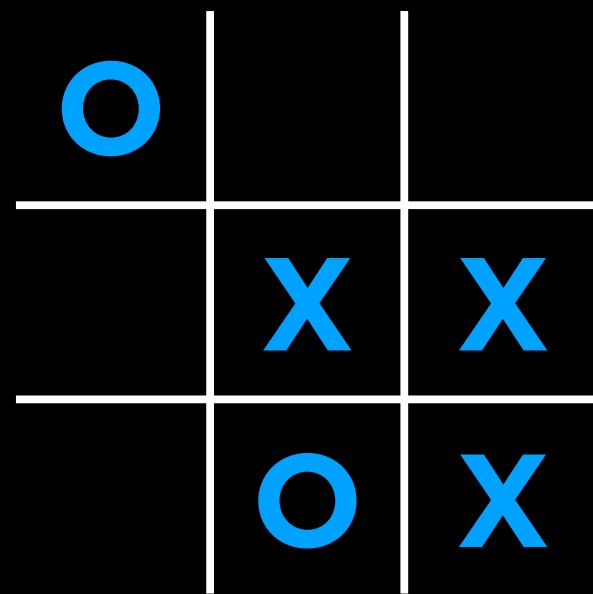


Introduction to  
**Artificial Intelligence**  
with Python

# Artificial Intelligence

Search

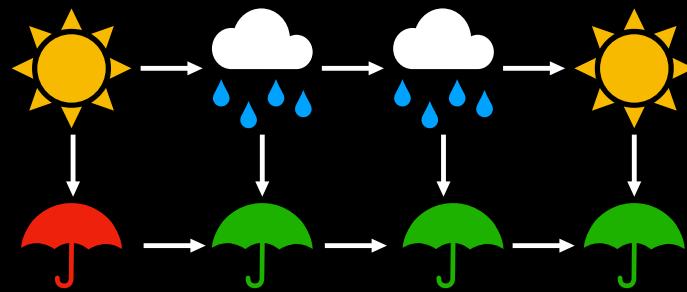


Knowledge

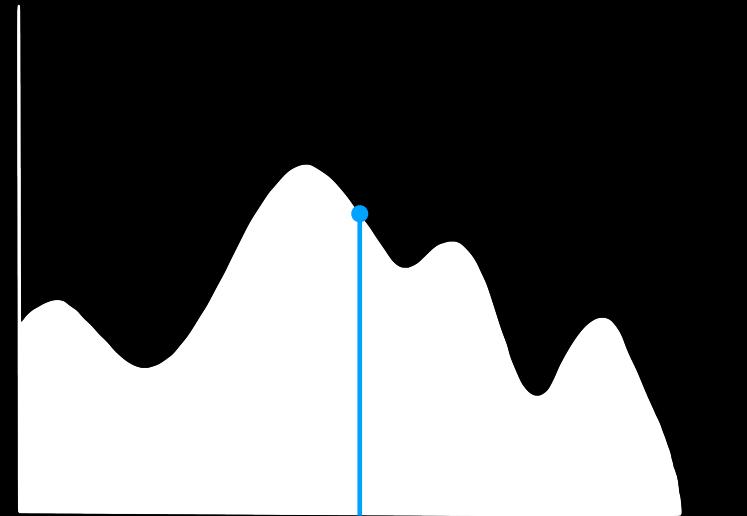
$$P \rightarrow Q$$

$$\begin{array}{c} P \\ \hline Q \end{array}$$

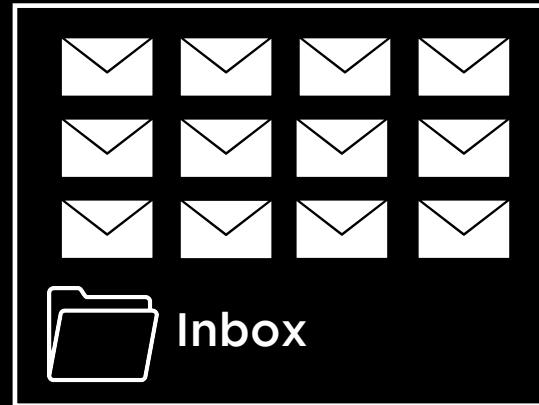
# Uncertainty



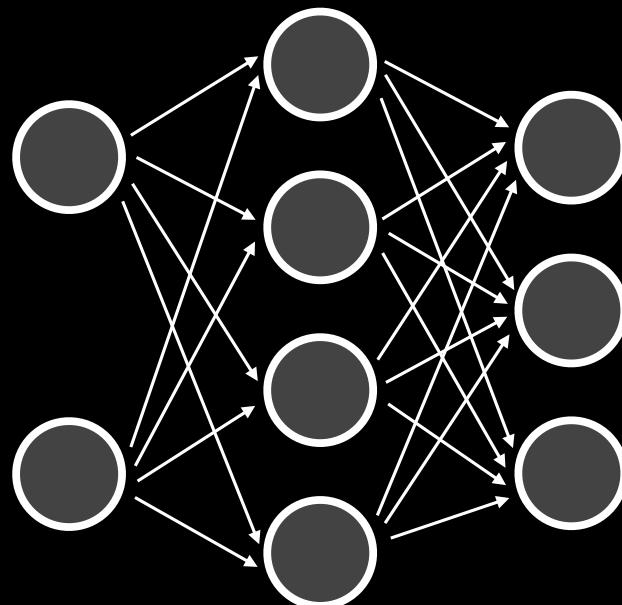
# Optimization



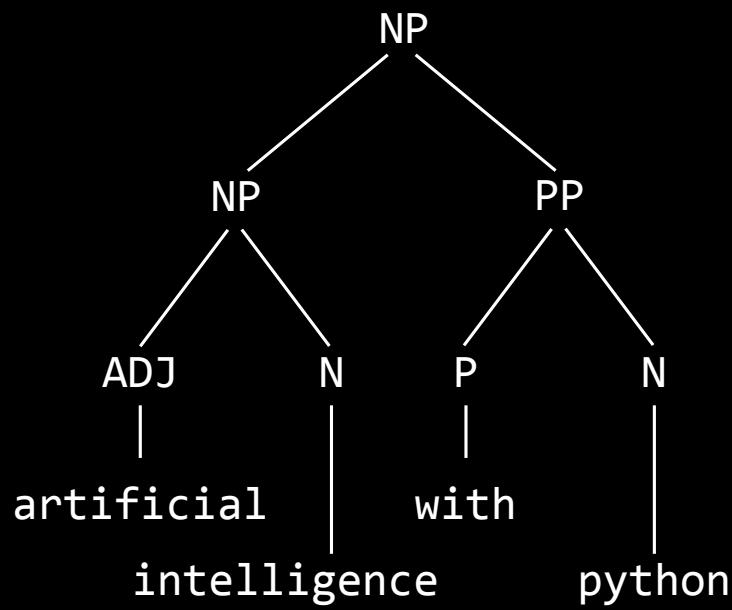
# Learning



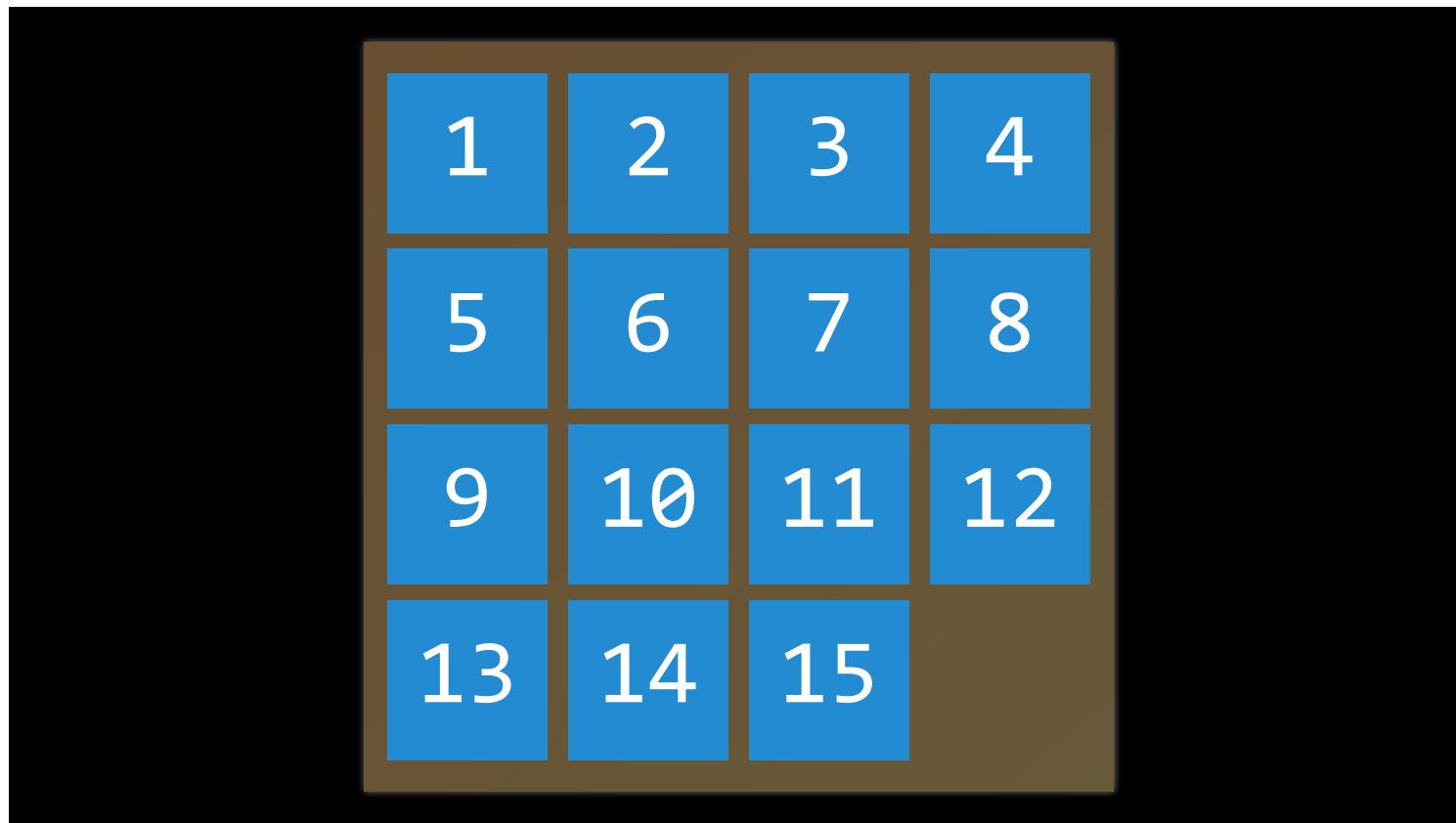
# Neural Networks

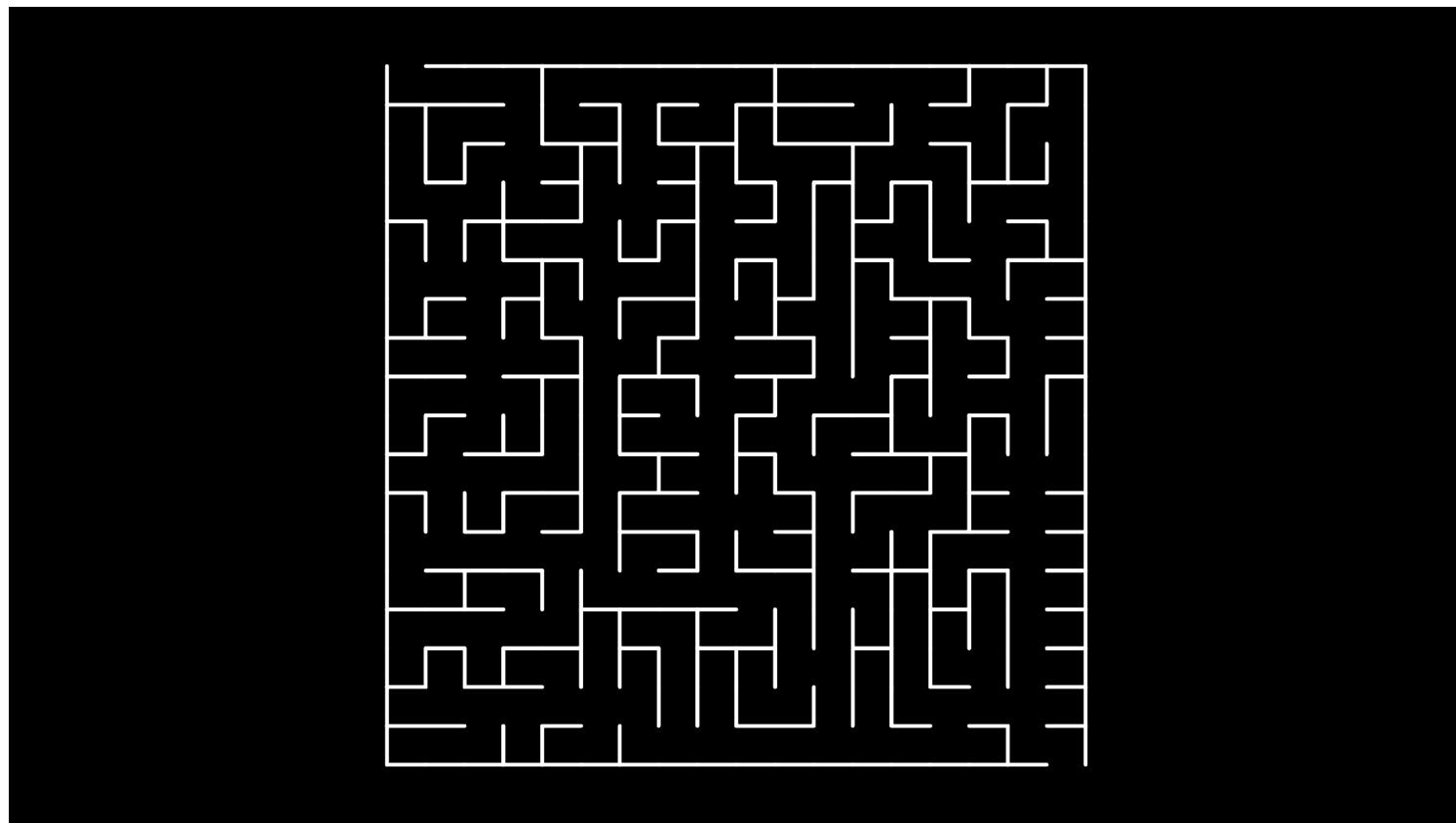


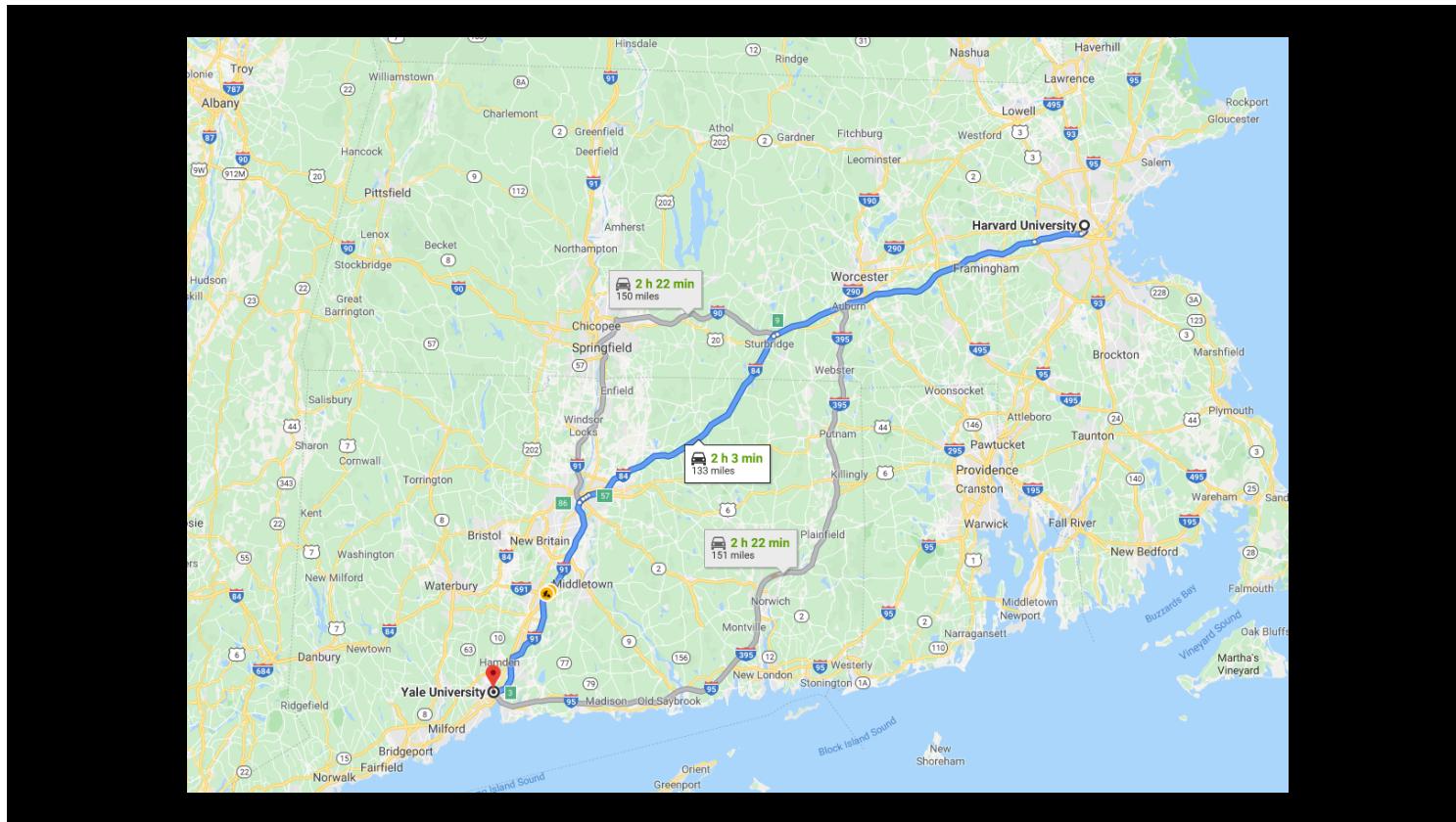
# Language



# Search







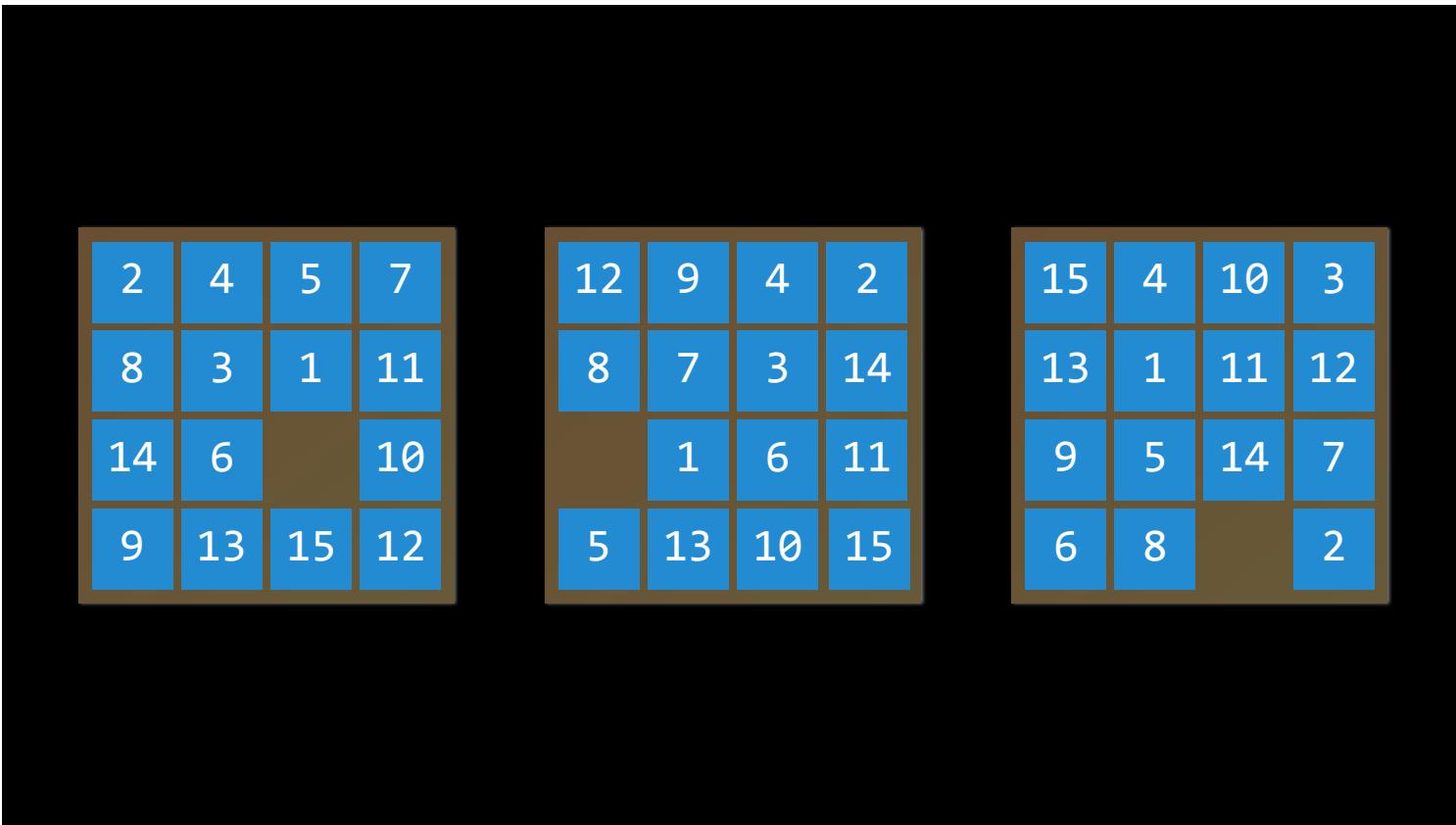
# Search Problems

# **agent**

entity that perceives its environment  
and acts upon that environment

# **state**

a configuration of the agent and  
its environment



# **initial state**

the state in which the agent begins

**initial state**

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

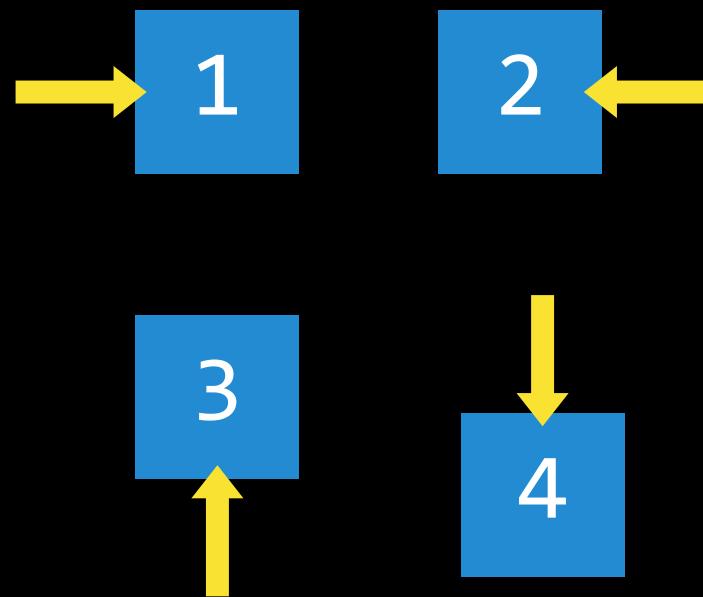
# **actions**

choices that can be made in a state

# actions

ACTIONS( $s$ ) returns the set of actions that can be executed in state  $s$

actions



## **transition model**

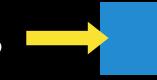
a description of what state results from performing any applicable action in any state

## transition model

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

RESULT(

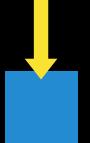
2	4	5	7
8	3	1	11
14	6	10	12
9	13	15	

,  ) =

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	

,  ) =

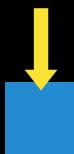
2	4	5	7
8	3	1	11
14	6	10	
9	13	15	12

# transition model

RESULT(

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

,

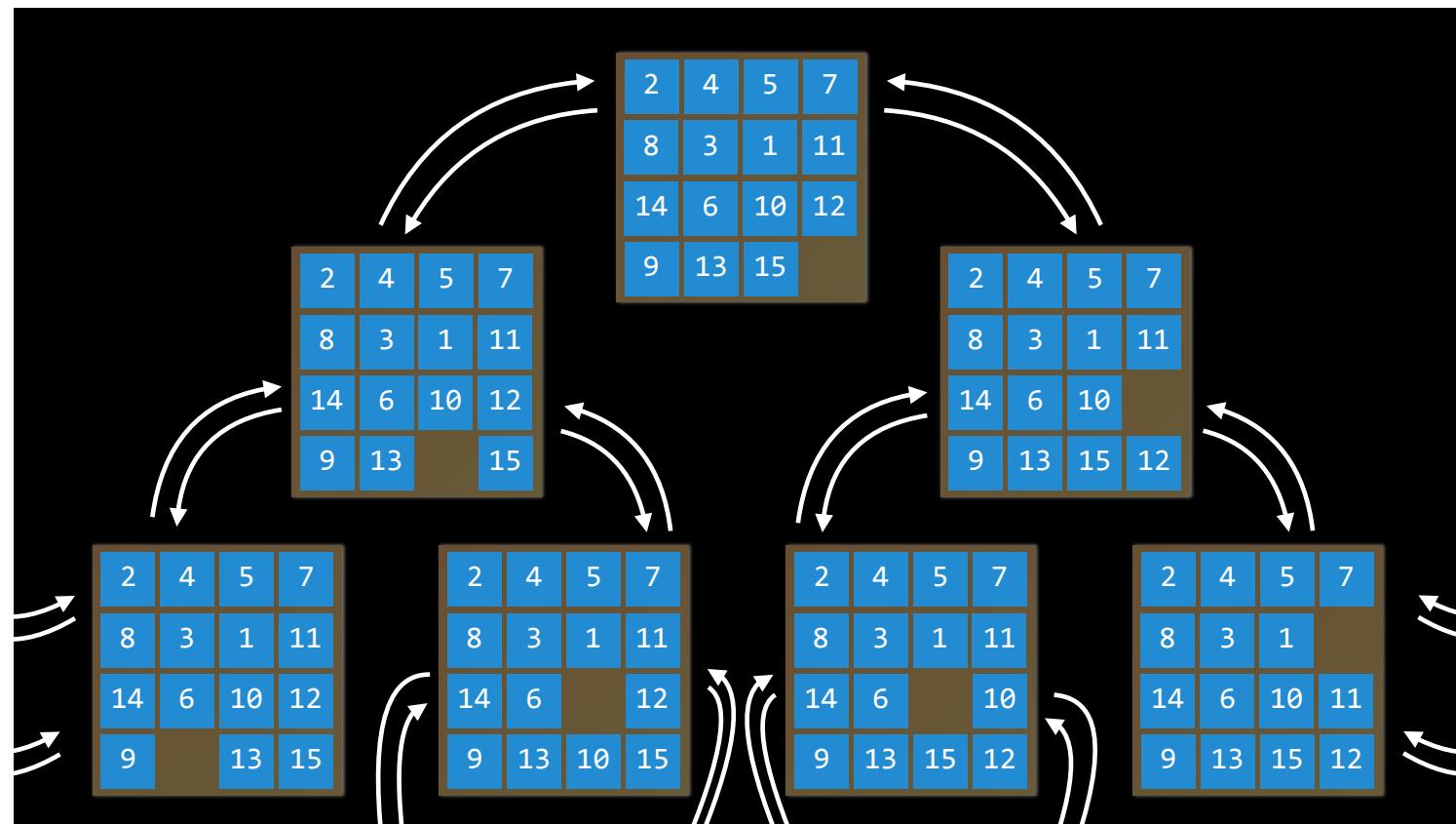


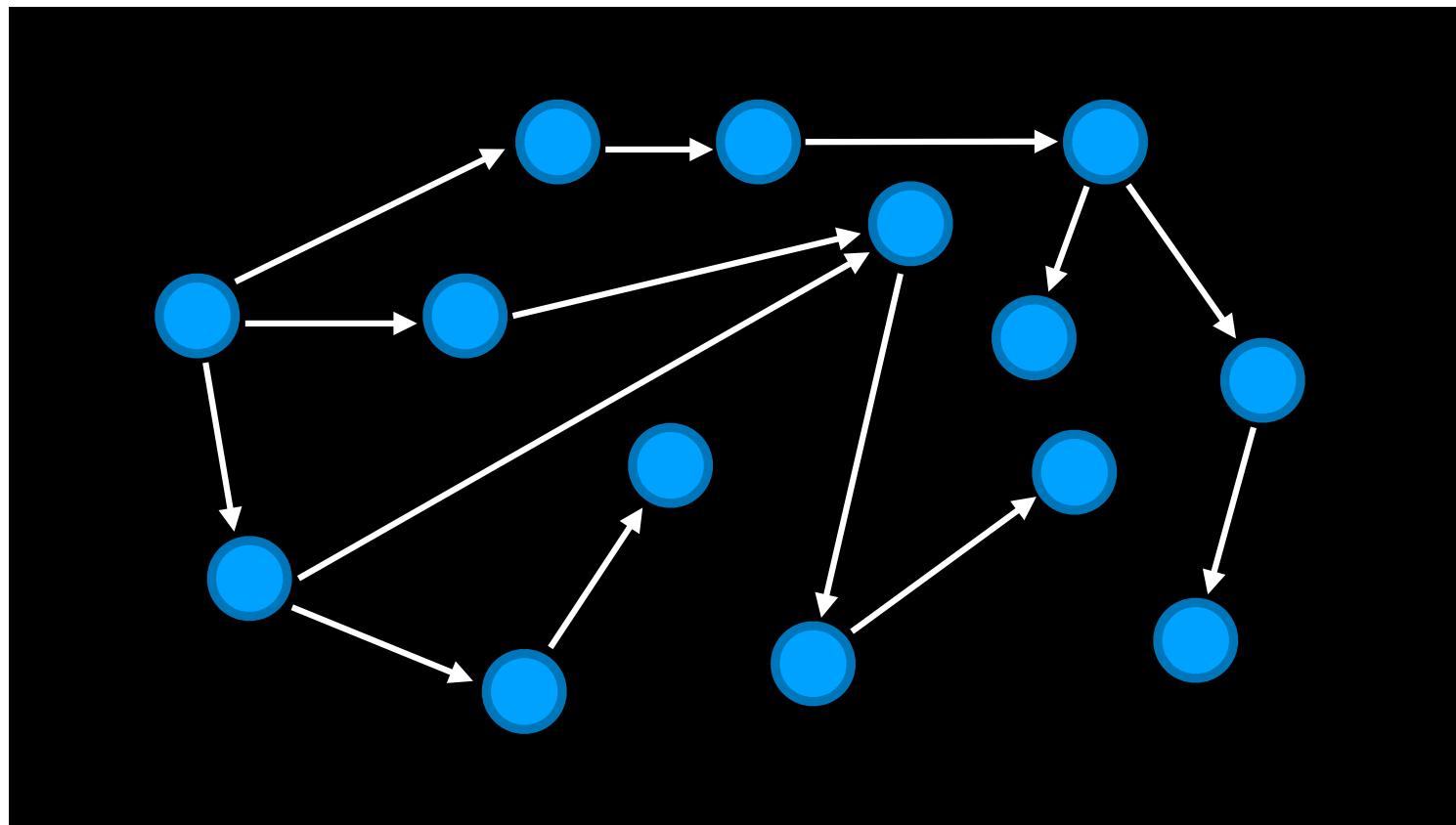
) =

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



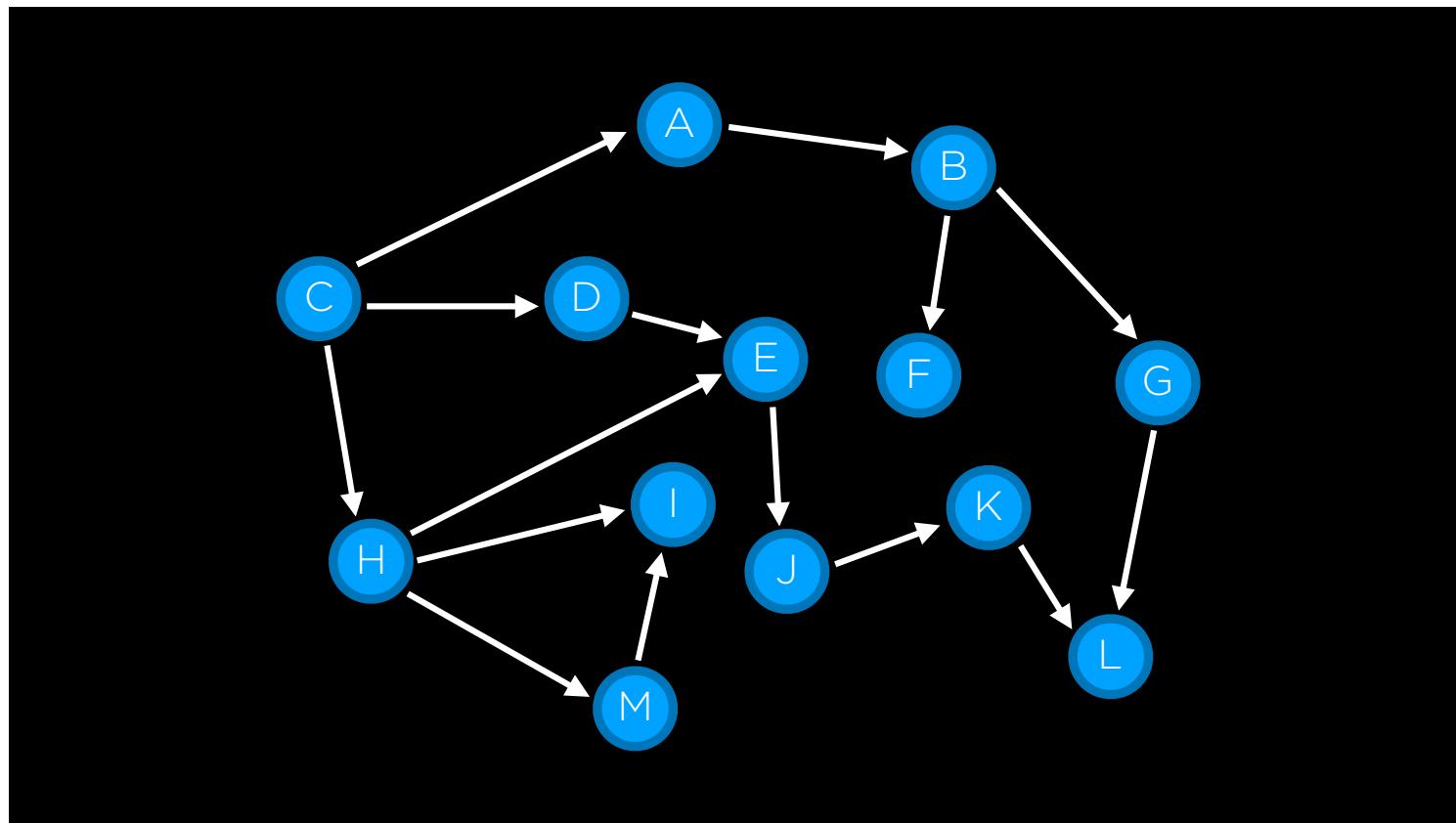


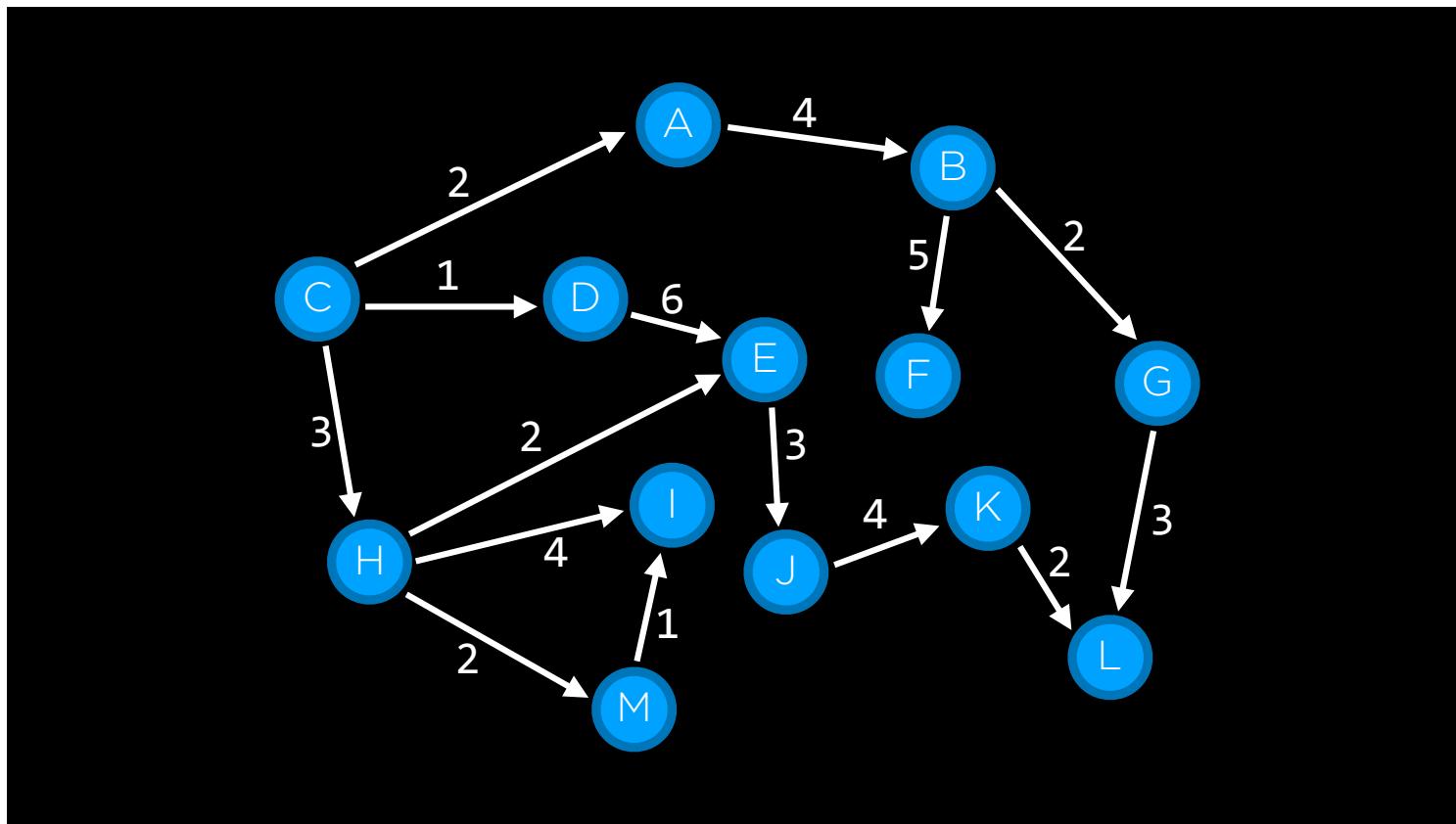
## **goal test**

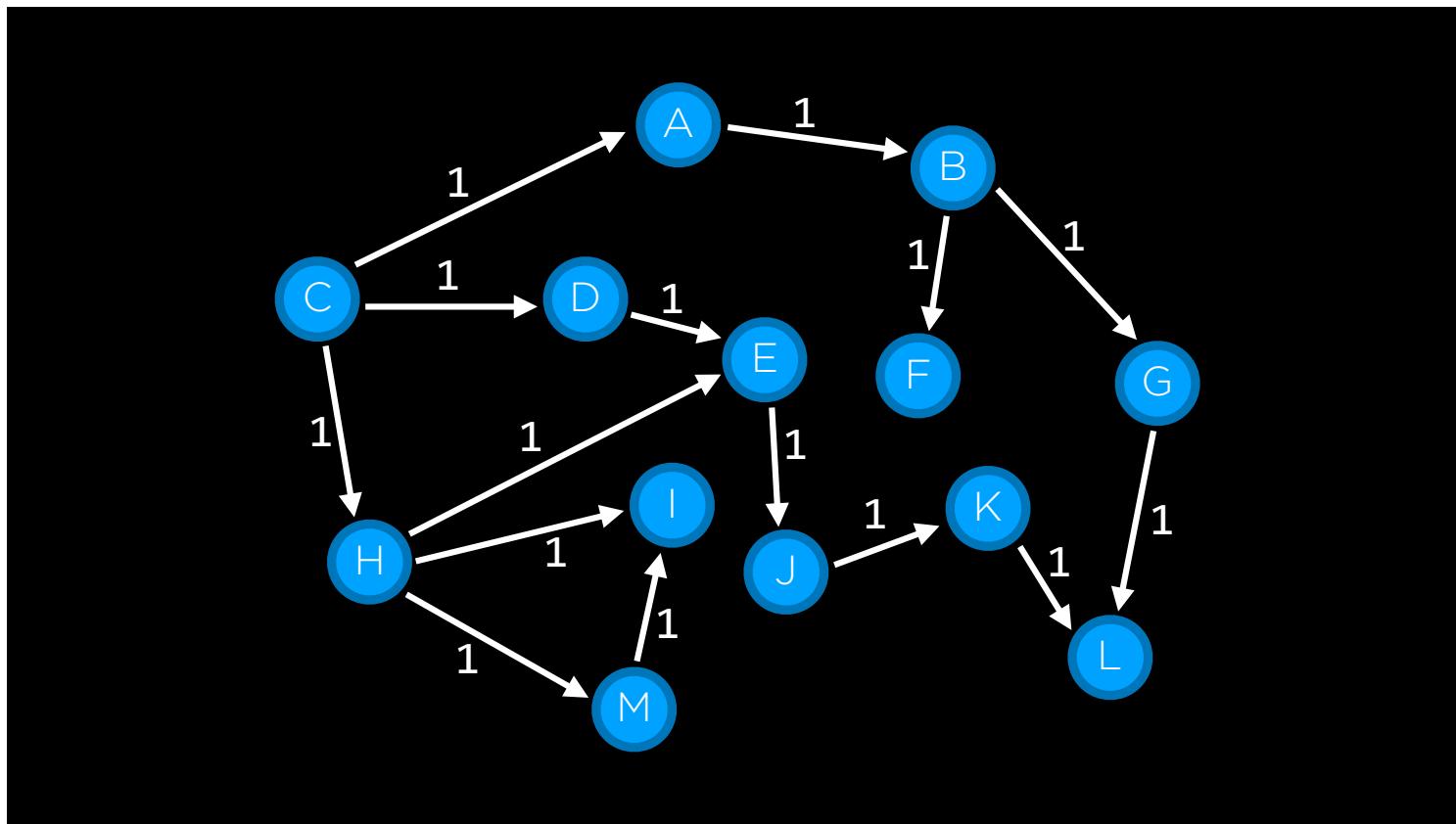
way to determine whether a given state  
is a goal state

# **path cost**

numerical cost associated with a given path







# Search Problems

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

# **solution**

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

# **optimal solution**

a solution that has the lowest path cost  
among all solutions

# node

a data structure that keeps track of

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

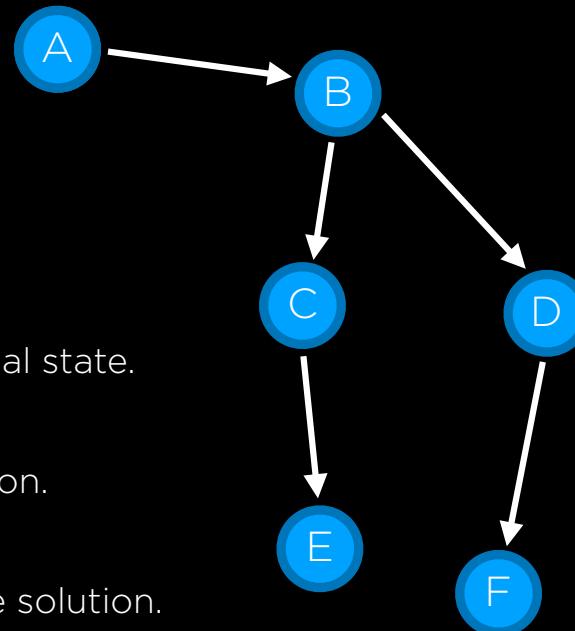
# Approach

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

Find a path from A to E.

