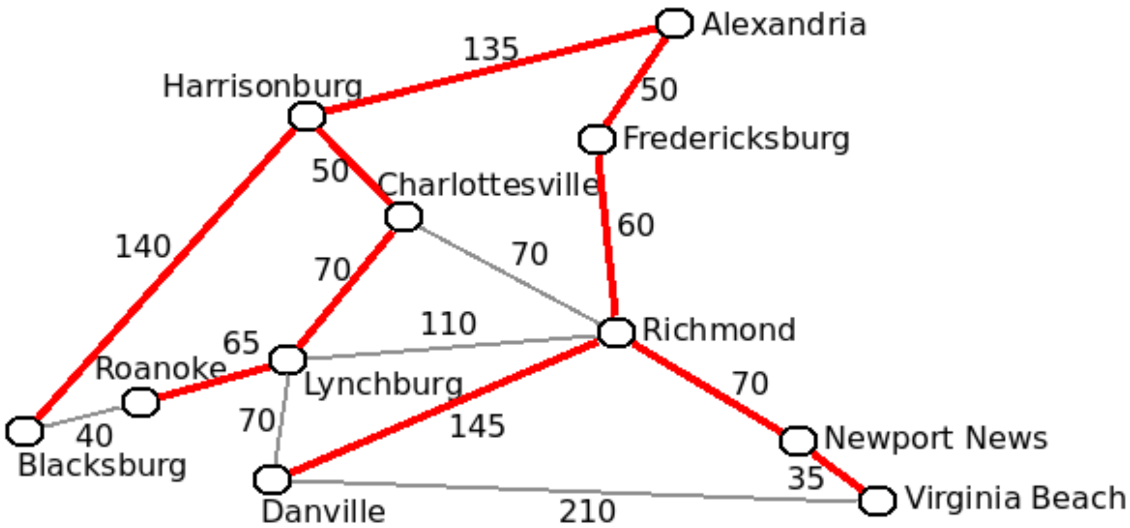
Minimum spanning trees

Computation of a minimal-cost tree
covering a connected undirected graph



Various applications in network design (telephone, electric, roads,…)

Algorithms

Prim's algorithm

suitable for adjacency matrices

$O(n^2)$, $O(|A| \log n)$, can be made $O(|A| + n \cdot \log n)$

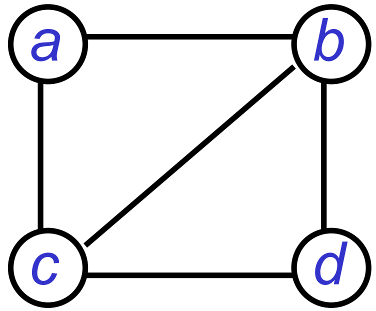
Kruskal's algorithm

suitable for adjacency lists and sparse graphs ($|A| \ll n^2$)

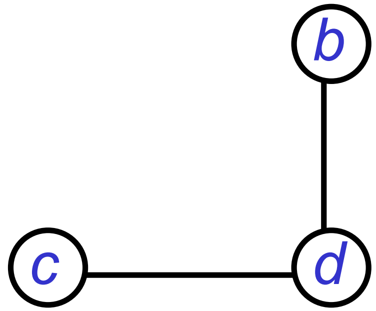
$O(|A| \log |A|)$, $O(|A| \log n)$

Undirected graphs

$G=(S, A)$, $|S| = n$, no self-loops

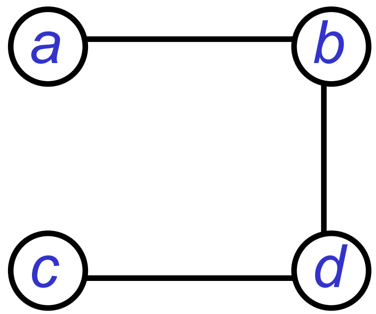


Subgraph
$G' \subseteq G$: G' graph (S', A')
with $S' \subseteq S$ and $A' \subseteq A$
(not to be confused with *induced subgraphs*)



