
- Usando una pila se necesitan $\Theta(2^V)$ operaciones de relajación de aristas en el peor caso!
- Usando colas o montículos tenemos algoritmos mucho más eficientes.
- Si implementamos la bolsa con un montículo donde la llave de cada vértice v es d[v], obtendremos el Algoritmo de Dijkstra.

Algoritmo de Moore (Bellman-Ford)

- Dado una gráfica dirigida con pesos en sus aristas con fuente s y función de peso $w: E \rightarrow R$, el algoritmo de Moore regresa un valor booleano indicando si hay o no un cíclo negativo alcanzable desde la fuente.
- Si existe un cíclo negativo, el algoritmo declara que no existe una solución.
- Si no existe, el algoritmo produce el camino más corto.
- El algoritmo usa relajación, reduciendo progresivamente el estimado $d[v]$ en el peso del camino más corto a partir de s hasta todos los vértices $v \in V$ hasta que alcanza el peso del camino más corto $\delta(s,v)$.

Algoritmo de Moore (Bellman-Ford)

BELLMAN-FORD (G, w, s)

```
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to |V[G]| − 1
3      do for each edge (u, v) ∈ E[G]
4          do RELAX(u, v, w)
5  for each edge (u, v) ∈ E[G]
6      do if d[v] > d[u] + w(u, v)
7          then return FALSE
8  return TRUE
```

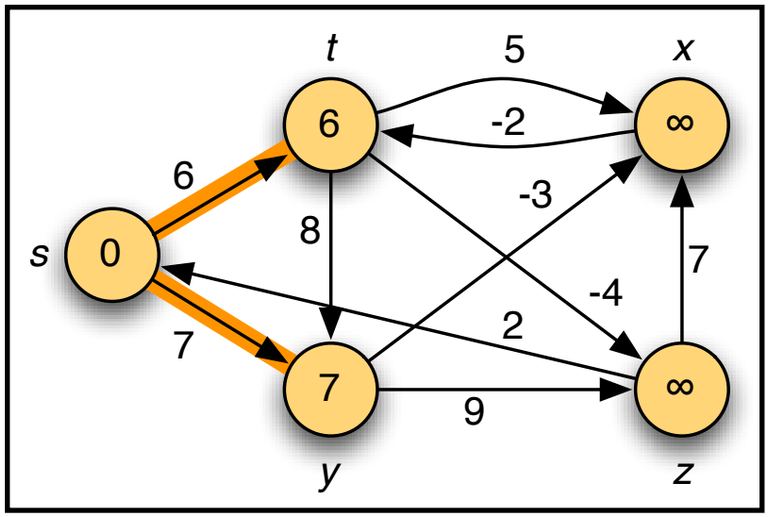
INITIALIZE-SINGLE-SOURCE (G, s)

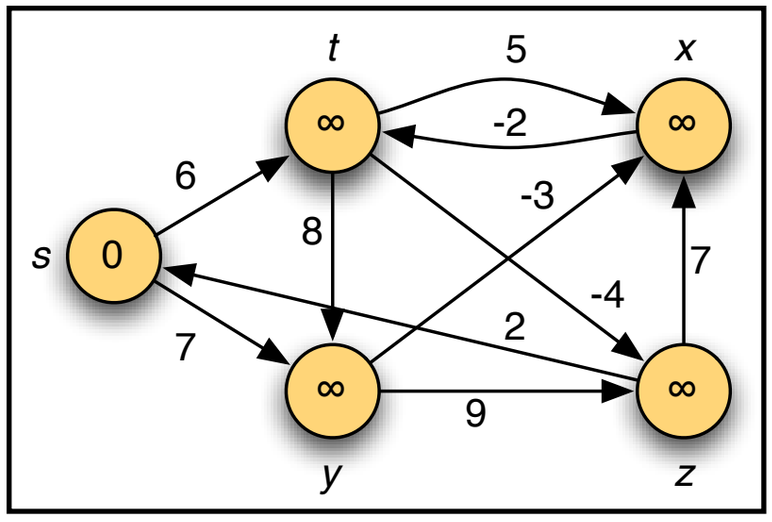
```
1  for each vertex v ∈ V[G]
2      do d[v] ← ∞
3         π[v] ← NIL
4  d[s] ← 0
```

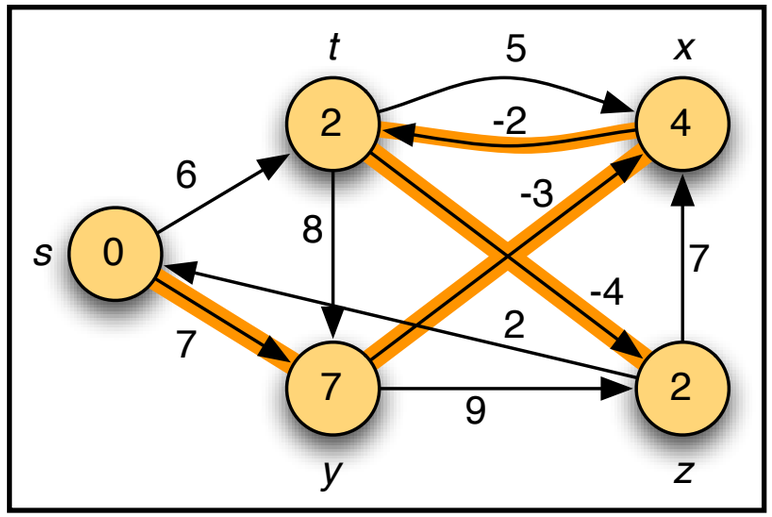
RELAX (u, v, w)

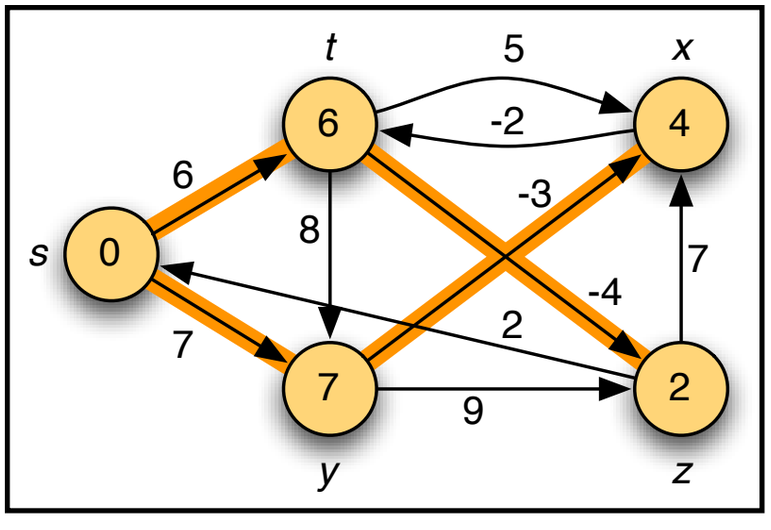
```
1  if d[v] > d[u] + w(u, v)
2      then d[v] ← d[u] + w(u, v)
3           π[v] ← u
```

