Date: 07/06/21

Black Board

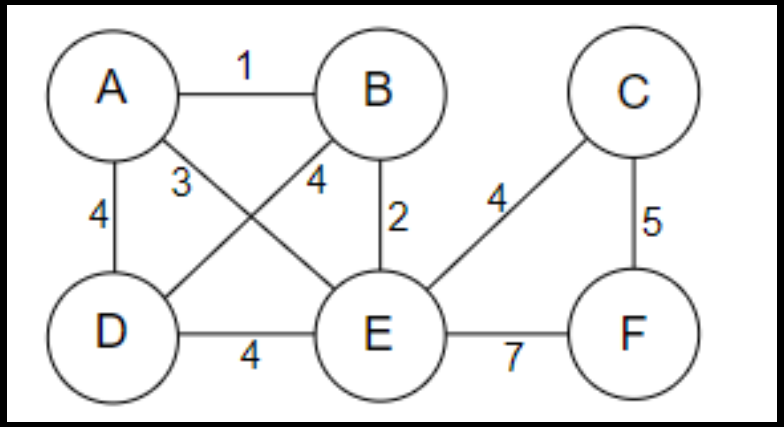
Design and Analysis of Algorithms

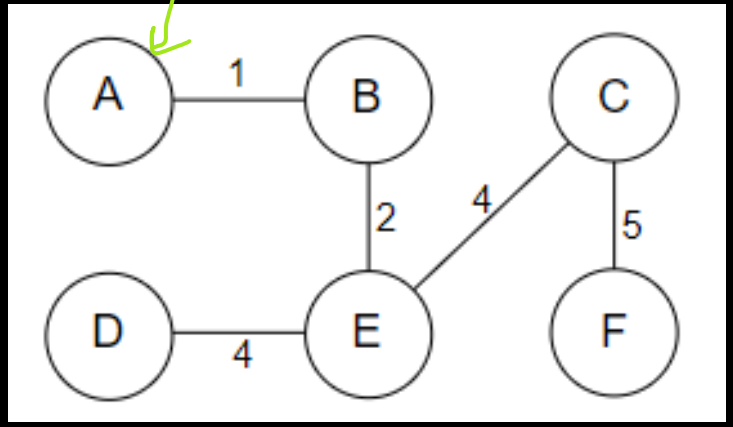
Topics:

- Minimum Spanning Trees
  - Prim's Algorithm
  - Kruskal's Algorithm

What is a Minimum Spanning Tree (MST)?

Tree + Spanning + Minimum Cost





✓ **An undirected weighted graph G**

An MST of G

$$Cost(T=(V,E')) = \sum_{e \in E'} w(e)$$

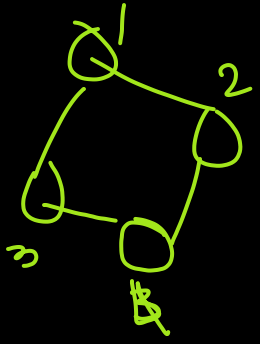
16

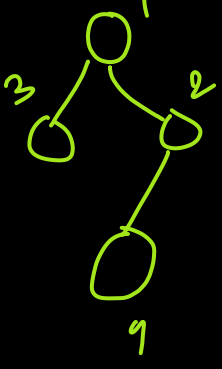
Trees & Spanning Trees of Undirected Graphs

Tree: a subgraph of an undirected graph $G=(V,E)$

(unrooted) $T=(V',E')$

that has no cycles.



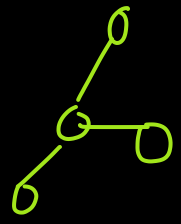


Spanning tree: $T=(V',E')$

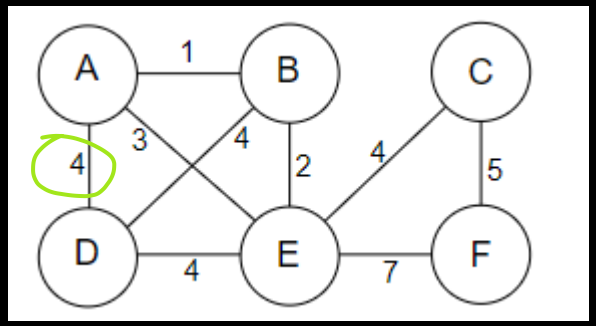
$V'=V$

all vertices are spanned by the tree.