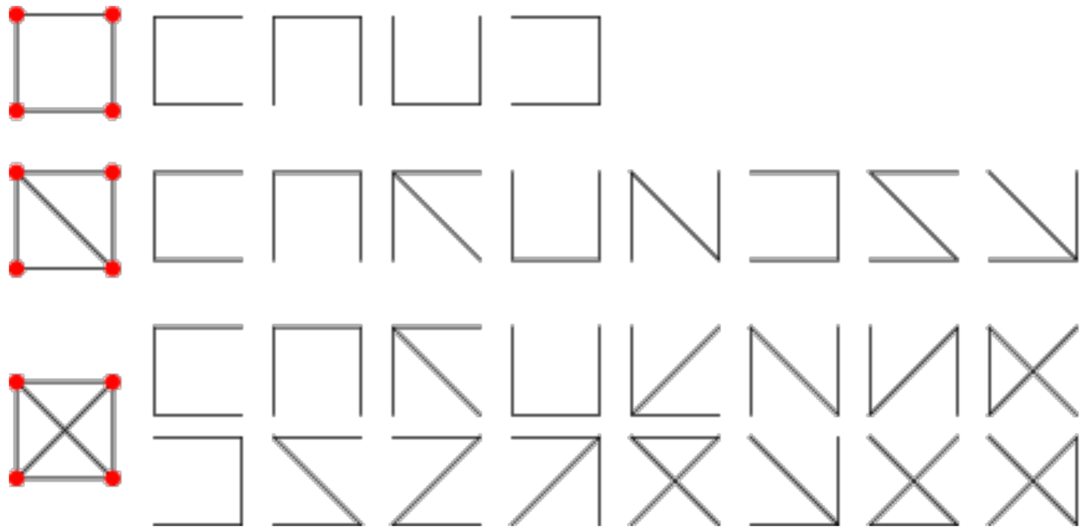
Tree
connected acyclic graph
(note the difference with *rooted trees*)

Forest
set of disjoint trees



Spanning tree
a subgraph (S', A') with $S' = S$ which is a tree

Many different spanning trees



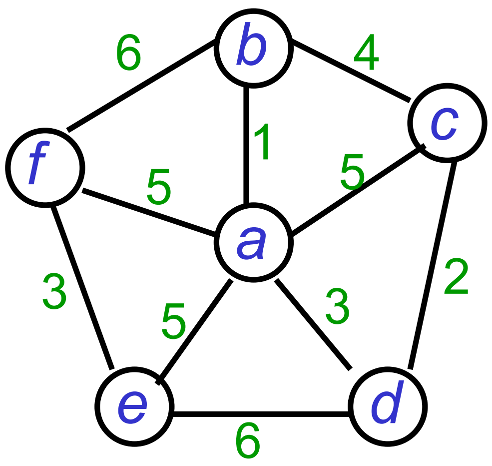
n^{n-2} spanning trees in complete graph K_n (Cayley formula)

Minimal spanning tree problem

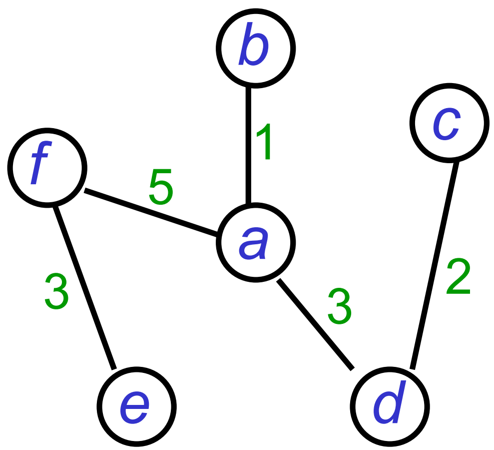
Weighted graph $G = (S, A, v)$ with weights $v : A \rightarrow \mathbf{R}$
undirected and connected

Cost of a subgraph $G' = (S', A') : \Sigma\ (v(p,q) \mid (p,q) \in A')$

Problem: compute a spanning tree of G with minimal cost



cost = 14

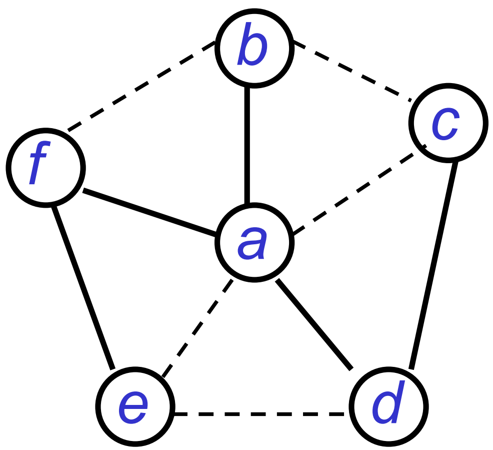


Spanning tree

$G=(S, A)$ undirected connected graph, $|S| = n$
T spanning tree, B set of edges of T

