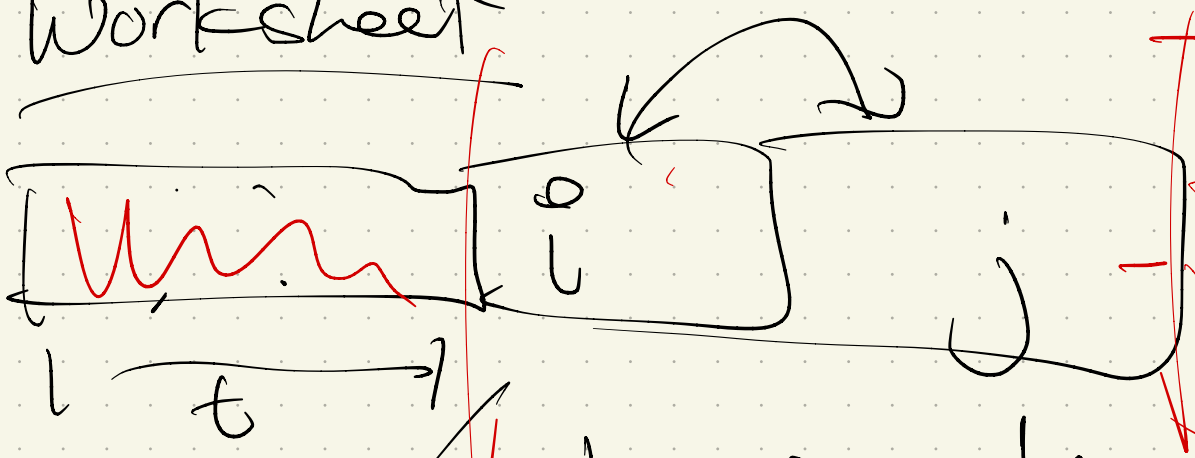
Transition: DS + Algorithms

MSTs

Worksheet



+ j now finishes earlier

i finishes later, but before j used to.

