The greedy algorithm runs to completion:

Let $e$ be an uncolored edge.

If $e$'s ends are in different blue trees, apply blue rule.

If $e$'s ends are in the same blue tree, apply red rule.
Correctness of blue rule:

\[ e = \min \text{ uncolored edge across cut.} \]

\[ T = \min \text{ tree containing all blue edges, no red ones.} \]

If \( e \) not in \( T \): find path in \( T \) connecting ends of \( e \), edge \( e' \) on path crossing cut. 

Swap \( e \) and \( e' \) to give \( T' \)

\[ c(e') \leq c(e') \text{ by blue rule}, \]

\[ c(e) \geq c(e') \text{ by minimality of } T \]

\( T' \) satisfies invariant after \( e \) is blue

(swapping can only occur if equal costs)
Correctness of red rule:

\( e = \text{max uncolored edge on cycle.} \)

\( T = \text{min tree containing all blue edges, no red ones} \)

If \( e \in T \), delete \( e \) from \( T \), find edge \( e' \) on cycle (other than \( e \)) reconnecting two parts, form \( T' \) by swapping \( e' \) for \( e \) in \( T \).

\( c(e) \geq c(e') \) by red rule

\( c(e) \leq c(e') \) by minimality of \( T \)
“HON-EEE...CALL A MATHEMATICIAN!”
Shortest Paths

Digraph with edge weights (costs, distances)

Shortest path from s to t: path of minimum total wt.

Problems:

single pair: given s, t, find a shortest path from s to t

single source: given s, find shortest paths from s to all reachable vertices

all pairs: find shortest paths between all pairs

Cases:

acyclic

no negative wts

general

(planar, etc.)
Properties:

If a shortest path from \( s \) to \( t \) iff there is no negative (total wt.) cycle on a path from \( s \) to \( t \).

If there is no such cycle, there is a shortest path that is simple (no repeated vertex).

If no neg cycle reachable from \( s \), then \( s \) shortest path tree; rooted at \( s \), contains all vertices reachable from \( s \), all tree paths are shortest paths in graph.

New goal: find a negative cycle or construct a shortest path tree.

(single-source problem is central)
Given a spanning tree $T$, rooted at $s$,

\[ d(v) = \text{tree wt from } s \text{ to } v, \text{ is } T \text{ a shortest path tree?} \]

Yes, iff there is no $\{(v,w)\}$ with $d(v) + c(v,w) < d(w)$

Edge relaxation algorithm to find a shortest path tree:

\[ d(s) = 0, \quad d(v) = \infty \text{ for } v \neq s \]

while there exists $\{(v,w)\}$ with $d(v) + c(v,w) < d(w)$

\[ d(w) = d(v) + c(v,w); \quad p(w) = v \]

$d(v)$ is always the wt of some $s-v$ path

if algorithm stops and $p$ defines a tree, must be a shortest path tree

stops iff no neg cycle

(alg maintains $d(w) \geq d(v) + c(v,w)$ if $v = p(w)$)
Suppose \( T \) not a SP tree. Let \( x \) be such that \( d(x) > s-x \). Let \( P \) be a shortest path from \( s \) to \( x \). \( d'(v) = P \)-distance from \( s \), \( (v, w) \) first edge along \( P \) such that \( d'(w) < d(w) \). Then \( d(v) + c(v, w) = d'(v) + c(v, w) = d'(w) < d(w) \). (This gives the hard direction of SP tree test.)

Suppose edge relaxation algorithm creates a cycle.

