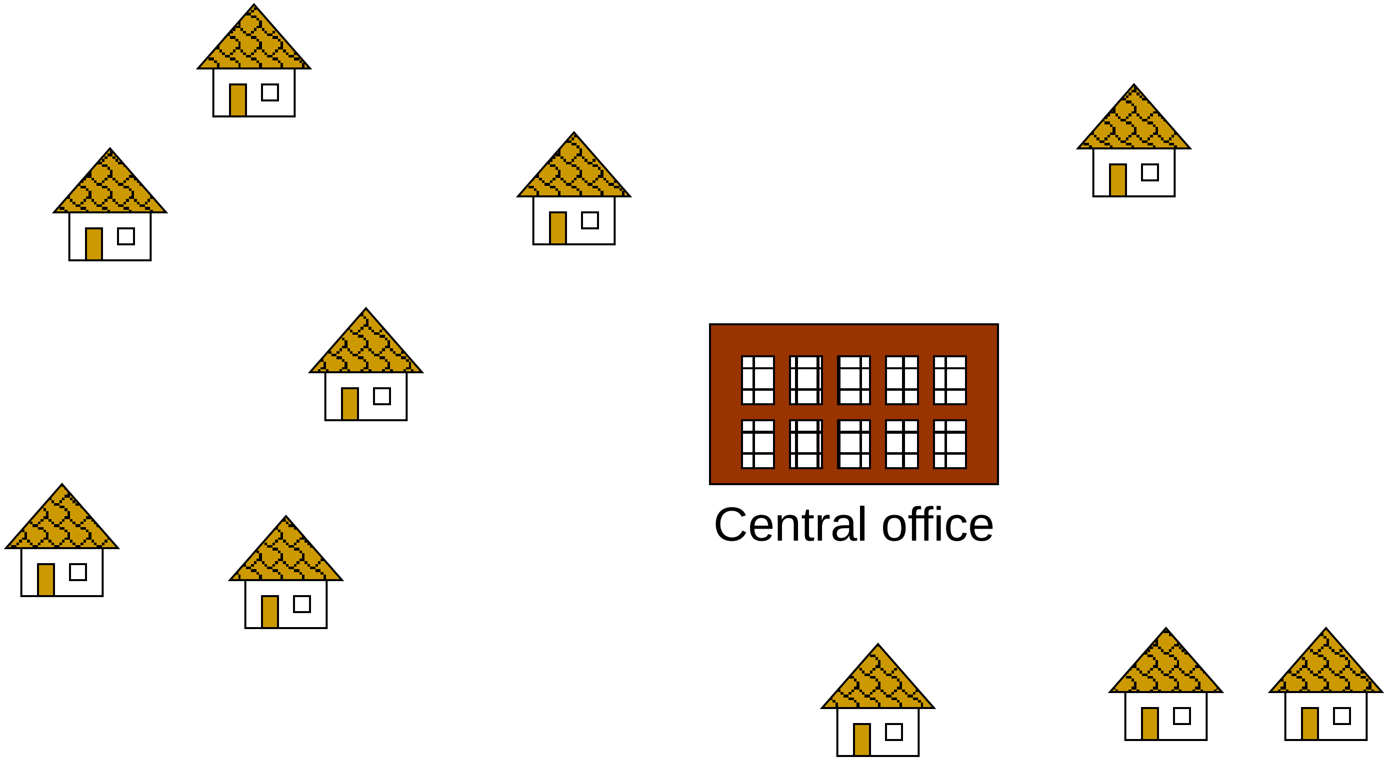
