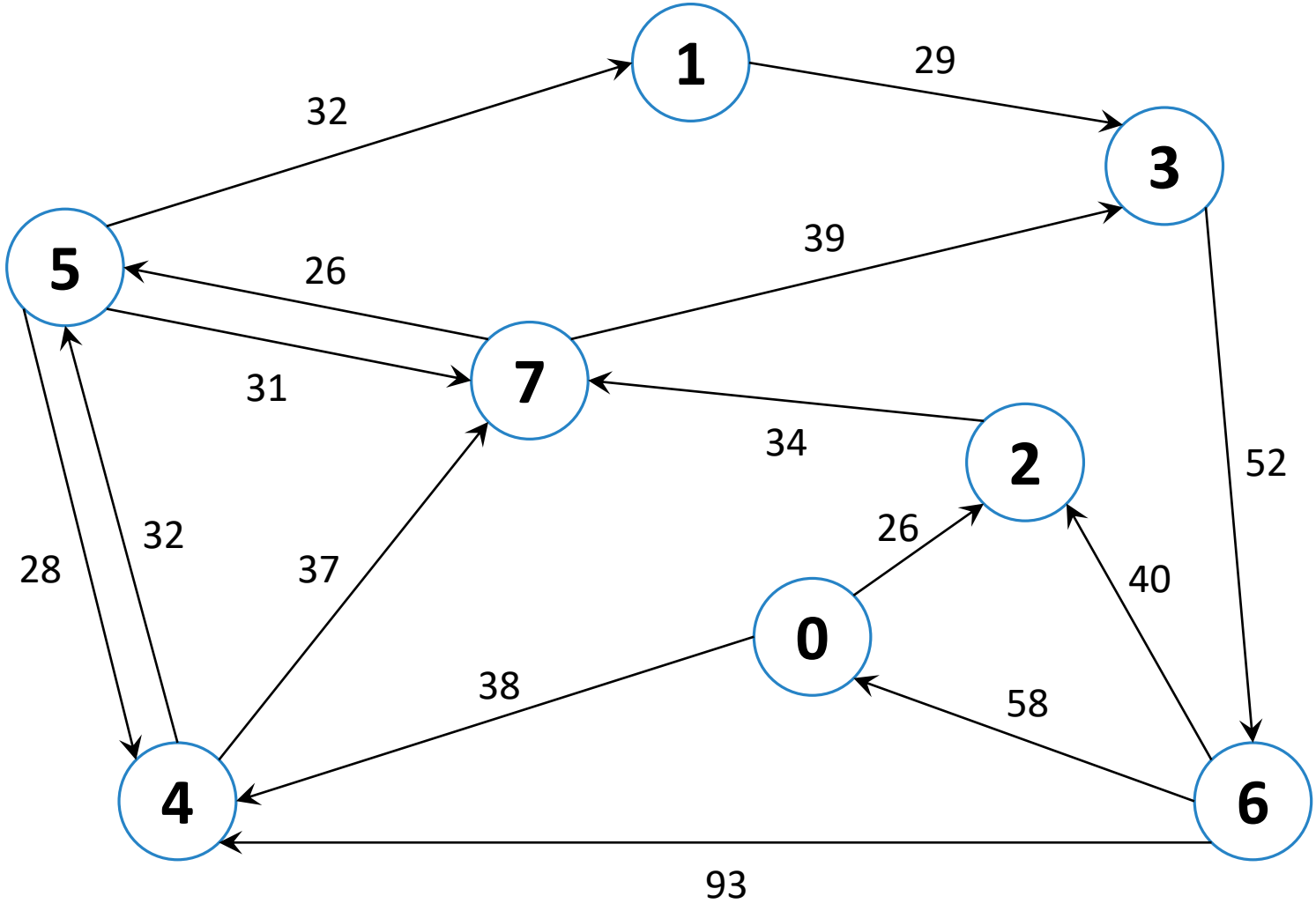
El viaje familiar

- Quieres planificar un viaje en auto desde A a B
- Los caminos tienen peajes y tiempos de recorrido

¿Cómo hacer para que el viaje te salga lo más barato posible?

¿Y lo más corto posible?

Grafo direccional con costos



La ruta más barata (o la más corta)

Debemos buscar la **ruta más barata** de A a B, es decir,

… la **suma de los costos** de sus aristas debe ser mínima

¿Podremos aprovechar un algoritmo que conozcamos?

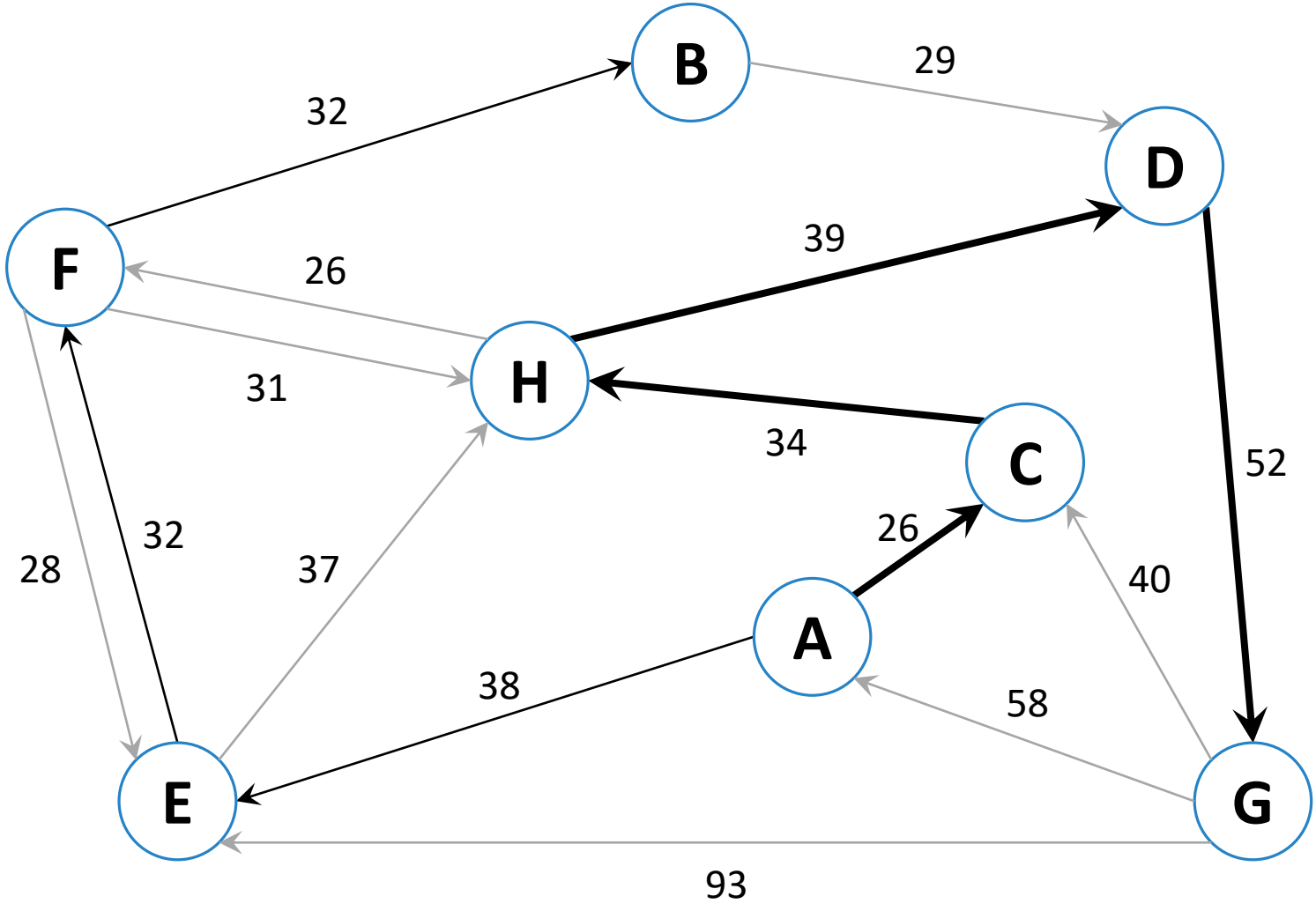
Propiedades de BFS

¿Cuál es la propiedad que garantiza la corrección de **BFS**?