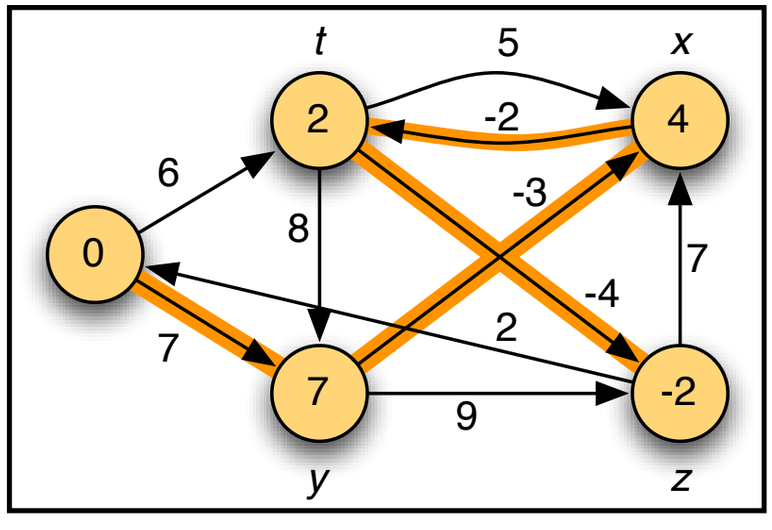
Algoritmo de Moore (Bellman-Ford)











- O(VE)

Algoritmo de Dijkstra

- Edsger W. Dijkstra (1930-2002)
- Pionero de la ciencia e industria de la computación.
- Matemático y físico teórico con PhD en Ciencias de la Computación.
- Turing Award en 1972.
- Fue de los primeros en opinar que la lógica matemática es indispensables para construir programas computacionales adecuados.
- Famoso por su algoritmo de caminos más cortos en gráficas, por diseñar y codificar el primer compilador de Algol 60.
- Líder del movimiento de abolición de la operación GOTO de la programación (1968): http://david.tribble.com/text/goto.html
- Primero en utilizar la expresión "programación estructurada" (1969).

Algoritmo de Dijkstra (1959)

- Adecuado para algoritmos que no tienen aristas con pesos negativos.
- Idea: algoritmo glotón.
  - Mantener un conjunto S de vértices cuyas distancias de camino más corto desde s son conocidas.
  - En cada paso agregar a S el vértice $v \in V - \{S\}$ cuyo estimado de distancia desde s es mínimo.
  - Actualizar los estimados de distancias para los vértices adyacentes a v.
  - Cada vértice es escaneado a lo más una vez y por lo tanto relajado a lo más una vez.

Algoritmo de Dijkstra

INITIALIZE-SINGLE-SOURCE (G, s)

```
1  for each vertex v ∈ V[G]
2      do d[v] ← ∞
3         π[v] ← NIL
4  d[s] ← 0
```

DIJKSTRA(G, w, s)

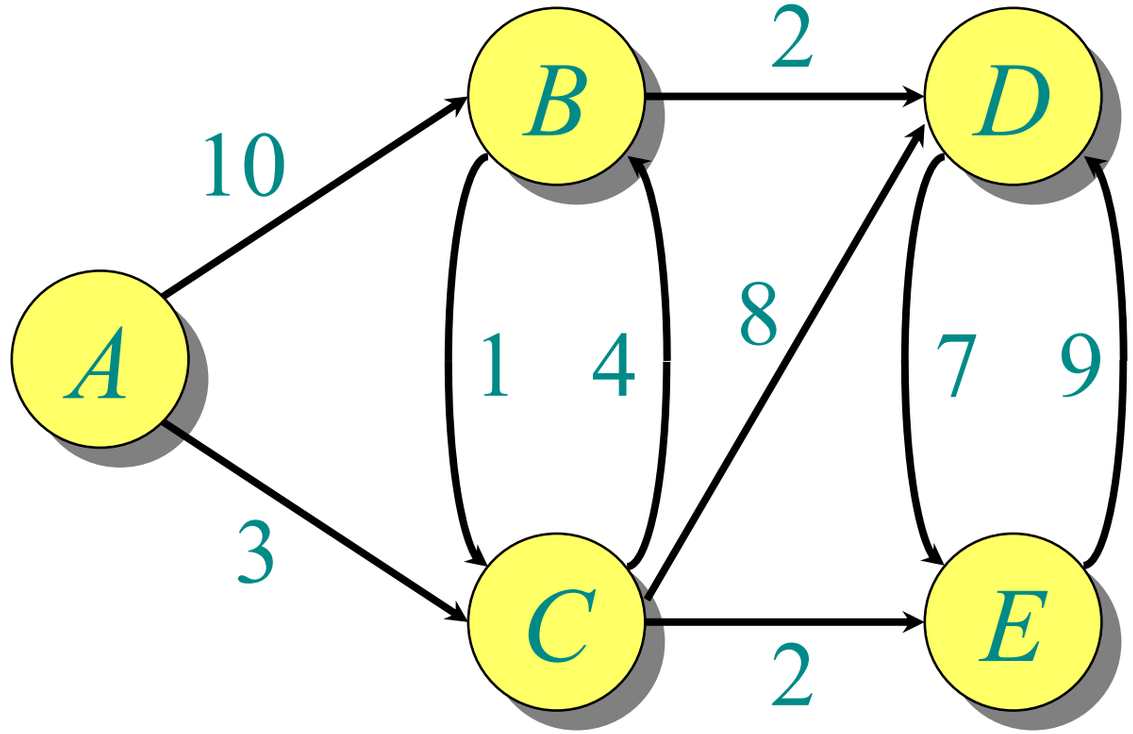
```
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5      do u ← EXTRACT-MIN(Q)
6         S ← S ∪ {u}
7         for each vertex v ∈ Adj[u]
8             do RELAX(u, v, w)
```

RELAX (u, v, w)

```
1  if d[v] > d[u] + w(u, v)
2      then d[v] ← d[u] + w(u, v)
3           π[v] ← u
```

