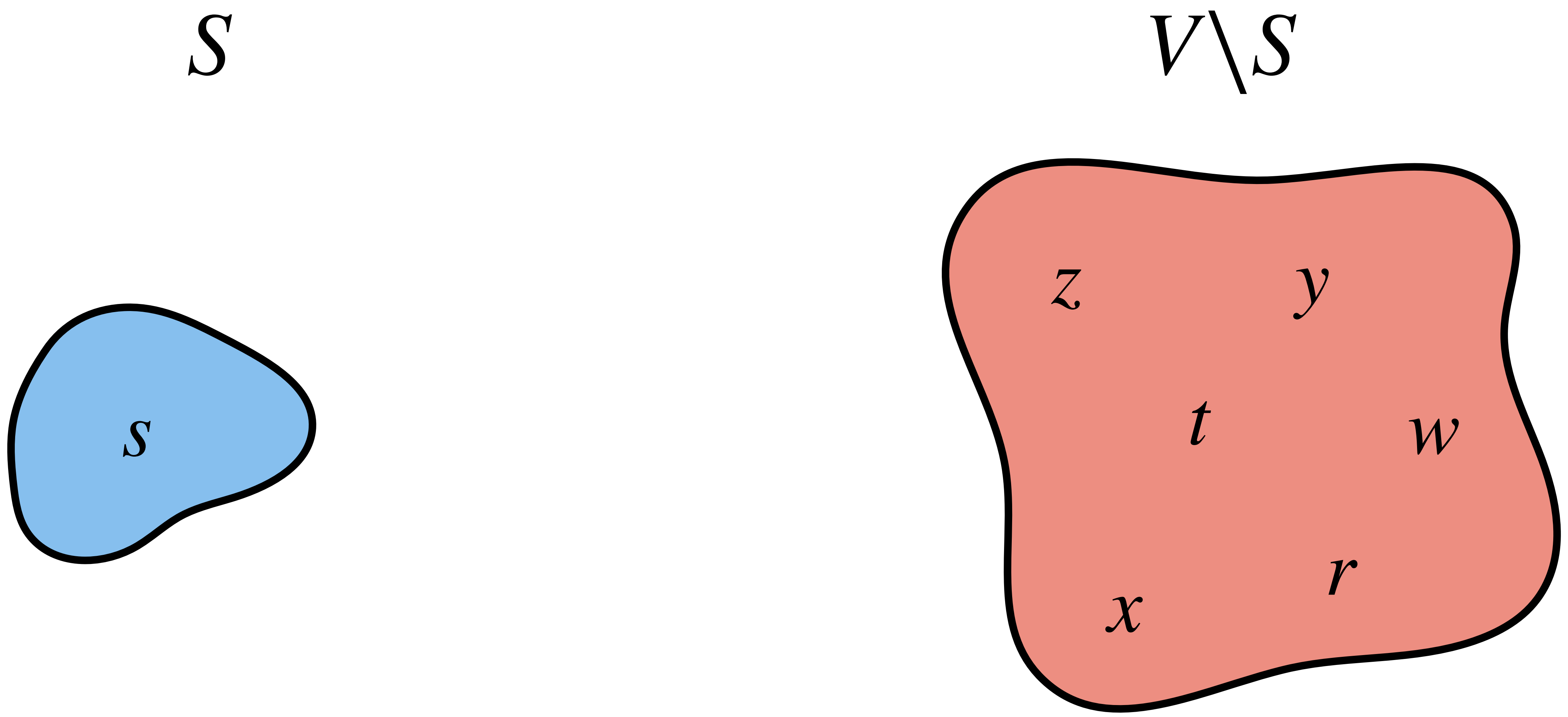
Lecture 16

Dijkstra (contd.), Flow Networks

Source: Introduction to Algorithms, CLRS and Kleinberg & Tardos

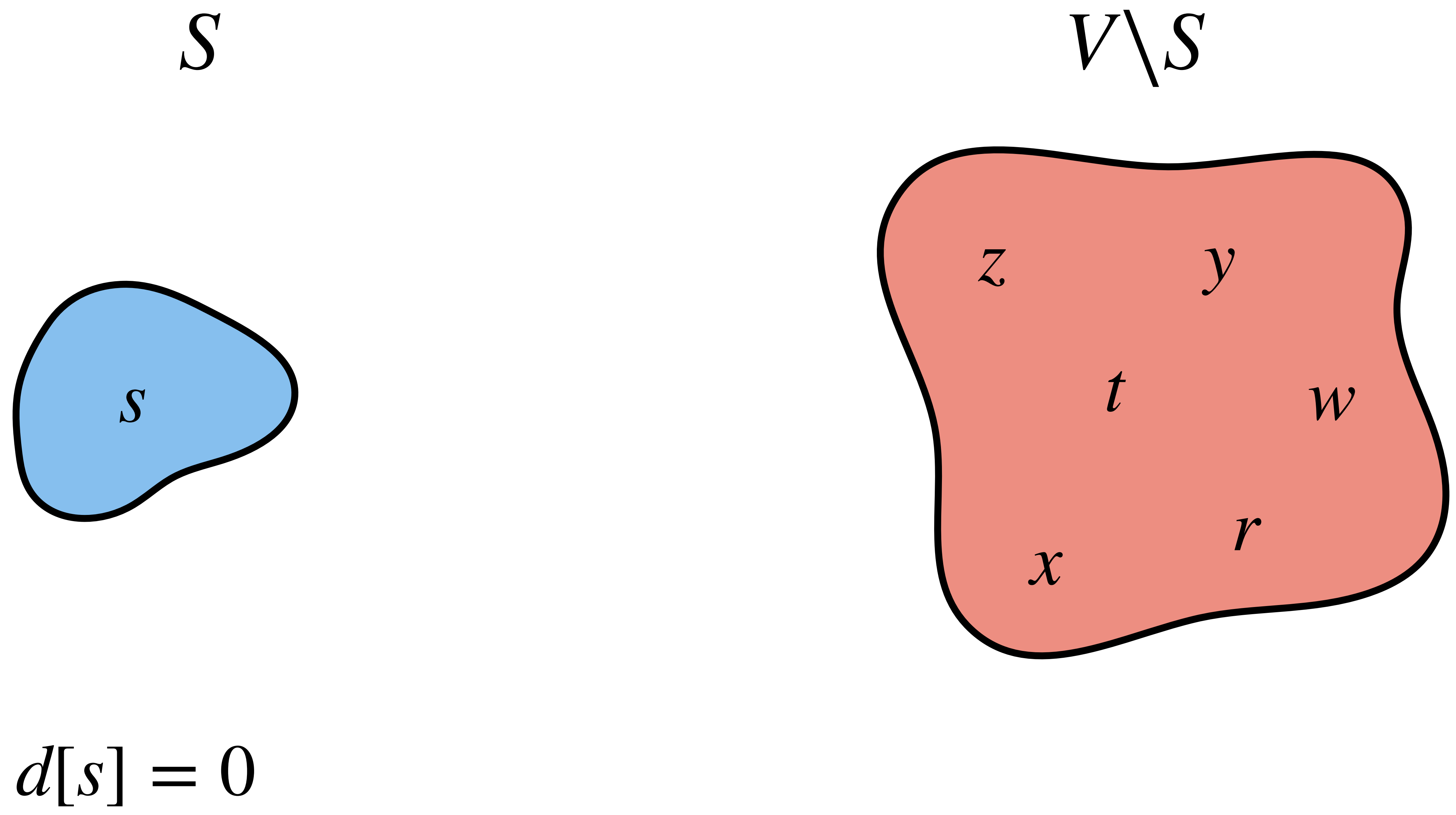
Dijkstra's Algorithm: Optimization

Computed $\pi[v]$ values in $V \backslash S$ in an efficient way.



Dijkstra's Algorithm: Optimization

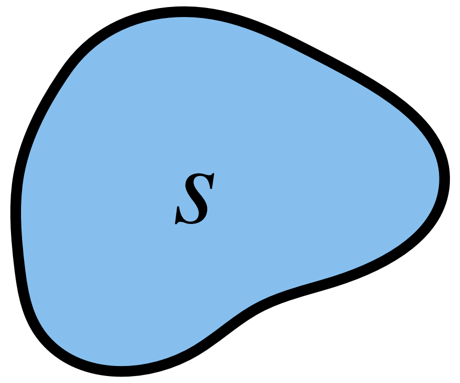
Computed $\pi[v]$ values in $V \backslash S$ in an efficient way.



Dijkstra's Algorithm: Optimization

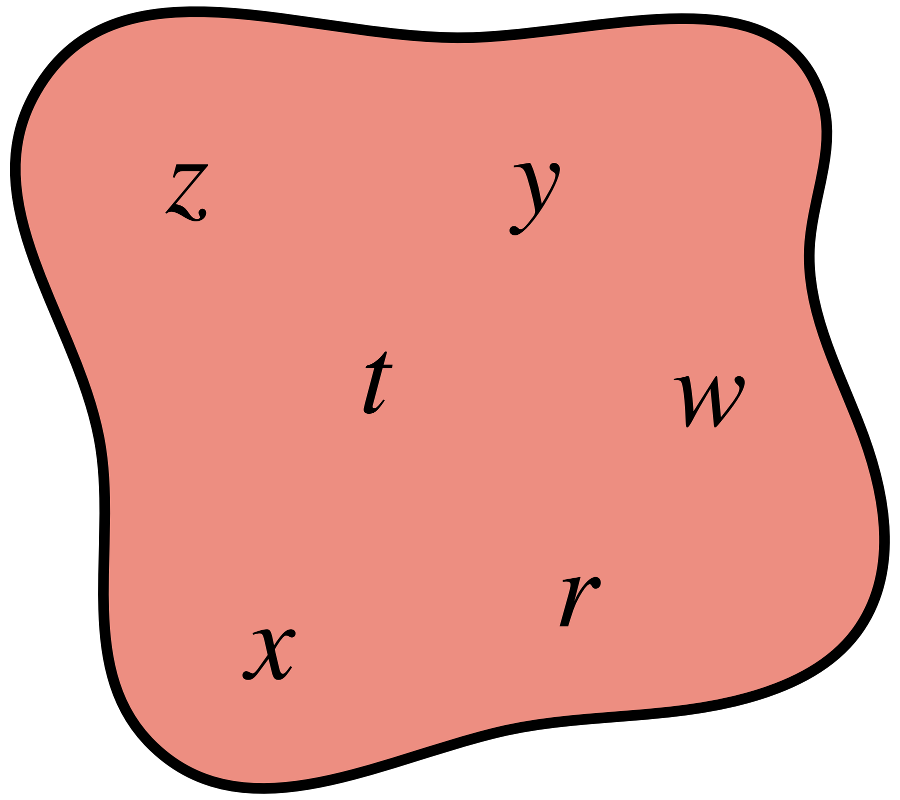
Computed $\pi[v]$ values in $V \backslash S$ in an efficient way.

S



$d[s] = 0$

$V \backslash S$

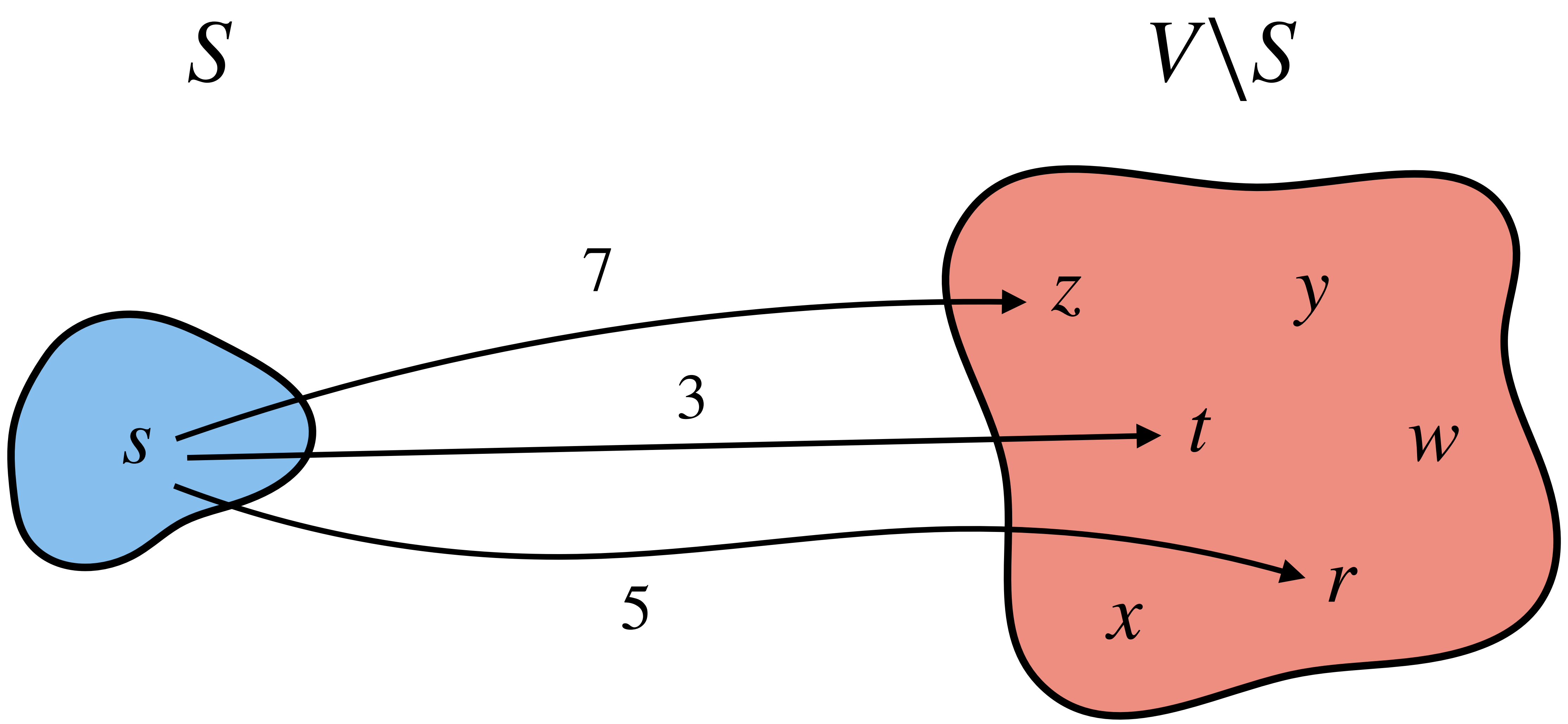


$\pi[z] = \infty,\ \pi[y] = \infty,\ \pi[t] = \infty,\ \pi[w] = \infty,$

$\pi[x] = \infty,\ \pi[r] = \infty$

Dijkstra's Algorithm: Optimization

Computed $\pi[v]$ values in $V \backslash S$ in an efficient way.



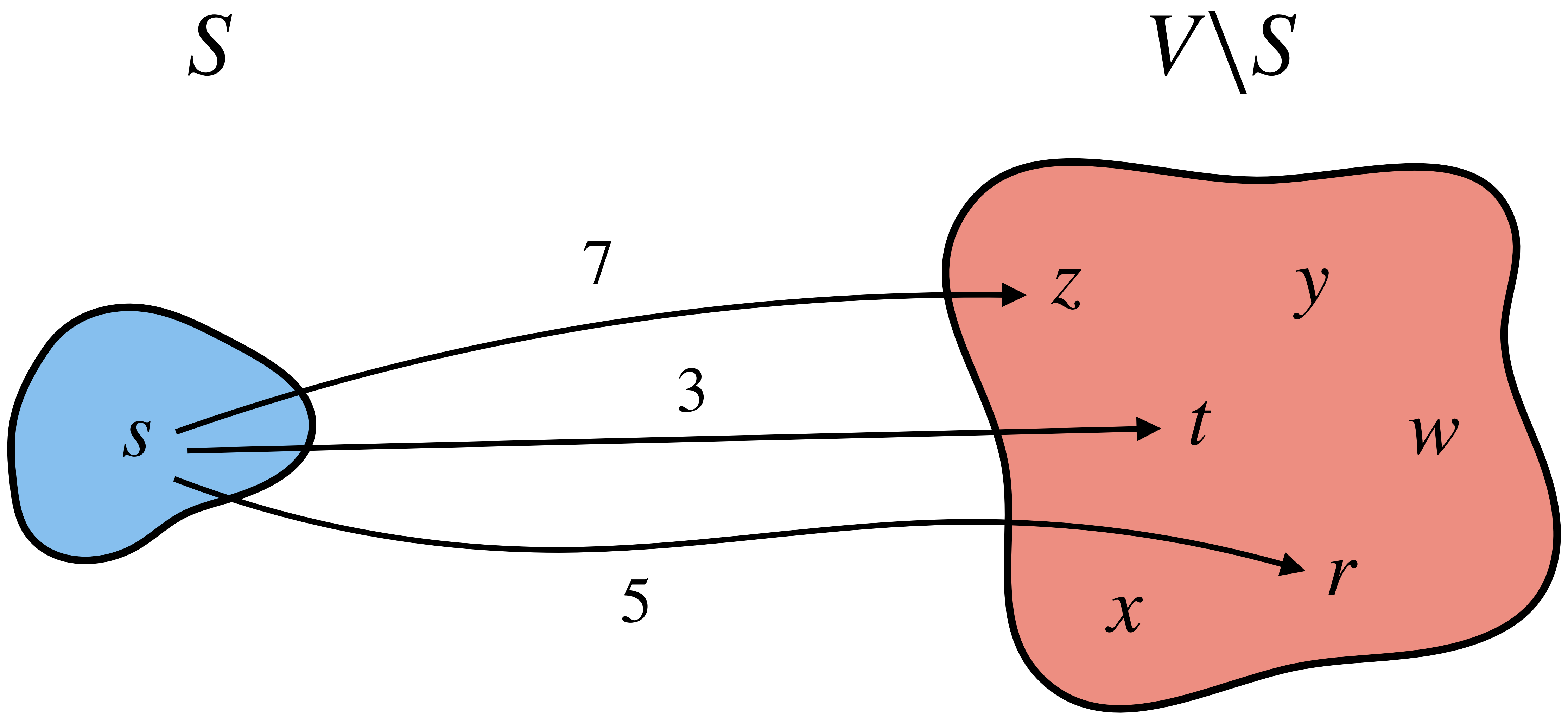
$d[s] = 0$

$\pi[z] = \infty, \pi[y] = \infty, \ \pi[t] = \infty, \ \pi[w] = \infty,$

$\pi[x] = \infty, \ \pi[r] = \infty$

Dijkstra's Algorithm: Optimization

Computed $\pi[v]$ values in $V \backslash S$ in an efficient way.



