Artificial Intelligence – 1(CS249)

2nd week notes by :-

2201AI07 – ANSH PHUTELA

2201AI08 – AKSHITHA REDDY BOREDDY

2201AI09 – DEVANSH JINDAL

2201AI10 – DIVYANSHU GUPTA

2201AI11 – UTTEJ DUNGA

”Basic search problem”

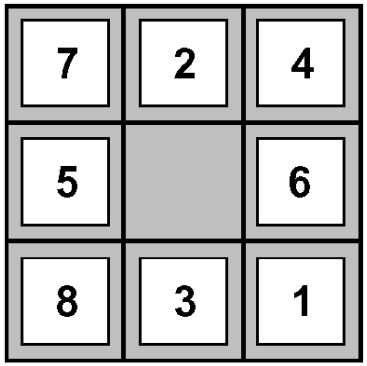
- Given {s,S,O,G} where:
- S is the implicitly specified set of states.
- s is the start state.
- O is the transition operators.
- G is the set of goal states
- Path cost (C) - e.g. sum of distances, number of operators executed...(Optional)
- Solution (path) – a sequence of operators leading from the initial state to a goal state.

One way to solve this is to search for a path

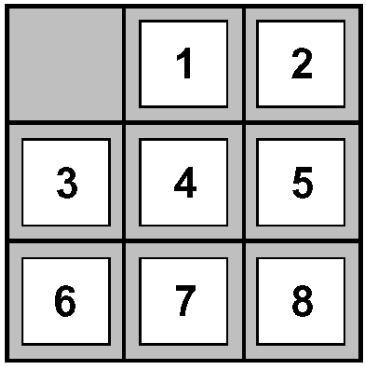
$$\theta(0) \rightarrow \theta(1) \rightarrow \theta(2) \rightarrow ... \rightarrow \theta(N)$$

such that $\theta(N)$ is a goal state.

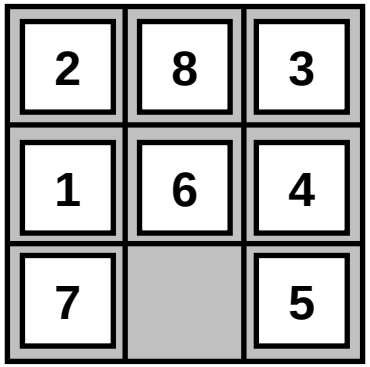
Example: 8-puzzle



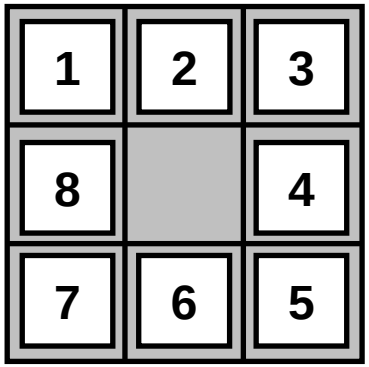
Start State



Goal State



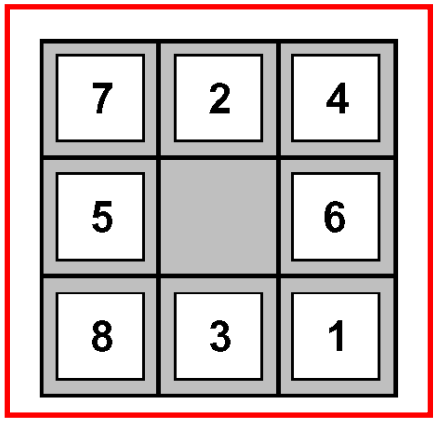
Start State



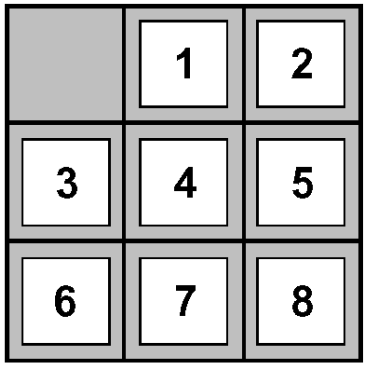
Goal State

- **State:** Specification of each of the eight tiles in the nine squares (the blank is in the remaining square).
- **Initial state:** Any state.
- **Successor function (actions):** Blank moves *Left, Right, Up*, or *Down.*
- **Goal test:** Check whether the goal state has been reached.
- **Path cost:** Each move costs 1. The path cost = the number of moves.

Example: 8-puzzle



Start State



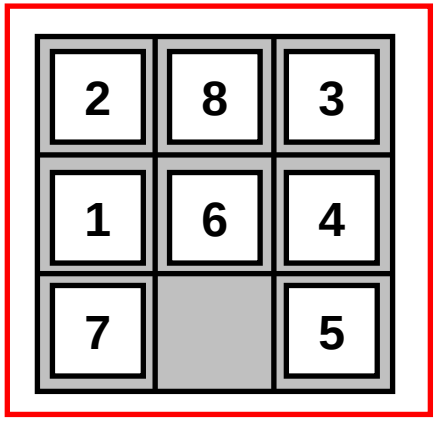
Goal State

- **State:** Specification of each of the eight tiles in the nine squares (the blank is in the remaining square).

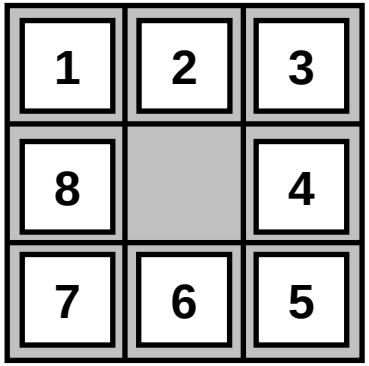
Examples:

$\theta = \{7, 2, 4, 5, 0, 6, 8, 3, 1\}$

$\theta = \{2, 8, 3, 1, 6, 4, 7, 0, 5\}$

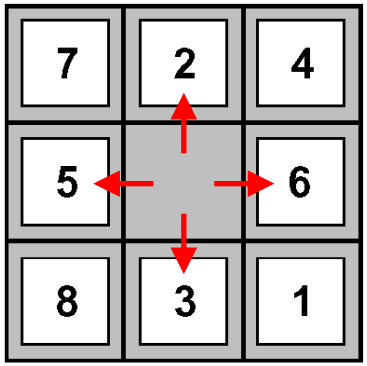


Start State

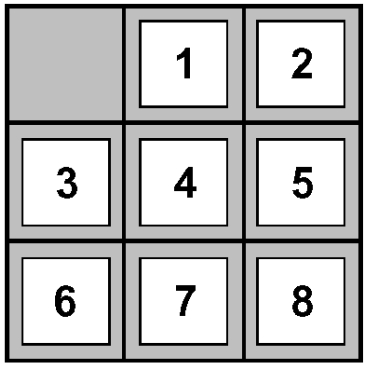


Goal State

Example: 8-puzzle

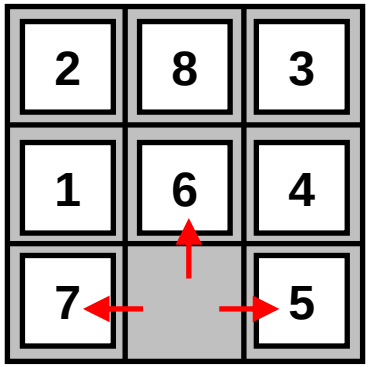


Start State

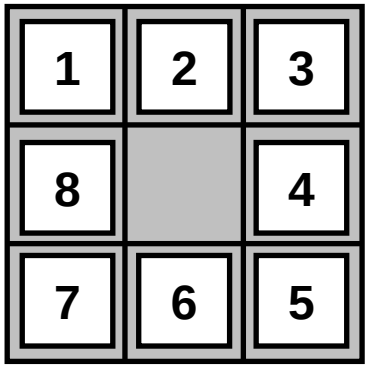


Goal State

- **Successor function (actions):** Blank moves *Left, Right, Up*, or *Down.*

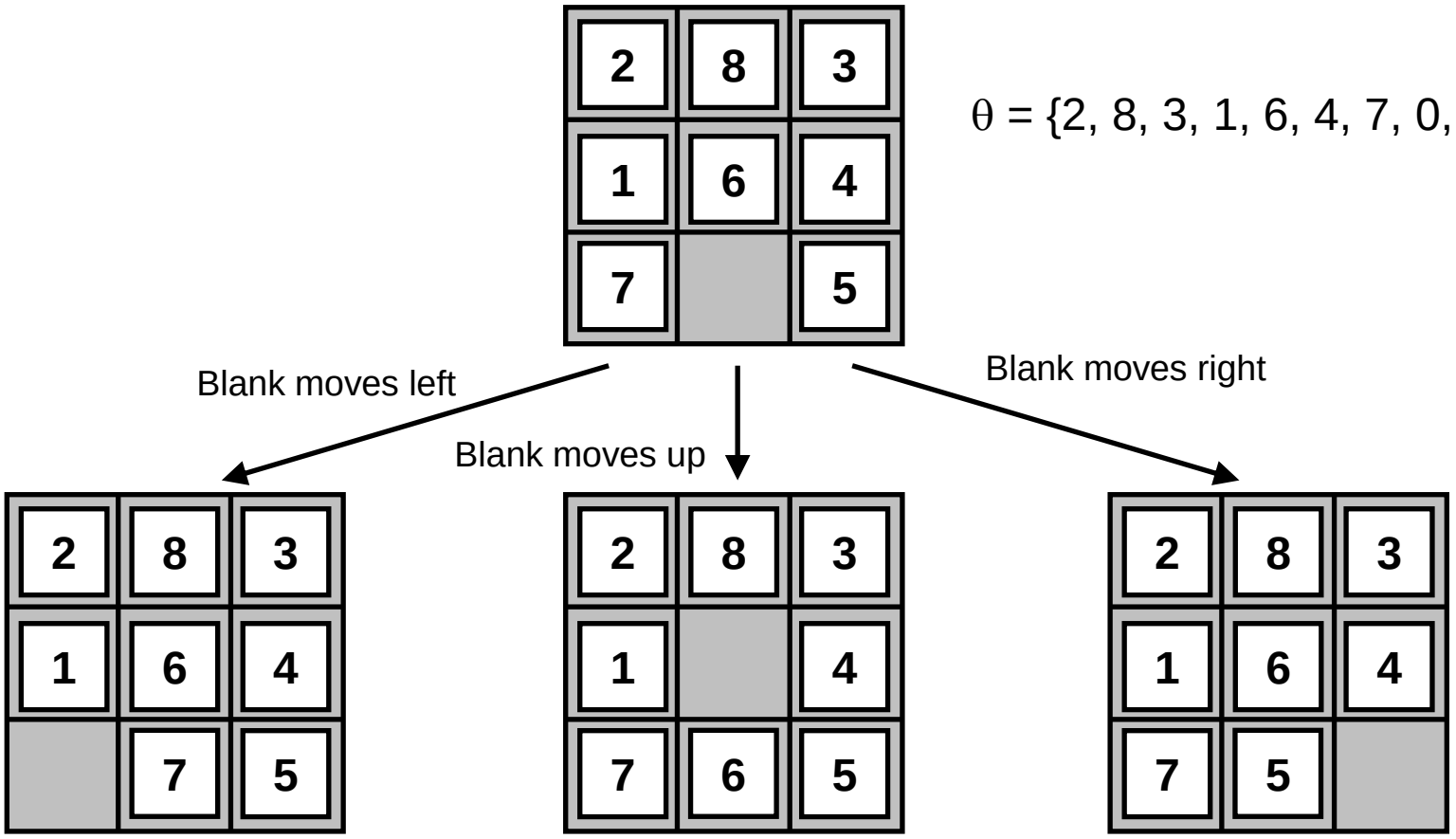


Start State



Goal State

Expanding 8-puzzle



Uninformed search

Searching for the goal without knowing in which direction it is.

- Breadth-first search
- Depth-first search

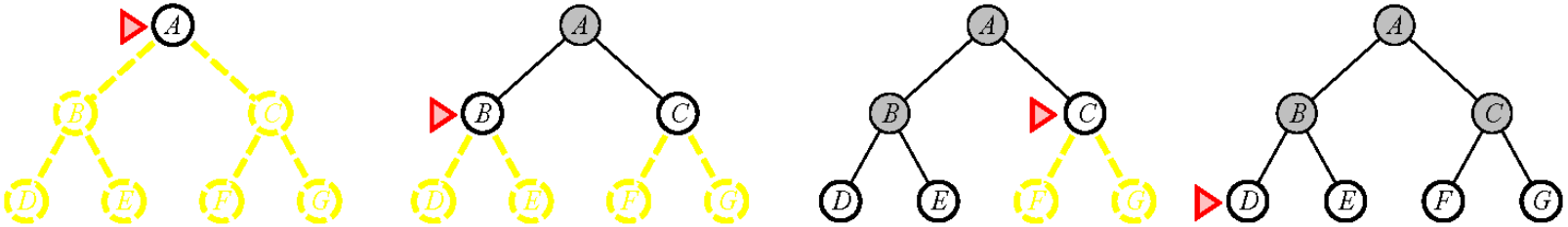
(Depth and breadth refers to the search tree)

We evaluate the algorithms by their:

- Completeness (do they explore all possibilities)
- Optimality (do they find the solution with minimum path cost)
- Time complexity (number of nodes expanded during search)
- Space complexity (maximum number of nodes in memory)

