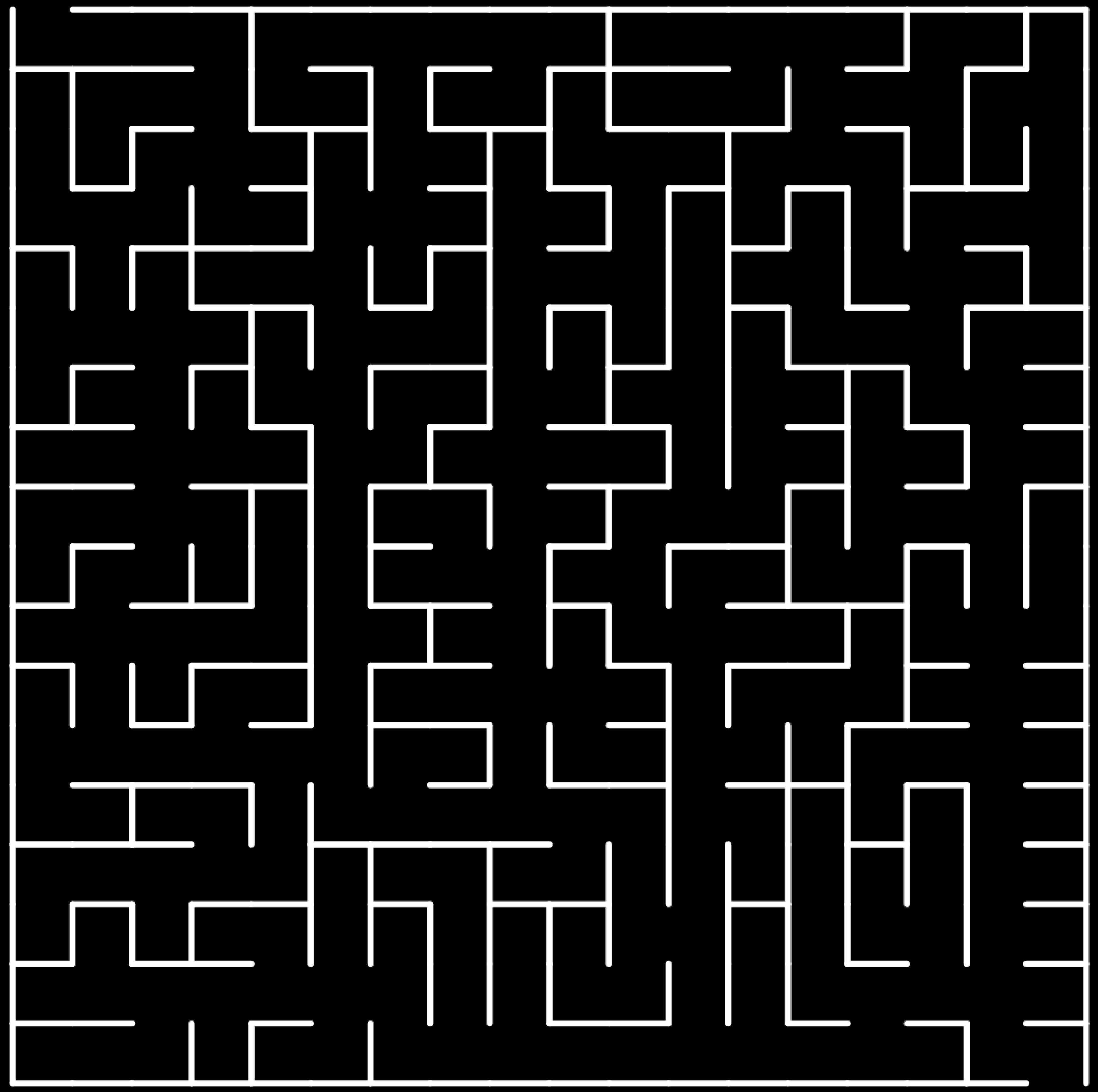
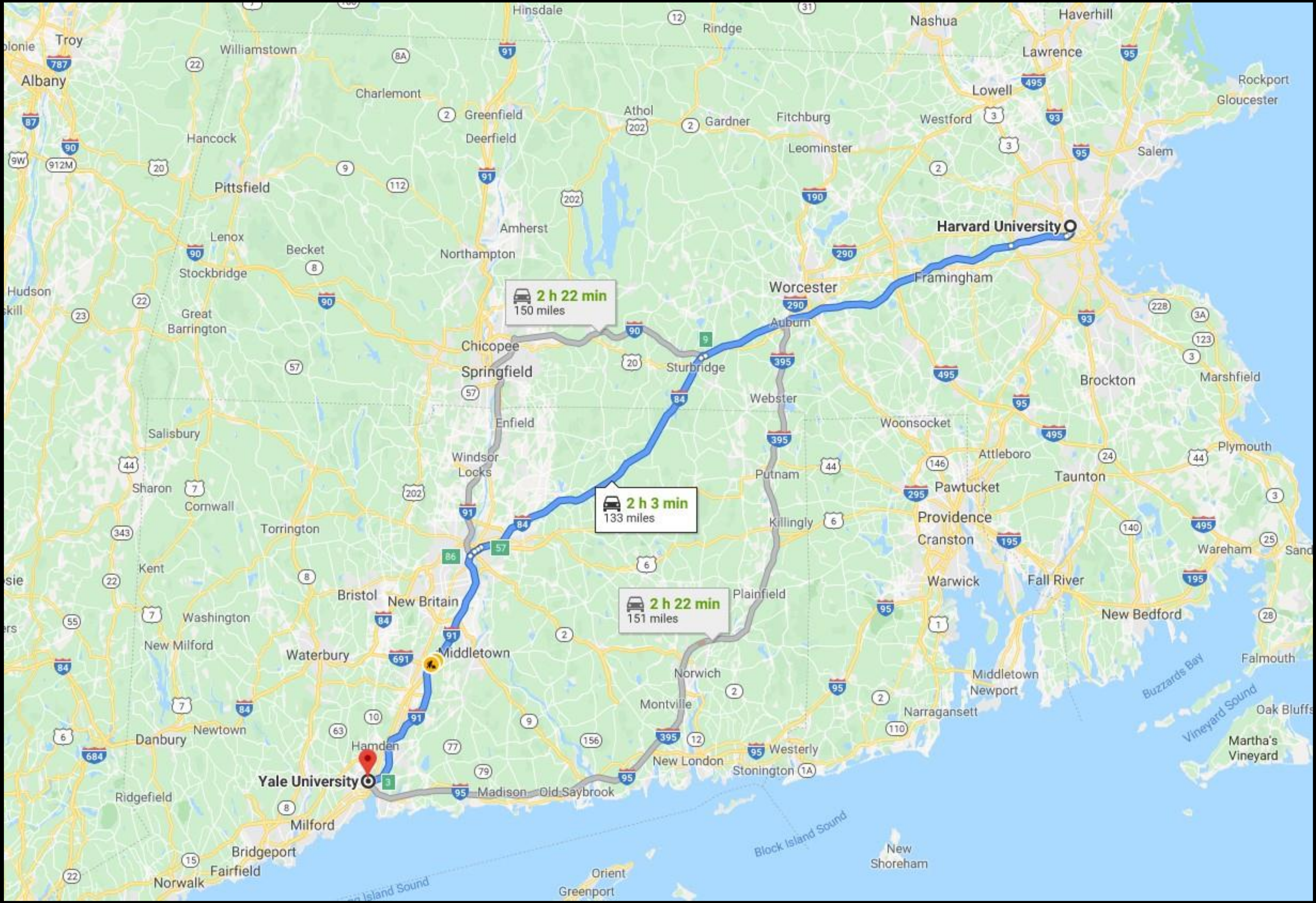
Artificial Intelligence

Search




Harvard University
Yale University
2 h 22 min
150 miles
2 h 3 min
133 miles
2 h 22 min
151 miles
Hinsdale
Ringde
Nashua
Haverhill
Troy
Albany
Williamstown
Lawrence
Lowell
Rockport
Gloucester
Charlemont
Greenfield
Deerfield
Athol
Gardner
Fitchburg
Westford
Leominster
Salem
Hancock
Pittsfield
Amherst
Northampton
Lenox
Becket
Stockbridge
Great Barrington
Hudson
Worcester
Framingham
Auburn
Chicopee
Springfield
Sturbridge
Webster
Brockton
Marshfield
Enfield
Salisbury
Windsor Locks
Woonsocket
Plymouth
Attleboro
Taunton
Putnam
Sharon
Cornwall
Torrington
Pawtucket
Providence
Cranston
Killingly
Kent
Warwick
Fall River
Wareham
Bristol
New Britain
Washington
Plainfield
New Bedford
New Milford
Waterbury
Middletown
Norwich
Falmouth
Montville
Middletown
Newport
Narragansett
Buzzards Bay
Vineyard Sound
Oak Bluffs
Martha's Vineyard
Danbury
Newtown
Hamden
Westerly
New London
Stonington
Madison
Old Saybrook
Ridgefield
Milford
Block Island Sound
New Shoreham
Bridgeport
Fairfield
Norwalk
Orient
Greenport

Search Problems in AI

Problem Solving

In computer science, problem-solving refers to artificial intelligence techniques, including various techniques such as forming efficient algorithms, heuristics, and performing root cause analysis to find desirable solutions.

The basic important of artificial intelligence is to solve problems just like humans.

Cont..

In today’s fast-paced digitized world, artificial intelligence techniques are used widely to automate systems that can use the resource and time efficiently. Some of the well-known problems experienced in everyday life are games and puzzles. Using AI techniques, we can solve these problems efficiently. In this sense, some of the most common problems resolved by AI are

- Travelling Salesman Problem
- Chess
- Crypt-arithmetic Problems
- Magic Squares
- Logical Puzzles and so on.

Problem Solving Techniques

In artificial intelligence, problems can be solved by using searching algorithms, evolutionary computations, knowledge representations, etc.

In this chapter, we’re going to discuss the various searching techniques that are used to solve a problem.

In general, searching is referred to as finding information one needs.

Cont..

The process of problem-solving using searching consists of the following steps.

- Define the problem
- Analyze the problem
- Identification of possible solutions
- Choosing the optimal solution
- Implementation

Let’s discuss some of the essential properties of search algorithms.

Properties of Search Algorithms

- **Completeness**

A search algorithm is said to be complete when it gives a solution or returns any solution for a given random input.

- **Optimality**

If a solution found is best (lowest path cost) among all the solutions identified, then that solution is said to be an optimal one.

- **Time Complexity**

The time taken by an algorithm to complete its task is called time complexity. If the algorithm completes a task in a lesser amount of time, then it is an efficient one.

- **Space Complexity**

It is the maximum storage or memory taken by the algorithm at any time while searching.

Types of Search Algorithms

Now let’s see the types of the search algorithm.

Based on the search problems, we can classify

the search algorithm as

- ❑**Uninformed search**
- ❑**Informed search**

agent

entity that perceives its environment and acts upon that environment

state

a configuration of the agent and its environment

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

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

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

initial state

the state in which the agent begins

initial state

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

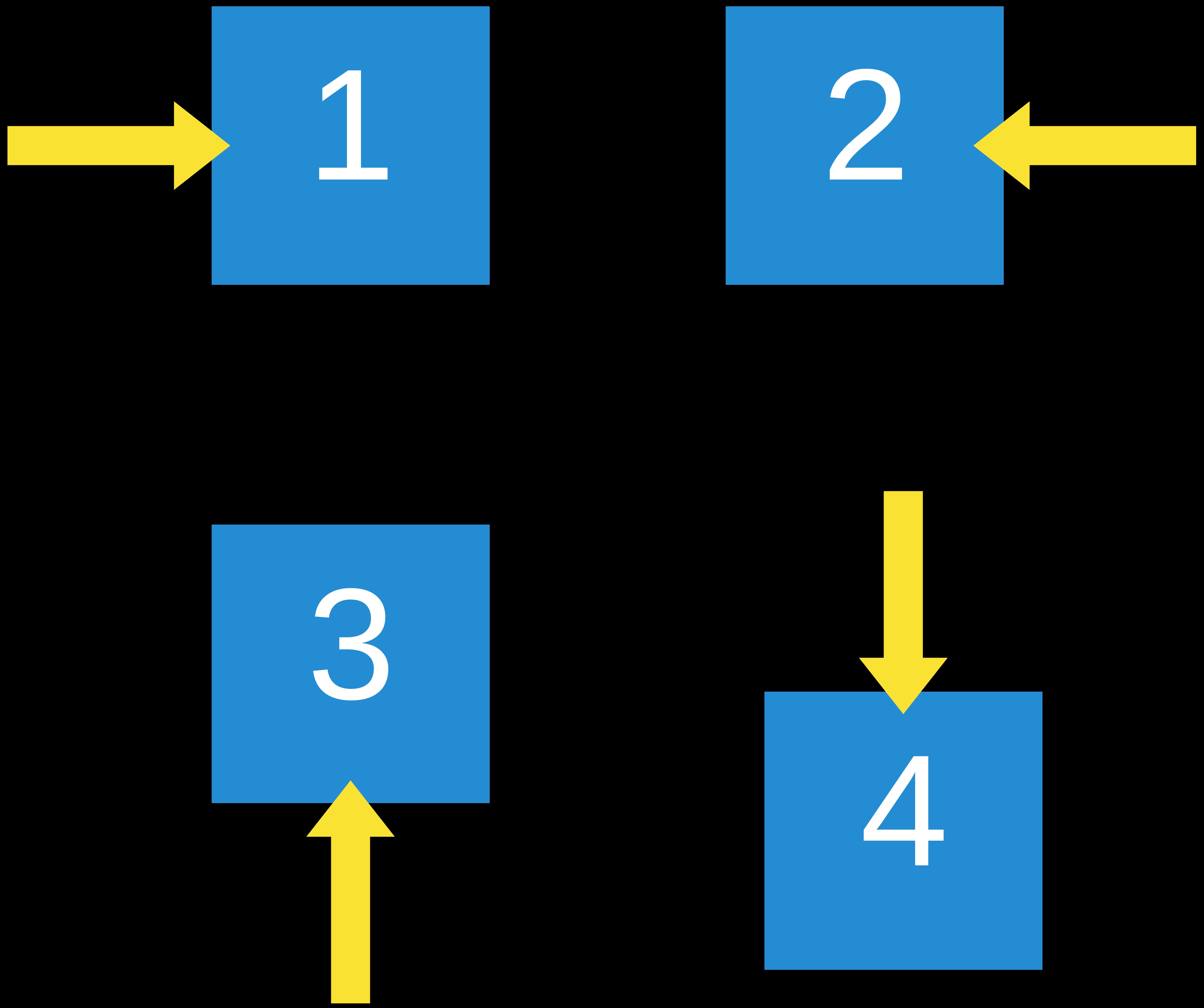
actions

choices that can be made in a state

actions

$\textsc{Actions}(s)$ returns the set of actions that can be executed in state s

actions



transition model

a description of what state results from performing any applicable action in any state

transition model

$\textsc{Result}(s, a)$ returns the state resulting from performing action a in state s

RESULT(

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

, $\rightarrow$) =

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

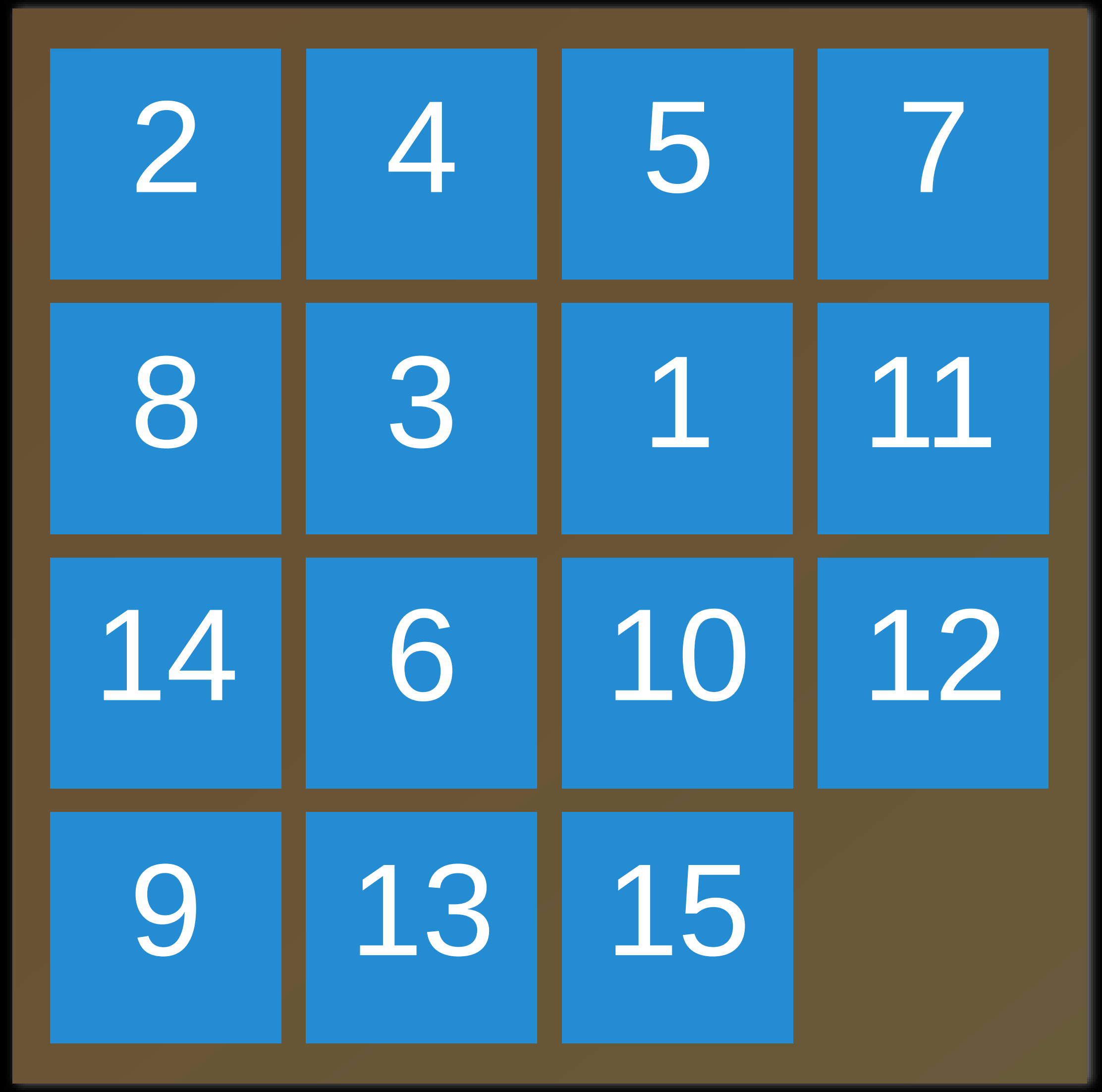
RESULT(

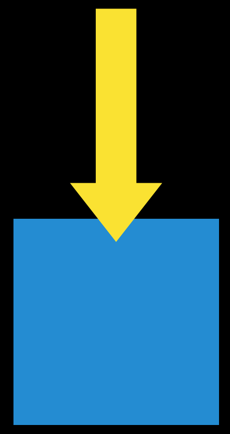
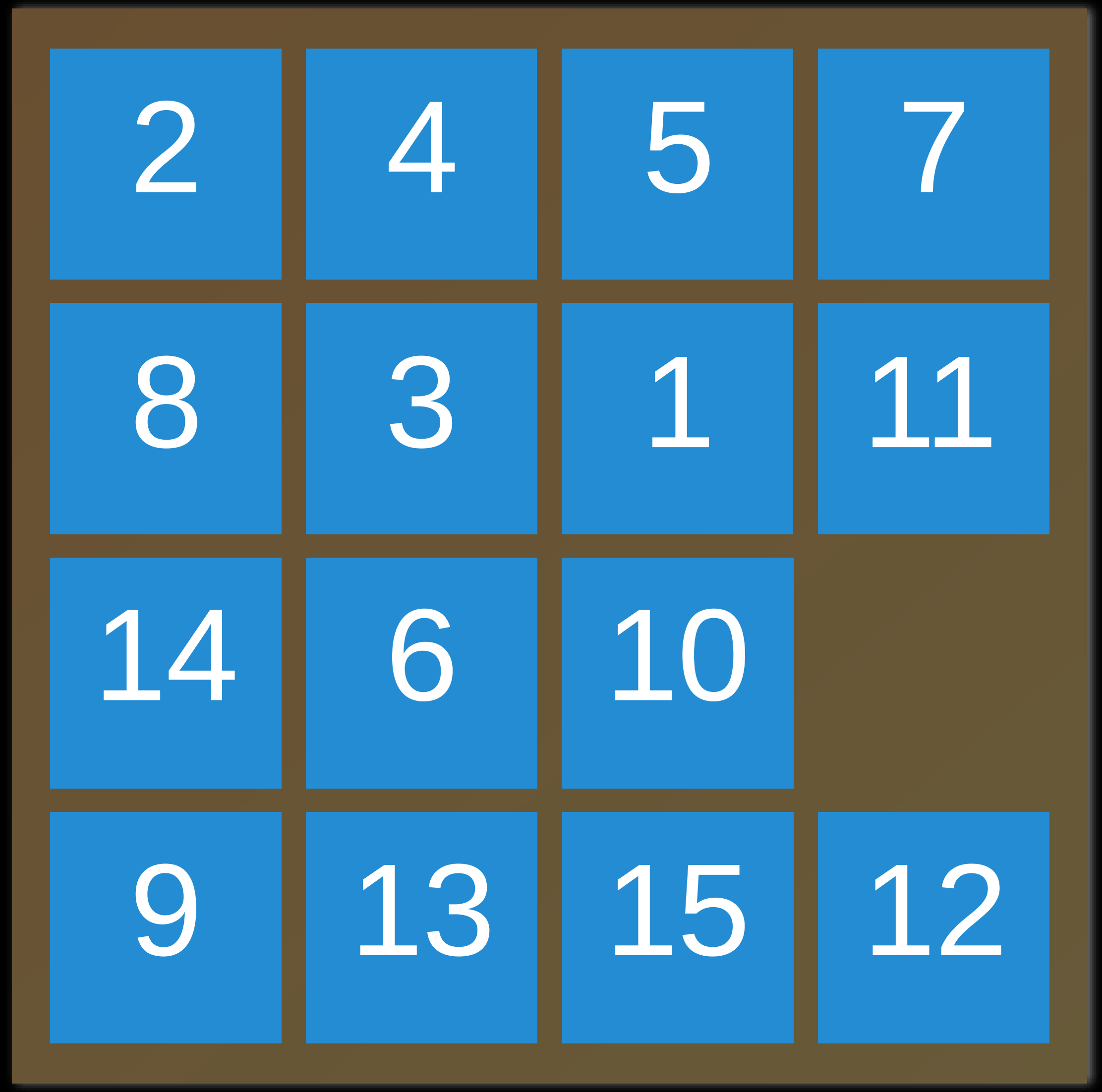
2	4	5	7
8	3	1	11
14	6	10	12
9	13	15	

