Minimum Spanning Trees

- MST Generic Algorithm
- Kruskal’s algorithm
- Prim’s algorithm

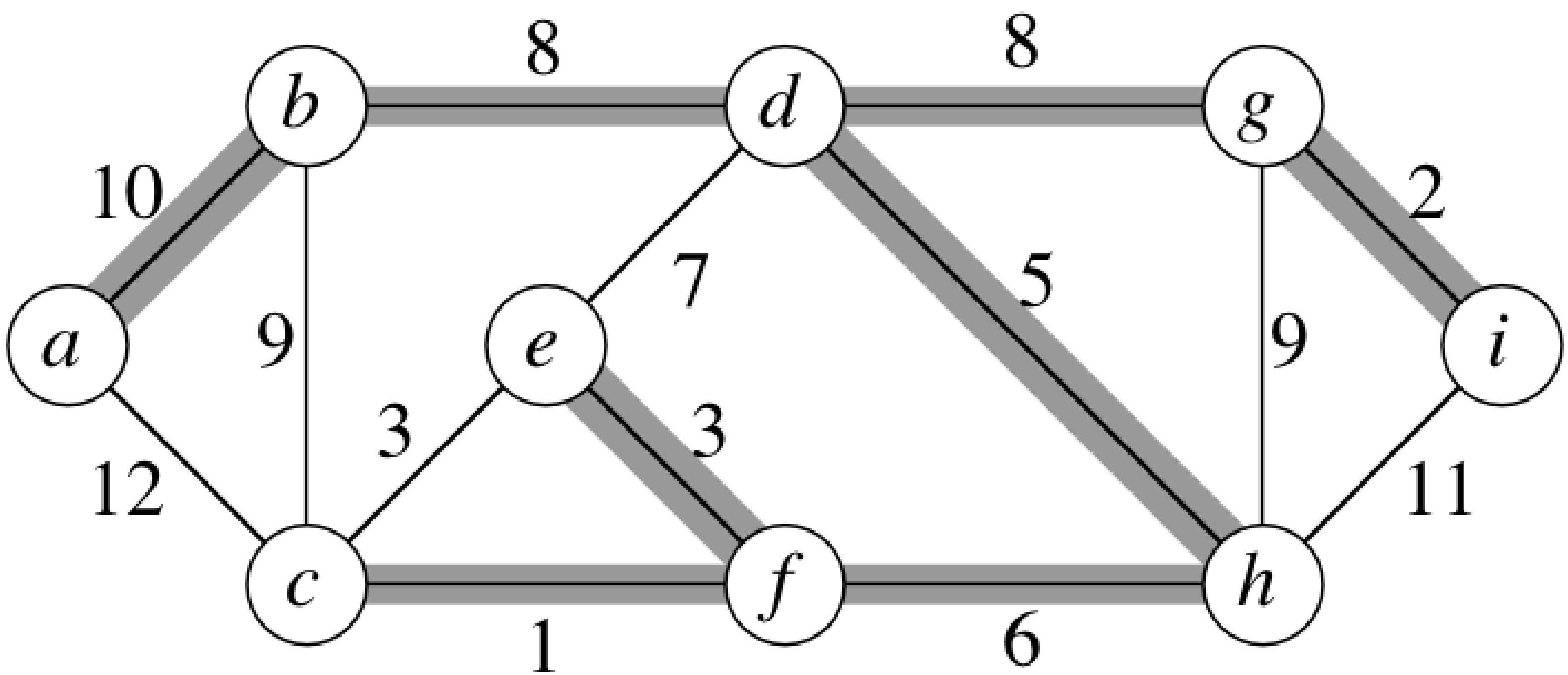
Definition

- Given a connected graph $G = (V, E)$, with weight function

  $w : E \;\; \text{-->} \;\; R$

- Min-weight connected subgraph
- Spanning tree T:
  - A tree that includes all nodes from V
  - $T = (V, E')$, where $E' \subseteq E$
  - Weight of T: $W(T) = \sum w(e)$
- Minimum spanning tree (MST):
  - A tree with minimum weight among all spanning trees

