metrics for search analysis

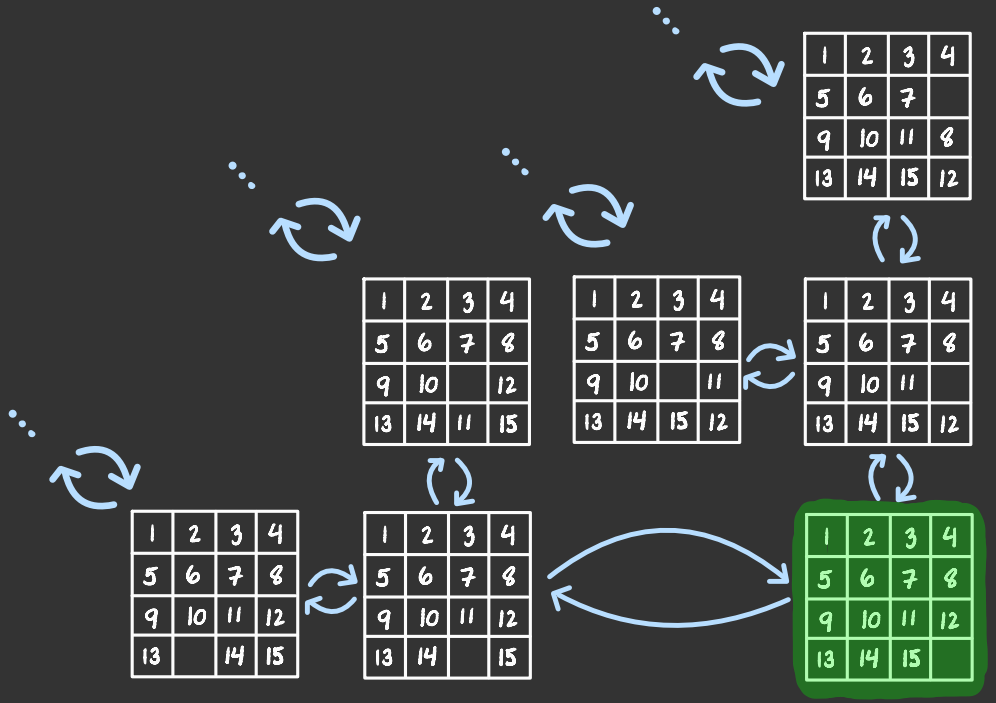
19 sept 2022

CSCI 373

from the wikipedia article on depth-first search

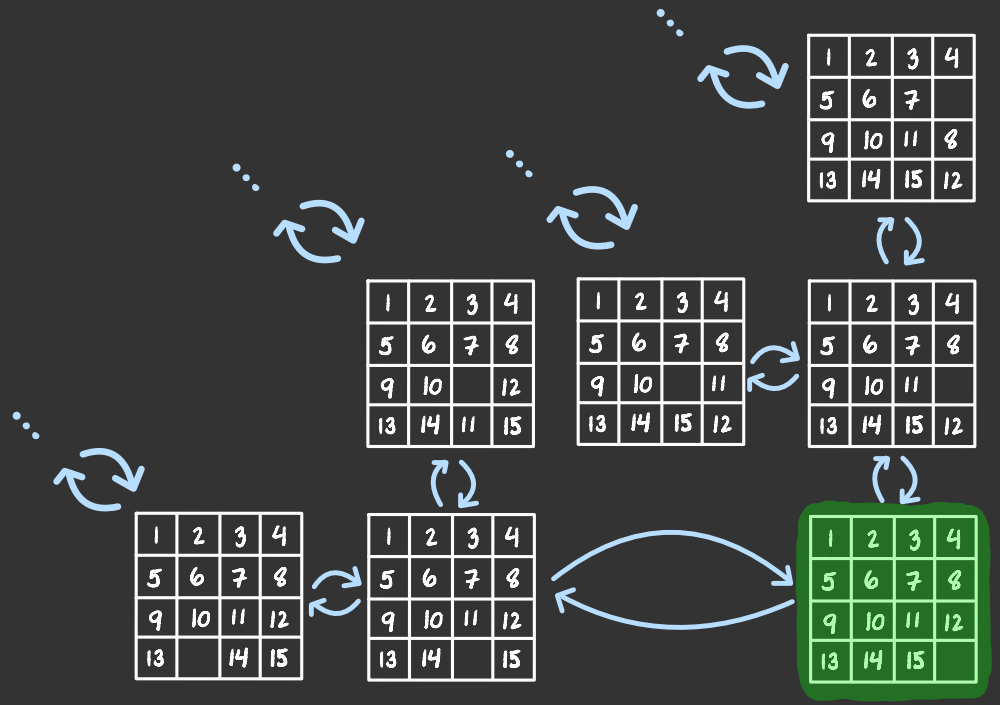
Properties [edit]