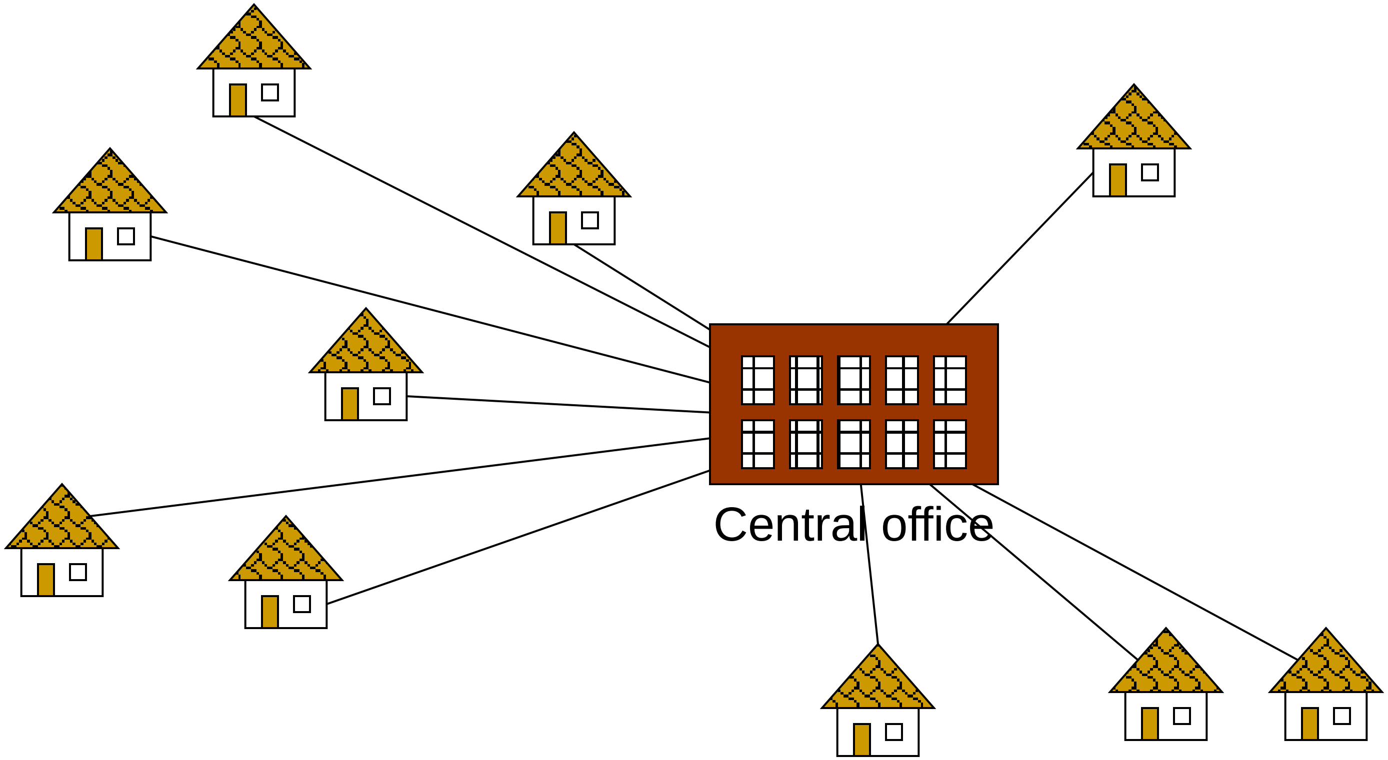
Minimum Spanning Trees

