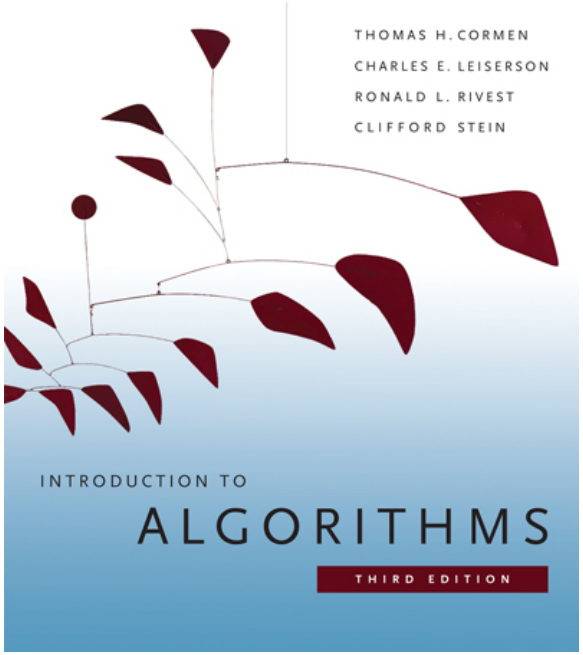
6.006- *Introduction to Algorithms*

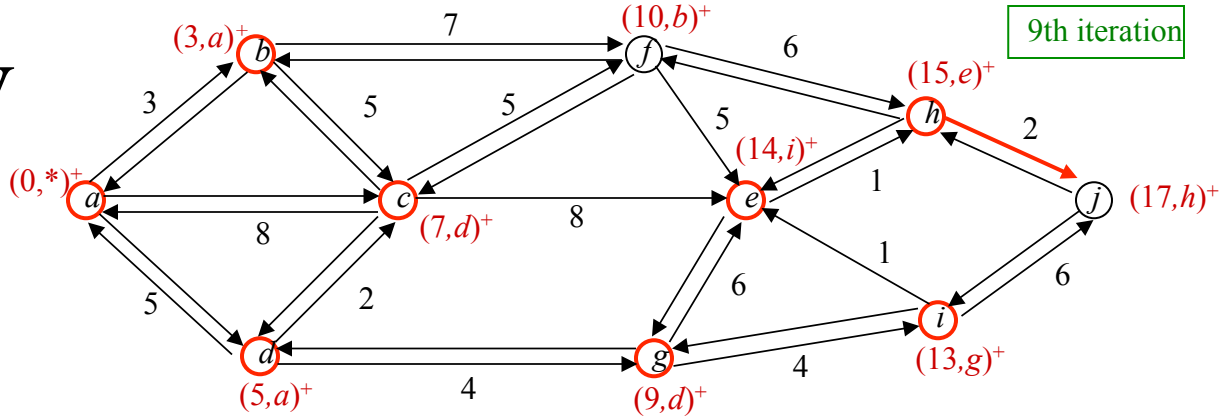


Lecture 17

Prof. Patrick Jaillet

Lecture overview

Shortest paths IV



– review and analysis of Djikstra

– speeding it up

- faster special cases/implementation
- one source-one target
  - bidirectional
  - goal-directed searches

Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$

initialization

while $Q \neq \varnothing$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        $S \leftarrow S \cup \{u\}$
        for each $v \in Adj[u]$
            do if $d[v] > d[u] + w(u, v)$
                then $d[v] \leftarrow d[u] + w(u, v)$

relaxation steps

(Implicit DECREASE-KEY)

