Prim's Algorithm

For every $v \in V \setminus S$ maintain

- $d[v] = \min_{u \in S:(u,v) \in E} w(u, v)$:
  the weight of the lightest edge between v and S
- $\pi[v] = \arg\min_{u \in S:(u,v) \in E} w(u, v)$:
  $(\pi[v], v)$ is the lightest edge between v and S

In every iteration

- Pick $u \in V \setminus S$ with the smallest $d[u]$ value
- Add $(\pi[u], u)$ to F
- Add u to S, update d and π values.

Prim's Algorithm

For every $v \in V \setminus S$ maintain

- $d[v] = \min_{u \in S:(u,v)\in E} w(u,v)$:
  the weight of the lightest edge between v and S
- $\pi[v] = \arg\min_{u \in S:(u,v)\in E} w(u,v)$:
  $(\pi[v], v)$ is the lightest edge between v and S

In every iteration

- Pick $u \in V \setminus S$ with the smallest $d[u]$ value — extract_min
- Add $(\pi[u], u)$ to F
- Add u to S, update d and π values. — decrease_key

Use a priority queue to support the operations

Def. A priority queue is an abstract data structure that maintains a set U of elements, each with an associated key value, and supports the following operations:

- insert(v, key_value): insert an element v, whose associated key value is key_value.
- decrease_key(v, new_key_value): decrease the key value of an element v in queue to new_key_value
- extract_min$()$: return and remove the element in queue with the smallest key value
- $\cdots$

Prim's Algorithm

MST-Prim(G, w)

```
s ← arbitrary vertex in G
S ← ∅, d(s) ← 0 and d[v] ← ∞ for every v ∈ V \ {s}

while S ≠ V do
    u ← vertex in V \ S with the minimum d[u]
    S ← S ∪ {u}
    for each v ∈ V \ S such that (u, v) ∈ E do
        if w(u, v) < d[v] then
            d[v] ← w(u, v)
            π[v] ← u
return {(u, π[u]) | u ∈ V \ {s}}
```

Prim's Algorithm Using Priority Queue

MST-Prim(G, w)

$s \leftarrow$ arbitrary vertex in G
$S \leftarrow \emptyset, d(s) \leftarrow 0$ and $d[v] \leftarrow \infty$ for every $v \in V \setminus \{s\}$
$Q \leftarrow$ empty queue, for each $v \in V$: Q.insert$(v, d[v])$
while $S \neq V$ **do**
    $u \leftarrow Q$.extract_min()
    $S \leftarrow S \cup \{u\}$
    for each $v \in V \setminus S$ such that $(u, v) \in E$ **do**
        if $w(u, v) < d[v]$ **then**
            $d[v] \leftarrow w(u, v)$, Q.decrease_key$(v, d[v])$
            $\pi[v] \leftarrow u$
return $\big\{(u, \pi[u]) | u \in V \setminus \{s\}\big\}$

Running Time of Prim's Algorithm Using Priority Queue

$O(n)\times$ (time for extract_min) $+\ O(m)\times$ (time for decrease_key)

Running Time of Prim's Algorithm Using Priority Queue

$O(n)\times$ (time for extract_min) $+ O(m)\times$ (time for decrease_key)

concrete DS	extract_min	decrease_key	overall time
heap	$O(\log n)$	$O(\log n)$	$O(m \log n)$
Fibonacci heap	$O(\log n)$	$O(1)$	$O(n \log n + m)$

Running Time of Prim's Algorithm Using Priority Queue

$O(n)\times$ (time for extract_min) $+$ $O(m)\times$ (time for decrease_key)

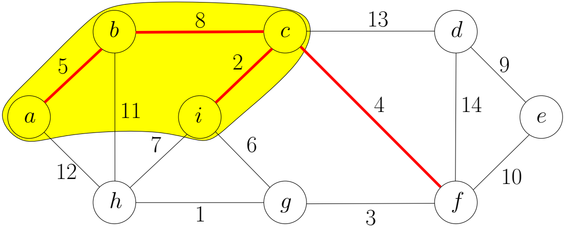
concrete DS	extract_min	decrease_key	overall time
heap	$O(\log n)$	$O(\log n)$	$O(m \log n)$
Fibonacci heap	$O(\log n)$	$O(1)$	$O(n \log n + m)$

Assumption Assume all edge weights are different.

Lemma (u, v) is in MST, if and only if there exists a cut $(U, V \setminus U)$, such that (u, v) is the lightest edge between U and $V \setminus U$.

Assumption Assume all edge weights are different.

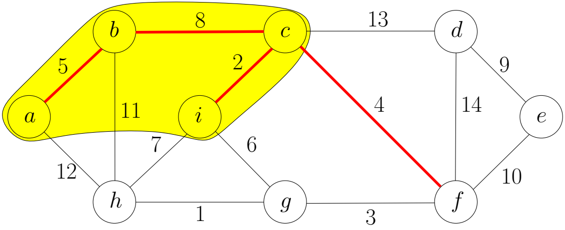
Lemma (u, v) is in MST, if and only if there exists a cut $(U, V \setminus U)$, such that (u, v) is the lightest edge between U and $V \setminus U$.



- (c, f) is in MST because of cut $\big(\{a, b, c, i\}, V \setminus \{a, b, c, i\}\big)$

Assumption Assume all edge weights are different.

Lemma (u, v) is in MST, if and only if there exists a cut $(U, V \setminus U)$, such that (u, v) is the lightest edge between U and $V \setminus U$.



- (c, f) is in MST because of cut $\big(\{a, b, c, i\}, V \setminus \{a, b, c, i\}\big)$
- (i, g) is not in MST because no such cut exists

"Evidence" for $e \in$ MST or $e \notin$ MST

Assumption Assume all edge weights are different.

- $e \in$ MST $\leftrightarrow$ there is a cut in which e is the lightest edge
- $e \notin$ MST $\leftrightarrow$ there is a cycle in which e is the heaviest edge

"Evidence" for $e \in$ MST or $e \notin$ MST

Assumption Assume all edge weights are different.

- $e \in$ MST $\leftrightarrow$ there is a cut in which e is the lightest edge
- $e \notin$ MST $\leftrightarrow$ there is a cycle in which e is the heaviest edge

Exactly one of the following is true:

- There is a cut in which e is the lightest edge
- There is a cycle in which e is the heaviest edge

“Evidence” for $e \in$ MST or $e \notin$ MST

Assumption Assume all edge weights are different.

- $e \in$ MST $\leftrightarrow$ there is a cut in which e is the lightest edge
- $e \notin$ MST $\leftrightarrow$ there is a cycle in which e is the heaviest edge

Exactly one of the following is true:

- There is a cut in which e is the lightest edge
- There is a cycle in which e is the heaviest edge

Thus, the minimum spanning tree is unique with assumption.