Example



MST

- MST for given G may not be unique

- Since MST is a spanning tree:
  - # edges : |V| - 1

- If the graph is unweighted:
  - All spanning trees have same weight

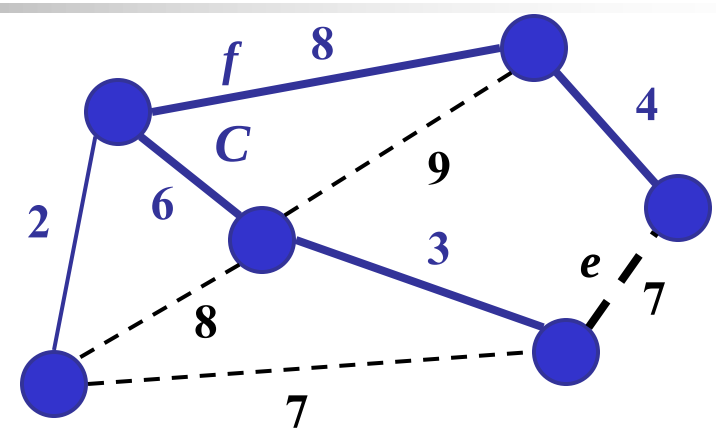
Cycle Property

Cycle Property:

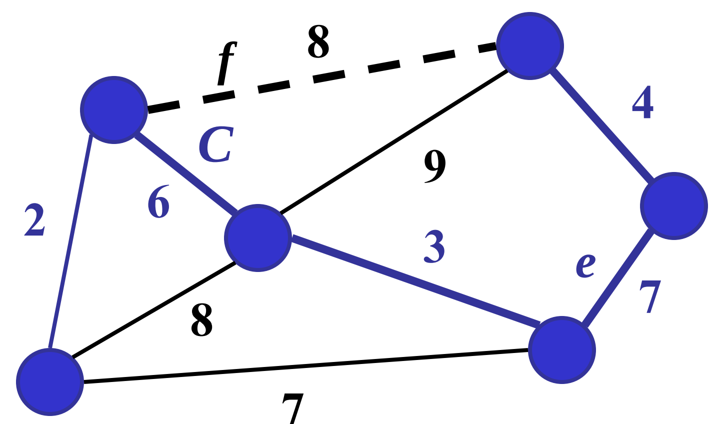
- Let ***T*** be a minimum spanning tree of a weighted graph ***G***
- Let ***e*** be an edge of ***G*** that is not in ***T*** and let C be the cycle formed by ***e*** with ***T***
- For every edge ***f*** of ***C***, ***weight***(***f***) ≤ ***weight***(***e***)

Proof:

- By contradiction
- If ***weight***(***f***) > ***weight***(***e***) we can get a spanning tree of smaller weight by replacing ***e*** with ***f***



Replacing *f* with *e* yields a better spanning tree



Cut Property

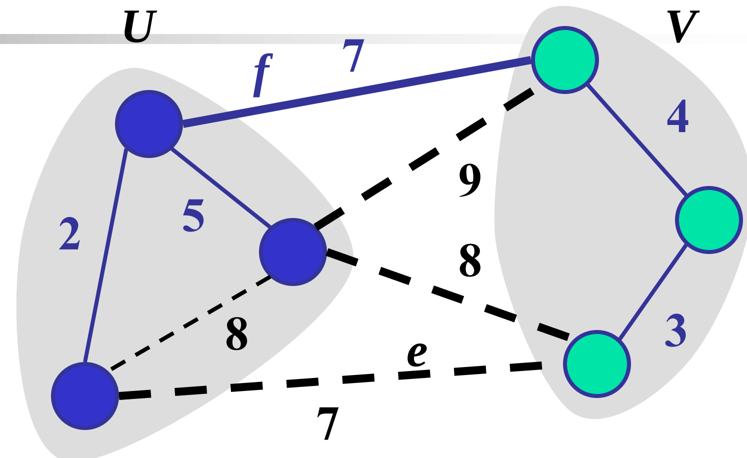
Cut Property:

- Consider a partition of the vertices of ***G*** into subsets ***U*** and ***V***
- Let ***e*** be an edge of minimum weight across the partition
- There is a minimum spanning tree of ***G*** containing edge ***e***

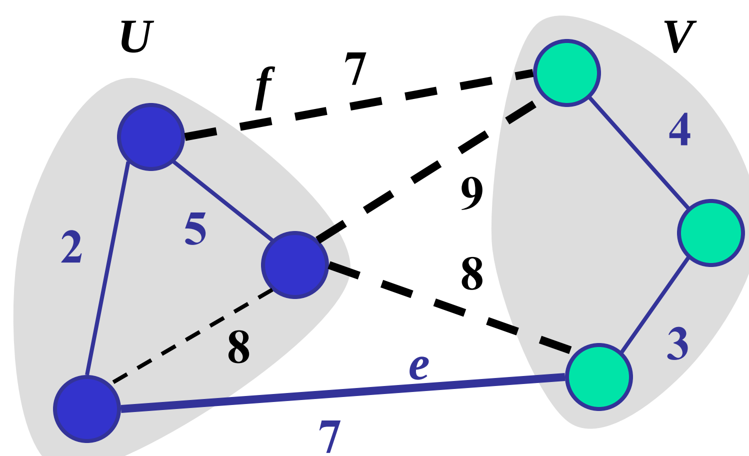
Proof:

- Let ***T*** be an MST of ***G***
- If ***T*** does not contain ***e*,** consider the cycle ***C*** formed by ***e*** with ***T*** and let ***f*** be an edge of ***C*** across the partition
- By the cycle property,

  $$\textit{weight}(f) \le \textit{weight}(e)$$
- Thus, $\textit{weight}(f) = \textit{weight}(e)$
- We obtain another MST by replacing ***f*** with ***e***



Replacing *f* with *e* yields another MST



Generic Algorithm

- Framework for $G = (V, E)$:
  - Goal: build a set of edges $A \subseteq E$
  - Start with A empty
  - Add edge into A one by one
  - At any moment, A is a subset of some MST for G

```
GENERIC-MST(G, w)
A ← Ø
while A is not a spanning tree
    do find an edge (u, v) that is safe for A
        A ← A ∪ {(u, v)}
return A
```

An edge is safe if adding it to A still maintains that A is a subset of a MST

Finding Safe Edges

- When A is empty, example of safe edges?
  - The edge with smallest weight
- Intuition:
  - Suppose $S \subseteq V$, --- a *cut* $(S, V\text{-}S)$
  - S and $V\text{-}S$ should be connected
    - By the ***crossing*** edge with the smallest weight !
    - That edge also called a ***light edge*** crossing the cut $(S, V\text{-}S)$
  - A cut $(S, V\text{-}S)$ respects edge set A
  - If no edges from A crosses the cut $(S, V\text{-}S)$

Safe-Edge Theorem

- Theorem:
  - Let A be a subset of some MST, $(S, V\text{-}S)$ be a cut that respects A, and (u, v) be a light edge crossing $(S, V\text{-}S)$. Then (u, v) is safe for A.
- Proof: let *T be an MST,* $A \subseteq T$, $A \neq T$. *Assume T does not contain* the light edge (*u, v*). *If it does, we are done. If not, we construct* another MST *T' that contains both A and (u, v).*

  $T \cup \{(u, v)\}$ *must contain a cycle, with edges on a simple path p*

  from *u to v* ***in T. u and v are on opposite sides of the cut***

  (S, V –*S), and at least one edge in T lies on p and crosses the cut.*

- Proof: (Contd)

  Let (x, y) *be any such edge – it cannot be in* A, because the cut respects A. *Since* (x, y) *is on the unique simple* path from u *to* v *in* T, *removing* (x, y) *breaks* T *into two* components. Adding (u, v) *reconnects them to form a new*

  spanning tree $T' = (T - \{(x, y)\})\ \{(u, v)\}$. *But* T' *is also an MST:* since (u, v) *is a light edge crossing* $(S, V - S)$ *and* (x, y) *also* crosses the cut, $w(u, v) \leq w(x, y)$ *and* $w(T') = w(T) - w(x, y) + w(u,v) \leq w(T)$. *The minimality of* T *implies* $w(T) \leq w(T')$, *so* T' *must be* minimal, also.

