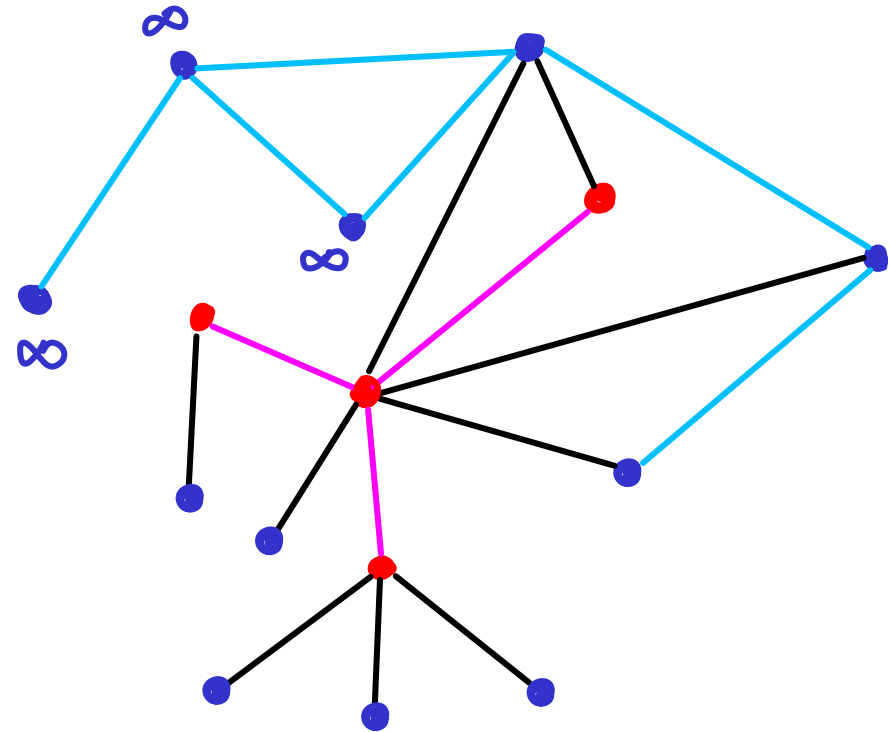
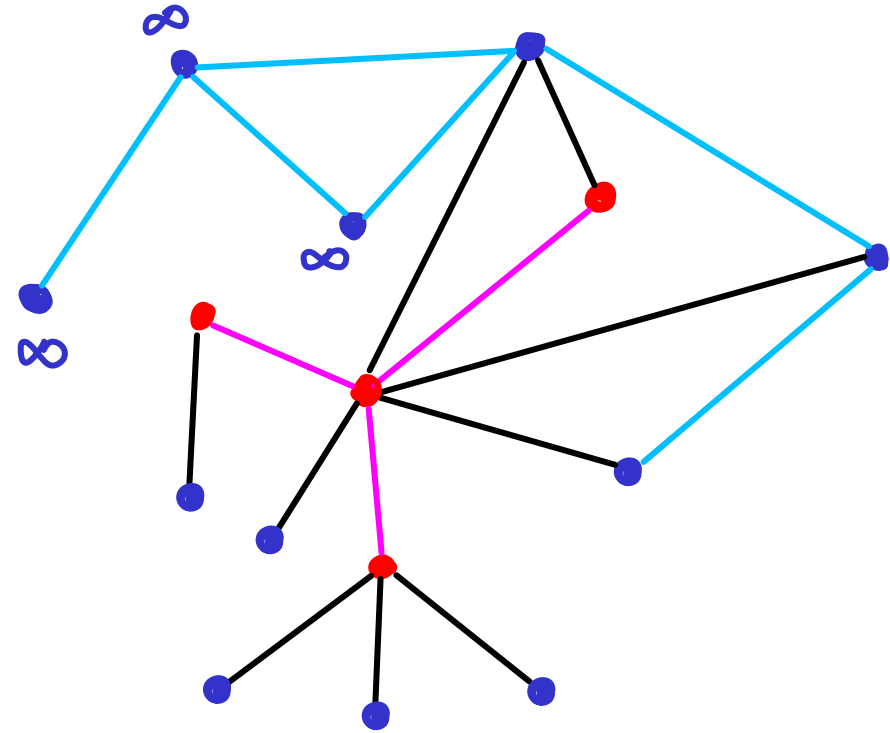


# remember PRIM'S ALGORITHM for MST ?



- 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  $x \rightarrow$  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

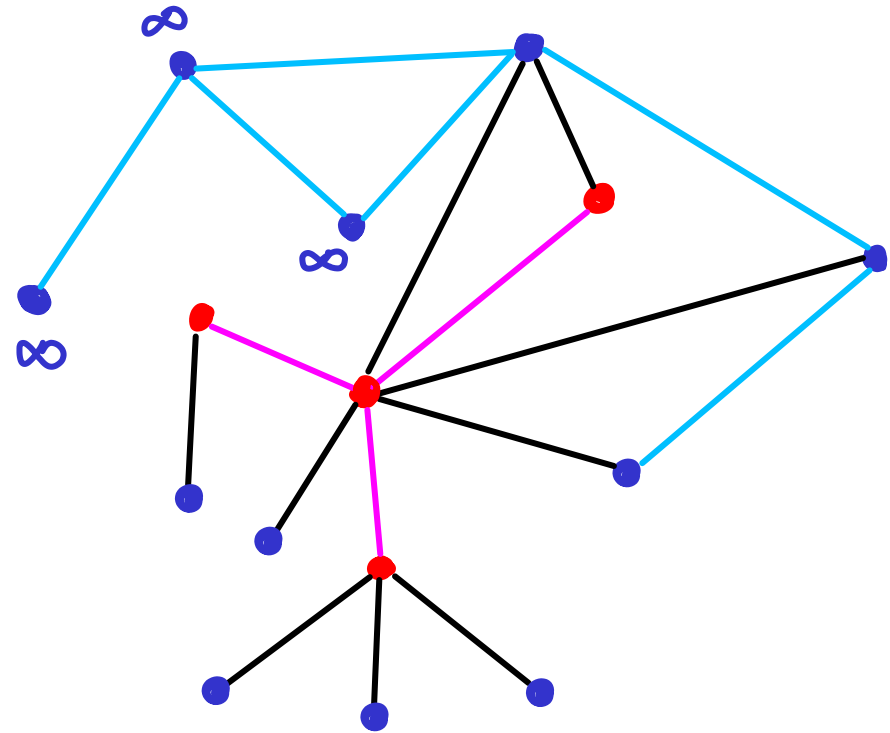
mark x → in T.

