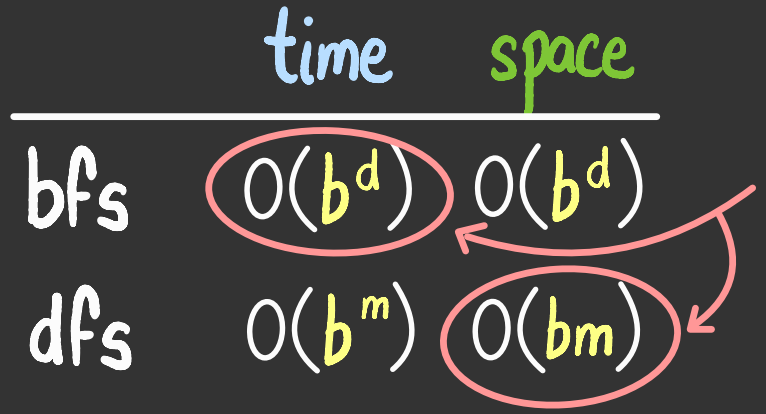
iterative deepening

CSCI 373

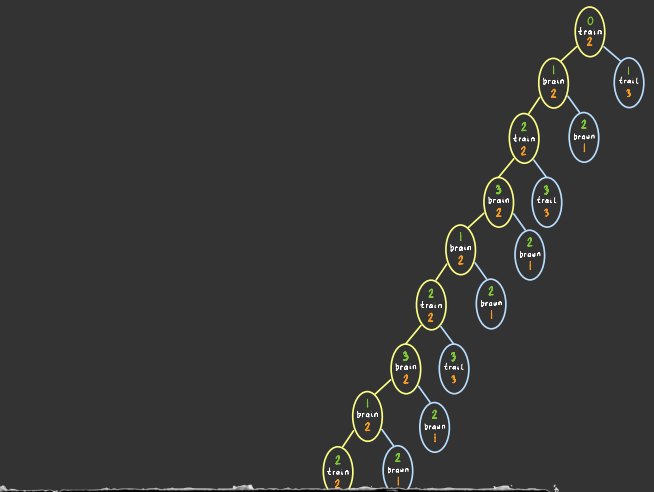


can we get the **best** of both worlds?

d — **solution depth**
the length of the shortest search path that leads to a final state

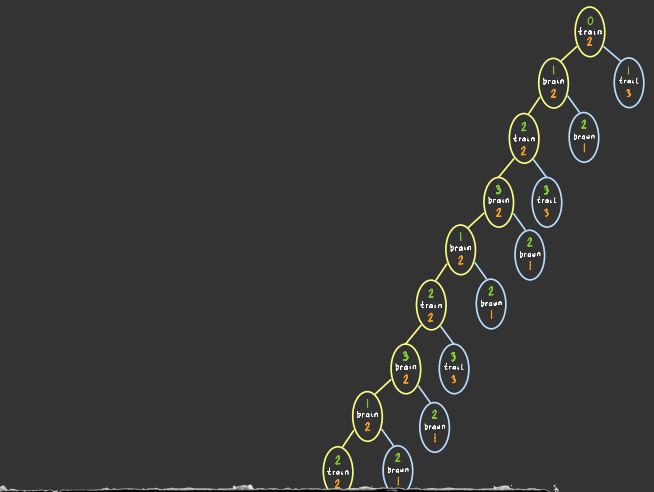
m — **maximum depth**
the length of the longest search path

