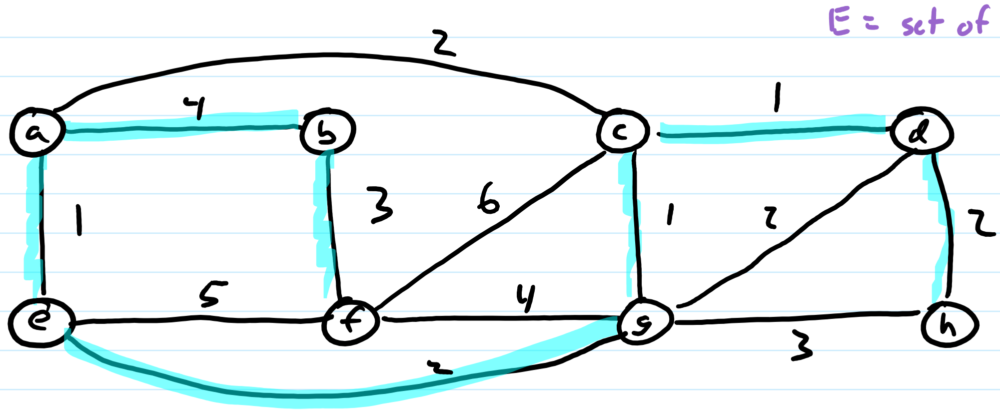
Light Edge Theorem

V= set of all vertices
E = set of all edges



THM: If $S, V-S$ is a cut, A is a proto-MST that respects that cut, and (u,v) is an edge of minimum weight across that cut ($u \in S$, $v \in V-S$ or vice versa), then $A \cup \{(u,v)\}$ is a proto-MST

Prim's Algorithm

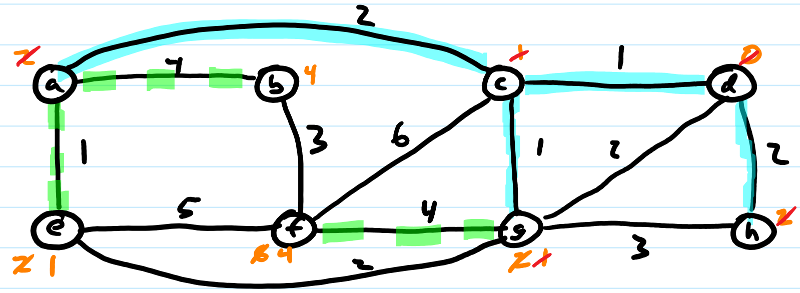
PRE: no negative weight edges, graph is connected

POST: T is a minimum spanning tree

```
for each v
   color[v], pred[v], d[v] ← IN_QUEUE, NIL, ∞
d[s] ← 0
S, T ← Ø

Q ← new PriorityQueue(d)

while Q not empty
   u ← dequeue(Q)
   S ← S U {u}
   T ← T U {(u, pred[u])}
   for each outneighbor v of u
      if color[v] = IN_QUEUE and d[v] > w(u, v)
         change_priority(Q, v, w(u, v))
         d[v] ← w(u, v)
         pred[v] ← u
   color[u] ← DONE
```



T is a proto-MST

for $u \in Q$, $d[u] = \min_{v \in S} w(u,v)$

$pred[u] = \operatorname{argmin}_{v \in S} w(u,v)$

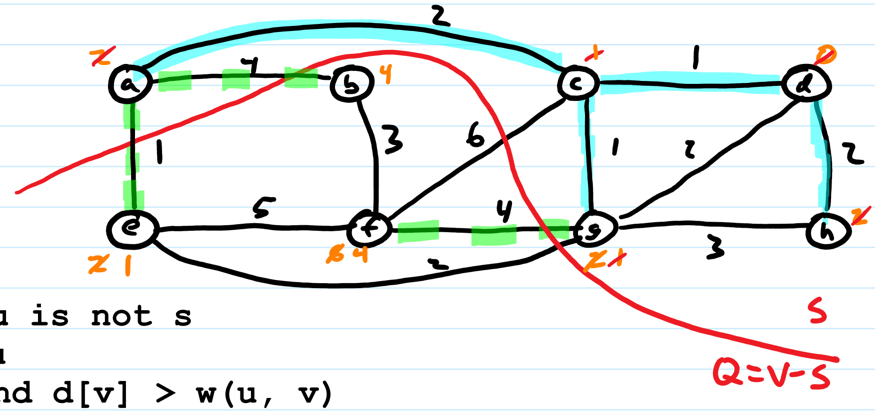
Prim's Algorithm

PRE: no negative weight edges, graph is connected
POST: T is a minimum spanning tree

```
for each v
   color[v], pred[v], d[v] ← IN_QUEUE, NIL, ∞
d[s] ← 0
S, T ← Ø

Q ← new PriorityQueue(d)

while Q not empty
   u ← dequeue(Q)
   S ← S U {u}
   T ← T U {(u, pred[u])} if u is not s
   for each outneighbor v of u
      if color[v] = IN_QUEUE and d[v] > w(u, v)
         change_priority(Q, v, w(u, v))
         d[v] ← w(u, v)
         pred[v] ← u
   color[u] ← DONE
```



