

iterative
deepening

19 sept
2022

CSCI
373

Can we get the

best

of both worlds?

	time	space
bfs	$O(b^d)$	$O(b^d)$
dfs	$O(b^m)$	$O(bm)$

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

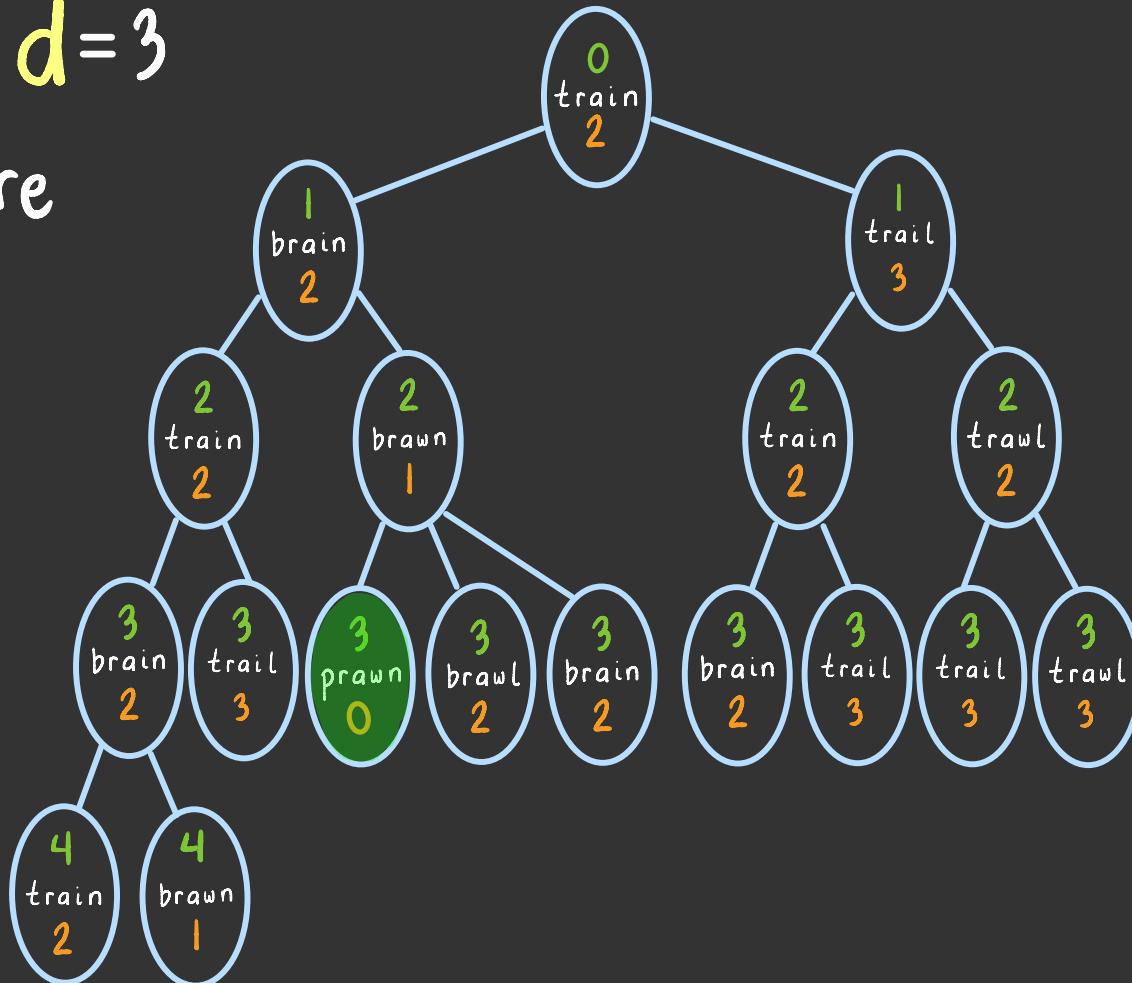
dfs

We usually don't know the solution depth d upfront

but what if we did?