The time and space analysis of DFS differs according to its application area. In theoretical computer science, DFS is typically used to traverse an entire graph, and takes time $O(|V| + |E|)$,[4] linear in the size of the graph. In these applications it also uses space $O(|V|)$ in the worst case to store the stack of vertices on the current search path as well as the set of already-visited vertices. Thus, in this setting, the time and space bounds are the same as for breadth-first search and the choice of which of these two algorithms to use depends less on their complexity and more on the different



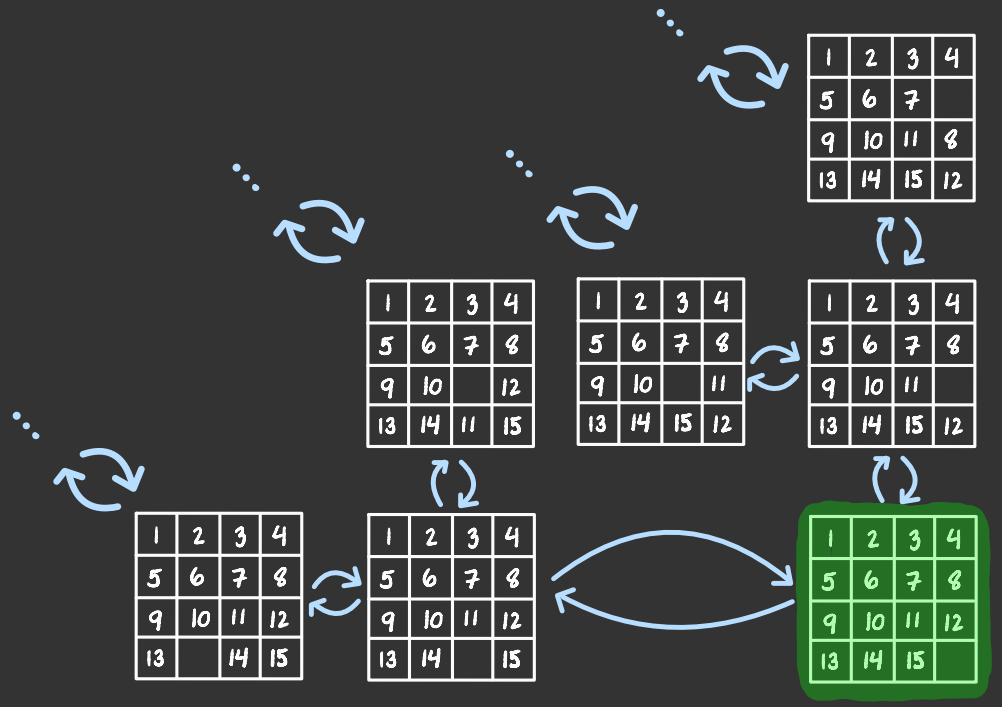
how many states are there for this sliding tile puzzle?

your answer here



how many states are there for this sliding tile puzzle?

16!



how many states are there for this sliding tile puzzle?

16!

= 20,922,789,888,000

(over twenty trillion)

Properties [edit]

The time and space analysis of DFS differs according to its application area. In theoretical computer science, DFS is typically used to traverse an entire graph, and takes time $O(|V| + |E|)$,[4] linear in the size of the graph. In these applications it also uses space $O(|V|)$ in the worst case to store the stack of vertices on the current search path as well as the set of already-visited vertices. Thus, in this setting, the time and space bounds are the same as for breadth-first search and the choice of which of these two algorithms to use depends less on their complexity and more on the different

so dfs is linear... in twenty trillion vertices

how many states are there for this sliding tile puzzle?

16!

