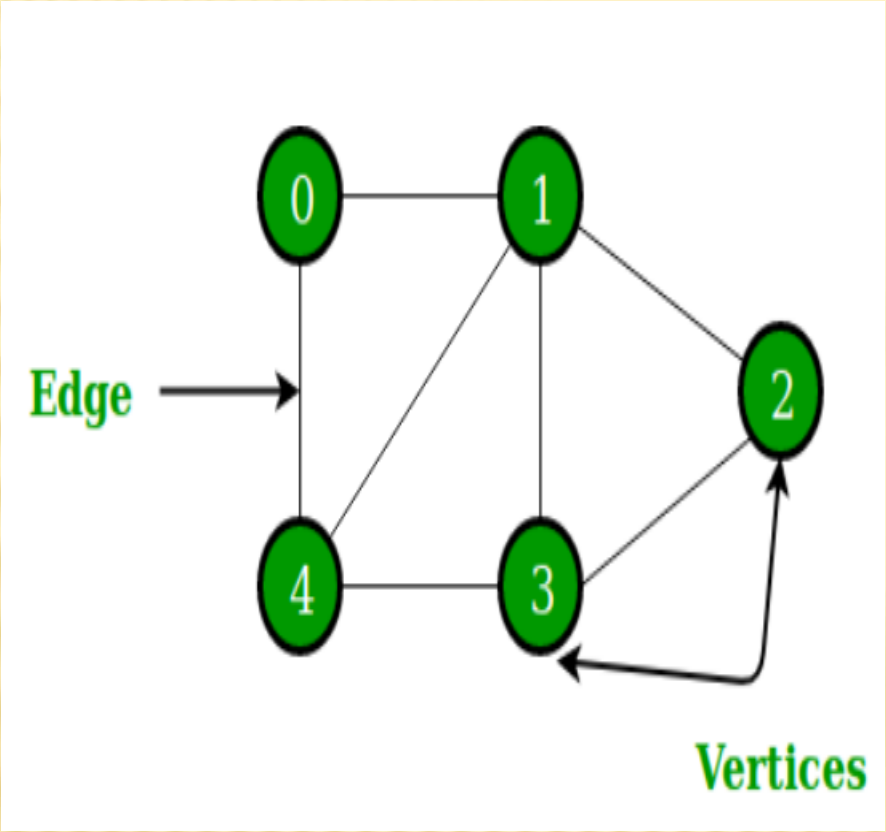
DSA MINI PROJECT

Visualisation Of Path Finding Algorithms In Graphs

❖GRAPH DATA STRUCTURE

GRAPH

- A Graph is a non-linear data structure consisting of nodes and edges.
- A finite set of vertices also called as nodes.
- A finite set of ordered pair of the form (u, v) called as edge.
- Graph can be defined as, A Graph consists of a finite set of vertices(or nodes) and set of Edges which connect a pair of nodes.
- Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network.



In this Graph, the set of vertices V = {0,1,2,3,4} and the set of edges E = {01, 12, 23, 34, 04, 14, 13}.

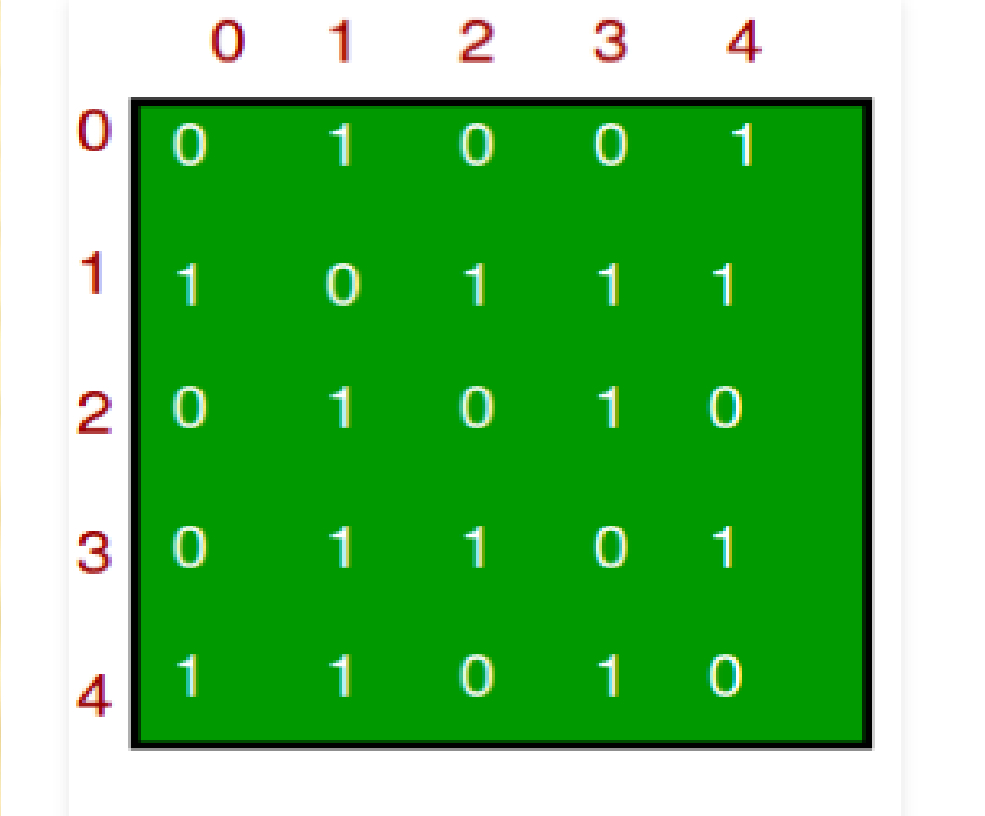
Graph and its representations

- The following two are the most commonly used representations of a graph.
  1. Adjacency Matrix
  2. Adjacency List

ADJACENCY MATRIX

- Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[ ][ ], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j.
- Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

The adjacency matrix example graph is:

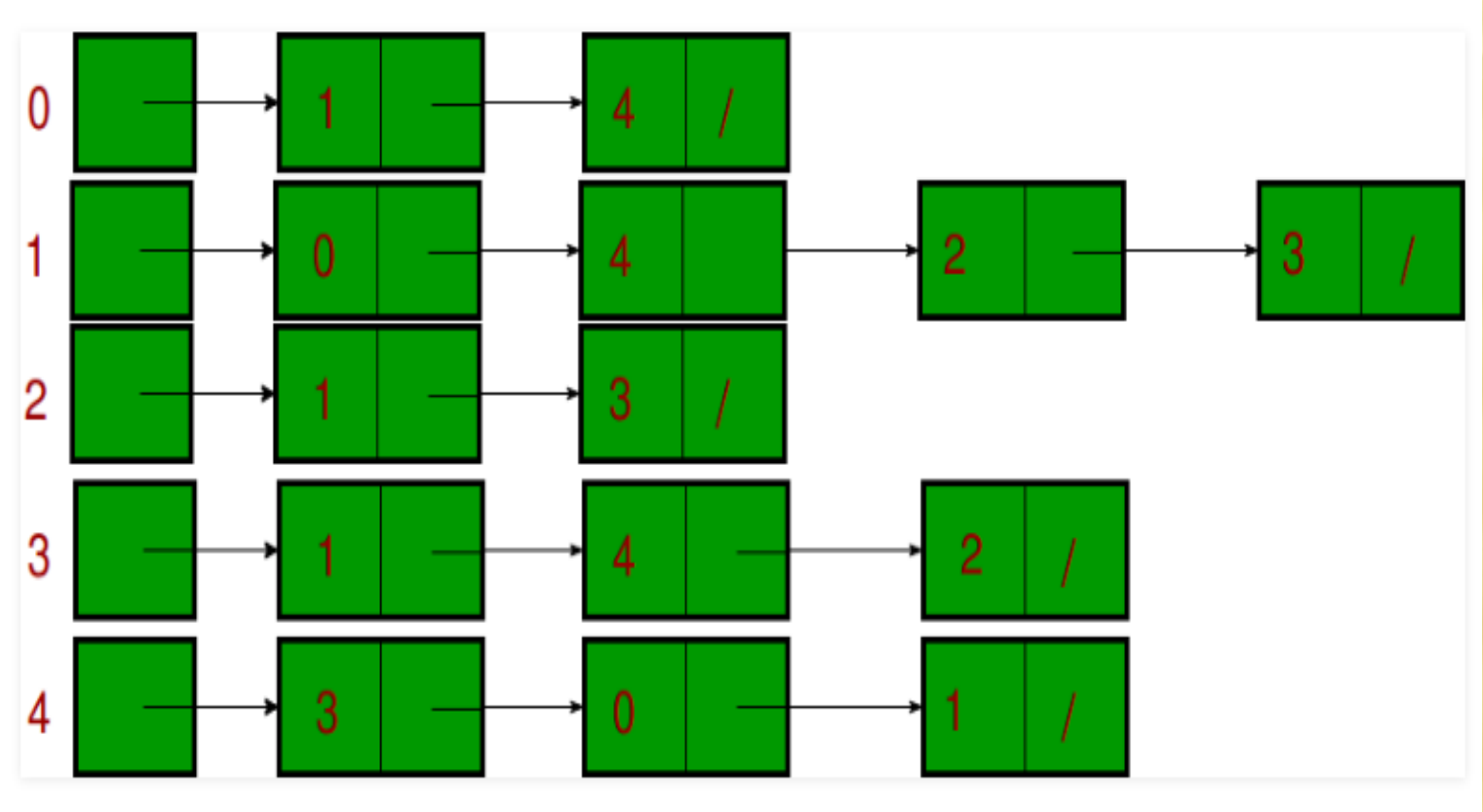


	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

ADJACENCY LIST:

- An array of lists is used. The size of the array is equal to the number of vertices. Let the array be an array[ ].
- An entry array[i] represents the list of vertices adjacent to the *i*th vertex.
- This representation can also be used to represent a weighted graph.
- The weights of edges can be represented as lists of pairs. Following is the adjacency list representation of the above graph.

Following is the adjacency list representation of the above graph is



APPLICATIONS OF GRAPH DATA STRUCTURE

In **Computer science** graphs are used to represent the flow of computation.

Google maps uses graphs for building transportation systems, where intersection of two(or more) roads are considered to be a vertex and the road connecting two vertices is considered to be an edge, thus their navigation system is based on the algorithm to calculate the shortest path between two vertices.

In **Facebook**, users are considered to be the vertices and if they are friends then there is an edge running between them. Facebook's Friend suggestion algorithm uses graph theory. Facebook is an example of **undirected graph**.

- In **World Wide Web**, web pages are considered to be the vertices. There is an edge from a page u to other page v if there is a link of page v on page u. This is an example of **Directed graph**. It was the basic idea behind Google Page Ranking Algorithm.
- In **Operating System**, we come across the Resource Allocation Graph where each process and resources are considered to be vertices. Edges are drawn from resources to the allocated process, or from requesting process to the requested resource. If this leads to any formation of a cycle then a deadlock will occur.

WELCOME

EVERYONE

- Breadth First Search Algorithm(BFS)
- Depth First Search Algorithm(DFS)

What is Breadth First Search Algorithm(BFS)?

- **Breadth First Search Algorithm(BFS)** is an algorithm for traversing or searching Tree or Graph data structure.
- A breadth-first search algorithm will follow multiple directions at the same time, taking one step in each possible direction before taking the second step in each direction.
- In this case, the frontier is managed as a queue data structure. The catchphrase you need to remember here is “first-in first-out.” In this case, all the new nodes add up in line, and nodes are being considered based on which one was added first (first come first served!).
- This results in a search algorithm that takes one step in each possible direction before taking a second step in any one direction.

- Pros:
- This algorithm is guaranteed to find the optimal solution.
- Cons:
- This algorithm is almost guaranteed to take longer than the minimal time to run.
- At worst, this algorithm takes the longest possible time to run.

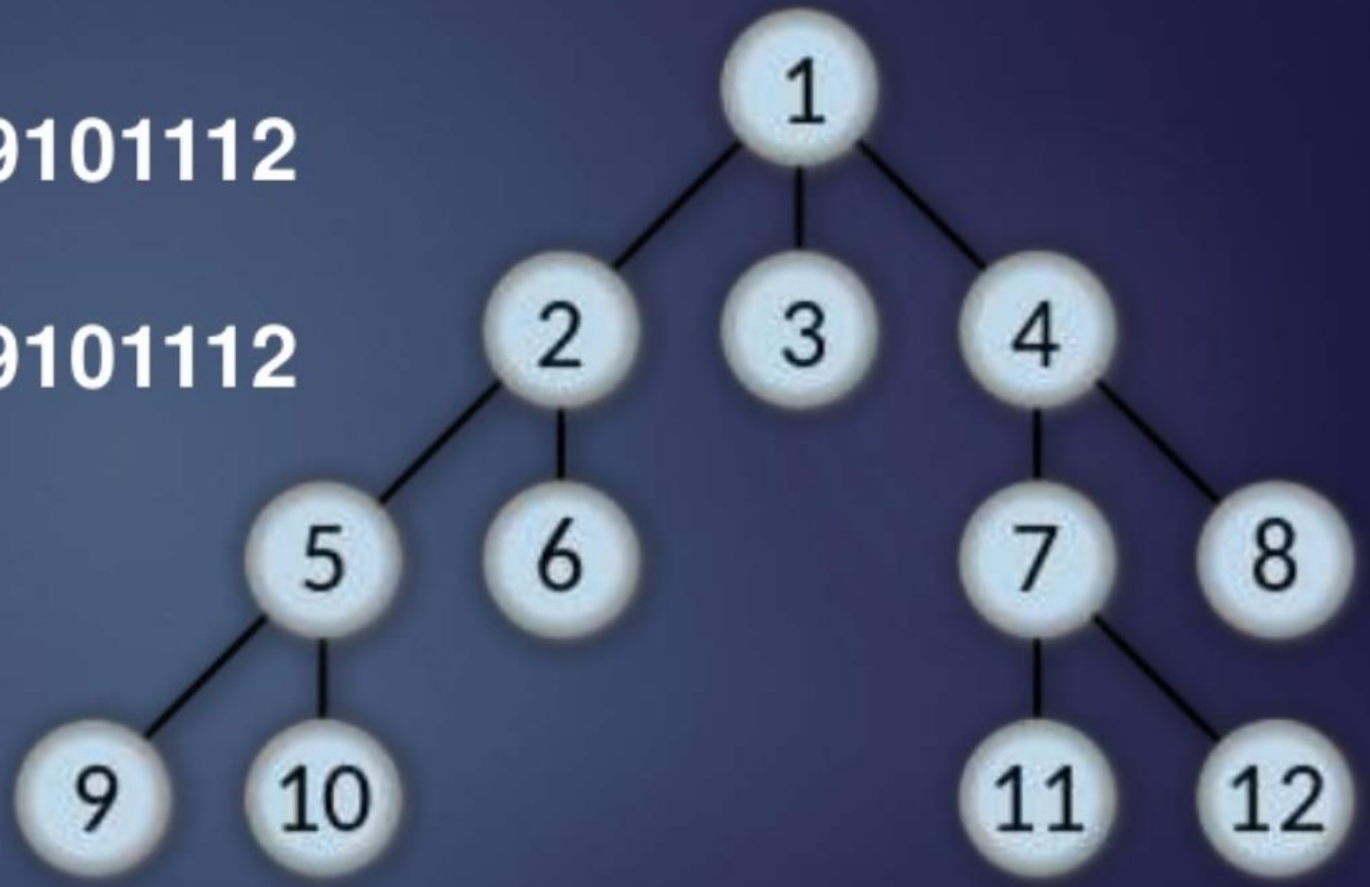
CODE EXAMPLE:

```
# Define the function that removes a node from the frontier and returns it.
        def remove(self):
# Terminate the search if the frontier is empty, because this means that there is no solution.
        if self.empty():
 raise Exception("empty frontier")
        else:
# Save the oldest item on the list (which was the first one to be added)
         node = self.frontier[0]
# Save all the items on the list besides the first one (i.e. removing the first node)
         self.frontier = self.frontier[1:]
         return node
```

BFS and Level Order Example

Level-Order: 1 234 5678 9101112

BFS: 1 234 5678 9101112



DEPTH FIRST SEARCH ALGORITHM(DFS)

- The aim of the DFS algorithm is travers the graph in such a way that is try to go for from the root node. Stack is use in the implementation of the depth first search. Lets see how depth first search work with respect to the following graph.

- A depth-first search algorithm exhausts each one direction before trying another direction.
- In these cases, the frontier is managed as a stack data structure. The catchphrase you need to remember here is “last-in first-out.” After nodes are being added to the frontier, the first node to remove and consider is the last one to be added.
- This results in a search algorithm that goes as deep as possible in the first direction that gets in its way while leaving all other directions for later.

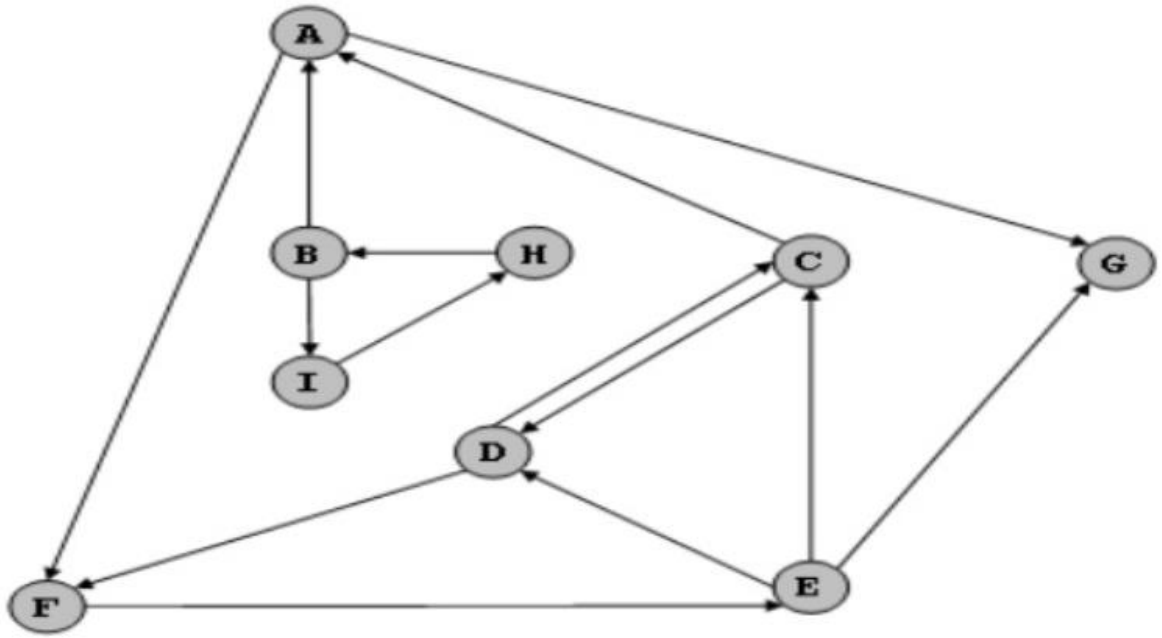
- Pros:
- At best, this algorithm is the fastest.
- If it “lucks out” and always chooses the right path to the solution (by chance), then depth-first search takes the least possible time to get to a solution.
- Cons:
- It is possible that the found solution is not optimal.
- At worst, this algorithm will explore every possible path before finding the solution, thus taking the longest possible time before reaching the solution.

Code Example:

```
    # Define the function that removes a node from the frontier and returns it.
        def remove(self):
    # Terminate the search if the frontier is empty, because this means that there is no solution.
        if self.empty():
  raise Exception("empty frontier")
        else:
 # Save the last item in the list (which is the newest node added)
        node = self.frontier[-1]
  # Save all the items on the list besides the last node (i.e. removing the last node)
         self.frontier = self.frontier[:-1]
         return node
```

DFS EXAMPLE:

Directed Depth First Search



Adjacency Lists

A: F G
B: A H
C: A D
D: C F
E: C D G
F: E
G:
H: B
I: H

Dijkstra's Algorithm

Dijkstra's algorithm:

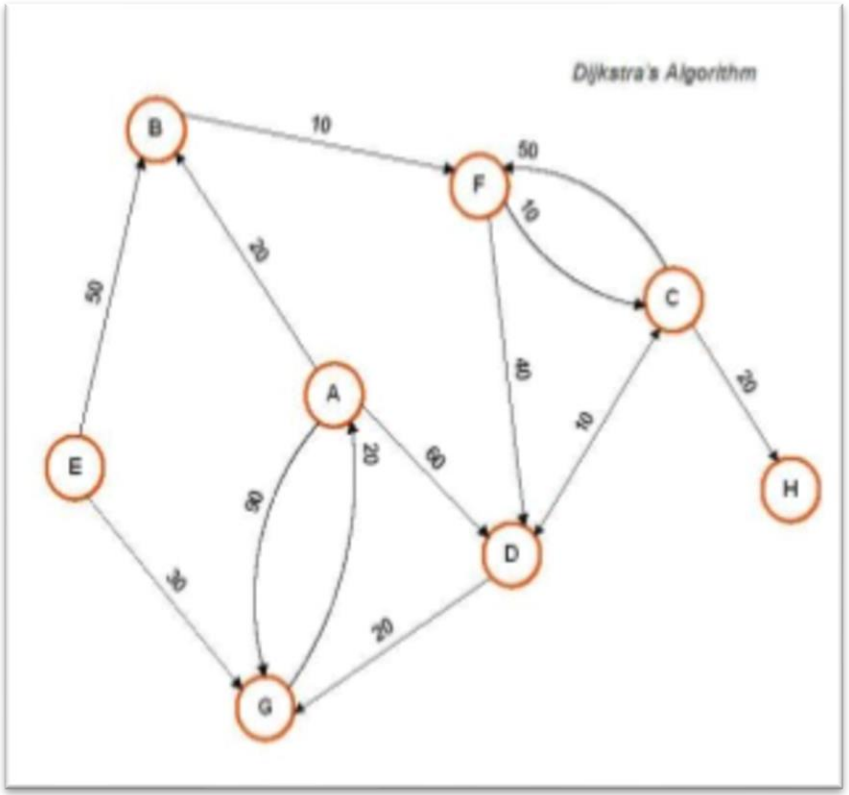
- Dijkstra's algorithm: It is a solution to the single-source shortest path problem in graph theory. Works on both directed and undirected graphs. However, all edges must have nonnegative weights.
- Approach: Greedy.
- Input: Weighted graph G={E,V} and source vertex, such that all edge weights are non-negative.
- Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex to all other vertices.

How it works?

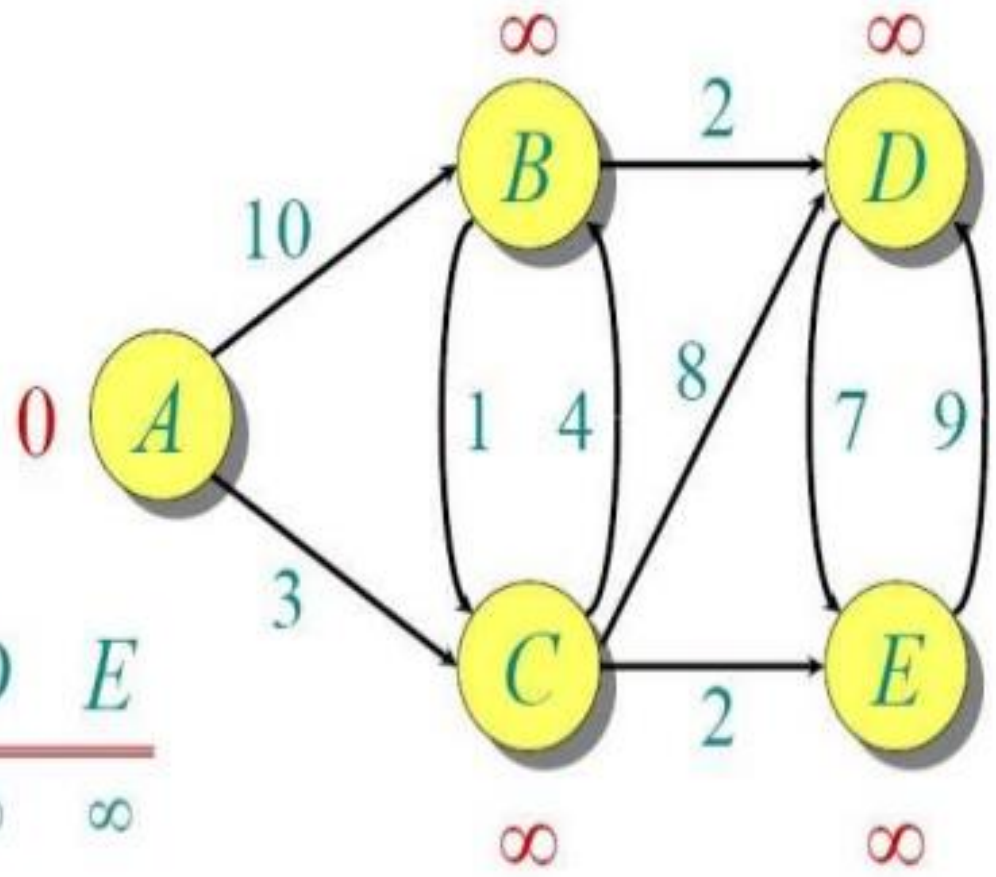
- This algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex.
- For example, if the vertices of the graph represent cities and edge path cost represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities.

Graph Algorithm

- In this interconnected 'Vertex' we'll use 'Dijkstra's Algorithm'.
- To use this algorithm in this network we have to start from a decided vertex and then continue to others.

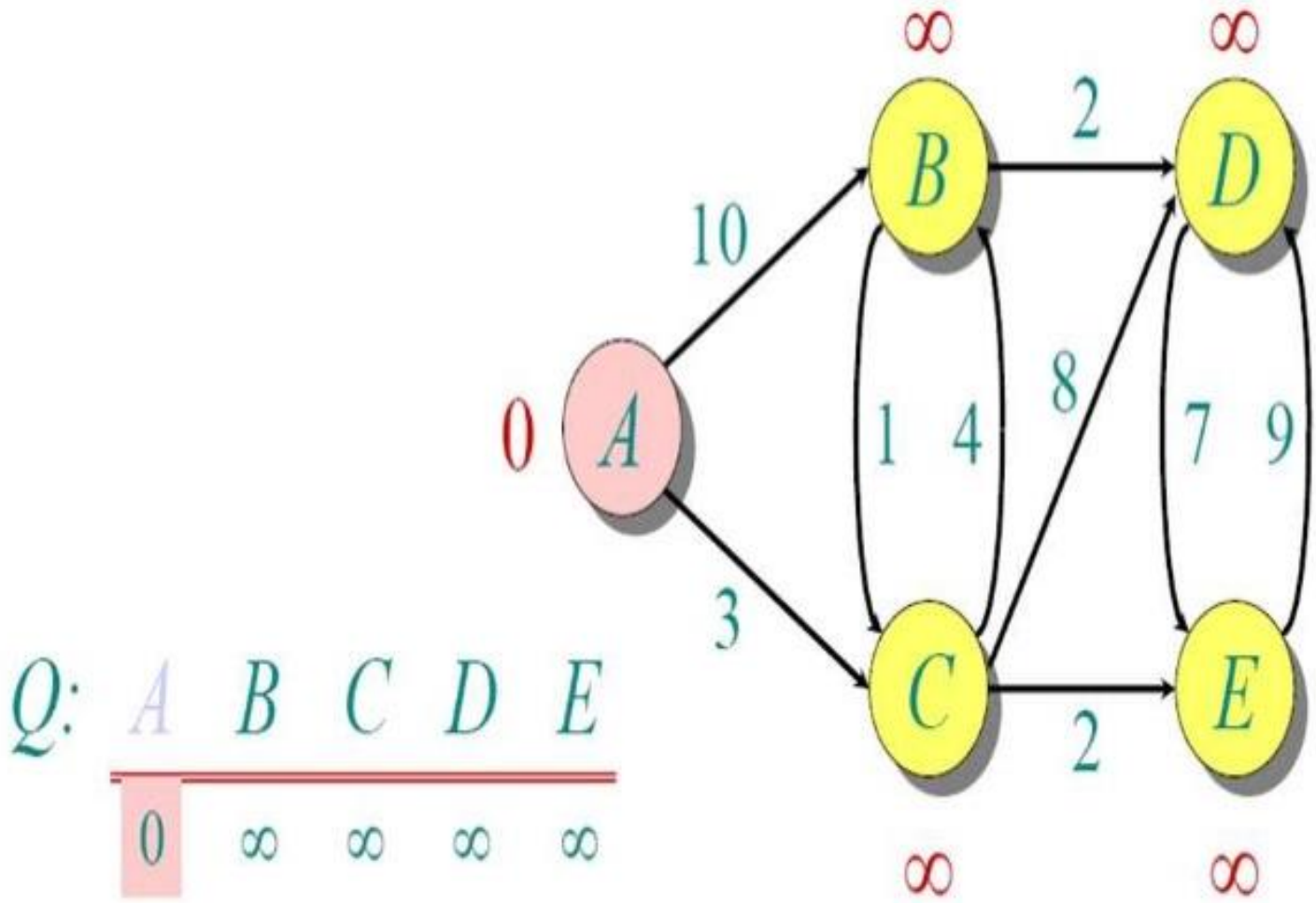


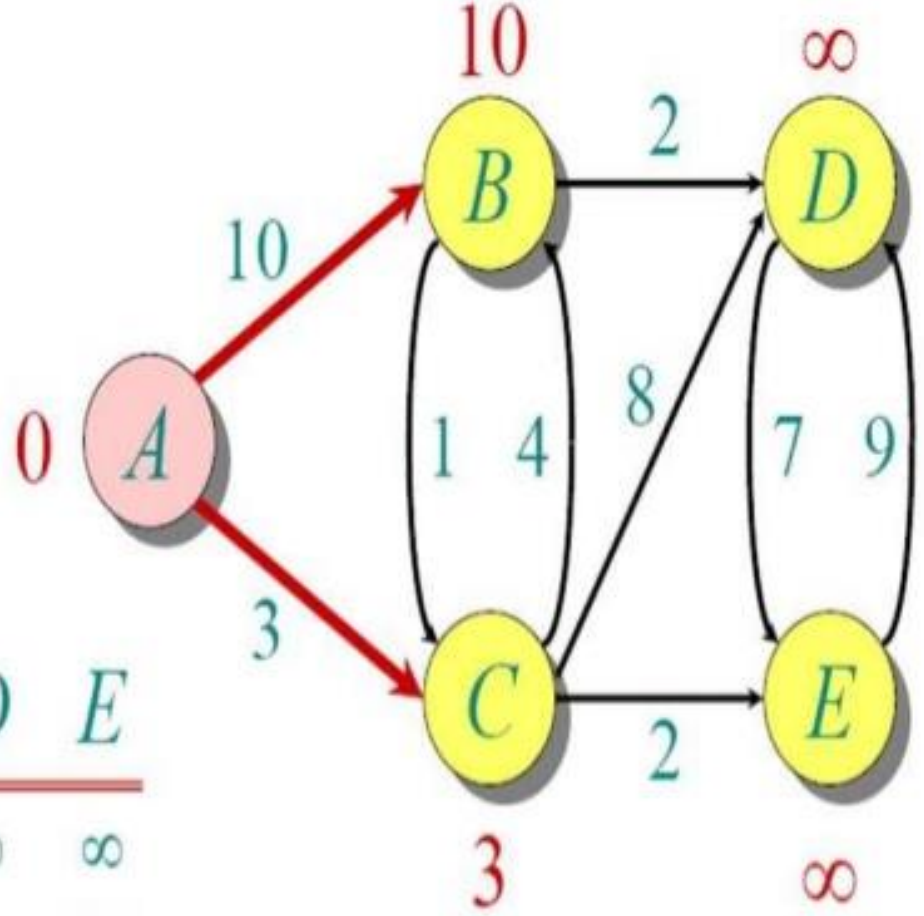
Initialize:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

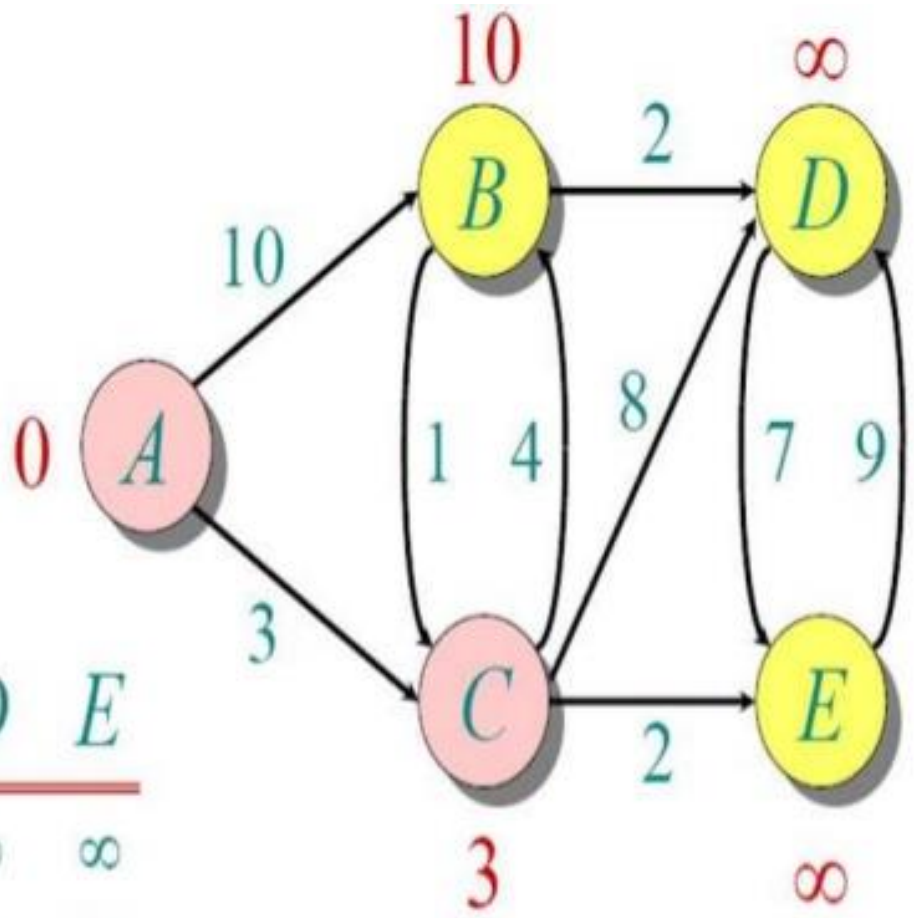
S: {}


∞
∞
B
2
D
10
0
A
1
4
8
7
9
3
C
E
2
∞
∞
Q: A B C D E
0 ∞ ∞ ∞ ∞



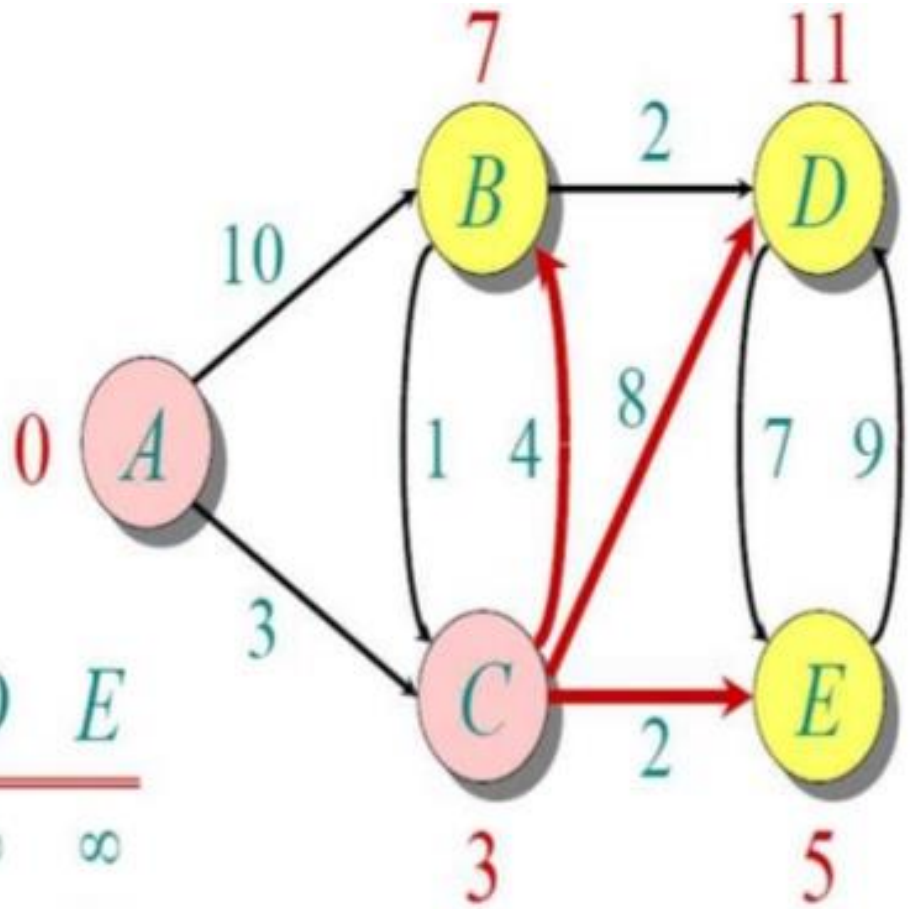
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }



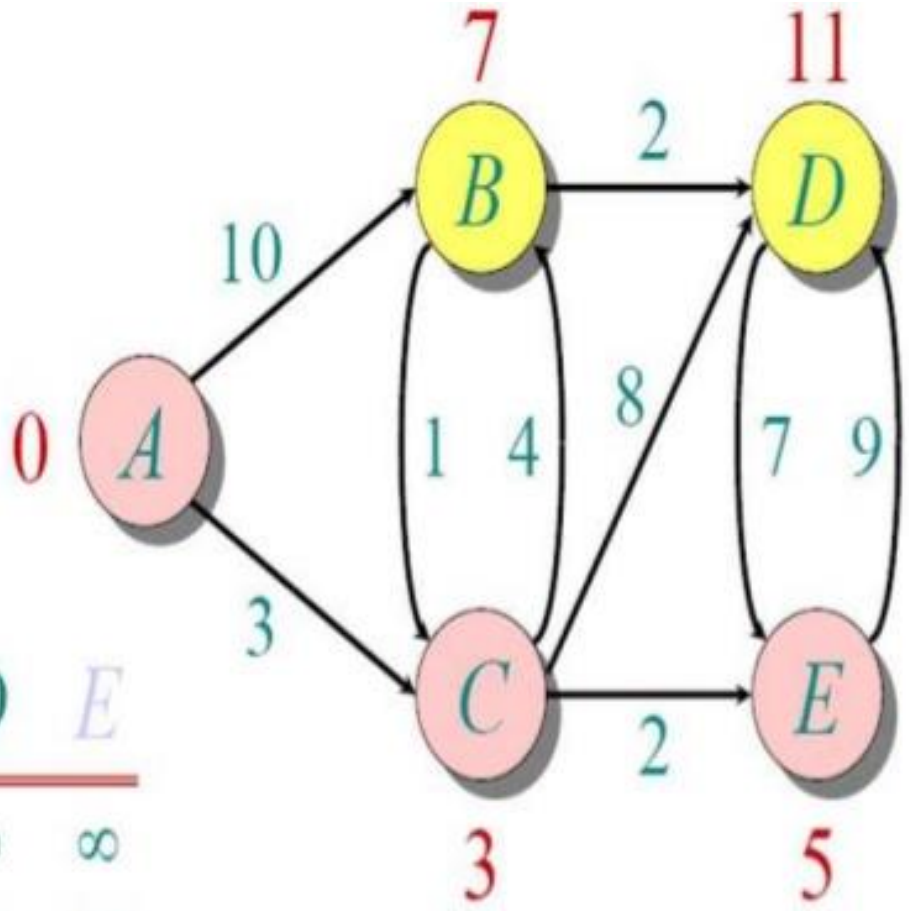
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }



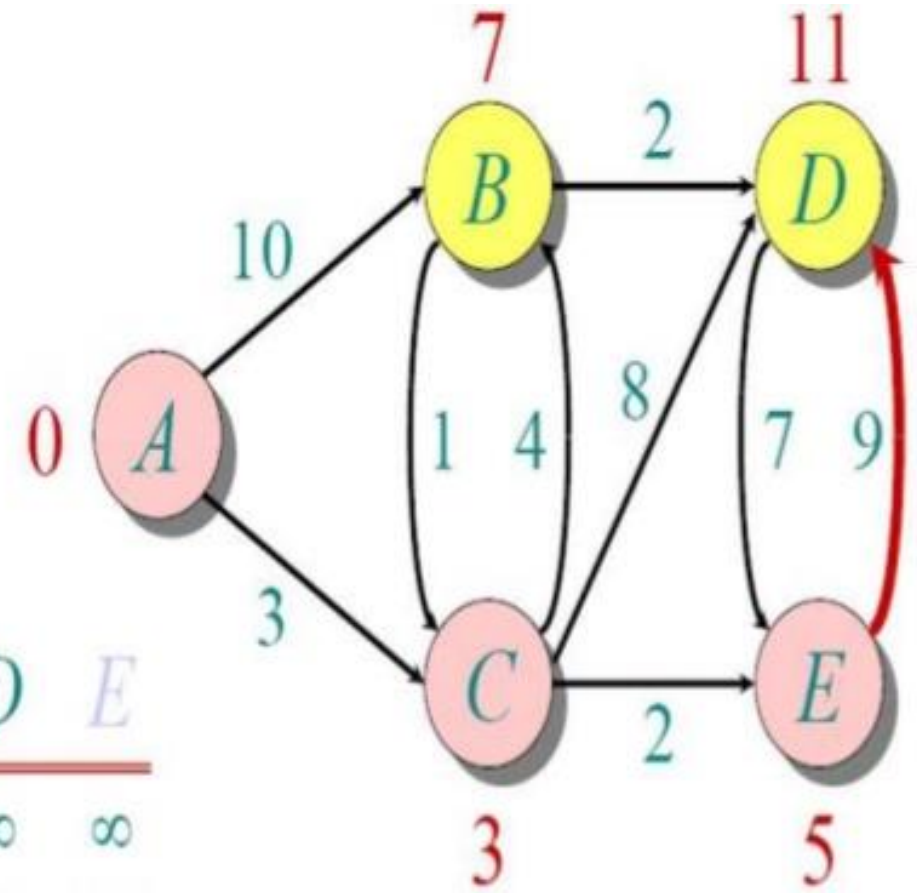
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

$S: \{A, C\}$



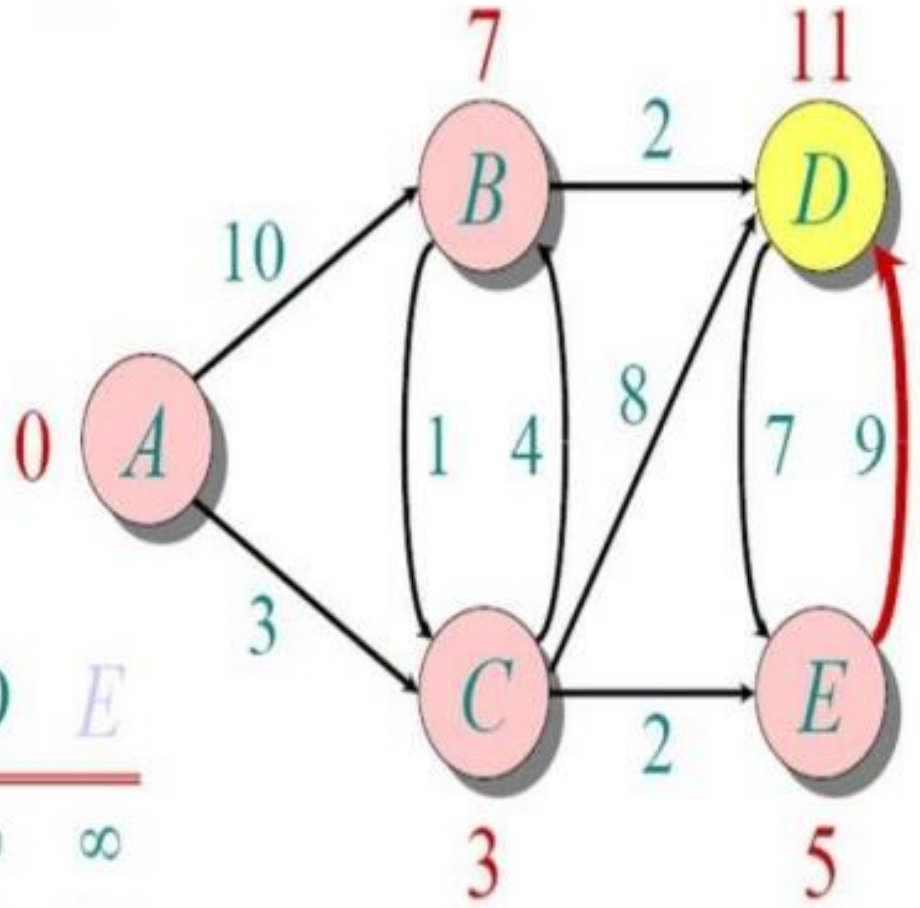
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

$S: \{A, C, E\}$



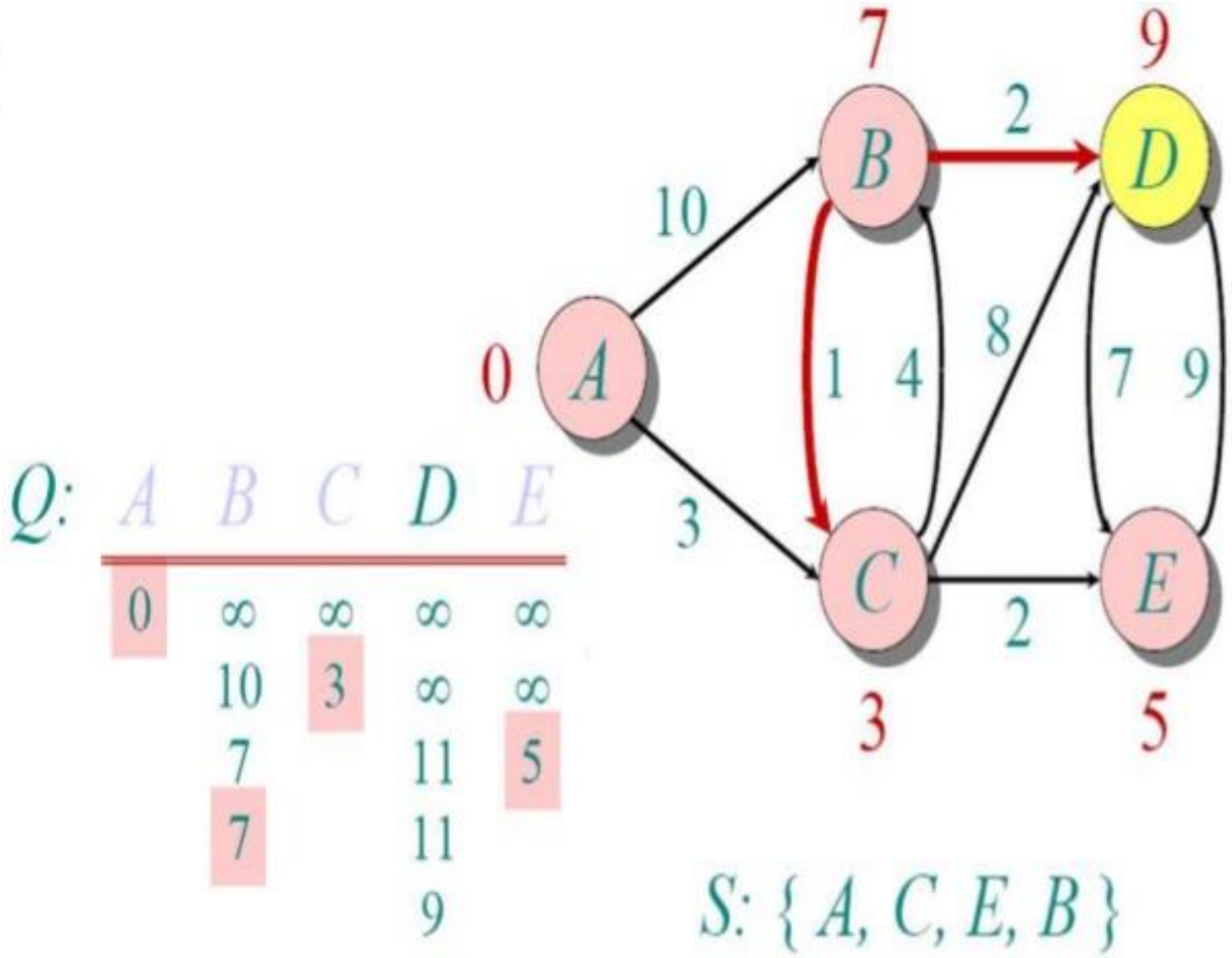
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

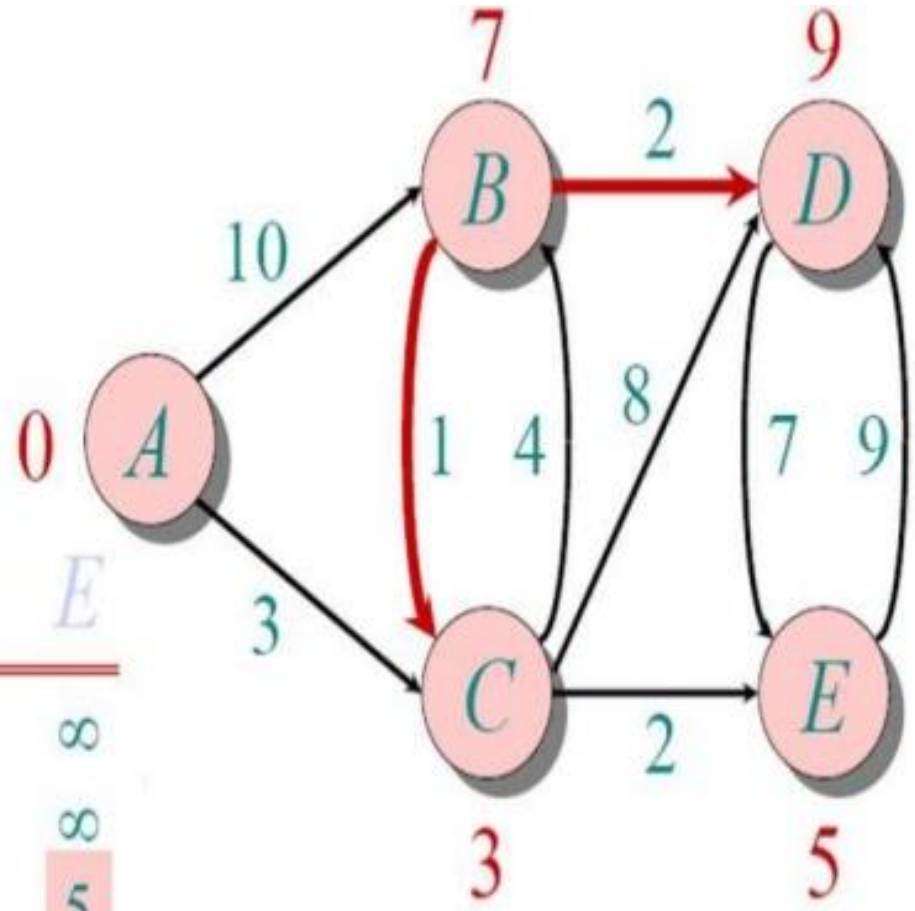
$S: \{A, C, E\}$



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

$S: \{ A, C, E, B \}$


7
9
2
B
D
10
0
A
1
4
8
7
9
3
C
E
2
3
5
Q: A B C D E
0 ∞ ∞ ∞ ∞
10 3 ∞ ∞
7 11 5
7 11
9
S: { A, C, E, B }



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B, D }

Implementation & Runtime:

The simplest implementation is to store vertices in an Array or Linked list. This will produce a running time of

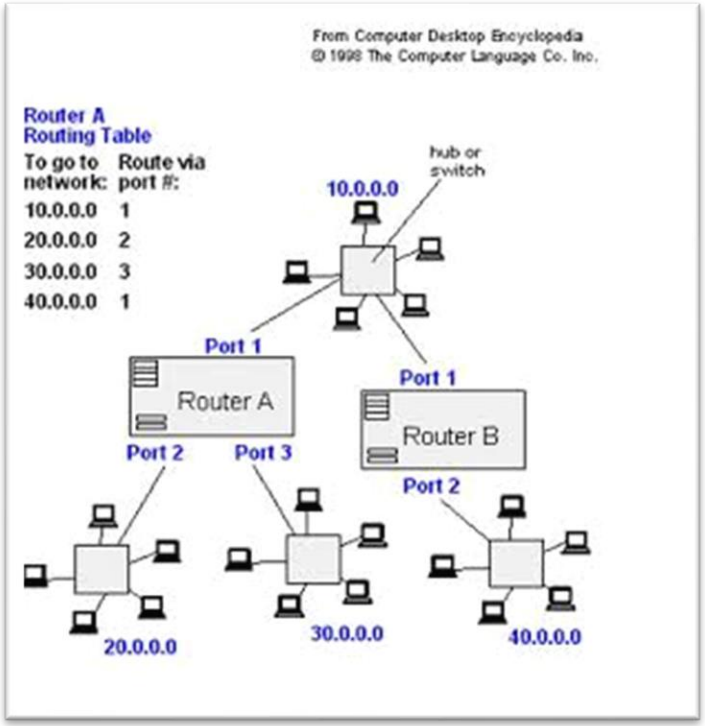
O(|V|^2 + |E|)

For graphs with very few edges and many nodes, it can be implemented more efficiently storing the graph in an adjacency list using a Binary Heap or Priority Queue. This will produce a running time of

Q((|E| + |V|) log |V|)

Applications:

- Traffic Information System are most prominent use.
- Mapping (Map Quest, Google Maps).
- Routing Systems.

