Problema de los caminos de costo mínimo partiendo desde un solo origen

(Single-Source Shortest Paths)

Bibliografía: "Introduction to Algorithms". Second Edition.

The MIT Press. Massachusetts Institute of Technology. Cambridge, Massachusetts 02142.

http://mitpress.mit.edu

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein

Facultad de Matemática y Computación
Universidad de La Habana

MODELACIÓN DEL PROBLEMA

- Las calles (**arcos del grafo**) tienen orientación $\Rightarrow$ **grafo dirigido**
- Hay algunas calles mejores (calzadas) que otras (calles estrechas) $\Rightarrow$ **grafo ponderado**
- Las esquinas serían los **vértices del grafo**

PROBLEMA

Un estudiante desea encontrar la ruta más «económica» posible, para ir desde su casa a la Universidad de la Habana.

Cómo podríamos encontrar la misma?

Problema de los caminos de costo mínimo

PROBLEMA

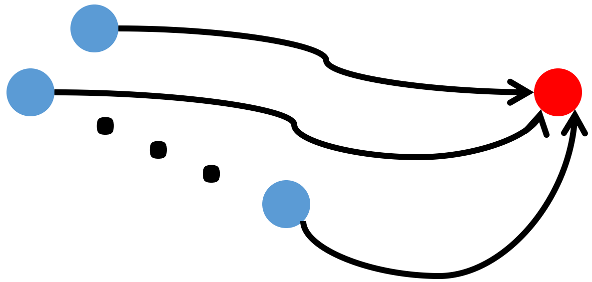
"Caminos de costo mínimo a partir de solo un origen"

Dado un grafo ***G*** *= (V, E), dirigido y ponderado,* encontrar el ***camino de costo mínimo*** desde un vértice origen $s \in V$ hasta los restantes vértices de *V*

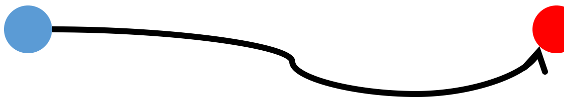
Otros problemas que pueden ser resueltos

Colateralmente, otros problemas pueden ser resueltos a partir del problema anterior :

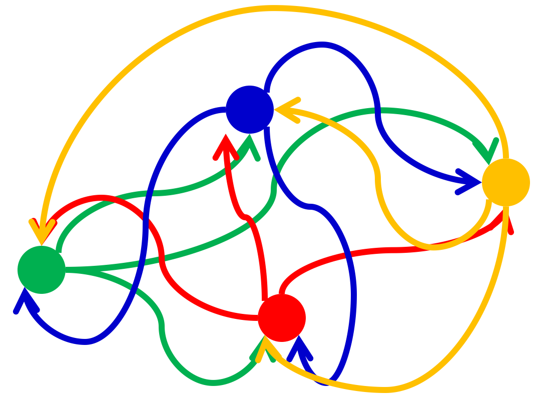
- **Caminos de costo mínimo hacia un único destino** *(Single-destination shortest-paths problem)*: Encontrar el camino de costo mínimo desde cada vértice del grafo a un vértice de destino v de G



- **Camino de costo mínimo entre un determinado par de vértices** *(Single-pair shortest-path problem):* Hallar el camino de costo mínimo entre un par de vértices dados u y v de G



- **Camino de costo mínimo entre cualquier par de vértices** *(All-pairs shortest-paths problem):* Encontrar el camino de costo mínimo entre u y v para cualquier par de vértices u y v de G



Consideraciones para el problema de los caminos de costo mínimo

En un problema de ***caminos de costo mínimo*** (*shortest-paths problem)* se tienen en cuenta Grafos del tipo:

$G=(V, E)$

- **Dirigido**,
- **Ponderado**: con función de ***costo*** $w : E \to \mathrm{R}$

  $w((u, v) \in E) = \boldsymbol{costo} \in R$

- $s \in V$ es el **vértice origen**

camino de costo mínimo

Sea *p* un camino de la forma, $p = \{v_0, \ ..., \ v_k\}$, $v_i \in V$, *el* ***costo*** ***del camino*** *se define de la siguiente forma:*

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

El **costo** del ***camino de costo mínimo*** de *u* a *v*, ***δ(u, v)***, se define de la siguiente forma:

$$\delta(u,v) = \begin{cases} min\left\{w(p): u \overset{p}{\rightarrow} v\right\} & \text{- si existe un camino de } u \text{ a } v \\ \infty & \text{- en otro caso} \end{cases}$$

Un ***camino de costo mínimo*** de *u* a *v* es cualquier camino *p* de *u* a *v* tal que ***w(p) = (δ(u, v))***

El camino de costo mínimo NO ES UNICO !!

Subestructura óptima de un camino de costo mínimo

Los algoritmos relacionados con el **problema de los caminos de costo mínimo** se basan, fundamentalmente, en la siguiente **PROPIEDAD:**

Todo **camino de costo mínimo** entre dos vértices, contiene dentro de si, otros **caminos de costo mínimo**

Lema 24.1: ***Los subcaminos de los caminos de costo mínimo, son también caminos de costo mínimo***

Sea *G=(V,E),* dirigido, ponderado, con función de costo *w: E → R*, definida sobre G. Sea $p = \{v_1, v_2, \ldots, v_k\}$ un **camino de costo mínimo** entre v_1 y v_k, $\forall i, j: 1 \leq i \leq j \leq k$, sea

$p_{ij} = \{v_i, v_{i+1}, \ldots, v_j\}$ un **subcamino de *p*** entre v_i y v_j, entonces, **p_{ij} es un camino de costo mínimo** de v_i a v_j

Demostración – Lema 24. 1:

Descompongamos *p* de la forma

$$p : v_1 \xrightarrow{p_{1i}} v_i \xrightarrow{p_{ij}} v_j \xrightarrow{p_{jk}} v_k$$

supongamos que p_{ij} NO es un camino de costo mínimo entre v_i y v_j, entonces existirá un camino p'_{ij} de v_i a v_j que cumplirá w(p'_{ij}) < w(p_{ij}) y

$$p' = v_1 \xrightarrow{p_{1i}} v_i \xrightarrow{p'_{ij}} v_j \xrightarrow{p_{jk}} v_k$$

será un camino de v_1 a v_k cuyo costo

$w(p') = w(p_{1i}) + w(p'_{ij}) + w(p_{jk}) < w(p)$ ***(contradicción)***

pues *p* es de costo mínimo de v_1 a v_k

Arcos de costo negativo

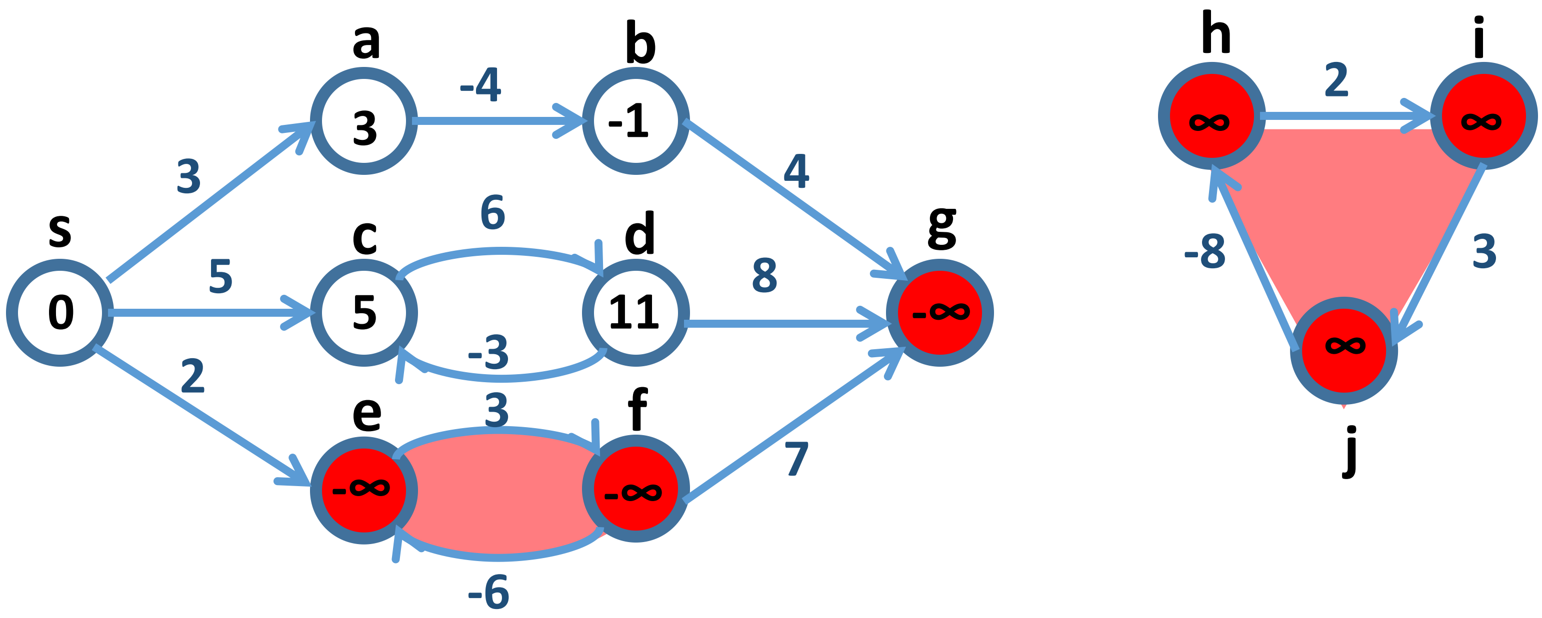
En este tipo de problemas, no se excluye la posibilidad de que en el grafo **existan arcos con costo negativo**

Si G, **NO tiene ciclos de costo negativo, alcanzables desde *s***, entonces, $\forall v \in V$, alcanzable desde s, $\delta(s, v)$ está siempre definido, aún cuando dicho valor puede ser negativo

Si existe un ciclo de costo negativo alcanzable desde *s*, entonces, los ***caminos de costo mínimo* no están bien definidos**