$d_i > d_j$

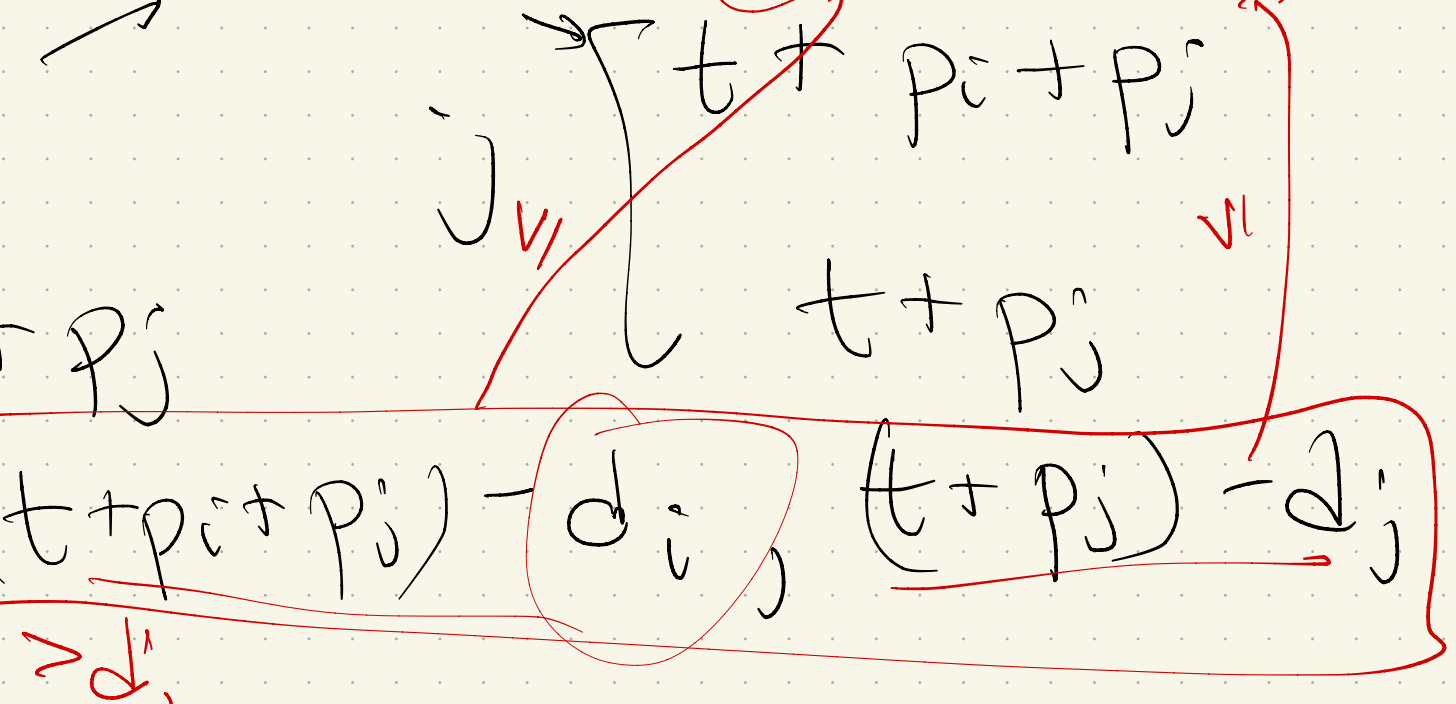
$\max\{(t+p_i)-d_i, (t+p_i+p_j)-d_j\}$

before:

$t + p_i$

i after:

$t + p_i + p_j$



Minimum Spanning Trees

Goal: Given a weighted Graph G, $w: E \to \mathbb{R}^+$ the weigh function, find a spanning tree T of G that minimizes

$$w(T) = \sum_{e \in T} w(e)$$

every node to have a path to every other node

in MST

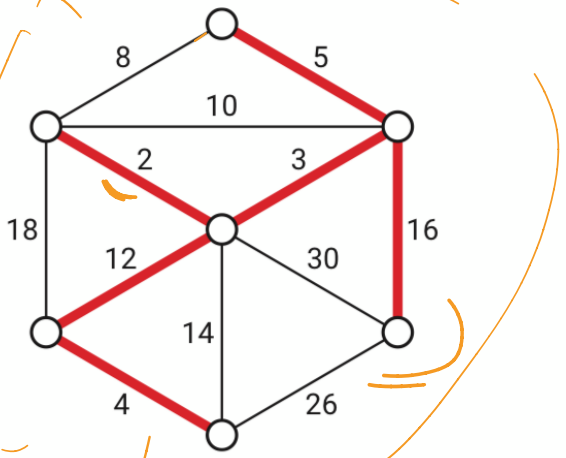
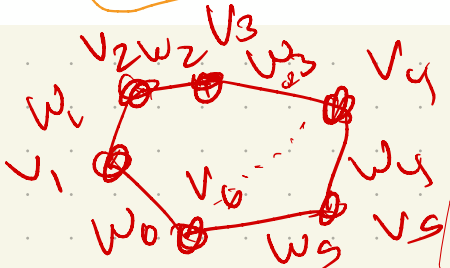
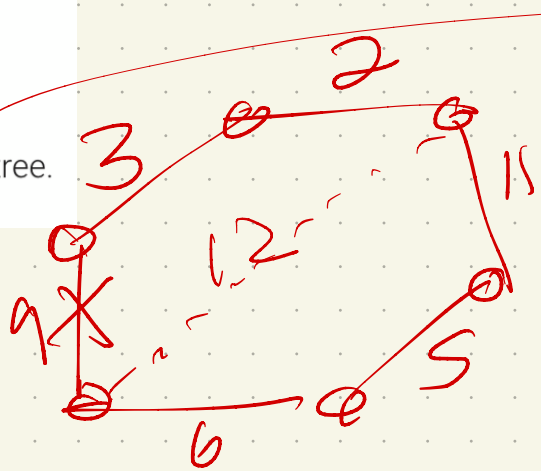


Figure 7.1. A weighted graph and its minimum spanning tree.

Why a tree?





The magic truth of MSTs:

You can be SUPER greedy!