b — **branching factor**
the maximum number of successors of a search node



the problem with dfs

we usually don't know the solution depth d upfront



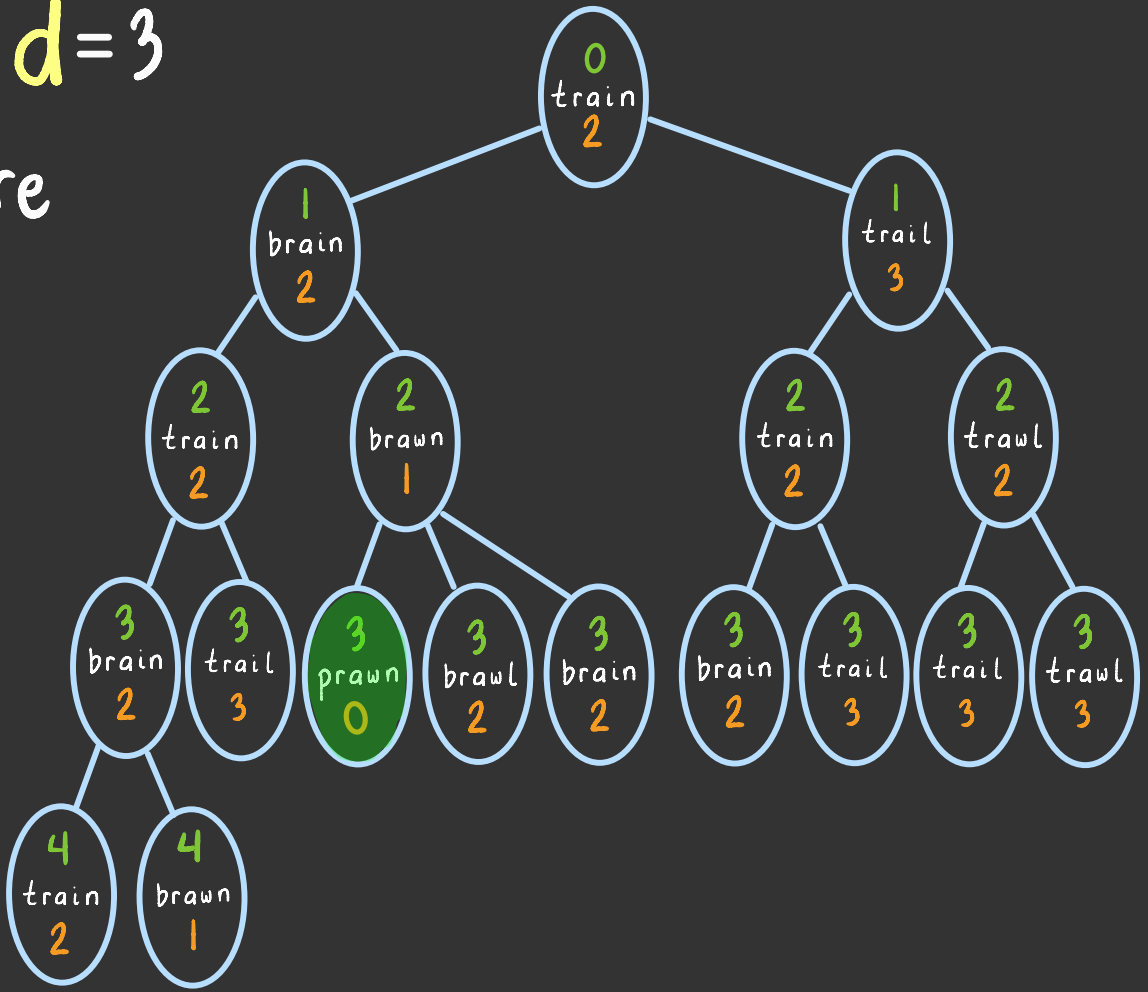
the problem with dfs

we usually don't know the solution depth d upfront

but what if we did?