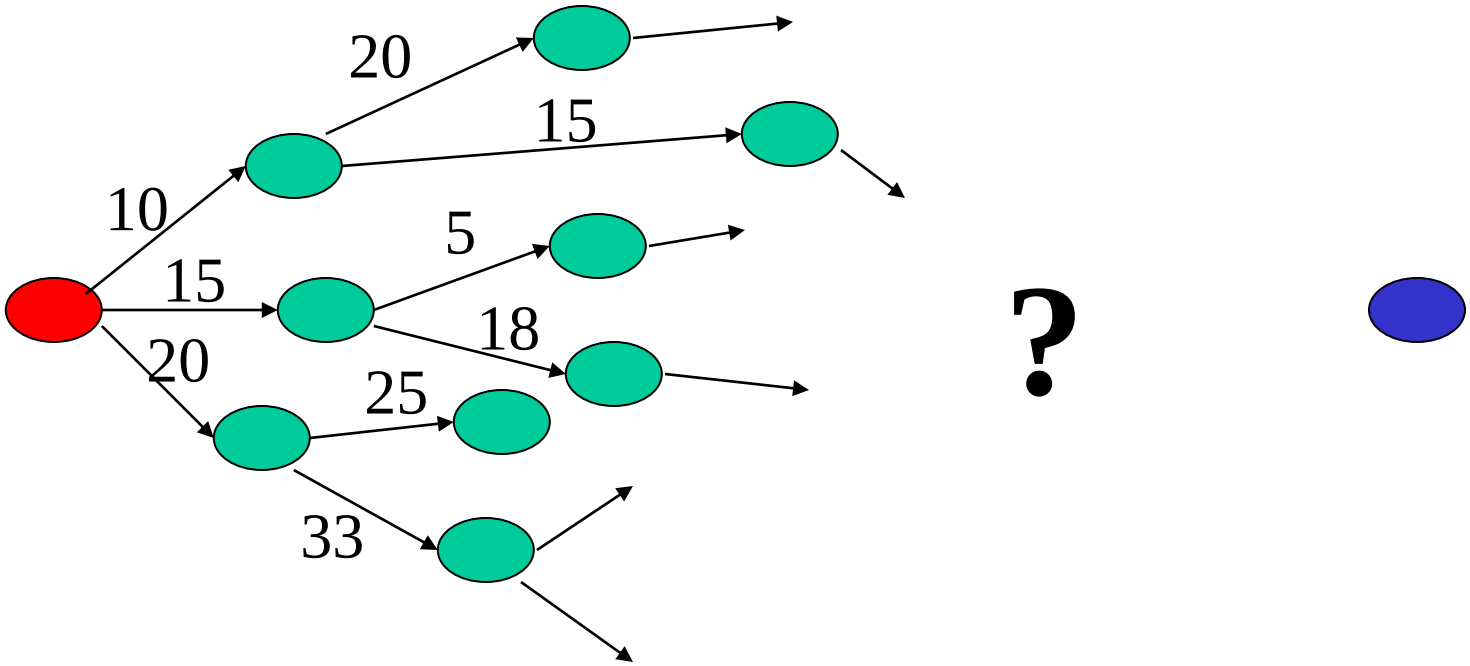


The A* Algorithm

Tejas Kadam

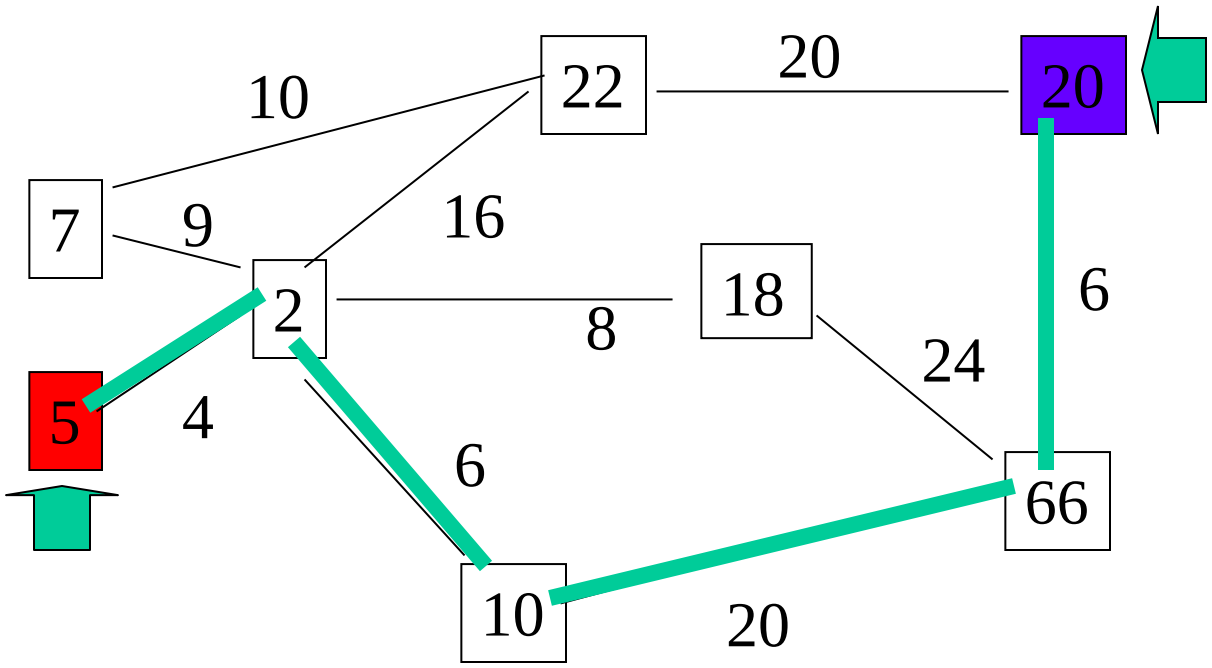
The Search Problem

Starting from a node n find the shortest path to a goal node g



Shortest Path

Consider the following weighted undirected graph:



We want: A path 5 → v1 → v2 → … → 20

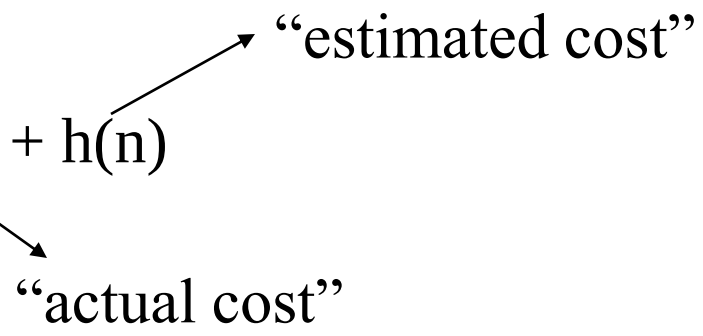
Such that g(20) = cost(5→v1) + cost(v1→ v2) + … + cost (→20) is minimum

The A* Search

- **Difficulty**: we want to still be able to generate the path with minimum cost

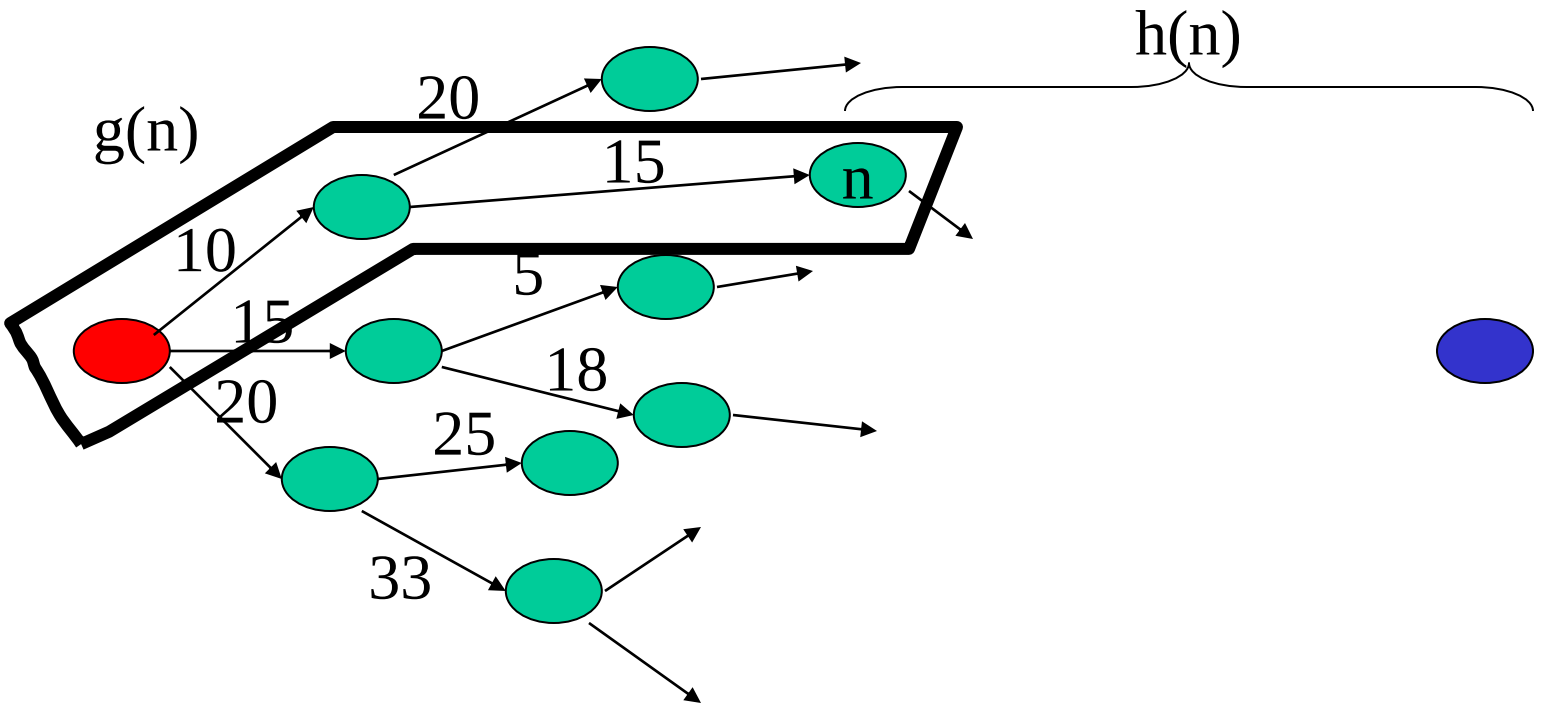
- A* is an algorithm that:
  - Uses heuristic to guide search
  - While ensuring that it will compute a path with minimum cost

- A* computes the function $f(n) = g(n) + h(n)$

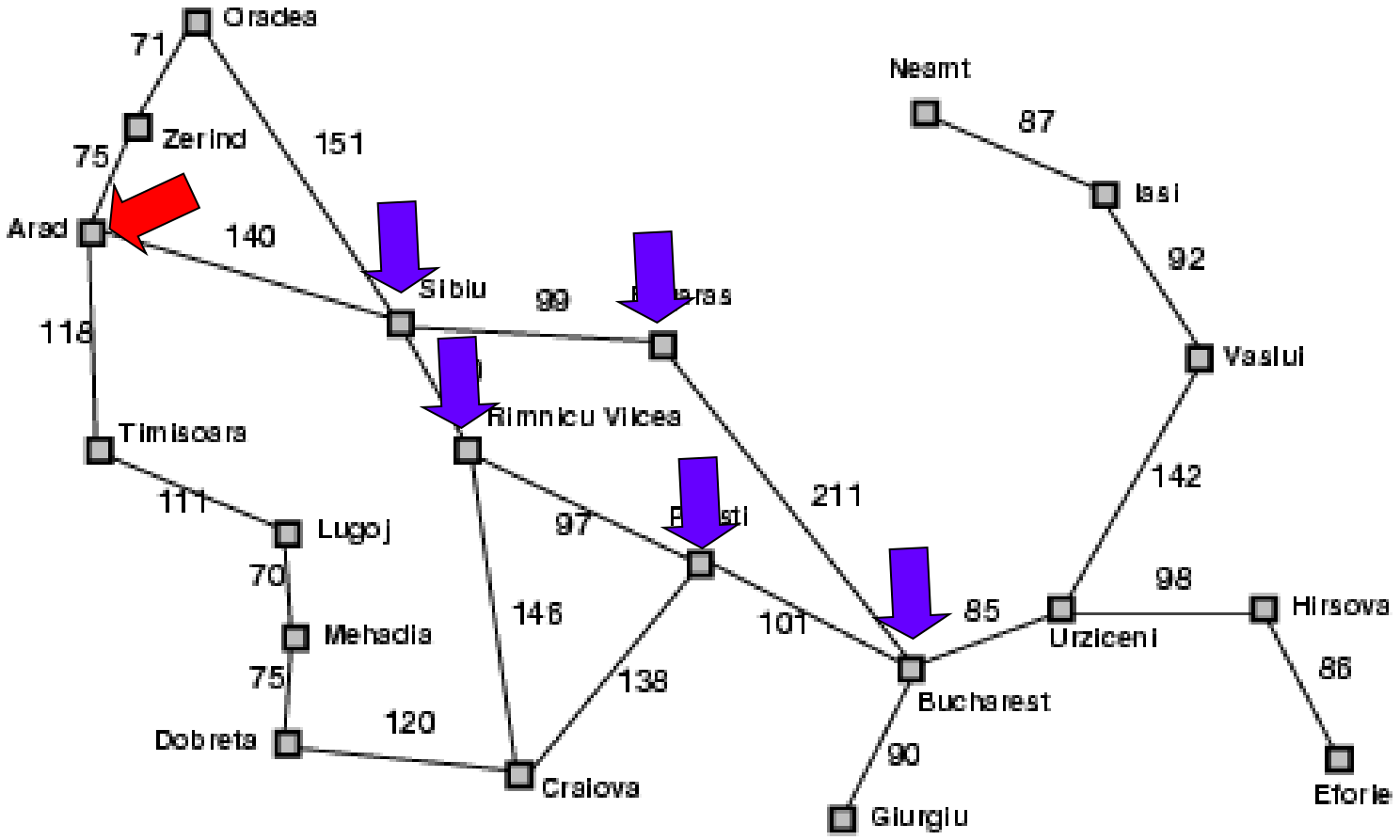


A*

- f(n) = g(n) + h(n)
  - g(n) = “cost from the starting node to reach n”
  - h(n) = “estimate of the cost of the cheapest path from n to the goal node”



Example



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

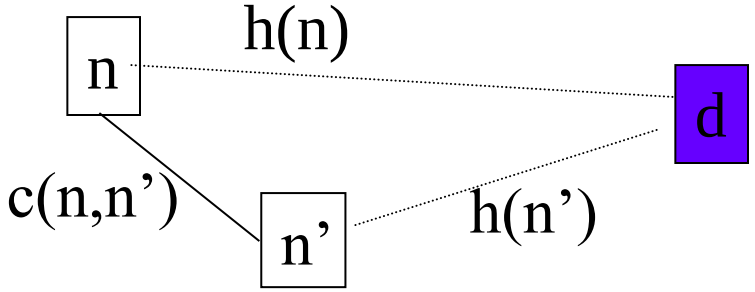
A*: minimize f(n) = g(n) + h(n)

Properties of A*

- A* generates an optimal solution if h(n) is an admissible heuristic and the search space is a tree:
  - h(n) is **admissible** if it never overestimates the cost to reach the destination node

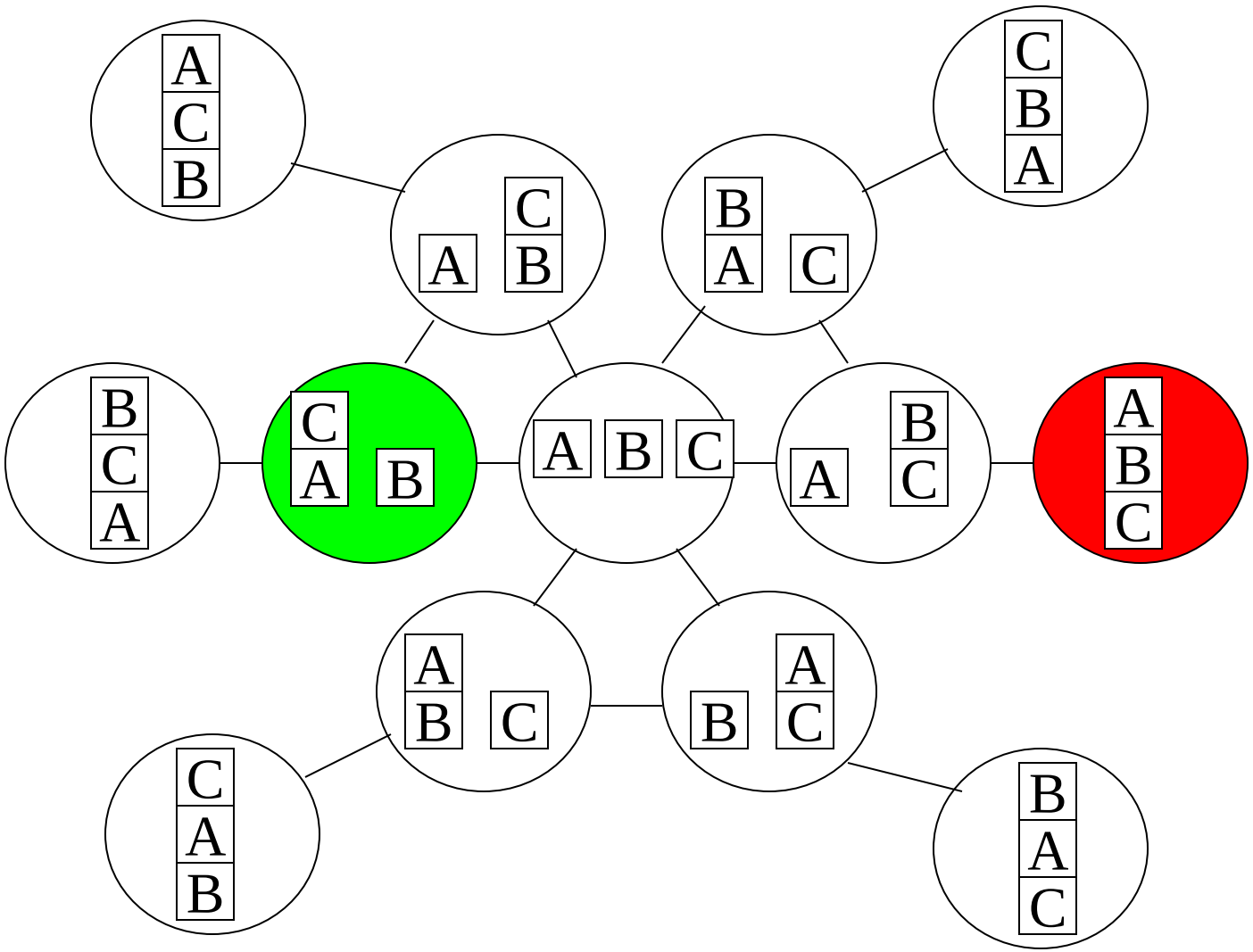
- A* generates an optimal solution if h(n) is a consistent heuristic and the search space is a graph:
  - h(n) is **consistent** if for every node n and for every successor node n' of n:

$$h(n) \leq c(n,n') + h(n')$$



- If h(n) is consistent then h(n) is admissible
- Frequently when h(n) is admissible, it is also consistent

Using A* in Planning

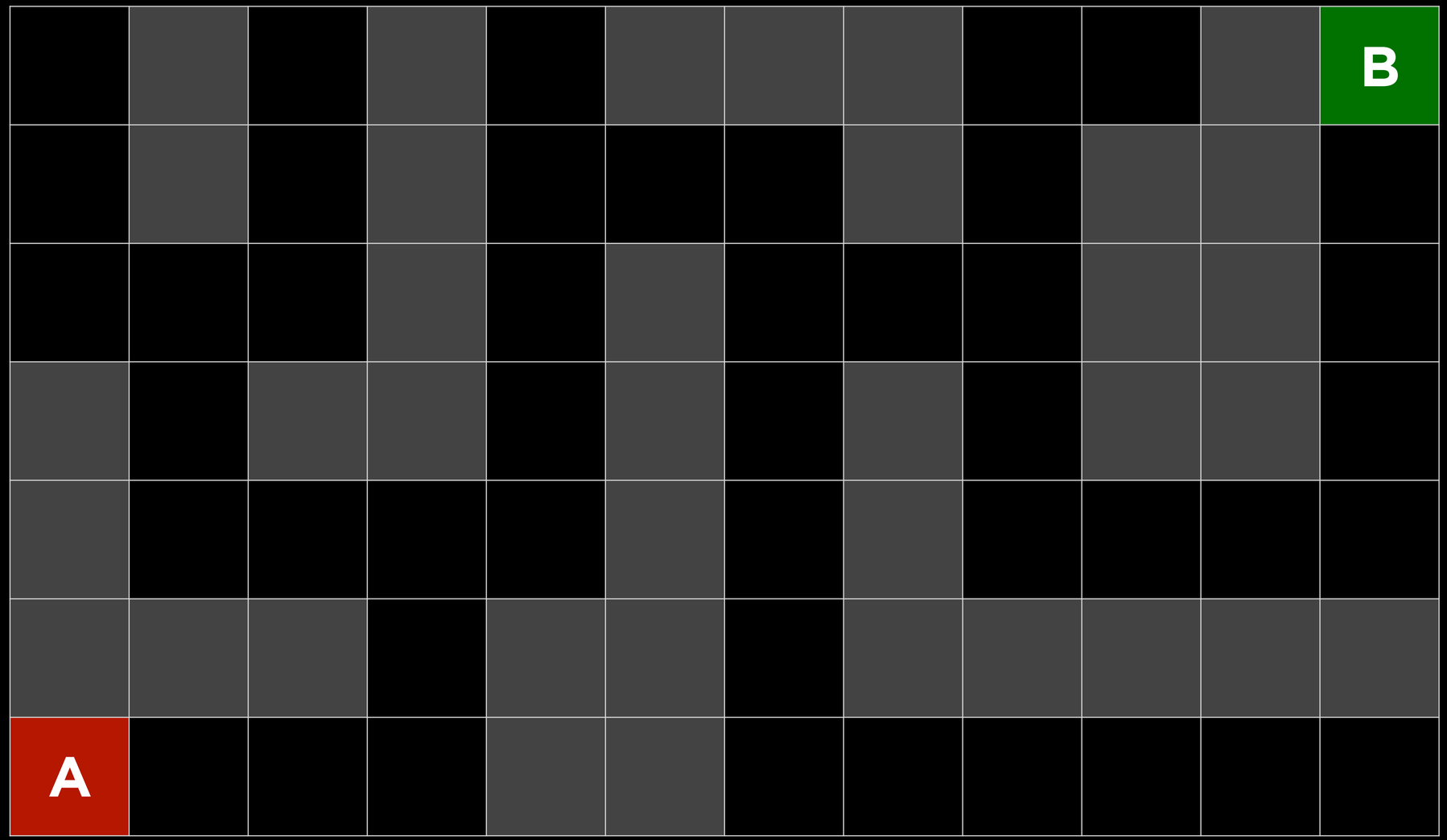


h(n) = "# of goals remaining to be satisfied" g(n) = "# of steps so far"

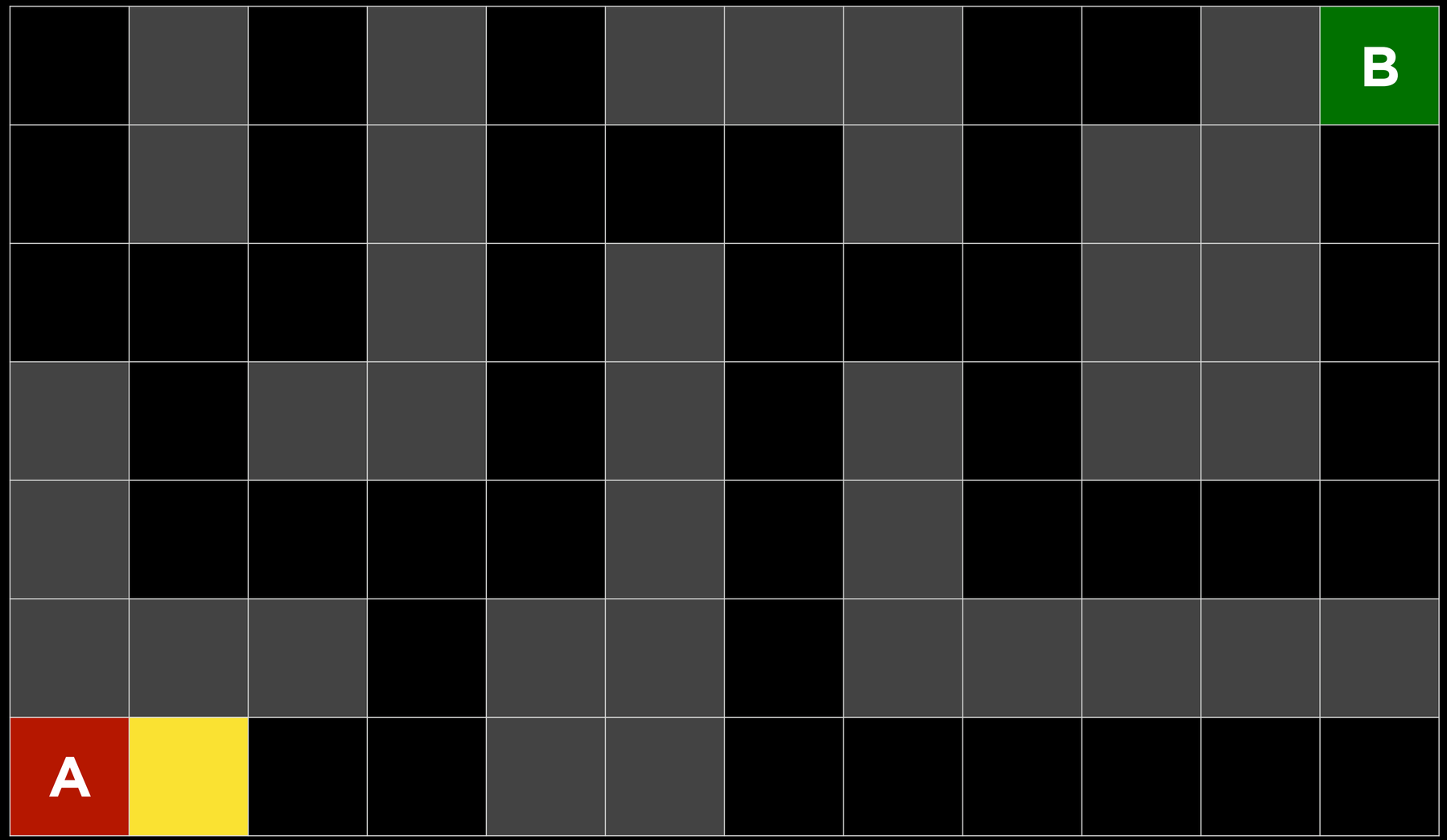
A* in Games

- Path finding (duh!)
  - We will have several presentations on the topic
  - We will see that sometimes even A* speed improvements are not sufficient
  - Additional improvements are required
- A* can be used for planning moves computer-controlled player (e.g., chess)
- F.E.A.R. uses A* to plan its search

