Updating values in a heap

Dijkstra's algorithm - heap contains Distance[i]

Vertices   Heap (Dist[j])
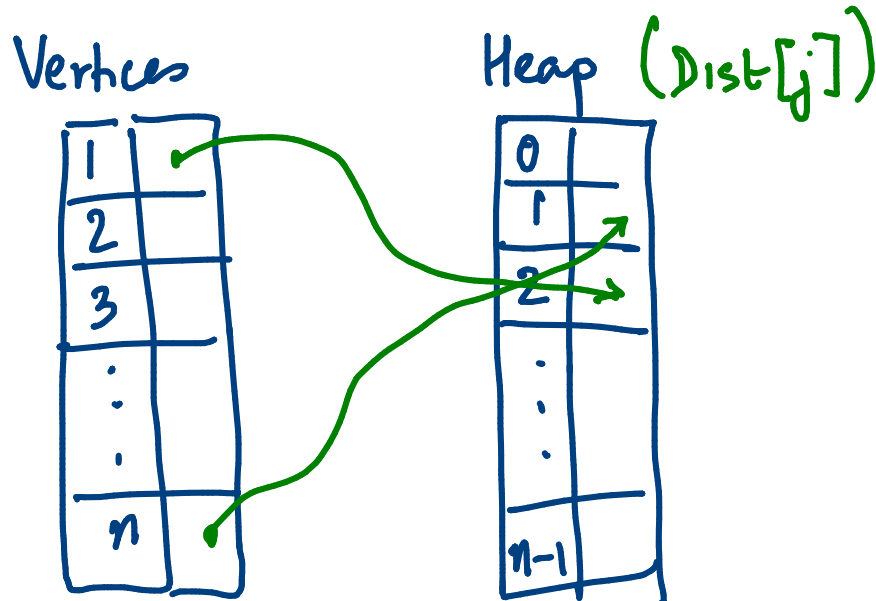
H to V is $(V$ to $H)^{-1}$

Update Dist[7]   Find position in heap
                                          V to H

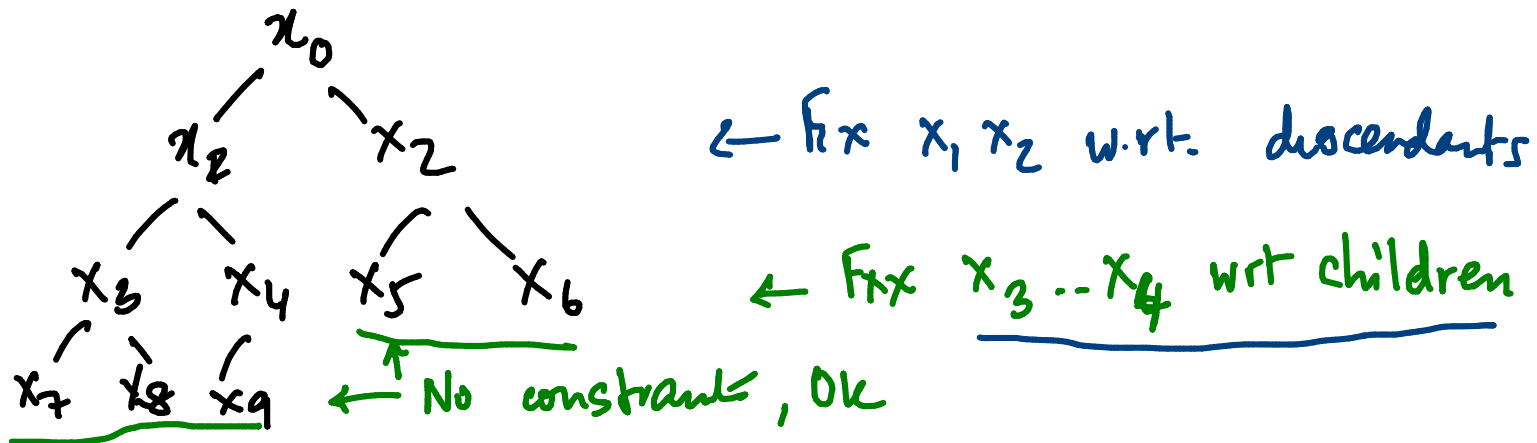When swapping in heap, need to go back via H to V

