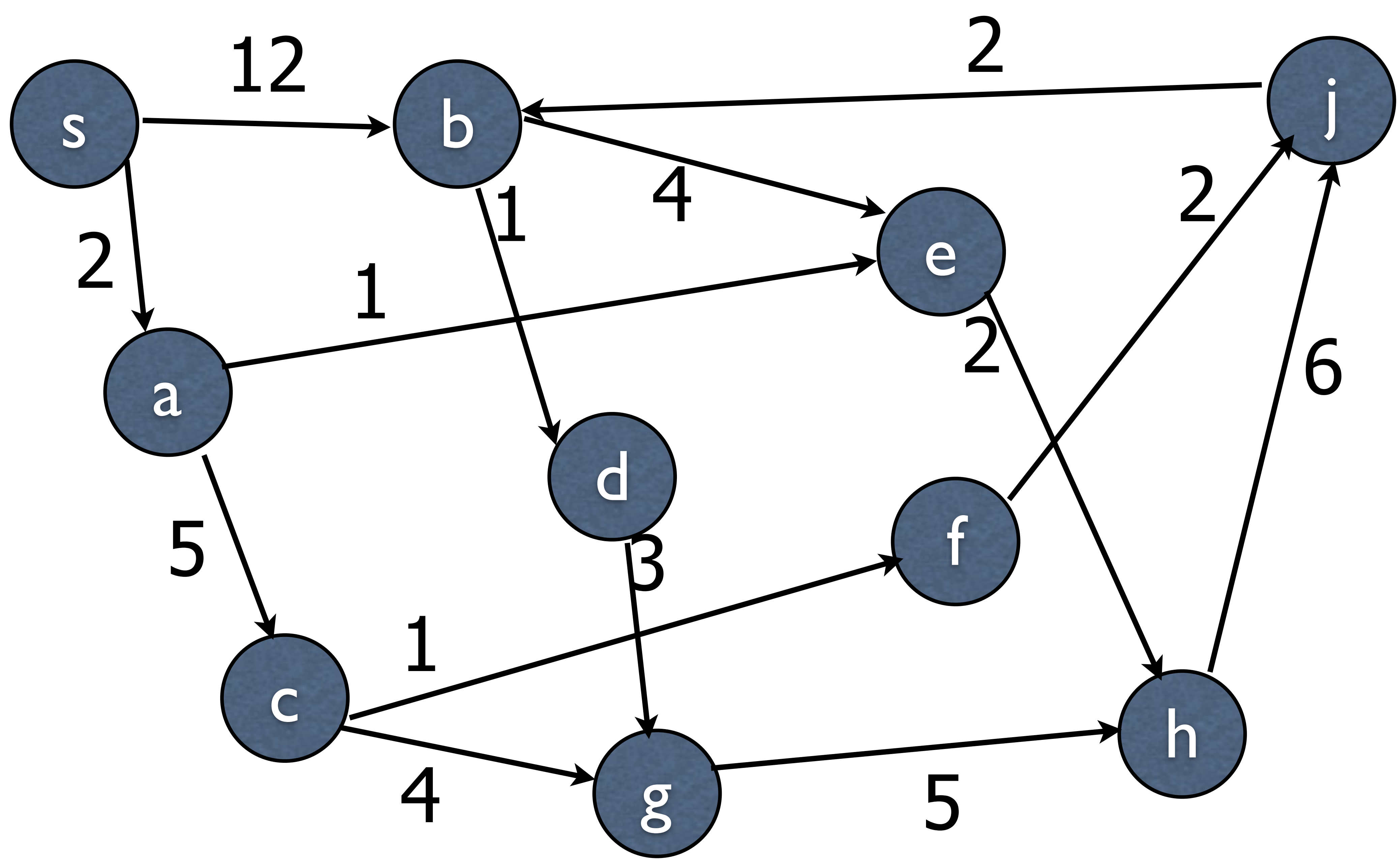
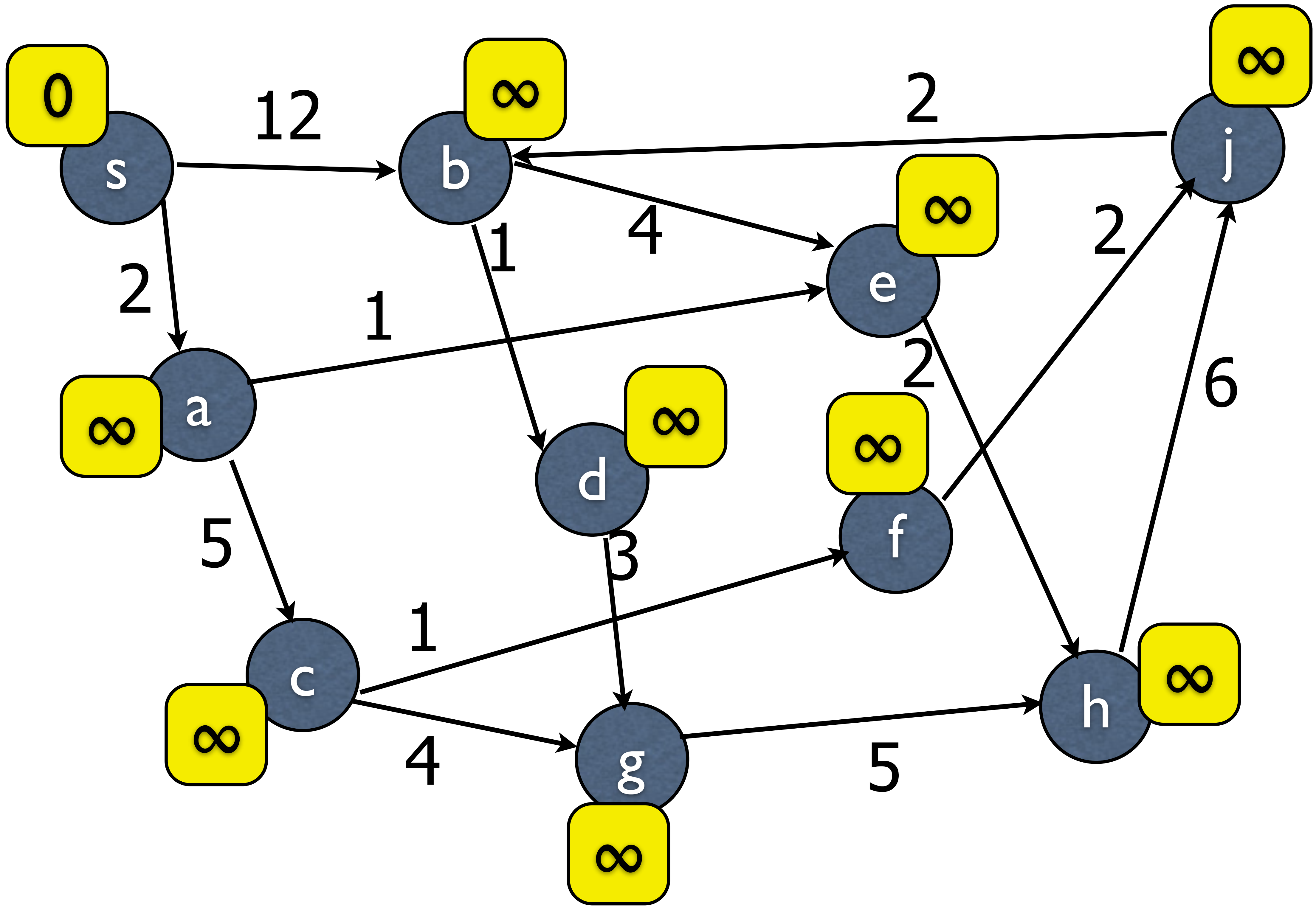


Given directed graph with non-negative edge lengths $l_{u,v}$. Compute all shortest paths from s to every other vertex.

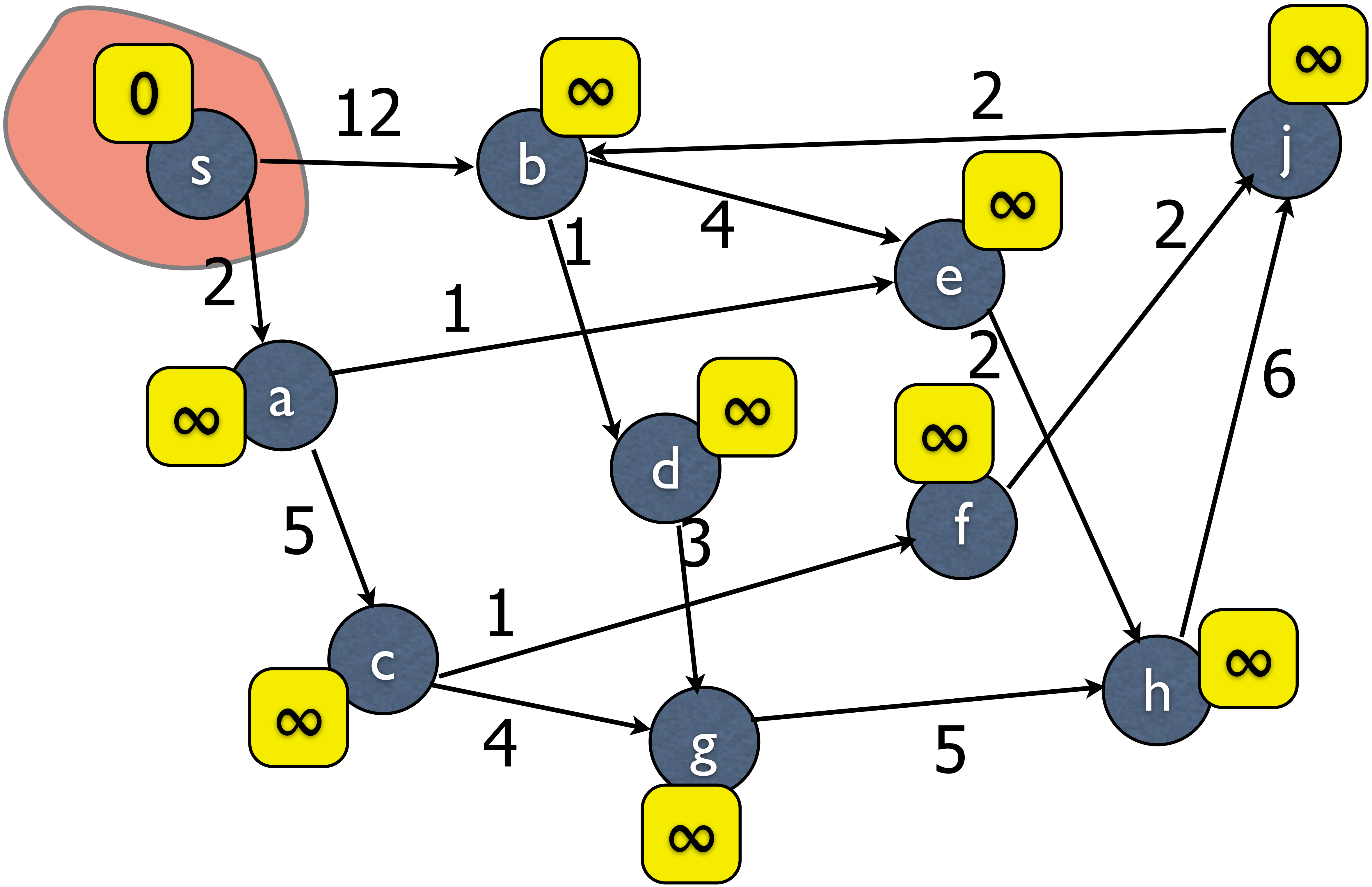
Disjkstra(s)

Set all vertices v undiscovered, d(v)= ∞
Set d(s) = 0, mark s discovered.
while there is edge from discovered vertex to undiscovered vertex,
    let (u,v) be such edge minimizing $d(u)+l_{u,v}$
    set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

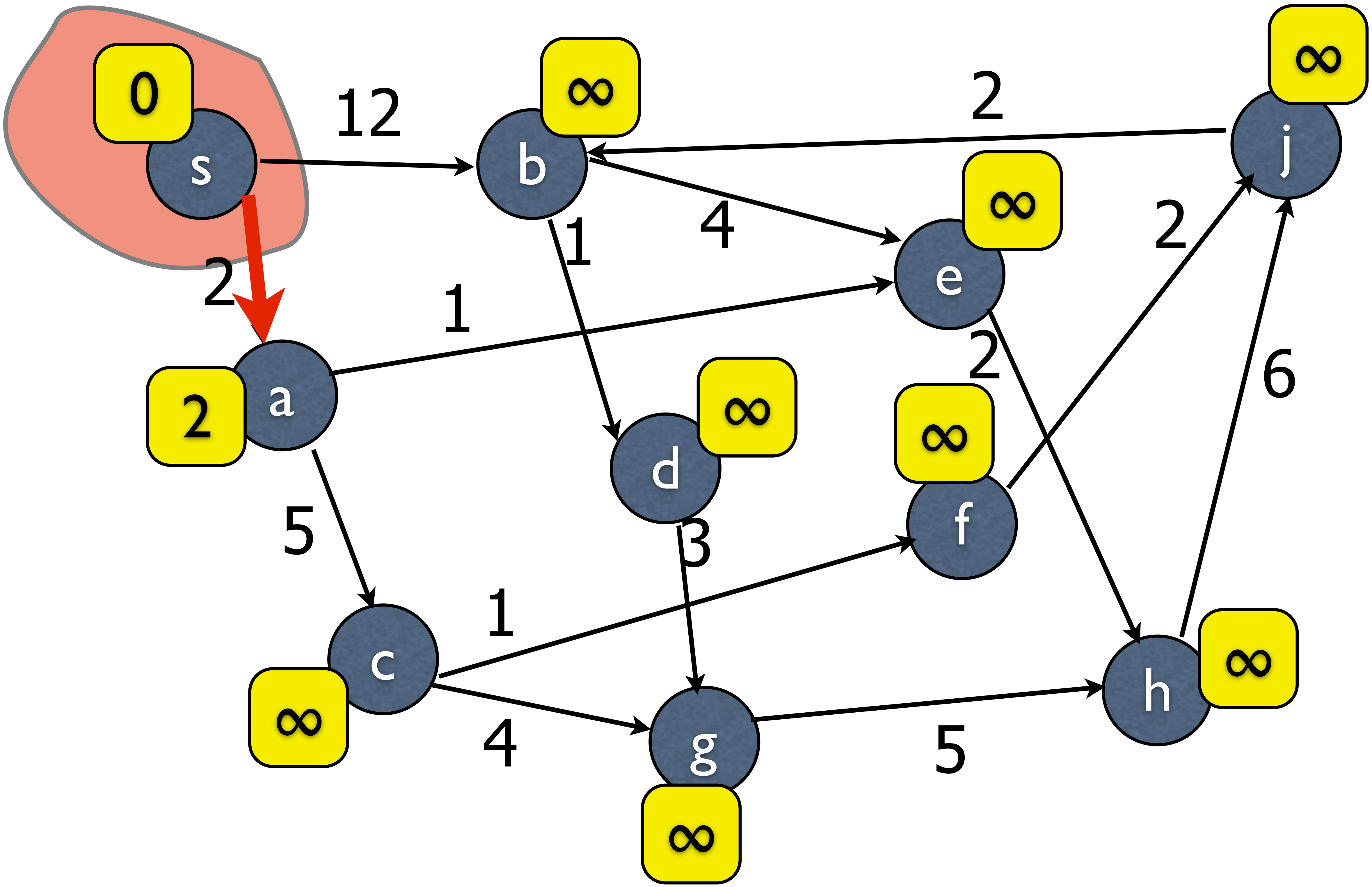


Dijkstra's Algorithm



while there is edge from discovered vertex to undiscovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

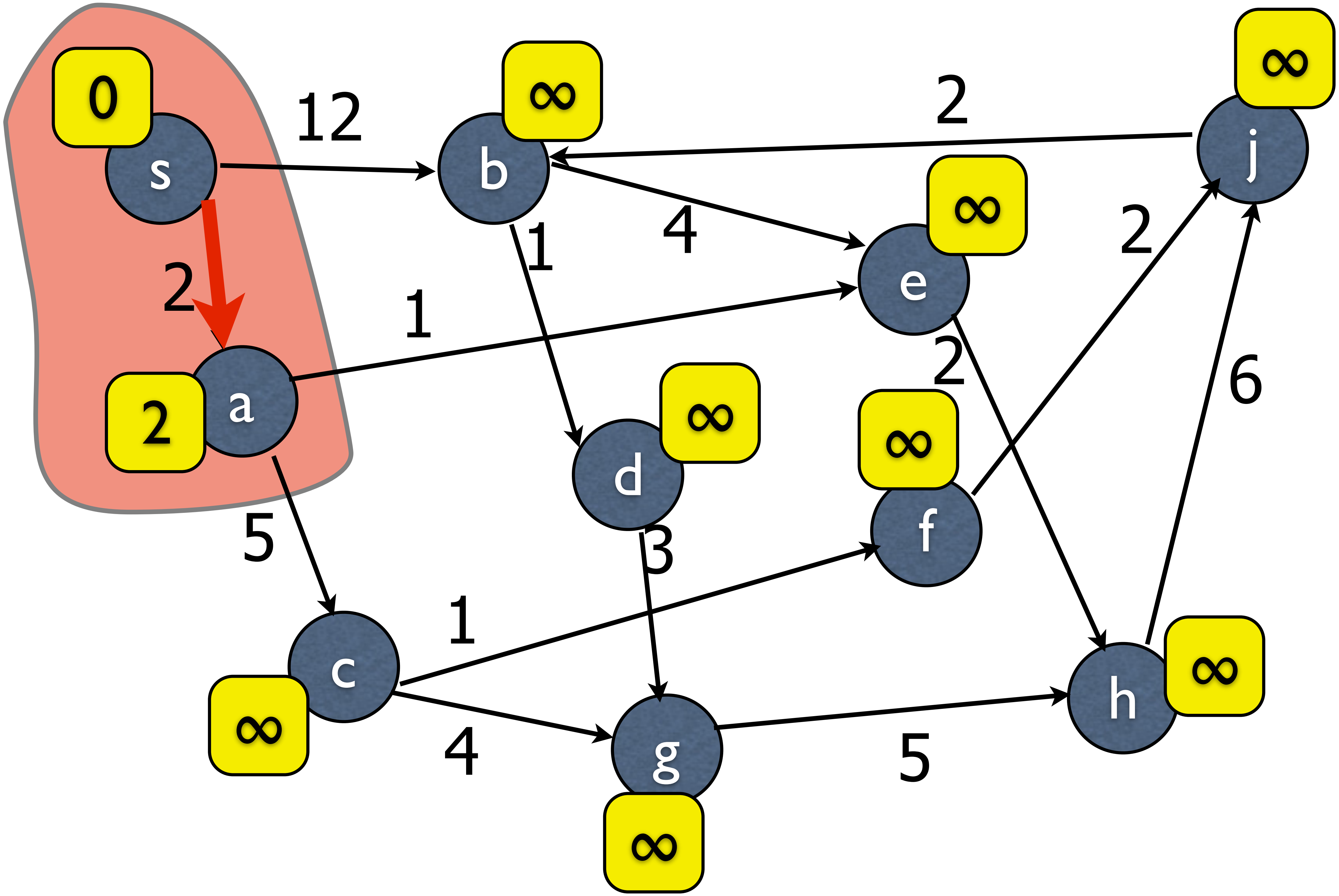


while there is edge from discovered vertex to undiscovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

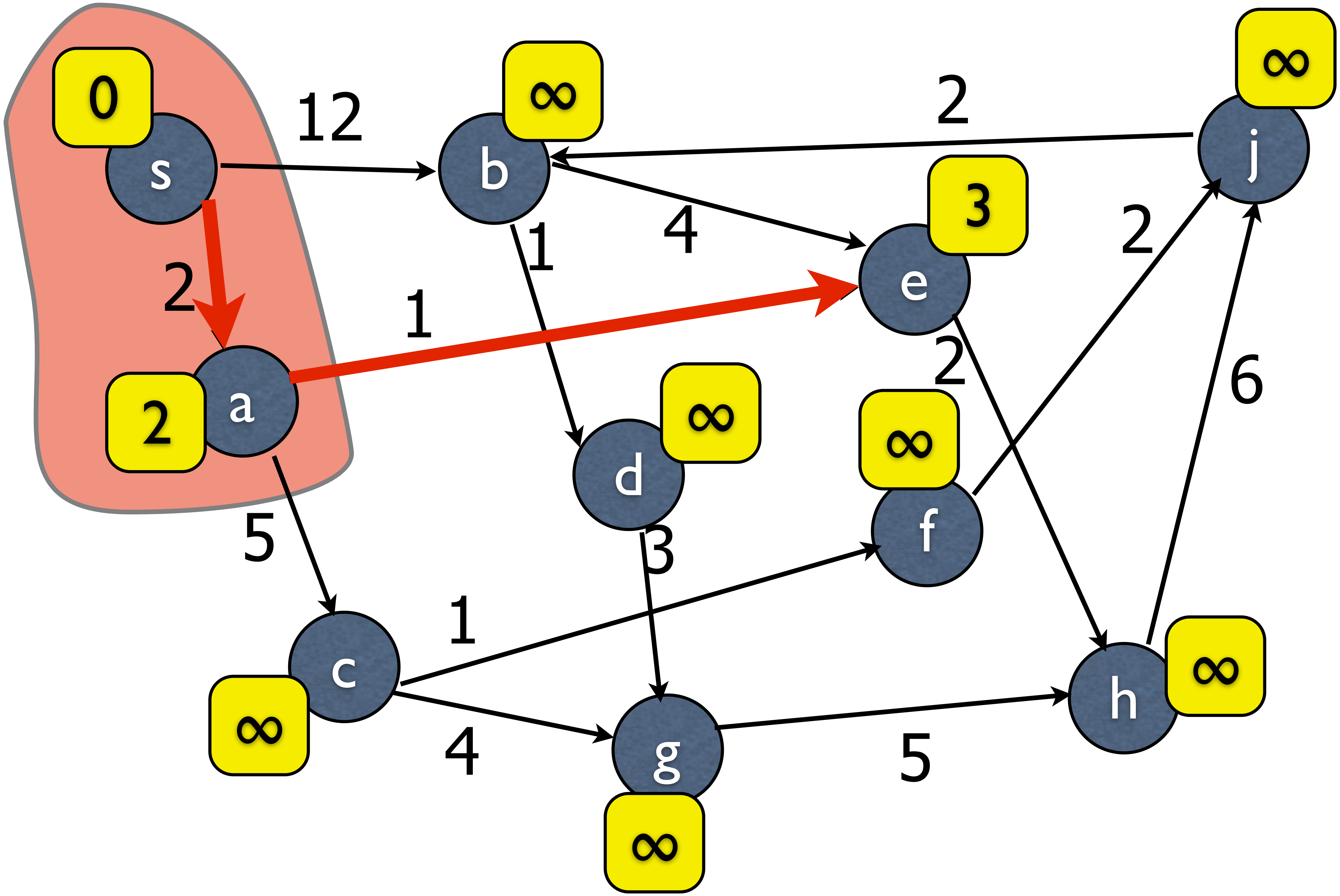
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



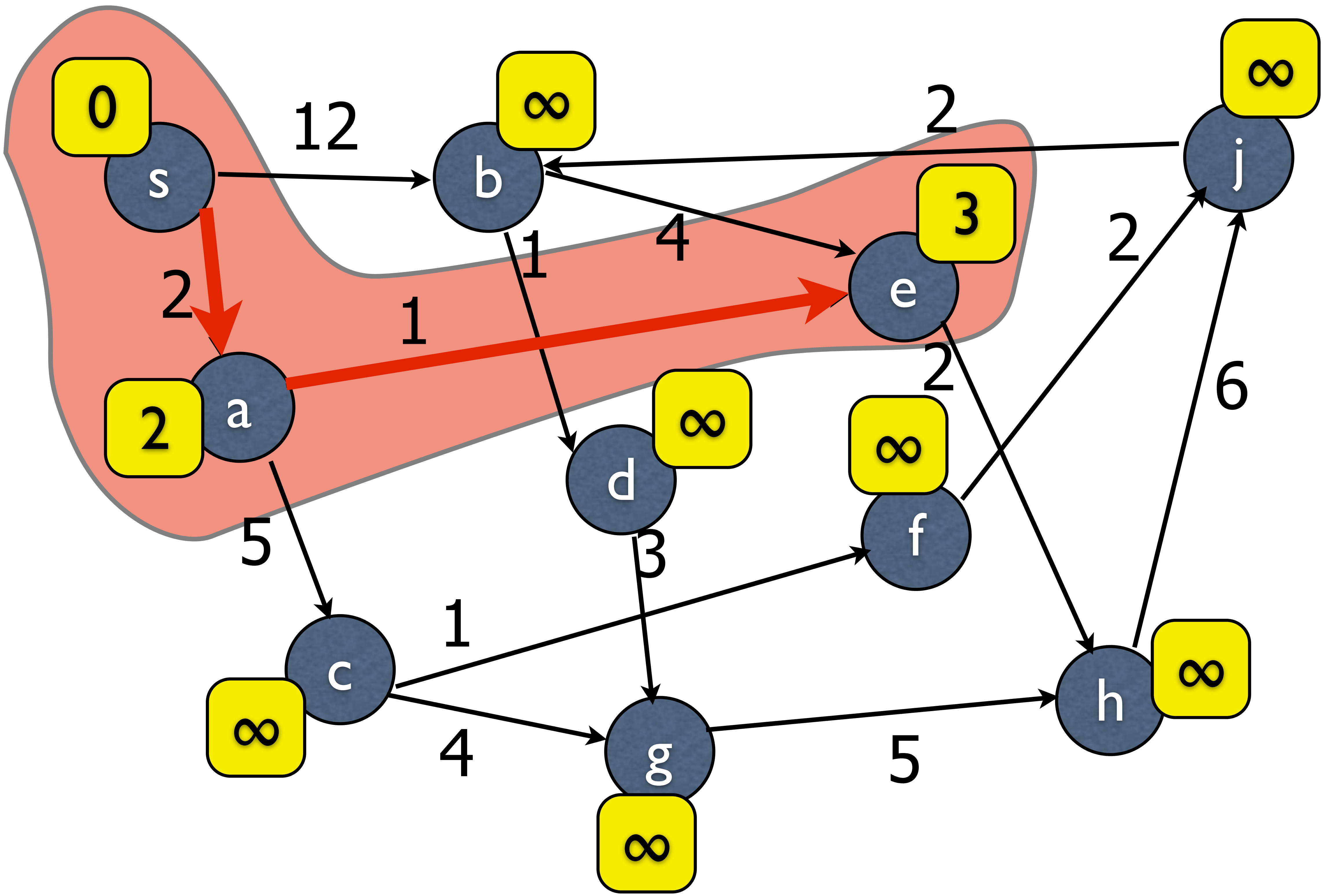
while there is edge from discovered vertex to undiscovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



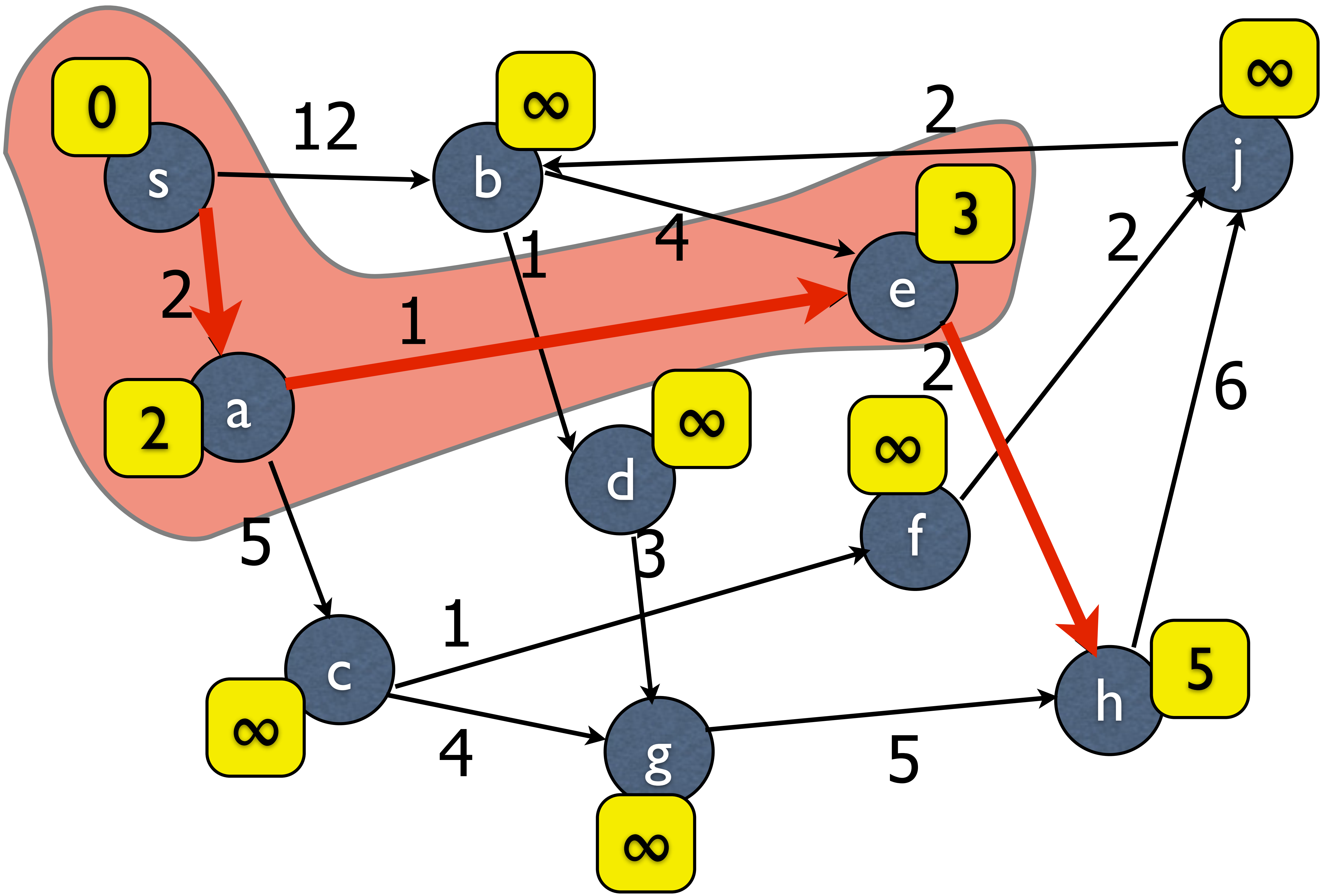
while there is edge from discovered vertex to undiscovered vertex,
  let (u,v) be such edge minimizing $d(u)+l_{u,v}$
  set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



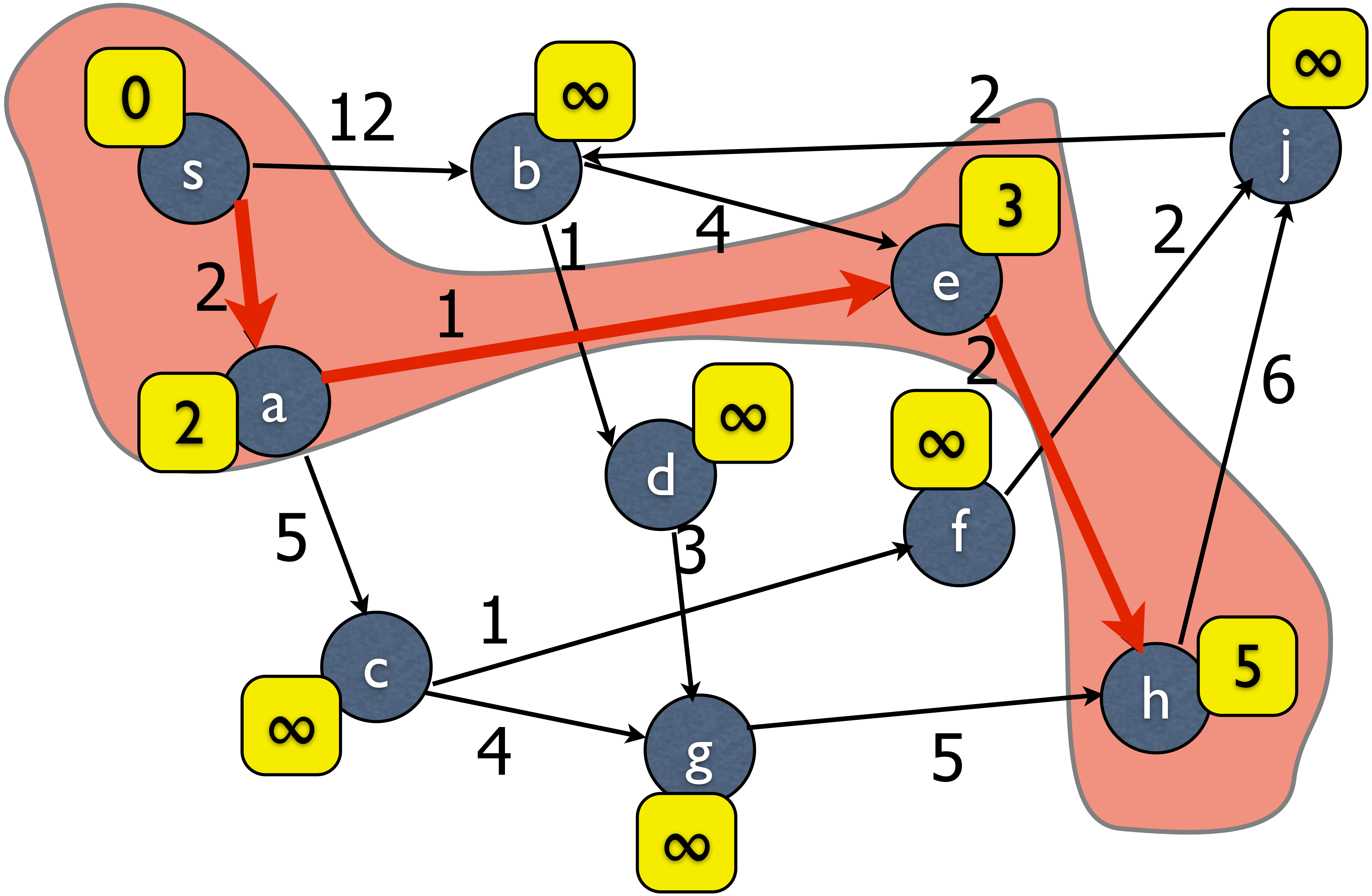
while there is edge from discovered vertex to undiscovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



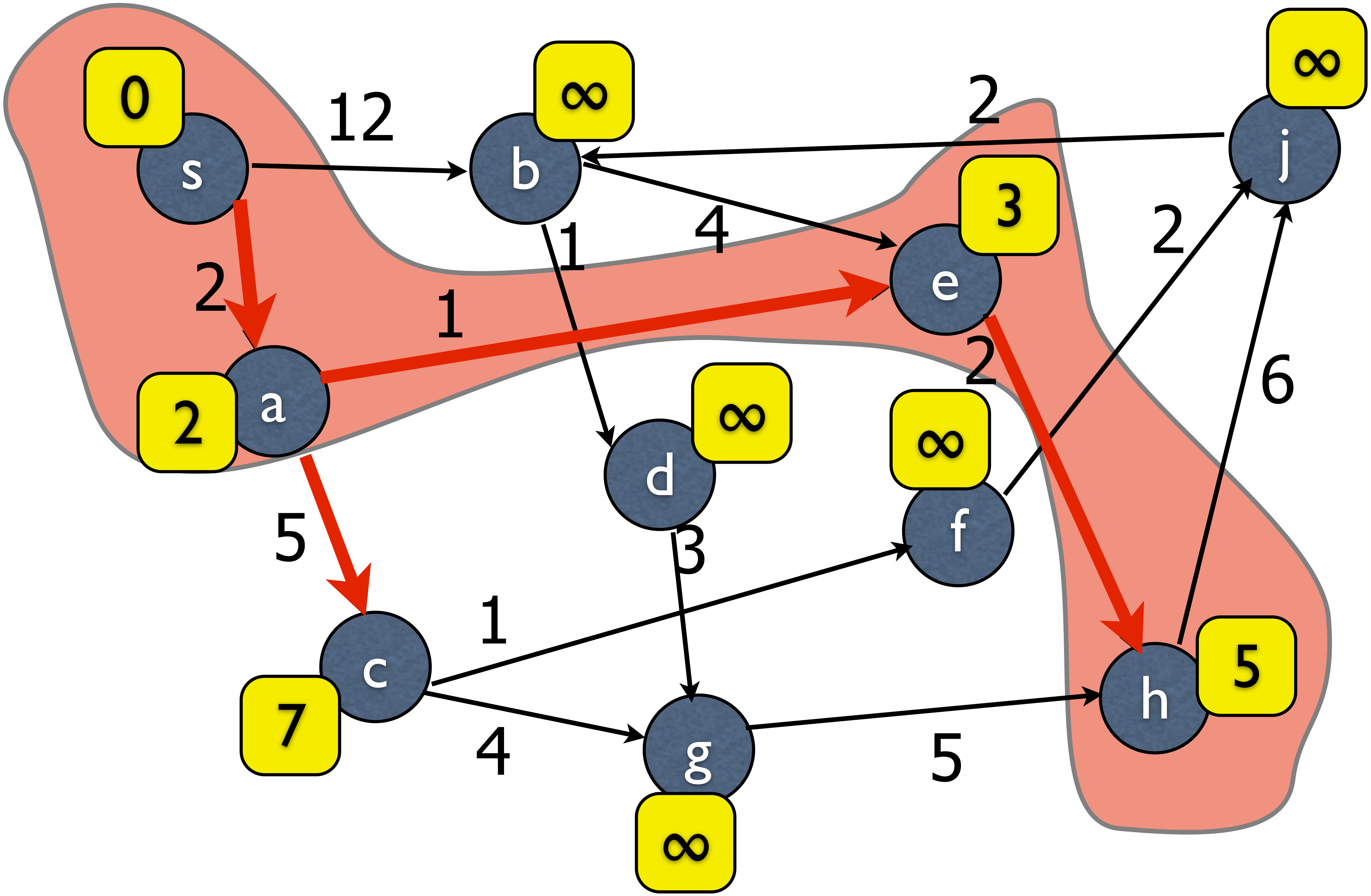
while there is edge from discovered vertex to undiscovered vertex,
    let (u,v) be such edge minimizing $d(u)+l_{u,v}$
    set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



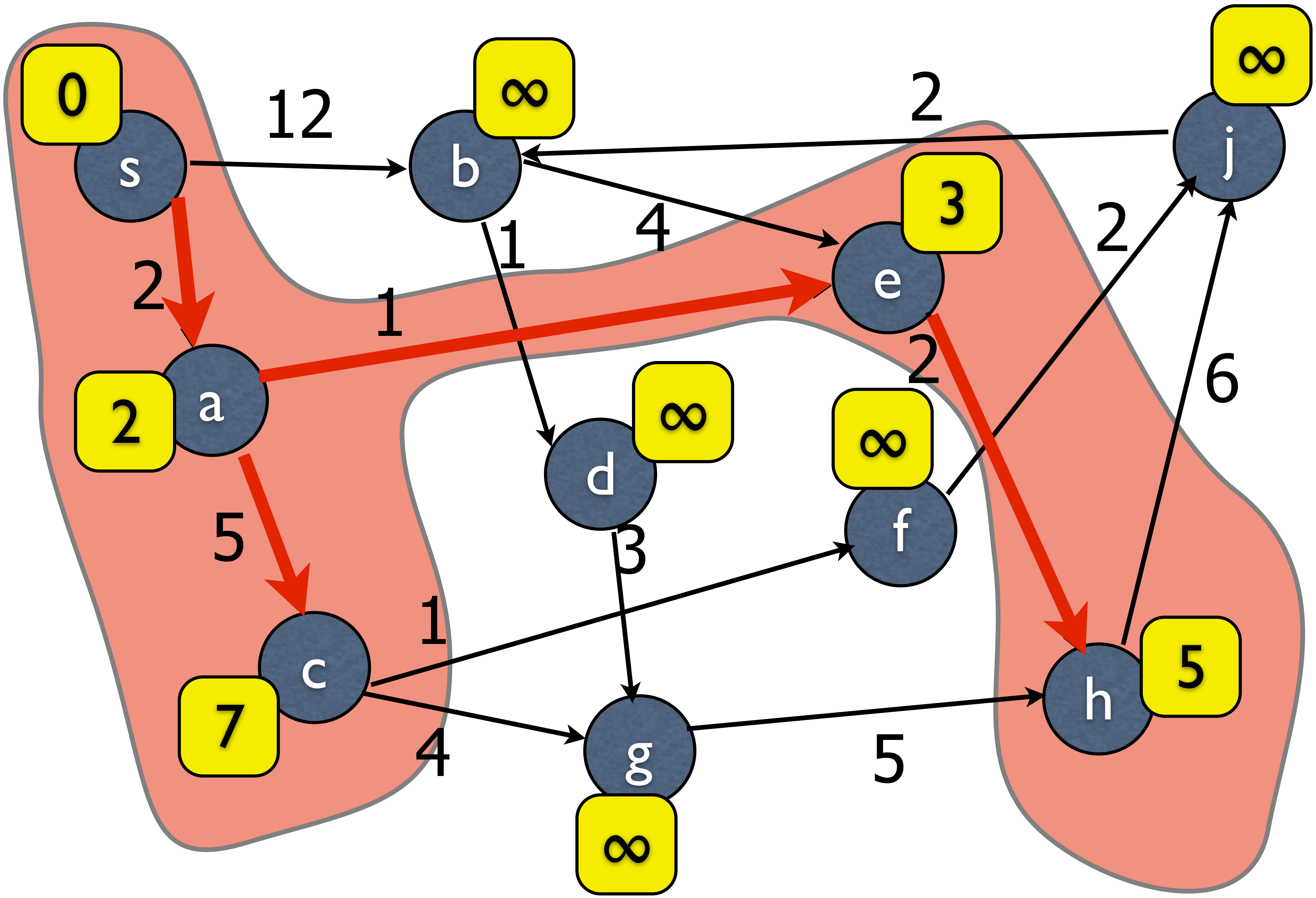
while there is edge from discovered vertex to undiscovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



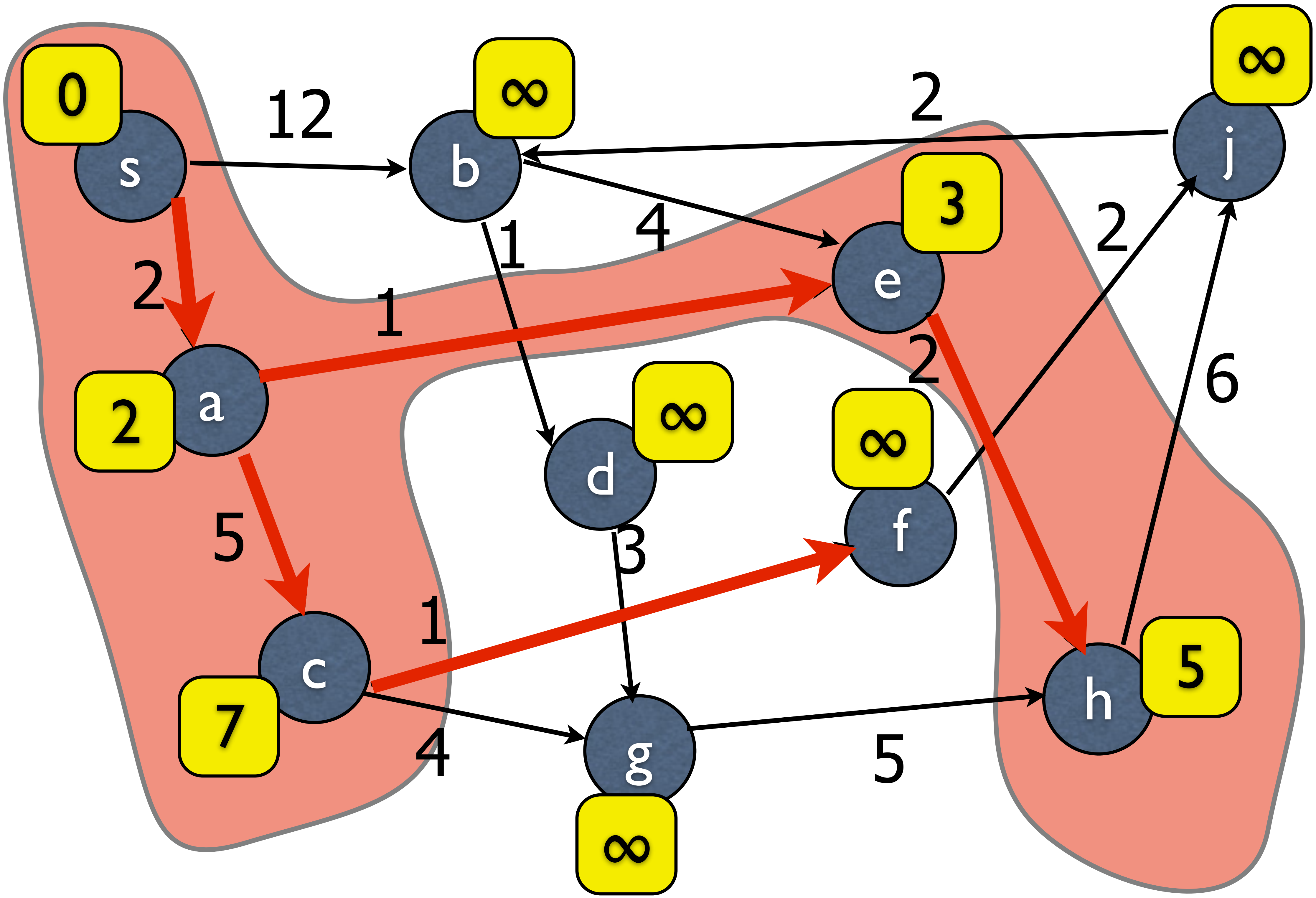
while there is edge from discovered vertex to undiscovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



while there is edge from discovered vertex to undiscovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

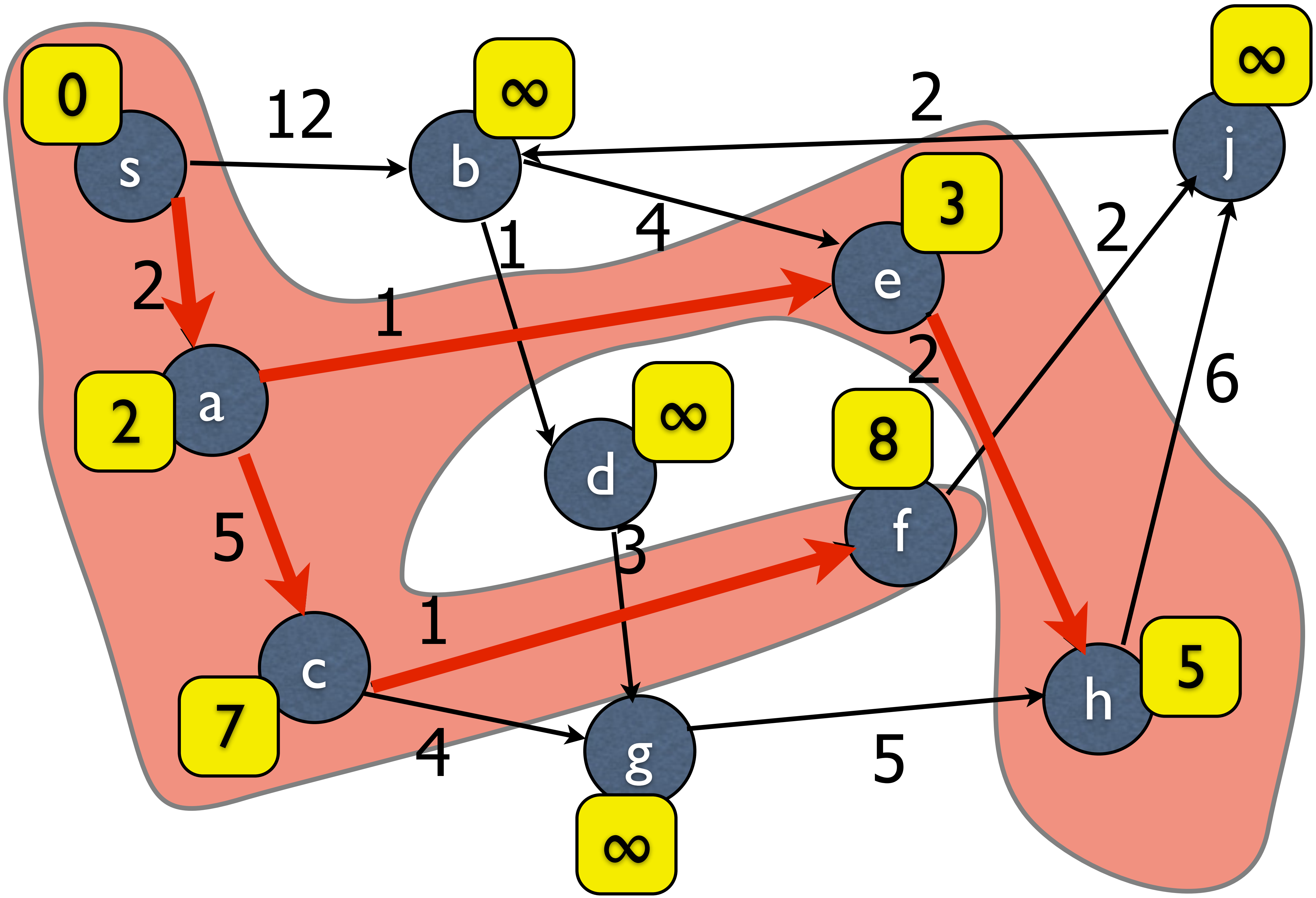


while there is edge from discovered vertex to undiscovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

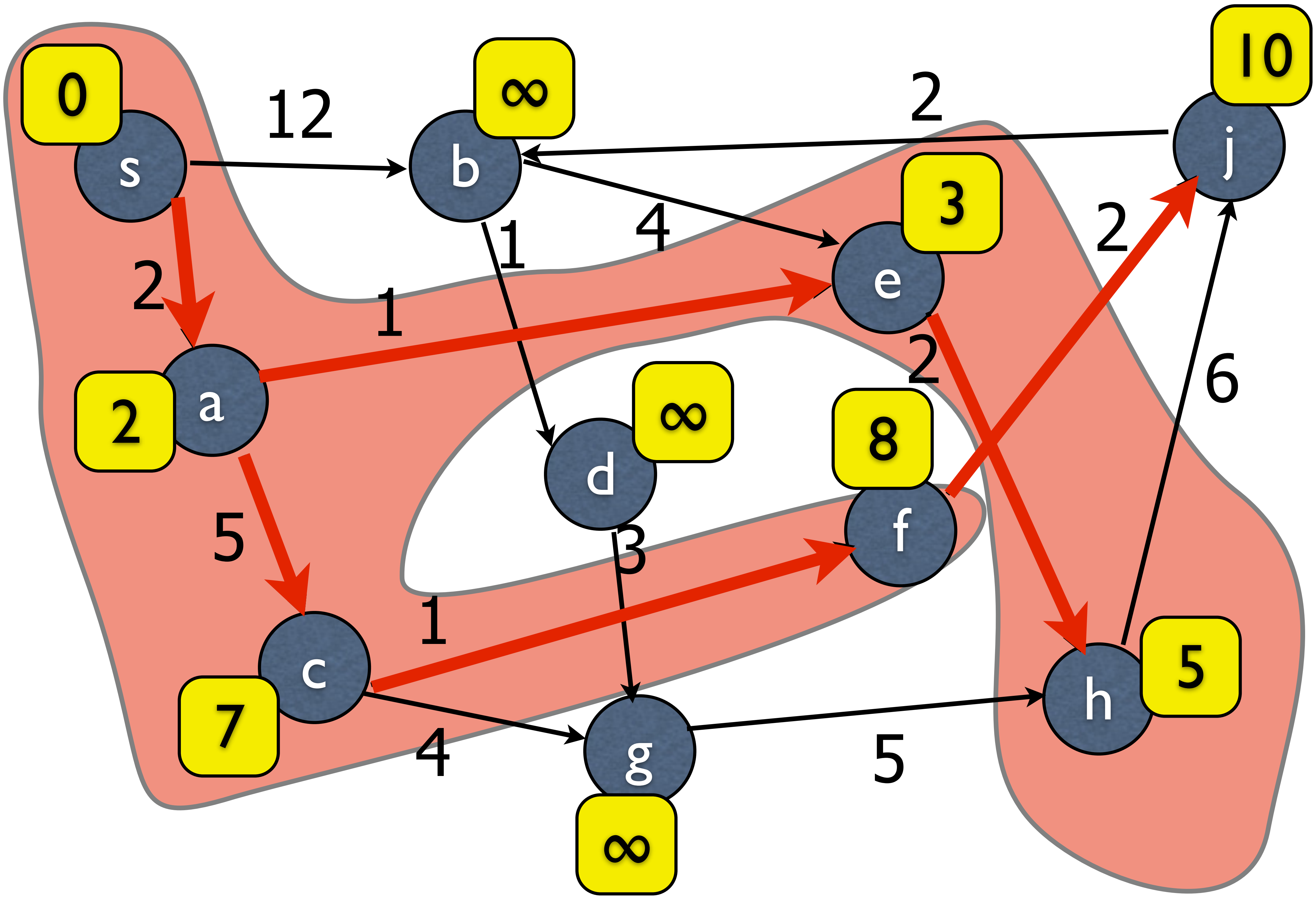
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