Outline

algorithm	graph	weights	SS?	running time
Simple DP	DAG	$\mathbb{R}$	SS	$O(n+m)$
Dijkstra	U/D	$\mathbb{R}_{\geq 0}$	SS	$O(n \log n + m)$
Bellman-Ford	U/D	$\mathbb{R}$	SS	$O(nm)$
Floyd-Warshall	U/D	$\mathbb{R}$	AP	$O(n^3)$

- DAG = directed acyclic graph   U = undirected   D = directed
- SS = single source   AP = all pairs

s-t Shortest Paths

Input: (directed or undirected) graph $G = (V, E)$, $s, t \in V$

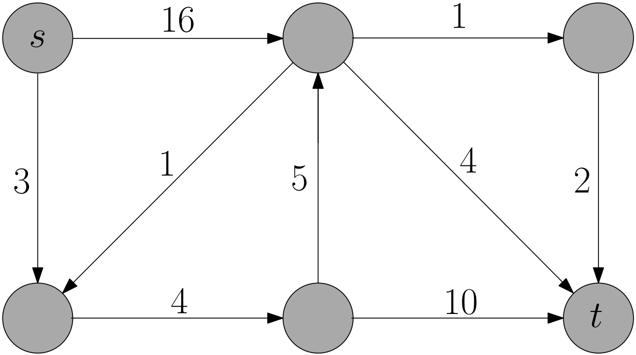
$w : E \to \mathbb{R}_{\geq 0}$

Output: shortest path from s to t

s-t Shortest Paths

Input: (directed or undirected) graph $G = (V, E)$, $s, t \in V$
$w : E \to \mathbb{R}_{\geq 0}$

Output: shortest path from s to t

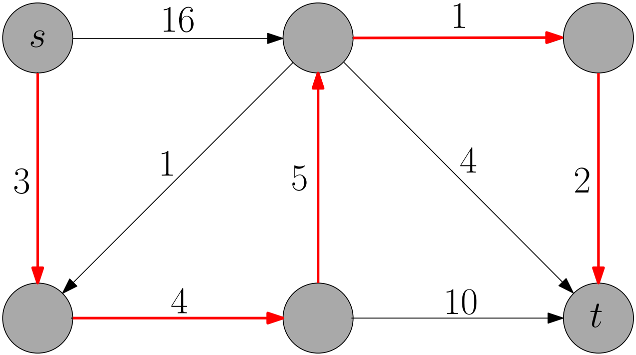


s-t Shortest Paths

Input: (directed or undirected) graph $G = (V, E)$, $s, t \in V$

$w : E \to \mathbb{R}_{\geq 0}$

Output: shortest path from s to t



Single Source Shortest Paths

Input: (directed or undirected) graph $G = (V, E)$, $s \in V$

$w : E \to \mathbb{R}_{\geq 0}$

Output: shortest paths from s to all other vertices $v \in V$

Single Source Shortest Paths

Input: (directed or undirected) graph $G = (V, E)$, $s \in V$
$w : E \to \mathbb{R}_{\geq 0}$

Output: shortest paths from s to all other vertices $v \in V$

Reason for Considering Single Source Shortest Paths Problem

- We do not know how to solve s-t shortest path problem more efficiently than solving single source shortest path problem

Single Source Shortest Paths

Input: (directed or undirected) graph $G = (V, E)$, $s \in V$

$w : E \to \mathbb{R}_{\geq 0}$

Output: shortest paths from s to all other vertices $v \in V$

Reason for Considering Single Source Shortest Paths Problem

- We do not know how to solve s-t shortest path problem more efficiently than solving single source shortest path problem

- Shortest paths in directed graphs is more general than in undirected graphs: we can replace every undirected edge with two anti-parallel edges of the same weight

Single Source Shortest Paths

Input: (directed or undirected) graph $G = (V, E)$, $s \in V$
$w : E \to \mathbb{R}_{\geq 0}$

Output: shortest paths from s to all other vertices $v \in V$

Reason for Considering Single Source Shortest Paths Problem

- We do not know how to solve s-t shortest path problem more efficiently than solving single source shortest path problem

- Shortest paths in directed graphs is more general than in undirected graphs: we can replace every undirected edge with two anti-parallel edges of the same weight

Single Source Shortest Paths

Input: directed graph $G = (V, E)$, $s \in V$

$w : E \to \mathbb{R}_{\geq 0}$

Output: shortest paths from s to all other vertices $v \in V$

Reason for Considering Single Source Shortest Paths Problem

- We do not know how to solve s-t shortest path problem more efficiently than solving single source shortest path problem

- Shortest paths in directed graphs is more general than in undirected graphs: we can replace every undirected edge with two anti-parallel edges of the same weight

Single Source Shortest Paths

Input: directed graph $G = (V, E)$, $s \in V$

$w : E \to \mathbb{R}_{\geq 0}$

Output: $\pi[v], v \in V \setminus s$: the parent of v in shortest path tree

$d[v], v \in V \setminus s$: the length of shortest path from s to v

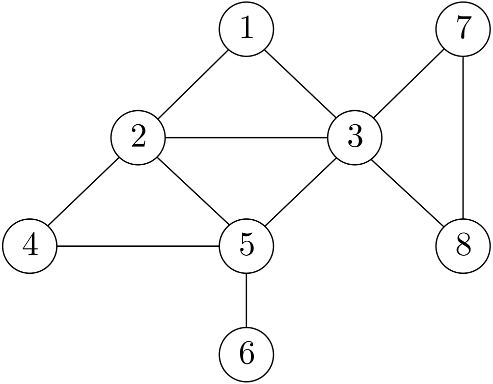
Q: How to compute shortest paths from s when all edges have weight 1?

Q: How to compute shortest paths from s when all edges have weight 1?

A: Breadth first search (BFS) from source s

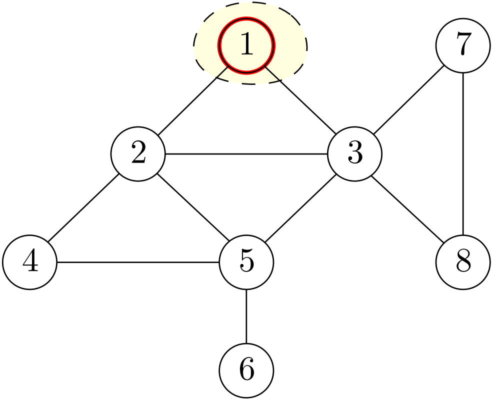
Q: How to compute shortest paths from s when all edges have weight 1?

A: Breadth first search (BFS) from source s



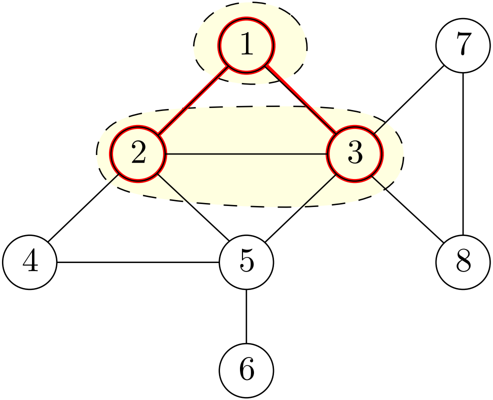
Q: How to compute shortest paths from s when all edges have weight 1?

A: Breadth first search (BFS) from source s



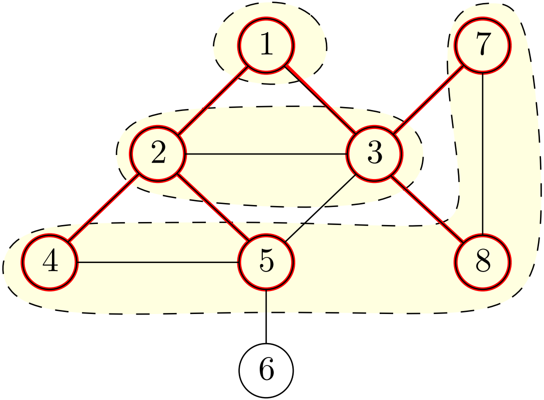
Q: How to compute shortest paths from s when all edges have weight 1?

A: Breadth first search (BFS) from source s



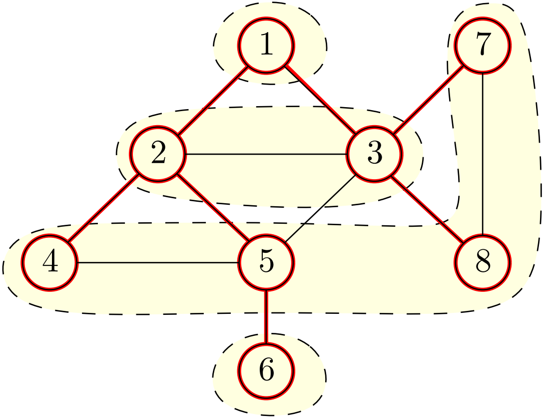
Q: How to compute shortest paths from s when all edges have weight 1?