- Si existe un ciclo de costo negativo sobre algún camino de *s* a *v*, entonces se define $\delta(s, v) = -\infty$
- Si *v* NO es alcanzable desde s, entonces $\delta(s, v) = \infty$

Ciclos de costo negativo



- Entre los vértices *e* y *f* se forma un ciclo de costo negativo, alcanzable desde s, entonces dichos vértices tienen como costo del camino de costo mínimo del origen a ellos el valor -∞.
- Como el vértice *g* se alcanza desde un vértice con costo -∞, para el camino de costo mínimo del origen a él, entonces el costo del camino de costo mínimo de *s* a *g* es también -∞.
- Los vértices *h*, *i* e *j* no son alcanzables desde *s*, entonces el costo del camino de costo mínimo de *s* a ellos es ∞ aún cuando entre ellos se forme un ciclo de costo negativo

Algoritmos que dan solución al problema

- **Algoritmo de Dijkstra:**

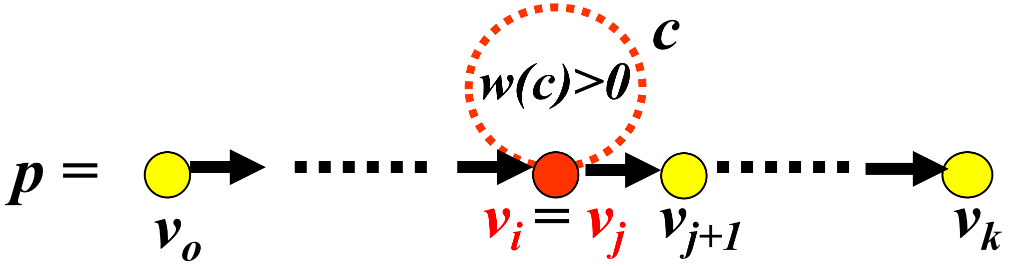
Asume que el **costo de todos los arcos del grafo son NO negativos**

- **Algoritmo DAG y Bellman – Ford:**

En el grafo **pueden haber arcos de costo negativo**. Resuelve el problema de manera correcta, siempre y cuando en dicho **grafo NO existan ciclos de costo negativo alcanzables desde el origen**

Existencia de ciclos en el camino de costo mínimo

Podría ser *p* un camino de costo mínimo ?



$p = <v0, v1, \ldots, vi, \ldots\ldots, vj=vi, vj+1, \ldots\ldots vk>$

***R/* NO**

$p' = <v0, v1, \ldots, vj, vj+1, \ldots\ldots vk>$

$w(p') = w(p) - w(c) < w(p)$, *por tanto*

$p \rightarrow$ ***NO ES DE COSTO MINIMO***

.... y si w(c) = 0 ?

Existencia de ciclos en el camino de costo mínimo

R/ En tal caso, **SI**

- Los **ciclos de costo 0** se pueden extraer de cualquier **camino** sin que se altere el costo del mismo, es decir, el **camino** que resulta tras extraer el **ciclo**, tiene el mismo costo que tenía el **camino** que inicialmente lo incluía

- Por tanto, si hay **un camino de costo mínimo** desde ***s*** hasta otro vértice ***v*** del grafo que contiene **un ciclo de costo 0**, entonces existe **otro camino de costo mínimo** de igual costo que no contiene a dicho ciclo

Por consiguiente, y sin pérdida de generalidad, podemos asumir que:

cuando estamos encontrando caminos de costo mínimo, en ellos, no hay ciclos

Representando caminos de costo mínimo

Además del costo **del camino de costo mínimo** puede que sea necesario conocer los vértices que forman parte de dicho **camino**

Dado un grafo $G = (V, E)$, para cada vértice $\boldsymbol{v \in V}$ se establece su **predecesor** $\boldsymbol{\pi[v]}$ en algún **camino de costo mínimo**:

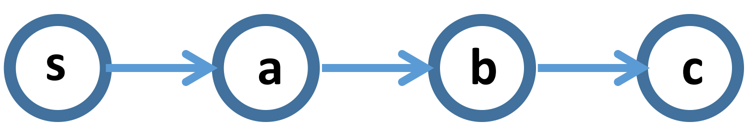
$$\boldsymbol{\pi[v]=r: r \in V \quad ó \quad \pi[v]= null}$$

Los algoritmos para determinar **caminos de costo mínimo** calculan los valores $\boldsymbol{\pi[v]}$ para cada $\boldsymbol{v \in V}$, por tanto, siguiendo la ***cadena de predecesores*** a partir de un $\boldsymbol{v}$ dado, se puede obtener la secuencia de vértices en el **camino de costo mínimo** de $\boldsymbol{s}$ a $\boldsymbol{v}$

Determinar secuencia de vértices en un camino

PRINT-PATH(*G*, *s*, *v*)

```
if v = s // caso base
    then print s
else if π[v] = NULL
    then print "there is no path from" s "to" v
else PRINT-PATH(G, s, π[v])
        print v // imprime el camino desde s hasta π[v]
                // antes de imprimir al propio v
```



$\pi[c]=b$
$\pi[b]=a$
$\pi[a]=s$
$\pi[s]=null$

Secuencia de llamados

PRINT-PATH(*G*, *s*, *c*)
PRINT-PATH(*G*, *s*, *b*)
PRINT-PATH(*G*, *s*, *a*)
PRINT-PATH(*G*, *s*, *s*)
s
a
b
c

Árbol de los caminos de costo mínimo

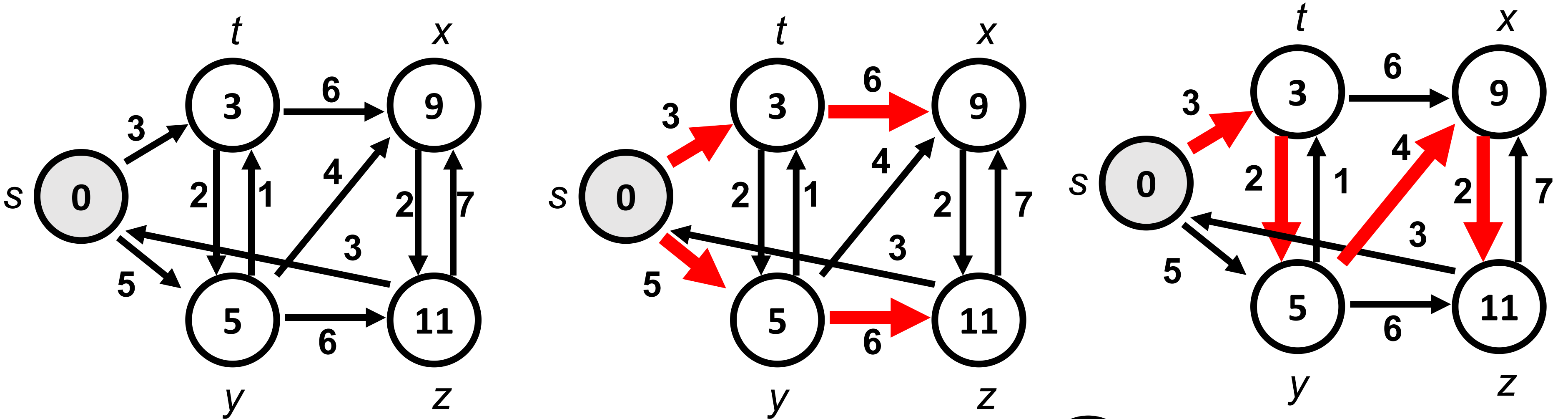
- Sea $G = (V, E)$ un grafo dirigido y ponderado con función de costo $w: E \rightarrow R$
- En G no existen **ciclos de costo negativo** alcanzables desde el origen
- Sea $s \in V$ el origen

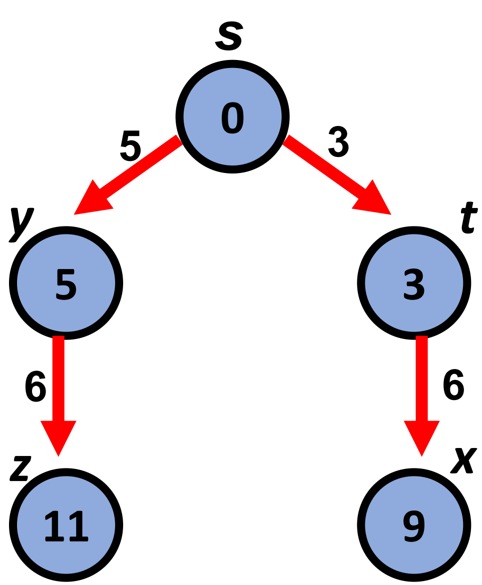
Un **árbol de caminos de costo mínimo** con raíz s es un **subgrafo dirigido** de G, $G'=(V', E')$ donde $V' \subseteq V$ y $E \subseteq E'$, tal que se cumple:

- V' es el conjunto de vértices alcanzables desde s en G
- G' forma un árbol con raíz s
- $\forall\, v \in V$, el **único camino simple de s a v en G'** es UN **camino de costo mínimo** de s a v en G

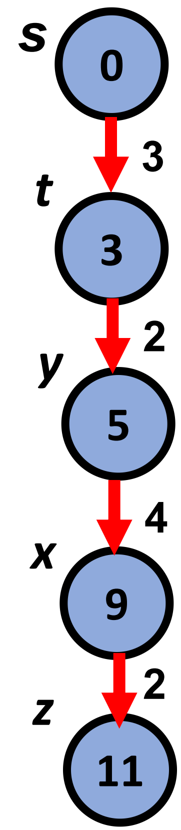
Ni los **caminos de costo mínimo** ni el "***árbol de caminos de costo mínimo***" son necesariamente UNICOS

Árbol de los caminos de costo mínimo





un árbol de caminos de costo mínimo



OTRO árbol de caminos de costo mínimo

Técnica de *relajación* (RELAX)

d[v] ***- costo mínimo estimado***:

Representa la **cota superior** **para el costo del camino de costo mínimo** de ***s*** a ***v***

d[v] y **π[*v*]** se inicializan en *O(V)* por el siguiente método:

INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex $v \in V[G]$

  do $\boxed{d[v]} \leftarrow \infty$

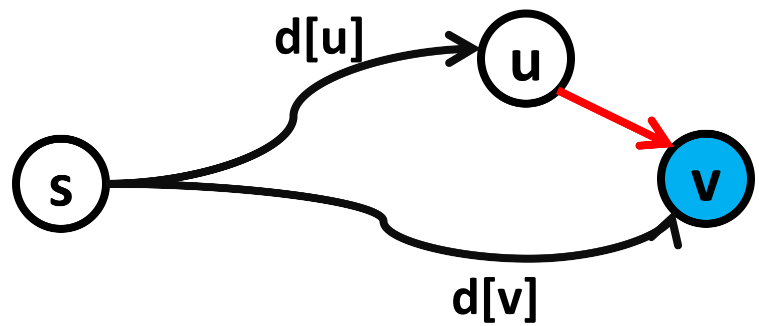
    $\boxed{\pi[v]} \leftarrow$ **NULL**

$\boxed{d[s]} \leftarrow 0$

Técnica de *relajación* (RELAX)

***RELAX* se aplica sobre los arcos**

RELAX sobre ***(u, v)*:** Su funcionalidad puede resumirse en función de la siguiente interrogante: **podemos mejorar el costo del camino** para ir de ***s*** a ***v*****,** hasta ahora asumido como mínimo (**d[v]**), pasando por ***u*** ? Si se puede, entonces: actualizar ***d[v]*** y ***π[v]***




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

Consideraciones sobre los algoritmos

En los algoritmos que resuelven el **problema de los caminos de costo mínimo con un solo origen**:

- Las inicializaciones se hacen mediante **INITIALIZE-SINGLE-SOURCE**
- El **RELAX** es quien único puede hacer que **d[v]** y **π[v]** modifiquen su valor
- Los algoritmos difieren entre sí en el **número de veces y en el orden, en que se hacen los RELAX sobre los arcos**:

  1. En **Dijkstra** y en el **DAG** a cada arco se le hace **RELAX, exactamente, una sola vez**
  2. En el **Bellman-Ford**, sobre cada arco se aplica **RELAX, varias veces**

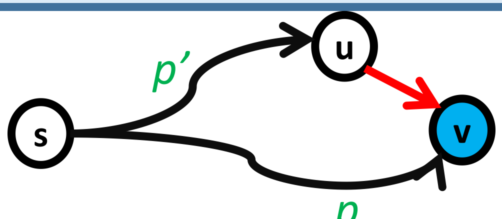
Propiedades

Consideraciones: Sea ***G=(V, E)*** **dirigido** y **ponderado** con ***función de costo*** *w: E → R* y sea ***s*** el vértice origen. Se asume que ***G*** se ha inicializado por **INITIALIZE-SINGLE-SOURCE(*G*, *s*)** y que la única forma en que varían los **d[v]** estimados y los valores de **π[v]** es a través del **RELAX,** $\forall v \in V$

- **Desigualdad Triangular (Lema 24.10)**

Para todo arco **(u, v)** $\in$ E se cumple $\delta(s, v) \leq \delta(s, u) + w(u, v)$

Demostración:



Supongamos que existe un camino ***p*** de **costo mínimo de *s* a *v*.** El costo de ***p*** no puede ser mayor que el de cualquier otro camino de ***s*** a ***v***. En particular, no puede tener un **costo mayor** que un camino ***p'***= «*un camino de costo mínimo de **s** a **u***» + ***(u, v)***

Por tanto $\delta\ (p) \leq \delta\ (p') \Rightarrow \delta(s, v) \leq \delta(s, u) + w(u, v)$

Propiedades

- **Propiedad de la Cota Superior (Lema 24.11)**

Para todo $\boldsymbol{v \in V}$ se cumple $\boldsymbol{d[v] \geq \delta(s, v)}$ y una vez que **d[v]** alcanza el valor $\boldsymbol{\delta(s, v)}$, este no varía nunca más.

Demostración

Demostremos *por inducción*, sobre la **cantidad de pasos de *relajación*,** que se cumple la invariante $\boldsymbol{d[v] \geq \delta(s, v)}, \forall \boldsymbol{v \in V},$

caso base:

Tras las inicializaciones se cumplirá la invariante: $\boldsymbol{d[v] \geq \delta(s, v)}$ pues

$\boldsymbol{d[v] = \infty, \forall\, v \in V - \{s\} \Rightarrow d[v] \geq \delta(s, v)}$

Para el caso del origen $\mathbf{d[s] = 0 \geq \delta(s, s)}$**:**

- Si $\boldsymbol{s}$ está sobre un ciclo de costo negativo, $\boldsymbol{\delta(s, s)} = \mathbf{-\infty} \Rightarrow \mathbf{0 > -\infty}$
- En otro caso, $\boldsymbol{\delta(s, s)} = \mathbf{0} \Rightarrow \mathbf{0 = 0}$

Propiedades

Para el **paso de inducción**, consideremos el momento en que se hace ***RELAX*** a un arco ***(u, v)***:

Por **hipótesis de Inducción**: $d[x] \geq \delta(s, x)$, $\forall x \in V$, antes de hacer *RELAX(u, v, w)*

Tras el *RELAX(u, v, w)* **el único valor que podría cambiar** es $d[v]$

Si cambiase, $d[v] = d[u] + w(u, v)$

$\geq \delta(s, u) + w(u, v)$ (por Hipótesis de Inducción)

$\geq \delta(s, v)$ (por P. de la Desigualdad Triangular)

Por tanto, se cumple la invariante

Además, una vez que se alcanza la igualdad $d[v] = \delta(s, v)$, *d[v] alcanzó el valor de su cota inferior* y este valor nunca cambiará pues hemos demostrado que $d[v] \geq \delta(s, v)$ y la única modificación que podría hacer el RELAX al valor de d[v] sería disminuirlo

Propiedades

- **Propiedad de la no existencia de camino (Corolario 24.12)**

Si no existe camino de s a v, entonces el valor de d[v] se mantendrá invariante y se cumplirá $d[v] = \delta(s, v) = \infty$

Demostración:

Por la **Propiedad de la Cota Superior** (Lema 24.11) tenemos

$\boldsymbol{d[v] = \infty \geq \delta(s, v)}$ **y por tanto,** $d[v] = \infty = \delta(s, v)$

Propiedades

- **Lema 24.13**

Sea $(u, v) \in E$. Inmediatamente después de hacer **RELAX(u, v, w),** se cumple, $d[v] \leq d[u] + w(u, v)$

Demostración

Si antes de hacer **RELAX(u, v, w):**

- $d[v] > d[u] + w(u, v) \Rightarrow$ tras hacerlo, $d[v] \mathbf{=} d[u] + w(u, v)$
- $d[v] \leq d[u] + w(u, v) \Rightarrow$ tras hacerlo, ni d[u] ni d[v] cambian, por tanto, la desigualdad $d[v] \mathbf{\leq} d[u] + w(u, v)$ se mantiene

Propiedades

- **Propiedad de la convergencia (Lema 24.14)**

Si $s \rightsquigarrow u \rightarrow v$ es un **camino de costo mínimo** en *G*, $u,v \in V$. Si se alcanza la igualdad $d[u] = \delta(s, u)$ en cualquier momento antes de hacer *RELAX* sobre el arco (u, v), entonces, después de haberlo hecho, $d[v] = \delta(s, v)$ y dicha igualdad se mantiene en lo sucesivo

Demostración,

Si antes de hacer RELAX(u, v, w) se alcanza la igualdad $\boldsymbol{d[u] = \delta(s, u)}$ entonces la misma se mantiene en lo sucesivo (por **P. Cota Superior.** ***Lema 24.11***)

En particular, tras hacer RELAX(u, v, w):

$$d[v] \leq d[u] + w(u, v) \qquad \text{(por el } \textbf{Lema 24.13}\text{)}$$
$$= \delta(s, u) + w(u, v)$$
$$= \delta(s, v) \qquad \text{(por el } \textbf{Lema 24.1} \text{ pues por Hipót. } s \sim u \rightarrow v \textbf{ es de costo mínimo}\text{)}$$

Por tanto, $d[v] \leq \delta(s, v)$ **(1)**

Por la **Propiedad de la Cota Superior** (Lema 24.11): $d[v] \geq \delta(s, v)$ **(2)**

De **(1)** y **(2)**, $d[v] = \delta(s, v)$ y la igualdad se mantiene en lo sucesivo

Propiedades

- **Propiedad del RELAX (Lema 24.15)**

Si p = <v_0, v_1,..., v_k> es un **camino de costo mínimo** de $s=v_0$ a v_k y supongamos que a los arcos de p se les aplica *RELAX* en el siguiente orden: (v_0, v_1), (v_1, v_2),..., (v_{k-1}, v_k), entonces $d[v_k] = \delta(s, v_k)$ después de estas "relajaciones". Esta propiedad se cumple, incluso, cuando se mezcle el RELAX sobre otros arcos con los arcos de p. (Note que k puede ser = 0, 1, ..., |V|-1)

Demostración:

Demostraremos, por inducción, que tras el ***i-ésimo*** *RELAX* sobre un arco de p, se cumple, $d[v_i] = \delta(s, v_i)$

Propiedades

caso base: i=0

Antes de hacer **RELAX** a ningún arco de p, tenemos, como resultado de la inicialización, $d[v_0] = d[s] = 0 = \delta(s, s)$ y por la **Propiedad de la Cota Superior**, el valor de d[s] no cambia en lo sucesivo

Para el **paso de inducción** asumimos $d[v_{i-1}] = \delta(s, v_{i-1})$ y analicemos que sucede tras hacer **RELAX** sobre el arco (v_{i-1}, v_i):

Por la **Propiedad de la Convergencia**, $d[v_i] = \delta(s, v_i)$ y la igualdad se mantiene en lo sucesivo

Preliminares

Convenios:

$\forall a \in \Re, a + \infty = \infty + a = \infty$ $\qquad$ $a + (-\infty) = (-\infty) + a = -\infty$

En los algoritmos se asume:

- *G* **dirigido** y representado mediante una **Lista de Adyacencia**
- Junto a cada **arco** se almacena su **costo** y por tanto, al recorrer la **Lista de Adyacencia** dicho valor se puede obtener en *O(1)*