for each unmarked neighbor v of x  
if  $w(v) > w(v, x)$  then decrease.

modified

# PRIM'S ALGORITHM

~~for MST~~



(after initializing)

while pr.queue not empty

x: extract-min & add edge to T

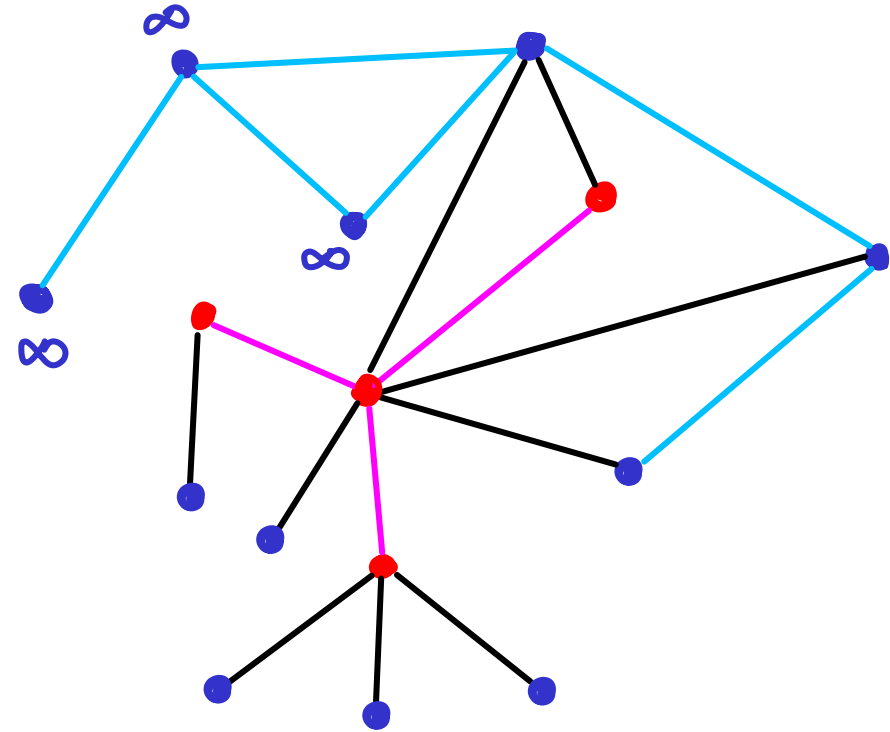
mark  $x \rightarrow$  in T.

for each ~~unmarked~~ neighbor v of x  
if  ~~$w(v) > w(v, x)$~~  then decrease.

RELAX(x, v)

# DIJKSTRA'S ALGORITHM for SSSP

(1956 ~ 1959)



(after initializing)

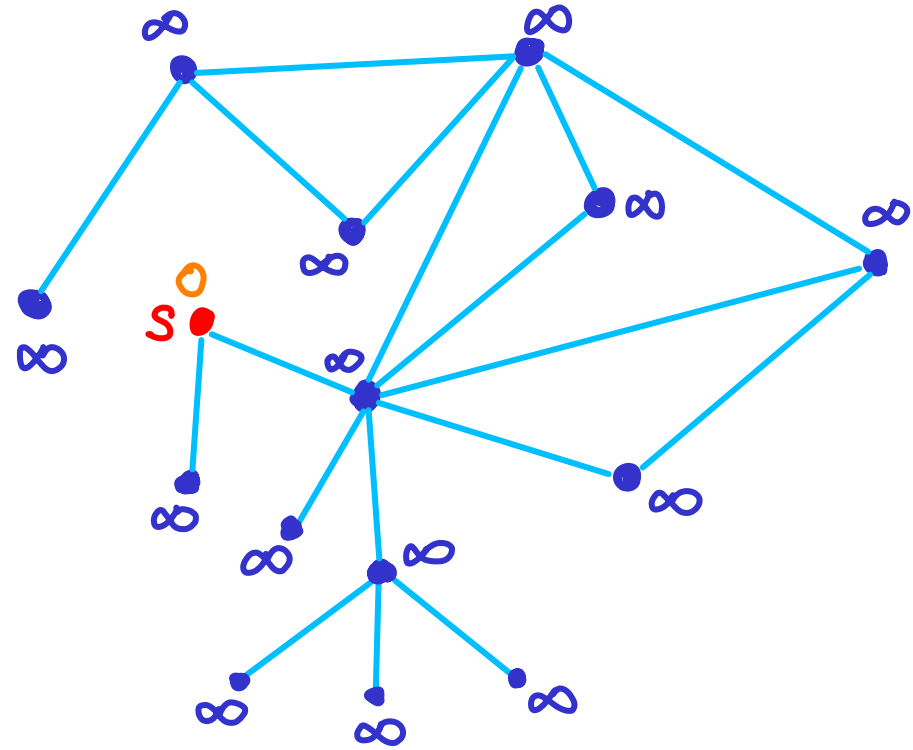
while pr.queue not empty

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



Initialize

(after initializing)

