

# 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**



A map of La Habana, Cuba, showing a network of streets. A red line traces a path from a home icon (a black silhouette of a person sitting) in the upper left towards the University of Havana (Universidad) in the lower left. A green line traces a path from the home icon towards the University of Havana. A black dot is located on the Paseo de Martí (Prado) street, with a blue arrow pointing from a text box to it. The map includes various street names such as Calle K, Calle L, Calle M, Calle N, Calle O, Calle P, Calle Q, Calle R, Calle S, Calle T, Calle U, Calle V, Calle W, Calle X, Calle Y, Calle Z, Calle AA, Calle AB, Calle AC, Calle AD, Calle AE, Calle AF, Calle AG, Calle AH, Calle AI, Calle AJ, Calle AK, Calle AL, Calle AM, Calle AN, Calle AO, Calle AP, Calle AQ, Calle AR, Calle AS, Calle AT, Calle AU, Calle AV, Calle AW, Calle AX, Calle AY, Calle AZ, Calle BA, Calle BB, Calle BC, Calle BD, Calle BE, Calle BF, Calle BG, Calle BH, Calle BI, Calle BJ, Calle BK, Calle BL, Calle BM, Calle BN, Calle BO, Calle BP, Calle BQ, Calle BR, Calle BS, Calle BT, Calle BU, Calle BV, Calle BW, Calle BX, Calle BY, Calle BZ, Calle CA, Calle CB, Calle CC, Calle CD, Calle CE, Calle CF, Calle CG, Calle CH, Calle CI, Calle CJ, Calle CK, Calle CL, Calle CM, Calle CN, Calle CO, Calle CP, Calle CQ, Calle CR, Calle CS, Calle CT, Calle CU, Calle CV, Calle CW, Calle CX, Calle CY, Calle CZ, Calle DA, Calle DB, Calle DC, Calle DD, Calle DE, Calle DF, Calle DG, Calle DH, Calle DI, Calle DJ, Calle DK, Calle DL, Calle DM, Calle DN, Calle DO, Calle DP, Calle DQ, Calle DR, Calle DS, Calle DT, Calle DU, Calle DV, Calle DW, Calle DX, Calle DY, Calle DZ, Calle EA, Calle EB, Calle EC, Calle ED, Calle EE, Calle EF, Calle EG, Calle EH, Calle EI, Calle EJ, Calle EK, Calle EL, Calle EM, Calle EN, Calle EO, Calle EP, Calle EQ, Calle ER, Calle ES, Calle ET, Calle EU, Calle EV, Calle EW, Calle EX, Calle EY, Calle EZ, Calle FA, Calle FB, Calle FC, Calle FD, Calle FE, Calle FF, Calle FG, Calle FH, Calle FI, Calle FJ, Calle FK, Calle FL, Calle FM, Calle FN, Calle FO, Calle FP, Calle FQ, Calle FR, Calle FS, Calle FT, Calle FU, Calle FV, Calle FW, Calle FX, Calle FY, Calle FZ, Calle GA, Calle GB, Calle GC, Calle GD, Calle GE, Calle GF, Calle GG, Calle GH, Calle GI, Calle GJ, Calle GK, Calle GL, Calle GM, Calle GN, Calle GO, Calle GP, Calle GQ, Calle GR, Calle GS, Calle GT, Calle GU, Calle GV, Calle GW, Calle GX, Calle GY, Calle GZ, Calle HA, Calle HB, Calle HC, Calle HD, Calle HE, Calle HF, Calle HG, Calle HH, Calle HI, Calle HJ, Calle HK, Calle HL, Calle HM, Calle HN, Calle HO, Calle HP, Calle HQ, Calle HR, Calle HS, Calle HT, Calle HU, Calle HV, Calle HW, Calle HX, Calle HY, Calle HZ, Calle IA, Calle IB, Calle IC, Calle ID, Calle IE, Calle IF, Calle IG, Calle IH, Calle II, Calle IJ, Calle IK, Calle IL, Calle IM, Calle IN, Calle IO, Calle IP, Calle IQ, Calle IR, Calle IS, Calle IT, Calle IU, Calle IV, Calle IW, Calle IX, Calle IY, Calle IZ, Calle JA, Calle JB, Calle JC, Calle JD, Calle JE, Calle JF, Calle JG, Calle JH, Calle JI, Calle JJ, Calle JK, Calle JL, Calle JM, Calle JN, Calle JO, Calle JP, Calle JQ, Calle JR, Calle JS, Calle JT, Calle JU, Calle JV, Calle JW, Calle JX, Calle JY, Calle JZ, Calle KA, Calle KB, Calle KC, Calle KD, Calle KE, Calle KF, Calle KG, Calle KH, Calle KI, Calle KJ, Calle KK, Calle KL, Calle KM, Calle KN, Calle KO, Calle KP, Calle KQ, Calle KR, Calle KS, Calle KT, Calle KU, Calle KV, Calle KW, Calle KX, Calle KY, Calle KZ, Calle LA, Calle LB, Calle LC, Calle LD, Calle LE, Calle LF, Calle LG, Calle LH, Calle LI, Calle LJ, Calle LK, Calle LL, Calle LM, Calle LN, Calle LO, Calle LP, Calle LQ, Calle LR, Calle LS, Calle LT, Calle LU, Calle LV, Calle LW, Calle LX, Calle LY, Calle LZ, Calle MA, Calle MB, Calle MC, Calle MD, Calle ME, Calle MF, Calle MG, Calle MH, Calle MI, Calle MJ, Calle MK, Calle ML, Calle MM, Calle MN, Calle MO, Calle MP, Calle MQ, Calle MR, Calle MS, Calle MT, Calle MU, Calle MV, Calle MW, Calle MX, Calle MY, Calle MZ, Calle NA, Calle NB, Calle NC, Calle ND, Calle NE, Calle NF, Calle NG, Calle NH, Calle NI, Calle NJ, Calle NK, Calle NL, Calle NM, Calle NN, Calle NO, Calle NP, Calle NQ, Calle NR, Calle NS, Calle NT, Calle NU, Calle NV, Calle NW, Calle NX, Calle NY, Calle NZ, Calle OA, Calle OB, Calle OC, Calle OD, Calle OE, Calle OF, Calle OG, Calle OH, Calle OI, Calle OJ, Calle OK, Calle OL, Calle OM, Calle ON, Calle OO, Calle OP, Calle OQ, Calle OR, Calle OS, Calle OT, Calle OU, Calle OV, Calle OW, Calle OX, Calle OY, Calle OZ, Calle PA, Calle PB, Calle PC, Calle PD, Calle PE, Calle PF, Calle PG, Calle PH, Calle PI, Calle PJ, Calle PK, Calle PL, Calle PM, Calle PN, Calle PO, Calle PP, Calle PQ, Calle PR, Calle PS, Calle PT, Calle PU, Calle PV, Calle PW, Calle PX, Calle PY, Calle PZ, Calle QA, Calle QB, Calle QC, Calle QD, Calle QE, Calle QF, Calle QG, Calle QH, Calle QI, Calle QJ, Calle QK, Calle QL, Calle QM, Calle QN, Calle QO, Calle QP, Calle QQ, Calle QR, Calle QS, Calle QT, Calle QU, Calle QV, Calle QW, Calle QX, Calle QY, Calle QZ, Calle RA, Calle RB, Calle RC, Calle RD, Calle RE, Calle RF, Calle RG, Calle RH, Calle RI, Calle RJ, Calle RK, Calle RL, Calle RM, Calle RN, Calle RO, Calle RP, Calle RQ, Calle RR, Calle RS, Calle RT, Calle RU, Calle RV, Calle RW, Calle RX, Calle RY, Calle RZ, Calle SA, Calle SB, Calle SC, Calle SD, Calle SE, Calle SF, Calle SG, Calle SH, Calle SI, Calle SJ, Calle SK, Calle SL, Calle SM, Calle SN, Calle SO, Calle SP, Calle SQ, Calle SR, Calle SS, Calle ST, Calle SU, Calle SV, Calle SW, Calle SX, Calle SY, Calle SZ, Calle TA, Calle TB, Calle TC, Calle TD, Calle TE, Calle TF, Calle TG, Calle TH, Calle TI, Calle TJ, Calle TK, Calle TL, Calle TM, Calle TN, Calle TO, Calle TP, Calle TQ, Calle TR, Calle TS, Calle TT, Calle TU, Calle TV, Calle TW, Calle TX, Calle TY, Calle TZ, Calle UA, Calle UB, Calle UC, Calle UD, Calle UE, Calle UF, Calle UG, Calle UH, Calle UI, Calle UJ, Calle UK, Calle UL, Calle UM, Calle UN, Calle UO, Calle UP, Calle UQ, Calle UR, Calle US, Calle UT, Calle UY, Calle UZ, Calle VA, Calle VB, Calle VC, Calle VD, Calle VE, Calle VF, Calle VG, Calle VH, Calle VI, Calle VJ, Calle VK, Calle VL, Calle VM, Calle VN, Calle VO, Calle VP, Calle VQ, Calle VR, Calle VS, Calle VT, Calle VU, Calle VV, Calle VW, Calle VX, Calle VY, Calle VZ, Calle WA, Calle WB, Calle WC, Calle WD, Calle WE, Calle WF, Calle WG, Calle WH, Calle WI, Calle WJ, Calle WK, Calle WL, Calle WM, Calle WN, Calle WO, Calle WP, Calle WQ, Calle WR, Calle WS, Calle WT, Calle WY, Calle WZ, Calle XA, Calle XB, Calle XC, Calle XD, Calle XE, Calle XF, Calle XG, Calle XH, Calle XI, Calle XJ, Calle XK, Calle XL, Calle XM, Calle XN, Calle XO, Calle XP, Calle XQ, Calle XR, Calle XS, Calle XT, Calle XU, Calle XV, Calle XW, Calle XX, Calle XY, Calle XZ, Calle YA, Calle YB, Calle YC, Calle YD, Calle YE, Calle YF, Calle YG, Calle YH, Calle YI, Calle YJ, Calle YK, Calle YL, Calle YM, Calle YN, Calle YO, Calle YP, Calle YQ, Calle YR, Calle YS, Calle YT, Calle YU, Calle YV, Calle YW, Calle YX, Calle YY, Calle YZ, Calle ZA, Calle ZB, Calle ZC, Calle ZD, Calle ZE, Calle ZF, Calle ZG, Calle ZH, Calle ZI, Calle ZJ, Calle ZK, Calle ZL, Calle ZM, Calle ZN, Calle ZO, Calle ZP, Calle ZQ, Calle ZR, Calle ZS, Calle ZT, Calle ZU, Calle ZV, Calle ZW, Calle ZX, Calle ZY, Calle ZZ.