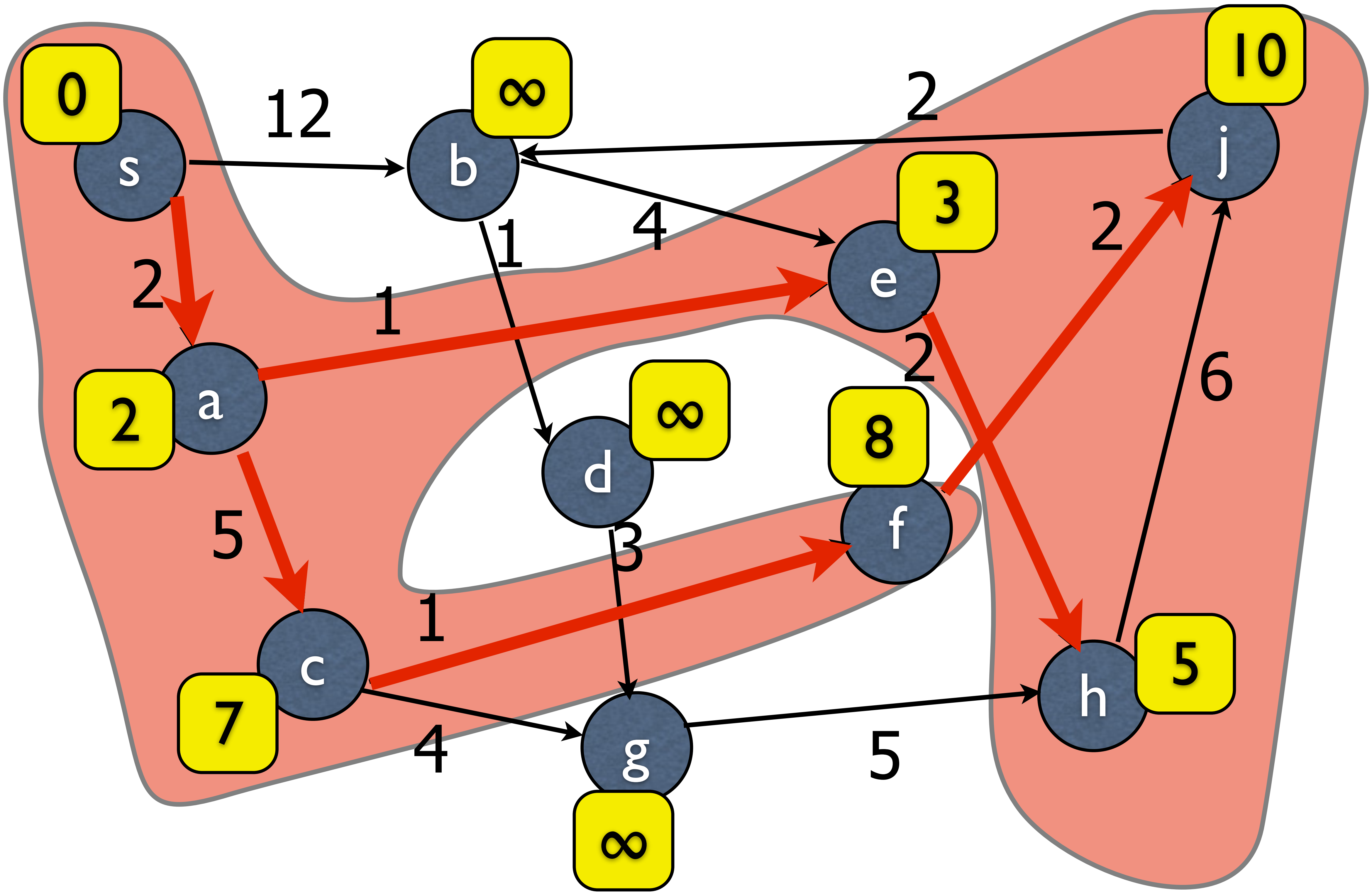
while there is edge from discovered vertex to undiscovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



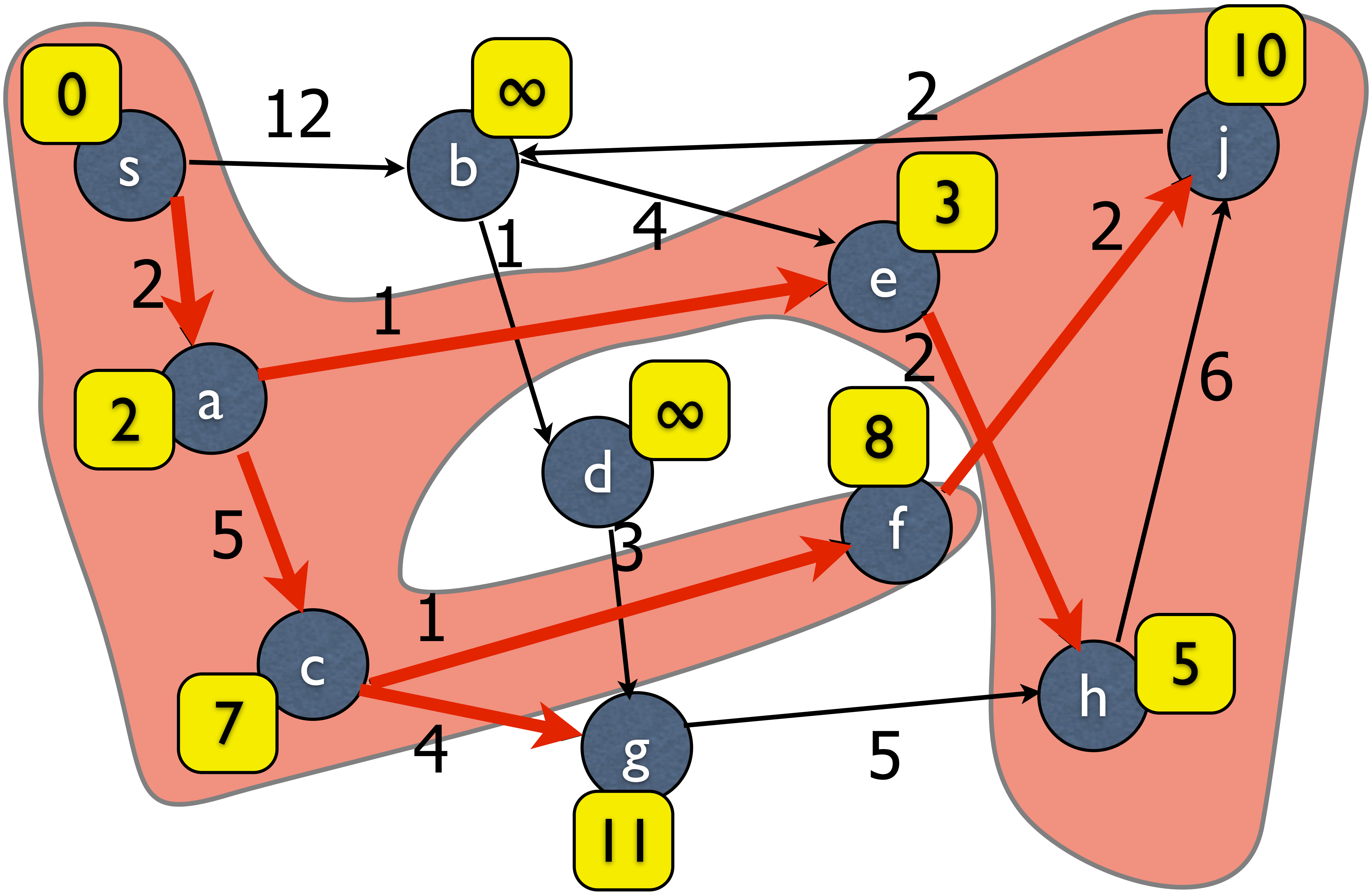
while there is edge from discovered vertex to undiscovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



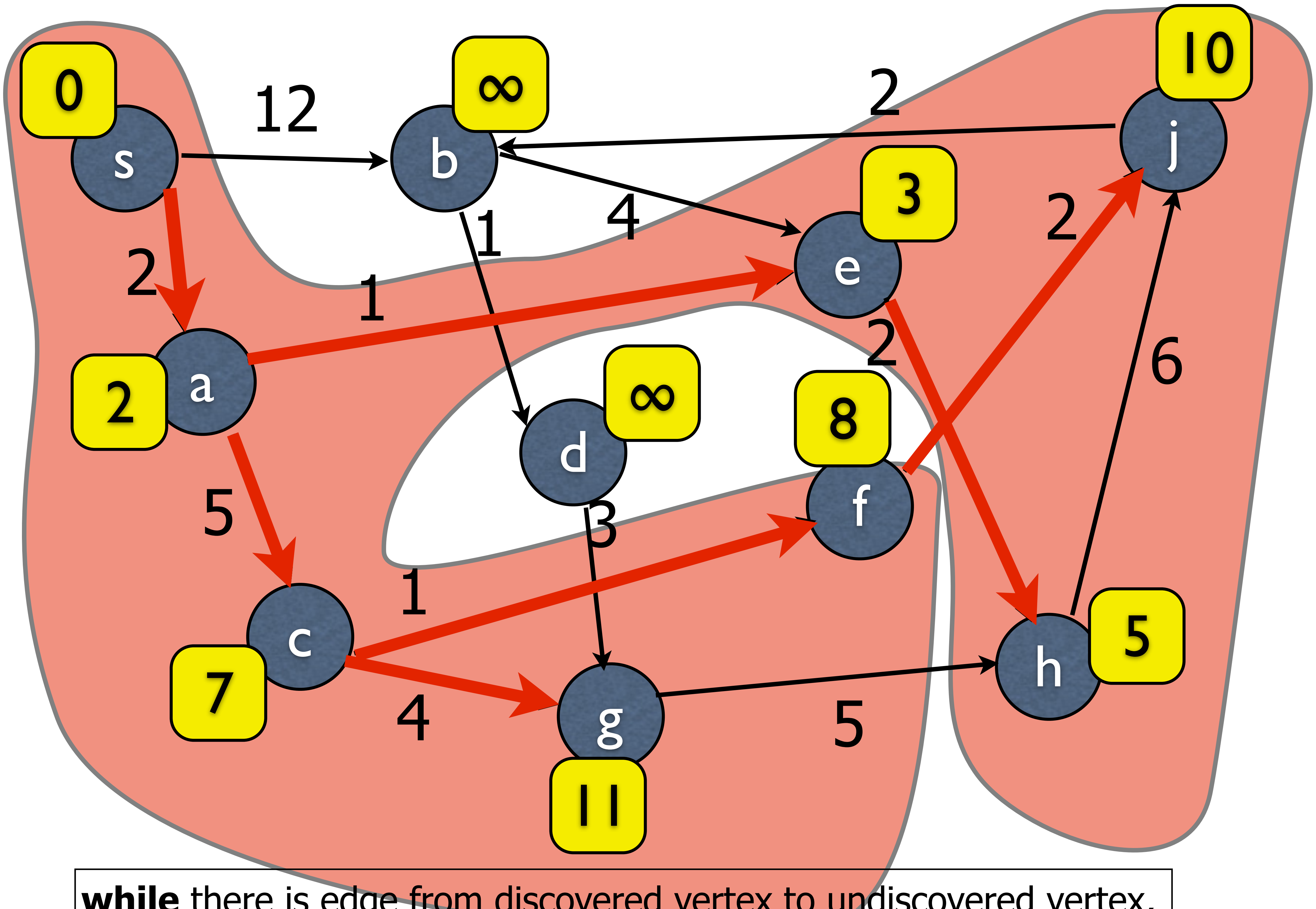
while there is edge from discovered vertex to undiscovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



while there is edge from discovered vertex to undiscovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

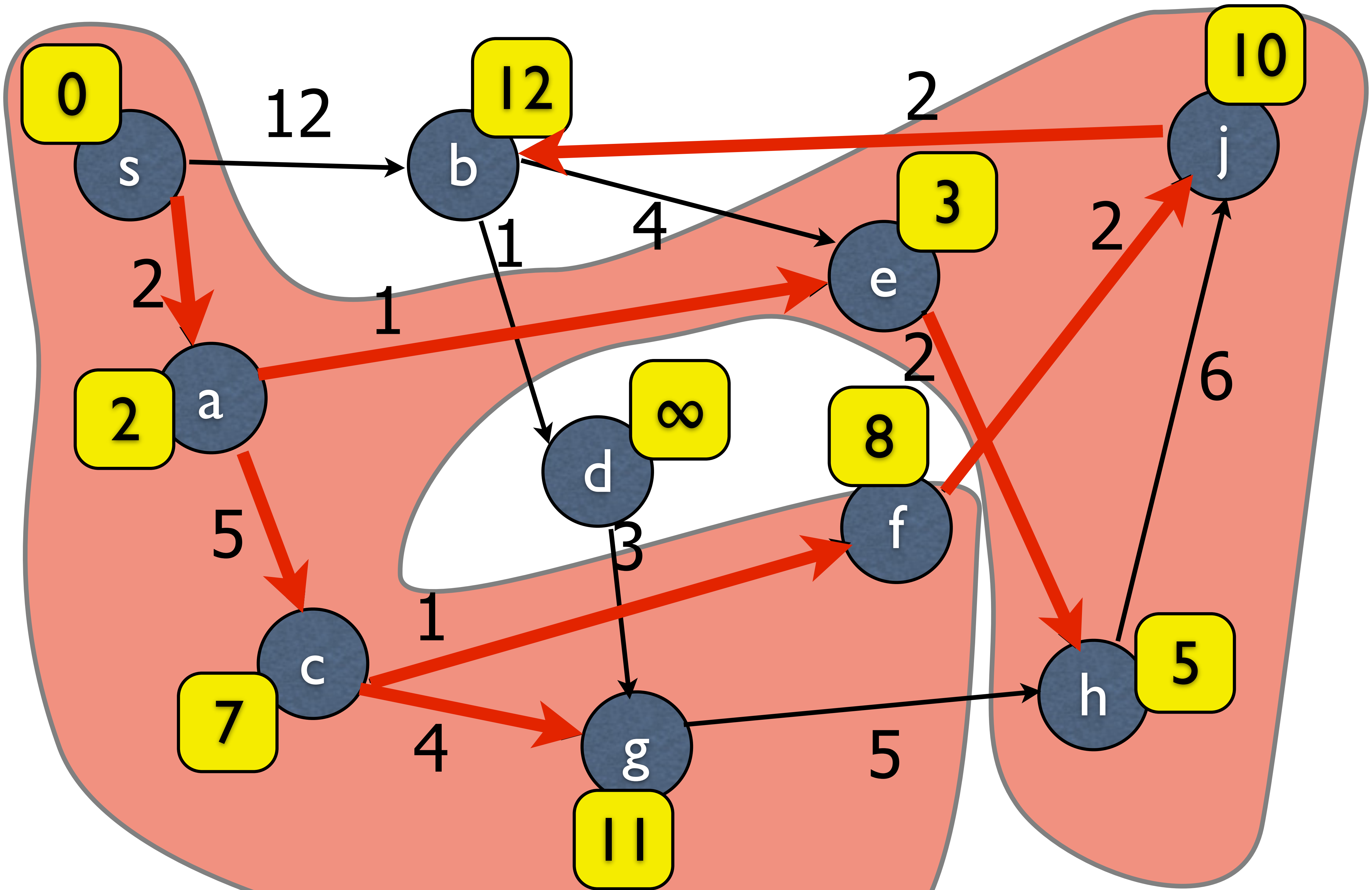


while there is edge from discovered vertex to undiscovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

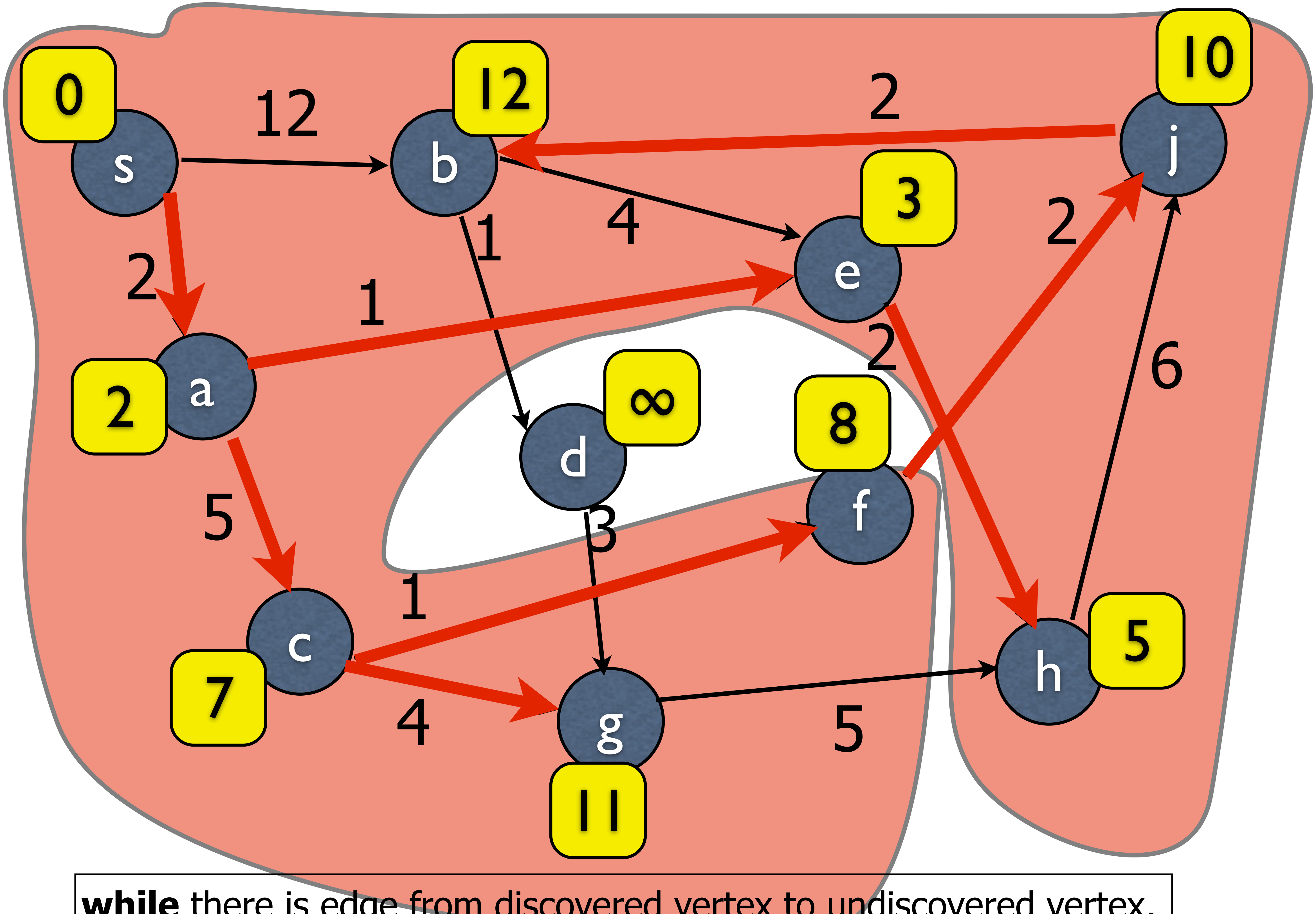
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



while there is edge from discovered vertex to undiscovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

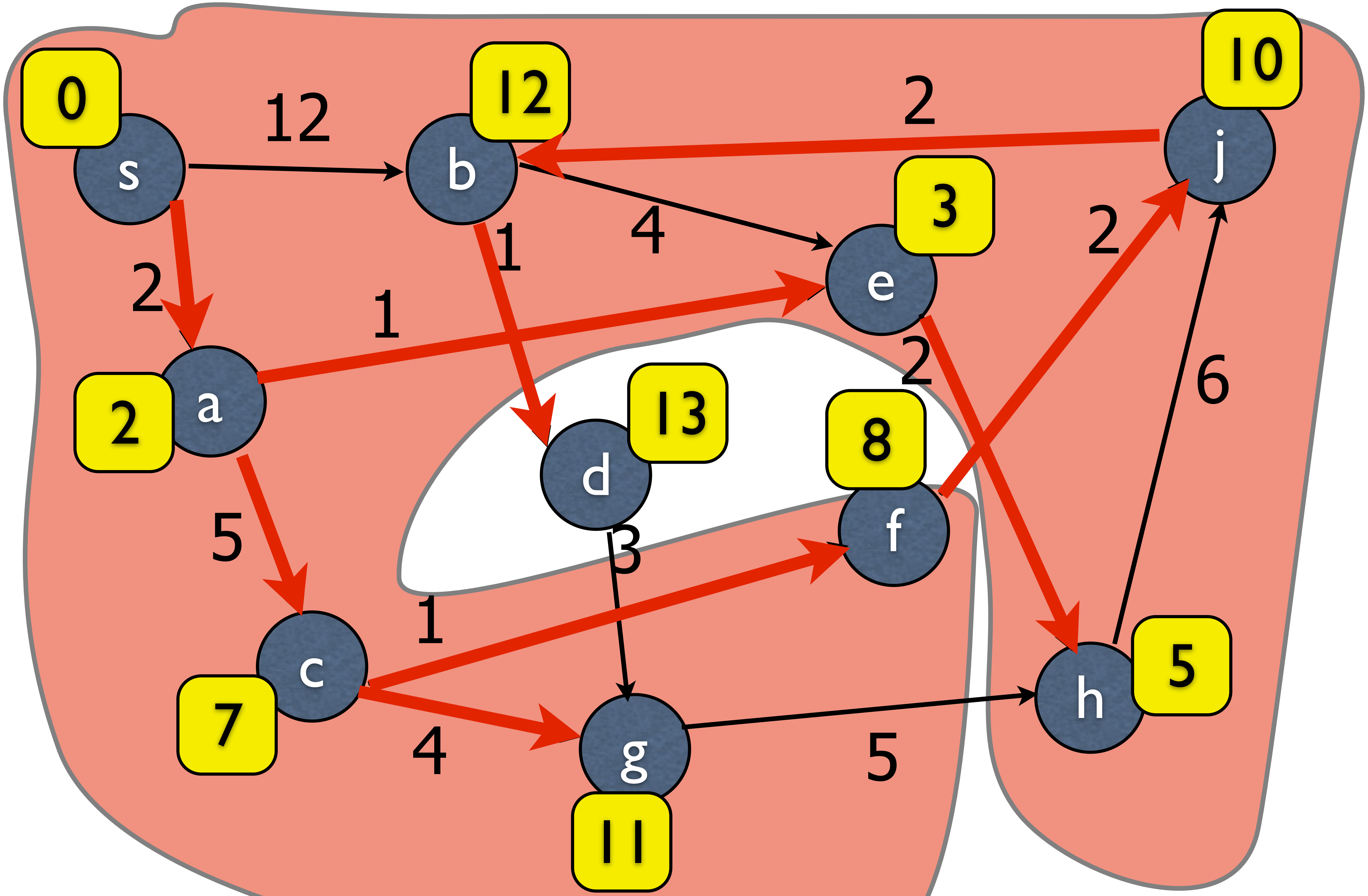


while there is edge from discovered vertex to undiscovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

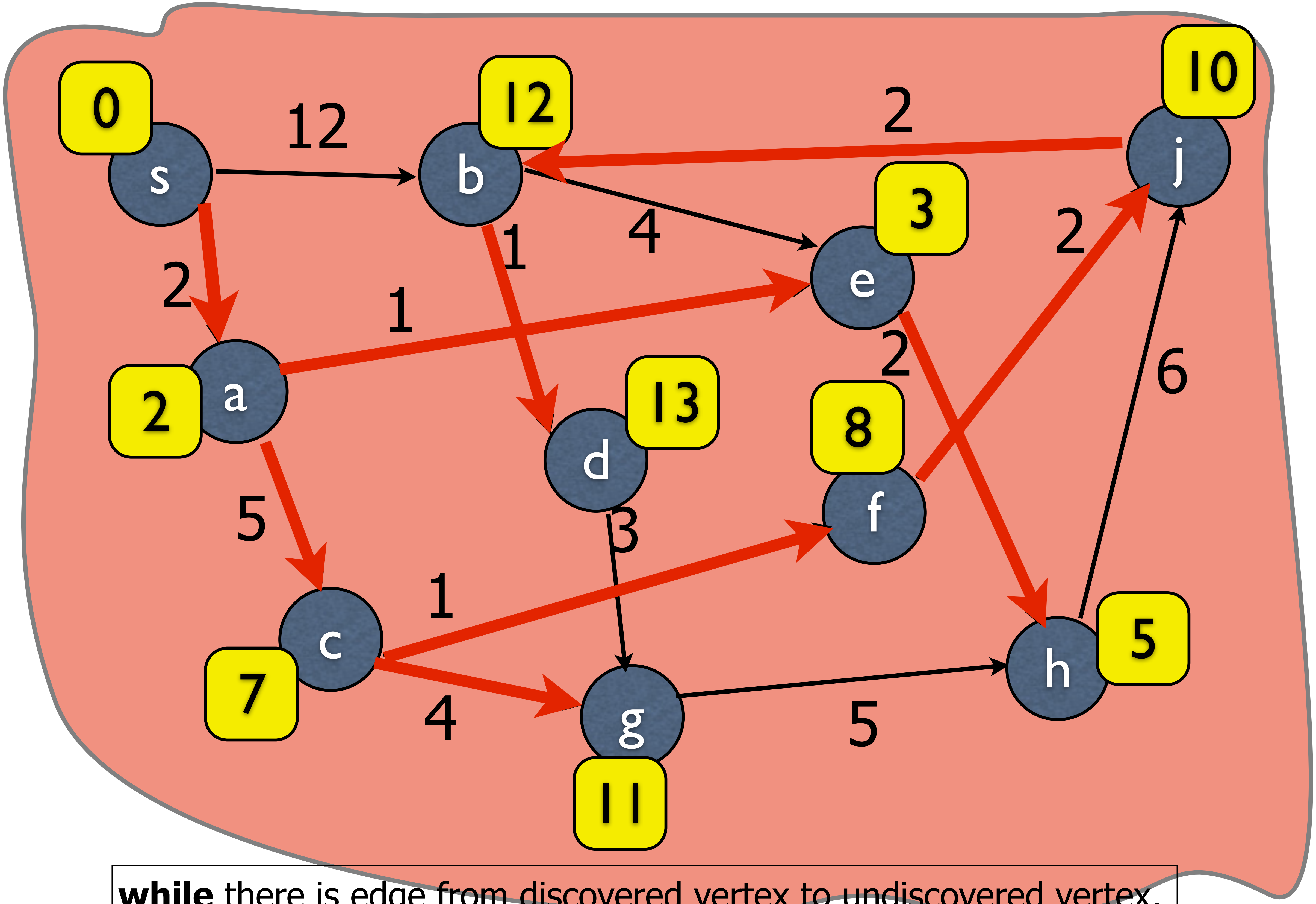
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



while there is edge from discovered vertex to undiscovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



while there is edge from discovered vertex to undiscovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Disjkstra(s)
Set all vertices v undiscovered, $d(v) = \infty$
Set $d(s) = 0$, mark s discovered.
while there is edge from undiscovered vertex to discovered vertex,
  let (u,v) be such edge minimizing $d(u)+l_{u,v}$
  set $d(v) = d(u) + l_{u,v}$, mark v discovered

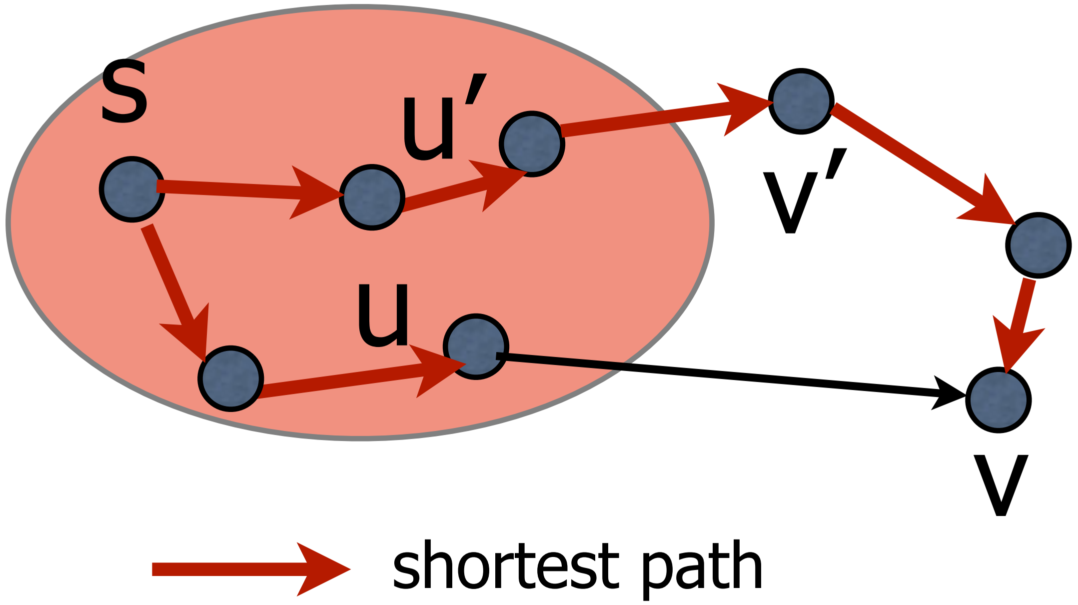
Correctness analysis:

Prove that if v is discovered d(v) is distance of v from s.

Initially this is true, since $d(s)=0$, and s is only discovered vertex.

Let v be next discovered vertex, using edge (u,v). $d(v) = d(u) + l_{u,v}$. Then distance of v from s is at most d(v) since d(u) is correct.

If distance v from s is < d(v), must be v′ s.t. $d(u') + l_{u',v'} < d(u) + l_{u,v}$. This contradicts algorithm, v′ would be chosen instead of v.

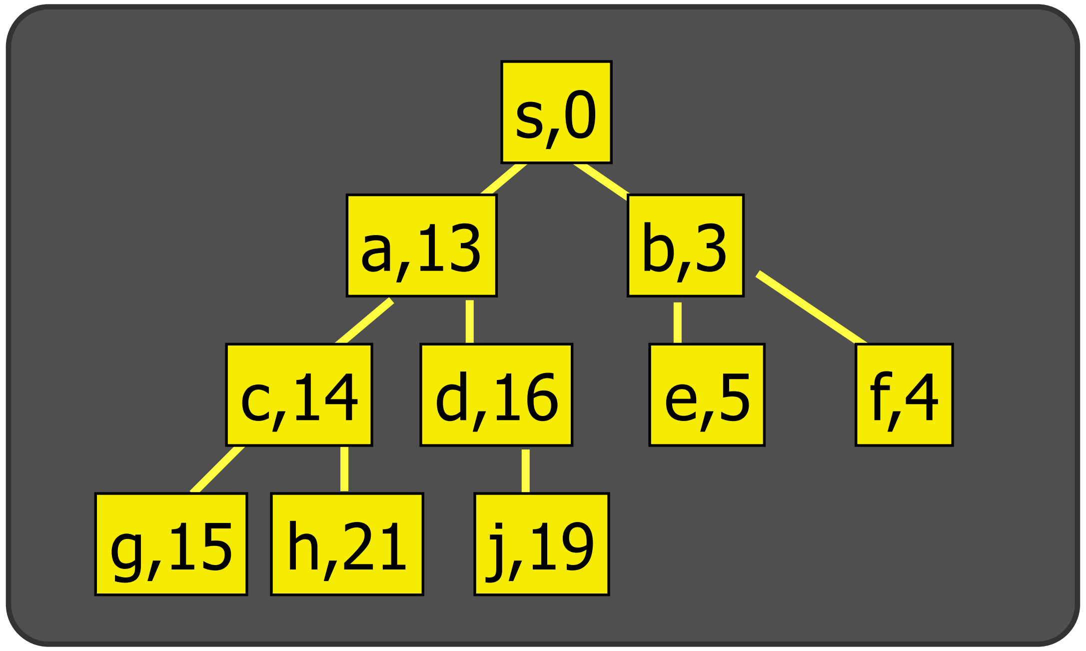


<u>Disjkstra(s)</u>
Set all vertices v undiscovered, $d(v) = \infty$
Set $d(s) = 0$, mark s discovered.
while there is edge from undiscovered vertex to discovered vertex,
    let (u,v) be such edge minimizing $d(u)+l_{u,v}$
    set $d(v) = d(u) + l_{u,v}$, mark v discovered

Running time analysis:

O(mn).

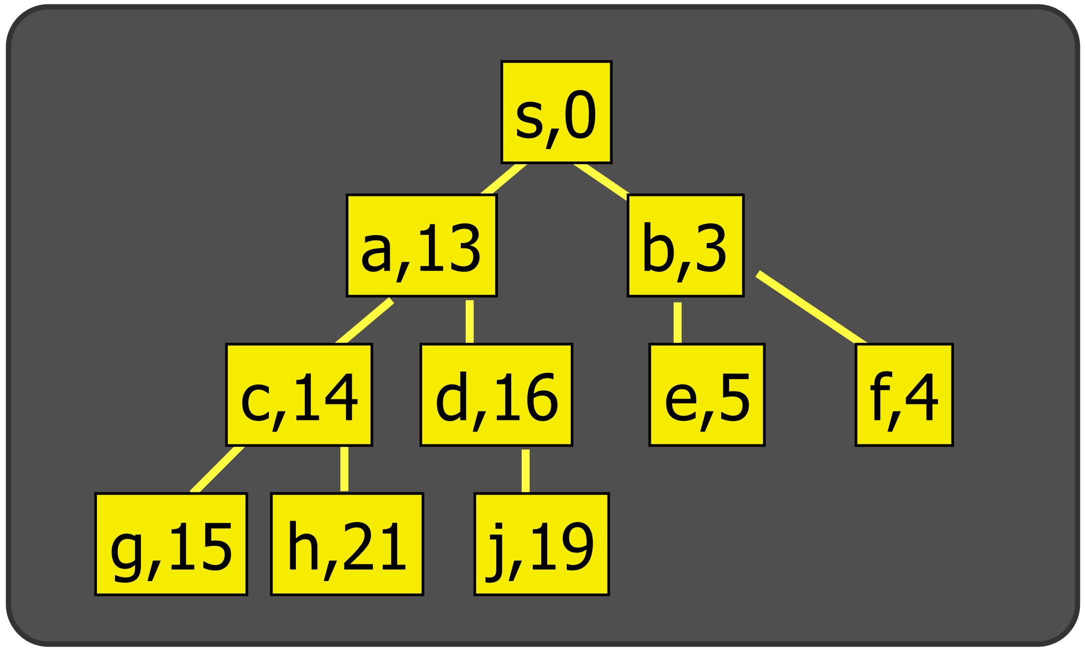
Heaps



binary tree, every vertex has value at most that of its children

Supported operations:

Heaps

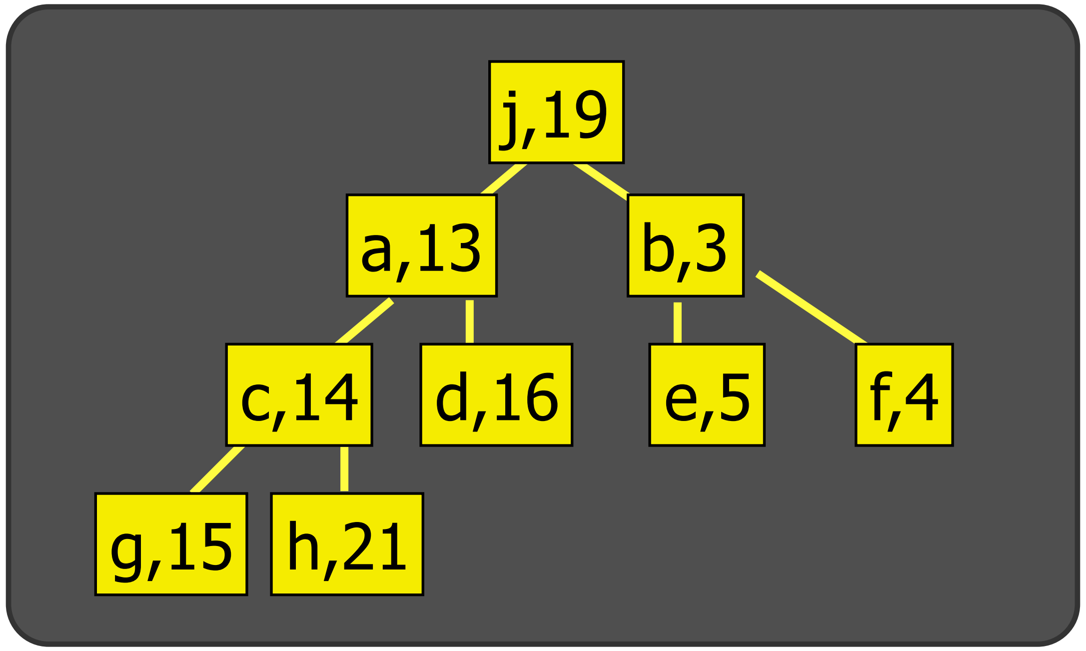


binary tree, every vertex has value at most that of its children

Supported operations:

delete min: delete root,
replace with last leaf,
swap with min-child until order restored.

Heaps

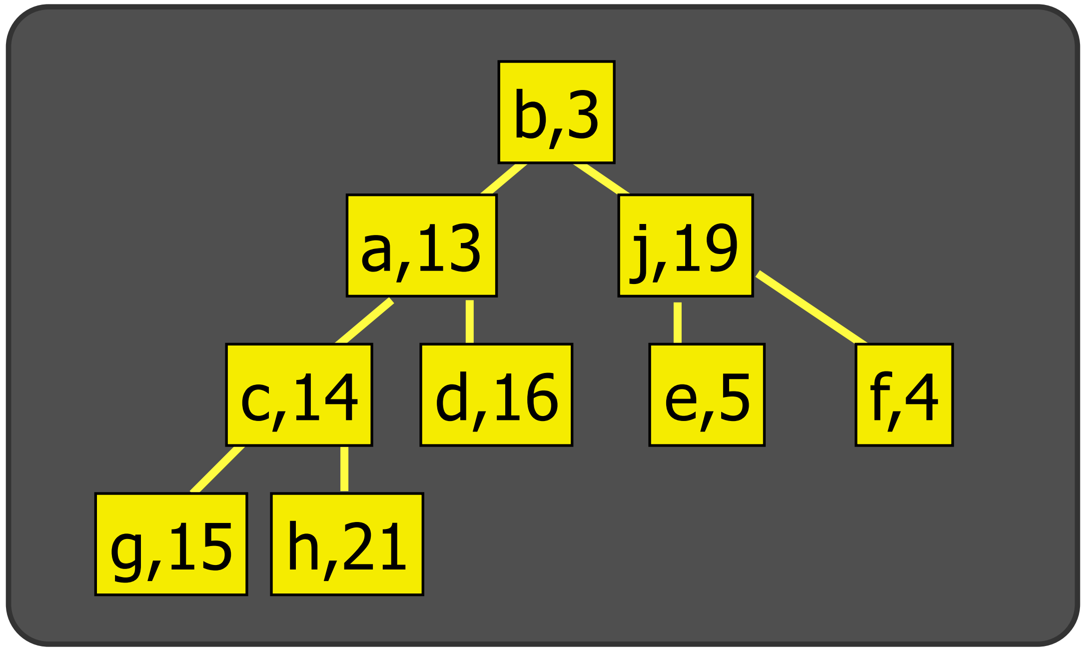


binary tree, every vertex has value at most that of its children

Supported operations:

delete min: delete root, replace with last leaf, swap with min-child until order restored.

Heaps

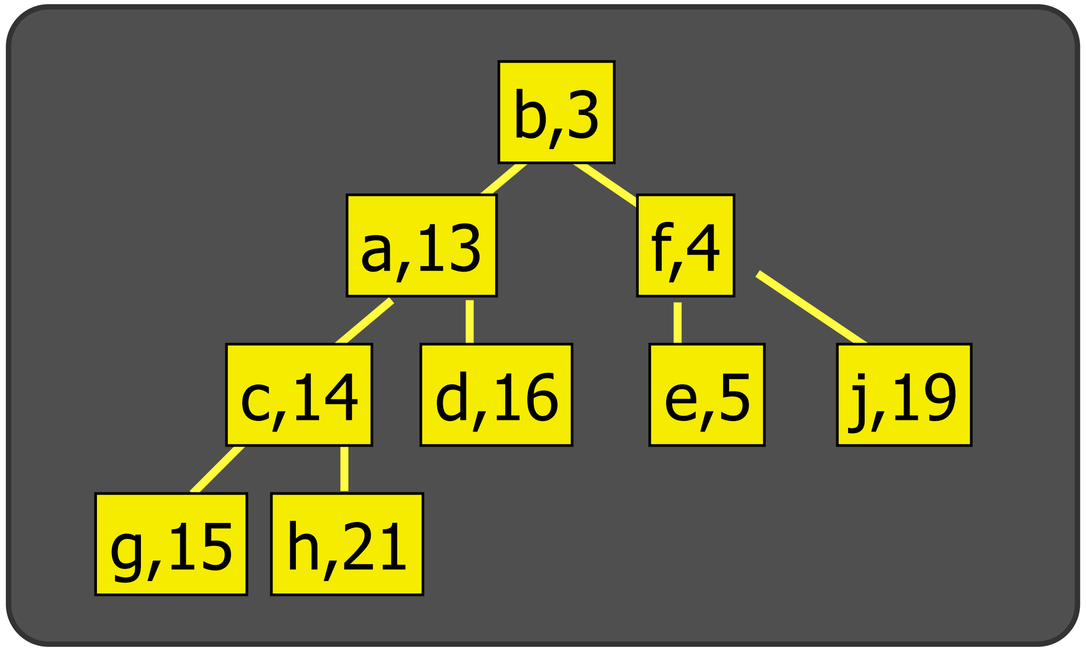


binary tree, every vertex has value at most that of its children

Supported operations:

delete min: delete root, replace with last leaf, swap with min-child until order restored.

Heaps

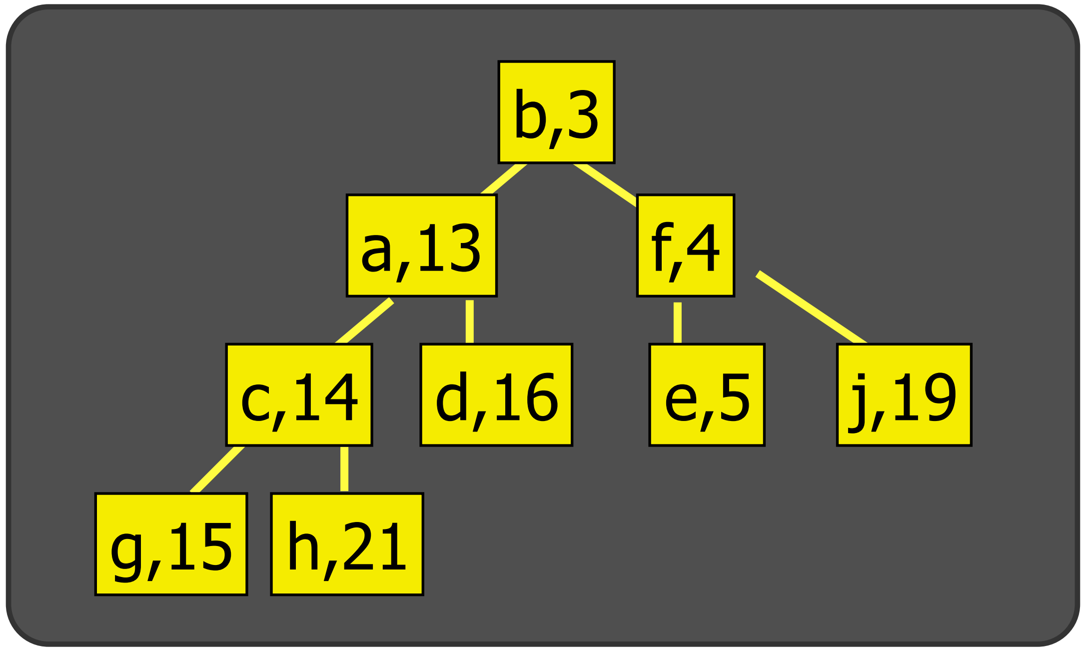


binary tree, every vertex has value at most that of its children

Supported operations:

delete min: delete root, replace with last leaf, swap with min-child until order restored.

Heaps



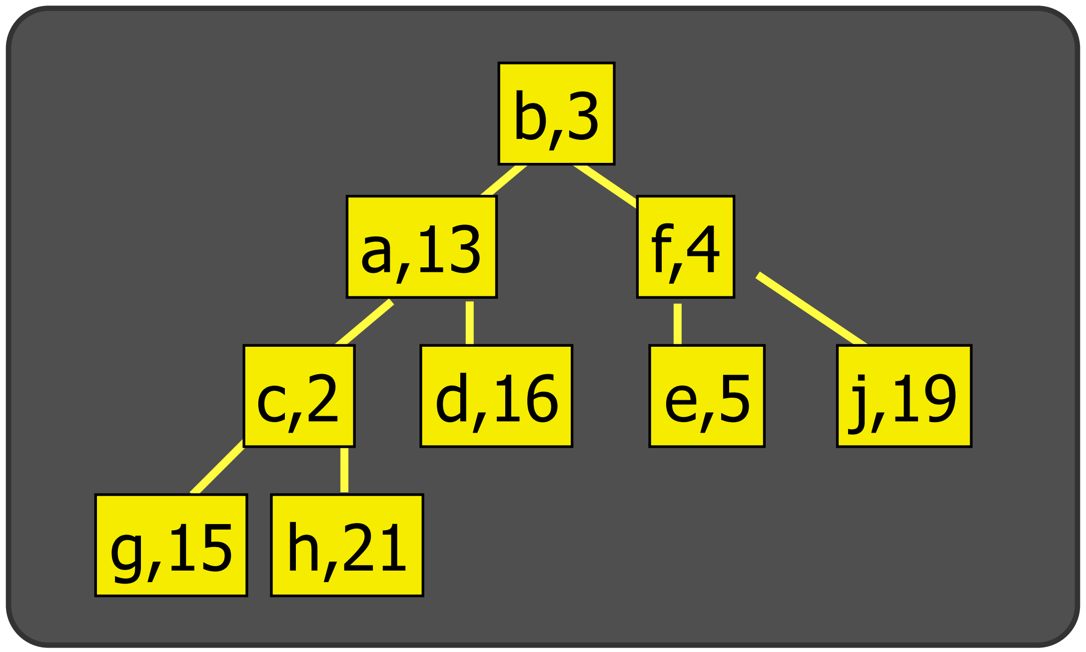
binary tree, every vertex has value at most that of its children

Supported operations:

delete min: delete root, replace with last leaf, swap with min-child until order restored.

reduce value of node: bubble up value until order restored

Heaps



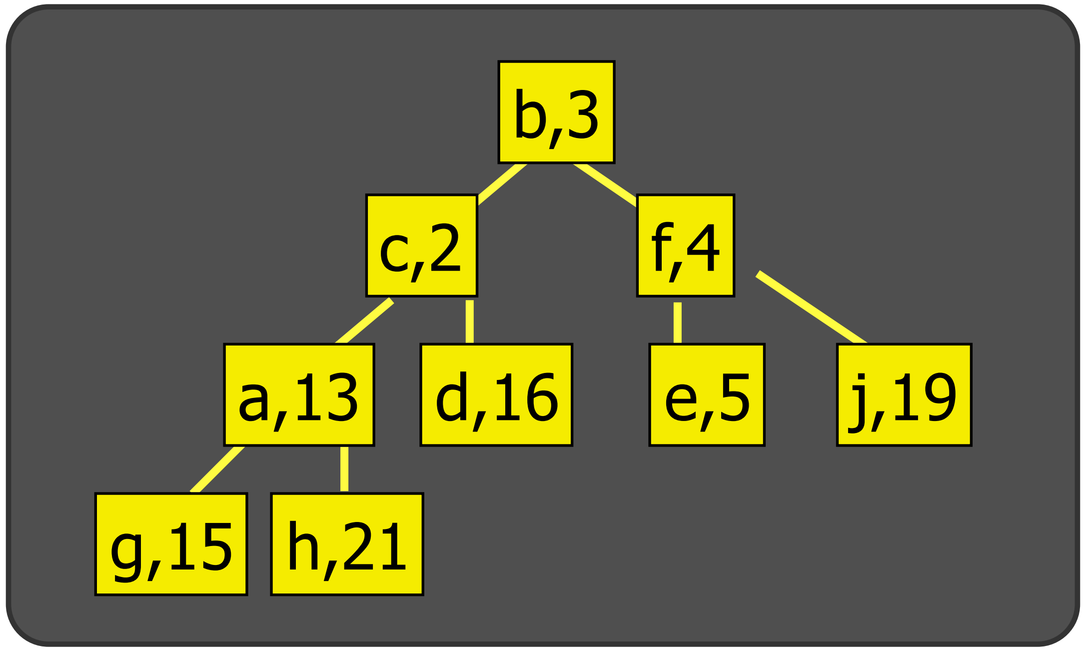
binary tree, every vertex has value at most that of its children

<u>Supported operations:</u>

delete min: delete root, replace with last leaf, swap with min-child until order restored.

reduce value of node: bubble up value until order restored

Heaps



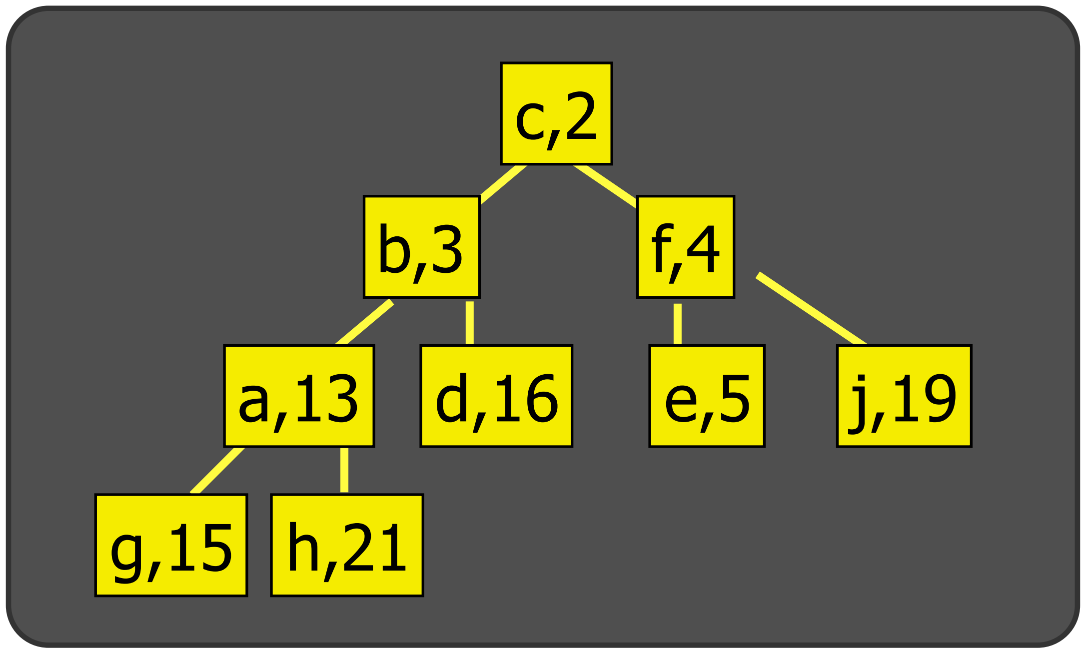
binary tree, every vertex has value at most that of its children

Supported operations:

delete min: delete root, replace with last leaf, swap with min-child until order restored.

reduce value of node: bubble up value until order restored

Heaps



binary tree, every vertex has value at most that of its children

<u>Supported operations:</u>

delete min: delete root, replace with last leaf, swap with min-child until order restored.

reduce value of node: bubble up value until order restored

all operations take O(log n) time

```
Disjkstra(s)
Set all vertices v undiscovered, d(v)= ∞
Set d(s) = 0, mark s discovered.
while there is edge from undiscovered vertex to discovered vertex,
    let (u,v) be such edge minimizing d(u)+l_{u,v}
    set d(v) = d(u) + l_{u,v}, mark v discovered
```

Running time analysis:

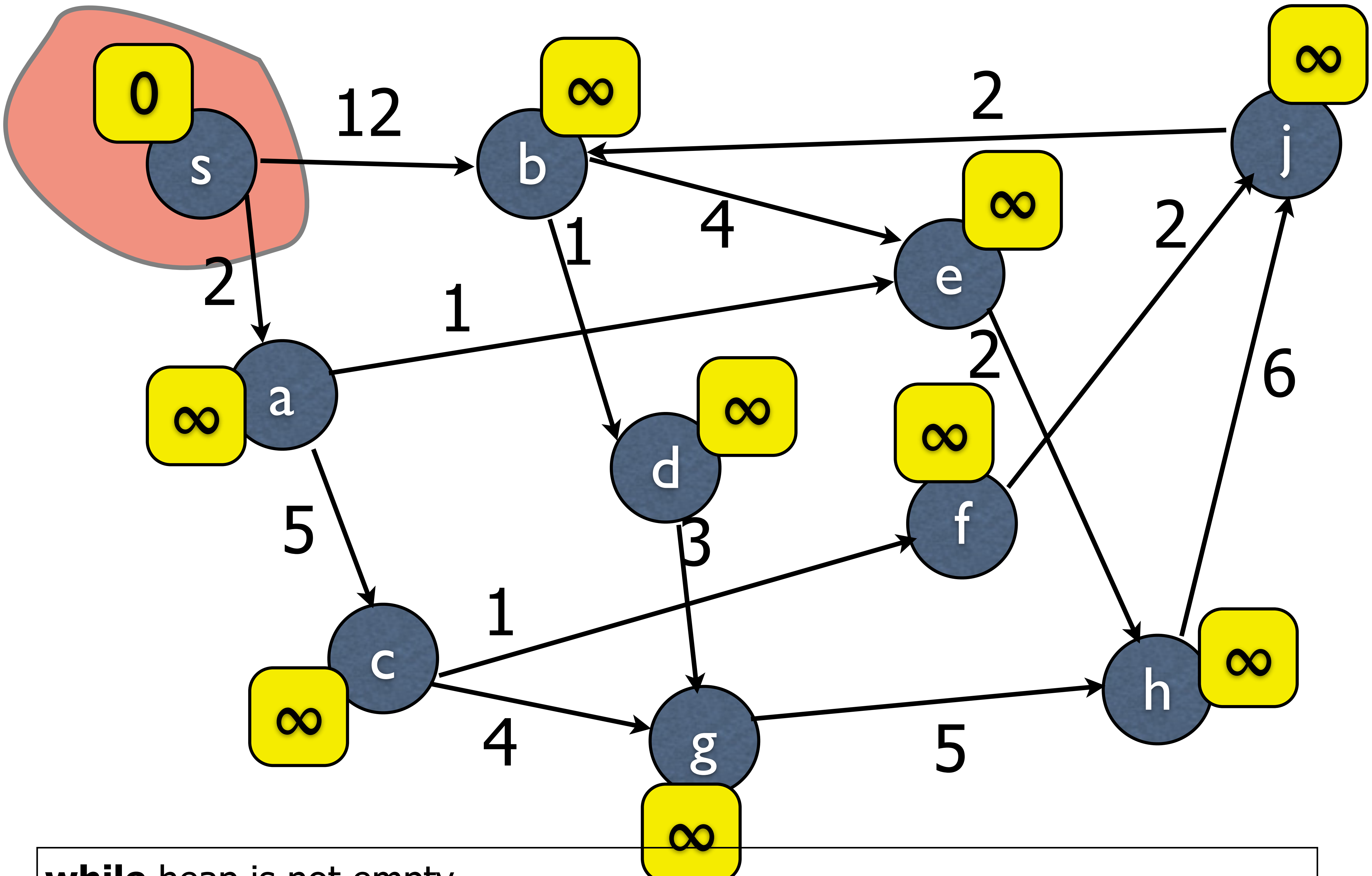
O(mn).

```
Disjkstra(s)
Set all vertices v undiscovered, d(v)= ∞
Set d(s) = 0, mark s discovered. Make heap.
while heap is not empty,
    delete u with minimum d(u) value from heap
    for each edge (u,v)
        if d(v) > d(u) + l_{u,v}, update d(v) = d(u) + l_{u,v}.
```

Running time analysis:

O((m+n) log n).

