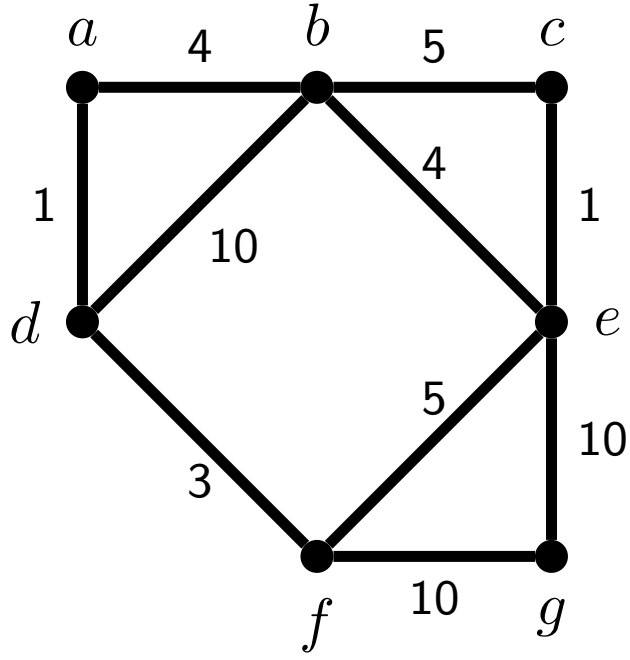
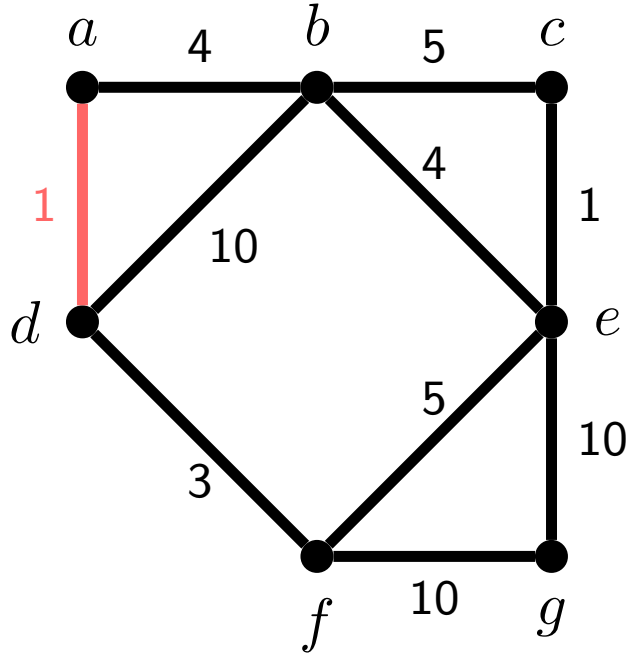
Kruskal's algorithm (for minimal trees)

Iteratively add smallest edge possible.
(Ties broken arbitrarily)



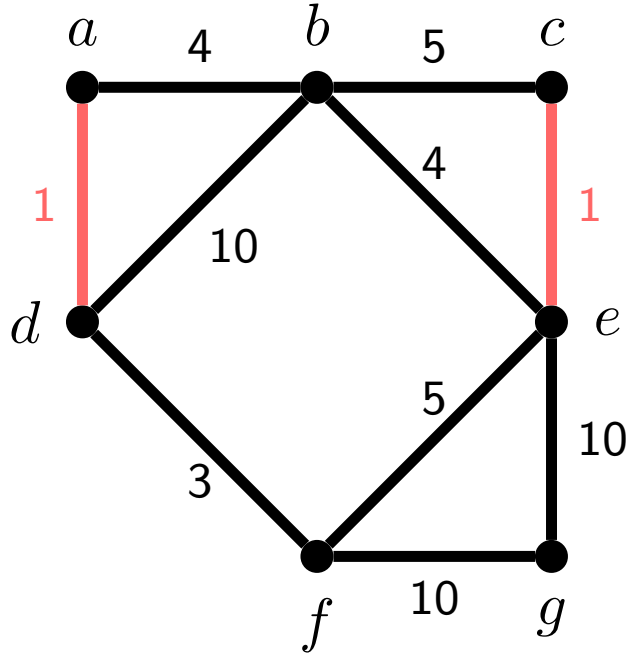
Kruskal's algorithm (for minimal trees)

Iteratively add smallest edge possible.
(Ties broken arbitrarily)