**Frontier**

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

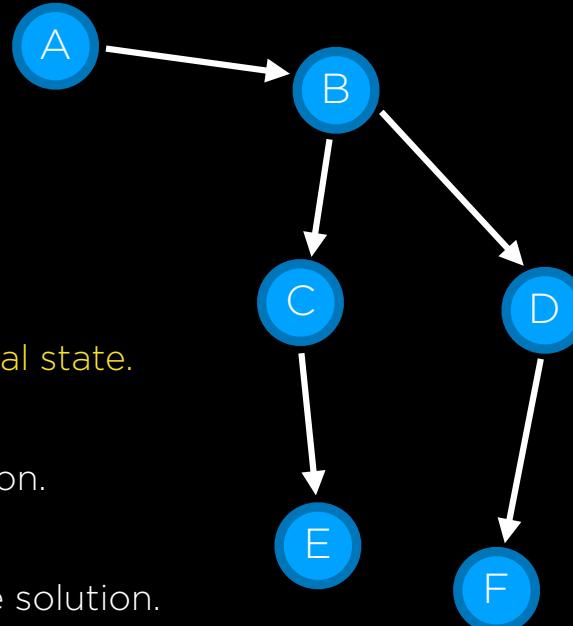


Find a path from A to E.

**Frontier**



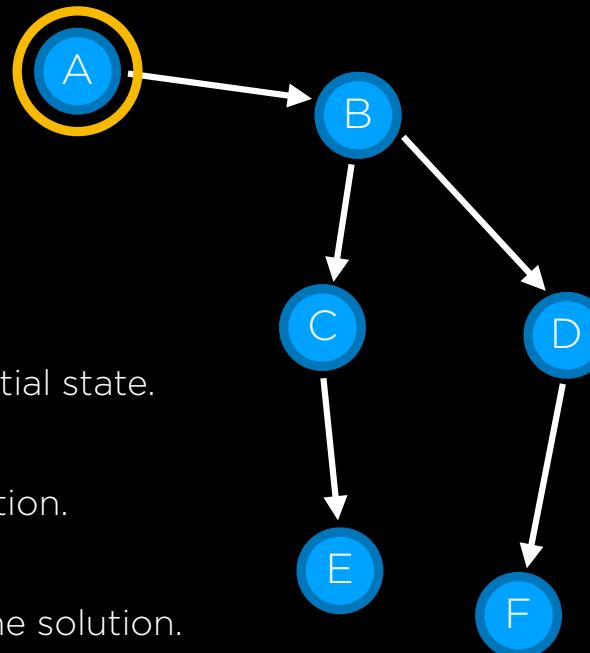
- Start with a **frontier** that contains the initial state.
- Repeat:
  - If the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - If node contains goal state, return the solution.
  - **Expand** node, add resulting nodes to the frontier.



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.

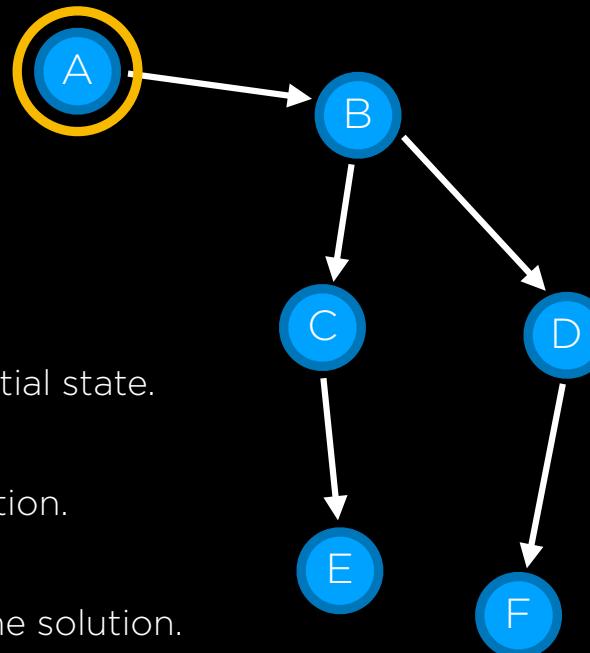


Find a path from A to E.

**Frontier**



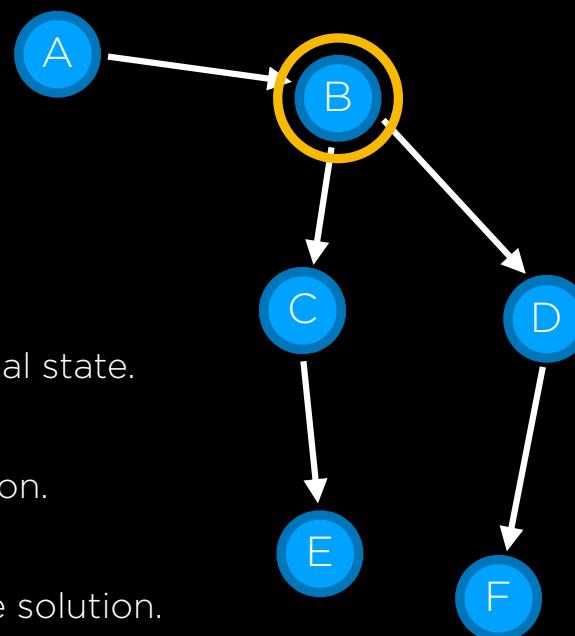
- Start with a **frontier** that contains the initial state.
- Repeat:
  - If the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - If node contains goal state, return the solution.
  - **Expand** node, add resulting nodes to the frontier.



Find a path from A to E.

**Frontier**

- 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.

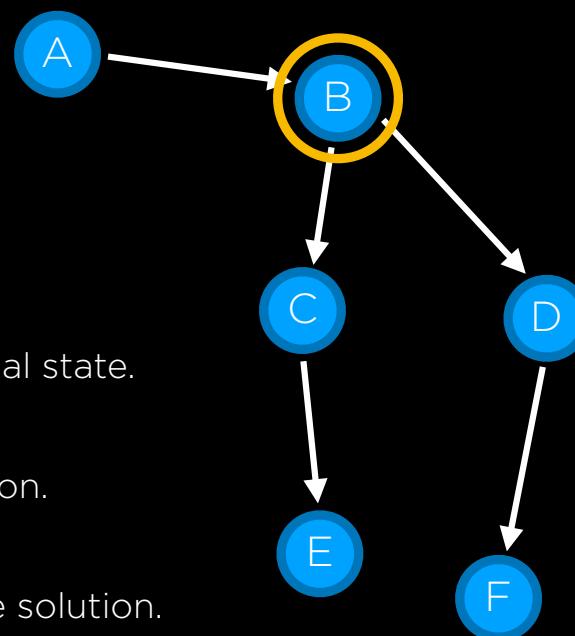


Find a path from A to E.

**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.

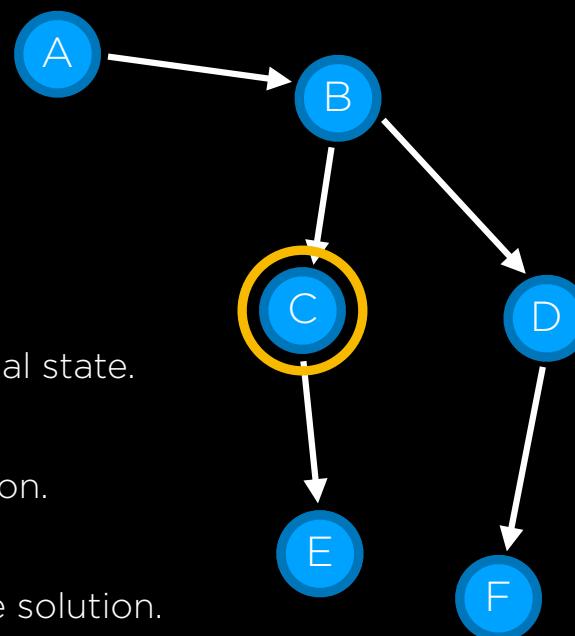


Find a path from A to E.

**Frontier**

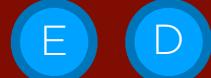


- Start with a **frontier** that contains the initial state.
- Repeat:
  - If the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - If node contains goal state, return the solution.
  - **Expand** node, add resulting nodes to the frontier.

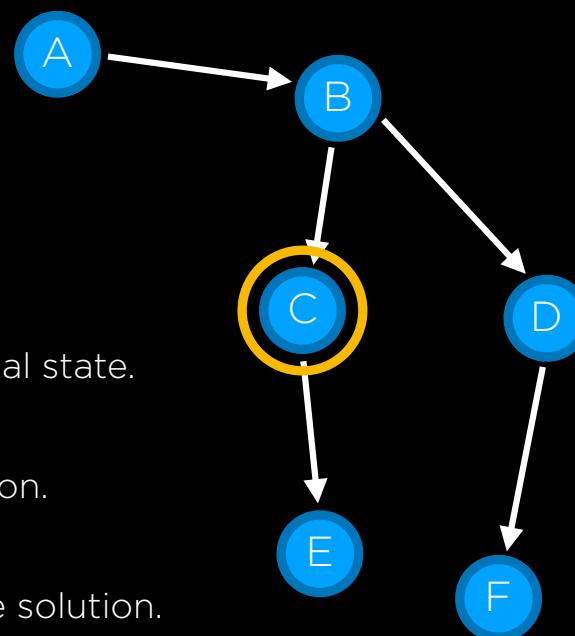


Find a path from A to E.

**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.

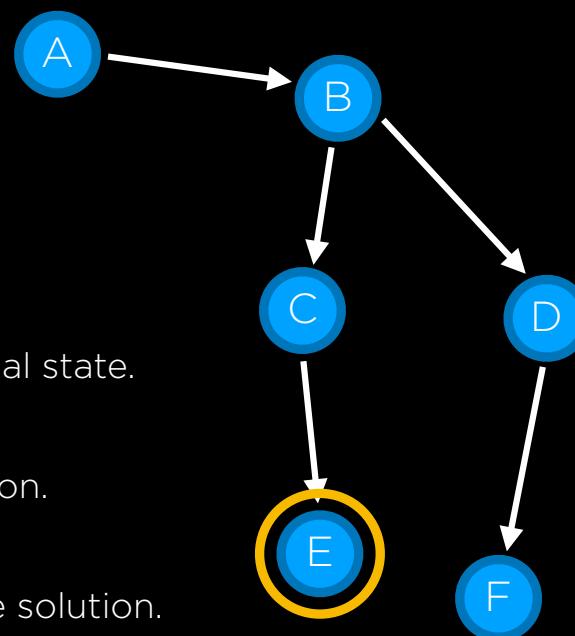


Find a path from A to E.

**Frontier**



- 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.

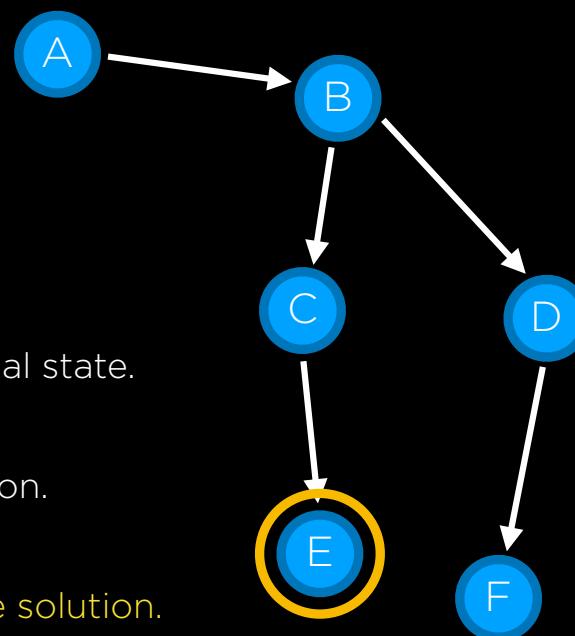


Find a path from A to E.

**Frontier**



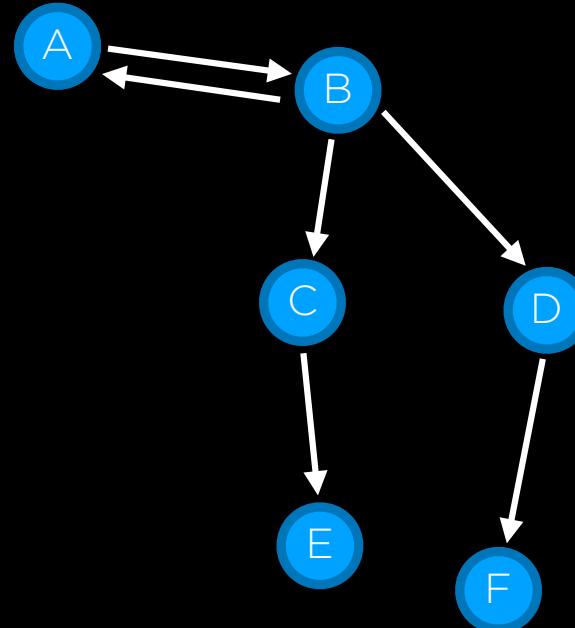
- Start with a **frontier** that contains the initial state.
- Repeat:
  - If the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - If node contains goal state, return the solution.
  - **Expand** node, add resulting nodes to the frontier.



**What could go wrong?**

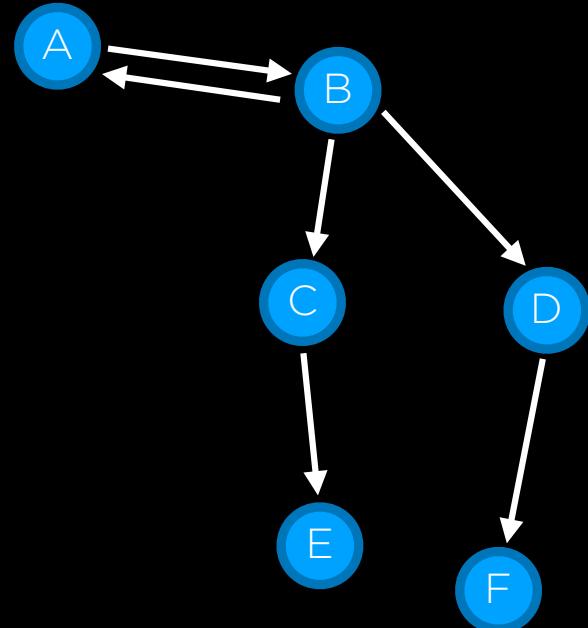
Find a path from A to E.

**Frontier**



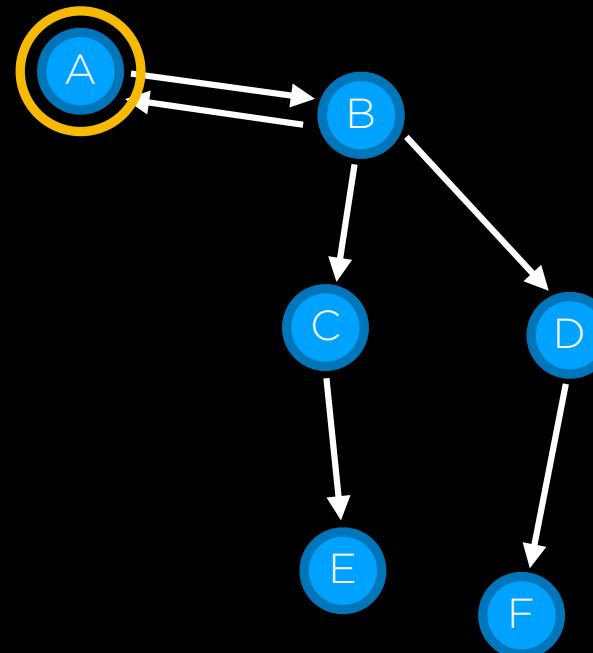
Find a path from A to E.

**Frontier**



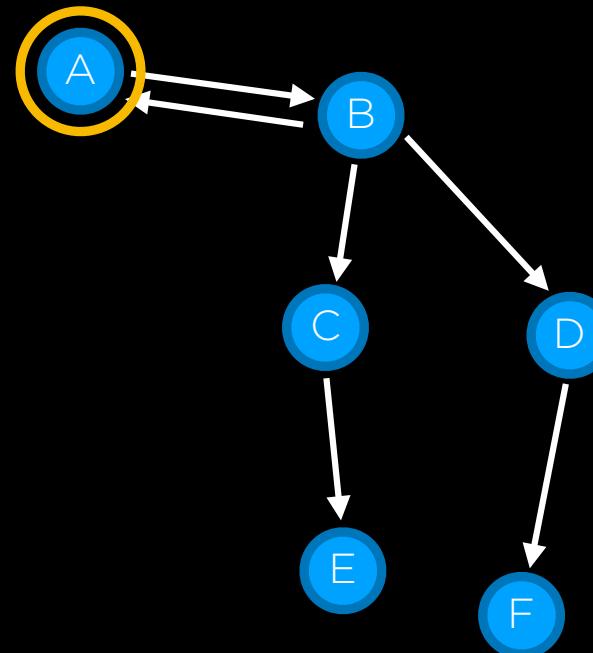
Find a path from A to E.

**Frontier**



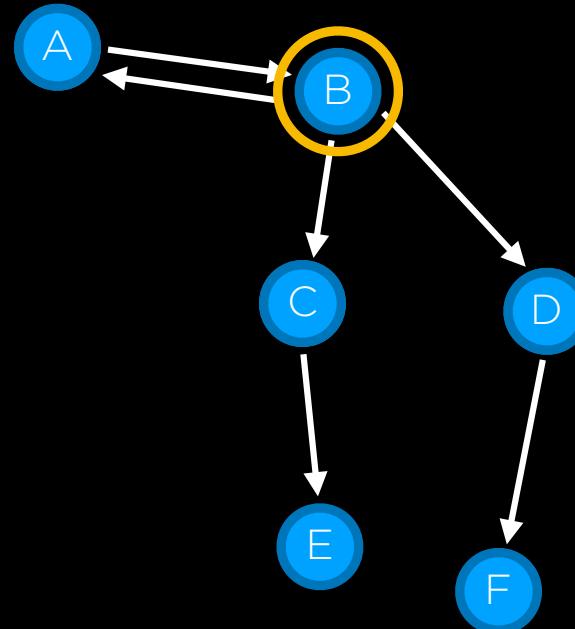
Find a path from A to E.

**Frontier**



Find a path from A to E.

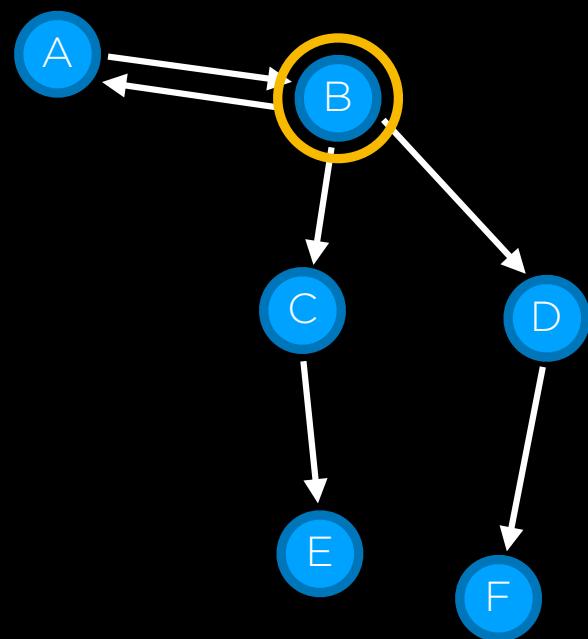
**Frontier**



Find a path from A to E.

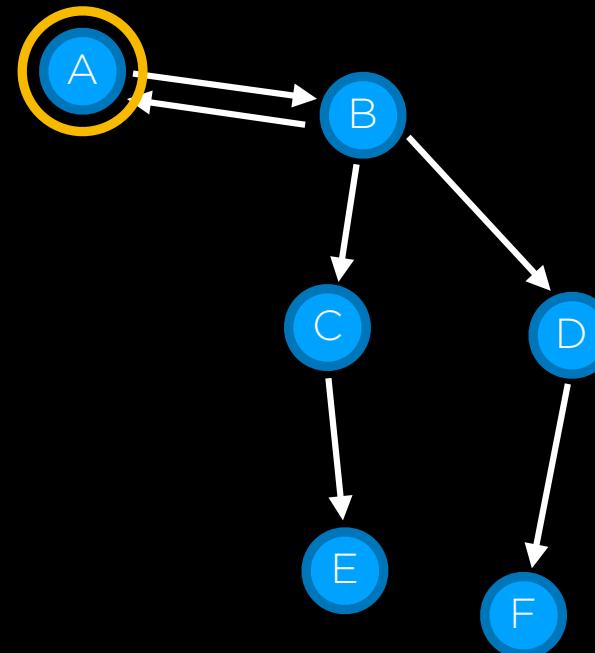
**Frontier**

A C D



Find a path from A to E.

**Frontier**



# 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.

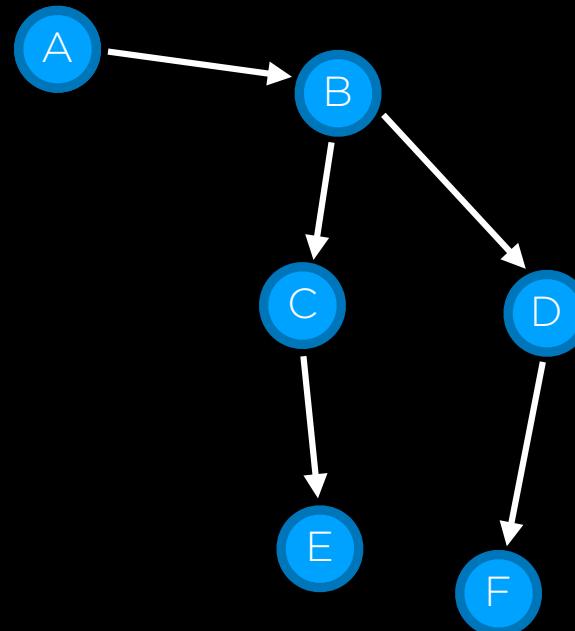
# **stack**

last-in first-out data type

Find a path from A to E.

**Frontier**

**Explored Set**

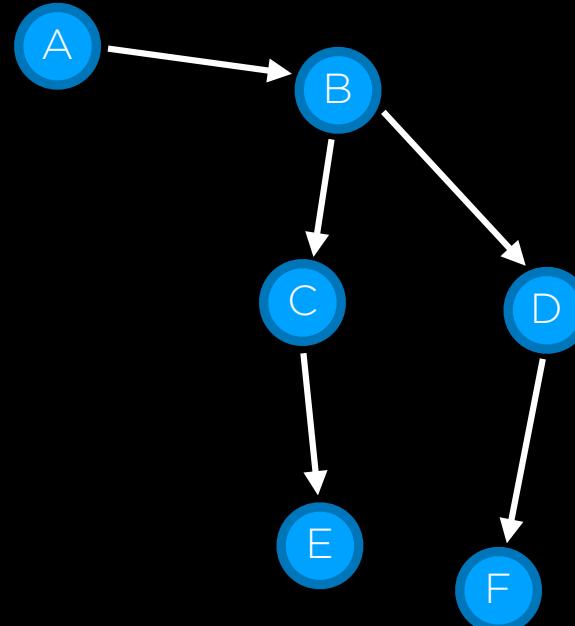


Find a path from A to E.

**Frontier**



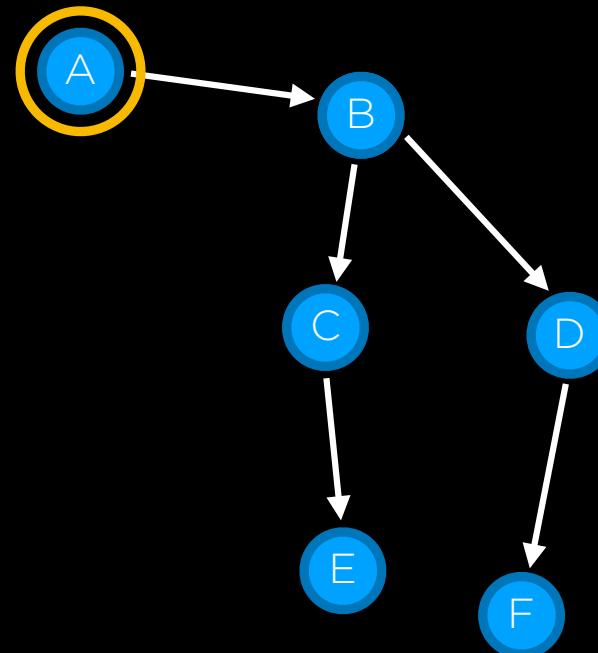
**Explored Set**



Find a path from A to E.

**Frontier**

**Explored Set**

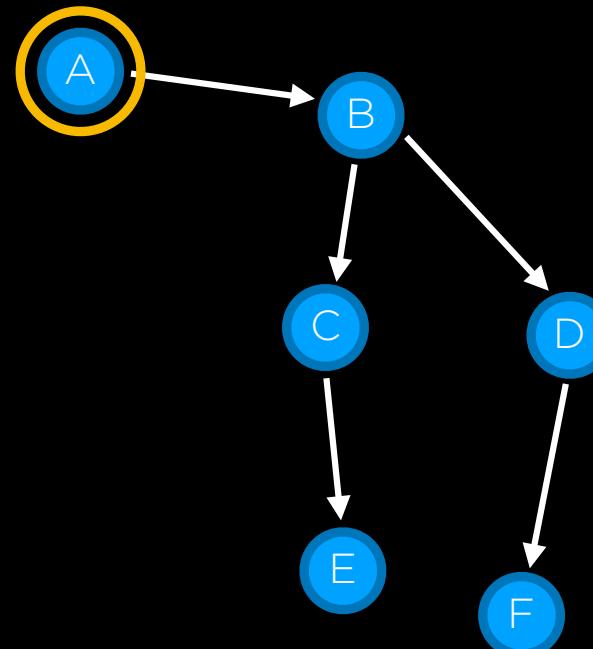


Find a path from A to E.

**Frontier**



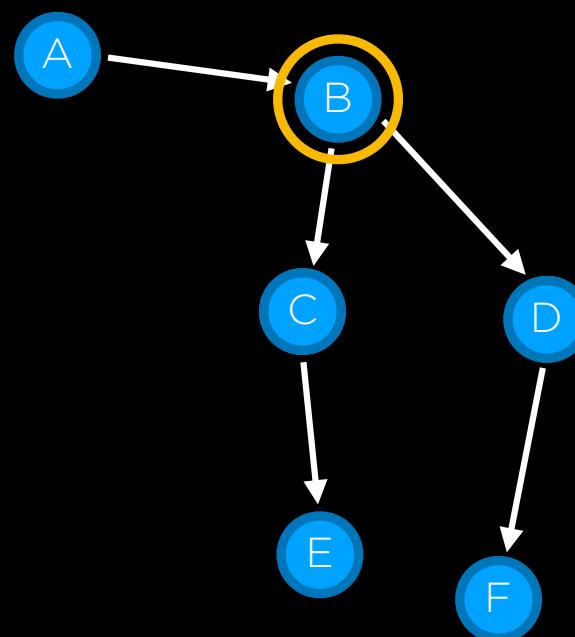
**Explored Set**



Find a path from A to E.

**Frontier**

**Explored Set**

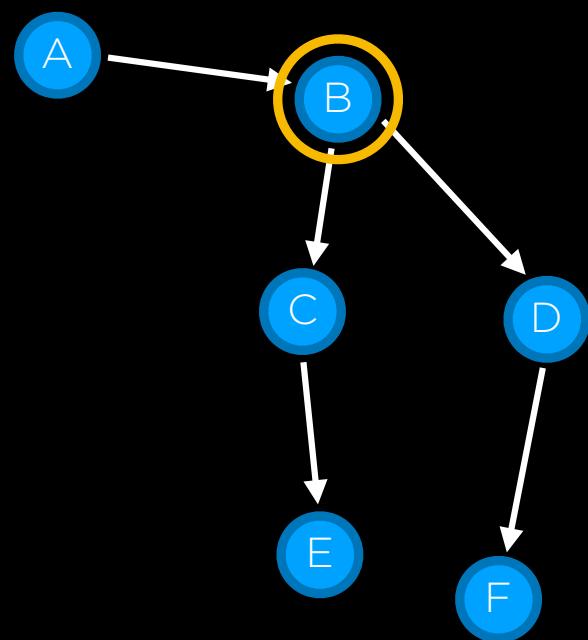


Find a path from A to E.

**Frontier**



**Explored Set**

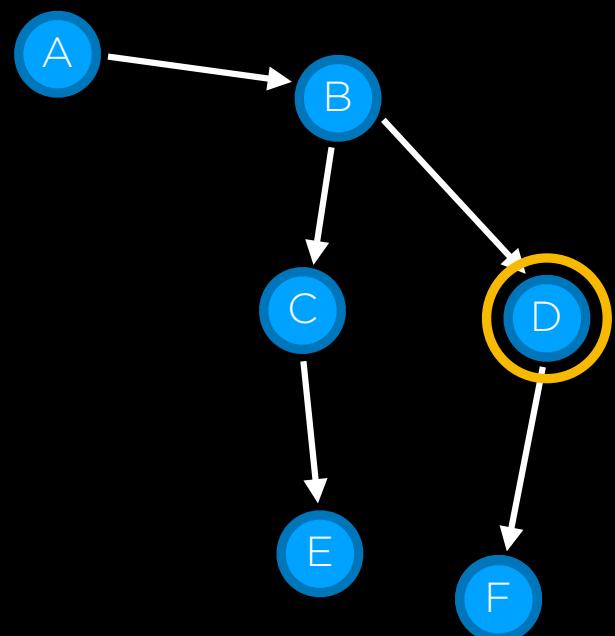


Find a path from A to E.

**Frontier**



**Explored Set**

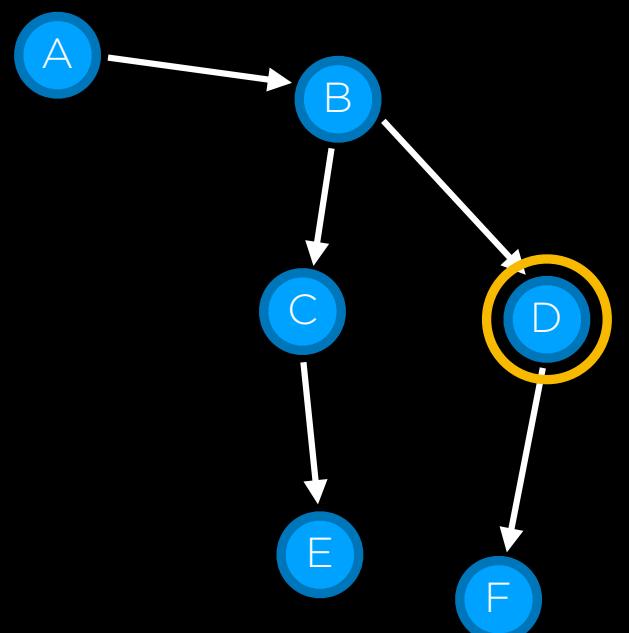


Find a path from A to E.

**Frontier**



**Explored Set**

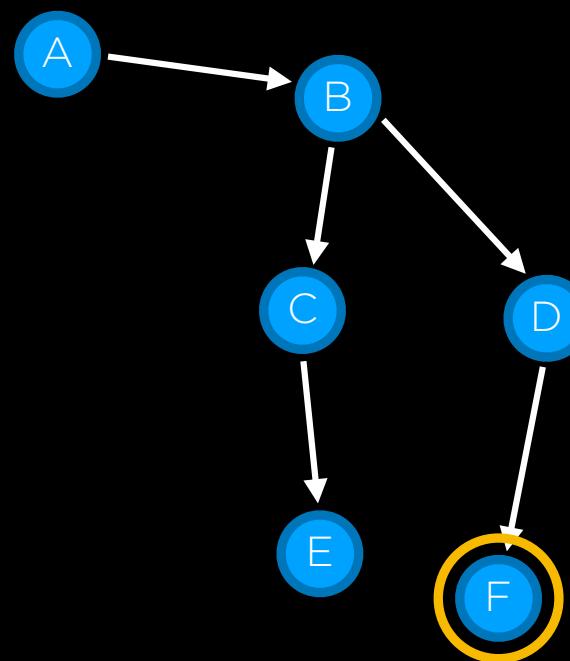


Find a path from A to E.

**Frontier**



**Explored Set**

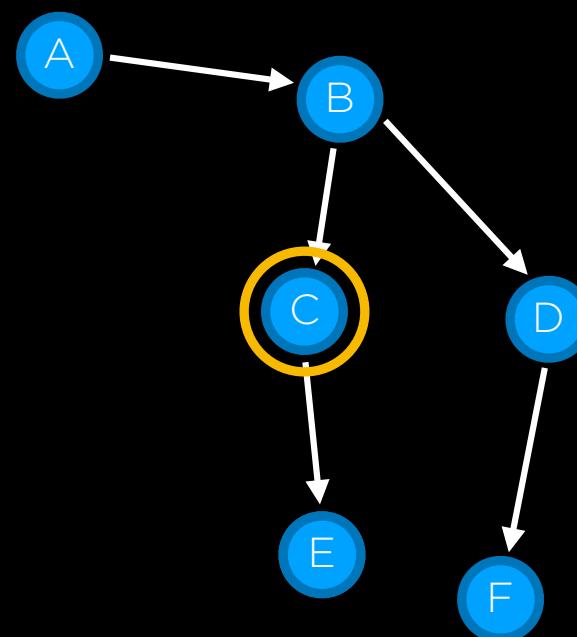


Find a path from A to E.

**Frontier**

**Explored Set**

- A
- B
- D
- F
- C

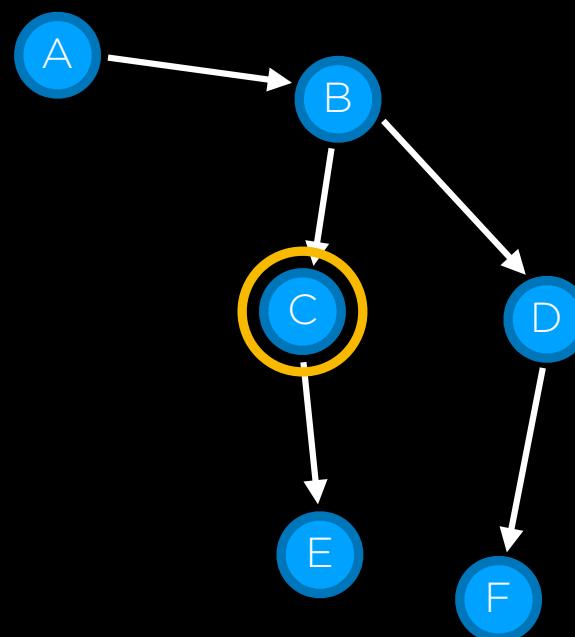


Find a path from A to E.

**Frontier**



**Explored Set**

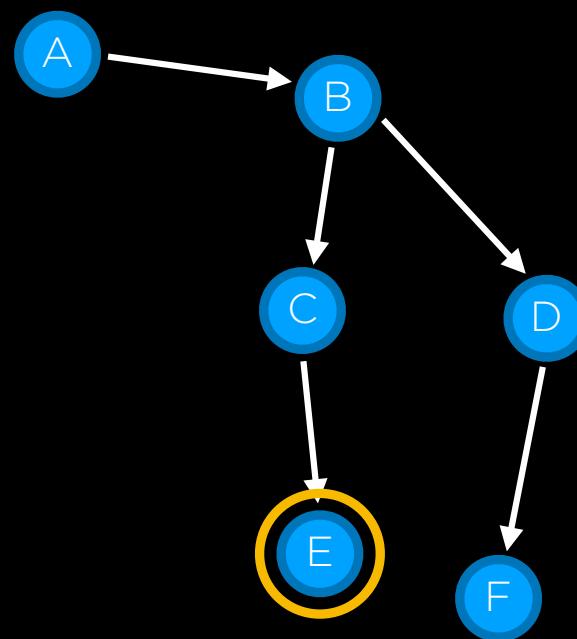


Find a path from A to E.

**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

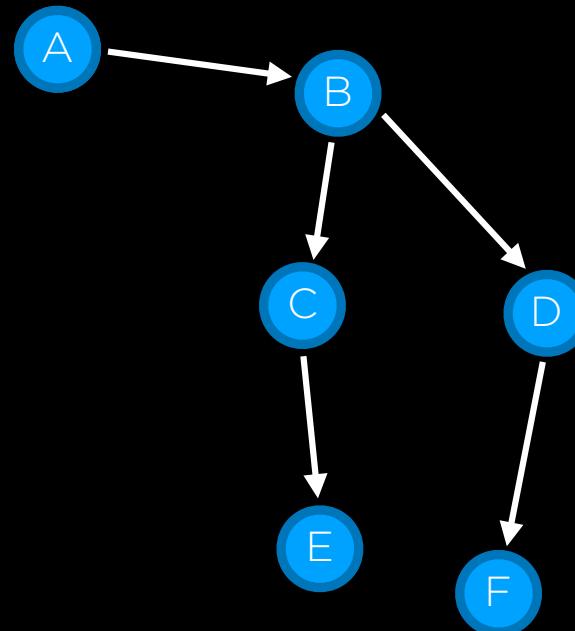
# **queue**

first-in first-out data type

Find a path from A to E.

**Frontier**

**Explored Set**

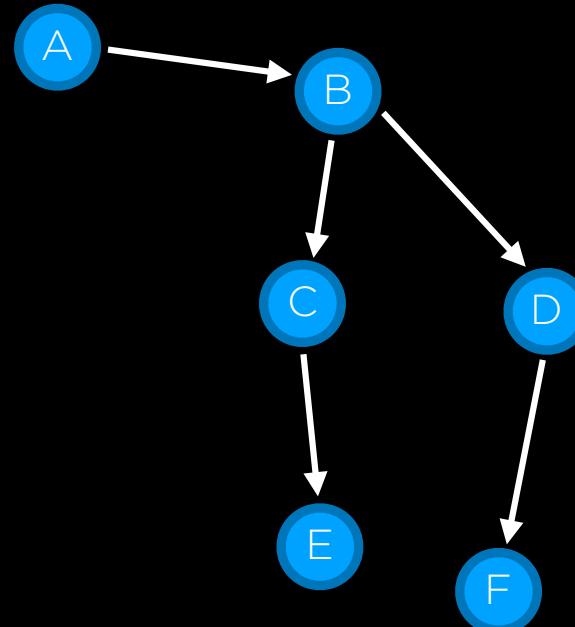


Find a path from A to E.

