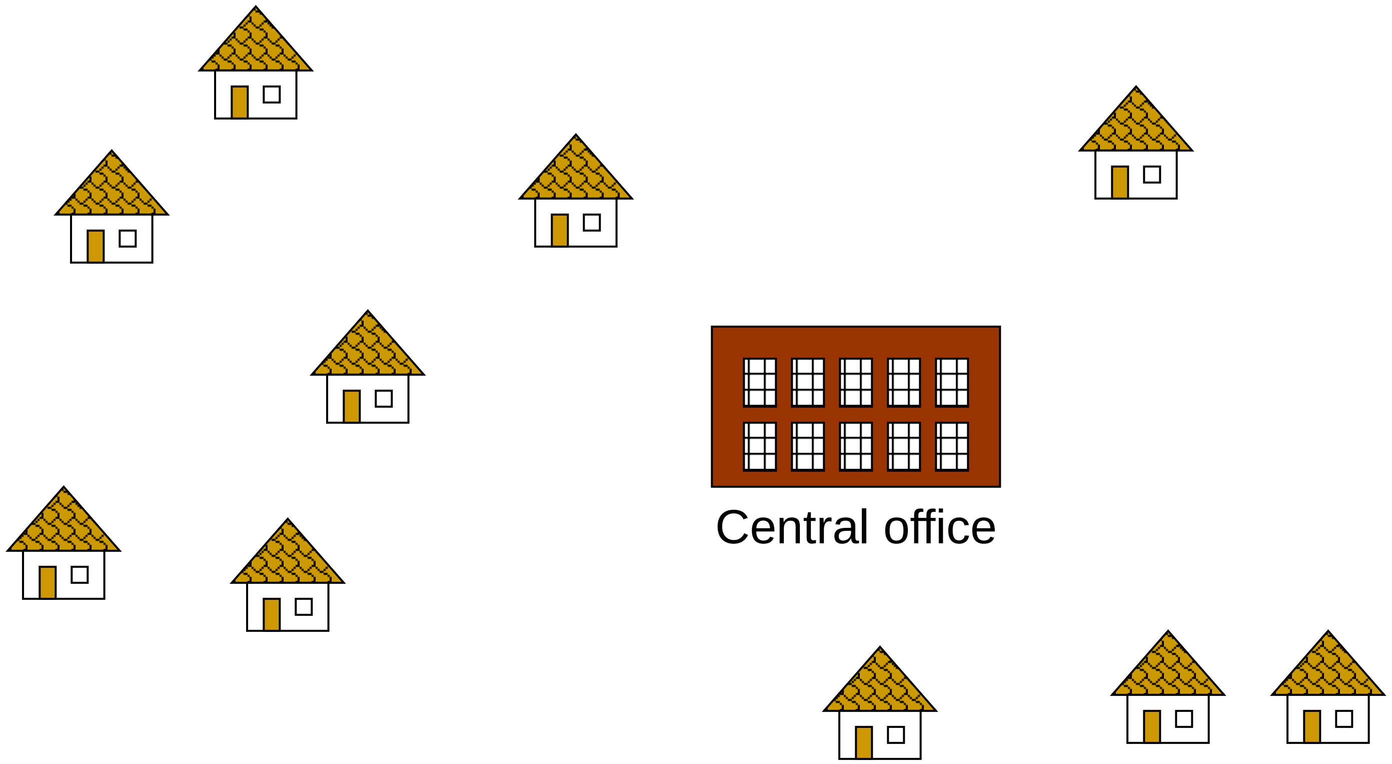
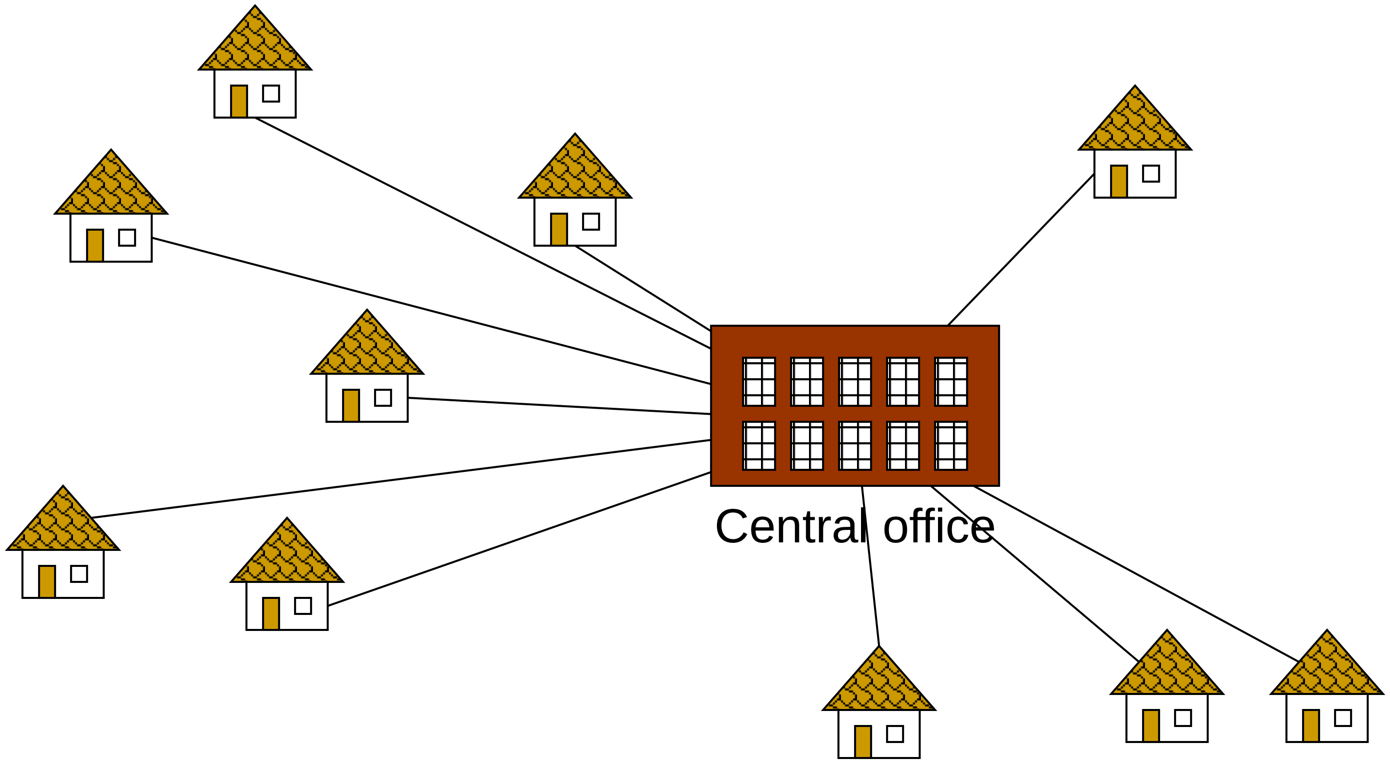
ADVANCED ANALYSIS OF ALGORITHMS
CPS 5440