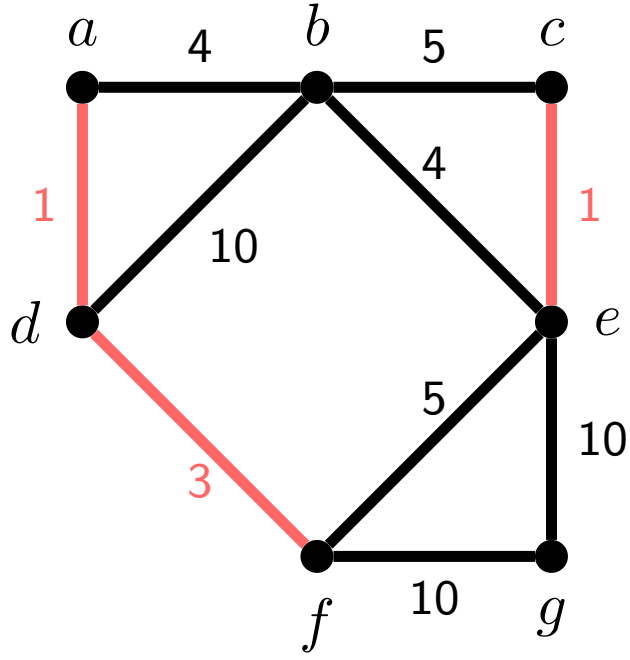
Kruskal's algorithm (for minimal trees)

Iteratively add smallest edge possible.
(Ties broken arbitrarily)



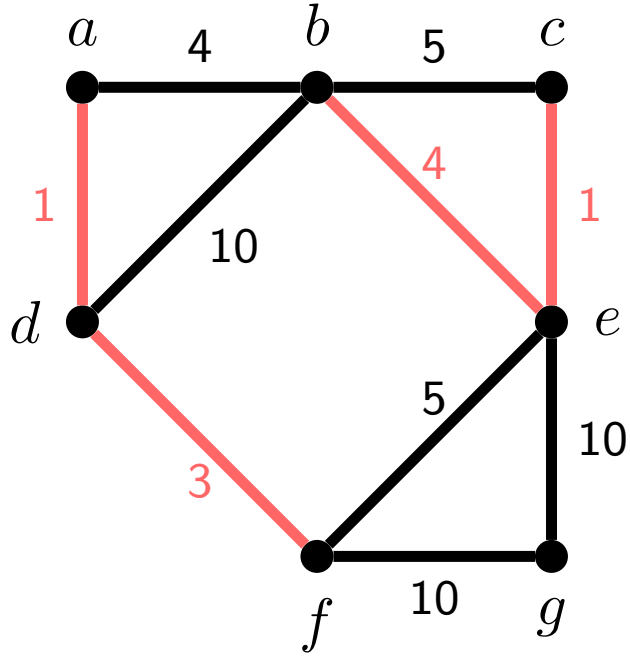
Kruskal's algorithm (for minimal trees)

Iteratively add smallest edge possible.
(Ties broken arbitrarily)



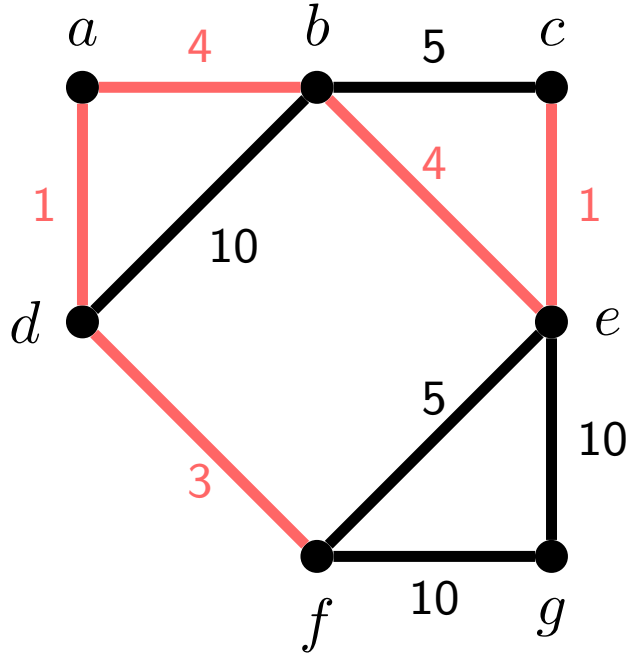
Kruskal's algorithm (for minimal trees)

Iteratively add smallest edge possible.
(Ties broken arbitrarily)