= 20,922,789,888,000

(over twenty trillion)

and what about

this

sliding tile puzzle?

how many states?

and what about

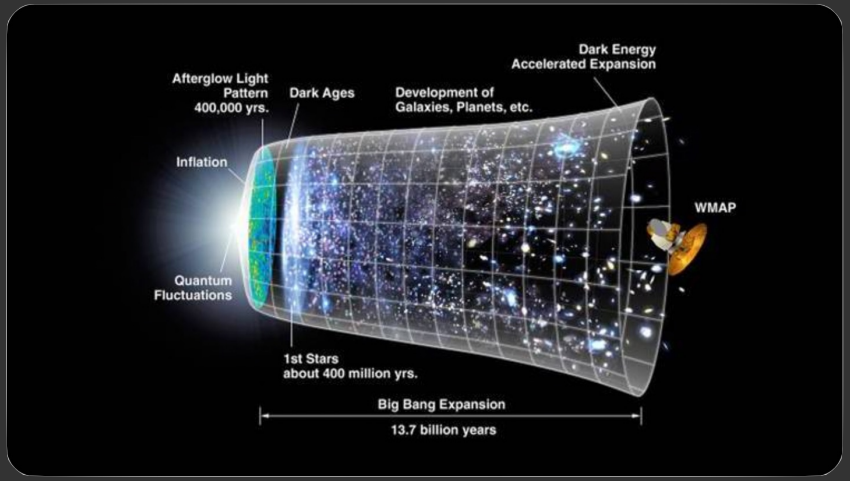
this

sliding tile puzzle?

$36!$

$\approx 3.72 \times 10^{41}$

(over three hundred duodecillion)



the 64-puzzle has more states than **atoms** in the observable universe

from the wikipedia article on depth-first search

Properties [edit]

The time and space analysis of DFS differs according to its application area. In theoretical computer science, DFS is typically used to traverse an entire graph, and takes time $O(|V| + |E|)$,[4] linear in the size of the graph. In these applications it also uses space $O(|V|)$ in the worst case to store the stack of vertices on the current search path as well as the set of already-visited vertices. Thus, in this setting, the time and space bounds are the same as for breadth-first search and the choice of which of these two algorithms to use depends less on their complexity and more on the different

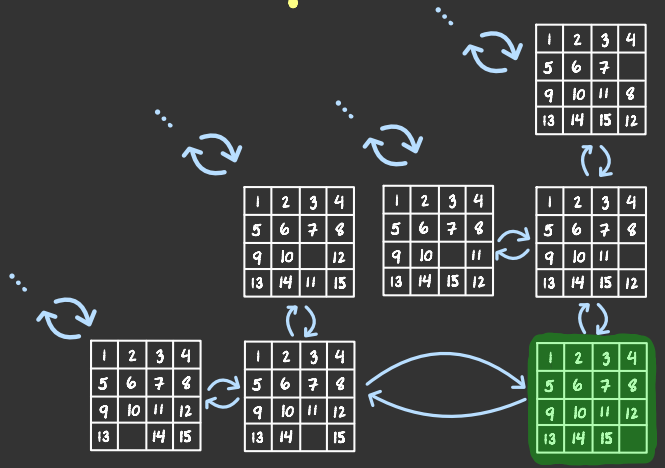
the number of states (i.e. |V|) doesn't provide much insight into the performance of search algorithms for most AI applications