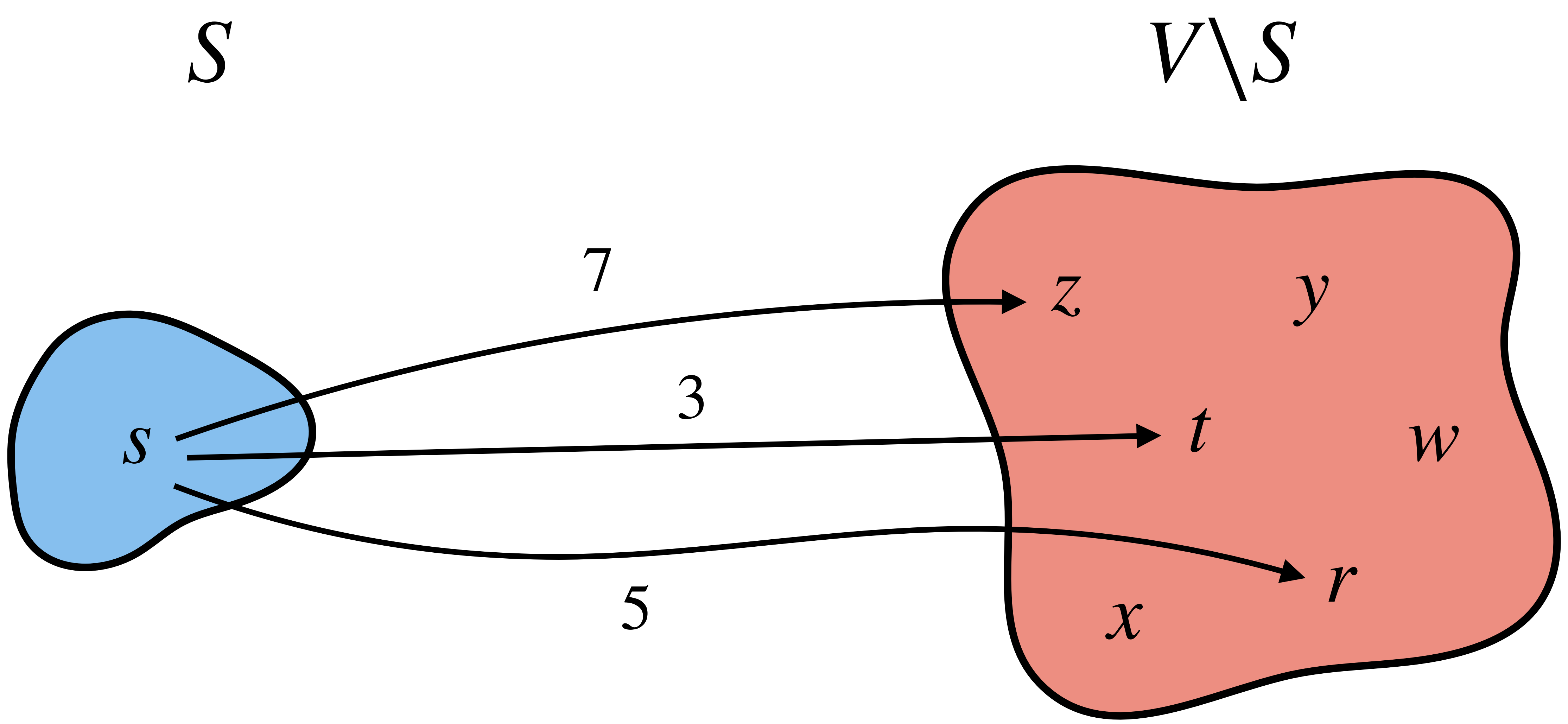
$d[s] = 0$

$\pi[z] = 7, \ \pi[y] = \infty, \ \pi[t] = \infty, \ \pi[w] = \infty,$

$\pi[x] = \infty, \ \pi[r] = \infty$

Dijkstra's Algorithm: Optimization

Computed $\pi[v]$ values in $V \backslash S$ in an efficient way.



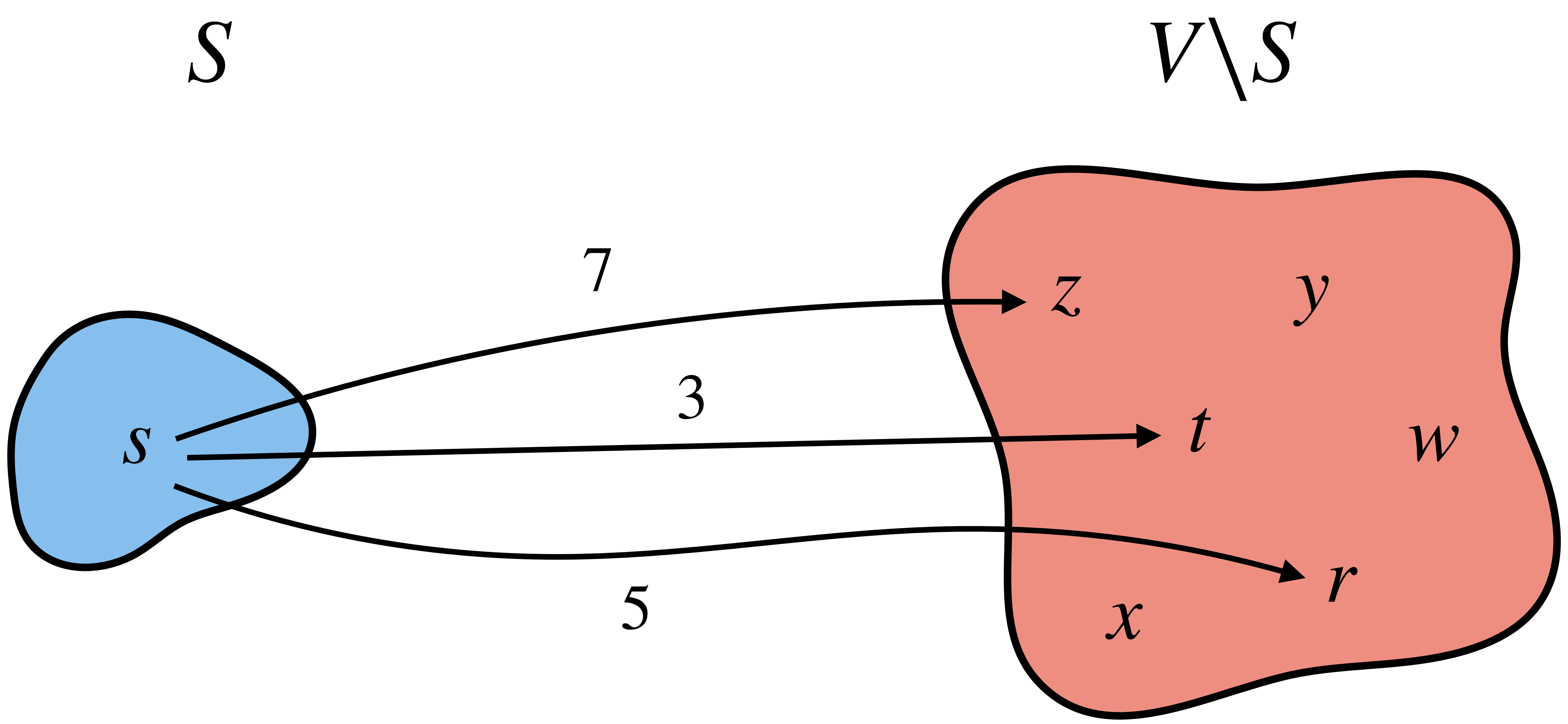
$d[s] = 0$

$\pi[z] = 7,\ \pi[y] = \infty,\ \pi[t] = 3,\ \pi[w] = \infty,$

$\pi[x] = \infty,\ \pi[r] = \infty$

Dijkstra’s Algorithm: Optimization

Computed $\pi[v]$ values in $V \backslash S$ in an efficient way.



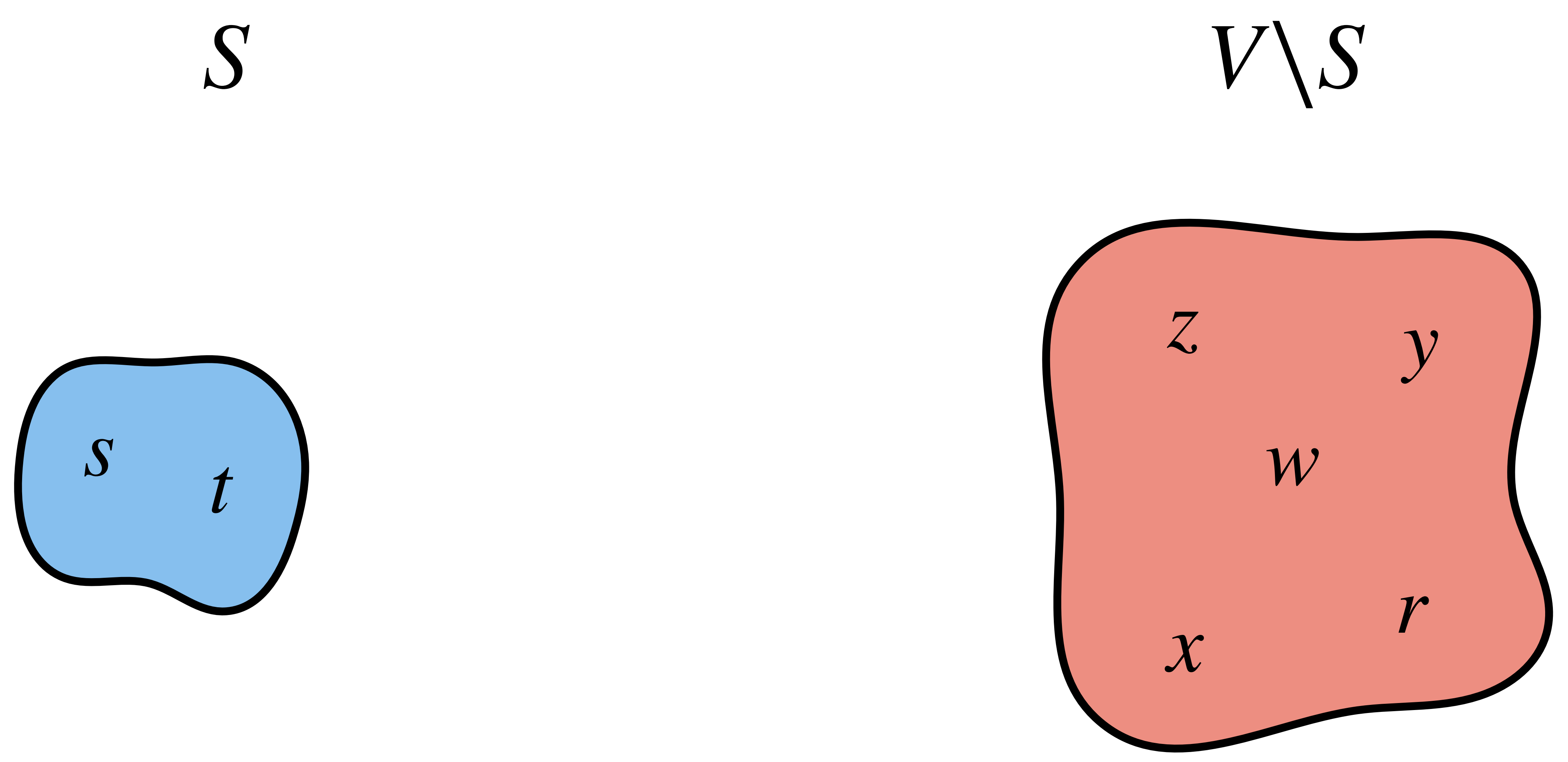
$d[s] = 0$

$\pi[z] = 7,\ \pi[y] = \infty,\ \pi[t] = 3,\ \pi[w] = \infty,$

$\pi[x] = \infty,\ \pi[r] = 5$

Dijkstra's Algorithm: Optimization

Computed $\pi[v]$ values in $V \backslash S$ in an efficient way.

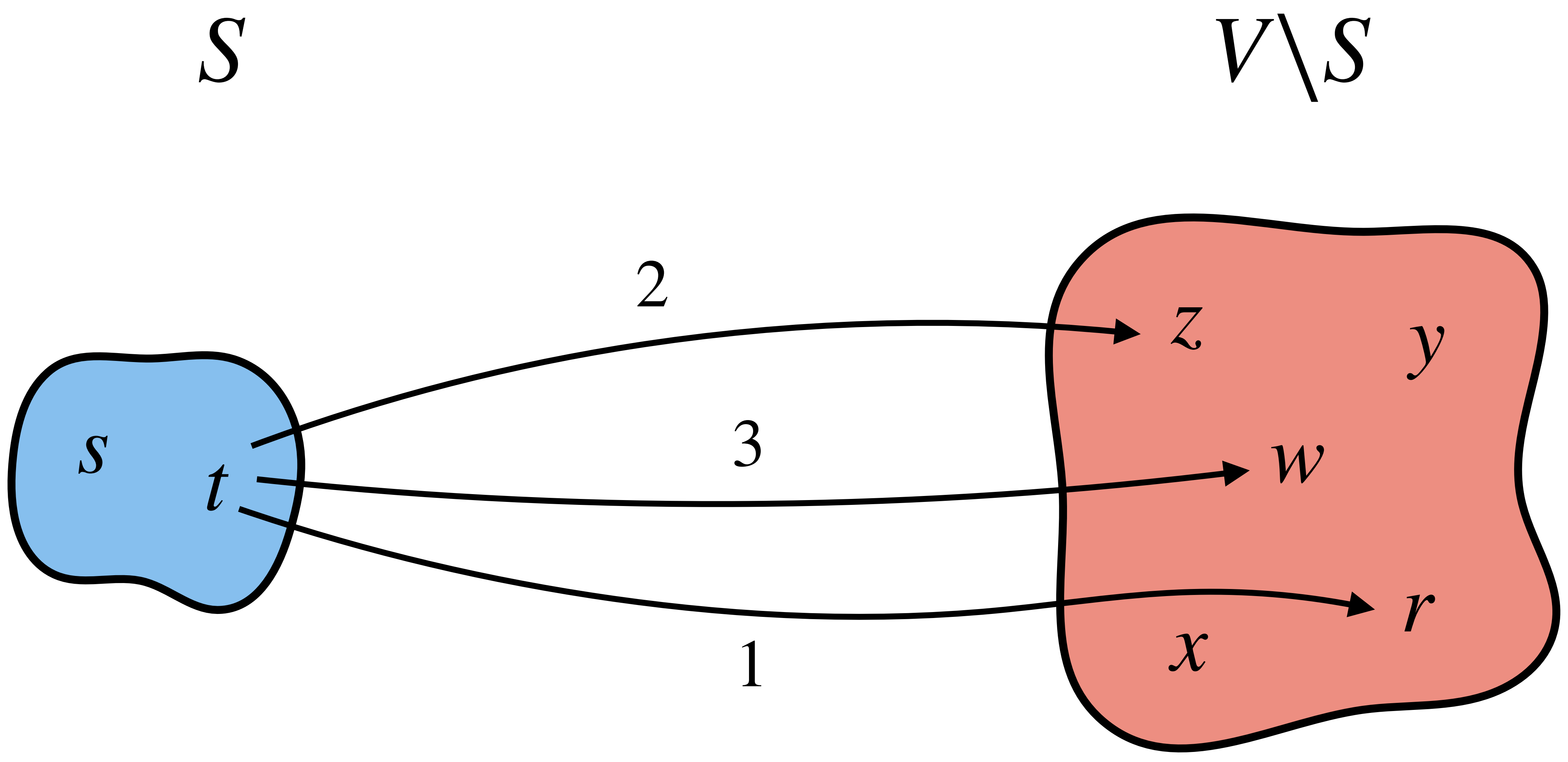


$d[s] = 0, d[t] = 3$

$\pi[z] = 7,\ \pi[y] = \infty,\ \pi[w] = \infty,$
$\pi[x] = \infty,\ \pi[r] = 5$

Dijkstra's Algorithm: Optimization

Computed $\pi[v]$ values in $V \backslash S$ in an efficient way.