Dijkstra's Algorithm



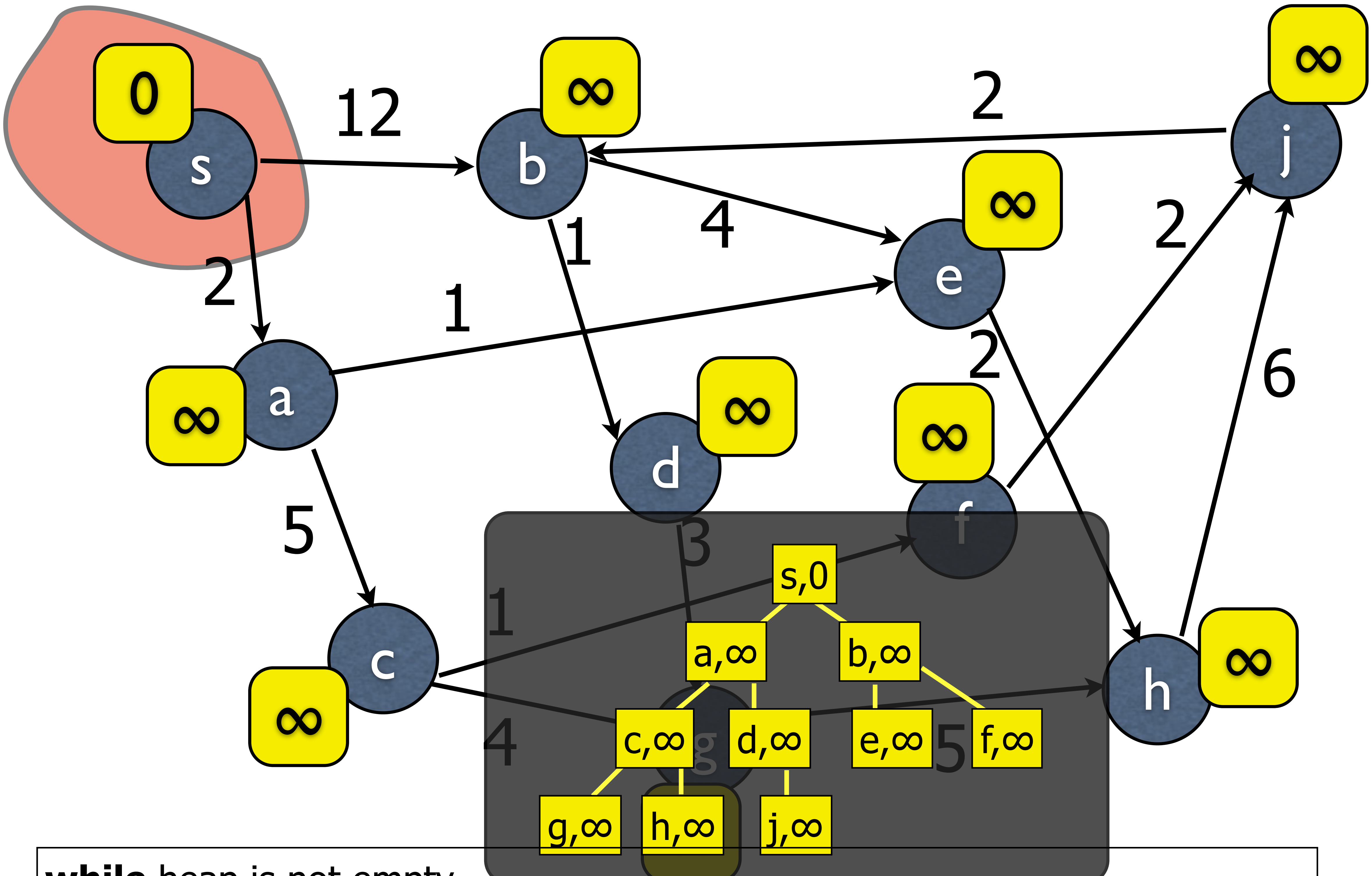
while heap is not empty,

delete u with minimum d(u) value from heap

for each edge (u,v)

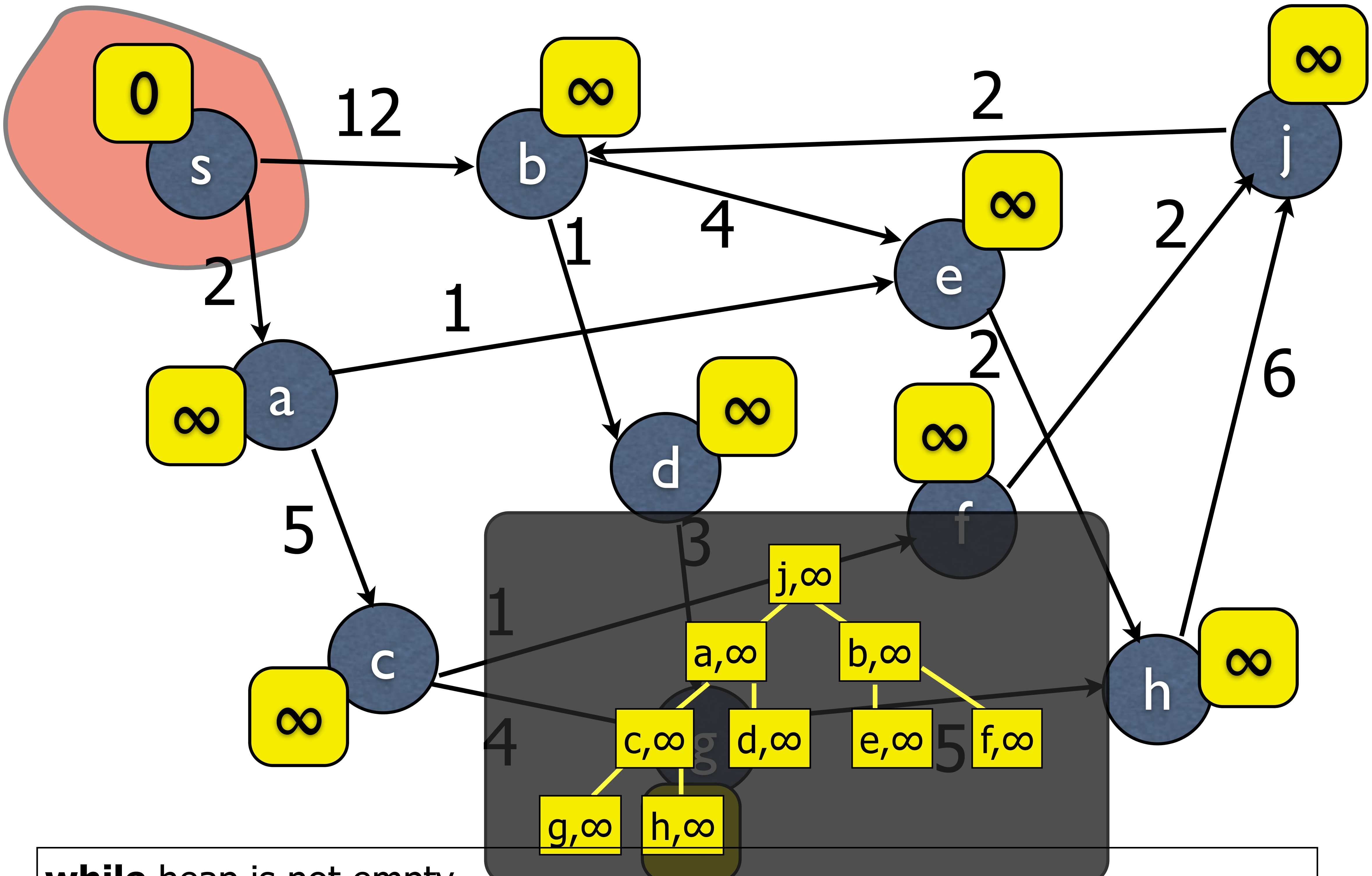
if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



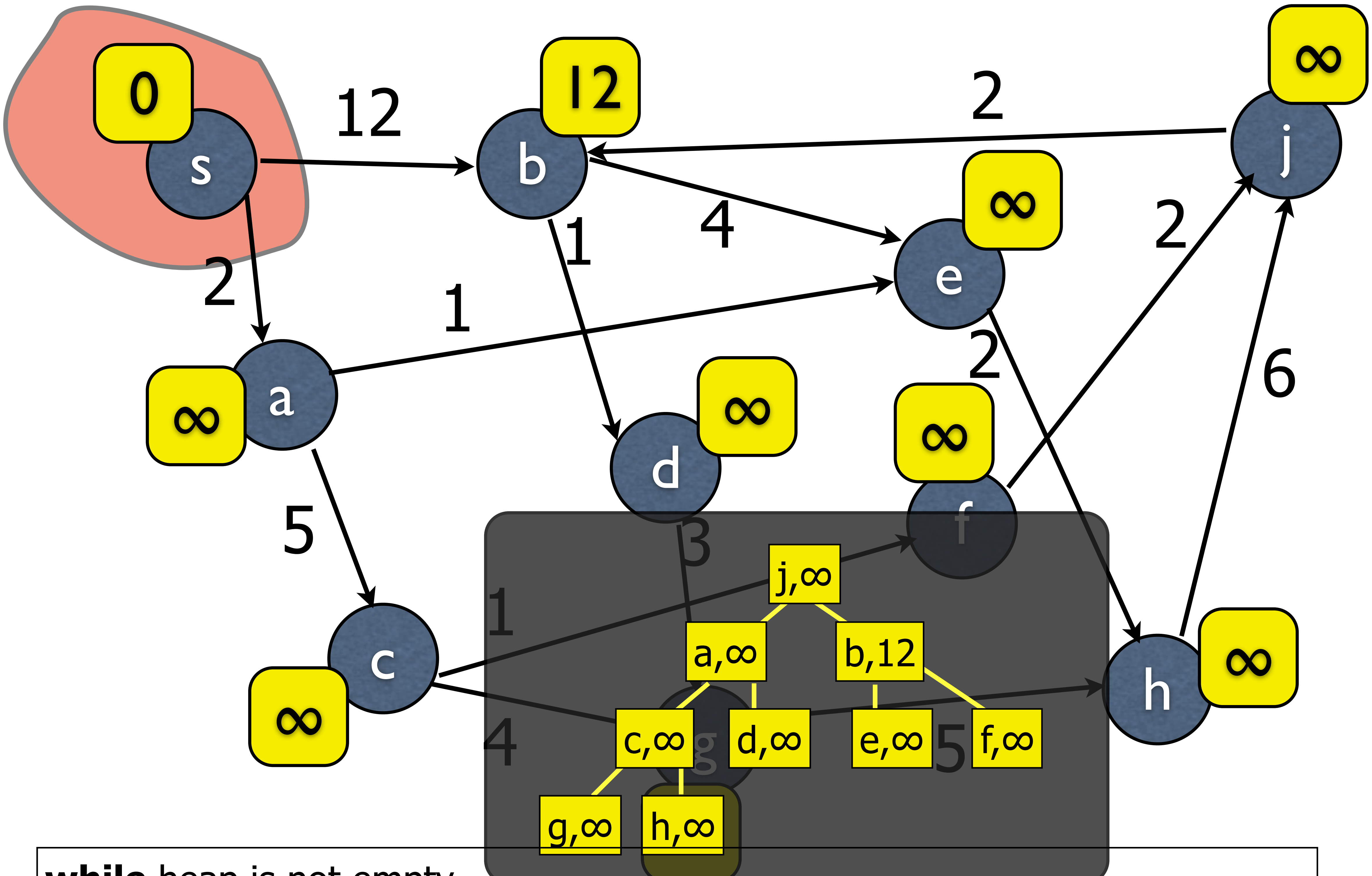
while heap is not empty,
    delete u with minimum d(u) value from heap
    for each edge (u,v)
        if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



while heap is not empty,
delete u with minimum d(u) value from heap
for each edge (u,v)
if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



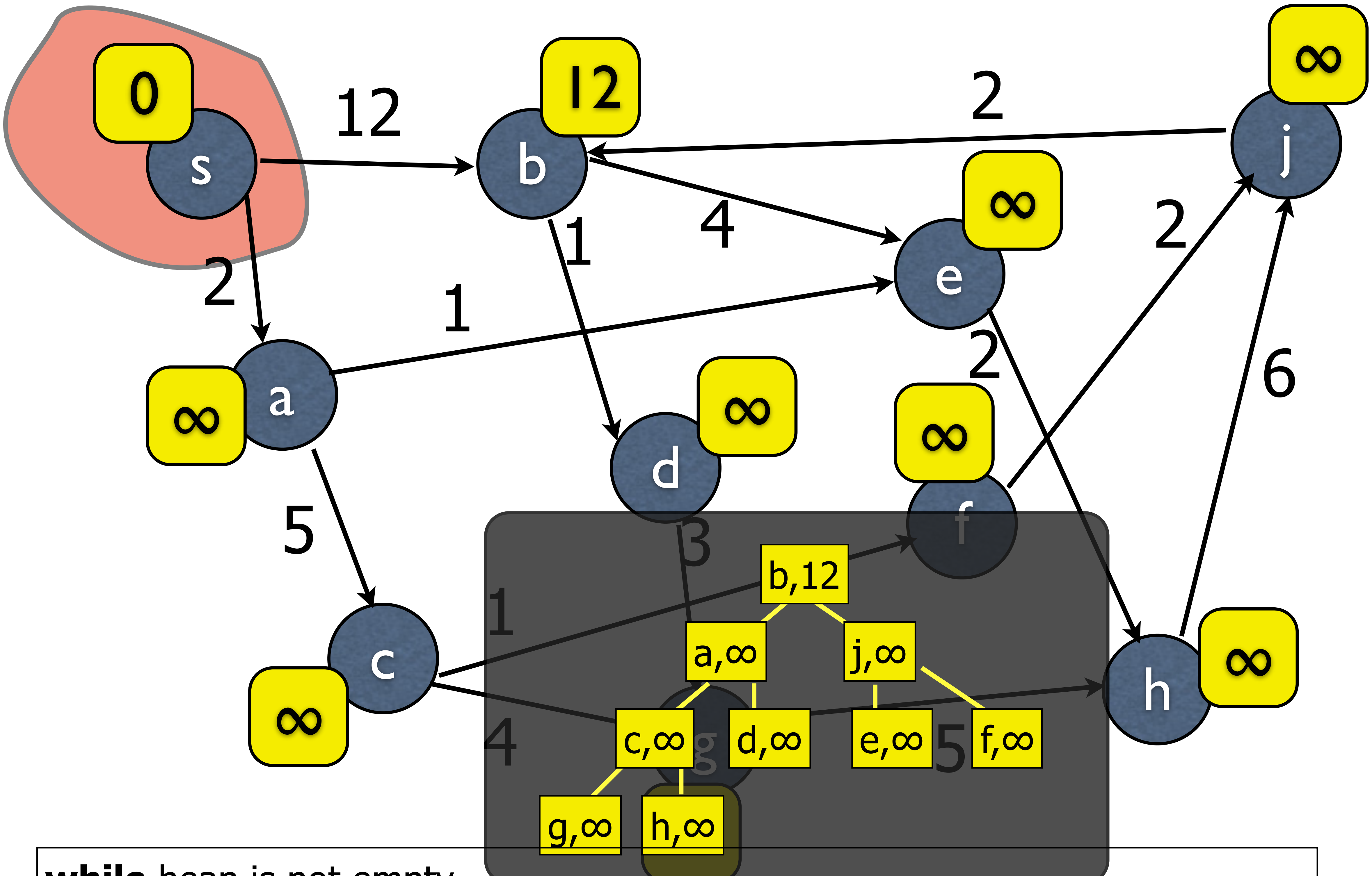
while heap is not empty,

delete u with minimum d(u) value from heap

for each edge (u,v)

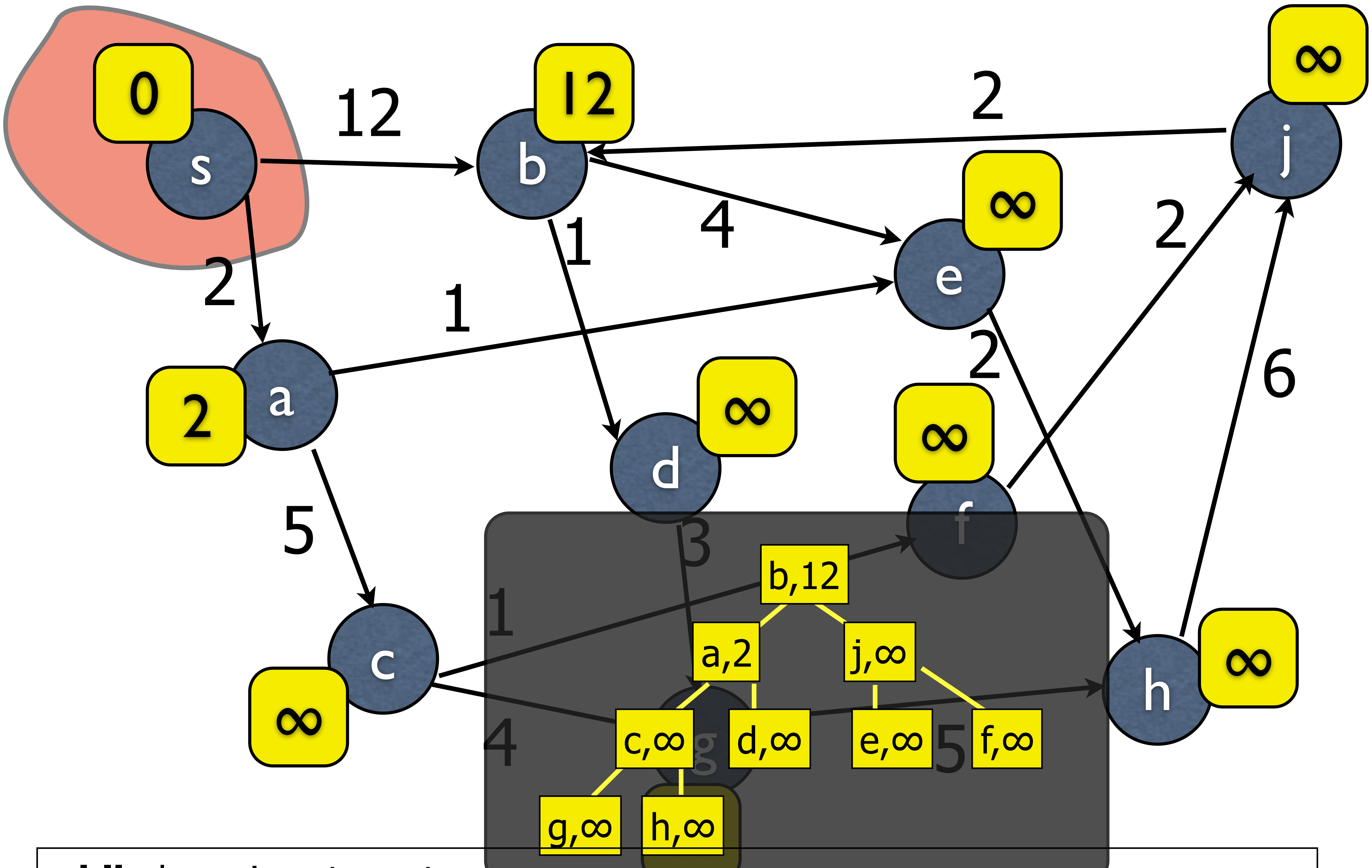
if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



while heap is not empty,
    delete u with minimum d(u) value from heap
    for each edge (u,v)
        if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



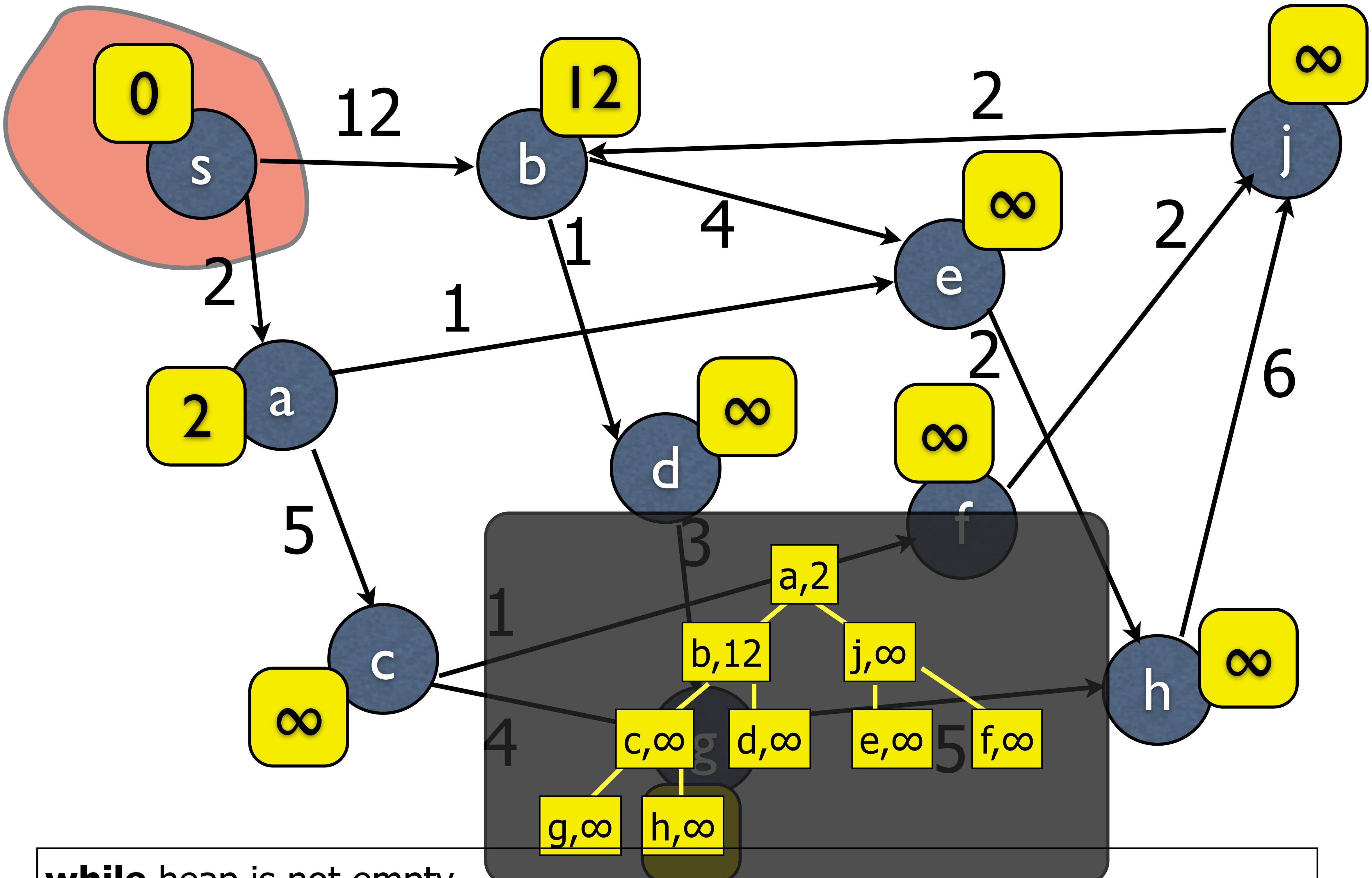
while heap is not empty,

  delete u with minimum d(u) value from heap

  for each edge (u,v)

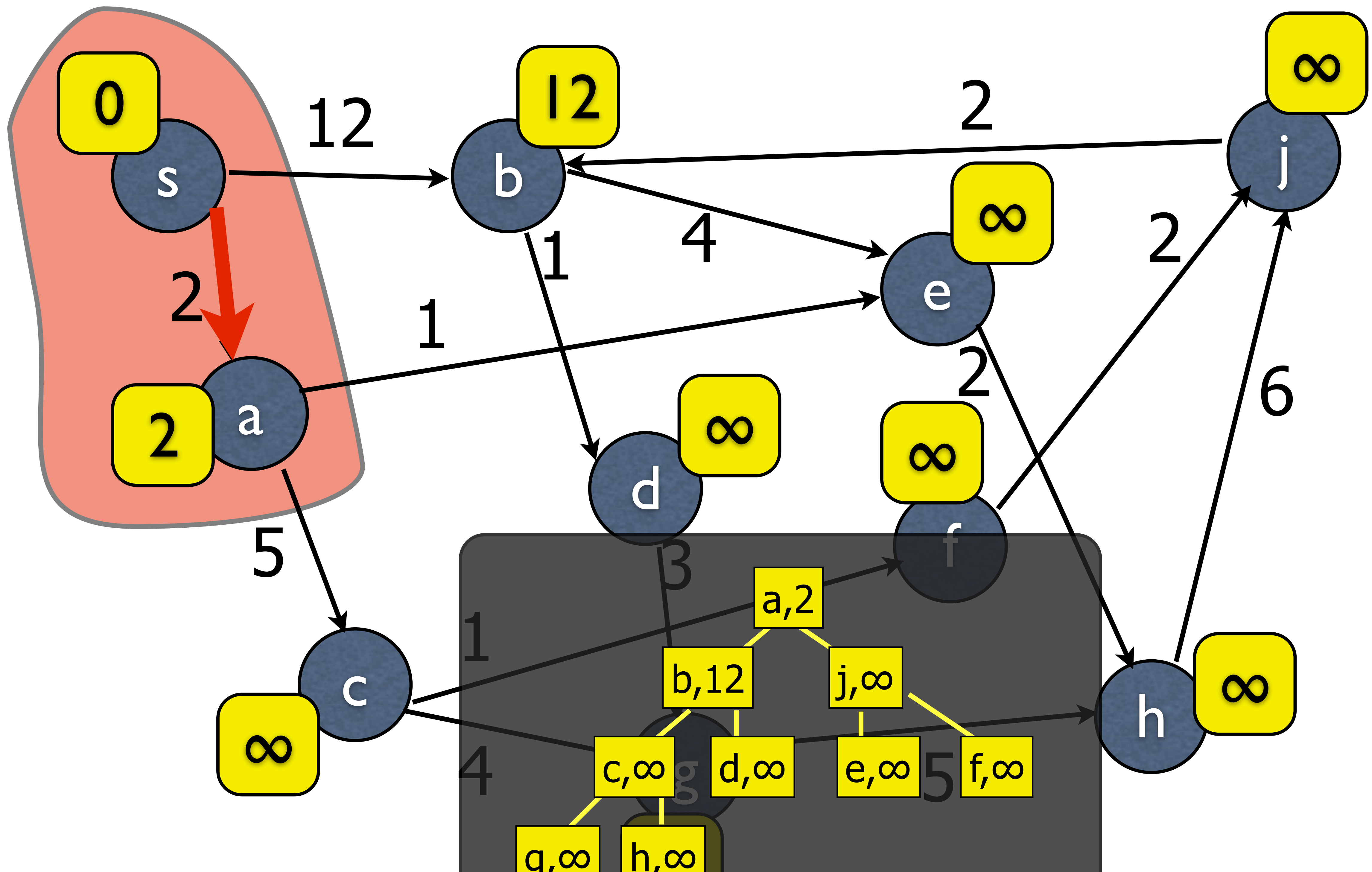
    if d(v) > d(u) + $l_{u,v}$, update d(v) = d(u) + $l_{u,v}$.

Dijkstra's Algorithm



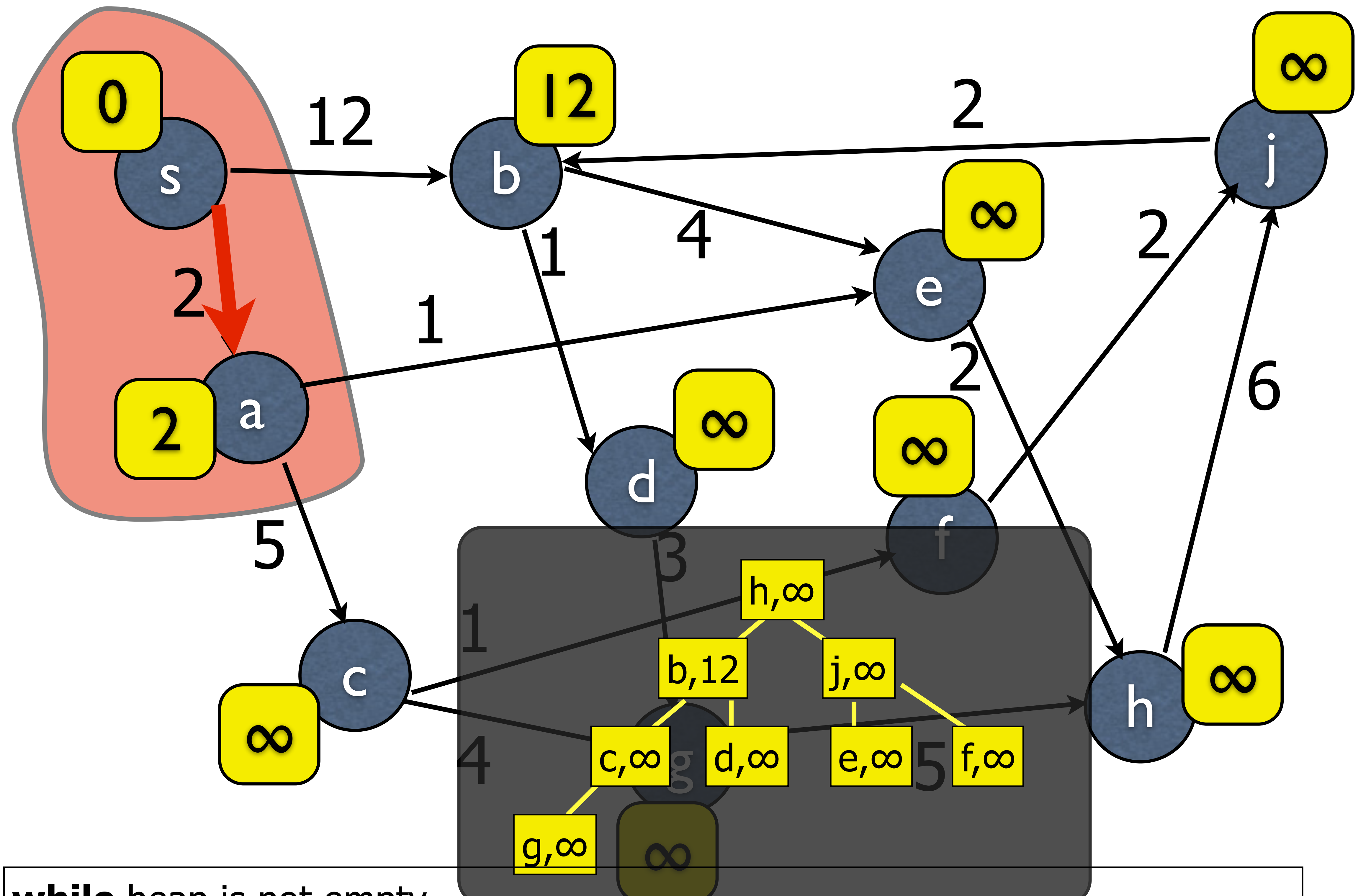
while heap is not empty,
    delete u with minimum d(u) value from heap
    for each edge (u,v)
        if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



while heap is not empty,
    delete u with minimum d(u) value from heap
    for each edge (u,v)
        if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



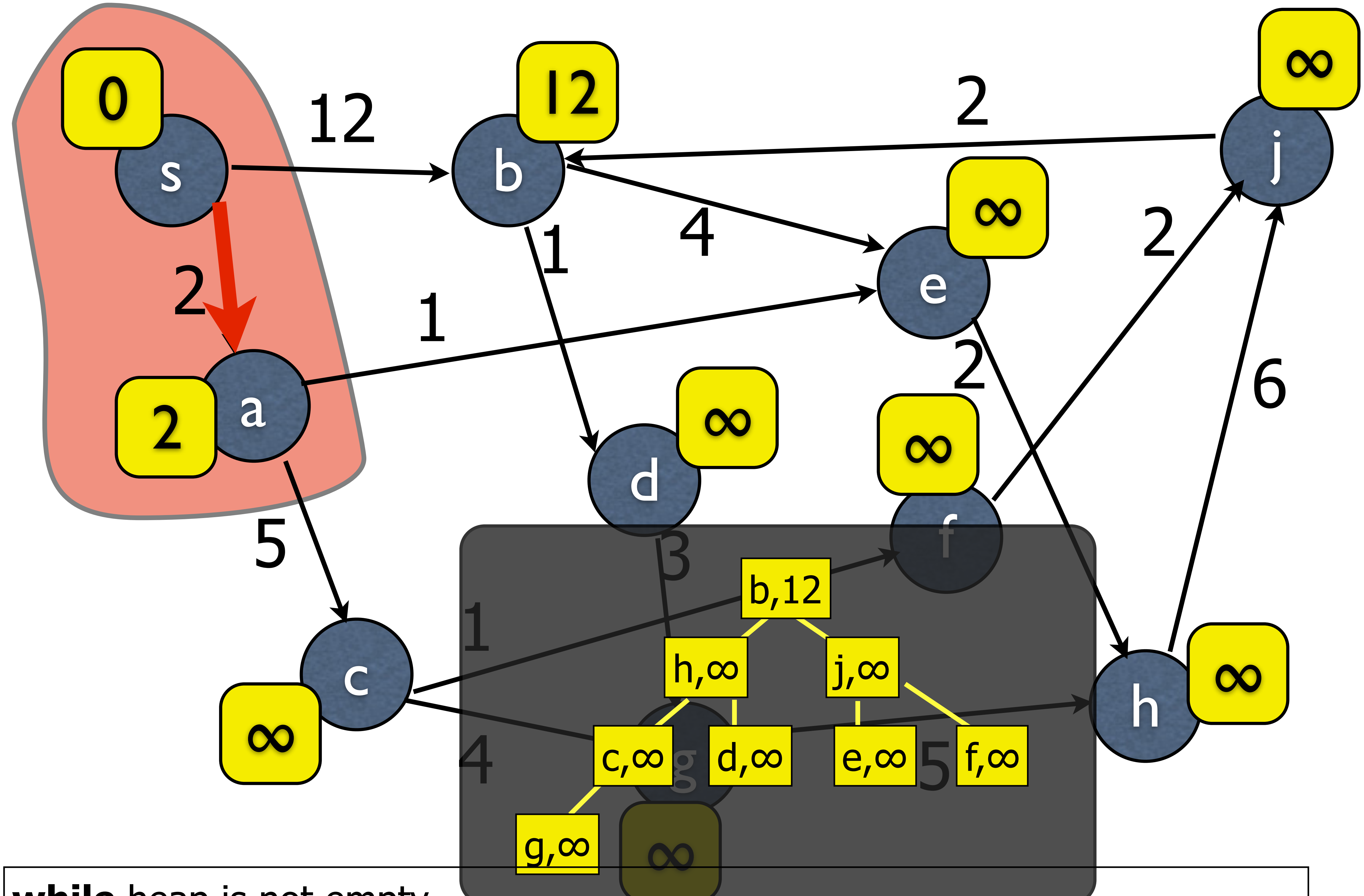
while heap is not empty,

delete u with minimum d(u) value from heap

for each edge (u,v)

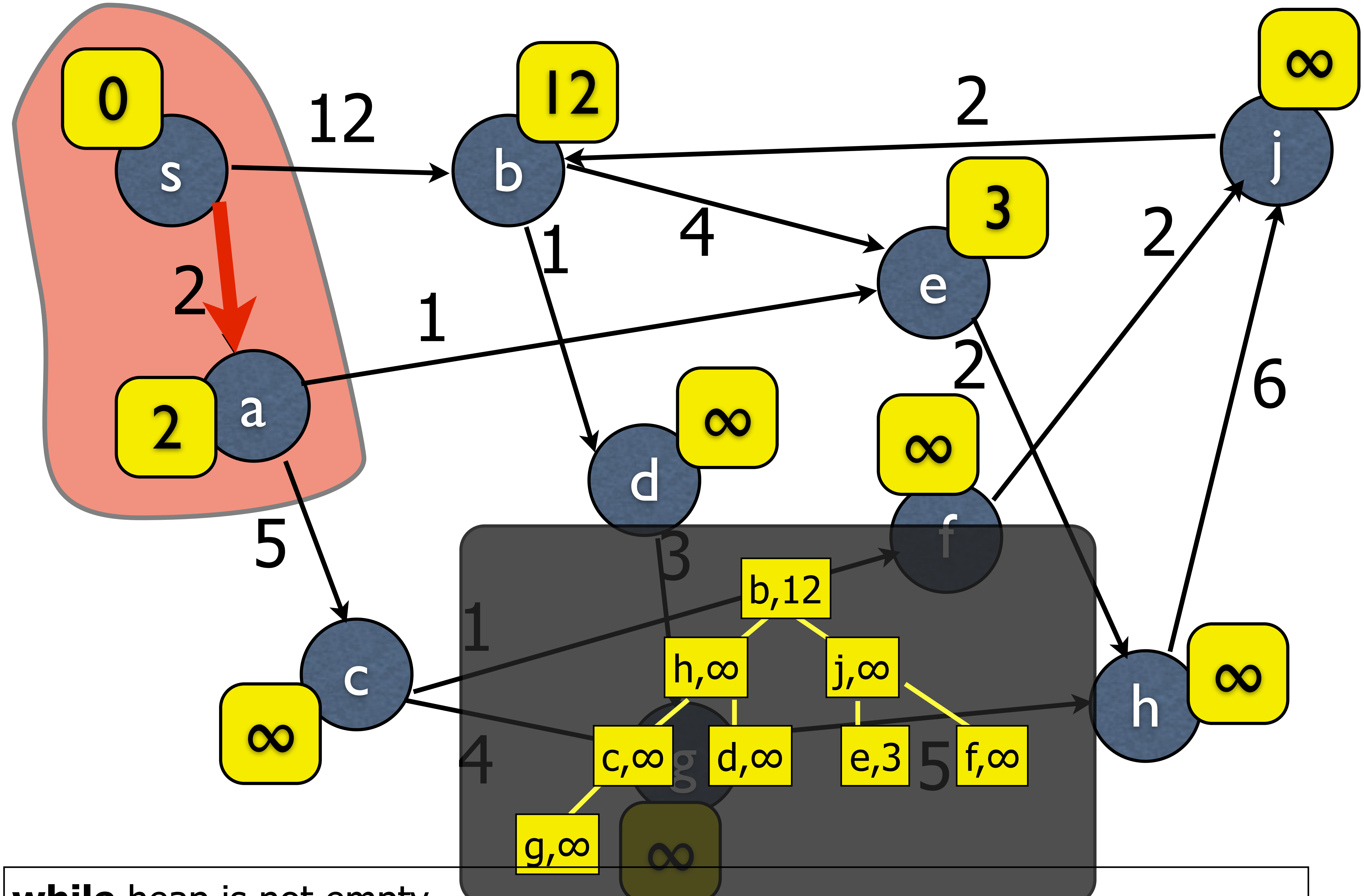
if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



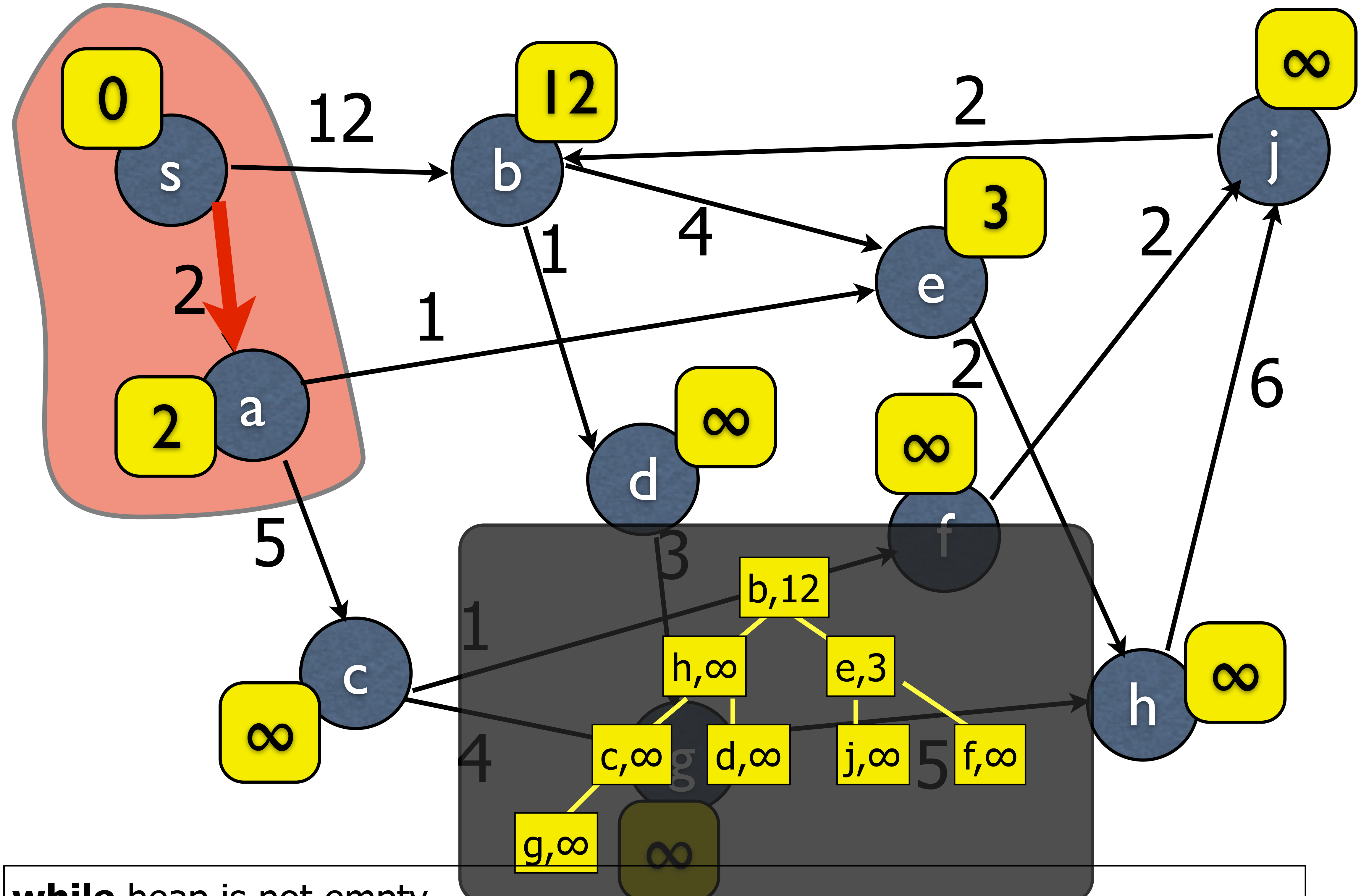
while heap is not empty,
  delete u with minimum d(u) value from heap
  for each edge (u,v)
    if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



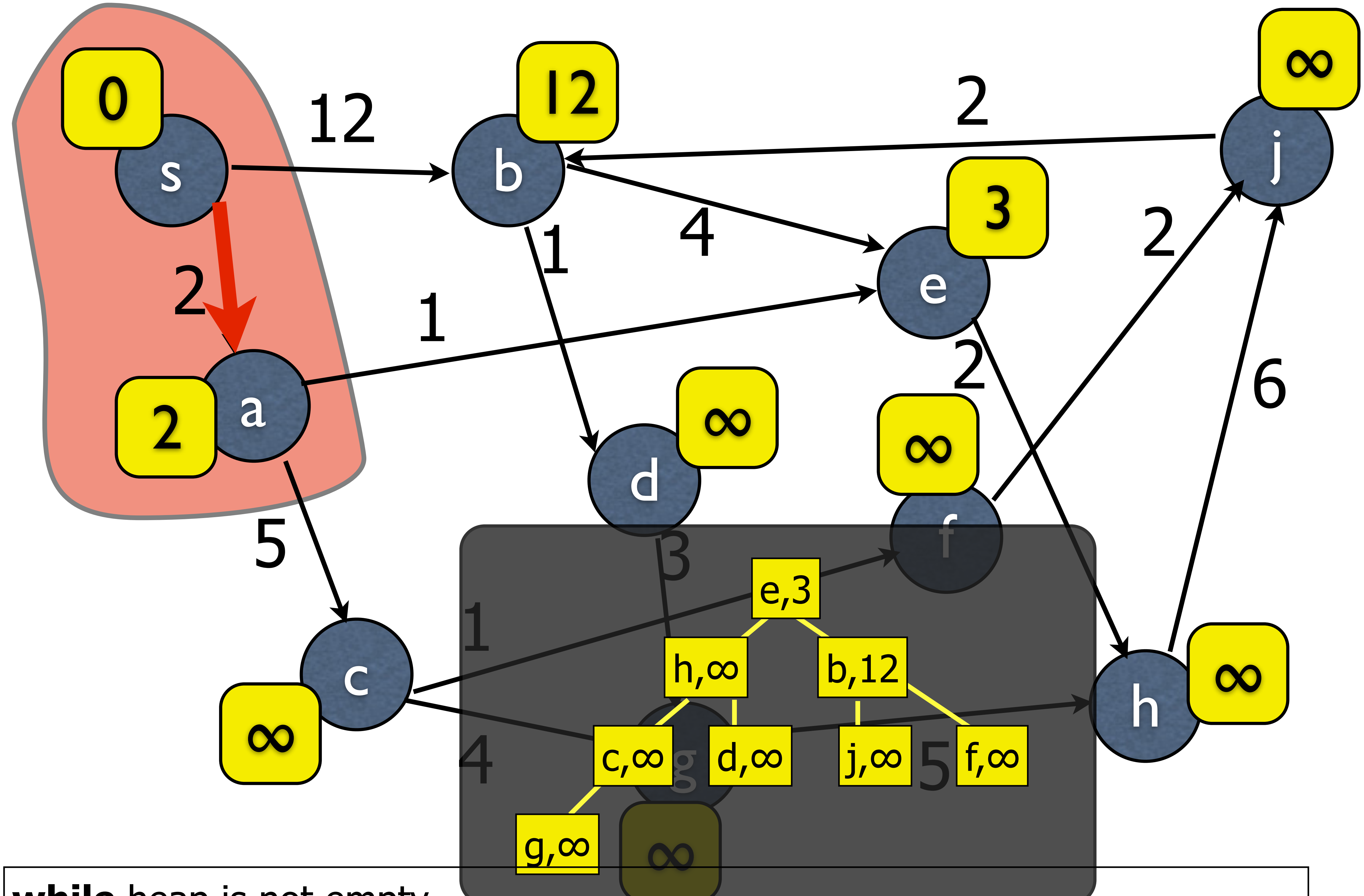
while heap is not empty,
    delete u with minimum d(u) value from heap
    for each edge (u,v)
        if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



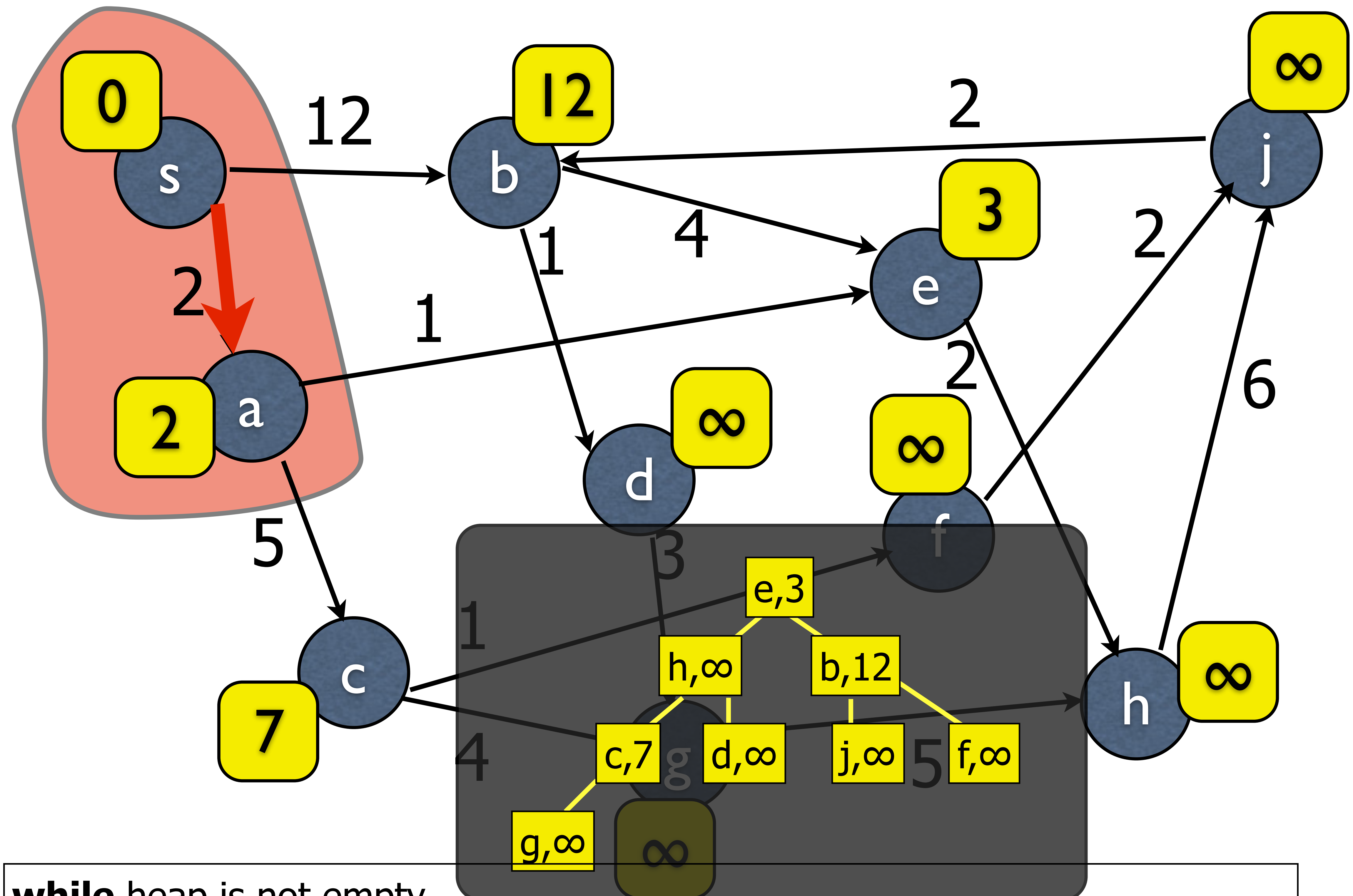
while heap is not empty,
delete u with minimum d(u) value from heap
for each edge (u,v)
if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



while heap is not empty,
  delete u with minimum d(u) value from heap
  for each edge (u,v)
    if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



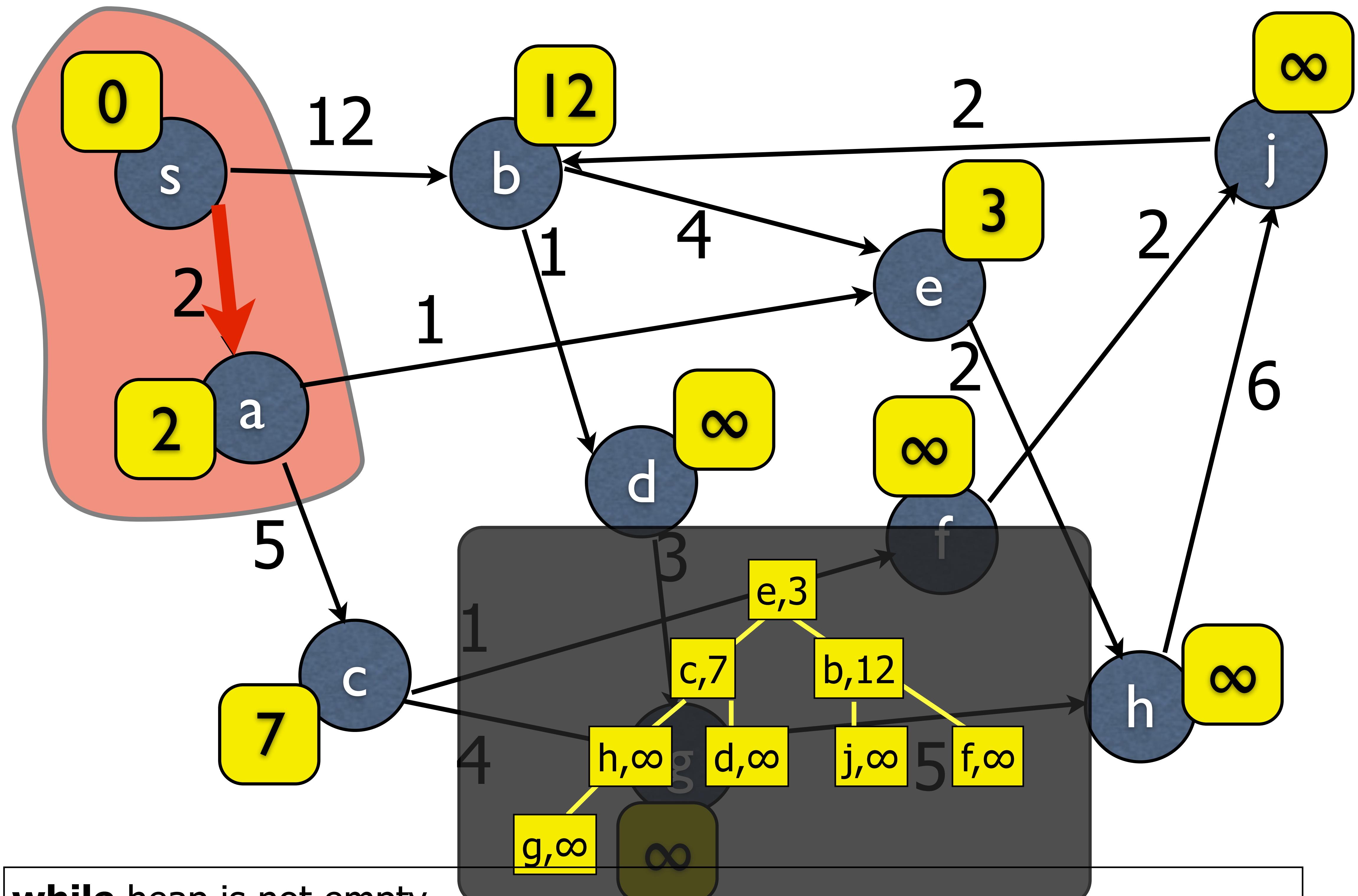
while heap is not empty,

delete u with minimum d(u) value from heap

for each edge (u,v)

