remember PRIM'S ALGORITHM for MST?

1) start w/ any vertex $s$; set $w(s) = 0$
2) set $w(\neq s) = \infty$ & put all in pr.queue
3) while pr.queue not empty
   
   $x$: extract-min & add edge to $T$
   mark $x \to$ in $T$.
   for each unmarked neighbor $v$ of $x$
   if $w(v) > w(v,x)$ then decrease.
remember PRIM'S ALGORITHM for MST?

(after initializing)
while pr.queue not empty
    x: extract-min & add edge to T
    mark x\rightarrow in T.
    for each unmarked neighbor v of x
        if w(v) > w(v,x) then decrease.
PRIM'S ALGORITHM for MST

(modified)

(after initializing)

while pr.queue not empty
    x: extract-min & add edge to T
    mark x → in T.
    for each unmarked neighbor v of x
        if w(v) > w(v,x) then decrease.  
        RELAX (x,v)
(after initializing)
while pr.queue not empty
  x: extract-min & add edge to T
  mark x → in T.
  for each neighbor v of x
    RELAX(x,v)
DIJKSTRA'S ALGORITHM for SSSP

(after initializing)
while pr.queue not empty
    x: extract-min & add edge to T
    mark x -> in T.
    for each neighbor v of x
        RELAX(x, v)

(new example)
DIJKSTRA'S ALGORITHM for SSSP

(after initializing)
while pr.queue not empty
    x: extract-min & add edge to T
    mark x -> in T.
    for each neighbor v of x
        RELAX(x,v)

extract source & relax two edges
Dijkstra’s Algorithm for SSSP

(after initializing)
while pr.queue not empty
    x: extract-min & add edge to T
    mark x -> in T.
    for each neighbor v of x
        RELAX(x, v)

extract min: 10
DIJKSTRA'S ALGORITHM for SSSP

(after initializing)

while pr.queue not empty

x: extract-min & add edge to T
mark x → in T.

for each neighbor v of x

RELAX(x, v)
DIJKSTRA'S ALGORITHM for SSSP

(after initializing)
while pr.queue not empty
    x: extract-min & add edge to T
    mark x -> in T.
    for each neighbor v of x
        RELAX(x,v)
DIJKSTRA'S ALGORITHM for SSSP

(after initializing)
while pr.queue not empty
    x: extract-min & add edge to T
    mark x → in T.
    for each neighbor v of x
        RELAX(x,v)

No outcome from relaxing 13
Dijkstra's Algorithm for SSSP

(after initializing)
while pr.queue not empty
    x: extract-min & add edge to T
    mark x \rightarrow in T.
    for each neighbor v of x
        RELAX(x, v)
DIJKSTRA'S ALGORITHM for SSSP

(after initializing)
while pr.queue not empty
  x: extract-min & add edge to T
  mark x \to in T.
  for each neighbor v of x
    RELAX(x, v)
DIJKKTRA'S ALGORITHM for SSSP

(after initializing)
while pr.queue not empty
  \( x: \) extract-min & add edge to \( T \)
  mark \( x \rightarrow \) in \( T \).
  for each neighbor \( v \) of \( x \)
    RELAX(\( x, v \))
DIJKSTRA'S ALGORITHM for SSSP

(after initializing)

while pr.queue not empty

  x: extract-min & add edge to T

  mark x \rightarrow in T.

  for each neighbor v of x

  RELAX(x, v)
DIJKSTRA'S ALGORITHM for SSSP

(after initializing)
while pr.queue not empty
    x: extract-min & add edge to T
    mark x -> in T.
    for each neighbor v of x
        RELAX(x,v)
**Dijkstra's Algorithm** for *SSSP*

(after initializing)

while pr.queue not empty

\[ x : \text{extract-min} \text{ & add edge to } T \]

mark \( x \rightarrow \text{in } T. \)

for each neighbor \( v \) of \( x \)

\[ \text{RELAX}(x,v) \]
Dijkstra's Algorithm for SSSP

(after initializing)

while pr.queue not empty

- x: extract-min & add edge to T

mark x → in T.

for each neighbor v of x

RELAX(x,v)

time?
**DIJKSTRA'S ALGORITHM for SSSP**

(after initializing)

while pr.queue not empty

- \( x \): extract-min & add edge to \( T \)
- mark \( x \rightarrow \) in \( T \).

for each neighbor \( v \) of \( x \),

- RELAX(\( x,v \))

**Time:** \( O(\min\{V^2, E \log V\}) \)

(like Prim's algo, for fancier see CLRS)
Correctness:

Assume we have shortest path to a set of red vertices.
Correctness:

Assume we have shortest path to a set of red vertices.

Somewhere outside this set is a vertex $v$ with shortest path $= \ [\text{a path in known set}] + \text{black edge}$.
Correctness:

Assume we have shortest path to a set of red vertices.

Somewhere outside this set is a vertex $v$ with shortest path = $\text{[a path in known set]} + \text{black edge}$.

Any other path $s \rightarrow v$ will cost more.
Correctness:

Assume we have shortest path to a set of red vertices.

Somewhere outside this set is a vertex \( v \) with shortest path =

\[ \text{[a path in known set]} + \text{black edge} \]

Any other path \( s \rightarrow v \) will cost more \( \{ \) assuming weights \( \geq 0 \)}