GRAPH ALGORITHMS: MINIMUM SPANNING TREES

PROBLEM: LAYING TELEPHONE WIRE

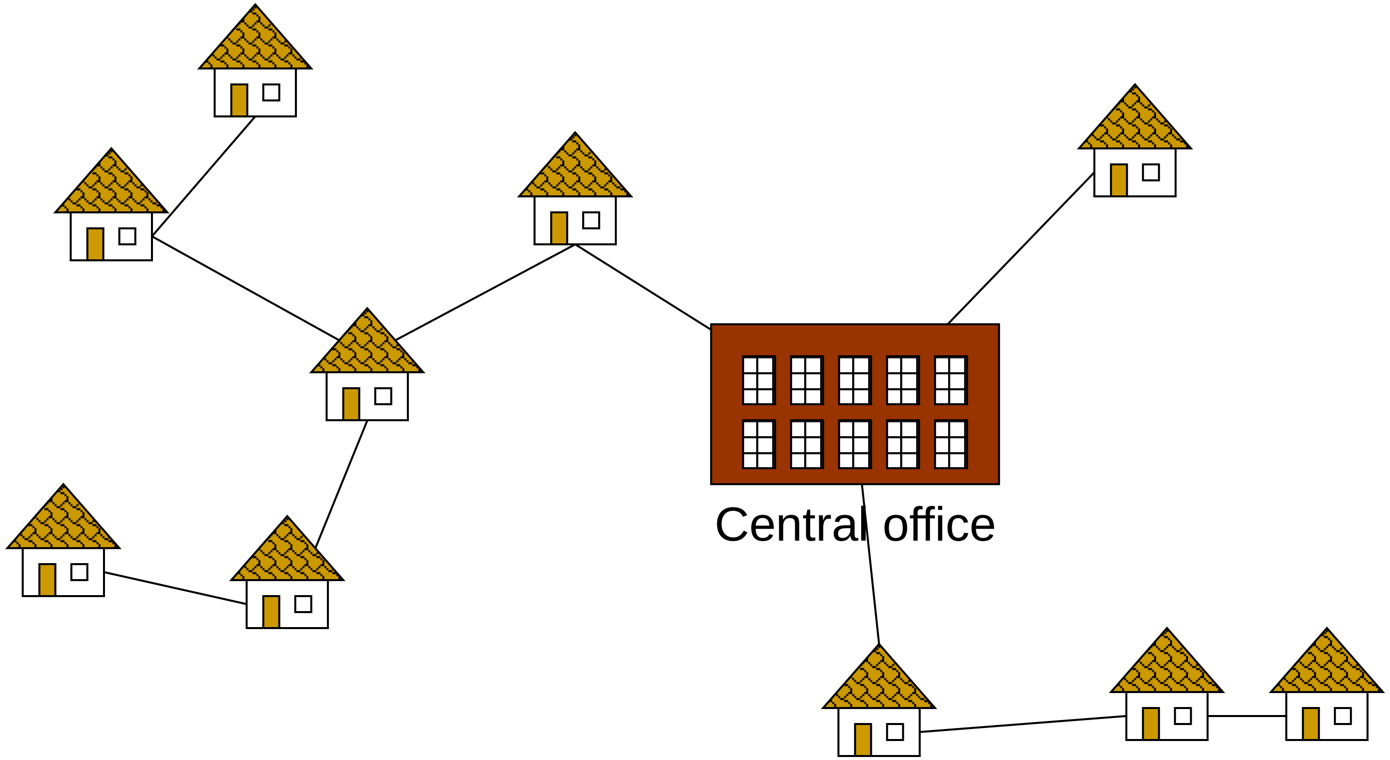


WIRING: NAÏVE APPROACH



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

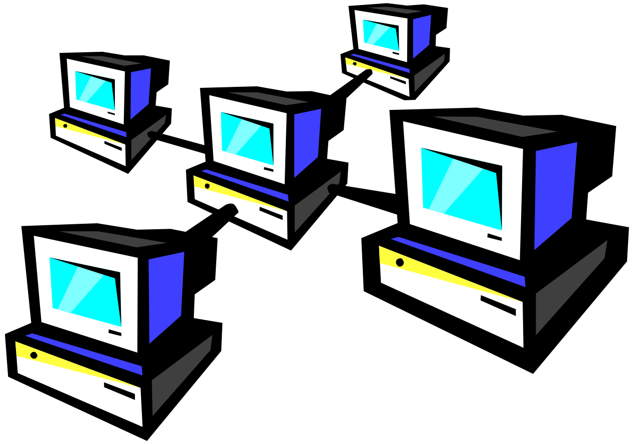
- it is a tree $(i.e.,$ it is acyclic)
- it covers all the vertices $\boldsymbol{V}$
  - contains $|V|$ - $\boldsymbol{1}$ edges
- the total cost associated with tree edges is the minimum among all possible spanning trees
- not necessarily unique

