Artificial Intelligence DT8012

Uninformed search

Chapter 3, AIMA

A ”problem” consists of

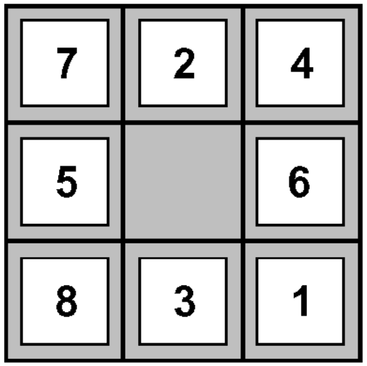
- An initial state, $\theta(0)$
- A list of possible actions, α, for the agent
- A goal test (there can be many goal states)
- A path cost

One way to solve this is to search for a path

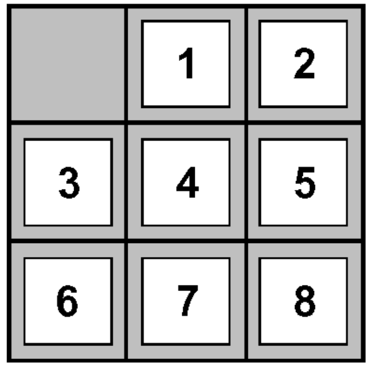
$$\theta(0) \rightarrow \theta(1) \rightarrow \theta(2) \rightarrow \ldots \rightarrow \theta(N)$$

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

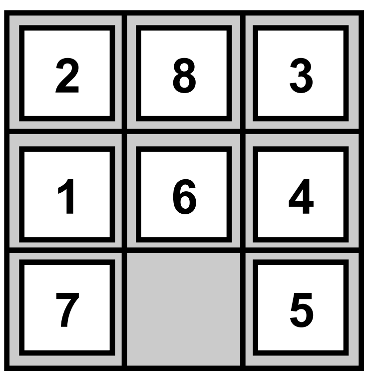
Example: 8-puzzle



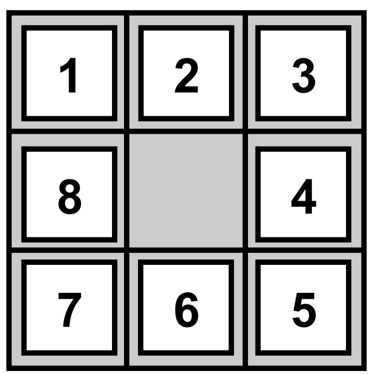
Start State



Goal State



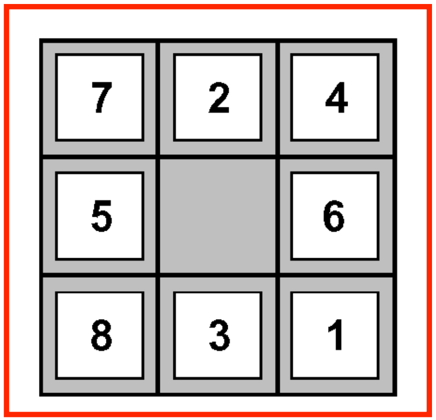
Start State



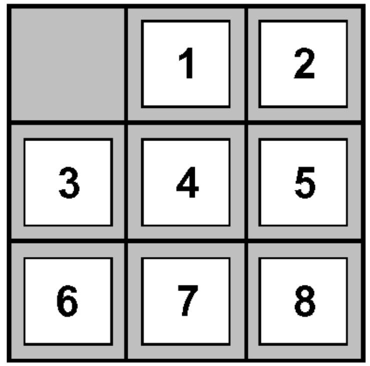
Goal State

- **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.

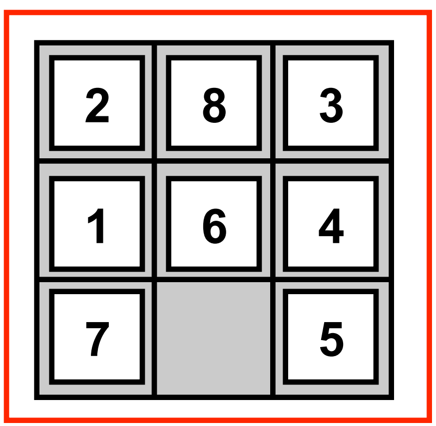
Example: 8-puzzle



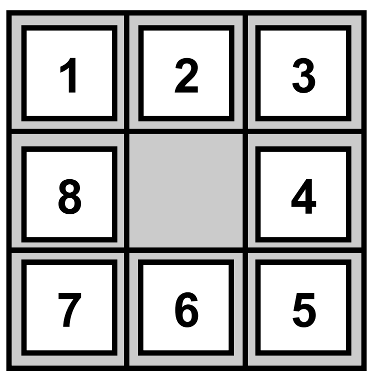
Start State



Goal State



Start State



Goal State

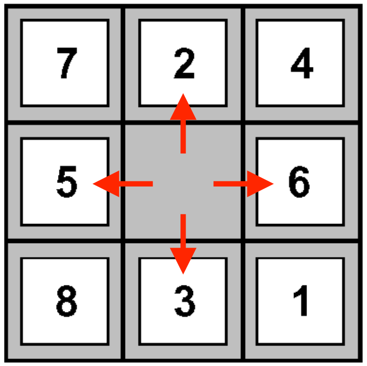
- **State:** Specification of each of the eight tiles in the nine squares (the blank is in the remaining square).

Examples:

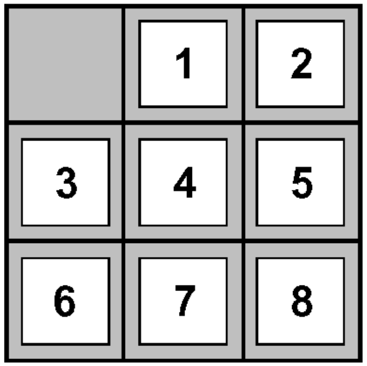
$\theta = \{7, 2, 4, 5, 0, 6, 8, 3, 1\}$