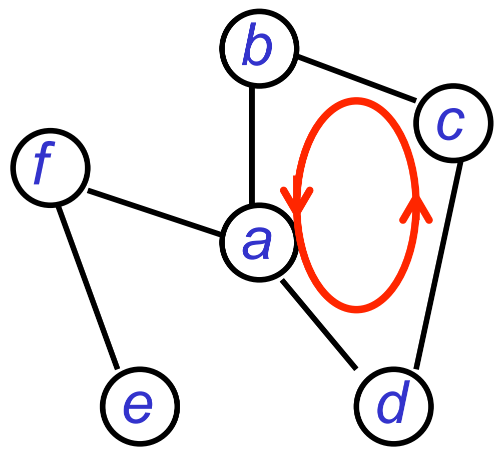
Properties

- T has n-1 edges : $|B| = n-1$
- if $\{p, q\} \in A-B$ then $H = (S, B + \{p,q\})$ has a cycle



$|S| = 6$

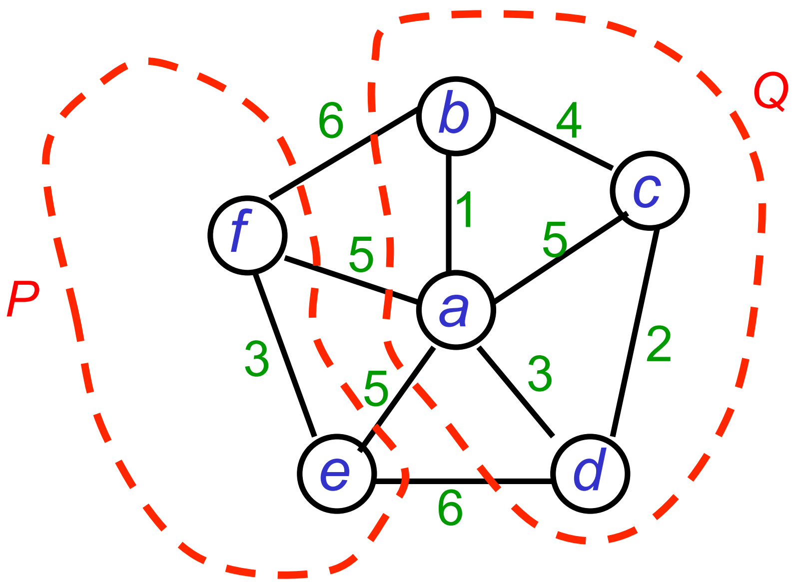
$|B| = 5$

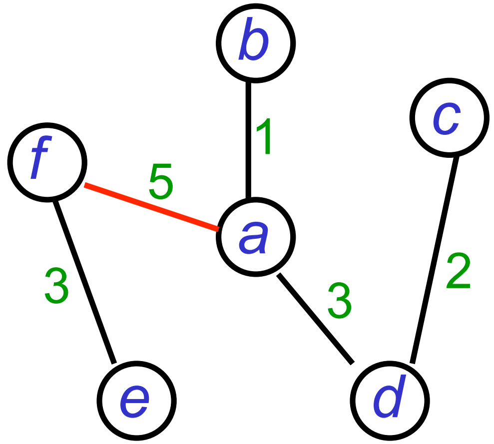


Cut

Let $G = (S, A, v)$ and $\{P, Q\}$ be a partition of S

Theorem If $\{p, q\}$ is an minimal cost crossing edge between P and Q, then there exists a minimal spanning tree containing $\{p, q\}$





