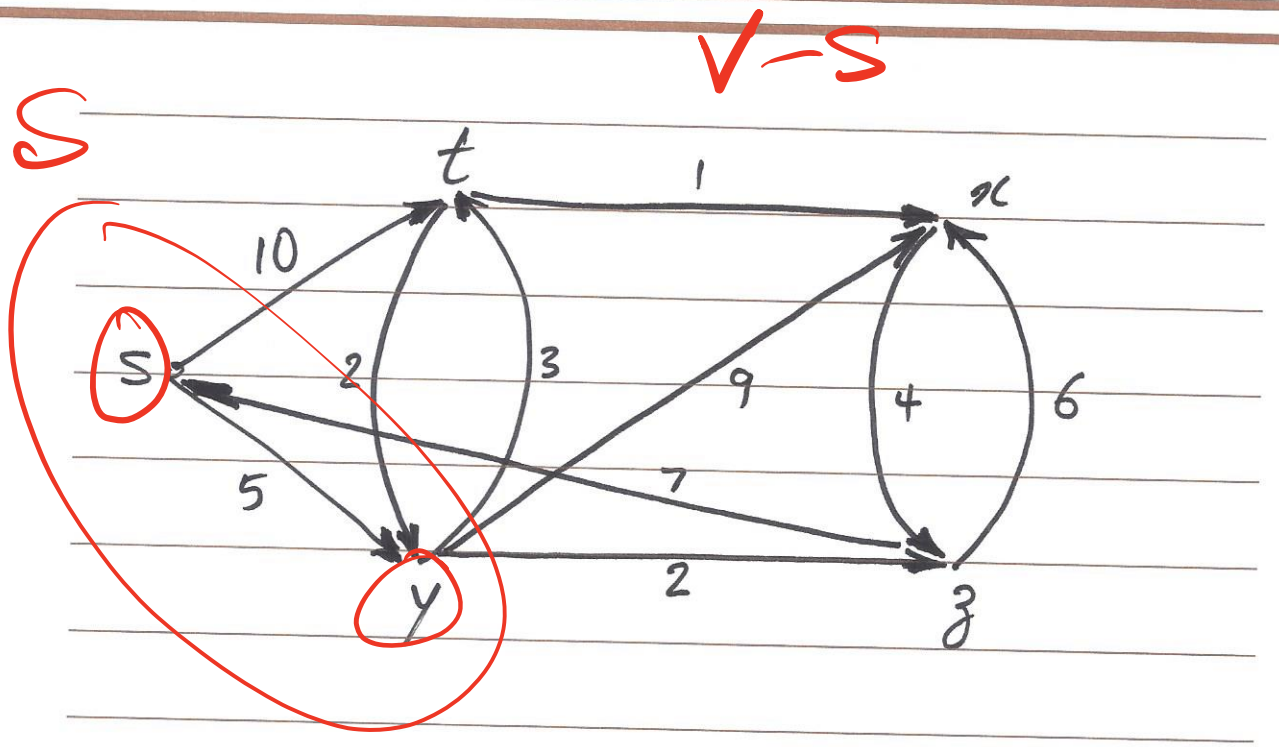
Shortest Path

Problem Statement:

Given $G = (V, E)$ with $w(u, v) \geq 0$ for each edge $(u, v) \in E$, find the shortest path from $s \in V$ to $V - s$.

