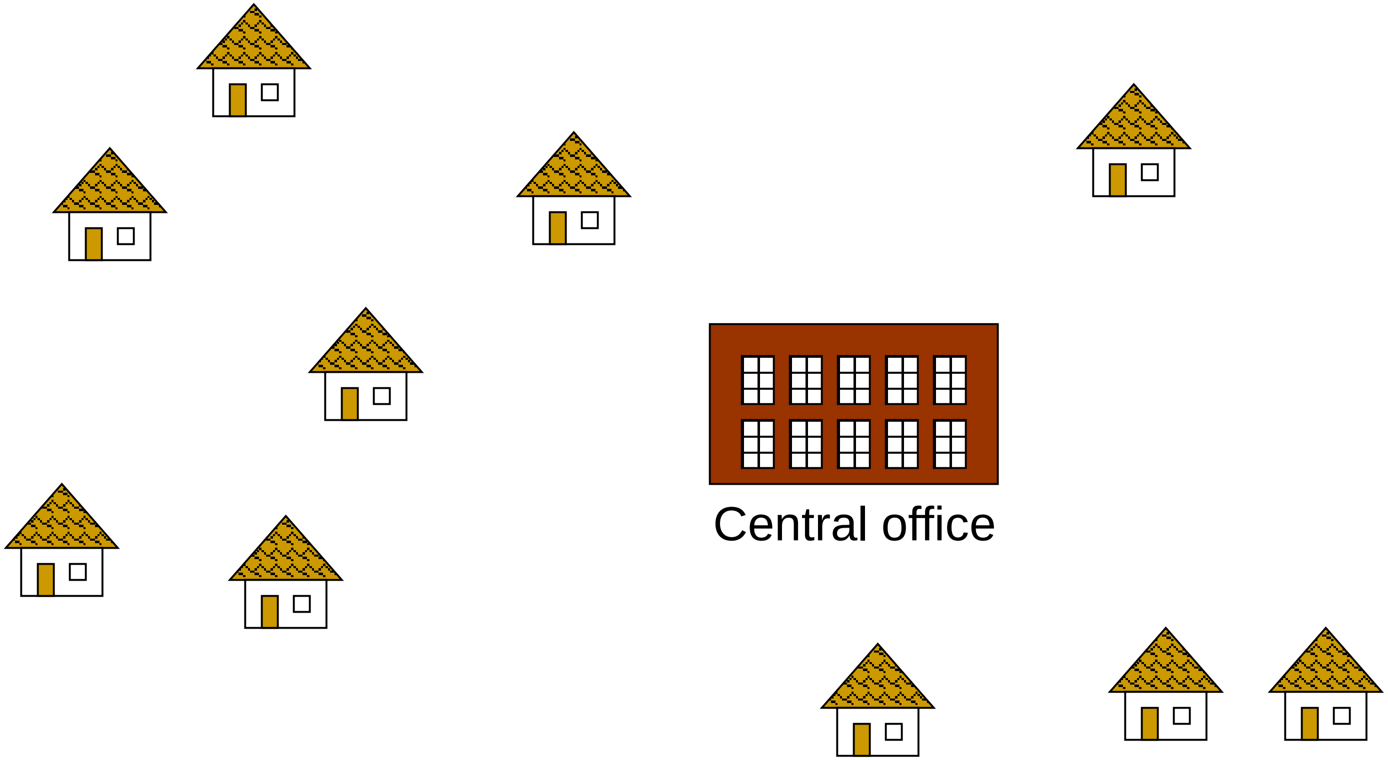