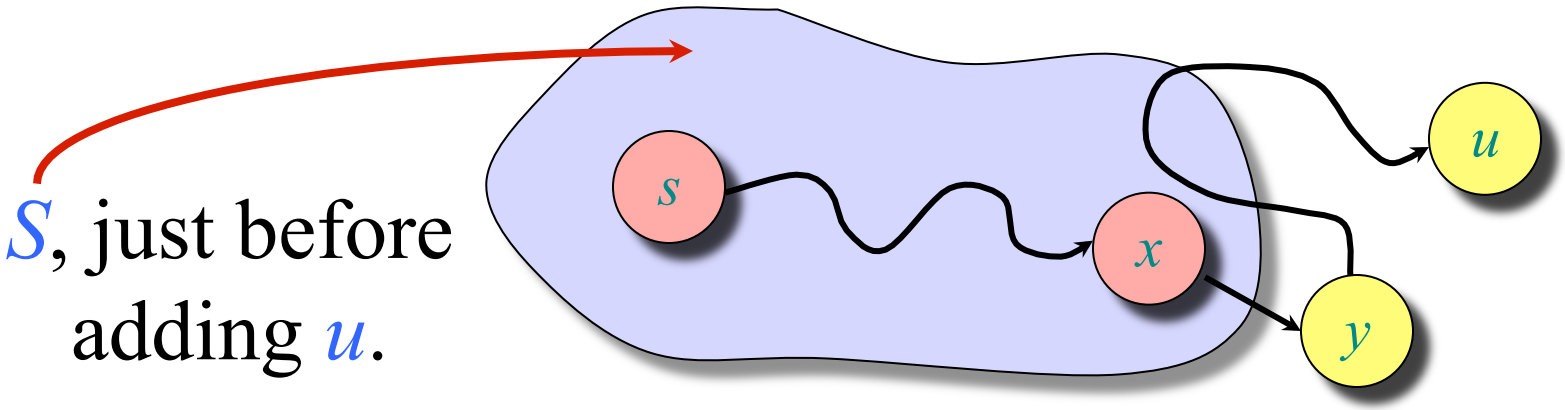
Correctness — main argument

Theorem. Dijkstra's algorithm terminates with $d[v] = \delta(s, v)$ for all $v \in V$.

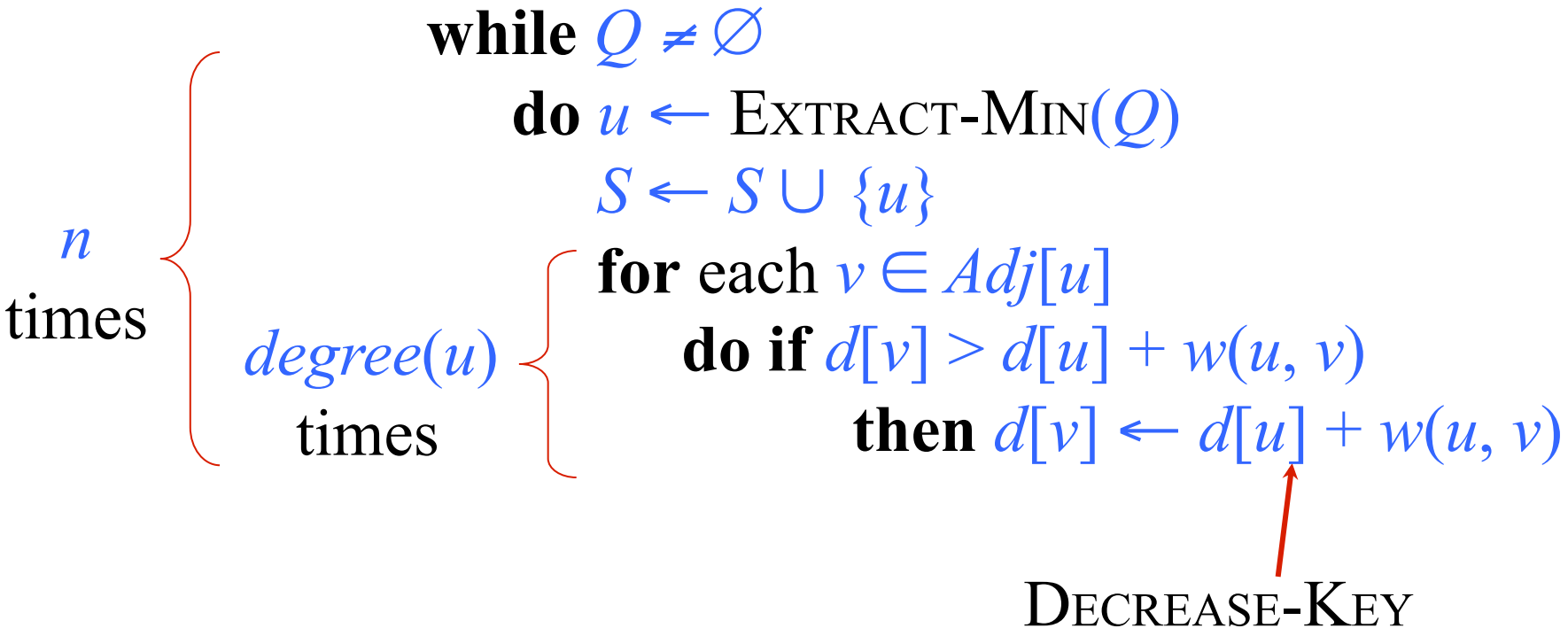
Proof.