Breadth-first

Image from Russel & Norvig, AIMA, 2003



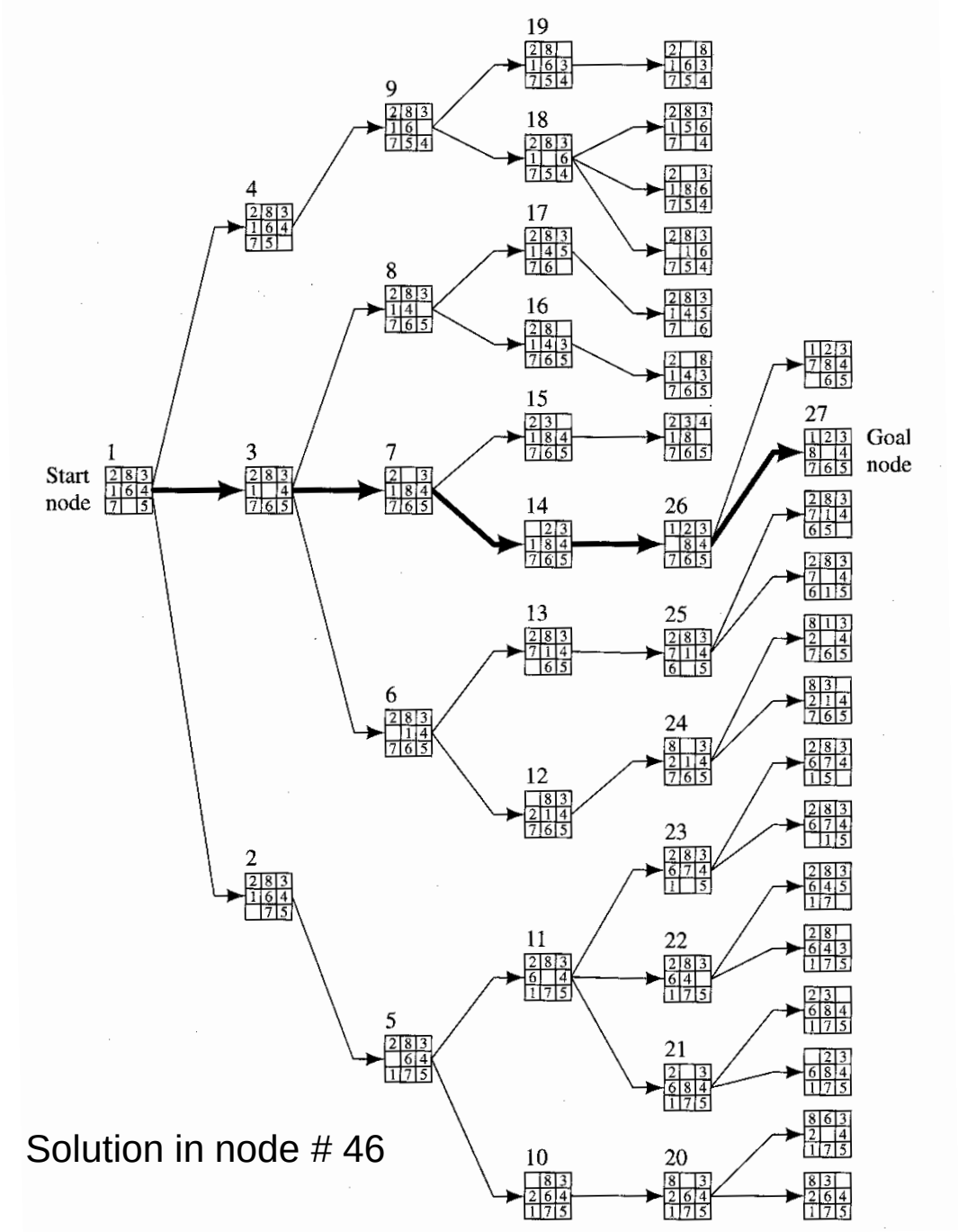
Nodes marked with open circles = fringe = in the memory

- Breadth-first finds the solution that is closest (in the graph) to the start node (always expands the shallowest node).
- Keeps $O(b^d)$ nodes in memory $\rightarrow$ exponential memory requirement!

- Complete (finds a solution if there is one)
- Not necessarily optimal (optimal if cost is the same for each step)
- Exponential space complexity (very bad)
- Exponential time complexity

b = branching factor, d = depth

Image from N. J. Nilsson, *Artificial Intelligence – A New Synthesis,* 1998



Solution in node # 46

Breadth-first search for 8-puzzle.

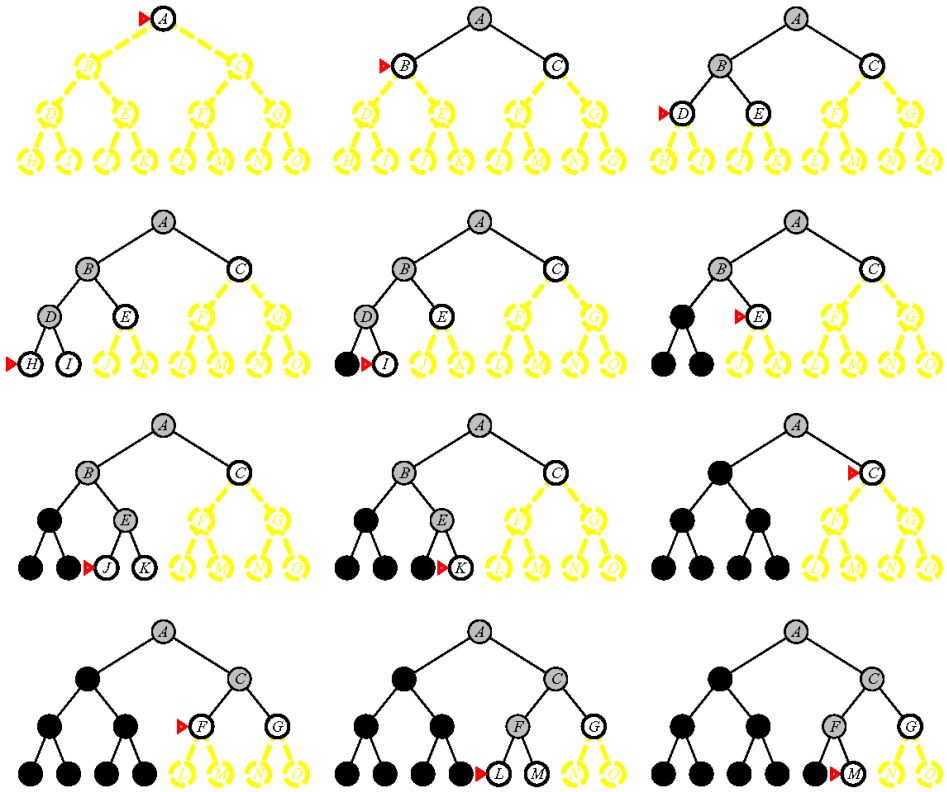
The path marked by bold arrows is the solution.

Note: This assumes that you apply goal test immediately after expansion
(not the case for AIMA implementation)

If we keep track of visited states → *Graph search*
(rather than *tree search*)