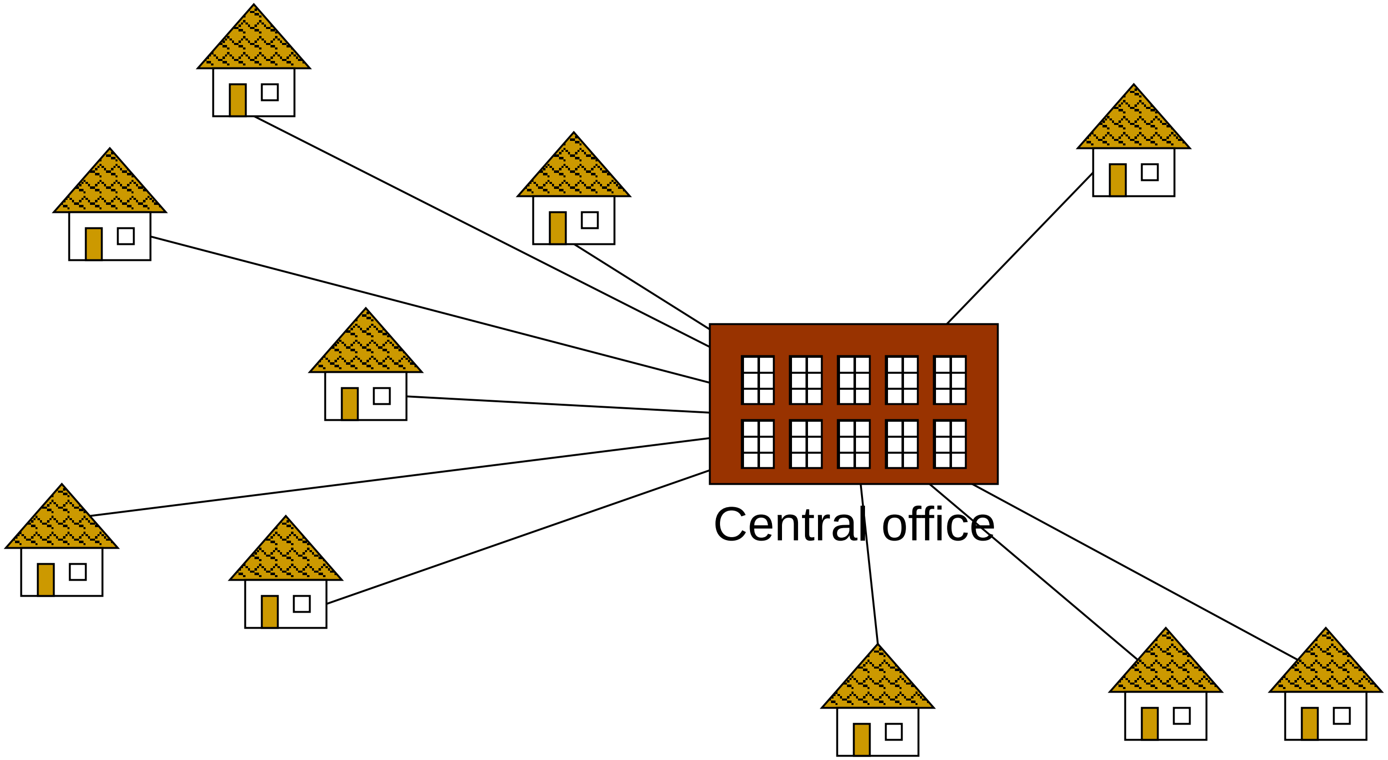
Minimum Spanning Trees

