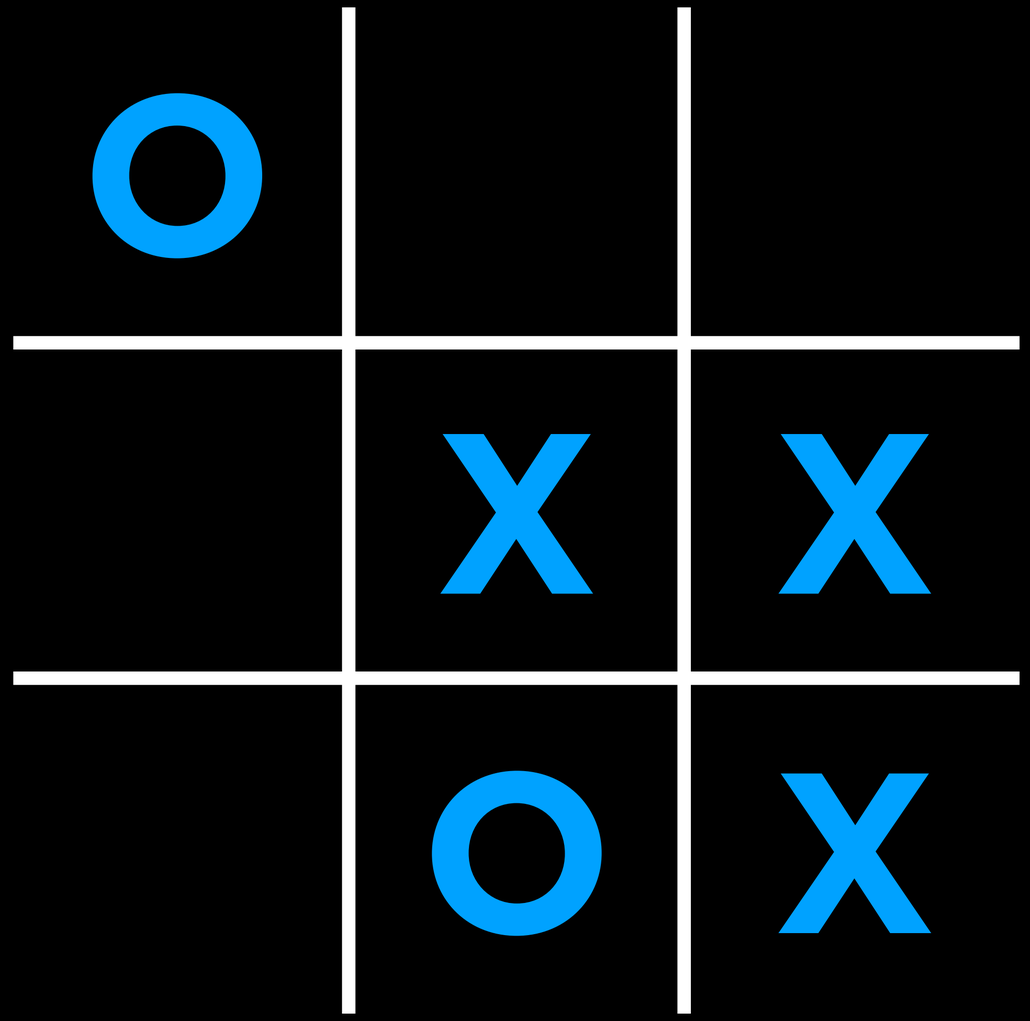
# Introduction to **Artificial Intelligence** with Python

# Artificial Intelligence

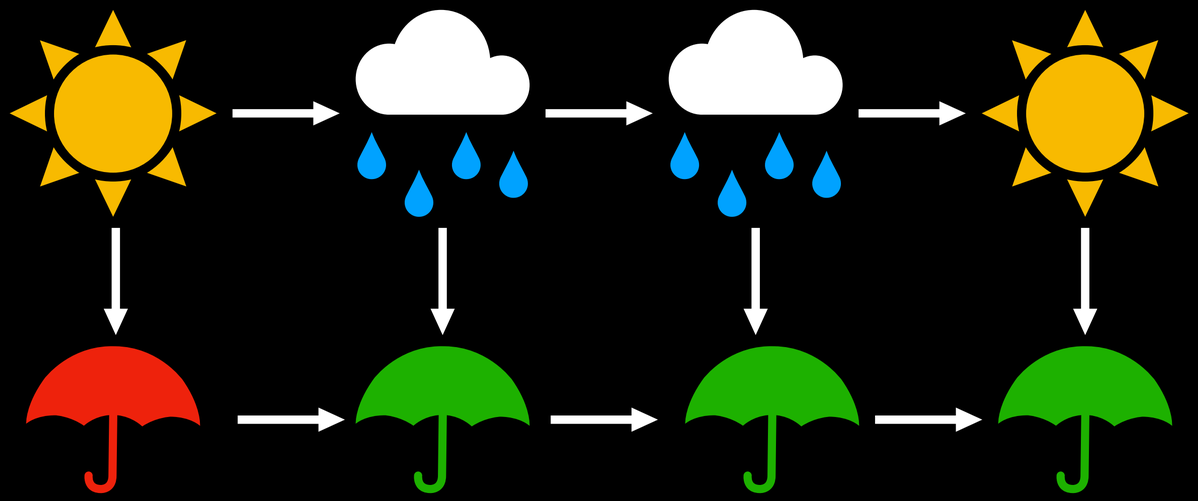
# Search



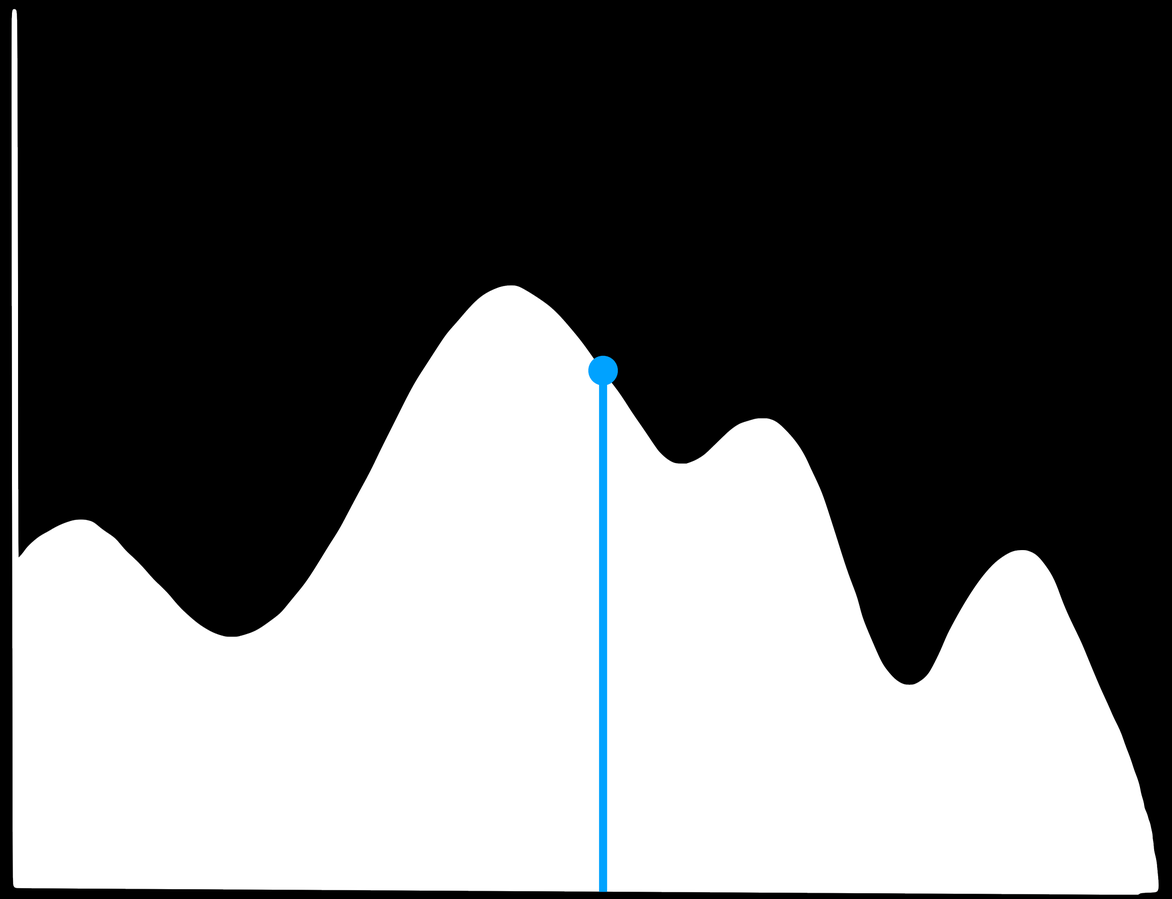
Knowledge

$$\frac{P \to Q \qquad P}{Q}$$

# Uncertainty

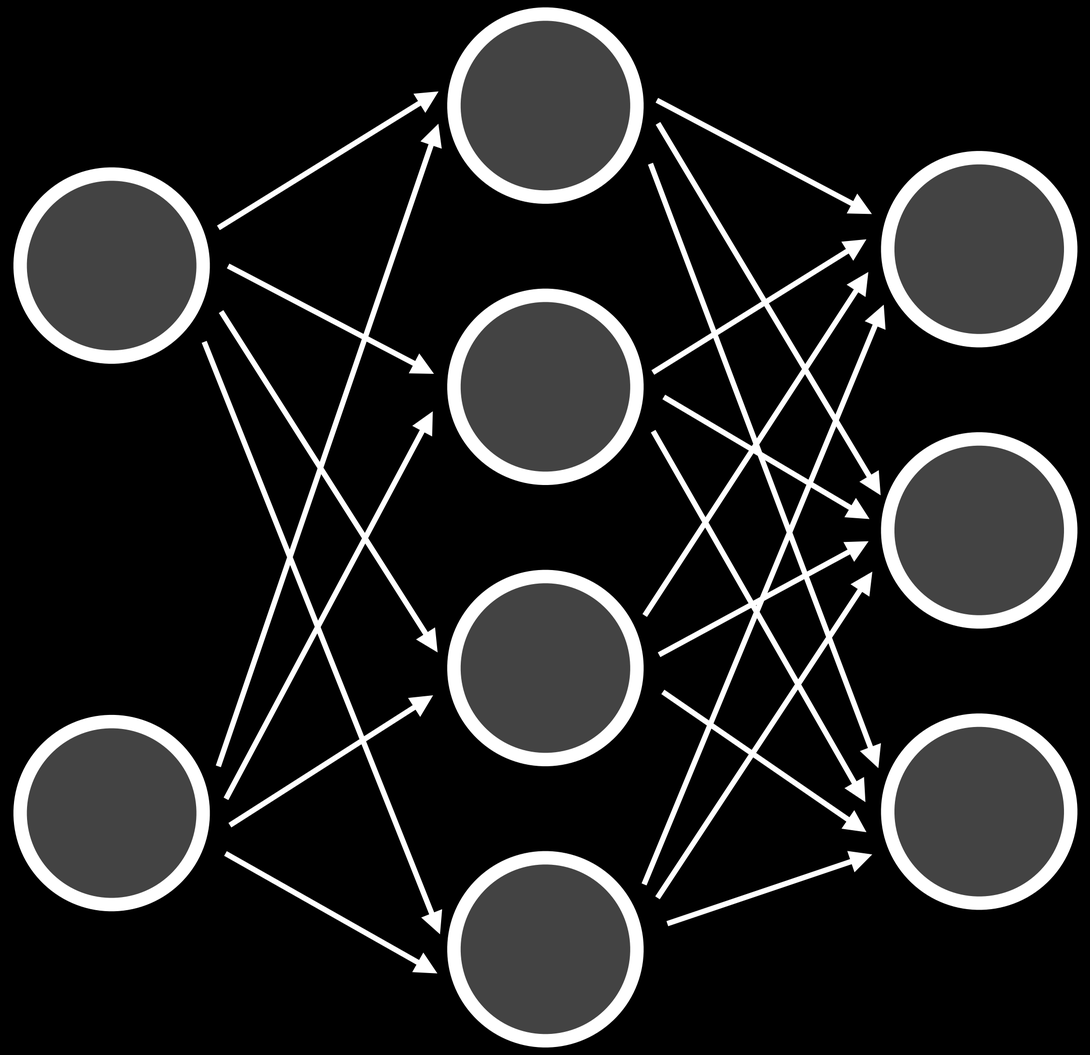


# Optimization

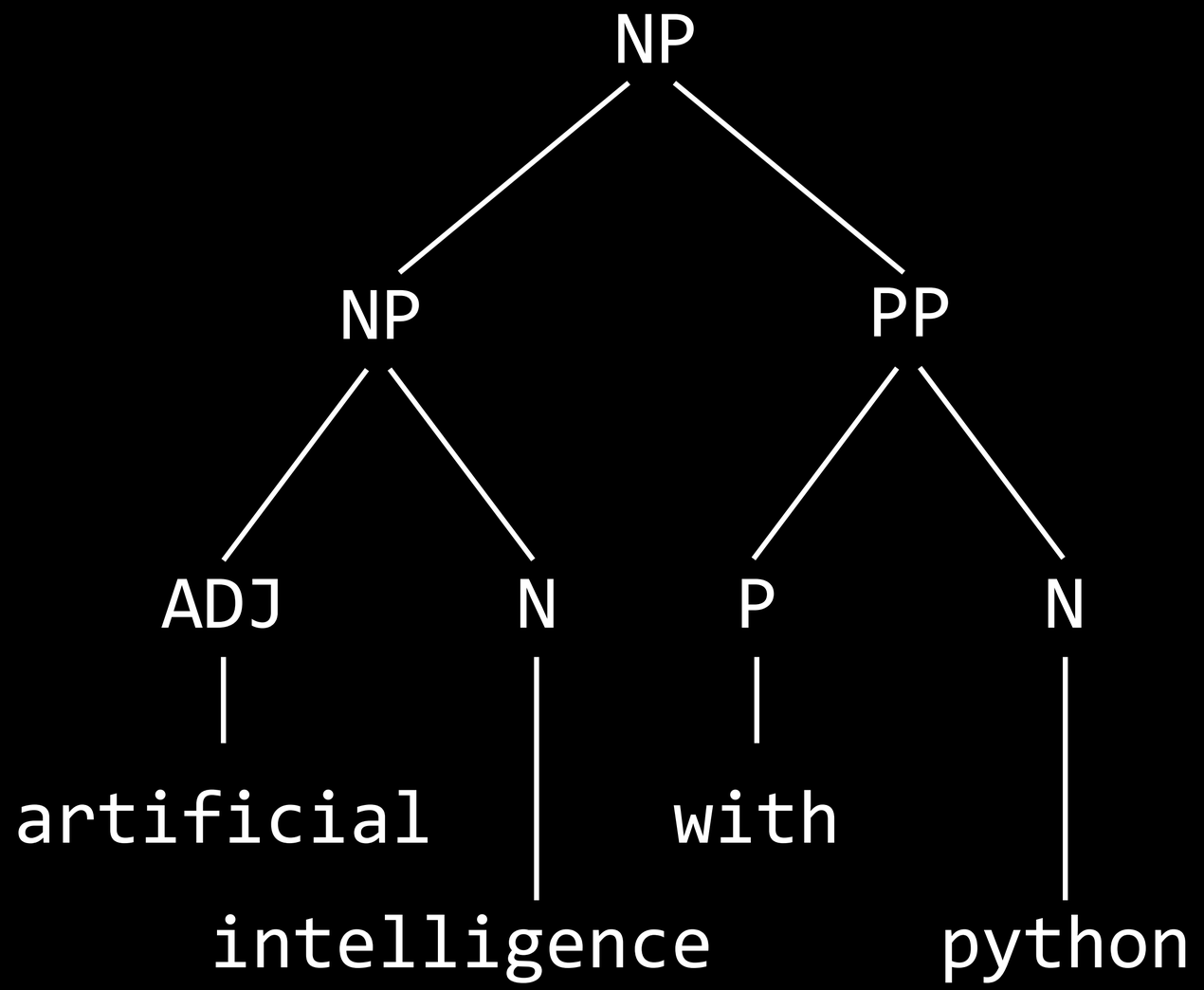


# Learning

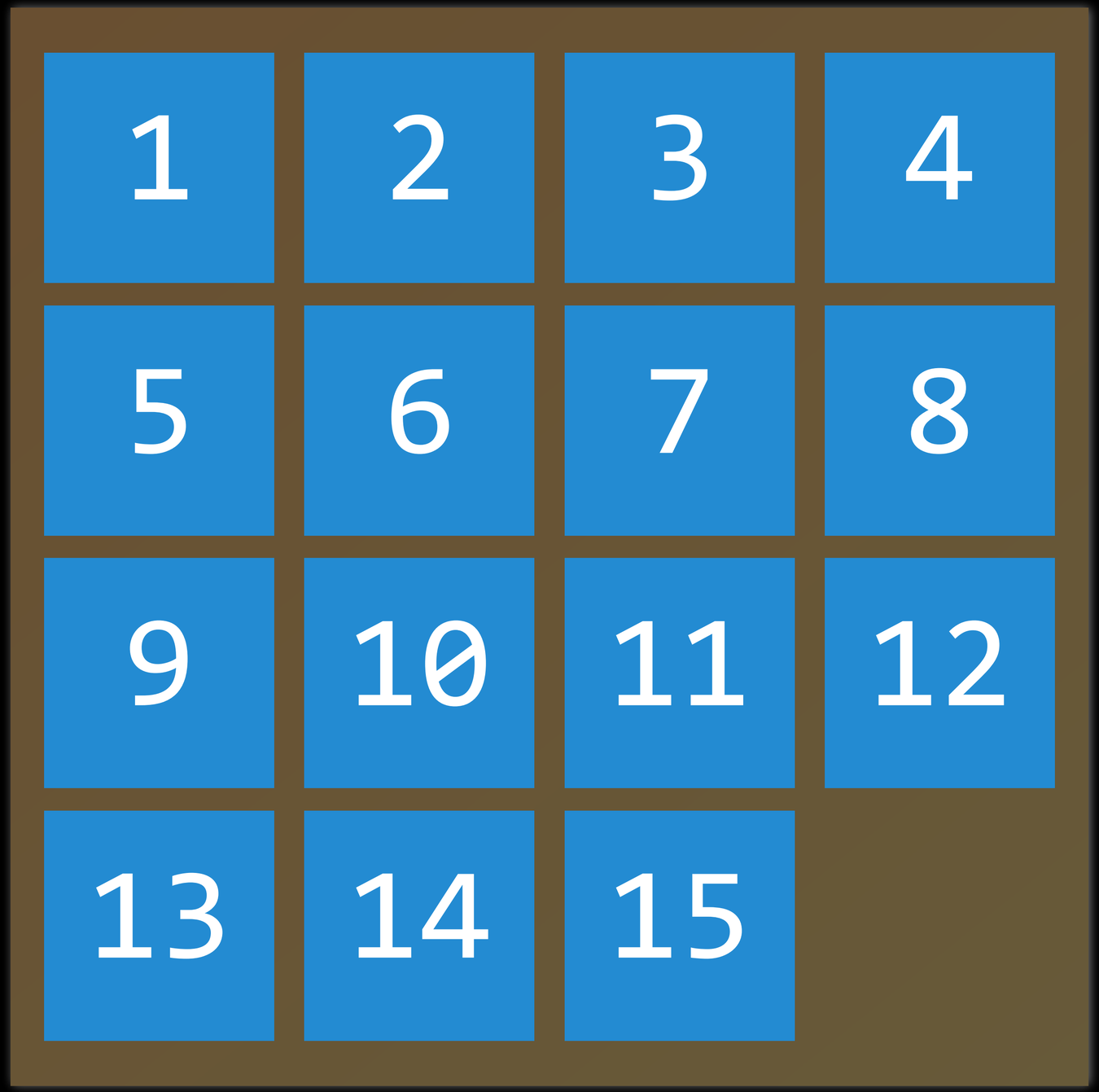
# Neural Networks

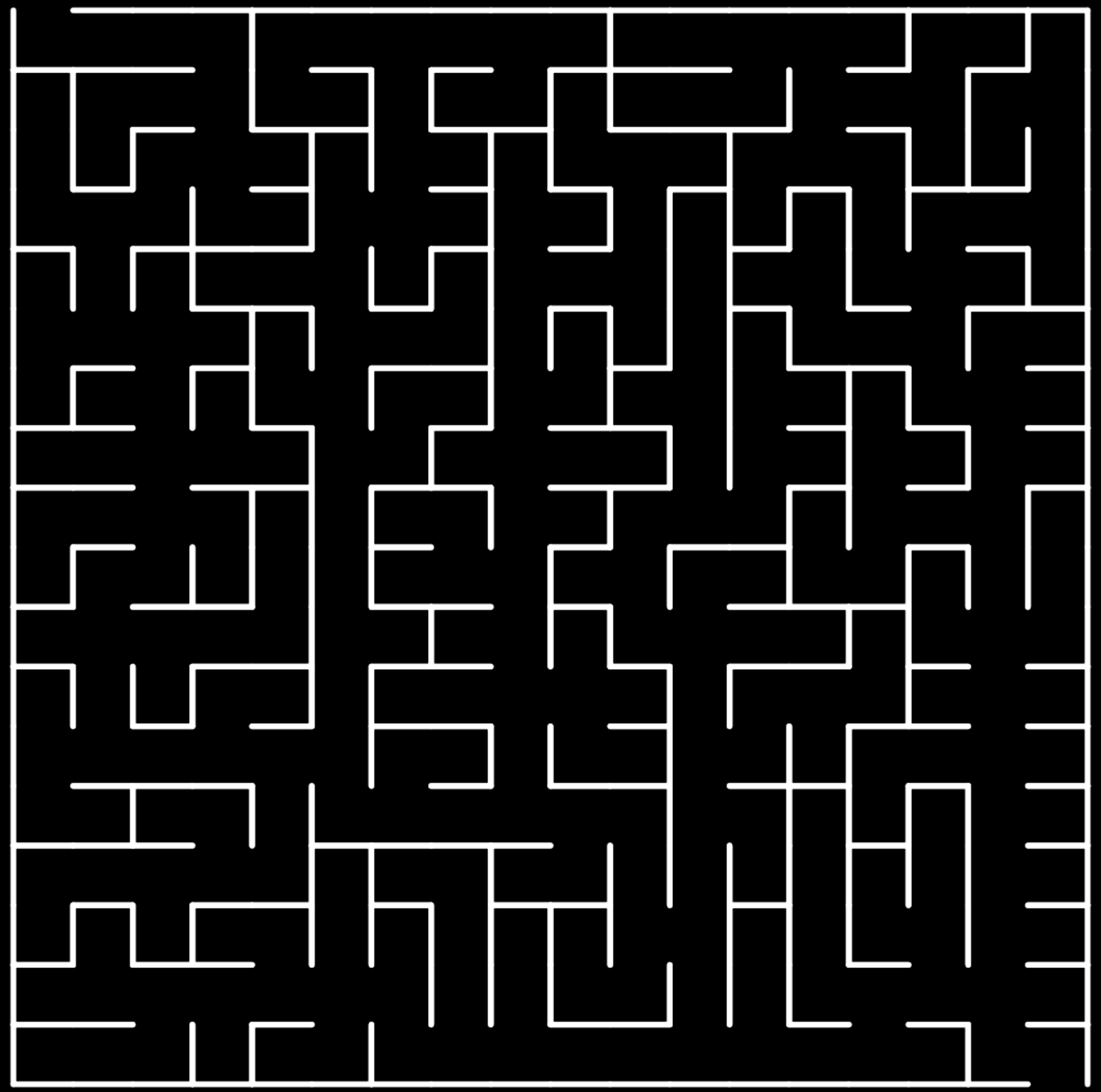


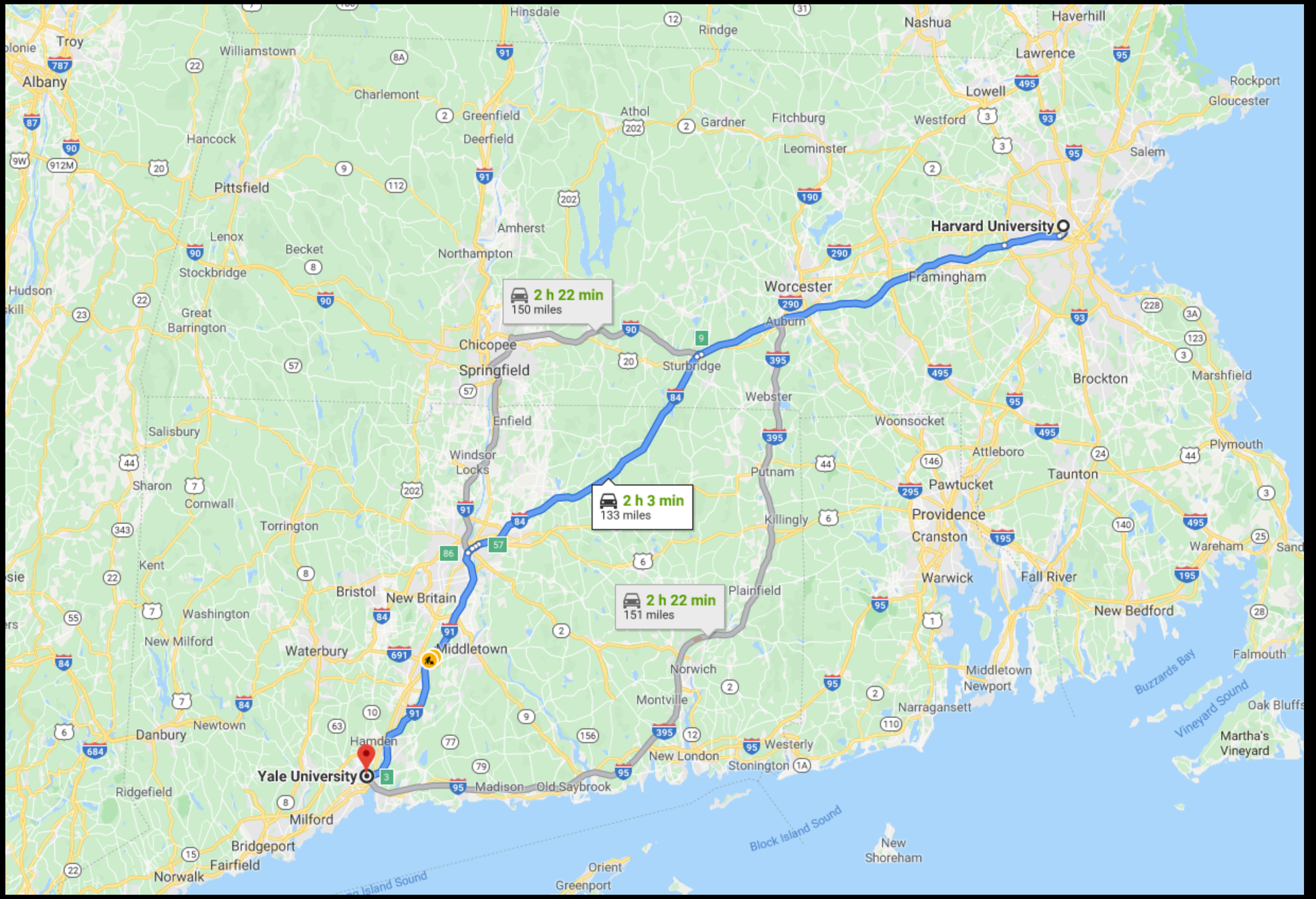
# Language



# Search


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




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

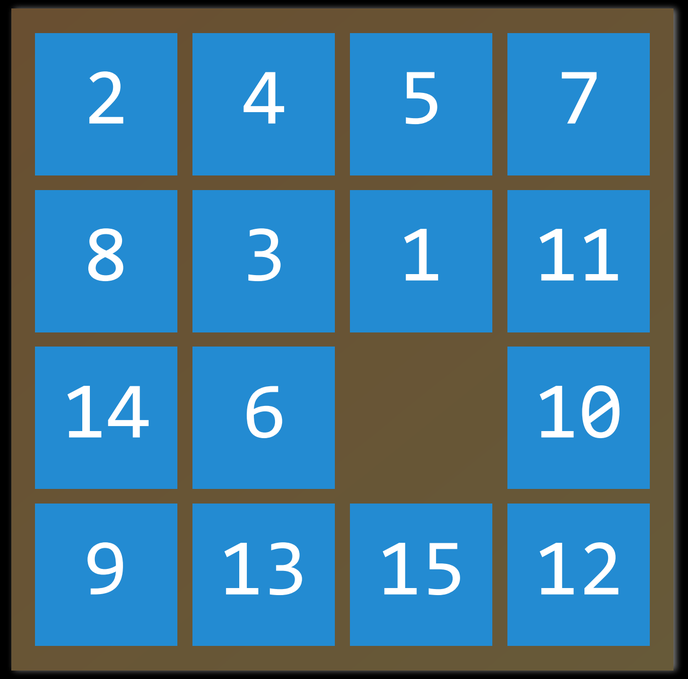
# Search Problems

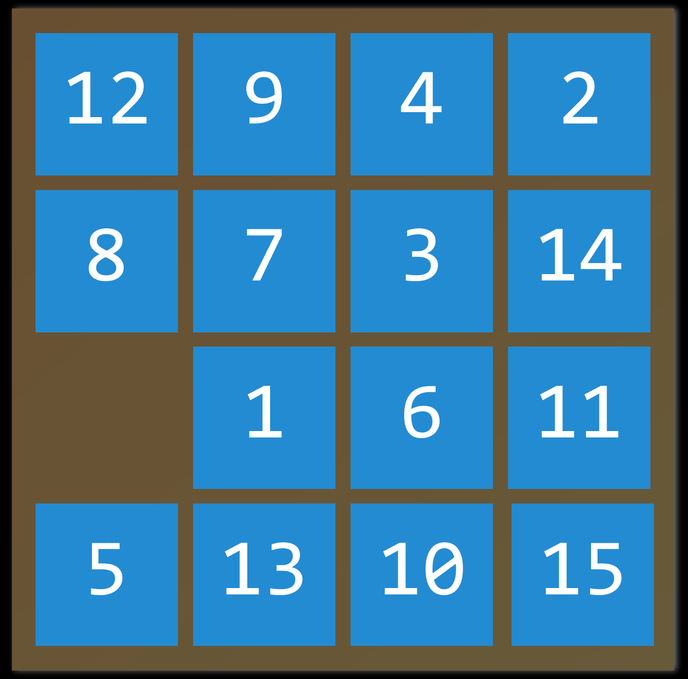
# agent

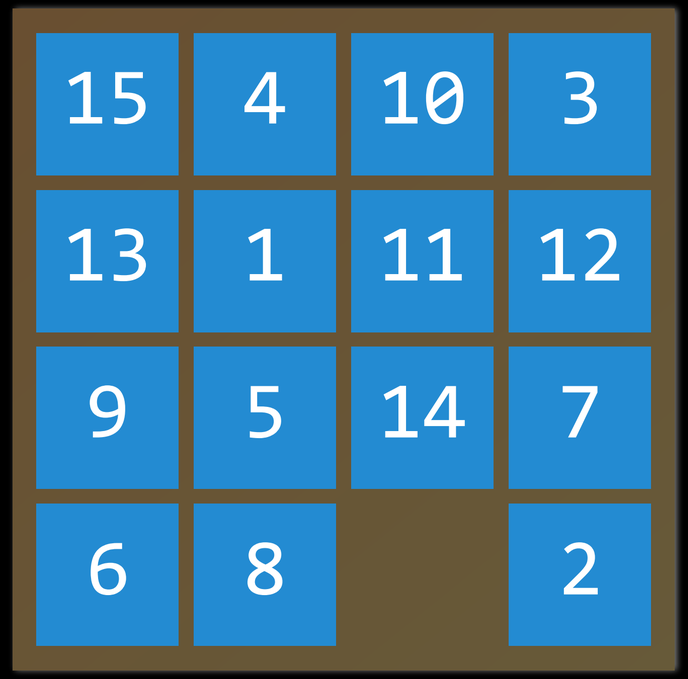
entity that perceives its environment
and acts upon that environment

# state

a configuration of the agent and
its environment


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

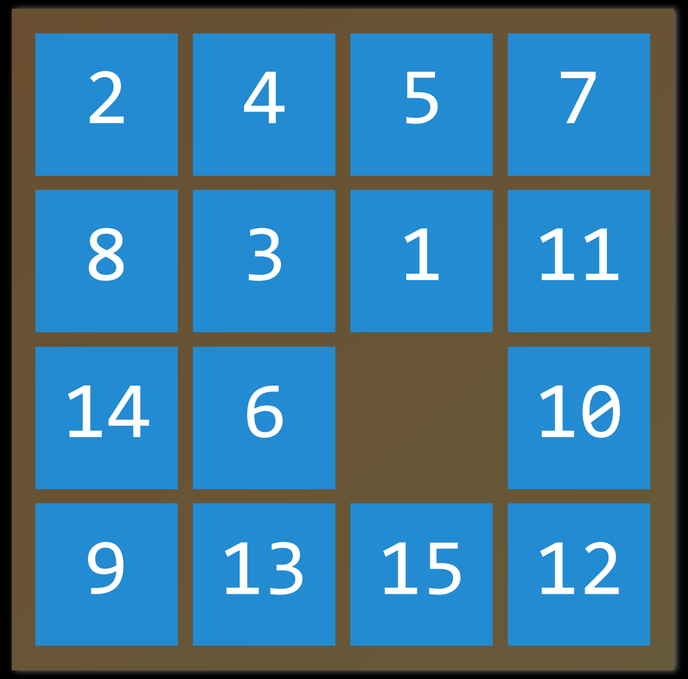

12 9 4 2
8 7 3 14
1 6 11
5 13 10 15


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



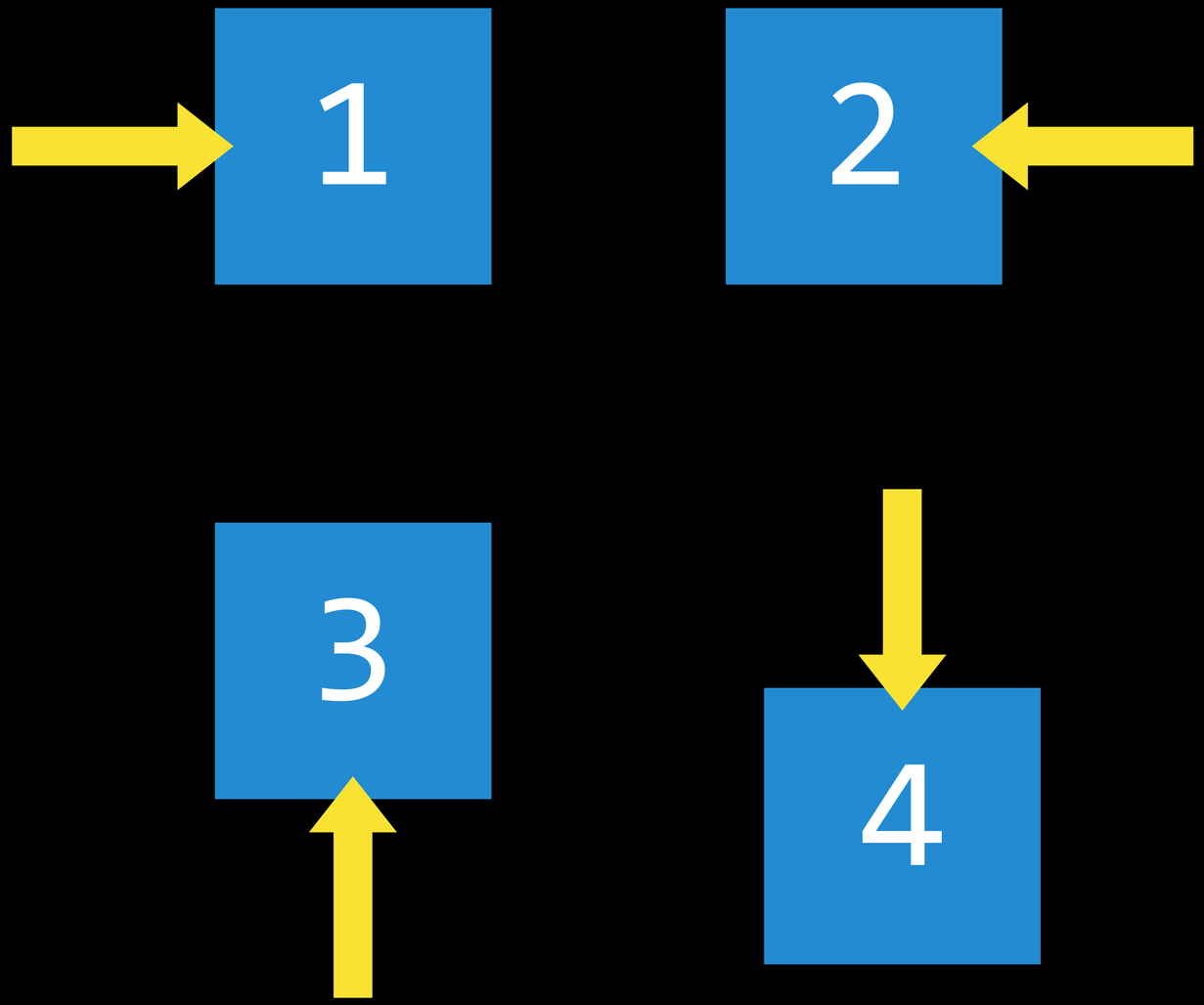
# actions

choices that can be made in a state

# actions

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

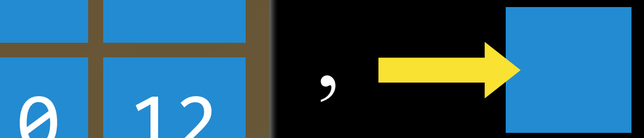
# transition model

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 | |

,



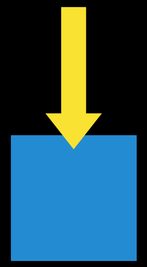
) =

| | | | |
|---|---|---|---|
| 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 | |

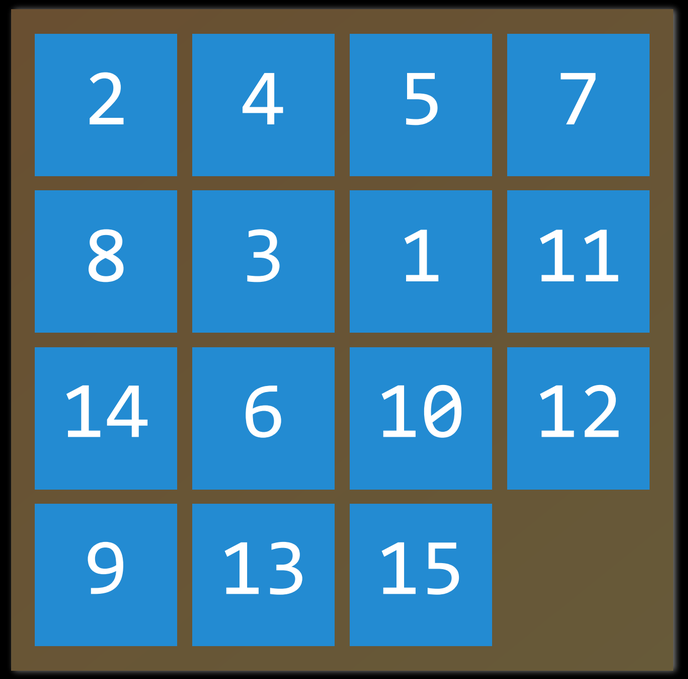
,

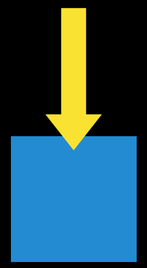


) =

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

# transition model

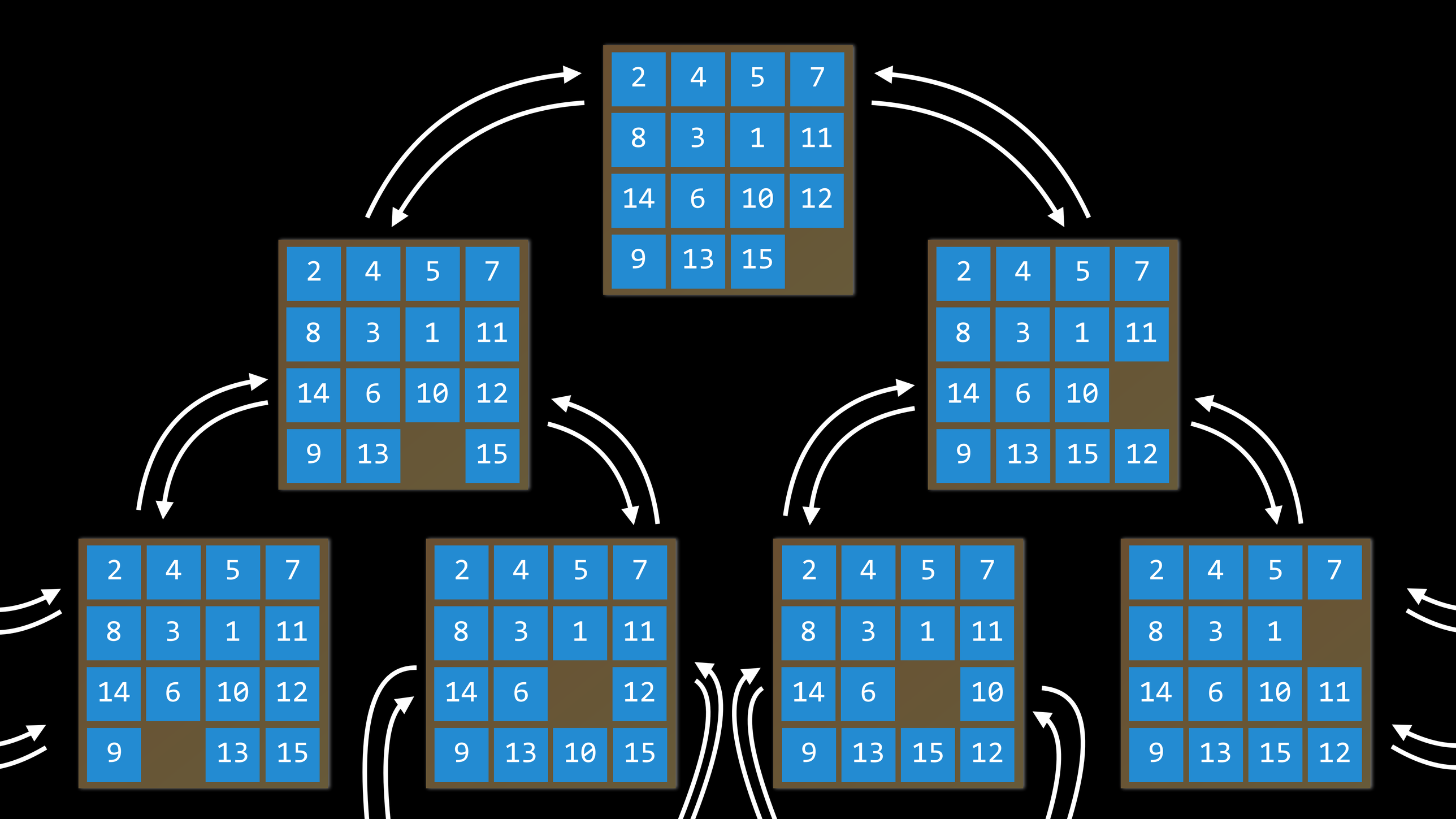
RESULT( 

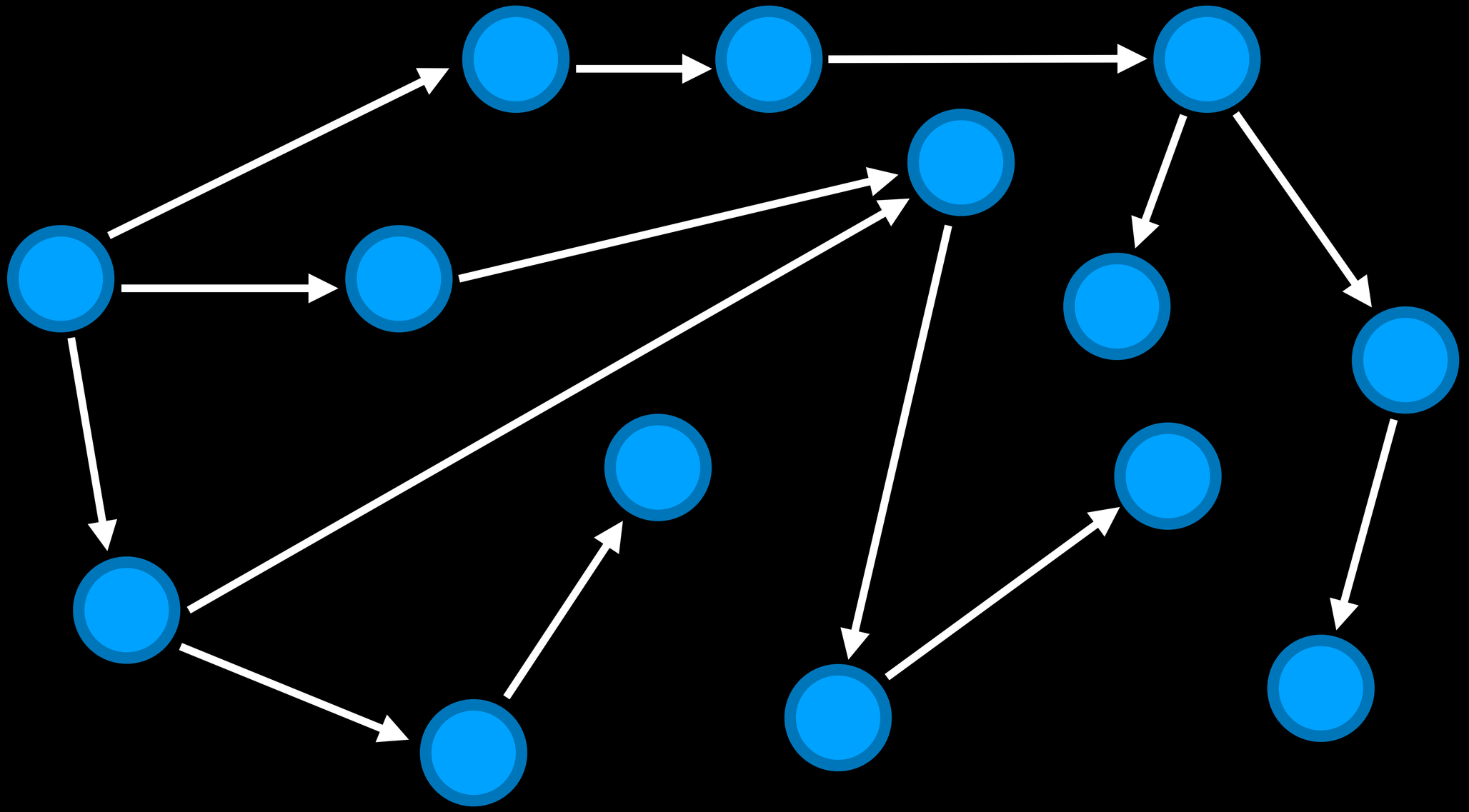
,  ) =

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

# 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

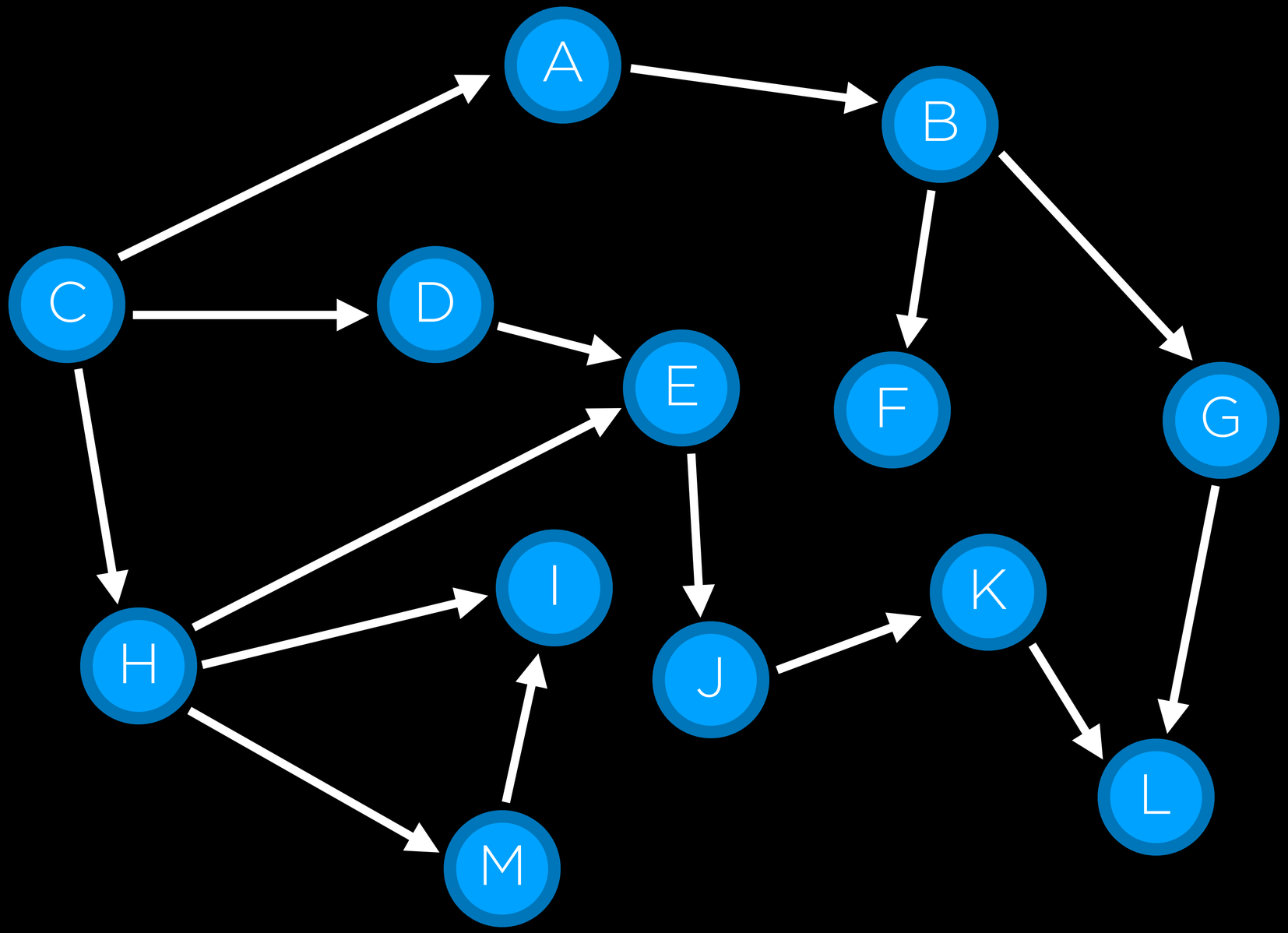


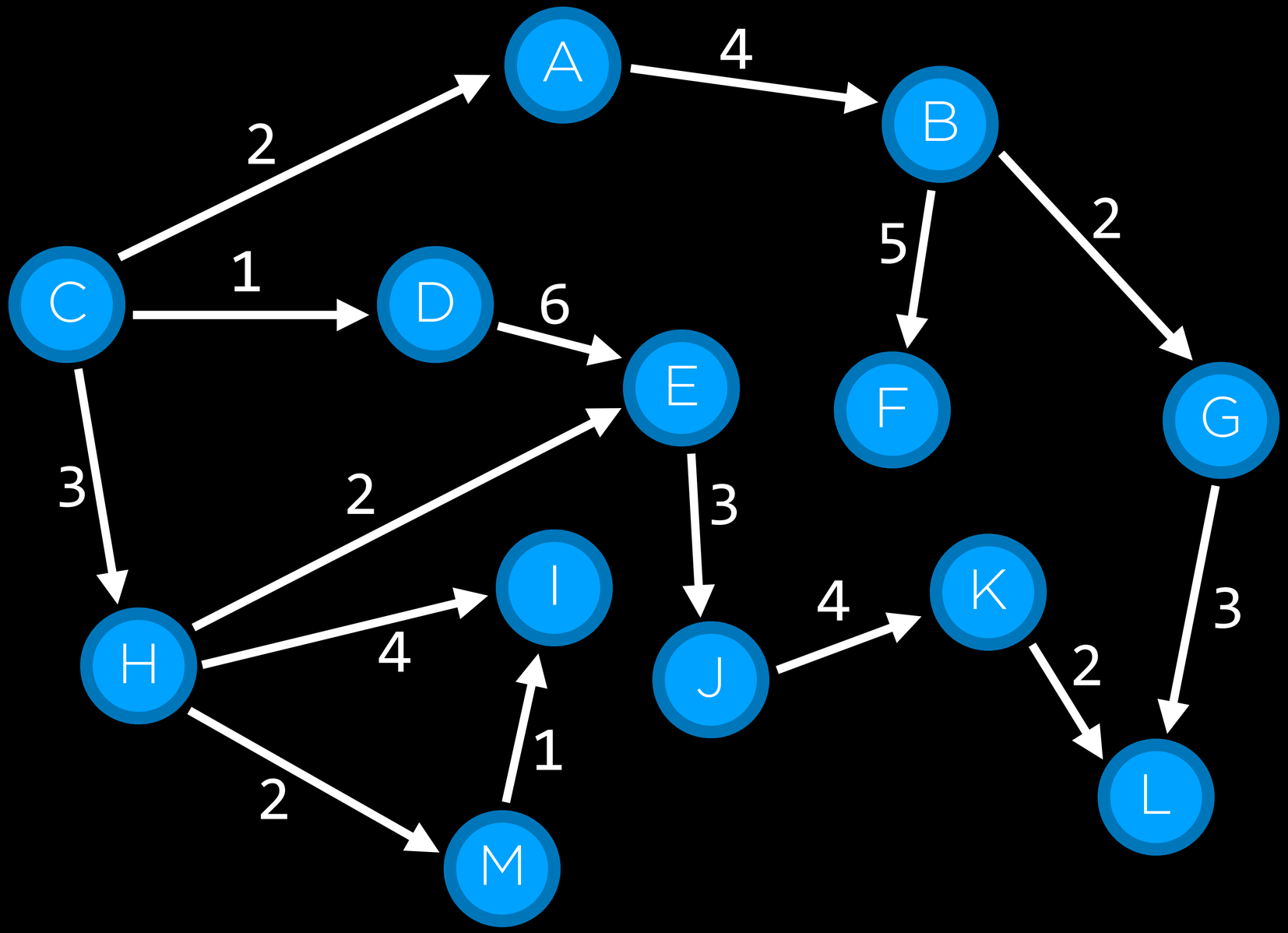
# goal test

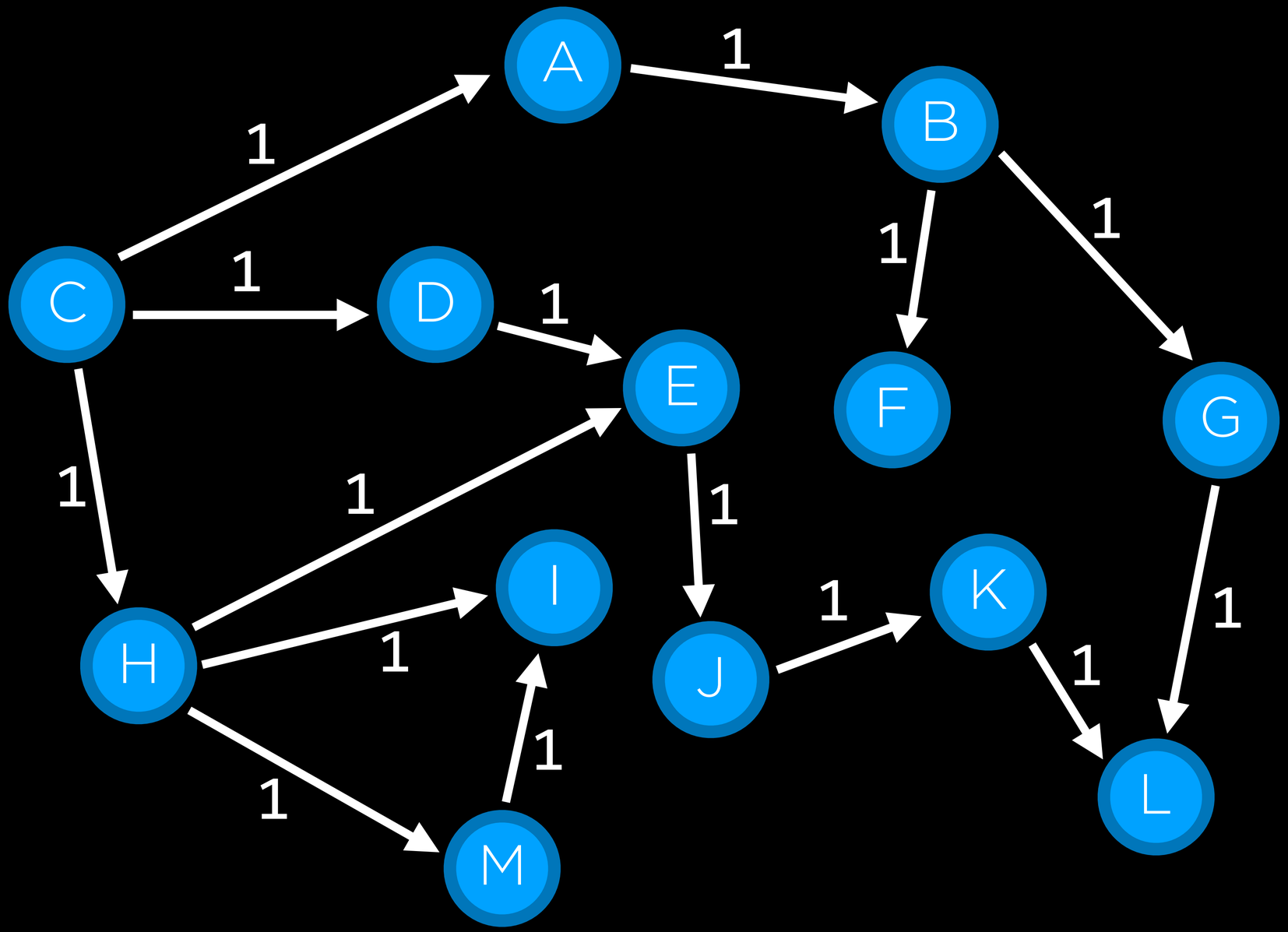
way to determine whether a given state
is a goal state

# 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
4
2
1
6
5
2
3
2
3
4
4
2
3
1
2


A
B
C
D
E
F
G
H
I
J
K
L
M
1
1
1
1
1
1
1
1
1
1
1
1
1
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)

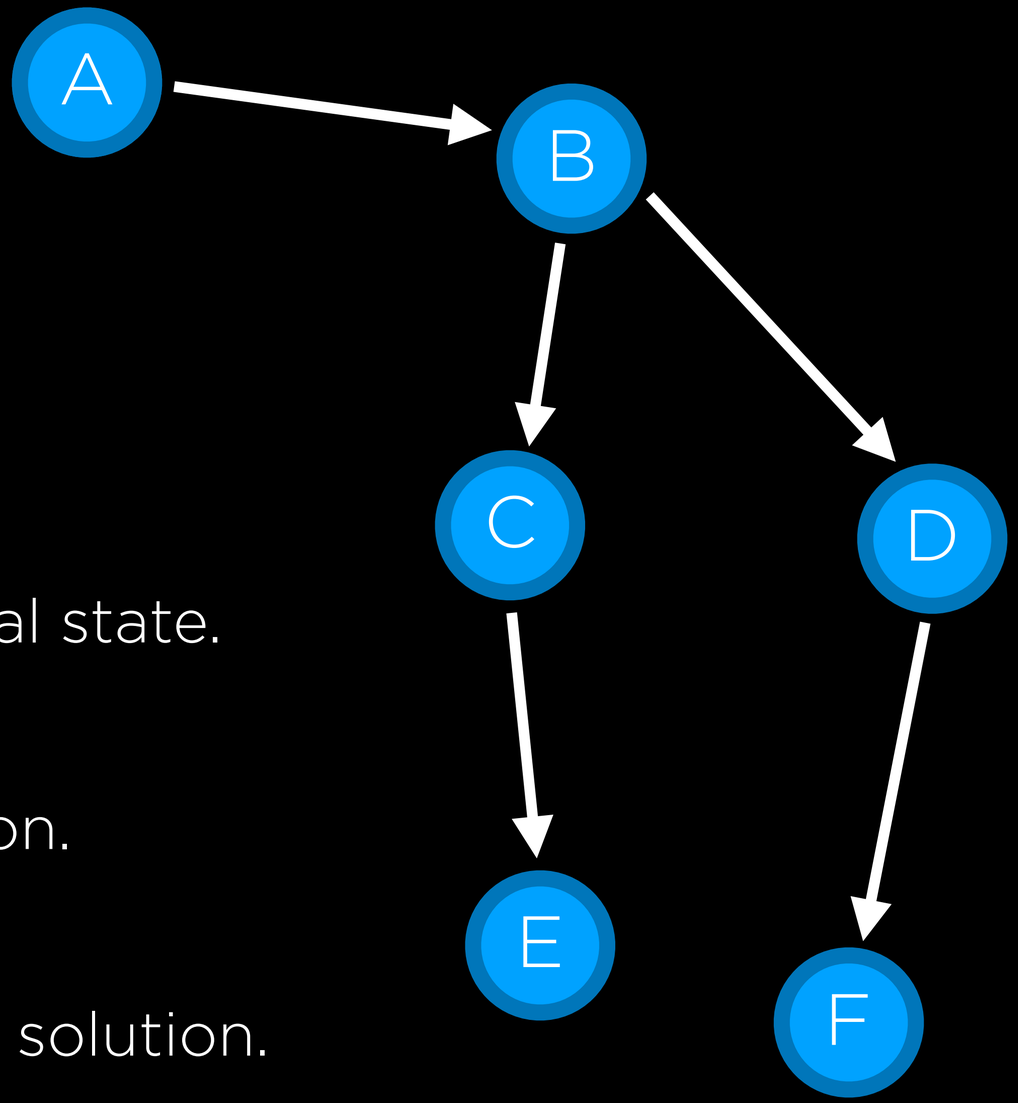
# Approach

frontier is all nodes we can explore next and haven't yet explored

- 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**

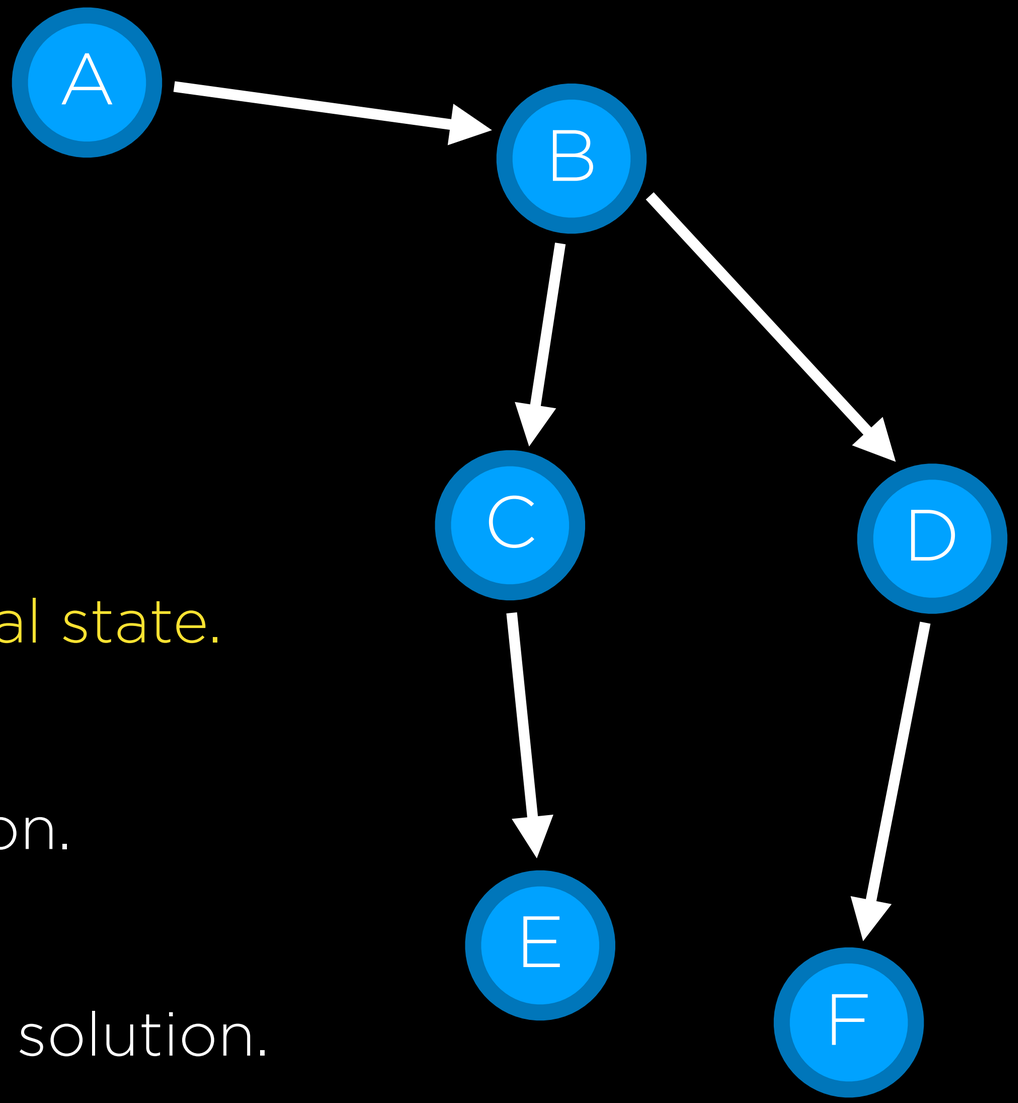



- Start with a **frontier** that contains the initial state.
- Repeat:
  - If the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - If node contains goal state, return the solution.
  - **Expand** node, add resulting nodes to the frontier.

# Find a path from A to E.

**Frontier**

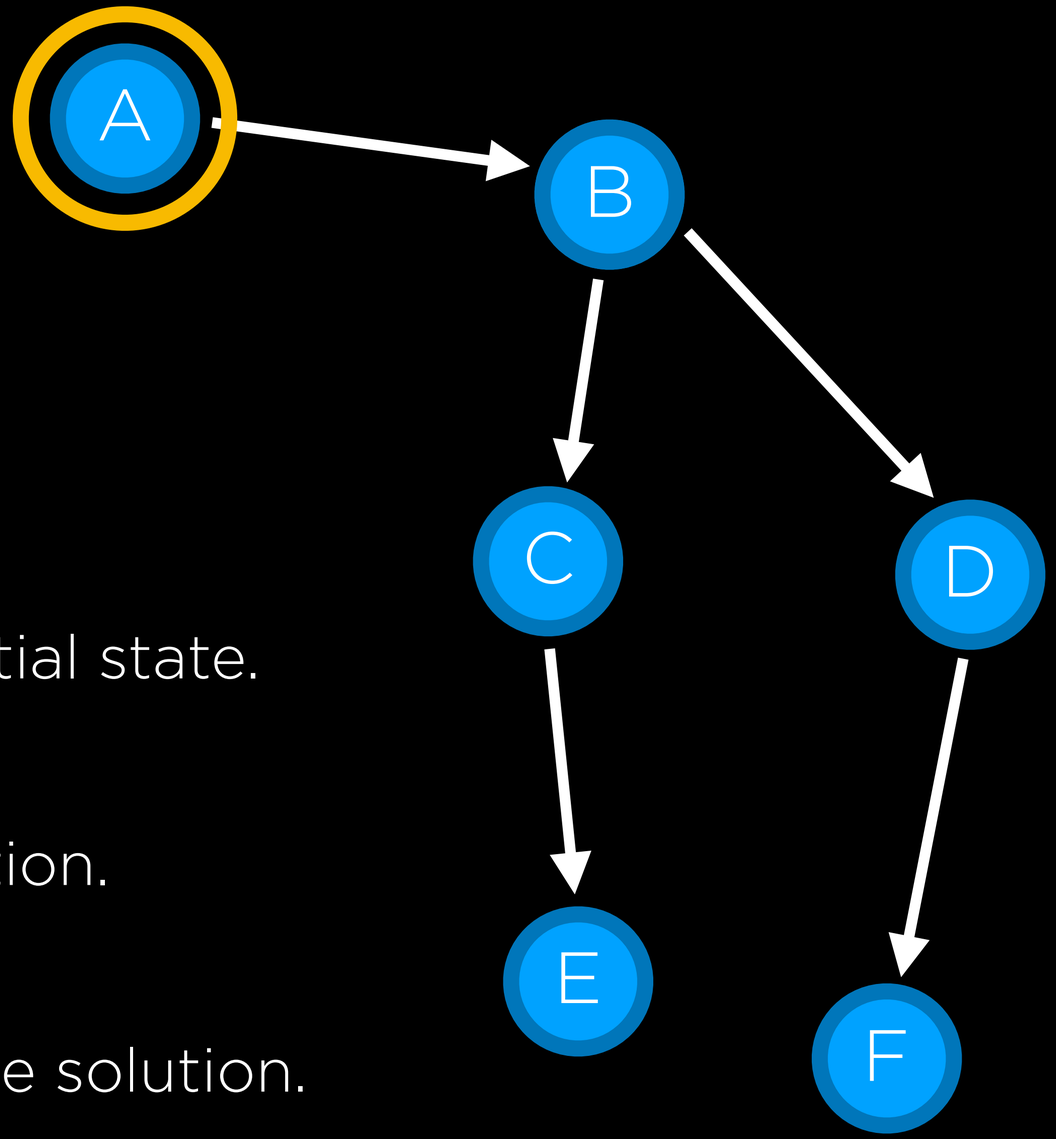
A




- 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.

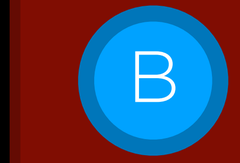
**Frontier**

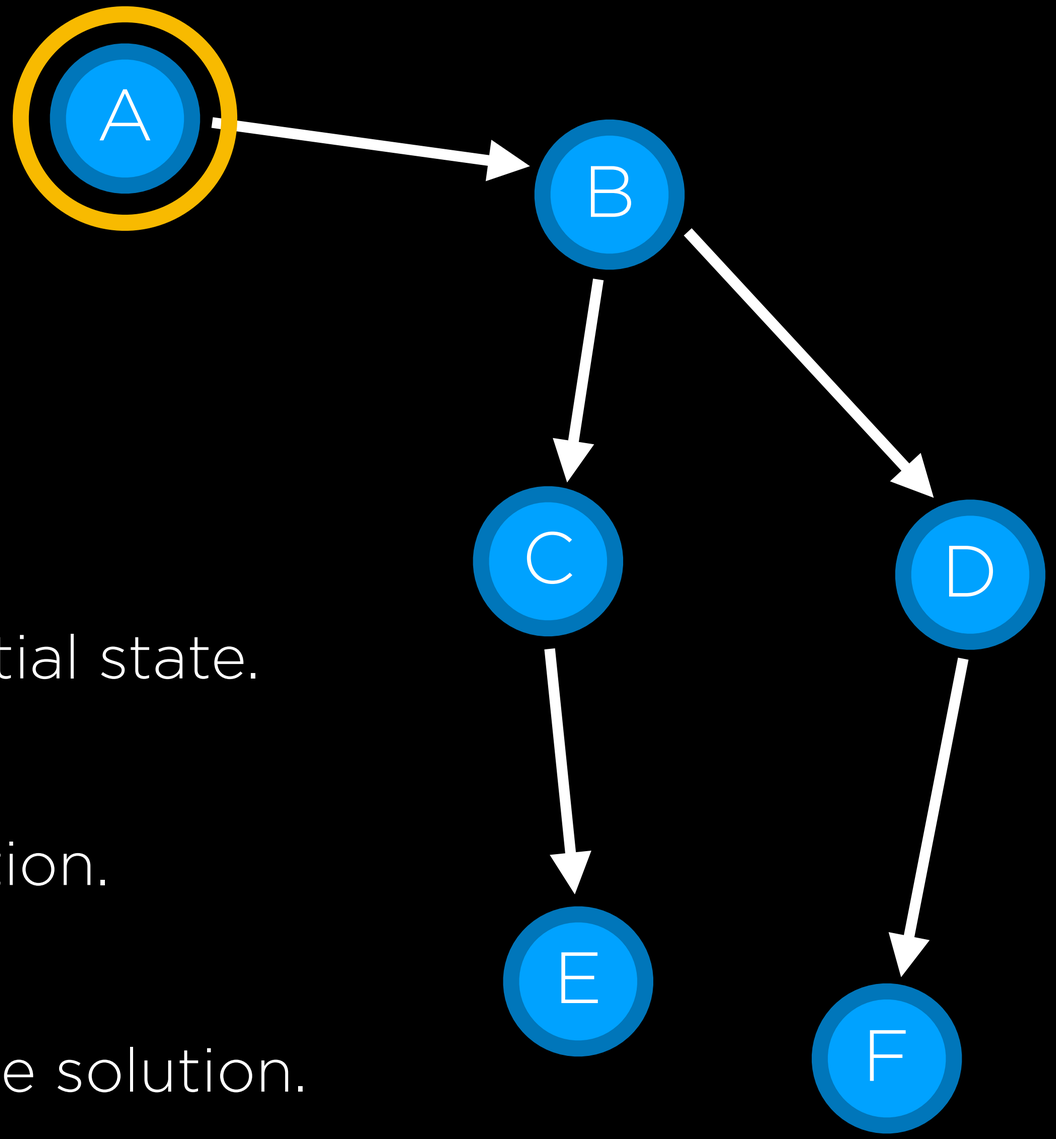



- Start with a **frontier** that contains the initial state.
- Repeat:
  - If the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - If node contains goal state, return the solution.
  - **Expand** node, add resulting nodes to the frontier.

# Find a path from A to E.

**Frontier**

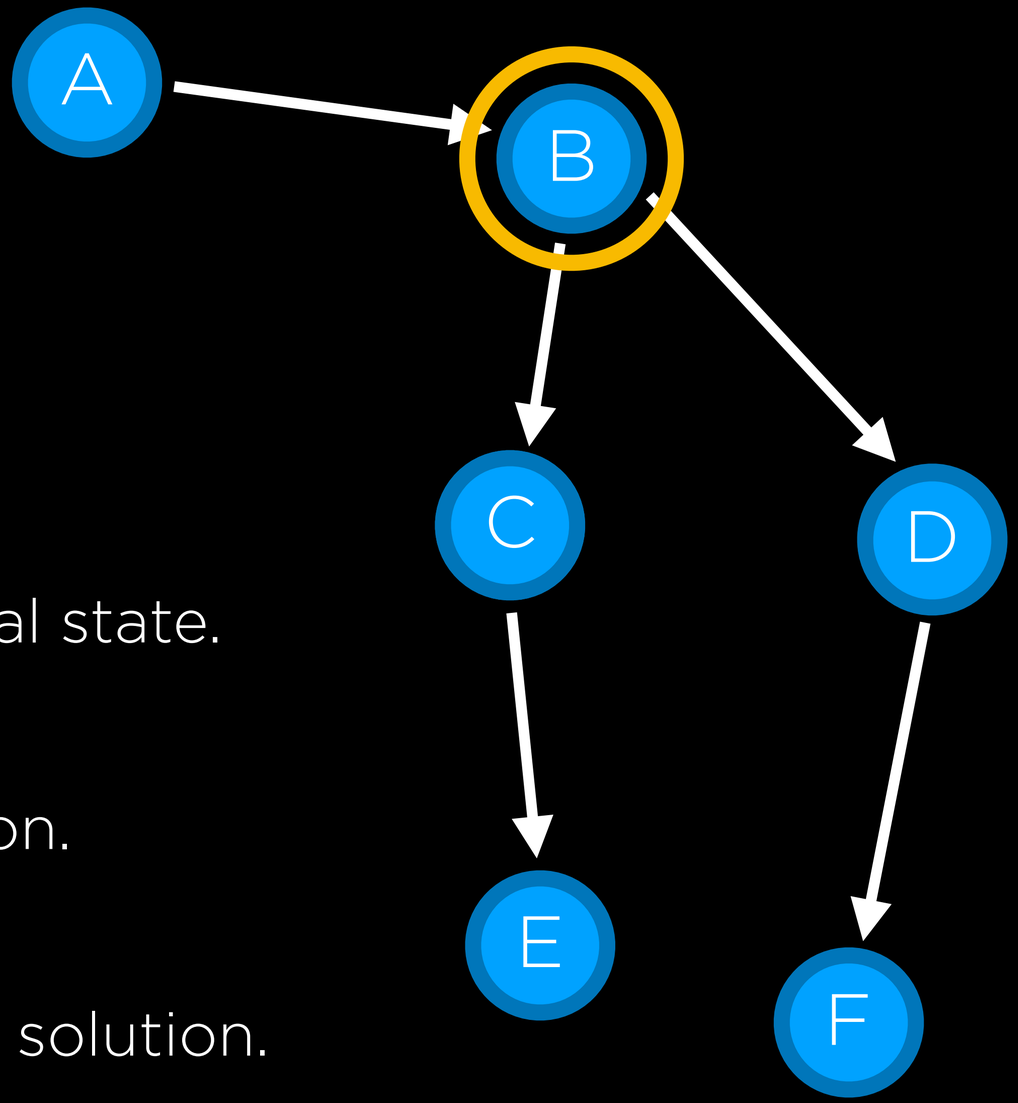



- Start with a **frontier** that contains the initial state.
- Repeat:
  - If the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - If node contains goal state, return the solution.
  - **Expand** node, add resulting nodes to the frontier.

# Find a path from A to E.

**Frontier**

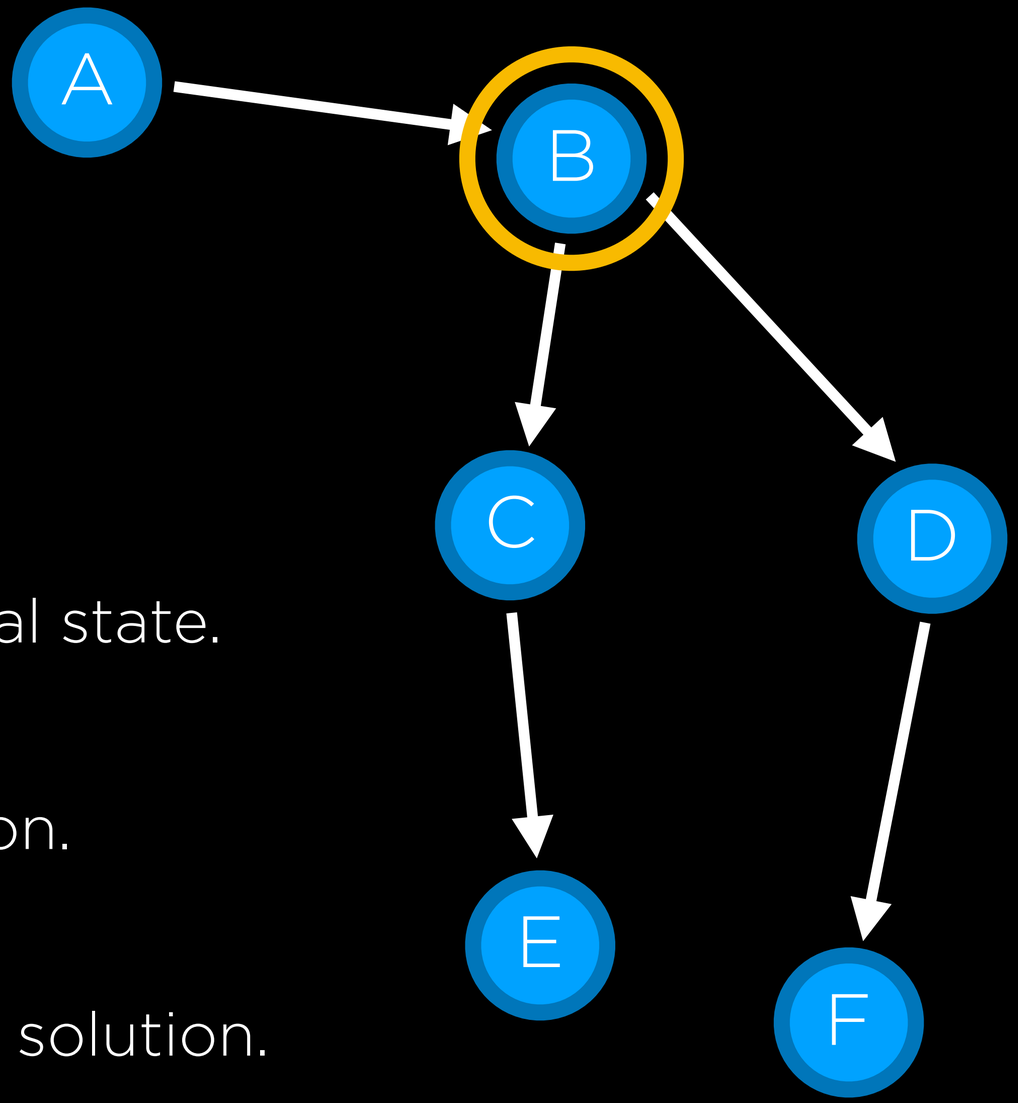



- Start with a **frontier** that contains the initial state.
- Repeat:
  - If the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - If node contains goal state, return the solution.
  - **Expand** node, add resulting nodes to the frontier.

# Find a path from A to E.

**Frontier**

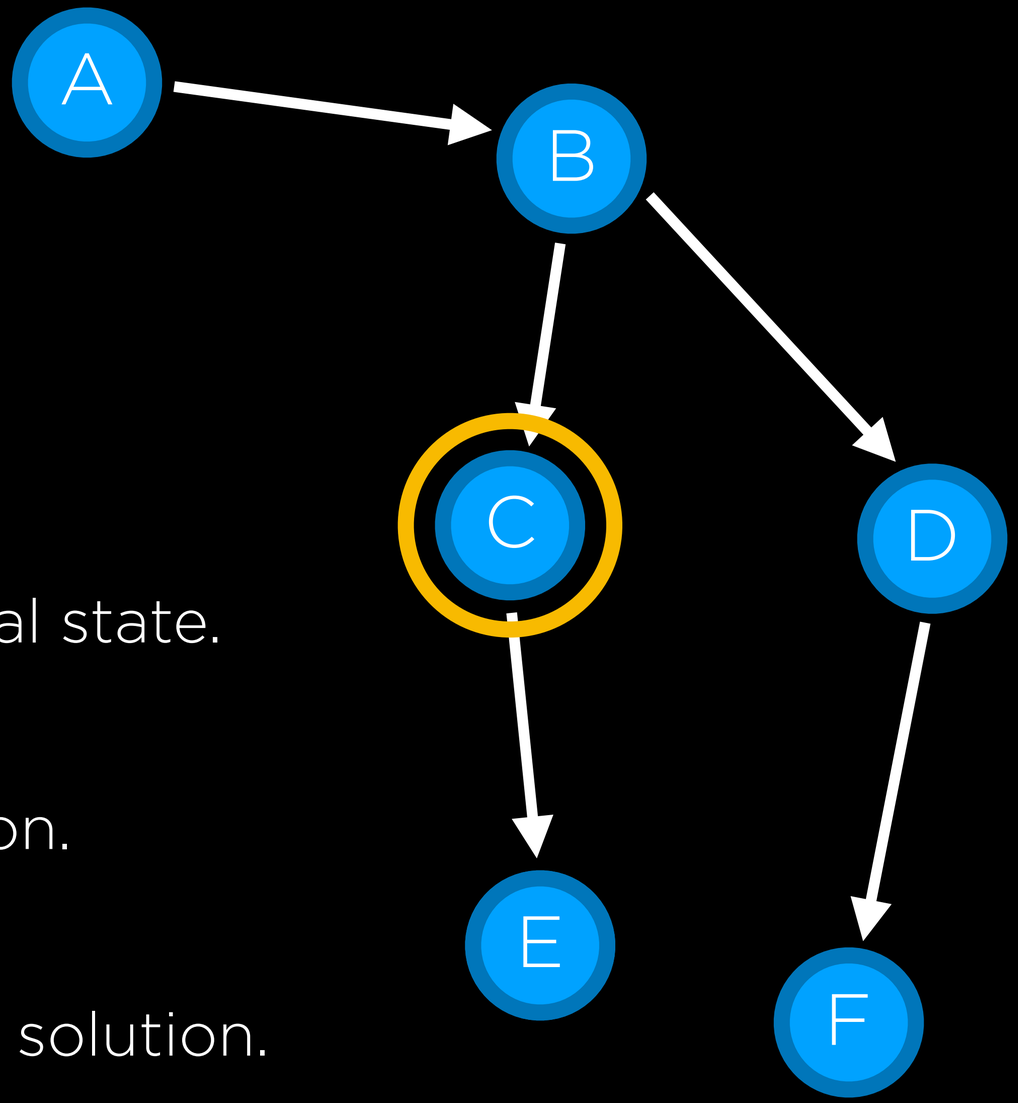
C D




- 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**

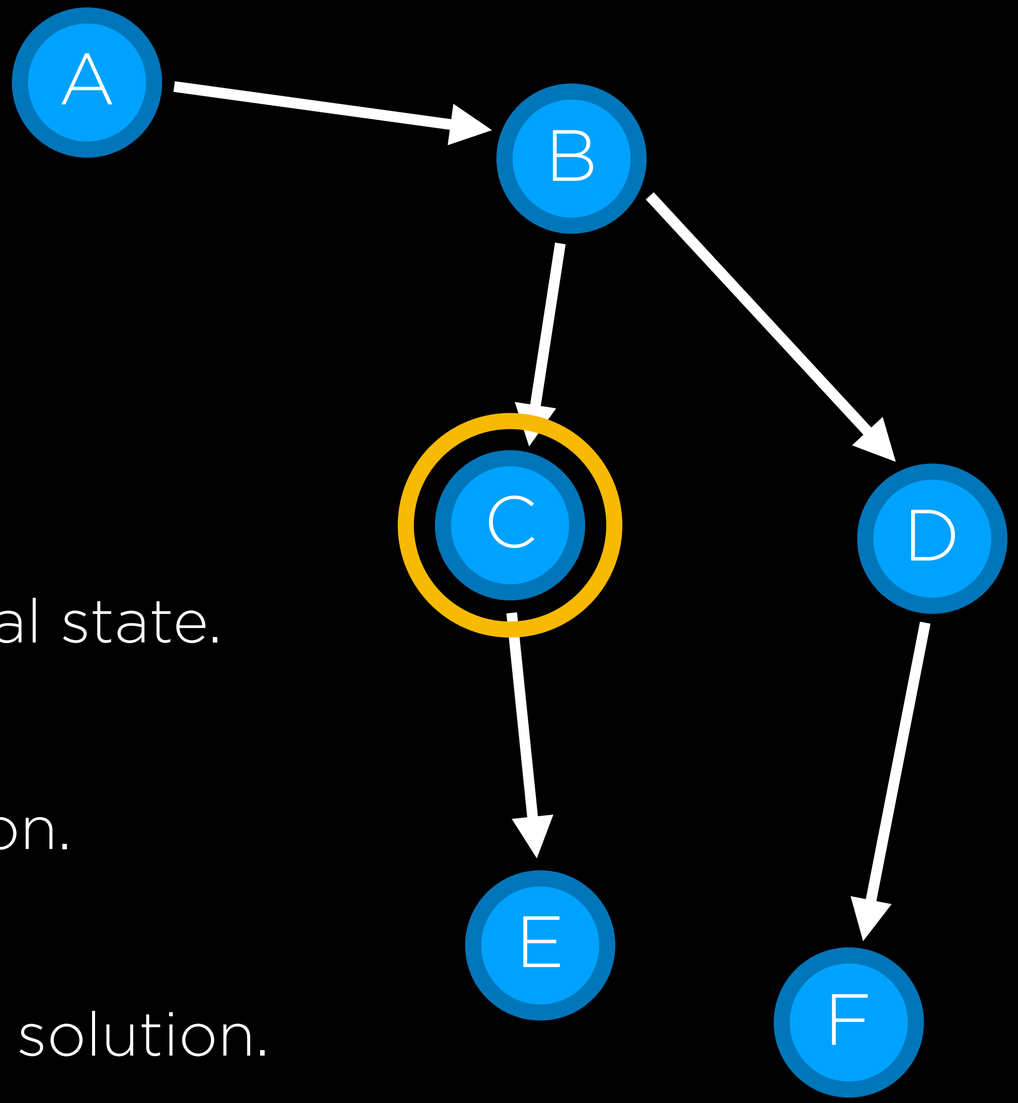




- Start with a **frontier** that contains the initial state.
- Repeat:
  - If the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - If node contains goal state, return the solution.
  - **Expand** node, add resulting nodes to the frontier.

# Find a path from A to E.

**Frontier**

E D

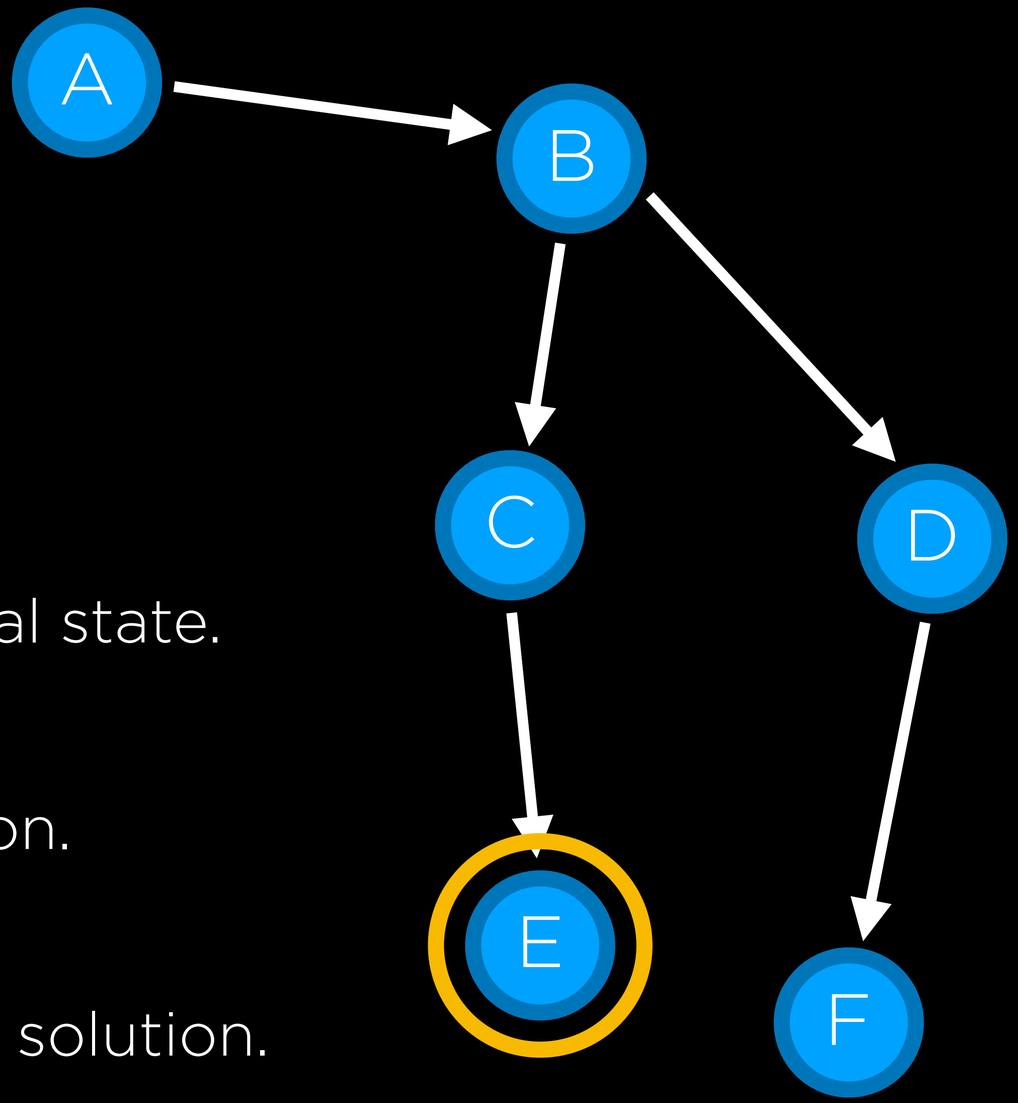



- Start with a **frontier** that contains the initial state.
- Repeat:
  - If the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - If node contains goal state, return the solution.
  - **Expand** node, add resulting nodes to the frontier.

