Chapter 2 State-Space Search

COMP 472 Artificial Intelligence

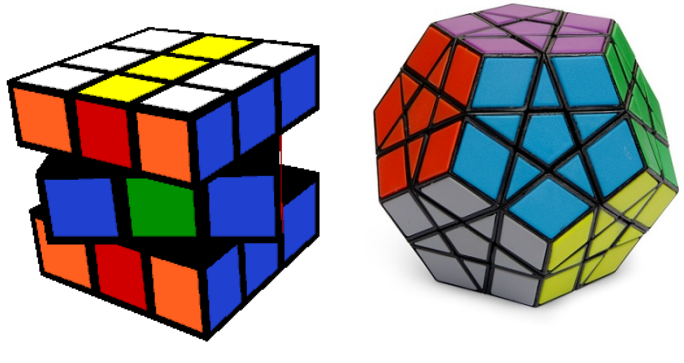
Russell & Norvig – Chapter 3 & Section 4.11

Some slides from: robotics.stanford.edu/~latombe

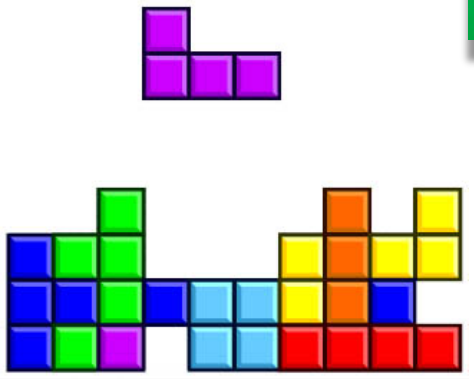
Motivation

8-puzzle

Google Map



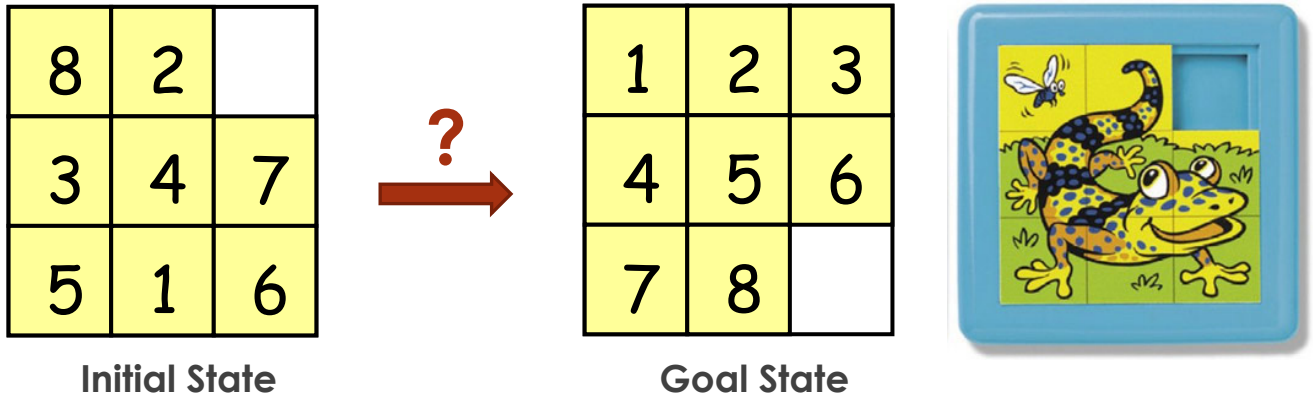
Rubik's cube



Tetris

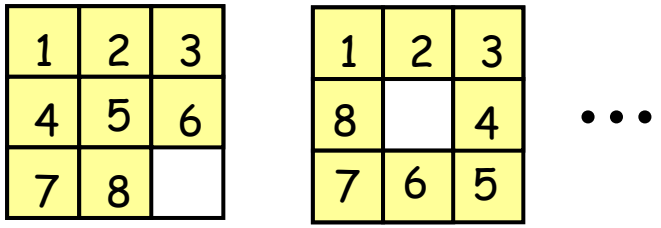
Example: 8-Puzzle

- **State**: Any arrangement of 8 numbered tiles and an empty tile on a 3x3 board.



Initial State **Goal State**

- There are several standard goals states for the 8-puzzle



Example: (n^2-1)-Puzzle

8	2	
3	4	7
5	1	6

8-puzzle

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

15-puzzle

. . . .

15 - Puzzle

- Invented in 1874 by Noyes Palmer Chapman … but Sam Loyd claimed he invented it!

SAM LOYD,

Journalist and Advertising Expert,

ORIGINAL

Games, Novelties, Supplements, Souvenirs,

Etc., for Newspapers.

Unique Sketches, Novelties, Puzzles, &c.,

FOR ADVERTISING PURPOSES.

Author of the famous

"Get Off The Earth Mystery," "Trick Donkeys,"

"15 Block Puzzle," "Pigs in Clover,"

"Parchesi," Etc., Etc.

P. O. BOX 376.

New York, April 15 1903

15 - Puzzle

- Sam Loyd even offered $1,000 of his own money to the first person who would solve the following problem:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

?

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

15 - Puzzle

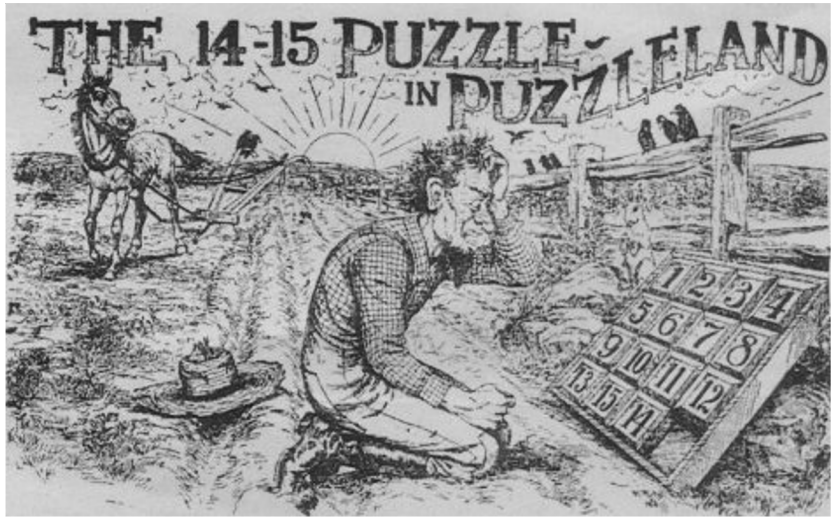
- Sam Loyd even offered $1,000 of his own money to the first person who would solve the following problem:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

?

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

- **But no one ever won the prize …**



What is State Space?

- Many AI problems, can be expressed in terms of going from an ***initial state*** to a **goal state**
  - *Ex: to solve a puzzle, to drive from home to Concordia…*

Often, there is no direct way to find a solution to a problem, but we can list the possibilities and search through them

- **Brute force search:**

  generate and search <u>all</u> possibilities (but inefficient)

- **Heuristic search:**

  only try the possibilities that you *think* (based on your current best guess) are more likely to lead to good solutions

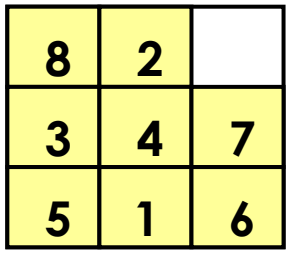
What is State Space?

- Problem is represented by:
  1. Initial State
     - starting state e.g. unsolved puzzle, being at home
  2. Set of operators
     - actions responsible for transition between states
  3. Goal test function
     - Applied to a state to determine if it is a goal state
     - ex. solved puzzle, being at Concordia
  4. Path cost function
     - Assigns a cost to a path to tell if a path is preferable to another

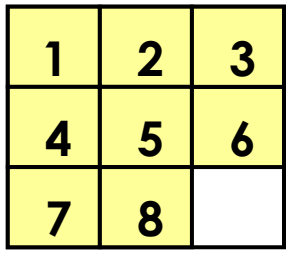
What is State-Space Search?