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

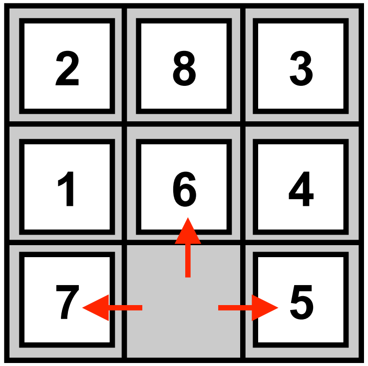
Example: 8-puzzle



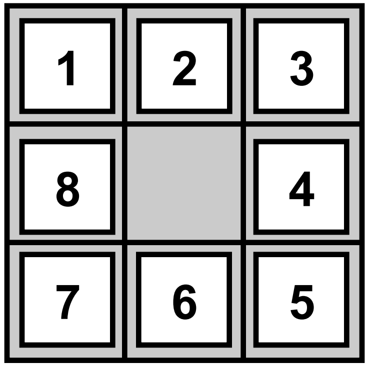
Start State



Goal State



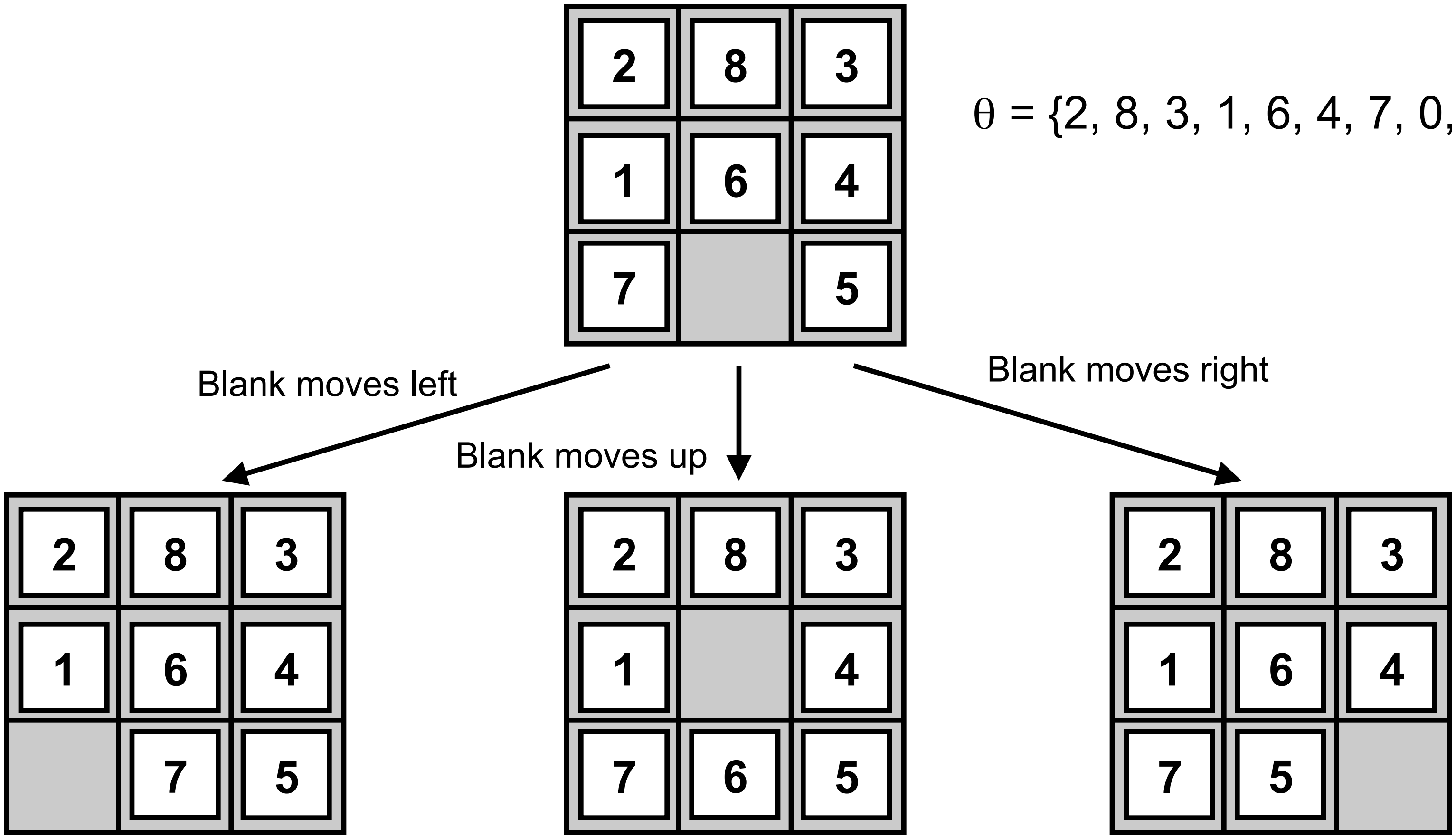
Start State



Goal State

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

Expanding 8-puzzle



θ = {2, 8, 3, 1, 6, 4, 7, 0, 5}

θ = {2, 8, 3, 1, 6, 4, 0, 7, 5}

θ = {2, 8, 3, 1, 6, 4, 7, 5, 0}

θ = {2, 8, 3, 1, 0, 4, 7, 6, 5}

Uninformed search

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

- Breadth-first
- Depth-first
- Iterative deepening

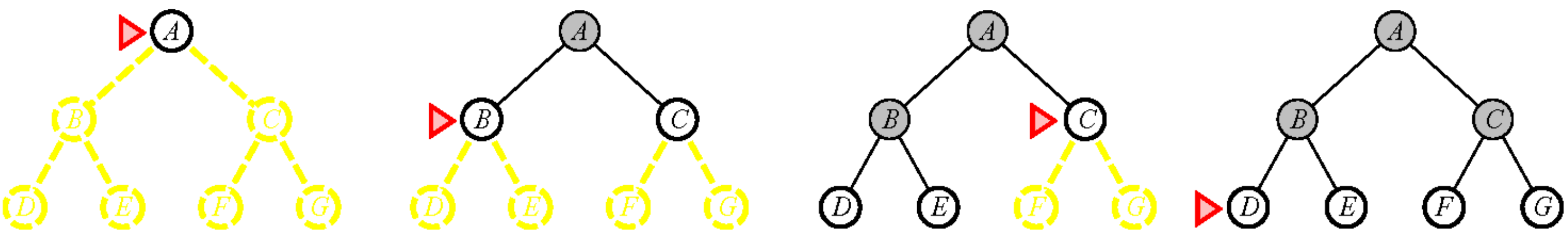
(Depth and breadth refers to the search tree)

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



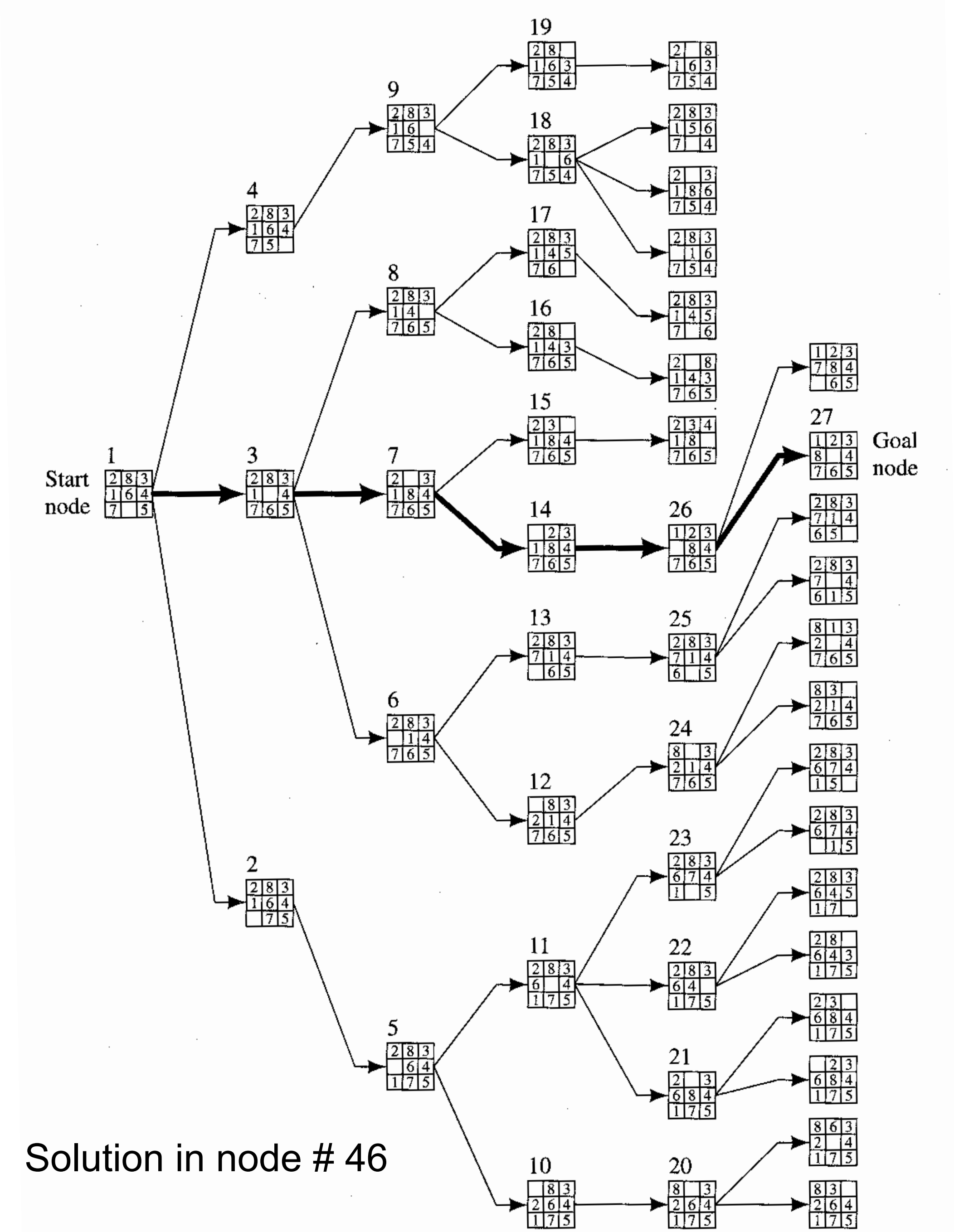
Nodes marked with open circles = fringe = in the memory

- 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 → exponential memory requirement!