It is a subgraph of Graph such that it covers all the vetex and must not contain any cycle

Problem: Laying Telephone Wire

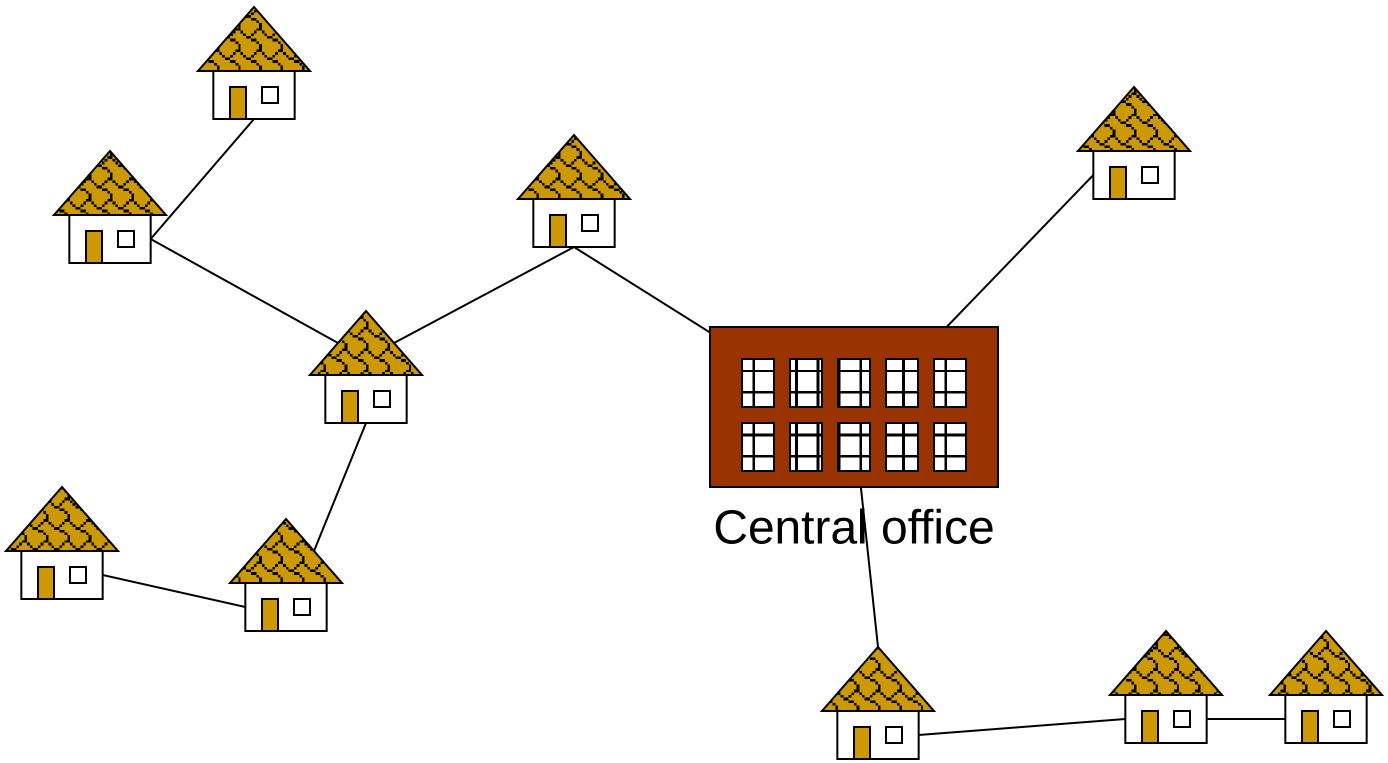


Wiring: Naïve Approach



Expensive!

Wiring: Better Approach



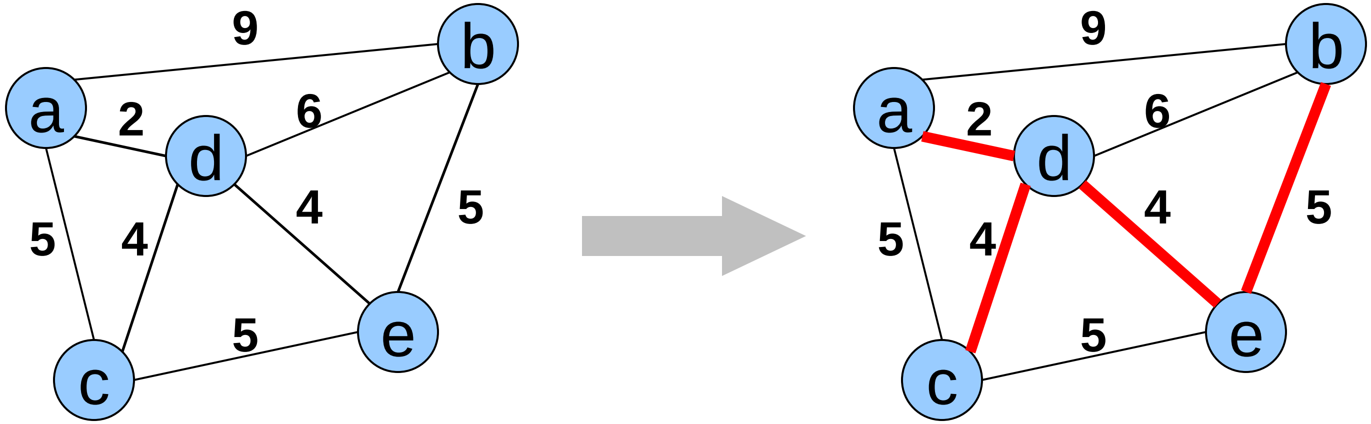
Minimize the total length of wire connecting the customers

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

How Can We Generate a MST?