, $\downarrow$) =

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

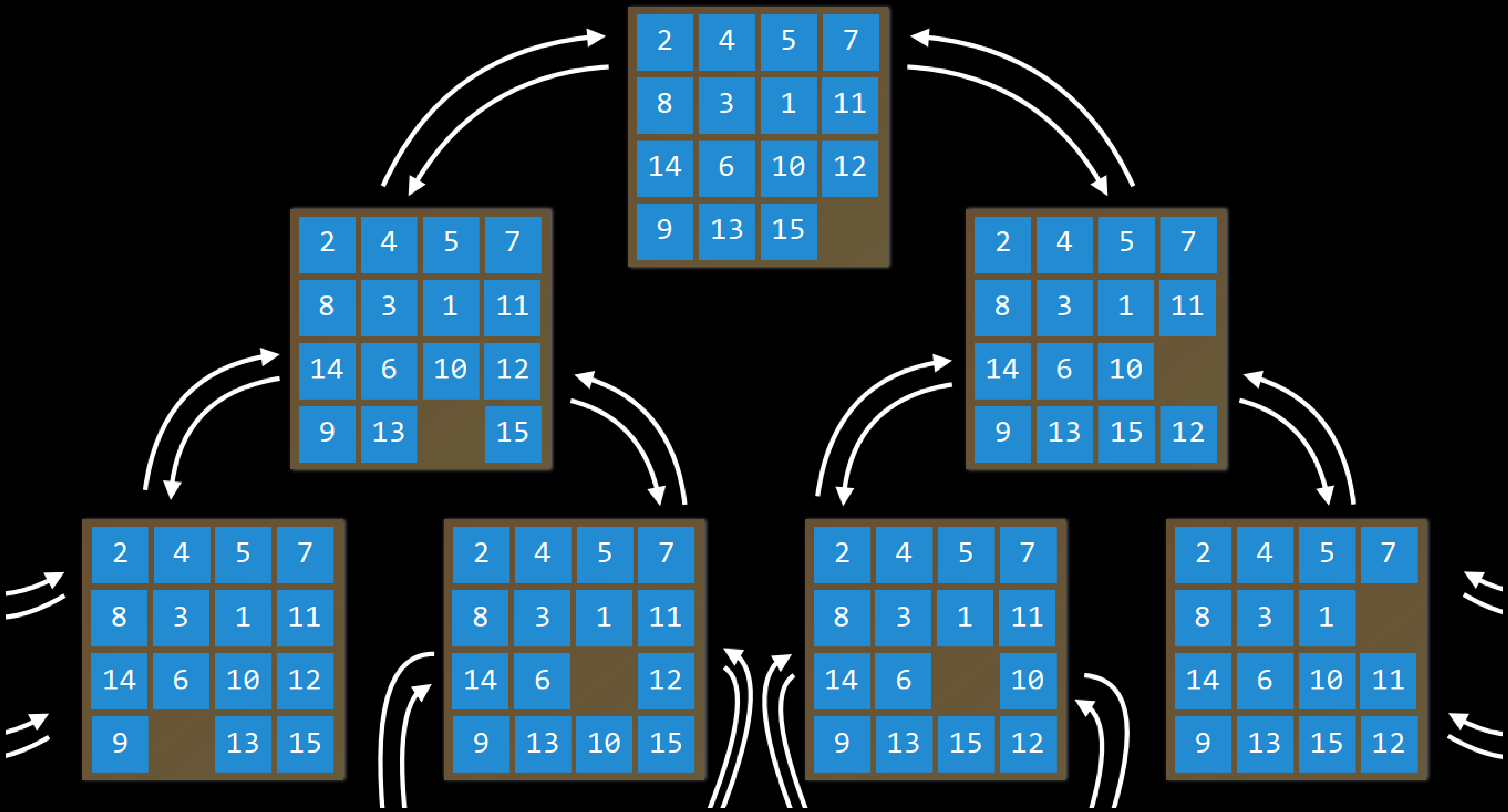
transition model

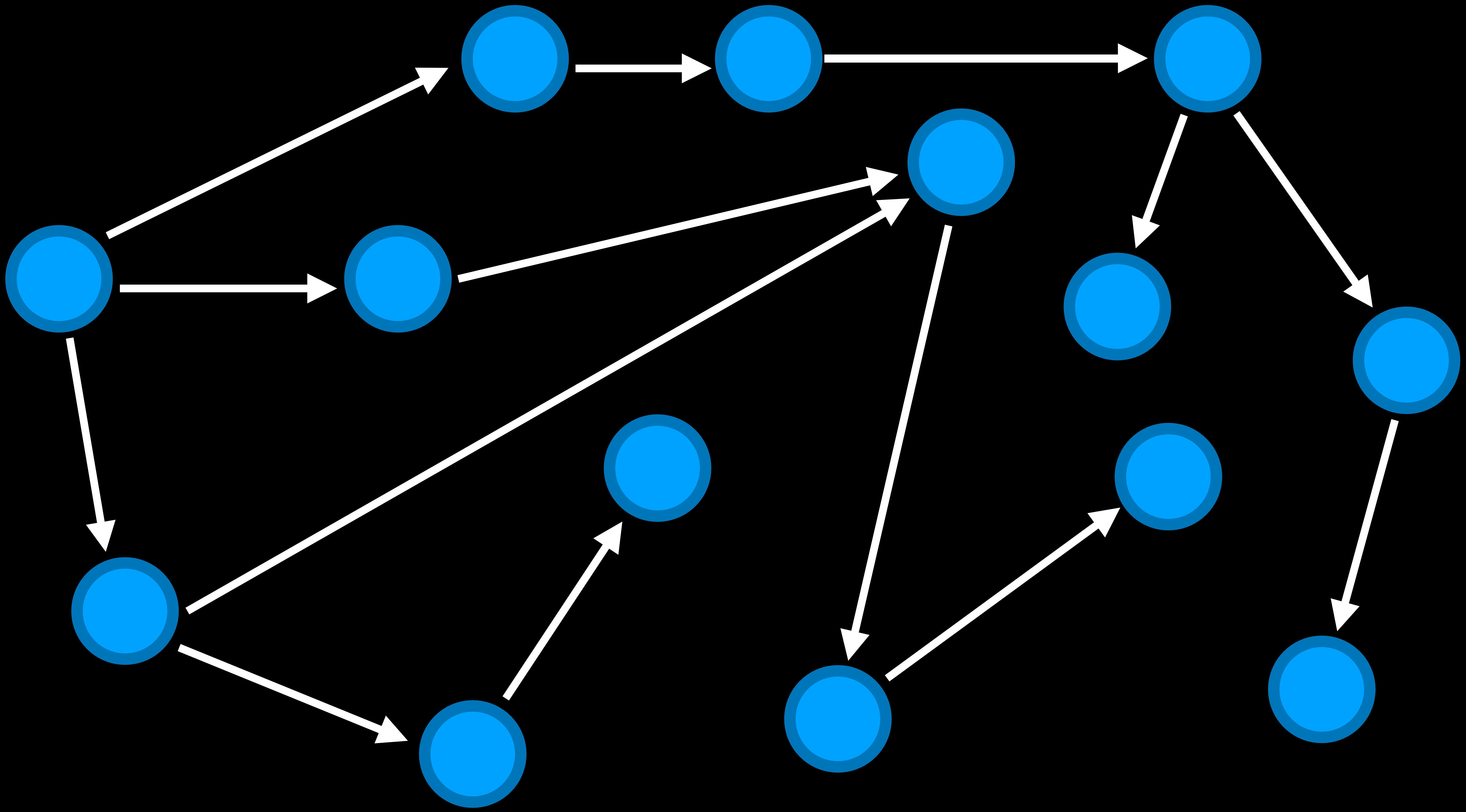
RESULT(

, ) = 

state space

the set of all states reachable from the initial state by any sequence of actions


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

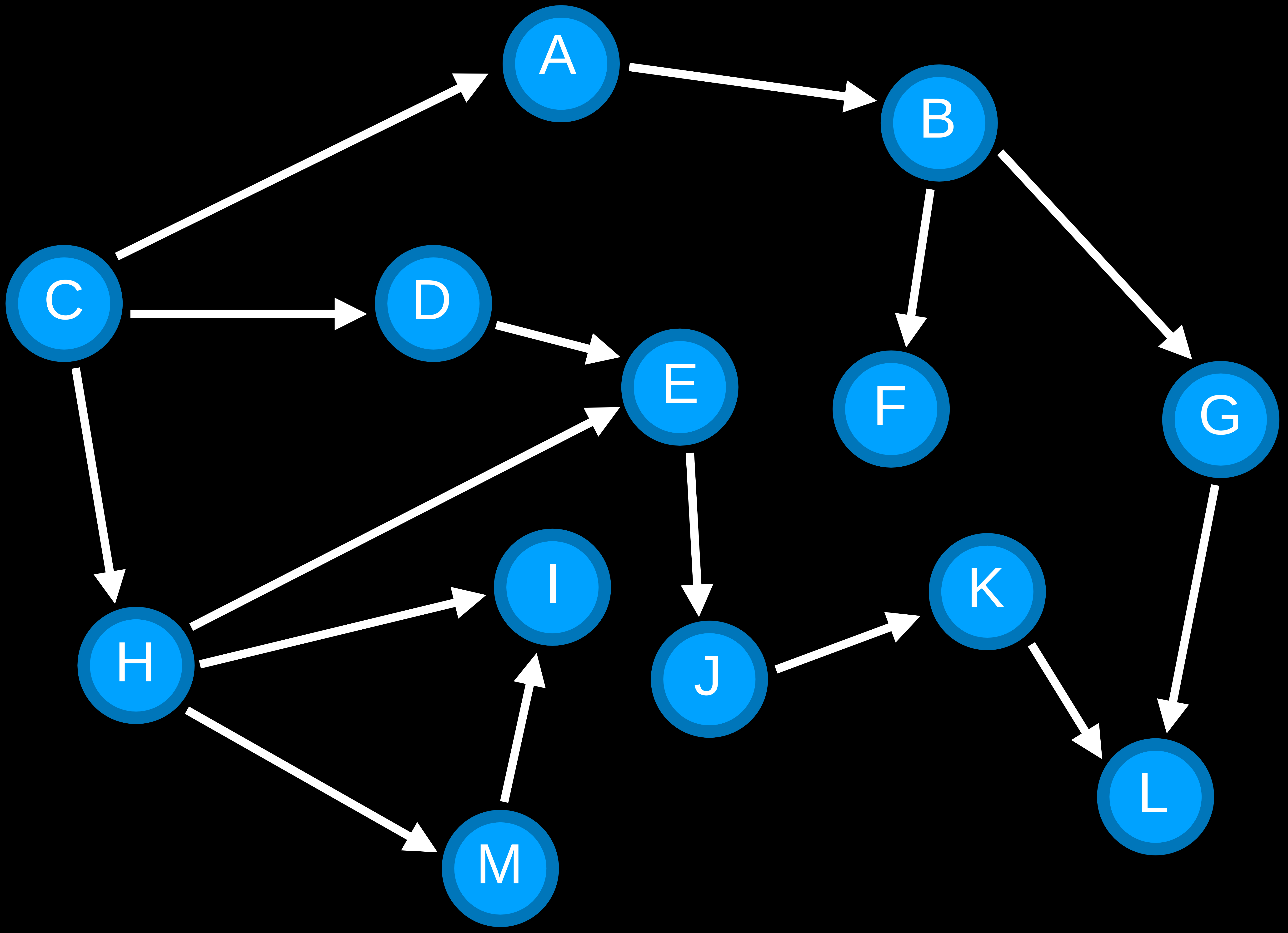


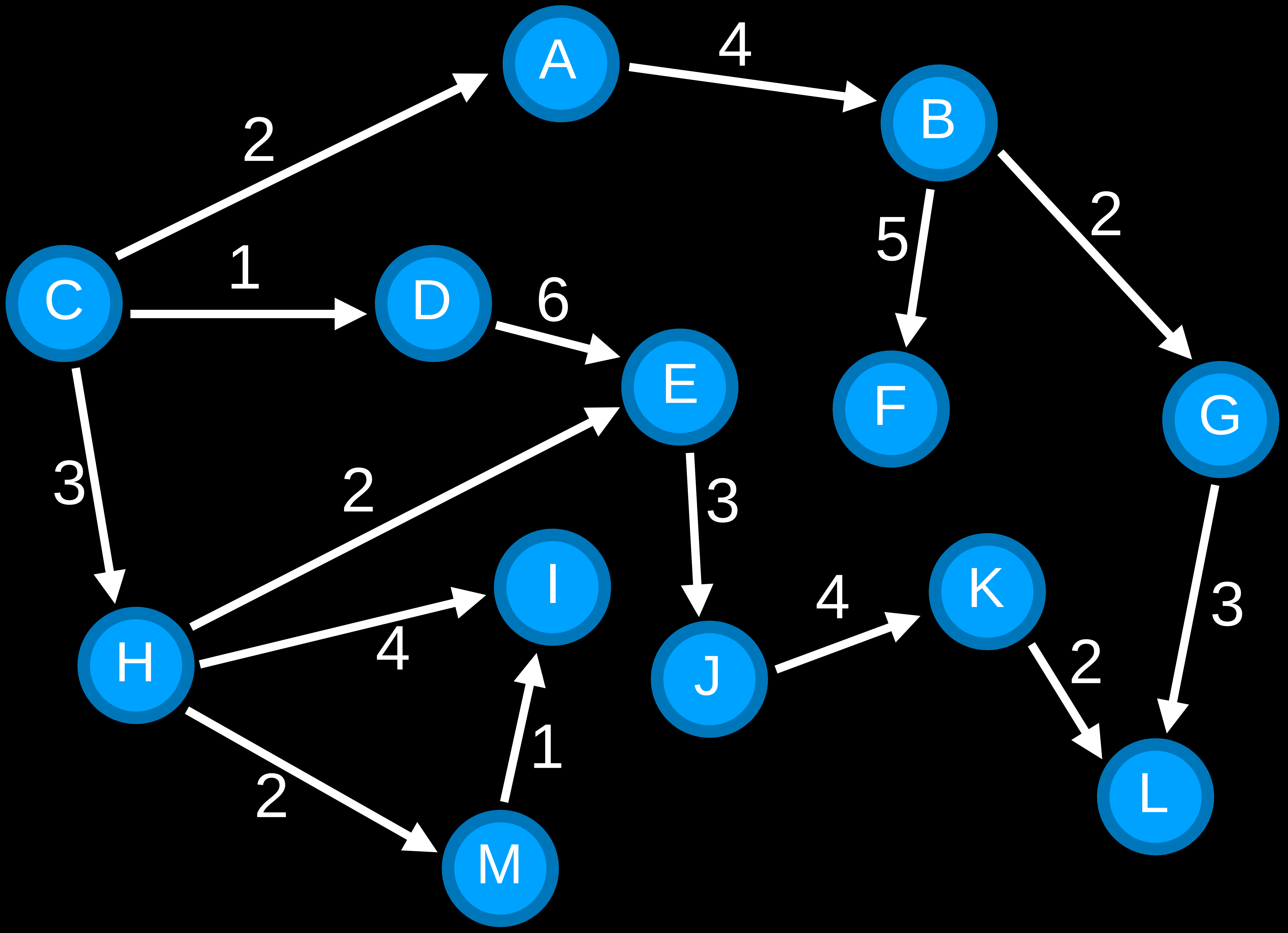
goal test

way to determine whether a given state is a goal state

path cost

numerical cost associated with a given path


A
B
C
D
E
F
G
H
I
J
K
L
M


A
B
C
D
E
F
G
H
I
J
K
L
M
2
4
1
6
5
2
3
2
3
4
4
2
3
2
1

Search Problems

- initial state
- actions
- transition model
- goal test
- path cost function

solution

a sequence of actions that leads from the initial state to a goal state

optimal solution

a solution that has the lowest path cost among all solutions

node

a data structure that keeps track of

- a **state**
- a **parent** (node that generated this node)
- an **action** (action applied to parent to get node)
- a **path cost** (from initial state to node)

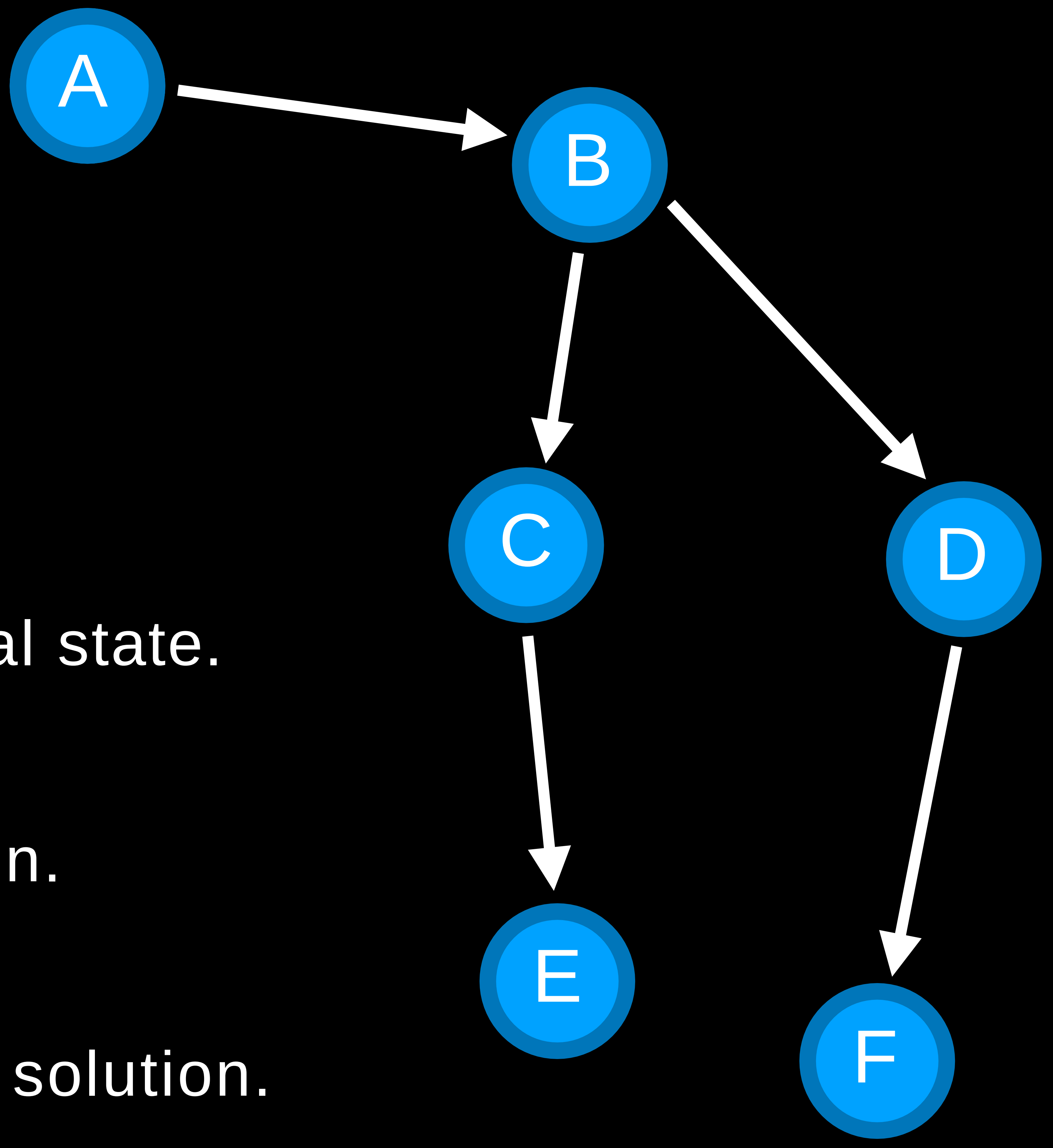
Approach

- Start with a **frontier** that contains the initial state.
- Repeat:
    - If the frontier is empty, then no solution.
    - Remove a node from the frontier.
    - If node contains goal state, return the solution.
    - **Expand** node, add resulting nodes to the frontier.