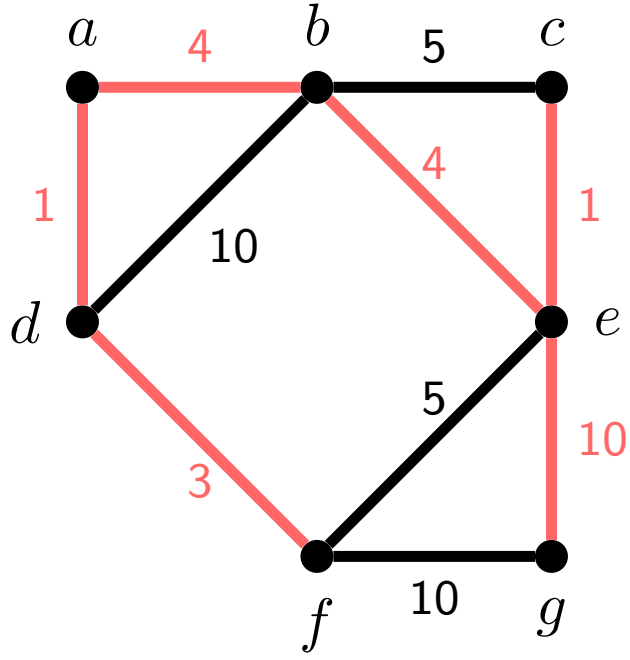
Kruskal's algorithm (for minimal trees)

Iteratively add smallest edge possible.
(Ties broken arbitrarily)



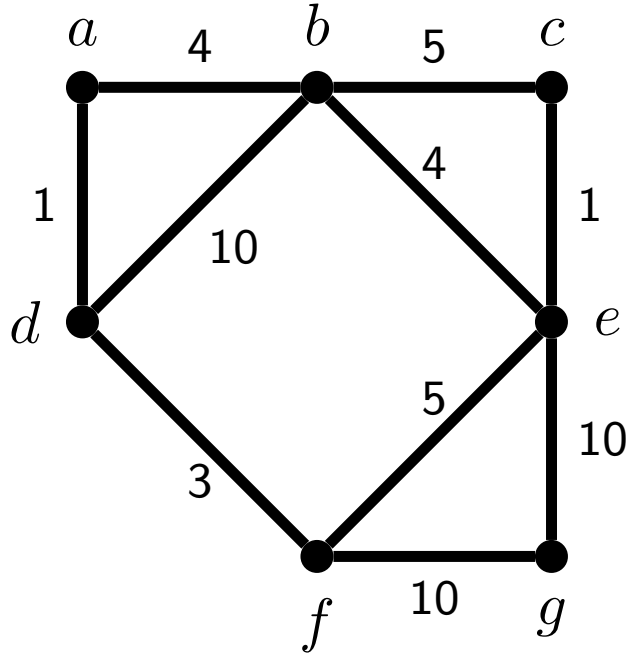
Kruskal's algorithm (for minimal trees)

Iteratively add smallest edge possible.
(Ties broken arbitrarily)



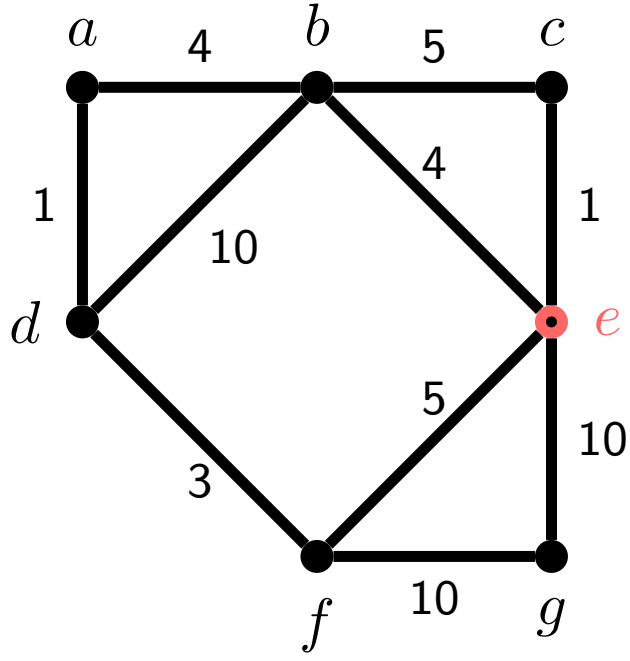
Prim's algorithm (for minimal trees)

