breadth-first search

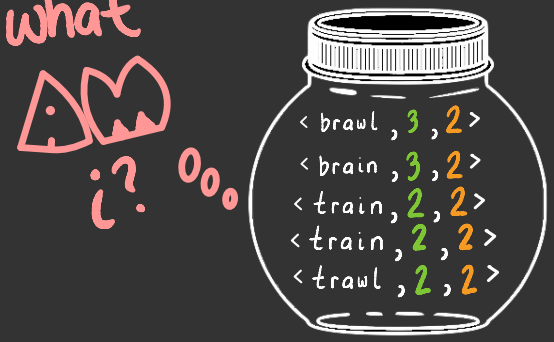
14 sept 2022

CSCI 373

SEARCH$(M = (Q, \Sigma, \Delta, q_0, F, w), H)$:

- container = new Container()
- container.put($\langle q_0, 0, H(q_0) \rangle$)
- repeat:
  - if container.empty() then return ∞
  - n = container.get()
  - if $q(n) \in F$ then return $g(n)$
  - let $\text{successors}_{M,H}(n) = \{ \langle q', g(n) + w(q(n), \sigma, q'), H(q') \rangle \mid (q(n), \sigma, q') \in \Delta \}$
  - for $n' \in \text{successors}_{M,H}(n)$:
    container.put(n')

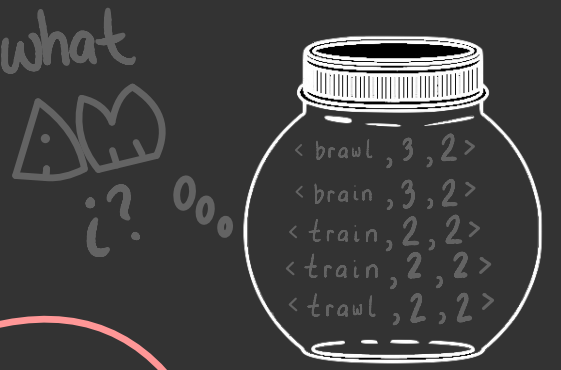
"visit" a node



SEARCH($M = (Q, \Sigma, \Delta, q_0, F, w)$, H):

- container = new Container()
- container.put($\langle q_0, 0, H(q_0) \rangle$)
- repeat:
  - if container.empty() then return ∞
  - n = container.get()
  - if $q(n) \in F$ then return $g(n)$
  - let $\text{successors}_{M,H}(n) = \{ \langle q', g(n) + w(q(n), \sigma, q'), H(q') \rangle \mid (q(n), \sigma, q') \in \Delta \}$
  - for $n' \in \text{successors}_{M,H}(n)$:
    container.put(n')