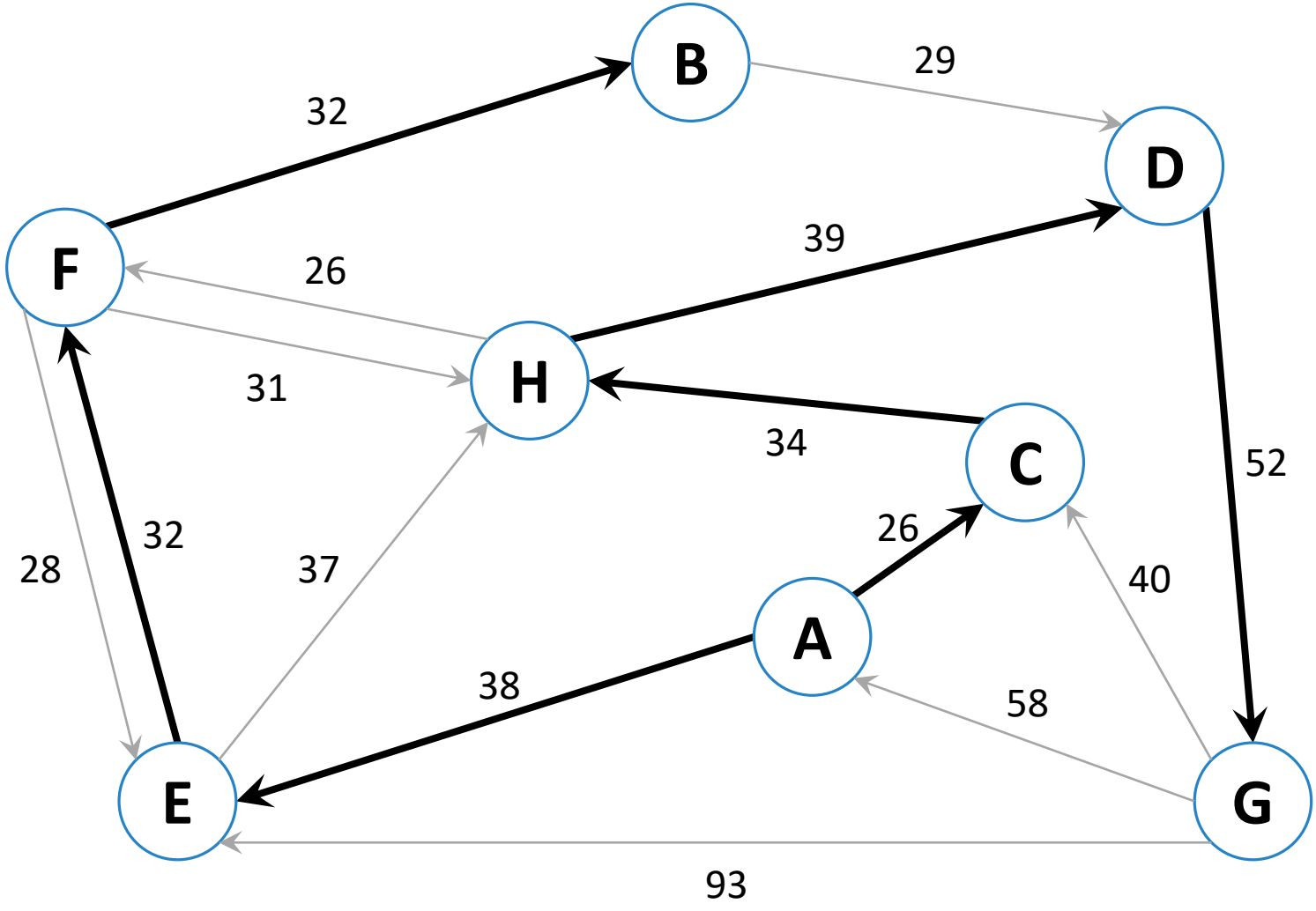
Si **marcamos** cada arista que queda como padre de un nodo,

… ¿qué **representa** el conjunto de aristas marcadas?

