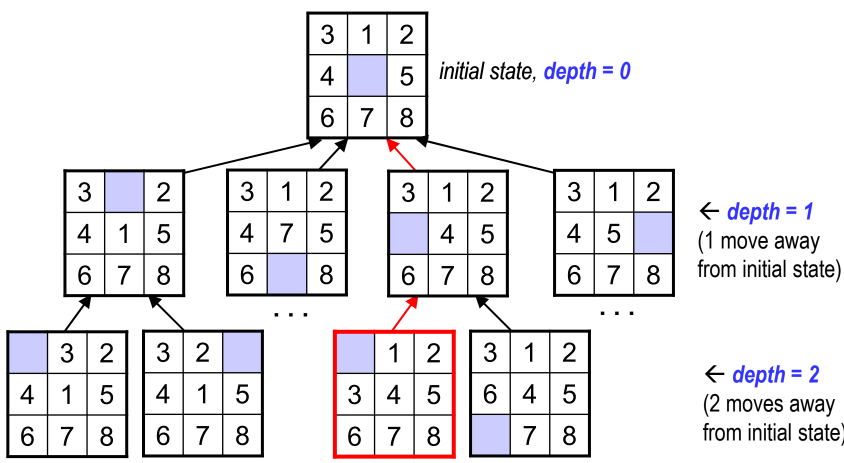
Final Project Revisited; Informed State-Space Search

Computer Science 111
Boston University

David G. Sullivan, Ph.D.

Recall: State-Space Search Tree

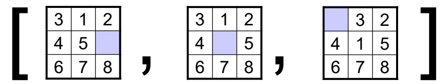
- The predecessor references connect the `State` objects, creating a structure known as a *tree*.



- When we reach a goal, we trace up the tree to get the solution – i.e., the sequence of moves from the initial state to the goal.

Part III: Initial `Searcher` Class

- The searcher object maintains a list of yet-to-be-tested states:



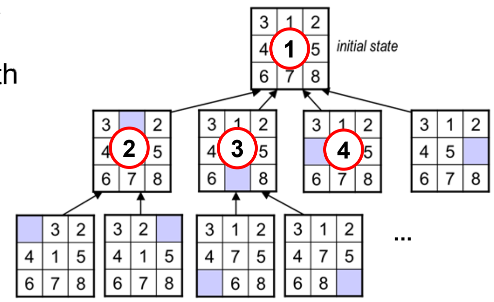
- Searcher methods include:
  - `next_state()` – get and return the next state that should be considered (removing it from the searcher's list of states)
    - in Part III – pick a state at random!
  - `find_solution(init_state)` – search from `init_state` until you find a goal state, and return it when it's found
    - ***see pseudocode from last week!***

Part IV: Subclasses for Other Search Algorithms

- Each algorithm will have its own type of searcher object.
  - with its own version of at least one of the key methods
  - ***take advantage of inheritance!***

Recall: Breadth-First Search (BFS)

- When choosing from the list of yet-to-be-tested states, choose one of the states with the ***smallest depth***.



- Thus, BFS considers:
  - all states at depth 0
  - all states at depth 1
  - all states at depth 2

  …

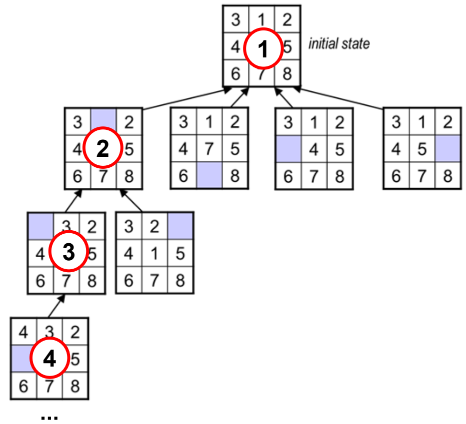
- The breadth-first searcher should follow FIFO ("first in, first out").
  - its `next_state()` should remove the state that has been in the list the longest

Recall: Features of Breadth-First Search

- It is *complete:* if there is a solution, BFS will find it.
- If each move has the same cost, BFS is *optimal*—it will find a minimal-cost solution.
- Key problems:
  - It can require too much *time.*
  - It can require too much *memory.*
    - *all* previously tested states must be kept in memory