Depth-first

Image from Russel & Norvig, AIMA, 2003



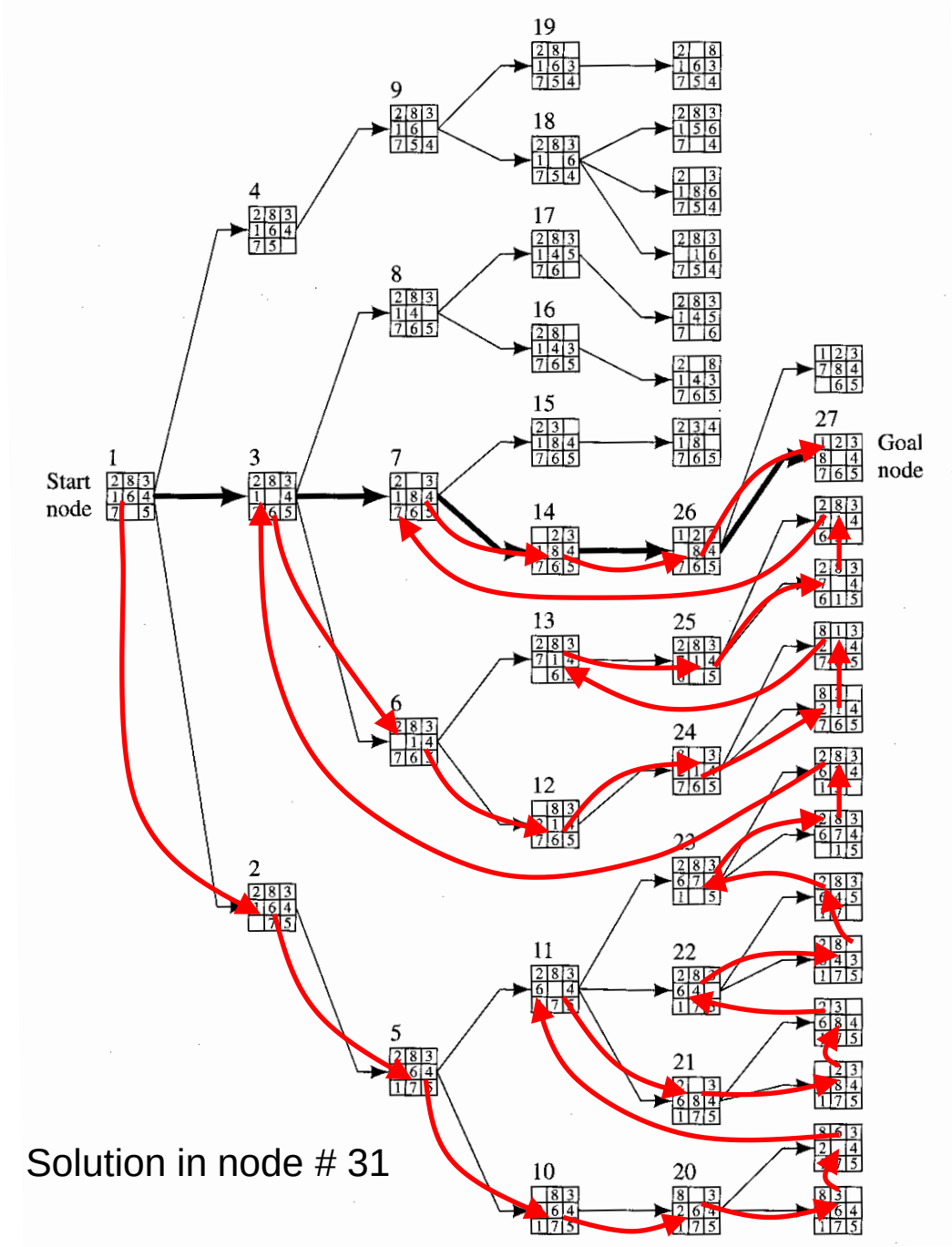
Black nodes are removed from memory

b = branching factor, d = depth

- Keeps $O(bd)$ nodes in memory.
- Requires a depth limit to avoid infinite paths
  (limit is 3 in the figure).

- Incomplete (is not guaranteed to find a solution)
- Not optimal
- Linear space complexity (good)
- Exponential time complexity

Image from N. J. Nilsson, *Artificial Intelligence – A New Synthesis,* 1998



Depth-first on the 8-puzzle example.

Depth = 5

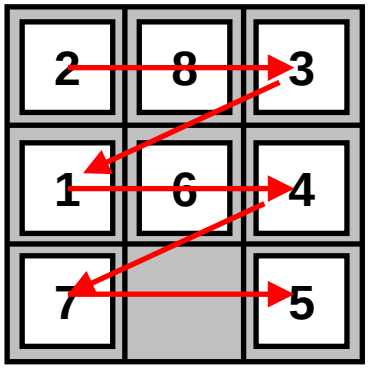
Solution in node # 31

8-puzzle Problem

Show that the 8-puzzle states are divided into two disjoint sets, such that no state in one set can be transformed into a state in the other set by any number of moves. Devise a procedure that will tell you which class a given state is in, and explain why this is a good thing to have for generating random states.

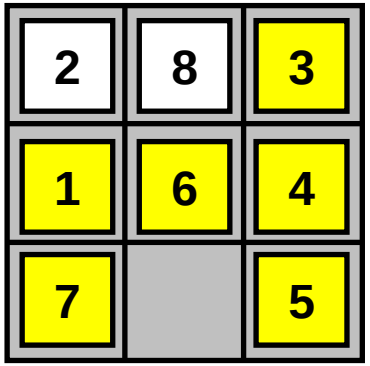
Proof for this problem:

Definition: Define the order of counting from the upper left corner to the lower right corner (see figure).

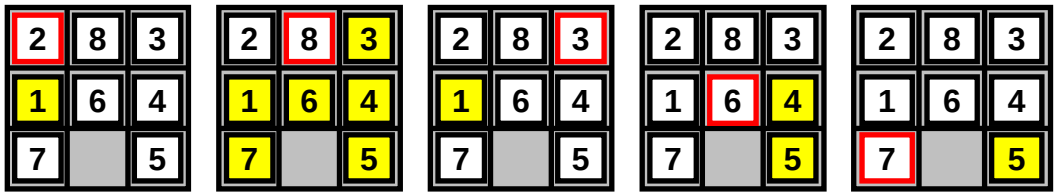


Let N denote the number of lower numbers following a number (so-called ”inversions”) when counting in this fashion.

$N = 11$ in the figure.



Yellow tiles are inverted relative to the tile with ”8” in the top row.



Proof:

Proposition: N is either always even or odd (i.e. $N \bmod 2$ is conserved).

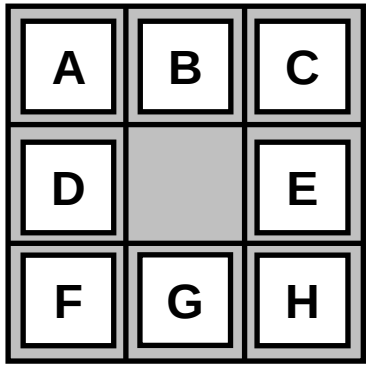
Proof:

(1) Sliding the blank along a row does not change the row number and not the internal order of the tiles, i.e. N (and thus also $N \bmod 2$) is conserved.

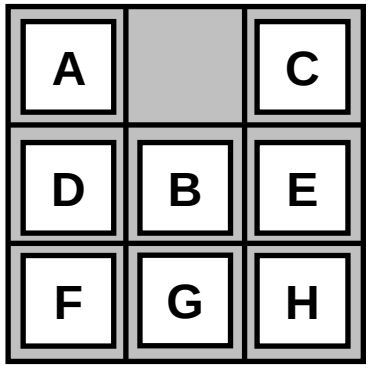
(2) Sliding the blank between rows does not change $N \bmod 2$ either, as shown on the following slide.

Proof:

We only need to consider tiles B, C, and D since the relative order of the other tiles remains the same.



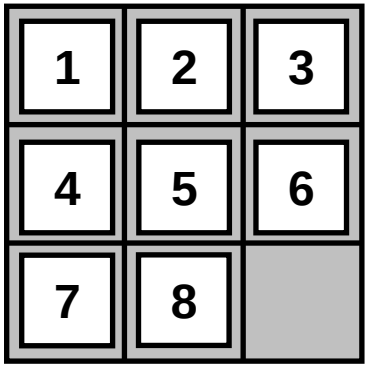
- If B > C and B > D, then the move removes two inversions.
- If B > C and B < D, then the move adds one inversion and removes one (sum = 0).
- If B < C and B < D, then the move adds two inversions.



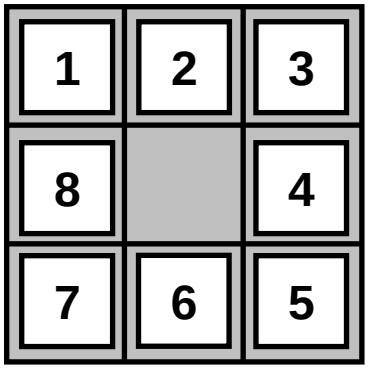
The number of inversions changes in steps of 2.