A: Breadth first search (BFS) from source s



Q: How to compute shortest paths from s when all edges have weight 1?

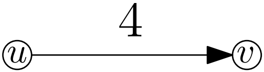
A: Breadth first search (BFS) from source s

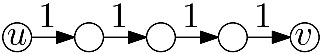


Assumption Weights $w(u, v)$ are integers (w.l.o.g).

Assumption Weights $w(u, v)$ are integers (w.l.o.g).

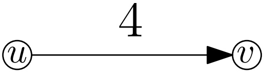
- An edge of weight $w(u, v)$ is equivalent to a pah of $w(u, v)$ unit-weight edges

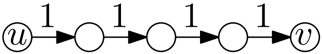




Assumption Weights $w(u,v)$ are integers (w.l.o.g).

- An edge of weight $w(u,v)$ is equivalent to a pah of $w(u,v)$ unit-weight edges





Shortest Path Algorithm by Running BFS

1: replace (u,v) of length $w(u,v)$ with a path of $w(u,v)$ unit-weight edges, for every $(u,v) \in E$
2: run BFS
3: $\pi[v] \leftarrow$ vertex from which v is visited
4: $d[v] \leftarrow$ index of the level containing v

Assumption Weights $w(u,v)$ are integers (w.l.o.g).

- An edge of weight $w(u,v)$ is equivalent to a pah of $w(u,v)$ unit-weight edges

Shortest Path Algorithm by Running BFS

1: replace (u,v) of length $w(u,v)$ with a path of $w(u,v)$ unit-weight edges, for every $(u,v) \in E$
2: run BFS
3: $\pi[v] \leftarrow$ vertex from which v is visited
4: $d[v] \leftarrow$ index of the level containing v

- Problem: $w(u,v)$ may be too large!

Assumption Weights $w(u, v)$ are integers (w.l.o.g).

- An edge of weight $w(u, v)$ is equivalent to a pah of $w(u, v)$ unit-weight edges

Shortest Path Algorithm by Running BFS

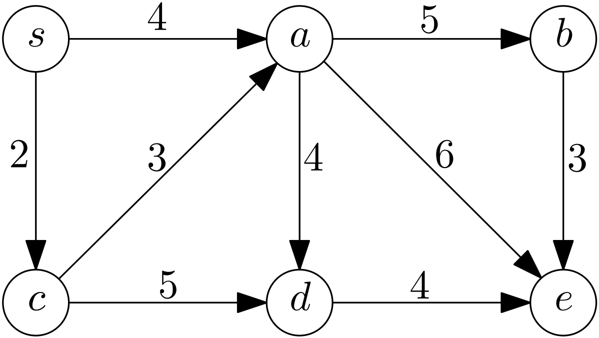
1: replace (u, v) of length $w(u, v)$ with a path of $w(u, v)$ unit-weight edges, for every $(u, v) \in E$
2: run BFS virtually
3: $\pi[v] \leftarrow$ vertex from which v is visited
4: $d[v] \leftarrow$ index of the level containing v

- Problem: $w(u, v)$ may be too large!

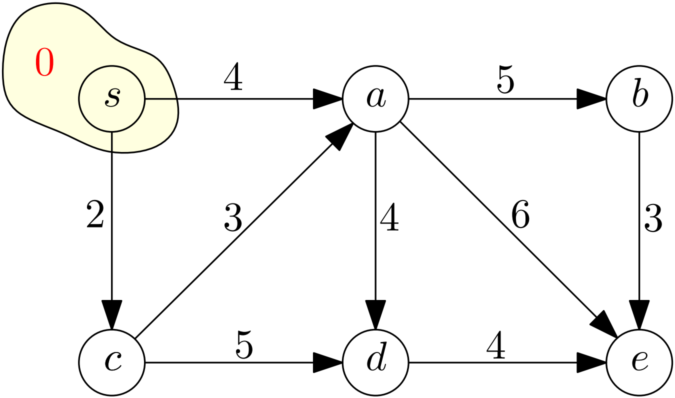
Shortest Path Algorithm by Running BFS Virtually

$S \leftarrow \{s\}, d(s) \leftarrow 0$
while $|S| \leq n$ **do**
    find a $v \notin S$ that minimizes $\min_{u \in S:(u,v) \in E} \{d[u] + w(u,v)\}$
    $S \leftarrow S \cup \{v\}$
    $d[v] \leftarrow \min_{u \in S:(u,v) \in E}\{d[u] + w(u,v)\}$