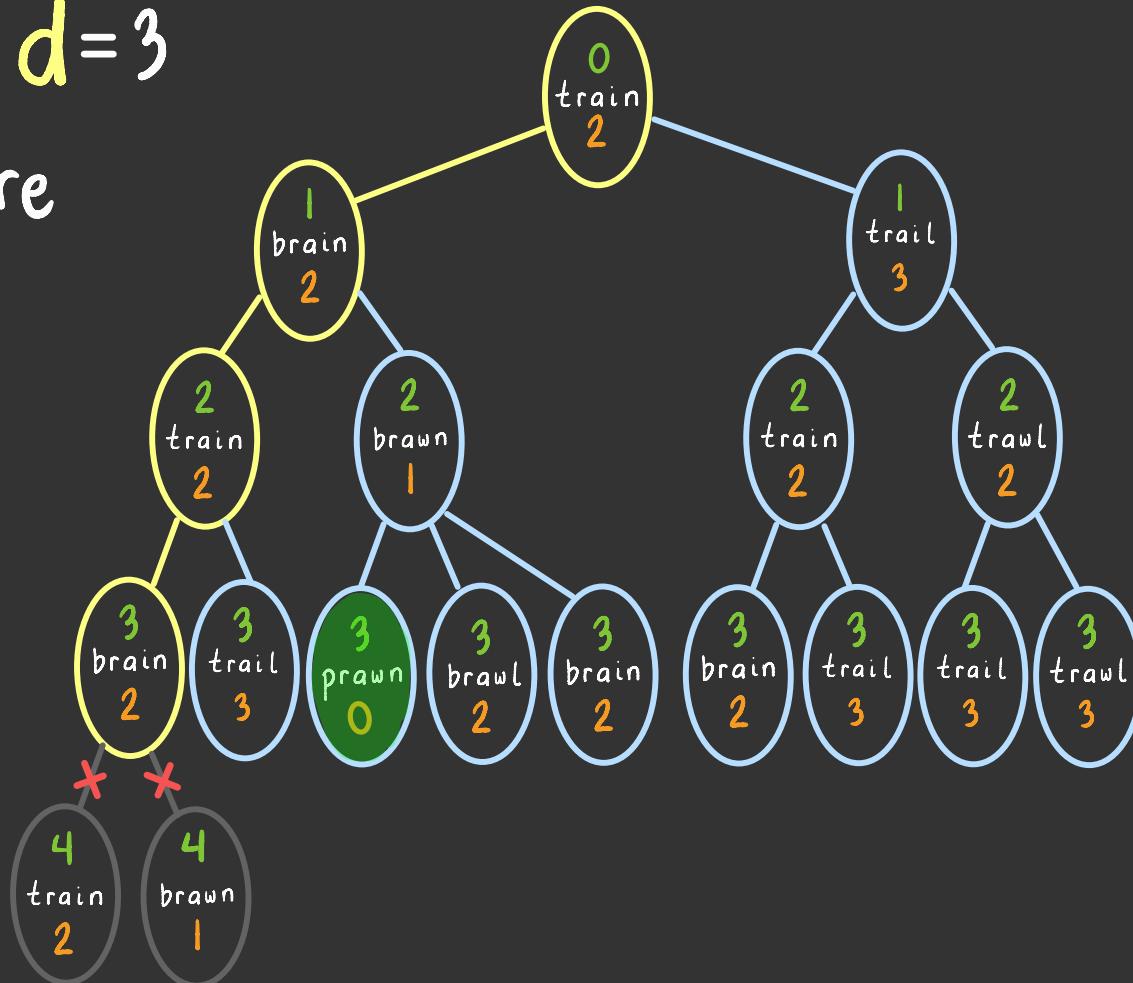
solution depth $d=3$

So don't explore
deeper than
depth 3



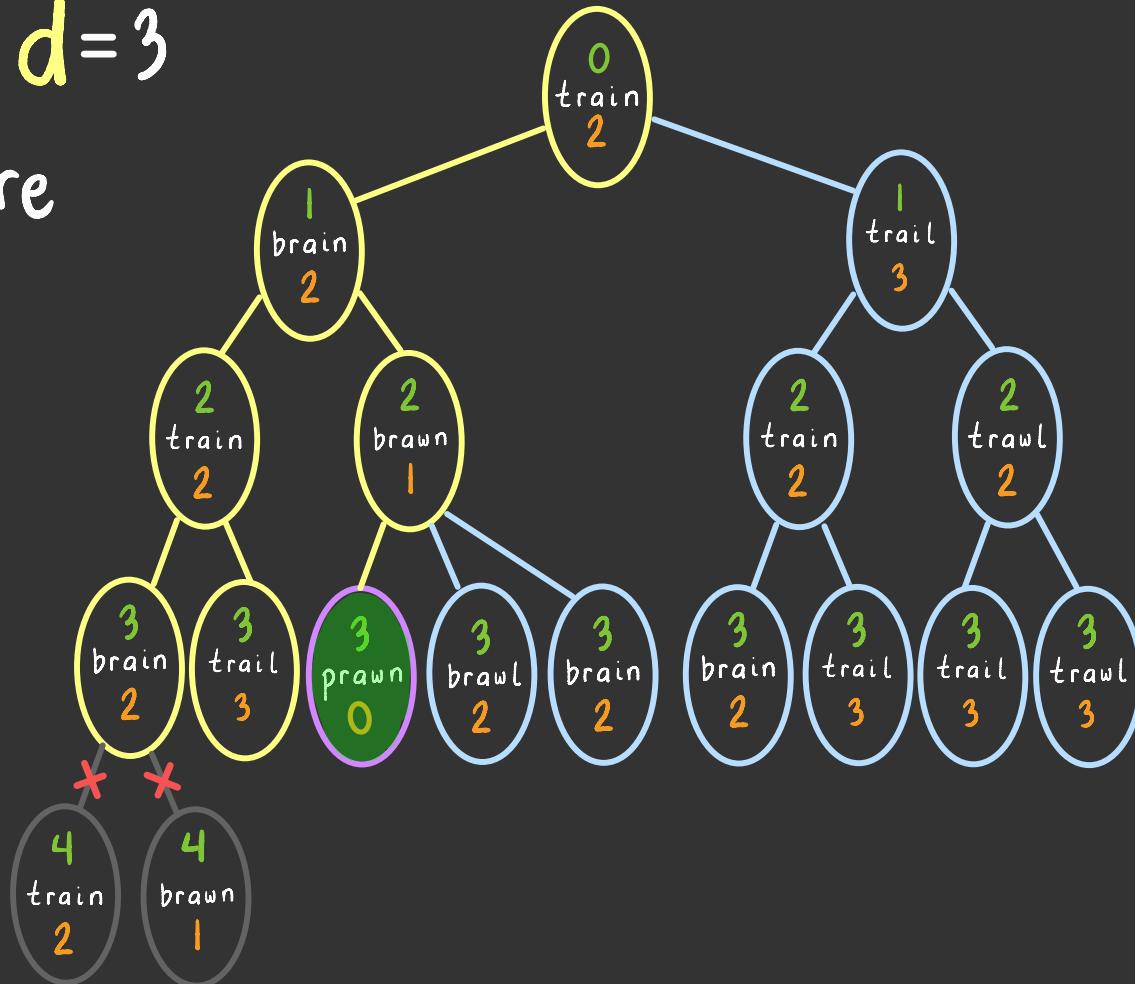
solution depth $d=3$

so don't explore
deeper than
depth 3



solution depth $d=3$

So don't explore
deeper than
depth 3



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

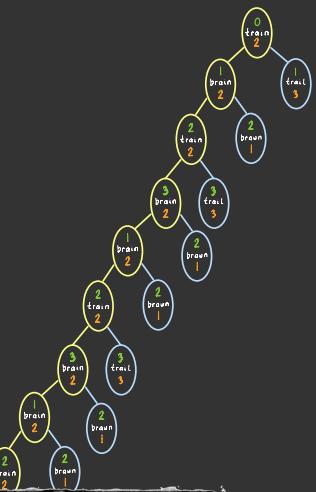
- ▶ container = new Container()
 - ▶ container.put($\langle q_0, 0, H(q_0) \rangle$)
 - ▶ repeat:
 - ▶ if container.empty() then return ∞
 - ▶ n = container.get()
 - ▶ if $q(n) \in F$ then return $g(n)$
 - ▶ let successors _{M, H} (n) = $\{ \langle q', g(n) + w(q(n), \sigma, q'), H(q') \rangle \mid (q(n),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