T is a proto-MST

for $u \in Q$, $d[u] = \min_{v \in S} w(u,v)$

$pred[u] = \operatorname{argmin}_{v \in S} w(u,v)$

$\min_{u \in Q} d[u]$

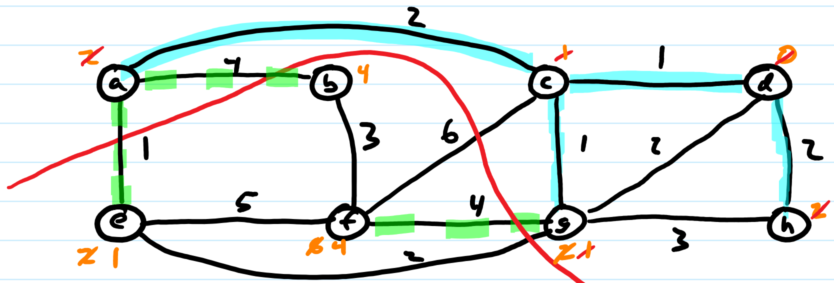
Prim's Algorithm

PRE: no negative weight edges, graph is connected
POST: T is a minimum spanning tree

```
for each v
   color[v], pred[v], d[v] ← IN_QUEUE, NIL, ∞
d[s] ← 0
S, T ← Ø

Q ← new PriorityQueue(d)

while Q not empty
   u ← dequeue(Q)
   S ← S U {u}
   T ← T U {(u, pred[u])} if u is not s
   for each outneighbor v of u
      if color[v] = IN_QUEUE and d[v] > w(u, v)
         change_priority(Q, v, w(u, v))
         d[v] ← w(u, v)
         pred[v] ← u
   color[u] ← DONE
```



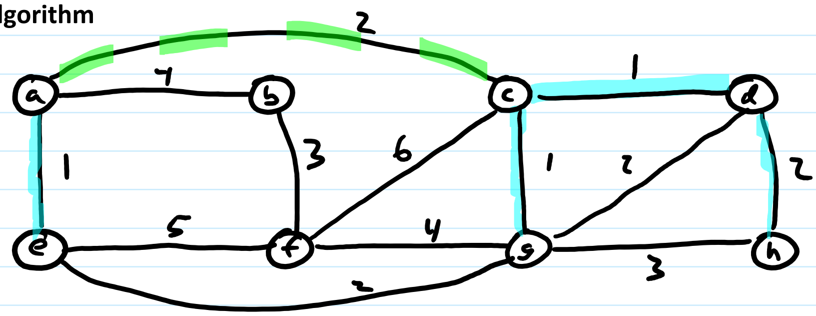
T is a proto-MST

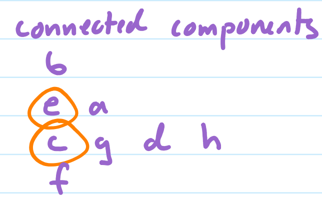
for $u \in Q$, $d[u] = \min_{v \in S} w(u,v)$

$pred[u] = \operatorname{argmin}_{v \in S} w(u,v)$

$\min_{u \in Q} \min_{v \in S} w(u,v)$

Kruskal's Algorithm



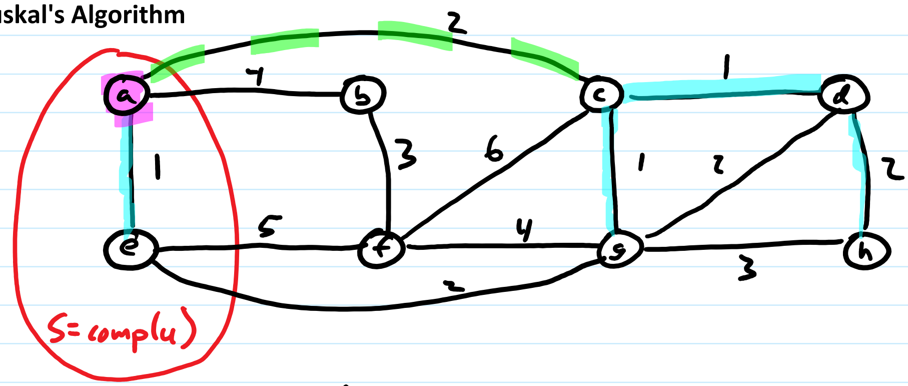


sort edges in order of increasing weight (a,e)(c,g)(c,d)(d,g)(a,c)(d,h)(e,g)(g,h)(b,f)(f,g)(a,b)(c,f)