Problem: Laying Telephone Wire

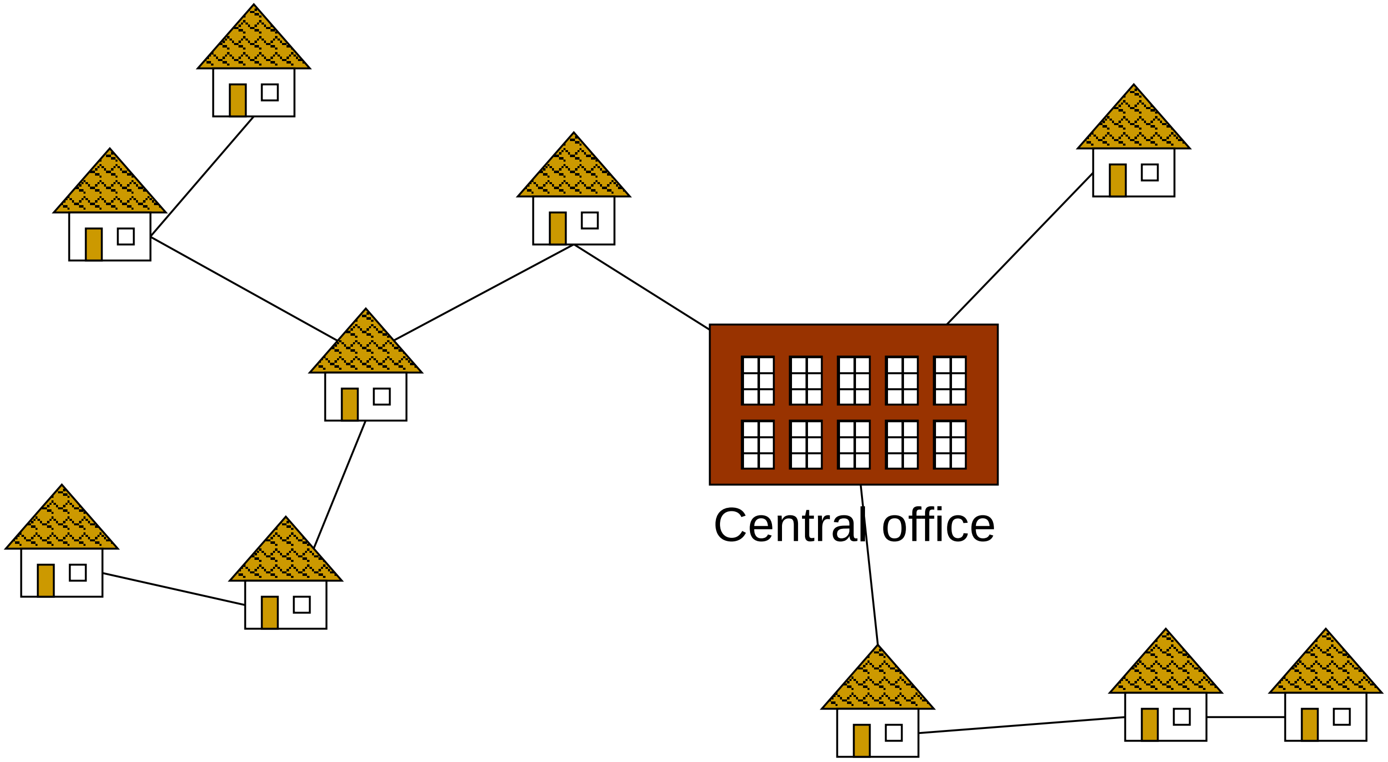


Wiring: Naïve Approach



Expensive!

Wiring: Better Approach



Minimize the total length of wire connecting the customers

Minimum Spanning Tree (MST)

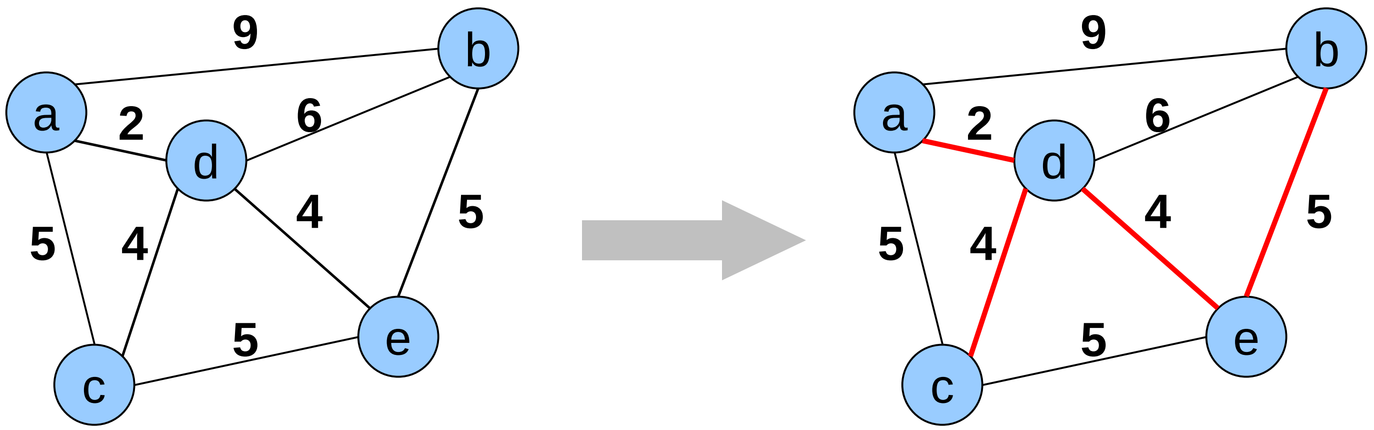
A **minimum spanning tree** is a subgraph of an undirected weighted graph ***G***, such that