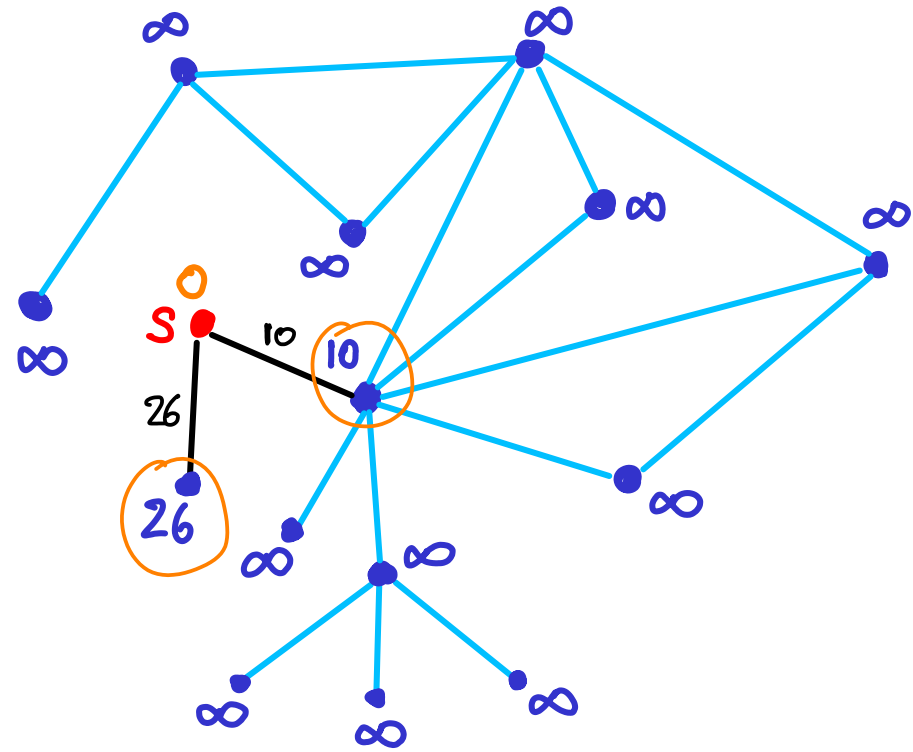
while pr.queue not empty

| 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

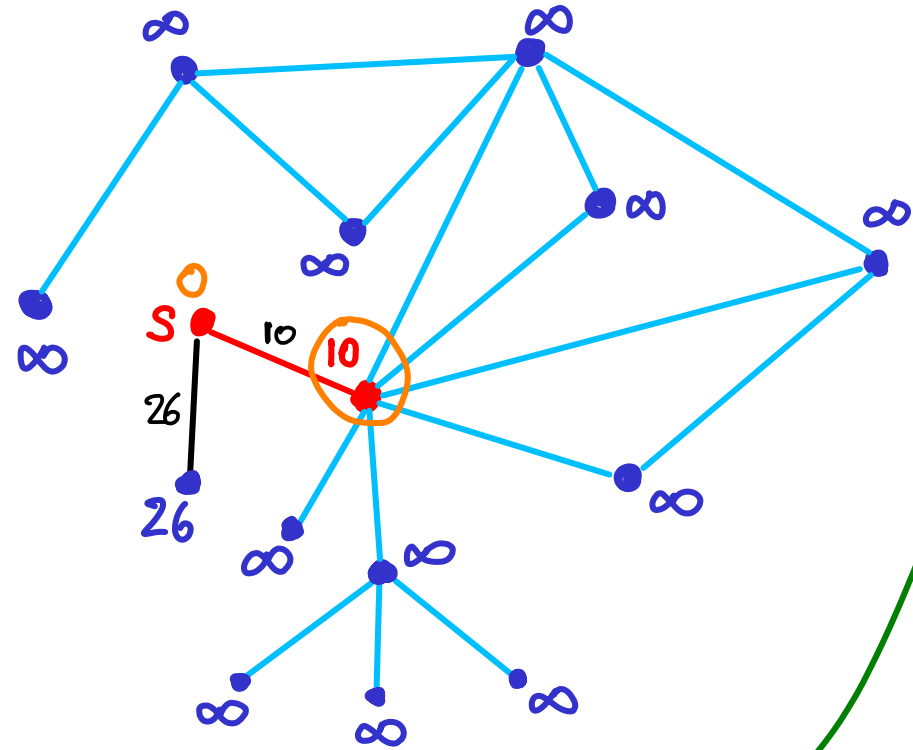
x: extract-min & add edge to T

mark  $x \rightarrow$  in T.

for each neighbor  $v$  of  $x$   
RELAX( $x, v$ )

extract source & relax two edges

# DIJKSTRA'S ALGORITHM for SSSP



(after initializing)