  Is (u, v) *safe? Since* $A \subseteq T$ *and* $(x, y) \notin A$, *we have* $A \subseteq T'$. Thus $A \cup \{(u, v)\} \subseteq T'$. *Since* T' *is an MST,* (u, v) *is safe for* A.

Safe-Edge Theorem

- Corollary:
  - Let (u, v) be a light edge crossing $(V', V-V')$, where graph $G' = (V', E')$ is a connected component of the graph (forest) $G'' = (V, A)$, then (u, v) is safe for A.

Greedy Approach:
Based on the generic algorithm and the corollary,
to compute MST we only need a way to find
a safe edge at each moment.

Corollary: Let *G = (V, E) be a connected undirected graph* with a real-valued weight function *w defined on E. Let A be a* subset of *E that is included in some minimum spanning tree for G, let C = (V_C, E_C) be a connected component (tree) in the forest G_A = (V, A). If (u, v) is a light edge connecting C to some other* component in *G_A, then (u, v) is safe for A.*

- Proof**: the cut (*V_C, V – V_C) respects A, and (u, v) is a light edge for*** this cut. Therefore (*u, v) is safe for A*.

Kruskal's Algorithm

- Start with *A* empty, and each vertex being its own connected component
- Repeatedly merge two components by connecting them with a light edge crossing them
- Two issues:
  - Maintain sets of components
  - Choose light edges

Disjoint set data structure

Scan edges from low to high weight

Pseudo-code

KRUSKAL(V, E, w)

$A \leftarrow \emptyset$

for each vertex $v \in V$

  do MAKE-SET(v)

sort E into nondecreasing order by weight w

for each (u, v) taken from the sorted list

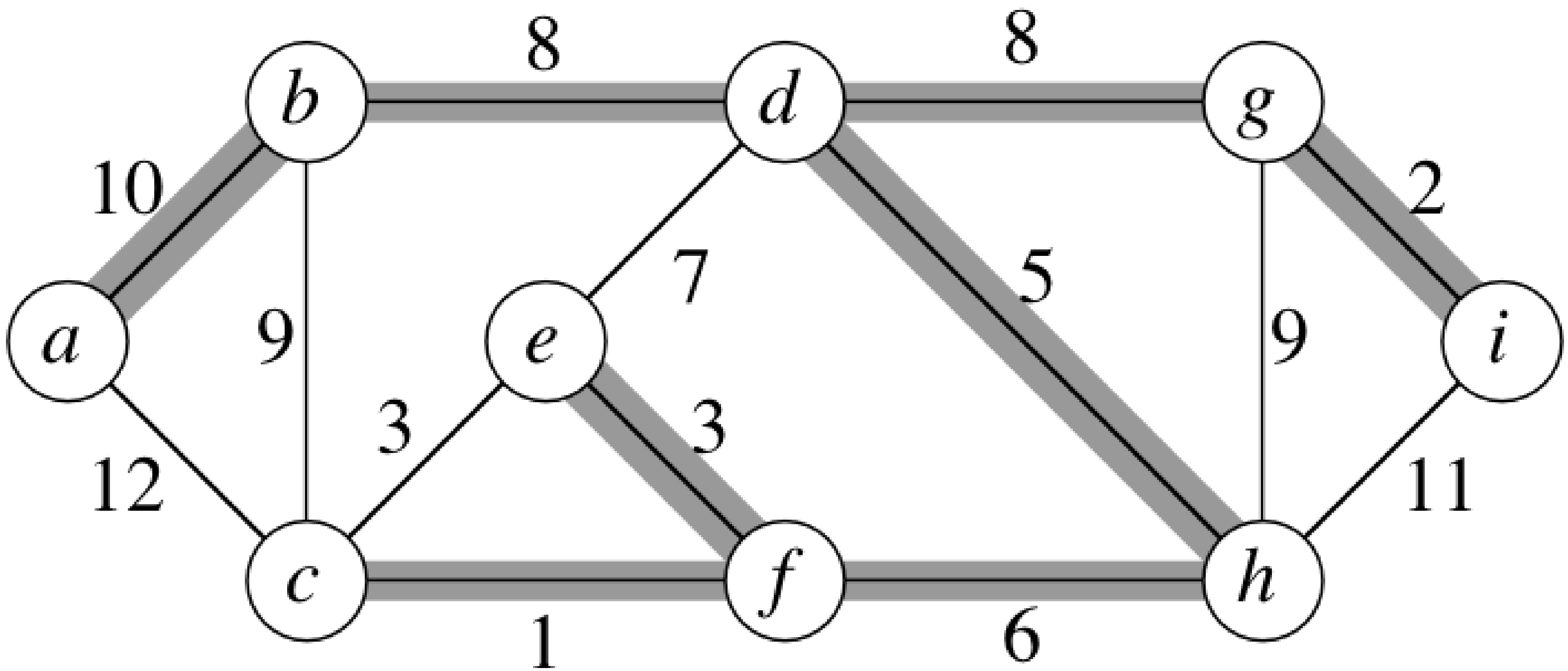
  do if FIND-SET$(u) \neq$ FIND-SET(v)