Prim's Algorithm

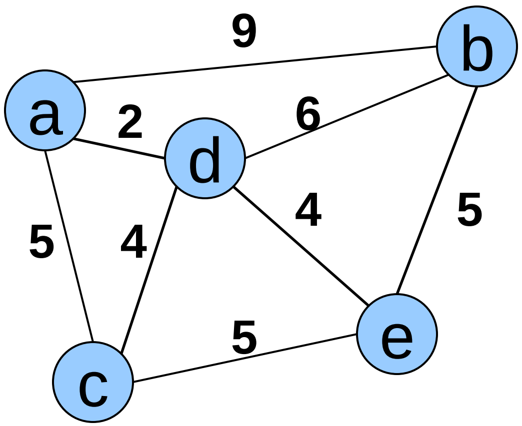
Initialization

a. Pick a vertex $\boldsymbol{r}$ to be the root

b. Set $\boldsymbol{D(r) = 0}$, $\boldsymbol{parent(r) = null}$

c. For all vertices $\boldsymbol{v \in V}$, $\boldsymbol{v \neq r}$, set $\boldsymbol{D(v) = \infty}$

d. Insert all vertices into priority queue $\boldsymbol{P}$, 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***)

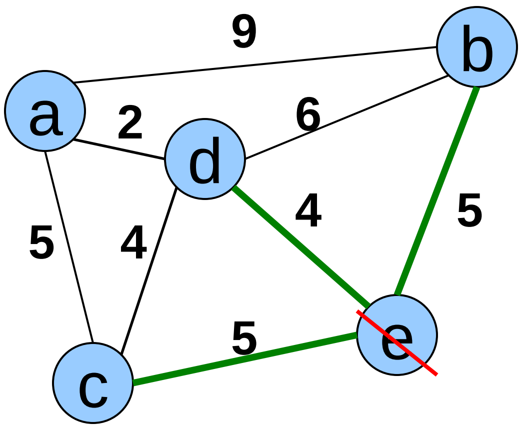
    If ***weight***(***u,w***) **<** ***D***(***w***),

    a. ***parent***(***w***) **=** ***u***

    b. ***D***(***w***) **=** ***weight***(***u,w***)