Observation

The upper state has $N = 0$



The lower (goal) state has $N = 7$



We cannot go from one to the other.

Missionaries & Cannibals problem

Exercise 3.9: The **missionaries and cannibals**: Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people (one for rowing). Find a way to get everyone to the other side, without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place (the cannibals eat the missionaries then).

a. Formulate the problem precisely, making only those distinctions necessary to ensure a valid solution. Draw a diagram of the complete state space.

b. Implement and solve the problem optimally using an appropriate search algorithm. Is it a good idea to check for repeated states?

c. Why do you think people have a hard time solving this puzzle, given that the state space is so simple?

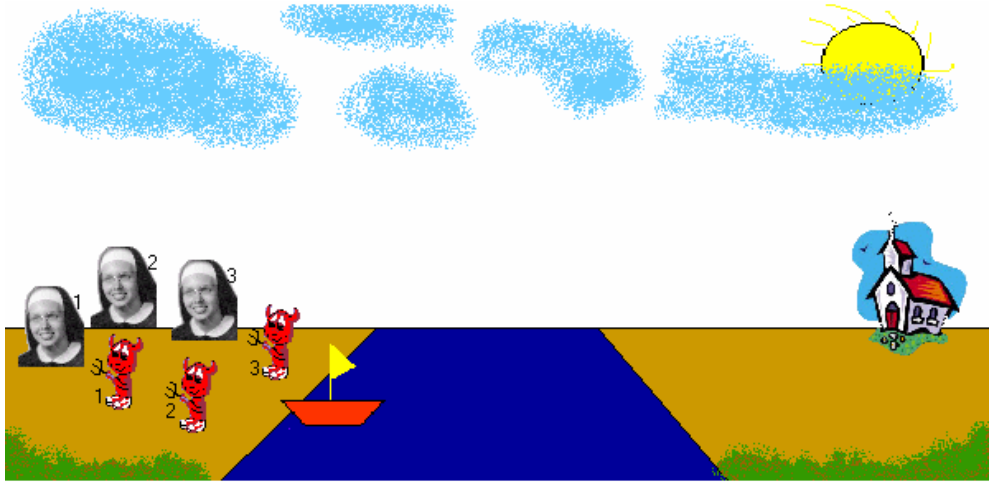


Image from http://www.cse.msu.edu/~michmer3/440/Lab1/cannibal.html

Missionaries & Cannibals

State: θ = (M,C,B) signifying the number of missionaries, cannibals, and boats on the left bank. The start state is (3,3,1) and the goal state is (0,0,0).

Actions (successor function): (10 possible but only 5 available each move due to boat)

- One cannibal/missionary crossing L $\rightarrow$ R: subtract (0,1,1) or (1,0,1)
- Two cannibals/missionaries crossing L $\rightarrow$ R: subtract (0,2,1) or (2,0,1)
- One cannibal/missionary crossing R $\rightarrow$ L: add (1,0,1) or (0,1,1)
- Two cannibals/missionaries crossing R $\rightarrow$ L: add (2,0,1) or (0,2,1)
- One cannibal and one missionary crossing: add/subtract (1,1,1)

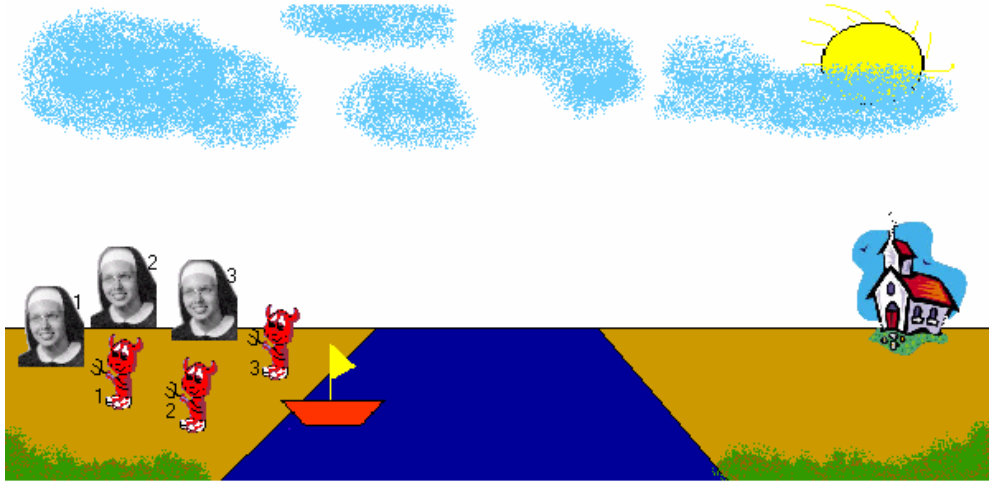
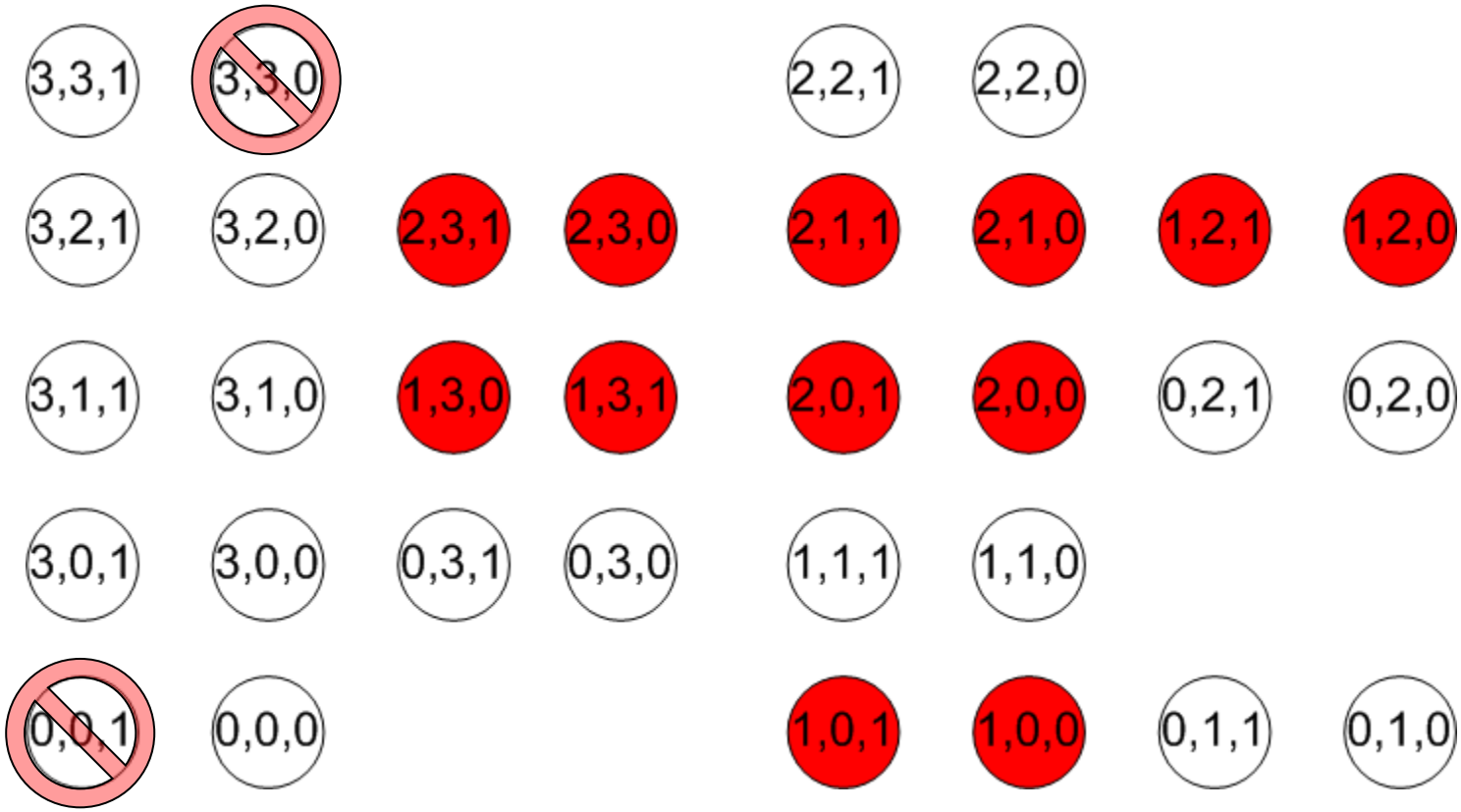