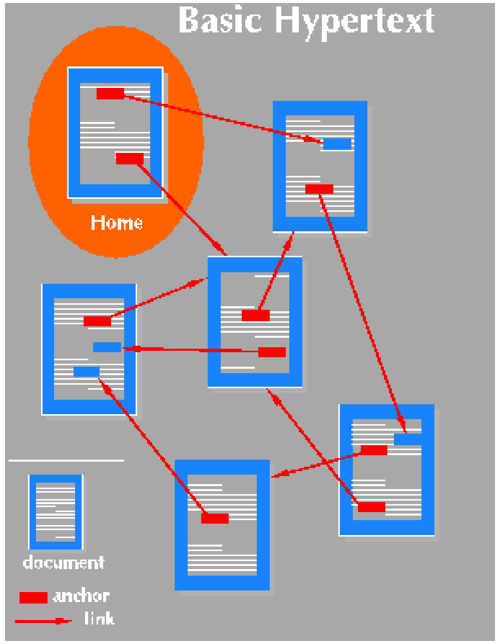
PROPERTIES OF MINIMUM SPANNING TREES

- Minimum spanning tree is **not** unique
- MST has no cycles – see why:
  - We can take out an edge of a cycle and still have the vertices connected while reducing the cost
- # of edges in an MST:
  - $|V| - 1$

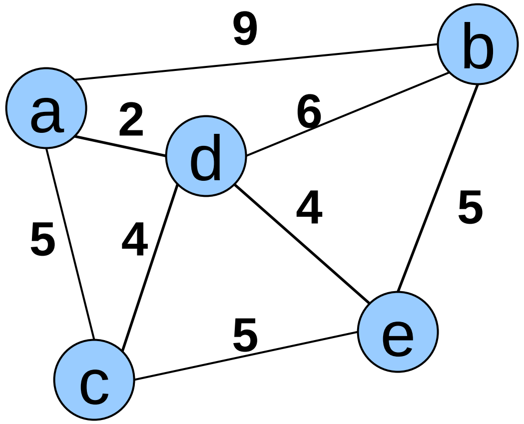
APPLICATIONS OF MST

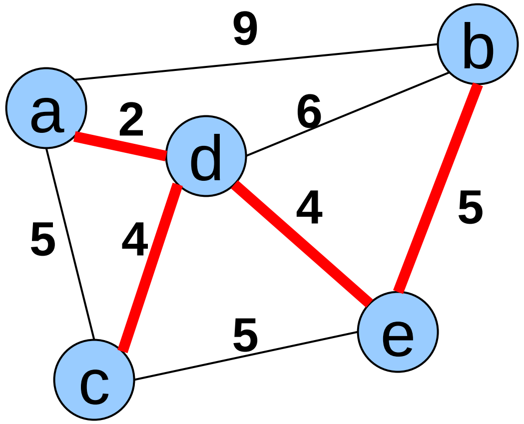
- Find the least expensive way to connect a set of cities, terminals, computers, etc.