# Find a path from A to E.

**Frontier**

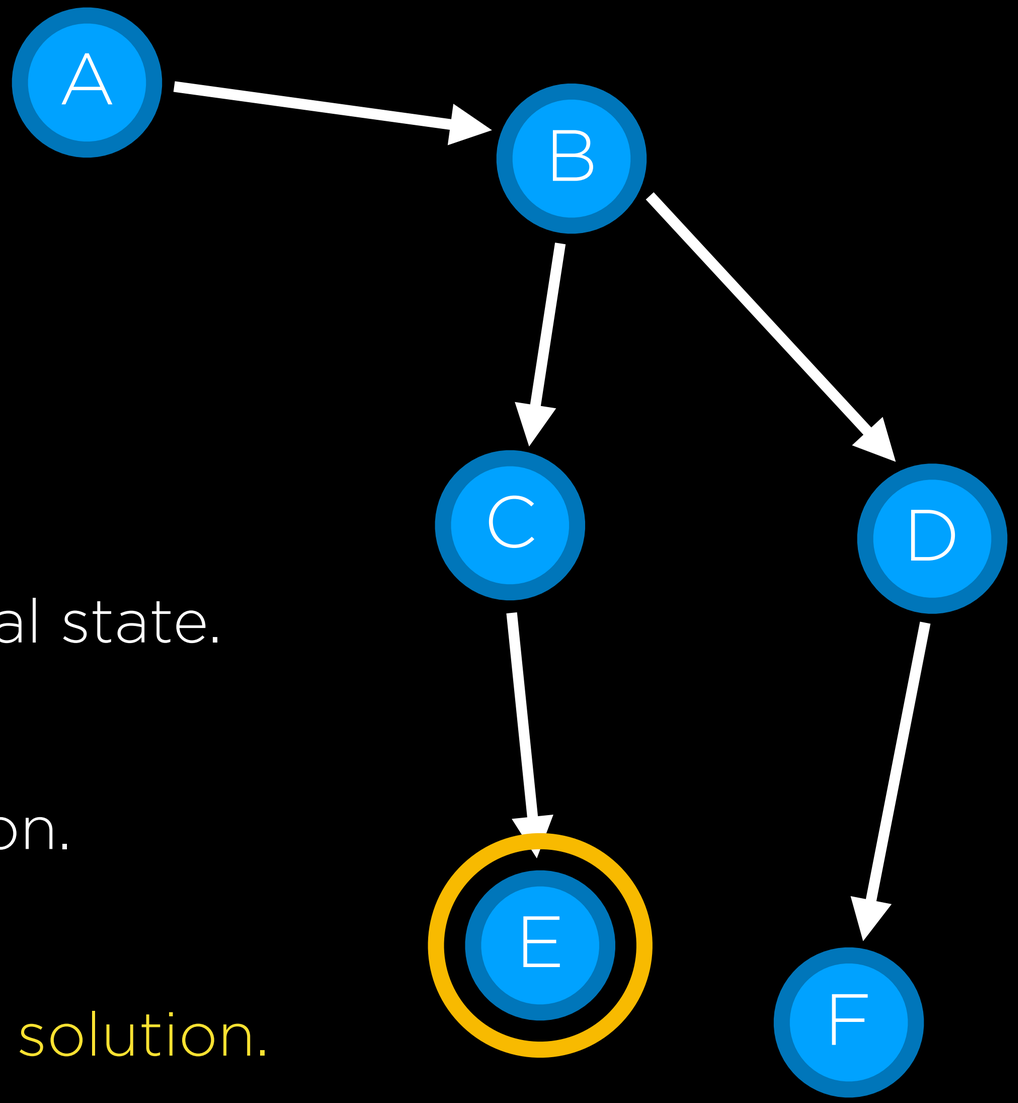
D




- 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**

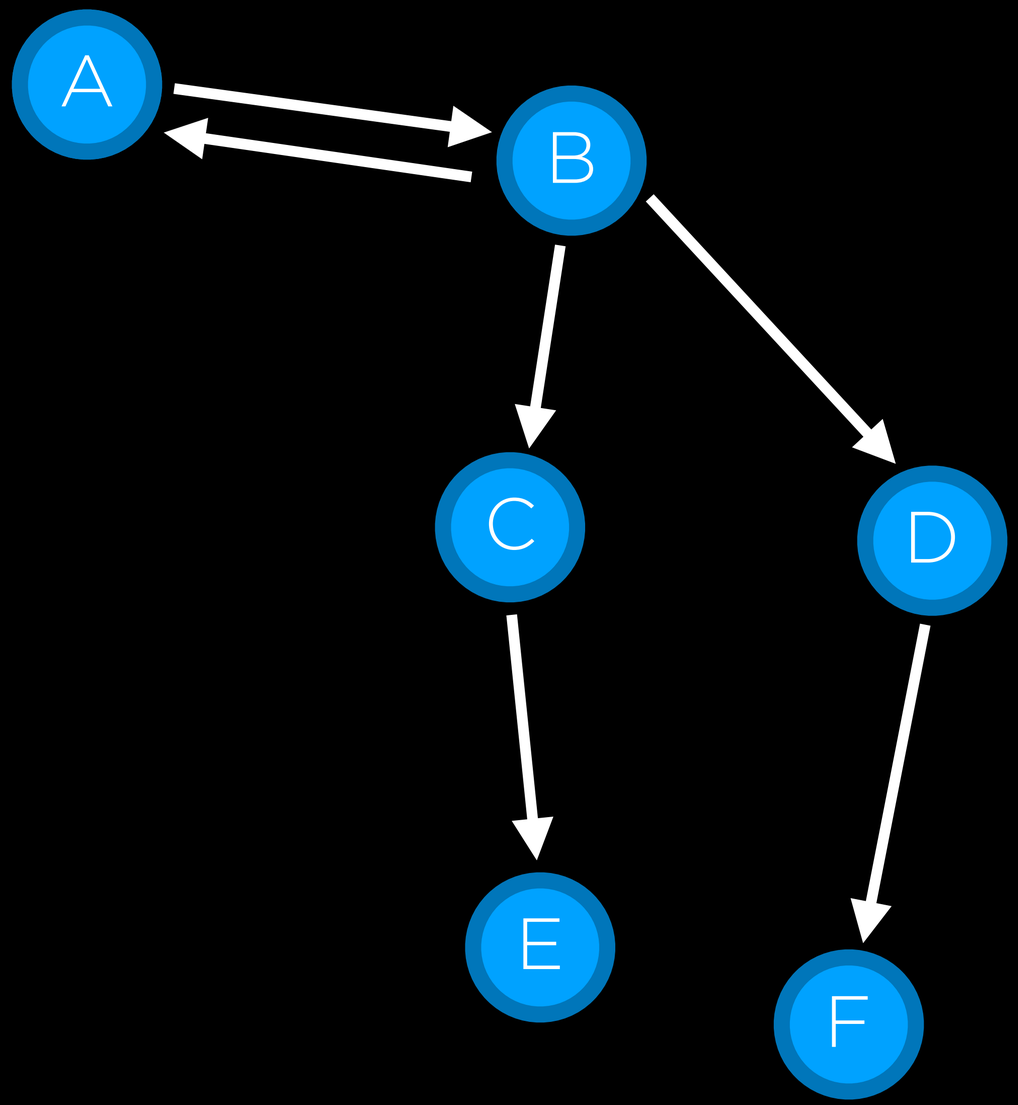




- Start with a **frontier** that contains the initial state.
- Repeat:
  - If the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - If node contains goal state, return the solution.
  - **Expand** node, add resulting nodes to the frontier.

# What could go wrong?

Find a path from A to E.

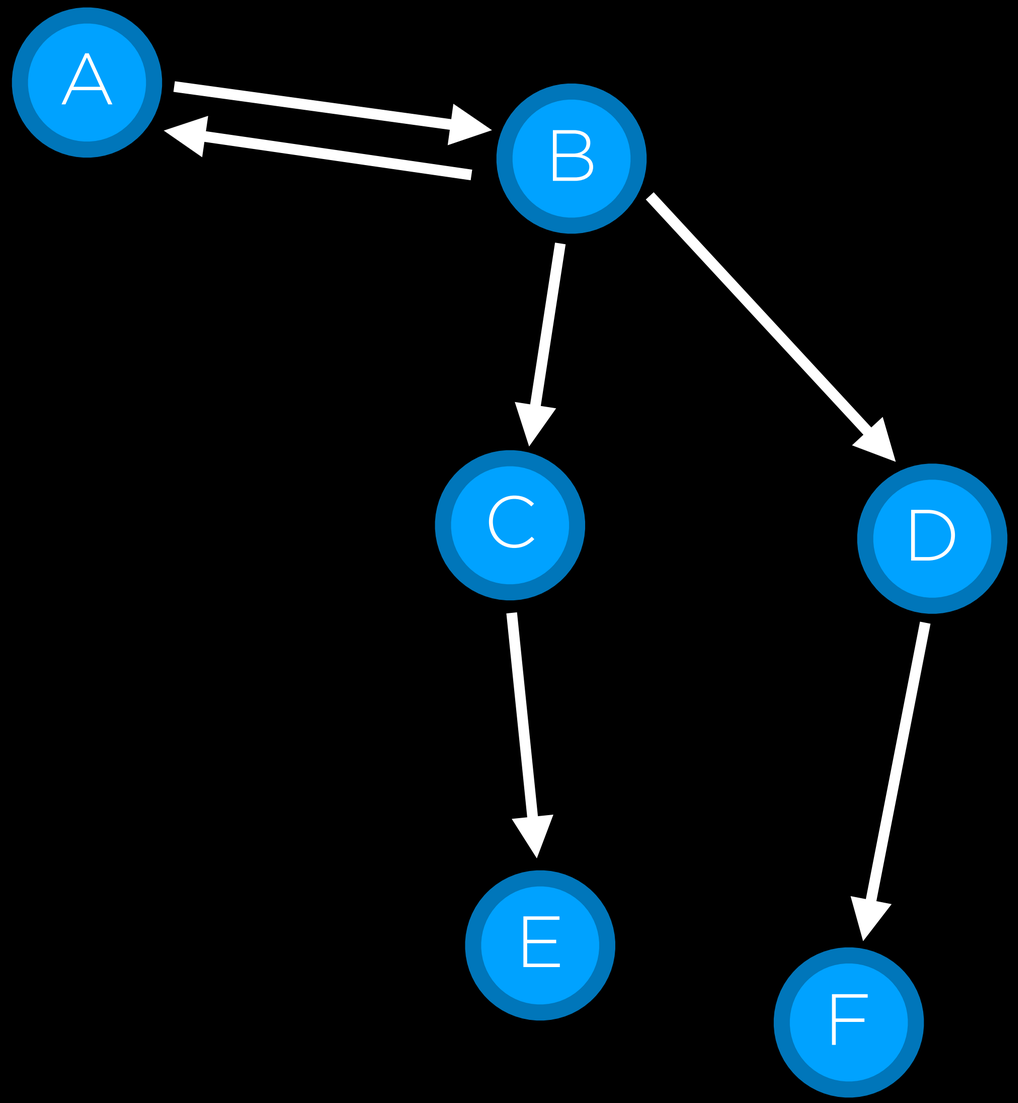
**Frontier**



Find a path from A to E.

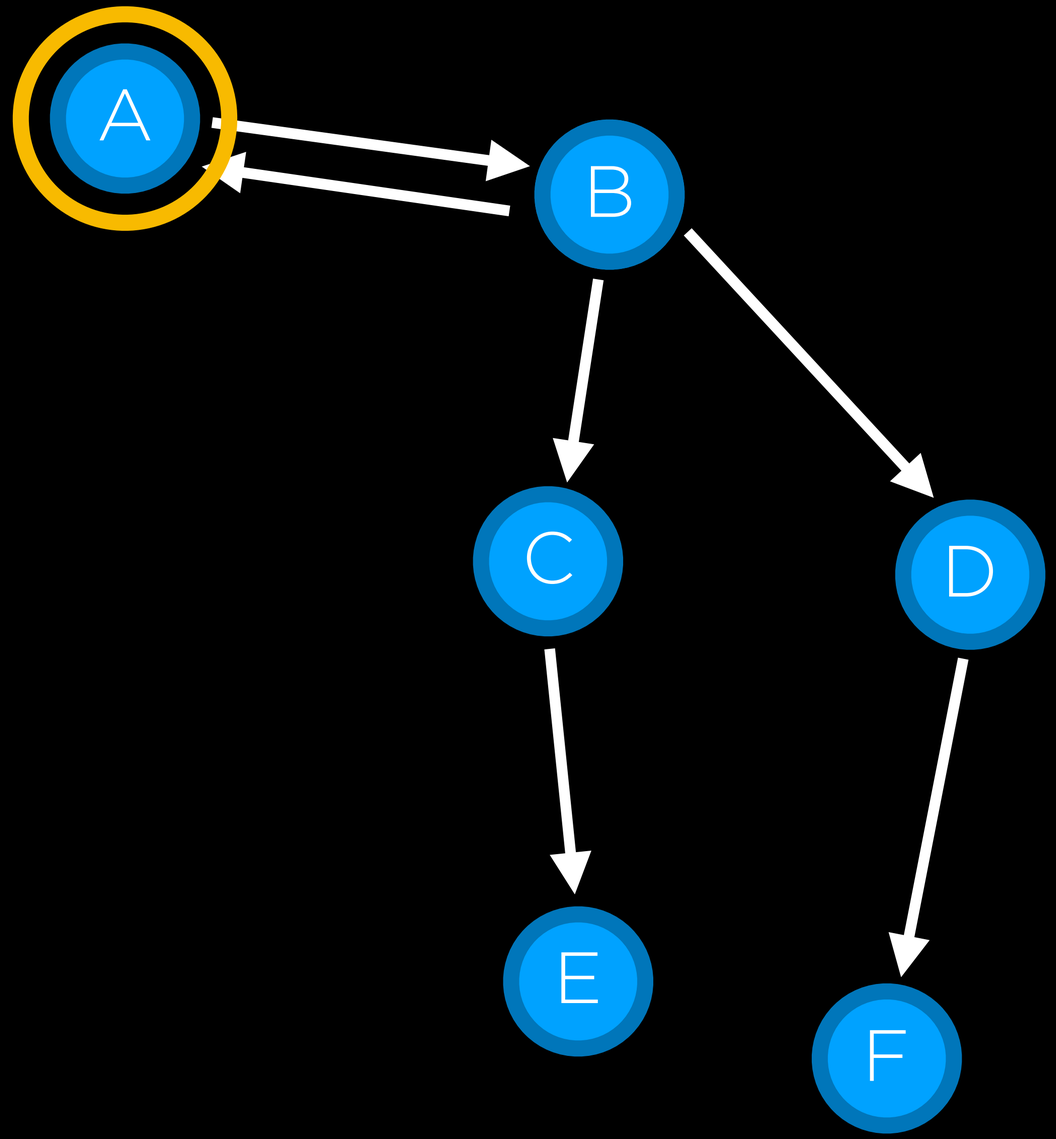
**Frontier**

A



Find a path from A to E.

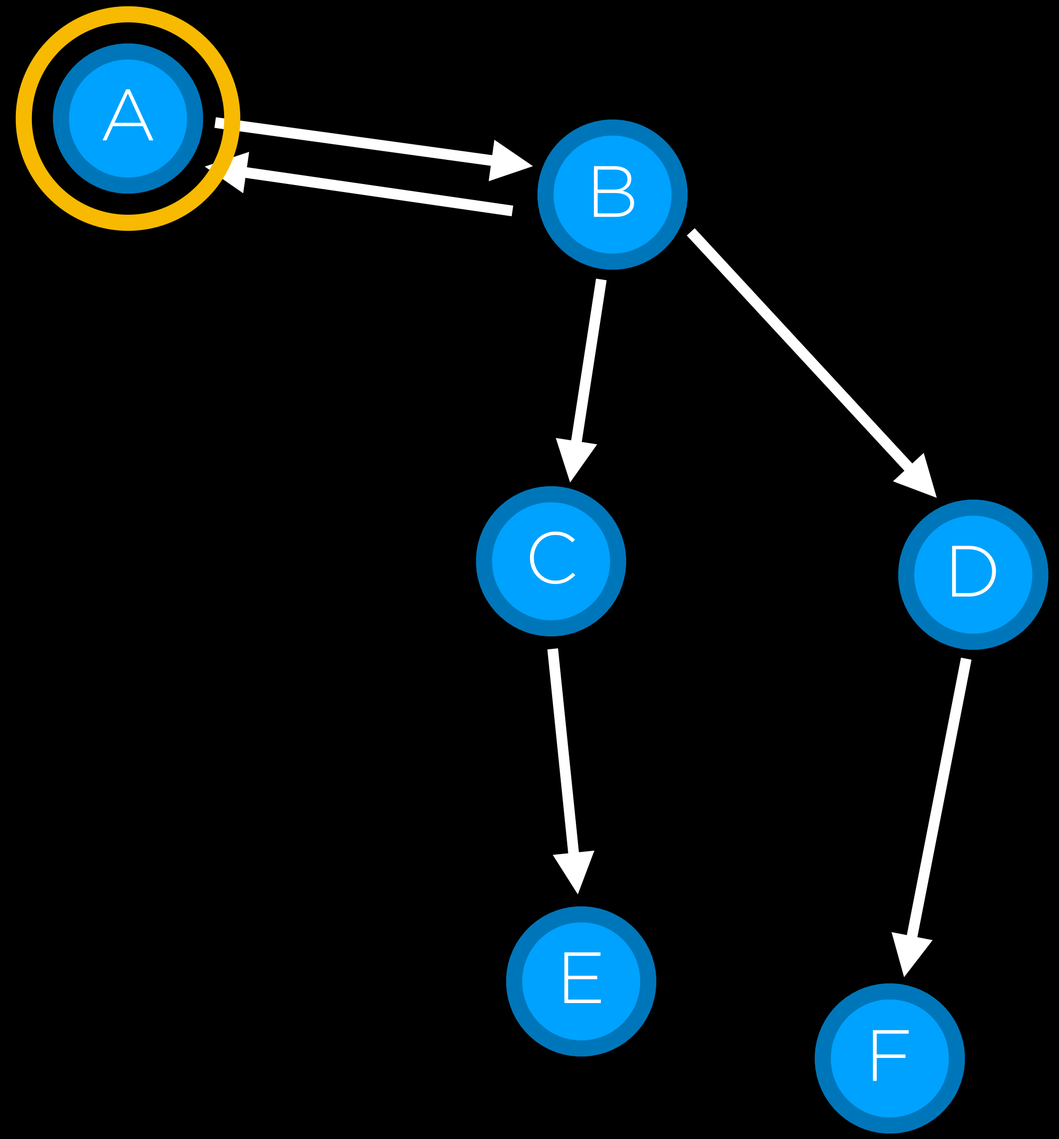
**Frontier**



Find a path from A to E.

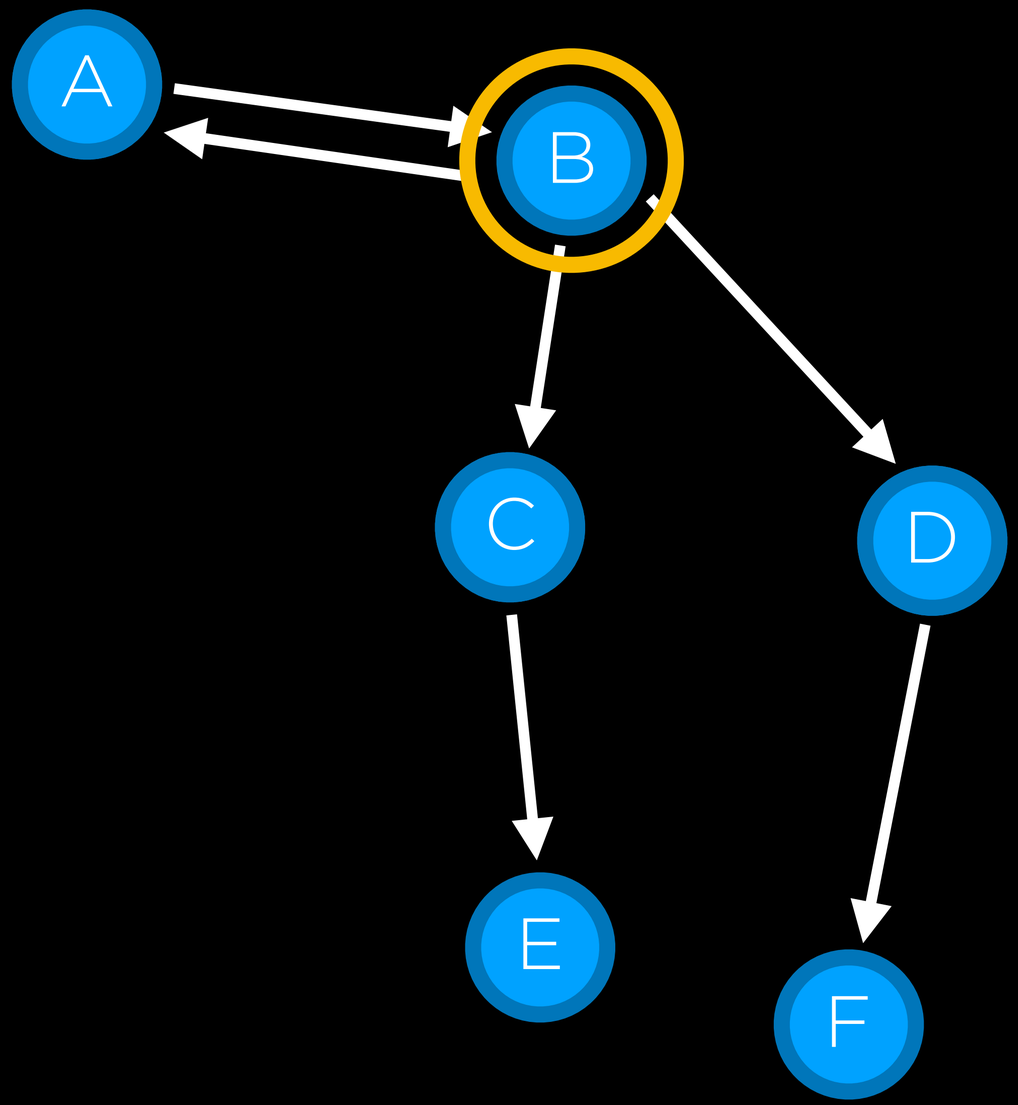
**Frontier**

B



Find a path from A to E.

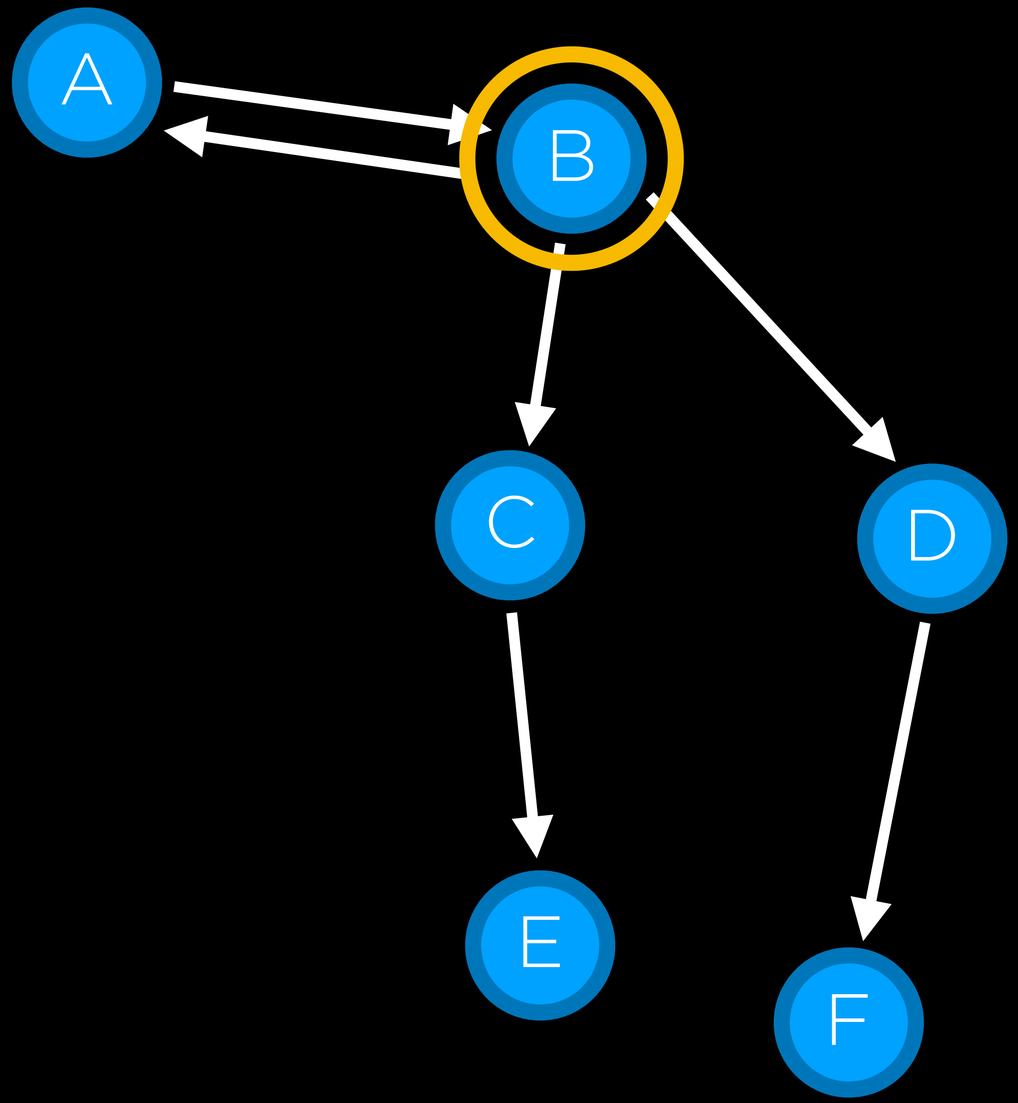
**Frontier**



Find a path from A to E.

**Frontier**

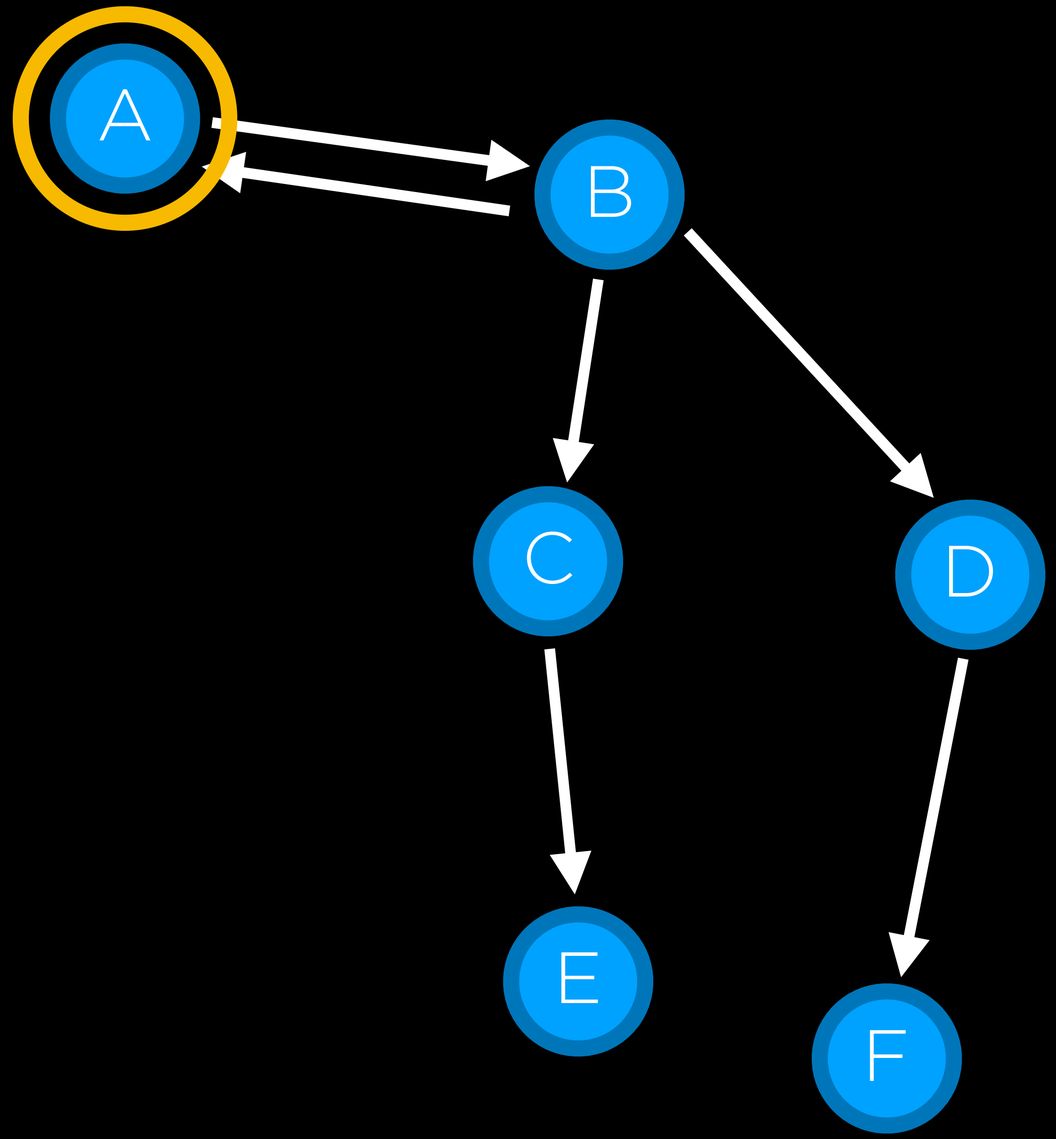
A C D



Find a path from A to E.

**Frontier**

C D



# 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.

# 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.

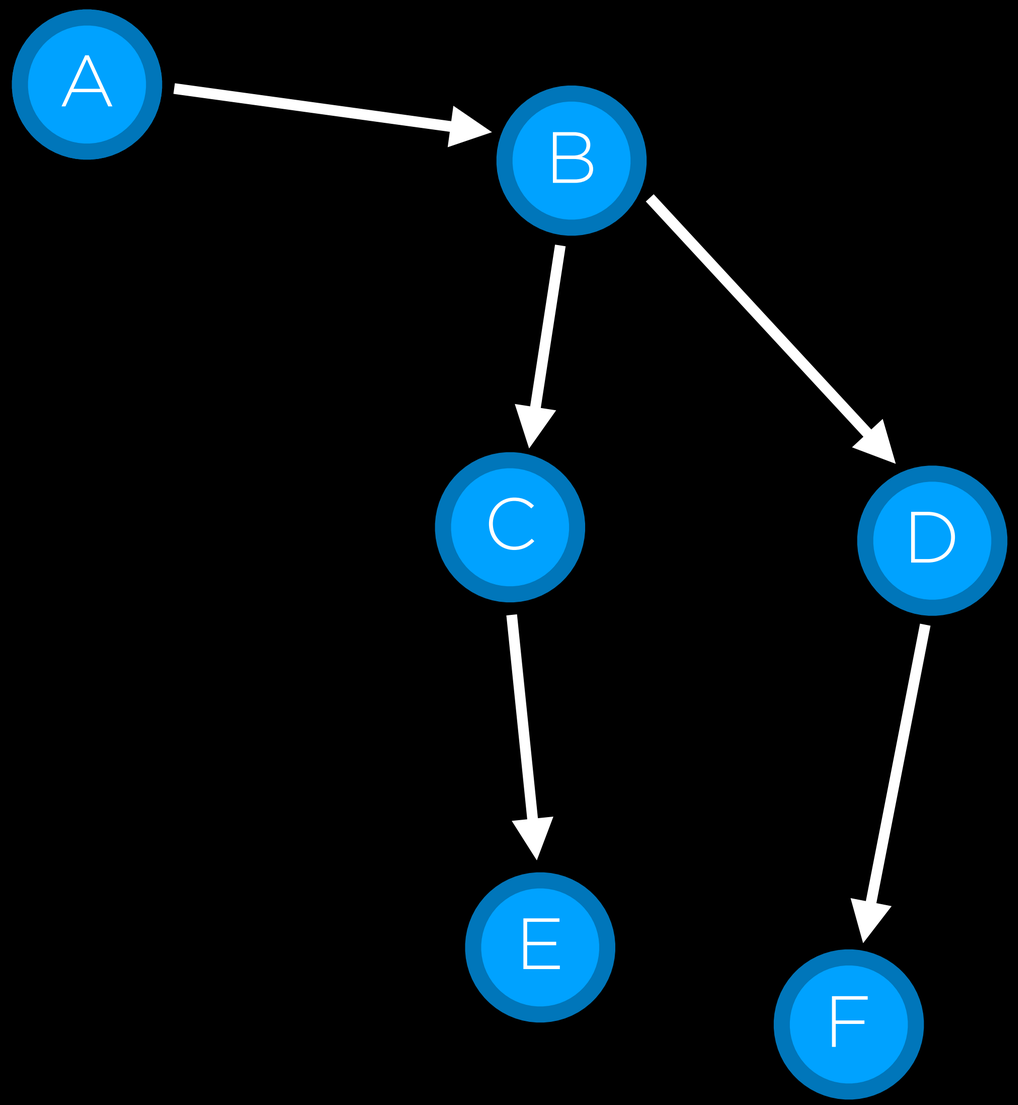
# stack

last-in first-out data type

Find a path from A to E.

**Frontier**

**Explored Set**

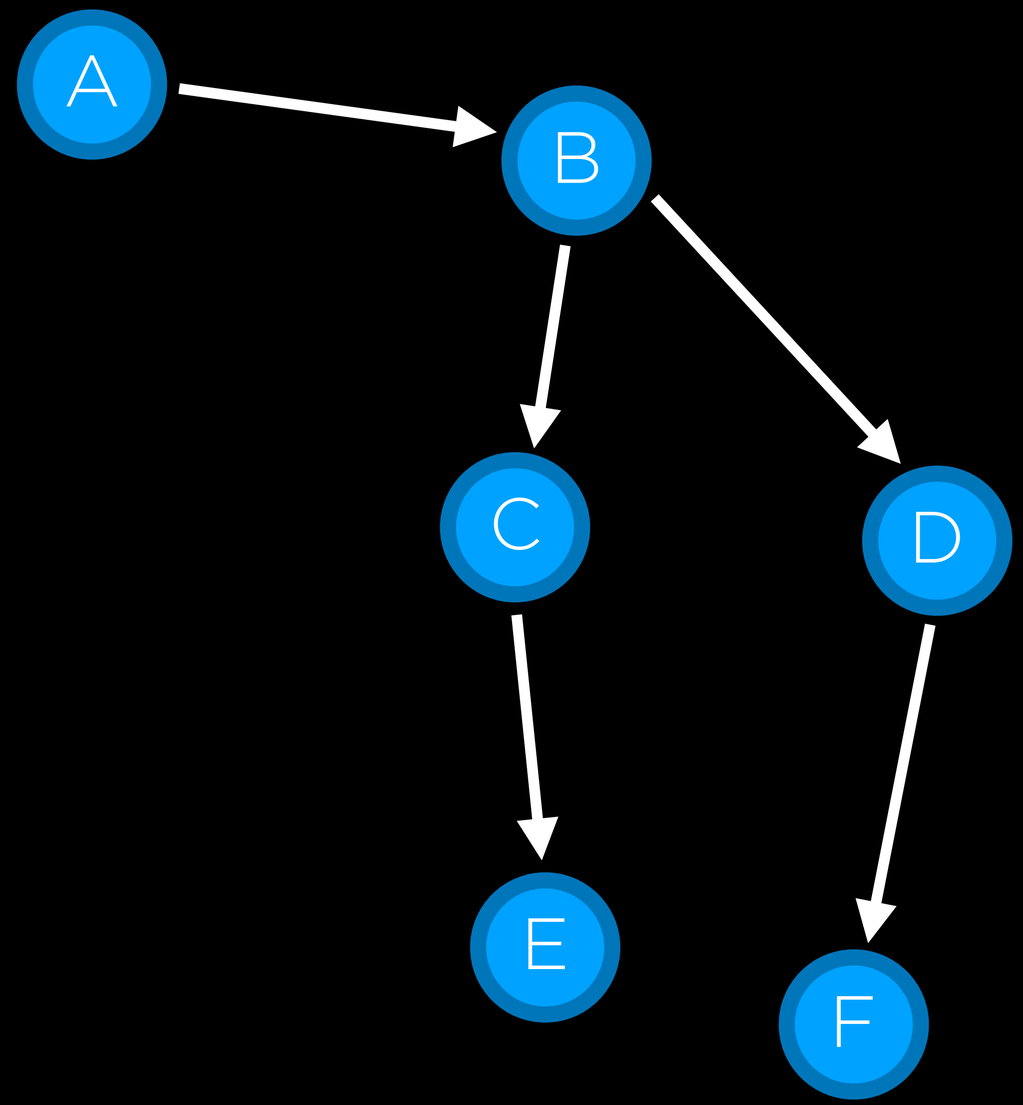


Find a path from A to E.

**Frontier**

A

**Explored Set**

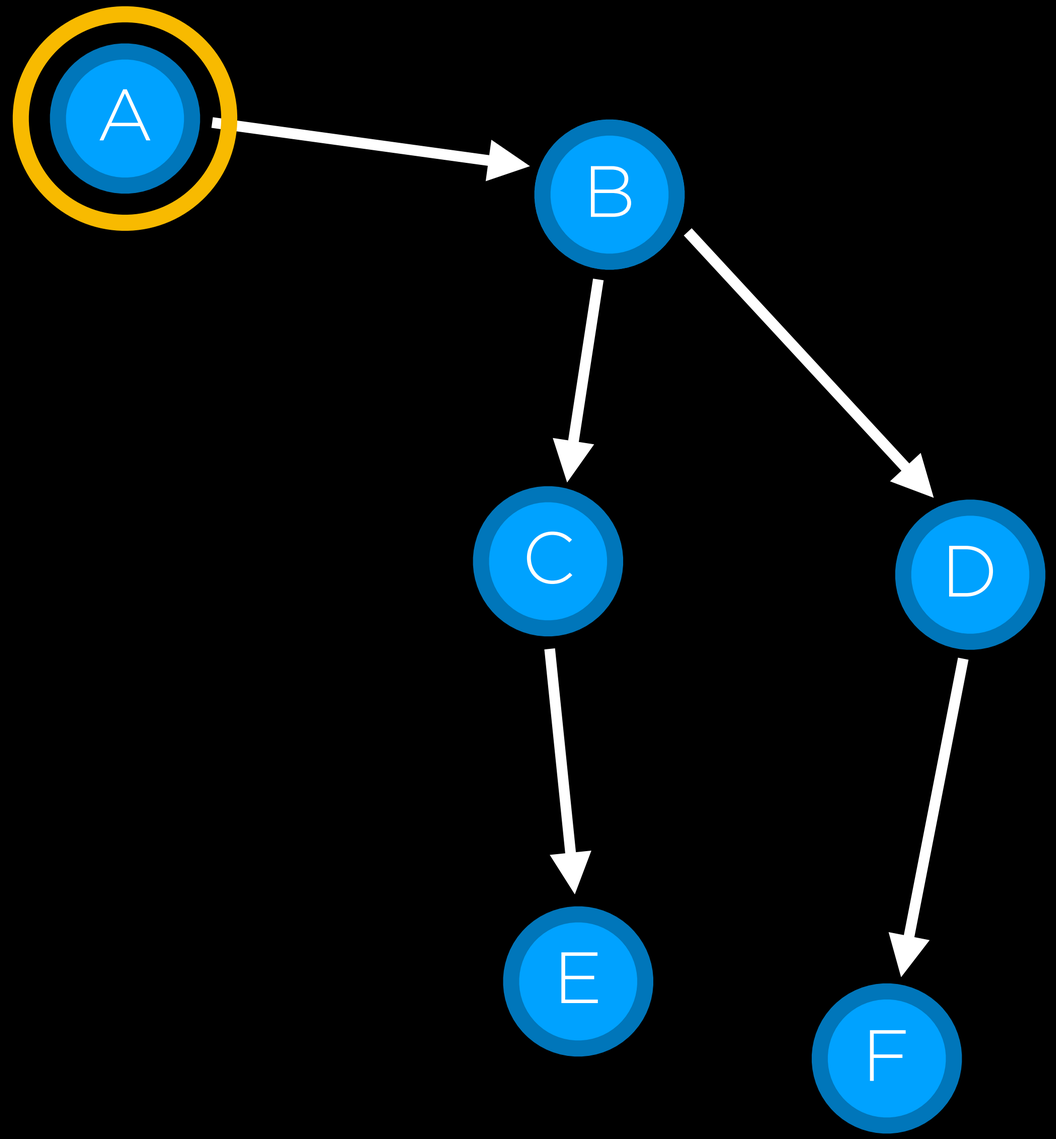


Find a path from A to E.

**Frontier**

**Explored Set**

A



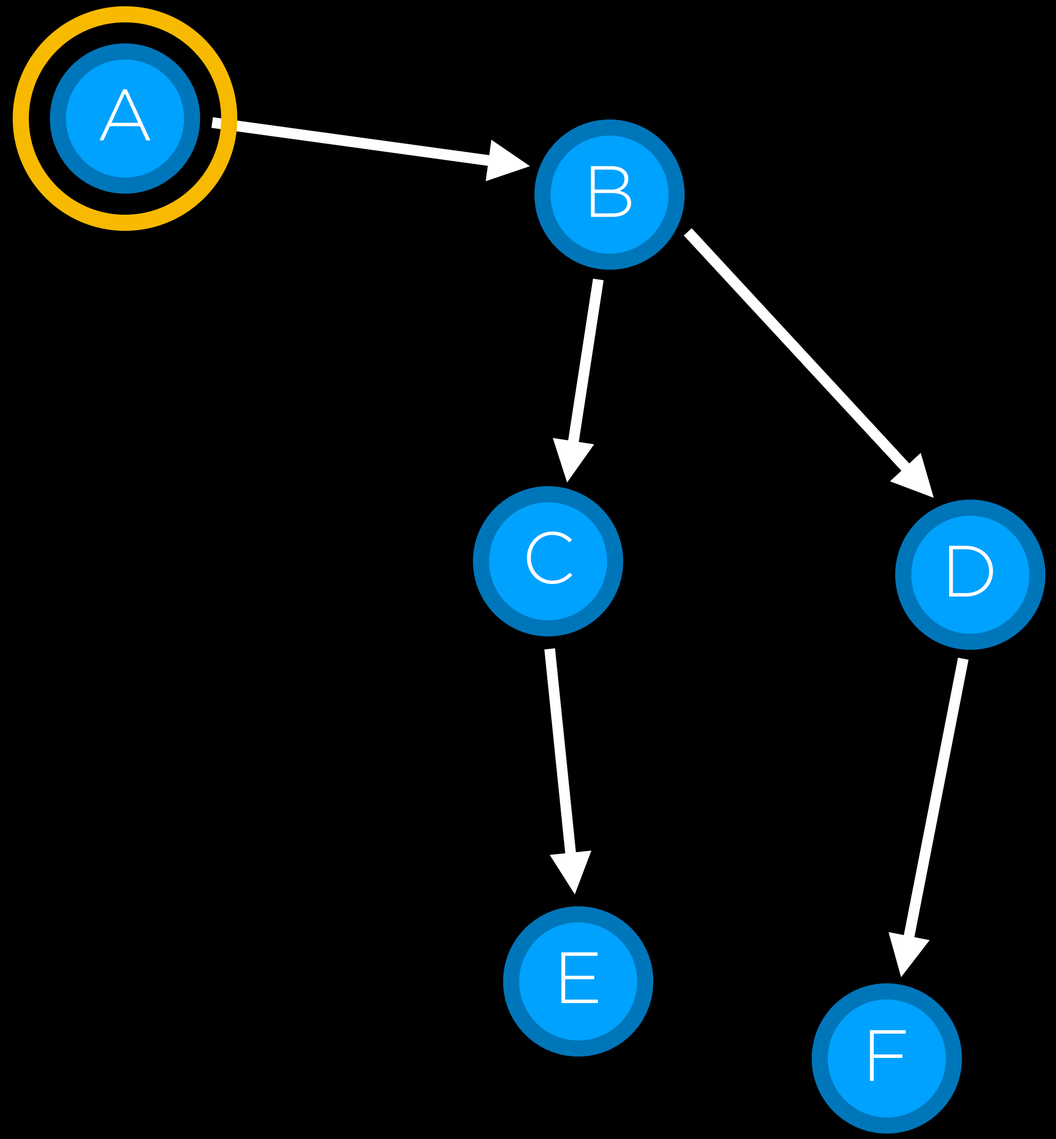
Find a path from A to E.

**Frontier**

B

**Explored Set**

A

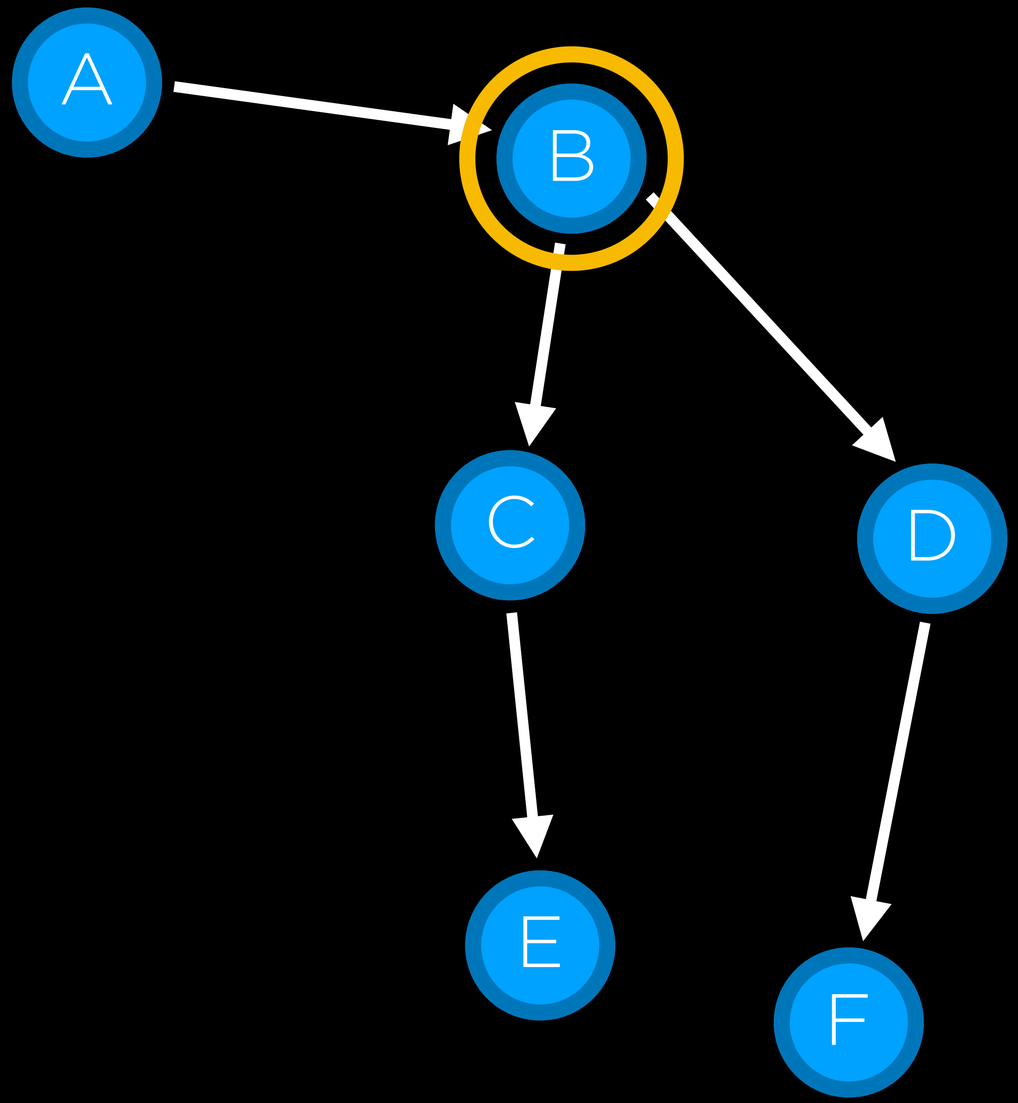


Find a path from A to E.

**Frontier**

**Explored Set**

A B



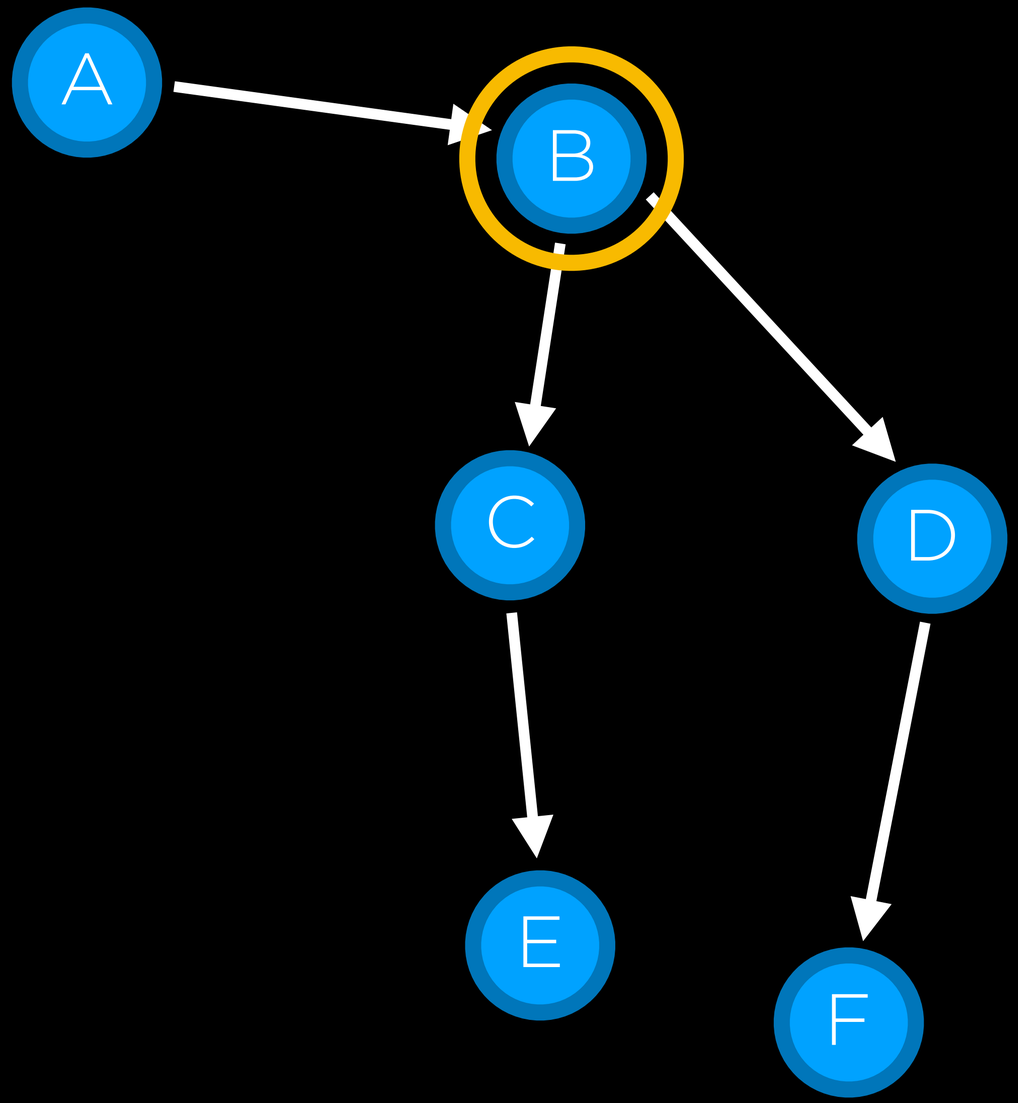
Find a path from A to E.

**Frontier**

C D

**Explored Set**

A B



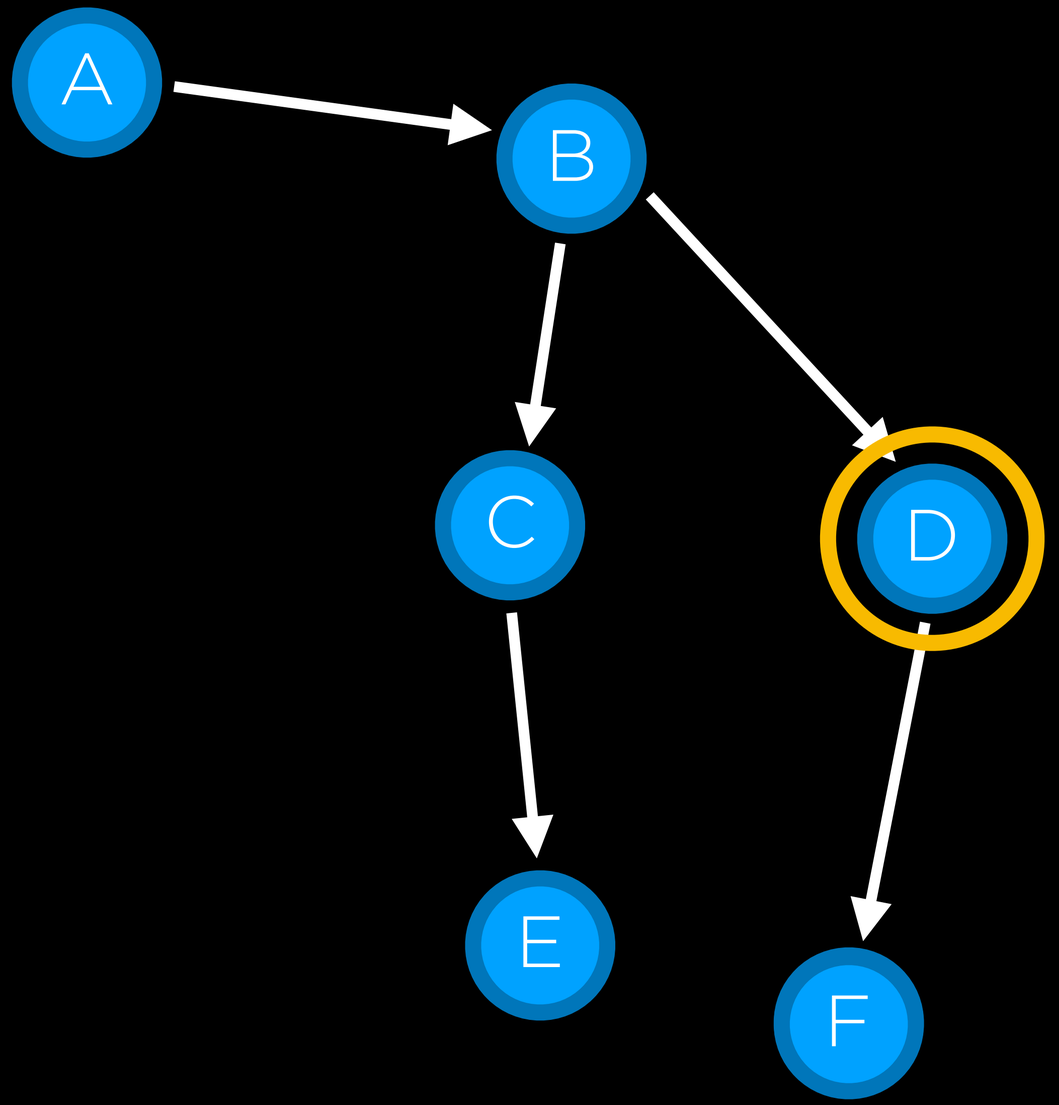
Find a path from A to E.

**Frontier**

C

**Explored Set**

A B D



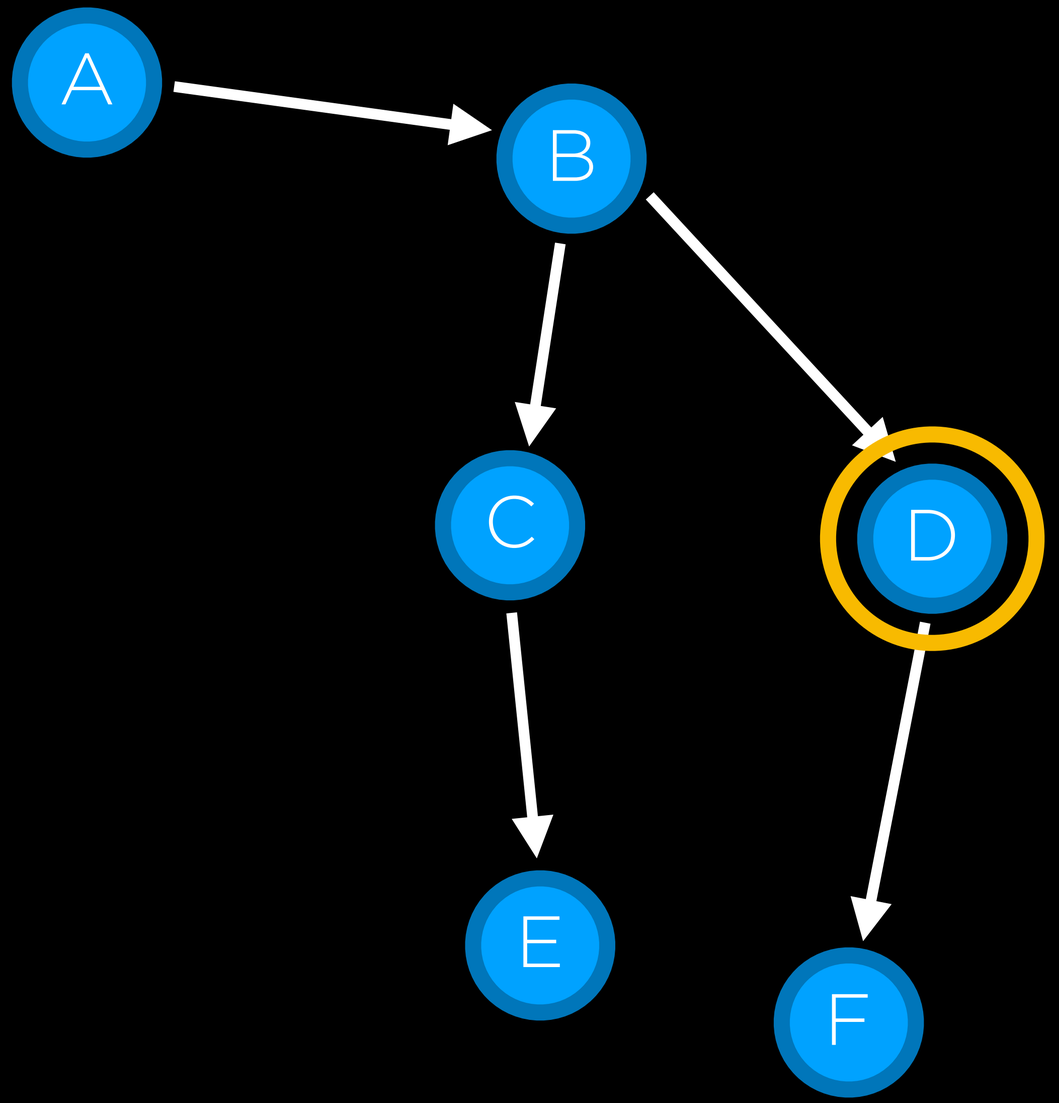
Find a path from A to E.

**Frontier**

C F

**Explored Set**

A B D



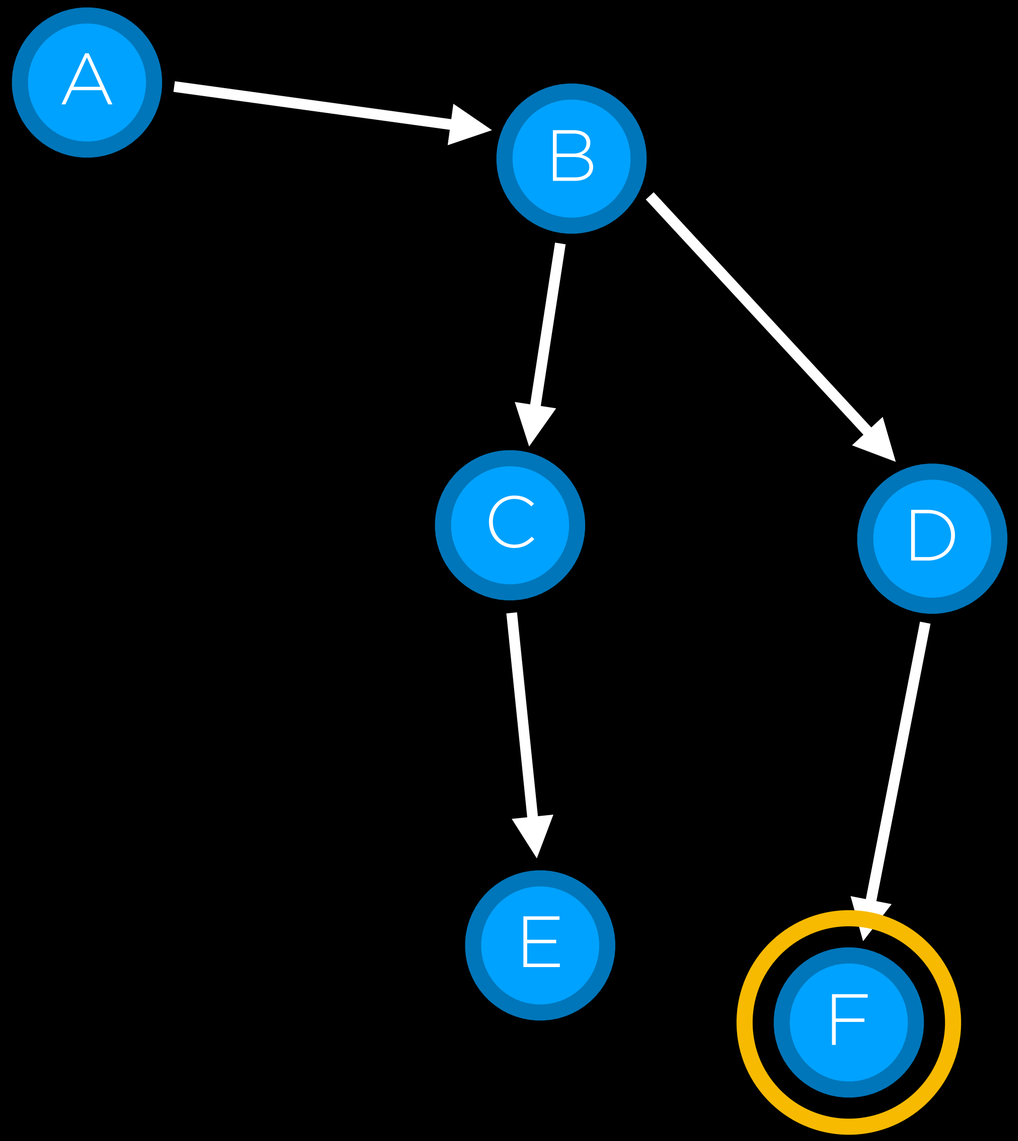
Find a path from A to E.

**Frontier**

C

**Explored Set**

A B D F

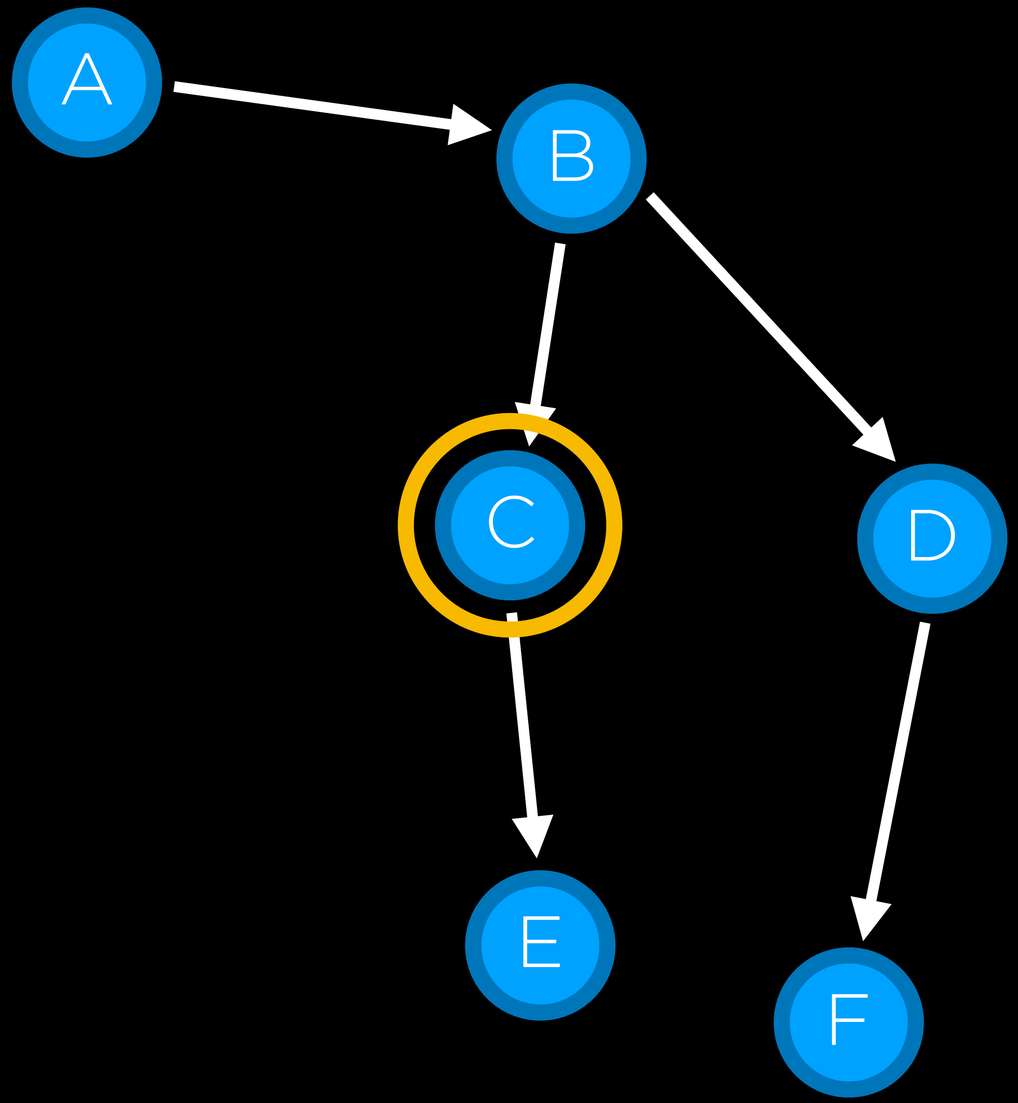


Find a path from A to E.

**Frontier**

**Explored Set**

A B D F C



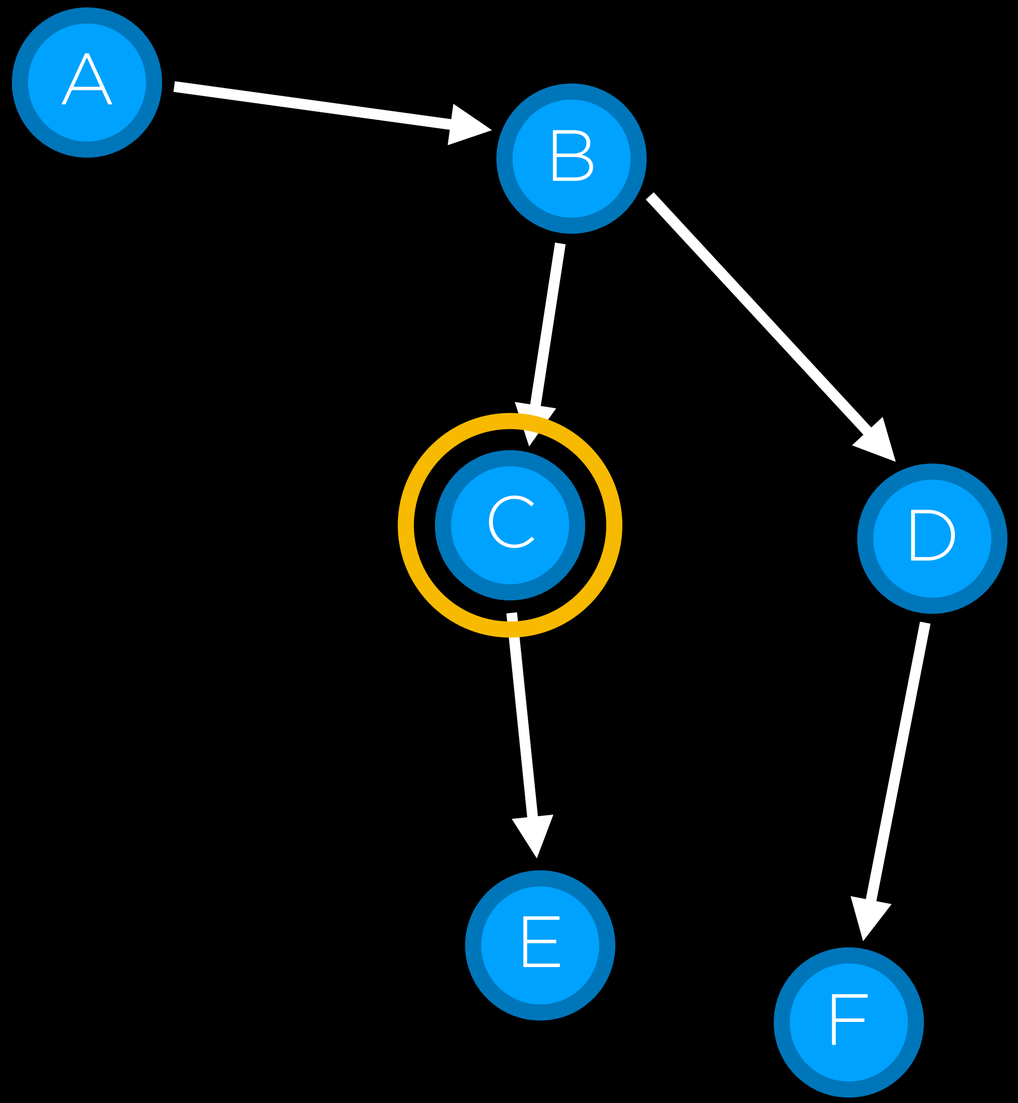
Find a path from A to E.

**Frontier**

E

**Explored Set**

A B D F C

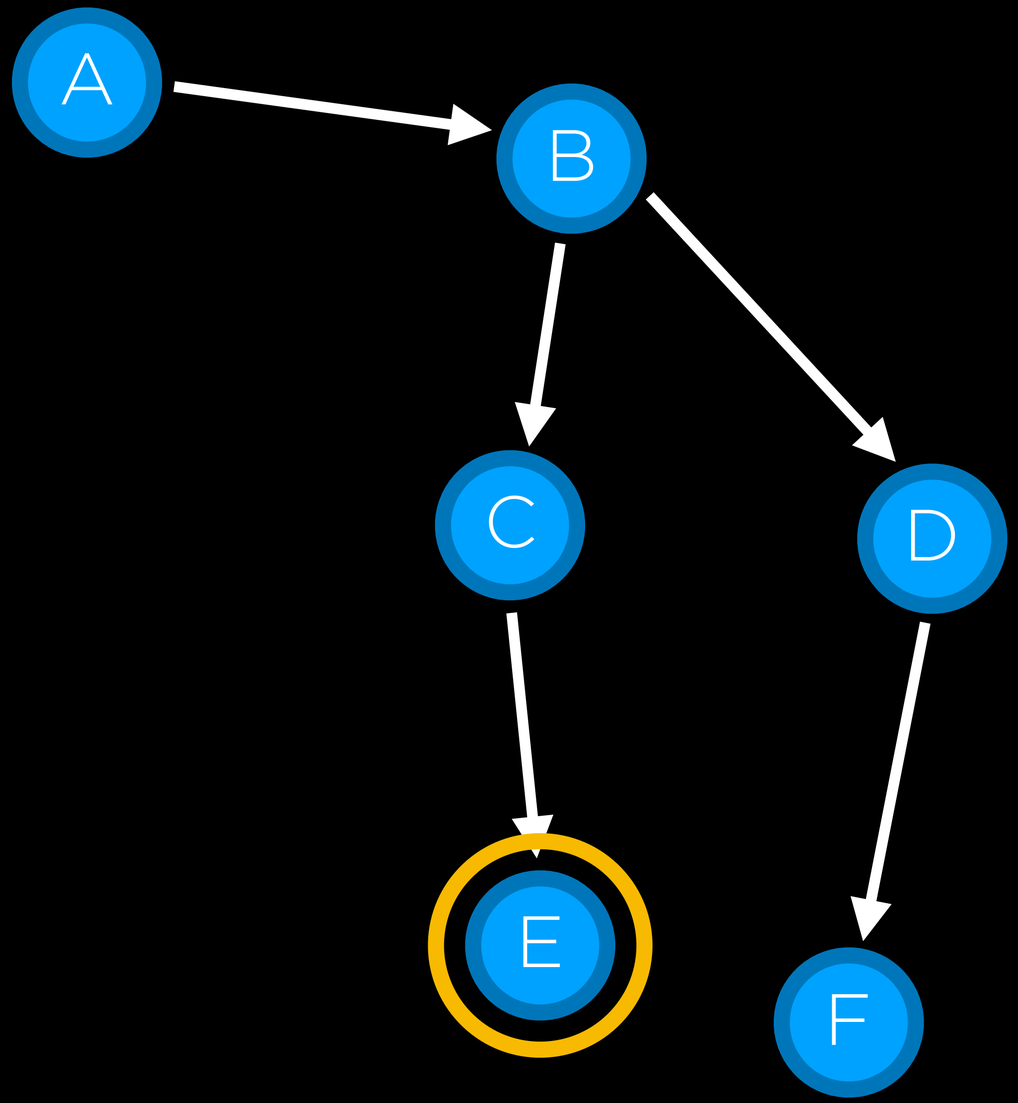


Find a path from A to E.

**Frontier**

**Explored Set**

A B D F C



# 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

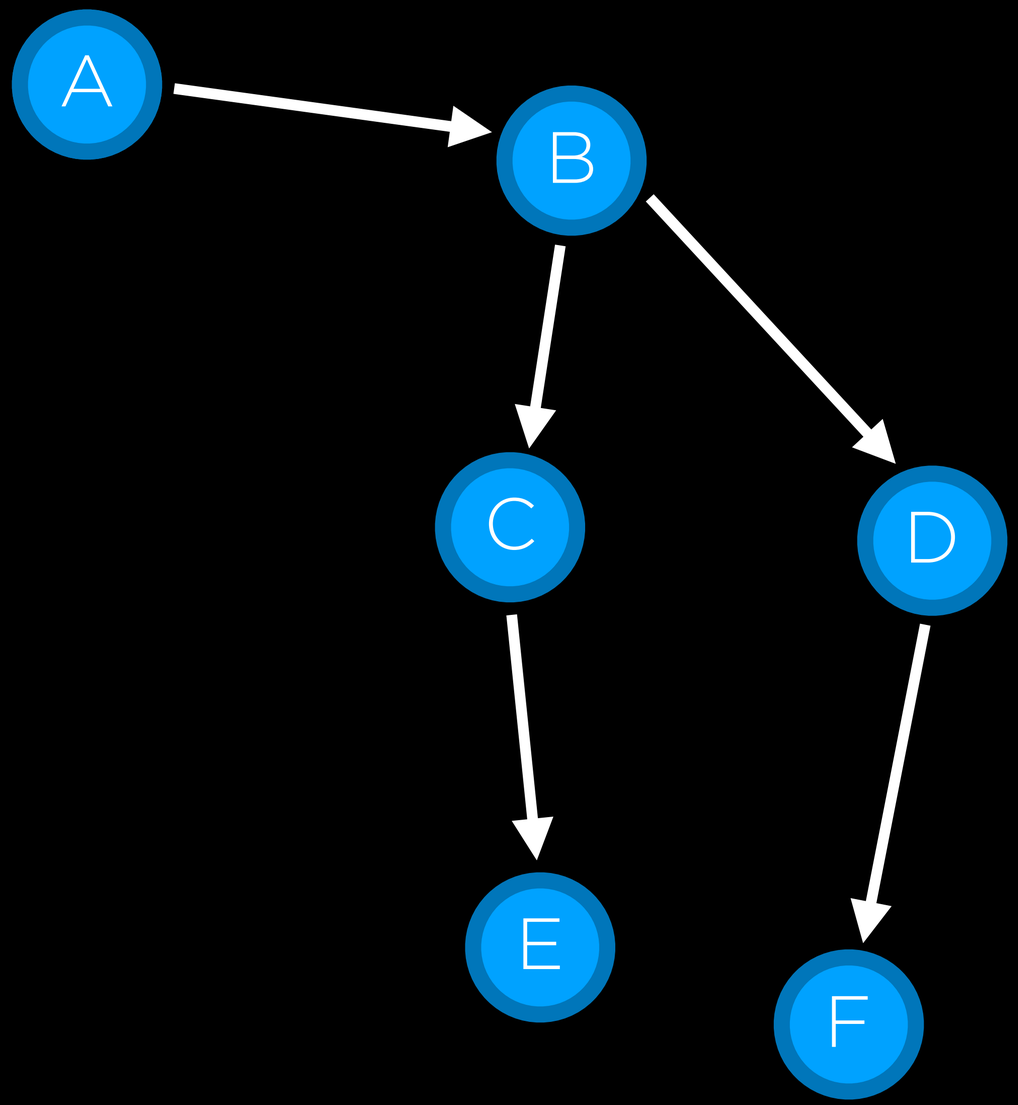
# queue

first-in first-out data type

Find a path from A to E.

**Frontier**

**Explored Set**

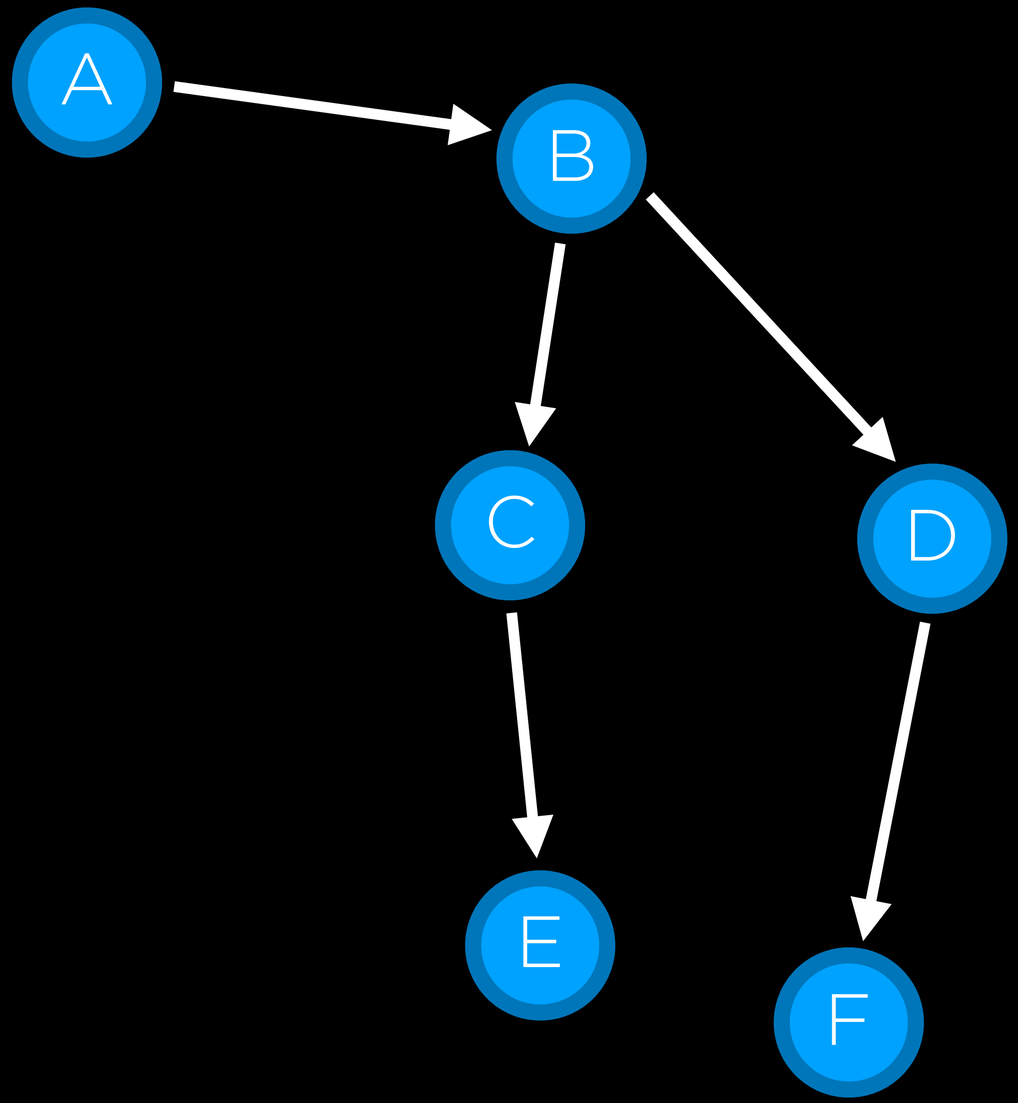


Find a path from A to E.

**Frontier**

A

**Explored Set**

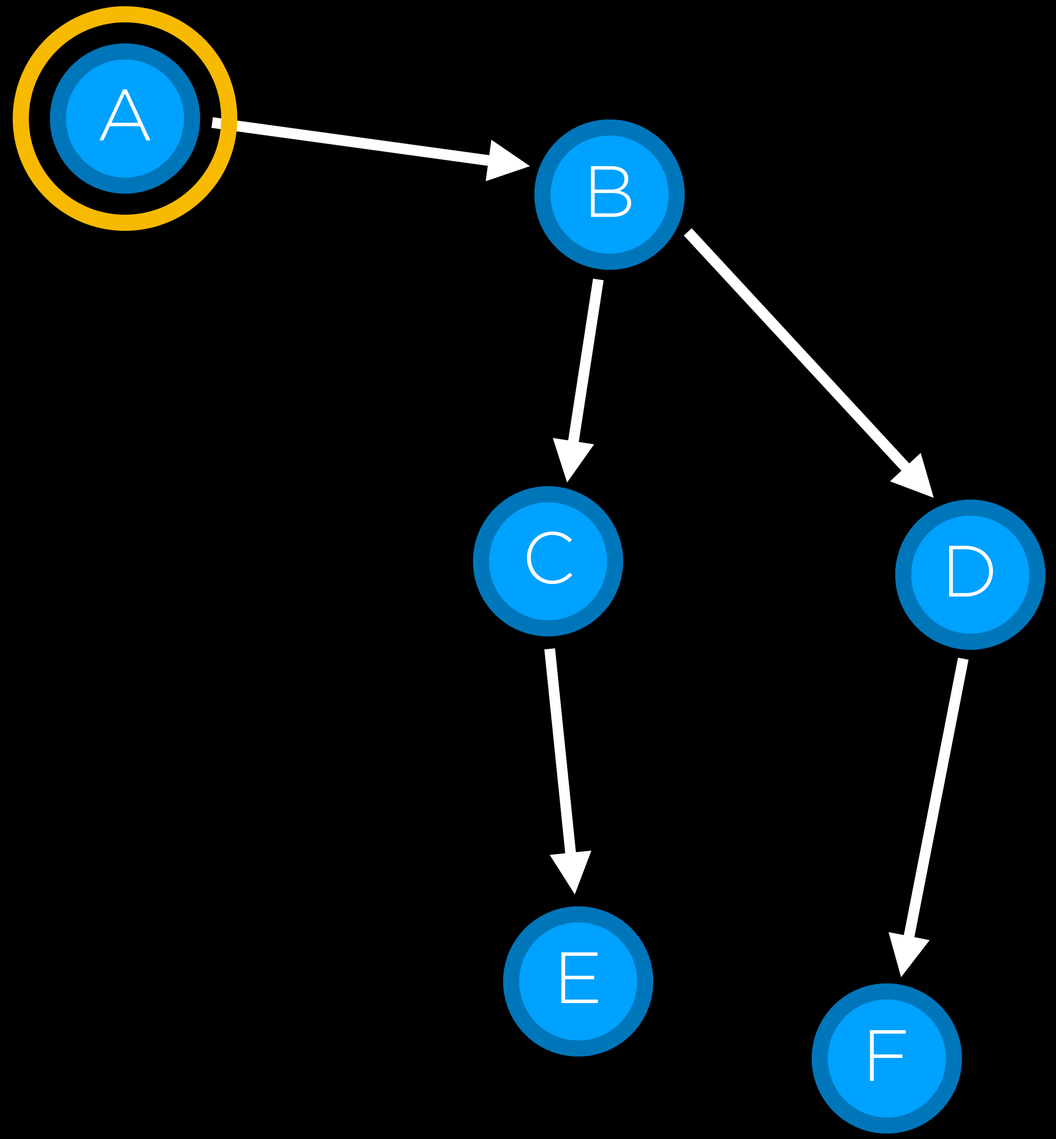


Find a path from A to E.

**Frontier**

**Explored Set**

A



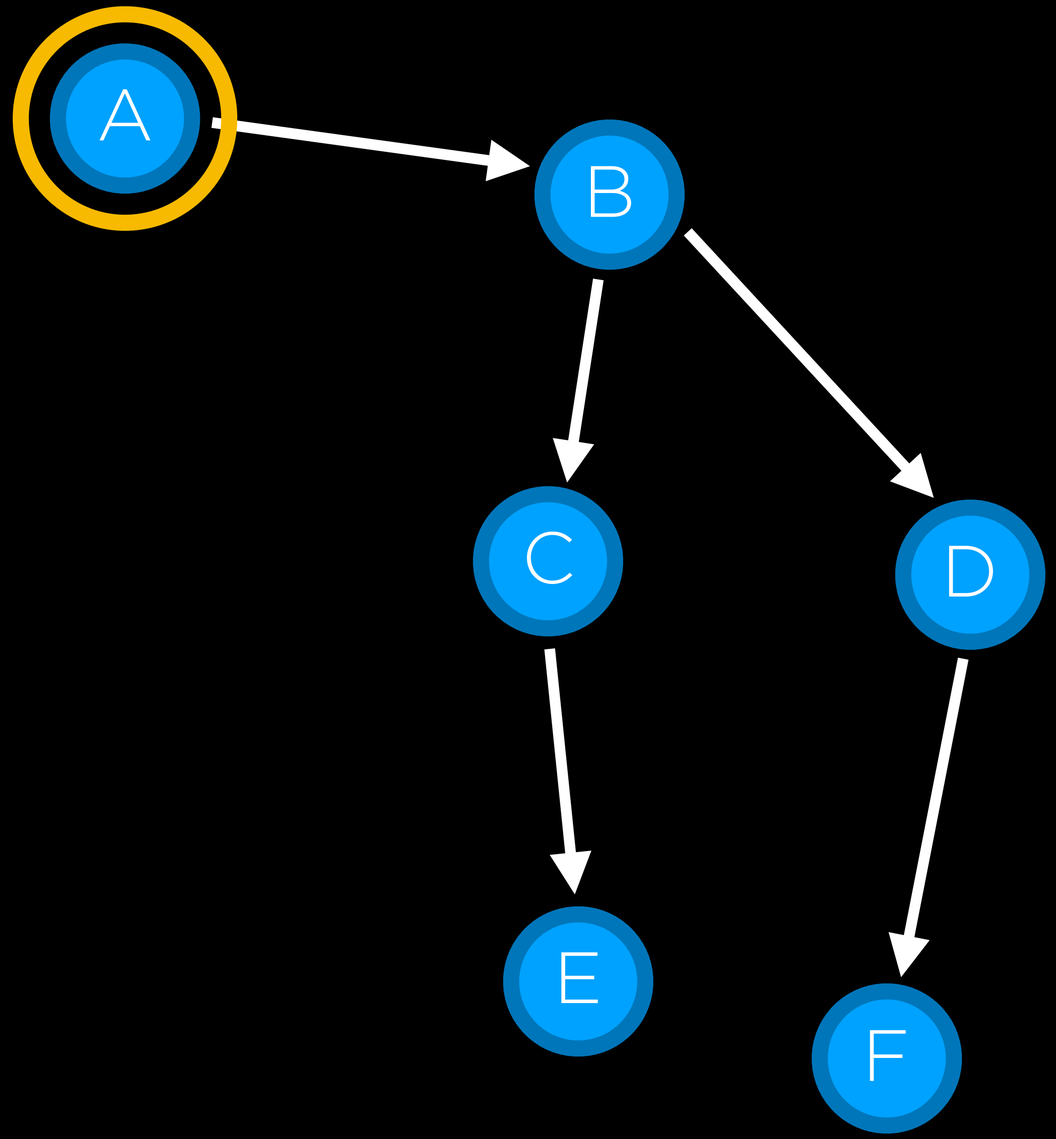
Find a path from A to E.

**Frontier**

B

**Explored Set**

A

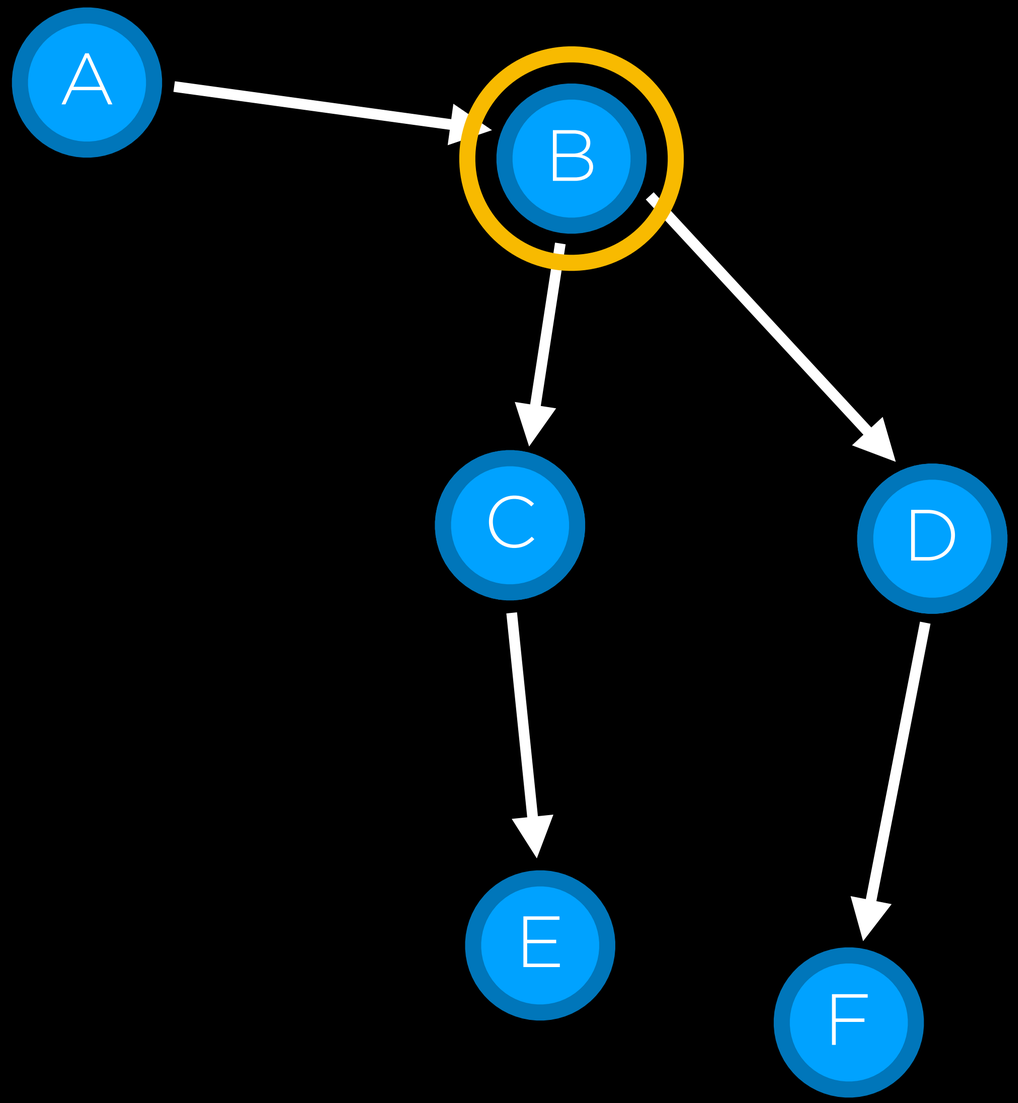


Find a path from A to E.

**Frontier**

**Explored Set**

A B



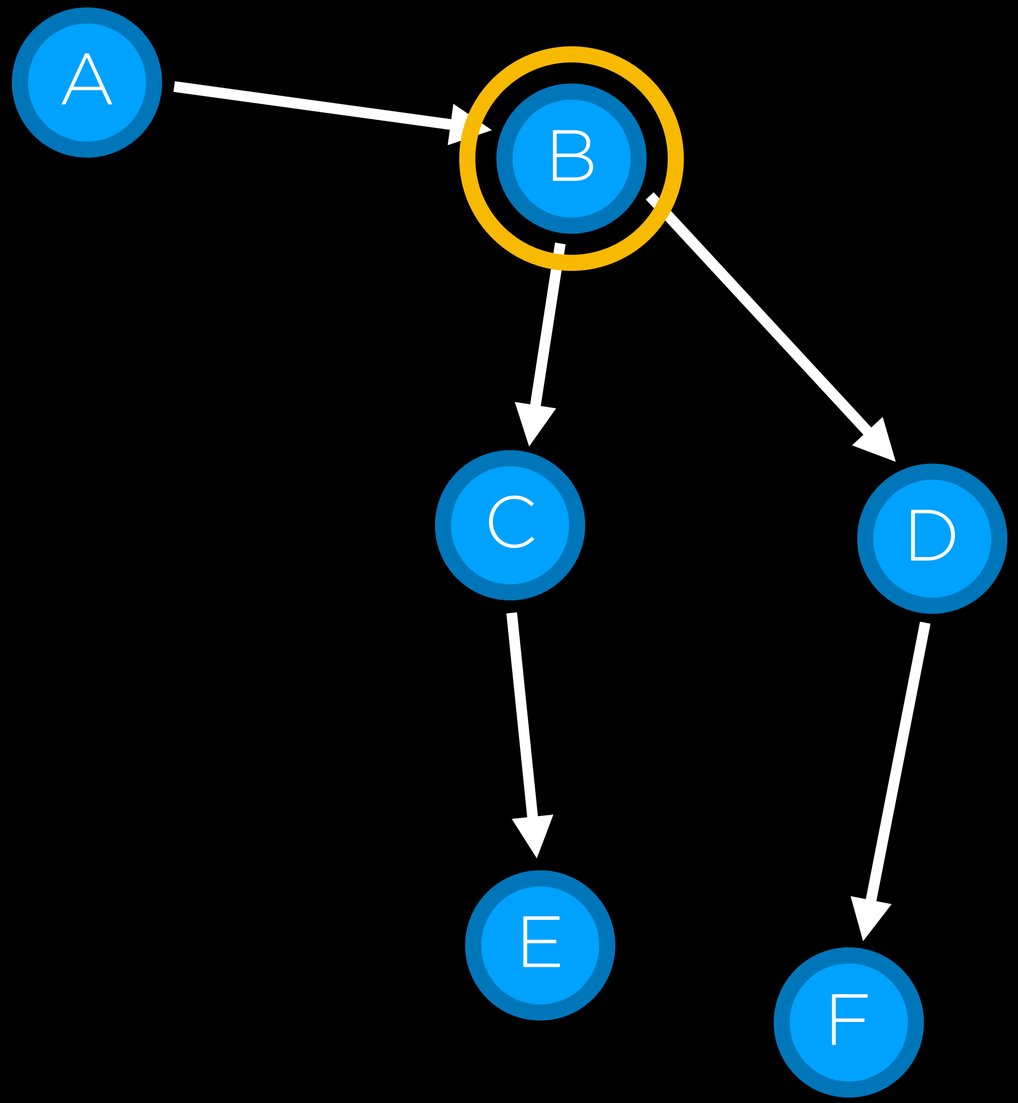
Find a path from A to E.