Algoritmo DAG

Caminos de costo mínimo desde un vértice origen en *Grafos Dirigidos y Acíclicos (DAG)*

(Single-source shortest paths in Directed Acyclic Graphs)

Objetivo:

Calcular el **camino de costo mínimo** desde un vértice origen hacia los restantes vértices del grafo

Complejidad Temporal:

$O(|V|+|E|)$

Estrategia:

Relajación de los arcos de un **DAG ponderado** *G = (V, E)* a partir de un **orden topológico** de sus vértices

Algoritmo DAG

ASPECTOS A RESALTAR:

Los caminos de costo mínimo están siempre bien definidos en un **DAG**, pues aunque hayan **arcos de costo negativo**, por su condición de **grafo acíclico**, se garantiza que **en el grafo NO HAY ciclos de costo negativo**

PRECISIONES SOBRE EL ALGORITMO:

1. Establecer un **orden topológico sobre el DAG**

2. Para cualesquiera ***u*, *v*** $\in$ ***V*, si existe un camino entre *u* y *v*, entonces *u* precede a *v* en el orden topológico**

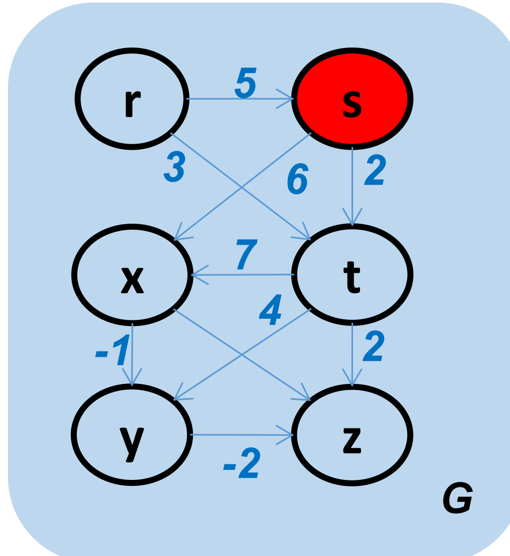
3. El algoritmo hace exactamente **una sola pasada sobre los vértices en el orden topológico establecido**

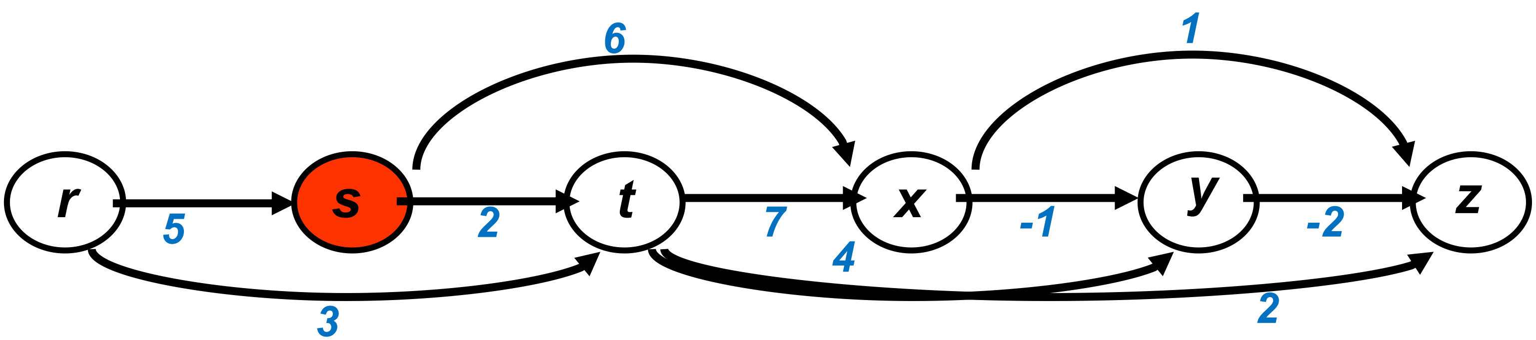
4. Cuando se procesa **un vértice particular, se aplica *RELAX* a cada arco que parte de dicho vértice**

Algoritmo DAG

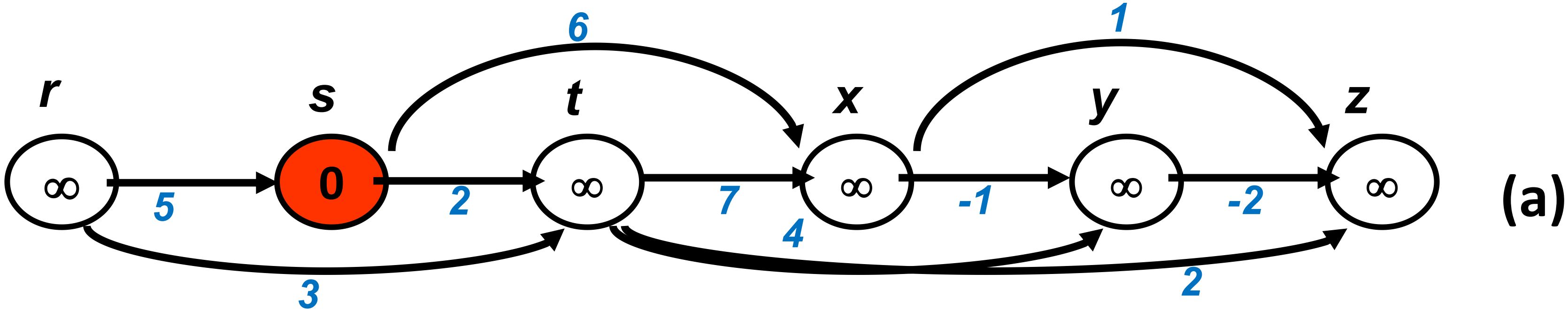
```
DAG-SHORTEST-PATHS(G, w, s)
    topologically sort the vertices of G
    INITIALIZE-SINGLE-SOURCE(G, s)
    for each vertex u, taken in topologically sorted order
        do for each vertex v ∈ Adj[u]
            do RELAX(u, v, w)
```

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



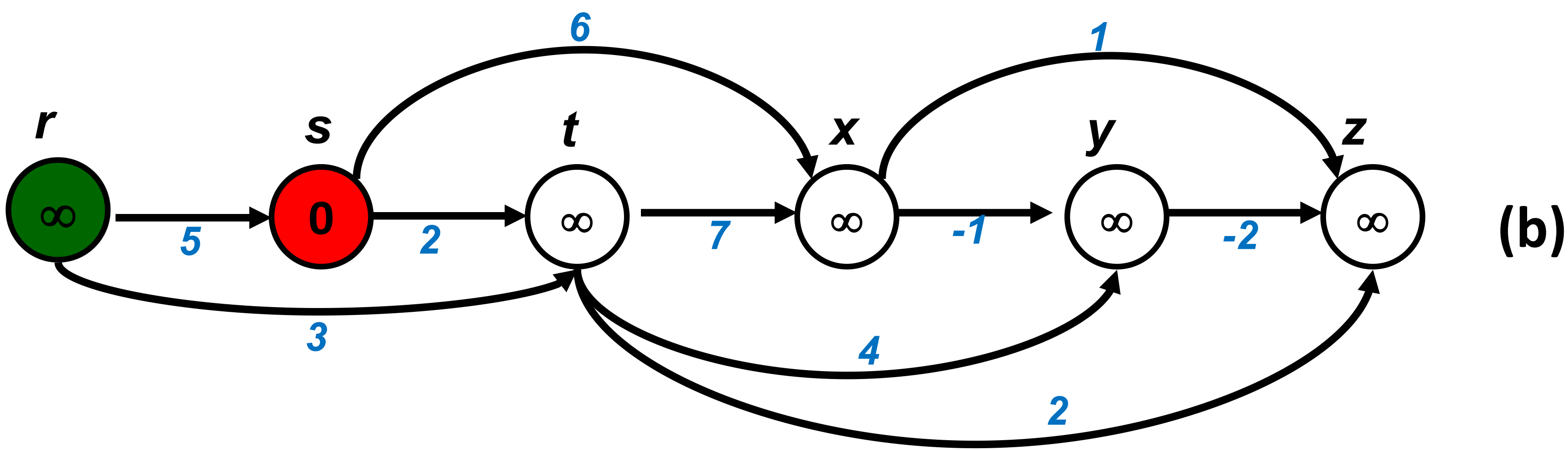


Los vértices de G están topológicamente ordenados de izquierda a derecha

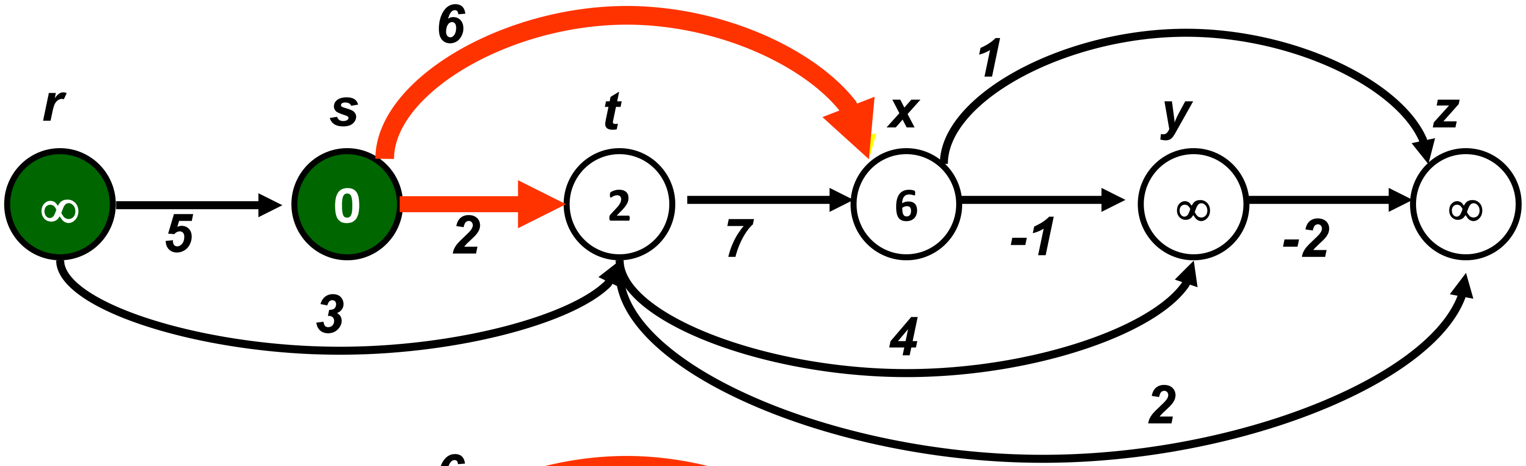


(a) La situación del grafo antes de la primera iteración del ciclo ***for*** comprendido entre las líneas 3-5.