**Frontier**



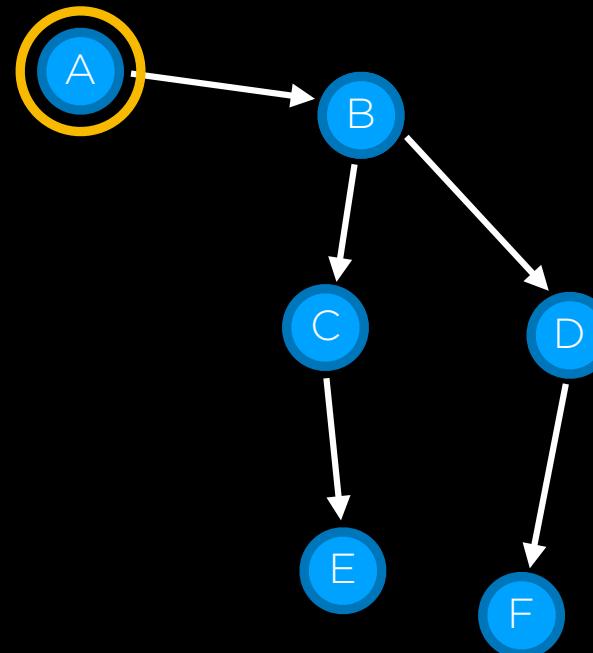
**Explored Set**



Find a path from A to E.

**Frontier**

**Explored Set**

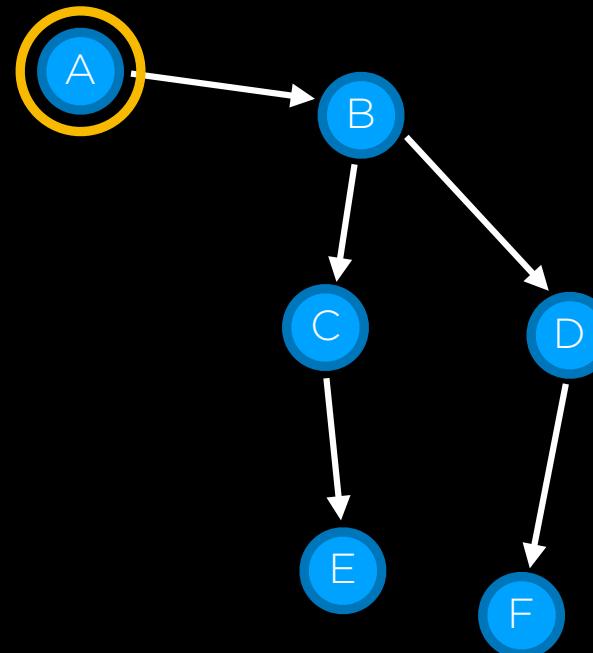


Find a path from A to E.

**Frontier**



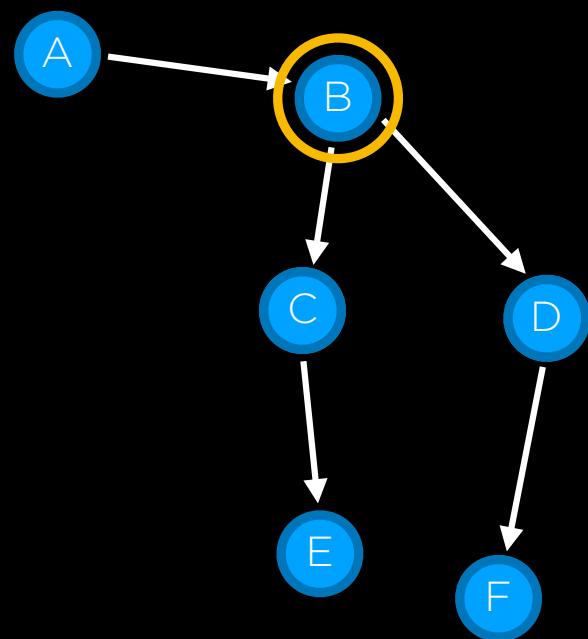
**Explored Set**



Find a path from A to E.

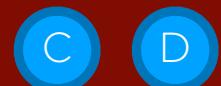
**Frontier**

**Explored Set**

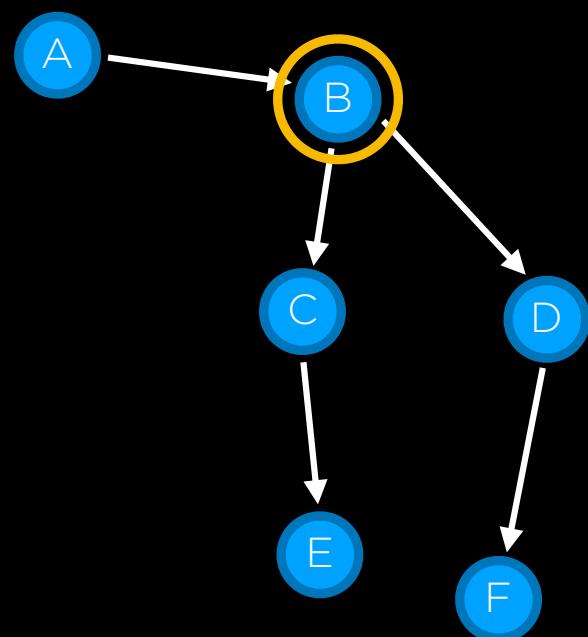


Find a path from A to E.

**Frontier**



**Explored Set**

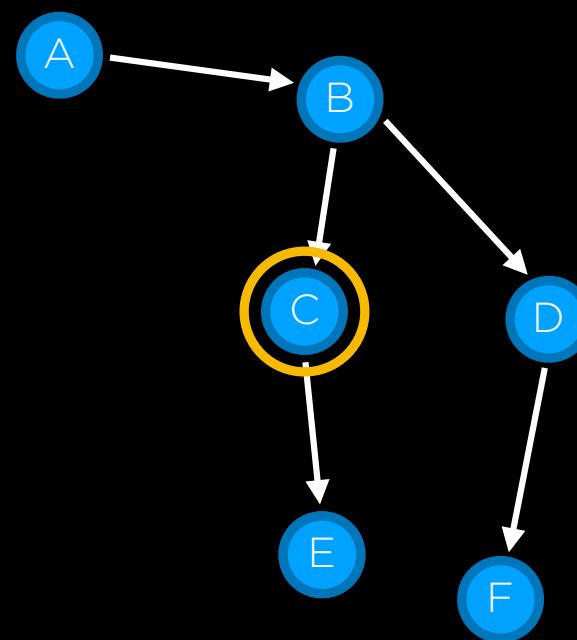


Find a path from A to E.

**Frontier**



**Explored Set**

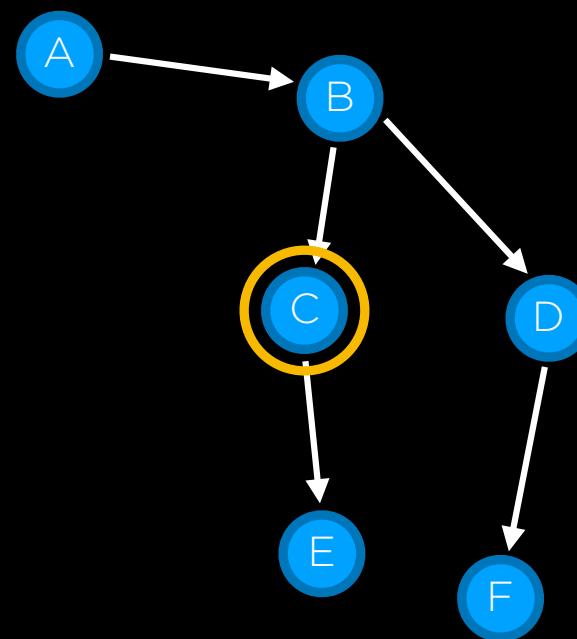


Find a path from A to E.

**Frontier**



**Explored Set**

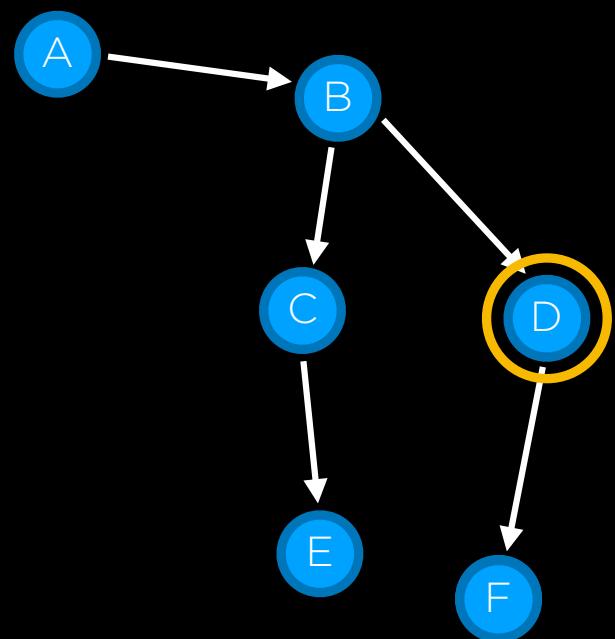


Find a path from A to E.

**Frontier**



**Explored Set**

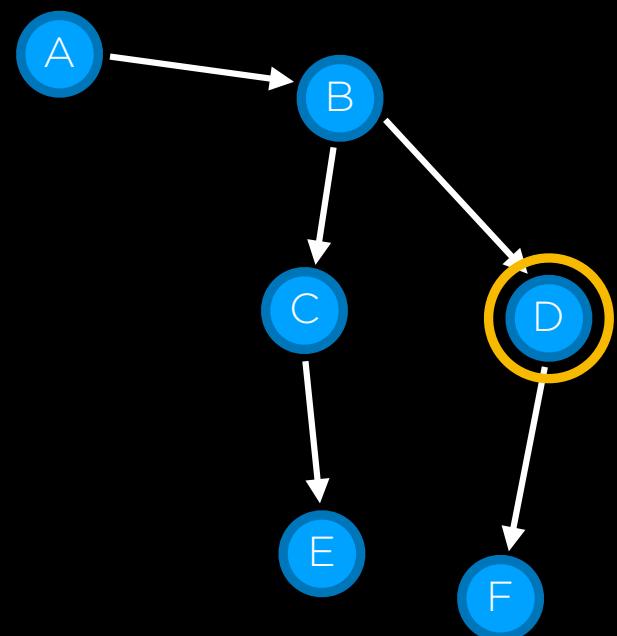


Find a path from A to E.

**Frontier**



**Explored Set**



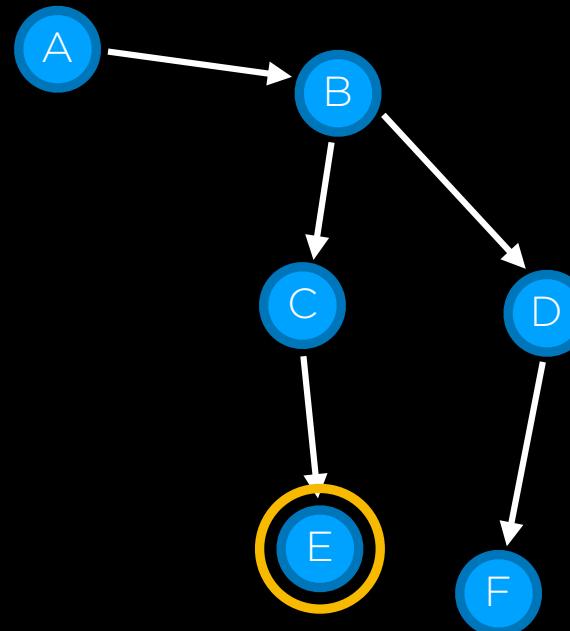
Find a path from A to E.