"visit" a node

$\left\{ \begin{array}{l} \text{▶ } n = \text{container.get()} \\ \text{▶ if } q(n) \in F \text{ then return } g(n) \\ \text{▶ let } \text{successors}_{M,H}(n) = \left\{ \langle q', g(n) + w(q(n), \sigma, q'), H(q') \rangle \mid (q(n), \sigma, q') \in \Delta \right\} \\ \text{▶ for } n' \in \text{successors}_{M,H}(n): \\ \quad \text{if } g(n') \leq d \text{ then container.put}(n') \end{array} \right.$



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

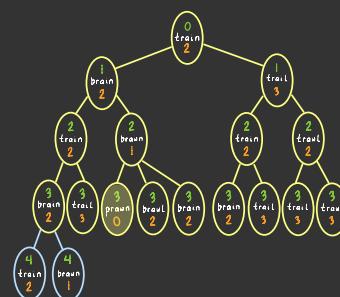
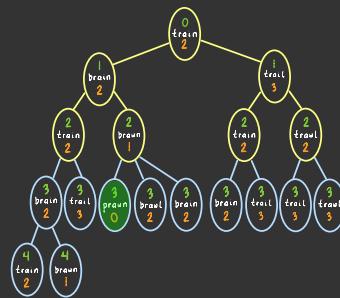
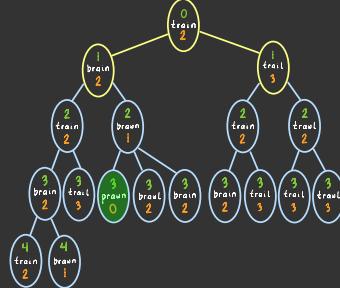
assume solution depth $d = 1$
and run limited search

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 ∞
 - ▶ $n = \text{container.get}()$
 - ▶ "visit" a node
 - ▶ if $g(n) \in F$ then return $g(n)$
 - ▶ let 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')

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

ITERATIVE DEEPENING (m, h):

- ▶ $d = 0$
- ▶ repeat:
 - ▶ $\text{cost} = \text{LIMITED SEARCH}(m, h, d)$
 - ▶ if $\text{cost} \neq \infty$ 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 ITERATIVE DEEPENING 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
 $O(\boxed{?})$

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

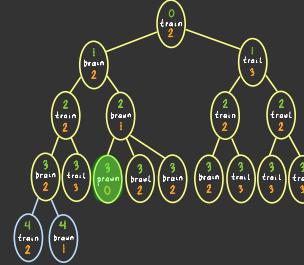
the length of the longest search path

b

branching factor

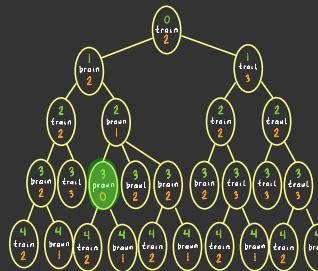
the maximum number of successors of a search node