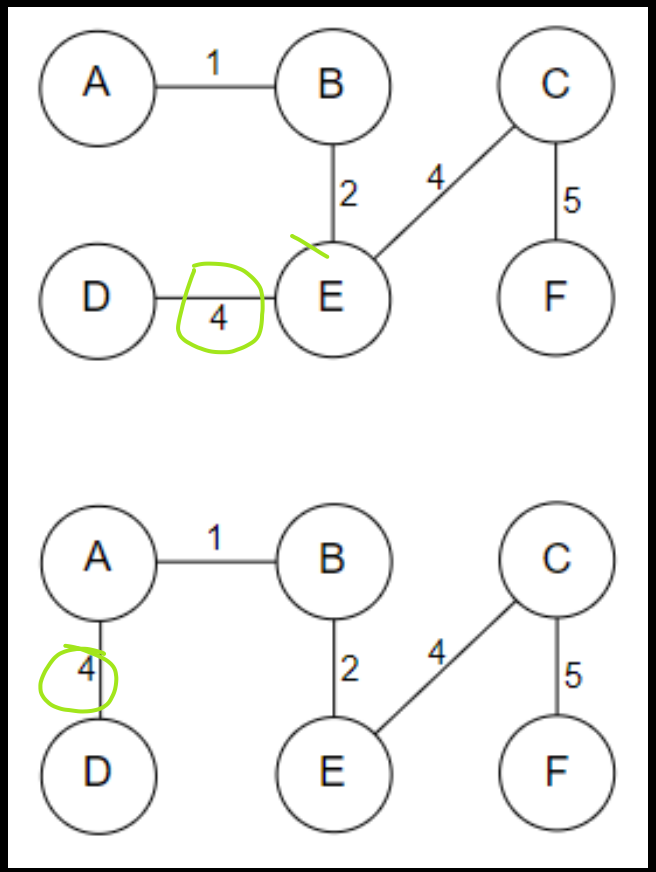
$|E'|=|V|-1$



MSTs need not be unique



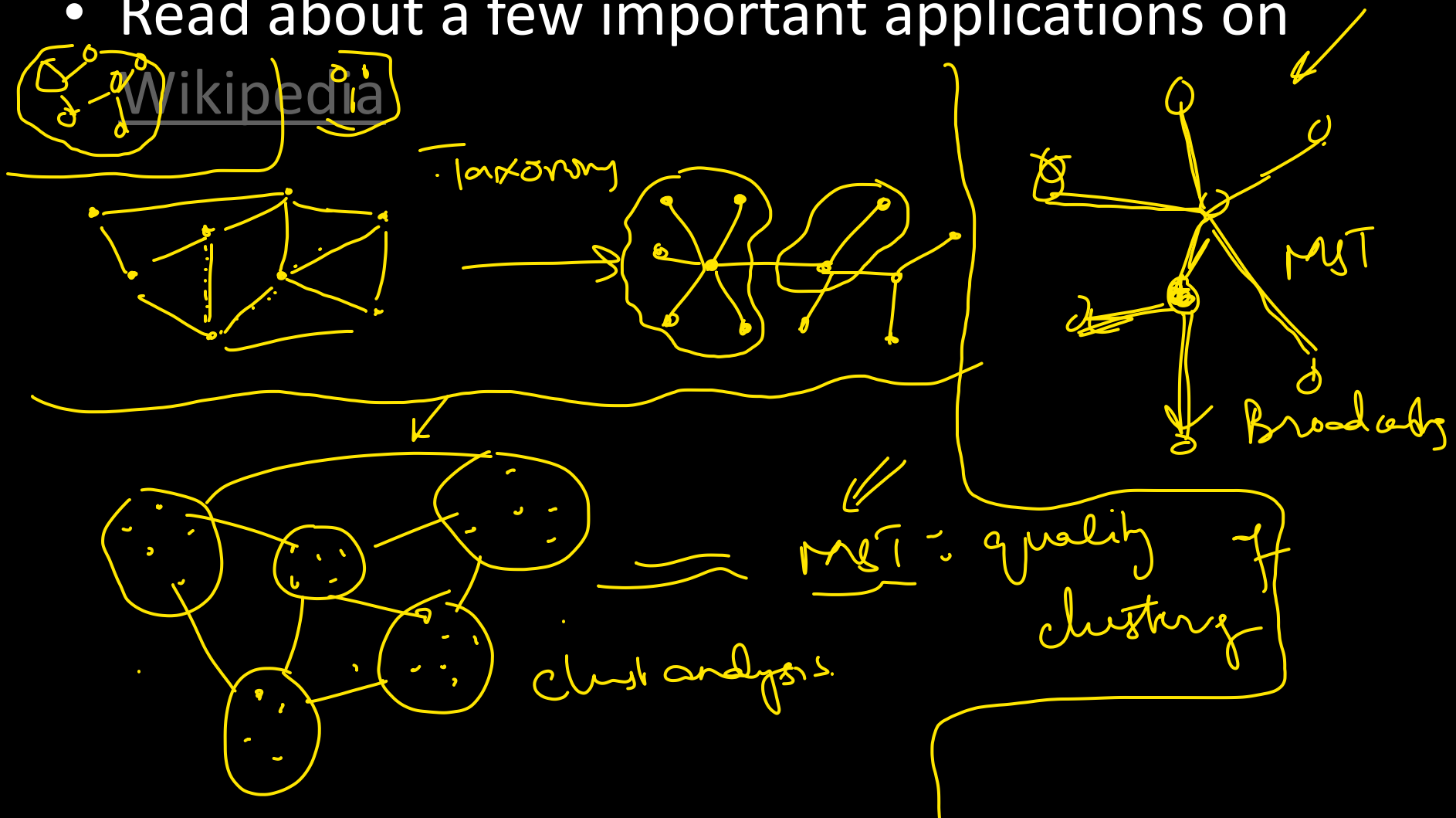
An undirected weighted graph G



Two different MSTs of G

Applications of MST

- Read about a few important applications on Wikipedia



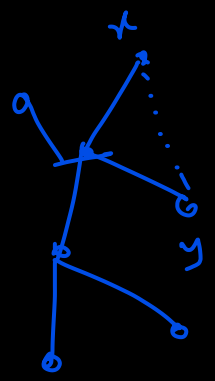
