Given connected, undirected graph $G=(V,E)$ with edge weights $w: E \to \mathbb{R}$.

Assume edge weights are distinct.

$\Rightarrow$ Min spanning tree is unique.

Goal: Find spanning tree T minimizing $\sum_{e \in T} w(e)$.

The alg:

T: the min spanning tree

(find this)

Maintain spanning forest $F \subseteq T$, adding edge as we go.

(Starts as $|V|$ isolated vertices)

Two special kinds of edges for a particular forest F.

e is <u>useless</u> if $F+e$ has a cycle.

(T has no useless edges.)

e is <u>safe</u> if it the lightest edge with one endpoint for some component of F.

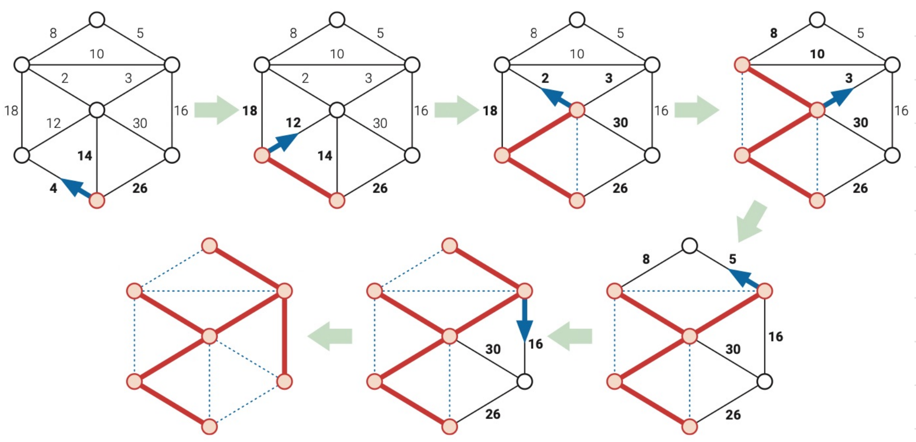
(T has <u>every</u> safe edge.)

Kruskal: Scan all edges in increasing weight order; add each safe edge you find to F.

$O(E \log V)$ time

Jarník [1929], Prim [1957]

Prim-Jarník: Pick one vertex to be in the only non-trivial component. Repeatedly add the safe edge for that component.



Keep a priority queue of edges incident to component. Priority is edge weight. Until queue is empty, remove edge uv & add uv to component if safe.

Prim-Jarník

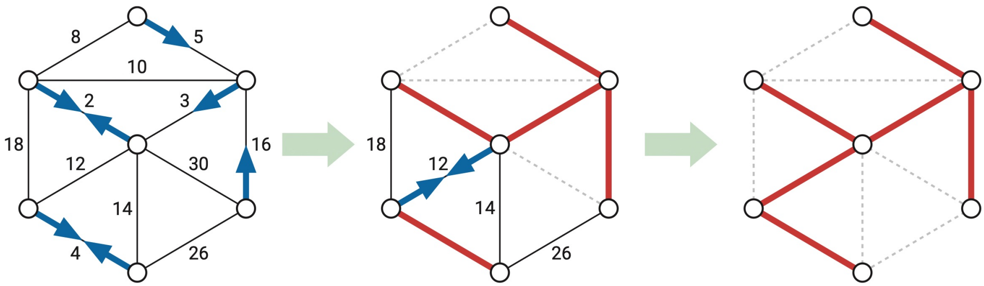
~~WHATEVERFIRSTSEARCH~~(s):
  put $(\varnothing, s)$ in ~~bag~~ priority queue
  queue
  while the ~~bag~~ is not empty
    take (p, v) from the ~~bag~~ queue $(\star)$
    if v is unmarked
      mark v
      parent$(v) \leftarrow p$
      for each edge vw $(\dagger)$
        put (v, w) into the ~~bag~~ $(\star\star)$

queue

$O(E \log V)$ time ($\log E = O(\log V)$ per p. queue operation)

Borůvka [1926]:
(Sollin [1961])

Add ALL the safe edges + recurse.




```
ADDALLSAFEEDGES(E, F, count):
  for i ← 1 to count
      safe[i] ← NULL
  for each edge uv ∈ E
      if comp(u) ≠ comp(v)
          if safe[comp(u)] = NULL  or  w(uv) < w(safe[comp(u)])
              safe[comp(u)] ← uv
          if safe[comp(v)] = NULL  or  w(uv) < w(safe[comp(v)])
              safe[comp(v)] ← uv
  for i ← 1 to count
      add safe[i] to F
```

```
BORŮVKA(V, E):
  F = (V, ∅)
  count ← COUNTANDLABEL(F)
  while count > 1
      ADDALLSAFEEDGES(E, F, count)
      count ← COUNTANDLABEL(F)
  return F
```

Each iteration takes $O(E)$ time.