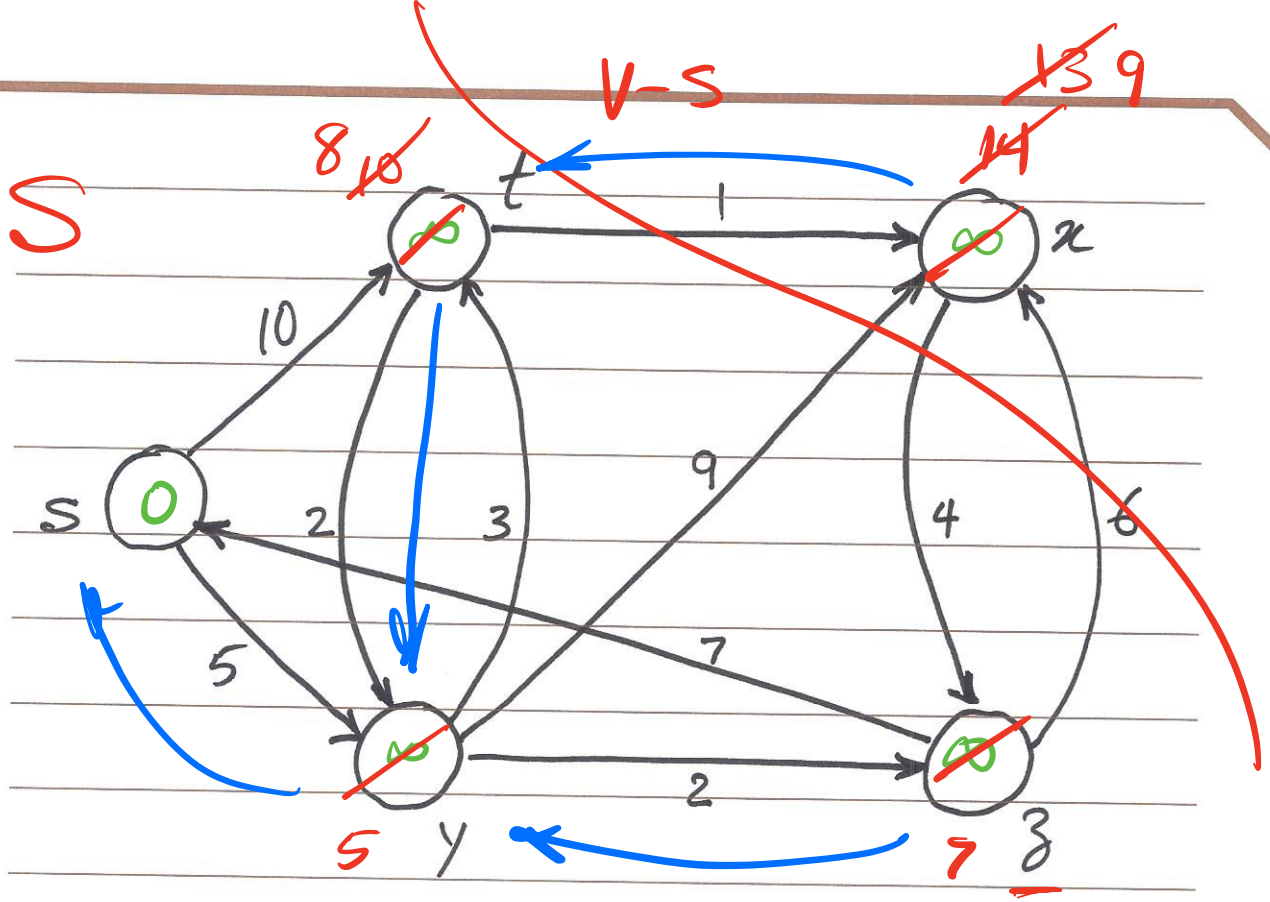


Solution

1. Start with a set S of vertices whose final shortest path we already know.

2. At each step, find a vertex $v \in V - S$ with shortest distance from s.

3. Add v to S, and repeat.



$S_1 = \{s\}$, $S_2 = \{s, y\}$, $S_3 = \{s, y, z\}$

$S_4 = \{s, y, z, t\}$, $S_5 = \{s, y, z, t, x\}$

Dijkstras' Alg.

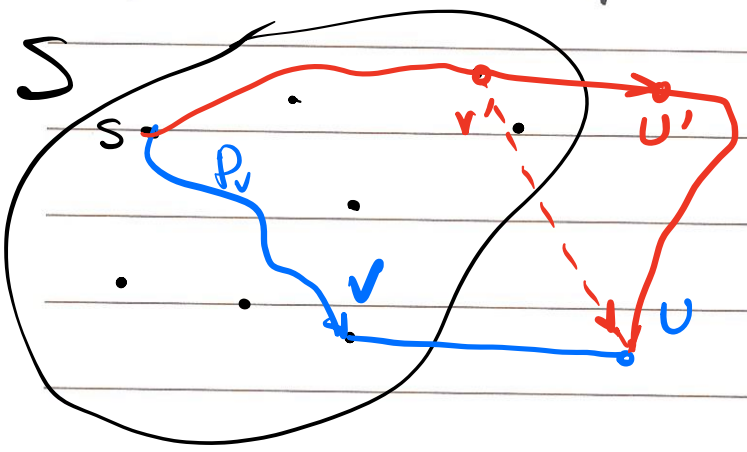
Proof of Correctness

We will prove that at each step, Dijkstra's algorithm finds the shortest path to a new node in the graph.