Pick a vertex to start from.
Iteratively absorb smallest edge possible.
(Ties broken arbitrarily)



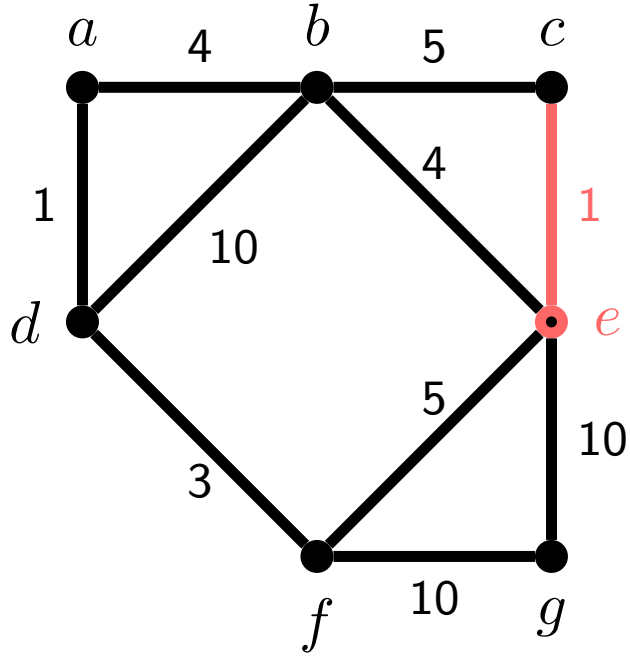
Prim's algorithm (for minimal trees)

Pick a vertex to start from.
Iteratively absorb smallest edge possible.
(Ties broken arbitrarily)



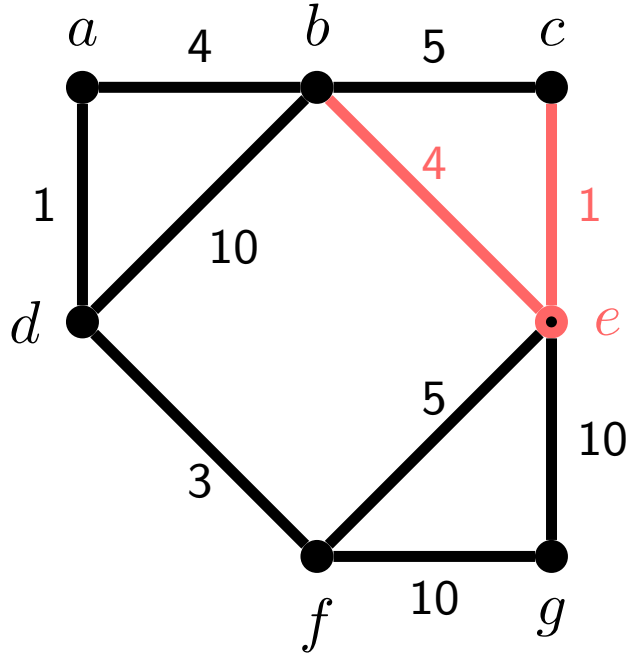
Prim's algorithm (for minimal trees)

Pick a vertex to start from.
Iteratively absorb smallest edge possible.
(Ties broken arbitrarily)



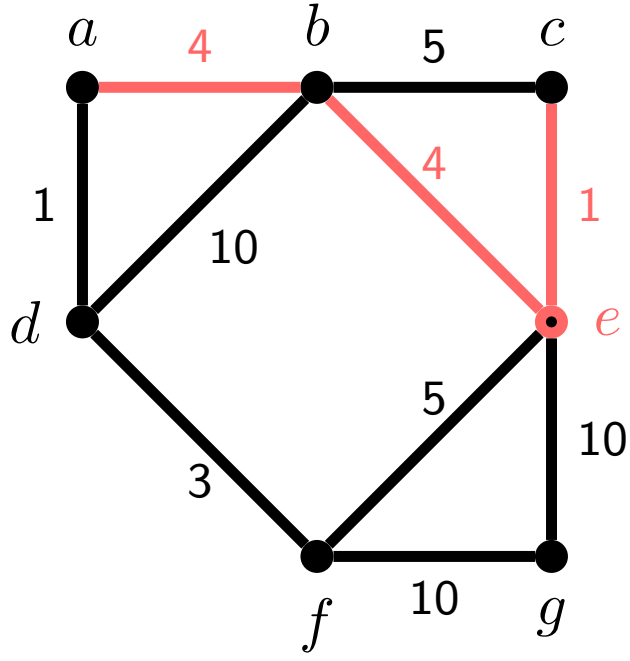
Prim's algorithm (for minimal trees)