T ← ∅

for each edge (u,v) in order of sort
    if find-set(u) ≠ find-set(v)
        add (u,v) to T
        union(u,v)

Kruskal's Algorithm



connected components
b
e a
c g d h
f

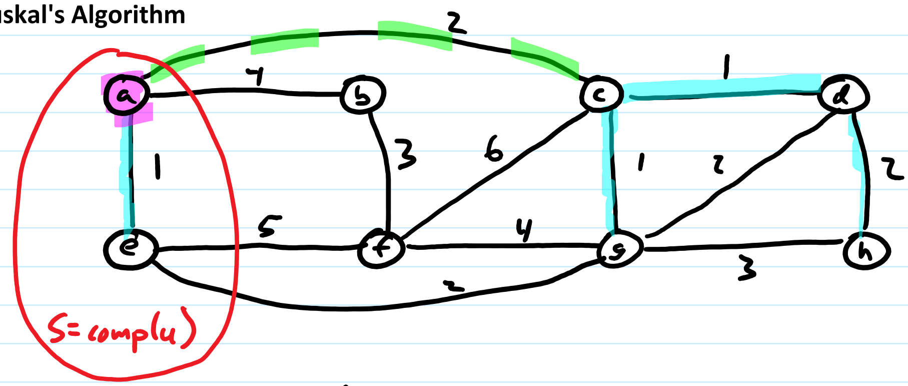
sort edges in order of increasing weight (a,e) (c,g) (c,d) (d,g) (a,c) (d,h) (e,g) (g,h) (b,f) (f,g) (a,b) (c,f)

T ← ∅

for each edge (u,v) in order of sort
    if find-set(u) ≠ find-set(v)
        add (u,v) to T
        union(u,v)

INV: T is a proto-MST
for (u,v) already processed, u and v are in the same connected component

Kruskal's Algorithm



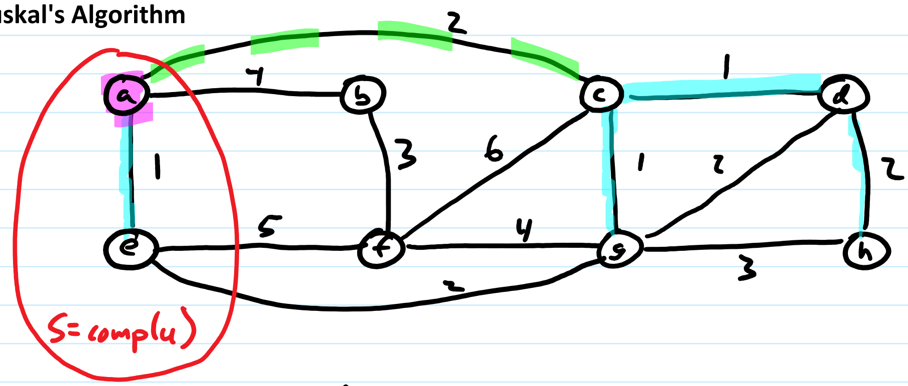
connected components
b
e a
c g d h
f

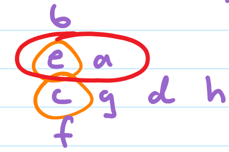
sort edges in order of increasing weight (a,e)(c,g)(c,d)(d,g)(a,c)(d,h)(e,g)(g,h)(b,f)(f,g)(a,b)(c,f)
T ← ∅
for each edge (u,v) in order of sort
    if find-set(u) ≠ find-set(v)
        → add (u,v) to T
        union(u,v)

INV: T is a proto-MST
for (u,v) already processed, u and v are in the same connected component

~~Suppose (u,v) is not a light edge across cut~~
Find (x,y) that is
$w(x,y) < w(u,v)$ — def. light edge
$x \in comp(u)$, $y \notin comp(u)$ — (x,y) crosses cut
(x,y) already processed — order of loop
x, y in same connected component — INV
⇒⇐

Kruskal's Algorithm





sort edges in order of increasing weight (a,e) (c,g) (c,d) (d,g) (a,c) (d,h) (e,g) (g,h) (b,f) (f,g) (a,b) (c,f)

$T \leftarrow \emptyset$

for each edge (u,v) in order of sort
 if find-set$(u) \neq$ find-set(v)
 add (u,v) to T
 union(u,v)

INV: T is a proto-MST
 for (u,v) already processed, u and v are in the same connected component

For any u,v there is a path $u = v_0 \to v_1 \to v_2 \to \cdots \to v_k = v$ in G
 each (v_i, v_{i+1}) is in the same connected component in T
 there is a path $v_i \rightsquigarrow v_{i+1}$ in T
 u, v are connected by $u = v_0 \rightsquigarrow v_1 \rightsquigarrow v_2 \rightsquigarrow \cdots \rightsquigarrow v_k = v$ in T