Proof by mathematical induction:

Base Case: $|S| = 1$, $S = \{s\}$ and $d(s) = 0$

Inductive Step: Suppose the claim holds when $|S| = k$ for some $k \geq 1$. We now grow S to size $k+1$ and prove that we have found the shortest path to the new node.



Implementation of Dijkstra's

Initially $S = \{s\}$ and $d(s) = 0$
for all other nodes $d(u) = \infty$

While $S \neq V$
  Select a node $v \notin S$ with at least one edge from S for which
  $d(v) = \min_{e(u,v): u \in S} (d(u) + l_e)$
  Add v to S
endwhile

More Detailed Implementation of Dijkstra's

S = Null

Initialize priority Queue Q with all nodes V where d(v) is the key value.
(All d(v)'s are $=\infty$, except for s where d(s)=0)

While $S \neq V$
    v = Extract_Min(Q)
    S = S ∪ {v}
    for each vertex $u \in Adj(v)$
        if d(u) > d(v) + le;
            Decrease_key(Q, u, d(v)+le)
    end for
end while

O(n)

O(n)

$O(n^2)$ times!

↳ O(m)

m: total no. of edges

n: total no. of nodes

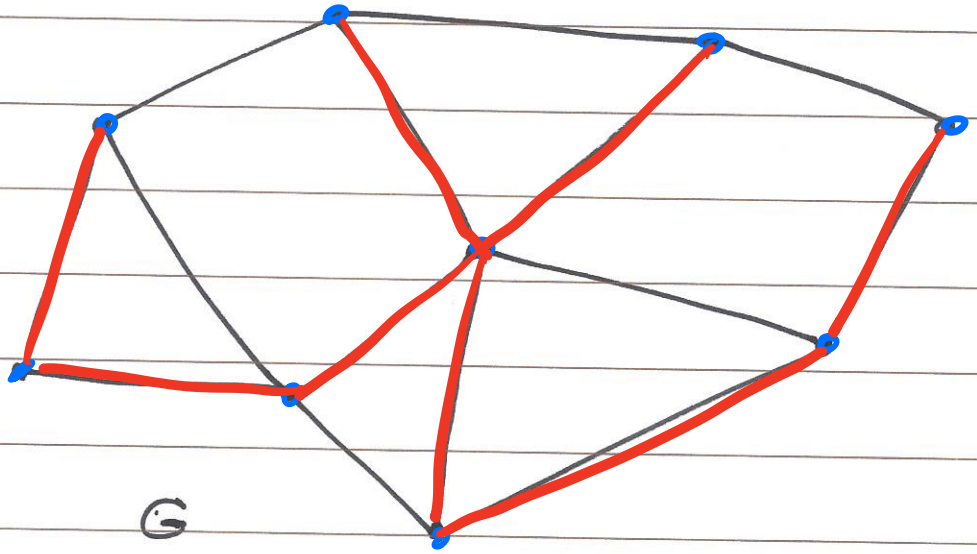
Complexity Analysis

- Initialize Priority Queue $O(n)$
- Max. no. of Extract-Min op's : n
- Max. no. of Decrease-key op's : m

Cost of Dijkstra's	Binary Heap	Binomial Heap	Fibonacci Heap
n. Extract-Min	$O(n \cdot \lg n)$	$O(n \cdot \lg n)$	$O(n \cdot \lg n)$
m. Decrease-key	$O(m \lg n)$	$O(m \lg n)$	$O(m)$
Total Cost	$O(m \cdot \lg n)$	$O(m \lg n)$	$O(m + n \lg n)$
Sparse graphs	$O(n \cdot \lg n)$	$O(n \lg n)$	$O(n \lg n)$
Dense "	$O(n^2 \lg n)$	$O(n^2 \lg n)$	$O(n^2)$

Problem Statement

Find minimum cost network that connects all nodes in G.



Def. Any tree that covers all nodes of a graph is called a spanning tree.

Def. A spanning tree with minimum total edge cost is a minimum spanning tree. (MST)

Problem Statement

Find a MST in an undirected graph

Sol. 1: Sort all edges in increasing order of cost. Add edges to T in this order as long as it does not create a cycle. If it does, discard the edge.

