# Algorithms of Kruskal and Prim

Adapted from the CLRS book slides

# KRUSKAL'S ALGORITHM

$G = (V, E)$ is a connected, undirected, weighted graph. $w : E \to \mathbb{R}$.

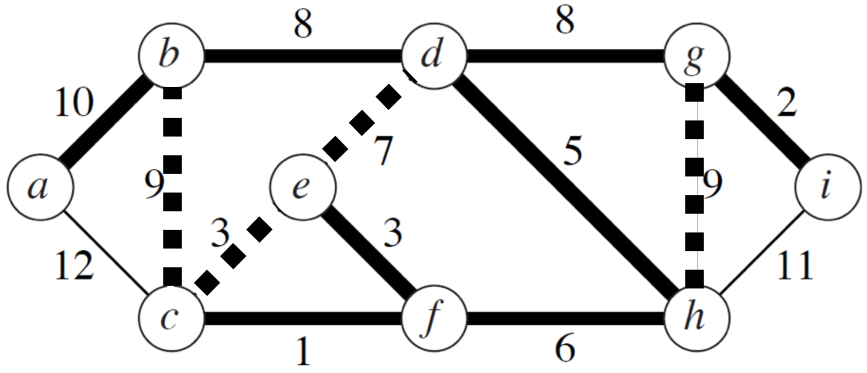
- Starts with each vertex being its own component.
- Repeatedly merges two components into one by choosing the light edge that connects them (i.e., the light edge crossing the cut between them).
- Scans the set of edges in monotonically increasing order by weight.
- Uses a disjoint-set data structure to determine whether an edge connects vertices in different components.

# PSUEDOCODE

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
create a single list of the edges in G.E
sort the list of edges into monotonically increasing order by weight w
for each edge (u, v) taken from the sorted list in order
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

# EXAMPLE

Let's see Kruskal's algorithm on this graph.




$(c, f)$ : safe
$(g, i)$ : safe
$(e, f)$ : safe
$(c, e)$ : reject
$(d, h)$ : safe
$(f, h)$ : safe
$(e, d)$ : reject
$(b, d)$ : safe
$(d, g)$ : safe
$(b, c)$ : reject
$(g, h)$ : reject
$(a, b)$ : safe

# ANALYSIS

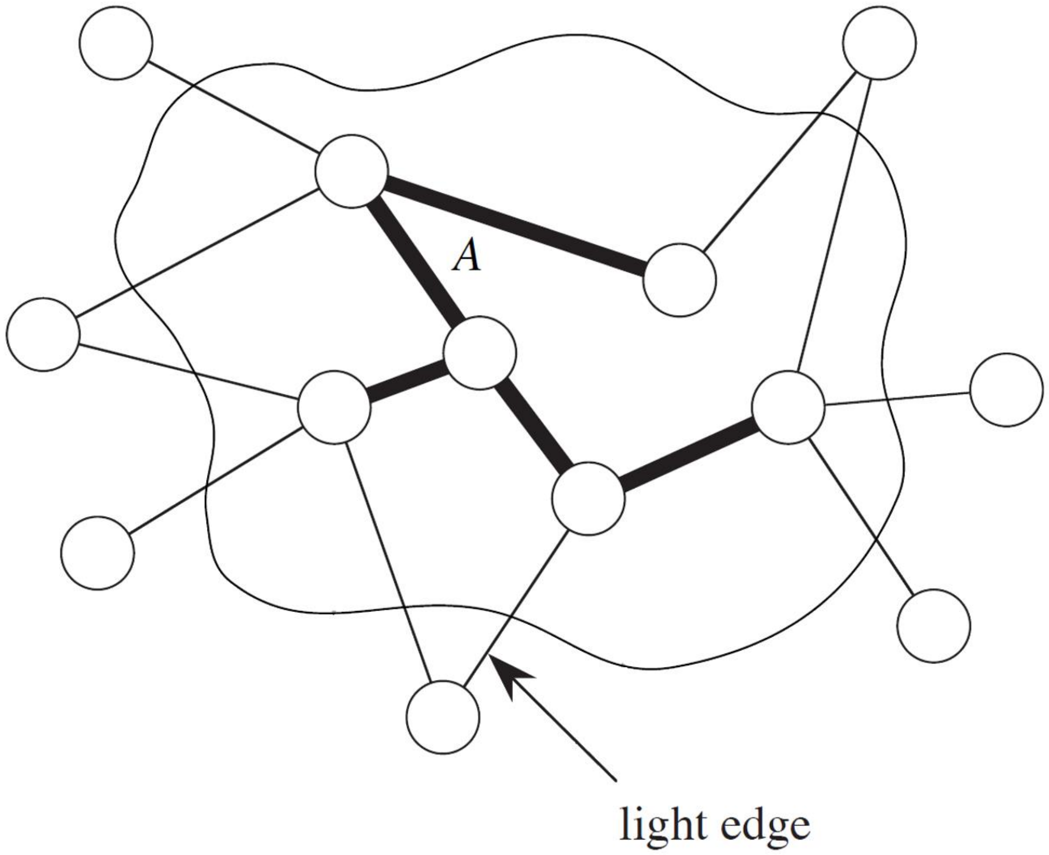
| | |
|---|---|
| Initialize $A$: | $O(1)$ |
| First **for** loop: | $\lvert V \rvert$ MAKE-SETS |
| Sort $E$: | $O(E \lg E)$ |
| Second **for** loop: | $O(E)$ FIND-SETS and UNIONS |