## 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

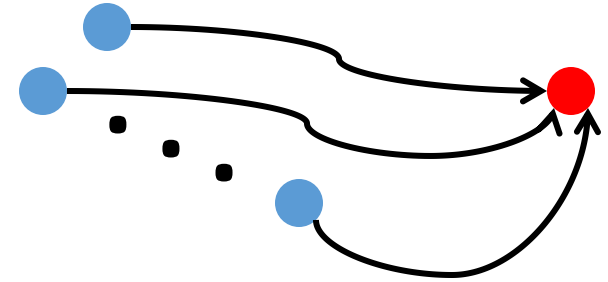
**“Caminos de costo mínimo a partir de solo un origen”**

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 :

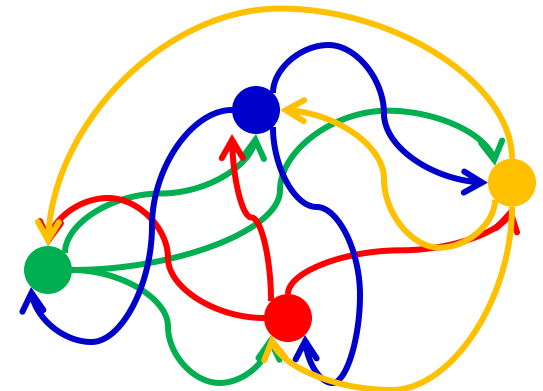
- **Caminos de costo mínimo hacia un único destino** (*Single-destination shortest-paths problem*): Encontrar el camino de costo mínimo desde cada vértice del grafo a un vértice de destino  $v$  de  $G$



- **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**  $w : E \rightarrow \mathbb{R}$

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

- $s \in V$  es el **vértice origen**

# camino de costo mínimo

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

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

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

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

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

**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*

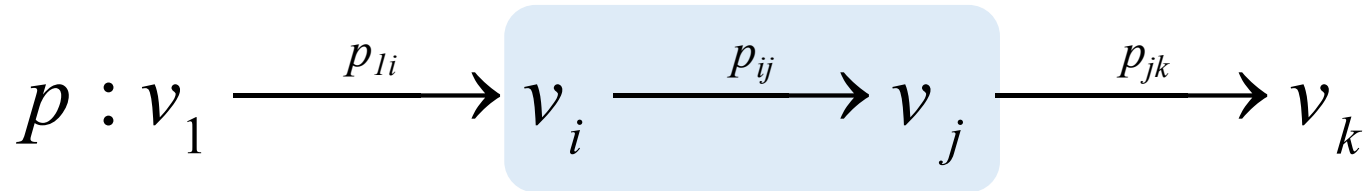
Sea  $G=(V,E)$ , dirigido, ponderado, con función de costo  $w: E \rightarrow R$ , definida sobre  $G$ . Sea  $p = \{v_1, v_2, \dots, v_k\}$  un **camino de costo mínimo** entre  $v_1$  y  $v_k$ ,  $\forall i, j: 1 \leq i \leq j \leq k$ , sea

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

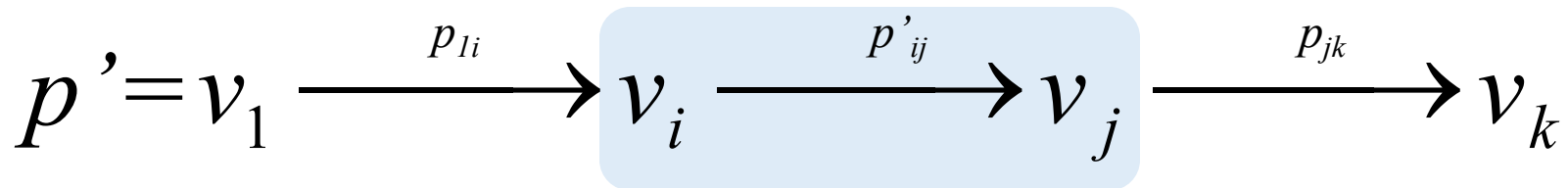


## Demostración – Lema 24. 1:

Descompongamos  $p$  de la forma



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



será un camino de  $v_1$  a  $v_k$  cuyo costo

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

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



# Arcos de costo negativo

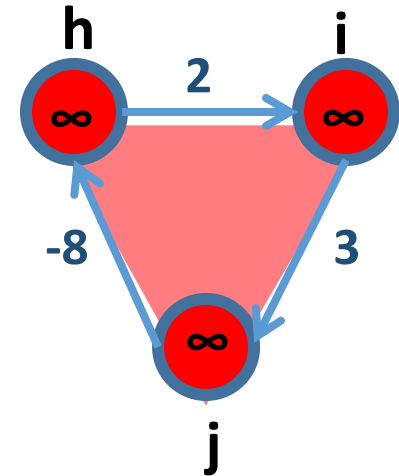
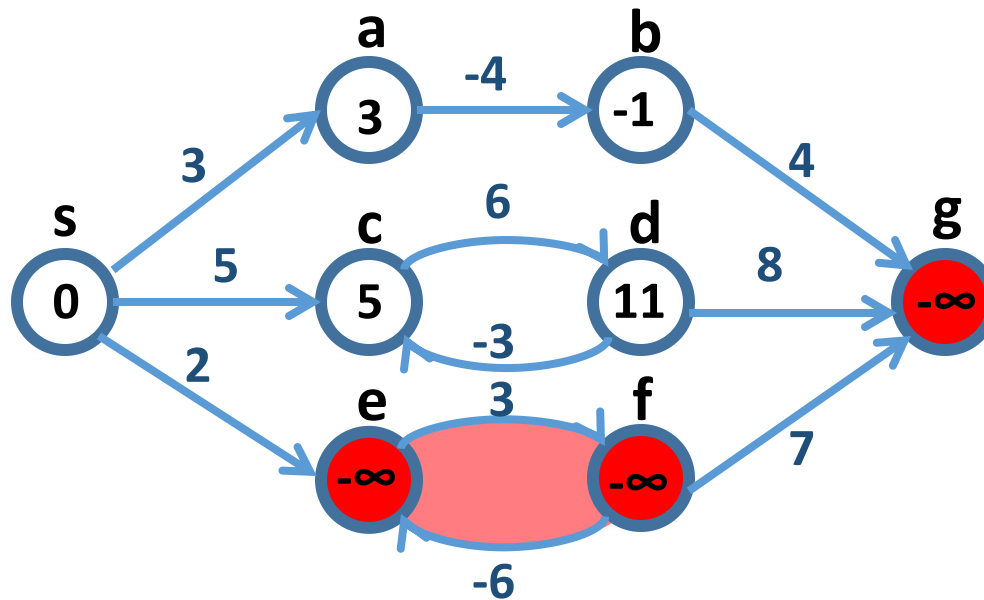
En este tipo de problemas, no se excluye la posibilidad de que en el grafo existan arcos con costo negativo