**Frontier**

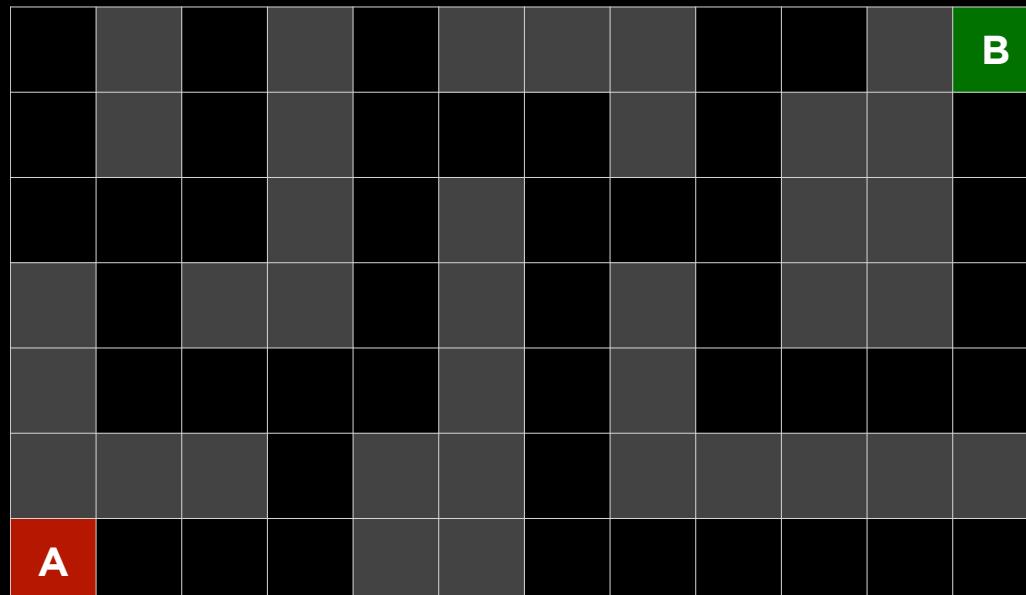
F

**Explored Set**

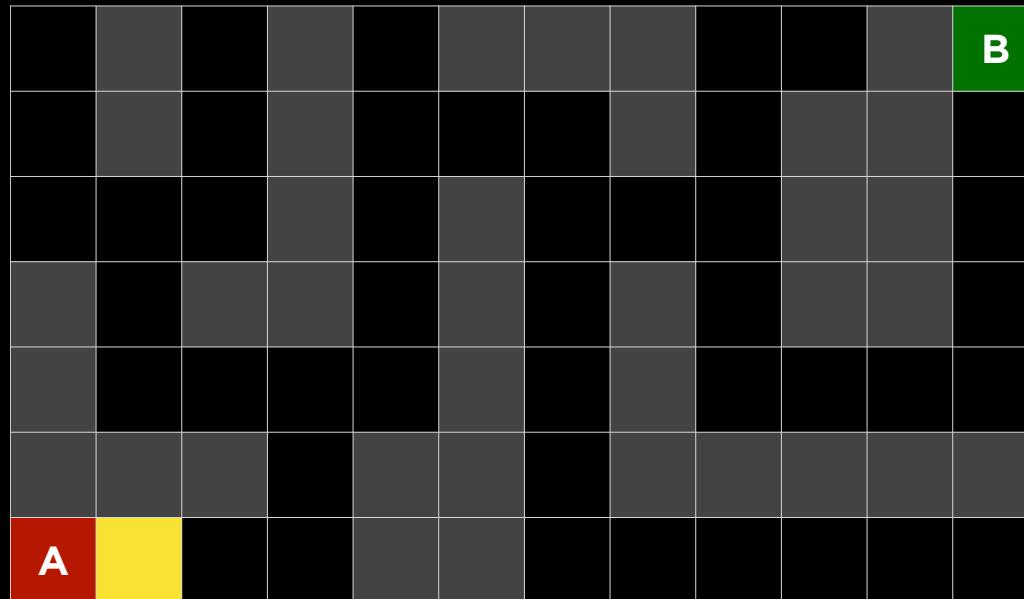
A B C D



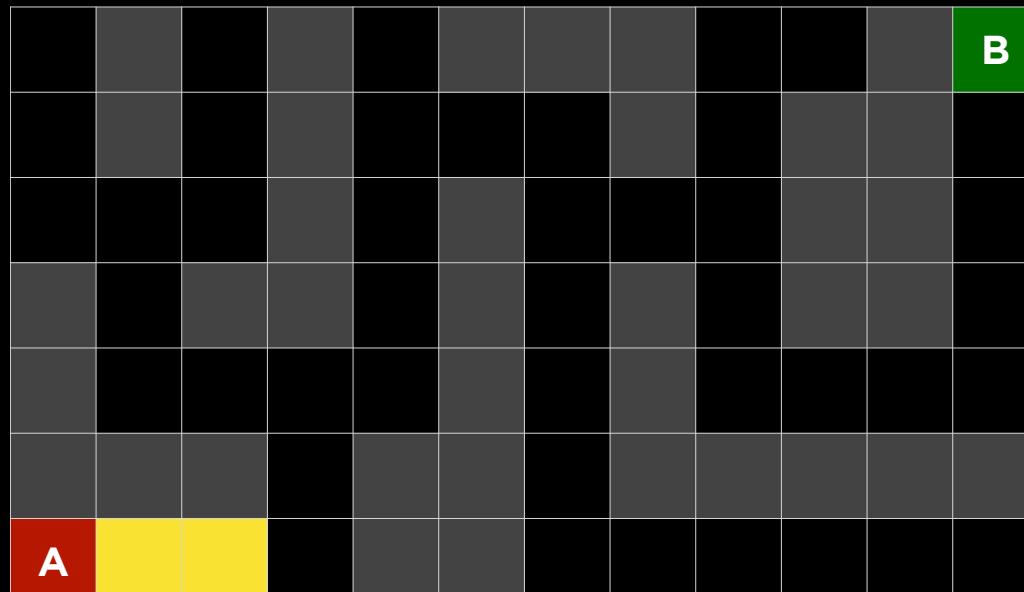
## Depth-First Search



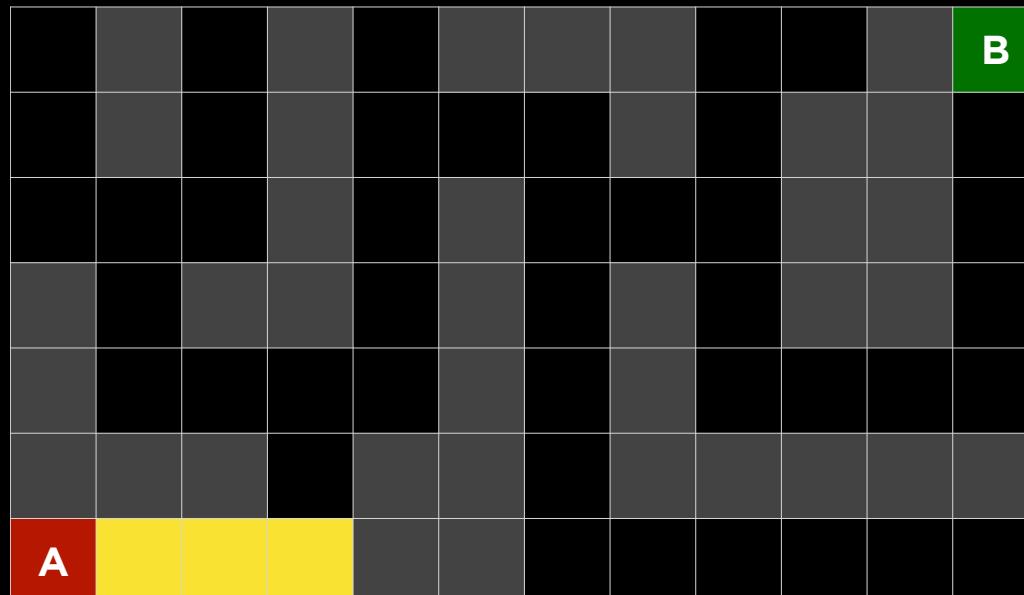
## Depth-First Search



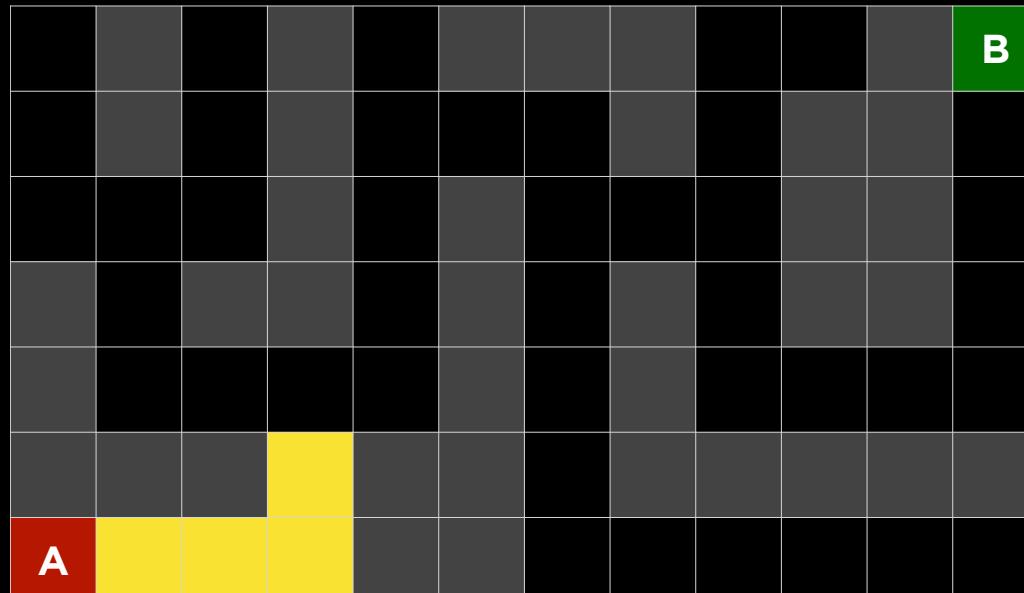
# Depth-First Search



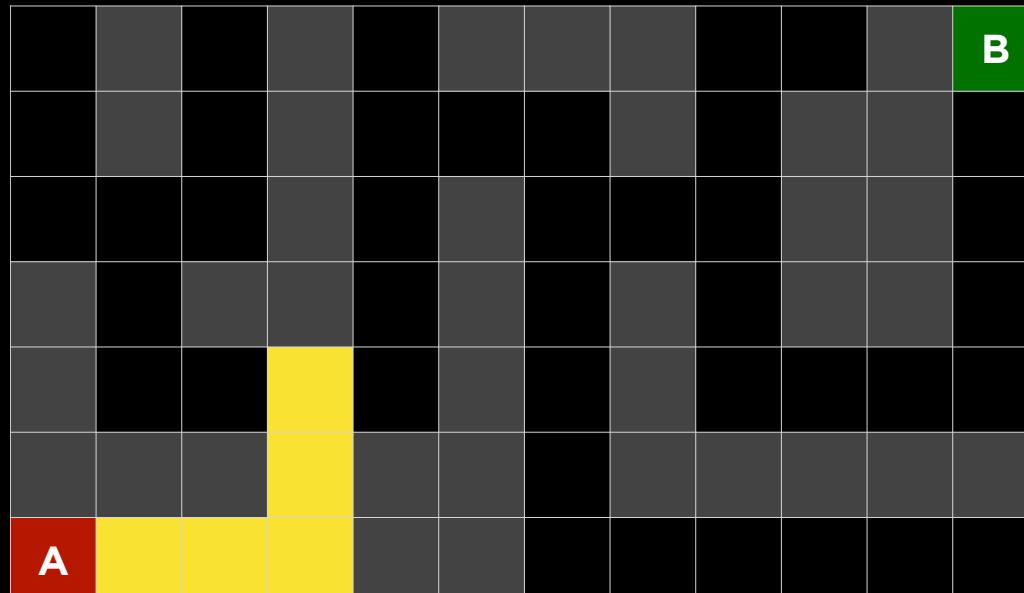
## Depth-First Search



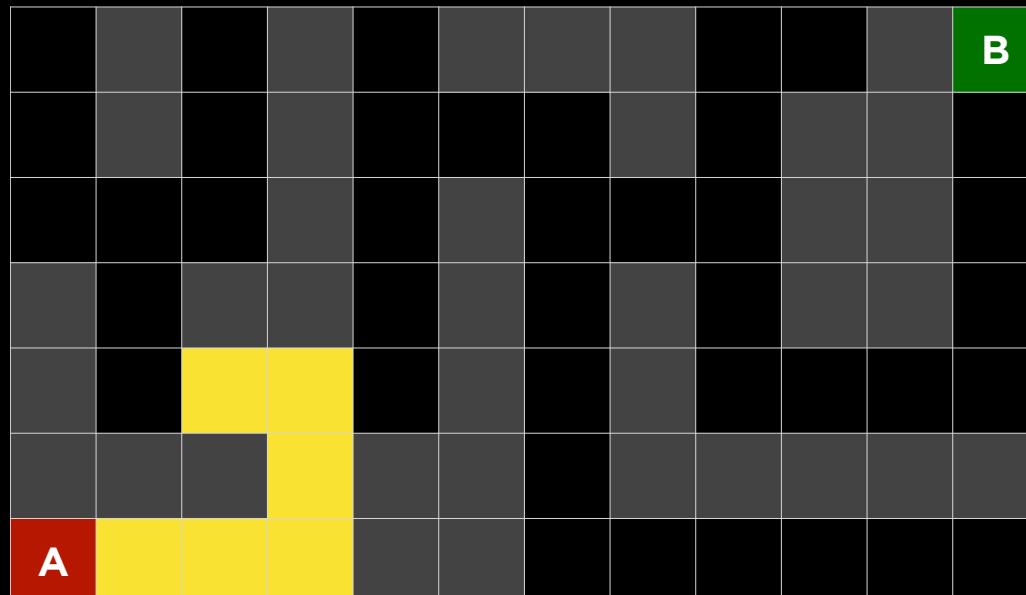
## Depth-First Search



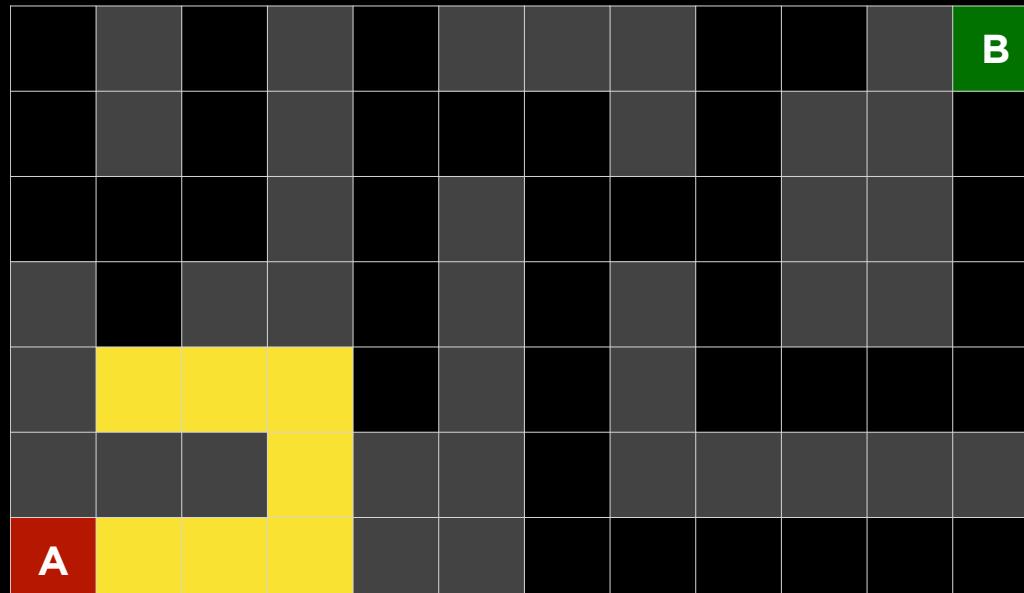
## Depth-First Search



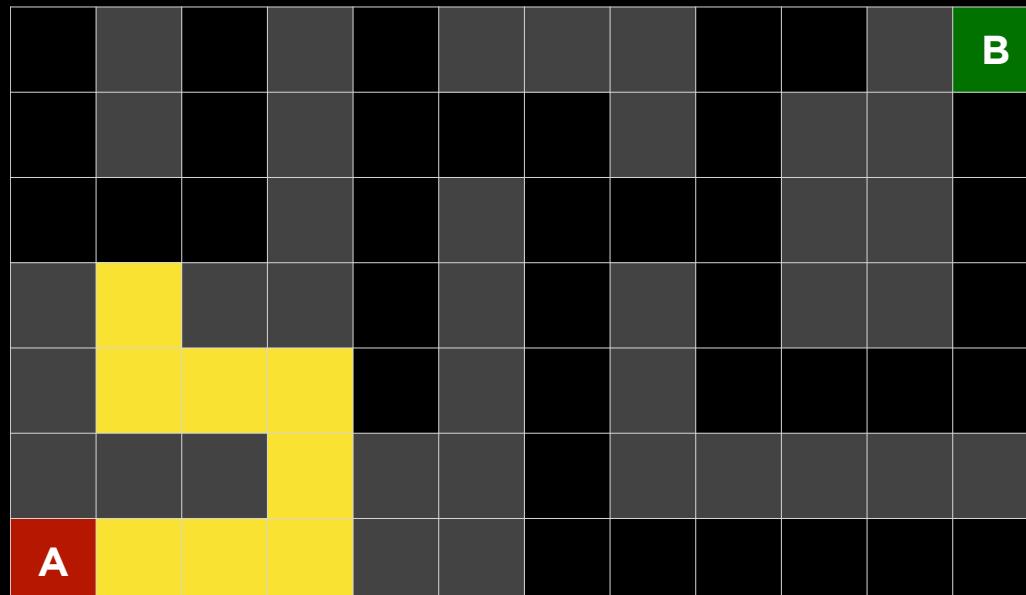
## Depth-First Search



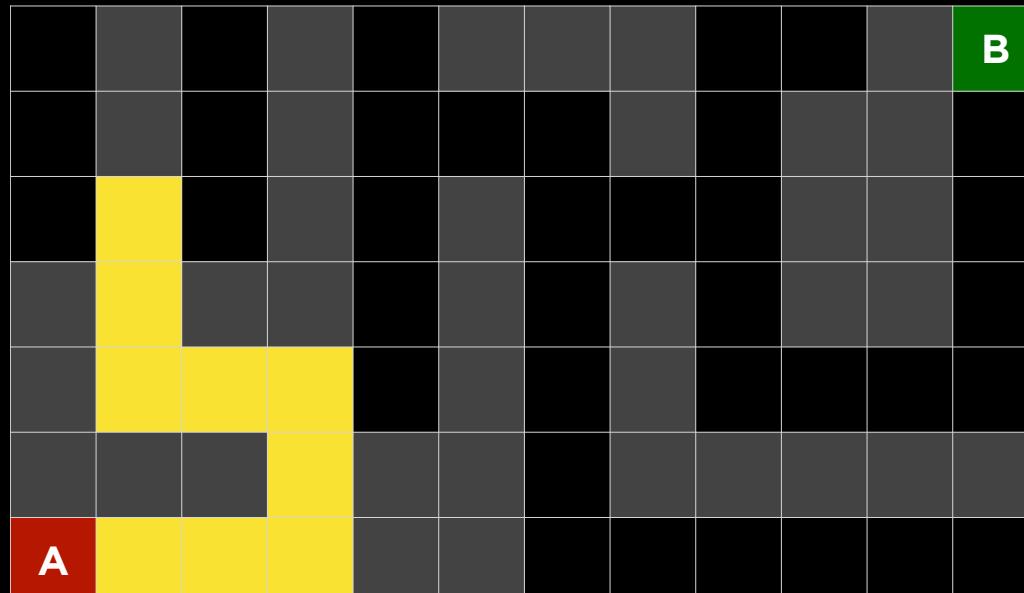
## Depth-First Search



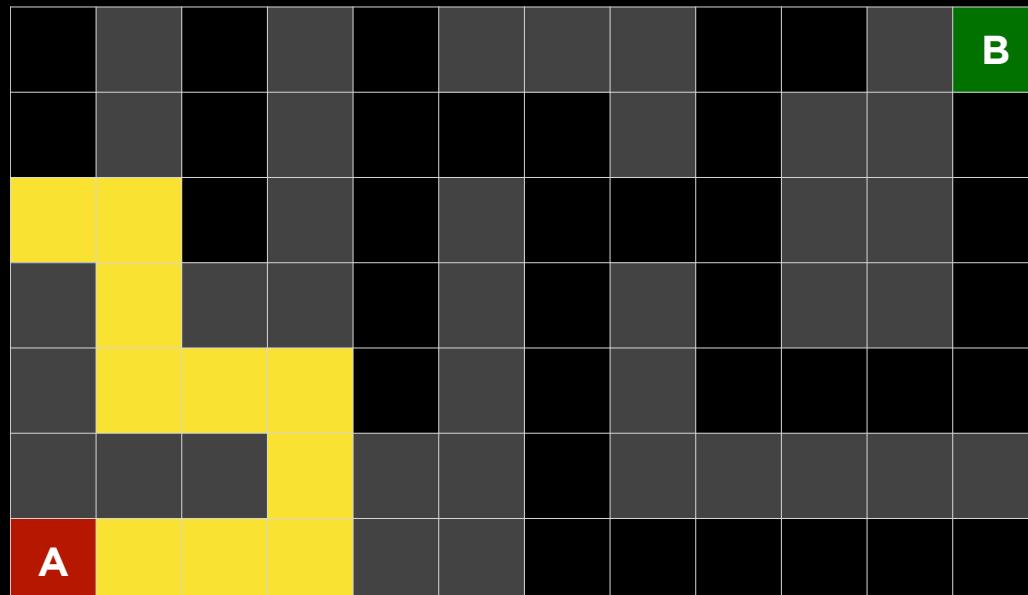
## Depth-First Search



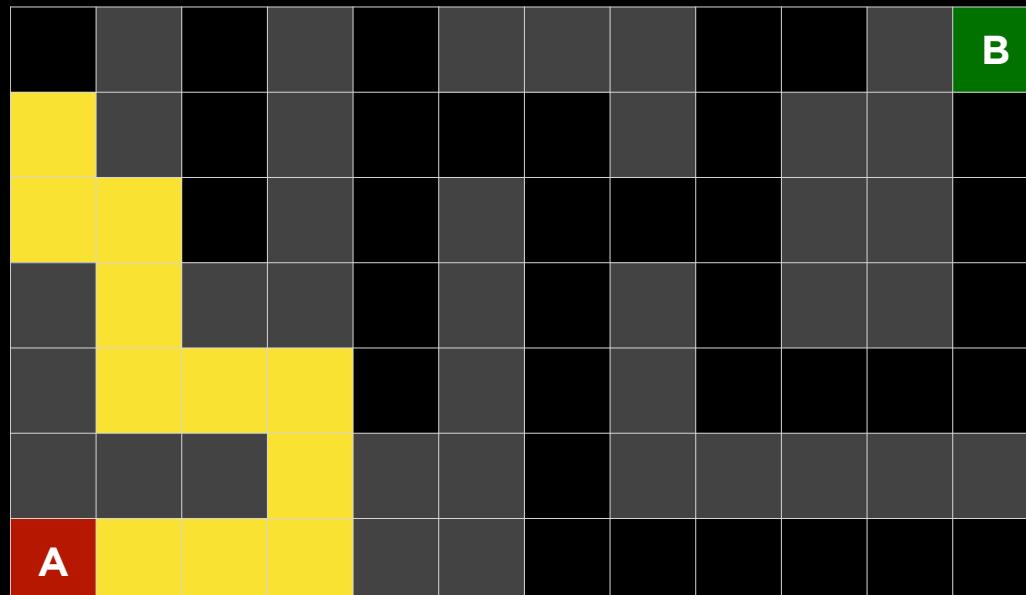
## Depth-First Search



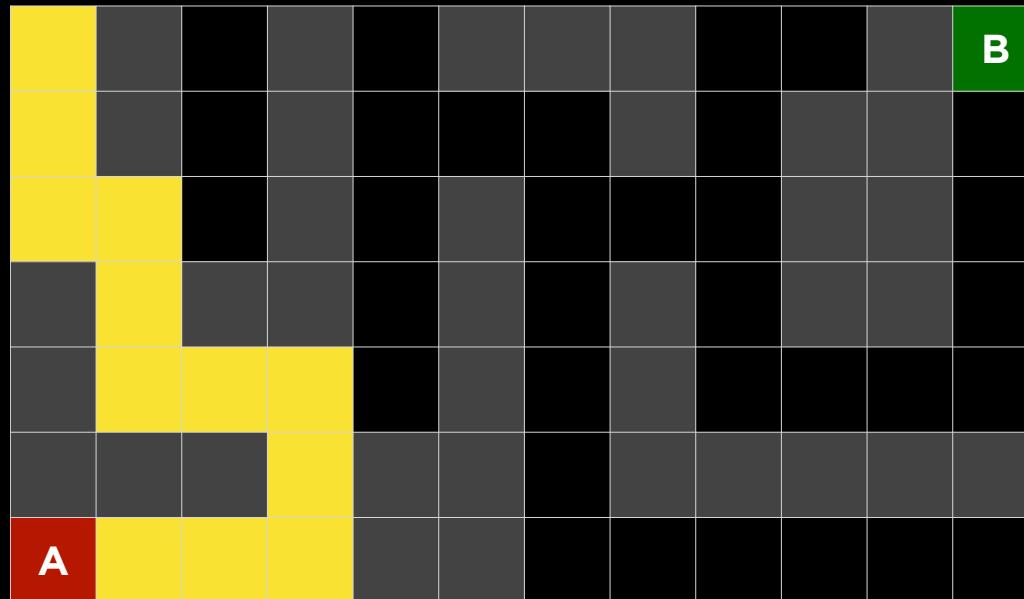
## Depth-First Search



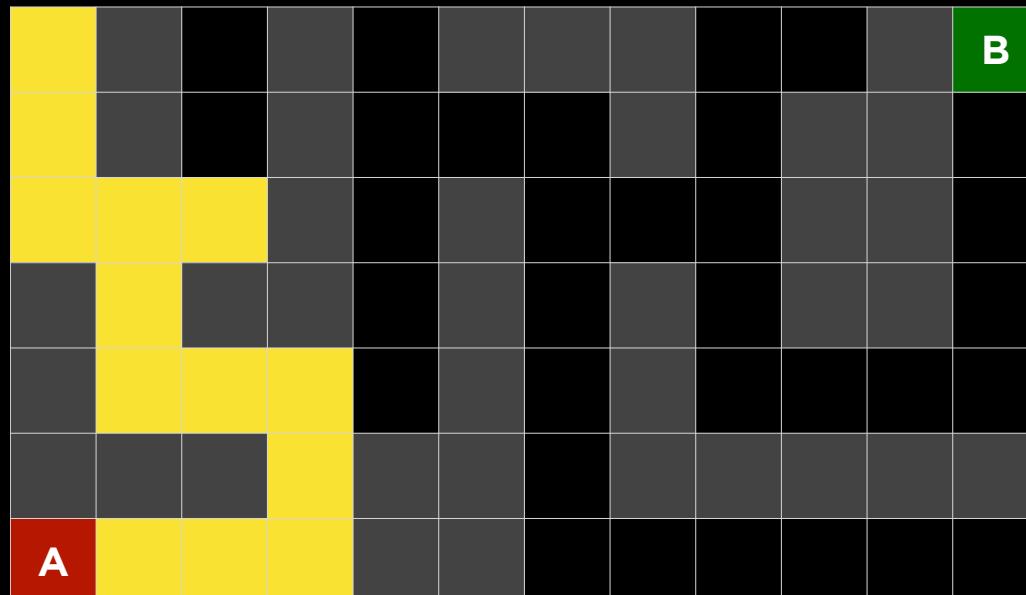
## Depth-First Search



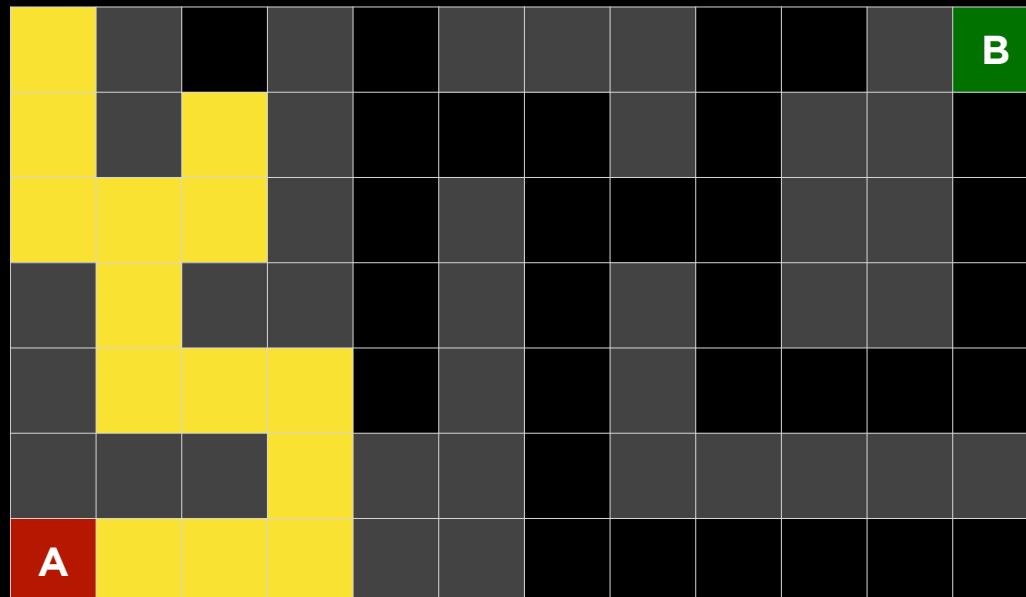
## Depth-First Search



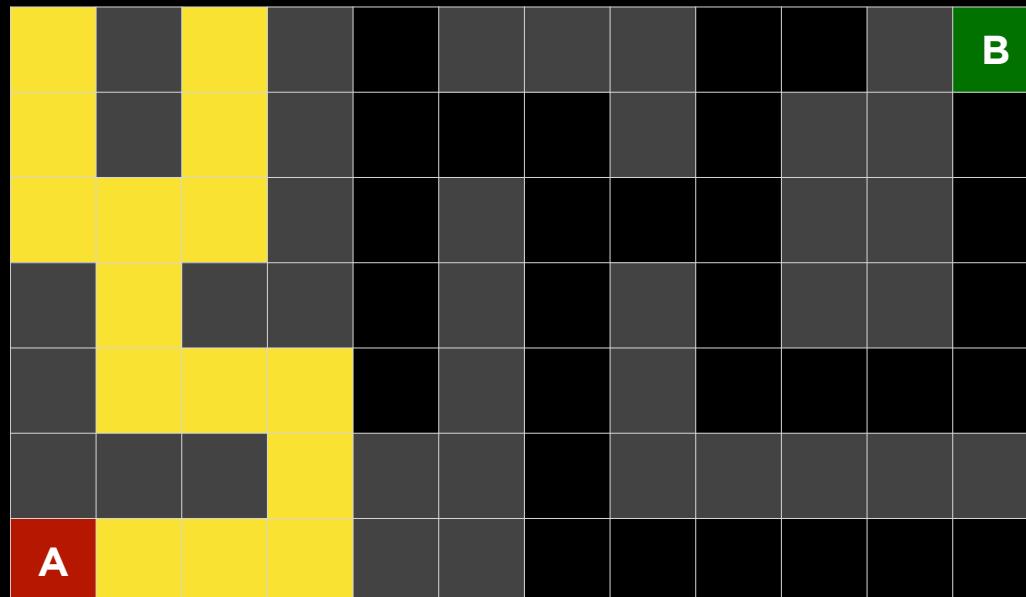
## Depth-First Search



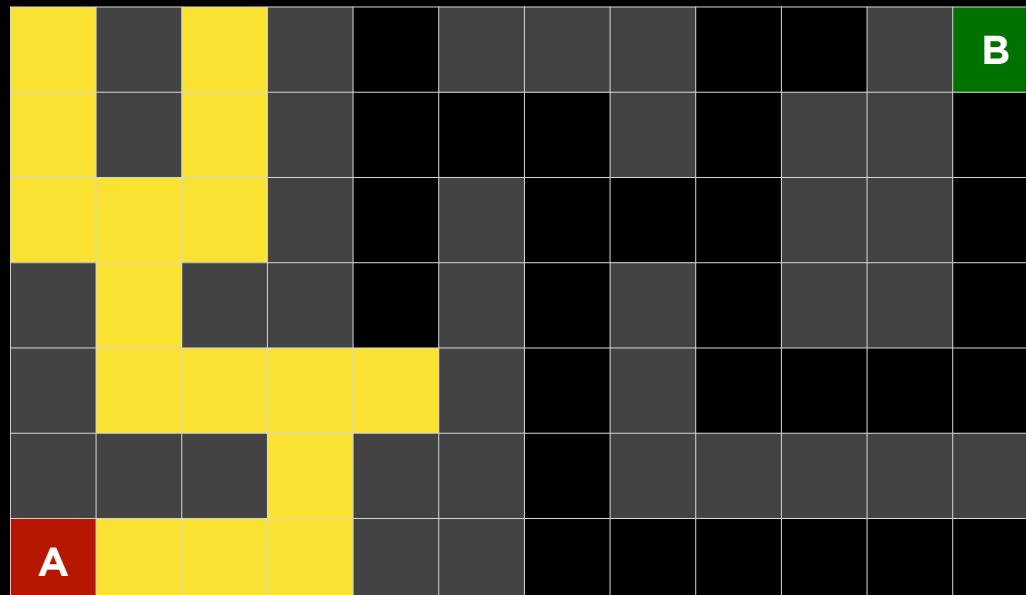
## Depth-First Search



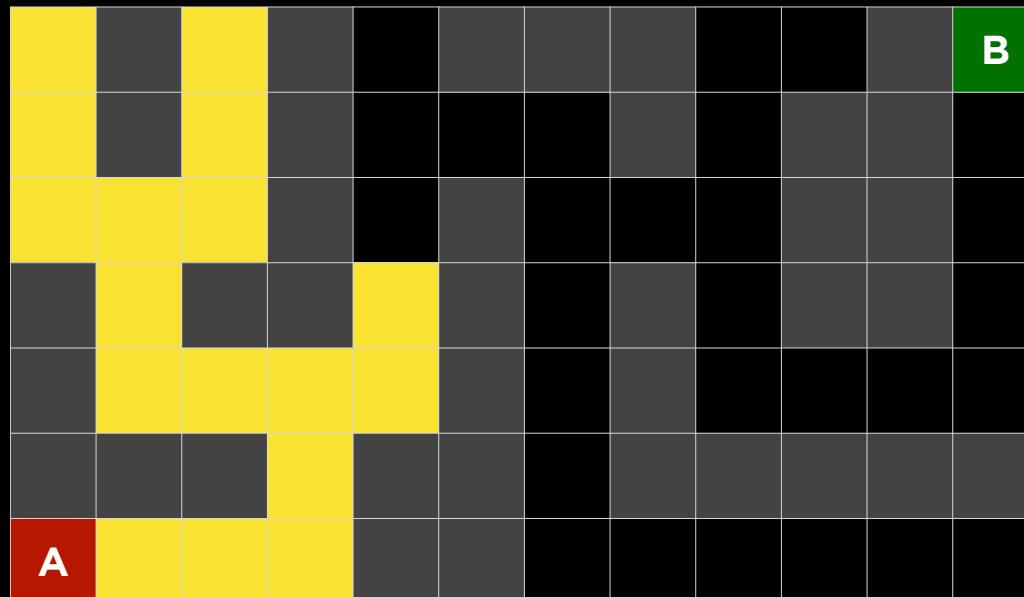
## Depth-First Search



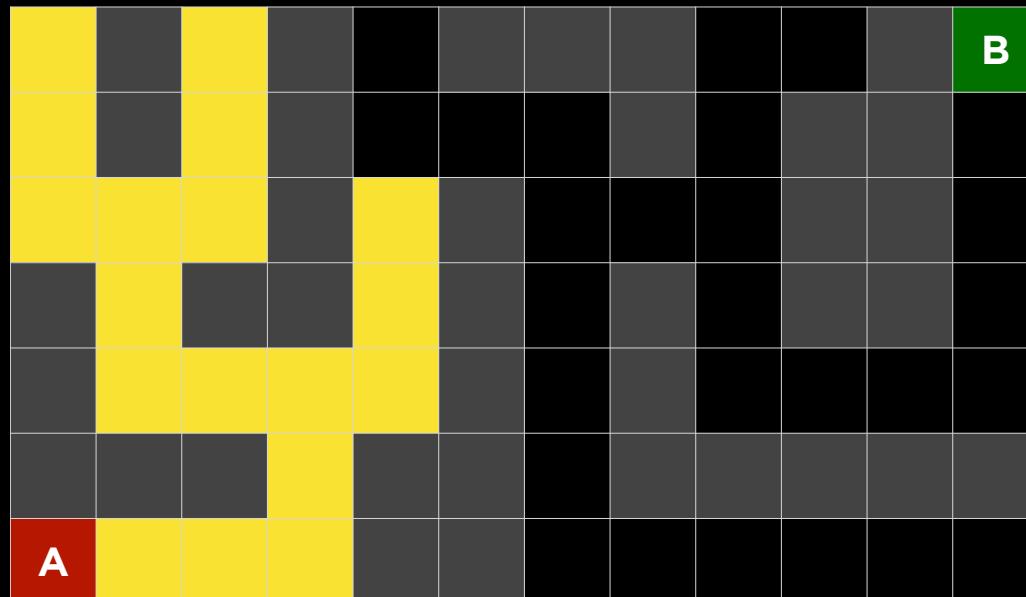
## Depth-First Search



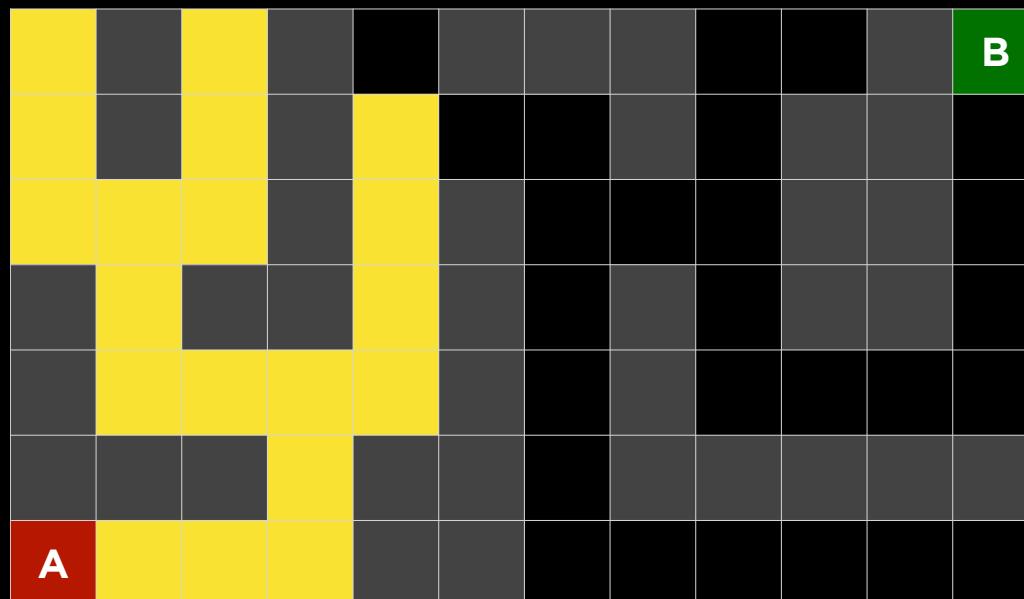
## Depth-First Search



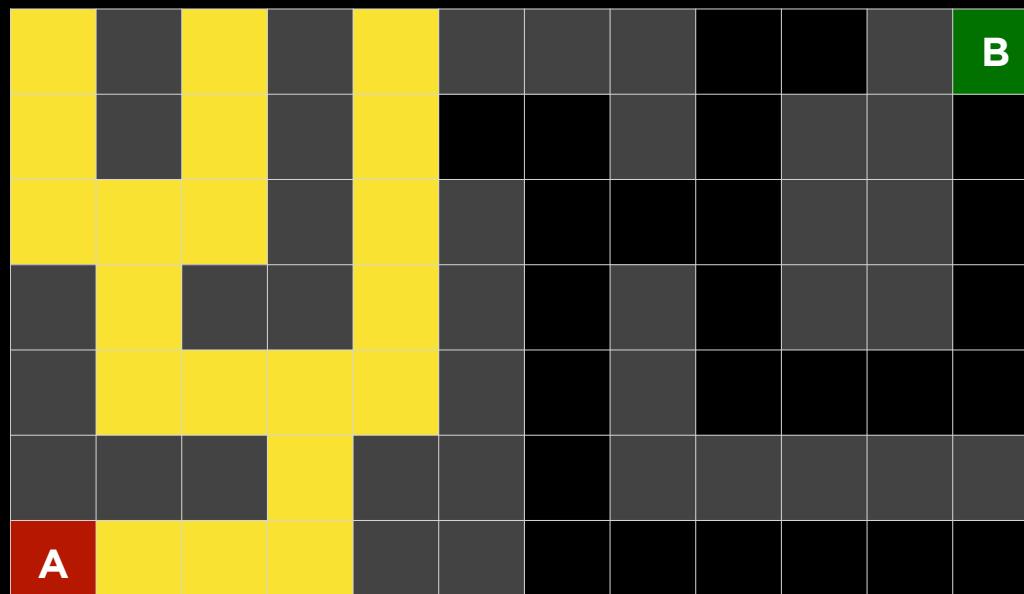
## Depth-First Search



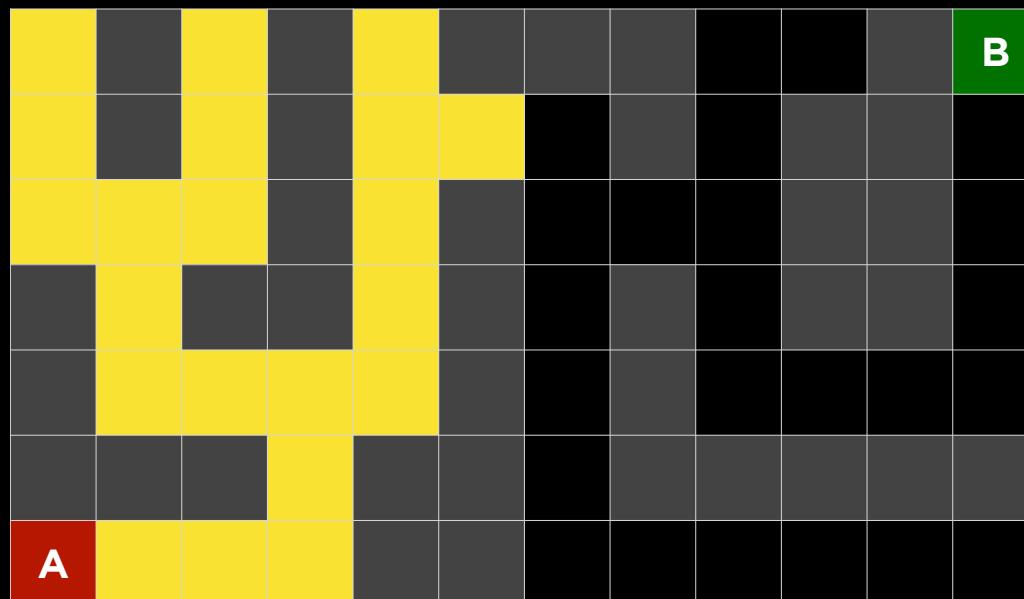
## Depth-First Search



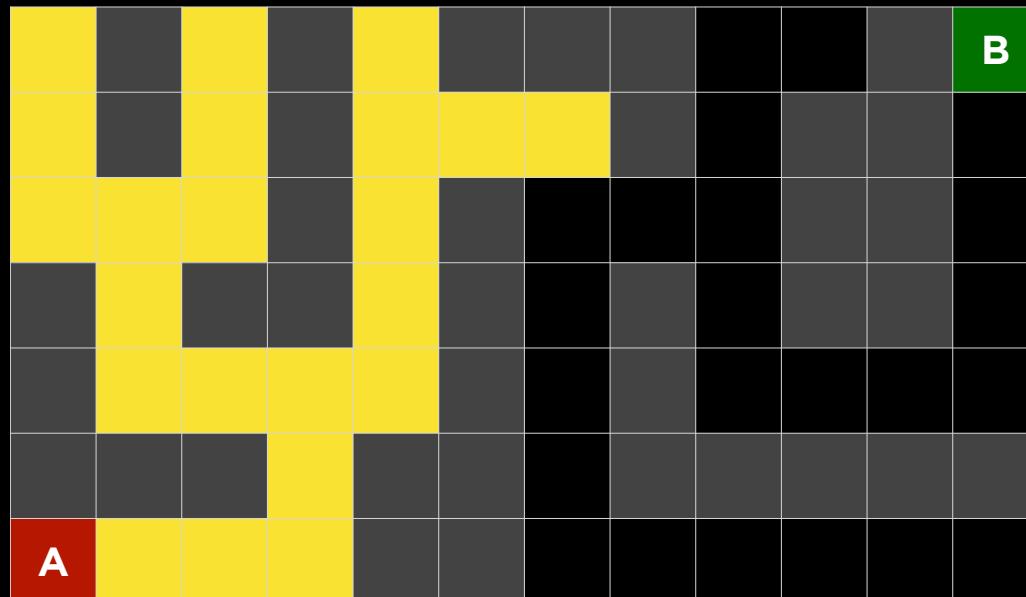
## Depth-First Search



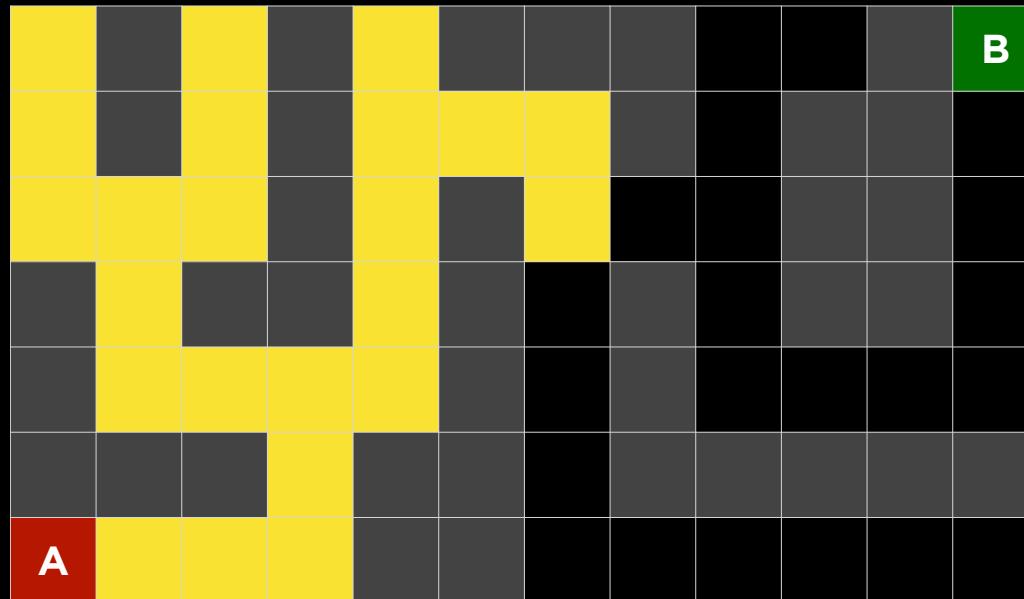
## Depth-First Search



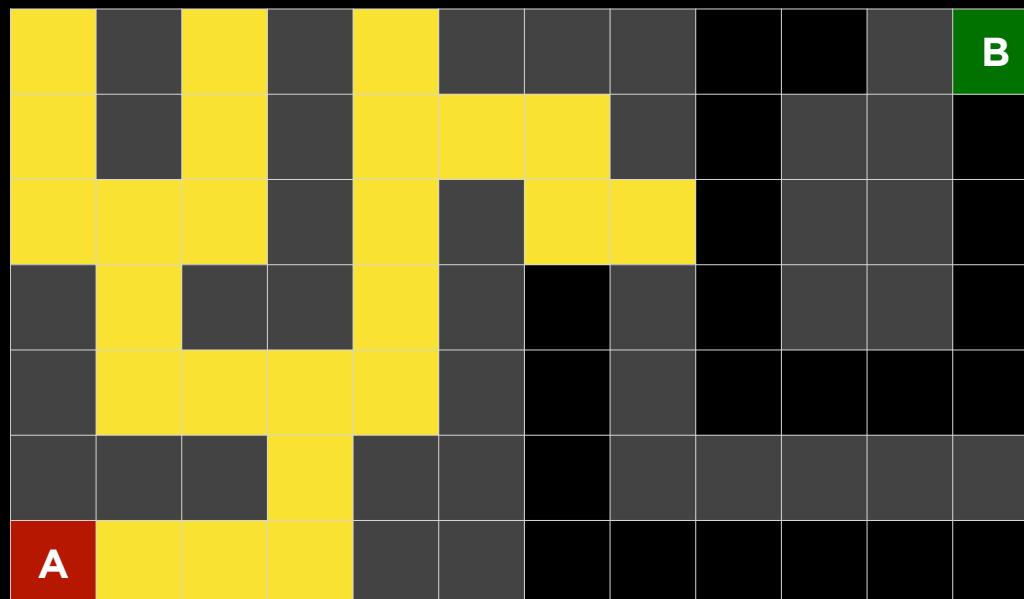
## Depth-First Search



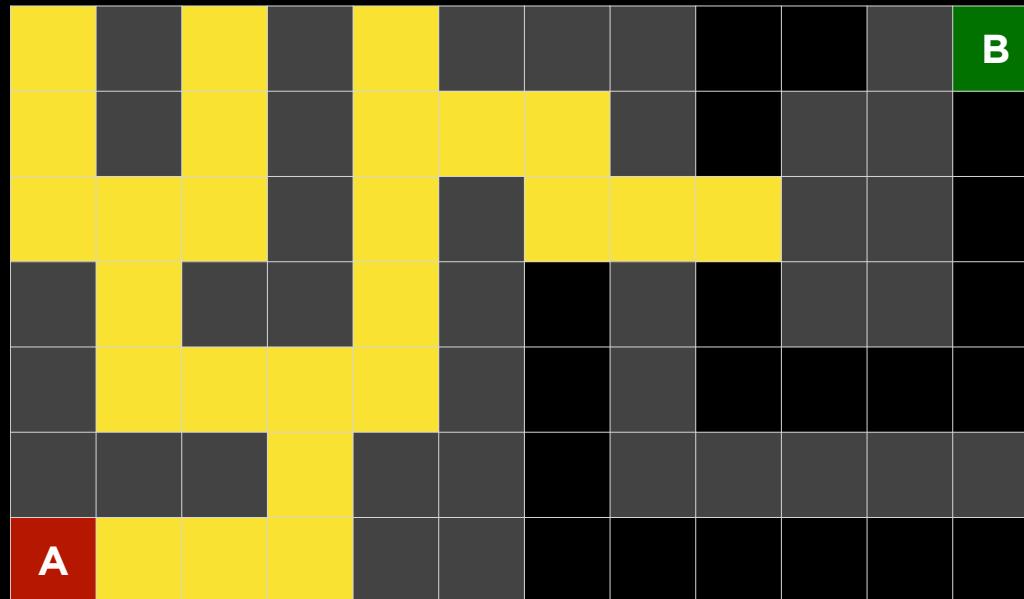
## Depth-First Search



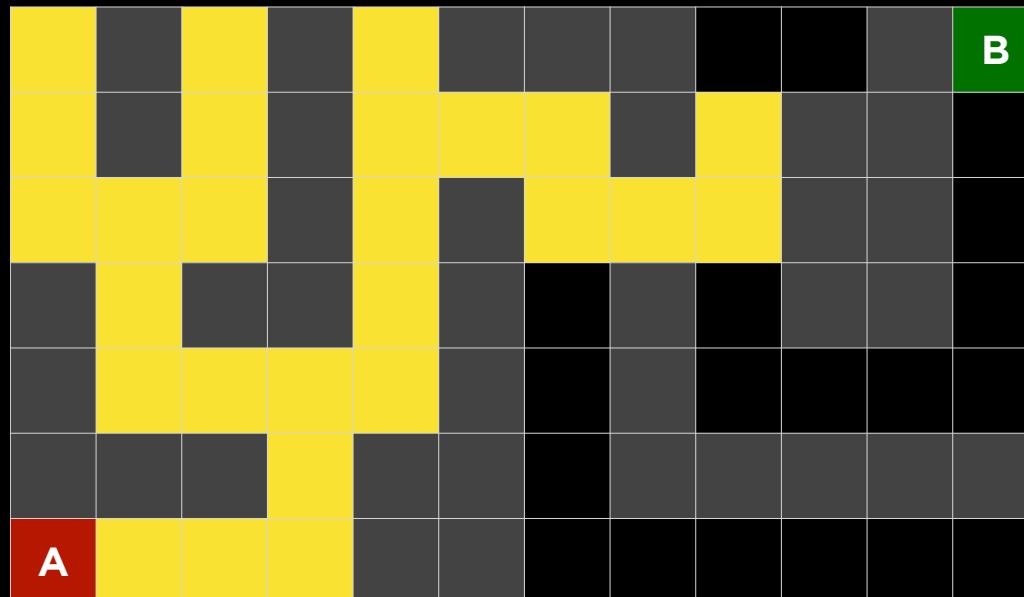
# Depth-First Search



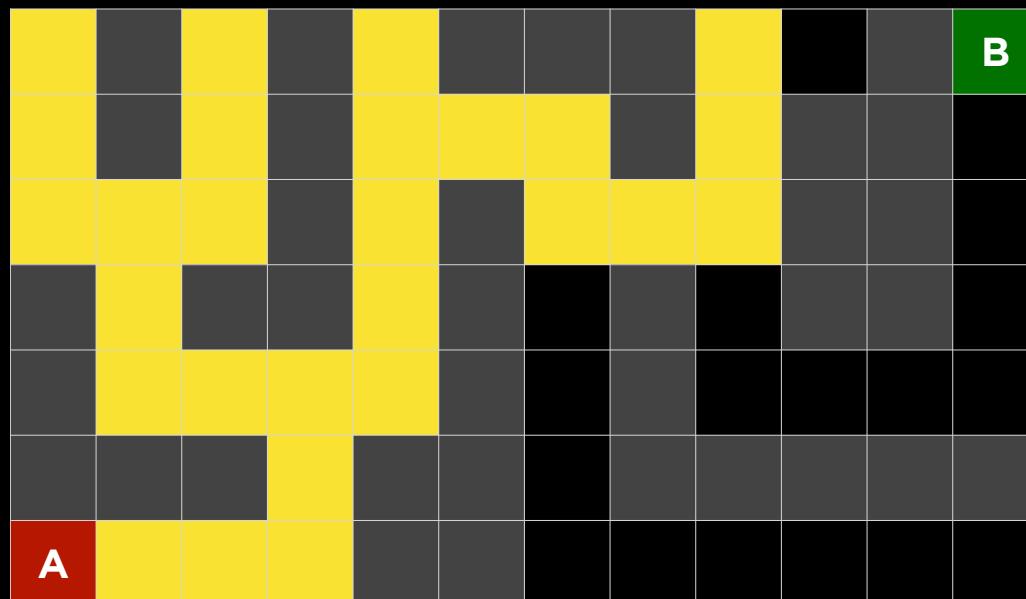
## Depth-First Search



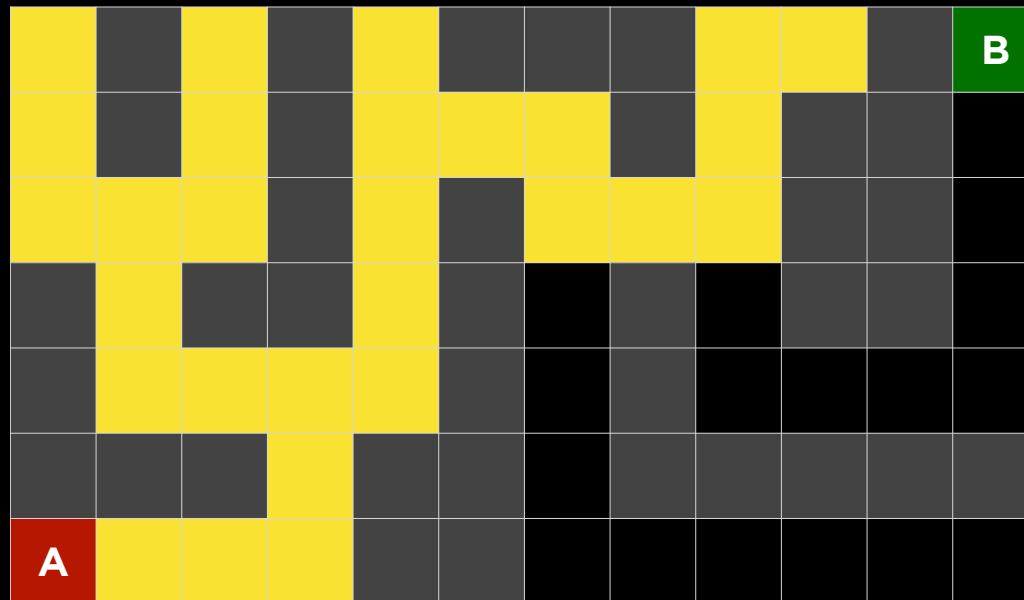