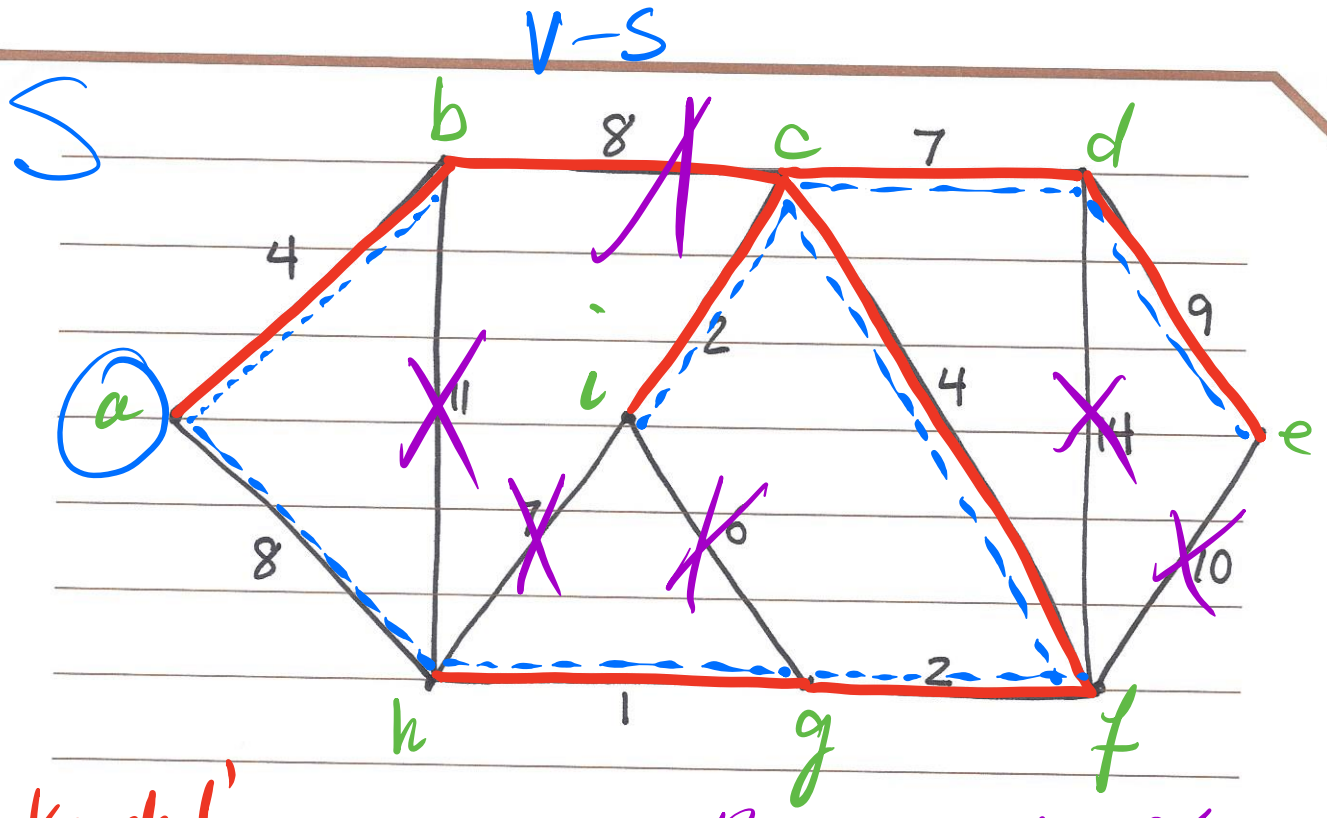
Kruskal's Alg.

Sol. 2: Similar to Dijkstra's algorithm, start with a node set S (initially the root node) on which a minimum spanning tree has been constructed so far. At each step, grow S by one node, adding the node v that minimizes the attachment cost.