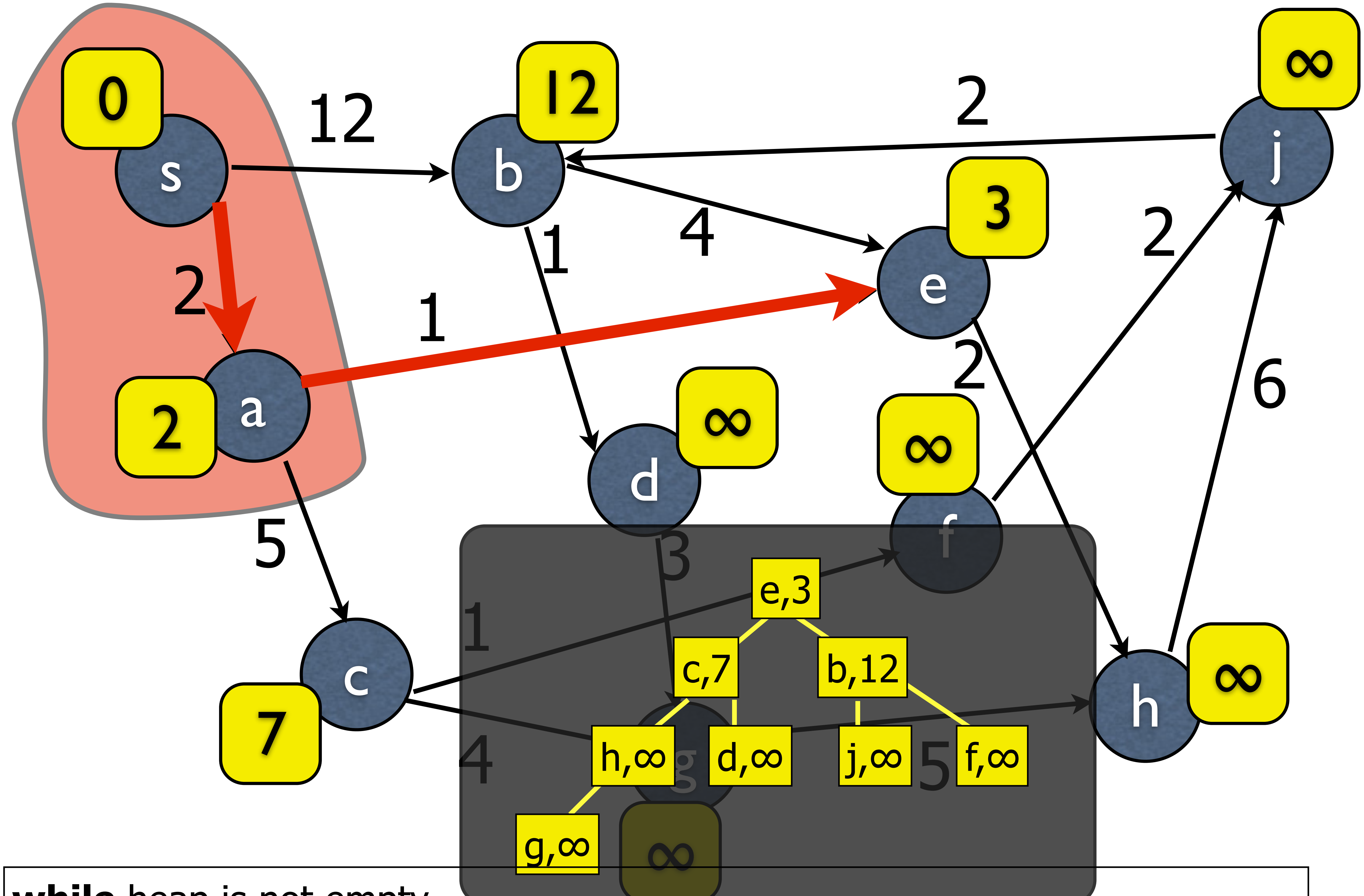
if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



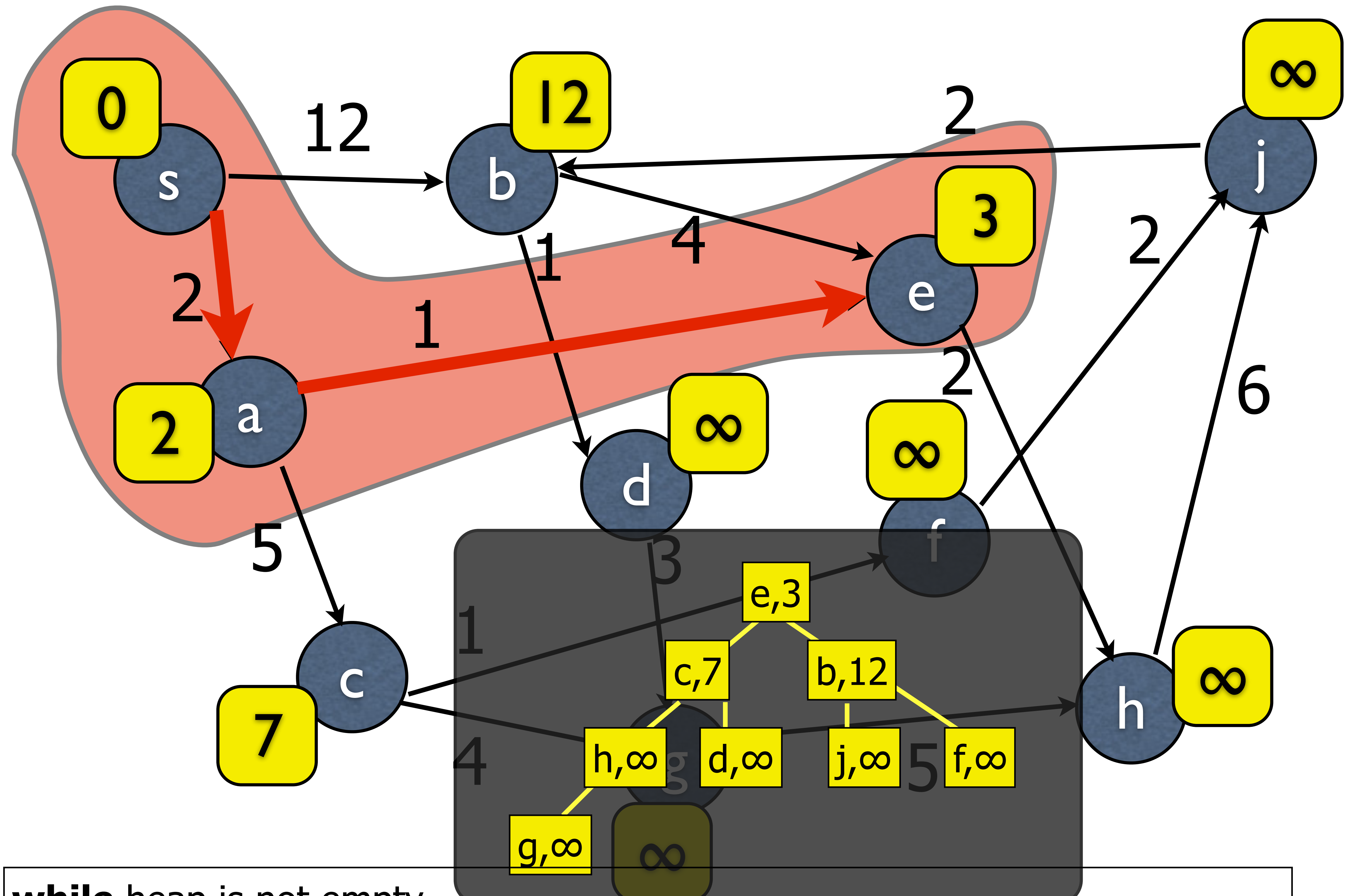
while heap is not empty,
delete u with minimum d(u) value from heap
for each edge (u,v)
if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



while heap is not empty,
    delete u with minimum d(u) value from heap
    for each edge (u,v)
        if d(v) > d(u) + $l_{u,v}$, update d(v) = d(u) + $l_{u,v}$.

Dijkstra's Algorithm



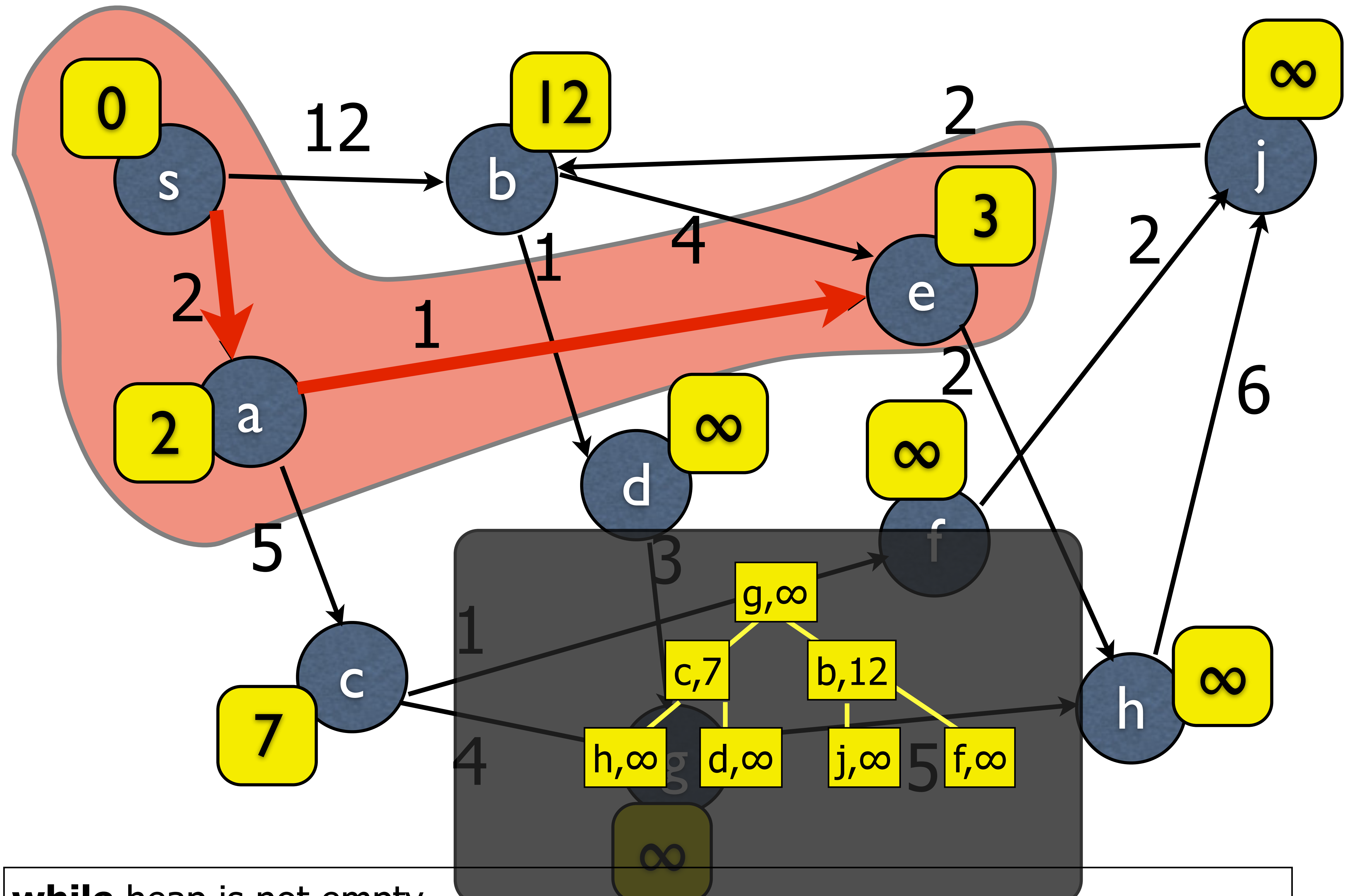
while heap is not empty,

delete u with minimum d(u) value from heap

for each edge (u,v)

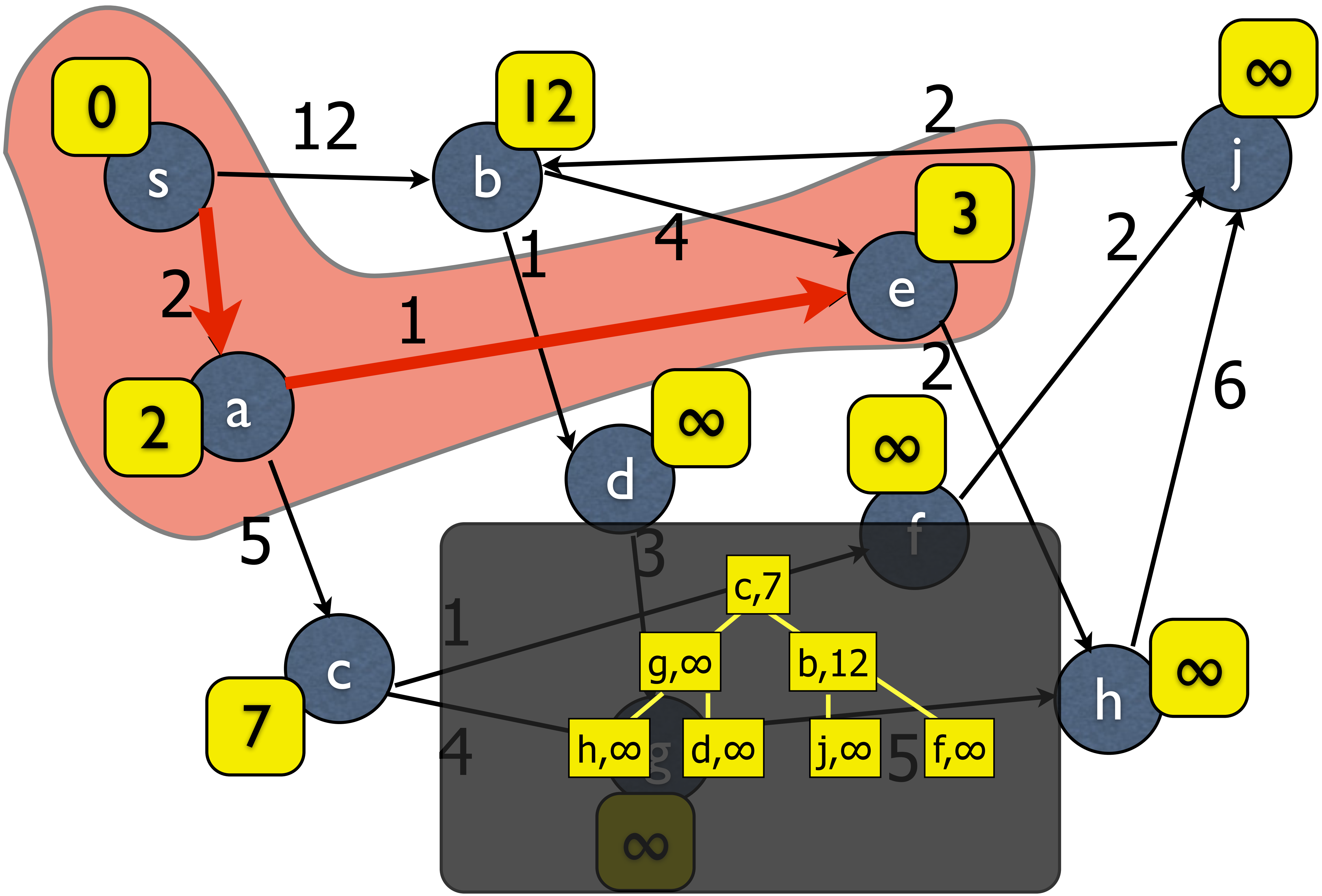
if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm



while heap is not empty,
delete u with minimum d(u) value from heap
for each edge (u,v)
if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Dijkstra's Algorithm

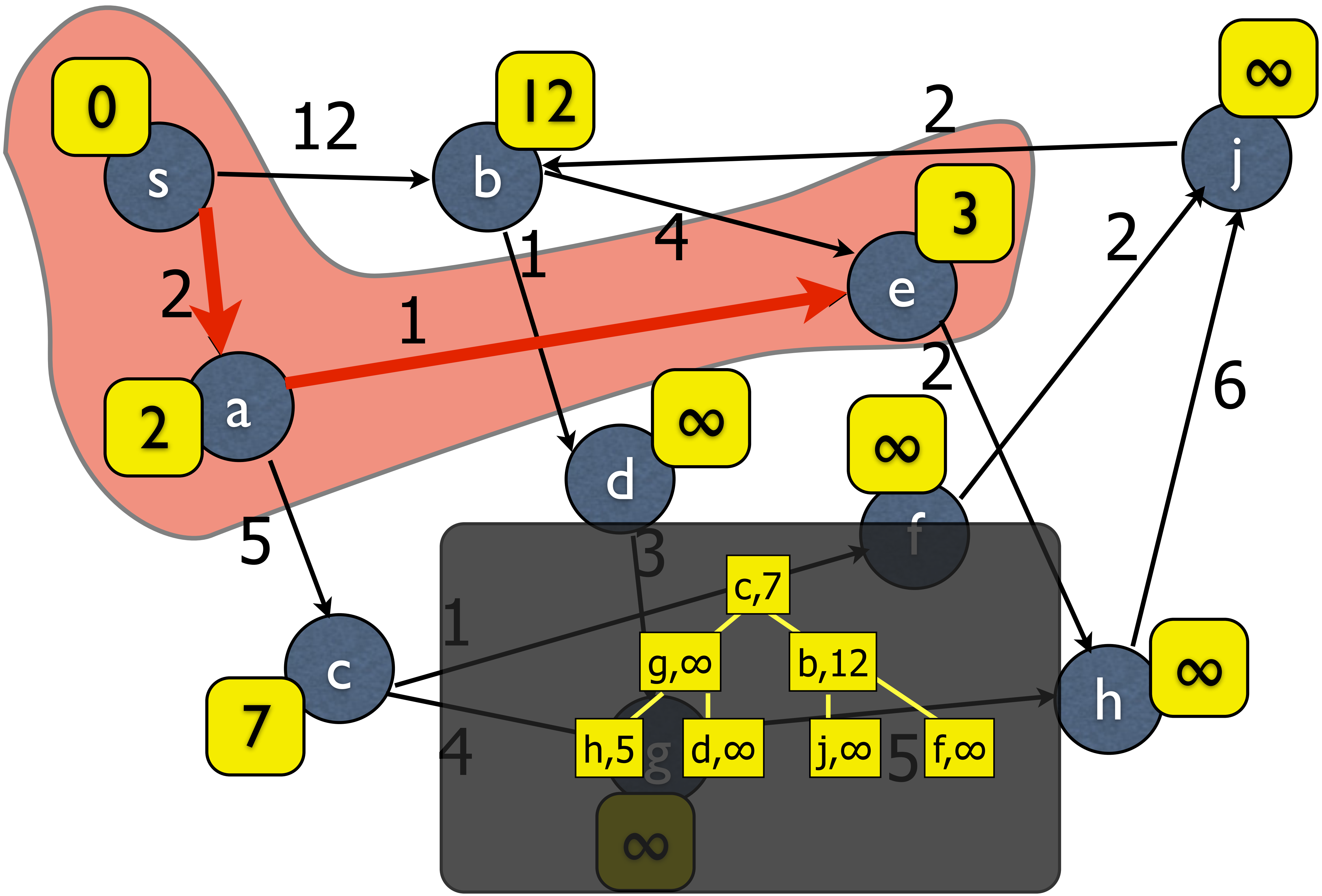


while there is edge from undiscovered vertex to discovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

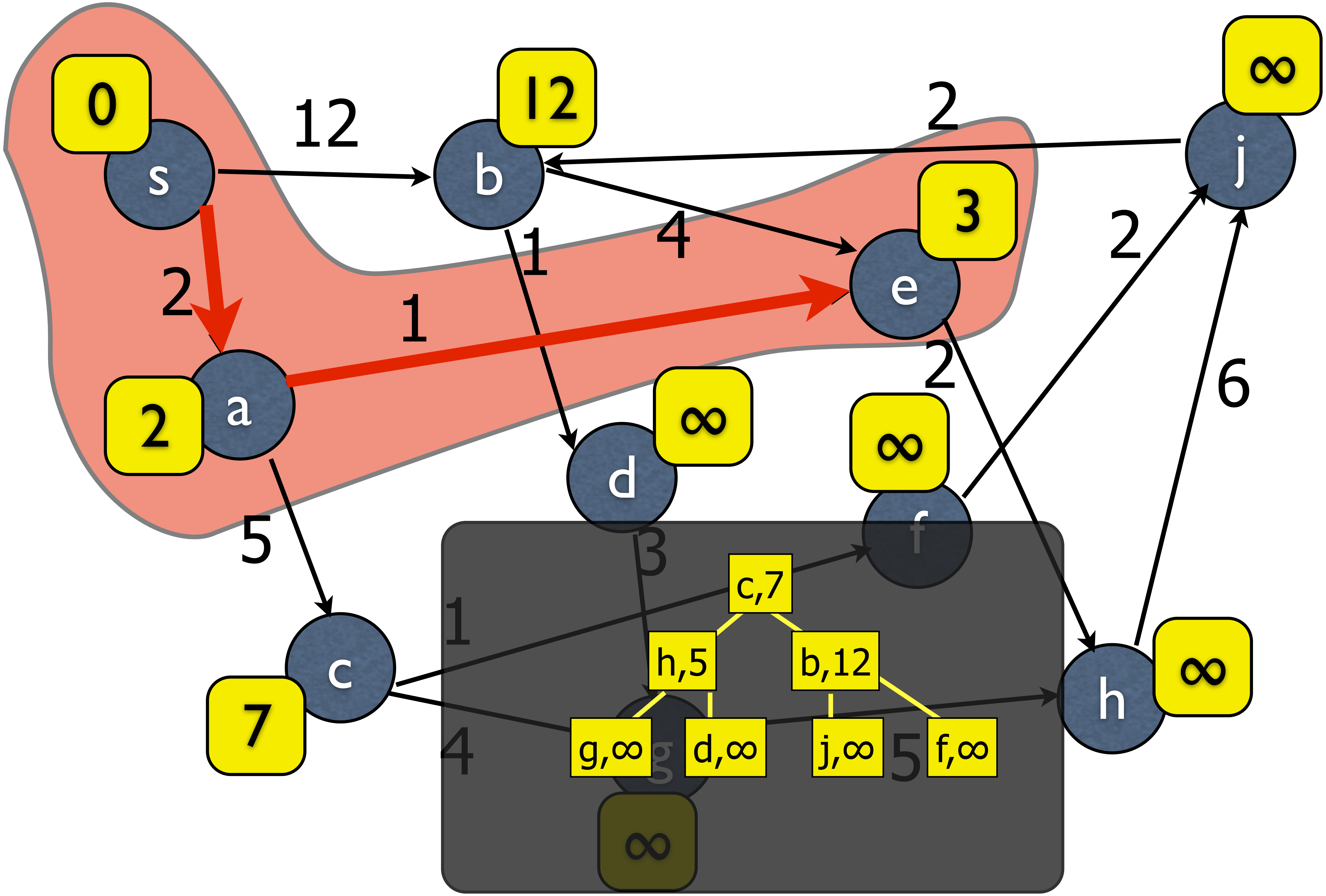
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra’s Algorithm



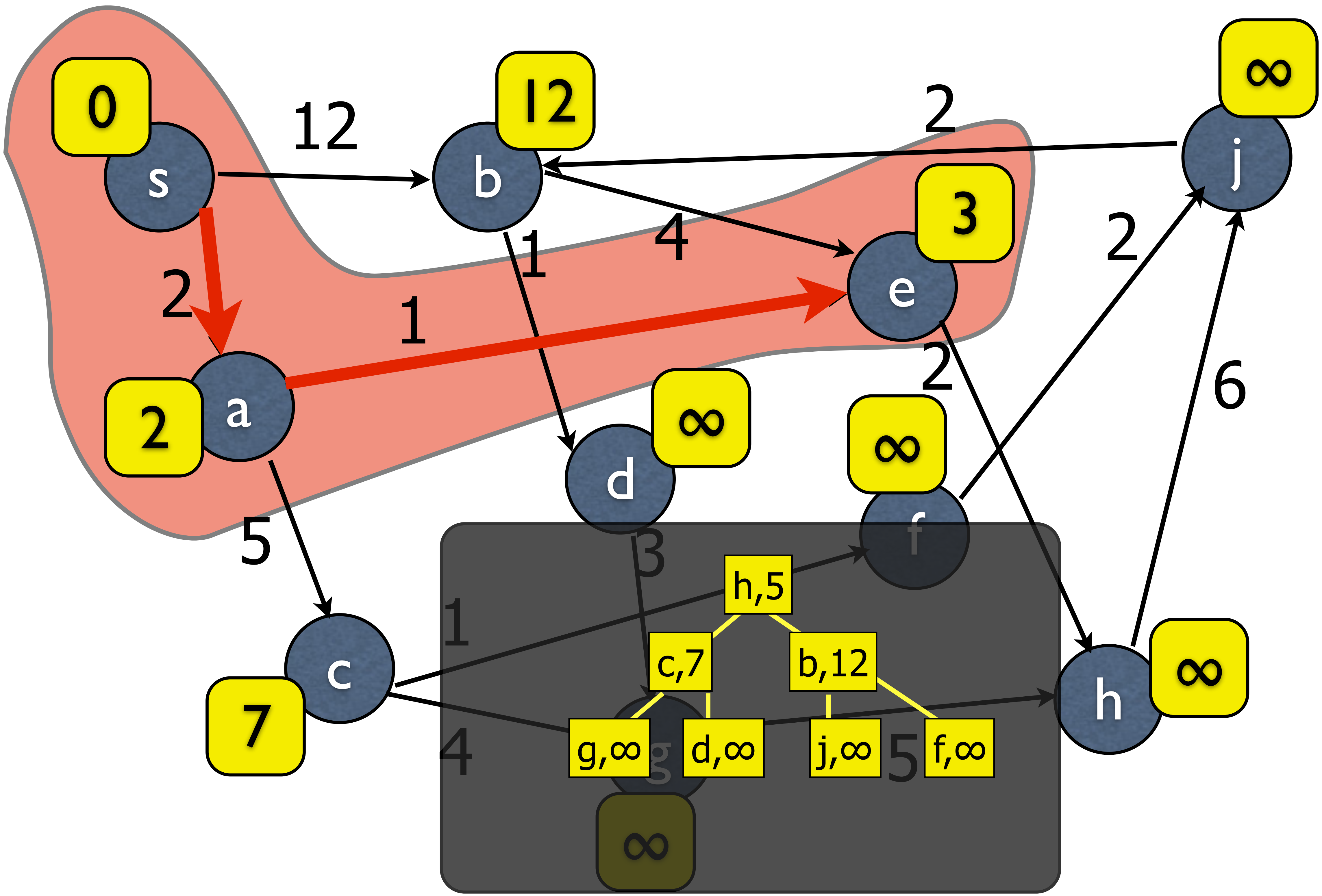
while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

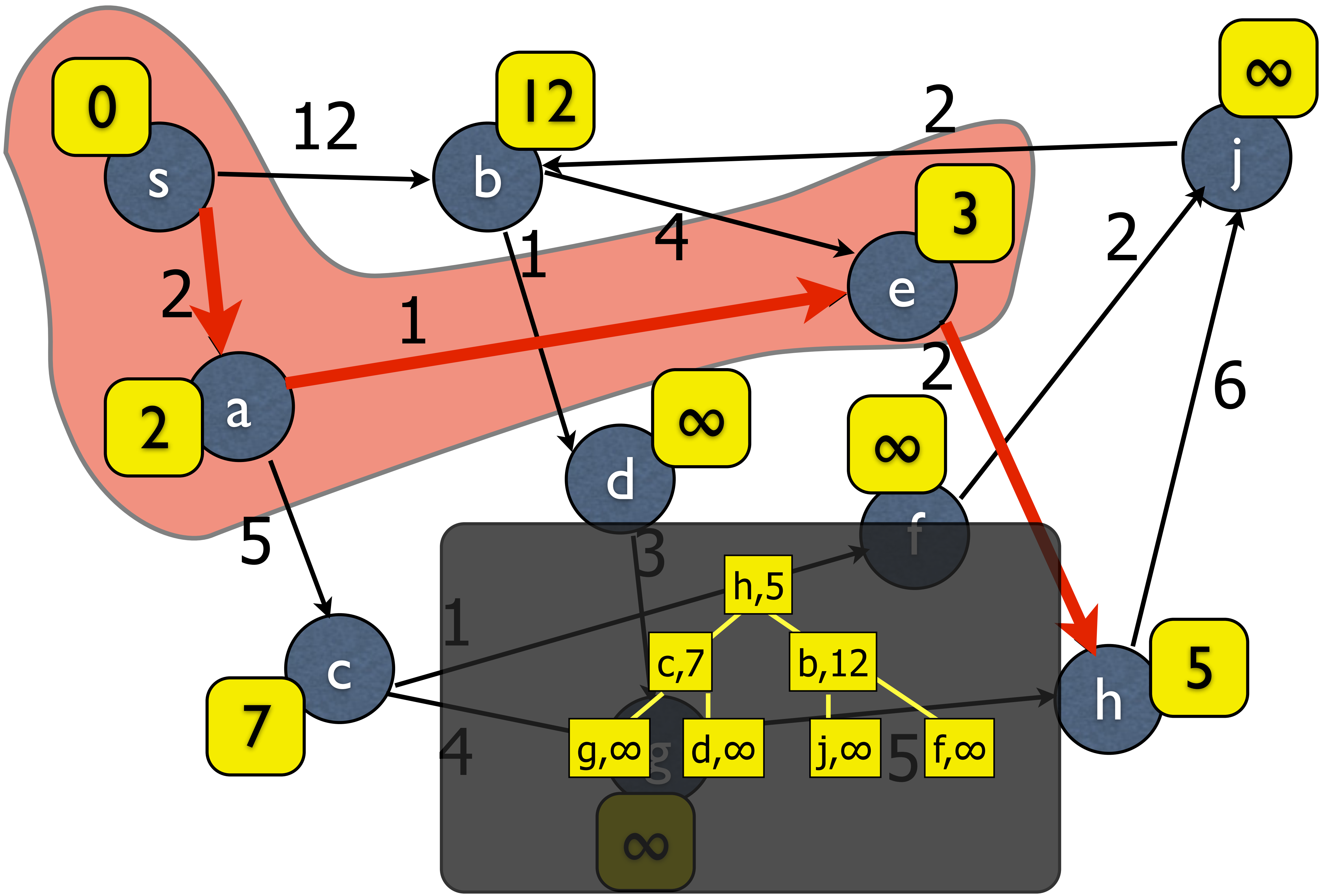


while there is edge from undiscovered vertex to discovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

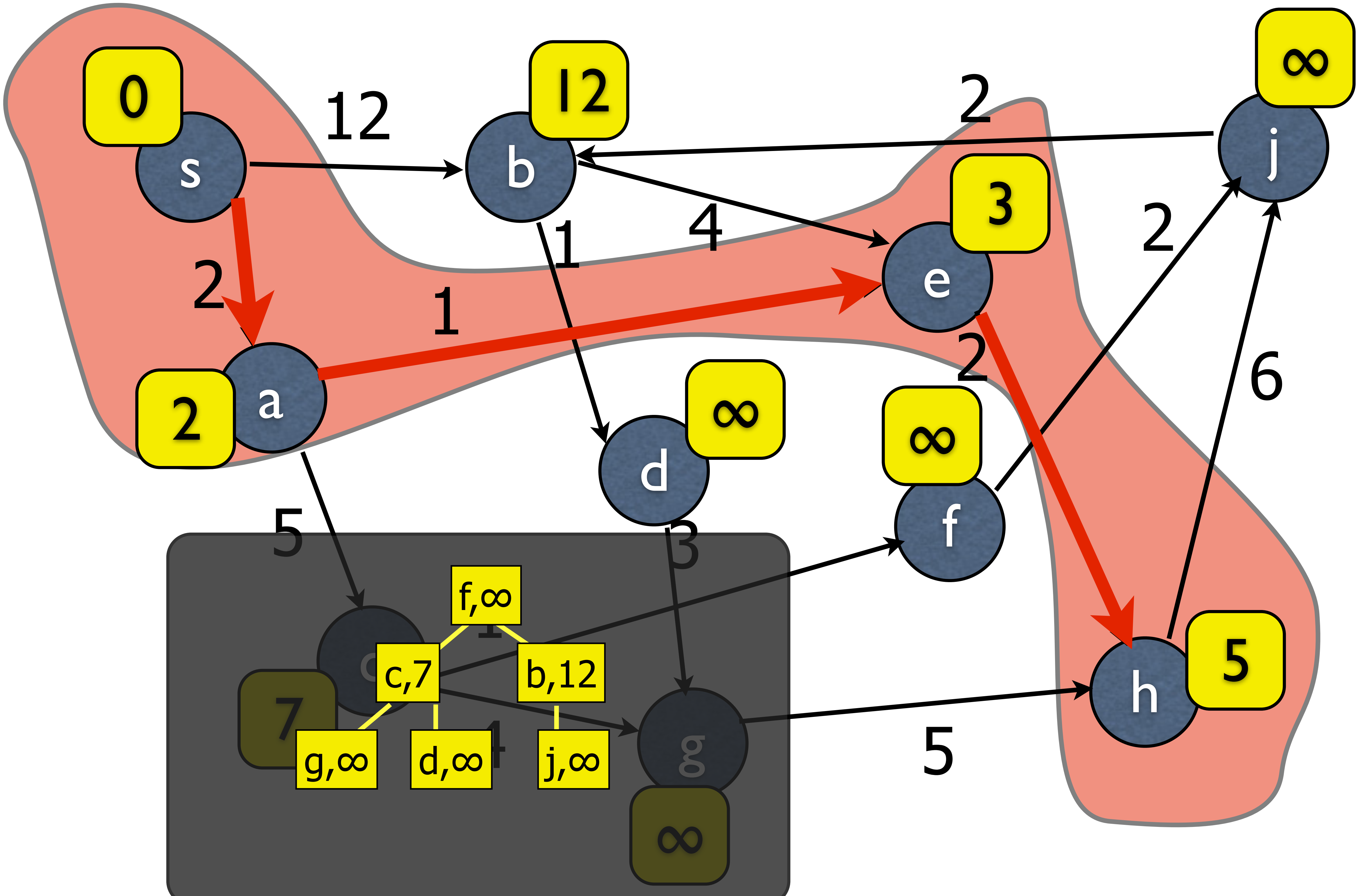
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



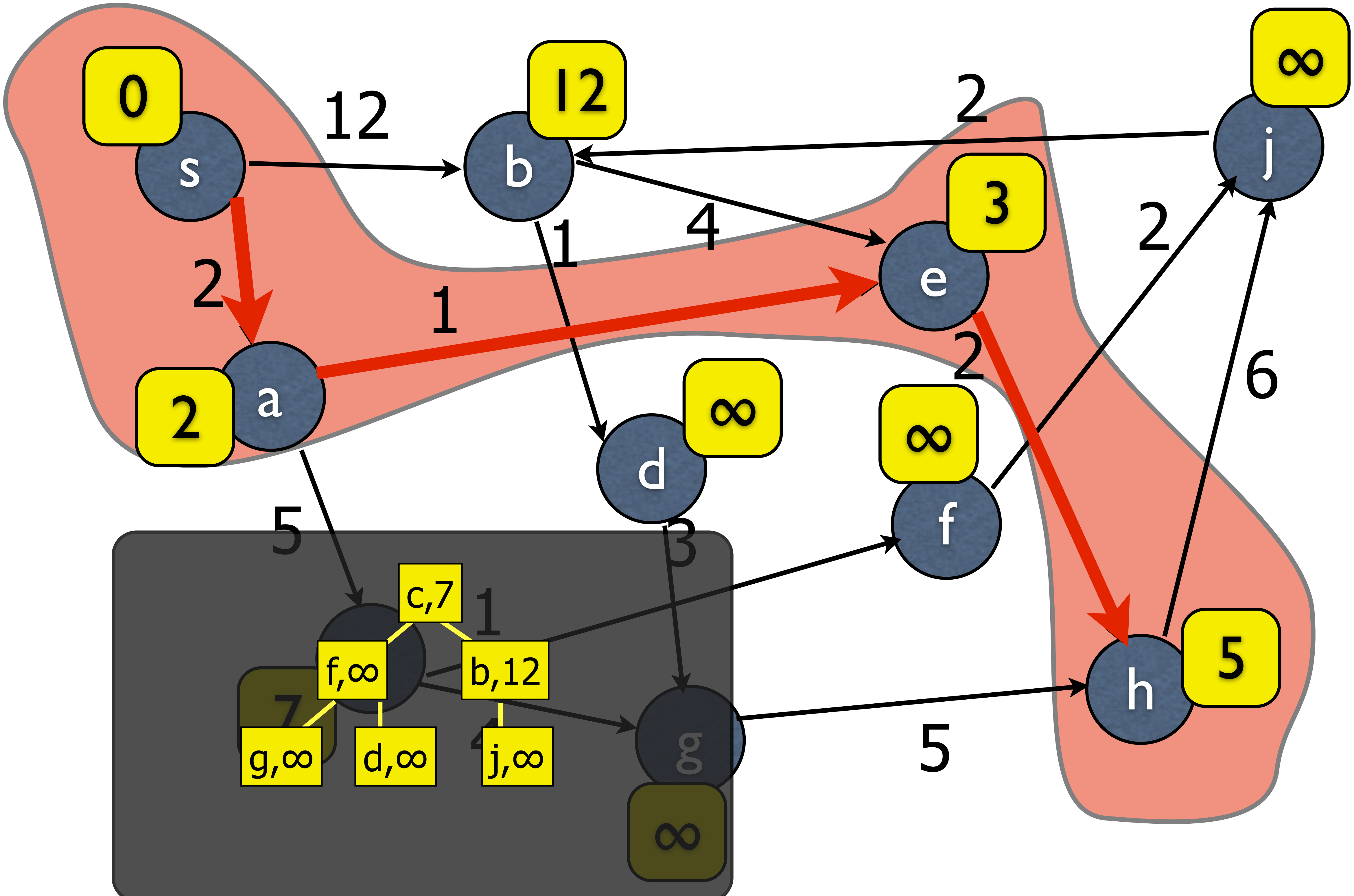
while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



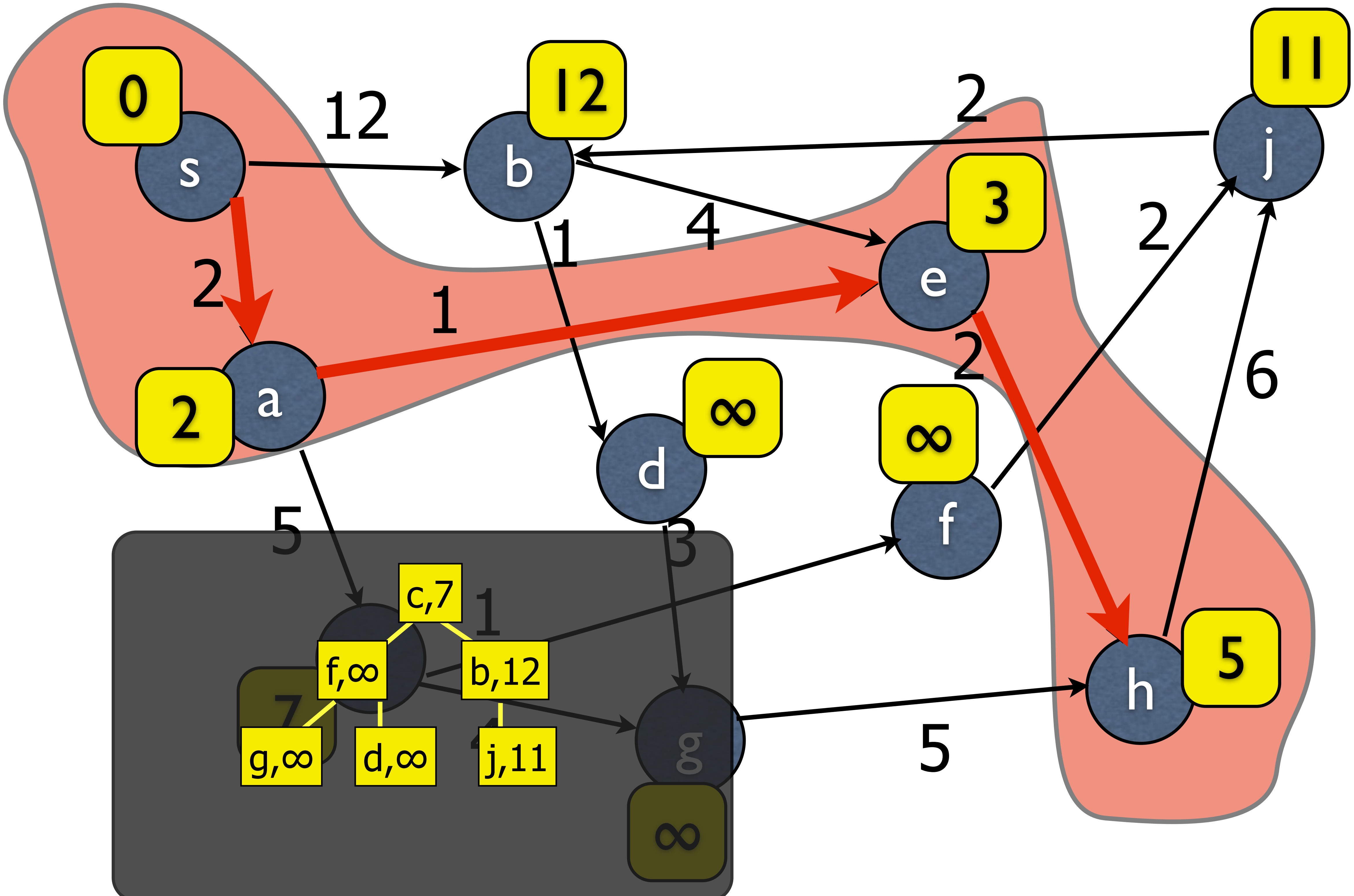
while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra’s Algorithm



while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

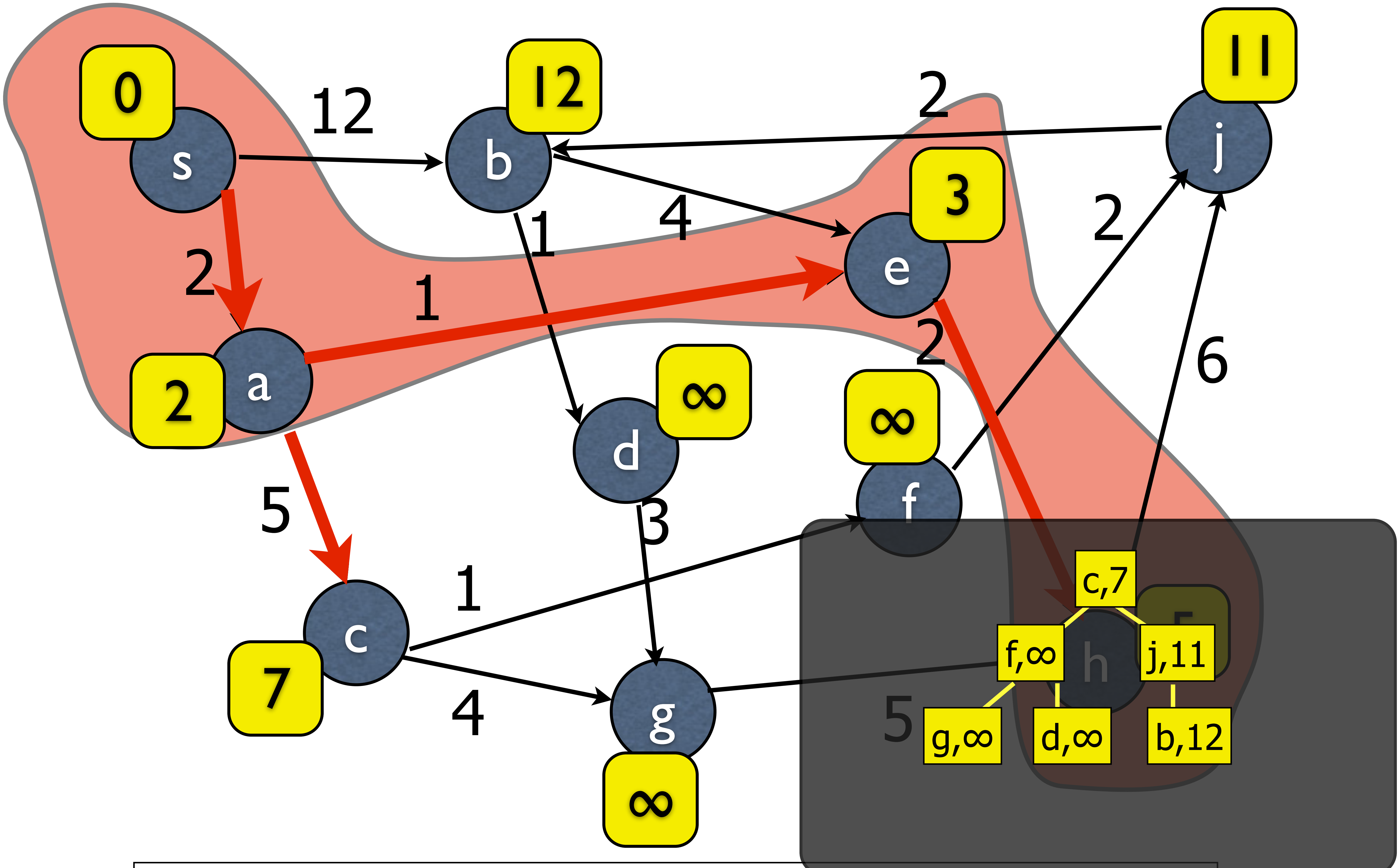


while there is edge from undiscovered vertex to discovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

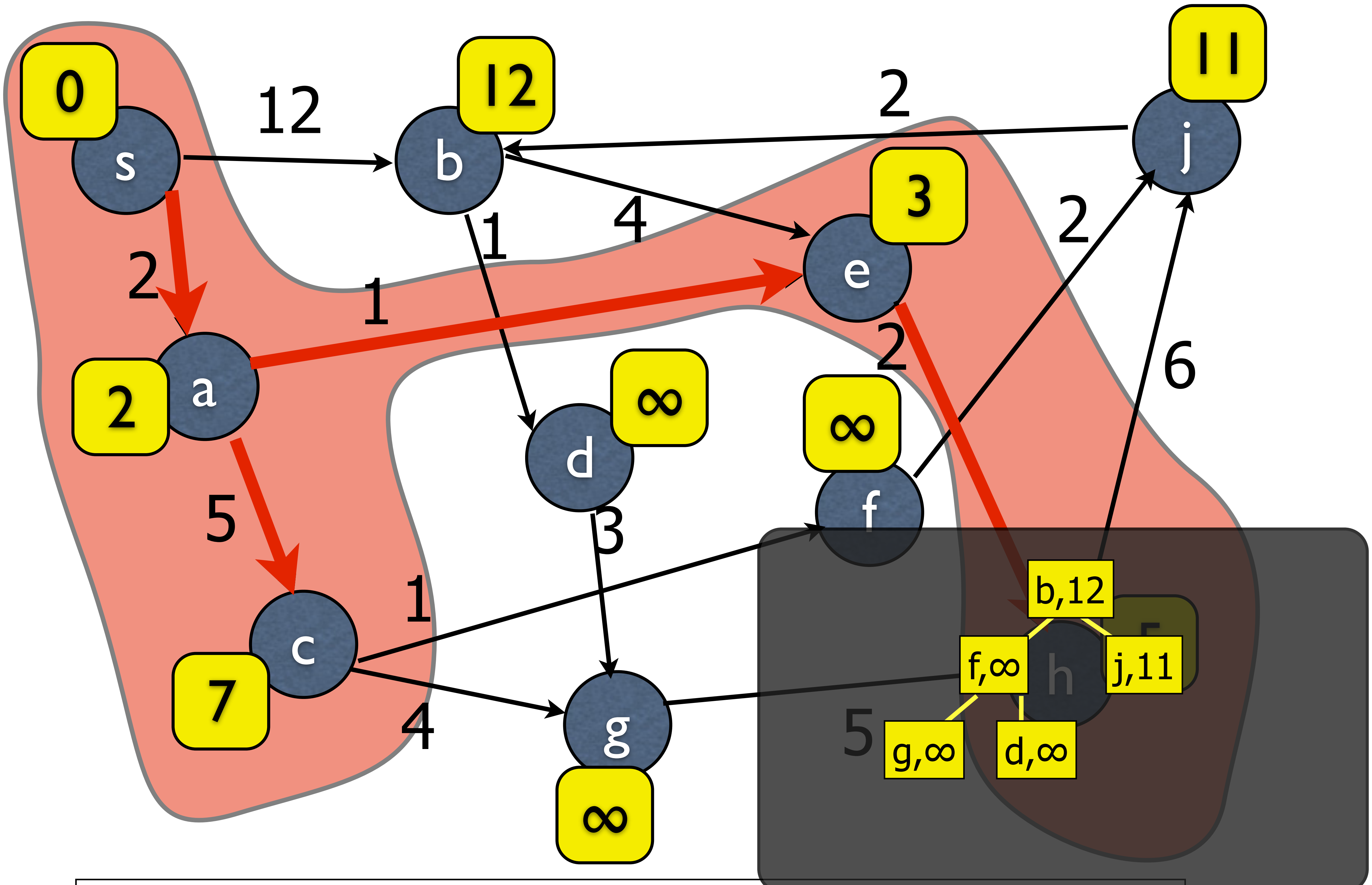


while there is edge from undiscovered vertex to discovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

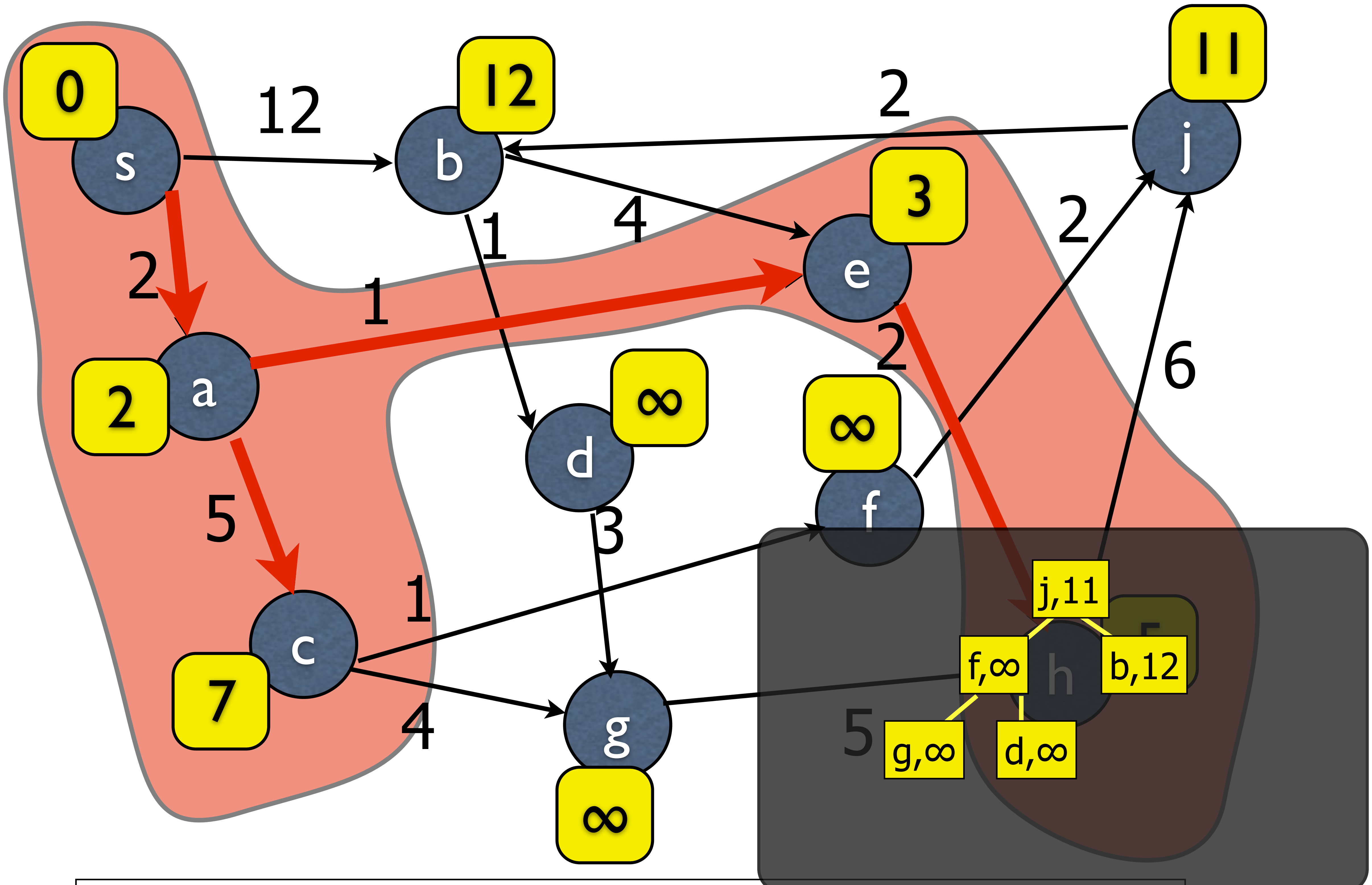
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