- **Search space:** the set of all states that can be reached from the initial state by any sequence of action
- **Search algorithm:** how the search space is visited

Example: 8-Puzzle



Initial state



Goal state

- Set of operators:

blank moves up, blank moves down, blank moves left, blank moves right

- Goal test function:

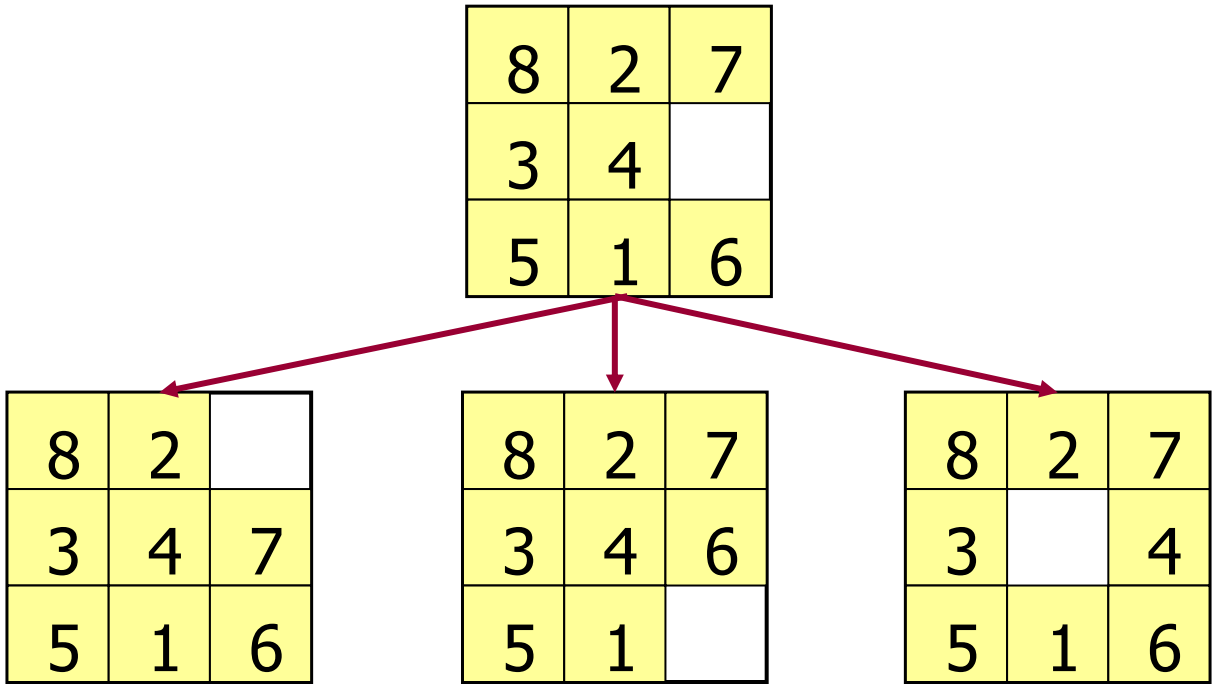
state matches the goal state

- Path cost function:

each movement costs 1

so the path cost is the length of the path (the number of moves)

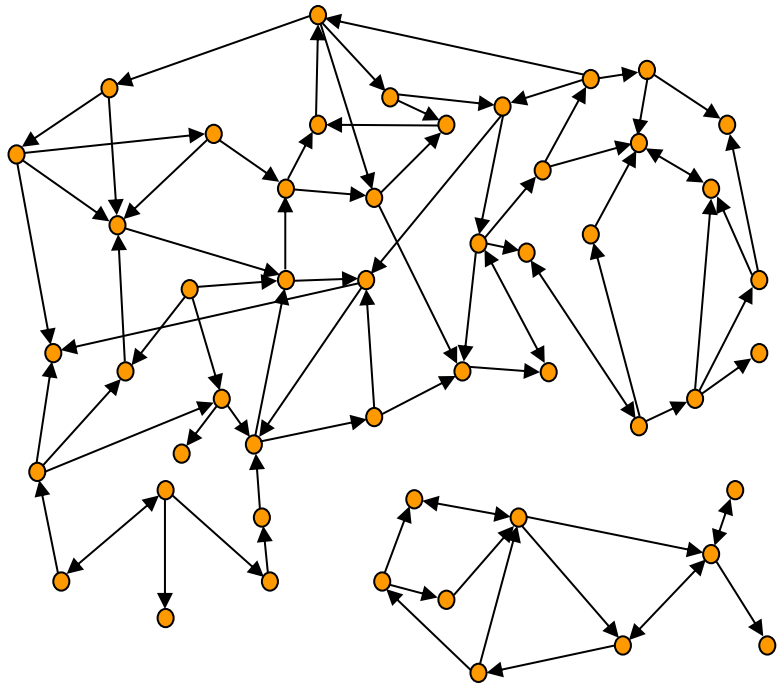
Example: 8-Puzzle: Successor Function



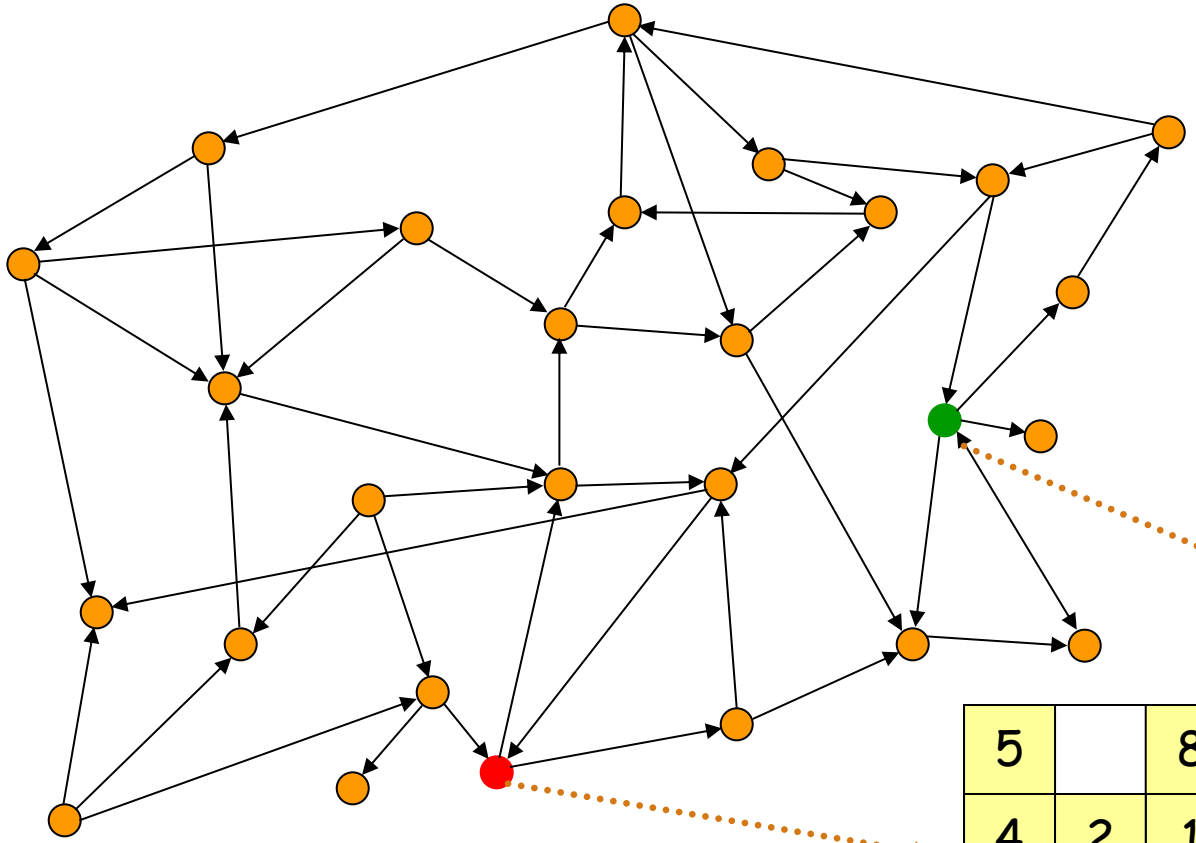
- Search is about the exploration of alternatives

State-Space Graph

- Each state is represented by a distinct node
- An arc (or edge) connects a node s to a node s' if s' $\in$ SUCCESSOR(s)
- The state graph may contain more than one connected component