    c. Update the priority queue to reflect new distance for ***w***

Prim’s algorithm



e	d	b	c	a
0	∞	∞	∞	∞

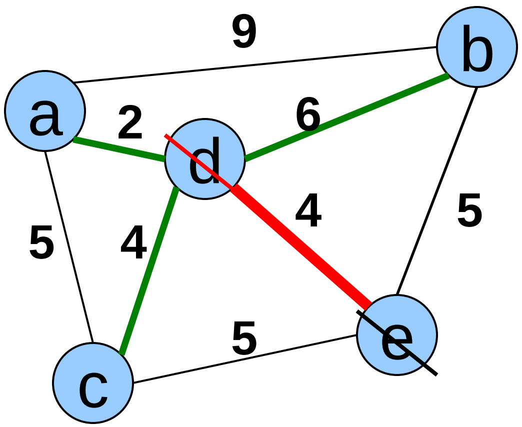
Vertex	Parent
e	-
b	-
c	-
d	-

d	b	c	a
4	**5**	**5**	∞

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



d	b	c	a
4	**5**	**5**	∞

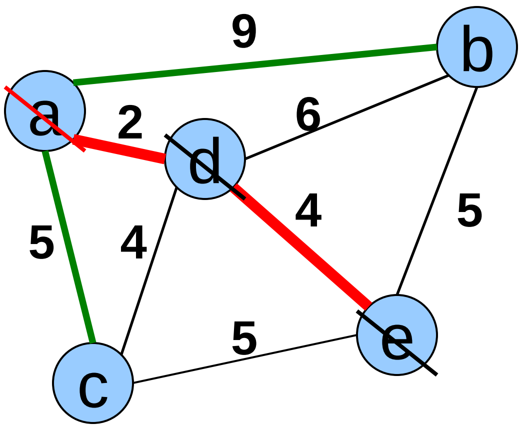
Vertex	Parent
e	-
b	e
c	e
d	e

---

a	c	b
2	**4**	**5**

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

Prim's algorithm



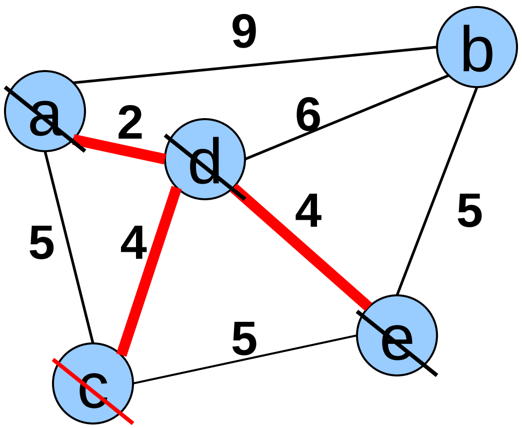
a	c	b
2	**4**	**5**

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

c	b
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

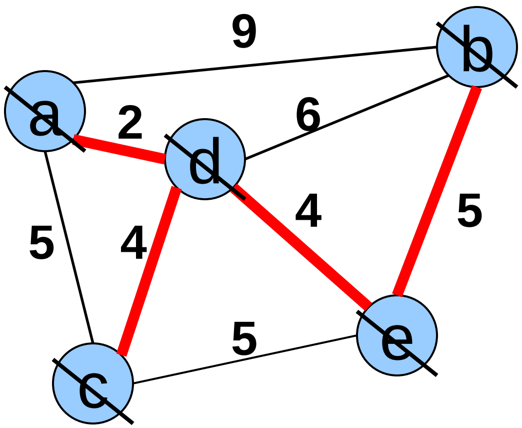
b
5

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

b
5



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

The final minimum spanning tree

Running time of Prim's algorithm (without heaps)

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 without heaps: O(|E| + |V| ^{2})

When heaps are used, apply same analysis as for Dijkstra's algorithm (p.469) (*good exercise*)

Prim's Algorithm Invariant

- At each step, we add the edge $(\boldsymbol{u},\boldsymbol{v})$ s.t. the weight of $(\boldsymbol{u},\boldsymbol{v})$ is **minimum** among all edges where $\boldsymbol{u}$ is in the tree and $\boldsymbol{v}$ 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!

Correctness of Prim's

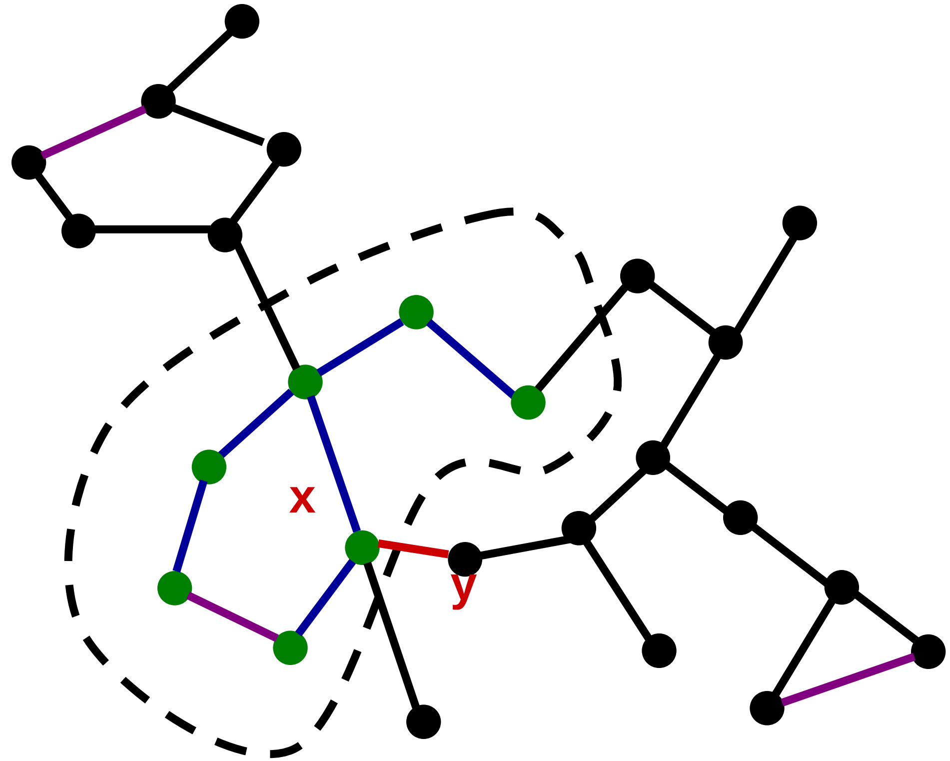
- This algorithm adds *n-1* edges without creating a cycle, so clearly it creates a spanning tree of any connected graph (*you should be able to prove this*).

But is this a *minimum* spanning tree?

Suppose it wasn't.

