Minimum Spanning Tree

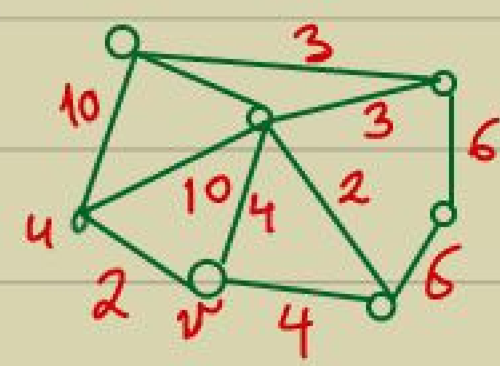
Jarnik (1930) & Prim (1957)

– Defn: For a graph $G=(V,E)$ a Spanning tree $T=(V,E')$ is a subgraph covering all the vertices V & is cycle-free.

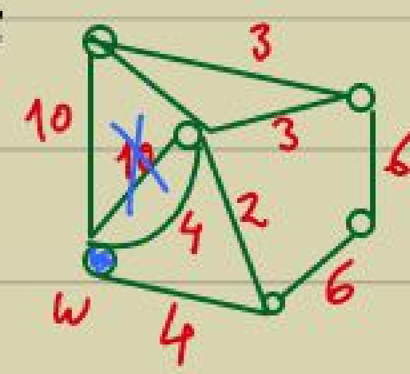
How's G given? List or Matrix?

If E is weighted then we can ask for min. weighted T. (mst)

G:



G':



Lemma 1: $\exists$ MST with the min. edge (u,v).

Proof:

- Let T be an MST of G without $(u,v) =: e$.
- Adding e in T creates a cycle C.
- We can remove the cycle by deleting an edge $e' \in C$ from T s.t. $e' \neq e$.
- Since $wt(e') \geq wt(e)$ the wt. cannot increase & we still have an MST of G.

$\square$

- The lemma motivates the following transformation on G to get G':
  - Let $e=(u,v) \in E$ be a min. wt. edge in G.
  - Remove u & v & add a new vertex w in G'.
  - For each $(u,x) \in E$ add (w,x) in G'. (with same wt.)
  - " " (v,x) " " " ". "
  - In case of multiple (w,x)'s keep the least weighted one in G'.

Lemma 2: $wt.MST(G') = wt.MST(G) - wt(e)$.

Proof:
- $MST(G') \cup \{e\}$ is a spanning tree of G $\Rightarrow wt.MST(G) \leq wt.MST(G') + wt(e)$.
- Any $MST(G)$ containing e, gives a spanning tree of G' $\Rightarrow wt.MST(G') \leq wt.MST(G) - wt(e)$.

$\square$

Complexity:
- We keep the edges in an AVL tree according to the weight. $O(m \lg m)$
- On deleting $e=(u,v)$ we make $deg(u)+deg(v)$ many tree operations.

(Why?) $\Rightarrow$ Overall it takes $O(\sum_{u \in V} deg(u) \cdot \lg m) = O(m \lg n)$ time.

- Using an advanced data structure (Fibonacci heap) it can be done in $O(m+n\lg n)$ time.

- Fast algorithms require the graph to be given as adjacency list.

[Kruskal '56]

- Pick the min. wt. edge e_1.
- Pick next min. wt. edge e_2.
- " " " " " e_3, s.t. no cycle gets formed.

  .... & so on.

Exercise: This also gives a fast algorithm for mst.

Shortest Paths   Dijkstra (1956)

- $G=(V,E)$ is a directed graph with n vertices V & m edges E.
  The edge weight is given by $w: E \to \mathbb{R}_+$.
  The edges may be given as an adjacency matrix A_G or an adjacency list.

- A path p, from s to t, is a sequence $s=v_1, v_2, \dots, v_{k-1}, v_k=t$ s.t. $\forall i, (v_i, v_{i+1}) \in E$

- The length, or weight, of p is:
  $$\sum_{e \in p} w(e).$$

- Output: A shortest path $P(s,t)$ from s to t.

- Distance from s to t is the length of the shortest path p; denoted as $\delta(s,t)$.