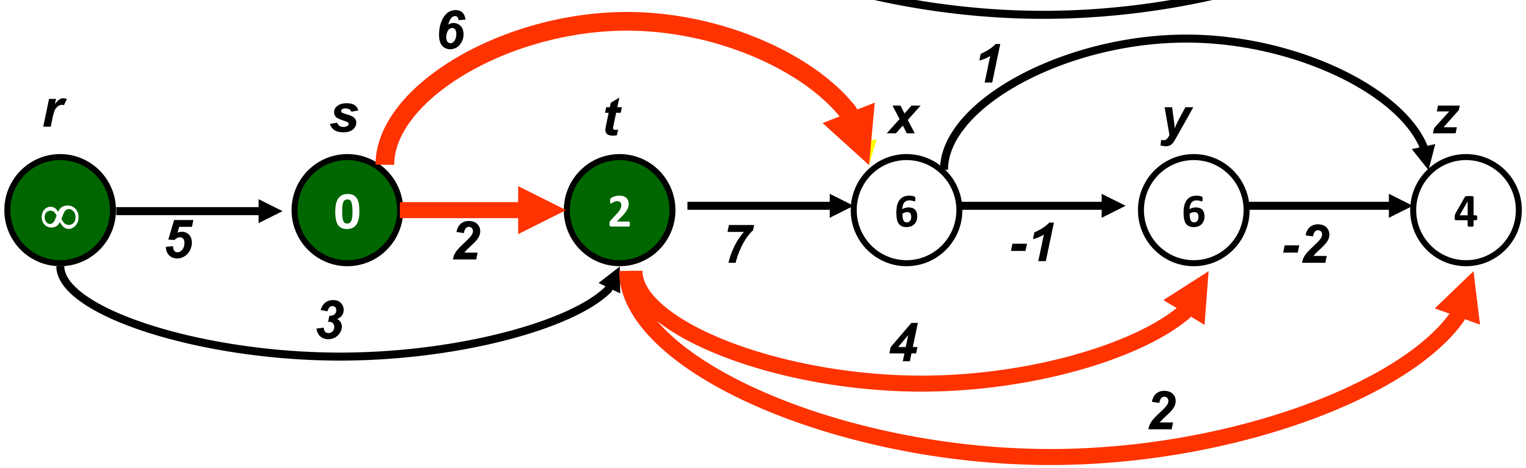
Los vértices están topológicamente ordenados de izquierda a derecha. s: origen. El interior de los vértices refleja el valor de d[]



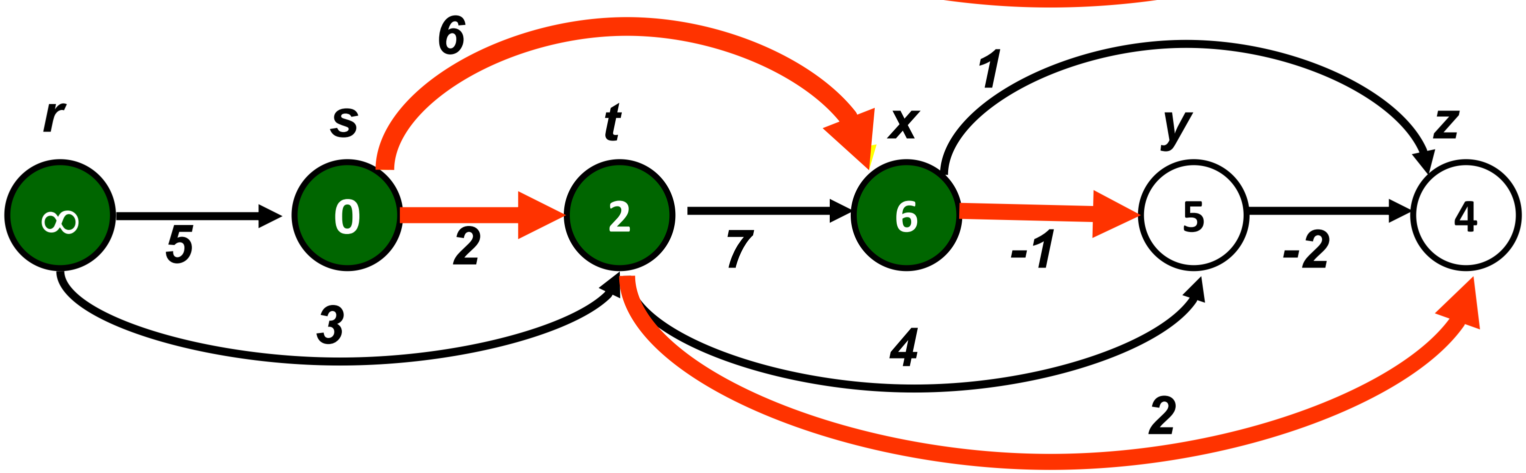
En lo sucesivo, los arcos en rojo reflejan los valores de π


r
s
t
x
y
z
∞
0
2
6
∞
∞
6
1
5
2
7
-1
-2
3
4
2

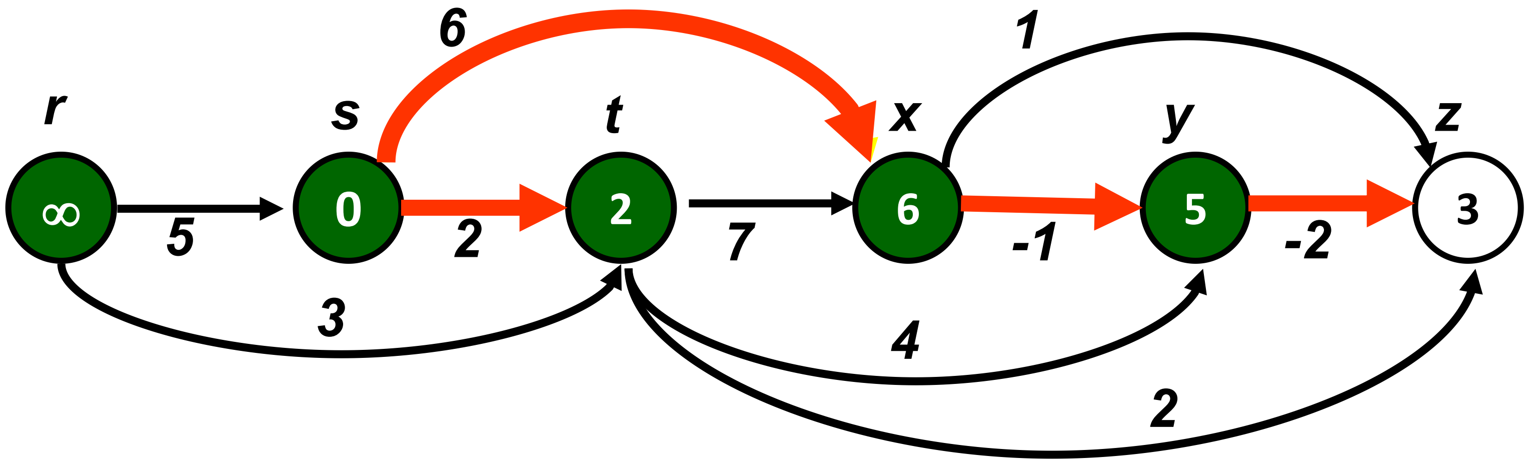
(c)


r
s
t
x
y
z
∞
0
2
6
6
4
6
1
5
2
7
-1
-2
3
4
2

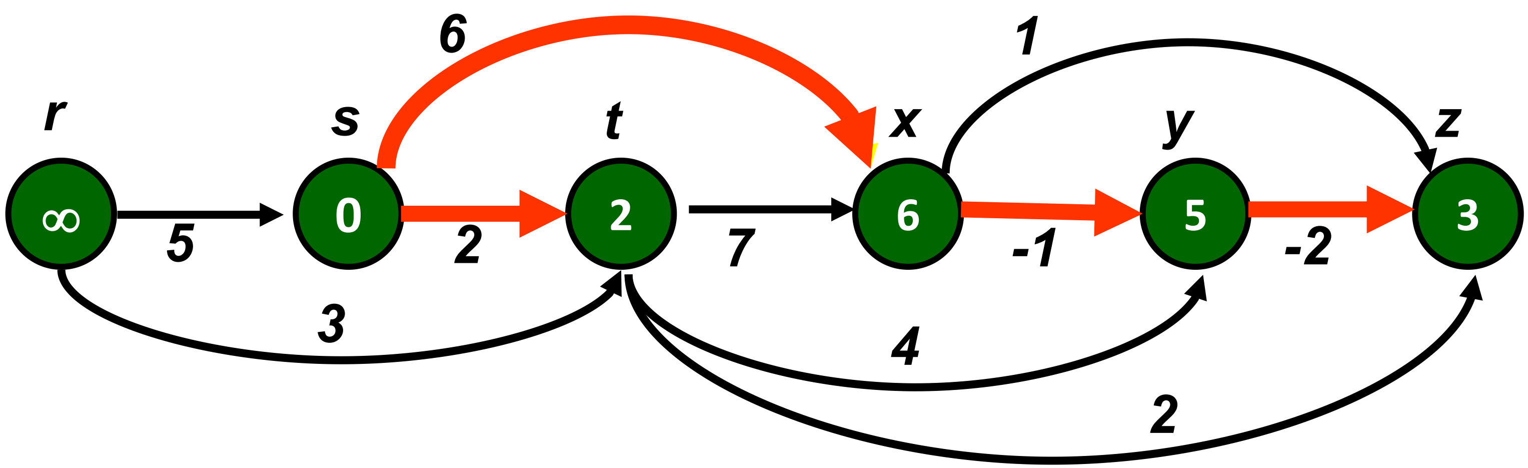
(d)


r
s
t
x
y
z
∞
0
2
6
5
4
6
1
5
2
7
-1
-2
3
4
2

(e)