Virtual BFS: Example

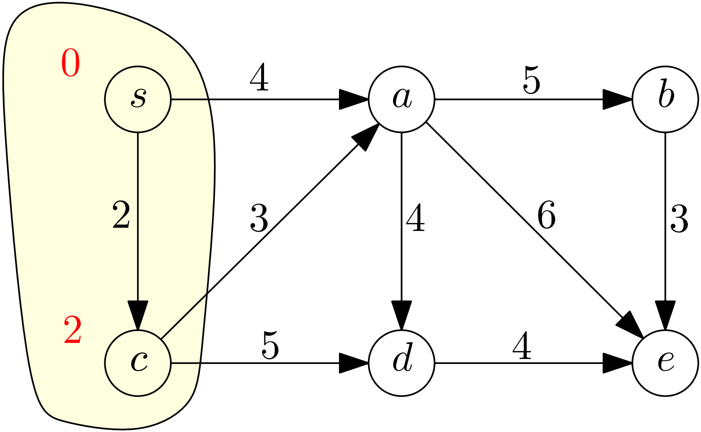


Virtual BFS: Example



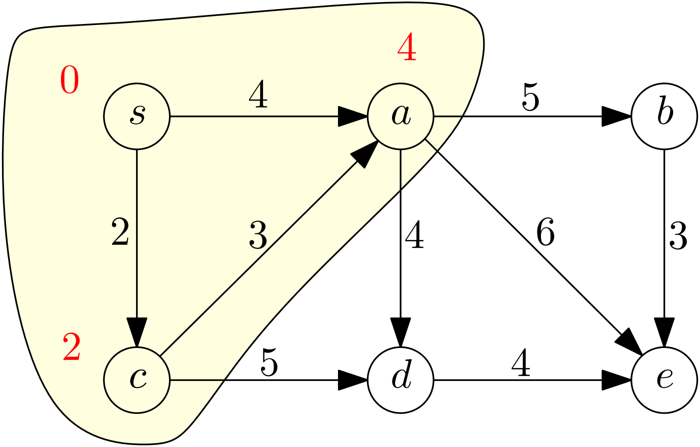
Time 0

Virtual BFS: Example



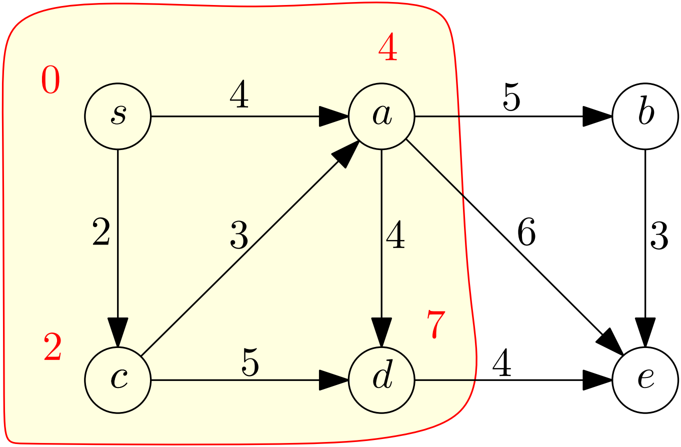
Time 2

Virtual BFS: Example



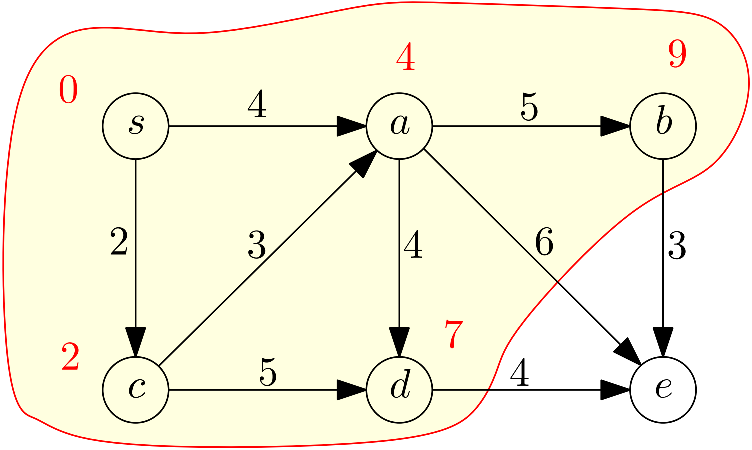
Time 4

Virtual BFS: Example



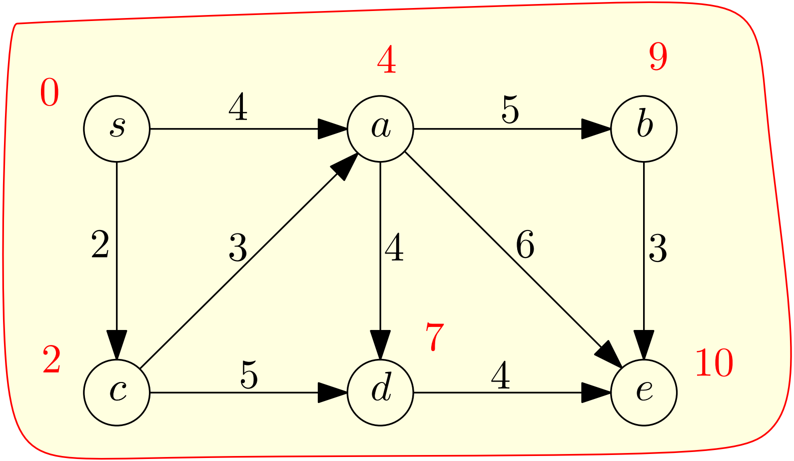
Time 7

Virtual BFS: Example



Time 9

Virtual BFS: Example



Time 10

Outline

Dijkstra's Algorithm

Dijkstra(G, w, s)

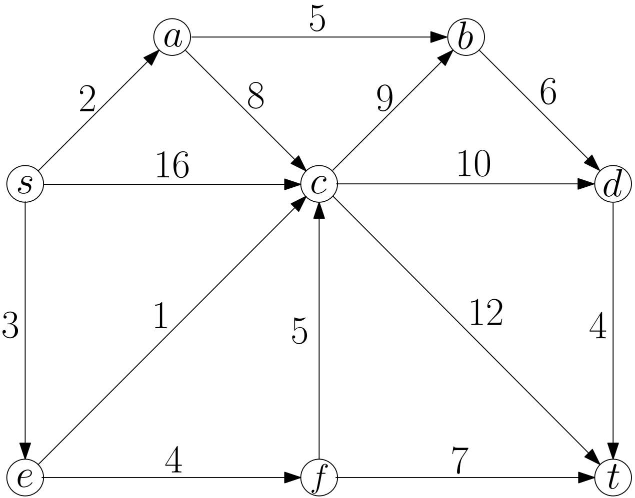
$S \leftarrow \emptyset, d(s) \leftarrow 0$ and $d[v] \leftarrow \infty$ for every $v \in V \setminus \{s\}$
while $S \neq V$ **do**
  $u \leftarrow$ vertex in $V \setminus S$ with the minimum $d[u]$
  add u to S
  for each $v \in V \setminus S$ such that $(u, v) \in E$ **do**
    if $d[u] + w(u, v) < d[v]$ **then**
      $d[v] \leftarrow d[u] + w(u, v)$
      $\pi[v] \leftarrow u$
return (d, π)

Dijkstra's Algorithm

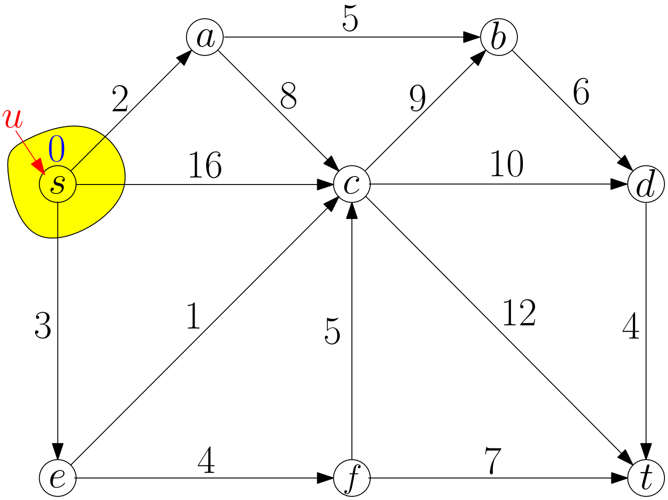
Dijkstra(G, w, s)

$S \leftarrow \emptyset, d(s) \leftarrow 0$ and $d[v] \leftarrow \infty$ for every $v \in V \setminus \{s\}$
while $S \neq V$ **do**
    $u \leftarrow$ vertex in $V \setminus S$ with the minimum $d[u]$
    add u to S
    for each $v \in V \setminus S$ such that $(u, v) \in E$ **do**
        if $d[u] + w(u, v) < d[v]$ **then**
            $d[v] \leftarrow d[u] + w(u, v)$
            $\pi[v] \leftarrow u$
return (d, π)