## Depth-First Search



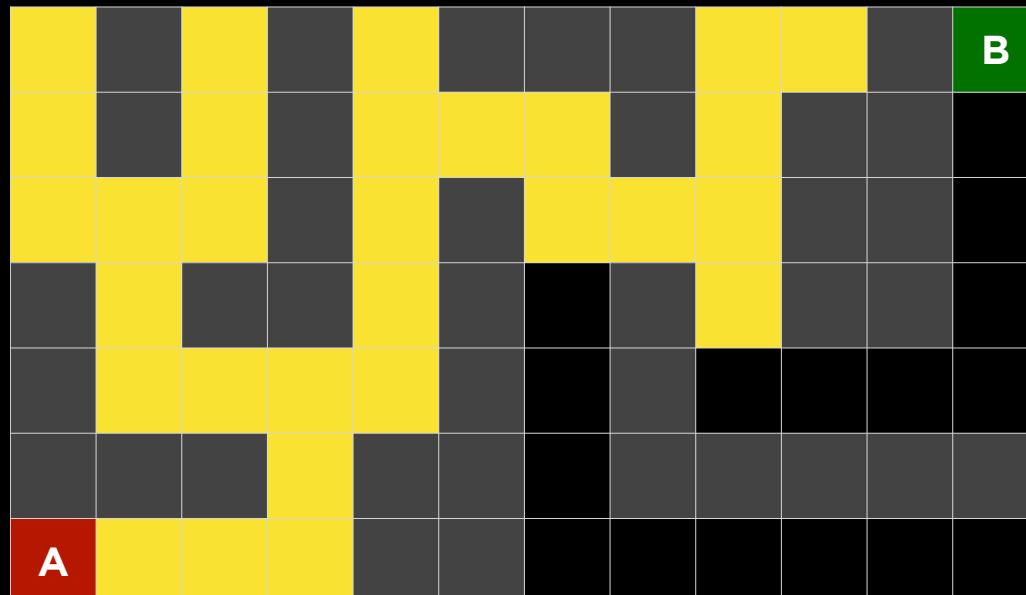
## Depth-First Search



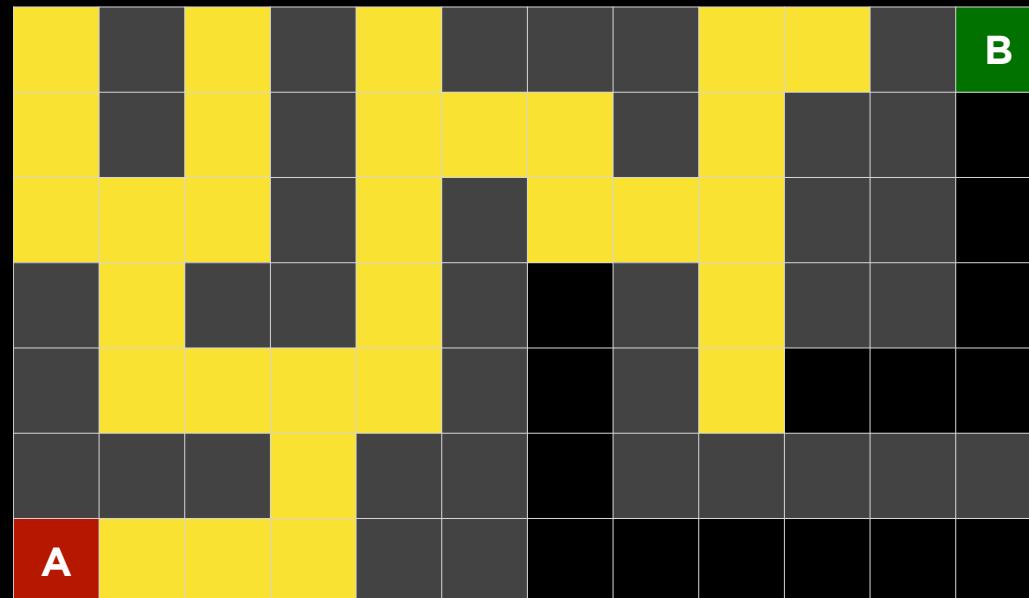
## Depth-First Search



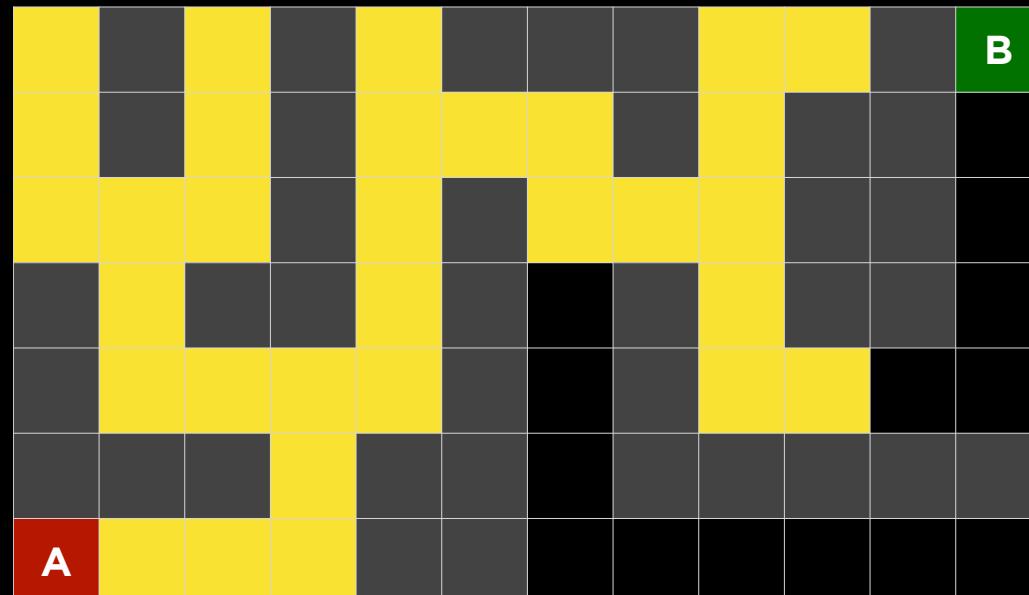
## Depth-First Search



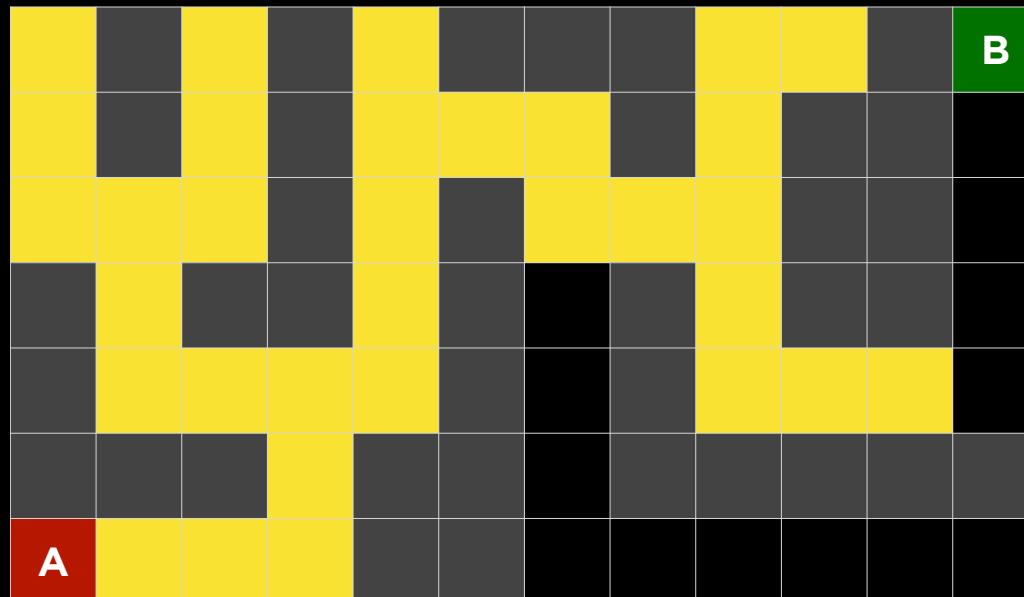
## Depth-First Search



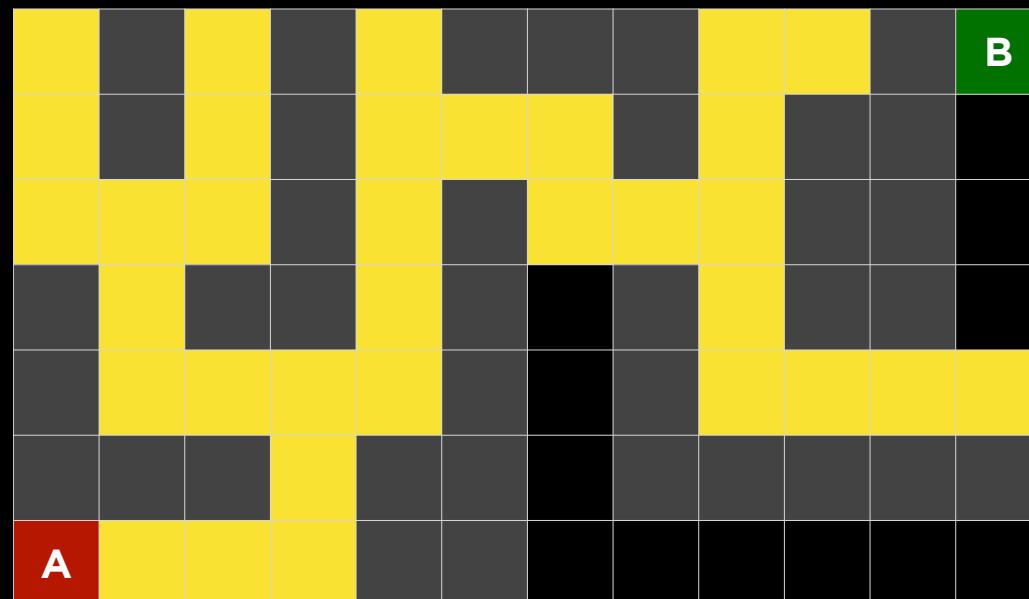
## Depth-First Search



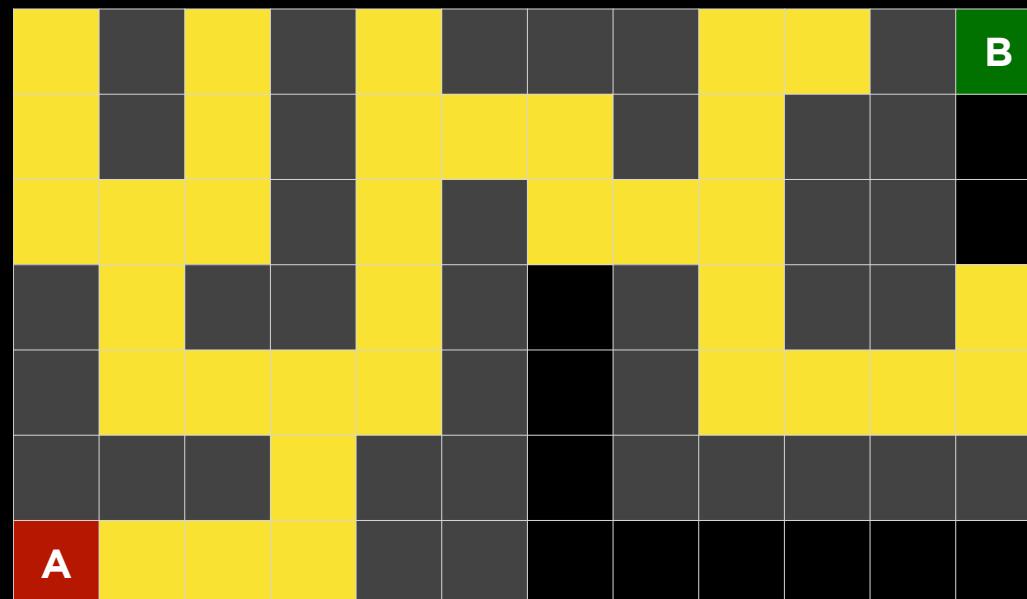
## Depth-First Search



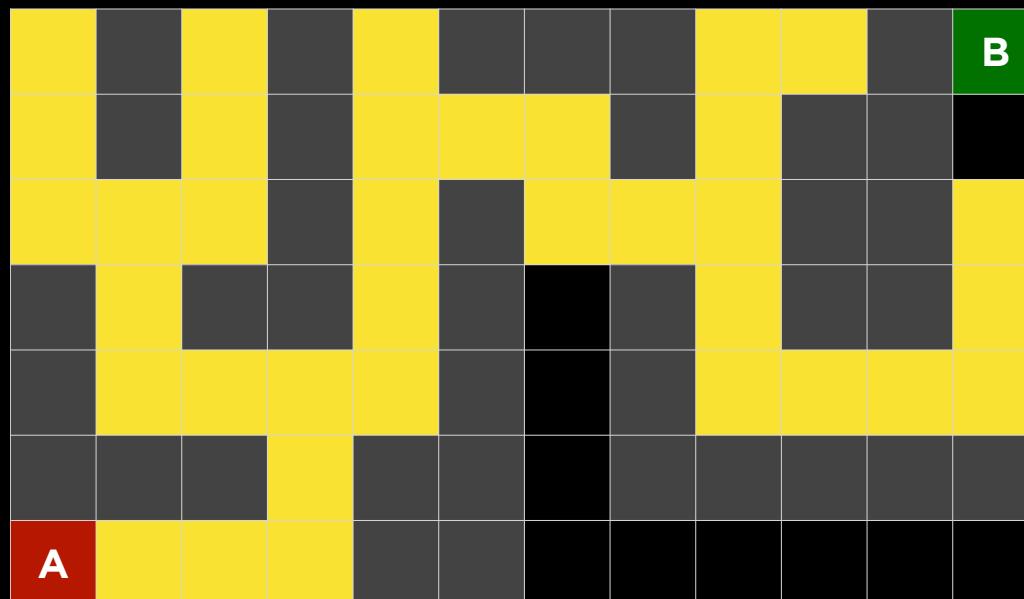
## Depth-First Search



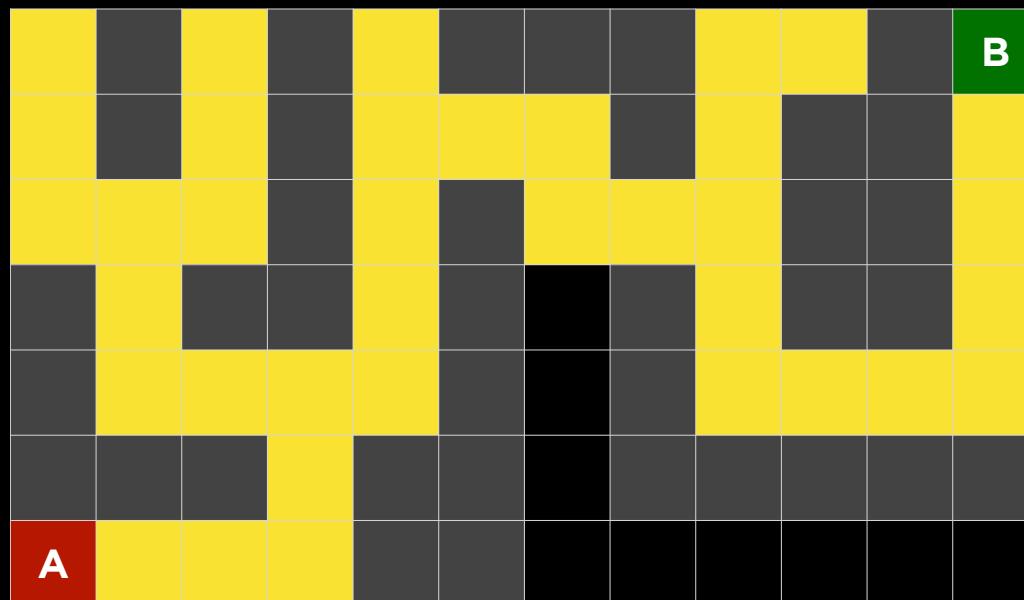
## Depth-First Search



## Depth-First Search



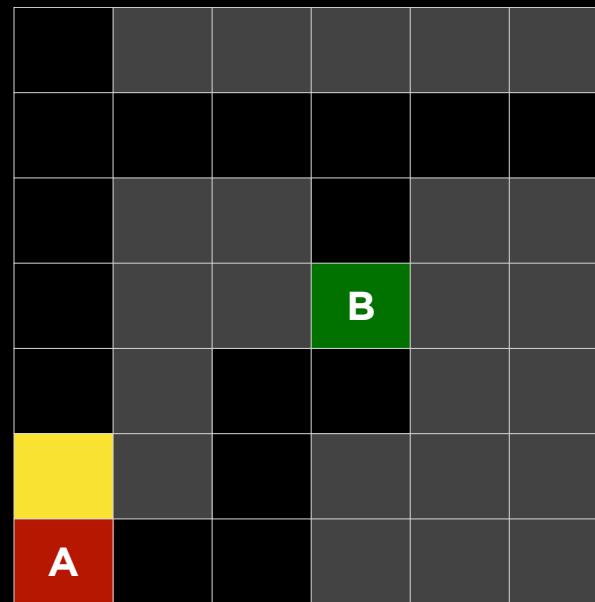
## Depth-First Search



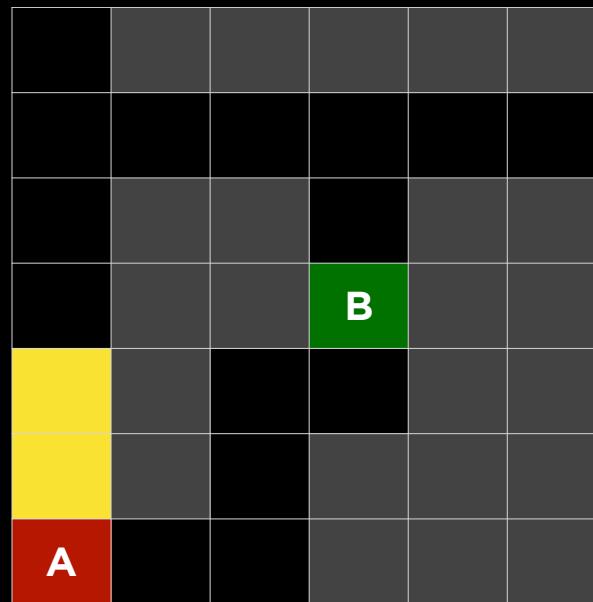
## Depth-First Search



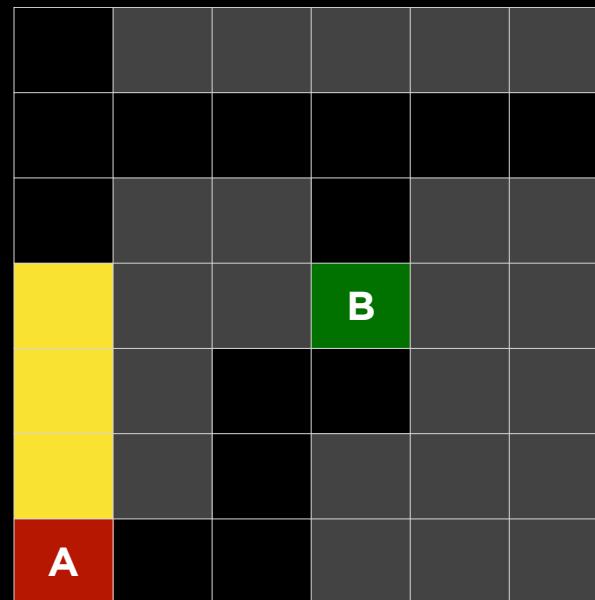
## Depth-First Search



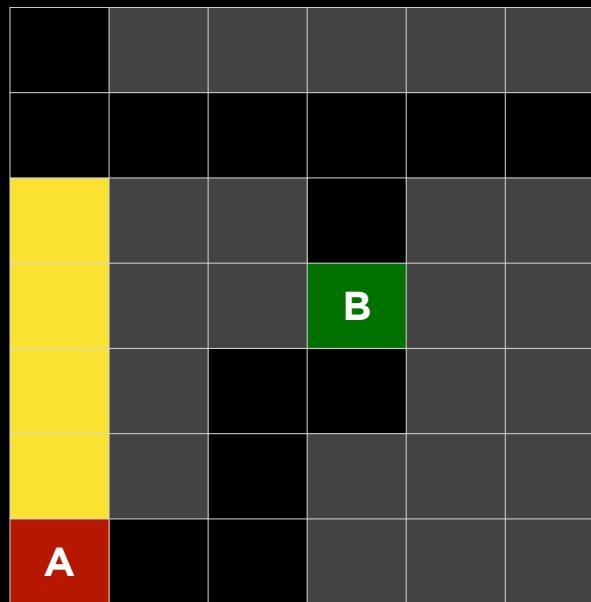
## Depth-First Search



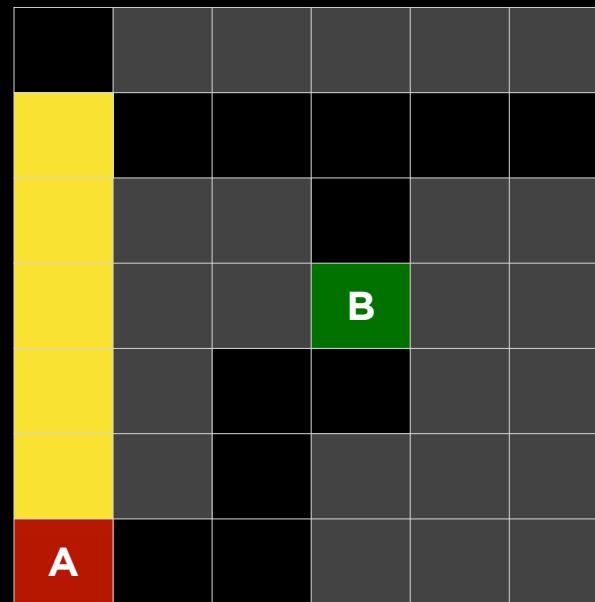
## Depth-First Search



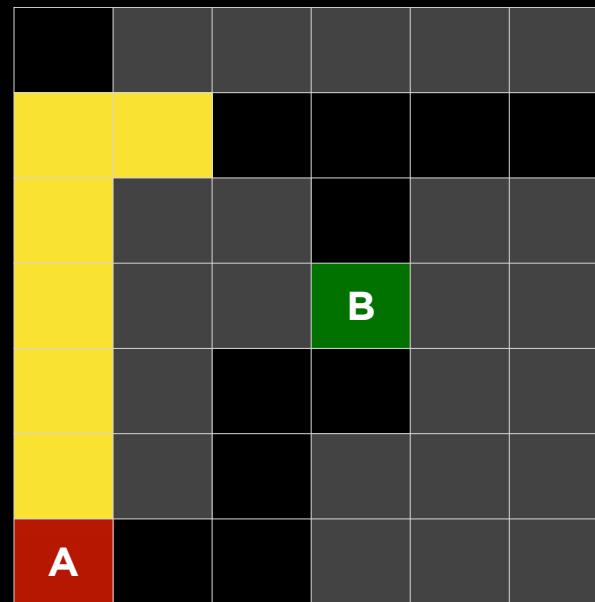
# Depth-First Search



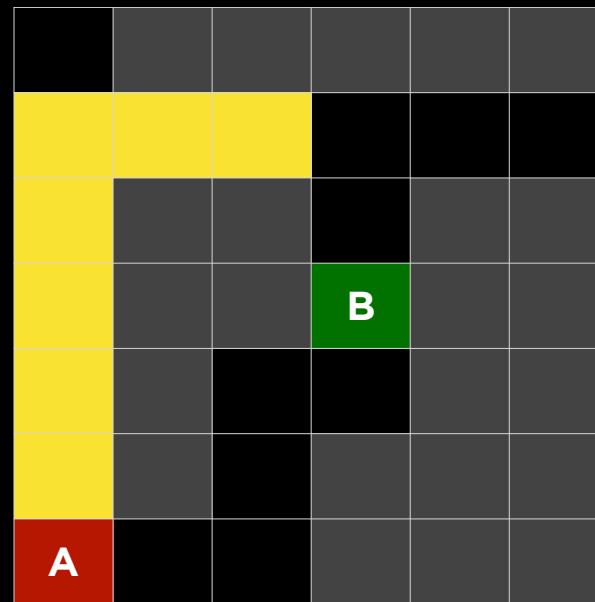
## Depth-First Search



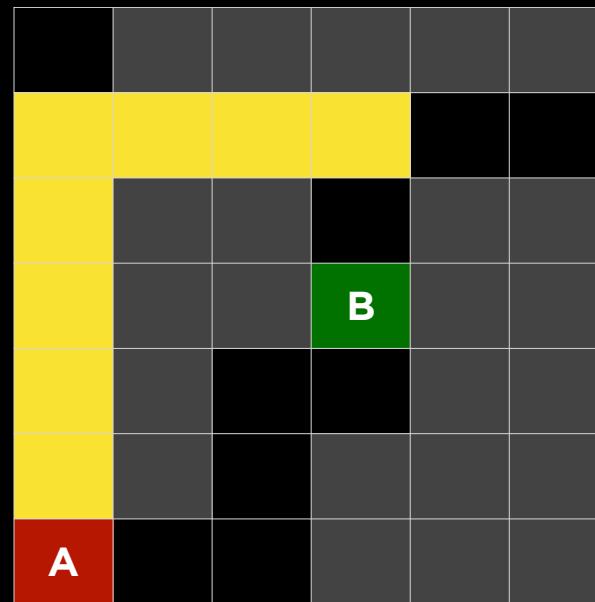
## Depth-First Search



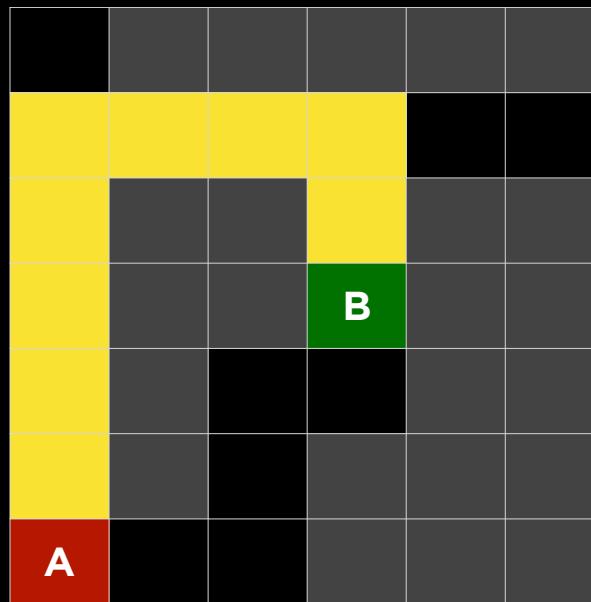
## Depth-First Search



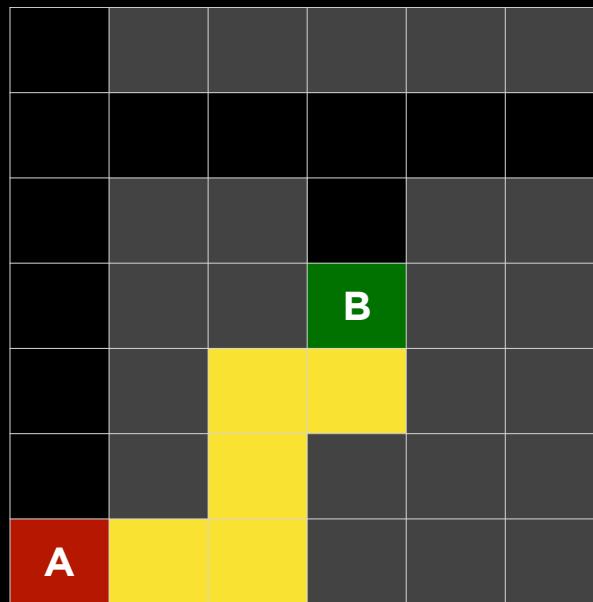
## Depth-First Search



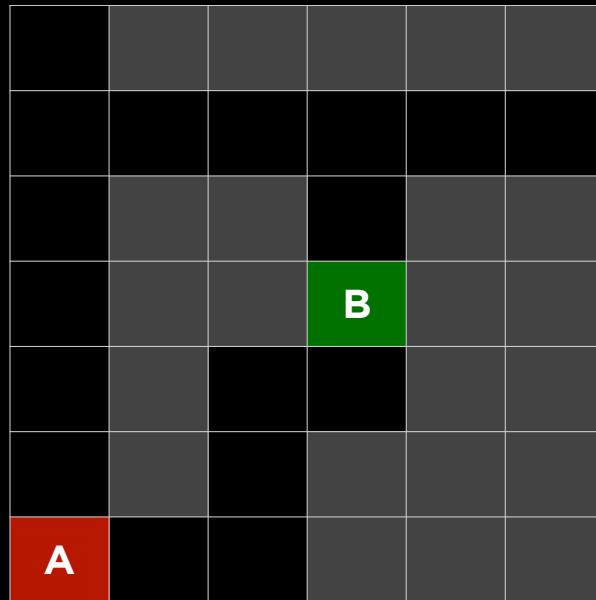
## Depth-First Search



## Depth-First Search



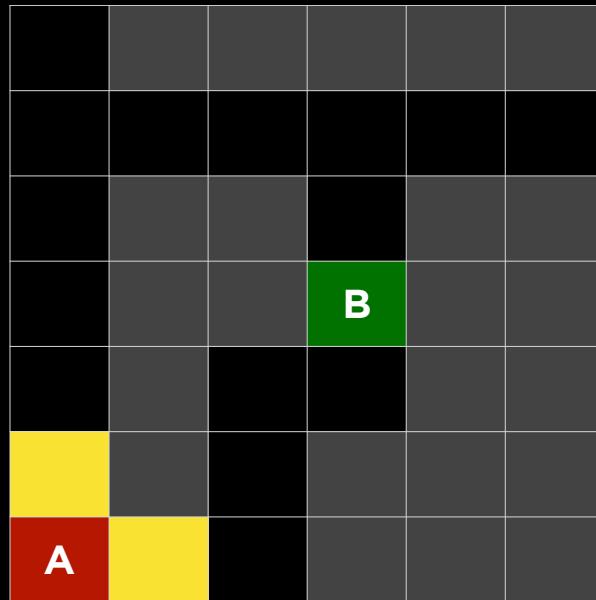
## Breadth-First Search



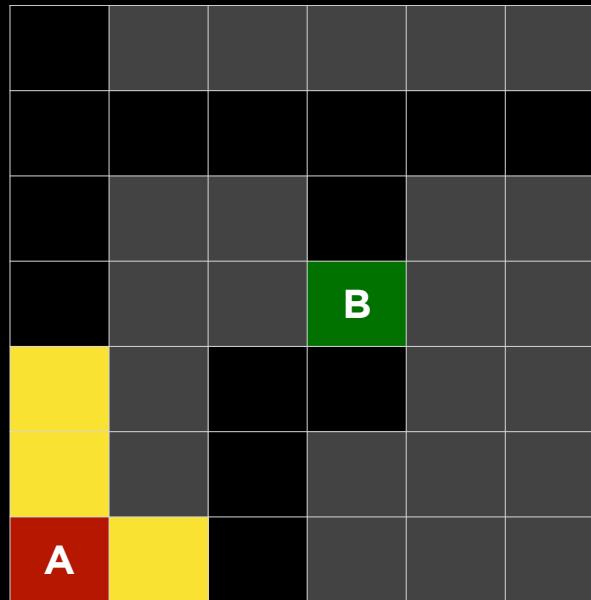
## Breadth-First Search



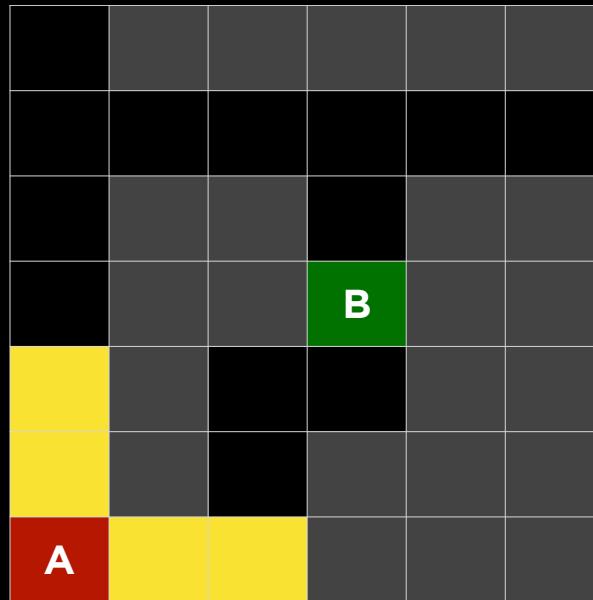
## Breadth-First Search



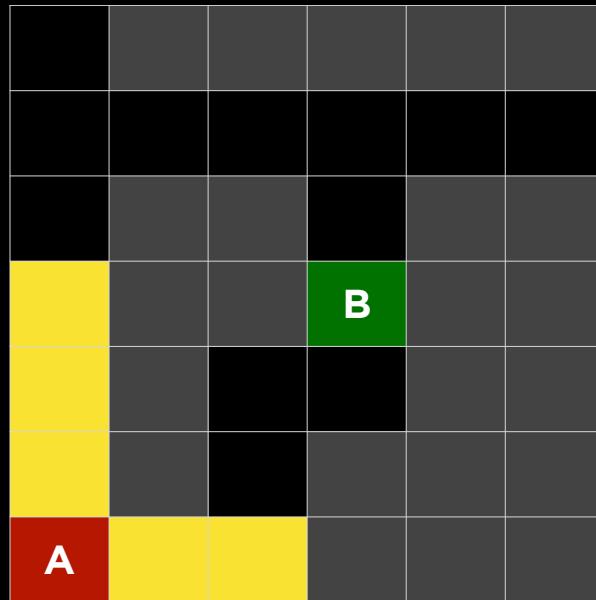
## Breadth-First Search



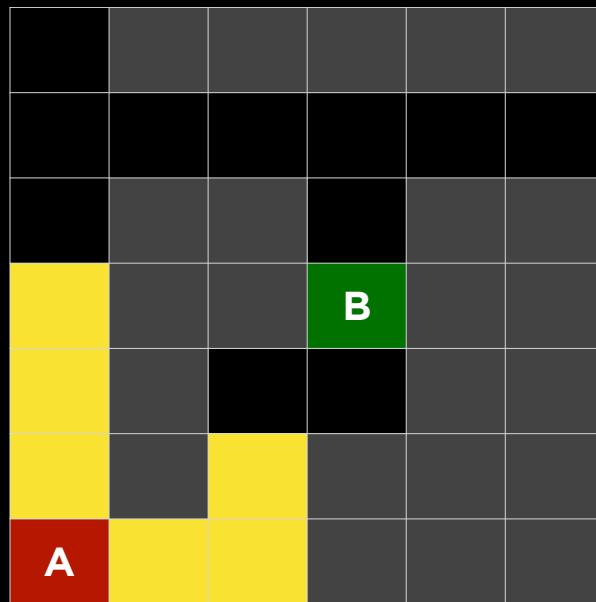
## Breadth-First Search



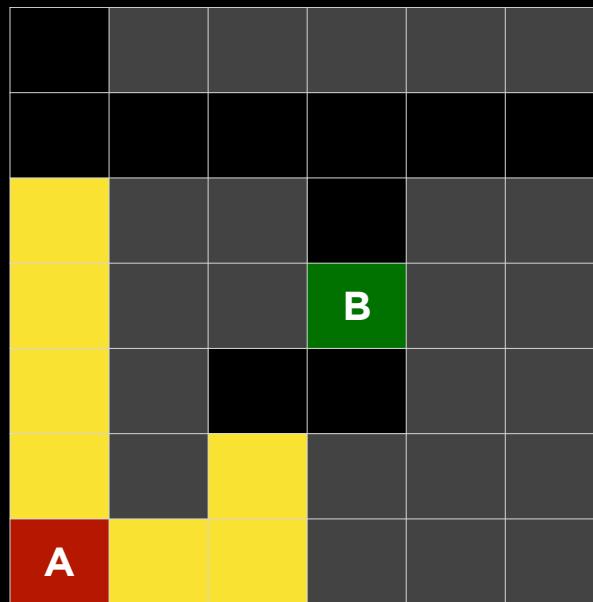
## Breadth-First Search



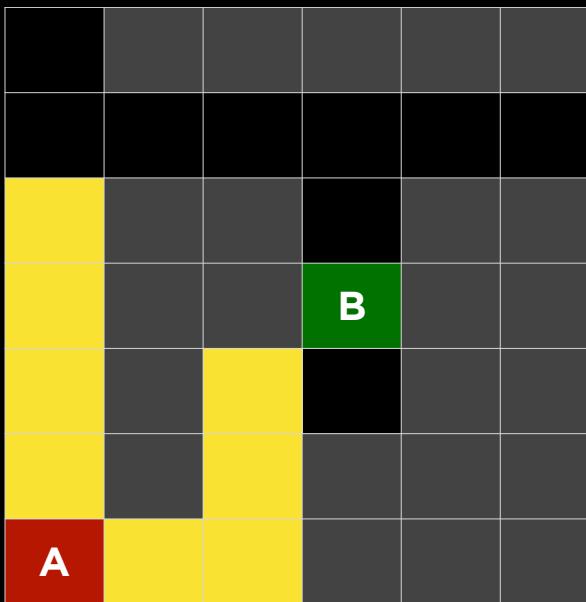
## Breadth-First Search



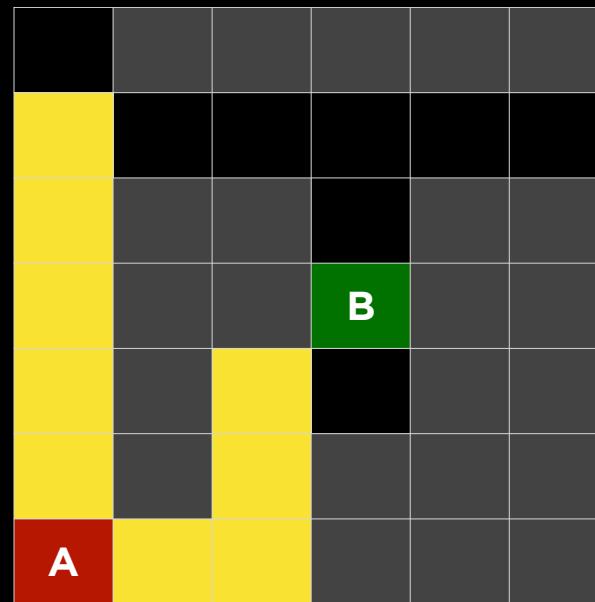
## Breadth-First Search



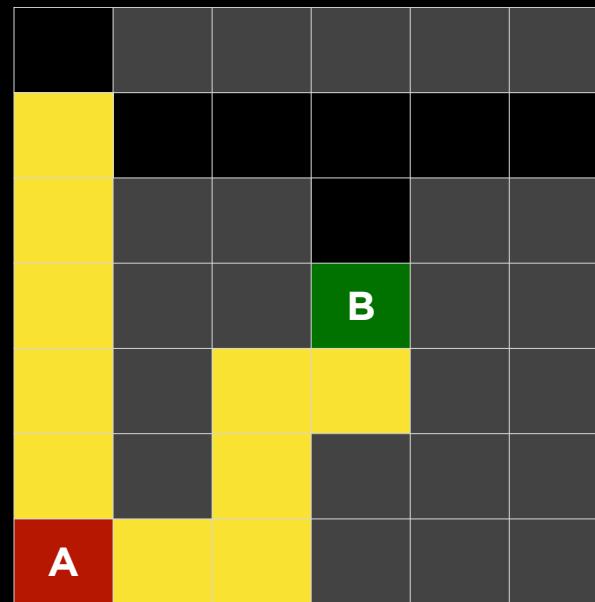