**Frontier**

C D

**Explored Set**

A B



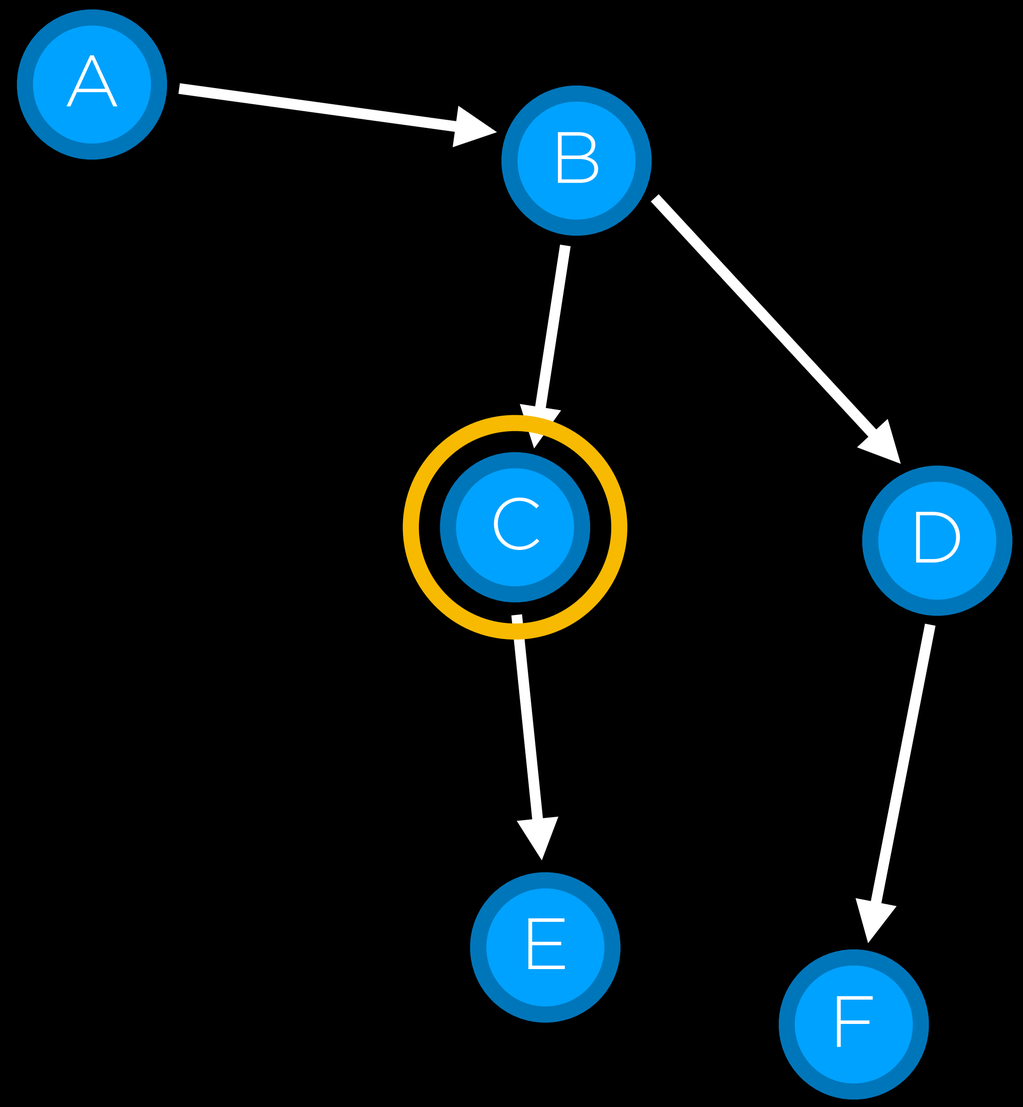
Find a path from A to E.

**Frontier**

D

**Explored Set**

A B C



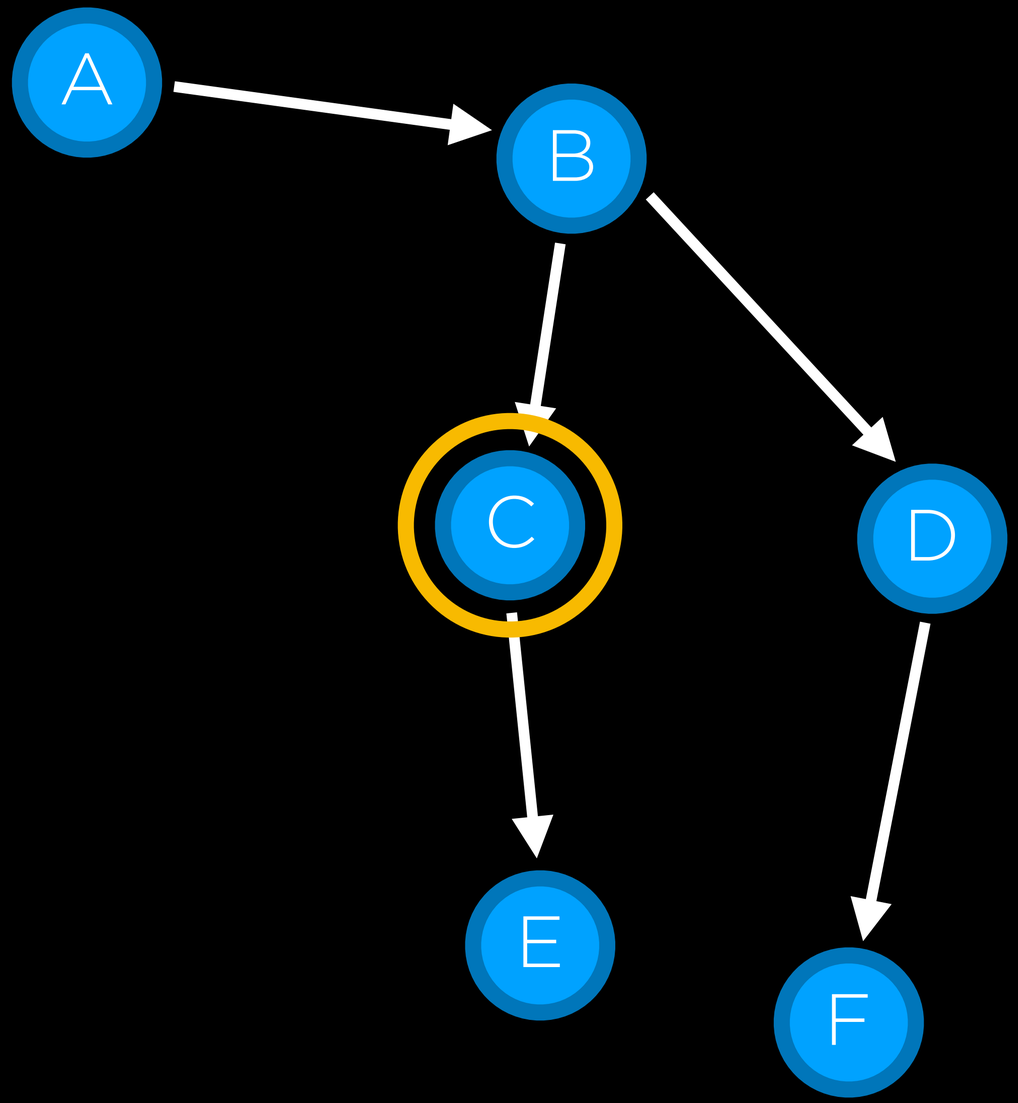
Find a path from A to E.

**Frontier**

D E

**Explored Set**

A B C



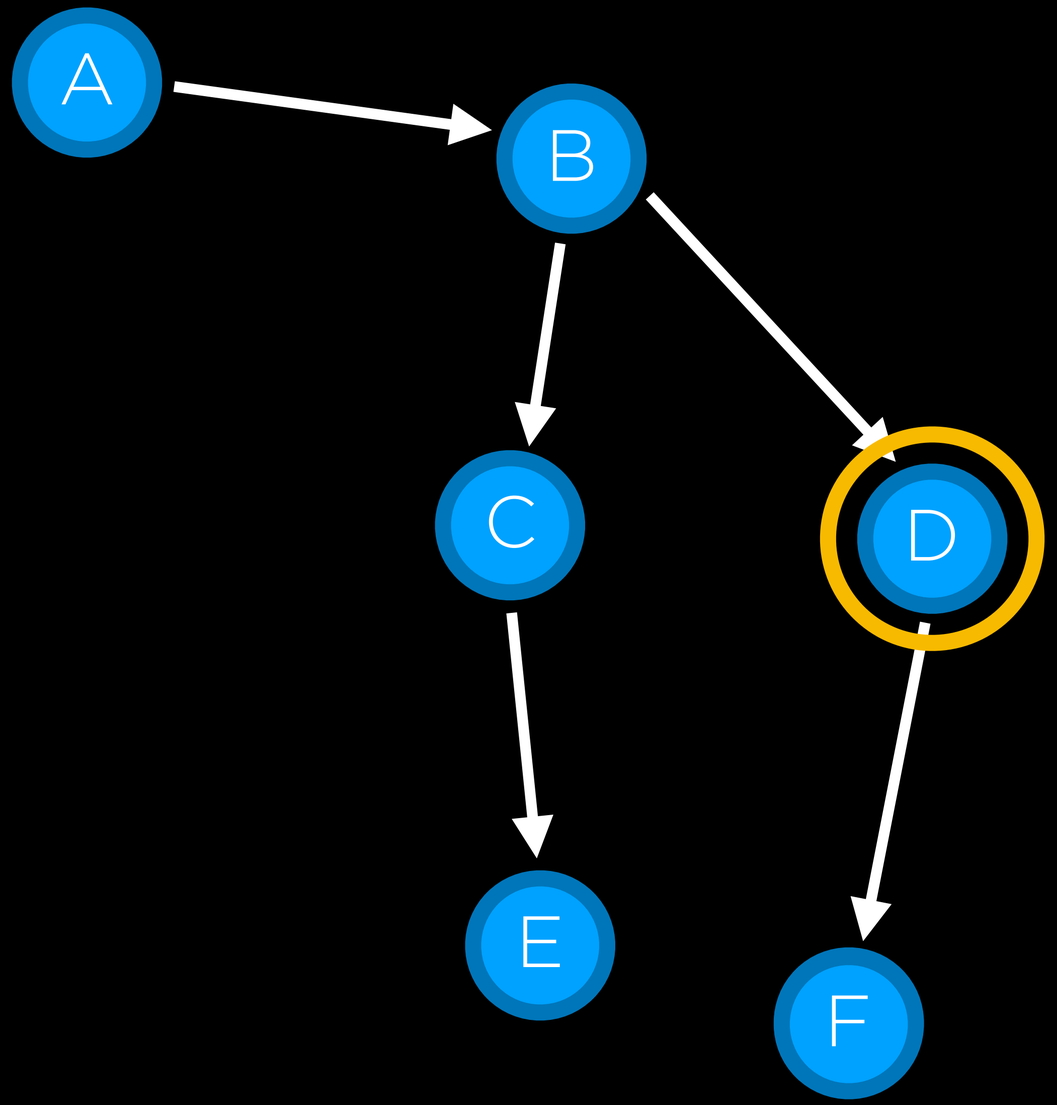
Find a path from A to E.

**Frontier**

E

**Explored Set**

A B C D



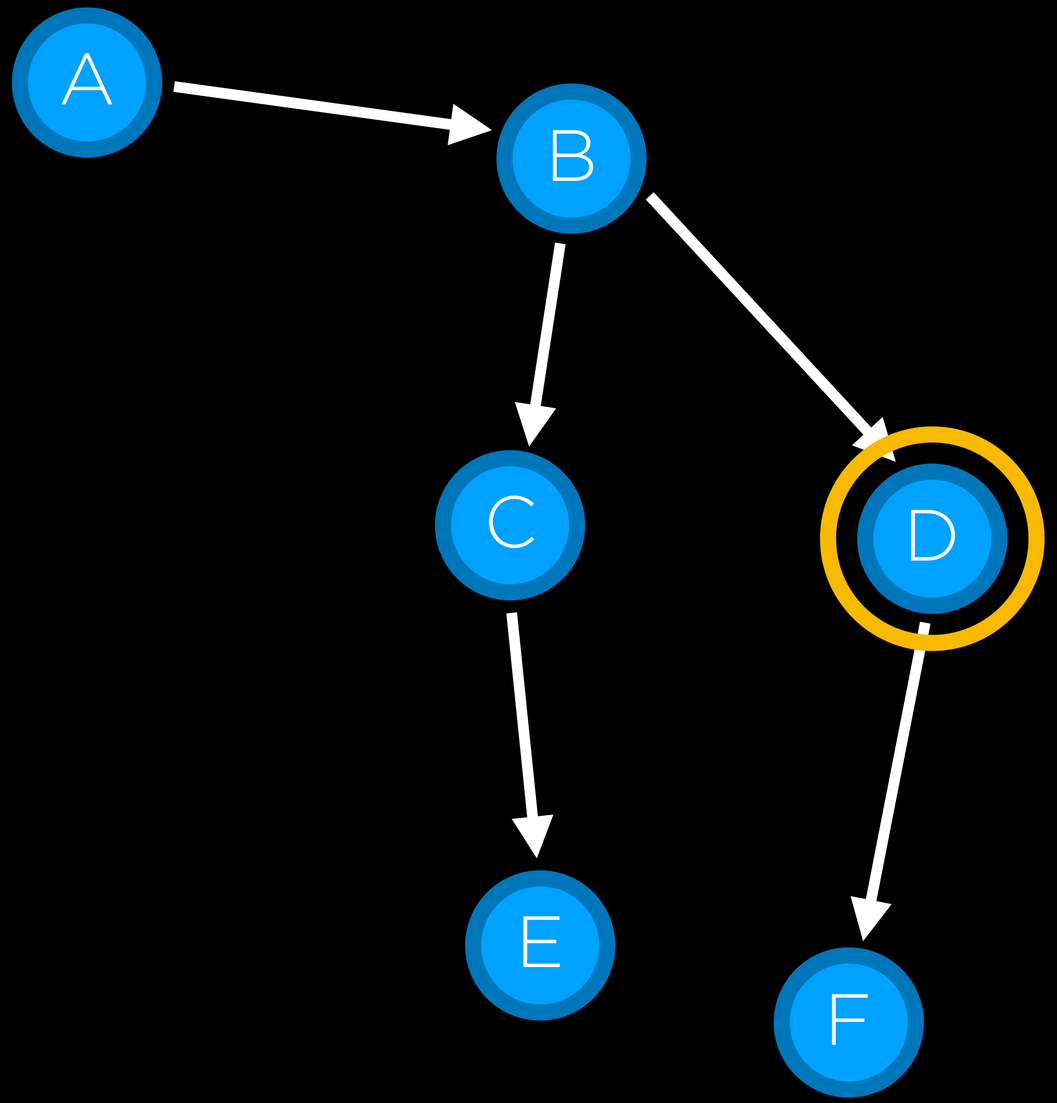
Find a path from A to E.

**Frontier**

E F

**Explored Set**

A B C D



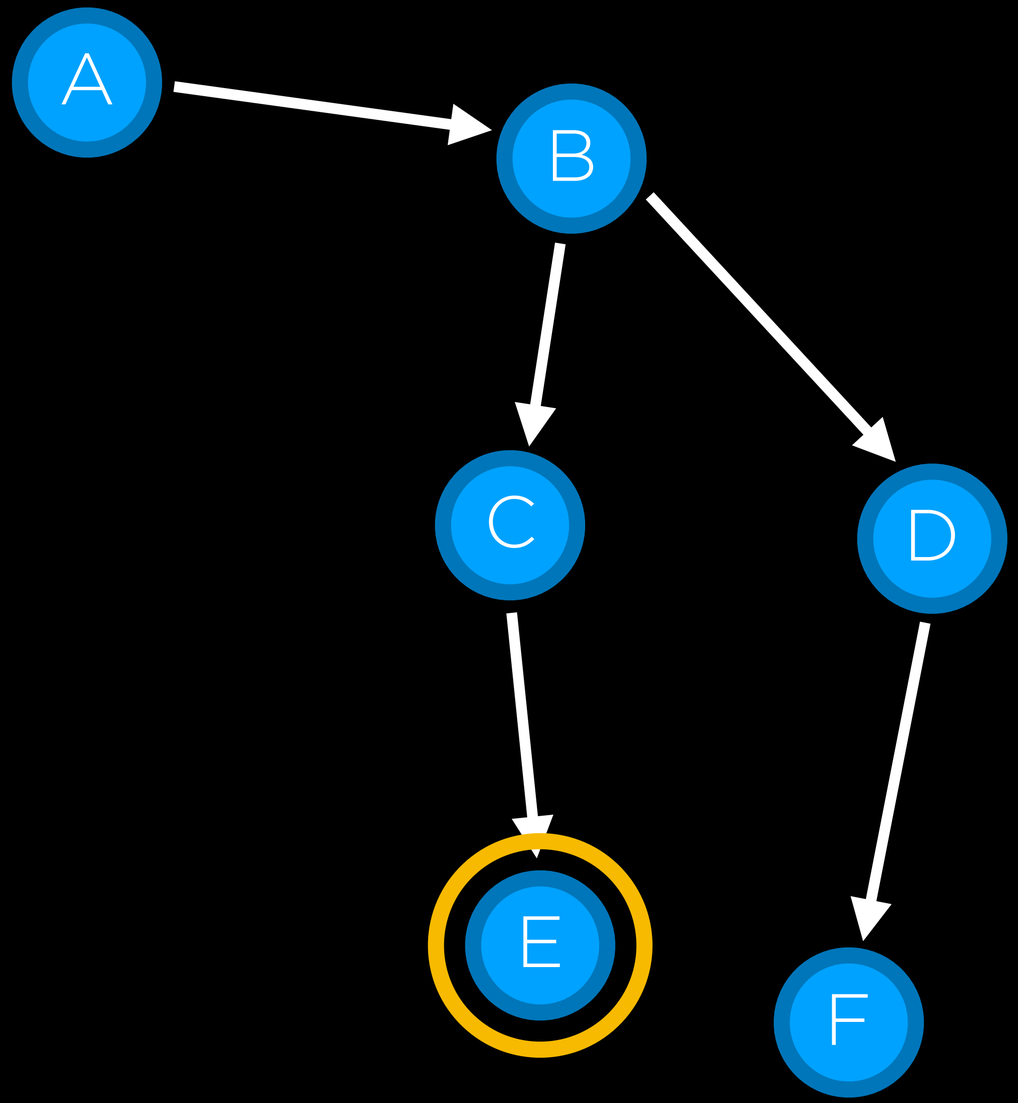
Find a path from A to E.