"visit" a node

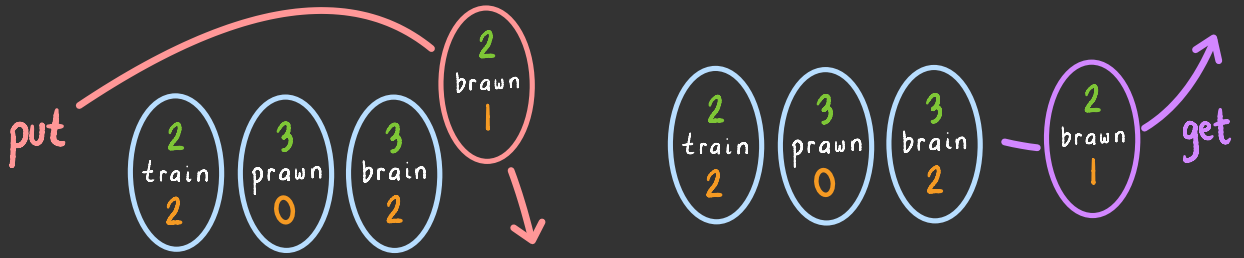


a container is a data structure that supports these methods

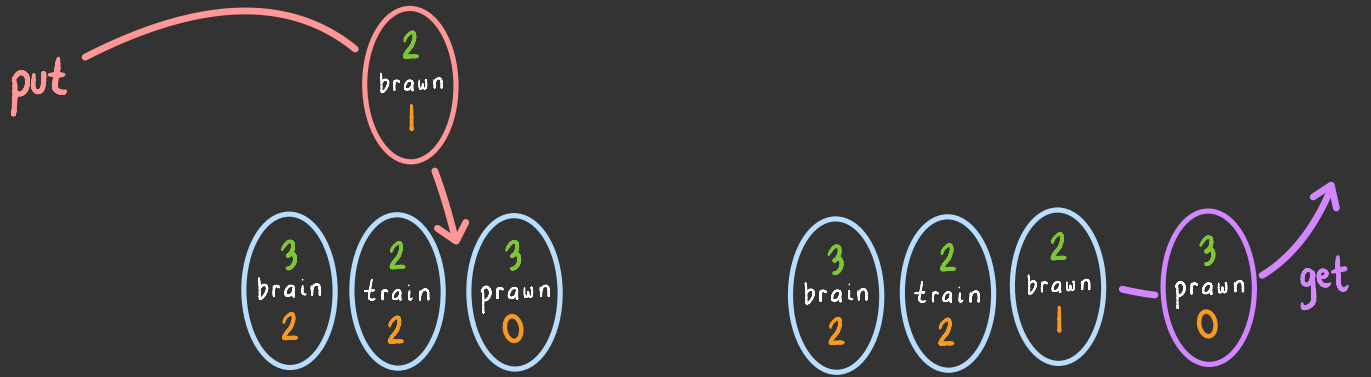
example containers

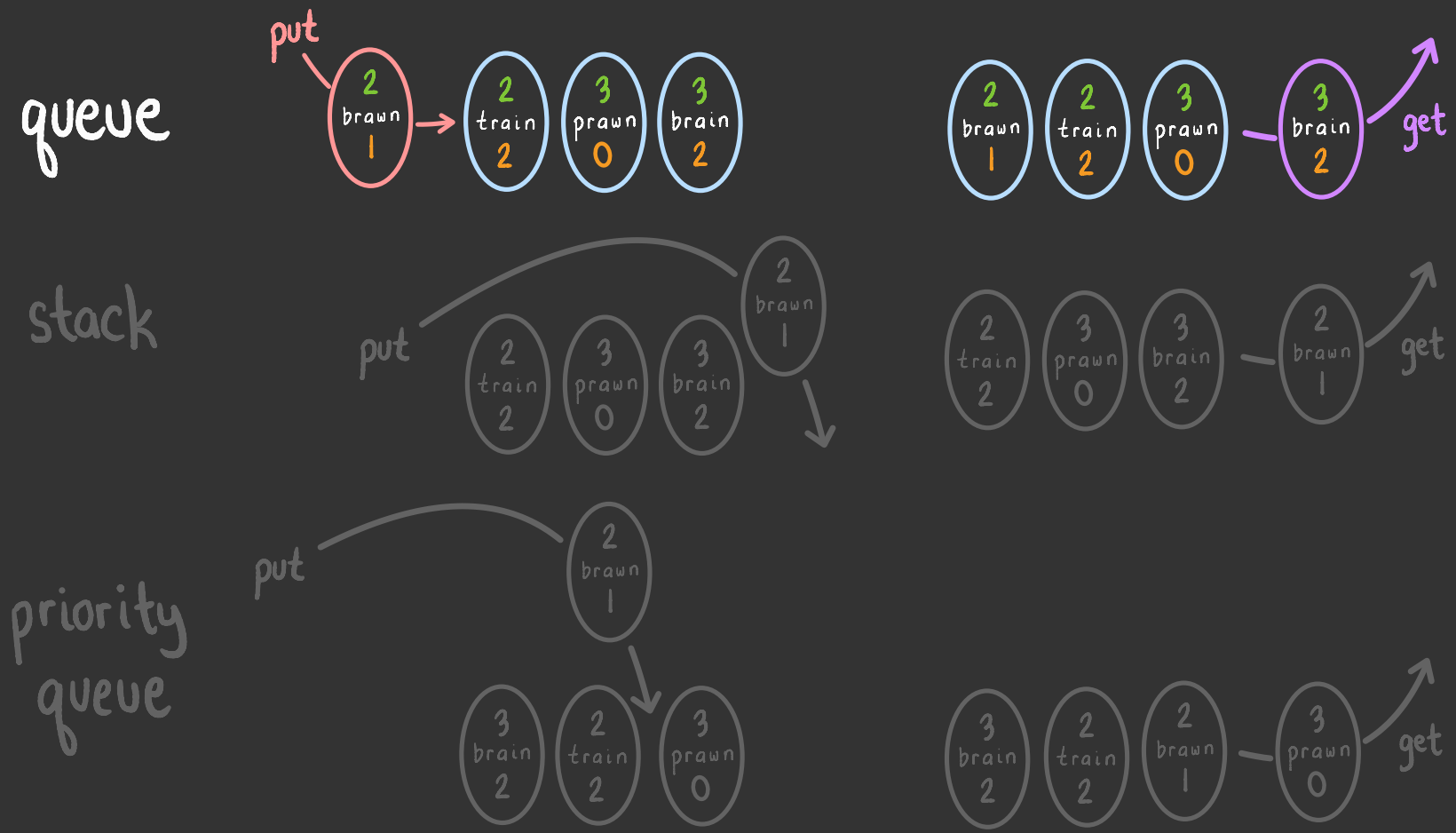
queue

stack



priority queue

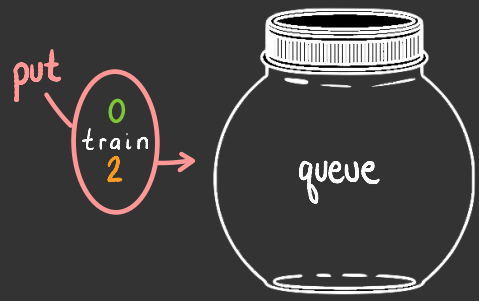


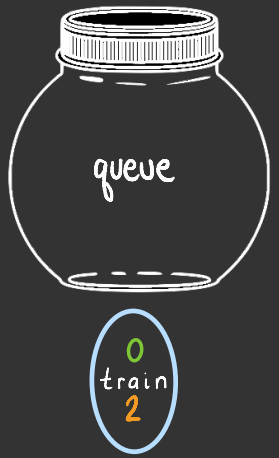

queue
put
2
brawn
1
2
train
2
3
prawn
0
3
brain
2
2
brawn
1
2
train
2
3
prawn
0
3
brain
2
get
stack
put
2
brawn
1
2
train
2
3
prawn
0
3
brain
2
2
train
2
3
prawn
0
3
brain
2
2
brawn
1
get
priority queue
put
2
brawn
1
3
brain
2
2
train
2
3
prawn
0
3
brain
2
2
train
2
2
brawn
1
3
prawn
0
get

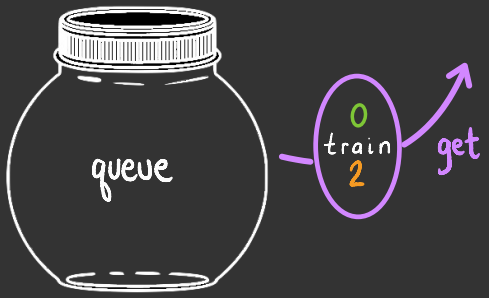
SEARCH$(M = (Q, \Sigma, \Delta, q_0, F, w), H)$:

- container = new queue()
- container.put($\langle q_0, 0, H(q_0) \rangle$)
- repeat:
  - if container.empty() then return ∞
  - n = container.get()
  - if $q(n) \in F$ then return $g(n)$
  - let $\text{successors}_{M,H}(n) = \{ \langle q', g(n) + w(q(n), \sigma, q'), H(q') \rangle \mid (q(n), \sigma, q') \in \Delta \}$
  - for $n' \in \text{successors}_{M,H}(n)$:
    container.put(n')