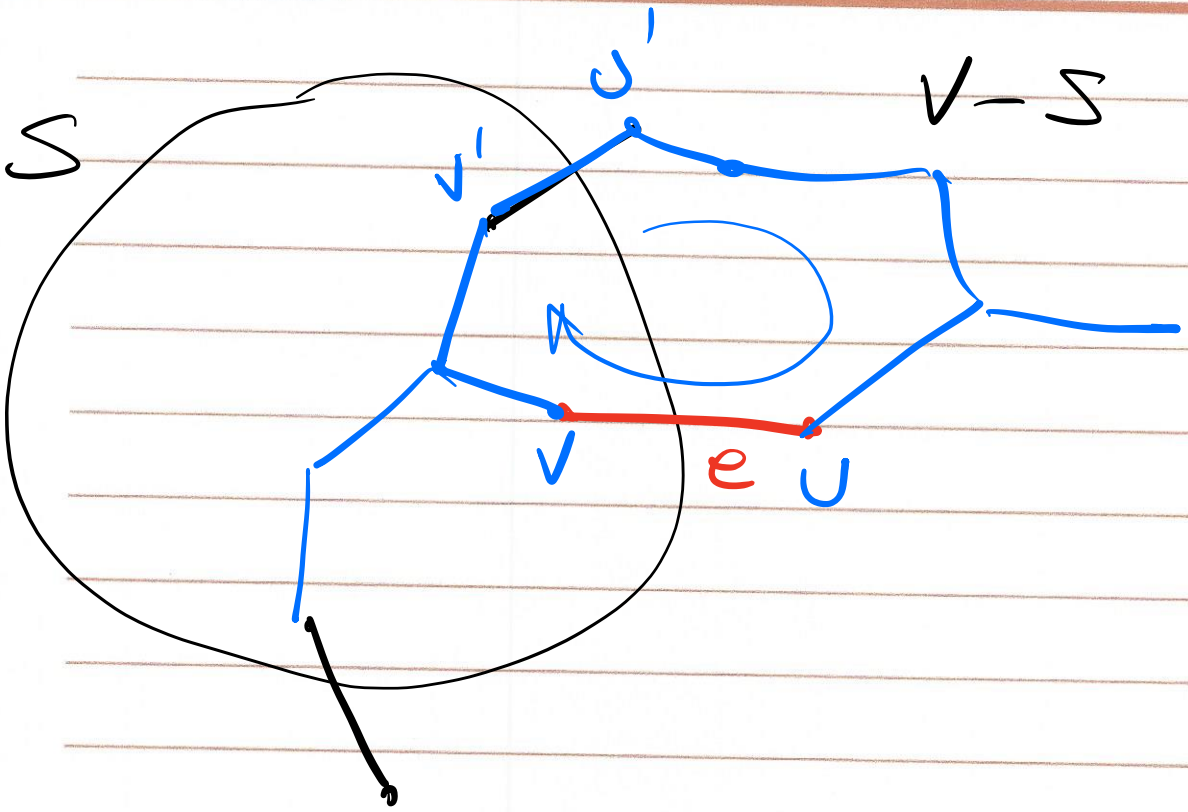
Prim's Alg.

Sol. 3: Backward version of Kruskal's.
Start with a full graph (V, E).
Begin deleting edges in order of decreasing cost as long as it does not disconnect the graph