It suffices to show that $d[v] = \delta(s, v)$ for every $v \in V$ when v is added to S.

• Suppose u is the first vertex added to S for which $d[u] \neq \delta(s, u)$. Let y be the first vertex in $V - S$ along a shortest path from s to u, and let x be its predecessor:



Analysis of Dijkstra



Time = $\Theta(n) \cdot T_{\text{EXTRACT-MIN}} + \Theta(m) \cdot T_{\text{DECREASE-KEY}}$

Analysis of Dijkstra (continued)

Time = $\Theta(n) \cdot T_{\text{EXTRACT-MIN}} + \Theta(m) \cdot T_{\text{DECREASE-KEY}}$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(n)$	$O(1)$	$O(n^2)$
binary heap	$O(\lg n)$	$O(\lg n)$	$O(m \lg n)$
Fibonacci heap	$O(\lg n)$ amortized	$O(1)$ amortized	$O(n \lg n + m)$ worst case

not covered in 6.006

Special cases of edge weights

- if edge weights are integral and bounded by a small constant C
- use an array of lists ($nC+1$ "buckets") for implementing the priority queue Q

- $T_{\text{EXTRACT-MIN}}$ and $T_{\text{DECREASE-KEY}}$ take $O(1)$
- *Time_Total is then* $O(n+m)$

Single-source s, Single-target t, S.P.P.

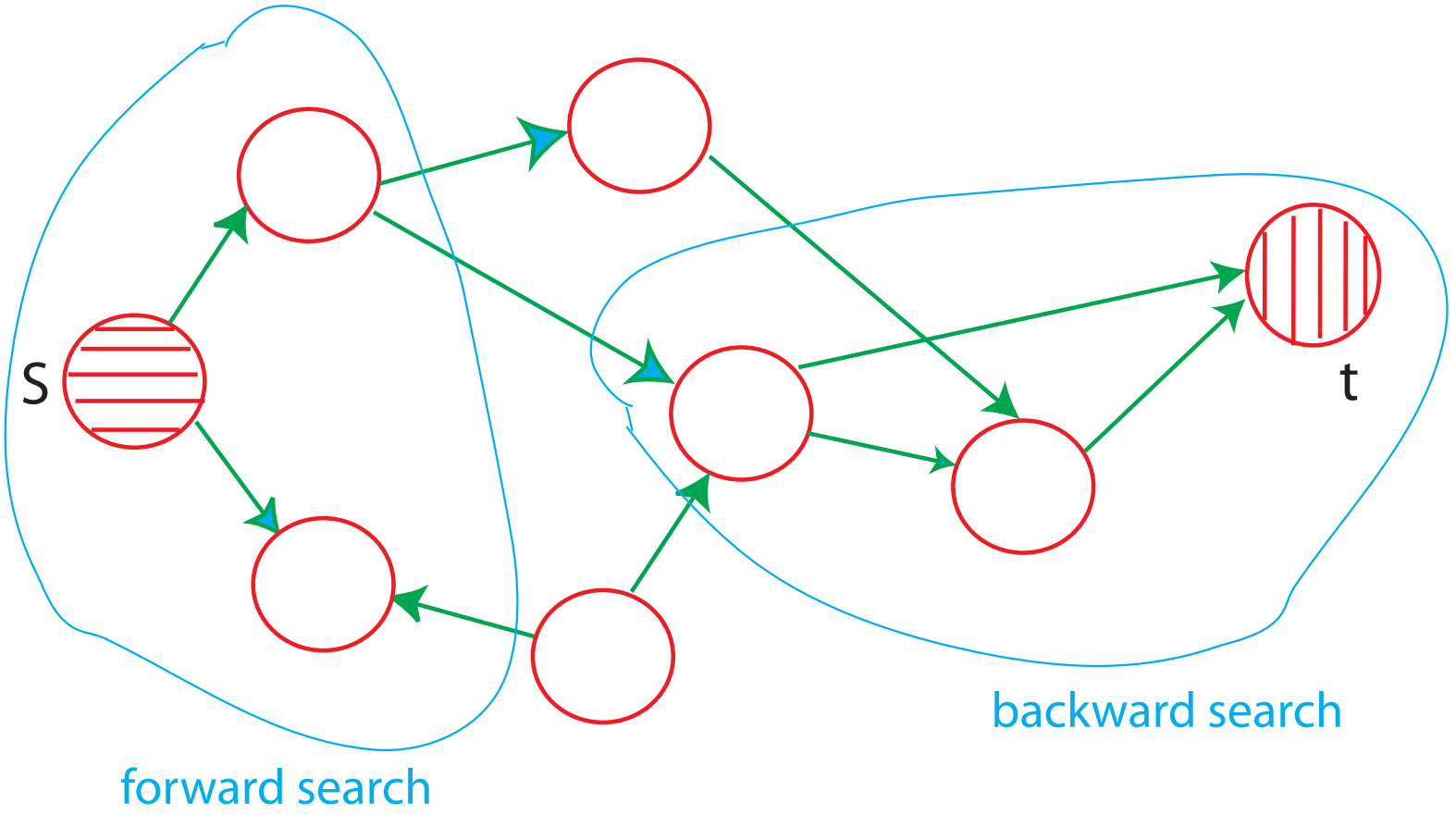
$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$
while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$
      do if $d[v] > d[u] + w(u, v)$
        then $d[v] \leftarrow d[u] + w(u, v)$