Find a path from A to E.

Frontier

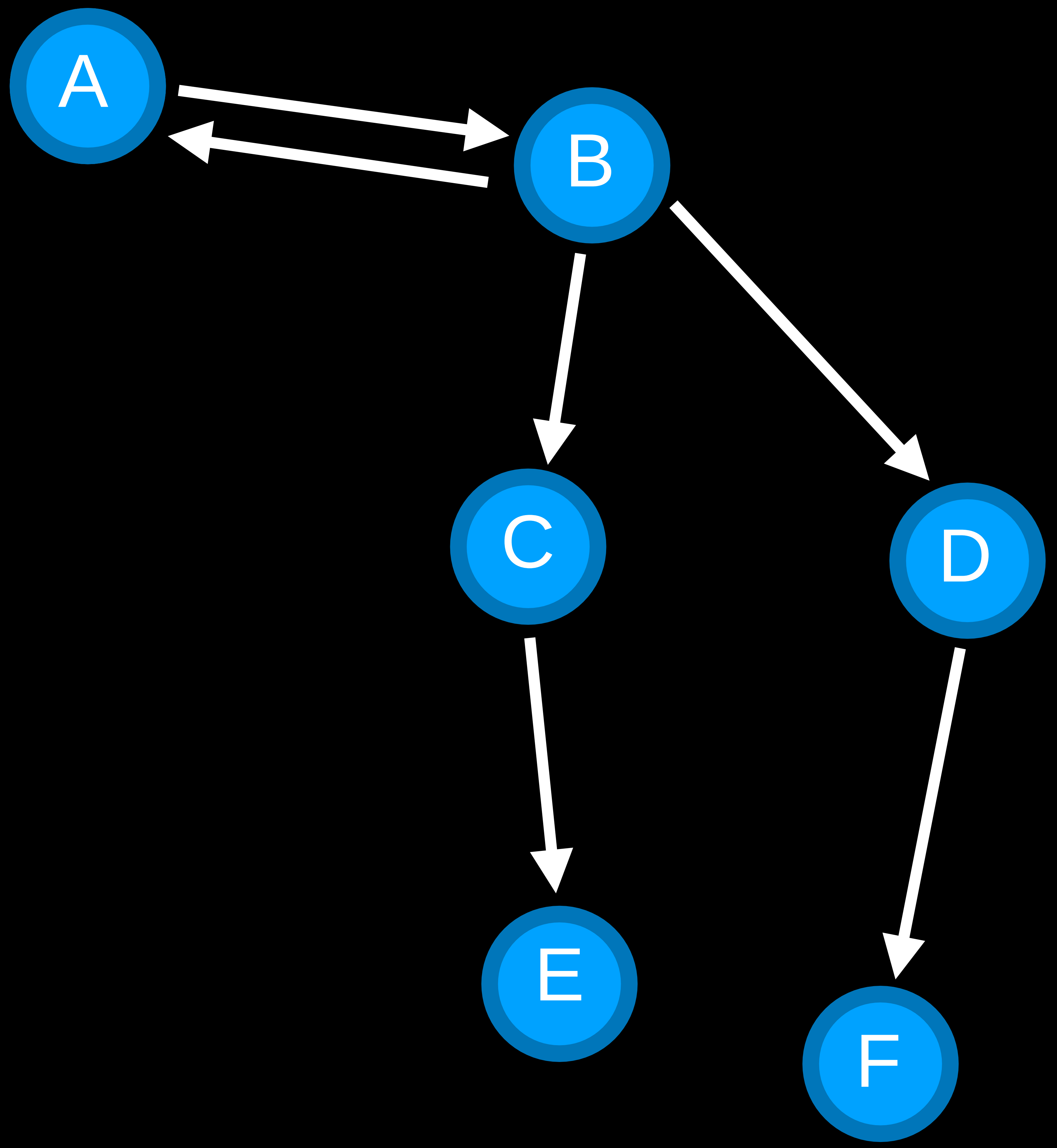
- Start with a **frontier** that contains the initial state.
- Repeat:
  - If the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - If node contains goal state, return the solution.
  - **Expand** node, add resulting nodes to the frontier.



What could go wrong?

Find a path from A to E.

Frontier



Revised Approach

- Start with a **frontier** that contains the initial state.
- Start with an empty **explored set**.
- Repeat:
  - If the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - If node contains goal state, return the solution.
  - Add the node to the explored set.
  - **Expand** node, add resulting nodes to the frontier if they aren't already in the frontier or the explored set.

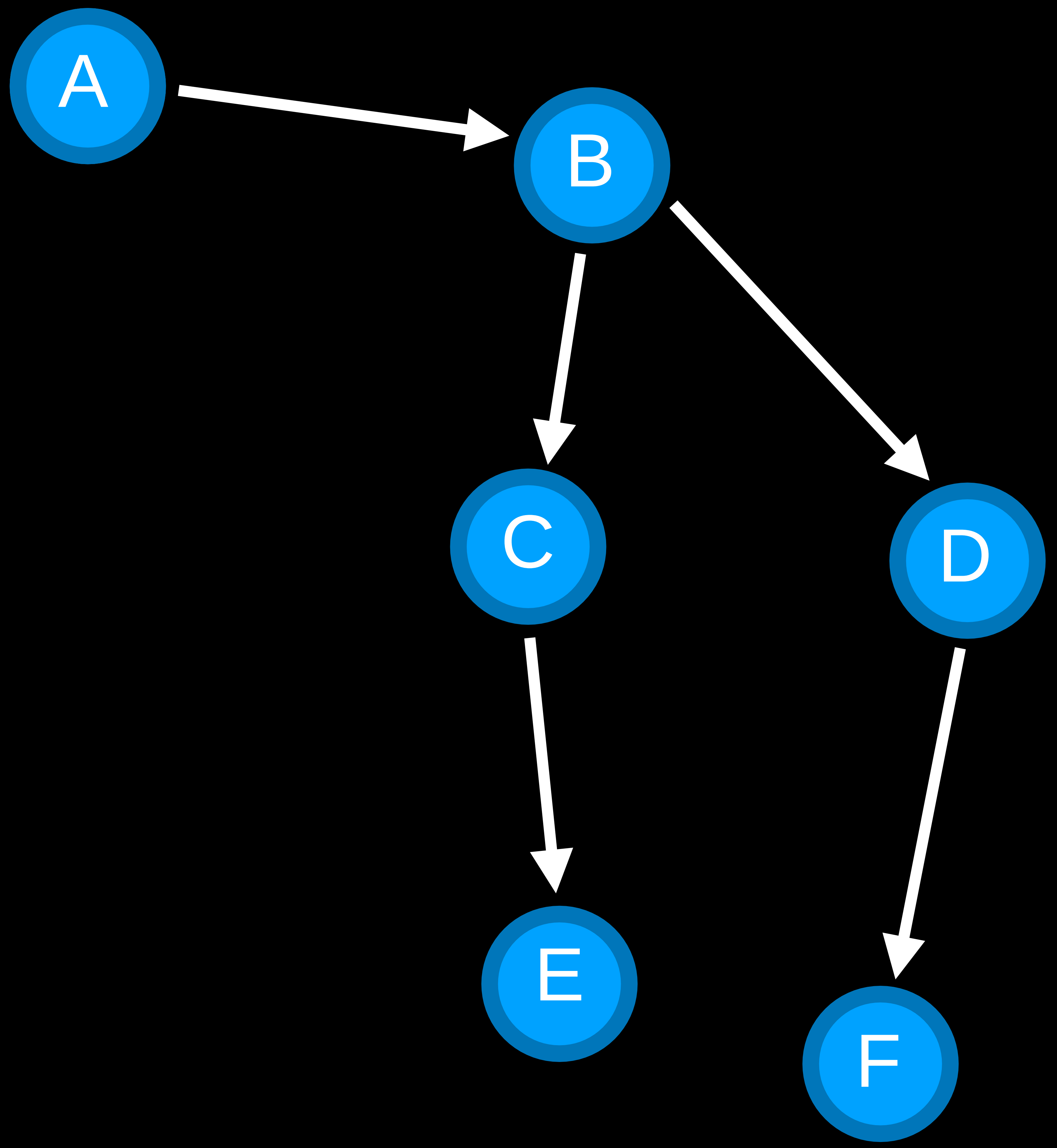
stack

last-in first-out data type

Find a path from A to E.

Frontier

Explored Set



Depth-First Search

depth-first search

search algorithm that always expands the deepest node in the frontier

Breadth-First Search

breadth-first search

search algorithm that always expands the shallowest node in the frontier

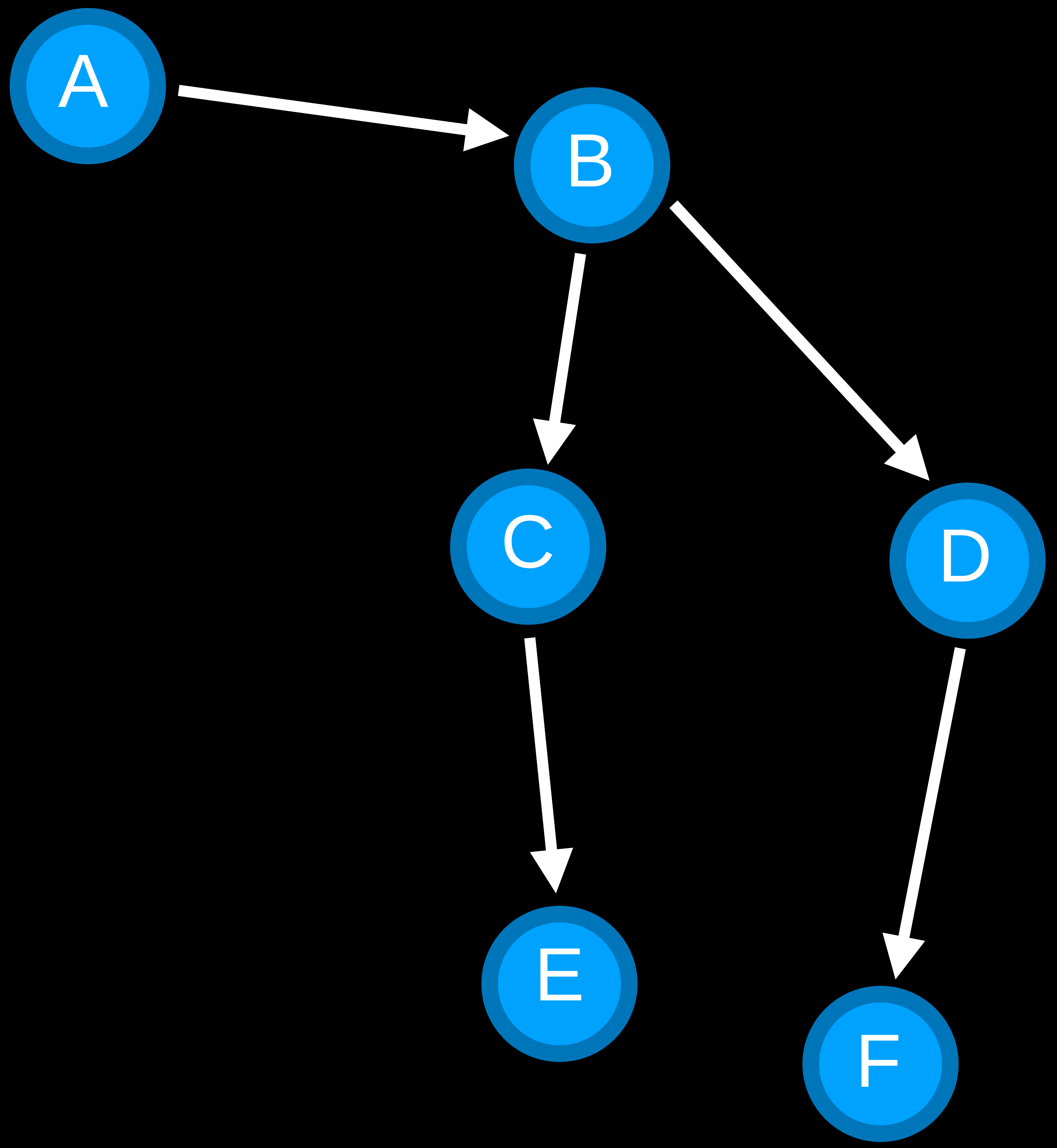
queue

first-in first-out data type

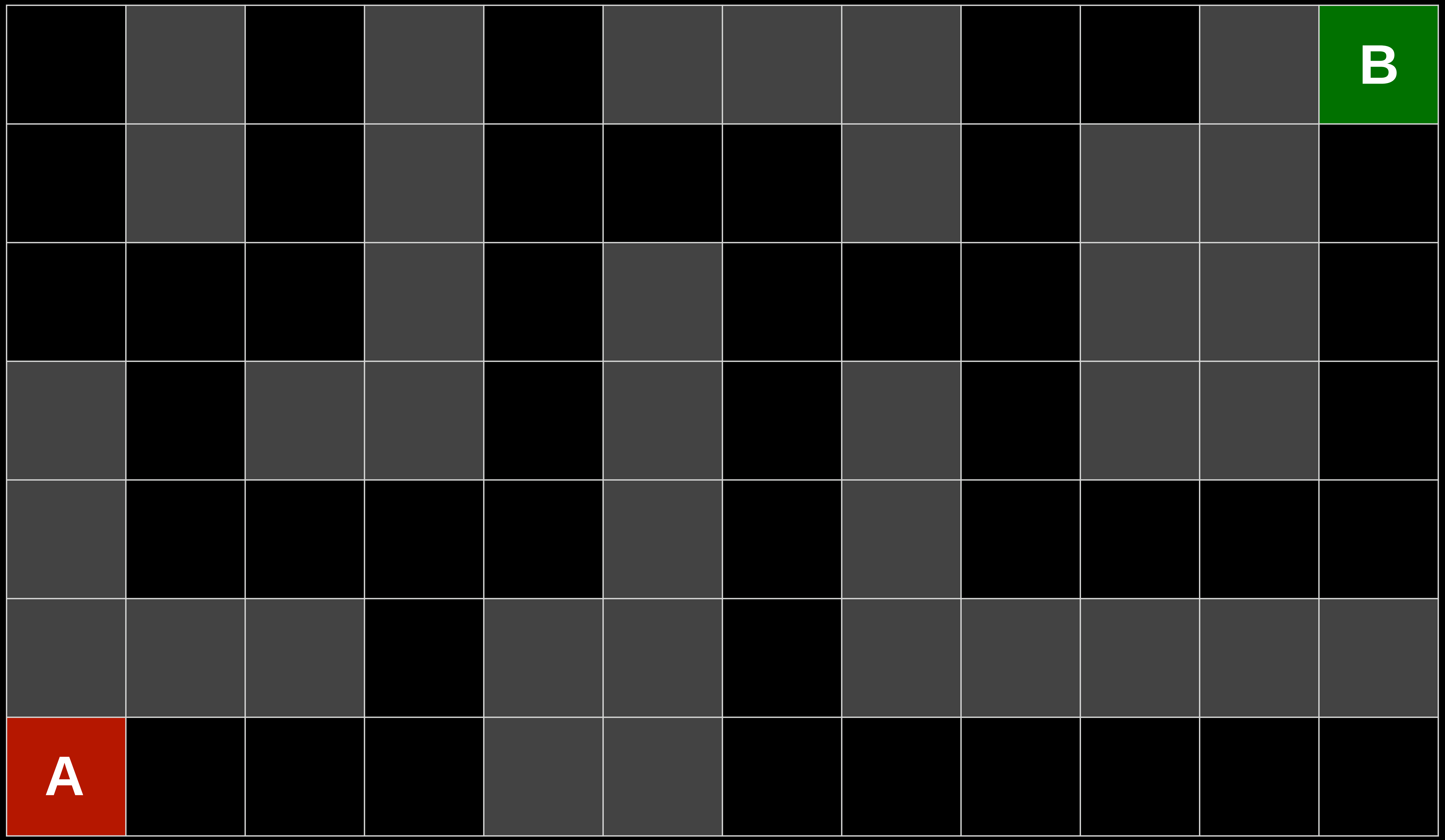
Find a path from A to E.