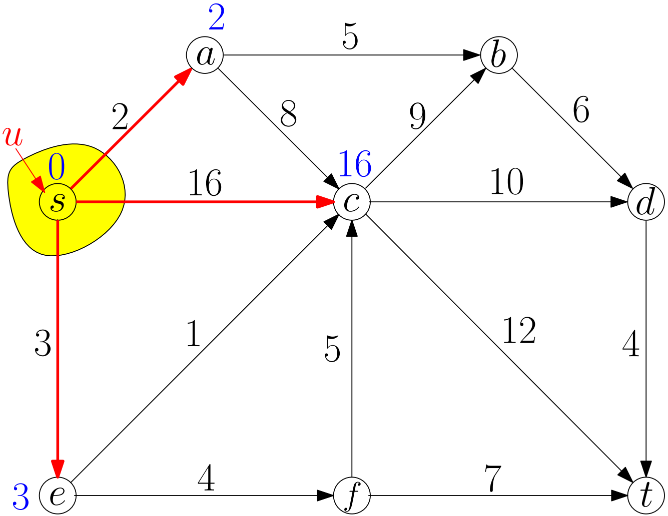
- Running time $= O(n^2)$

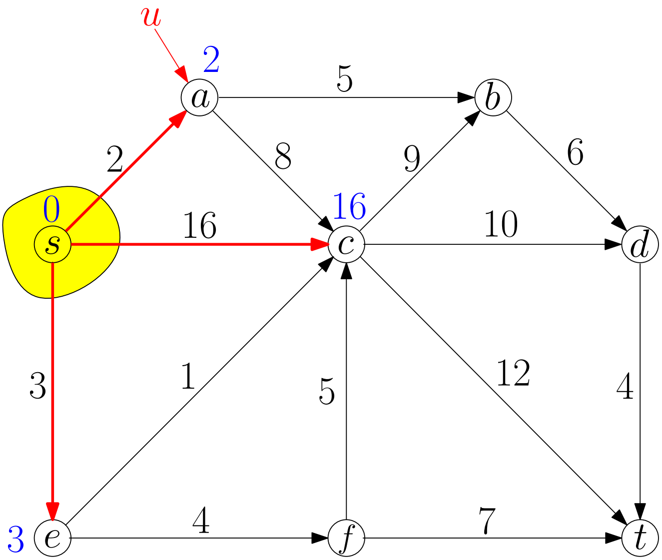


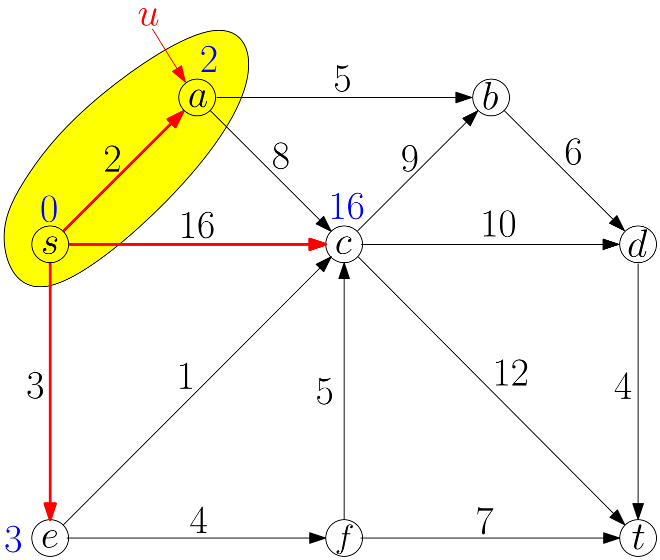
a
b
c
d
s
e
f
t
5
2
8
9
6
16
10
3
1
5
12
4
4
7