- Complete (finds a solution if there is one)
- Not necessarily optimal (optimal if cost is the same for each step)
- <u>Exponential</u> 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.

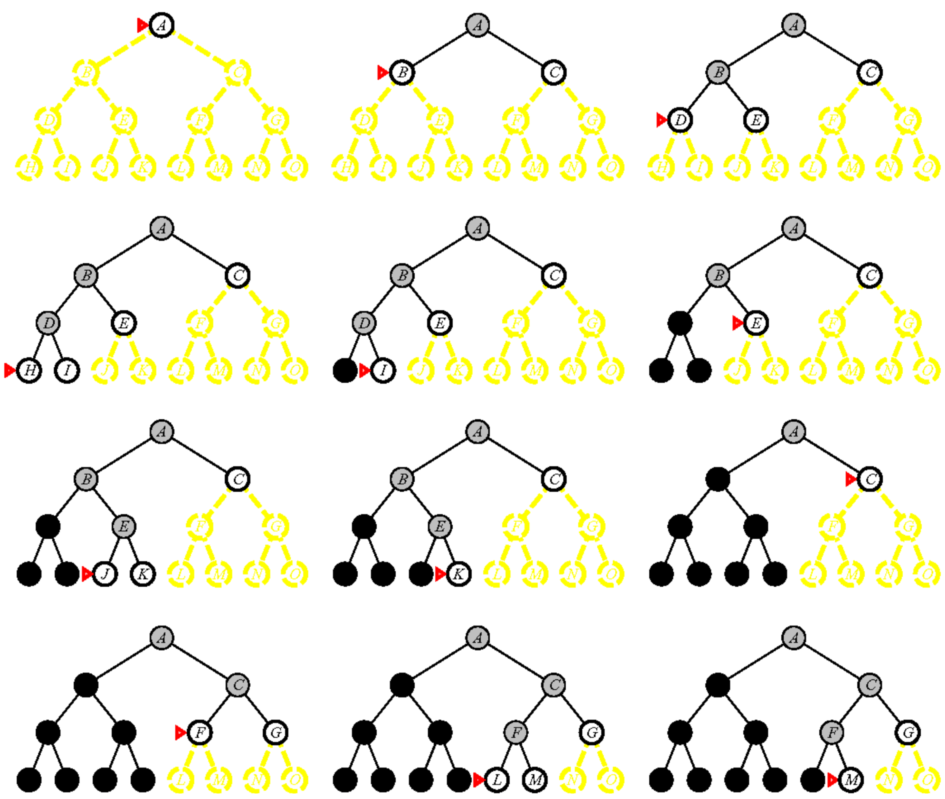
The path marked by bold arrows is the solution.