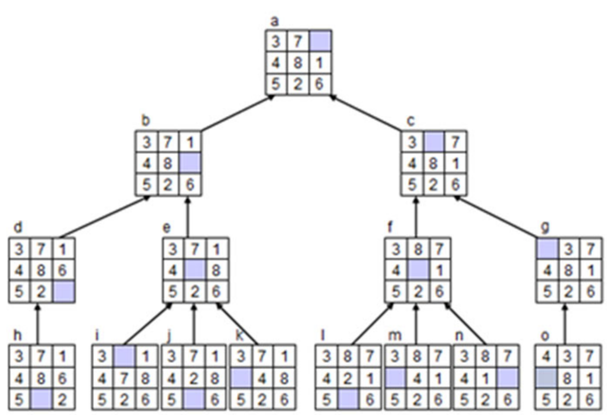
Depth-First Search (DFS)

- When choosing from the list of yet-to-be-tested states, choose one of the states with the ***largest depth***.
- Thus, DFS keeps going down a given path in the tree until it can't go any further.



- The depth-first searcher should follow LIFO ("last in, first out").
  - its `next_state()` should remove the state that was most recently added to the list

What are the first 4 states DFS would consider? (break ties alphabetically)



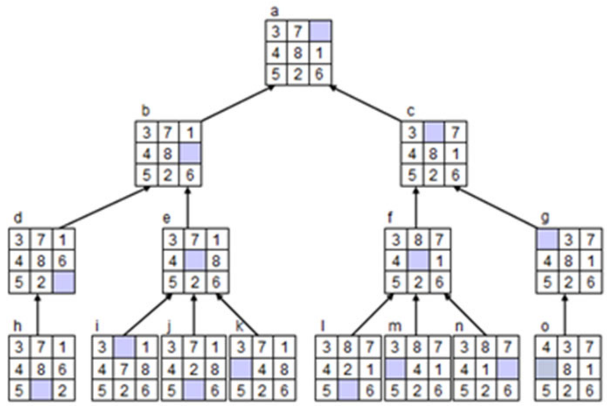
A. a, b, c, d

B. a, b, d, e

C. a, b, d, h

D. none of these

What are the first 4 states DFS would consider? (break ties alphabetically)



A. a, b, c, d

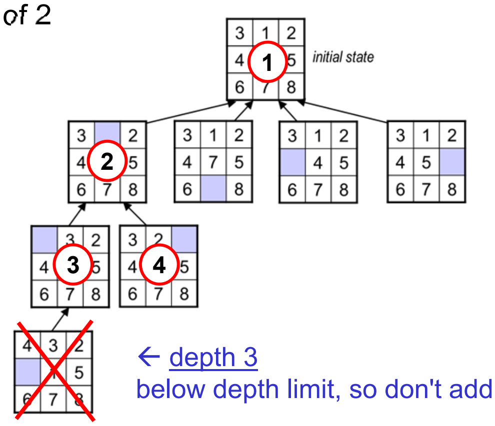
B. a, b, d, e

C. **a, b, d, h**

D. none of these

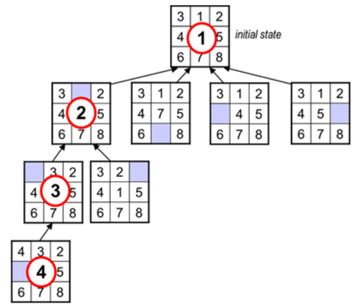
Imposing a Depth Limit

- DFS can end up going down very long paths.
  - can lead to solutions with too many steps
- To prevent this, we can use a *depth limit.*
  - the searcher won't add states whose depth > depth limit
  - example: depth limit of 2



Features of Depth-First Search

- Much better memory usage than BFS
  - DFS only stores a single path down the tree at a given time – along with the untested successors of states on that path



- What about time?
  - if there are many solutions, DFS can often find one quickly
  - if not, it can still be slow
- Key problem: it can get stuck going down long/"bad" paths.
  - ➔ thus, it is neither complete nor optimal.