Frontier

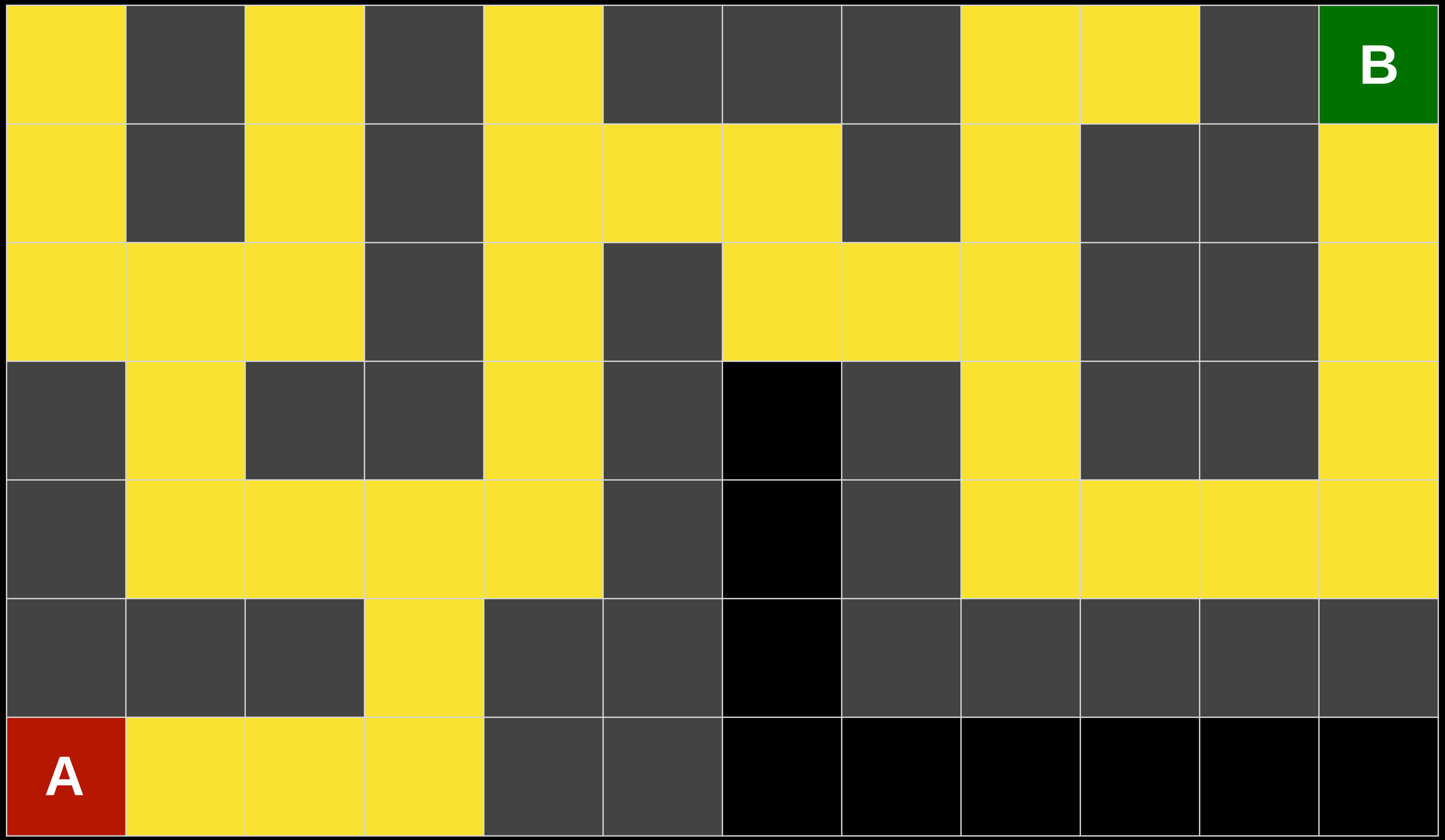
Explored Set



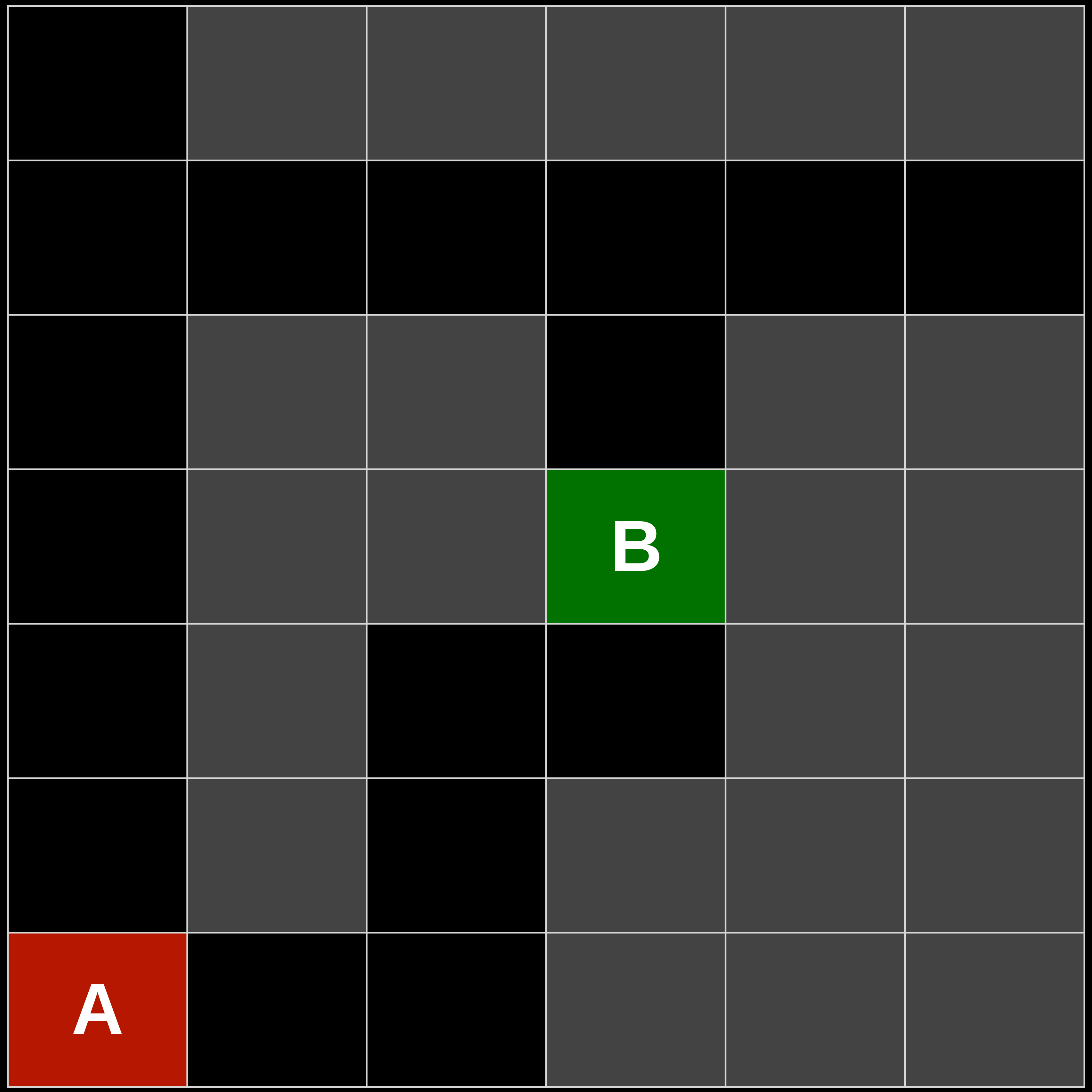
Depth-First Search



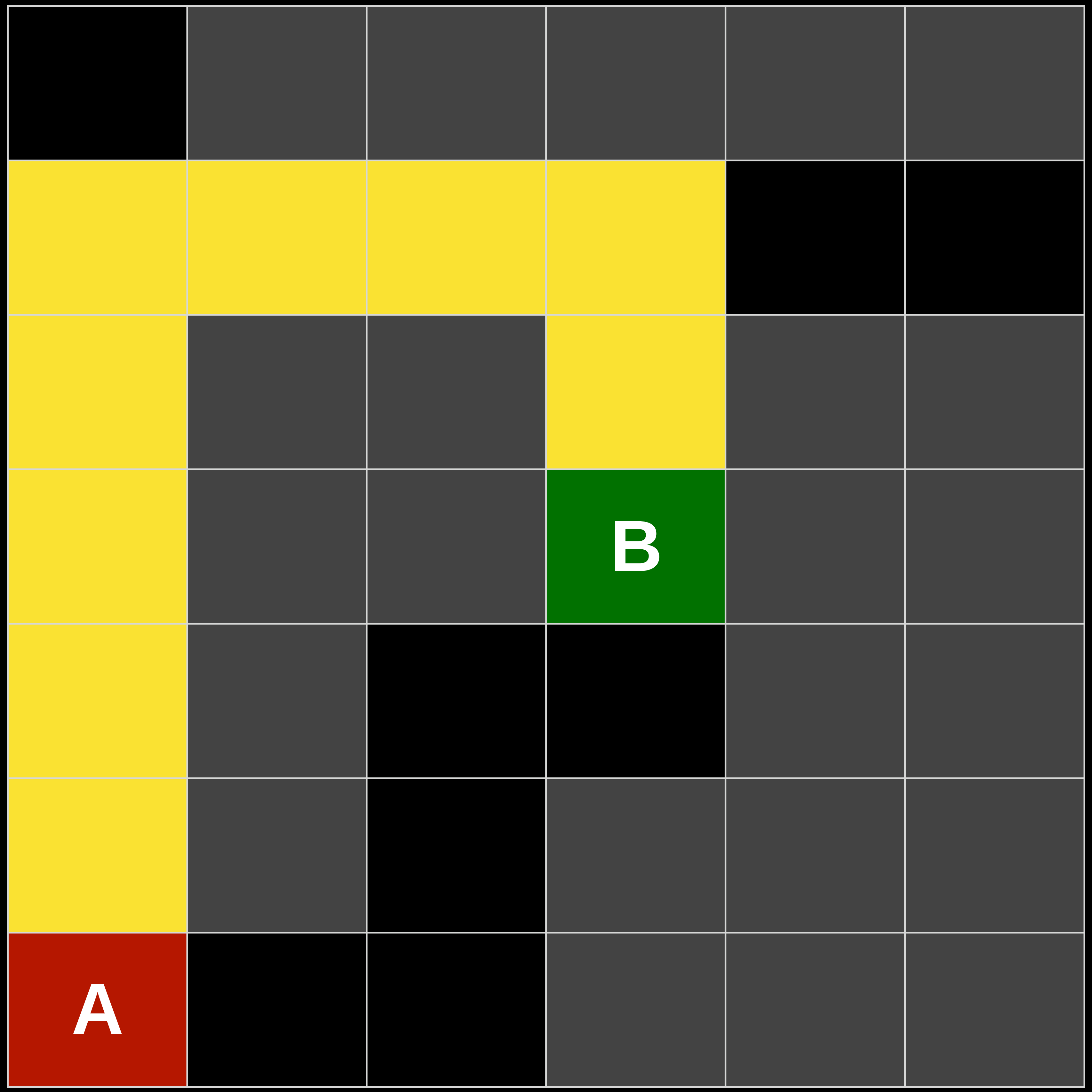
Depth-First Search



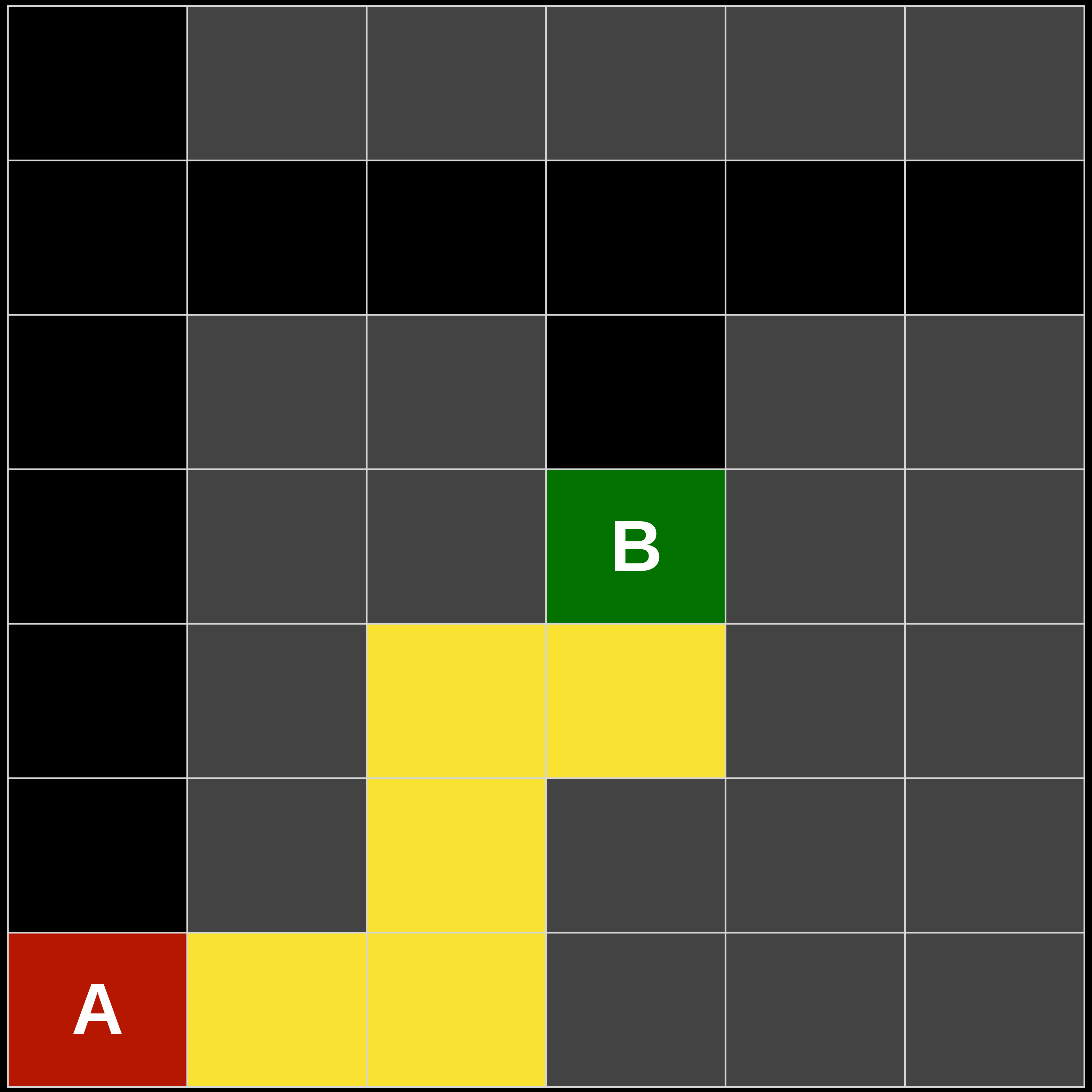
Depth-First Search



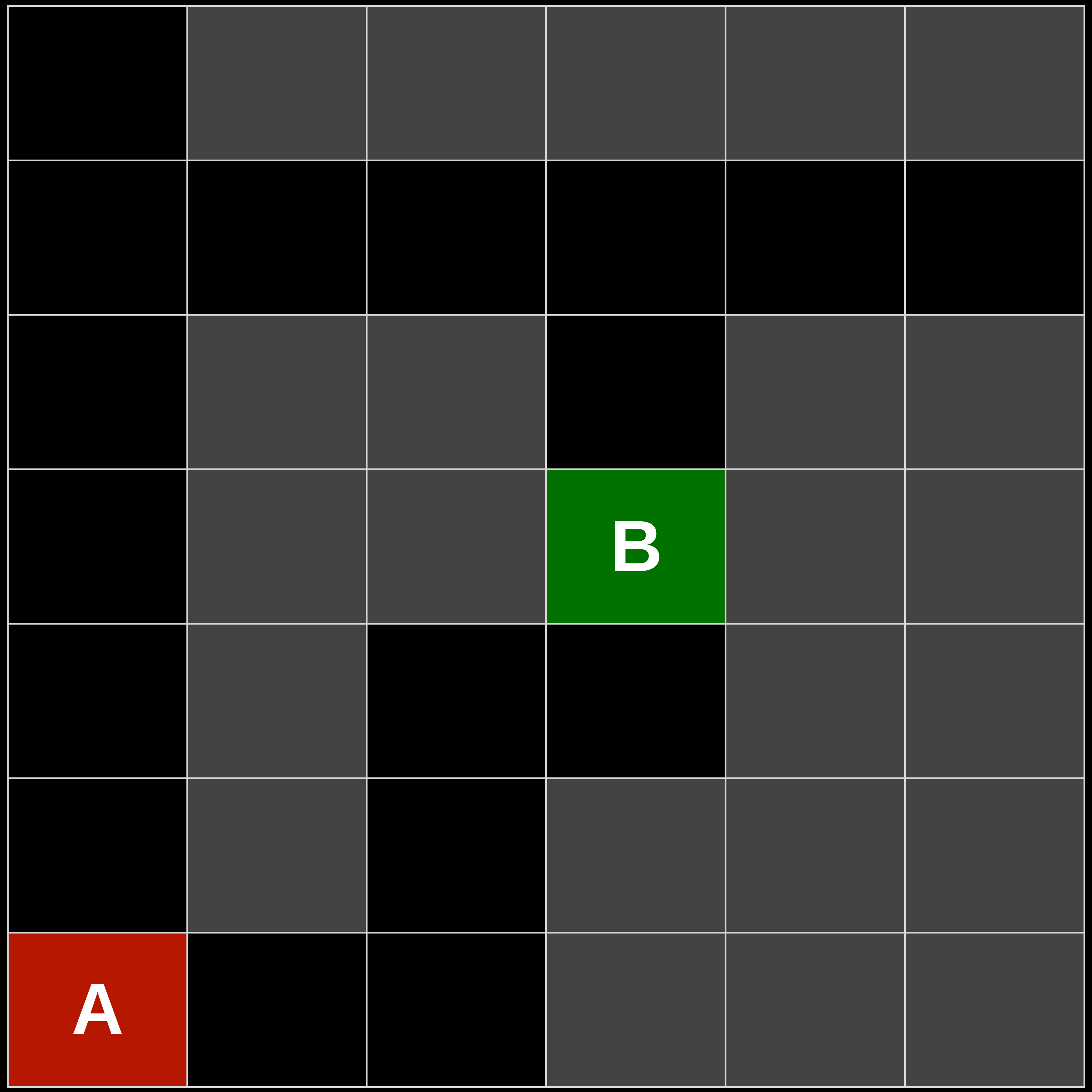
Depth-First Search



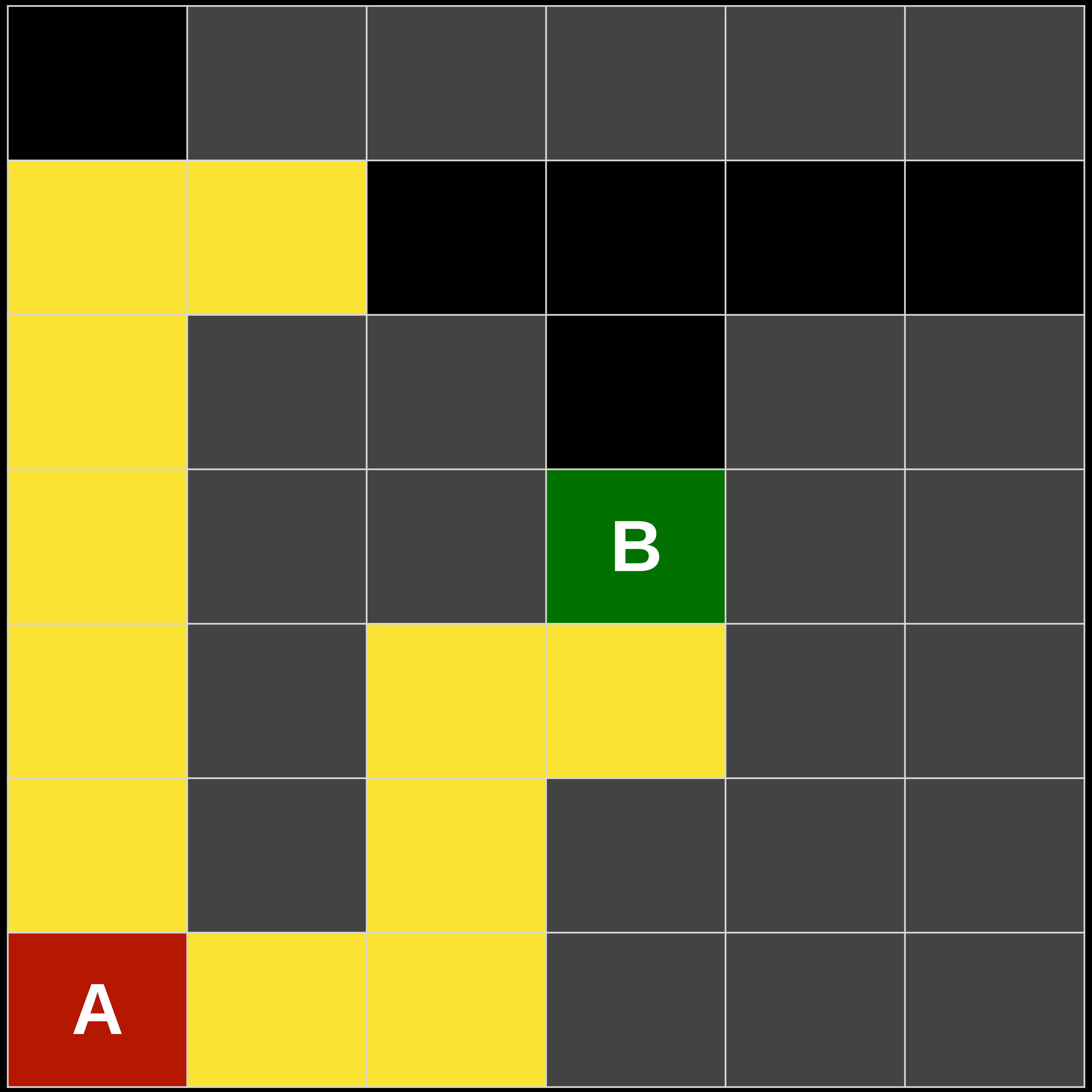
Depth-First Search



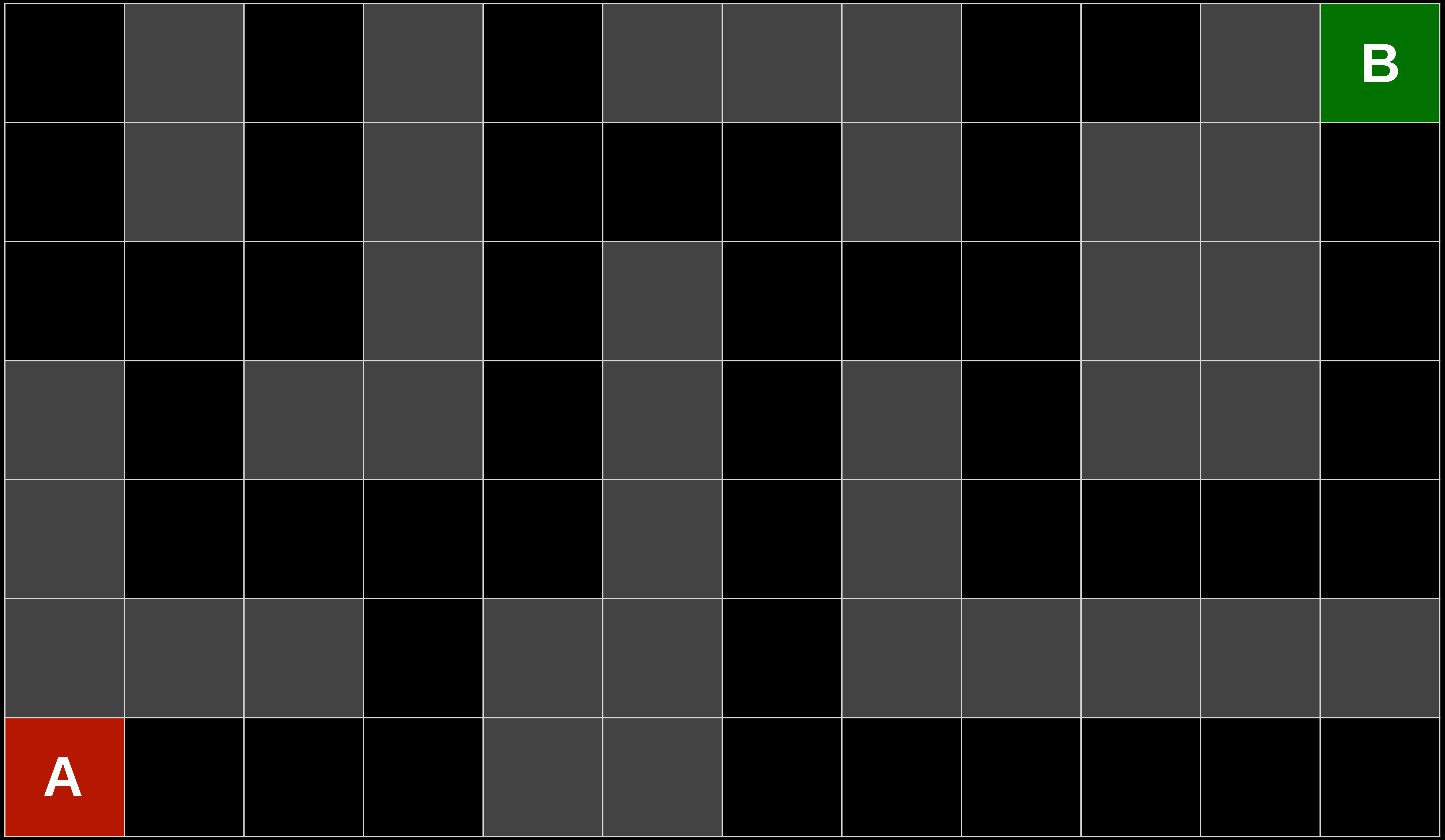
Breadth-First Search



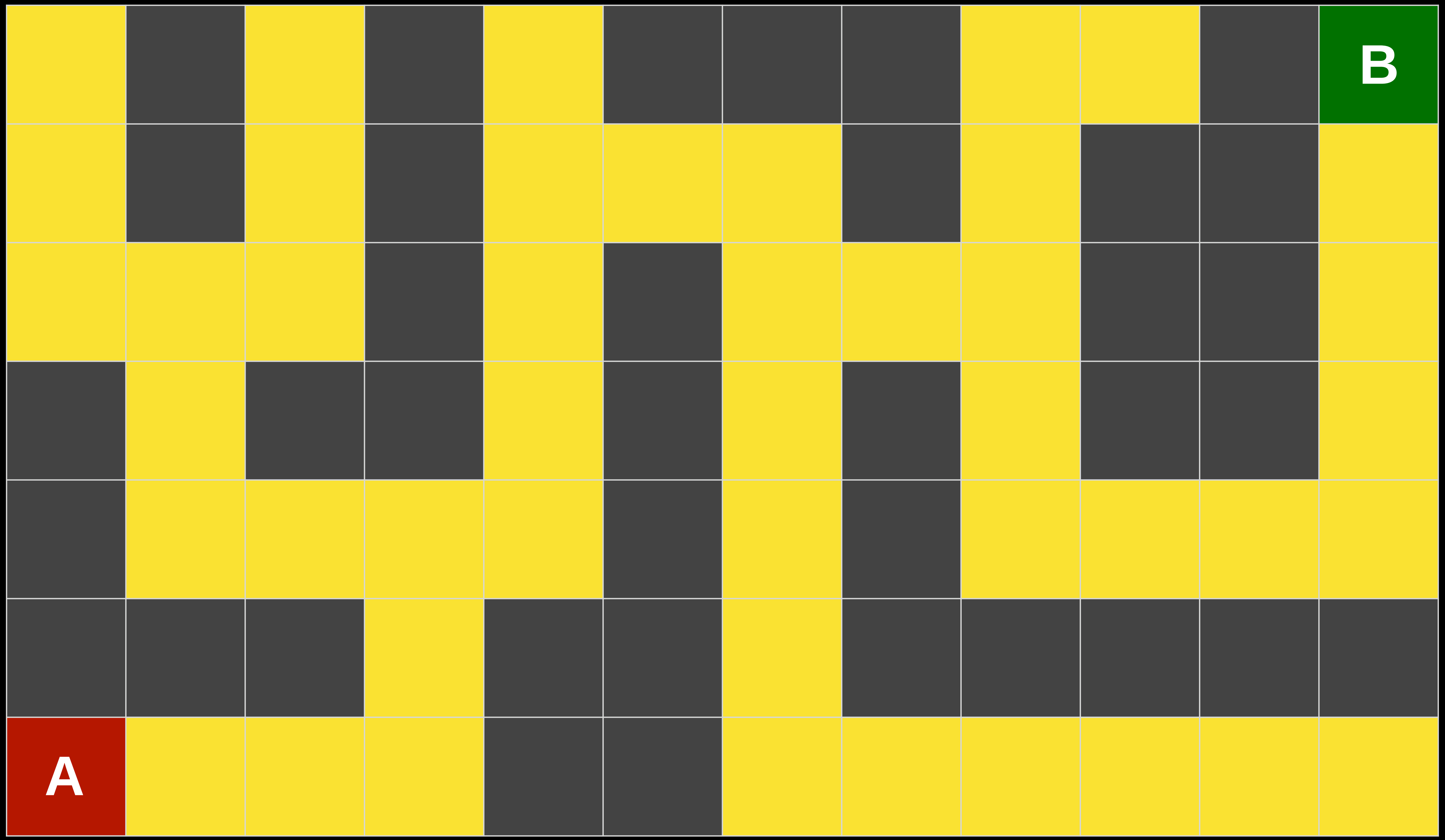
Breadth-First Search



Breadth-First Search