Building a heap

List with n values $\rightsquigarrow$ heap

Start with empty heap, do n inserts

$O(n \log n)$

Smarter way $\rightarrow$ "bottom up" $\qquad L = [x_0, x_1 .. , x_{n-1}]$

$\leftarrow$ Fix $x_1 \, x_2$ w.r.t. descendants

$\leftarrow$ Fix $x_3 .. x_4$ wrt children

$\leftarrow$ No constraint, Ok

leaf level $(n/2$ nodes)   0 swaps

One level above $(n/4$ ")   1 swap

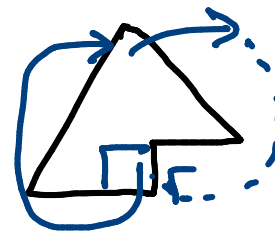Two levels " $(n/8$ ..)   2 swaps

Count carefully — $O(n)$

Heap Sort

  Build a heap $O(n) / O(n \log n)$ ⎤
  Delete - max n times $O(n \log n)$ ⎦ $O(n \log n)$

  Like selection sort, in place

    → Ascending order

Move max value to end of current heap

Internet Service Provider
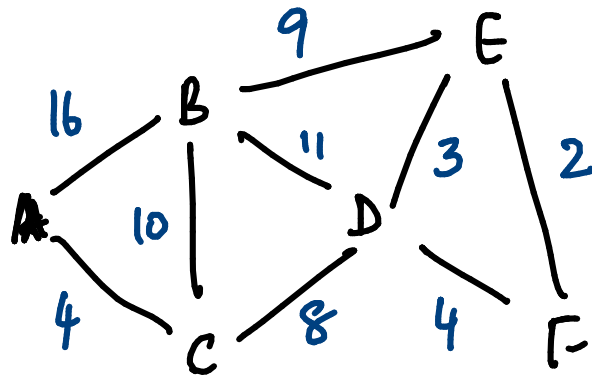
Replace some links by
high speed fibre so
that all nodes are
connected by high speed
paths

Tree — min connected graph

Should "span" all nodes

Min Cost Upgrade

sum of
edge
weights↓
in
tree

Subtree of given graph

SPANNING TREE of G
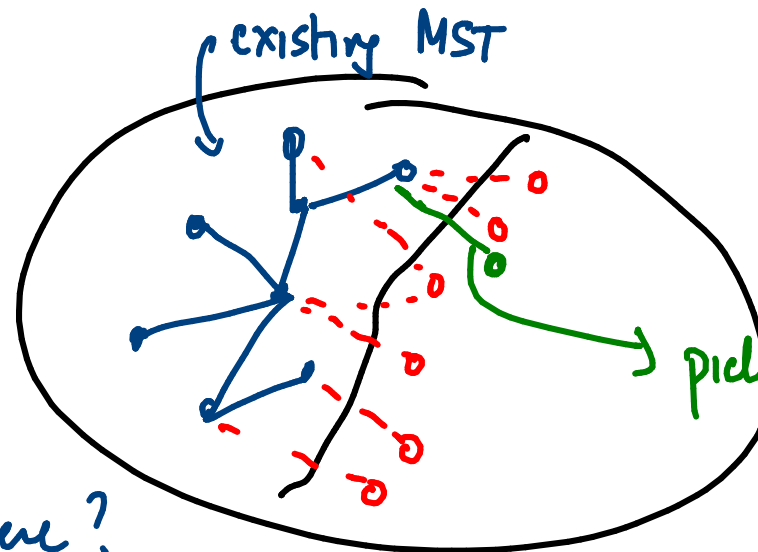
MINIMUM COST SPANNING TREE   (MCST)