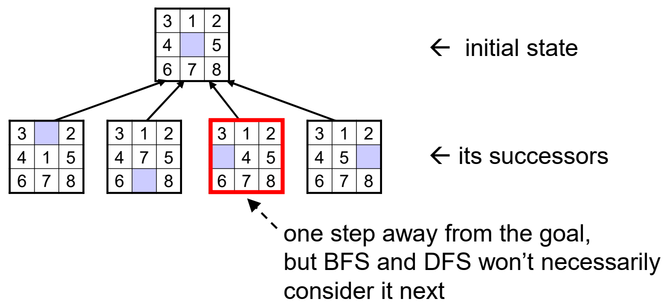
Comparing BFS and DFS

- BFS:
  - is *complete* – if there is a solution, it will eventually find it
  - is *optimal* – will find the solution with the fewest moves
  - can end up requiring too much time and memory
- DFS:
  - uses less memory than BFS
  - can also require too much time
  - is neither complete nor optimal!
- Examples: use CTRL-C as needed to terminate a search

```
>>> eight_puzzle('142635708', 'BFS', -1)
>>> eight_puzzle('142635708', 'DFS', -1)
>>> eight_puzzle('312065748', 'BFS', -1)
>>> eight_puzzle('312065748', 'DFS', -1)  # -1 -> 31
>>> eight_puzzle('603872541', 'BFS', -1)
>>> eight_puzzle('603872541', 'DFS', -1)  # -1 -> 31
```

Uninformed State-Space Search

- BFS and DFS are *uninformed* search algorithms.
  - always consider the states in a certain order
  - do not consider how close a given state is to the goal
- Example:



Part V: Informed State-Space Search

- *Informed* search algorithms attempt to consider more promising states first.
- They associate a *priority* with each successor state.
  - give a higher priority to states that seem closer to the goal
- When choosing the next state to consider, they select one with the highest priority.

Estimating the Remaining Cost

- We need some *heuristic function* h(x) that takes a state x and computes an estimate of the remaining cost.
  - heuristic = rule of thumb
- To find optimal solutions, we need an *admissable* heuristic – one that never overestimates the remaining cost.
  - it's okay to underestimate!

One Heuristic Function for the Eight Puzzle

- h1(x) = # of misplaced tiles in the board for state x
  - i.e., # of tiles that aren't where they belong in the goal state
  - example:

1	**4**	2
3	**5**	**8**
6		**7**

state s

	1	2
3	4	5
6	7	8

goal state

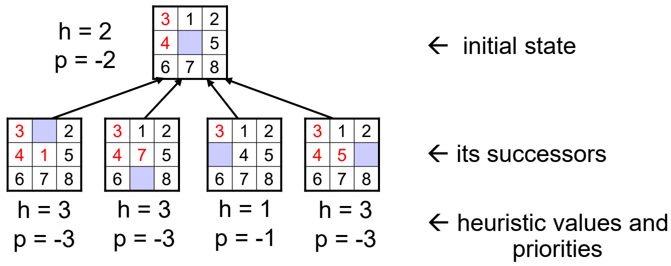
h1(s) = 5, because the 1, 4, 5, 7, and 8 tiles are misplaced

  - note that we do *not* include the blank in the count
- This heuristic is admissible (doesn't overestimate). Why?

  every misplaced tile requires at least one move to get to where it belongs
- In the final project, you will experiment with other heuristics!

Greedy Search

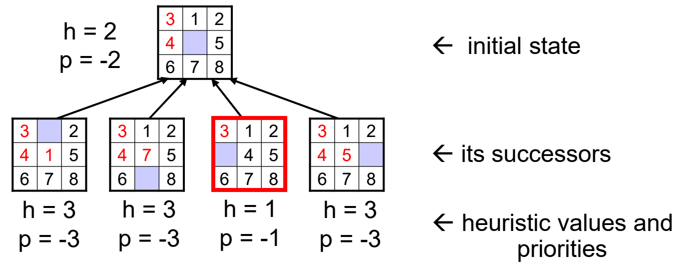
- Priority of state x, p(x) = −1 * h(x)
  - mult. by −1 so states closer to the goal have higher priorities



- Which successor would greedy test first?

