Heap

change_priority(5, 25)
change_priority(0, 30)

Heap:

loc: 0 2 0 6 4 2 3 1 3 5 6

id: 0 1 2 3 4 5 6 7 8 9

priority: 20 16 15 10 7 4 6 3 8 17

enqueue: add at next available spot
enqueue(18, 18)

O(log n) → while curr not at root and curr > parent
1. swap (curr, parent)
2. move curr up one level

heap-reheap-up

left(i) = 2i + 1
right(i) = 2i + 2
parent = \lfloor i/2 \rfloor

degree:
gain max from root O(1)
mov last leaf to root O(1)
O(log n) while curr not leaf and curr < one child
swap w/ largest child
O(1)
mov curr down to where swap was
O(log n) total
heap-reheap-down
don’t need to simulate horses between cities - just need arrival/departure times

don’t care about which horses arrive at a city at a given time, just that horses arrive once horses leave city X, don’t bother with horses that leave/arrive later

keep track of next horses to arrive at each city
simulate departure of horses at next arrival

use priority queue: elements are cities
priorities are arrival times
min queue
Dijkstra's Algorithm

for each vertex v
  d(v) = \infty
  pred[v] = NULL
  color[v] = in queue

precondition: no neg weight edges

Q ← \{ start \}  // min priority queue of all vertices with priorities given by d[v]

color[start] ← in queue
d[start] ← 0
pred[start] ← NULL

while Q not empty
  u ← dequeue(Q)  // the vertex with min priority among those still in queue
  for each outneighbor v of u
    if color[v] = unseen in queue
      enqueue(Q, v)  // change-priority(Q, v, d[u] + w(u, v))
      color[v] ← in queue
      pred[v] ← u
      d[v] ← d[u] + w(u, v)
    color[u] ← done
Breadth-First Search

for each \( v \) in iterations
  \( \text{color}[v] \leftarrow \text{in queue} \)
  \( \text{pred}[v] \leftarrow \text{NULL} \)
  \( d[v] \leftarrow \infty \) \( \mathcal{O}(1) \)

\( d[\text{start}] \leftarrow 0 \)
\( \text{pred}[\text{start}] \leftarrow \text{NULL} \) \( \mathcal{O}(1) \)

\( Q \leftarrow \text{new PQ with els } 0, \ldots, n-1 \), priorities given by \( d \)

\( O(n) \)

while \( Q \) not empty in iterations
  \( u \leftarrow \text{dequeue} \ (Q) \)
  for each outneighbor \( v \) of \( u \)
    if \( \text{color}[v] = \text{in queue} \) and \( d[v] > d[u] + w(u,v) \)
      change-priority \((Q, v, d[u] + w(u,v))\) \( O(\log n) \)
      \( d[v] \leftarrow d[u] + w(u,v) \)
      \( \text{color}[v] \leftarrow \text{done} \)
    \( \mathcal{O}(1) \)

\( \text{total: } O(\left( \log n \right) + \left( \log n \right) \) \)

assuming \( m \geq n-1 \) \( \mathcal{O}(m \log n) \) \( \mathcal{O}(n^2 + m) \) better than \( m \approx n^2 \)