Almost any natural idea will work!

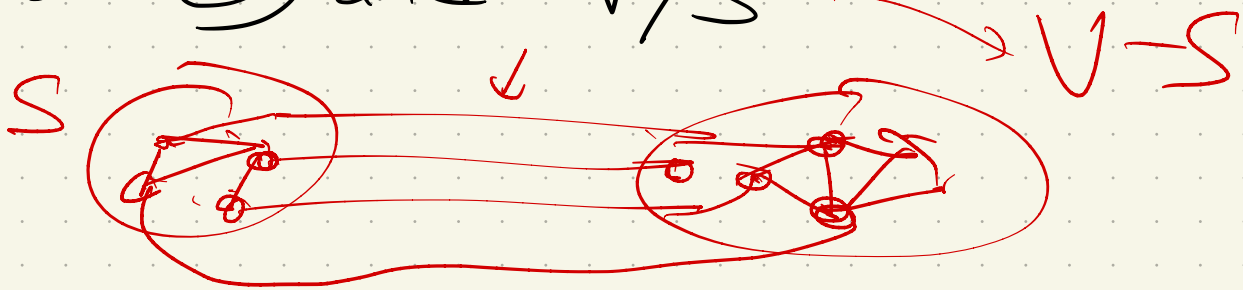
This is highly unusual, & there's a reason for it:

these are a (rare) example of something called a matroid

(Way beyond this class...)

Key property:

Consider breaking G into two sets: S and V/S



The MST will always contain the lowest edge connecting the two sides.

No matter what!

