Dijkstra's Alg.

Complexity Analysis

initialize the priority Queue $O(n)$

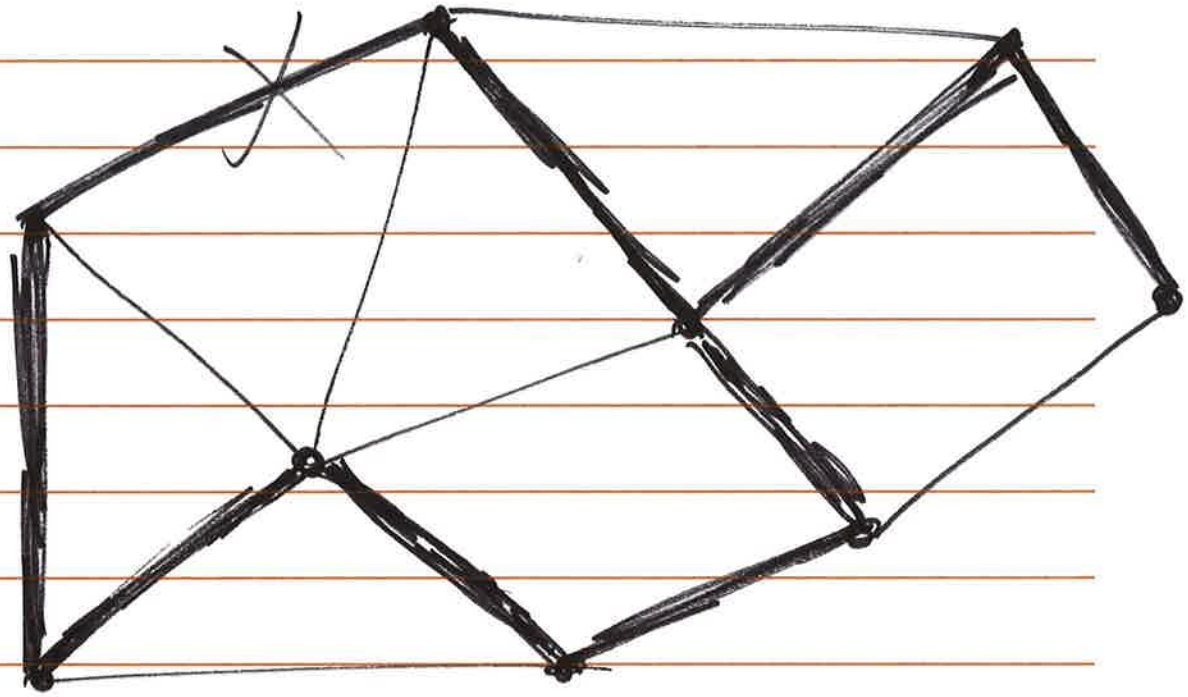
while loop goes n times

n - extract_Min ops

n^2 m Decrease-key ops.

Cost of Dijkstra's	binary heap	binomial heap	Fib. Heap
n-Extract_Min	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$
m-Decrease-key	$O(m \lg n)$	$O(m \lg n)$	$O(m)$
total	$O(m \lg n)$	$O(m \lg n)$	$O(n \lg n + m)$
Sparse graph	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$
Dense graph	$O(n^2 \lg n)$	$O(n^2 \lg n)$	$O(n^2)$


n2 log n
Bin. Heaps
n2
Fib.



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

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

Find a MST in an undirected graph.

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.