State-Space Graph



5		8
4	2	1
7	3	6

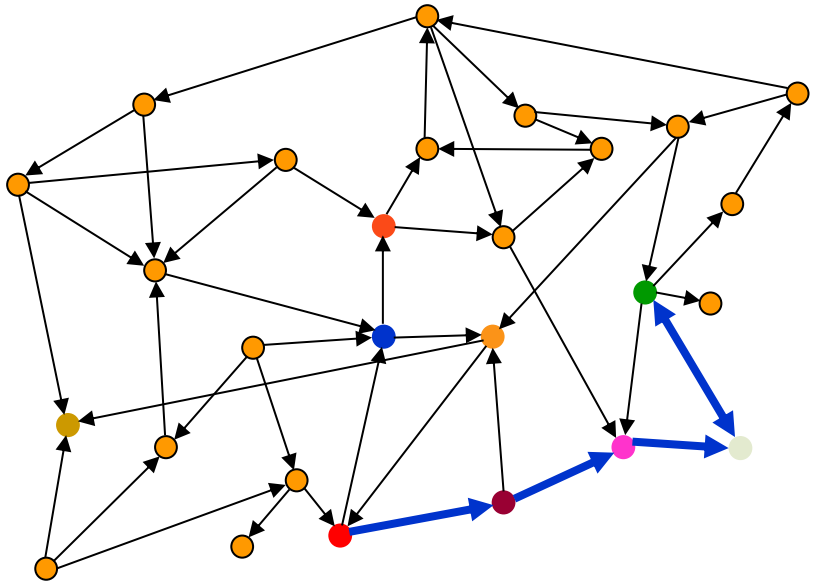
Initial state

1	2	3
4	5	6
7	8	

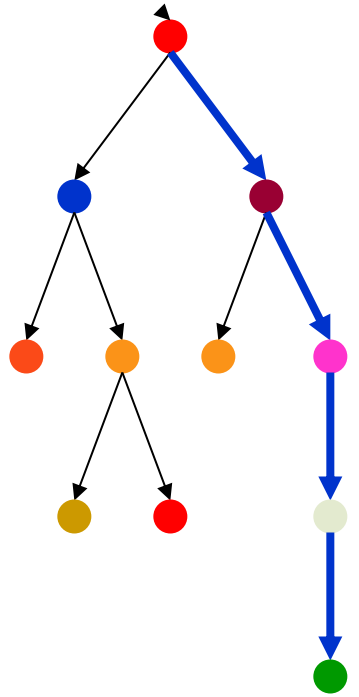
Goal state

State-Space as a Search Tree

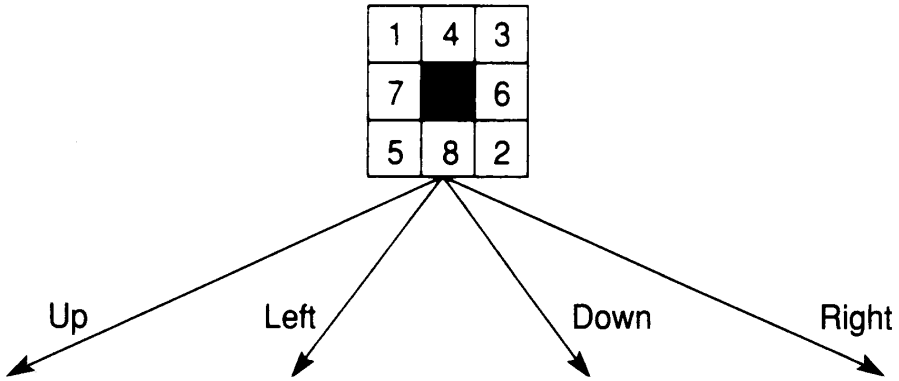
- In graph representation, cycles can prevent termination
  - Blind search without cycle check may never terminate
- Use a tree representation, and cycle checking



Search Tree

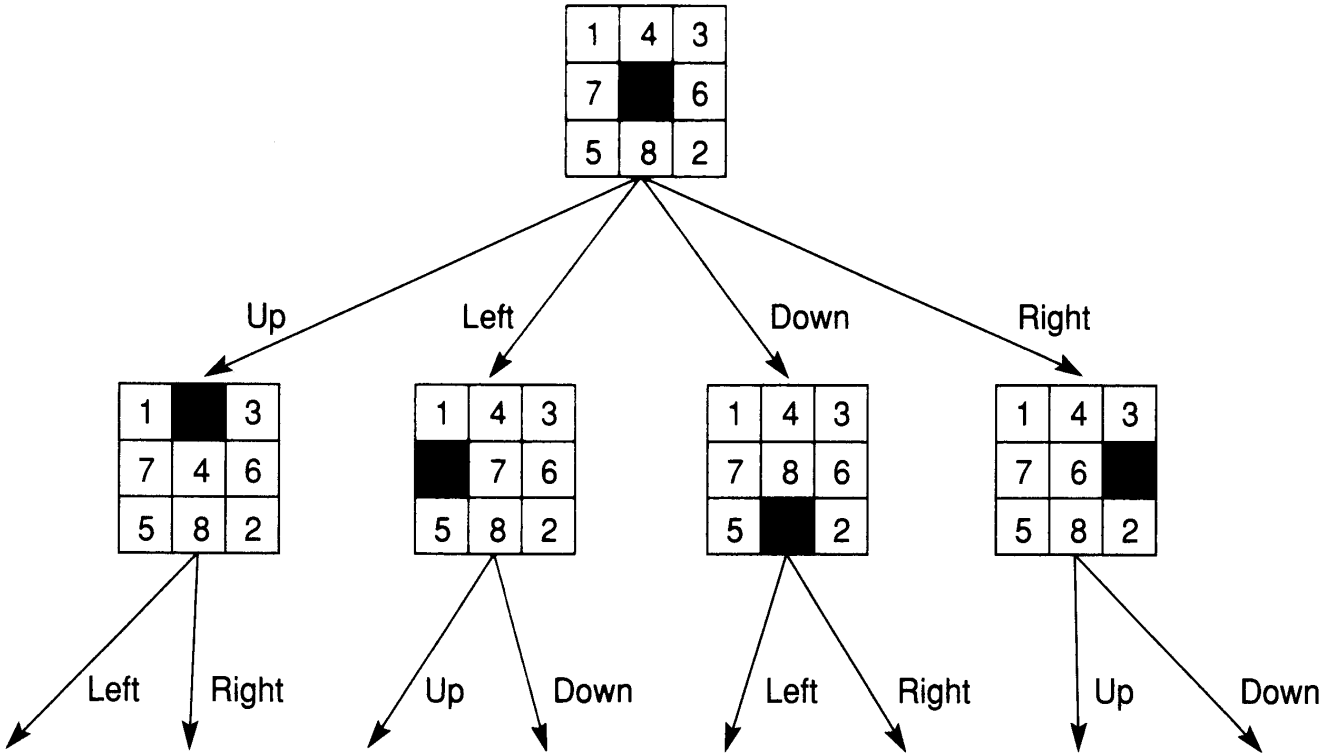


State-Space for 8-Puzzle



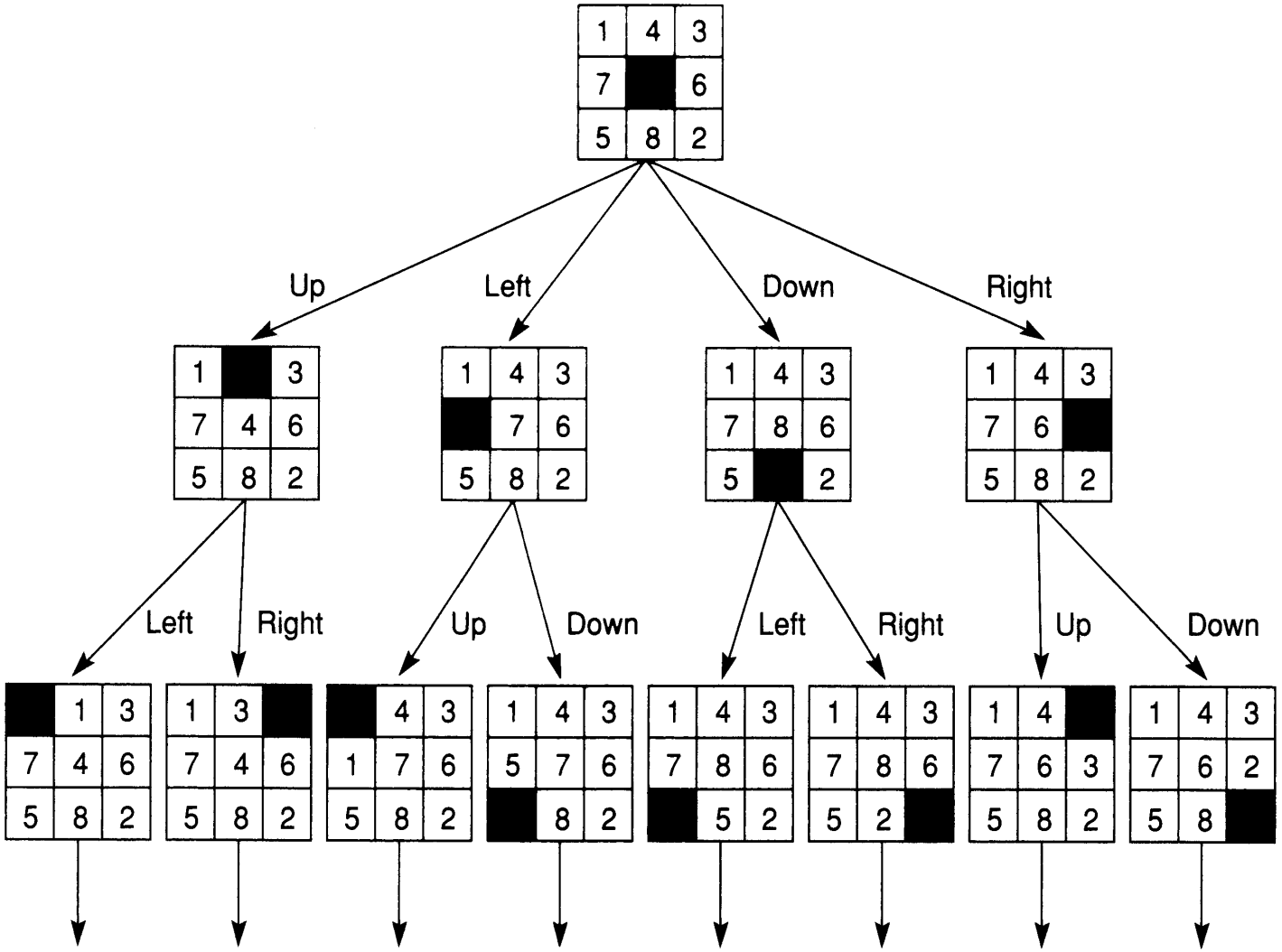
source: G. Luger (2005)

State-Space for 8-Puzzle



source: G. Luger (2005)

State-Space for 8-Puzzle



source: G. Luger (2005)

State-Space of (n^2-1) - Puzzle

- **Number of states:**
  - 8-puzzle --> 9! = 362,880 states
  - 15-puzzle --> 16! ~ 2.09 x 10^{13} states
  - 24-puzzle --> 25! ~ 10^{25} states
- **At 100 millions states/sec:**
  - 8-puzzle --> 0.036 sec
  - 15-puzzle --> ~ 55 hours
  - 24-puzzle --> > 10^9 years