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



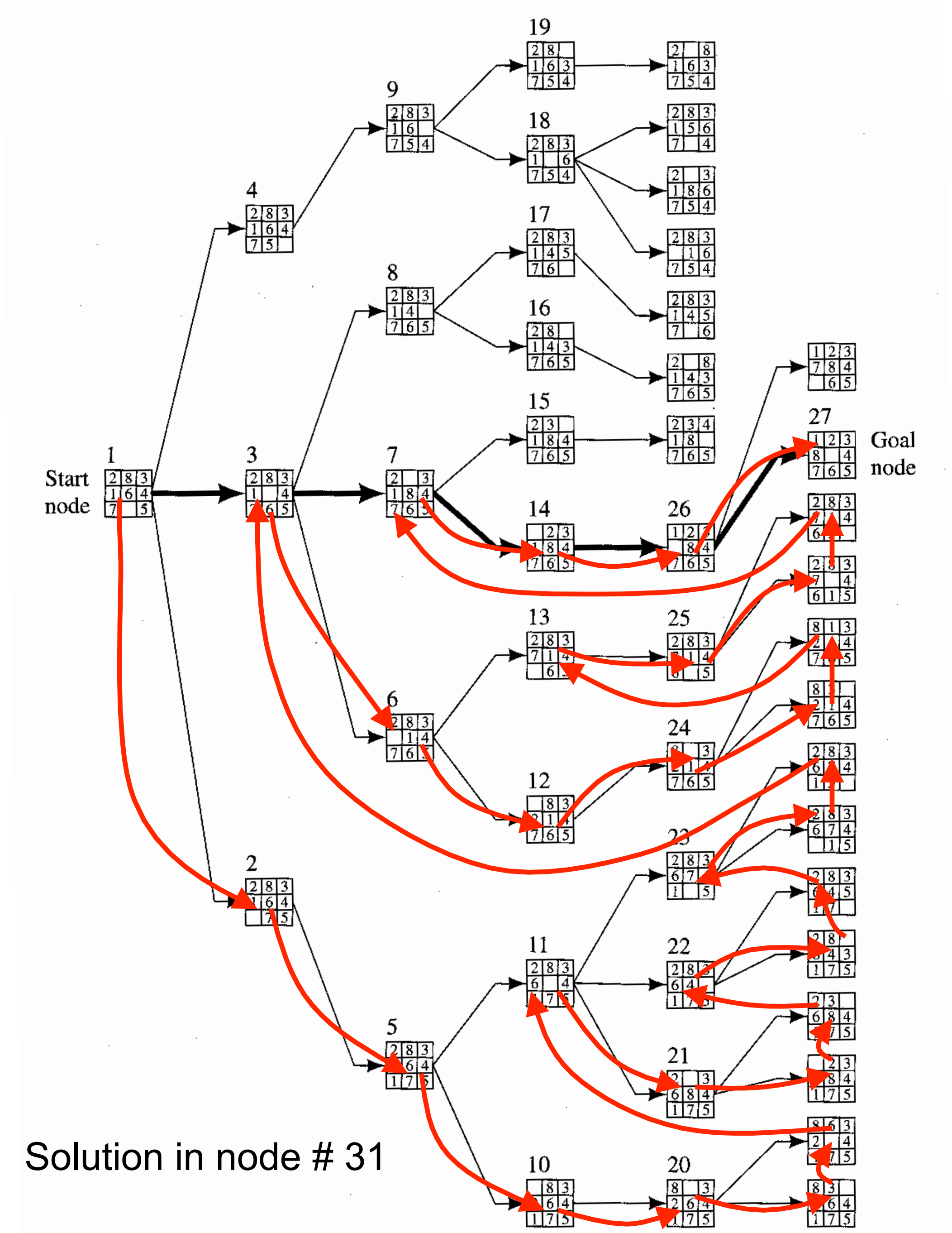
Black nodes are removed from memory

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



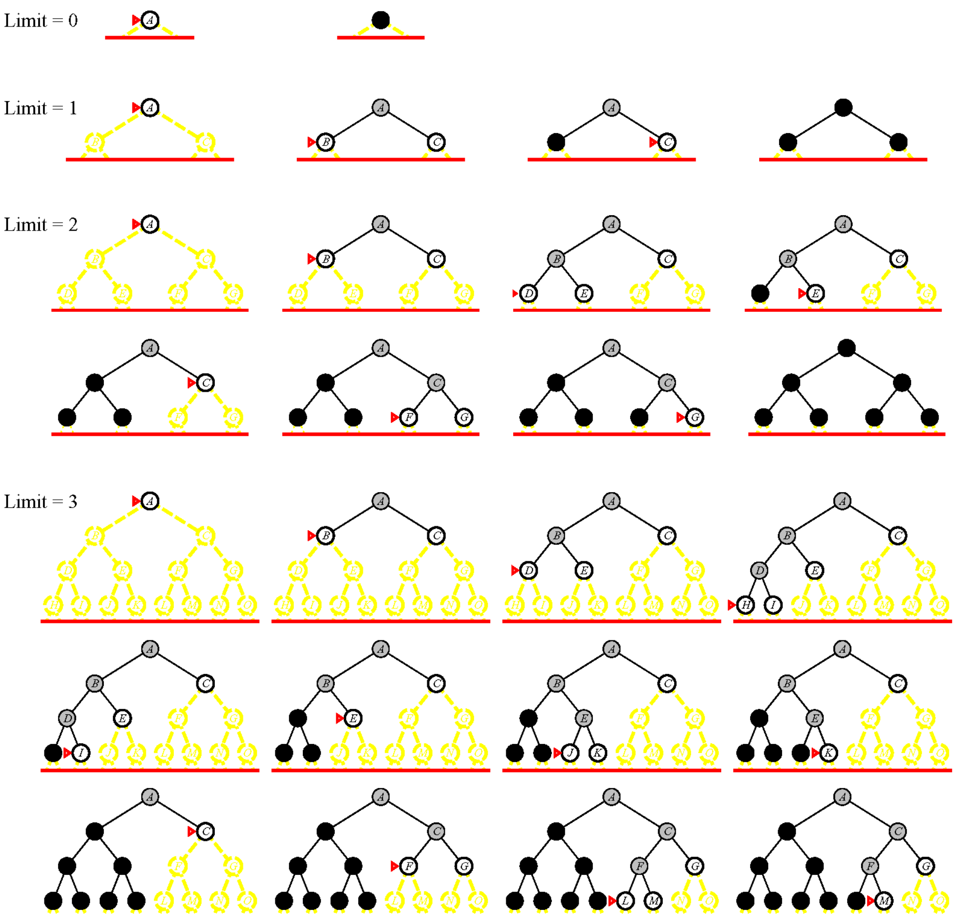
Solution in node # 31

Depth-first on the 8-puzzle example.

Depth = 5

Iterative deepening

Image from Russel & Norvig, AIMA, 2003



Black nodes are removed from memory

- Keeps $O(bd)$ nodes in memory.
- Iteratively increases the depth limit.

- Complete (like BFS)
- Optimal (if step costs are same)
- Linear space complexity (like DFS)
- Exponential time complexity

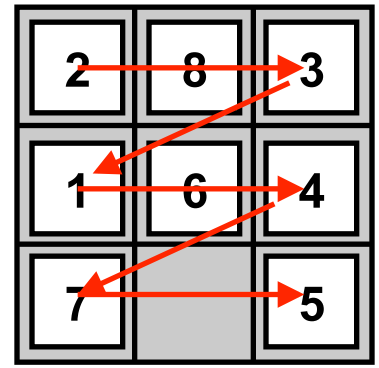
- The preferred search method for large search spaces with unknown depth.

b = branching factor, d = depth

Exercise

Exercise 3.4: 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.

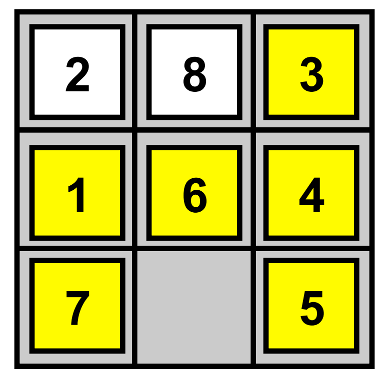
Proof for exercise 3.4:



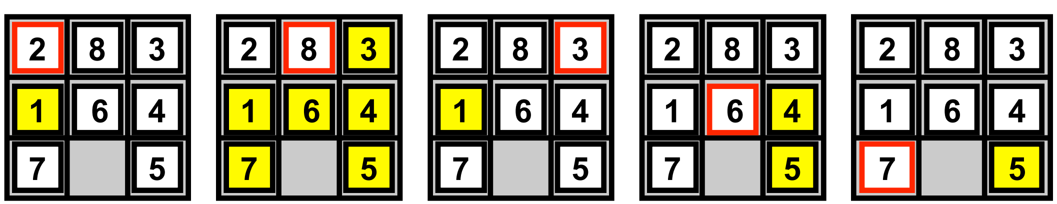
Definition: Define the order of counting from the upper left corner to the lower right corner (see figure).

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.



1 + 6 + 1 + 2 + 1 = 11

Proof for exercise 3.4:

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

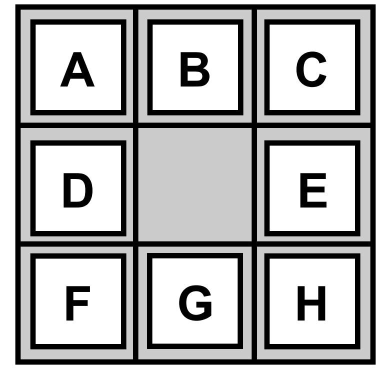
Proof:

(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.

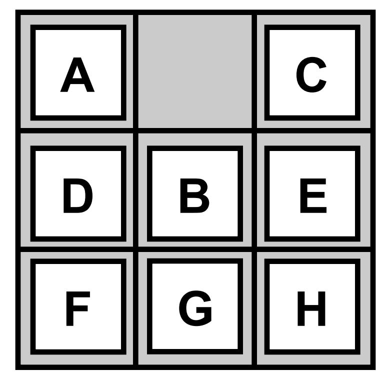
(2) Sliding the blank between rows does not change $N \bmod 2$ either, as shown on the following slide.

Proof for exercise 3.4:

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 $N = 0$

1	2	3
4	5	6
7	8	

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

1	2	3
8		4
7	6	5

We cannot go from one to the other.

Exercise

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?

