(G, w, s)

digraph — source (root)

$w : E \to \mathbb{R}$ weight

Single-source min-cost (weight) paths

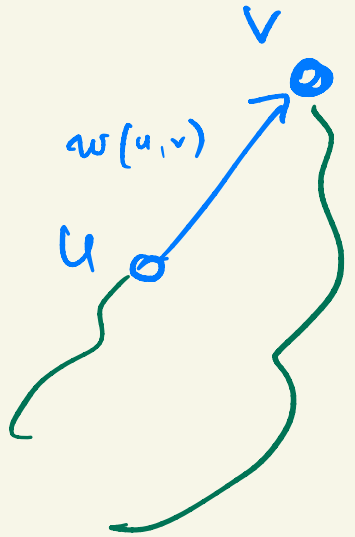
"shortest" — misnomer

DIJKSTRA's ALGORITHM

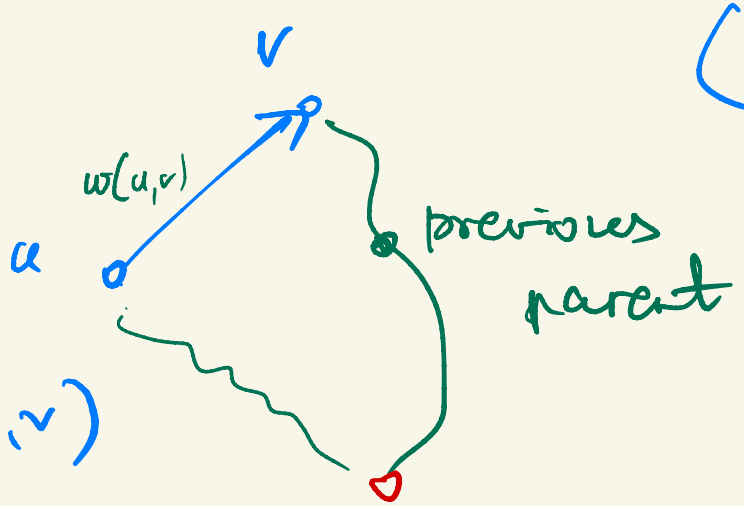
$s \in V$ source

Q: priority queue

```
for v ∈ V                Q = ∅
    status(v) = WHITE
    p(v) = NIL
    cost(v) = ∞
-------------------------------
status(s) := GRAY
INSERT(Q, s)          cost[s] = 0
    p(s) := s
-------------------------------
while Q ≠ ∅
    u ← EXTRACT-MIN(Q)
    for v ∈ Adj[u]
        if status(v) = WHITE
            status(v) := GRAY
                RELAX(u, v)        ← updates parent cost
            INSERT(Q, v)
        elseif status(v) = GRAY
            RELAX(u, v)
    endfor
    status(u) := BLACK             u finished (explored)
endwhile
```



RELAX (u, v)



if $cost(v) > cost(u) + w(u,v)$

then $cost(v) := cost(u) + w(u,v)$

$p(v) := u$

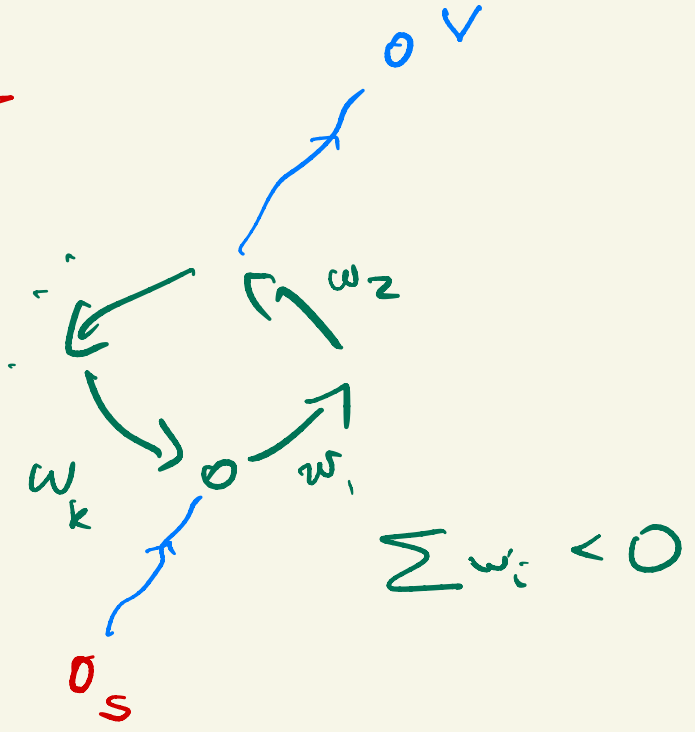
NOTE: if status(v) was WHITE
then "if" not needed, answer always "YES"
b/c LHS was ∞