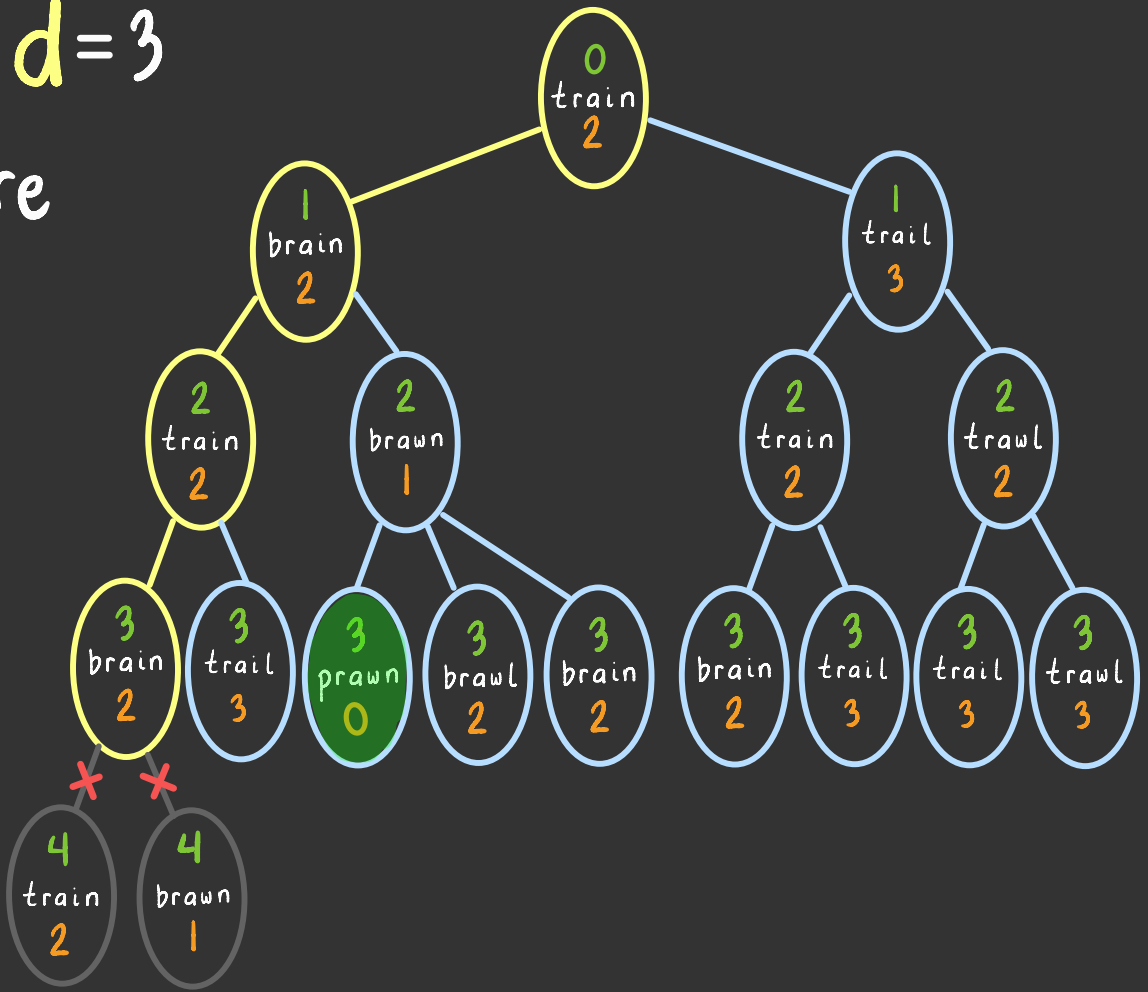
solution depth $d = 3$

So don't explore deeper than depth 3



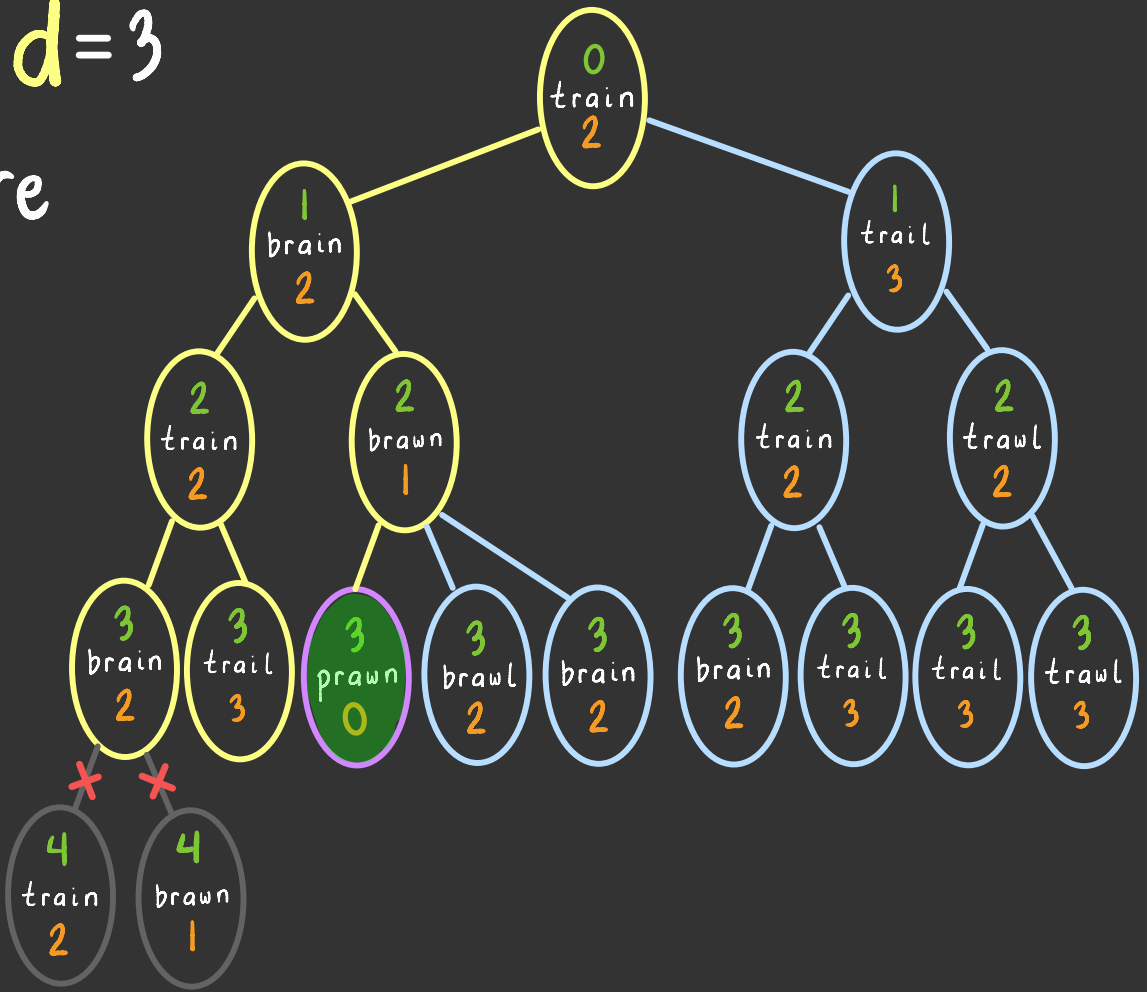
solution depth $d = 3$

So don't explore deeper than depth 3



solution depth $d = 3$

So don't explore deeper than depth 3



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

- 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')

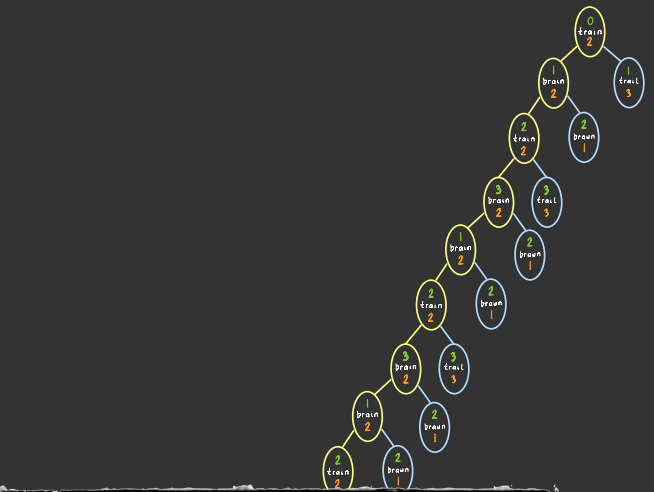
"visit" a node

how do we extend the code so we don't explore deeper than solution depth d?

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