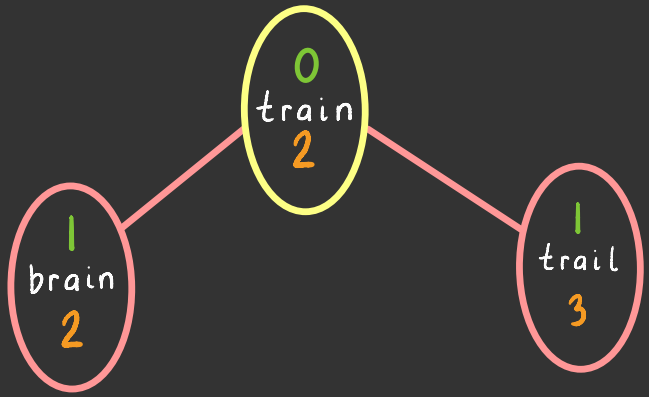
"visit" a node

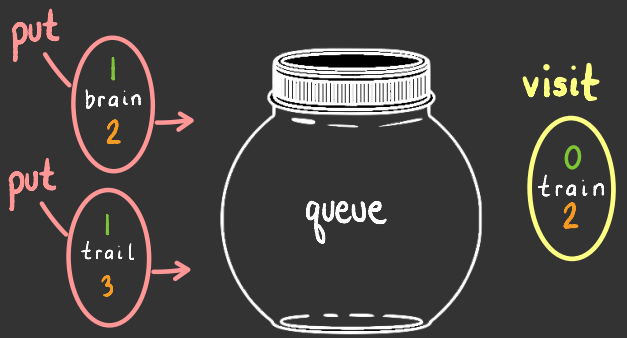


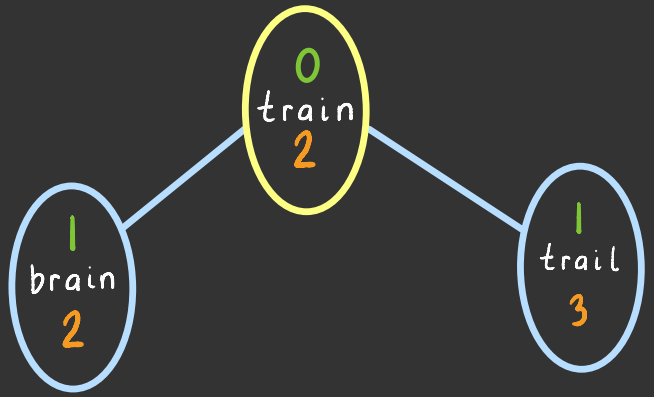

queue
0
train
2

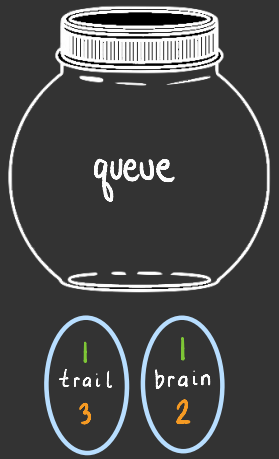

queue
0
train
2
get

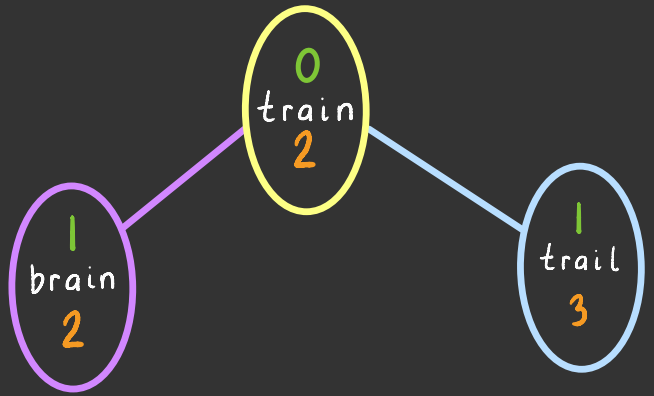
visit

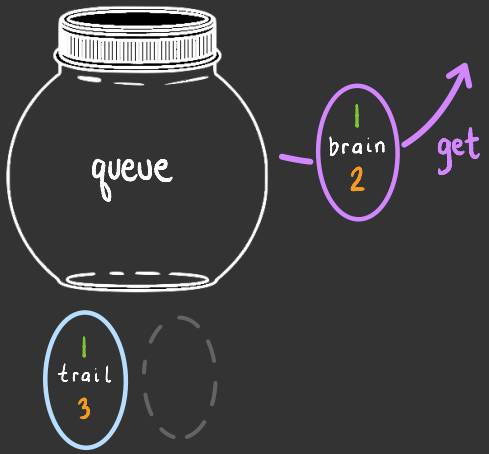

0
train
2
1
brain
2
1
trail
3

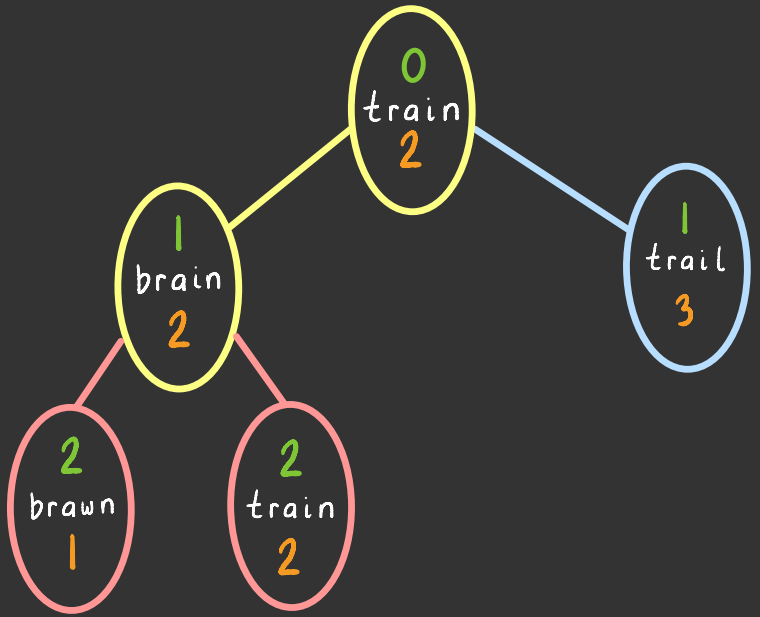

put
1
brain
2
put
1
trail
3
queue
visit
0