initialization

stop whenever $u = t$!!

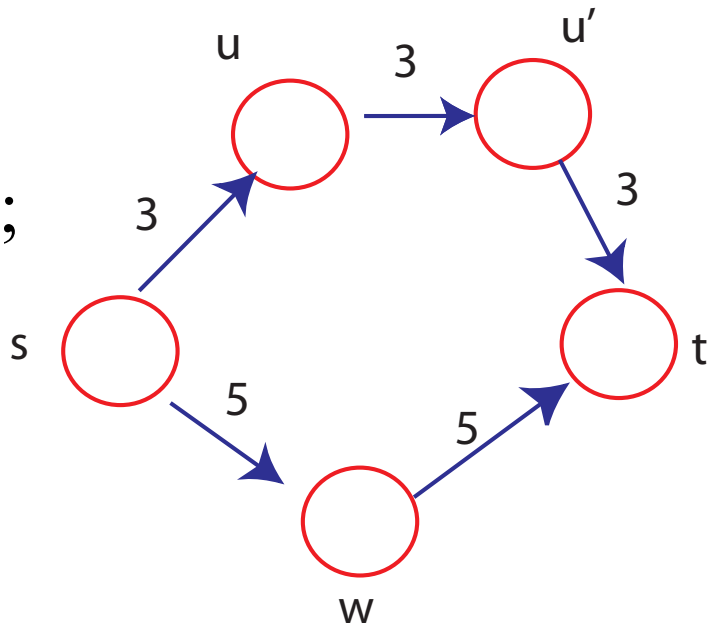
relaxation steps

Bi-directional Search

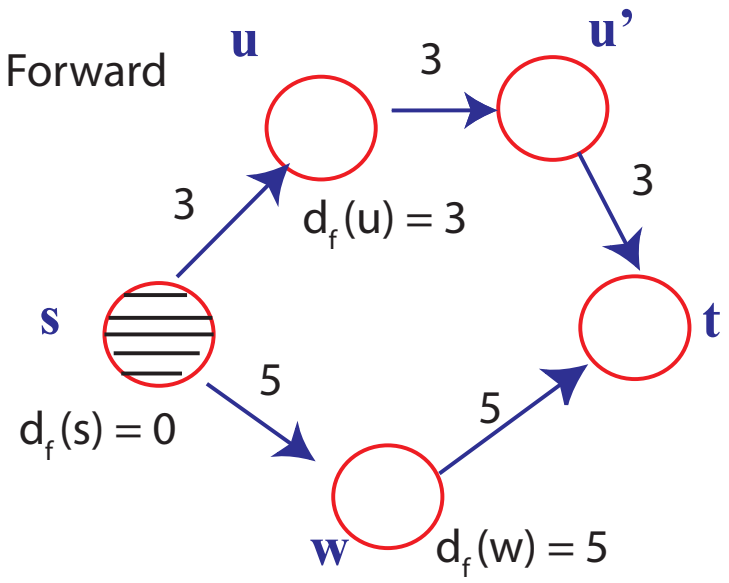


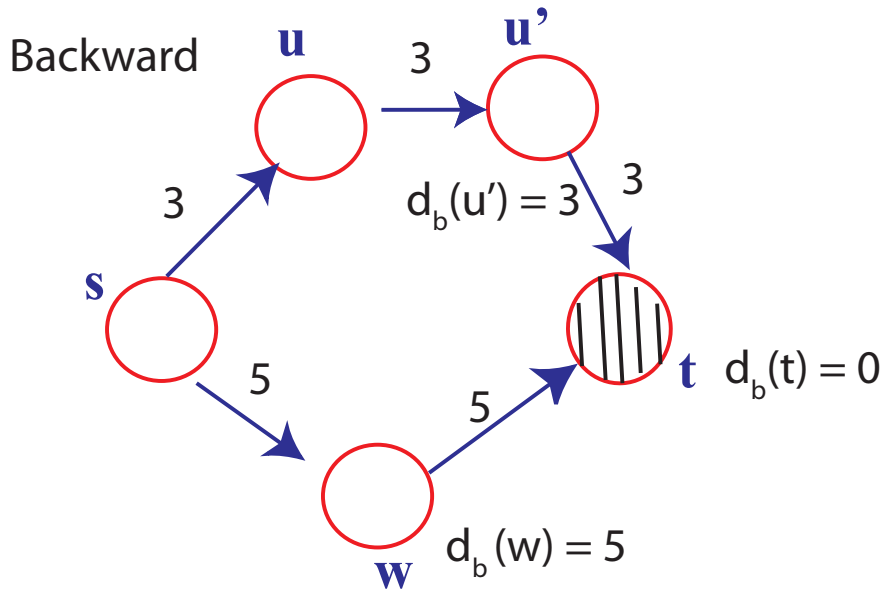
Bi-directional Djikstra

- Alternate forward search from s, backward search from t (follow edges backward)
- $d_f(u)$ distances for forward search; $d_b(u)$ distances for backward search
- Algorithm terminates when some vertex w has been processed, i.e., deleted from the queue of both searches, Q_f and Q_b