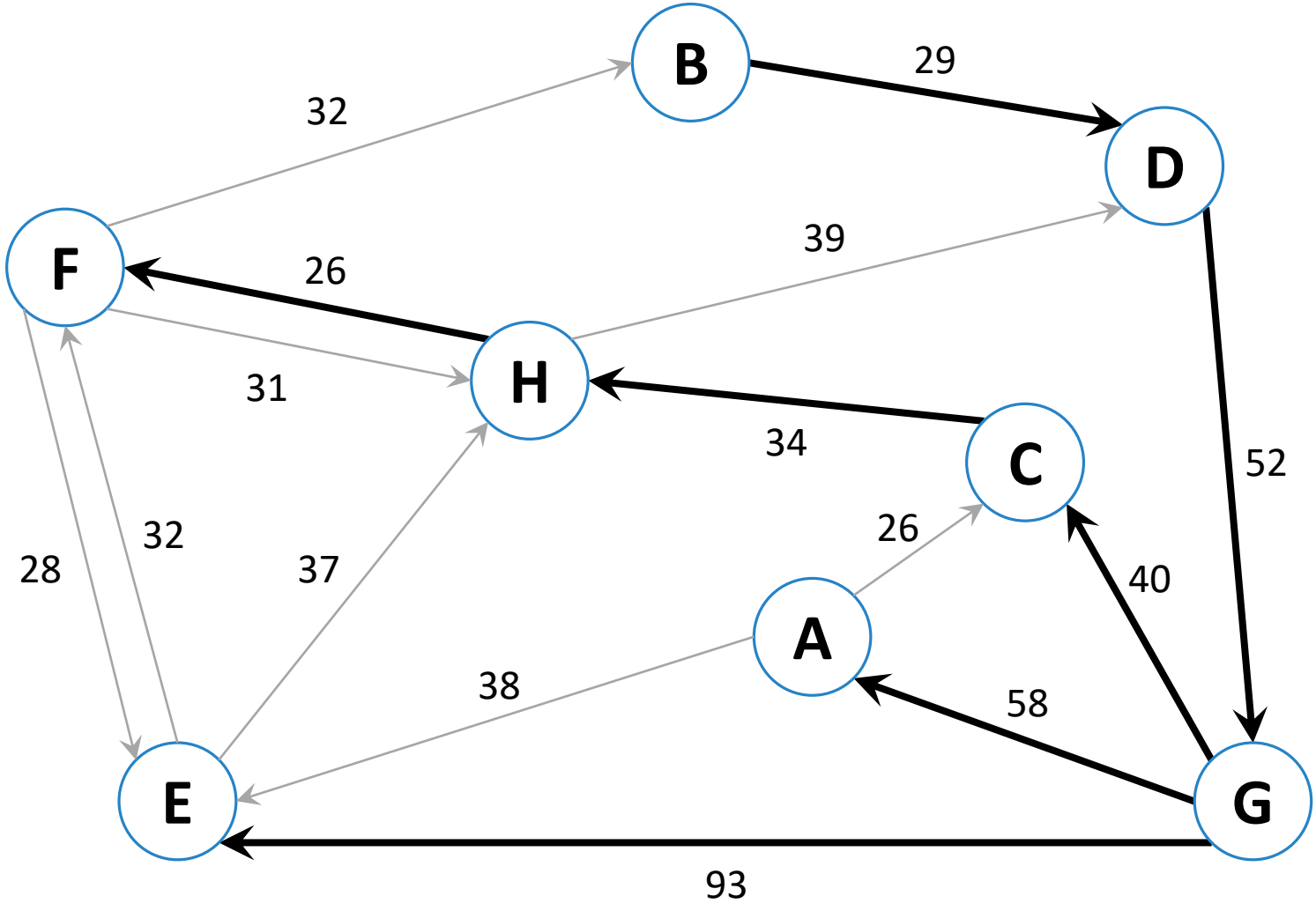
Ruta más cortas de *A* a *G*



Árbol de rutas más cortas desde *A*



Árbol de rutas más cortas desde *B*



Propiedades del problema de rutas más cortas

Las rutas son direccionales

Los costos pueden representar distancias, tiempos de viajes, consumo de combustible, costos de peajes, etc.

Puede que haya vértices inalcanzables desde el vértice de partida

Si hay costos negativos, es más complicado resolver el problema

Las rutas más cortas pueden no ser únicas

BFS++

¿Cómo podemos extender **BFS** para este problema?

Queremos **garantizar** que al sacar un nodo de **Open**, hemos encontrado ese nodo por la ruta más corta: de menor costo (y no necesariamente con el menor número de aristas)

$bfs(G(V,E), s, g)$:

$s.dist \leftarrow 0$

$Open \leftarrow$ una cola conteniendo únicamente a s

while $Open \neq \emptyset$:

$u \leftarrow$ extraer y pintar el siguiente elemento de $Open$

if $u = g$, *return true*

foreach $(u, v) \in E$:

if v está pintado, *continue*

$d \leftarrow u.dist + 1$

if $d \geq v.dist$, *continue*

$v.parent \leftarrow u$, $v.dist \leftarrow d$

Insertar v en $Open$

return false

$bfs++(G(V,E), s, g)$:

$s.dist \leftarrow 0$

$Open \leftarrow$ una cola de prioridades conteniendo únicamente a s

while $Open \neq \emptyset$:

$u \leftarrow$ extraer y pintar el siguiente elemento de $Open$

if $u = g$, *return true*

foreach $(u, v) \in E$:

if v está pintado, *continue*

$d \leftarrow u.dist + w(u, v)$

if $d \geq v.dist$, *continue*

$v.parent \leftarrow u$, $v.dist \leftarrow d$

Insertar o actualizar v en $Open$

con prioridad $v.dist$

return false

Algoritmo de Dijkstra

Propuesto por **E.W. Dijkstra**, hace unos 60 años

Parece tener sentido, pero, ¿es **correcto**?

¿Se está cumpliendo la **garantía** que propusimos?

$dijkstra(G(V,E), s, g)$:

$s.dist \leftarrow 0$

$Open \leftarrow$ una cola de prioridades conteniendo únicamente a s

Insertar s en $Open$ con prioridad $s.dist$

while $Open \neq \emptyset$:

$u \leftarrow$ extraer y pintar el siguiente elemento de $Open$

if $u = g$, *return true*

foreach $(u, v) \in E$:

if v está pintado, *continue*

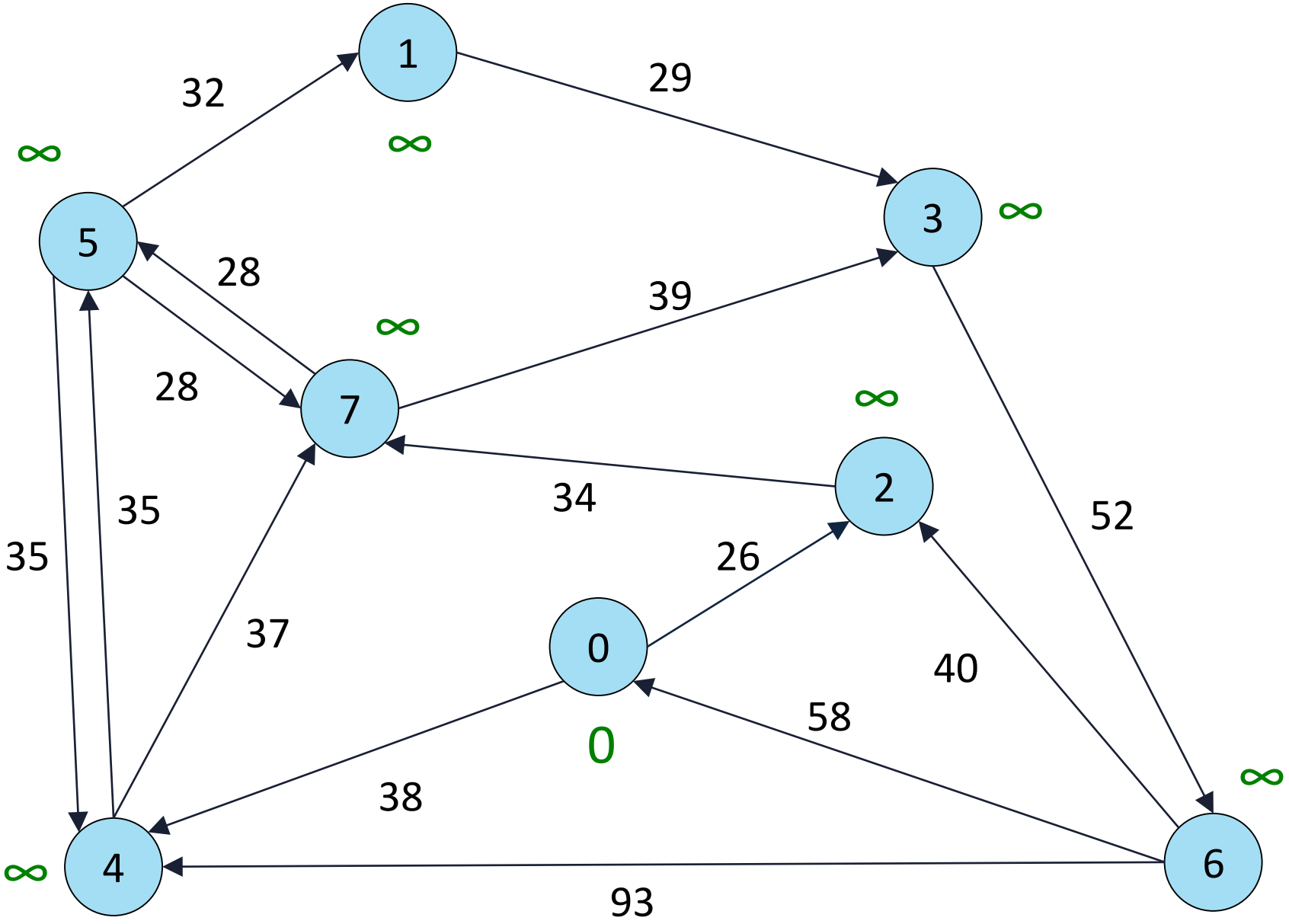
$d \leftarrow u.dist + w(u, v)$

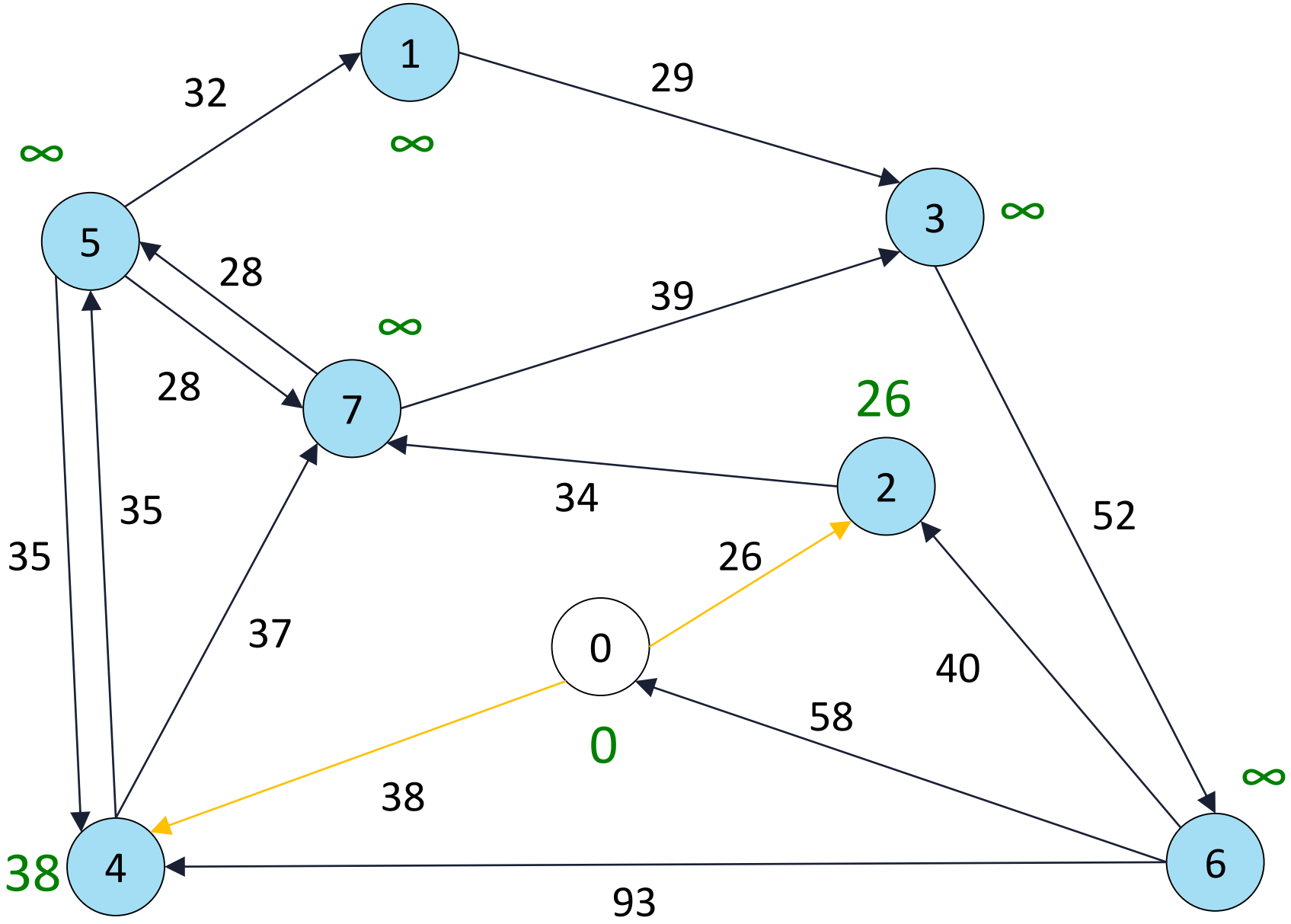
if $d \geq v.dist$, *continue*

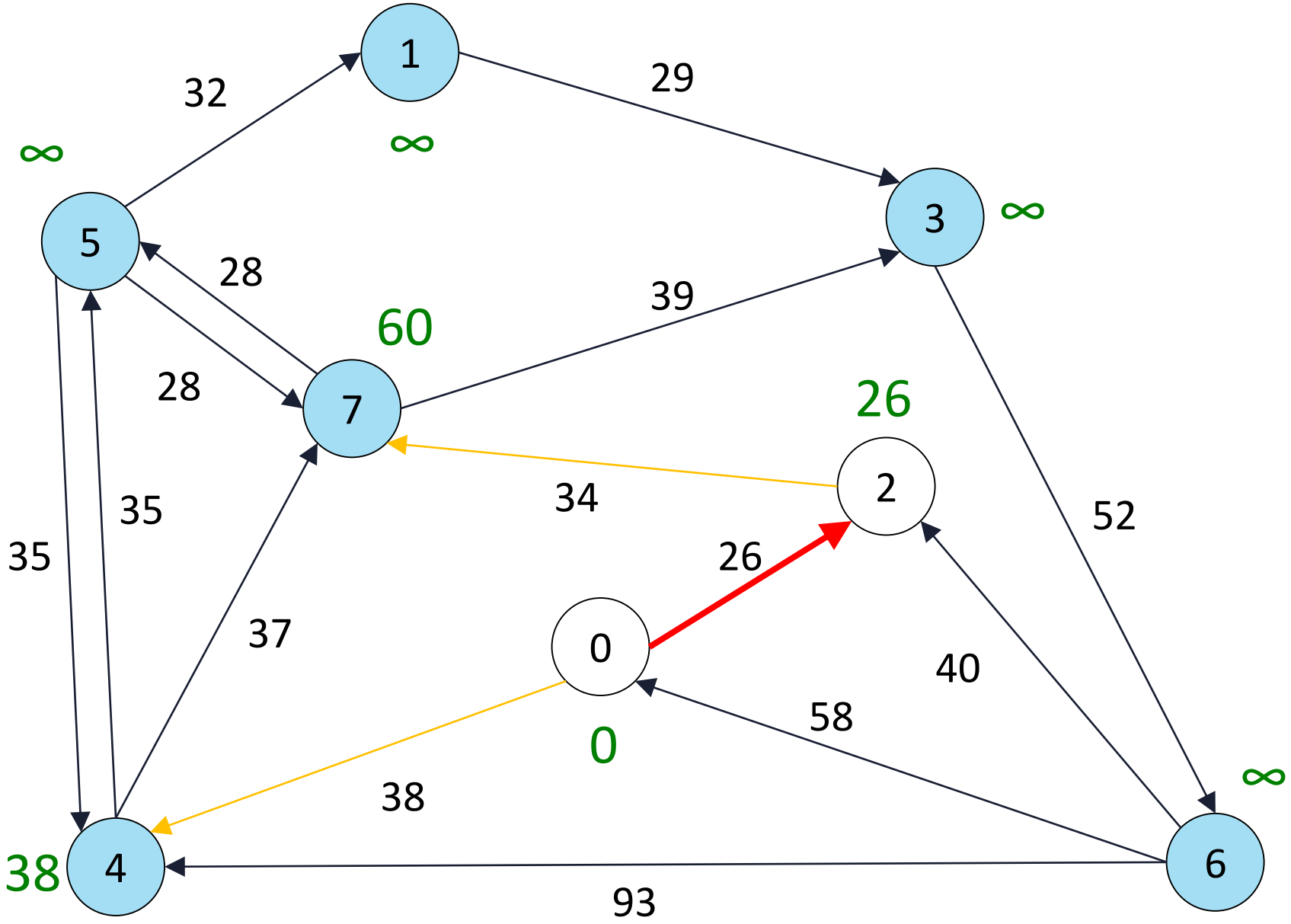
$v.parent \leftarrow u$, $v.dist \leftarrow d$