**Frontier**

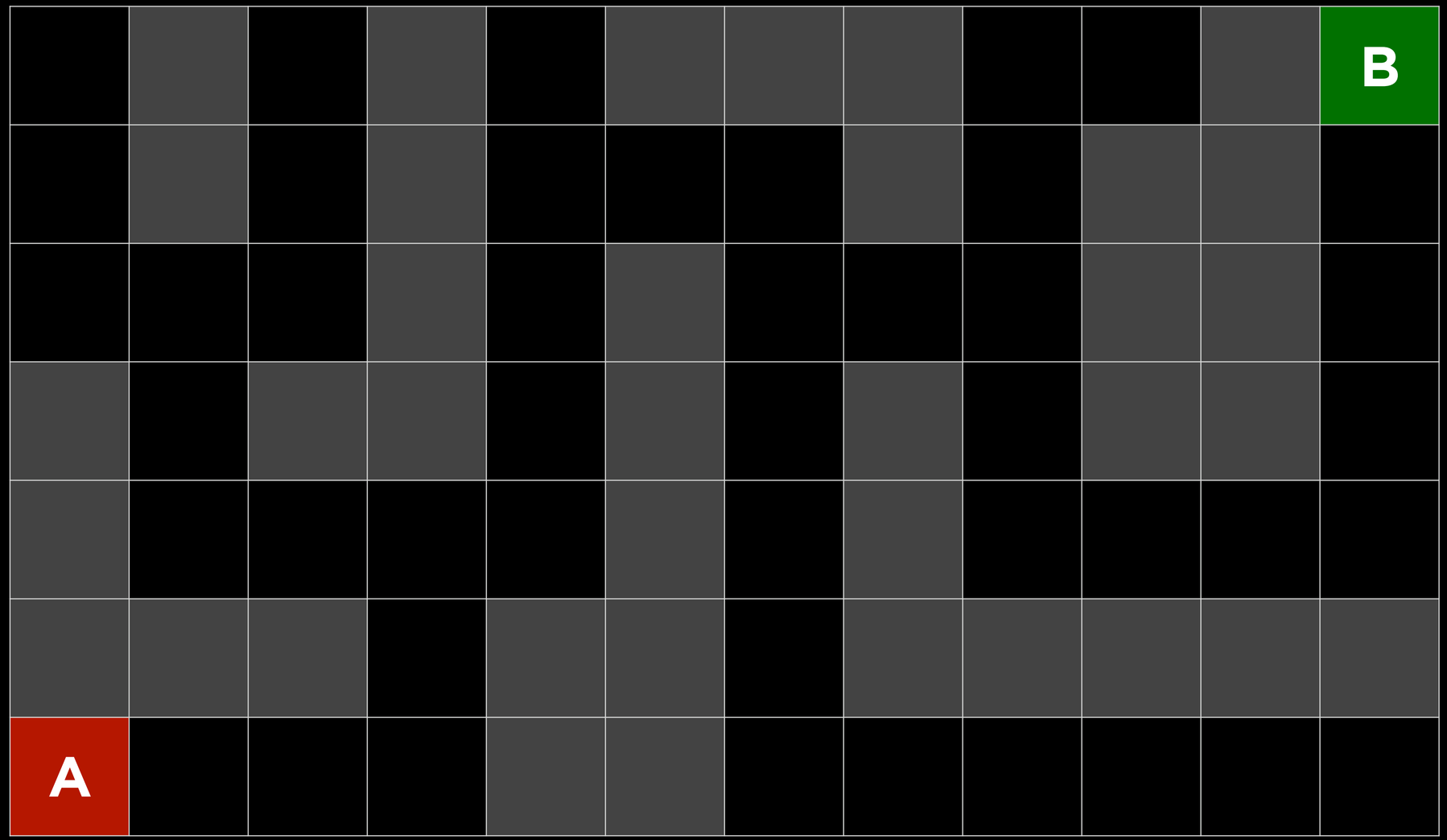
F

**Explored Set**

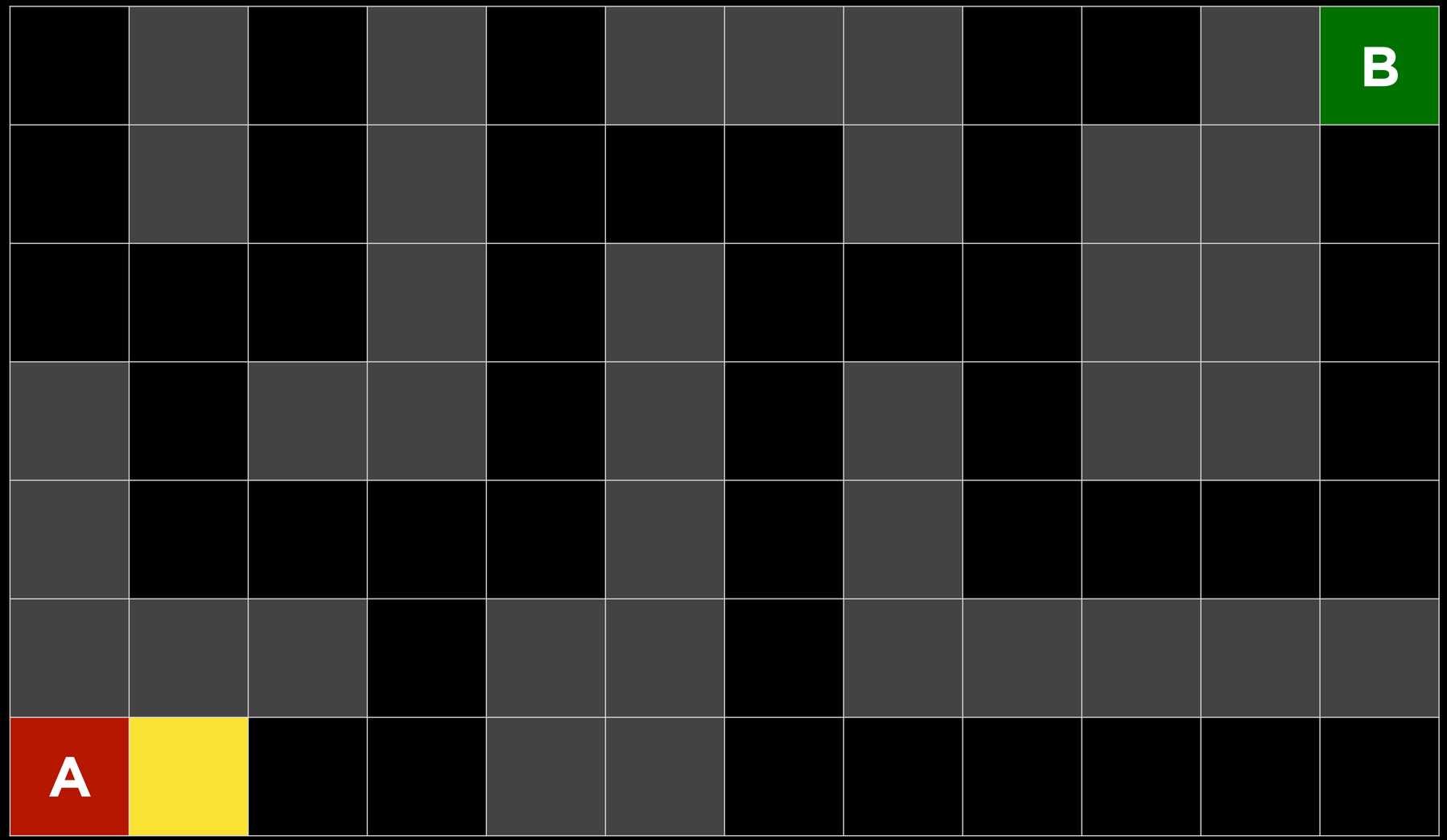
A B C D



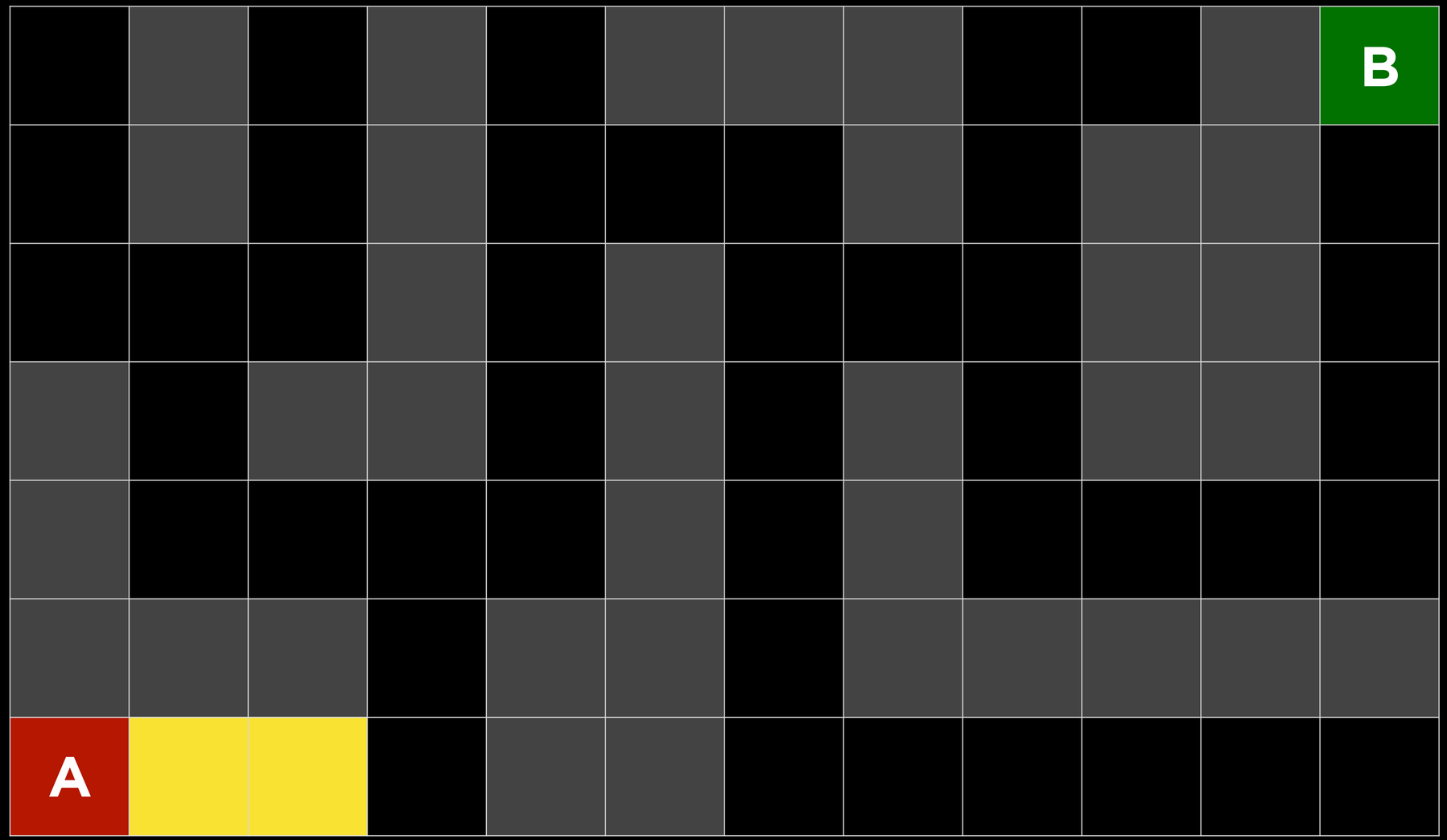
# Depth-First Search



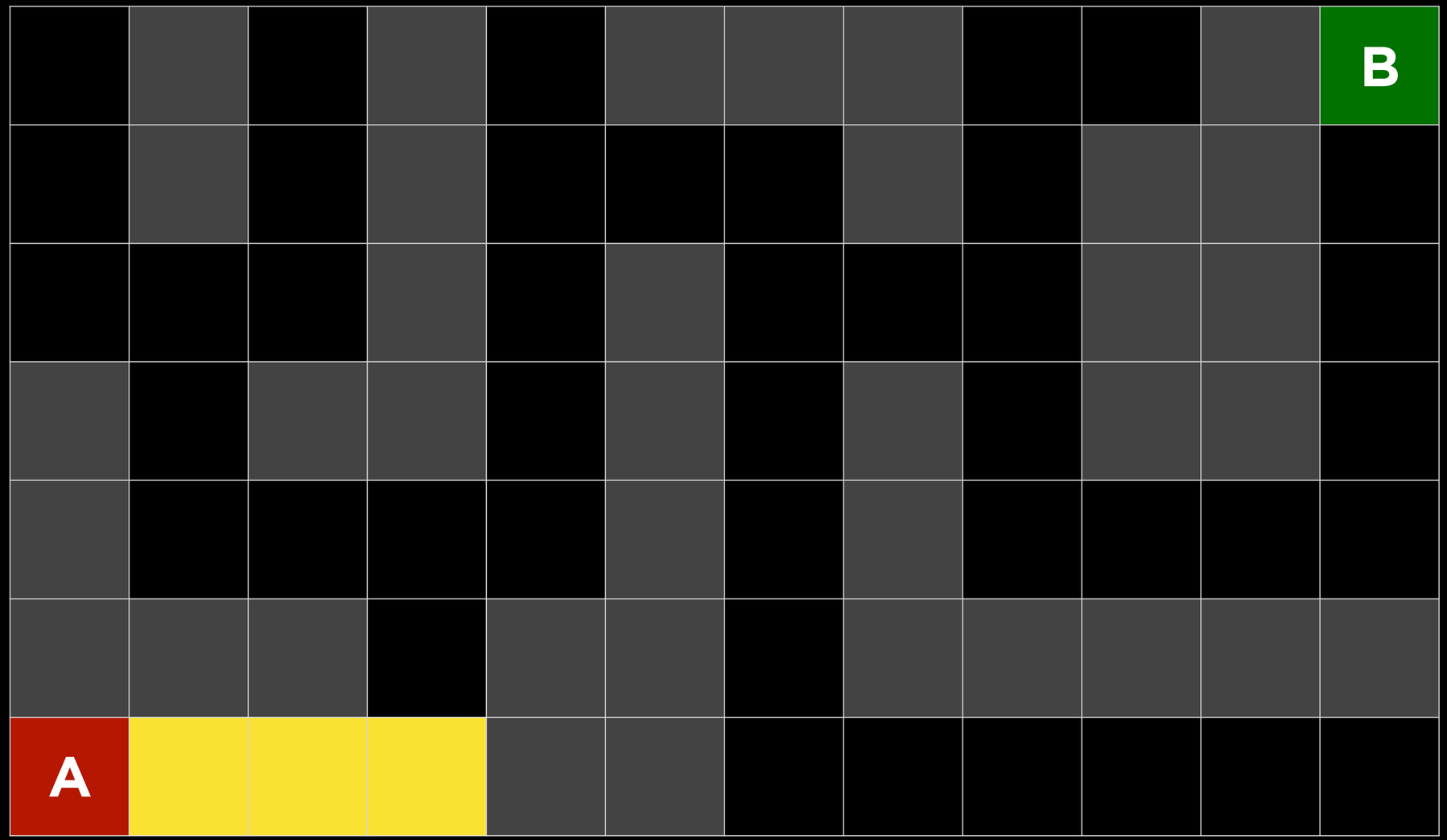
# Depth-First Search



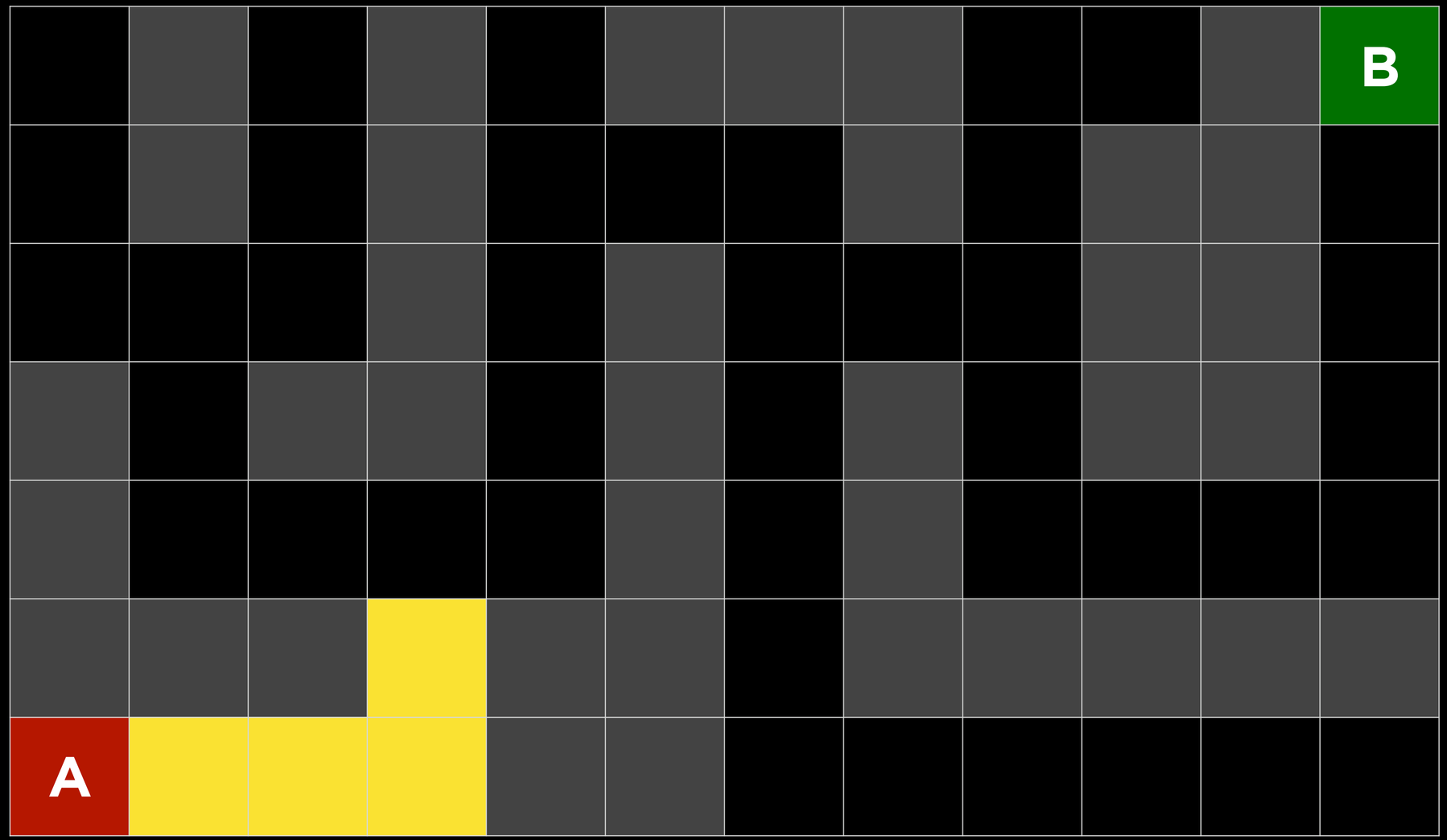
# Depth-First Search



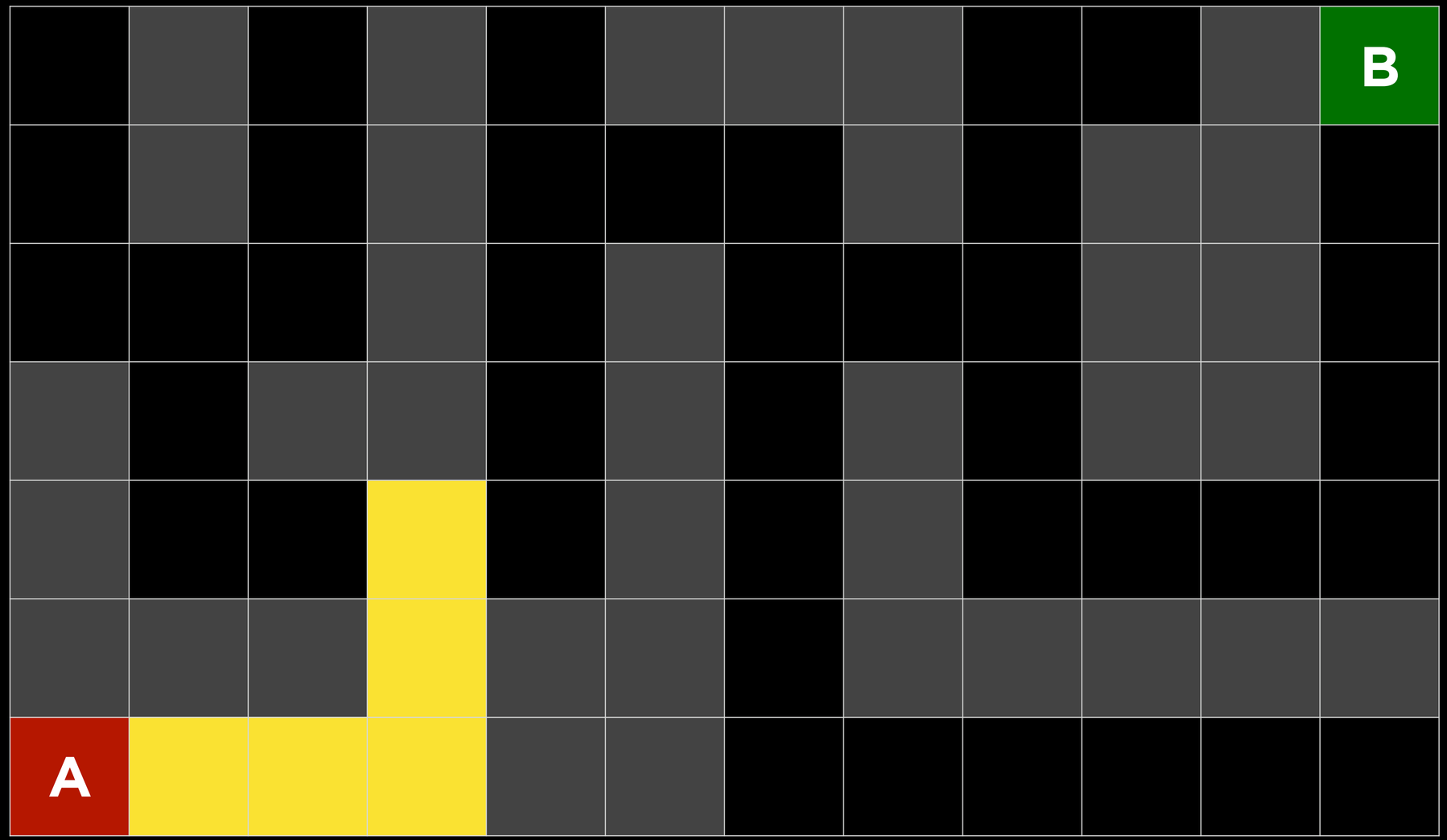
# Depth-First Search



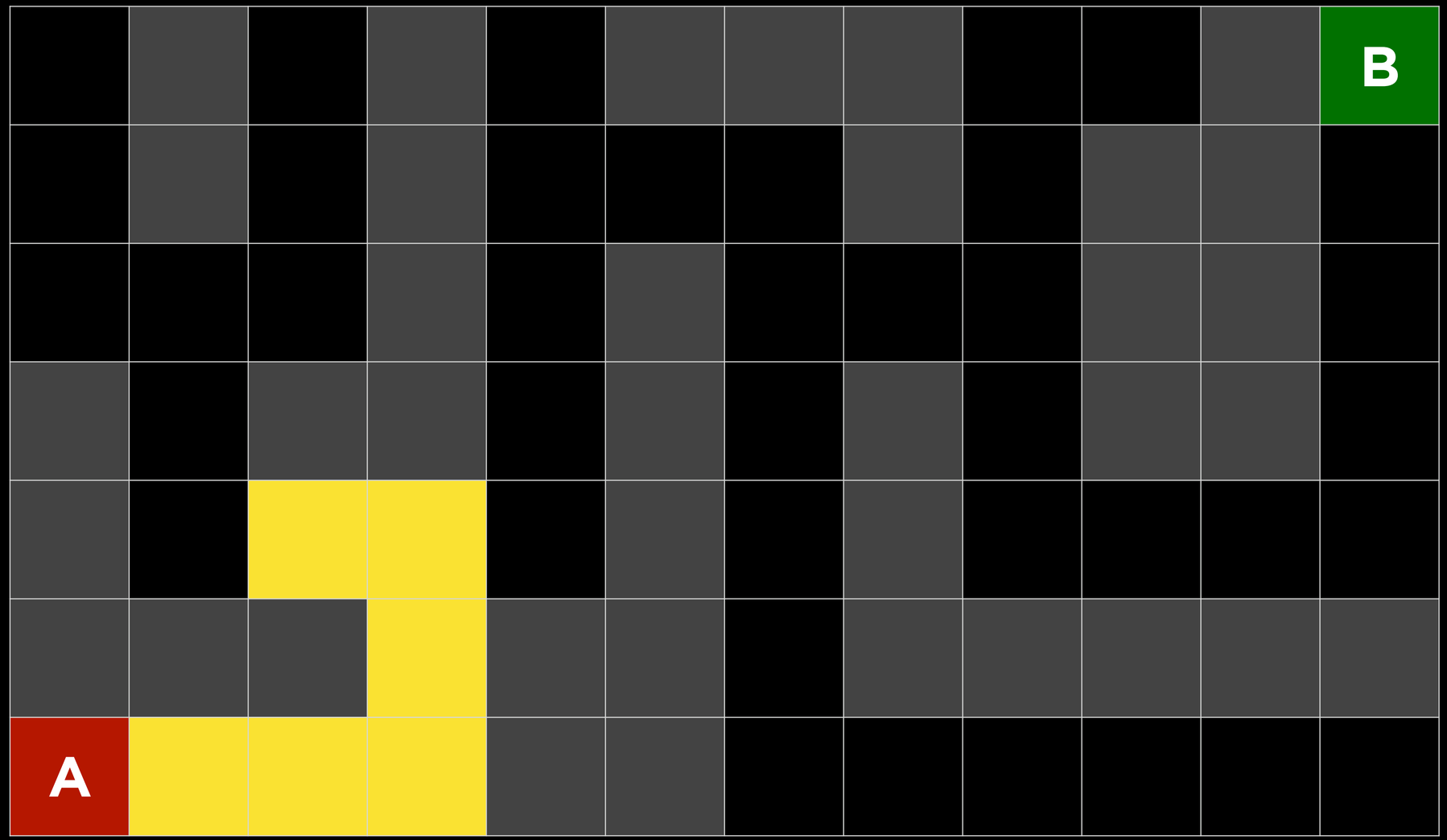
# Depth-First Search



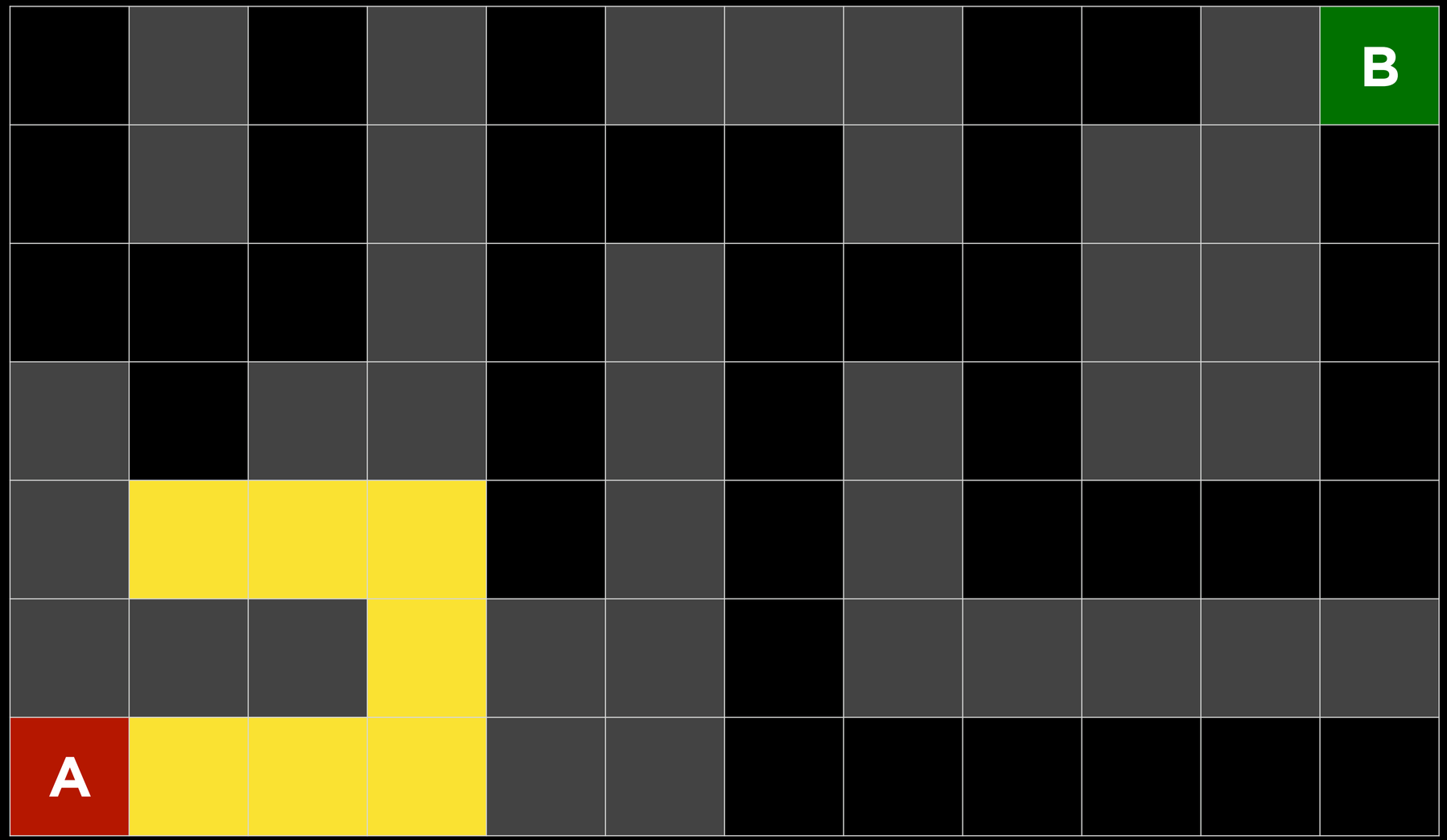
# Depth-First Search



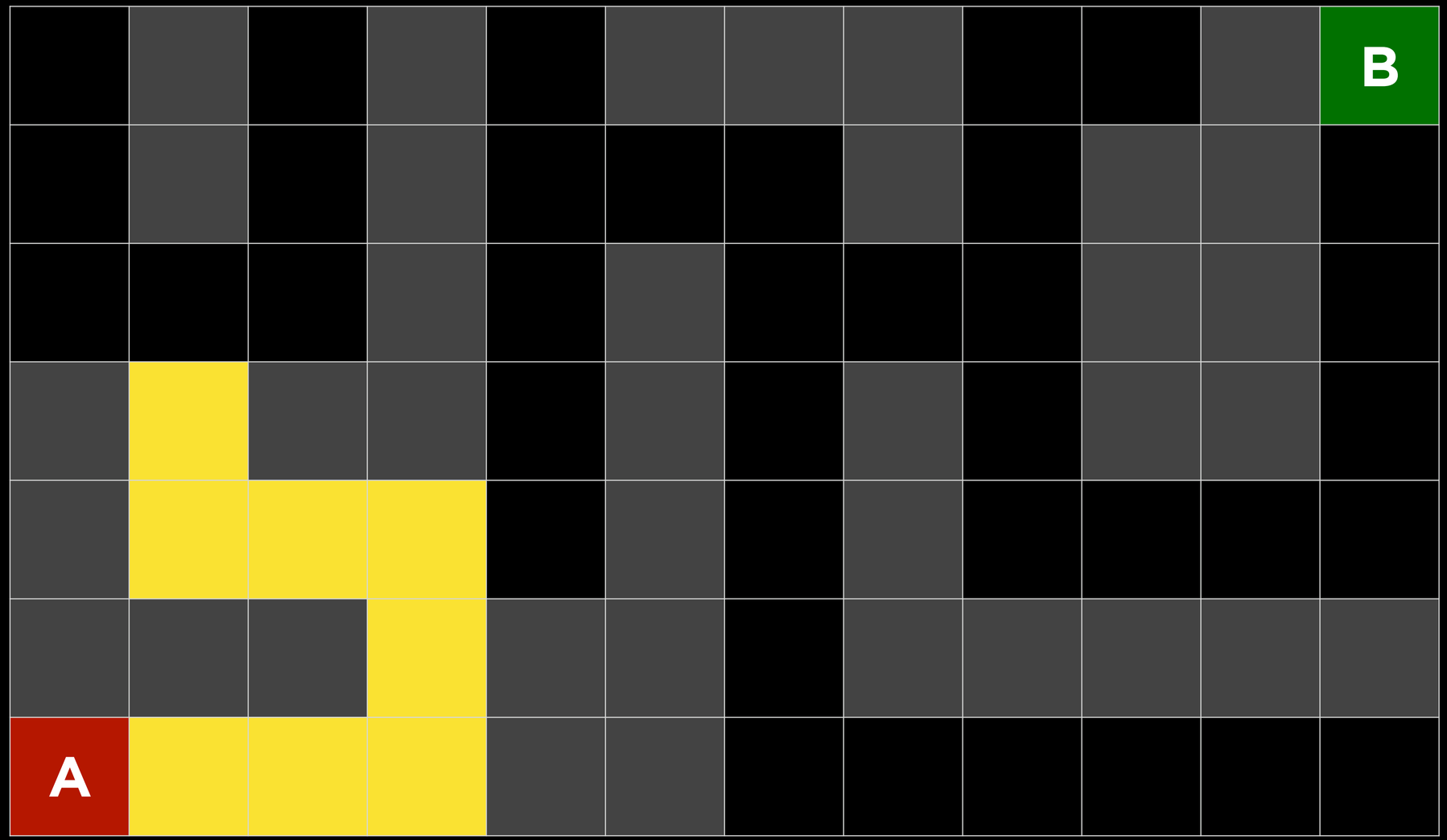
# Depth-First Search



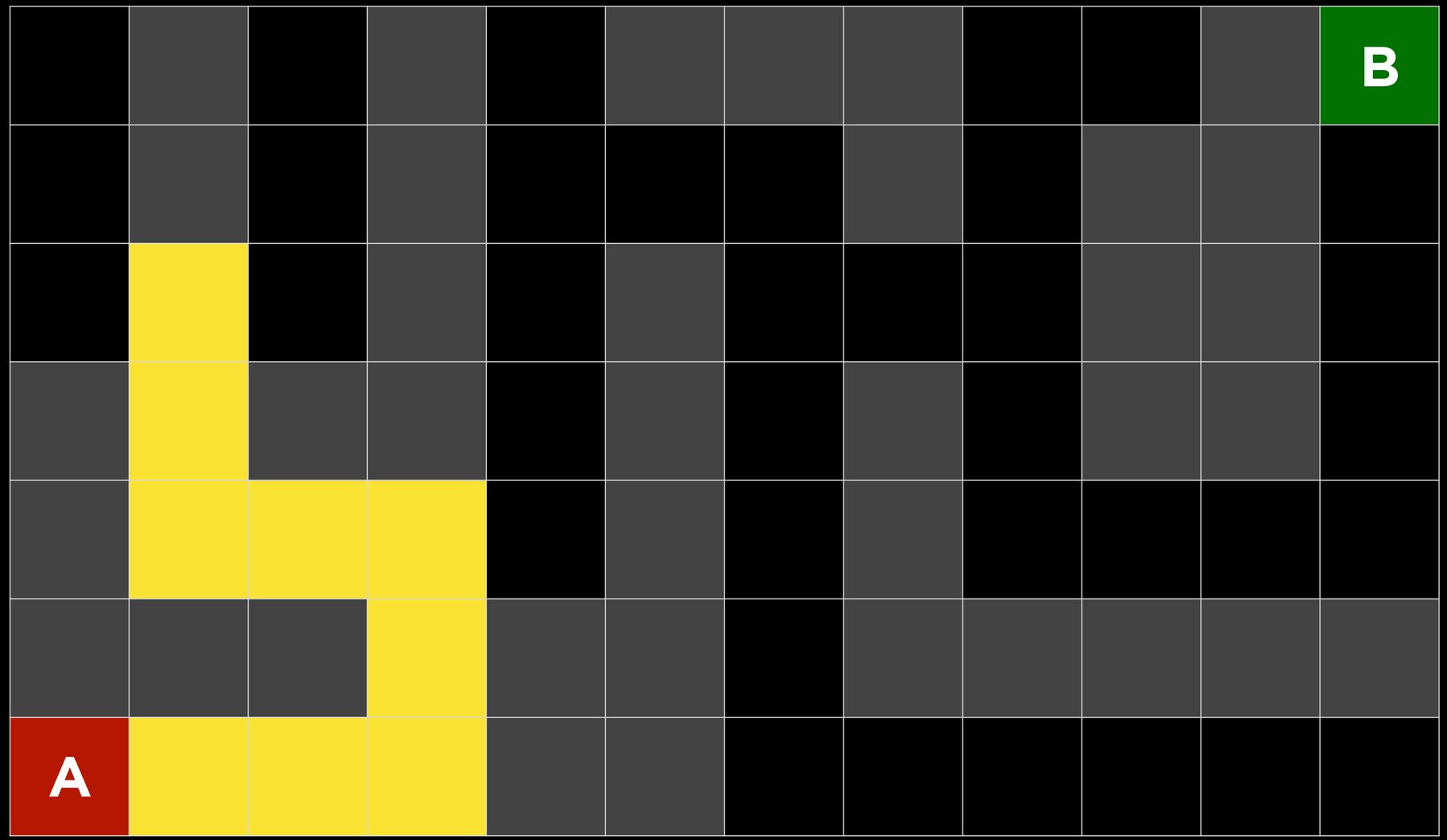
# Depth-First Search



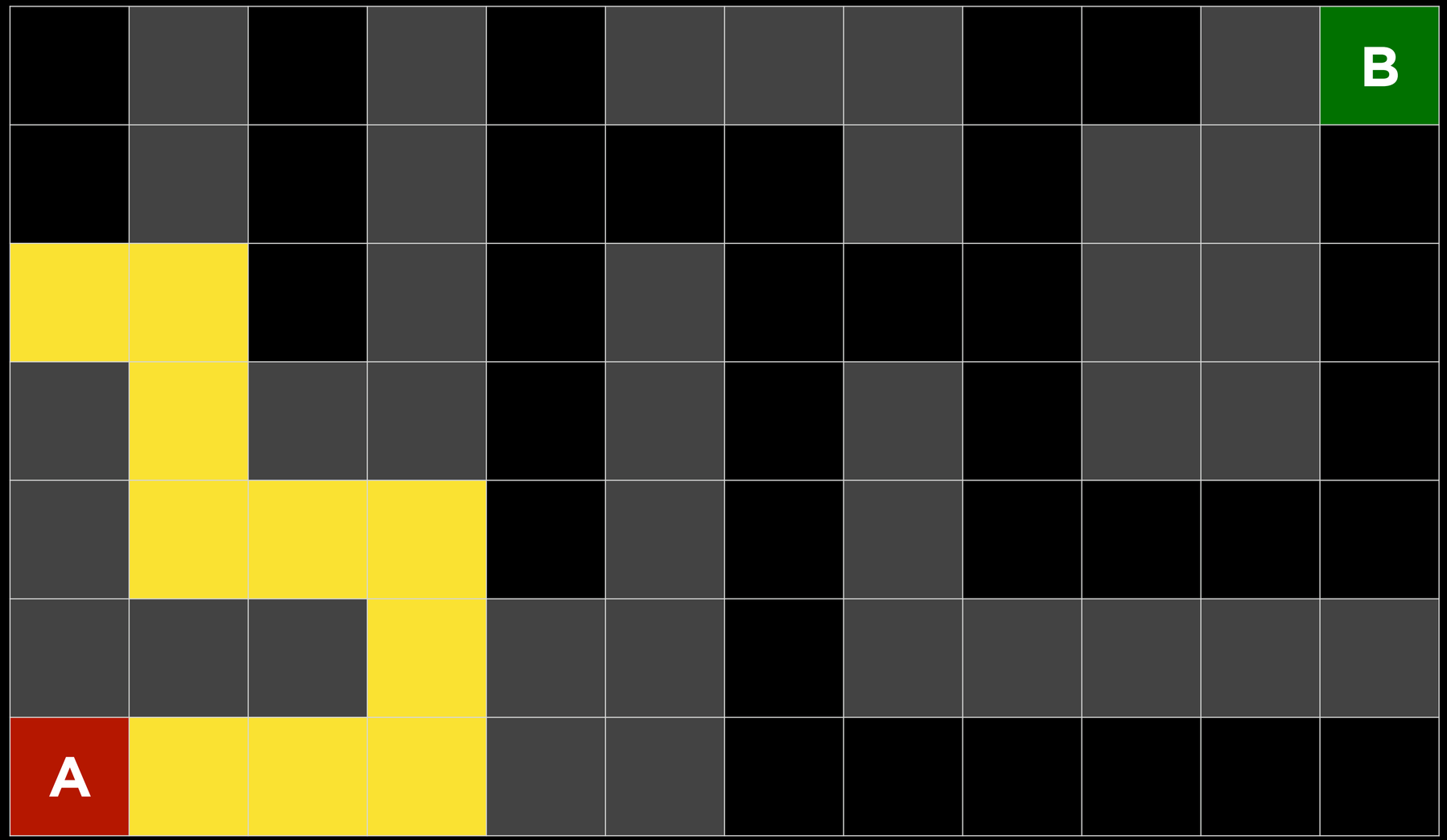
# Depth-First Search



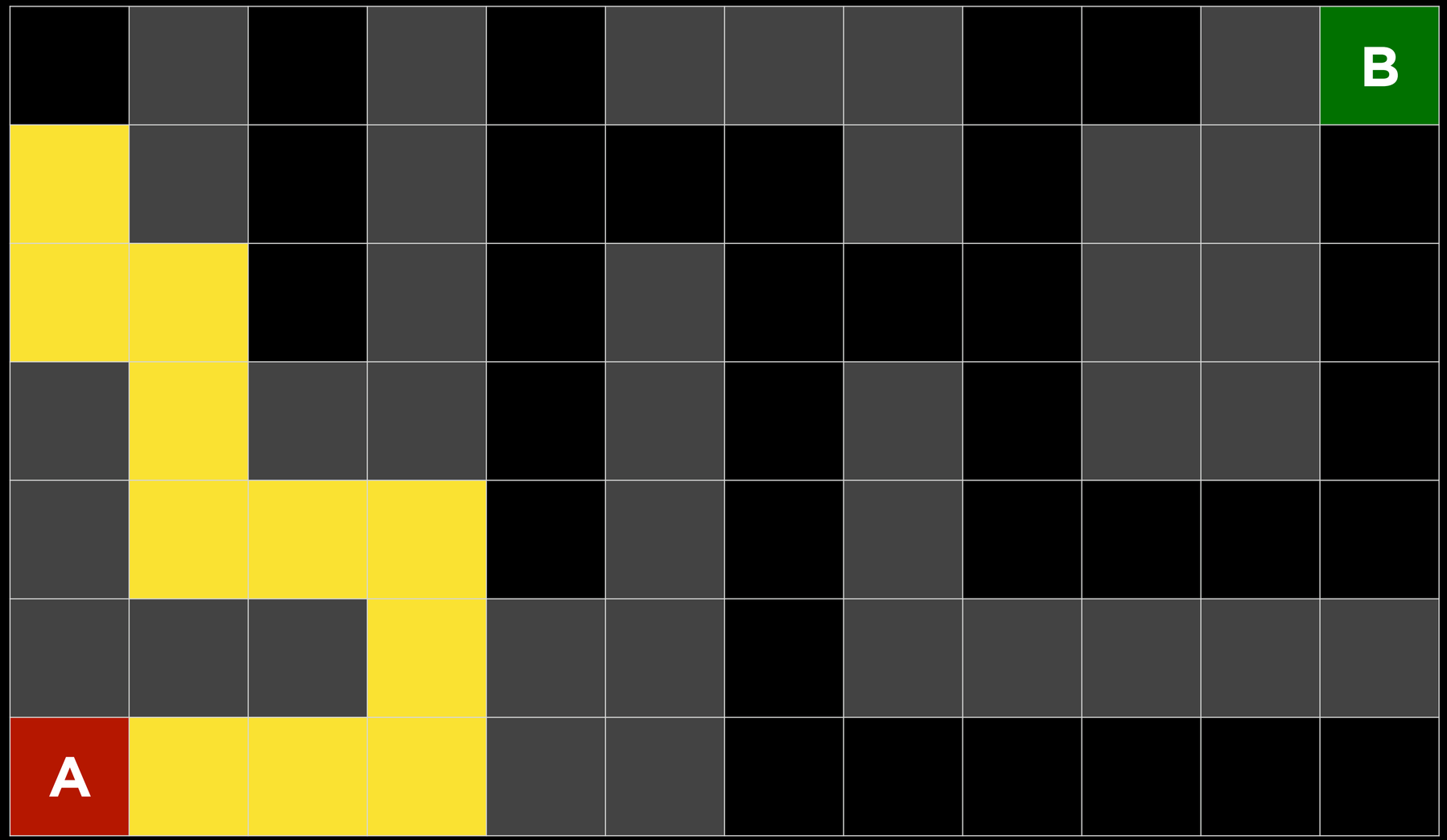
# Depth-First Search



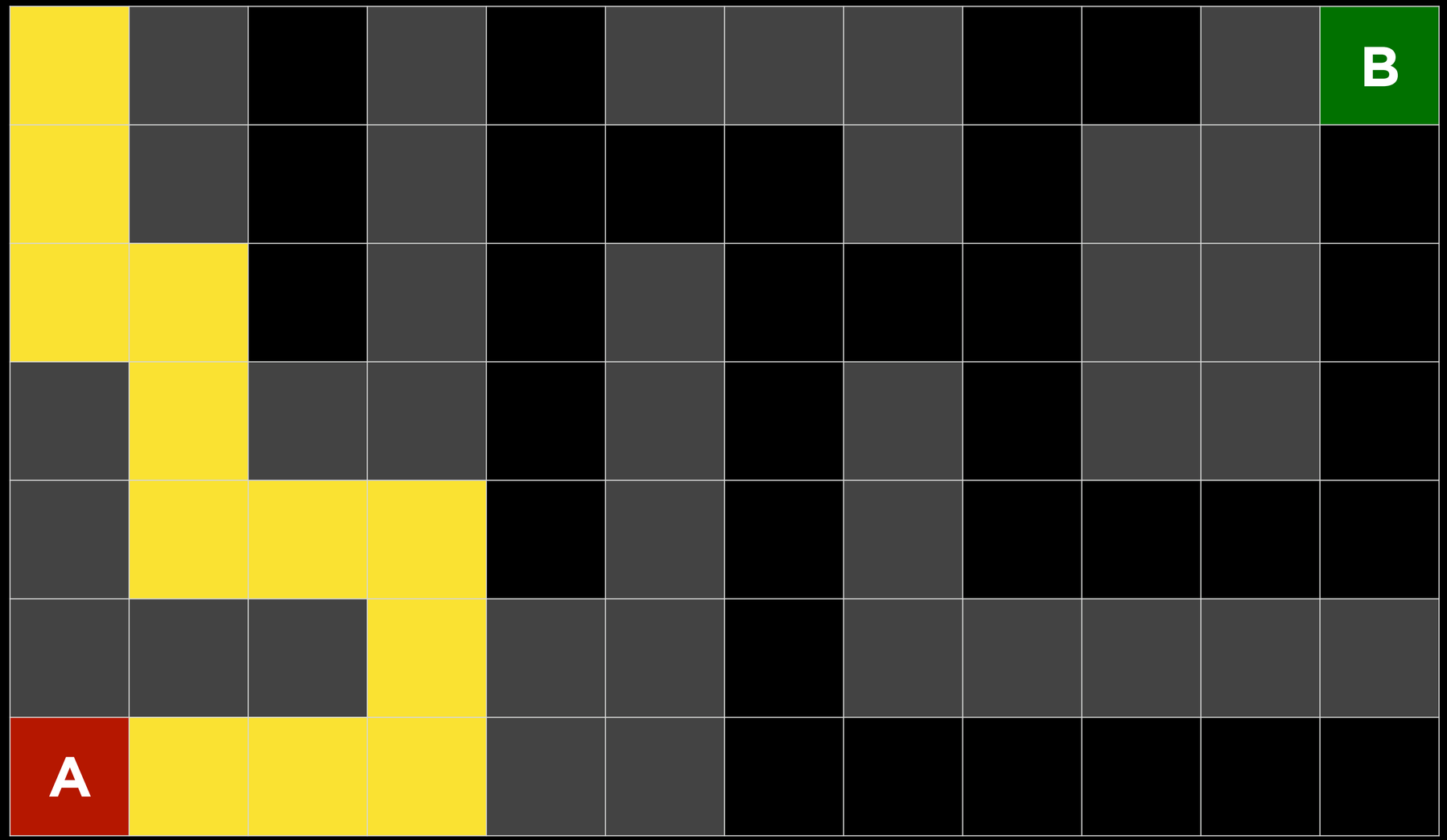
# Depth-First Search



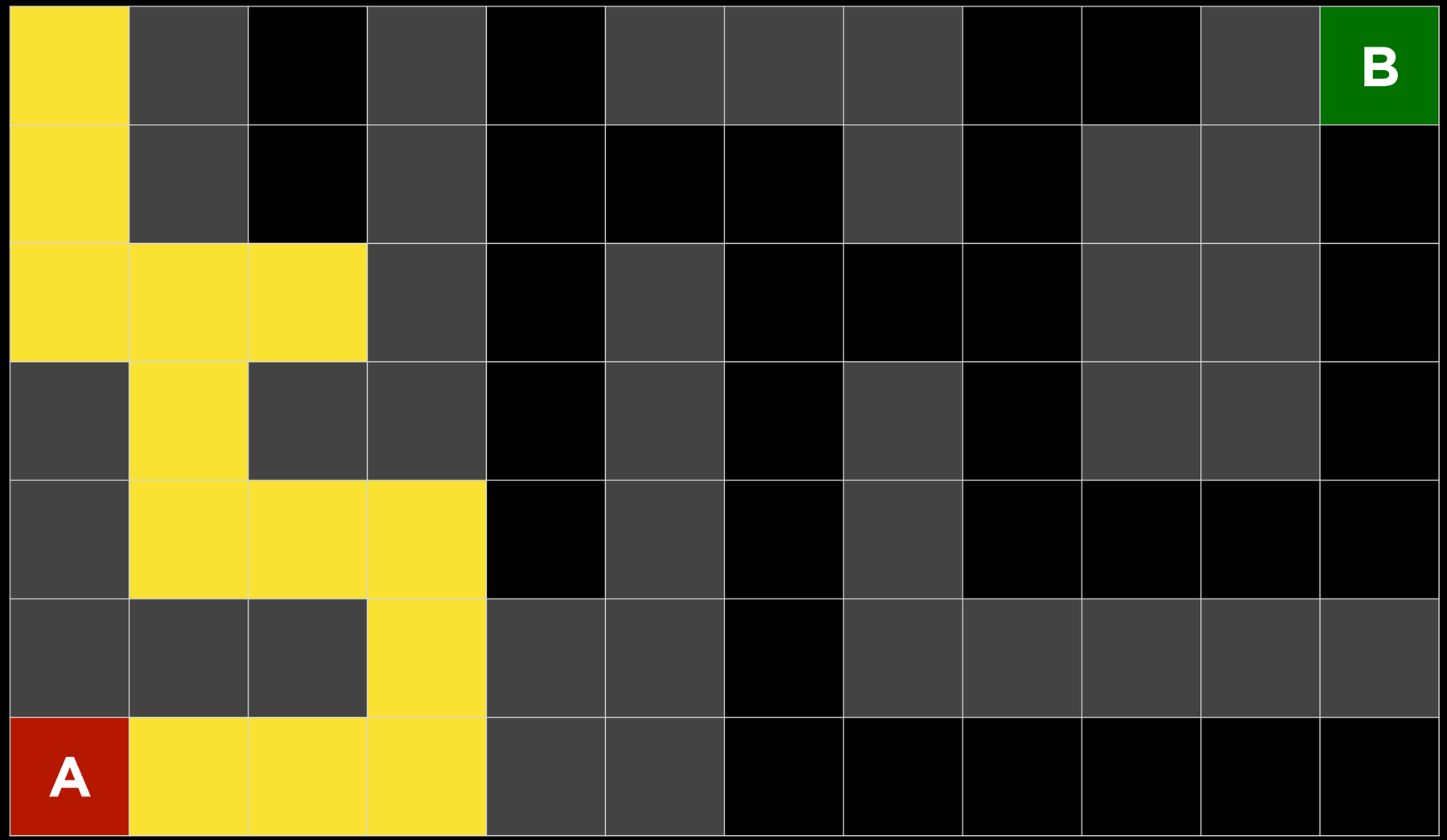
# Depth-First Search



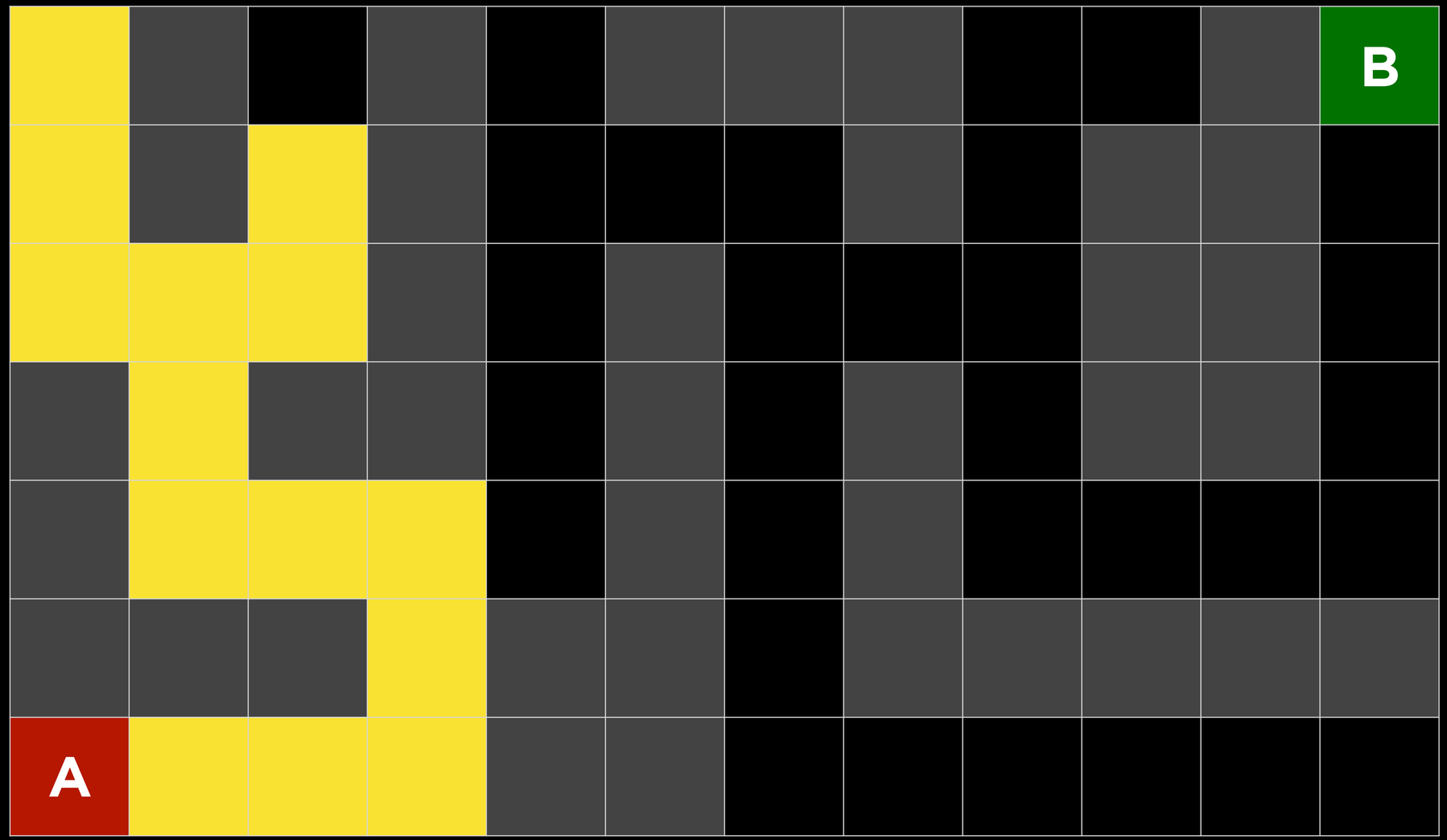
# Depth-First Search



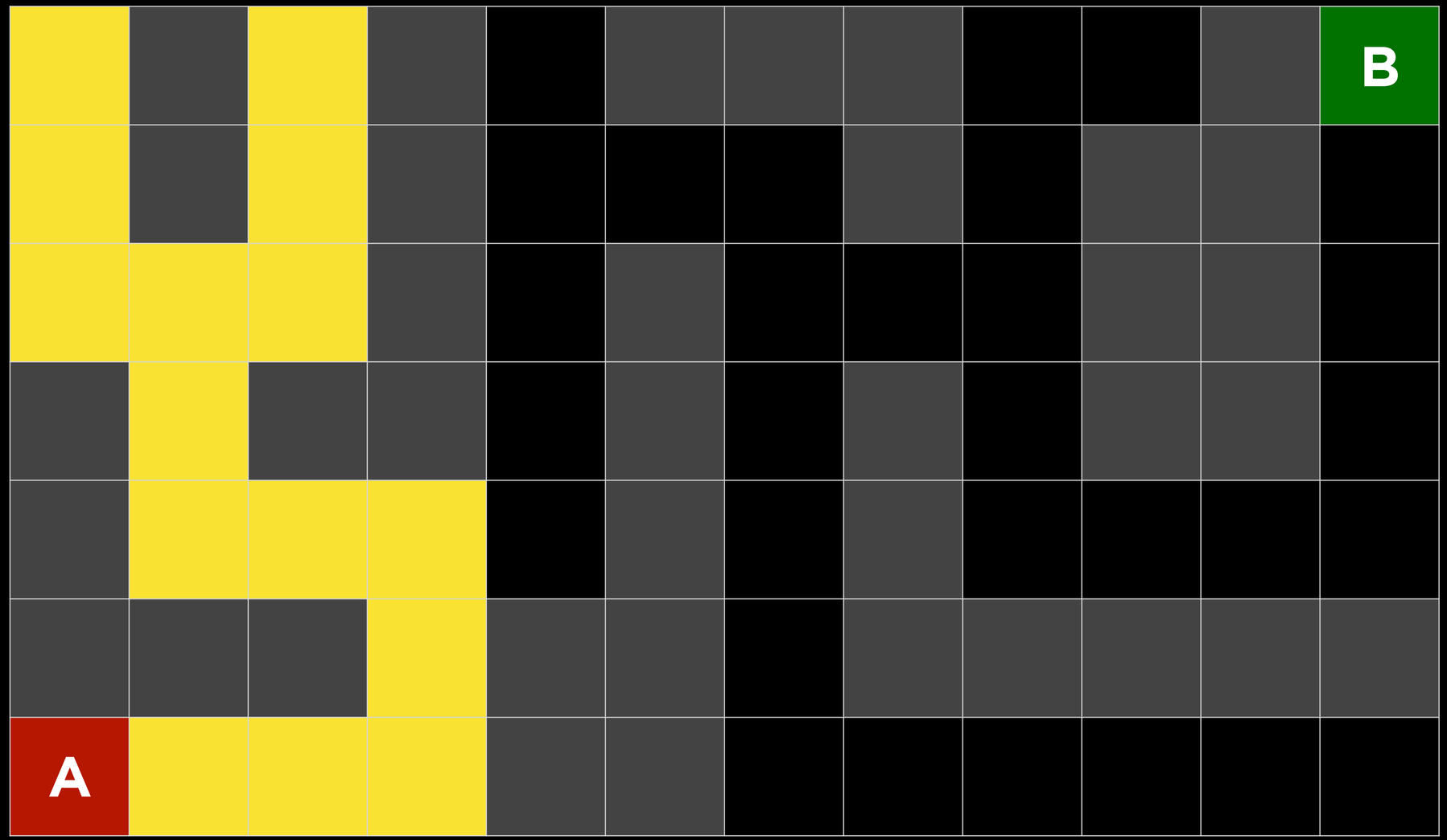
# Depth-First Search



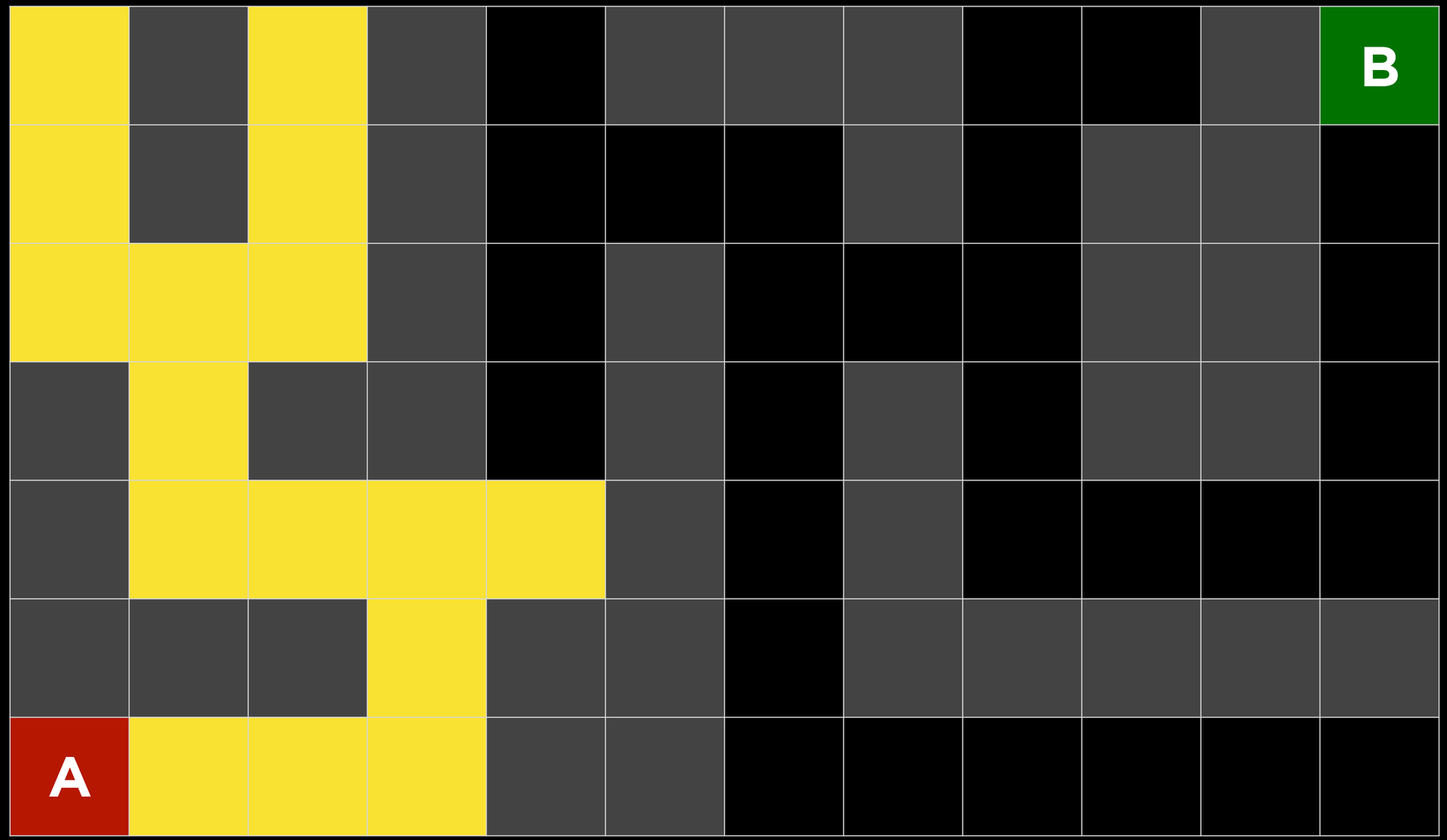
# Depth-First Search



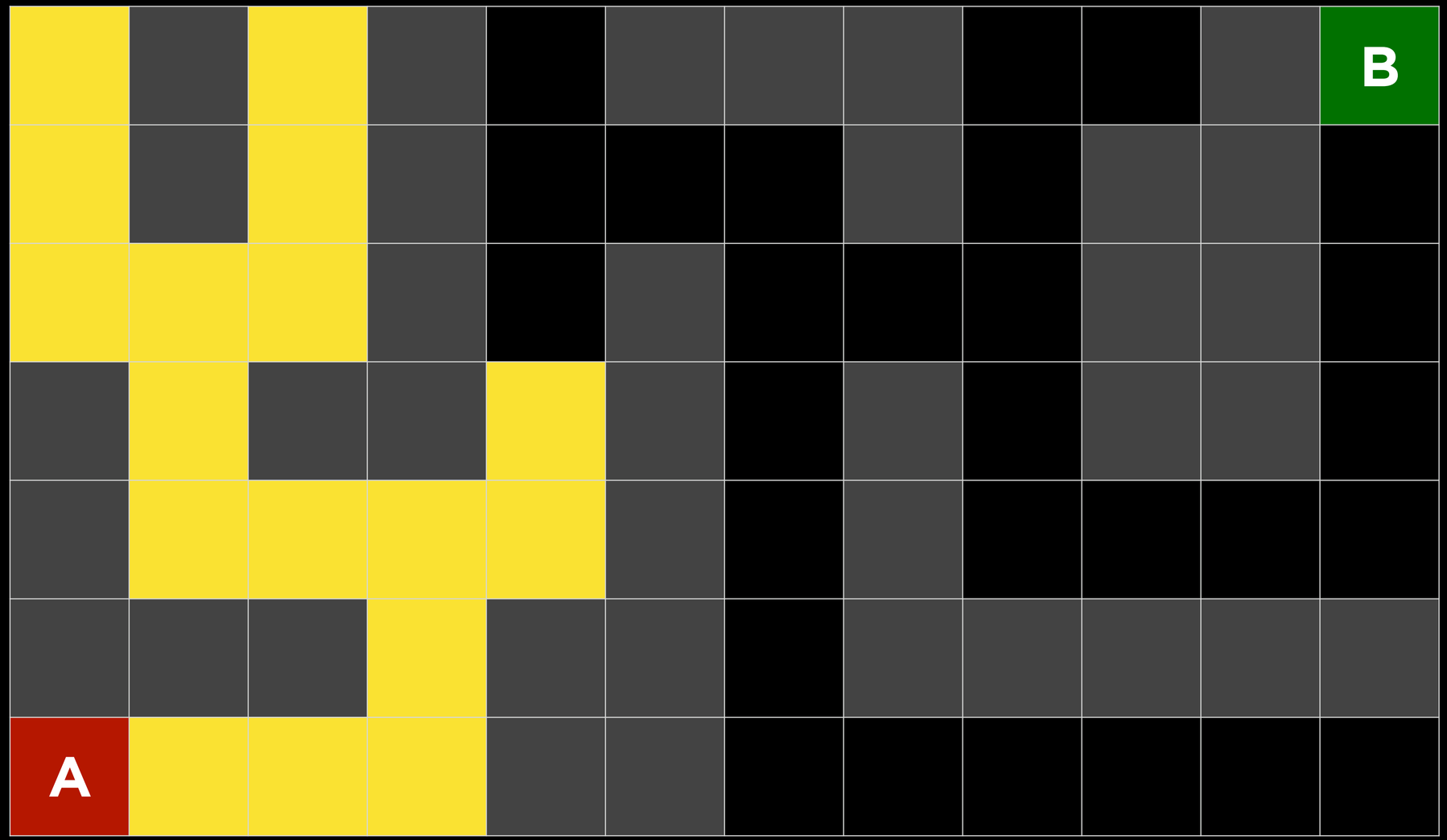
# Depth-First Search



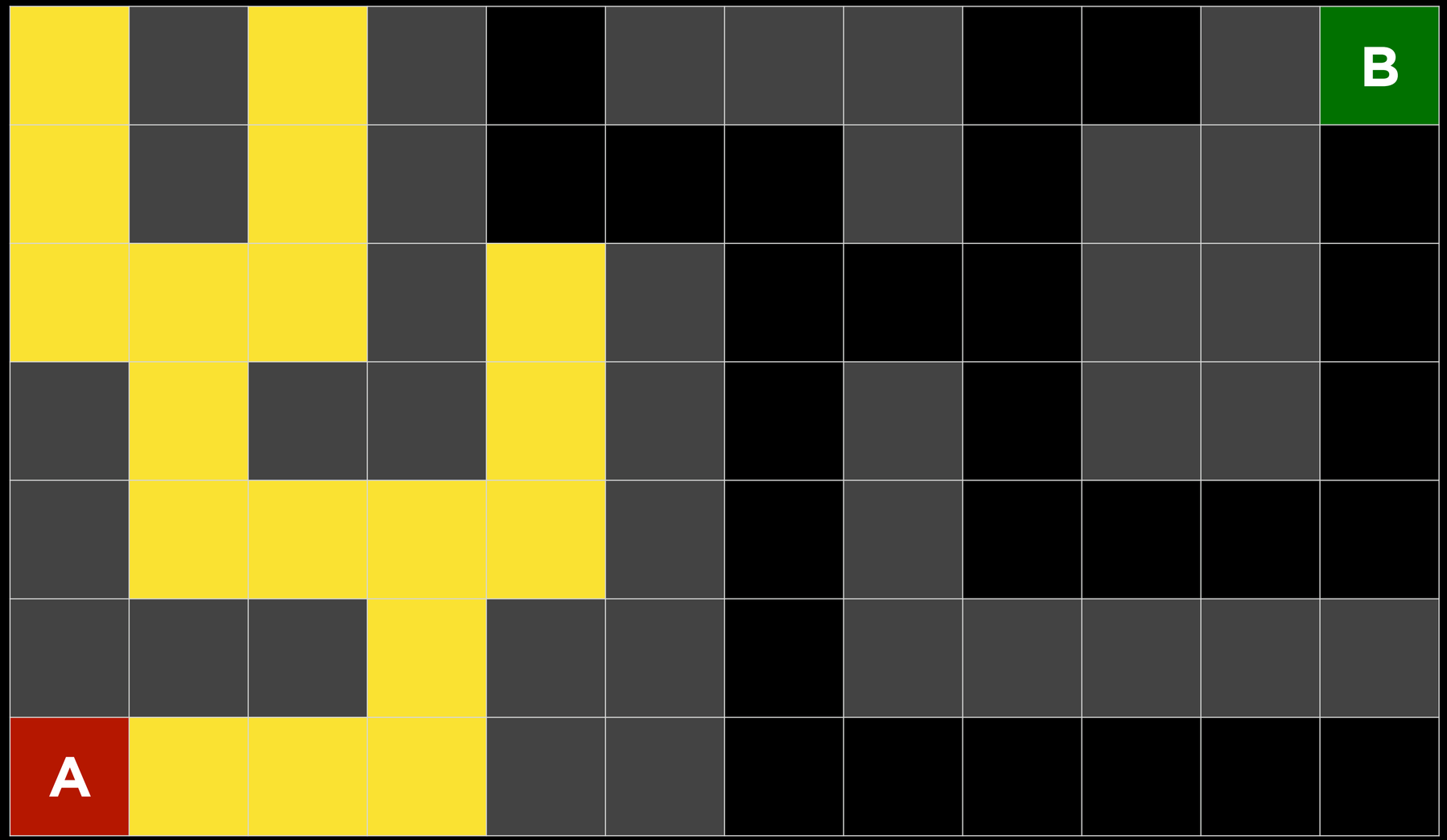
# Depth-First Search



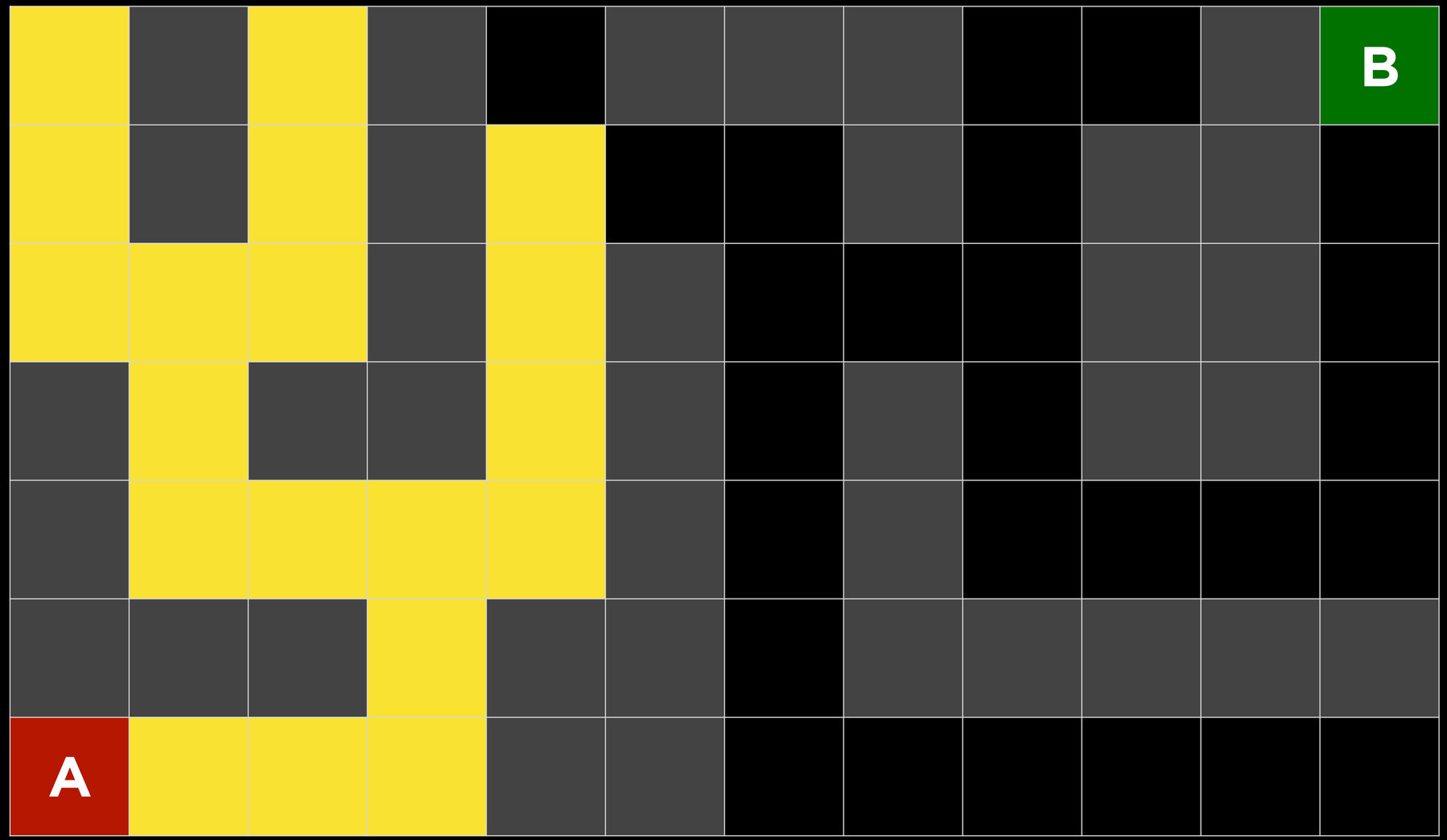
# Depth-First Search



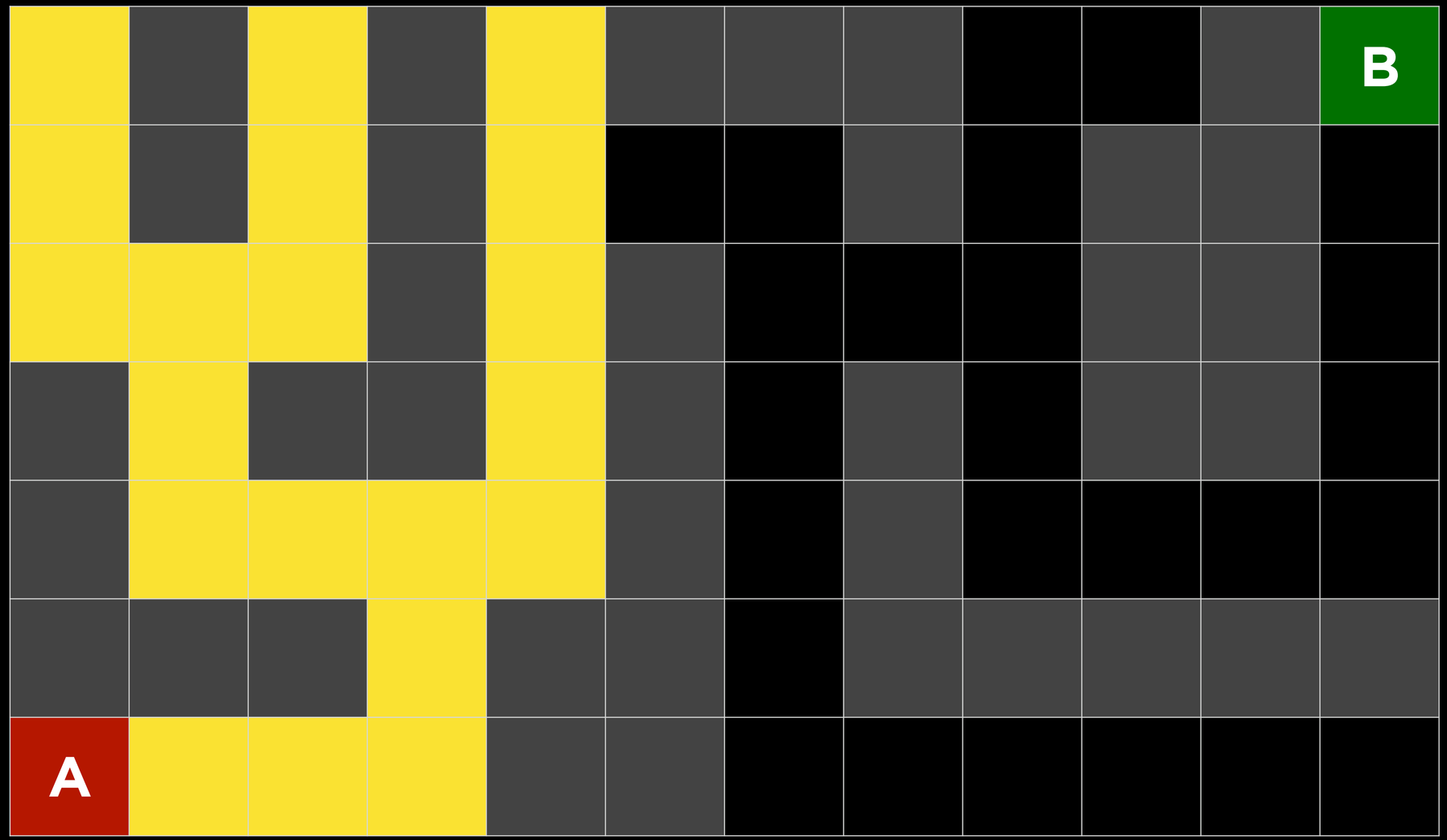
# Depth-First Search



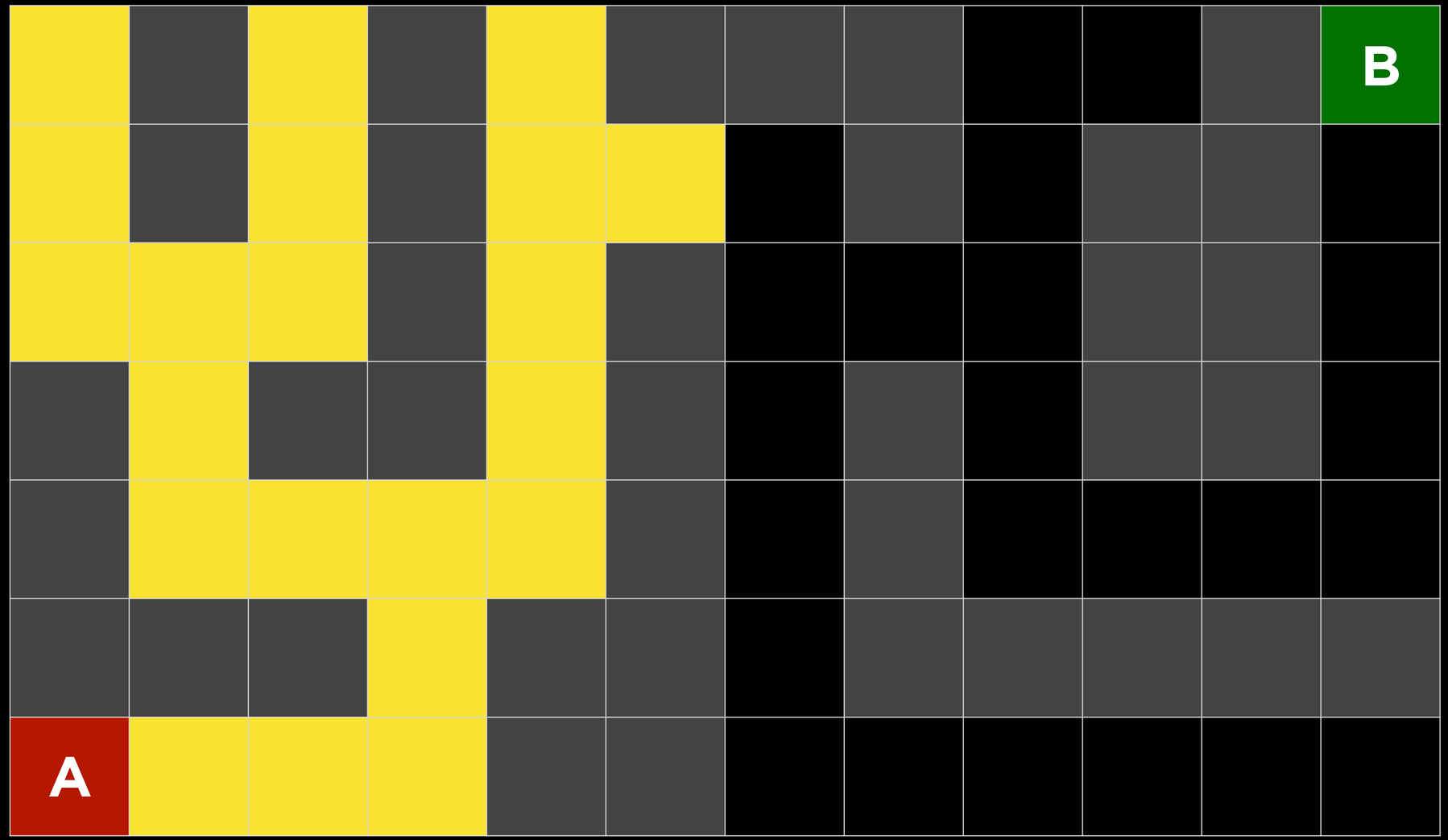
# Depth-First Search



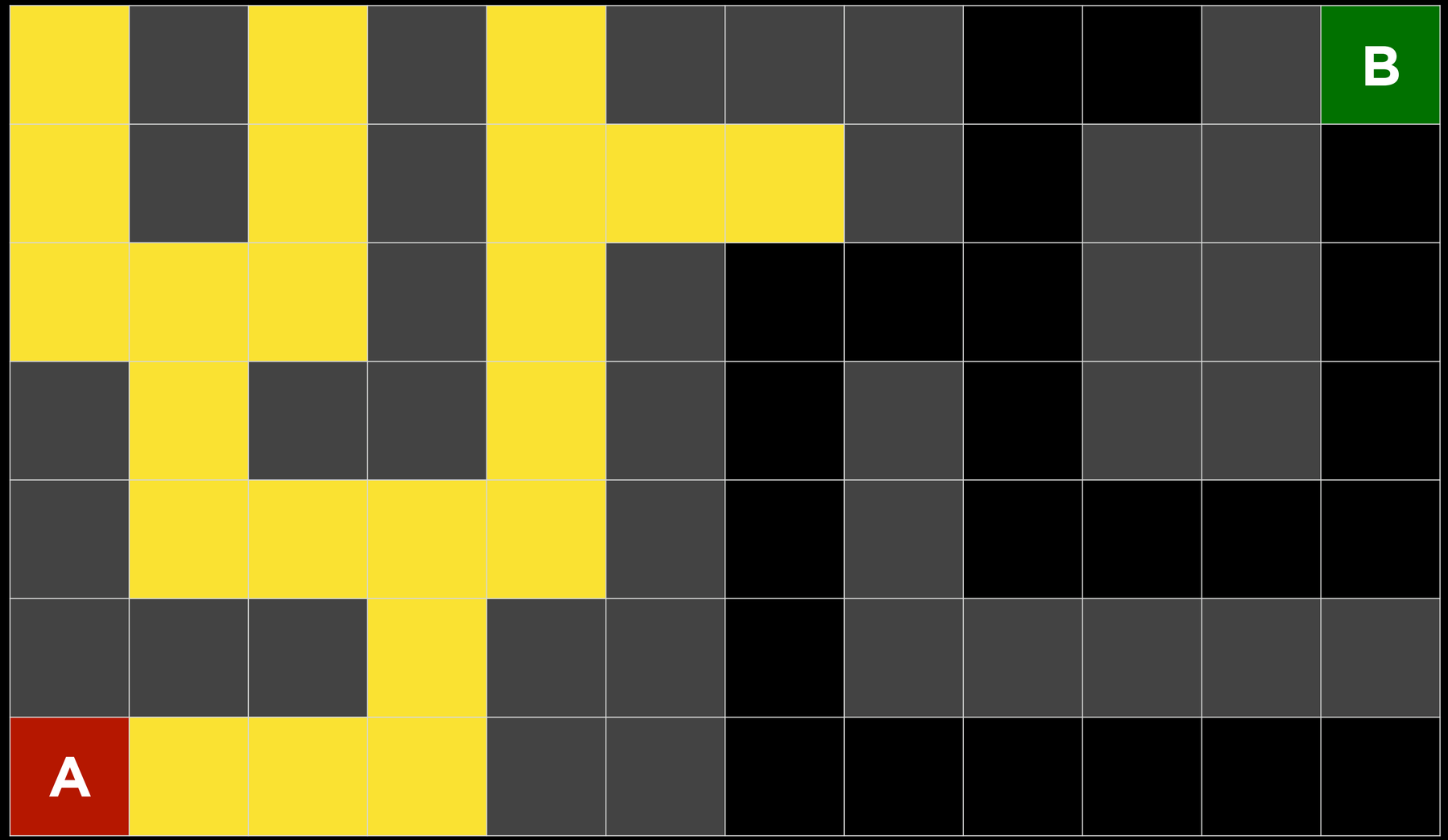
# Depth-First Search



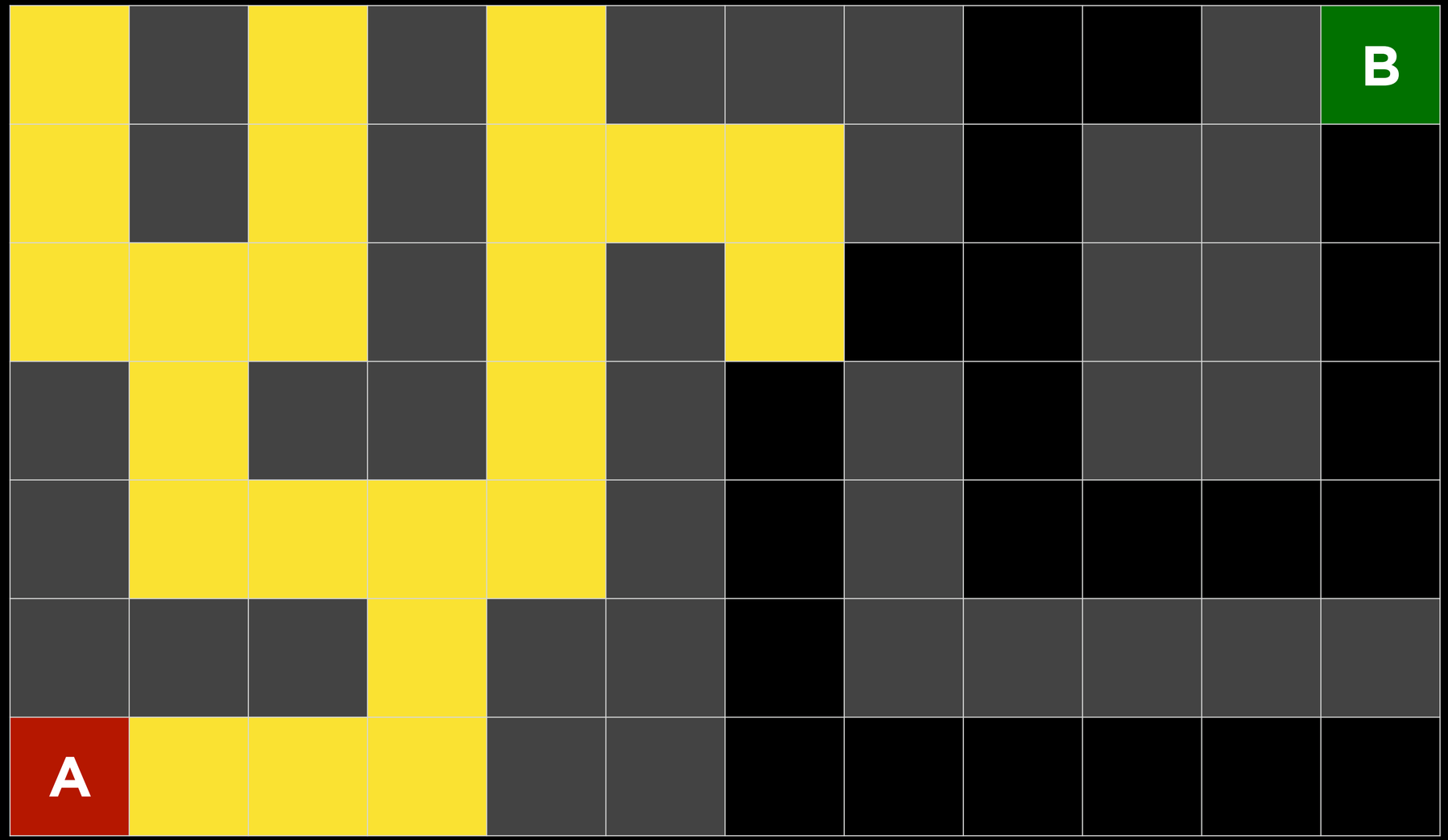
# Depth-First Search



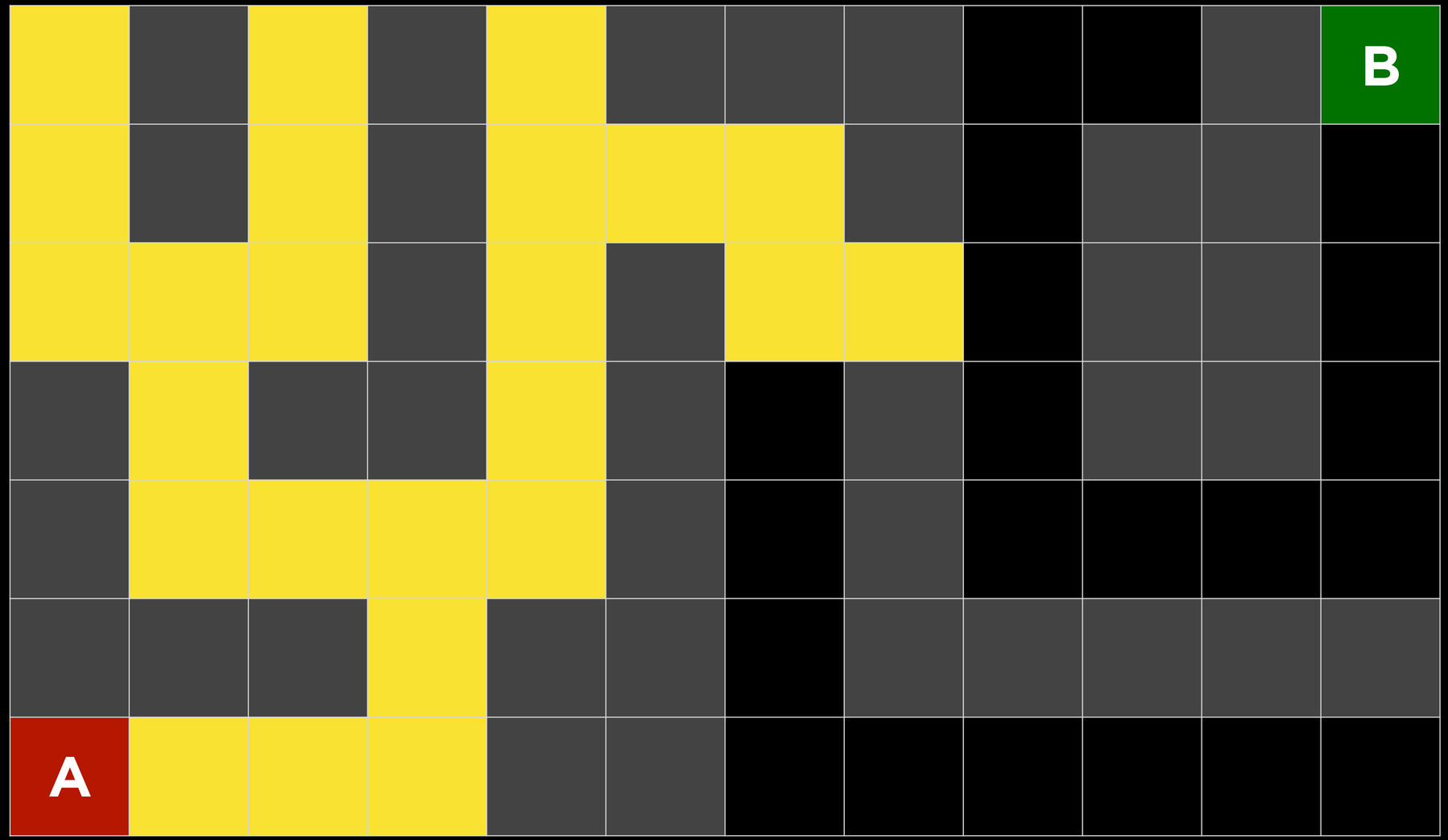
# Depth-First Search



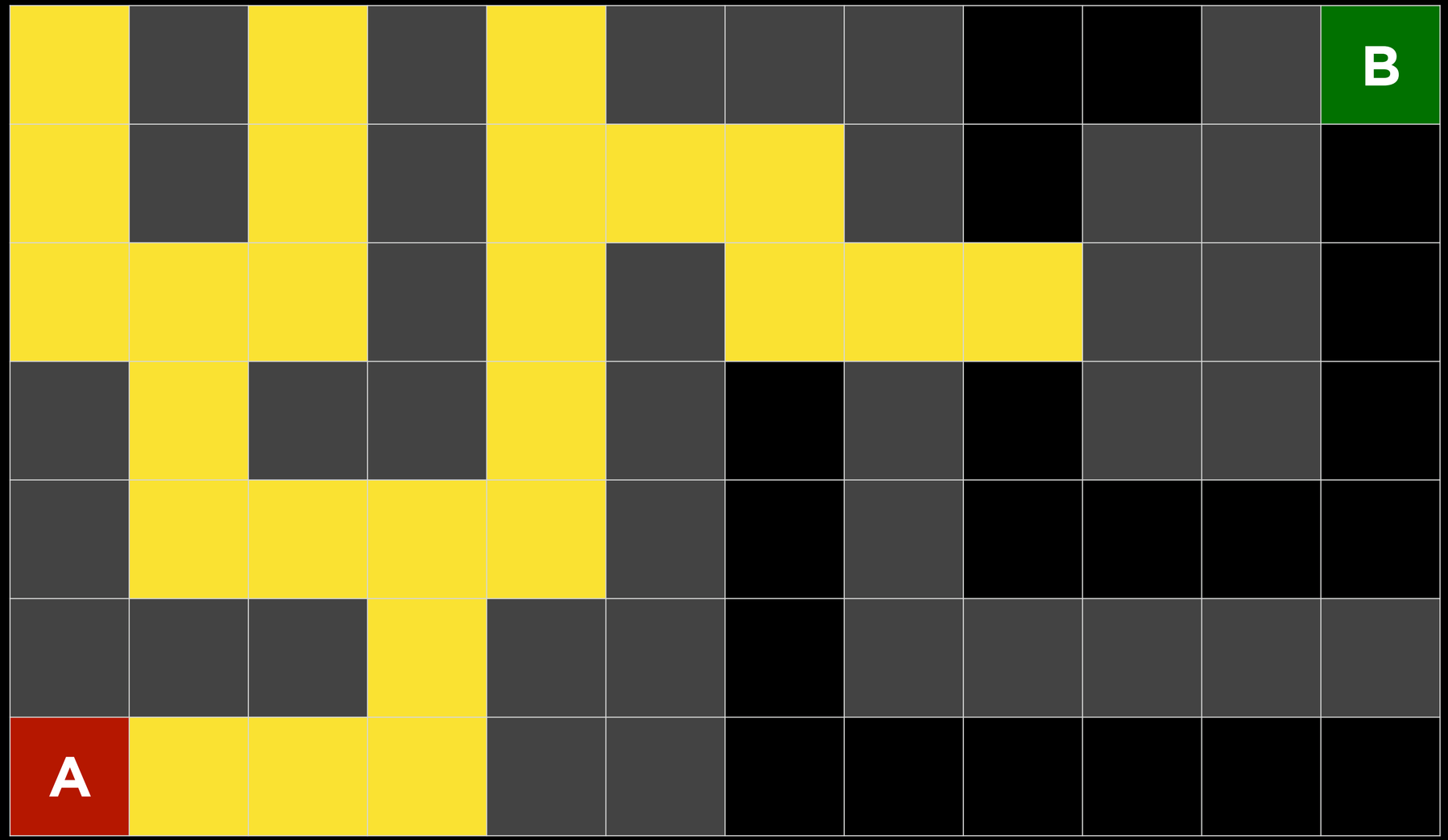
# Depth-First Search



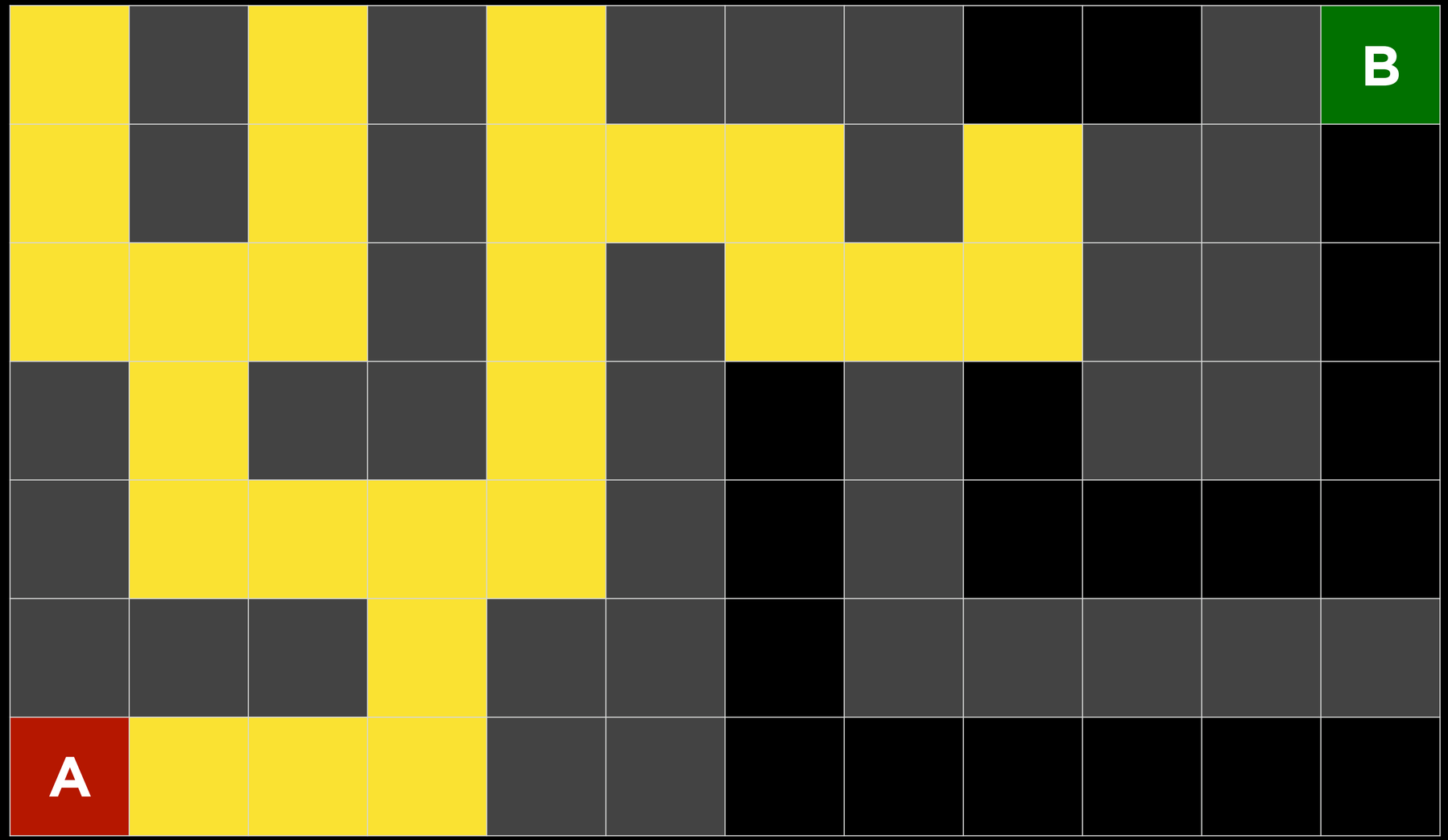
# Depth-First Search



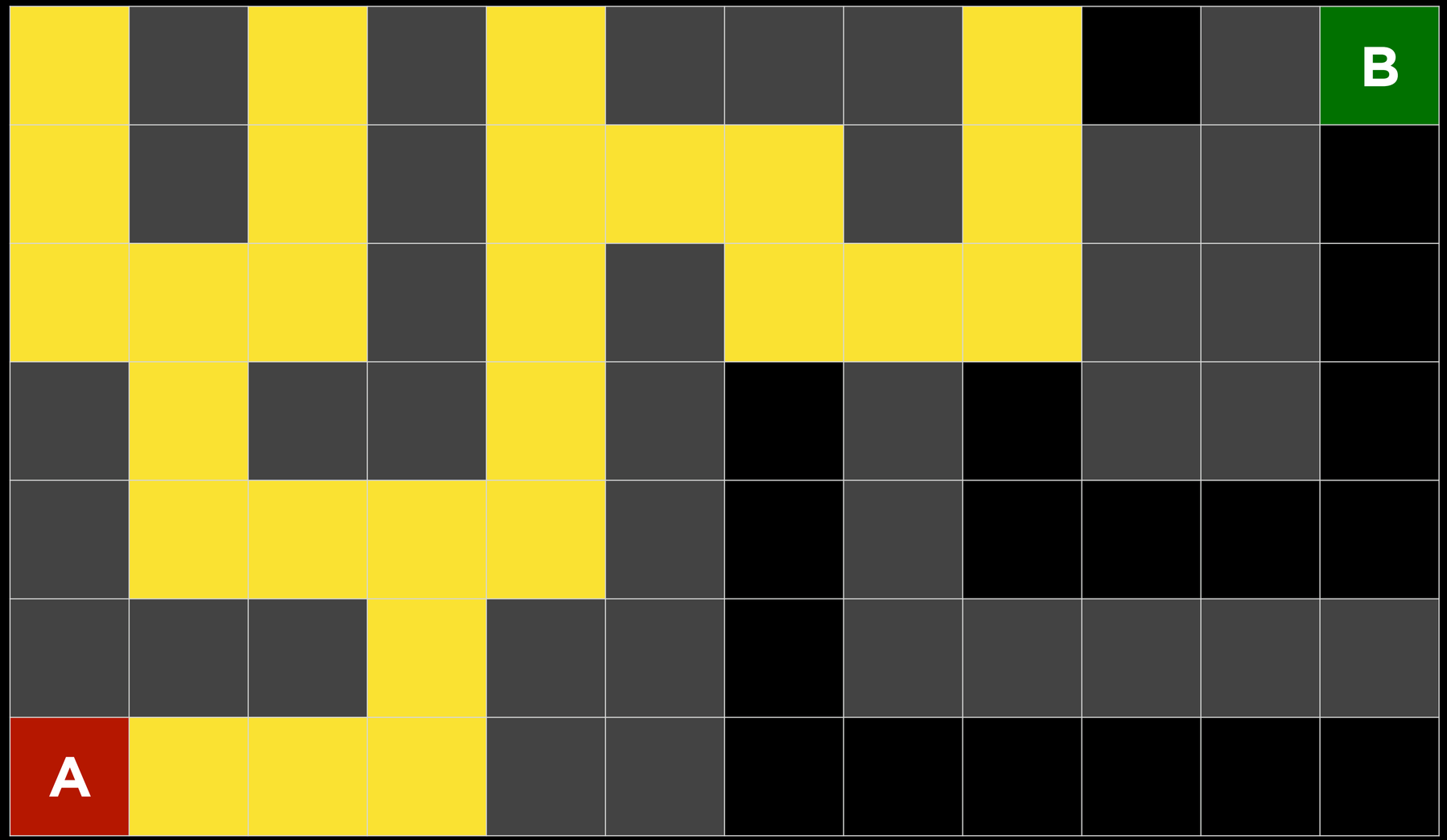
# Depth-First Search



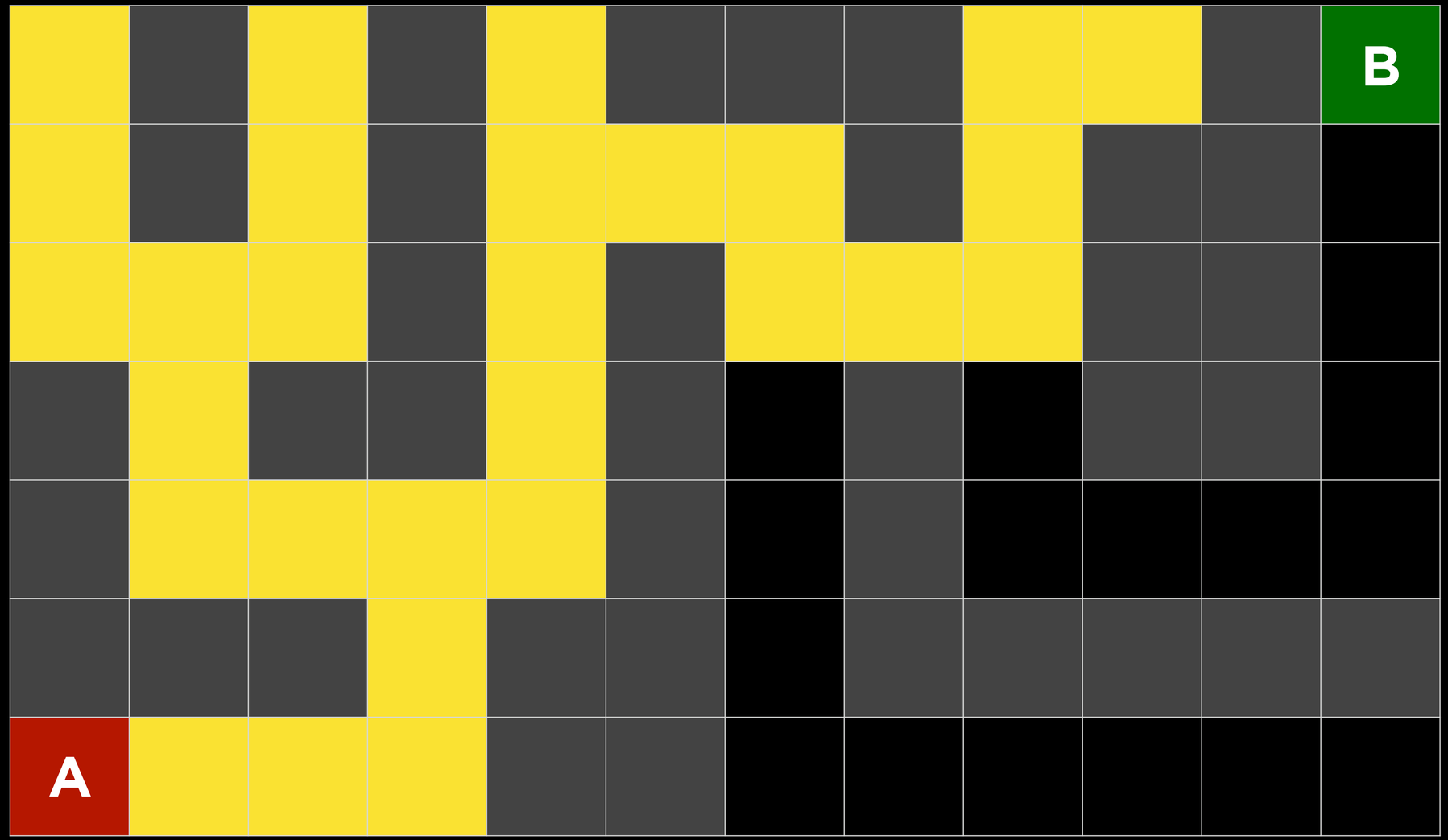