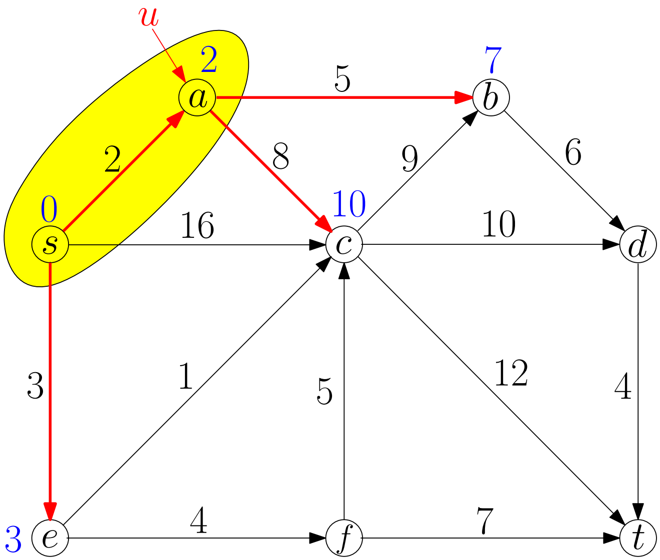


u
0
s
a
b
c
d
e
f
t
2
5
8
9
6
16
10
3
1
5
12
4
4
7

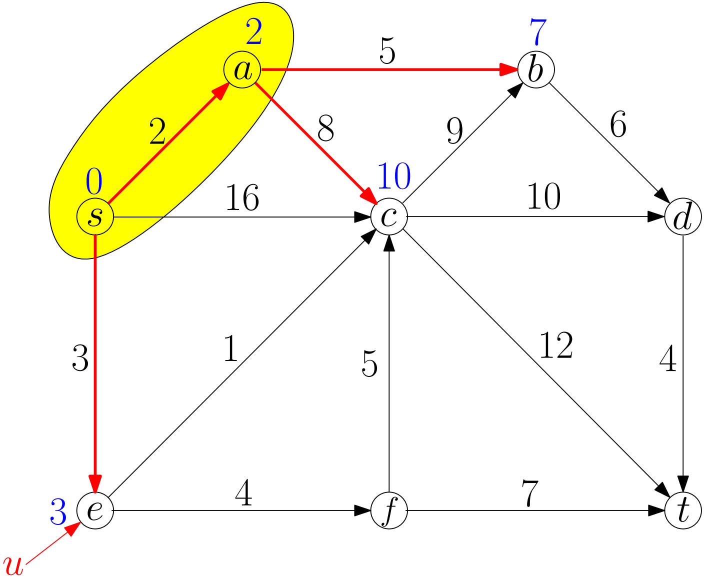

2
a
5
b
8
9
6
u
2
0
s
16
16
c
10
d
3
1
5
12
4
3
e
4
f
7
t

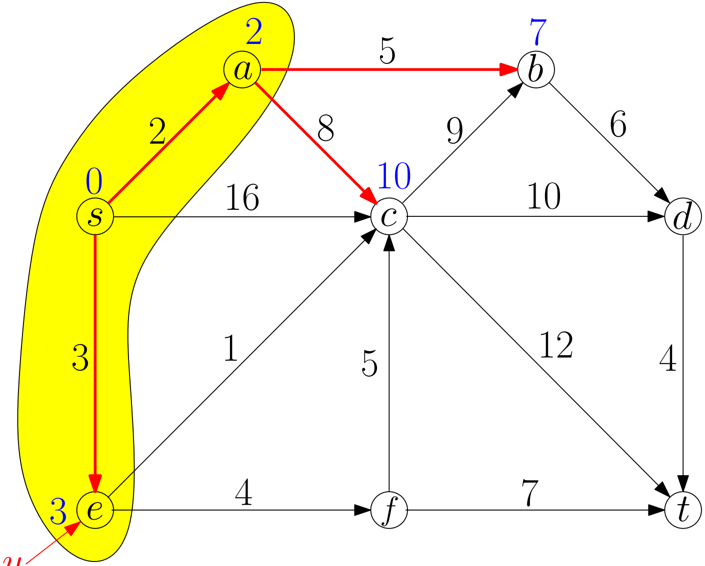

u
2
a
5
b
2
8
9
6
0
16
s
16
c
10
d
3
1
5
12
4
3
e
4
f
7
t


u
2
a
5
b
2
8
9
6
0
16
16
10
s
c
d
3
1
5
12
4
3
e
4
f
7
t

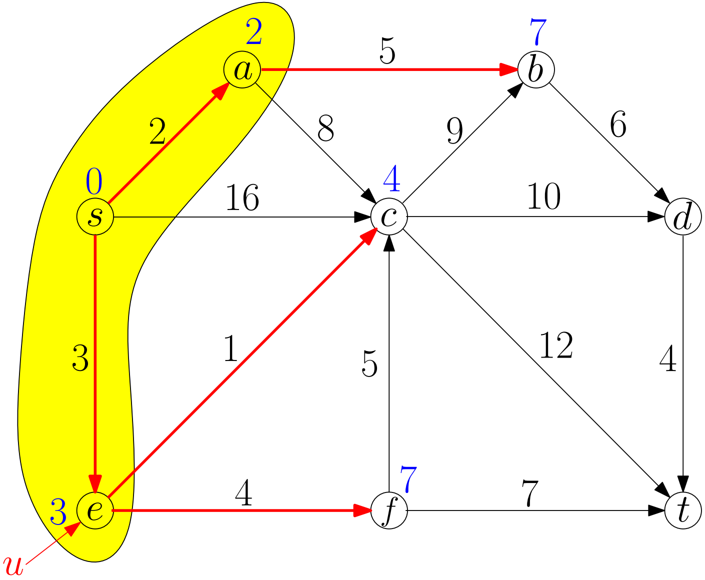


u
2
a
7
b
5
2
8
9
6
0
s
10
c
16
10
d
3
1
5
12
4
3
e
4
f
7
t

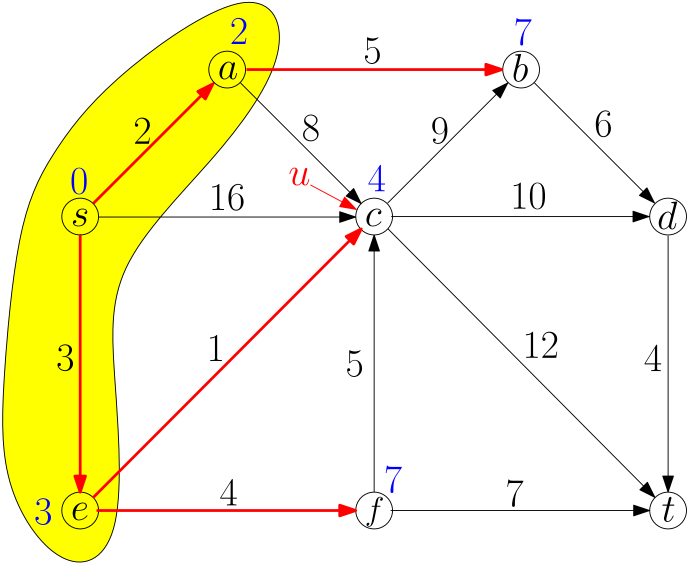

2
7
5
a
b
2
8
9
6
0
10
16
10
s
c
d
3
1
5
12
4
3
e
4
f
7
t
u

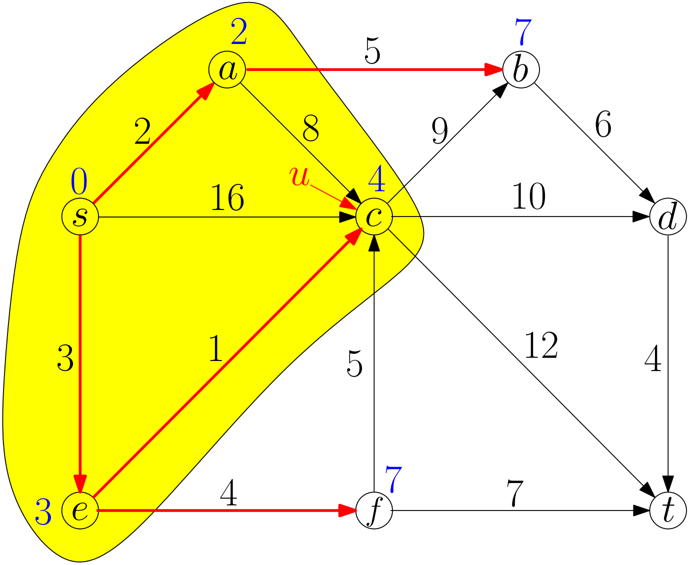


2
a
7
b
5
2
8
9
6
0
s
10
c
16
10
d
3
1
5
12
4
3
e
4
f
7
t
u

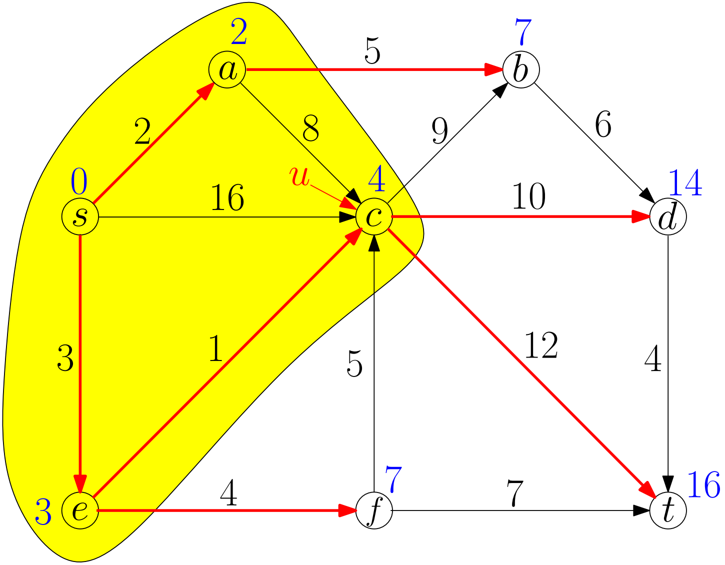


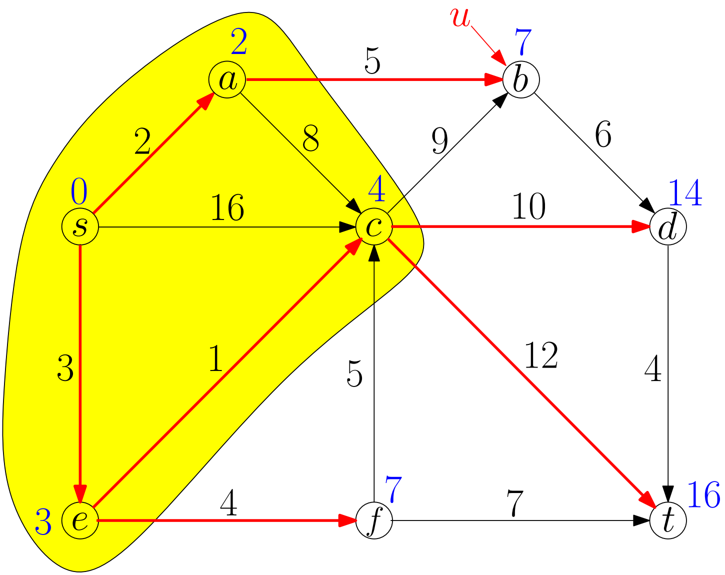
2
7
a
5
b
2
8
9
6
0
4
s
16
c
10
d
3
1
5
12
4
7
3
e
4
f
7
t
u