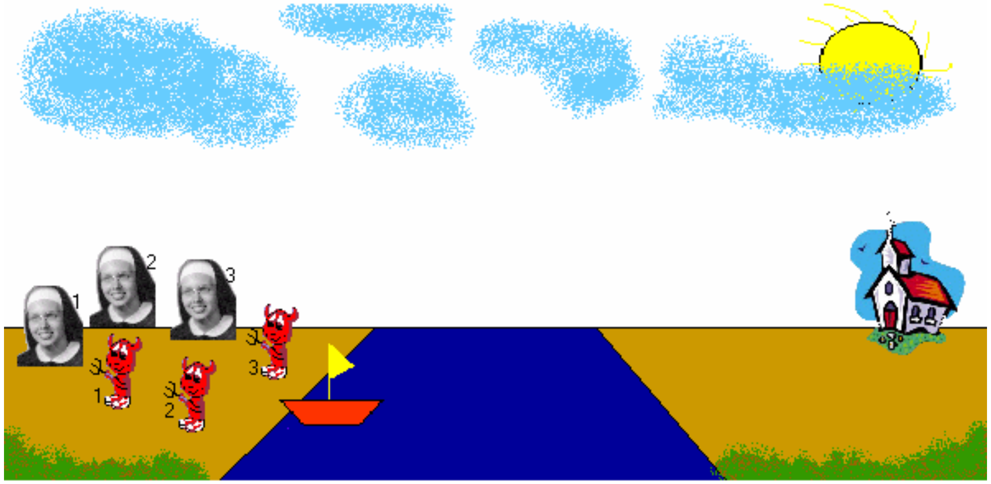


Image from http://www.cse.msu.edu/~michmer3/440/Lab1/cannibal.html

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

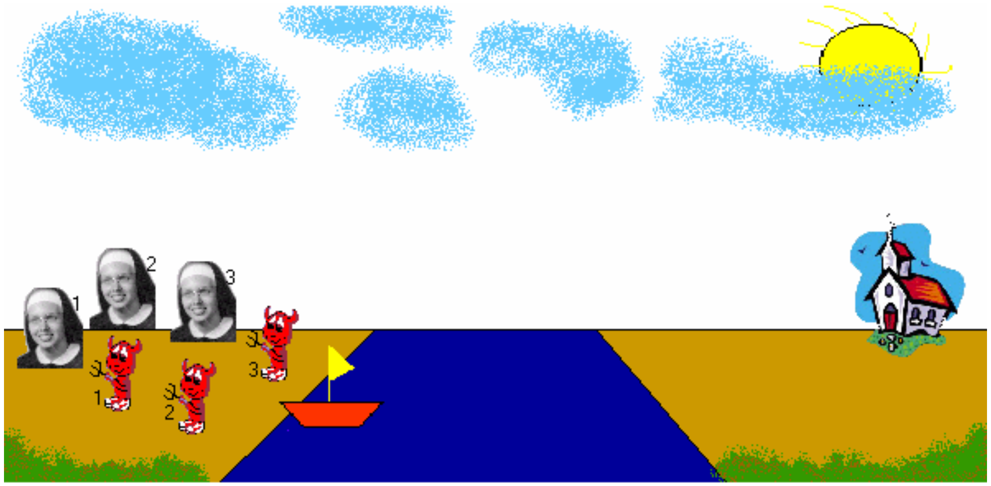
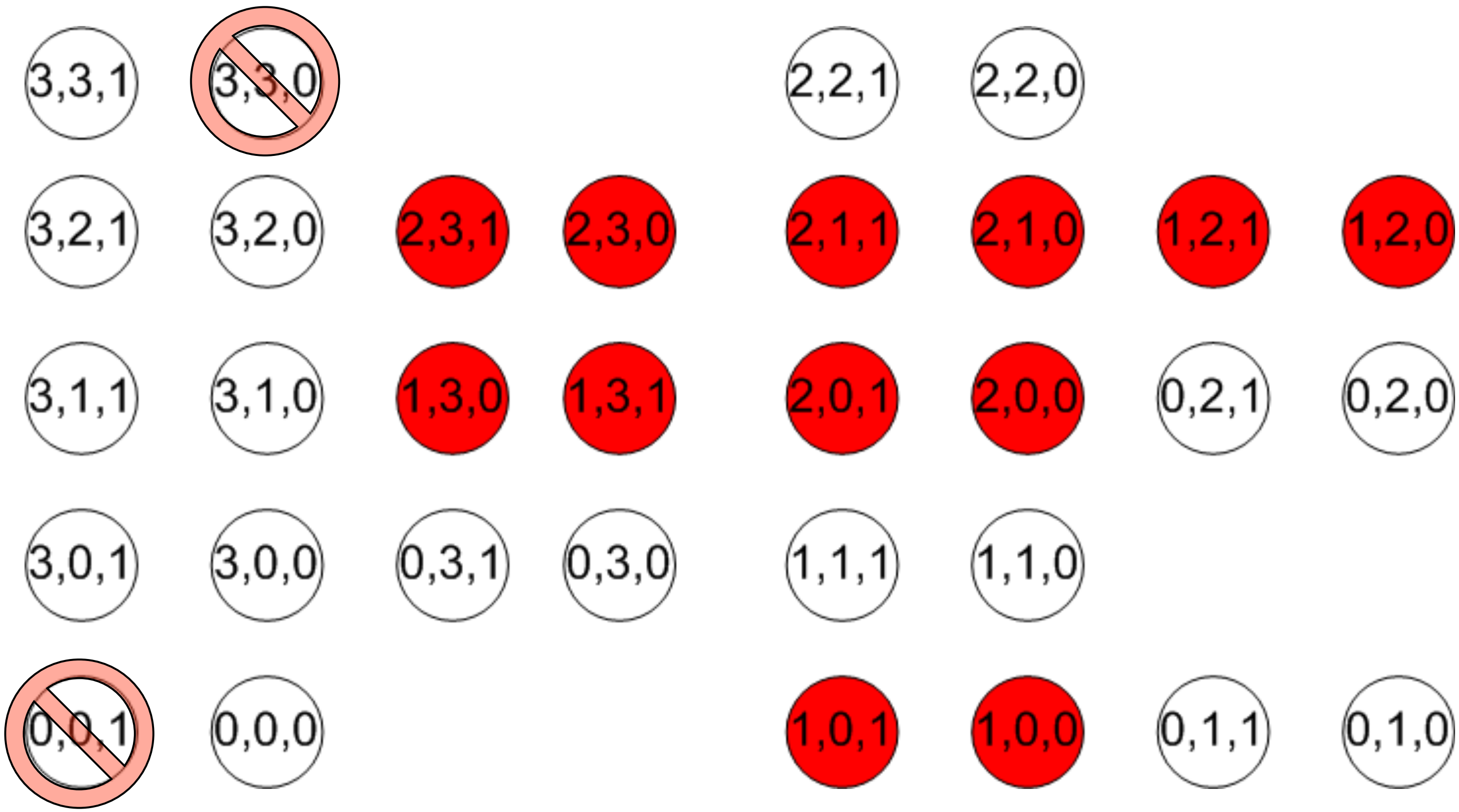


Image from http://www.cse.msu.edu/~michmer3/440/Lab1/cannibal.html

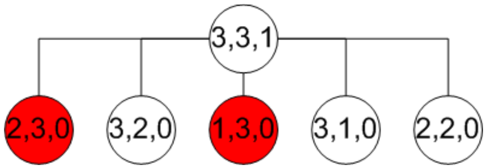
Missionaries & Cannibals states



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

Breadth-first search on Missionaries & Cannibals

Breadth-first search on Missionaries & Cannibals



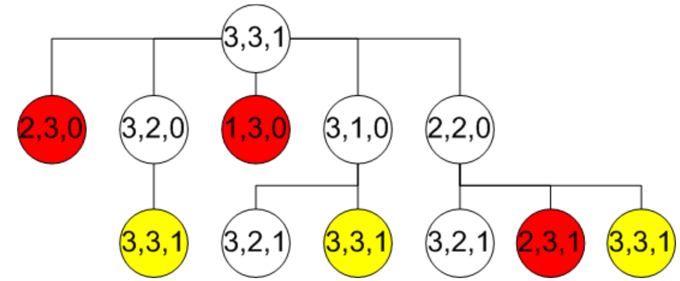
States are generated by applying:

+/- (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:

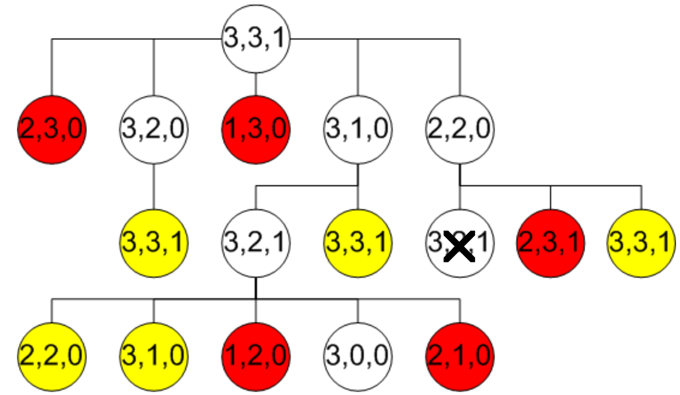
+/- (1,0,1)
+/- (0,1,1)
+/- (2,0,1)
+/- (0,2,1)
+/- (1,1,1)

In that order (left to right)

Red states = missionaries get eaten

Yellow states = repeated states

Breadth-first search on Missionaries & Cannibals



States are generated by applying:

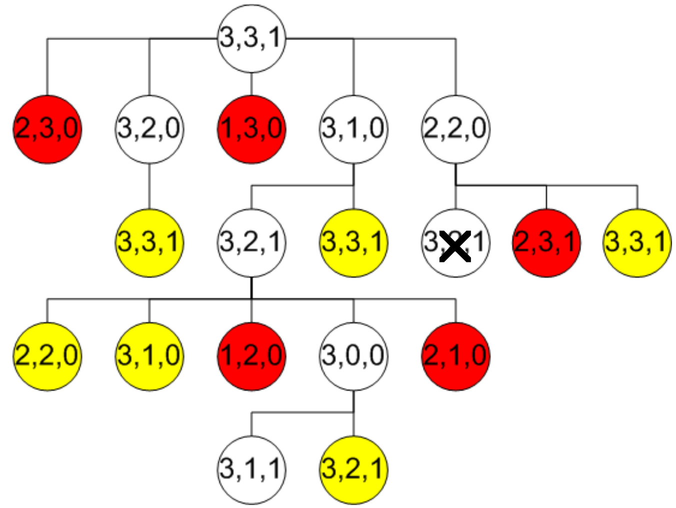
+/- (1,0,1)
+/- (0,1,1)
+/- (2,0,1)
+/- (0,2,1)
+/- (1,1,1)