(f)



(g)

(b)-(g) La situación tras cada iteración del ciclo ***for*** comprendido entre las líneas 3-5. El nuevo vértice que aparece en verde, en cada iteración, es el usado como *u* en dicha iteración. Los valores mostrados en el esquema (g) son los valores finales calculados por el algoritmo

Complejidad temporal – Algoritmo DAG

- topologically sort the vertices of *G* → ***O(|V| + |E|)***
- INITIALIZE-SINGLE-SOURCE(*G*, *s*) → ***O(|V|)***

- Se itera una vez por cada vértice de G en el ciclo `for` de las líneas 3-5 O(|V|):

  - Para cada vértice, los arcos que salen del mismo son analizados **una sola vez**. Por tanto, en total se analizan |E| arcos

  - Cada iteración del ciclo interior (línea 5) se hace en O(1)

→ ***O(|V| + |E|)***

Por tanto, el tiempo **TOTAL** de ejecución del algoritmo es ***O(|V|+|E|)*** para una representación por Listas de Adyacencia del Grafo

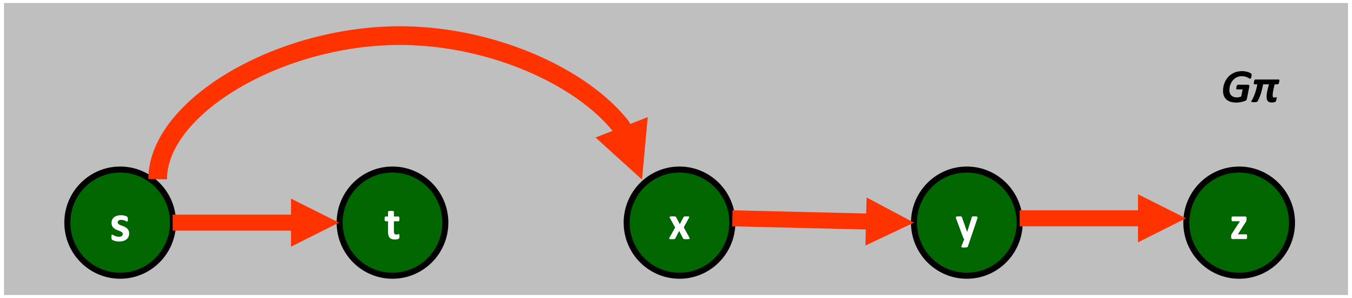
Subgrafo predecesor

Gπ* = (*Vπ*, *Eπ*)** es el ***subgrafo predecesor generado por los valores de ***π***, donde:

- $V\pi = \{v \in V : \pi[v] \neq null\} \cup \{s\}$

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

Gπ* es un *árbol* con raíz *s



Correctitud del DAG

Teorema 24.5

Dado un grafo dirigido, ponderado y acíclico G=(V, E), donde $s \in V$ es el origen, entonces, al finalizar la ejecución del algoritmo **DAG-SHORTEST-PATHS**, ***d*[*v*] = *δ*(*s*, *v*)** para todo ***v*** $\in$ ***V*** y el subgrafo predecesor ***Gπ*** que se forma **es un árbol de caminos de costo mínimo**

Correctitud del DAG

Demostración

Demostremos que $d[v] = \delta(s, v)$ para todo $v \in V$ al terminar el DAG

i) Si v no es alcanzable desde $s \Rightarrow d[v] = \delta(s, v) = \infty$ por **Cor. 24.12.**
ii) Supongamos que v es alcanzable desde s, por tanto hay un <u>camino de costo mínimo</u> $p = <v_0, v_1,..., v_k>$, donde $v_0 = s$ y $v_k = v$

Como los v … os arcos
en p se les …

Por la **Prop** … nar el
algoritmo, p…

Lema 24.17
Una vez que se alcanza la igualdad: $d[v] = \delta(s, v)$
$\forall vV$, el subgrafo predecesor es un árbol de caminos
de costo mínimo que tiene como raíz al vértice
origen *s*

Por **Lema 24.17** (ver demostración en el I. to A.) **-** *Gπ* es un ***árbol de caminos de costo mínimo***

RESUMEN DEL MARCO TEORICO PARA EL PROBLEMA *de los Caminos de Costo Mínimo con un solo Origen*

- **Lema 24.1*:*** ***Los subcaminos de los caminos de costo mínimo, son también caminos de costo mínimo***

- **Desigualdad Triangular (Lema 24.10)**

Para todo arco (u, v) $\in$ E se cumple $\delta(s, v) \leq \delta(s, u) + w(u, v)$

- **Propiedad de la Cota Superior (Lema 24.11)**

Para todo $v \in V$ se cumple $\boldsymbol{d[v] \geq \delta(s, v)}$ y una vez que d[v] alcanza el valor $\delta(s, v)$, este no varía nunca más.

- **Propiedad de la no existencia de camino (Corolario 24.12)**

Si no existe camino de s a v, entonces el valor de d[v] se mantendrá invariante y se cumplirá d[v] = δ(s, v) = ∞

- **Lema 24.13**

Sea (u, v)$\in$E. Inmediatamente después de hacer **RELAX(u, v, w),** se cumple, $d[v] \leq d[u] + w(u, v)$

RESUMEN DEL MARCO TEORICO PARA EL PROBLEMA *de los Caminos de Costo Mínimo con un solo Origen*

- **Propiedad de la convergencia (Lema 24.14)**

Si $s \rightsquigarrow u \rightarrow v$ es un **camino de costo mínimo** en *G*, $u,v \in V$. Si se alcanza la igualdad d[u] = $\delta(s, u)$ en cualquier momento antes de hacer *RELAX* sobre el arco (u, v), entonces, después de haberlo hecho, d[v] = $\delta(s, v)$ y dicha igualdad se mantiene en lo sucesivo

- **Propiedad del RELAX (Lema 24.15)**

Si $p = <v_0, v_1, \ldots, v_k>$ es un **camino de costo mínimo** de $s=v_0$ a v_k y supongamos que a los arcos de *p* se les aplica *RELAX* en el siguiente orden: $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, entonces $d[v_k] = \delta(s, v_k)$ después de estas "relajaciones". Esta propiedad se cumple, incluso, cuando se mezcle el RELAX sobre otros arcos con los arcos de *p*. (Note que k puede ser = 0, 1, ..., |V|-1)

Algoritmo de Dijkstra

Resuelve el problema en **grafos dirigidos y ponderados** ***G = (V, E)****,* donde:

todos los arcos tienen un costo no negativo
por tanto, se asume que $\mathbf{w(u, v) \geq 0 \ \forall \ (u, v) \in E}$

IDEA GENERAL del Algoritmo de Dijkstra

- Mantiene un conjunto S con los vértices para los cuales el **costo** del ***camino de costo mínimo*** de ***s*** a ellos ya ha sido calculado
- En cada iteración, selecciona un vértice ***u*** tal que $u \in V\text{-}S$ y además cumple que: $\forall v \in V\text{-}S$, $\mathbf{d}[u] \leq \mathbf{d}[v]$
- Adiciona ***u*** a S
- Hace ***RELAX*** sobre todos los arcos **(*u, v*)** *tq.* $v \in V\text{-}S$ verificando si **d[*v*]** mejora*,* llegando a ***v*** desde ***u***

Algoritmo de Dijkstra

Q: ***cola con prioridad*** de vértices, ordenados parcialmente según el valor ***d*** (los menores valores de ***d*** son los más prioritarios)

```
DIJKSTRA(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)
 S ← Ø
 Q ← V[G]  // BUILD-HEAP → en O(V)
  while Q ≠ Ø
        do    u ← EXTRACT-MIN(Q)
              S ← S ∪ {u}
              for each vertex v ∈ Adj[u]
                    do RELAX(u, v, w)
```

NOTE Similitud con el Algoritmo de PRIM

Análisis del algoritmo

línea 1 - Inicializaciones - d y π

línea 2 - Inicialización de $S = \phi$

• El algoritmo al comienzo de cada iteración del ciclo ***while*** de las **líneas 4-8** **mantiene la siguiente invariante**: $\boxed{Q=V-S}$

línea 3 - Inicialización de Q = $|V|$

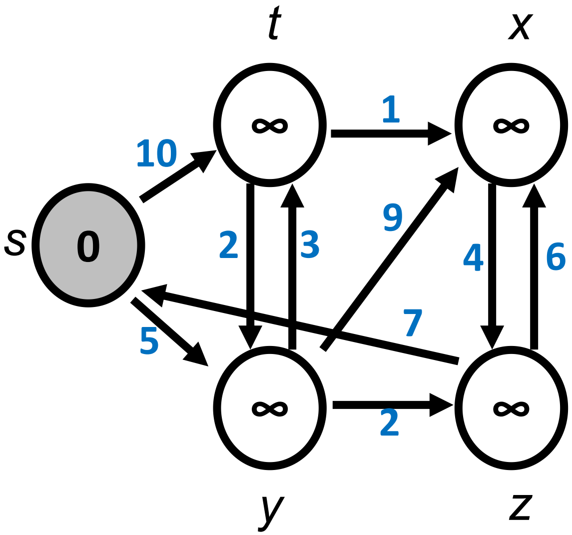
• Tras la inicialización se cumple la invariante Q = V-S = V-$\emptyset$ = V

• En cada iteración del ciclo ***while***, un vértice *u* se extrae de Q =V-S y se inserta en el conjunto *S*, por tal motivo, se garantiza que se mantenga también la invariante

• La primera vez que se itera sobre el ciclo ***while***, *u=s*.