Proof

Let $\{p, q\}$ be a minimal cost crossing edge between P and Q

$p \in P \quad q \in Q$

Let $T = (S, B)$ be a minimal spanning tree of G

if $\{p, q\} \in B$ we are done

otherwise

$H = (S, B + \{p, q\})$ contains a cycle

this cycle contains $\{u, v\} \in B,\ u \in P,\ v \in Q$

Let $T' = (S, B - \{u, v\} + \{p, q\})$

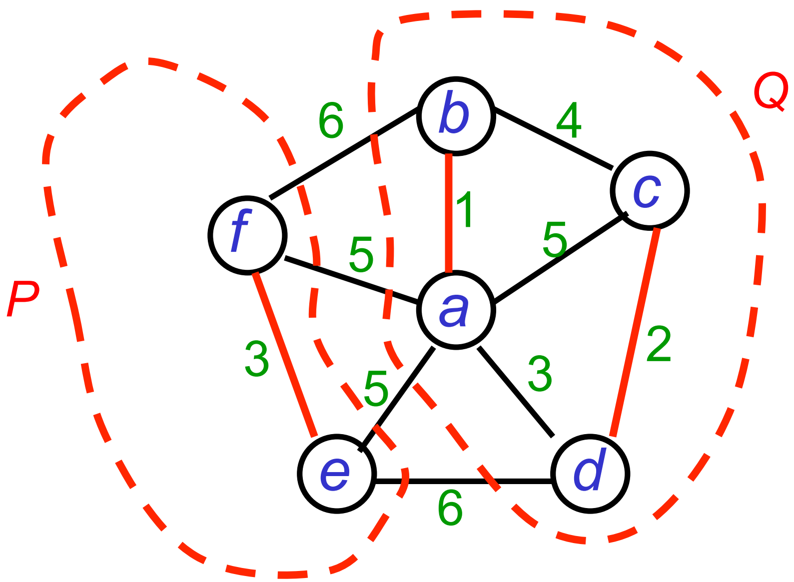
T' is a spanning tree, and

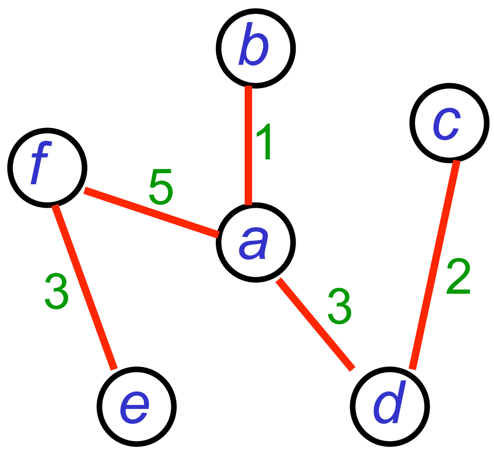
$\text{cost}(T') \leq \text{cost}(T) \Rightarrow \text{cost}(T') = \text{cost}(T)$

and $\{p, q\}$ is contained in T'

Cut

Theorem (generalization) Let F be a forest contained in a minimal spanning tree. Let $\{P, Q\}$ be a partition of S that respect F (has no crossing edge between P et Q). If $\{p, q\}$ is a minimal cost edge between P and Q, then there exists a minimal spanning tree of G that contains $F \cup \{\{p, q\}\}$

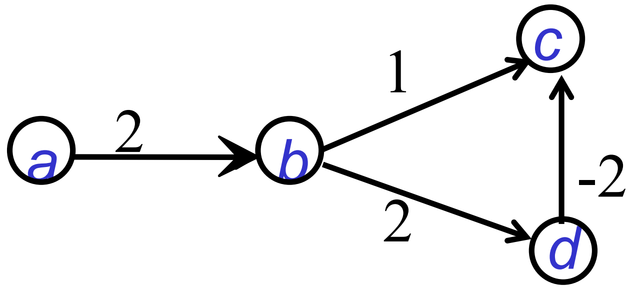




Greedy algorithms

Sequential processing with locally optimal steps

Does not produce a global optimum in general



Minimal cost path from *a* to *c* ?

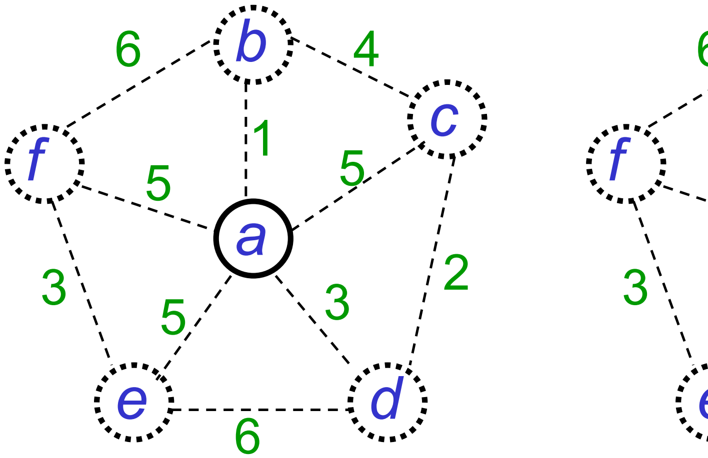
Optimality for

- change-making problem (for canonical coin systems)
- Huffman codes
- Dijkstra's algorithm
- Prim's and Kruskal's algorithms for minimal ST