The two most famous algorithms for MSTs are both ***greedy***

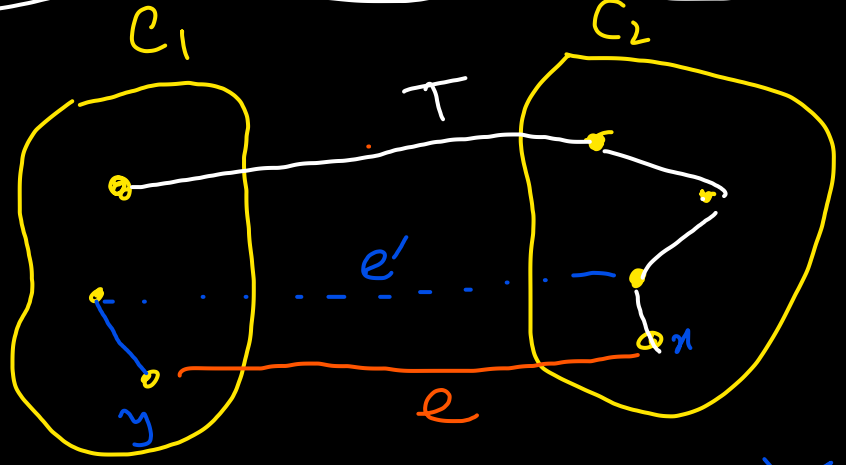
- Prim's Algorithm
- Kruskal's Algorithm
  - Both rely on something called the **Cut Property**

The Cut Property

Version 1 (Used by Prim's Algorithm)

"Any 'cut' of nodes V must be connected by the lightest crossing edge in an MST"