## Breadth-First Search



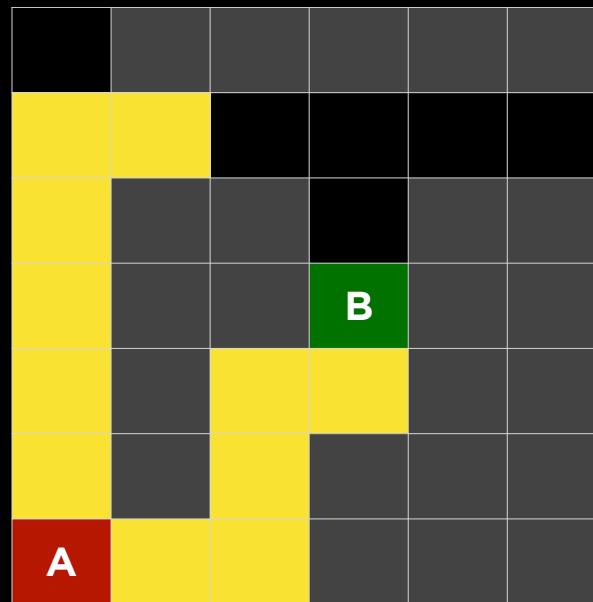
## Breadth-First Search



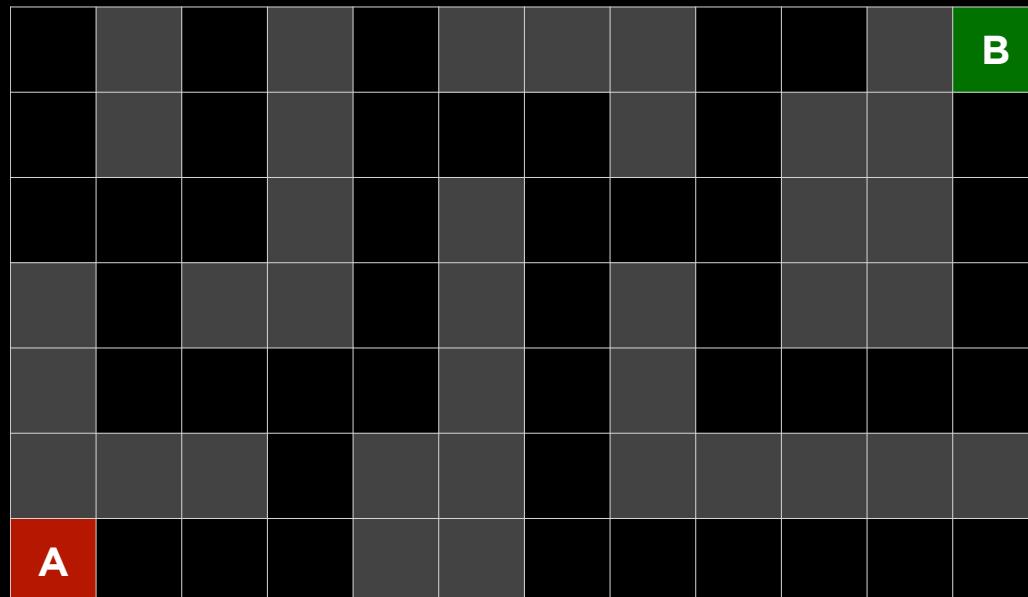
## Breadth-First Search



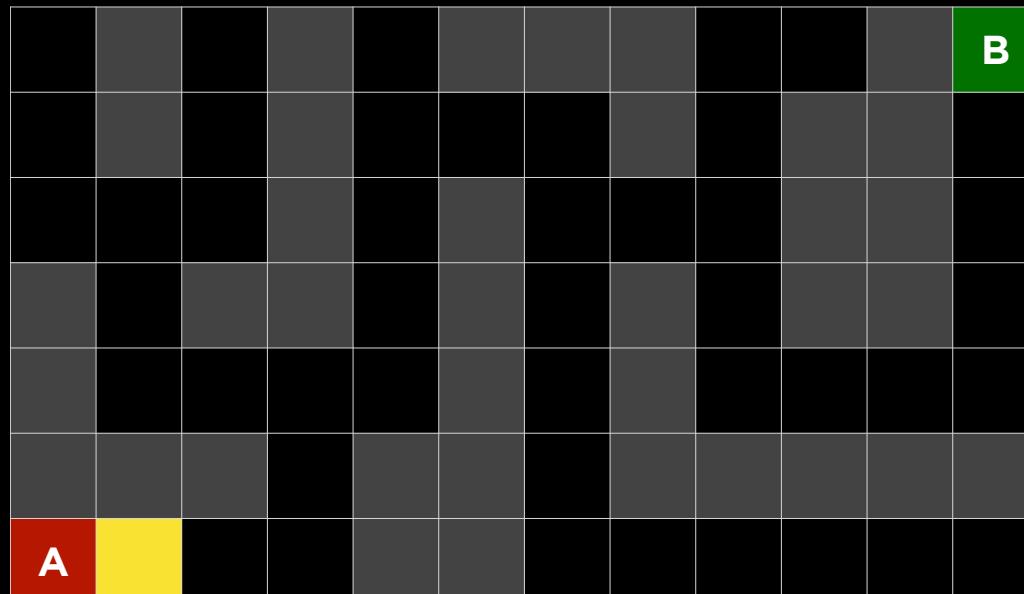
## Breadth-First Search



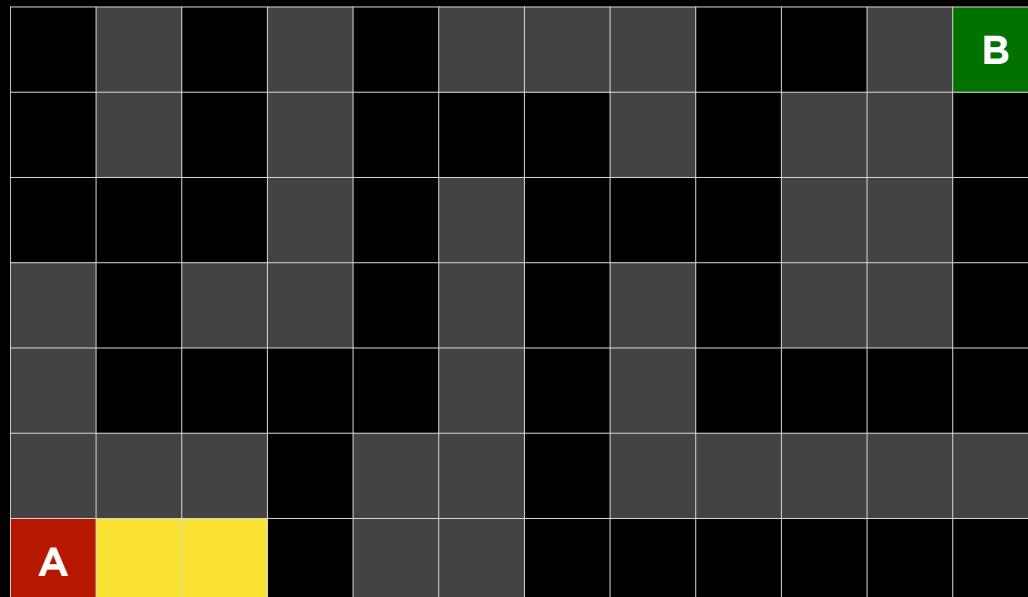
## Breadth-First Search



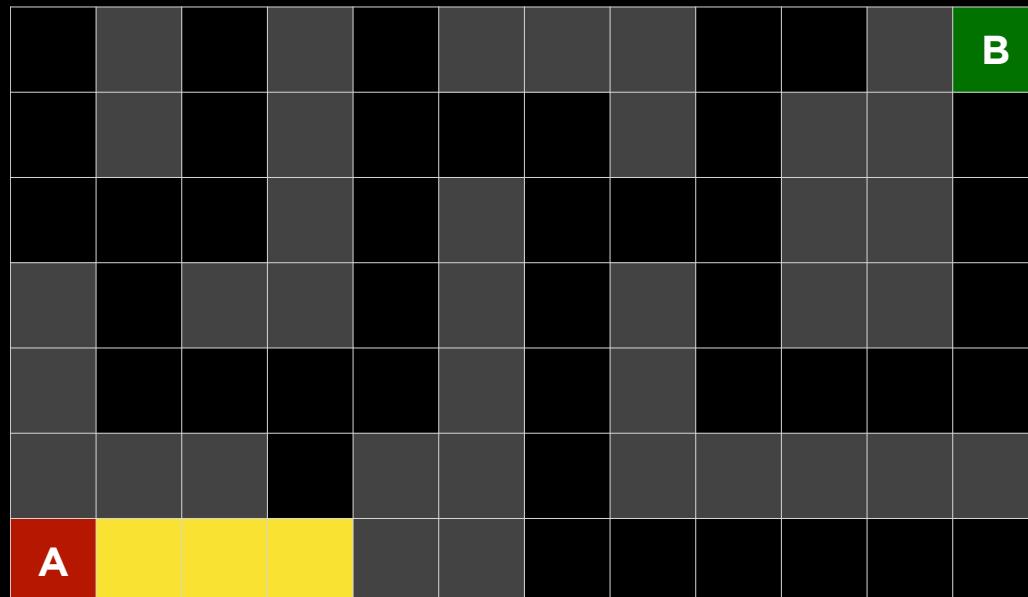
## Breadth-First Search



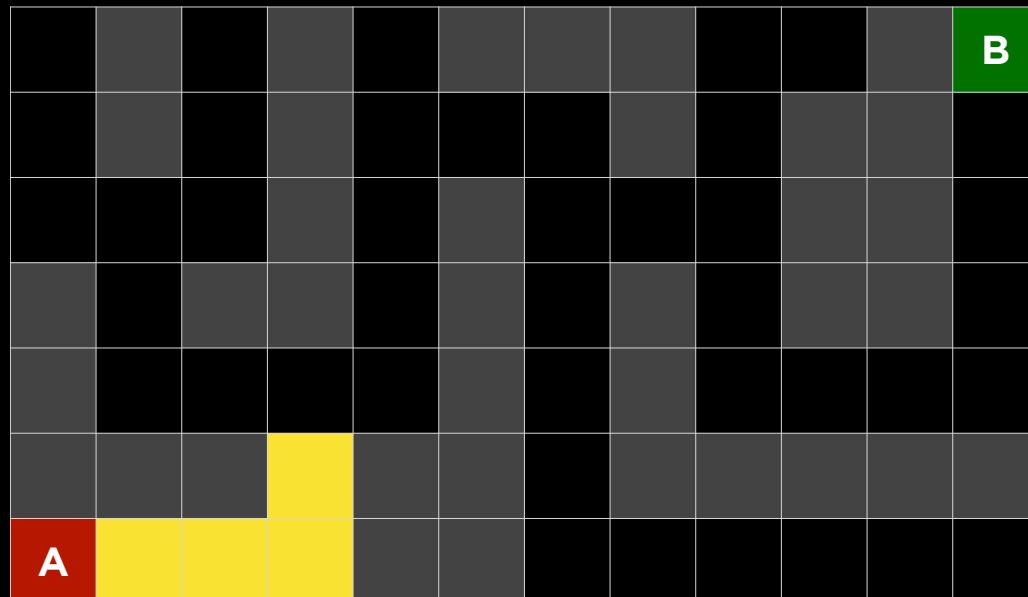
## Breadth-First Search



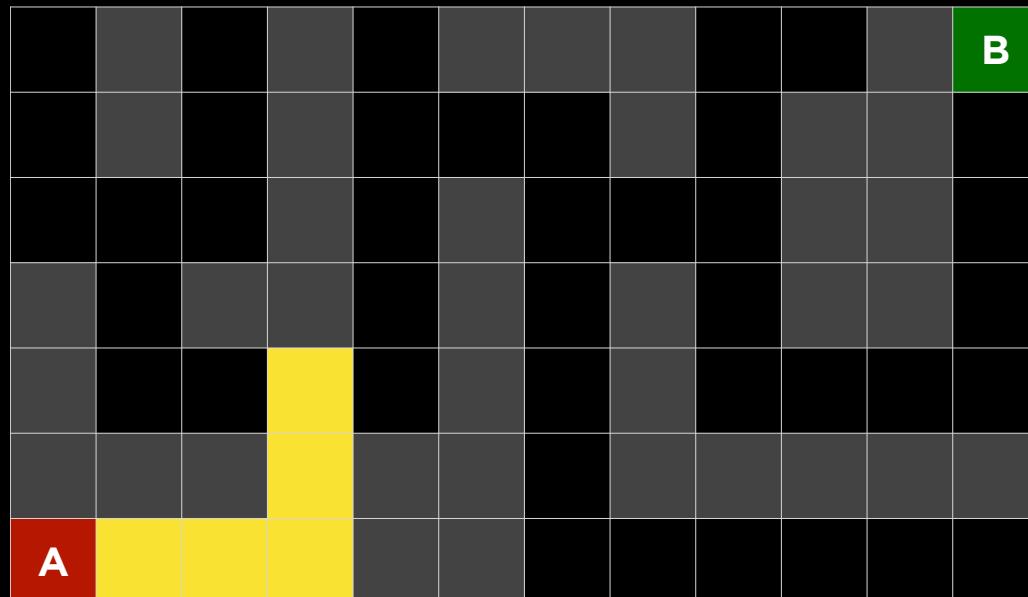
## Breadth-First Search



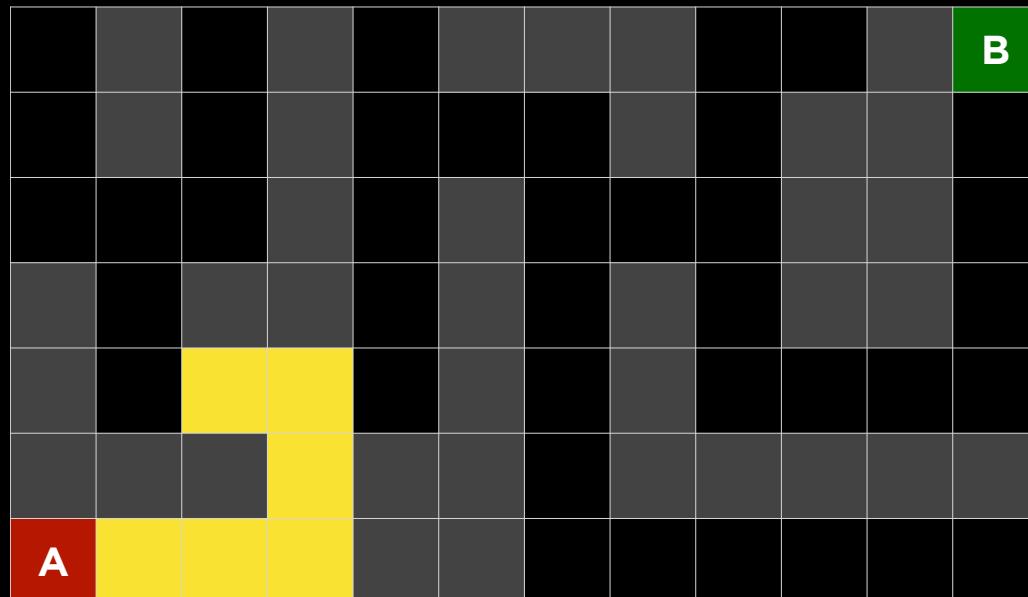
## Breadth-First Search



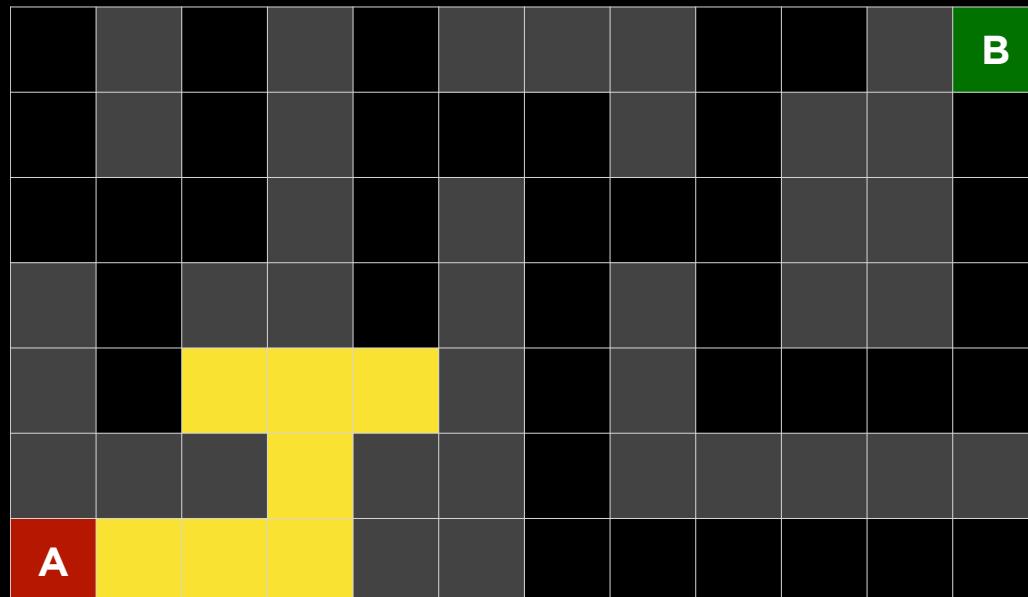
## Breadth-First Search



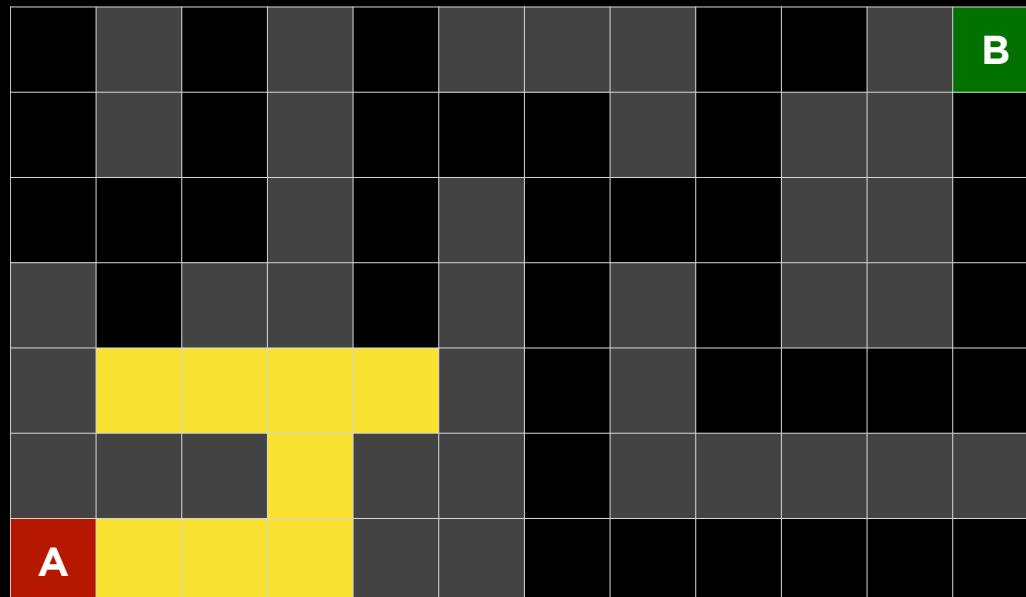
## Breadth-First Search



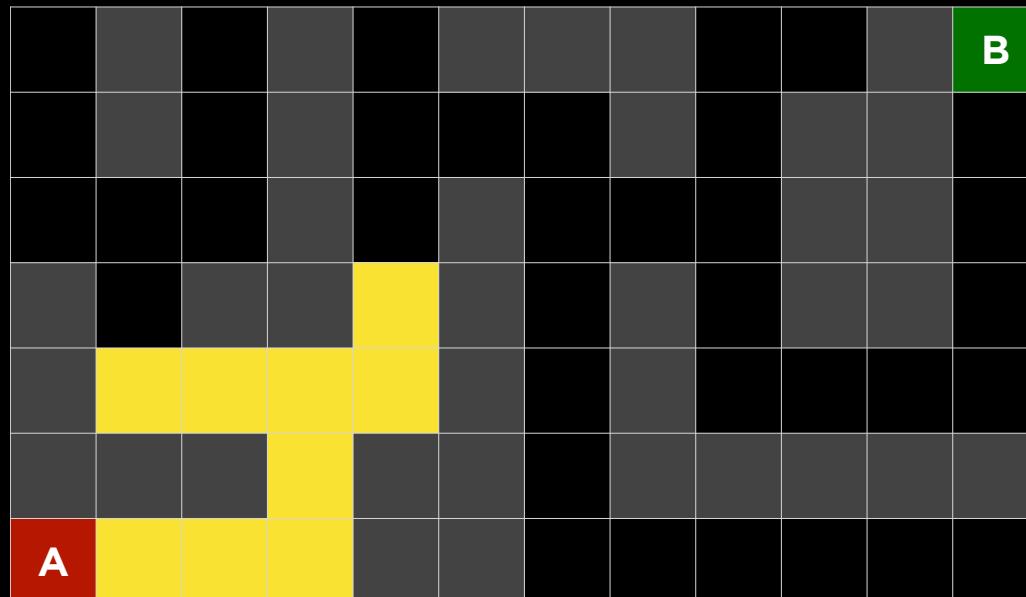
## Breadth-First Search



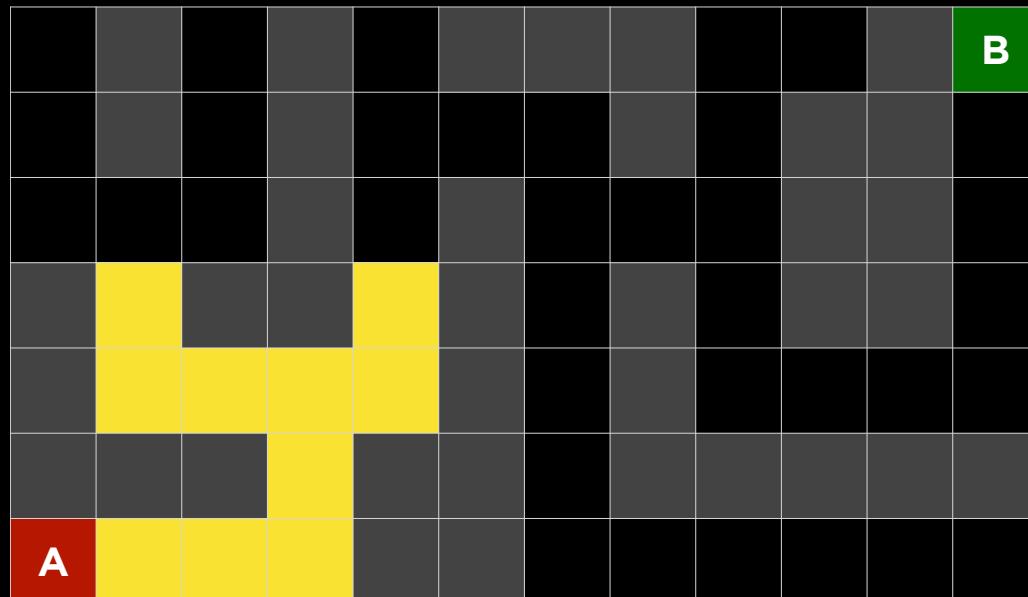
## Breadth-First Search



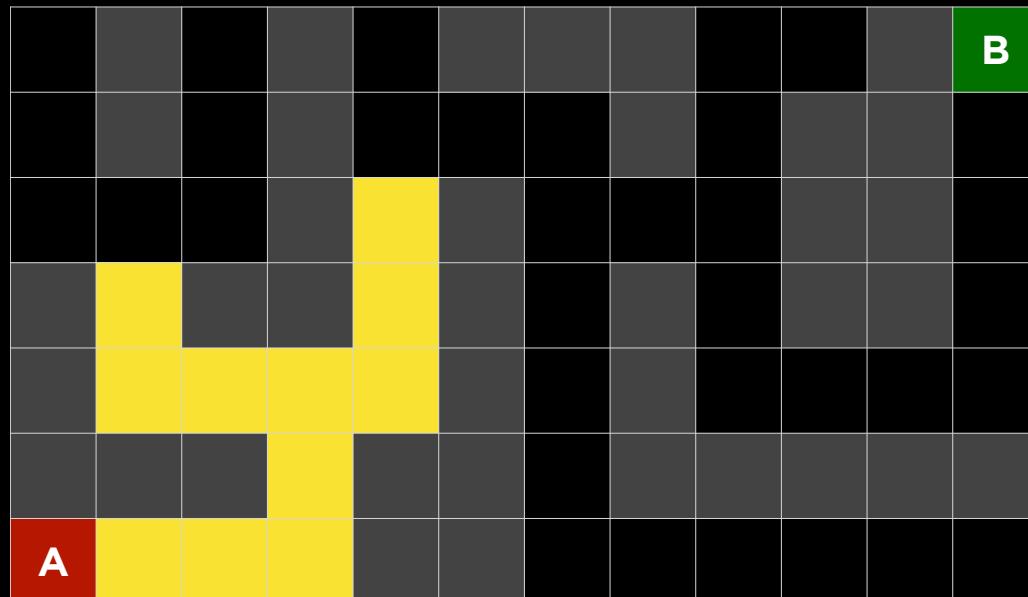
## Breadth-First Search



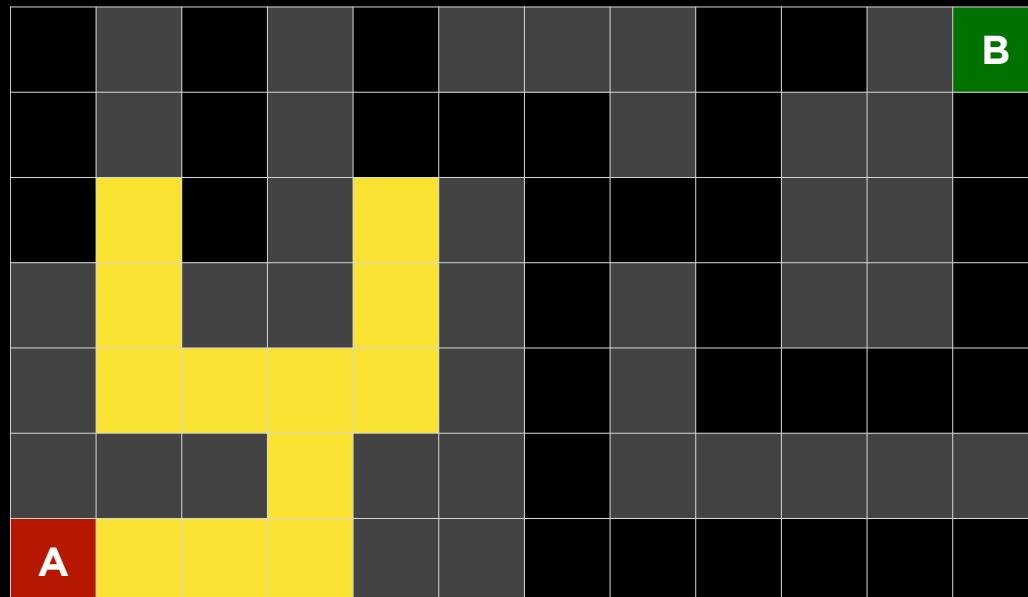
## Breadth-First Search



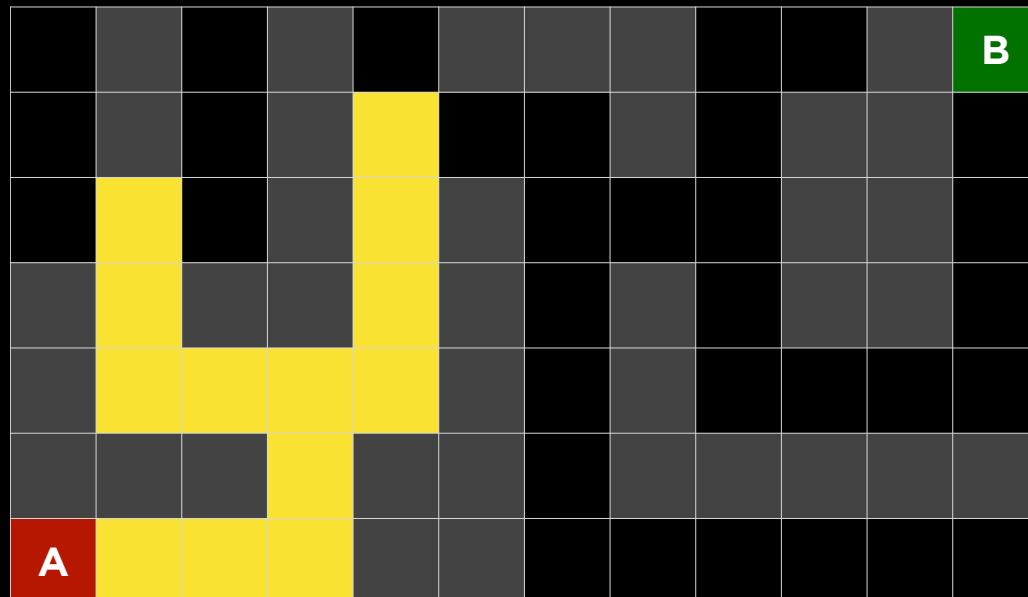
## Breadth-First Search



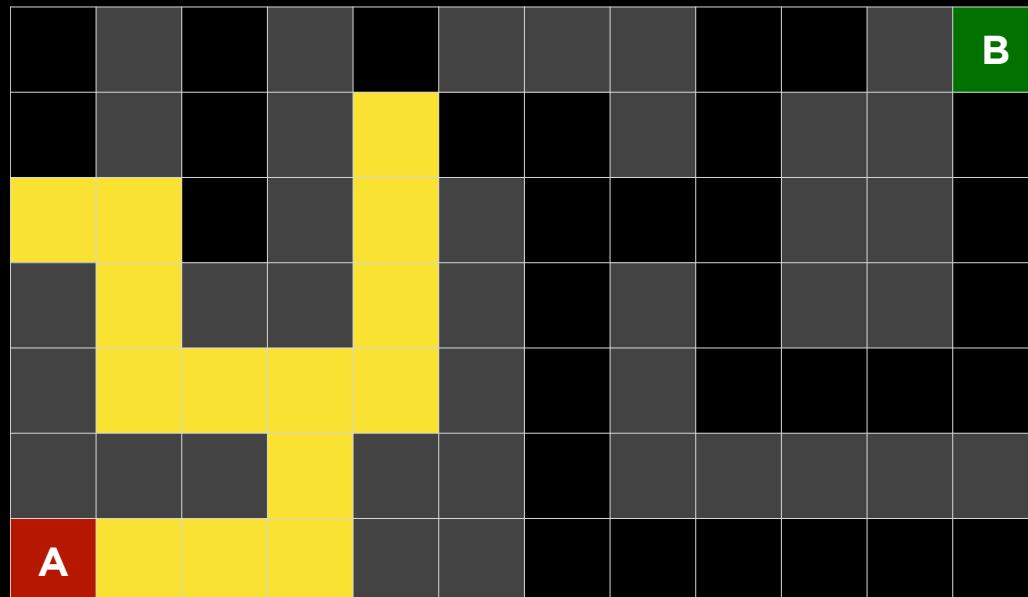
## Breadth-First Search



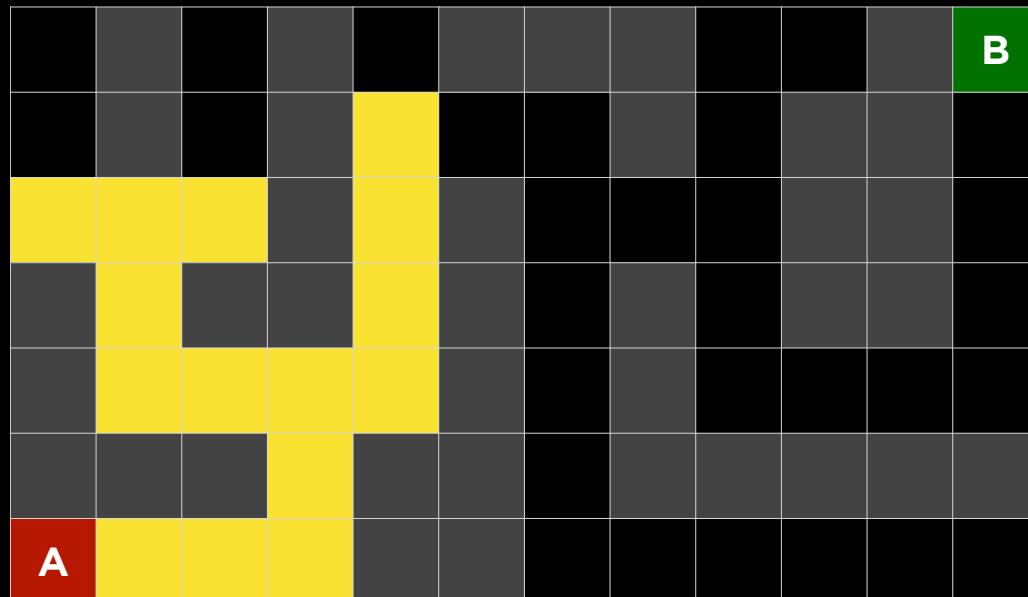
## Breadth-First Search



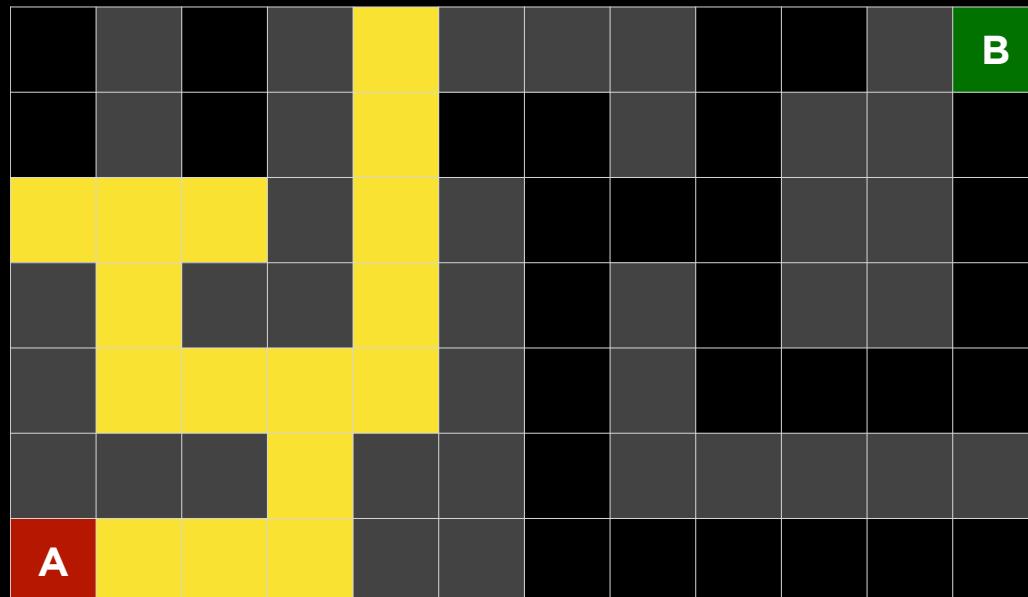
## Breadth-First Search



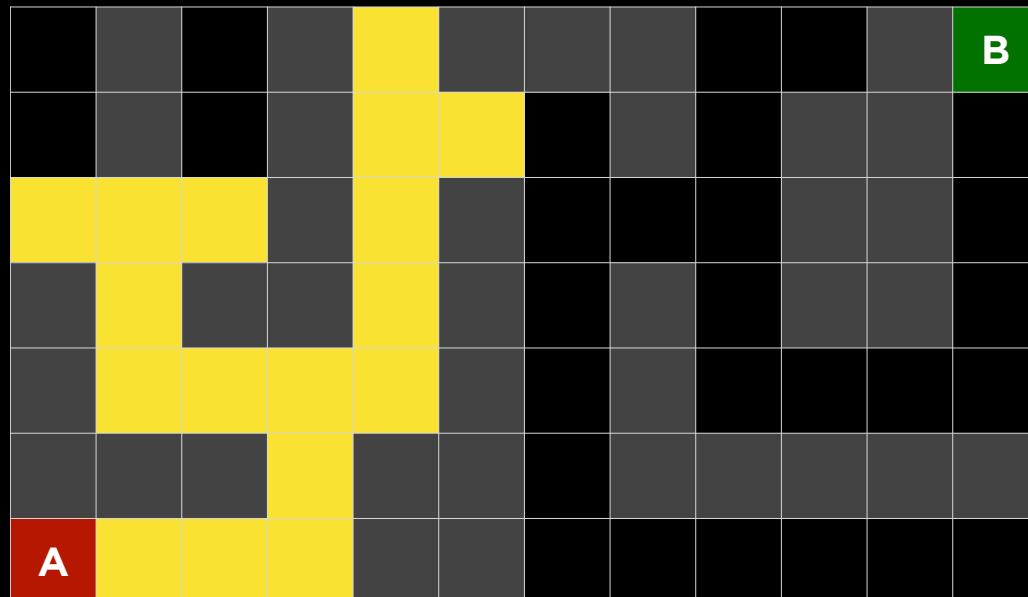
## Breadth-First Search



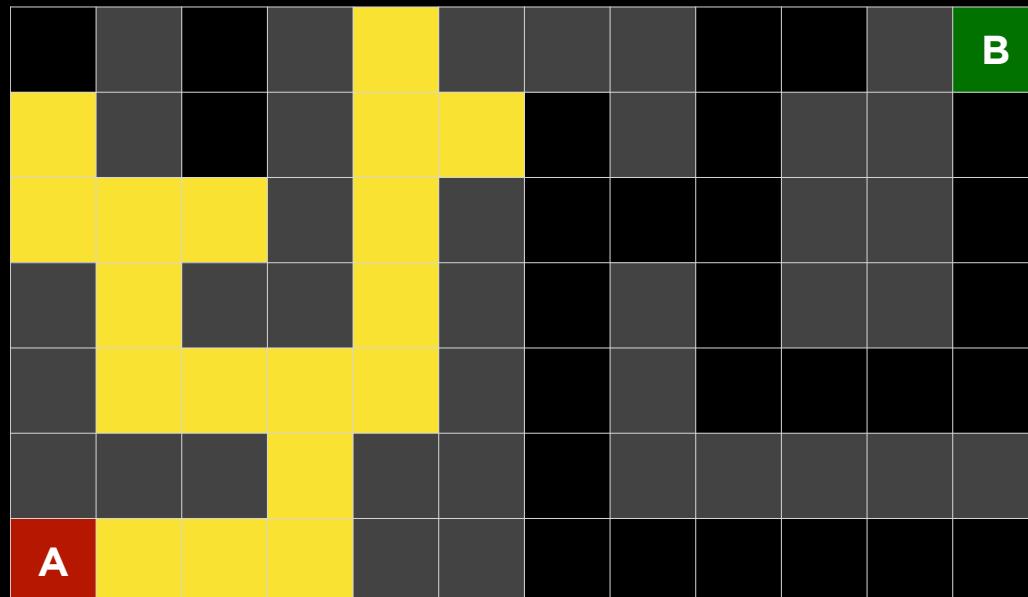
## Breadth-First Search



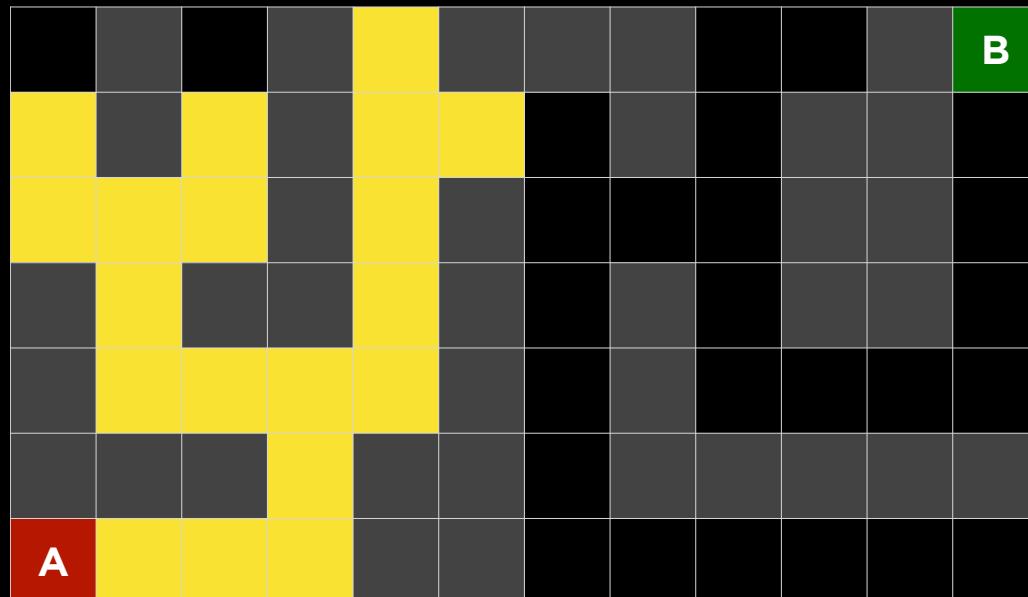
## Breadth-First Search



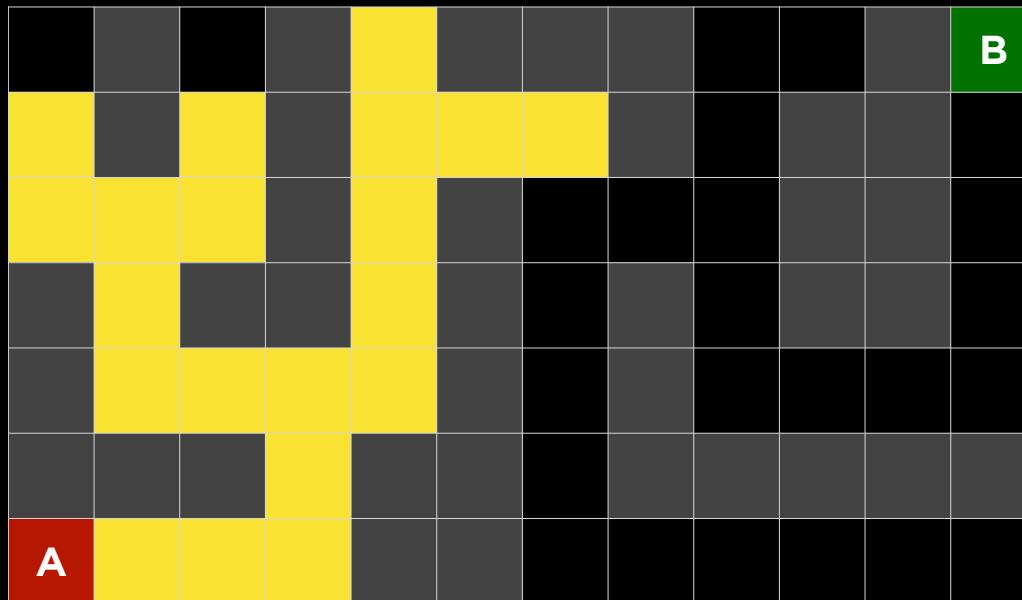
## Breadth-First Search



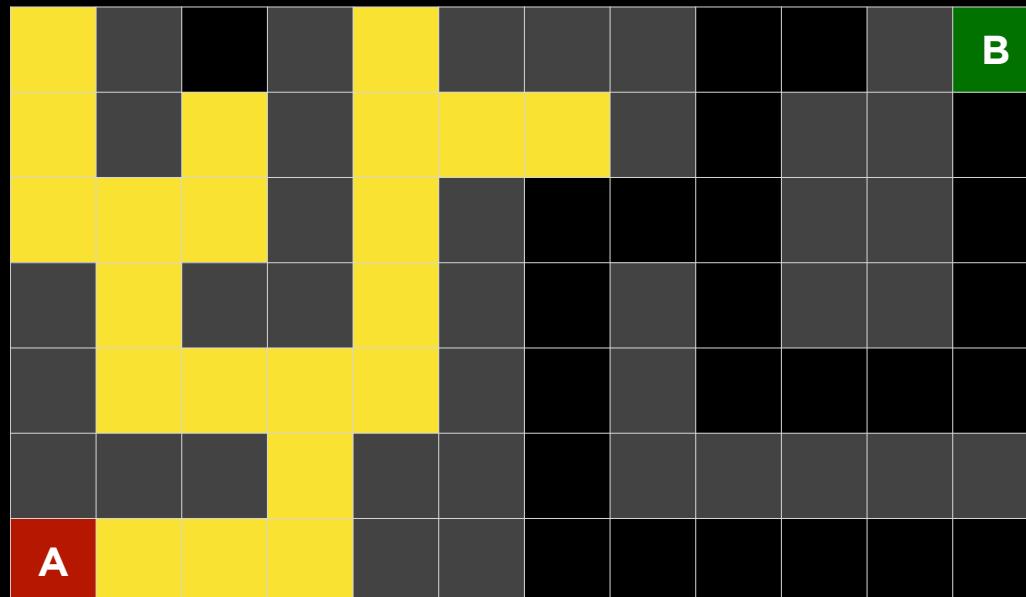
## Breadth-First Search



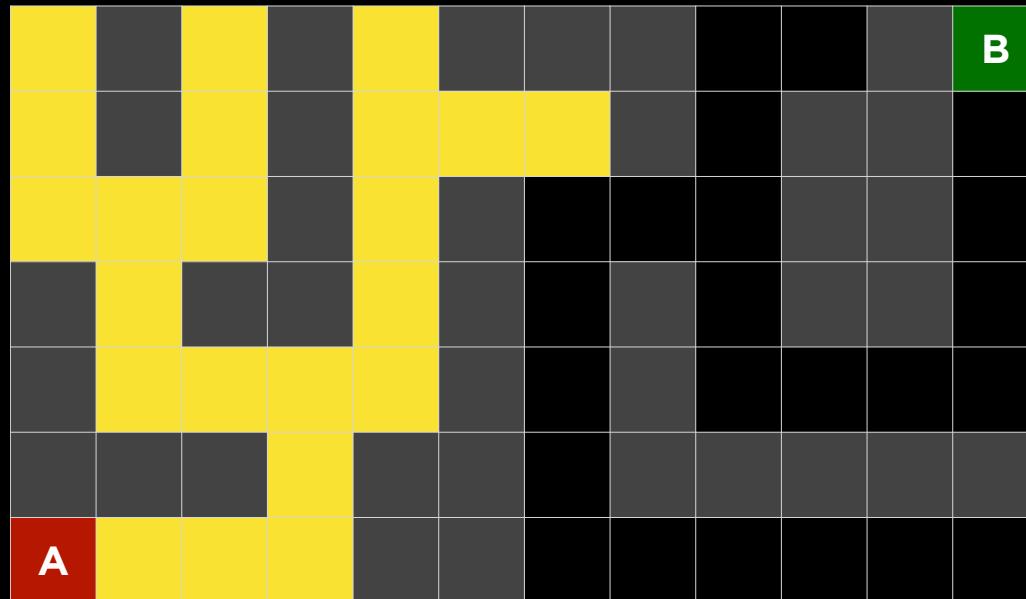
## Breadth-First Search



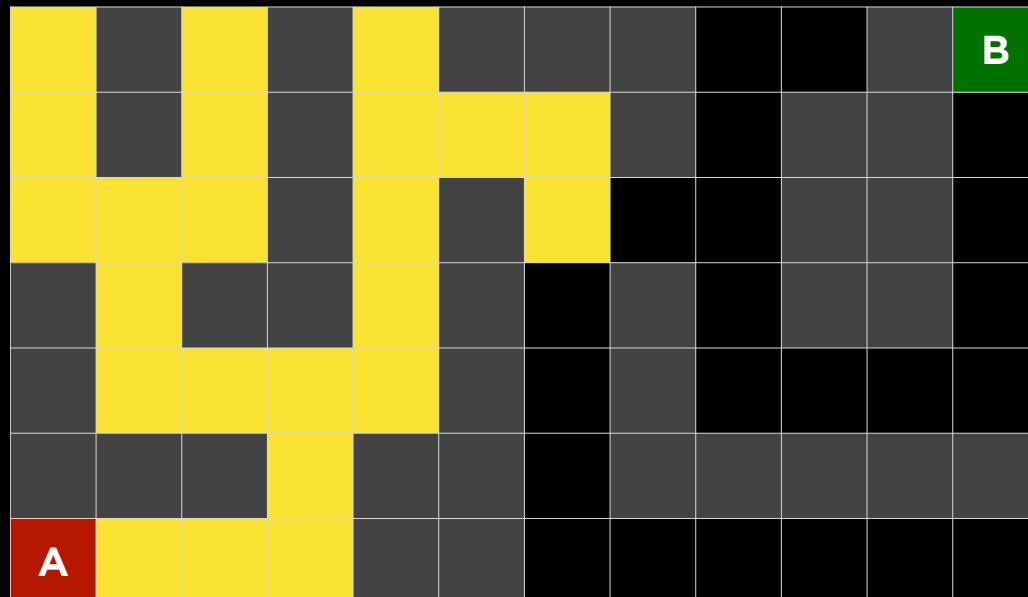
## Breadth-First Search



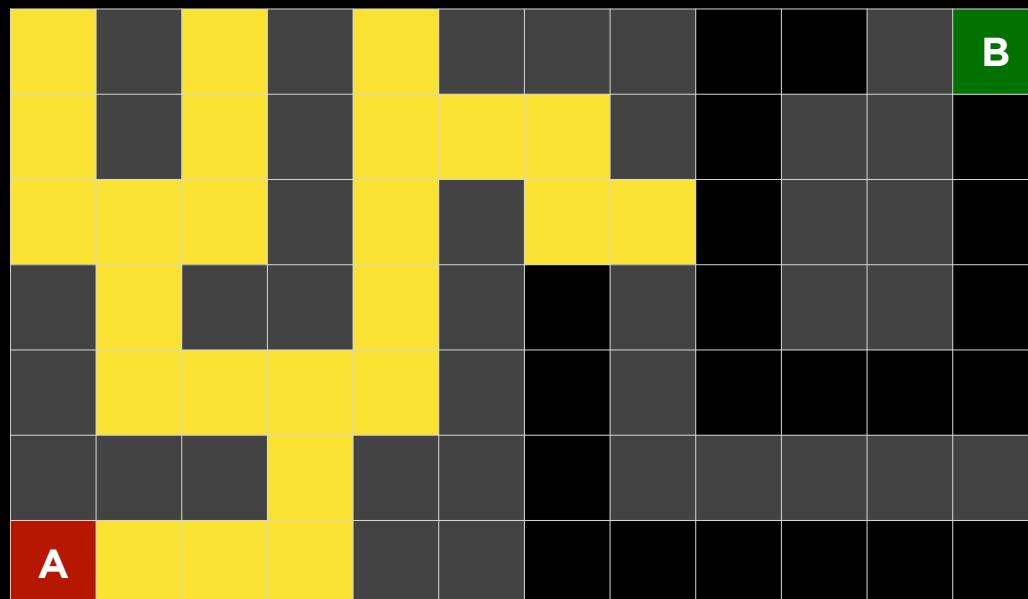
## Breadth-First Search



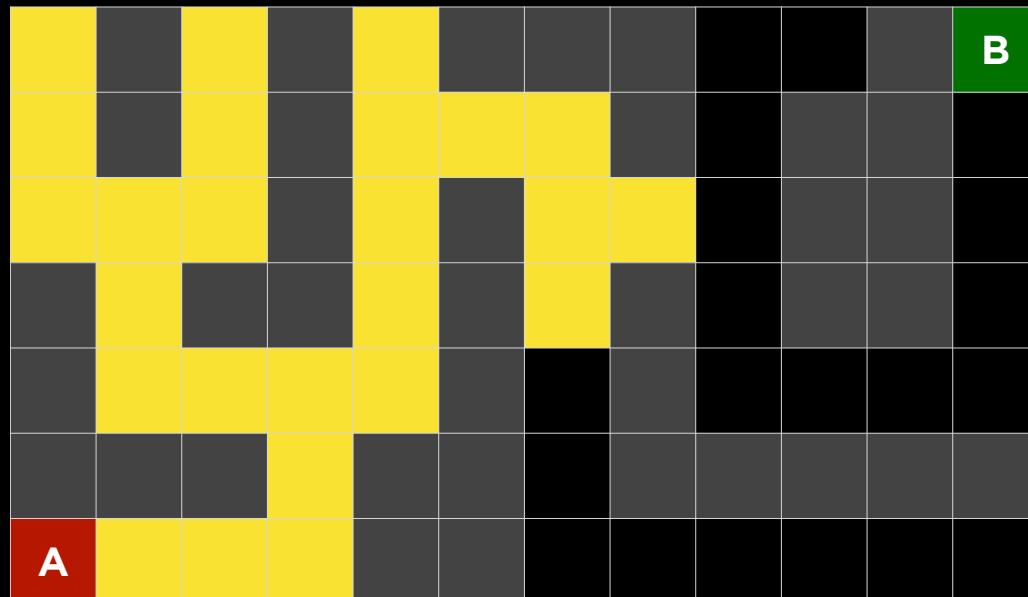