- it is a tree (i.e., it is acyclic)
- it covers all the vertices ***V***
  - contains ***|V| - 1*** edges
- the total cost associated with tree edges is the minimum among all possible spanning trees
- not necessarily unique

Applications of MST

- Any time you want to visit all vertices in a graph at minimum cost (e.g., wire routing on printed circuit boards, sewer pipe layout, road planning…)
- Internet content distribution
  - $$$, also a hot research topic
  - Idea: publisher produces web pages, content distribution network replicates web pages to many locations so consumers can access at higher speed
  - MST may not be good enough!
    - content distribution on minimum cost tree may take a long time!

How Can We Generate a MST?



Prim's Algorithm

- Let $V = \{1,2,..,n\}$ and U be the set of vertices that makes the MST and T be the MST
- Initially : $U = \{1\}$ and $T = \phi$
- while $(U \neq V)$

  let (u,v) be the lowest cost edge such that

  $u \in U$ and $v \in V-U$

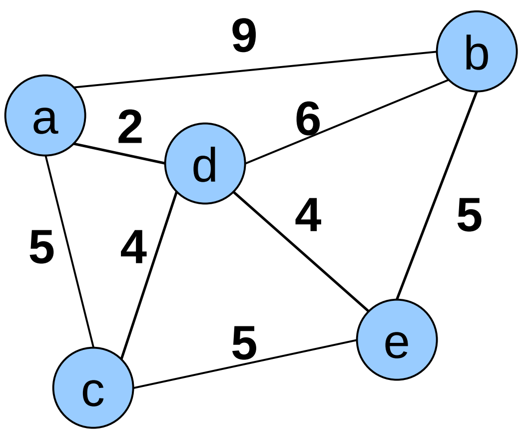
  $T = T \cup \{(u,v)\}$

  $U = U \cup \{v\}$

Prim’s Algorithm implementation

Initialization

a. Pick a vertex ***r*** to be the root
b. Set ***D***(***r***) ***= 0, parent***(***r***) ***= null***
c. For all vertices ***v*** $\in$ ***V***, ***v*** $\neq$ ***r***, set ***D***(***v***) ***=*** ∞
d. Insert all vertices into priority queue ***P***, using distances as the keys



e	a	b	c	d
0	∞	∞	∞	∞

Vertex	Parent
e	-

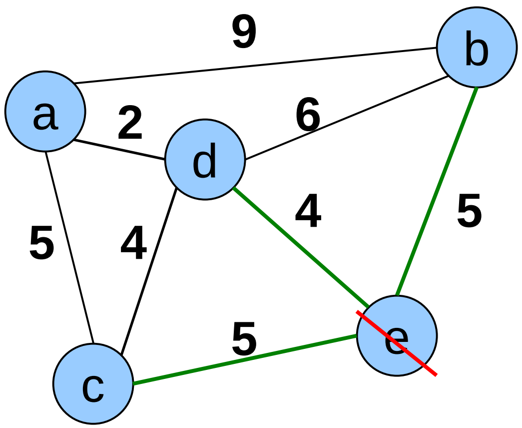
Prim’s Algorithm

While *P* is not empty:

1. Select the next vertex ***u*** to add to the tree
   u = P.deleteMin()

2. Update the weight of each vertex ***w*** adjacent to ***u*** which is not in the tree (i.e., $w \in P$)
   If ***weight***(***u,w***) **<** ***D***(***w***),
   a. ***parent***(***w***) **=** ***u***
   b. ***D***(***w***) **=** ***weight***(***u,w***)
   c. Update the priority queue to reflect new distance for ***w***