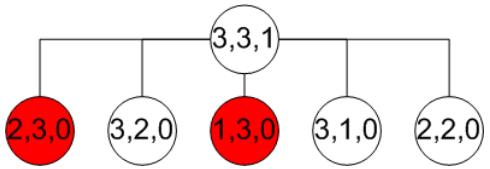
Image from http://www.cse.msu.edu/~michmer3/440/Lab1/cannibal.html

Missionaries & Cannibals states



Assumes that passengers have to get out of the boat after the trip.
Red states = missionaries get eaten.

Breadth-first search on Missionaries & Cannibals



States are generated by applying:

+/- (1,0,1)
+/- (0,1,1)
+/- (2,0,1)
+/- (0,2,1)
+/- (1,1,1)

In that order (left to right)

Red states = missionaries get eaten

Breadth-first search on Missionaries & Cannibals

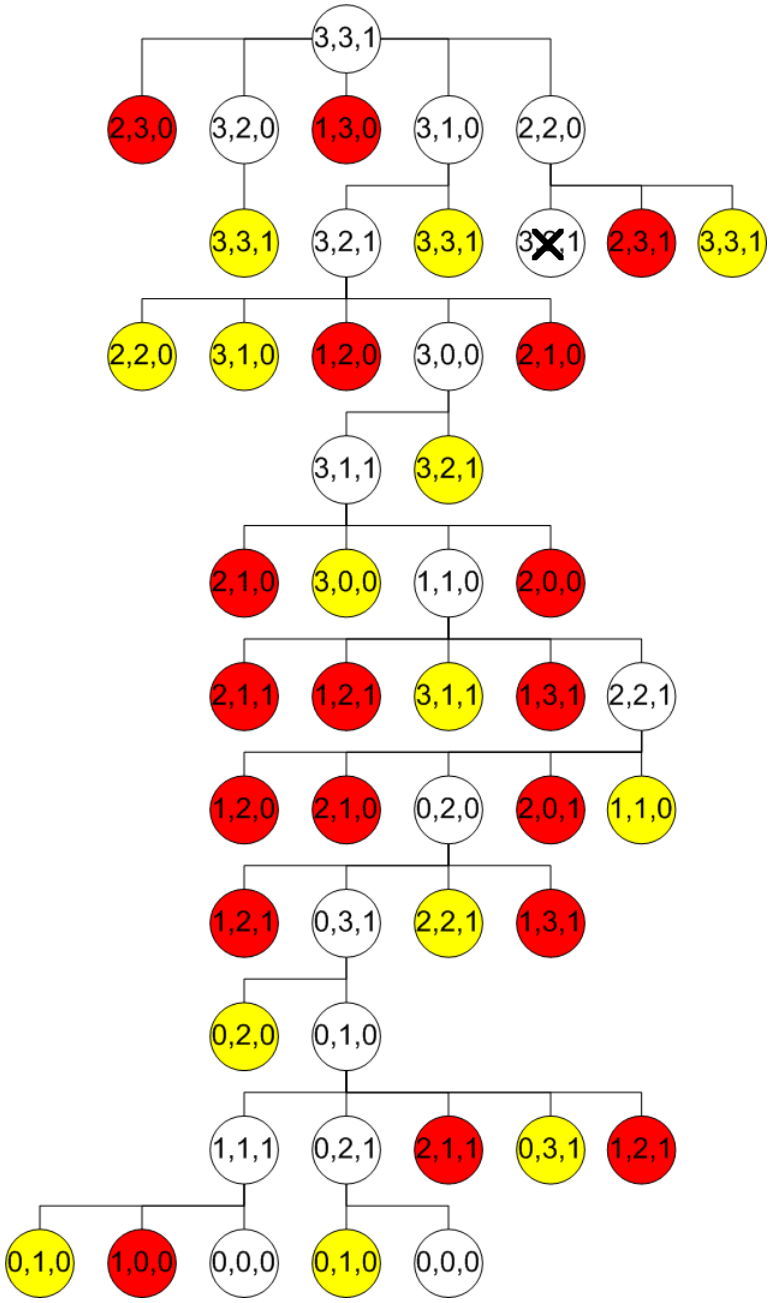
States are generated by applying:

+/- (1,0,1)
+/- (0,1,1)
+/- (2,0,1)
+/- (0,2,1)
+/- (1,1,1)

In that order (left to right)

