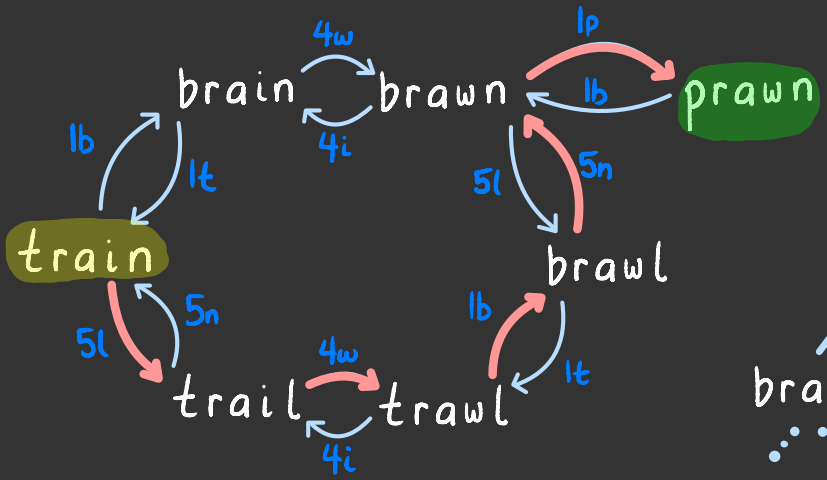
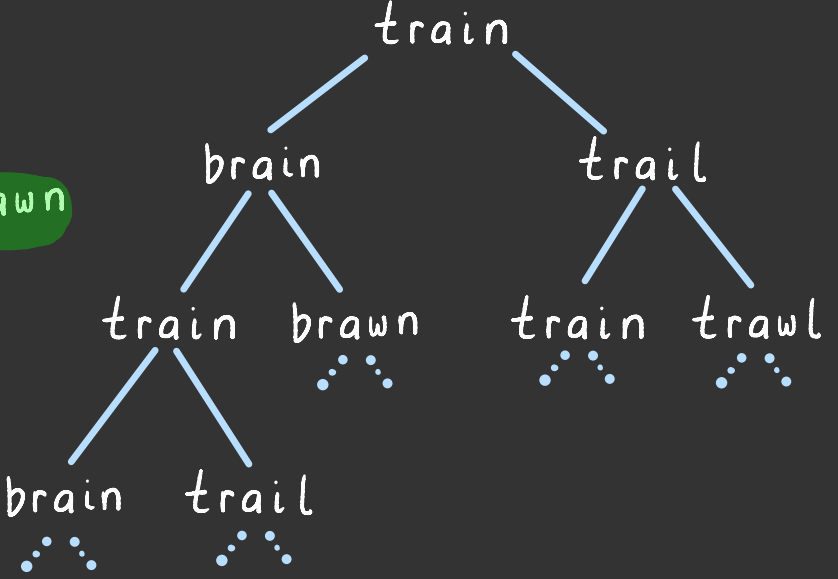
uninformed search

14 sept 2022

CSCI 373

a state machine can be unraveled into a search tree





the search tree is a compact representation of all the search paths

a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

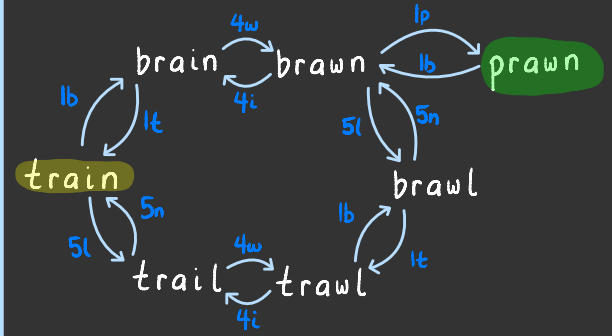
- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate, i.e. a guess at the minimal cost of reaching a final state from state q

helpful notation: for search node $n = \langle q, g, h \rangle$, we will refer to its components by defining $q(n) = q$, $g(n) = g$, $h(n) = h$

a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:

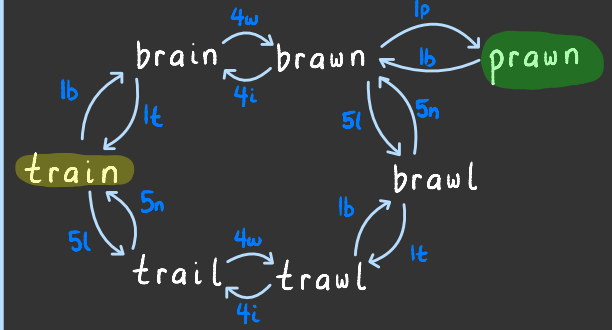


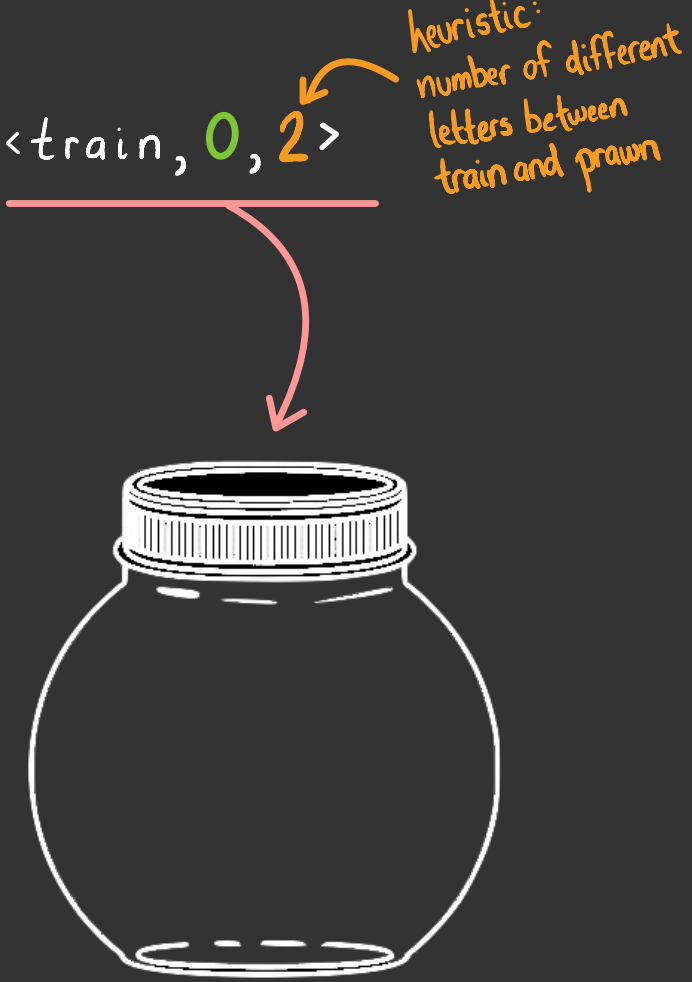
the goal of search:
to find a search path of minimal cost whose last transition leads to a final state

a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:

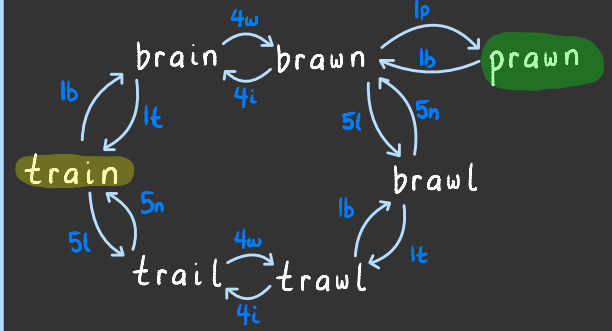




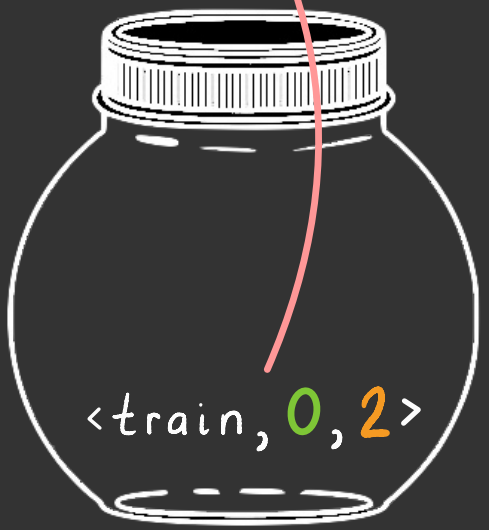
a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:



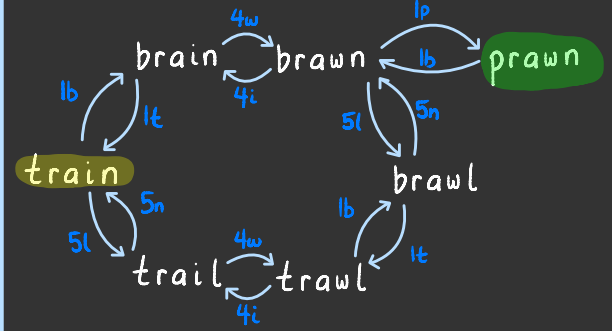
draw a node from the container

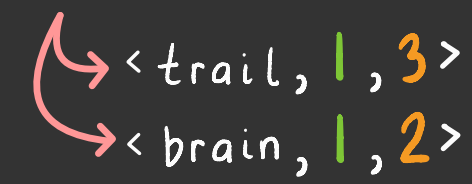


a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:

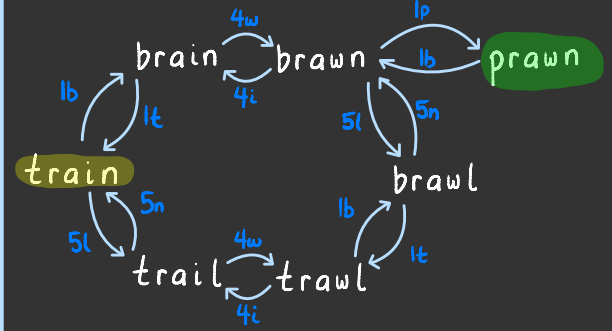


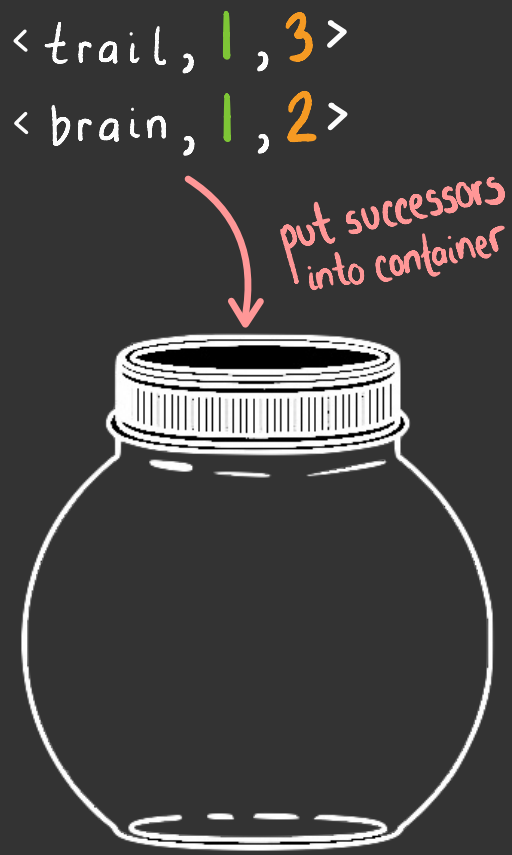


a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:

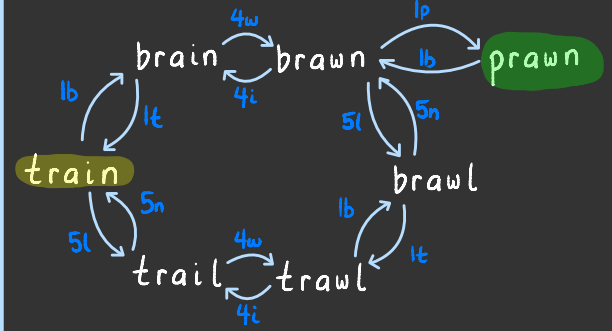


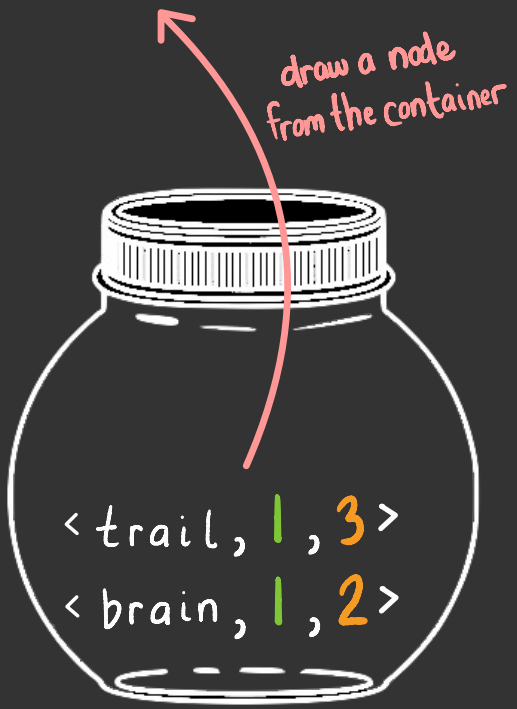


a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

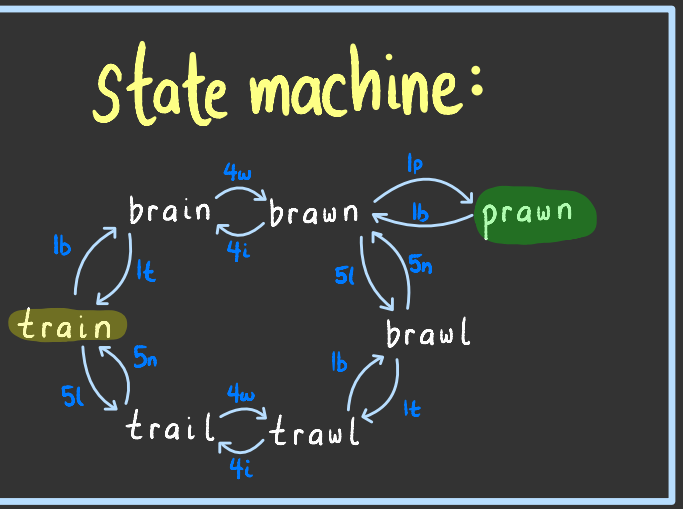
State machine:





a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

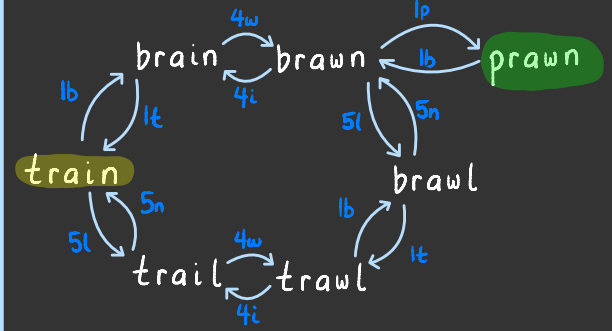
- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate



a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:



<train, 2, 2>

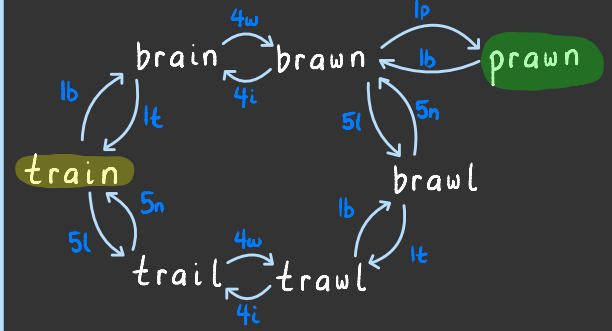
<trawl, 2, 2>

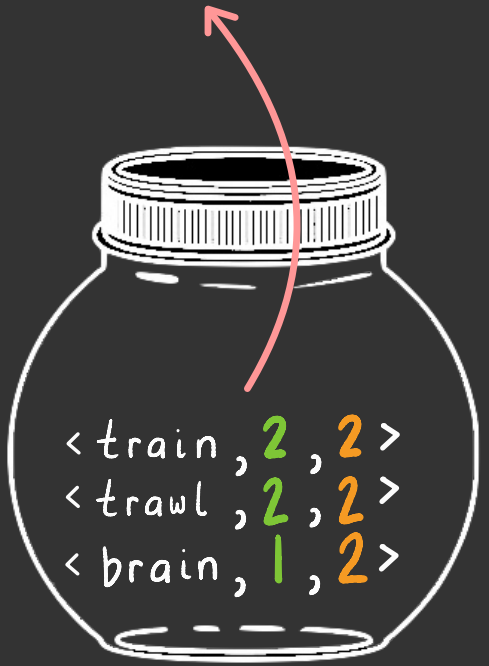
<brain, 1, 2>

a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:

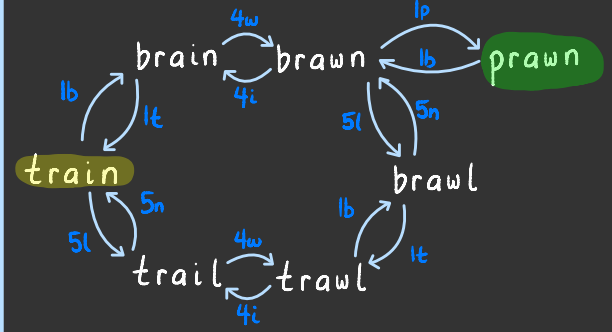




a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

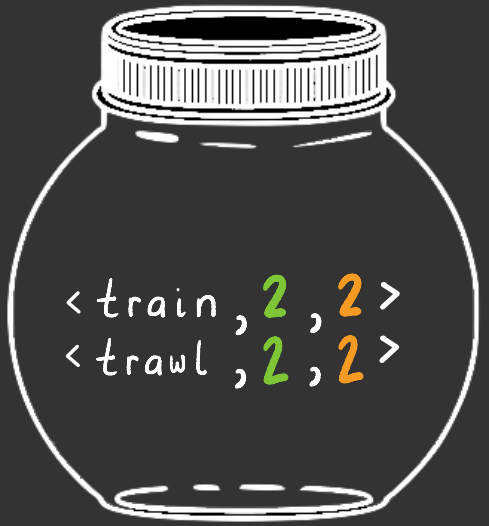
State machine:



$\langle$ brain , 1 , 2 $\rangle$

your answer here

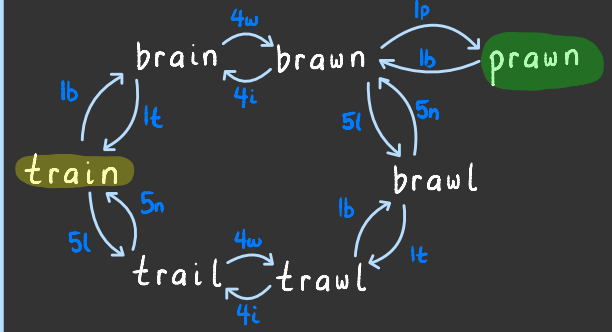
derive successors

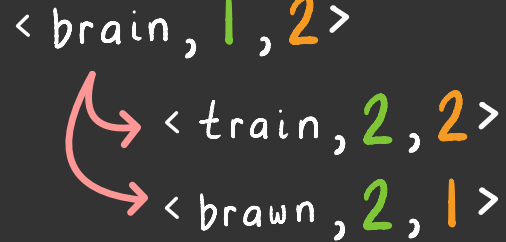


a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

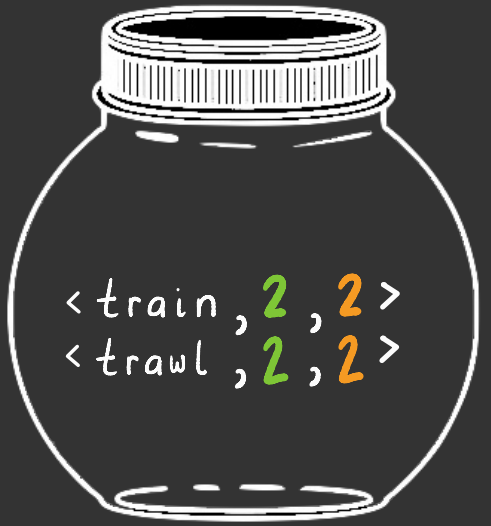
- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:





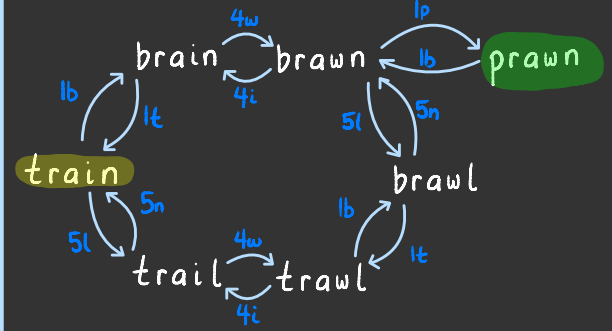
derive successors

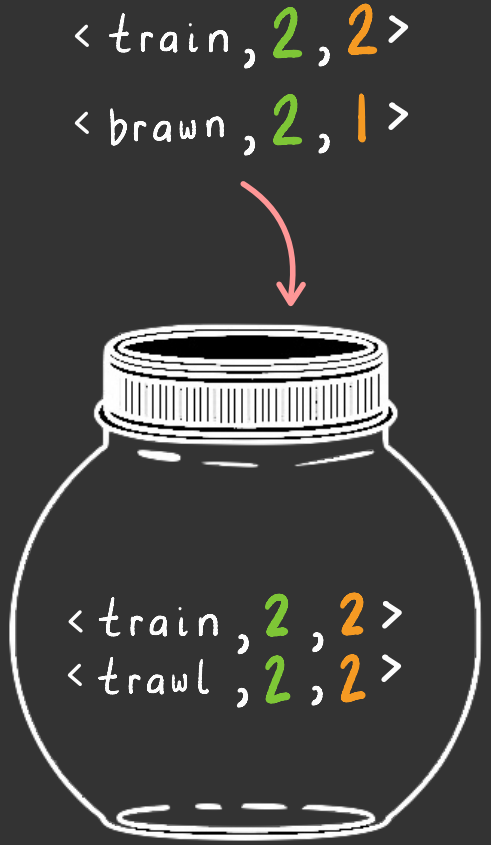


a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:

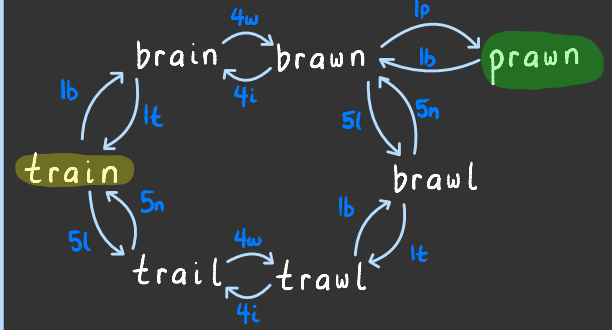


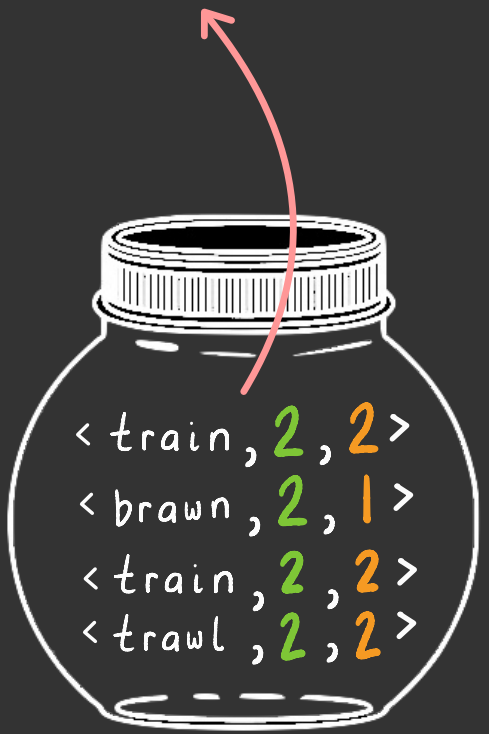


a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:

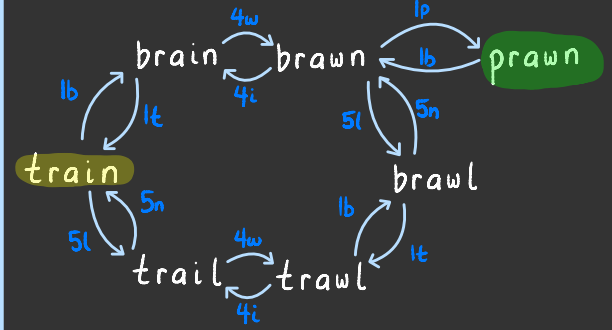


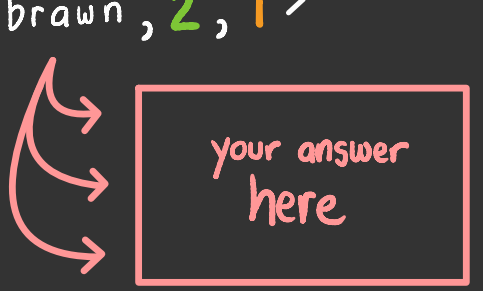


a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:



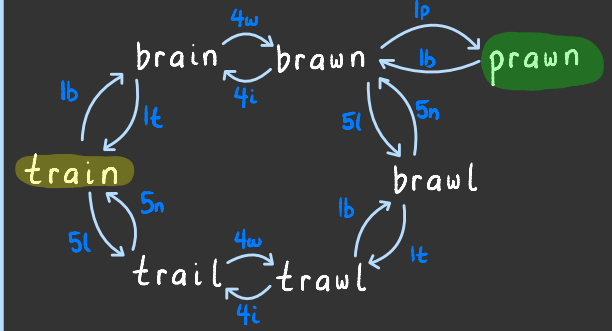


derive successors

a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:



$\langle$ brawn , 2 , 1 $\rangle$

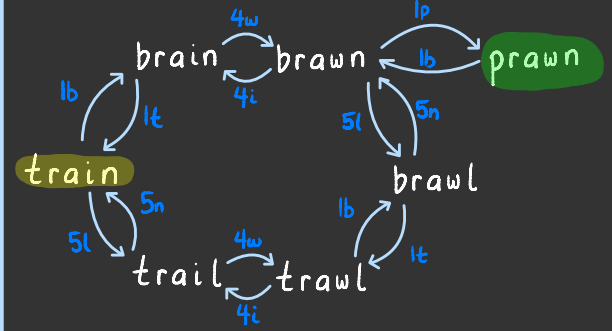
derive successors

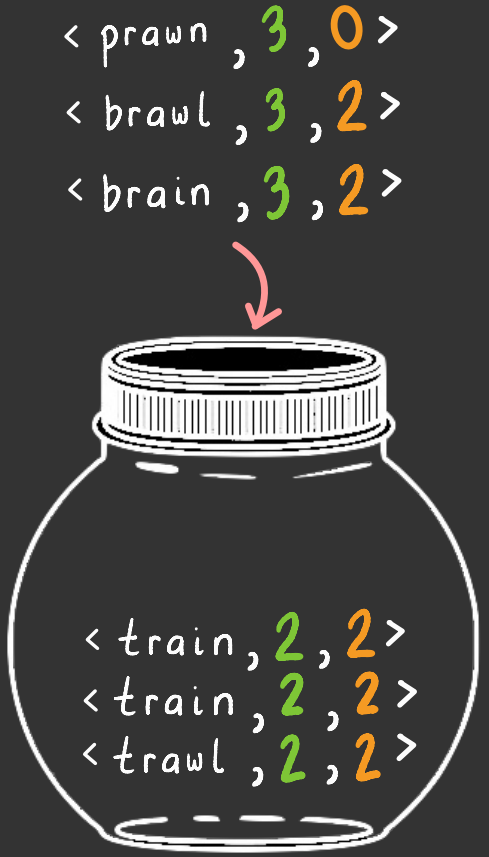
- $\langle$ prawn , 3 , 0 $\rangle$
- $\langle$ brawl , 3 , 2 $\rangle$
- $\langle$ brain , 3 , 2 $\rangle$

a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:

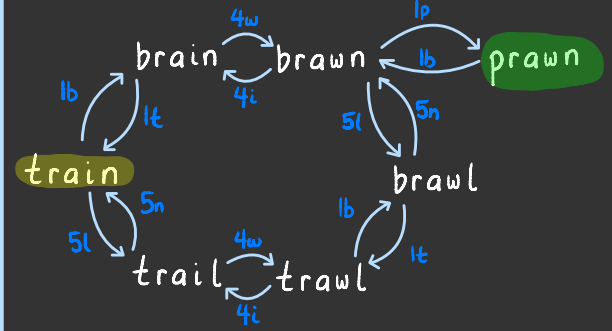


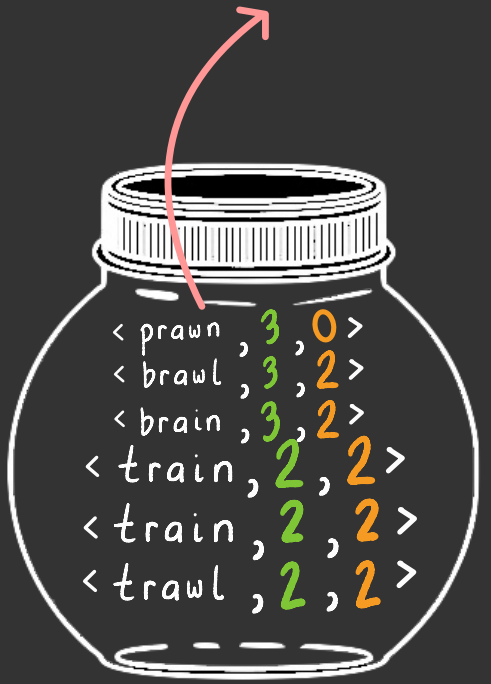


a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:

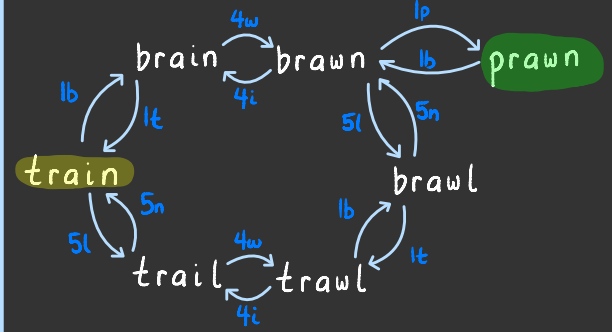




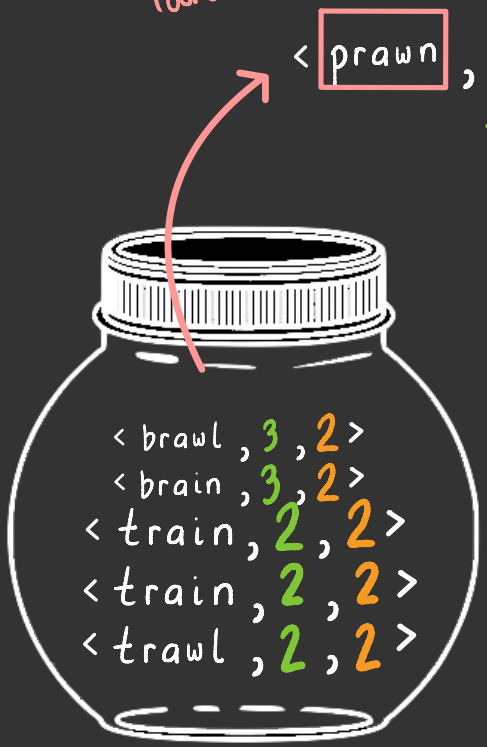
a search node of state machine $(Q, \Sigma, \Delta, q_0, F)$ is a triple $\langle q, g, h \rangle$ where:

- $q \in Q$ is a state
- $g \in \mathbb{R}$ is the cost of the node
- $h \in \mathbb{R}$ is a heuristic estimate

State machine:



found a final state!



this is the **cost** of the solution we found

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')

draw a node from the container

derive successors

put successors into container

function BEST-FIRST-SEARCH(*problem*, *f*) **returns** a solution node or *failure*
  node ← NODE(STATE=*problem*.INITIAL)
  frontier ← a priority queue ordered by *f*, with *node* as an element
  reached ← a lookup table, with one entry with key *problem*.INITIAL and value *node*
  while not IS-EMPTY(*frontier*) **do**
    node ← POP(*frontier*)
    if *problem*.IS-GOAL(*node*.STATE) **then return** *node*
    for each *child* **in** EXPAND(*problem*, *node*) **do**
      s ← *child*.STATE
      if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**
        reached[*s*] ← *child*
        add *child* to *frontier*
  return *failure*

function EXPAND(*problem*, *node*) **yields** nodes
  s ← *node*.STATE
  for each *action* **in** *problem*.ACTIONS(*s*) **do**
    s' ← *problem*.RESULT(*s*, *action*)
    cost ← *node*.PATH-COST + *problem*.ACTION-COST(*s*, *action*, *s'*)
    yield NODE(STATE=*s'*, PARENT=*node*, ACTION=*action*, PATH-COST=*cost*)

Figure 3.7 The best-first search algorithm, and the function for expanding a node. The data structures used here are described in Section 3.3.2. See Appendix B for **yield**.

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')

draw a node from the container

derive successors

put successors into container

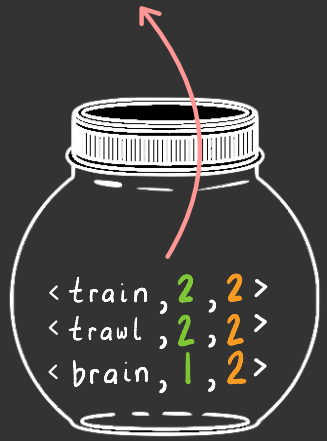
SEARCH$(M = (Q, \Sigma, \Delta, q_0, F, w), H)$:

- 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')

draw a node from the container

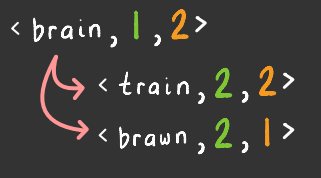
derive successors

put successors into container

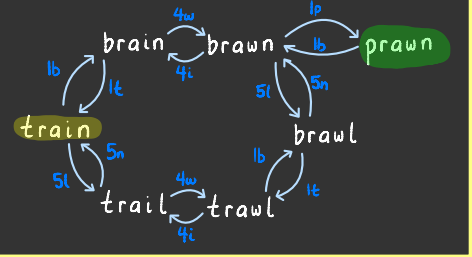


SEARCH($M = (Q, \Sigma, \Delta, q_0, F, w)$, H):

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



State machine:



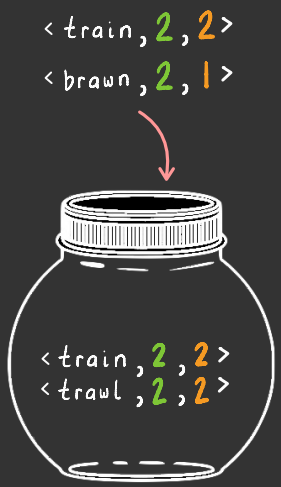
SEARCH$(M=(Q,\Sigma,\Delta,q_0,F,w),H)$:

- 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')

draw a node from the container

derive successors

put successors into container



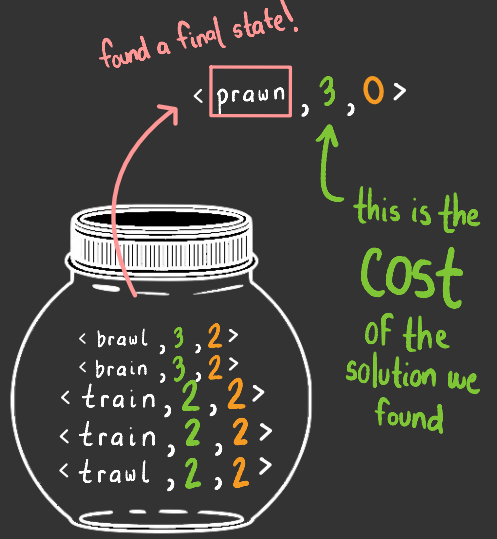
SEARCH$(M = (Q, \Sigma, \Delta, q_0, F, w), H)$:

- 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')

draw a node from the container

derive successors

put successors into container



```
SEARCH(M = (Q, Σ, Δ, q₀, F, w), H):
  container = new Container()
  container.put(<q₀, 0, H(q₀)>)
  repeat:
    if container.empty() then return ∞
    n = container.get()                          ← draw a node from the container
    if q(n) ∈ F then return g(n)
    let successors_{M,H}(n) = { <q', g(n) + w(q(n), σ, q'), H(q')> | (q(n), σ, q') ∈ Δ }   ← derive successors
    for n' ∈ successors_{M,H}(n):                ← put successors into container
      container.put(n')
```

what is the purpose of this line?

your answer here

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')

draw a node from the container

derive successors

put successors into container

what is the purpose of this line?