Types of State-Space Search

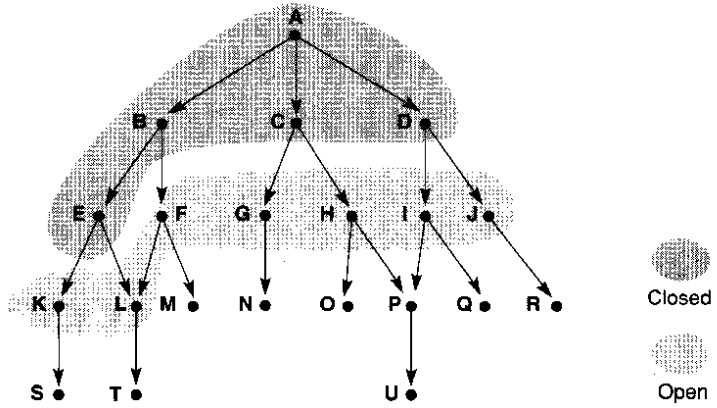
- **State-Space Search**
  - Uninformed Search
    - Breadth-first and Depth-first
    - Depth-limit Search
    - Iterative Deepening
    - Uniform Cost
  - Informed Search
    - Hill Climbing
    - Best – Frist
    - Design Heuristics
    - A*

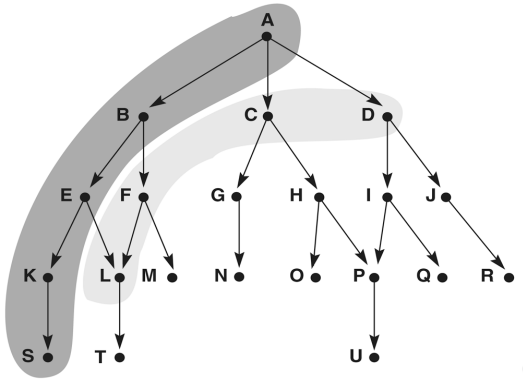
Uninformed VS Informed Search

- Uninformed Search -- systematically explore the alternatives
  - aka: systematic/exhaustive/blind/brute force search
  - Breadth-first
  - Depth-first
  - Uniform-cost
  - Depth-limited search
  - Iterative deepening search
  - Bidirectional search
  - …
- Informed Search -- try to choose smartly
  - Hill Climbing
  - Best – Frist
  - Design Heuristics
  - A*

Breadth-first VS Depth-first Search

- Determine order of examining states
  - **Breadth First Search (BFS)**
    - Visit siblings before successors, i.e., visit level by level
  - **Depth First Search (DFS)**
    - Visit successors before siblings



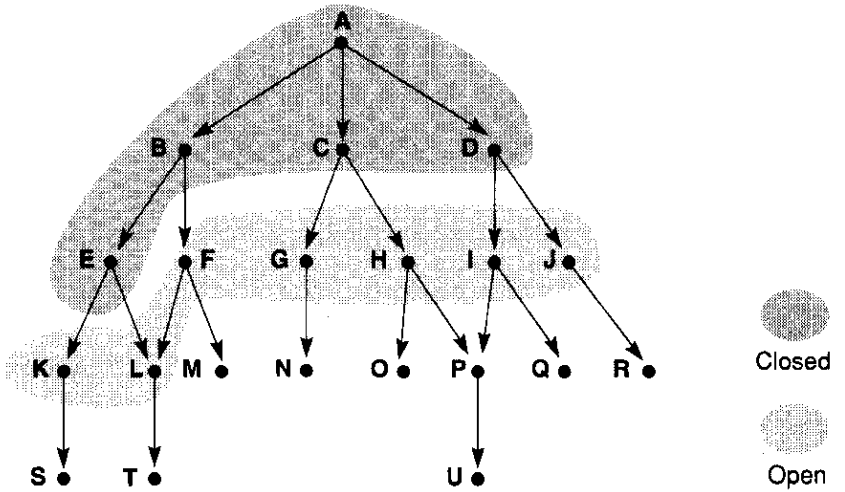


Data Structures

- ***open list***
  - lists generated states not yet expanded
  - order of states controls order of search
- ***closed list***
  - stores all the nodes that have already been visited (to avoid cycles).
- E.g.

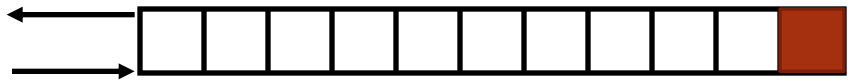
  Closed = [A, B, C, D, E]

  Open = [F, G, H, I, J, K, L]

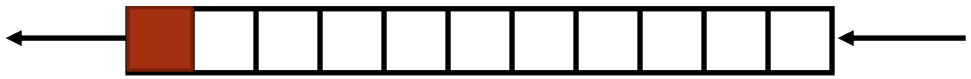


Data Structures

- DFS and BFS differ only in the way they order nodes in the open list:
  - DFS uses a ***stack***:
    - nodes are added on the top of the list.



  - BFS uses a ***queue***:
    - nodes are added at the end of the list.

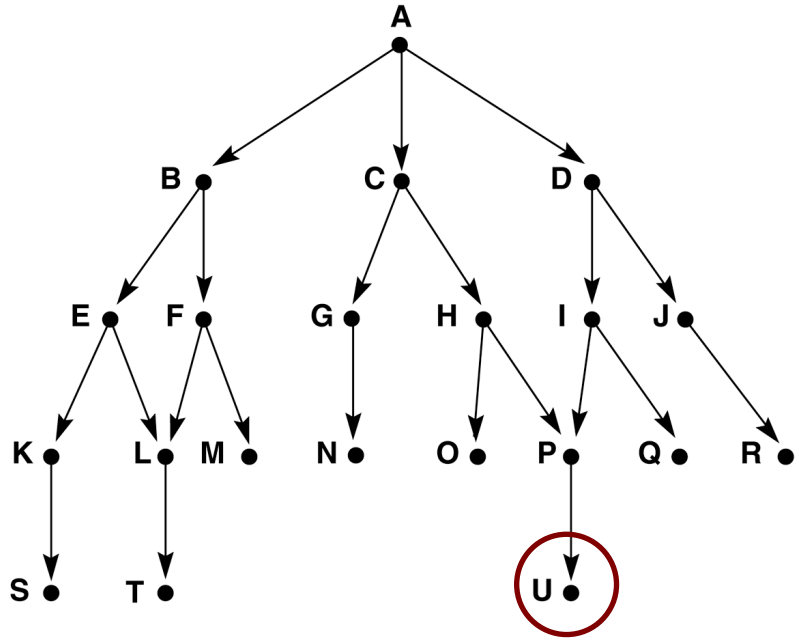


Breadth-First Search Algorithm

```
begin
   open := [Start];                                              % initialize
   closed := [ ];
   while open ≠ [ ] do                                           % states remain
      begin
         remove leftmost state from open, call it X;
            if X is a goal then return SUCCESS                  % goal found
               else begin
                  generate children of X;
                  put X on closed;
                  discard children of X if already on open or closed;   % loop check
                  put remaining children on right end of open           % queue
               end
      end
   return FAIL                                                   % no states left
end.
```

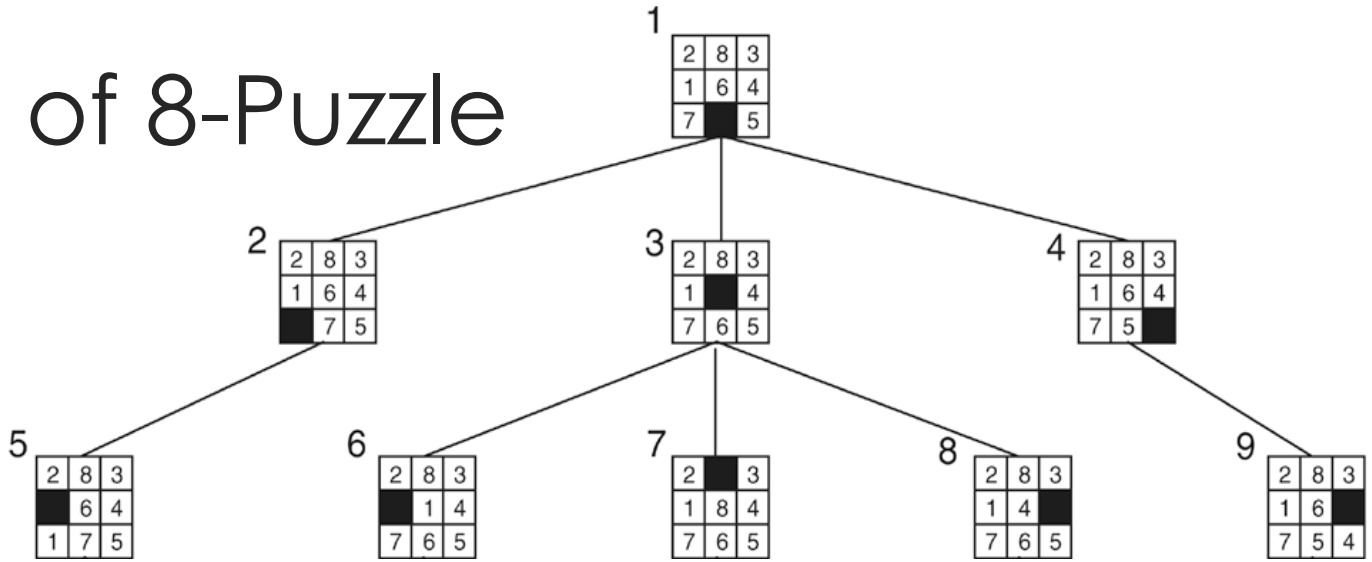
Breadth-First Search Example

- BFS uses a ***queue***.
- Assume U is goal State.