Ejemplo: algoritmo de Dijkstra

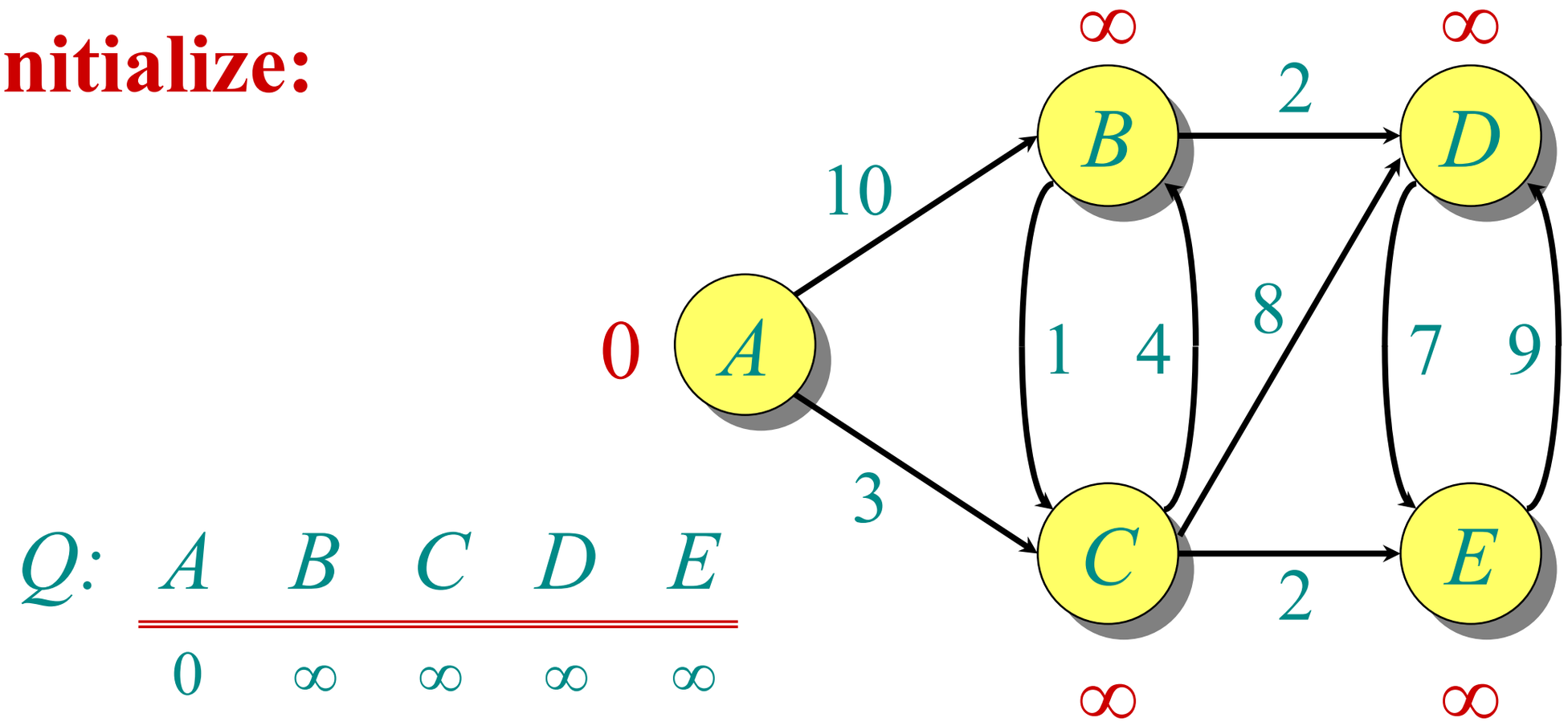
Gráfica sin aristas con peso negativo



Ejemplo de las notas del curso de C. Leiserson del MIT

Ejemplo: algoritmo de Dijkstra

Initialize:



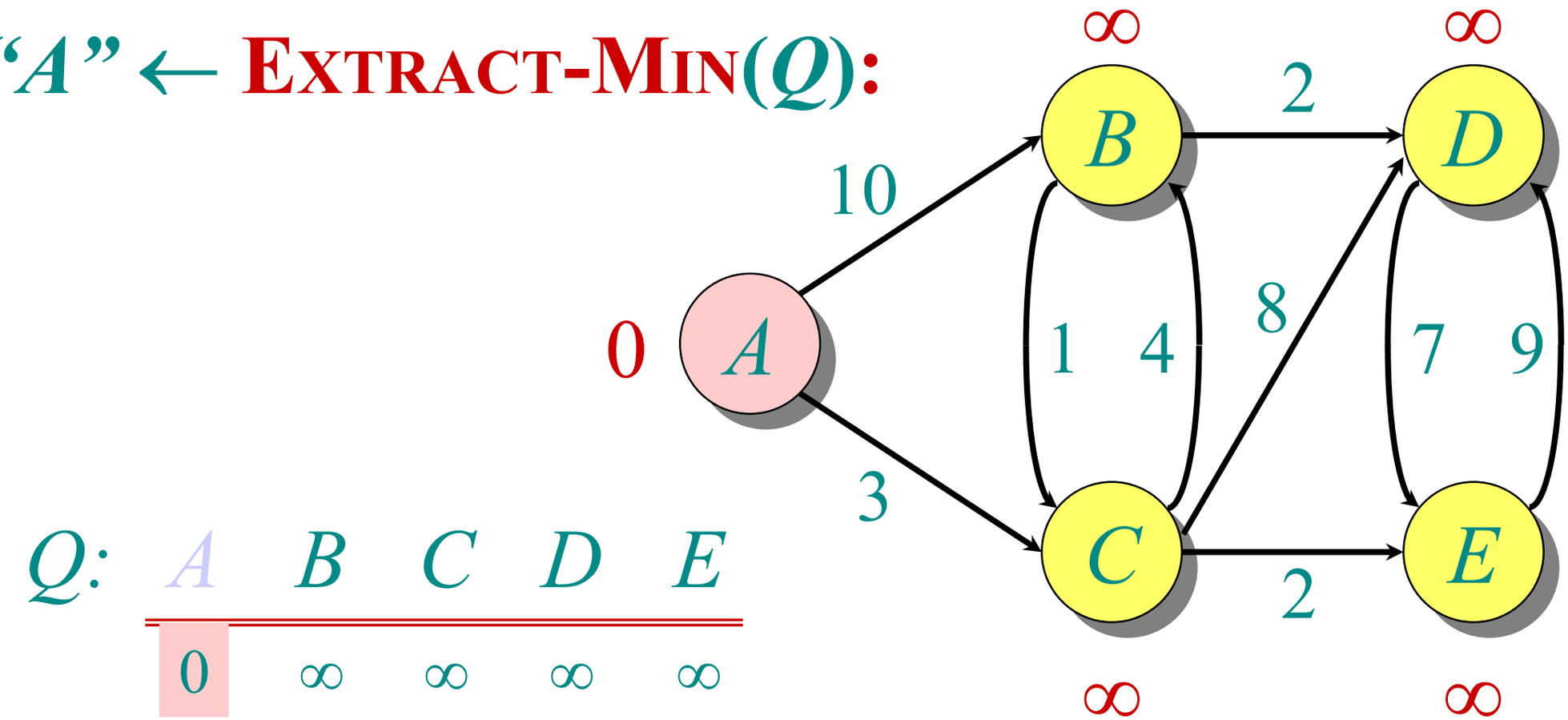
Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