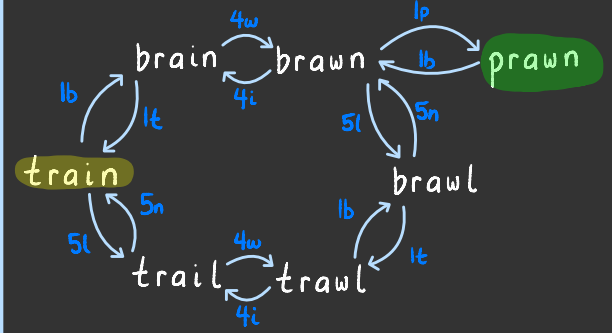
if the container ever becomes empty, then there are no paths from an initial state to a final state

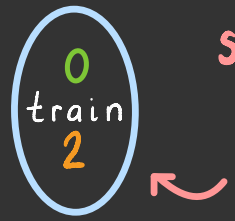
SEARCH$(M = (Q, \Sigma, \Delta, q_0, F, w), H)$:

- 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')

"visiting" (or "expanding") a node

state machine:

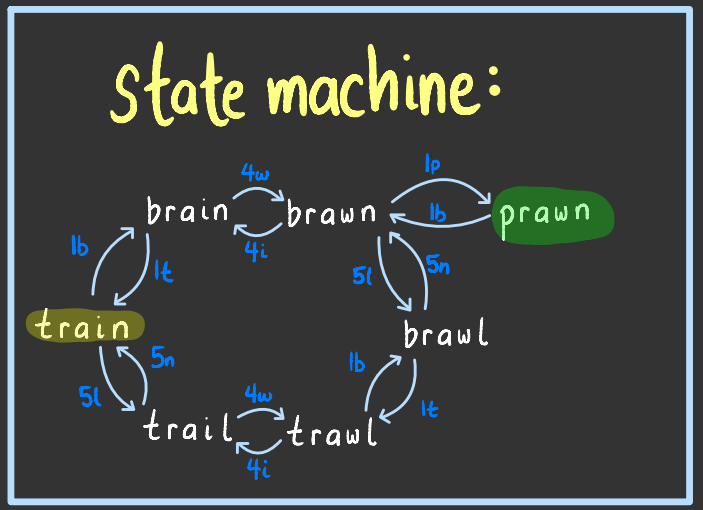


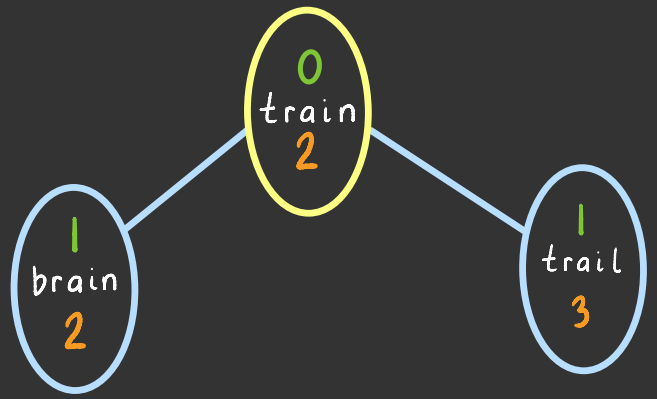


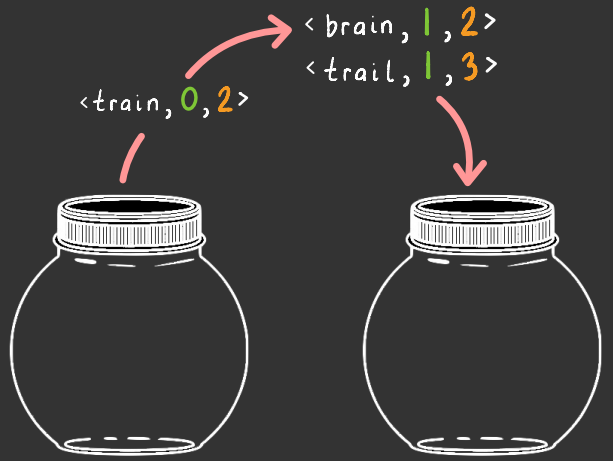
shorthand for

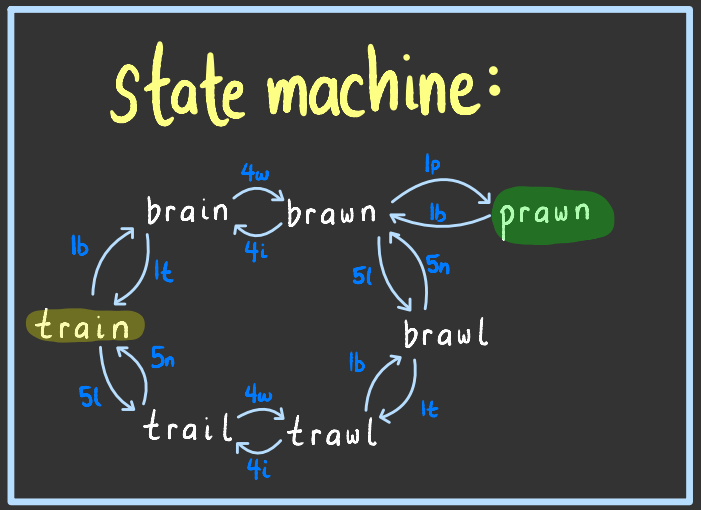
<train, 0, 2>

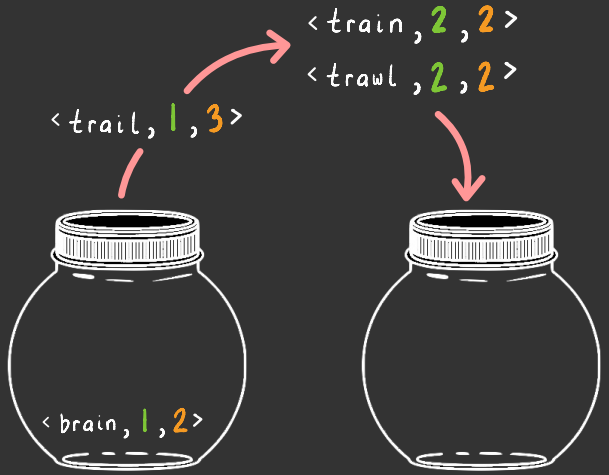
<train, 0, 2>

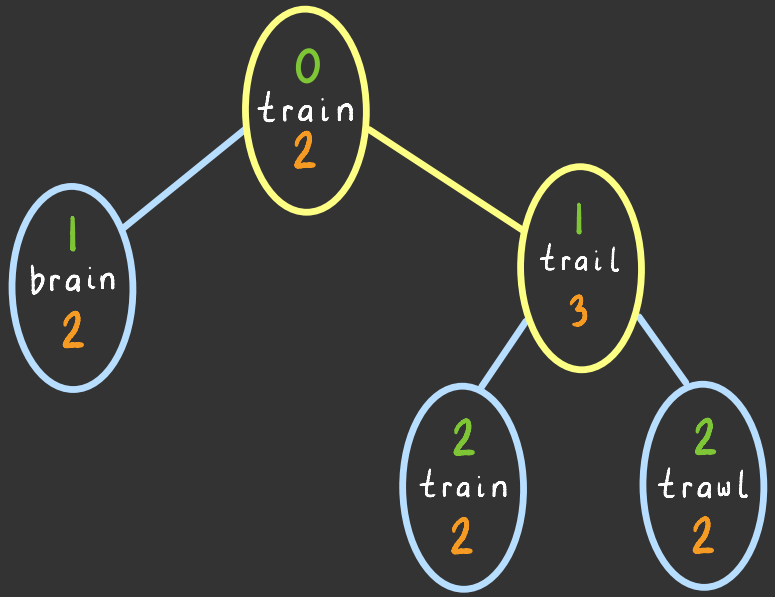

State machine:
brain
brawn
prawn
train
brawl
trail
trawl
4w
4i
1p
1b
1b
1t
5l
5n
5n
5l
1b
1t
4w
4i