while pr.queue not empty

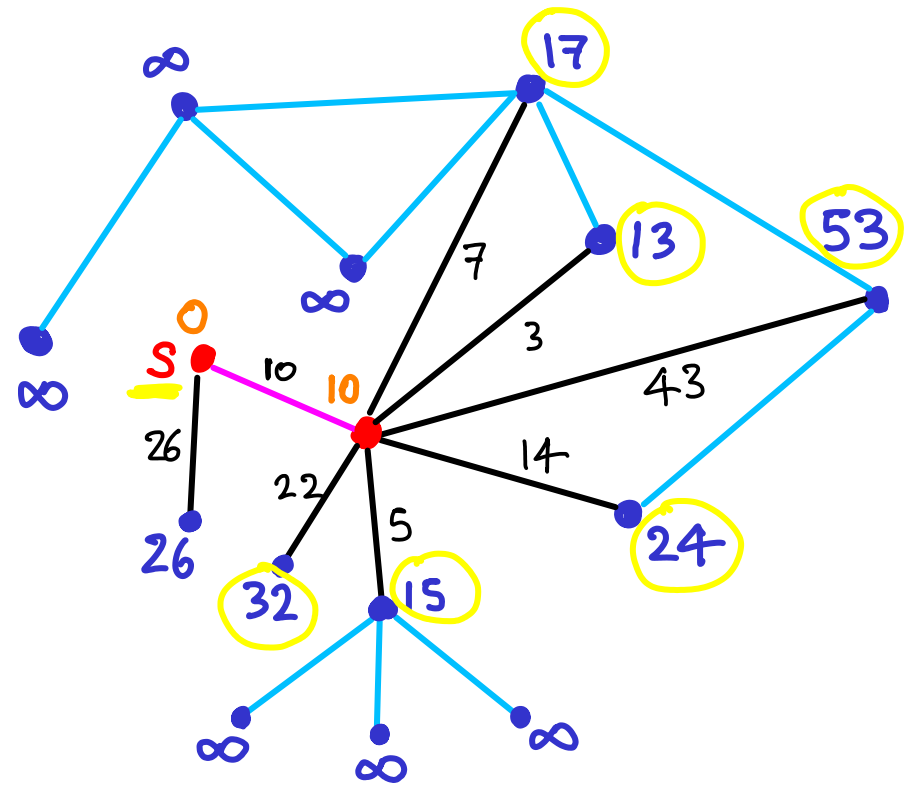
x: extract-min & add edge to T

mark  $x \rightarrow$  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

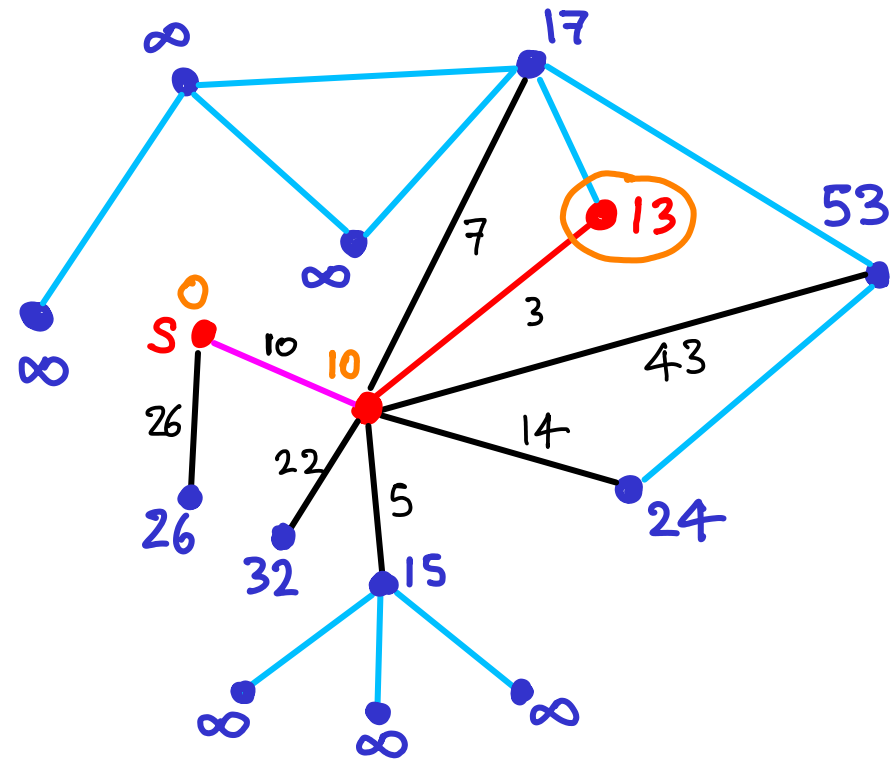
mark x → in T.

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

Relax neighbors of 10



# DIJKSTRA'S ALGORITHM for SSSP



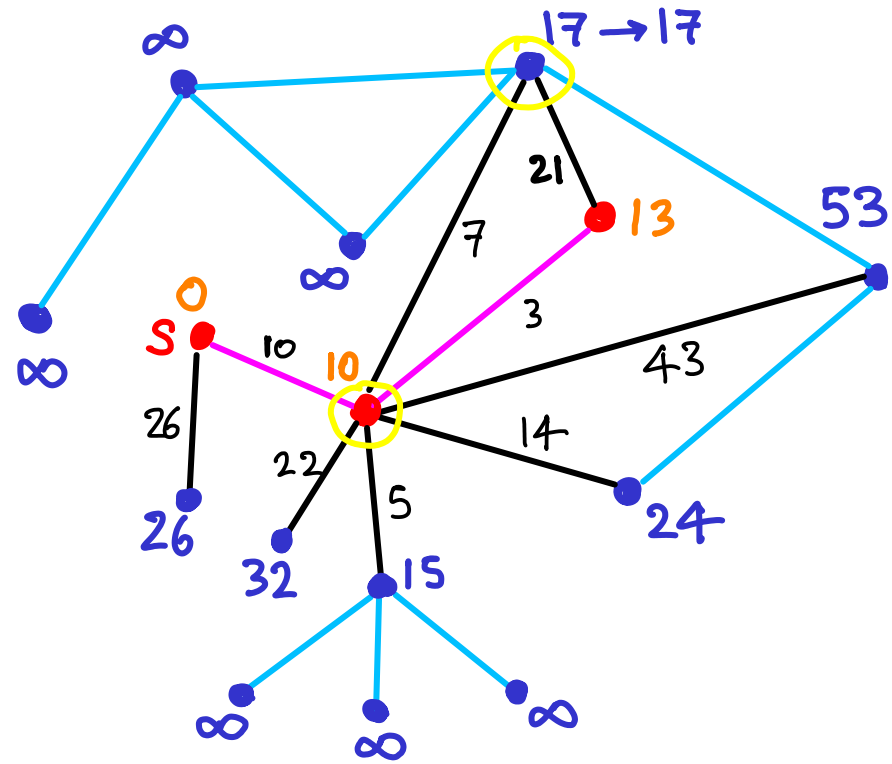
(after initializing)