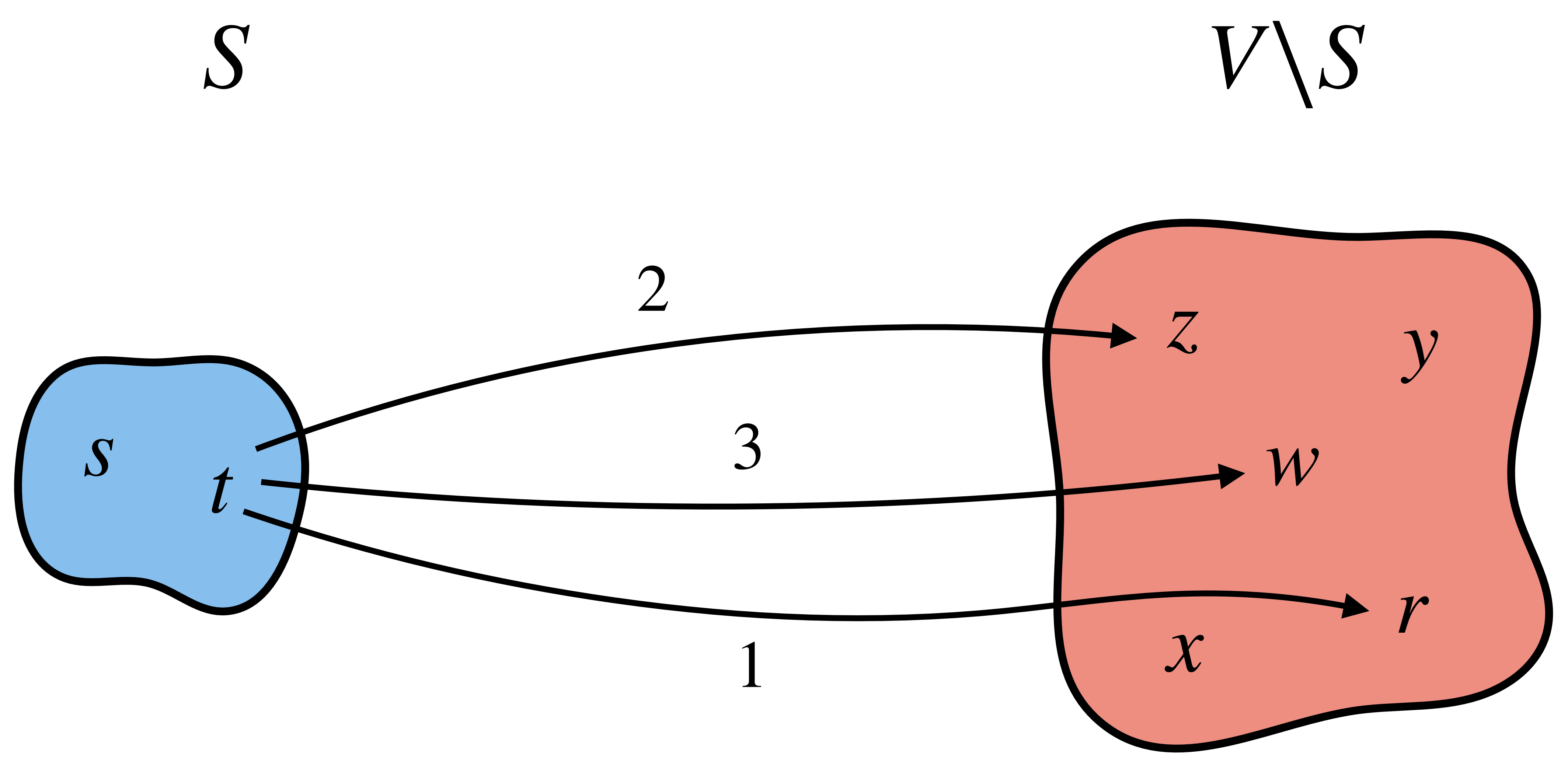
$d[s] = 0, d[t] = 3$

$\pi[z] = 7,\ \pi[y] = \infty,\ \pi[w] = \infty,$

$\pi[x] = \infty,\ \pi[r] = 5$

Dijkstra's Algorithm: Optimization

Computed $\pi[v]$ values in $V \backslash S$ in an efficient way.



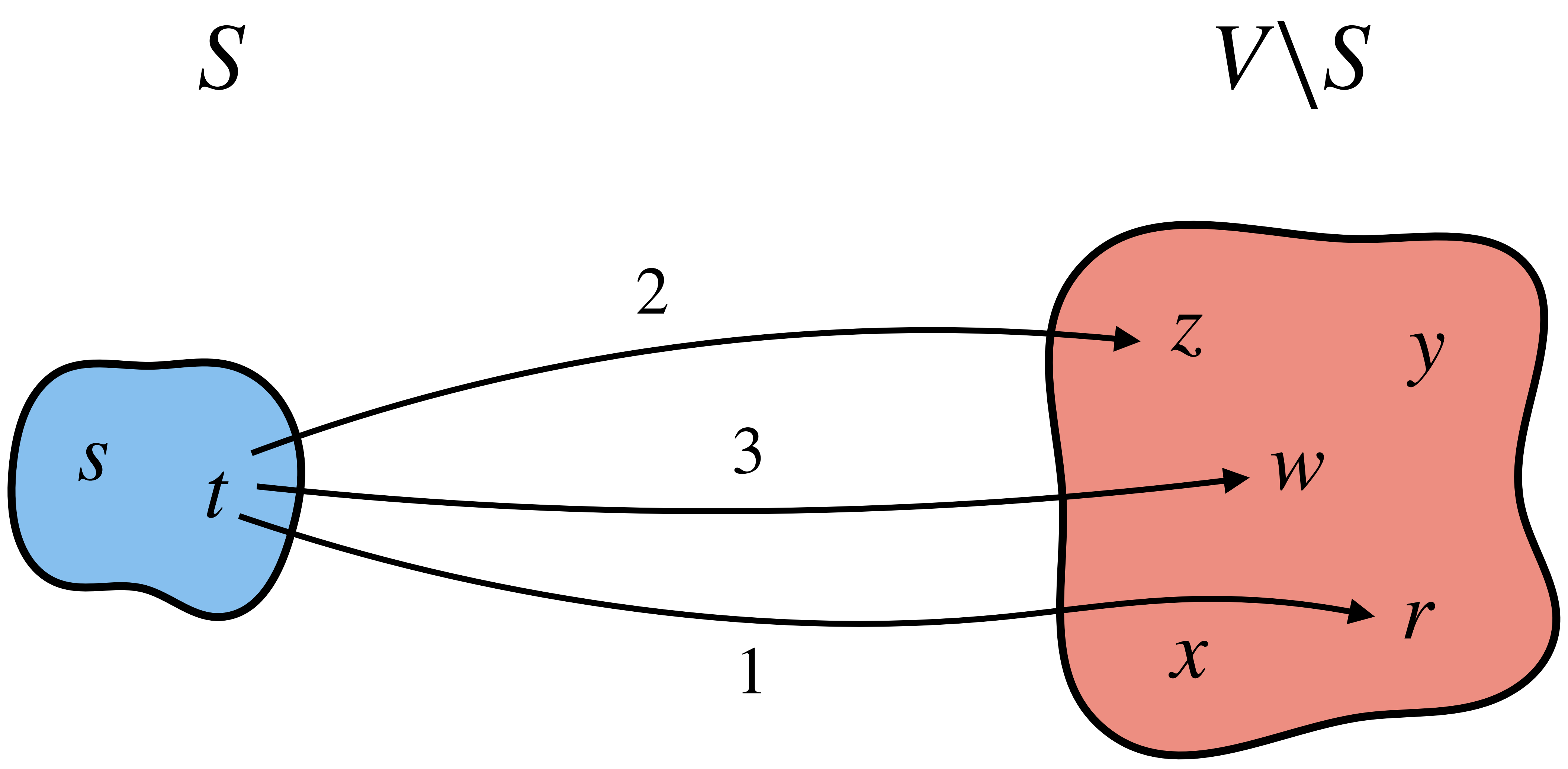
$d[s] = 0, d[t] = 3$

$\pi[z] = 5, \ \pi[y] = \infty, \ \pi[w] = \infty,$

$\pi[x] = \infty, \ \pi[r] = 5$

Dijkstra's Algorithm: Optimization

Computed $\pi[v]$ values in $V \backslash S$ in an efficient way.



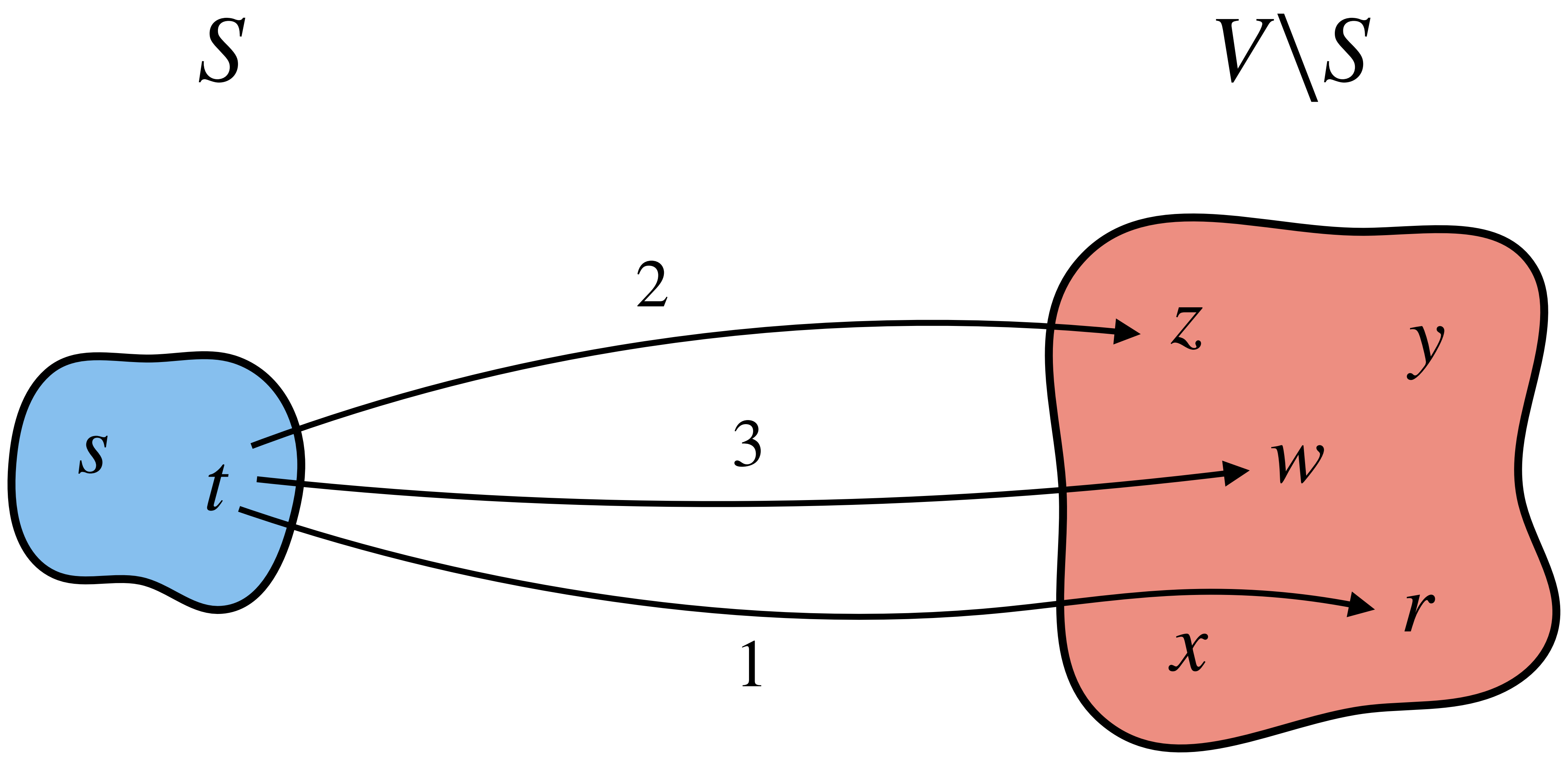
$d[s] = 0, d[t] = 3$

$\pi[z] = 5,\ \pi[y] = \infty,\ \pi[w] = 6,$

$\pi[x] = \infty,\ \pi[r] = 5$

Dijkstra's Algorithm: Optimization

Computed $\pi[v]$ values in $V \backslash S$ in an efficient way.



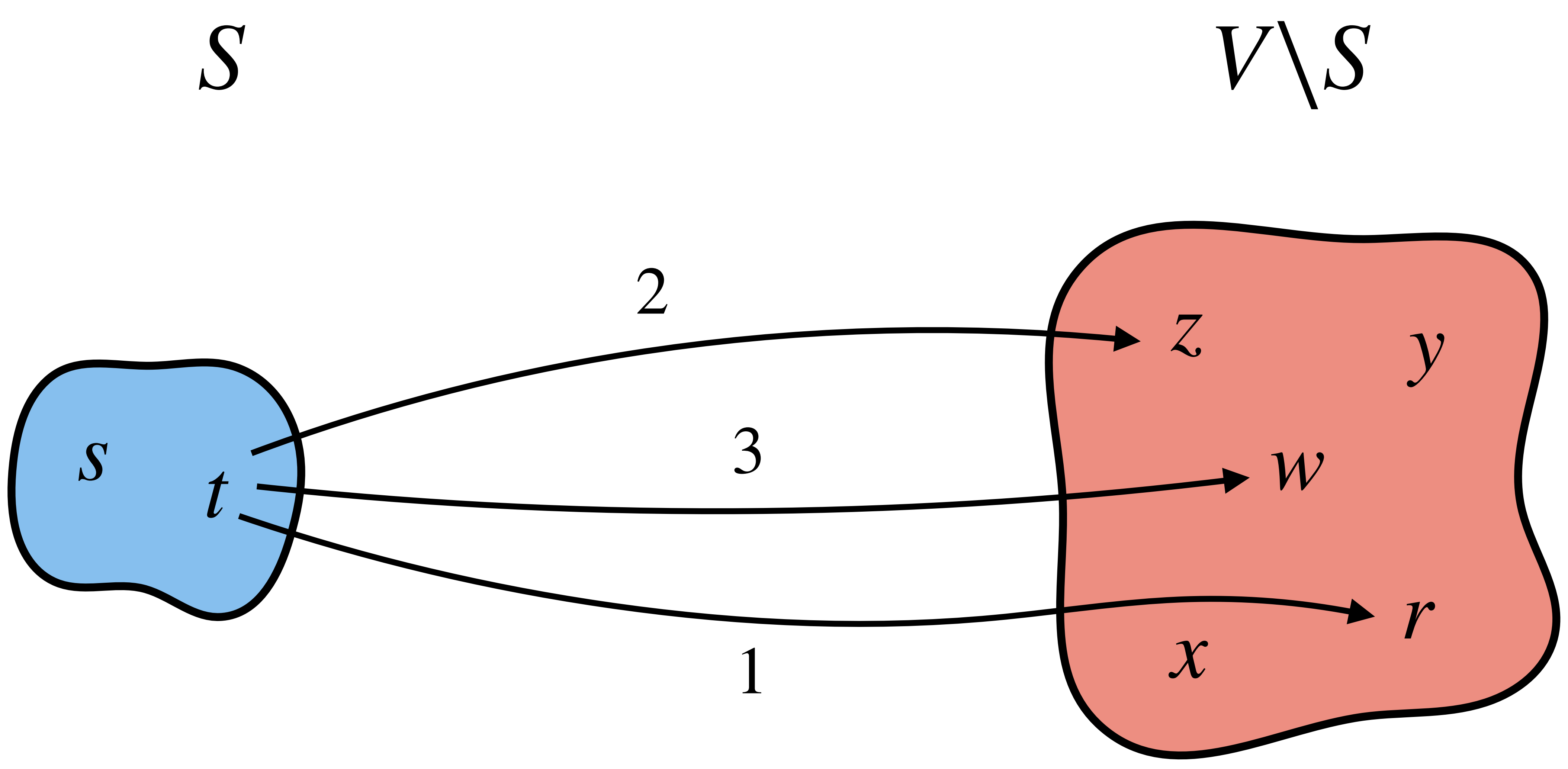
$d[s] = 0, d[t] = 3$

$\pi[z] = 5, \ \pi[y] = \infty, \ \pi[w] = 6,$

$\pi[x] = \infty, \ \pi[r] = 4$

Dijkstra's Algorithm: Optimization

Computed $\pi[v]$ values in $V \backslash S$ in an efficient way.



$d[s] = 0, d[t] = 3$

$\pi[z] = 5, \ \pi[y] = \infty, \ \pi[w] = 6,$

$\pi[x] = \infty, \ \pi[r] = 4$

What data structure is suitable to keep updating π values and removing the one with minimum?

Dijkstra's Algorithm: Implementation

Dijkstra's Algorithm: Implementation

On $V \setminus S$ we are performing two operations:

Dijkstra's Algorithm: Implementation

On $V \backslash S$ we are performing two operations:

- Maintaining and updating π values.