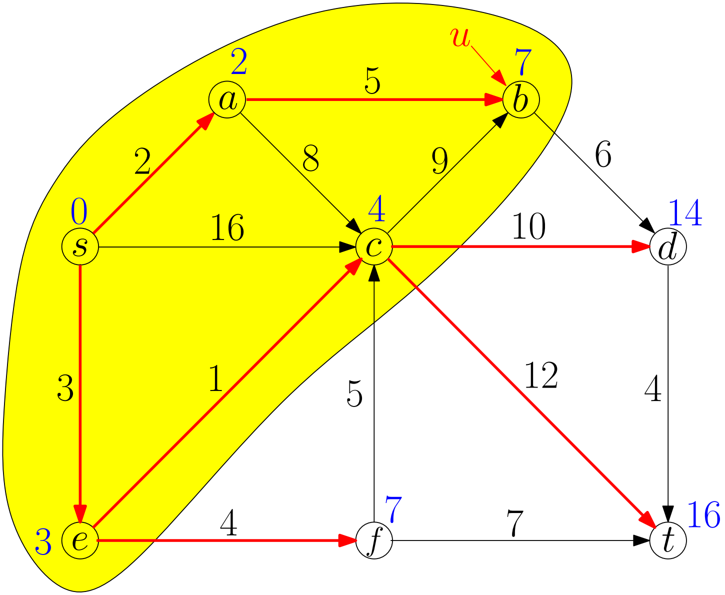

2
7
5
a
b
2
8
9
6
0
4
u
16
10
s
c
d
3
1
5
12
4
7
7
3
4
e
f
t

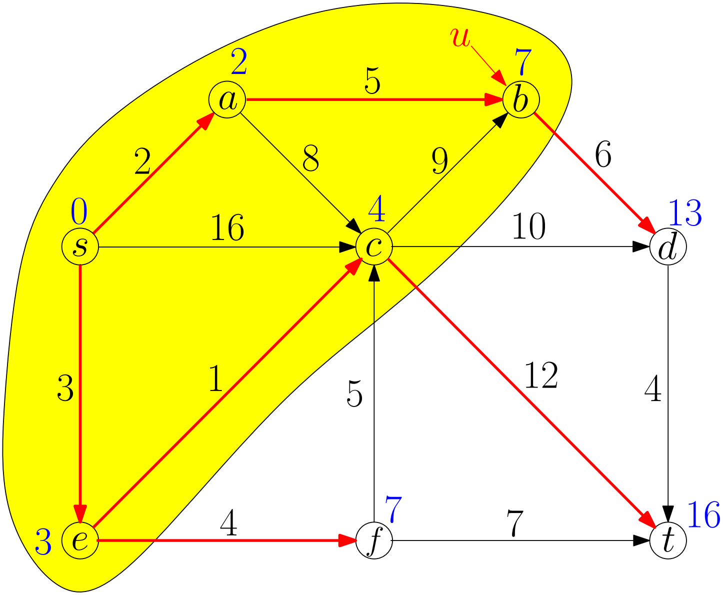


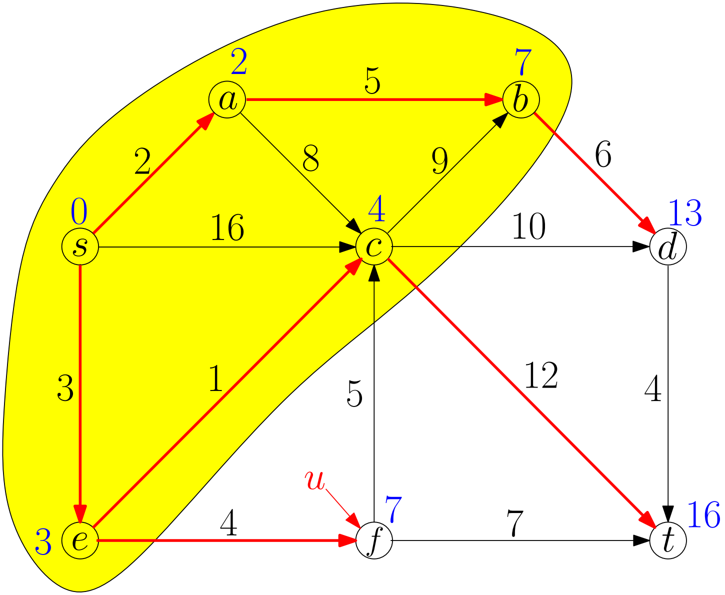
2
7
5
a
b
2
8
9
6
0
u
4
16
10
s
c
d
3
1
5
12
4
7
3
4
7
e
f
t