Then it must be a negative cycle.

\[
s \rightarrow w \rightarrow v_2 \rightarrow v_3 \rightarrow w \rightarrow x \rightarrow C \rightarrow s
\]

\[
d(v) + c(v, w) < d(w) \Rightarrow d(v) - d(w) + c(v, w) < 0
\]

Turn around cycle: \[
\sum_{i=1}^{k} d(v_i) - d'(v_{i+1}) + c(v_i, v_{i+1}) < 0
\]
Labeling and scanning algorithm:

$L = \{s\};$ $d(s) = 0;$ $d(v) = \infty$ for $v \neq s$;

while $L \neq \emptyset$ do
  remove $v$ from $L$;
  scan($v$); for each $(v,w)$ do
    if $d(v) + c(v,w) < d(w)$ then
      $d(w) = d(v) + c(v,w);$ $p(w) = v;$ add $w$ to $L$;

(unlabeled)

(labeled)

(scanned)
Any edge: topological scanning order

$D(x)$

Non-negative weight: shortest-first scanning order

$D(x^4)$ original  $D(x^4 + n)$ standard heap

$D(x^{n^2} + n^2)$ Fibonacci heap

No vertex scanned more than once:

Invariant $\phi(x) = x(x) = x(\phi(x))$

Induct $x$

\[ x \]

\[ \phi(x) \]
General case: FIFO scanning order

Maintain $L$ as an (ordinary) queue.

Phases:

phase 0: scan of $s$

phase $k$: scan of vertices added to $L$ during phase $k-1$

After phase $k$, all distances for shortest paths of $k+1$ or fewer edges are correct.

$\Rightarrow$ $n-1$ or fewer phases

$\Rightarrow \mathcal{O}(nm)$ time.
Negative cycle detection:

Method 1: Count phases, stop after first scan of n-th phase. Parent pointers will define a (negative) cycle.

Method 2 (early detection): Maintain a predecessor list of vertices in the current shortest path tree. When re-labeling w using \((v,w)\), explore the subtree rooted at w, disassembling it and looking for v.

Both methods take \(O(m)\) time total.

(Theoretically) inferior methods:

Method 3: When re-labeling v using \((v,w)\), follow parent pointers from v looking for w.

Method 4: Maintain tentative shortest path tree as a dynamic tree.
bad (negative cycle)
Definition: New vomit\(\frac{1}{l}\) level of products

\[ (c_{old} + m) \rightarrow (c_{new} + m) \rightarrow 0 \]
All pairs:

Dynamic prog.

\[ d(x, x) = 0 \]

\[ d(x, y) = d(x, z) + d(z, y) \text{ if } x \neq y \text{ and } (x, y) \in E \]

\[ d(z, y) = d(x, z) \text{ if } x = y \text{ and } (x, y) \in E \]

for \( z \)

for \( y \)

for \( x \)

for \( y \)

if \( d(x, z) + d(z, y) \leq d(x, y) \) then

\[ d(y) \leftarrow d(x, z) + d(z, y) \]

\[ O(n^3) \]
single source

\[ \text{Dijkstra: } O(nm + n^2 \log n) \]

Bellman-Ford: 

\[ \text{eliminate neg edge cost} \]

\[ c(v, w) = c(v, w) + p(v) - p(w) \geq 0 \]
Heuristic Search: Let $e(v)$ be an estimate of the distance from $v$ to the goal $t$.

Use Dijkstra's algorithm with $d(v) + e(v)$ as the selection criterion.

The method works if

$$e(v) \leq d(v, w) + e(w) \text{ for all } v, w$$

(Estimate $e$ is a consistent lower bound on the actual distance.)

In Euclidean graphs the distance "as the crow flies" works.

Hart, Nilsson, Rafael (1968)
Heuristic Search: Let $e(v)$ be an estimate of the distance from $v$ to the goal vertex $t$.

Use Dijkstra's algorithm, but expand the frontier vertex $v$ with $d(v) + e(v)$ minimum.

This method is correct if

$$e(v) \leq d(v, w) + e(w) \quad \text{for all } v, w.$$  

Estimate $e$ is a consistent lower bound on the actual distance.

Distance "as the crow flies" works.

Hart, Nilsson, Raphael (1964)
Dijkstra's Algorithm

Heuristic Search
Bidirectional Search: Search forward from s and backward from t concurrently.

⇒ Getting the stopping rule correct is tricky, especially for bidirectional heuristic search.