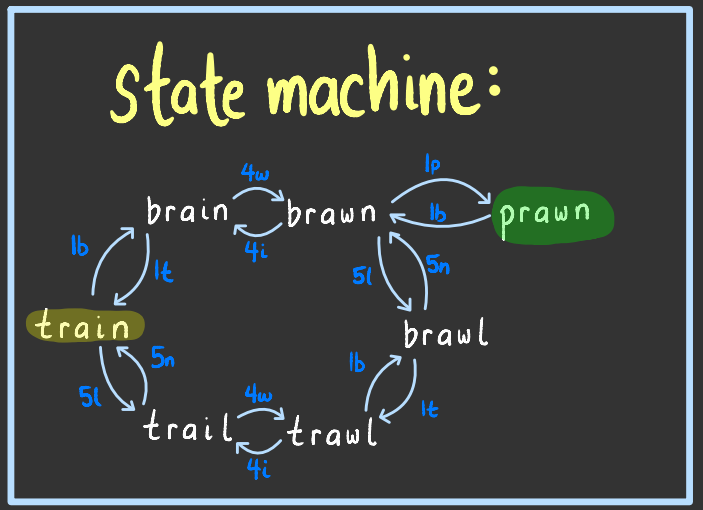

0
train
2
1
brain
2
1
trail
3

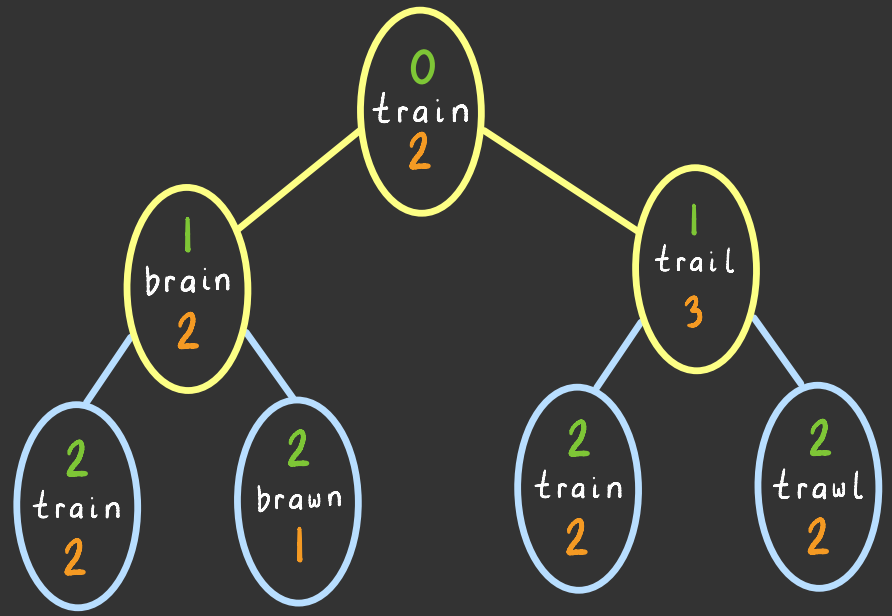

<brain, 1, 2>
<trail, 1, 3>
<train, 0, 2>

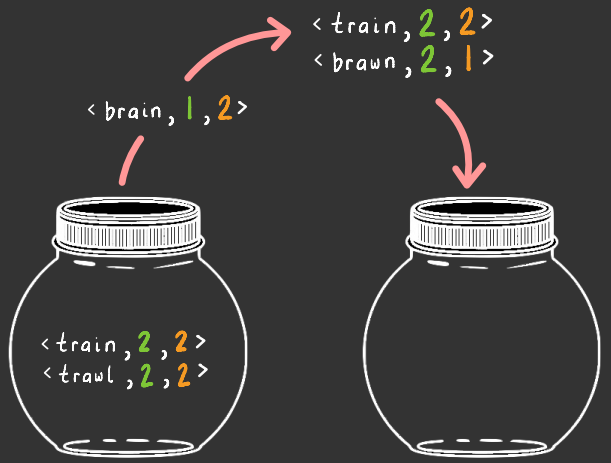

State machine:
brain
brawn
prawn
train
brawl
trail
trawl
4w
4i
1p
1b
1t
5n
5l

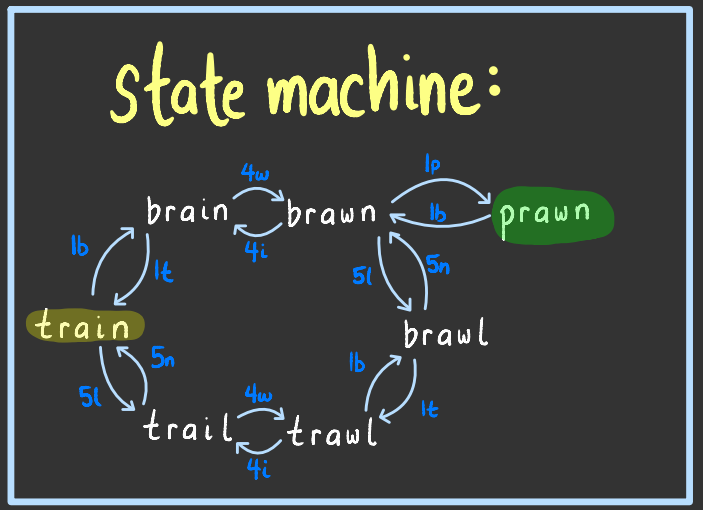

<train, 2, 2>
<trawl, 2, 2>
<trail, 1, 3>
<brain, 1, 2>

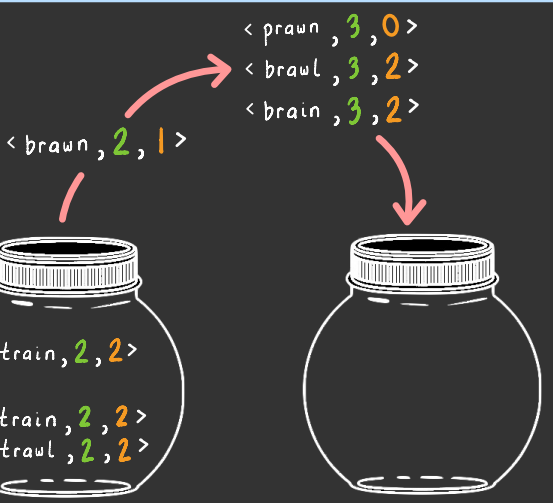

0
train
2
1
brain
2
1
trail
3
2
train
2
2
trawl
2

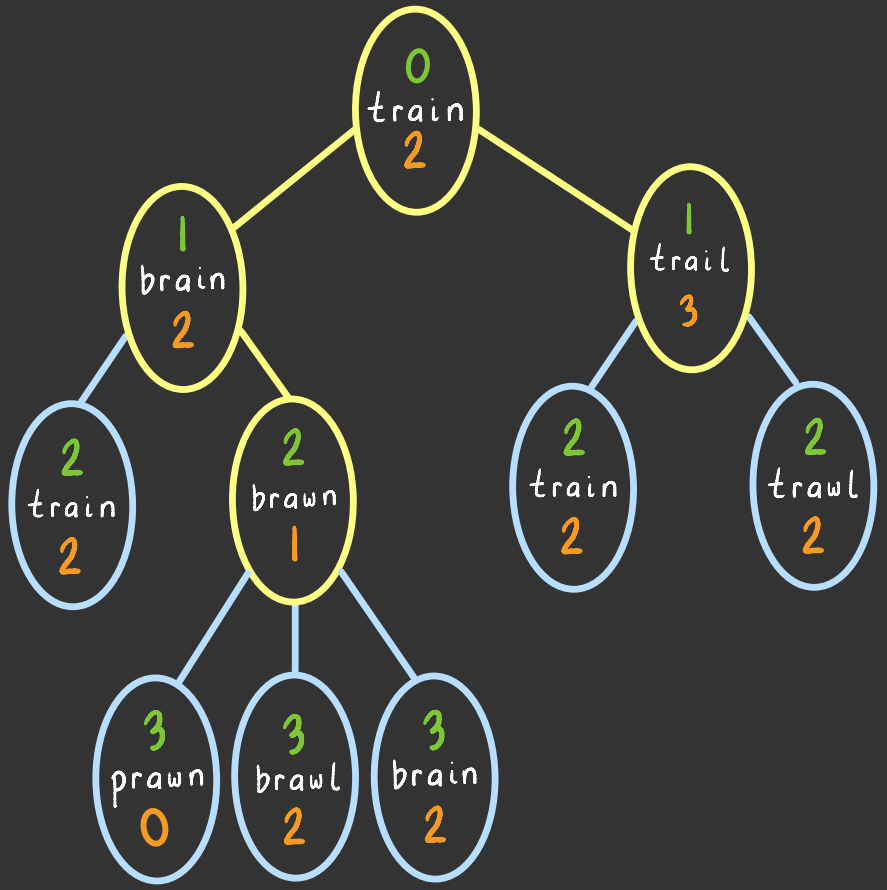

State machine:
brain
brawn
prawn
train
brawl
trail
trawl
4w
4i
1p
1b
1b
1t
5l
5n
5n
5l
1b
1t
4w
4i