Pick a vertex to start from.
Iteratively absorb smallest edge possible.
(Ties broken arbitrarily)



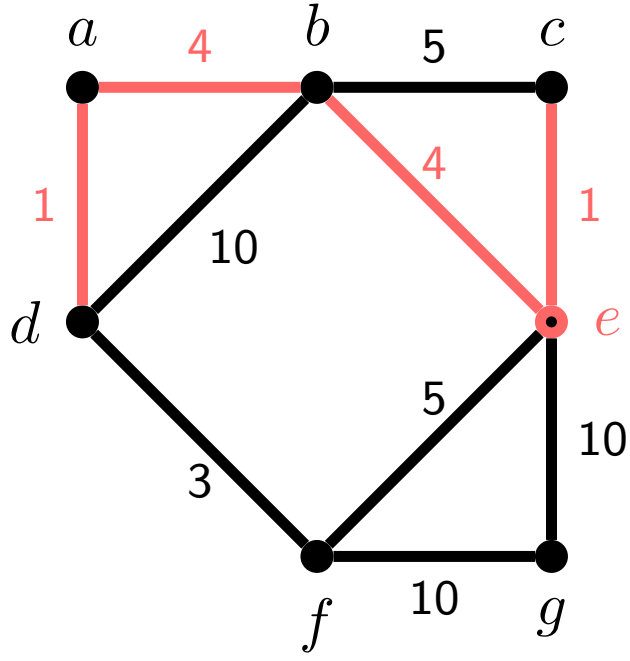
Prim's algorithm (for minimal trees)

Pick a vertex to start from.
Iteratively absorb smallest edge possible.
(Ties broken arbitrarily)



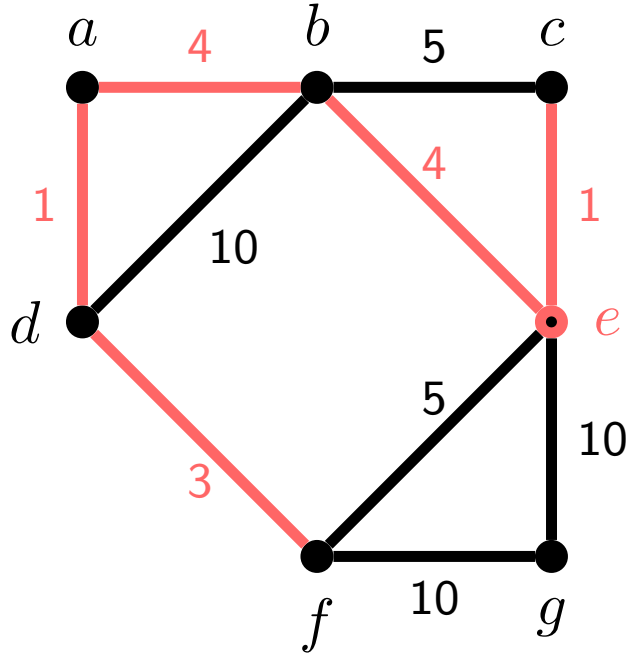
Prim's algorithm (for minimal trees)

Pick a vertex to start from.
Iteratively absorb smallest edge possible.
(Ties broken arbitrarily)



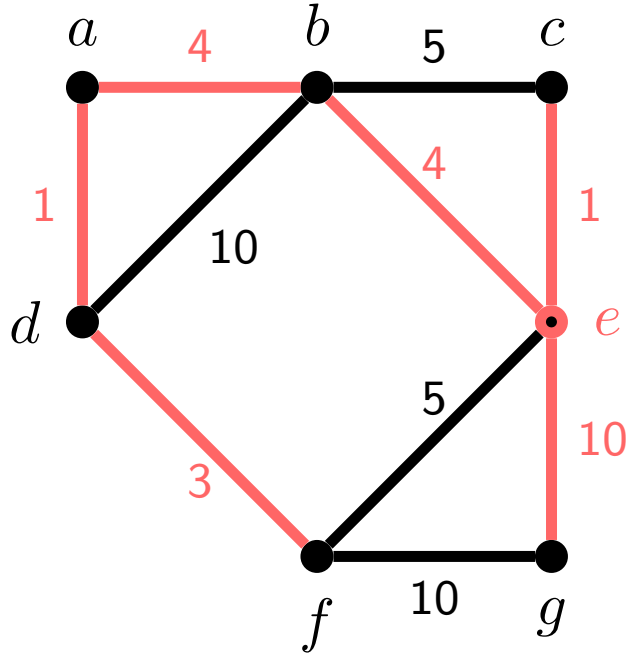
Prim's algorithm (for minimal trees)