Greedy Search

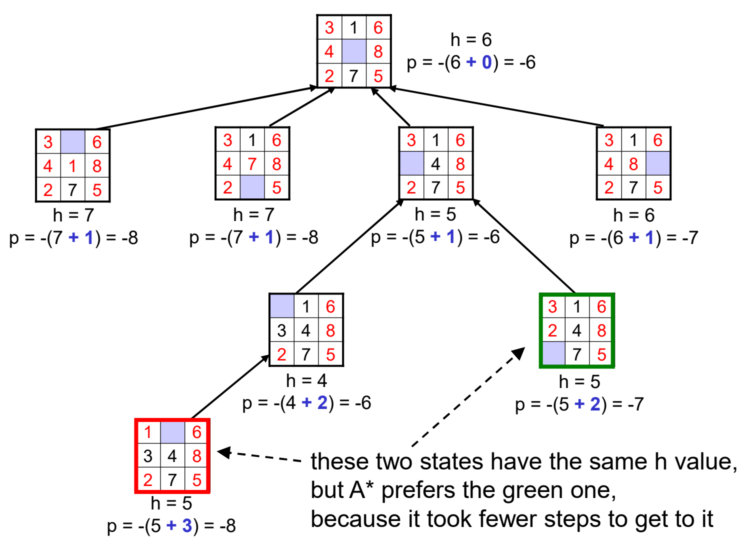
- Priority of state x, p(x) = −1 * h(x)
  - mult. by −1 so states closer to the goal have higher priorities



- Greedy search would test the red successor before the other ones, because it has the highest priority.
- However, greedy search is neither complete nor optimal!
  - key problem: fails to consider the cost that was already spent to get *to* the current state

A* Search

- Priority of state x, p(x) = −1 * (h(x) **+ g(x)**)
  where g(x) = the cost of getting from the initial state to x

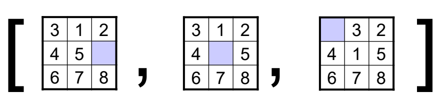


Characteristics of A*

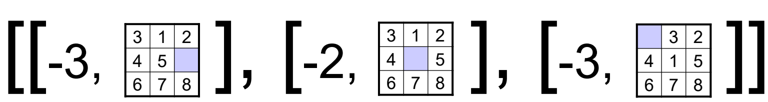
- Incorporating g(x) allows A* to find an optimal solution – one with the minimal *total* cost.
- Time and memory usage can still be problematic, but much less so than in uninformed search!

Implementing Informed Search

- In *uninformed* search, the searcher object maintains a list of untested states.



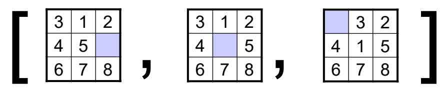
- In *informed* search, the searcher will maintain of a *list of lists*!



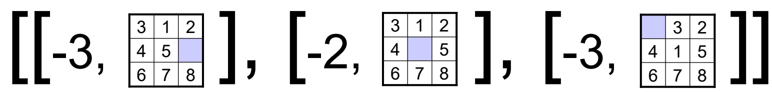
- How could it find the next state to be tested?

Implementing Informed Search

- In *uninformed* search, the searcher object maintains a list of untested states.



- In *informed* search, the searcher will maintain of a *list of lists*!



- How could it find the next state to be tested?
  using max()!

Part VI: Make Things Faster!

- With our simple heuristic, even A* can be slow.
  - for puzzles that require many moves to solve
- You will devise your own heuristics.
  - try to get better estimates of the remaining cost
  - be careful not to overestimate!
- You will also perform experiments comparing approaches.

What's Left in CS 111

S	M	T	W	T	F	S
28	29	30	1	$\boxed{2}$	3	4
$\boxed{5}$	6	7	8	$\boxed{9}$	10	11
12	13	$\boxed{14}$	15	16	17	18

Problem Set 10

- FSM problems
- 50 points
- due **Sun (12/5)**

Final Project

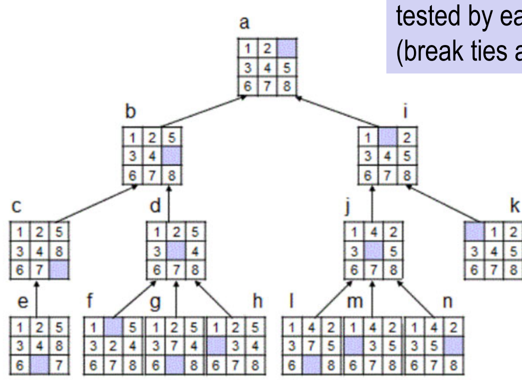
- late submissions of parts I & II through tonight
- full project due **12/9**