- Disjoint-set data structure: $O(\alpha(V))$ which is effectively constant time.
- Total time: $O((V + E)\alpha(V)) + O(E \lg E)$.
- Since $G$ is connected, $|E| \geq |V| - 1 \Rightarrow O(E\ \alpha(V)) + O(E \lg E)$.
- $\alpha(|V|) = O(\lg V) = O(\lg E)$.
- Therefore, total time is $O(E \lg E)$.
- $|E| \leq |V|^2 \Rightarrow \lg |E| = O(2 \lg V) = O(\lg V)$.
- Therefore, $O(E \lg V)$ time. (If edges are already sorted, $O(E\ \alpha(V))$, which is almost linear.)

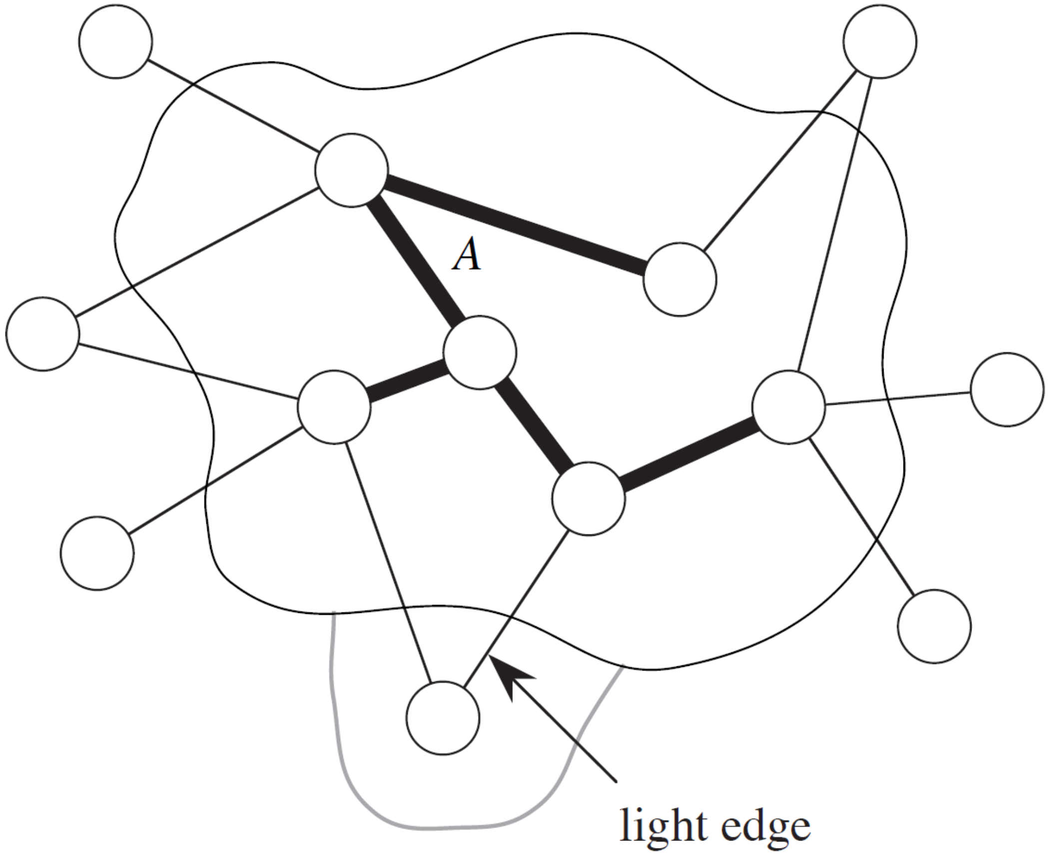
# PRIM'S ALGORITHM

- Builds one tree, so $A$ is always a tree.
- Starts from an arbitrary "root" $r$.
- At each step, find a light edge connecting $A$ to an isolated vertex. Such an edge must be safe for $A$. Add this edge to $A$.