HOW CAN WE GENERATE A MST?





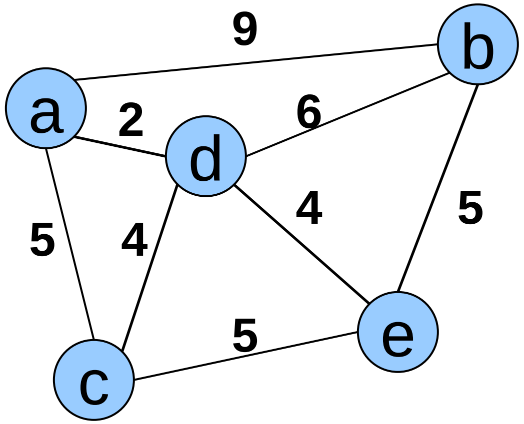
Initialization

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

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

c. For all vertices $\boldsymbol{v} \in \boldsymbol{V}$, $\boldsymbol{v \neq r}$, 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	-

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., $w \in P$)

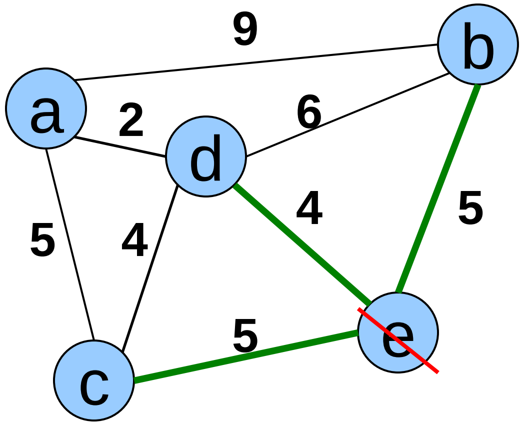
   If $weight(u,w) < D(w)$,

   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	∞	∞	∞	∞

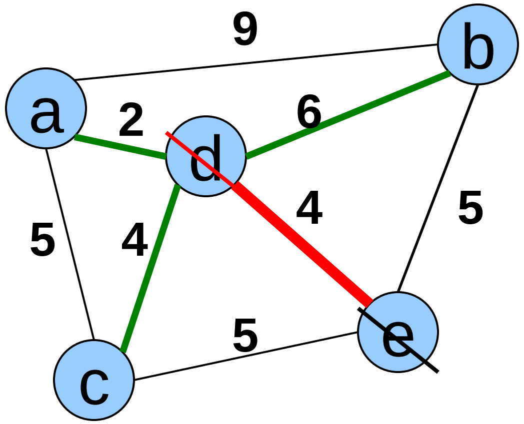
Vertex	Parent
e	-
b	-
c	-
d	-

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



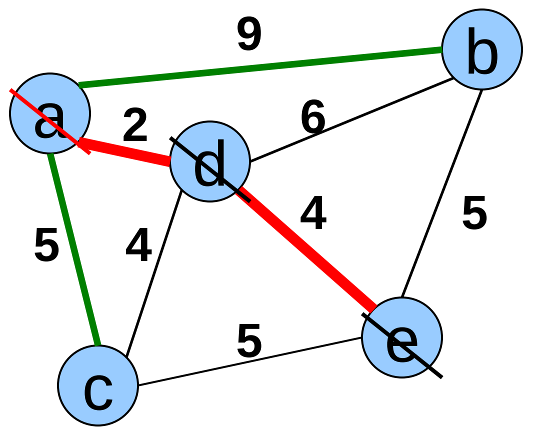
d	b	c	a
4	5	5	∞

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**

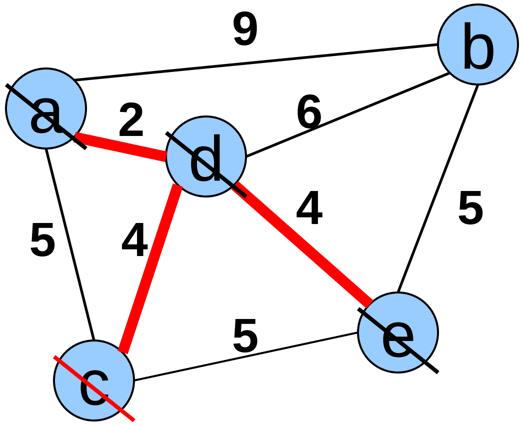
Vertex	Parent
e	-
b	e
c	d
d	e
a	d

---

c	b
4	**5**

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

PRIM'S ALGORITHM



c	b
4	**5**

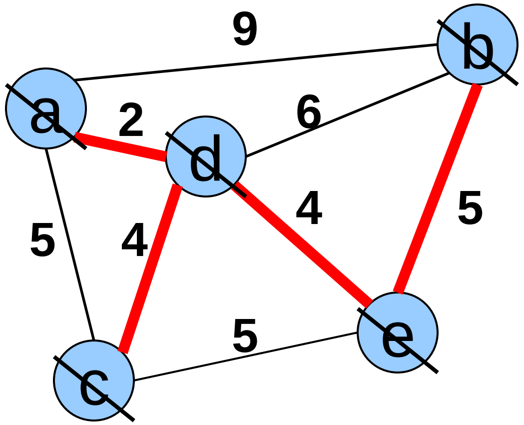
Vertex	Parent
e	-
b	e
c	d
d	e
a	d

---

b
5

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

PRIM'S ALGORITHM



b
5

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

The final minimum spanning tree

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

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

RUNNING TIME OF PRIM'S ALGORITHM (WITHOUT HEAPS)