# Depth-First Search



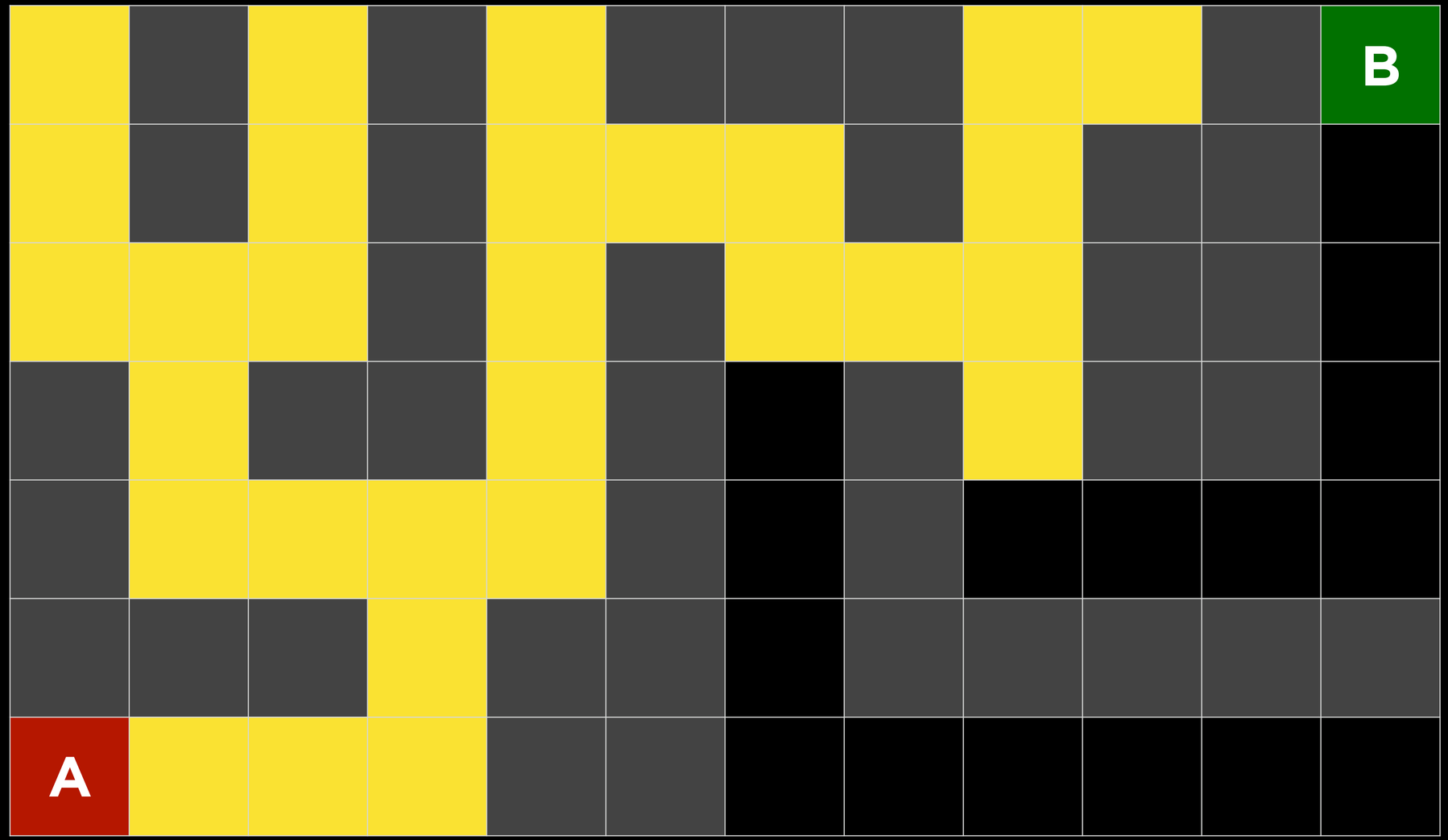
# Depth-First Search



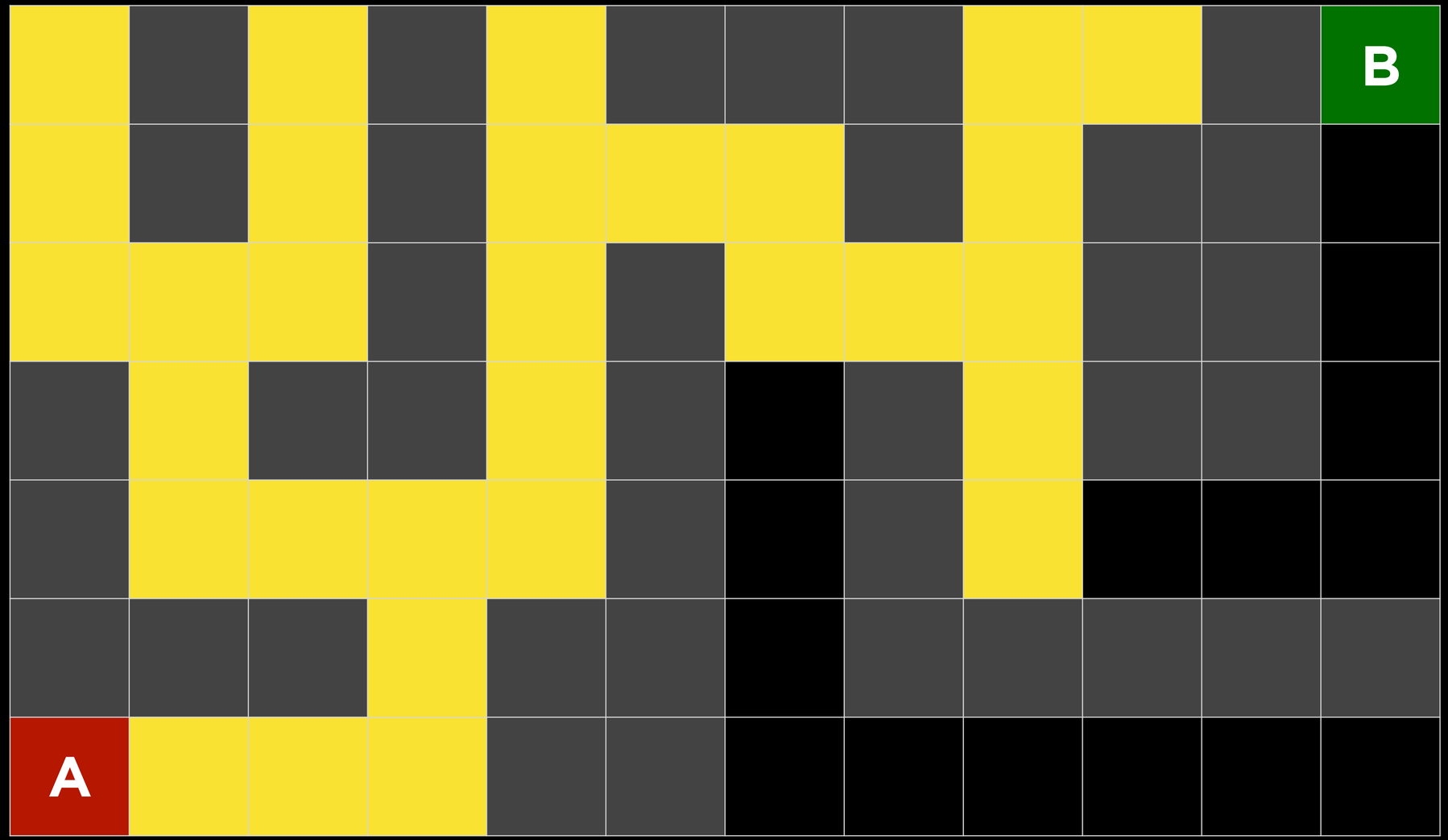
# Depth-First Search



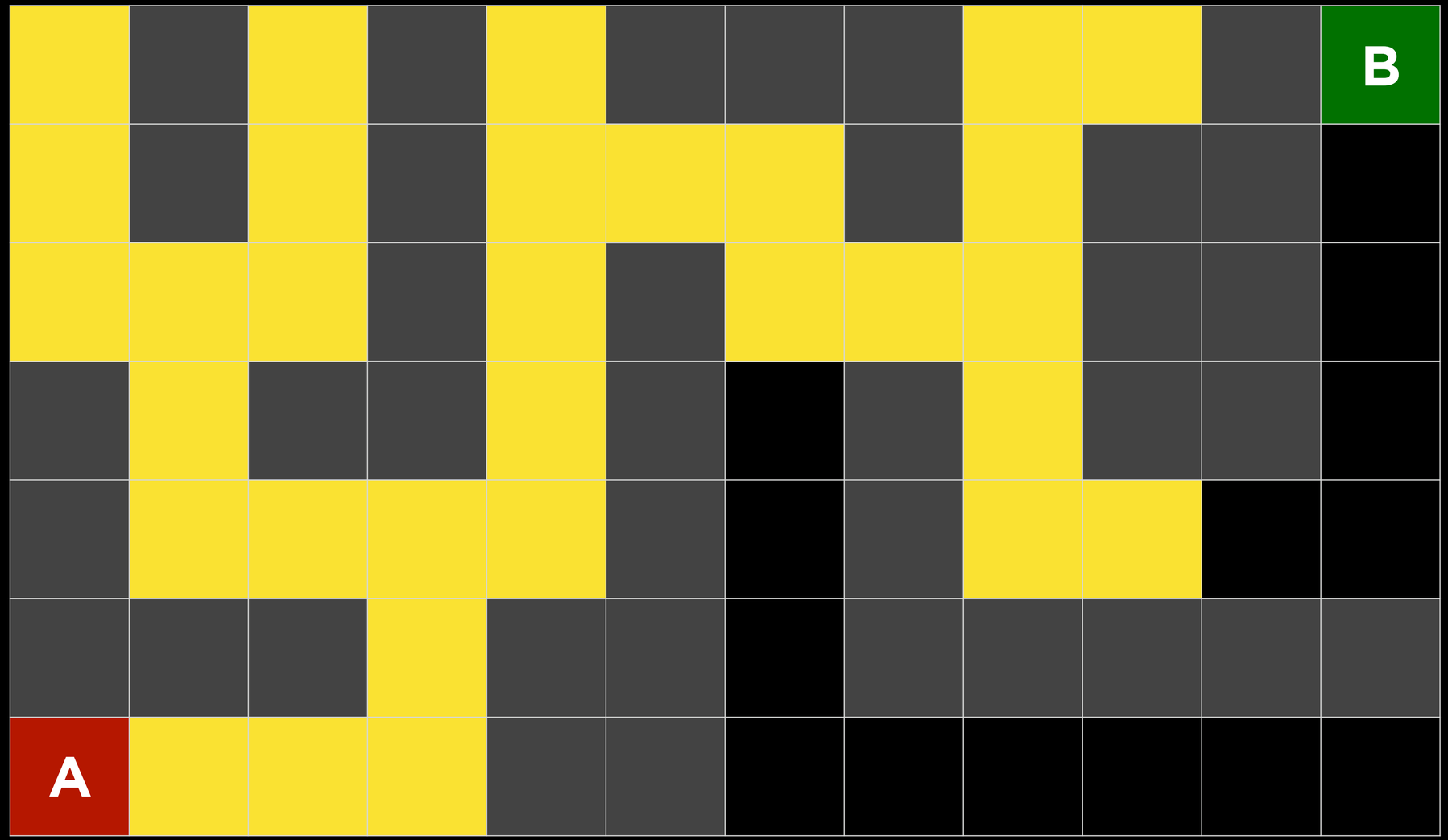
# Depth-First Search



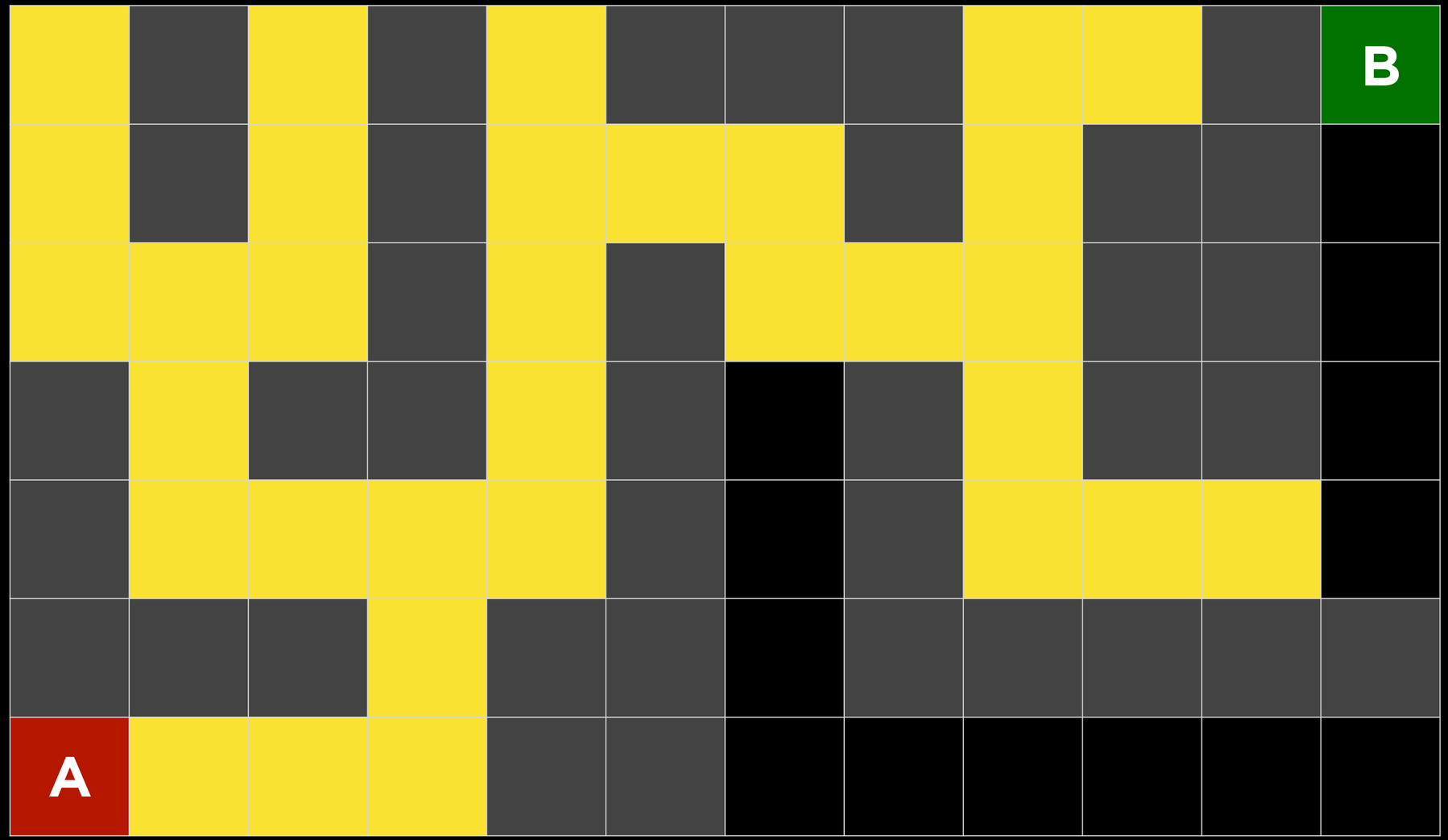
# Depth-First Search



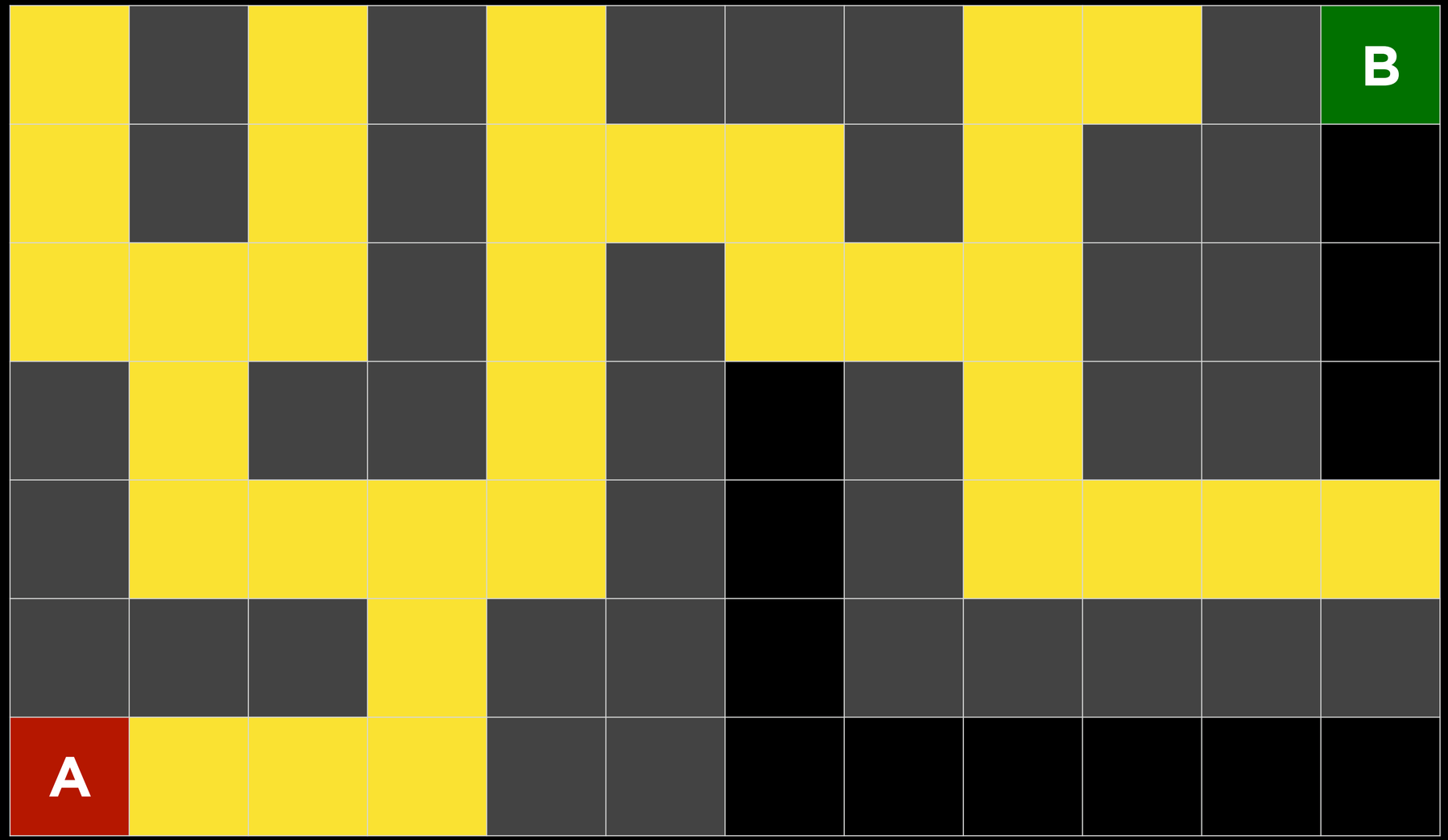
# Depth-First Search



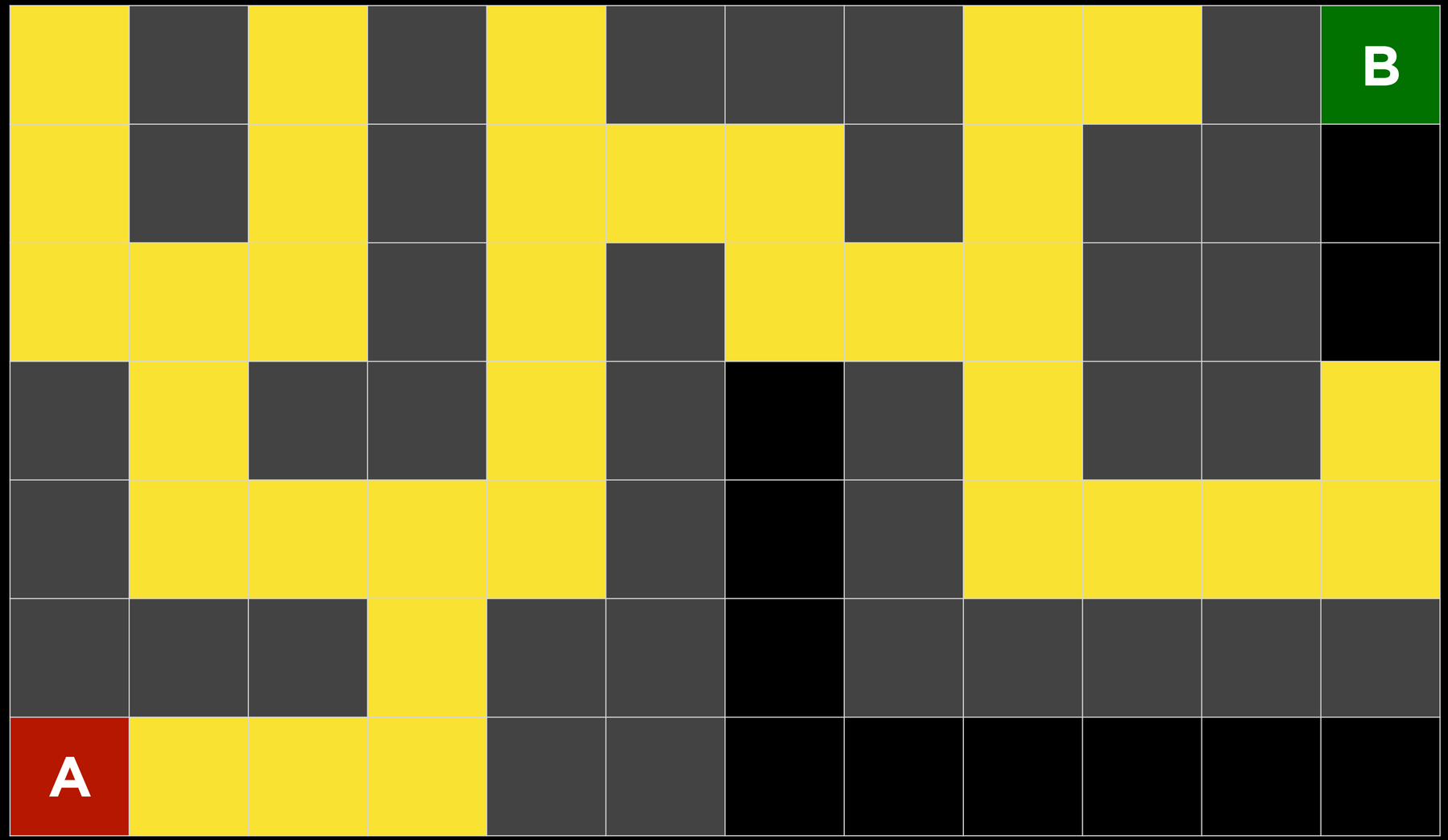
# Depth-First Search



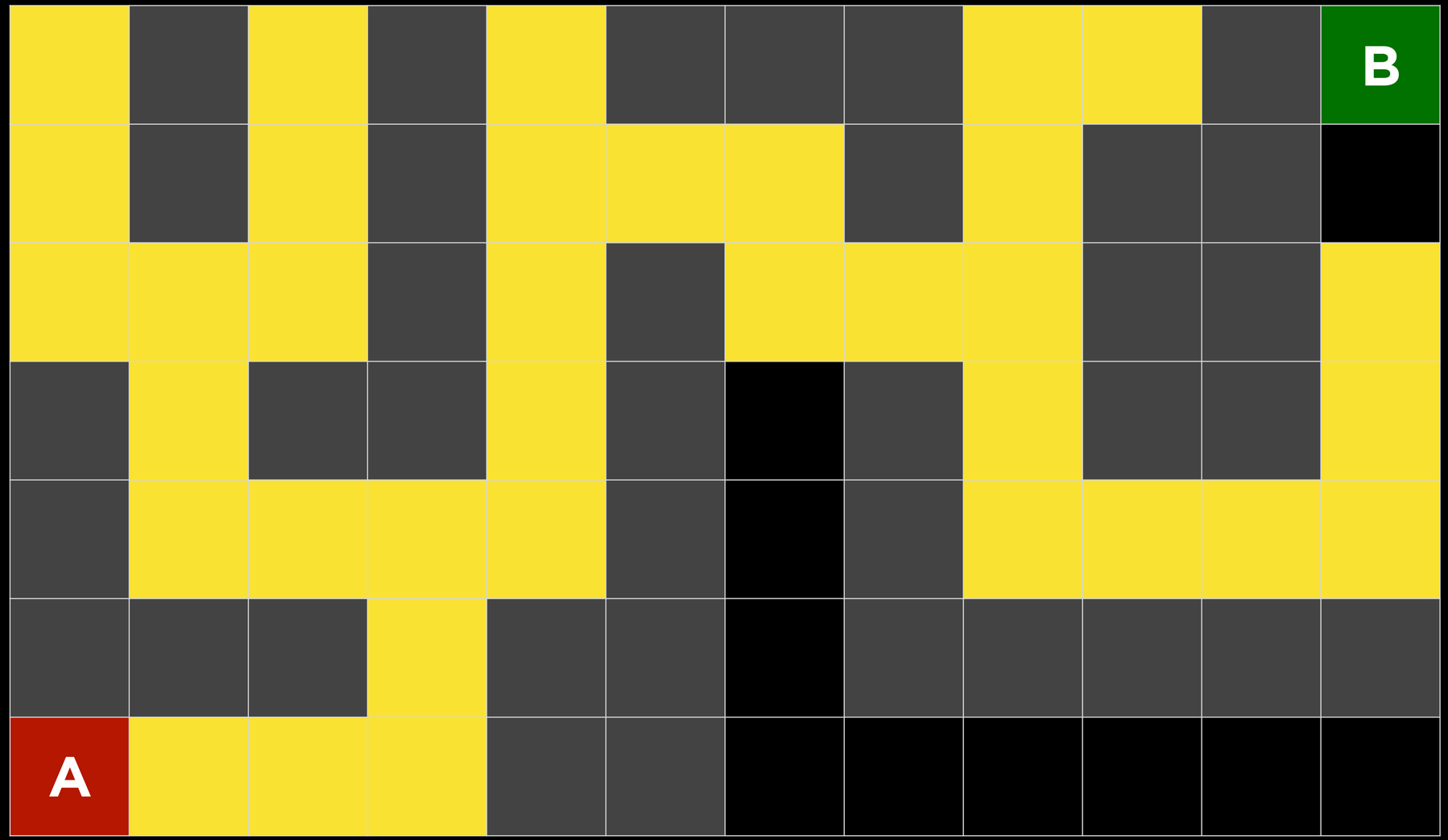
# Depth-First Search



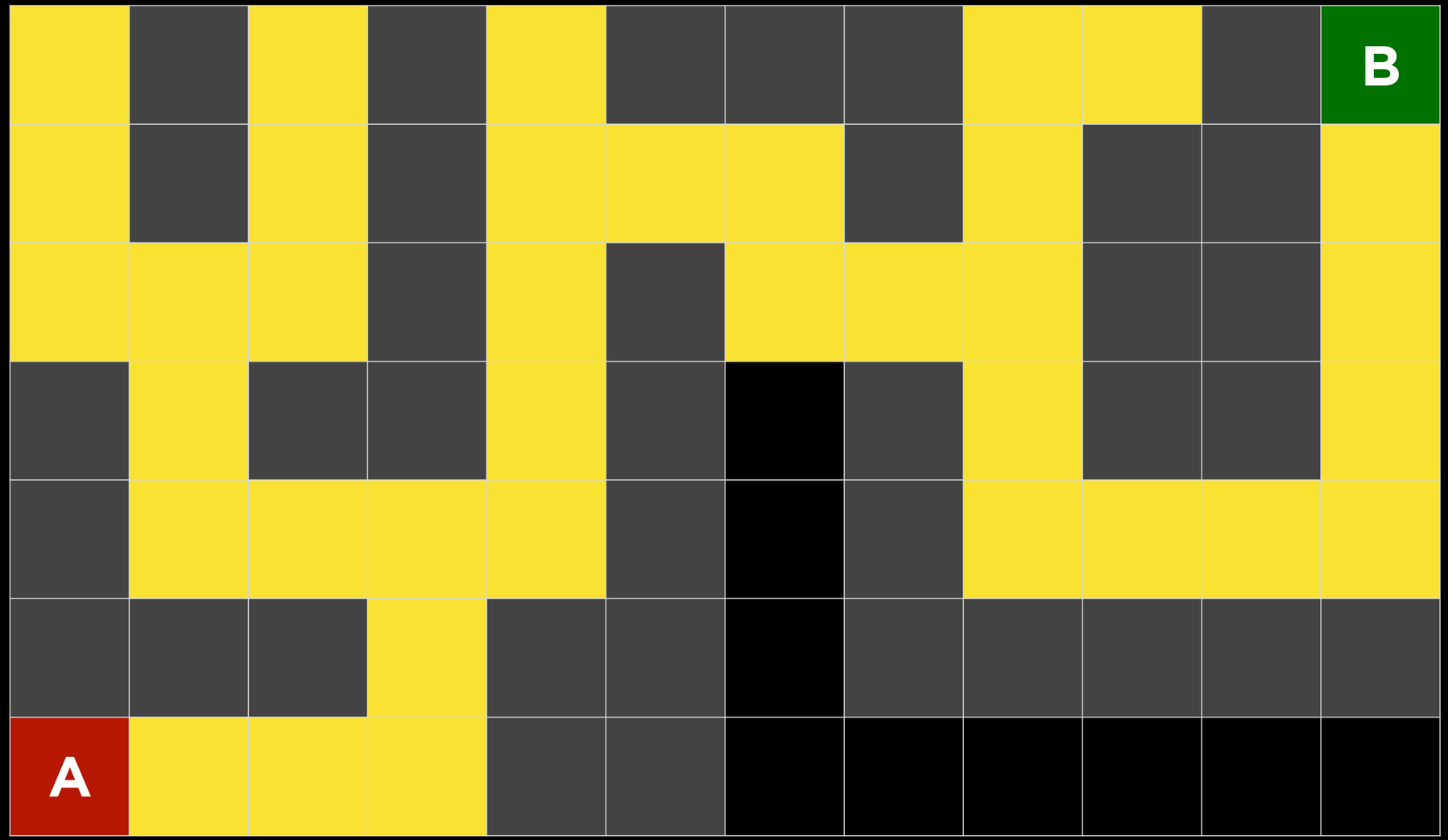
# Depth-First Search



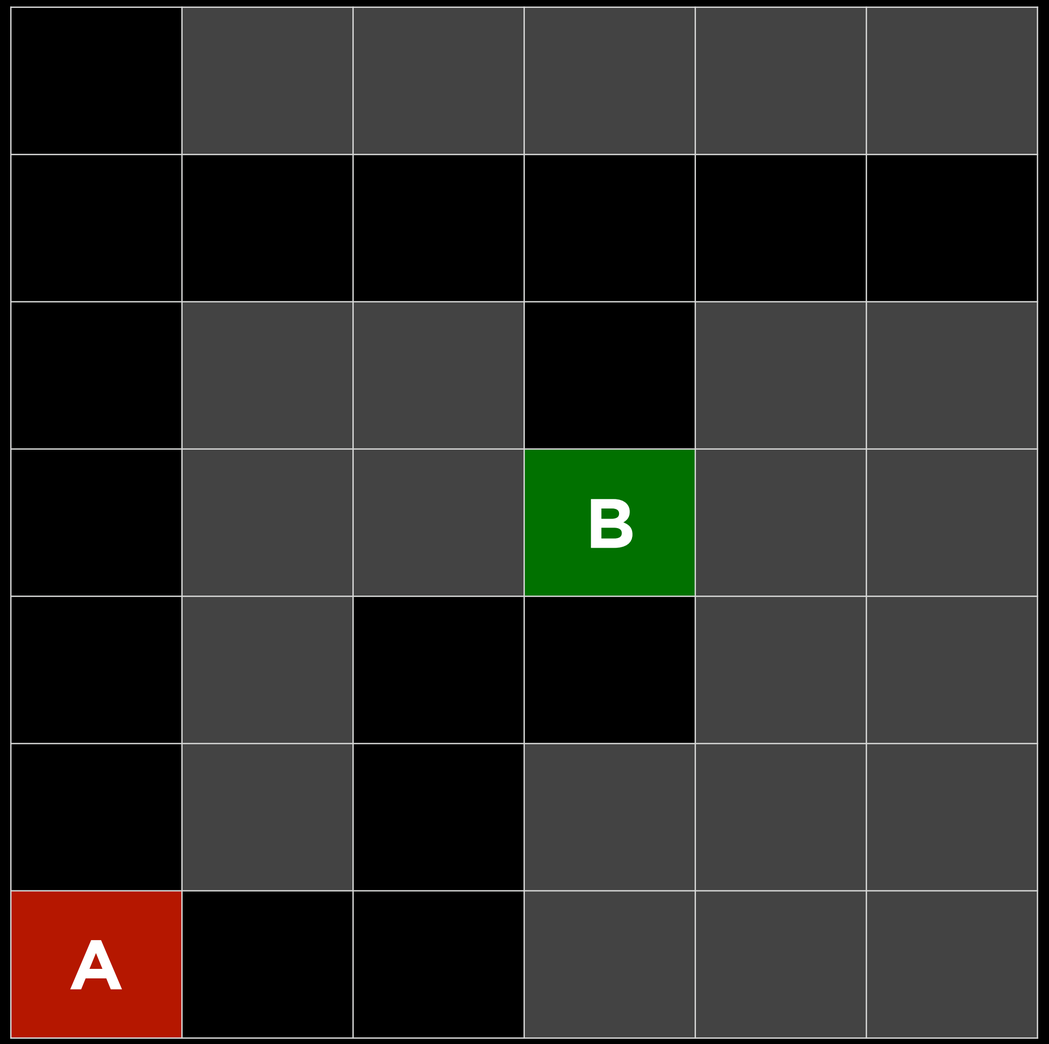
# Depth-First Search



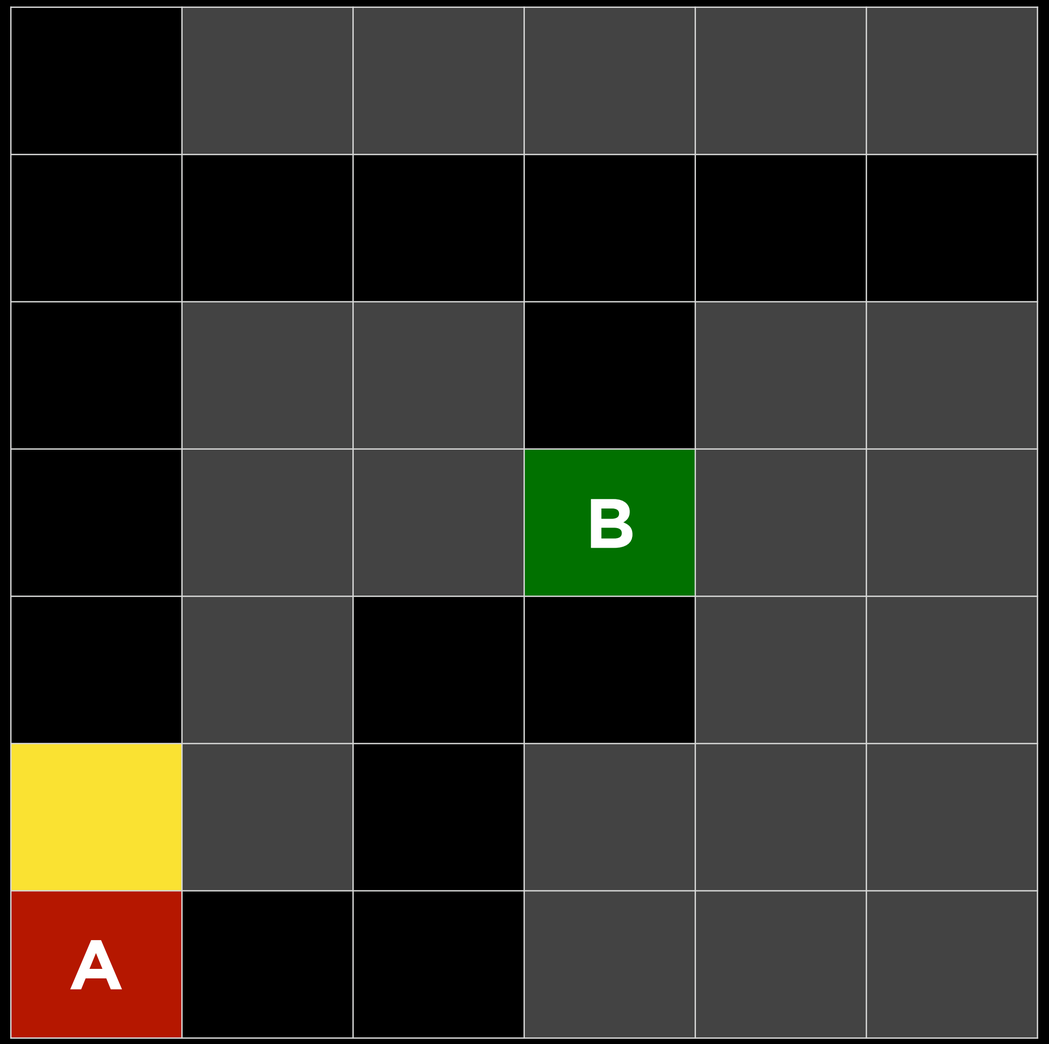
# Depth-First Search



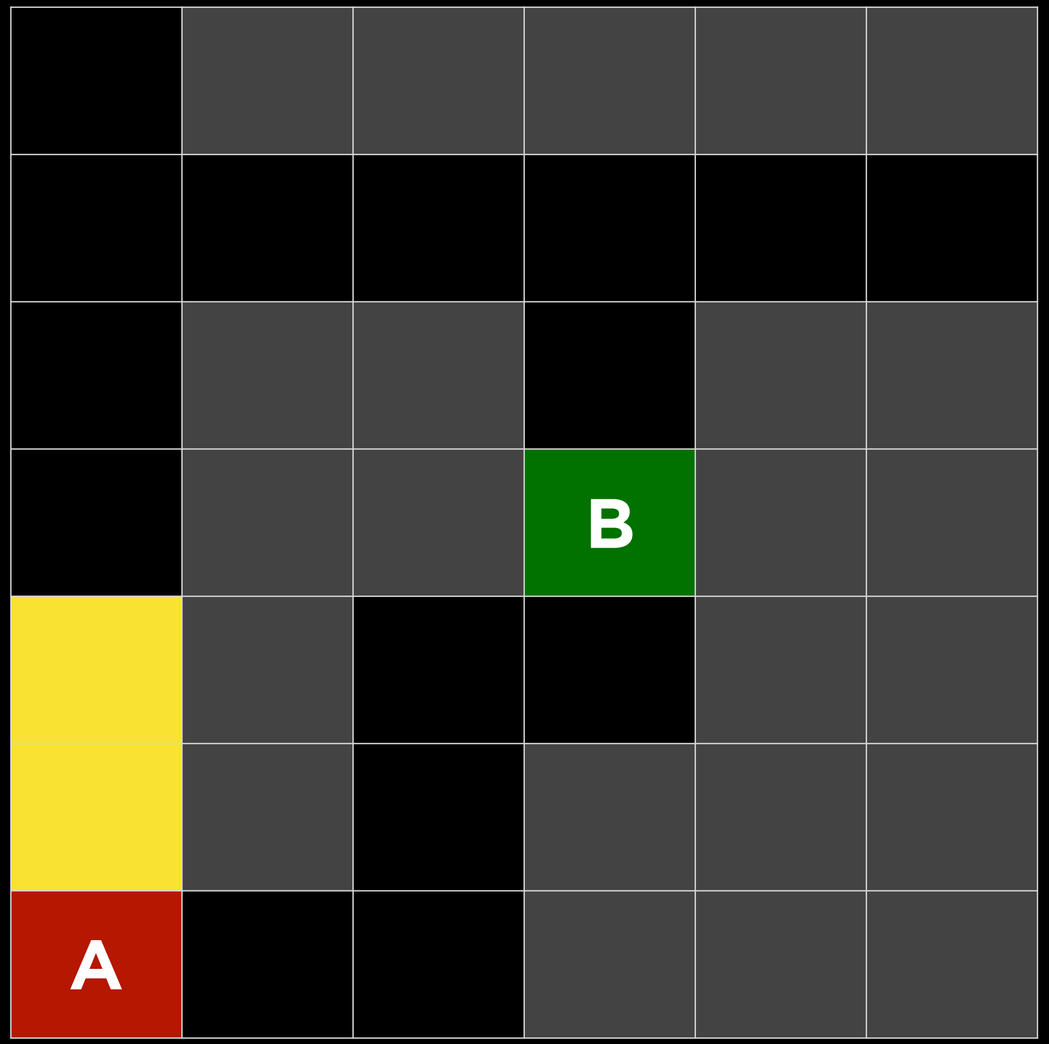
# Depth-First Search



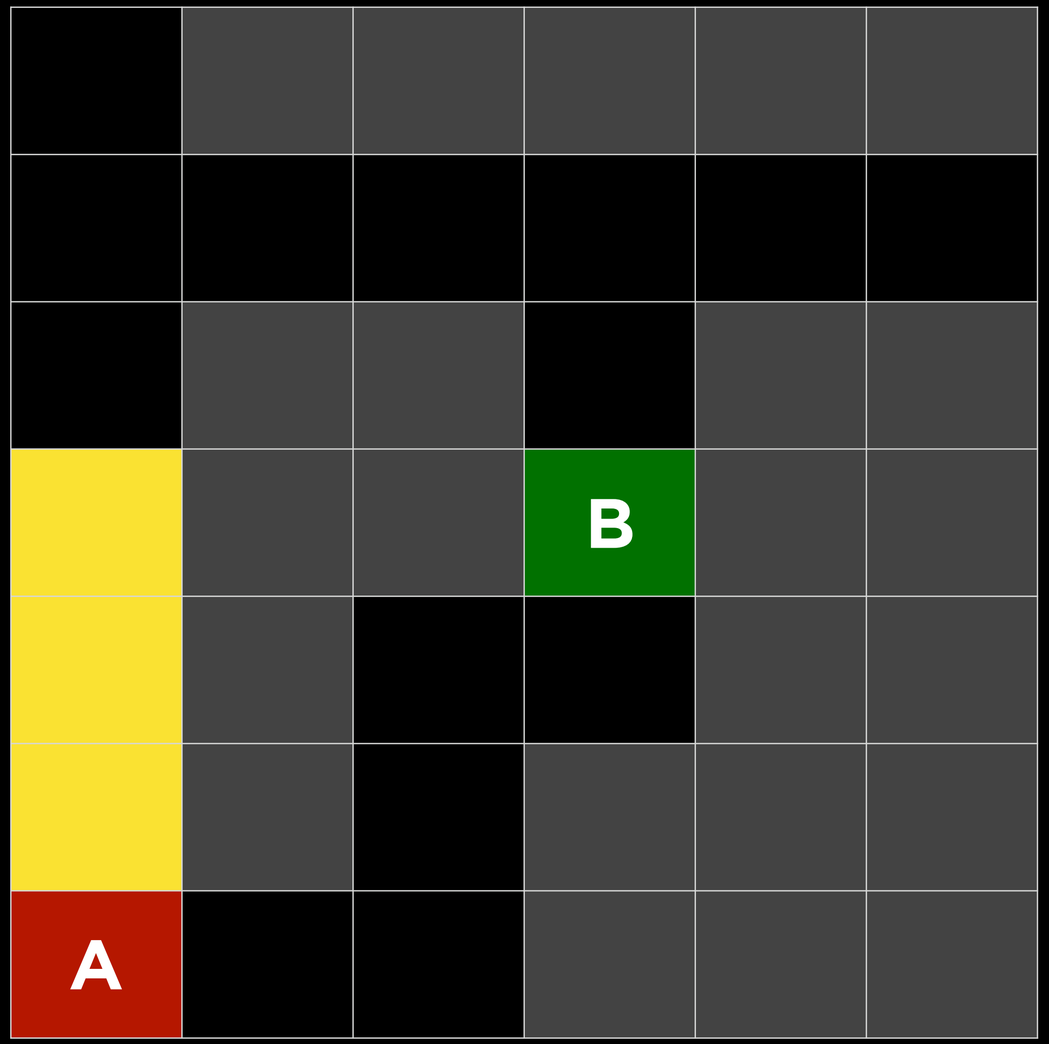
# Depth-First Search



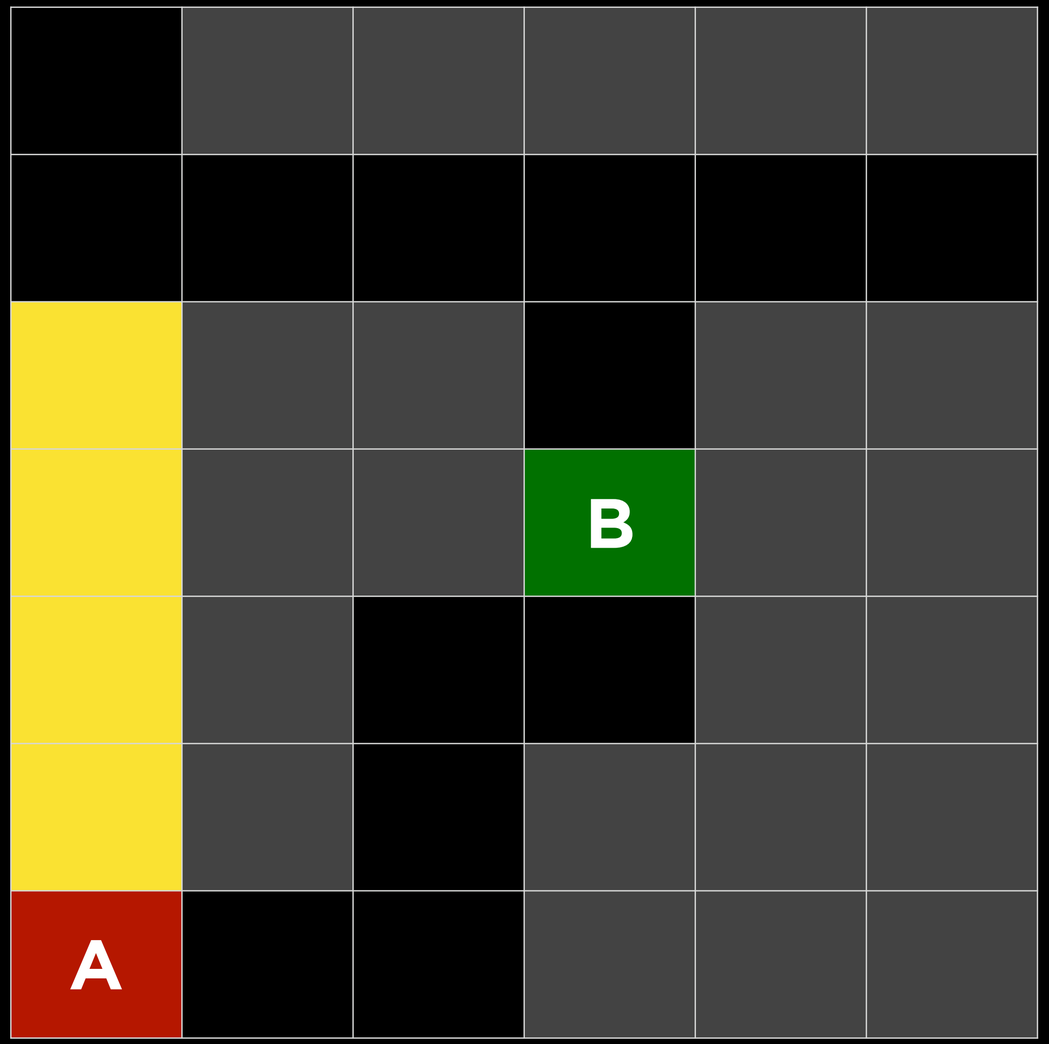
# Depth-First Search



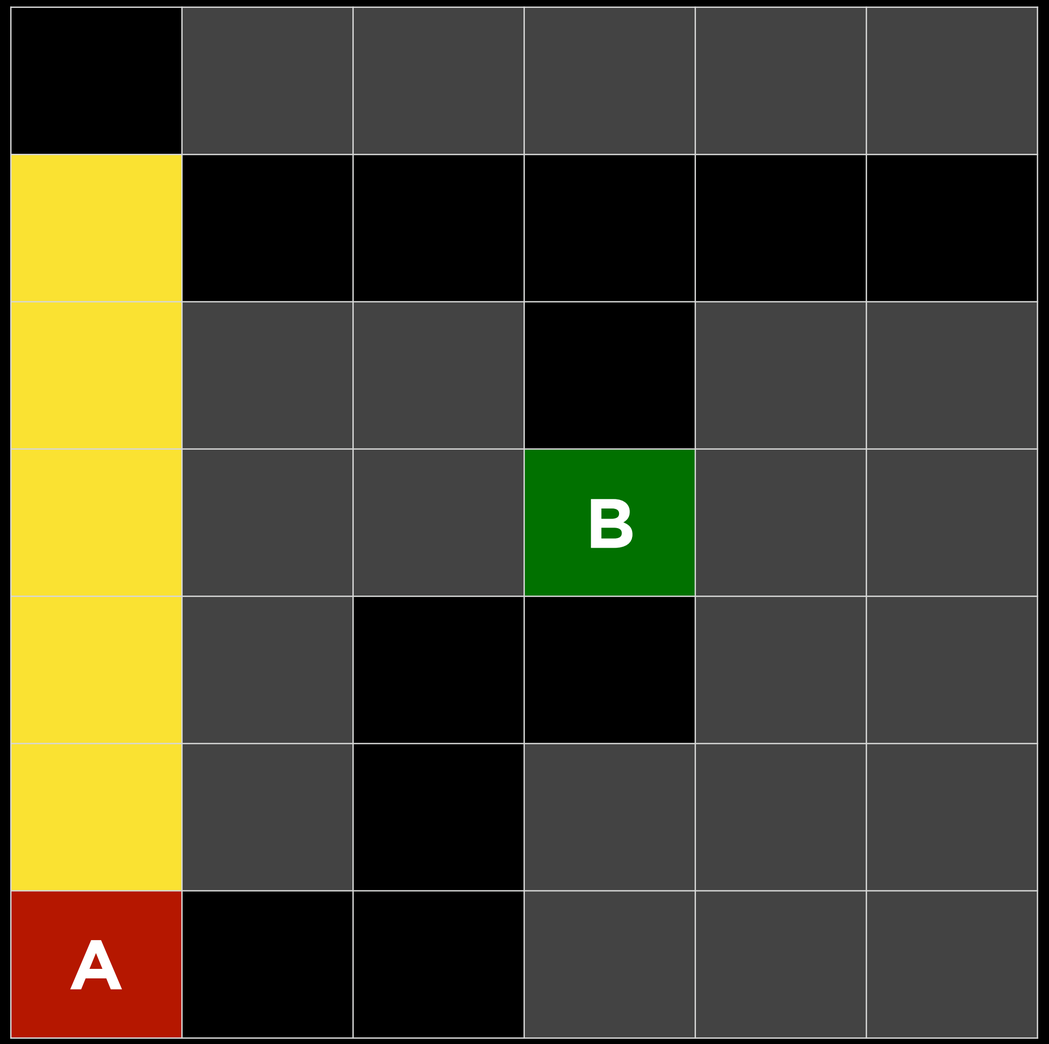
# Depth-First Search



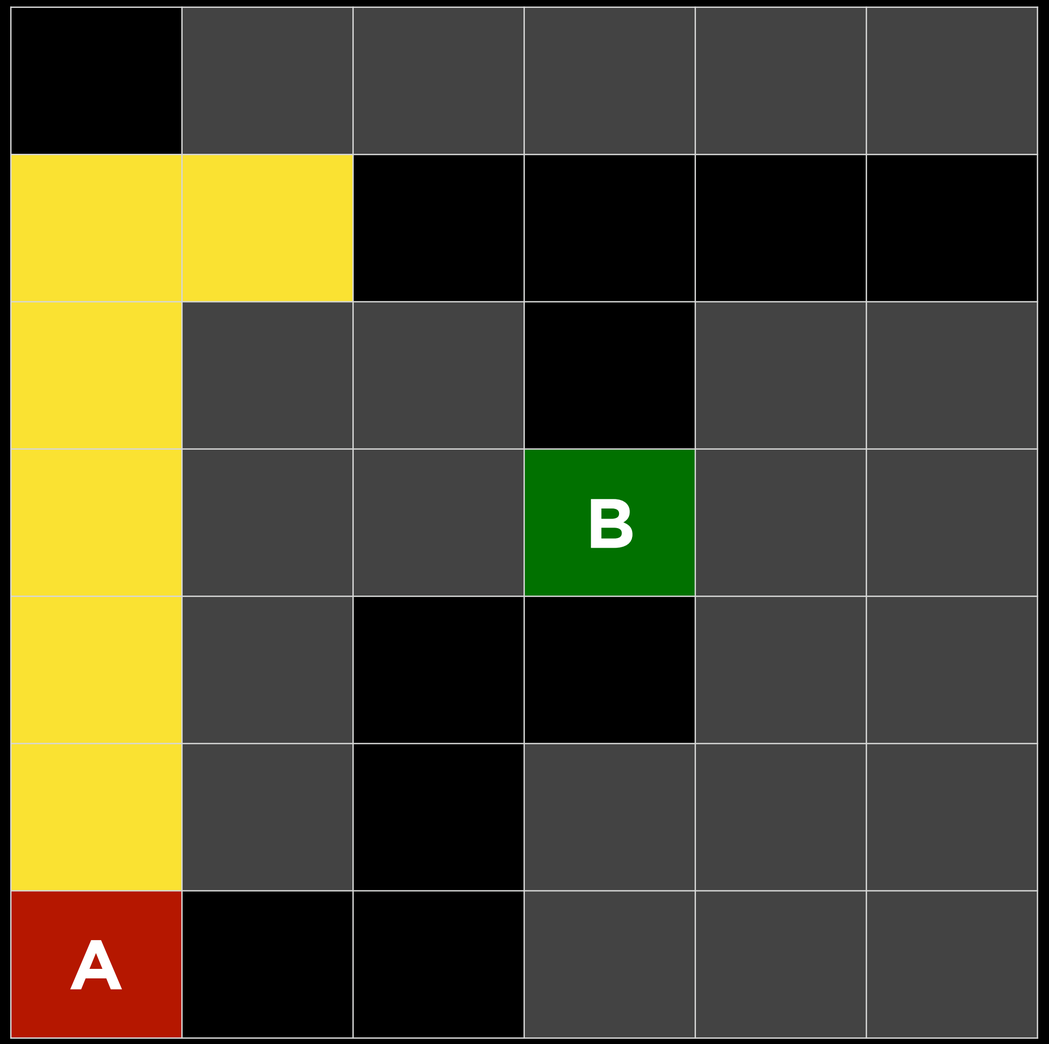
# Depth-First Search



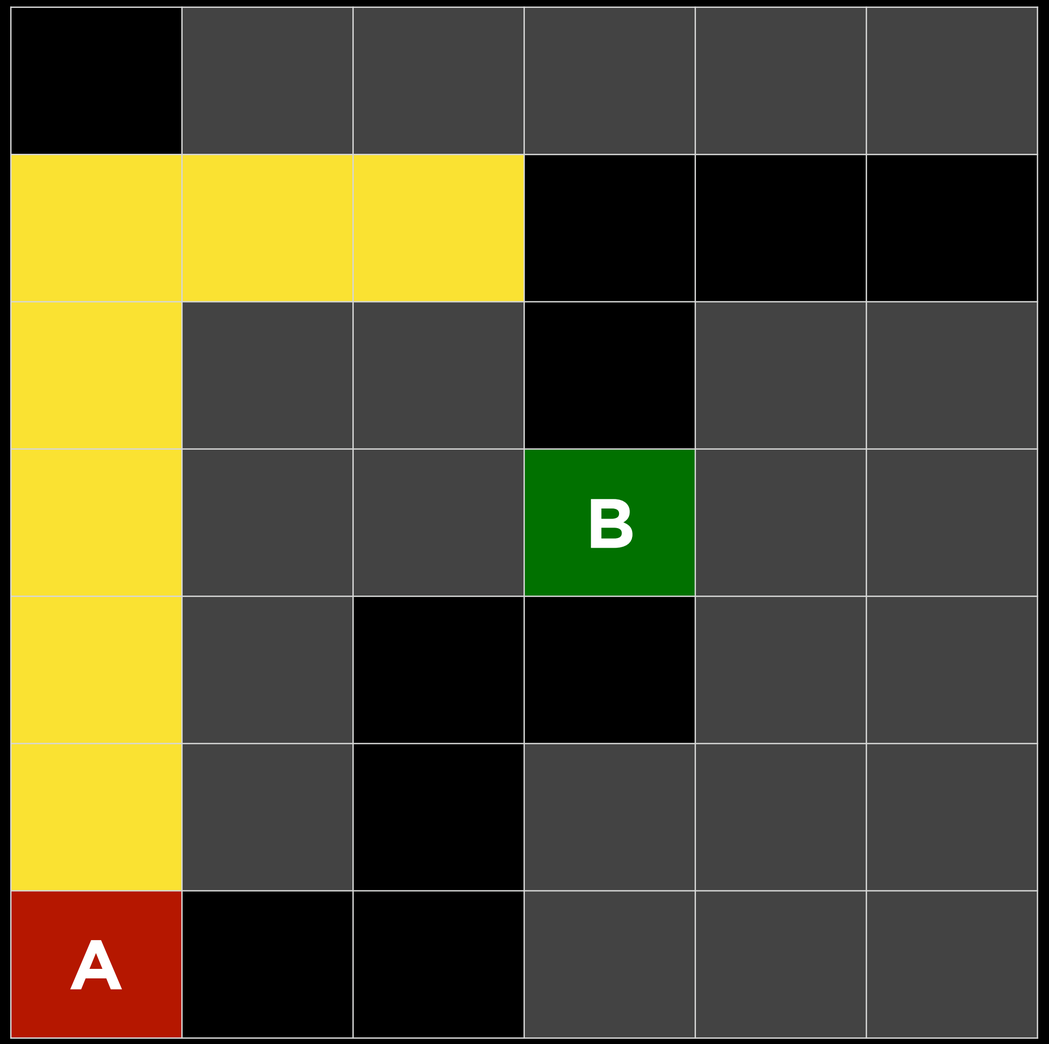
# Depth-First Search



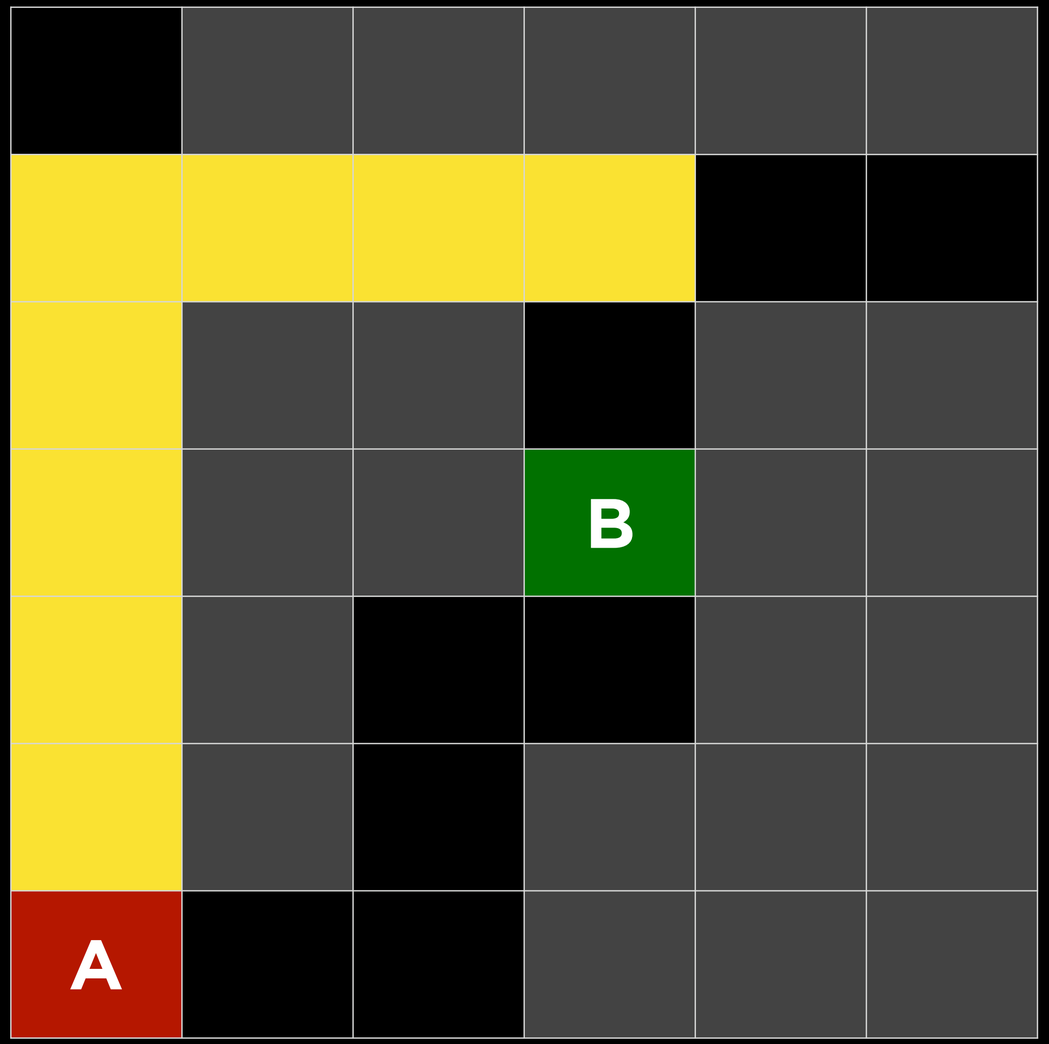
# Depth-First Search



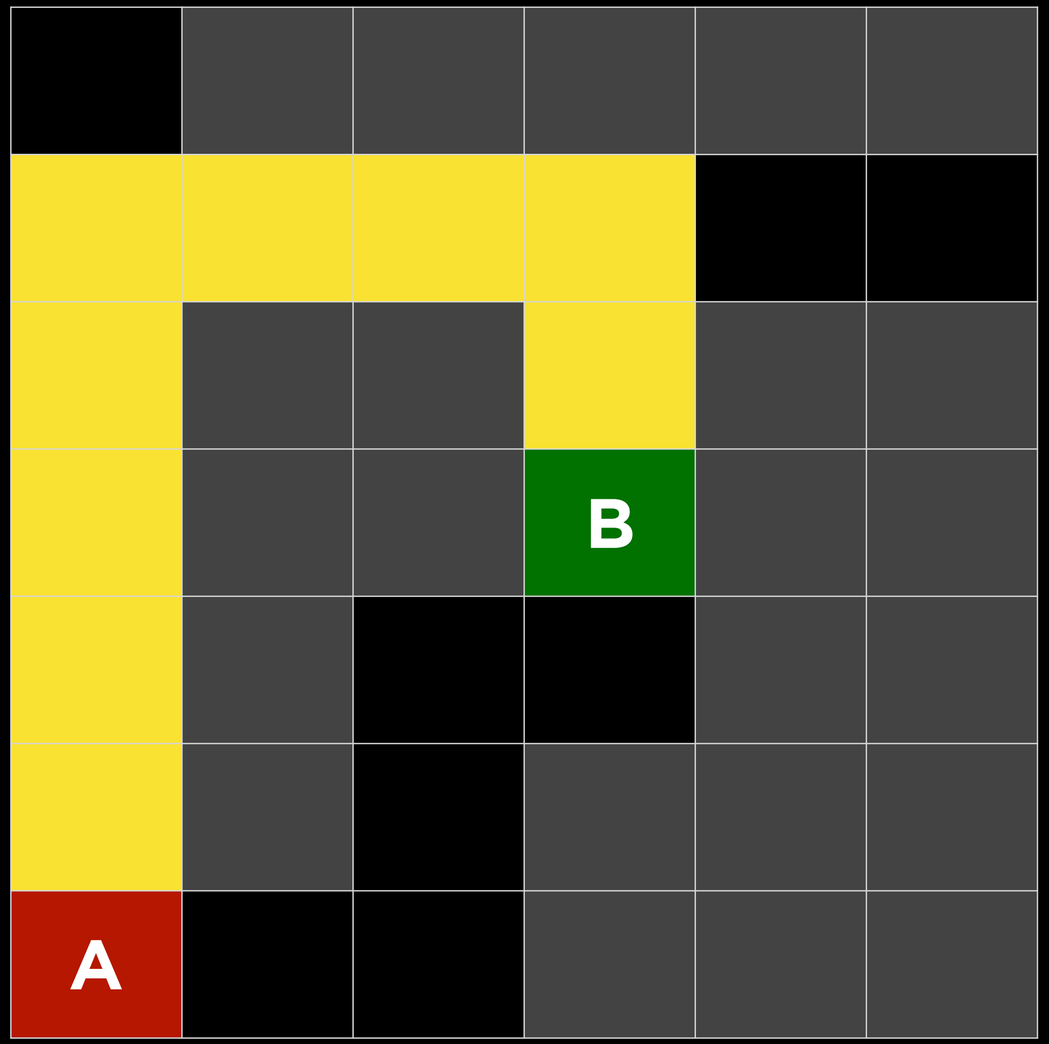
# Depth-First Search



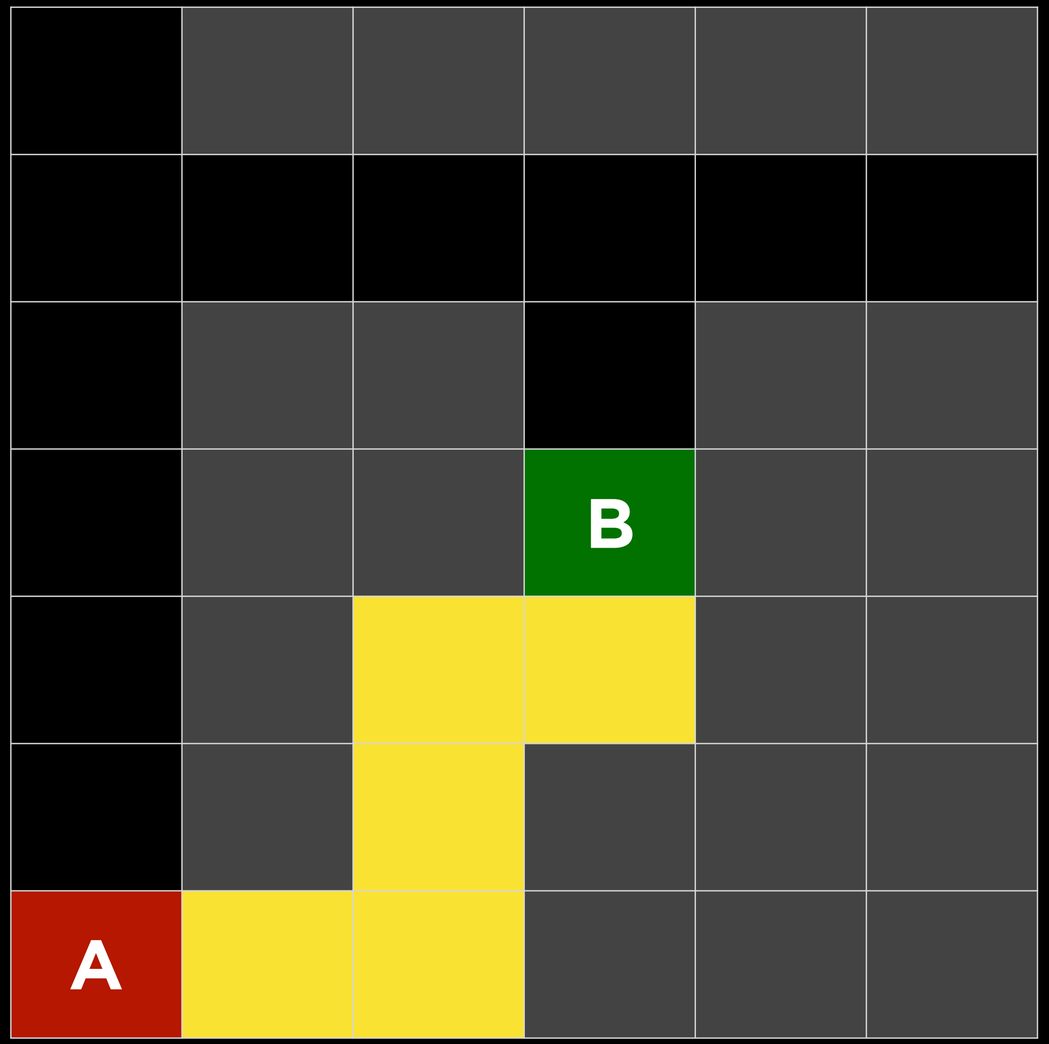
# Depth-First Search



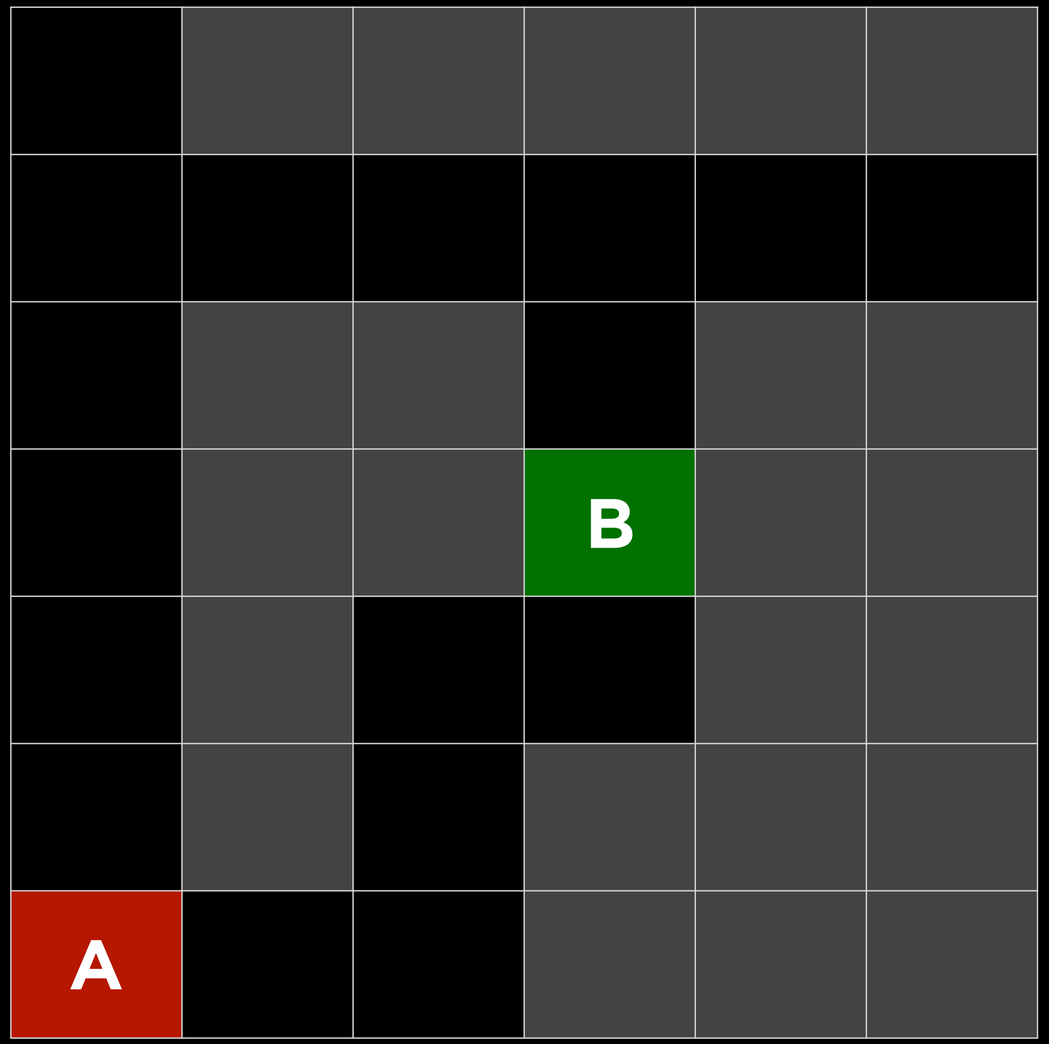
# Depth-First Search



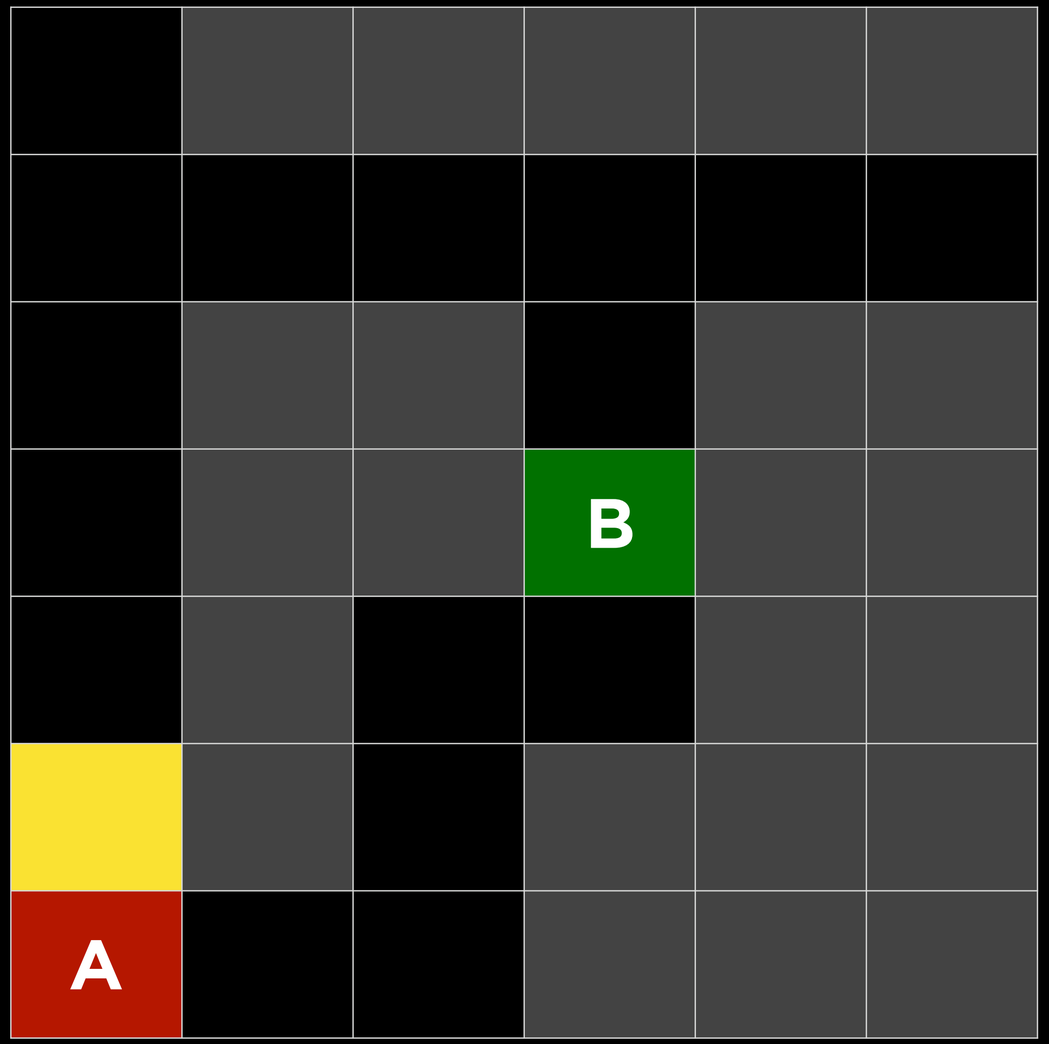
# Depth-First Search



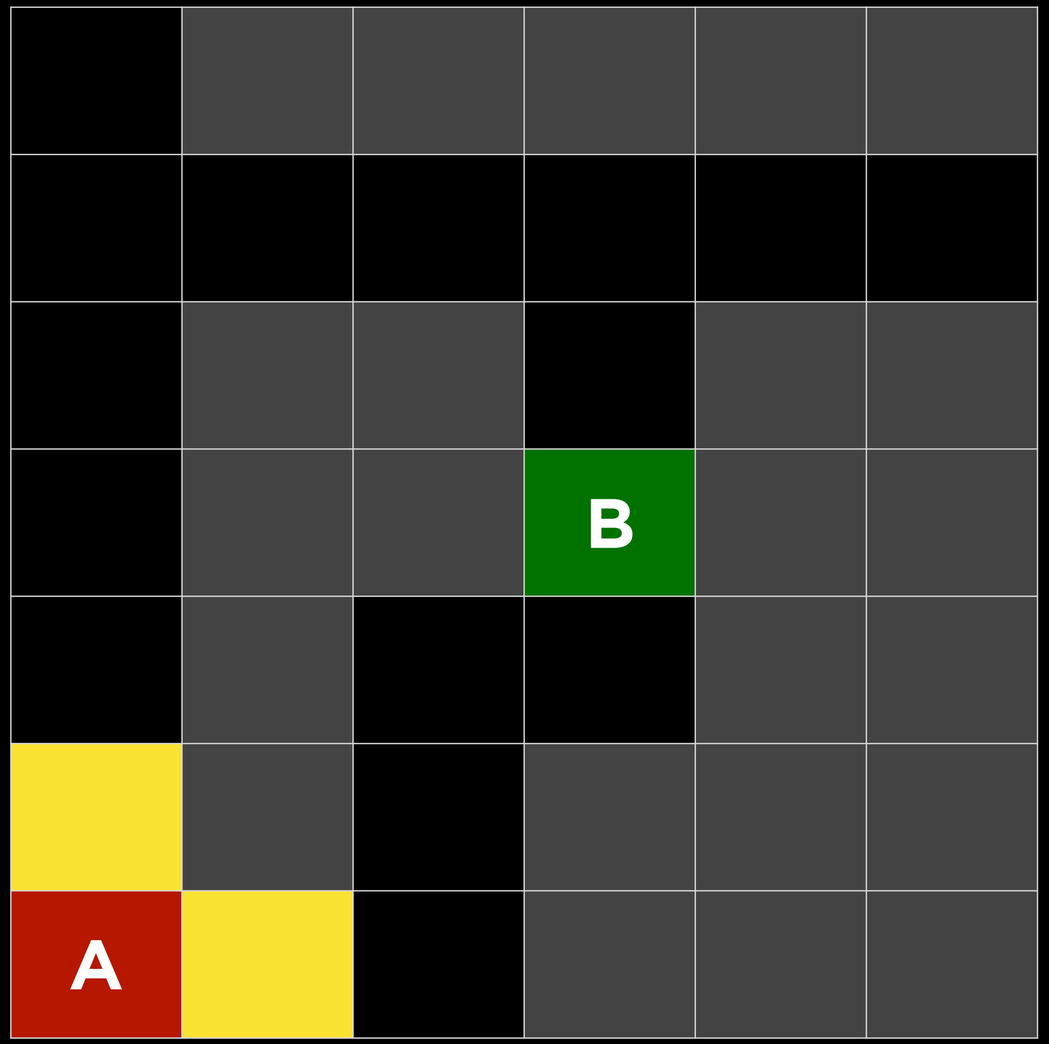
# Breadth-First Search



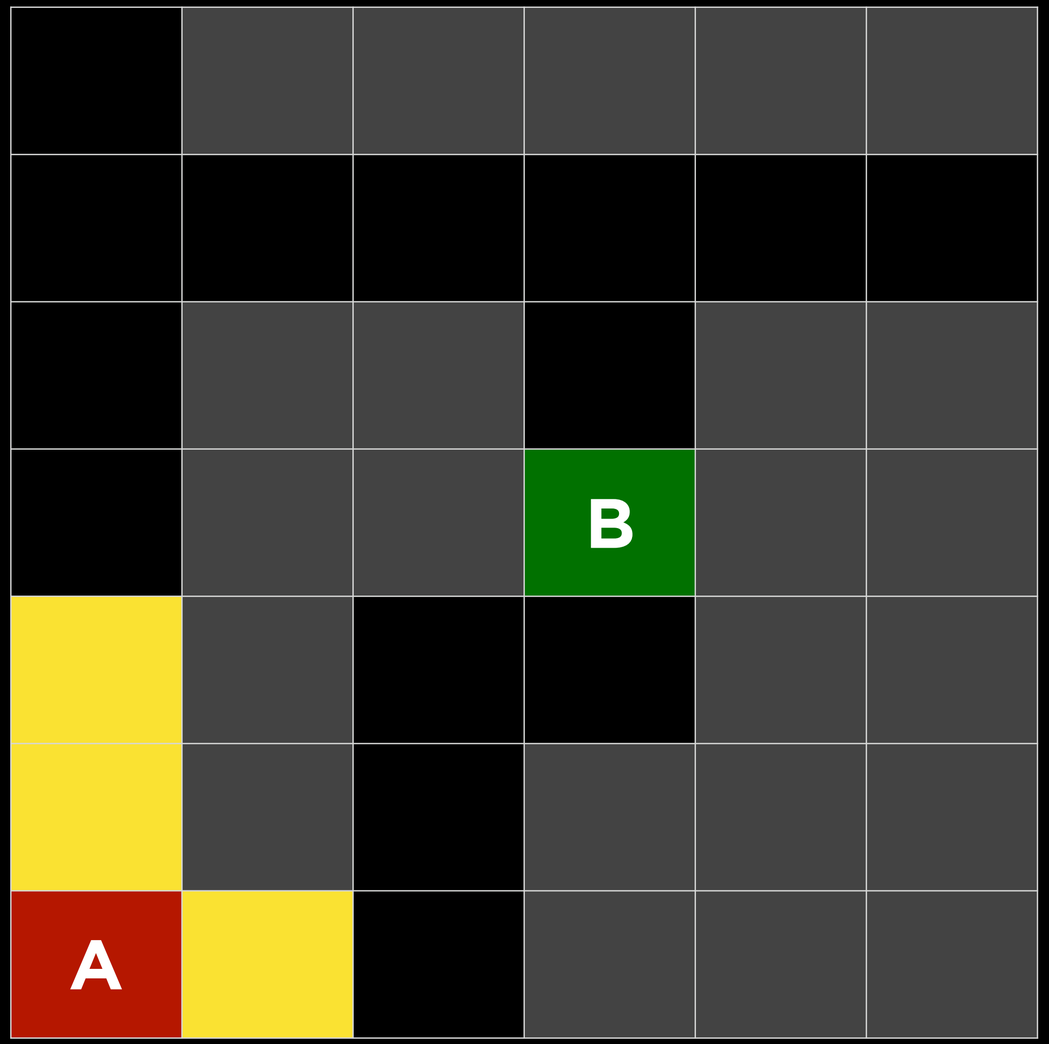
# Breadth-First Search



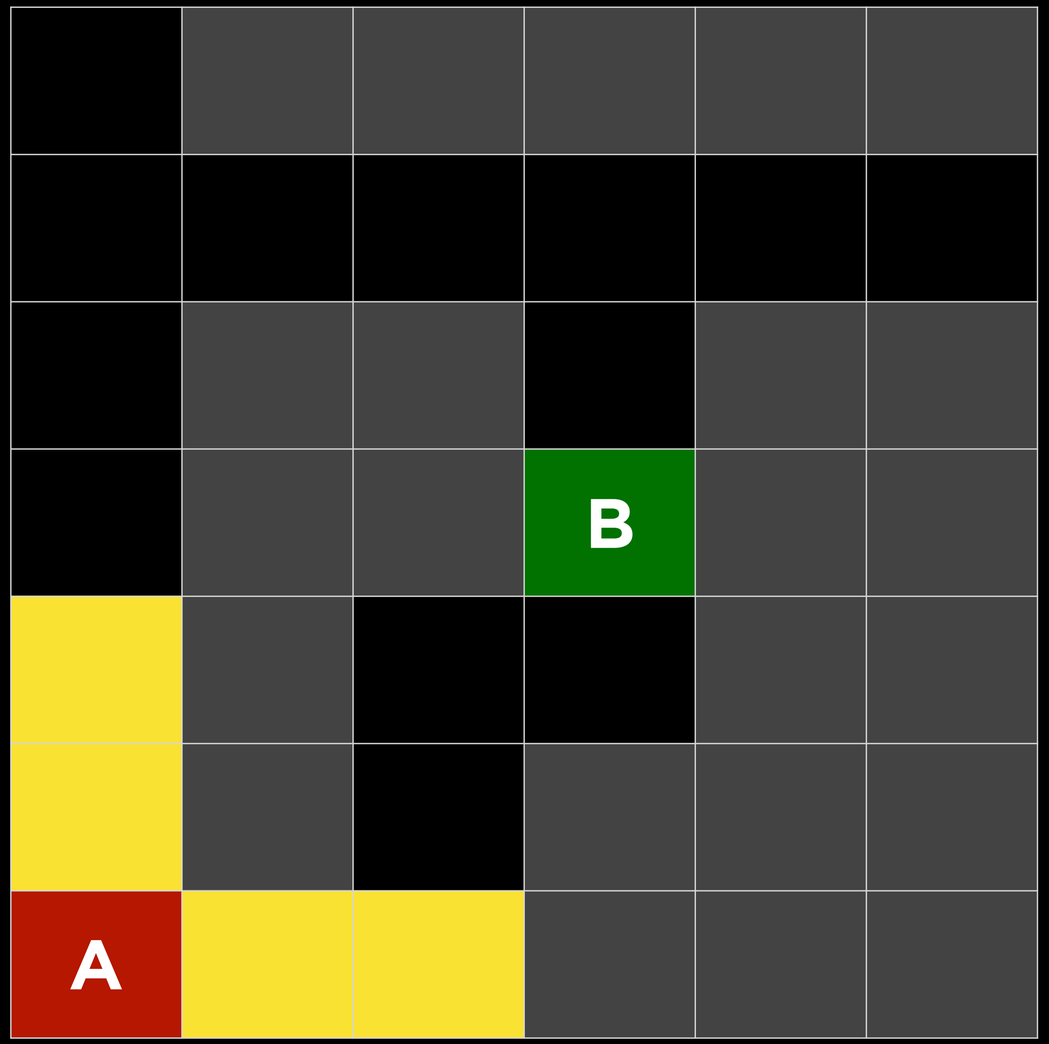
# Breadth-First Search



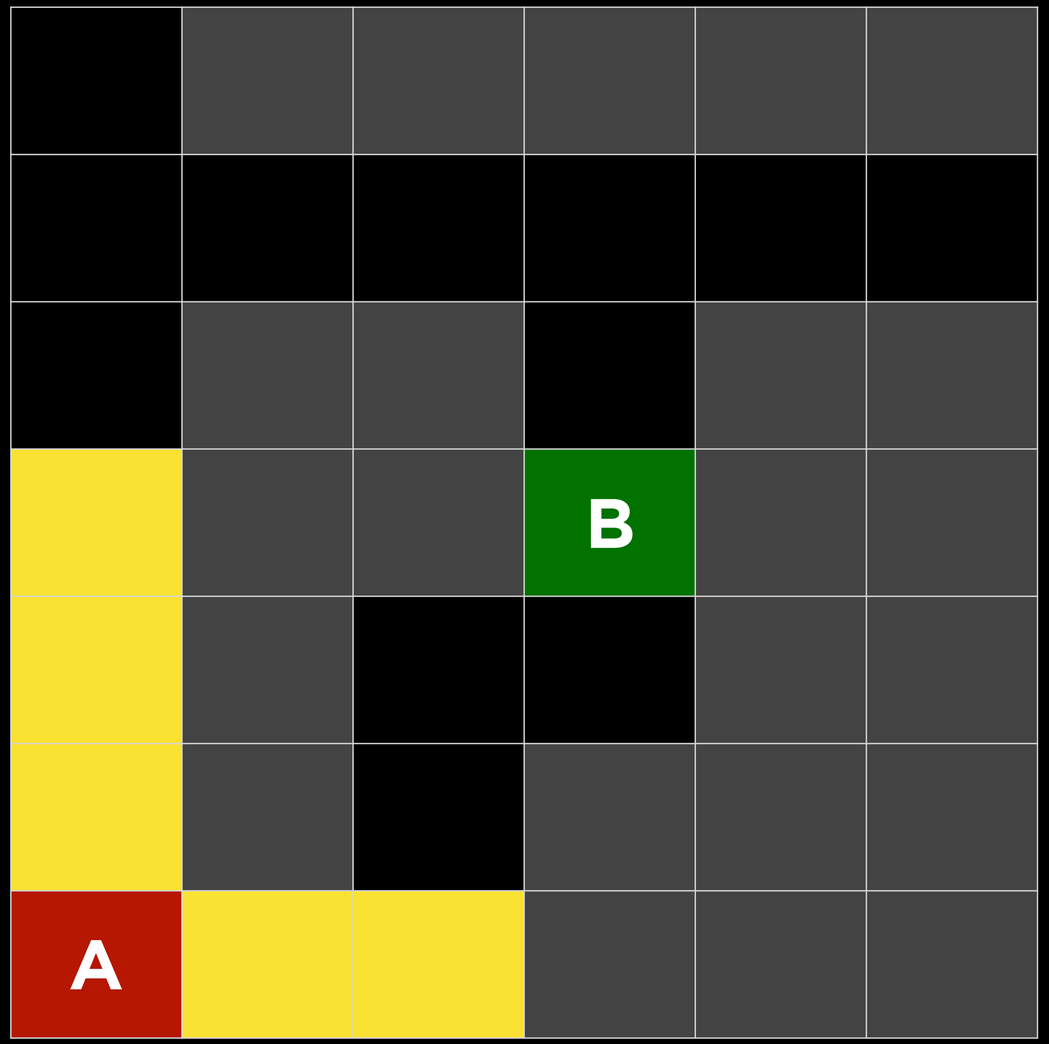
# Breadth-First Search



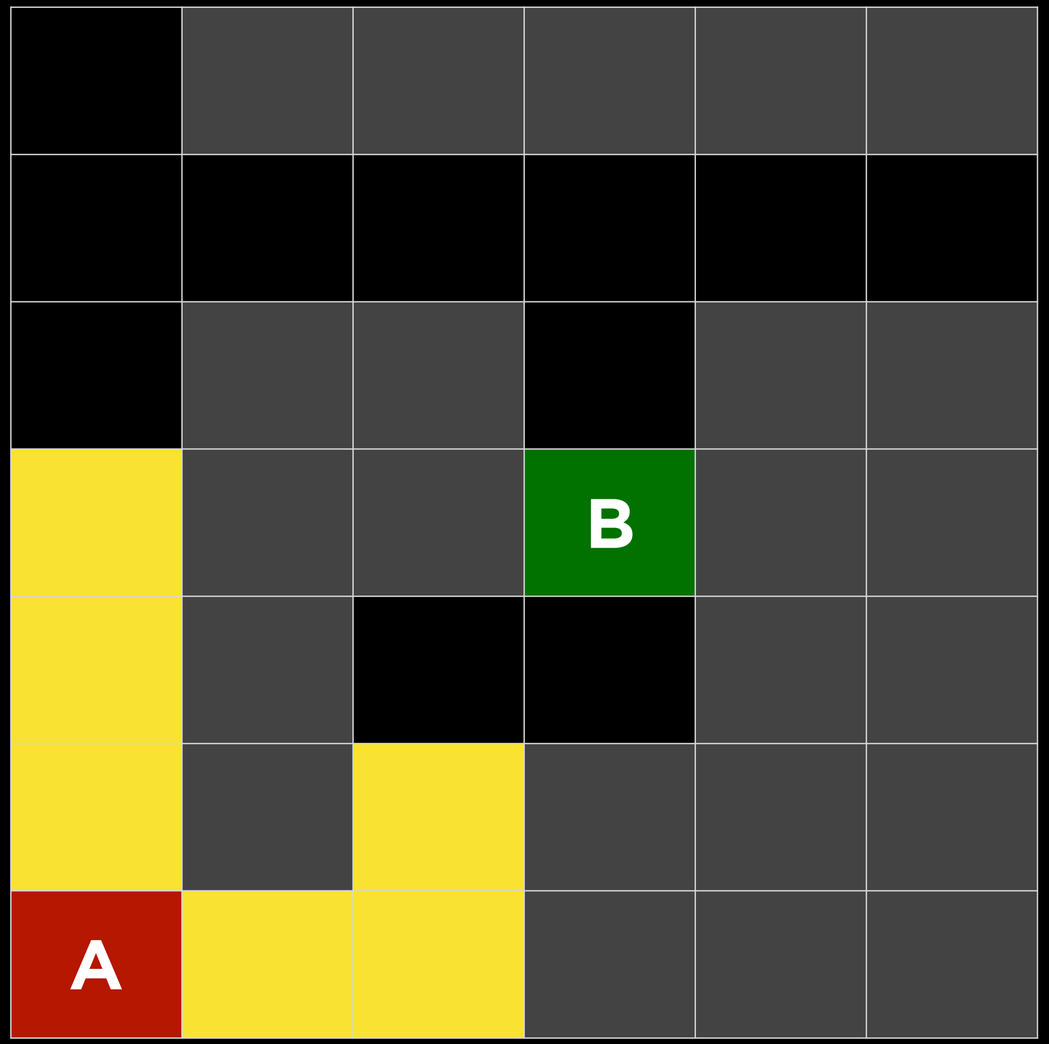
# Breadth-First Search



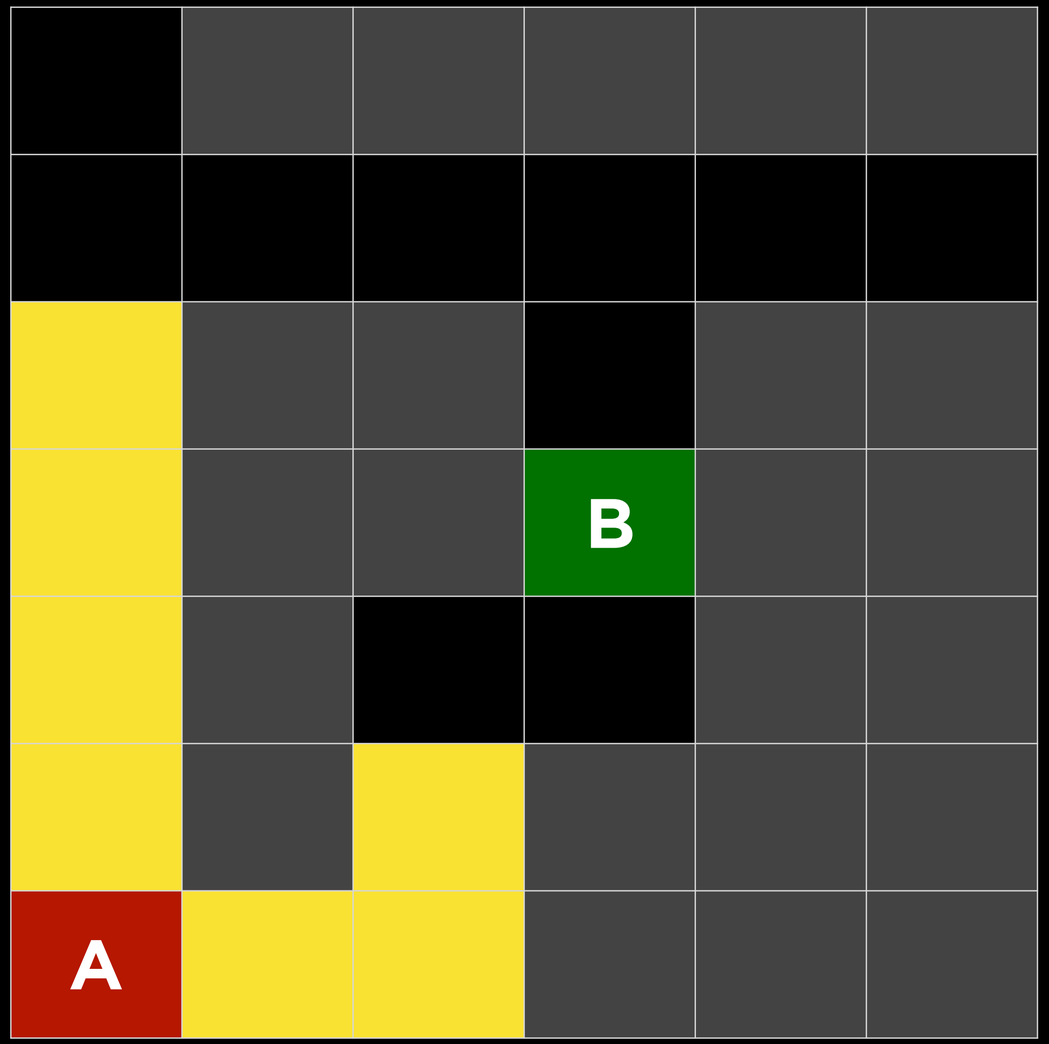
# Breadth-First Search



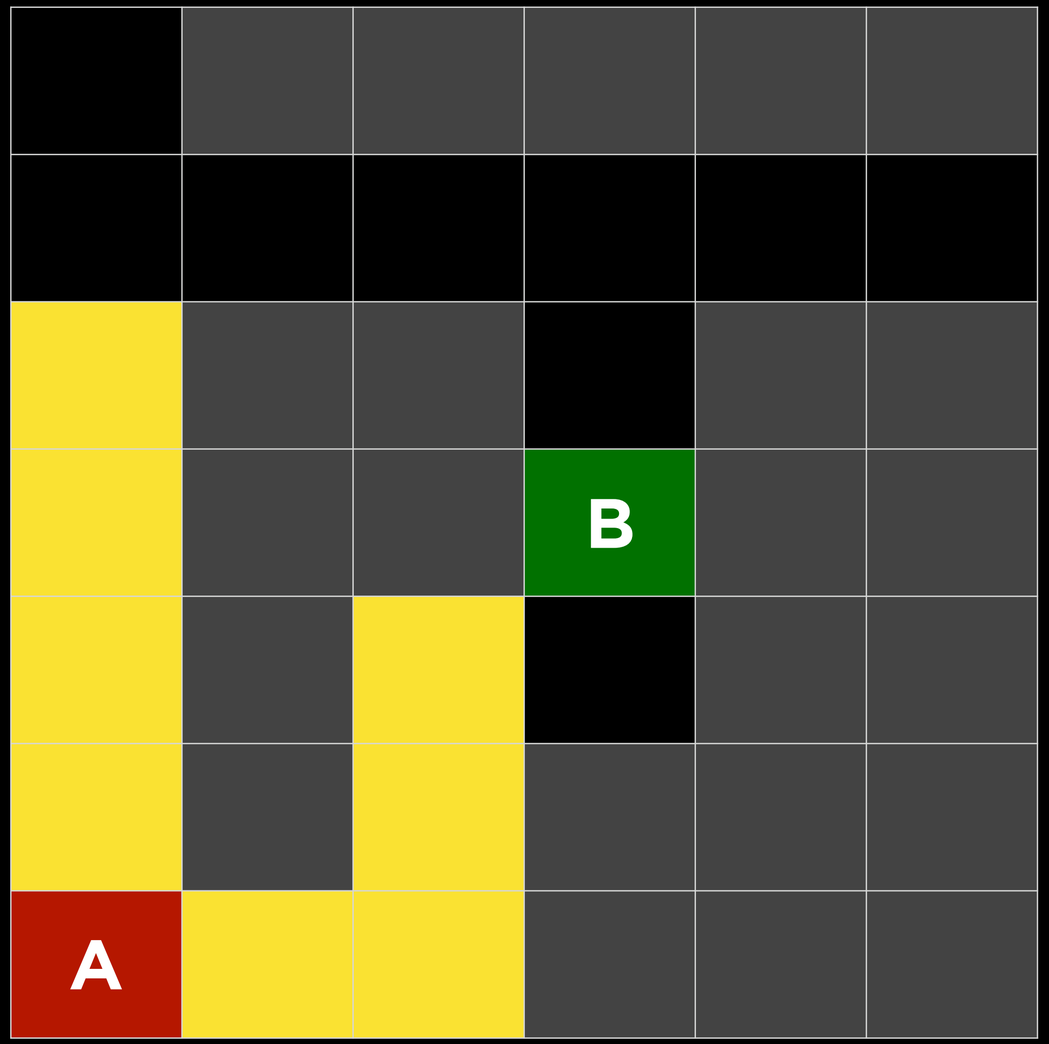
# Breadth-First Search



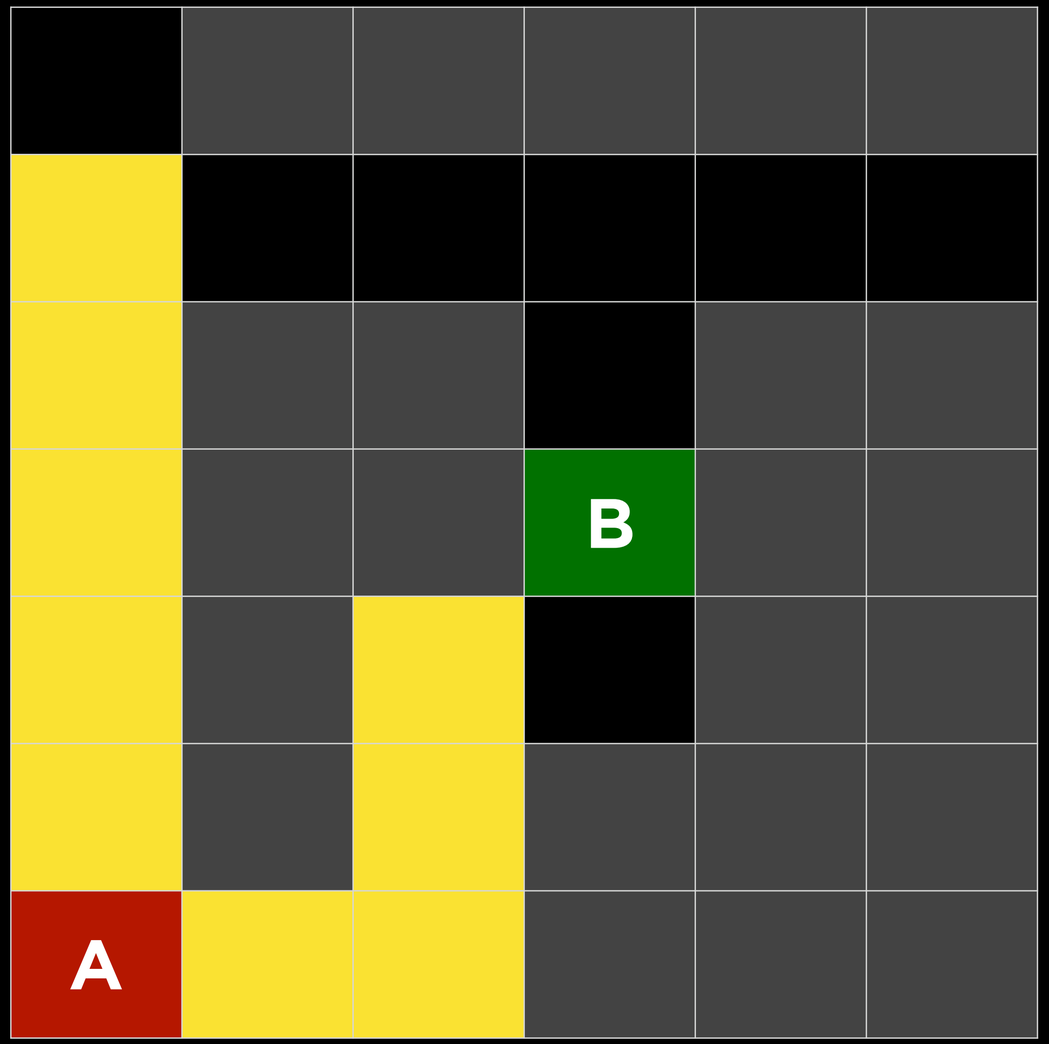