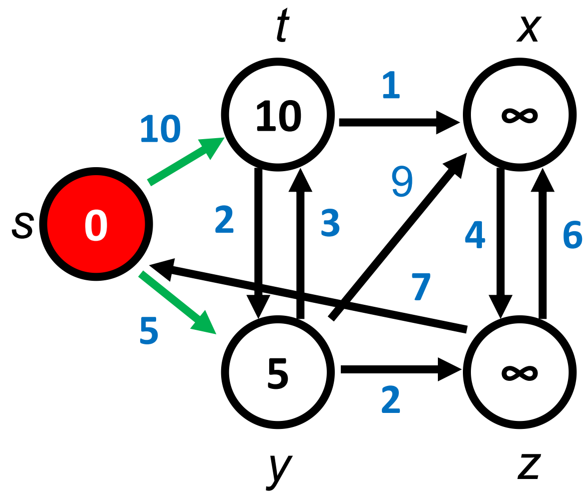
líneas 7-8 - Se aplica ***RELAX*** a cada arco *(u, v)* que parte desde *u* (arcos entre *u* y sus adyacentes). Con ello se actualizan los valores de *d[v]* y de *π[v]*

Ejecutando el algoritmo de Dijkstra



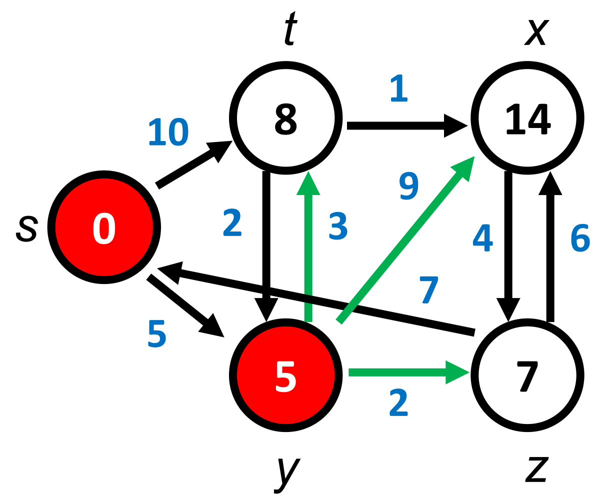
S=ϕ
V-S{s, t, y, x, z}

d[s]=0	Π[s]=null
d[t]=∞	Π[t]=null
d[y]=∞	Π[y]=null
d[x]=∞	Π[x]=null
d[z]=∞	Π[z]=null



S={s}
V-S{t, y, x, z}

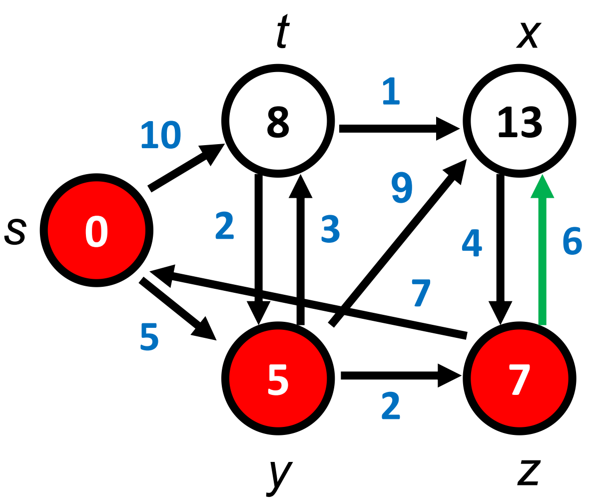
d[s]=0	**Π[s]=null**
d[t]=10	Π[t]=s
d[y]=5	Π[y]=s
d[x]=∞	Π[x]=null
d[z]=∞	Π[z]=null



S={s, y}
V-S{t, x, z}

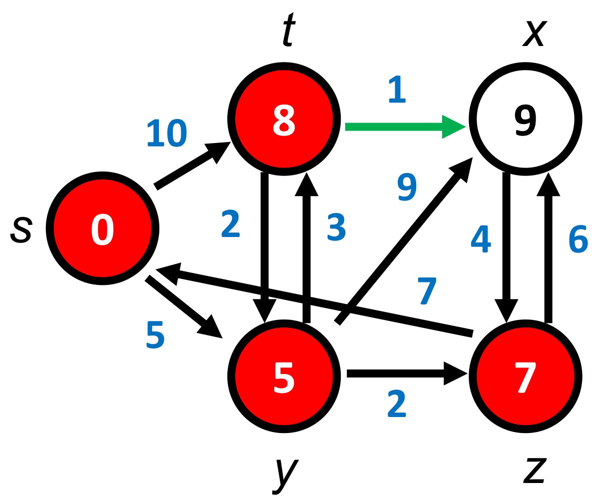
d[s]=0	**Π[s]=null**
d[t]=8	Π[t]=y
d[y]=5	**Π[y]=s**
d[x]=14	Π[x]=y
d[z]=7	Π[z]=y

Ejecutando el algoritmo de Dijkstra



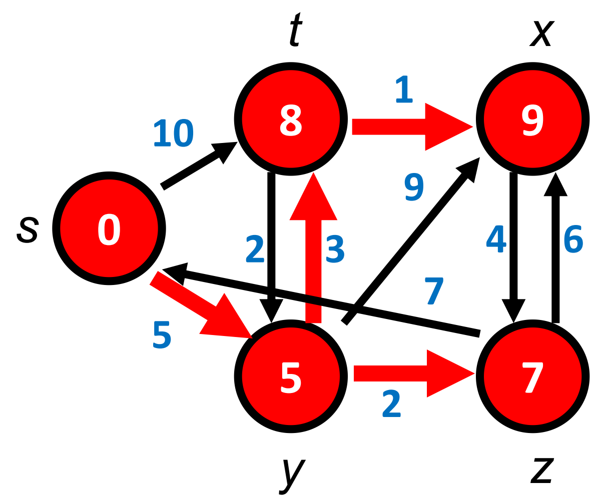
S={s, y, z}
V-S{t, x}

d[s]=0 Π[s]=null
d[t]=8 Π[t]=y
d[y]=5 Π[y]=s
d[x]=13 Π[x]=z
d[z]=7 Π[z]=y



S={s, y, z, t}
V-S{x}

d[s]=0 Π[s]=null
d[t]=8 Π[t]=y
d[y]=5 Π[y]=s
d[x]=9 Π[x]=t
d[z]=7 Π[z]=y



S={s, y, z, t, x}
V-S=ϕ

d[s]=0 Π[s]=null
d[t]=8 Π[t]=y
d[y]=5 Π[y]=s
d[x]=9 Π[x]=t
d[z]=7 Π[z]=y

Correctitud del algoritmo de Dijkstra

Teorema 24.6:

Si el algoritmo de Dijkstra se ejecuta sobre un grafo dirigido y ponderado $G = (V, E)$ con una función de costos **no negativos** w definida sobre él y se tiene un vértice s como origen, entonces al concluir la ejecución del mismo se tiene $d[u] = \delta(s, u)$ para todo $u \in V$

```
DIJKSTRA(G, w, s)

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

Demostración

invariante de ciclo:
Al comenzar cada iteración del ciclo ***while*** comprendido entre las **líneas 4-8:**
$d[v] = \delta(s, v)$ para cada vértice $V \in S$

Correctitud del algoritmo de Dijkstra

Demostración (*continuación …*)

Basta con demostrar que:

$\forall u \in V;\ d[u] = \delta(s, u)$ cuando u se inserta en ***S***

Una vez demostrado esto, se usa la ***Propiedad de la Cota Superior***, para probar que la igualdad se mantiene después que se alcanza

- **Inicialización**

Inicialmente, $S = \emptyset \Rightarrow$ se cumple la invariante

Correctitud del algoritmo de Dijkstra

Demostración (*continuación* …)

- **Mantenimiento**

Se desea demostrar que en cada iteración, $\boldsymbol{d[u] = \delta(s, u)}$ para el vértice ***u*** que se inserta en ***S***

Supongamos que esto no es cierto:

Sea ***u* el primer vértice** para el cual $\boldsymbol{d[u] \neq \delta(s, u)}$ **en el momento en que dicho vértice se decide insertar en el conjunto *S***

Correctitud del algoritmo de Dijkstra

Centraremos la atención en la situación existente en el **momento en que comienza la iteración del ciclo *while* en el cual se decide que *u* es el vértice que va a entrar a *S***:

- $u \neq s$

  s fue el primer vértice que se insertó en S y la propia inicialización garantiza que $d[s] = \delta(s, s) = 0$

- $S \neq \emptyset$

  exactamente antes de que *u* se haya insertado en *S*

- **EXISTE, al menos, un camino de *s* a *u***

  De lo contrario, $d[u] = \delta(s, u) = \infty$ por la **Propiedad de la no existencia de caminos**. Esto niega lo asumido: $d[u] \neq \delta(s, u)$

Correctitud del algoritmo de Dijkstra

Si existe, al menos, UN camino de ***s*** a ***u*** $\Rightarrow$ existe, al menos, UN **camino de costo mínimo** de ***s*** a ***u***. Sea *p* dicho camino

Situación PREVIA a que *u* sea insertado en S:
p podría descomponerse de la siguiente forma:

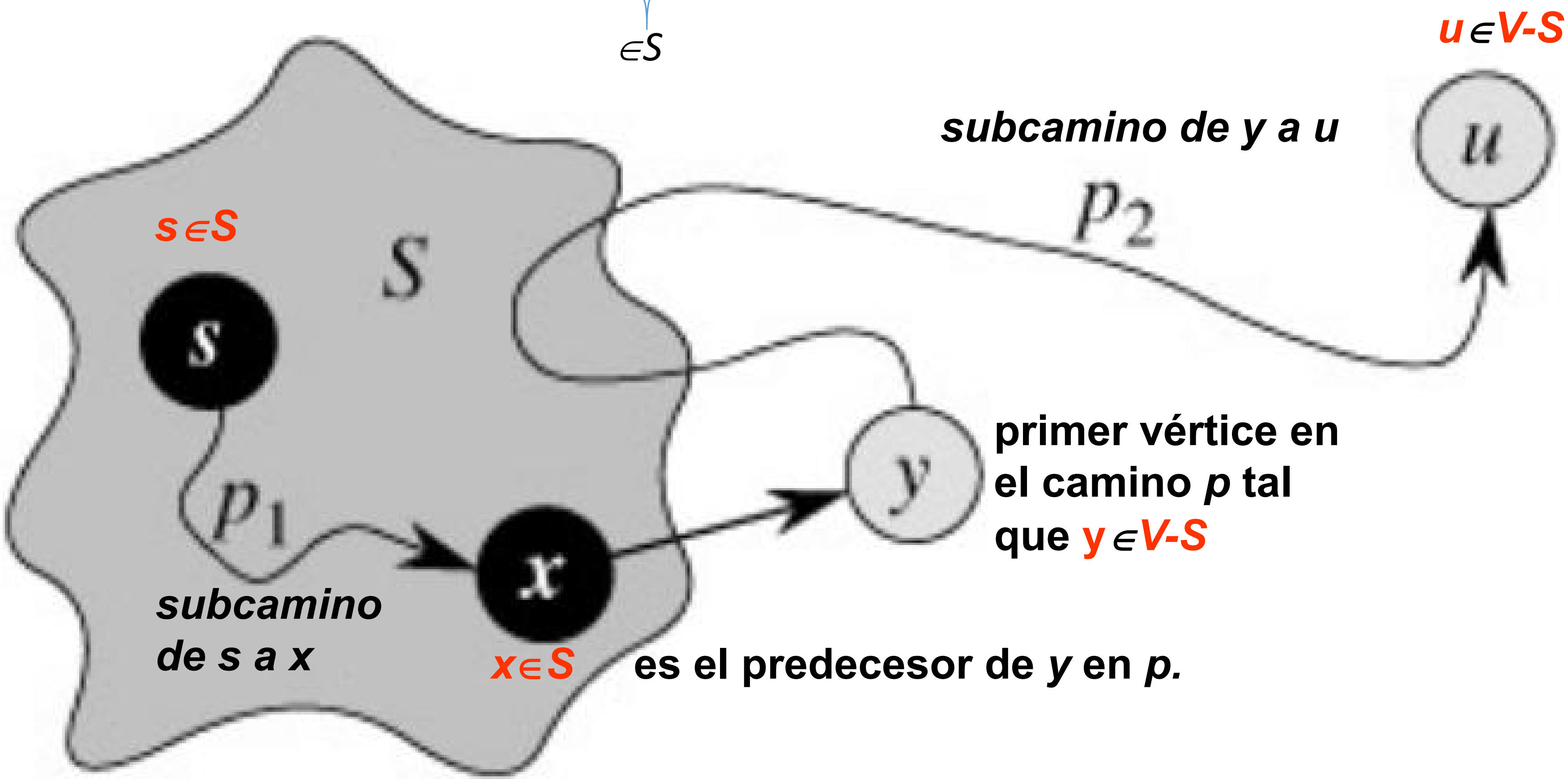
$$p = \underbrace{s \in S \xrightarrow{p_1} x}_{\in S} \rightarrow y \xrightarrow{p_2} u \in V - S$$

Donde:
p1: subcamino de ***s*** a ***x*** (**todos sus vértices en *S***)
p2: subcamino de ***y*** a ***u*** (**todos sus vértices en *V-S* *ó en* *S***)
y*:* primer vértice en ***p*** *t.q.* ***y*** $\in$ ***V-S***
x*:* último vértice en ***p*** *t.q.* ***x*** $\in$ ***S*** (predecesor de ***y*** en el camino ***p***)

Tanto *p1* como *p2* pueden ser de longitud 0

Correctitud del algoritmo de Dijkstra

$$p = \underbrace{s \in S \xrightarrow{p_1} x}_{\in S} \rightarrow y \xrightarrow{p_2} u \in V - S$$



Observaciones:

- ***x*** ≠ ***y*** pero podría suceder ***s*** = ***x*** o ***y*** = ***u***

- el subcamino ***p2*** podría o no, reentrar nuevamente en ***S***

Correctitud del algoritmo de Dijkstra

Situación CUANDO SE DECIDE insertar *u* en S:

- Se tiene que cumplir $d[y] = \delta(s, y)$

Justificación:

- $\boldsymbol{x \in S}$, por tanto, como asumimos que ***u*** es el primer vértice para el cual $\boldsymbol{d[u] \neq \delta(s, u)}$, cuando dicho vértice se adiciona a ***S***, se tenía que $\boldsymbol{d[x] = \delta(s, x)}$.

- Cuando ***x*** se insertó en ***S***, al arco ***(x, y)*** se le hizo ***RELAX*** y en ese momento se hizo $\boldsymbol{d[y] = \delta(s, y)}$ por la ***propiedad de la convergencia***

> Si $s \sim u \rightarrow v$ es un camino de costo mínimo en G con $u, v \in V$. Si se hace $d[u] = \delta(s, u)$ en cualquier momento antes de hacer *RELAX* sobre el arco (u, v), entonces, tras él, se hace $d[v] = \delta(s, v)$ y dicha igualdad se mantiene en lo sucesivo

A partir de lo anterior, lleguemos a una contradicción para probar **$d[u] = \delta(s, u)$**:

Como ***y*** está antes que ***u*** en ***p*****:camino de costo mínimo de *s* a *u***, donde todos los arcos tienen costos no negativos, entonces,

$\delta(s, y) \leq \delta(s, u)$

$$d[y] = \delta(s, y)$$
$$\leq \delta(s, u) \qquad (*)$$
$$\leq d[u] \quad \text{por la propiedad de la Cota Superior}$$

Como ***u*** y ***y*** estaban en *V-S* en el momento en que ***u*** fue seleccionado en la línea 5, entonces, **$d[u] \leq d[y]$** y teniendo en cuenta (*) tenemos

$$d[u] \leq d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u]$$

Por tanto, las dos desigualdades expresadas en (*) son realmente IGUALDADES, es decir: $d[u] = d[y] = \delta(s, y) = \delta(s, u)$ → **contradicción** pues se supuso $d[u] \neq \delta(s, u)$ $\therefore$ $d[u] = \delta(s, u)$ cuando *u* es insertado en *S* y la igualdad se mantiene hasta que el algoritmo concluye

Correctitud del algoritmo de Dijkstra

- **Terminación**

Al terminar el algoritmo, $Q = \emptyset$ lo cual, a partir de nuestra invariante de que siempre $Q=V - S$, implica $S = V$, por tanto,

$$\boldsymbol{d[u] = \delta(s, u) \ \forall u \in V}$$

Complejidad temporal del algoritmo de Dijkstra

```
DIJKSTRA(G, w, s)

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

O(V. T(**INSERT** en Q)) Cada vértice se inserta una sola vez

O(V. T(**EXTRACT-MIN**))
Cada vértice se elimina una sola vez

O(E. T(**DECREASE-KEY**))
implícito en el ***RELAX***

- Cada arco en ***Adj[u]*** se analiza una sola vez
- total de arcos de **G = |E|** $\Rightarrow$ **for** - **líneas 7-8**; itera **|E|** veces

Por tanto, habrán a lo sumo **|E|** llamados a **DECREASE-KEY** **(análisis amortizado)**

Complejidad temporal del algoritmo de Dijkstra

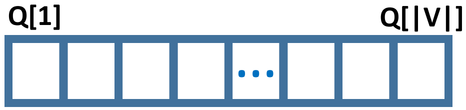
El tiempo de ejecución del algoritmo de Dijkstra depende de cómo se implemente la cola con prioridad Q

⇓

T(Dijkstra) = O(V.T(INSERT) + V.T(EXTRACT-MIN) + E.T(DECREASE-KEY))

Complejidad temporal del algoritmo de Dijkstra

CASO 1: *Q* no se trata propiamente como una *cola con prioridad* sino como un *array* tradicional : Q[1 … |V|] tal que; si v∈V = i entonces Q[i] = d[v]



En este caso **EXTRACTMIN** ⇔ **BUSCAR-MIN** en un *array*

- **INSERT** y **DECREASE-KEY** son O(1) ⇒ **O(V.T(INSERT) = $\boxed{\text{O(V)}}$**

  O(E. T(DECREASE-KEY)) = $\boxed{\text{O(E)}}$

- **BUSCAR-MIN** es **O(V)** (necesidad de buscar por todo el arreglo)

  ⇒ **O(V.T(EXTRACTMIN) = O(V.V)= $\boxed{O(|V|^2)}$**

$$\boxed{T(\text{Dijkstra}) = O(V) + O(V^2) + O(E) = O(V^2)}$$

Esta es la versión de Dijkstra más tradicional y se utiliza cuando el grafo es denso (E $\approx V^2$)

CASO 2: Implementar Q como una cola con prioridad mediante un HEAP binario donde HEAPIFY es O(log n) y BUILD-HEAP es O(V) *(ver conferencia 1er semestre)*

Si el **grafo es esparcido** es práctico implementar la cola con prioridad con un **heap binario**

- Cada operación de **EXTRACT-MIN** es $O(\log |V|)$

$$\Rightarrow O(V.T(\text{EXTRACTMIN})) = \boxed{O(V \log V)}$$

- **BUILD-HEAP** es $O(|V|)$

$$\Rightarrow O(\ V.T(\text{INSERT}) \Leftrightarrow \text{BUILD-HEAP}\ ) = \boxed{O(V)}$$

- Cada operación **DECREASE-KEY** es $O(\log |V|)$

$$\Rightarrow O(E.T(\text{DECREASE-KEY})) = \boxed{O(E \log V)}$$

$$T(\text{Dijkstra}) = O(V) + O(V \log V) + O(E \log V) = O(V \log V) + O(E \log V)$$
$$= O\big((V + E) \log V\big) = O(E \log V)$$

La igualdad se cumple si todos los vértices son alcanzables desde s, por tanto, el grafo predecesor es un árbol donde están todos los vértices y en ese árbol E=V—1 ⇒en G, E ≥ V-1, o sea, E domina V

NOTA: Si el grafo fuese denso ($|\mathbf{E}| \approx |V^2|$) entonces el algoritmo es $O(|V|^2 \log |V|)$