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  $-\infty$ .
- Como el vértice  $g$  se alcanza desde un vértice con costo  $-\infty$ , para el camino de costo mínimo del origen a él, entonces el costo del camino de costo mínimo de  $s$  a  $g$  es también  $-\infty$ .
- Los vértices  $h$ ,  $i$  e  $j$  no son alcanzables desde  $s$ , entonces el costo del camino de costo mínimo de  $s$  a ellos es  $\infty$  aún cuando entre ellos se forme un ciclo de costo negativo

# Algoritmos que dan solución al problema

- **Algoritmo de Dijkstra:**

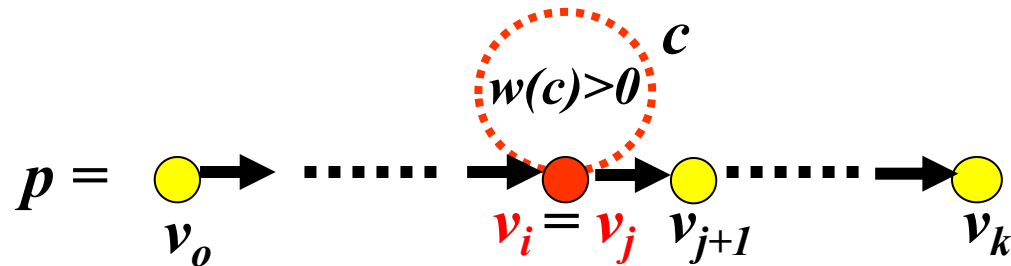
Asume que el **costo** de todos los arcos del grafo son **NO negativos**

- **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 ?



$$p = \langle v_0, v_1, \dots, v_i, \dots, v_j = v_i, v_{j+1}, \dots, v_k \rangle$$

**R/ NO**

$$p' = \langle v_0, v_1, \dots, v_j, v_{j+1}, \dots, v_k \rangle$$

$$w(p') = w(p) - w(c) < w(p), \text{ por tanto}$$

$p \rightarrow$  **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  $v \in V$  se establece su **predecesor  $\pi[v]$**  en algún **camino de costo mínimo**:

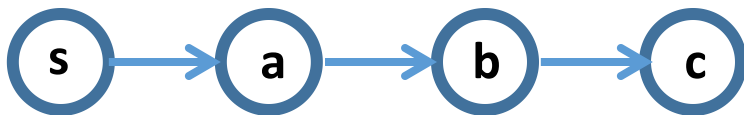
$$\pi[v]=r: r \in V \quad \text{ó} \quad \pi[v]= \text{null}$$

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

# Determinar secuencia de vértices en un camino

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

```
1  if  $v = s$  // caso base
2      then print  $s$ 
3  else if  $\pi[v] = \text{NULL}$ 
4      then print "there is no path from"  $s$  "to"  $v$ 
5  else PRINT-PATH( $G, s, \pi[v]$ )
6      print  $v$  // imprime el camino desde  $s$  hasta  $\pi[v]$ 
           // antes de imprimir al propio  $v$ 
```



$\pi[c] = b$   
 $\pi[b] = a$   
 $\pi[a] = s$   
 $\pi[s] = \text{null}$

**Secuencia de llamados**

PRINT-PATH( $G, s, c$ )

PRINT-PATH( $G, s, b$ )

PRINT-PATH( $G, s, a$ )

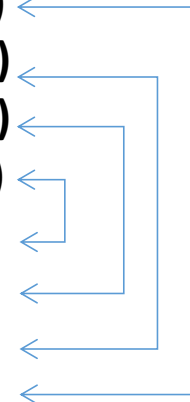
PRINT-PATH( $G, s, s$ )

$s$

$a$

$b$

$c$





# Árbol de los caminos de costo mínimo

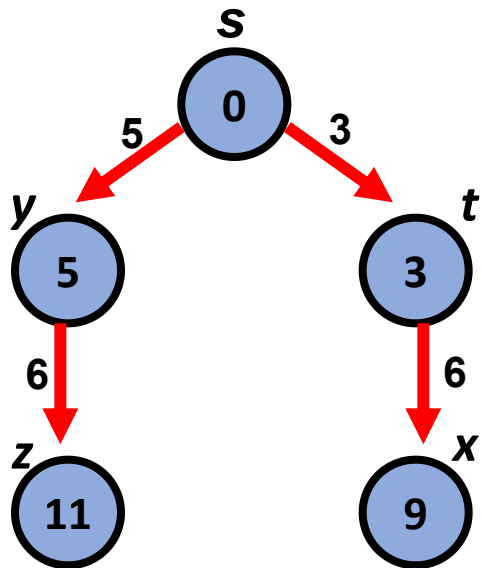
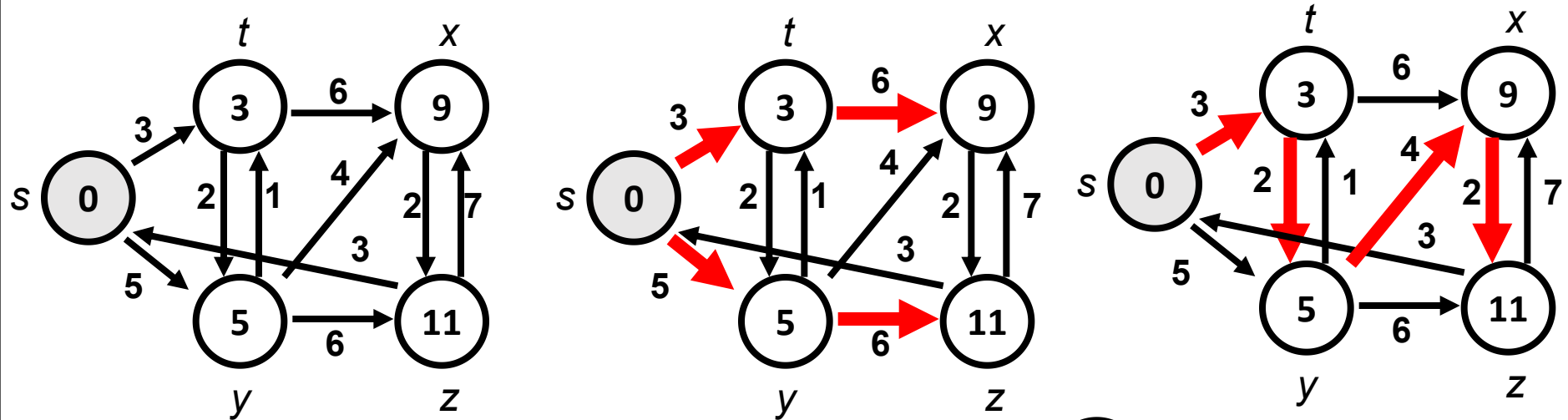
- Sea  $G = (V, E)$  un grafo dirigido y ponderado con función de costo  $w: E \rightarrow R$
- En  $G$  no existen **ciclos de costo negativo** alcanzables desde el origen
- Sea  $s \in V$  el origen

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:

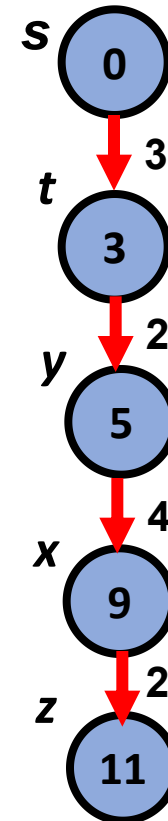
- $V'$  es el conjunto de vértices alcanzables desde  $s$  en  $G$
- $G'$  forma un árbol con raíz  $s$
- $\forall v \in V$ , el **único camino simple de  $s$  a  $v$  en  $G'$**  es UN **camino de costo mínimo** de  $s$  a  $v$  en  $G$