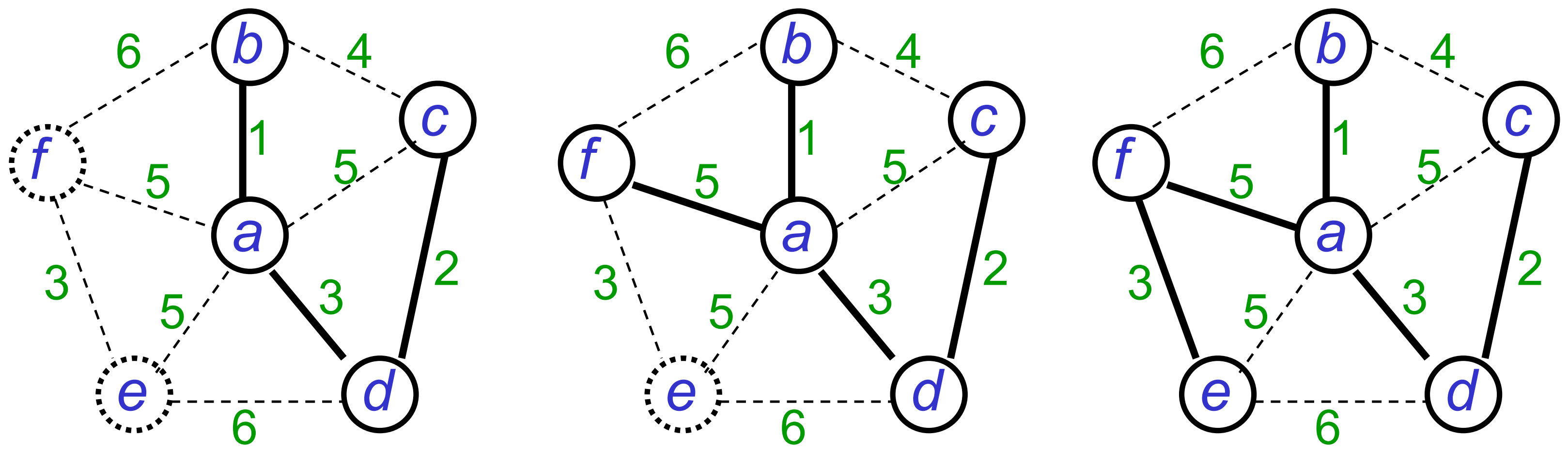
Prim’s algorithm (1957)

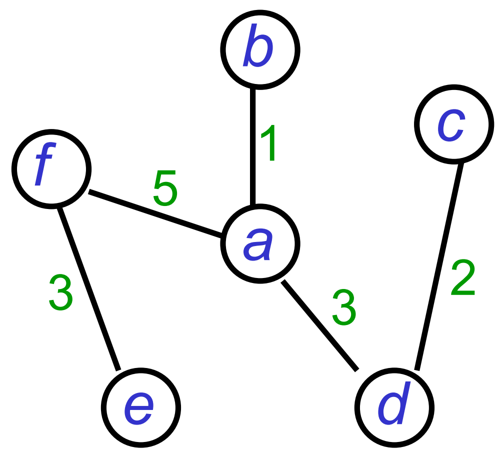
Compute a minimal spanning tree:
”grow“ a tree until all graph nodes are covered

Example:



Example (cont)





minimal ST
cost = 14

Prims algorithm

```
MST-PRIM( graph ({1, 2, ..., n}, A, v) ) {
        T ← {1} ;
        B ← ∅ ;
        while card T < n do {
                {p, q} ← minimal cost edge
                            such that p ∈ T and q ∉ T ;
                T ← T + {q} ;
                B ← B + {p, q} ;
        }
        return (T, B) ;
}
```

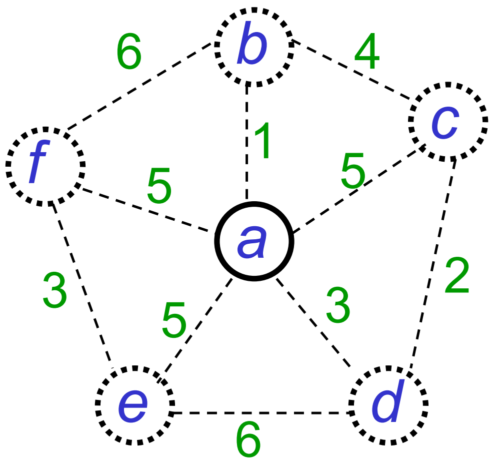
Implementation

Arrays **`dist-to-tree`** and **`cost`** to find the edge $\{p, q\}$

$q \notin T$ **`dist-to-tree`**$[q] = p \in T$
iff $v(p, q) = \min \{ v(p', q) \mid p' \in T \}$

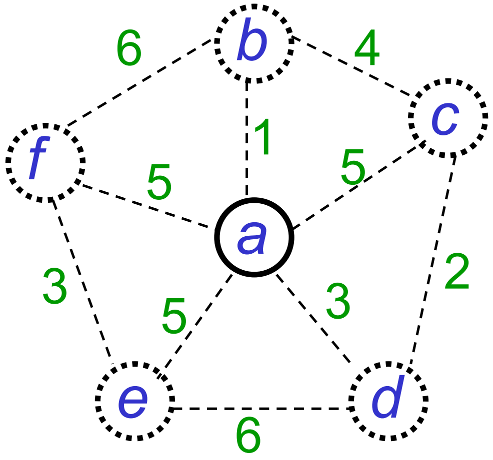
$q \notin T$ **`cost`**$[q] = v($**`dist-to-tree`**$[q], q)$