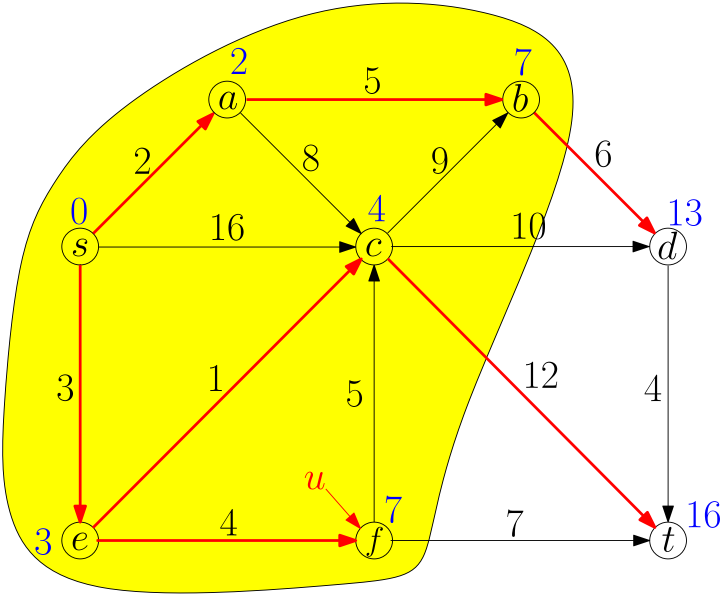

2
7
5
a
b
2
8
9
6
0
16
u
4
10
14
s
c
d
3
1
5
12
4
7
7
16
3
e
4
f
t


u
2
a
5
7
b
2
8
9
6
0
s
16
4
c
10
14
d
3
1
5
12
4
3
e
4
7
f
7
16
t

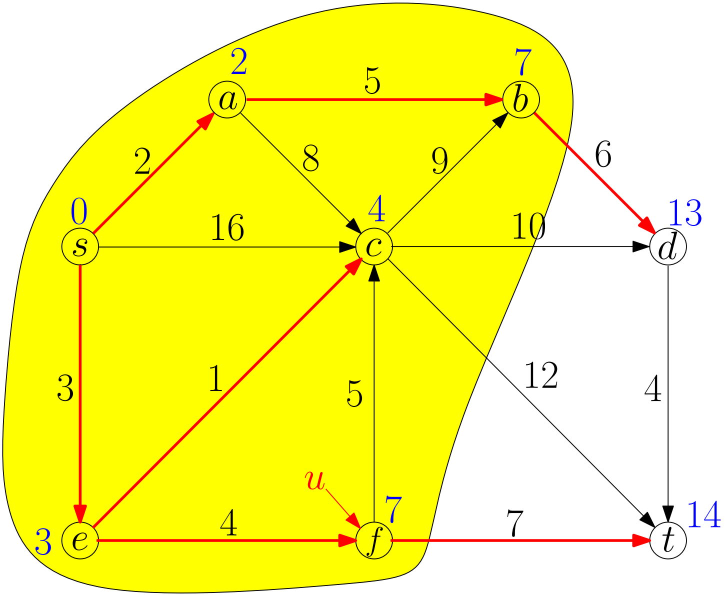

u
2
7
5
a
b
2
8
9
6
0
4
14
16
10
s
c
d
3
1
5
12
4
7
16
3
4
7
e
f
t


u
2
7
0
4
13
3
7
16
a
b
s
c
d
e
f
t
5
2
8
9
6
16
10
3
1
5
12
4
4
7

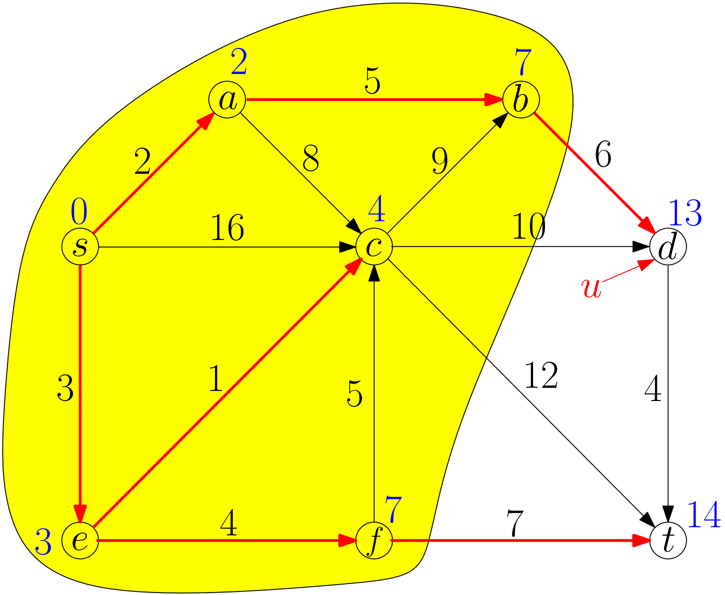

2
7
a
5
b
2
8
9
6
0
4
13
s
16
c
10
d
3
1
5
12
4
u
7
16
3
e
4
f
7
t



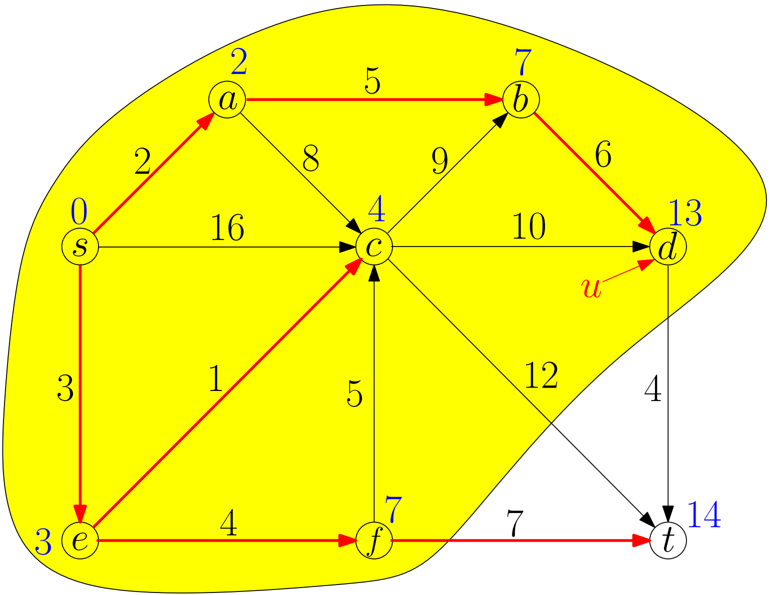
2
7
5
a
b
2
8
9
6
0
4
13
16
10
s
c
d
3
1
5
12
4
u
7
3
16
4
7
e
f
t



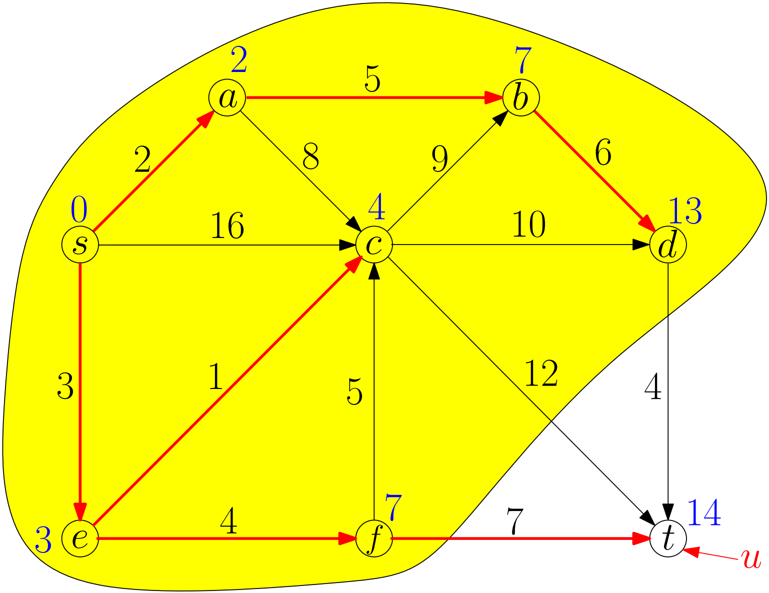
2
7
5
a
b
2
8
9
6
0
4
13
16
10
s
c
d
3
1
5
12
4
u
7
14
3
4
7
e
f
t



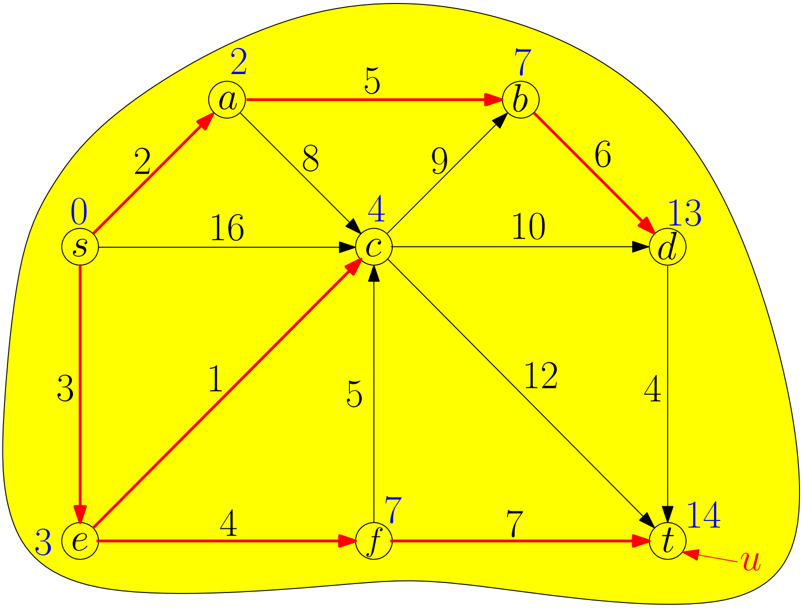
2
7
a
5
b
2
8
9
6
0
4
13
s
16
c
10
d
u
3
1
5
12
4
7
14
3
e
4
f
7
t



2
7
a
b
5
8
9
6
2
0
4
13
s
16
c
10
d
u
3
1
5
12
4
7
14
3
e
4
f
7
t



2
7
5
a
b
2
8
9
6
0
4
13
16
10
s
c
d
3
1
5
12
4
7
14
3
4
7
e
f
t
u



0
2
7
4
13
3
7
14
s
a
b
c
d
e
f
t
u
5
2
8
9
6
16
10
3
1
5
12
4
4
7

Improved Running Time using Priority Queue

Dijkstra(G, w, s)

```
s ← arbitrary vertex in G
S ← ∅, d(s) ← 0 and d[v] ← ∞ for every v ∈ V \ {s}
Q ← empty queue, for each v ∈ V: Q.insert(v, d[v])
while S ≠ V do
    u ← Q.extract_min()
    S ← S ∪ {u}
    for each v ∈ V \ S such that (u, v) ∈ E do
        if d[u] + w(u, v) < d[v] then
            d[v] ← d[u] + w(u, v), Q.decrease_key(v, d[v])
            π[v] ← u
return (π, d)
```

Recall: Prim's Algorithm for MST

MST-Prim(G, w)

$s \leftarrow$ arbitrary vertex in G
$S \leftarrow \emptyset, d(s) \leftarrow 0$ and $d[v] \leftarrow \infty$ for every $v \in V \setminus \{s\}$
$Q \leftarrow$ empty queue, for each $v \in V$: Q.insert$(v, d[v])$
while $S \neq V$ **do**
    $u \leftarrow Q$.extract_min()
    $S \leftarrow S \cup \{u\}$
    for each $v \in V \setminus S$ such that $(u, v) \in E$ **do**
        if $w(u, v) < d[v]$ **then**
            $d[v] \leftarrow w(u, v)$, Q.decrease_key$(v, d[v])$
            $\pi[v] \leftarrow u$
return $\{(u, \pi[u]) | u \in V \setminus \{s\}\}$

Improved Running Time

Running time:
$O(n) \times$ (time for extract_min) $+ O(m) \times$ (time for decrease_key)

Priority-Queue	extract_min	decrease_key	Time
Heap	$O(\log n)$	$O(\log n)$	$O(m \log n)$
Fibonacci Heap	$O(\log n)$	$O(1)$	$O(n \log n + m)$