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  $\pi[v]$  se inicializan en  $O(V)$  por el siguiente método:

INITIALIZE-SINGLE-SOURCE( $G, s$ )

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

2     **do**  $d[v] \leftarrow \infty$

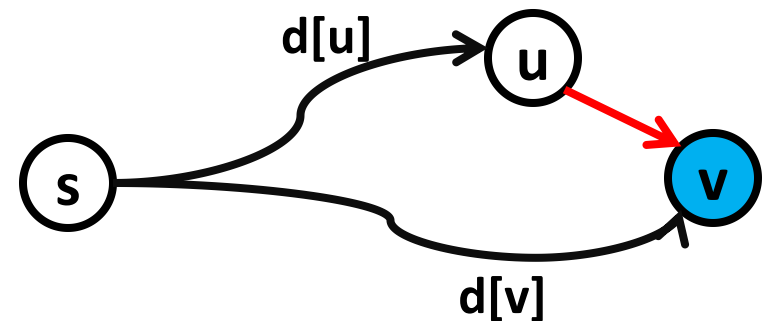
3      $\pi[v] \leftarrow \text{NULL}$

4      $d[s] \leftarrow 0$

# 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  $\pi[v]$



**RELAX**( $u, v, w$ )

- 1 **if**  $d[v] > d[u] + w(u, v)$
- 2     **then**  $d[v] \leftarrow d[u] + w(u, v)$
- 3          $\pi[v] \leftarrow u$

# 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  $\pi[v]$  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**

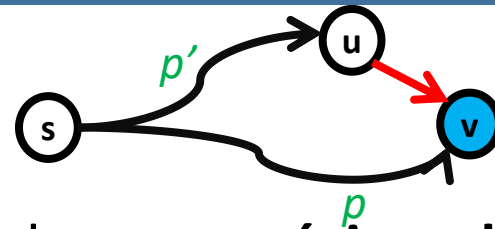
# Propiedades

**Consideraciones:** Sea  $G=(V, E)$  **dirigido** y **ponderado** con *función de costo*  $w: E \rightarrow R$  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  $\pi[v]$  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  $p' = \text{«un camino de costo mínimo de } s \text{ a } u\text{»} + (u, v)$

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

## Demostración

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

### caso base:

Tras las inicializaciones se cumplirá la invariante:  $d[v] \geq \delta(s, v)$  pues  $d[v] = \infty, \forall v \in V - \{s\} \Rightarrow d[v] \geq \delta(s, v)$

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

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



# 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  $d[v] = \infty \geq \delta(s, v)$  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,  $d[v] = d[u] + w(u, v)$
- $d[v] \leq d[u] + w(u, v) \Rightarrow$  tras hacerlo, ni  $d[u]$  ni  $d[v]$  cambian, por tanto, la desigualdad  $d[v] \leq d[u] + w(u, v)$  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  $d[u] = \delta(s, u)$  entonces la misma se mantiene en lo sucesivo (por **P. Cota Superior. Lema 24.11**)

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

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

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  $p = \langle v_0, v_1, \dots, v_k \rangle$  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:  $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$ , 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  $= 0, 1, \dots, |V|-1$ )

## 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:

$$\forall a \in \mathbb{R}, a + \infty = \infty + a = \infty \qquad a + (-\infty) = (-\infty) + a = -\infty$$

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

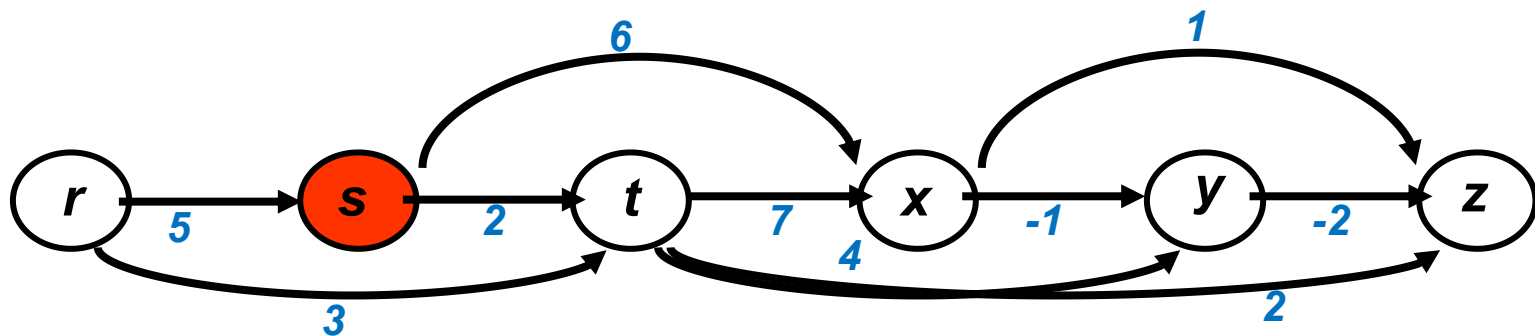
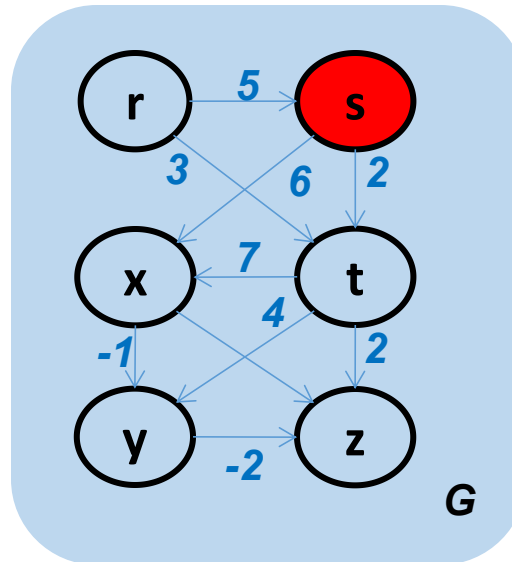
# Algoritmo DAG

DAG-SHORTEST-PATHS( $G, w, s$ )

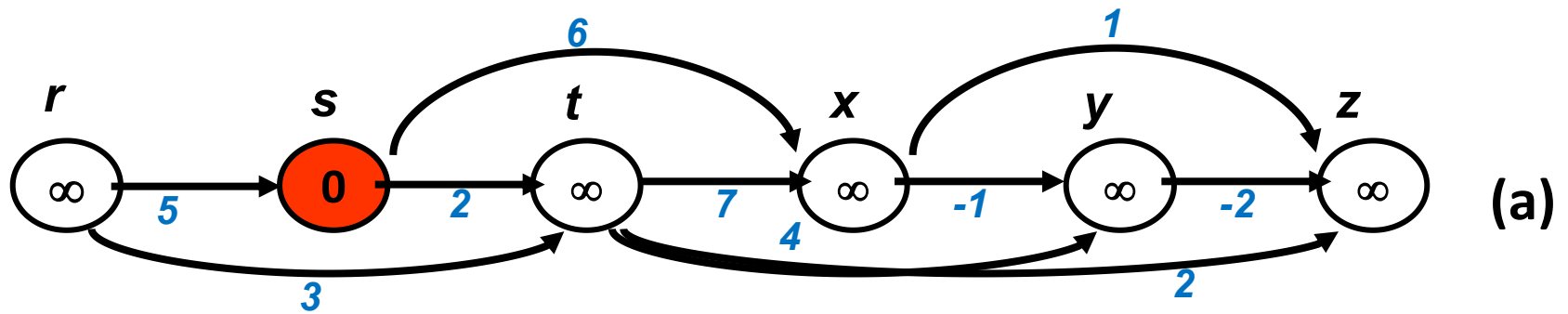
- 1 topologically sort the vertices of  $G$
- 2 INITIALIZE-SINGLE-SOURCE( $G, s$ )
- 3 **for** each vertex  $u$ , taken in topologically sorted order
- 4     **do for** each vertex  $v \in Adj[u]$
- 5         **do** RELAX( $u, v, w$ )

**RELAX**( $u, v, w$ )

- 1 **if**  $d[v] > d[u] + w(u, v)$
- 2     **then**  $d[v] \leftarrow d[u] + w(u, v)$
- 3          $\pi[v] \leftarrow u$