$q \in T$ **`cost`**$[q] = \perp$



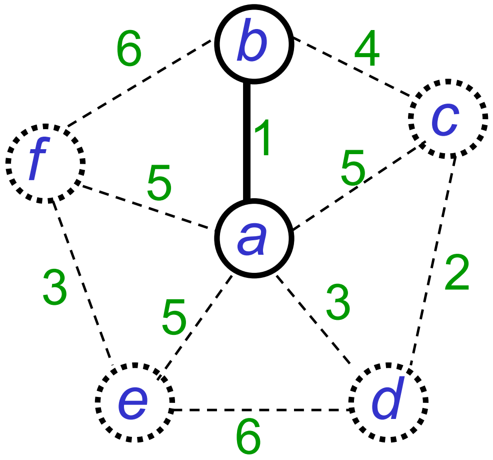
	a	b	c	d	e	f
dist-to-tree		a	a	a	a	a
cost	$\perp$	1	5	3	5	5

One step



	a	b	c	d	e	f
`dist-to-tree`		a	a	a	a	a
`cost`	⊥	1	5	3	5	5

adding *b* and {*a*, *b*}



	a	b	c	d	e	f
`dist-to-tree`		a	b	a	a	a
`cost`	⊥	⊥	4	3	5	5

Updating `cost`: *O*(degree(*b*))

Total running time: O(n^2)

Better implementation: priority queue

With a binary heap:

extracting the minimal cost edge: $O(\log n)$

updating the costs: $O(|A|)$ updates overall, each in time $O(\log n)$

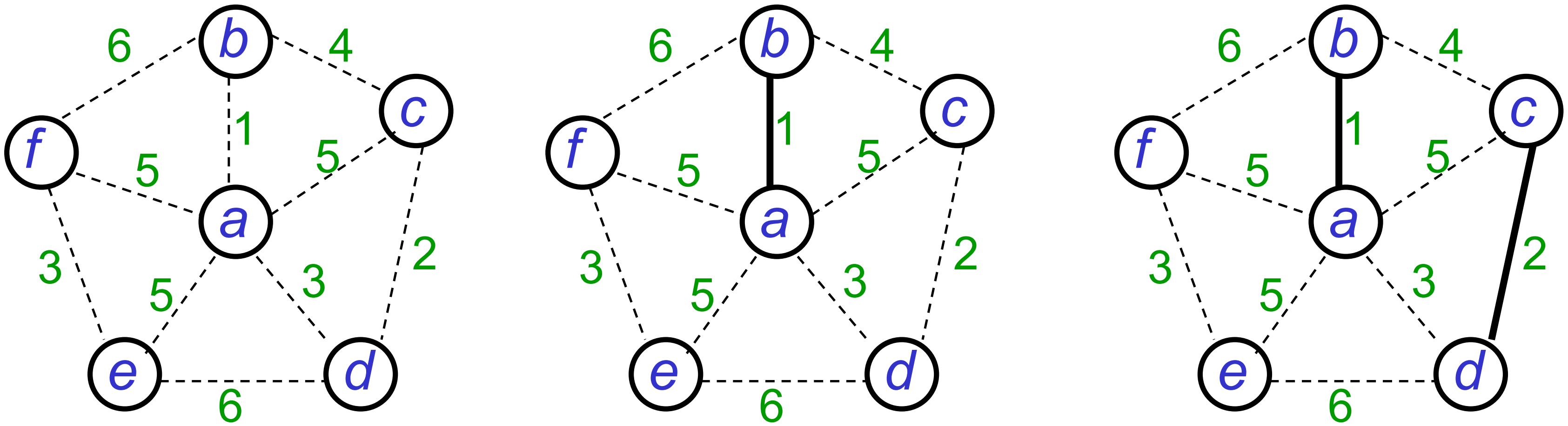
In total: $O(n \cdot \log n + |A| \cdot \log n) = O(|A| \cdot \log n)$