Dijkstra's Algorithm: Implementation

On $V \setminus S$ we are performing two operations:

- Maintaining and updating π values.
- Removing the element with minimum π value.

Dijkstra's Algorithm: Implementation

On $V \backslash S$ we are performing two operations:

- Maintaining and updating π values.
- Removing the element with minimum π value.

Min-priority queue can perform the above operations in $O(\log n)$ time.

Dijkstra's Algorithm: Implementation

On $V \backslash S$ we are performing two operations:

- Maintaining and updating π values.
- Removing the element with minimum π value.

Min-priority queue can perform the above operations in $O(\log n)$ time.

- $\textbf{Insert}(Q, u)$: Inserts element element u with a key in Q

Dijkstra's Algorithm: Implementation

On $V \backslash S$ we are performing two operations:

- Maintaining and updating π values.
- Removing the element with minimum π value.

Min-priority queue can perform the above operations in $O(\log n)$ time.

- $\textbf{Insert}(Q, u)$: Inserts element element u with a key in Q
- $\textbf{Extract-Min}(Q)$: Removes the element with minimum key from Q.

Dijkstra's Algorithm: Implementation

On $V \backslash S$ we are performing two operations:

- Maintaining and updating π values.
- Removing the element with minimum π value.

Min-priority queue can perform the above operations in $O(\log n)$ time.

- $\mathbf{Insert}(Q, u)$: Inserts element element u with a key in Q
- $\mathbf{Extract\text{-}Min}(Q)$: Removes the element with minimum key from Q.
- $\mathbf{Decrease\text{-}Key}(Q, u, d)$: Decreases the key of u to d.

Dijkstra's Algorithm: Implementation

On $V \backslash S$ we are performing two operations:

- Maintaining and updating π values.
- Removing the element with minimum π value.

Min-priority queue can perform the above operations in $O(\log n)$ time.

- $\textbf{Insert}(Q, u)$: Inserts element element u with a key in Q
- $\textbf{Extract-Min}(Q)$: Removes the element with minimum key from Q.
- $\textbf{Decrease-Key}(Q, u, d)$: Decreases the key of u to d.

Idea: Form a min-priority queue of $V \backslash S$ where $keys$ are π values.

Dijkstra's Algorithm: Pseudocode

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s)**:**

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s)**:**

1. $S = \varnothing, Q = \varnothing$ // $S = \{u \mid u\text{'s distance is computed}\}$ *and* Q *is a min-priority queue*

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s):

1. $S = \varnothing, Q = \varnothing$ // $S = \{u \mid u$'s distance is computed$\}$ and Q is a min-priority queue
2. Create array $d[1 : |V|]$, $\pi[1 : |V|]$ // for storing distance and π values

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s):

1. $S = \emptyset, Q = \emptyset$ // $S = \{u \mid u\text{'s distance is computed}\}$ *and* Q *is a min-priority queue*
2. Create array $d[1 : |V|], \pi[1 : |V|]$ // *for storing distance and* π *values*
3. **for** each vertex $v \in V(G)$

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s):

$S = \emptyset, Q = \emptyset$ *// $S = \{u \mid u's$ distance is computed$\}$ and Q is a min-priority queue*

Create array $d[1 : |V|], \pi[1 : |V|]$ *// for storing distance and π values*

for each vertex $v \in V(G)$

    $d[v] = \infty, \pi[v] = \infty$, **Insert**$(Q, v, \pi[v])$ *// Q stores vertices with their π value as key.*

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s):

1. $S = \emptyset, Q = \emptyset$ *// $S = \{u \mid u$'s distance is computed$\}$ and Q is a min-priority queue*
2. Create array $d[1:|V|], \pi[1:|V|]$ *// for storing distance and π values*
3. **for** each vertex $v \in V(G)$
4. $d[v] = \infty, \pi[v] = \infty$, **Insert**$(Q, v, \pi[v])$ *// Q stores vertices with their π value as key.*
5. $\pi[s] = 0$, **Decrease-Key**$(Q, s, 0)$

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s):

$S = \varnothing, Q = \varnothing$ *// $S = \{u \mid u's$ distance is computed$\}$ and Q is a min-priority queue*

Create array $d[1 : |V|], \pi[1 : |V|]$ *// for storing distance and π values*

for each vertex $v \in V(G)$

    $d[v] = \infty, \pi[v] = \infty$, **Insert**$(Q, v, \pi[v])$ *// Q stores vertices with their π value as key.*

$\pi[s] = 0$, **Decrease-Key**$(Q, s, 0)$

while $Q \neq \varnothing$

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s):

$S = \emptyset, Q = \emptyset$ *// $S = \{u \mid u$'s distance is computed$\}$ and Q is a min-priority queue*

Create array $d[1 : |V|], \pi[1 : |V|]$ *// for storing distance and π values*

for each vertex $v \in V(G)$

    $d[v] = \infty, \pi[v] = \infty$, **Insert**$(Q, v, \pi[v])$ *// Q stores vertices with their π value as key.*

$\pi[s] = 0$, **Decrease-Key**$(Q, s, 0)$

while $Q \neq \emptyset$

    $u =$ **Extract-Min**(Q) *// u's distance is computed*

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s):

1. $S = \emptyset, Q = \emptyset$ *// $S = \{u \mid u\text{'s distance is computed}\}$ and Q is a min-priority queue*
2. Create array $d[1 : |V|]$, $\pi[1 : |V|]$ *// for storing distance and π values*
3. **for** each vertex $v \in V(G)$
4. $\quad d[v] = \infty, \pi[v] = \infty$, **Insert**$(Q, v, \pi[v])$ *// Q stores vertices with their π value as key.*
5. $\pi[s] = 0$, **Decrease-Key**$(Q, s, 0)$
6. **while** $Q \neq \emptyset$
7. $\quad u =$ **Extract-Min**(Q) *// u's distance is computed*
8. $\quad S = S \cup \{u\}, d[u] = \pi[u]$ *// Add u to S and update its d*

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s):

1. $S = \varnothing, Q = \varnothing$ *// $S = \{u \mid u's$ distance is computed$\}$ and Q is a min-priority queue*
2. Create array $d[1 : |V|], \pi[1 : |V|]$ *// for storing distance and π values*
3. **for** each vertex $v \in V(G)$
4. $\quad d[v] = \infty, \pi[v] = \infty,$ **Insert**$(Q, v, \pi[v])$ *// Q stores vertices with their π value as key.*
5. $\pi[s] = 0,$ **Decrease-Key**$(Q, s, 0)$
6. **while** $Q \neq \varnothing$
7. $\quad u =$ **Extract-Min**(Q) *// u's distance is computed*
8. $\quad S = S \cup \{u\}, d[u] = \pi[u]$ *// Add u to S and update its d*
9. $\quad$ **for** each vertex $v \in Adj[u]$

Dijkstra's Algorithm: Pseudocode

```
Dijkstra(G, s):
    S = ∅, Q = ∅                          // S = {u | u's distance is computed} and Q is a min-priority queue
    Create array d[1 : |V|], π[1 : |V|]   // for storing distance and π values
    for each vertex v ∈ V(G)
        d[v] = ∞, π[v] = ∞, Insert(Q, v, π[v])   // Q stores vertices with their π value as key.
    π[s] = 0, Decrease-Key(Q, s,0)
    while Q ≠ ∅
        u = Extract-Min(Q)                // u's distance is computed
        S = S ∪ {u}, d[u] = π[u]          // Add u to S and update its d
        for each vertex v ∈ Adj[u]
            if π[v] > d[u] + w(u, v)      // Recall setting π[v] = Min(π[v], d[u] + w(u, v))
```

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s):

$S = \varnothing, Q = \varnothing$ *// $S = \{u \mid u's$ distance is computed$\}$ and Q is a min-priority queue*

Create array $d[1 : |V|], \pi[1 : |V|]$ *// for storing distance and π values*

for each vertex $v \in V(G)$

$d[v] = \infty, \pi[v] = \infty$, **Insert**$(Q, v, \pi[v])$ *// Q stores vertices with their π value as key.*

$\pi[s] = 0$, **Decrease-Key**$(Q, s, 0)$

while $Q \neq \varnothing$

$u =$ **Extract-Min**(Q) *// u's distance is computed*

$S = S \cup \{u\}, d[u] = \pi[u]$ *// Add u to S and update its d*

for each vertex $v \in Adj[u]$

if $\pi[v] > d[u] + w(u, v)$ *// Recall setting $\pi[v] =$ **Min**$(\pi[v], d[u] + w(u, v))$*

$\pi[v] = d[u] + w(u, v)$

Dijkstra's Algorithm: Pseudocode

```
Dijkstra(G, s):
S = ∅, Q = ∅                  // S = {u | u's distance is computed} and Q is a min-priority queue
Create array d[1 : |V|], π[1 : |V|]        // for storing distance and π values
for each vertex v ∈ V(G)
    d[v] = ∞, π[v] = ∞, Insert(Q, v, π[v])        // Q stores vertices with their π value as key.
π[s] = 0, Decrease-Key(Q, s,0)
while Q ≠ ∅
    u = Extract-Min(Q)          // u's distance is computed
    S = S ∪ {u}, d[u] = π[u]          // Add u to S and update its d
    for each vertex v ∈ Adj[u]
        if  π[v] > d[u] + w(u, v)        // Recall setting π[v] =  Min(π[v], d[u] + w(u, v))
            π[v] = d[u] + w(u, v)
            Decrease-Key(Q, v, π[v])
```

Dijkstra's Algorithm: Pseudocode

```
Dijkstra(G, s):
S = ∅, Q = ∅
Create array d[1 : |V|], π[1 : |V|]
for each vertex v ∈ V(G)
    d[v] = ∞, π[v] = ∞, Insert(Q, v, π[v])
π[s] = 0, Decrease-Key(Q, s,0)
while Q ≠ ∅
    u = Extract-Min(Q)
    S = S ∪ {u}, d[u] = π[u]
    for each vertex v ∈ Adj[u]
        if  π[v] > d[u] + w(u, v)
            π[v] = d[u] + w(u, v)
            Decrease-Key(Q, v, π[v])
```

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s):

$S = \varnothing, Q = \varnothing$
Create array $d[1:|V|], \pi[1:|V|]$
for each vertex $v \in V(G)$
    $d[v] = \infty, \pi[v] = \infty$, **Insert**$(Q, v, \pi[v])$
$\pi[s] = 0$, **Decrease-Key**$(Q, s, 0)$
while $Q \neq \varnothing$
    $u =$ **Extract-Min**(Q)
    $S = S \cup \{u\}, d[u] = \pi[u]$
    for each vertex $v \in Adj[u]$
        if $\pi[v] > d[u] + w(u, v)$
            $\pi[v] = d[u] + w(u, v)$
            Decrease-Key$(Q, v, \pi[v])$

What if $v \in S$?

Dijkstra's Algorithm: Pseudocode

```
Dijkstra(G, s):
1.   S = ∅, Q = ∅
2.   Create array d[1 : |V|], π[1 : |V|]
3.   for each vertex v ∈ V(G)
4.       d[v] = ∞, π[v] = ∞, Insert(Q, v, π[v])
5.   π[s] = 0, Decrease-Key(Q, s,0)
6.   while Q ≠ ∅
7.       u = Extract-Min(Q)
8.       S = S ∪ {u}, d[u] = π[u]
9.       for each vertex v ∈ Adj[u]
10.          if  π[v] > d[u] + w(u, v)
11.              π[v] = d[u] + w(u, v)
12.              Decrease-Key(Q, v, π[v])
```

What if $v \in S$? Then line 10 condition will be false as

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s):

1. $S = \varnothing, Q = \varnothing$
2. Create array $d[1 : |V|], \pi[1 : |V|]$
3. **for** each vertex $v \in V(G)$
4. $\quad d[v] = \infty, \pi[v] = \infty$, **Insert**$(Q, v, \pi[v])$
5. $\pi[s] = 0$, **Decrease-Key**$(Q, s, 0)$
6. **while** $Q \neq \varnothing$
7. $\quad u =$ **Extract-Min**(Q)
8. $\quad S = S \cup \{u\}, d[u] = \pi[u]$
9. $\quad$ **for** each vertex $v \in Adj[u]$
10. $\quad\quad$ **if** $\pi[v] > d[u] + w(u, v)$
11. $\quad\quad\quad \pi[v] = d[u] + w(u, v)$
12. $\quad\quad\quad$ **Decrease-Key**$(Q, v, \pi[v])$

What if $v \in S$? Then line 10 condition will be false as $\pi[v]$ became $\delta(s, v)$ earlier and cannot further decrease.

Dijkstra's Algorithm: Pseudocode

```
Dijkstra(G, s):
S = ∅, Q = ∅
Create array d[1 : |V|], π[1 : |V|]
for each vertex v ∈ V(G)
    d[v] = ∞, π[v] = ∞, Insert(Q, v, π[v])
π[s] = 0, Decrease-Key(Q, s,0)
while Q ≠ ∅
    u = Extract-Min(Q)
    S = S ∪ {u}, d[u] = π[u]
    for each vertex v ∈ Adj[u]
        if  π[v] > d[u] + w(u, v)
            π[v] = d[u] + w(u, v)
            Decrease-Key(Q, v, π[v])
```

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s):

$S = \emptyset, Q = \emptyset$

Create array $d[1 : |V|], \pi[1 : |V|]$

for each vertex $v \in V(G)$

  $d[v] = \infty, \pi[v] = \infty$, **Insert**$(Q, v, \pi[v])$

$\pi[s] = 0$, **Decrease-Key**$(Q, s, 0)$

while $Q \neq \emptyset$

  $u =$ **Extract-Min**(Q)

  $S = S \cup \{u\}, d[u] = \pi[u]$

  for each vertex $v \in Adj[u]$

    if $\pi[v] > d[u] + w(u, v)$

      $\pi[v] = d[u] + w(u, v)$

      Decrease-Key$(Q, v, \pi[v])$

What happens when $\pi[u]$ is ∞?

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s):

```
S = ∅, Q = ∅
Create array d[1 : |V|], π[1 : |V|]
for each vertex v ∈ V(G)
    d[v] = ∞, π[v] = ∞, Insert(Q, v, π[v])
π[s] = 0, Decrease-Key(Q, s,0)
while Q ≠ ∅
    u = Extract-Min(Q)
    S = S ∪ {u}, d[u] = π[u]
    for each vertex v ∈ Adj[u]
        if  π[v] > d[u] + w(u, v)
            π[v] = d[u] + w(u, v)
            Decrease-Key(Q, v, π[v])
```

What happens when $\pi[u]$ is ∞?

$d[u]$ becoming ∞ is fine.

Dijkstra's Algorithm: Pseudocode

Dijkstra(G, s):

1. $S = \varnothing, Q = \varnothing$
2. Create array $d[1 : |V|], \pi[1 : |V|]$
3. **for** each vertex $v \in V(G)$
4. $d[v] = \infty, \pi[v] = \infty$, **Insert**$(Q, v, \pi[v])$
5. $\pi[s] = 0$, **Decrease-Key**$(Q, s, 0)$
6. **while** $Q \neq \varnothing$
7. $u =$ **Extract-Min**(Q)
8. $S = S \cup \{u\}, d[u] = \pi[u]$
9.   **for** each vertex $v \in Adj[u]$
10.   **if** $\pi[v] > d[u] + w(u, v)$
11. $\pi[v] = d[u] + w(u, v)$
12.   **Decrease-Key**$(Q, v, \pi[v])$

What happens when $\pi[u]$ is ∞?

$d[u]$ becoming ∞ is fine.

Condition of line 10 will be false.

Dijkstra's Algorithm: Analysis

```
Dijkstra(G, s):
S = ∅, Q = ∅
Create array d[1 : |V|], π[1 : |V|]
for each vertex v ∈ V(G)
    d[v] = ∞, π[v] = ∞, Insert(Q, v, π[v])
π[s] = 0, Decrease-Key(Q, s,0)
while Q ≠ ∅
    u = Extract-Min(Q)
    S = S ∪ {u}, d[u] = π[u]
    for each vertex v ∈ Adj[u]
        if π[v] > d[u] + w(u, v)
            π[v] = d[u] + w(u, v)
            Decrease-Key(Q, v, π[v])
```

Dijkstra's Algorithm: Analysis

Dijkstra(G, s):

$S = \varnothing, Q = \varnothing$

Create array $d[1 : |V|], \pi[1 : |V|]$

for each vertex $v \in V(G)$

  $d[v] = \infty, \pi[v] = \infty$, **Insert**$(Q, v, \pi[v])$

$\pi[s] = 0$, **Decrease-Key**$(Q, s, 0)$

while $Q \neq \varnothing$

  $u =$ **Extract-Min**(Q)

  $S = S \cup \{u\}, d[u] = \pi[u]$

  for each vertex $v \in Adj[u]$

    if $\pi[v] > d[u] + w(u, v)$

      $\pi[v] = d[u] + w(u, v)$

      Decrease-Key$(Q, v, \pi[v])$

Suppose G has n vertices and m edges.

