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 uncolored edge across cut.

T = min 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'

$c(e) \le c(e')$ by blue rule,

$c(e) \ge c(e')$ by minimality of T

T' satisfies invariant after e is blue

(swapping can only occur if equal costs.)

Correctness of red rule:

e = max uncolored edge on cycle.

T = 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-EEEEE ...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:

∃ 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 ∃ 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,

$d(v)$ = tree wt from s to v, is T a shortest path tree?

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

Edge relaxation algorithm to find a shortest path tree:

$d(s) = 0$, $d(v) = \infty$ for $v \neq s$

while $\exists$ edge (v,w) with $d(v) + c(v,w) < d(w)$
do $\{ d(w) = d(v) + c(v,w);\ p(w) = v \}$

$d(v)$ 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 $v = p(w)$)

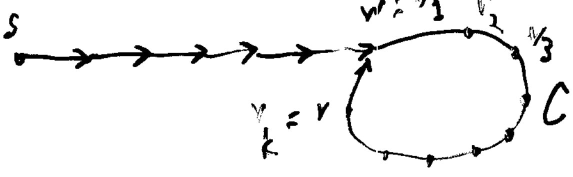
Suppose T not a sp tree. Let x be such that $d(x) >$ s-x distance. Let P be a shortest path from s to x, $d'(v) =$ P-distance from s, (v,w) first edge along P such that $d'(w) < d(w)$.

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 C.

Then it must be a negative cycle.



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

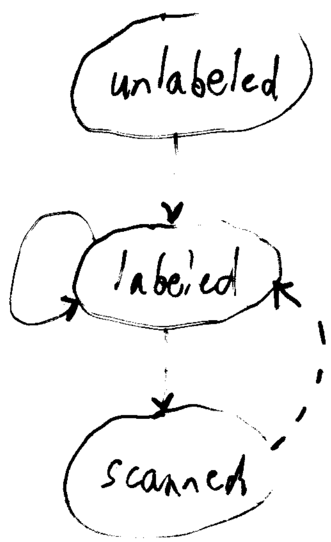
Sum around cycle: $\sum_{i=1}^{k} d(v_i) - d(v_{i+1}) + c(v_i, v_{i+1}) < 0$

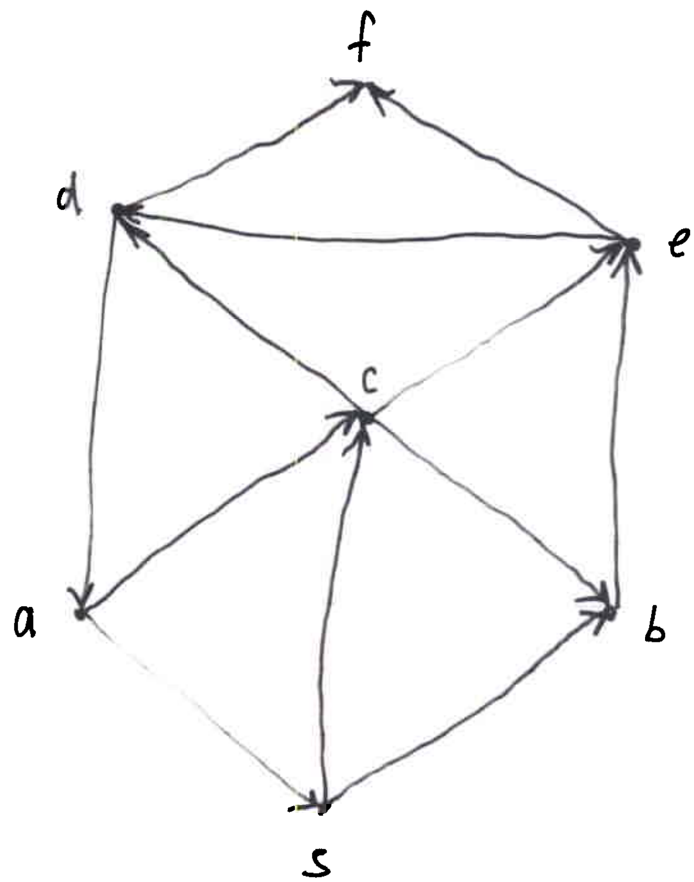
$$\sum_i c(v_i, v_{i+1})$$

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 $L\}\}$




f
d
e
c
a
b
s

3 2

1

9

5

10 12

8 7

4

6 2

3 2.