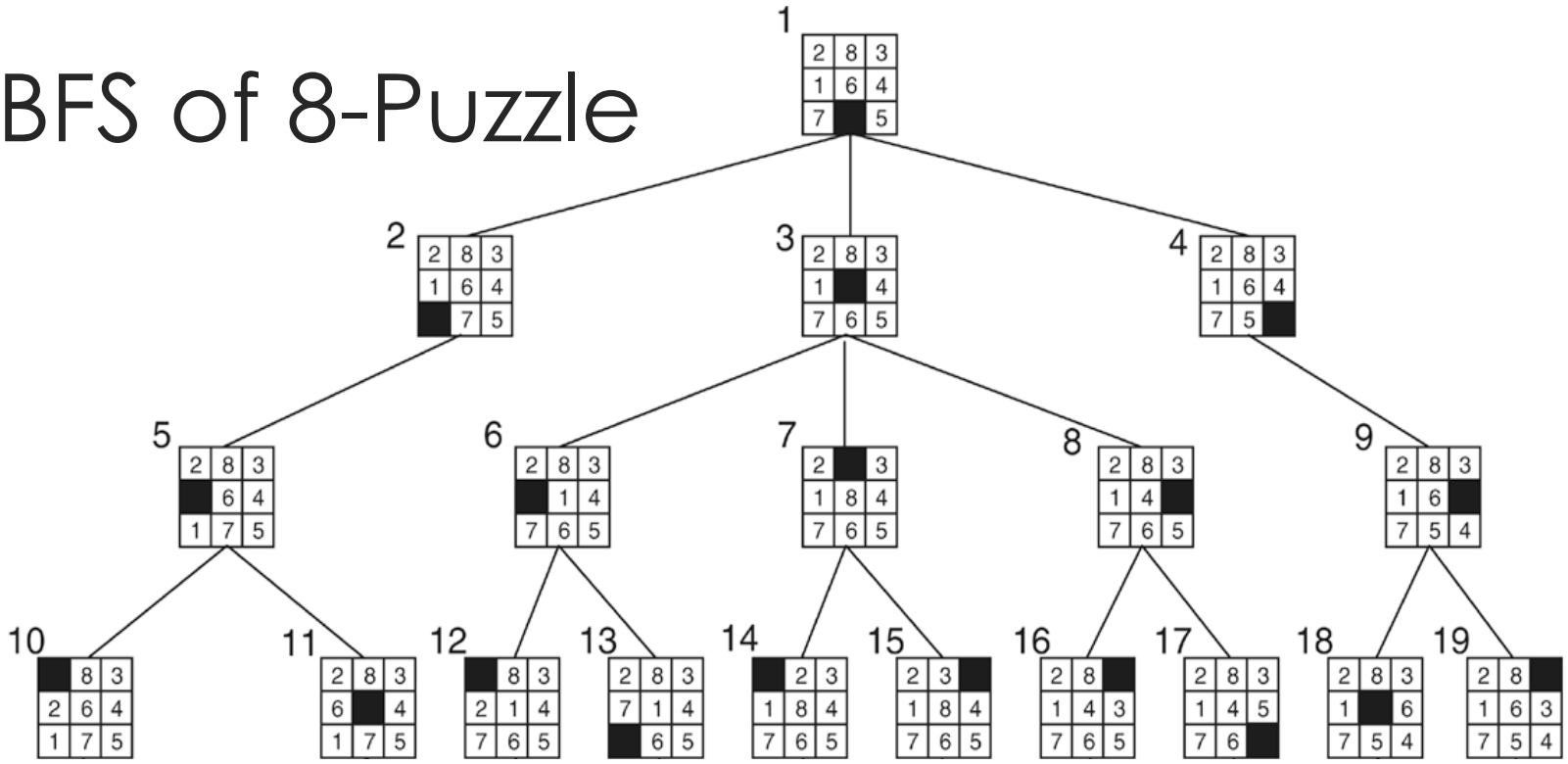


1. open = [A-null] closed = []
2. open = [B-A C-A D-A] closed=[A]
3. open = [C-A D-A E-B F-B] closed = [B A]
4. open = [D-A E-B F-B G-C H-C] closed = [C B A]
5. open = [E-B F-B G-C H-C I-D J-D] closed = [D C B A]
6. open = [F-B G-C H-C I-D J-D K-E L-E] closed = [E D C B A]
7. open = [G-C H-C I-D J-D K-E L-E M-F] *as L is already on closed* closed = [F E D C B A]
8. and so on until either U is found or open = []