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



No update from relaxing

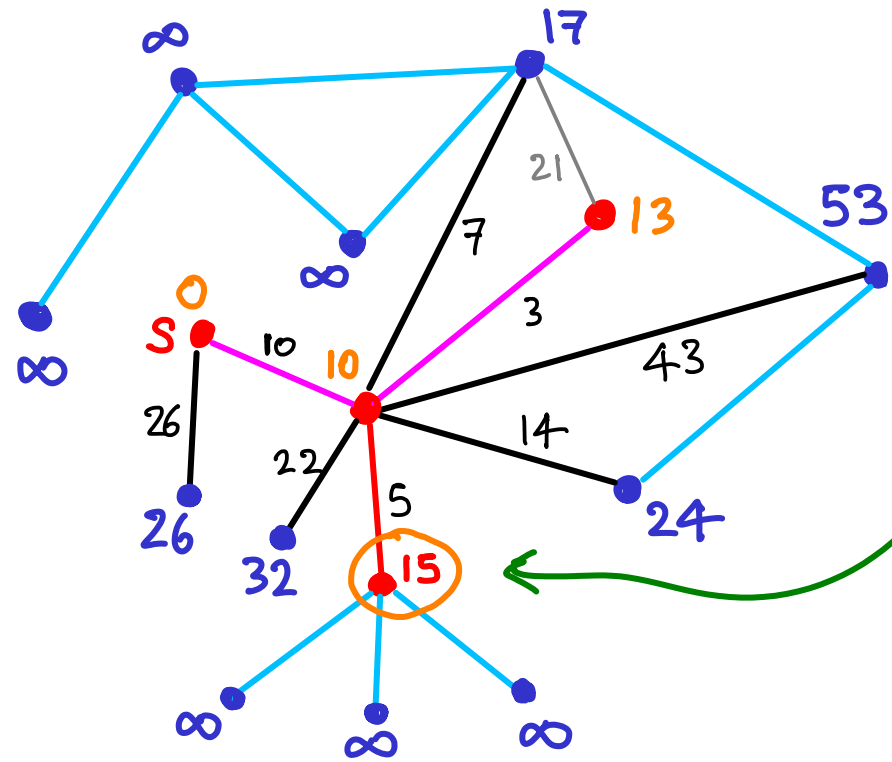
(after initializing)

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



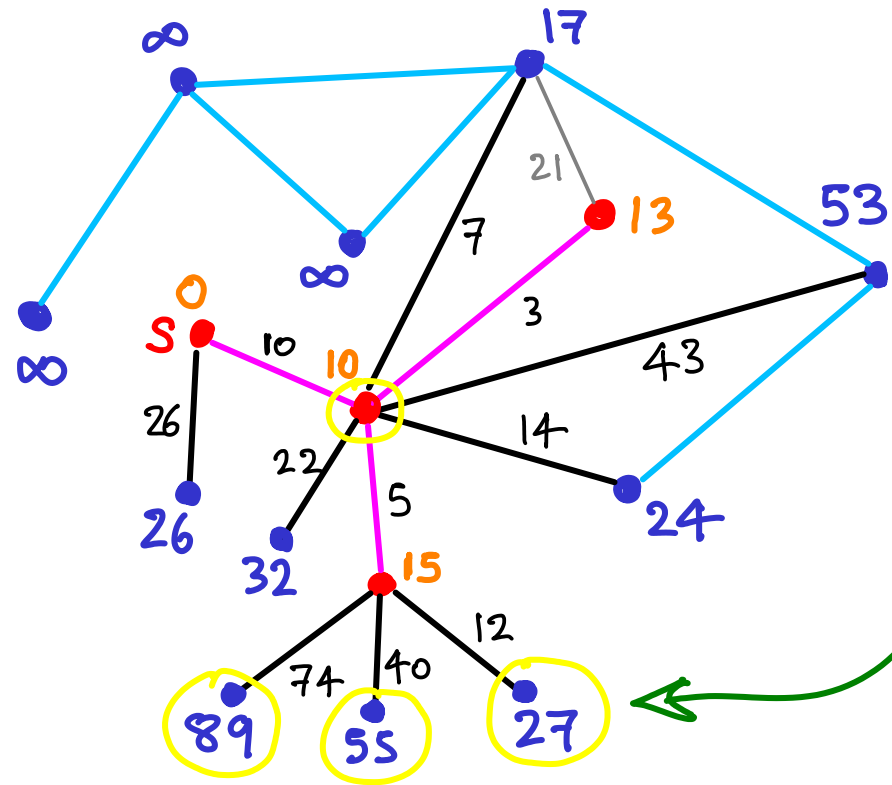
(after initializing)

while pr.queue not empty

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

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

# DIJKSTRA'S ALGORITHM for SSSP



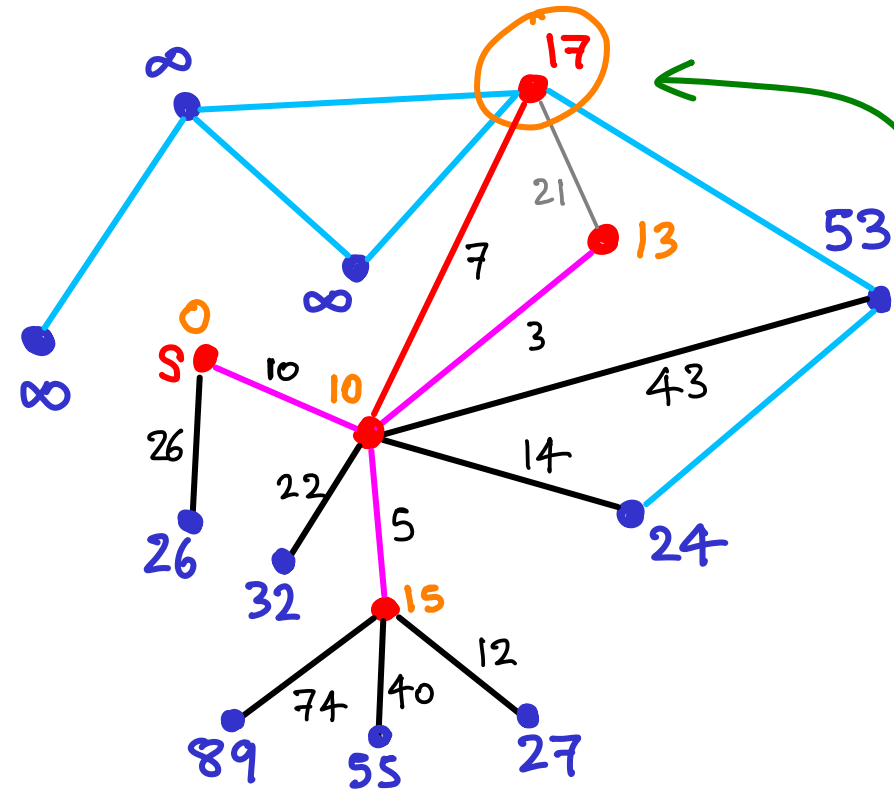
(after initializing)