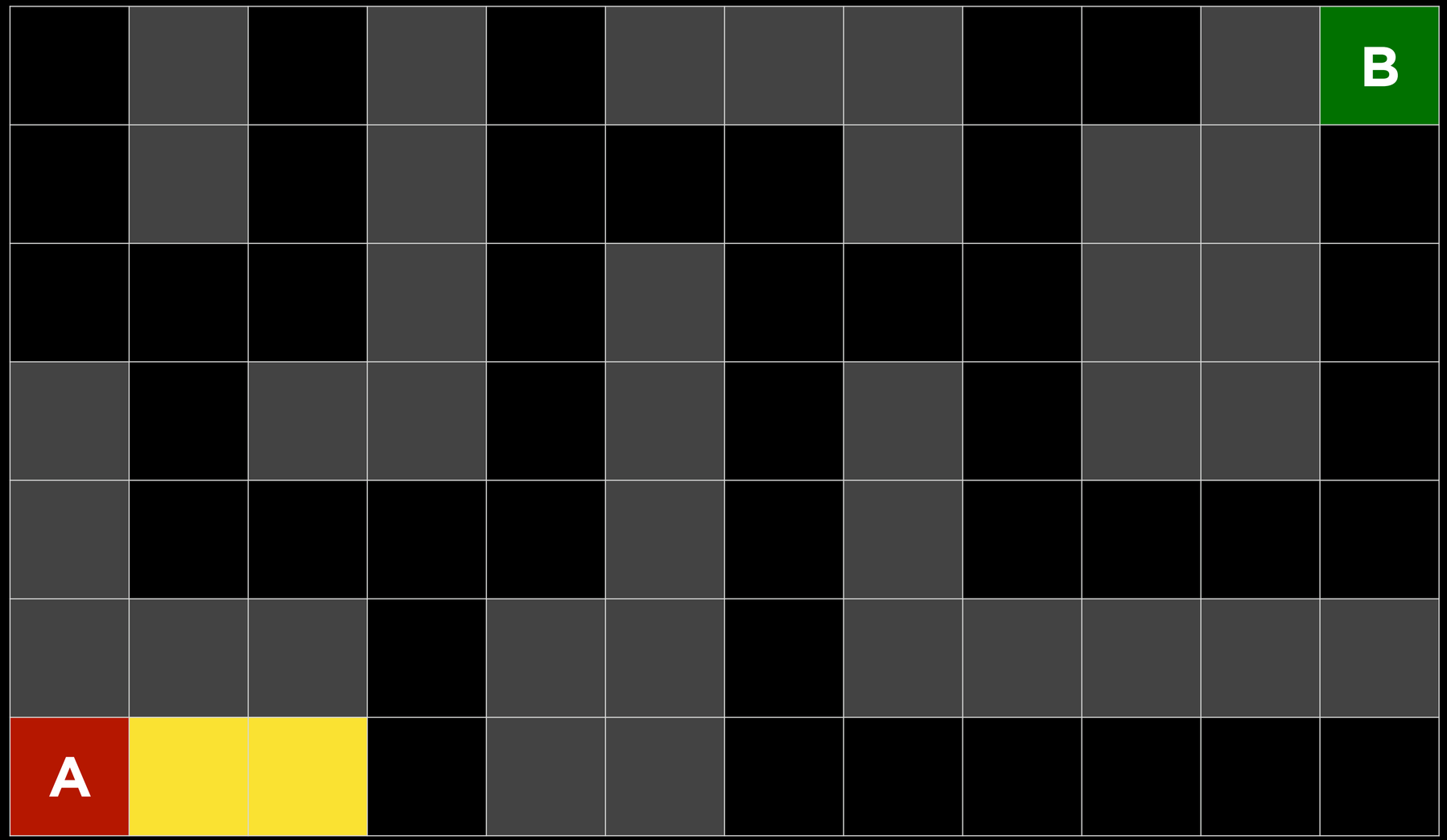
Depth-First Search



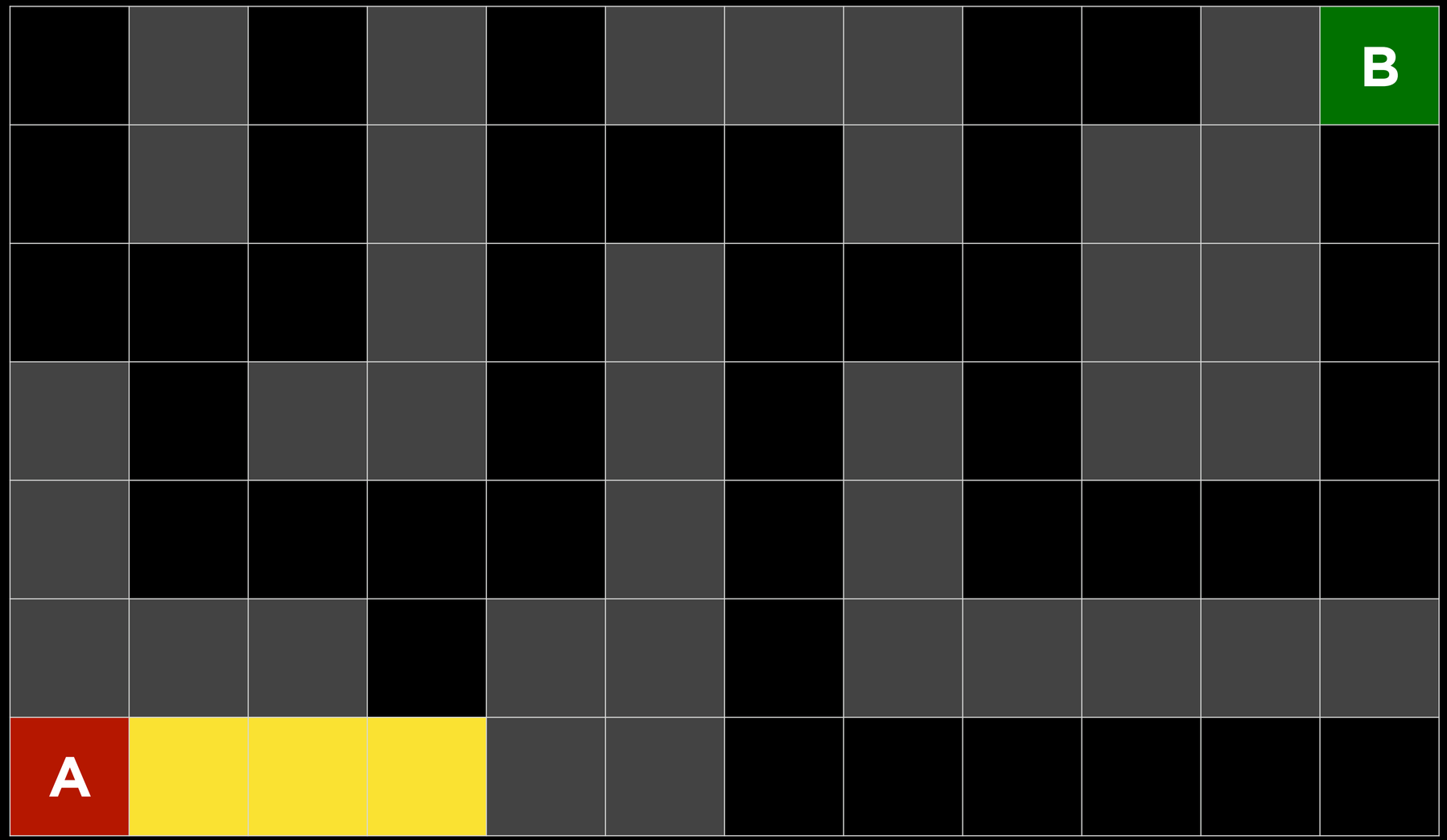
Depth-First Search



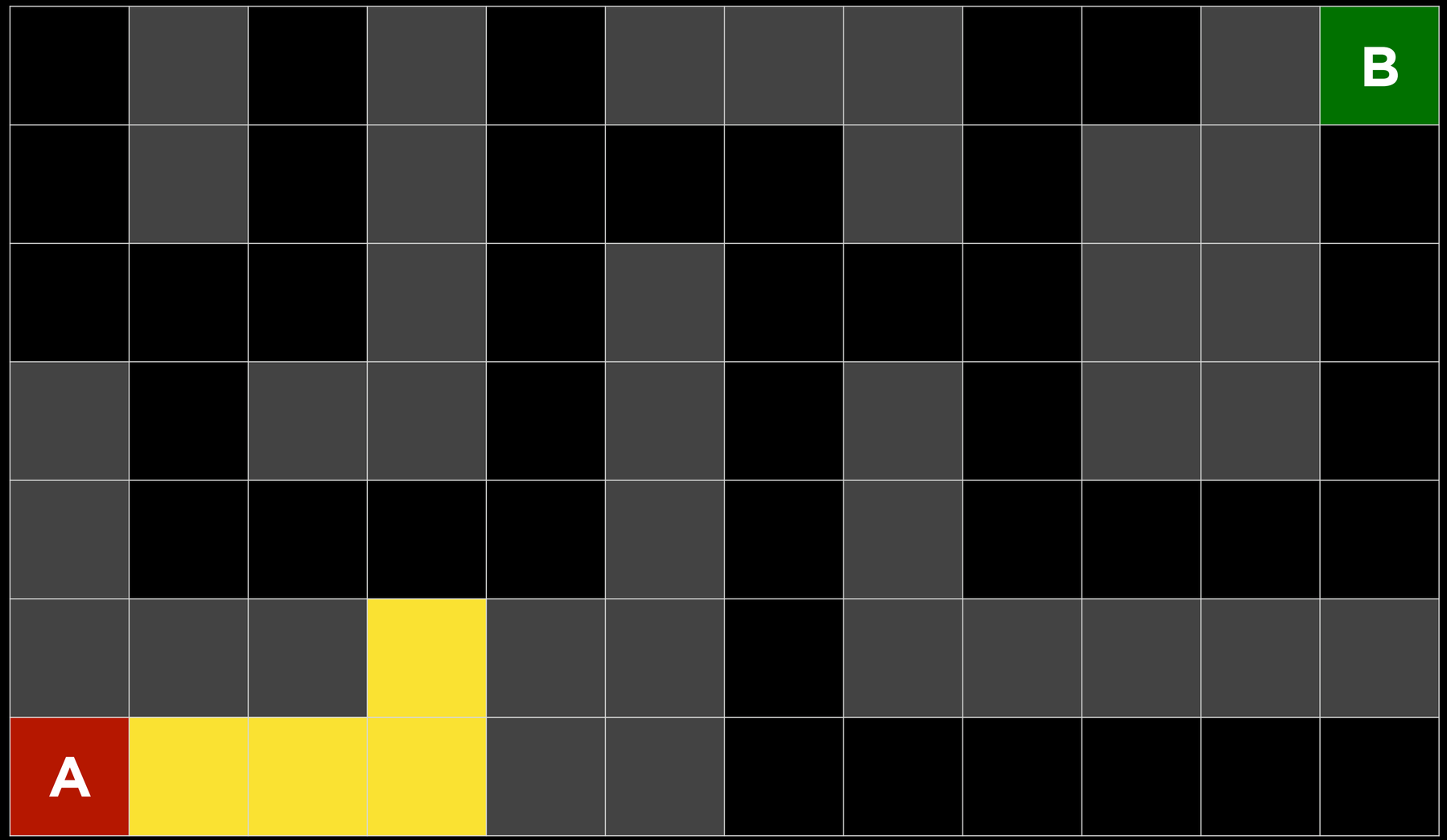
Depth-First Search



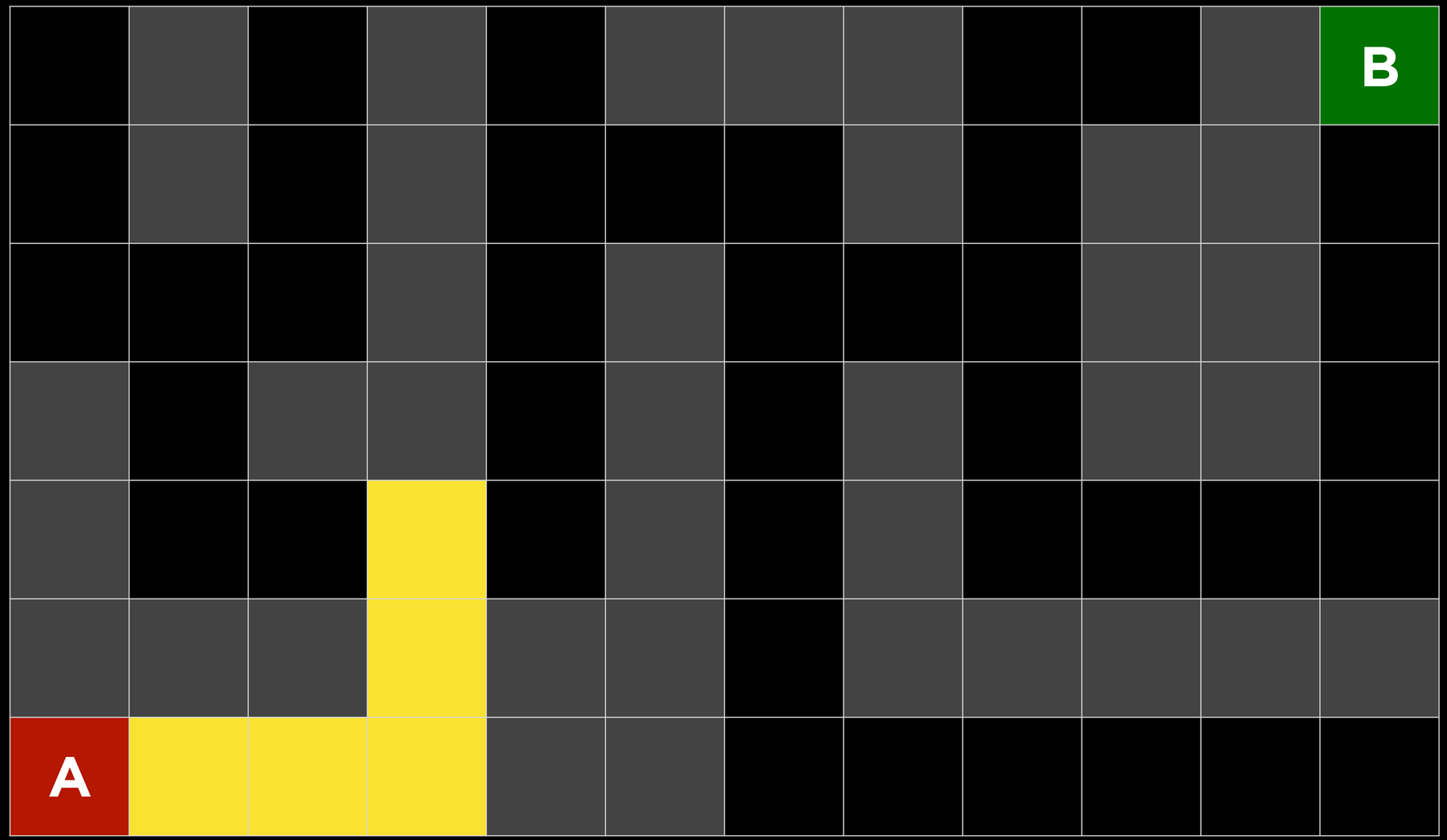
Depth-First Search



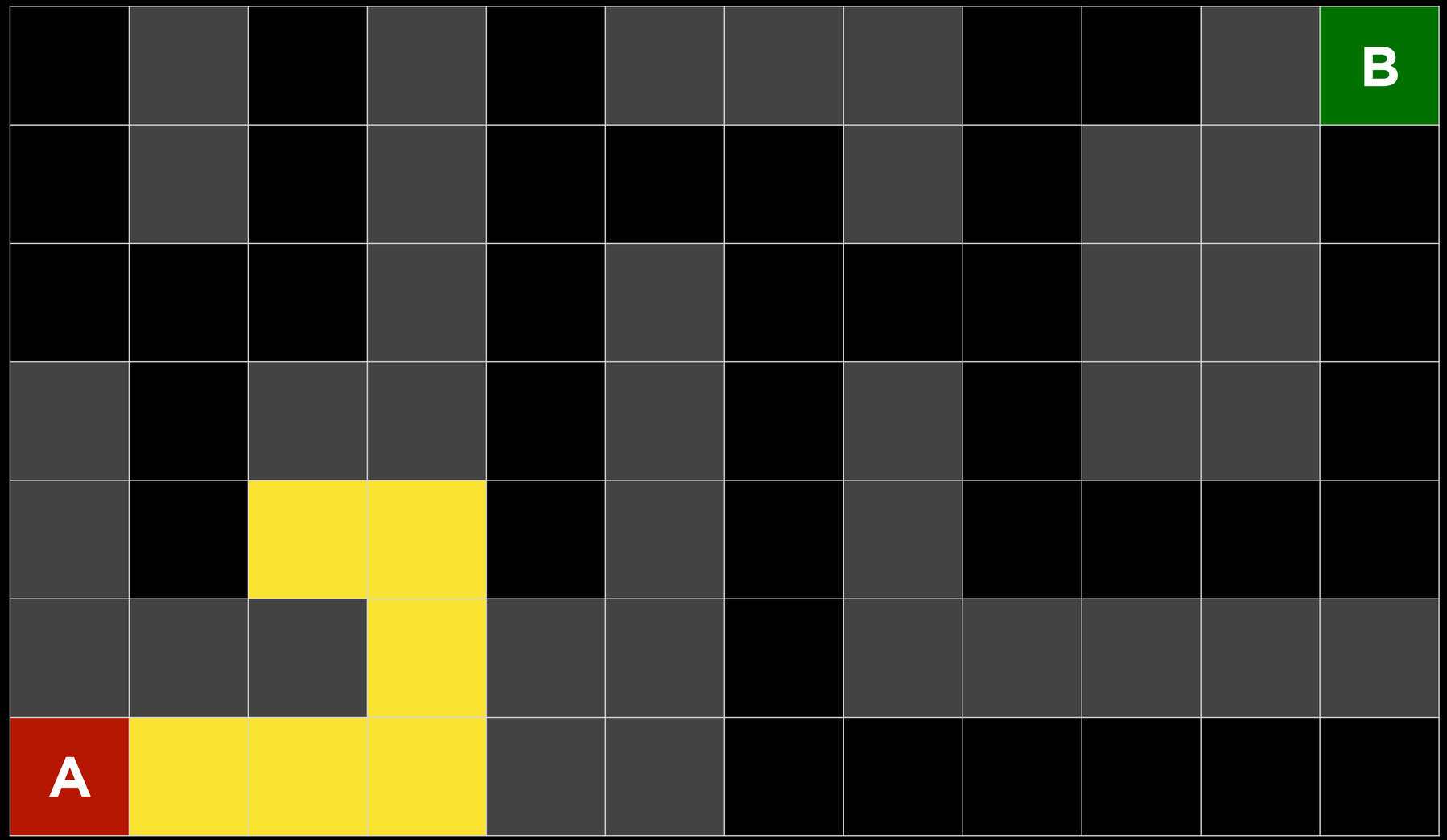
Depth-First Search



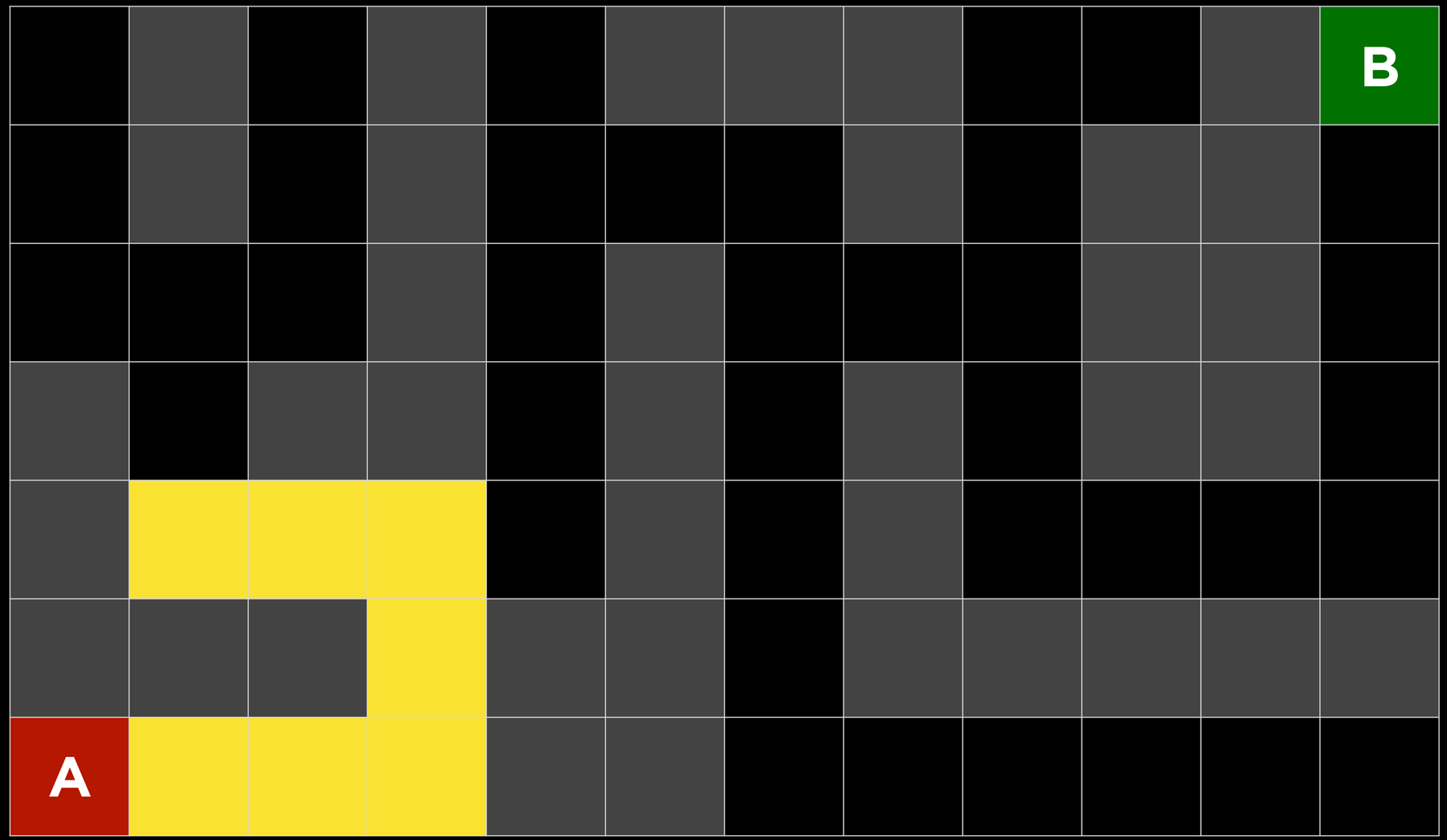
Depth-First Search



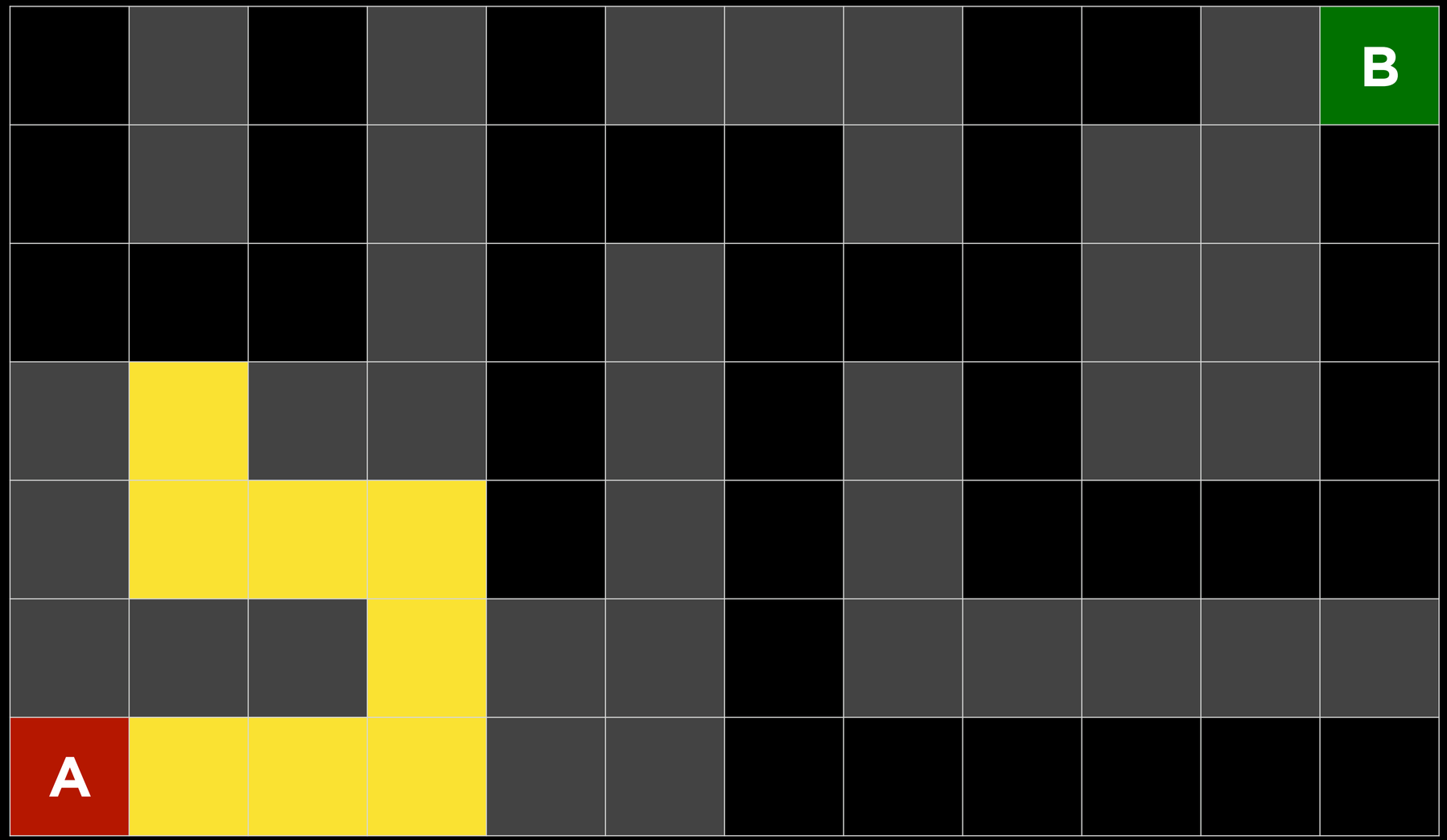
Depth-First Search



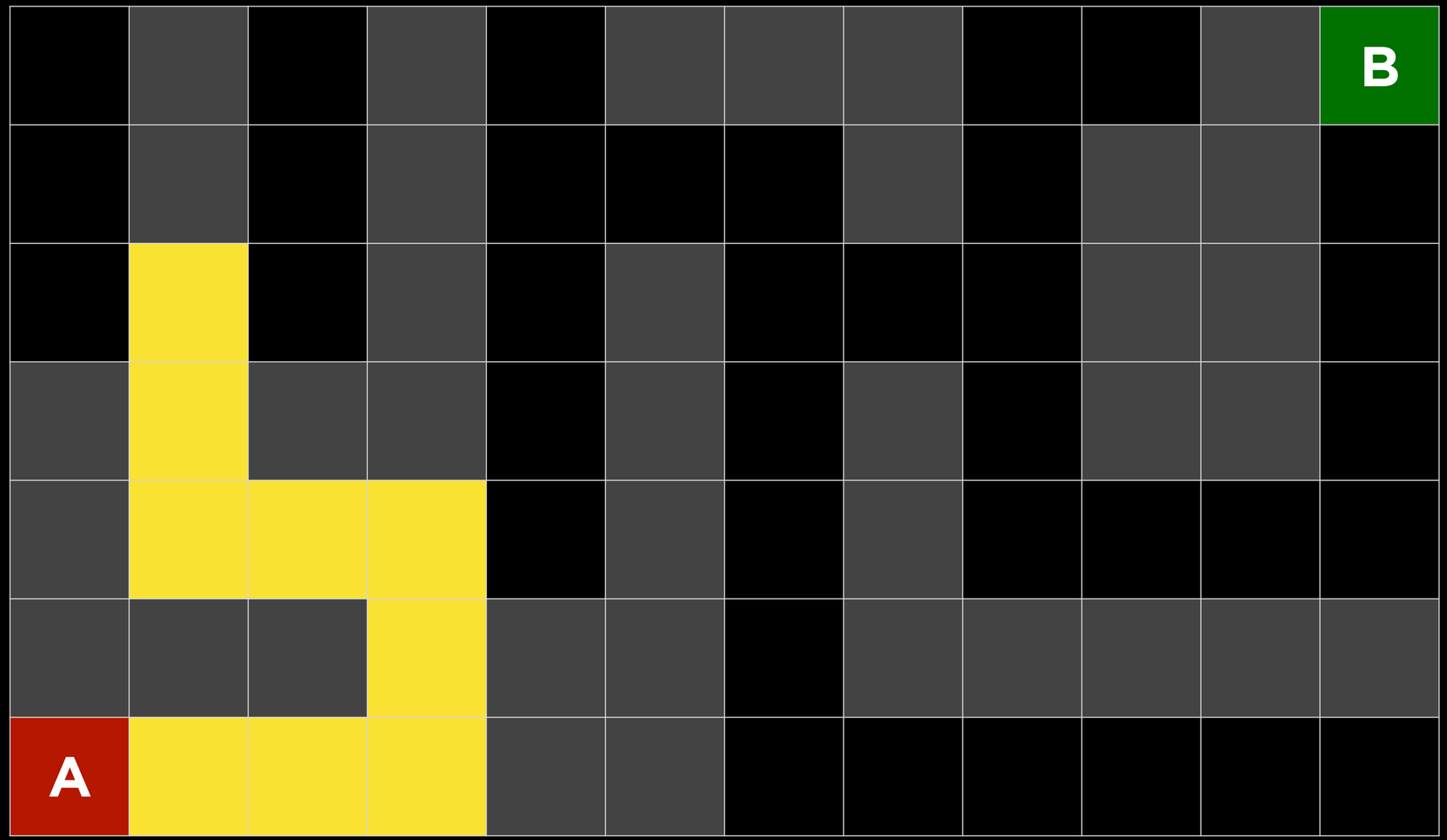
Depth-First Search



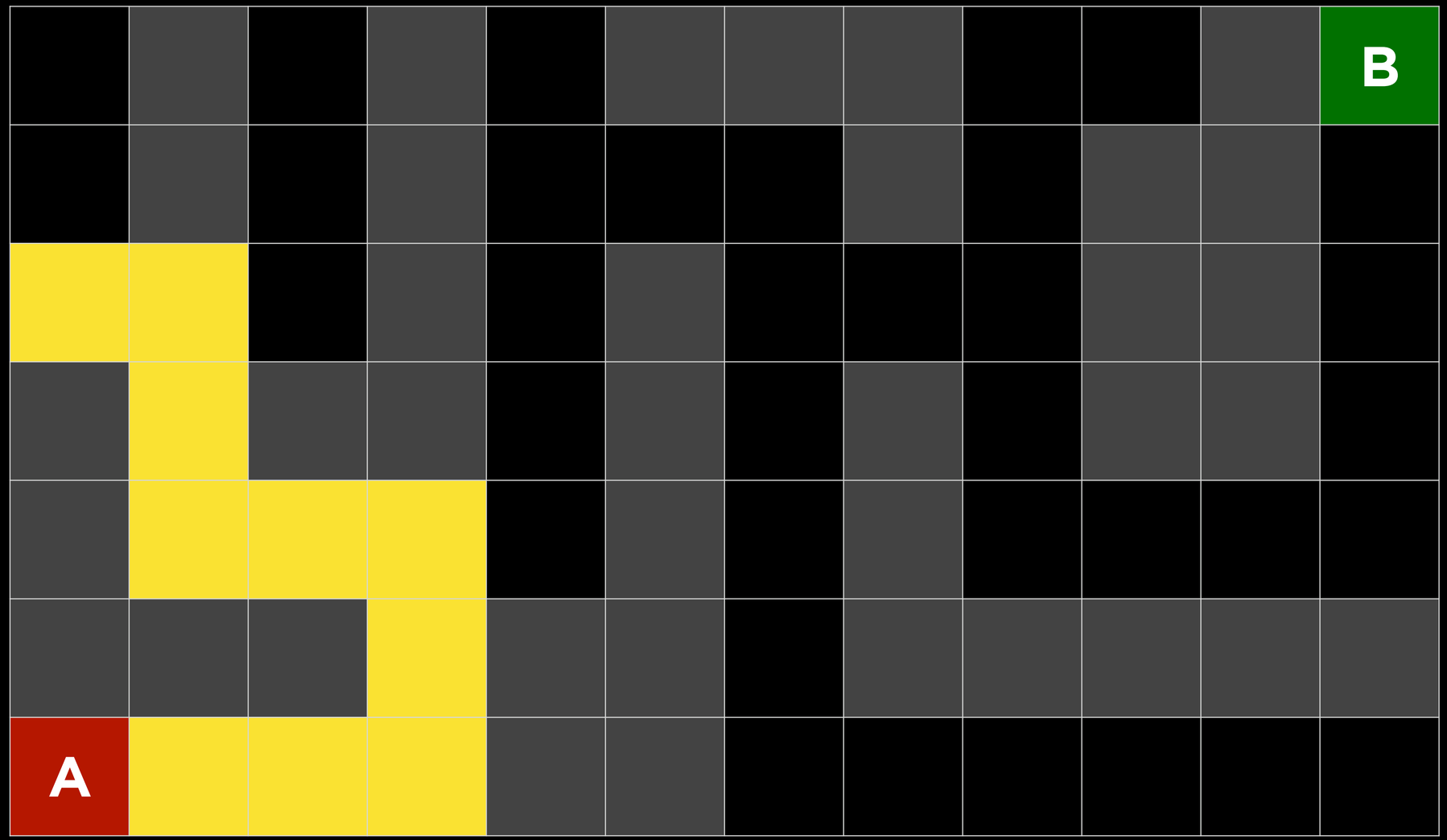
Depth-First Search



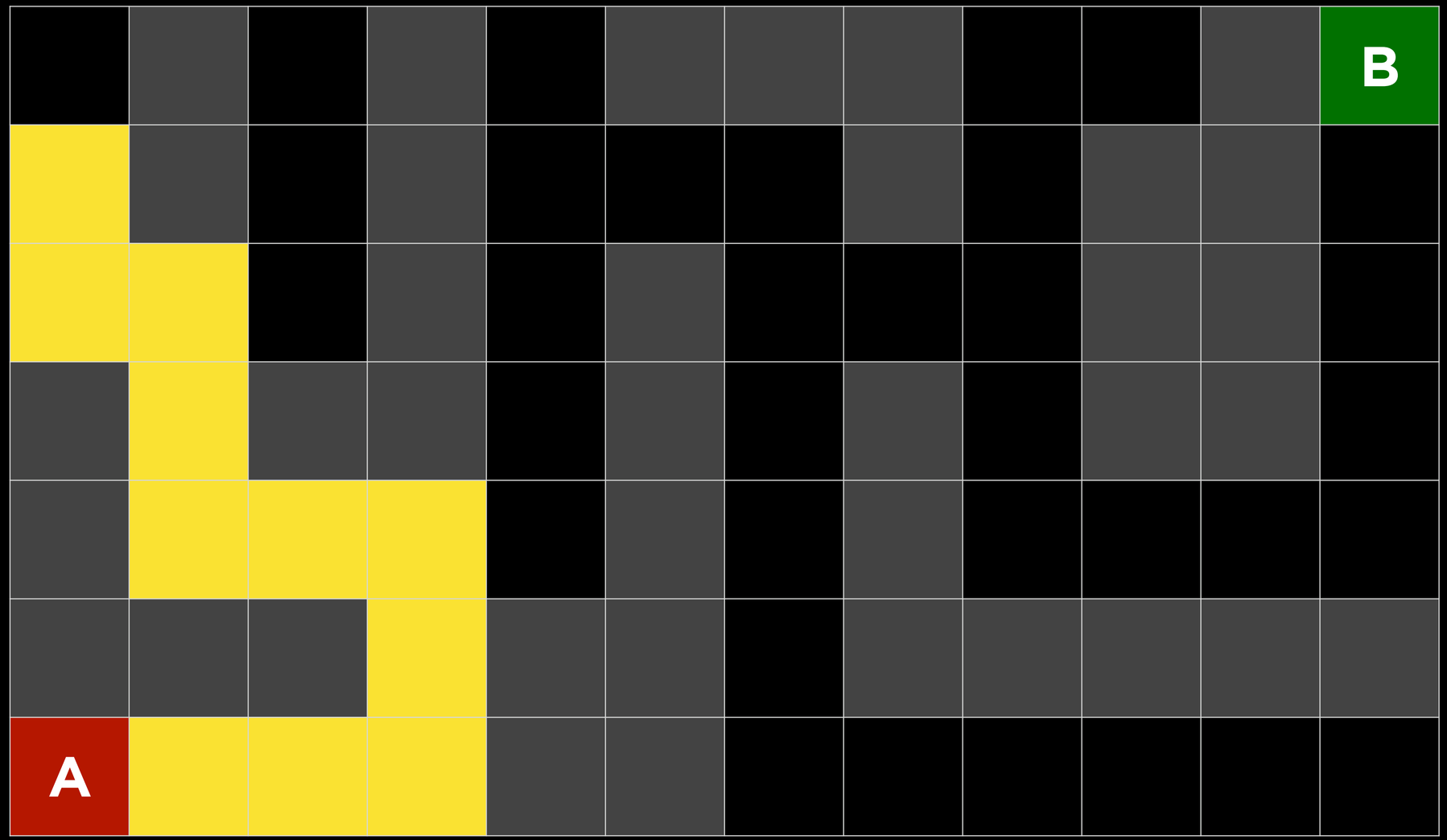
Depth-First Search



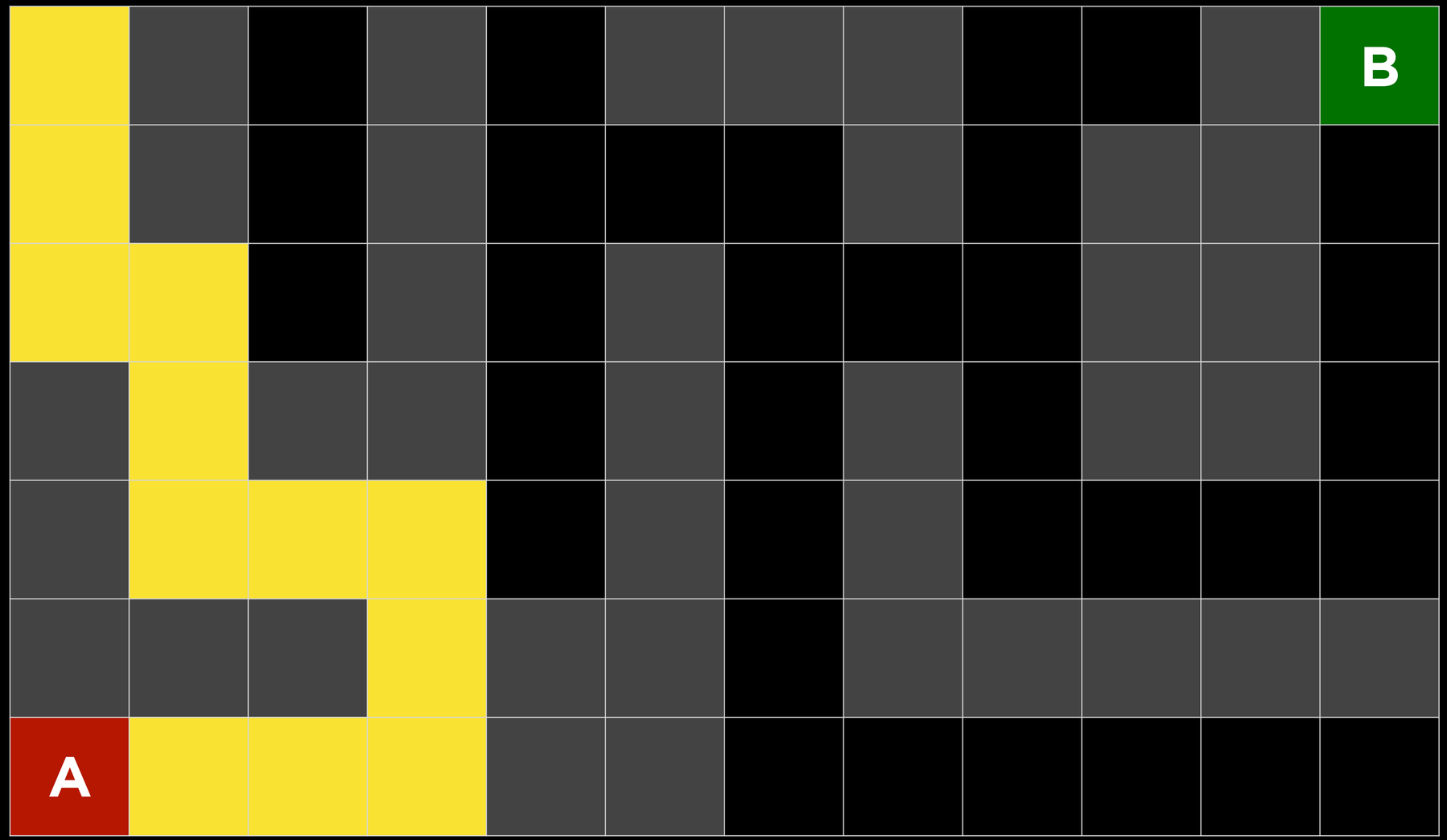
Depth-First Search



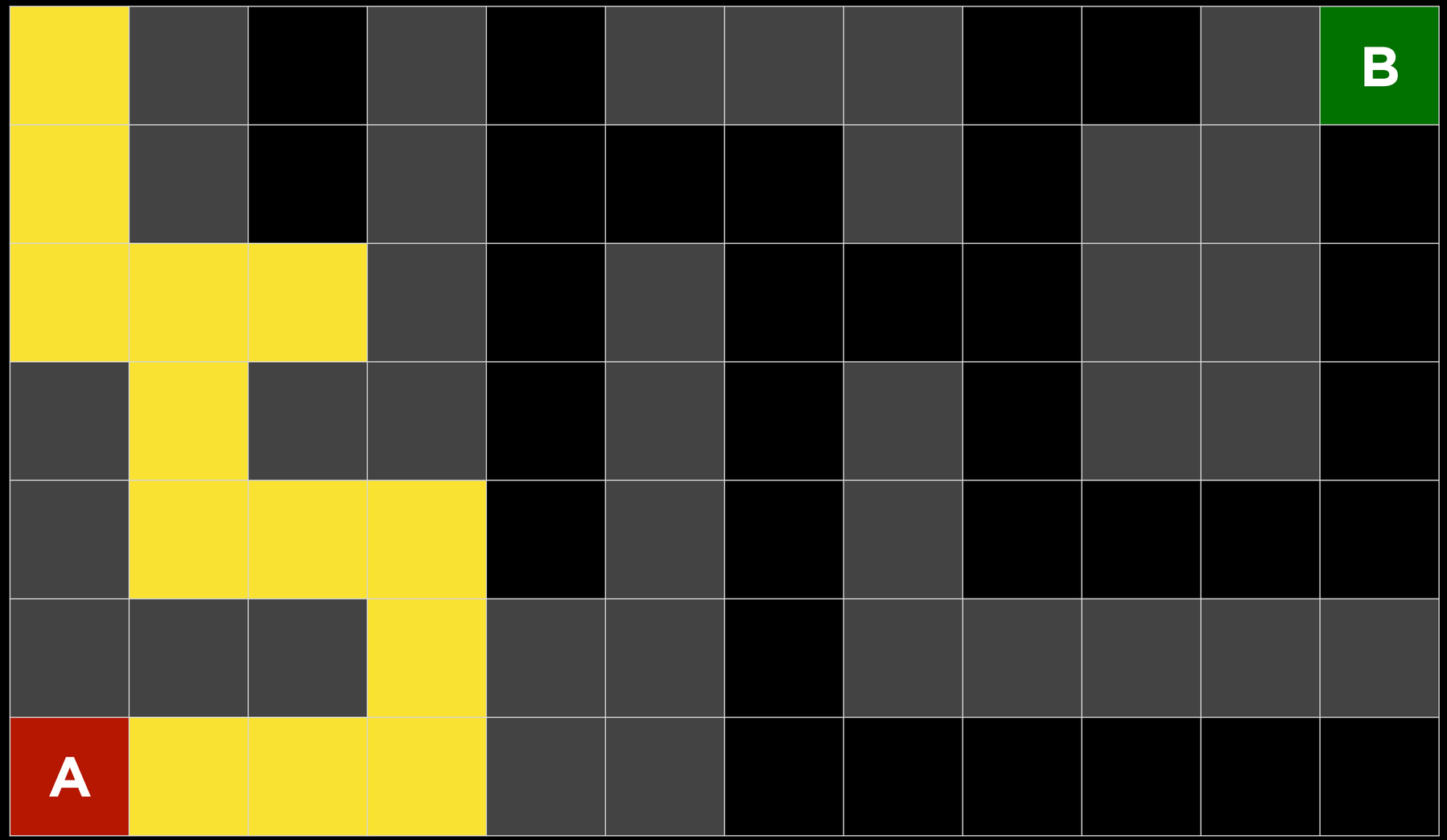
Depth-First Search



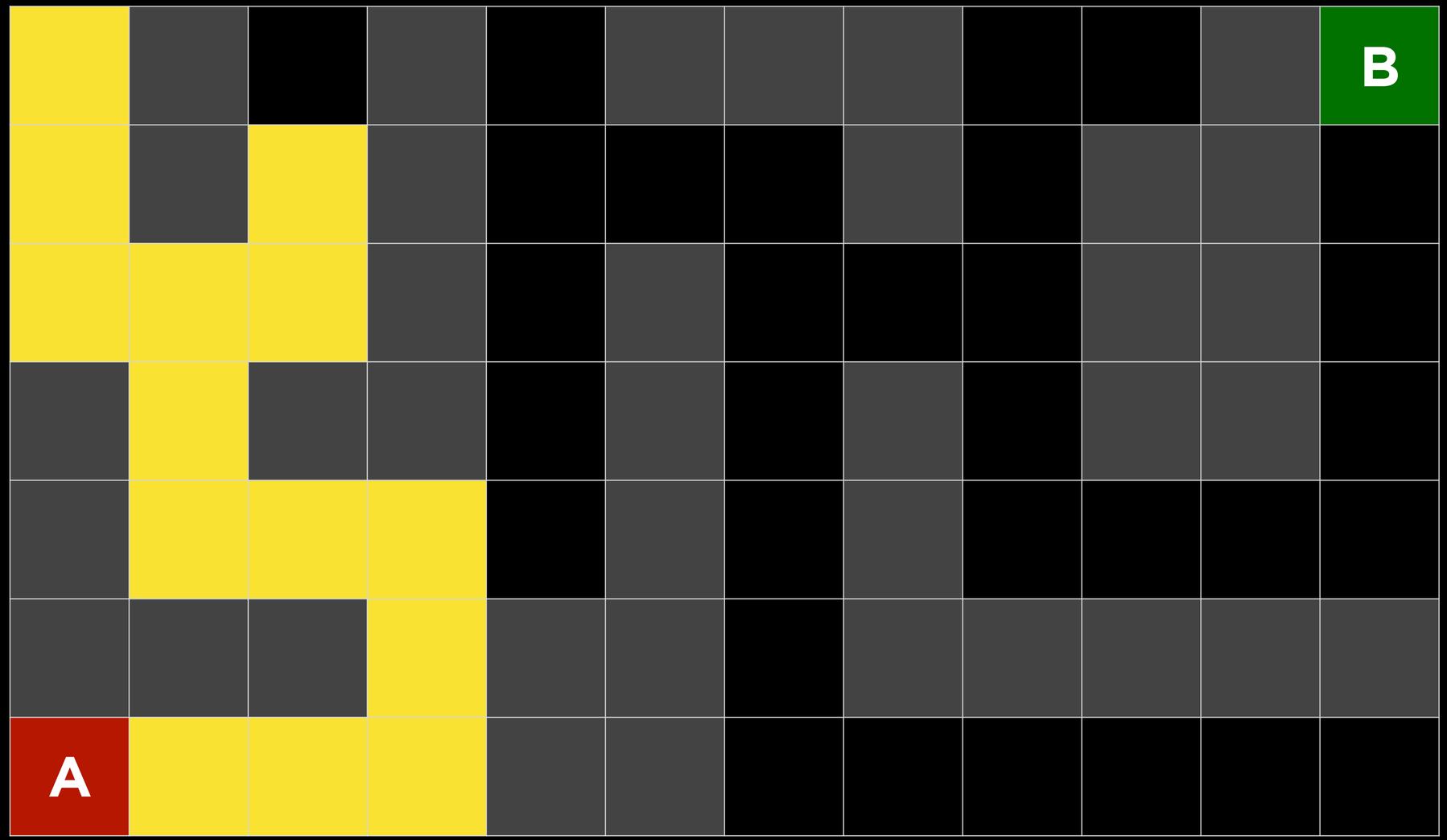
Depth-First Search



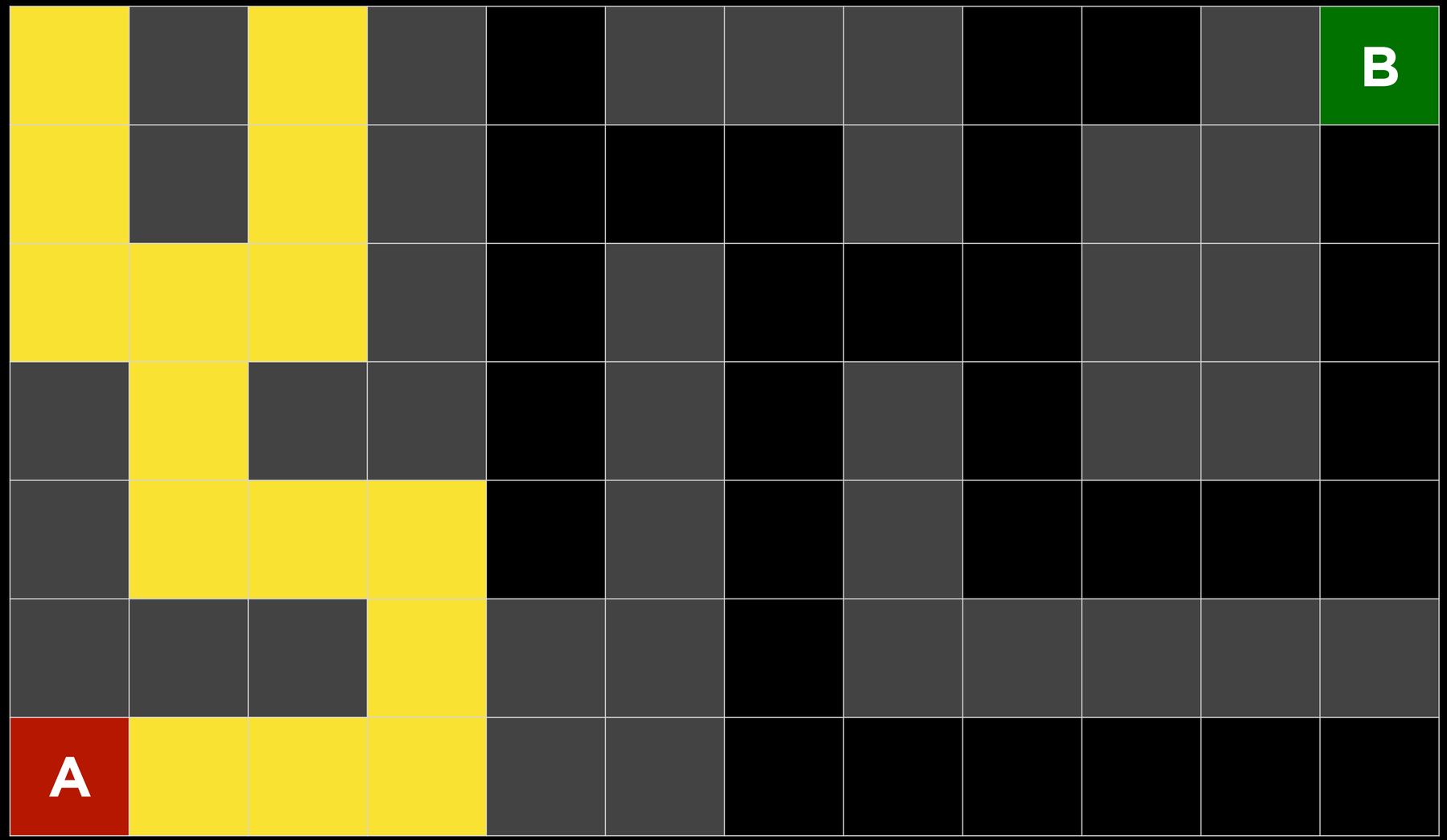
Depth-First Search



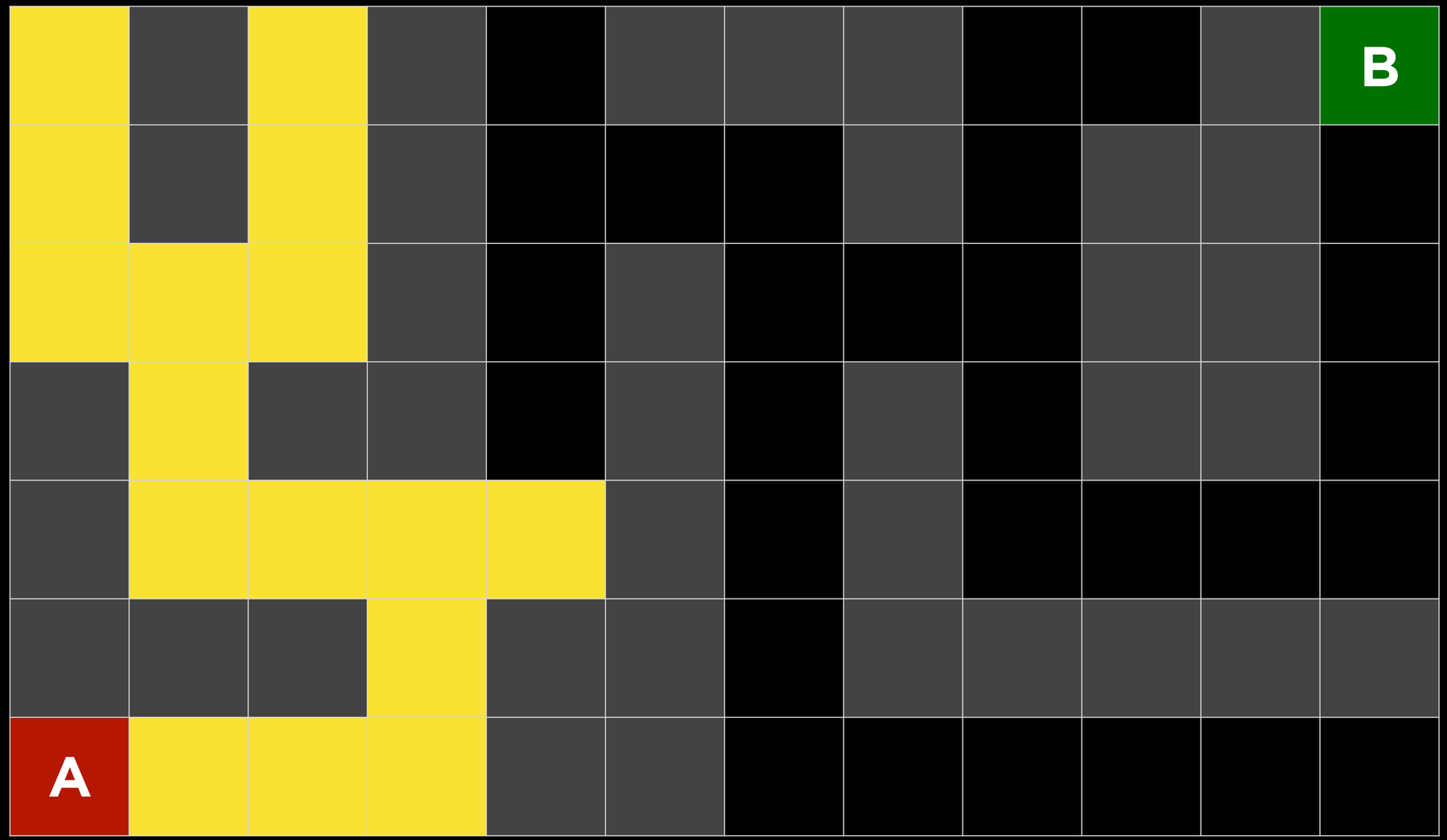
Depth-First Search



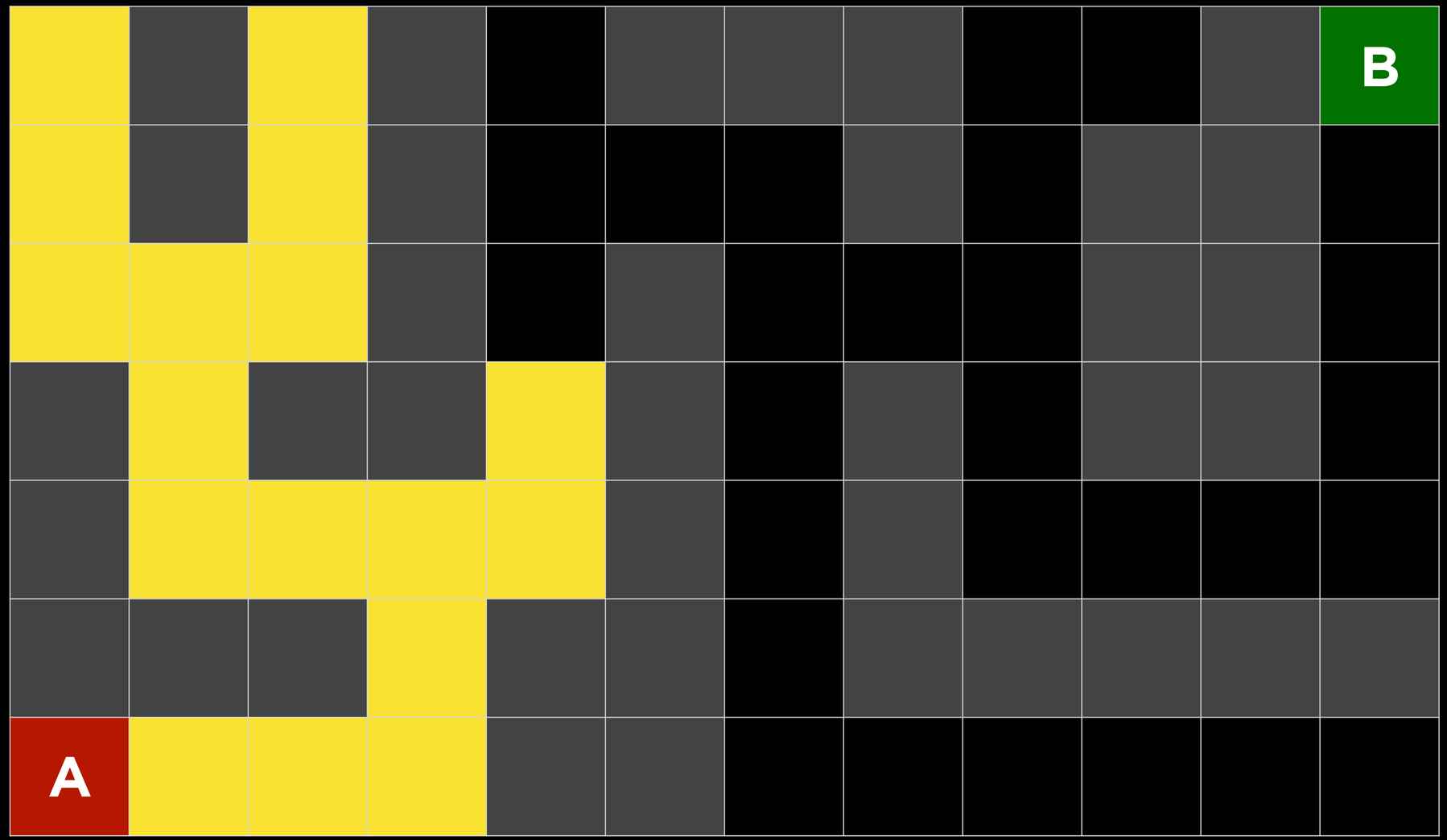
Depth-First Search



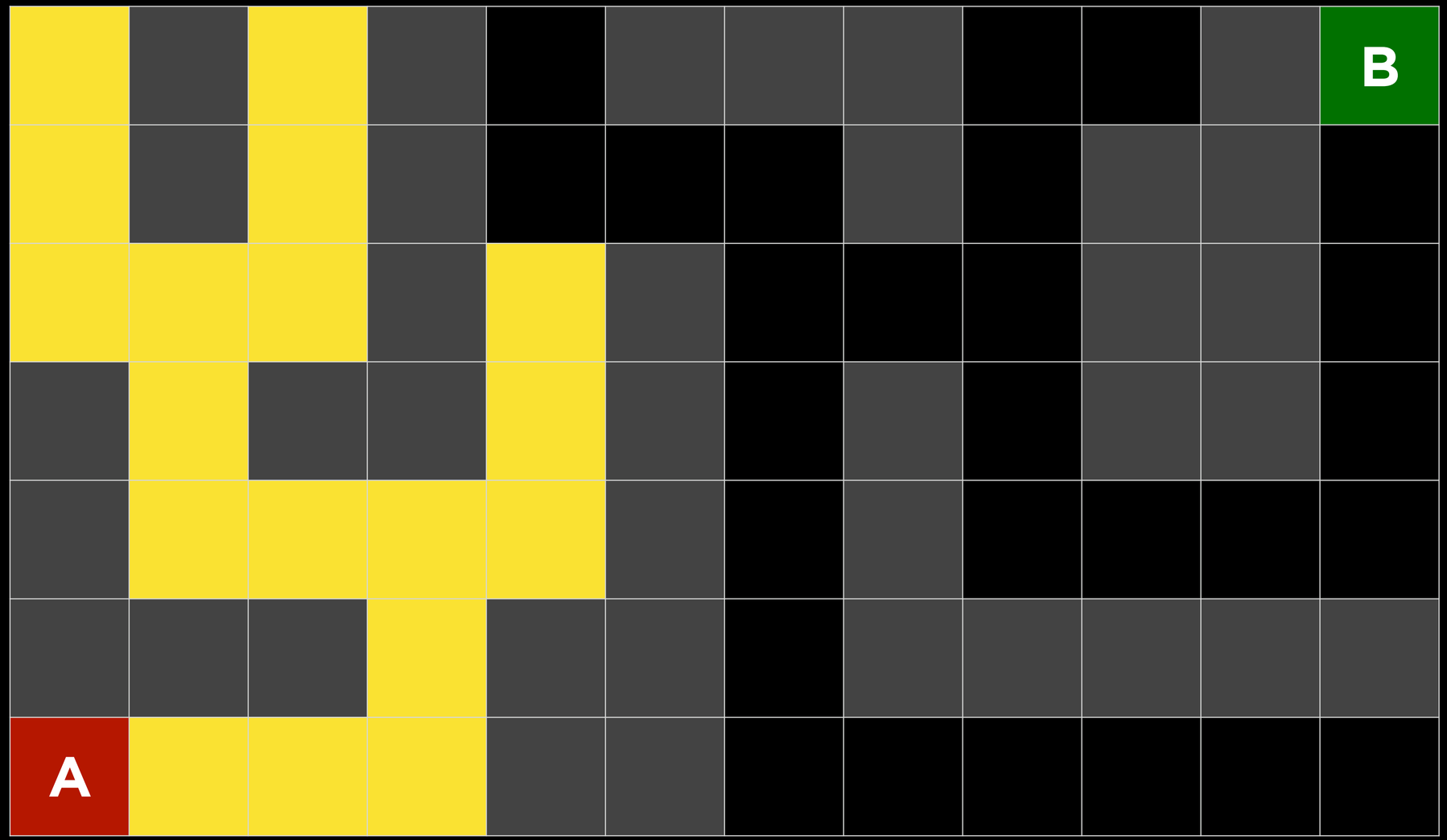
Depth-First Search