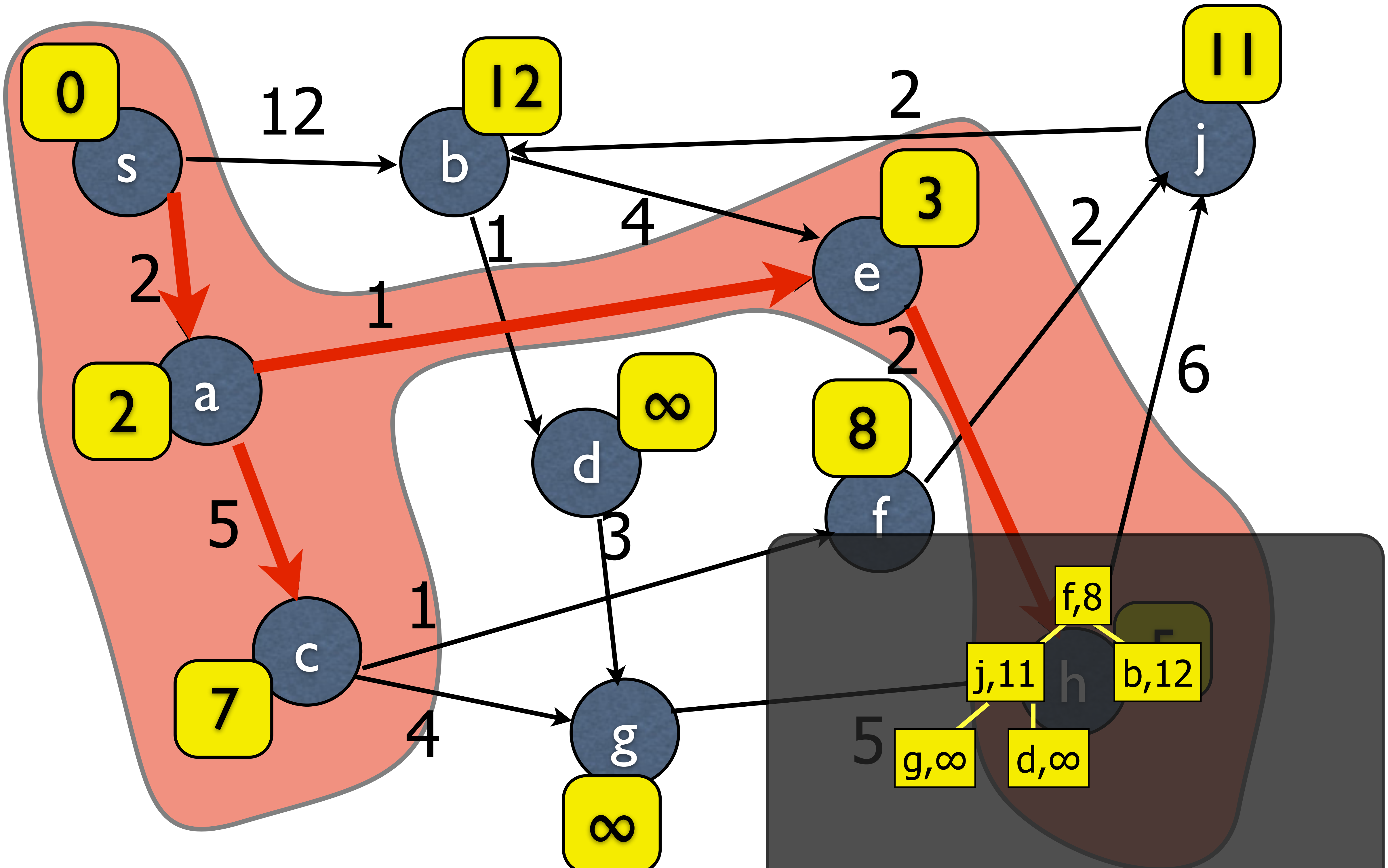
while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



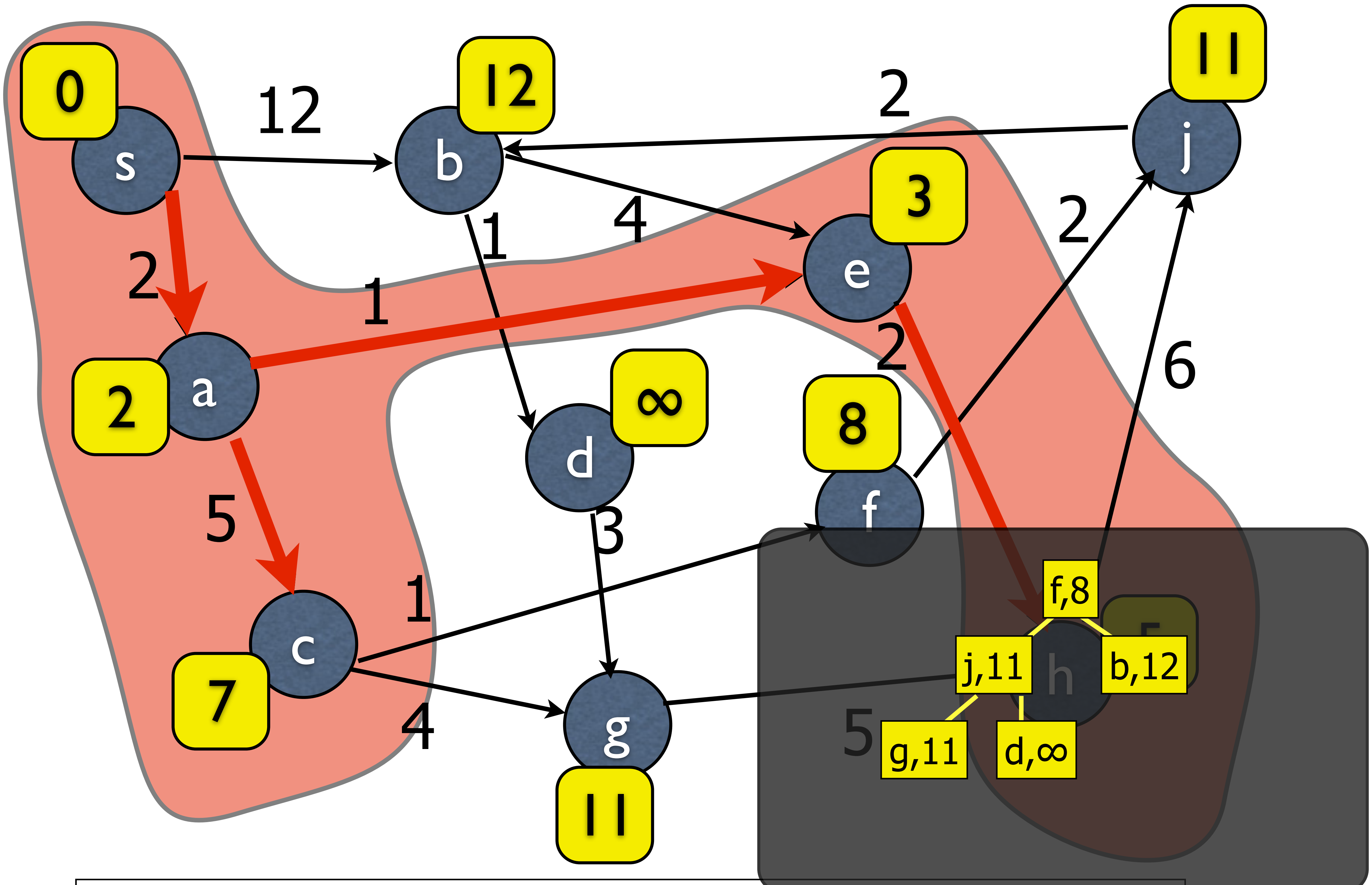
while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



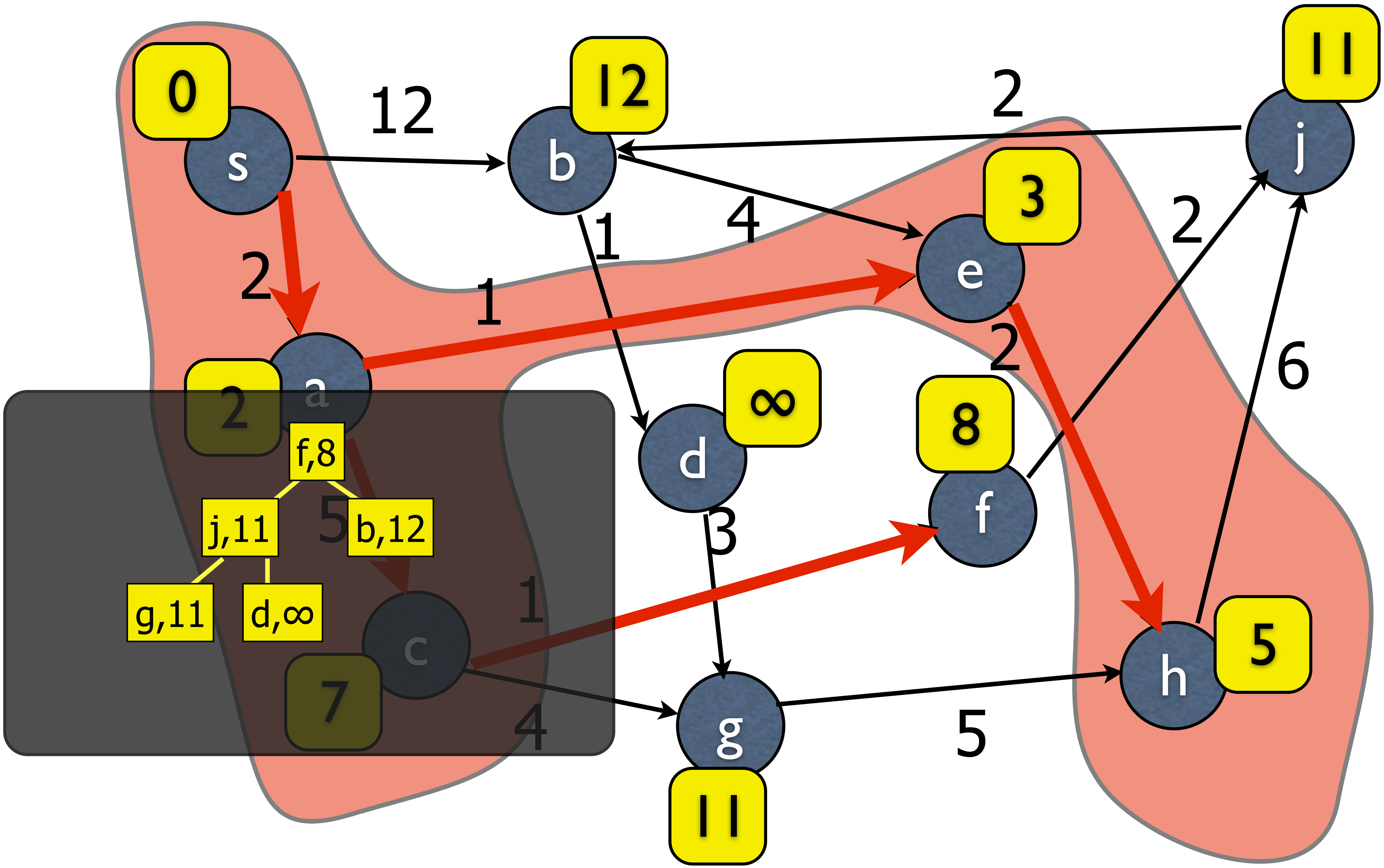
while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

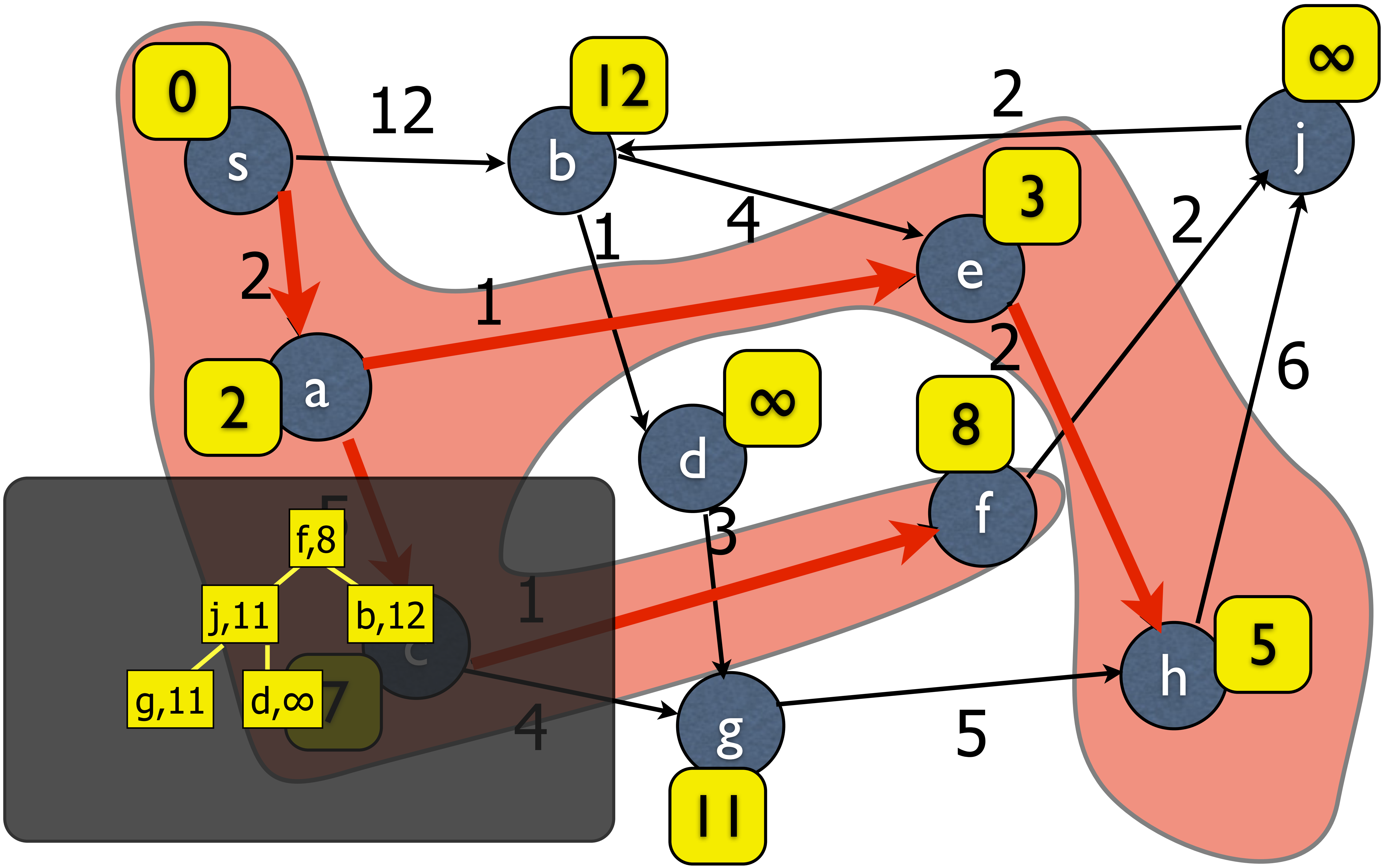


while there is edge from undiscovered vertex to discovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

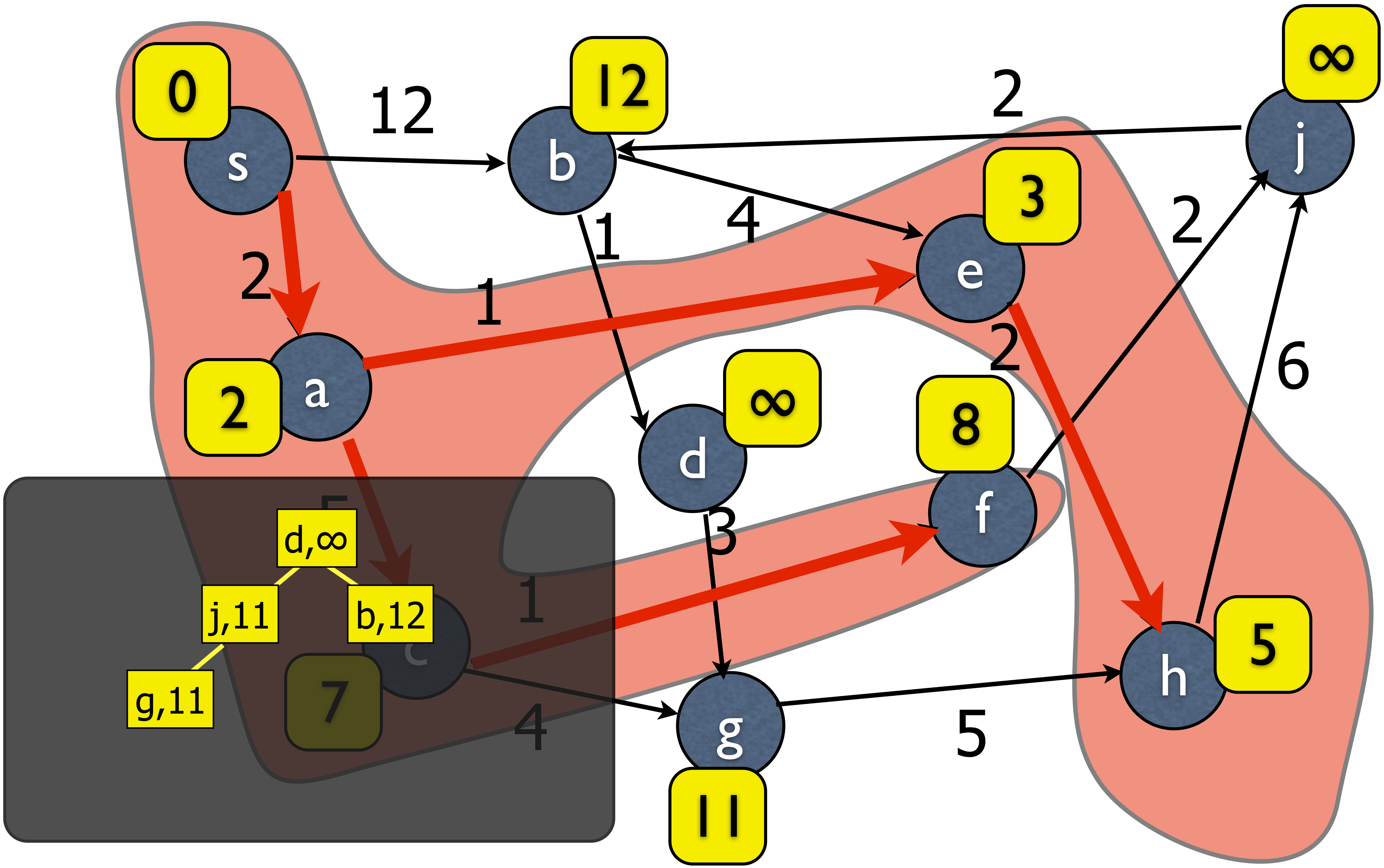


while there is edge from undiscovered vertex to discovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

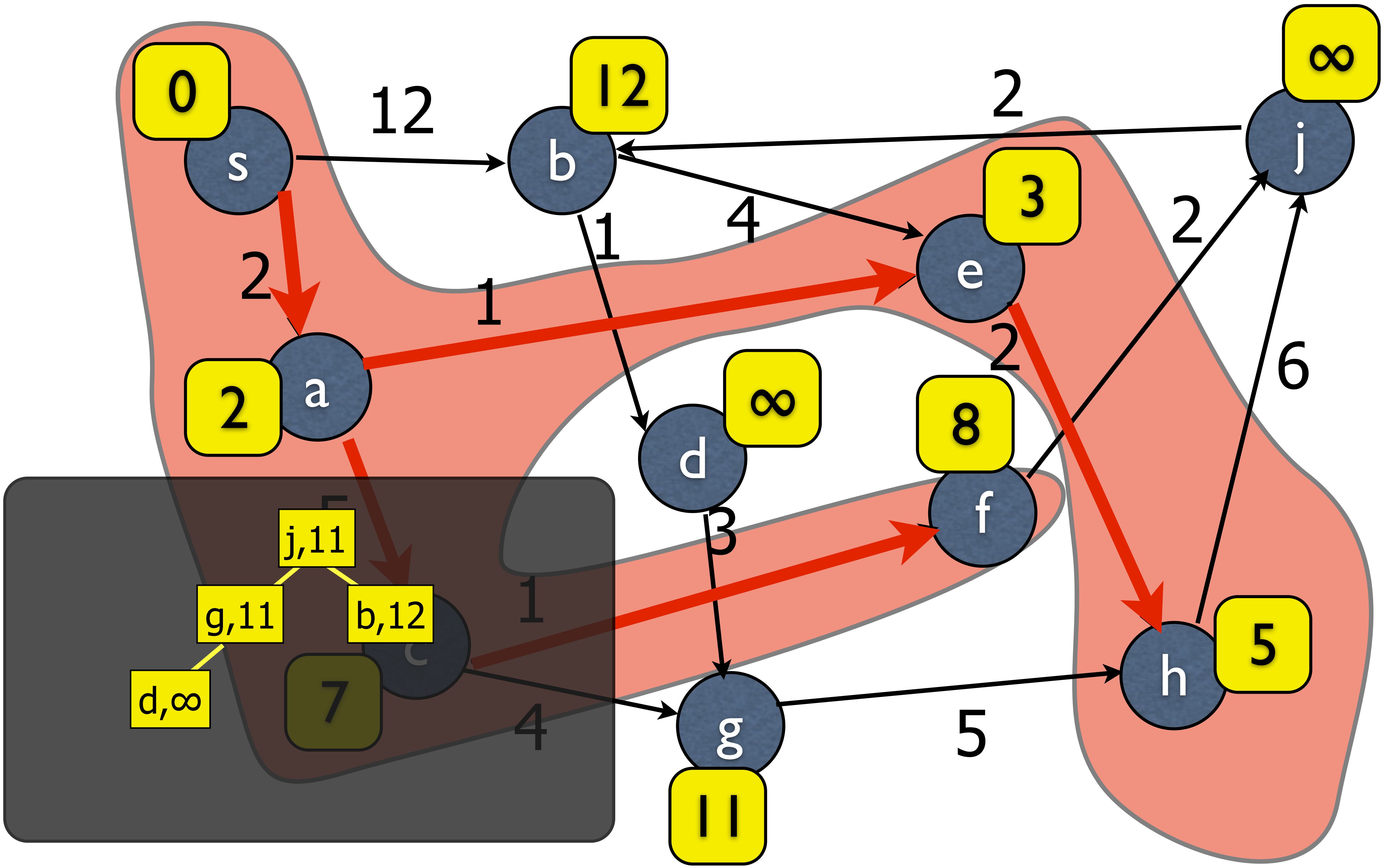


while there is edge from undiscovered vertex to discovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

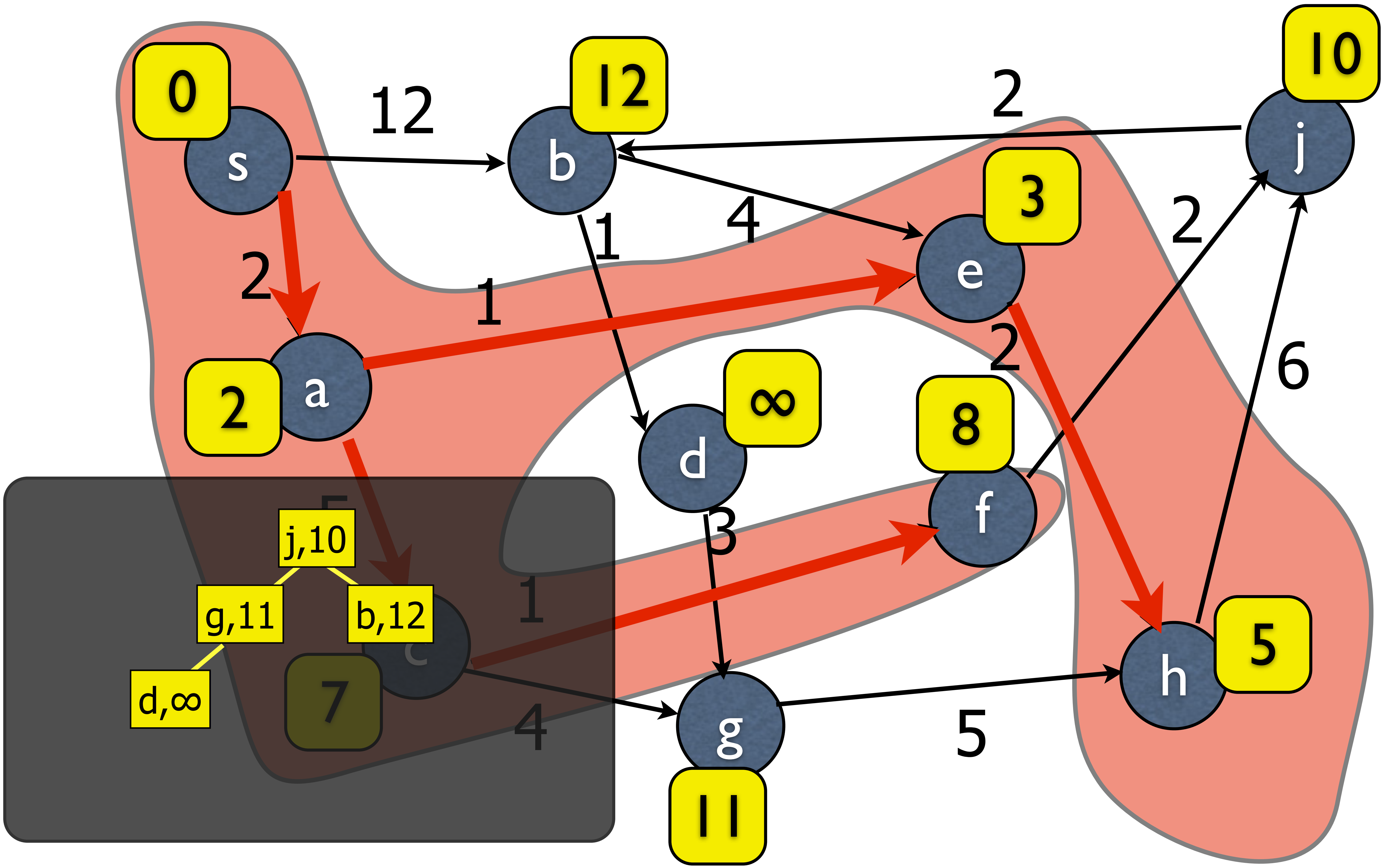


while there is edge from undiscovered vertex to discovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

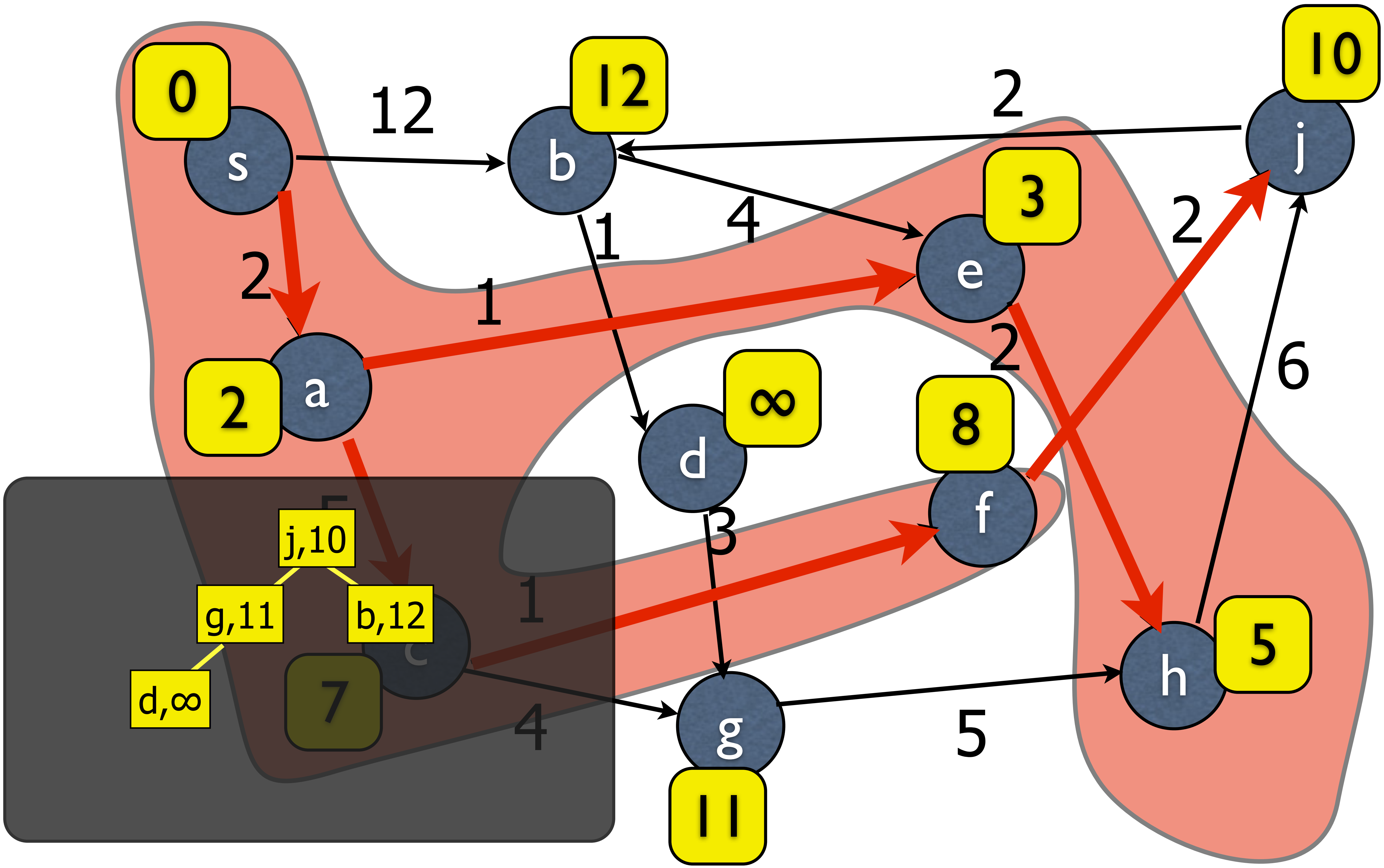


while there is edge from undiscovered vertex to discovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

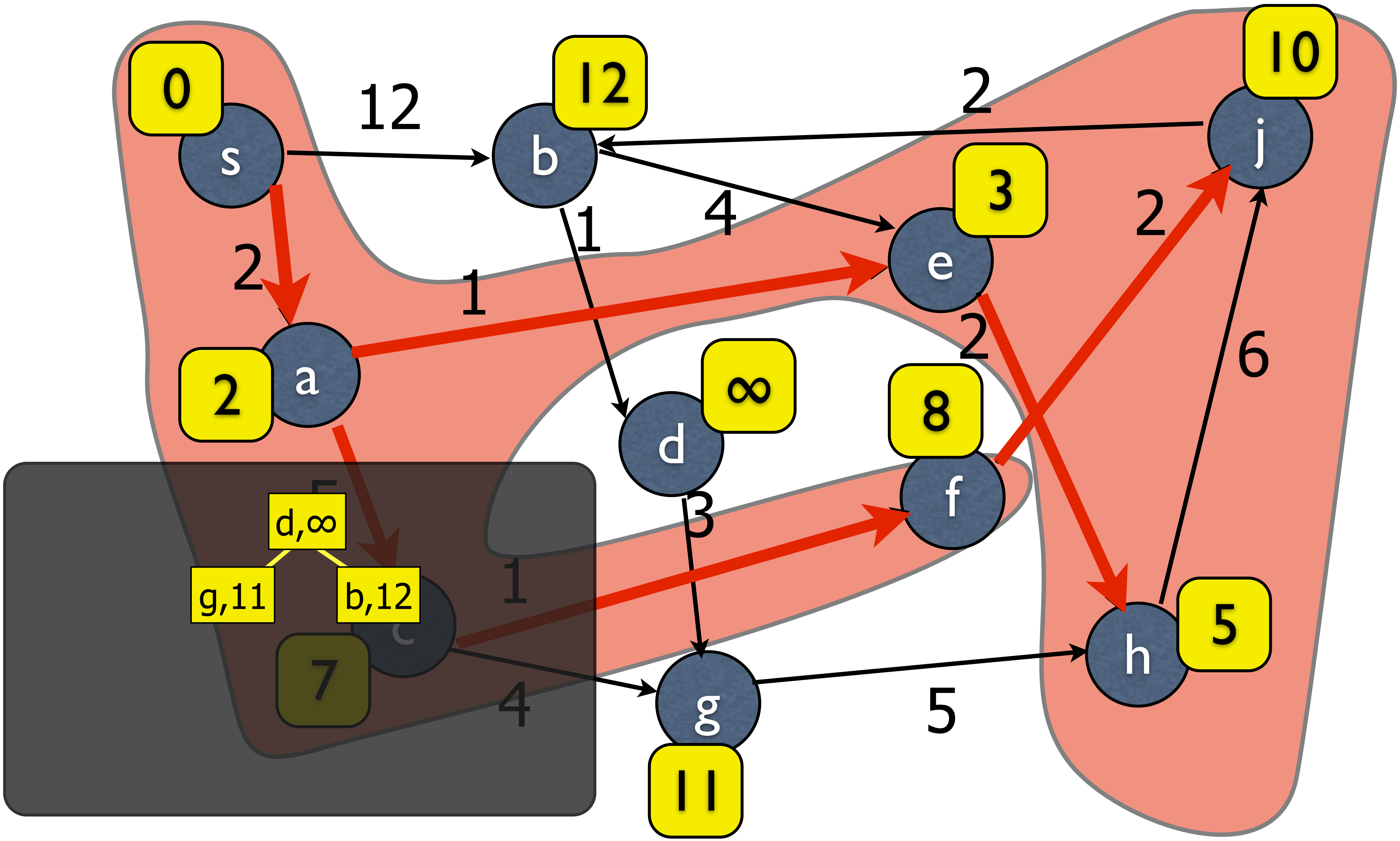


while there is edge from undiscovered vertex to discovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

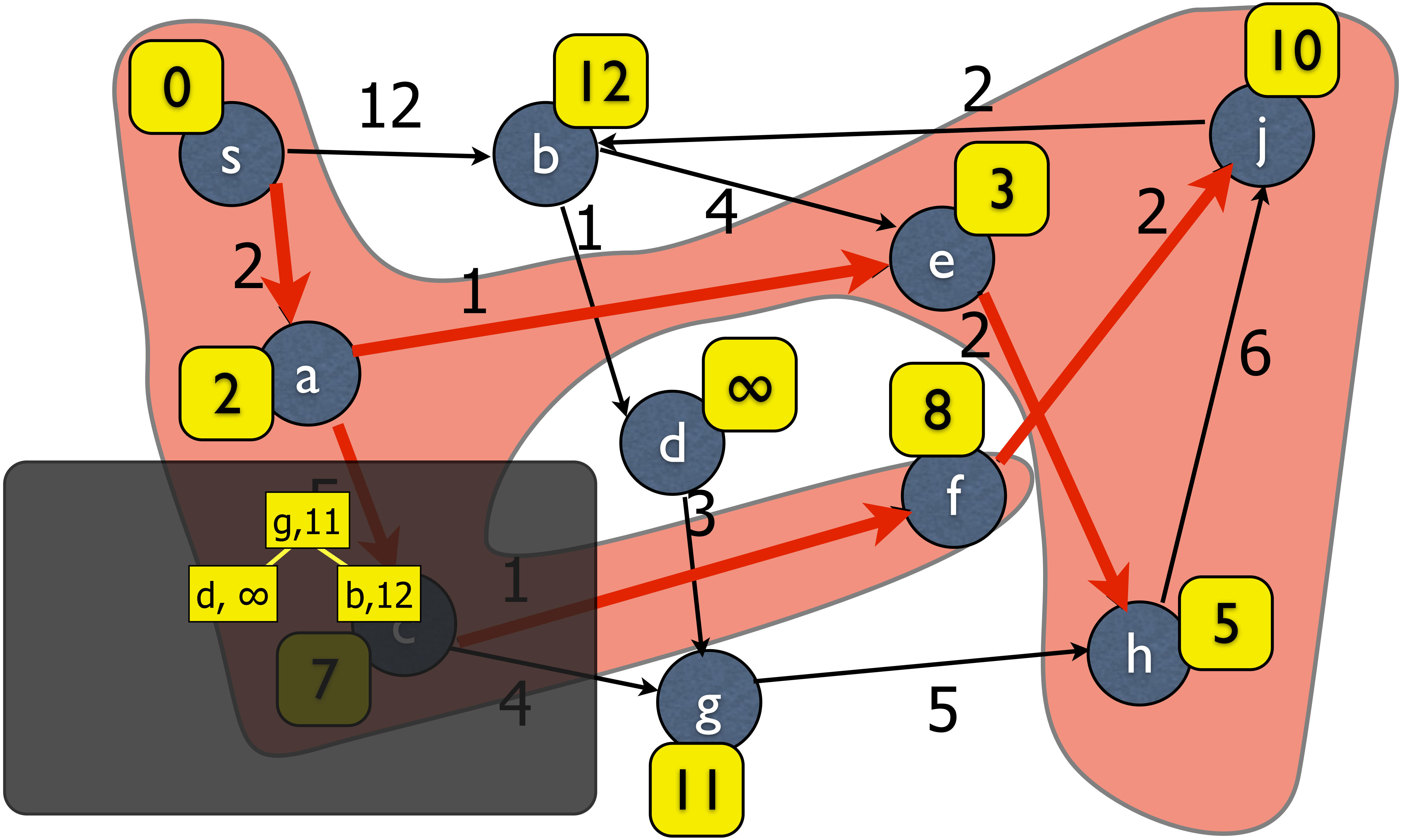
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra’s Algorithm

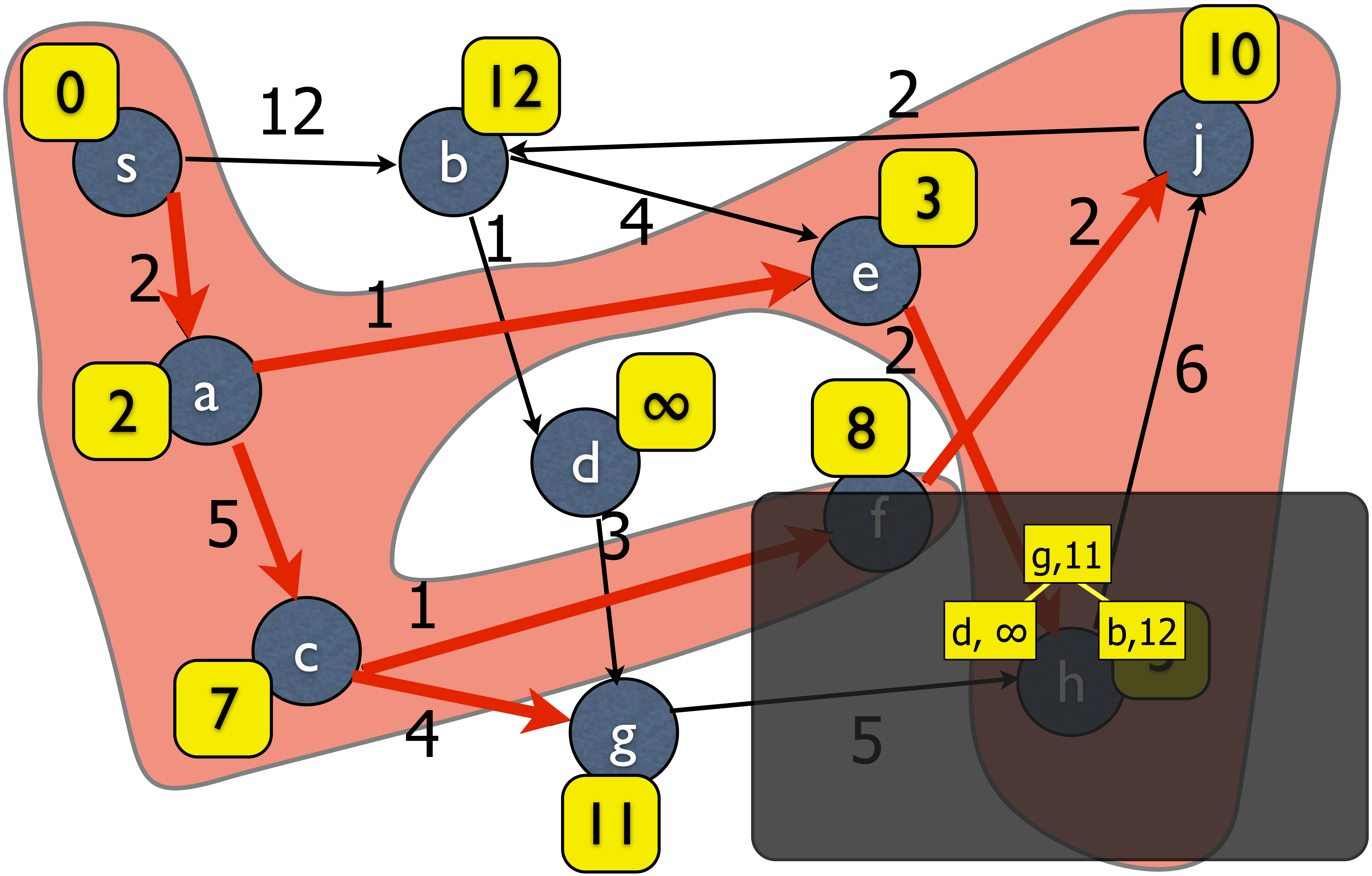


while there is edge from undiscovered vertex to discovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

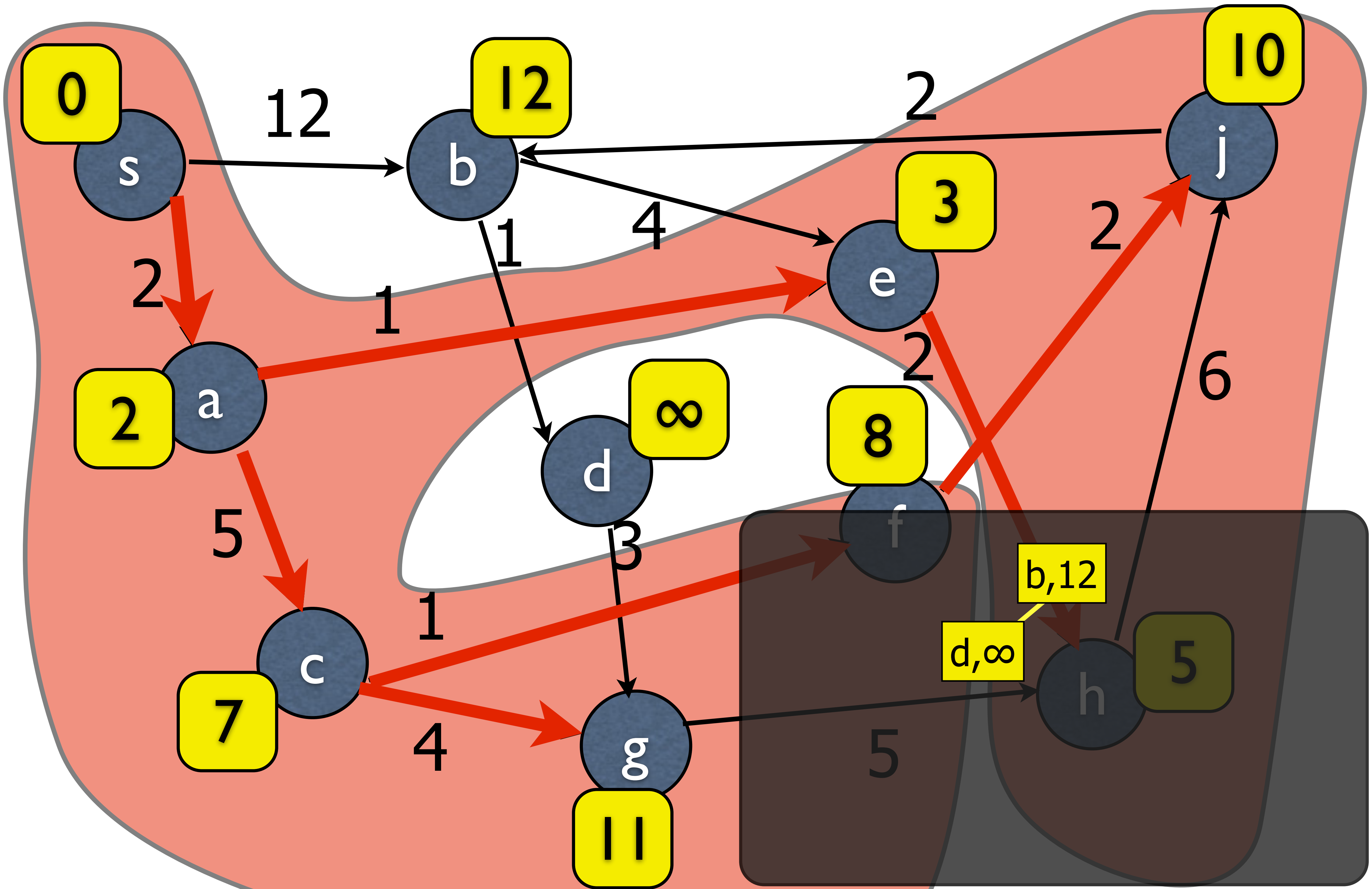
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



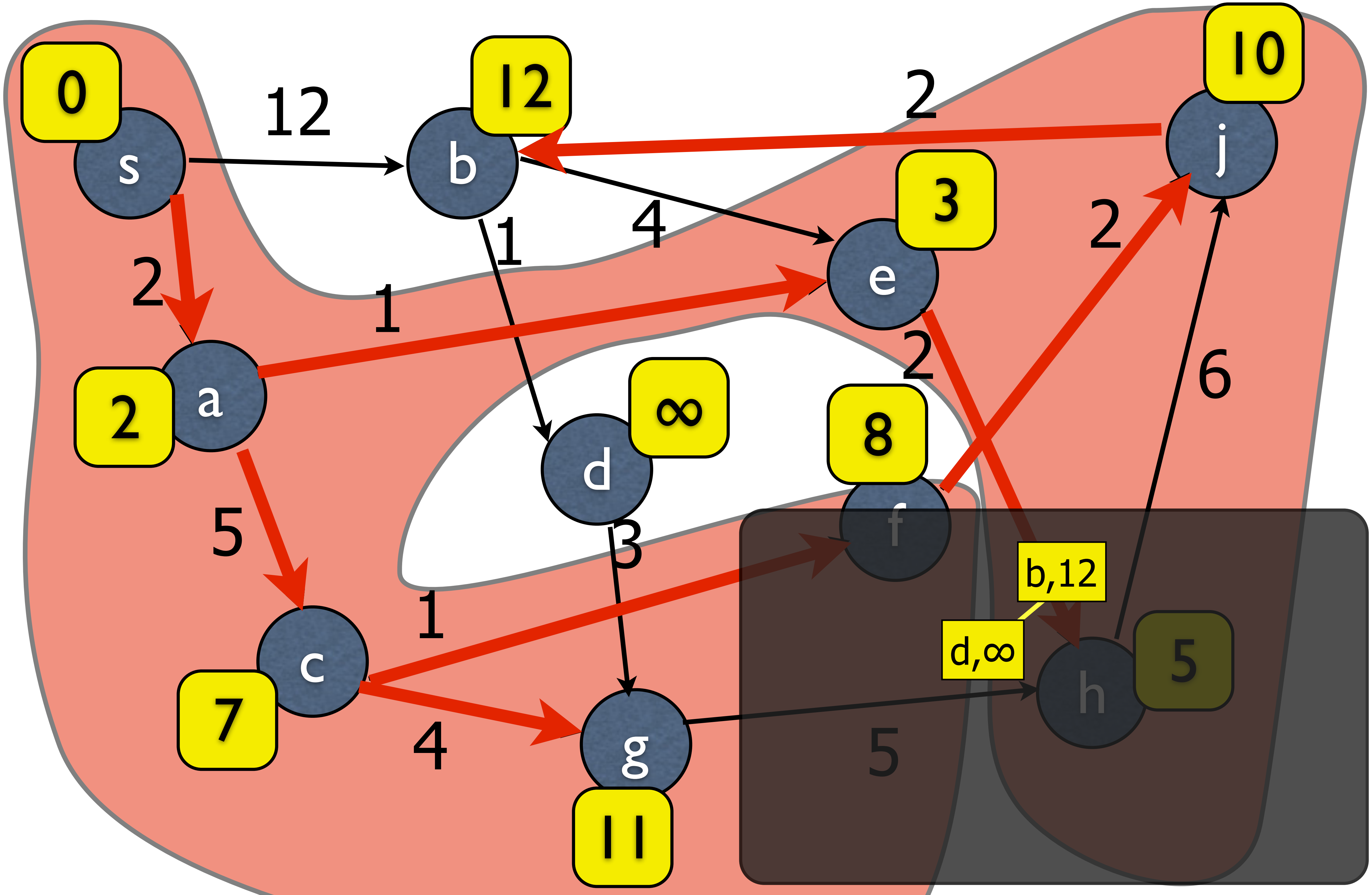
while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



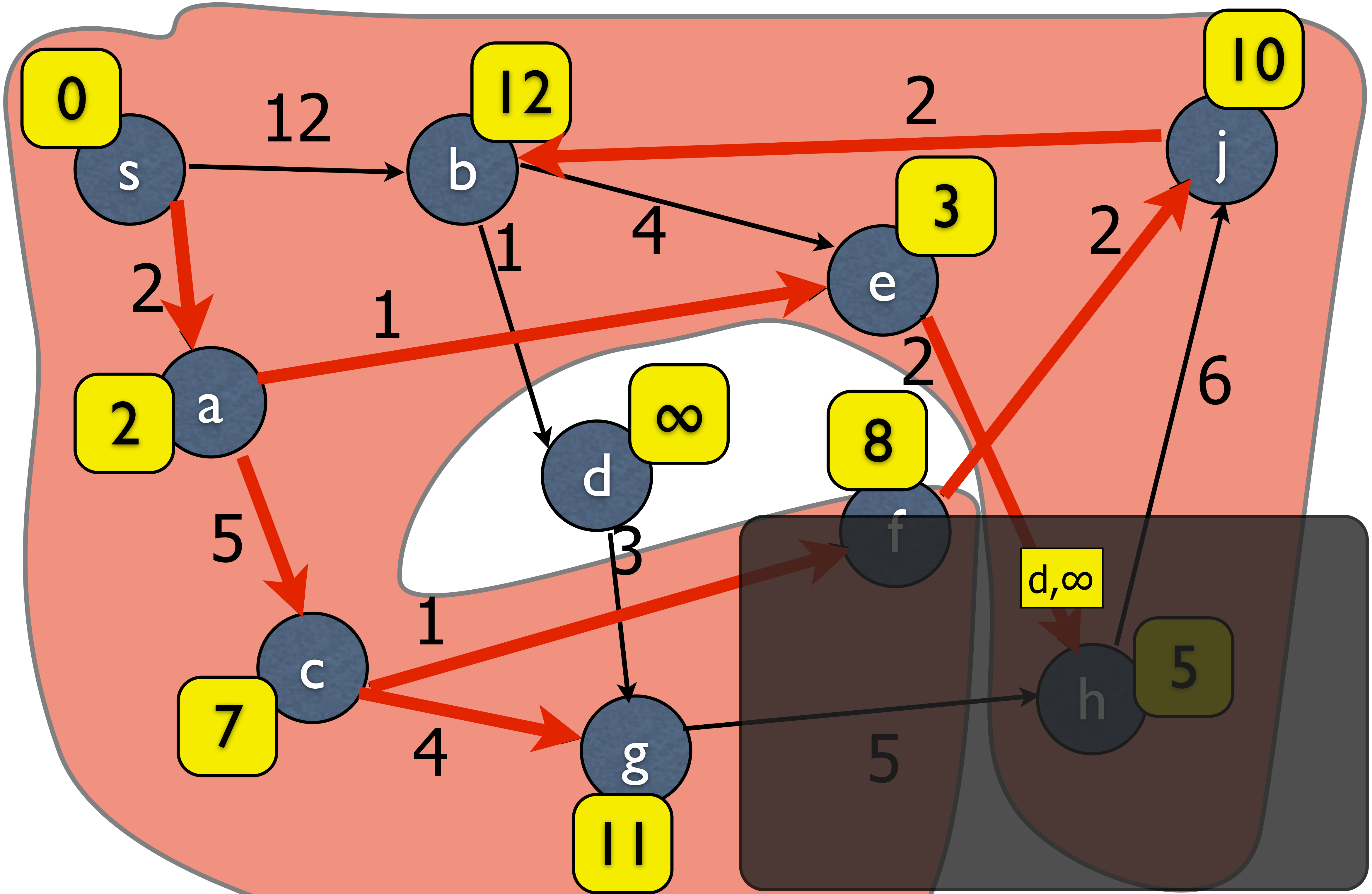
while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