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



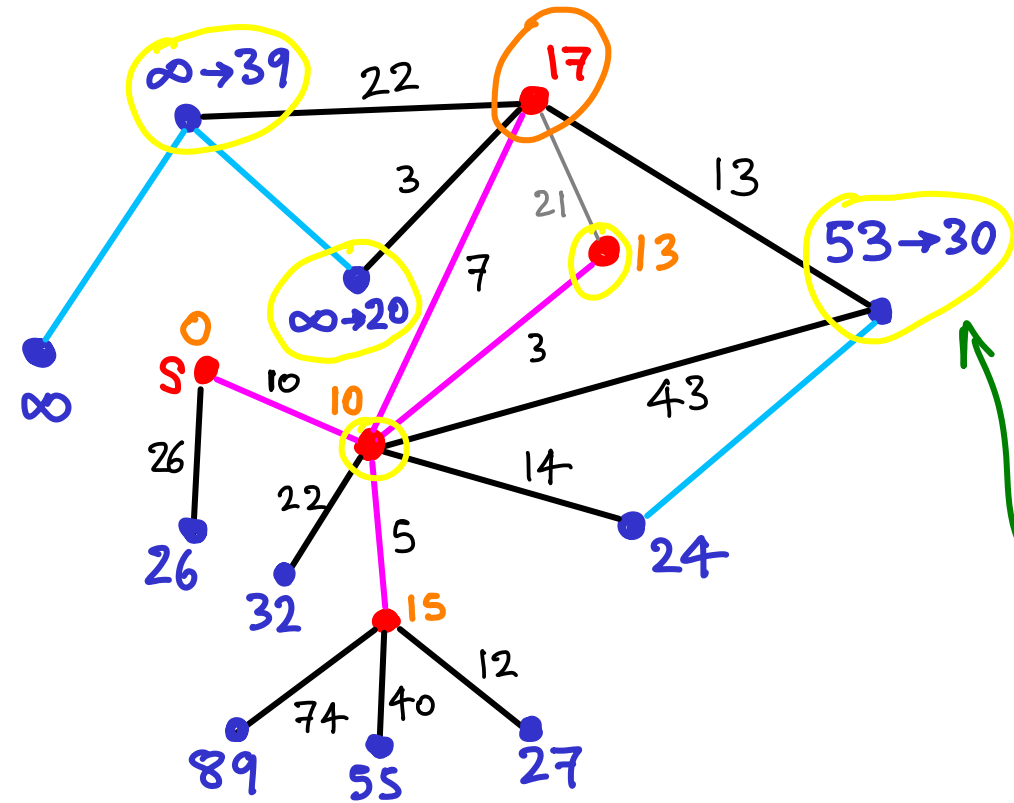
(after initializing)

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)

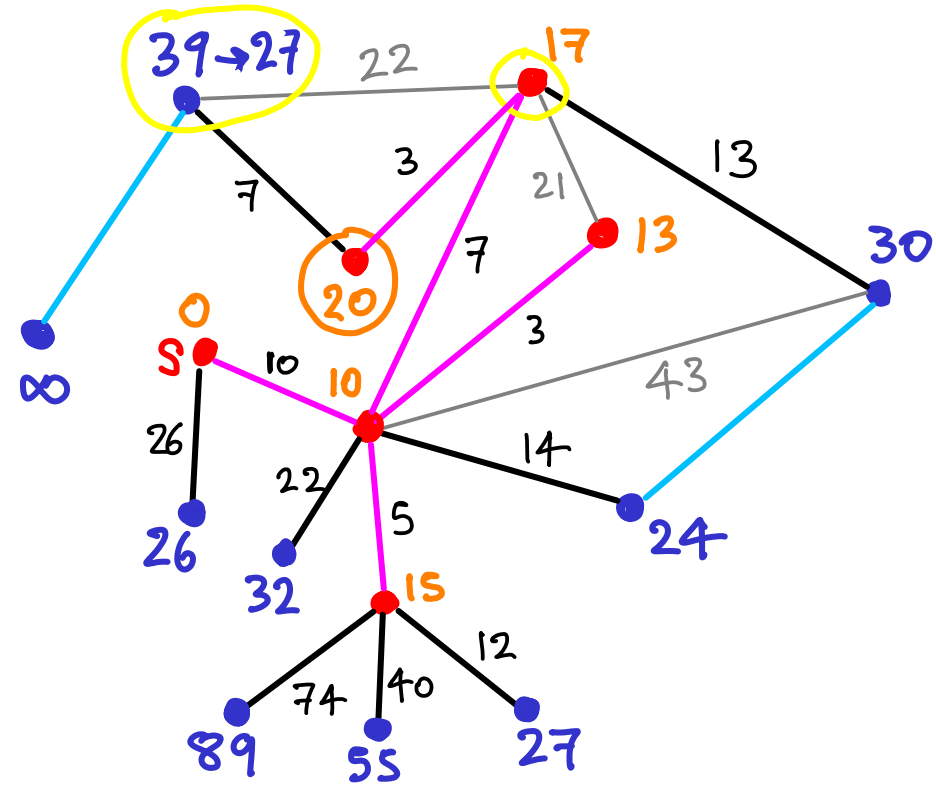
while pr.queue not empty

$x$ : 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)