train
2

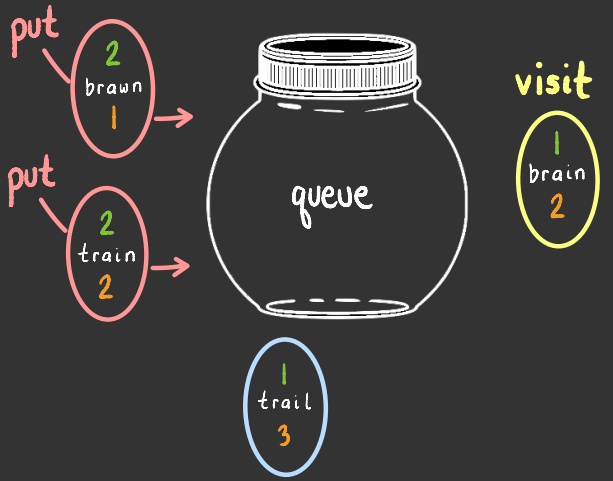

0
train
2
1
brain
2
1
trail
3

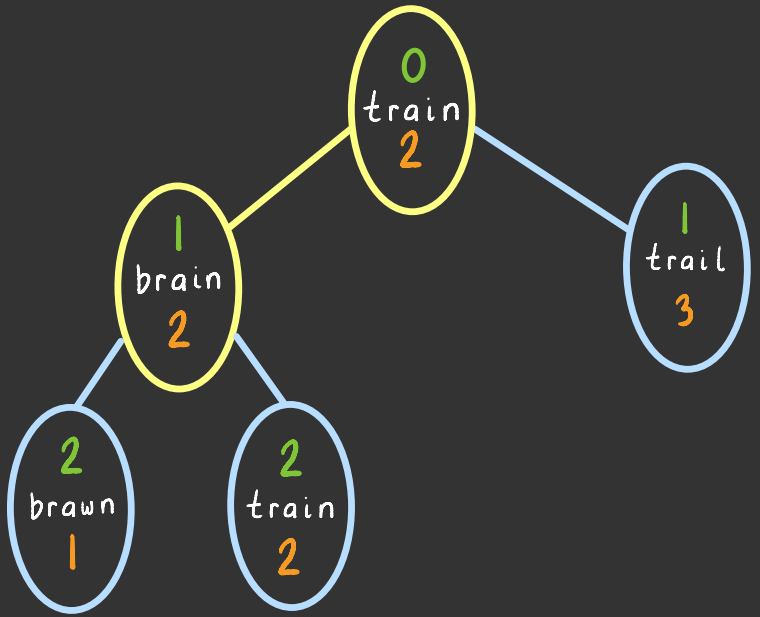

queue
1
trail
3
1
brain
2


0
train
2
1
brain
2
1
trail
3

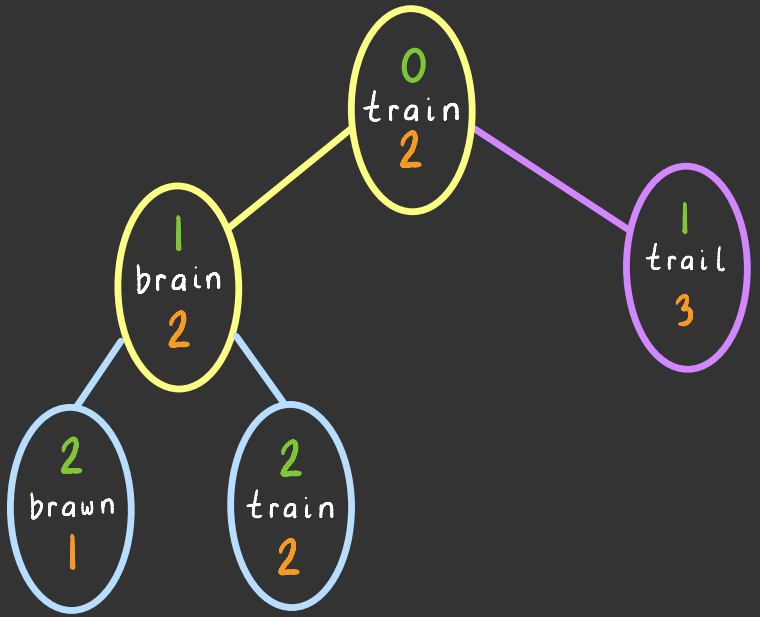

queue
1
brain
2
get
1
trail
3

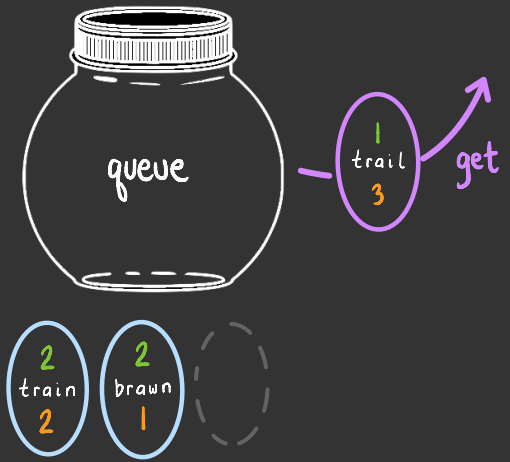

0
train
2
1
brain
2
1
trail
3
2
brawn
1
2
train
2

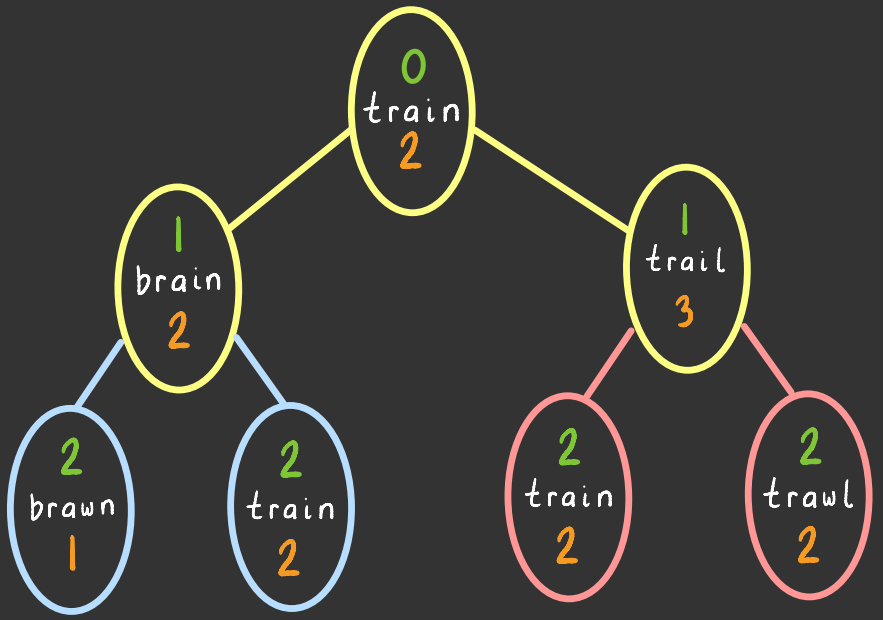

put
2
brawn
1
put
2
train
2
queue
visit