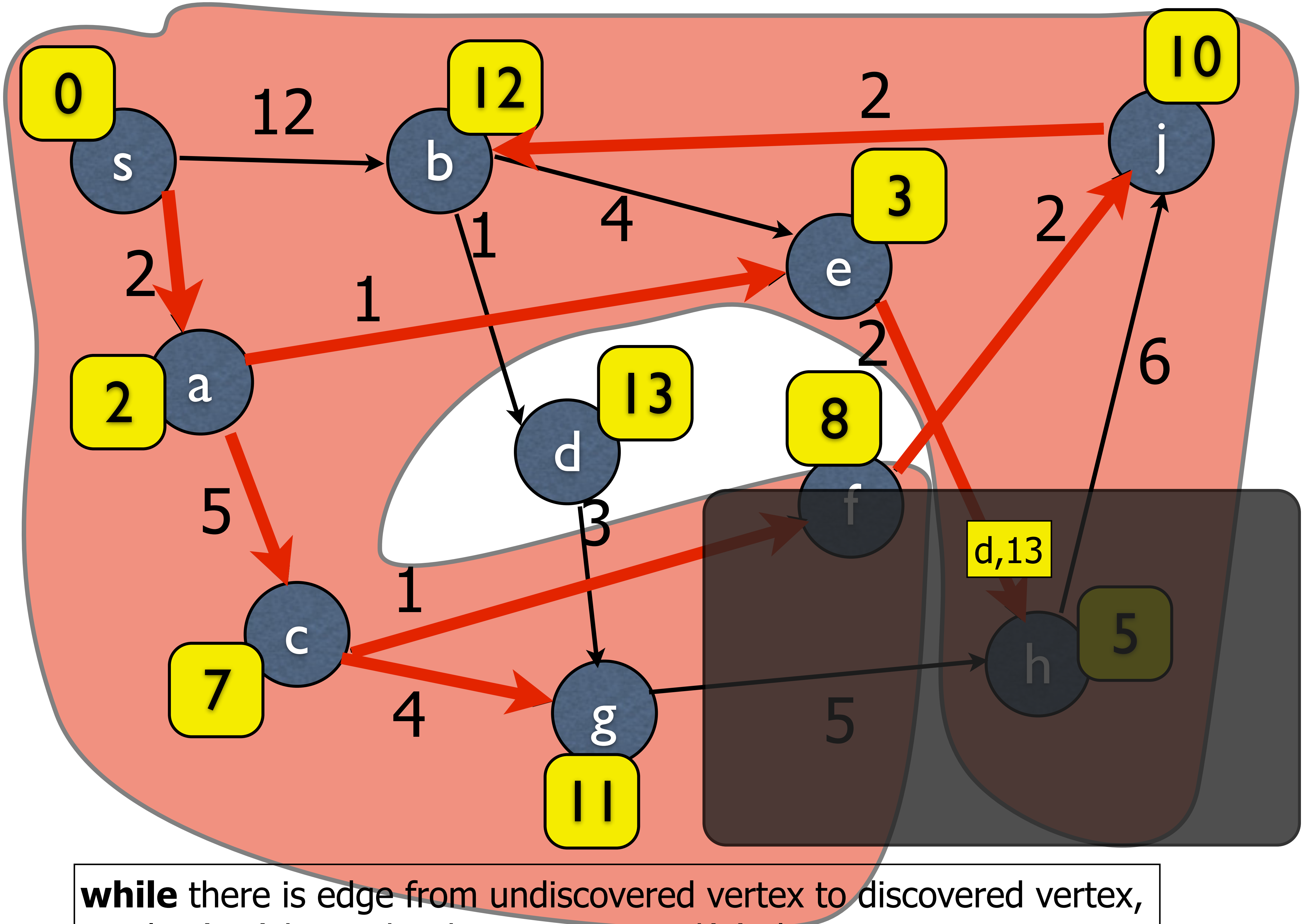
while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



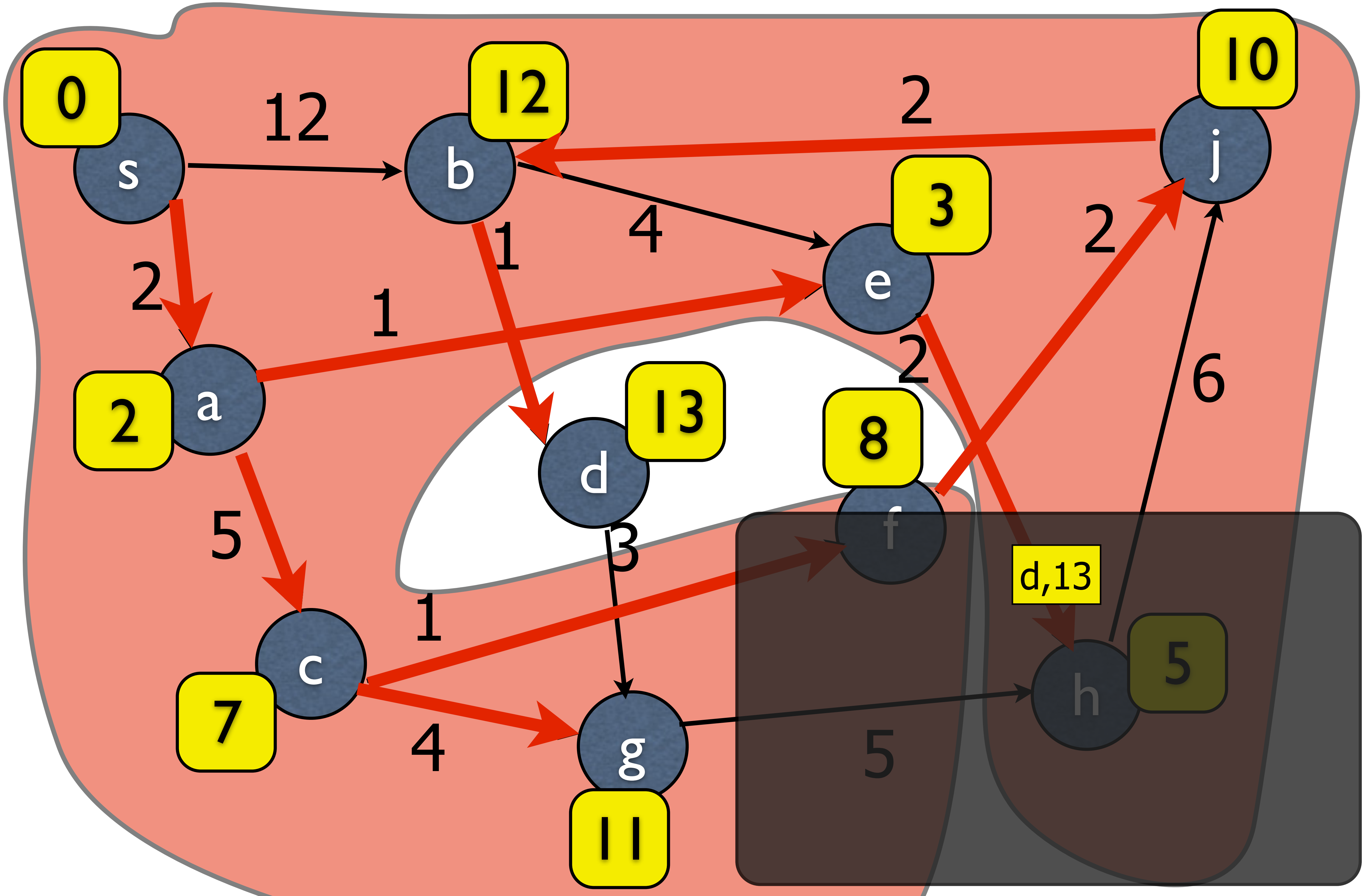
while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



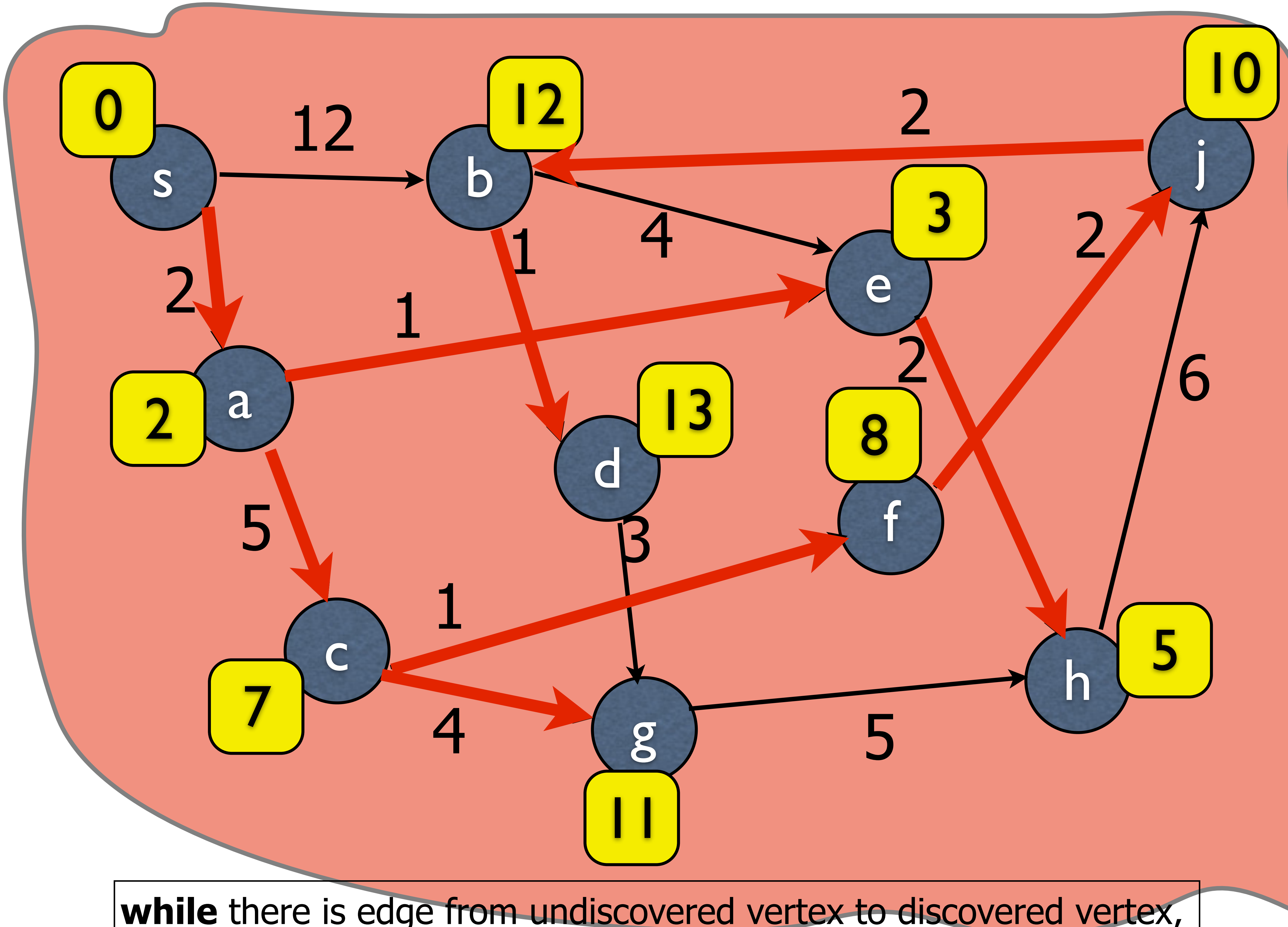
while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm

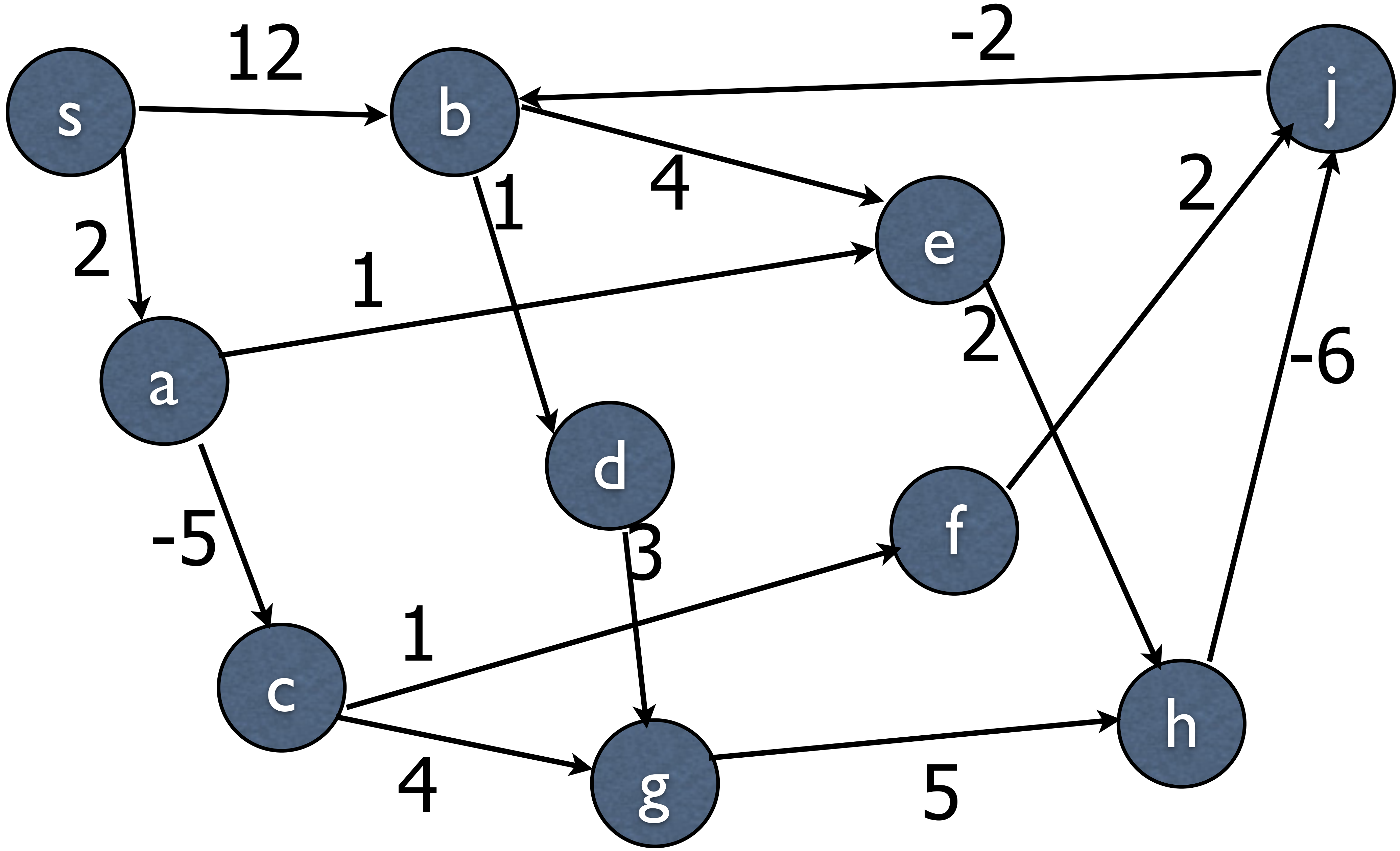


while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

Dijkstra's Algorithm



while there is edge from undiscovered vertex to discovered vertex,
let (u,v) be such edge minimizing $d(u)+l_{u,v}$
set $d(v) = d(u) + l_{u,v}$, mark v discovered

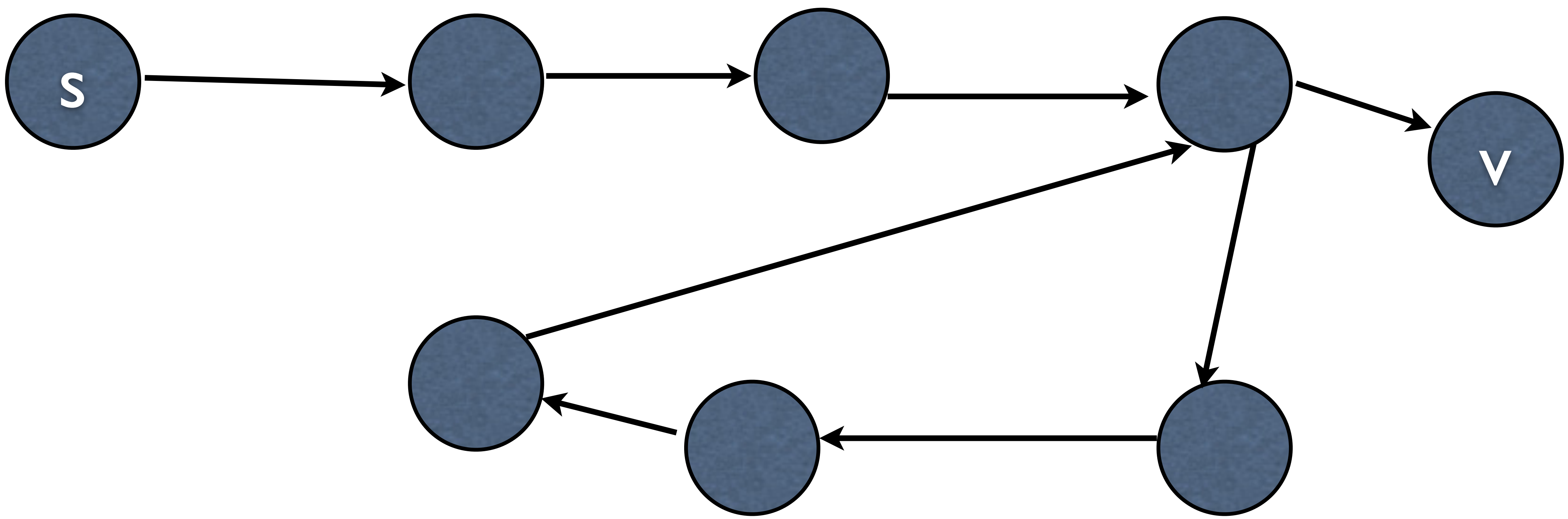


What about negative edge weights?

Assume no negative cycles.

Claim: If graph has no negative length cycles, then shortest walk between (s,v) has length at most n-1.

Pf: Suppose not. Then by pigeonhole, the shortest walk must contain a cycle! Removing it gives a shorter walk. Contradiction.



<u>Bellman-Ford</u>

For all vertices set d(v)= ∞

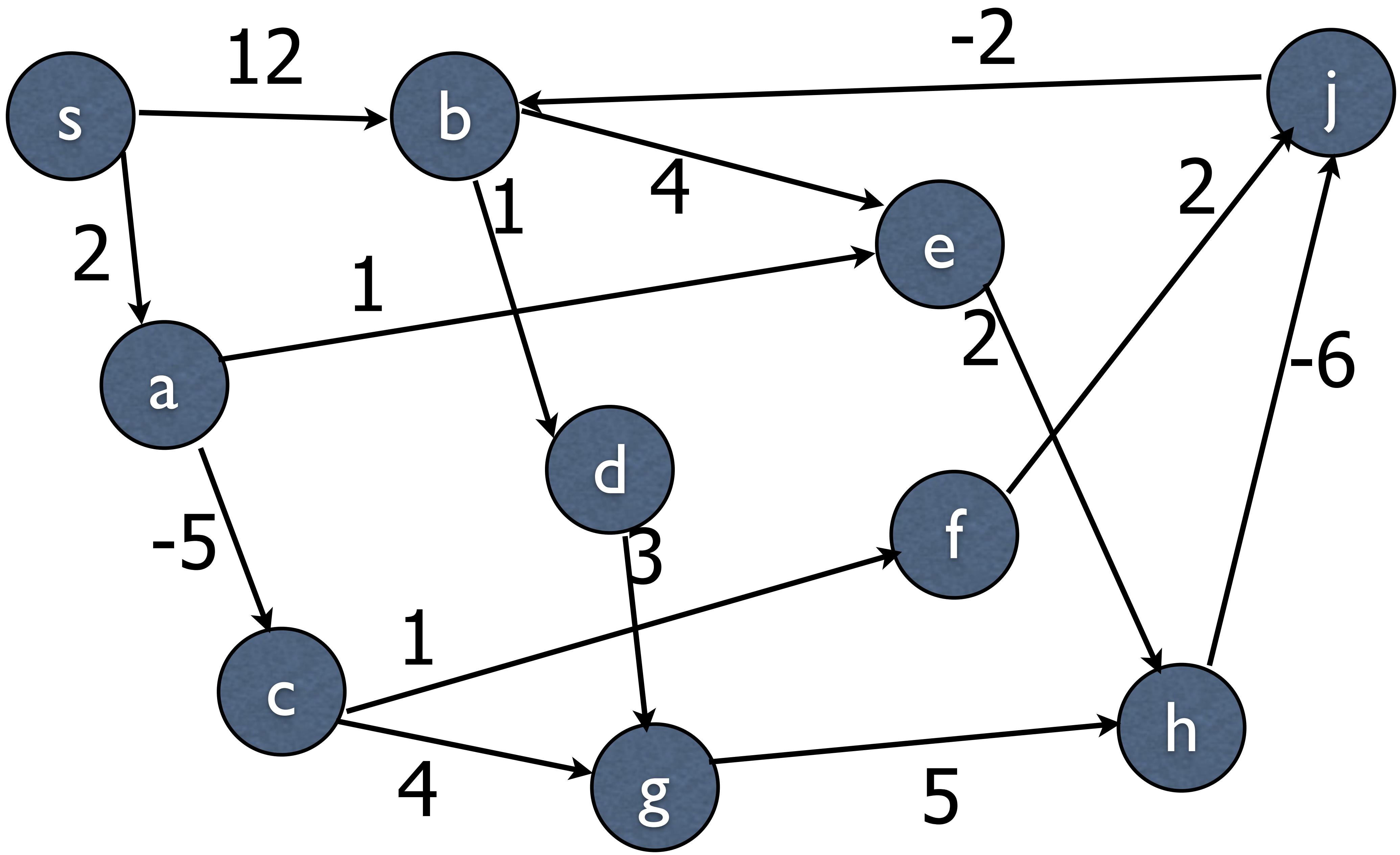
Set d(s) = 0

for i=1,2,…,n-1

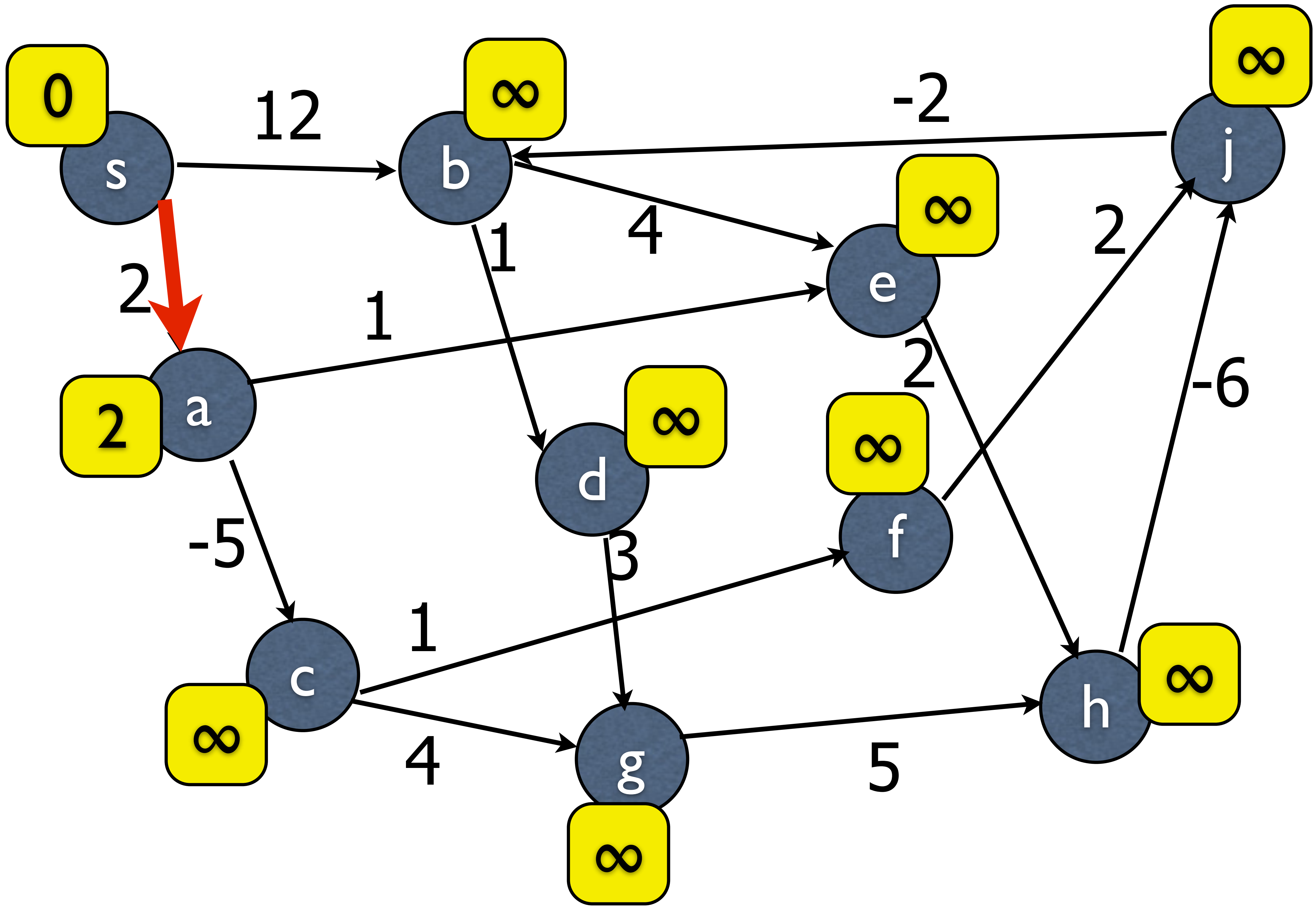
  for every edge (u,v)

    if d(v) > d(u) + $l_{u,v}$, update d(v) = d(u) + $l_{u,v}$.

Bellman-Ford



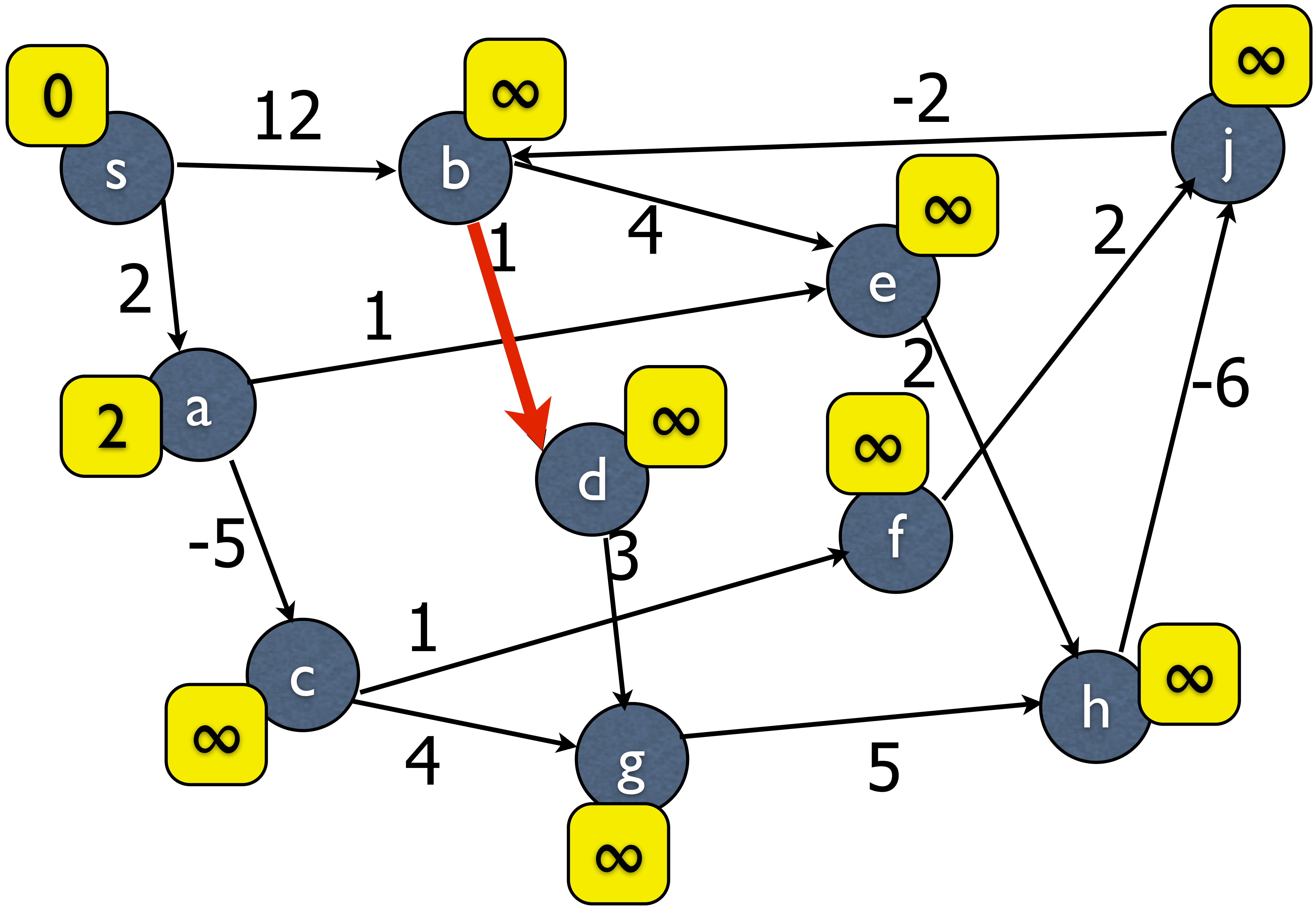
Bellman-Ford



update (u,v):

$$d(v) = \min\{d(v) + l_{(u,v)}\}$$

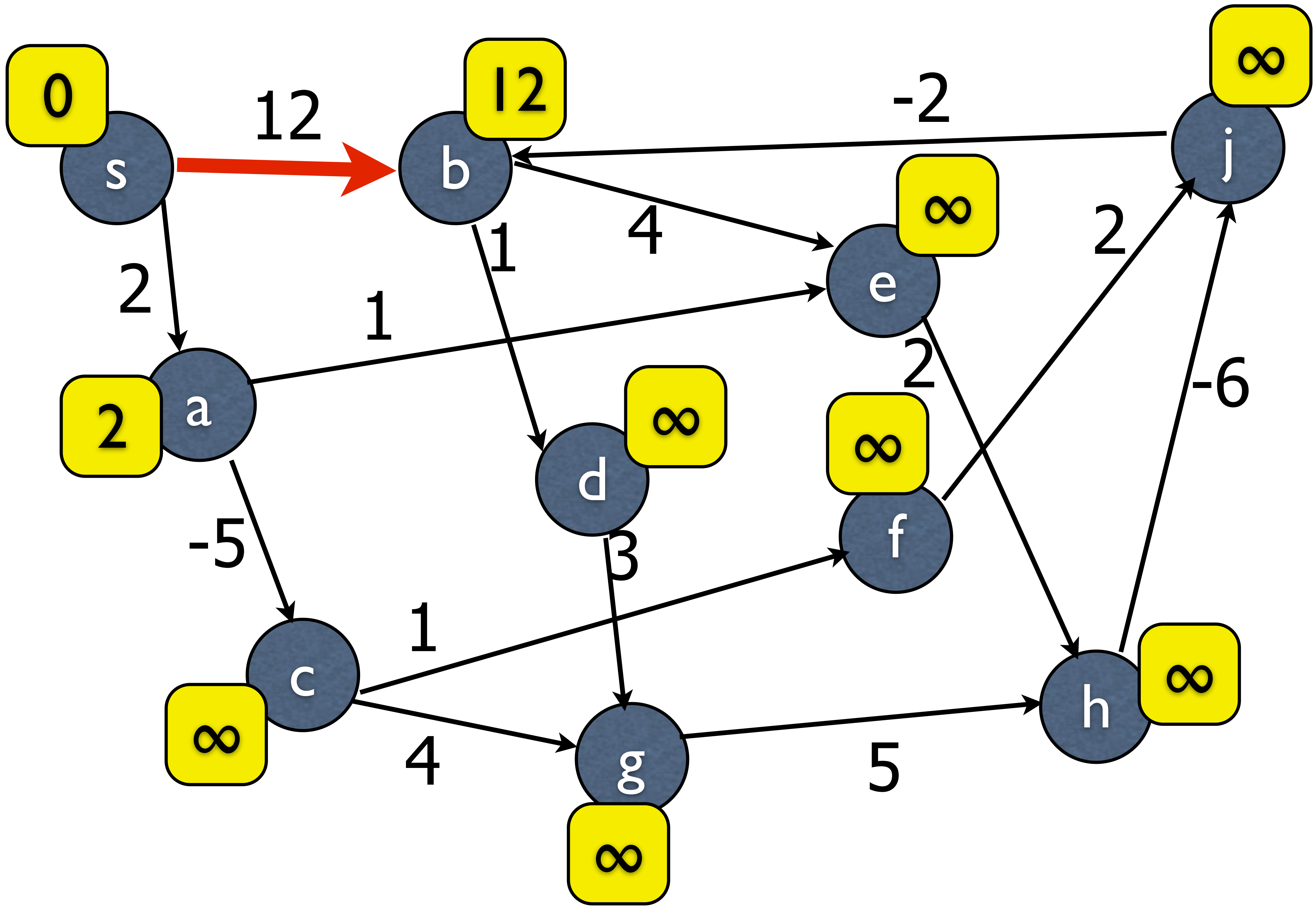
Bellman-Ford



update (u,v):

$d(v) = \min\{d(v) + l_{(u,v)}\}$

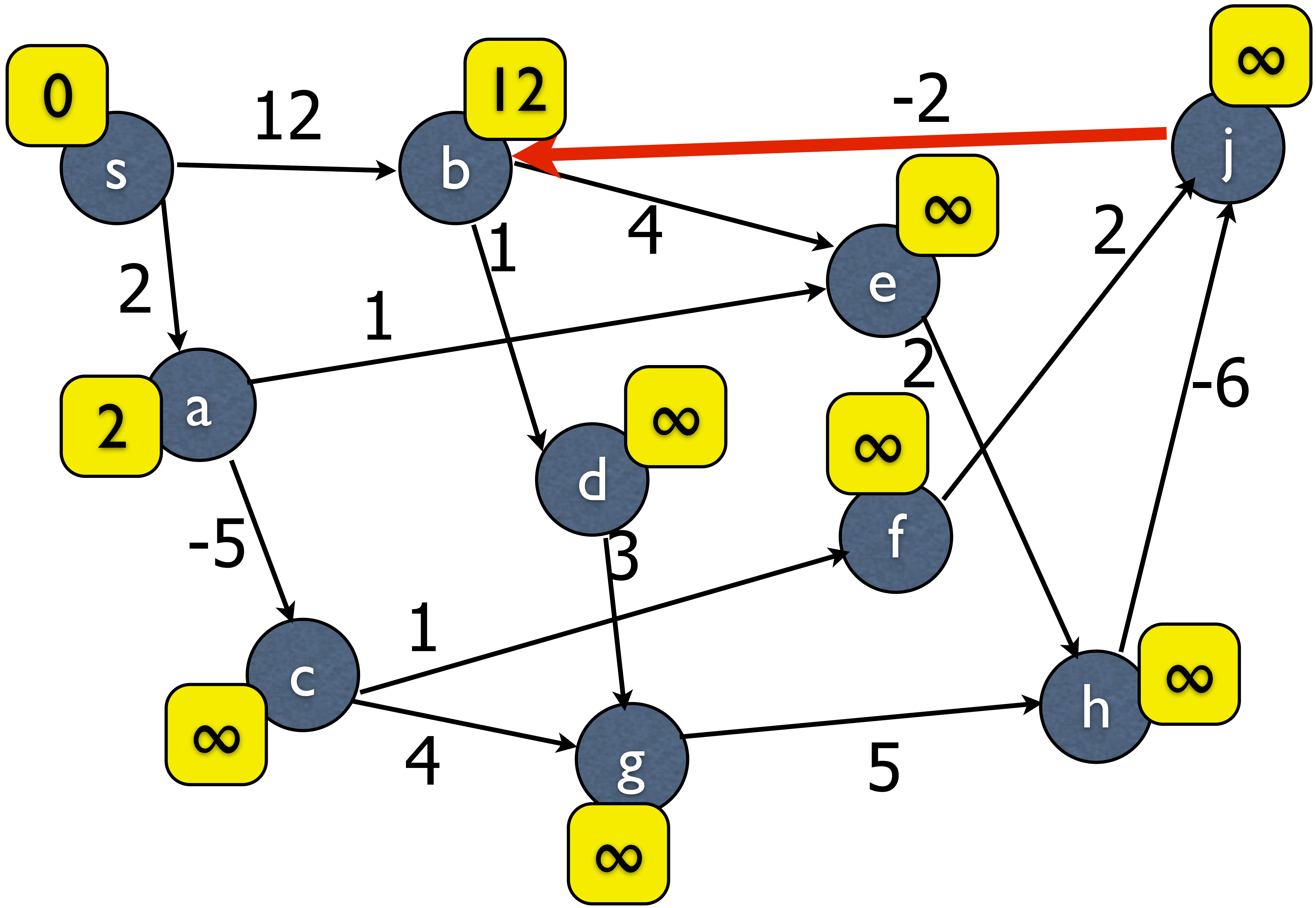
Bellman-Ford



update (u,v):

$$d(v) = \min\{d(v) + l_{(u,v)}\}$$

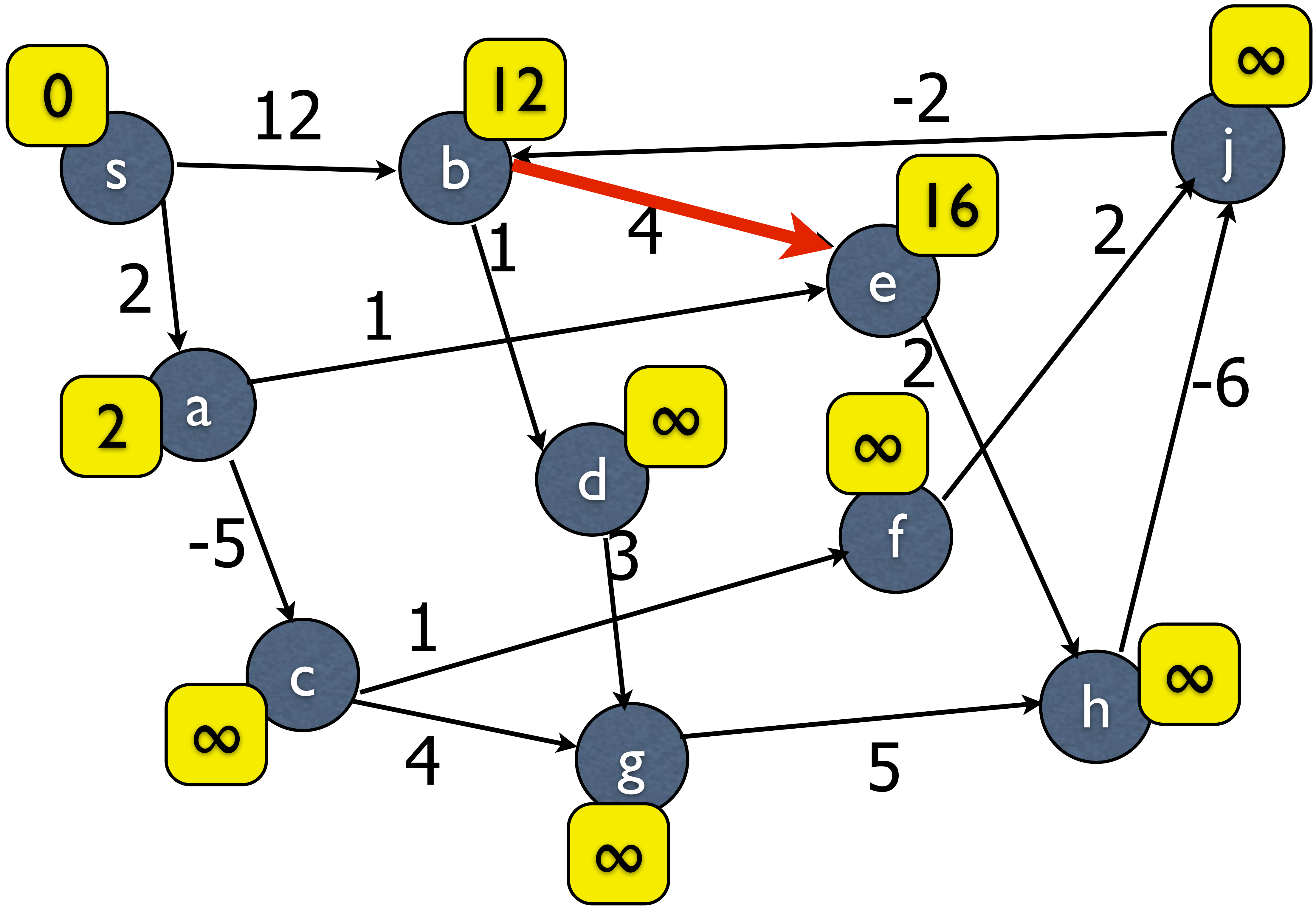
Bellman-Ford



update (u,v):

d(v) = min{d(v) + $l_{(u,v)}$}

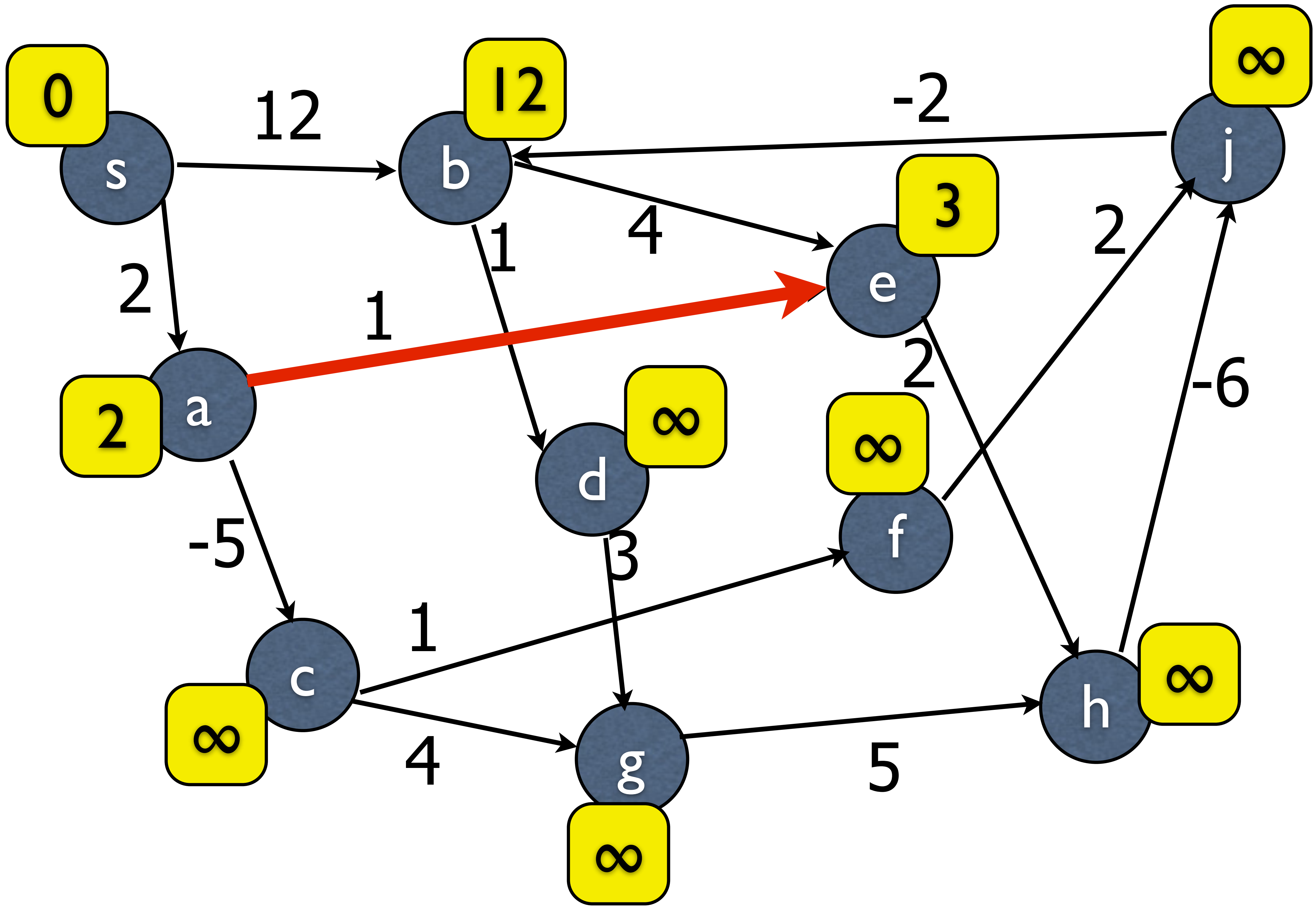
Bellman-Ford



update (u,v):

$$d(v) = \min\{d(v) + l_{(u,v)}\}$$

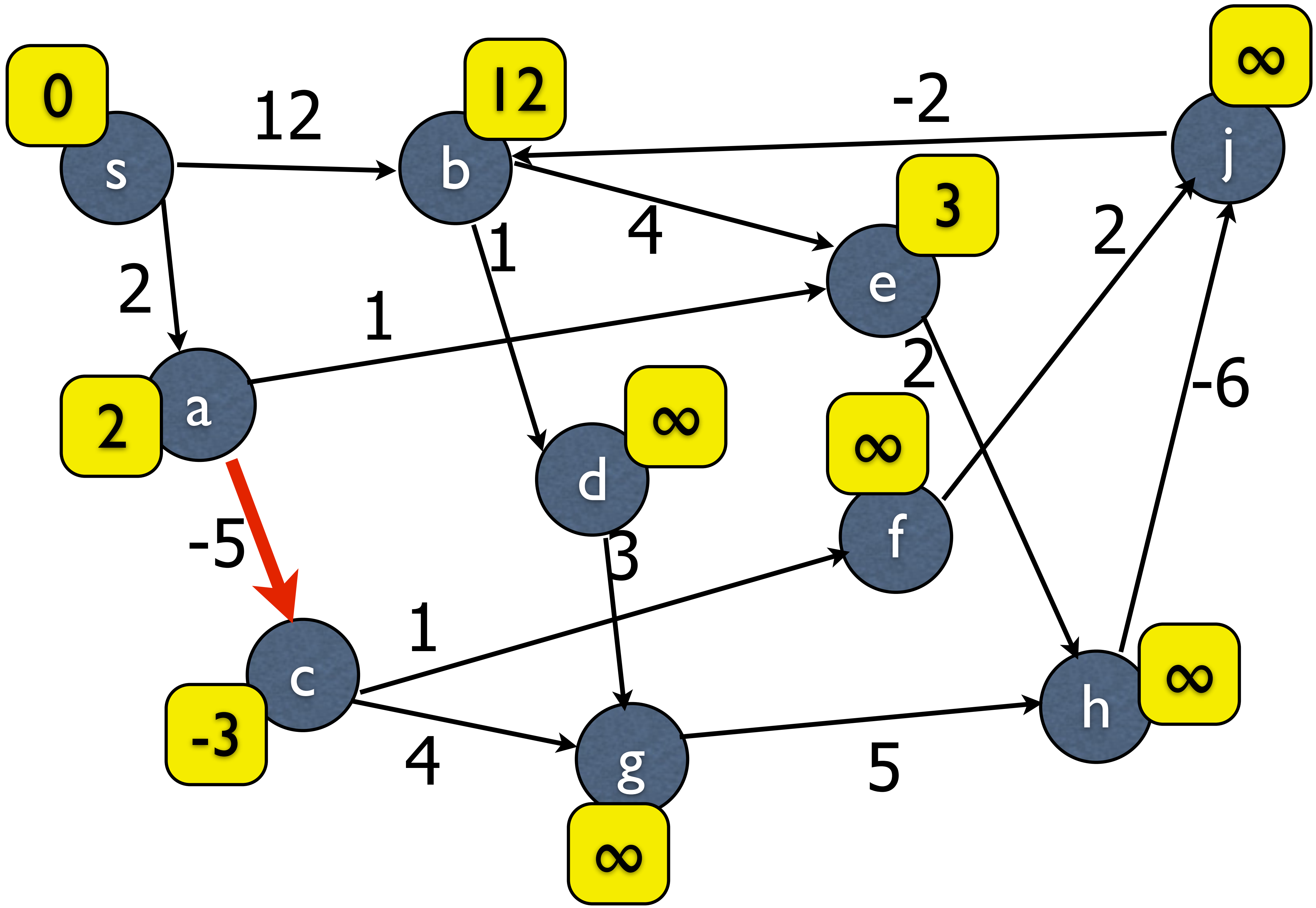
Bellman-Ford



update (u,v):

$$d(v) = \min\{d(v) + l_{(u,v)}\}$$

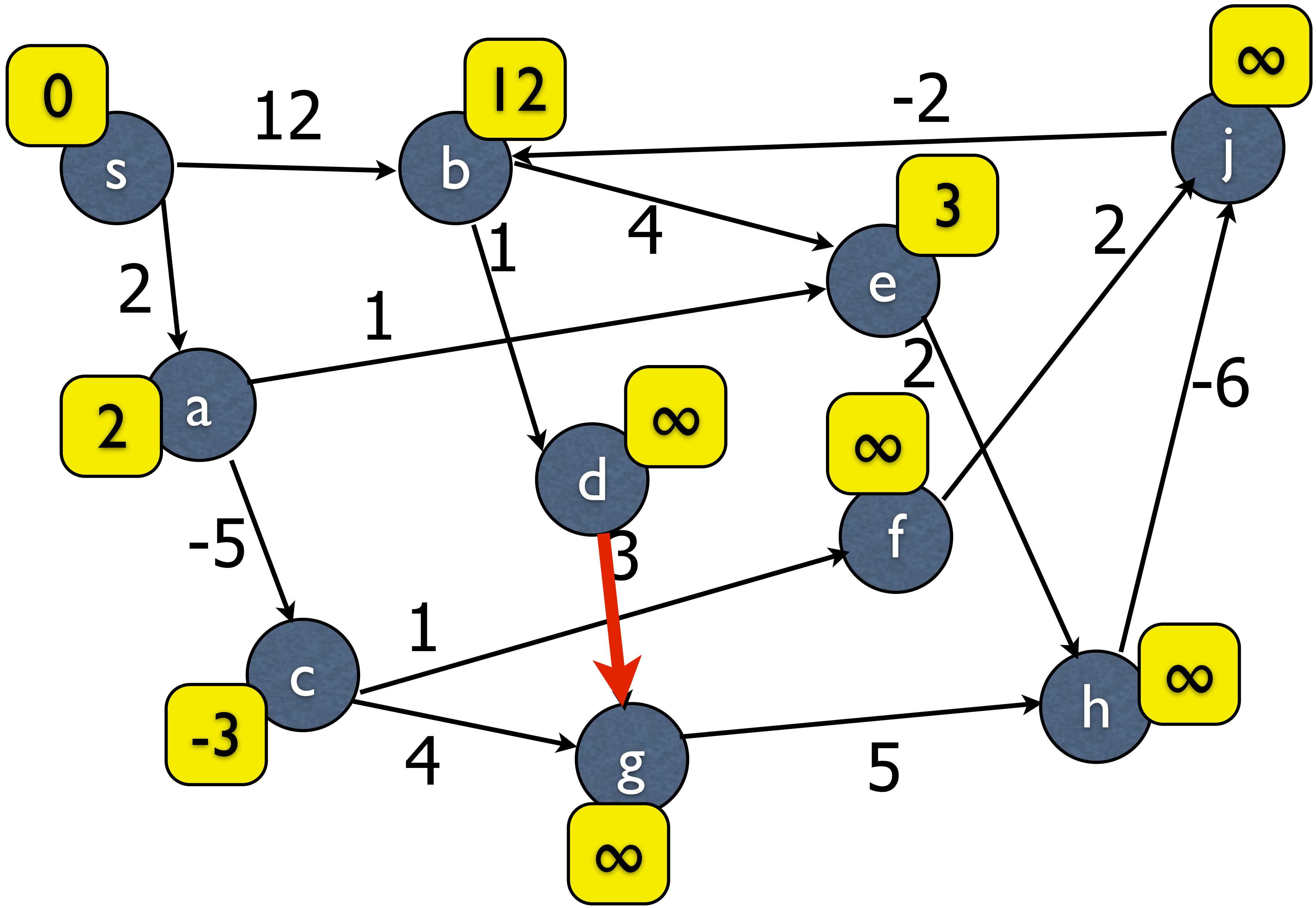
Bellman-Ford



update (u,v):