- 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 $n' \in \text{successors}_{M,H}(n)$:
    - if $g(n) \leq d$ then container.put(n')

"visit" a node



the problem with dfs

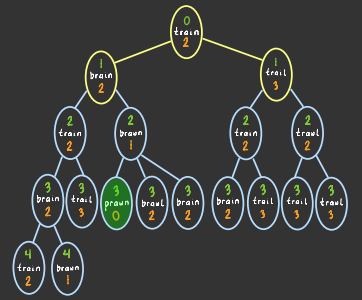
we usually don't know the solution depth d upfront

but what if we did?

ok... but what if we don't?

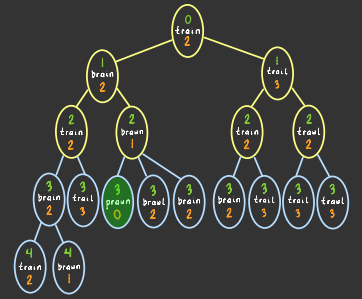
here's a dumb idea

assume solution depth $d = 1$
and run limited search



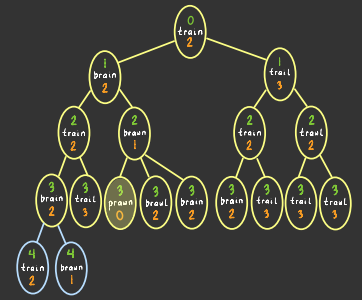
if that doesn't work...

assume solution depth $d = 2$
and run limited search



if that doesn't work...

assume solution depth $d = 3$
and run limited search



etc...