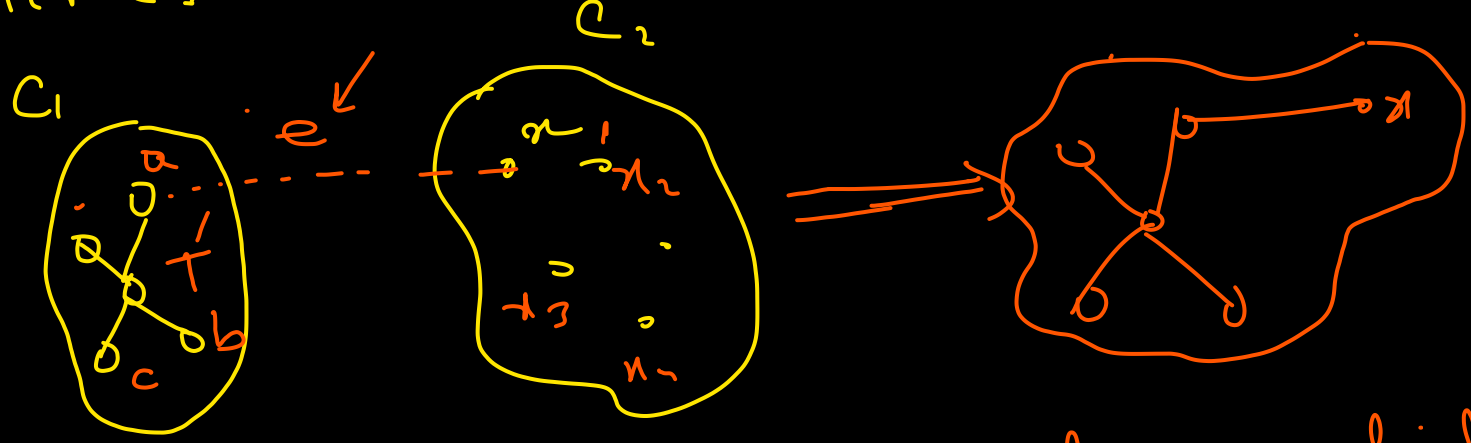


$w(e) < w(e')$

Assume: T is MST and it does not include the lightest crossing edge (e)

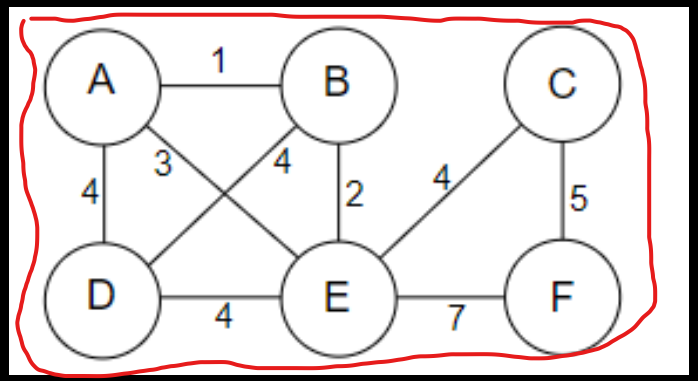
The main idea of Prim's algorithm

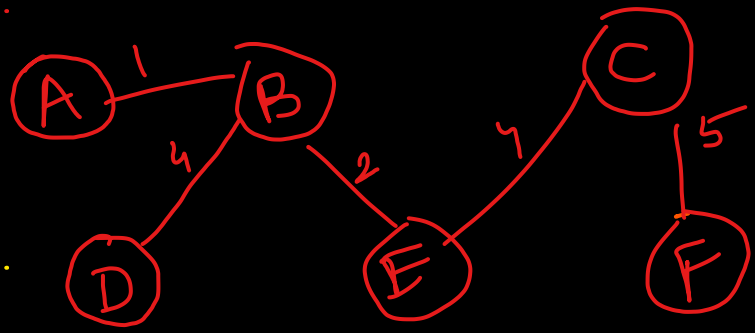
→ Grow the MST one node at a time:

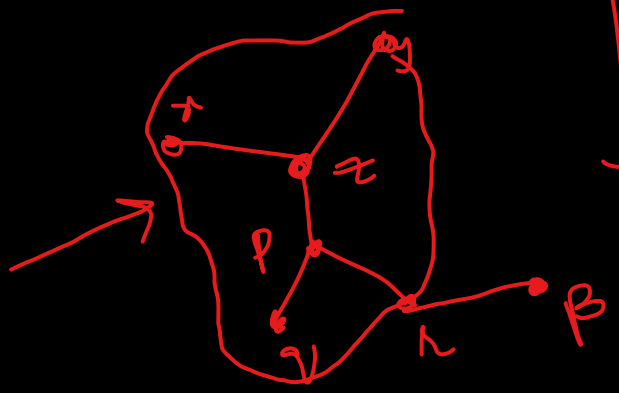


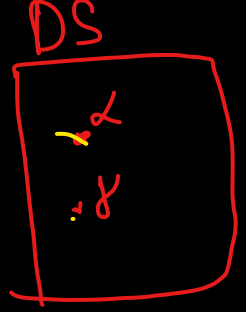
→ each time add the node x, which is connected to the growing tree through the lightest edge e.