## Breadth-First Search



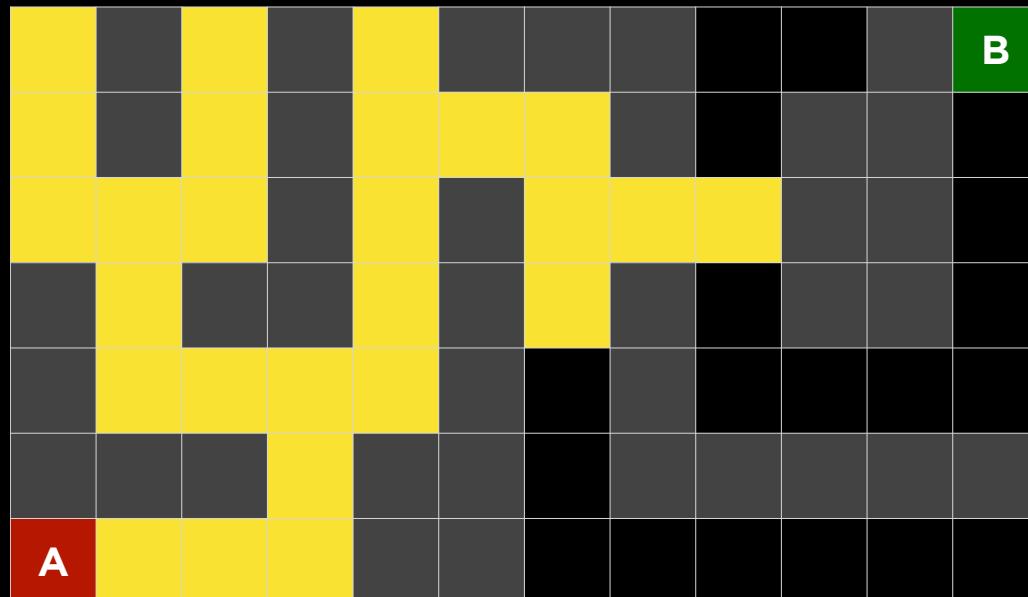
## Breadth-First Search



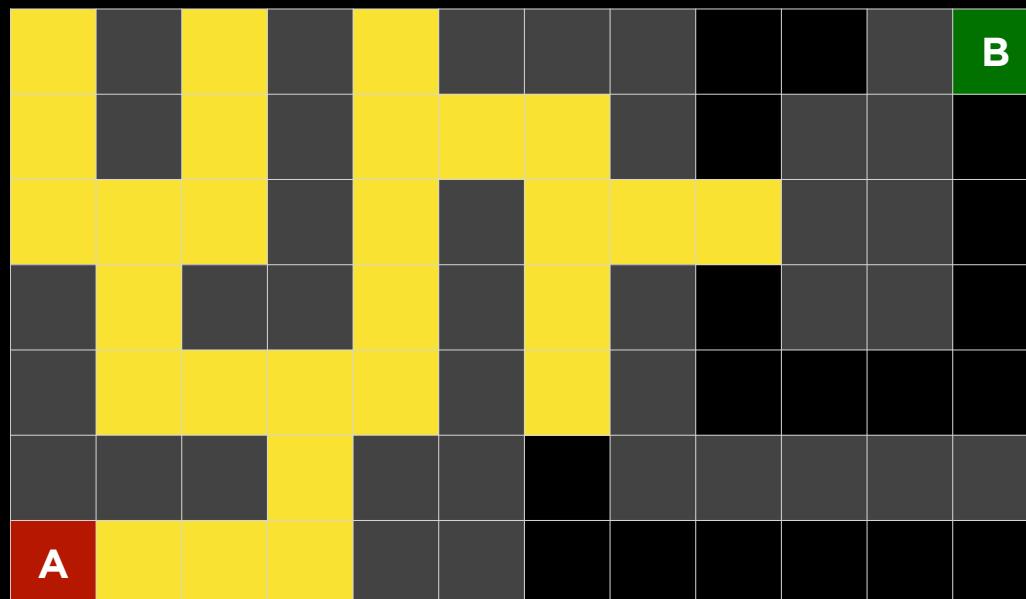
## Breadth-First Search



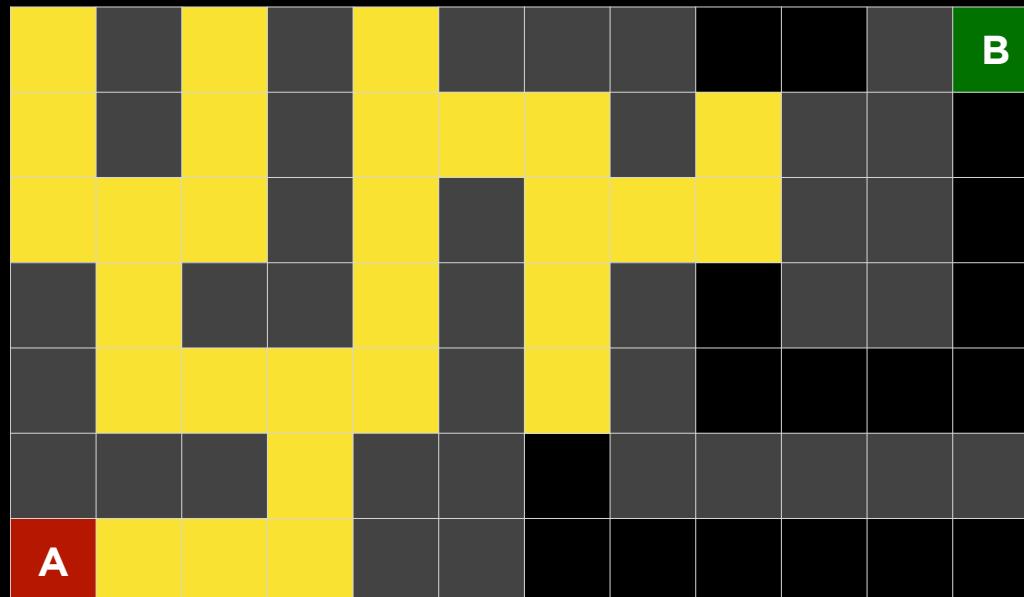
## Breadth-First Search



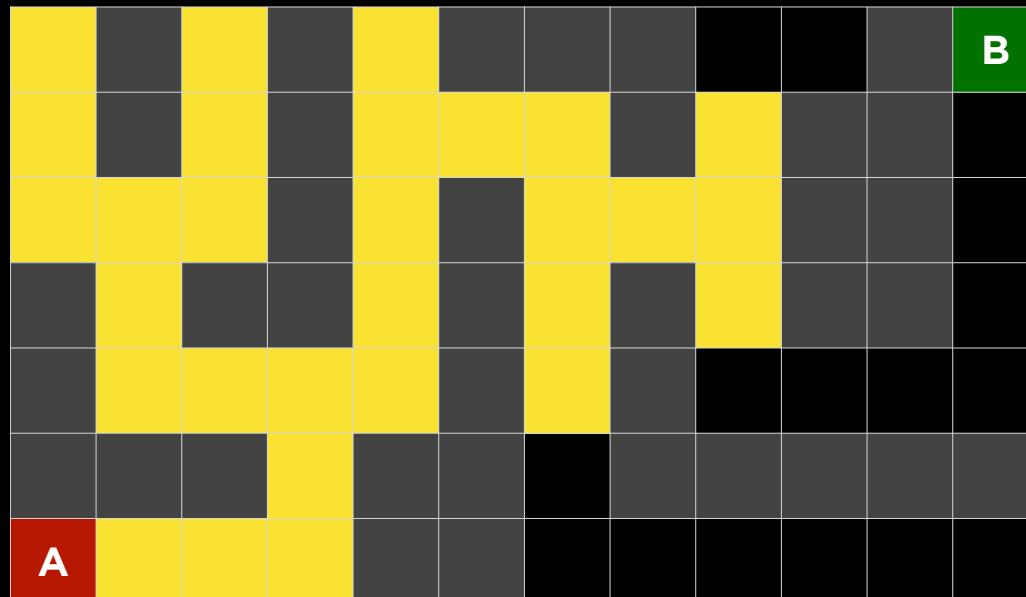
## Breadth-First Search



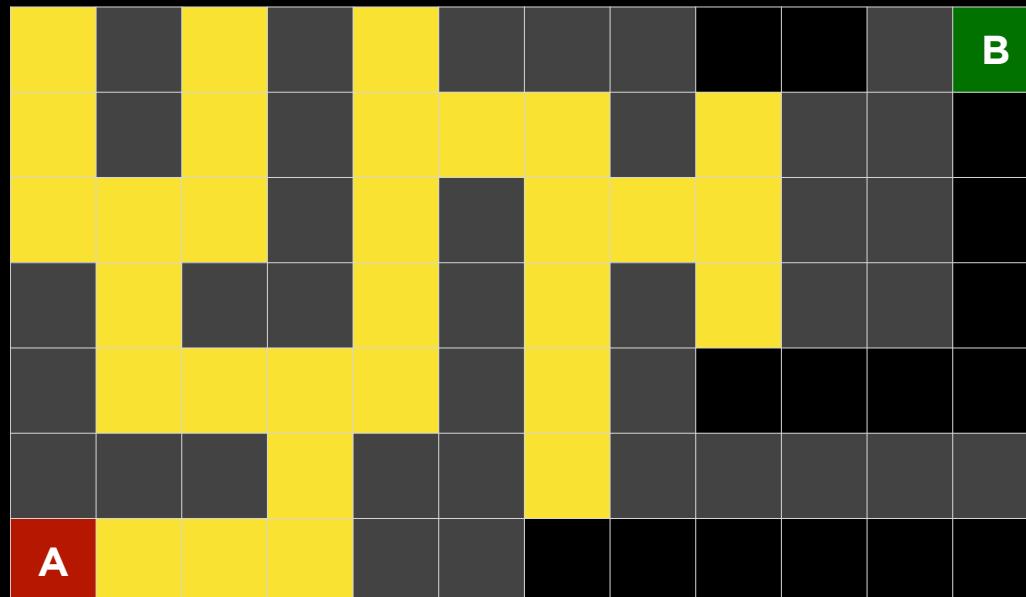
## Breadth-First Search



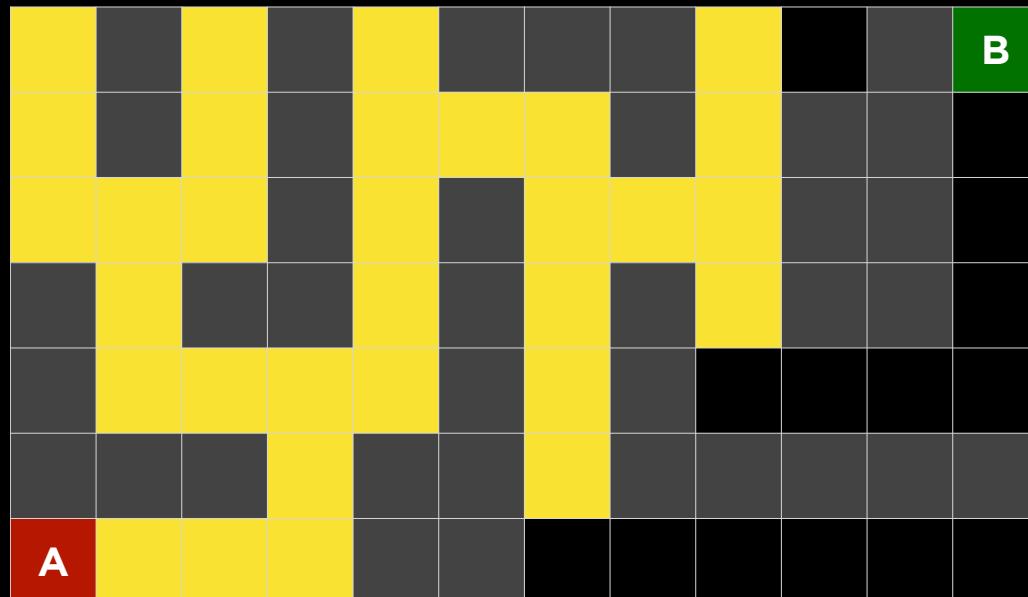
## Breadth-First Search



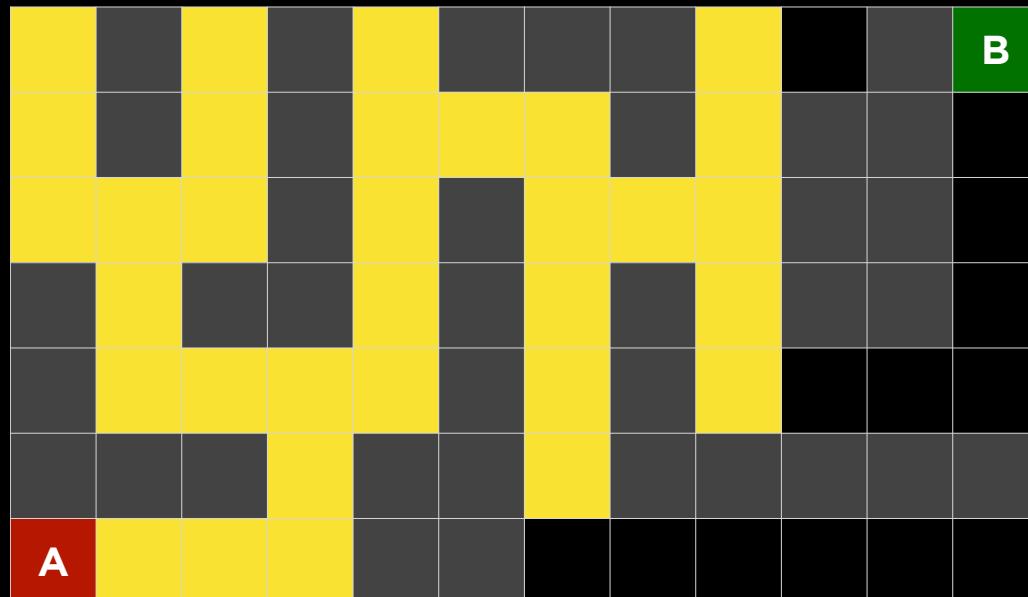
## Breadth-First Search



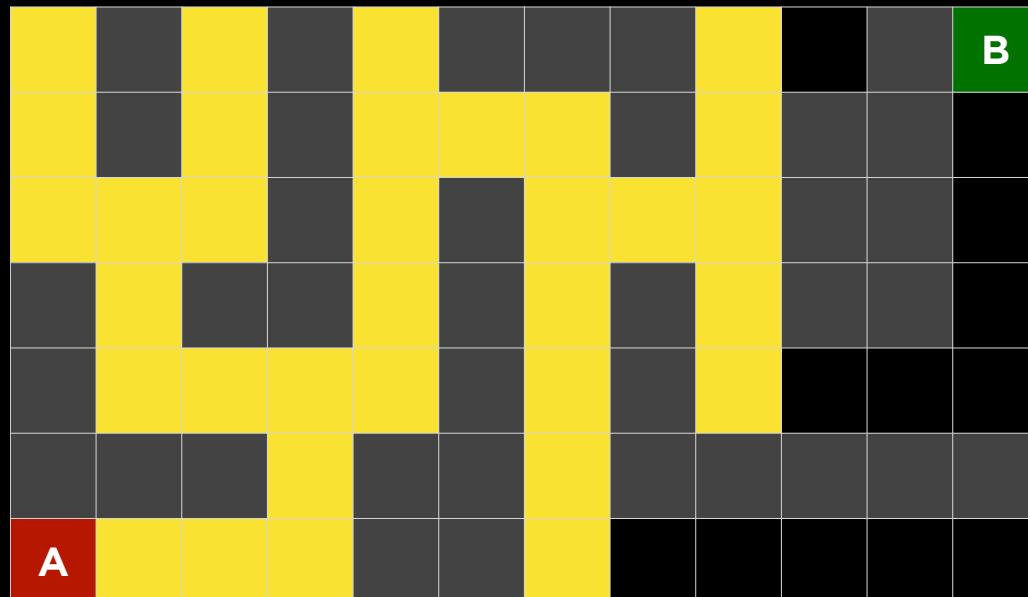
## Breadth-First Search



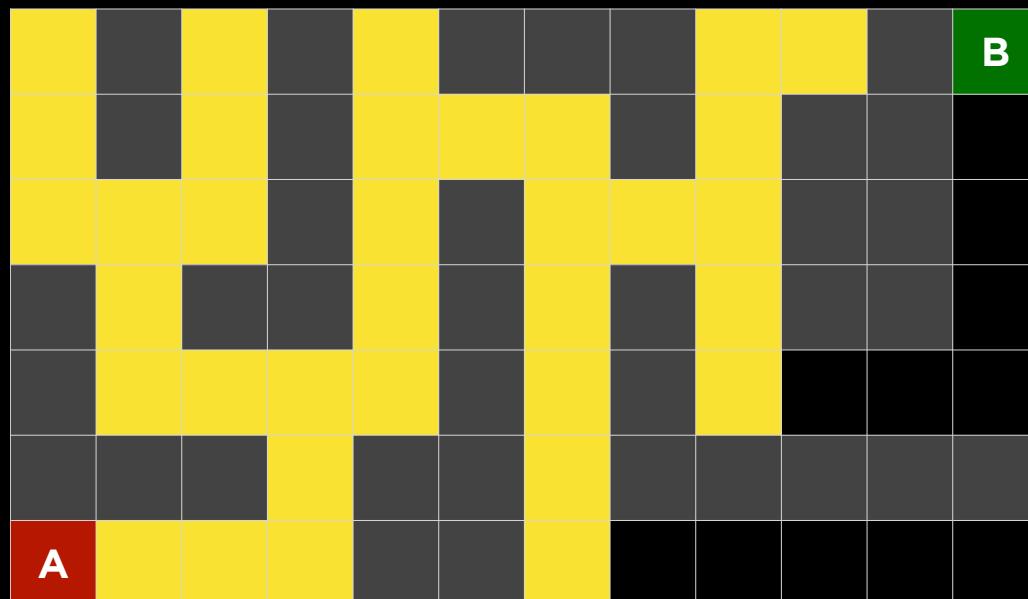
## Breadth-First Search



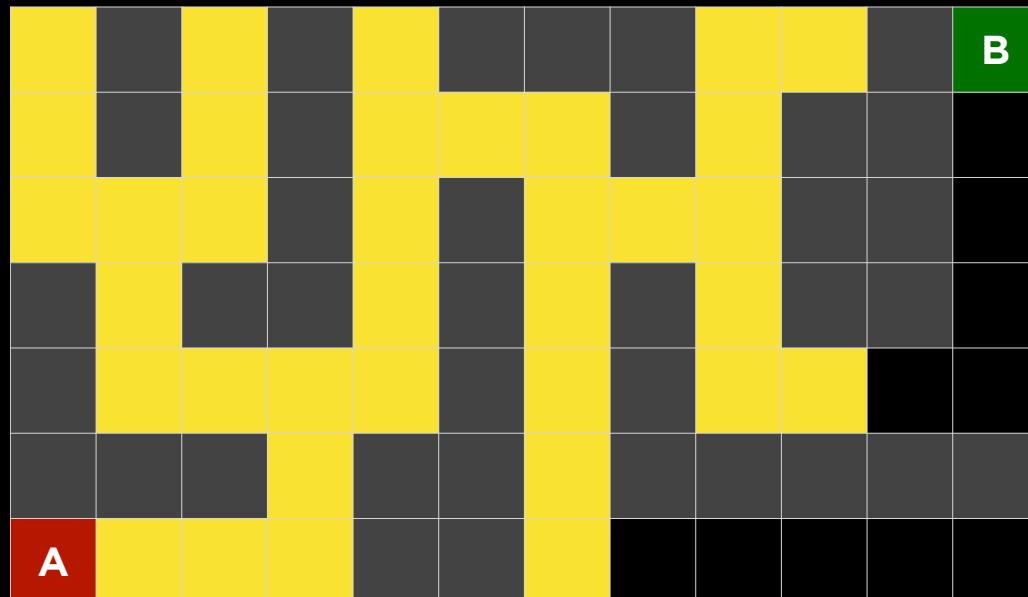
## Breadth-First Search



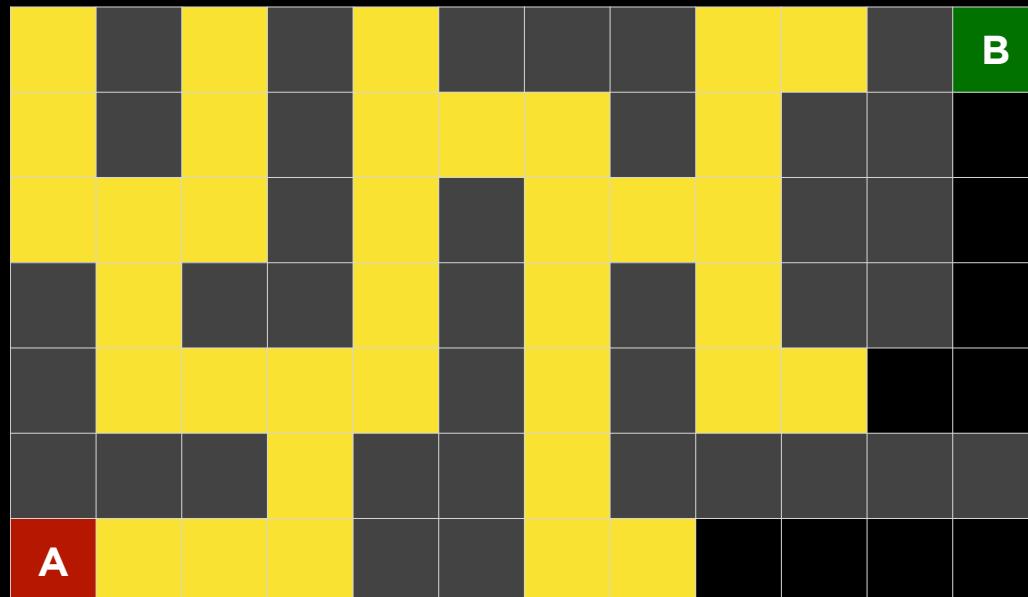
## Breadth-First Search



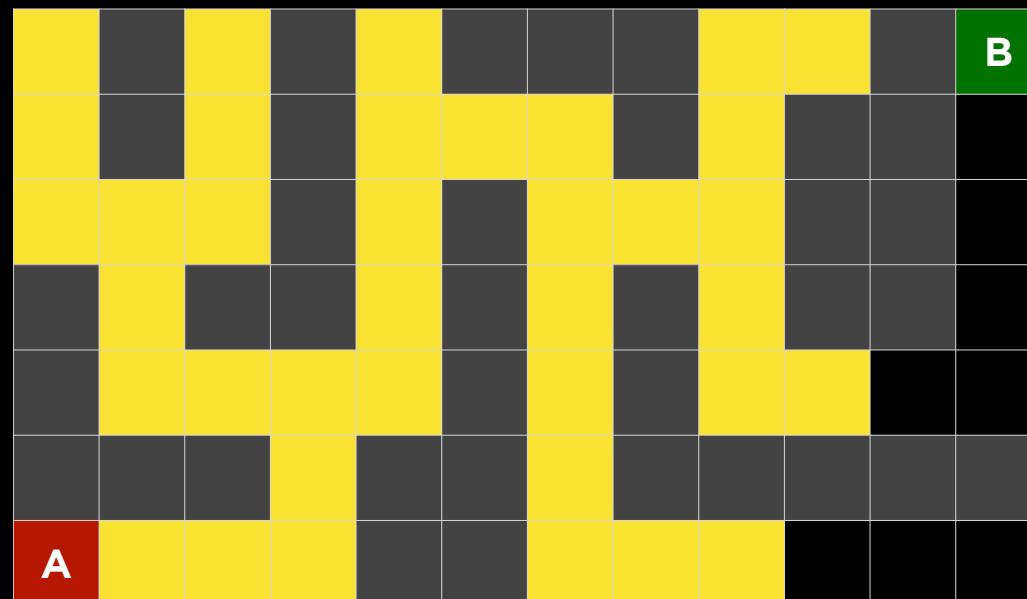
## Breadth-First Search



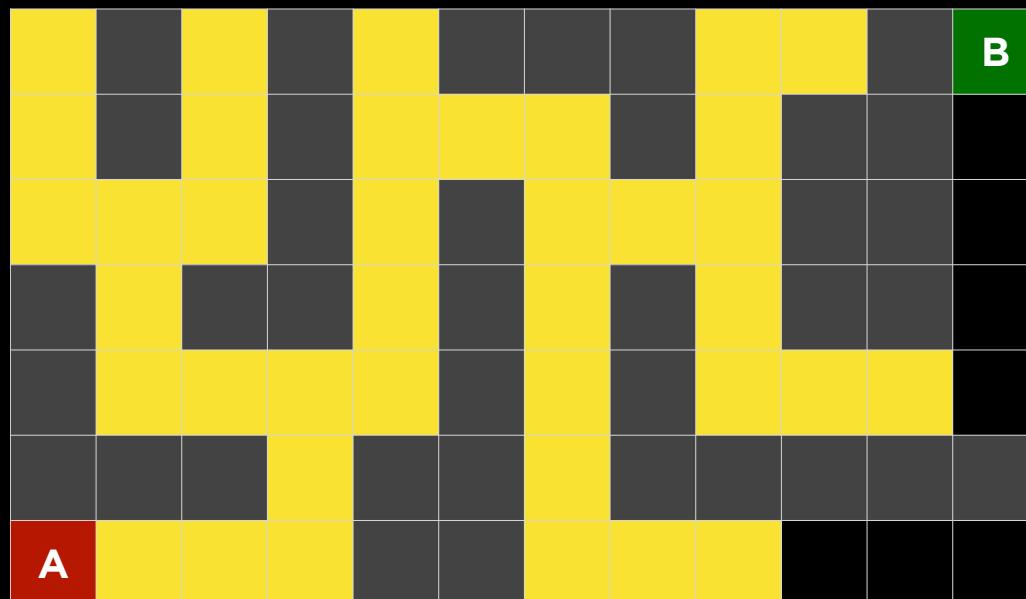
## Breadth-First Search



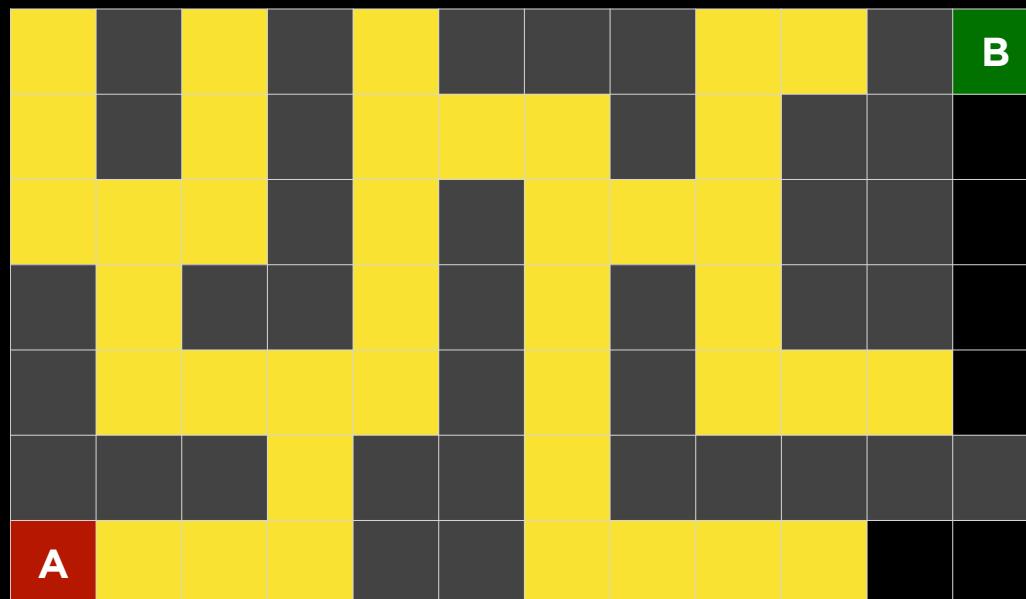
## Breadth-First Search



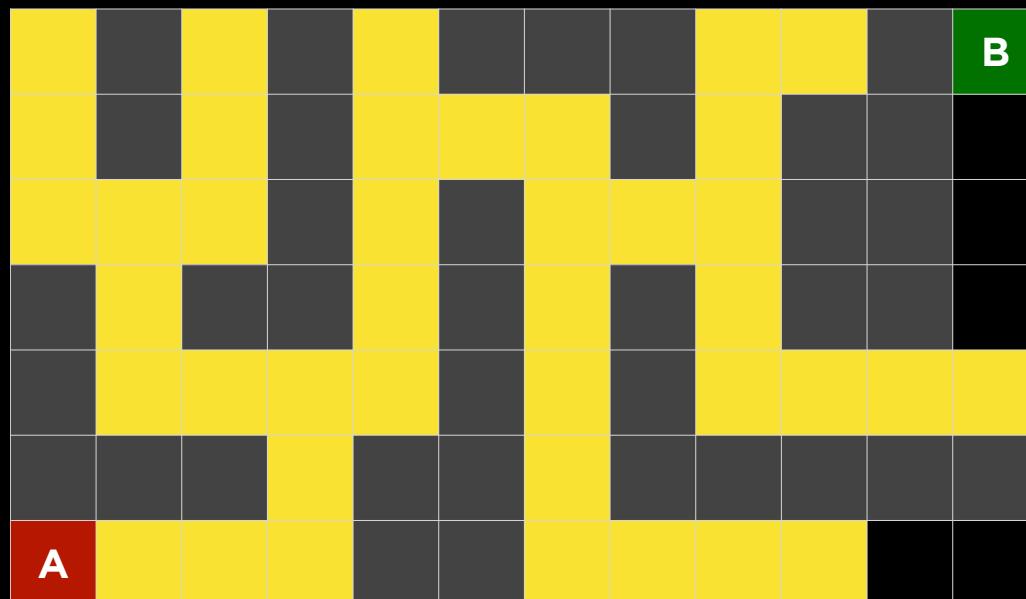
## Breadth-First Search



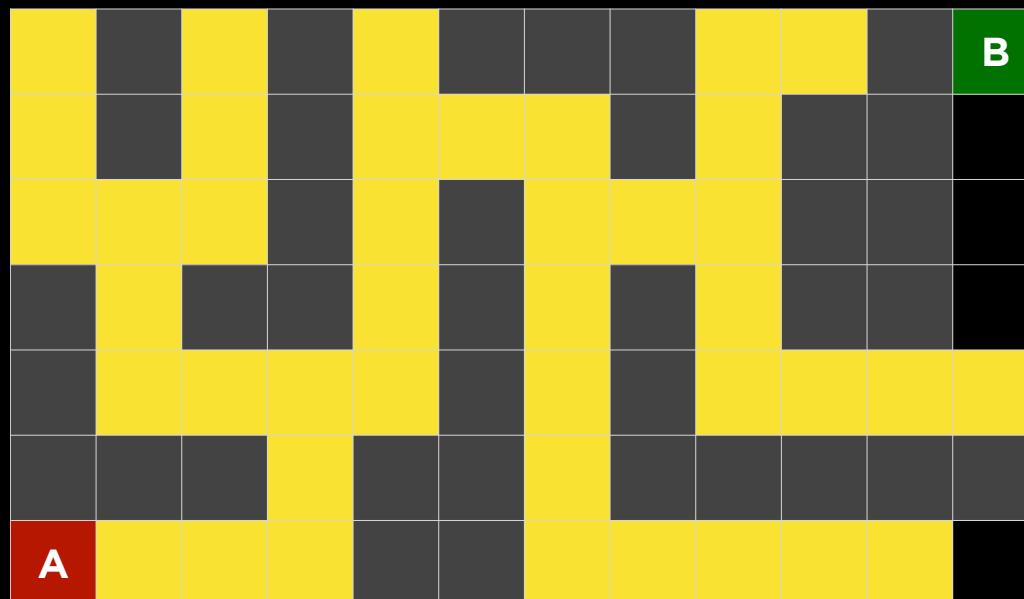
## Breadth-First Search



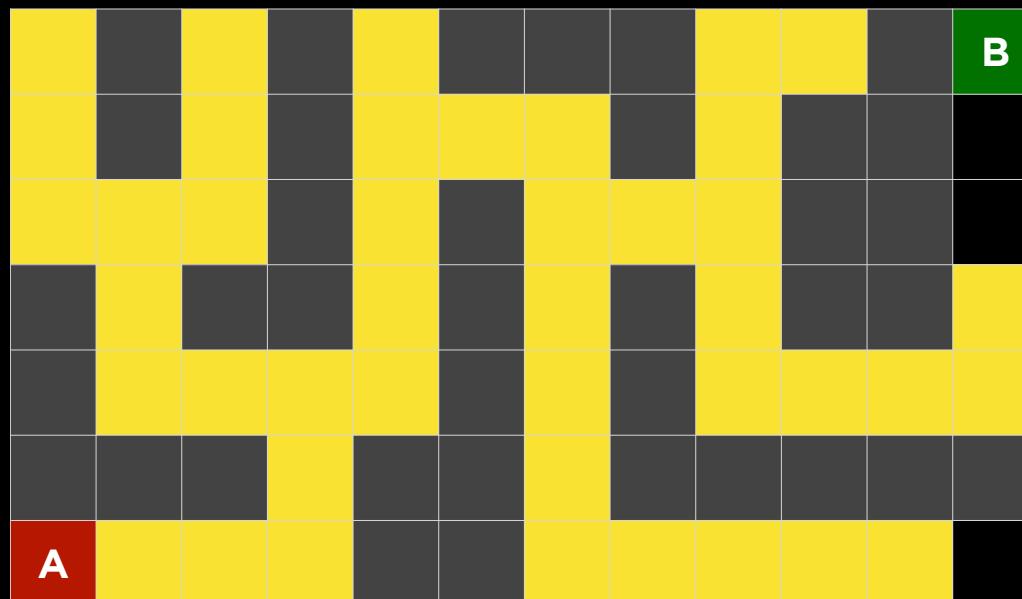
## Breadth-First Search



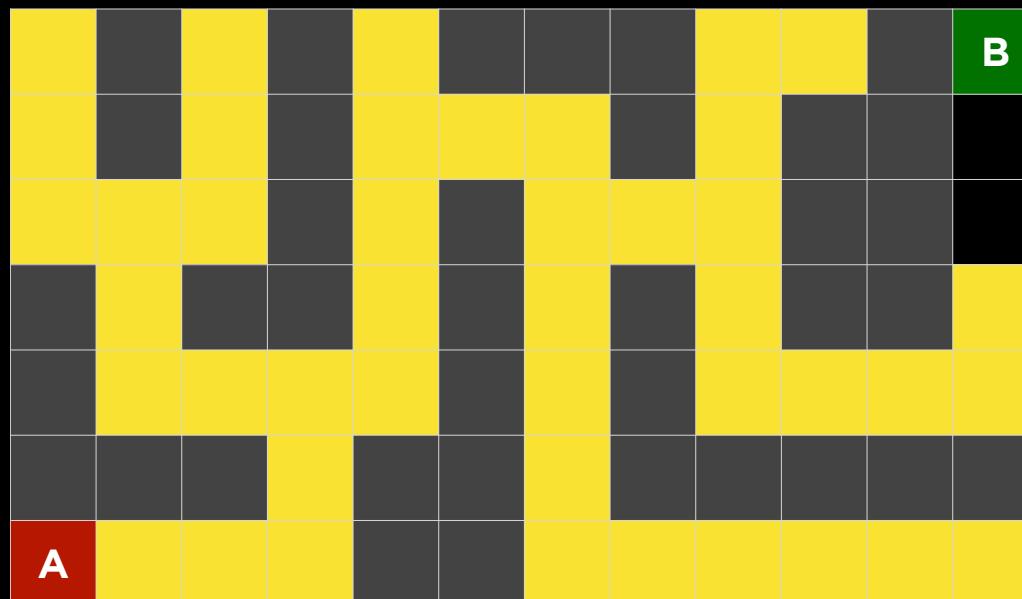
## Breadth-First Search



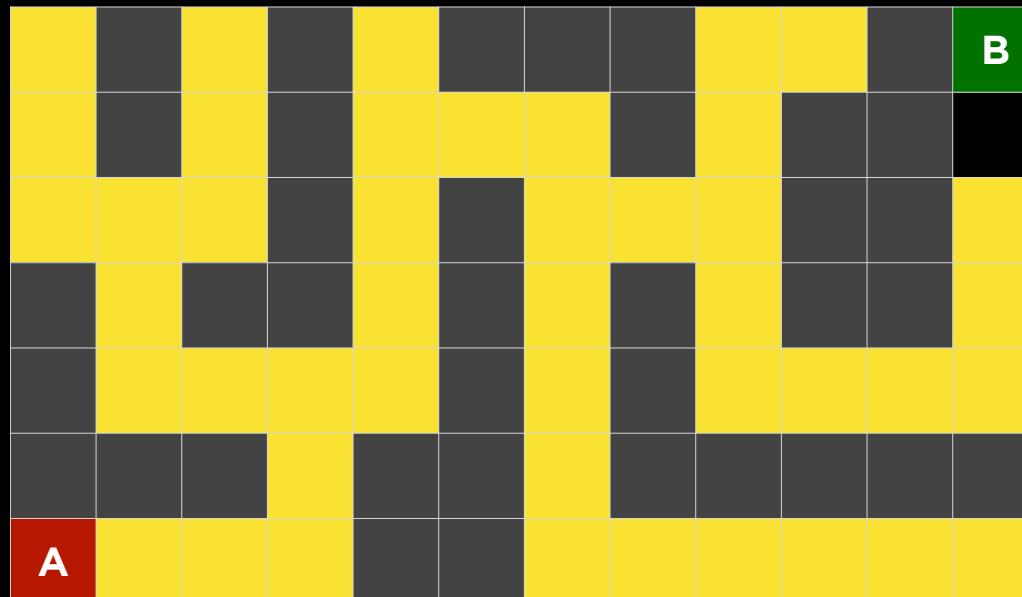
## Breadth-First Search



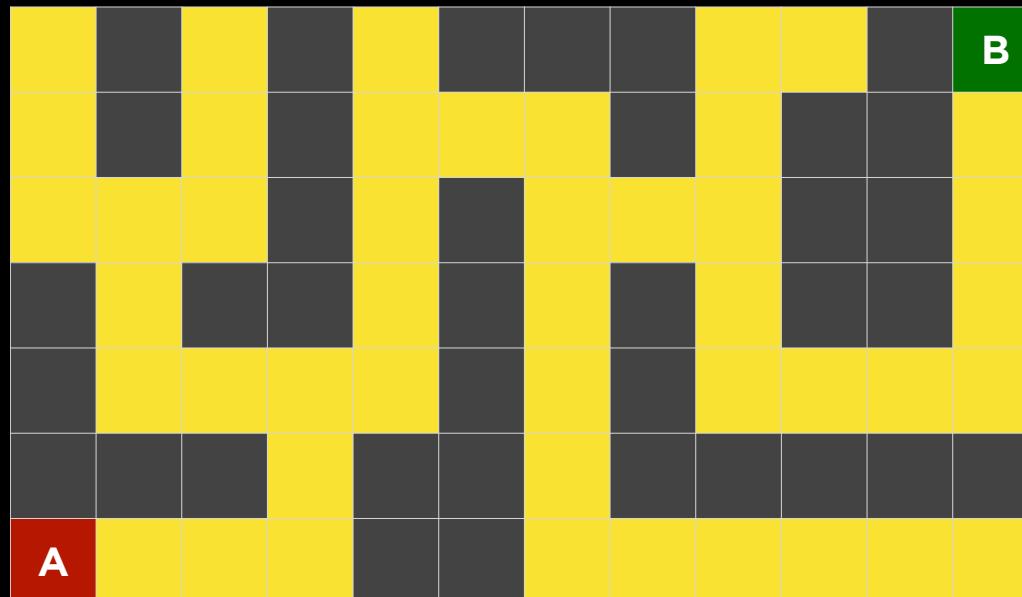
## Breadth-First Search



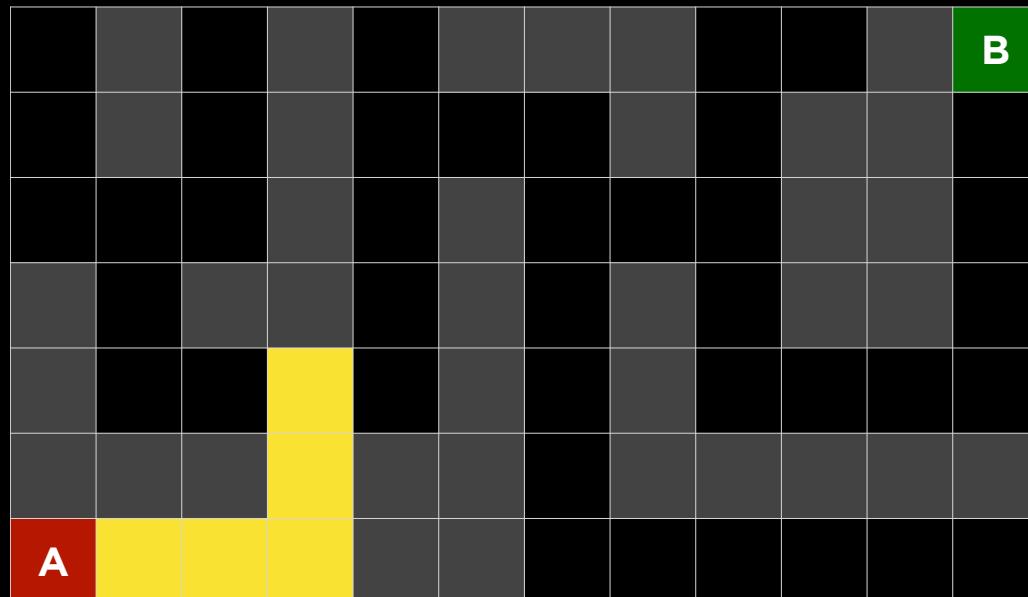
## Breadth-First Search



## Breadth-First Search



## Breadth-First Search



# **uninformed search**

search strategy that uses no problem-specific knowledge

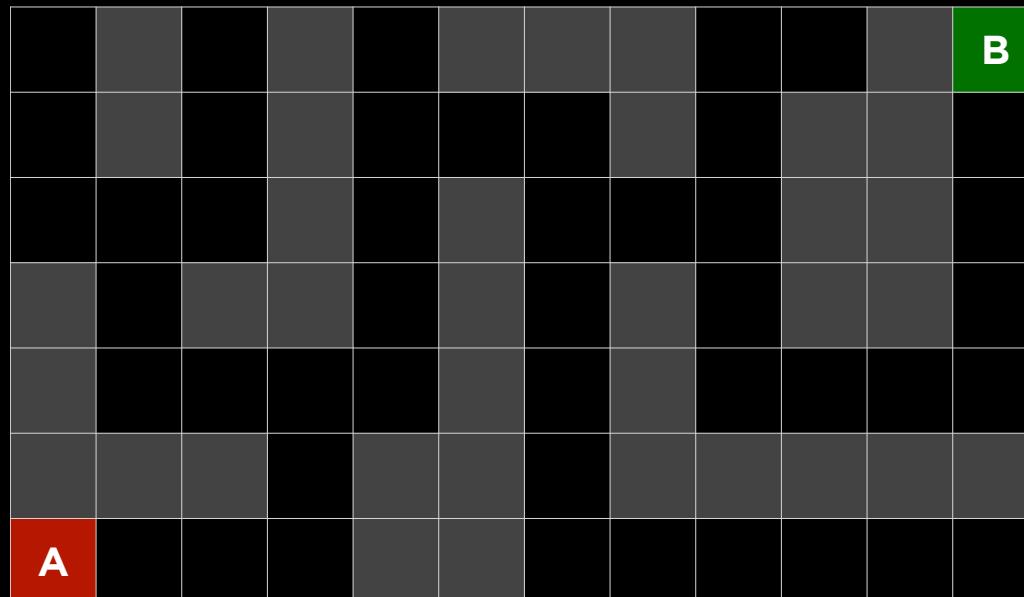
# **informed search**

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

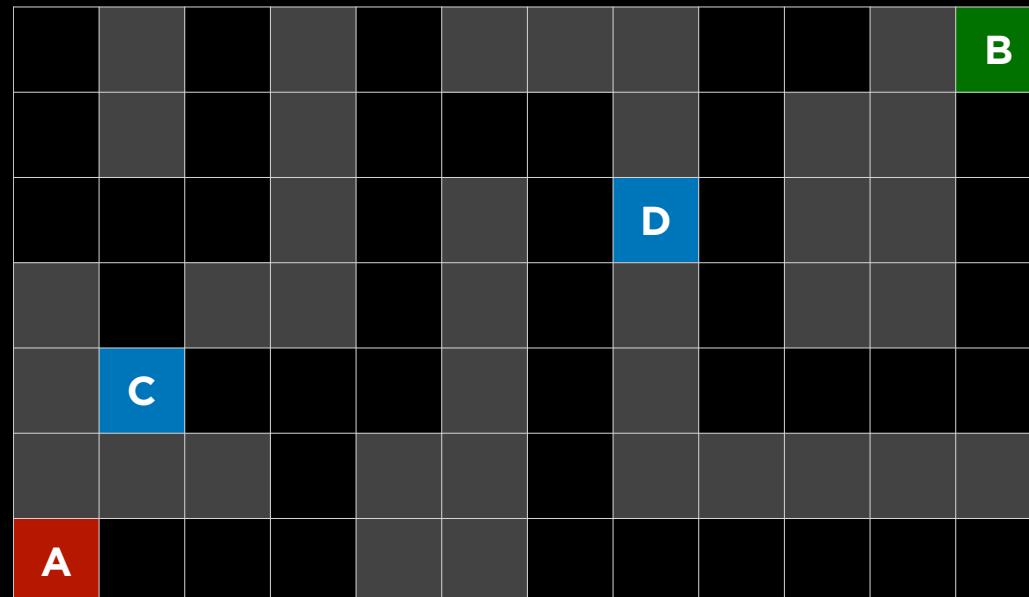
## greedy best-first search

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

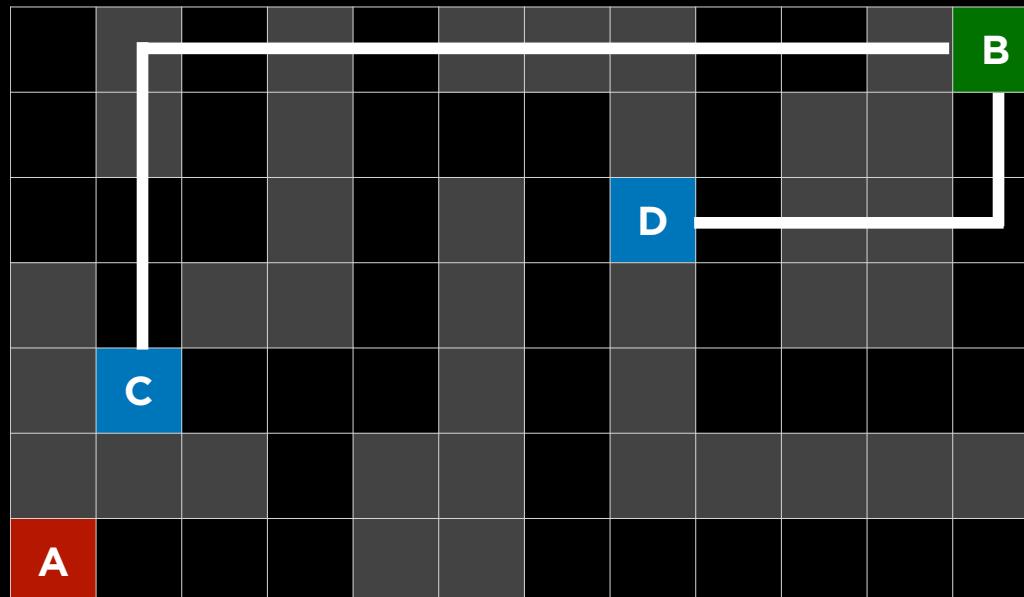
Heuristic function?