Iterative Deepening (m, H):

- $d = 0$
- repeat:
  - cost = Limited Search (m, H, d)
  - if cost $\neq \infty$ then return cost
  - $d = d + 1$

Limited Search $(m = (Q, \Sigma, \Delta, q_0, F, w), H, d)$:

- 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 $n' \in \text{successors}_{m,H}(n)$:
    - if $g(n) \leq d$ then container.put(n')

"visit" a node

ITERATIVE DEEPENING (m, H):

- $d = 0$
- repeat:
  - cost = LIMITED SEARCH (m, H, d)
  - if cost $\neq \infty$ then return cost
  - $d = d + 1$

will ITERATIVE DEEPENING find an optimal solution?

your answer here

IterativeDeepening(m, H):

- d = 0
- repeat:
  - cost = LimitedSearch(m, H, d)
  - if cost ≠ ∞ then return cost
  - d = d + 1

because of this:
if j < k, then every node at search depth j is visited before any node at search depth k

will IterativeDeepening find an optimal solution?

if all transitions have the same weight

	optimality	time	space
bfs	yes*	$O(b^d)$	$O(b^d)$
dfs	no	$O(b^m)$	$O(bm)$
ids	yes*	?	?

* if all transitions have the same weight

d — **solution depth**
the length of the shortest search path that leads to a final state

m — **maximum depth**
the length of the longest search path

b — **branching factor**
the maximum number of successors of a search node

	optimality	time	space
bfs	yes*	$O(b^d)$	$O(b^d)$
dfs	no	$O(b^m)$	$O(bm)$
ids	yes*	?	?

* if all transitions have the same weight

we run dfs on
a search tree of depth 1
then a search tree of depth 2
⋮
then a search tree of depth d

so the space is
$O(?)$

d solution depth
the length of the shortest search path that leads to a final state

m maximum depth
the length of the longest search path

b branching factor
the maximum number of successors of a search node

	optimality	time	space
bfs	yes*	$O(b^d)$	$O(b^d)$
dfs	no	$O(b^m)$	$O(bm)$
ids	yes*	?	$O(bd)$

* if all transitions have the same weight

we run dfs on
a search tree of depth 1
then a search tree of depth 2
⋮
then a search tree of depth d

so the space is
$O(bd)$

d — **solution depth**
the length of the shortest search path that leads to a final state

m — **maximum depth**
the length of the longest search path

b — **branching factor**
the maximum number of successors of a search node

	optimality	time	space
bfs	yes*	$O(b^d)$	$O(b^d)$
dfs	no	$O(b^m)$	$O(bm)$
ids	yes*	?	$O(bd)$

but time is where the magic happens

* if all transitions have the same weight

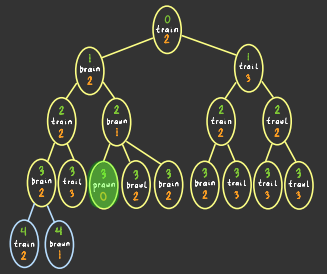
d — **solution depth**
the length of the shortest search path that leads to a final state

m — **maximum depth**
the length of the longest search path

b — **branching factor**
the maximum number of successors of a search node

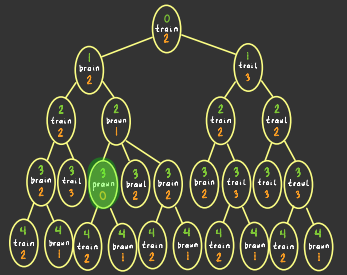
what is faster?