Pick a vertex to start from.
Iteratively absorb smallest edge possible.
(Ties broken arbitrarily)



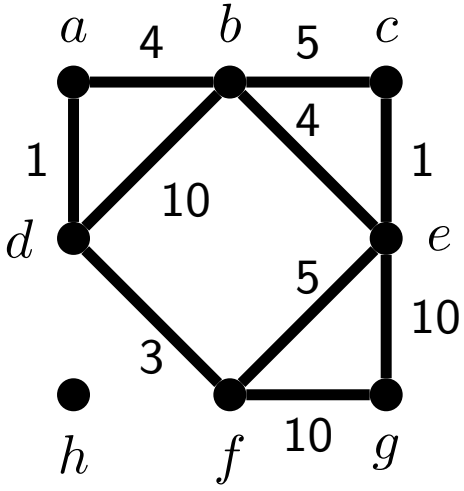
Prim's algorithm (for minimal trees)

Pick a vertex to start from.
Iteratively absorb smallest edge possible.
(Ties broken arbitrarily)



Dijkstra's algorithm (for minimal paths/distance)

Pick a vertex. Let $t(v)$ be the "temporary" distance from that vertex. Iteratively absorb closest vertices possible (minimal $t(v)$ and update distances to $\min(t(v), t(u) + w(uv))$.
(Ties broken arbitrarily)

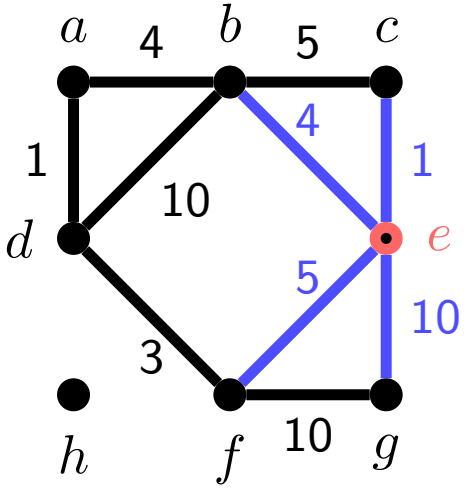


Distance from e:

add to S	$t(a)$	$t(b)$	$t(c)$	$t(d)$	$t(f)$	$t(g)$	$t(h)$

Dijkstra's algorithm (for minimal paths/distance)

Pick a vertex. Let $t(v)$ be the "temporary" distance from that vertex. Iteratively absorb closest vertices possible (minimal $t(v)$ and update distances to $\min(t(v), t(u) + w(uv))$.
(Ties broken arbitrarily)

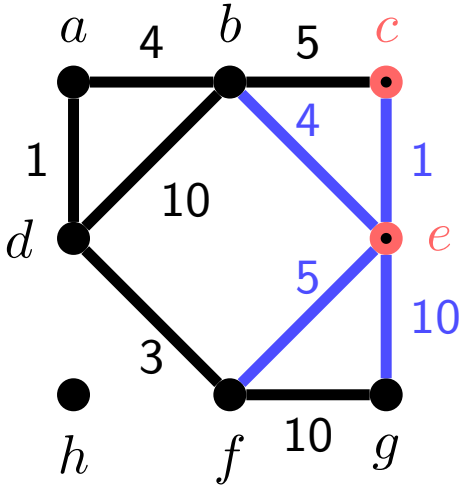


Distance from e:

add to S	$t(a)$	$t(b)$	$t(c)$	$t(d)$	$t(f)$	$t(g)$	$t(h)$
e	∞	4	1	∞	5	10	∞

Dijkstra's algorithm (for minimal paths/distance)

Pick a vertex. Let $t(v)$ be the "temporary" distance from that vertex. Iteratively absorb closest vertices possible (minimal $t(v)$ and update distances to $\min(t(v), t(u) + w(uv))$.
(Ties broken arbitrarily)