-2.

-1 .5

10 12.

-1

7

4.

8 3

Acyclic: topological scanning order

$O(m)$

Non-negative weights: shortest-first scanning order
(Dijkstra)

$O(n^2)$ original $O(m \log n)$ standard heap

$O(n \log n + m)$ Fibonacci heap

No vertex scanned more than once:

Invariant $d(S) \leq d(L) \leq d(U)$

smallest x

≥ 0

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

$\Rightarrow$ $O(nm)$ time

Negative cycle detection:

Method 1: count phases, stop after first scan of n^{th} phase. Parent ptrs will define a (negative) cycle.

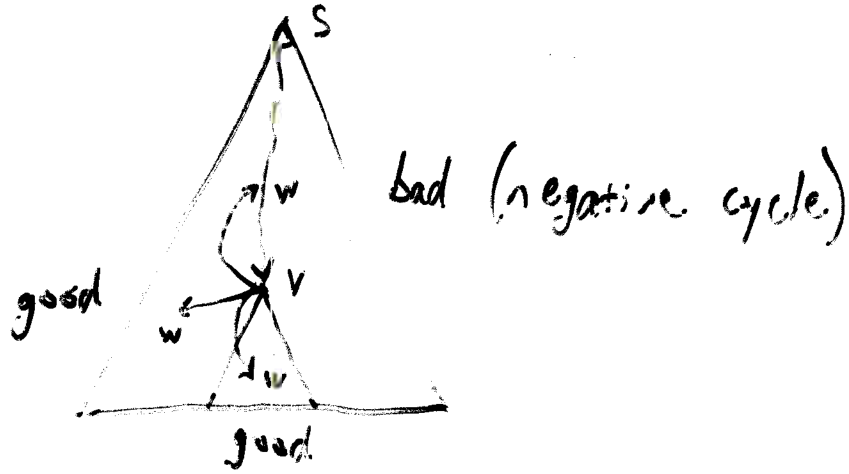
Method 2 (early detection): Maintain a preorder list of vertices in tentative shortest path tree. When relabeling w using (v, w), explore the subtree rooted at w, disassembling it and looking for v.

Both methods take $O(nm)$ time total.

(Theoretically) inferior methods:

Method 3: When relabeling v using (v, w), follow parent pointers from v looking for w.

Method 4: Maintain tentative shortest path tree as a dynamic tree.


s
w
bad (negative cycle)
good
v
w
w
good

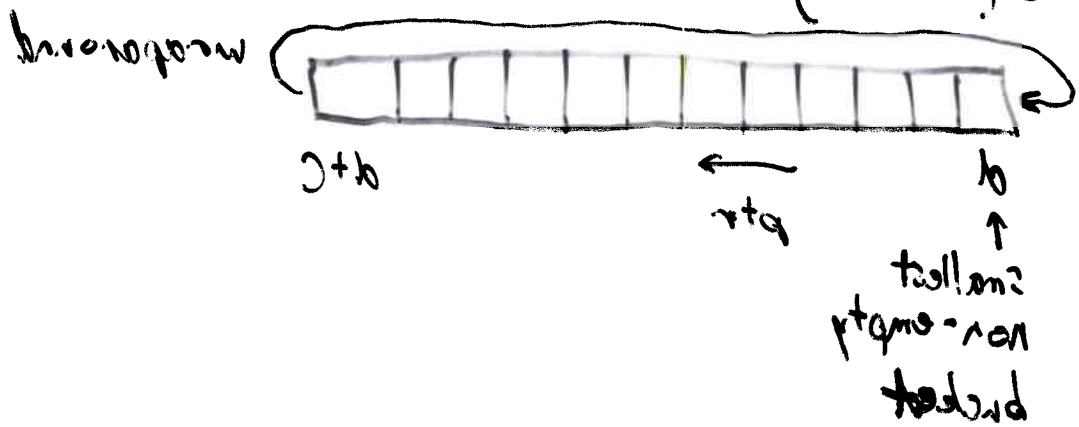
Dijkstra algorithm:

heap is monotone: vertices are removed in increasing order by tentative distance

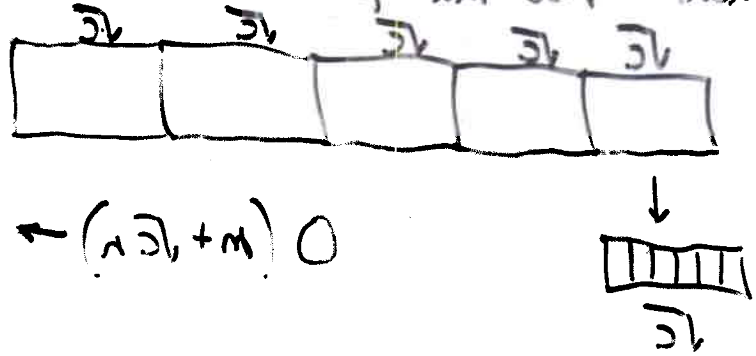
Can exploit this if edge wts are (small!) integers

Dial: buckets for tentative distances
buckets = max edge wt. (C) + 1

$O(m + Cn)$ time



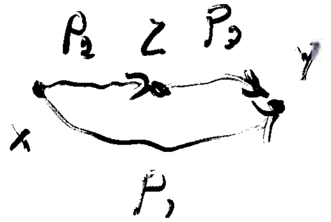
Refinement: Use multiple levels of buckets



$O(m + \sqrt{C}\, n) \rightarrow O(m + n \log C)$
(binary levels)

All pairs:

Dynamic prog.



$d(x,x) = 0$

$d(x,y) = \infty$ if $x \neq y$, $(x,y) \notin E$

$d(x,y) = c(x,y)$ if $x \neq y$, $(x,y) \in E$

for z

for x

for y

if $d(x,z) + d(z,y) < d(x,y)$ then

$d(x,y) = d(x,z) + d(z,y)$

$O(n^3)$

n single sources

Dijkstra: $O(nm + n^2 \log n)$

1 Bellman-Ford $\Rightarrow$ eliminate neg edge costs

$p(v)$

$$c'(v,w) = c(v,w) + p(v) - p(w) \geq 0$$

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

$$e(v) \le \ell(v,w) + e(w) \quad \text{for all } v, w$$

(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

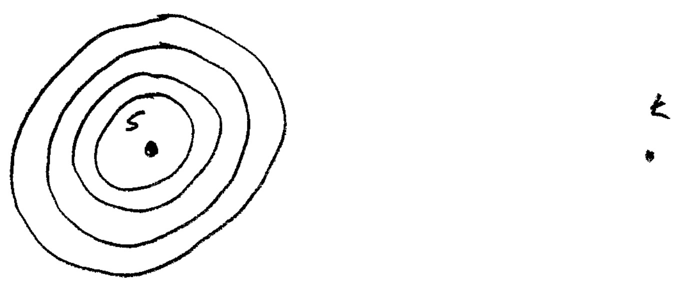
$e(v) \le \ell(v,w) + e(w)$ for all v, w:

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

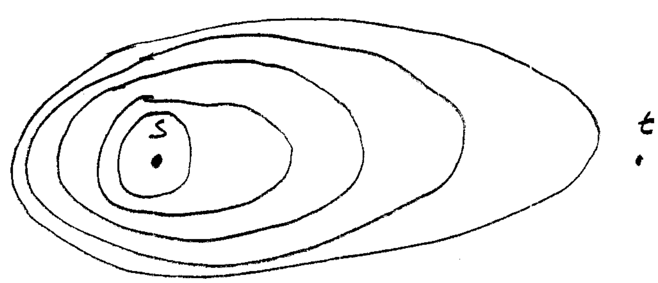
Distance "as the crow flies" works.

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.