one bfs to
solution depth d



$O(b^d)$

what is
faster? one dfs to
maximum depth m



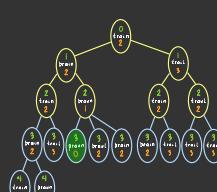
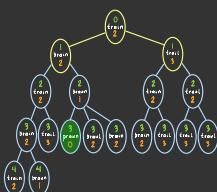
$O(b^m)$

d dfses

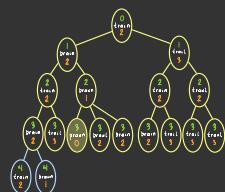
... one to depth 1

... one to depth 2

... one to depth d



...



$O(b^d)$

one bfs to
solution depth d

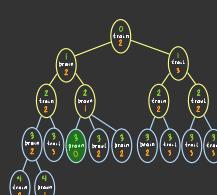
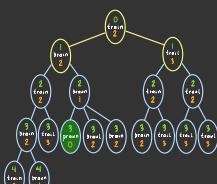
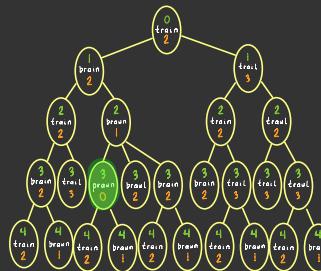
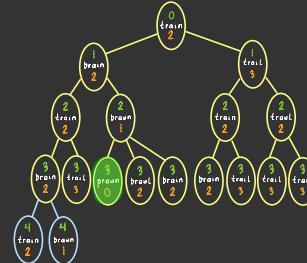
$O(b^m)$

one dfs to
maximum depth m

d *dfses*

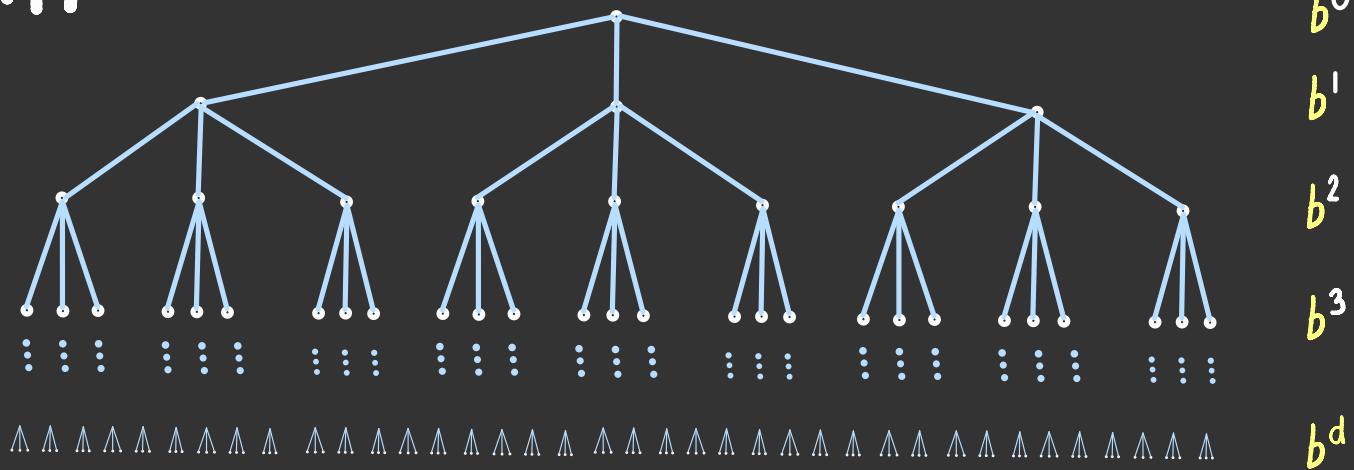
... one to depth 1
... one to depth 2

... one to depth d



2

recall:



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

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

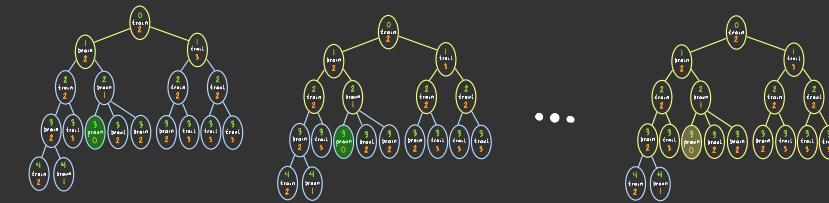
if $b \geq 2$

d dfses

... one to depth 1

... one to depth 2

... one to depth d



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

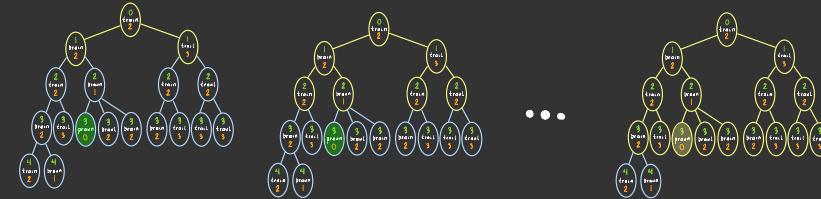
$$< 2b^d$$

d dfses

... one to depth 1

... one to depth 2

... one to depth d



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

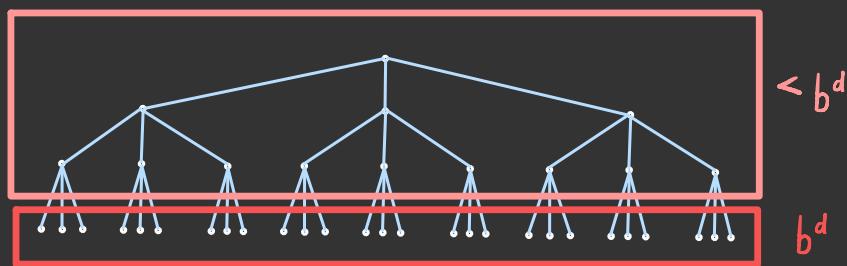
so we visit at most $2b^1 + 2b^2 + \dots + 2b^d$ nodes

$$= 2(b^1 + b^2 + \dots + b^{d-1} + b^d) \text{ nodes}$$

and now recall|| this:

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

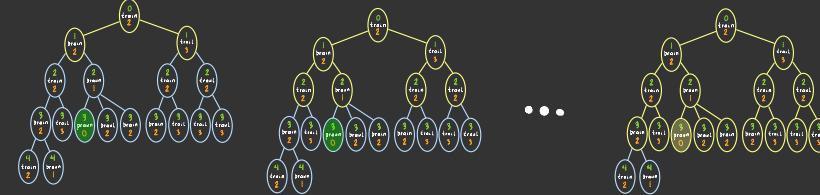
$$\sum_{i=0}^d b^i < 2b^d$$



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 $2b^1 + 2b^2 + \dots + 2b^d$ nodes

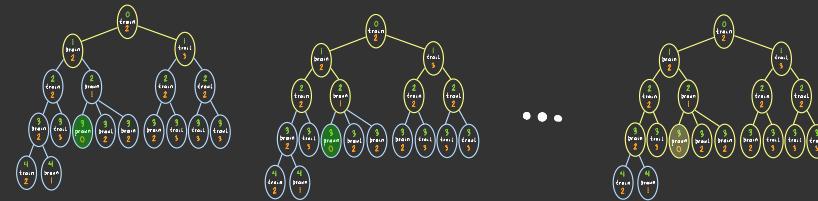
$$= 2 \underbrace{(b^1 + b^2 + \dots + b^{d-1} + b^d)}_{< b^d} \text{ nodes}$$