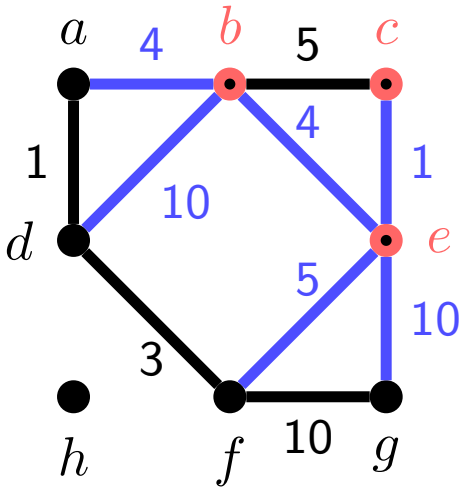


Distance from e:

add to S	$t(a)$	$t(b)$	$t(c)$	$t(d)$	$t(f)$	$t(g)$	$t(h)$
e	∞	4	1	∞	5	10	∞
c	∞	4	1	∞	5	10	∞

Dijkstra's algorithm (for minimal paths/distance)

Pick a vertex. Let $t(v)$ be the "temporary" distance from that vertex. Iteratively absorb closest vertices possible (minimal $t(v)$ and update distances to $\min(t(v), t(u) + w(uv))$.
(Ties broken arbitrarily)

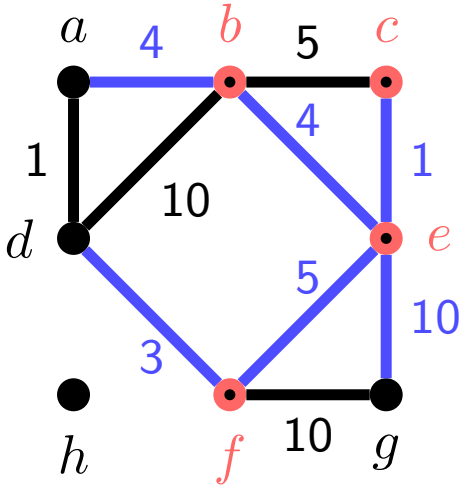


Distance from e:

add to S	$t(a)$	$t(b)$	$t(c)$	$t(d)$	$t(f)$	$t(g)$	$t(h)$
e	∞	4	1	∞	5	10	∞
c	∞	4	1	∞	5	10	∞
b	8	4	1	14	5	10	∞

Dijkstra's algorithm (for minimal paths/distance)

Pick a vertex. Let $t(v)$ be the "temporary" distance from that vertex. Iteratively absorb closest vertices possible (minimal $t(v)$ and update distances to $\min(t(v), t(u) + w(uv))$.
(Ties broken arbitrarily)

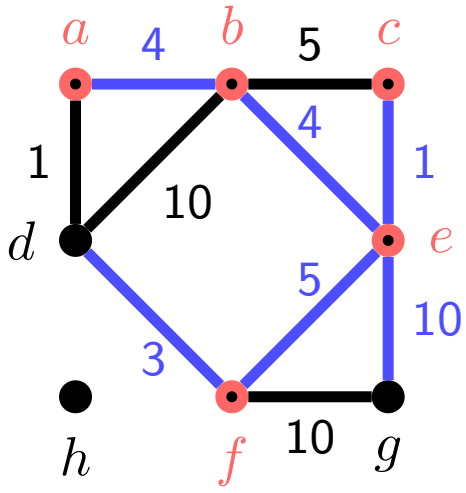


Distance from e:

add to S	$t(a)$	$t(b)$	$t(c)$	$t(d)$	$t(f)$	$t(g)$	$t(h)$
e	∞	4	1	∞	5	10	∞
c	∞	4	1	∞	5	10	∞
b	8	4	1	14	5	10	∞
f	8	4	1	8	5	10	∞

Dijkstra's algorithm (for minimal paths/distance)

Pick a vertex. Let $t(v)$ be the "temporary" distance from that vertex. Iteratively absorb closest vertices possible (minimal $t(v)$ and update distances to $\min(t(v), t(u) + w(uv))$.
(Ties broken arbitrarily)

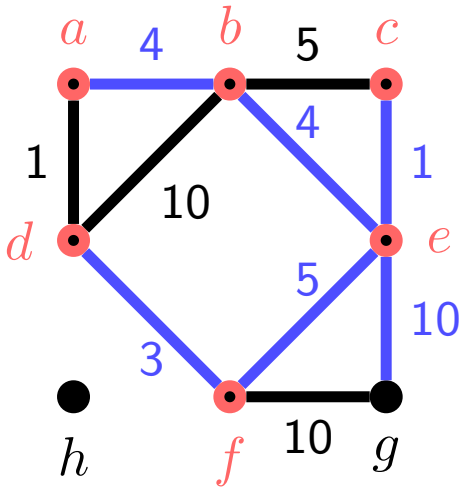


Distance from e:

add to S	$t(a)$	$t(b)$	$t(c)$	$t(d)$	$t(f)$	$t(g)$	$t(h)$
e	∞	4	1	∞	5	10	∞
c	∞	4	1	∞	5	10	∞
b	8	4	1	14	5	10	∞
f	8	4	1	8	5	10	∞
a	8	4	1	8	5	10	∞