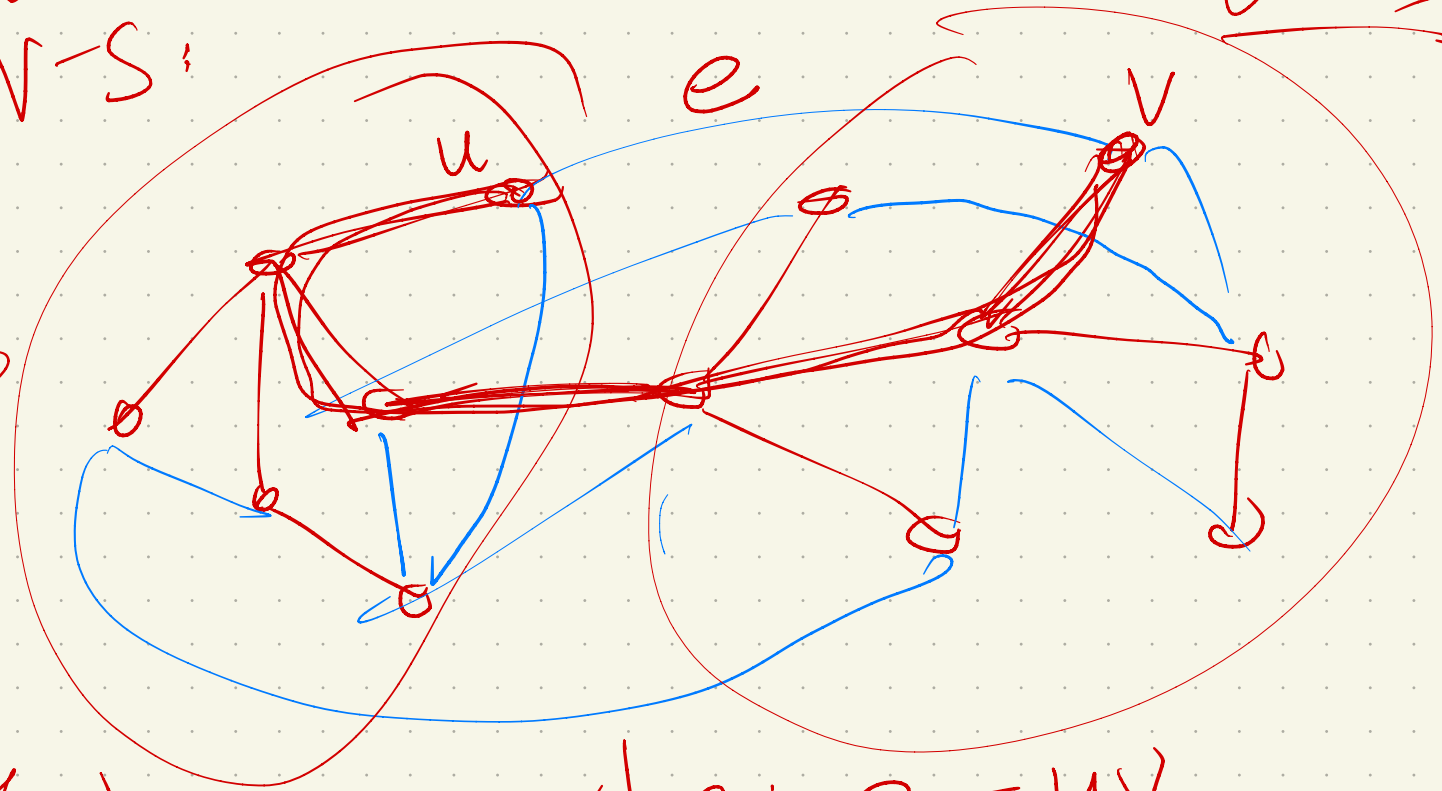
min edge of ~~of~~ V is in MST



No matter what set S of VI pick lowest edge out of S is in MST

Proof (by contradiction) Suppose not:

Consider MST, which does not have min edge from S to V−S:



MST edges

other edges in G not in MST

take that min edge: $e = uv$

MST has some u-v path, T

Note: T+e is a cycle. → has some other S to V−S edge

Generic Algorithm:

Build a forest: an acyclic subgraph

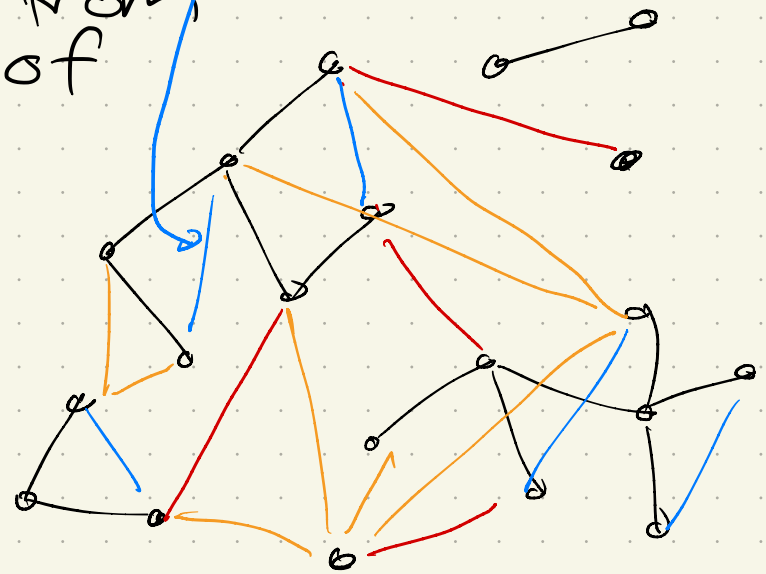
Black edges: MST so far

Dfn: An edge is useless if it connects 2 endpts in same component of F

also edges that are neither.

An edge is safe if it is minimum edge from some component of F to another

F:



Algorithm:

Find minimum edges & grow a forest:

loop until all vertices are connected:

identify safe edges

add them

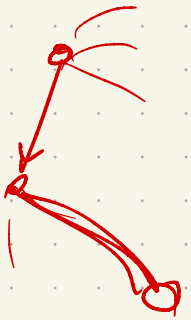
DFS: O(V+E)

We'll see 3 ways:

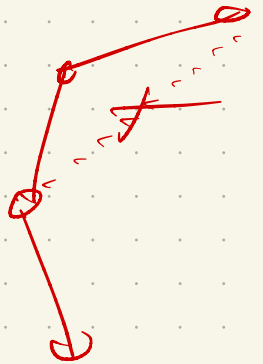
(1) Find <u>all</u> safe edges.
Add them + recurse.