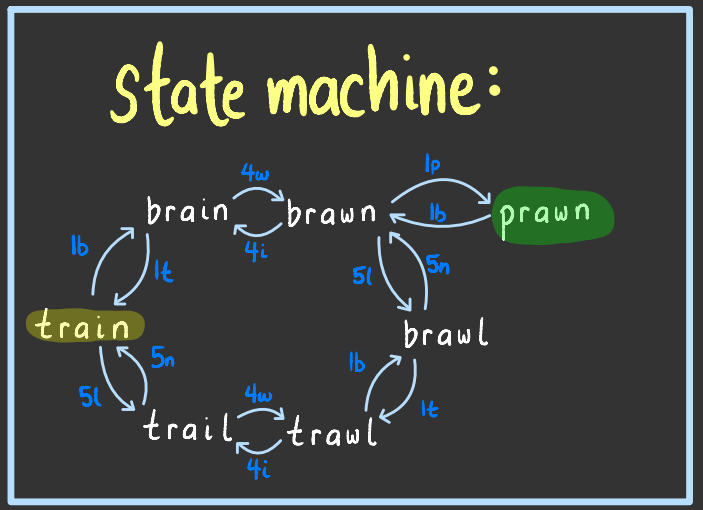

0
train
2
1
brain
2
1
trail
3
2
train
2
2
brawn
1
2
train
2
2
trawl
2

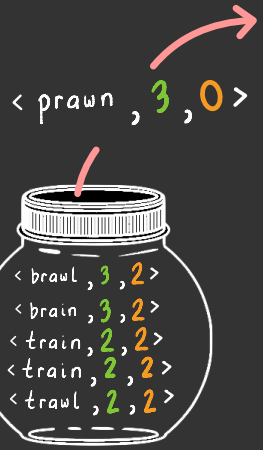

<train, 2, 2>
<brawn, 2, 1>
<brain, 1, 2>
<train, 2, 2>
<trawl, 2, 2>

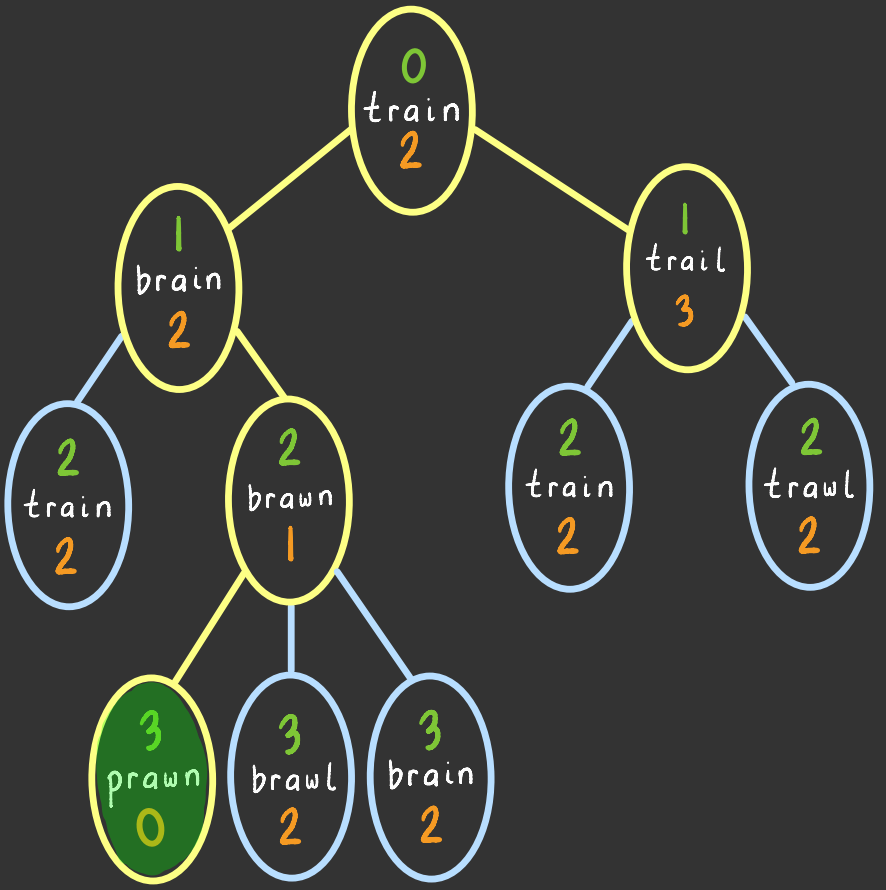

State machine:
brain
brawn
prawn
train
brawl
trail
trawl
4w
4i
1p
1b
1t
5l
5n


< prawn , 3 , 0 >
< brawl , 3 , 2 >
< brain , 3 , 2 >
< brawn , 2 , 1 >
< train , 2 , 2 >
< train , 2 , 2 >
< trawl , 2 , 2 >


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

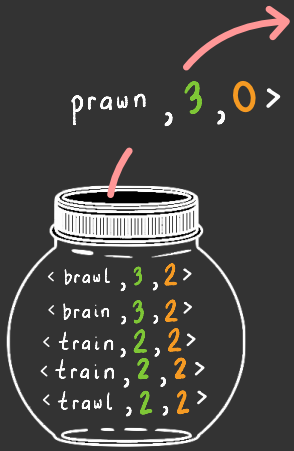

State machine:
brain
brawn
prawn
train
brawl
trail
trawl
4w
4i
1p
1b
1b
1t
5t
5n
5n
5t
1b
1t
4w
4i

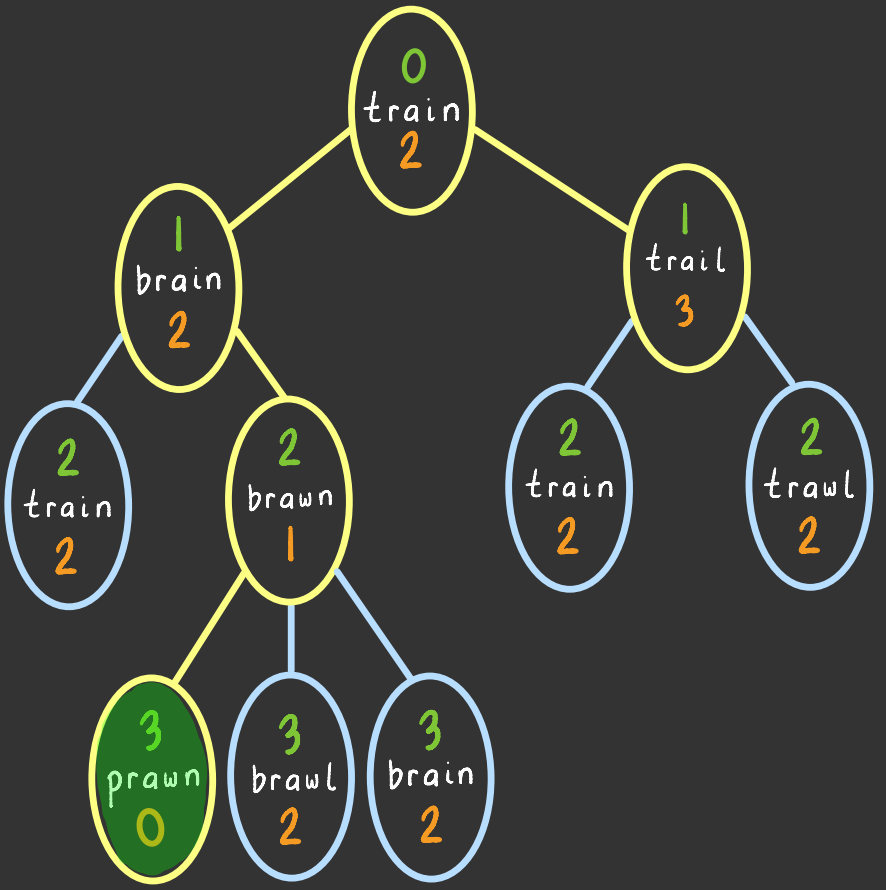

< prawn , 3 , 0 >
< brawl , 3 , 2 >
< brain , 3 , 2 >
< train , 2 , 2 >
< train , 2 , 2 >
< trawl , 2 , 2 >


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

what determines the order in which search nodes are visited?