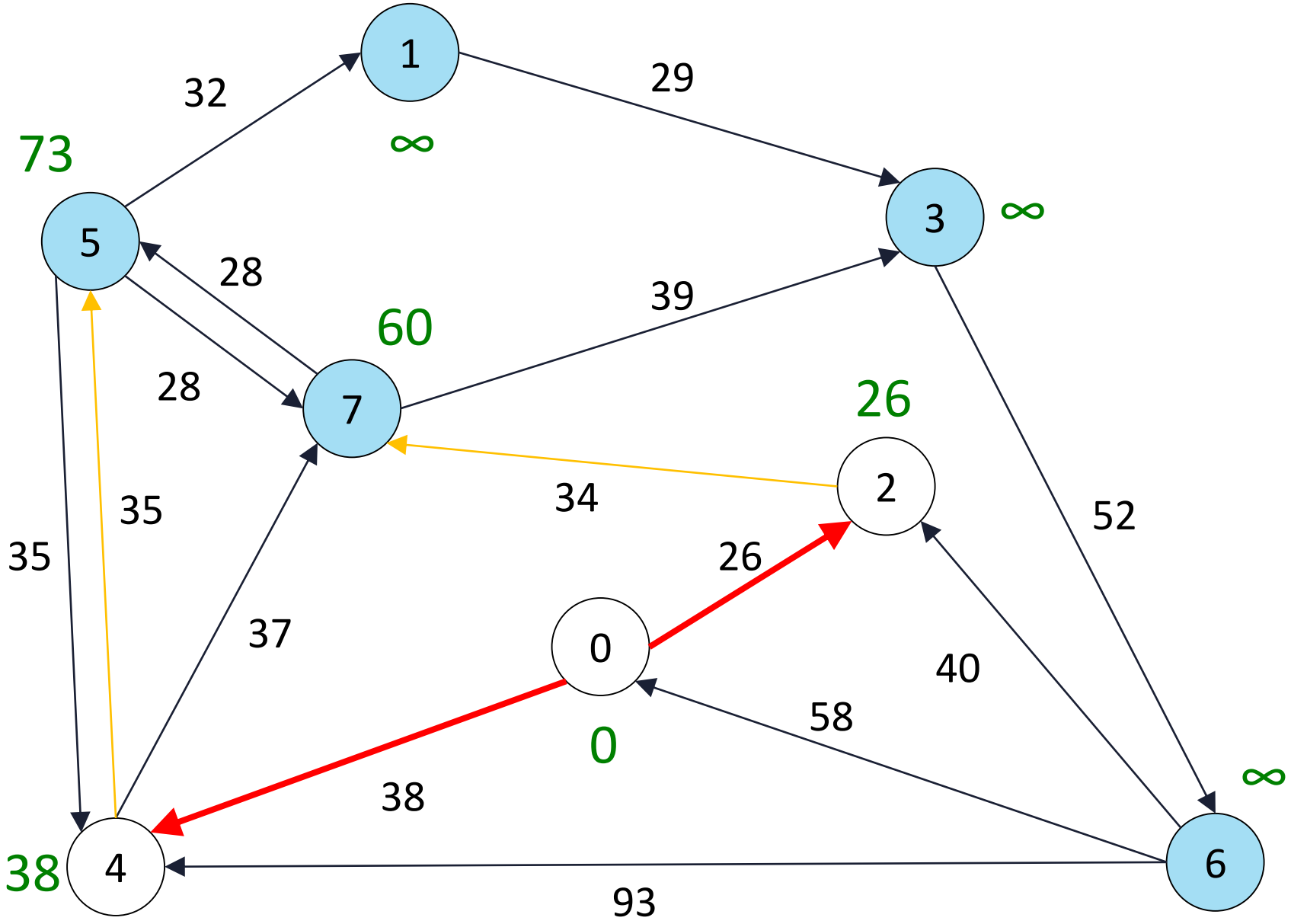
Insertar o actualizar v en $Open$ con prioridad $v.dist$