# Breadth-First Search



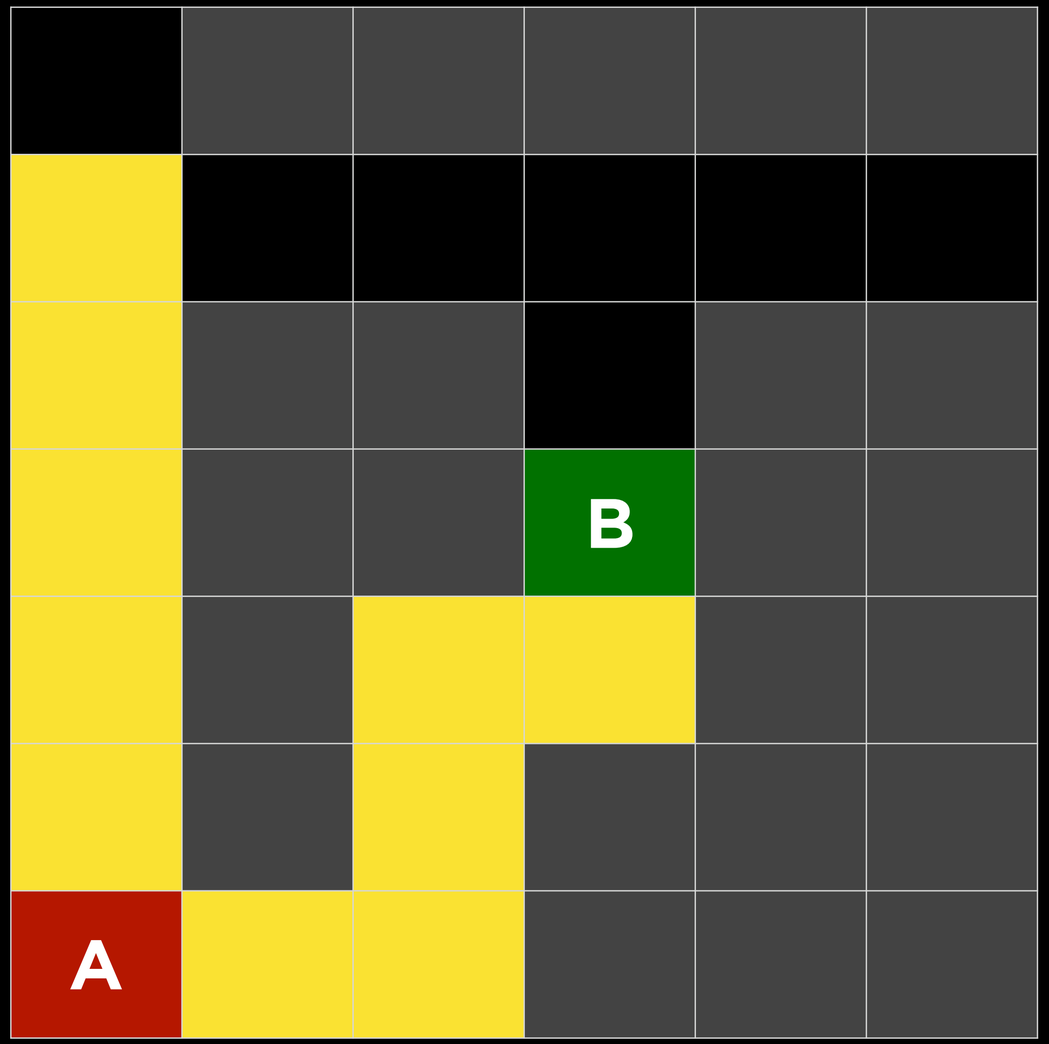
# Breadth-First Search



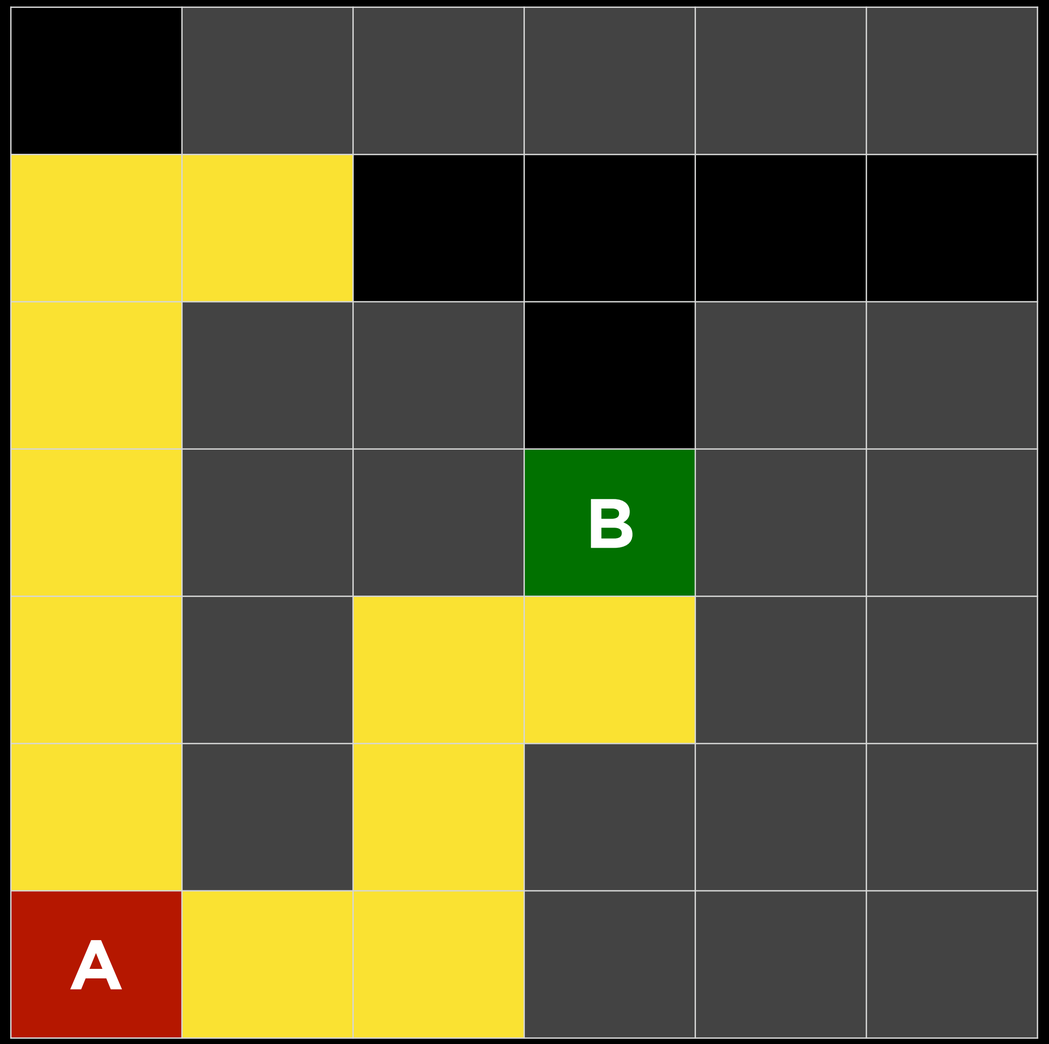
# Breadth-First Search



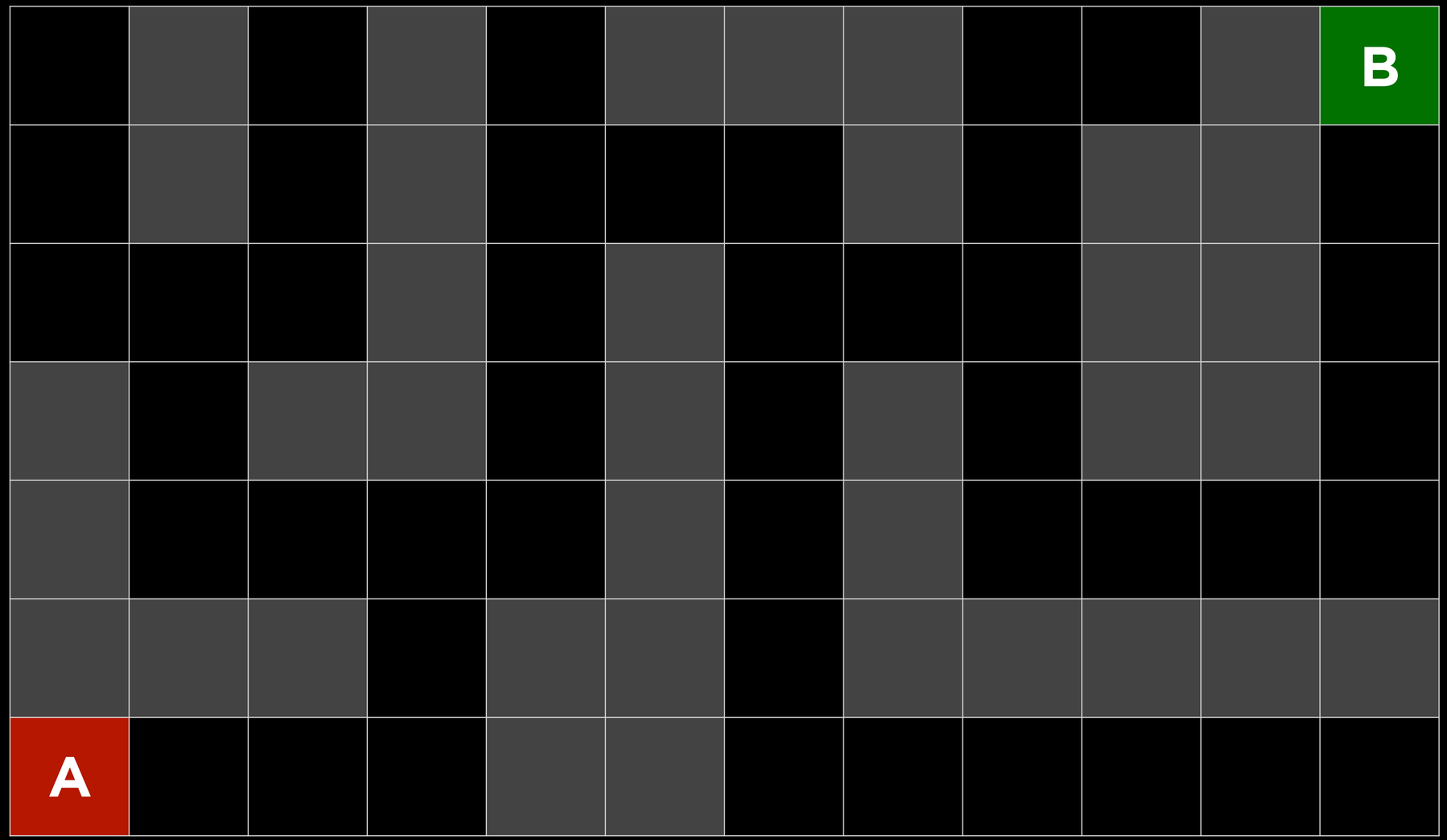
# Breadth-First Search



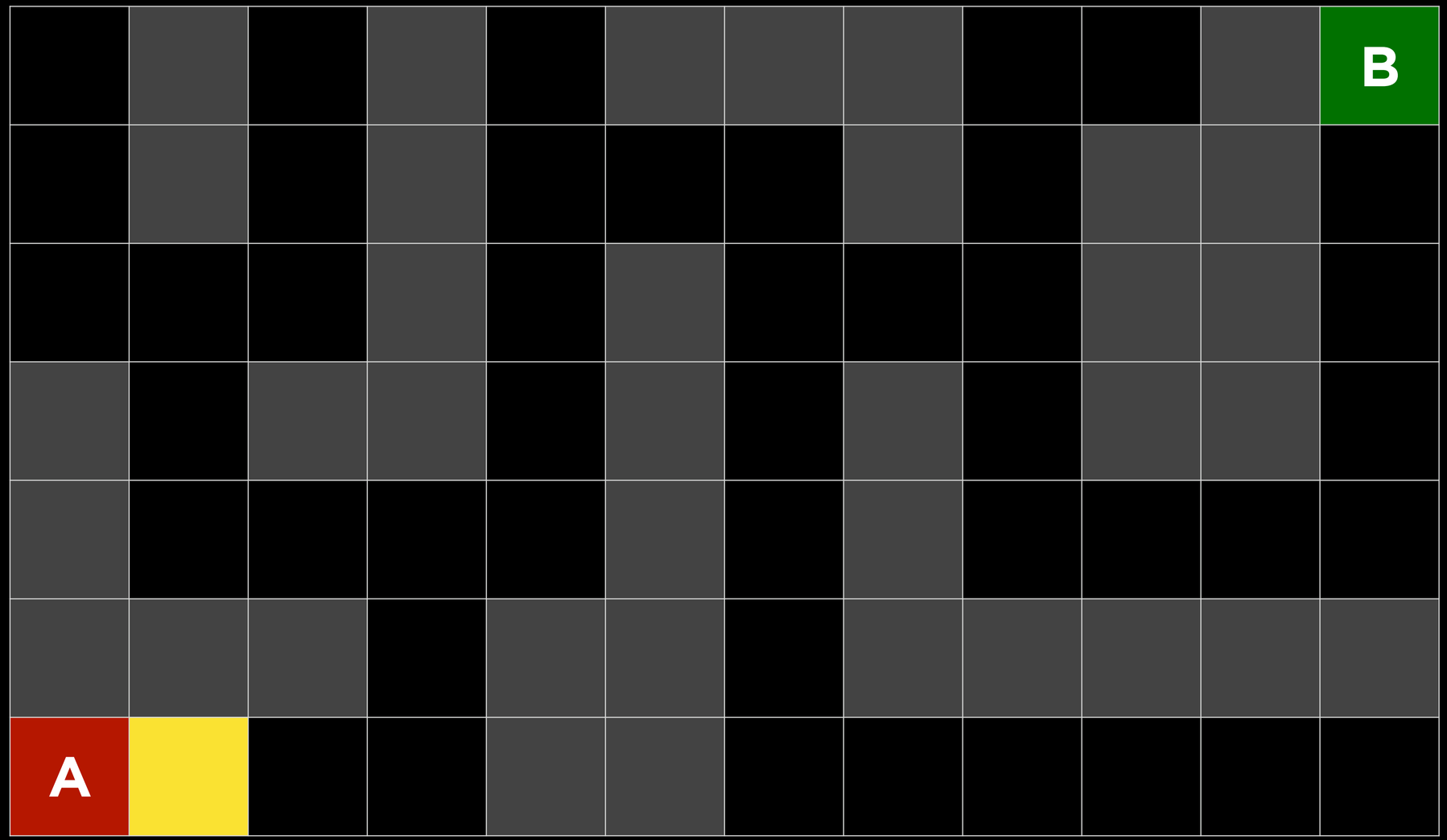
# Breadth-First Search



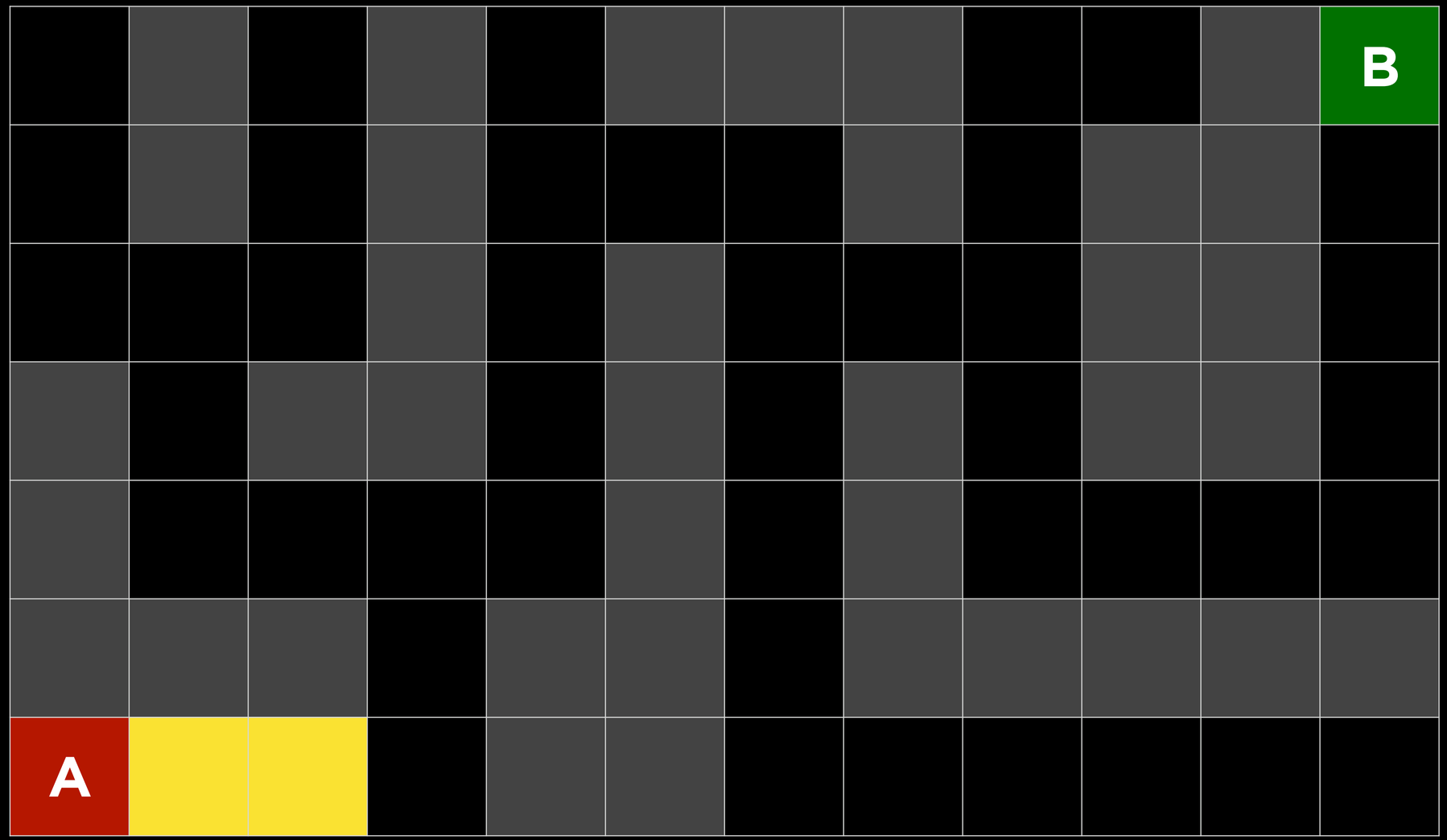
# Breadth-First Search



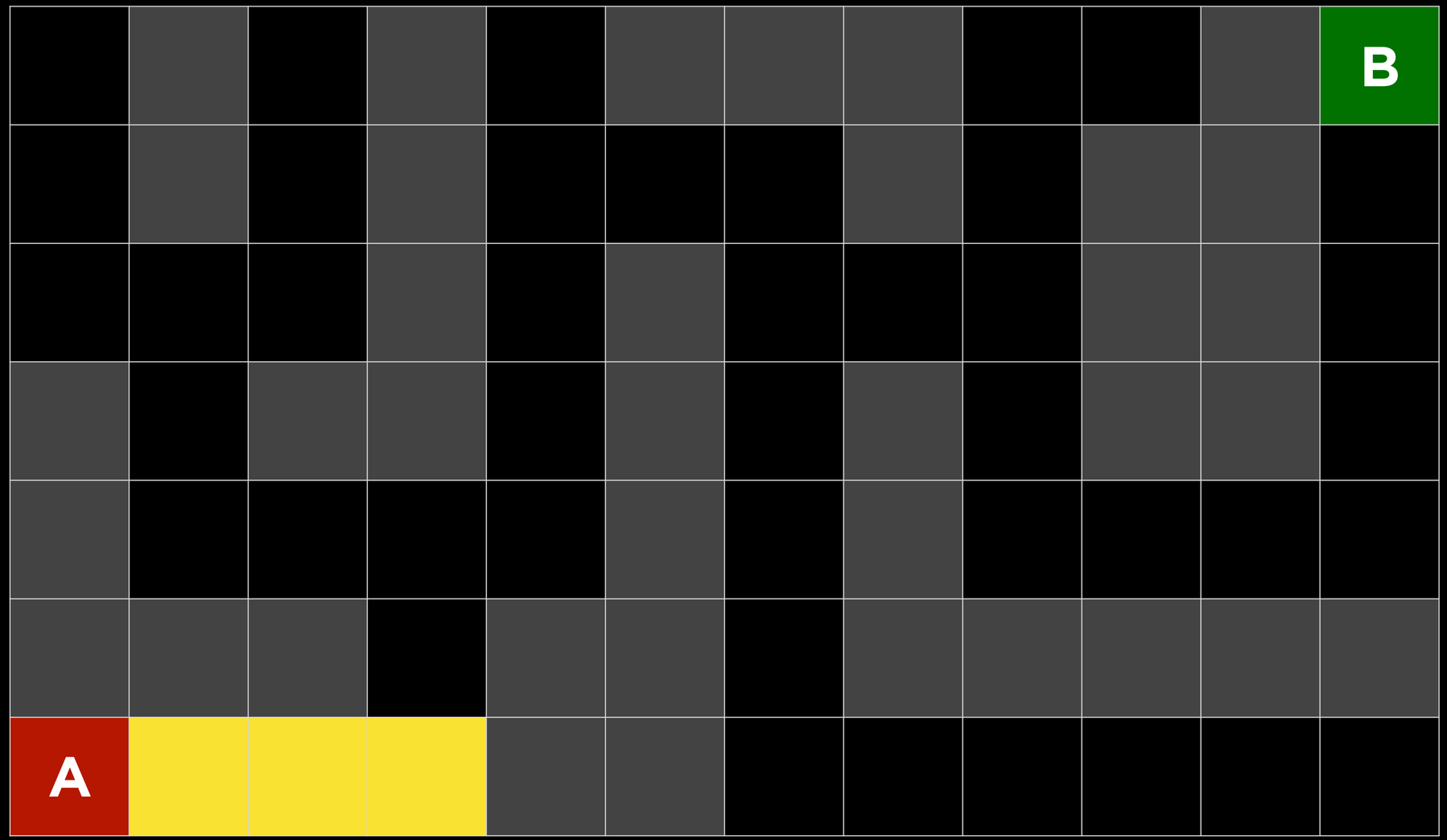
# Breadth-First Search



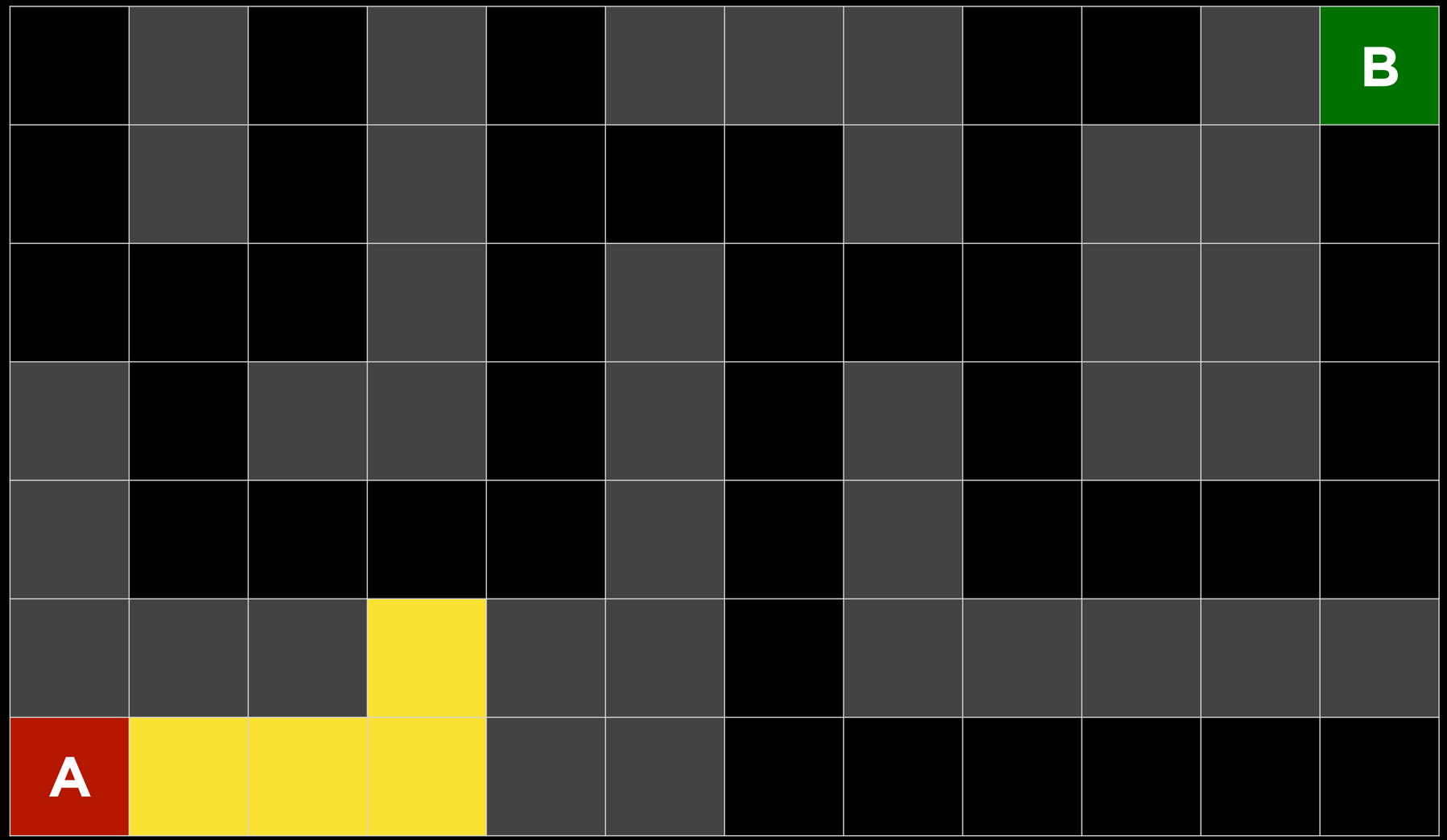
# Breadth-First Search



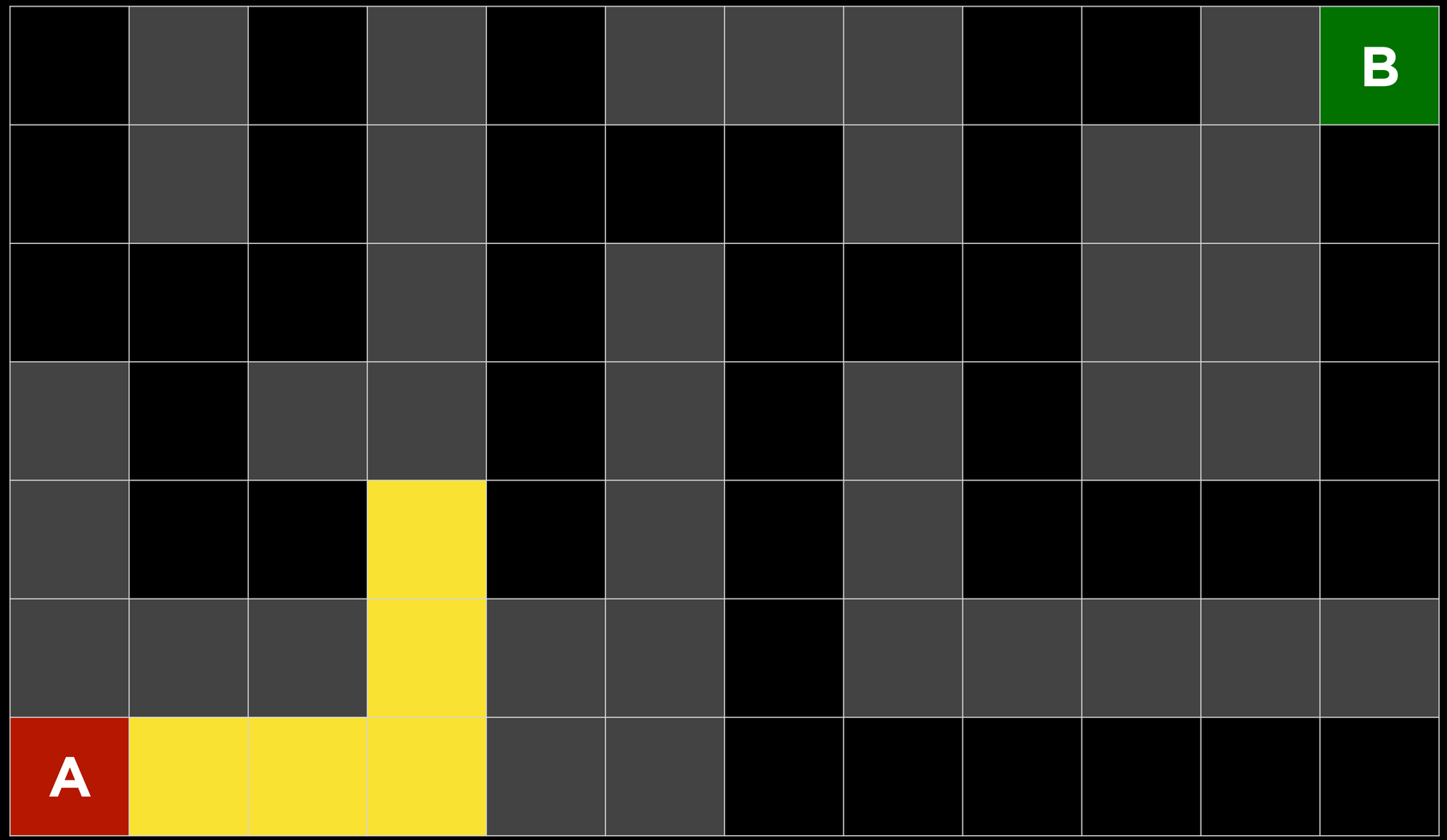
# Breadth-First Search



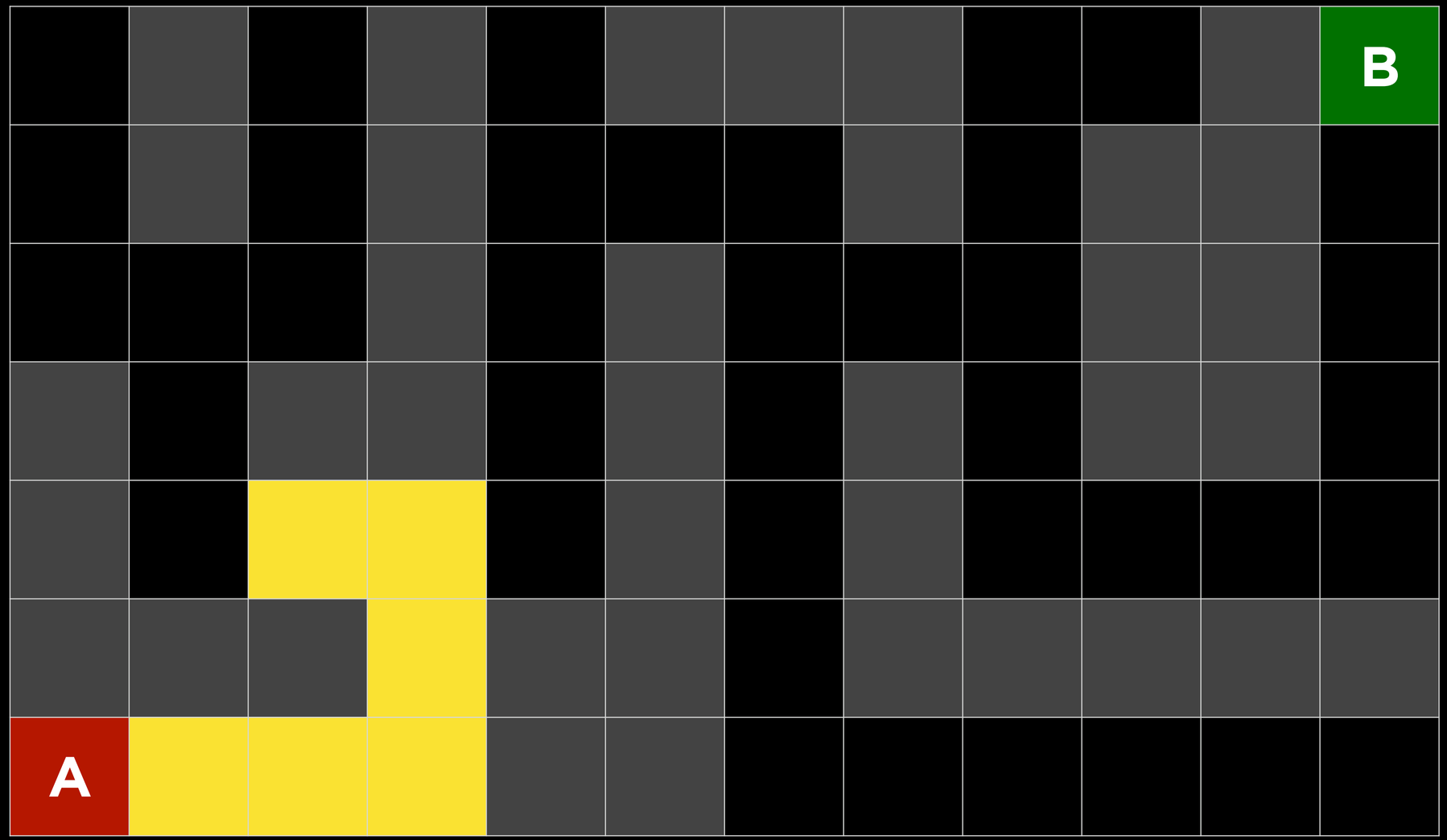
# Breadth-First Search



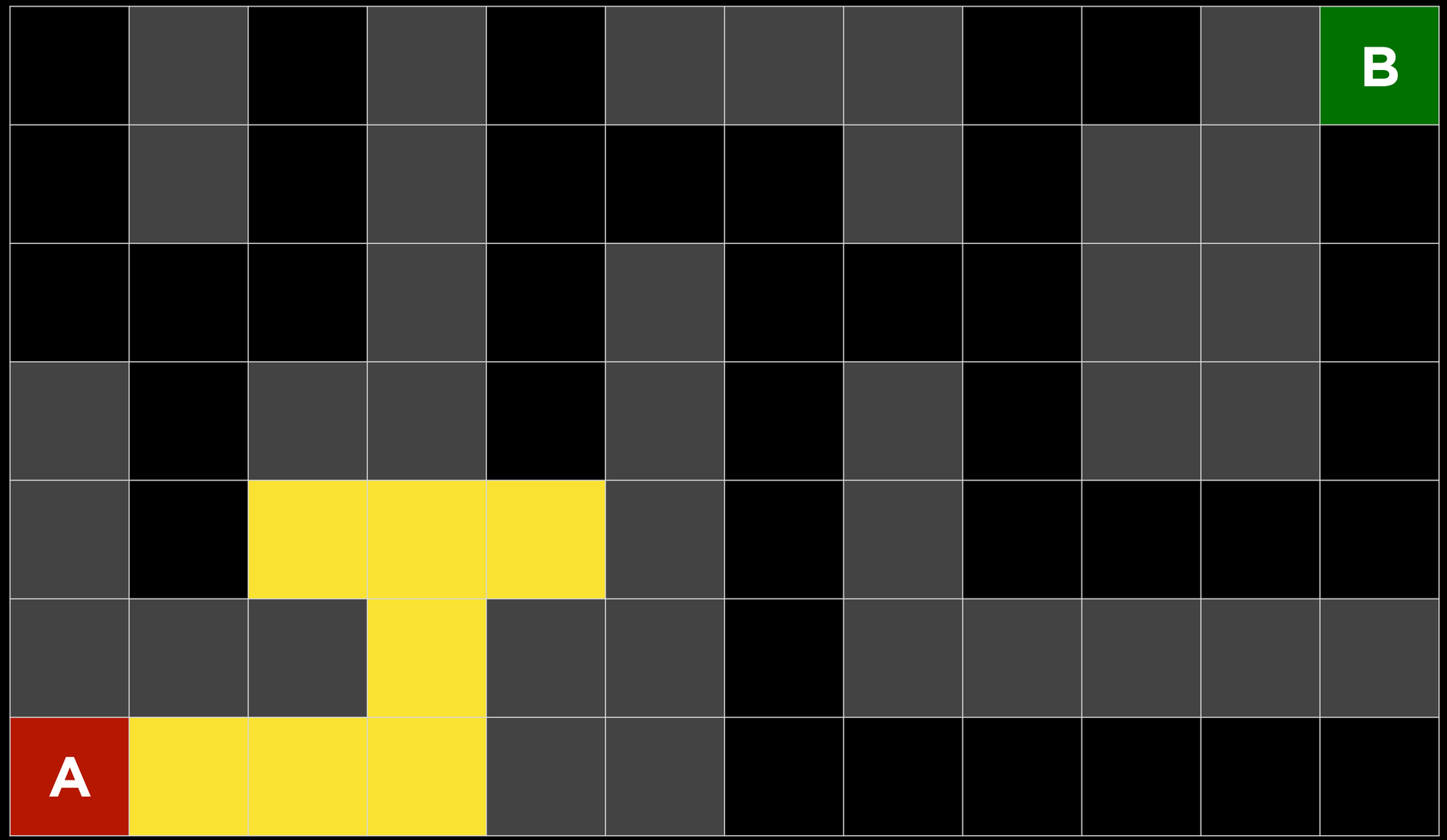
# Breadth-First Search



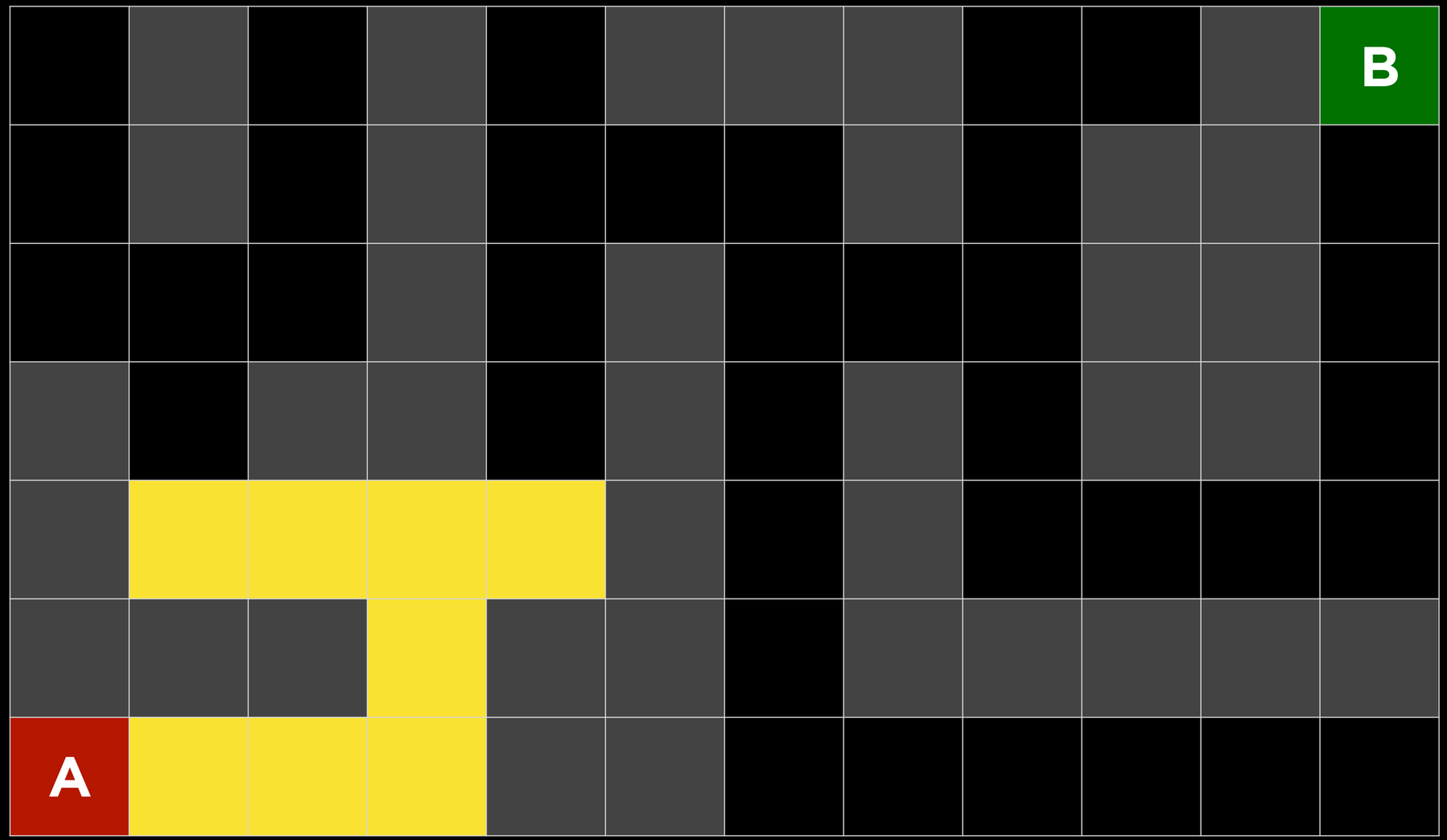
# Breadth-First Search



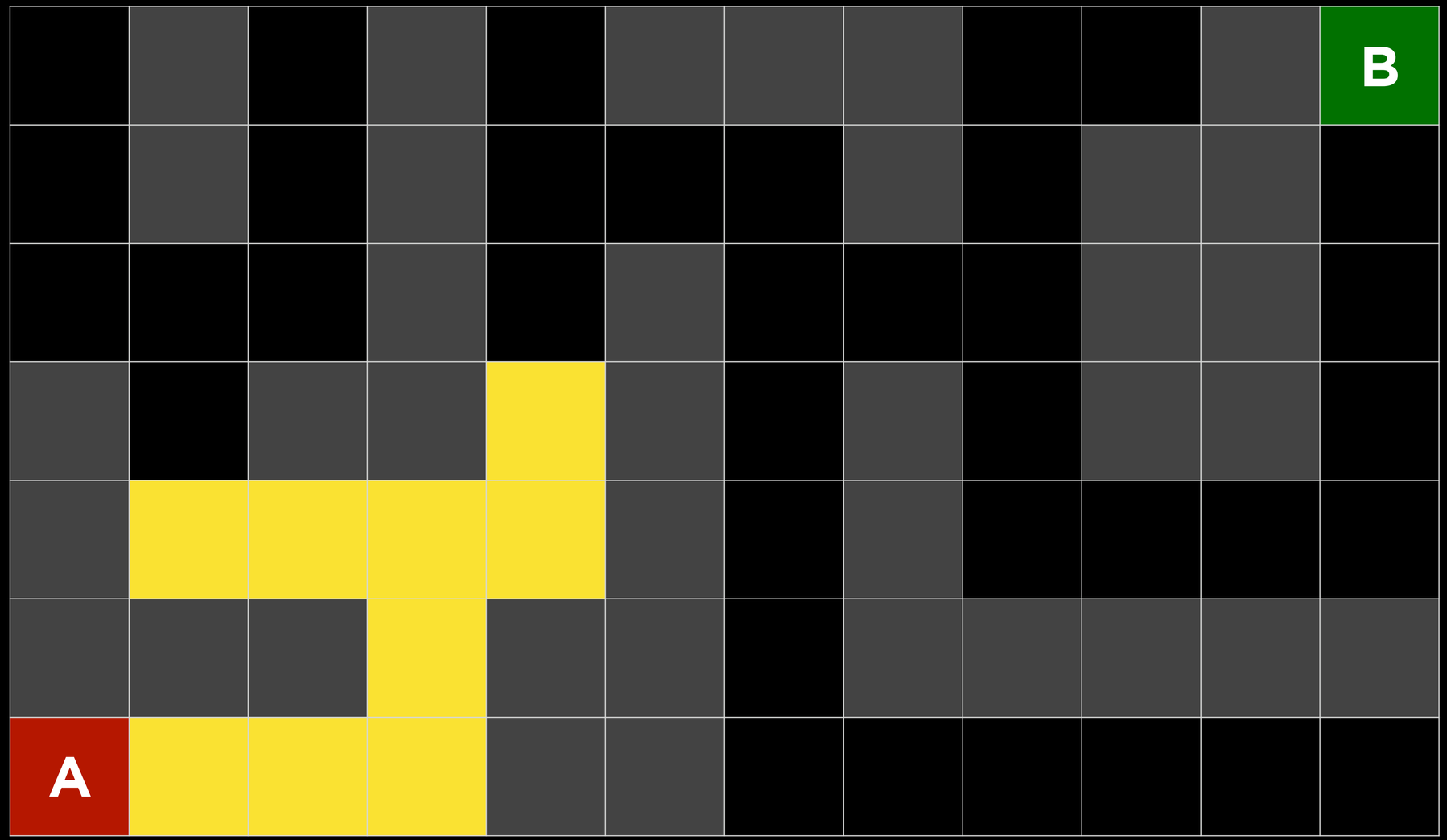
# Breadth-First Search



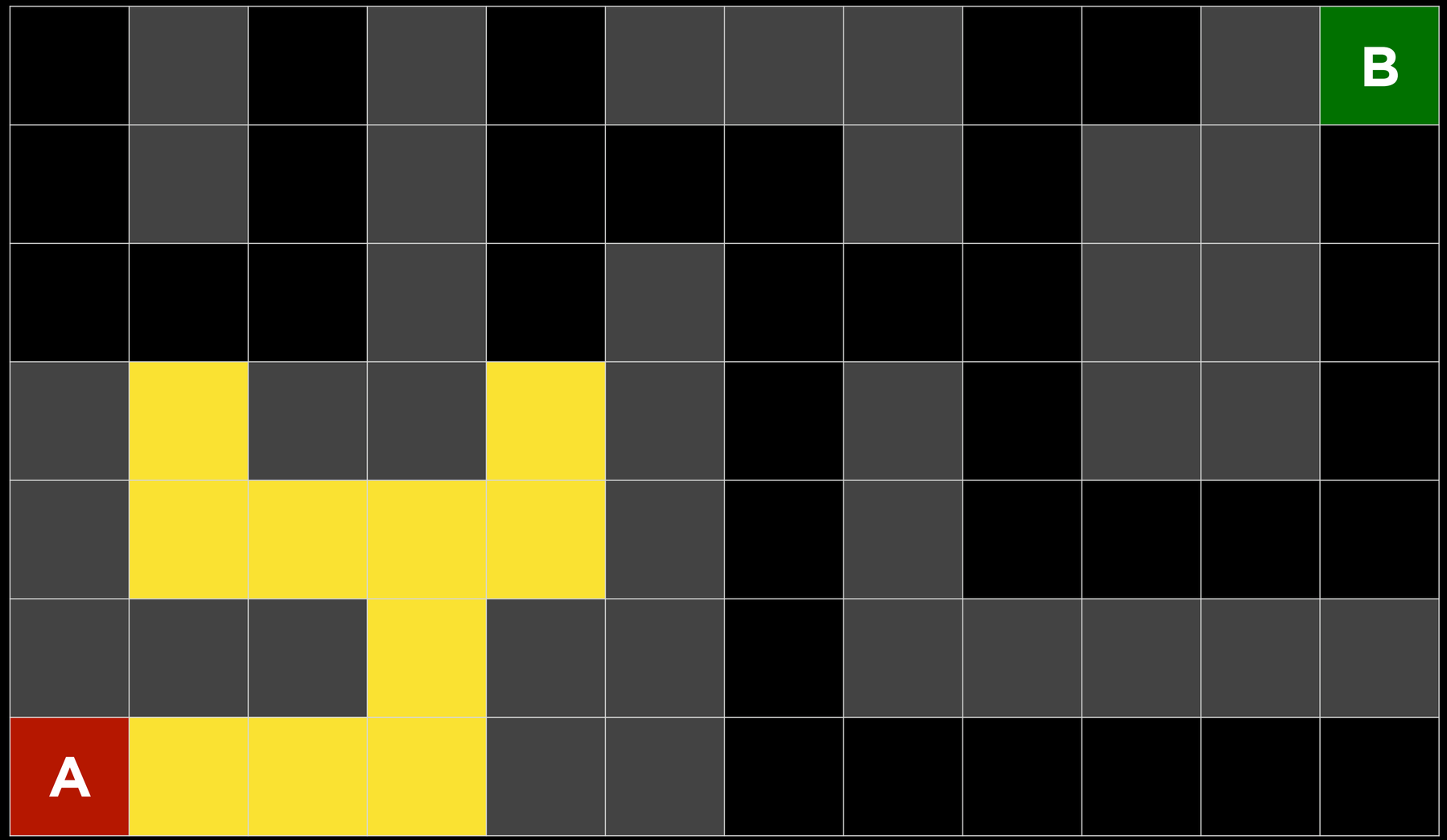
# Breadth-First Search



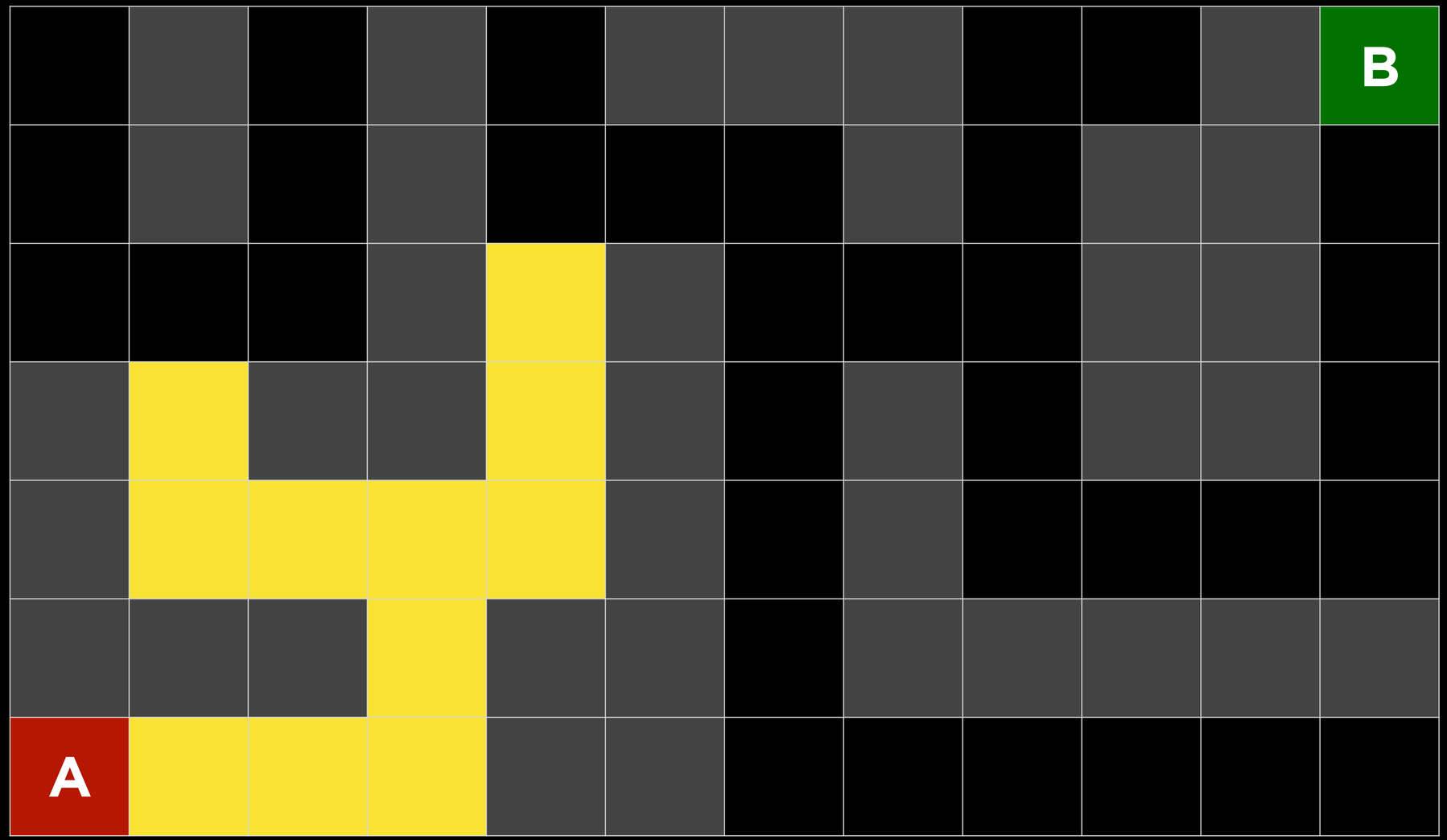
# Breadth-First Search



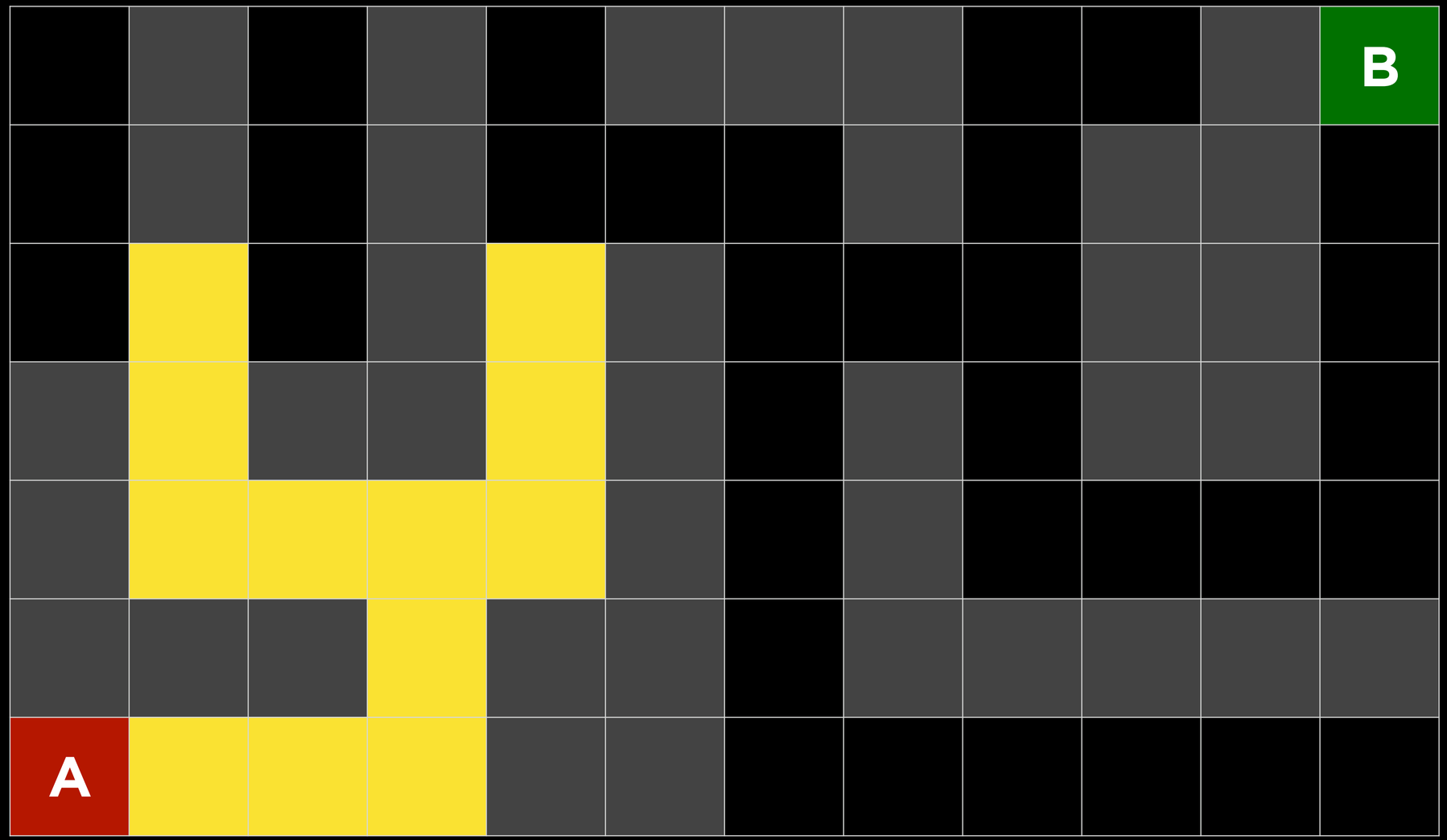
# Breadth-First Search



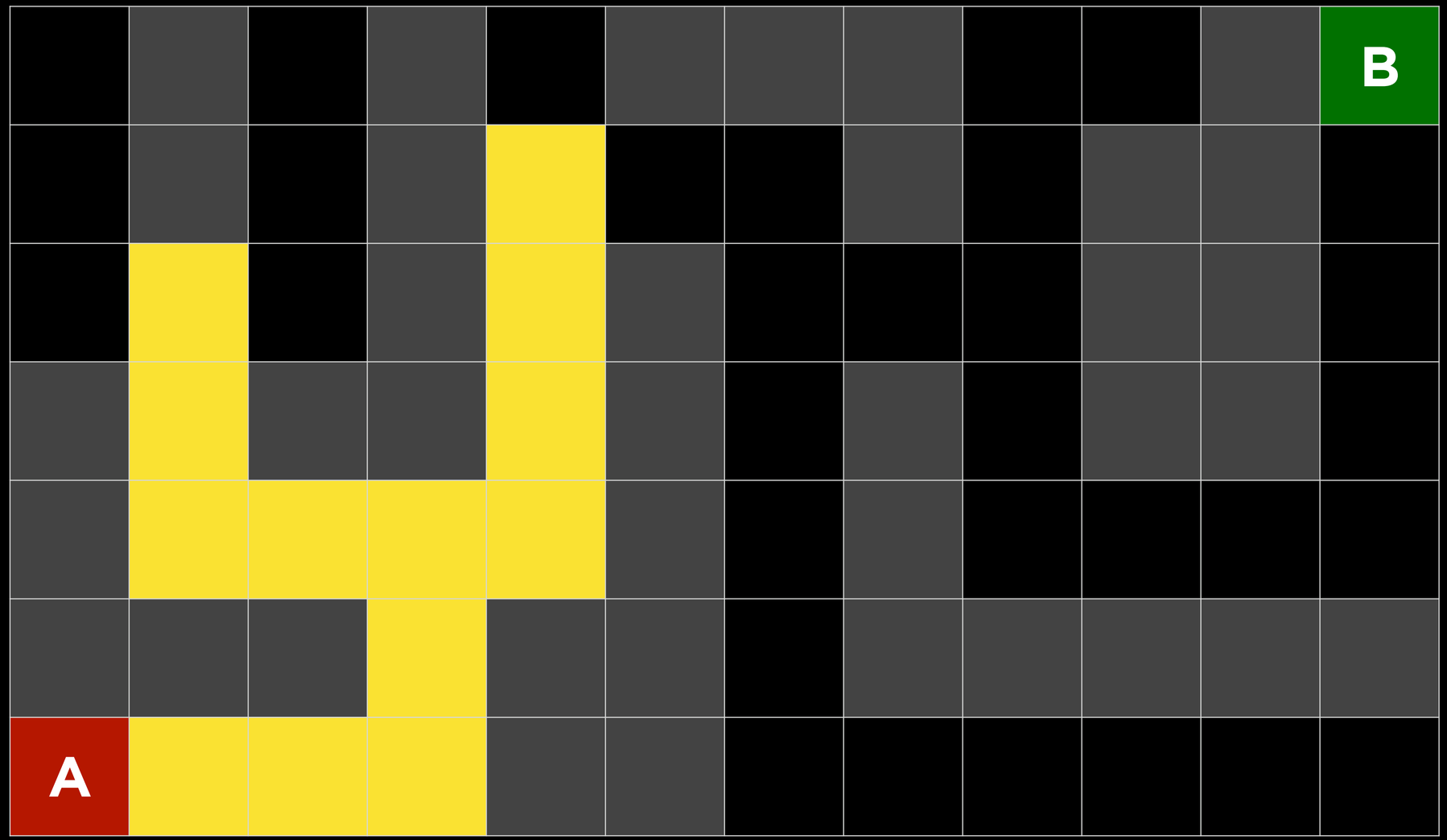
# Breadth-First Search



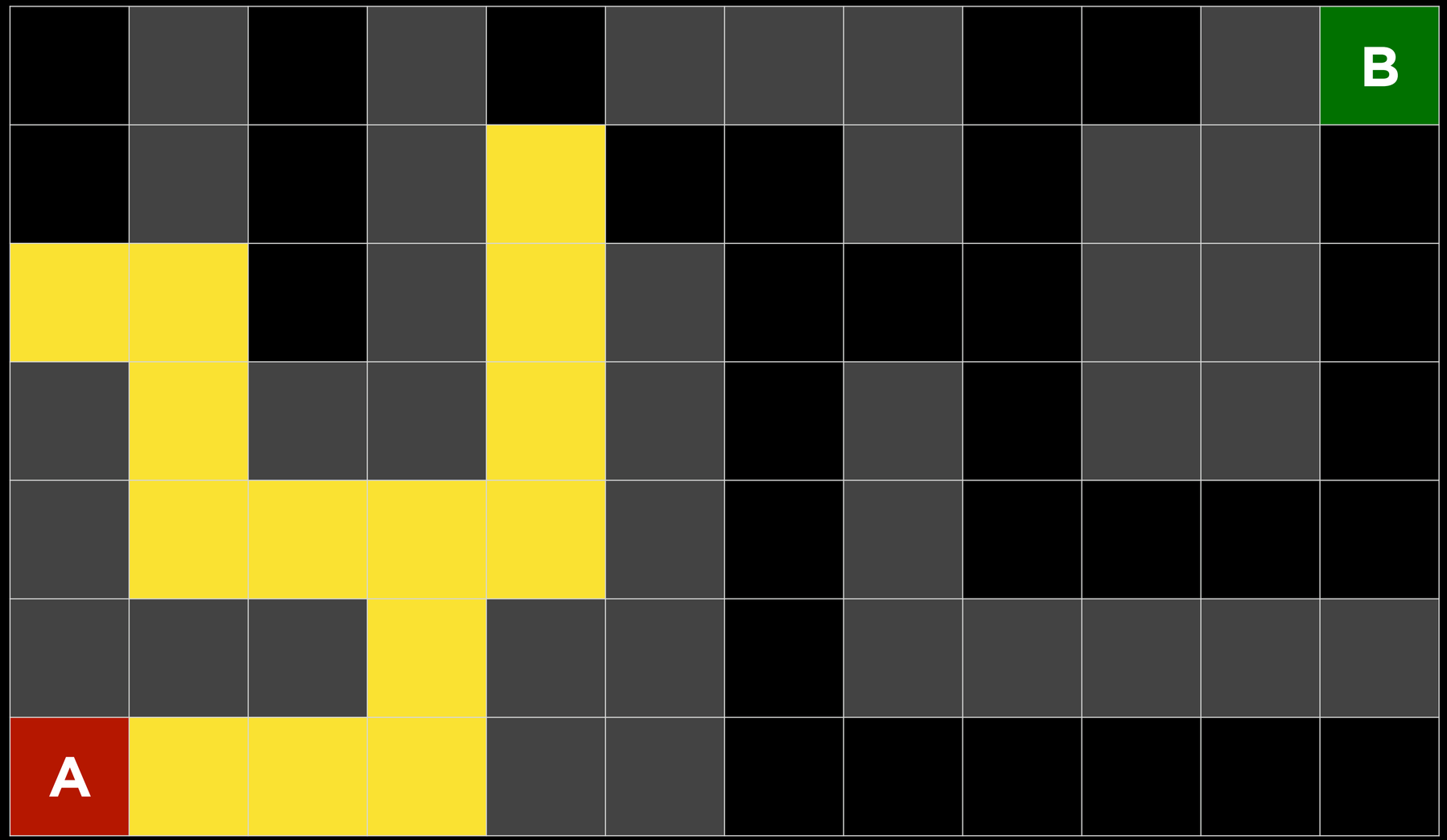
# Breadth-First Search



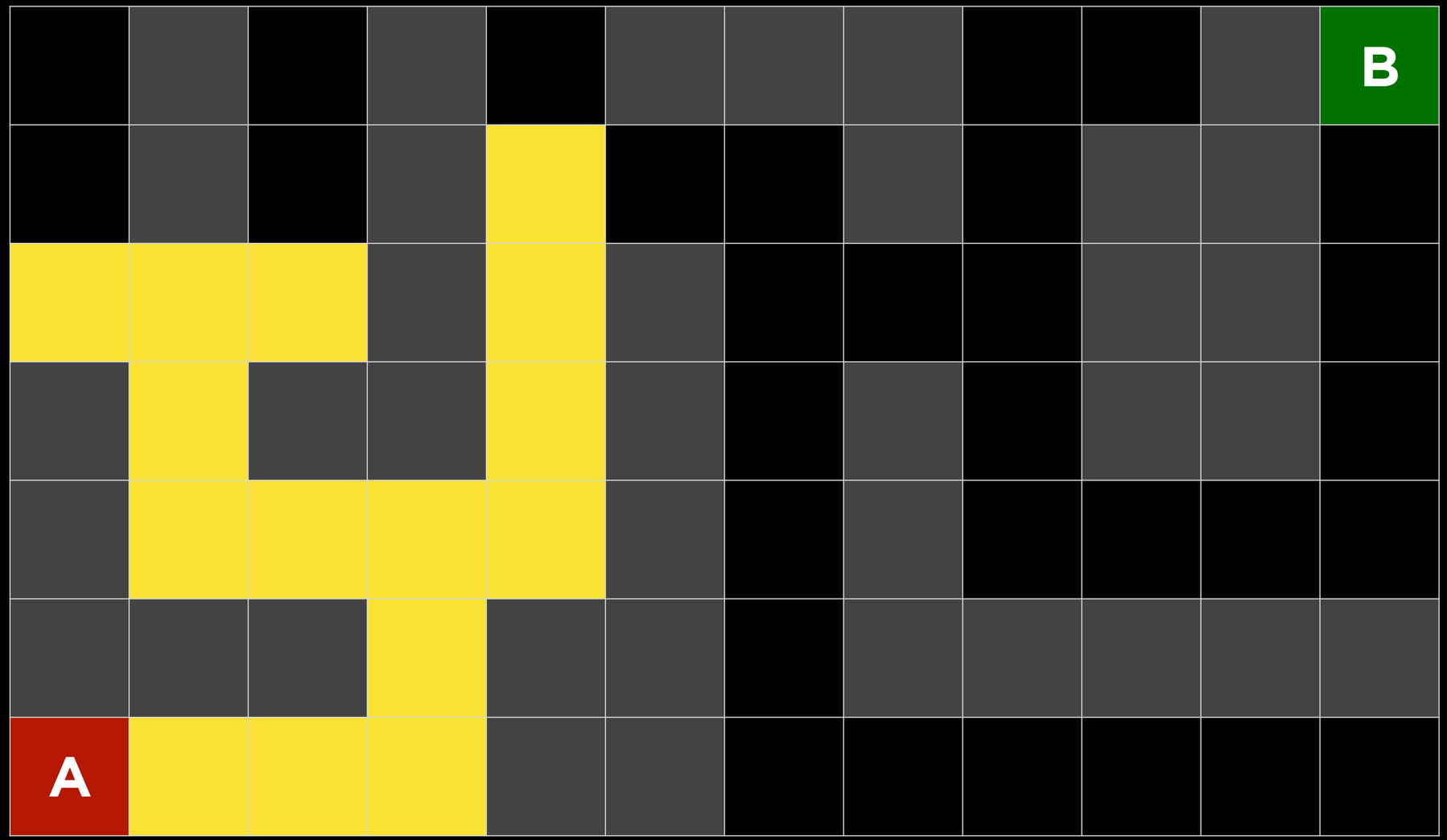
# Breadth-First Search



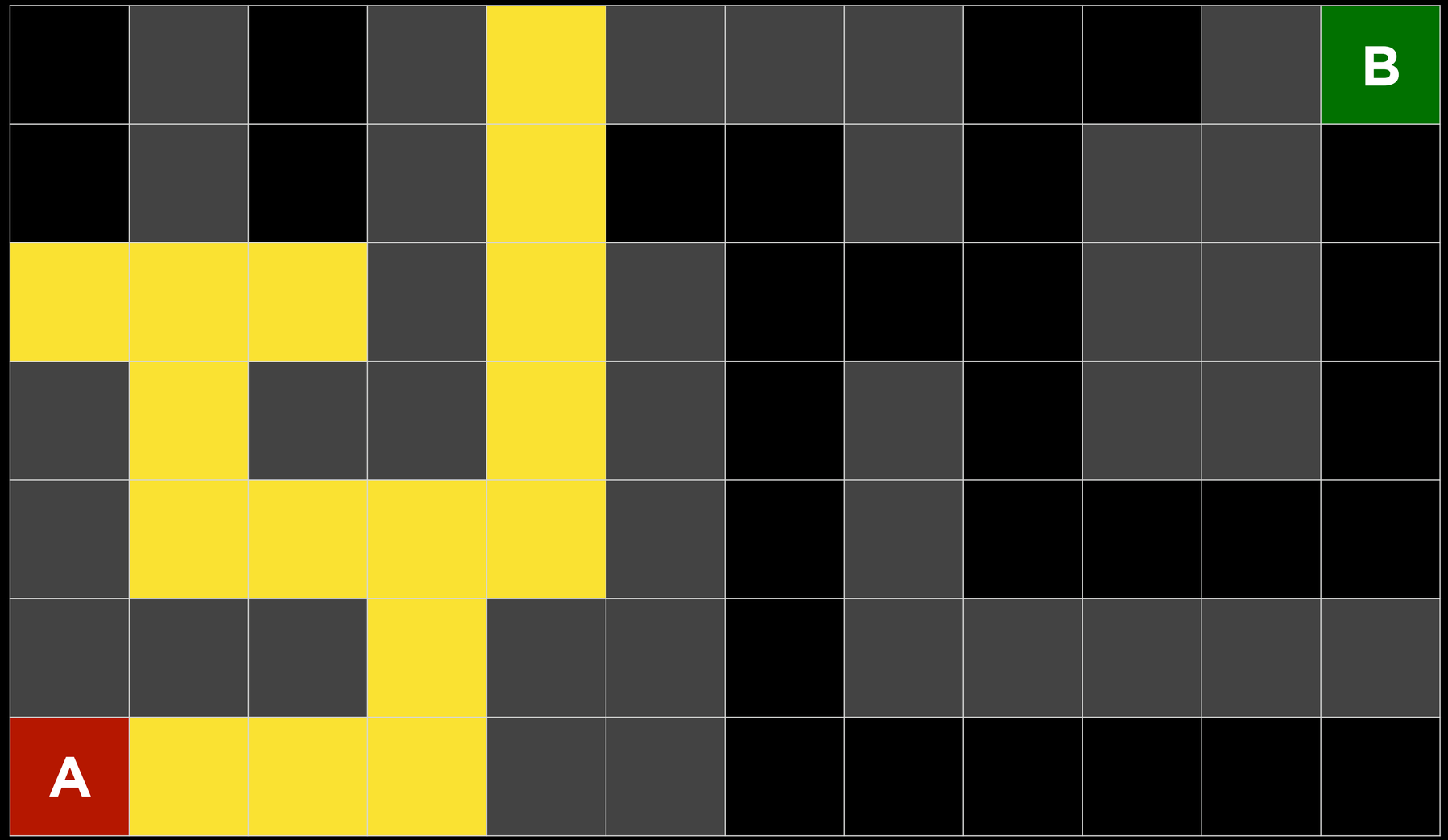
# Breadth-First Search



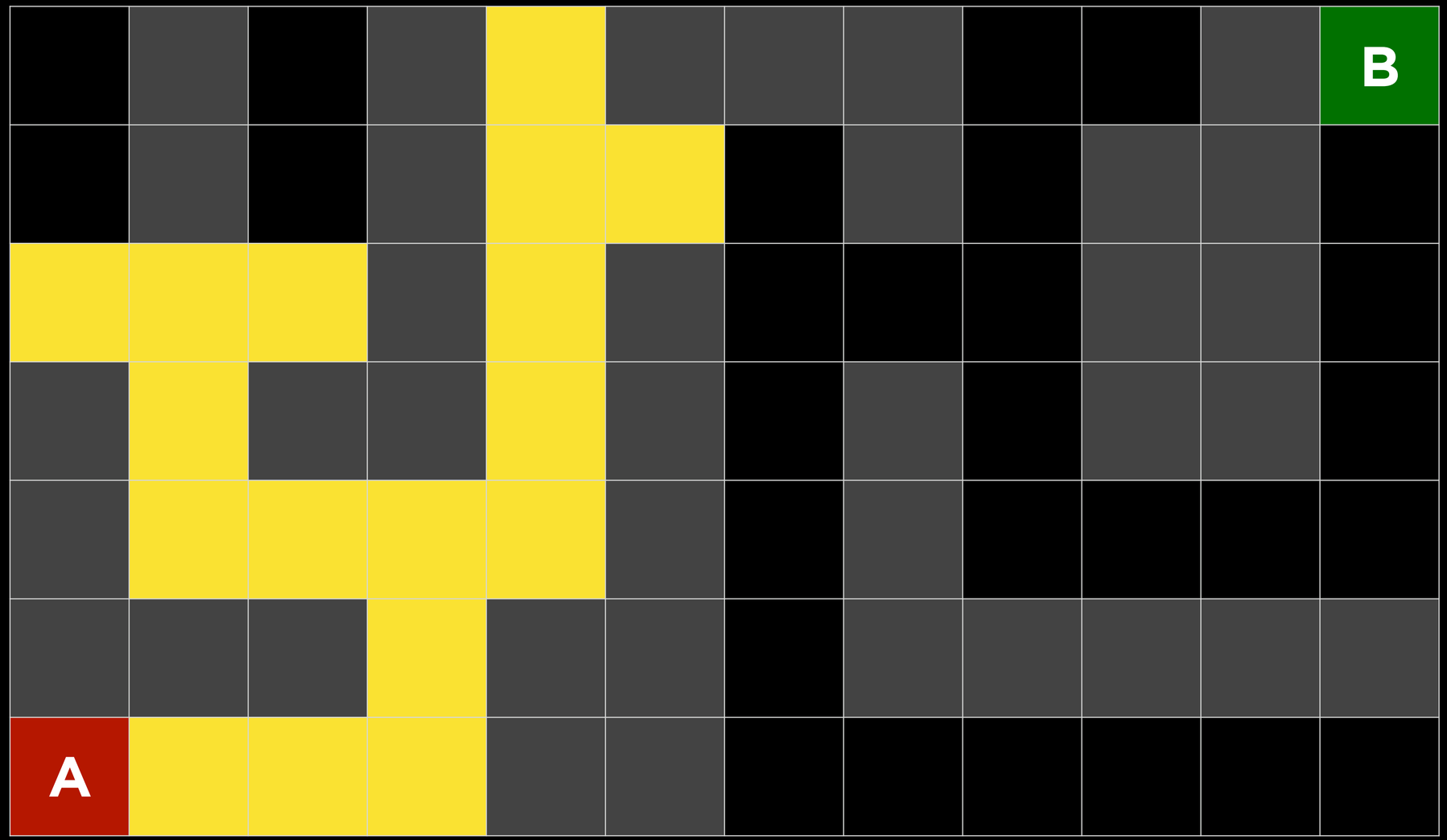
# Breadth-First Search



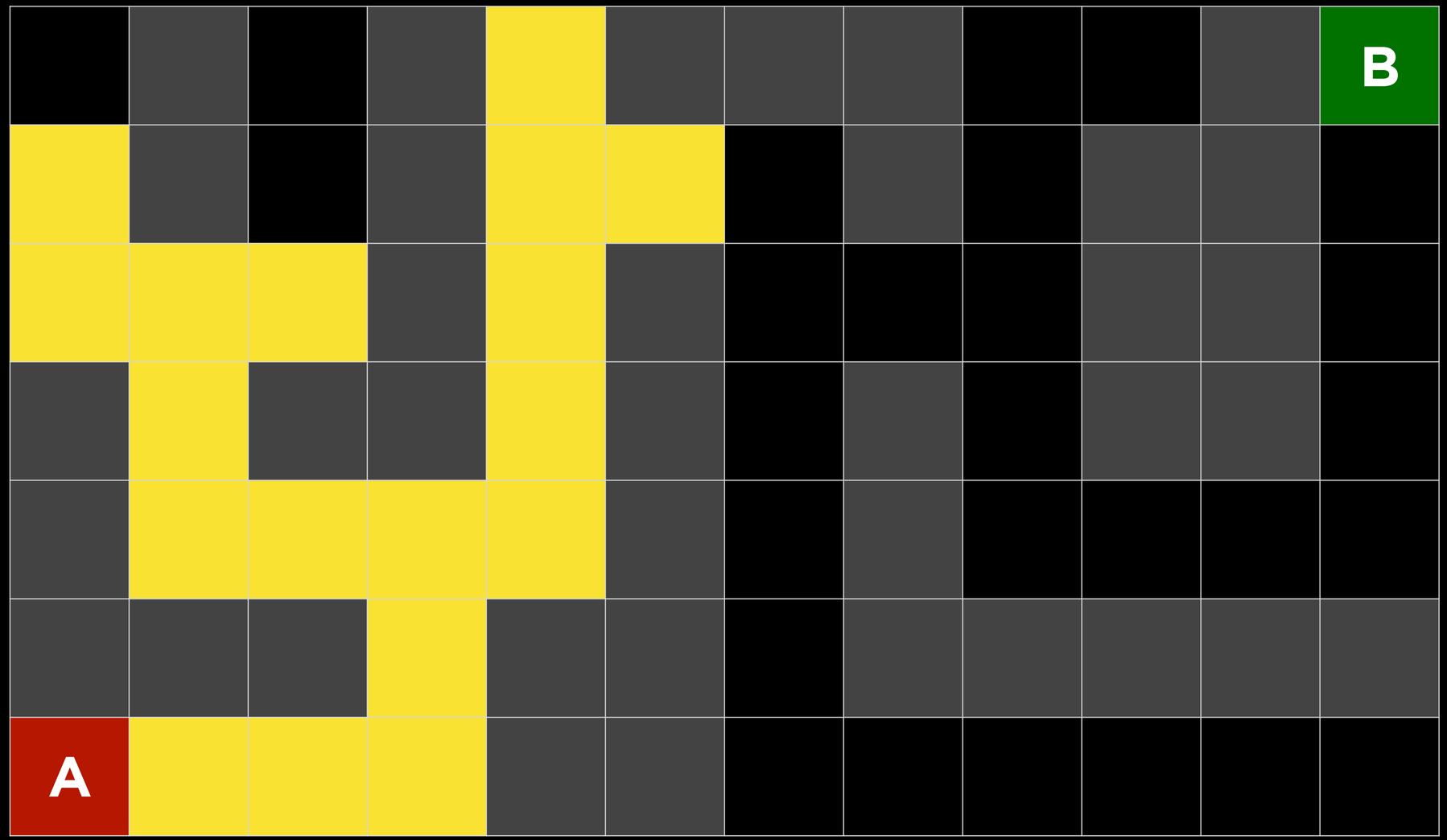
# Breadth-First Search



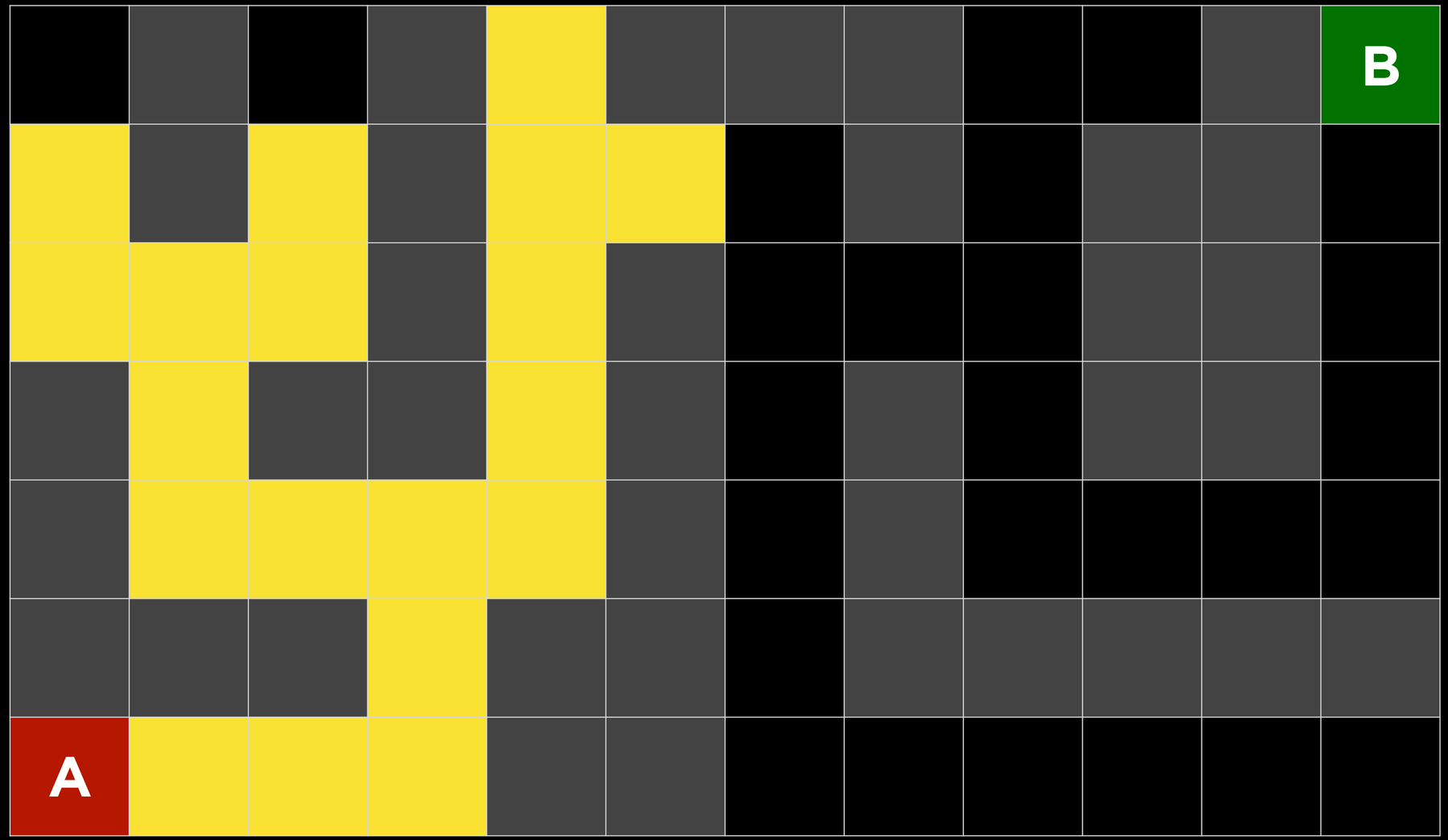
# Breadth-First Search



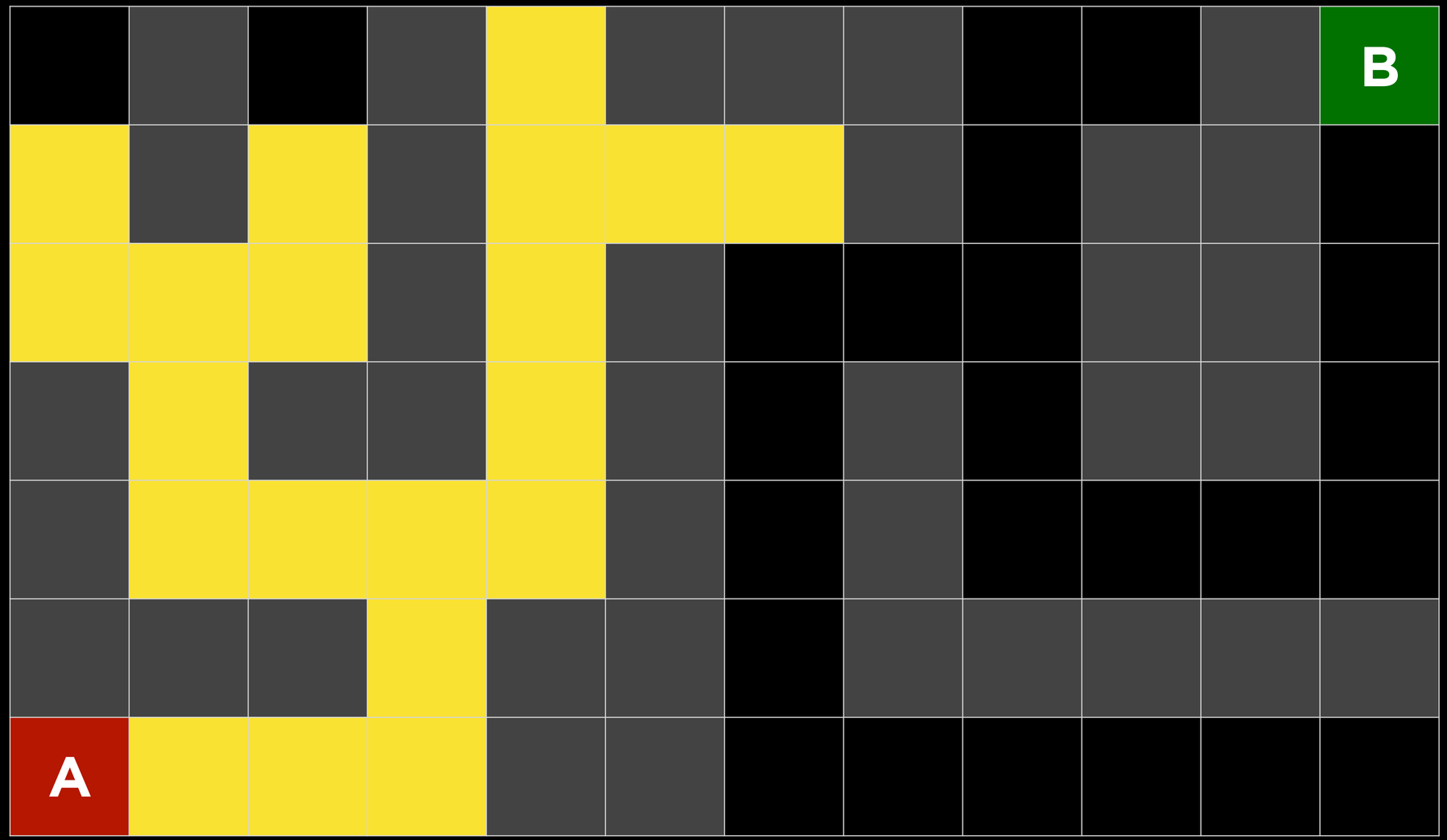
# Breadth-First Search



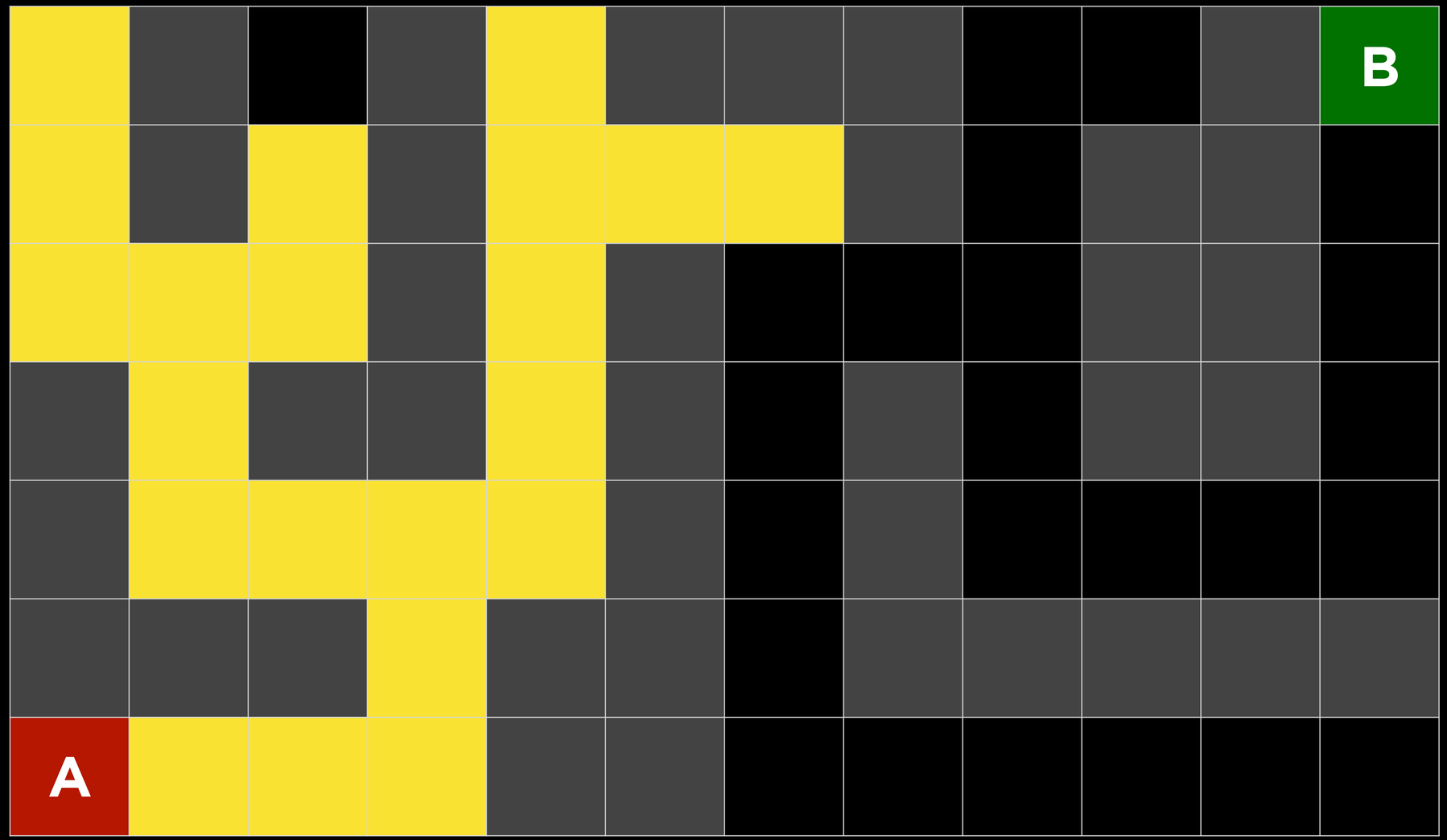
# Breadth-First Search



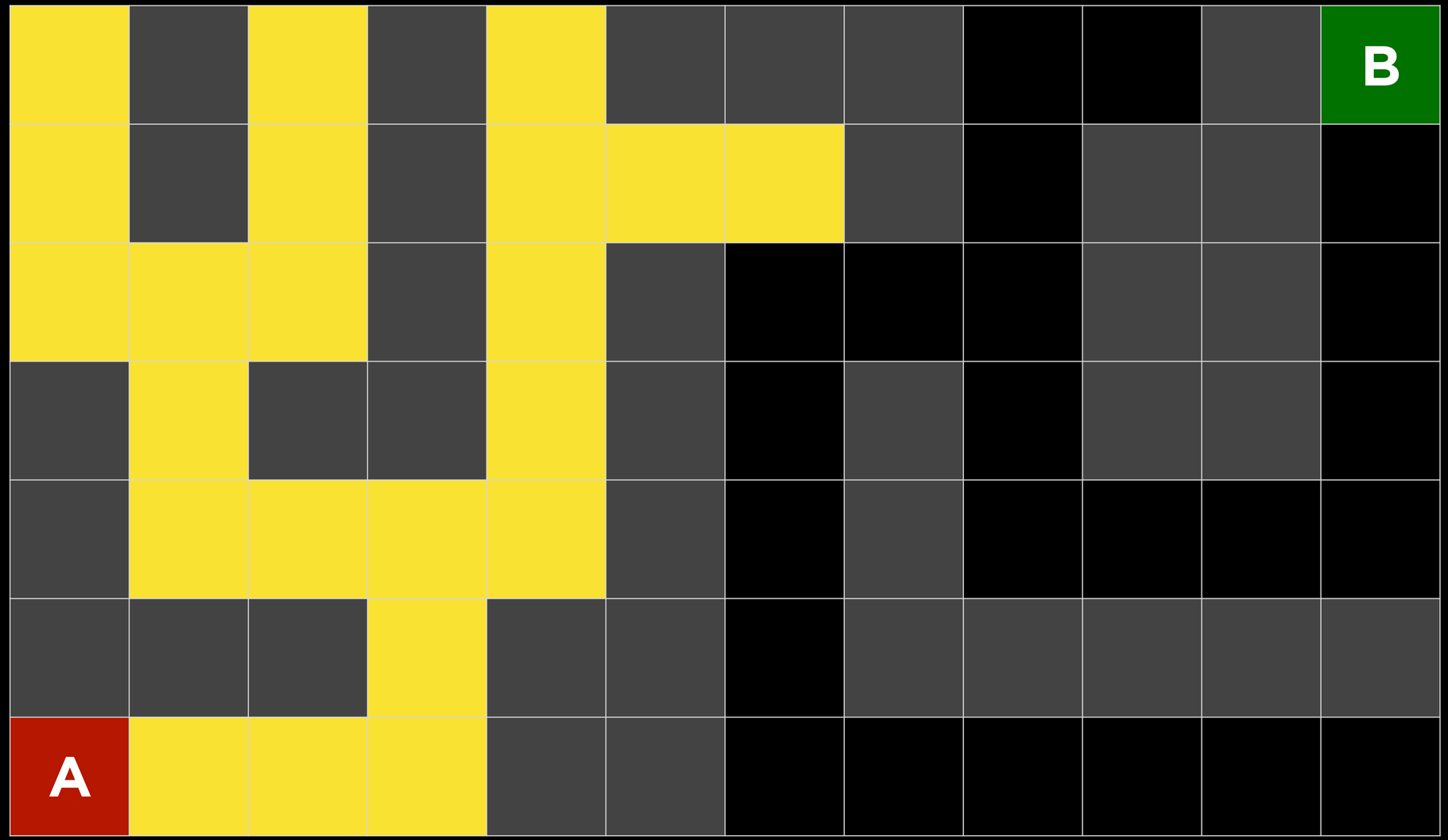
# Breadth-First Search



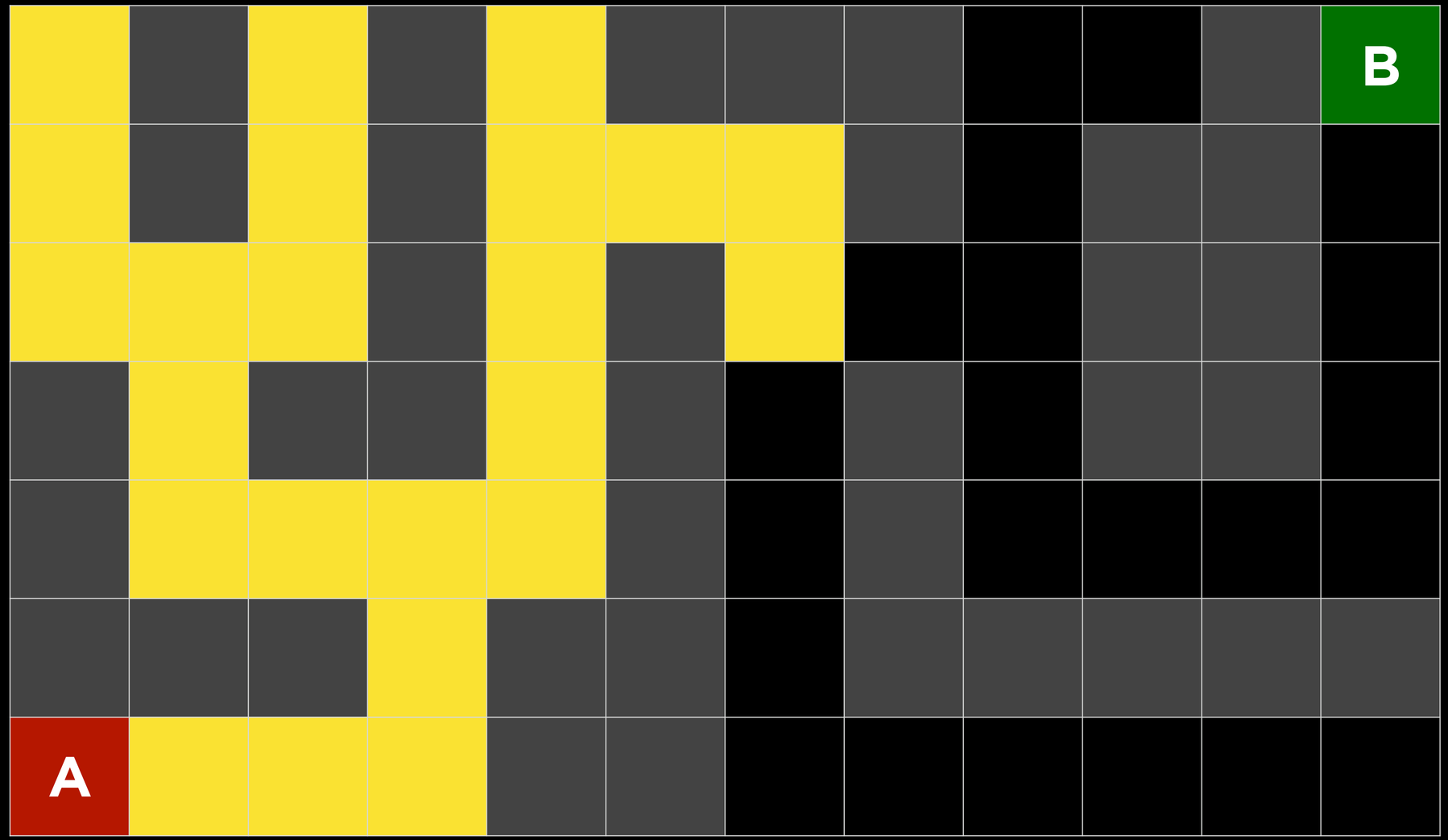
# Breadth-First Search



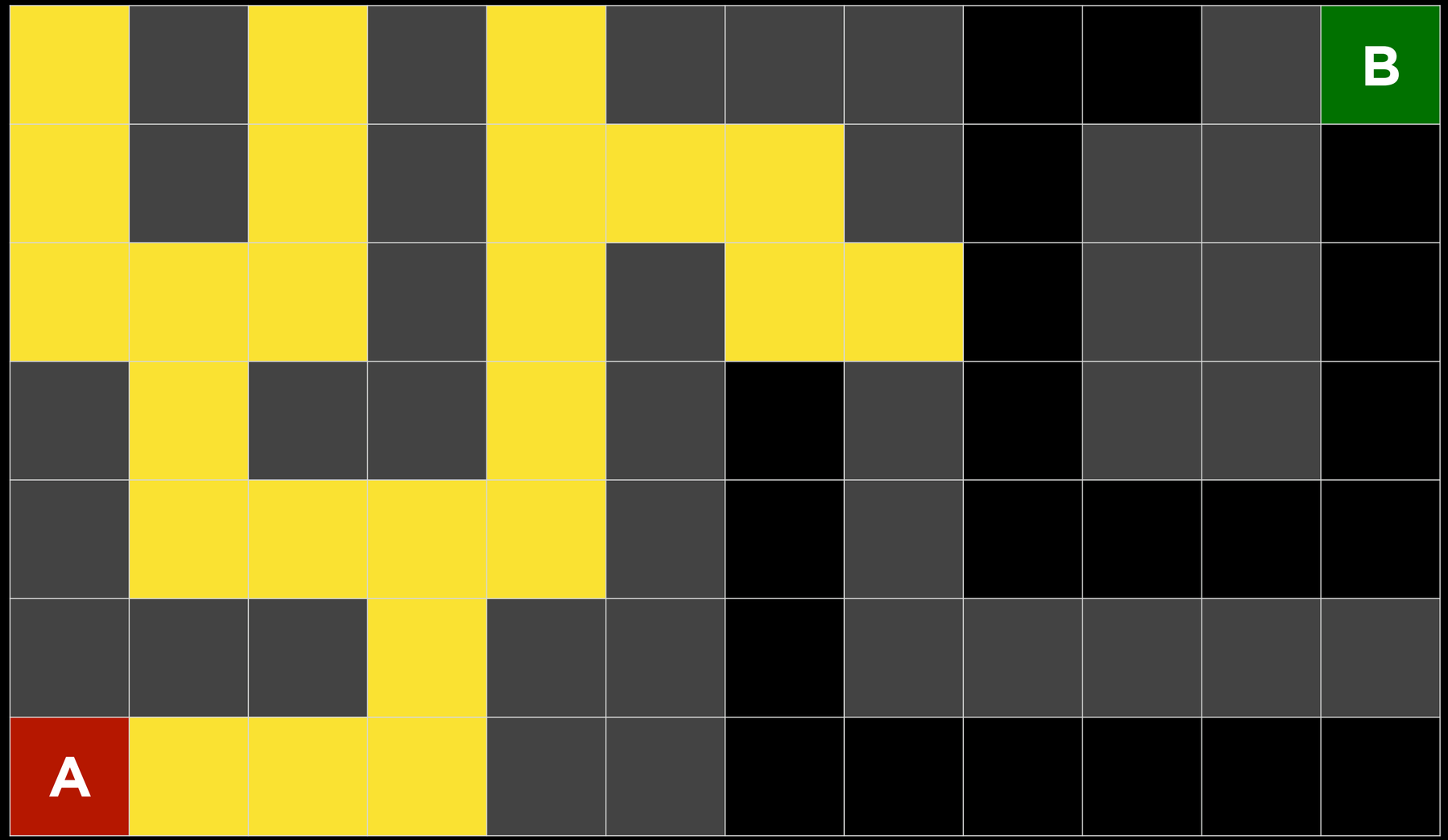
# Breadth-First Search



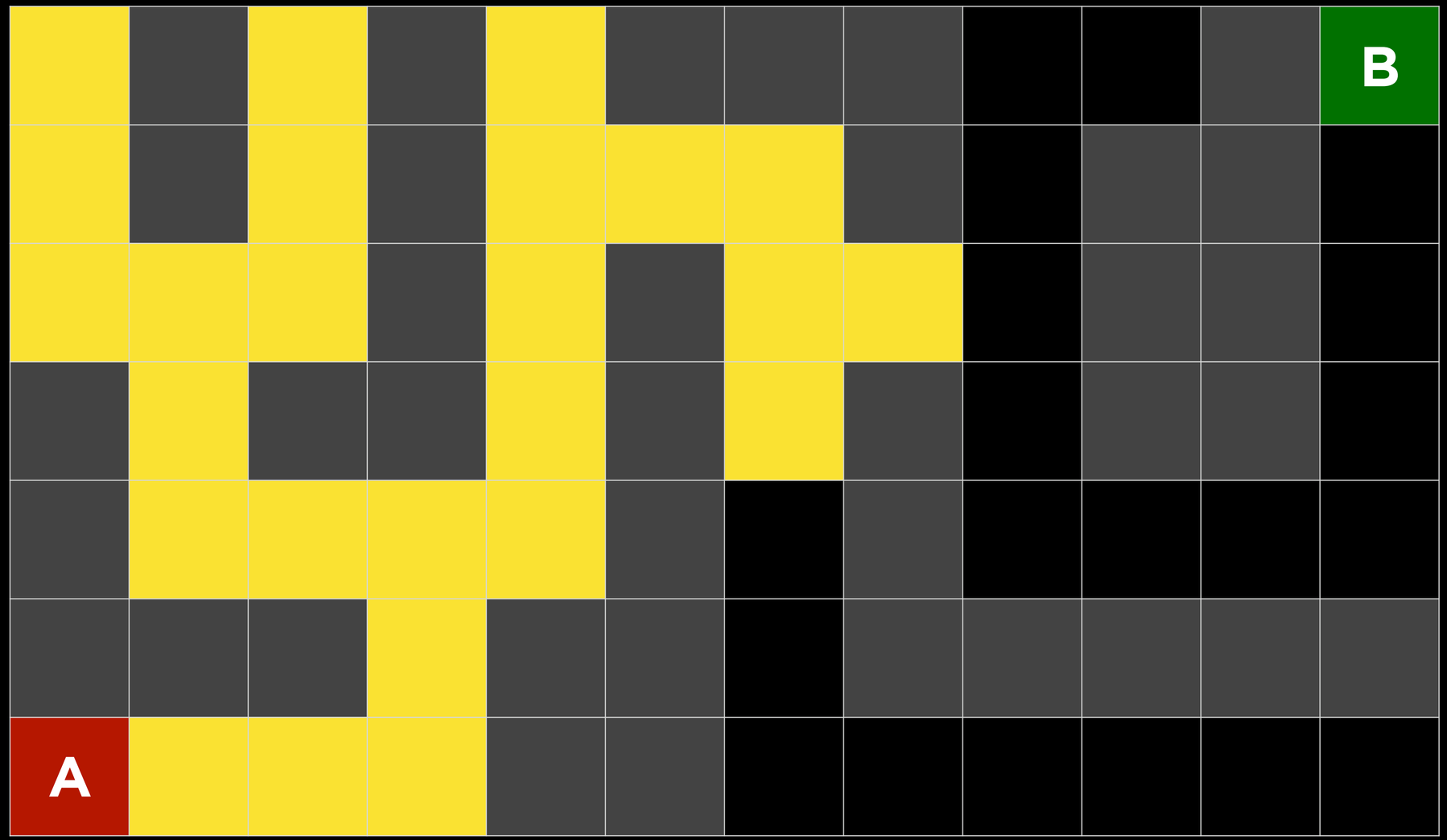