    then $A \leftarrow A \cup \{(u, v)\}$

      UNION(u, v)

return A

Example

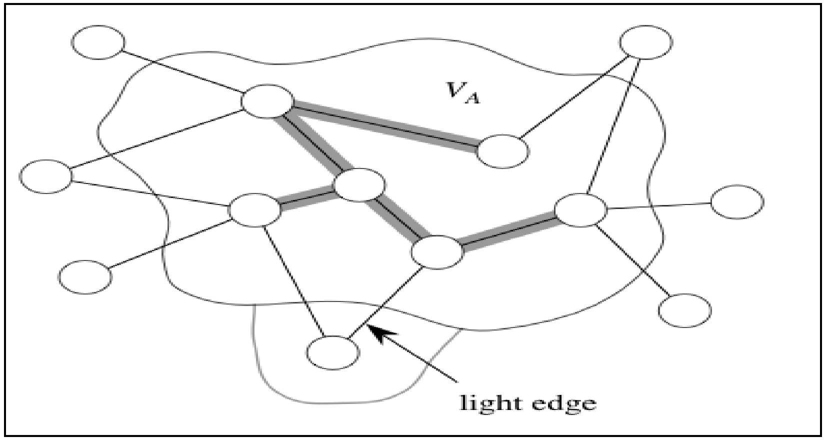


Analysis

- Time complexity:
  - #make-set, find-set and union operations: $O(|V| + |E|)$
    - $O((|V| + |E|) \alpha (|V| + |E|))$
  - Sorting:
    - $O(|E| \log |E|) = O(|E| \log |V|)$
  - Total:
    - $O(|E| \log |V|)$

Prim's Algorithm

- Start with an arbitrary node from V
- Instead of maintaining a forest, grow a MST
  - At any time, maintain a MST for $V' \subseteq V$
- At any moment, find a light edge connecting V' with $(V-V')$ i.e., the edge with smallest weight connecting some vertex in V' with some vertex in $V-V'$!



Prim's Algorithm cont.