Dijkstra's algorithm (for minimal paths/distance)

Pick a vertex. Let $t(v)$ be the "temporary" distance from that vertex. Iteratively absorb closest vertices possible (minimal $t(v)$ and update distances to $\min(t(v), t(u) + w(uv))$.
(Ties broken arbitrarily)

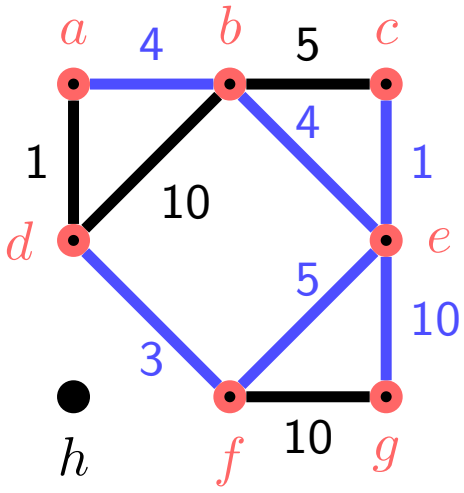


Distance from e:

add to S	$t(a)$	$t(b)$	$t(c)$	$t(d)$	$t(f)$	$t(g)$	$t(h)$
e	∞	4	1	∞	5	10	∞
c	∞	4	1	∞	5	10	∞
b	8	4	1	14	5	10	∞
f	8	4	1	8	5	10	∞
a	8	4	1	8	5	10	∞
d	8	4	1	8	5	10	∞

Dijkstra's algorithm (for minimal paths/distance)

Pick a vertex. Let $t(v)$ be the "temporary" distance from that vertex. Iteratively absorb closest vertices possible (minimal $t(v)$ and update distances to $\min(t(v), t(u) + w(uv))$.
(Ties broken arbitrarily)

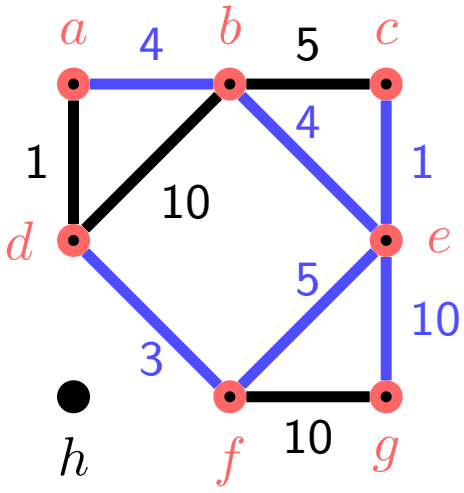


Distance from e:

add to S	$t(a)$	$t(b)$	$t(c)$	$t(d)$	$t(f)$	$t(g)$	$t(h)$
e	∞	4	1	∞	5	10	∞
c	∞	4	1	∞	5	10	∞
b	8	4	1	14	5	10	∞
f	8	4	1	8	5	10	∞
a	8	4	1	8	5	10	∞
d	8	4	1	8	5	10	∞
g	8	4	1	8	5	10	∞

Dijkstra's algorithm (for minimal paths/distance)

Pick a vertex. Let $t(v)$ be the "temporary" distance from that vertex. Iteratively absorb closest vertices possible (minimal $t(v)$ and update distances to $\min(t(v), t(u) + w(uv))$.
(Ties broken arbitrarily)



Distance from e:

add to S	$t(a)$	$t(b)$	$t(c)$	$t(d)$	$t(f)$	$t(g)$	$t(h)$
e	∞	4	1	∞	5	10	∞
c	∞	4	1	∞	5	10	∞
b	8	4	1	14	5	10	∞
f	8	4	1	8	5	10	∞
a	8	4	1	8	5	10	∞
d	8	4	1	8	5	10	∞
g	8	4	1	8	5	10	∞

(Stop when $t(v) = \infty$ for all $v \notin S$,
and set $d(e, v) = t(v)$.)