- Depending on the heap implementation, running time could be improved!

	EXTRACT-MIN	DECREASE-KEY	Total
binary heap	O(lgV)	O(lgV)	O(ElgV)
Fibonacci heap	O(lgV)	O(1)	O(VlgV + E)

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 an MST for the graph!

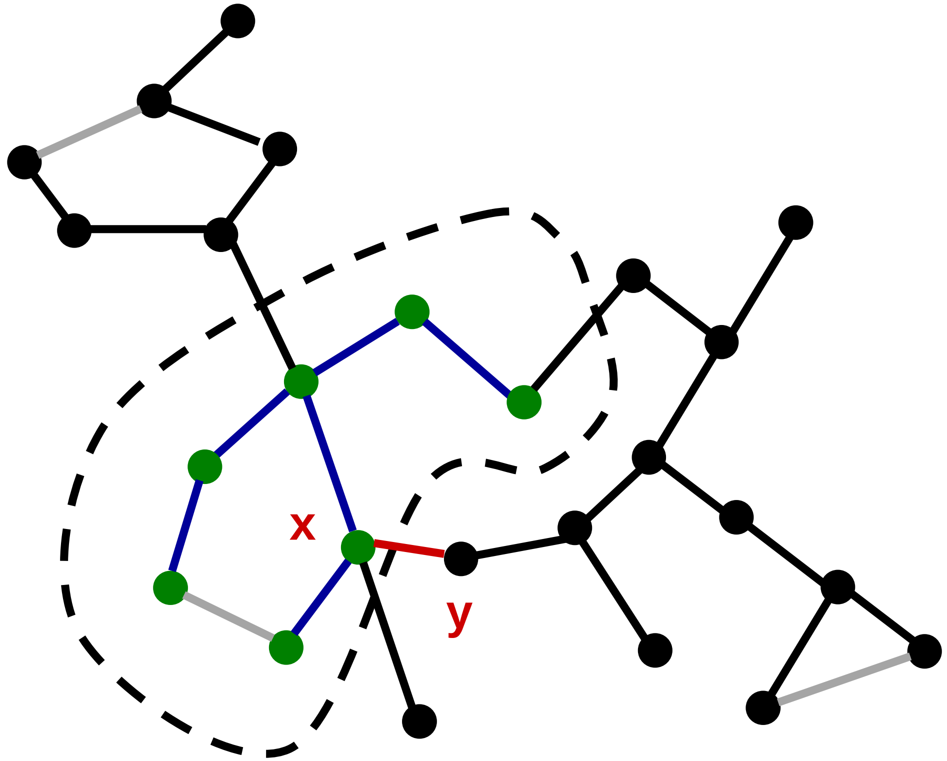
CORRECTNESS OF PRIM'S

This algorithm adds $V - 1$ edges without creating a cycle, so clearly, it creates a spanning tree of any connected graph.

But is this a *minimum-spanning* tree?
Suppose it wasn't. $(Proof\ by\ contradiction!)$

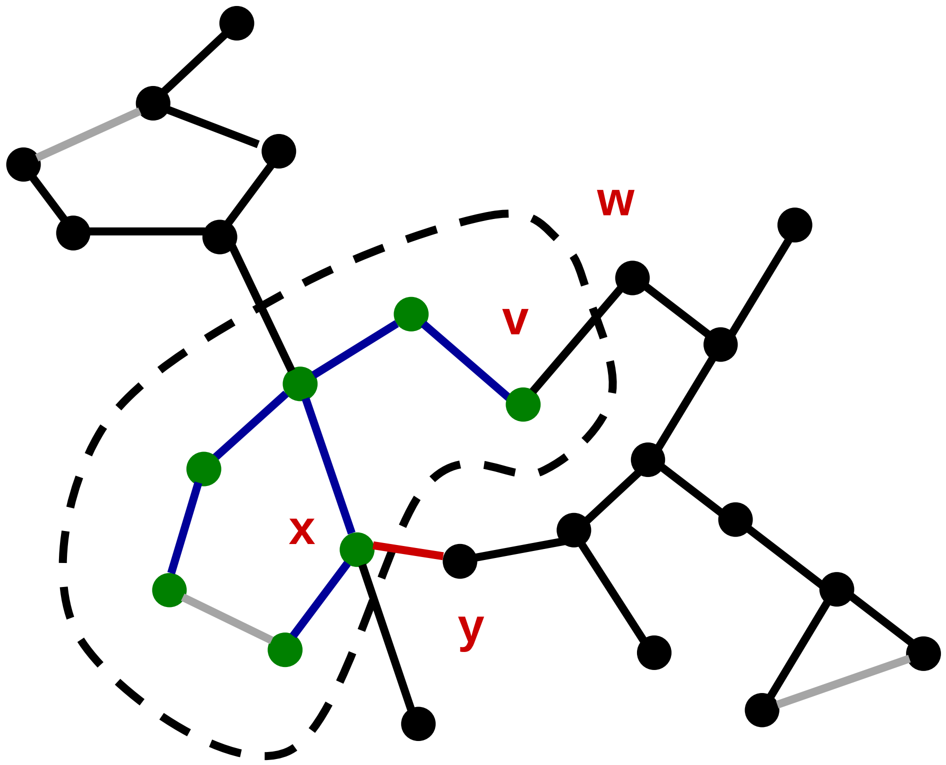
There must be a point at which it fails, and in particular, there must be a single edge whose insertion first prevented the spanning tree from being a minimum spanning tree.