Breadth-First Search



uninformed search

search strategy that uses no problem-specific knowledge

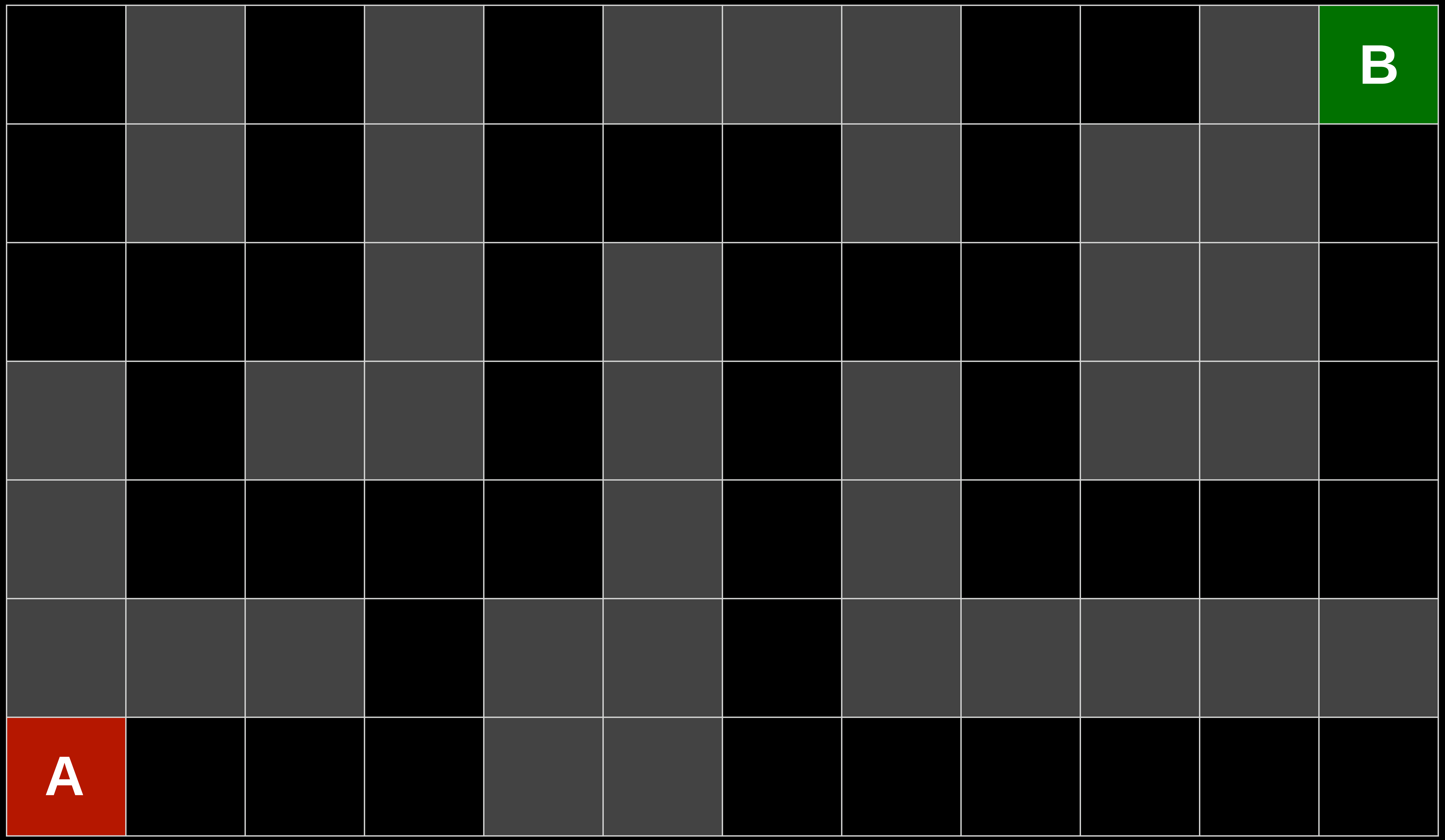
informed search

search strategy that uses problem-specific knowledge to find solutions more efficiently

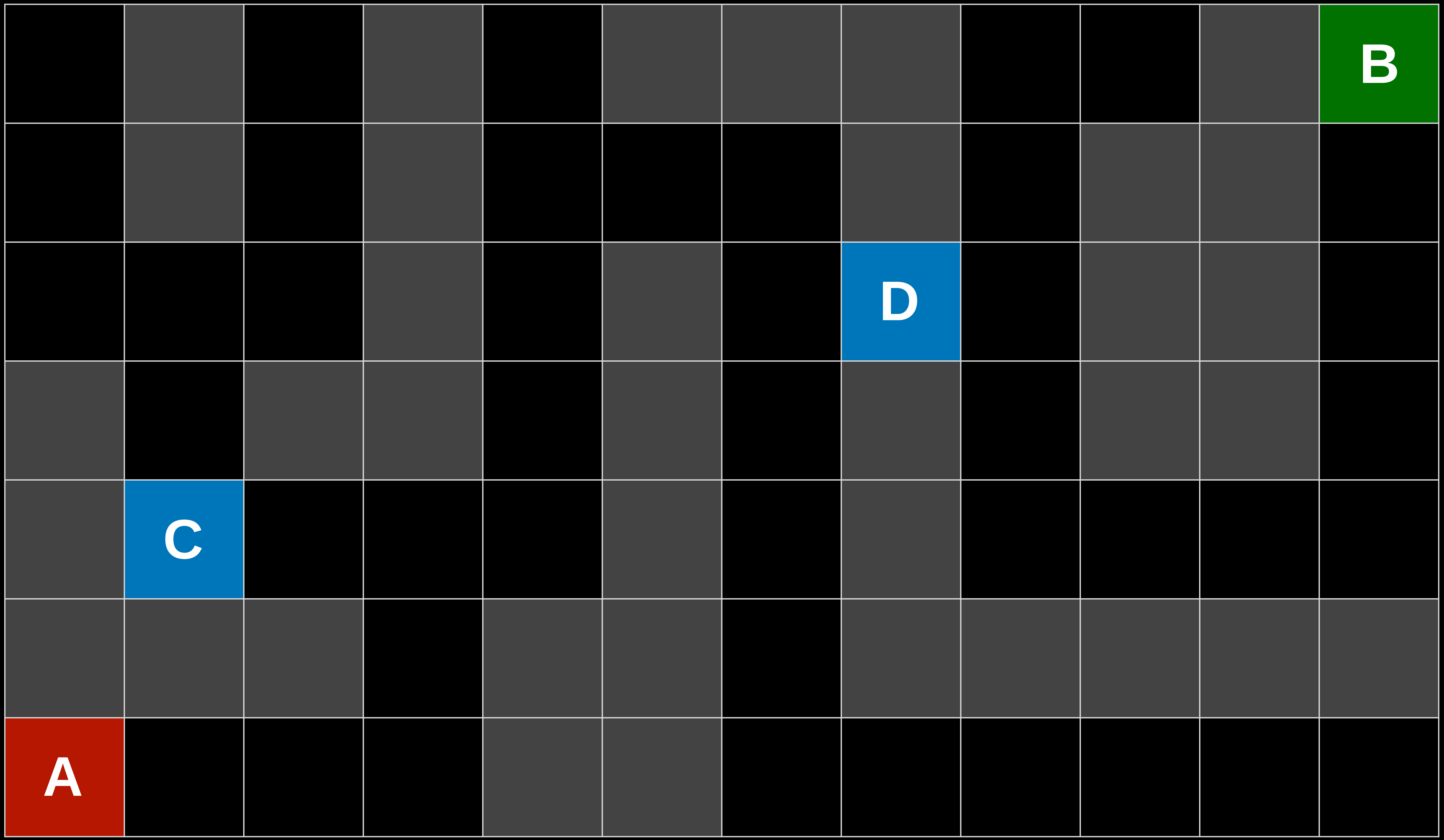
greedy best-first search

search algorithm that expands the node that is closest to the goal, as estimated by a heuristic function $h(n)$

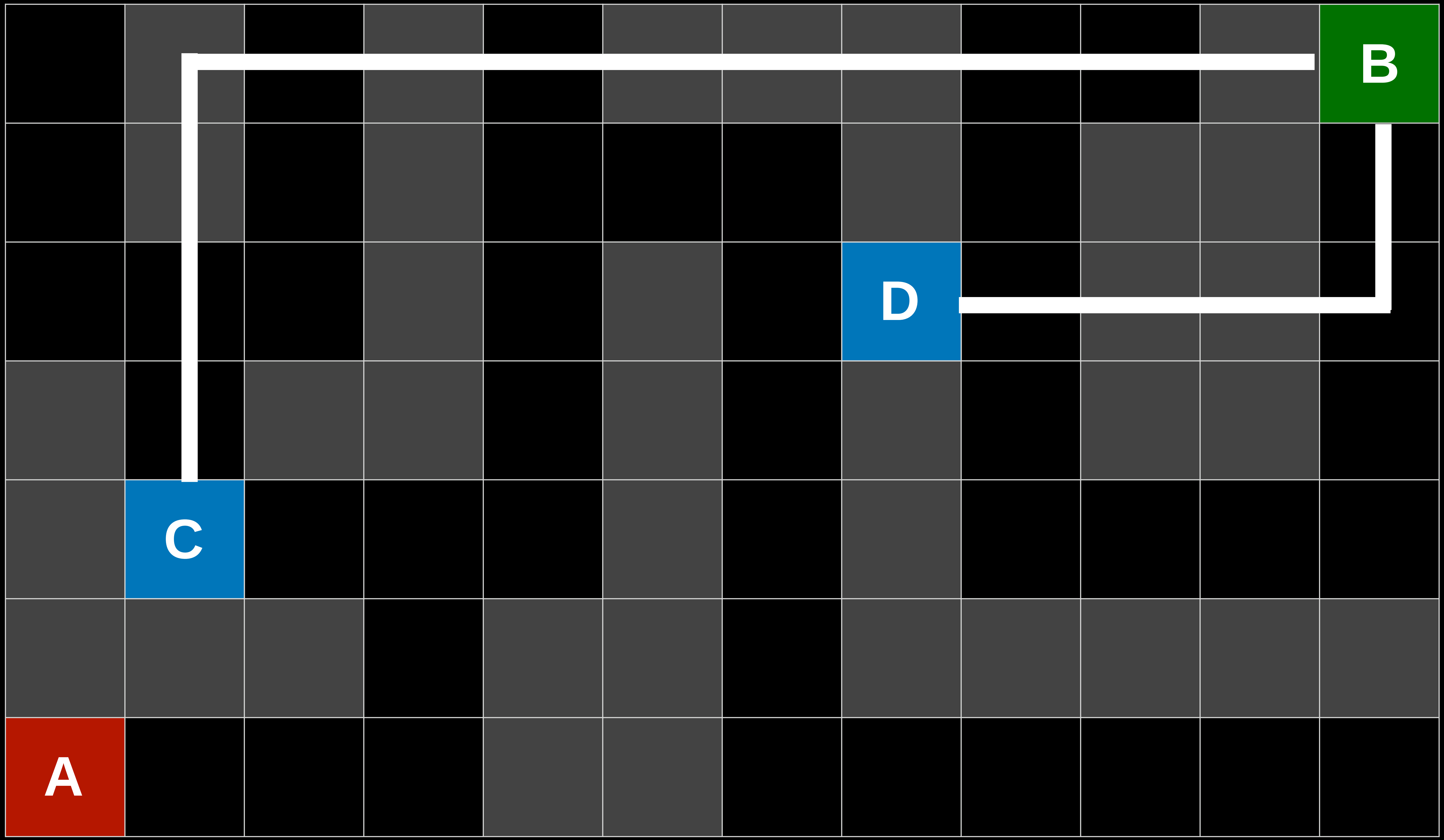
Heuristic function?



Heuristic function?