1
brain
2
1
trail
3

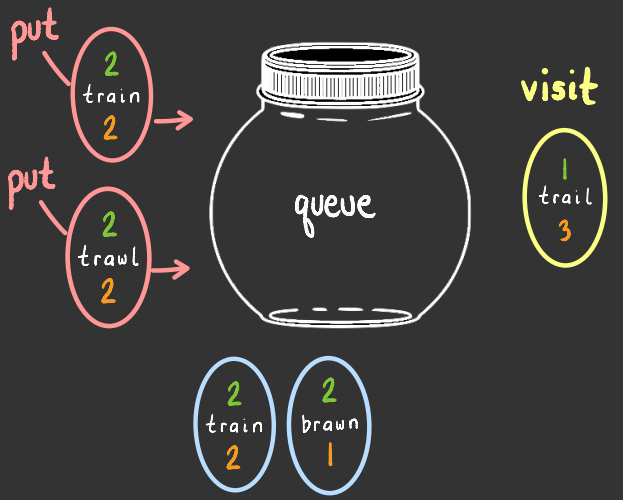

0
train
2
1
brain
2
1
trail
3
2
brawn
1
2
train
2

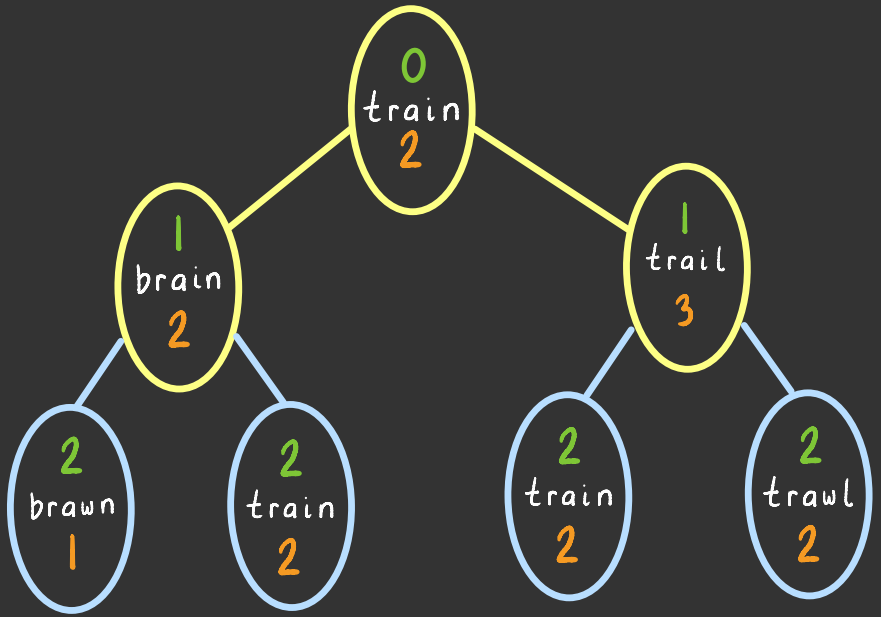

queue
2
train
2
2
brawn
1
1
trail
3

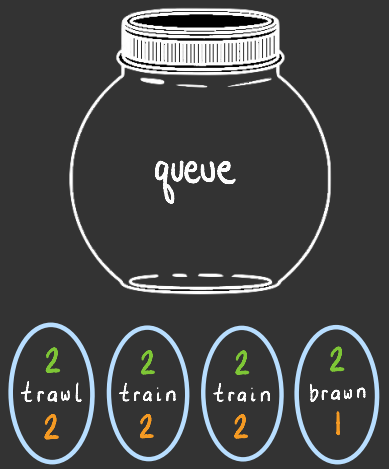

0
train
2
1
brain
2
1
trail
3
2
brawn
1
2
train
2

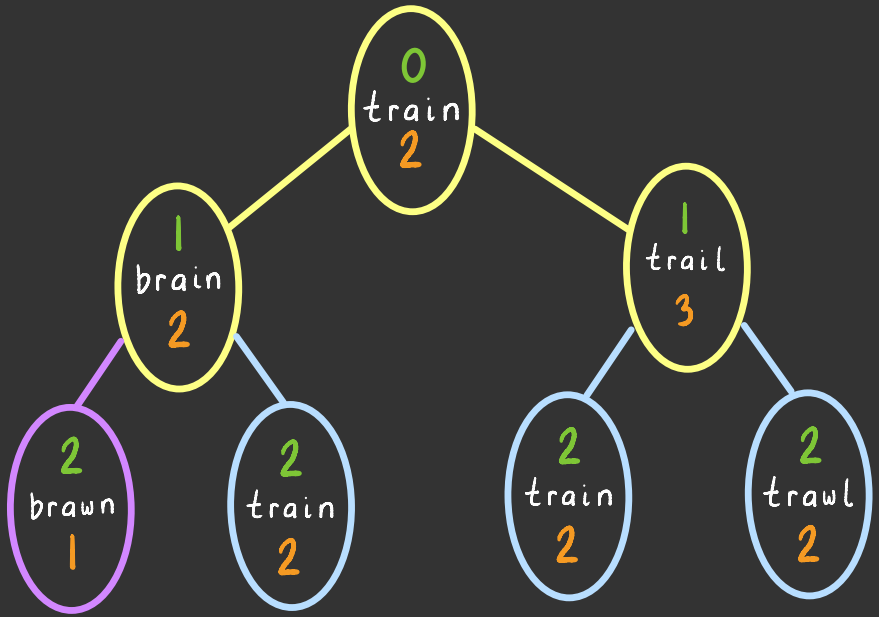

queue
1
trail
3
get
2
train
2
2
brawn
1

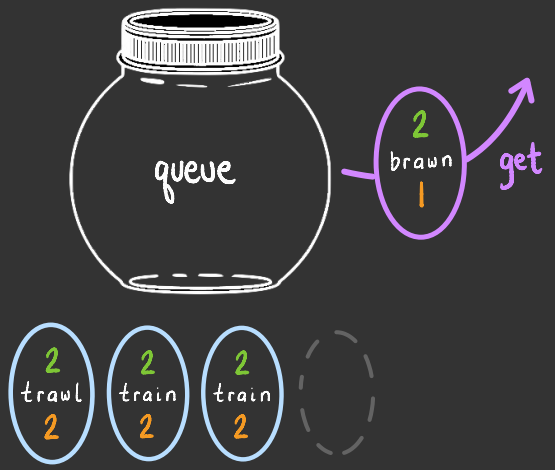

0
train
2
1
brain