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 $(\boldsymbol{x}, \boldsymbol{y})$ over $(\boldsymbol{v}, \boldsymbol{w})$, $w(\boldsymbol{v}, \boldsymbol{w}) \geq w(\boldsymbol{x}, \boldsymbol{y})$.
- We could form a new spanning tree **T'** by swapping $(\boldsymbol{x}, \boldsymbol{y})$ for $(\boldsymbol{v}, \boldsymbol{w})$ in **T**.
- $w(\boldsymbol{T'})$ is clearly no greater than $w(\boldsymbol{T})$
- But that means **T'** is an MST
- And yet it contains all the edges in **S**, and also $(\boldsymbol{x}, \boldsymbol{y})$

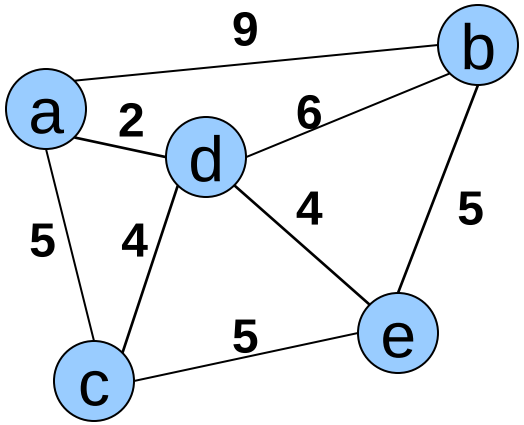
…Contradiction

PRIM'S ALGORITHM

- Prim's algorithm is a **"greedy"** algorithm
  - Greedy algorithms find solutions based on a sequence of choices
  - The choices are "$\boldsymbol{locally}$" optimal at each step
  - We make the best decision at this point of time
  - We ignore the future impact of the current move.
- Nevertheless, Prim's greedy strategy produces a globally optimum solution!
- It is not usual that greedy approaches find a globally optimum solution
- MST has some properties that make Greedy approaches optimal.

ANOTHER APPROACH

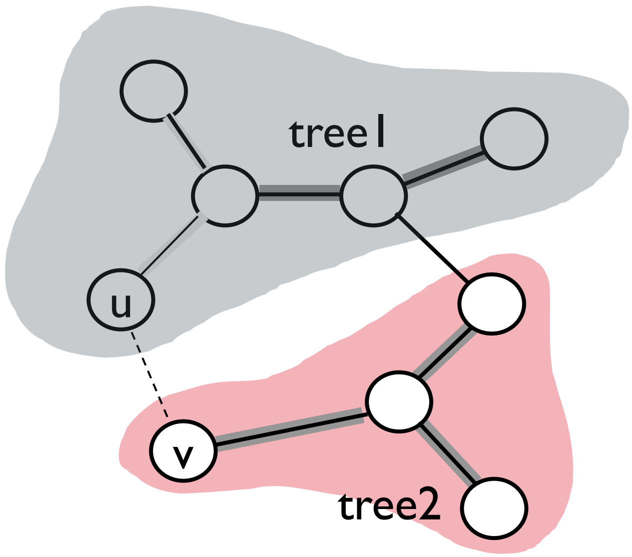
- Create a forest of trees from the vertices
- Repeatedly merge trees by adding "**safe edges**" until only one tree remains
- A "safe edge" is an edge of minimum weight which does not create a cycle



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

KRUSKAL'S ALGORITHM

- How is it different from Prim's algorithm?
  - Prim's algorithm grows one tree all the time
  - Kruskal's algorithm grows multiple trees $(i.e.,$ a forest) at the same time.
  - Trees are merged together using **safe** edges
  - Since an MST has exactly $|V| - 1$ edges, after $|V| - 1$ merges, we would have only one component