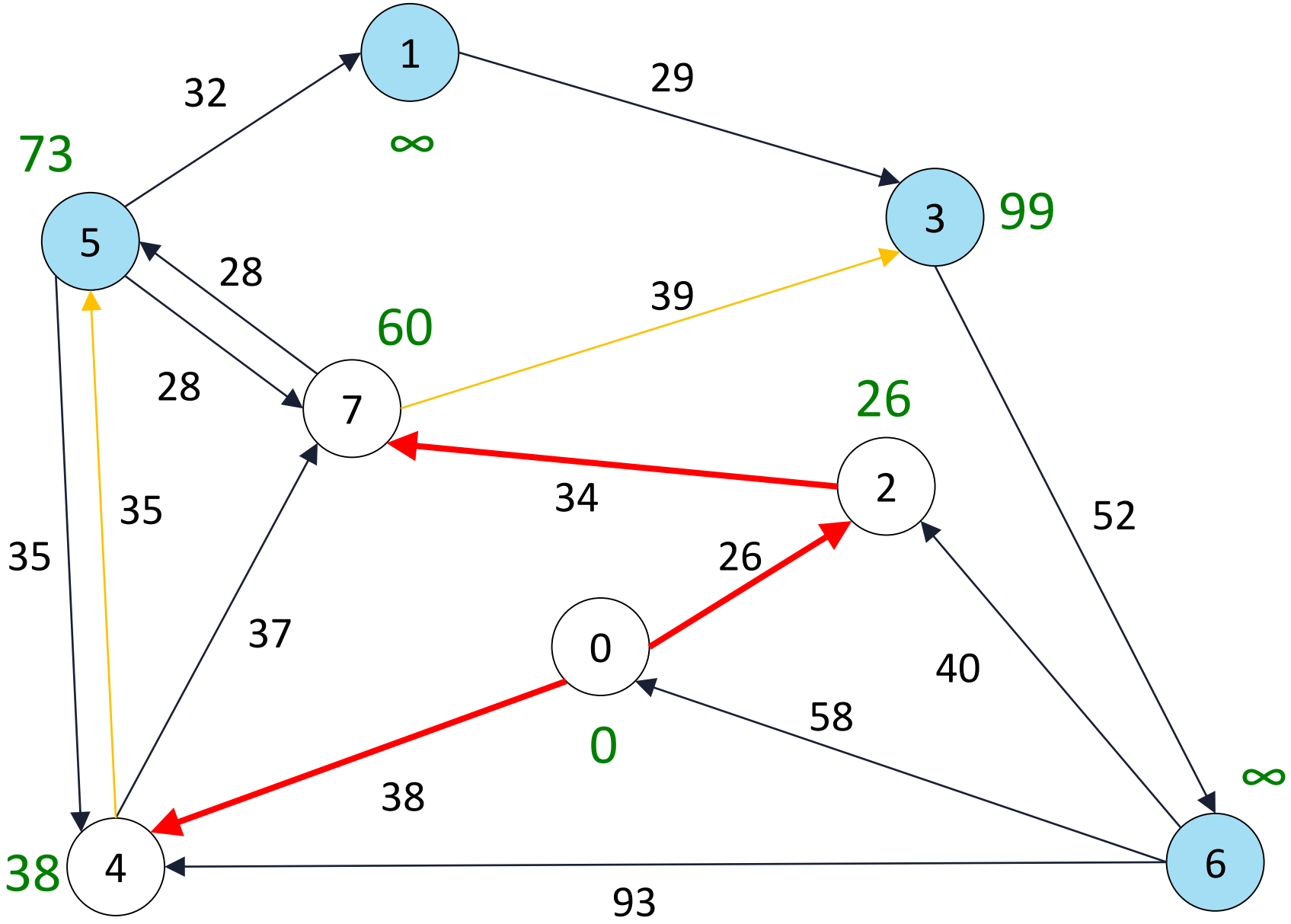
return false

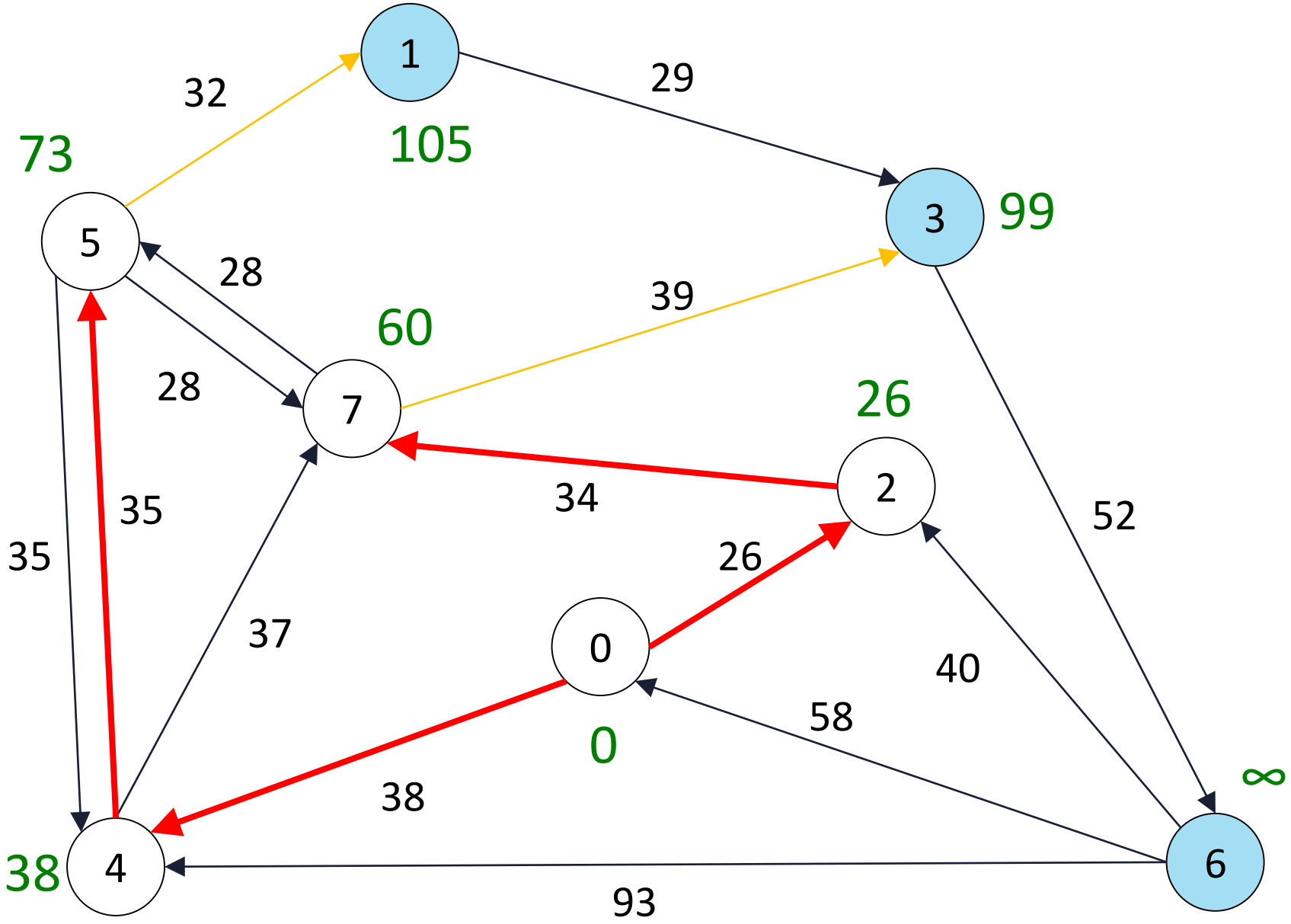

1
2
3
4
5
6
7
0
32
29
28
39
28
35
35
34
52
26
37
40
58
38
93
∞
∞
∞
∞
∞
∞
∞
0

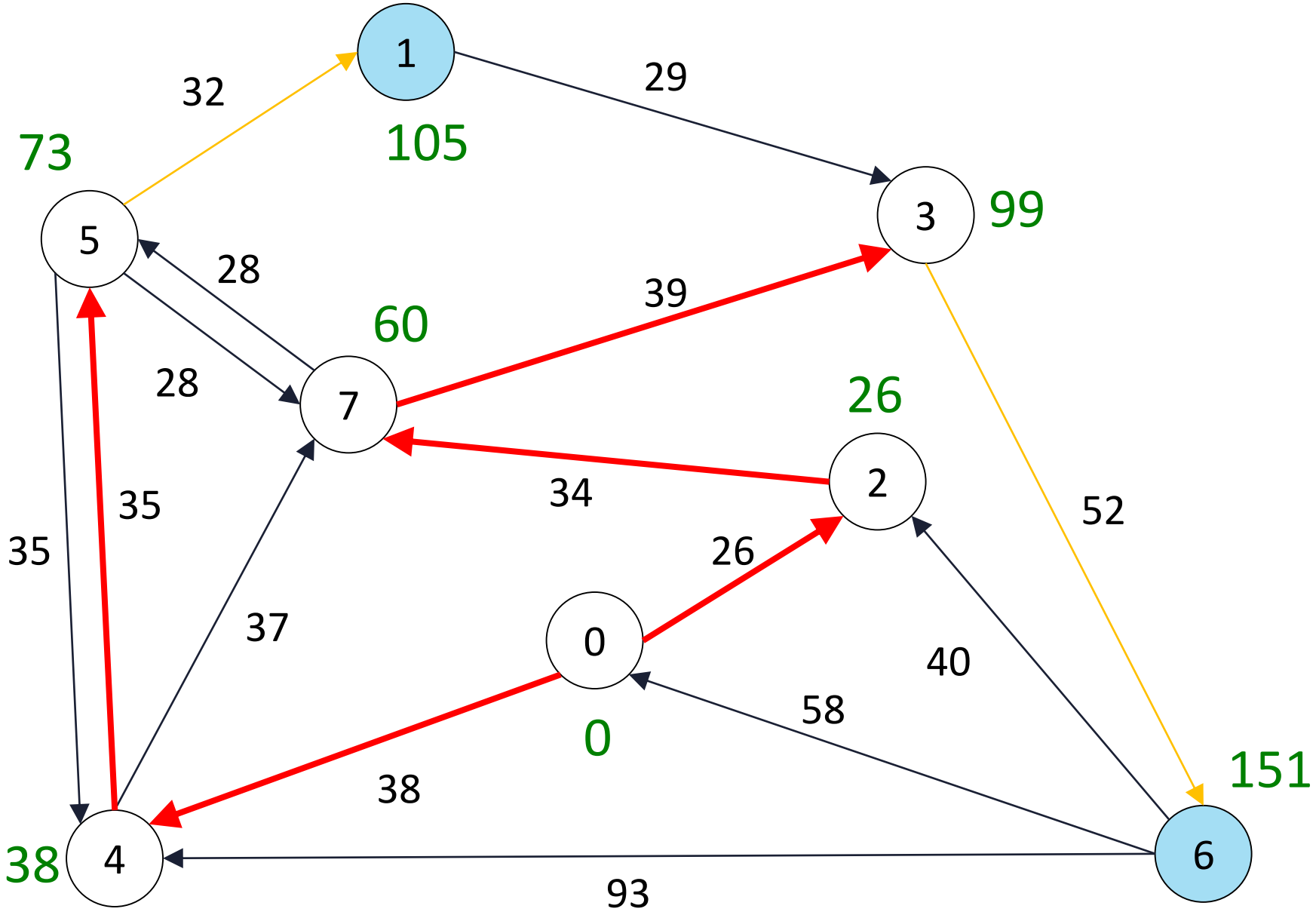

1
32
29
∞
∞
5
3
∞
28
39
∞
28
7
26
2
35
34
52
35
26
37
0
40
58
0
38
∞
38
4
6
93

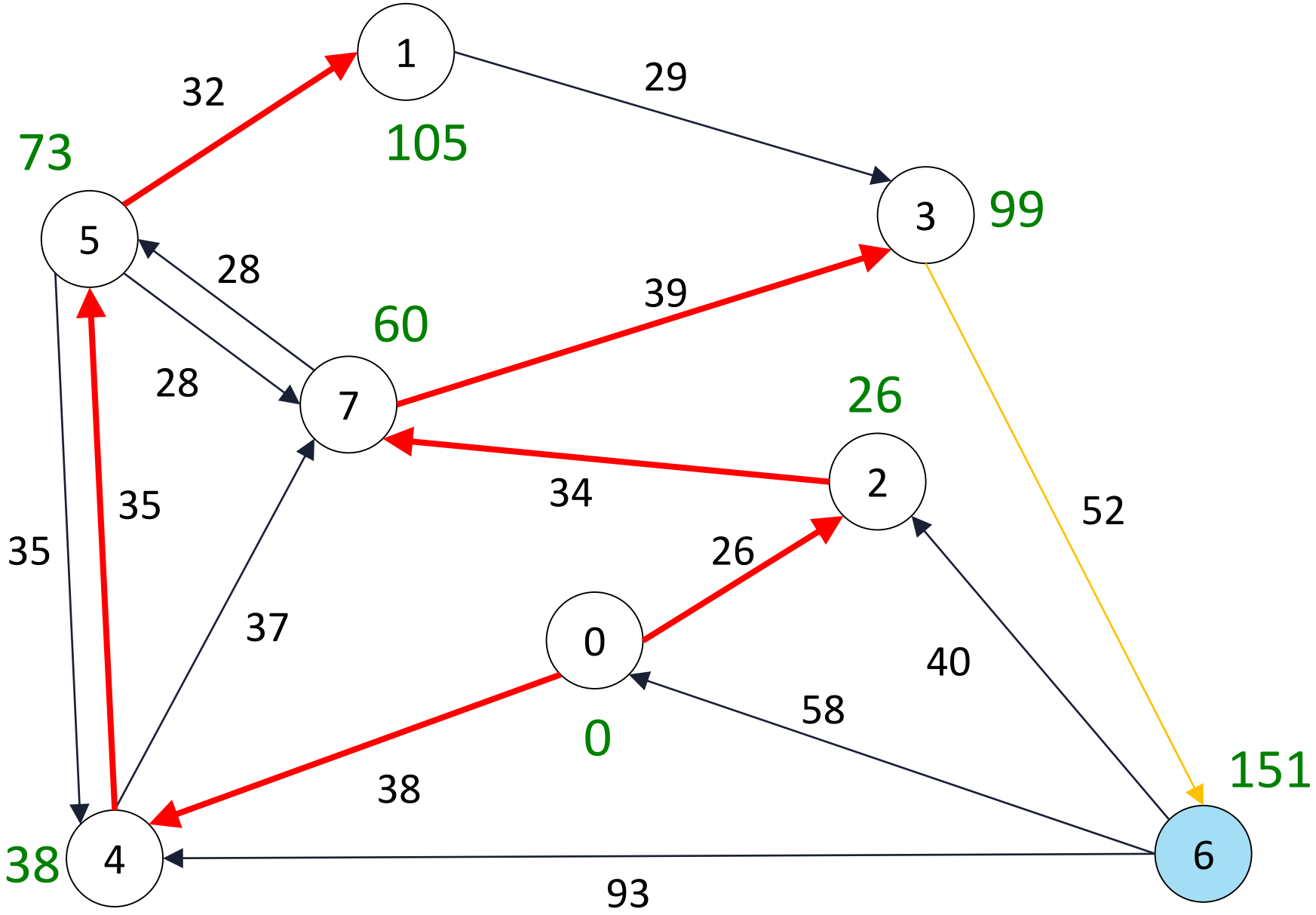

1
2
3
4
5
6
7
0
32
29
28
28
39
60
26
34
35
35
52
26
37
40
58
0
38
93
∞
∞
∞
∞
38

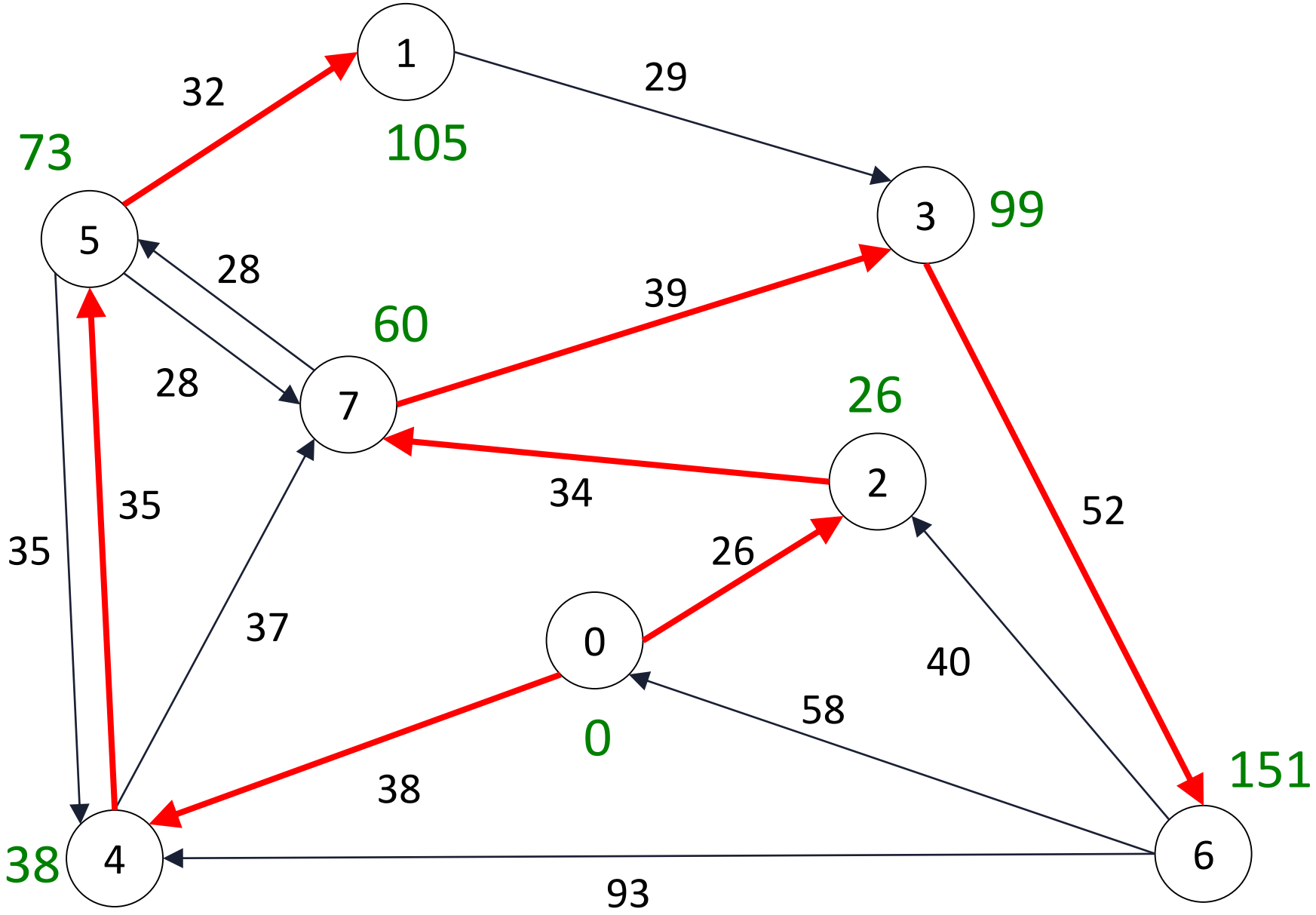

1
∞
3
∞
5
73
7
60
2
26
0
0
4
38
6
∞
32
29
28
39
28
35
35
34
52
26
37
40
58
38
93


1
∞
3
99
5
73
7
60
2
26
0
0
4
38
6
∞
32
29
28
28
39
35
35
34
52
26
37
40
58
38
93


1
105
3
99
5
73
7
60
2
26
0
0
4
38
6
∞
32
29
28
28
39
35
35
34
52
26
37
40
58
38
93


1
105
32
29
73
5
3
99
28
39
60
28
7
26
2
34
35
52
35
26
37
0
40
58
0
151
38
4
6
38
93


1
32
29
105
73
5
3
99
28
39
60
28
7
26
2
34
35
52
35
26
37
0
40
58
0
151
38
4
6
38
93


1
105
32
29
73
5
3
99
28
39
60
28
7
26
2
34
35
52
35
26
37
0
40
58
0
151
38
38
4
6
93

Una ruta más corta cumple la propiedad de *subestructura óptima*