(2) Keep a single connected component.
At each iteration, add 1 safe edge.



(3) Sort edges + loop through them.
If edge is safe, add it.



First one: (1926-ish)

BORŮVKA: Add ***ALL*** the safe edges and recurse.

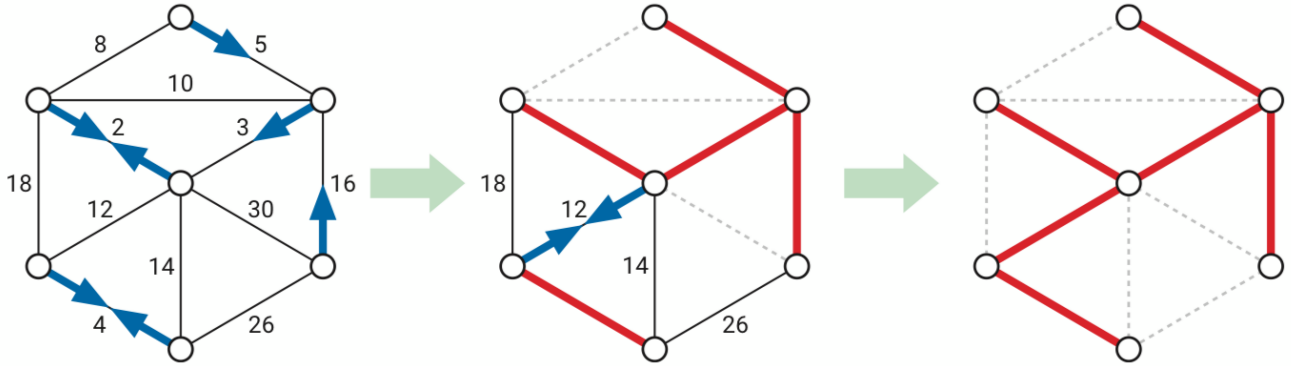


Figure 7.3. Borůvka's algorithm run on the example graph. Thick red edges are in F; dashed edges are useless. Arrows point along each component's safe edge. The algorithm ends after just two iterations.

So we need to: continually remove min. thing → PQ

While more than 1 component:

- Track components ← hard code
- Find all safe edges ← loop through edges & find min
- Add them ←

Prim's algorithm:

(really Jarník, we think)

Keep one spanning sub tree.

While $|T| \neq V$:

add next safe edge

heap / PQ to get next smallest edge

JARNÍK: Repeatedly add T's safe edge to T.