The working of Prim's Algorithm (A Bird's Eye View)





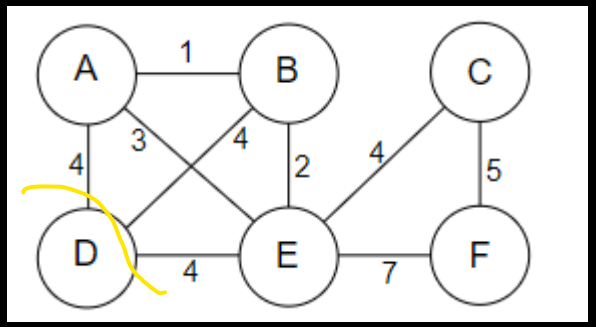




⇒ Min Heap maintaining the lightest connection cost of every vertex in CZ (non-tree nodes)

⇒ Each getMin will give near node

Prim's Algorithm Precise Working



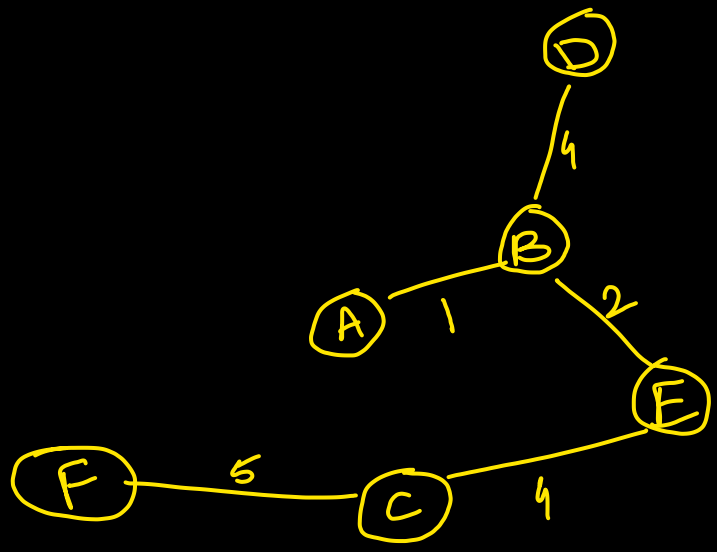
Original Graph

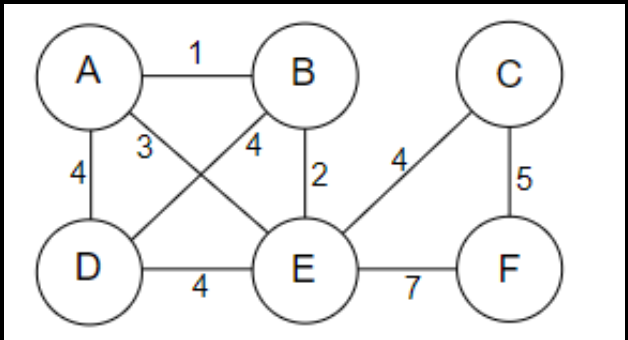
Iteration 0

h

A 1/B	D 0
B 4/D	E 2/B
C 4/E	E 5/E

Iteration 1





Original Graph

Prim's Algorithm Code

①

```
PrimMST(G=(V,E), w)   // V = {0,1,2,...,n}   ②
  Create cc[|V|], cn[|V|], T                  O(1)
  For each x in V:                            O(|V|)
      cc[x] = ∞
      cn[x] = nil
  cc[0] = 0                                   O(1)
  Create MinHeap h(V, cc)                     O(|V|)
  WHILE !h.empty()                            → O(|V|)
      x = h.extractMin()                      → O(|V| lg |V|)
      T.insert(x)                             → O(|V|)
      For each (x,y) ∈ E:                     → O(|E|)
          IF (!T.exists(y) && cc[y] > w(x,y))
              h.decreaseKey(y, w(x,y))        → O(|E| lg |V|)
              cc[y] = w(x,y)
              cn[y] = x
```

$$T(G=(V,E)) = O((|V|+|E|)\lg|V|)$$

Prim's Algo.

improve to

$O(|V|\lg|V| + |E|)$ average case

through Fibonacci Heap

Prim’s Algorithm Complexity