CS314 - Shortest paths

Note Title 10/7/2013

Announcements

- Exam a week from Friday (review next Wed.)
- Grading tomorrow

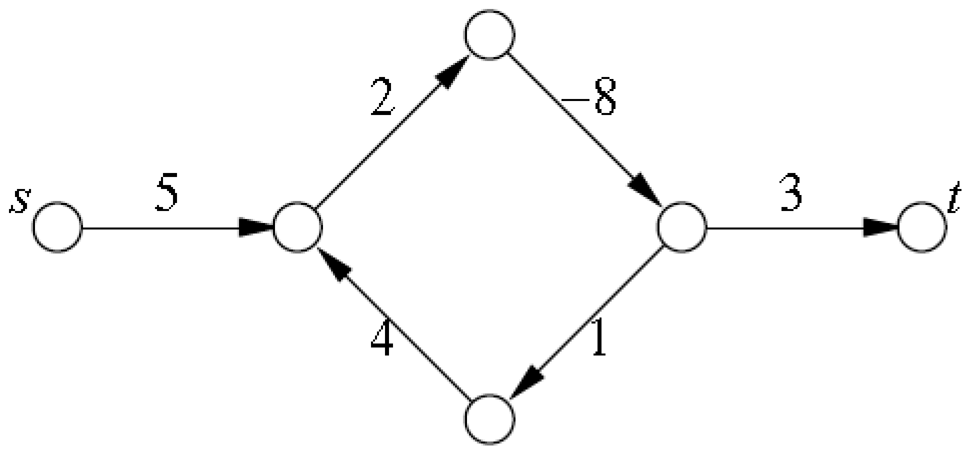
Shortest paths

Input: a directed graph $G = (V, E, w)$

Goal: a shortest path from s to t, for $s, t \in V$

Note: We'll assume shortest paths are unique, just to keep things simple.

We'll also keep edge weights positive.
Why?



Dykstra's algorithm ('59)

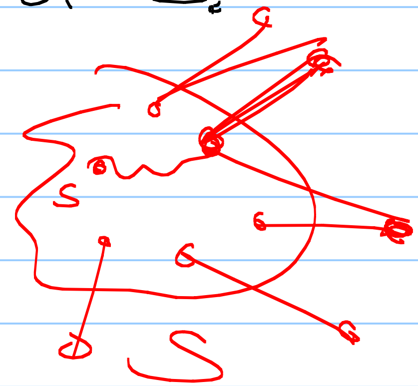
(Discovered by Leyzorek, Gray, Johnson .. in '57)

Keep an "explored" part of the graph, S.

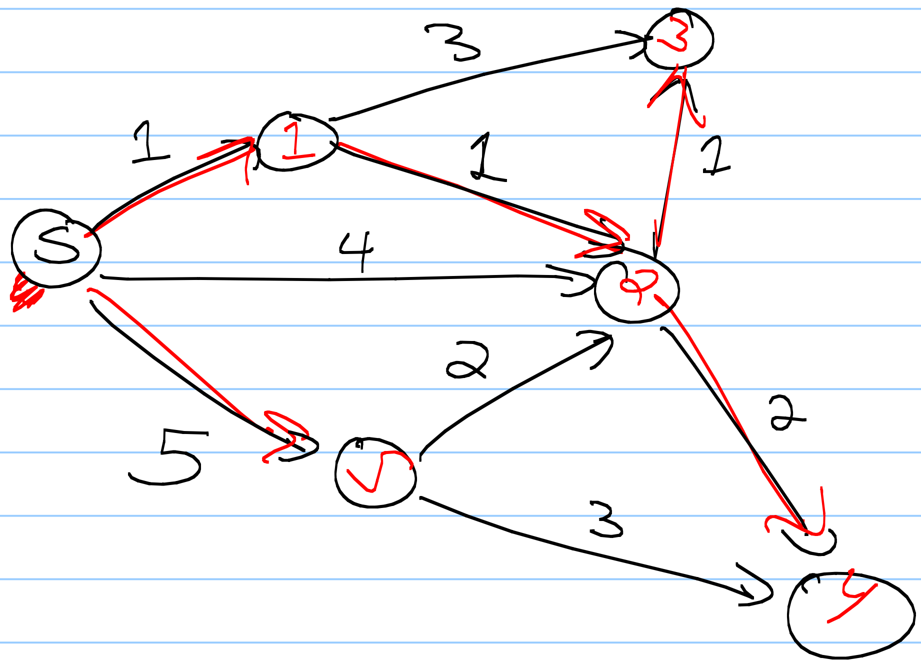
Initially, $S = \{s\}$ and $d(s) = 0$.