or MST

Greedy Strategy

Dijksta variant

PRIM'S ALGORITHM

Assume we have an MST over k vertics
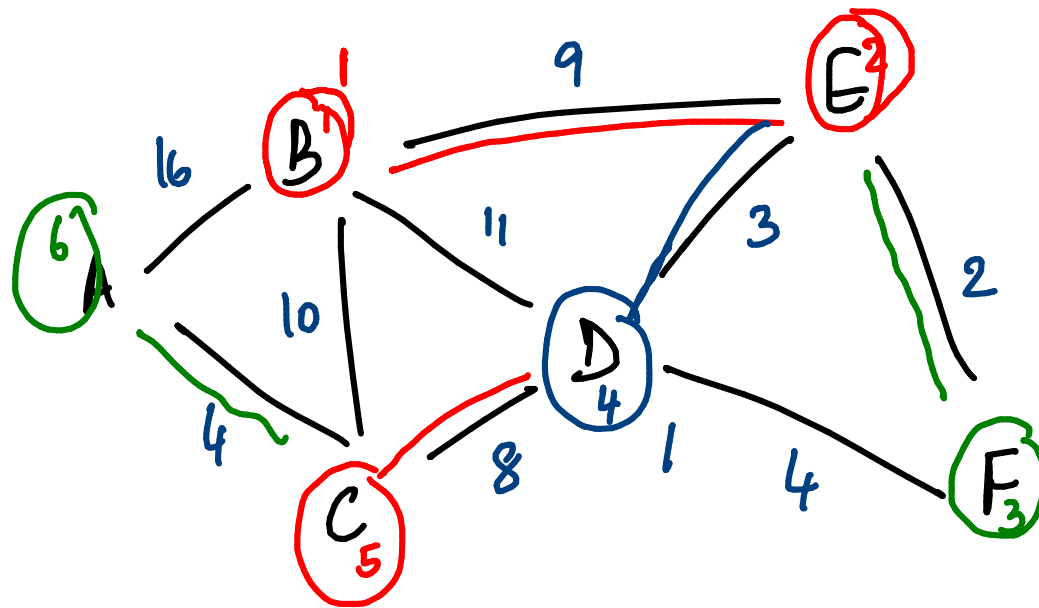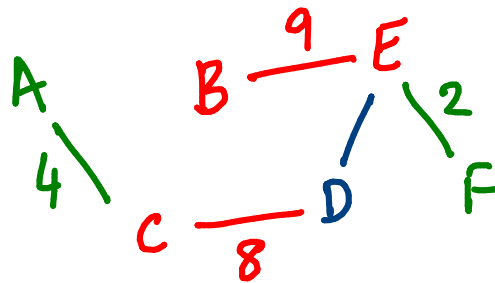

existing MST

→ pick smallest edge from current tree going out, add it to tree

Proof?

Start anywhere?

Why is this minimal?

Start at B

Correctness?

Assume distinct edge weights
(and non negative)

Separation Lemma $V \begin{cases} S \\ V-S \end{cases}$

MST not
unique

Add e to T
~cycle
Current path
$V \to V'$ crosses
S to V-S
Drop that edge — smaller spanning tree!