In that order (left to right)

Red states = missionaries get eaten

Yellow states = repeated states

Breadth-first search on Missionaries & Cannibals



States are generated by applying:

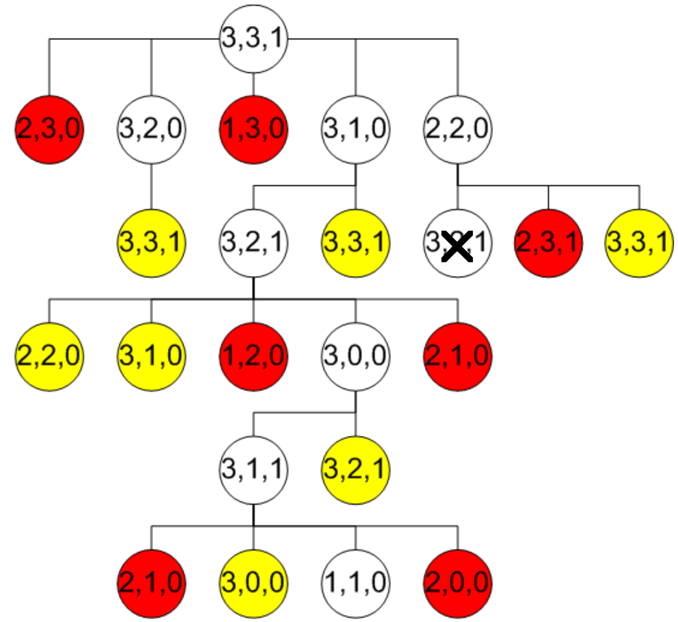
+/- (1,0,1)
+/- (0,1,1)
+/- (2,0,1)
+/- (0,2,1)
+/- (1,1,1)

In that order (left to right)

Red states = missionaries get eaten

Yellow states = repeated states

Breadth-first search on Missionaries & Cannibals



States are generated by applying:

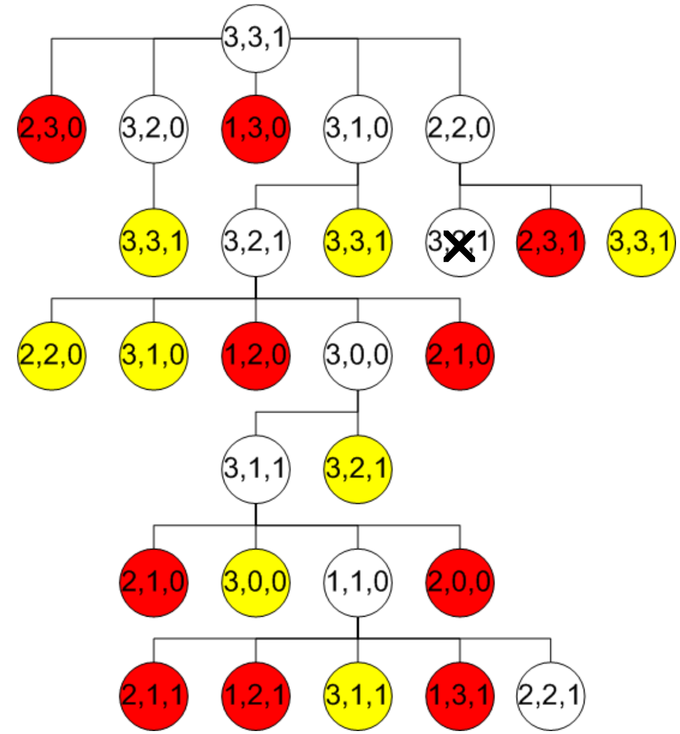
+/- (1,0,1)
+/- (0,1,1)
+/- (2,0,1)
+/- (0,2,1)
+/- (1,1,1)

In that order (left to right)

Red states = missionaries get eaten

Yellow states = repeated states

Breadth-first search on Missionaries & Cannibals



States are generated by applying:

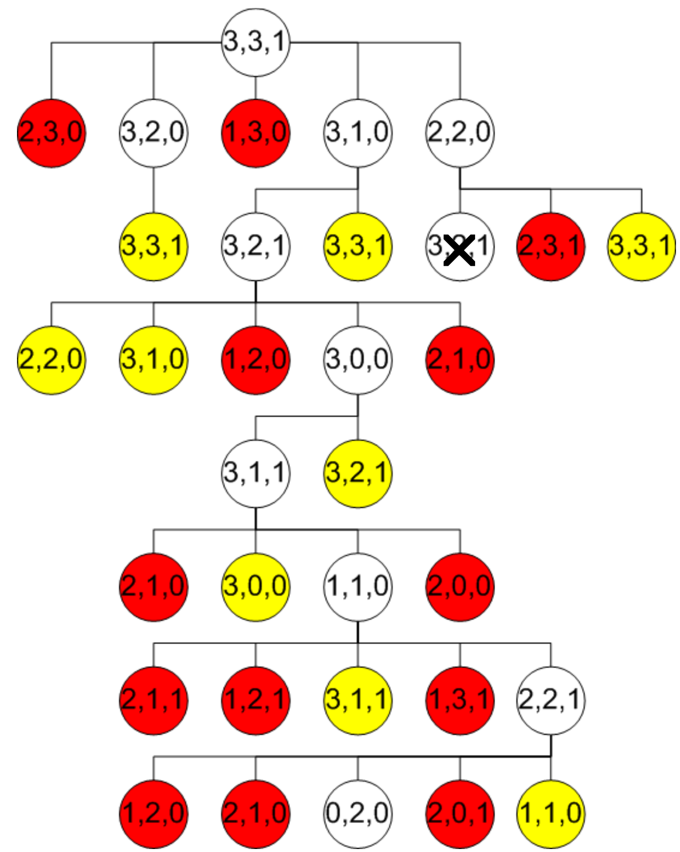
+/- (1,0,1)
+/- (0,1,1)
+/- (2,0,1)
+/- (0,2,1)
+/- (1,1,1)

In that order (left to right)

Red states = missionaries get eaten

Yellow states = repeated states

Breadth-first search on Missionaries & Cannibals



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

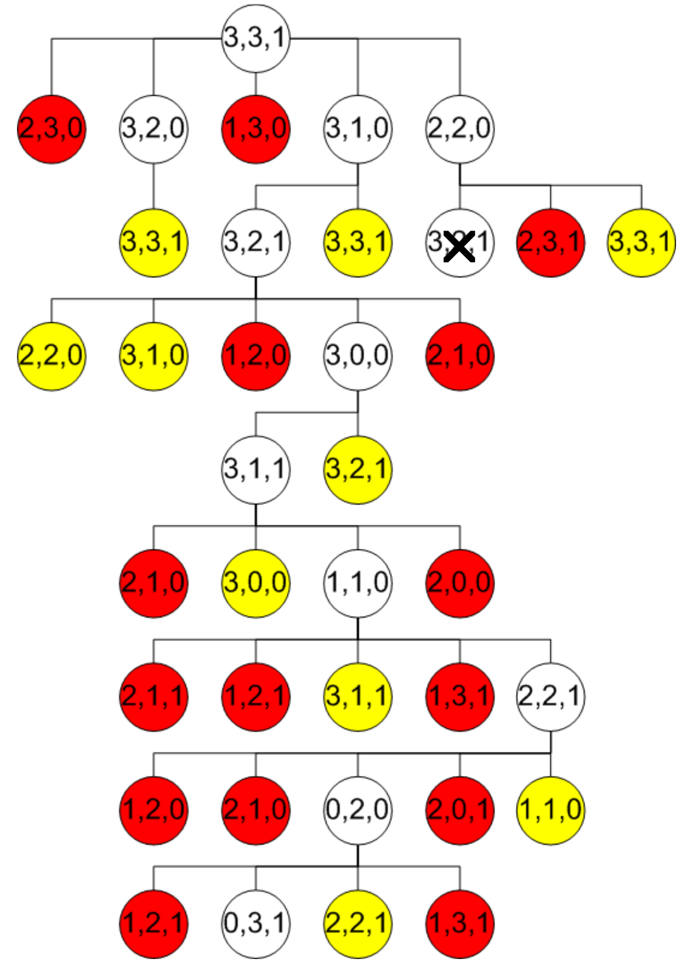
States are generated by applying:

+/- (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

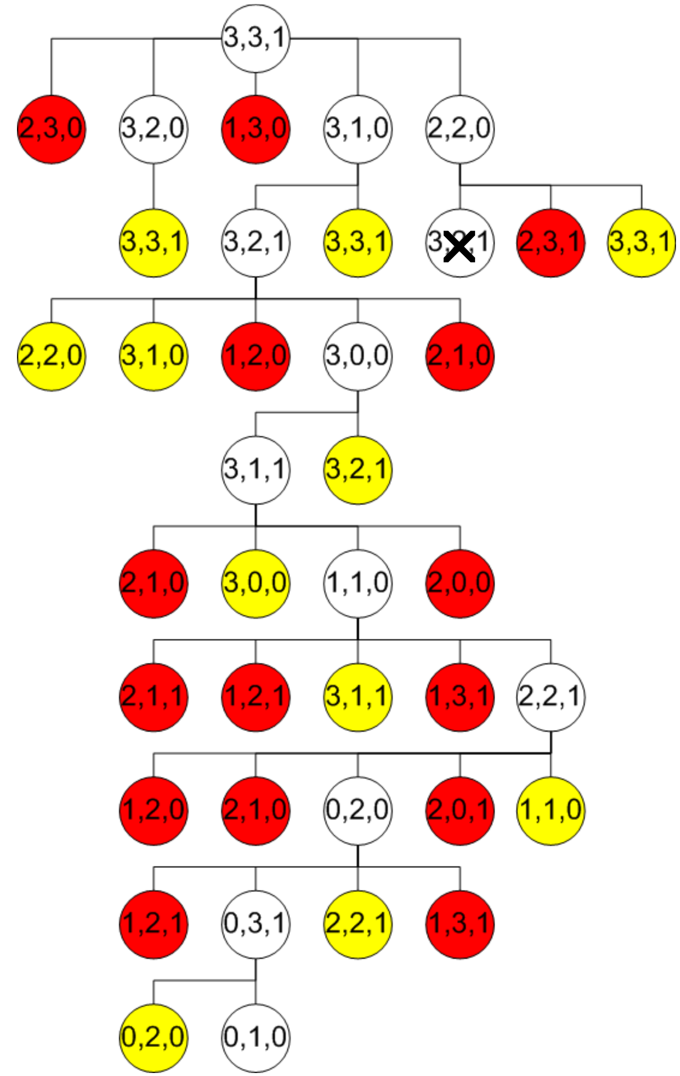
States are generated by applying:

+/- (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

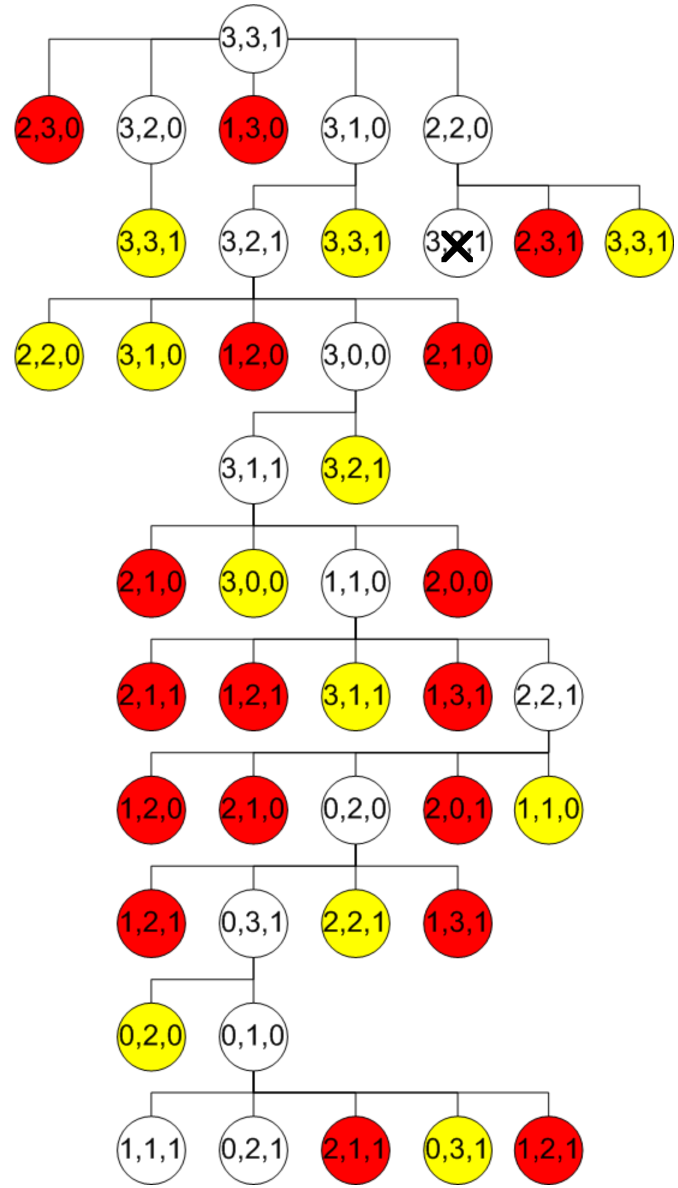
States are generated by applying:

+/- (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

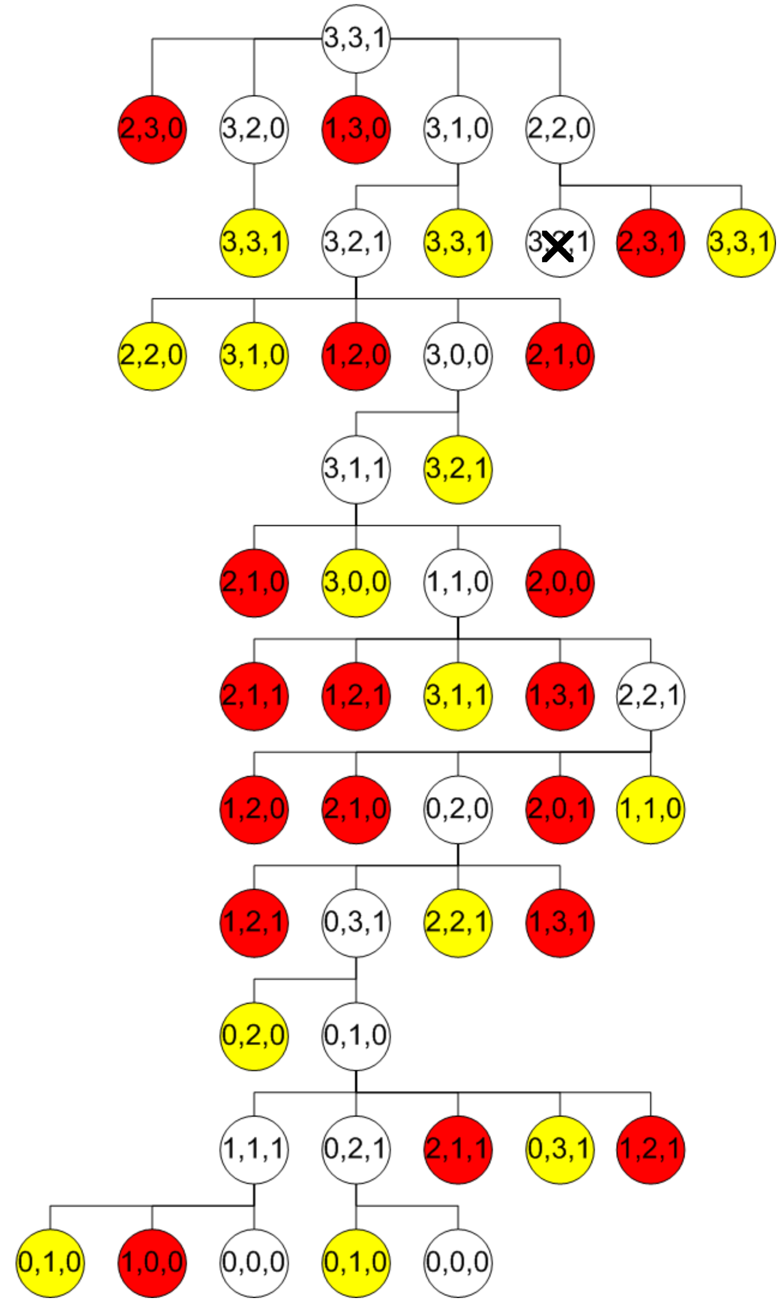
States are generated by applying:

+/- (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

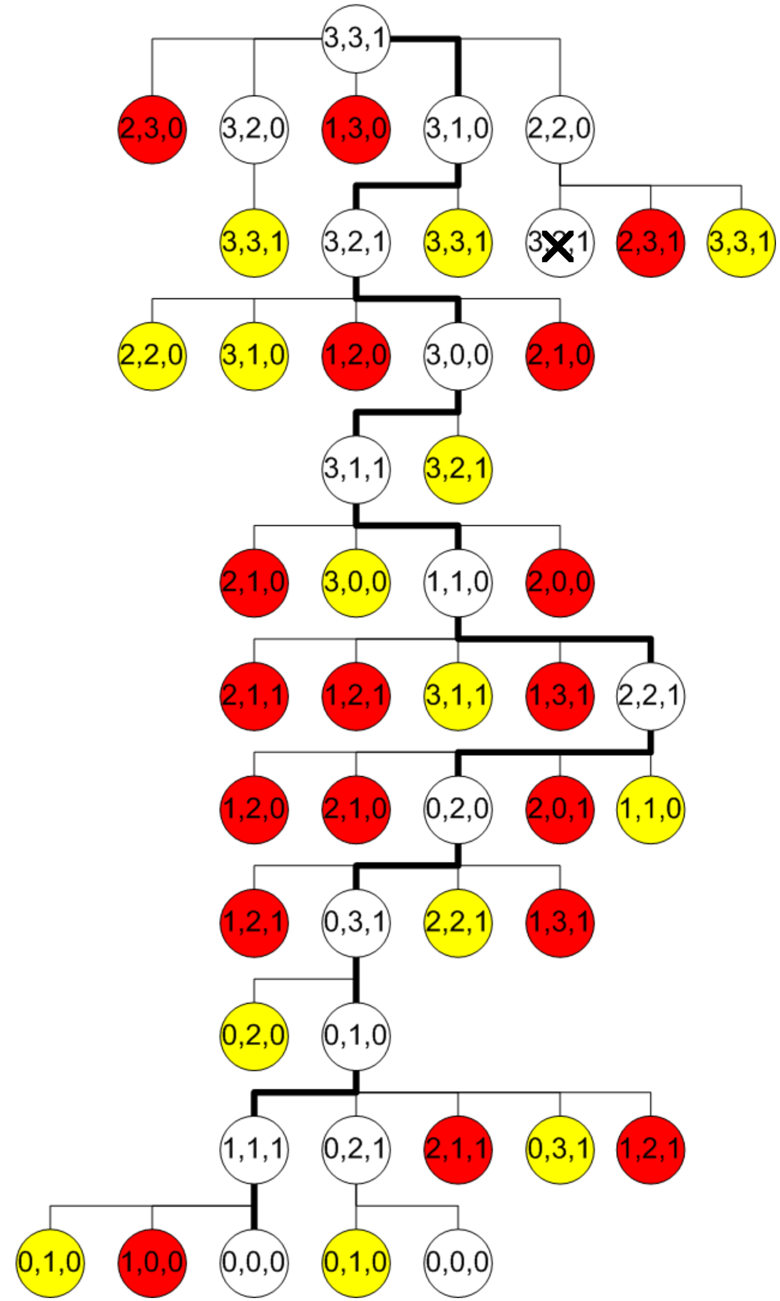
Yellow states = repeated states



Breadth-first search on Missionaries & Cannibals

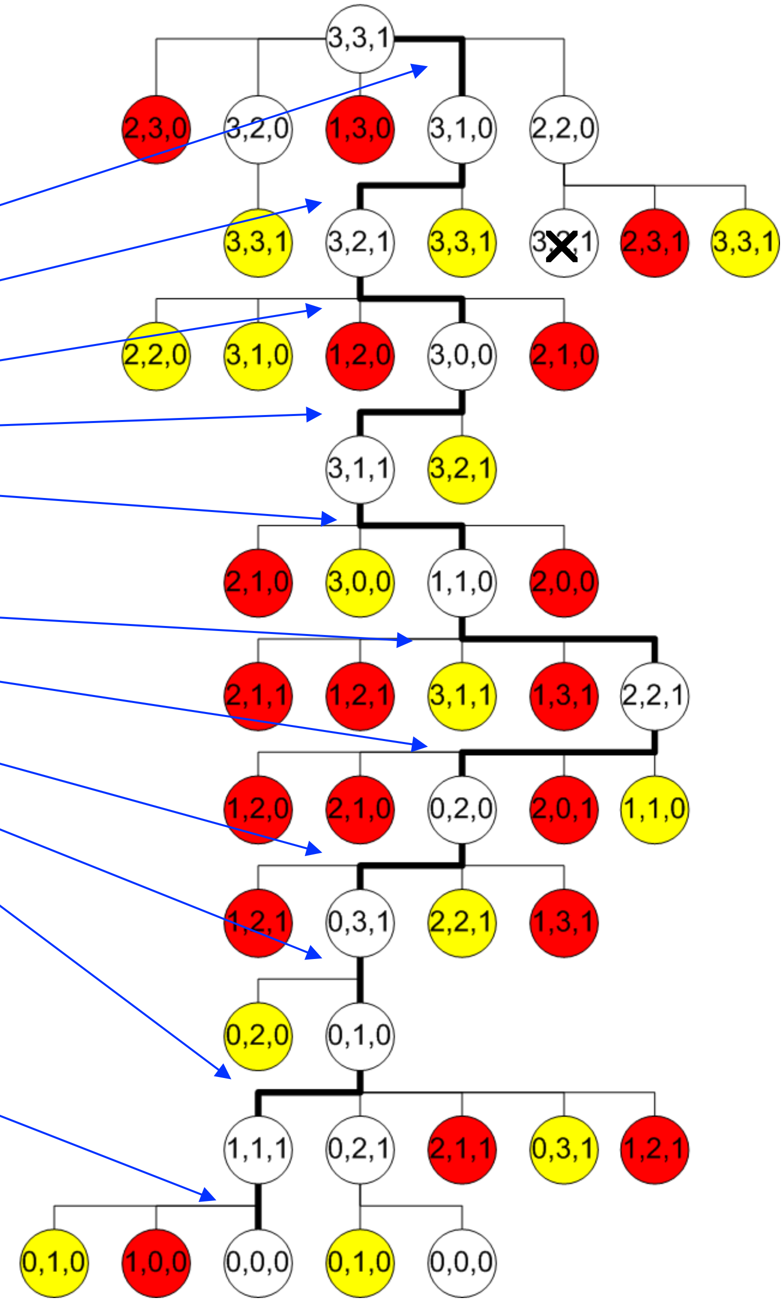
-(0,2,1) [2 cannibals cross L → R]
+(0,1,1) [1 cannibal crosses R → L]
-(0,2,1) [2 cannibals cross L → R]
+(0,1,1) [1 cannibal crosses R → L]
-(2,0,1) [2 missionaries cross L → R]
+(1,1,1) [1 cannibal & 1 missionary cross R → L]
-(2,0,1) [2 missionaries cross L → R]
+(0,1,1) [1 cannibal crosses R → L]
-(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 <u>optimal</u> solution (minimum number of crossings). 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 <u>optimal</u> 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)