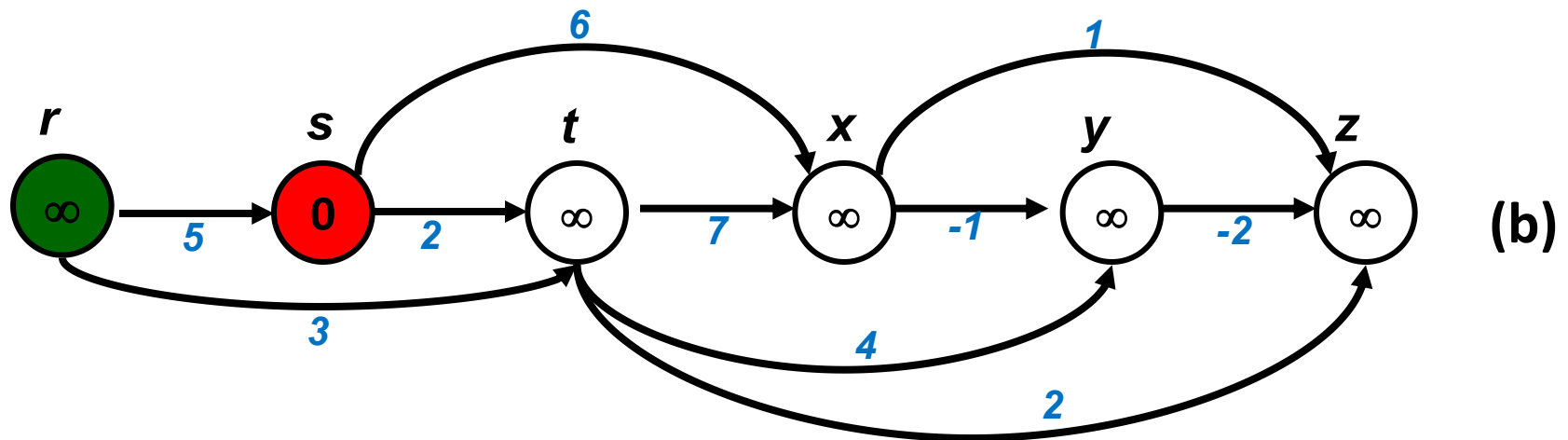


Los vértices de  $G$  están topológicamente ordenados de izquierda a derecha

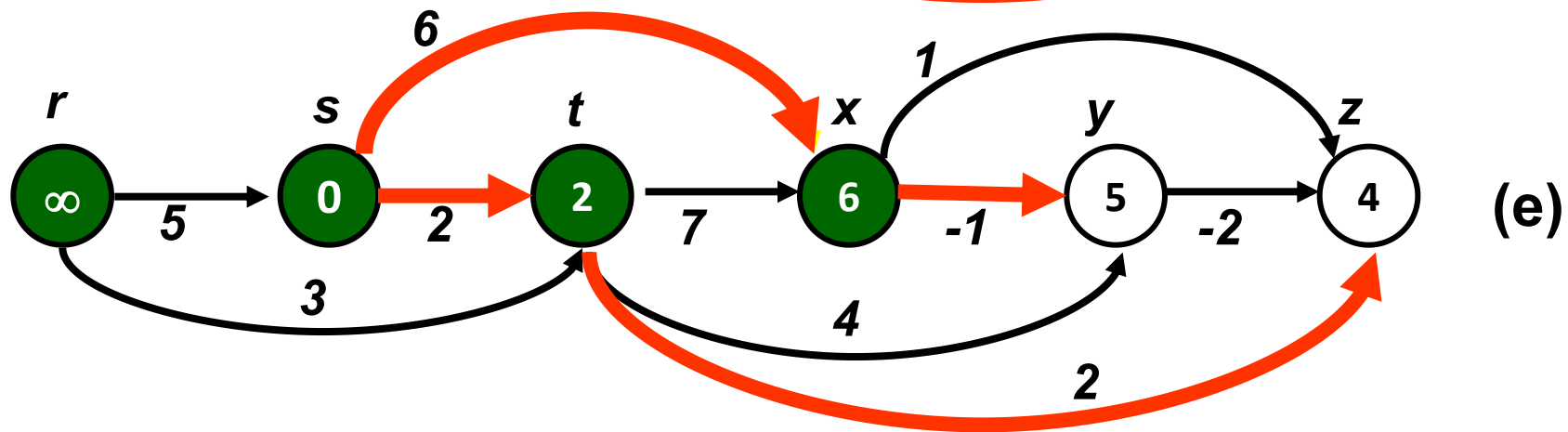
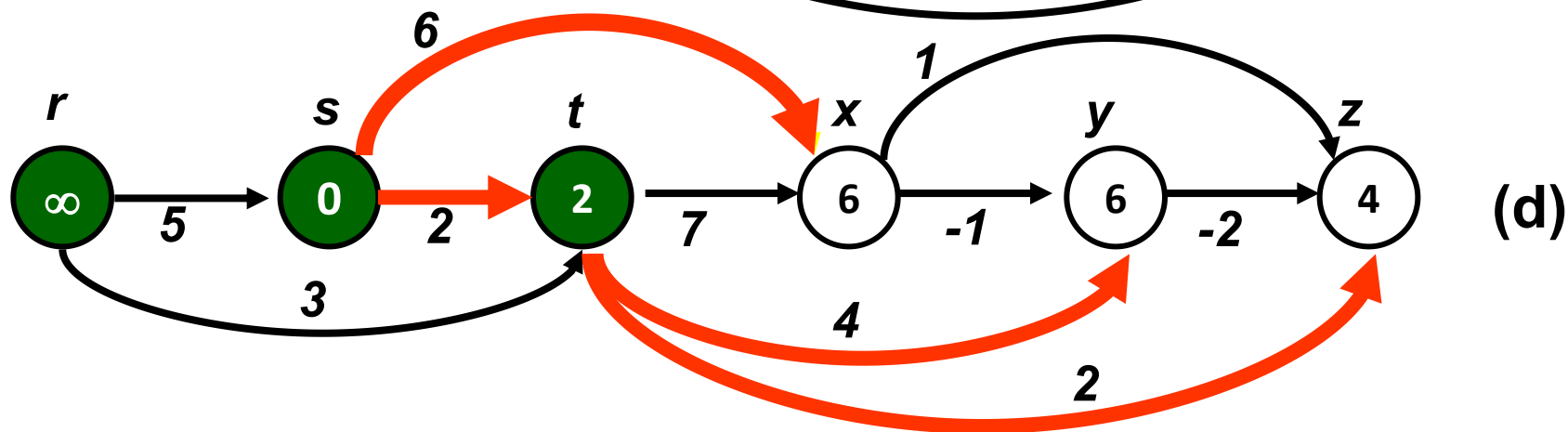
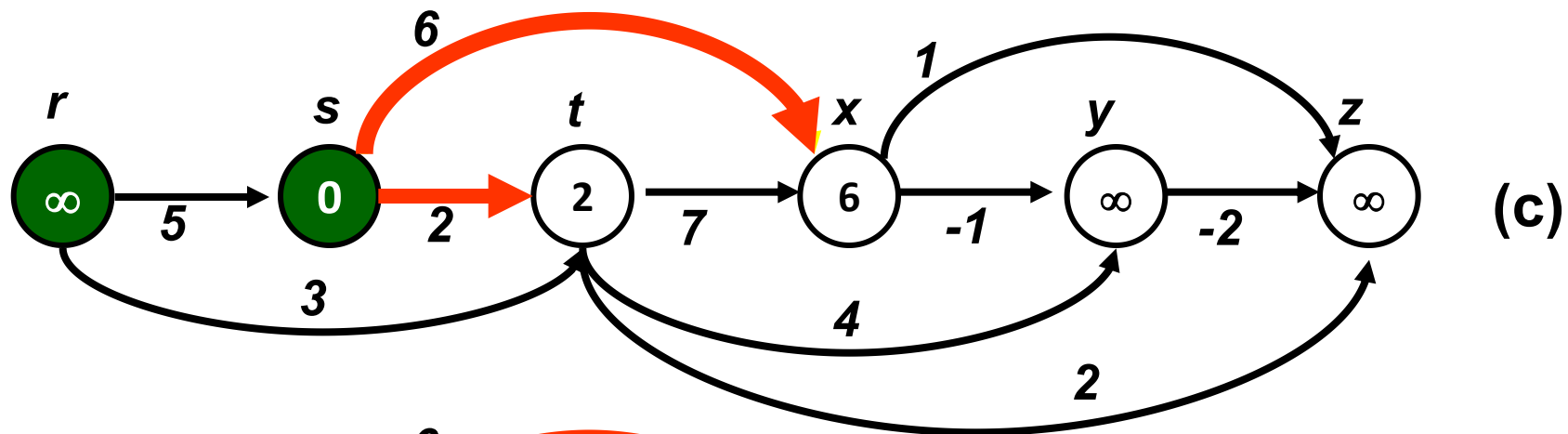