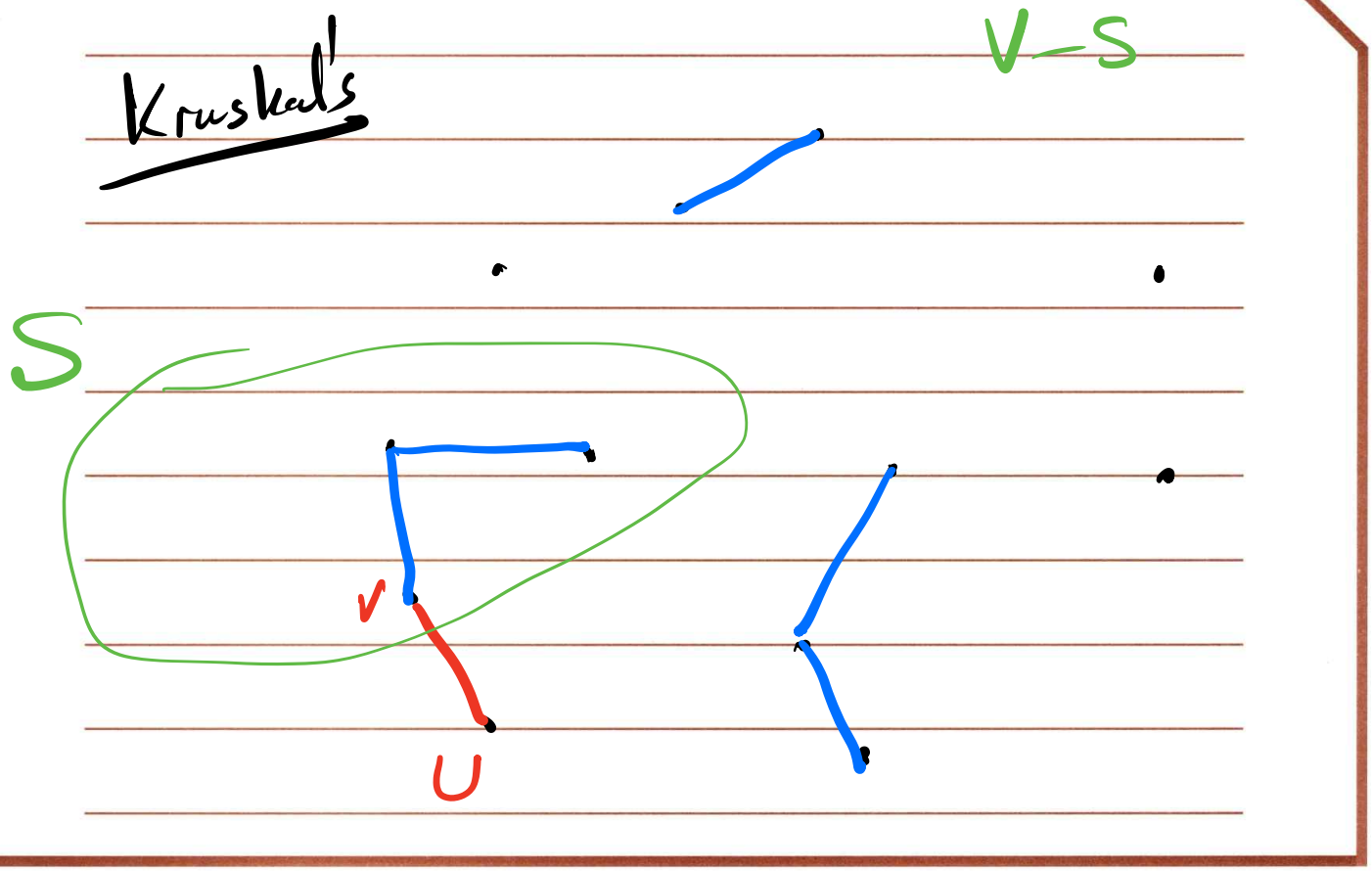
Reverse-Delete Alg.


V-S
S
a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
4
14
9
10
6
1
2
Kruskal's
Prim's
Reverse-Delete

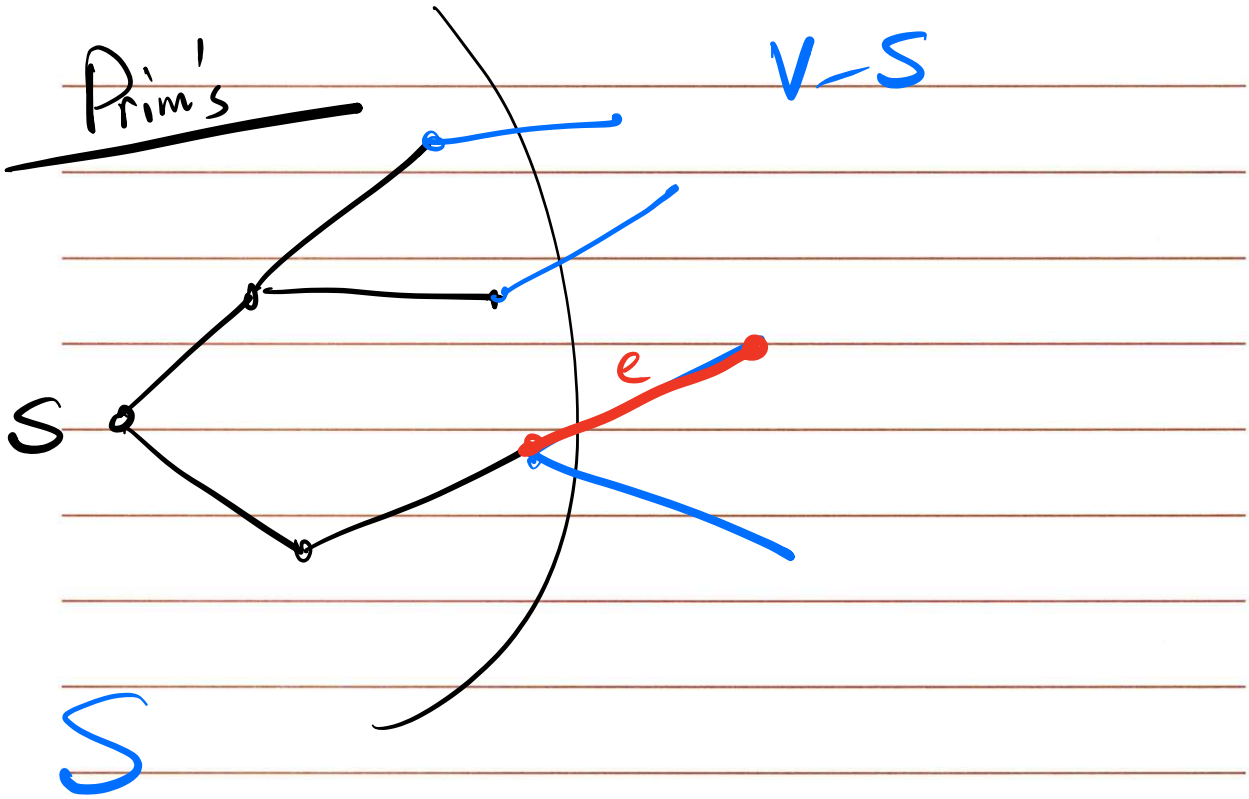
FACT: Let S be any subset of nodes that is neither empty nor equal to all of V, and let edge $e = (v, w)$ be the min cost edge with one end in S and the other end in $V - S$. Then every MST contains the edge e.