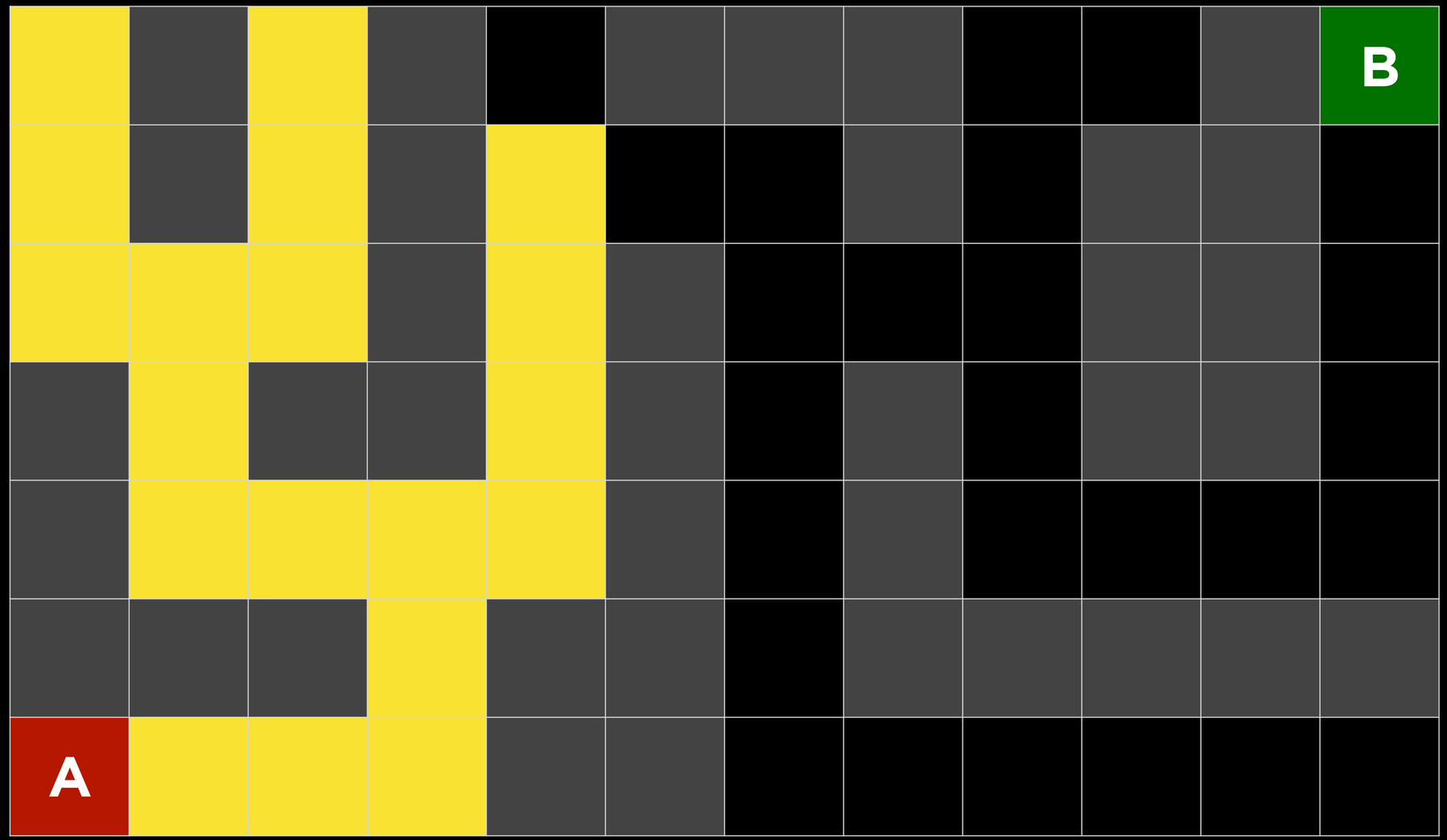
Depth-First Search



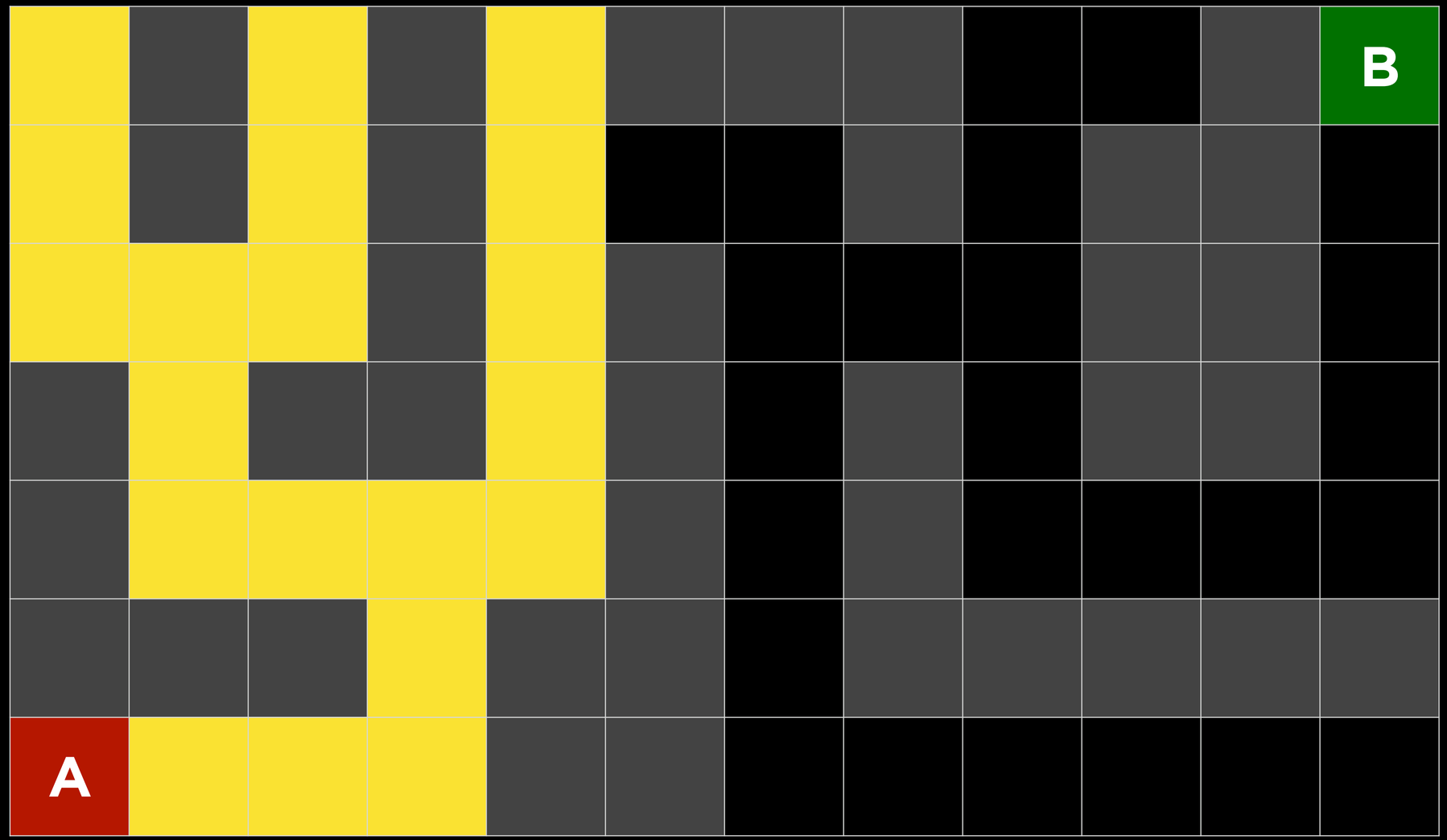
Depth-First Search



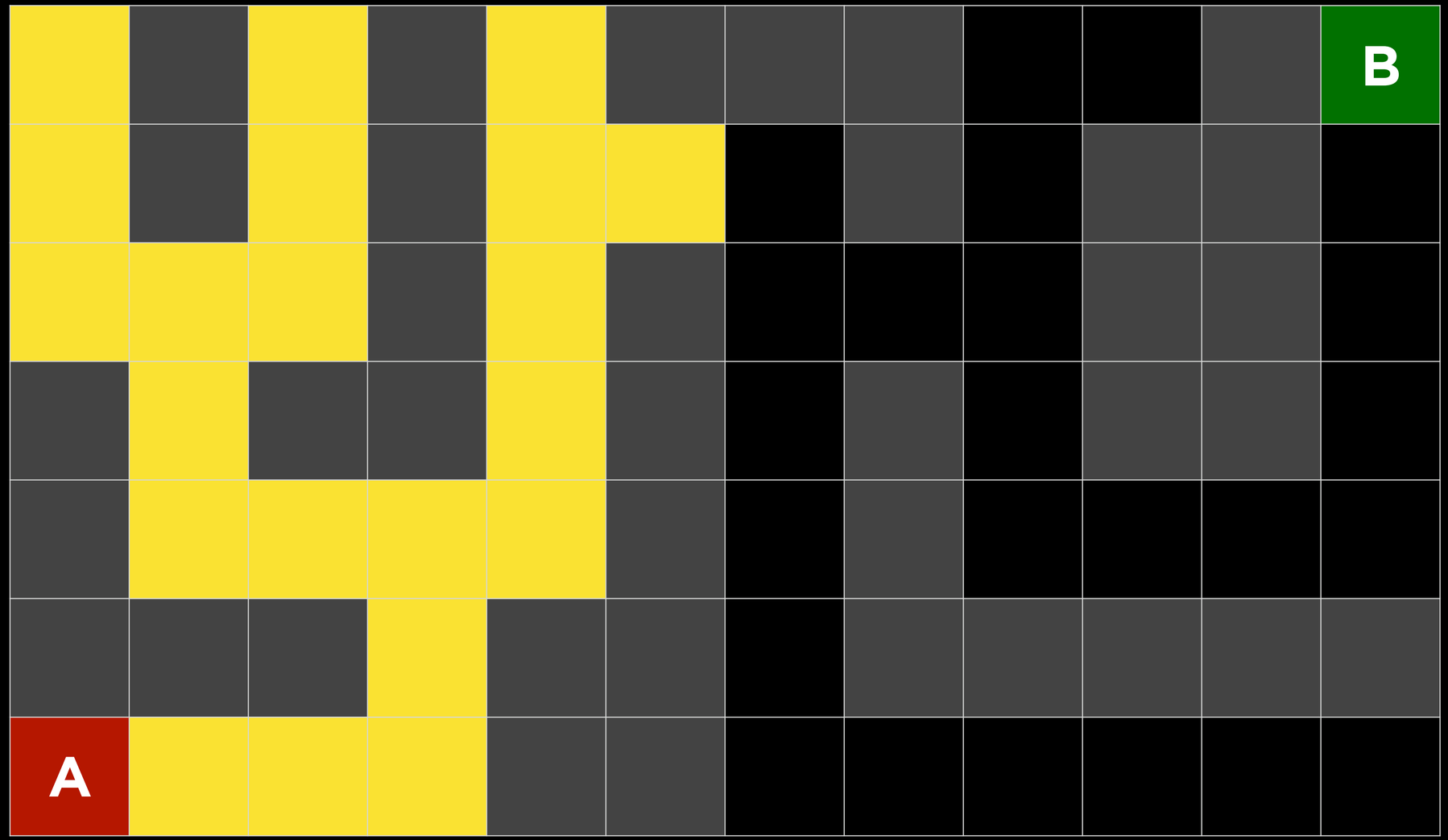
Depth-First Search



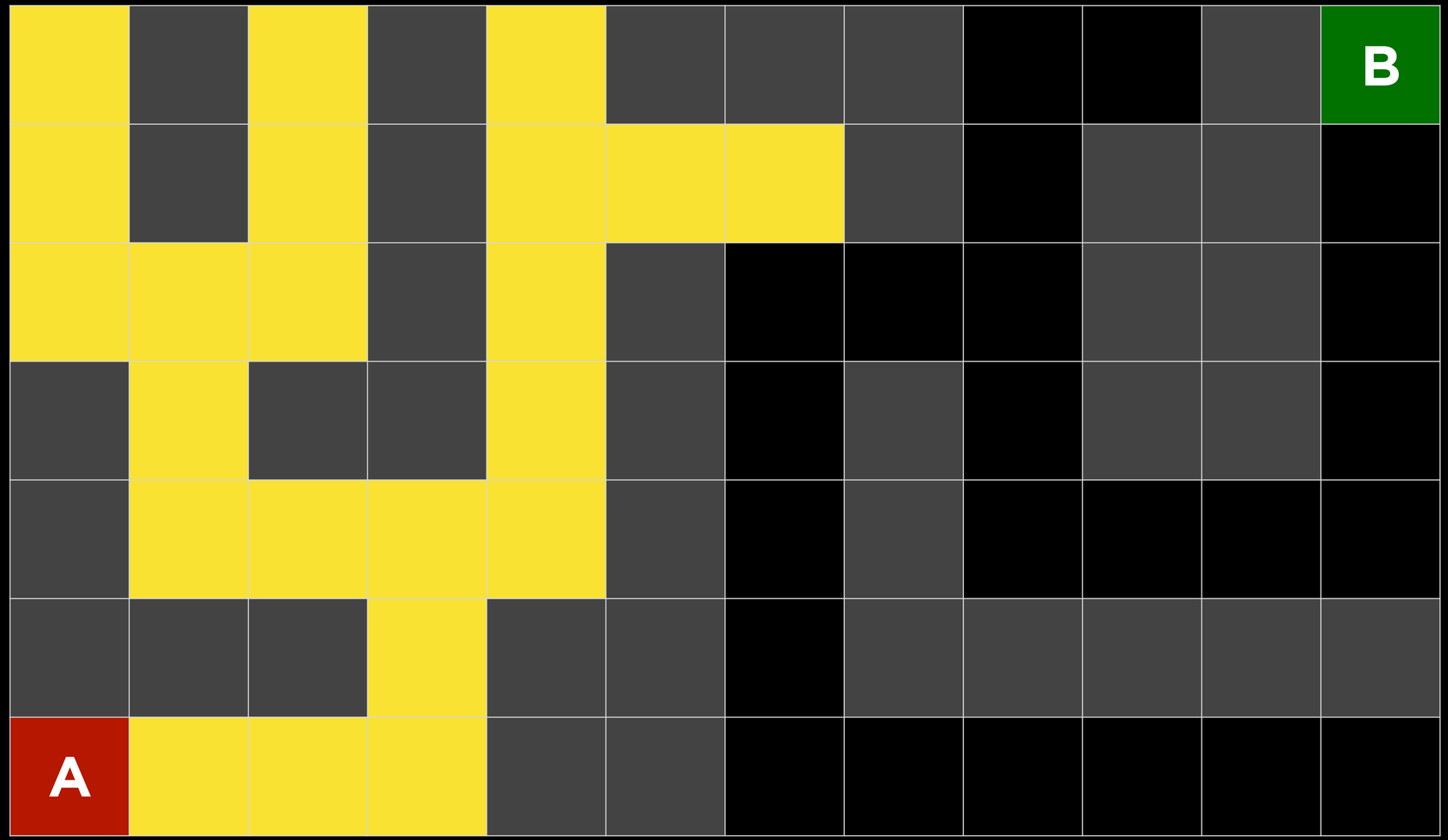
Depth-First Search



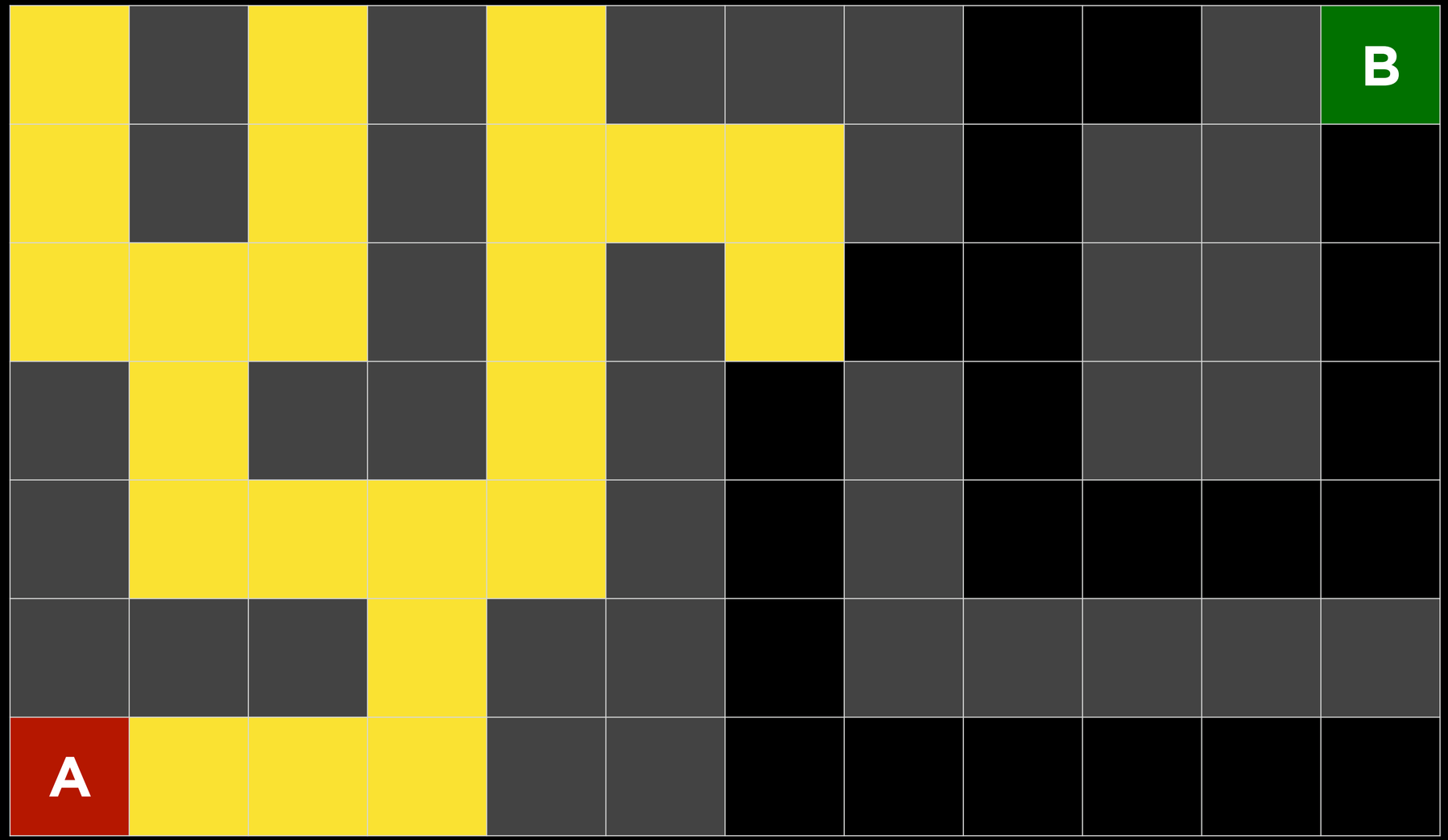
Depth-First Search



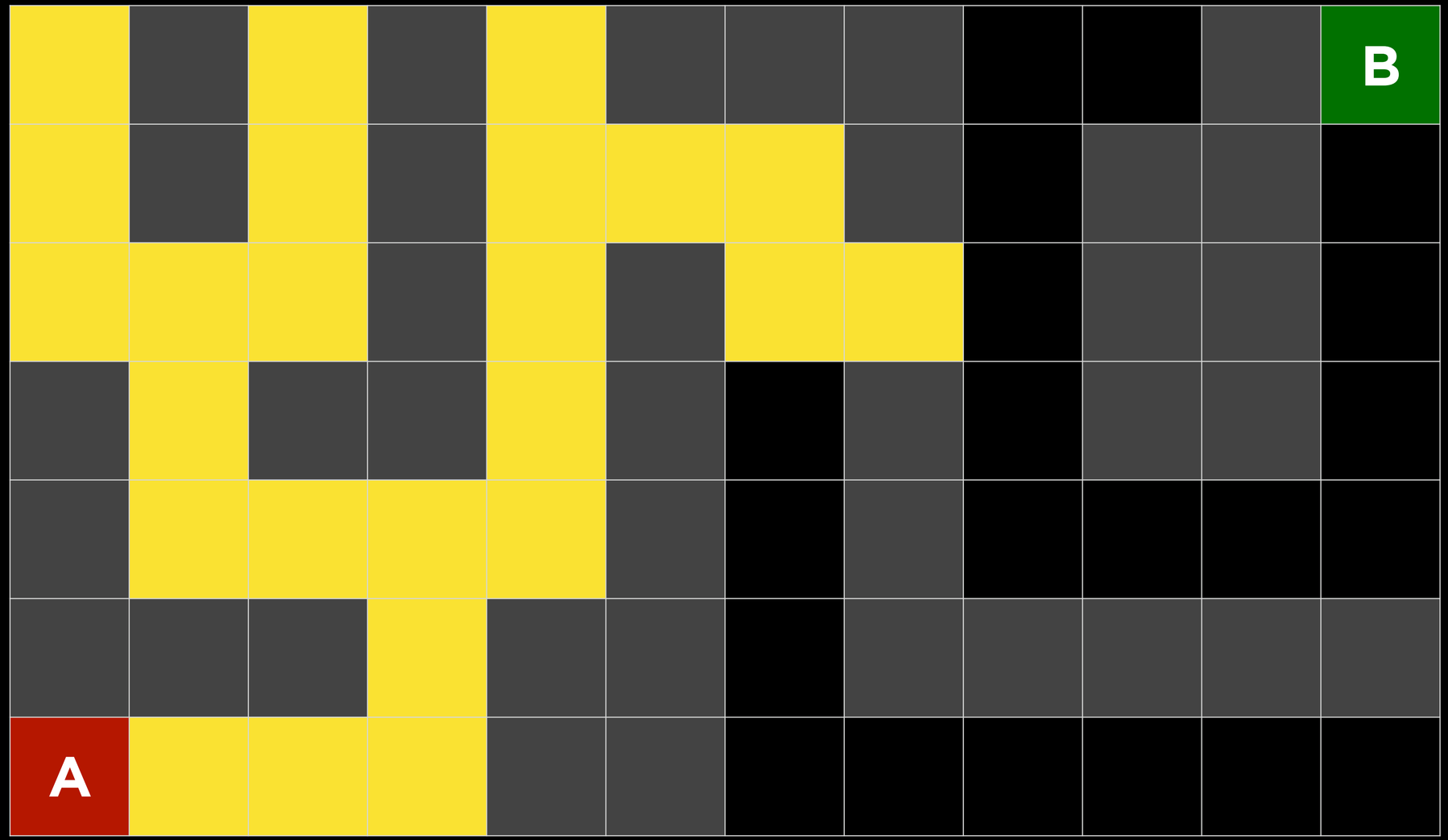
Depth-First Search



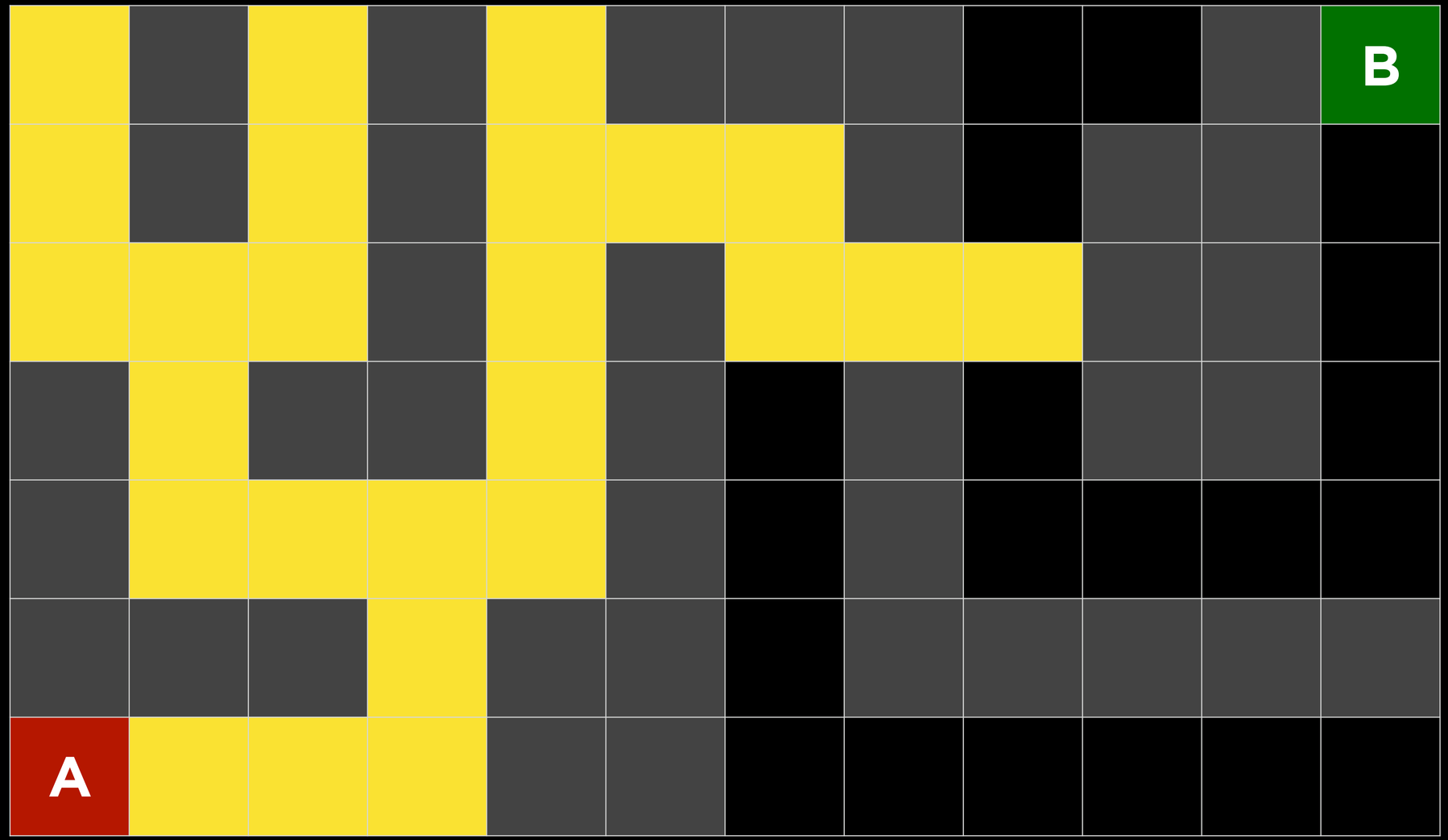
Depth-First Search



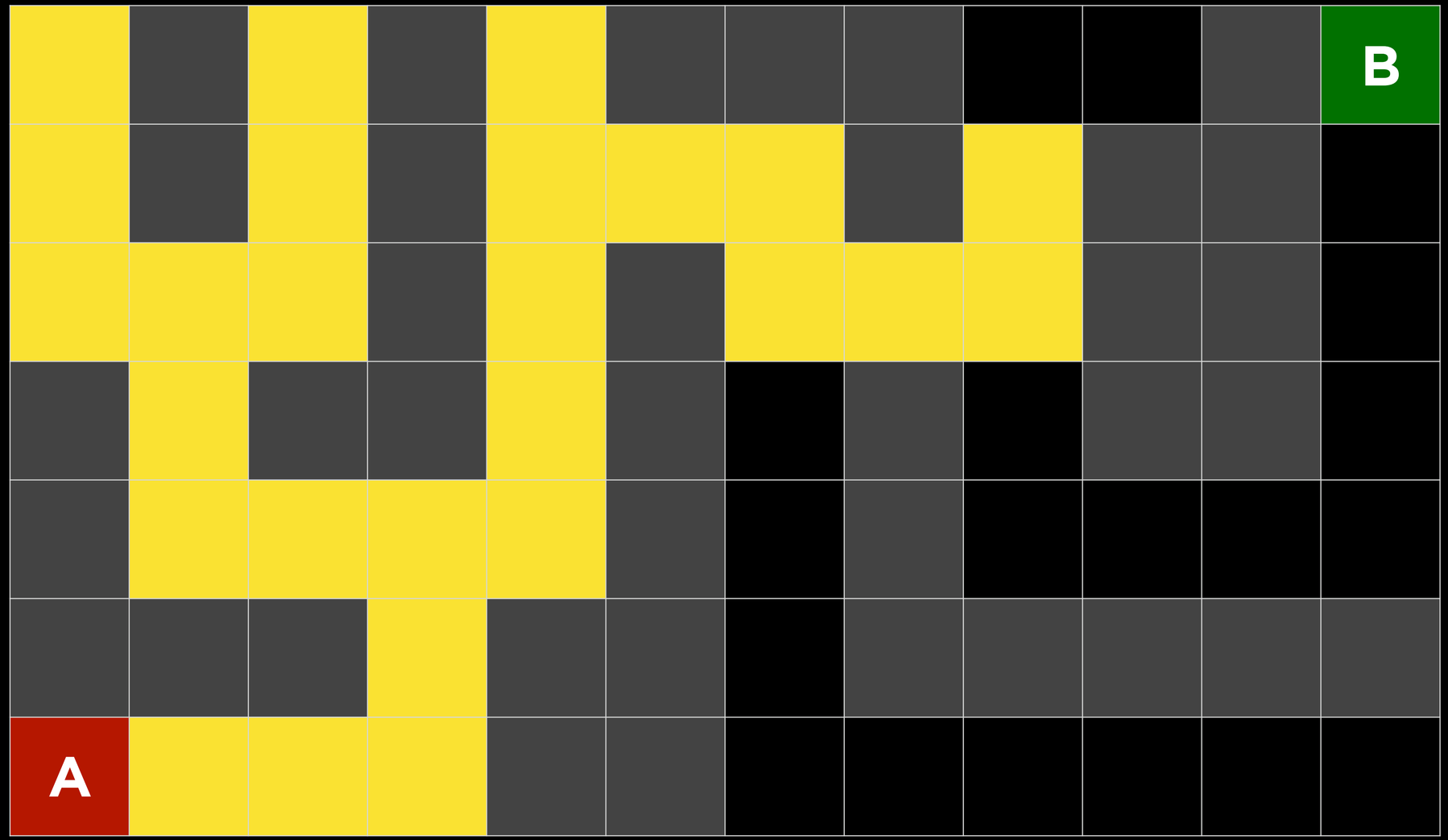
Depth-First Search



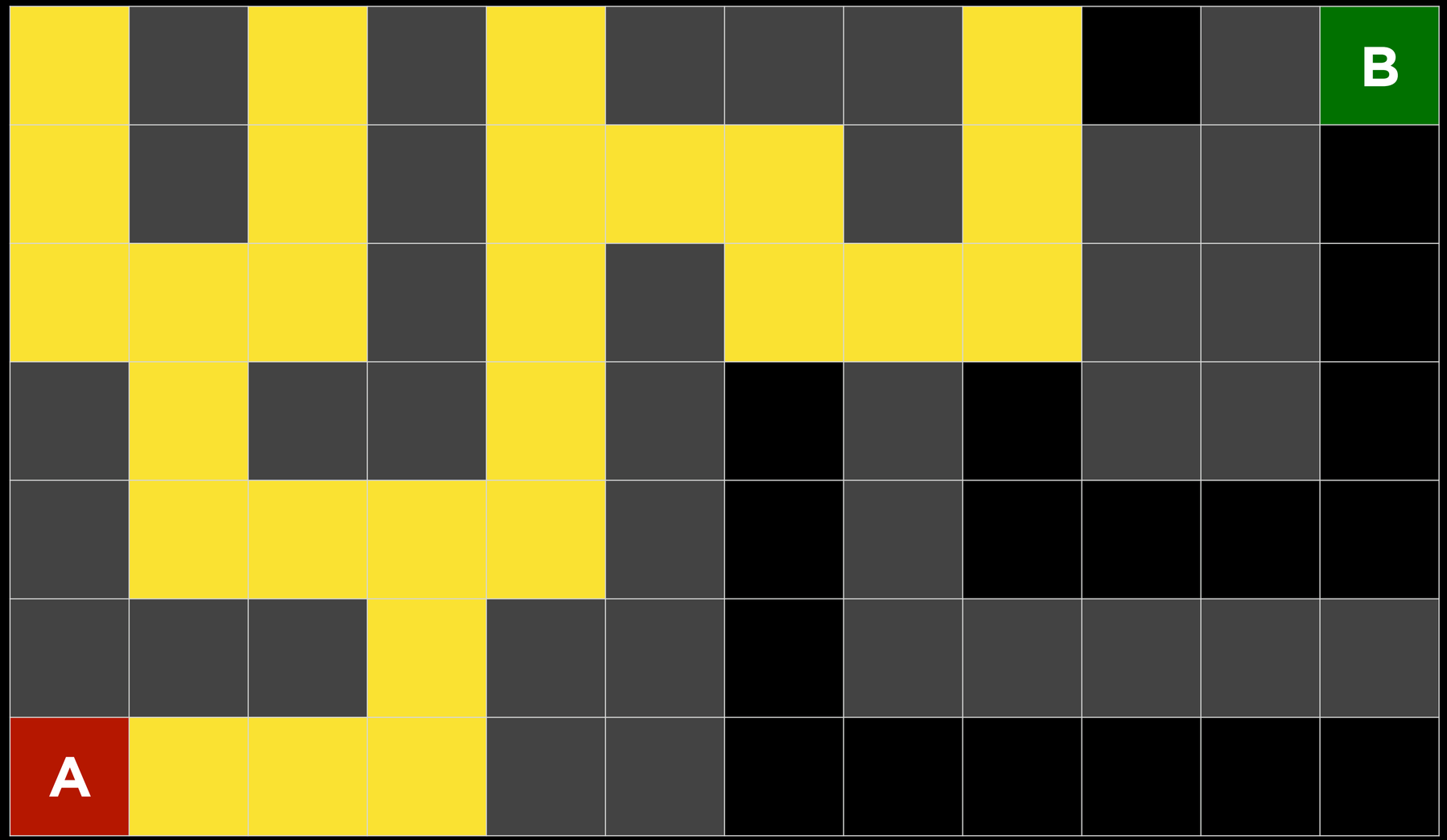
Depth-First Search



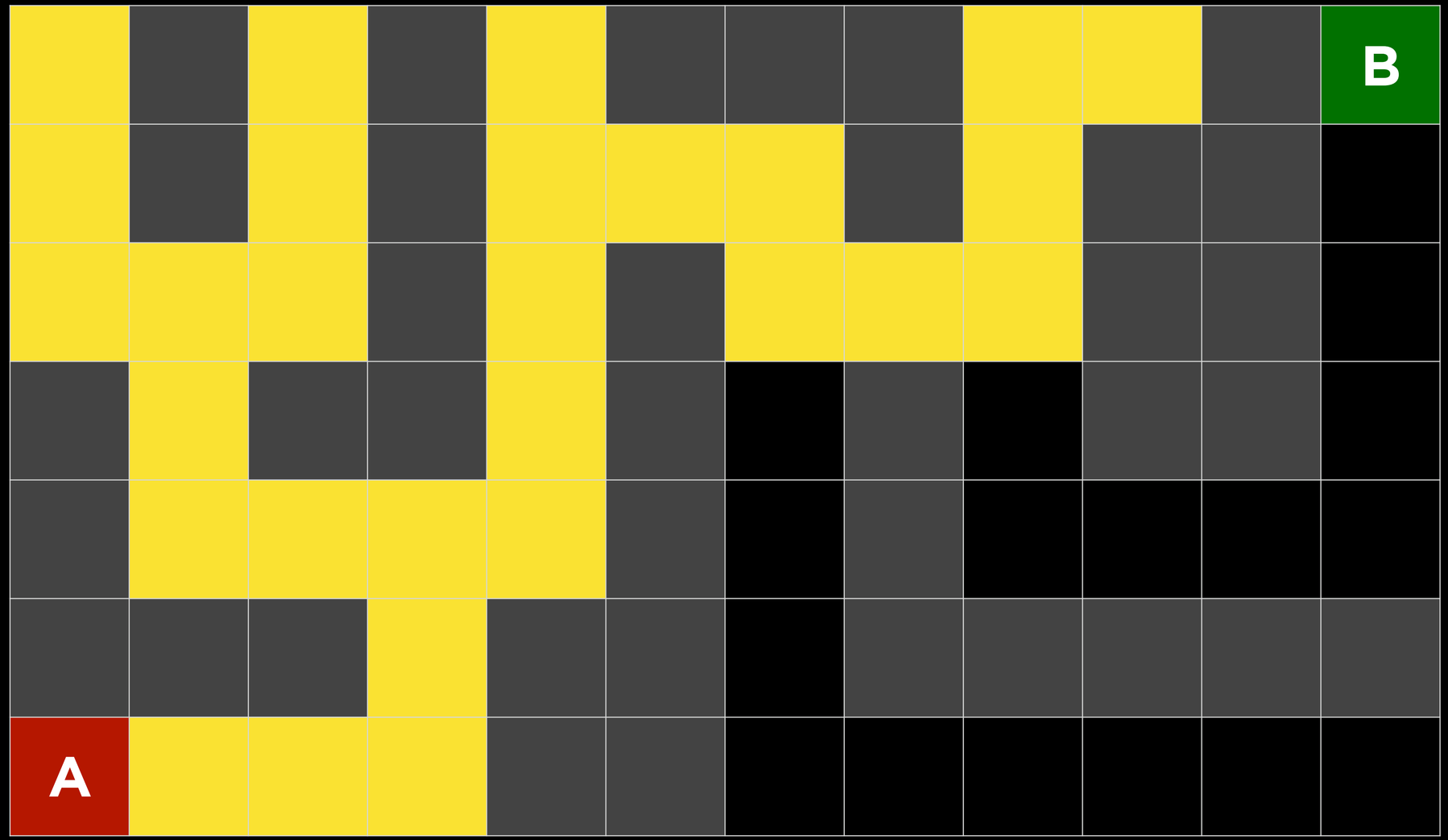
Depth-First Search



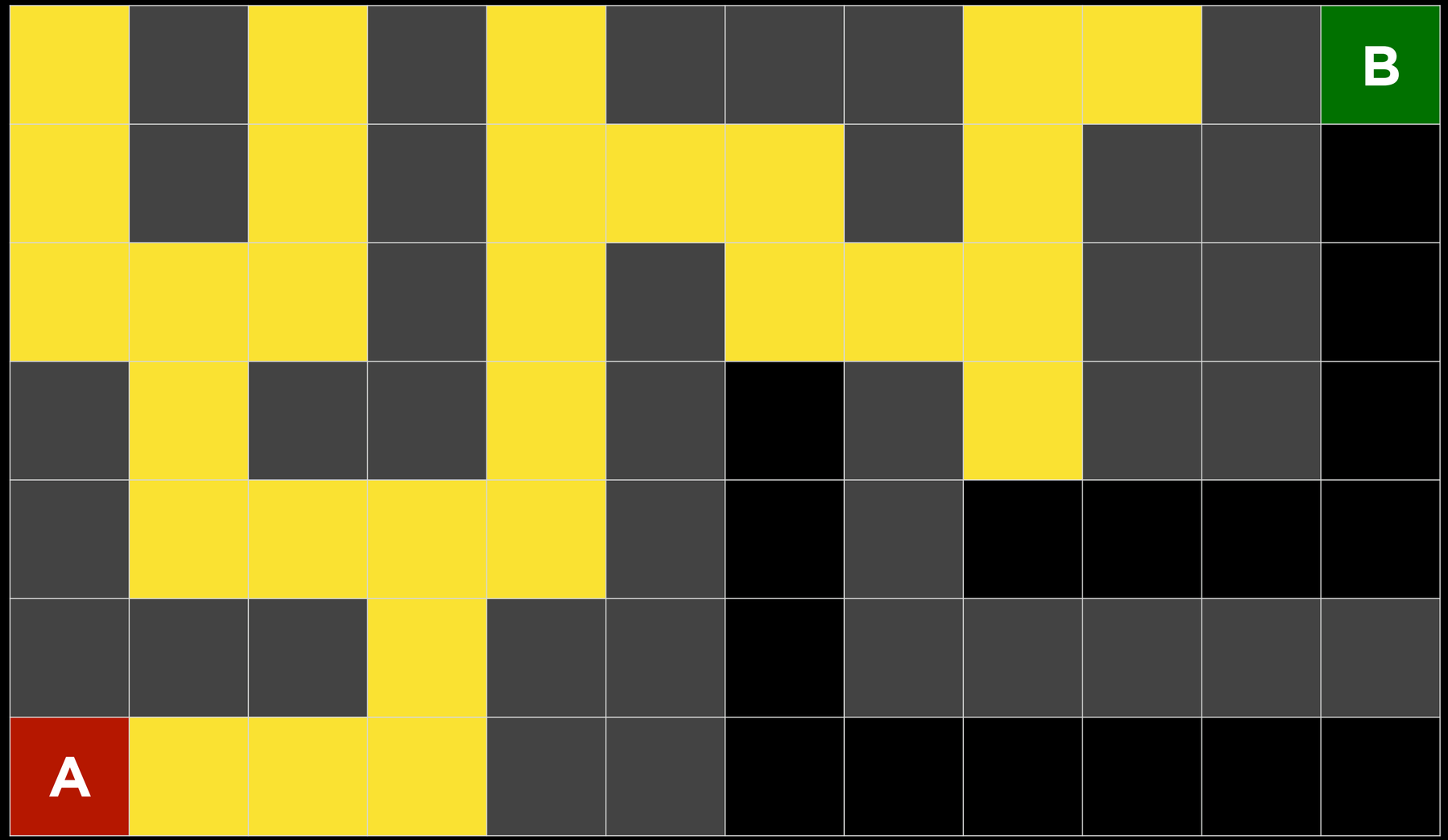
Depth-First Search



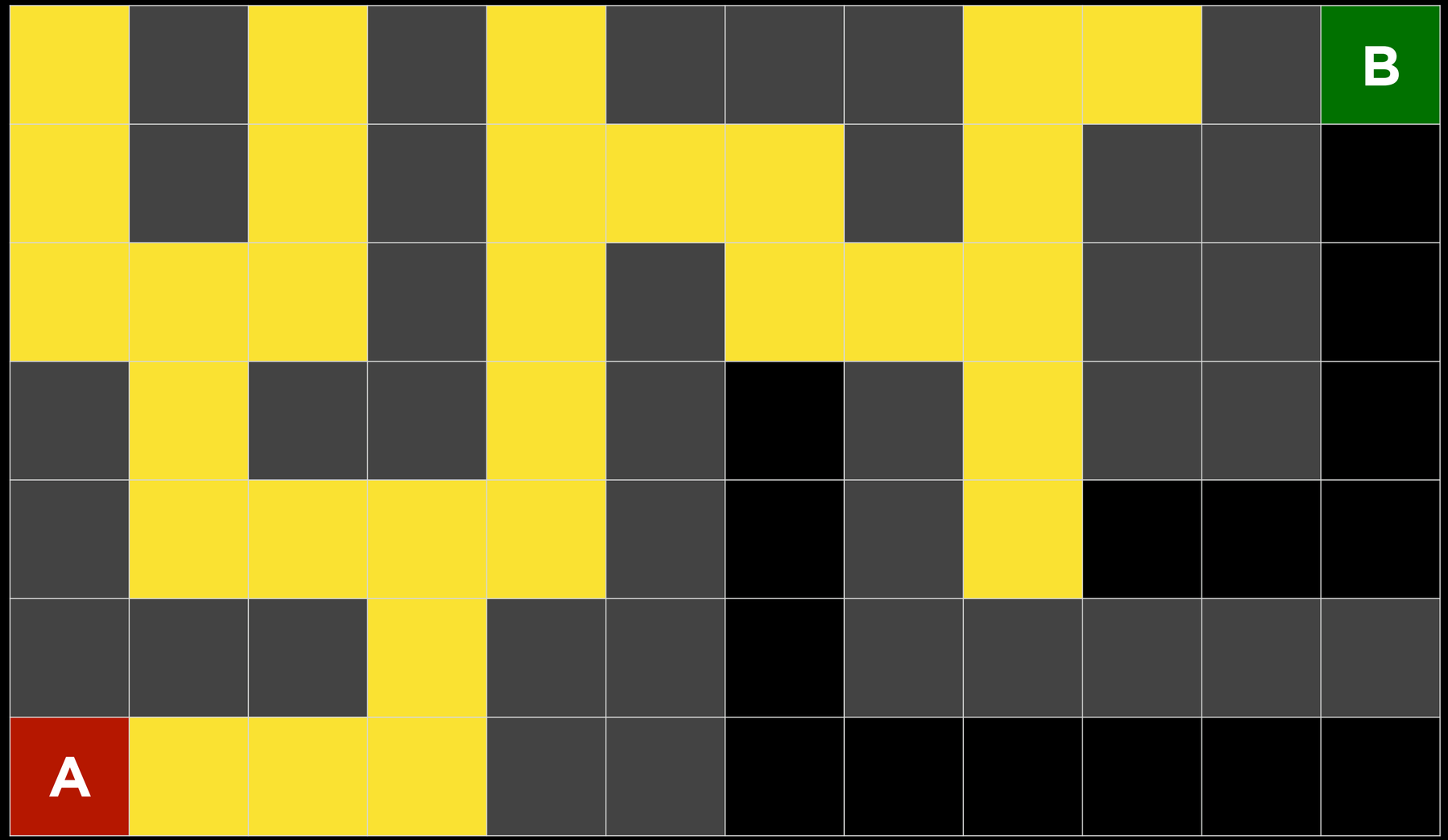
Depth-First Search



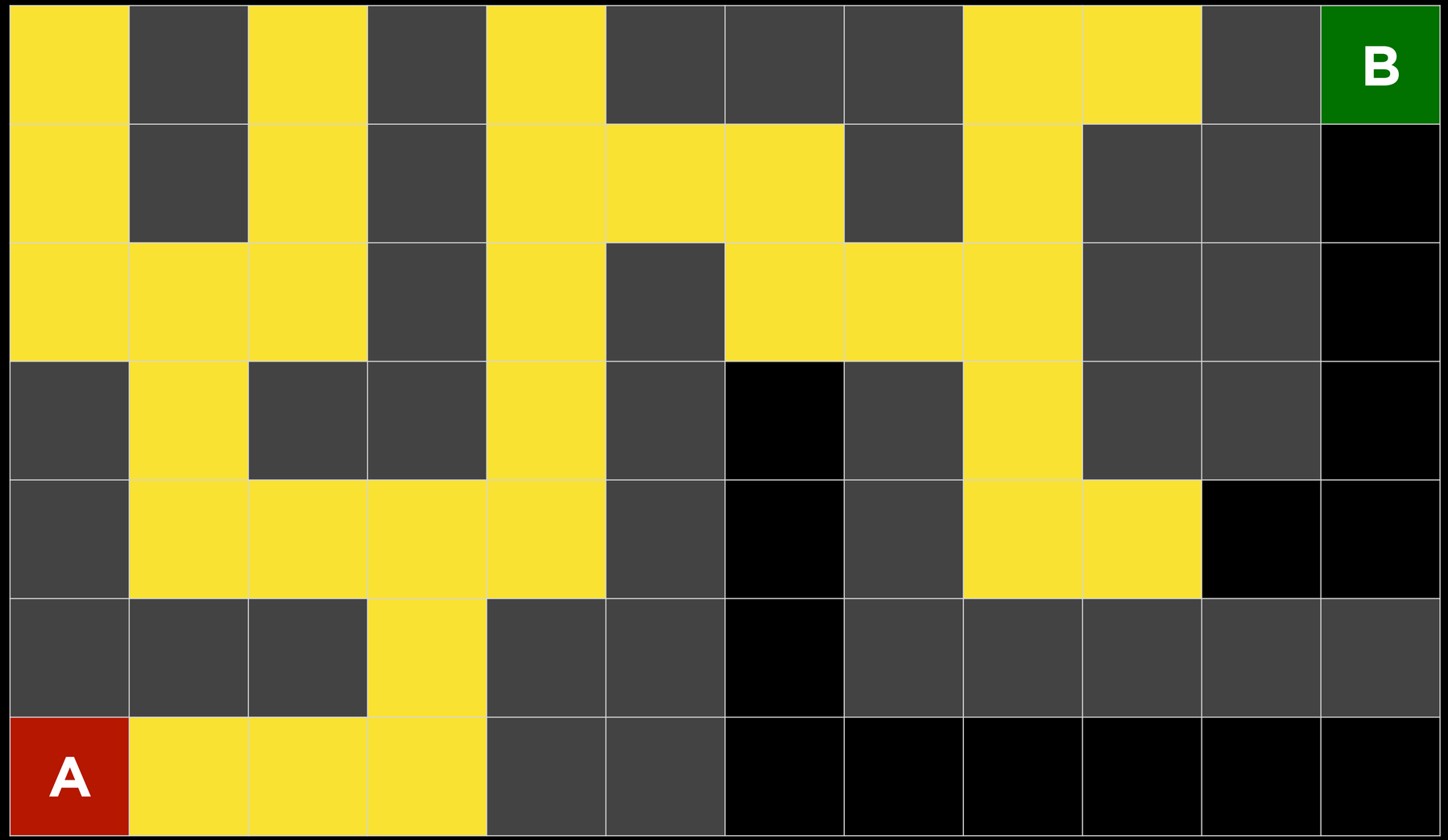
Depth-First Search



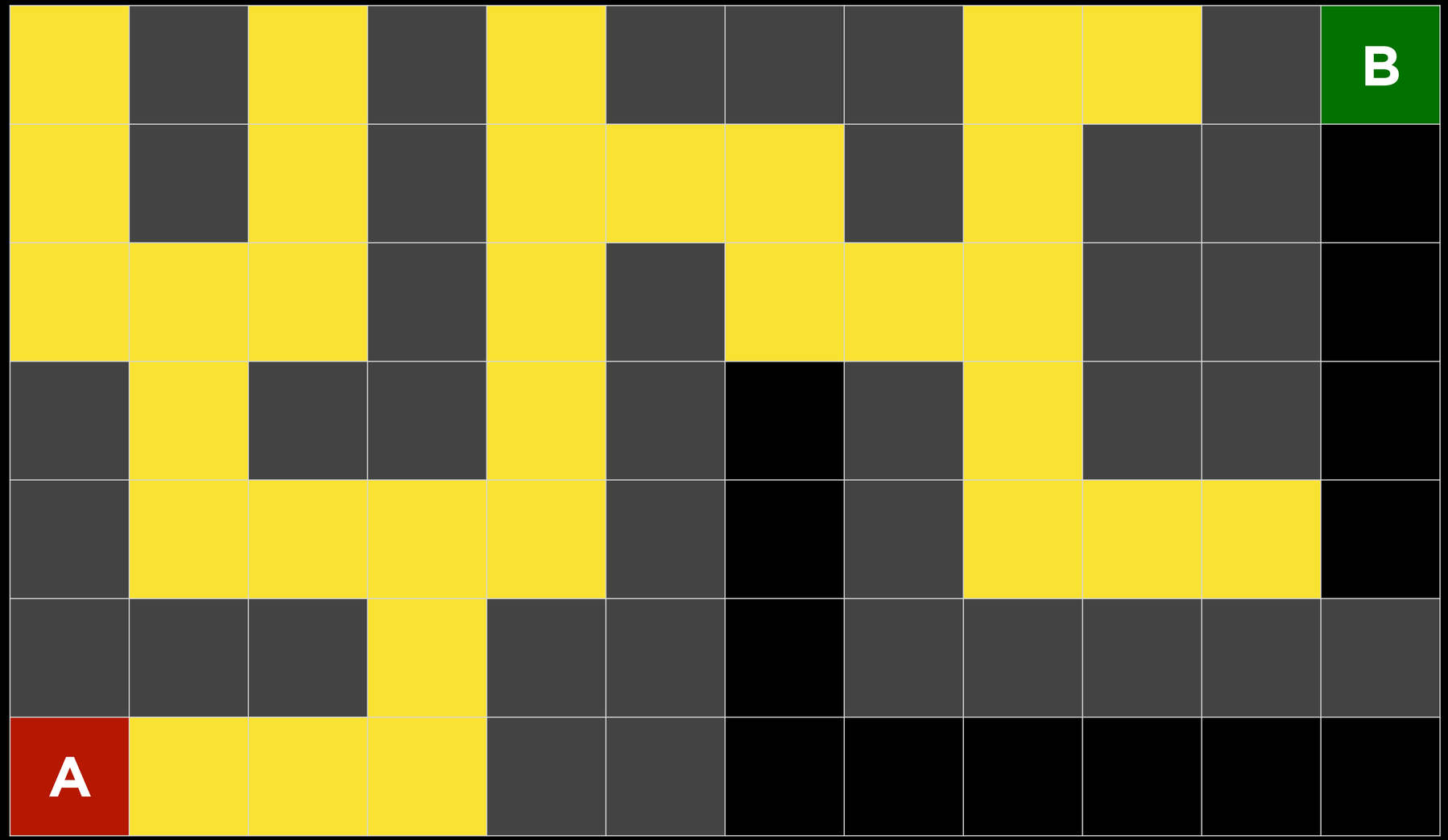
Depth-First Search



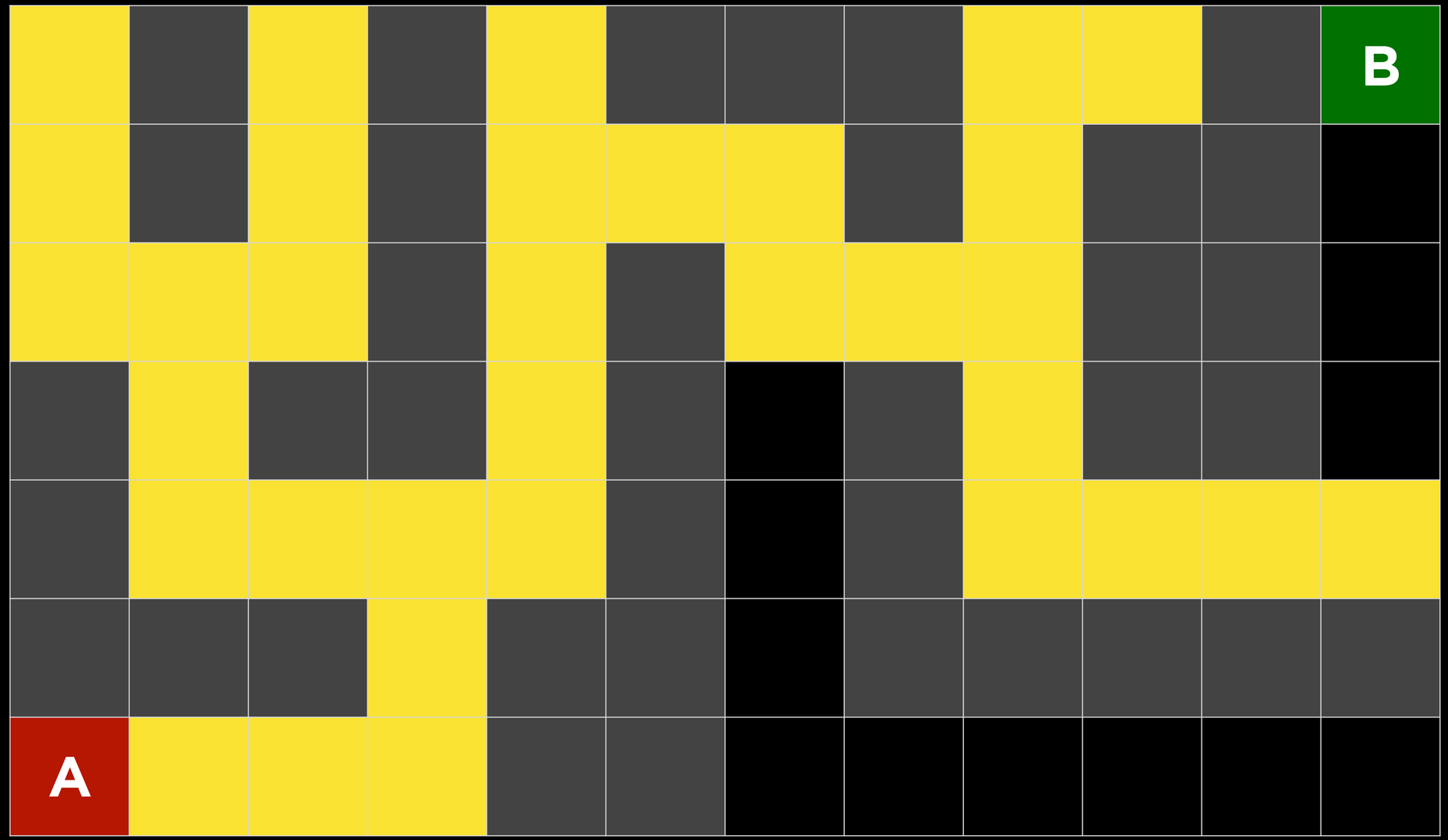
Depth-First Search



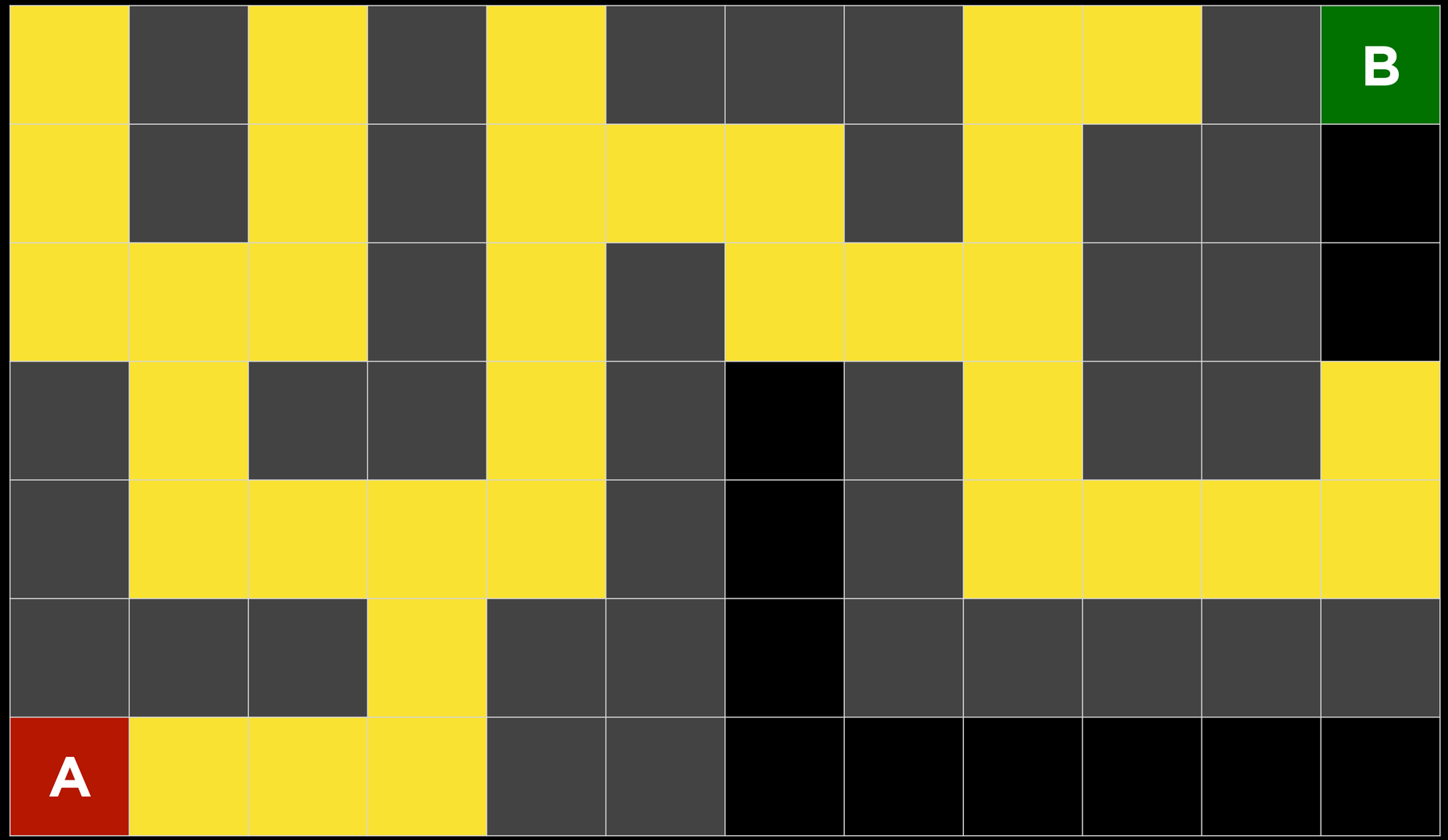
Depth-First Search



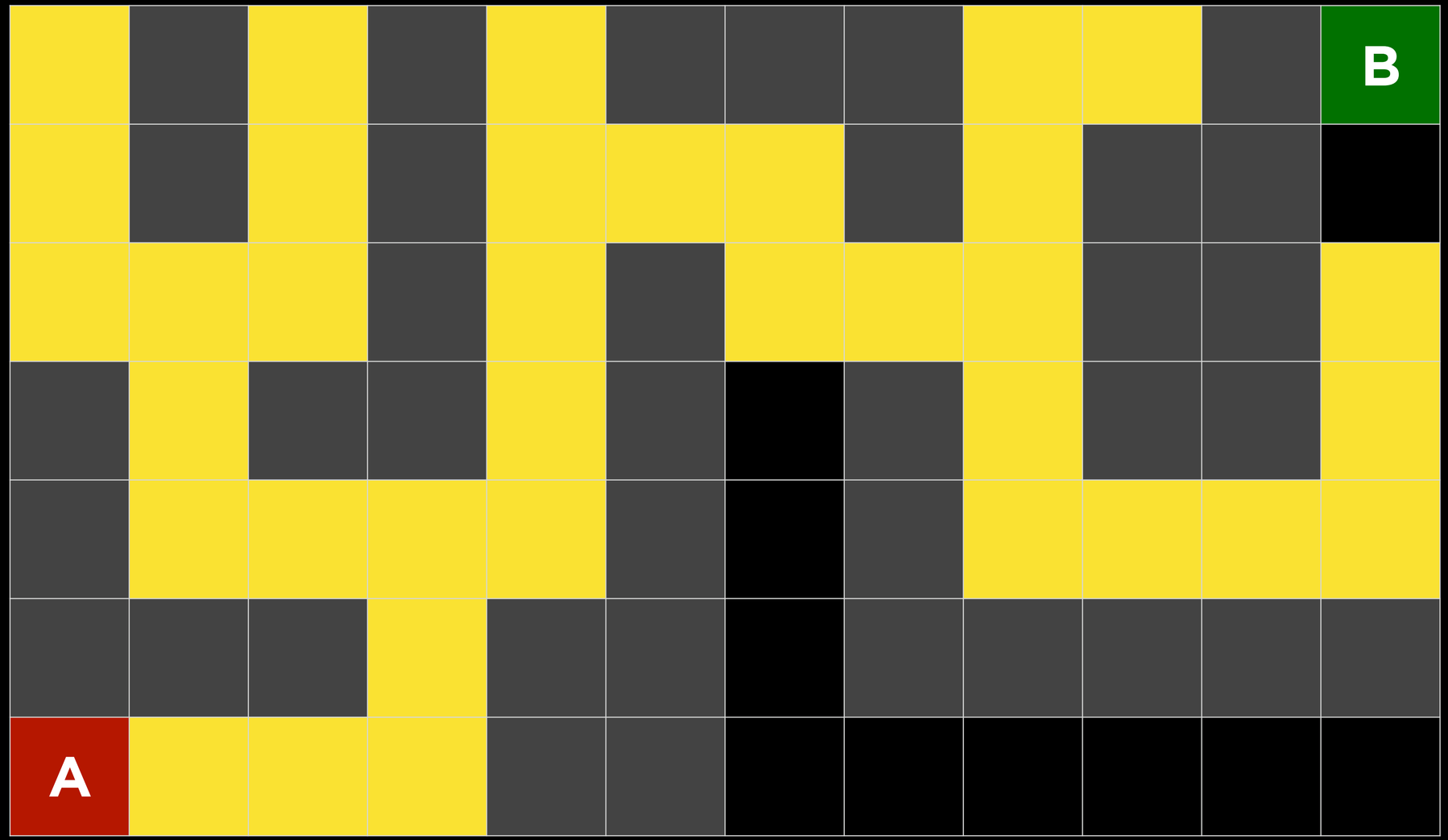
Depth-First Search



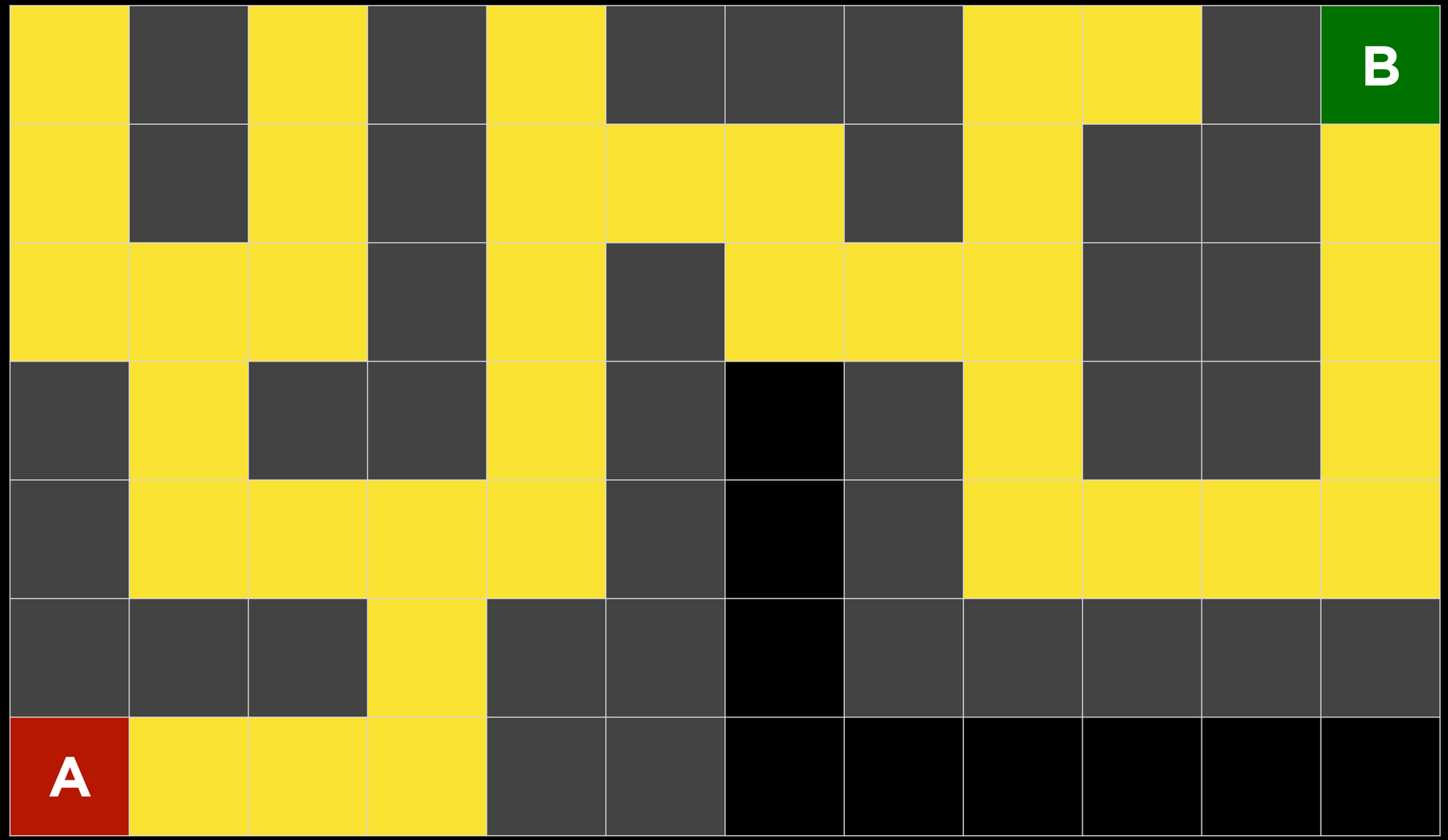