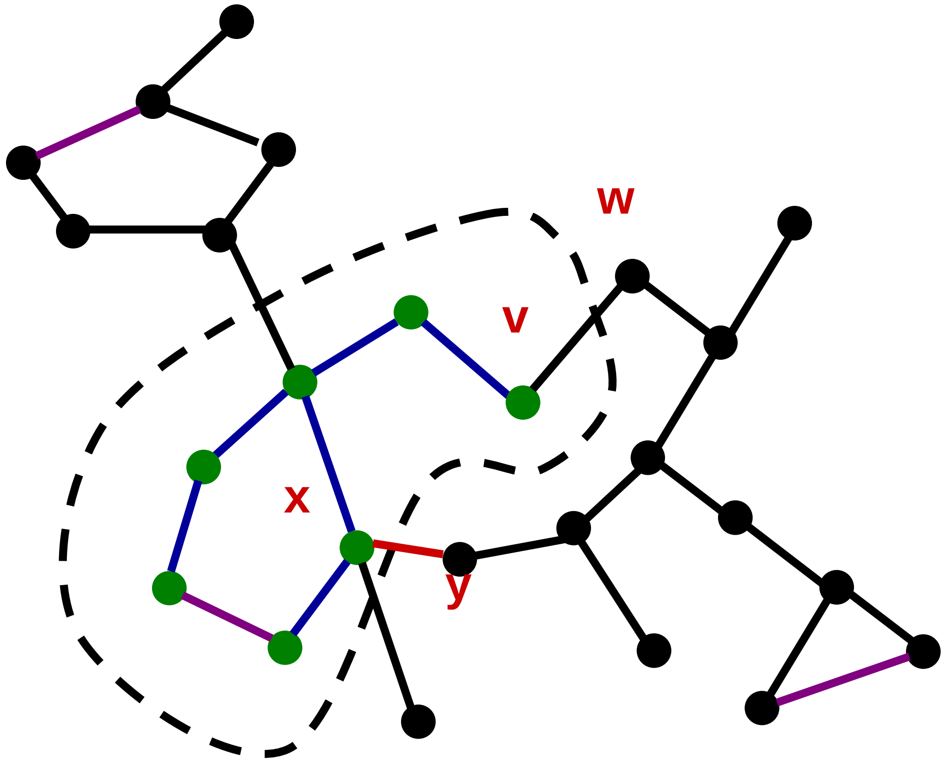
- There must be point at which it fails, and in particular there must a single edge whose insertion first prevented the spanning tree from being a minimum spanning tree.

Correctness of Prim's



- Let G be a connected, undirected graph
- Let **S** be the set of edges chosen by Prim's algorithm *before* choosing an errorful edge **(*x*,*y*)**
- Let **V'** be the vertices incident with edges in **S**
- Let **T** be a MST of G containing all edges in **S**, but not **(*x*,*y*)**.

Correctness of Prim's



- Edge ***(x,y)*** is not in **T**, so there must be a path in **T** from ***x*** to ***y*** since **T** is connected.
- Inserting edge ***(x,y)*** into **T** will create a cycle
- There is exactly one edge on this cycle with exactly one vertex in **V'**, call this edge **(*v*,*w*)**

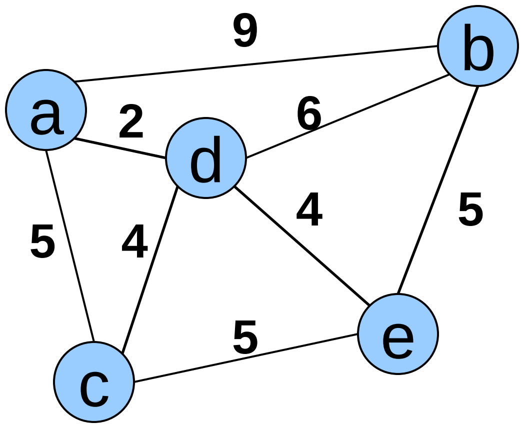
Correctness of Prim's

- Since Prim's chose ***(x,y)*** over ***(v,w)***, w(**v,w**) >= w(**x,y**).
- We could form a new spanning tree **T'** by swapping ***(x,y)*** for ***(v,w)*** in **T** (*prove this is a spanning tree*).
- w(**T'**) is clearly no greater than w(**T)**
- But that means **T'** is a MST
- And yet it contains all the edges in **S**, and also ***(x,y)***

...Contradiction

Another Approach

- 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}

Correctness of Kruskal’s

- Inserting edge **e** into **T** will create a cycle
- There must be an edge on this cycle which is not in **K** (*why??*). Call this edge **e’**
- **e’** must be in **T - S,** so (by our lemma) w(**e’**) >= w(**e**)
- We could form a new spanning tree **T’** by swapping **e** for **e’** in **T** (*prove this is a spanning tree*).
- w(**T’**) is clearly no greater than w(**T)**
- But that means **T’** is a MST
- And yet it contains all the edges in **S**, and also **e**

...Contradiction

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**!