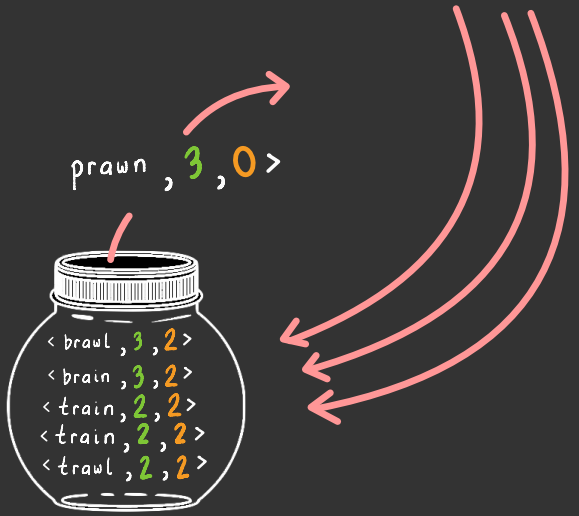
your answer here

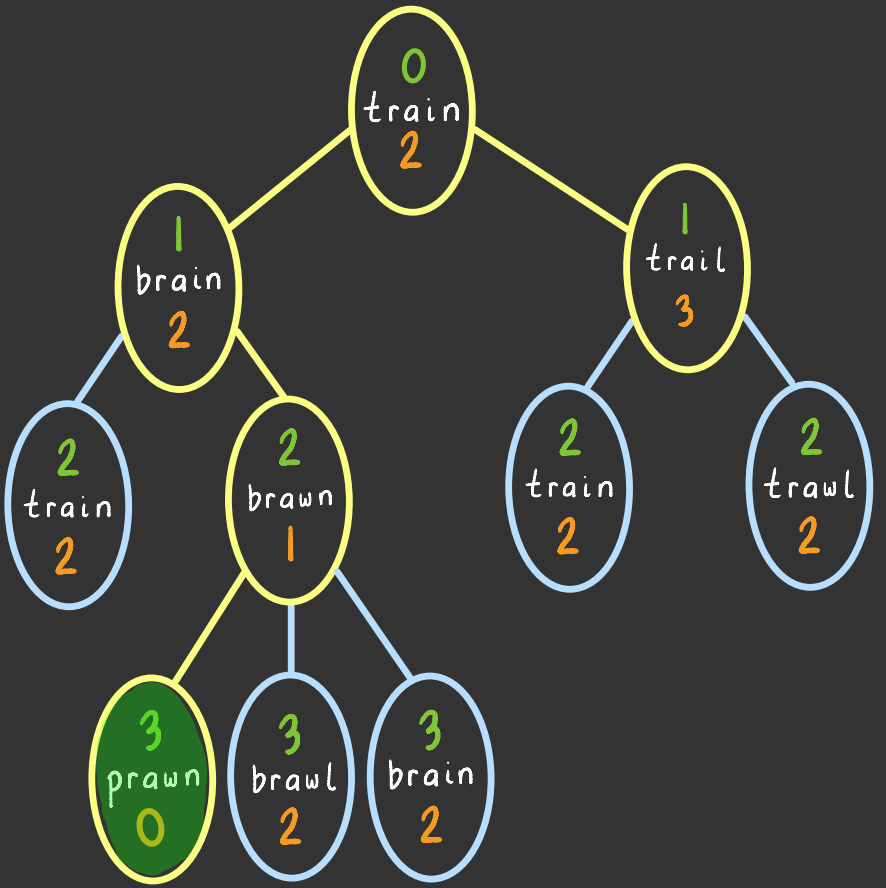




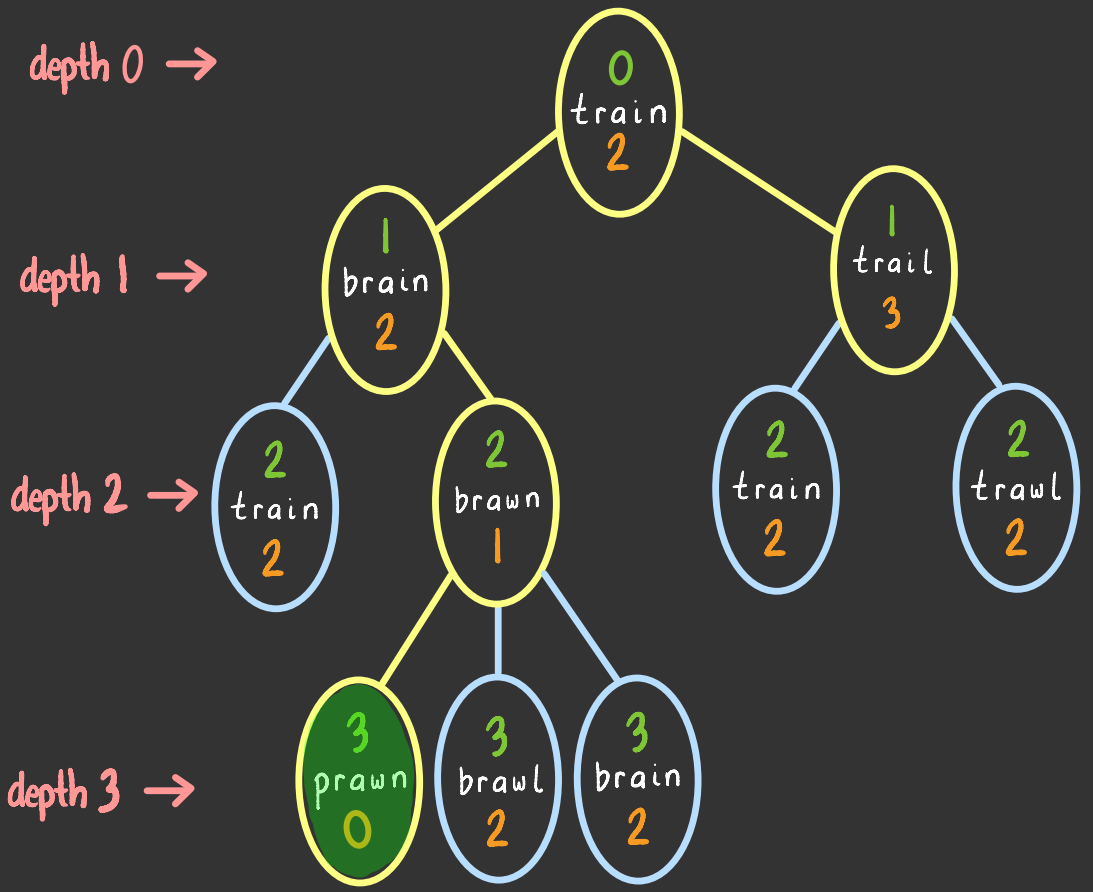
what determines the order in which search nodes are visited?

THE CONTAINER!





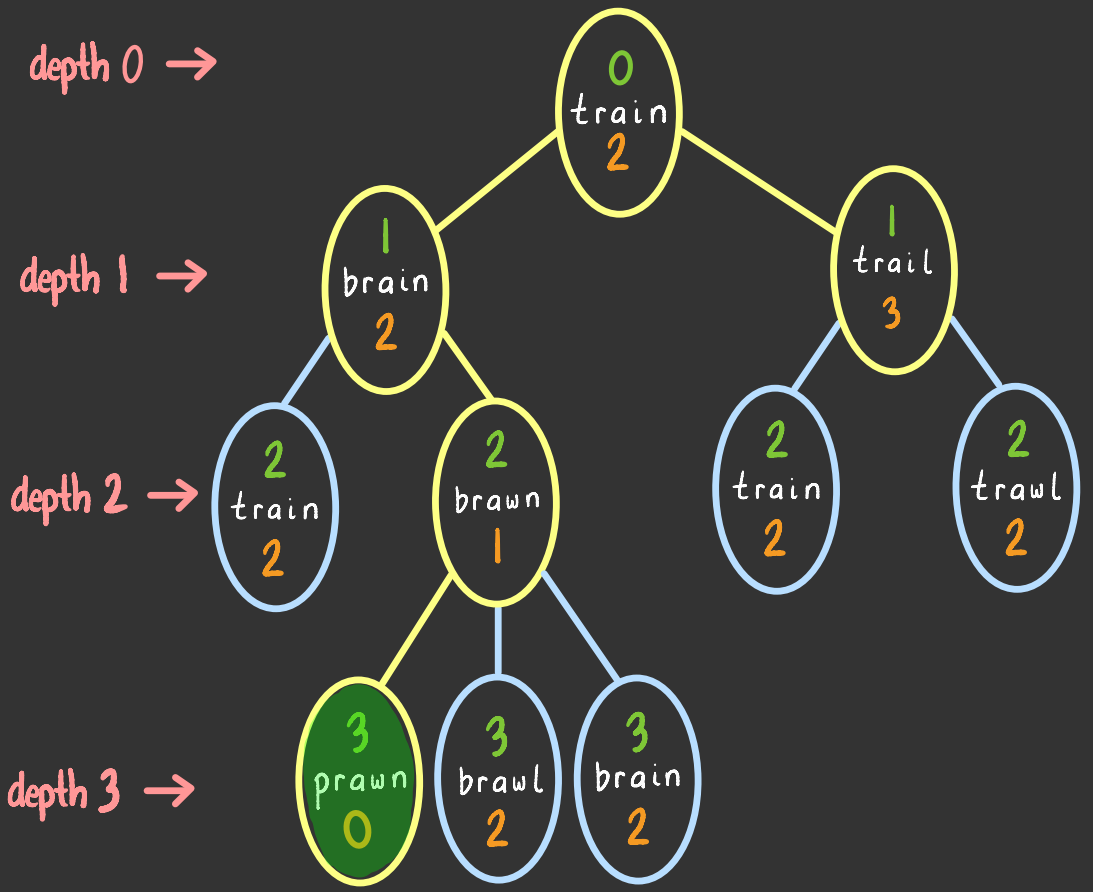
the search depth of a search node is its depth in the search tree



when is g-cost equal to search depth?

your answer here

the search depth of a search node is its depth in the search tree



when is g-cost equal to search depth? when all transitions have weight 1