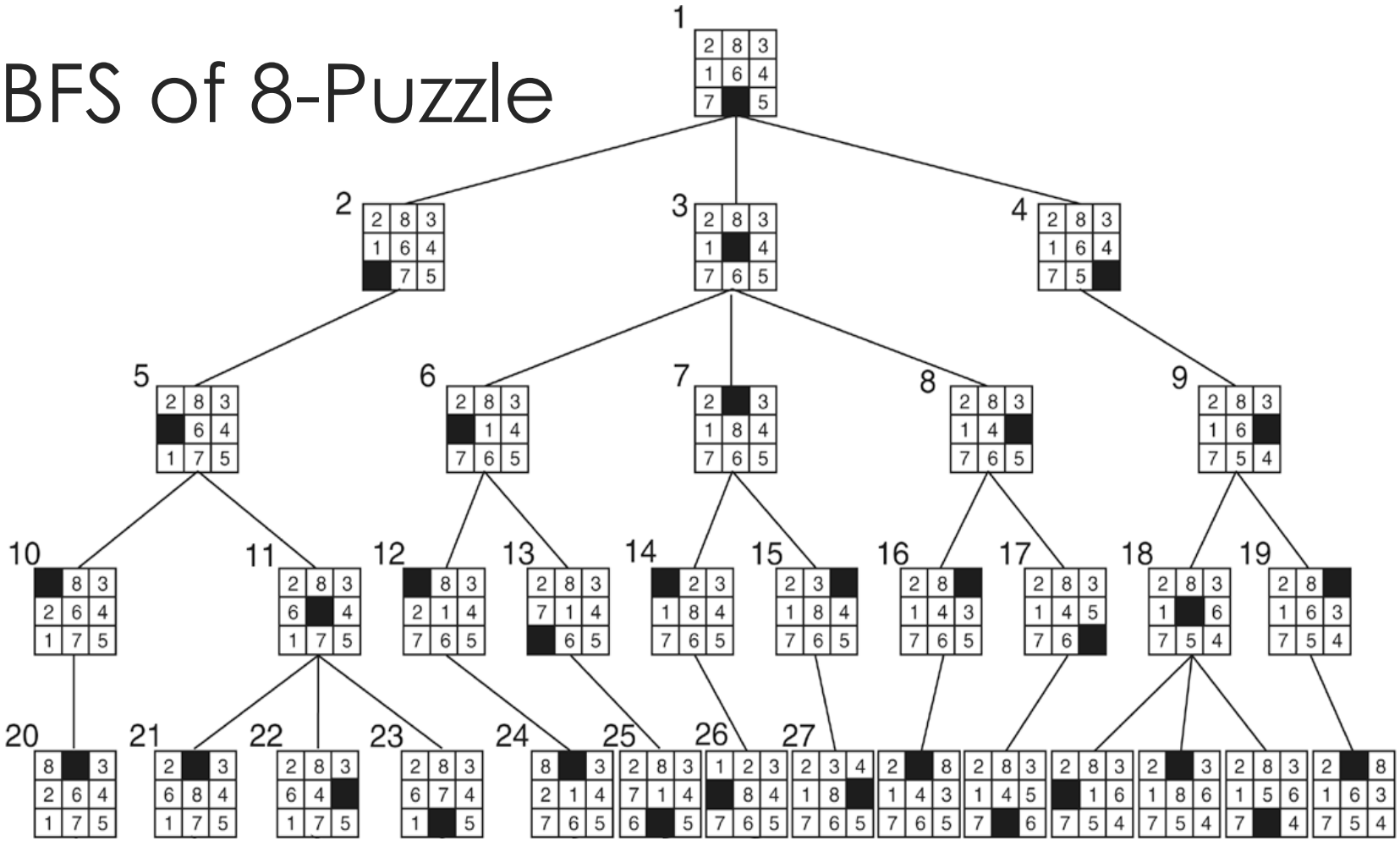
BFS of 8-Puzzle



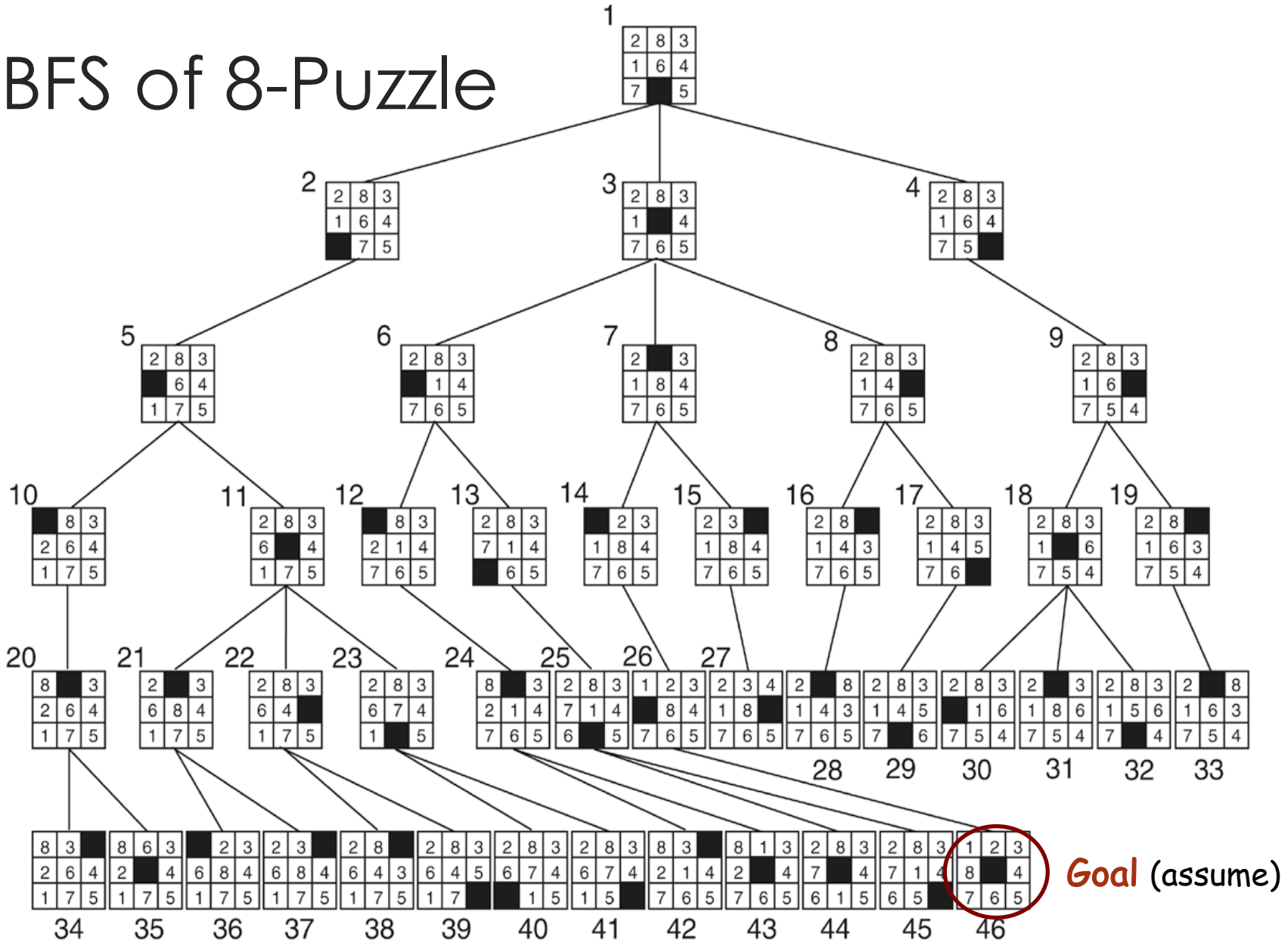
BFS of 8-Puzzle



BFS of 8-Puzzle



BFS of 8-Puzzle

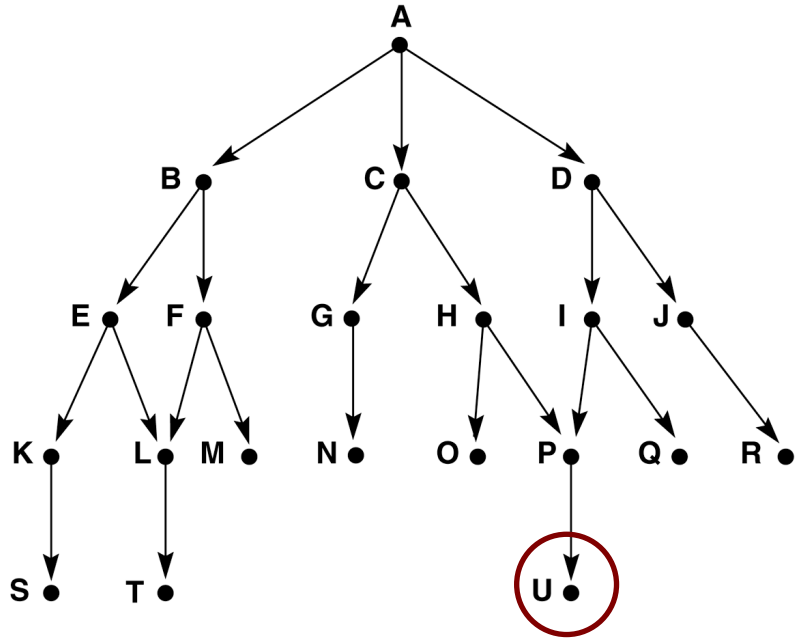


Depth-First Search Algorithm

```
begin
    open := [Start];
    closed := [];
    while open != [] do
    begin
        remove leftmost state from open, call it X // pop X from open
        if X is a goal then return SUCCESS
        else if X is not in closed
                begin
                    put X in closed
                    generate children of X and insert on left of open // i.e. push children of X in open
                end
    end
    return FAIL
end
```

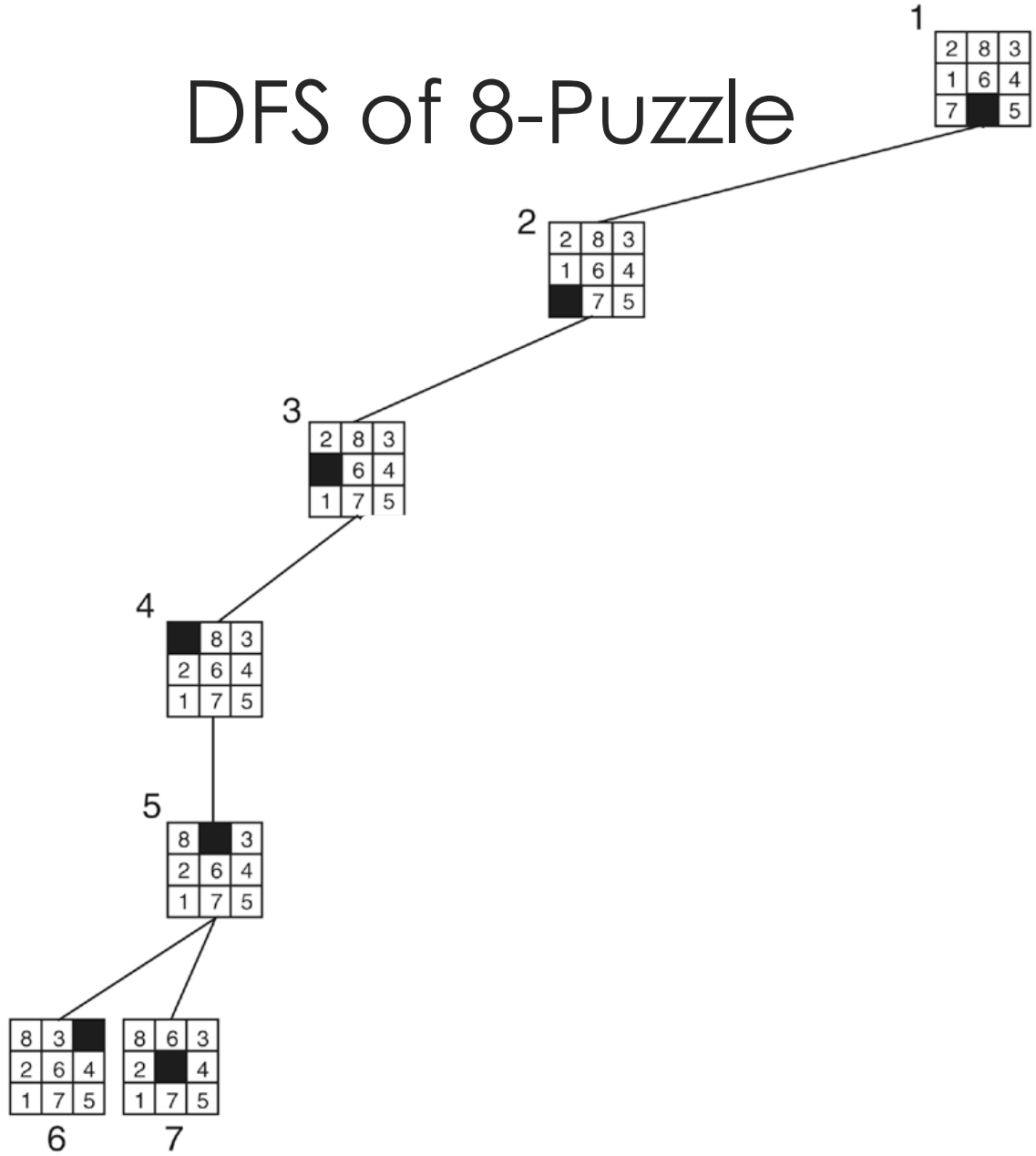
Depth-First Search Example

- DFS uses a ***stack***.
- Assume U is goal State.