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 $2b^1 + 2b^2 + \dots + 2b^d$ nodes

$$= 2(b^1 + b^2 + \dots + b^{d-1} + b^d) \text{ nodes}$$

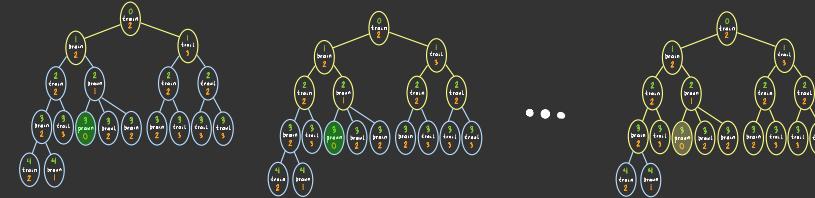
$$< 2(b^d + b^d) \text{ nodes}$$

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 $2b^1 + 2b^2 + \dots + 2b^d$ nodes

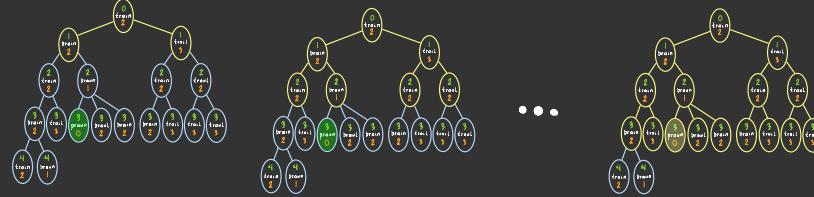
$$= 2(b^1 + b^2 + \dots + b^{d-1} + b^d) \text{ nodes}$$

$$< 2(b^d + b^d) \text{ nodes}$$

that's $4b^d$ nodes!

d dfses

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



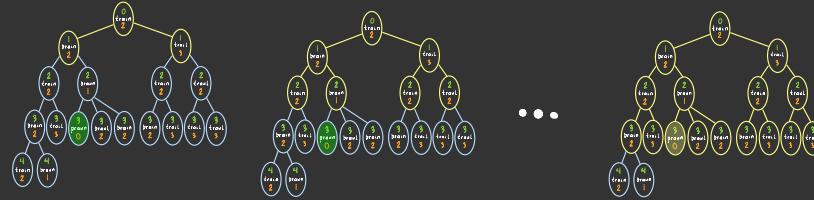
... one to depth d

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

d dfses

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

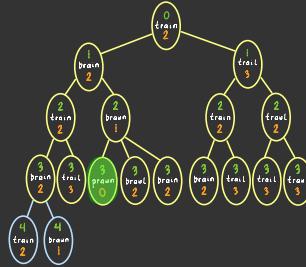
... one to depth *d*



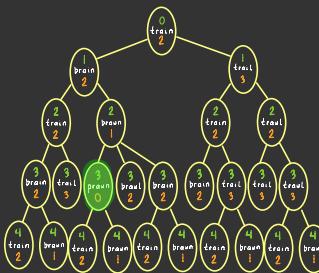
So we visit at most $4b^d$ nodes

$$O(b^d)$$

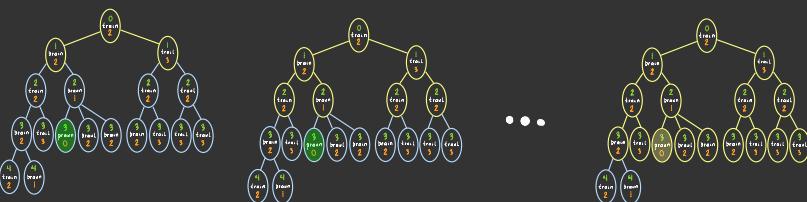
$O(b^d)$ one bfs to solution depth d



$O(b^m)$ 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*	$O(b^d)$	$O(bd)$

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

* 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
--	--	---