Kruskal's



Prim's



FACT: highest cost edge in a cycle cannot belong to any MST.

→ Can use this fact to prove correctness of Reverse-Delete

More Detailed Implementation of Dijkstra's

S = Null

Initialize priority Queue Q with all nodes V where d(v) is the key value.
(All d(v)'s are = ∞, except for s where d(s) = 0)

While $S \neq V$

v = Extract_Min (Q)

S = S ∪ {v}

for each vertex $u \in Adj(v)$

if $d(u) > d(v) + le$;

Decrease_key (Q, u, $d(v) + le$)

end for

end while

Kruskal's

Create an endp set for each node
A = Null
Sort edges in non-decreasing order of weight
for each edge $(u,v) \in E$, taken in this order, if u & v are NOT in the same set then
$A = A \cup \{(u,v)\}$
merge the two sets
end if
endfor

Union-Find data structure

- Make set — O(1) for set size = 1
- Find set — O(1) or O(lg n)
- Union — O(lg n) or O(1)

array impl. ptr impl.

Implementation of Kruskal's

```
A = Null
for each vertex v ∈ V                    O(n)
    Make-set(v)                          O(1)
endfor

Sort the edges of E into non-decreasing order of cost    O(m lg m)

for each edge (u,v) ∈ E in this order.                   O(m)
    if Find-set(u) ≠ Find-set(v) then
        A = A ∪ {(u,v)}
        Union(u,v)
    endif
endfor
```

overall complexity = $O(n) + O(m \lg m) + O(m \lg n)$

$= O(m \lg m)$

Kruskal's

$O(m \log m)$

$O(m \log n^2)$

$O(2m \log n)$

$O(m \log n)$

Prim's

$O(m \log n)$

$=$

Reverse-Delete

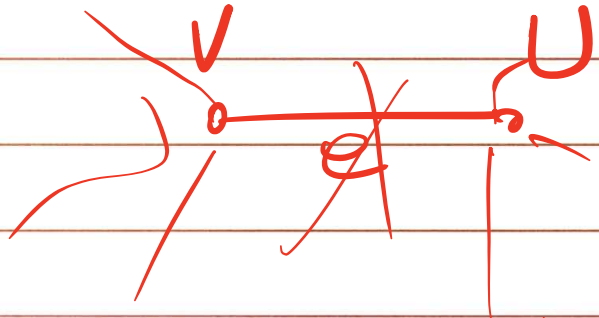
$O(m \log m)$ Sort edges in decreasing order of cost

loop over edges in this order

$O(m)$

Check if removing this edge will disconnect the graph

$O(m(m+n))$ / $O(m)$



overall complexity $= O(m \log m) + O(m^2) = O(m^2)$