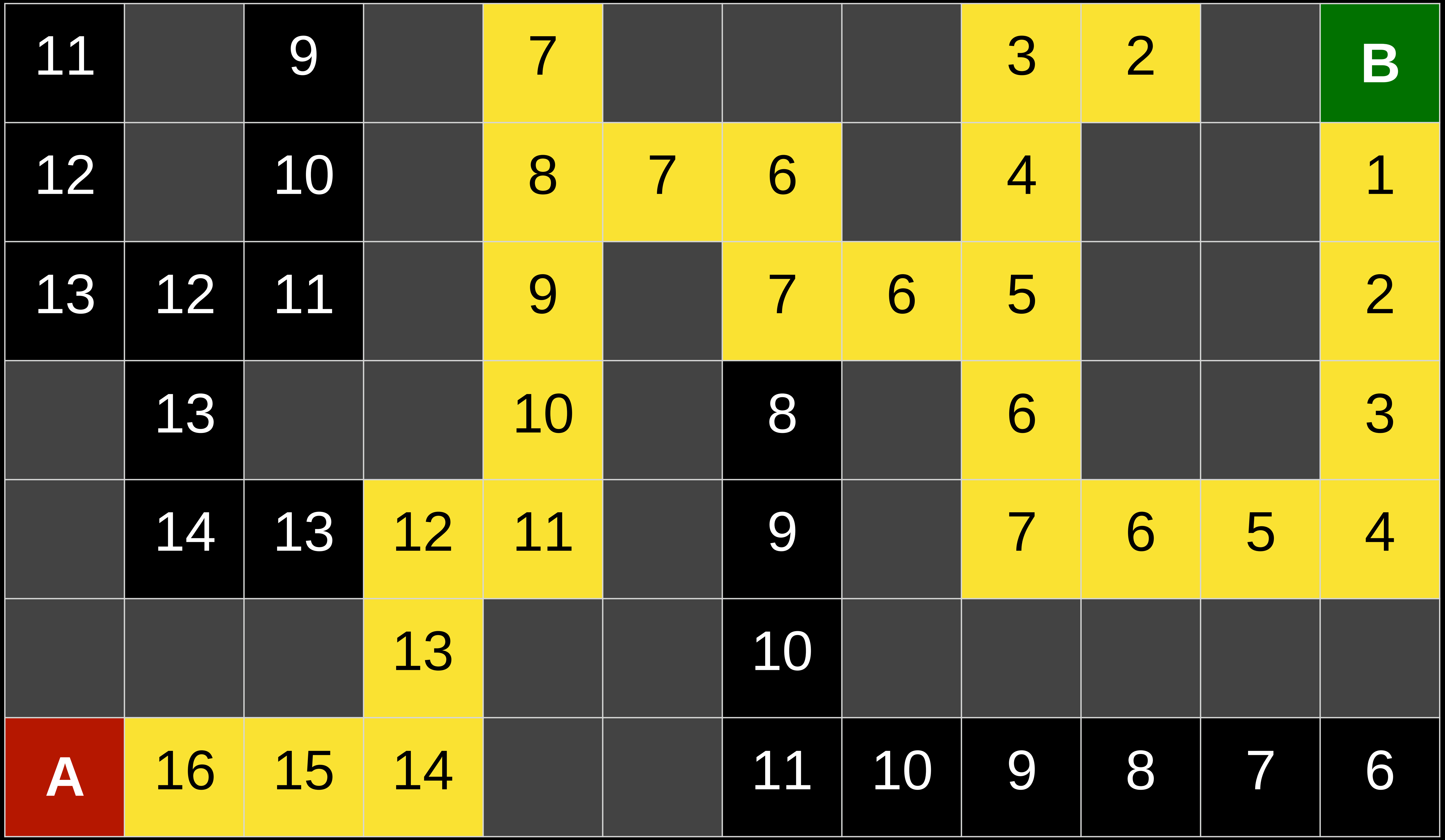
Heuristic function? Manhattan distance.



Greedy Best-First Search

11		9		7				3	2		B
12		10		8	7	6		4			1
13	12	11		9		7	6	5			2
	13			10		8		6			3
	14	13	12	11		9		7	6	5	4
			13			10					
A	16	15	14			11	10	9	8	7	6

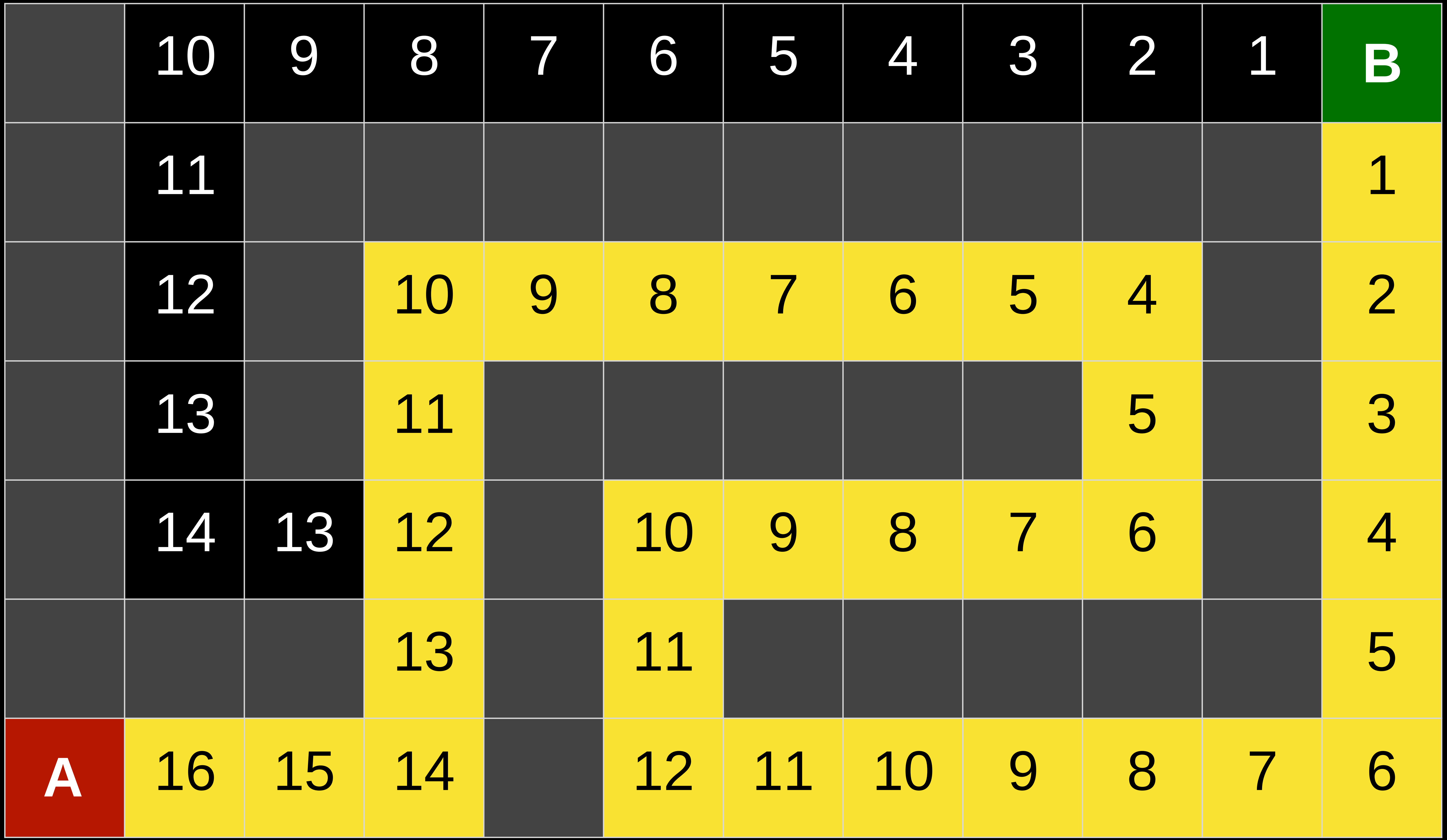
Greedy Best-First Search



Greedy Best-First Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		10	9	8	7	6	5	4		2
	13		11						5		3
	14	13	12		10	9	8	7	6		4
			13		11						5
A	16	15	14		12	11	10	9	8	7	6

Greedy Best-First Search



Greedy Best-First Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		10	9	8	7	6	5	4		2
	13		11						5		3
	14	13	12		10	9	8	7	6		4
			13		11						5
A	16	15	14		12	11	10	9	8	7	6

A* search

search algorithm that expands node with lowest value of $g(n) + h(n)$

$g(n)$ = cost to reach node
$h(n)$ = estimated cost to goal

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		10	9	8	7	6	5	4		2
	13		11						5		3
	14	13	12		10	9	8	7	6		4
			13		11						5
A	16	15	14		12	11	10	9	8	7	6

A* Search

	11+10	12+9	13+8	14+7	15+6	16+5	17+4	18+3	19+2	20+1	B
	10+11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* search

optimal if

- $h(n)$ is admissible (never overestimates the true cost), and
- $h(n)$ is consistent (for every node n and successor n' with step cost c, $h(n) \leq h(n') + c$)

Adversarial Search

	O	X
O	X	X
O	X	

	O	X
O	X	X
O	X	

Minimax

O	X	X
O	O	
O	X	X

-1

X	O	X
O	O	X
X	X	O

0

O		X
	X	O
X	O	X

1

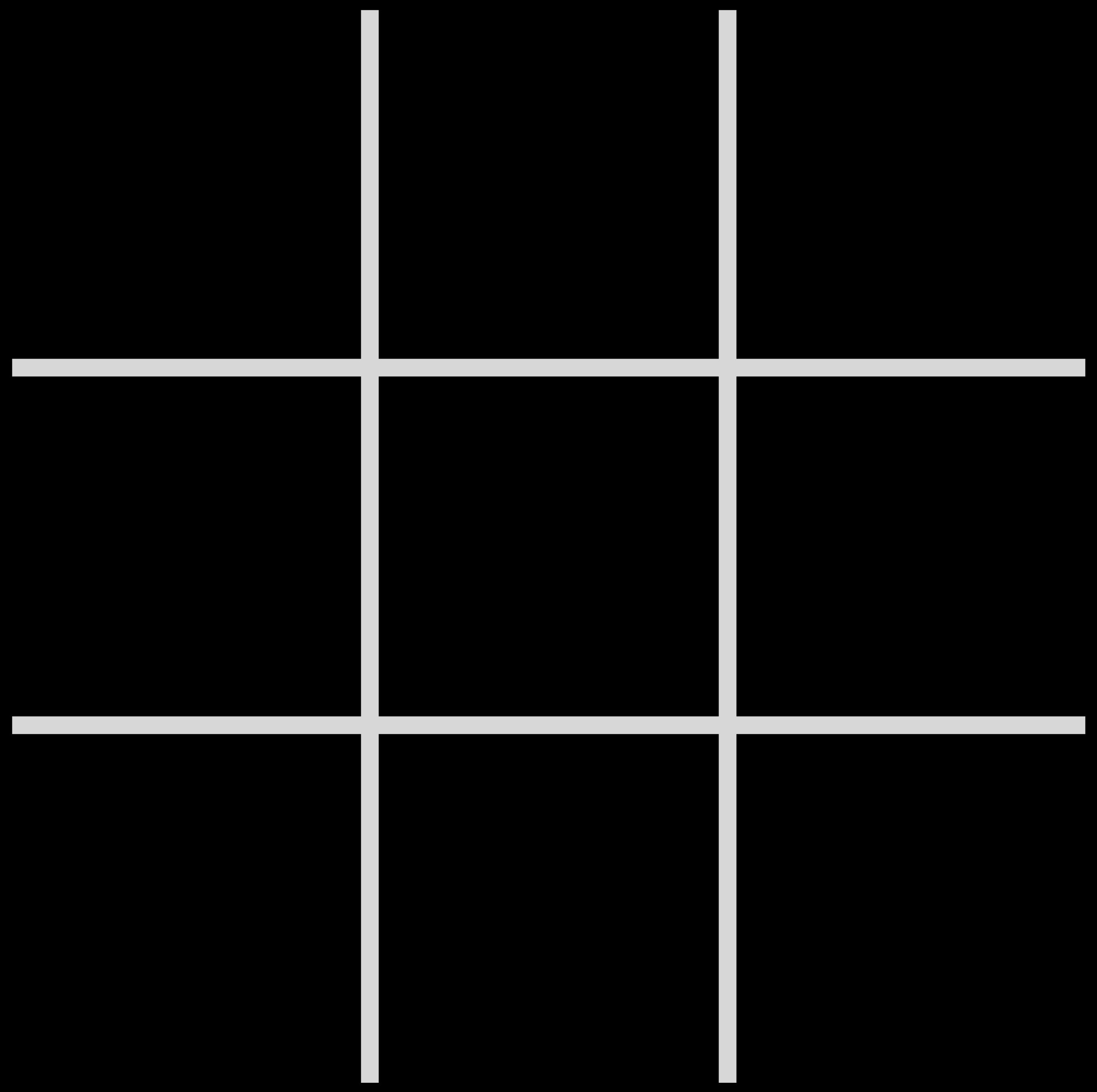
Minimax

- MAX (X) aims to maximize score.
- MIN (O) aims to minimize score.

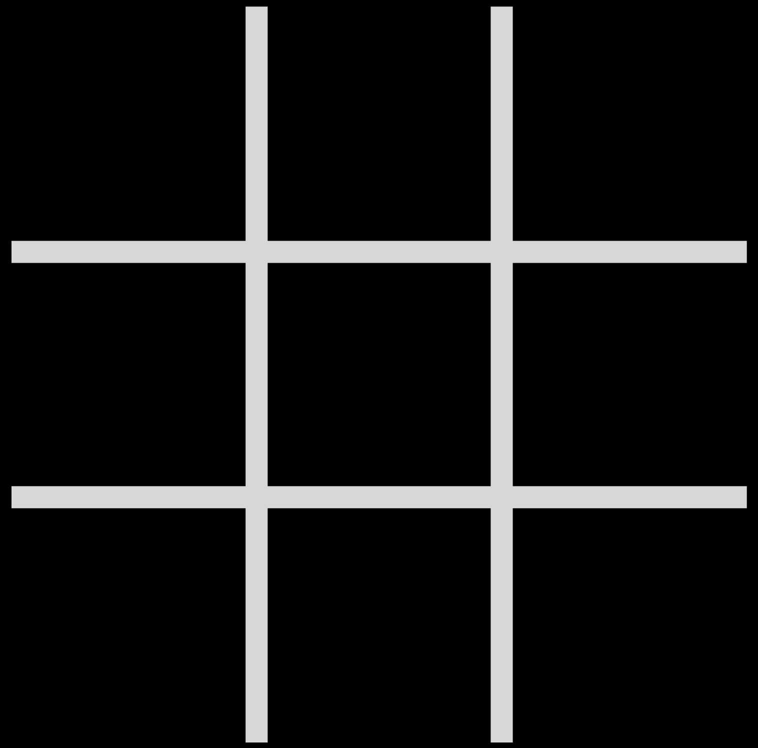
Game

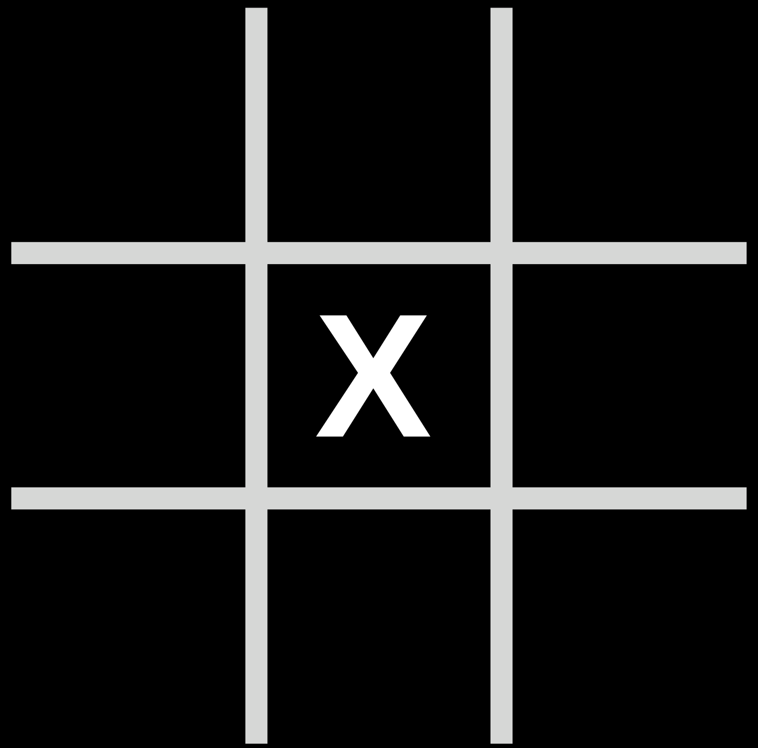
- S_0 : initial state
- $\text{P}\small{\text{LAYER}}(s)$: returns which player to move in state s
- $\text{A}\small{\text{CTIONS}}(s)$: returns legal moves in state s
- $\text{R}\small{\text{ESULT}}(s, a)$: returns state after action a taken in state s
- $\text{T}\small{\text{ERMINAL}}(s)$: checks if state s is a terminal state
- $\text{U}\small{\text{TILITY}}(s)$: final numerical value for terminal state s

Initial State

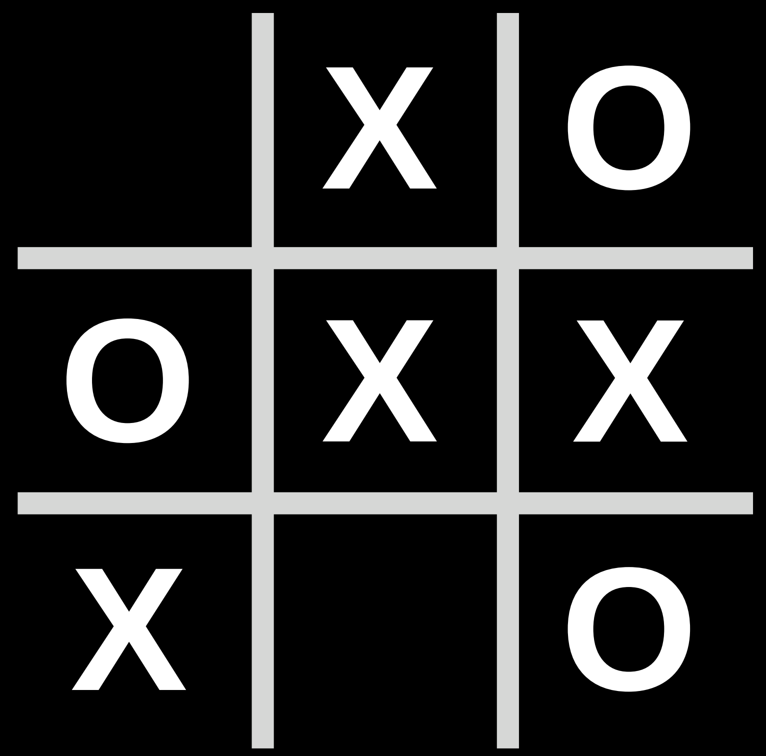


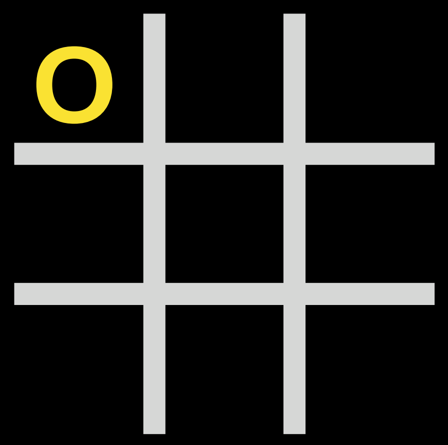
PLAYER(s)

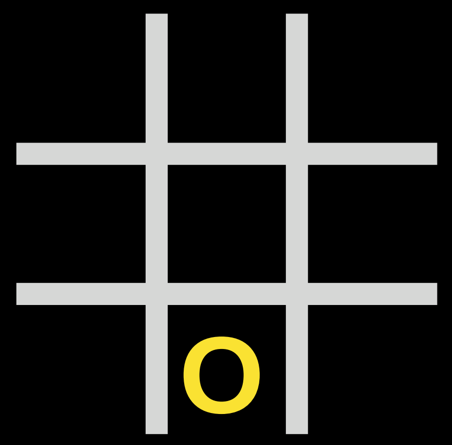
PLAYER() = **X**

PLAYER() = **O**

ACTIONS(s)