<u>At worst</u>, halves # components, so $\leq O(\log V)$ iterations.

$O(E \log V)$ time

$\log V$ is <u>worst case</u>

$O(E)$ time for some "nice" types of graphs like planar

Can find safe edges in parallel.

Faster algorithms based on Borůvka.

(Best algs are $O(E)$ time randomized

& $O(E\,\alpha(E))$ deterministic

inverse Ackermann (almost constant)

New problem!

Given <u>directed</u> graph $G=(V,E)$ with edge weights $w: E \to \mathbb{R}$.

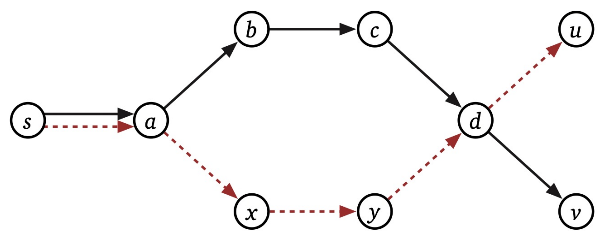
<u>Shortest</u> path P from s to t minimizes $\sum_{e \in P} w(e)$.

We'll find <u>all</u> shortest paths from s.

Single Source Shortest Paths (SSSP) problem.

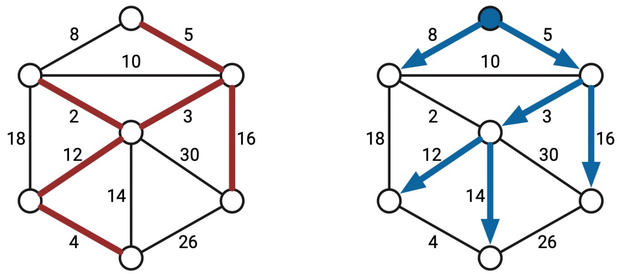
A subpath of a shortest path is a shortest path.

We can "break ties" to be consistent on our choice of subpaths.



So we get a tree rooted at s; the <u>shortest paths tree</u>.

Goal: Compute a shortest paths tree from a given vertex s.



If given an undirected graph, can replace each edge uv with $u \to v$ &

$v \to u$

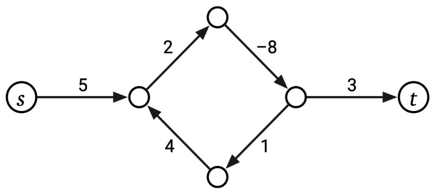
of same weight.

Edge weights may be negative...

The algorithms really work toward "shortest walks"

If a cycle has negative total weight, shortest walks don't exist!

Otherwise, these shortest walks are shortest paths!



Ford, Dantzig, + Minty:

Define two variables for each vertex v:

$dist(v)$: guess on distance to v

- always $\geq$ real distance
- initially, $dist(s) \leftarrow 0$

  $dist(v) \leftarrow \infty \ \forall v \neq s$

$pred(v)$: what we believe is previous vertex on shortest path to v

- initially, $pred(v) \leftarrow$ Null

All SSSP algs start with

```
INITSSSP(s):
    dist(s) ← 0
    pred(s) ← NULL
    for all vertices v ≠ s
        dist(v) ← ∞
        pred(v) ← NULL
```

Call an edge $u \to v$ "tense"

if $dist(u) + w(u \to v)$

$< dist(v)$.

Means $dist(v)$ is too high...

so "relax" the edge.

```
RELAX(u→v):
    dist(v) ← dist(u) + w(u→v)
    pred(v) ← u
```

FORDSSSP(s):
    INITSSSP(s)
    while there is at least one tense edge
        RELAX any tense edge

Will compute SSSP if there are no negative cycles!

(runs forever if s can reach a negative cycle)

$s \to \ldots \to pred(pred(v)) \to pred(v) \to v$

will be the shortest path

dist(v) will be its distance