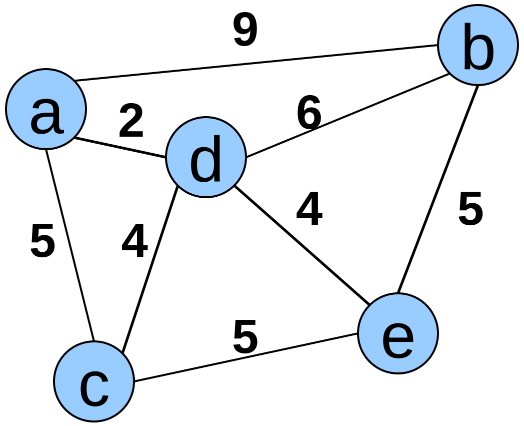


Initialization

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

b. Initialize the set of "safe edges" A comprising the MST to the empty set

c. Sort edges by increasing weight



F **=** {a}, {b}, {c}, {d}, {e}

A **=** $\varnothing$

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 U and V

a. add edge (u,v) to the safe edge set

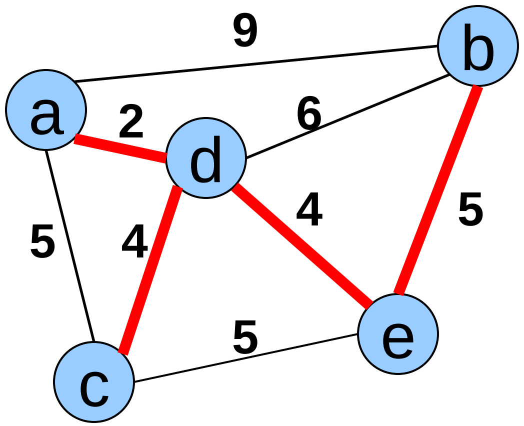
$A = A \cup \{(u,v)\}$

b. merge the sets U and V

$F = F - U - V + (U \cup V)$

Return A

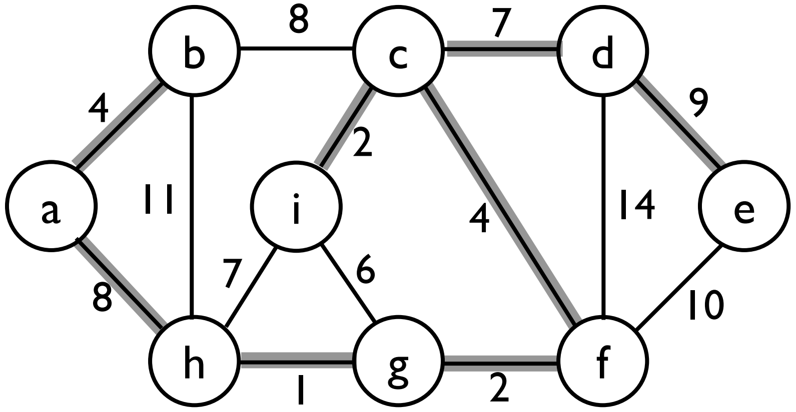
- Running time bounded by sorting (or findMin)
- $O(|E| \log |E|)$, or equivalently, $O(|E| \log |V|)$



***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



1: (h, g)
2: (c, i), (g, f)
4: (a, b), (c, f)
6: (i, g)
7: (c, d), (i, h)
8: (a, h), (b, c)
9: (d, e)
10: (e, f)
11: (b, h)
14: (d, f)

{a}, {b}, {c}, {d}, {e}, {f}, {g}, {h}, {i}

1.	Add (h, g)	{g, h}, {a}, {b}, {c}, {d}, {e}, {f}, {i}
2.	Add (c, i)	{g, h}, {c, i}, {a}, {b}, {d}, {e}, {f}
3.	Add (g, f)	{g, h, f}, {c, i}, {a}, {b}, {d}, {e}
4.	Add (a, b)	{g, h, f}, {c, i}, {a, b}, {d}, {e}
5.	Add (c, f)	{g, h, f, c, i}, {a, b}, {d}, {e}
6.	Ignore (i, g)	{g, h, f, c, i}, {a, b}, {d}, {e}
7.	Add (c, d)	{g, h, f, c, i, d}, {a, b}, {e}
8.	Ignore (i, h)	{g, h, f, c, i, d}, {a, b}, {e}
9.	Add (a, h)	{g, h, f, c, i, d, a, b}, {e}
10.	Ignore (b, c)	{g, h, f, c, i, d, a, b}, {e}
11.	Add (d, e)	{g, h, f, c, i, d, a, b, e}
12.	Ignore (e, f)	{g, h, f, c, i, d, a, b, e}
13.	Ignore (b, h)	{g, h, f, c, i, d, a, b, e}
14.	Ignore (d, f)	{g, h, f, c, i, d, a, 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