DIJKSTRA works for $w(e) \geq 0$

non-negative weights

What if $\nexists$ no negative cycles

What if $\exists$ negative cycles



if we allow $s \to v$ walks

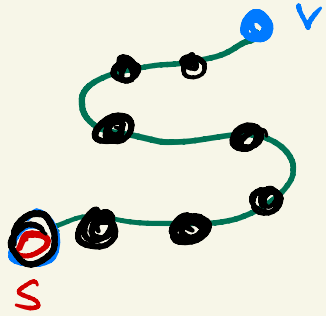
cost $= -\infty$

if we insist on PATH: NP-hard

Proof of correctness : relative loop invariant

R_4 : $\forall v \in V$ current cost $cost(v) =$

min cost { $c(s \to v)$ path through black vertices }



R_3 : if $status(v) = BLACK$
$status(w) \neq BLACK$
then $c(v) \leq c(w)$

IMPLEMENTING PRIORITY QUEUE

create empty P.Q.

add item to P.Q.

EXTRACT-MIN

UPDATE (key) — INCREASE KEY

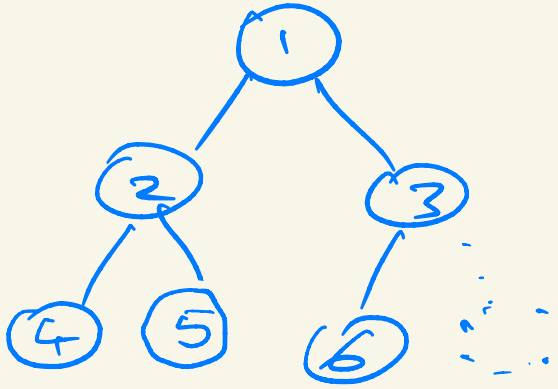
DECREASE KEY

item = (ID, key)

min by key

balanced binary tree with n vertices

fill levels
left to right →



given n, topology uniquely determined

d = depth of balanced binary tree

$$2^d \le n < 2^{d+1}$$

$$d \le \log_2 n < d+1$$

$$\boxed{d = \lfloor \log_2 n \rfloor}$$

HEAP - : implementation of PRIORITY QUEUE

balanced binary tree, keys at nodes

HEAP CONDITION $key(p(v)) \leq key(v)$

DECREASE-KEY(v, newkey) "bubbling up"

```
if newkey < key(v)
    key(v) := newkey
while key(p(v)) > key(v)
    swap v ⟷ p(v)
```

cost $\leq d$ comparisons

$O(d)$ operations

INCREASE-KEY(v, newkey) "bubbling down"

```
while key(v) > max(key(left child(v)), right child(v))
    if key(left child(v)) > key(right child(v))
        swap v ⟷ right child(v)
    else swap v ⟷ left child(v)
```

cost $\leq 2d$ comparisons

$O(d)$ operation

EXTRACT-MIN

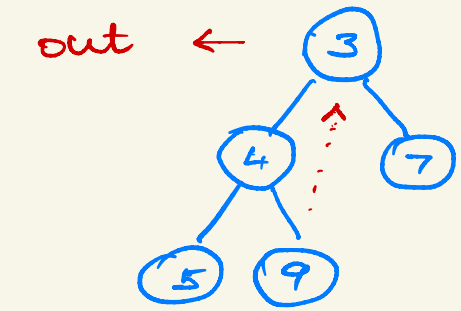
out := key(root)

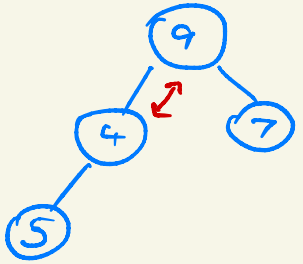
key(root) := key(n)

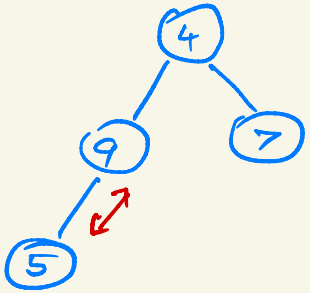
n := n − 1

bubble-down(root)

return out





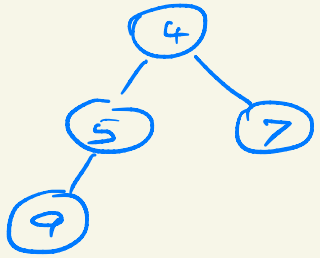


INSERT (newkey)

n := n + 1

key(n) := newkey

bubble-up(n)



COST OF DIJKSTRA

INSERT $\leq n$ times

EXTRACT-MIN $\leq n$ times

DECREASE-KEY $\leq m$ times

using HEAP implementation of PRIORITY QUEUE

$$\#\text{ comparisons} \leq nd + n \cdot 2d + m \cdot d = (3n+m)d \leq (3n+m)\log_2 n$$

total cost $O((n+m)\log_2 n)$