This can be improved to $O(|A| + n \cdot \log n)$ with Fibonacci heaps

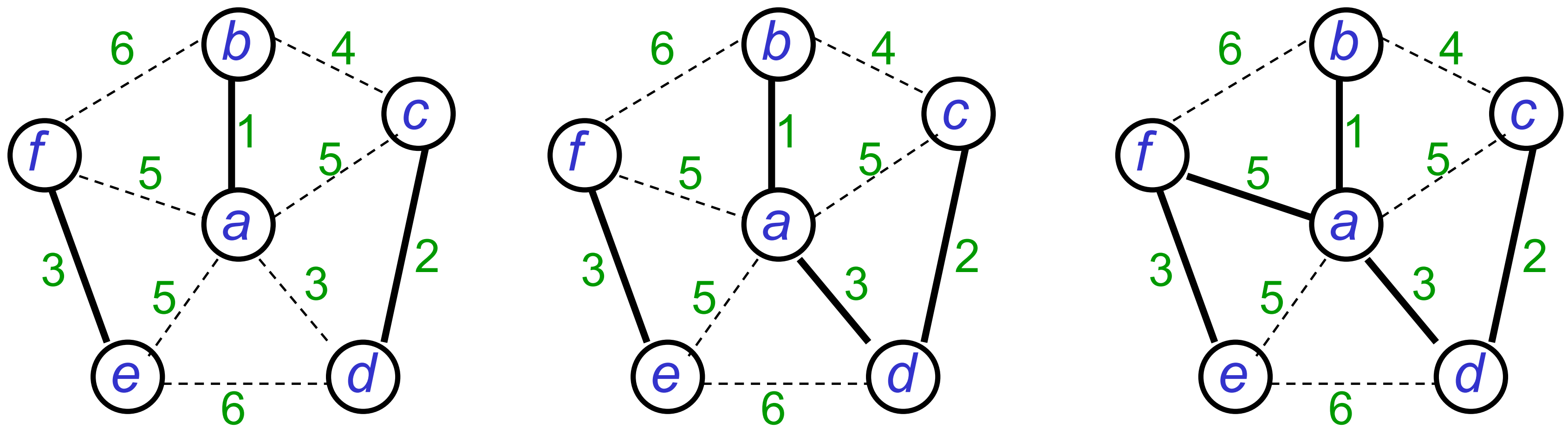
Kruskal's algorithm (1956)

Computation step:
connect two disjoint subtrees by a minimal cost edge

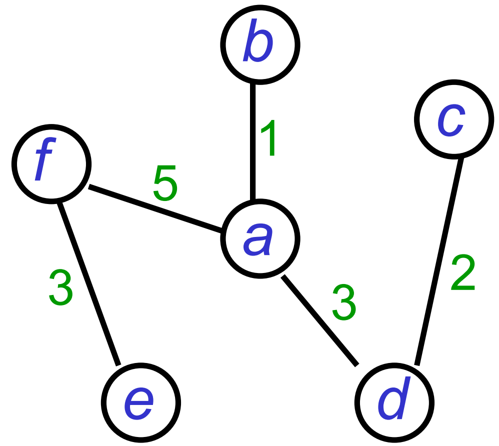
Example:



Example (cont)



minimal ST
cost = 14



Kruskal's algorithm

MST-KRUSKAL(graph ({1, 2, ..., n}, A, v)) **{**

    Partition ← { {1}, {2}, ..., {n} } ;

    B ← ∅ ;

    while |*Partition*| > 1 **do** **{**

        {p, q} ← minimal cost edge such that

            CLASS(p) ≠ **CLASS**(q) ;

        B ← B + {p, q} ;

        merge **CLASS**(p) and **CLASS**(q) in *Partition*;

    }

    return (({1, 2, ..., n}, B) ;

}

Implementation:

Pre-sort all the edges A by increasing costs, in time O(|A| · log |A|) = O(|A| · log n) ;

2·|A| operations **CLASS** and |A| operations **UNION**: O(|A|·α(|A|)), cf next slides

Resulting time bound: O(|A| · log n)

UNION / FIND

Maintaining partitions of {1, 2, …, *n*} under **operations**

- **`FIND`**(*p*) : compute a representative (identifier) of the class of *p*
- **`UNION`**(*p*,*q*) : union of disjoint classes of *p* and of *q*

Example: n = 7

`UNION`(1, 2);	{1, 2} {3} {4} {5} {6} {7}
`UNION`(5, 6);	{1, 2} {3} {4} {5, 6} {7}
`UNION`(3, 4);	{1, 2} {3, 4} {5, 6} {7}
`UNION`(1, 4);	{1, 2, 3, 4} {5, 6} {7}
`FIND`(2)=**`FIND`**(3)?	YES

Example of application : compute connected components of an undirected graph

How **`UNION`** and **`FIND`** are implemented?