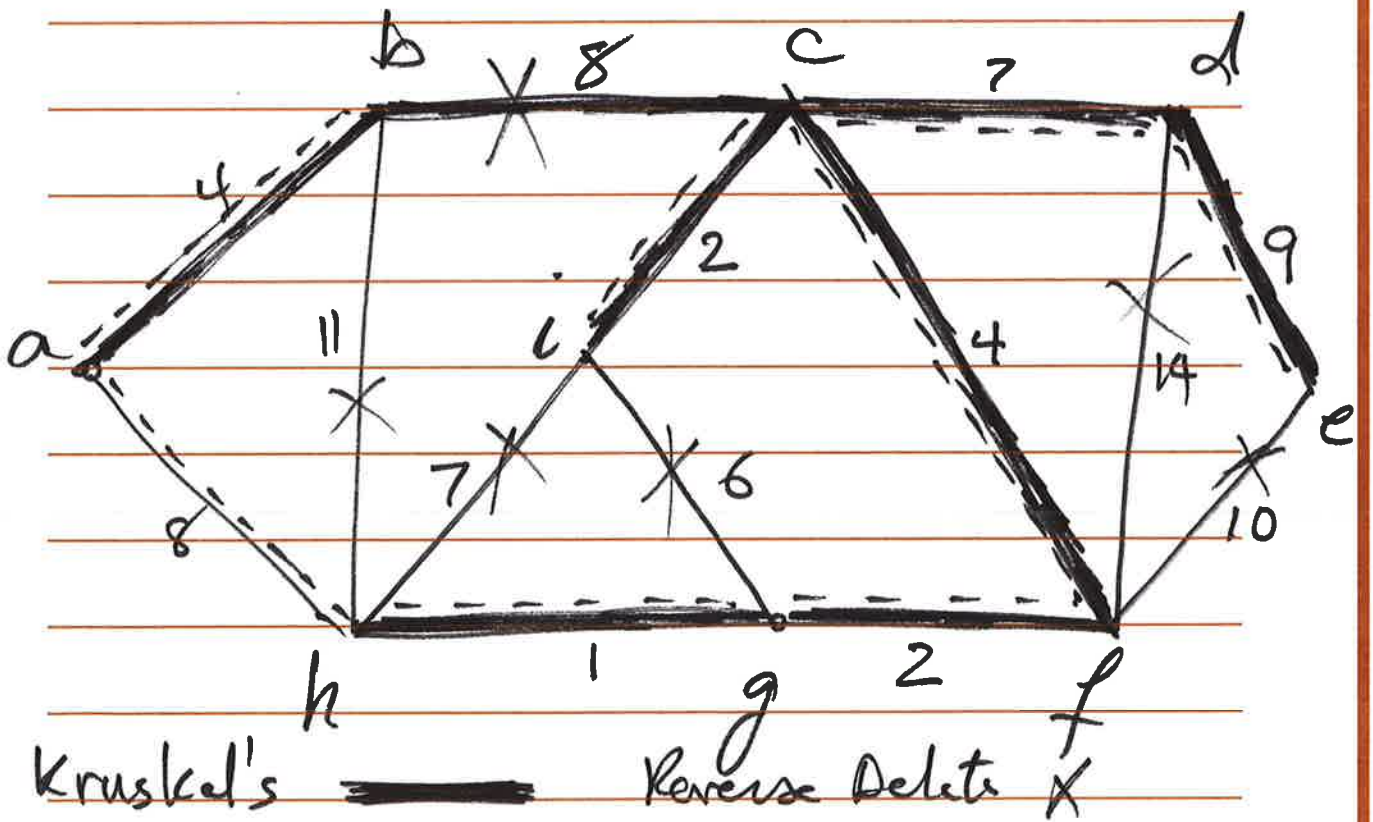
Kruskal's Alg.

2- Backwards version of Kruskal's
Start w/ a full graph (V, E)
begin deleting edges in order of decreasing cost as long as it does not disconnect the graph

Reverse Delete

3. Similar to Dijkstra's Alg., start w/ a node set S (initially the root nodes) on which a min. spanning tree has been constructed so far. At each step grow S by one node, adding the node v that minimizes attachment cost.

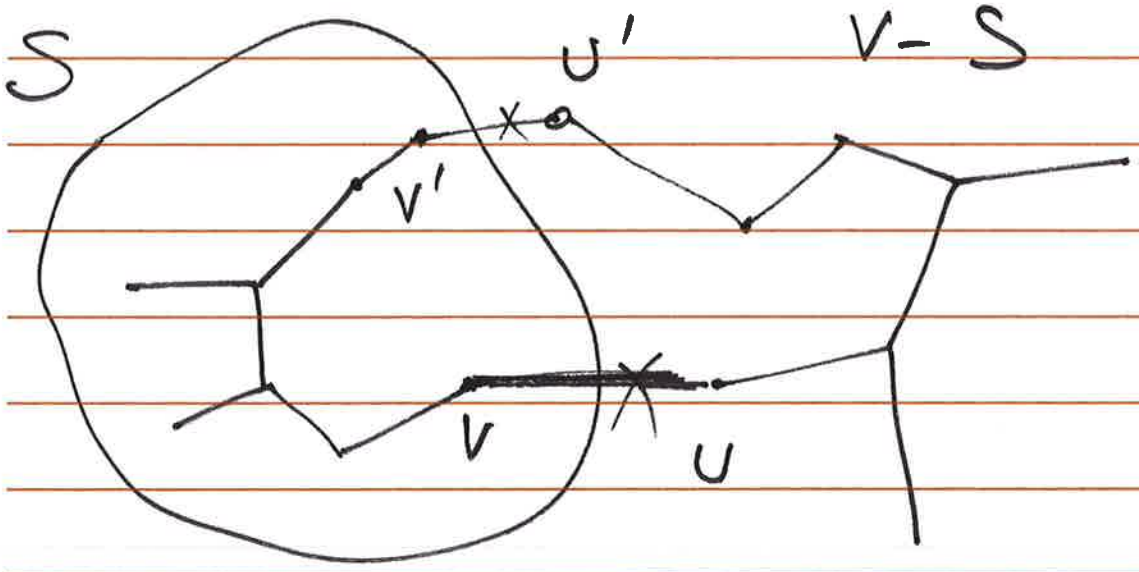
Prim's Alg.