Prim’s algorithm



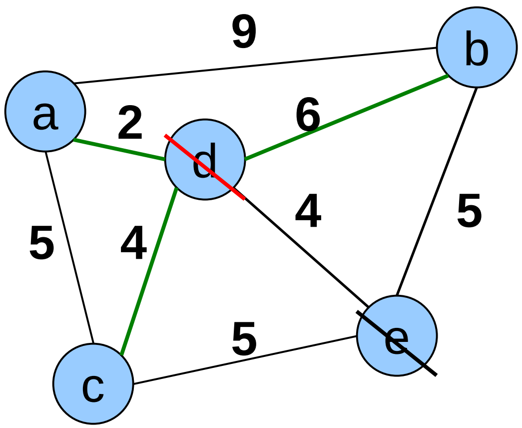
d	b	c	a
4	**5**	**5**	∞

<u>Vertex Parent</u>

Vertex	Parent
e	-
b	e
c	e
d	e

The MST initially consists of the vertex ***e,*** and we update the distances and parent for its adjacent vertices

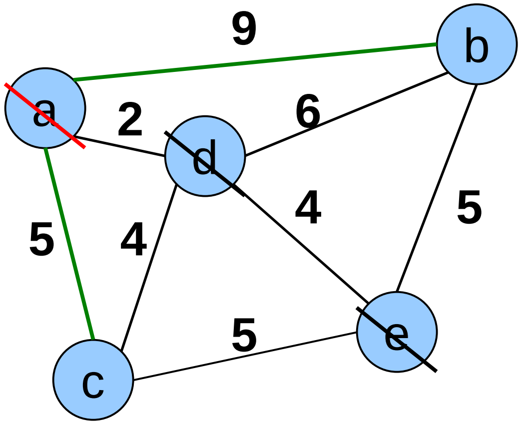
Prim's algorithm



a	c	b
2	4	5

Vertex	Parent
e	-
b	e
c	d
d	e
a	d

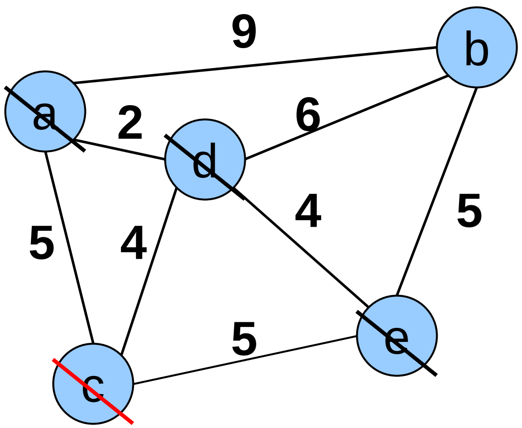
Prim's algorithm



c	b
4	**5**

Vertex	Parent
e	-
b	e
c	d
d	e
a	d

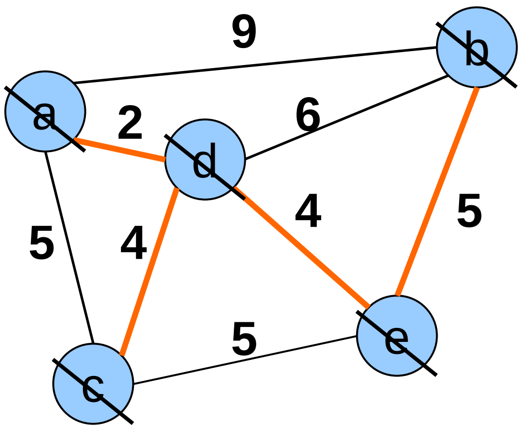
Prim’s algorithm



b
5

Vertex	Parent
e	-
b	e
c	d
d	e
a	d

Prim's algorithm



Vertex	Parent
e	-
b	e
c	d
d	e
a	d

The final minimum spanning tree

Prim's Algorithm Invariant

- At each step, we add the edge (u,v) s.t. the weight of (u,v) is **minimum** among all edges where u is in the tree and v is not in the tree
- Each step maintains a minimum spanning tree of the vertices that have been included thus far
- When all vertices have been included, we have a MST for the graph!