Spanning
Tree
T

S   V-S

e

v   v'
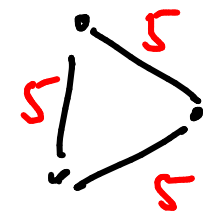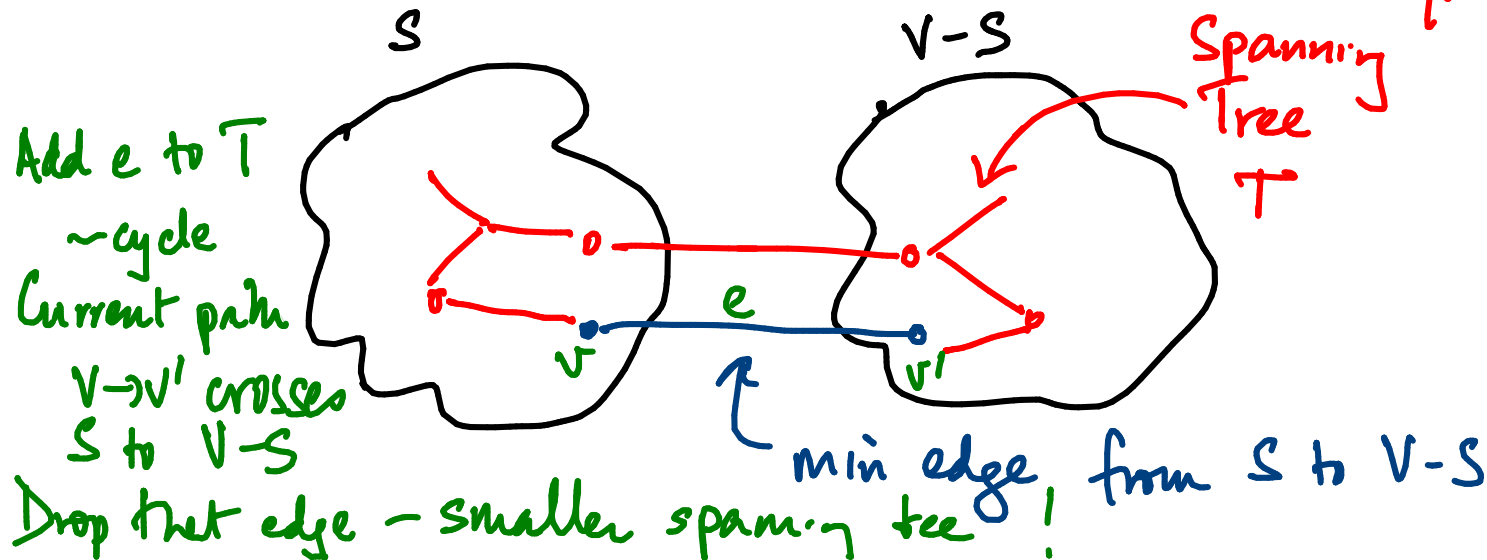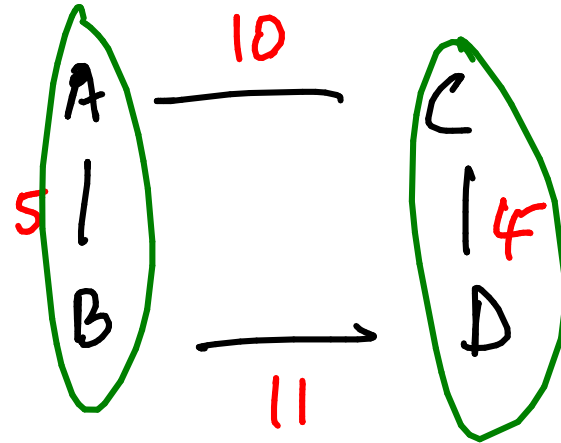
Min edge from S to V-S

Separator Lemma

Given a partition of V as $S, V \setminus S$,
min cost edge from S to $V \setminus S$ must
belong to every MST

Separator lemma does not replace algorithm

Another greedy strategy

Start with smallest edge overall
:
Add k^{th} smallest edge if it does not
form a cycle

Stop after $n-1$ edges

KRUSKAL'S
ALGORITHM

Sort edges in ascending order

In between — collection of connected components

Implementing Kruskal

Keep track of components

Components partition V

    — check for cycle $e = (i,j)$

       — Are i, j in same component?

    — add e — merge components