# Breadth-First Search



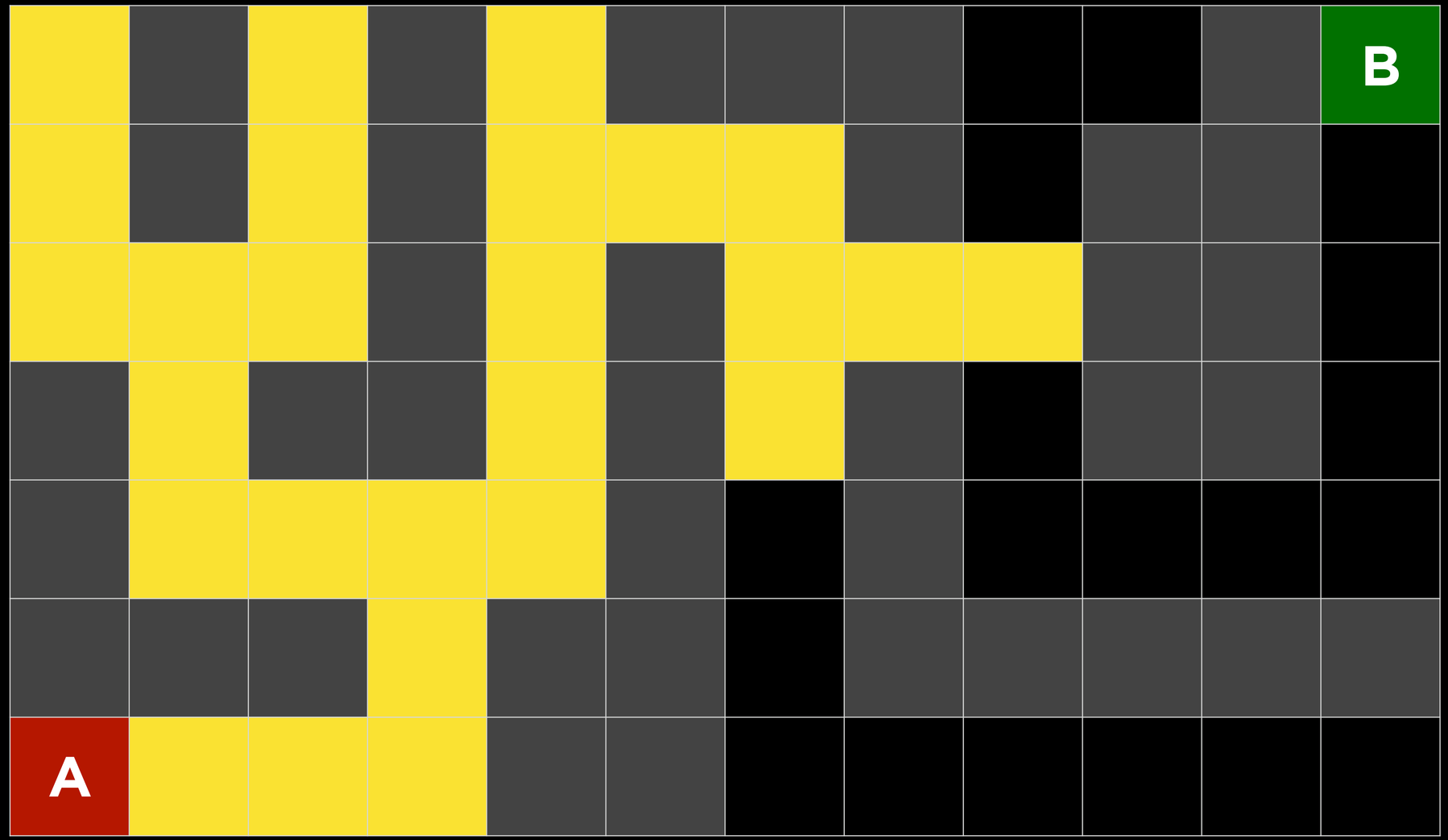
# Breadth-First Search



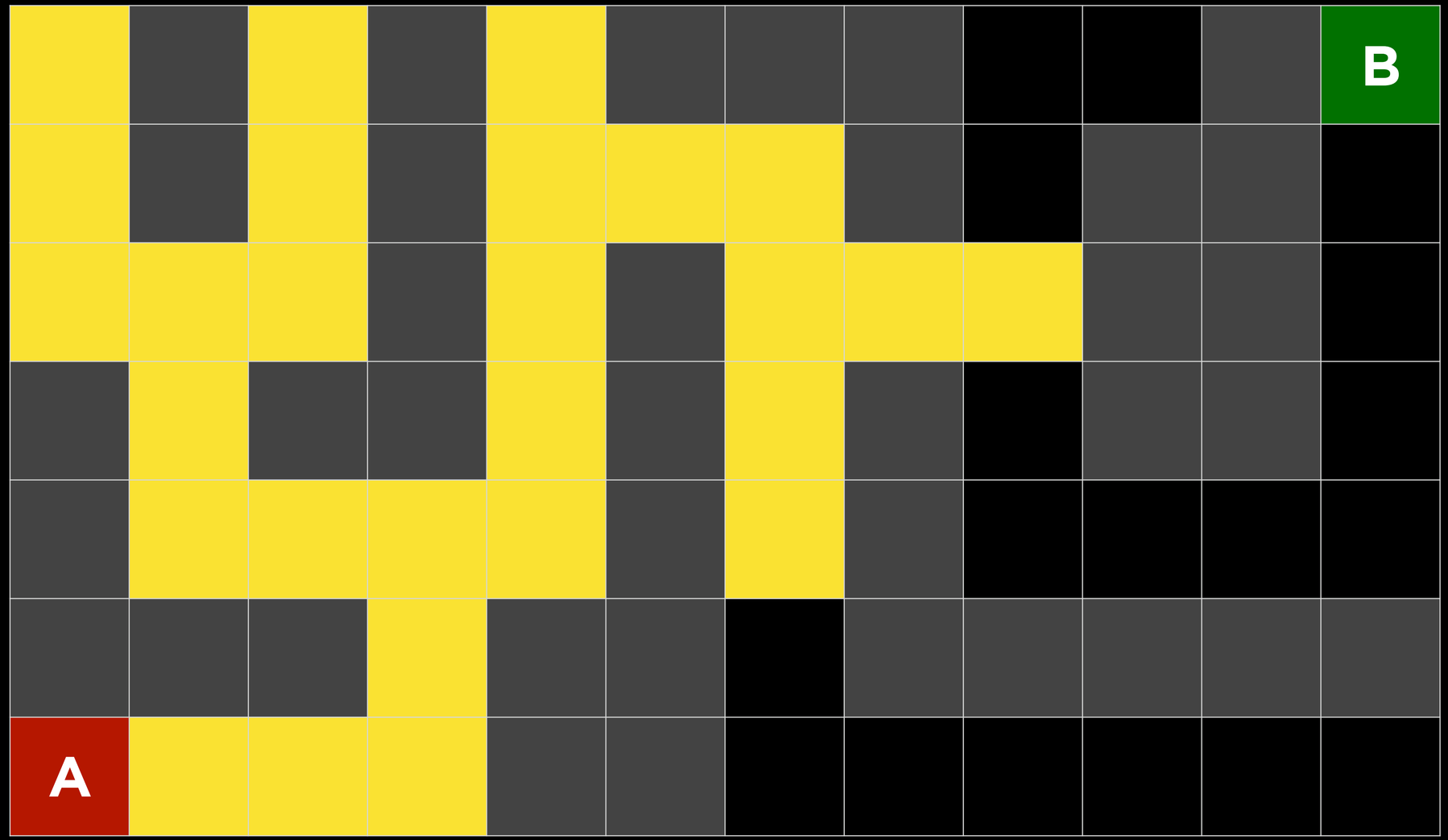
# Breadth-First Search



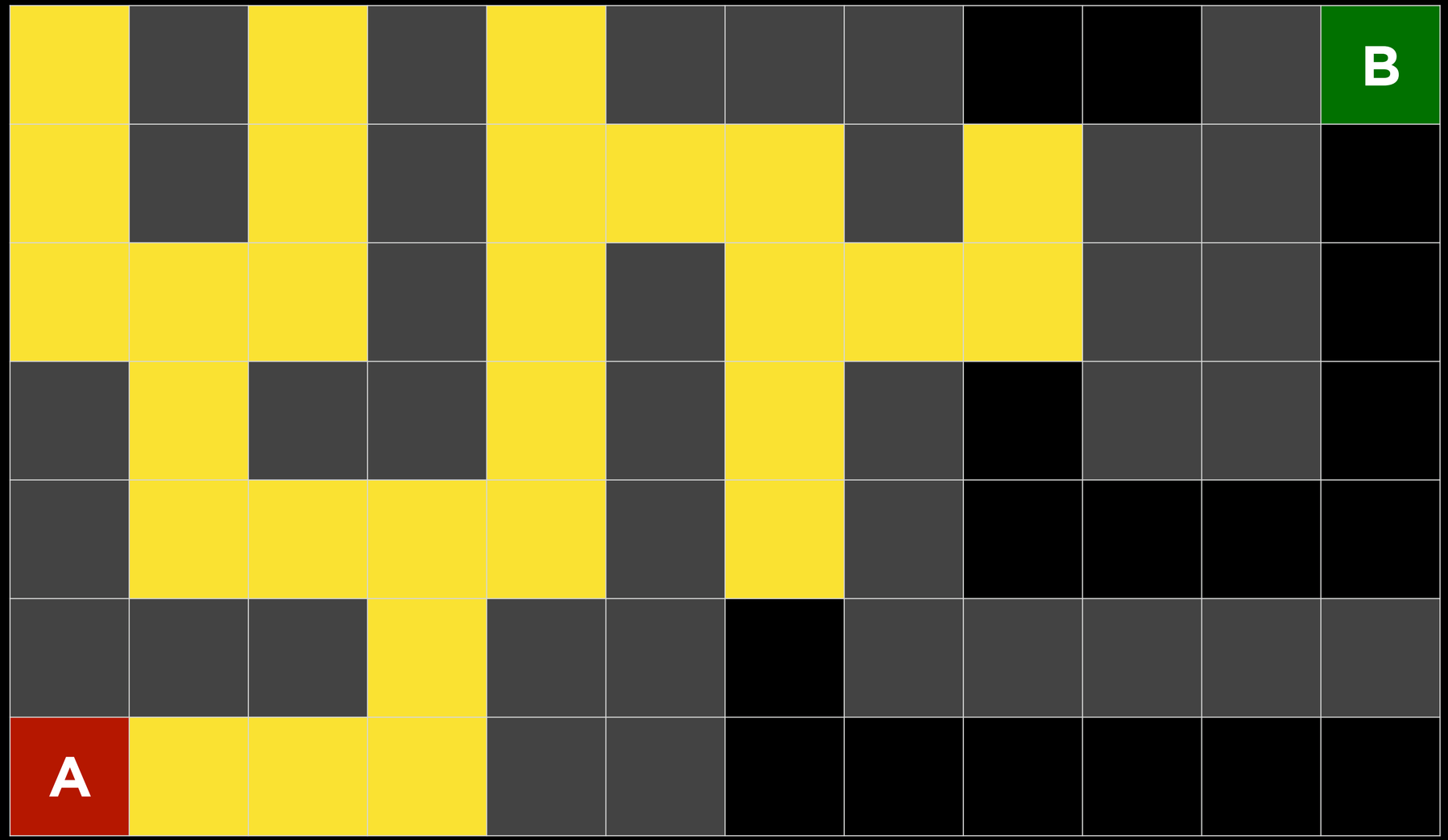
# Breadth-First Search



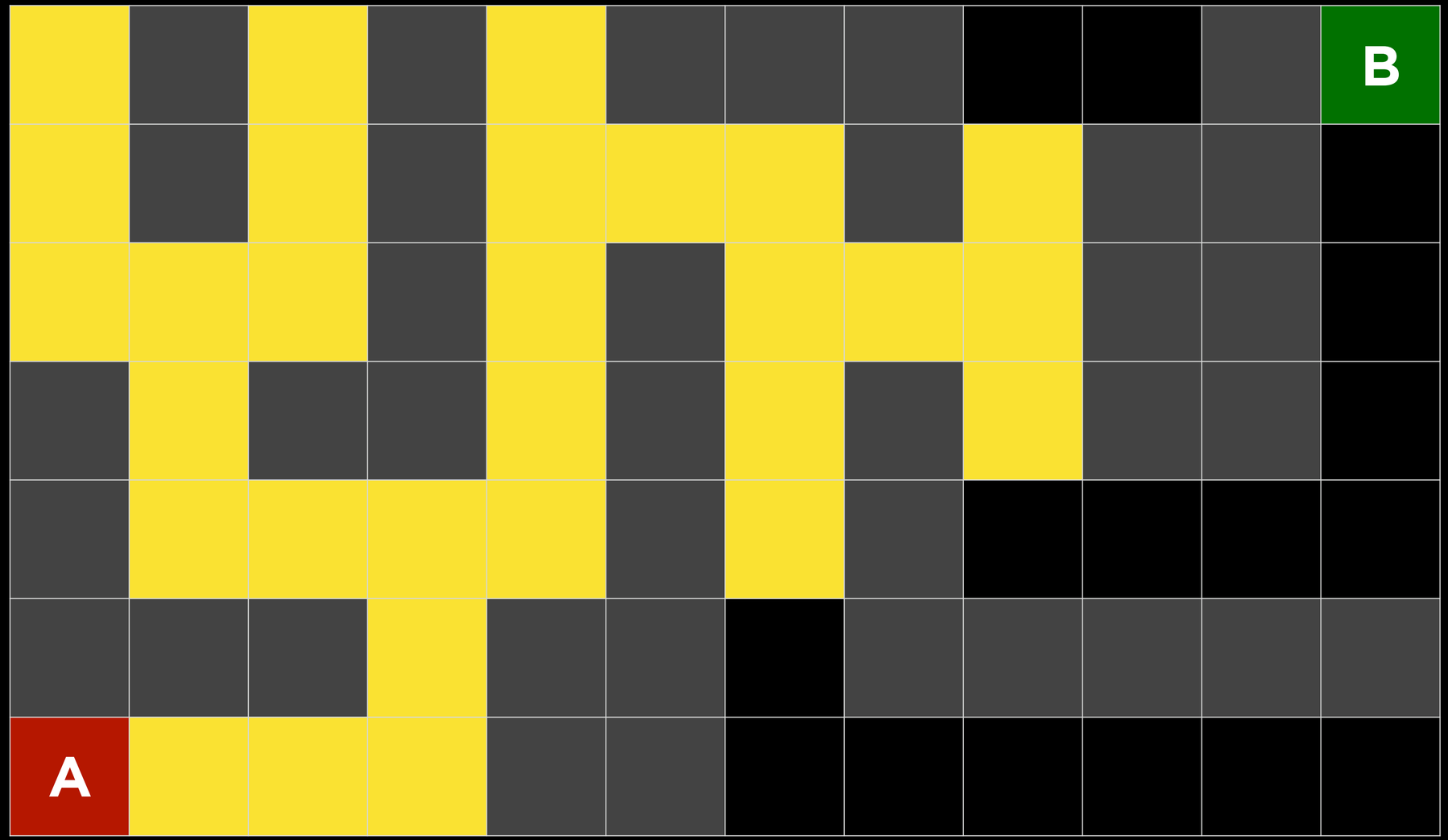
# Breadth-First Search



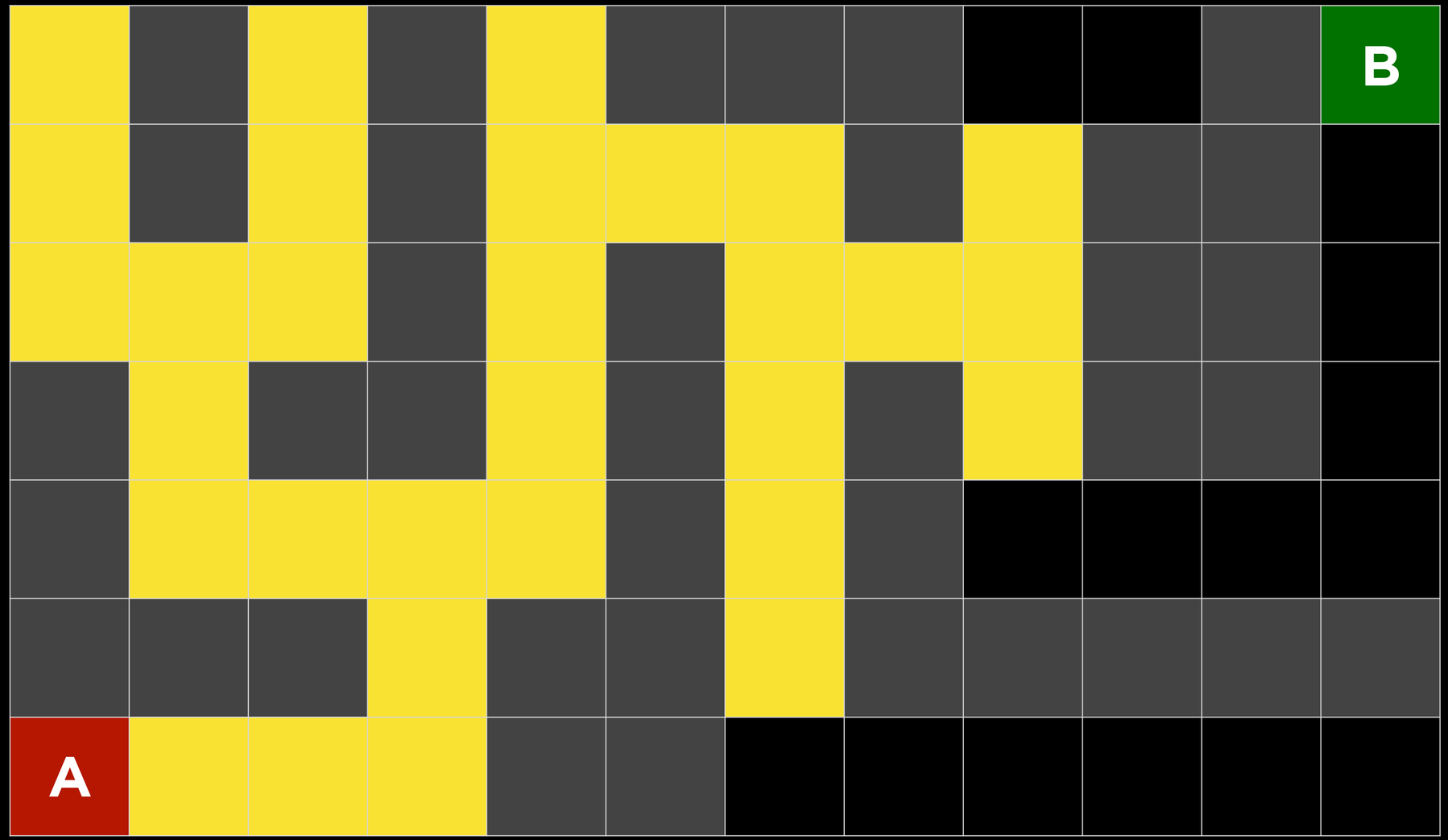
# Breadth-First Search



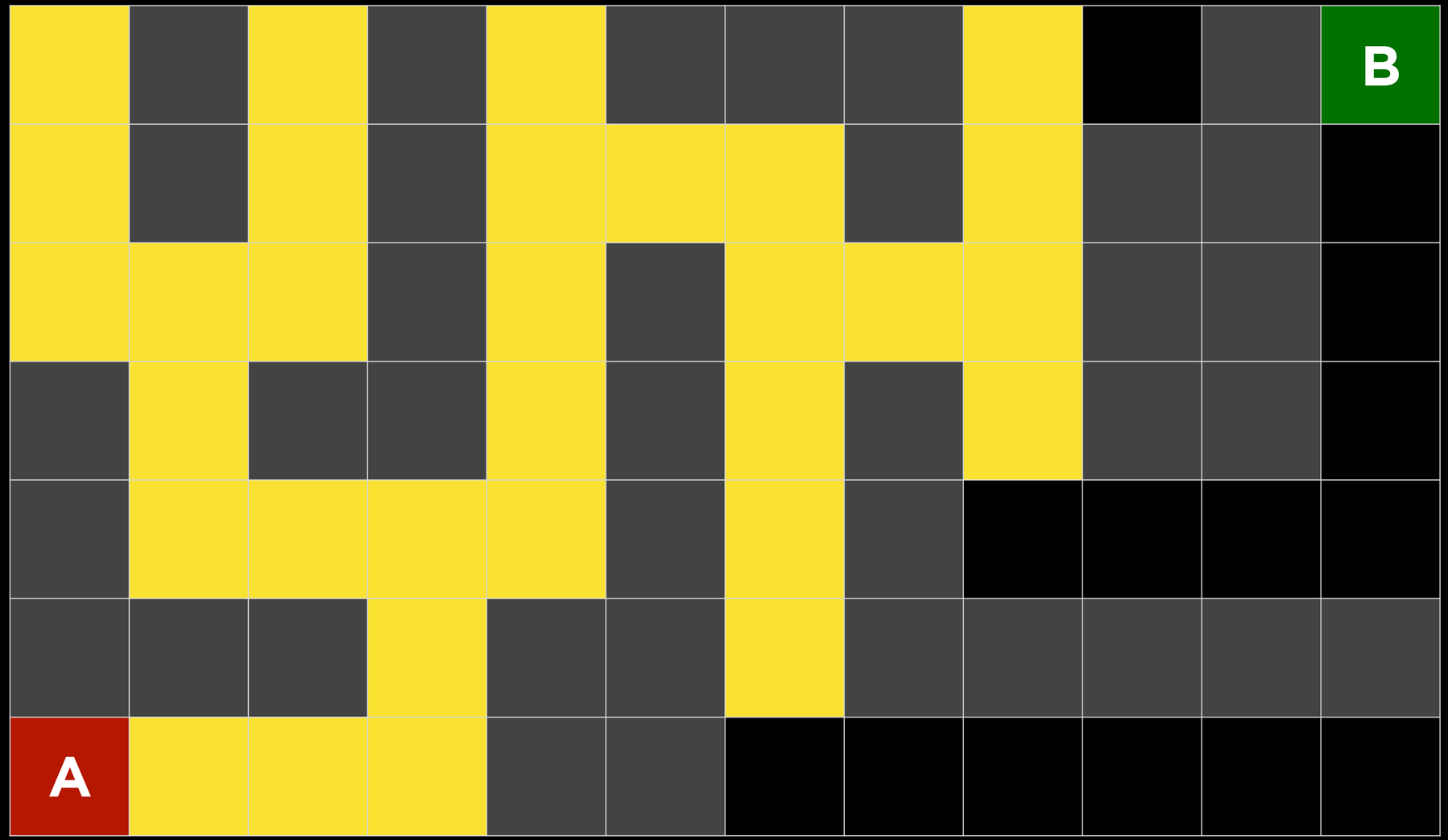
# Breadth-First Search



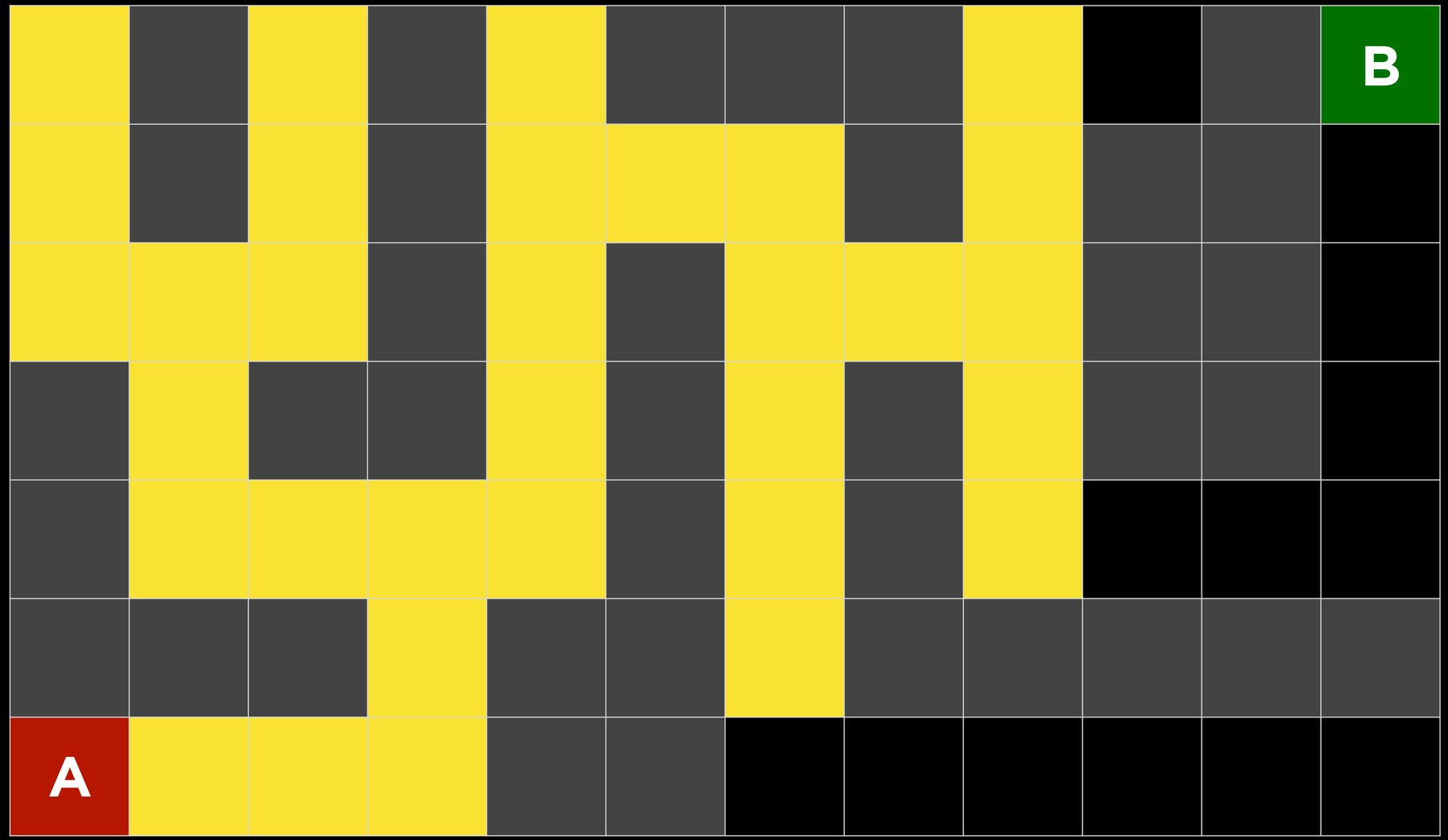
# Breadth-First Search



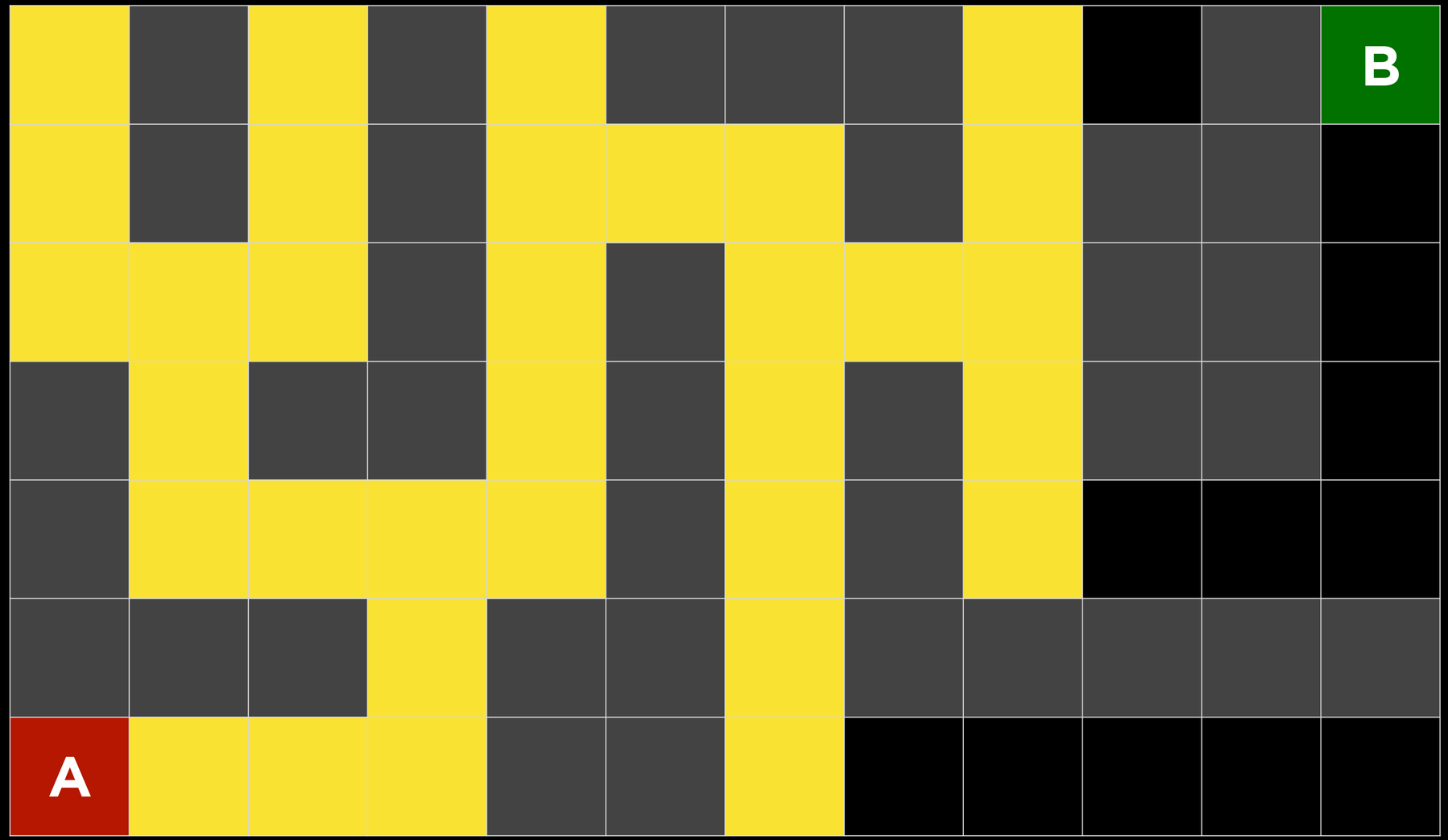
# Breadth-First Search



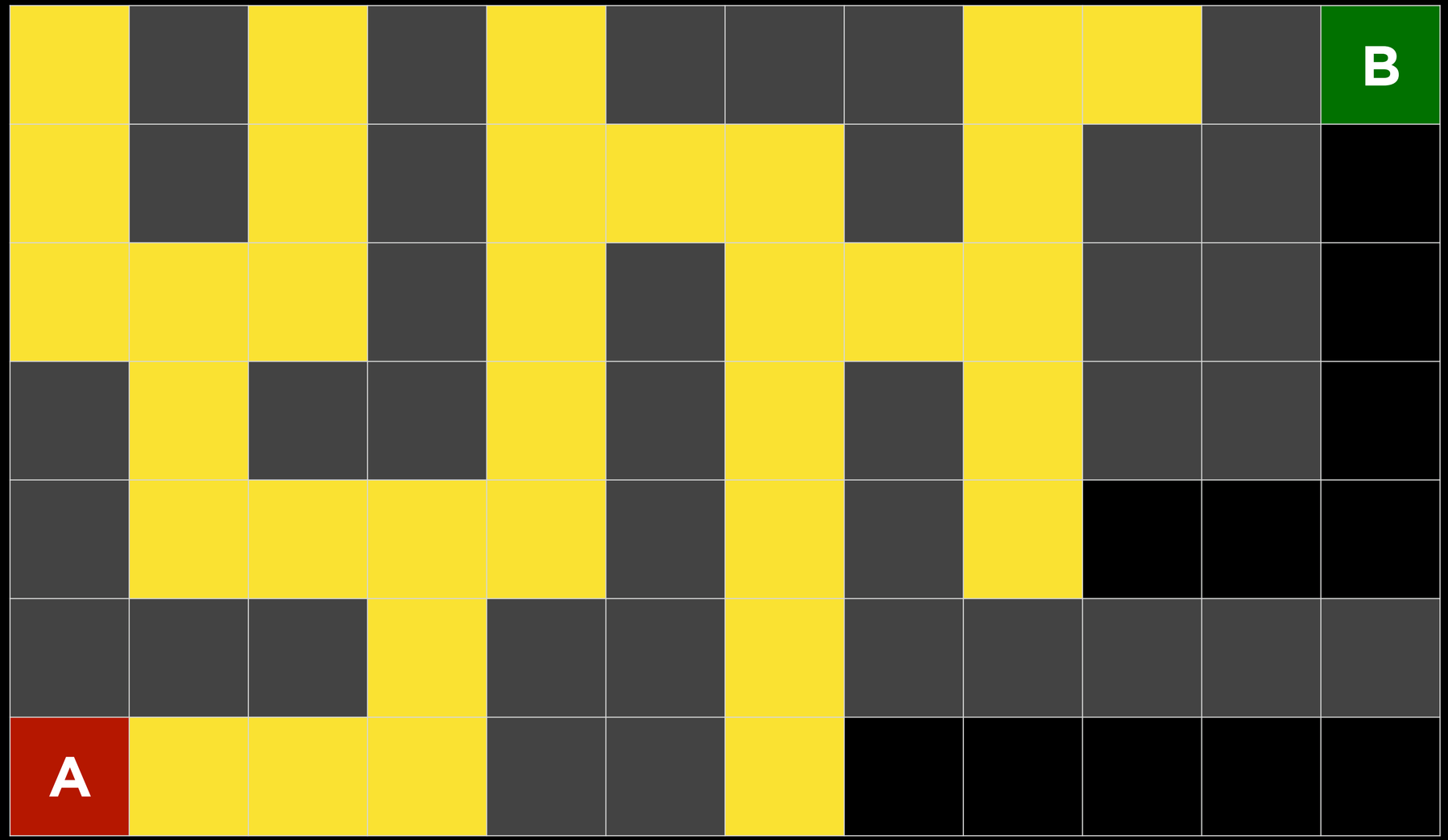
# Breadth-First Search



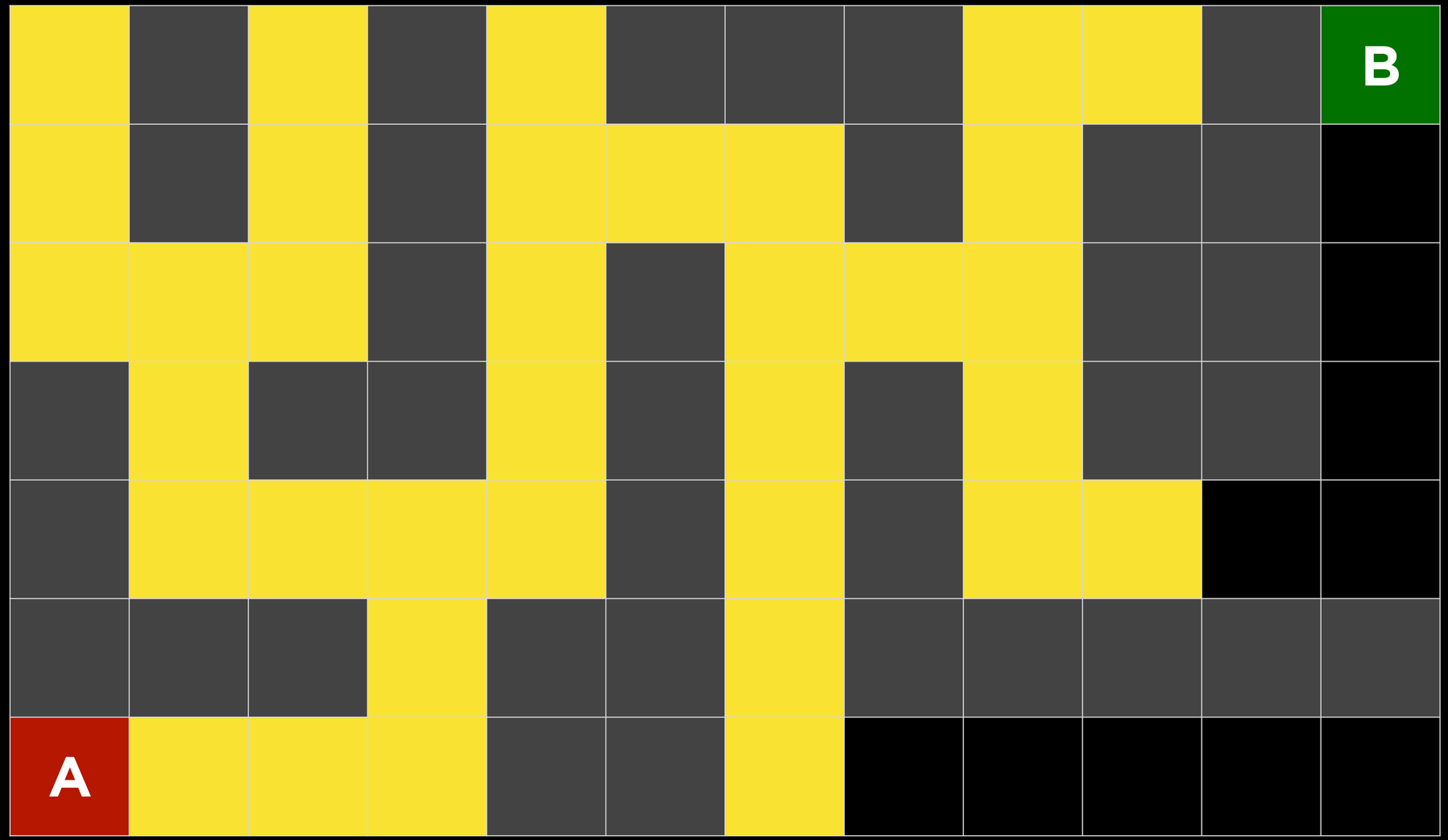
# Breadth-First Search



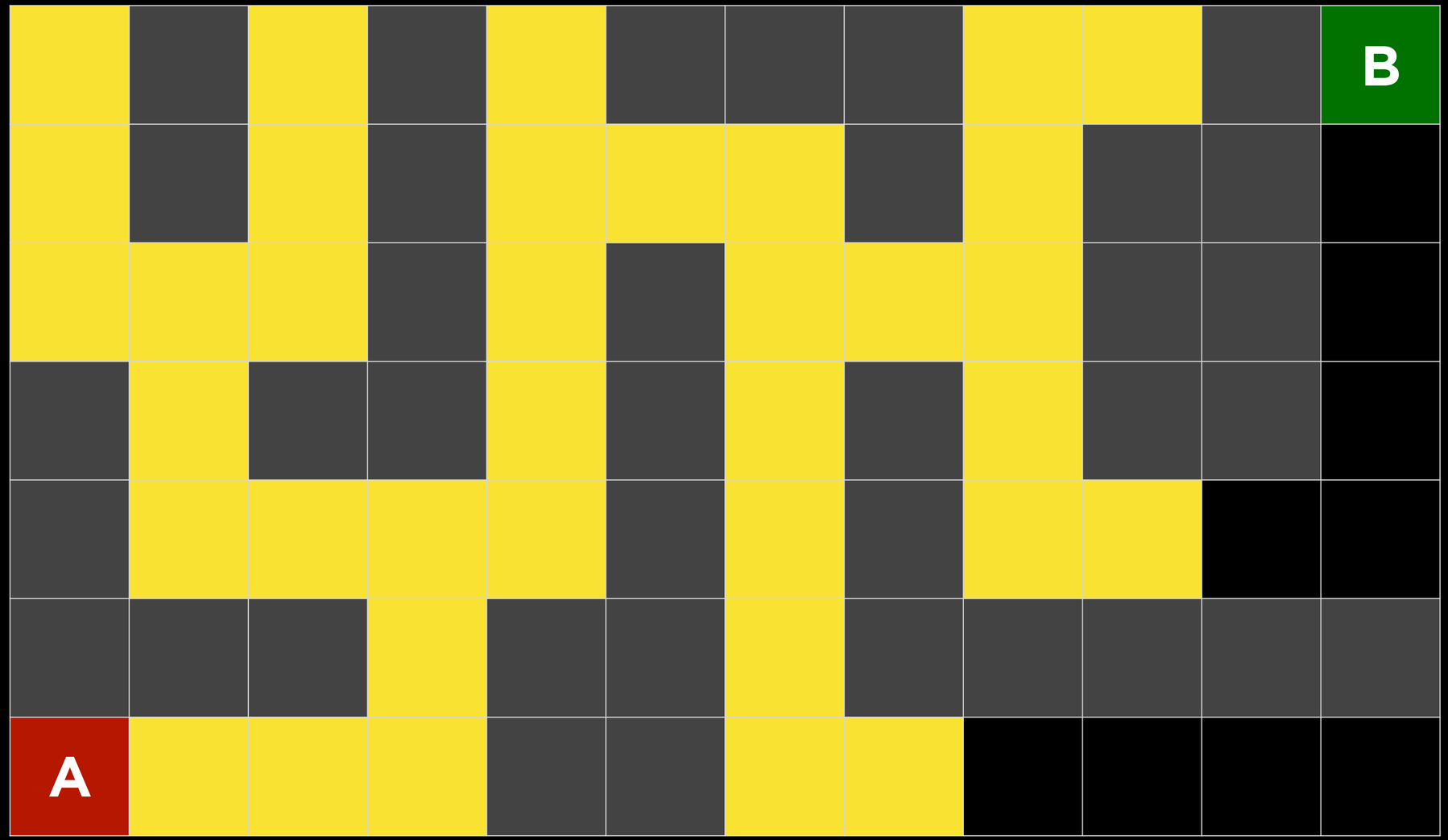
# Breadth-First Search



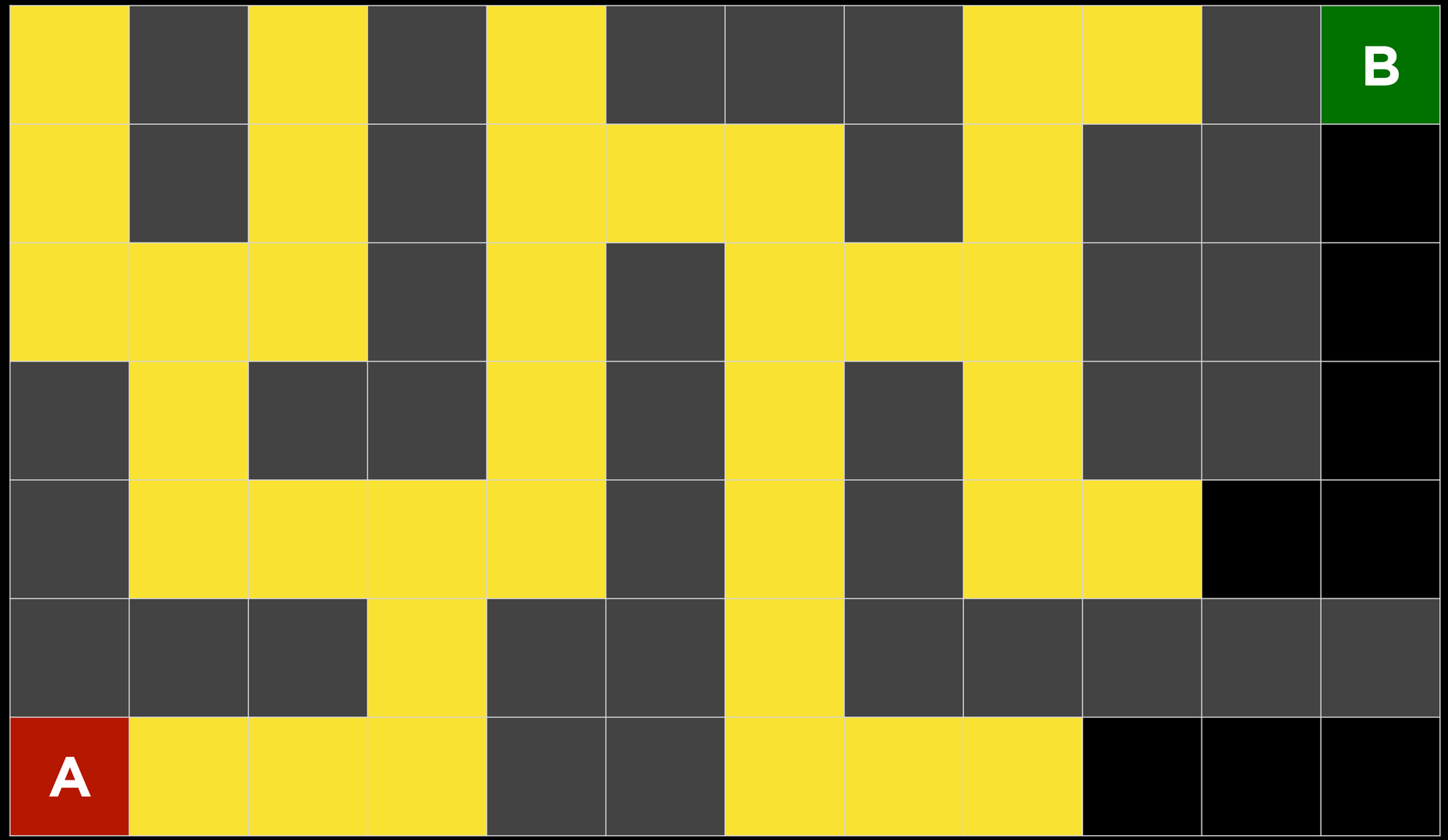
# Breadth-First Search



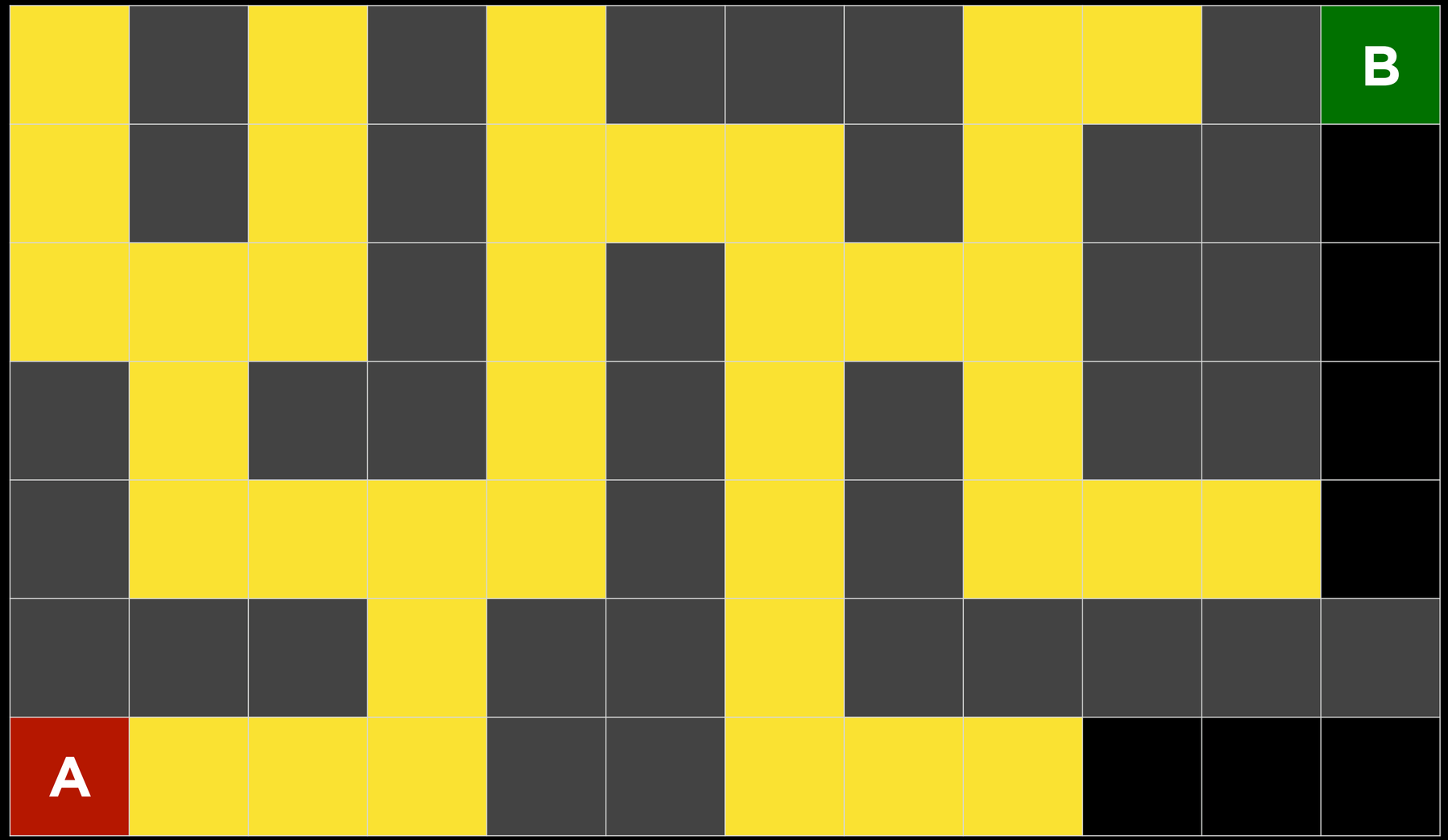
# Breadth-First Search



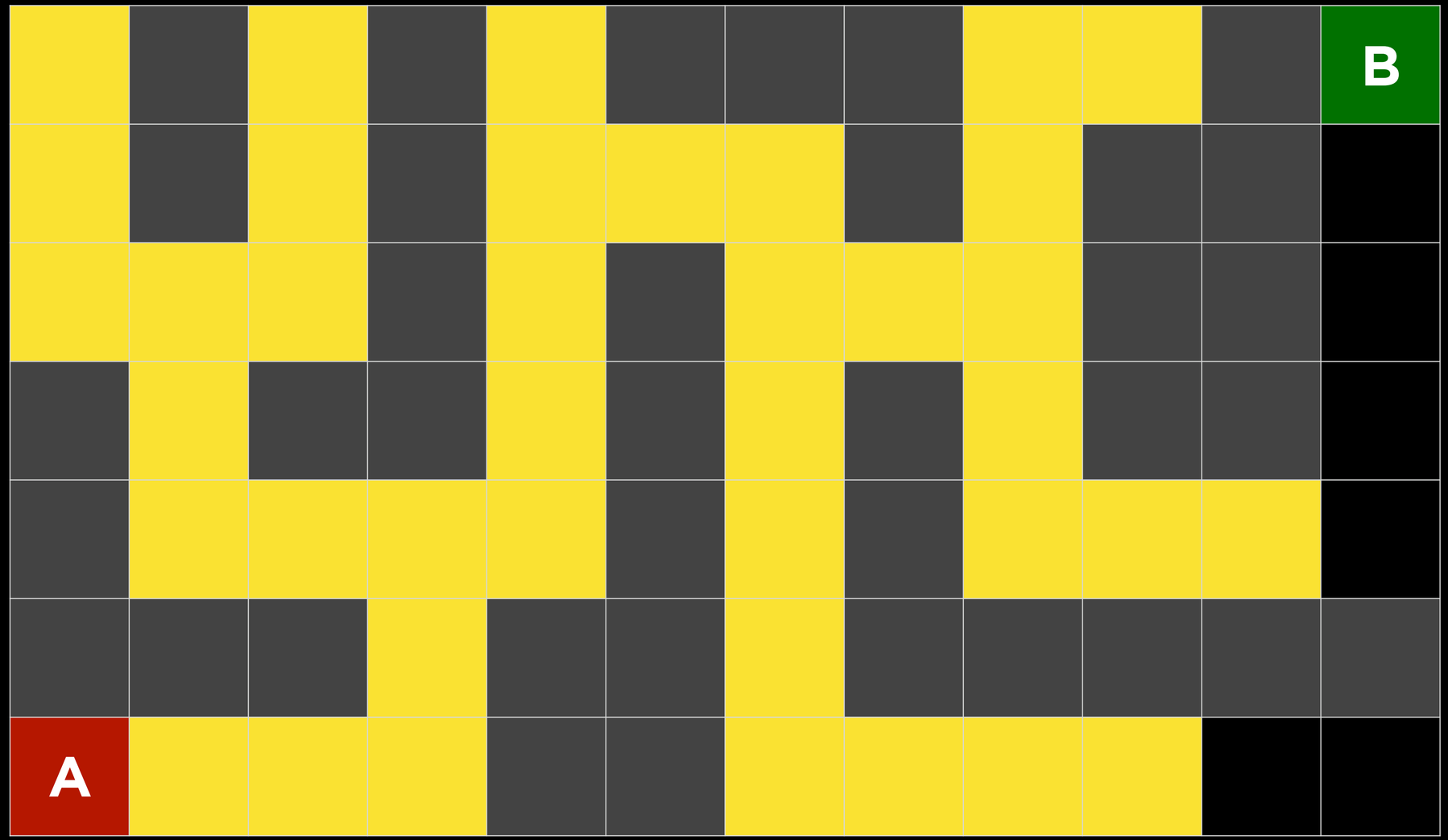
# Breadth-First Search



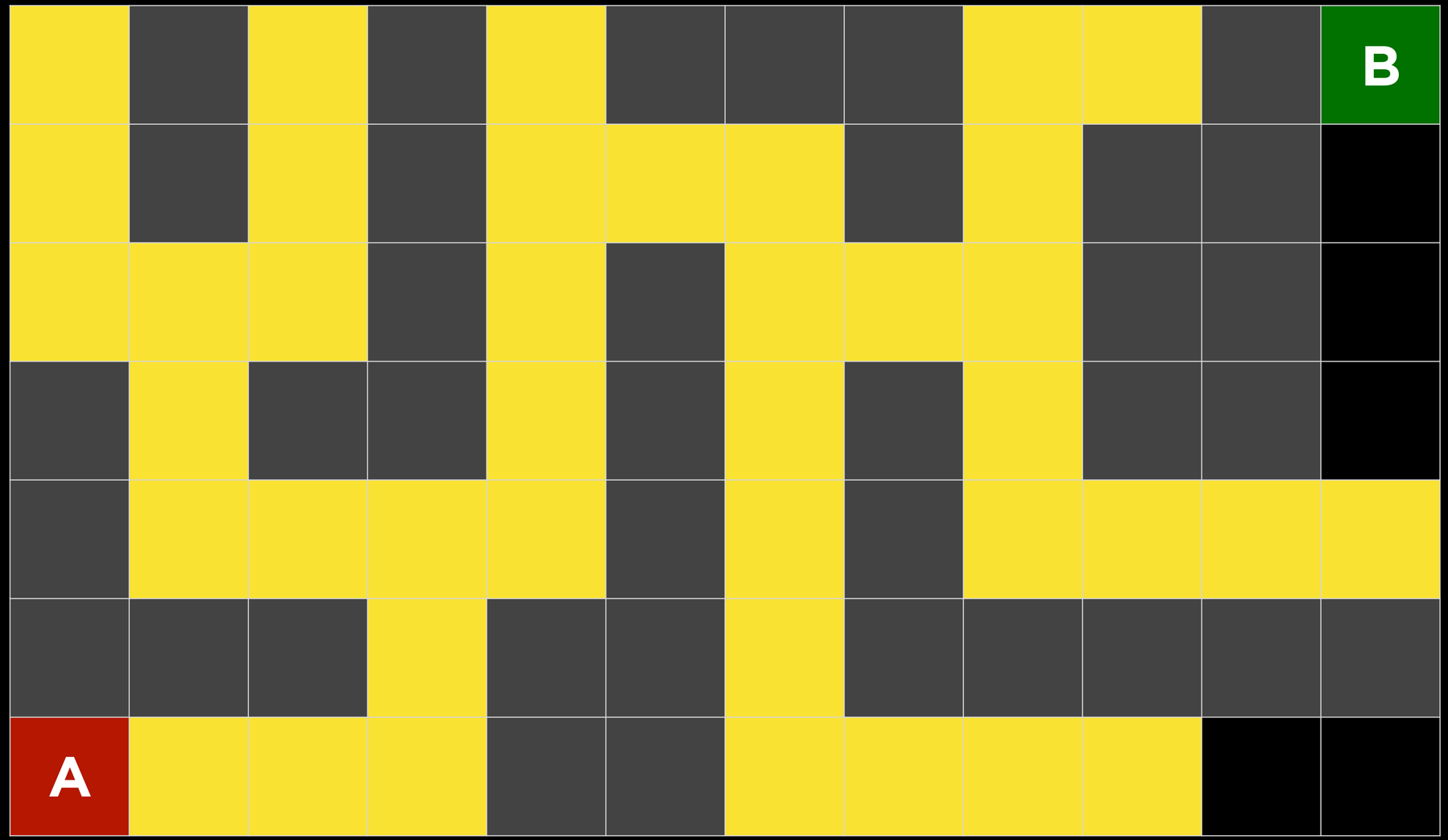
# Breadth-First Search



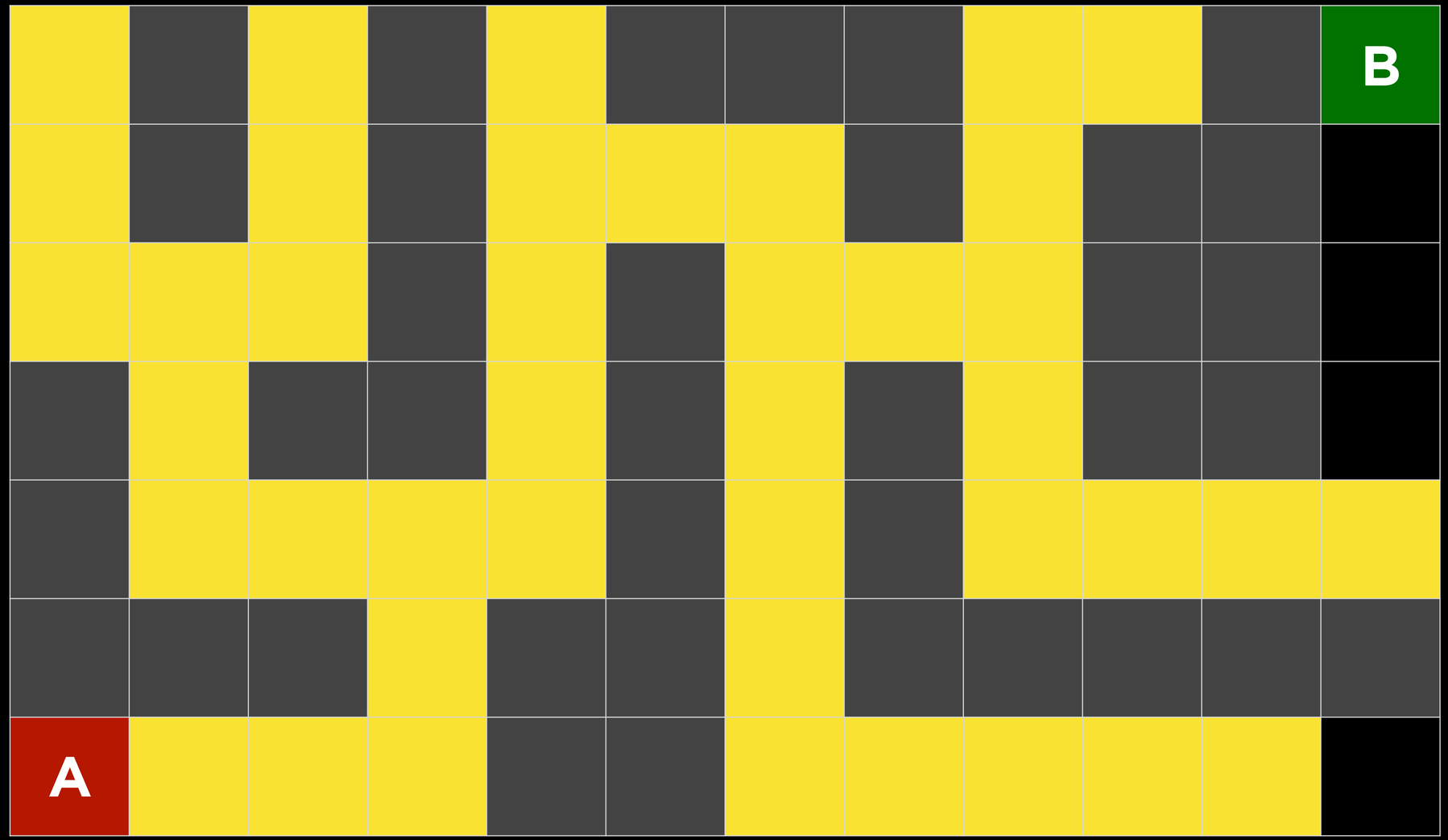
# Breadth-First Search



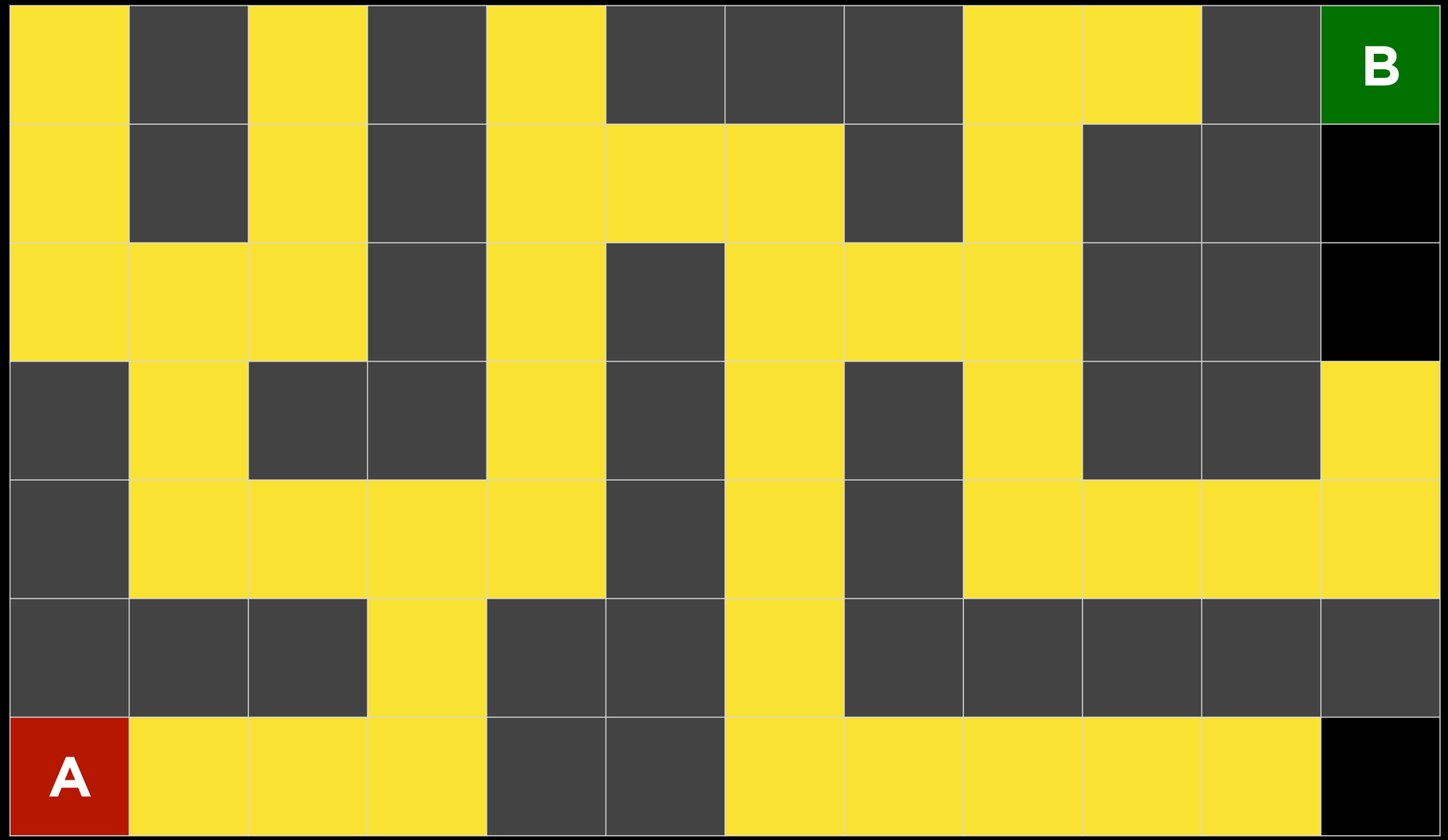
# Breadth-First Search



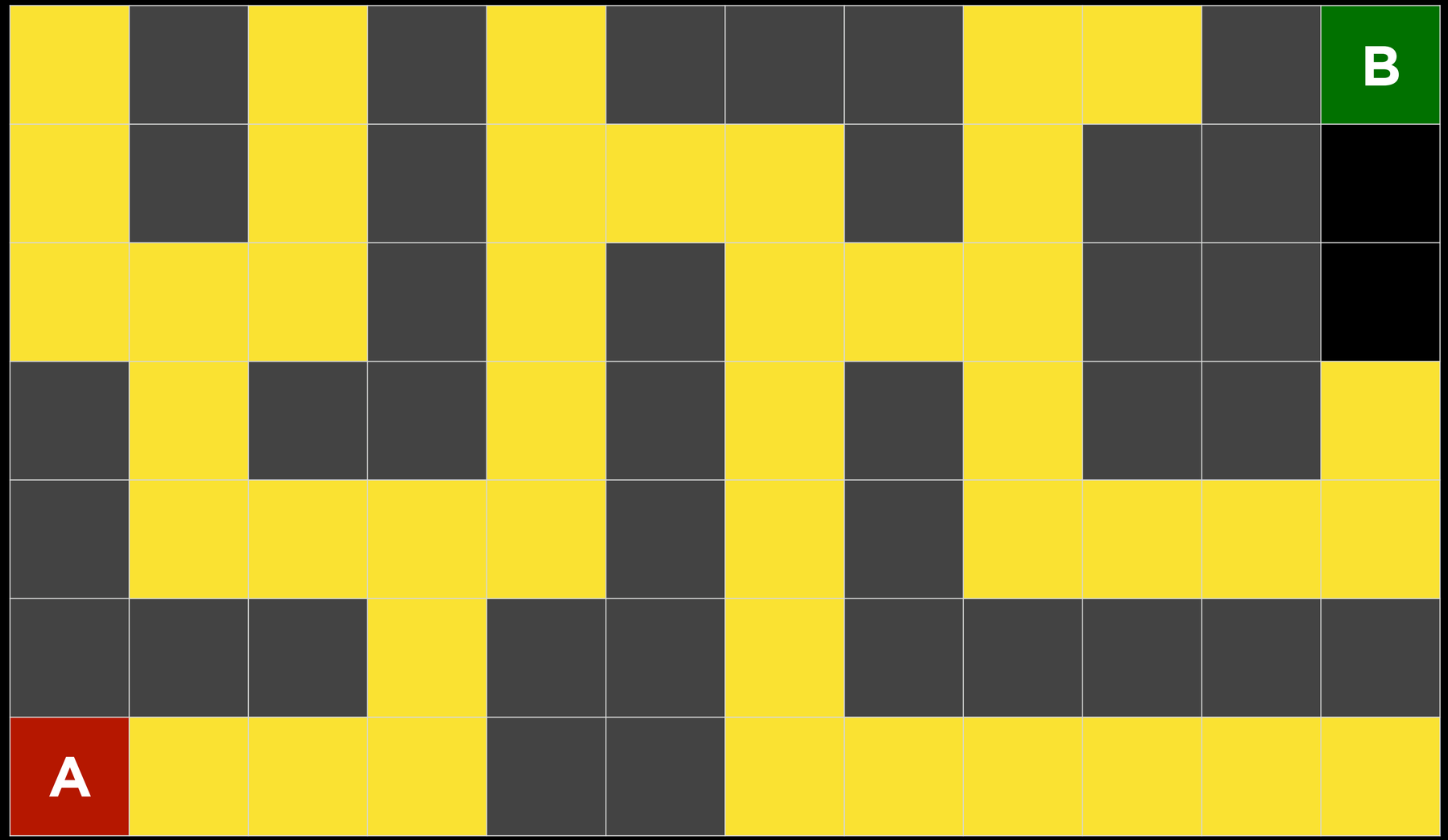
# Breadth-First Search



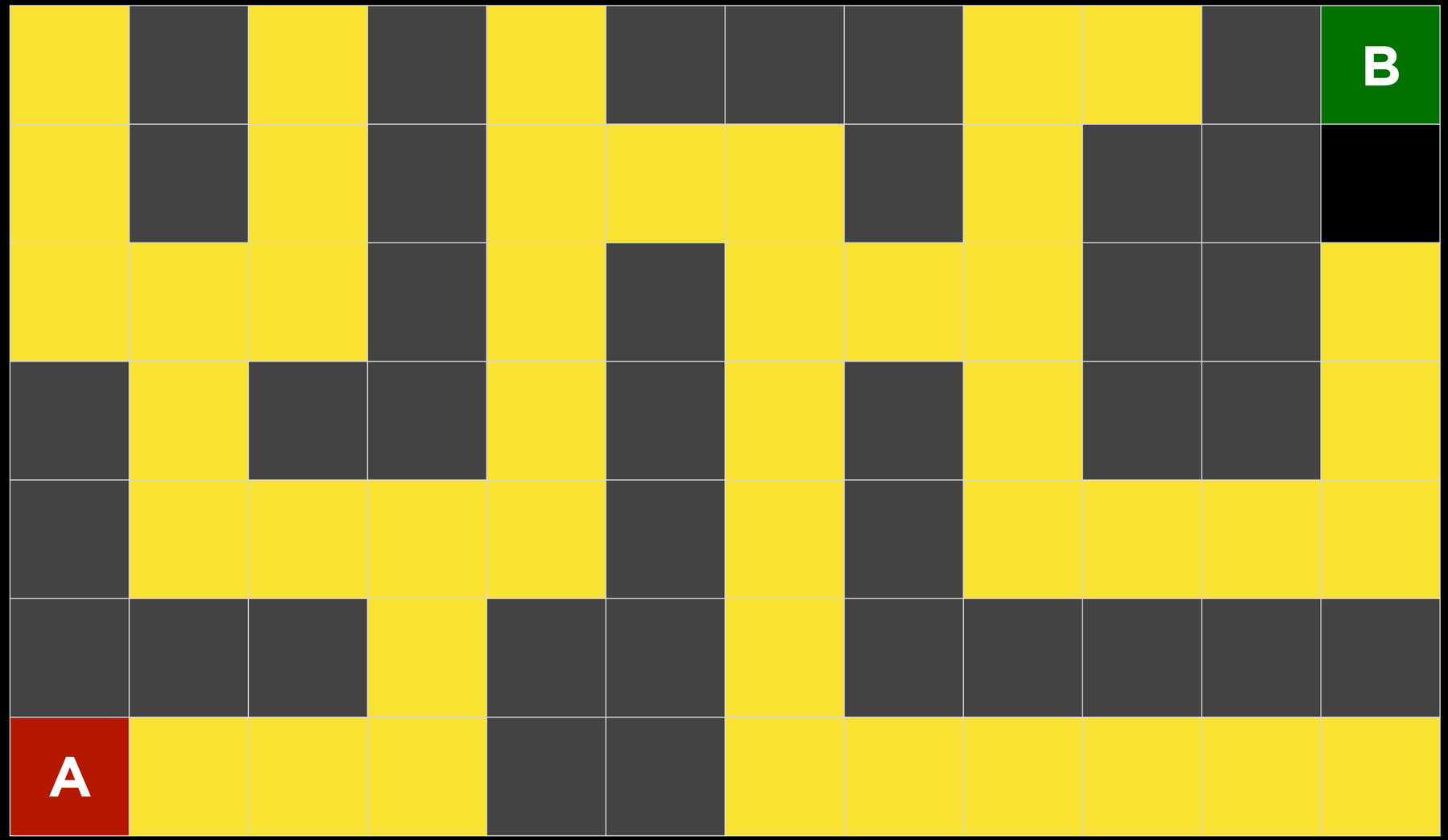
# Breadth-First Search



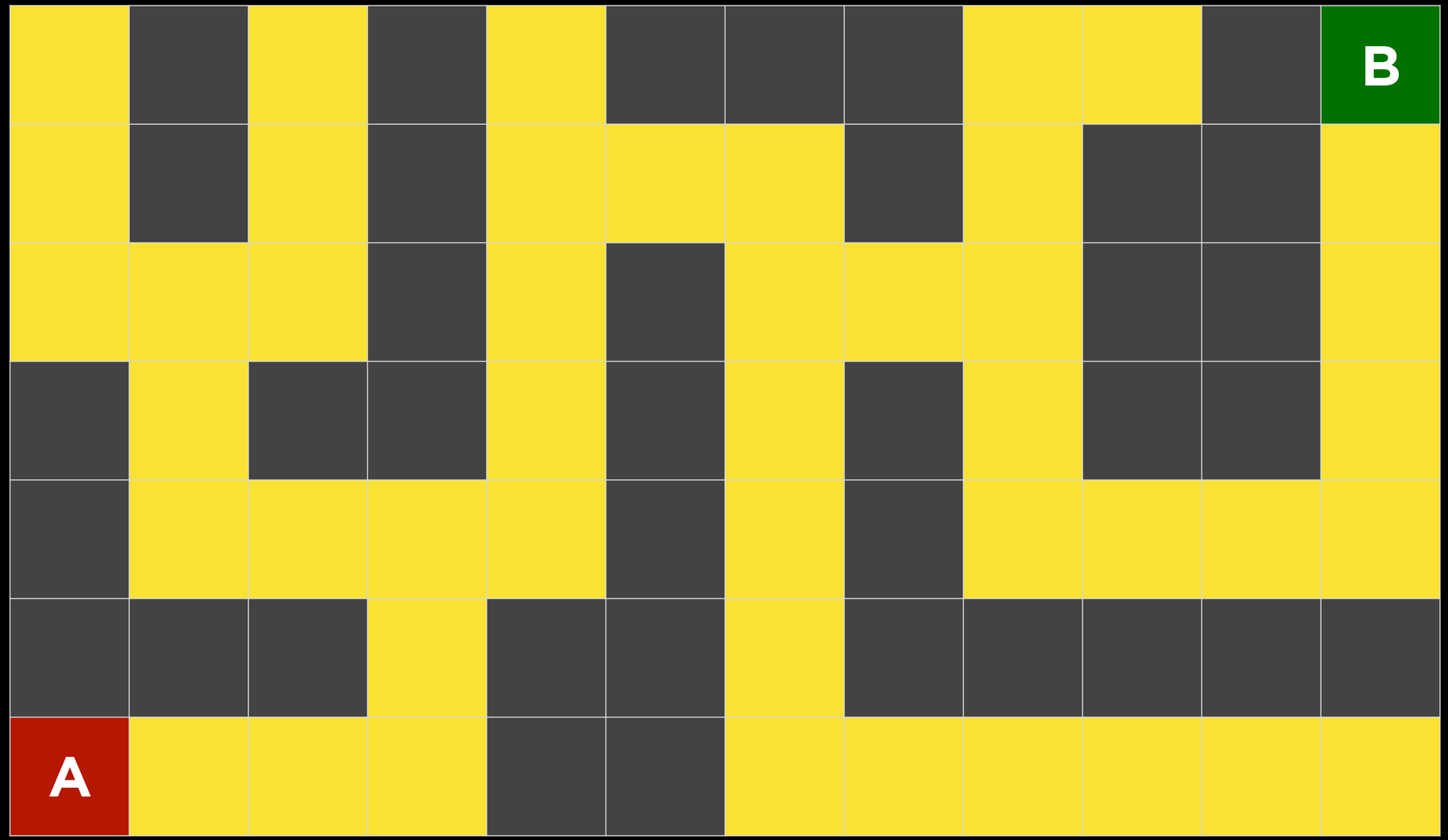
# Breadth-First Search



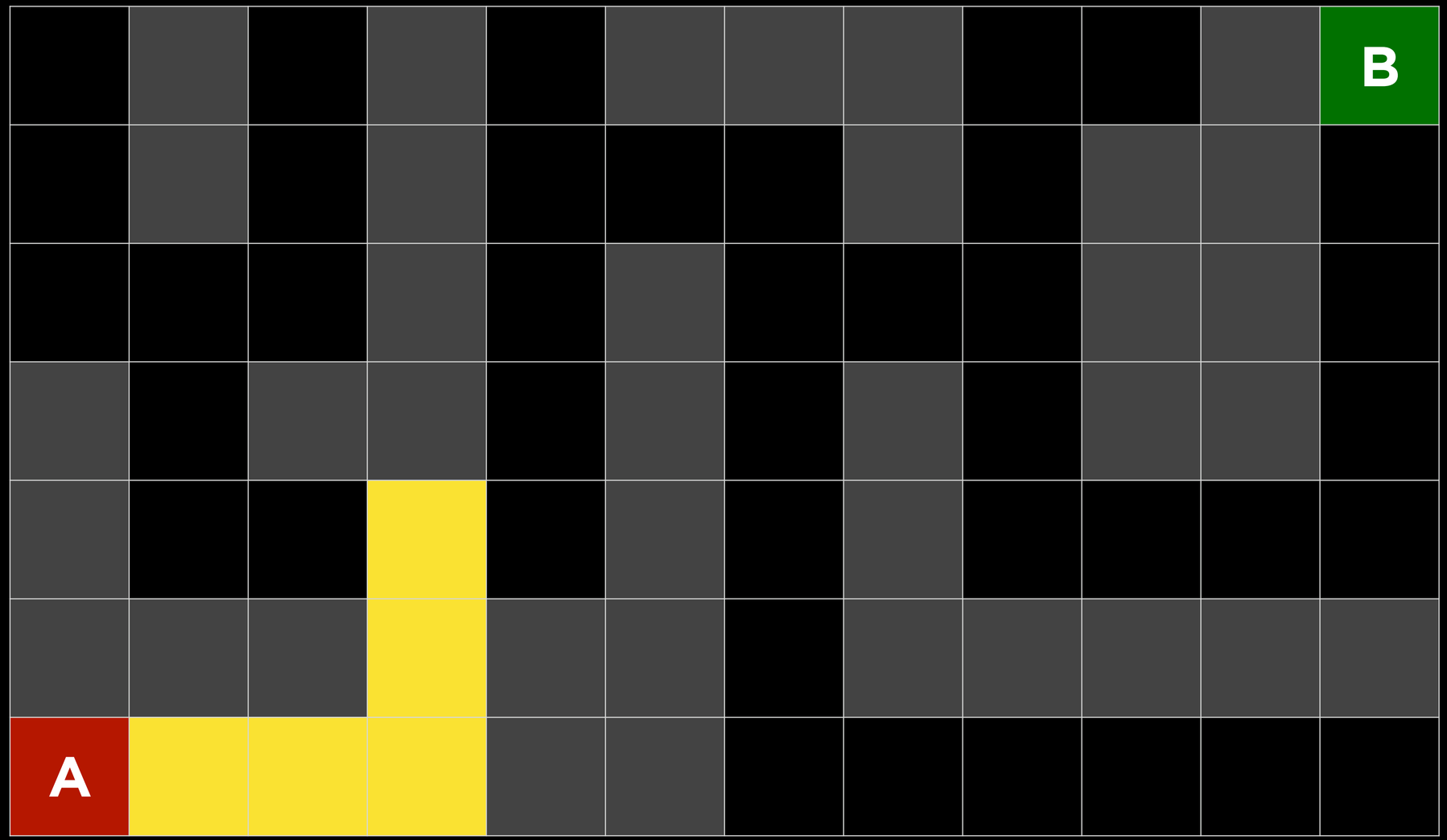
# Breadth-First Search



# Breadth-First Search



# Breadth-First Search



# uninformed search

search strategy that uses no problem-specific knowledge

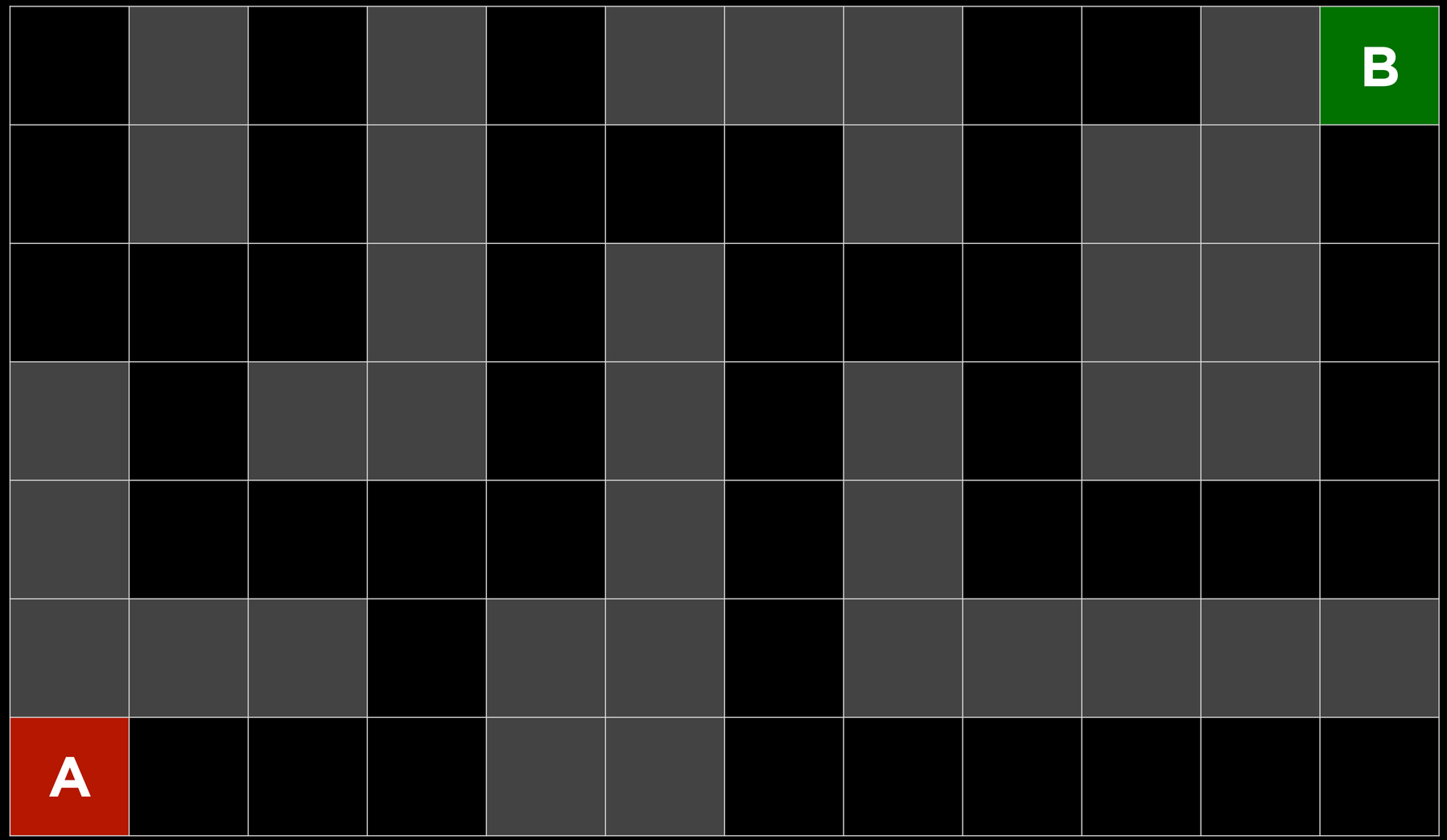
# informed search

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

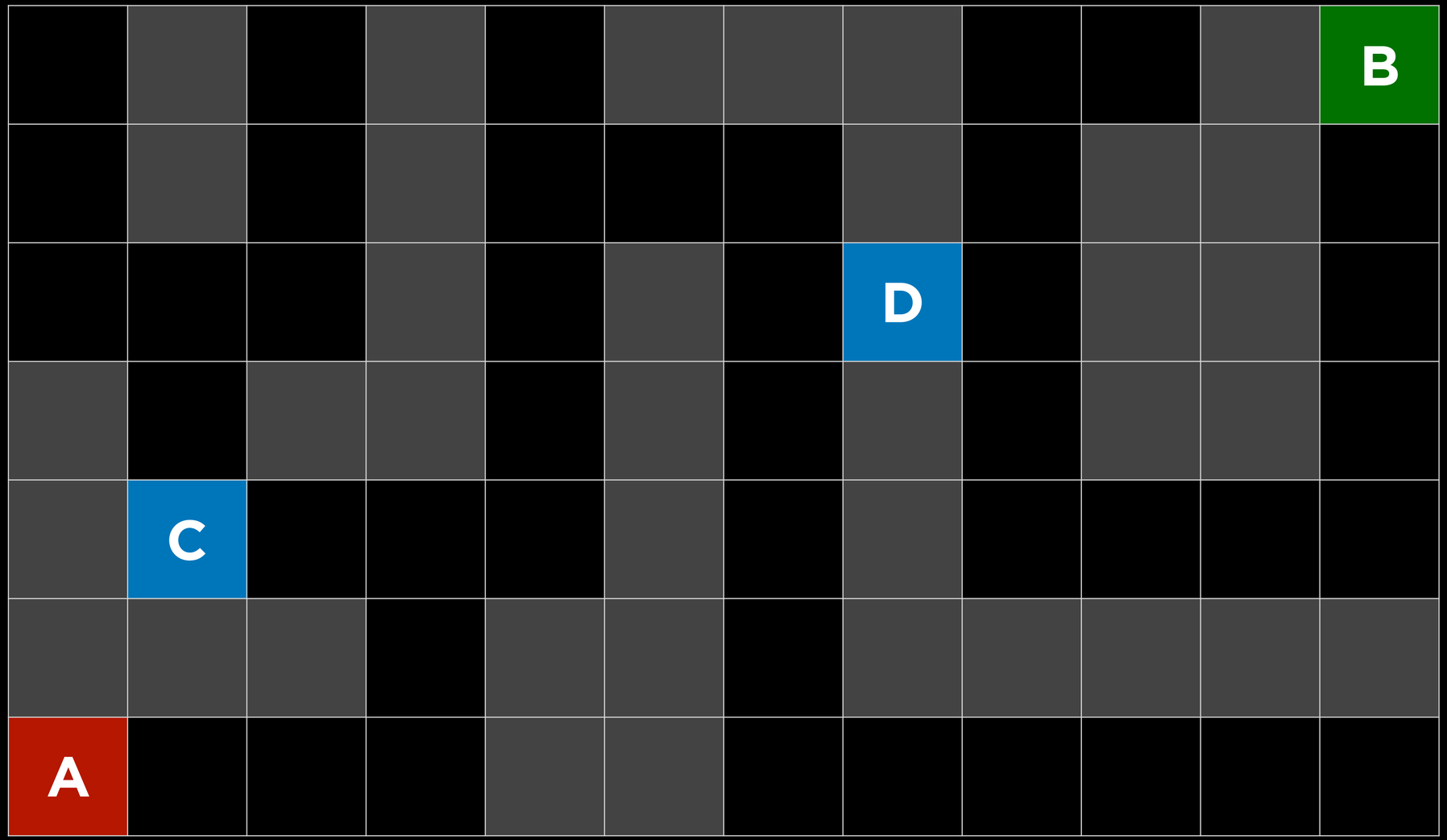
# greedy best-first search

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

# Heuristic function?

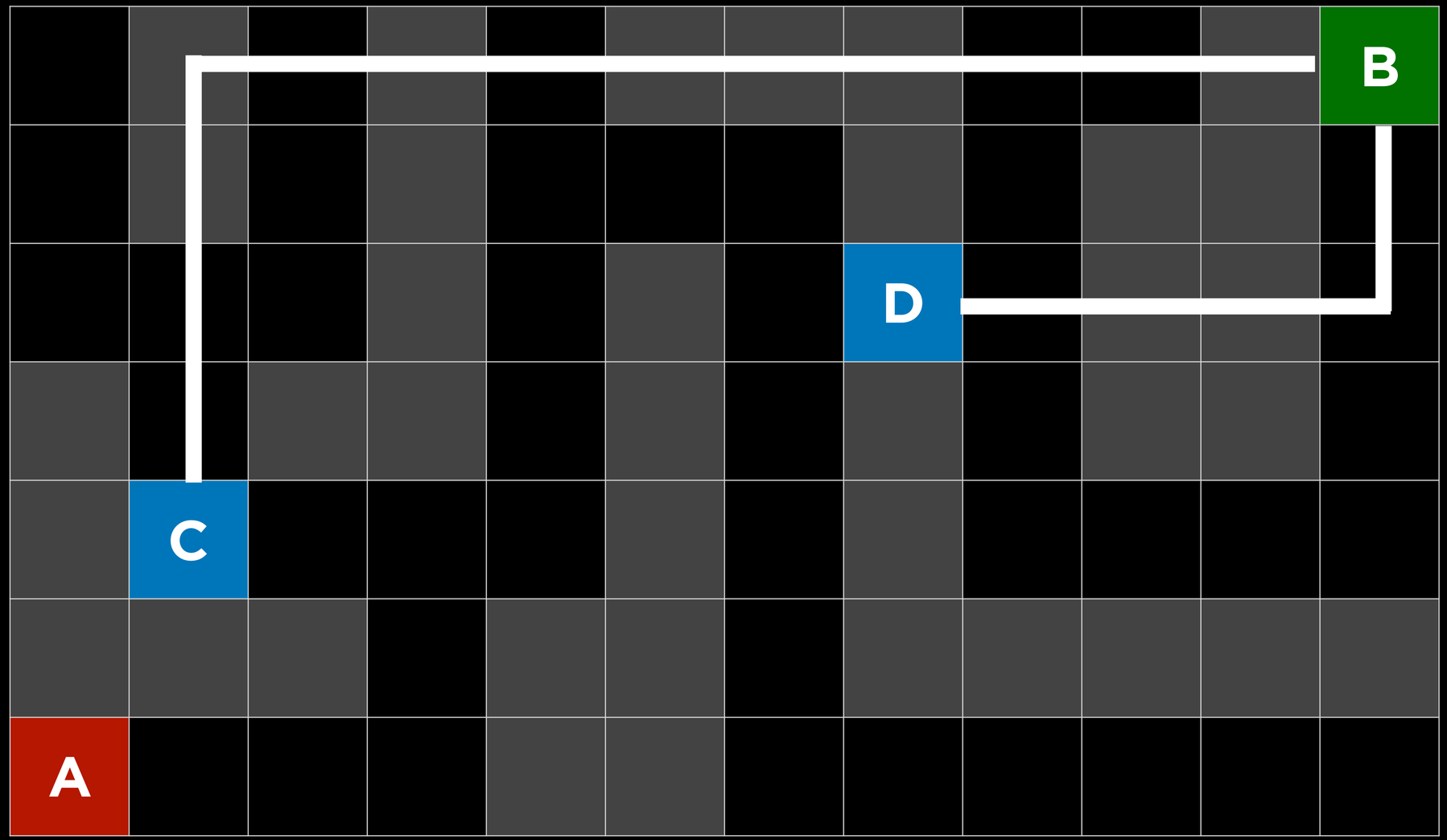


# Heuristic function?



manhattan(D) < manhattan(C)