one bfs to solution depth d



$O(b^d)$

one dfs to maximum depth m

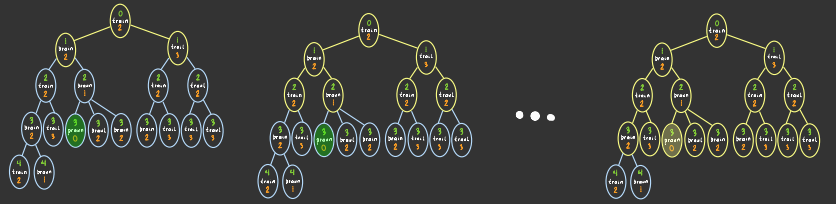


$O(b^m)$

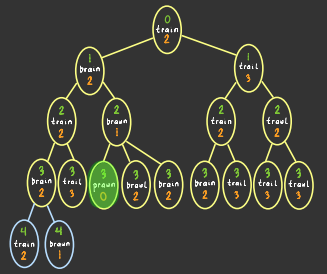
d dfses

... one to depth 1
... one to depth 2

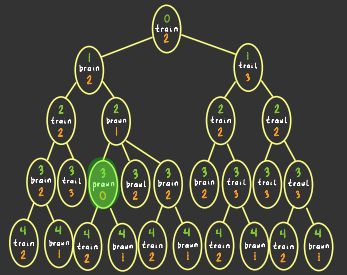
... one to depth d



$O(b^d)$ one bfs to solution depth d



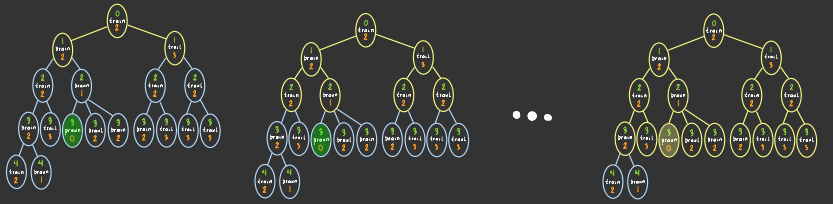
$O(b^m)$ one dfs to maximum depth m



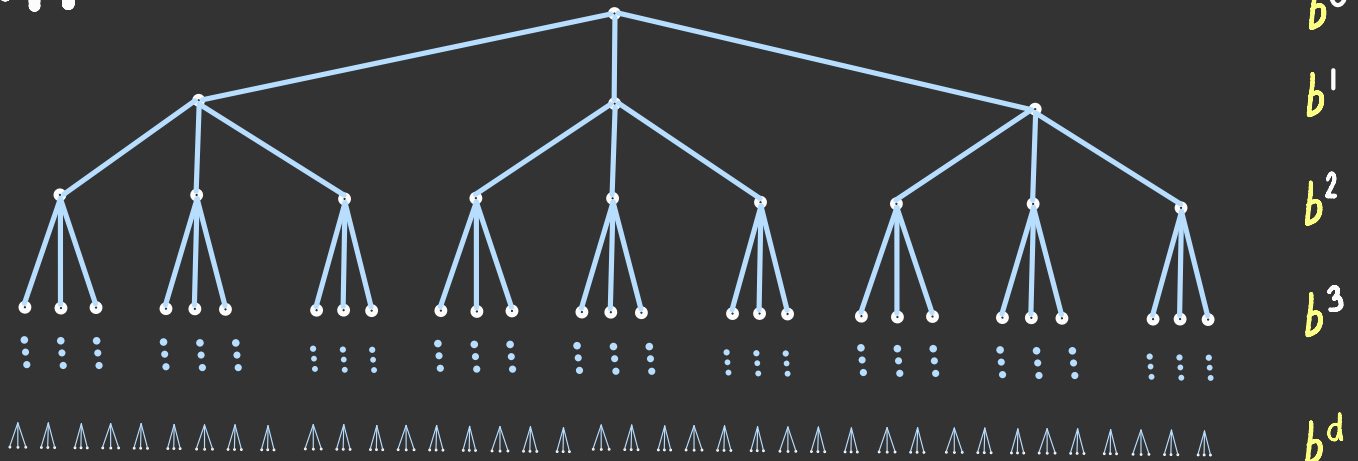
d dfses

... one to depth 1
... one to depth 2

... one to depth d



recall:



in a full tree with branching factor b
how many nodes are there in total?

$$\sum_{i=0}^{d} b^i < 2b^d$$

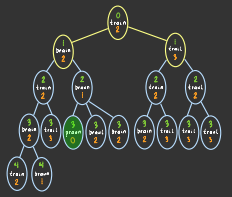
if $b \geq 2$

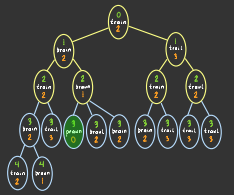
d dfses

... one to depth 1

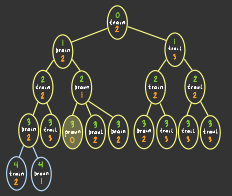
... one to depth 2

... one to depth d





...