Dijkstra's Algorithm: Analysis

```
Dijkstra(G, s):
    S = ∅, Q = ∅
    Create array d[1 : |V|], π[1 : |V|]
    for each vertex v ∈ V(G)
        d[v] = ∞, π[v] = ∞, Insert(Q, v, π[v])
    π[s] = 0, Decrease-Key(Q, s,0)
    while Q ≠ ∅
        u = Extract-Min(Q)
        S = S ∪ {u}, d[u] = π[u]
        for each vertex v ∈ Adj[u]
            if  π[v] > d[u] + w(u, v)
                π[v] = d[u] + w(u, v)
                Decrease-Key(Q, v, π[v])
```

Suppose G has n vertices and m edges.

Cost of this loop is $O(n)$ (arrow pointing to "for each vertex $v \in V(G)$")

Dijkstra's Algorithm: Analysis

Dijkstra(G, s):

$S = \varnothing, Q = \varnothing$
Create array $d[1 : |V|], \pi[1 : |V|]$
for each vertex $v \in V(G)$
    $d[v] = \infty, \pi[v] = \infty,$ **Insert**$(Q, v, \pi[v])$
$\pi[s] = 0,$ **Decrease-Key**$(Q, s, 0)$
while $Q \neq \varnothing$
    $u =$ **Extract-Min**(Q)
    $S = S \cup \{u\}, d[u] = \pi[u]$
    for each vertex $v \in Adj[u]$
        if $\pi[v] > d[u] + w(u, v)$
            $\pi[v] = d[u] + w(u, v)$
            Decrease-Key$(Q, v, \pi[v])$

Suppose G has n vertices and m edges.

Cost of this loop is $O(n)$

Cost of this loop is $O(n \log n + m \log n)$

Dijkstra's Algorithm: Analysis

Dijkstra(G, s):

$S = \varnothing, Q = \varnothing$

Create array $d[1 : |V|], \pi[1 : |V|]$

for each vertex $v \in V(G)$

  $d[v] = \infty, \pi[v] = \infty,$ **Insert**$(Q, v, \pi[v])$

$\pi[s] = 0,$ **Decrease-Key**$(Q, s, 0)$

while $Q \neq \varnothing$

  $u =$ **Extract-Min**(Q)

  $S = S \cup \{u\}, d[u] = \pi[u]$

  for each vertex $v \in Adj[u]$

    if $\pi[v] > d[u] + w(u, v)$

      $\pi[v] = d[u] + w(u, v)$

      Decrease-Key$(Q, v, \pi[v])$

Suppose G has n vertices and m edges.

Cost of this loop is $O(n)$ (the **for** loop over $V(G)$)

Cost of this loop is $O(n \log n + m \log n)$ (the **while** loop)

(Every vertex is dequeued at most once, and when dequeued its adjacency list is traversed.)

Dijkstra's Algorithm: Analysis

Dijkstra(G, s):

```
S = ∅, Q = ∅
Create array d[1 : |V|], π[1 : |V|]
for each vertex v ∈ V(G)
    d[v] = ∞, π[v] = ∞, Insert(Q, v, π[v])
π[s] = 0, Decrease-Key(Q, s,0)
while Q ≠ ∅
    u = Extract-Min(Q)
    S = S ∪ {u}, d[u] = π[u]
    for each vertex v ∈ Adj[u]
        if π[v] > d[u] + w(u, v)
            π[v] = d[u] + w(u, v)
            Decrease-Key(Q, v, π[v])
```

Suppose G has n vertices and m edges.

Cost of this loop is $O(n)$ (the loop over each vertex $v \in V(G)$)

Cost of this loop is $O(n \log n + m \log n)$ (the loop while $Q \neq \emptyset$)

(Every vertex is dequeued at most once, and when dequeued its adjacency list is traversed.)

Time complexity $= O(n \log n + m \log n)$.

Dijkstra's Algorithm: Analysis

Dijkstra(G, s):

```
S = ∅, Q = ∅
Create array d[1 : |V|], π[1 : |V|]
for each vertex v ∈ V(G)
    d[v] = ∞, π[v] = ∞, Insert(Q, v, π[v])
π[s] = 0, Decrease-Key(Q, s, 0)
while Q ≠ ∅
    u = Extract-Min(Q)
    S = S ∪ {u}, d[u] = π[u]
    for each vertex v ∈ Adj[u]
        if π[v] > d[u] + w(u, v)
            π[v] = d[u] + w(u, v)
            Decrease-Key(Q, v, π[v])
```

Suppose G has n vertices and m edges.

Cost of this loop is $O(n)$ (the "for each vertex $v \in V(G)$" loop)

Cost of this loop is $O(n \log n + m \log n)$ (the "while $Q \neq \varnothing$" loop)

(Every vertex is dequeued at most once, and when dequeued its adjacency list is traversed.)

Time complexity $= O(m \log n)$, when $m > n$.

Dijkstra's Algorithm: DIY

Dijkstra's Algorithm: DIY

Modify the Dijkstra's algorithm on the previous slide so that:

Dijkstra's Algorithm: DIY

Modify the Dijkstra's algorithm on the previous slide so that:

- It uses only **one extra array** d to calculate distances.

Dijkstra's Algorithm: DIY

Modify the Dijkstra's algorithm on the previous slide so that:

- It uses only one extra array d to calculate distances.
- You can produce shortest paths as well not just distance.

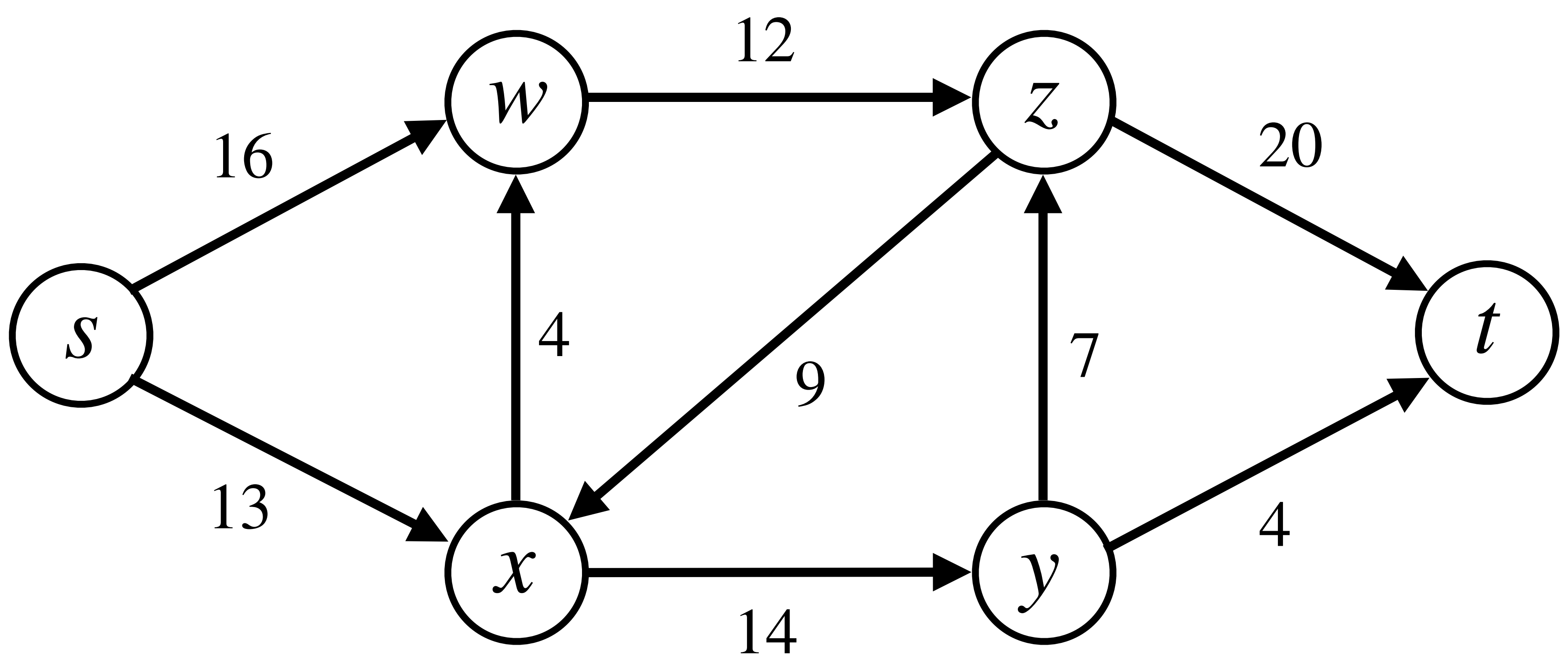
Dijkstra's Algorithm: History

"What is the shortest way to travel from Rotterdam to Groningen, in general: from given city to given city. It is the algorithm for the shortest path, which I designed in about twenty minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path. As I said, it was a twenty-minute invention. In fact, it was published in '59, three years later. The publication is still readable, it is, in fact, quite nice. One of the reasons that it is so nice was that I designed it without pencil and paper. I learned later that one of the advantages of designing without pencil and paper is that you are almost forced to avoid all avoidable complexities. Eventually, that algorithm became to my great amazement, one of the cornerstones of my fame."

— Edsger Dijkstra, in an interview with Philip L. Frana, Communications of the ACM, 2001

Flow Networks

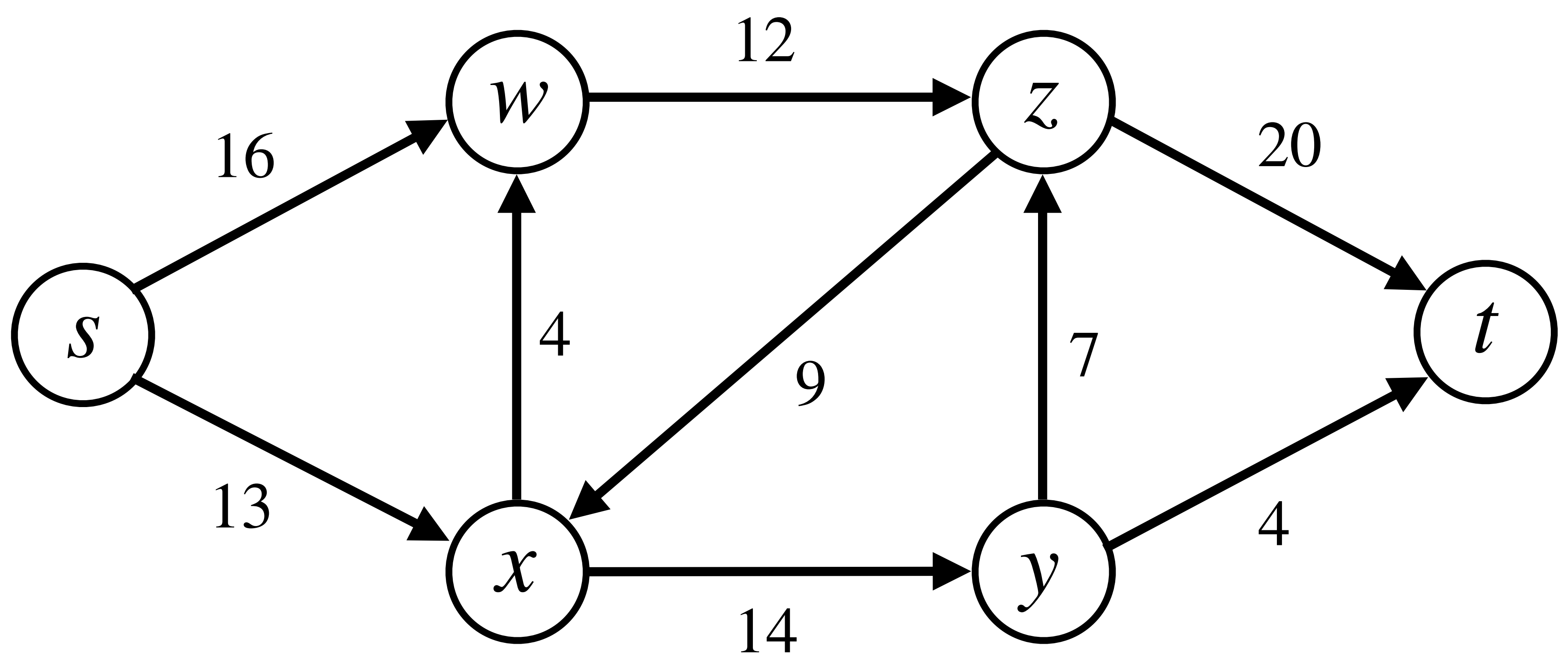
Flow Networks



(a)

Flow Networks

Figure (a) is flow network of a shipping company, where:

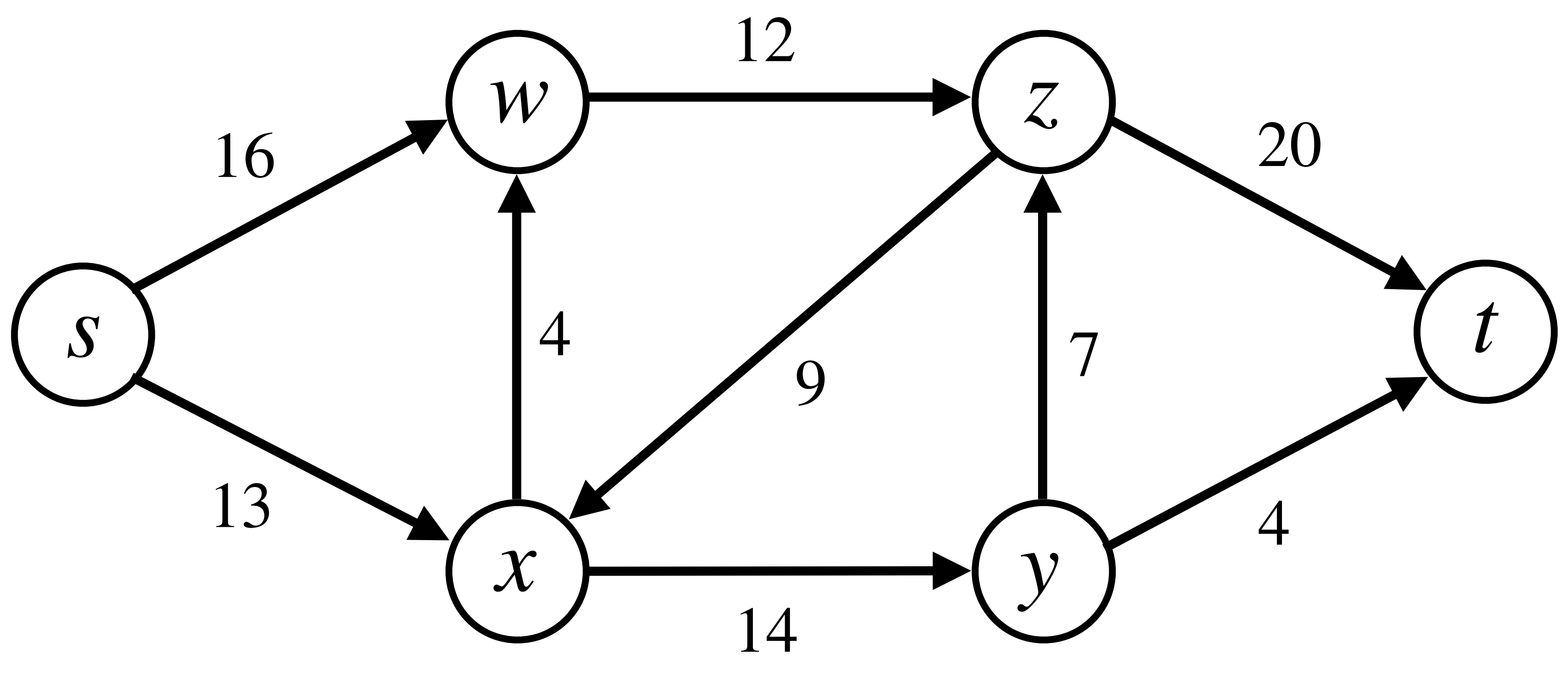


(a)

Flow Networks

Figure (a) is flow network of a shipping company, where:

- Vertices represent **cities**. s & t are the **source** & **sink** cities.