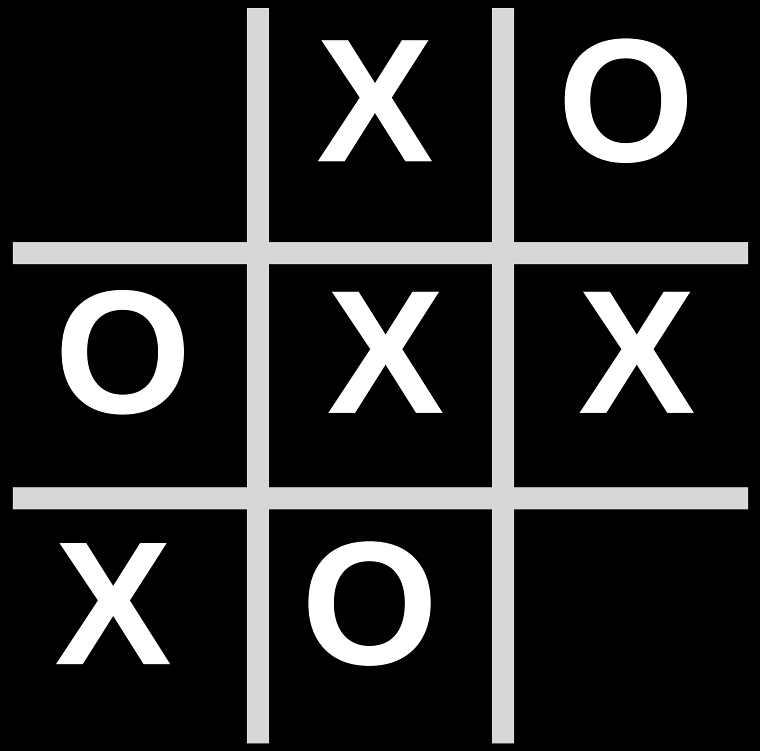
ACTIONS(

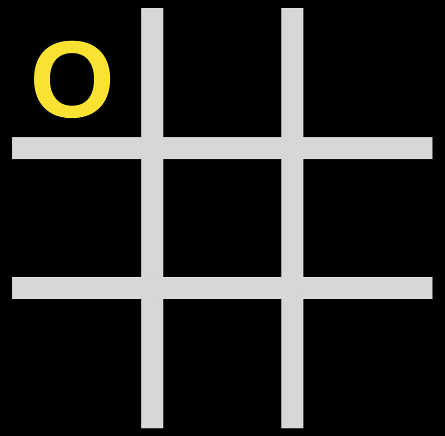
) = { 

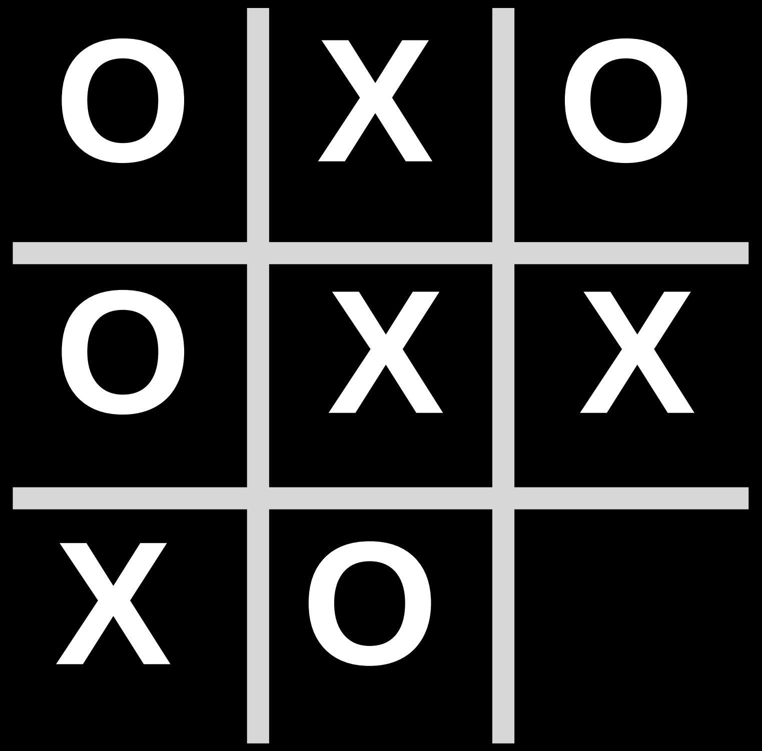
, 

}

$\textsc{Result}(s, a)$

$\textsc{Result}($ 

$,$ 

$) =$ 

$$\textsc{Terminal}(s)$$

$$\textsc{Terminal}\left(\begin{array}{c|c|c} \text{O} & & \\ \hline \text{O} & \text{X} & \\ \hline \text{X} & \text{O} & \text{X} \end{array}\right) = \text{false}$$

$$\textsc{Terminal}\left(\begin{array}{c|c|c} \text{O} & & \text{X} \\ \hline \text{O} & \text{X} & \\ \hline \text{X} & \text{O} & \text{X} \end{array}\right) = \text{true}$$

$$\textsc{Utility}(s)$$

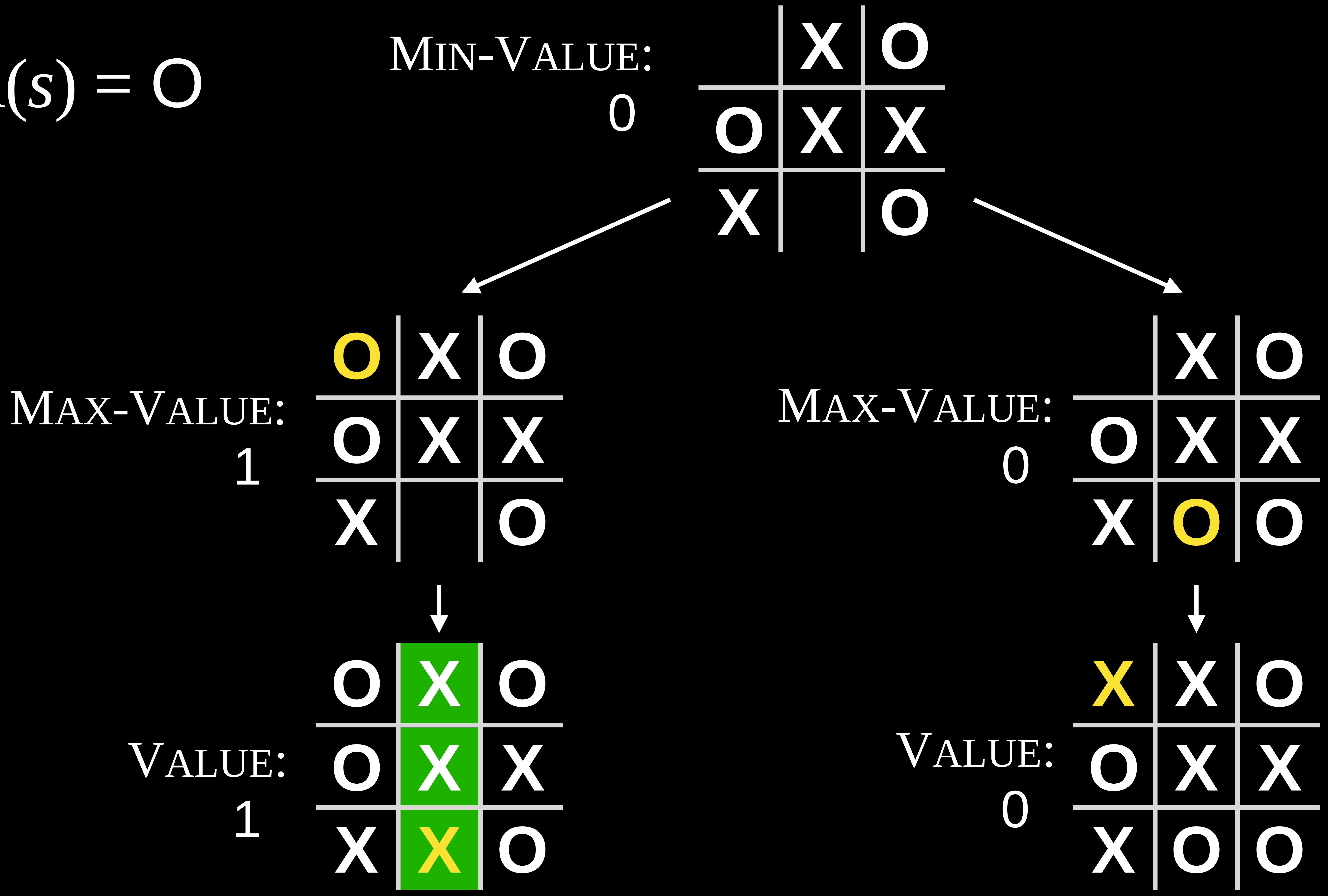
$$\textsc{Utility}\left(\begin{array}{c|c|c} \text{O} & & \text{X} \\ \hline \text{O} & \text{X} & \\ \hline \text{X} & \text{O} & \text{X} \end{array}\right) = 1$$

$$\textsc{Utility}\left(\begin{array}{c|c|c} \text{O} & \text{X} & \text{X} \\ \hline \text{X} & \text{O} & \\ \hline \text{O} & \text{X} & \text{O} \end{array}\right) = \text{-}1$$

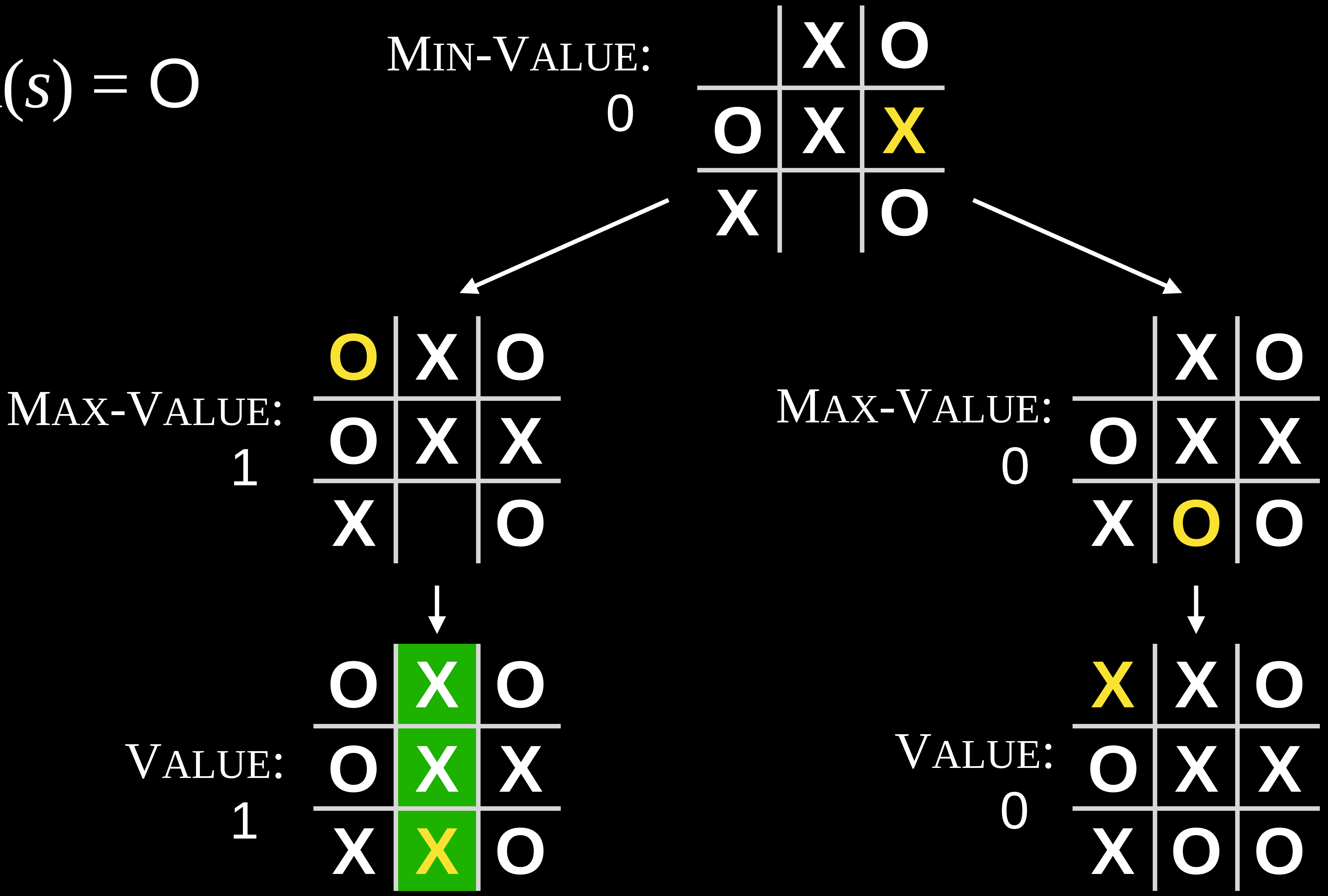
O	X	O
O	X	X
X	X	O

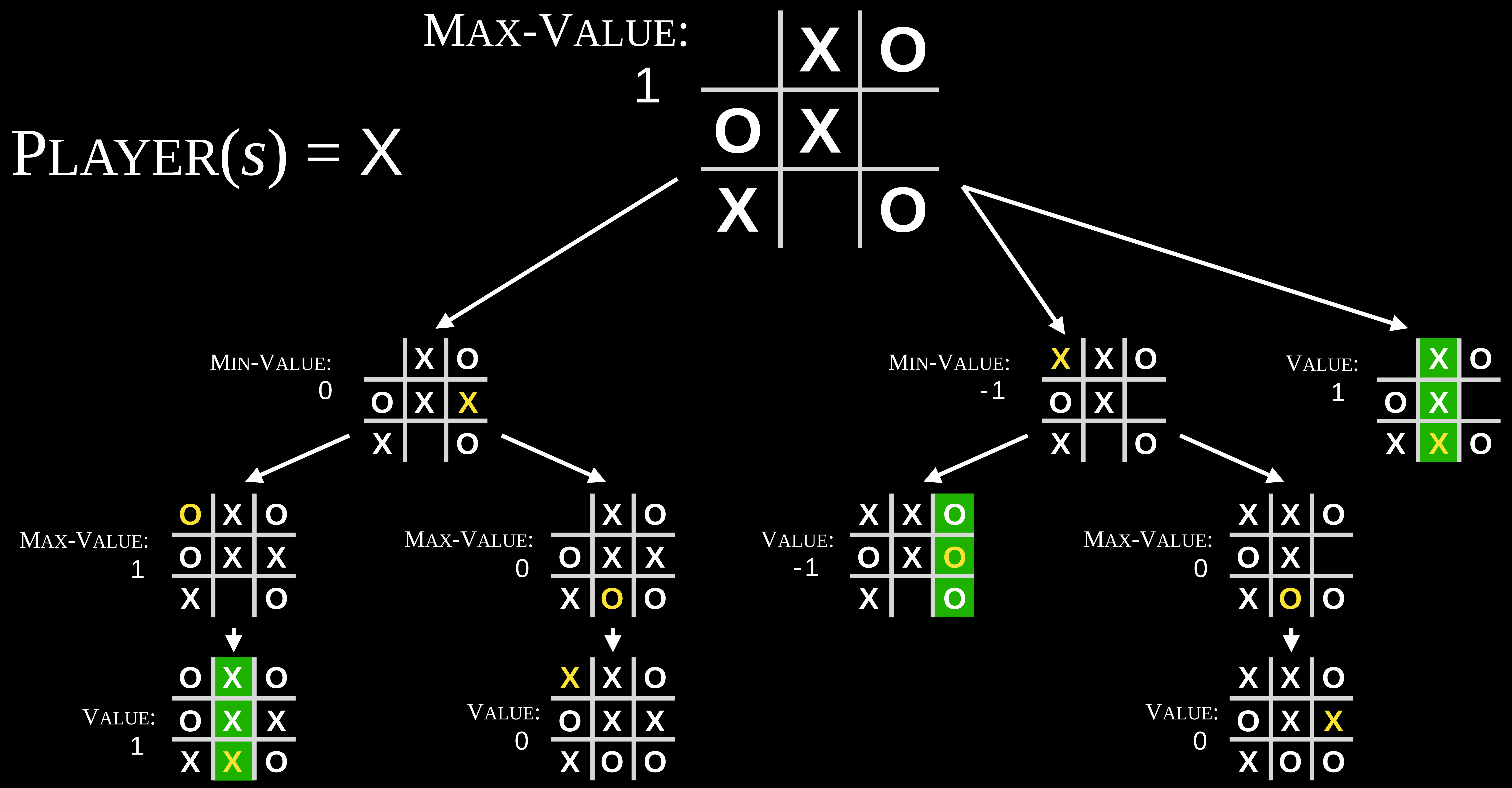
Value: 1

PLAYER(s) = O

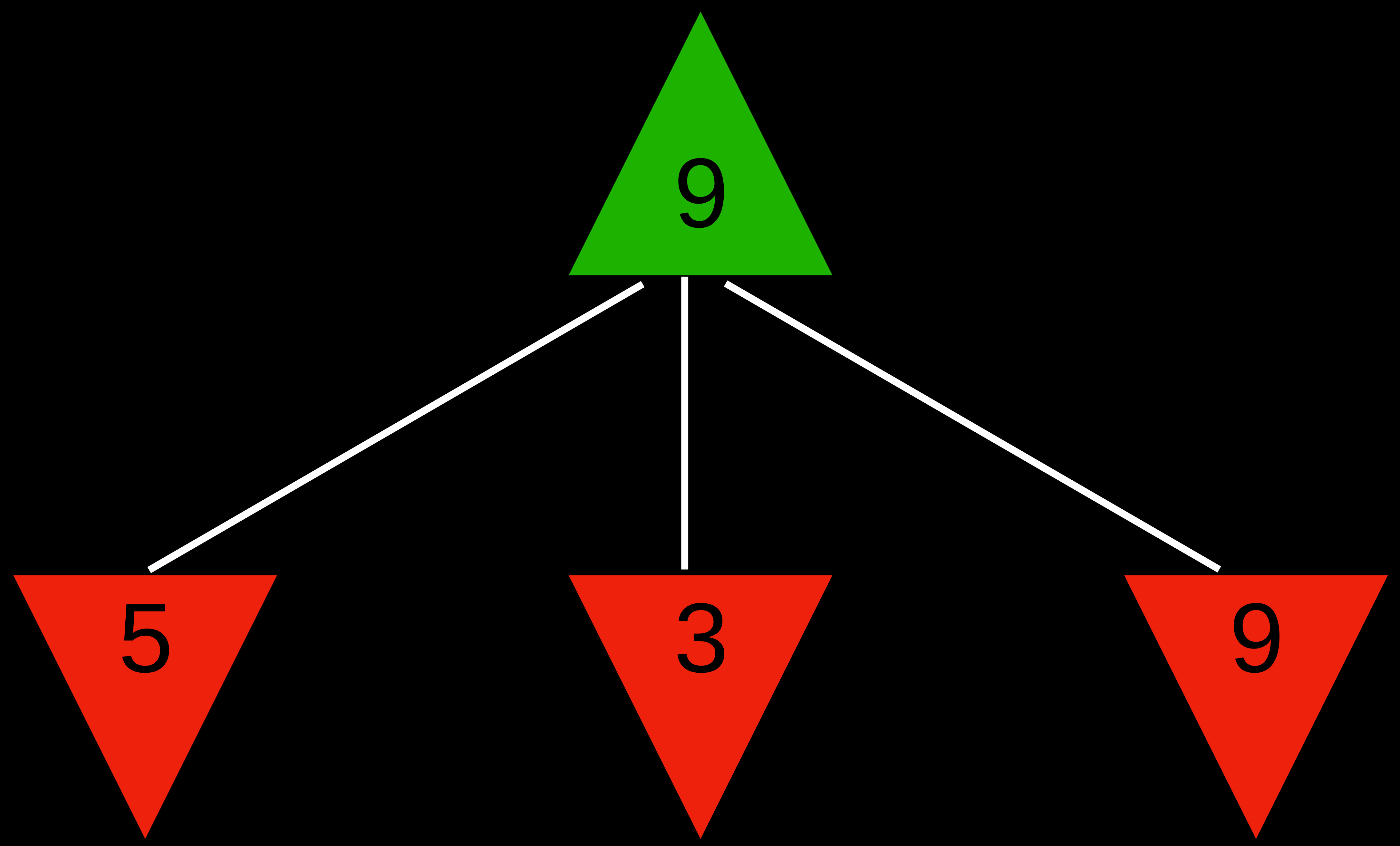


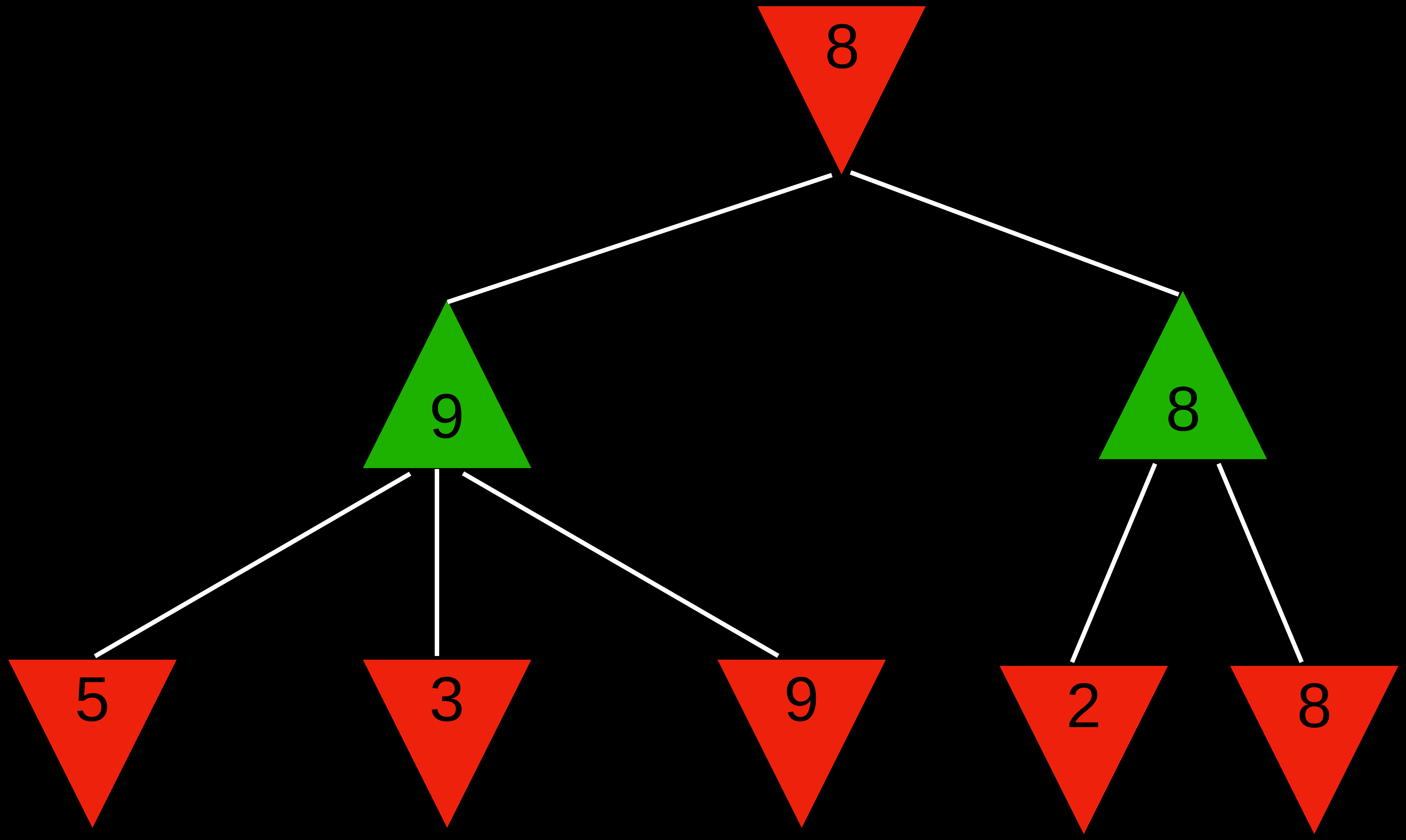
PLAYER(s) = O





PLAYER(s) = X
MAX-VALUE:
1
MIN-VALUE:
0
MIN-VALUE:
-1
VALUE:
1
MAX-VALUE:
1
MAX-VALUE:
0
VALUE:
-1
MAX-VALUE:
0
VALUE:
1
VALUE:
0
VALUE:
0


