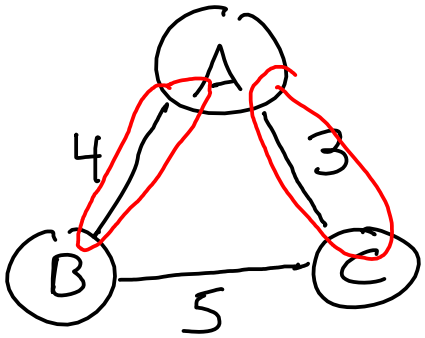
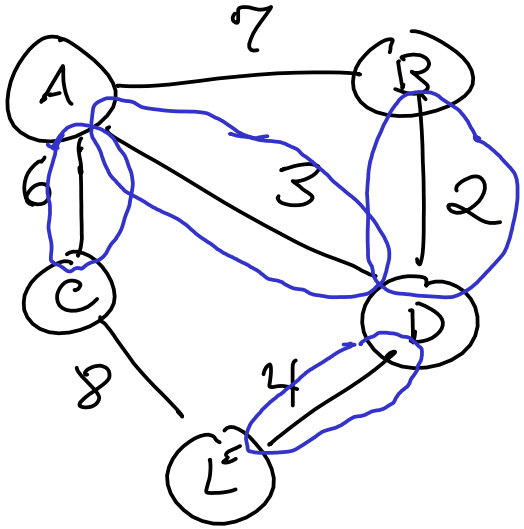
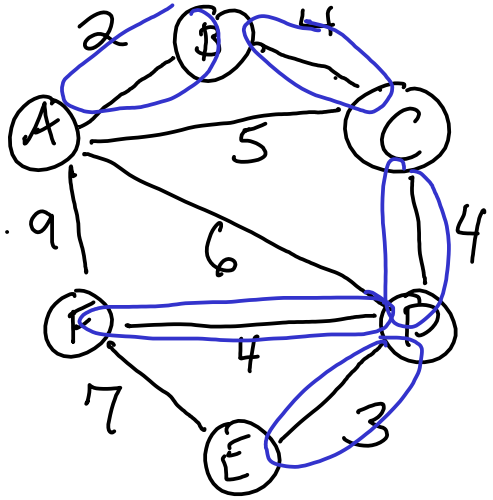
Minimum Spanning Trees

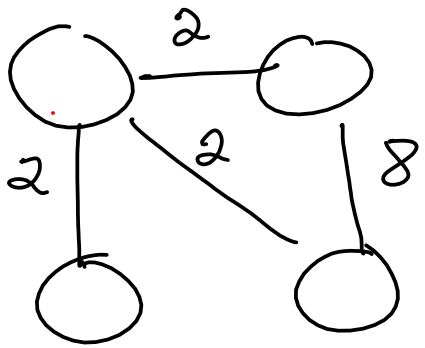
MST : set of edges which, given V in G_E, connect that graph, which $\subseteq E$, least cost





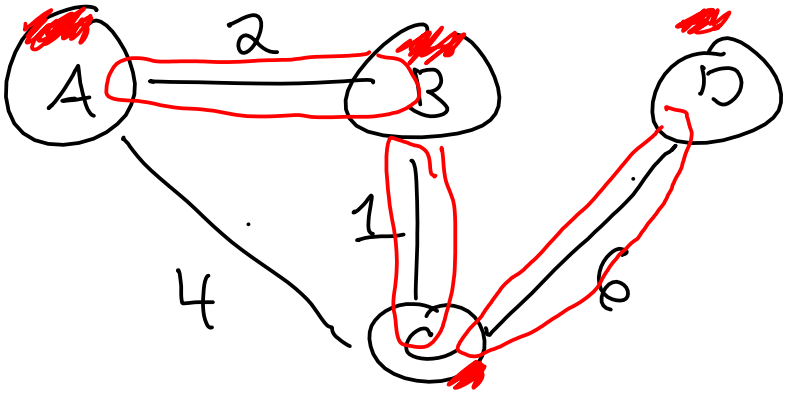
Start w/ cheapest edge





Prim's Algorithm

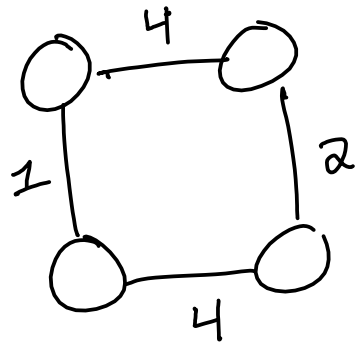
1. Start w/ cheapest edge in graph, add that
2. Remember: been to that edge's source & target
3. Find from edges leaving vertices to which you have been the cheapest that goes somewhere new
4. Add that edge; remember new place
5. Repeat 3 & 4 until connected. (been everywhere)



Prim's (more detail)

```
Function Prim's(g):
    visited ← new Set
    edges ← new Set
    e ← cheapest edge in g
    edges.add(e)
    visited.add(e.source)
    visited.add(e.target)
    q ← new PrioQueue
    for each edge leaving e.source or e.target:
        if edge ∉ edges:
            q.enqueue(edge.weight, edge)
    while q not empty:
        e ← q.dequeue()
        if e.target ∉ visited or e.source ∉ visited:
            edges.add(e)
            if e.source ∉ visited:
                for each edge leaving e.source
                    q.enqueue(edge.weight, edge)
                if e.target ∉ visited:
                    for each edge leaving ...
            visited.add(e.source)
            visited.add(e.target)
```

while visited ≠ g.V



Kruskal's Algorithm

Union-Find

```
q ← new PrioQueue
for every edge in g:
    q.enqueue(edge.weight, edge)
answer ← new Set
while q not empty:
    e ← q.dequeue
    if e connects two prev disconnected subgraphs:
        answer:add(e)
```