2
1
trail
3
2
brawn
1
2
train
2
2
train
2
2
trawl
2

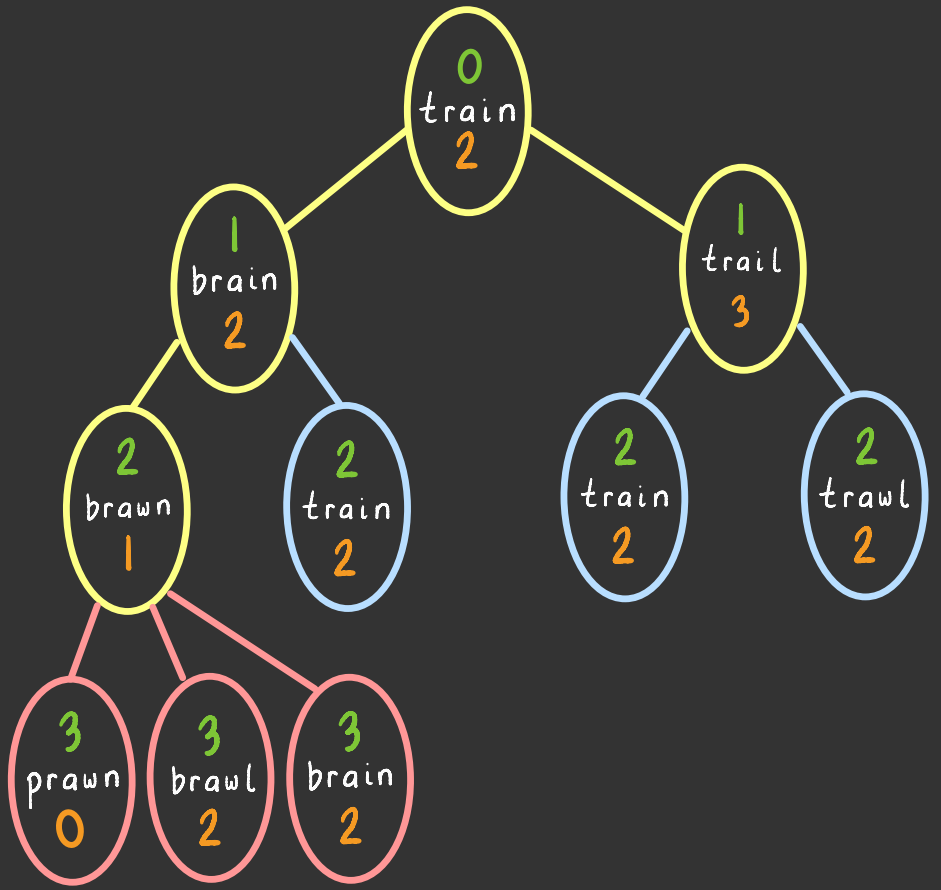

put
2
train
2
put
2
trawl
2
queue
visit
1
trail
3
2
train
2
2
brawn
1

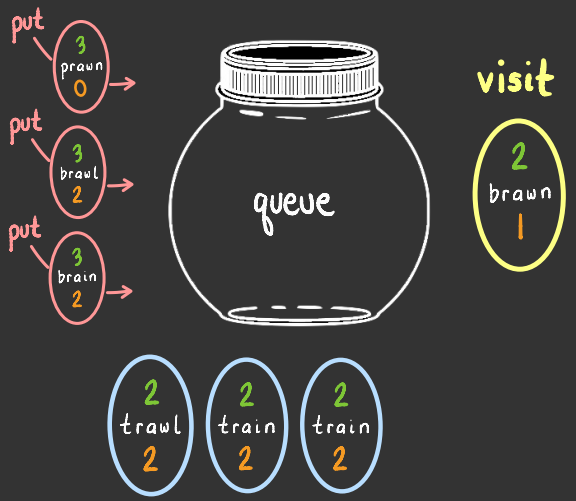

0
train
2
1
brain
2
1
trail
3
2
brawn
1
2
train
2
2
train
2
2
trawl
2

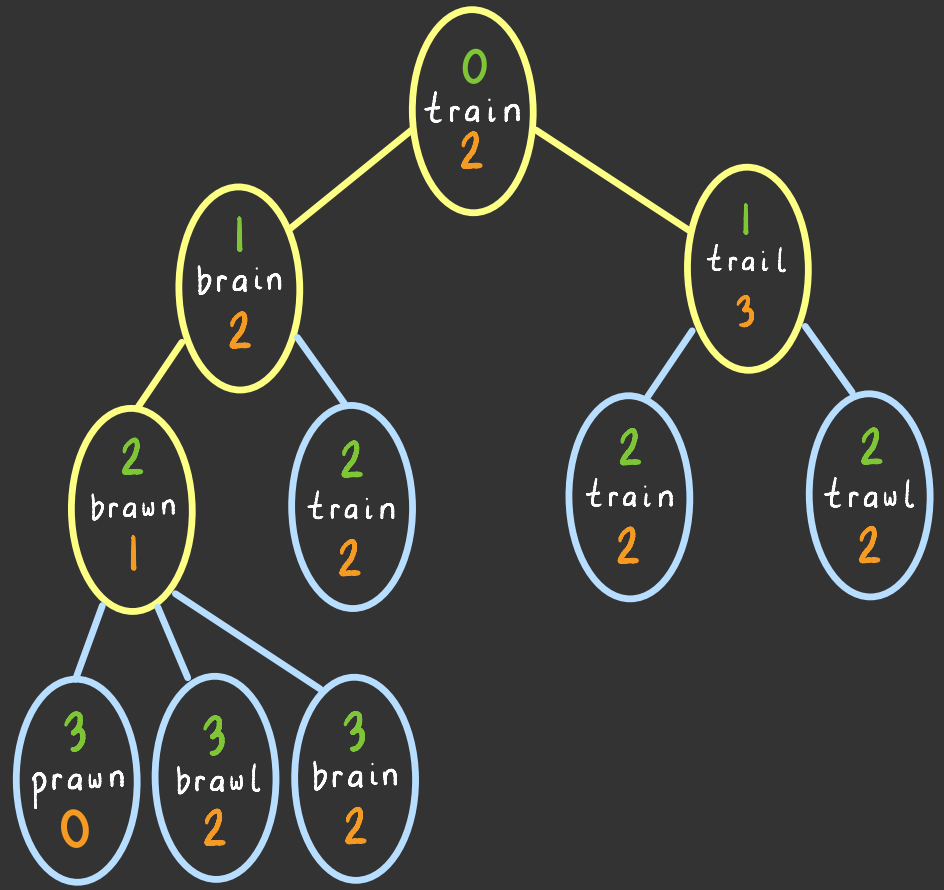

queue
2
trawl
2
2
train
2
2
train
2
2
brawn
1

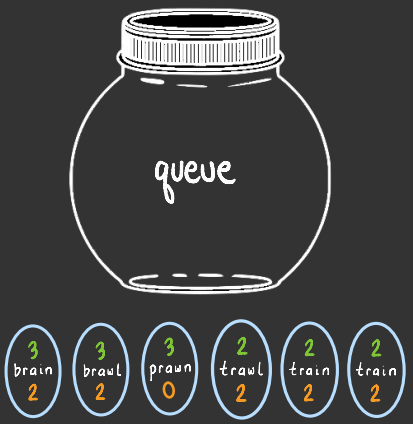

0
train
2
1
brain