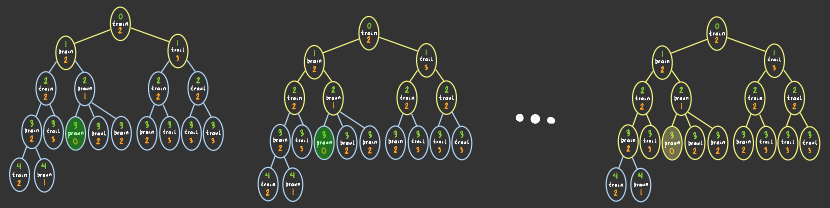
$< 2b^1$ $< 2b^2$ $< 2b^d$

d dfses

... one to depth 1

... one to depth 2

... one to depth d



$< 2b^1 \quad < 2b^2 \quad < 2b^d$

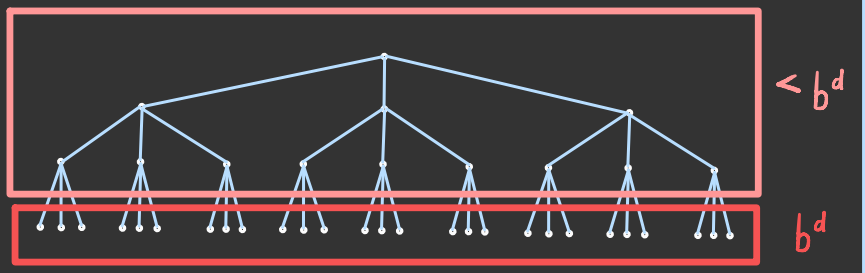
so we visit at most $2b^1 + 2b^2 + \cdots + 2b^d$ nodes

$= 2\left(b^1 + b^2 + \cdots + b^{d-1} + b^d\right)$ nodes

and now recall this:

fun fact: if $b \geq 2$, then

$$\sum_{i=0}^{d} b^i < 2b^d$$

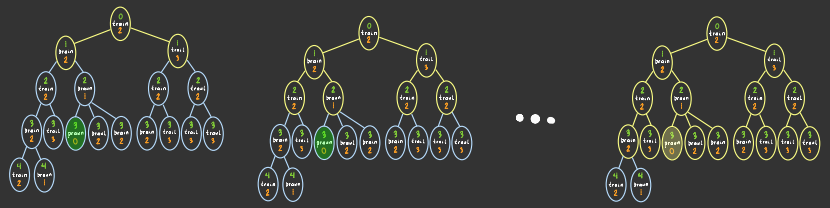


trees have more leaves than internal nodes

d dfses

... one to depth 1
... one to depth 2

... one to depth d



$< 2b^1 \quad < 2b^2 \quad < 2b^d$

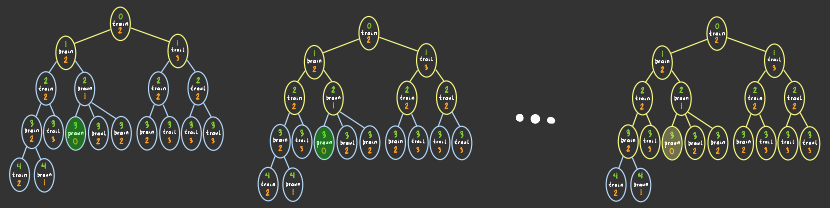
so we visit at most $2b^1 + 2b^2 + \cdots + 2b^d$ nodes

$= 2\left(\underbrace{b^1 + b^2 + \cdots + b^{d-1}}_{< b^d} + b^d\right)$ nodes

d dfses

... one to depth 1
... one to depth 2

... one to depth d



$< 2b^1 \quad < 2b^2 \quad < 2b^d$

so we visit at most $2b^1 + 2b^2 + \cdots + 2b^d$ nodes

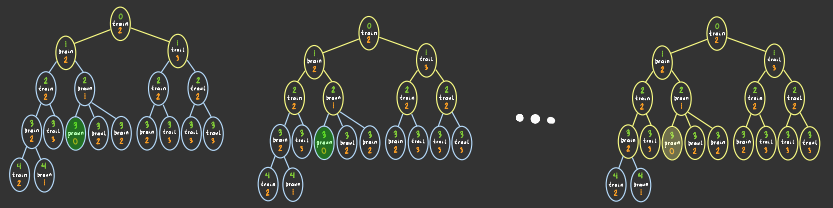
$= 2\left(b^1 + b^2 + \cdots + b^{d-1} + b^d\right)$ nodes