Heuristic function?



Heuristic function? Manhattan distance.



## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	<b>16</b>	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	16	15	14			11	10	9	8	7	6

## Greedy Best-First Search

11		9		7				3	2		<b>B</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					
<b>A</b>	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

	10	9	8	7	6	5	4	3	2	1	<b>B</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
<b>A</b>	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

## 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>B</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
<b>A</b>	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						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	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

	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

	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

	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

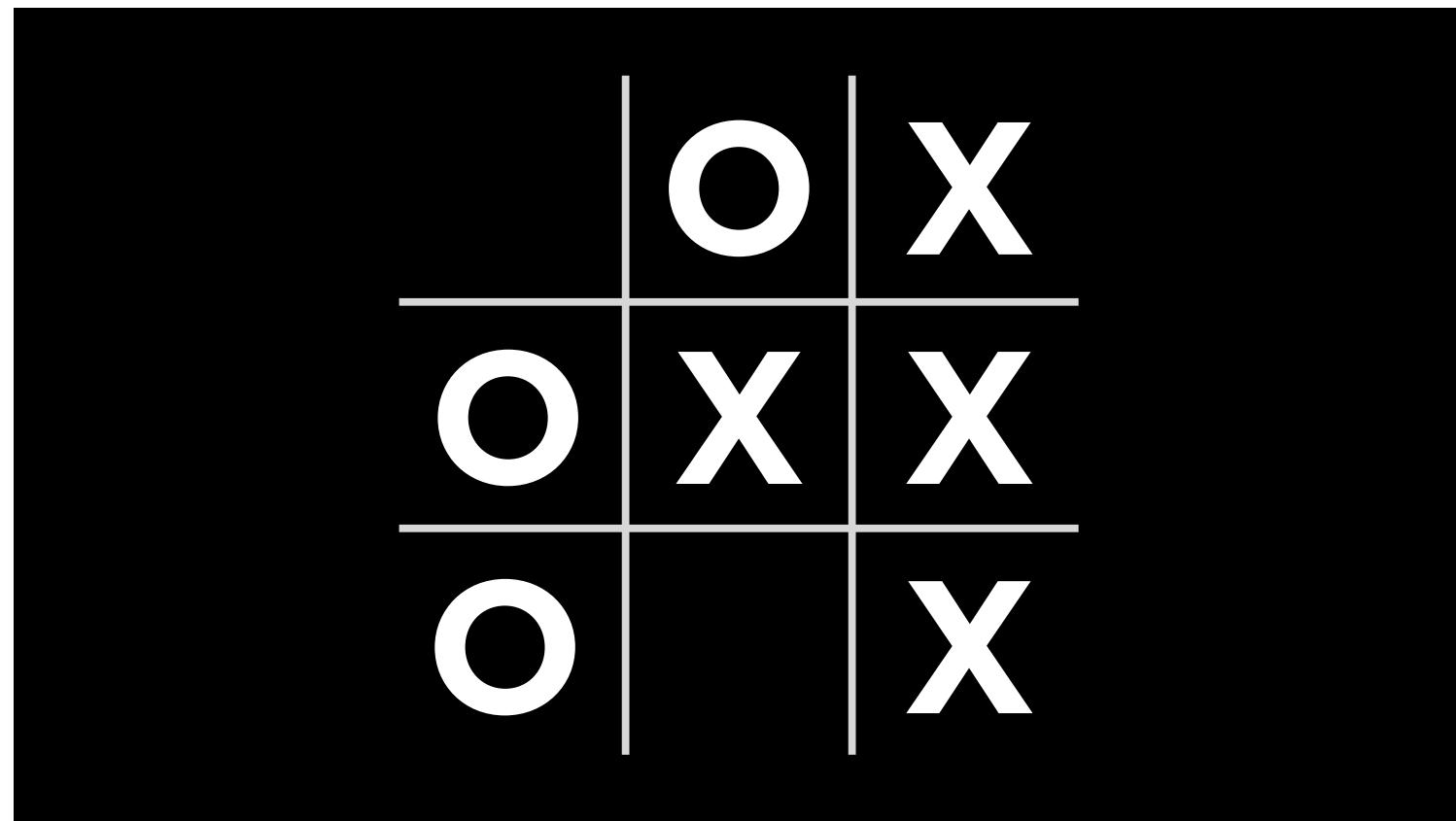
	11+10	12+9	13+8	14+7	15+6	16+5	17+4	18+3	19+2	20+1	B
	10+11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

# A\* search

optimal if

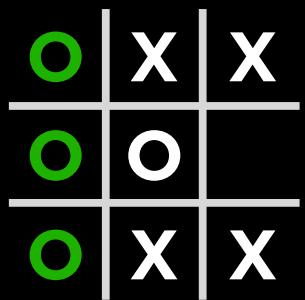
- $h(n)$  is admissible (never overestimates the true cost), and
- $h(n)$  is consistent (for every node  $n$  and successor  $n'$  with step cost  $c$ ,  $h(n) \leq h(n') + c$ )

# **Adversarial Search**

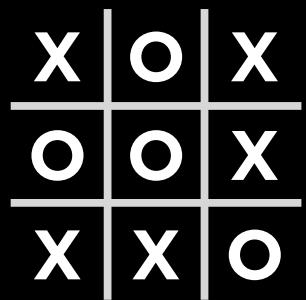




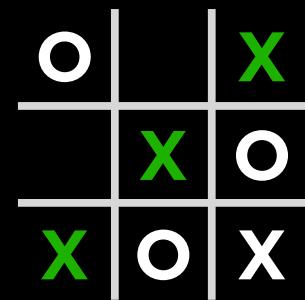
# Minimax



-1



0



1

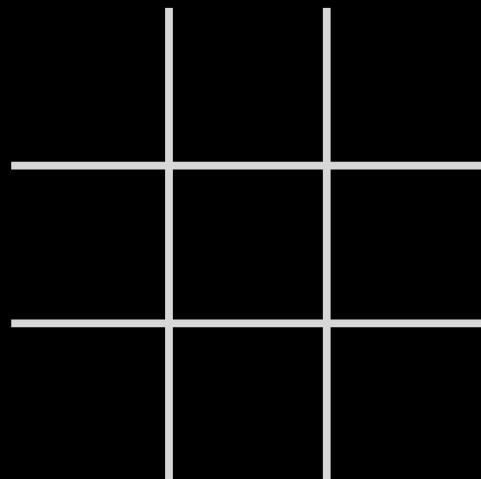
# Minimax

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

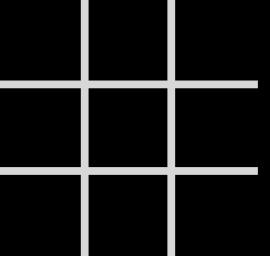
# Game

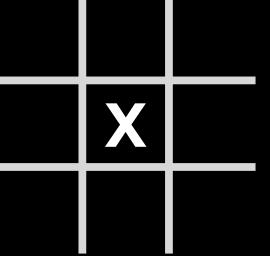
- $S_0$  : initial state
- $\text{PLAYER}(s)$  : returns which player to move in state  $s$
- $\text{ACTIONS}(s)$  : returns legal moves in state  $s$
- $\text{RESULT}(s, a)$  : returns state after action  $a$  taken in state  $s$
- $\text{TERMINAL}(s)$  : checks if state  $s$  is a terminal state
- $\text{UTILITY}(s)$  : final numerical value for terminal state  $s$

# Initial State



PLAYER( $s$ )

PLAYER(  ) = **X**

PLAYER(  ) = **O**

ACTIONS( $s$ )

$$\text{ACTIONS}\left(\begin{array}{|c|c|c|} \hline & x & o \\ \hline o & x & x \\ \hline x & & o \\ \hline \end{array}\right) = \left\{ \begin{array}{|c|c|c|} \hline & o & \\ \hline & & \\ \hline & & \\ \hline \end{array}, \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & o & \\ \hline \end{array} \right\}$$

RESULT( $s$ ,  $a$ )

$$\text{RESULT}\left(\begin{array}{|c|c|c|} \hline & x & o \\ \hline o & x & x \\ \hline x & & o \\ \hline \end{array}, \begin{array}{|c|c|} \hline \textcolor{yellow}{o} & \# \\ \hline \# & \# \\ \hline \end{array}\right) = \begin{array}{|c|c|c|} \hline o & x & o \\ \hline o & x & x \\ \hline x & & o \\ \hline \end{array}$$

$\text{TERMINAL}(s)$

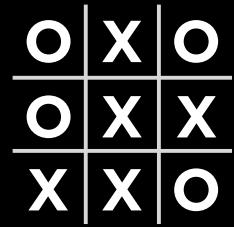
$\text{TERMINAL}(\begin{array}{|c|c|c|}\hline \text{o} & & \\ \hline \text{o} & \text{x} & \\ \hline \text{x} & \text{o} & \text{x} \\ \hline\end{array}) = \text{false}$

$\text{TERMINAL}(\begin{array}{|c|c|c|}\hline \text{o} & & \text{x} \\ \hline \text{o} & \text{x} & \\ \hline \text{x} & \text{o} & \text{x} \\ \hline\end{array}) = \text{true}$

UTILITY( $s$ )

$$\text{UTILITY}\left(\begin{array}{|c|c|c|} \hline \text{o} & & \text{x} \\ \hline \text{o} & \text{x} & \\ \hline \text{x} & \text{o} & \text{x} \\ \hline \end{array}\right) = 1$$

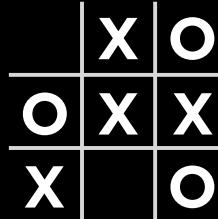
$$\text{UTILITY}\left(\begin{array}{|c|c|c|} \hline \text{o} & \text{x} & \text{x} \\ \hline \text{x} & \text{o} & \\ \hline \text{o} & \text{x} & \text{o} \\ \hline \end{array}\right) = -1$$



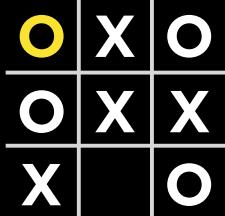
VALUE: 1

$\text{PLAYER}(s) = \circ$

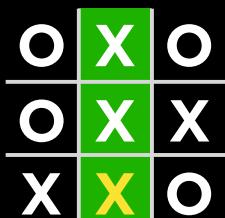
MN-VALUE:  
0



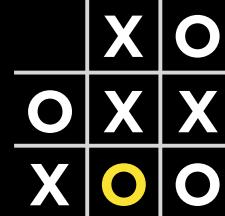
MAX-VALUE:  
1



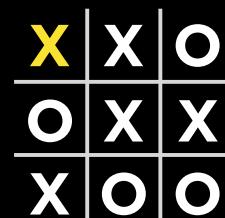
VALUE:  
1



MAX-VALUE:  
0



VALUE:  
0

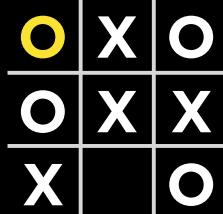


$\text{PLAYER}(s) = \circ$

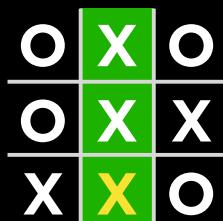
MN-VALUE:  
0



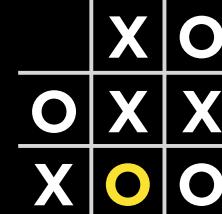
MAX-VALUE:  
1



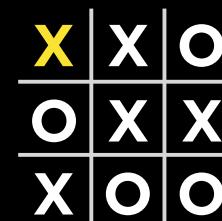
VALUE:  
1

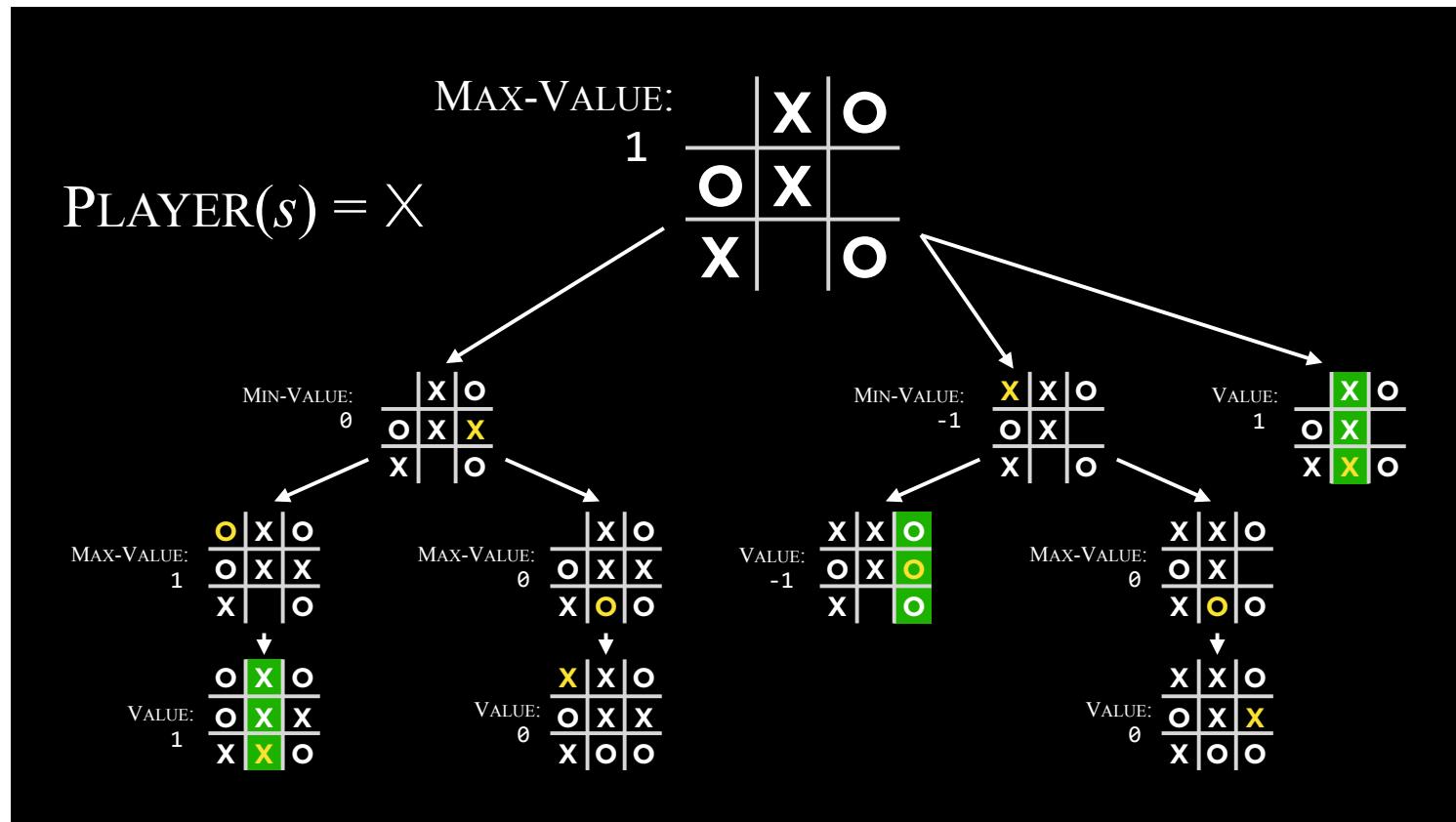


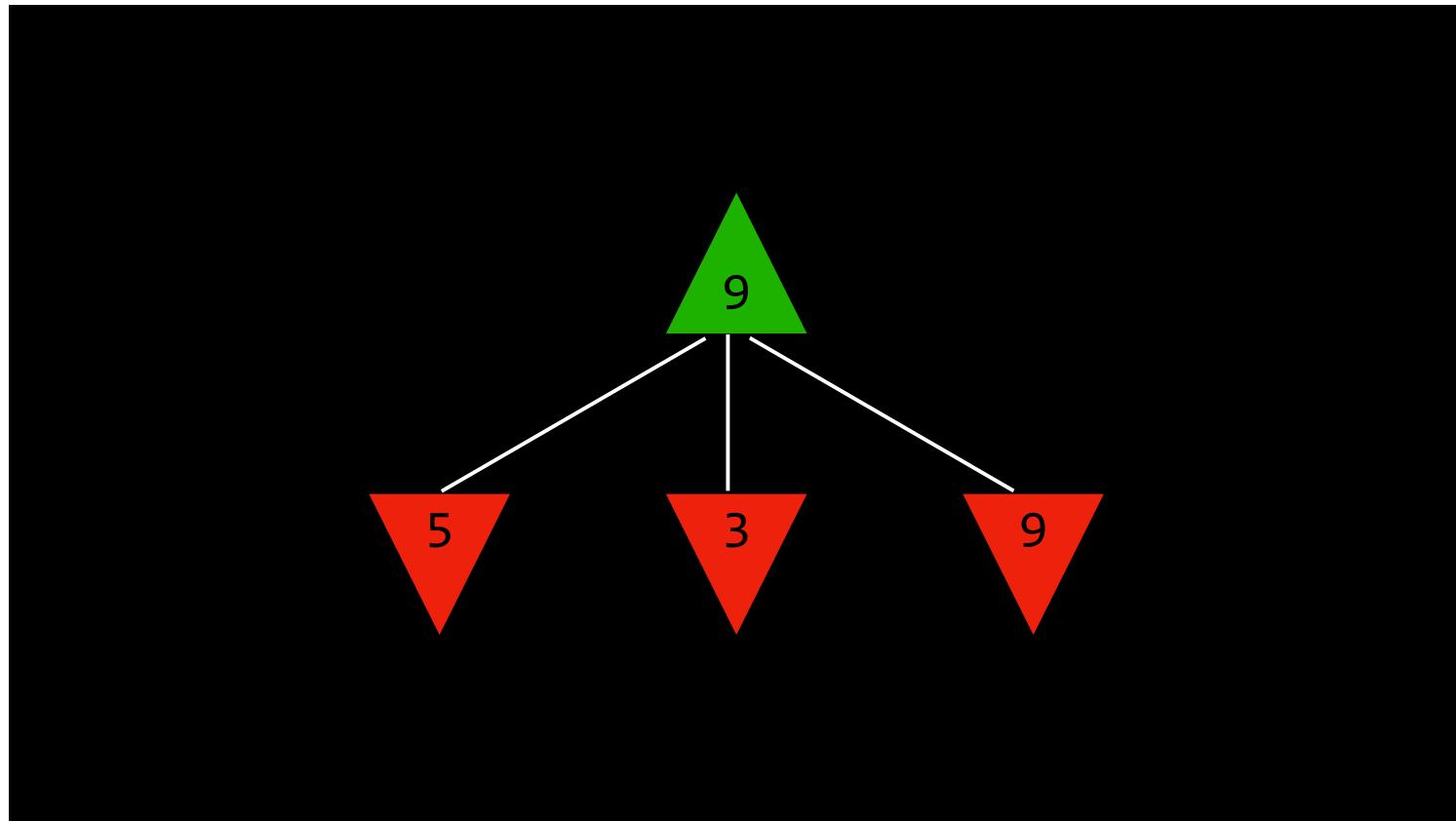
MAX-VALUE:  
0

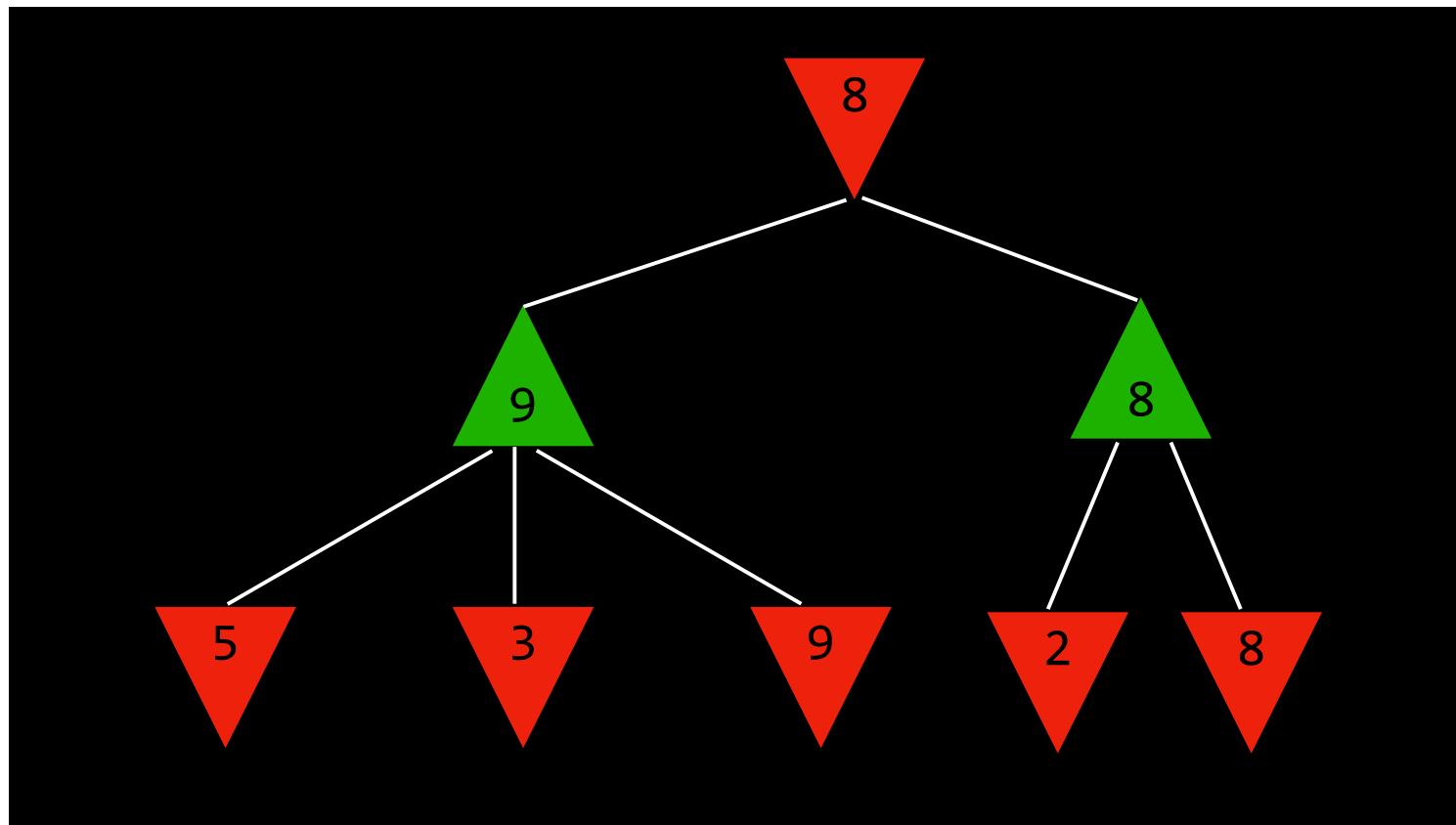


VALUE:  
0









# Minimax

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

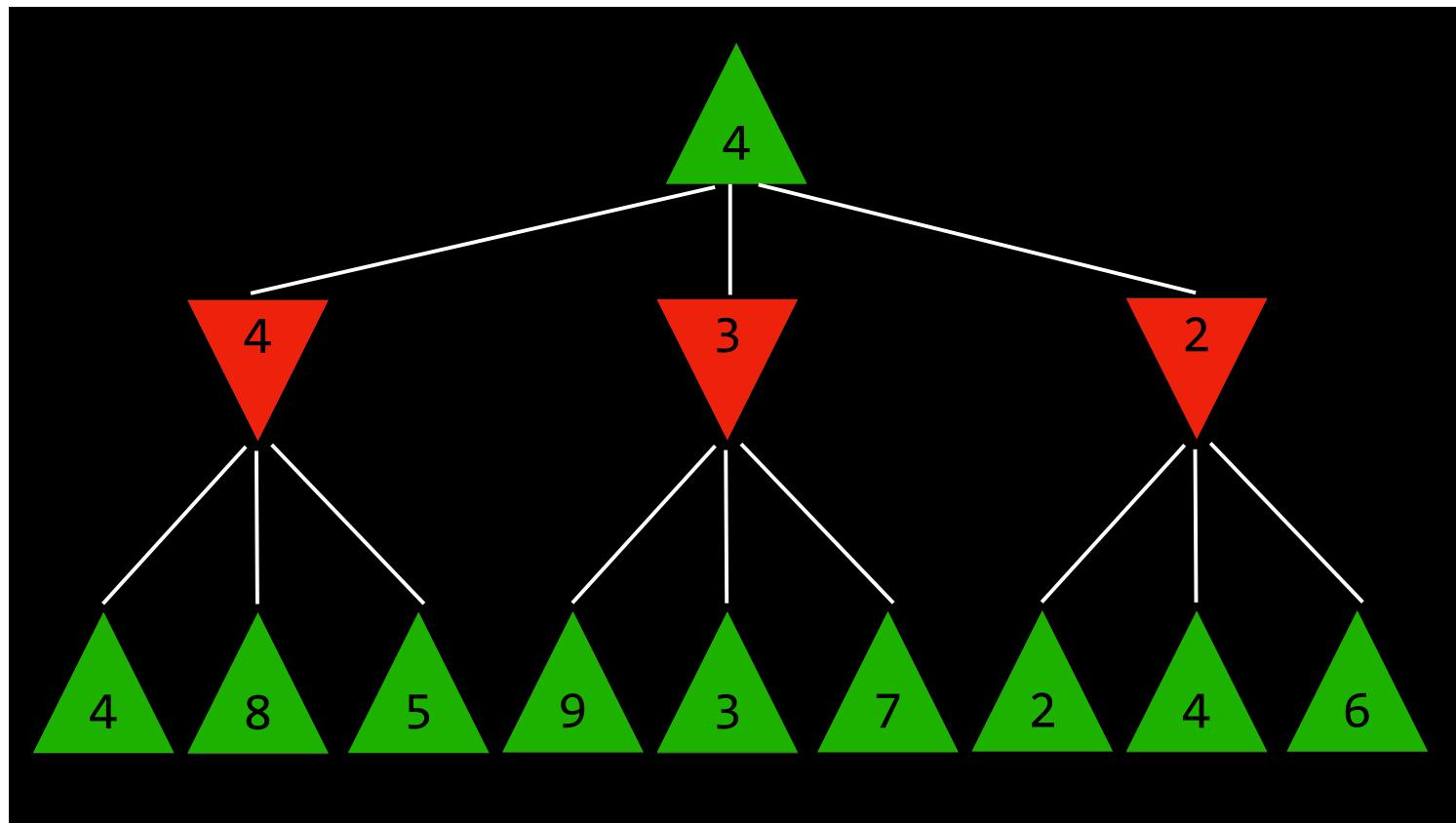
# Minimax

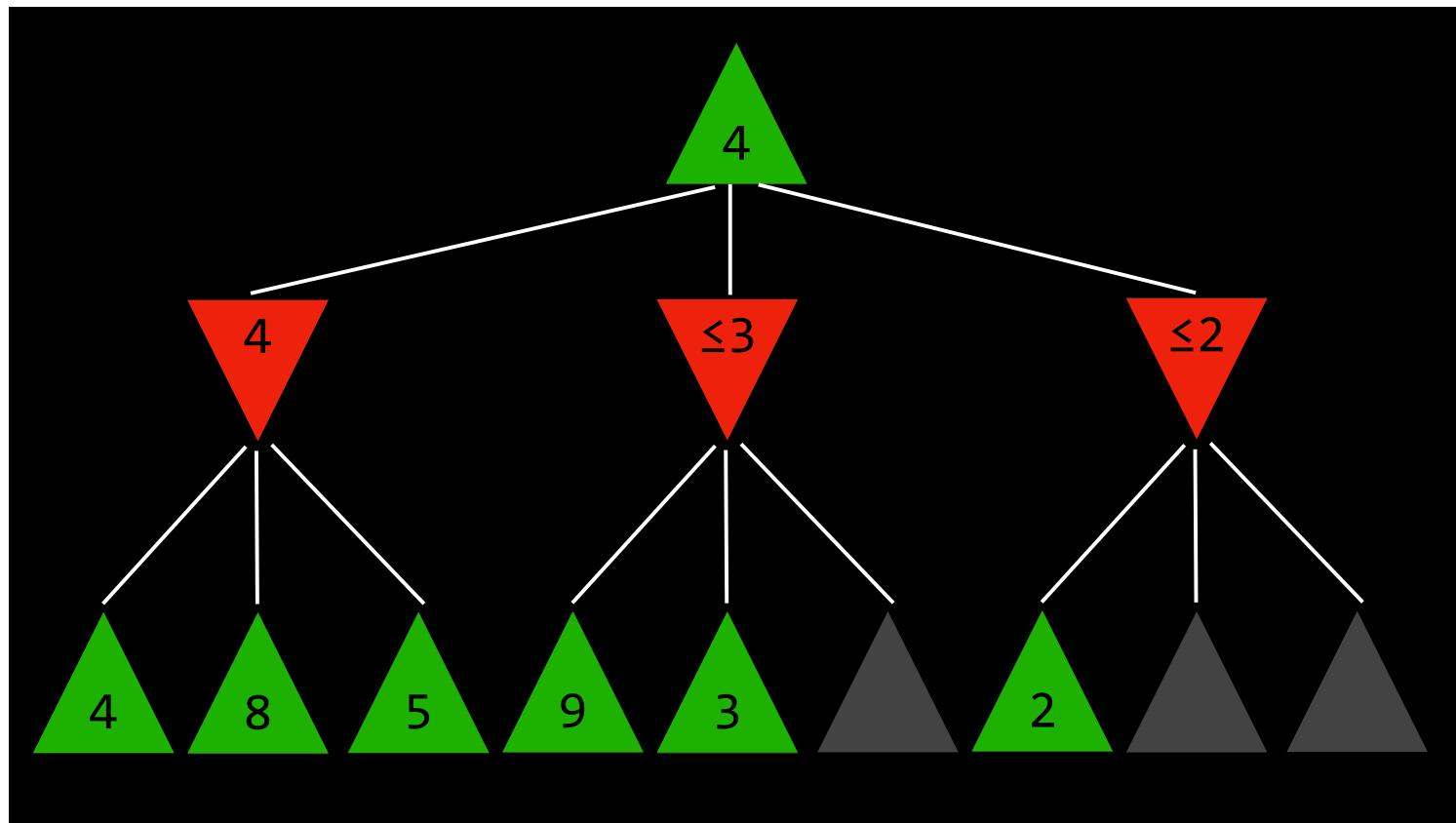
```
function MAX-VALUE(state):  
    if TERMINAL(state):  
        return UTILITY(state)  
    v = -∞  
    for action in ACTIONS(state):  
        v = MAX(v, MIN-VALUE(RESULT(state, action)))  
    return v
```

# Minimax

```
function MIN-VALUE(state):  
    if TERMINAL(state):  
        return UTILITY(state)  
     $v = \infty$   
    for action in ACTIONS(state):  
         $v = \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(\textit{state}, \textit{action}))))$   
    return  $v$ 
```

# Optimizations





# **Alpha-Beta Pruning**

255,168

total possible Tic-Tac-Toe games

288,000,000,000

total possible chess games  
after four moves each

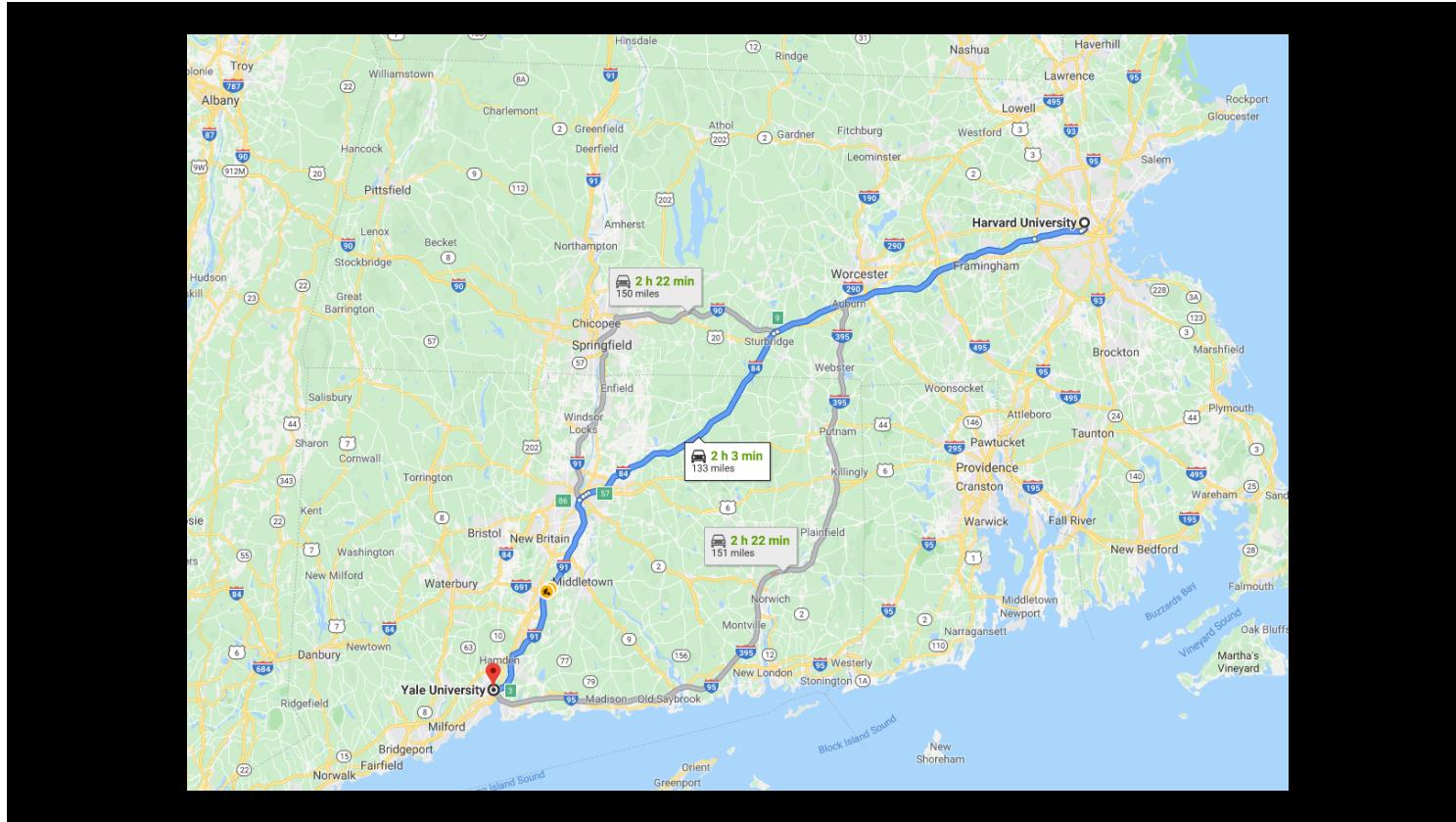
$10^{29000}$

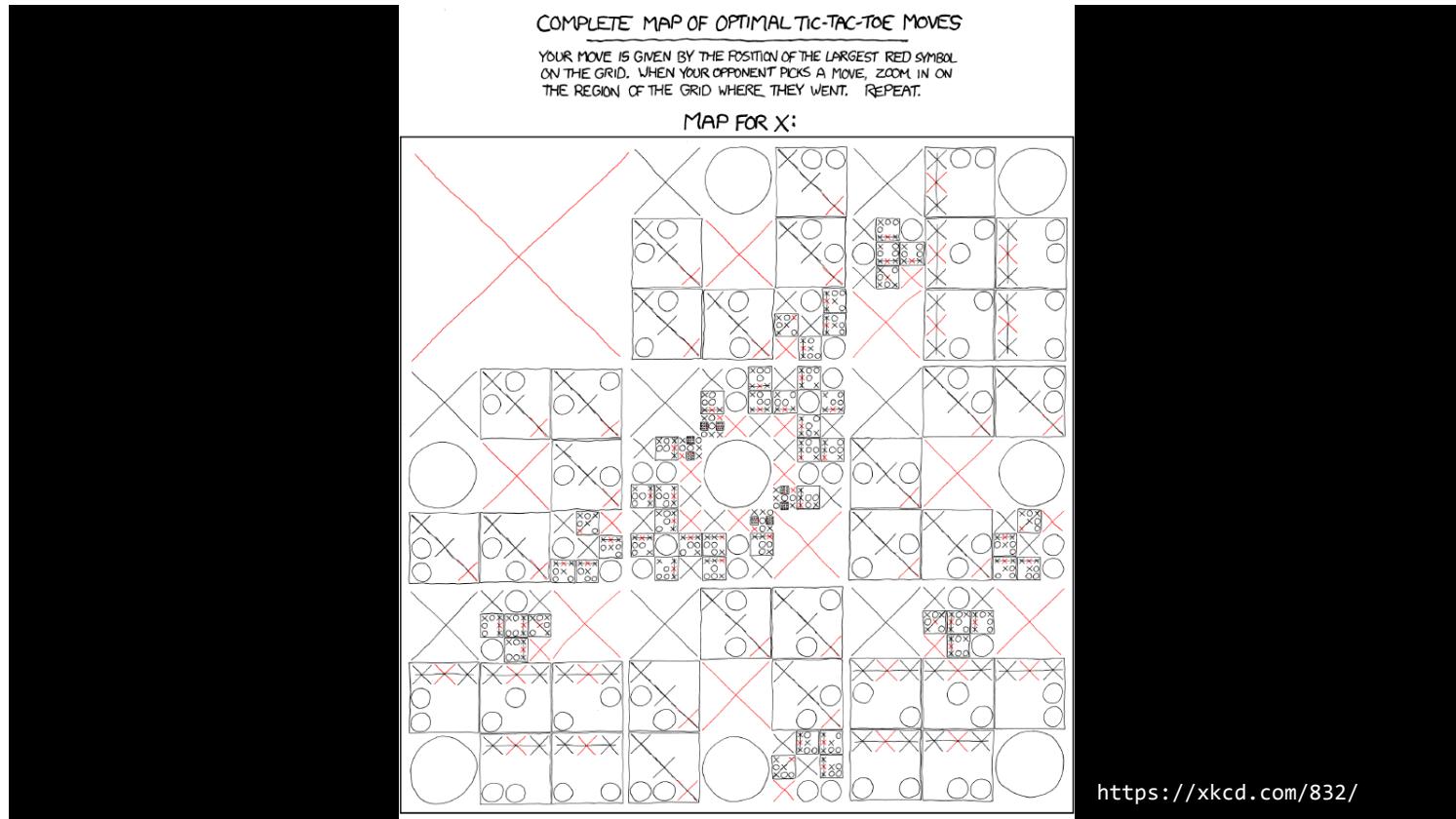
total possible chess games  
(lower bound)

# Depth-Limited Minimax

# **evaluation function**

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





# Search

Introduction to  
**Artificial Intelligence**  
with Python