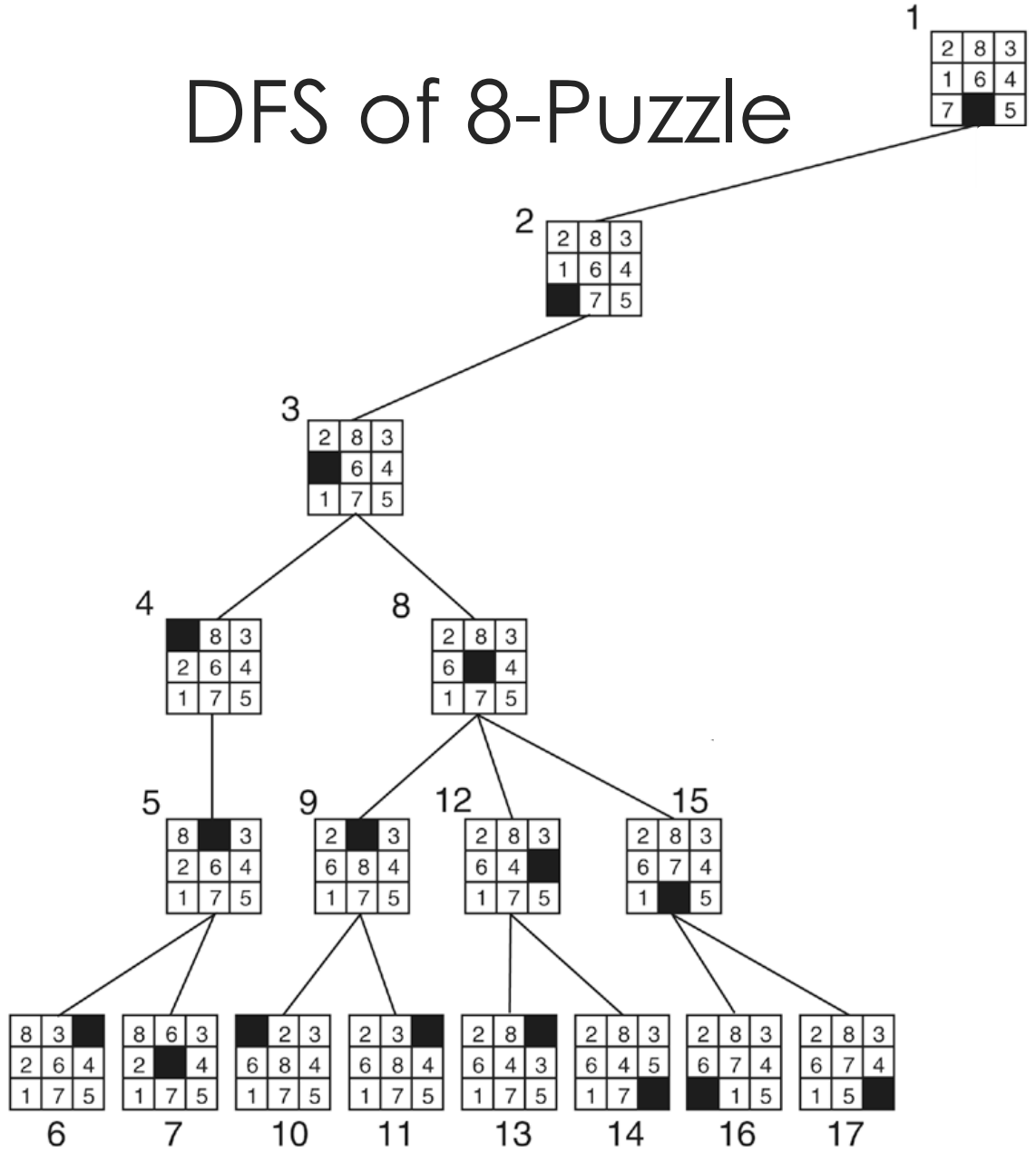


1. open = [A-null] closed = []
2. open = [B-A C-A D-A] closed [A]
3. open = [E-B F-B C-A D-A] closed = [B A]
4. open = [K-E L-E F-B C-A D-A] closed = [E B A]
5. open = [S-K L-E F-B C-A D-A] closed = [K E B A]
6. open = [L-E F-B C-A D-A] closed = [S K E B A]
7. open = [T-L F-B C-A D-A] closed = [L S K E B A]
8. open = [F-B C-A D-A] closed = [T L S K E B A]
9. open = [M-F C-A D-A] *as L is already on closed* closed = [F T L S K E B A]
10. open = [C-A D-A] closed = [M F T L S K E B A]
11. open = [G-C H-C D-A] closed = [C M F T L S K E B A]
12. and so on until either U is found or open = []

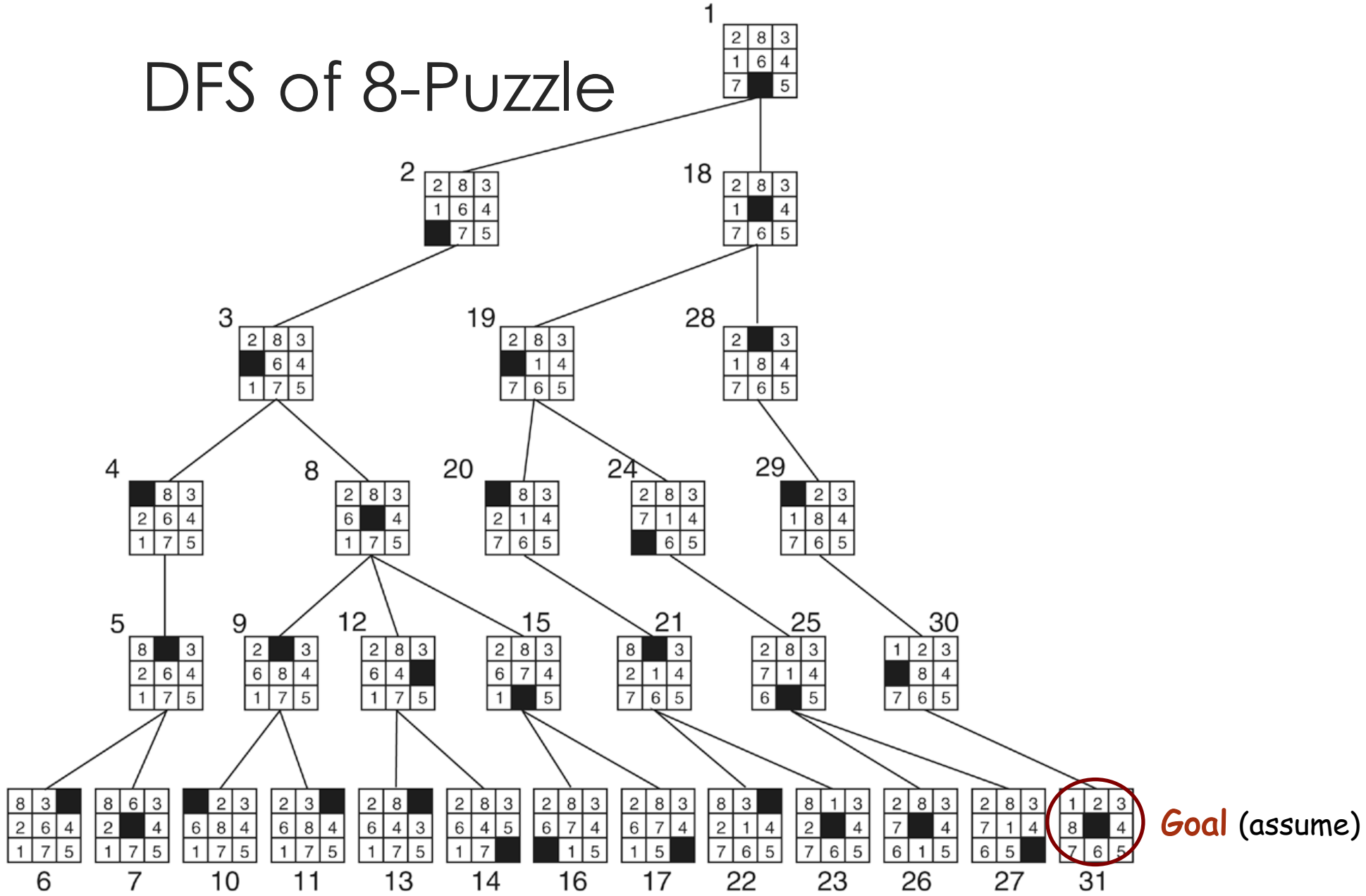
DFS of 8-Puzzle



DFS of 8-Puzzle



DFS of 8-Puzzle



BFS VS DFS

- **Breadth-first:**
  - Complete: always finds a solution if it exists
  - Optimal: always finds shortest path