$d(v) = \min\{d(v) + l_{(u,v)}\}$

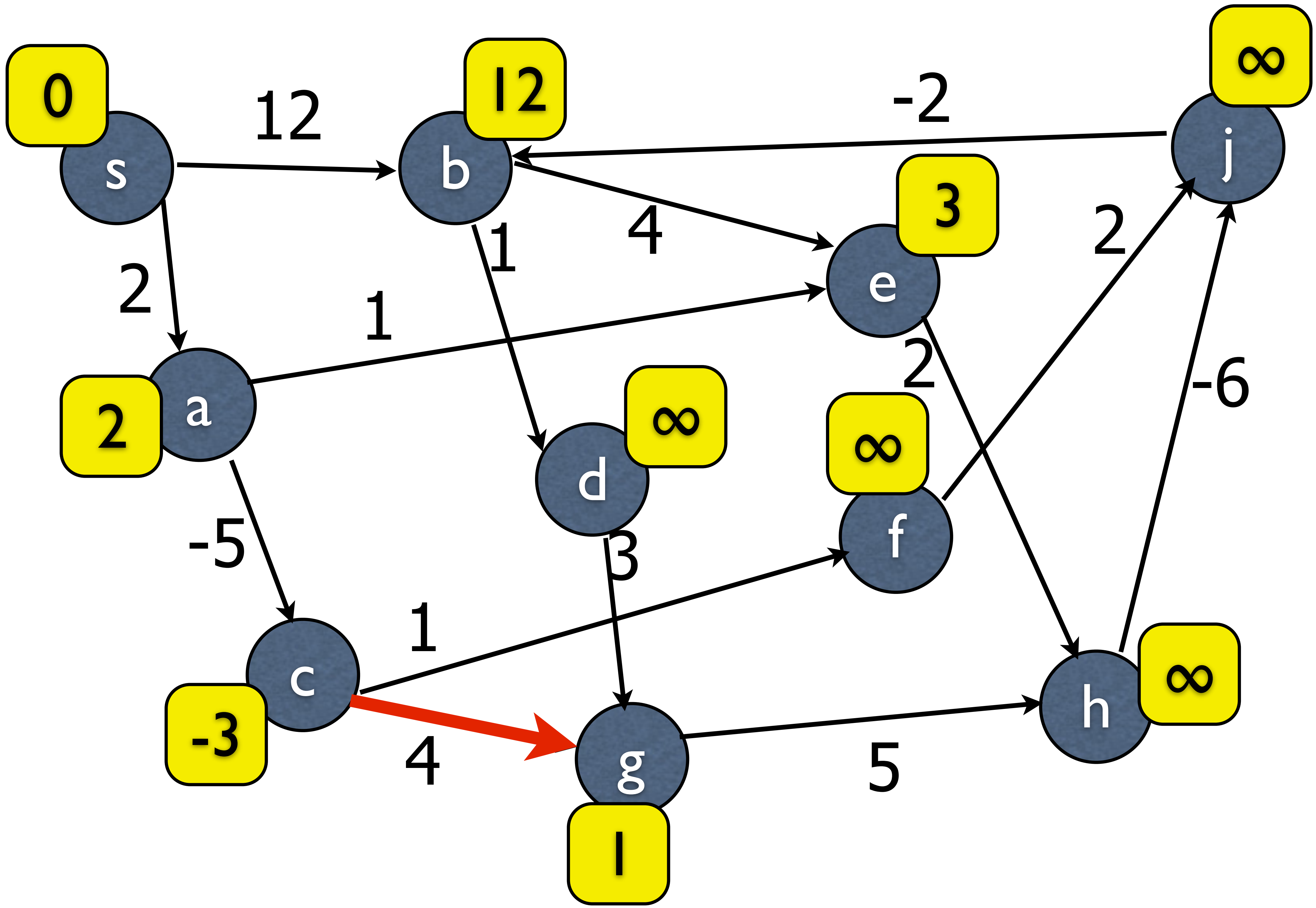
Bellman-Ford



update (u,v):

d(v) = min{d(v) + $l_{(u,v)}$}

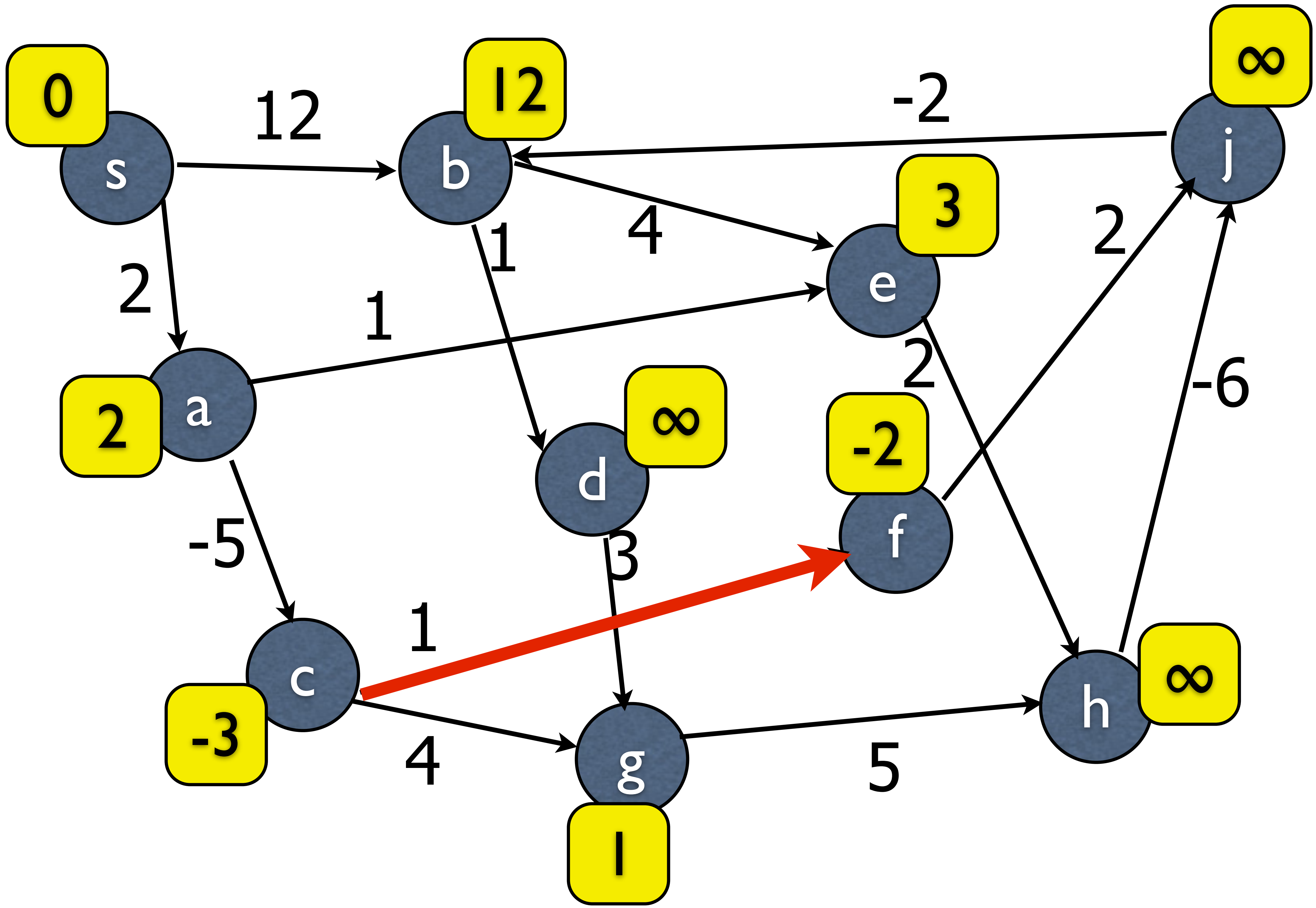
Bellman-Ford



update (u,v):

$$d(v) = \min\{d(v) + l_{(u,v)}\}$$

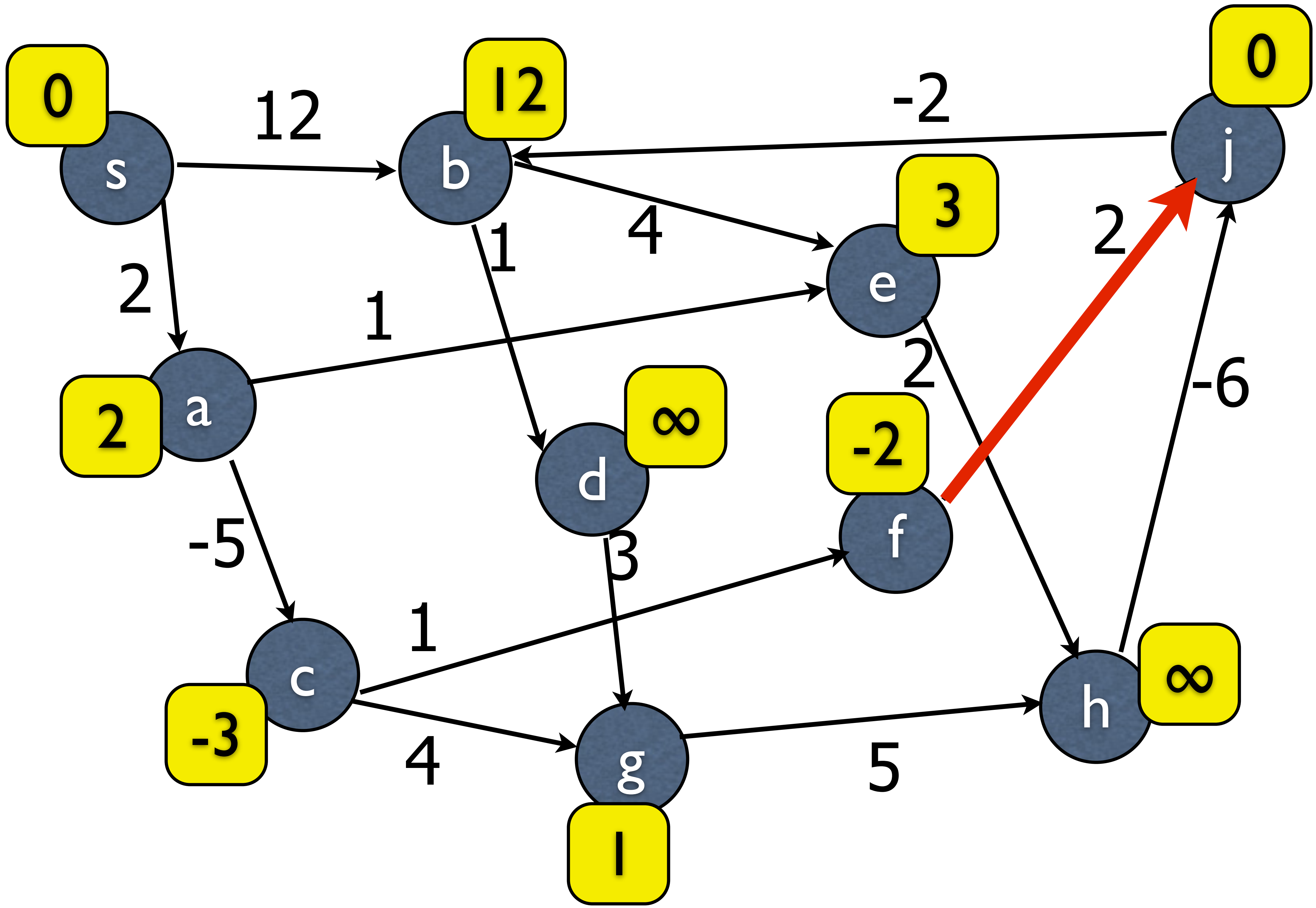
Bellman-Ford



update (u,v):

d(v) = min{d(v) + $l_{(u,v)}$}

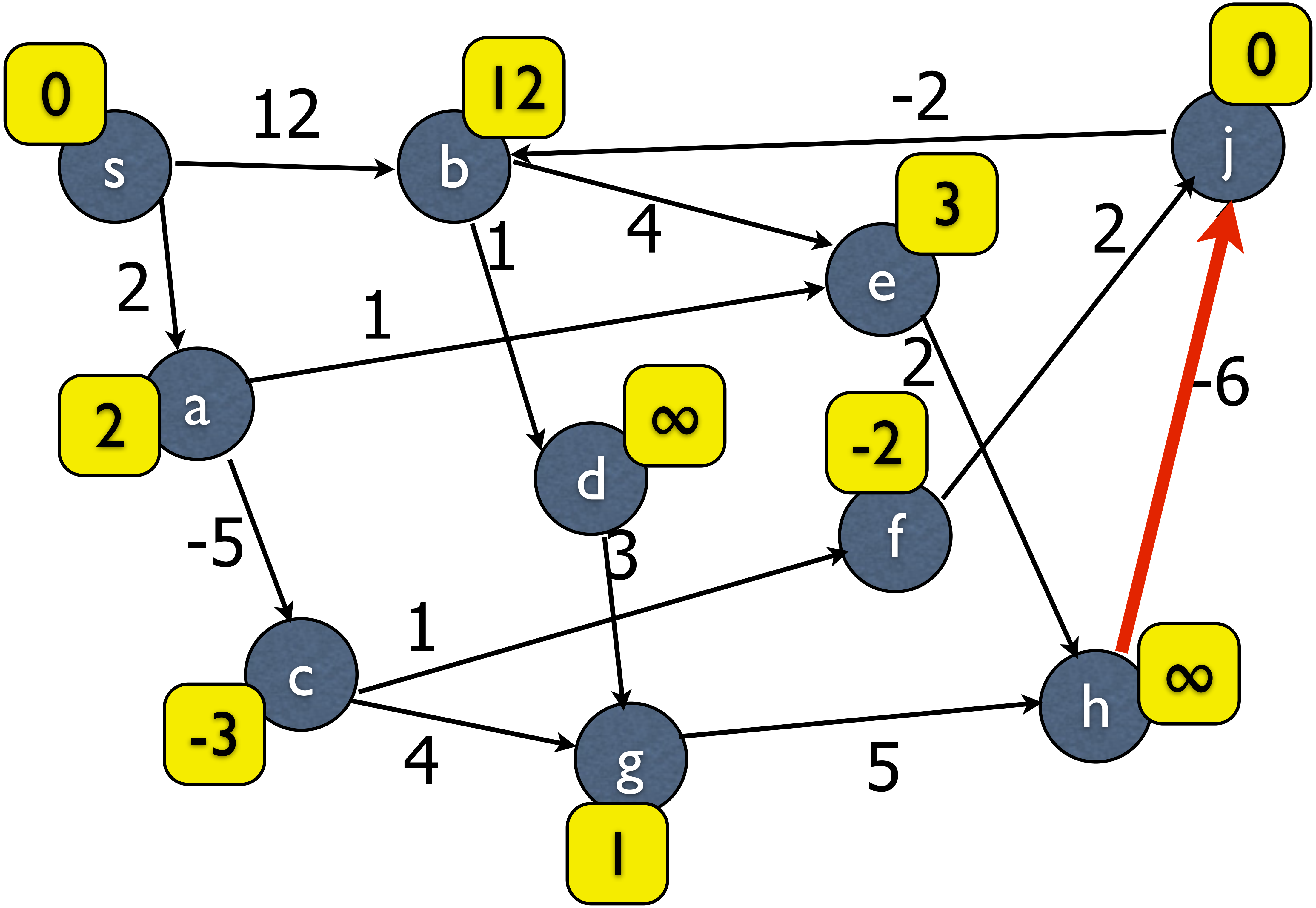
Bellman-Ford



update (u,v):

d(v) = min{d(v) + l$_{(u,v)}$}

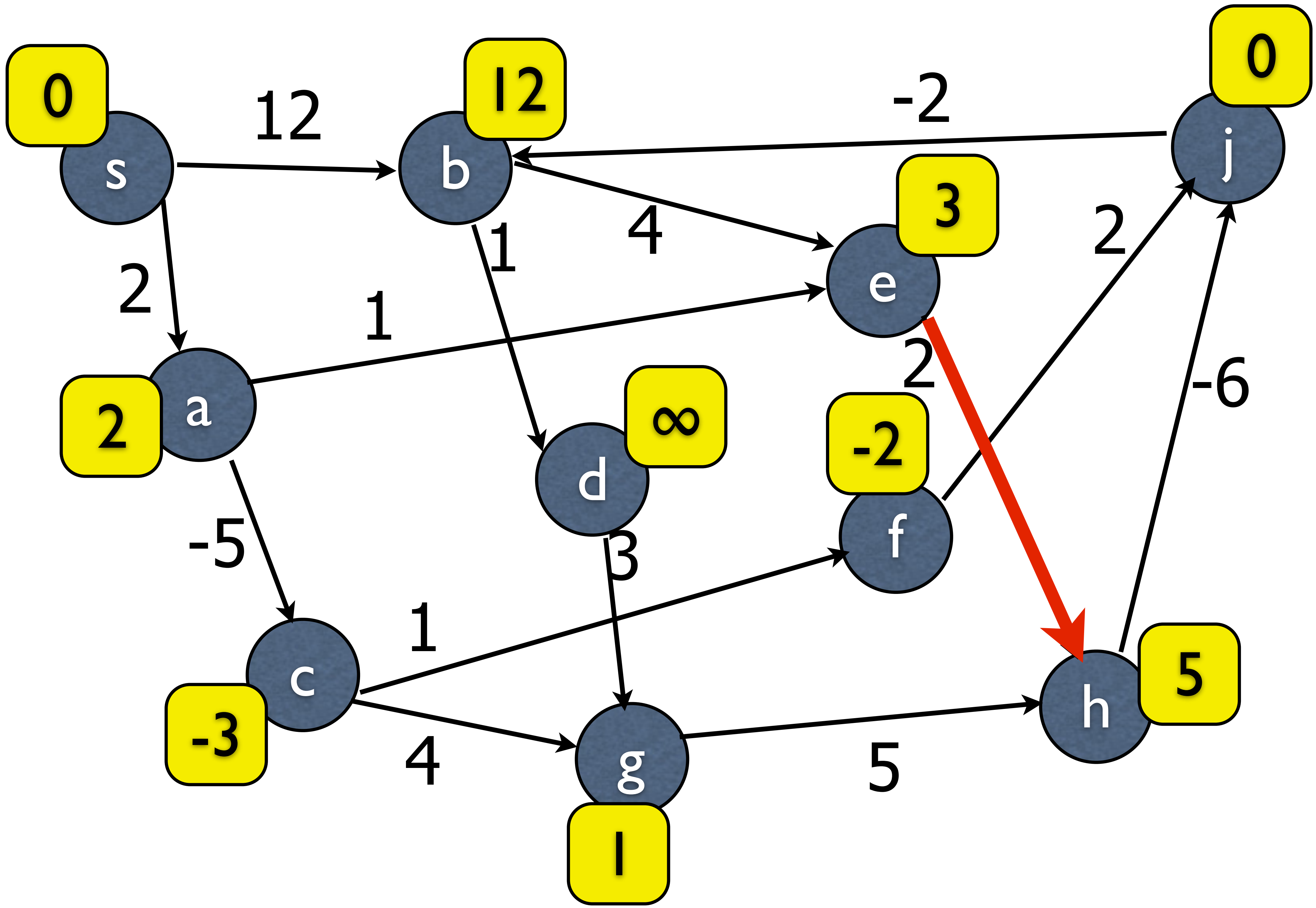
Bellman-Ford



update (u,v):

$$d(v) = \min\{d(v) + l_{(u,v)}\}$$

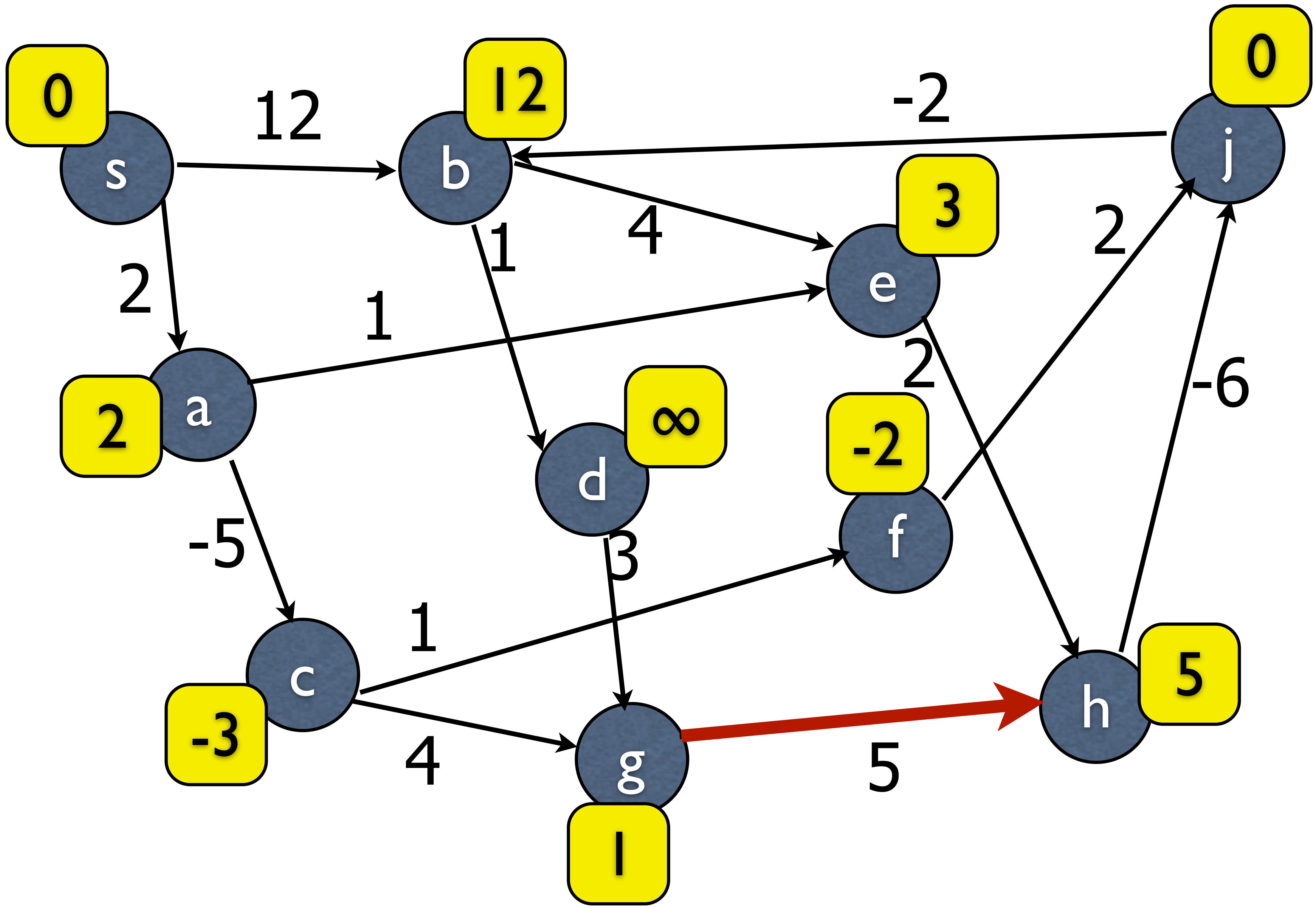
Bellman-Ford



update (u,v):

d(v) = min{d(v) + $l_{(u,v)}$}

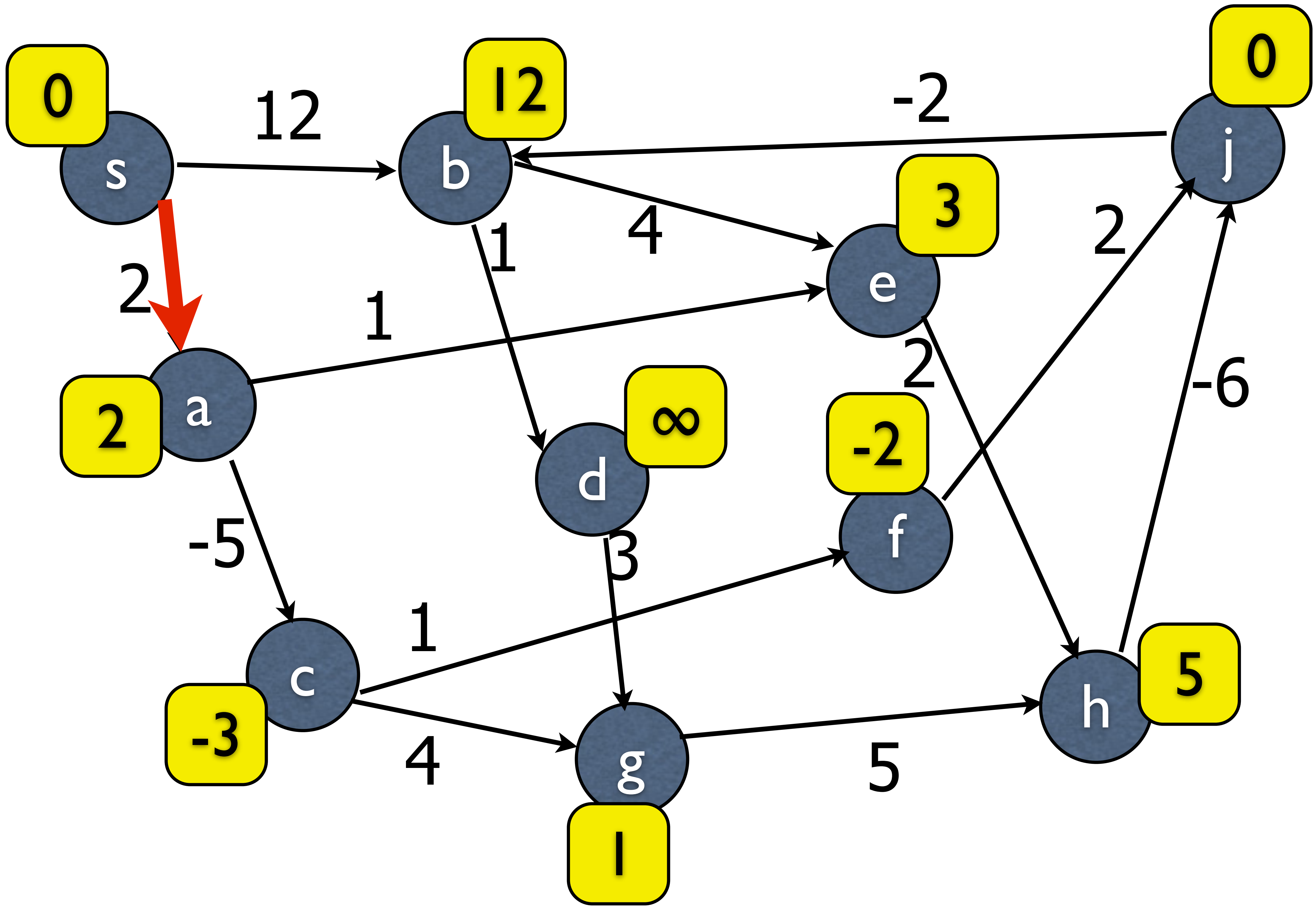
Bellman-Ford



update (u,v):

d(v) = min{d(v) + l(u,v)}

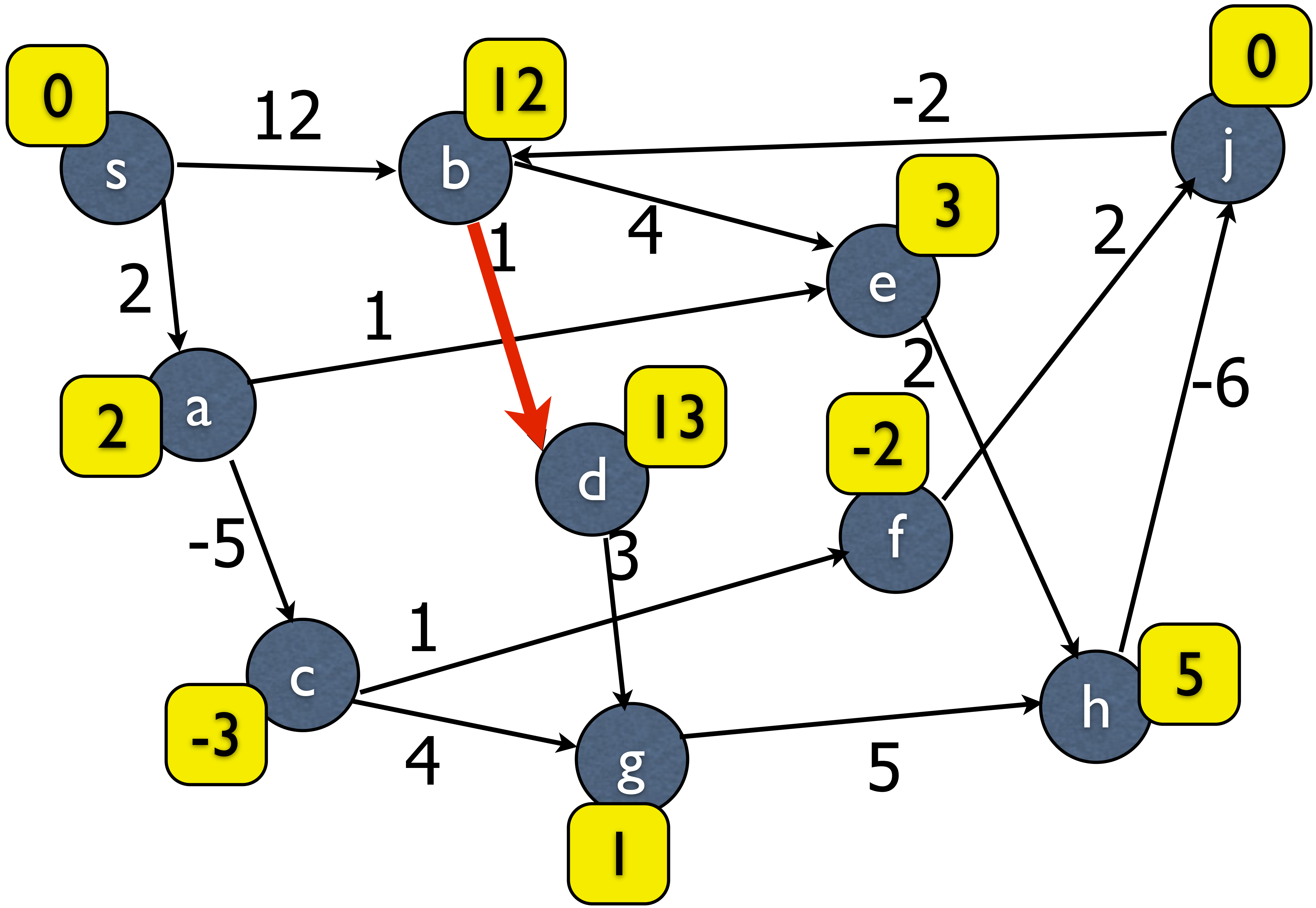
Bellman-Ford



update (u,v):

$d(v) = \min\{d(v) + l_{(u,v)}\}$

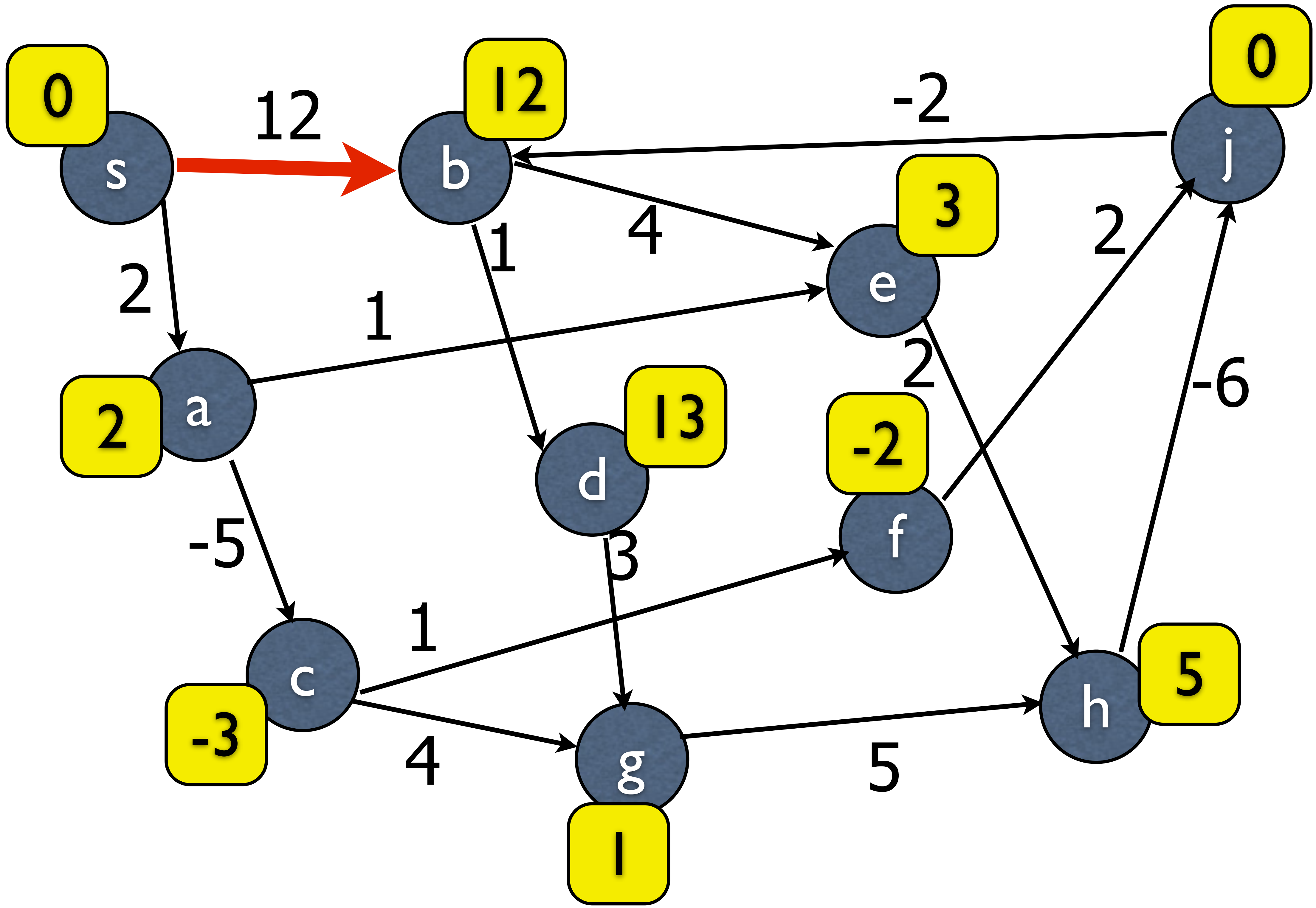
Bellman-Ford



update (u,v):

d(v) = min{d(v) + l(u,v)}

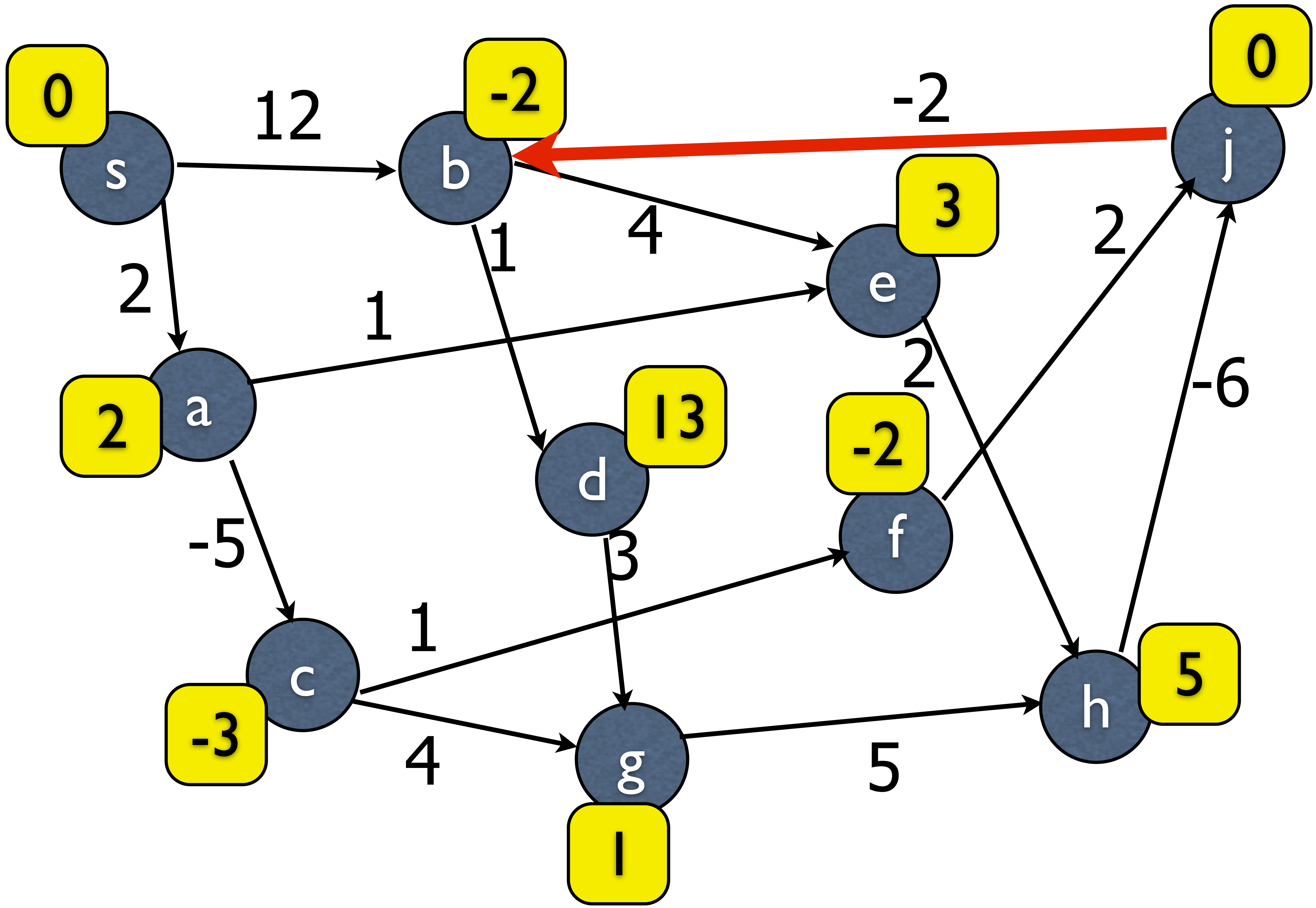
Bellman-Ford



update (u,v):

$$d(v) = \min\{d(v) + l_{(u,v)}\}$$

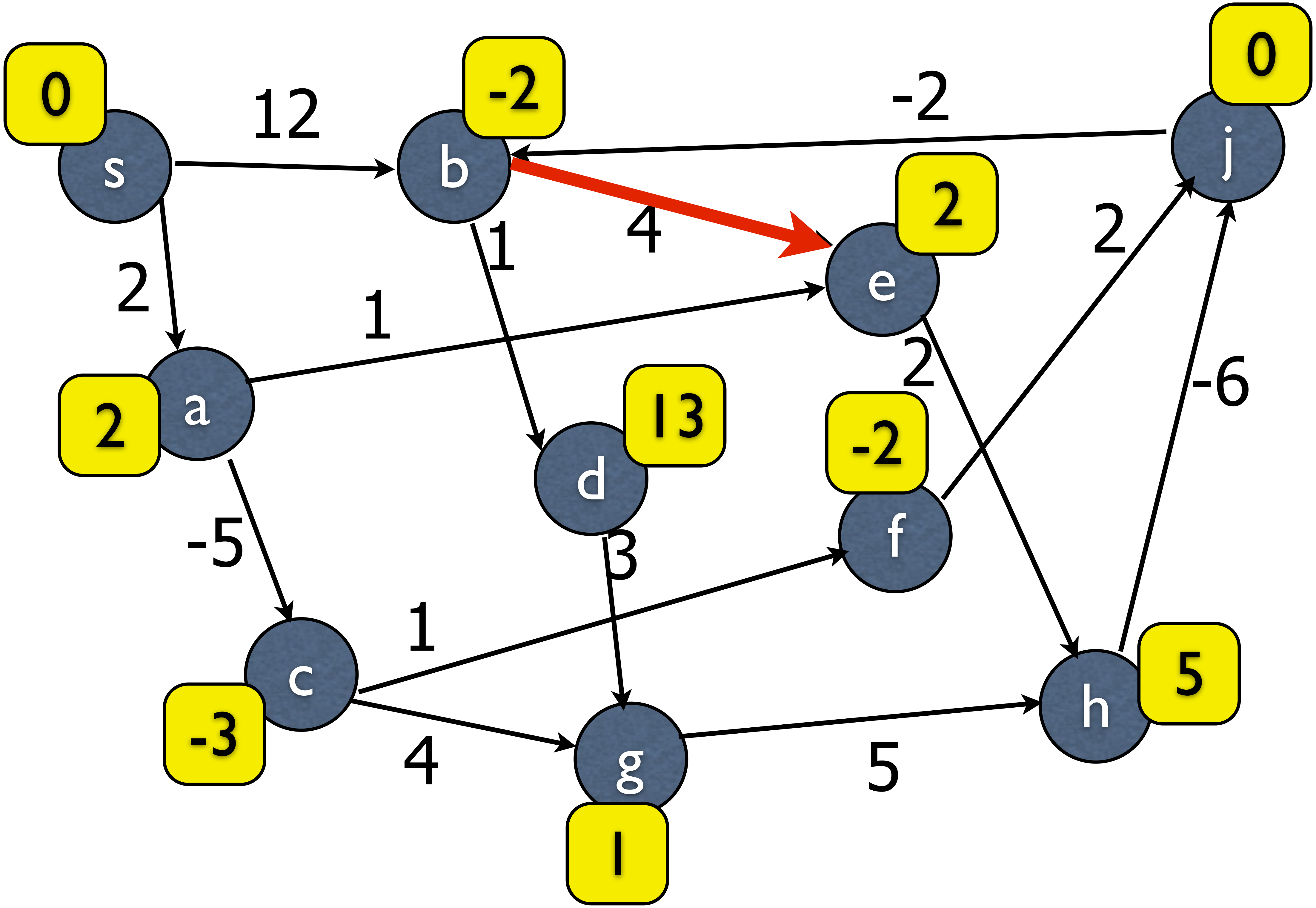
Bellman-Ford



update (u,v):

$$d(v) = \min\{d(v) + l_{(u,v)}\}$$

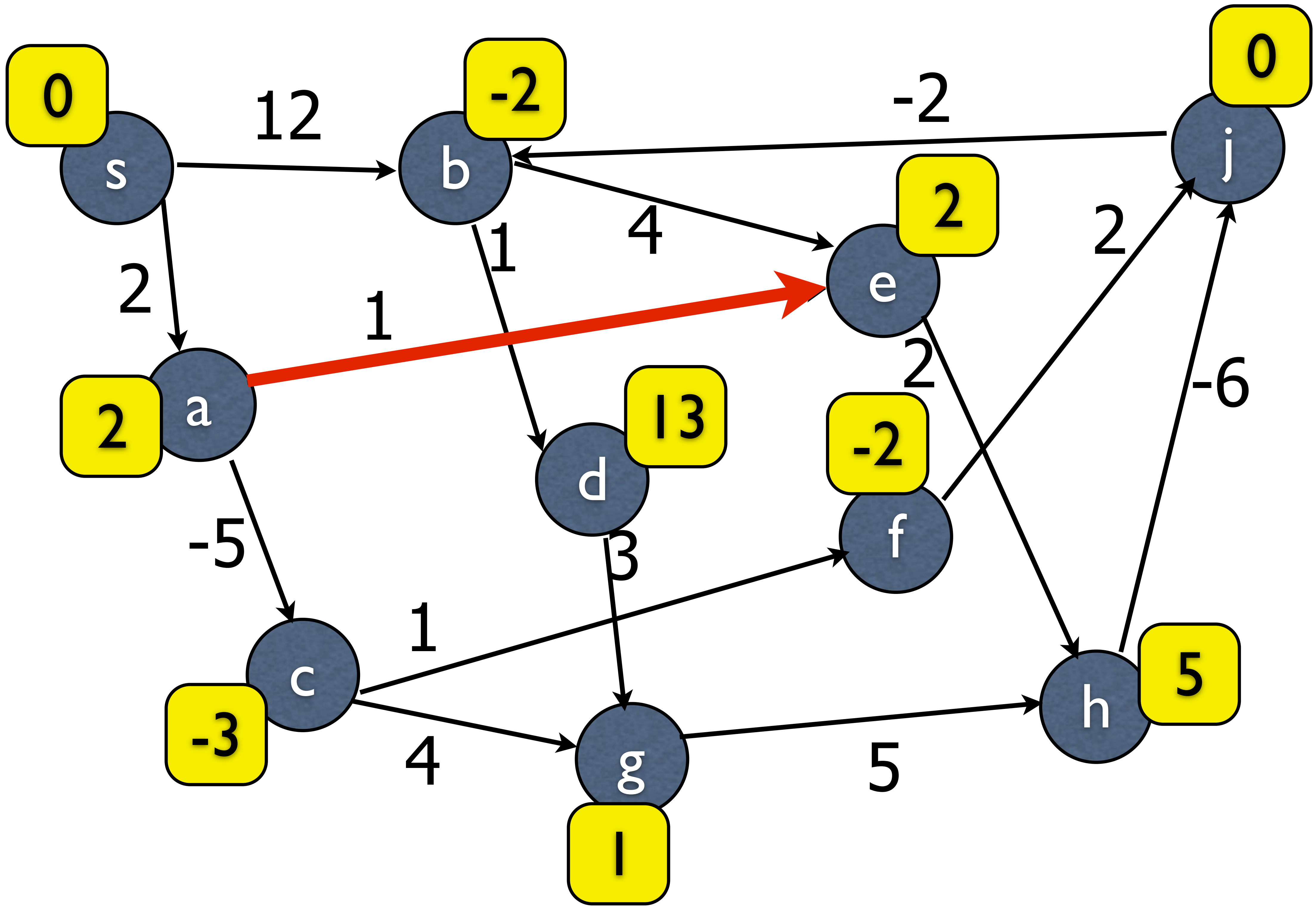
Bellman-Ford



update (u,v):

d(v) = min{d(v) + $l_{(u,v)}$}

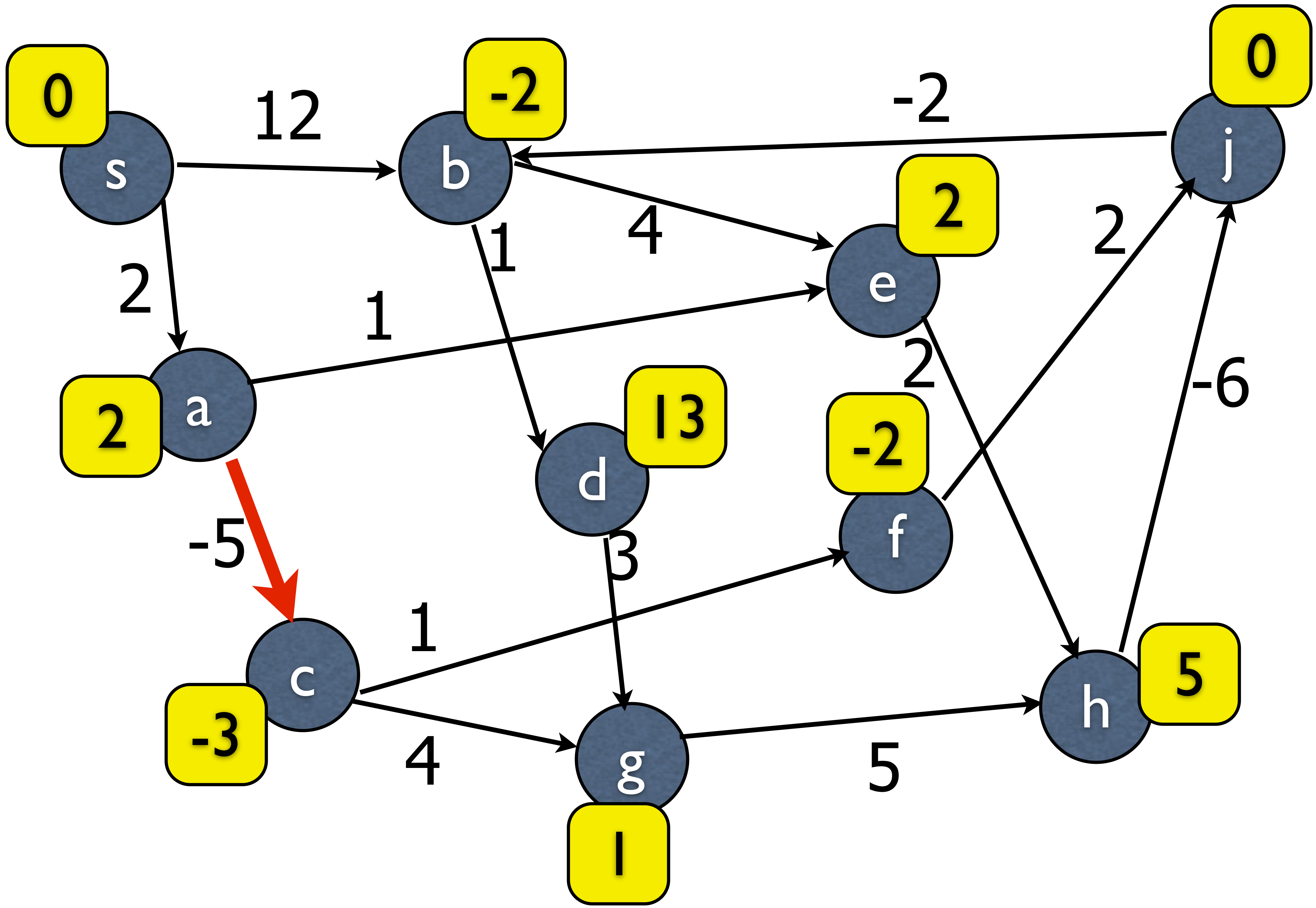
Bellman-Ford



update (u,v):

$$d(v) = \min\{d(v) + l_{(u,v)}\}$$

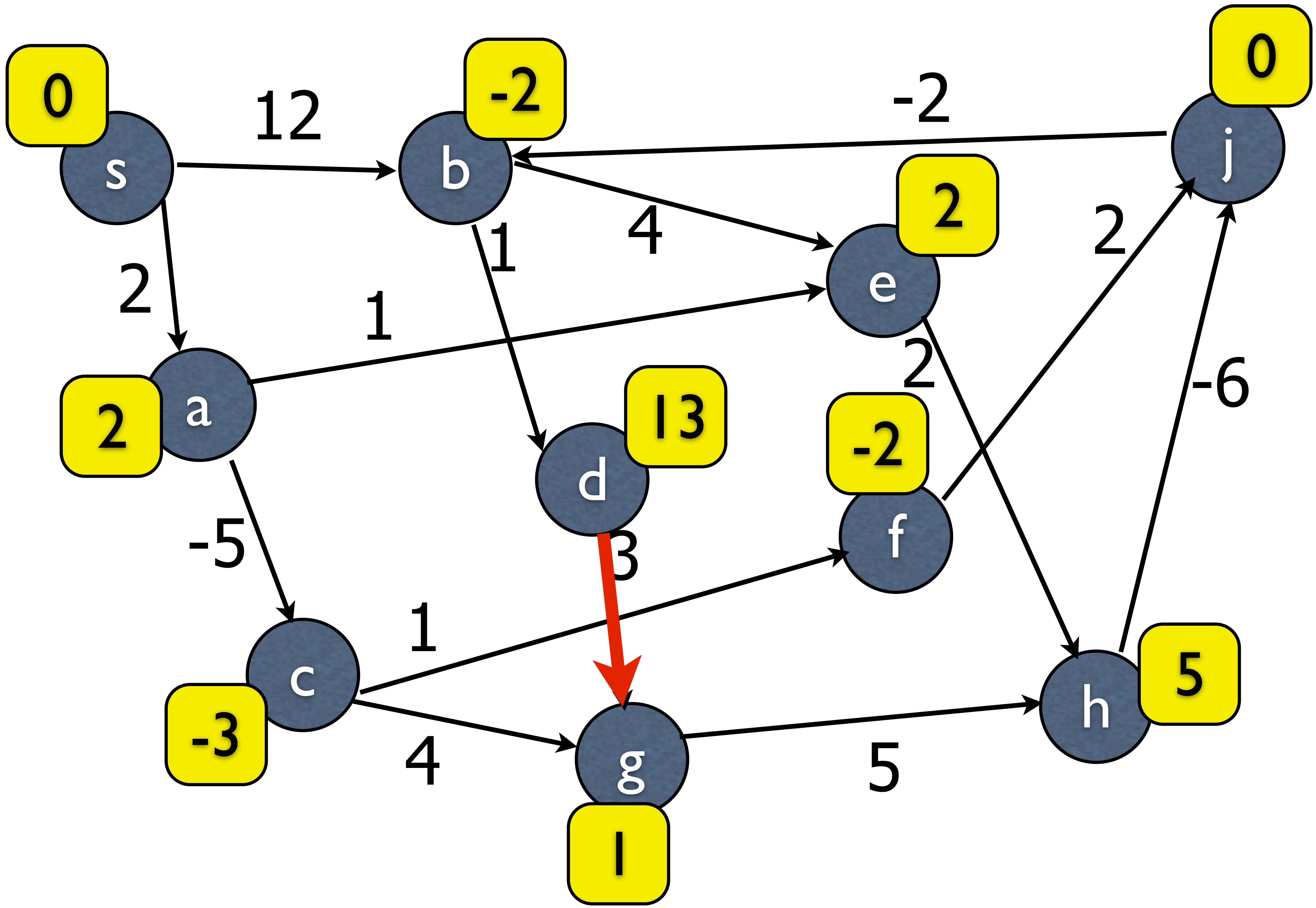
Bellman-Ford



update (u,v):

$d(v) = \min\{d(v) + l_{(u,v)}\}$

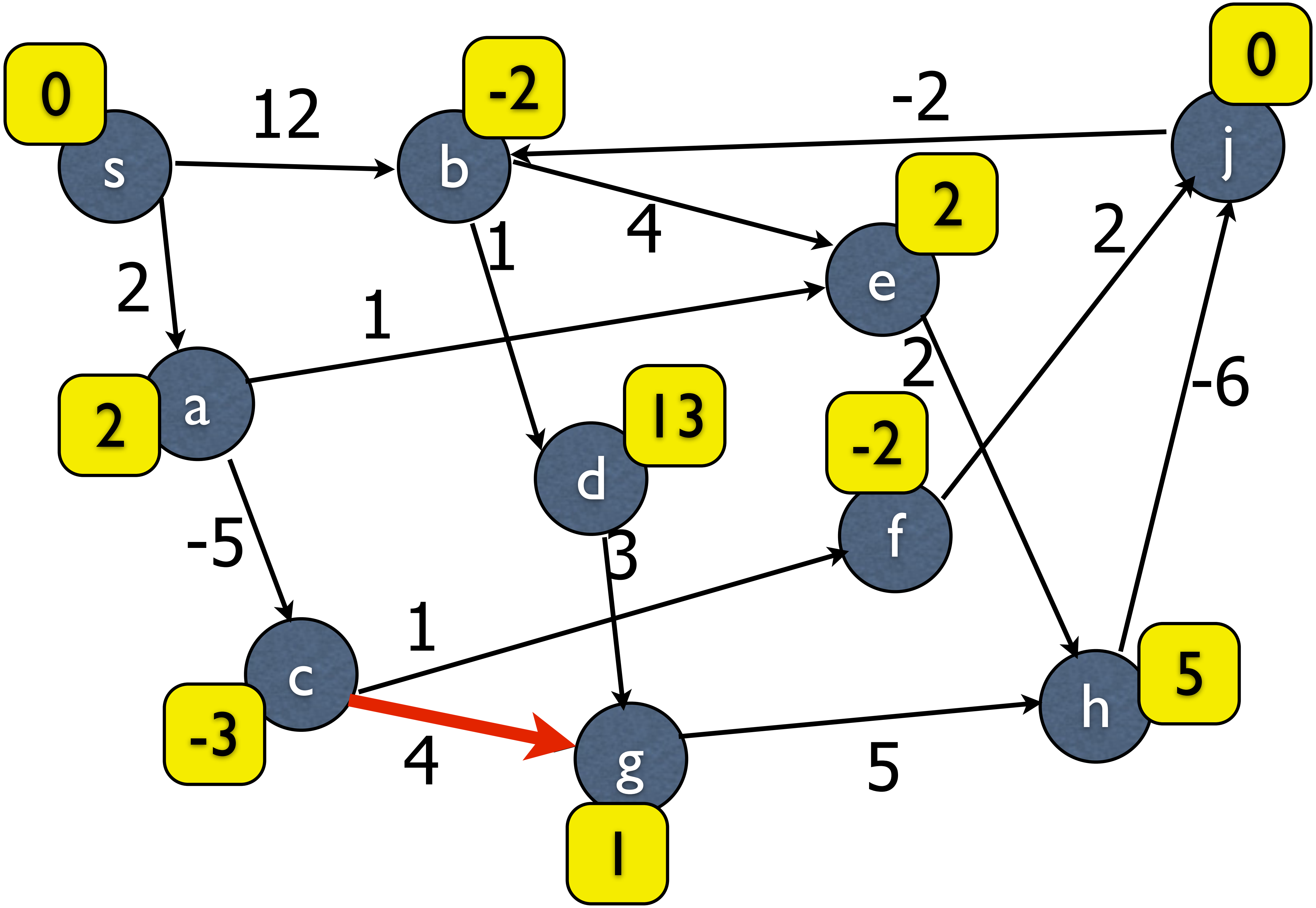
Bellman-Ford



update (u,v):