# PRIM'S ALGORITHM (continued)



# PRIM'S ALGORITHM (continued)



# FINDING A LIGHT EDGE

How to find the light edge quickly?

Use a priority queue $Q$:

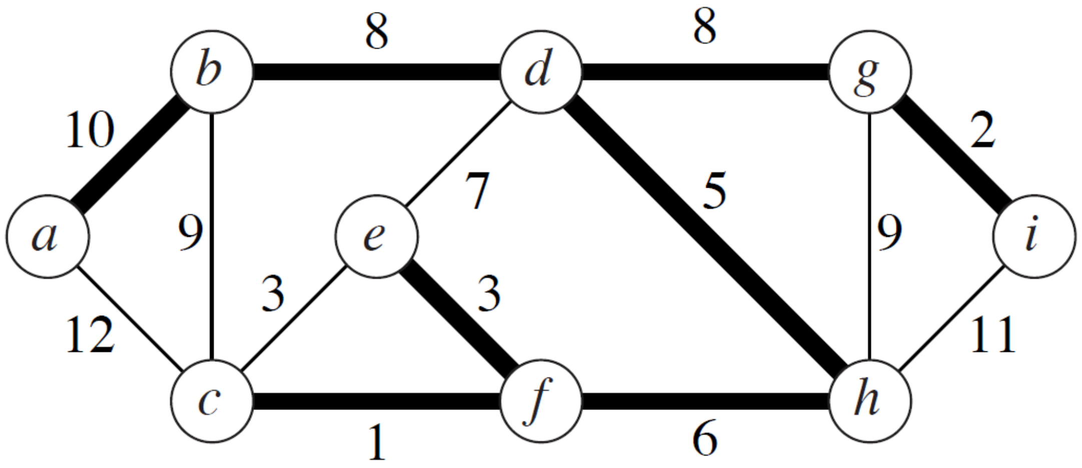
- Each object is a vertex *not* in $A$.
- $v.key$ is the minimum weight of any edge connecting $v$ to a vertex in $A$. $v.key = \infty$ if no such edge.
- $v.\pi$ is $v$'s parent in $A$.
- Maintain $A$ implicitly as $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.
- At completion, $Q$ is empty and the minimum spanning tree is $A = \{(v, v.\pi) : v \in V - \{r\}\}$.

# PSUEDOCODE

```
MST-PRIM(G, w, r)
for each vertex u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = ∅
for each vertex u ∈ G.V
    INSERT(Q, u)
while Q ≠ ∅
    u = EXTRACT-MIN(Q)             // add u to the tree
    for each vertex v in G.Adj[u]  // update keys of u's non-tree neighbors
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
            DECREASE-KEY(Q, v, w(u, v))
```

# EXAMPLE

Let’s see Prim’s algorithm on this graph. Pick a root.



# ANALYSIS

Depends on how the priority queue is implemented:

- Suppose $Q$ is a binary heap.

| | |
|---|---|
| Initialize $Q$ and first **for** loop: | $O(V \lg V)$ |
| Decrease key of $r$: | $O(\lg V)$ |
| **while** loop: | $\lvert V \rvert$ EXTRACT-MIN calls $\Rightarrow O(V \lg V)$ |
| | $\leq \lvert E \rvert$ DECREASE-KEY calls $\Rightarrow O(E \lg V)$ |
| Total: | $O(E \lg V)$ |

- Suppose DECREASE-KEY could take $O(1)$ *amortized* time.

  Then $\leq \lvert E \rvert$ DECREASE-KEY calls take $O(E)$ time altogether $\Rightarrow$ total time becomes $O(V \lg V + E)$.

  In fact, there is a way to perform DECREASE-KEY in $O(1)$ amortized time: Fibonacci heaps, mentioned in the introduction to Part V.