Los algoritmos para encontrar rutas más cortas usan la siguiente propiedad:

Todas las subrutas en una ruta más corta *p* entre dos vértices v_0 y v_k son también rutas más cortas

Si $p = \langle v_0, v_1, \ldots, v_k \rangle$

… sea $p_{ij} = \langle v_i, \ldots, v_j \rangle$, $0 \leq i \leq j \leq k$

… entonces p_{ij} es una ruta más corta de v_i a v_j

La propiedad de *desigualdad triangular*

Sea δ(s,v) el costo de la ruta más corta de *s* a *v*

Si la (una) ruta más corta de *s* a *v* puede descomponerse en una ruta de *s* a *u* seguida de la arista (*u*,*v*)

… entonces $\delta(s,v) = \delta(s,u) + \omega(u,v)$

… y para todas las aristas $(r,v) \in E$

… $\delta(s,v) \leq \delta(s,r) + \omega(r,v)$

Algoritmos codiciosos

El algoritmo de **Dijkstra** es **codicioso**

Estos algoritmos no necesariamente producen soluciones **óptimas**

¿Por qué funciona el enfoque codicioso en este caso?

1. Sea u el primer vértice tal que $d[u] \neq \delta(s, u)$ al ingresar a S

2. Sean p la ruta más corta de s a u

   y el primer vértice en p tal que $y \notin S$