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

maximum depth: the length of the longest search path

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

15-puzzle



solution depth = your answer here

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

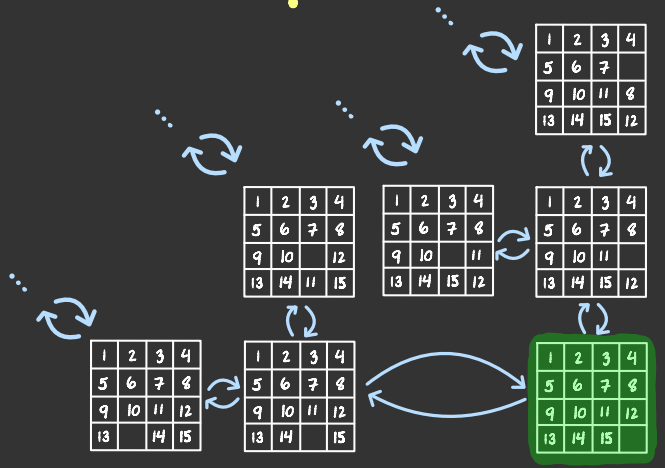
maximum depth = your answer here

the length of the longest search path

branching factor = your answer here

the maximum number of successors of a search node

15-puzzle



solution depth = depends on initial state

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

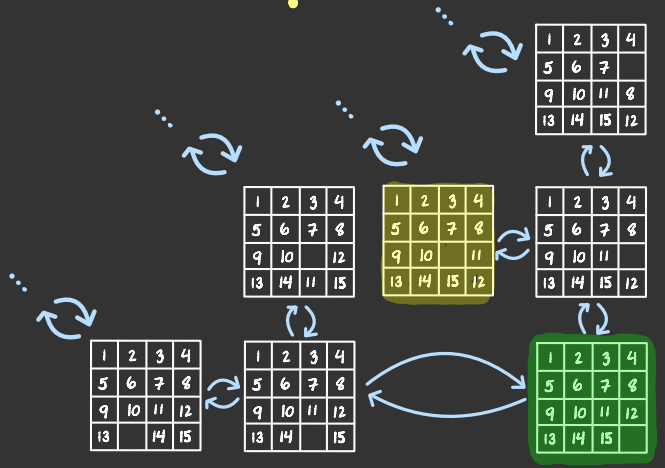
maximum depth = ∞

the length of the longest search path

branching factor = 4

the maximum number of successors of a search node

15-puzzle



solution depth = 2

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

maximum depth = ∞

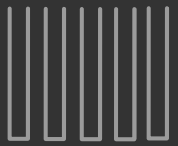
the length of the longest search path

branching factor = 4

the maximum number of successors of a search node

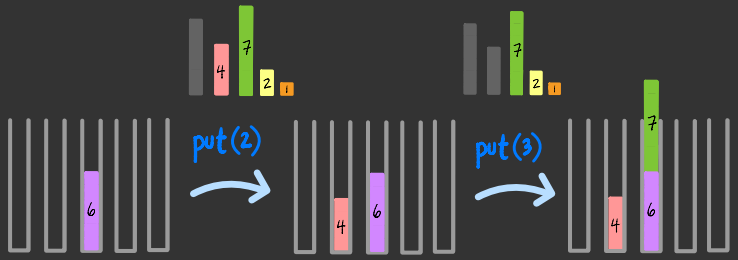
bin packing (formulation 1)

initial state:



n empty bins

action:



put next block into i^{th} bin

solution depth = your answer here

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

maximum depth = your answer here

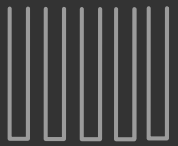
the length of the longest search path

branching factor = your answer here

the maximum number of successors of a search node

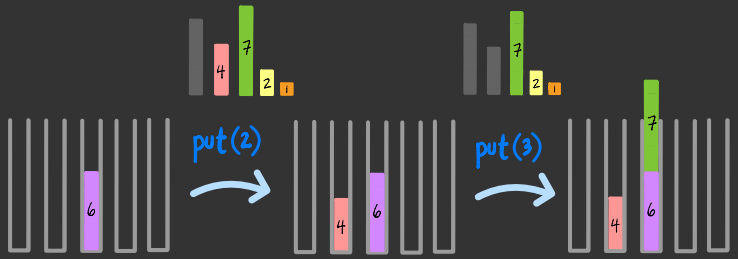
bin packing (formulation 1)

initial state:



n empty bins

action:



put next block into i^{th} bin

solution depth = n

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

maximum depth = n

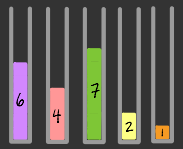
the length of the longest search path

branching factor = n

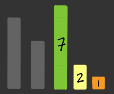
the maximum number of successors of a search node

bin packing (formulation 2)

initial state:



each block in its own bin



actions:

move leftmost unfrozen block to the left

freeze leftmost block

solution depth = your answer here

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

maximum depth = your answer here

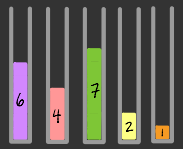
the length of the longest search path

branching factor = your answer here

the maximum number of successors of a search node

bin packing (formulation 2)

initial state:



each block in its own bin

actions:

move leftmost unfrozen block to the left

freeze leftmost block

solution depth = depends on initial state

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

maximum depth = $n + \sum_{i=0}^{n-1} i$

the length of the longest search path

$= n + \frac{n(n-1)}{2}$

branching factor = 2

the maximum number of successors of a search node