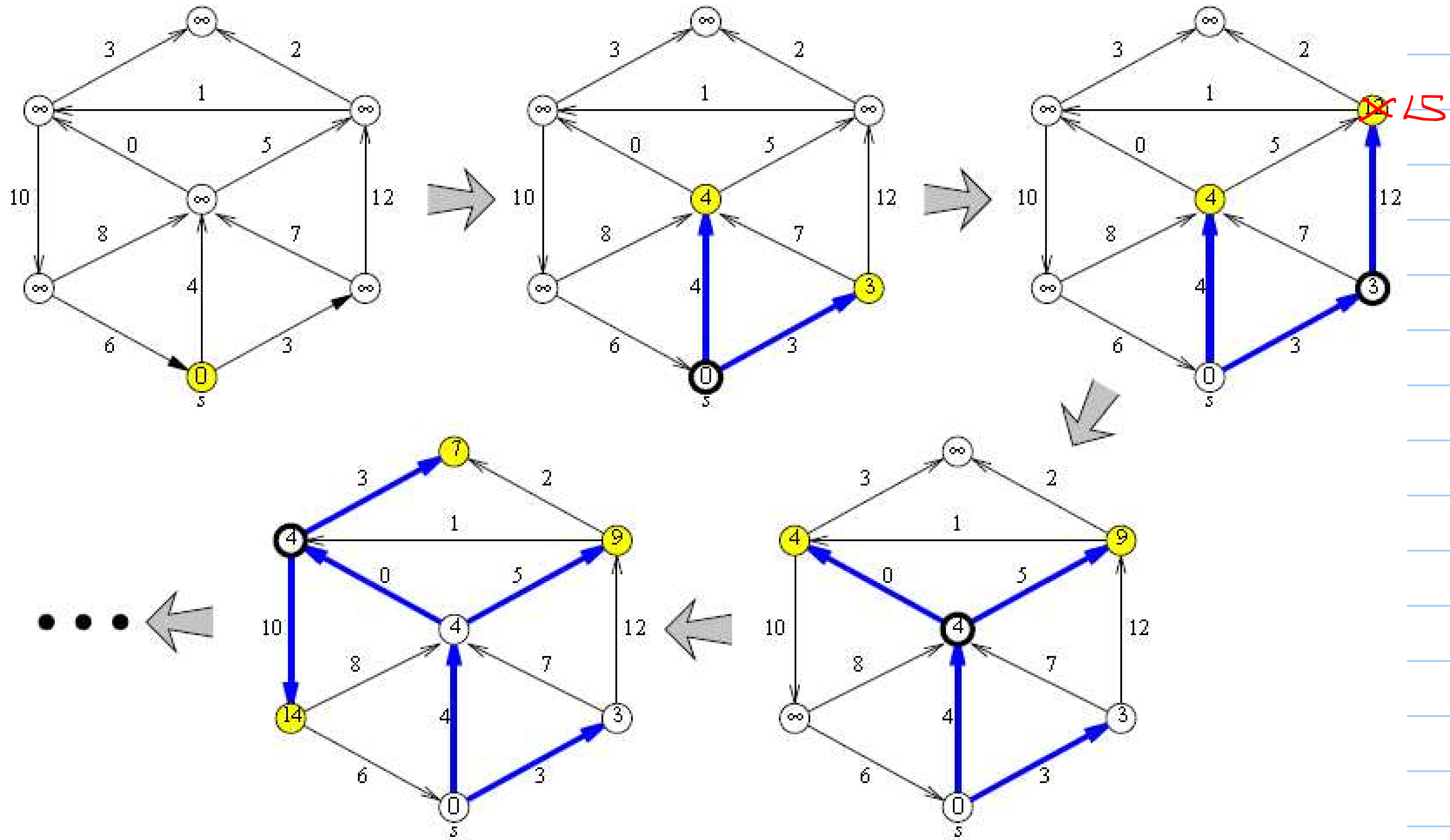
Now, find shortest path out of S:

$$\min_{\substack{e=(u,v) \\ u \in S,\, v \notin S}} d(u) + w(e)$$



$\to$ add this vertex


S
1
3
2
4
5
3
1
1
2
2
3
2
4



3 2 1 0 5 10 12 8 7 4 6 3 s
15
...

Code:

$S \leftarrow \{s\}$

for each vertex v,
    $D[v] \leftarrow \infty$

$D[s] \leftarrow 0$

($P[s] \leftarrow$ NULL)

while $S \neq V$:

    select node $v \notin S$ with at least one edge from S to v which:

    $\underset{\substack{e=(u,v)\\ u \in S}}{\min} D(u) + w(e)$ is smallest

    add v to S.

    (Store $P[v] \leftarrow u$.)

Correctness

Thm: Consider the set S at any point in algorithm. For each $u \in S$, the path P_u is a shortest $s \to u$ path.