Ejemplo de las notas del curso de C. Leiserson del MIT

Ejemplo: algoritmo de Dijkstra

"A" ← **EXTRACT-MIN(*Q*):**



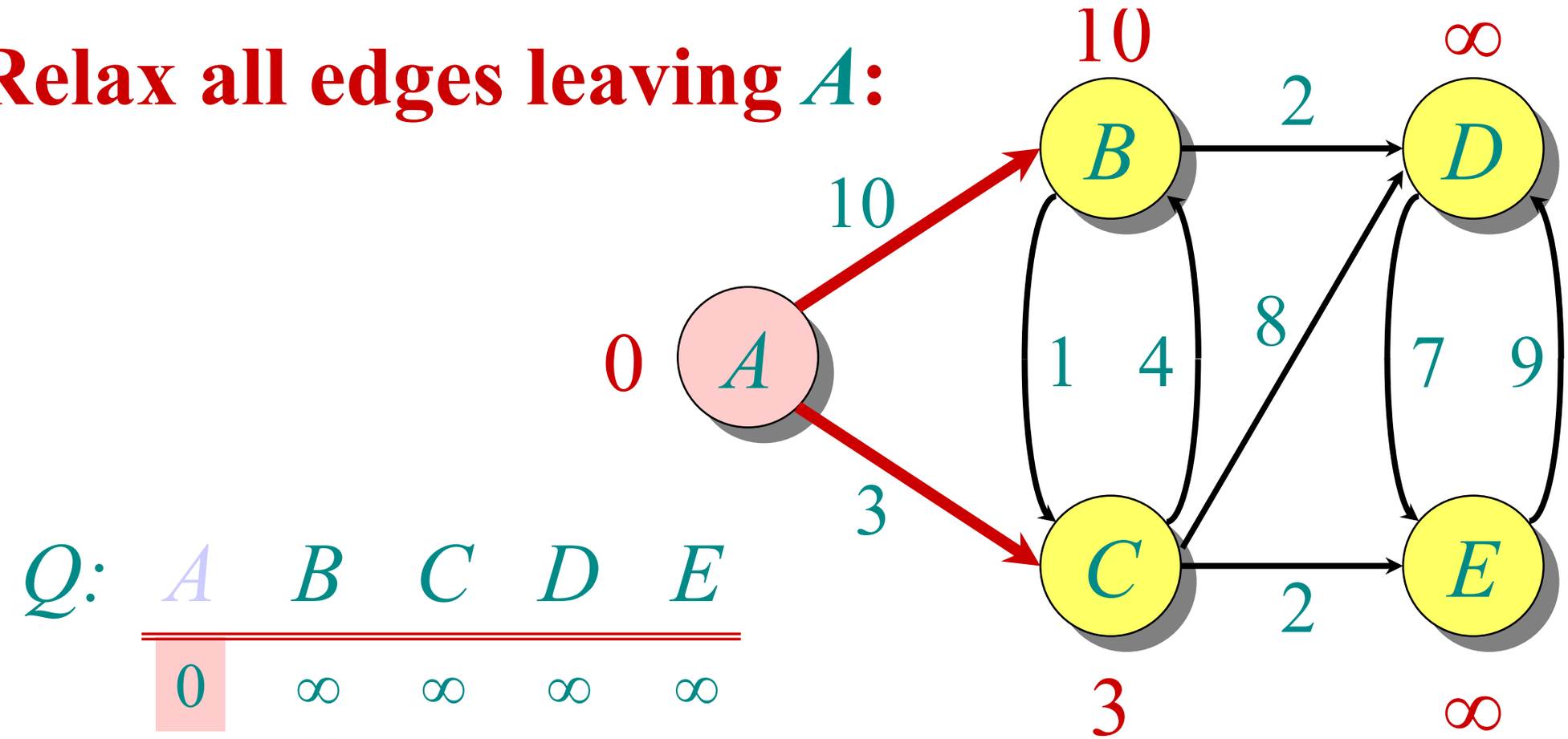
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞

S: { *A* }

Ejemplo de las notas del curso de C. Leiserson del MIT

Ejemplo: algoritmo de Dijkstra

Relax all edges leaving *A*:



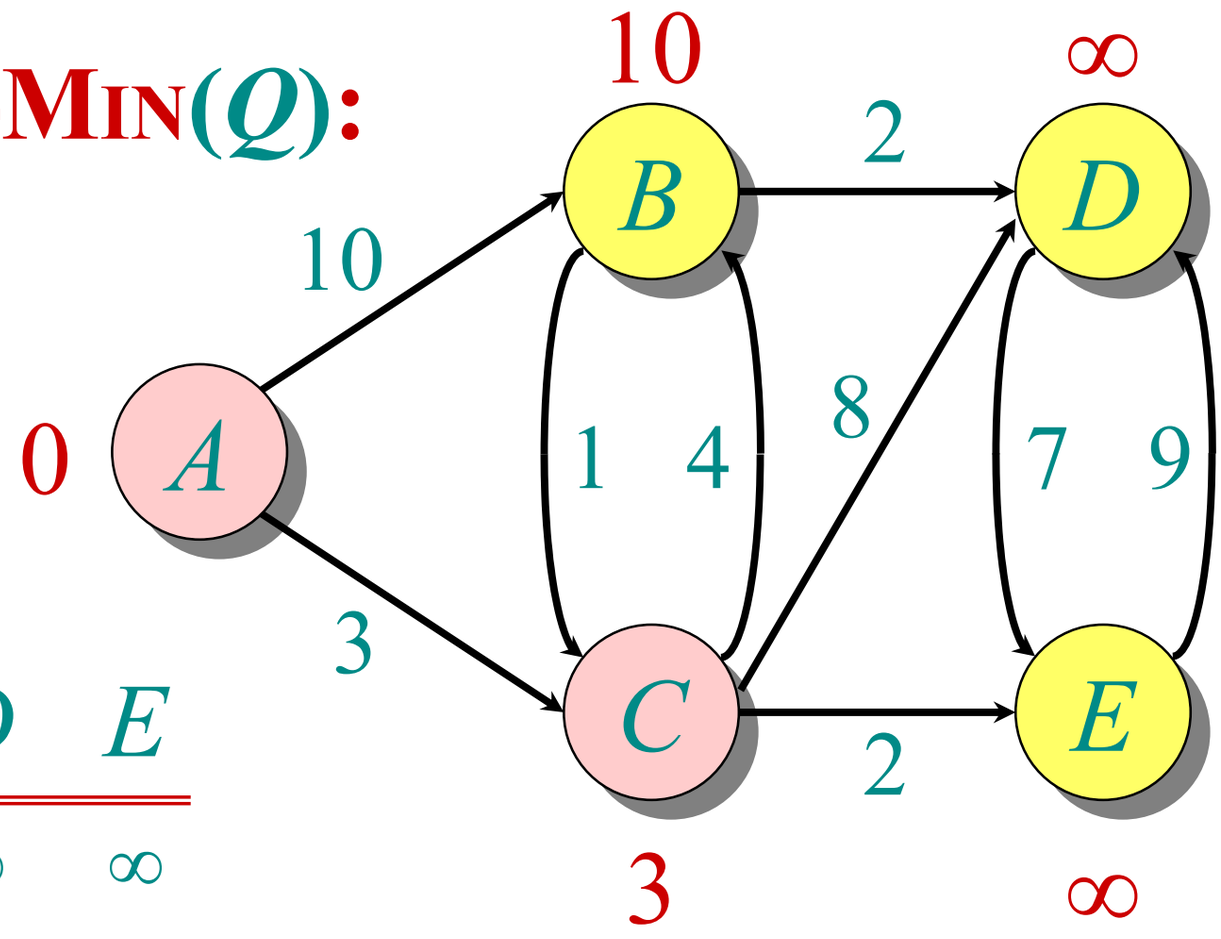
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A* }