Depth-First Search



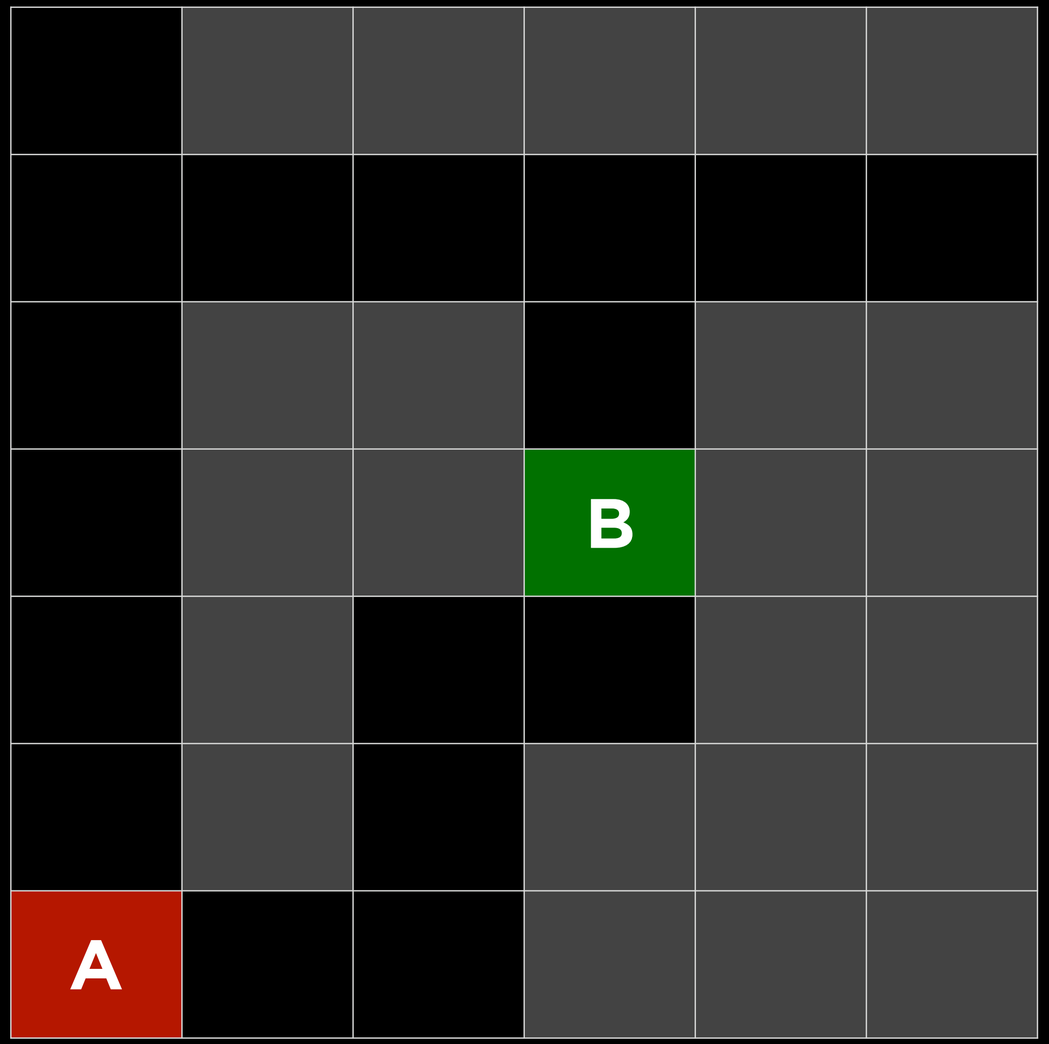
Depth-First Search



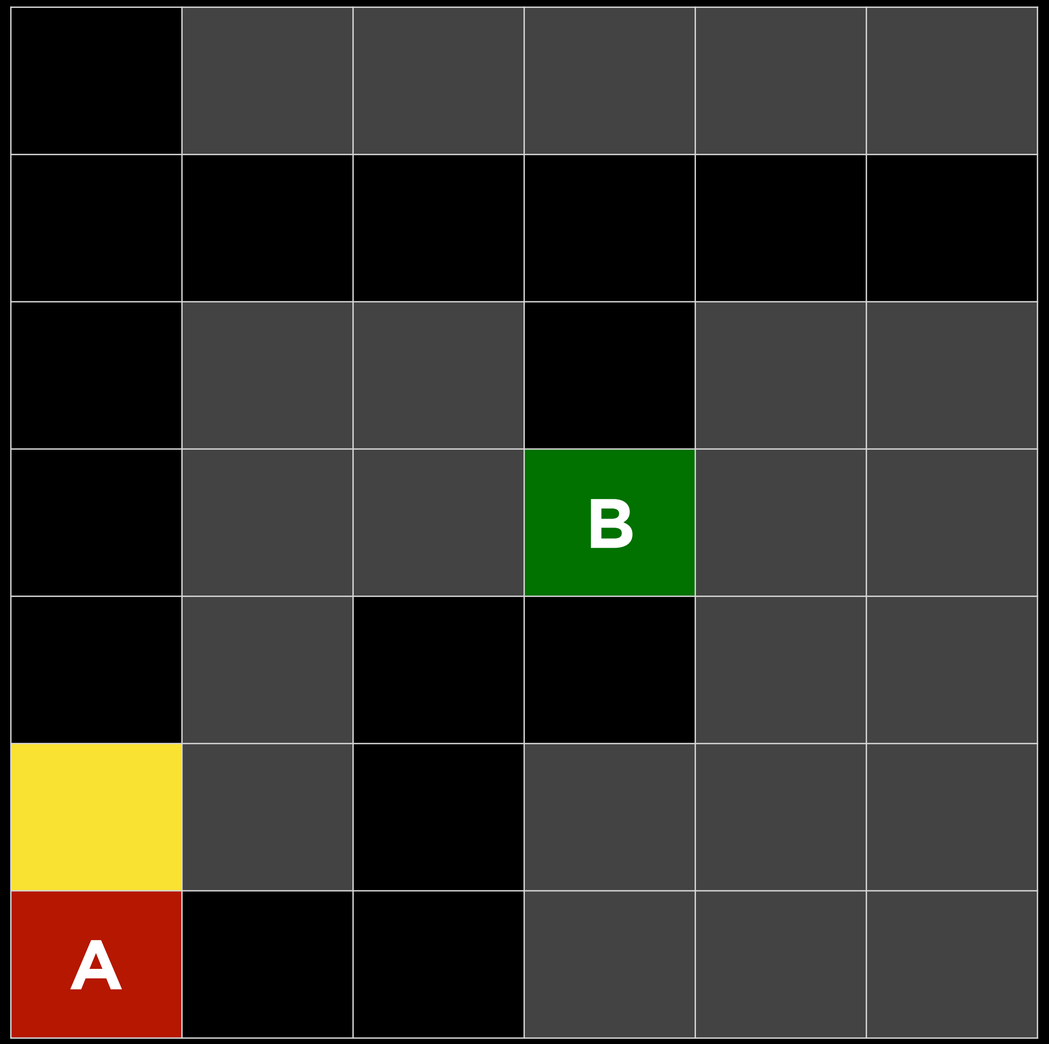
Depth-First Search



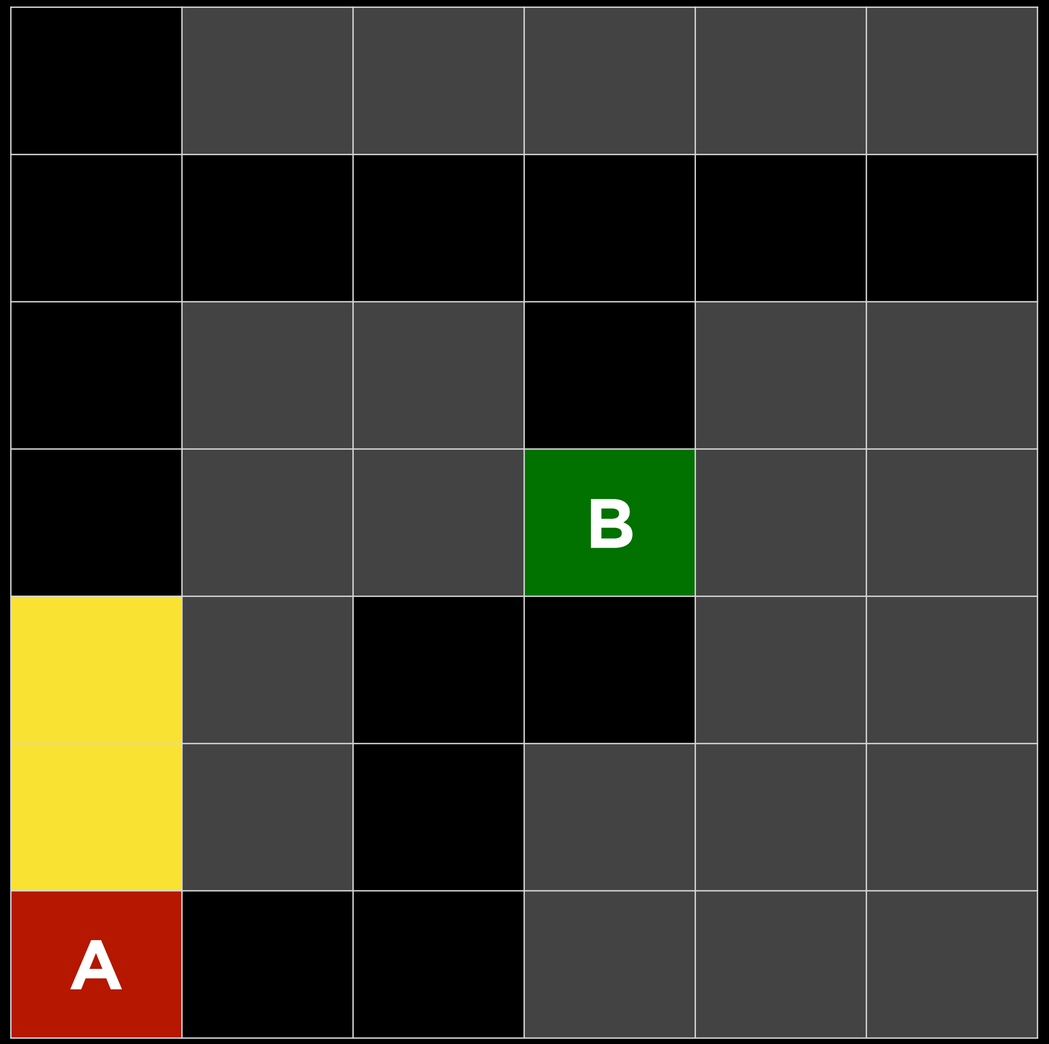
Depth-First Search



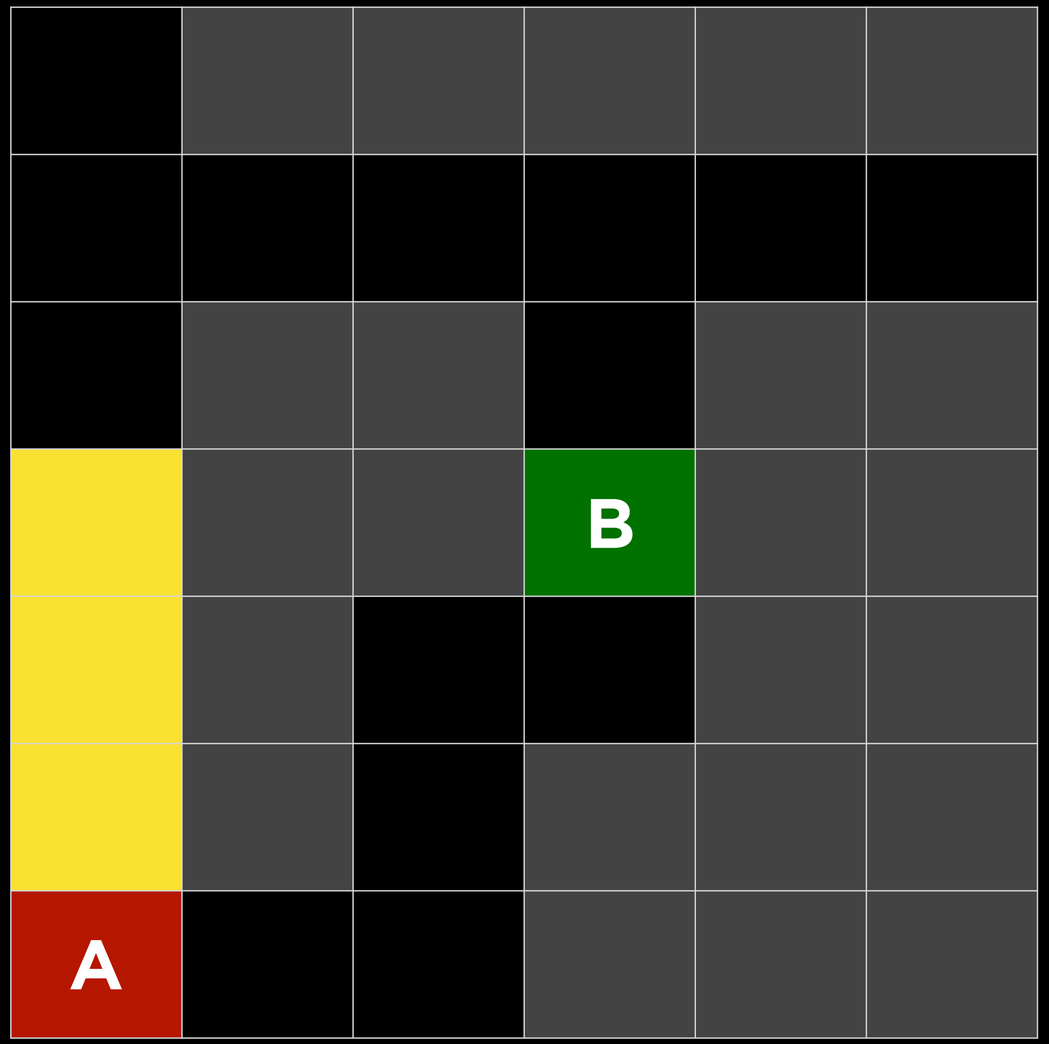
Depth-First Search



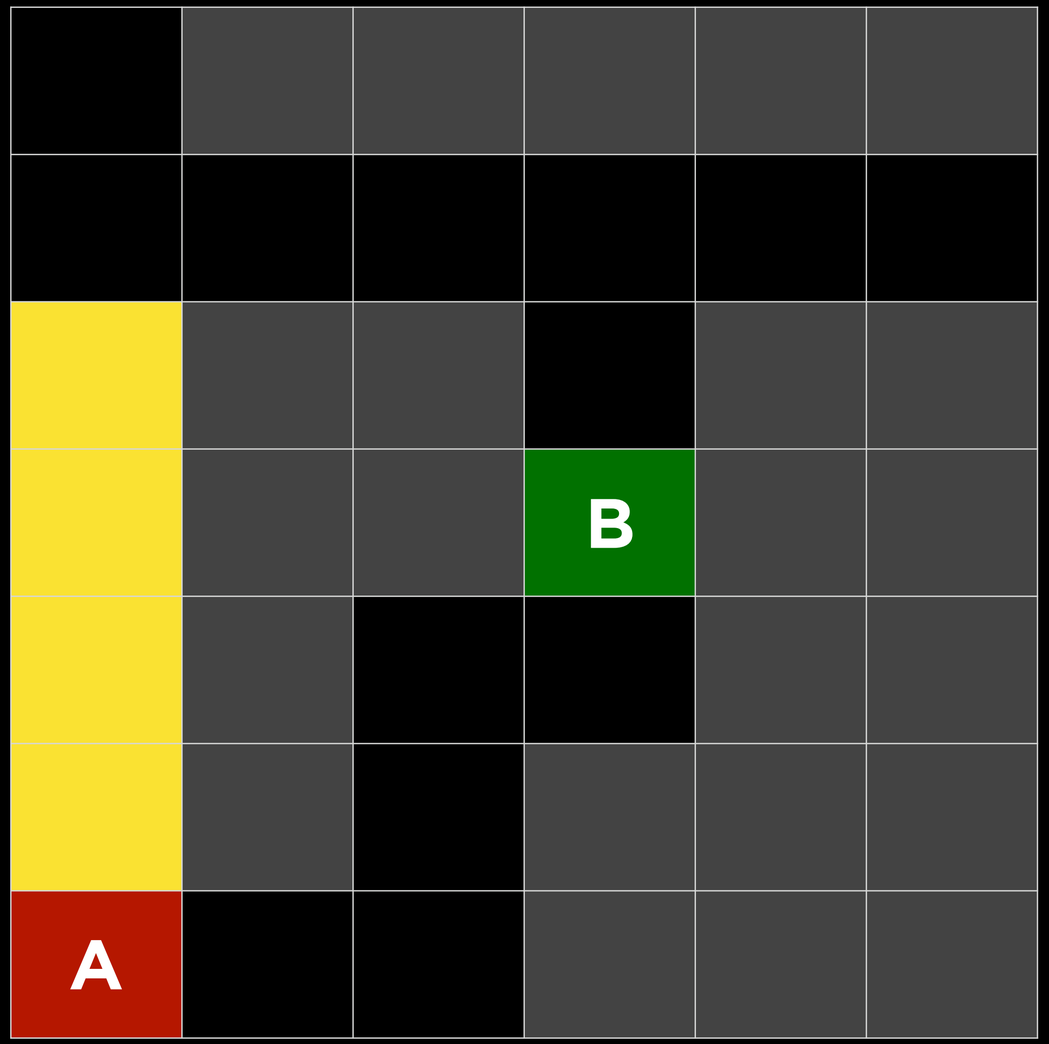
Depth-First Search



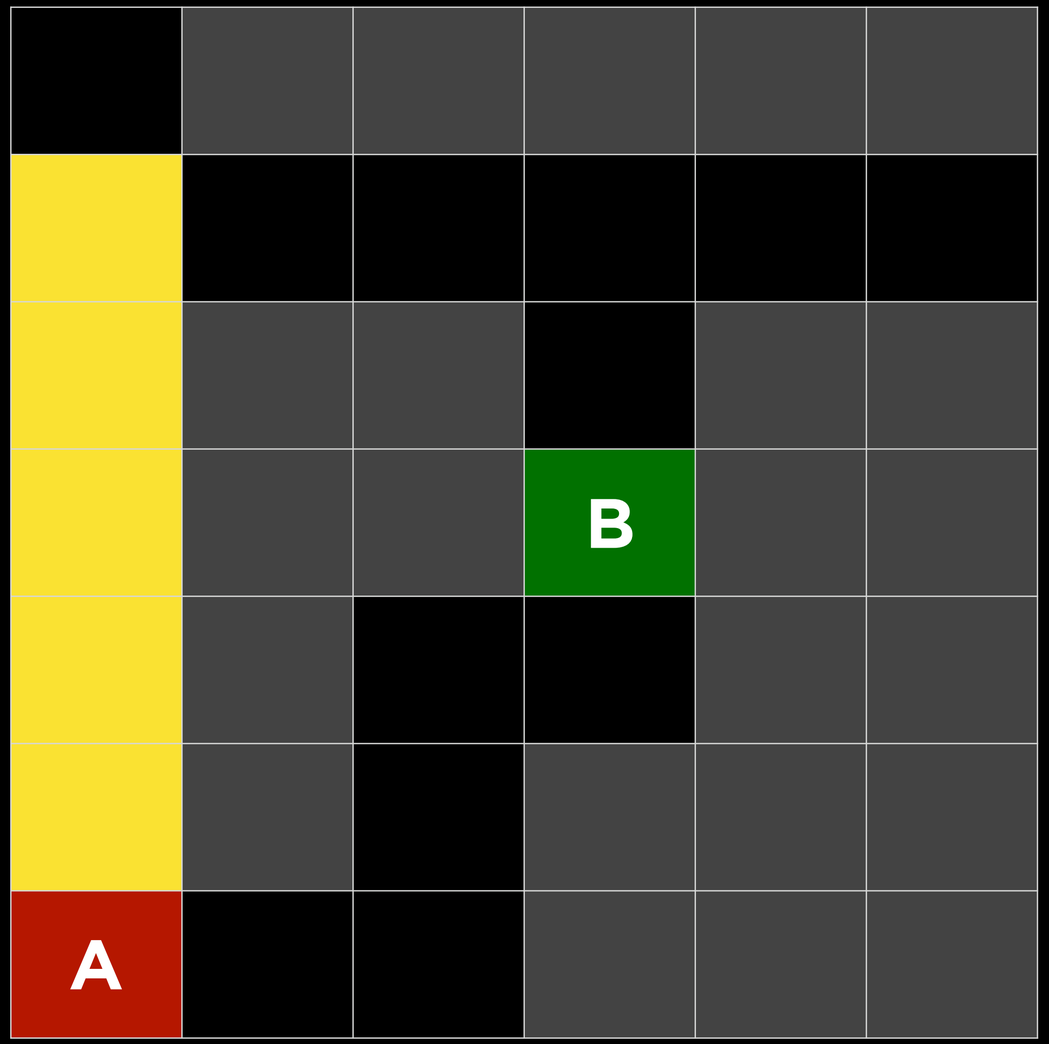
Depth-First Search



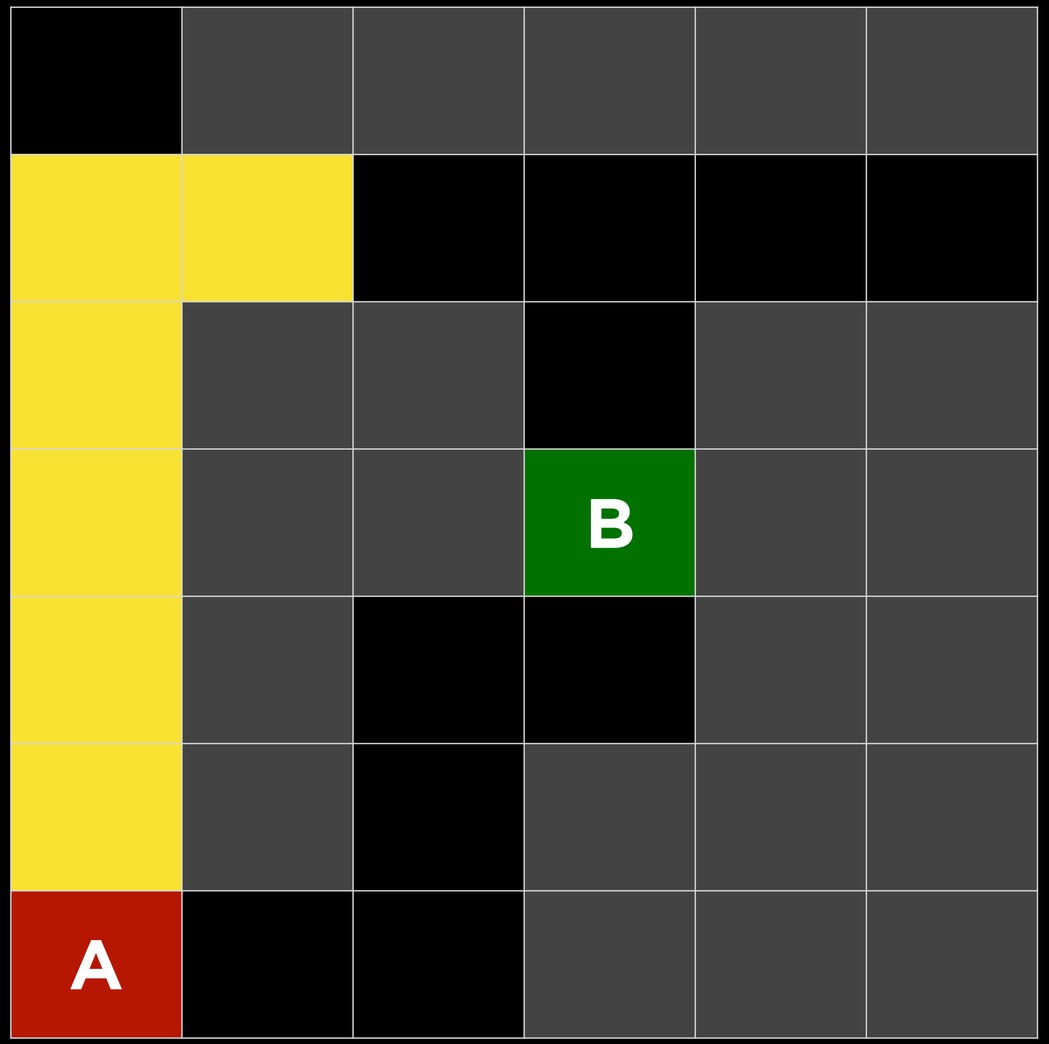
Depth-First Search



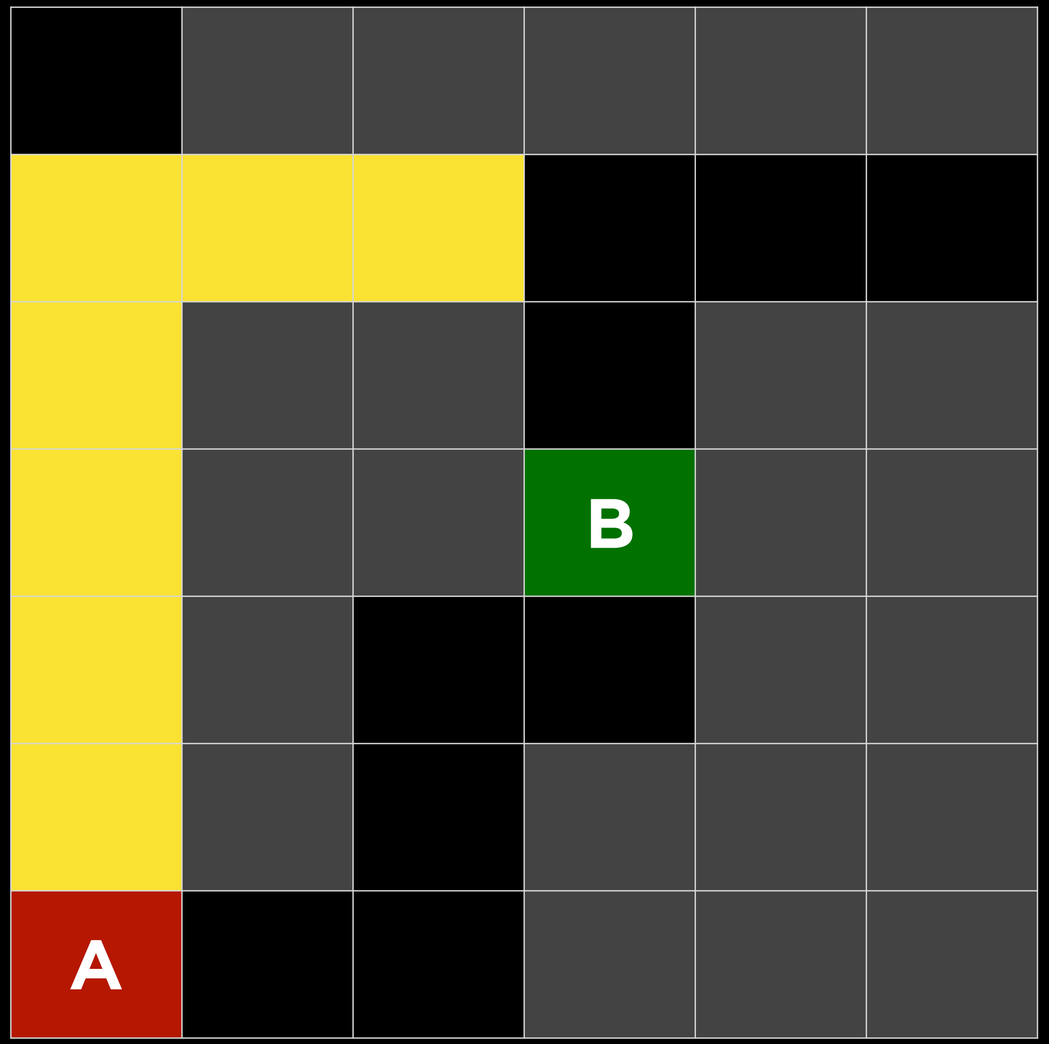
Depth-First Search



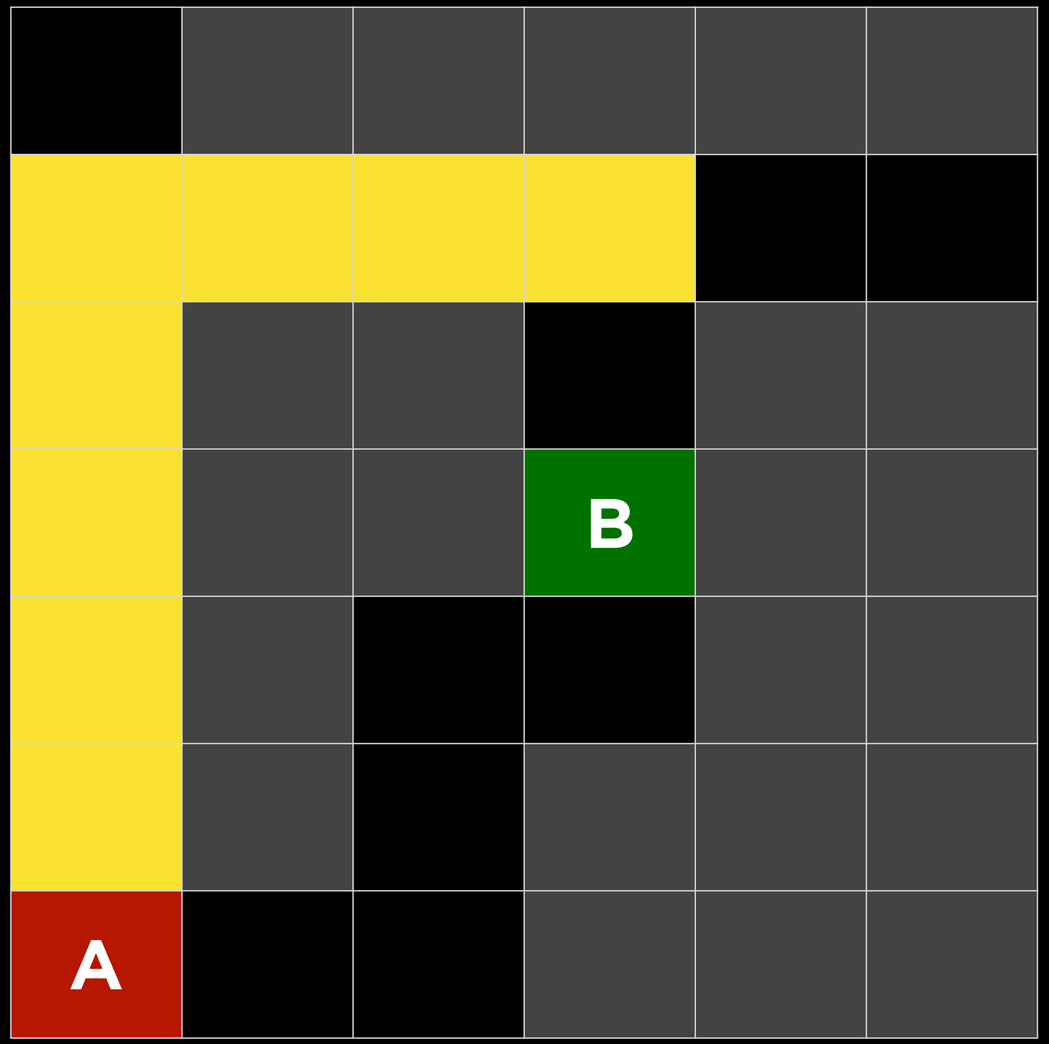
Depth-First Search



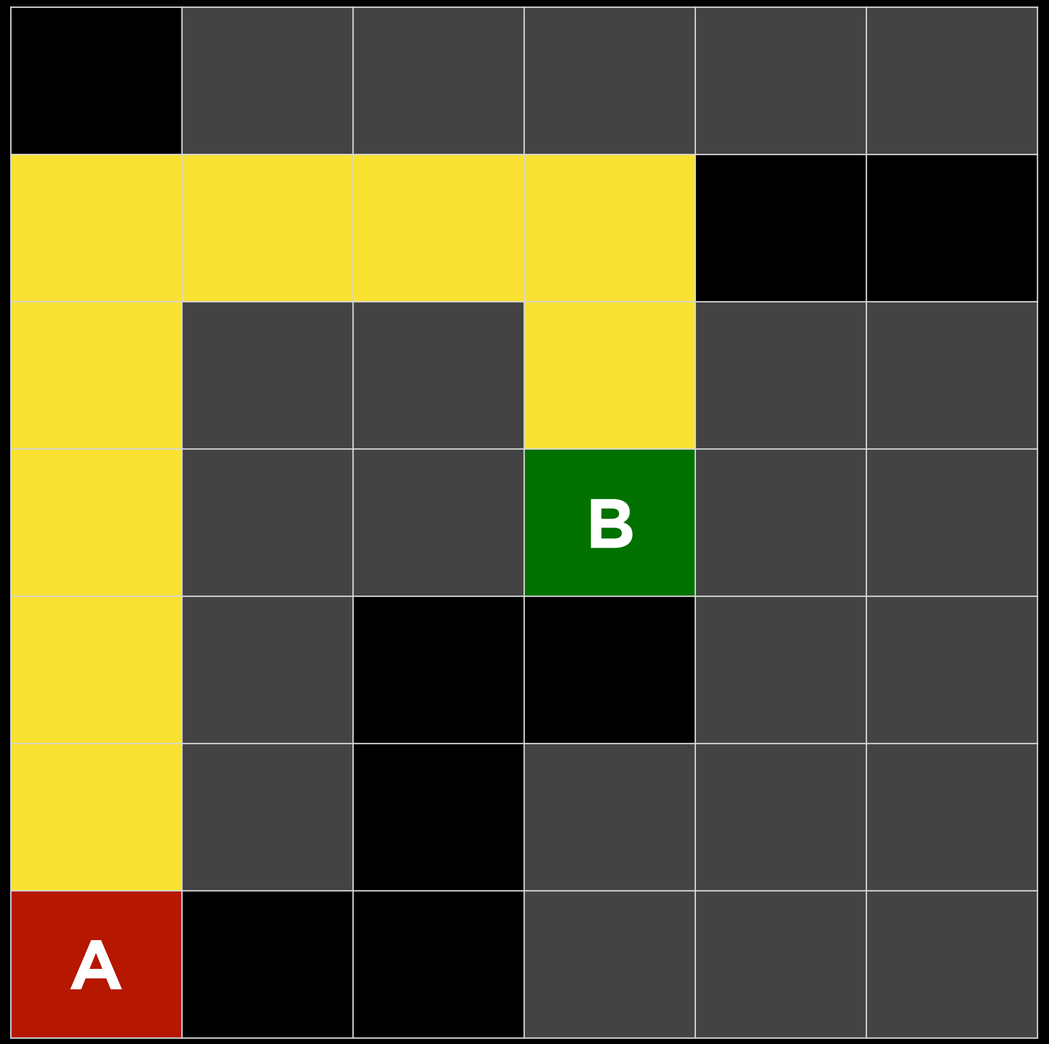
Depth-First Search



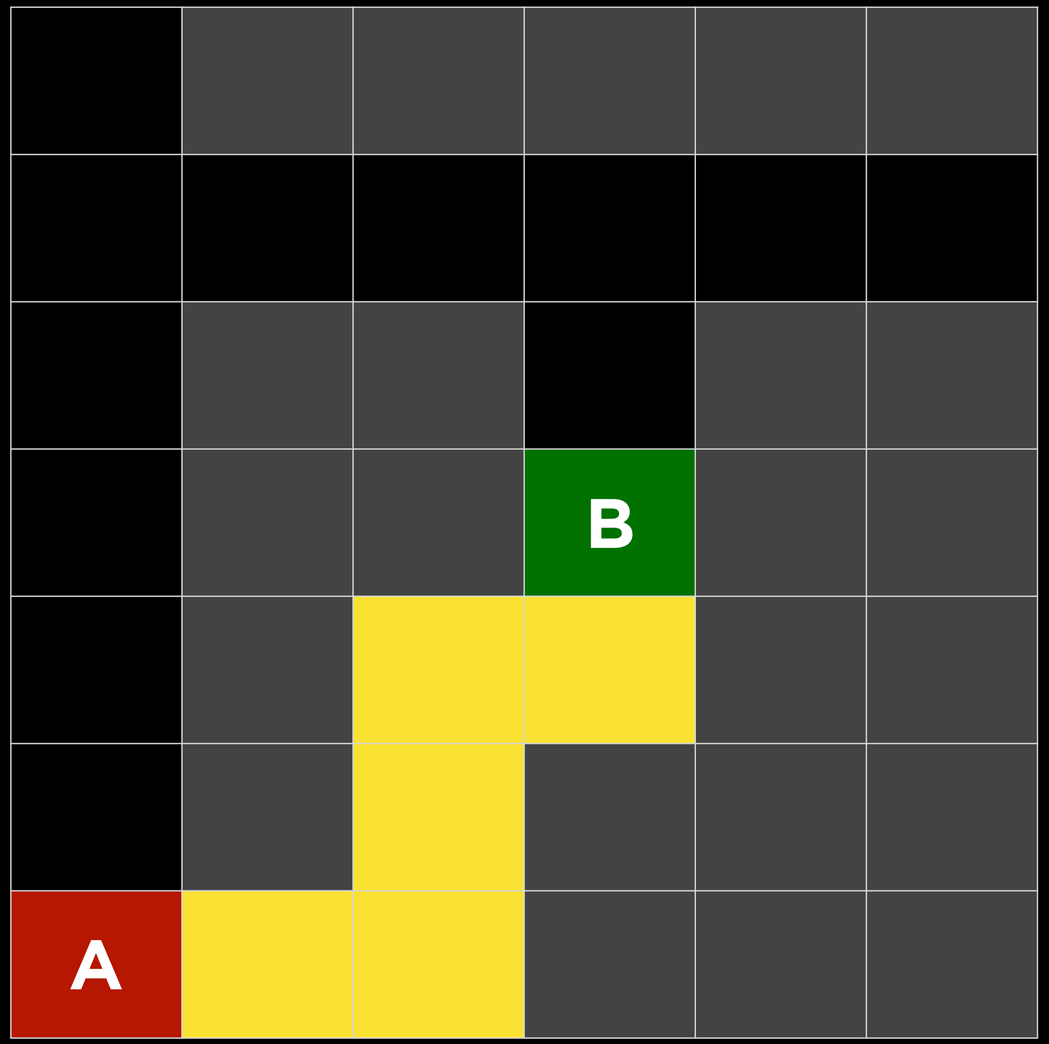
Depth-First Search



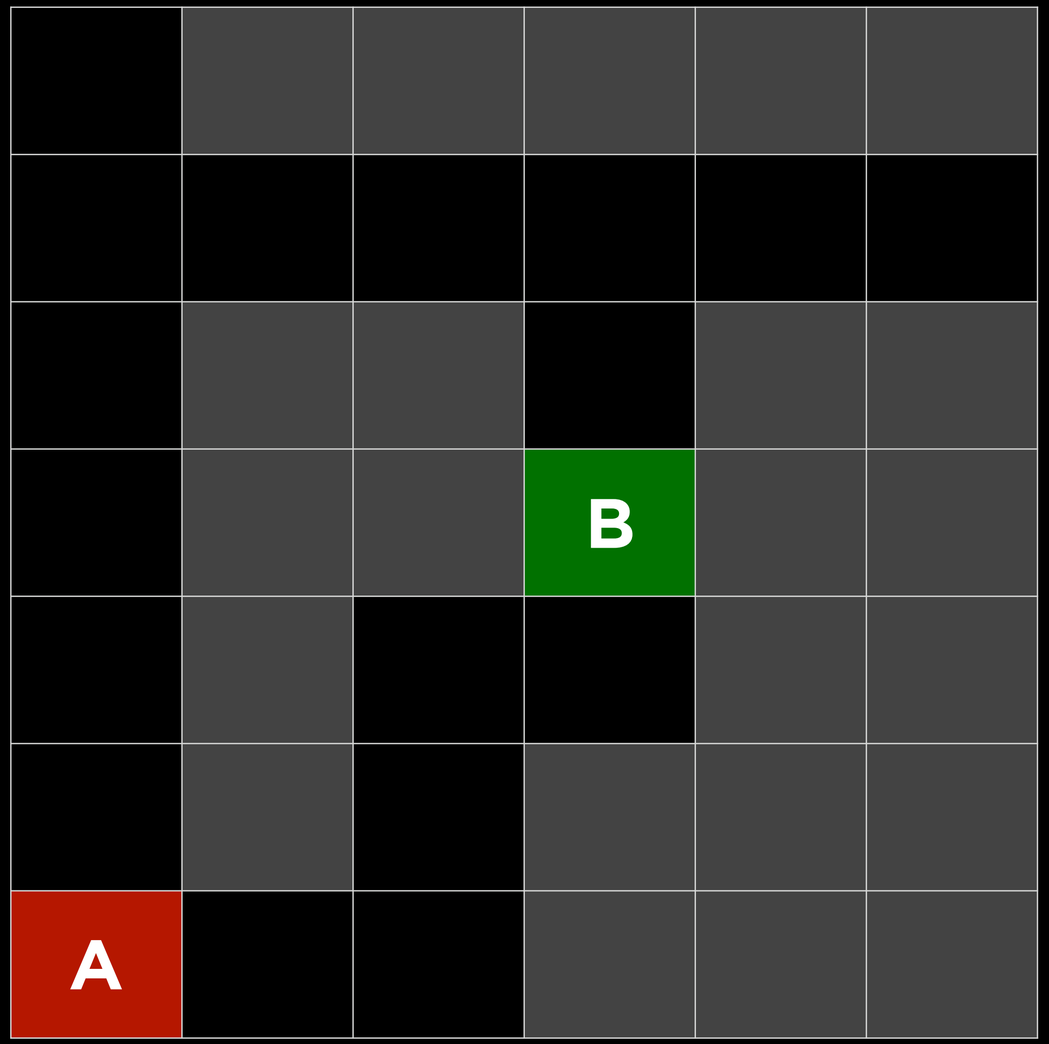
Depth-First Search



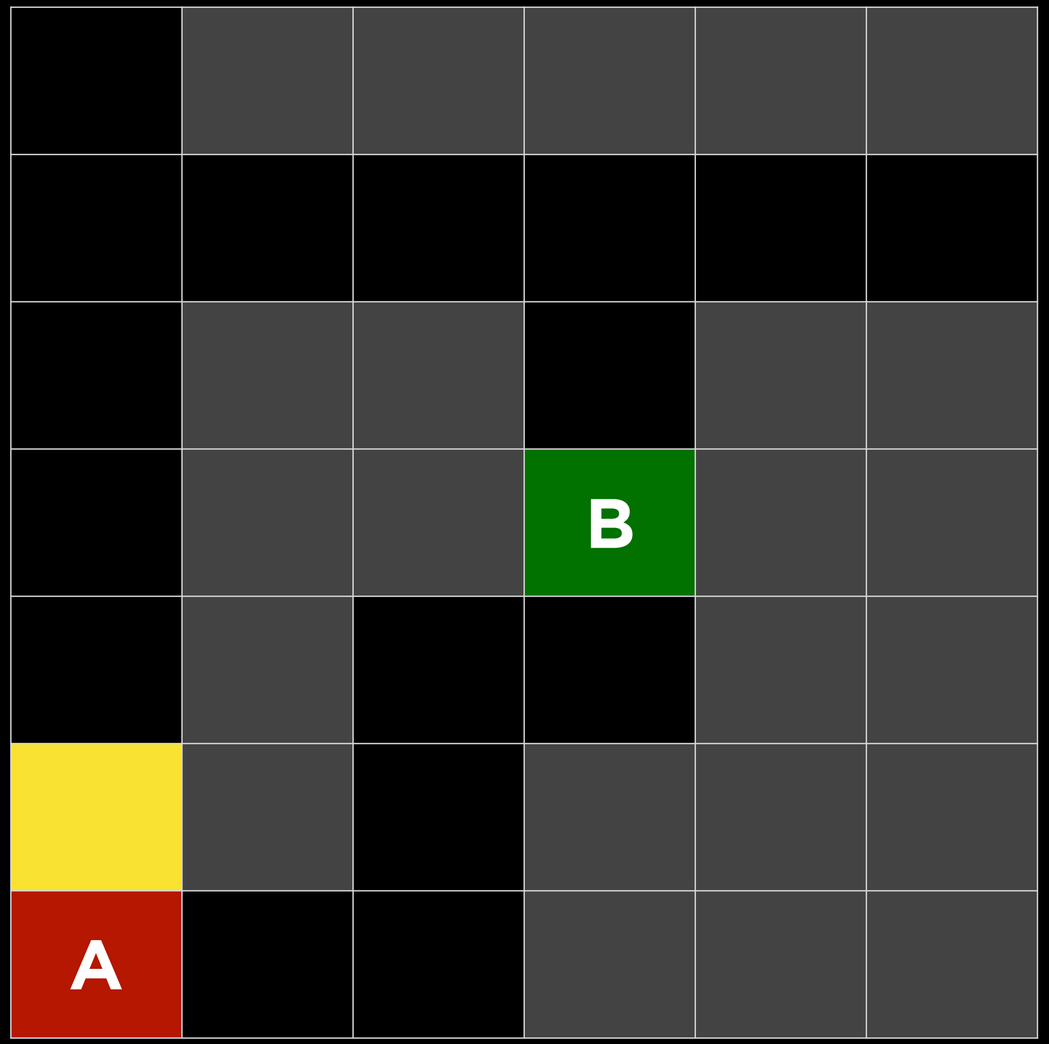
Depth-First Search



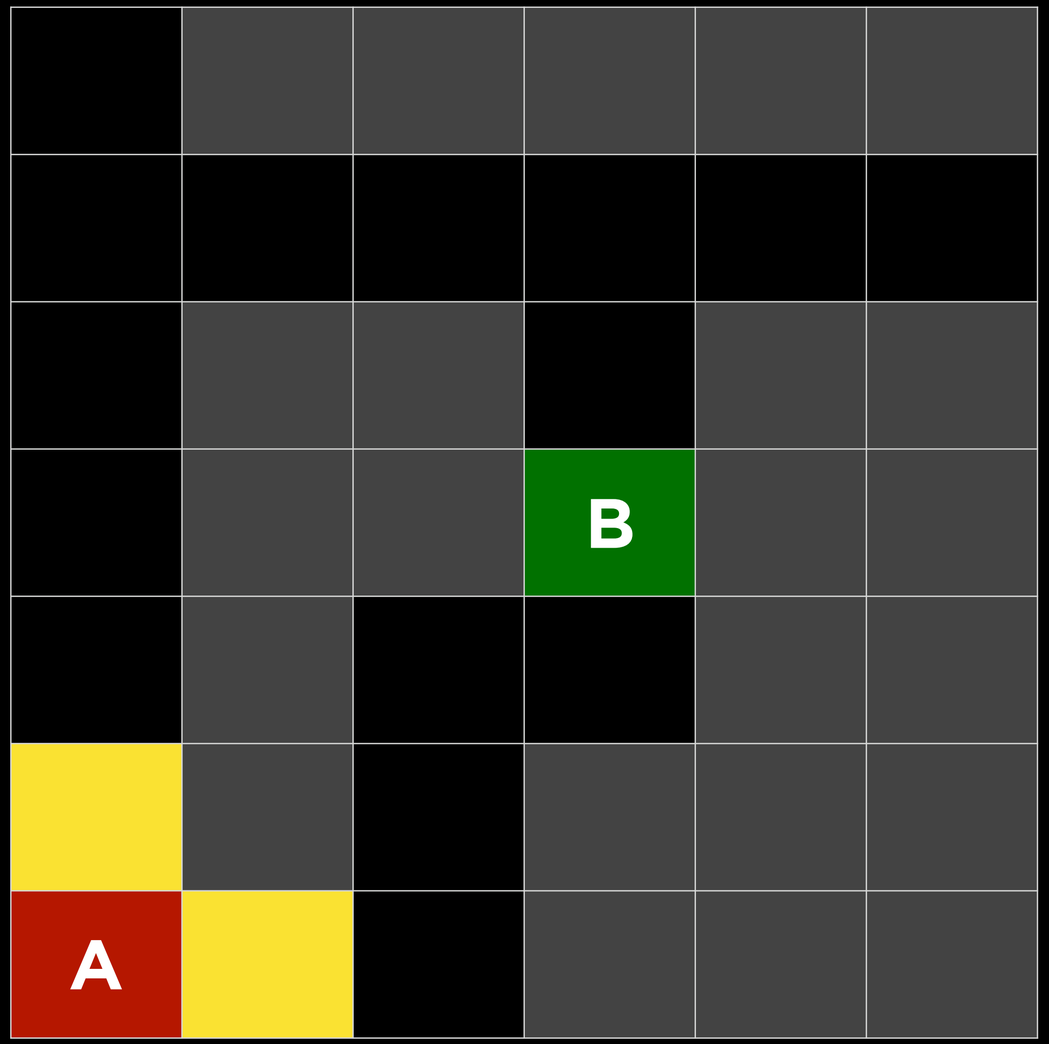
Breadth-First Search



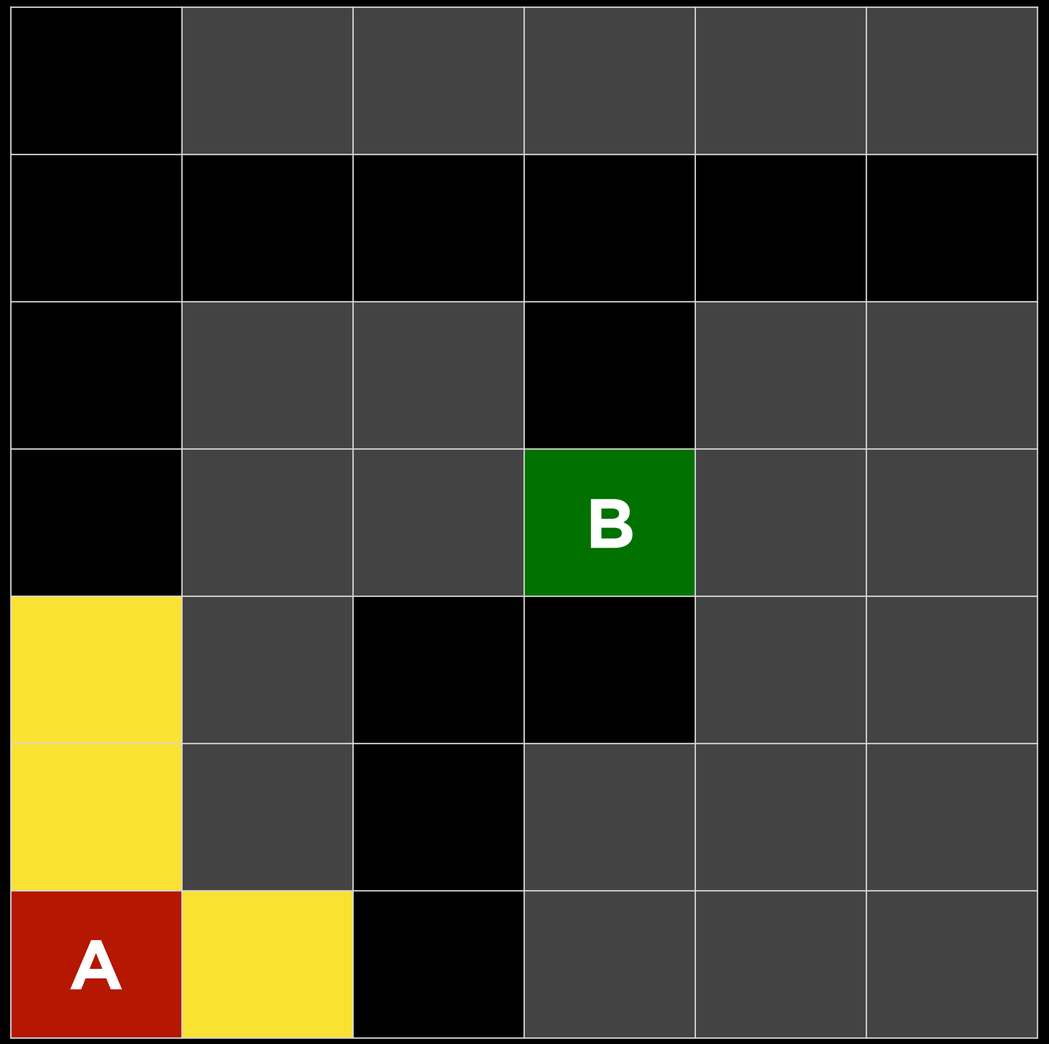
Breadth-First Search



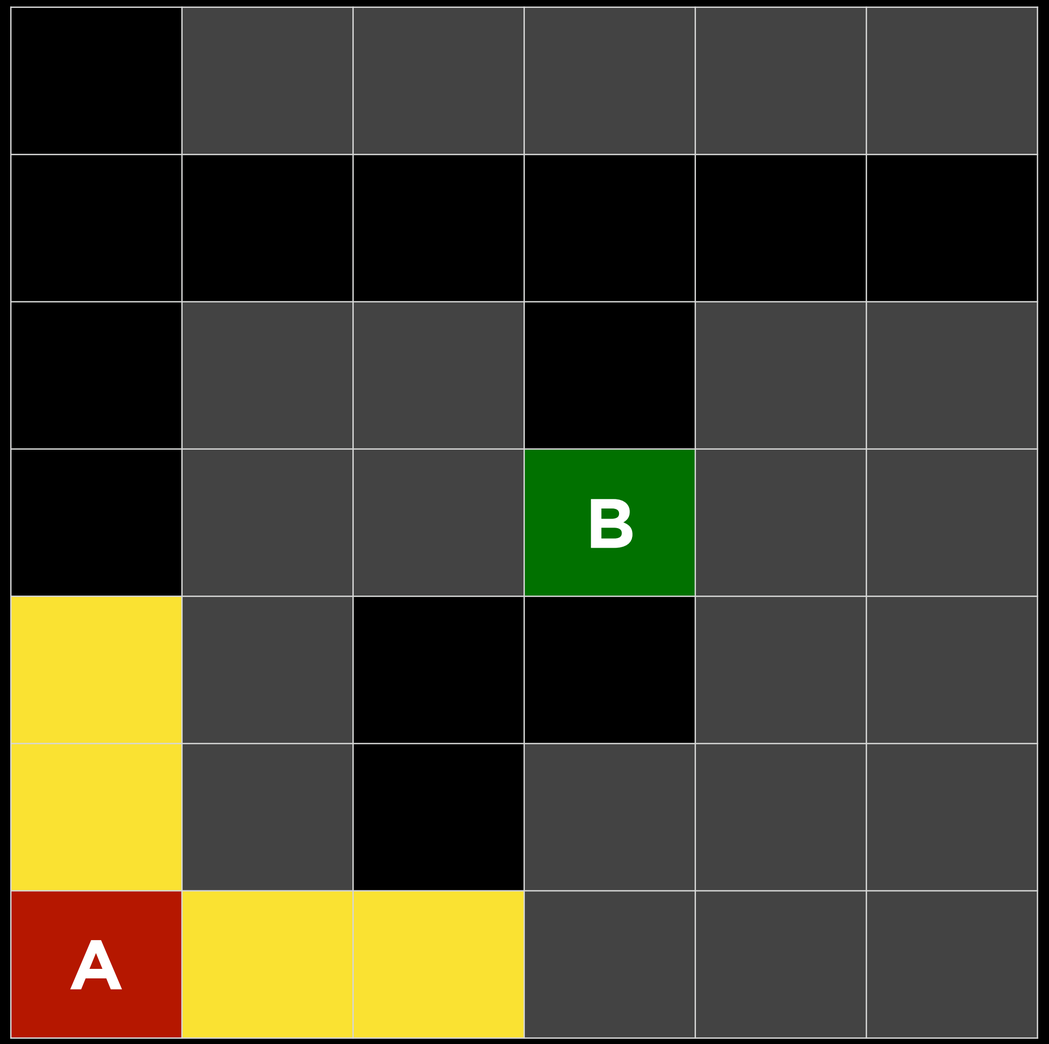
Breadth-First Search



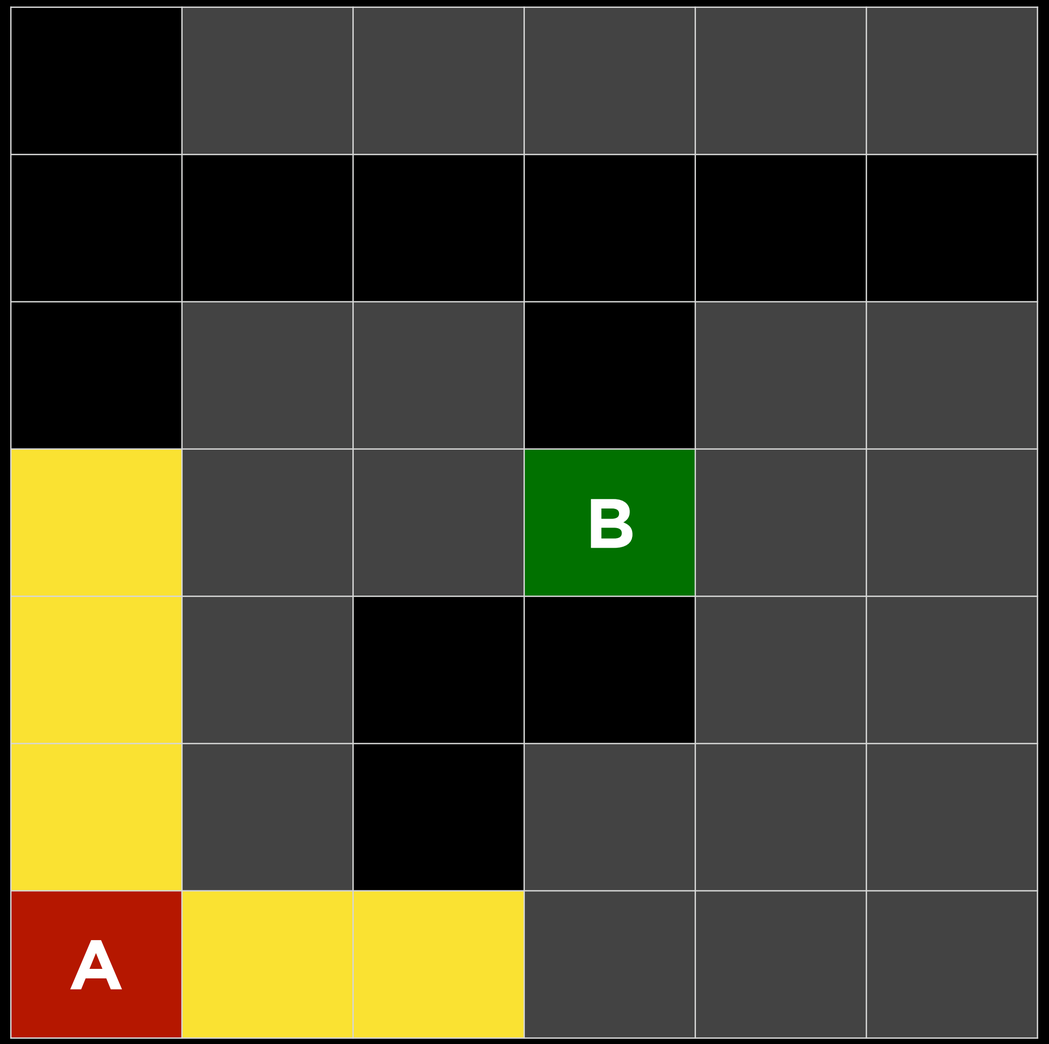