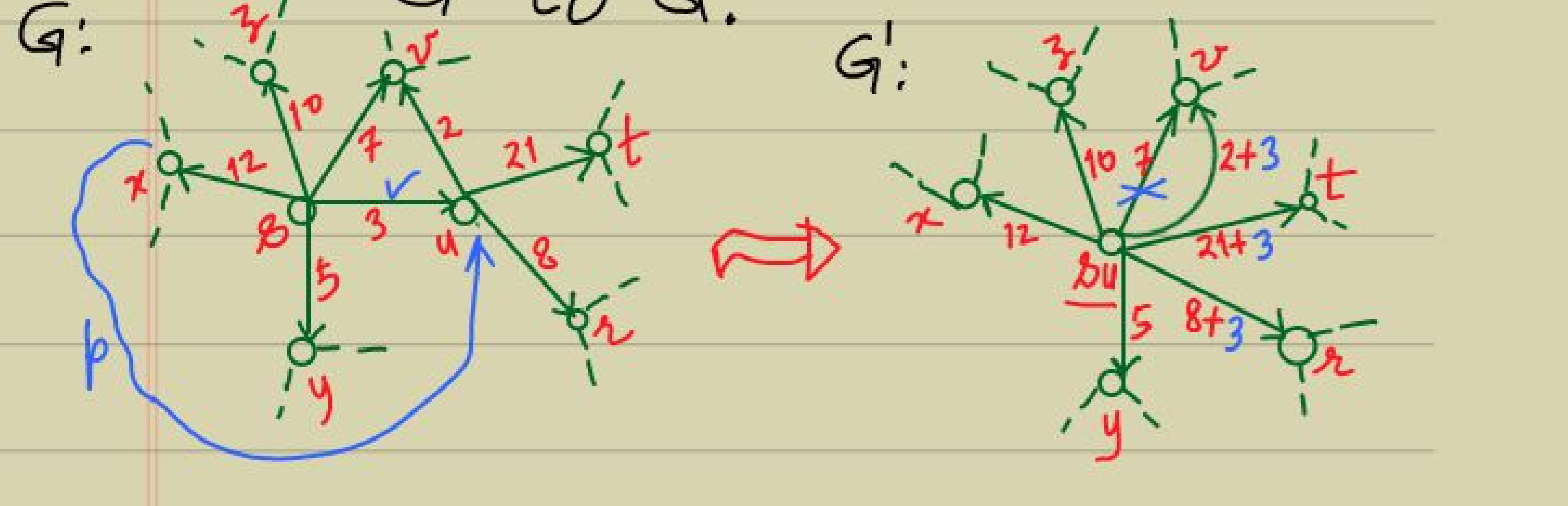
- We will study a slightly general problem:

Input: A directed graph $G=(V,E)$, wt. $w: E \to \mathbb{R}_+$ & a source vertex $s \in V$.

Output: $\forall v \in V$, compute $\delta(s,v)$ & $P(s,v)$.

- The applications of the problem are numerous — transportation map, wires connecting the pins on a circuit, etc.

- Idea: Starting from s, greedily reduce G to G'.



Claim: If u is a nearest neighbor of s, then $\delta(s,u)=3$.

Pf: • If we take another path $p: s \leadsto u$ then $wt(p) \geq \delta(s,u)$ as the weights are non-negative. $\square$

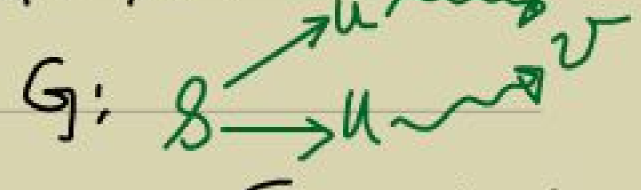
- This inspires the following transformation from G to G':
    - Let u be a nearest neighbor of s in G.
    - $\forall (u,x) \in E$, Add edge (s,x) with wt $w(s,x) := w(s,u) + w(u,x)$.
    - In case of multiple edges (s,x), keep the lighter one.
    - Remove u from G.

<u>Theorem:</u> $\forall v \in V \setminus \{u\}$, $\delta_G(s,v) = \delta_{G'}(s,v)$.

<u>Pf:</u>

- Note that $\delta_{G'}(s,v) = w(s,u) + \delta_G(u,v)$

[triangle inequality] $\geq \delta_G(s,v)$.



- Conversely, a shortest path $s \rightsquigarrow v$ in G, gives a path $s \rightsquigarrow v$ in G'

$$\Rightarrow \delta_{G'}(s,v) \leq \delta_G(s,v)$$

$$\Rightarrow \delta_{G'}(s,v) = \delta_G(s,v). \qquad \square$$

- Thus, greedily we are reducing G to G' with one fewer vertex. (in $O(n)$ time)
  $\Rightarrow$ In time $O(n^2)$ we can find
  $\{ \delta(s,v) \mid v \in V \}$.

- This is optimal if $m := |E| = \Theta(n^2)$.
  But, can we improve it for $m = o(n^2)$?

Lemma: Every subpath of a shortest path $s \rightsquigarrow u$ is also a shortest path.

Pf:

- Let $u_0 := s, u_1, u_2, \ldots, u_k := v$ be a shortest path.
- Suppose $(u_0, \ldots, u_i)$ is not a shortest. Then, we can use the shortest paths from $u_0 \rightsquigarrow u_i$ & $u_i \rightsquigarrow u_k$.
  $\Rightarrow$ We reduce $\delta(s,v)$, which is a contradiction.

$\square$

– More insights using the subpath property.