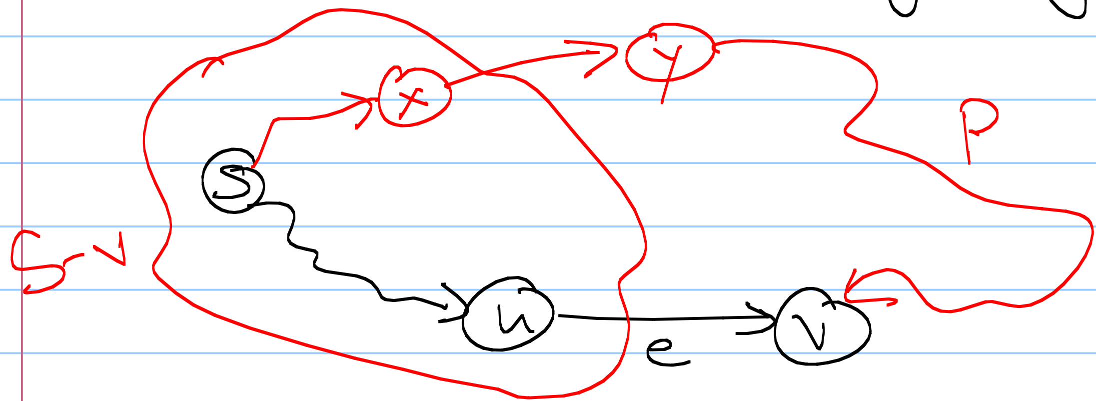
proof: induction on size of S.

Base case: $|S| = 1$, so $S = \{s\}$

IH: Suppose claim holds when $|S| = k-1$.

IS: Consider $|S| = k$, when v is added to S.

Let $e = (u,v)$ be edge getting us to v.



Claim: any other $s \rightsquigarrow v$ path P is longer.

Consider a path P.

pf (cont) At some vertex x along P, leave S + enter $V-S$ for first time.

Portion of P up to x was considered by my algorithm, since $x \in S$ + $y \in V-S$.

Means $D[x] + w(x \to y) \geq D[u] + w(e)$

and $w(P) \geq D[x] + w(x \to y)$

so $wt(P) \geq D[u] + w(e)$.

$\blacksquare$

Implementation:

Need to track current set of "reachable vertices".

First try:

For each $v \in S$, check every edge & calculate $D[v] + w(e)$ edge

worst case, $O(m)$ per loop

$\Rightarrow$ total $O(mn)$

Better: Use a heap!

priority queues can insert, delete, and change keys.

$\Rightarrow O(\log n)$ per operation

When v is added to S:

Look through all of v's edges, either insert node along with key = D[v] + w(e) or change key, if D[v] + w(e) is better.

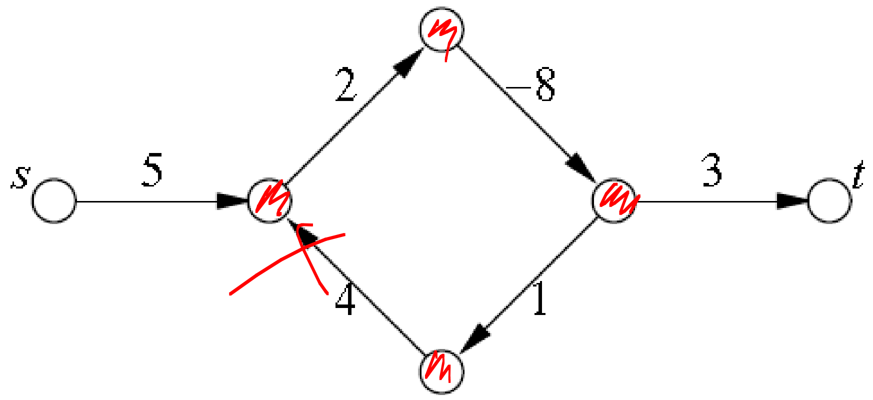
Runtime: n log n

(each vertex inserted & removed at least once)

& each edge could trigger change key

$\boxed{m \log n}$

What about negative edges?



dynamic programming!

Bell man-Ford ('58)

(actually Shimbel '55)

Force a path to use each edge at most once.

Essentially, builds this using dynamic programming!

Recursion:

$$dist_i(v) = \begin{cases} 0 & \text{if } i = 0 \text{ and } v = s \\ \infty & \text{if } i = 0 \text{ and } v \neq s \\ \min \left\{ \begin{array}{c} dist_{i-1}(v), \\ \min\limits_{u \to v \in E} (dist_{i-1}(u) + w(u \to v)) \end{array} \right\} & \text{otherwise} \end{cases}$$

(See notes for two ways to implement — uses a queue instead of a priority queue.)

Slower: $O(m \cdot n)$