Breadth-First Search



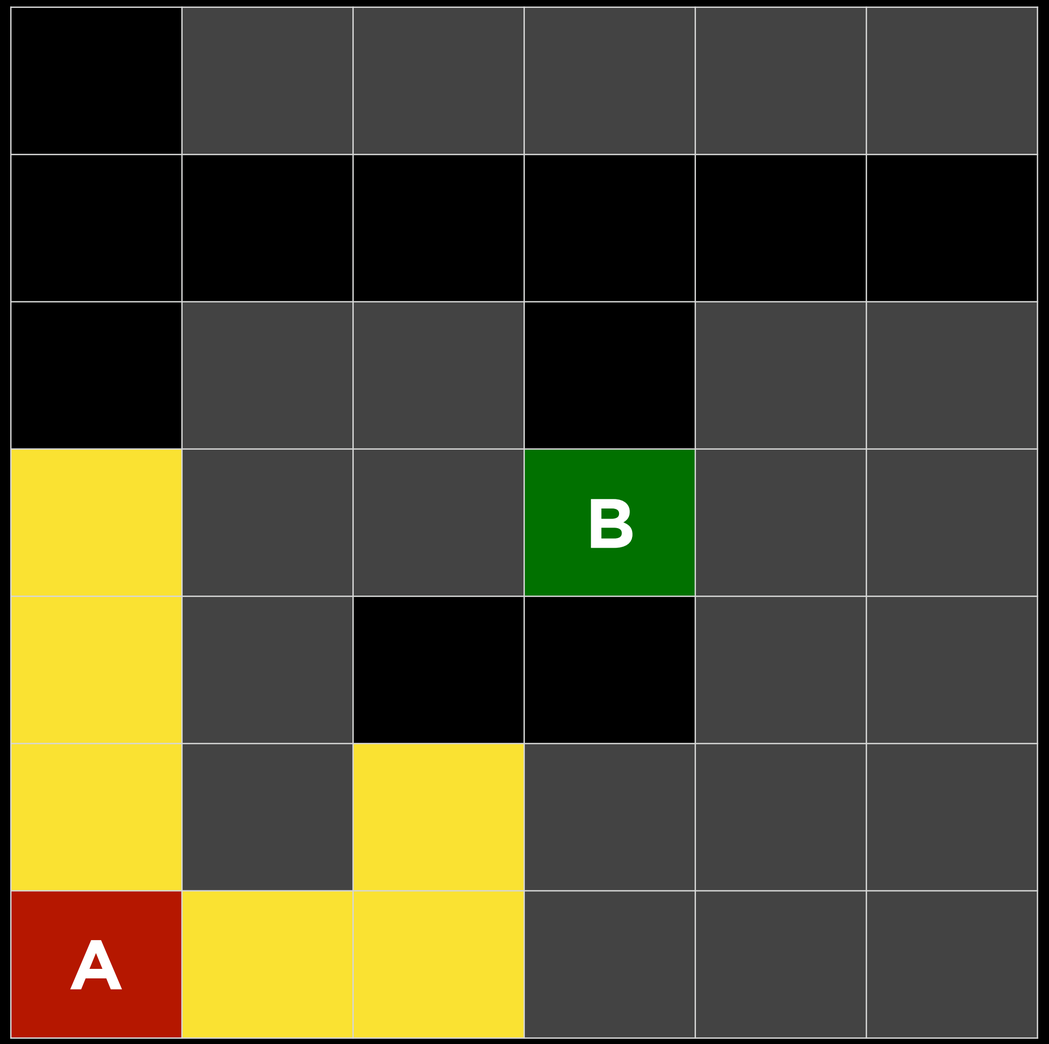
Breadth-First Search



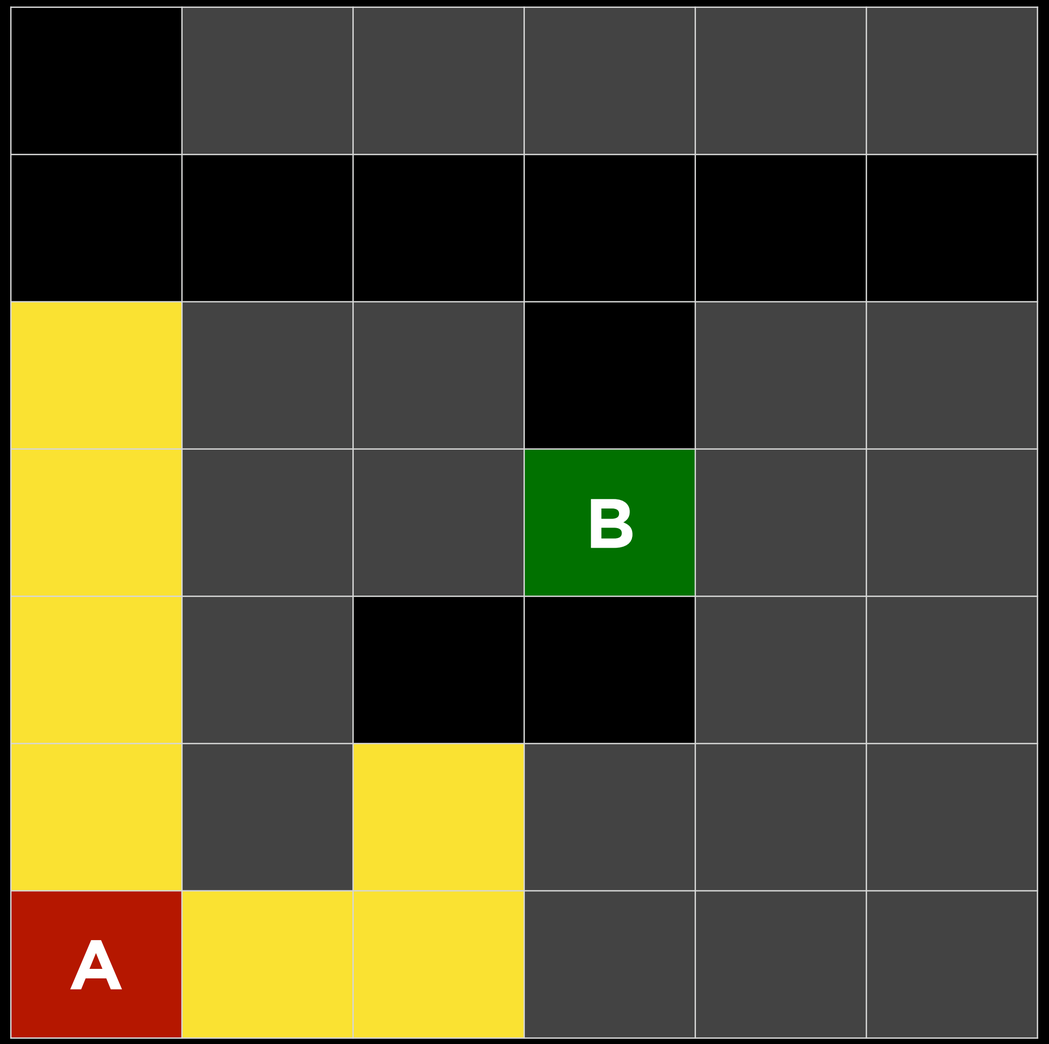
Breadth-First Search



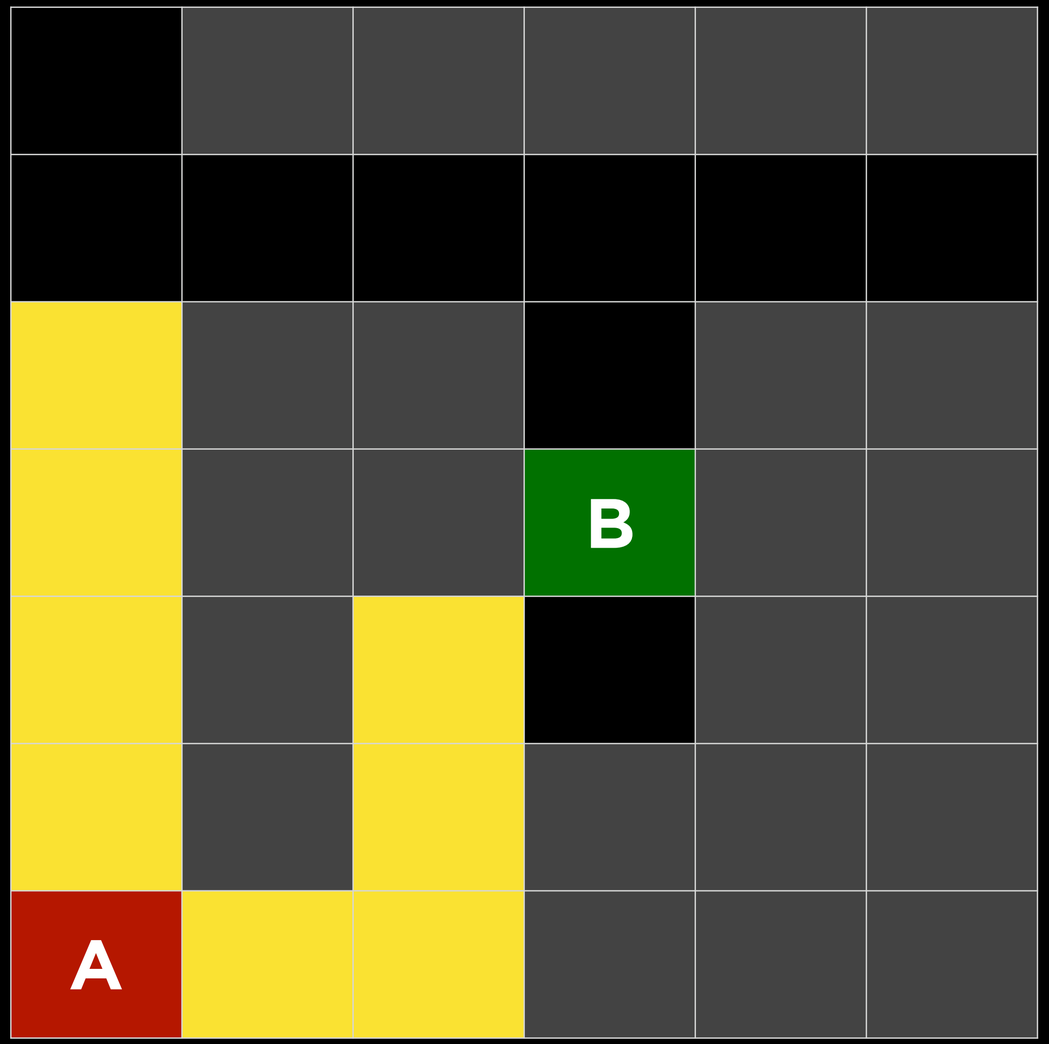
Breadth-First Search



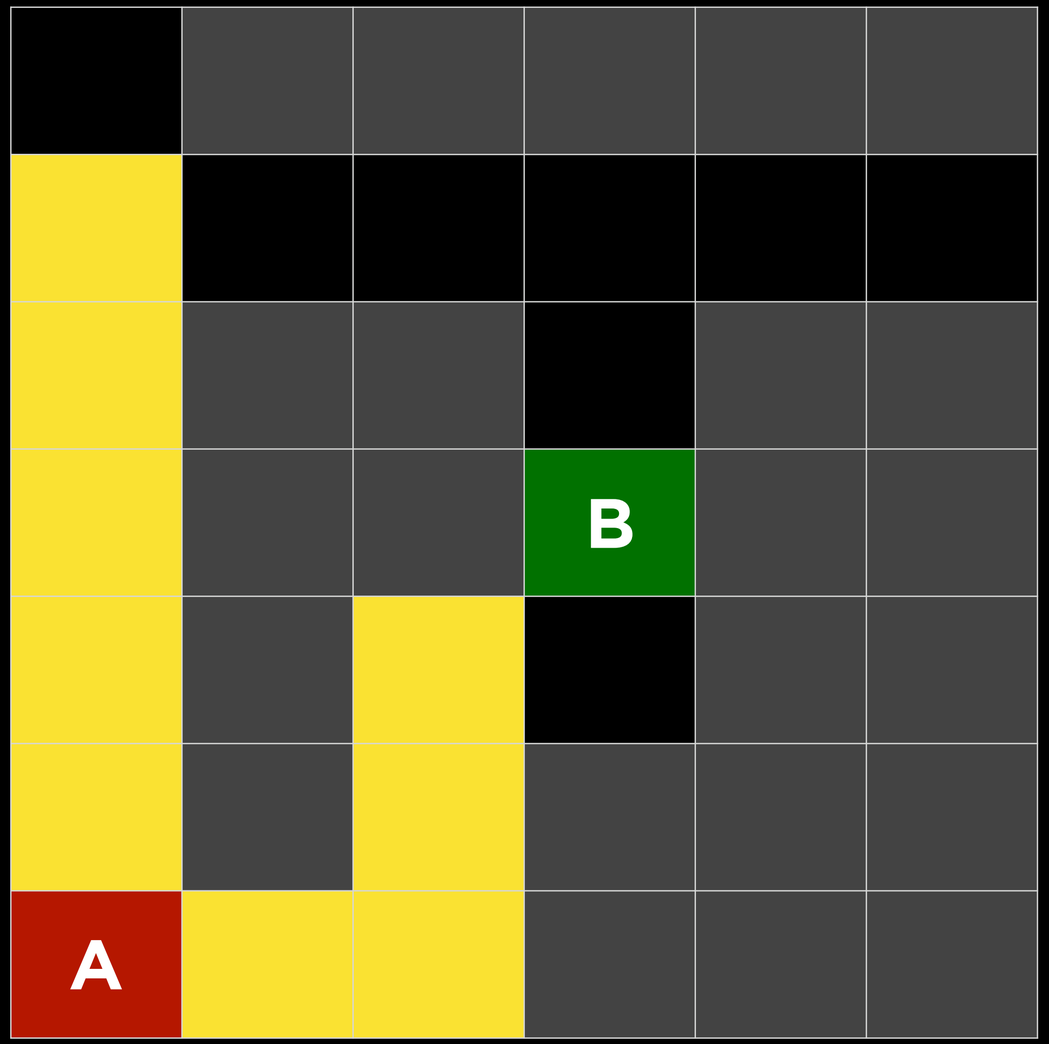
Breadth-First Search



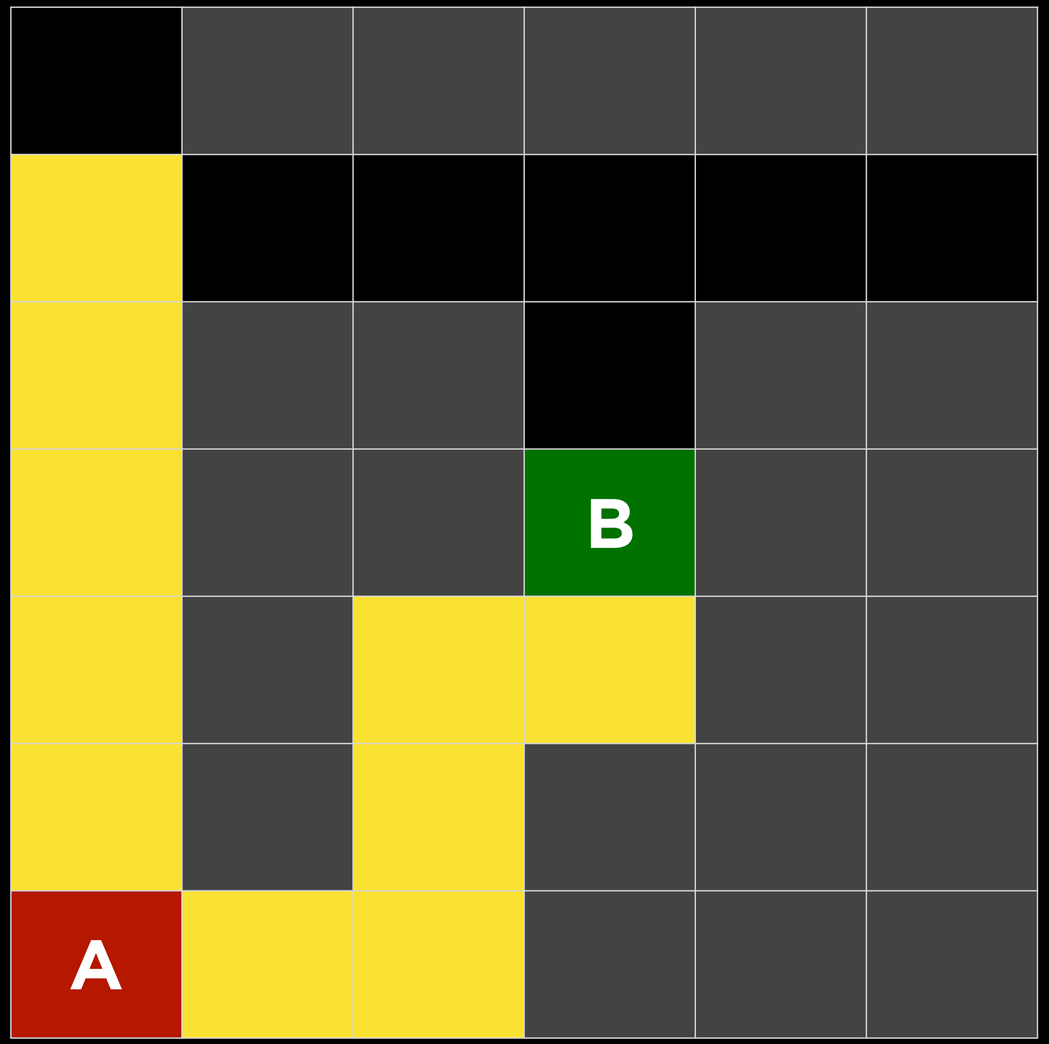
Breadth-First Search



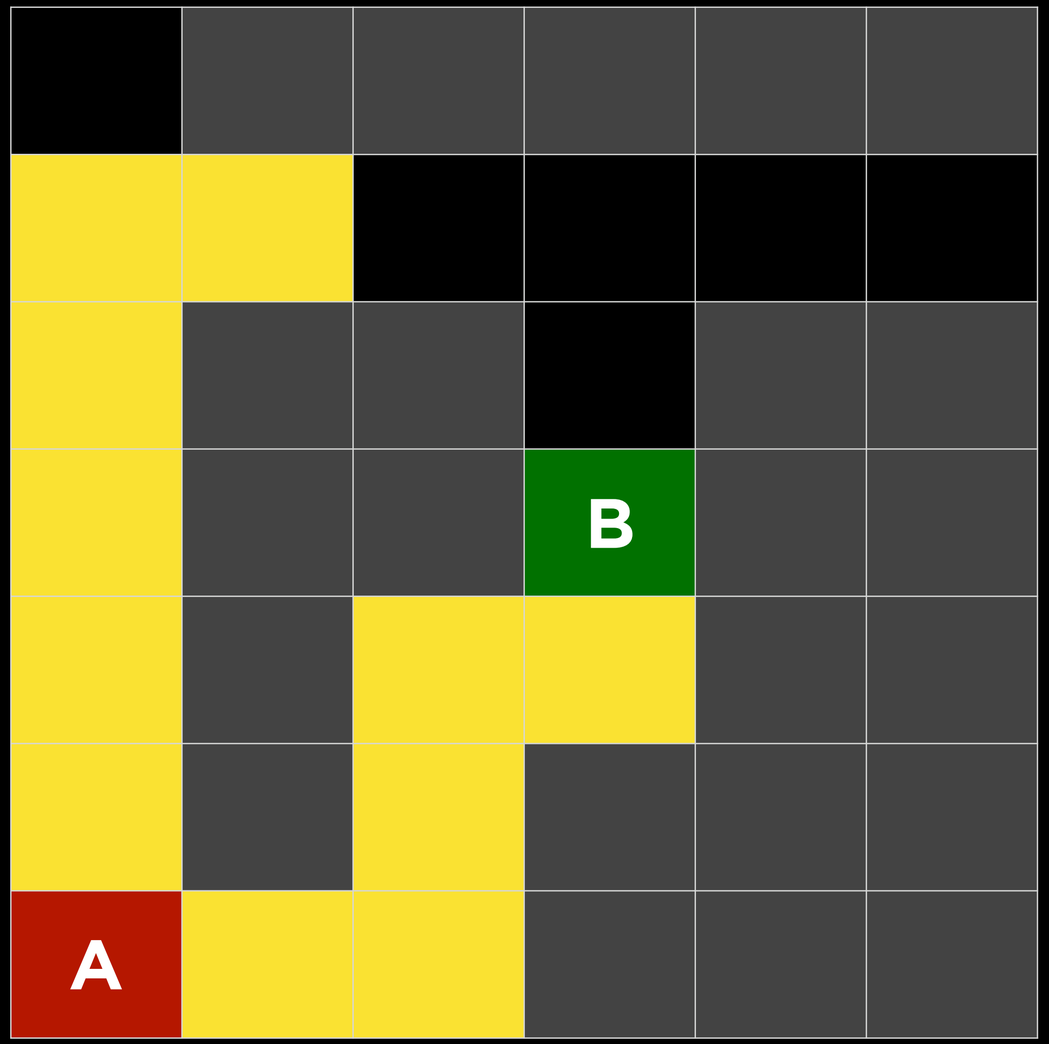
Breadth-First Search



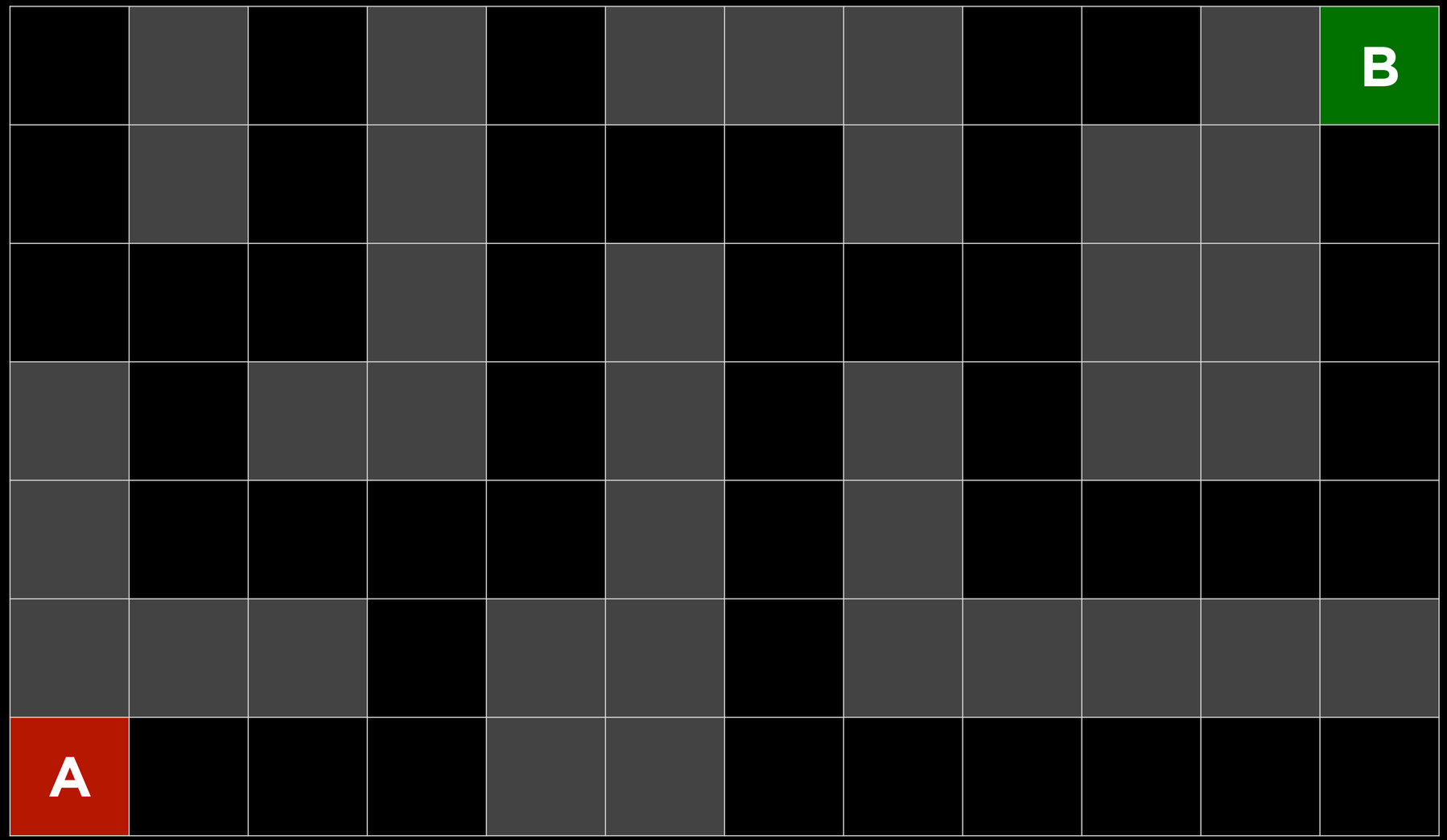
Breadth-First Search



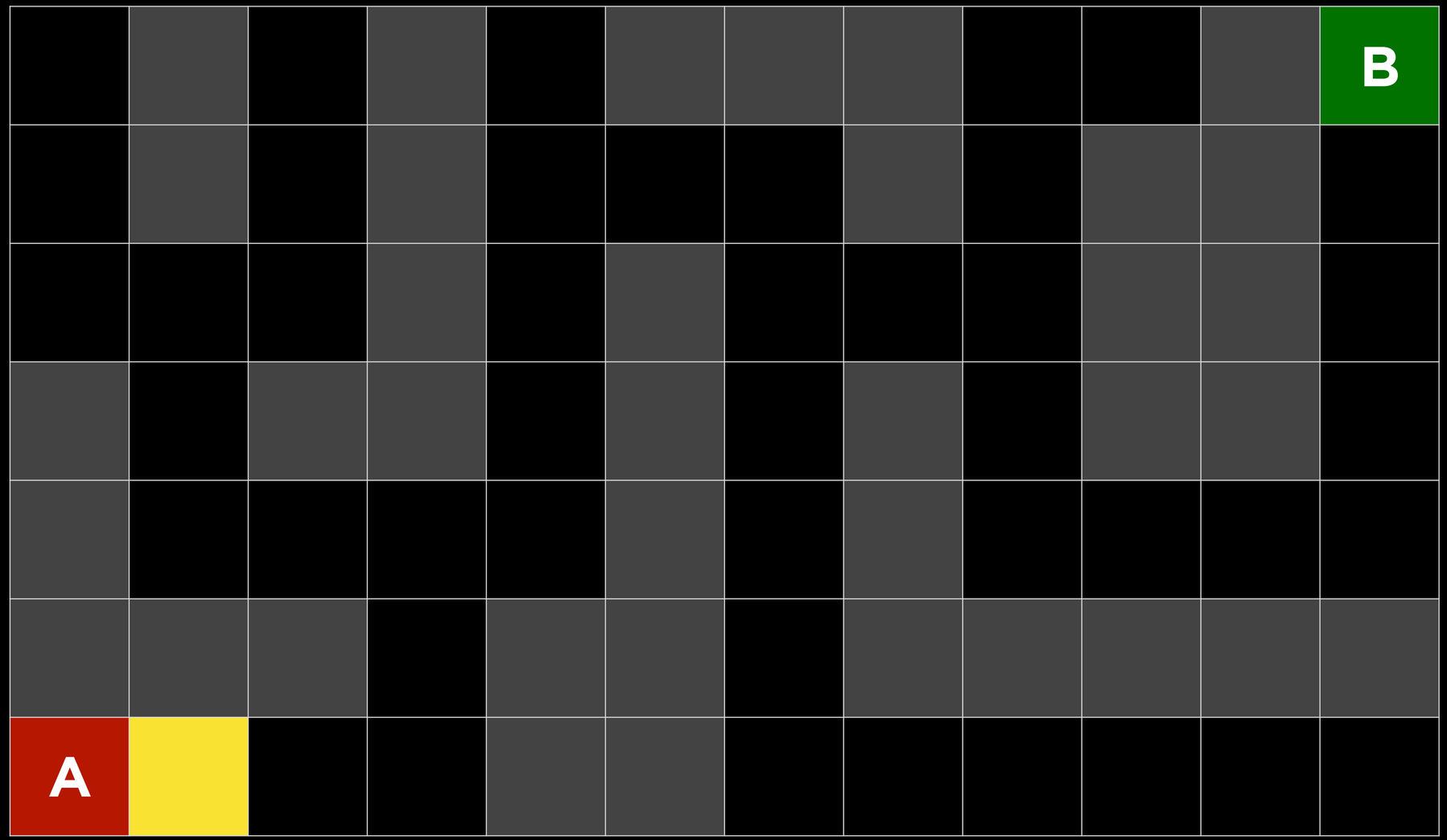
Breadth-First Search



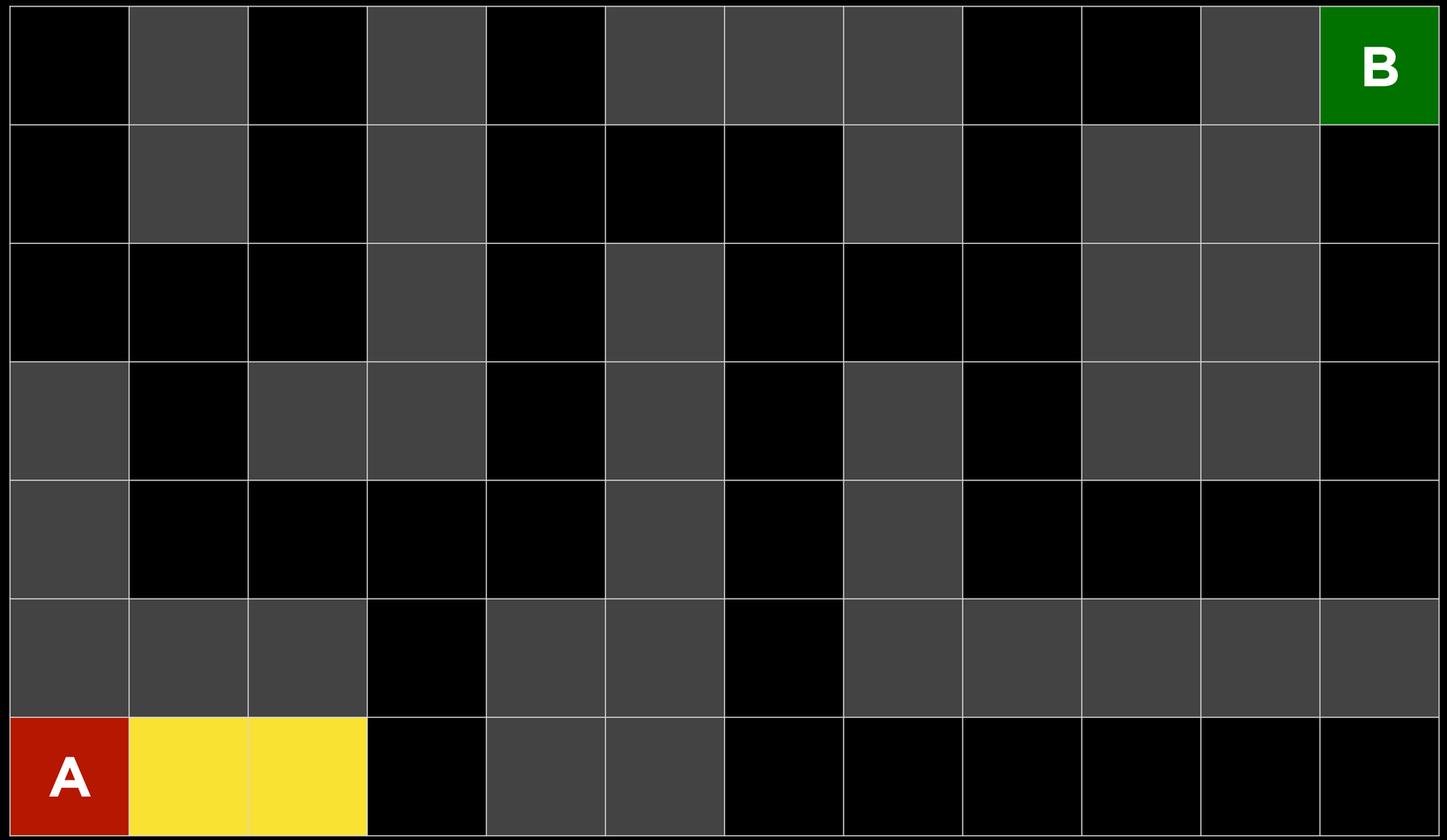
Breadth-First Search



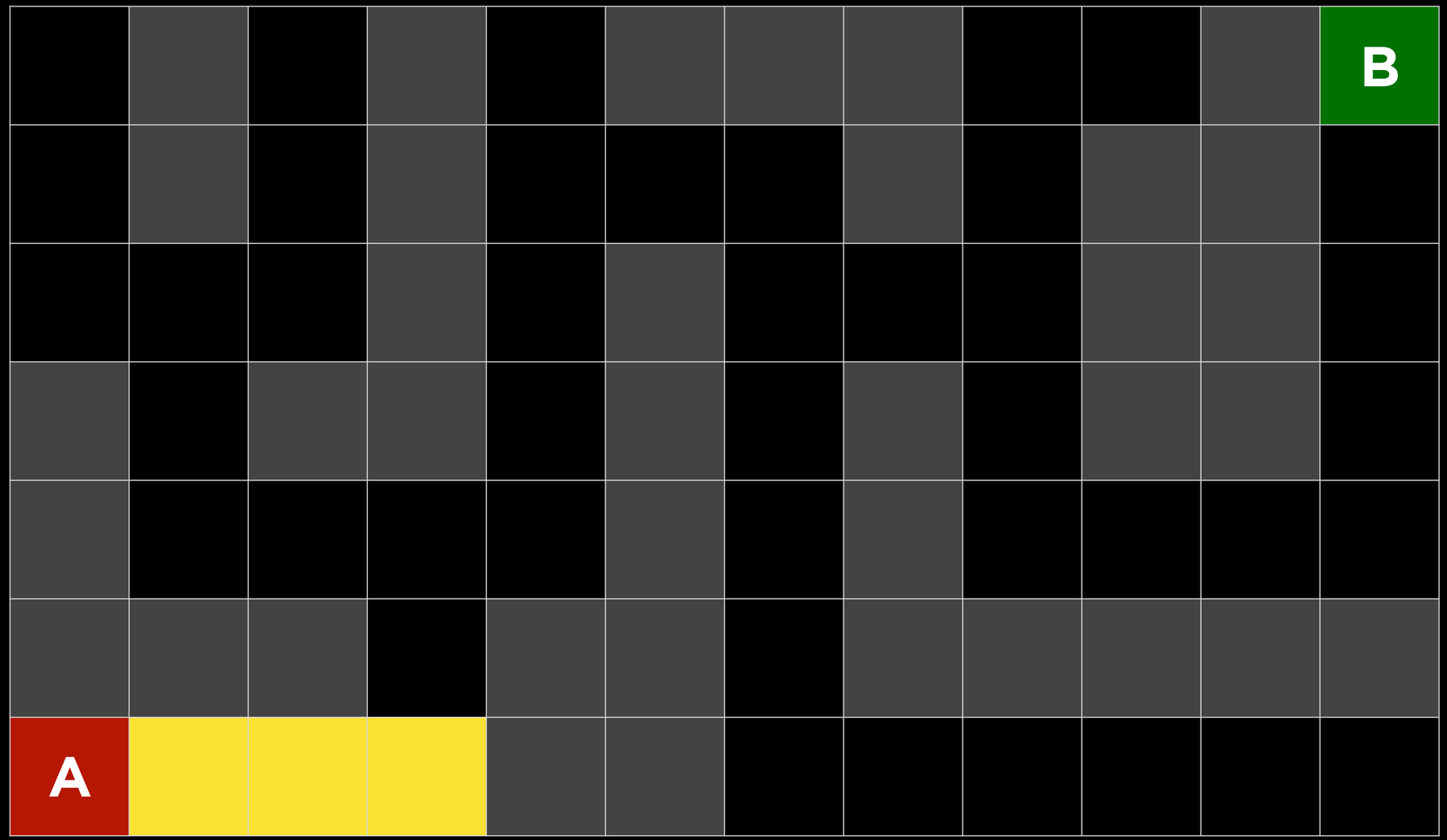
Breadth-First Search



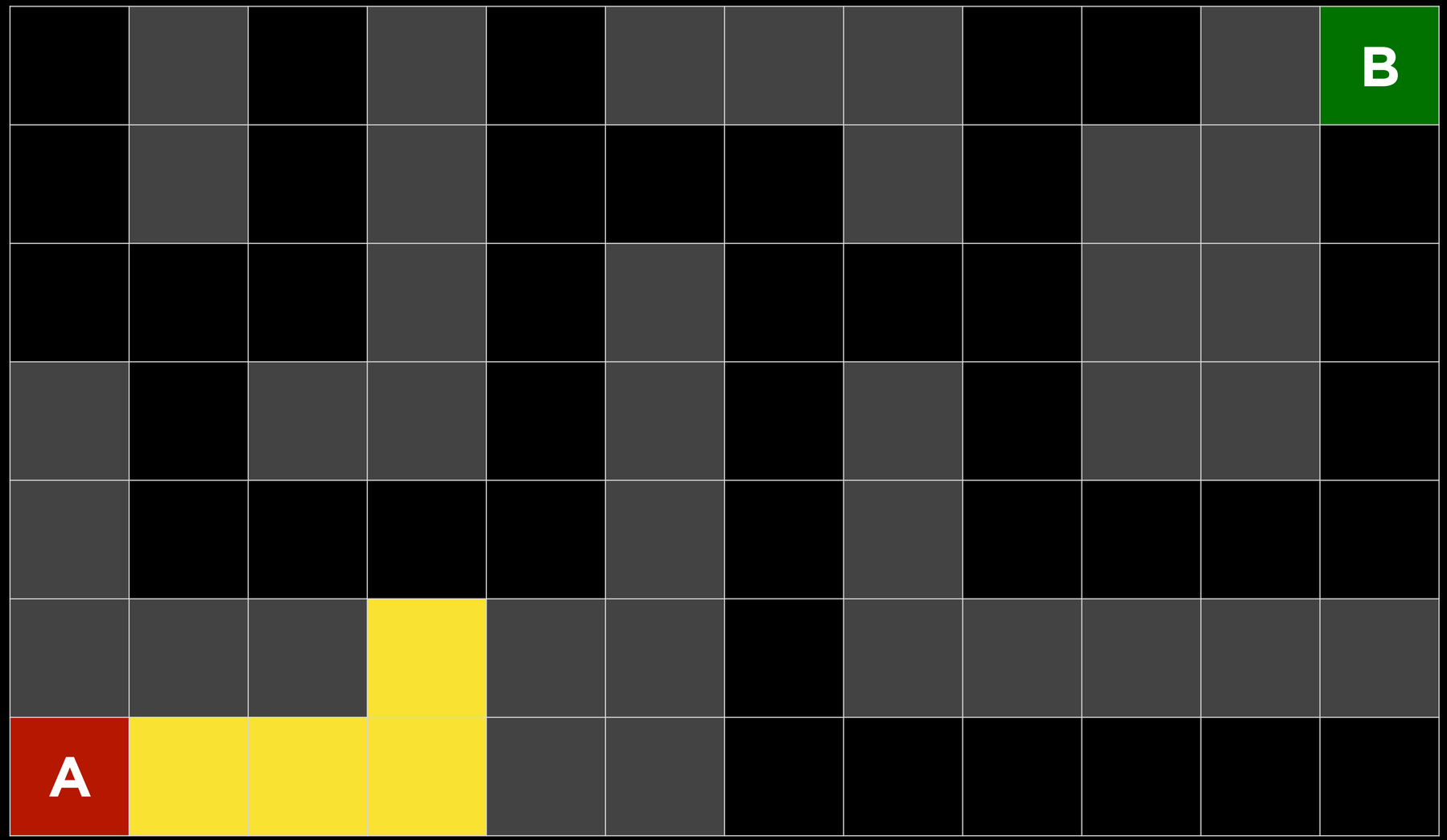
Breadth-First Search



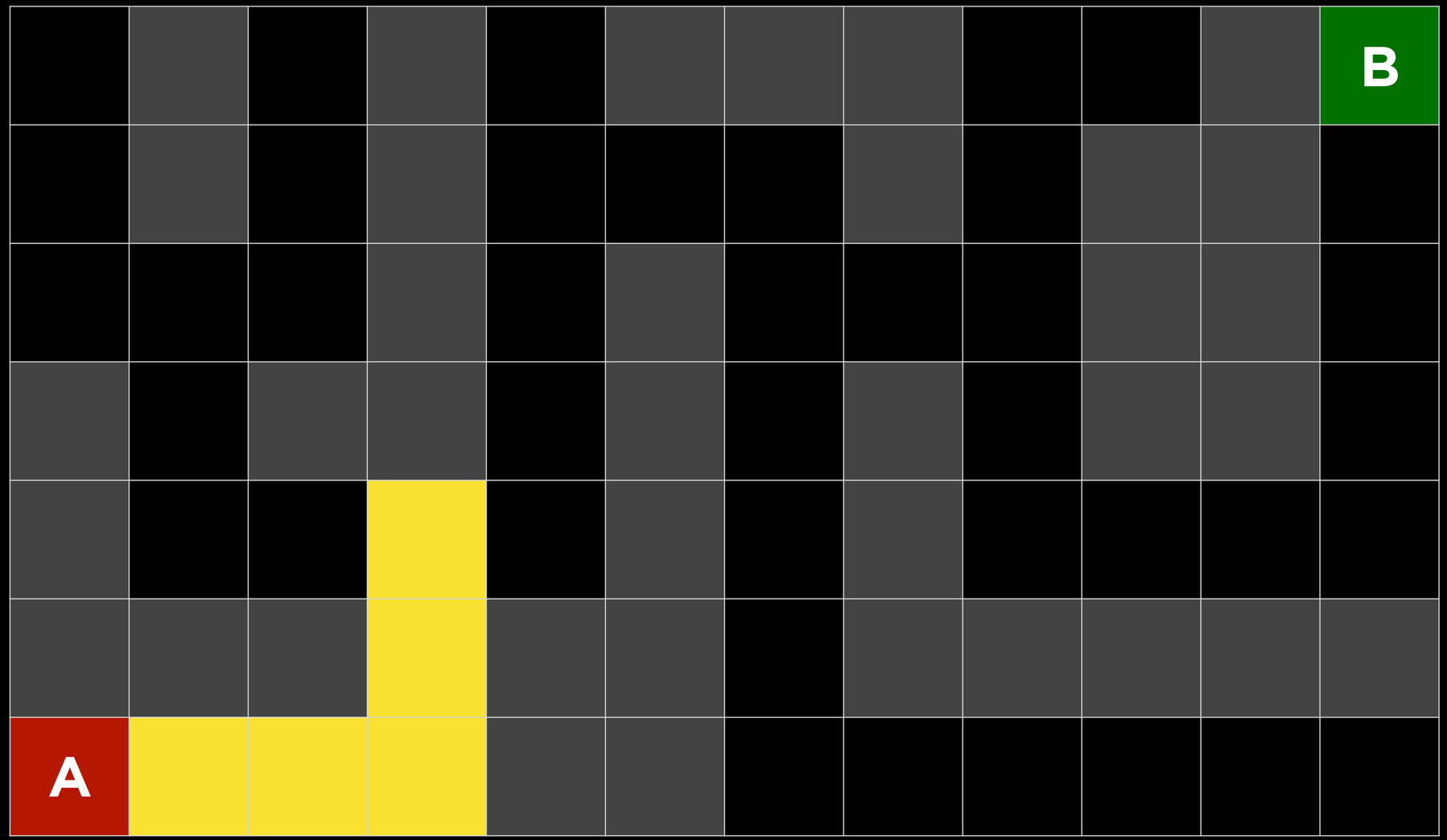
Breadth-First Search



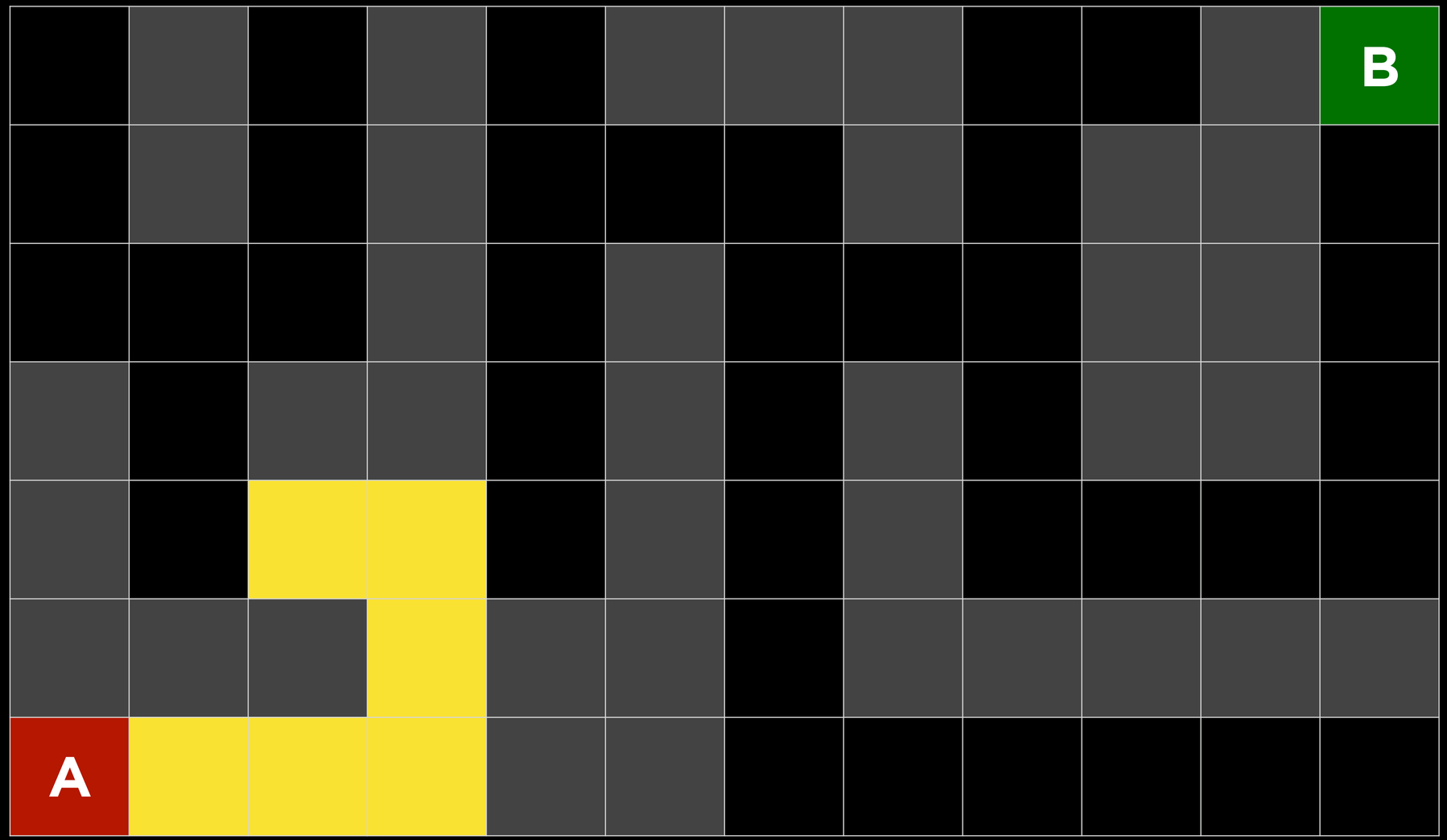
Breadth-First Search



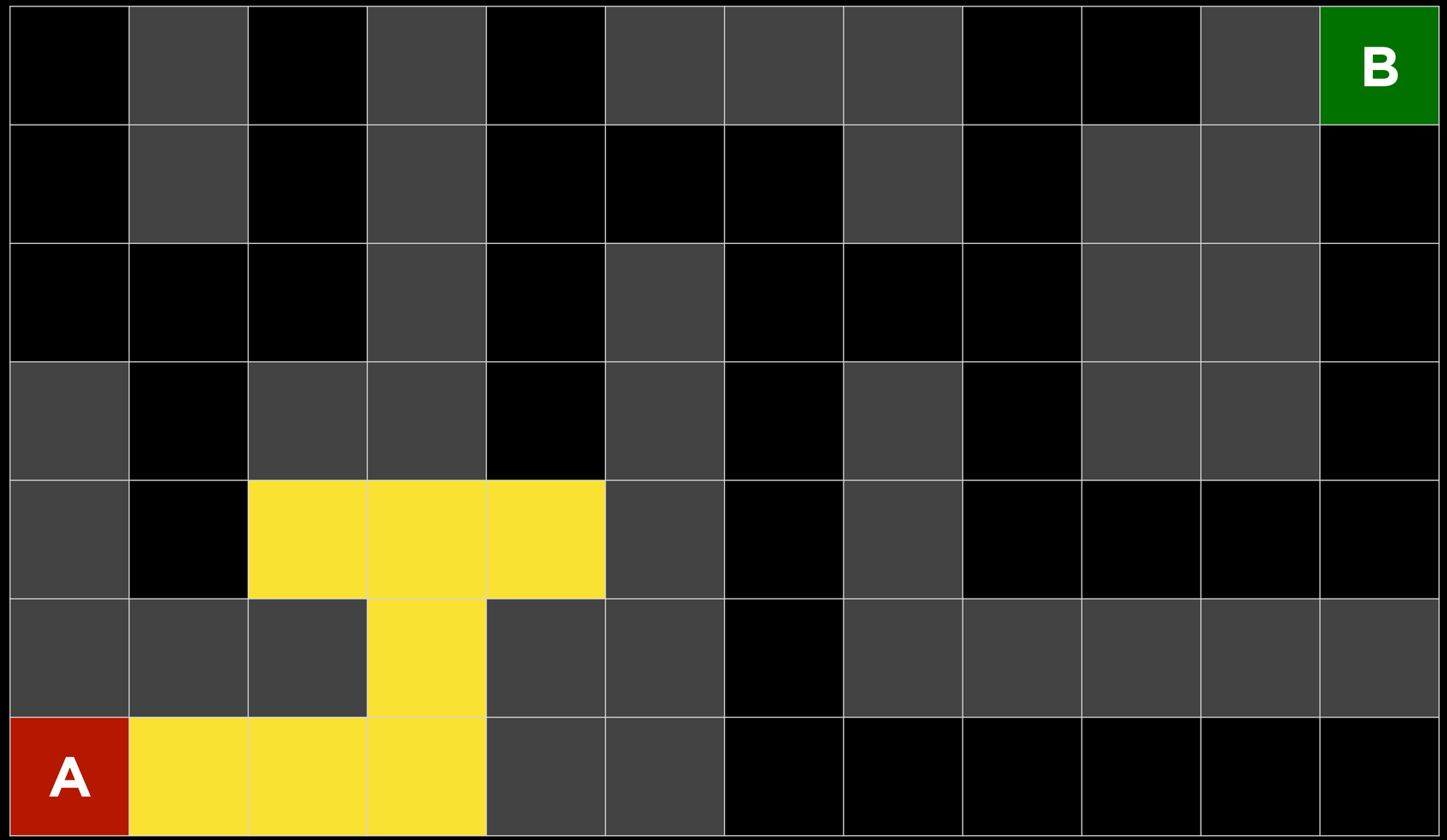
Breadth-First Search



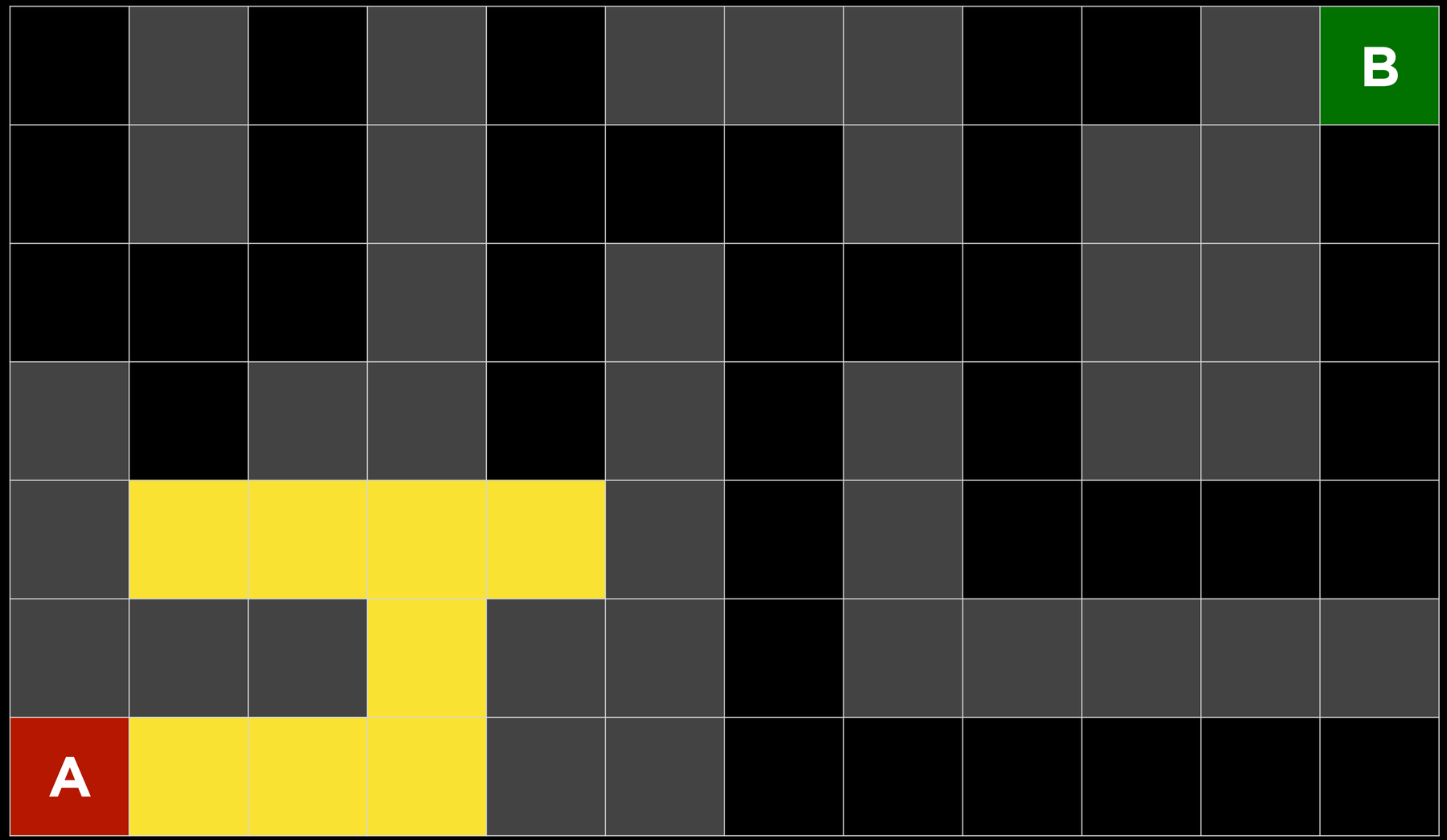
Breadth-First Search



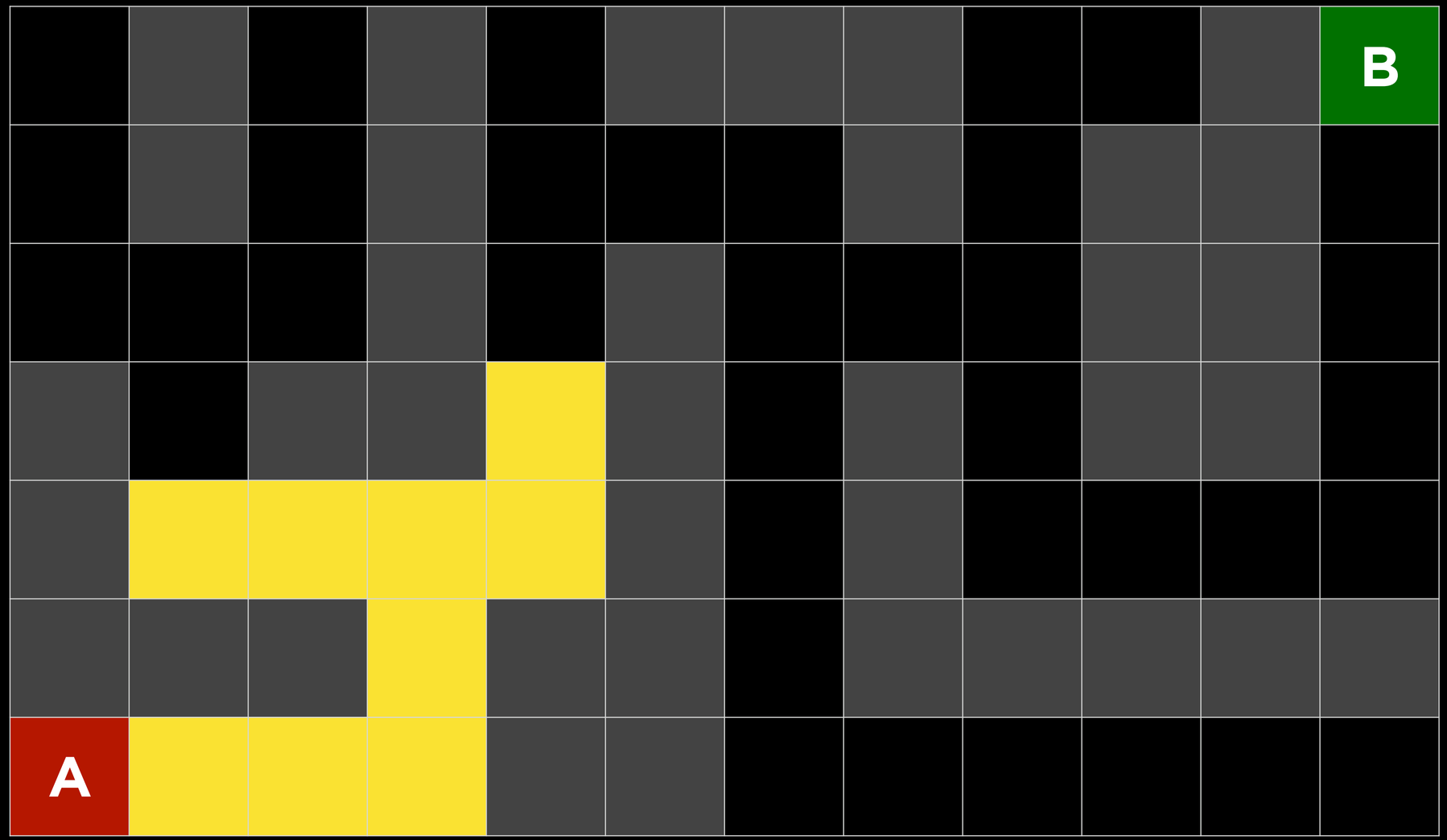
Breadth-First Search



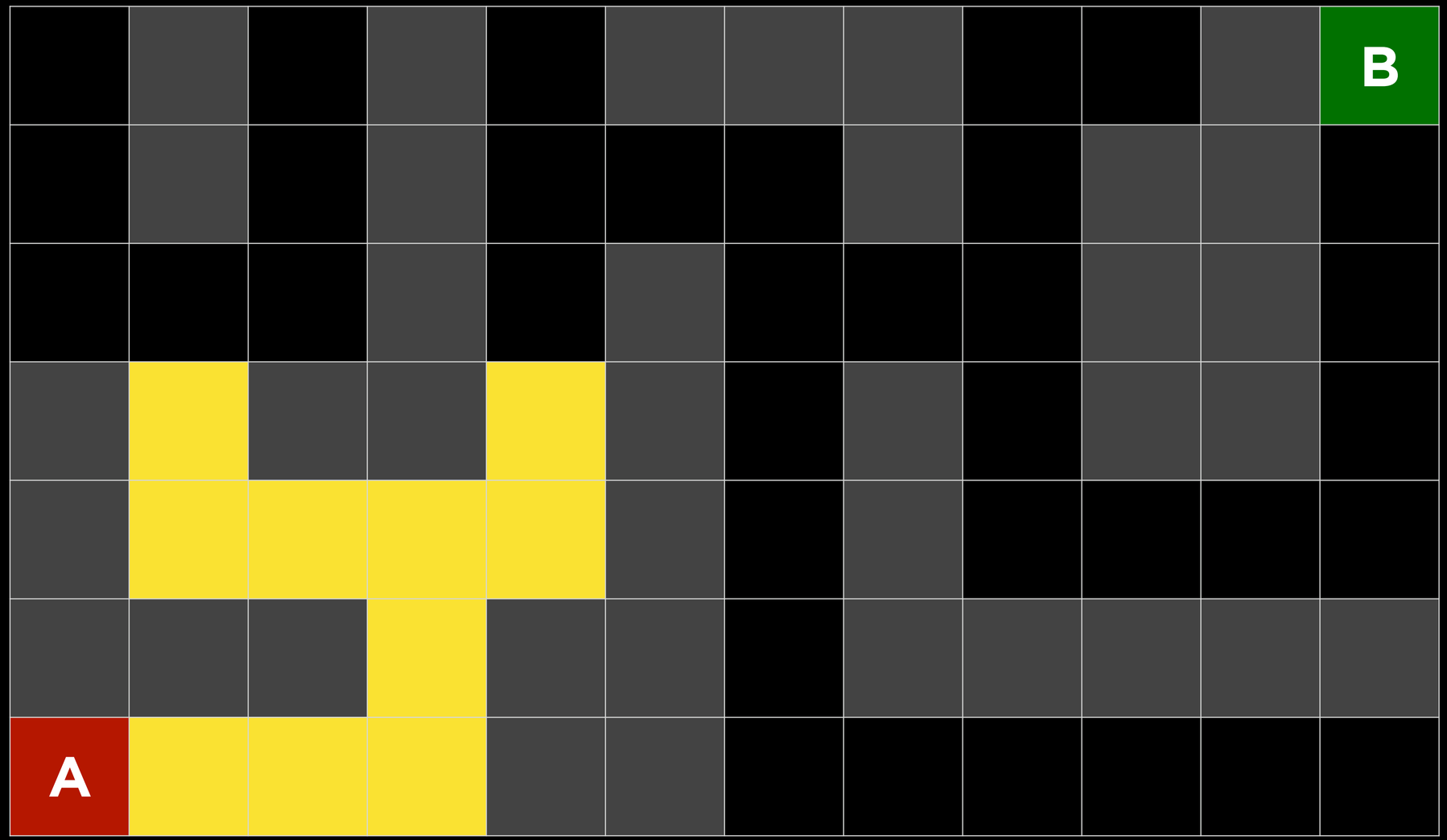