$< 2\left(b^d + b^d\right)$ nodes

d dfses

... one to depth 1
... one to depth 2

... one to depth d



$< 2b^1 \quad < 2b^2 \quad\quad < 2b^d$

so we visit at most $2b^1 + 2b^2 + \cdots + 2b^d$ nodes

$= 2\left(b^1 + b^2 + \cdots + b^{d-1} + b^d\right)$ nodes

$< 2\left(\quad b^d \quad\quad + b^d\right)$ nodes

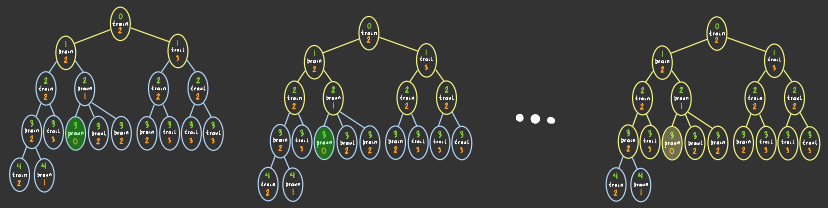
that's $4b^d$ nodes!

d dfses

... one to depth 1

... one to depth 2

... one to depth d



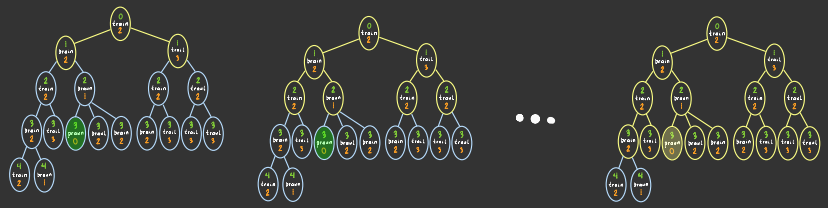
So we visit at most $4b^d$ nodes

d dfses

... one to depth 1

... one to depth 2

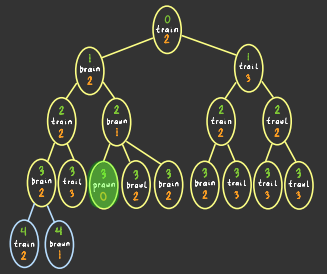
... one to depth d



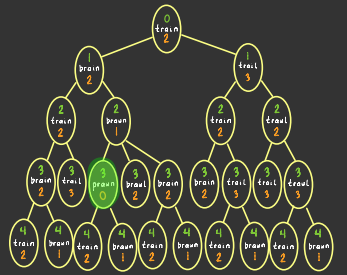
So we visit at most $4b^d$ nodes

$$O(b^d)$$

$O(b^d)$ one bfs to solution depth d



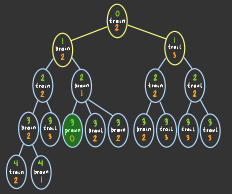
$O(b^m)$ one dfs to maximum depth m

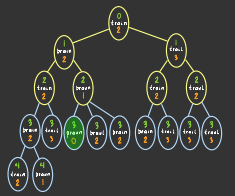


$O(b^d)$ d dfses

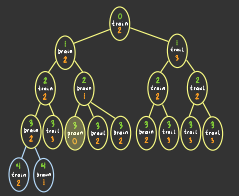
... one to depth 1
... one to depth 2

... one to depth d





...



	optimality	time	space
bfs	yes*	$O(b^d)$	$O(b^d)$
dfs	no	$O(b^m)$	$O(bm)$
ids	yes*	$O(b^d)$	$O(bd)$

so asymptotically, we can get the best of both worlds

* if all transitions have the same weight

d — **solution depth**
the length of the shortest search path that leads to a final state

m — **maximum depth**
the length of the longest search path

b — **branching factor**
the maximum number of successors of a search node

	optimality	time	space
bfs	yes*	$O(b^d)$	$O(b^d)$
dfs	no	$O(b^m)$	$O(bm)$
ids	yes*	$O(b^d)$	$O(bd)$

* if all transitions have the same weight

why might we still prefer bfs or dfs for certain state machines?

your answer here

d — **solution depth**
the length of the shortest search path that leads to a final state

m — **maximum depth**
the length of the longest search path

b — **branching factor**
the maximum number of successors of a search node

	optimality	time	space
bfs	yes*	$O(b^d)$	$O(b^d)$
dfs	no	$O(b^m)$	$O(bm)$
ids	yes*	$O(b^d)$	$O(bd)$

* if all transitions have the same weight

why might we still prefer bfs or dfs for certain state **machines**?

constant factors

d — solution depth
the length of the shortest search path that leads to a final state

m — maximum depth
the length of the longest search path

b — branching factor
the maximum number of successors of a search node