  But:
  - inefficient if branching factor **B** is very high
  - memory requirements high -- exponential space for states required: $\mathbf{B^n}$
- **Depth-first:**
  - Not complete (ex. may get stuck in a long branch or infinite branch if no cycle-checking)
  - Not optimal (will not find the shortest path)

  But:
  - Requires less memory -- only memory for states of one path needed: $\mathbf{B \times n}$
  - May find the solution without examining much of the search space
  - Efficient if solution path is known to be long

Types of State-Space Search

- **State-Space Search**
  - ***Uninformed Search***
    - ***Breadth-first and Depth-first***
    - ***Depth-limit Search***
    - Iterative Deepening
    - Uniform Cost
  - Informed Search
    - Hill Climbing
    - Best – Frist
    - Design Heuristics
    - A*

Depth – Limit Search

Compromise for DFS :

- Do depth-first but with depth cutoff k

(depth at which nodes are not expanded)

- Three possible outcomes:
  - Solution
  - Failure (no solution)
  - Cutoff (no solution within cutoff)

Types of State-Space Search

- **State-Space Search**
  - ***Uninformed Search***
    - ***Breadth-first and Depth-first***
    - ***Depth-limit Search***
    - ***Iterative Deepening***
    - Uniform Cost
  - Informed Search
    - Hill Climbing
    - Best – Frist
    - Design Heuristics
    - A*

Iterative Deepening

Compromise between BFS and DFS:

- use depth-first search, but
- with a maximum depth before going to next level

- Repeats depth first search with gradually increasing depth limits
  - Requires little memory (fundamentally, it's a depth first)
  - Finds the shortest path (limited depth)

- Preferred search method when there is a large search space and the depth of the solution is unknown

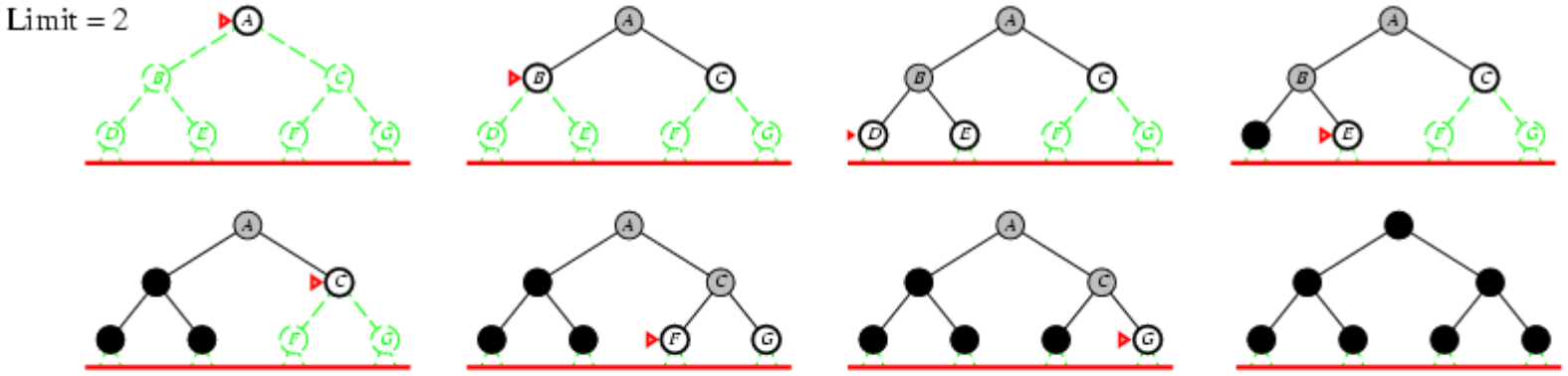
Iterative Deepening Example

Limit = 0

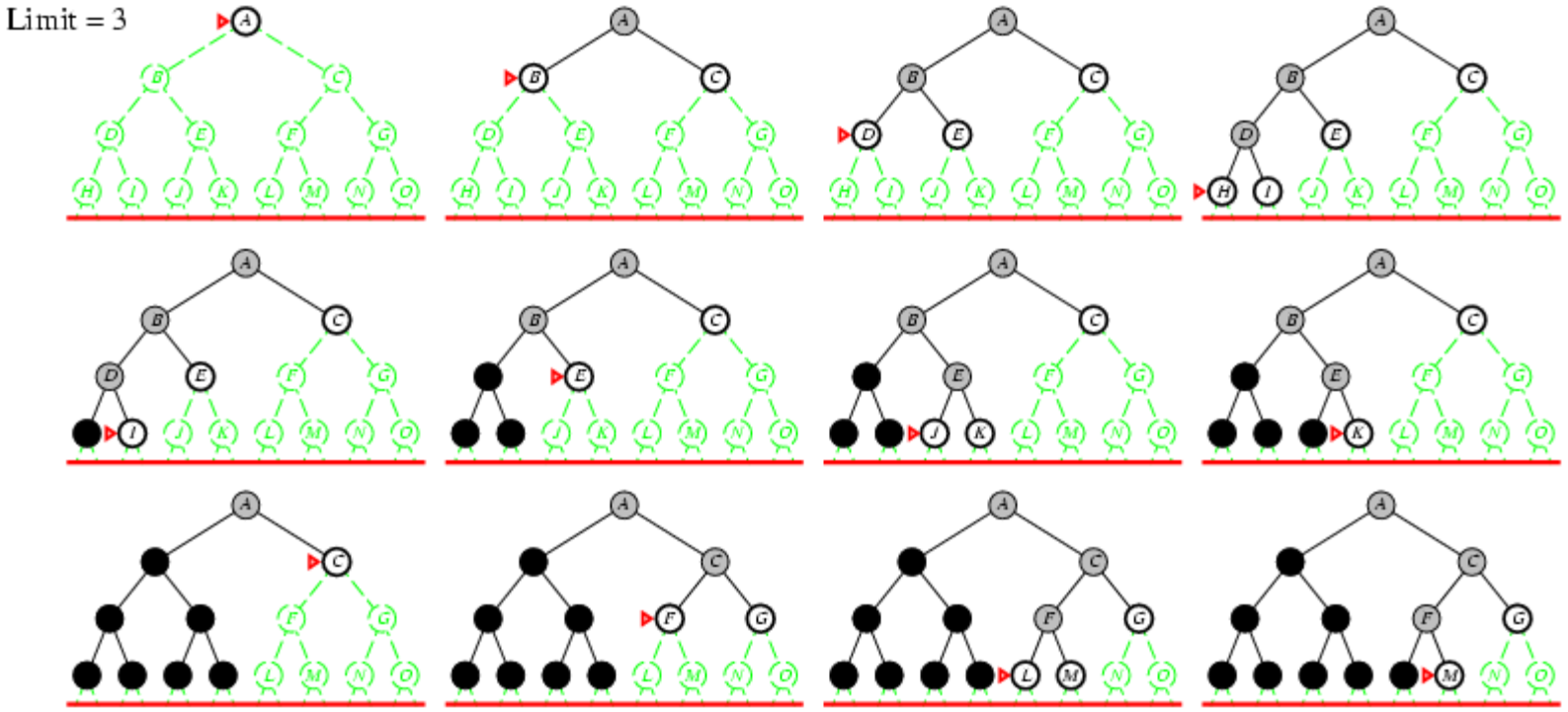
Iterative Deepening Example

Limit = 1

Iterative Deepening Example



Iterative Deepening Example

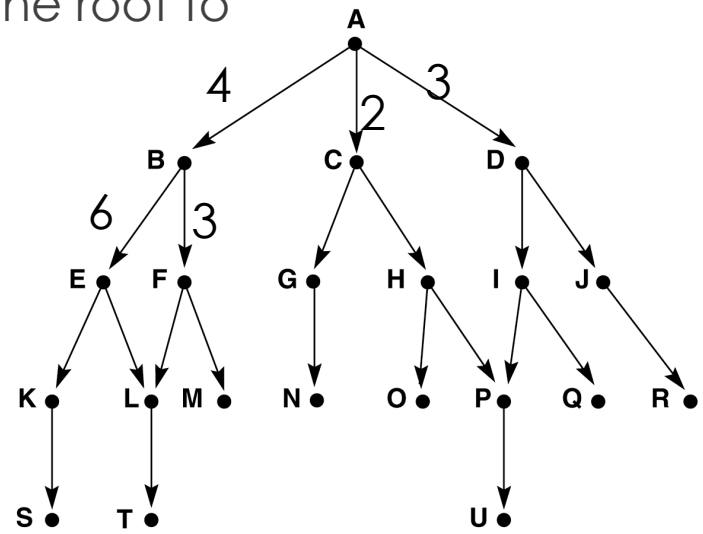


Types of State-Space Search

- **State-Space Search**
  - ***Uninformed Search***
    - ***Breadth-first and Depth-first***
    - ***Depth-limit Search***
    - ***Iterative Deepening***
    - ***Uniform Cost***
  - Informed Search
    - Hill Climbing
    - Best – Frist
    - Design Heuristics
    - A*

Uniform Cost Search

- Breadth First Search
  - uses a priority queue sorted using the depth of the nodes from the root
  - guarantees to find the shortest solution path
- But what if all edges/moves do not have the same cost?
- Uniform Cost Search
  - uses a priority queue sorted using the cost from the root to node n – later called g(n)
  - guarantees to find the lowest cost solution path



The End