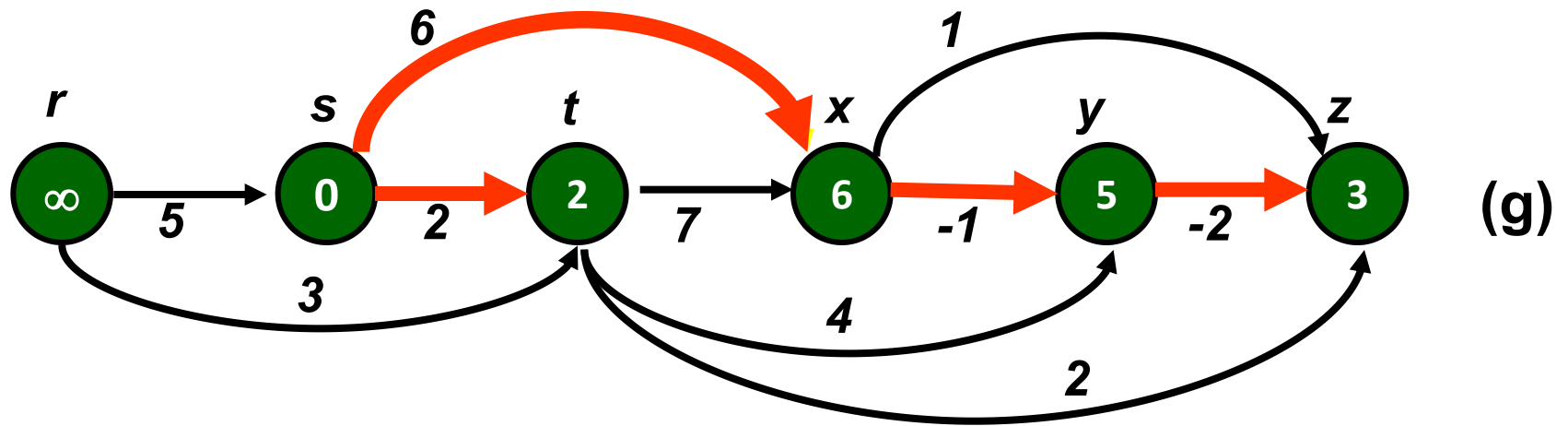
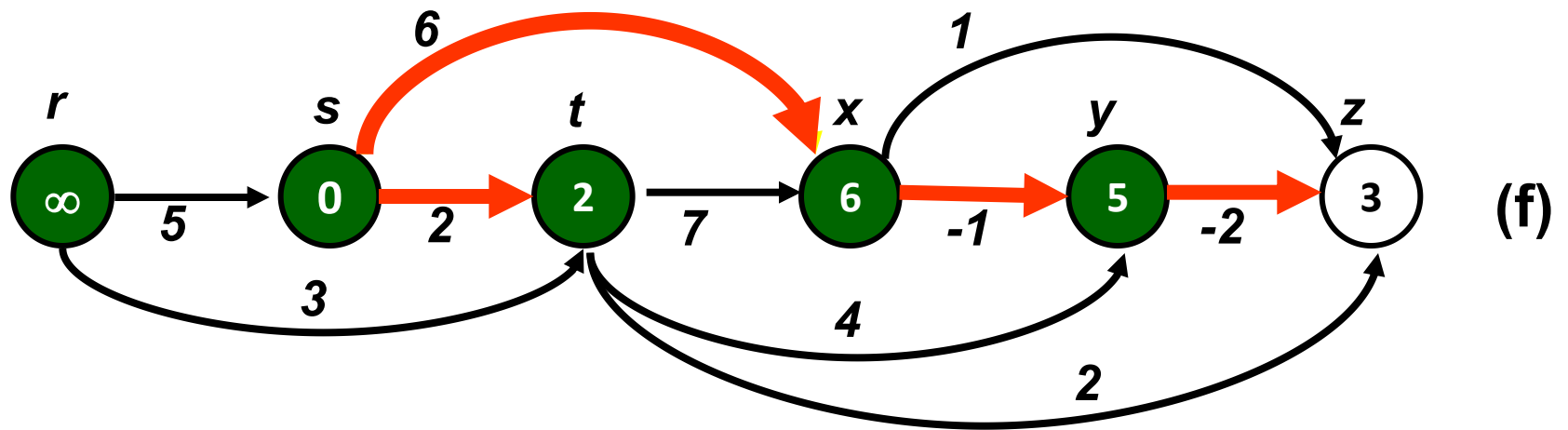
(a) La situación del grafo antes de la primera iteración del ciclo **for** comprendido entre las líneas 3-5.

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  $\pi$





(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  $G$   $\rightarrow O(|V| + |E|)$

- INITIALIZE-SINGLE-SOURCE( $G, s$ )  $\rightarrow O(|V|)$

- 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  $O(1)$

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

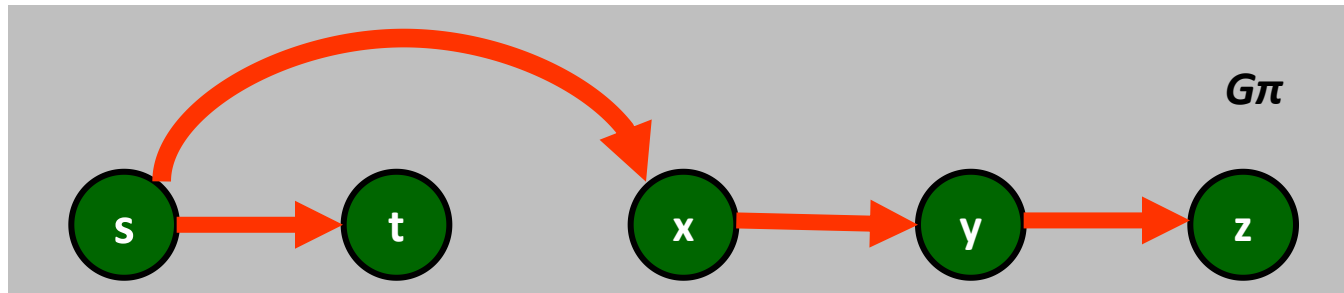
Por tanto, el tiempo **TOTAL** de ejecución del algoritmo es  $O(|V| + |E|)$  para una representación por Listas de Adyacencia del Grafo

# Subgrafo predecesor

$G\pi = (V\pi, E\pi)$  es el *subgrafo predecesor* generado por los valores de  $\pi$ , donde:

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

$G\pi$  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**,  $d[v] = \delta(s, v)$  para todo  $v \in V$  y el subgrafo predecesor  $G_\pi$  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 camino de costo mínimo  $p = \langle v_0, v_1, \dots, v_k \rangle$ , donde  $v_0 = s$  y  $v_k = v$

Como los  $v_i$  en  $p$  se les

### Lema 24.17

Una vez que se alcanza la igualdad:  $d[v] = \delta(s, v)$

$\forall v \in V$ , el subgrafo predecesor es un árbol de caminos de costo mínimo que tiene como raíz al vértice origen  $s$

Por la Prop. algoritmo, p

Por **Lema 24.17** (ver demostración en el l. to A.) -  $G_\pi$  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  $d[v] \geq \delta(s, v)$  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  $d[v] = \delta(s, v) = \infty$

- **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  $p = \langle v_0, v_1, \dots, v_k \rangle$  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:  $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$ , 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  $= 0, 1, \dots, |V|-1$ )

# 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  $w(u, v) \geq 0 \forall (u, v) \in E$

## 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  $u \in V - S$  y además cumple que:  $\forall v \in V - S, d[u] \leq d[v]$
- Adiciona **u** a **S**
- Hace **RELAX** sobre todos los arcos  $(u, v)$  tq.  $v \in V - S$  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( $G, w, s$ )

```
1  INITIALIZE-SINGLE-SOURCE( $G, s$ )
2   $S \leftarrow \emptyset$ 
3   $Q \leftarrow V[G]$  // BUILD-HEAP  $\rightarrow$  en  $O(V)$ 
4  while  $Q \neq \emptyset$ 
5      do  $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6           $S \leftarrow S \cup \{u\}$ 
7          for each vertex  $v \in \text{Adj}[u]$ 
8              do RELAX( $u, v, w$ )
```