Running time of Prim's algorithm

Initialization of priority queue (array): $O(|V|)$

Update loop: $|V|$ calls

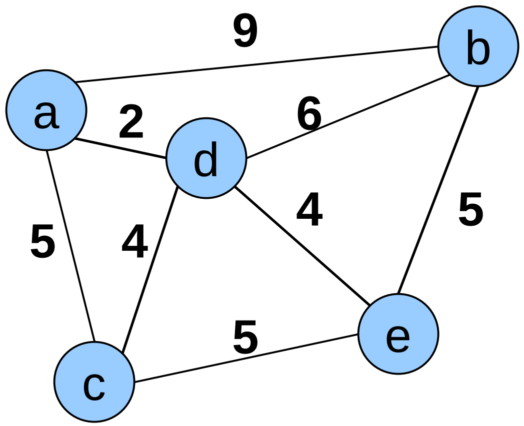
- Choosing vertex with minimum cost edge: $O(|V|)$
- Updating distance values of unconnected vertices: each edge is considered only **once** during entire execution, for a **total** of $O(|E|)$ updates

Overall cost:

$\boxed{O(|E| + |V|^2)}$

Another Approach – Kruskal's

- Create a forest of trees from the vertices
- Repeatedly merge trees by adding "**safe edges**" until only one tree remains
- A "safe edge" is an edge of minimum weight which does not create a cycle



forest: {a}, {b}, {c}, {d}, {e}

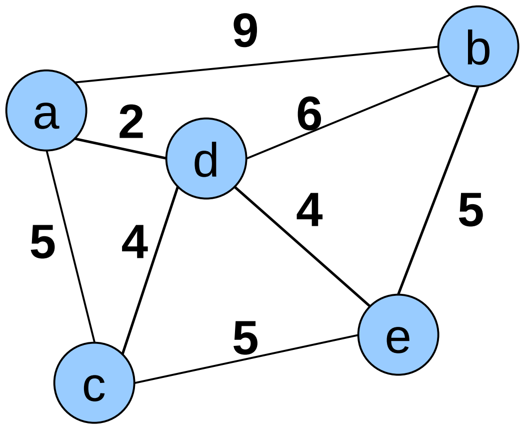
Kruskal's algorithm

Initialization

a. Create a set for each vertex $\boldsymbol{v} \in \boldsymbol{V}$

b. Initialize the set of “safe edges” $\boldsymbol{A}$ comprising the MST to the empty set

c. Sort edges by increasing weight



{a}, {b}, {c}, {d}, {e}

$\boldsymbol{A} = \varnothing$

$\boldsymbol{E}$ = {(a,d), (c,d), (d,e), (a,c), (b,e), (c,e), (b,d), (a,b)}

Kruskal's algorithm

For each edge (u,v) $\in$ E **in increasing order while more than one set remains:**

If u and v, belong to different sets

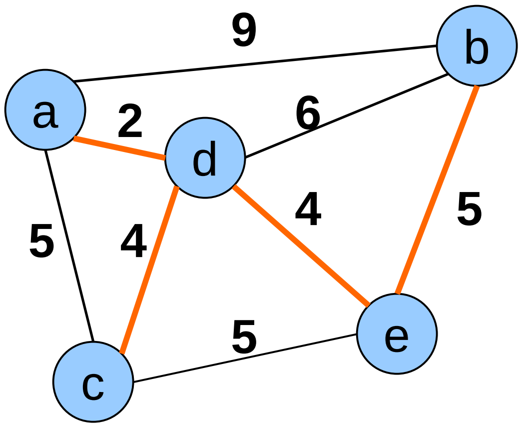
a. $A = A \cup \{(u,v)\}$

b. merge the sets containing u and v

Return A

- Use Union-Find algorithm to efficiently determine if u and v belong to different sets

Kruskal’s algorithm



E = {~~(a,d)~~, ~~(c,d)~~, ~~(d,e)~~, ~~(a,c)~~,
~~(b,e)~~, (c,e), (b,d), (a,b)}

<u>Forest</u>	<u>***A***</u>
{a}, {b}, {c}, {d}, {e}	∅
{a,d}, {b}, {c}, {e}	{(a,d)}
{a,d,c}, {b}, {e}	{(a,d), (c,d)}
{a,d,c,e}, {b}	{(a,d), (c,d), (d,e)}
{a,d,c,e,b}	{(a,d), (c,d), (d,e), (b,e)}

Kruskal’s Algorithm Invariant

- After each iteration, every tree in the forest is a MST of the vertices it connects
- Algorithm terminates when all vertices are connected into one tree

Greedy Approach

- Like Dijkstra’s algorithm, both Prim’s and Kruskal’s algorithms are **greedy algorithms**

- The greedy approach works for the MST problem; however, **it does not work for many other problems**!