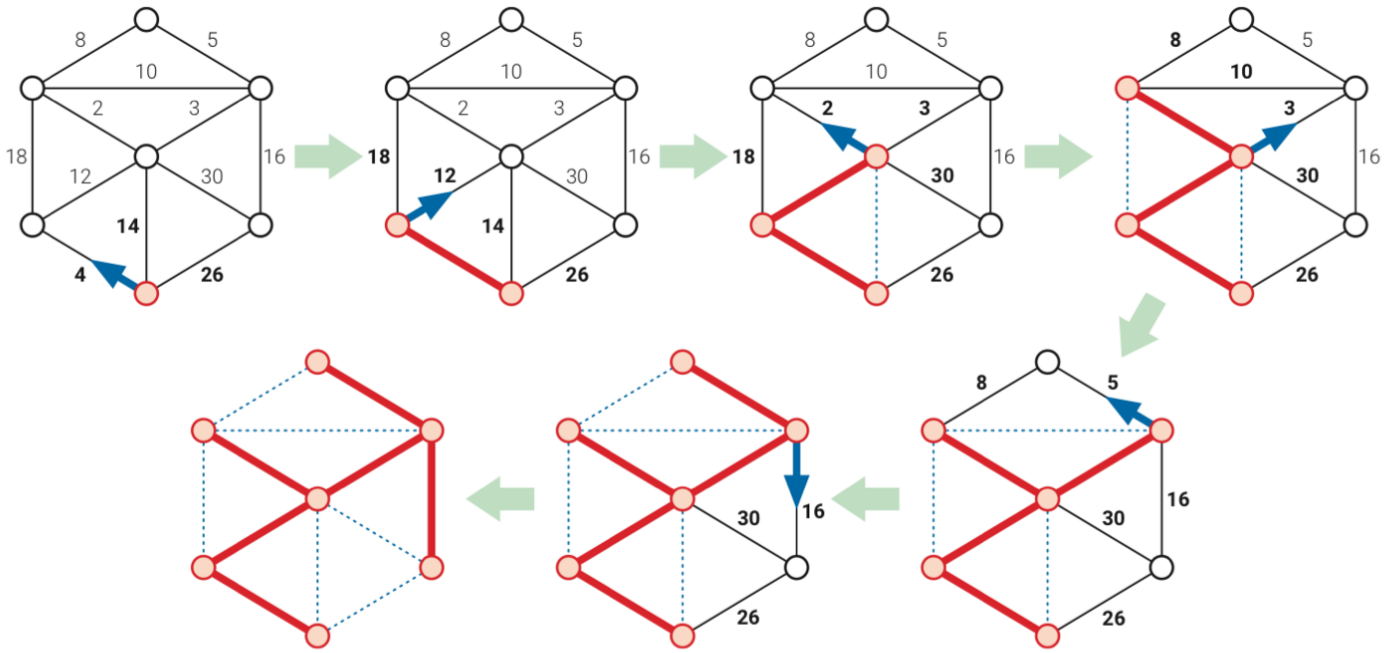


Figure 7.4. Jarník's algorithm run on the example graph, starting with the bottom vertex. At each stage, thick red edges are in T, an arrow points along T's safe edge; and dashed edges are useless.

Kruskal's Algorithm:

sort : $E \log E$

KRUSKAL: Scan all edges by increasing weight; if an edge is safe, add it to F.

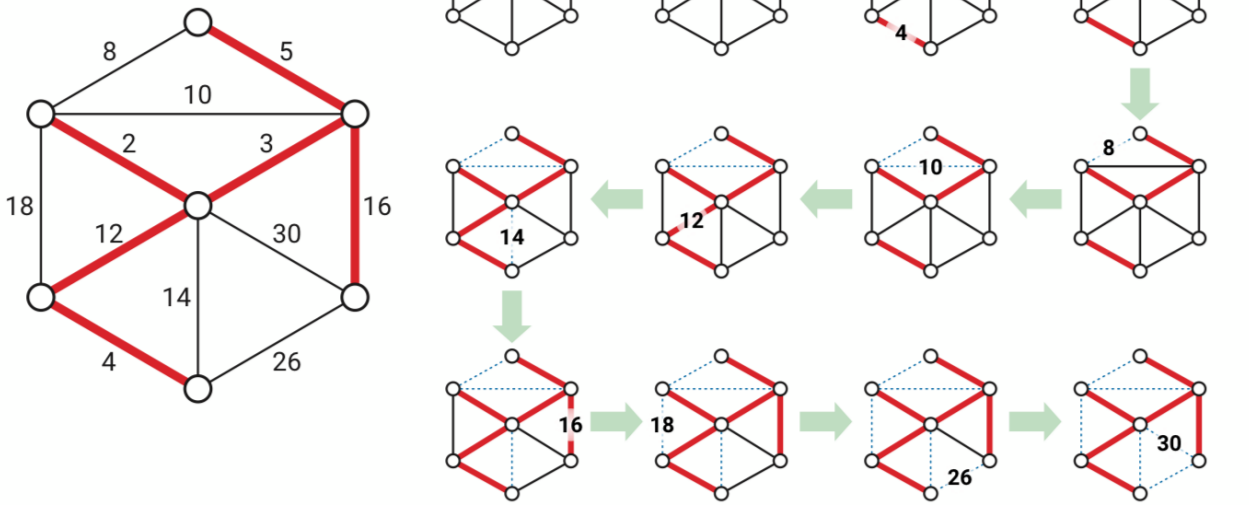


Figure 7.6. Kruskal's algorithm run on the example graph. Thick red edges are in F; thin dashed edges are useless.