# Heuristic function? Manhattan distance.

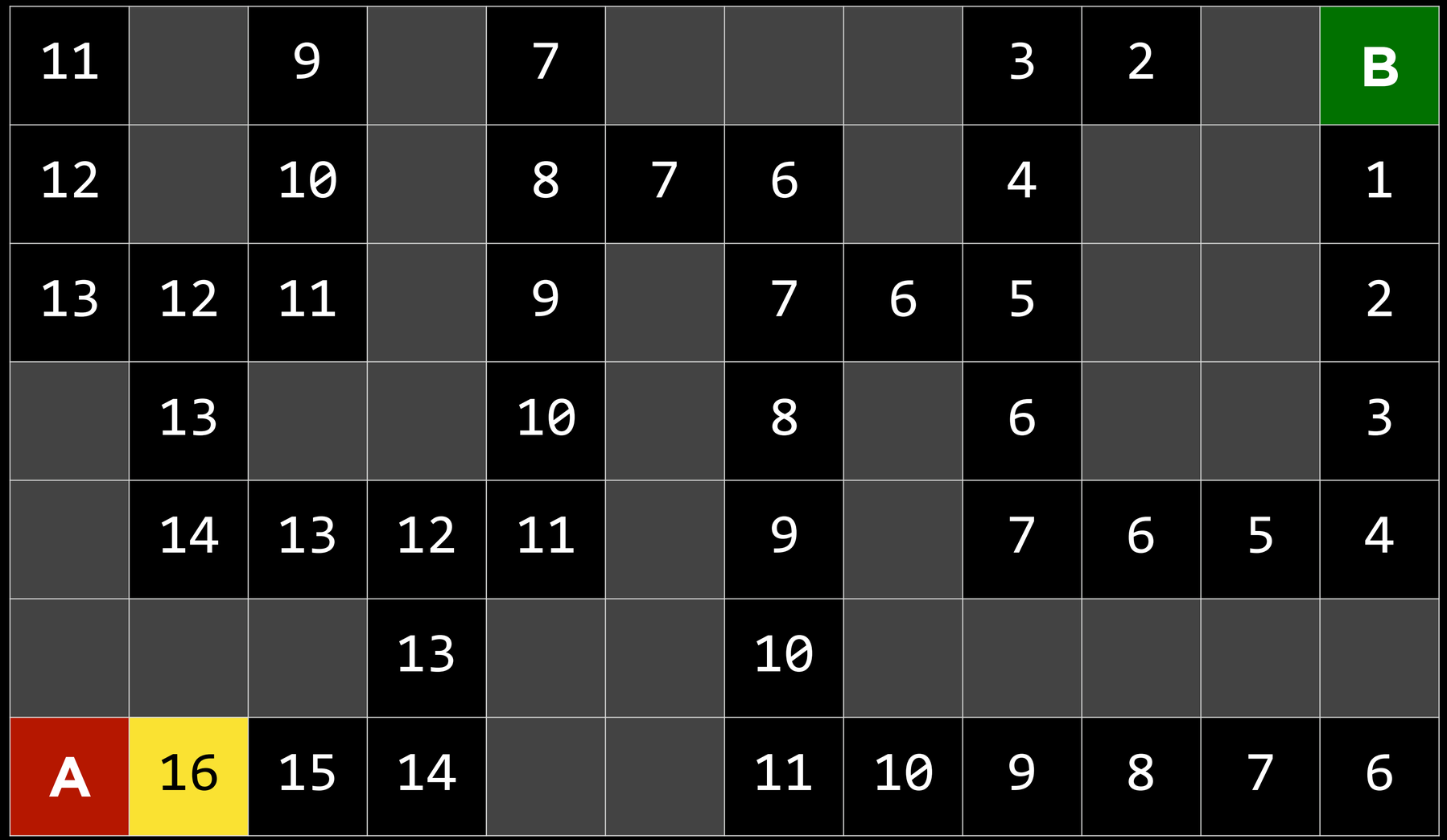


# Greedy Best-First Search

problem with manhattan distance shown ignoring walls

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

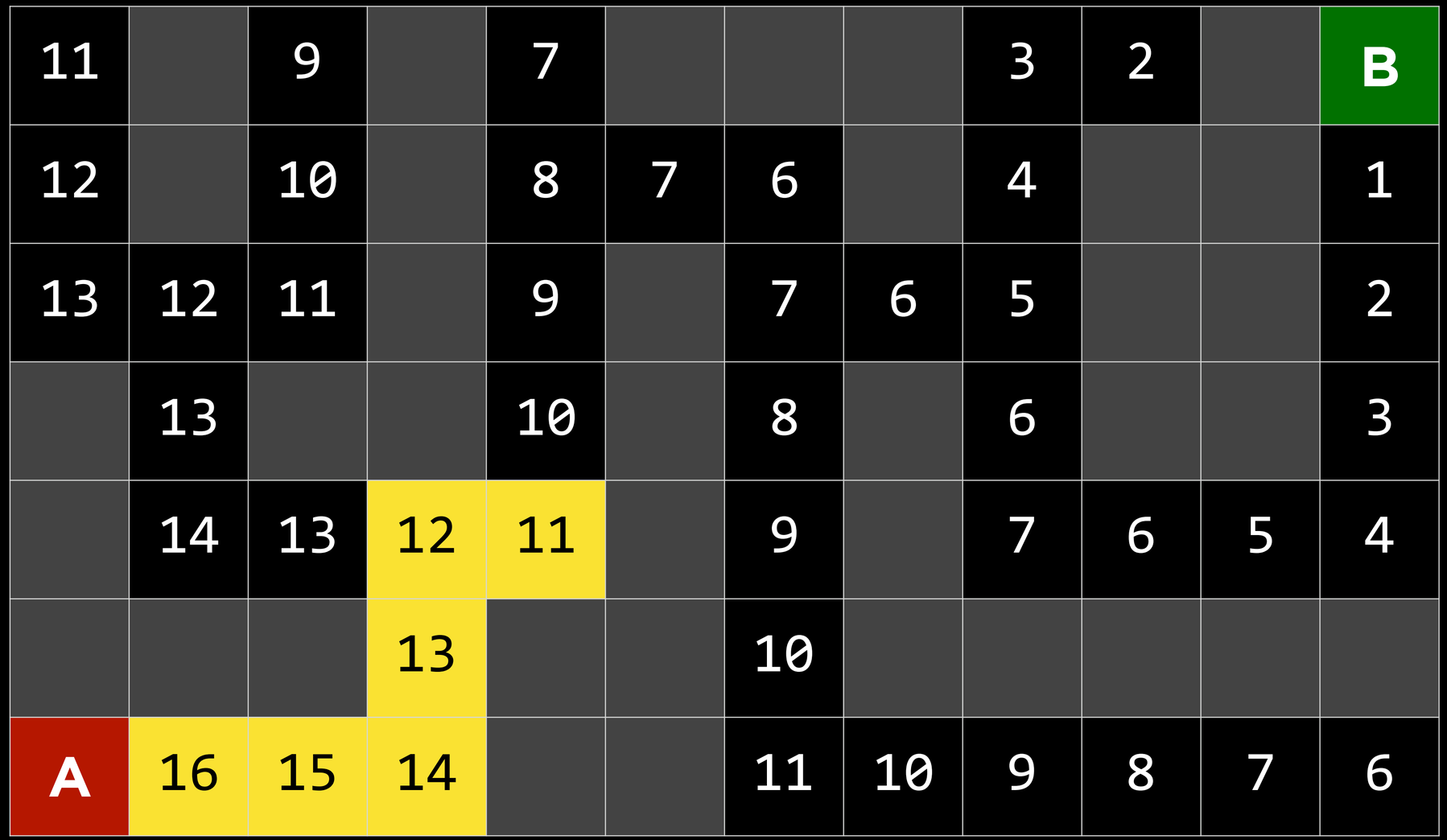
# Greedy Best-First Search

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

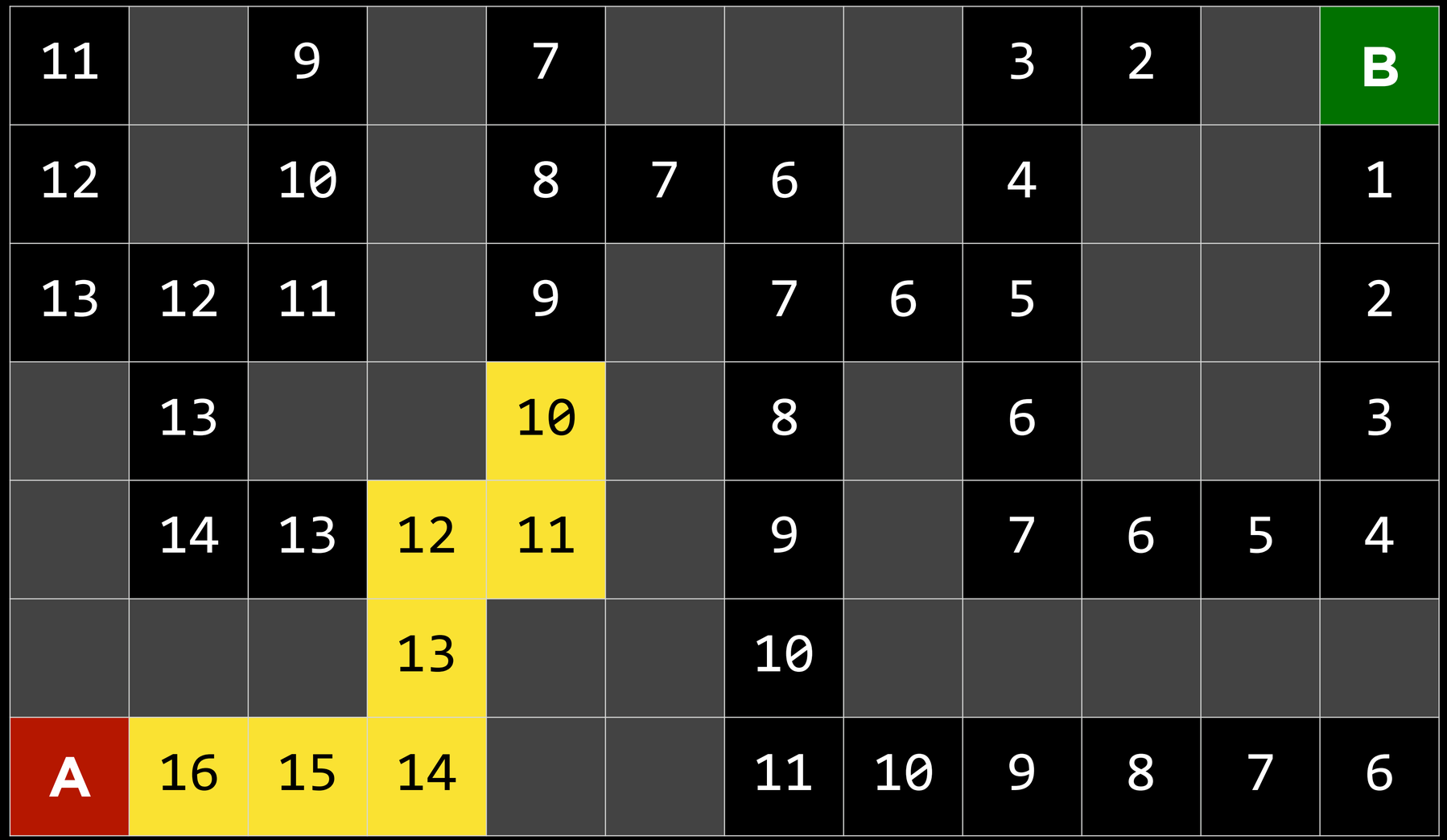
# Greedy Best-First Search

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

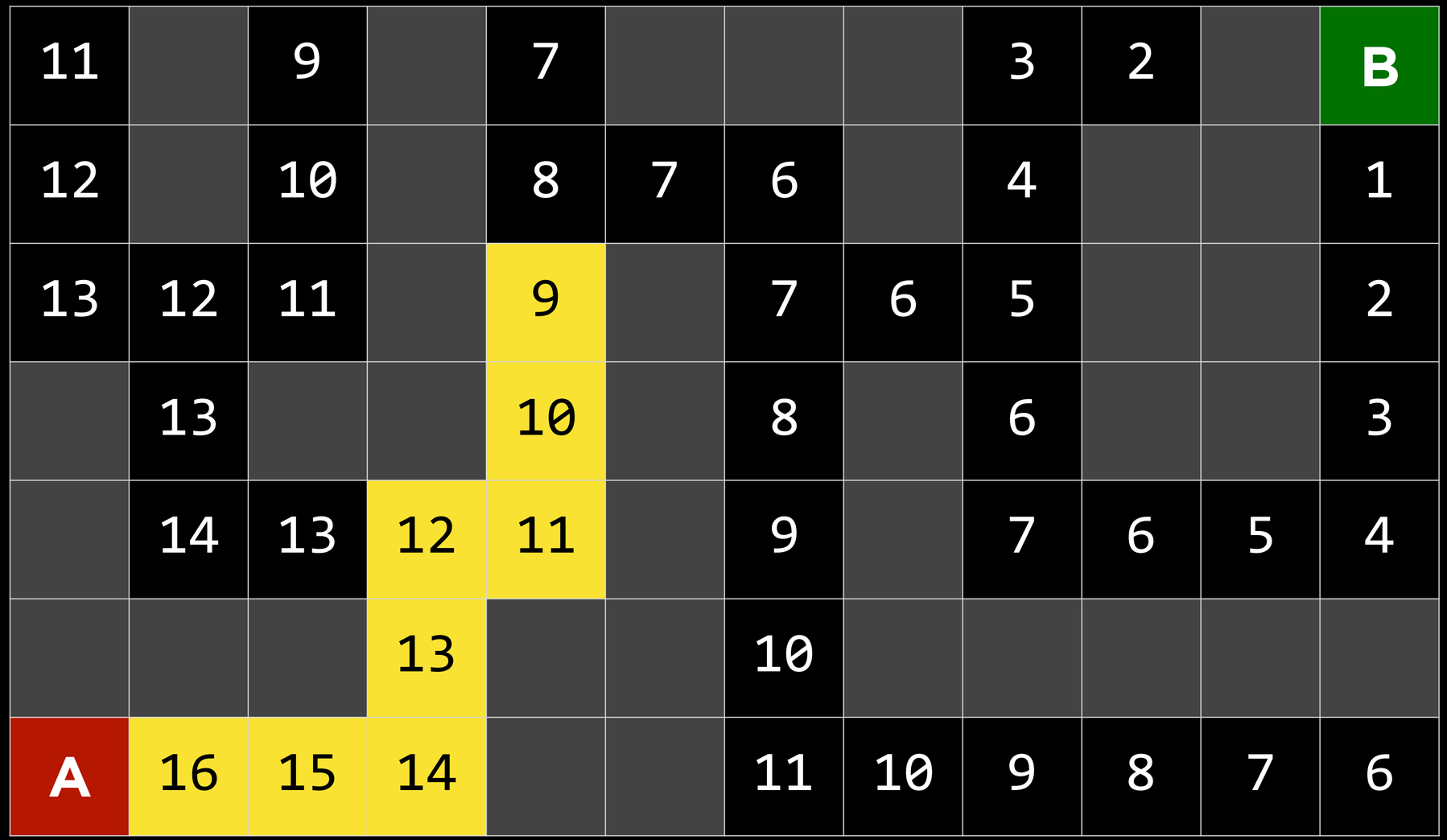
# Greedy Best-First Search



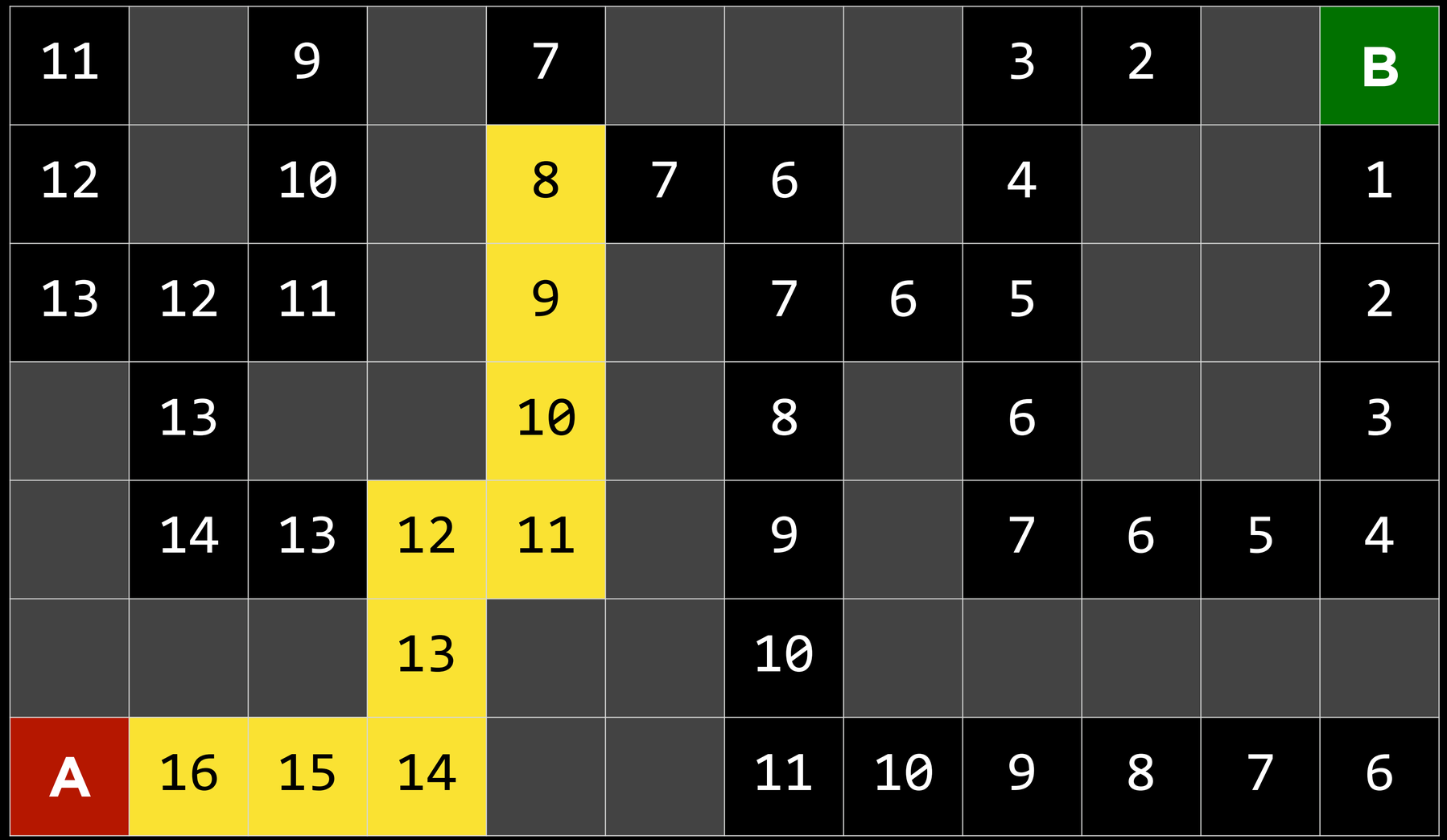
# Greedy Best-First Search



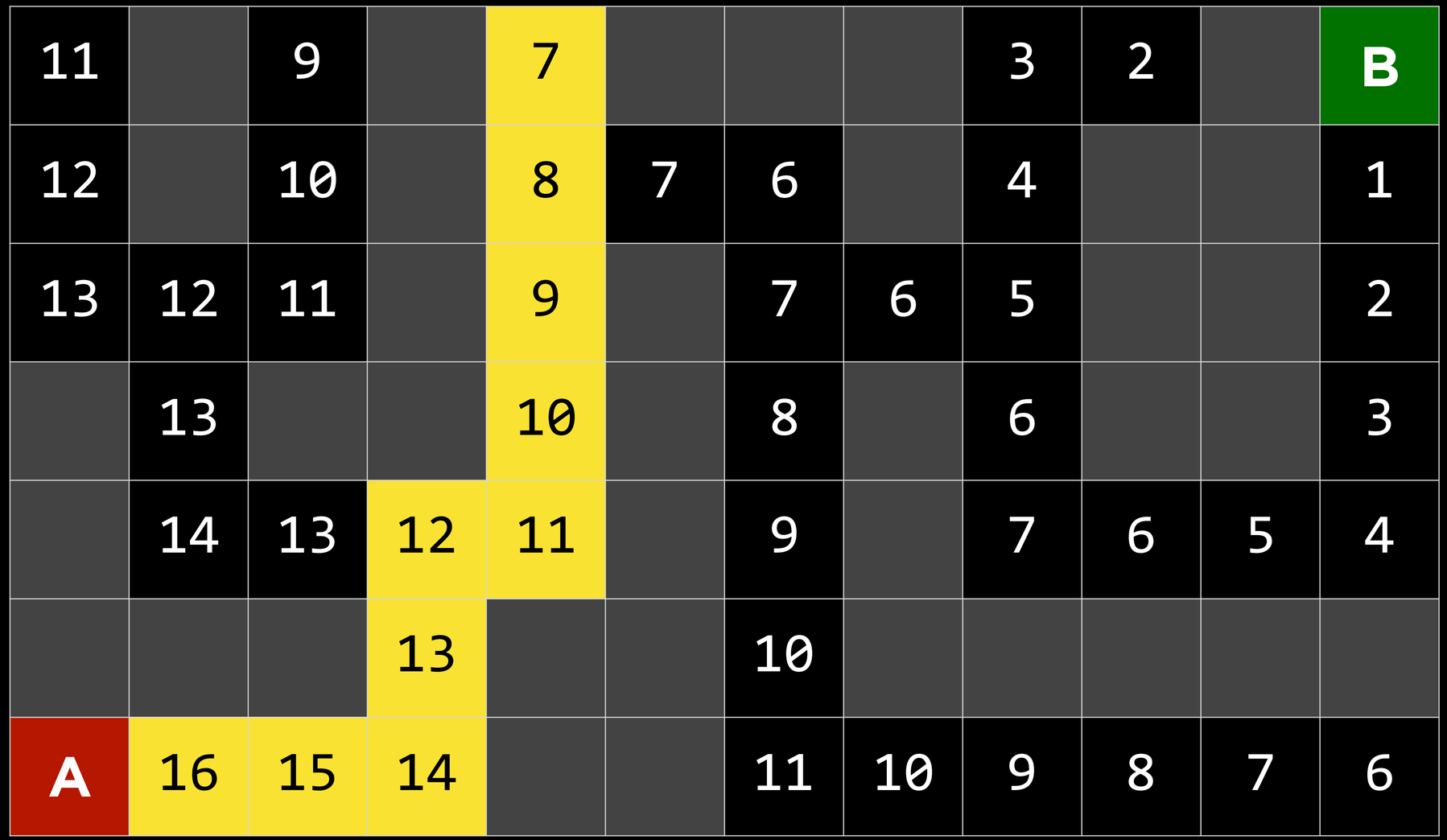
# Greedy Best-First Search



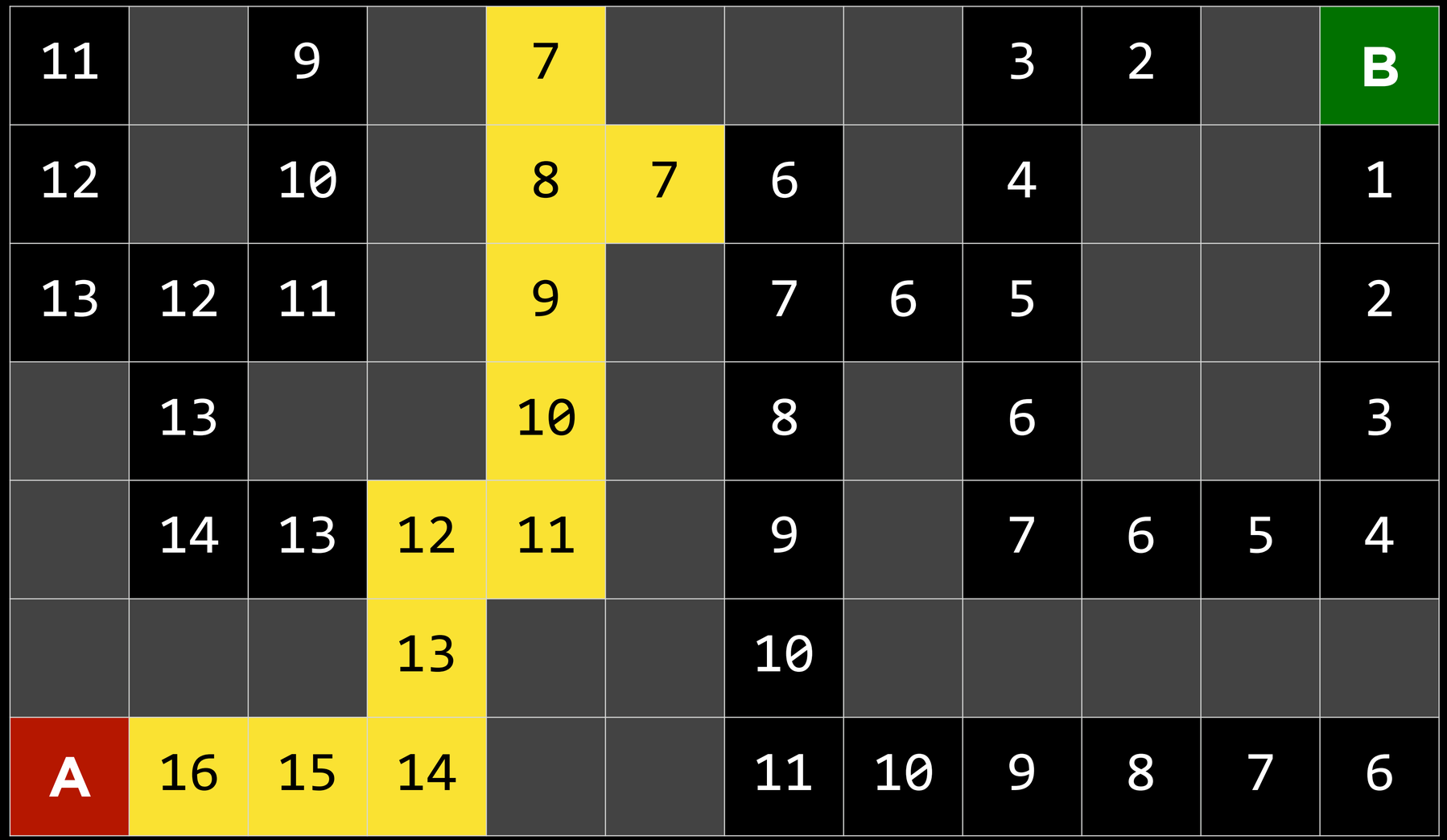
# Greedy Best-First Search



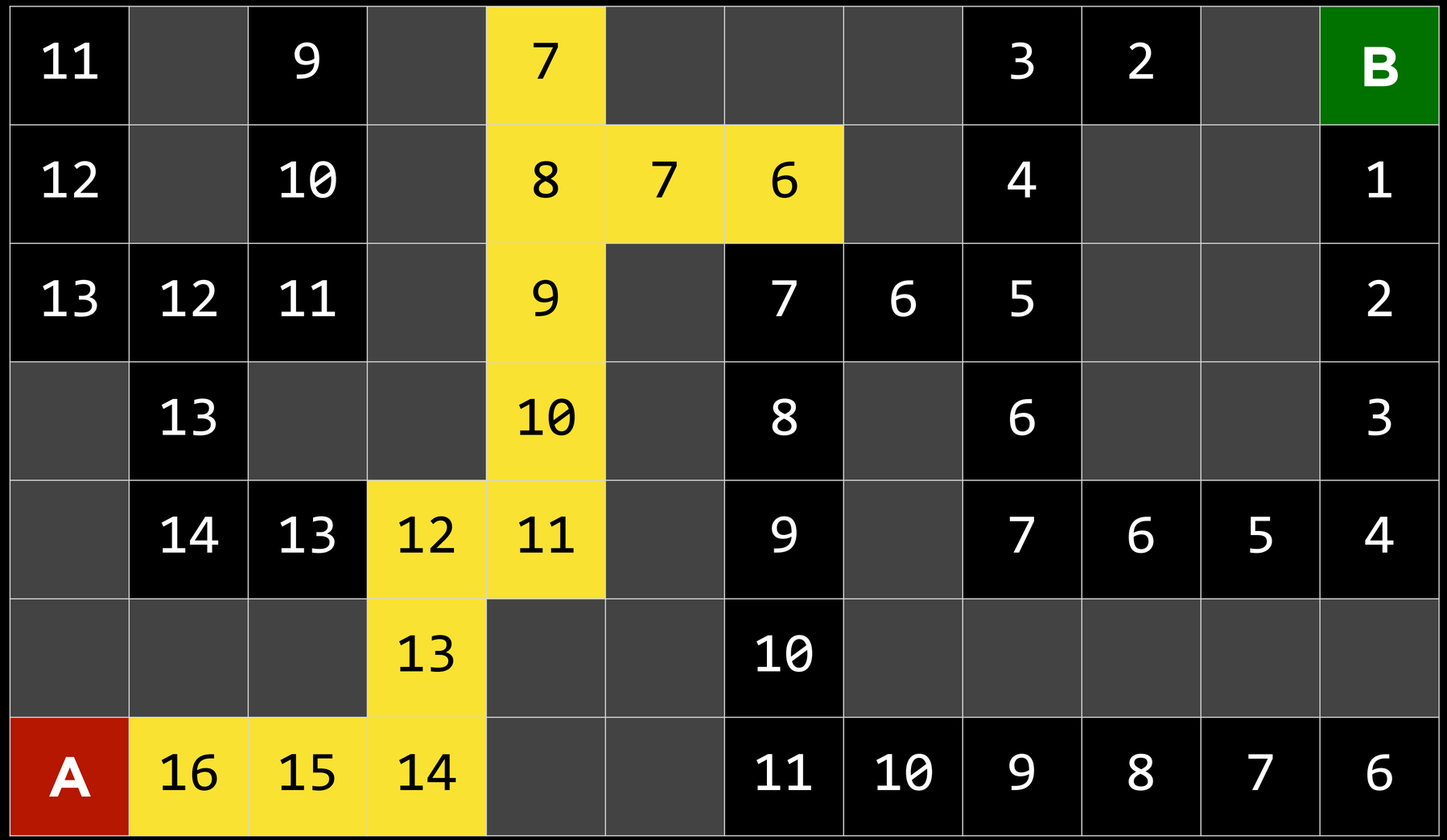
# Greedy Best-First Search



# Greedy Best-First Search

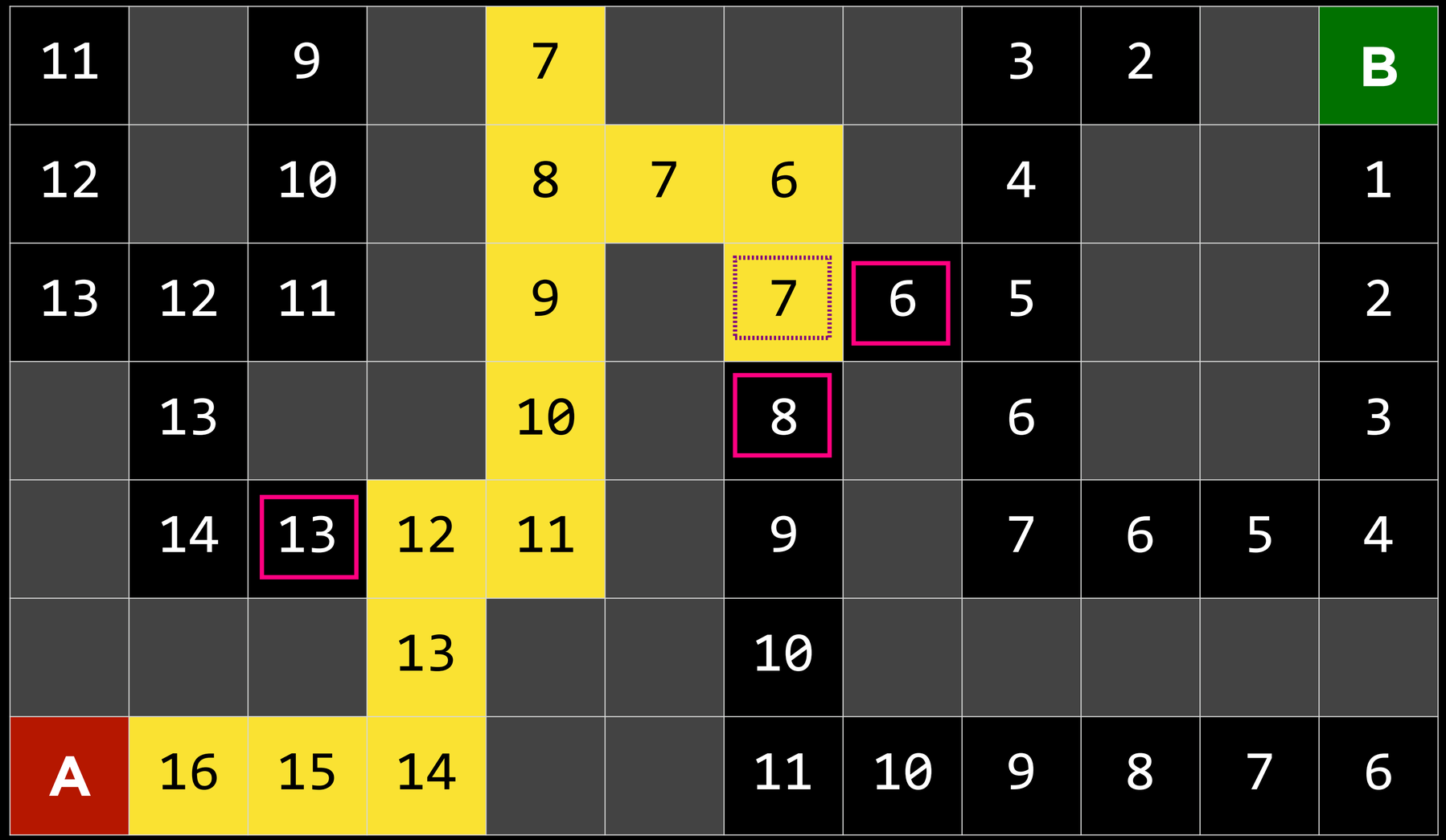


# Greedy Best-First Search

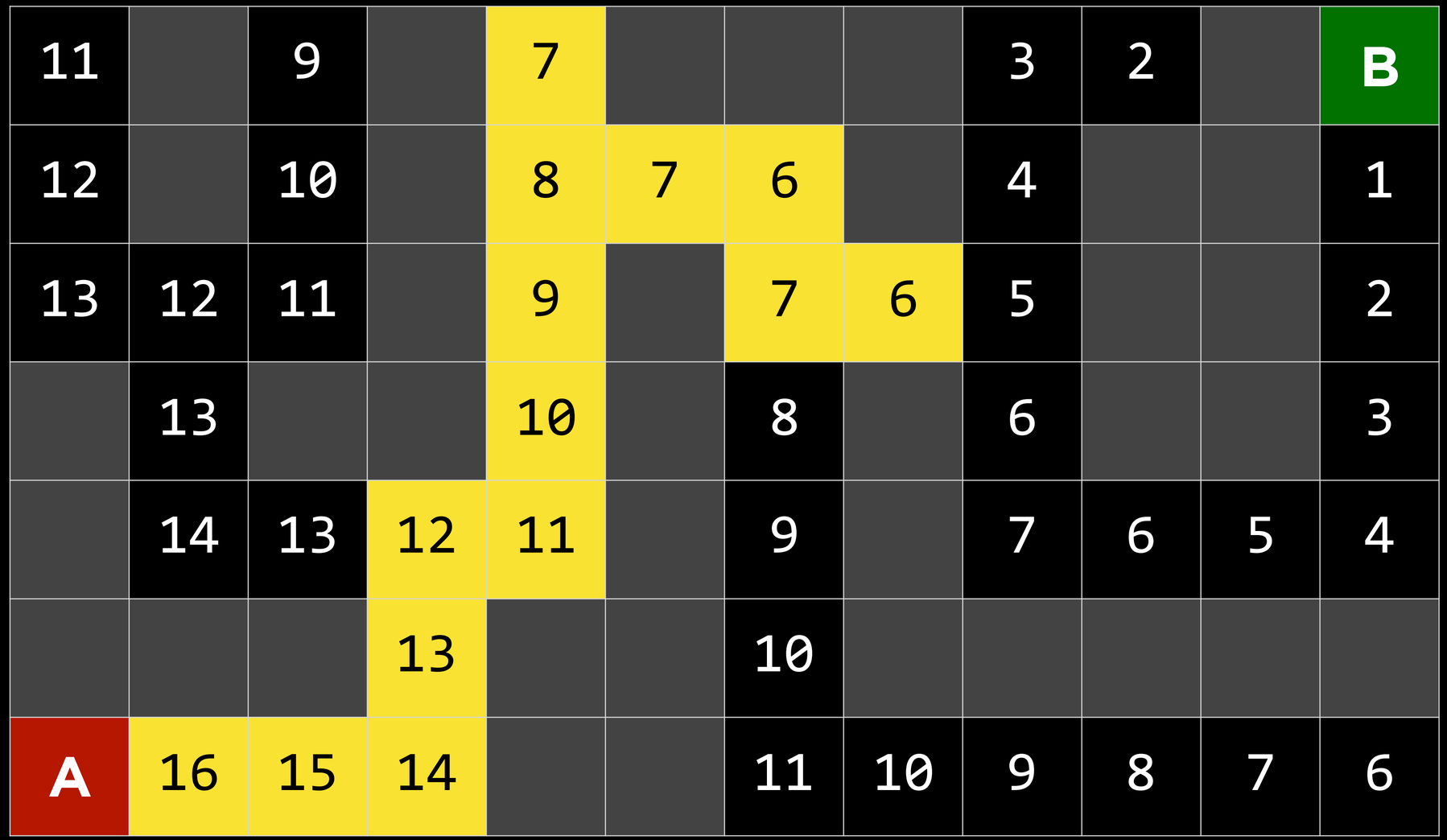


# Greedy Best-First Search

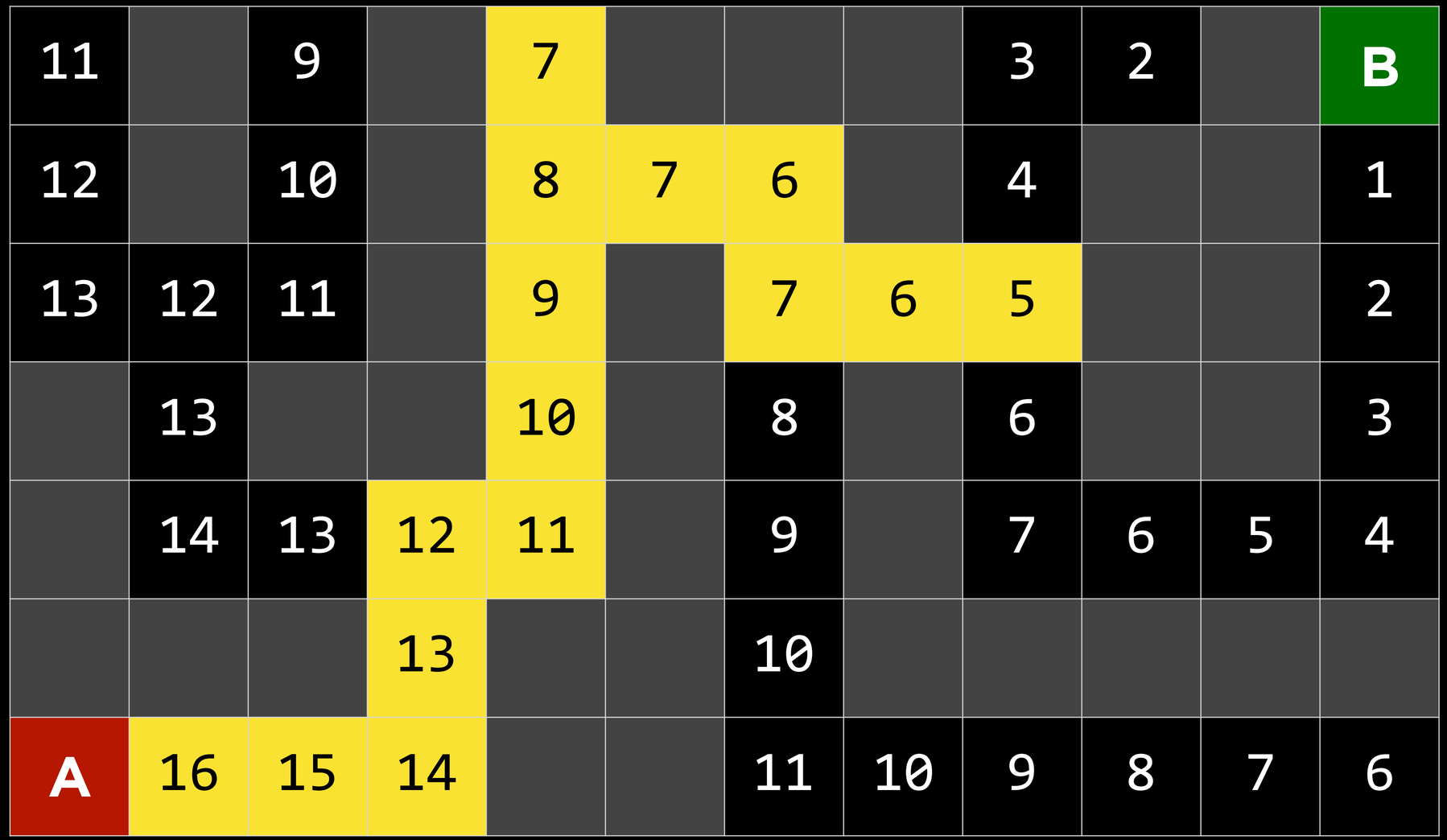
min(6,8,13) = 6



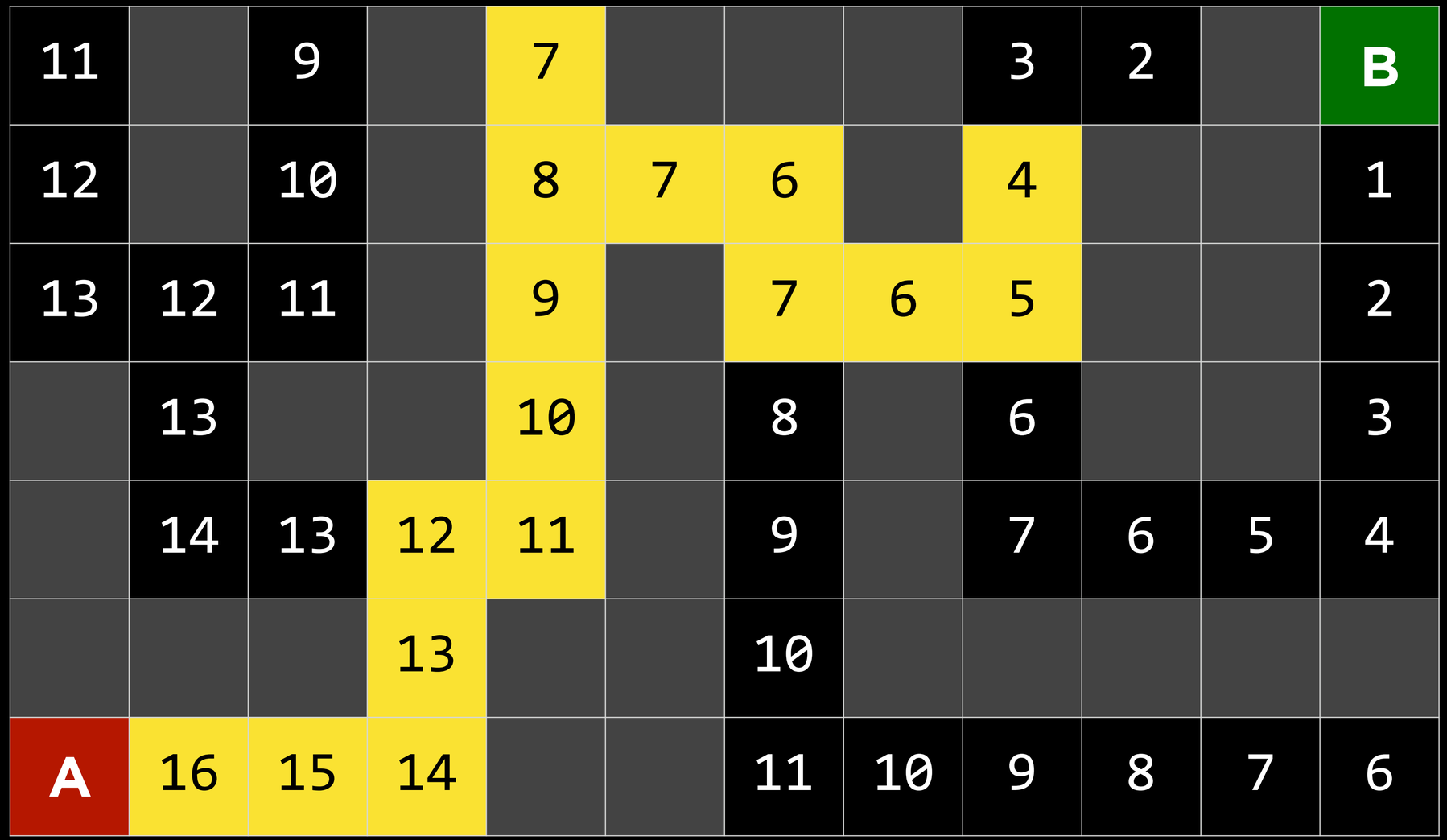
# Greedy Best-First Search



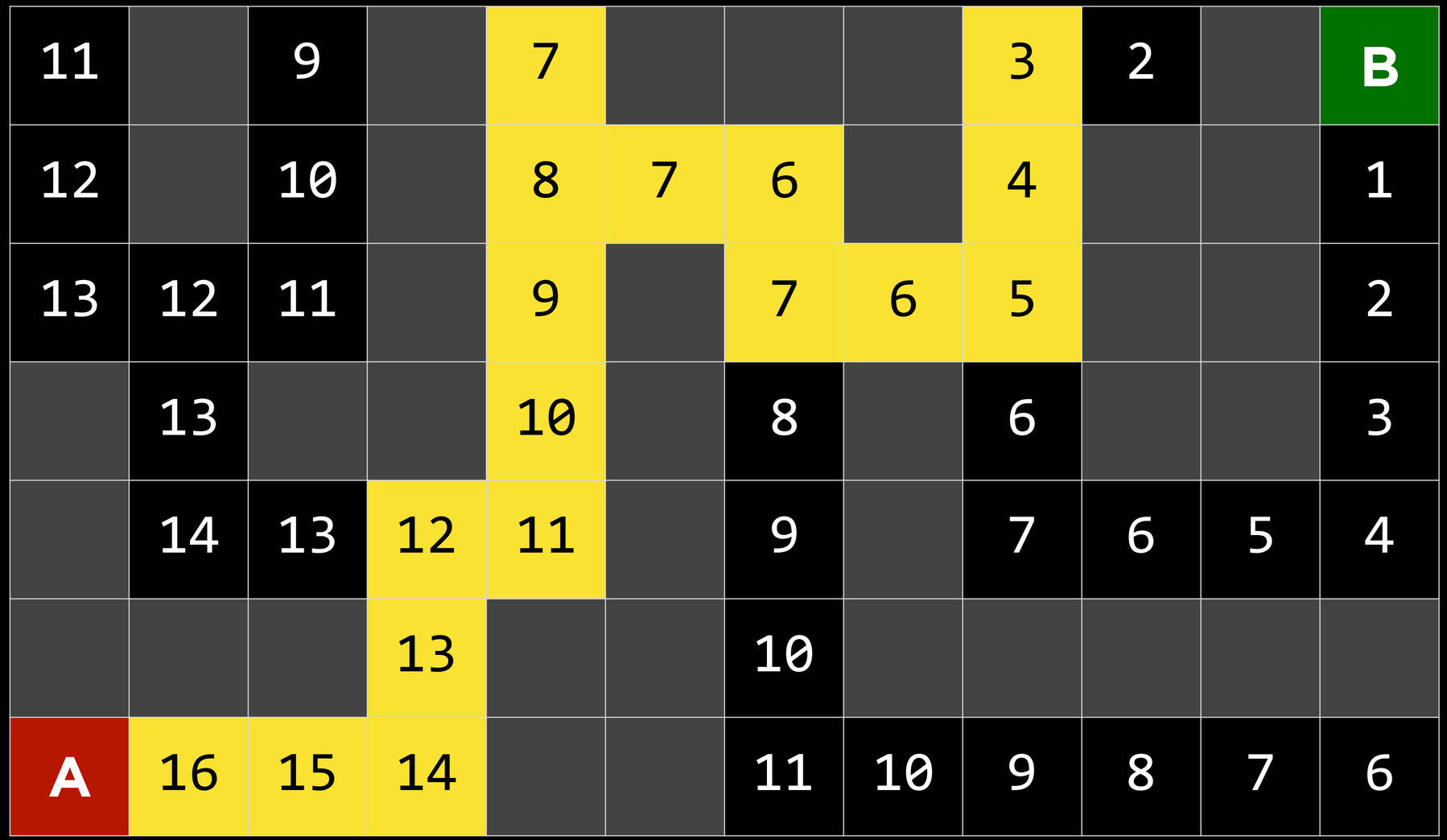
# Greedy Best-First Search



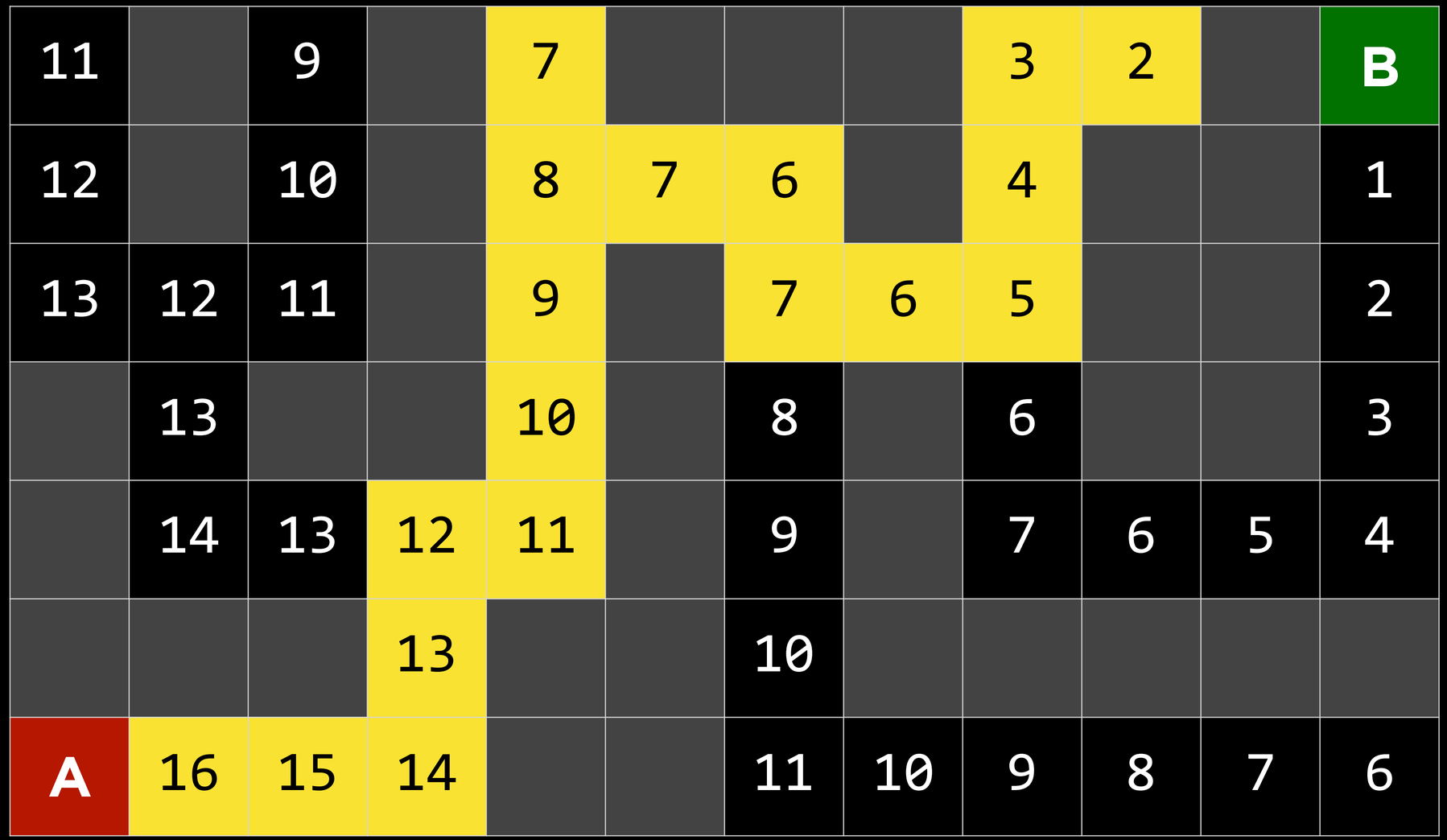
# Greedy Best-First Search



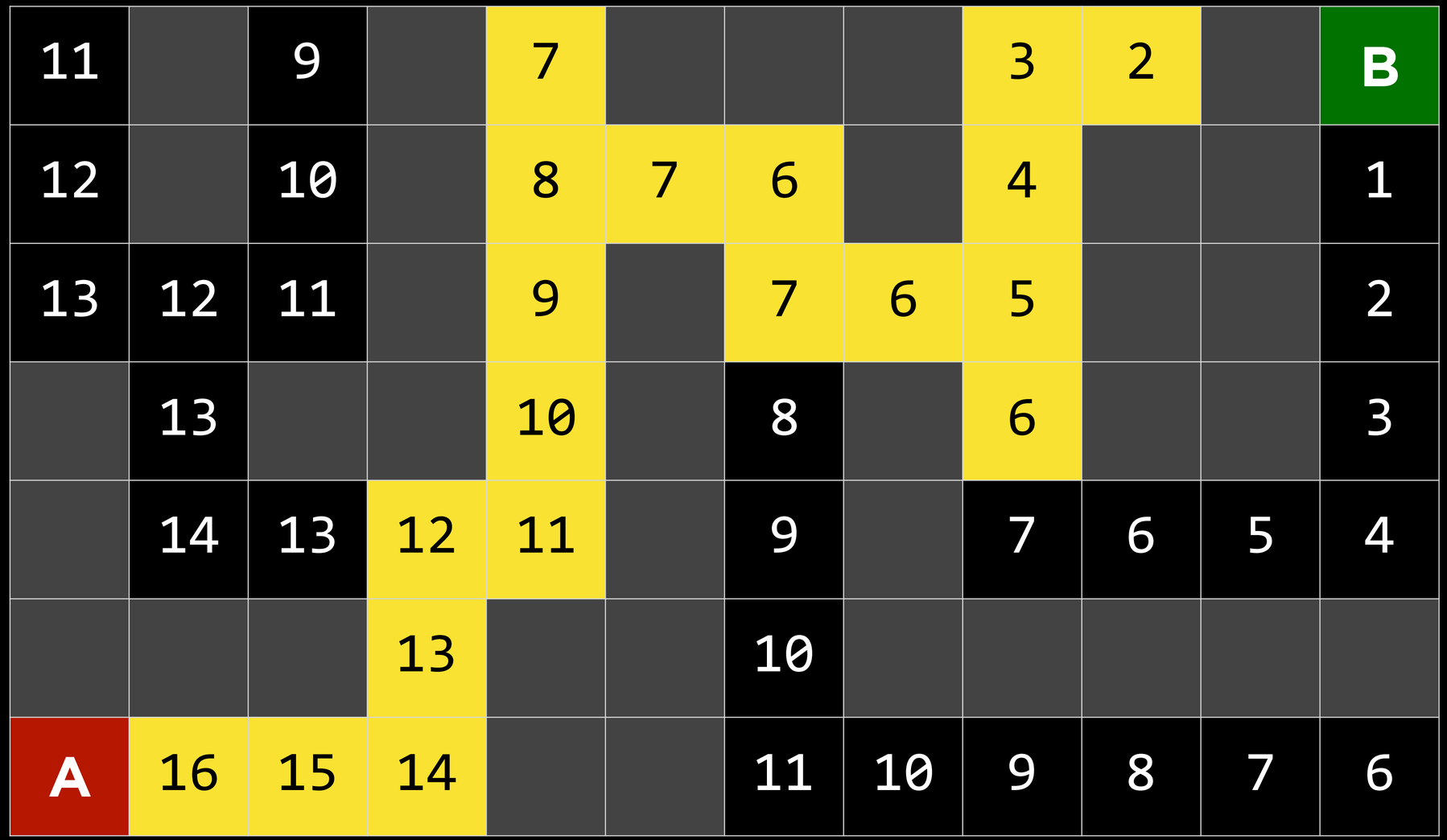
# Greedy Best-First Search



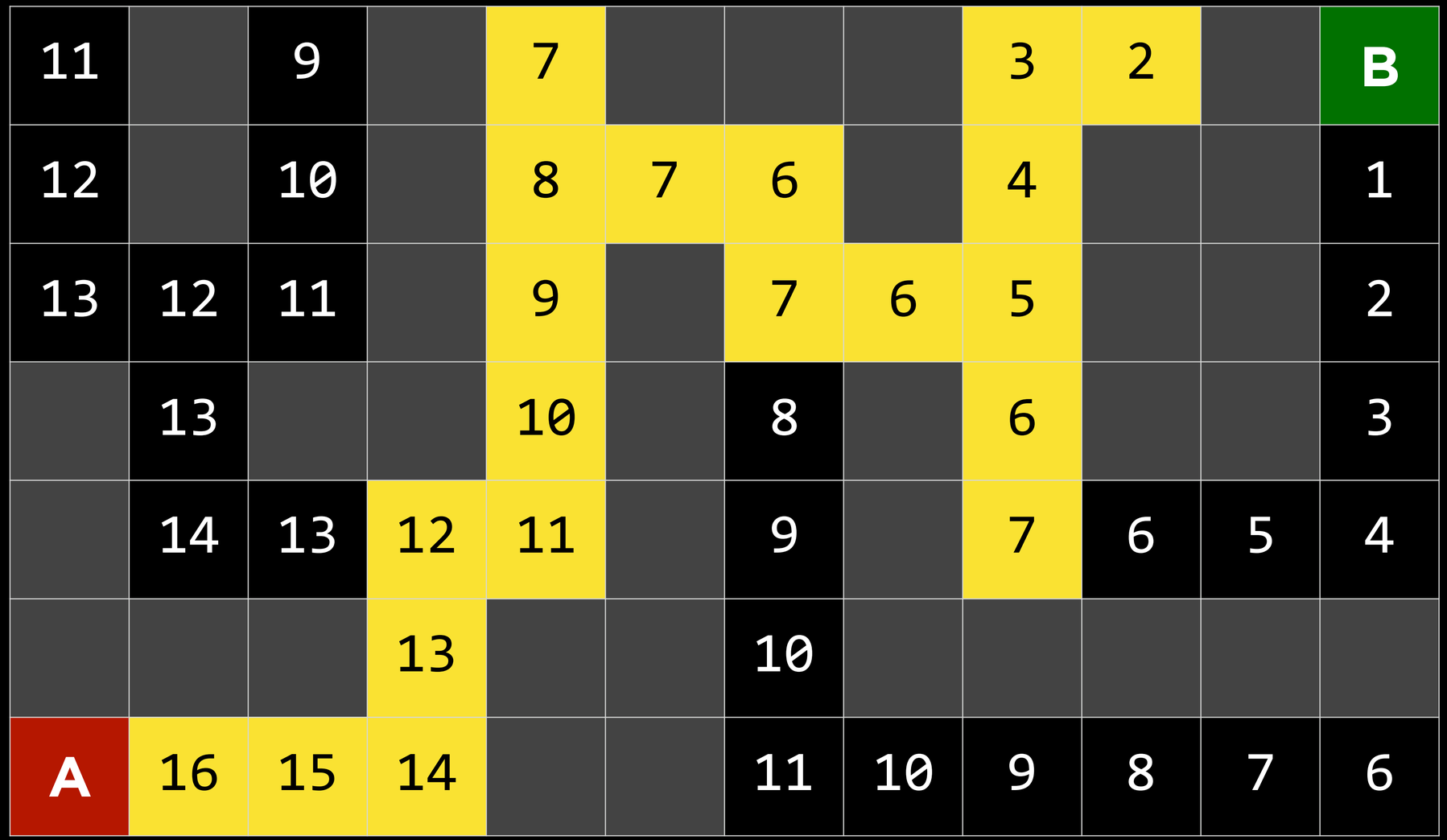
# Greedy Best-First Search



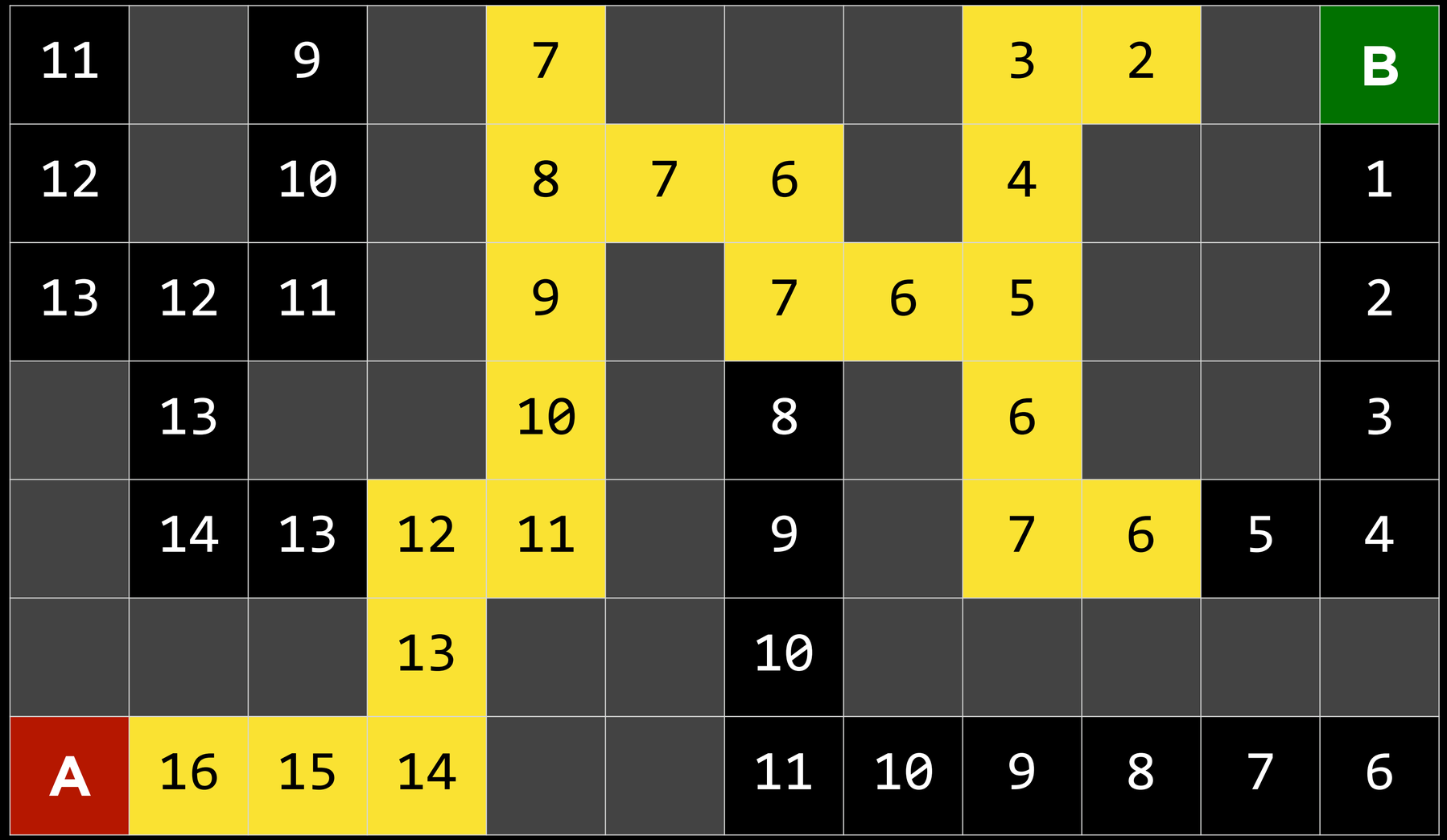
# Greedy Best-First Search



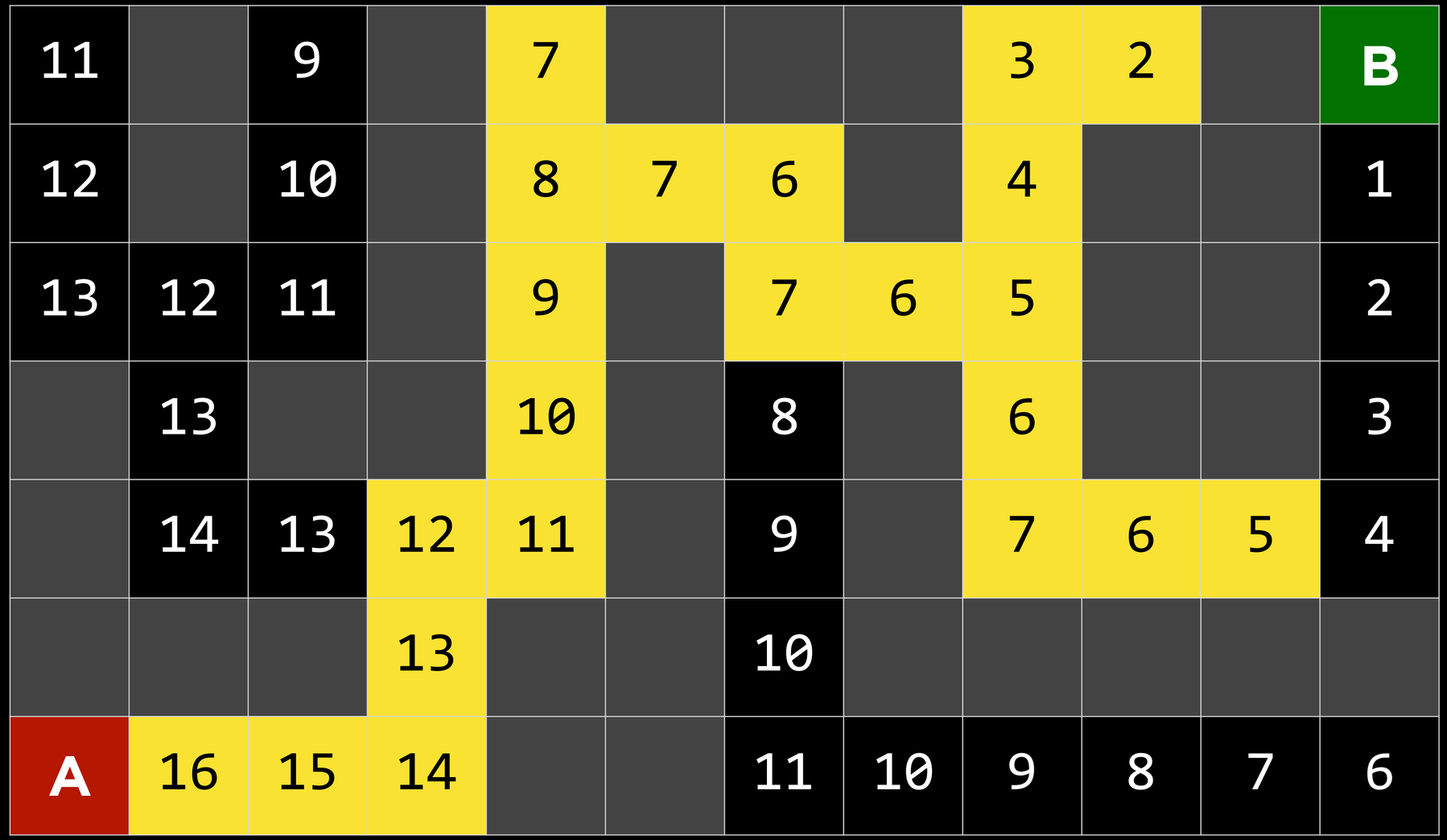
# Greedy Best-First Search



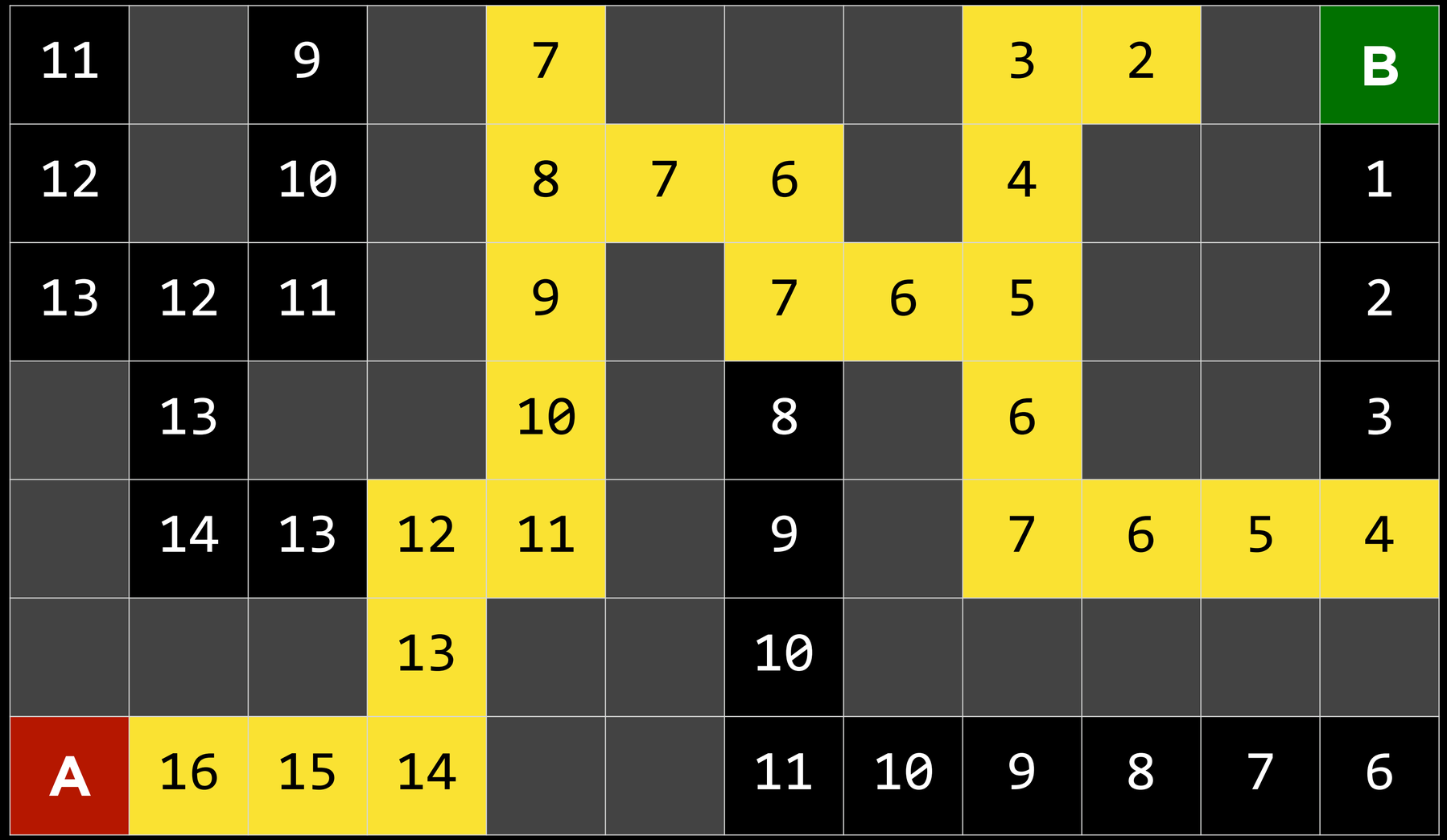
# Greedy Best-First Search



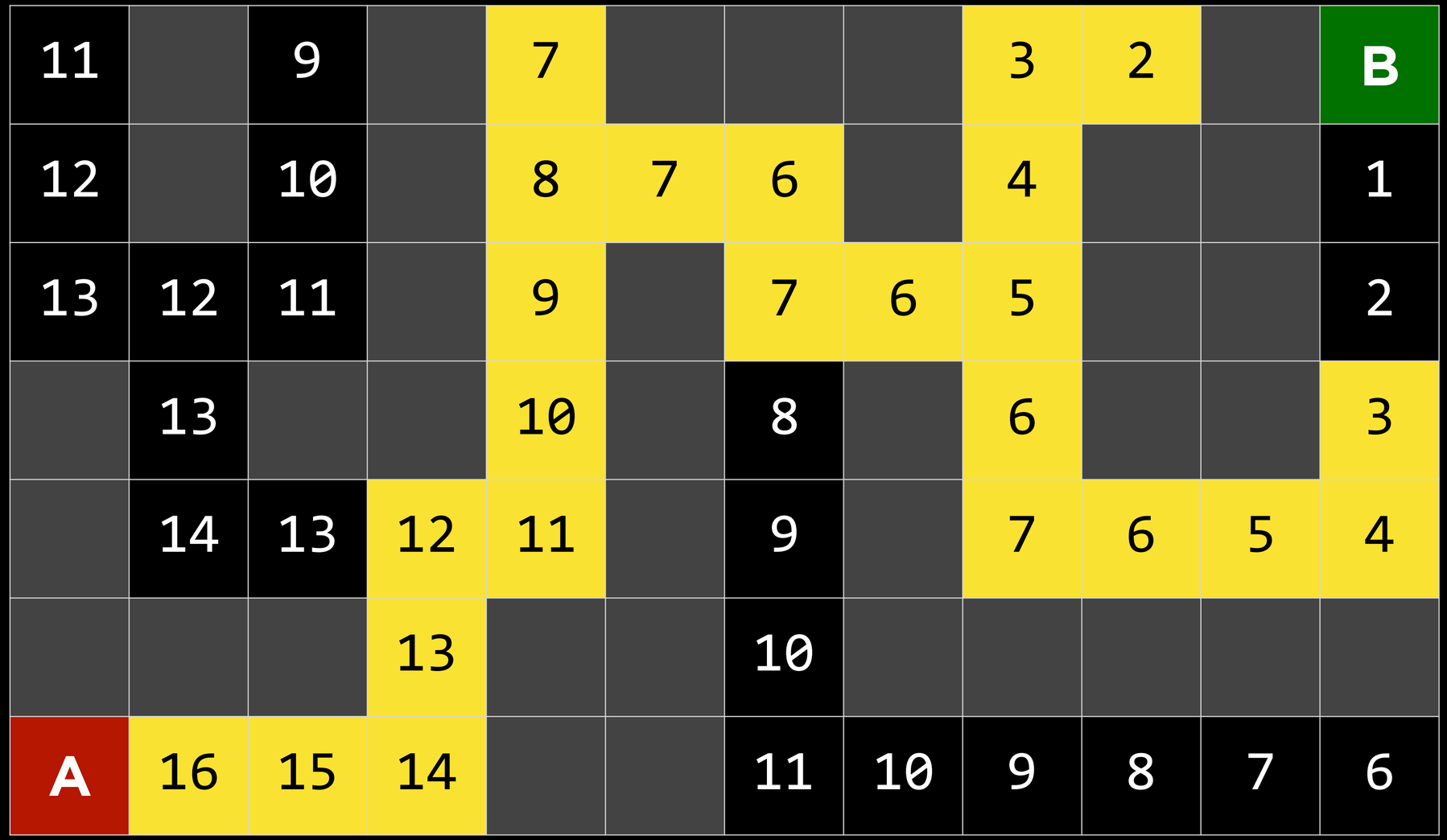
# Greedy Best-First Search



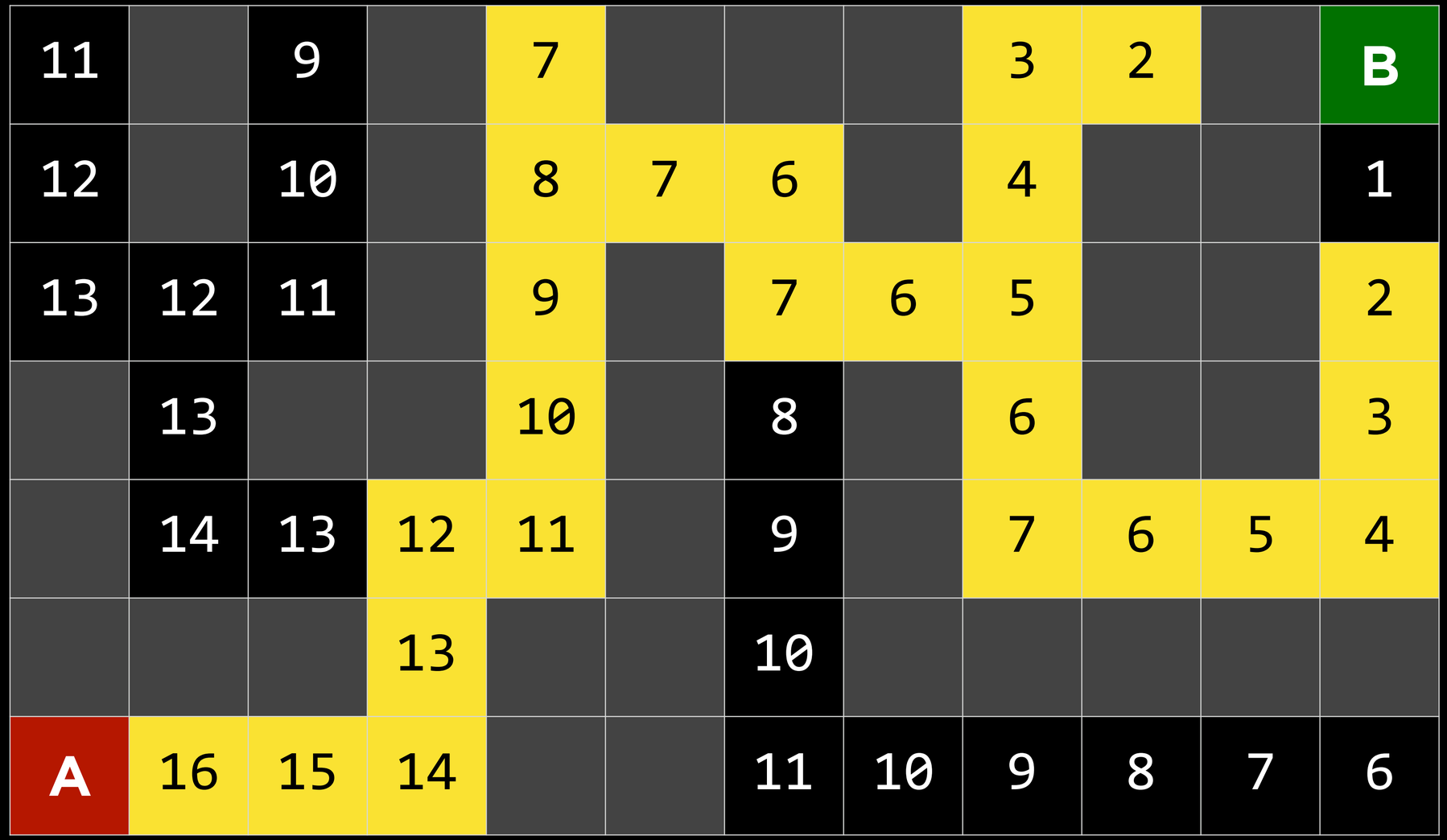
# Greedy Best-First Search



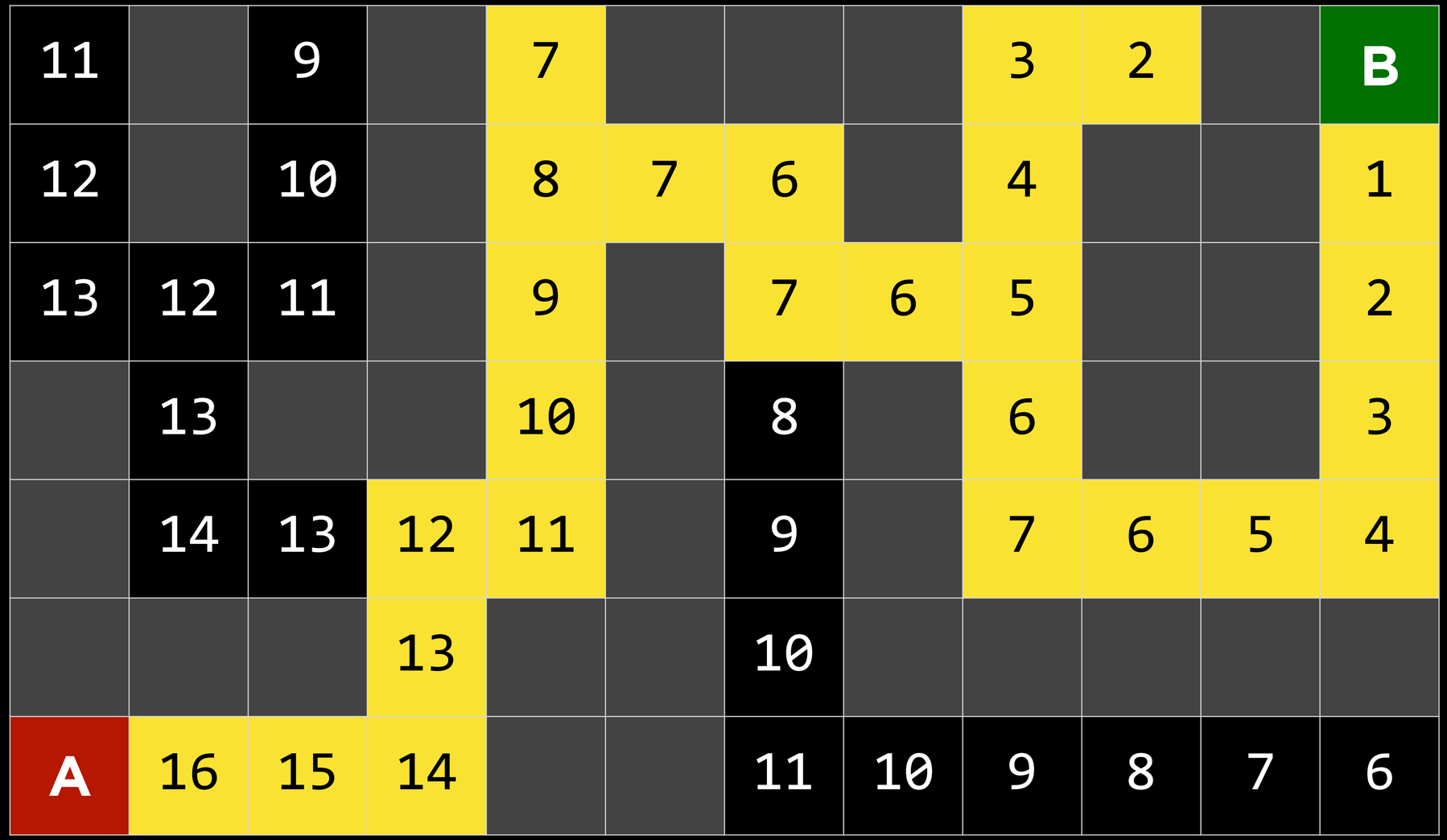
# Greedy Best-First Search



# Greedy Best-First Search



# Greedy Best-First Search



Will Greedy Best-First Search find the optimal solution?

# 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

6 steps needed to reach 13 is much less than steps needed to reach 12

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 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

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 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

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 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** | 1+16 | 15 | 14 | | 12 | 11 | 10 | 9 | 8 | 7 | 6 |

# 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** | 1+16 | 2+15 | 14 | | 12 | 11 | 10 | 9 | 8 | 7 | 6 |

# 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** | 1+16 | 2+15 | 3+14 | | 12 | 11 | 10 | 9 | 8 | 7 | 6 |

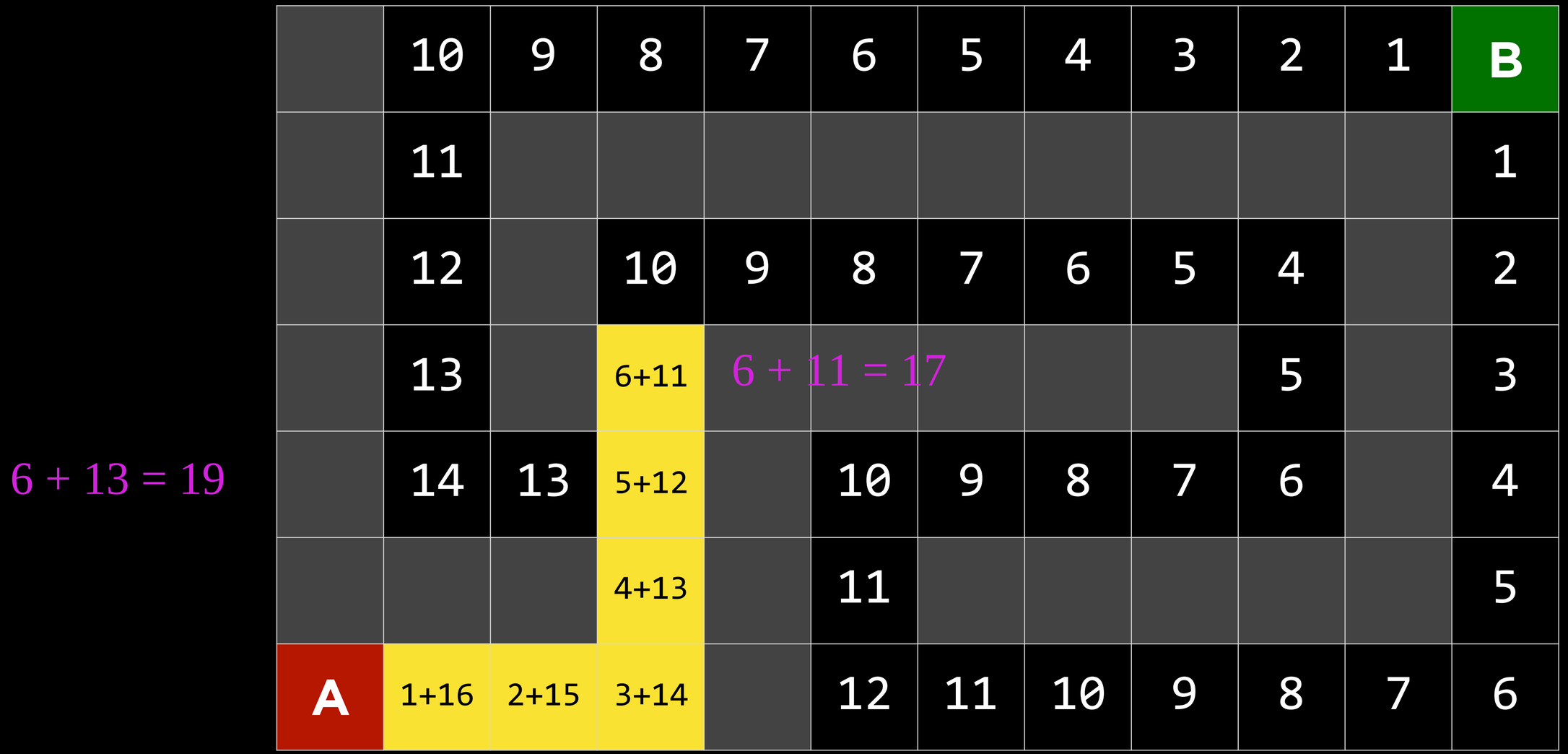
# A* Search

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 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 |
| | | | 4+13 | | 11 | | | | | | 5 |
| **A** | 1+16 | 2+15 | 3+14 | | 12 | 11 | 10 | 9 | 8 | 7 | 6 |

# 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 | 5+12 | | 10 | 9 | 8 | 7 | 6 | | 4 |
| | | | 4+13 | | 11 | | | | | | 5 |
| **A** | 1+16 | 2+15 | 3+14 | | 12 | 11 | 10 | 9 | 8 | 7 | 6 |

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

# A* Search

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

# A* Search

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

# A* Search

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

# A* Search

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

# A* Search

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

# A* Search

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

# A* Search

14 + 5 = 19

6 + 13 = 19

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

# A* Search

15 + 6 = 21

6 + 13 = 19

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

# A* Search

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

# A* Search

alternate between paths in favor of smaller sums
increasing Manhattan distance will make non-optimal path's sum too expensive

16 + 7 = 25

7 + 14 = 21

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | **B** |
| | 11 | | | | | | | | | | 1 |
| | 12 | | 7+10 | 8+9 | 9+8 | 10+7 | 11+6 | 12+5 | 13+4 | | 2 |
| | 13 | | 6+11 | | | | | | 14+5 | | 3 |
| | 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

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | **B** |
| | 11 | | | | | | | | | | 1 |
| | 12 | | 7+10 | 8+9 | 9+8 | 10+7 | 11+6 | 12+5 | 13+4 | | 2 |
| | 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

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | **B** |
| | 11 | | | | | | | | | | 1 |
| | 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

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | **B** |
| | 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

| | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 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

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 11+10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 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

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 11+10 | 12+9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 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

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 11+10 | 12+9 | 13+8 | 7 | 6 | 5 | 4 | 3 | 2 | 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

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 11+10 | 12+9 | 13+8 | 14+7 | 6 | 5 | 4 | 3 | 2 | 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

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 11+10 | 12+9 | 13+8 | 14+7 | 15+6 | 5 | 4 | 3 | 2 | 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

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 11+10 | 12+9 | 13+8 | 14+7 | 15+6 | 16+5 | 4 | 3 | 2 | 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

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 11+10 | 12+9 | 13+8 | 14+7 | 15+6 | 16+5 | 17+4 | 3 | 2 | 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

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 11+10 | 12+9 | 13+8 | 14+7 | 15+6 | 16+5 | 17+4 | 18+3 | 2 | 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

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 11+10 | 12+9 | 13+8 | 14+7 | 15+6 | 16+5 | 17+4 | 18+3 | 19+2 | 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

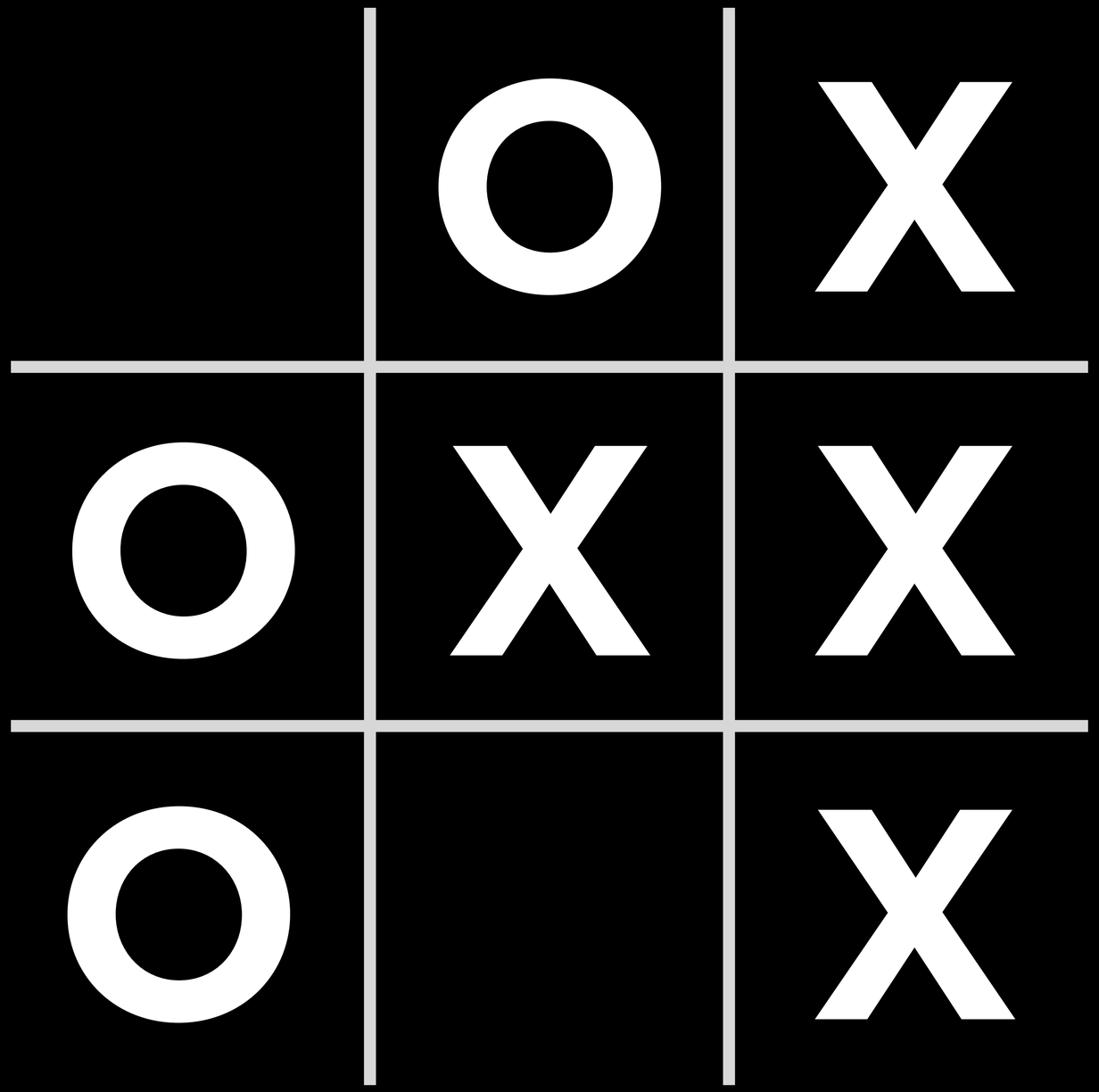
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 11+10 | 12+9 | 13+8 | 14+7 | 15+6 | 16+5 | 17+4 | 18+3 | 19+2 | 20+1 | B |
| | 10+11 | | | | | | | | | | 1 |
| | 9+12 | | 7+10 | 8+9 | 9+8 | 10+7 | 11+6 | 12+5 | 13+4 | | 2 |
| | 8+13 | | 6+11 | | | | | | 14+5 | | 3 |
| |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

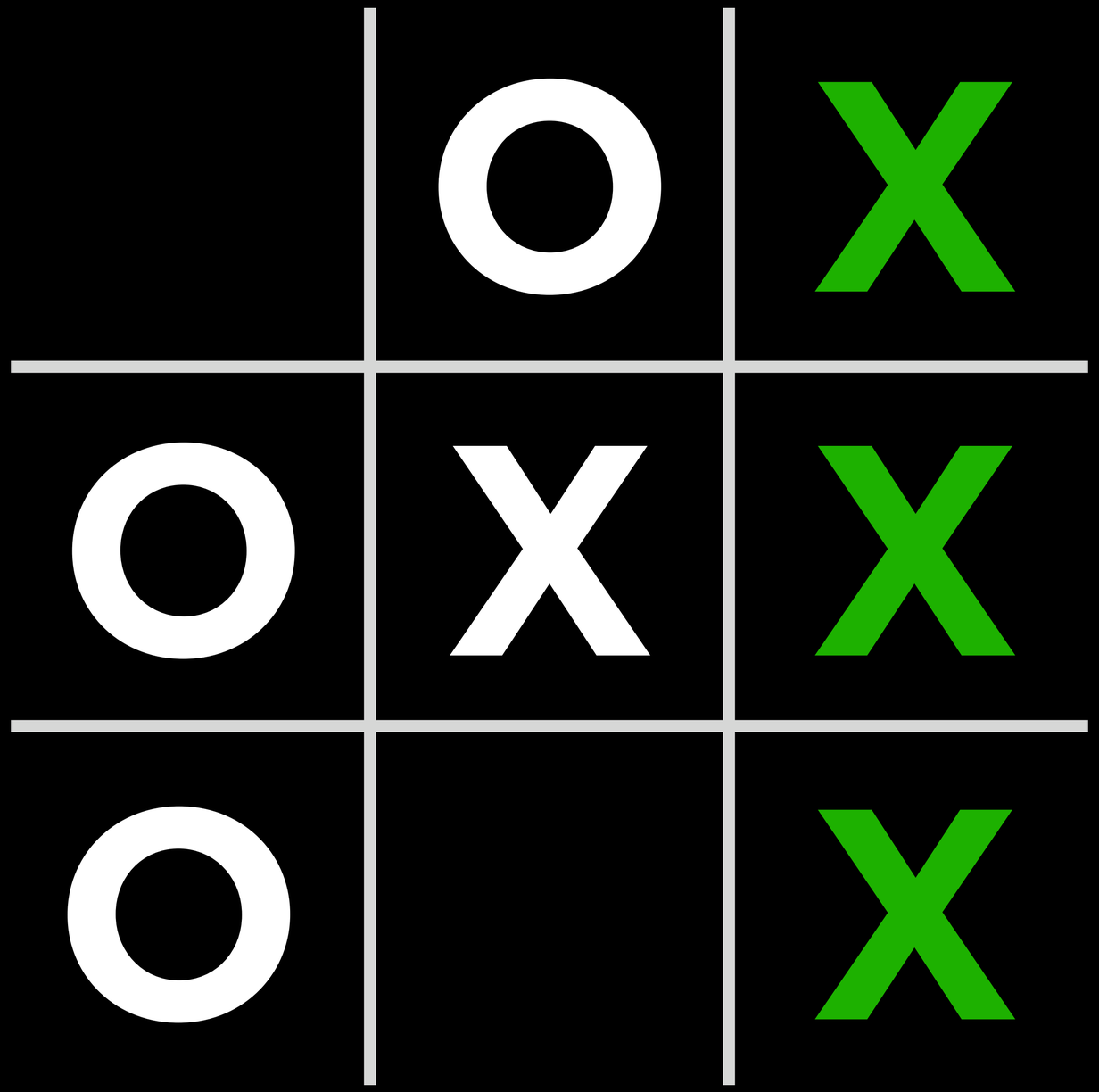
# A* search

optimal if

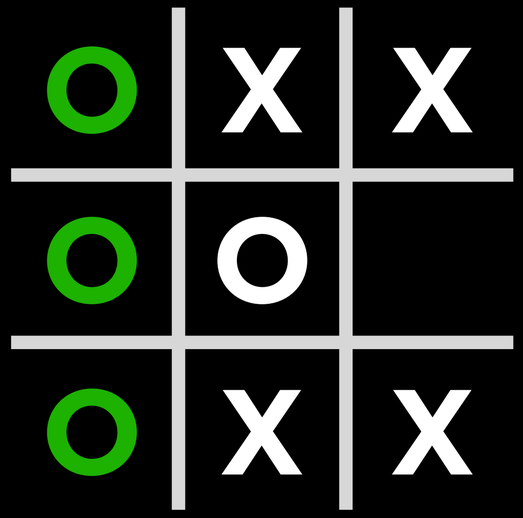
- $h(n)$ is admissible (never overestimates the true cost), and underestimates are acceptable
- $h(n)$ is consistent (for every node $n$ and successor $n'$ with step cost $c$, $h(n) \leq h(n') + c$)

# Adversarial Search

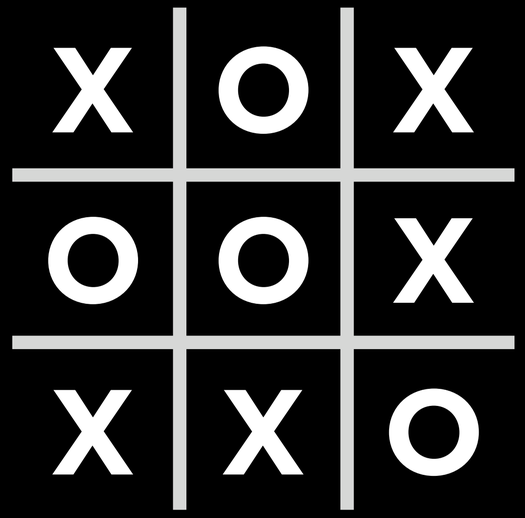

O X
O X X
O X


O X
O X X
O X

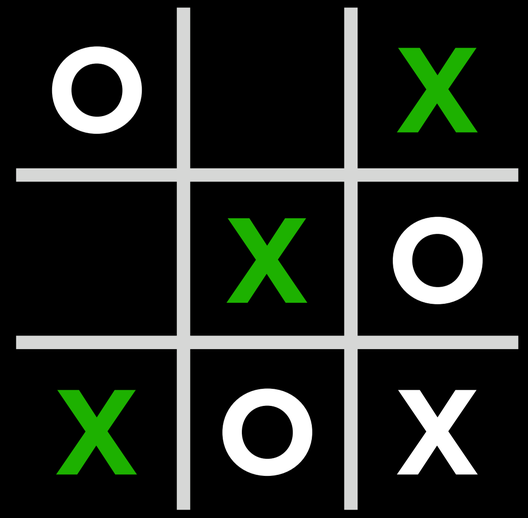
# Minimax




-1




0




1

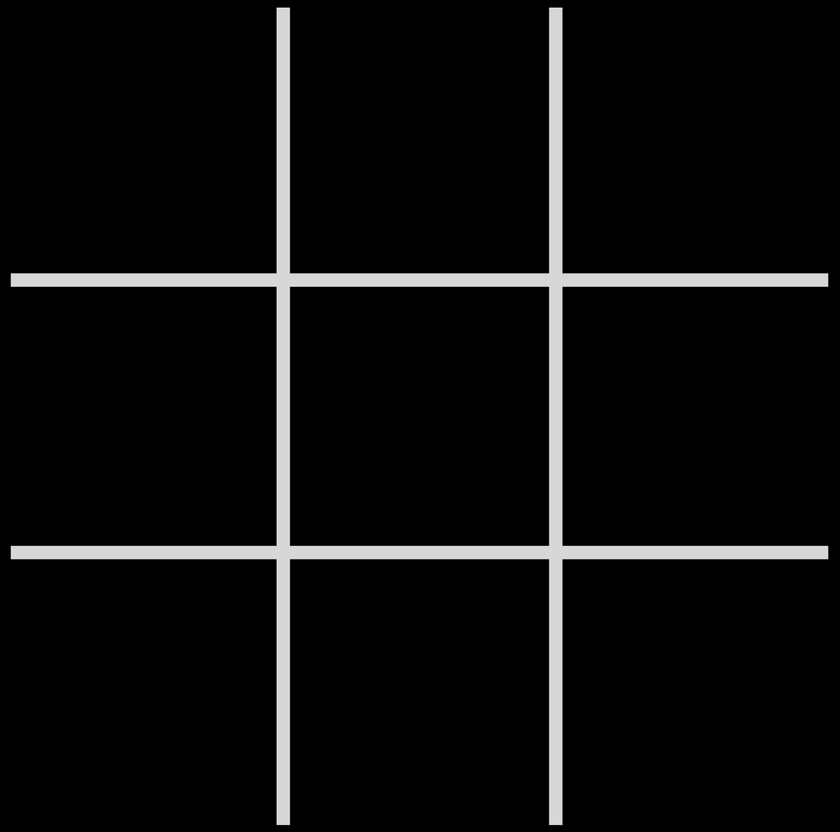
# Minimax

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

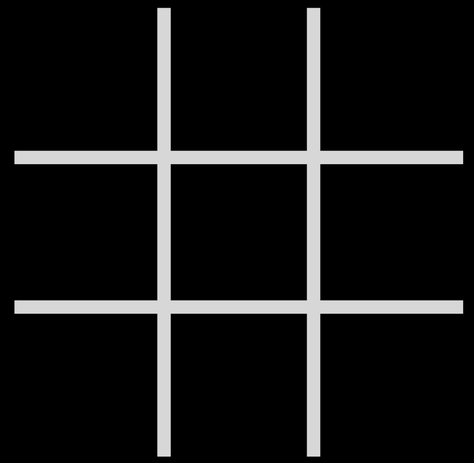
# Game

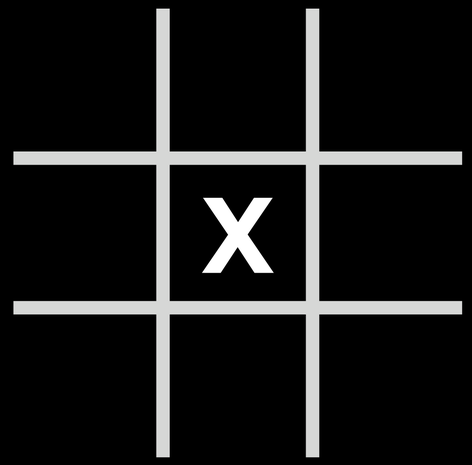
- $S_0$ : initial state
- $\textsc{Player}(s)$ : returns which player to move in state $s$
- $\textsc{Actions}(s)$ : returns legal moves in state $s$
- $\textsc{Result}(s, a)$ : returns state after action $a$ taken in state $s$
- $\textsc{Terminal}(s)$ : checks if state $s$ is a terminal state
- $\textsc{Utility}(s)$ : final numerical value for terminal state $s$

# Initial State



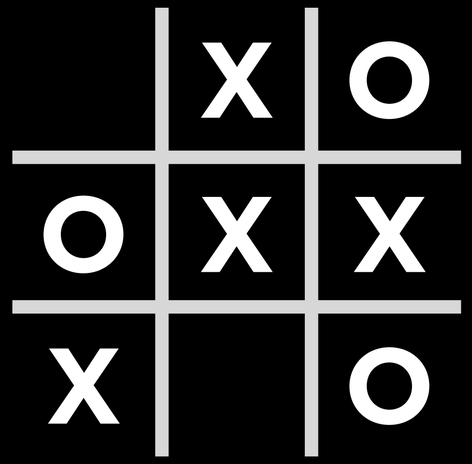
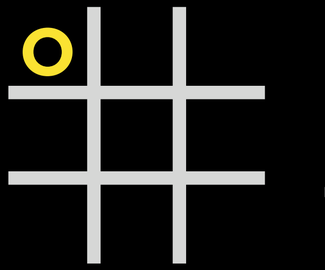
PLAYER(*s*)

PLAYER(  ) = X

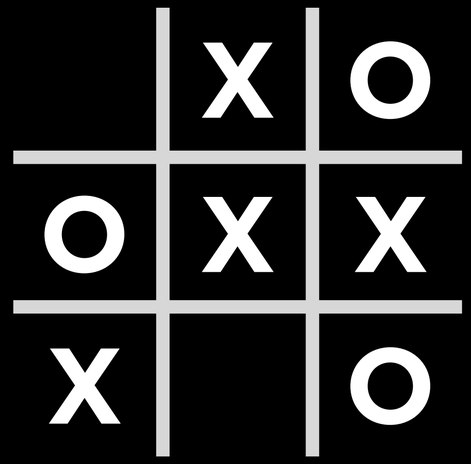
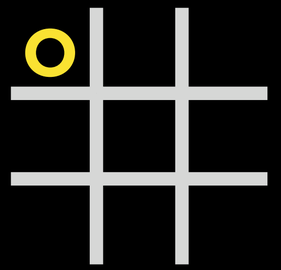
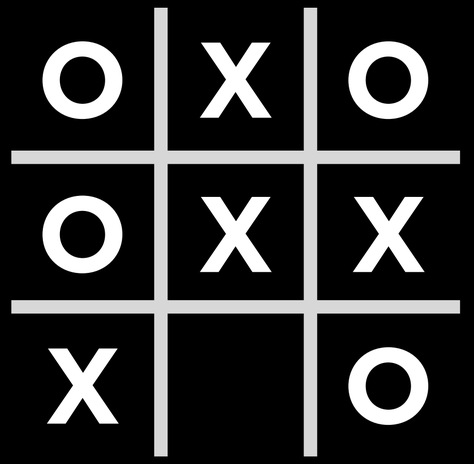
PLAYER( 

 ) = O

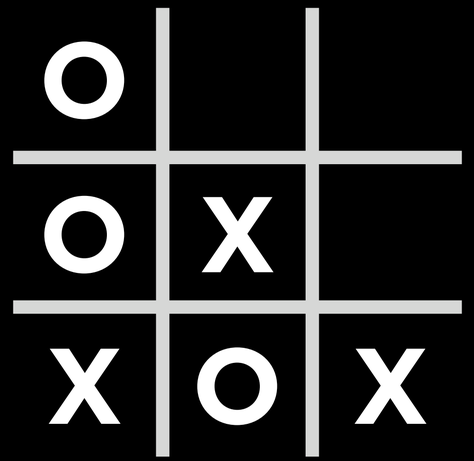
# Actions(*s*)

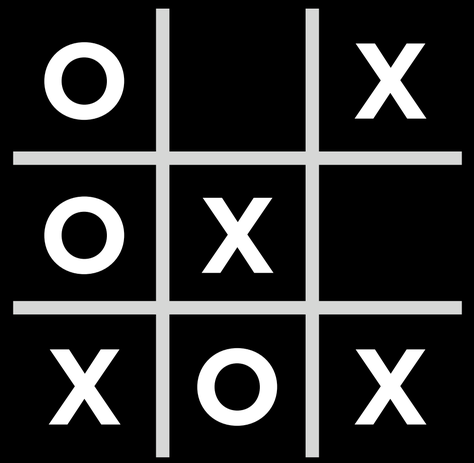
Actions(  ) = {  ,  }

# RESULT($s$, $a$)

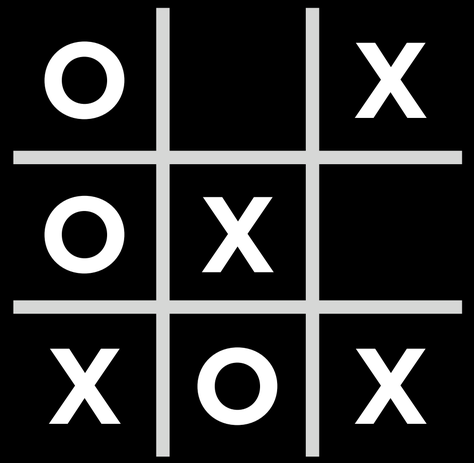
RESULT(  ,  ) = 

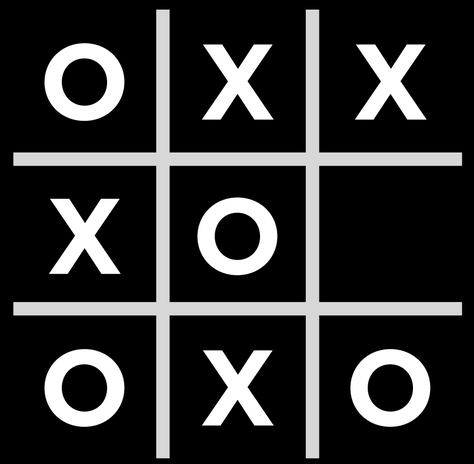
TERMINAL($s$)

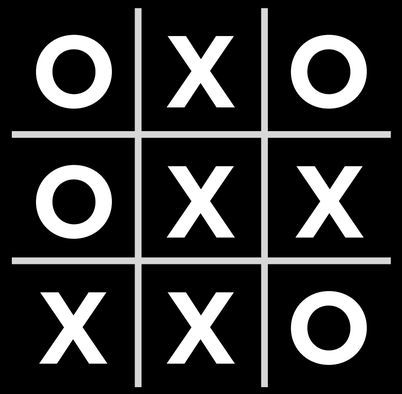
TERMINAL(  ) = `false`

TERMINAL(  ) = `true`

Utility($s$)

Utility(  ) = 1

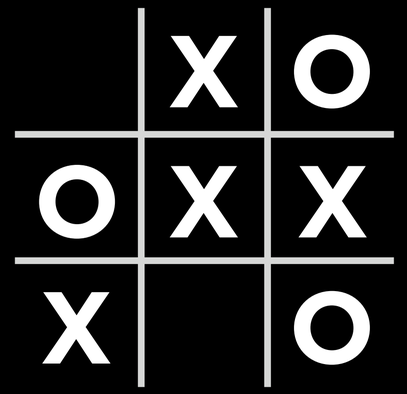
Utility(  ) = -1

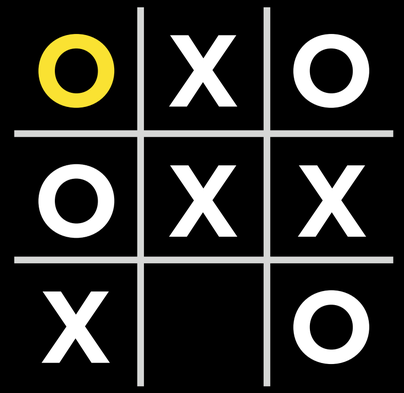



VALUE: 1

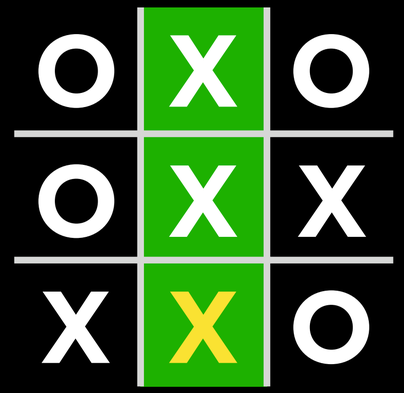
PLAYER($s$) = ○

MIN-VALUE: 0





MAX-VALUE: 1



VALUE: 1

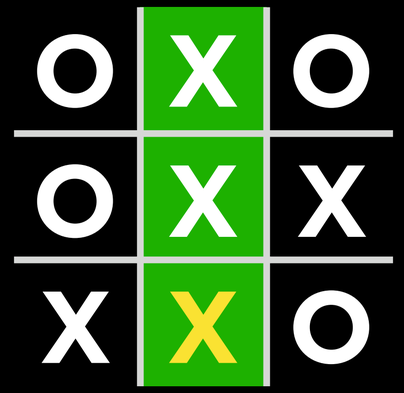
MAX-VALUE: 0

VALUE: 0

PLAYER($s$) = ○

MIN-VALUE: 0

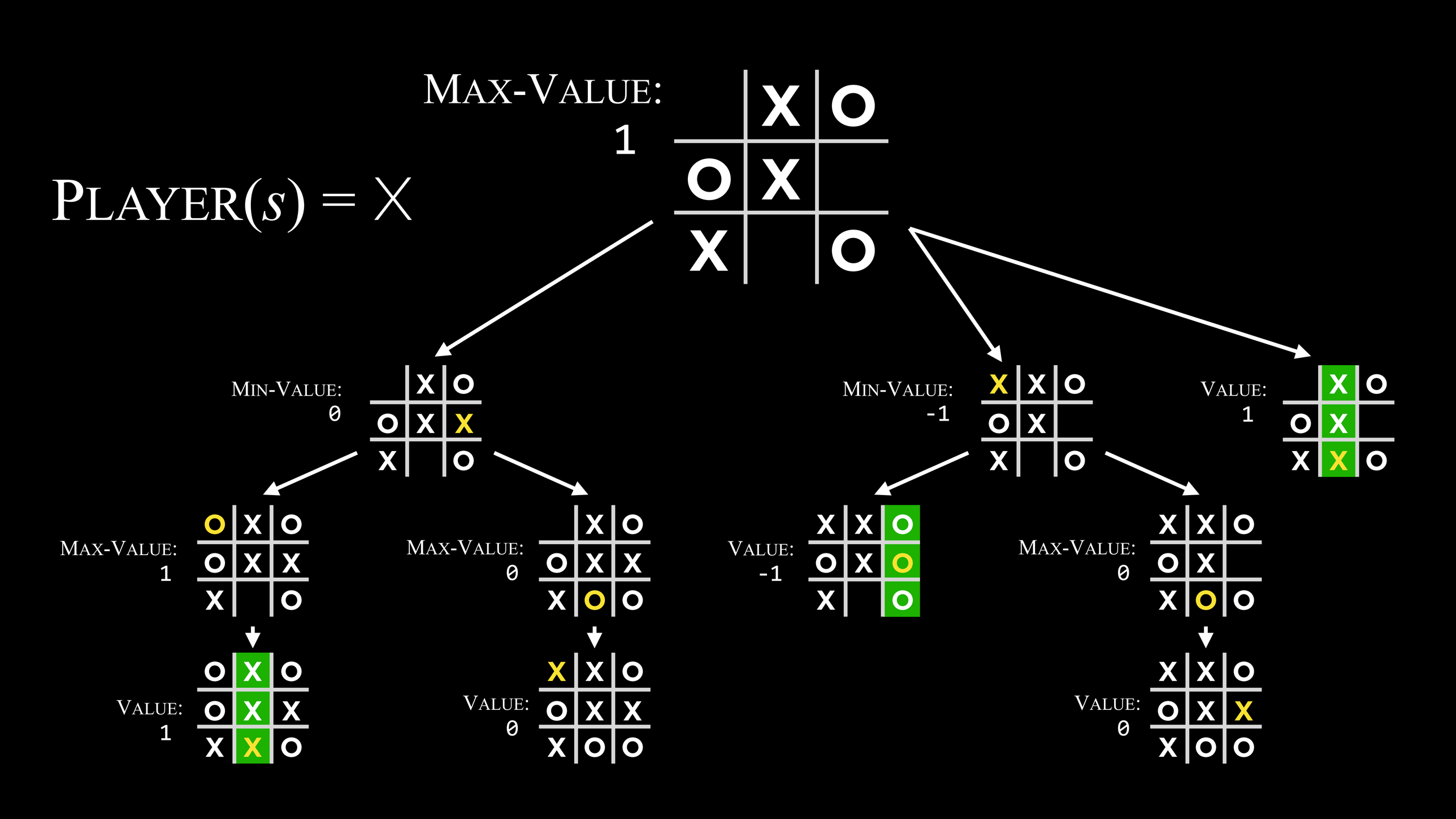
MAX-VALUE: 1

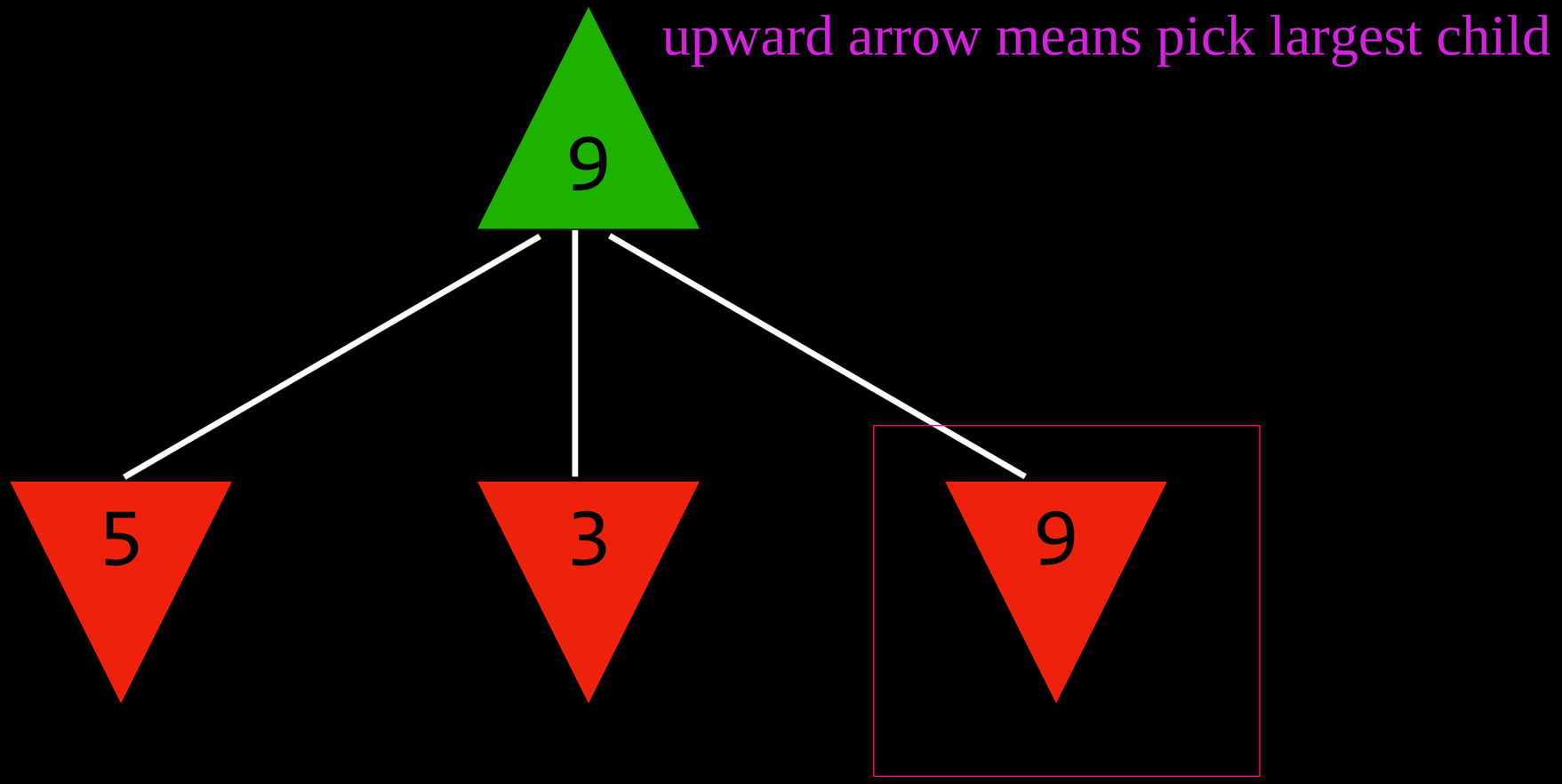


VALUE: 1

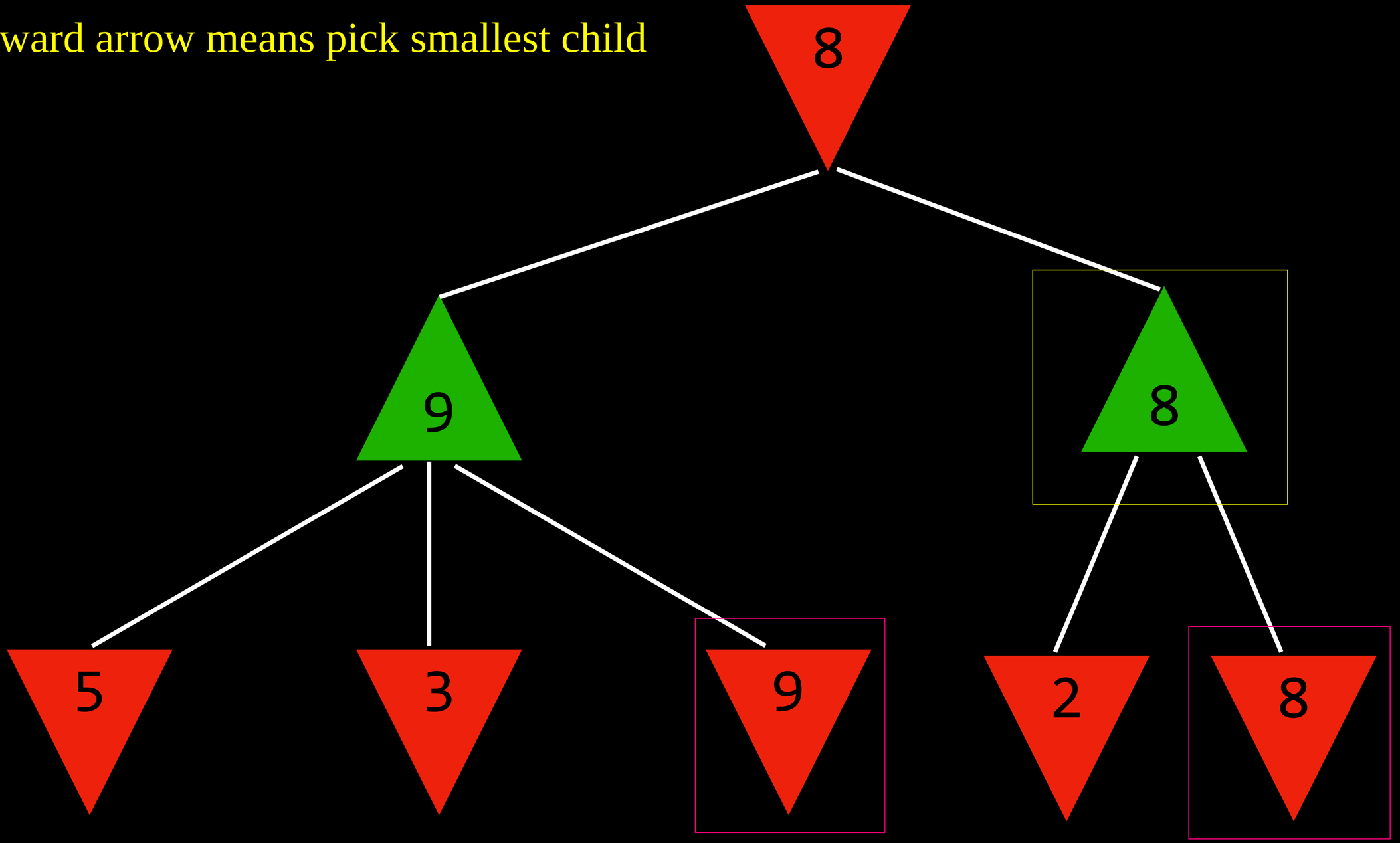
MAX-VALUE: 0

VALUE: 0


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


upward arrow means pick largest child
9
5
3
9

downward arrow means pick smallest child



# 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 $\text{MAX-VALUE}(\text{RESULT}(s, a))$

assuming opponent will choose action to maximize RESULT

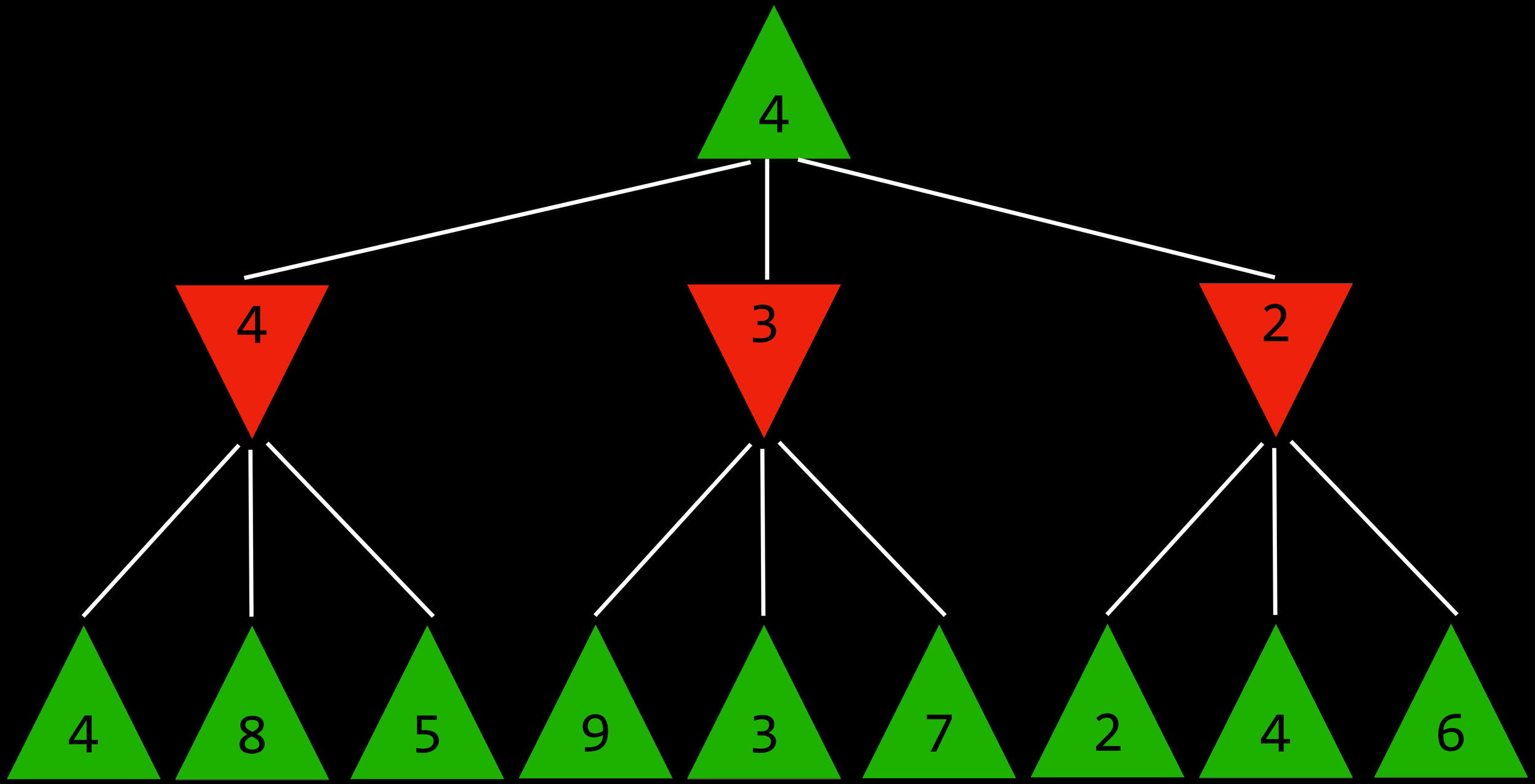
# Minimax

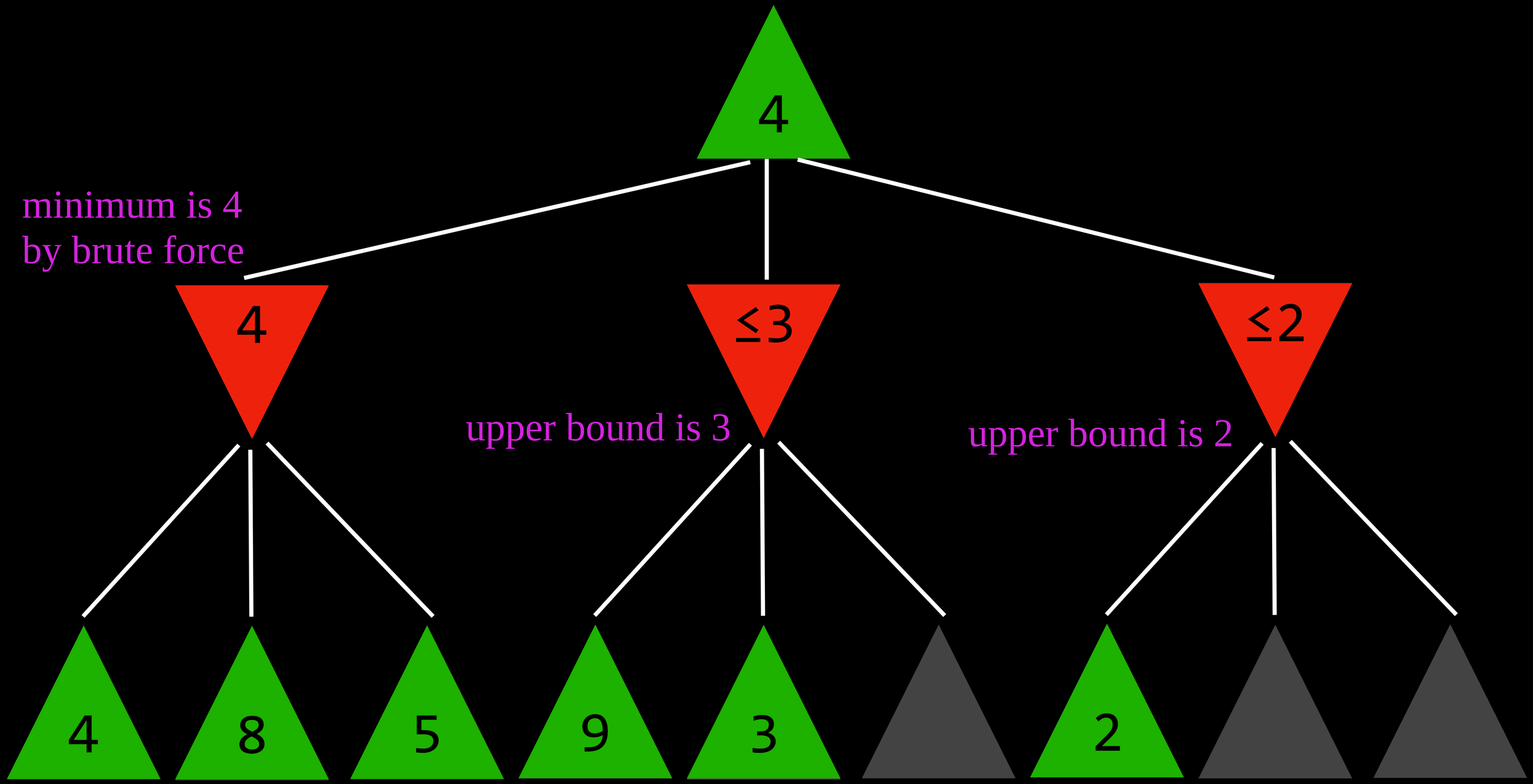
```
function MAX-VALUE(state):
    if TERMINAL(state):
        return UTILITY(state)
    v = -∞  initialize Value as low as possible
    for action in ACTIONS(state):
        v = MAX(v, MIN-VALUE(RESULT(state, action)))
    return v
```

# 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
minimum is 4
by brute force
4
≤3
≤2
upper bound is 3
upper bound is 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)

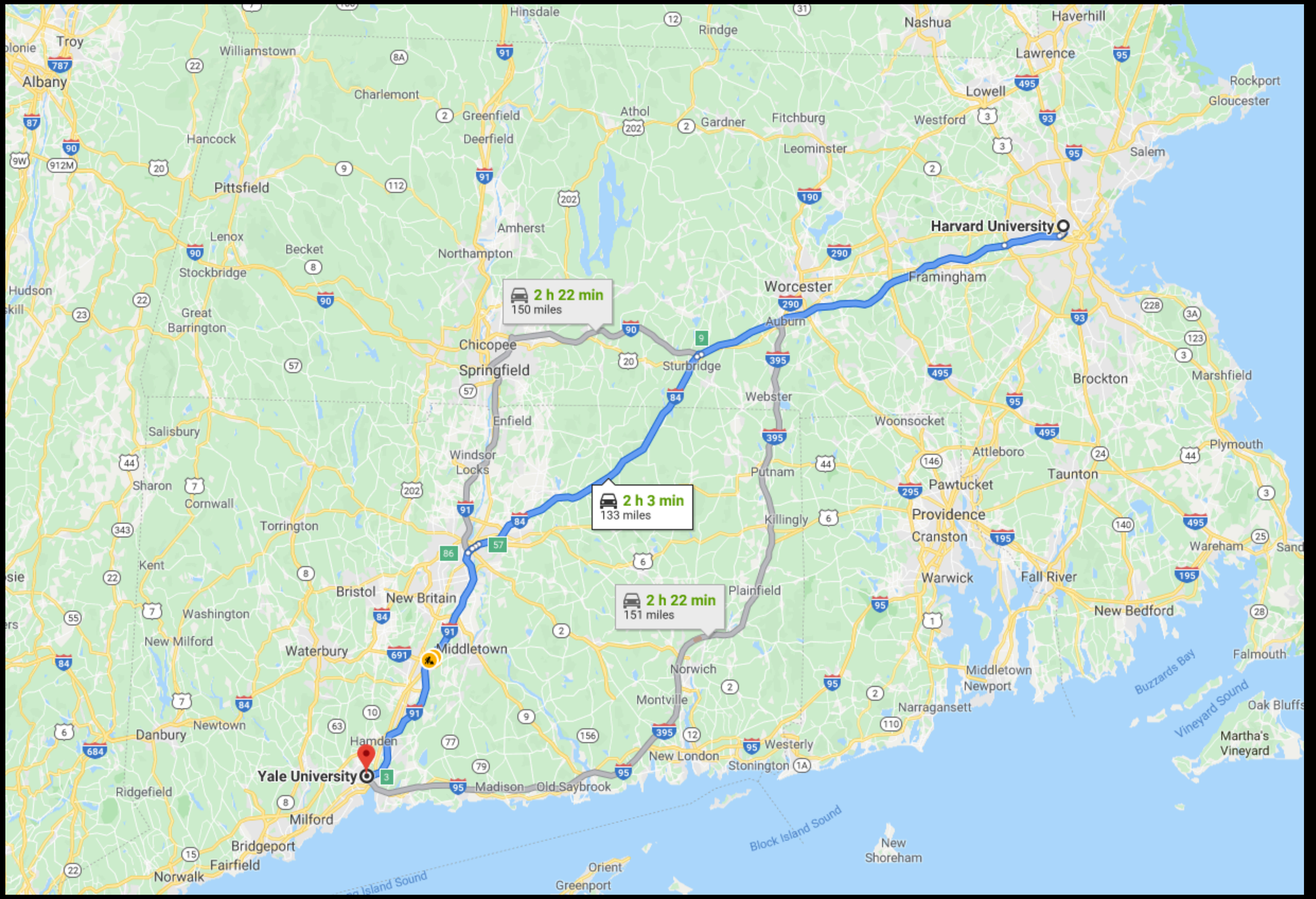
# Depth-Limited Minimax

we limit the depth of minimax and choose best outcome for that depth
we've considered best outcome if game over,
but if game is not over, we'll need an evaluation function

# evaluation function

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

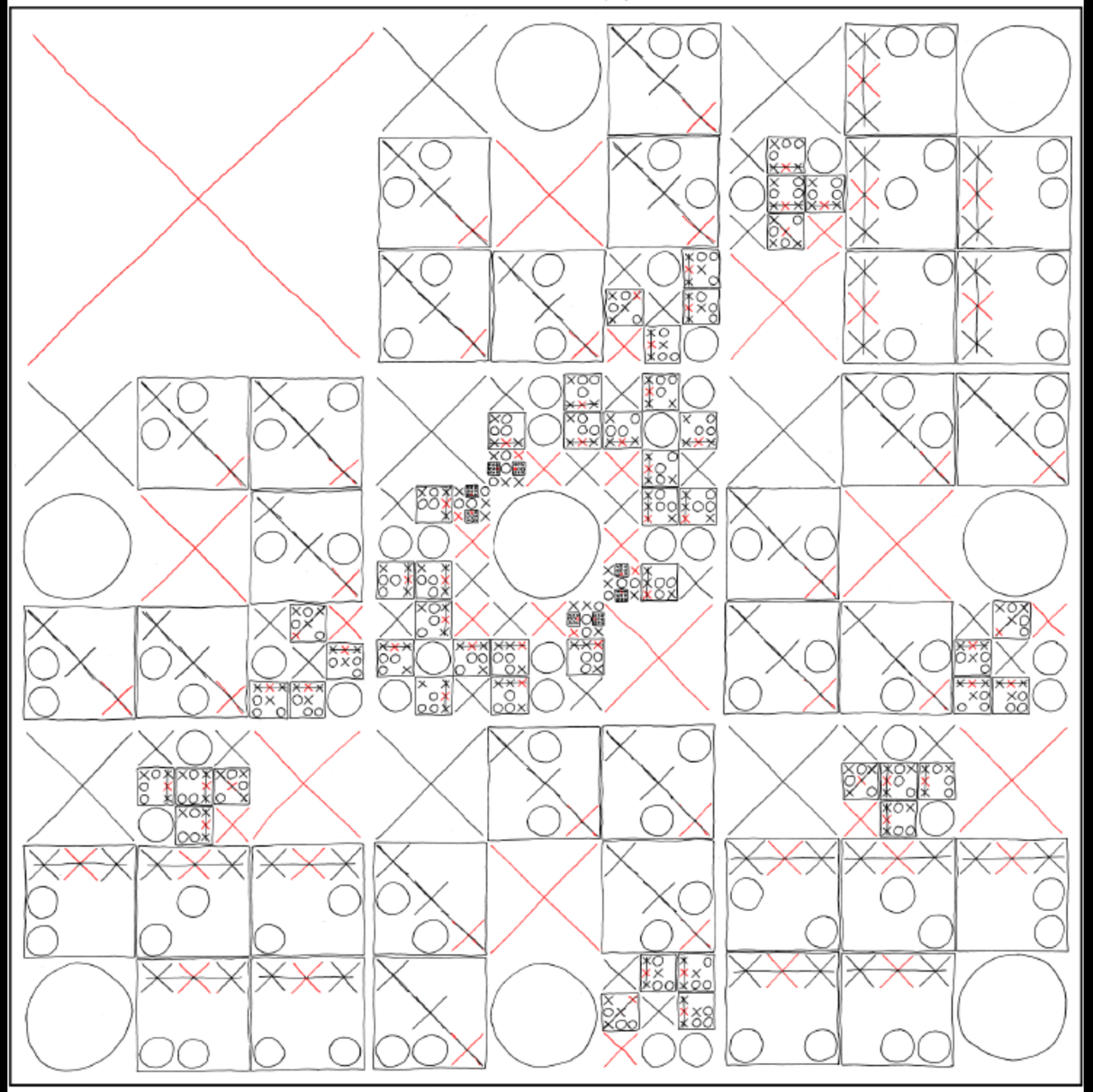
we need a function to estimate probability of winning [-1,1] for any state
where 1 is max player, -1 is min player


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

# 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/

# Search

# Introduction to **Artificial Intelligence** with Python