Breadth-First Search



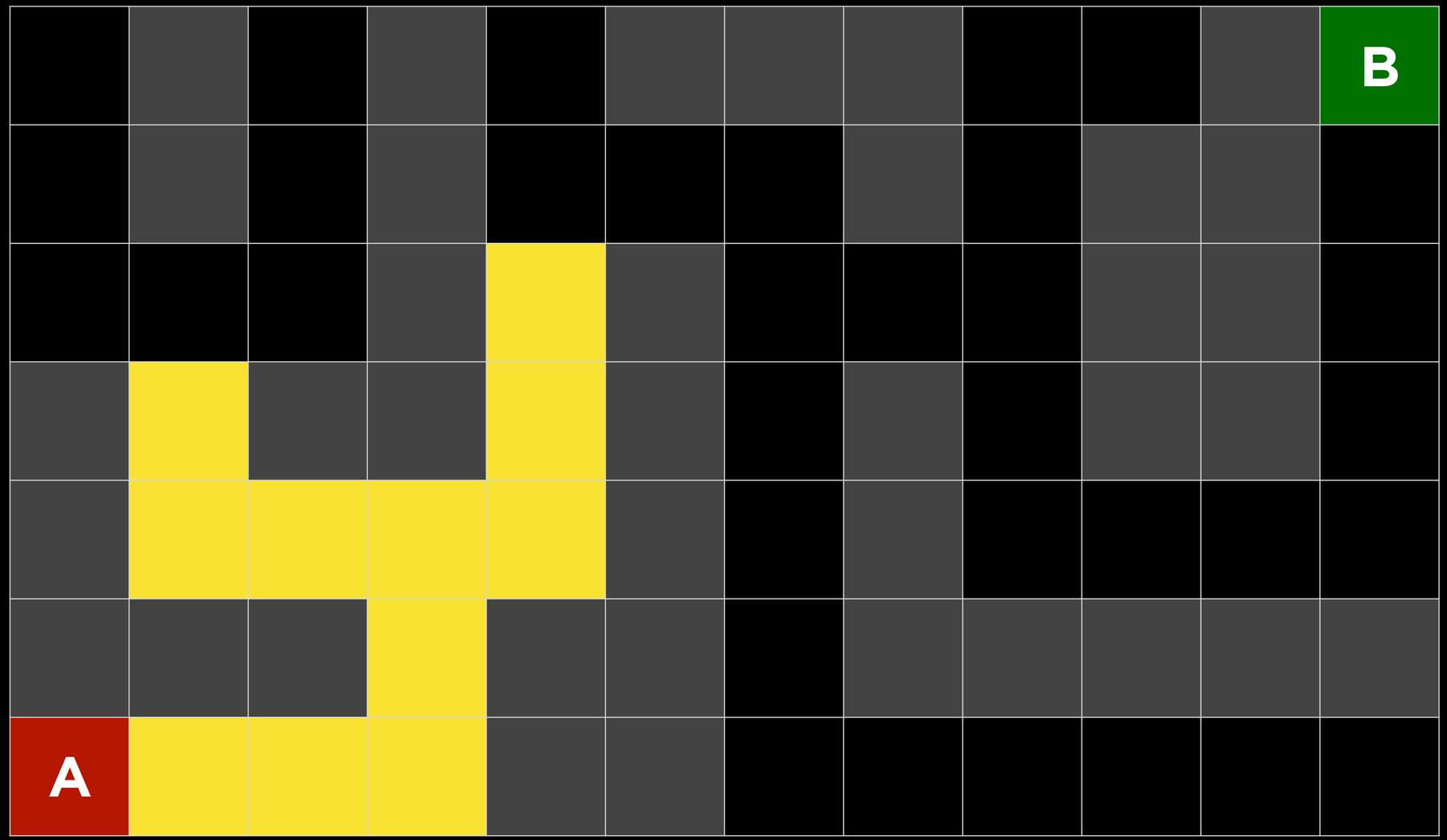
Breadth-First Search



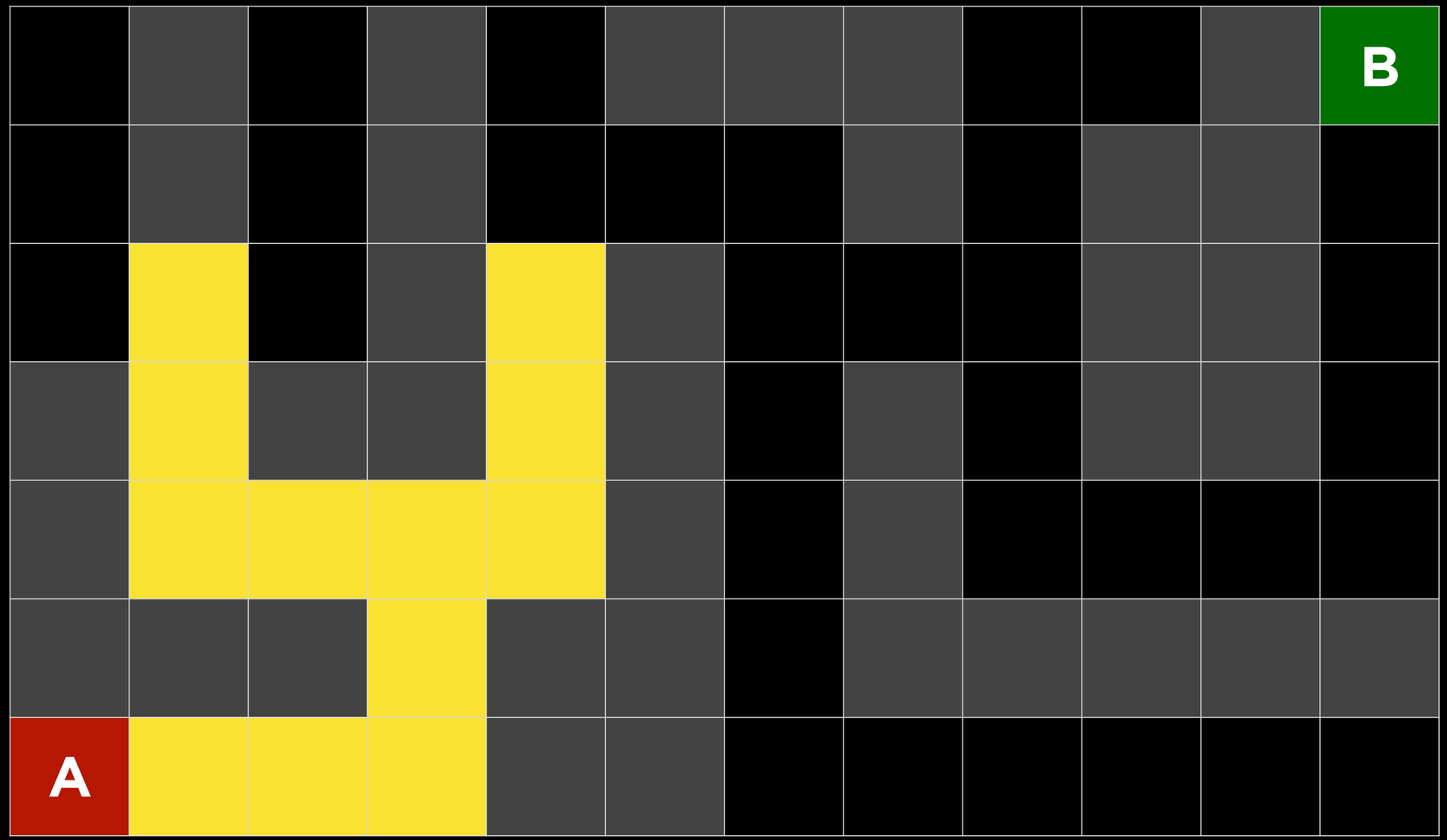
Breadth-First Search



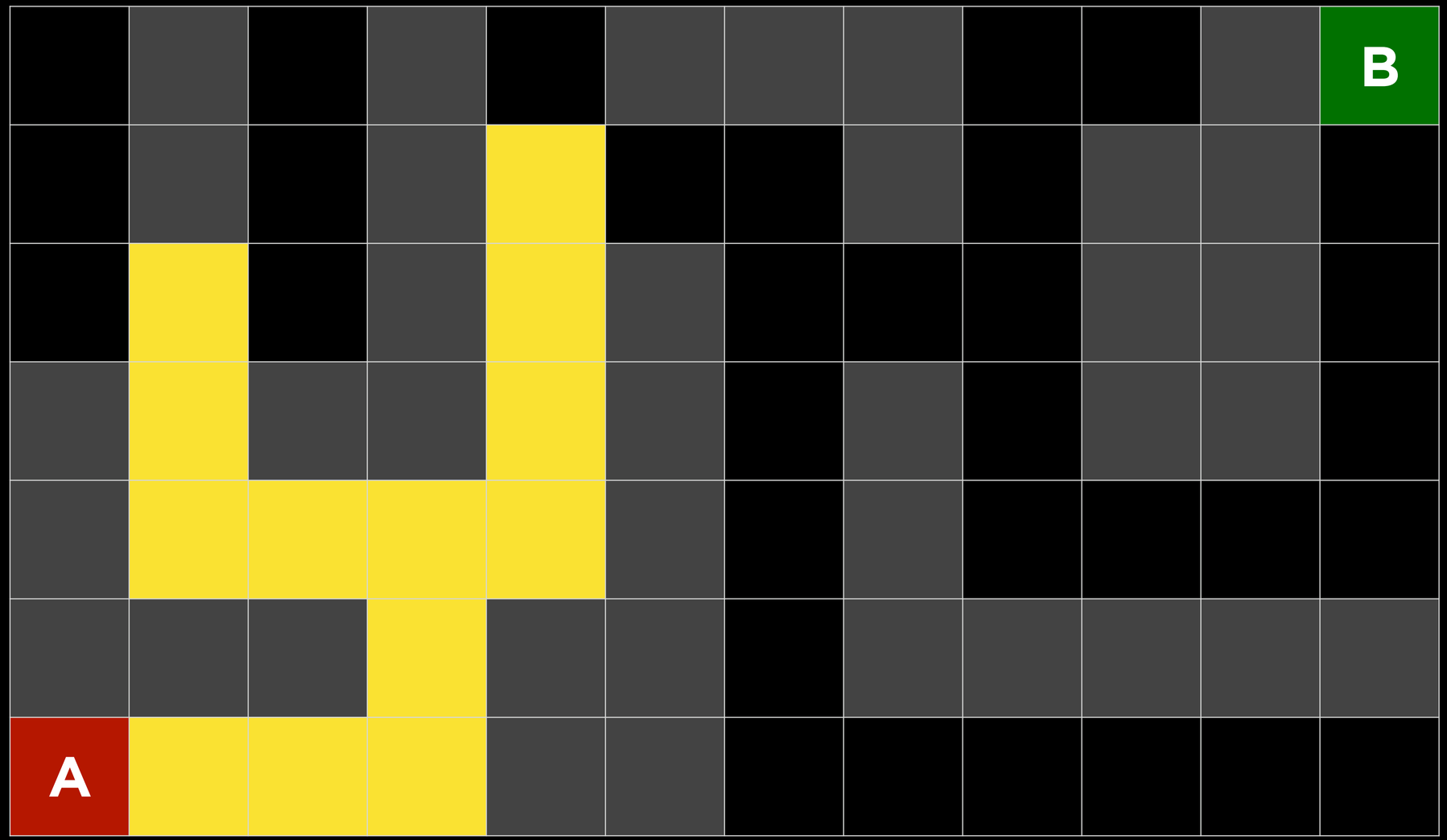
Breadth-First Search



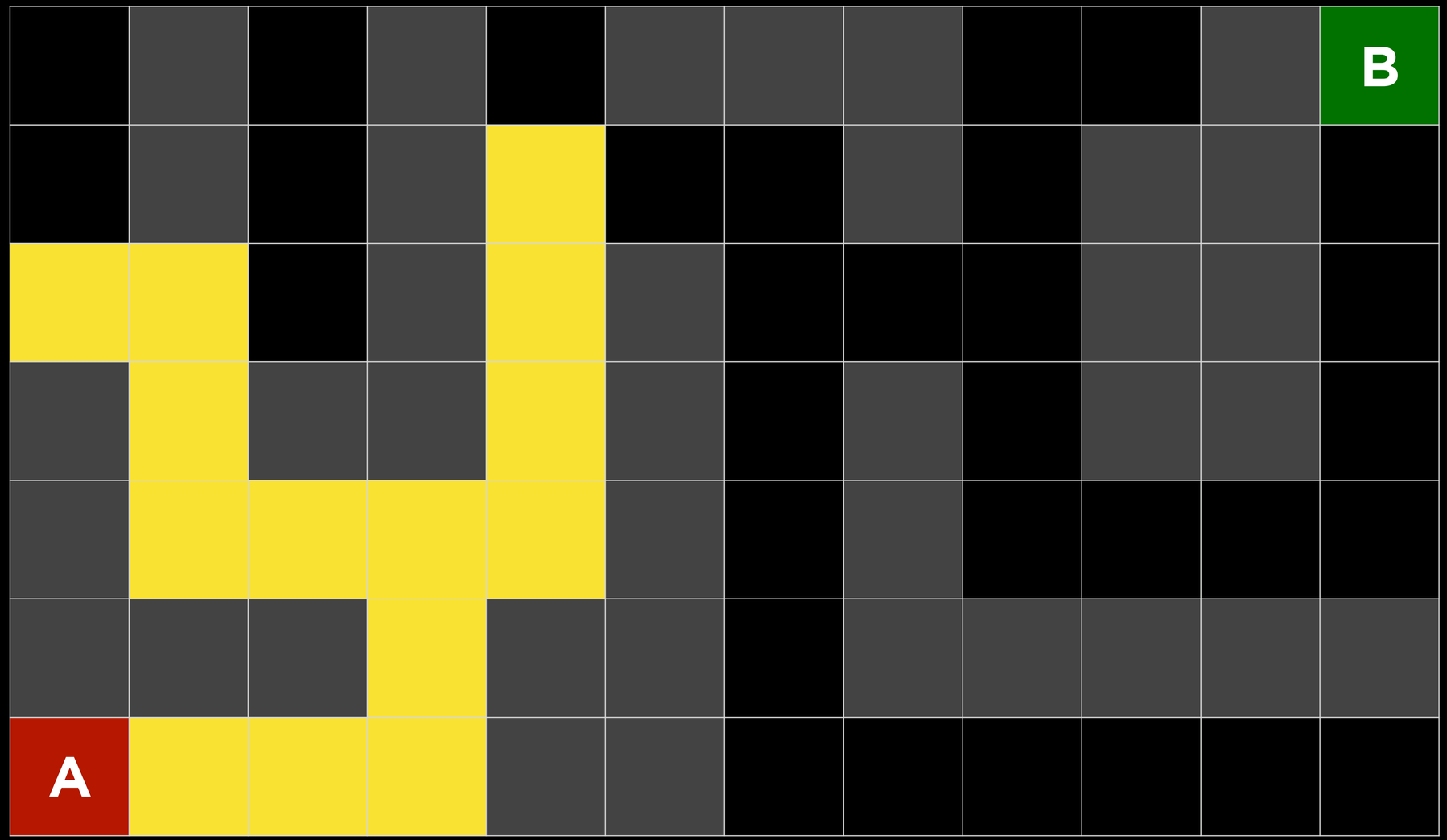
Breadth-First Search



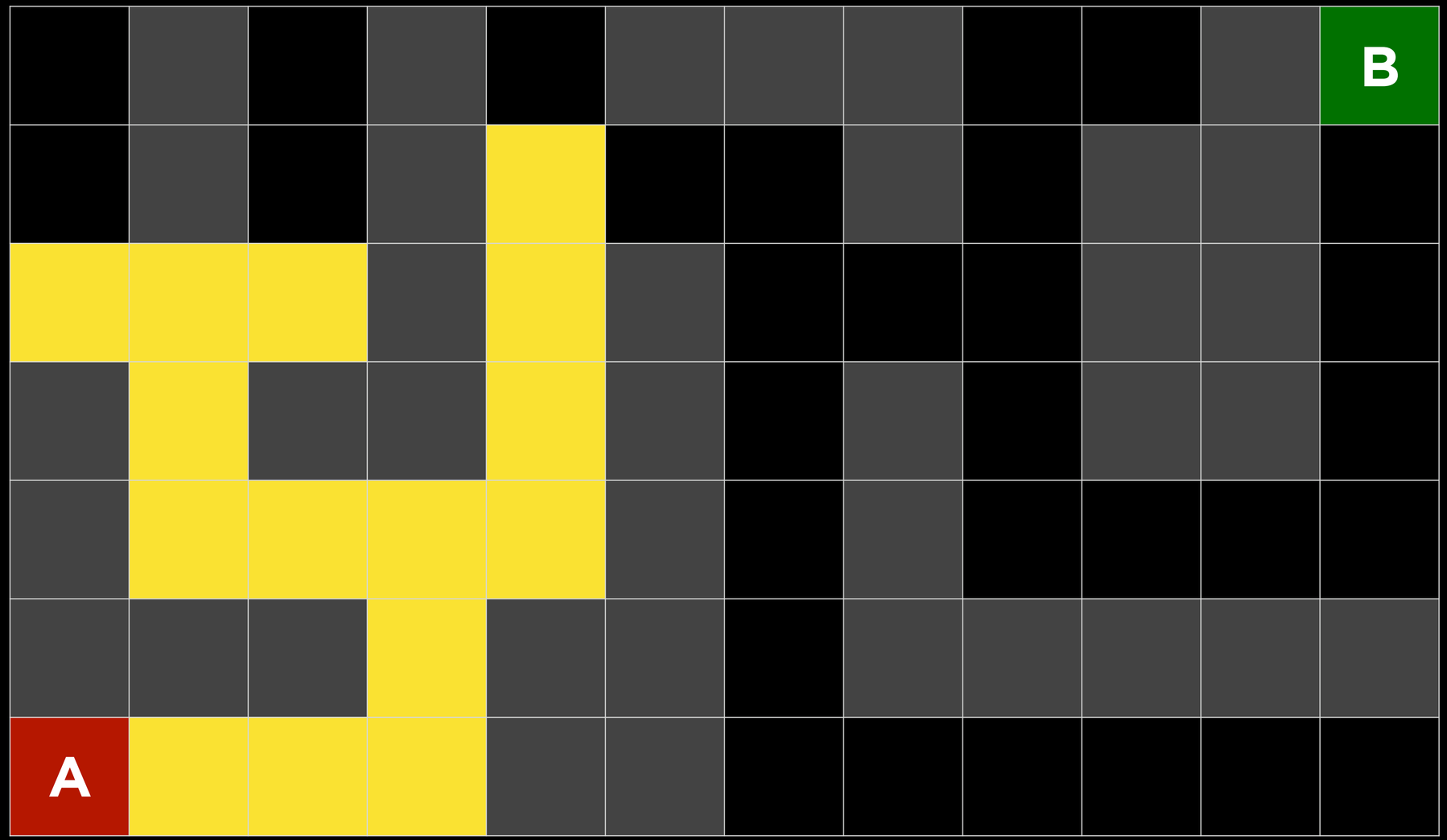
Breadth-First Search



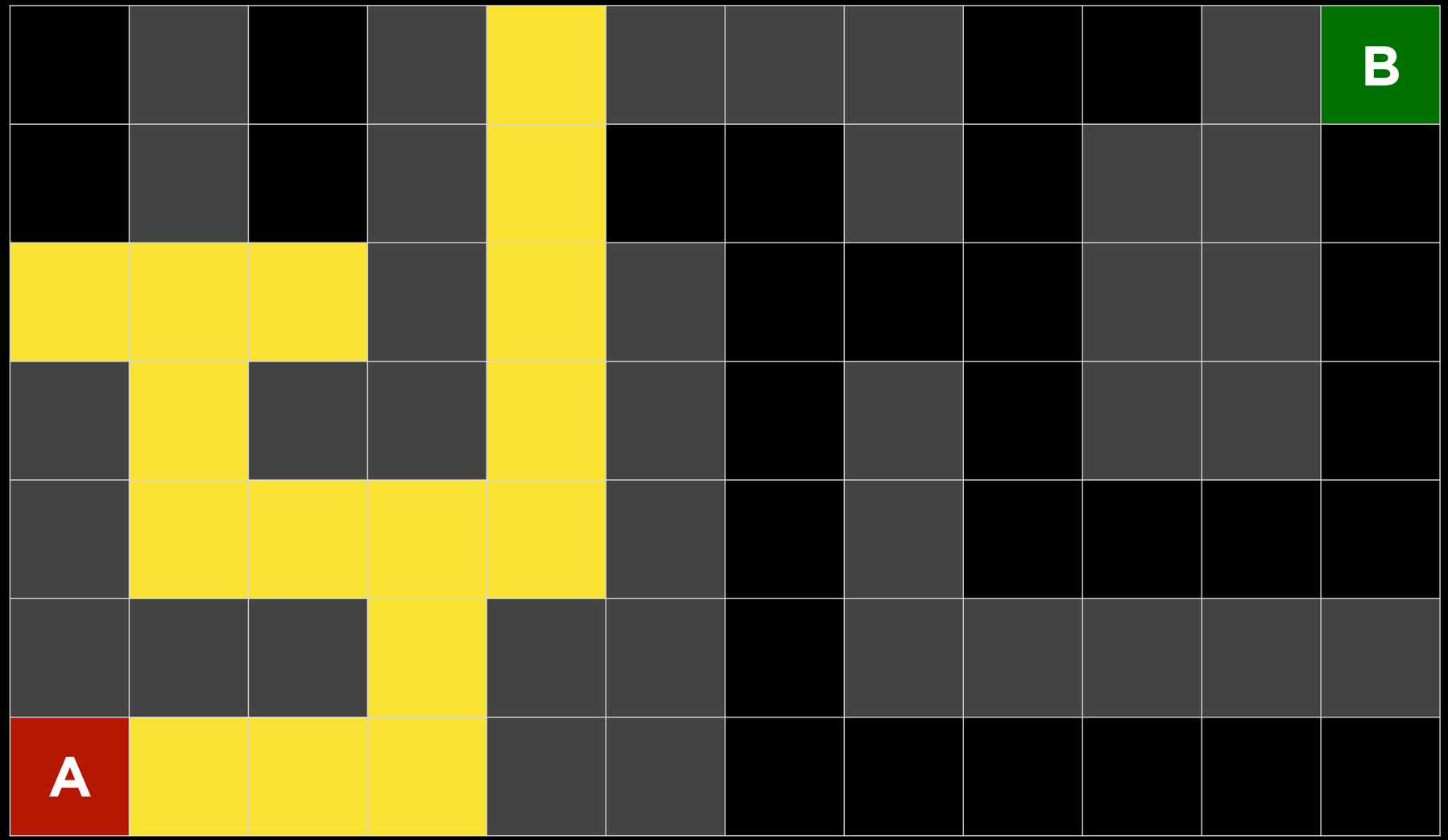
Breadth-First Search



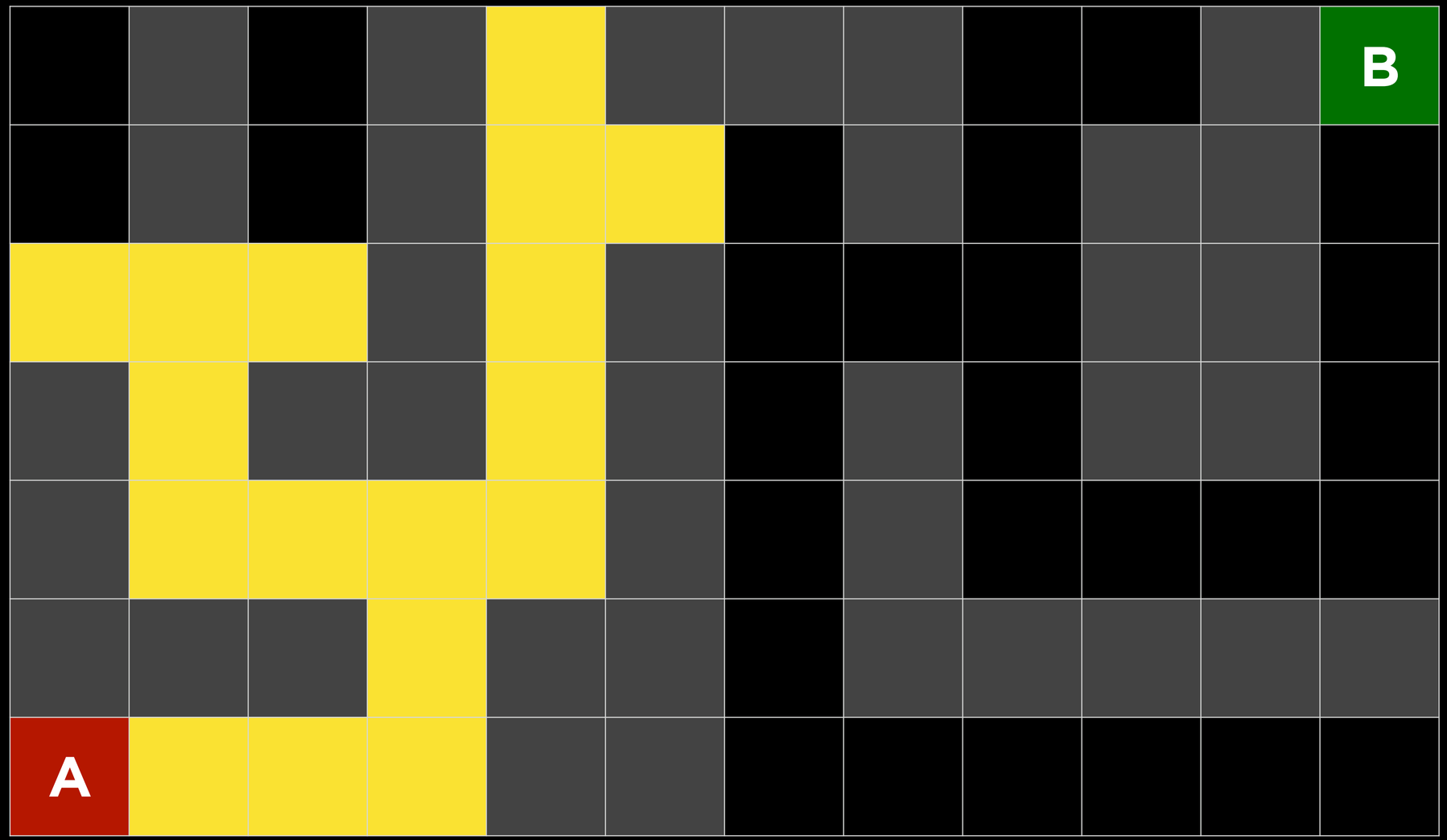
Breadth-First Search



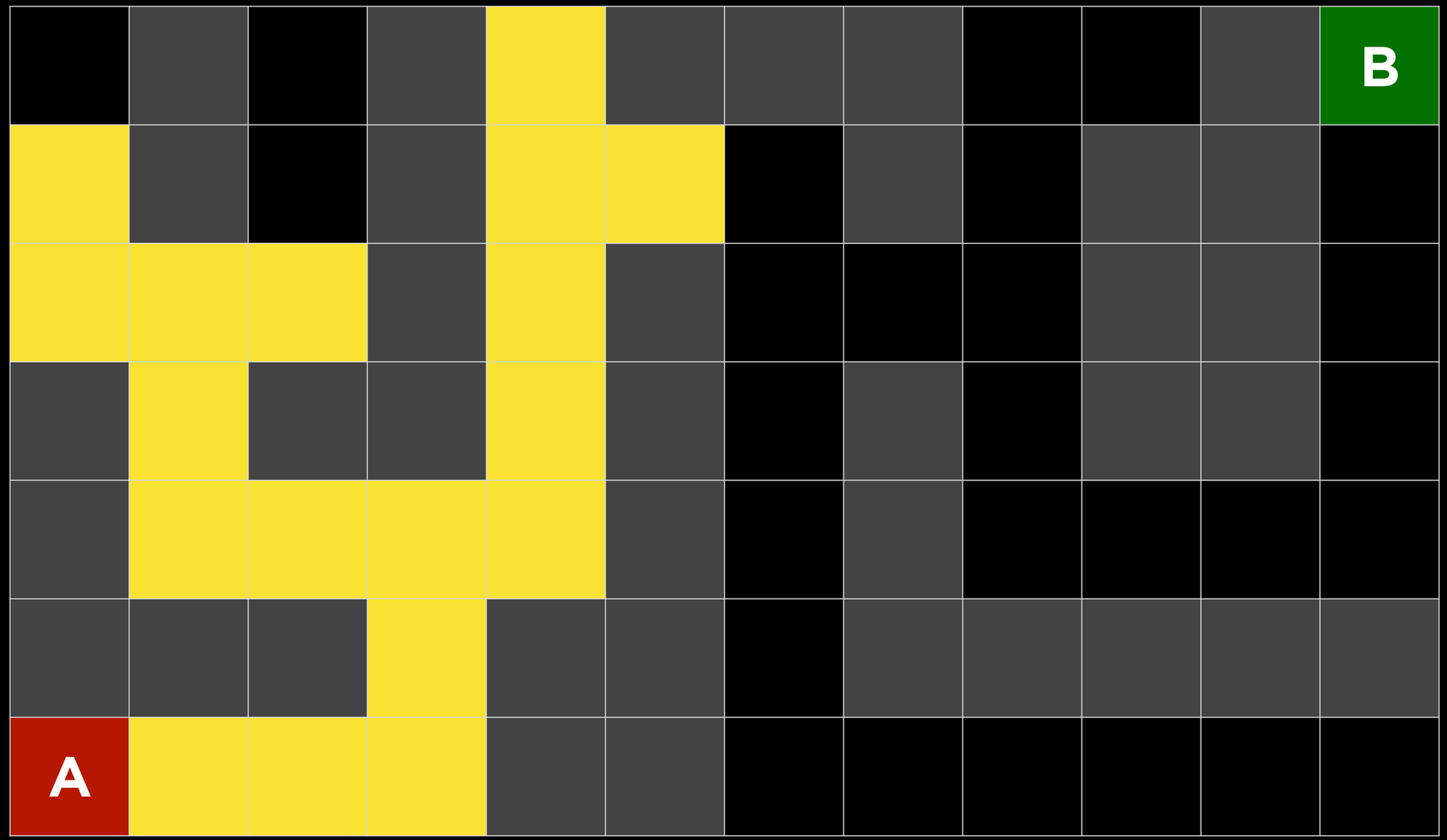
Breadth-First Search



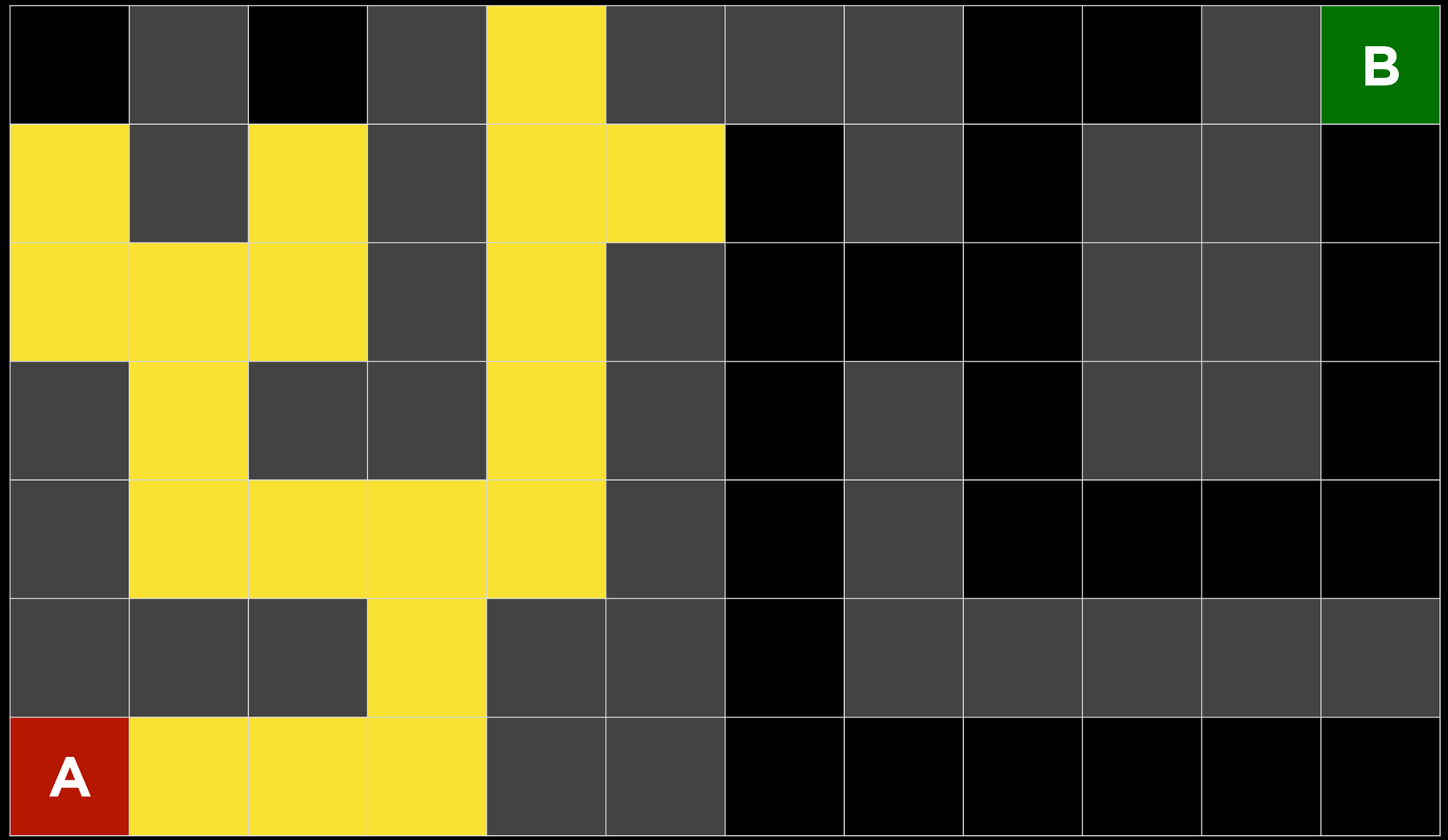
Breadth-First Search



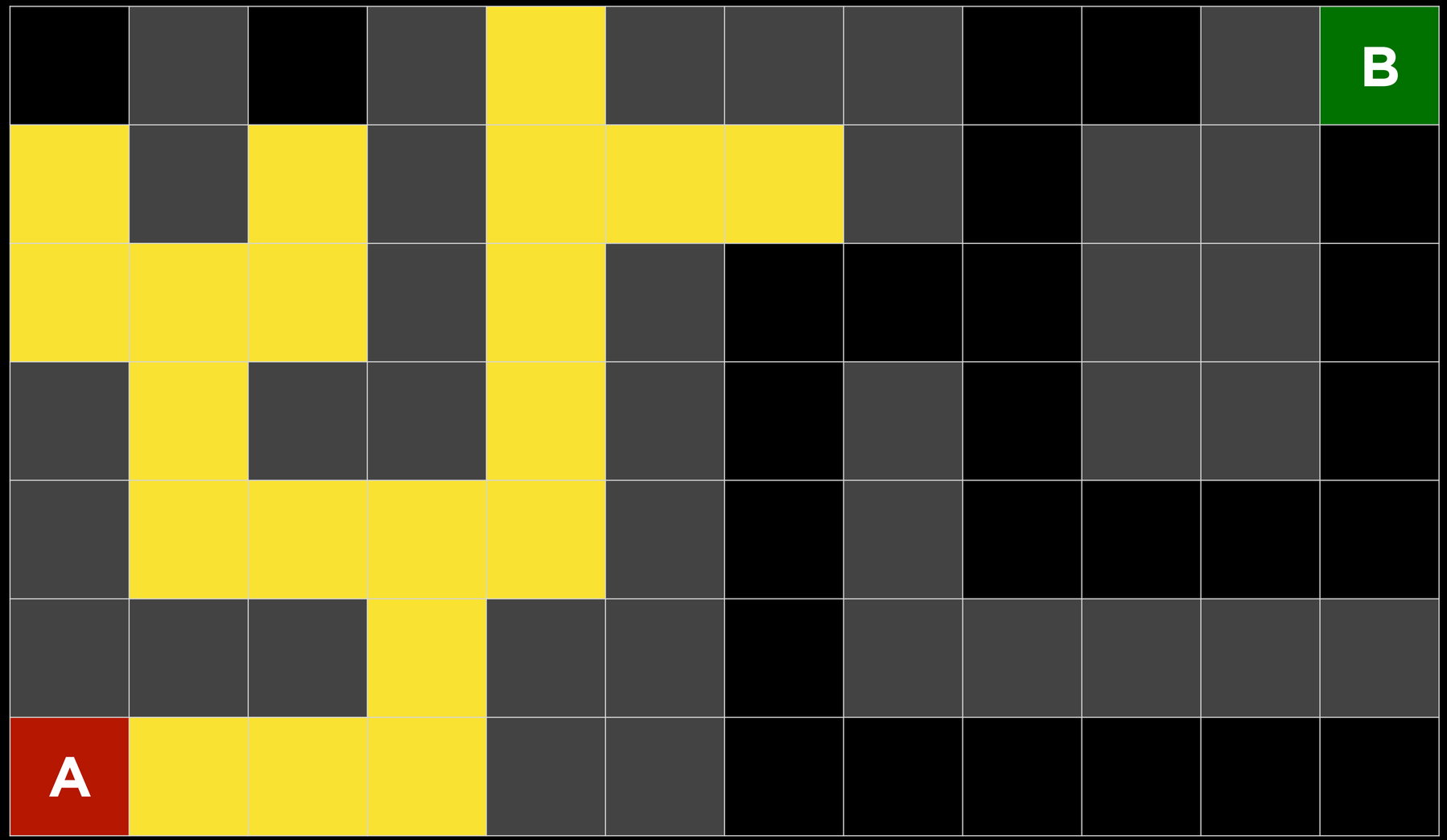
Breadth-First Search



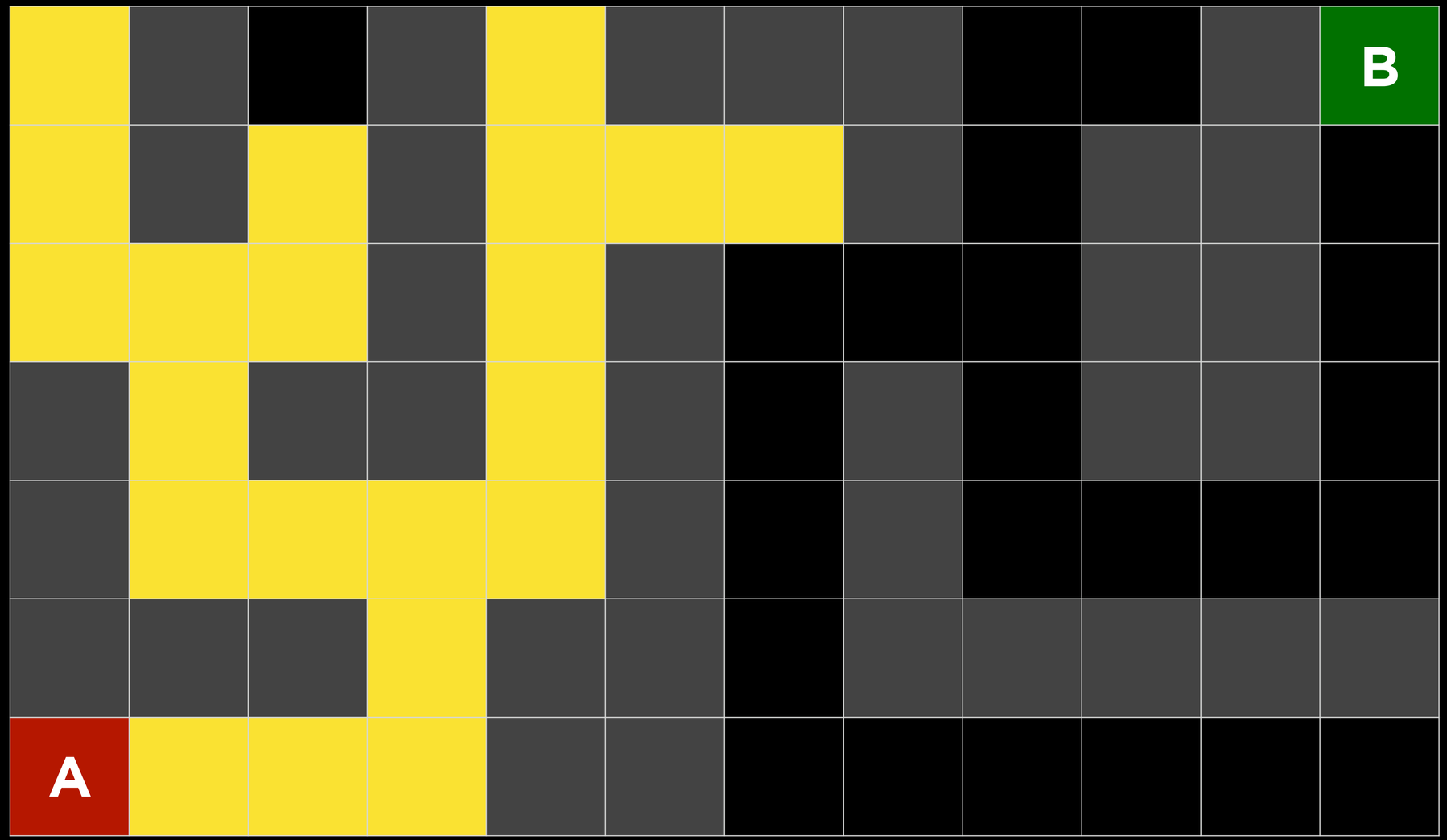
Breadth-First Search



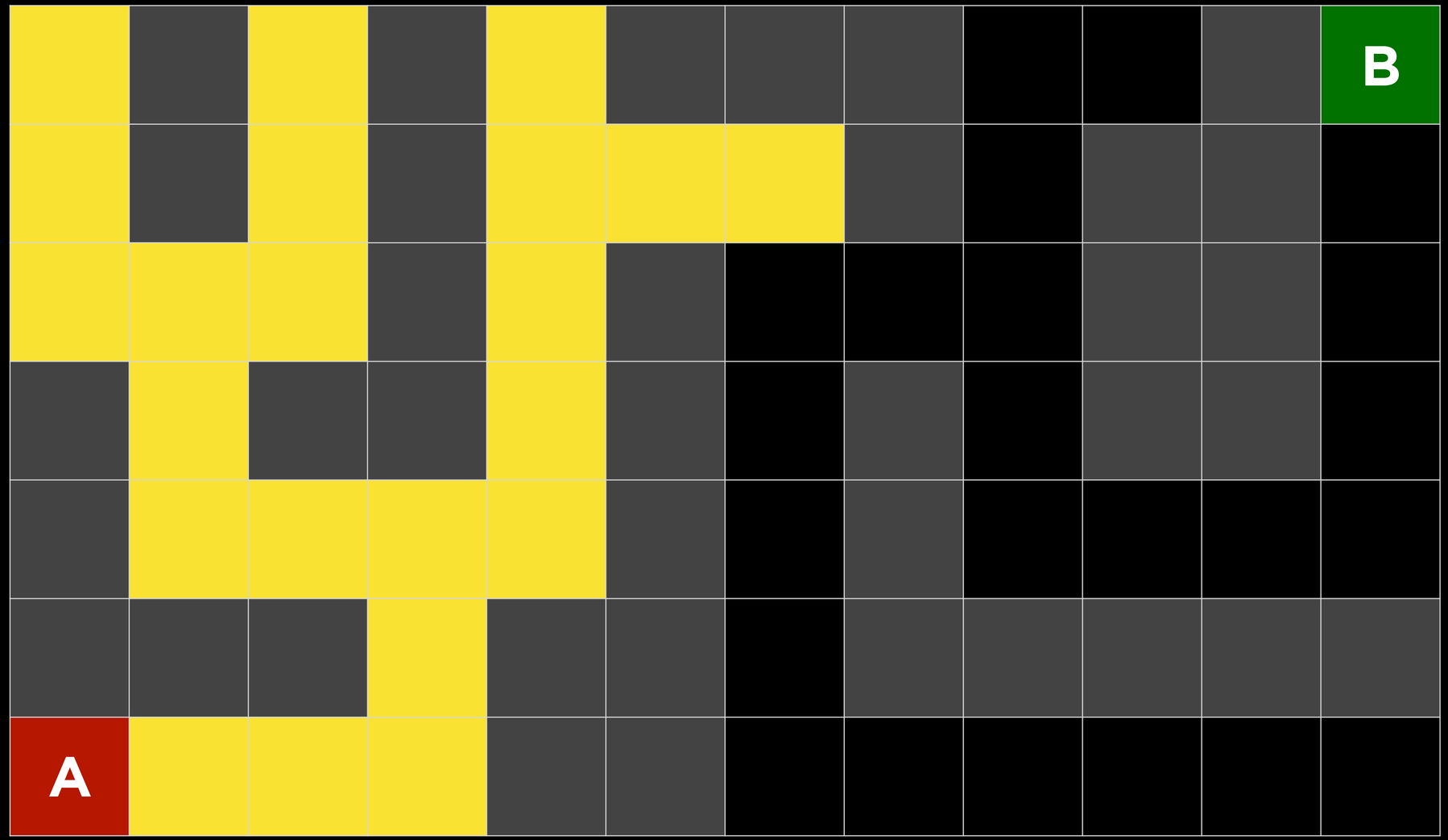
Breadth-First Search



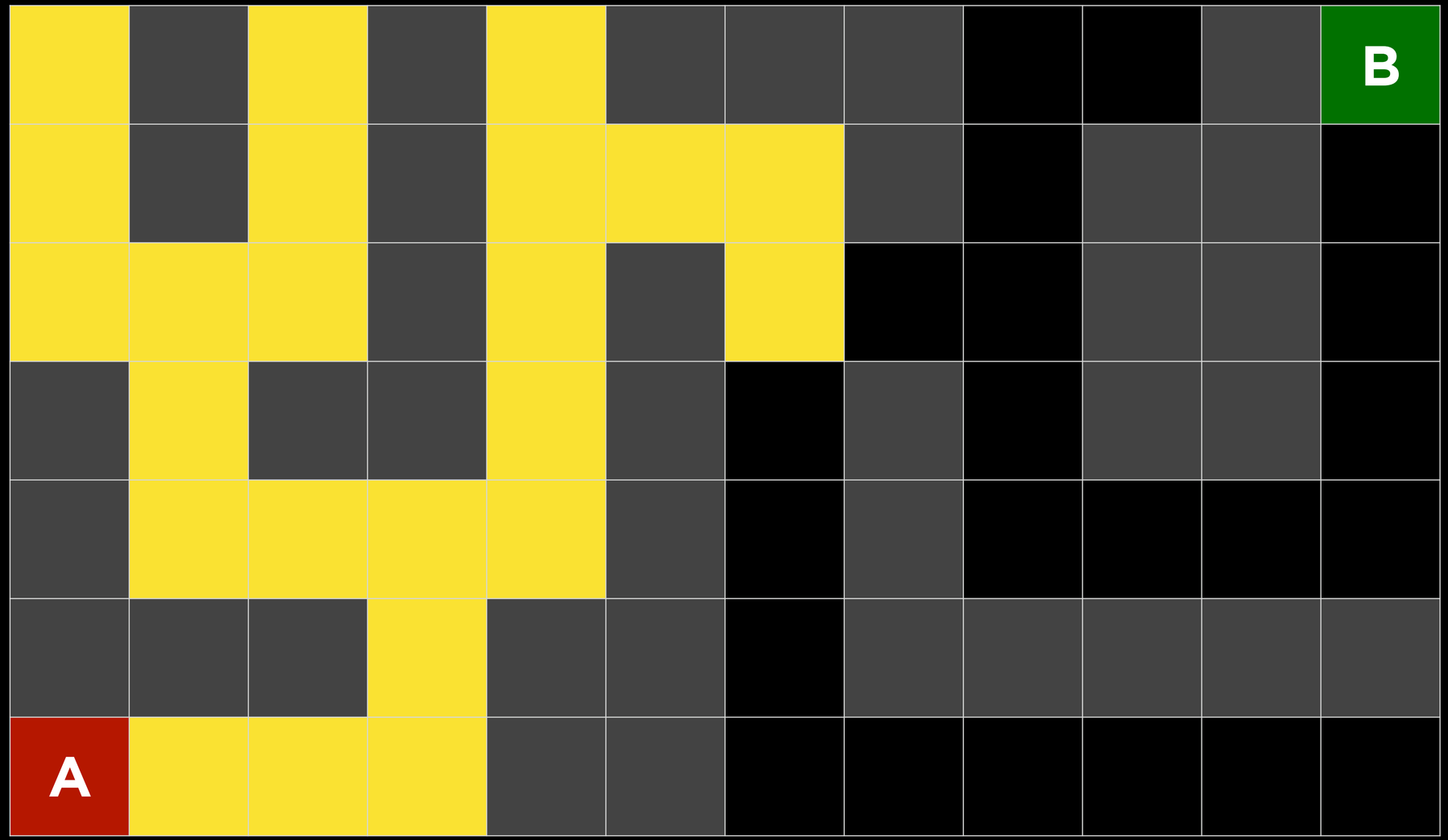
Breadth-First Search



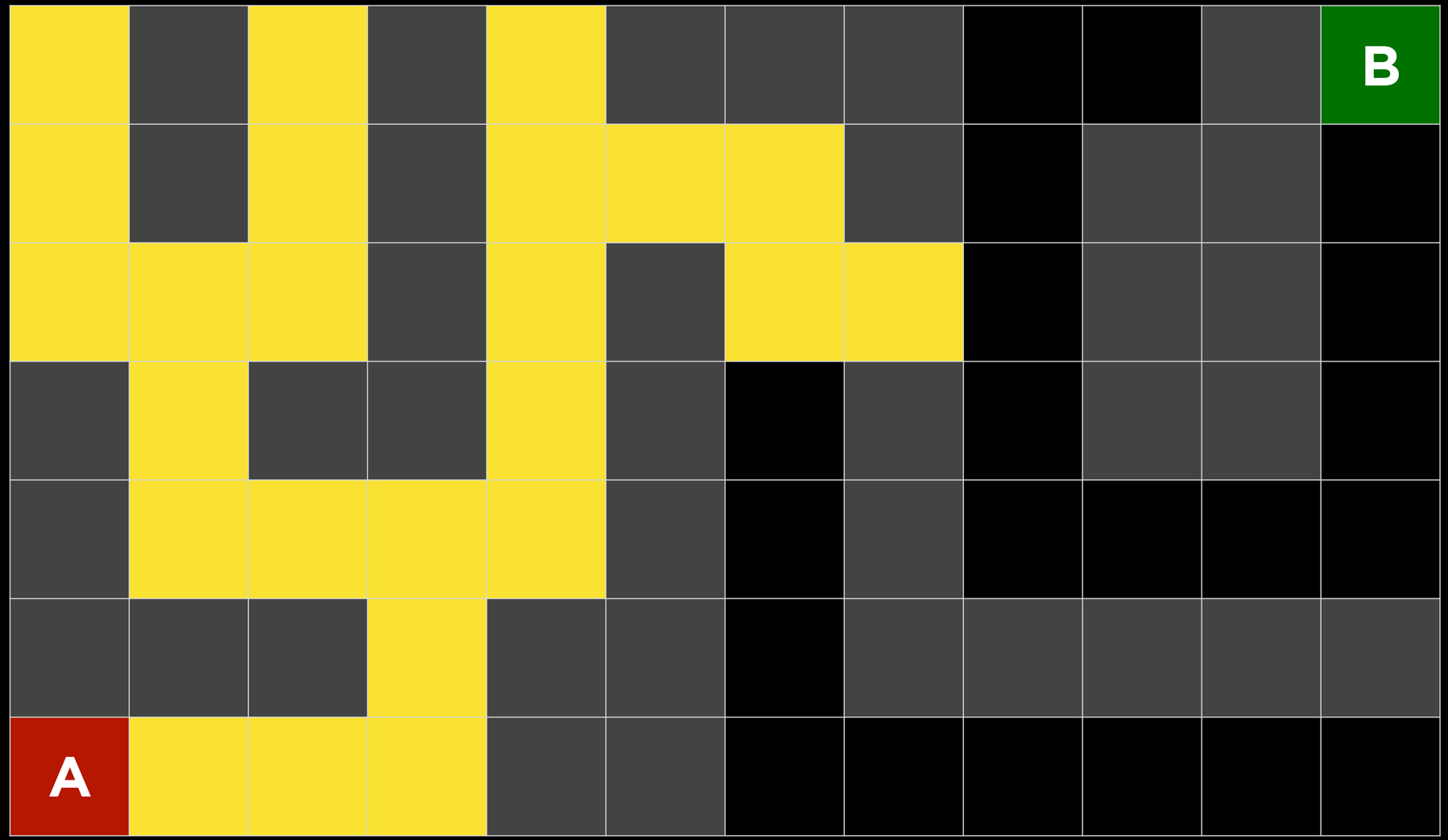
Breadth-First Search



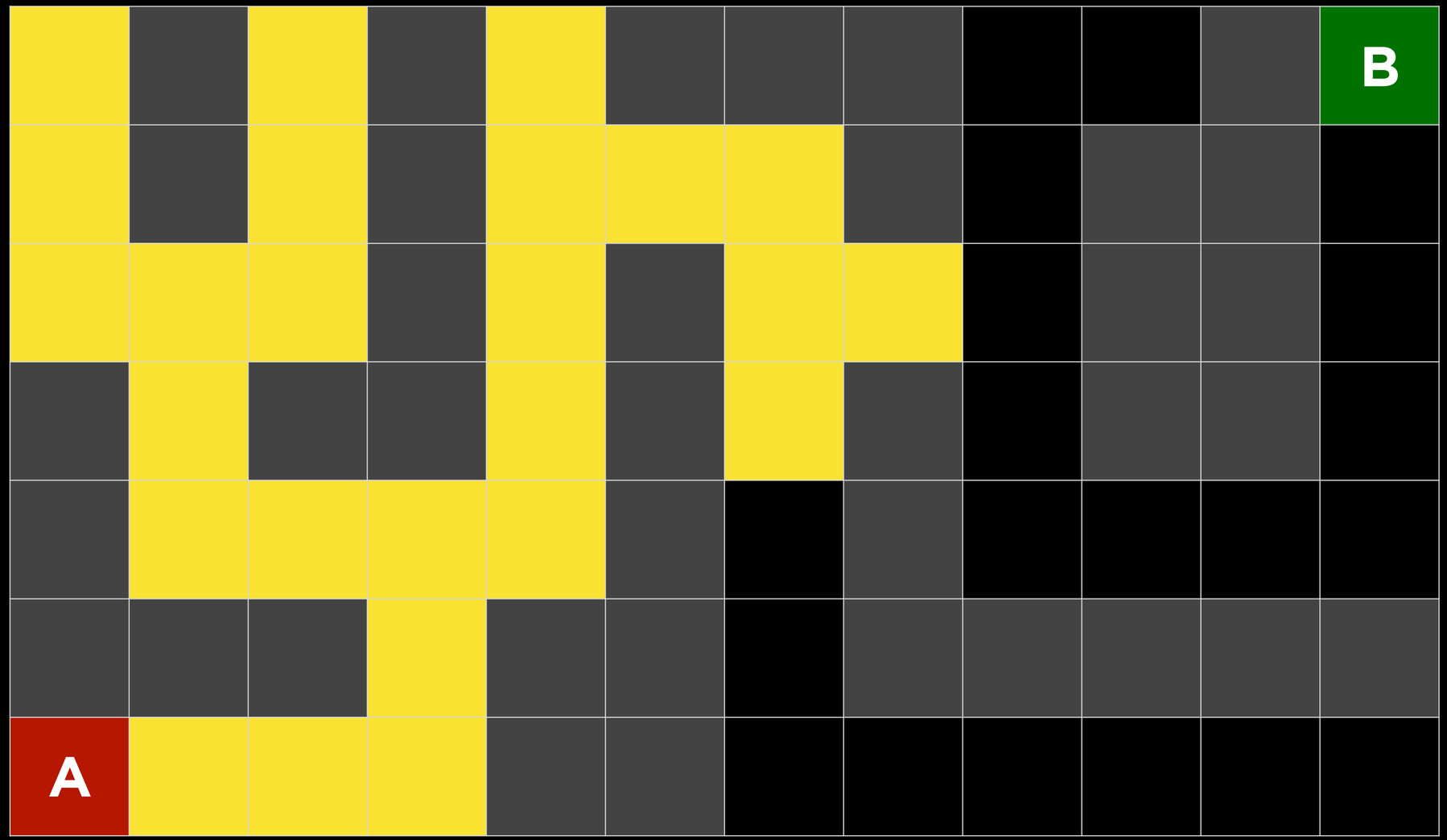
Breadth-First Search



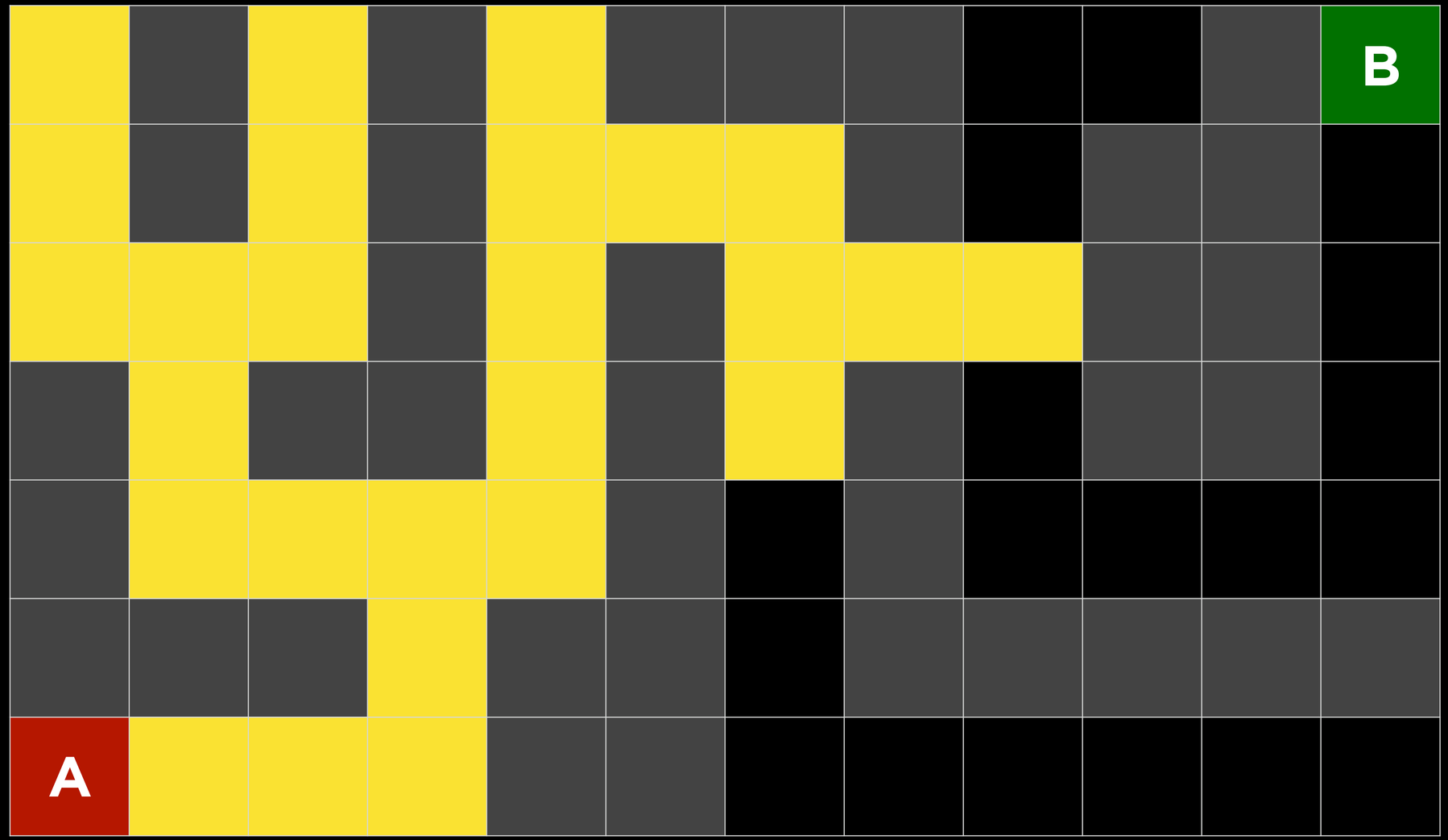
Breadth-First Search