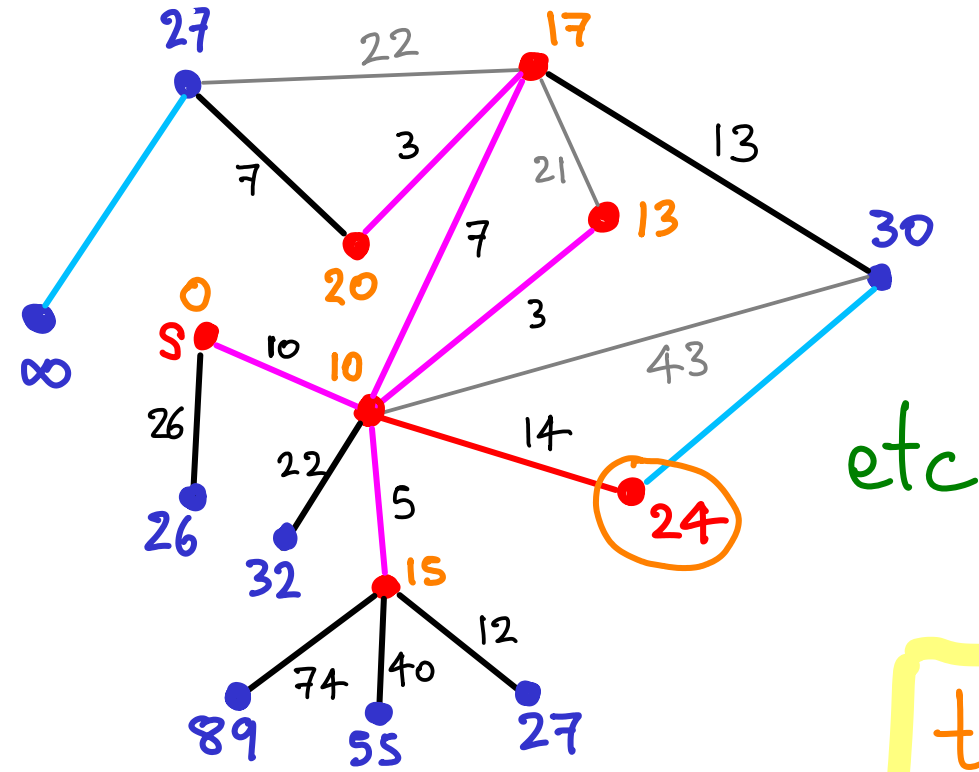
while pr.queue not empty

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

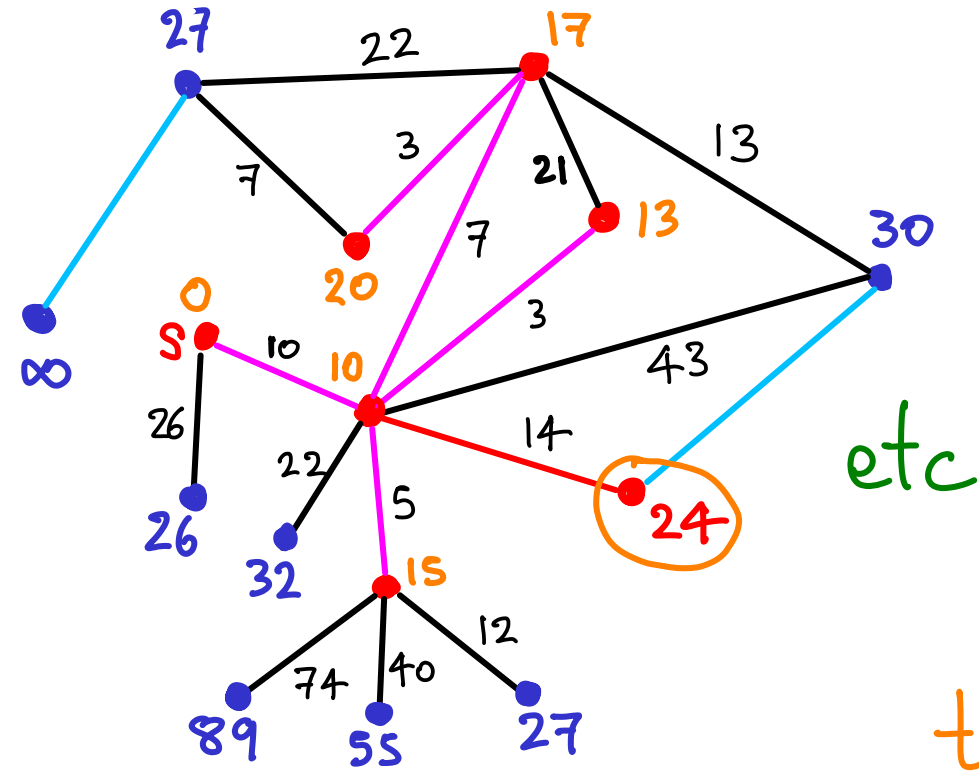
mark  $x \rightarrow$  in T.

for each neighbor  $v$  of  $x$   
RELAX( $x, v$ )

etc

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**( $x, v$ )