Array implementation

	1	2	3	4	5	6	7
CLASS	1	1	3	3	5	5	7

represents {1,2} {3,4} {5,6} {7}

```
UNION(p,q)
{       x ← FIND(p) ;   y ← FIND(q) ;
        for k ← 1 to n do
                if CLASS [k] = y then
                        CLASS [k] ← x ;
}
```

Time
FIND : $O(1)$
UNION : $O(n)$

UNION(1, 4)

	1	2	3	4	5	6	7
CLASS	1	1	1	1	5	5	7

represents {1,2,3,4} {5,6} {7}

Linked list implementation

`FIND`(*p*): return the head of the list of *p*
`UNION`(*p,q*): concatenate the list of *p* with the list of *q* (or vice versa)

1. **Simple linked lists**:
   `FIND`(*p*): O*(n)*
   `UNION`(*p,q*): *O(1)*

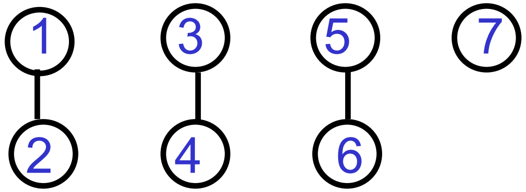
2. **Linked lists with elements pointing to the head**
   `FIND`(*p*): O(1)
   `UNION`(*p,q*): O(n)

3. **Linked lists with elements pointing to the head and length counter** (weighted-union heuristic)
   a sequence of *m* operations **`UNION/FIND`** on a set of *n* elements takes time $O(m+n \cdot \log(n))$

Tree implementation

partition	{1,2}	{3,4}	{5,6}	{7}
FIND(p),**SIZE**	1,2	3,2	5,2	7,1

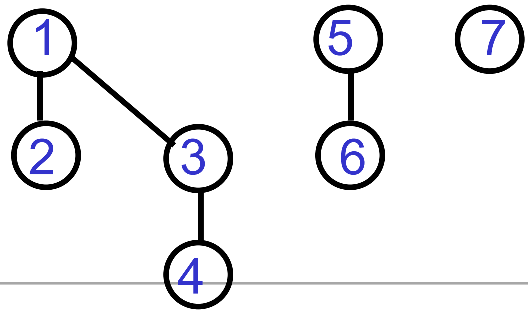
forest




```
FIND(p) {
        k ← p ;
        while parent(k) is defined do k ← parent(k) ;
        return (k) ;
}
```

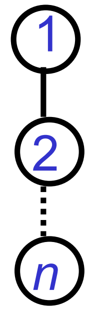
partition	{1,2,3,4}	{5,6}	{7}
FIND(p),**SIZE**	1,4	5,2	7,1

forest



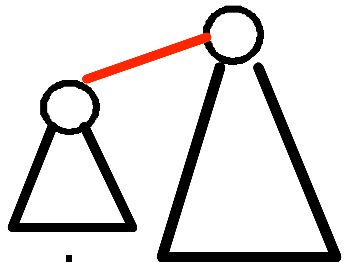
Time
FIND : O(n)
UNION : O(1)

Joining trees



Avoid linear-shape trees to reduce the computation time of **`FIND`**(p)

Strategy union-by-rank: attach the smaller tree to the root of the larger one



smaller class

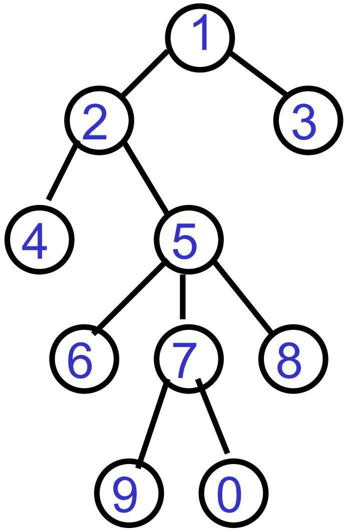
larger class

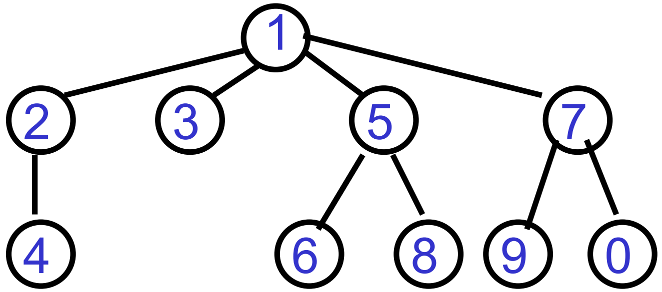
Time
`FIND` : O(log$n$)
`UNION` : O(1)

Proof
depth(i) increases by 1 at **`UNION`**(P,Q) when $|P| \leq |Q|$ and $i \in P$, *i.e.*, when the class size at least doubles.
This cannot happen more than $\lfloor \log_2 n \rfloor$ times.

Path compression

Idea : flatten the tree by attaching the nodes traversed by **`FIND`**(*p*) directly to the root





after **`FIND(7)`**

Time of *m* calls to **`UNION and FIND`**: $O(m \cdot \alpha(n))$

where $\alpha(n)$ is the inverse of the Ackermann function.
$\alpha(n) \leq 4$ for all practical purposes