Ejemplo de las notas del curso de C. Leiserson del MIT

Ejemplo: algoritmo de Dijkstra

"C" ← **EXTRACT-MIN(*Q*):**



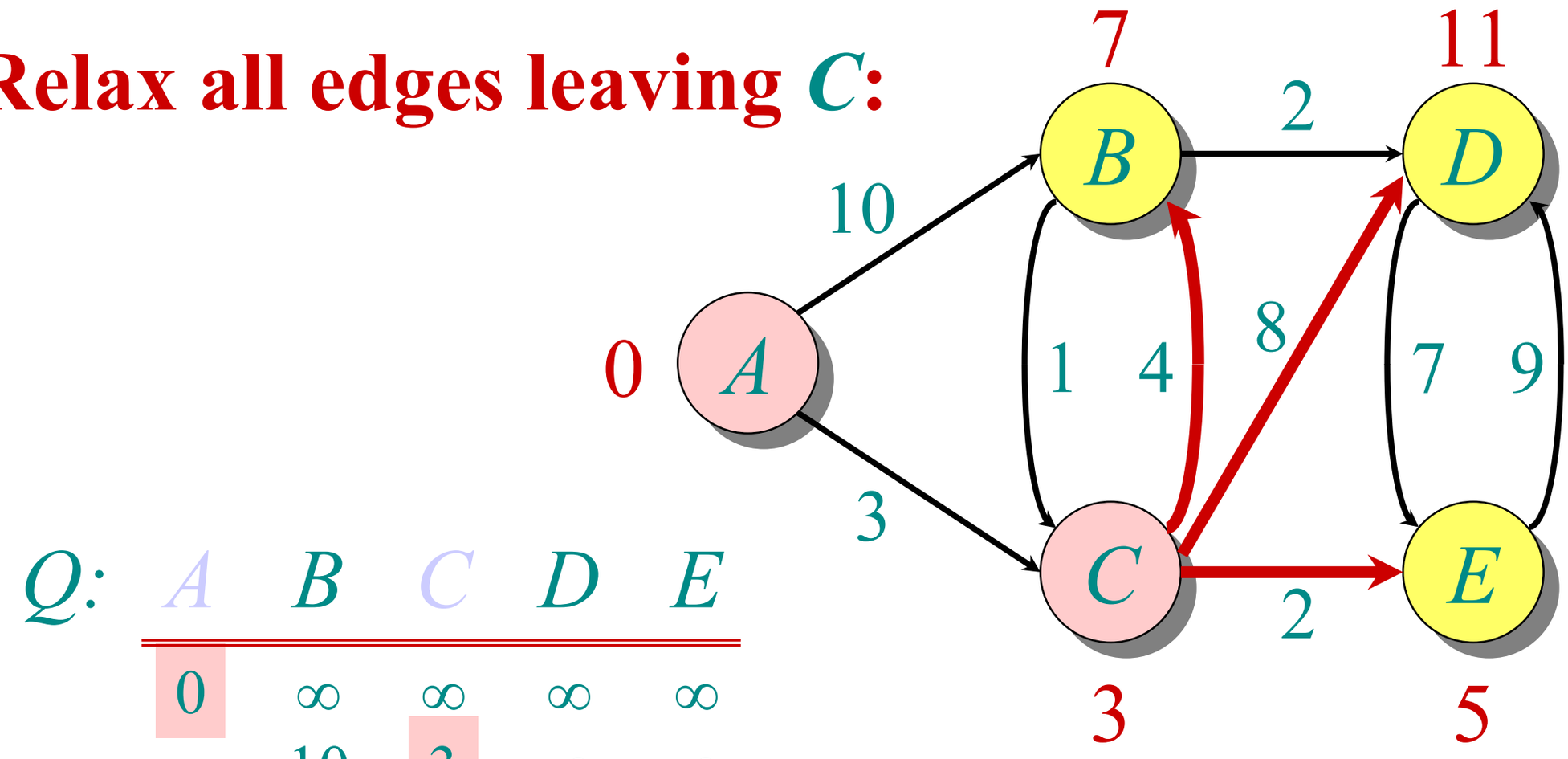
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A*, *C* }

Ejemplo de las notas del curso de C. Leiserson del MIT

Ejemplo: algoritmo de Dijkstra

Relax all edges leaving *C*:



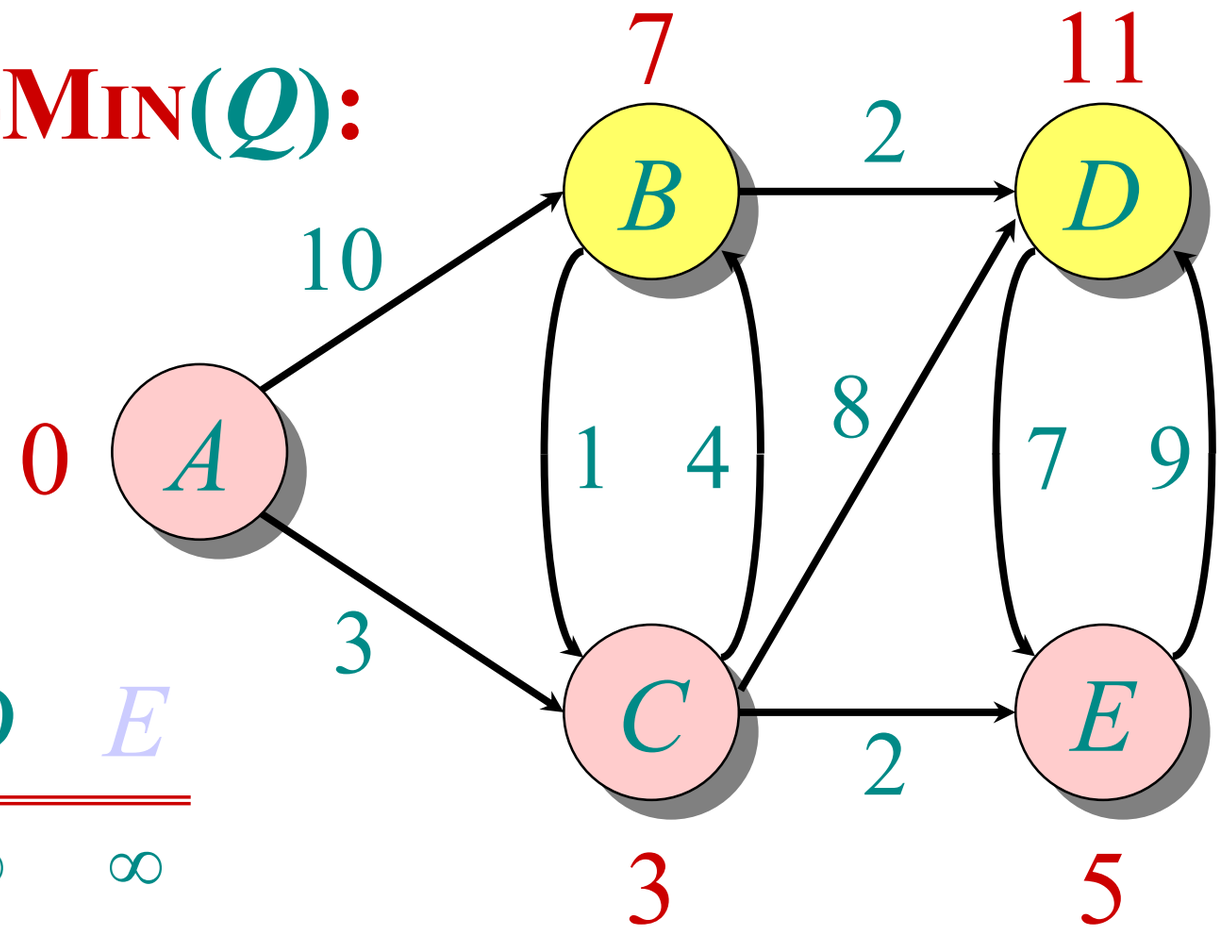
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C* }