NOTE Similitud con  
el Algoritmo de  
PRIM



# Análisis del algoritmo

**línea 1** - Inicializaciones -  $d$  y  $\pi$

**línea 2** - Inicialización de  $S = \emptyset$

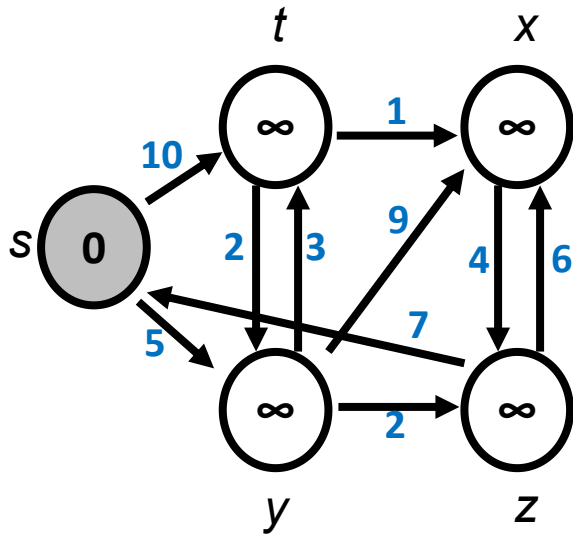
• El algoritmo al comienzo de cada iteración del ciclo **while** de las líneas 4-8 **mantiene la siguiente invariante:**  $Q=V-S$

**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**,  $u=s$ .

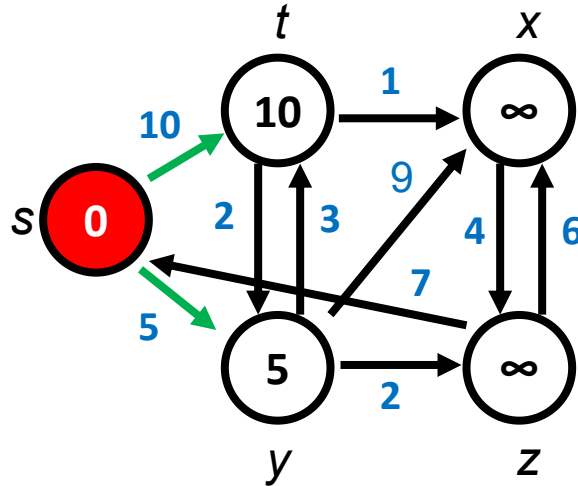
**líneas 7-8** - Se aplica **RELAX** a cada arco  $(u, v)$  que parte desde  $u$  (arcos entre  $u$  y sus adyacentes). Con ello se actualizan los valores de  $d[v]$  y de  $\pi[v]$

# Ejecutando el algoritmo de Dijkstra



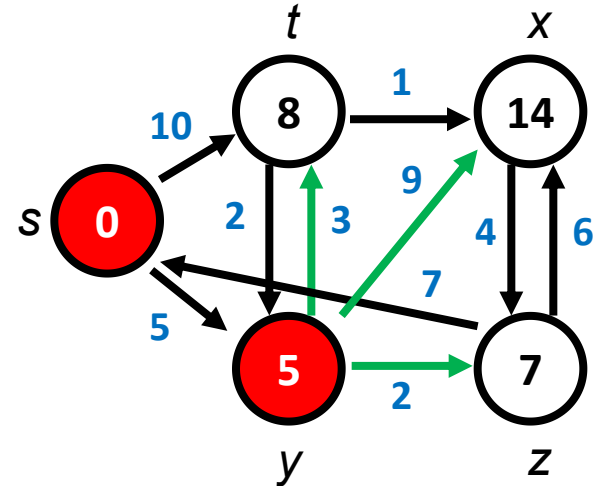
$S = \emptyset$   
 $V - S = \{s, t, y, x, z\}$

<b><math>d[s]=0</math></b>	$\Pi[s]=\text{null}$
$d[t]=\infty$	$\Pi[t]=\text{null}$
$d[y]=\infty$	$\Pi[y]=\text{null}$
$d[x]=\infty$	$\Pi[x]=\text{null}$
$d[z]=\infty$	$\Pi[z]=\text{null}$



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

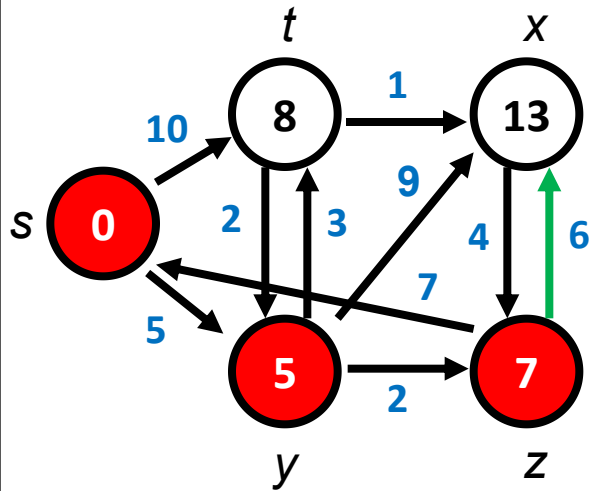
<b><math>d[s]=0</math></b>	$\Pi[s]=\text{null}$
$d[t]=10$	$\Pi[t]=s$
<b><math>d[y]=5</math></b>	$\Pi[y]=s$
$d[x]=\infty$	$\Pi[x]=\text{null}$
$d[z]=\infty$	$\Pi[z]=\text{null}$



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

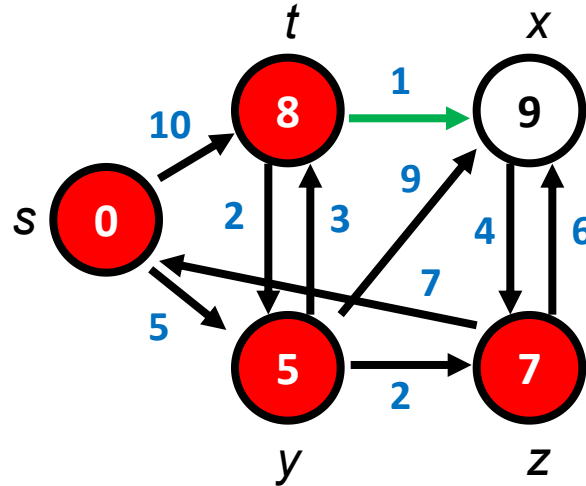
<b><math>d[s]=0</math></b>	$\Pi[s]=\text{null}$
$d[t]=8$	$\Pi[t]=y$
<b><math>d[y]=5</math></b>	$\Pi[y]=s$
$d[x]=14$	$\Pi[x]=y$
<b><math>d[z]=7</math></b>	$\Pi[z]=y$

# Ejecutando el algoritmo de Dijkstra



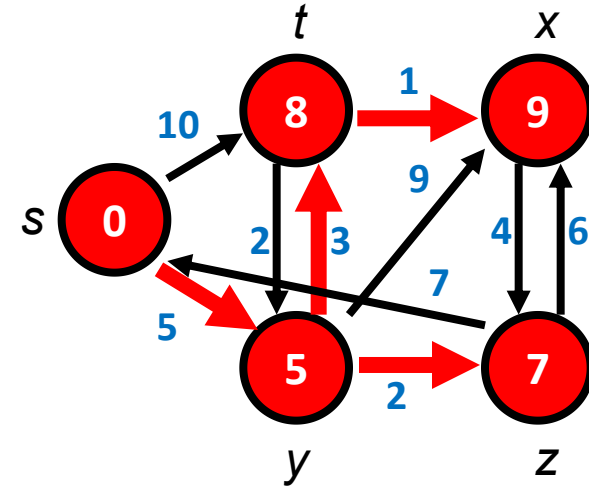
$S = \{s, y, z\}$   
 $V - S = \{t, x\}$

$d[s] = 0$      $\Pi[s] = \text{null}$   
 $d[t] = 8$      $\Pi[t] = y$   
 $d[y] = 5$      $\Pi[y] = s$   
 $d[x] = 13$      $\Pi[x] = z$   
 $d[z] = 7$      $\Pi[z] = y$



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

$d[s] = 0$      $\Pi[s] = \text{null}$   
 $d[t] = 8$      $\Pi[t] = y$   
 $d[y] = 5$      $\Pi[y] = s$   
 $d[x] = 9$      $\Pi[x] = t$   
 $d[z] = 7$      $\Pi[z] = y$