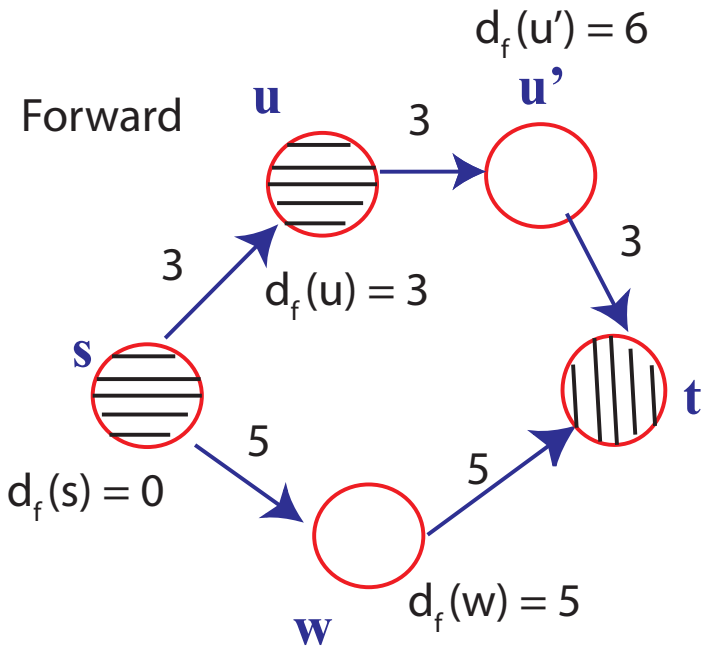


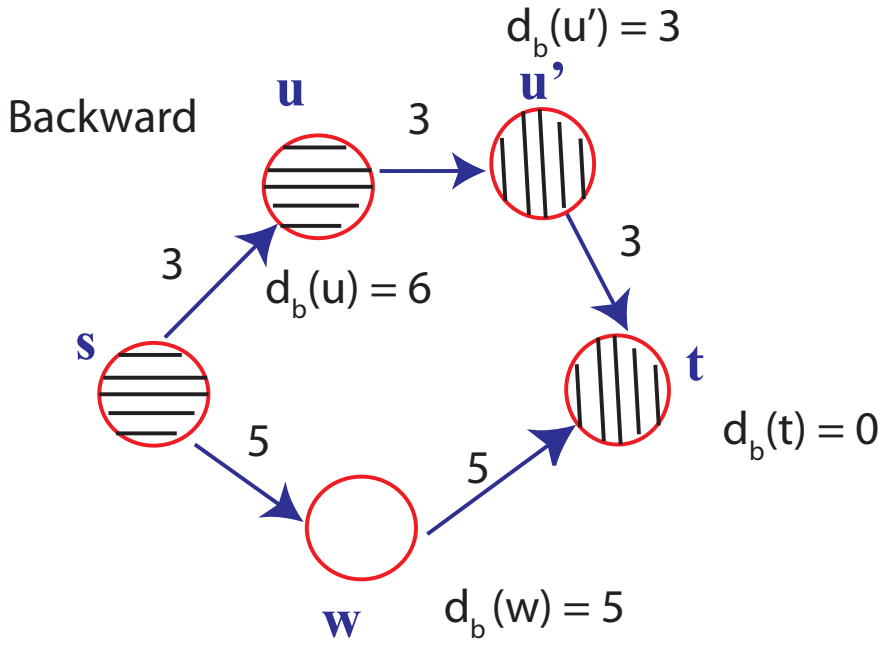
Bi-directional Dijkstra, example



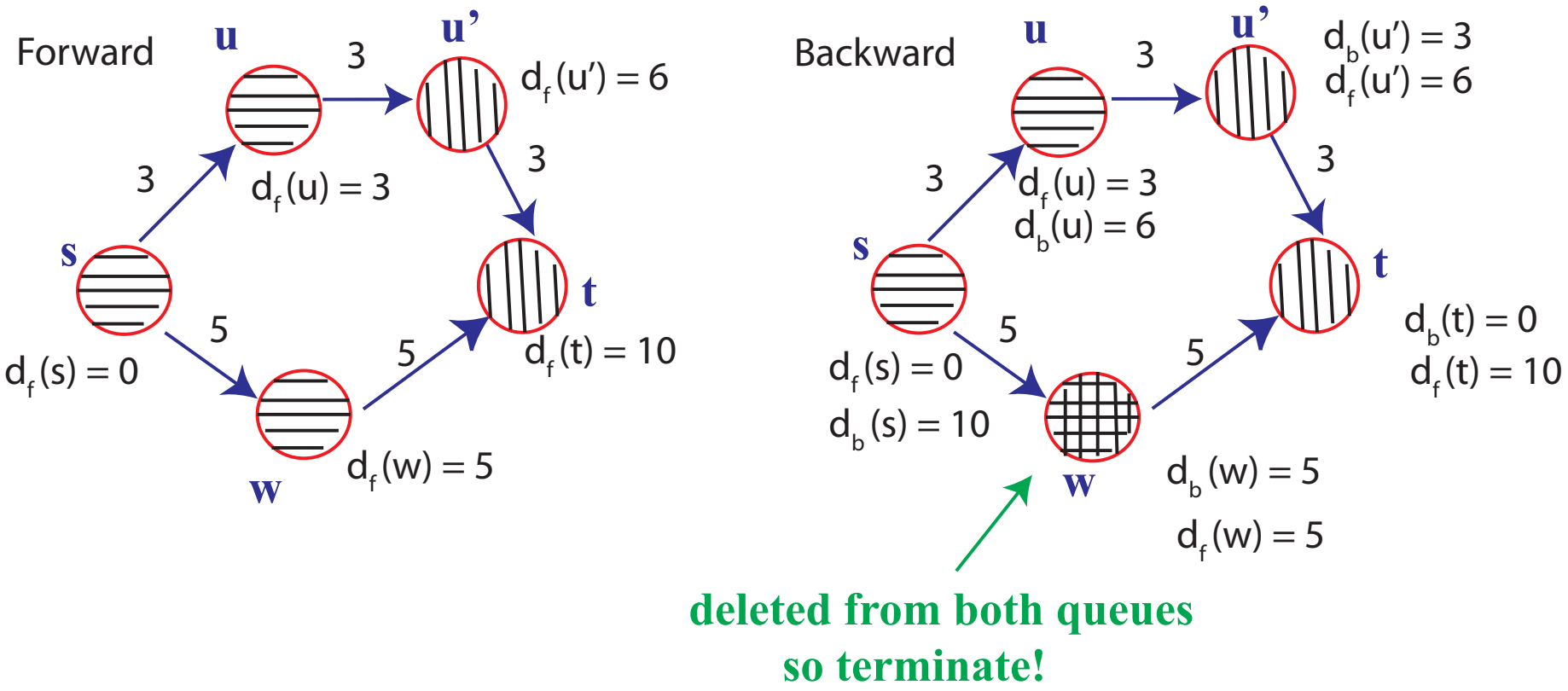


Bi-directional Dijkstra, example



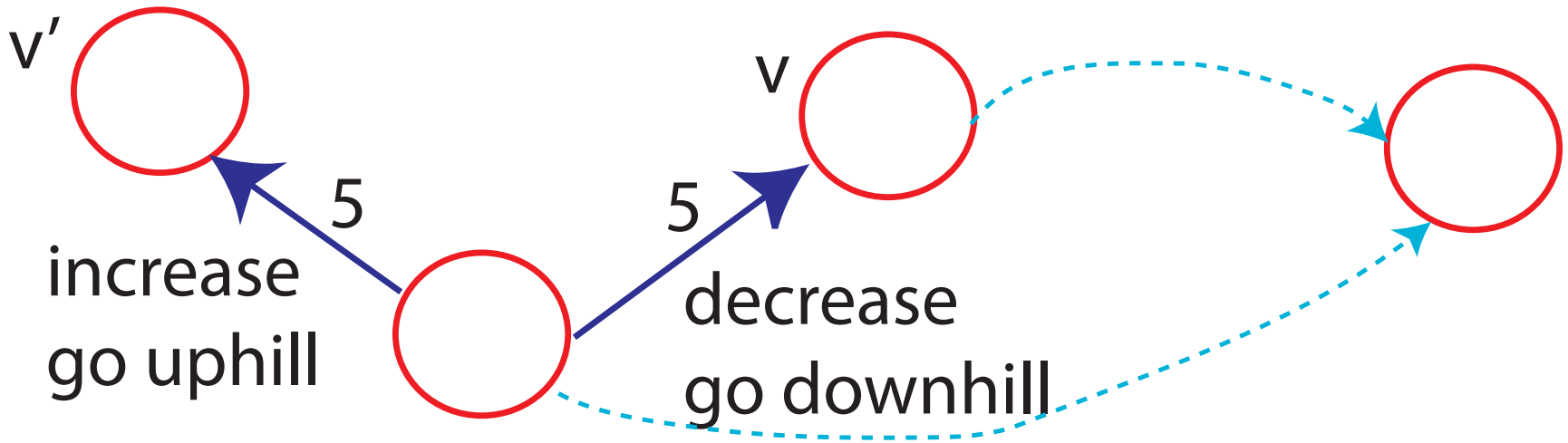


Bi-directional Dijkstra, example



Goal-directed search or A*

Idea: use a "potential function" $\lambda_t(u)$ over vertices to make the target vertex t "more attractive"



Implementation: Modify edge weights as follow:

$$w^*(u,v) = w(u,v) - \lambda_t(u) + \lambda_t(v)$$

Goal directed search or A*, cont.

Modify edge weights with potential function over vertices:

$$w^*(u,v) = w(u,v) - \lambda_t(u) + \lambda_t(v)$$

$\Rightarrow$for any path p between s and t we have:

$$w^*(p) = w(p) - \lambda_t(s) + \lambda_t(t)$$

$\Rightarrow$a path from s to t is a shortest path under w^* iff it is a shortest path under w

Feasible potential and example

- Potential $\lambda_t(u)$ is feasible if

$$w^*(u,v) = w(u,v) - \lambda_t(u) + \lambda_t(v) \geq 0$$

- As a result can use Dijkstra with w^*

- Examples:
  - Euclidean plane
  - Landmarks