Ejemplo de las notas del curso de C. Leiserson del MIT

Ejemplo: algoritmo de Dijkstra

"E" ← **EXTRACT-MIN(*Q*):**



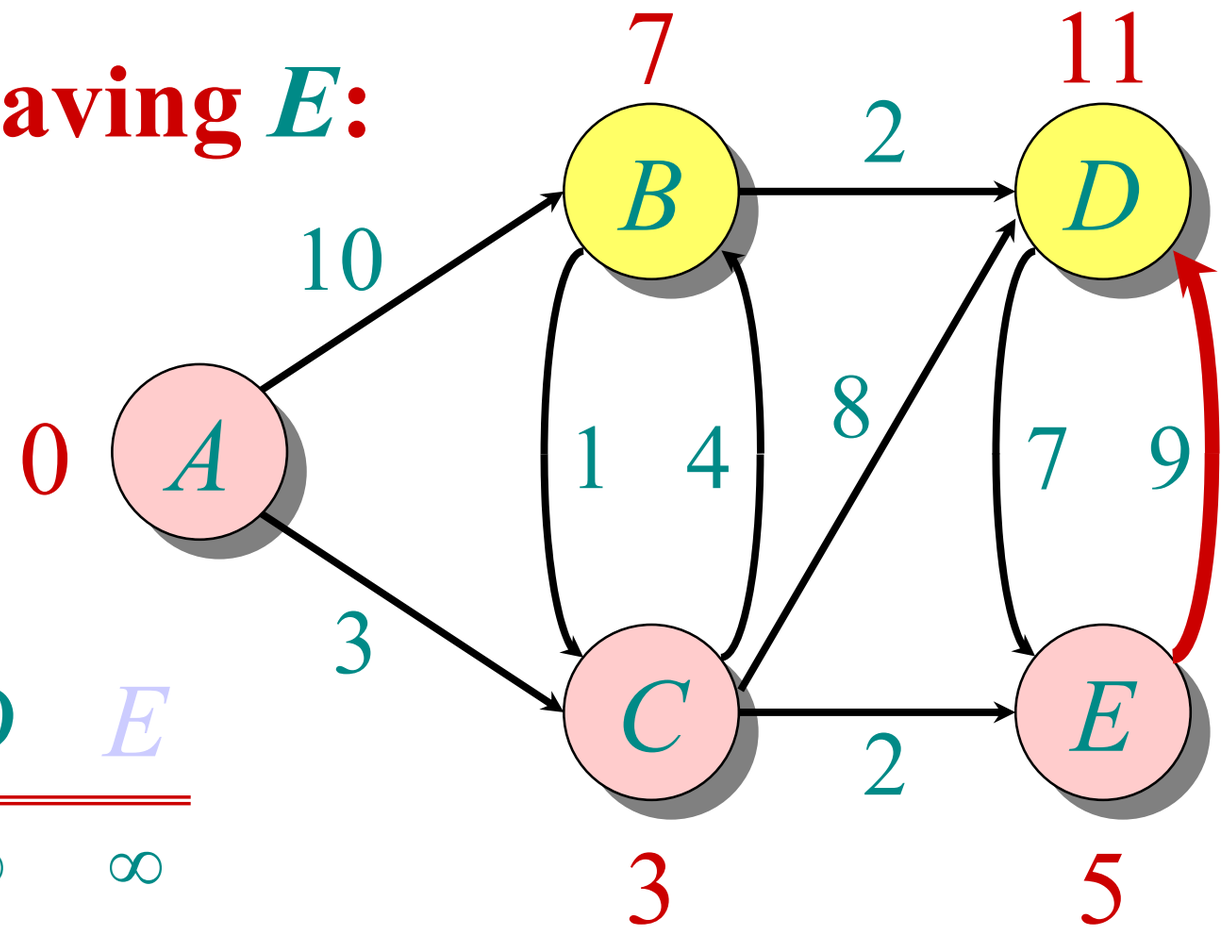
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C*, *E* }

Ejemplo de las notas del curso de C. Leiserson del MIT

Ejemplo: algoritmo de Dijkstra

Relax all edges leaving *E*:



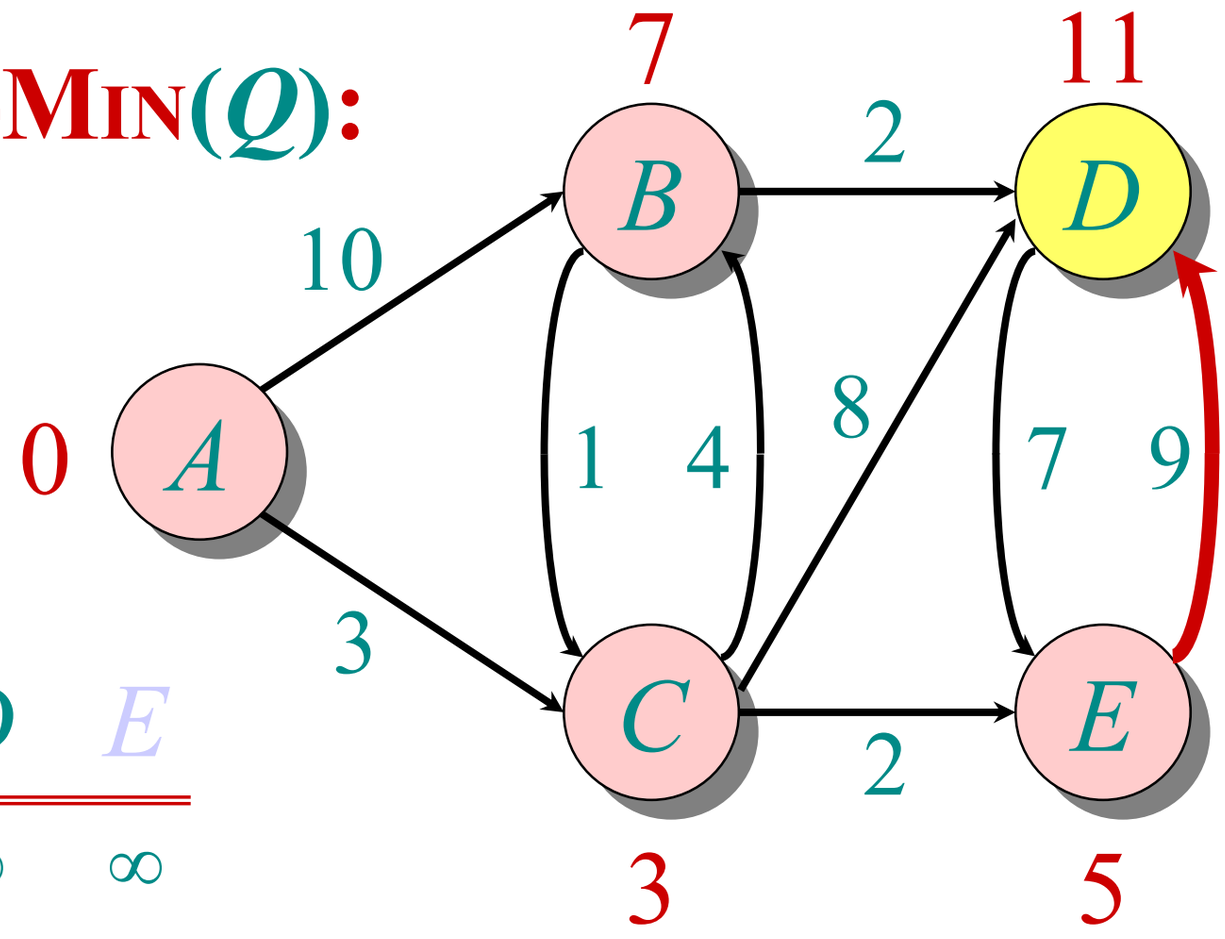
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { *A*, *C*, *E* }