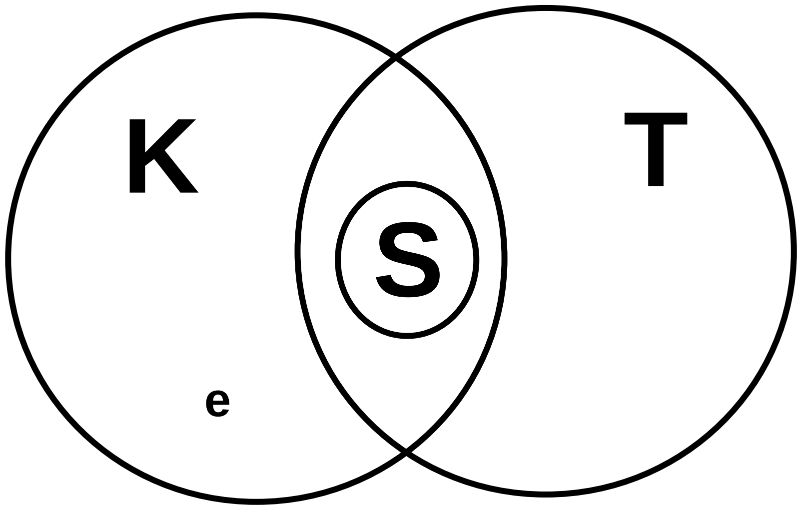
This algorithm adds $n - 1$ edges without creating a cycle, so clearly, it creates a spanning tree of any connected graph.

But is this a *minimum-spanning* tree?
Suppose it wasn't. $(Proof\ by\ contradiction!)$

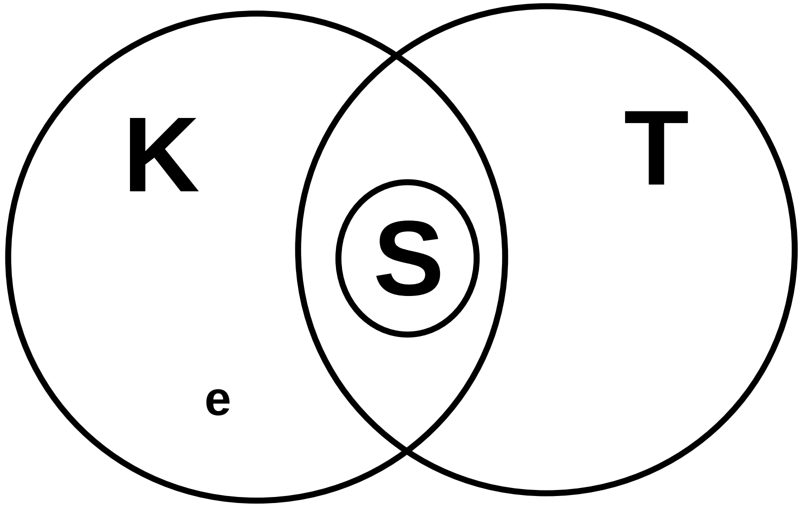
There must be a point at which it fails, and in particular, there must a single edge whose insertion first prevented the spanning tree from being a minimum spanning tree.

CORRECTNESS OF KRUSKAL'S

- Let e be this first errorful edge.
- Let **K** be the Kruskal spanning tree
- Let **S** be the set of edges chosen by Kruskal's algorithm *before* choosing e
- Let **T** be an MST containing all edges in **S**, but not e.



Lemma: $w(\boldsymbol{e'}) \geq w(\boldsymbol{e})$ for all edges $\boldsymbol{e'}$ in **T - S**



Proof (*by contradiction*):

- Assume there exists some edge $\boldsymbol{e'}$ in **T - S**, $w(\boldsymbol{e'}) < w(\boldsymbol{e})$
- Kruskal's must have considered $\boldsymbol{e'}$ before $\boldsymbol{e}$

- However, since $\boldsymbol{e'}$ is not in **K**, it must have been discarded because it caused a cycle with some of the other edges in **S**.
- But $\boldsymbol{e'} + \boldsymbol{S}$ is a subgraph of **T**, which means it cannot form a cycle **...Contradiction**

CORRECTNESS OF KRUSKAL'S

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

…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**!

PRIM VS KRUSKAL ALGORITHMS

- Compare Prim's algorithm with and Kruskal's algorithm assuming:

(a) **Sparse graphs:**

In this case, $E = O(V)$

Kruskal:

- $O(ElgE) = O(VlgV)$

Prim:

- Binary heap: $O(ElgV) = O(VlgV)$
- Fibonacci heap: $O(VlgV + E) = O(VlgV)$

(b) **dense graphs**

In this case, $E = O(V^2)$

Kruskal:

- $O(ElgE) = O(V^2lgV^2) = O(2V^2lgV) = O(V^2lgV)$

Prim:

- Binary heap: $O(ElgV) = O(V^2lgV)$
- Fibonacci heap: $O(VlgV + E) = O(VlgV + V^2) = O(V^2)$


Q
A

REFERENCES

https://www.usebackpack.com/resources/17659/download?1523493802