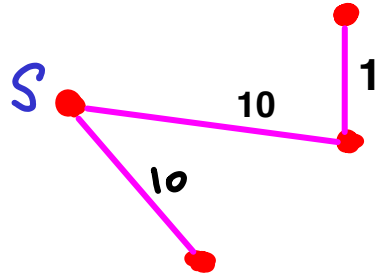
time :  $O(V^2)$  or  $O(E \log V)$

( like Prim's algo // for fancier see CLRS

Correctness :

assume

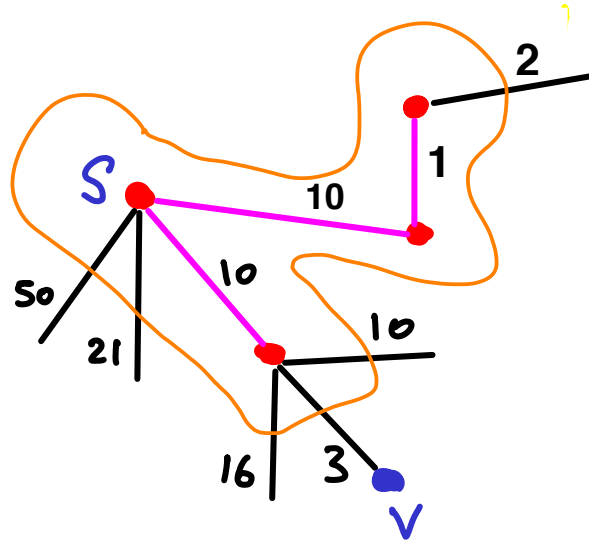
we have shortest path  
to a set of red vertices



# Correctness :

assume

we have shortest path  
to a set of red vertices



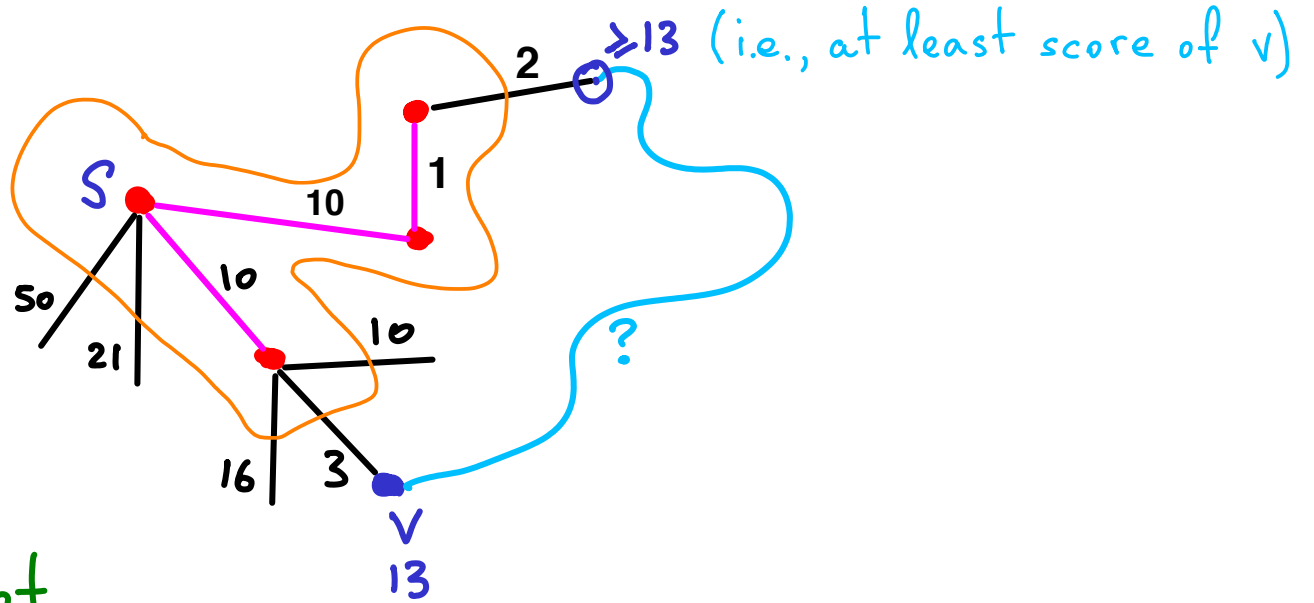
Somewhere outside this set

is a vertex  $v$  with shortest path =  
= [a path in known set] + black edge

# Correctness :

assume

we have shortest path  
to a set of red vertices



Somewhere outside this set

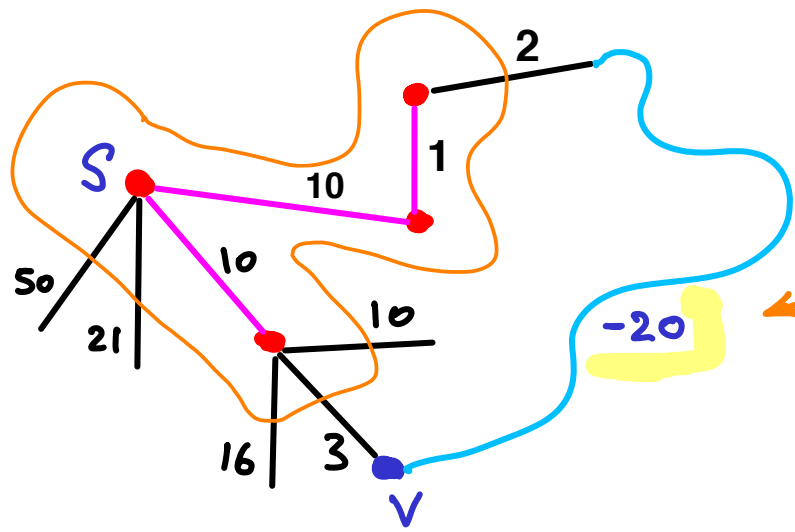
is a vertex  $v$  with shortest path =  
= [a path in known set] + black edge

Any other path  $s \rightsquigarrow v$   
will cost more

# Correctness :

assume

we have shortest path to a set of red vertices



w/ weights  $\leq 0$   
greedy choice  
now invalid

Somewhere outside this set

is a vertex  $v$  with shortest path =

= [a path in known set] + black edge

Any other path will cost more



} assuming weights  $\geq 0$