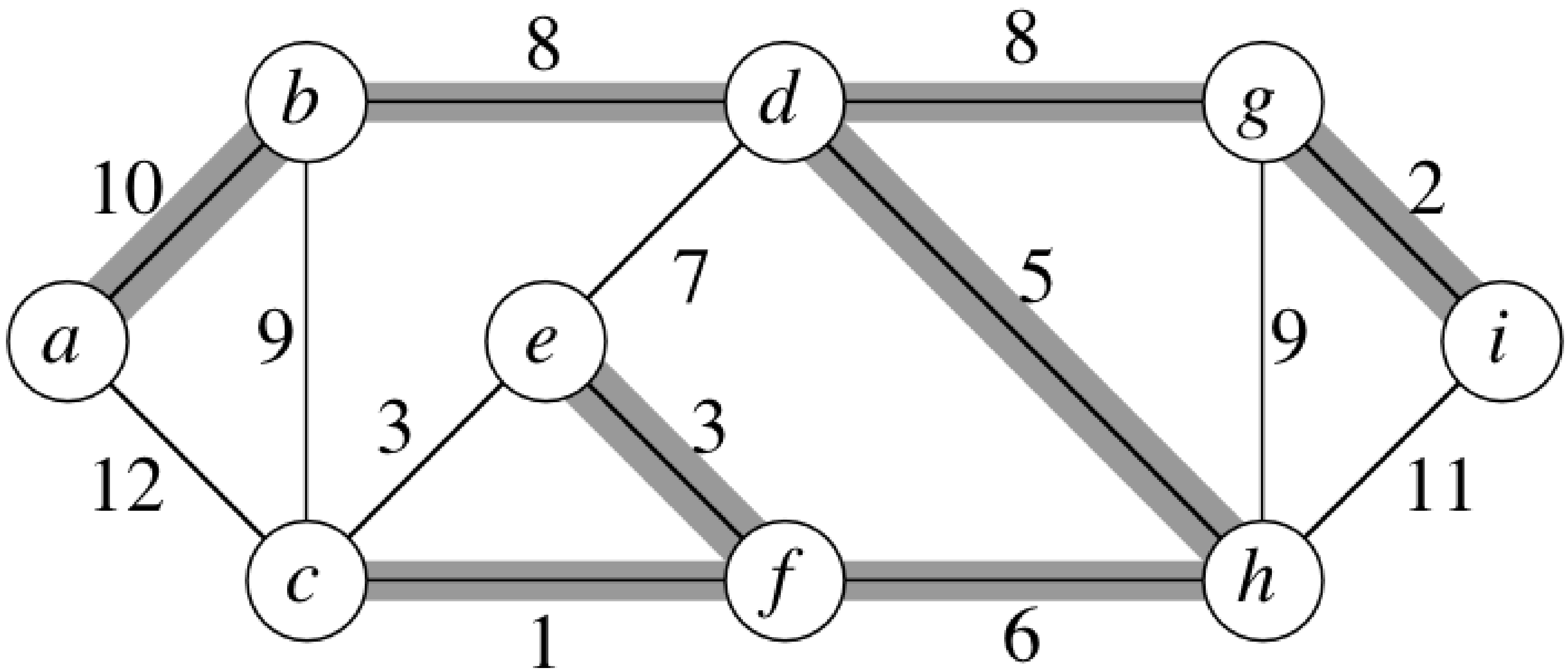
- Again two issues:
  - Maintain the tree already build at any moment
    - Easy: simply a tree rooted at r : the starting node
  - Find the next light edge efficiently
    - For $v \in V - V'$, define $key(v) =$ the min distance between v and some node from V'
    - At any moment, find the node with min key.

Use a priority queue !

Pseudo-code

```
PRIM(V, E, w, r)
Q ← ∅
for each u ∈ V
    do key[u] ← ∞
       π[u] ← NIL
       INSERT(Q, u)
DECREASE-KEY(Q, r, 0)     ▷ key[r] ← 0
while Q ≠ ∅
    do u ← EXTRACT-MIN(Q)
       for each v ∈ Adj[u]
           do if v ∈ Q and w(u, v) < key[v]
                 then π[v] ← u
                      DECREASE-KEY(Q, v, w(u, v))
```

Example



Analysis

- Time complexity
  - # insert:
    - $O(|V|)$
  - # Decrease-Key:
    - $O(|E|)$
  - # Extract-Min
    - $O(|V|)$
- Using heap for priority queue:
  - Each operation is $O (\log |V|)$
- Total time complexity: $O (|E| \log |V|)$

Using Fibonacci heap:
Decrease-Key:
$O(1)$ amortized time
=>
total time complexity
$O(|E| + |V| \log |V|)$

Applications

- Clustering
- Euclidean traveling salesman problem