Red states = missionaries get eaten

Yellow states = repeated states

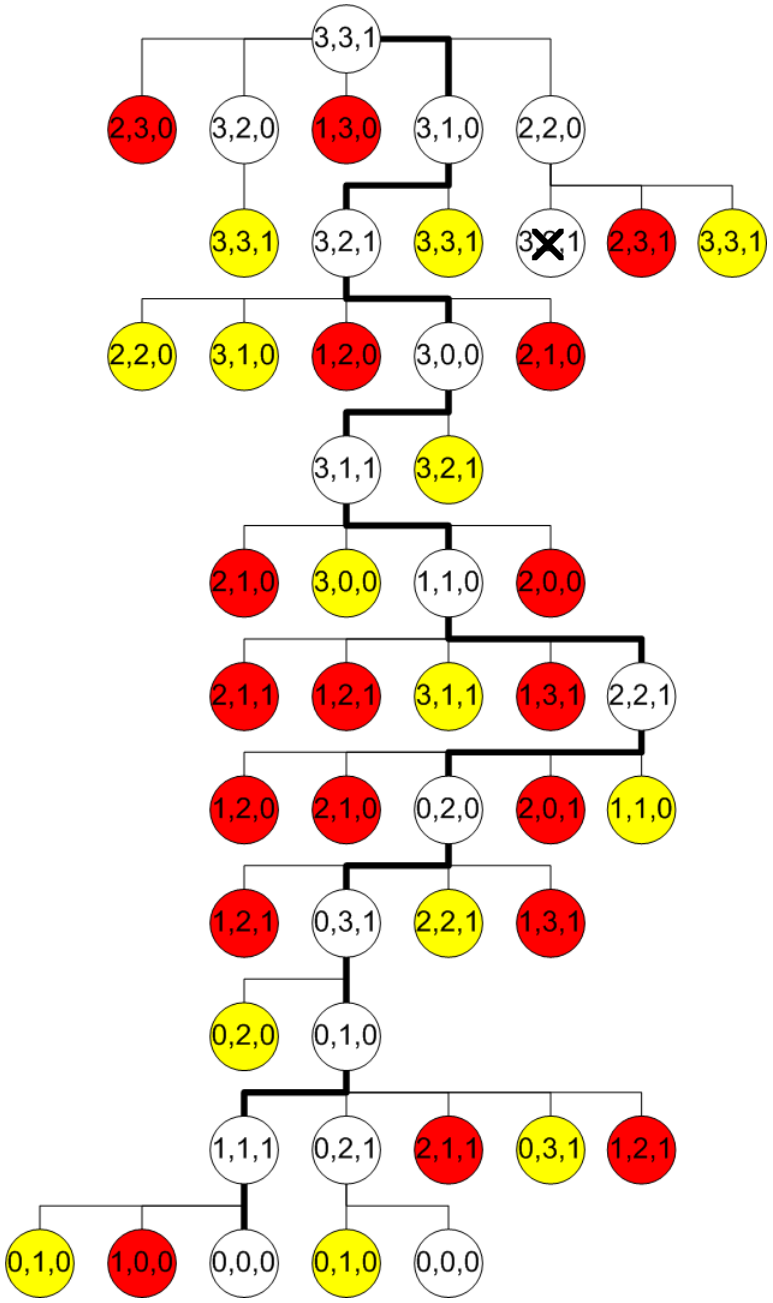


Breadth-first search on Missionaries & Cannibals

-(0,2,1) [2 cannibals cross L → R]
+(0,1,1) [1 cannibal crosses R → L]
-(0,2,1) [2 cannibals cross L → R]
+(0,1,1) [1 cannibal crosses R → L]
-(2,0,1) [2 missionaries cross L → R]
+(1,1,1) [1 cannibal & 1 missionary cross R → L]
-(2,0,1) [2 missionaries cross L → R]
+(0,1,1) [1 cannibal crosses R → L]
-(0,2,1) [2 cannibals cross L → R]
+(1,0,1) [1 missionary crosses R → L]
-(1,1,1) [1 cannibal & 1 missionary cross L → R]

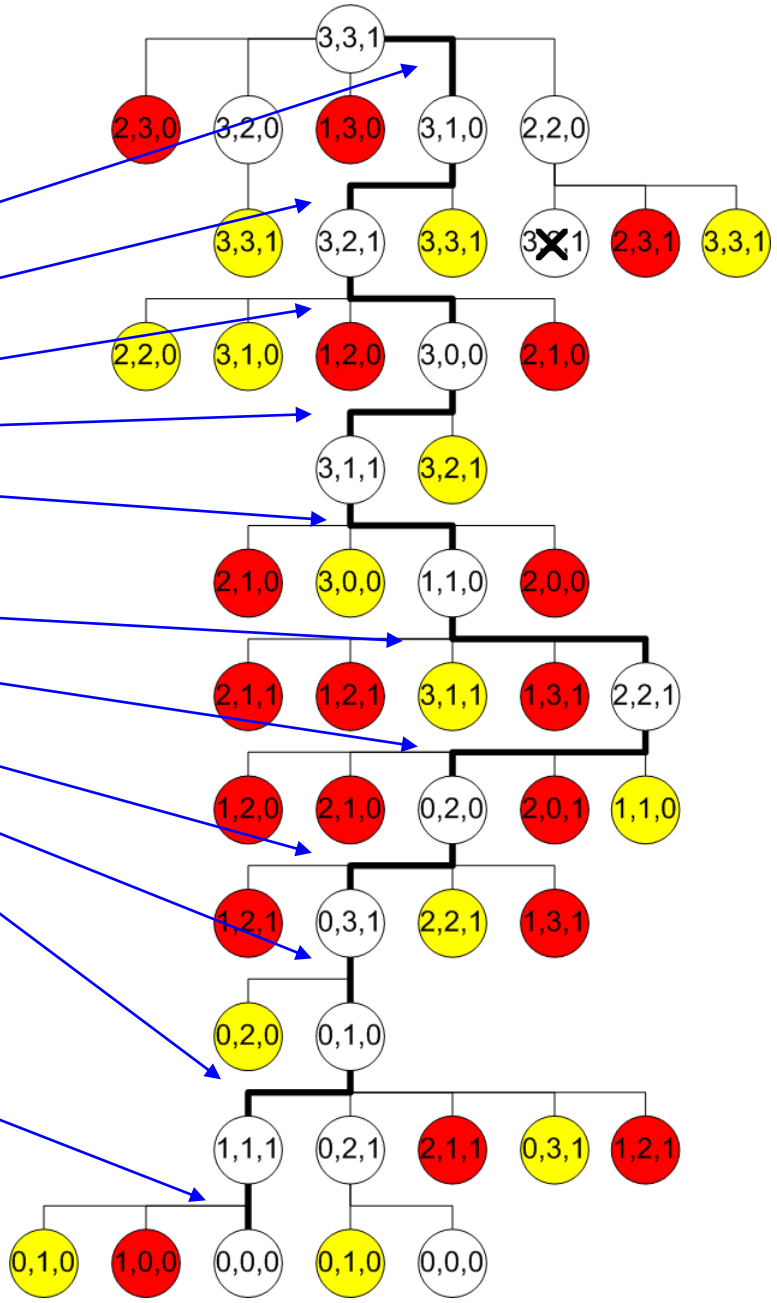
This is an optimal solution (minimum number of crossings). [Why?]

Would Depth-first work?



Breadth-first search on Missionaries & Cannibals

-(0,2,1) [2 cannibals cross L → R]
+(0,1,1) [1 cannibal crosses R → L]
-(0,2,1) [2 cannibals cross L → R]
+(0,1,1) [1 cannibal crosses R → L]
-(2,0,1) [2 missionaries cross L → R]
+(1,1,1) [1 cannibal & 1 missionary cross R → L]
-(2,0,1) [2 missionaries cross L → R]
+(0,1,1) [1 cannibal crosses R → L]
-(0,2,1) [2 cannibals cross L → R]
+(1,0,1) [1 missionary crosses R → L]
-(1,1,1) [1 cannibal & 1 missionary cross L → R]



This is an optimal solution (minimum number of crossings).

Would Depth-first work?