2
1
trail
3
2
brawn
1
2
train
2
2
train
2
2
trawl
2

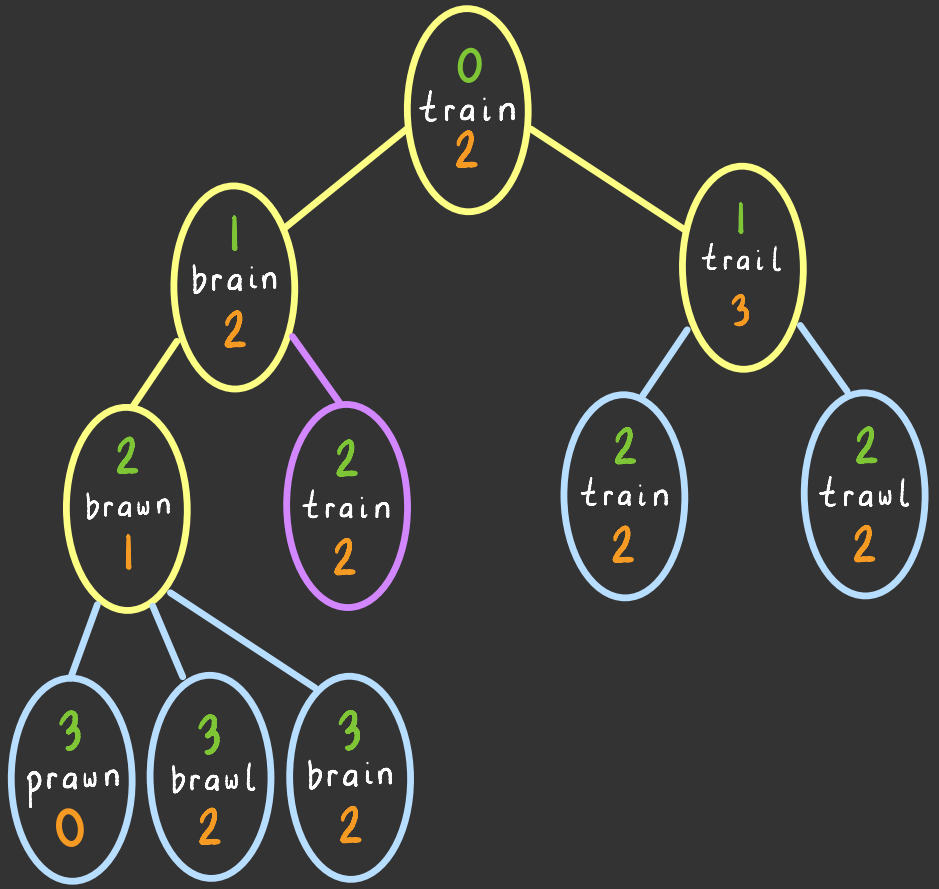

queue
2
brawn
1
get
2
trawl
2
2
train
2
2
train
2

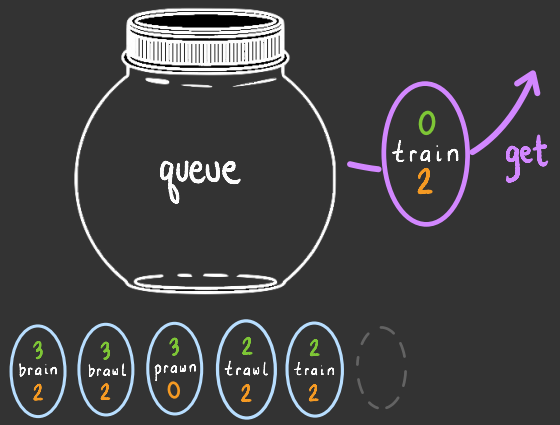

0
train
2
1
brain
2
1
trail
3
2
brawn
1
2
train
2
2
train
2
2
trawl
2
3
prawn
0
3
brawl
2
3
brain
2

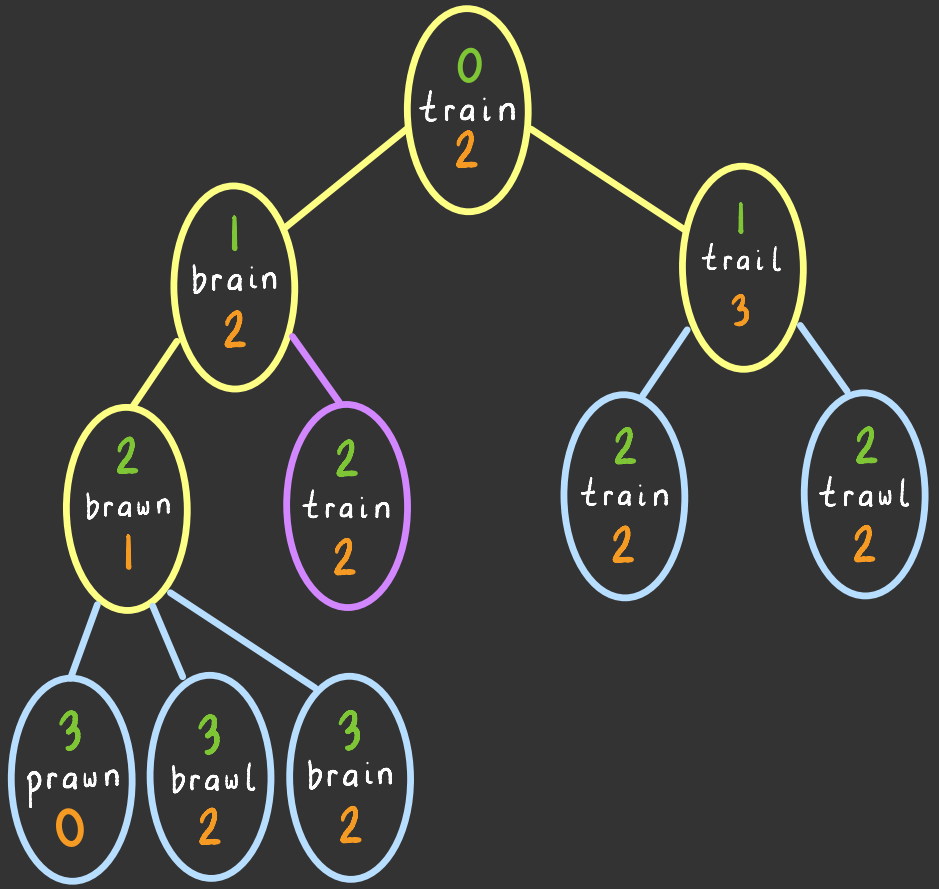

put
3
prawn
0
put
3
brawl
2
put
3
brain
2
queue
visit
2
brawn
1
2
trawl
2
2
train
2
2
train
2