9
5
3
9


8
9
8
5
3
9
2
8

Minimax

- Given a state s:
  - MAX picks action a in $\text{ACTIONS}(s)$ that produces highest value of $\text{MIN-VALUE}(\text{RESULT}(s, a))$
  - MIN picks action a in $\text{ACTIONS}(s)$ that produces smallest value of $\text{MAX-VALUE}(\text{RESULT}(s, a))$

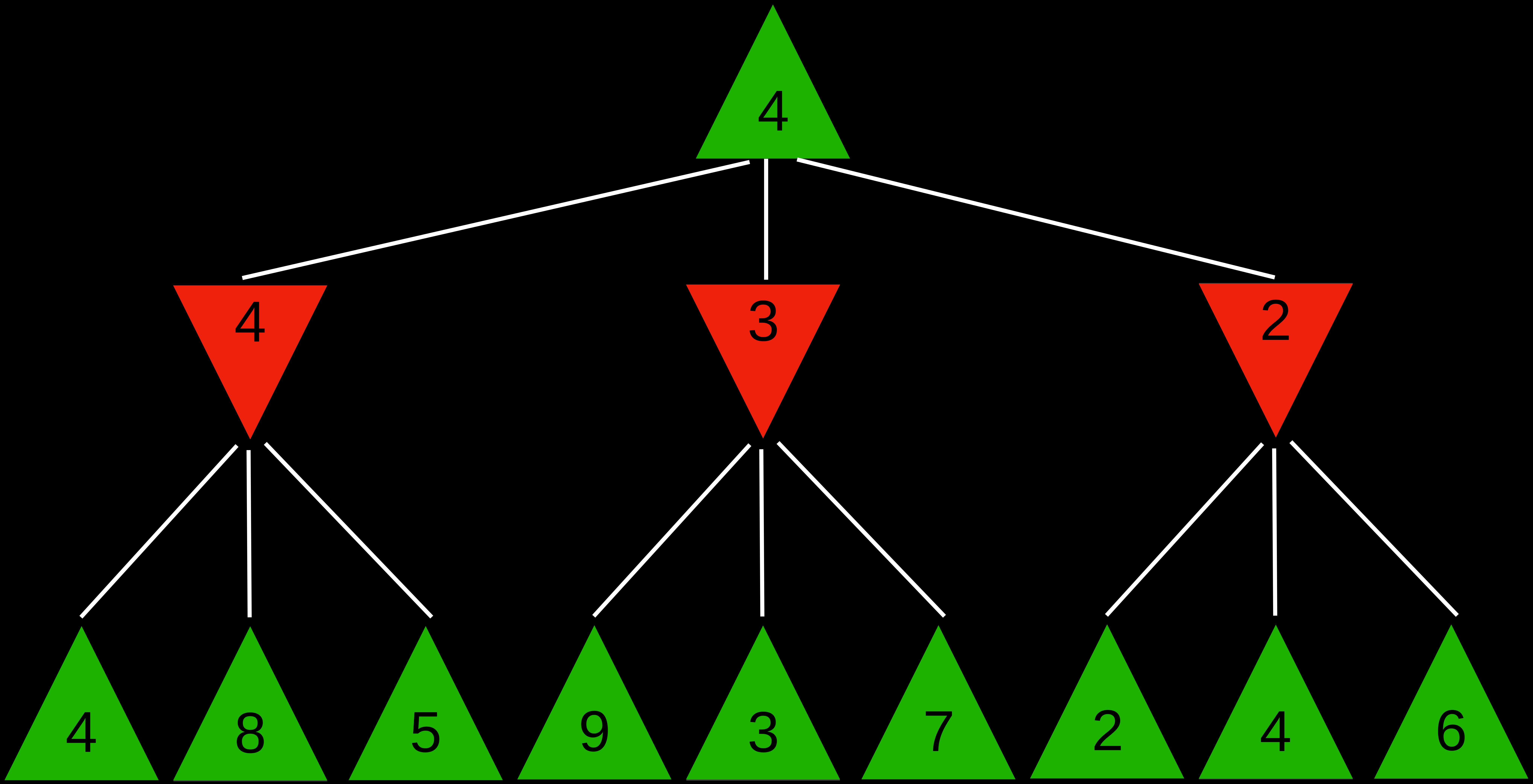
Minimax

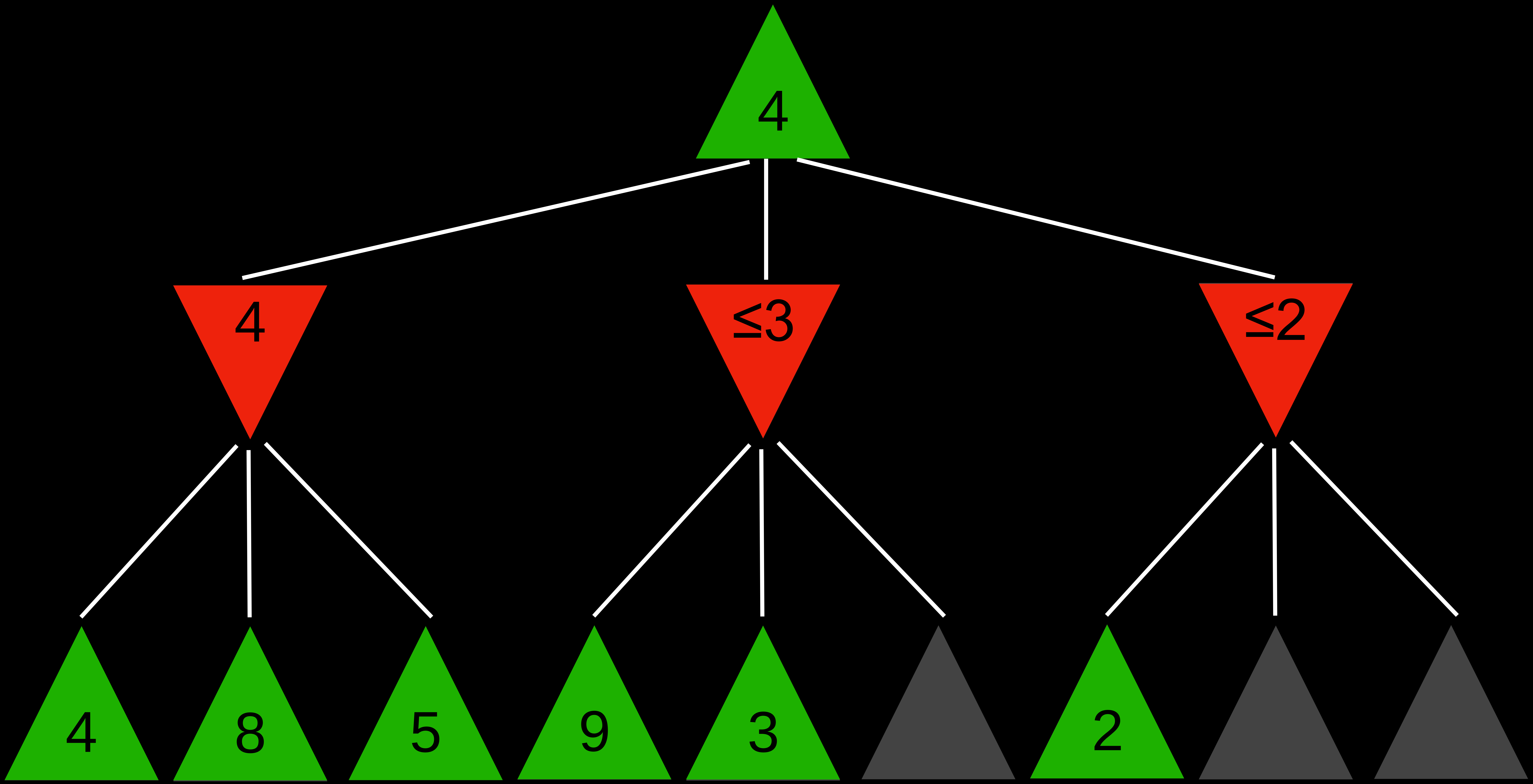
```
function MAX-VALUE(state):
    if TERMINAL(state):
        return UTILITY(state)
    v = -∞
    for action in ACTIONS(state):
        v = MAX(v, MIN-VALUE(RESULT(state, action)))
    return v
```

Minimax

```
function MIN-VALUE(state):
    if TERMINAL(state):
        return UTILITY(state)
    v = ∞
    for action in ACTIONS(state):
        v = MIN(v, MAX-VALUE(RESULT(state, action)))
    return v
```

Optimizations


4
4
3
2
4
8
5
9
3
7
2
4
6


4
4
≤3
≤2
4
8
5
9
3
2

Alpha-Beta Pruning

255,168

total possible Tic-Tac-Toe games

288,000,000,000

total possible chess games
after four moves each

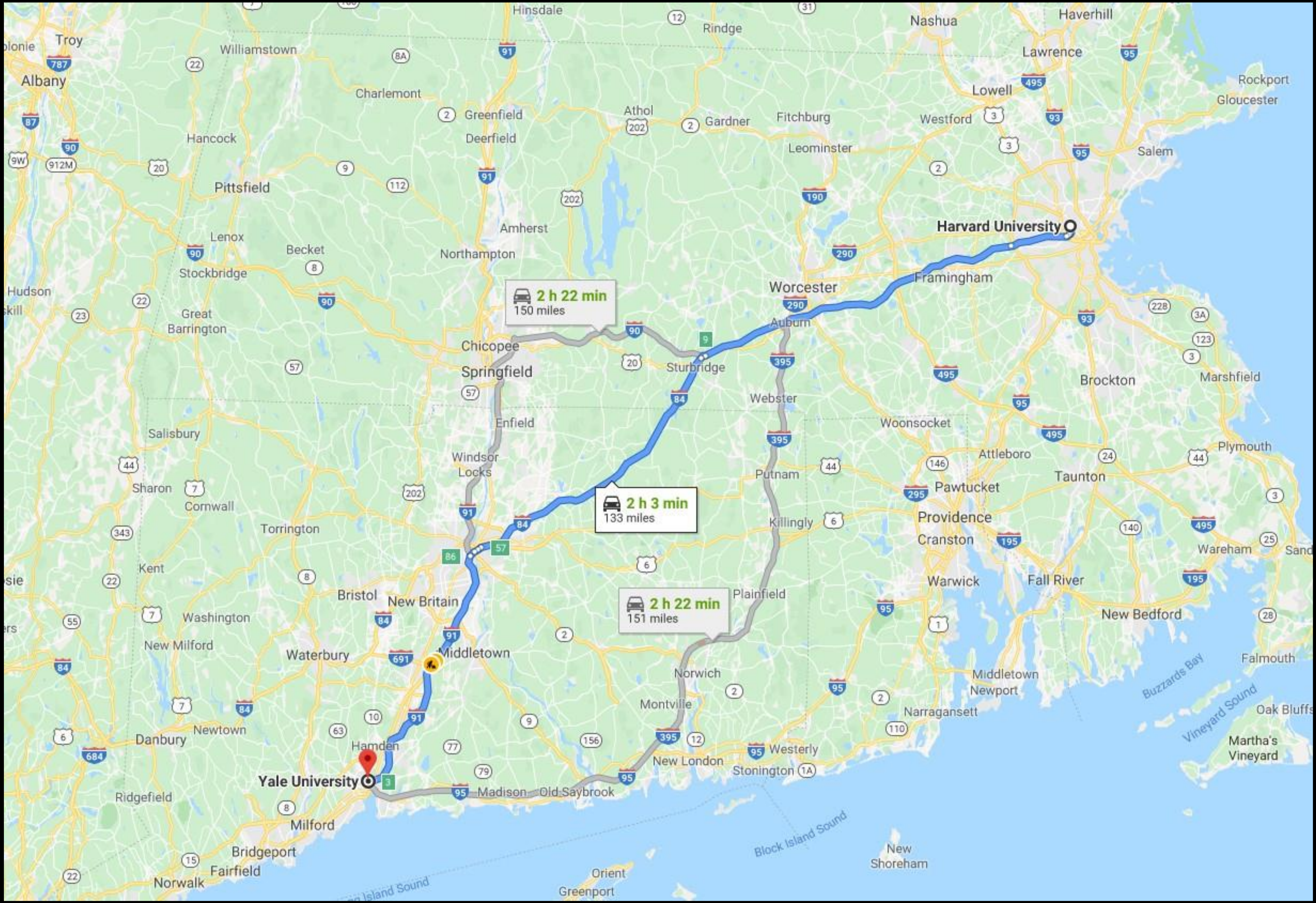
10_{29000}

total possible chess games
(lower bound)

Depth-Limited Minimax

evaluation function

function that estimates the expected utility of the game from a given state

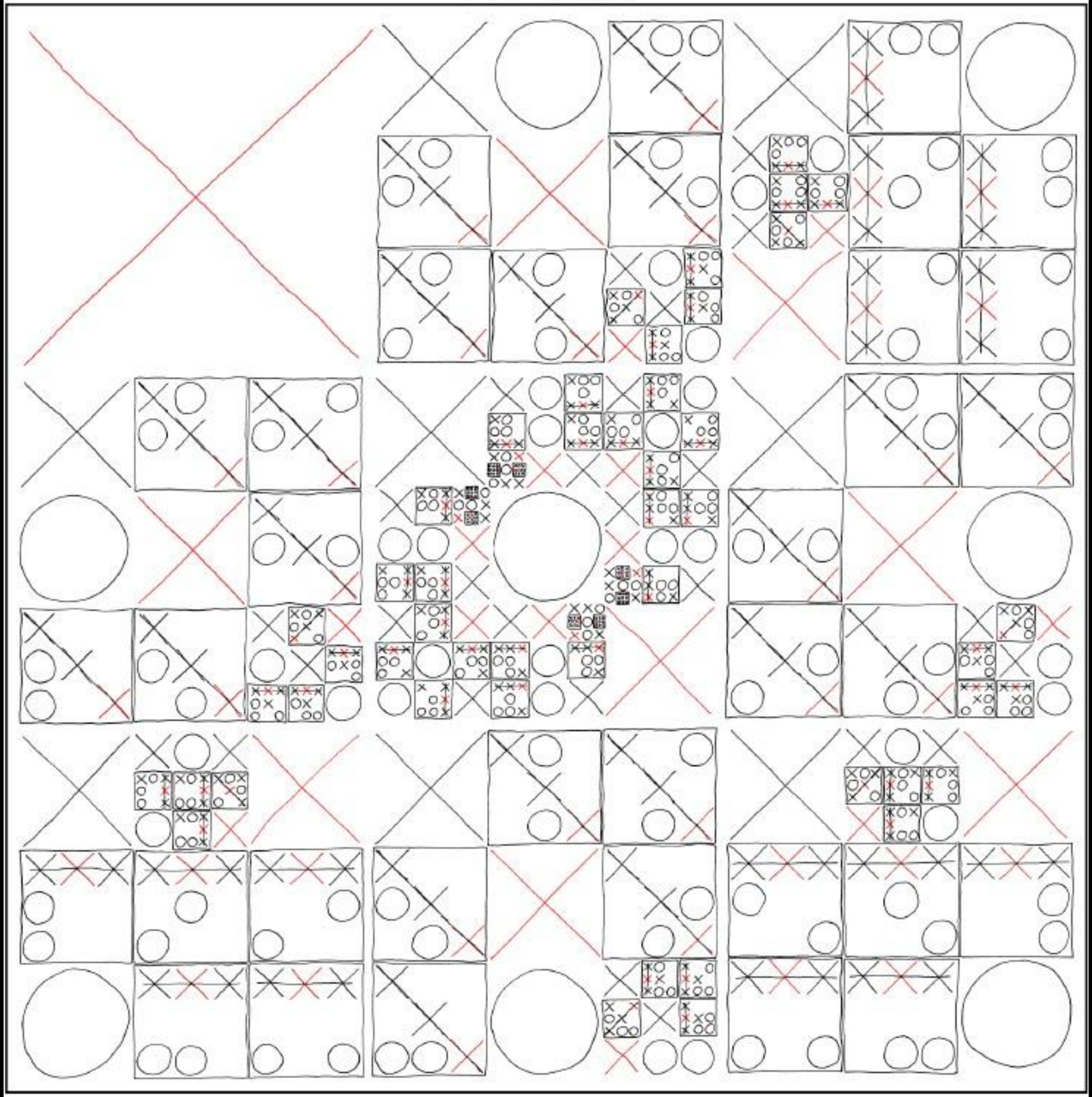


Harvard University
Yale University
2 h 22 min
150 miles
2 h 3 min
133 miles
2 h 22 min
151 miles
Albany
Troy
Williamstown
Hinsdale
Ringde
Nashua
Haverhill
Lawrence
Lowell
Rockport
Gloucester
Charlemont
Greenfield
Deerfield
Athol
Gardner
Fitchburg
Leominster
Westford
Salem
Hancock
Pittsfield
Amherst
Northampton
Lenox
Becket
Stockbridge
Worcester
Framingham
Auburn
Chicopee
Springfield
Sturbridge
Great Barrington
Hudson
Brockton
Marshfield
Webster
Enfield
Salisbury
Woonsocket
Windsor Locks
Putnam
Attleboro
Taunton
Plymouth
Sharon
Cornwall
Torrington
Pawtucket
Providence
Cranston
Killingly
Kent
Warwick
Fall River
Wareham
Bristol
New Britain
Plainfield
Washington
New Bedford
New Milford
Waterbury
Middletown
Norwich
Falmouth
Middletown
Newport
Buzzards Bay
Montville
Narragansett
Vineyard Sound
Oak Bluffs
Martha's Vineyard
Newtown
Danbury
Hamden
New London
Westerly
Stonington
Madison
Old Saybrook
Ridgefield
Milford
Block Island Sound
New Shoreham
Bridgeport
Fairfield
Norwalk
Orient
Greenport

COMPLETE MAP OF OPTIMAL TIC-TAC-TOE MOVES

YOUR MOVE IS GIVEN BY THE POSITION OF THE LARGEST RED SYMBOL ON THE GRID. WHEN YOUR OPPONENT PICKS A MOVE, ZOOM IN ON THE REGION OF THE GRID WHERE THEY WENT. REPEAT.

MAP FOR X:



https://xkcd.com/832/

Thank You!