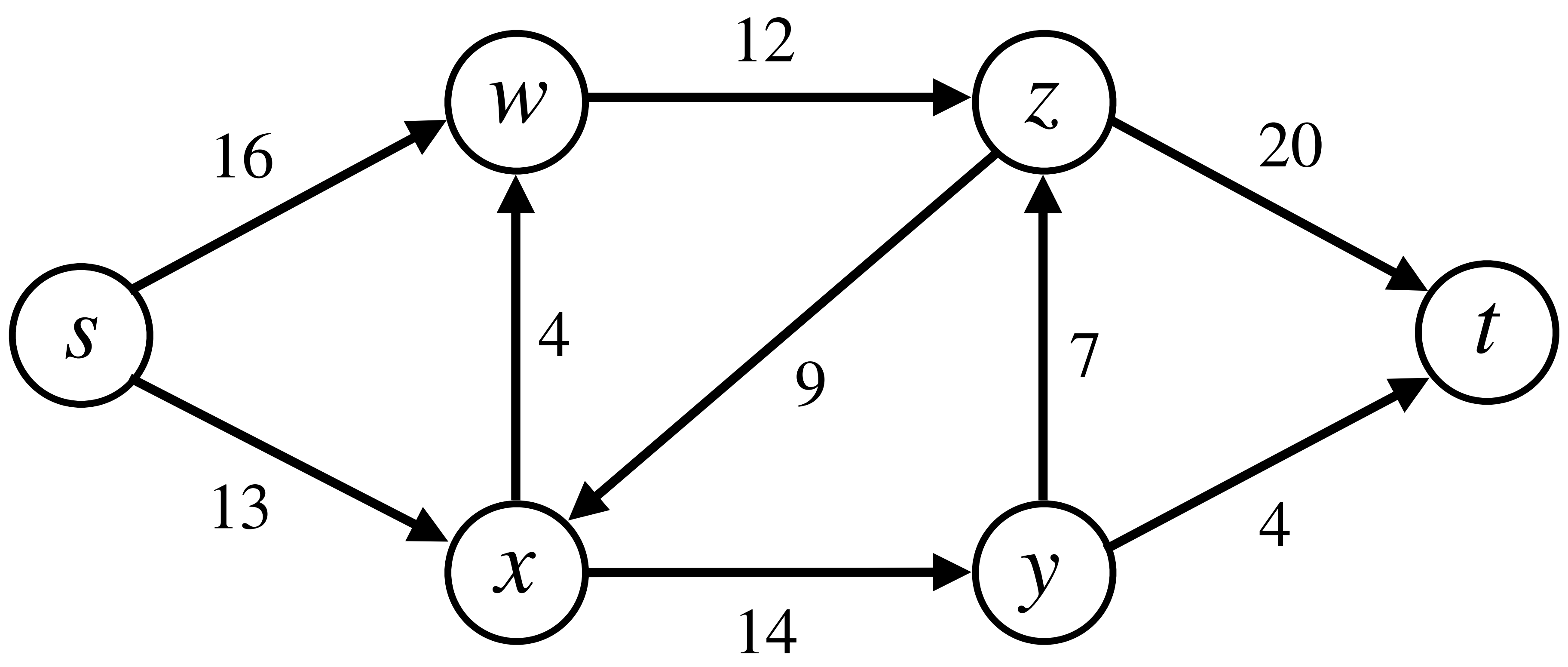


(a)

Flow Networks

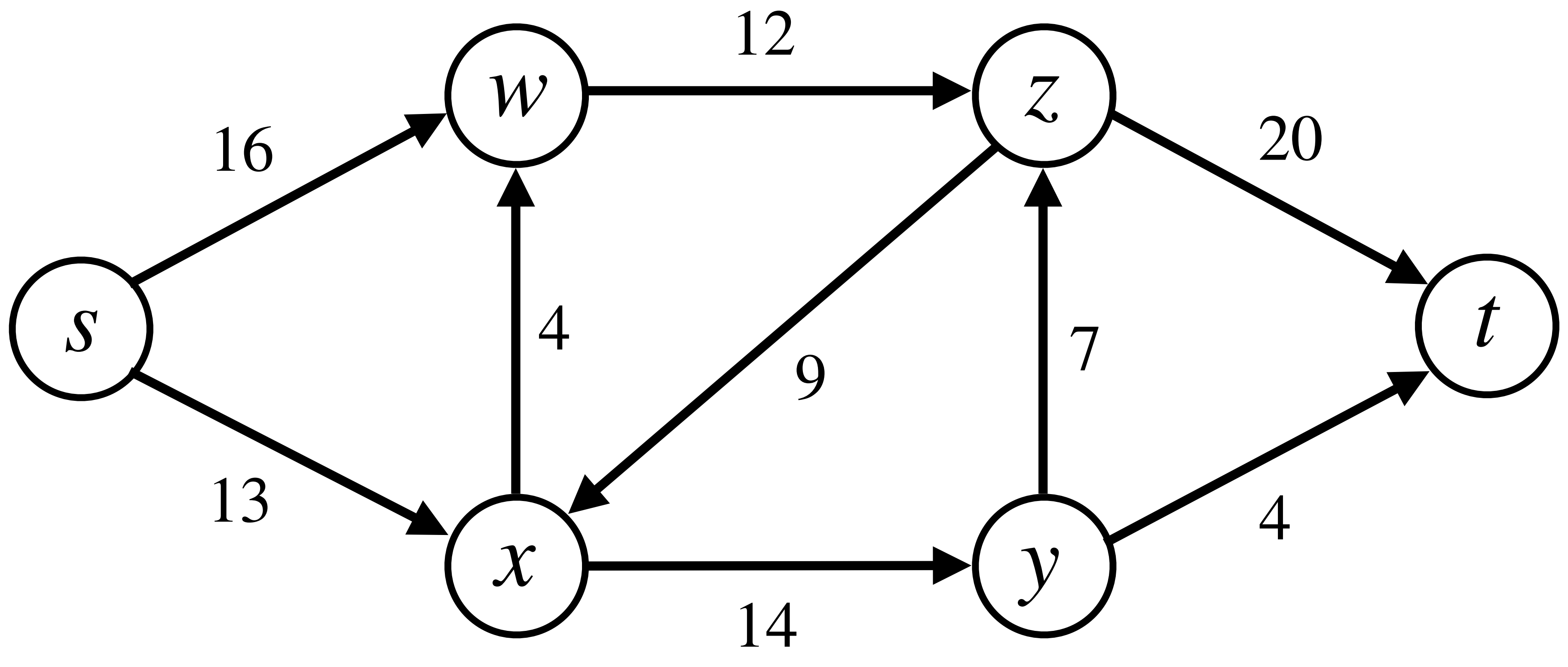
Figure (a) is flow network of a shipping company, where:

- Vertices represent **cities**. s & t are the **source** & **sink** cities.
- The number on any (u, v) edge is the **maximum number of packets** that can go from u to v per day.



(a)

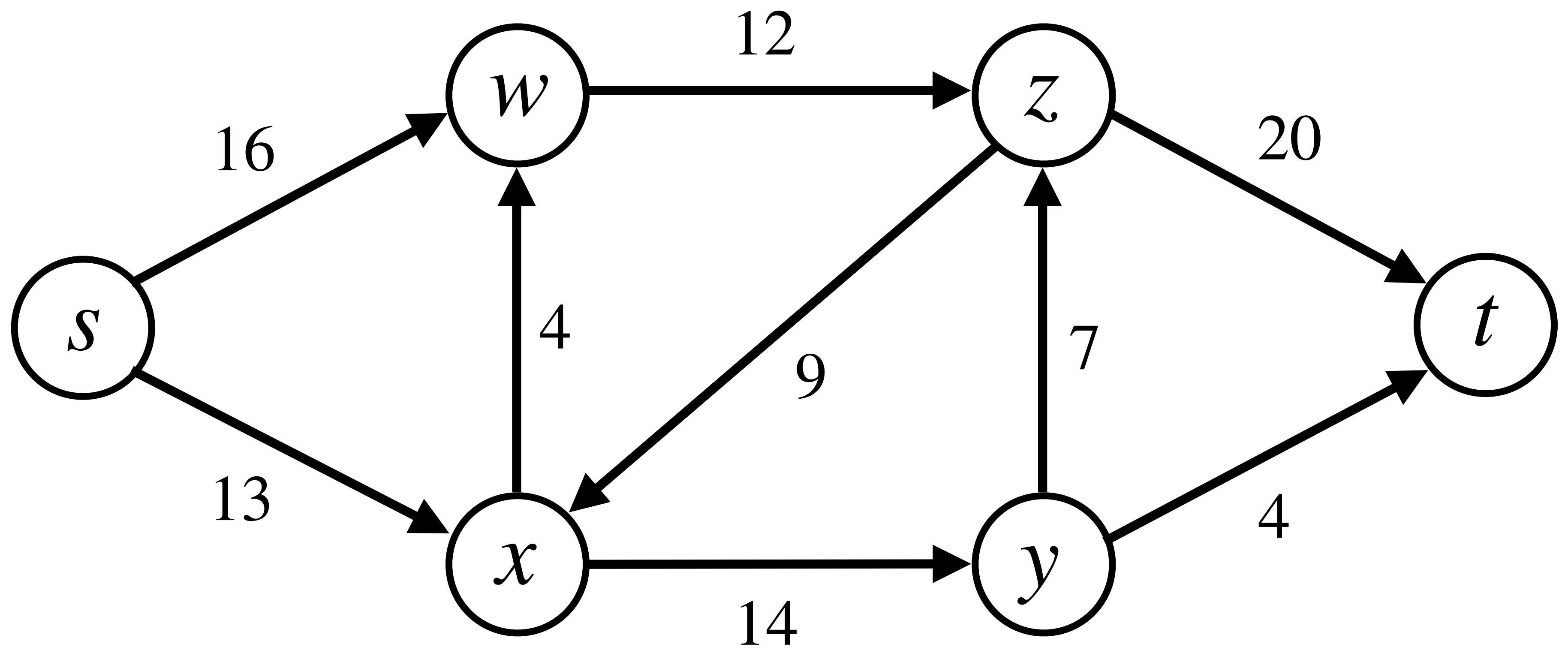
Flow Networks



(a)

Flow Networks

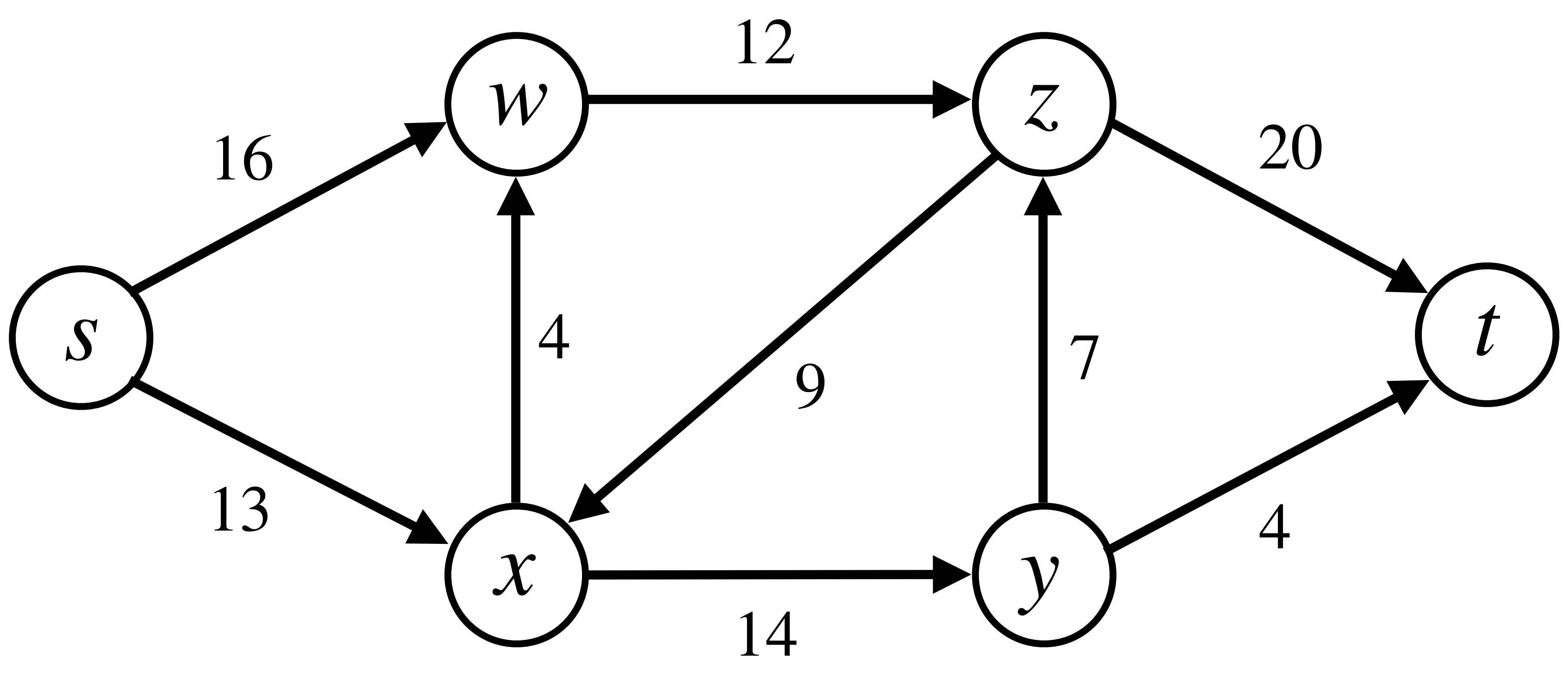
Goal: Find the **maximum number of packets** that can be shipped from s if the packets received and



(a)

Flow Networks

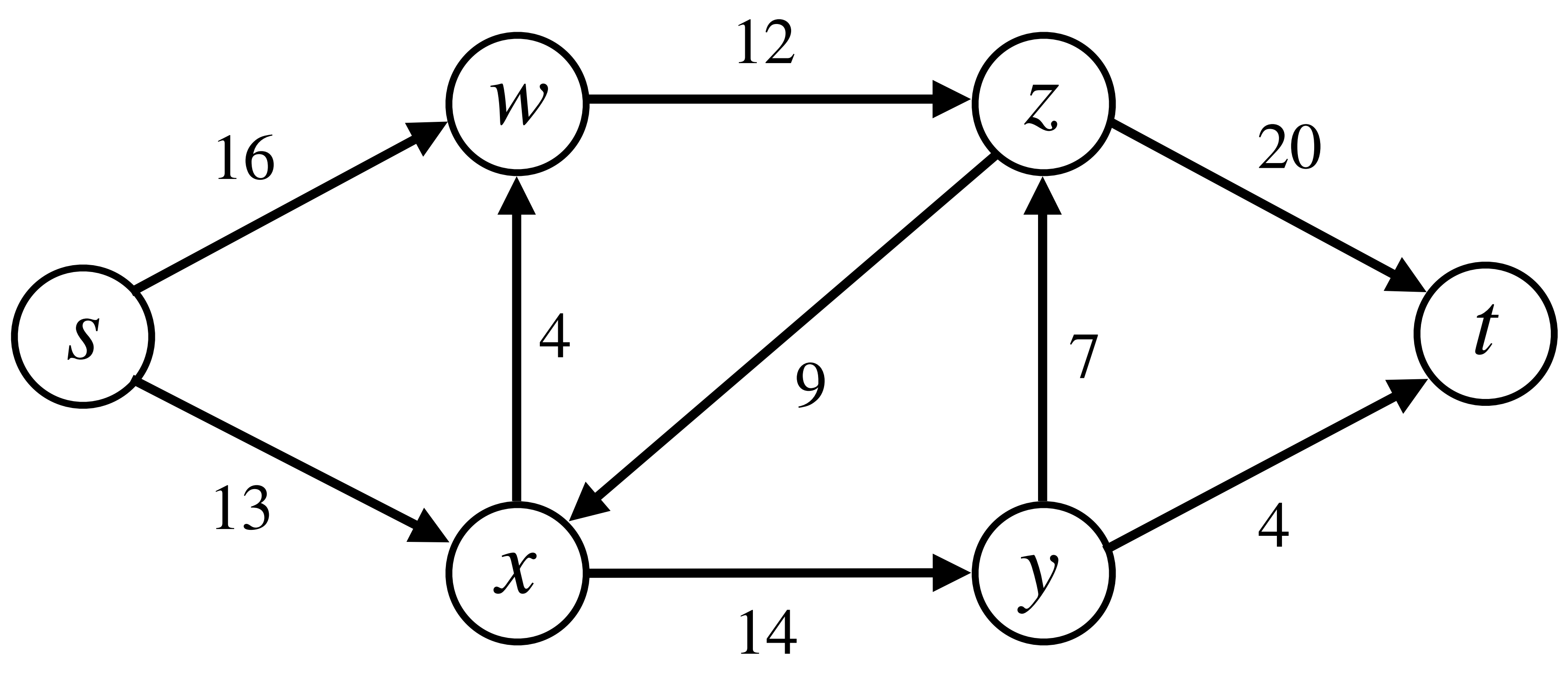
Goal: Find the **maximum number of packets** that can be shipped from s if the packets received and sent by intermediate cities are equal in numbers.