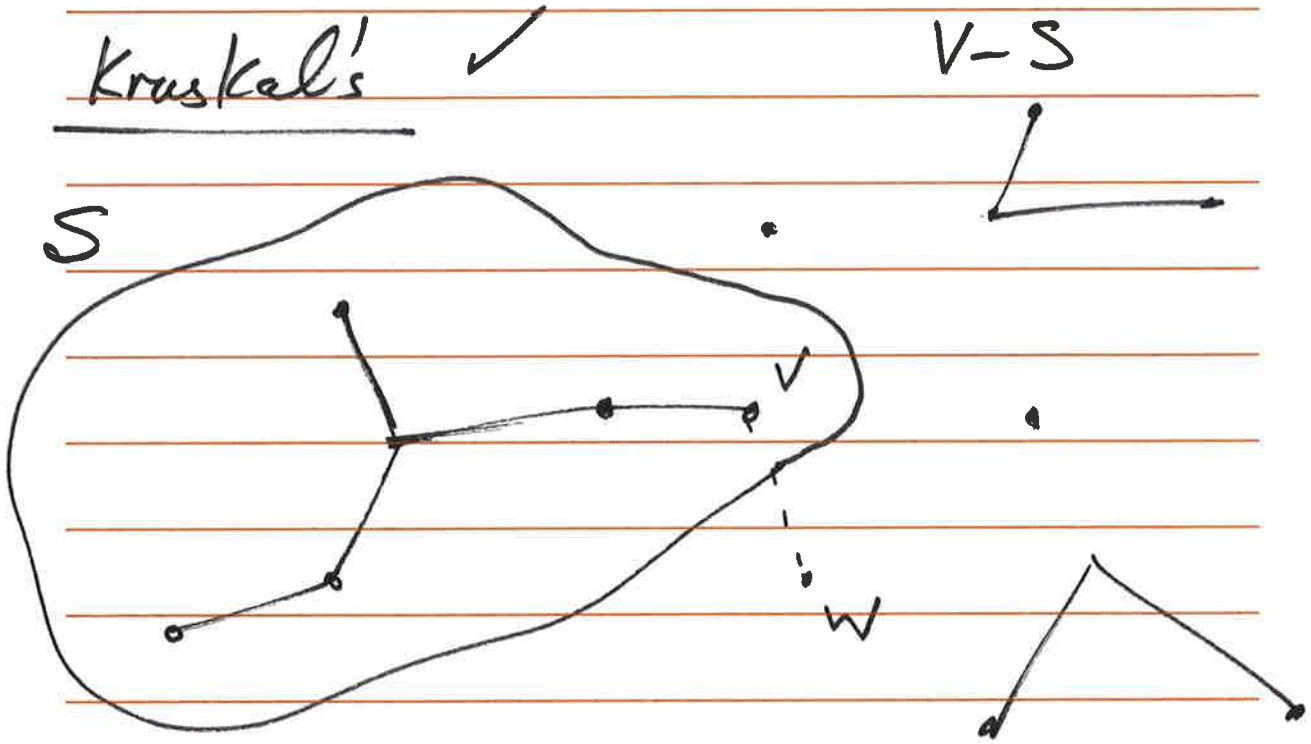
Kruskal's ——— Reverse Delete X

Prim's - - - -

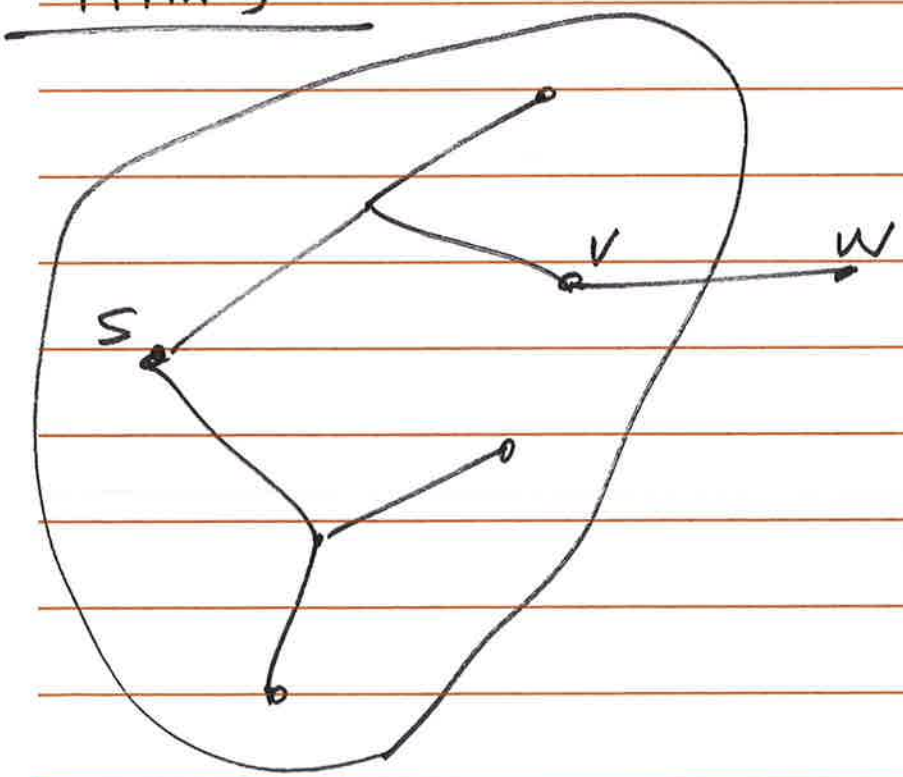
FACT: Let S be any subset of nodes that is neither empty nor equal to all of V, and let edge $e = (v, w)$ be the min. cost edge with one end in S and the other end in $V - S$. Then every MST contains the edge e.

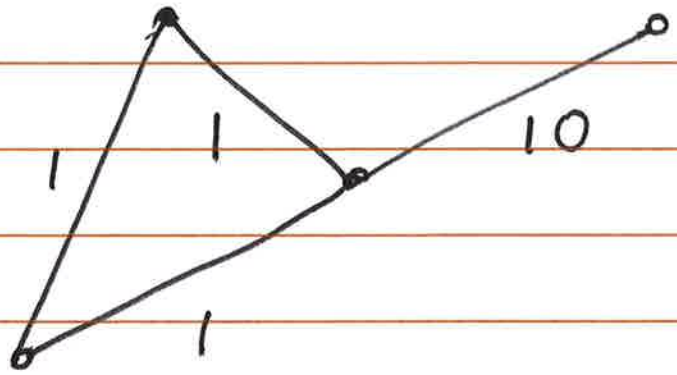


Kruskal's ✓



Prim's





FACT: The highest cost edge in a cycle cannot be part of any MST.

~~RR~~ Prim's

the only difference in the implementation compared to Dijkstra's is the relaxation step:

if $d(v) > le$
    decreasekey(Q, v, le)

Same complexity as Dijkstra's

using a binary heap ~~—~~ $\Rightarrow O(m \lg n)$

Kruskal's

Create an indep set for each node
$A = \text{Null}$
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

endif
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 imp. ptr imp.

```
A = Null
for each vertex v ∈ V
    Make-set(v)                          } O(n)
endfor
Sort the edges of E into non-decreasing   O(m lg m)
        order of weight
for each edge (u,v) ∈ E in this order     O(m)
    if Find-set(u) ≠ Find-set(v)
        A = A ∪ {(u,v)}
        Union(u,v)
    endif
endfor
```

Overall Complexity

$$O(n) + O(m \lg m) + O(m \lg n)$$

$$= O(m \lg m)$$

$\rightarrow O(m \lg n^2)$

$O(2m \lg n)$

$O(m \lg n)$

Reverse-delete

Sort takes $O(m \lg m)$

loop m times

$O(m+n)$ or $O(m)$ (Check if removing the next lower cost edge is going to disconnect the graph or not.

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