Final exam:

- **12/14**, 9-11 am
- email me with exam conflicts

Extra Practice

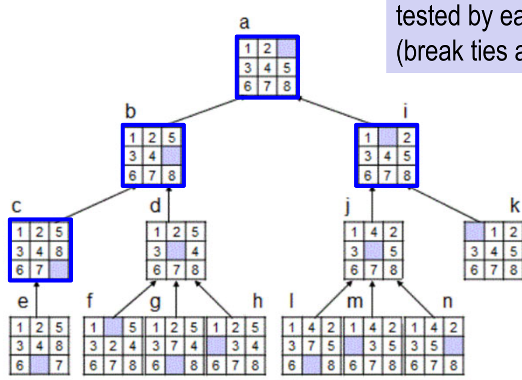
What are the first four states tested by each algorithm? (break ties alphabetically)



- BFS
- DFS (no depth limit)
- DFS (depth limit of 2)
- DFS (depth limit of 3)

Extra Practice

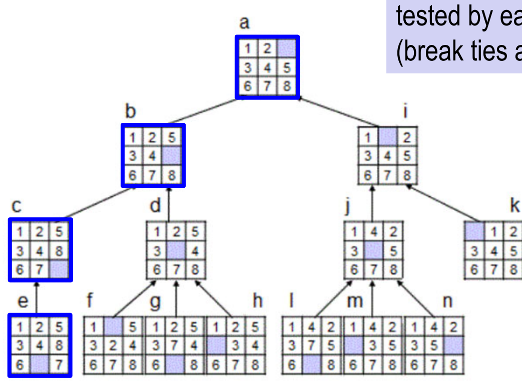
What are the first four states tested by each algorithm? (break ties alphabetically)



- *BFS* – a, b, i, c
- DFS (no depth limit)
- DFS (depth limit of 2)
- DFS (depth limit of 3)

Extra Practice

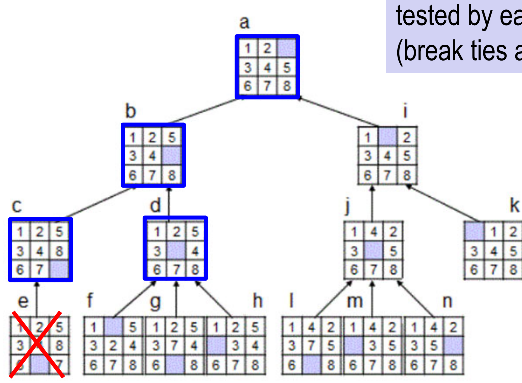
What are the first four states tested by each algorithm? (break ties alphabetically)



- BFS – a, b, i, c
- *DFS (no depth limit)* – a, b, c, e
- DFS (depth limit of 2)
- DFS (depth limit of 3)

Extra Practice

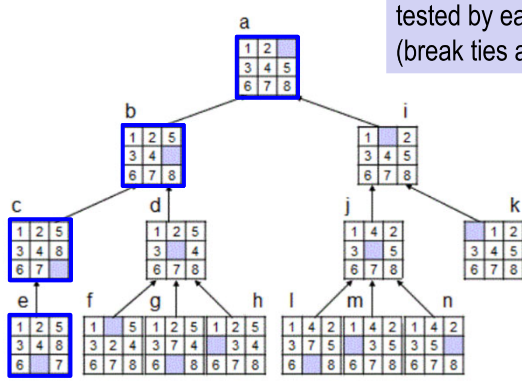
What are the first four states tested by each algorithm? (break ties alphabetically)



- BFS – a, b, i, c
- DFS (no depth limit) – a, b, c, e
- *DFS (depth limit of 2)* – a, b, c, d
- DFS (depth limit of 3)

Extra Practice

What are the first four states tested by each algorithm? (break ties alphabetically)



- BFS – a, b, i, c
- DFS (no depth limit) – a, b, c, e
- DFS (depth limit of 2) – a, b, c, d
- *DFS (depth limit of 3)* – a, b, c, e (would test d next)