<u>Lemma</u>: Let $\underline{N_i}(\underline{s})$ be the vertices that are <u>i-th nearest</u> to s. Let $v \in N_i(s)$.
Then, $\exists j \exists u \in N_j(s)$ for $j < i$ s.t. $(u, v) \in E$.

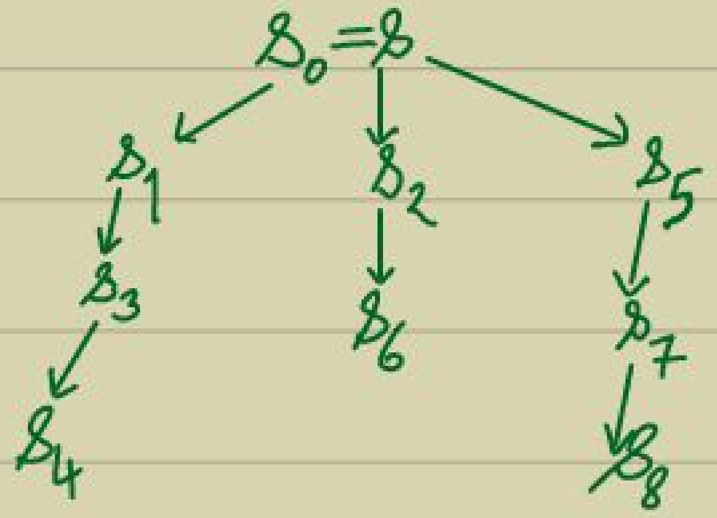
<u>Proof</u>:

- Let $(s =: u_0, u_1, \ldots, u_k := v)$ be a shortest path.
- Clearly, $\delta(s, u_{k-1}) < \delta(s, u_k = v)$ [assuming positive wts]

$\Rightarrow \exists j < i, \; u_{k-1} \in N_j(s)$. □

– Thus, one can think of the vertices as characterized by the distance from s:
s_i is the i-th nearest to s.

Eg.



shortest paths tree

- **Better idea:** Find the vertices $N_i(s)$ incrementally (as i grows).
    - Suppose $u_1, \dots, u_{i-1}$ are the vertices whose $\delta(s,\cdot)$ you know correctly.
    - Then, we can estimate the following distances for $v \in V \setminus \{u_1, \dots, u_{i-1}\}$:

$$\underline{L(v)} := \min_{(u_j, v) \in E,\, j \in [i-1]} \left(\delta(s, u_j) + w(u_j, v) \right)$$

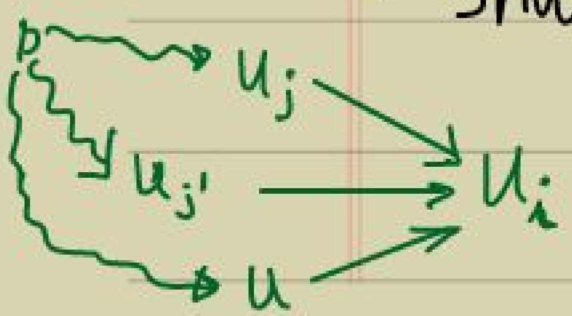
- Consider $\underline{u_i} := \operatorname{argmin}_v L(v)$.

closest to the pts. covered!

<u>Claim:</u> $\delta(s, u_i) = L(u_i)$.

<u>Pf:</u>
- Let (u_j, u_i) be the edge used in $L(u_i)$.
- Say, we get a path, <u>shorter</u> than $L(u_i)$, by using an edge $(u_{j'}, u_i)$ with $j' \in [i-1] \setminus \{j\}$ OR by an edge (u, u_i) with $u \notin \{u_1, \dots, u_{i-1}\}$.
- This will contradict the definition of $L(u_i)$ or even u_i.
- Thus, $L(u_i)$ is the shortest distance to u_i.

$\square$



Dijkstra's Algorithm

- Given $G=(V,E,w)$ & s.
  - $U \leftarrow V$; $\forall v \in U$, $L(v) \leftarrow \infty$; $S \leftarrow \phi$;
  - $L(s) \leftarrow 0$;
  - For $i = 0$ to $n-1$ {

ExtractMin →
    - $y \leftarrow$ vertex in U with min $L(\cdot)$;
    - $\delta(s,y) \leftarrow L(y)$; $s \rightsquigarrow y \to v$
    - Move y from U to S;
    - For each $(y,v) \in E$ with $v \in U$ {

DecreaseKey →
        $L(v) \leftarrow \min(L(v), \delta(s,y) + w(y,v))$;

      }

↑ Note: we're only updating neighbors of y.

  }

▷ There are n ExtractMin & m DecreaseKey calls in the algorithm.

▷ Using AVL tree wrt $L(\cdot)$ we get $O(m \lg n)$ time.

Later: Using Fibonacci heap we get $O(m + n \lg n)$ time.