   $x \in S$ el predecesor de y : $d[x] = \delta(s, x)$

3. Como la arista (x, y) fue reducida al ingresar x a S, entonces $d[y] = \delta(s, y)$ al ingresar u a S

4. Como y aparece antes que u en p y todos los costos son ≥ 0, entonces $\delta(s, y) \leq \delta(s, u)$

   … y como $d[y] = \delta(s, y)$ y $d[u] \geq \delta(s, u)$, entonces $d[y] \leq d[u]$

5. Pero u fue elegido antes que y para ingresar a S, por lo que deducimos que $d[u] \leq d[y]$

6. Estas dos desigualdades implican que $d[u] = \delta(s, u)$

¿Cuál es la complejidad del algoritmo?

`Dijkstra` realiza $|V|$ **`extraer`**'s y $|E|$ actualizaciones $\boldsymbol{v.dist \leftarrow d}$

Si la cola `Open` es implementada como un heap binario,

… entonces cada extracción de $\boldsymbol{u}$ y cada actualización de $\boldsymbol{v.dist}$ toma tiempo $O(\log V)$

Así, `Dijkstra` toma tiempo $O((V+E)\log V)$

Variantes

Rutas más cortas en grafos acíclicos

Rutas más cortas de un vértice a otro

Rutas más cortas entre todos los pares de vértices

Rutas más cortas en grafos euclideanos