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

$d[s] = 0$      $\Pi[s] = \text{null}$   
 $d[t] = 8$      $\Pi[t] = y$   
 $d[y] = 5$      $\Pi[y] = s$   
 $d[x] = 9$      $\Pi[x] = t$   
 $d[z] = 7$      $\Pi[z] = y$

# 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   $S \leftarrow \emptyset$ 
3   $Q \leftarrow V[G]$ 
4  while  $Q \neq \emptyset$ 
5      do  $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6           $S \leftarrow S \cup \{u\}$ 
7          for each vertex  $v \in \text{Adj}[u]$ 
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,  $d[u] = \delta(s, u)$  para el vértice  $u$  que se inserta en  $S$

Supongamos que esto no es cierto:

Sea  $u$  el primer vértice para el cual  $d[u] \neq \delta(s, u)$  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:

**$p_1$** : subcamino de  $s$  a  $x$  (todos sus vértices en  $S$ )

**$p_2$** : subcamino de  $y$  a  $u$  (todos sus vértices en  $V-S$  ó en  $S$ )

**$y$** : primer vértice en  $p$  t.q.  $y \in V-S$

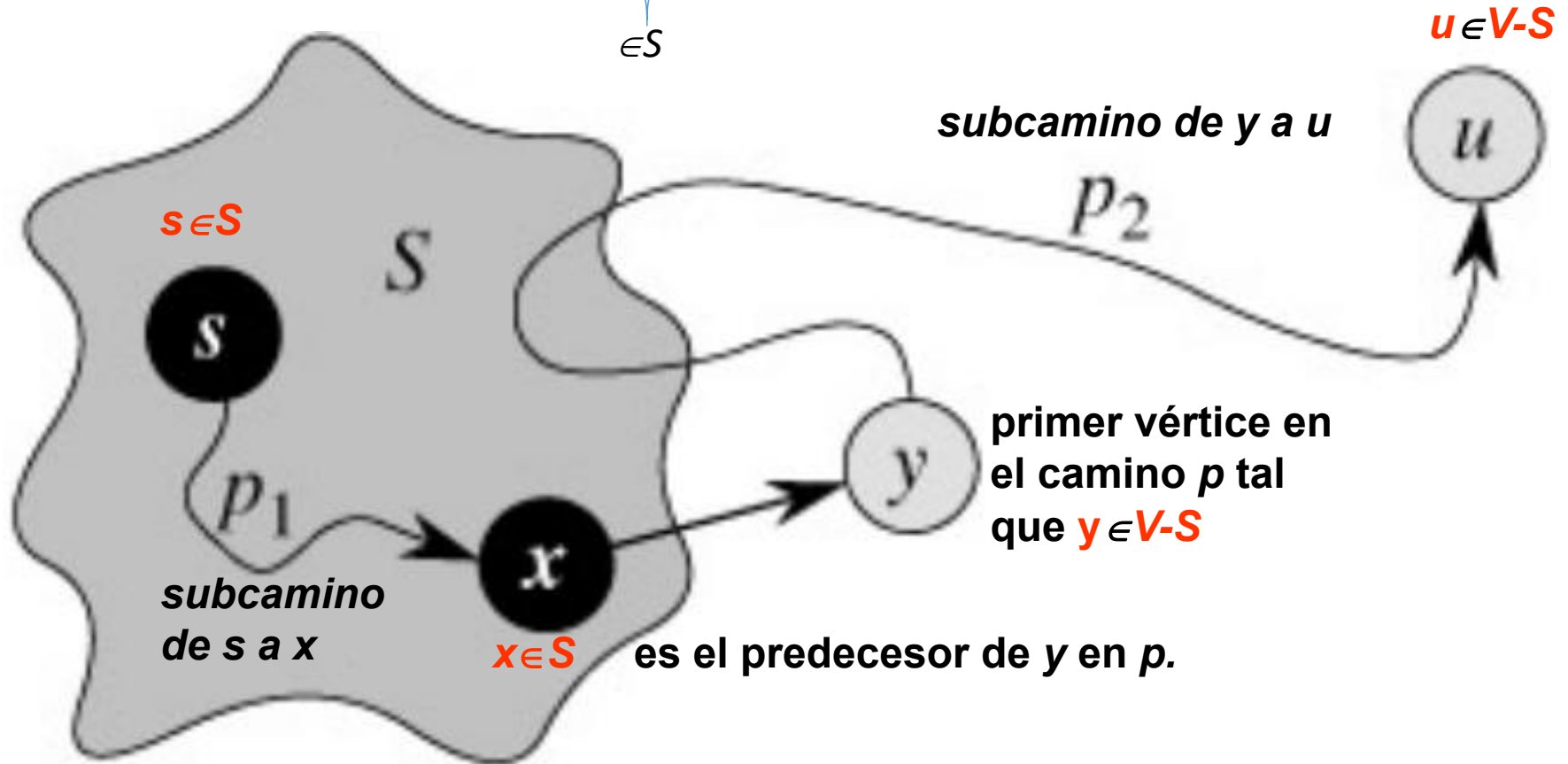
**$x$** : último vértice en  $p$  t.q.  $x \in S$  (predecesor de  $y$  en el camino  $p$ )

Tanto  $p_1$  como  $p_2$  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:

- $x \neq y$  pero podría suceder  $s = x$  o  $y = u$
- el subcamino  $p_2$  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:

- $x \in S$ , por tanto, como asumimos que  $u$  es el primer vértice para el cual  $d[u] \neq \delta(s, u)$ , cuando dicho vértice se adiciona a  $S$ , se tenía que  $d[x] = \delta(s, x)$ .
- Cuando  $x$  se insertó en  $S$ , al arco  $(x, y)$  se le hizo **RELAX** y en ese momento se hizo  $d[y] = \delta(s, y)$  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,

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

$$\begin{aligned} d[y] &= \delta(s, y) \\ &\leq \delta(s, u) \end{aligned} \quad (*)$$

$$\leq d[u] \quad \text{por la propiedad de la Cota Superior}$$

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:  $d[u] = d[y] = \delta(s, y) = \delta(s, u) \rightarrow$  **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,

$$d[u] = \delta(s, u) \quad \forall u \in V$$

# Complejidad temporal del algoritmo de Dijkstra

DIJKSTRA( $G, w, s$ )

```
1  INITIALIZE-SINGLE-SOURCE( $G, s$ )
2   $S \leftarrow \emptyset$ 
3   $Q \leftarrow V[G]$   $\longrightarrow O(V \cdot T(\text{INSERT en } Q))$  Cada vértice se inserta una sola vez
4  while  $Q \neq \emptyset$ 
5      do  $u \leftarrow \text{EXTRACT-MIN}(Q)$   $\longrightarrow O(V \cdot T(\text{EXTRACT-MIN}))$ 
6           $S \leftarrow S \cup \{u\}$  Cada vértice se elimina una sola vez
7          for each vertex  $v \in \text{Adj}[u]$ 
8              do  $\text{RELAX}(u, v, w)$   $\longrightarrow O(E \cdot T(\text{DECREASE-KEY}))$ 
                                     implícito en el RELAX
```

- Cada arco en  $\text{Adj}[u]$  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



$$T(\text{Dijkstra}) = O(V \cdot T(\text{INSERT}) + V \cdot T(\text{EXTRACT-MIN}) + E \cdot T(\text{DECREASE-KEY}))$$

# Complejidad temporal del algoritmo de Dijkstra

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



En este caso **EXTRACTMIN**  $\Leftrightarrow$  **BUSCAR-MIN** en un *array*

- **INSERT** y **DECREASE-KEY** son  $O(1) \Rightarrow O(V \cdot T(\text{INSERT})) = O(V)$   
 $O(E \cdot T(\text{DECREASE-KEY})) = O(E)$
- **BUSCAR-MIN** es  $O(V)$  (necesidad de buscar por todo el arreglo)  
 $\Rightarrow O(V \cdot T(\text{EXTRACTMIN})) = O(V \cdot V) = O(|V|^2)$

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

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  $O(\log |V|)$

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

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

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

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

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

$$\begin{aligned} T(\text{Dijkstra}) &= O(V) + O(V \log V) + O(E \log V) = O(V \log V) + O(E \log V) \\ &= O((V + E) \log V) = O(E \log V) \end{aligned}$$



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  $E = V - 1 \Rightarrow$  en  $G$ ,  $E \geq V - 1$ , o sea,  $E$  domina  $V$

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