$d(v) = \min\{d(v) + l_{(u,v)}\}$

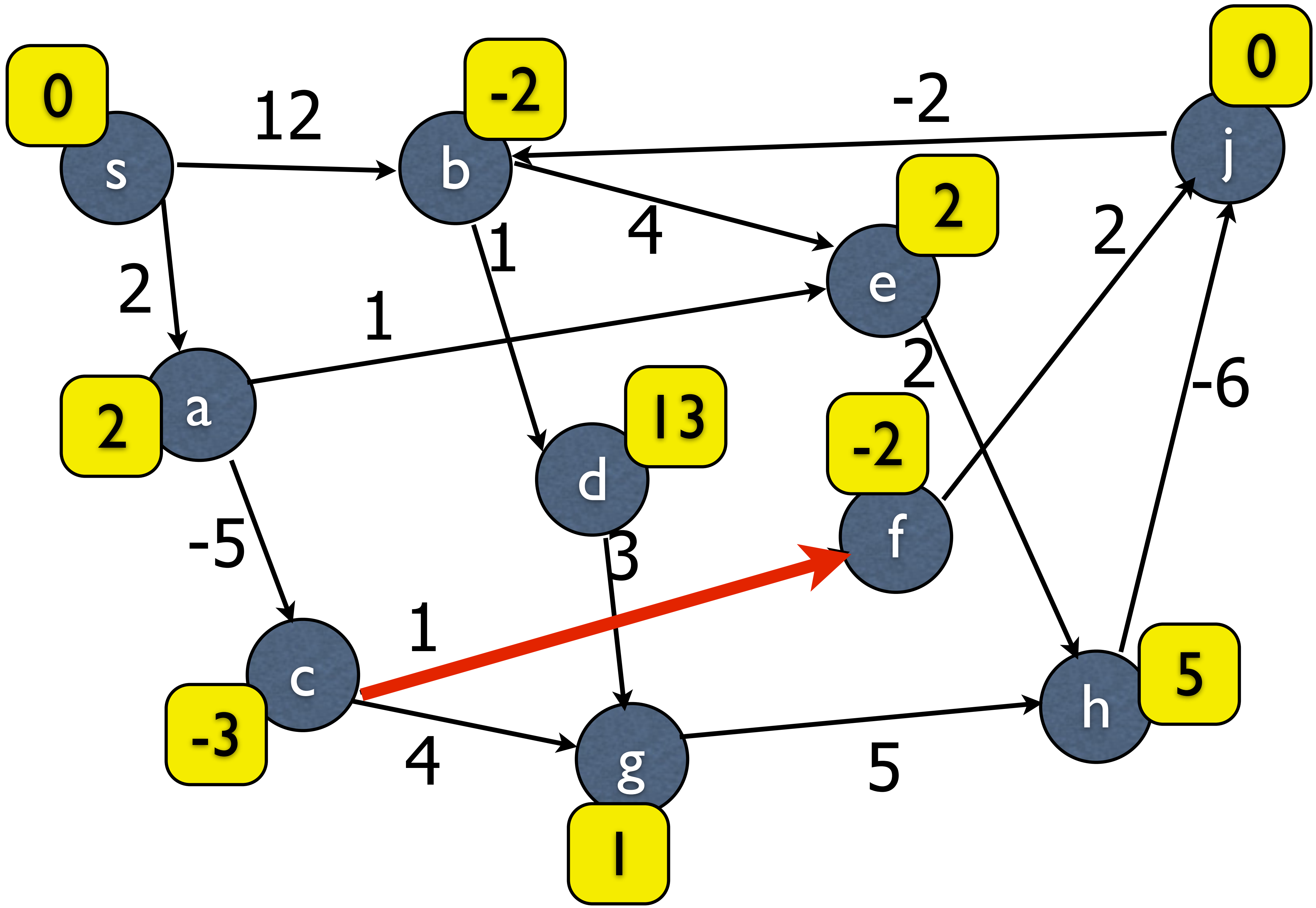
Bellman-Ford



update (u,v):

$$d(v) = \min\{d(v) + l_{(u,v)}\}$$

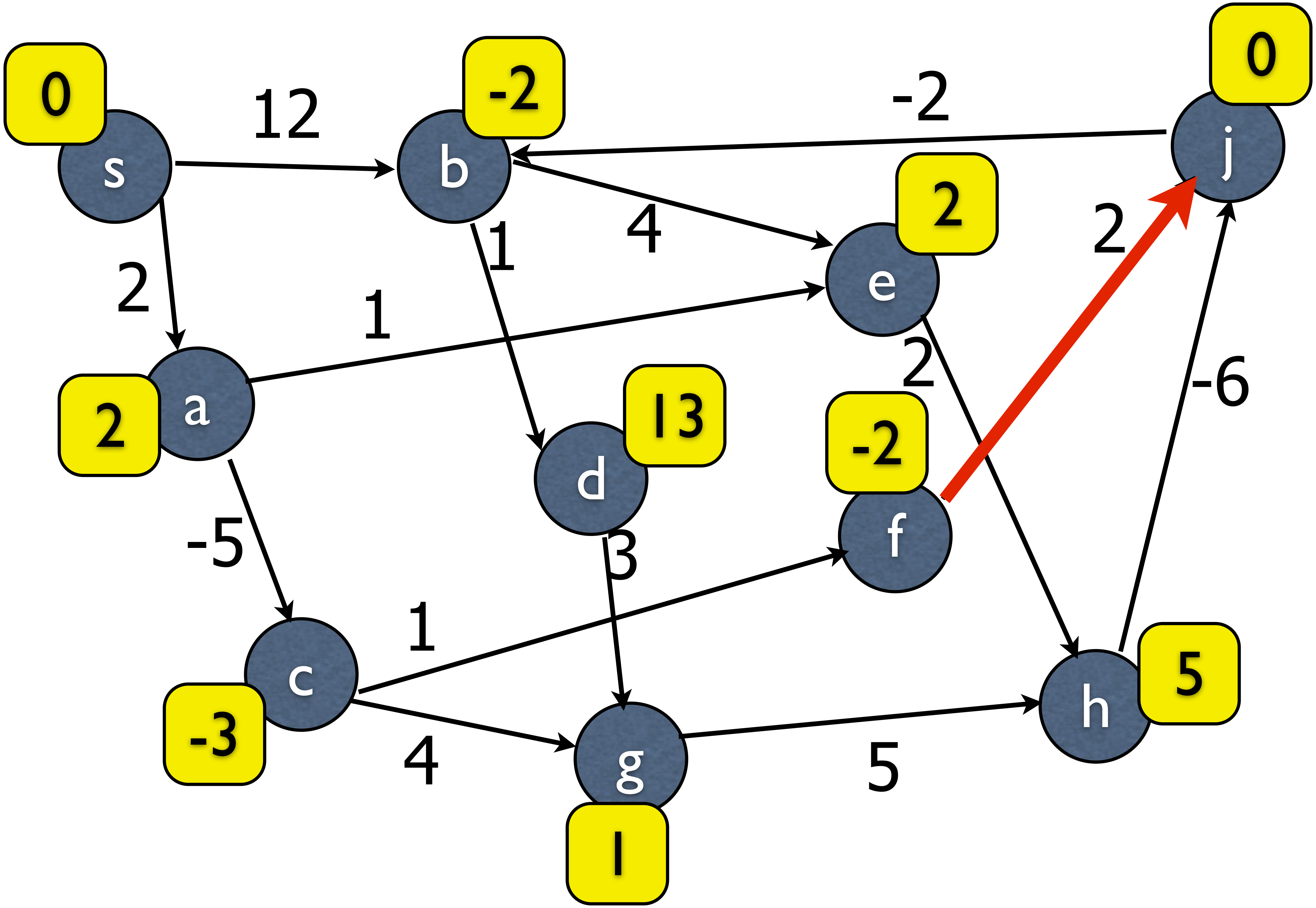
Bellman-Ford



update (u,v):

$$d(v) = \min\{d(v) + l_{(u,v)}\}$$

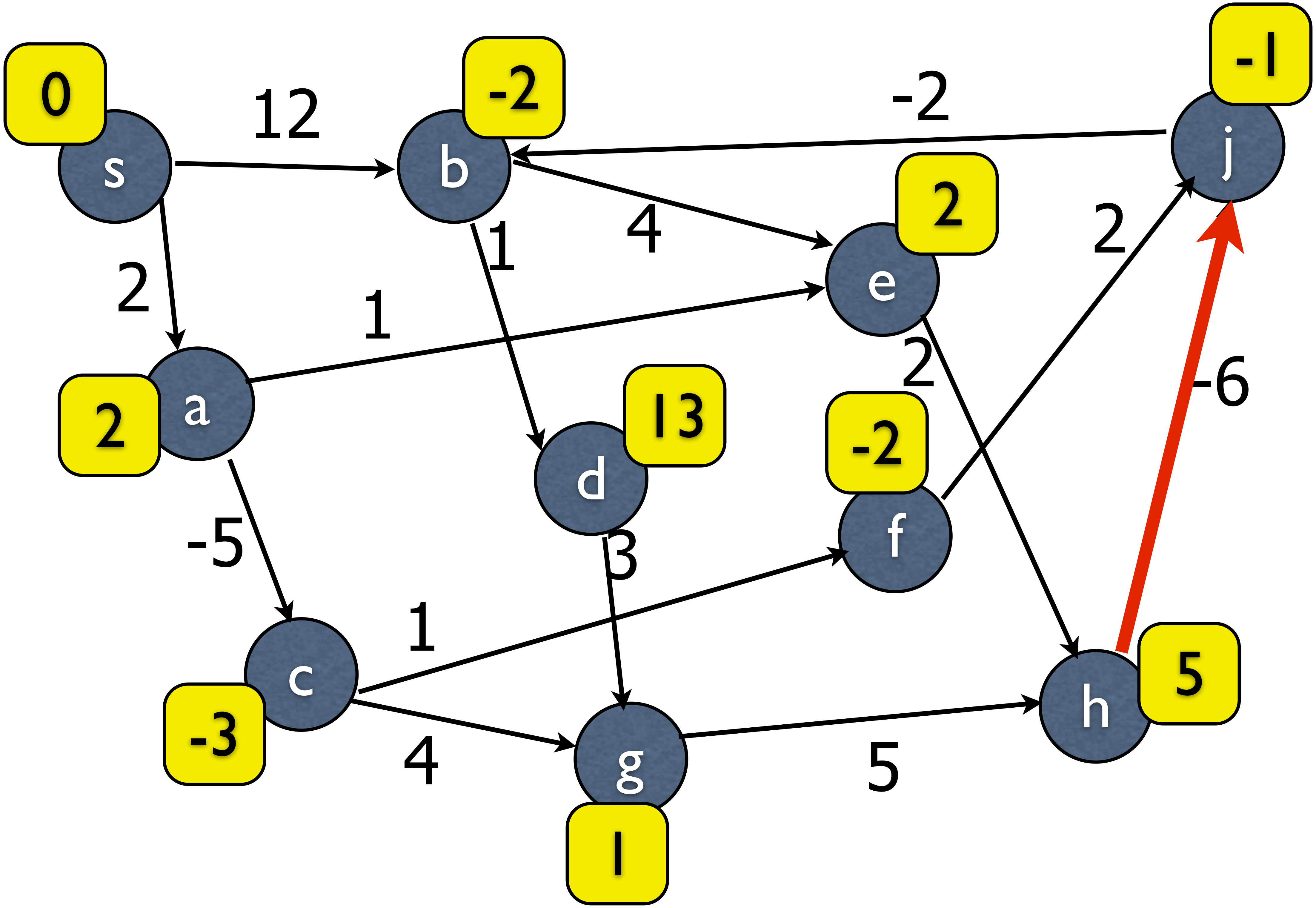
Bellman-Ford



update (u,v):

$$d(v) = \min\{d(v) + l_{(u,v)}\}$$

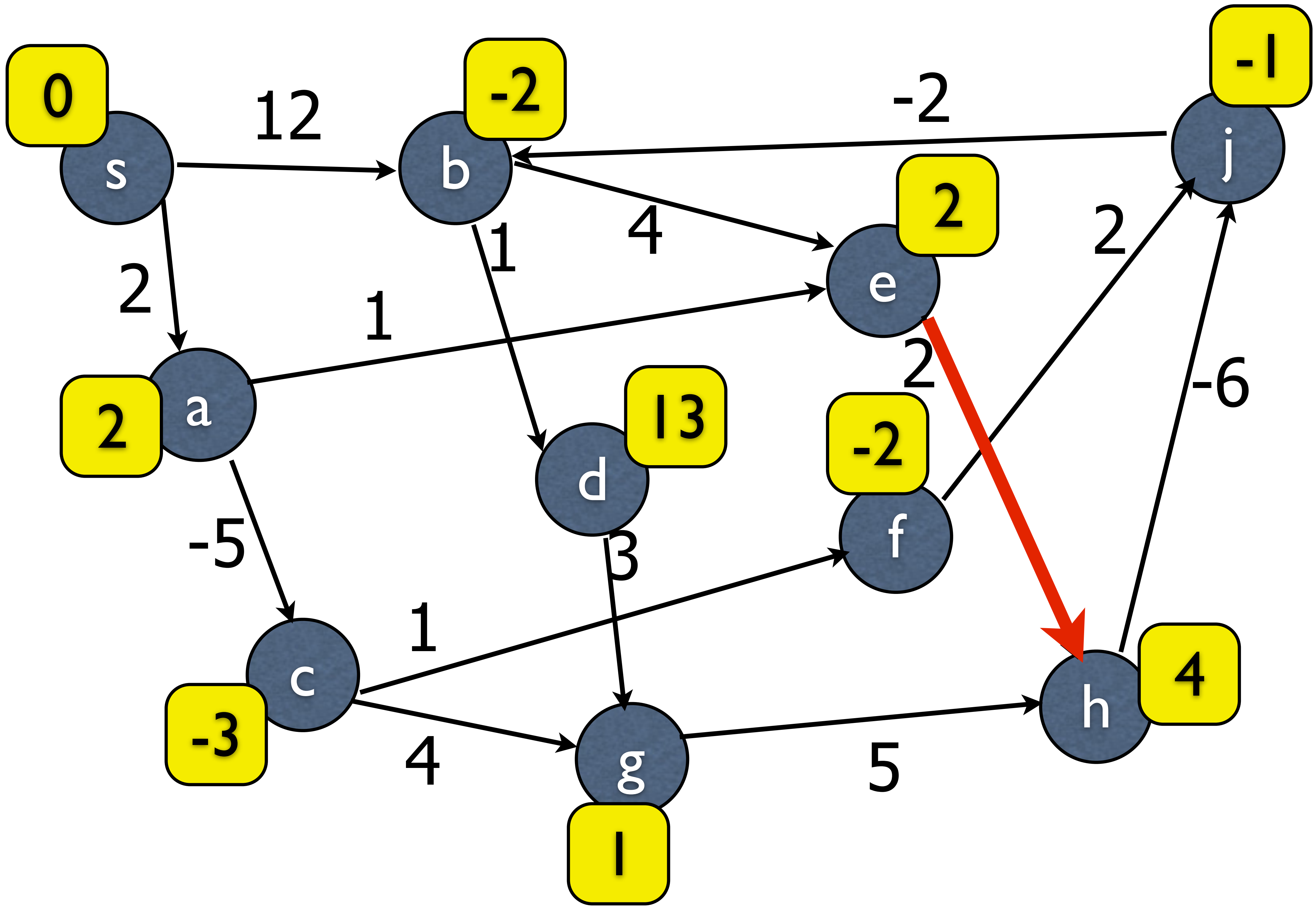
Bellman-Ford



update (u,v):

$$d(v) = \min\{d(v) + l_{(u,v)}\}$$

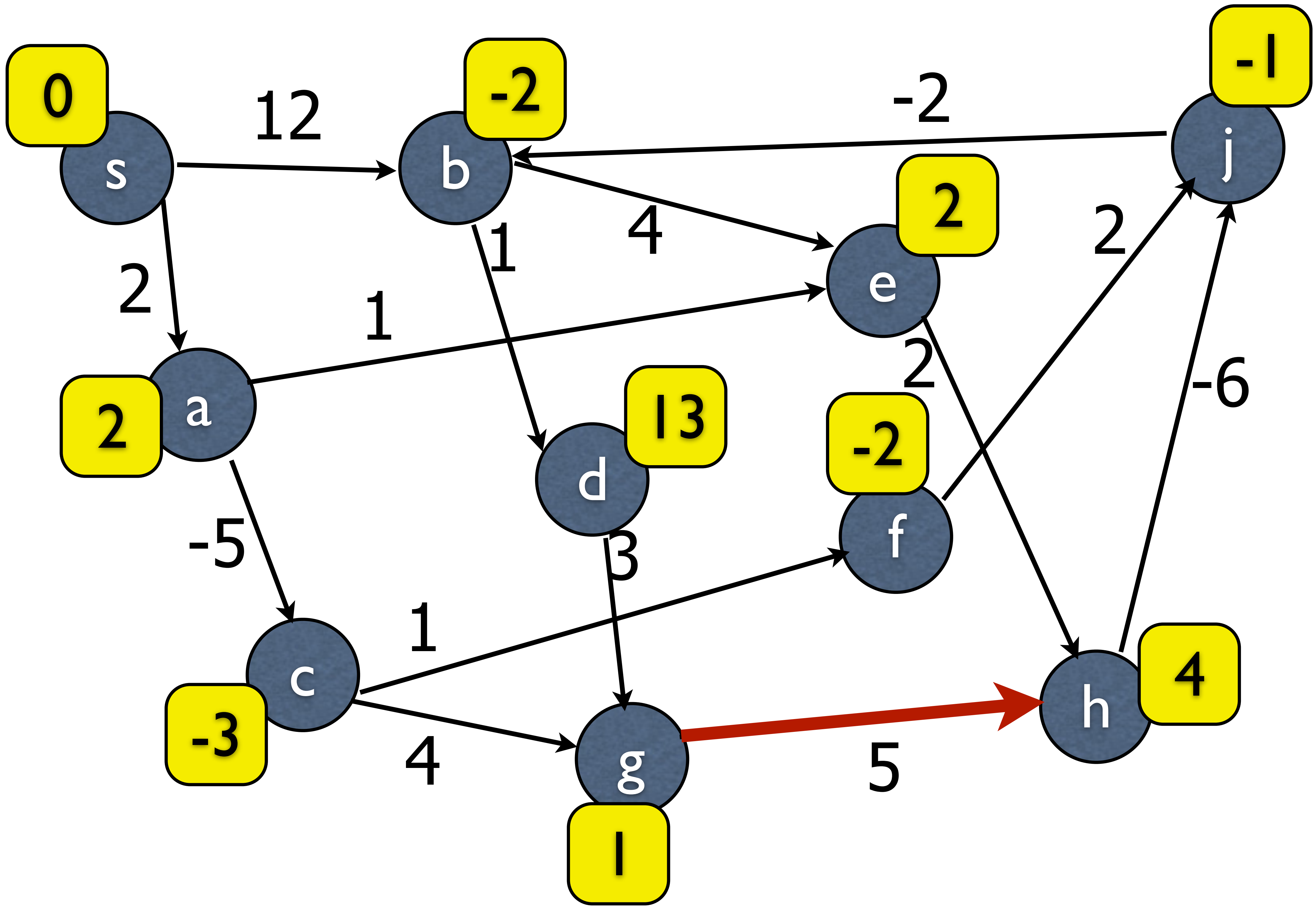
Bellman-Ford



update (u,v):

$$d(v) = \min\{d(v) + l_{(u,v)}\}$$

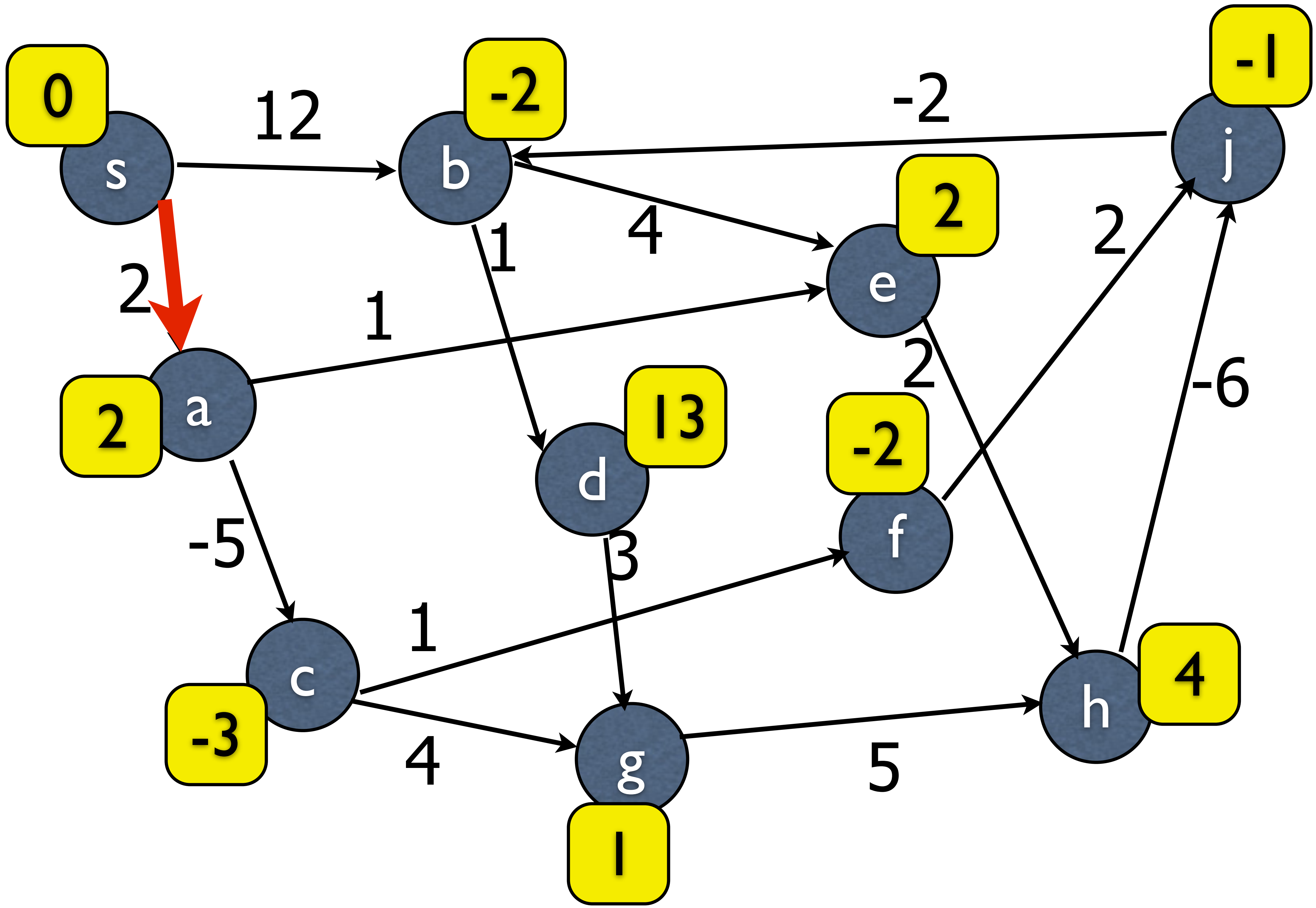
Bellman-Ford



update (u,v):

$d(v) = \min\{d(v) + l_{(u,v)}\}$

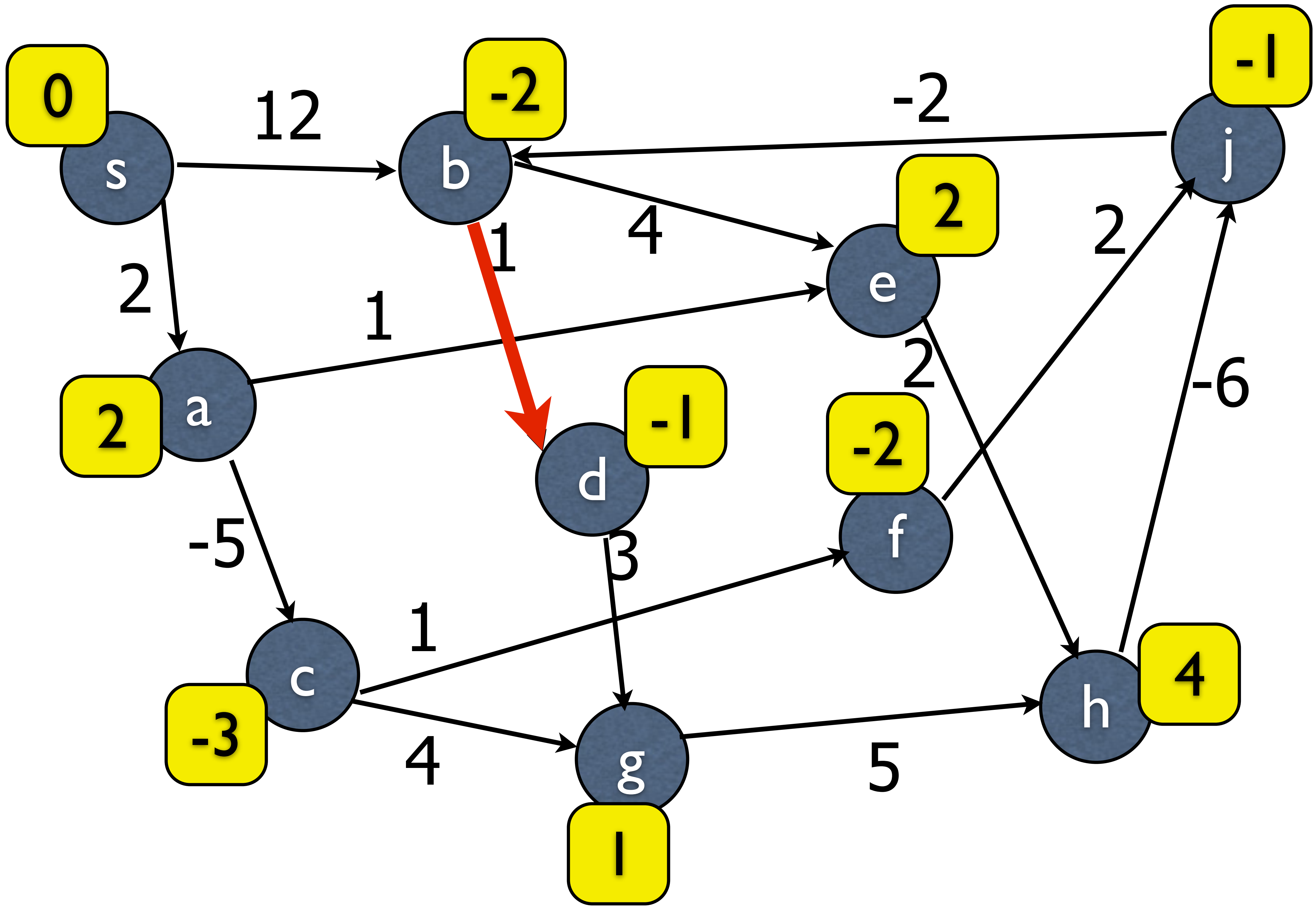
Bellman-Ford



update (u,v):

$d(v) = \min\{d(v) + l_{(u,v)}\}$

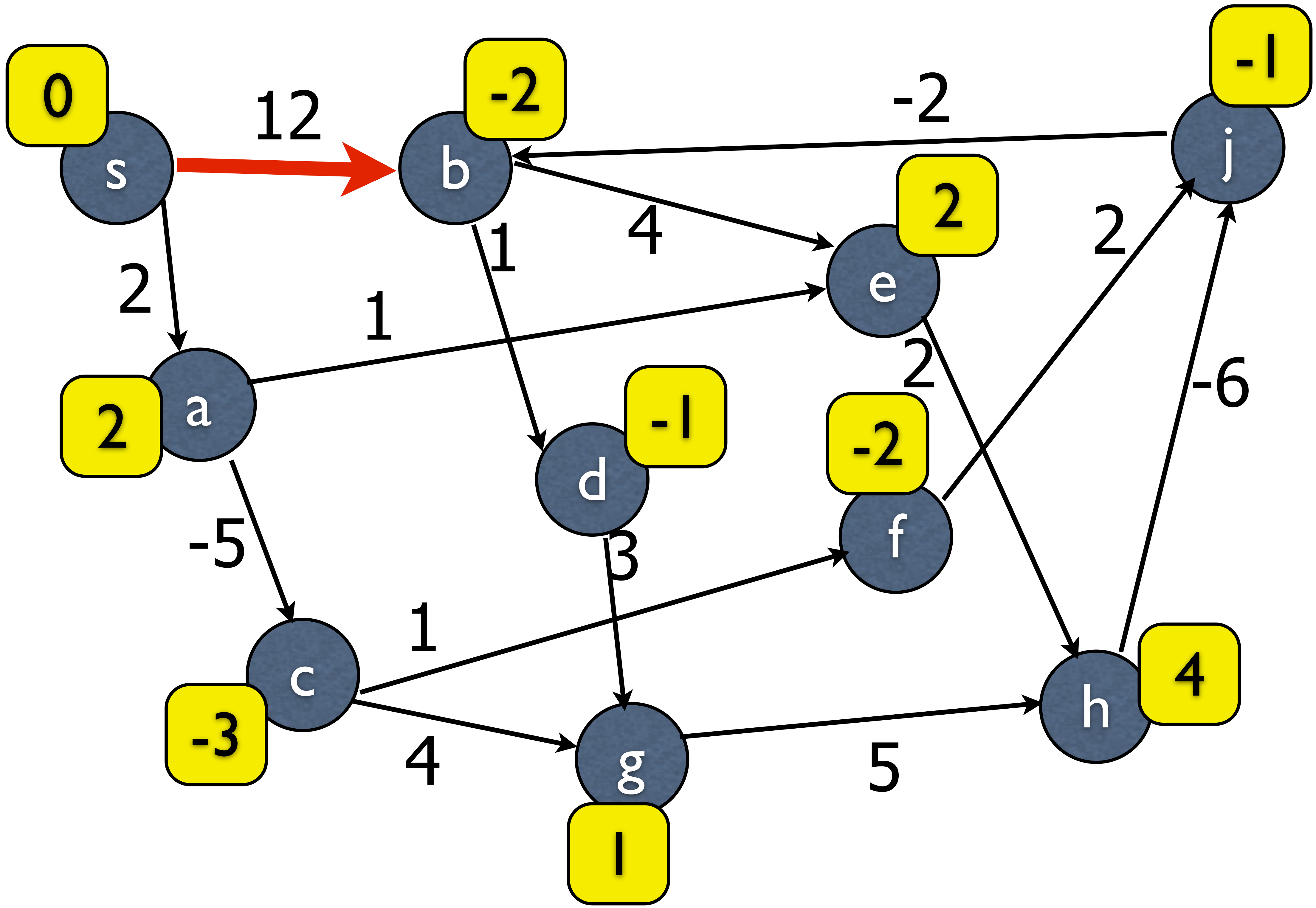
Bellman-Ford



update (u,v):

$d(v) = \min\{d(v) + l_{(u,v)}\}$

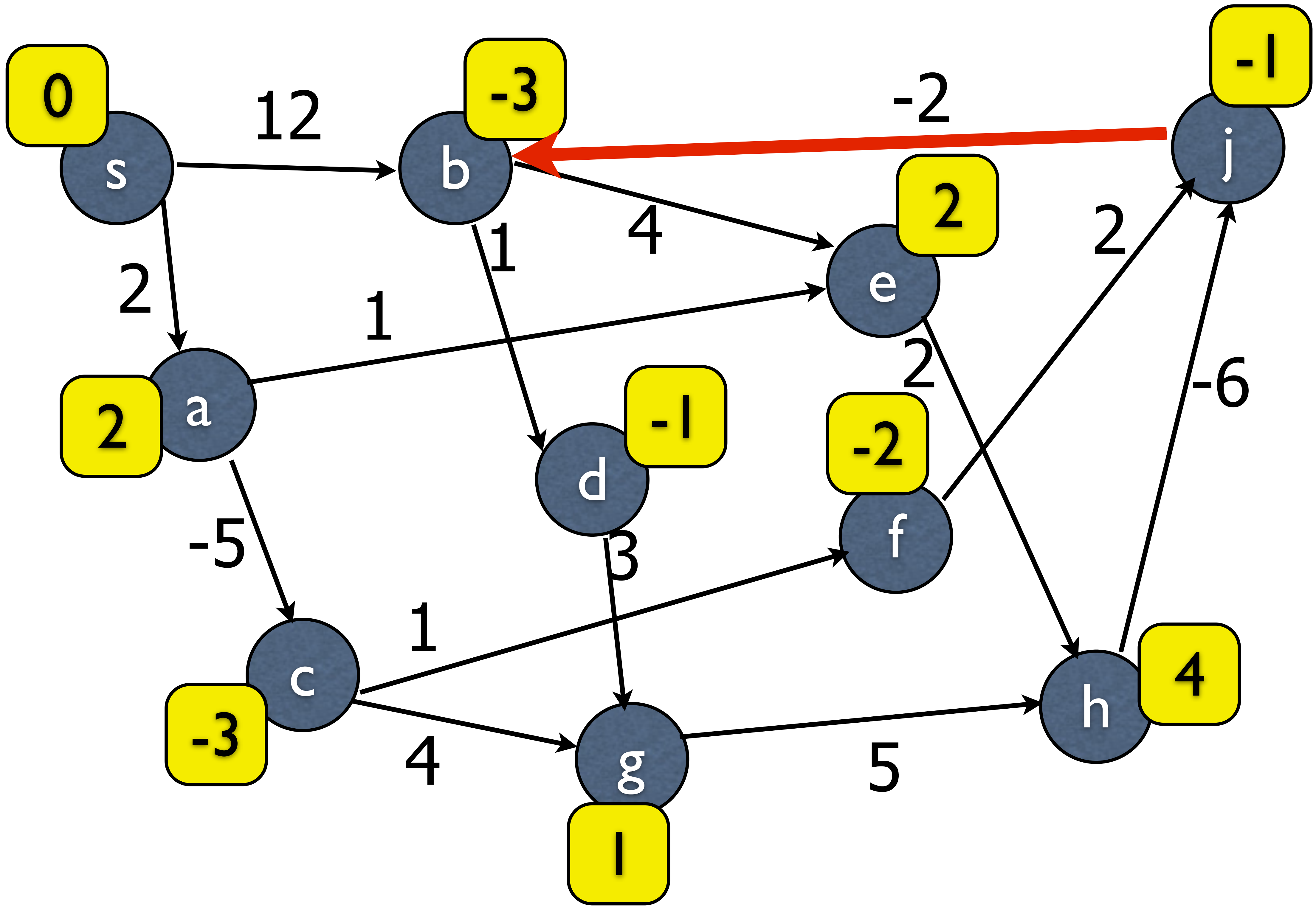
Bellman-Ford



update (u,v):

$d(v) = \min\{d(v) + l_{(u,v)}\}$

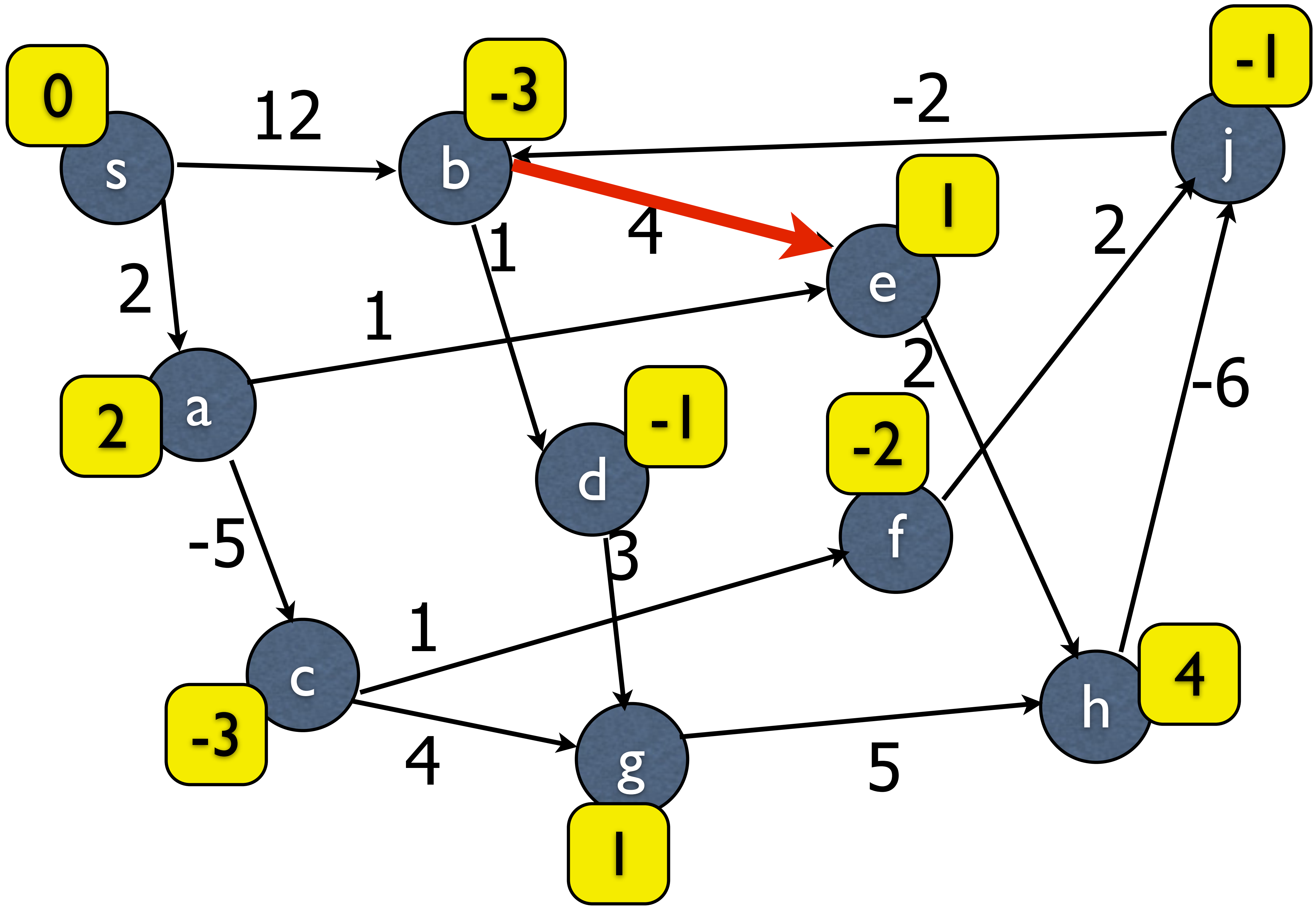
Bellman-Ford



update (u,v):

d(v) = min{d(v) + $l_{(u,v)}$}

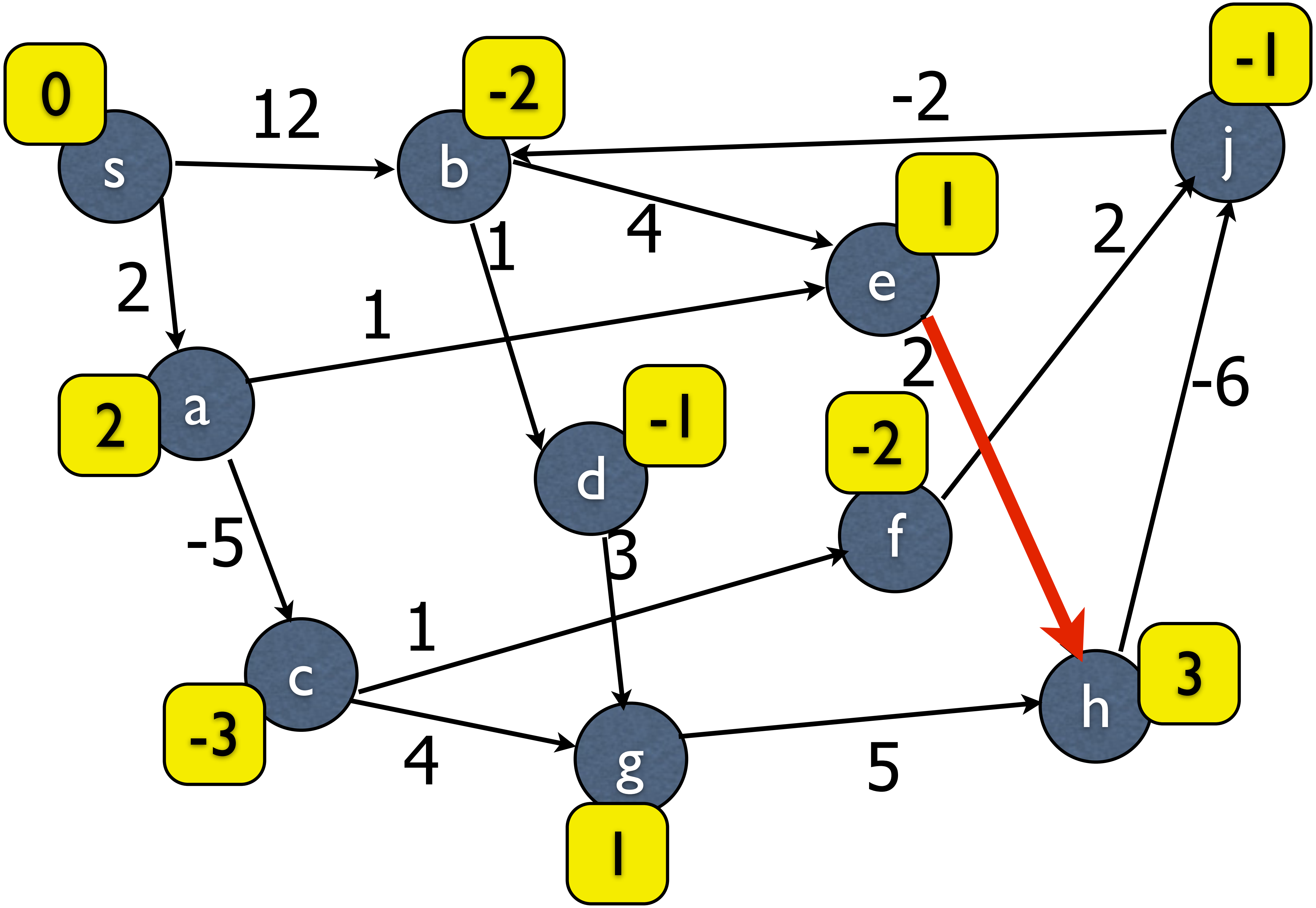
Bellman-Ford



update (u,v):

$d(v) = \min\{d(v) + l_{(u,v)}\}$

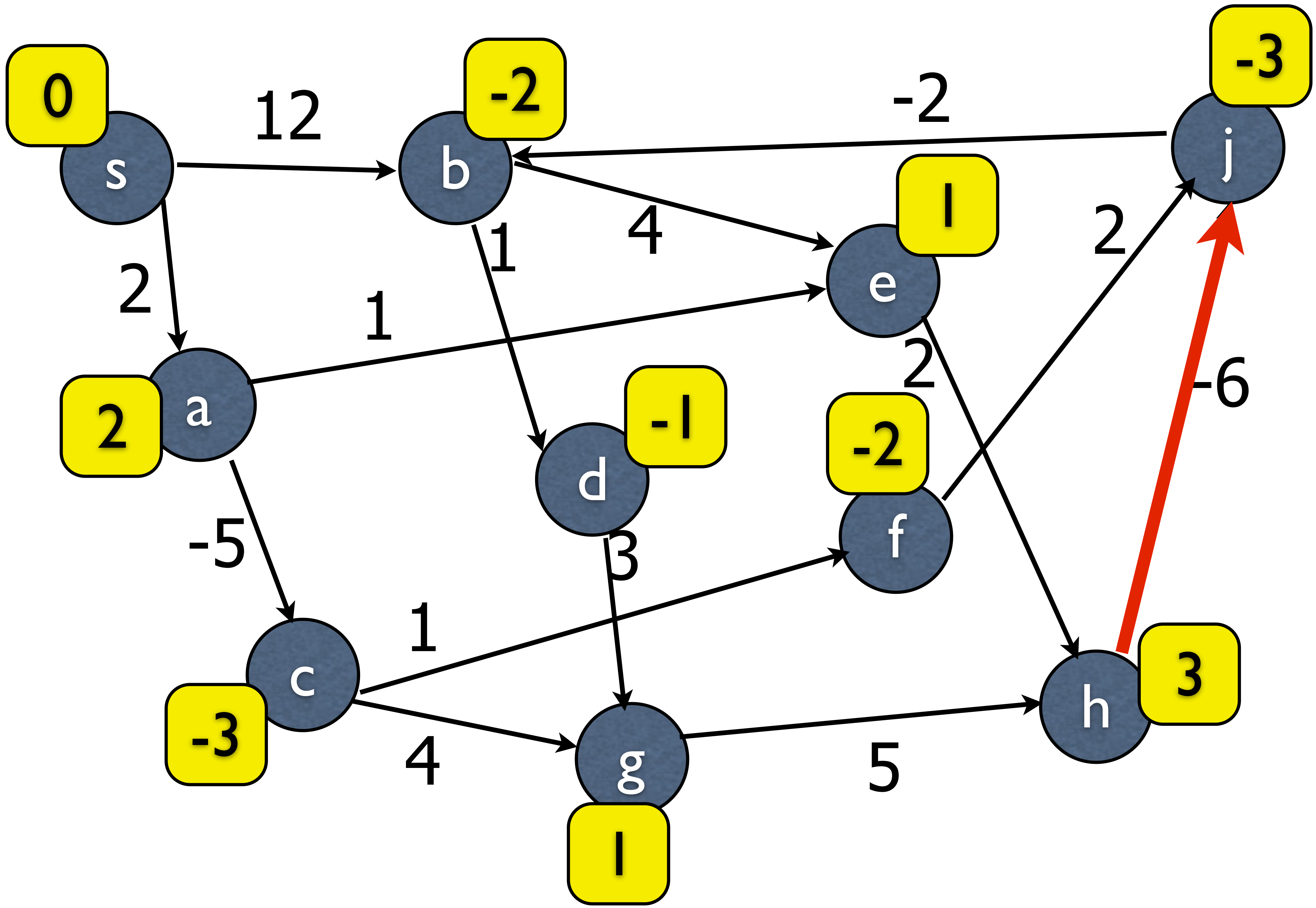
Bellman-Ford



update (u,v):

$d(v) = \min\{d(v) + l_{(u,v)}\}$

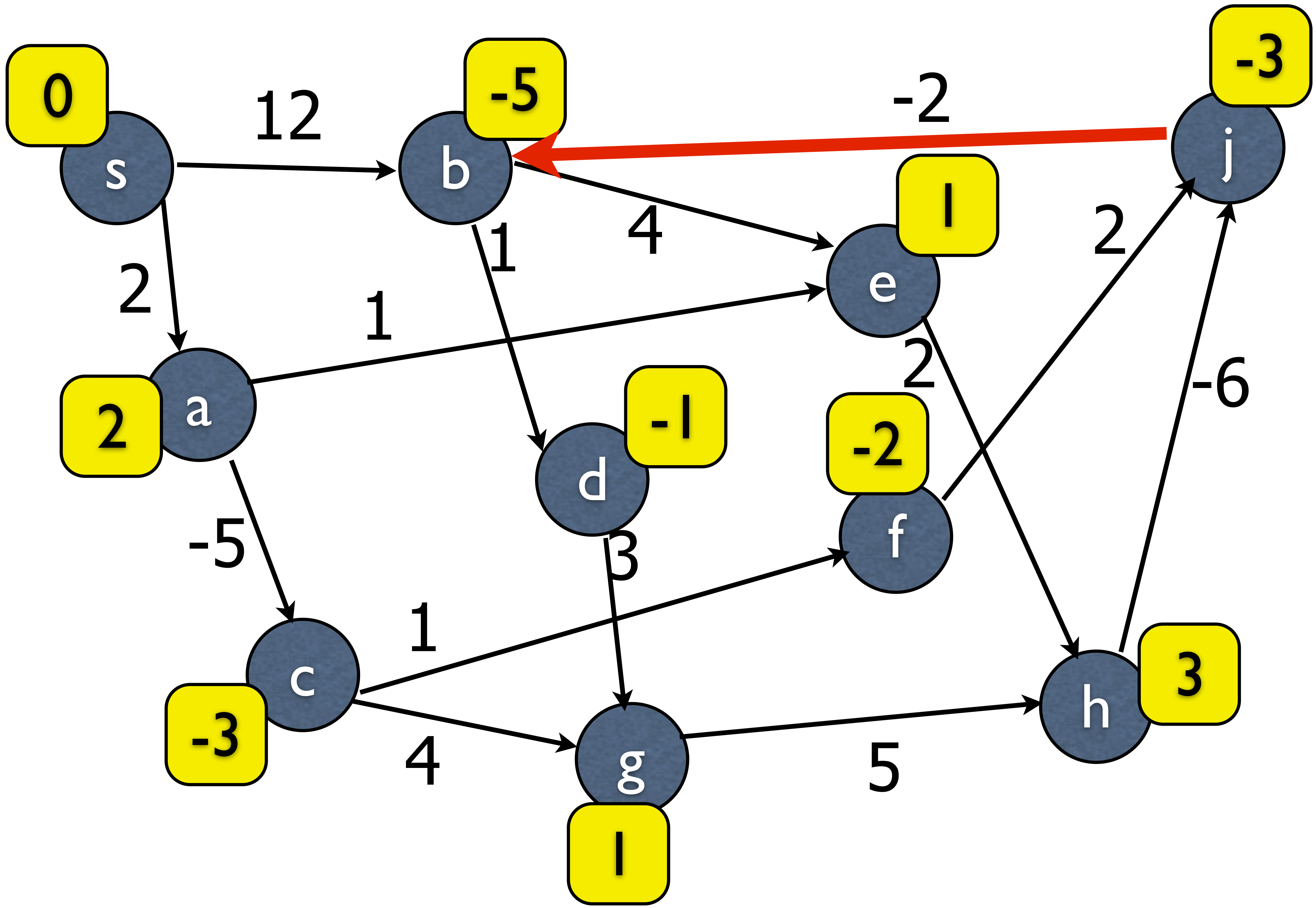
Bellman-Ford



update (u,v):

$d(v) = \min\{d(v) + l_{(u,v)}\}$

Bellman-Ford



update (u,v):

d(v) = min{d(v) + l(u,v)}

Bellman-Ford Algorithm

For all vertices set $d(v) = \infty$

Set $d(s) = 0$

for $i=1,2,\ldots,n-1$

  for every edge (u,v)

    if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Claim: If graph has no negative length cycles, then for every v, $d(v) \geq \text{distance}(s,v)$.

Bellman-Ford Algorithm

For all vertices set d(v)= ∞

Set d(s) = 0

for i=1,2,…,n-1

  for every edge (u,v)

    if d(v) > d(u) + $l_{u,v}$, update d(v) = d(u) + $l_{u,v}$.

Claim: If graph has no negative length cycles, then for every v, d(v) ≥ distance(s,v).

Pf: Initially it is true. If we update d(v) = d(u) + $l_{u,v}$, then
d(v)
= d(u) + $l_{u,v}$
≥ distance(s,u) + $l_{u,v}$
≥ distance(s,v)

Bellman-Ford Algorithm

For all vertices set $d(v) = \infty$

Set $d(s) = 0$

for $i=1,2,\ldots,n-1$

for every edge (u,v)

if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Claim: If graph has no negative length cycles, then for every v, $d(v) \geq \text{distance}(s,v)$.

Claim: If $(s,u_1),(u_1,u_2),\ldots,(u_{k-1},u_k)$ occur as a subsequence in the sequence of edge updates of algorithm, then $d(u_k) \leq l_{s,u1}+l_{u1,u2}+\ldots+l_{uk-1,uk}$

Pf: After (s,u_1) is updated, $d(u_1)$ is at most $l_{s,u1}$.
After (u_1,u_2) is updated, $d(u_2)$ is at most $l_{s,u1} + l_{u1,u2}$.
...

Bellman-Ford Algorithm

For all vertices set $d(v) = \infty$

Set $d(s) = 0$

for $i=1,2,\ldots,n-1$

    for every edge (u,v)

        if $d(v) > d(u) + l_{u,v}$, update $d(v) = d(u) + l_{u,v}$.

Claim: If graph has no negative length cycles, then for every v, $d(v) \geq \text{distance}(s,v)$.

Claim: If $(s,u_1),(u_1,u_2),\ldots,(u_{k-1},u_k)$ occur as a subsequence in the sequence of edge updates of algorithm, then $d(u_k) \leq l_{s,u1}+l_{u1,u2}+\ldots+l_{uk-1,uk}$

Claim: Every sequence of n-1 edges occurs as a subsequence of the edge sequence used in the algorithm, so d(u) is at most distance(s,u) at the end.

Bellman-Ford Algorithm

For all vertices set d(v)= ∞

Set d(s) = 0

for i=1,2,...,n-1

    for every edge (u,v)

        if d(v) > d(u) + $l_{u,v}$, update d(v) = d(u) + $l_{u,v}$.

Running time analysis:

O((m+n)n).

Detecting Negative Cycles

- Run Bellman-Ford n times. If any value d(v) changes in the n'th iteration, there is a negative cycle!