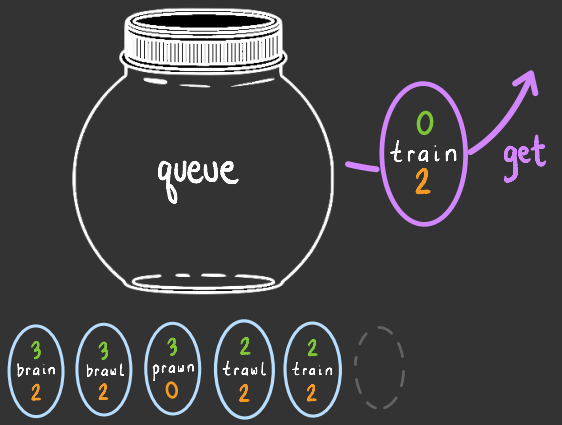

0
train
2
1
brain
2
1
trail
3
2
brawn
1
2
train
2
2
train
2
2
trawl
2
3
prawn
0
3
brawl
2
3
brain
2


queue
3
brain
2
3
brawl
2
3
prawn
0
2
trawl
2
2
train
2
2
train
2


0
train
2
1
brain
2
1
trail
3
2
brawn
1
2
train
2
2
train
2
2
trawl
2
3
prawn
0
3
brawl
2
3
brain
2

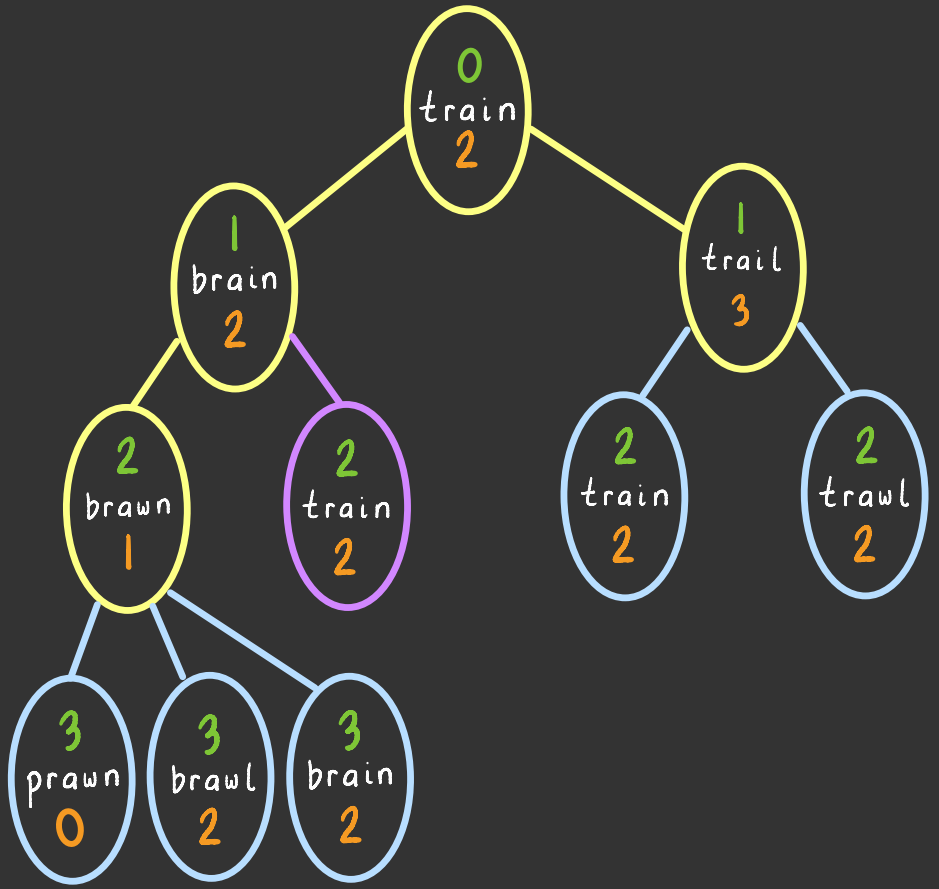

queue
0
train
2
get
3
brain
2
3
brawl
2
3
prawn
0
2
trawl
2
2
train
2

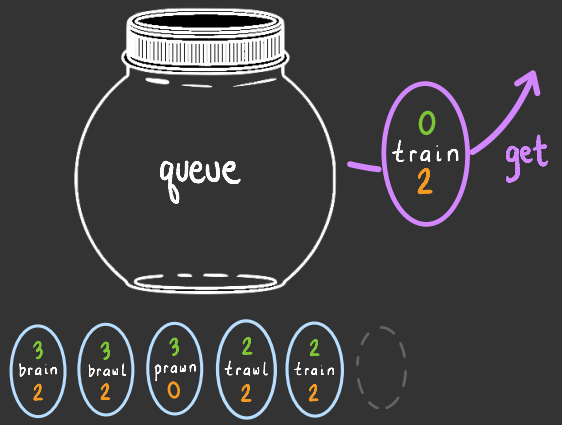




in what order are the nodes visited?

your answer here





in what order are the nodes visited?

if $j < k$, then every node at search depth j is visited before any node at search depth k

SEARCH($M = (Q, \Sigma, \Delta, q_0, F, w)$, H):

- container = new queue()
- container.put($\langle q_0, 0, H(q_0) \rangle$)
- repeat:
  - if container.empty() then return ∞
  - n = container.get()
  - if $q(n) \in F$ then return $g(n)$
  - let $\text{successors}_{M,H}(n) = \{ \langle q', g(n) + w(q(n), \sigma, q'), H(q') \rangle \mid (q(n), \sigma, q') \in \Delta \}$
  - for $n' \in \text{successors}_{M,H}(n)$:
    container.put(n')

"visit" a node

if we use a 

as the container, the algorithm is called breadth-first search, abbreviated BFS