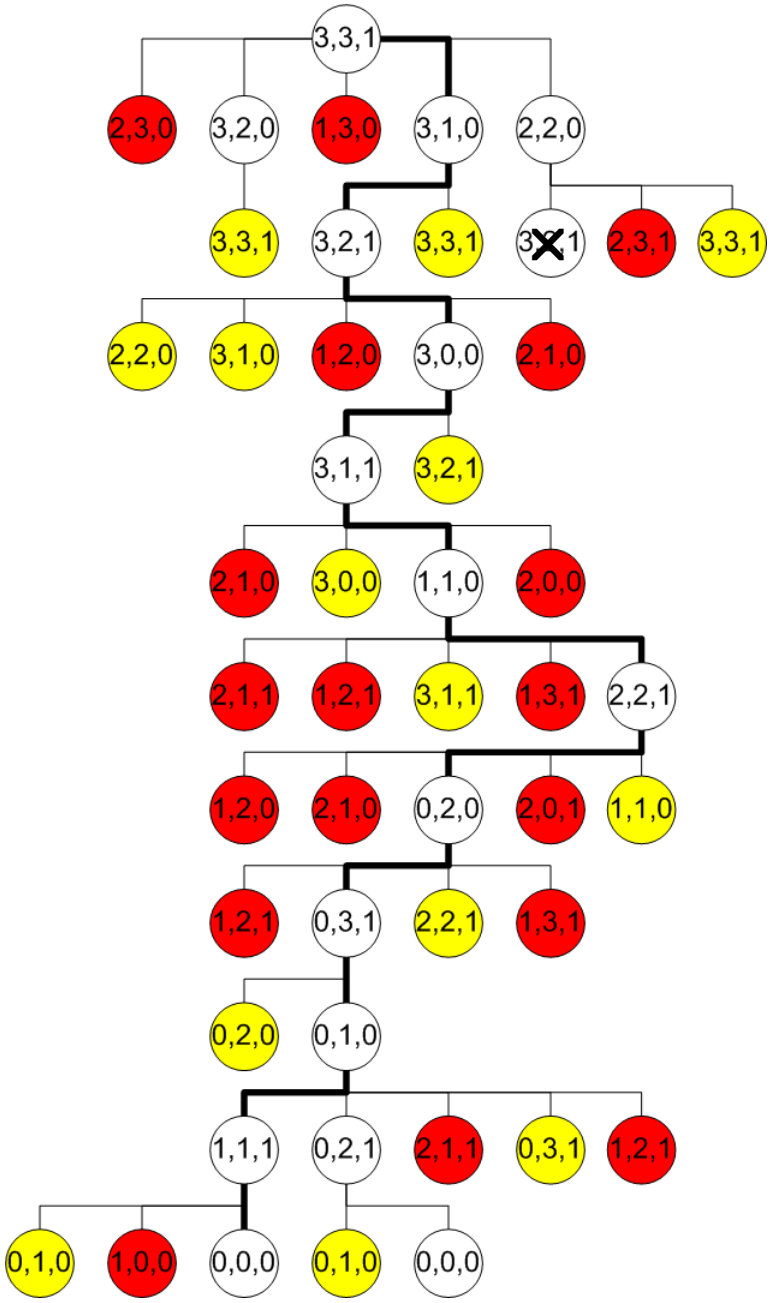
Breadth-first search on Missionaries & Cannibals

Expanded 48 nodes

Depth-first search on Missionaries & Cannibals

Expanded 30 nodes

(if repeated states are checked, otherwise we end up in an endless loop)



Example : Water Jug Problem

- **Problem statement**:
  - Given two jugs, a 4-gallon and 3-gallon having no measuring markers on them. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into 4-gallon jug.
- **Solution**:
  - State for this problem can be described as the set of ordered pairs of integers (X, Y) such that
    - X represents the number of gallons of water in 4-gallon jug and
    - Y for 3-gallon jug.
  - Start state is (0,0)
  - Goal state is (2, N) for any value of N.

Production Rules

- Following are the production rules for this problem.

R1: $(X, Y \mid X < 4) \rightarrow (4, Y)$
{Fill 4-gallon jug}

R2: $(X, Y \mid Y < 3) \rightarrow (X, 3)$
{Fill 3-gallon jug}

R3: $(X, Y \mid X > 0) \rightarrow (0, Y)$
{Empty 4-gallon jug}

R4: $(X, Y \mid Y > 0) \rightarrow (X, 0)$
{Empty 3-gallon jug}

R5: $(X, Y \mid X+Y >= 4 \Lambda Y > 0) \rightarrow (4, Y – (4 – X))$
{Pour water from 3- gallon jug into 4-gallon jug until 4-gallon jug is full}

Trace of steps involved in solving the water jug problem - First solution

Number of Steps	*Rules applied*	*4-g jug*	*3-g jug*	
1	**Initial State**	0	0	
2	R2 {Fill 3-g jug}	0	3	
3	R7{Pour all water from 3 to 4-g jug }	3	0	
4	R2 {Fill 3-g jug}	3	3	
5	R5 {Pour from 3 to 4-g jug until it is full}	4	2	
6	R3 {Empty 4-gallon jug}	0	2	
7	R7 {Pour all water from 3 to 4-g jug}	2	0	**Goal State**

Trace of steps involved in solving the water jug problem - Second solution

- Note that there may be more than one solutions.

Number of steps	*Rules applied*	*4-g jug*	*3-g jug*	
1	**Initial State**	0	0	
2	R1 {Fill 4-gallon jug}	4	0	
3	R6 {Pour from 4 to 3-g jug until it is full }	1	3	
4	R4 {Empty 3-gallon jug}	1	0	
5	R8 {Pour all water from 4 to 3-gallon jug}	0	1	
6	R1 {Fill 4-gallon jug}	4	1	
7	R6 {Pour from 4 to 3-g jug until it is full}	2	3	
8	R4 {Empty 3-gallon jug}	2	0	**Goal State**

Important Points

- For each problem
  - there is an initial description of the problem.
  - final description of the problem.
  - more than one ways of solving the problem.
  - a path between various solution paths based on some criteria of goodness or on some heuristic function is chosen.
  - there are set of rules that describe the actions called production rules.
    - Left side of the rules is current state and right side describes new state that results from applying the rule.

OTHER POPULAR PROBLEMS

- 8 QUEEN PROBLEM

Generate an arrangement of 8 queens on a chess board such that no queen is under attack by another queen.

- Travelling Salesman problem (TSP)

Objective is to find possible route with minimum cost that visits a set of cities exactly once and returns to the original city. Here cost can be distance between the cities or time taken to travel.

SEARCH ALGORITHM

- Initialize: set OPEN = {s}
- Fail: If OPEN = { }. Terminate with failure
- Select: Select a state from OPEN.
- Teminate: If n belongs to G ,Terminate with success
- Generate the successor or neighbour of ‘n’ using O(Operator) and insert them in OPEN
- Load: Repeat above steps from step 2

SEARCH ALGO FOR STATE SPACE SEARCH

Here we maintain 2 lists , OPEN , CLOSED for searching.

- Initialization: Insert ‘s’ to OPEN , CLOSED = [ ]
- Termination with Success :

  Remove node ‘n’ from OPEN if n belongs to G , then terminate with success and add ‘n’ in CLOSED
- Selection : Apply all O (Operations) on ‘n’ to generate it’s neighbours(m) , for all m , check if m belongs to OPEN union CLOSED , if not then add ‘m’ in OPEN
- Termination with Failure :

  if OPEN is empty , terminate with failure ,

  Otherwise go for next iteration

- ❖ The characteristics of the Algo depends on what data structure we are using for OPEN
- BFS – Queue , DFS – Stack .
- ❖ A Graph ‘G’ is created . However , the G is on IMPLICIT Graph because all the vertices and edges are not known beforehand.
- ❖ We begin with start state , keep on building ‘G’ till We reach goal state.

(b = branching , d = depth)

- Time complexity of BFS/DFS = O(b^d)
- Space complexity of DFS = O(b*d)

Here d is the maximum depth of the graph for the start state

- Space complexity of BFS = O(b^d)

Here d is min level at which goal state can be found

- NOTE: DFS is incomplete whereas BFS is optimal and complete

CITATIONS

- UNIVERSITY DO PORTO
- RUSSEL AND NORVIG BOOK

CONTRIBUTION

2201AI07 – ANSH PHUTELA - 20%
2201AI08 – AKSHITHA REDDY BOREDDY - 20%
2201AI09 – DEVANSH JINDAL - 20%
2201AI10 – DIVYANSHU GUPTA - 20%
2201AI11 – UTTEJ DUNGA - 20%