Ejemplo de las notas del curso de C. Leiserson del MIT

Ejemplo: algoritmo de Dijkstra

"B" ← **EXTRACT-MIN(*Q*):**

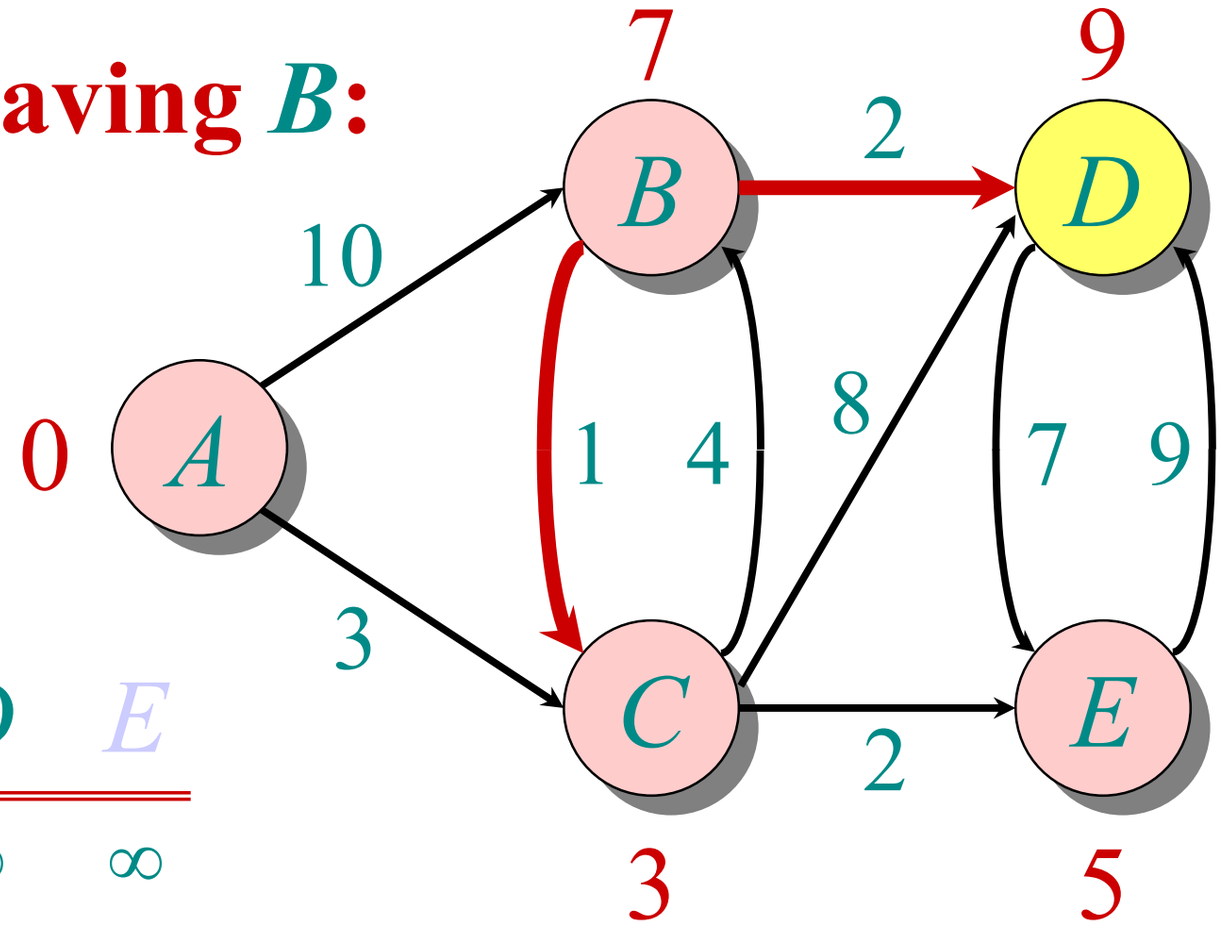


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

Ejemplo de las notas del curso de C. Leiserson del MIT

Ejemplo: algoritmo de Dijkstra

Relax all edges leaving *B*:



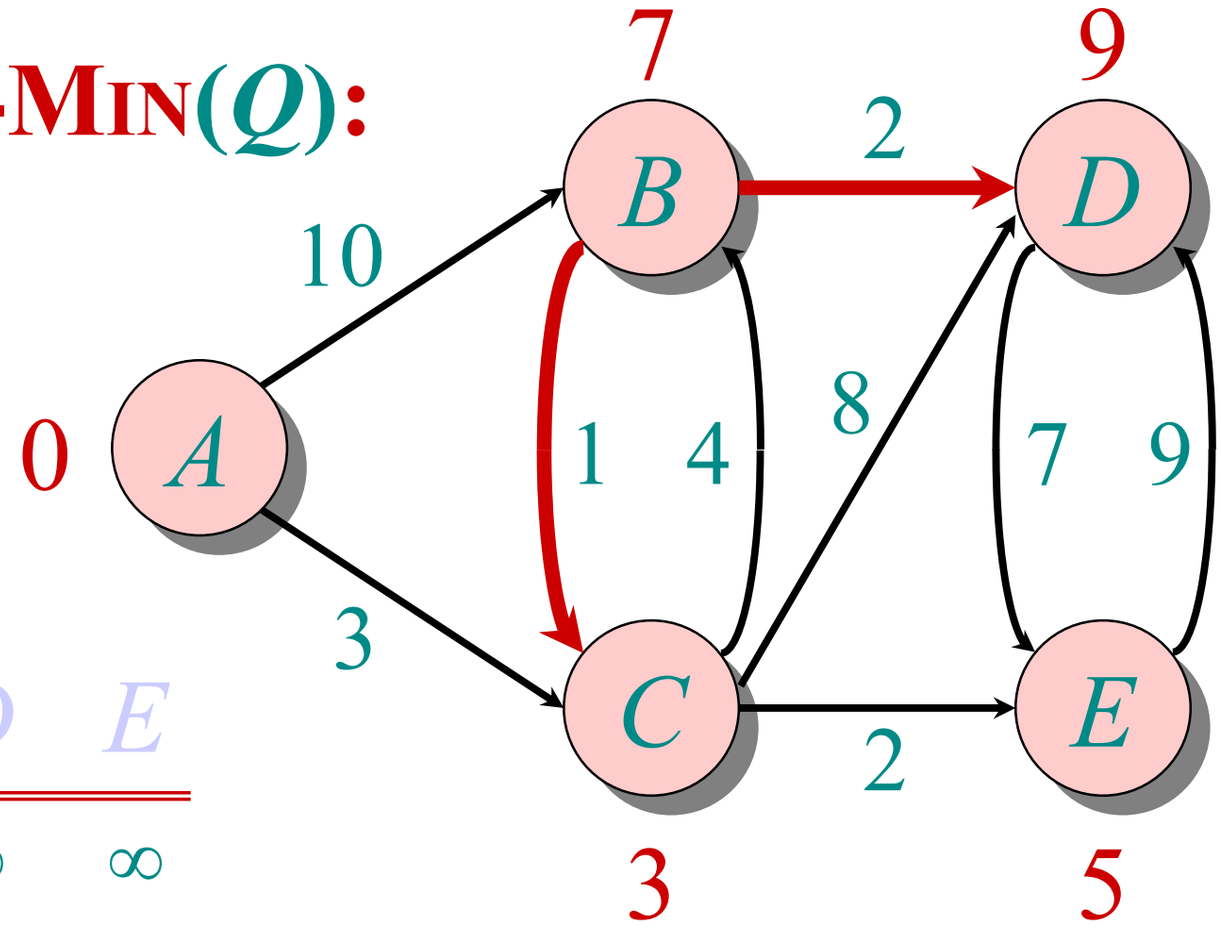
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { *A*, *C*, *E*, *B* }

Ejemplo de las notas del curso de C. Leiserson del MIT

Ejemplo: algoritmo de Dijkstra

"D" ← **EXTRACT-MIN(*Q*):**



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { *A*, *C*, *E*, *B*, *D* }

Ejemplo de las notas del curso de C. Leiserson del MIT

Algoritmo de Dijkstra

- Inicializar $d[s] \leftarrow 0$ y $d[v] \leftarrow \infty$ para todo $v \in V - \{s\}$ establece $d[v] \geq \delta(s,v)$ para todo $v \in V$ y esta invariante se mantiene durante cualquier secuencia de pasos de relajación.

- Como DIJKSTRA elige siempre el vértice más cercano en V-S para agregar al conjunto S, decimos que usa una estratégia glotona.

- El algoritmo de Dijkstra, ejecutado en una gráfica con peso dirigido G=(V,E) con funciones de peso w en sus aristas no-negativas y fuente s, termina con $d[u] = \delta(s,u)$ para todos los vértices $u \in V$.

Algoritmo de Dijkstra: análisis

- El algoritmo de Dijkstra resuelve el problema de encontrar los caminos más cortos para una sóla fuente en gráficas con peso no-negativo.

```
DIJKSTRA(G, w, s)
 1  INITIALIZE-SINGLE-SOURCE(G, s)
 2  S ← ∅
 3  Q ← V[G]
 4  while Q ≠ ∅
 5      do u ← EXTRACT-MIN
 6         S ← S ∪ {u}
 7         for each vertex v ∈
 8             do RELAX(u, v,
```

← Θ(V)

```
INITIALIZE-SINGLE-SOURCE (G, s)
 1  for each vertex v ∈ V[G]
 2      do d[v] ← ∞
 3         π[v] ← NIL
 4  d[s] ← 0
```

Algoritmo de Dijkstra: análisis

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5      do u ← EXTRACT-MIN(Q)
6         S ← S ∪ {u}
7         for each vertex v ∈ Adj[u]
8             do RELAX(u, v, w)
```

Mantiene la cola de prioridad min con tres operaciones:

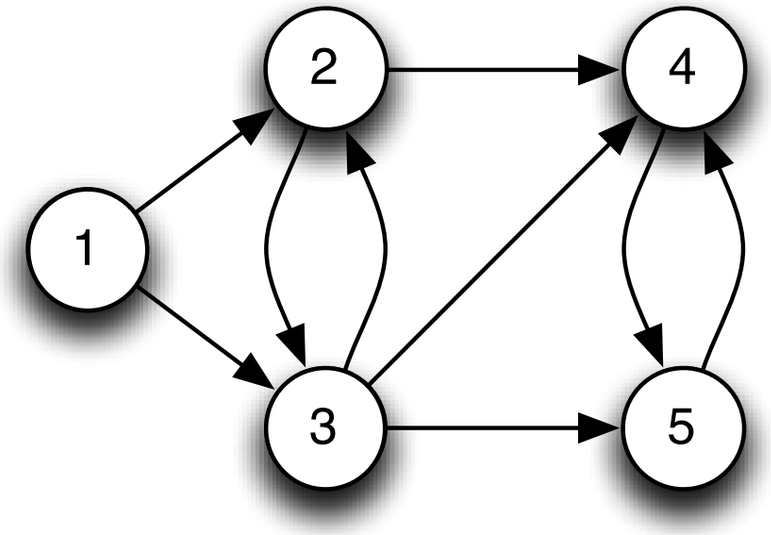
- INSERT (línea 3)
- EXTRACT-MIN (línea 5)
- DECREASE-KEY (línea 8)

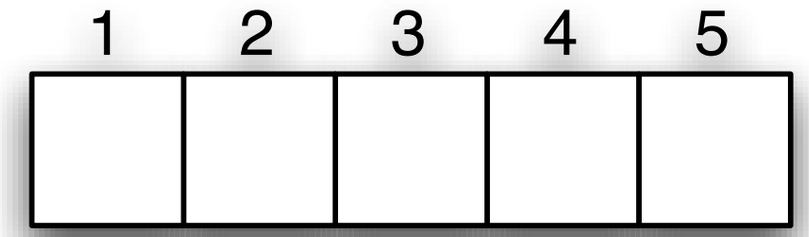
¿Cuántas veces por cada vértice se ejecutan estas funciones?

- INSERT - V
- EXTRACT-MIN - V
- DECREASE-KEY - E en el peor caso.

Algoritmo de Dijkstra: análisis

- El tiempo de ejecución dependerá de la forma de implementar la cola de prioridad. Vemos 2 casos:





- Vértices numerados de 1 a V en un arreglo.
- INSERT O(1)
- DECREASE-KEY O(1)
- EXTRACT-MIN O(V)

Total: $O(V^2+E) = O(V^2)$

Algoritmo de Dijkstra: análisis

- Si la cola de prioridad se implementa con un montículo binario:
  - Tiempo para construir el montículo $O(V)$
  - EXTRACT-MIN $O(\lg V) \times V$ veces
  - DECREASE-KEY $O(\lg V) \times E$ veces a lo mas.

- Total:

  $O((V+E)\lg V) = O(E \lg V)$ si todos los vértices son alcanzables de la raíz.