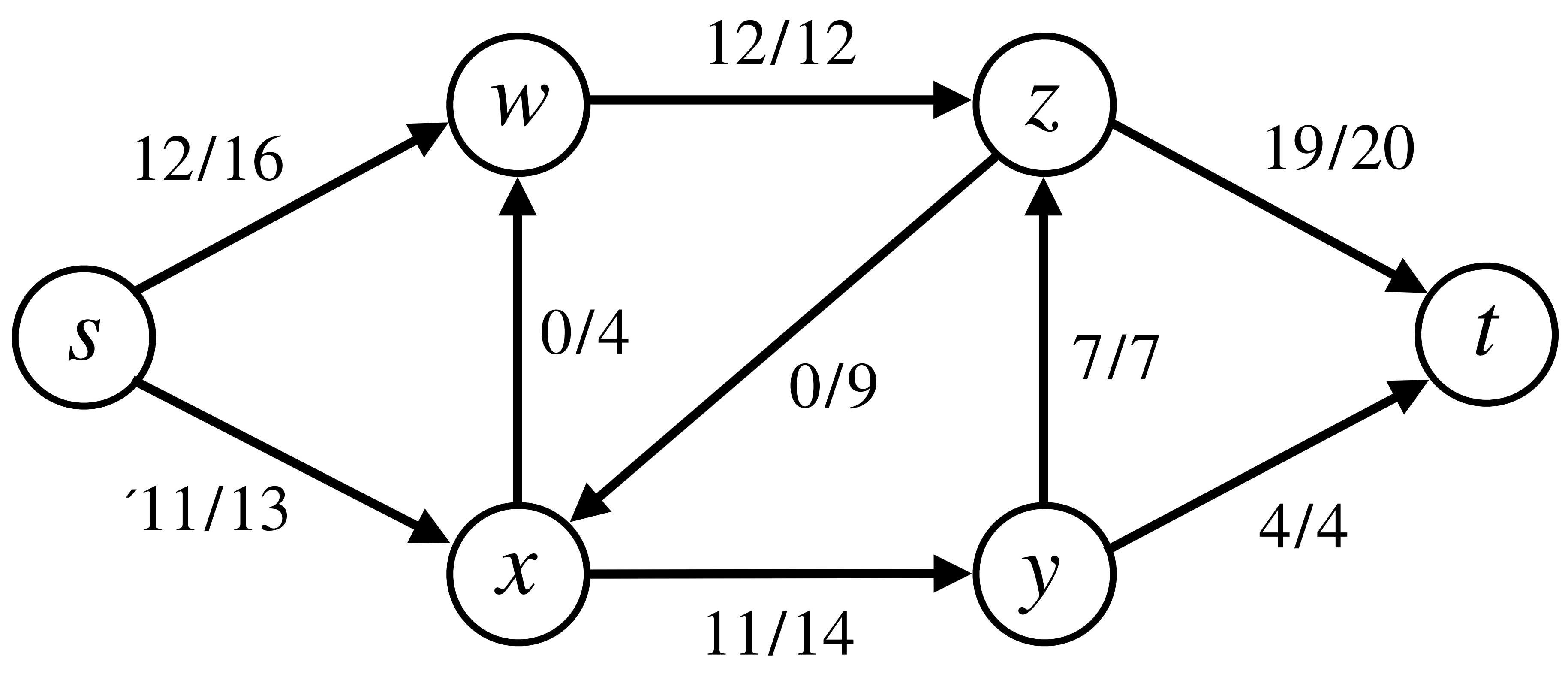
(a)

Flow Networks

Goal: Find the **maximum number of packets** that can be shipped from s if the packets received and sent by intermediate cities are equal in numbers.



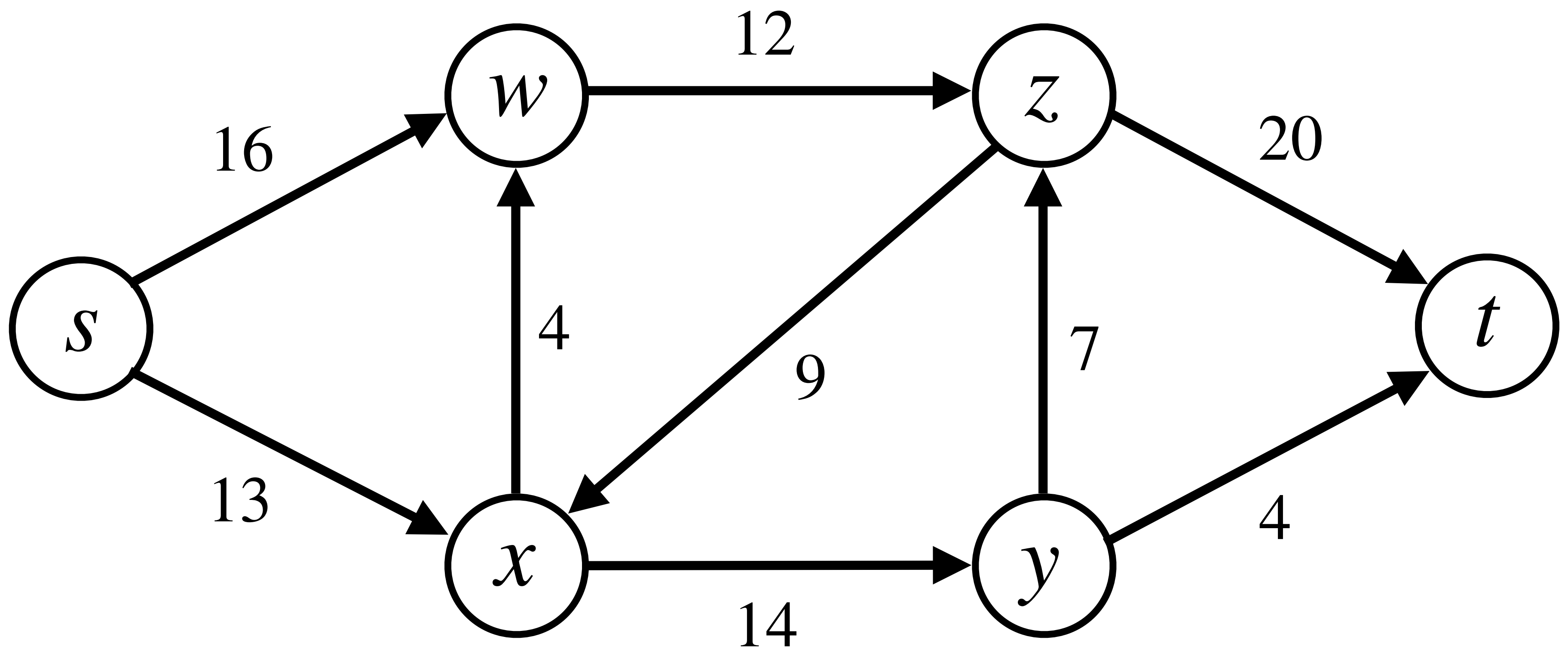
(a)



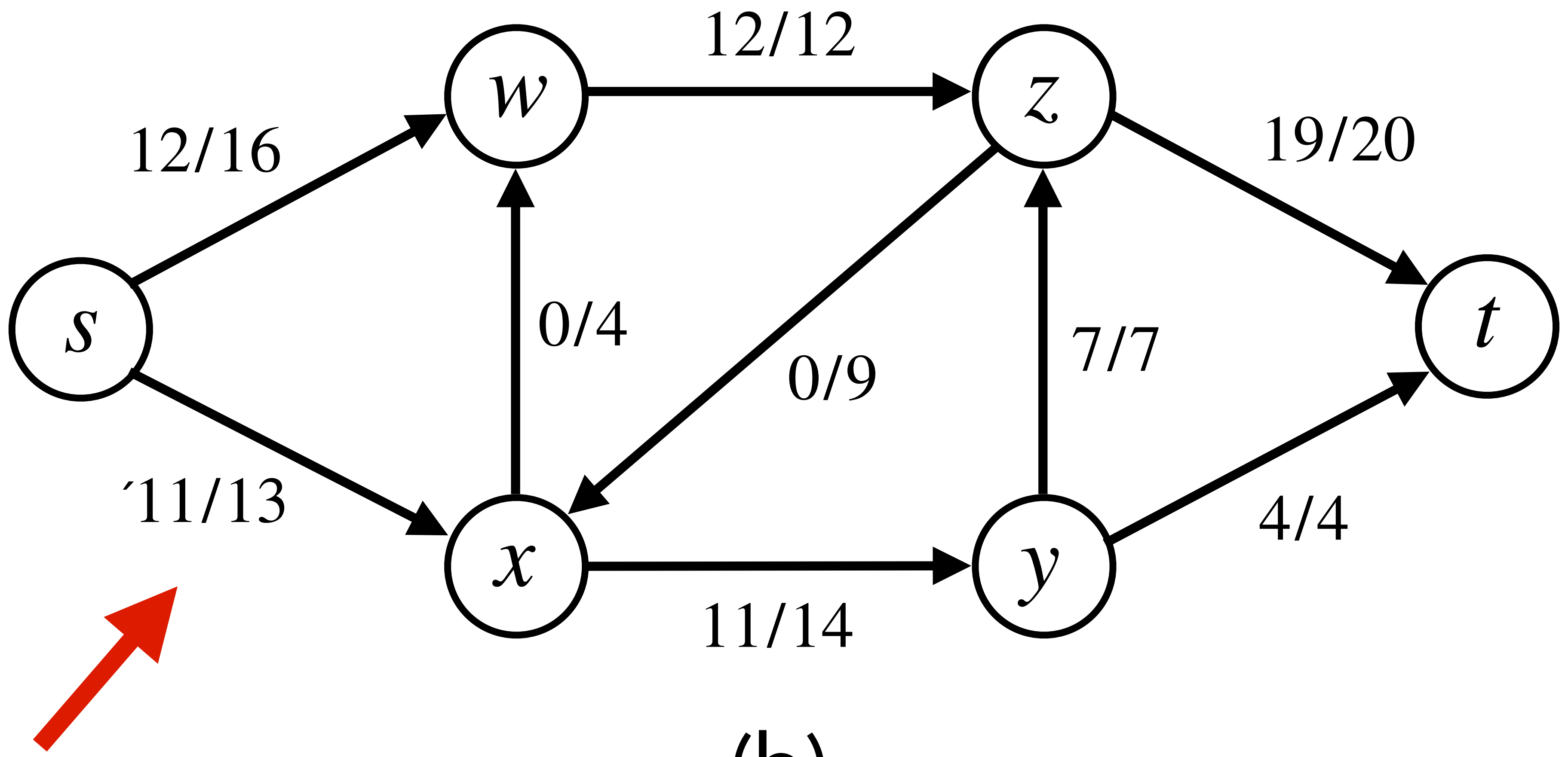
(b)

Flow Networks

Goal: Find the **maximum number of packets** that can be shipped from s if the packets received and sent by intermediate cities are equal in numbers.



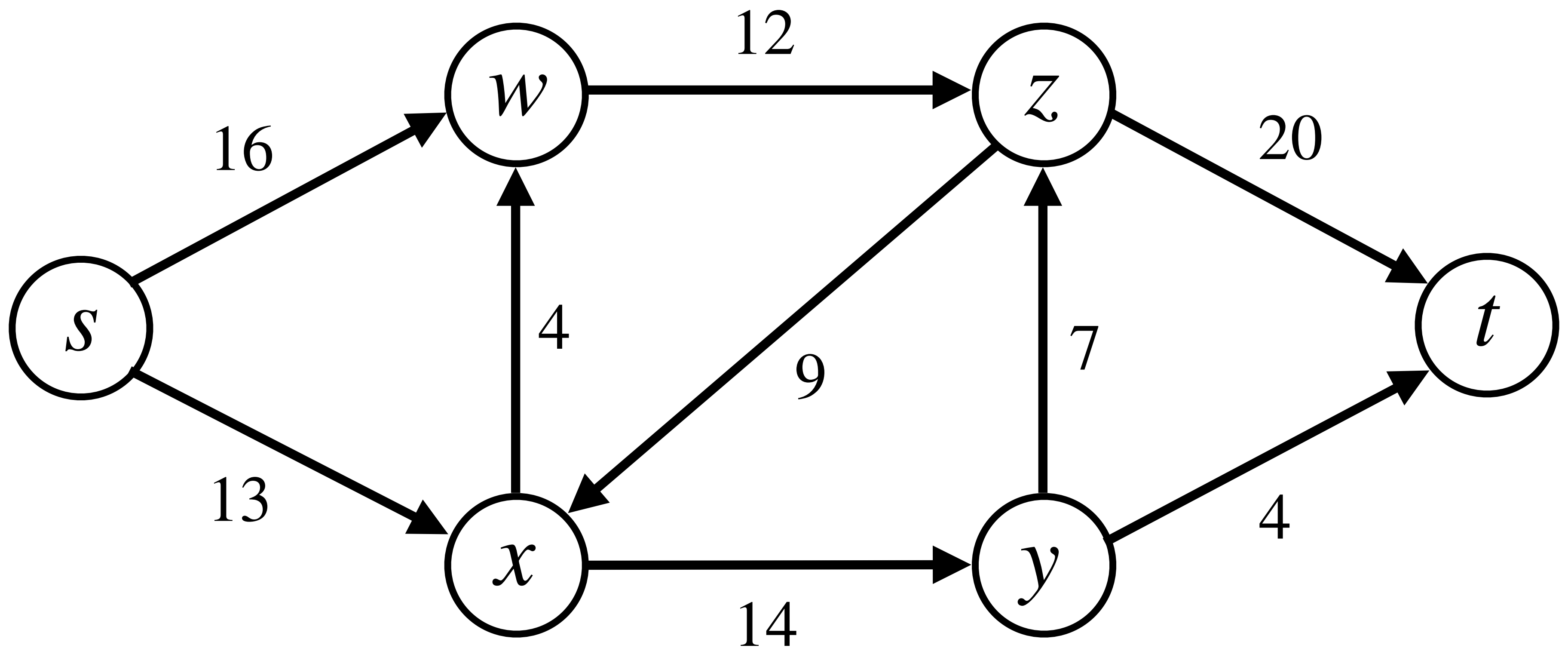
(a)



(b)

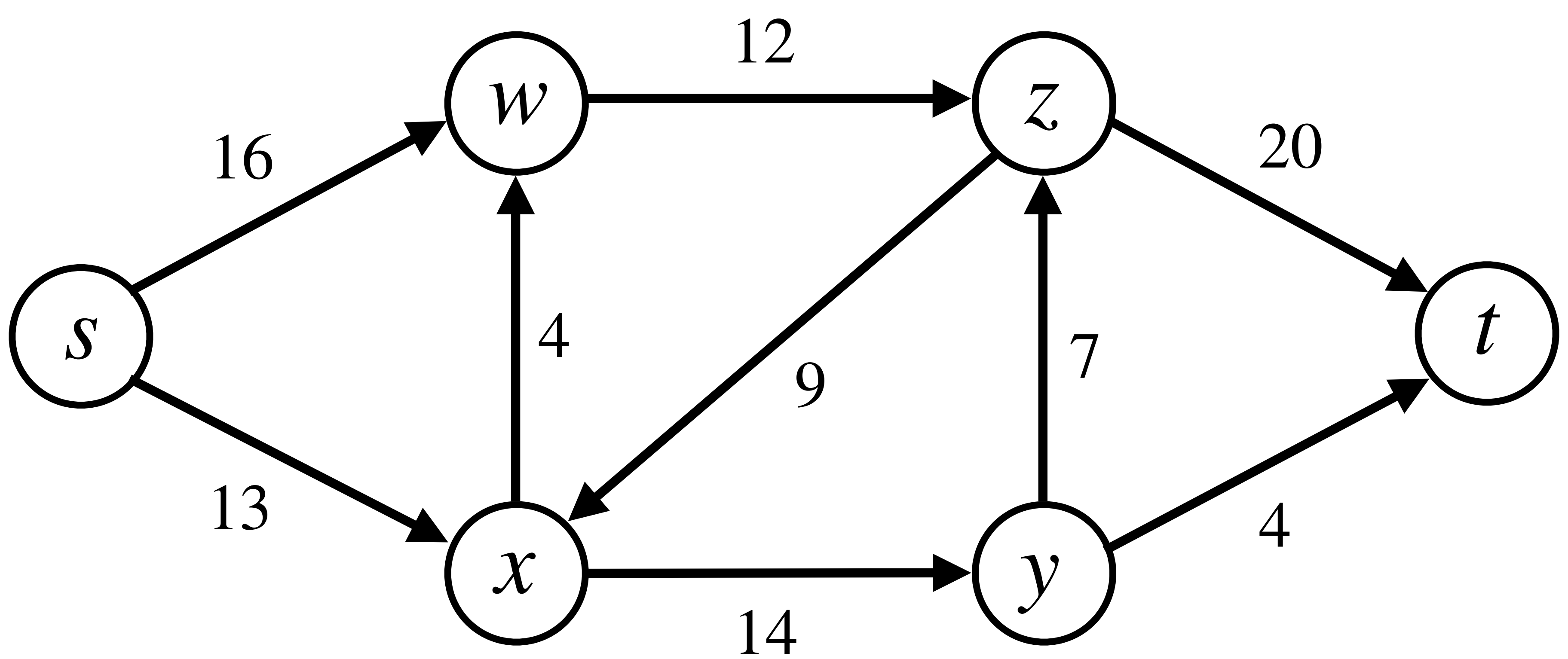
max packets $= 23$

Flow Networks



Flow Networks

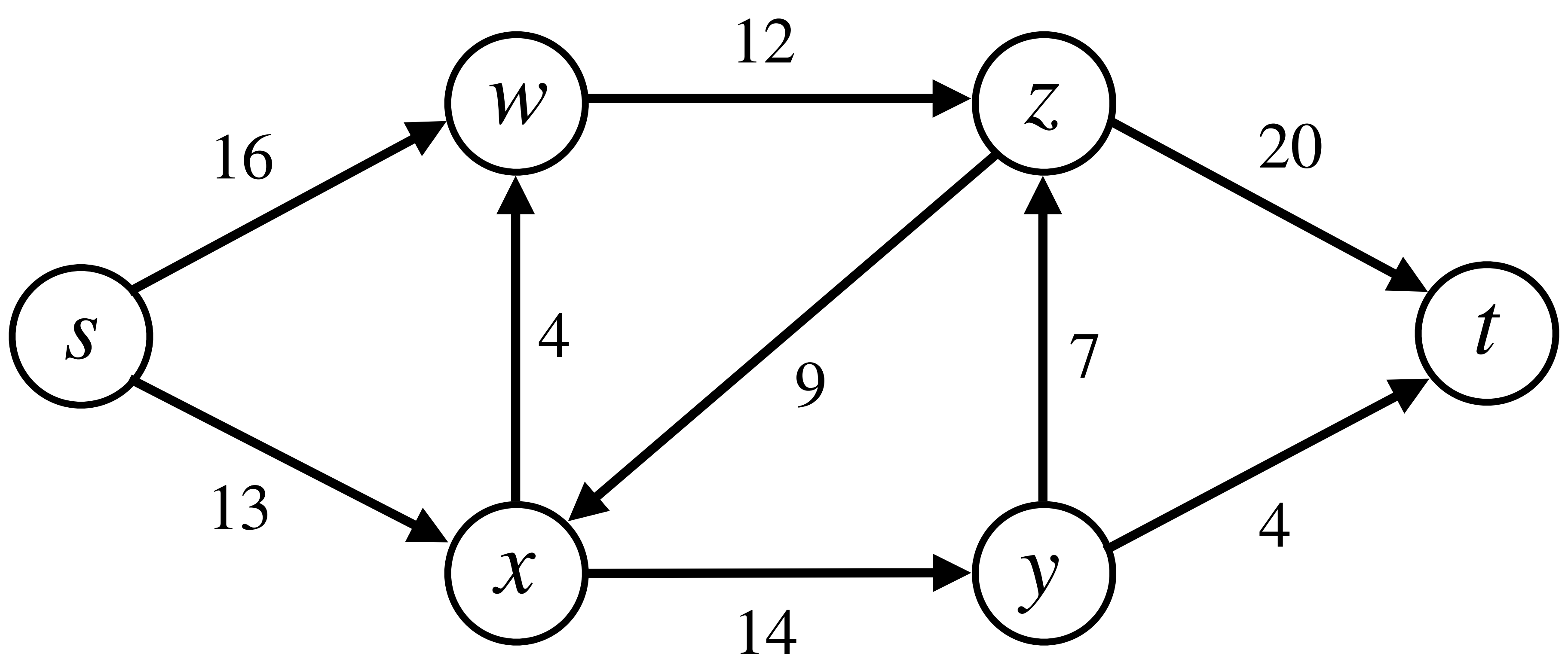
Defn: A flow network $G = (V, E)$ is a directed graph in which:



Flow Networks

Defn: A flow network $G = (V, E)$ is a directed graph in which:

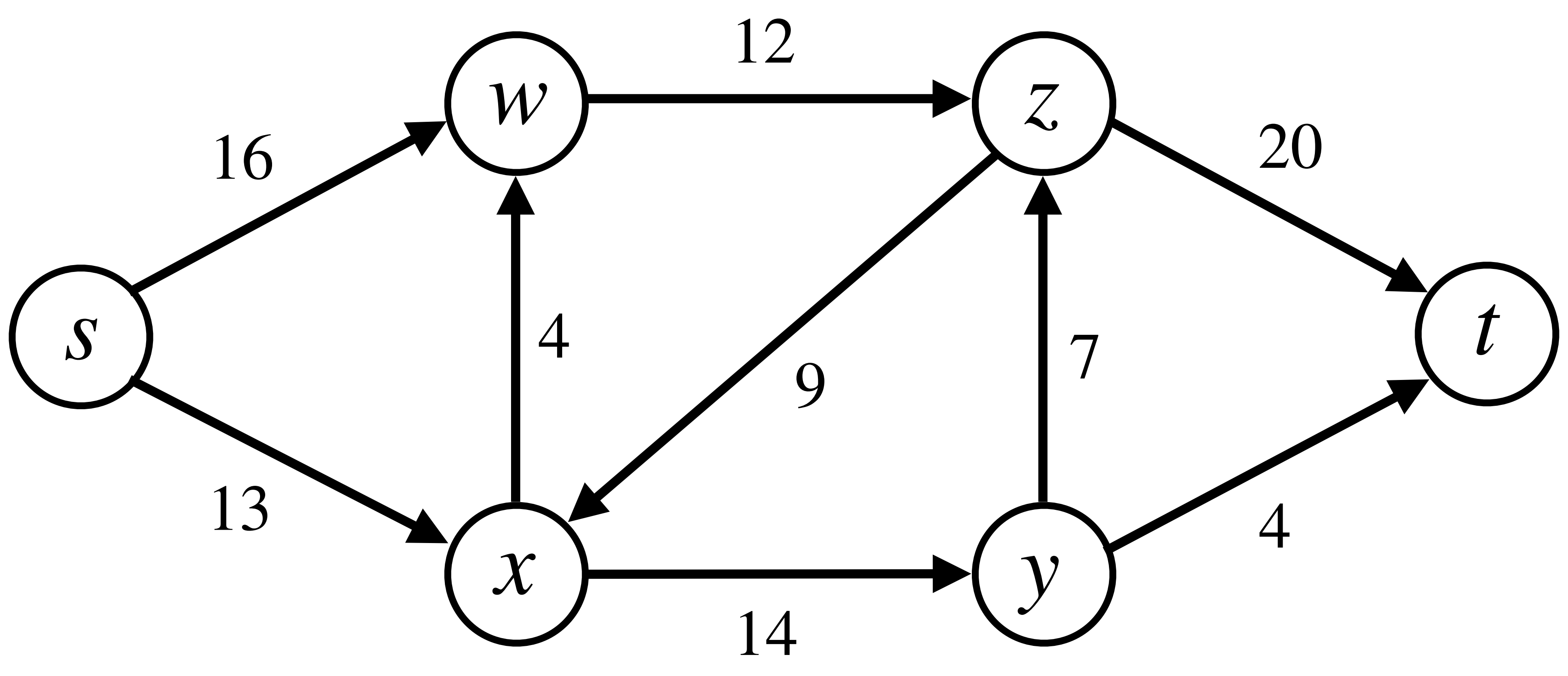
- Each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$.



Flow Networks

Defn: A flow network $G = (V, E)$ is a directed graph in which:

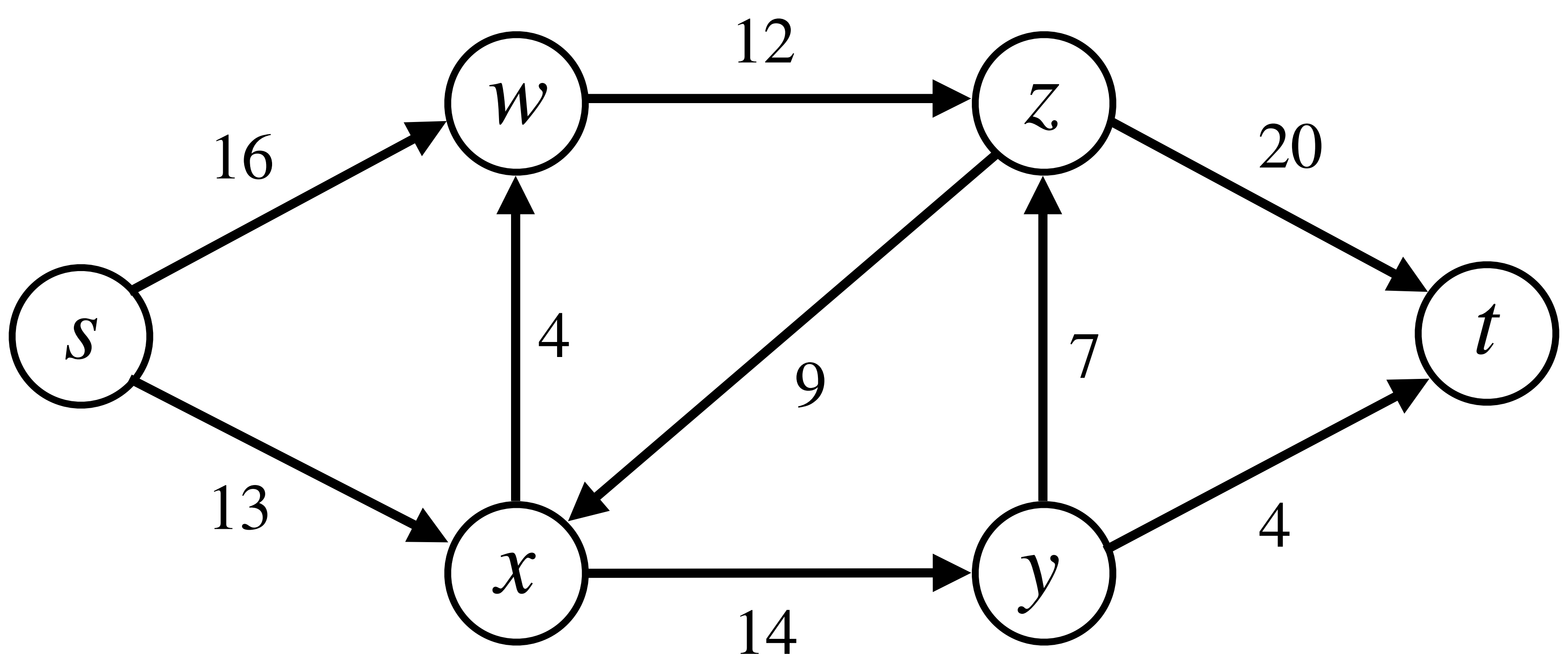
- Each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$.
- If $(u, v) \in E$, then $(v, u) \notin E$.



Flow Networks

Defn: A flow network $G = (V, E)$ is a directed graph in which:

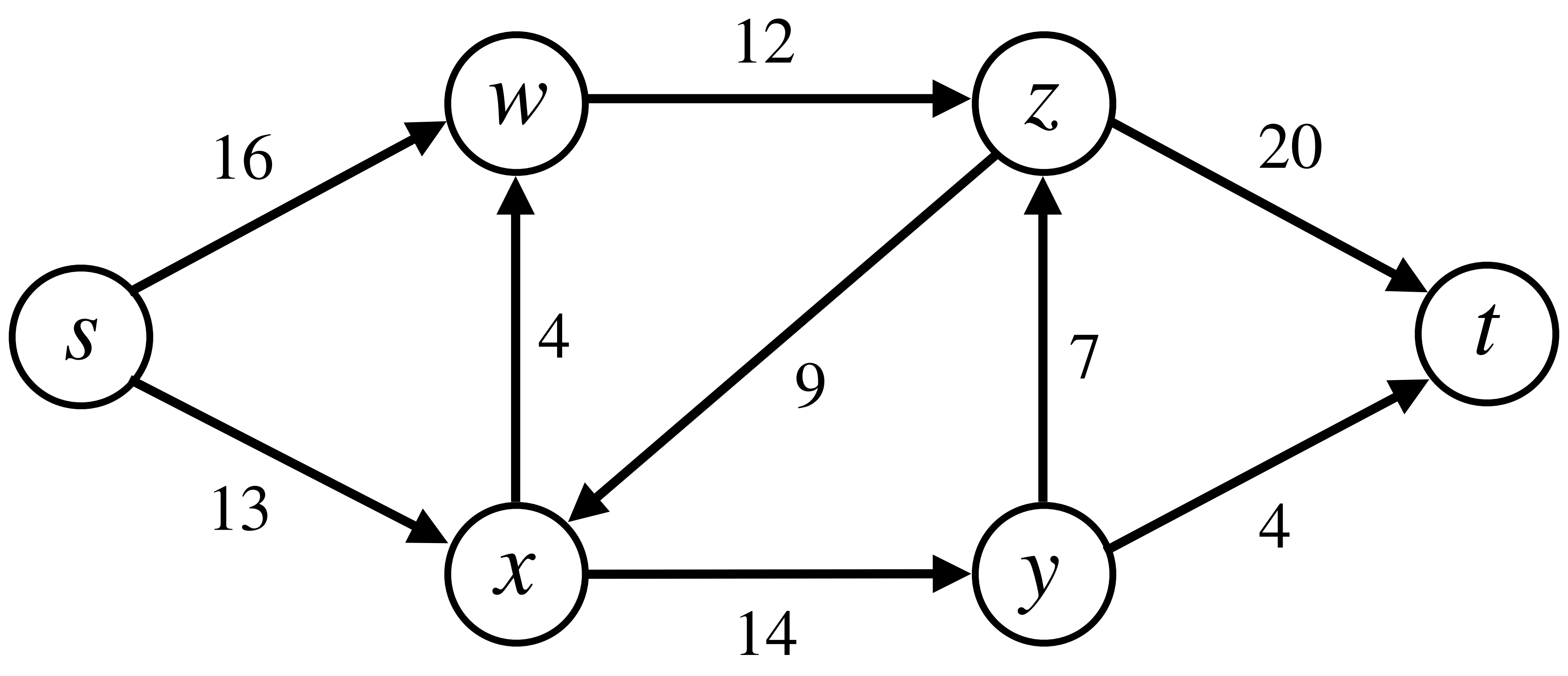
- Each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$.
- If $(u, v) \in E$, then $(v, u) \notin E$. (Reason will become clear soon.)



Flow Networks

Defn: A flow network $G = (V, E)$ is a directed graph in which:

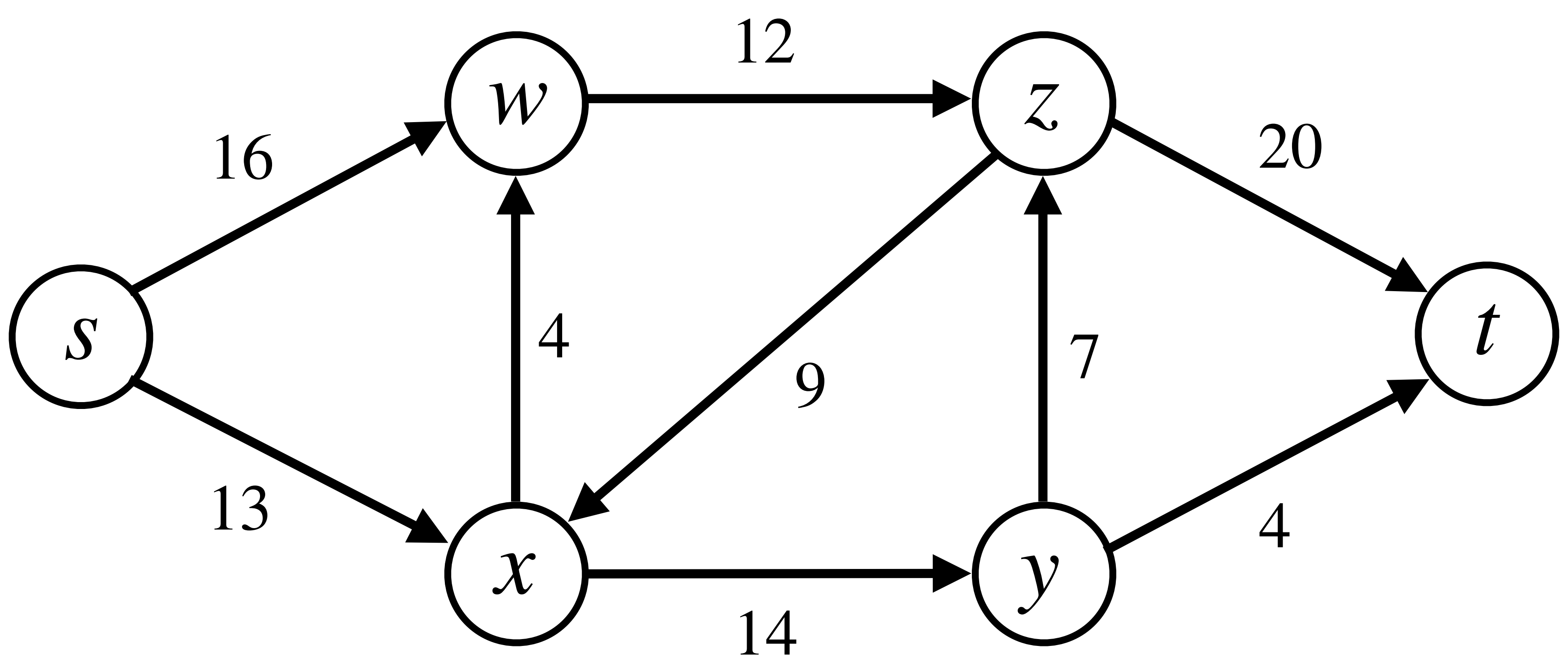
- Each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$.
- If $(u, v) \in E$, then $(v, u) \notin E$. (Reason will become clear soon.)
- If $(u, v) \notin E$, we define $c(u, v) = 0$. No self-loops are present.



Flow Networks

Defn: A flow network $G = (V, E)$ is a directed graph in which:

- Each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$.
- If $(u, v) \in E$, then $(v, u) \notin E$. (Reason will become clear soon.)
- If $(u, v) \notin E$, we define $c(u, v) = 0$. No self-loops are present.
- Two distinguished vertices: source s (no incoming edges) and sink t (no outgoing edges).



Flow Networks

Defn: A flow network $G = (V, E)$ is a directed graph in which:

- Each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$.
- If $(u, v) \in E$, then $(v, u) \notin E$. (Reason will become clear soon.)
- If $(u, v) \notin E$, we define $c(u, v) = 0$. No self-loops are present.
- Two distinguished vertices: source s (no incoming edges) and sink t (no outgoing edges).
- For every $v \in V$, some $s \rightsquigarrow v \rightsquigarrow t$ path exists. Hence, $|E| \geq |V| - 1$.