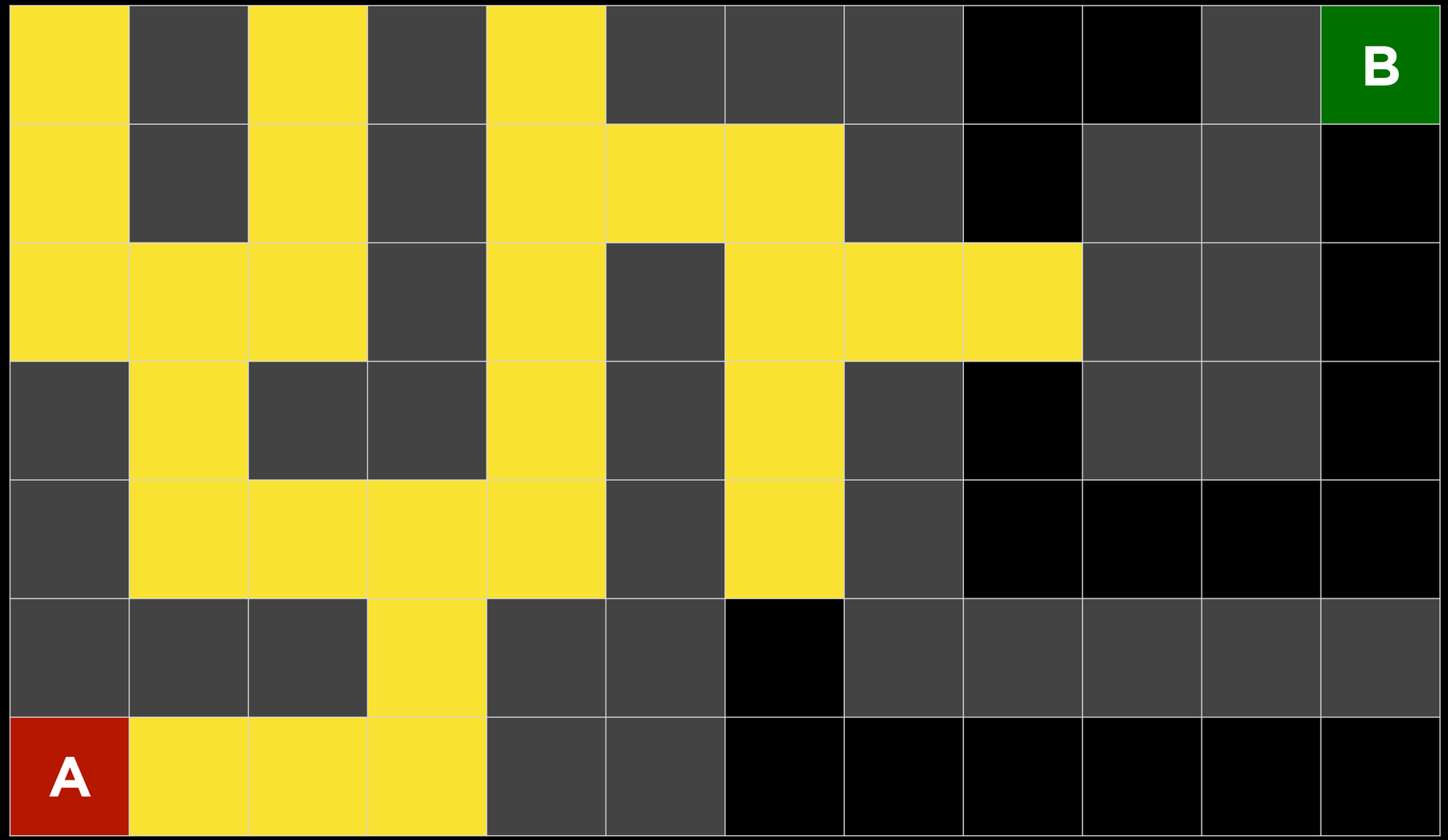
Breadth-First Search



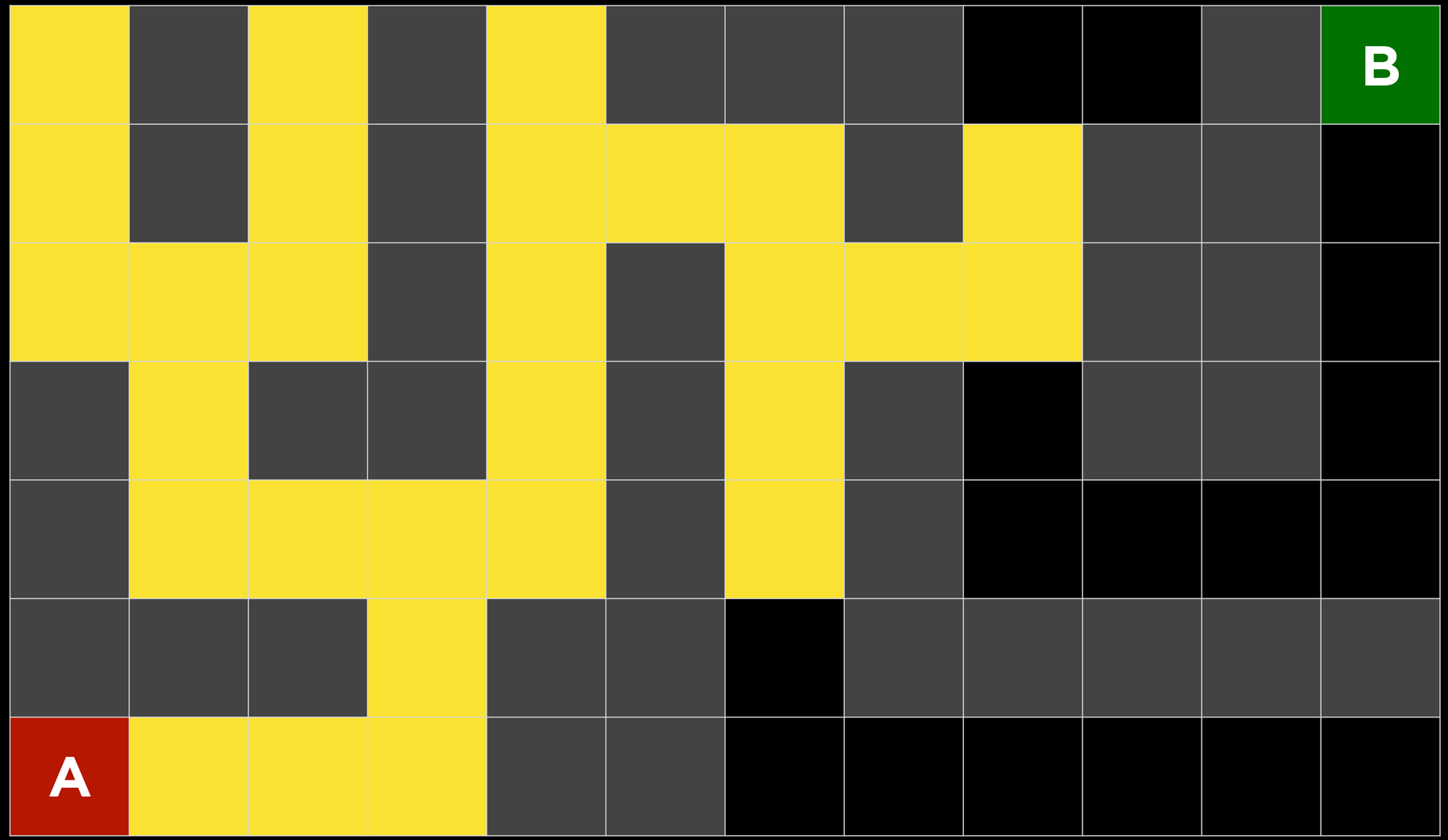
Breadth-First Search



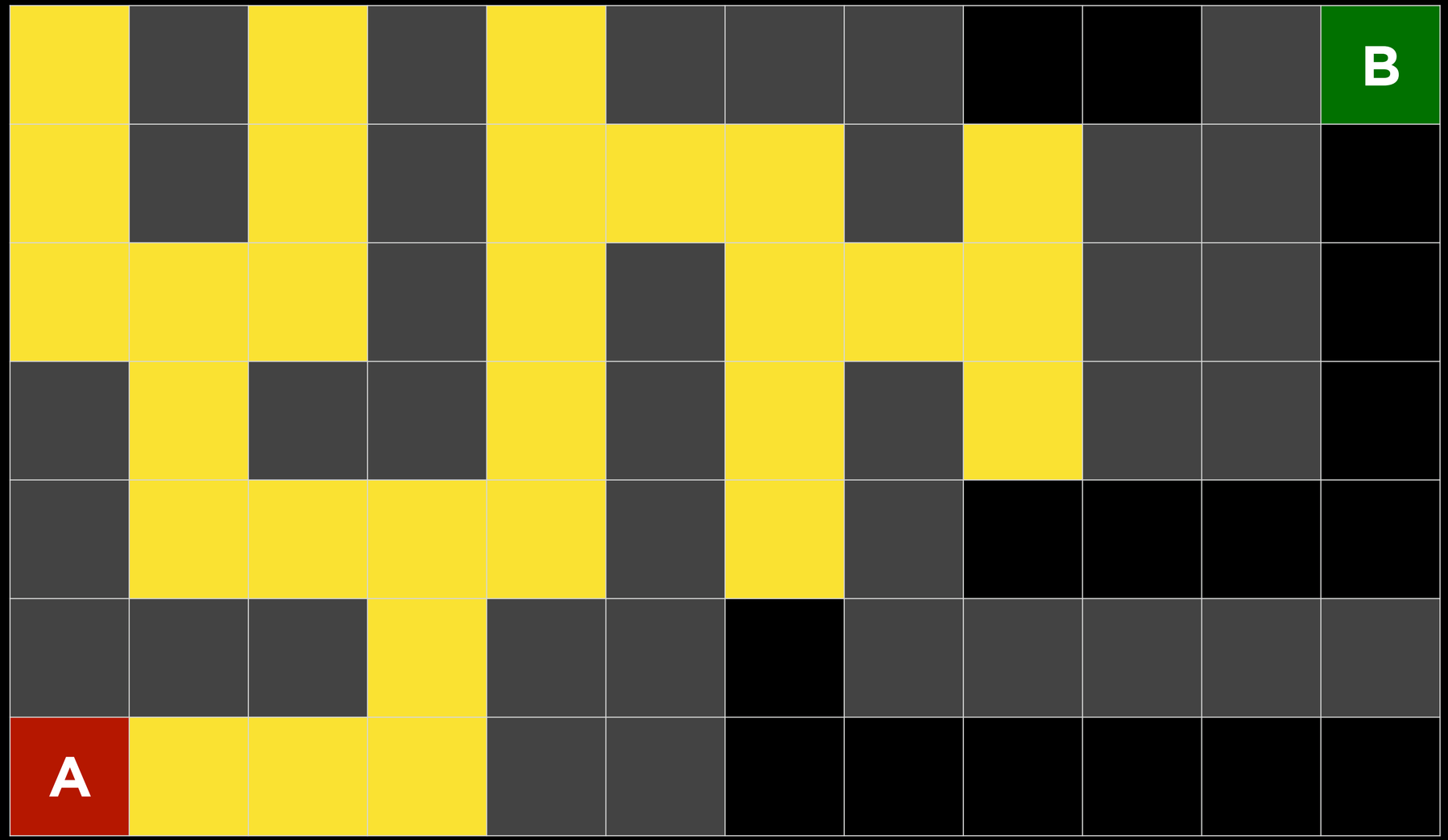
Breadth-First Search



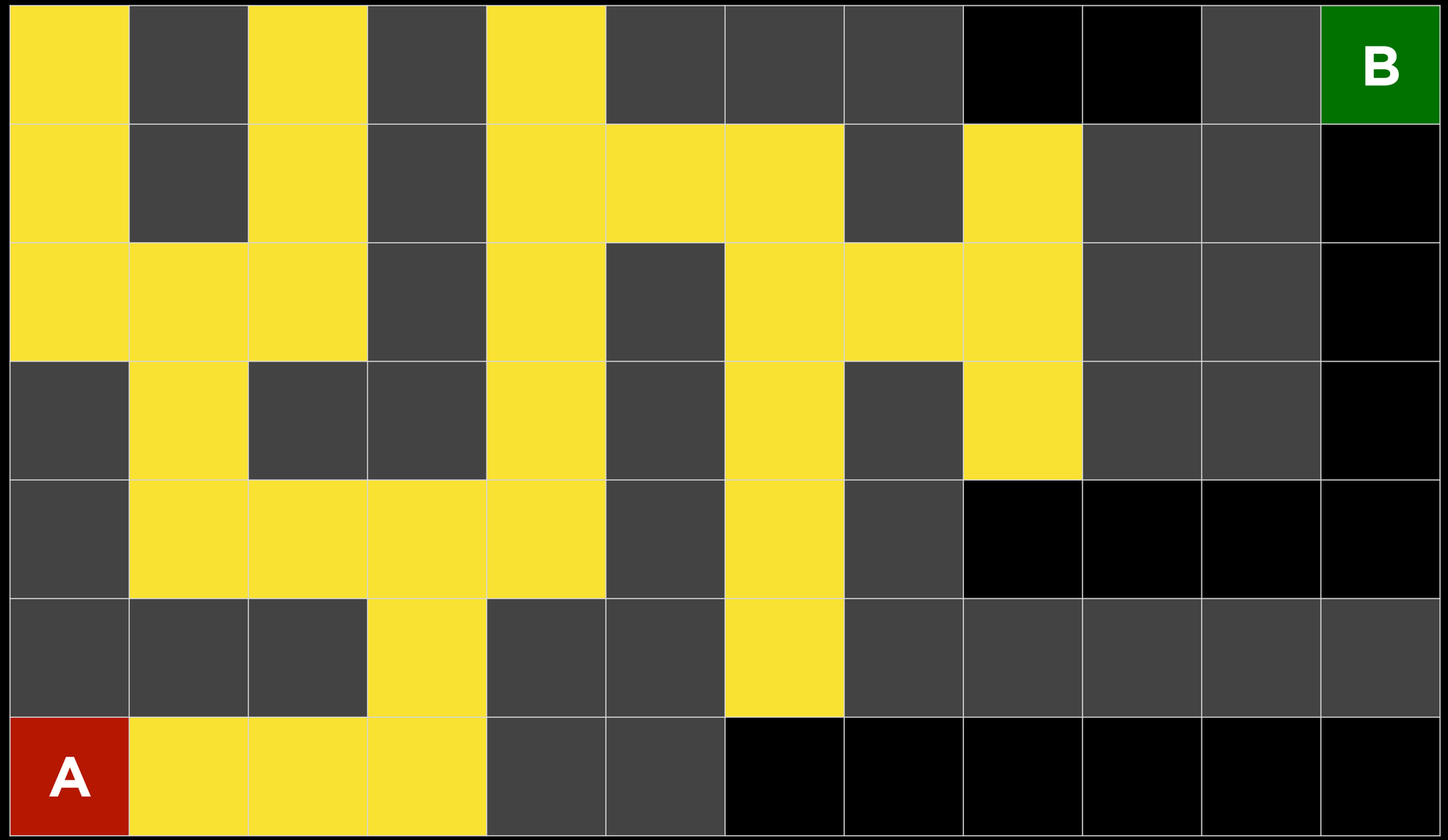
Breadth-First Search



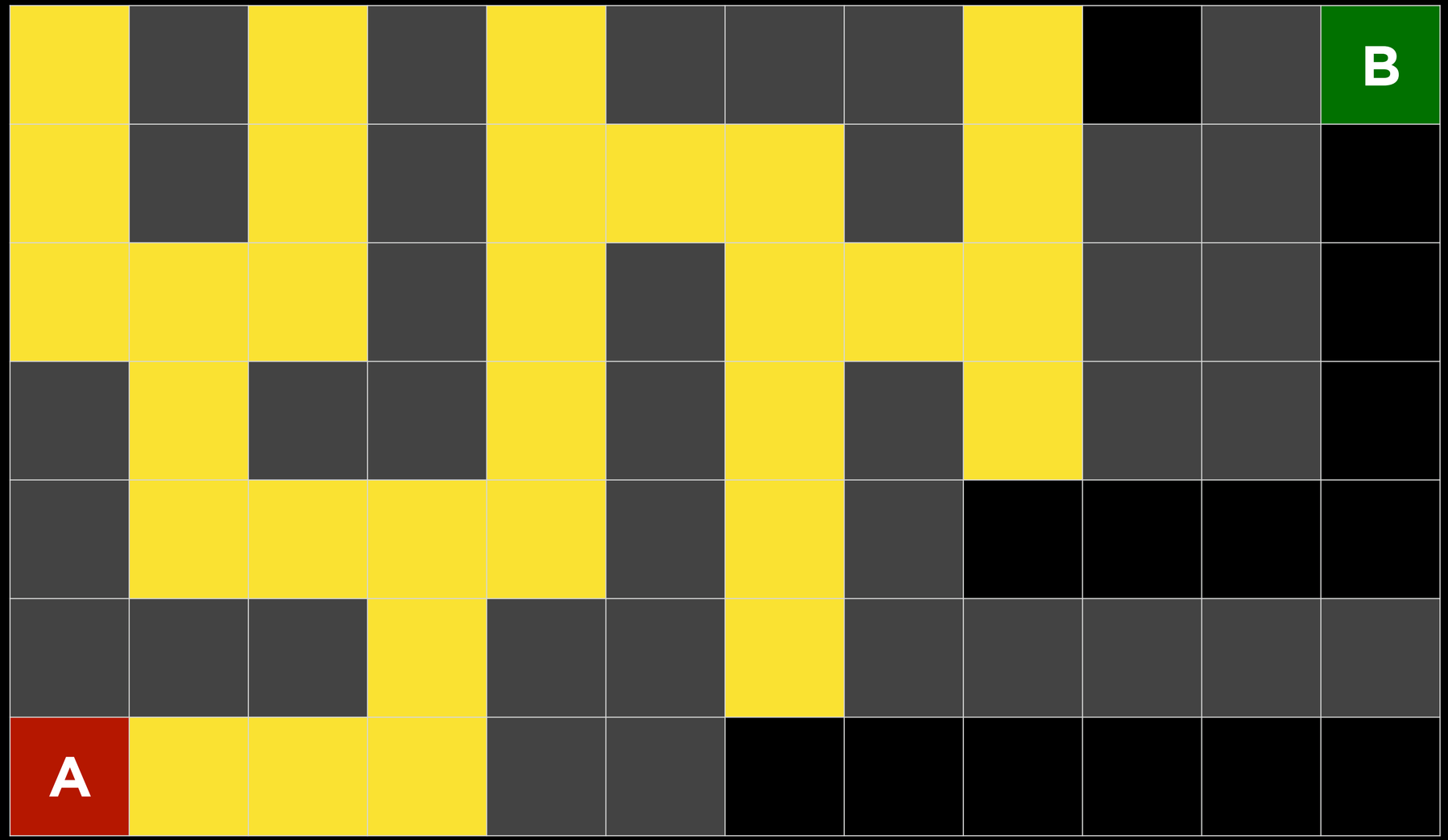
Breadth-First Search



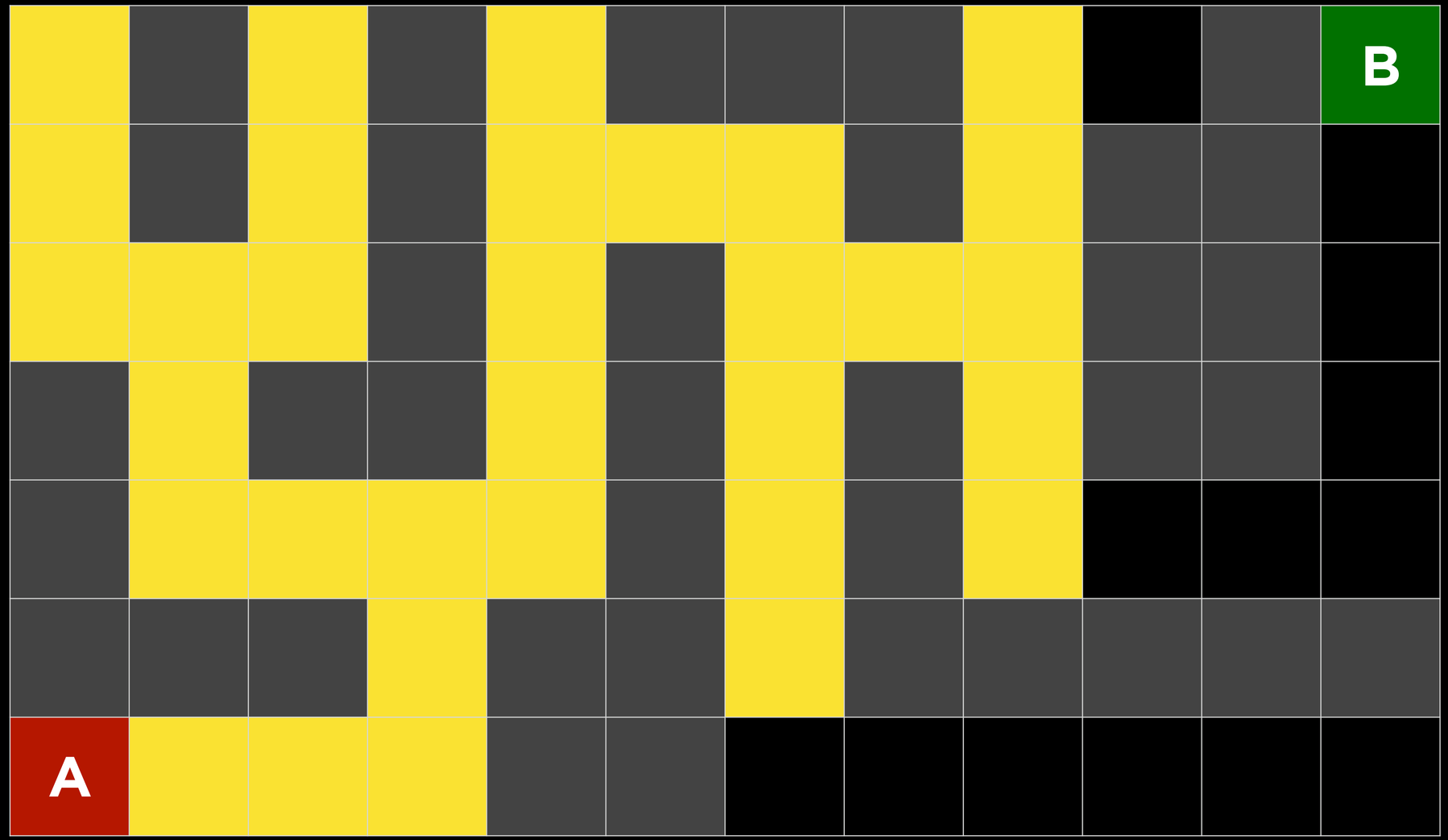
Breadth-First Search



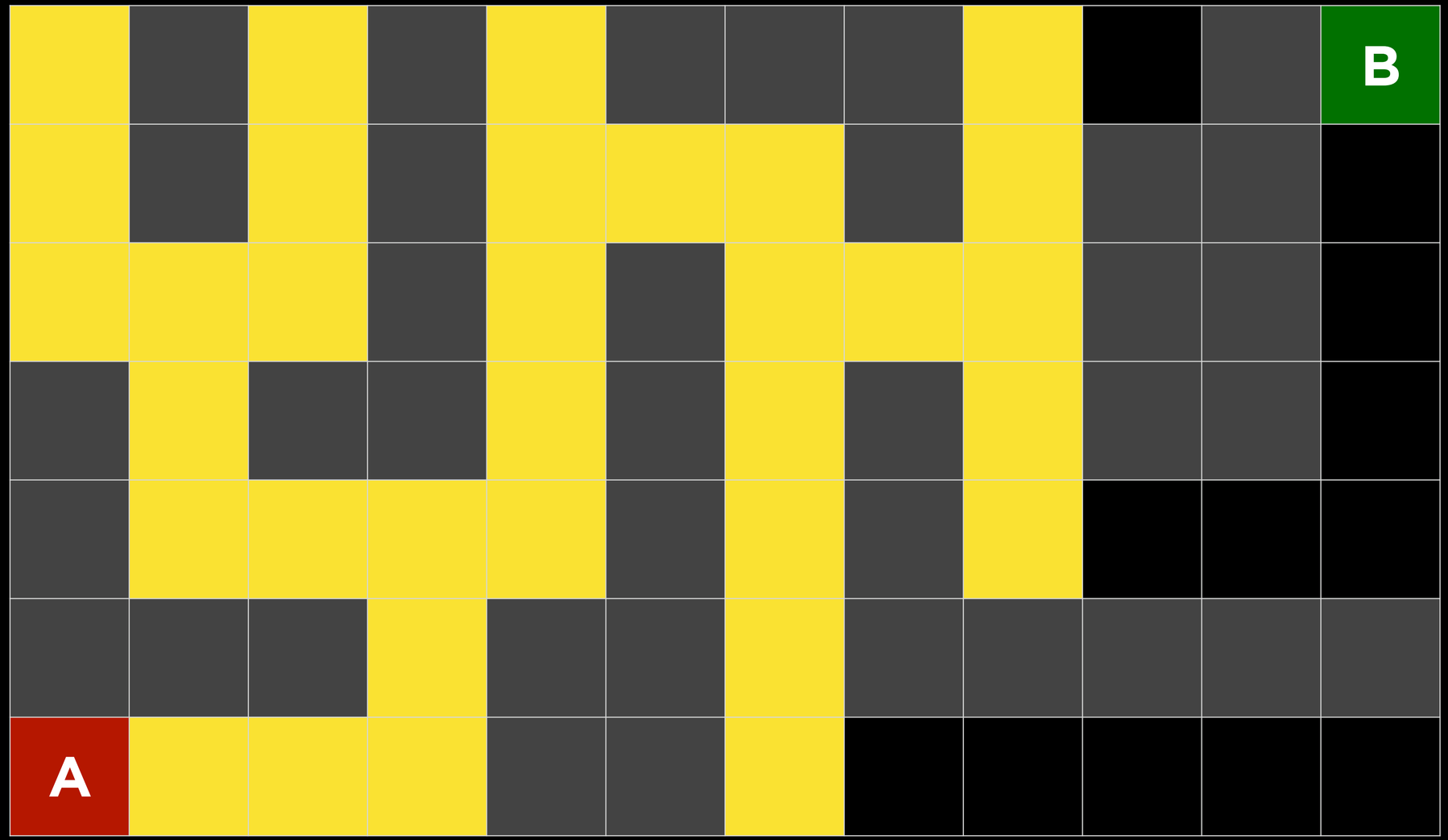
Breadth-First Search



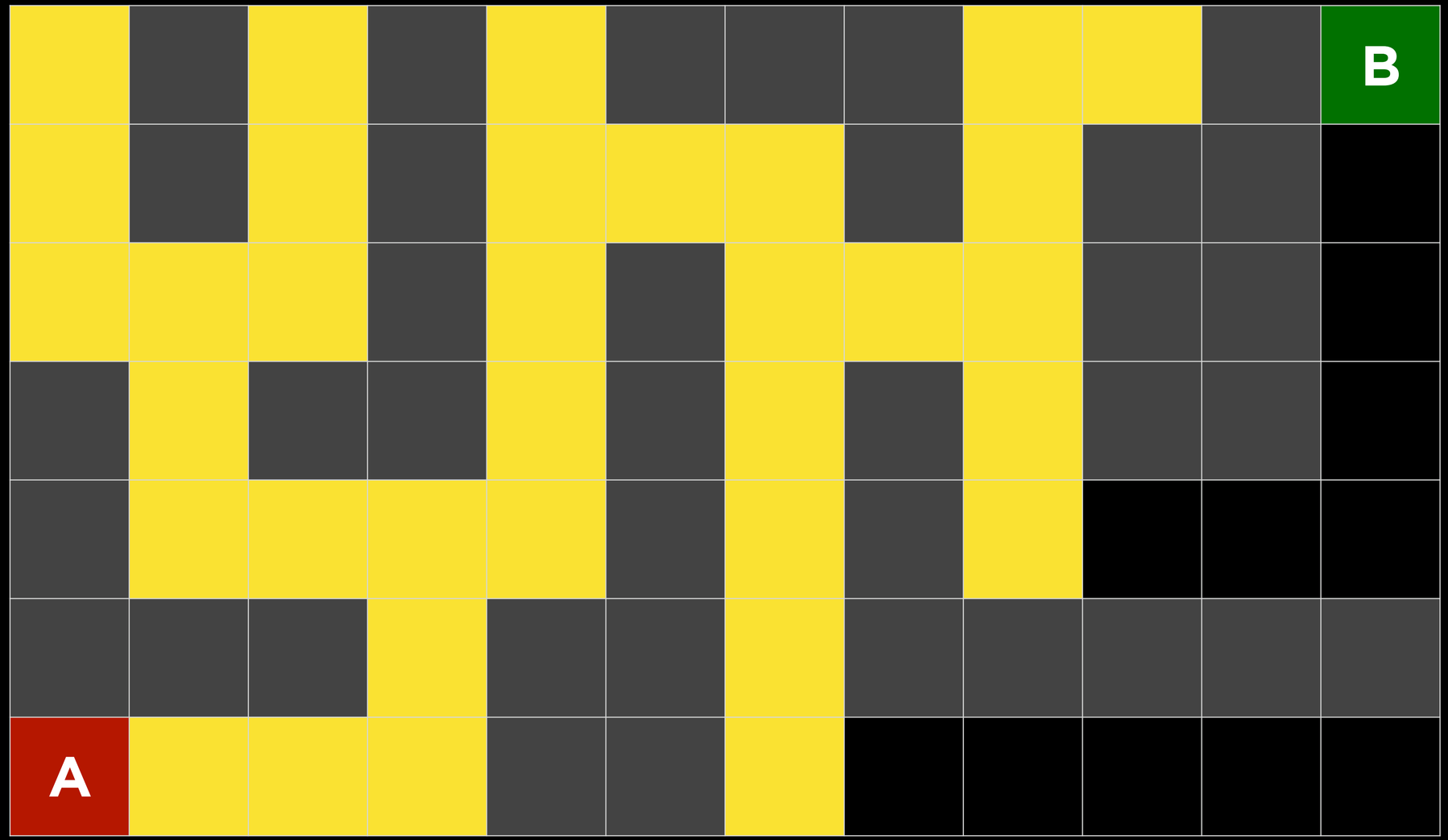
Breadth-First Search



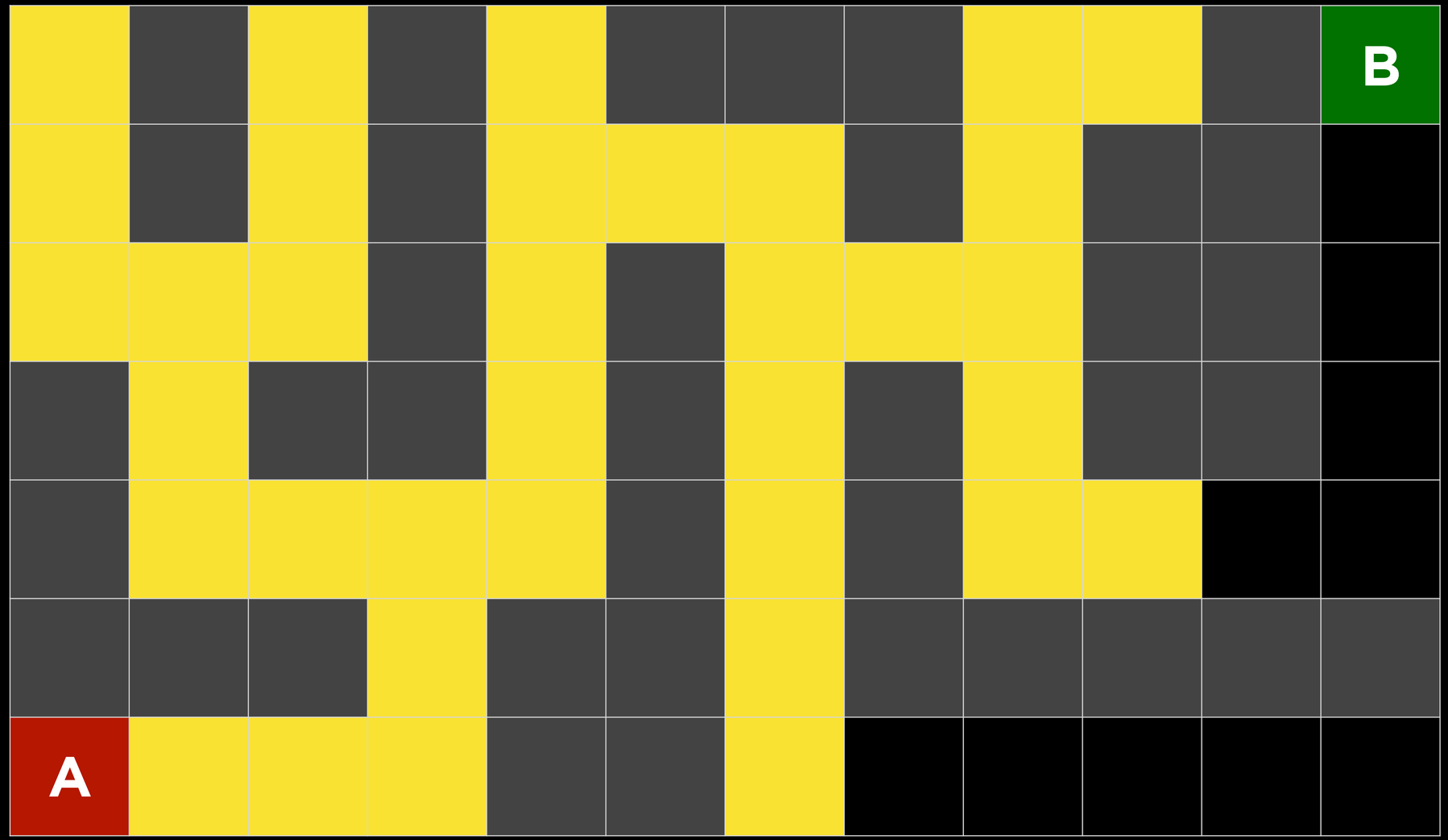
Breadth-First Search



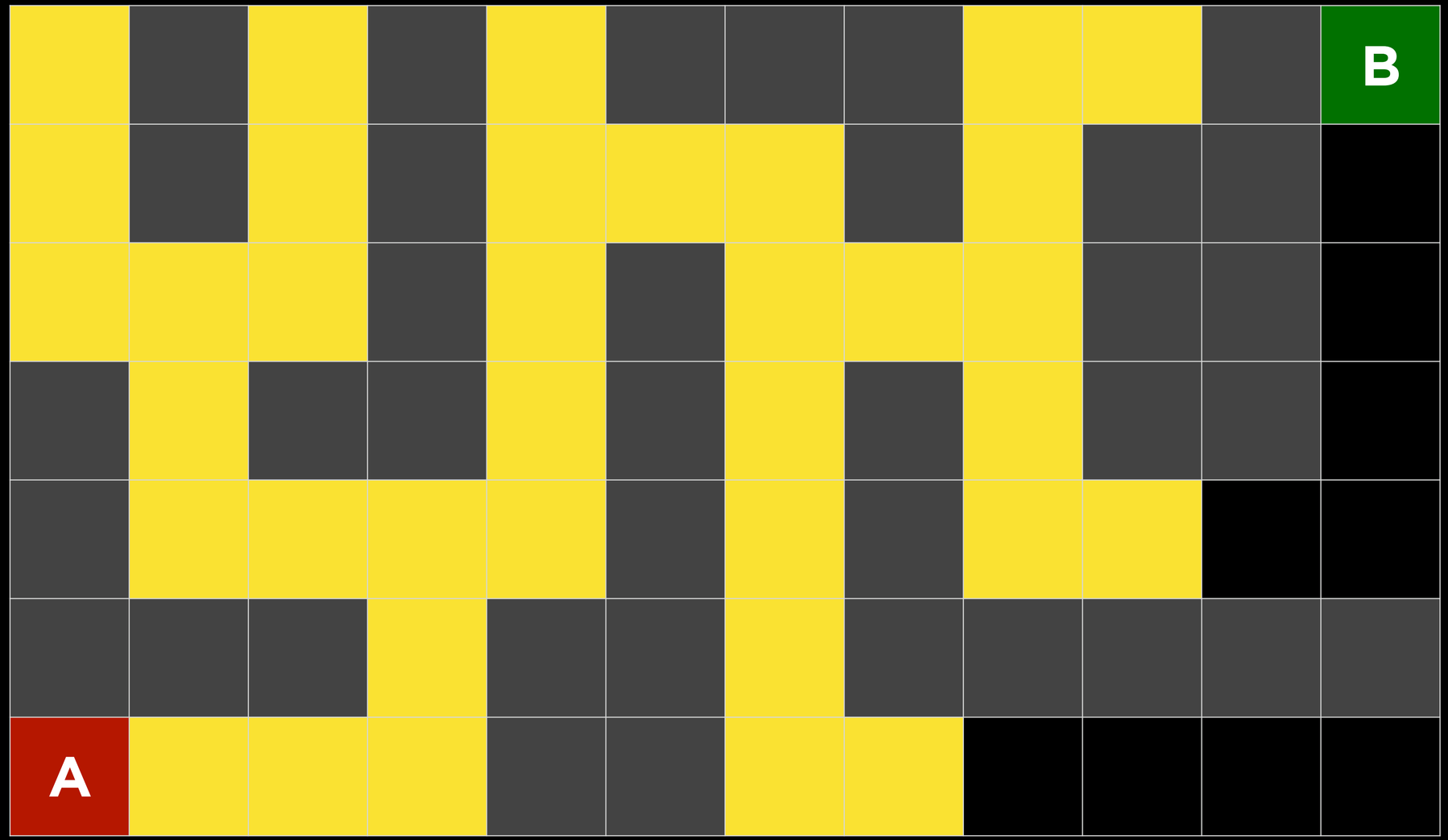
Breadth-First Search



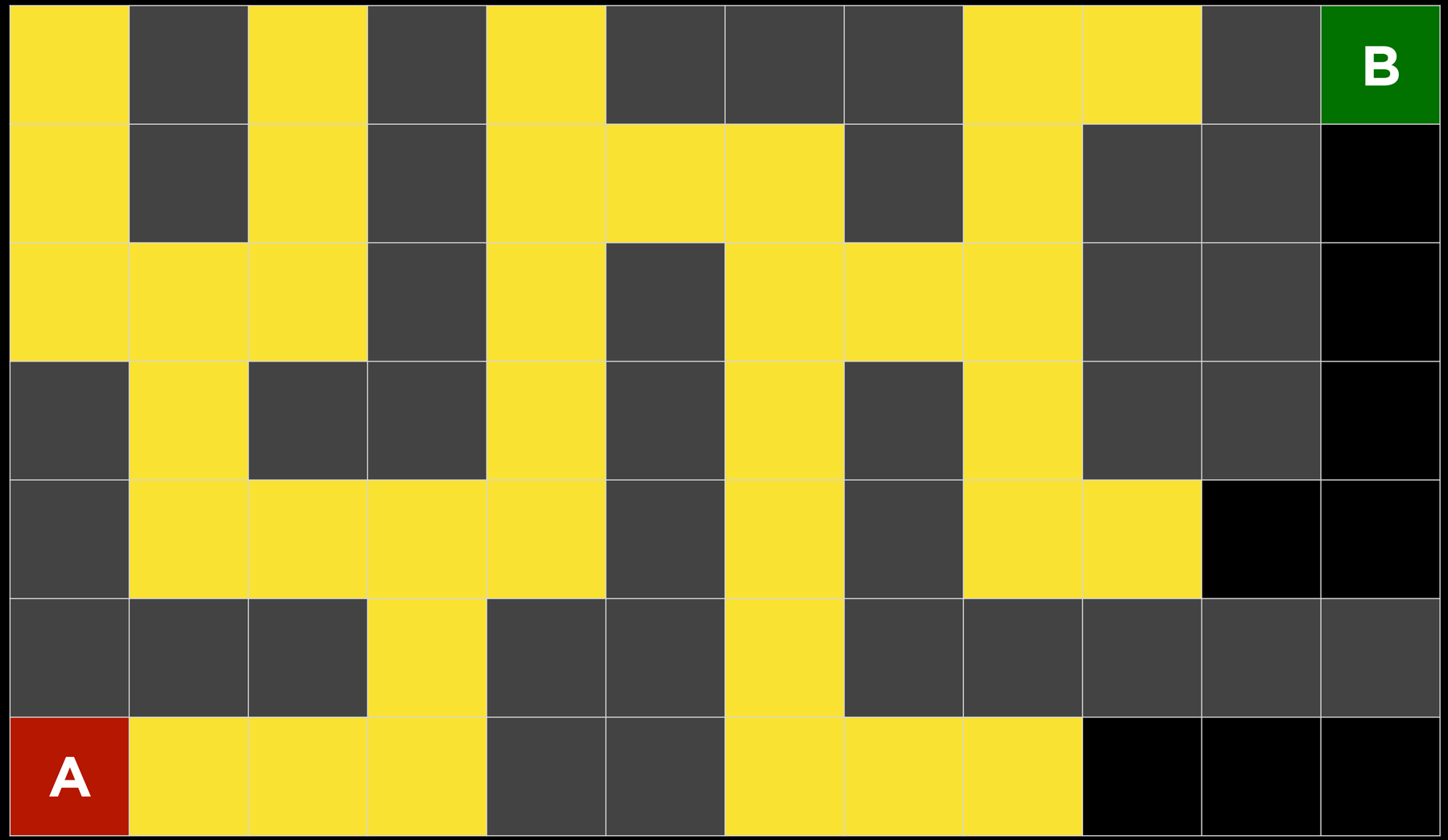
Breadth-First Search



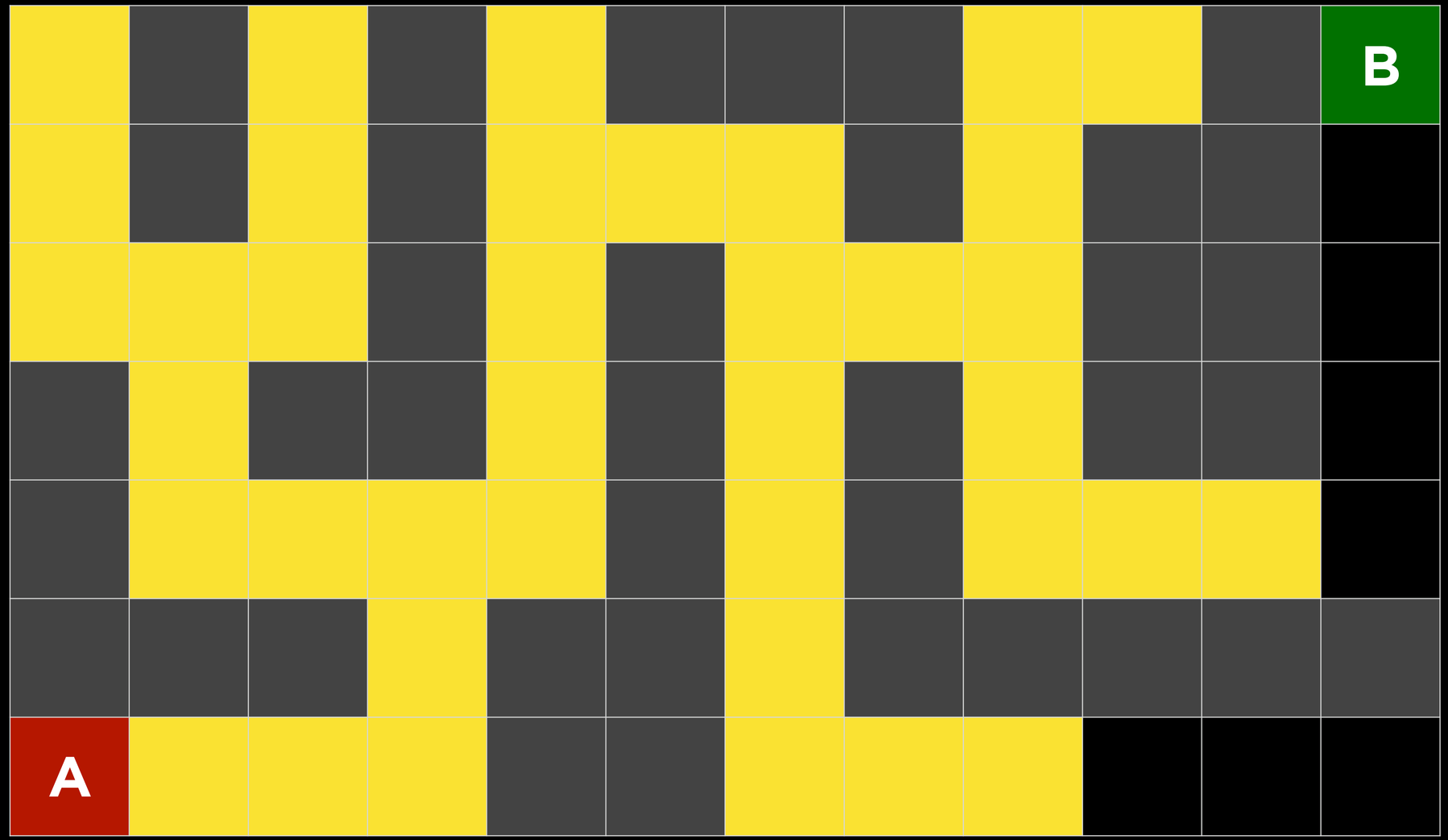
Breadth-First Search



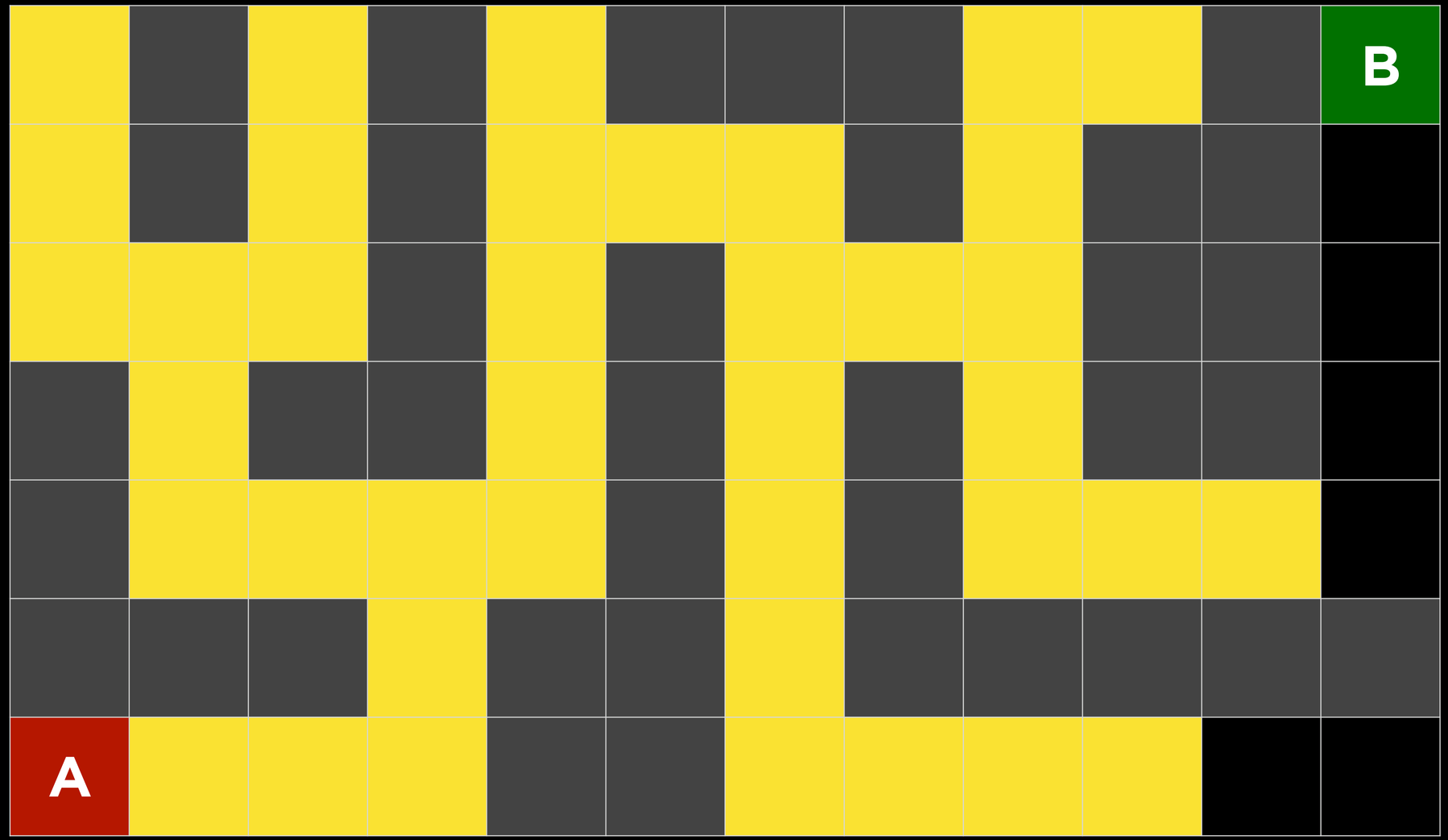
Breadth-First Search



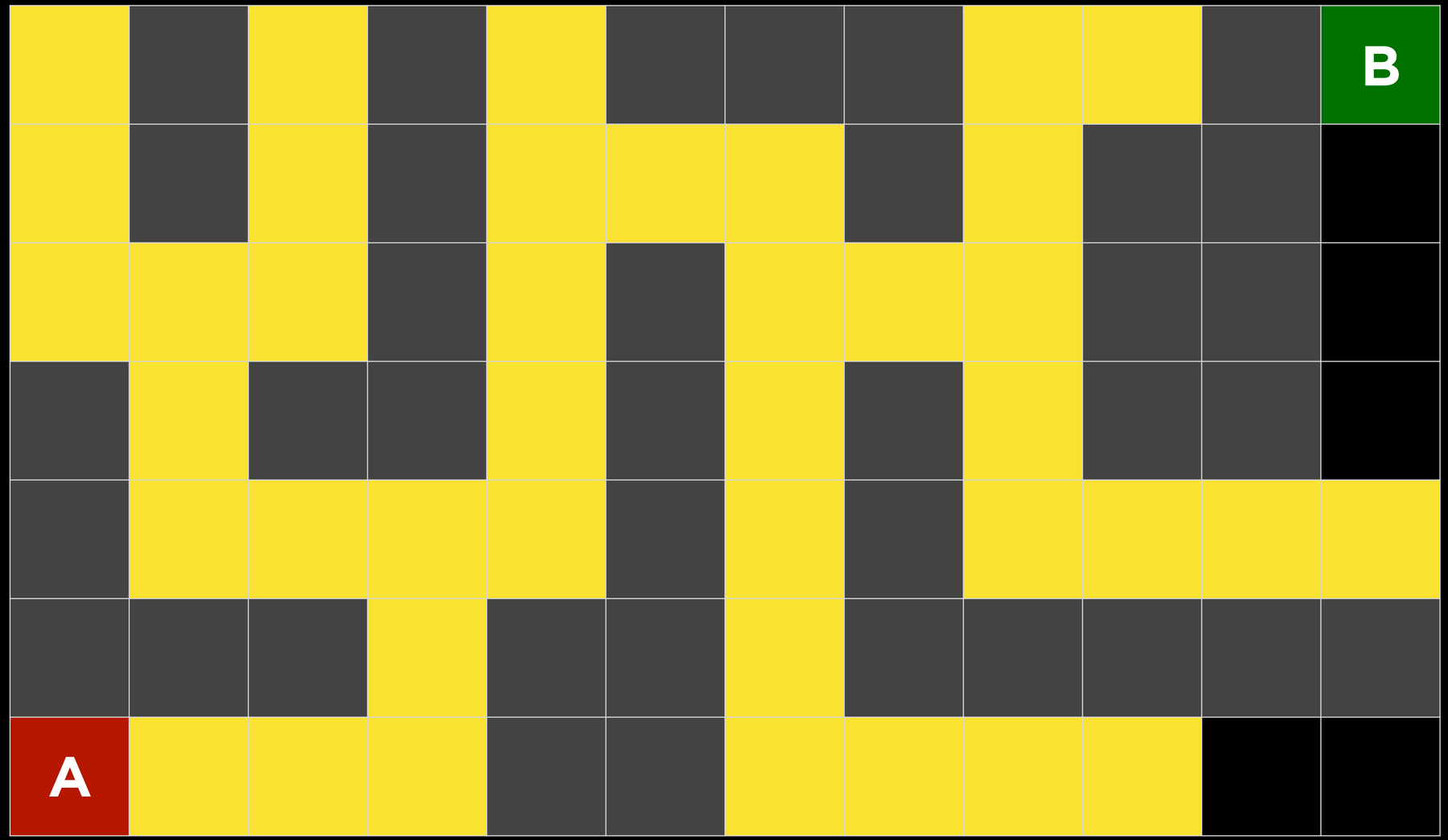
Breadth-First Search



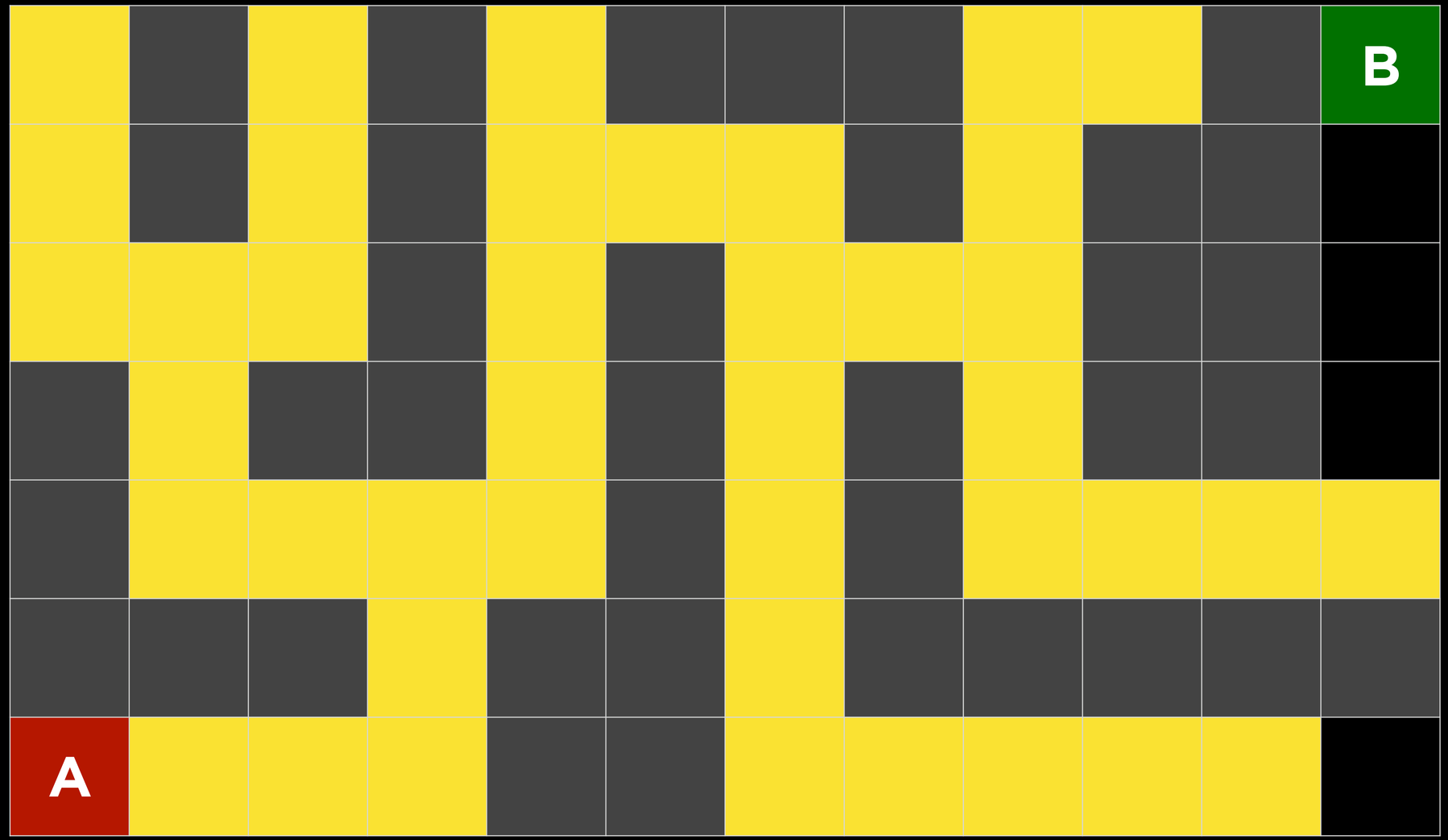
Breadth-First Search



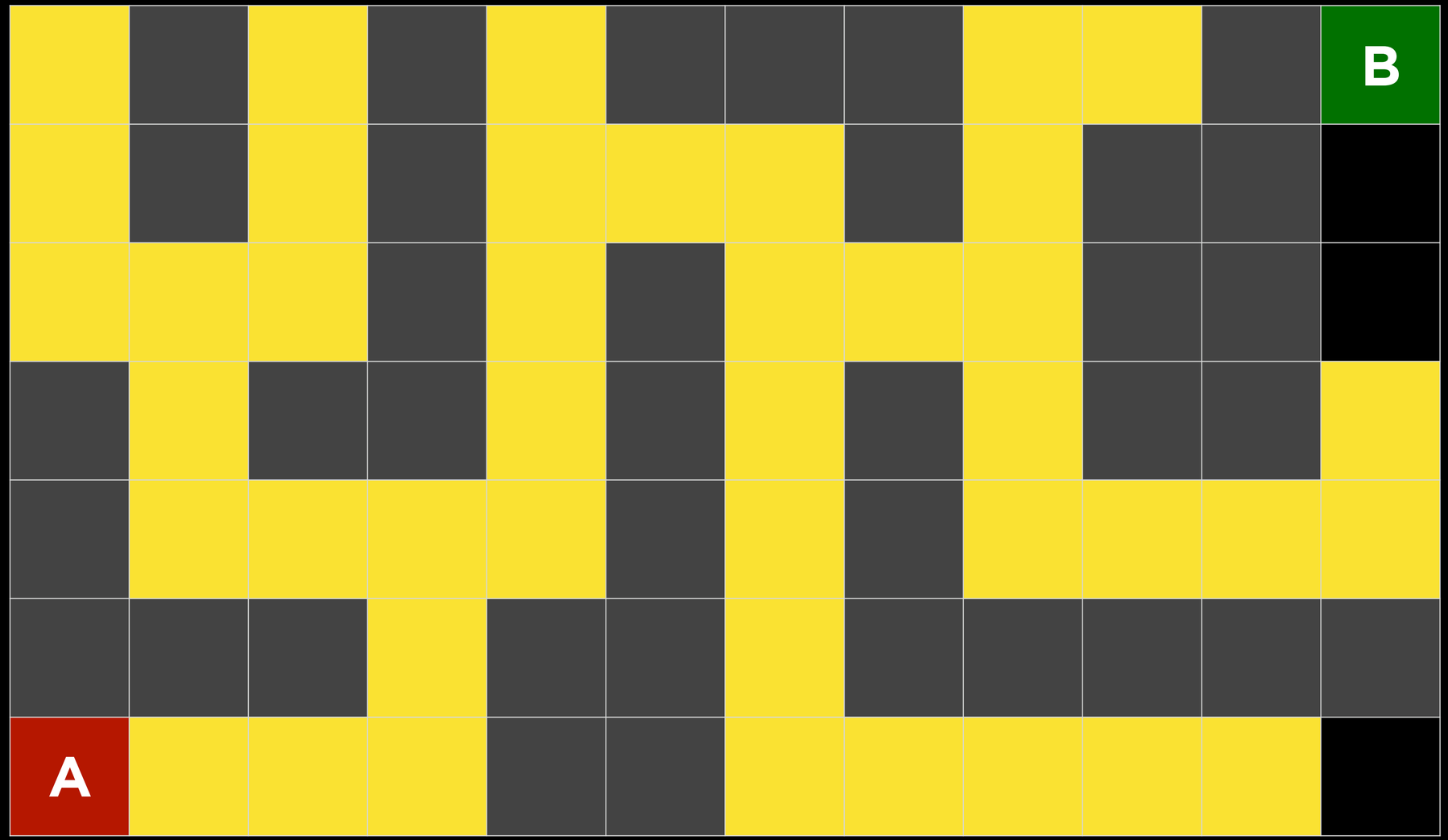
Breadth-First Search



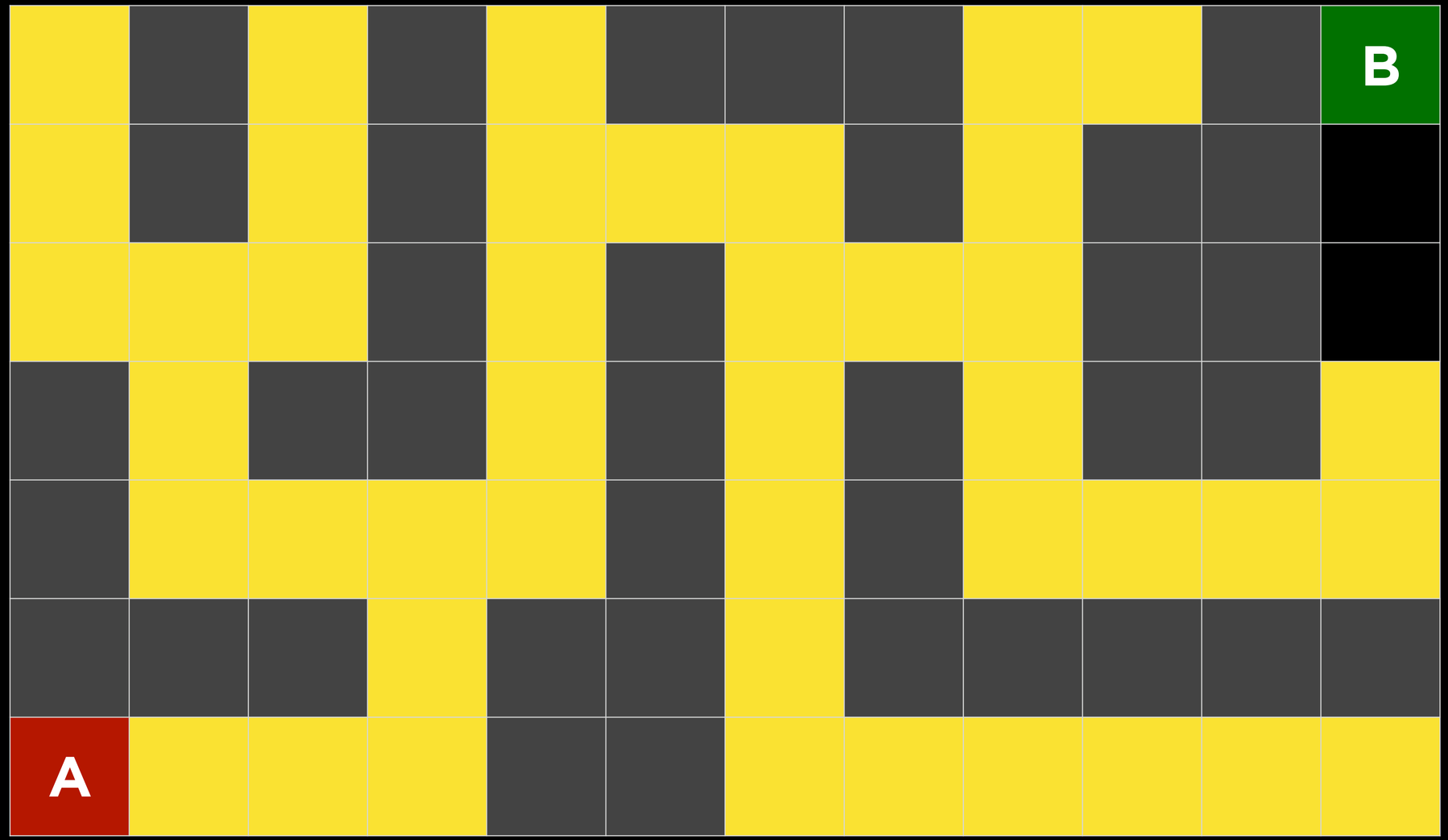
Breadth-First Search



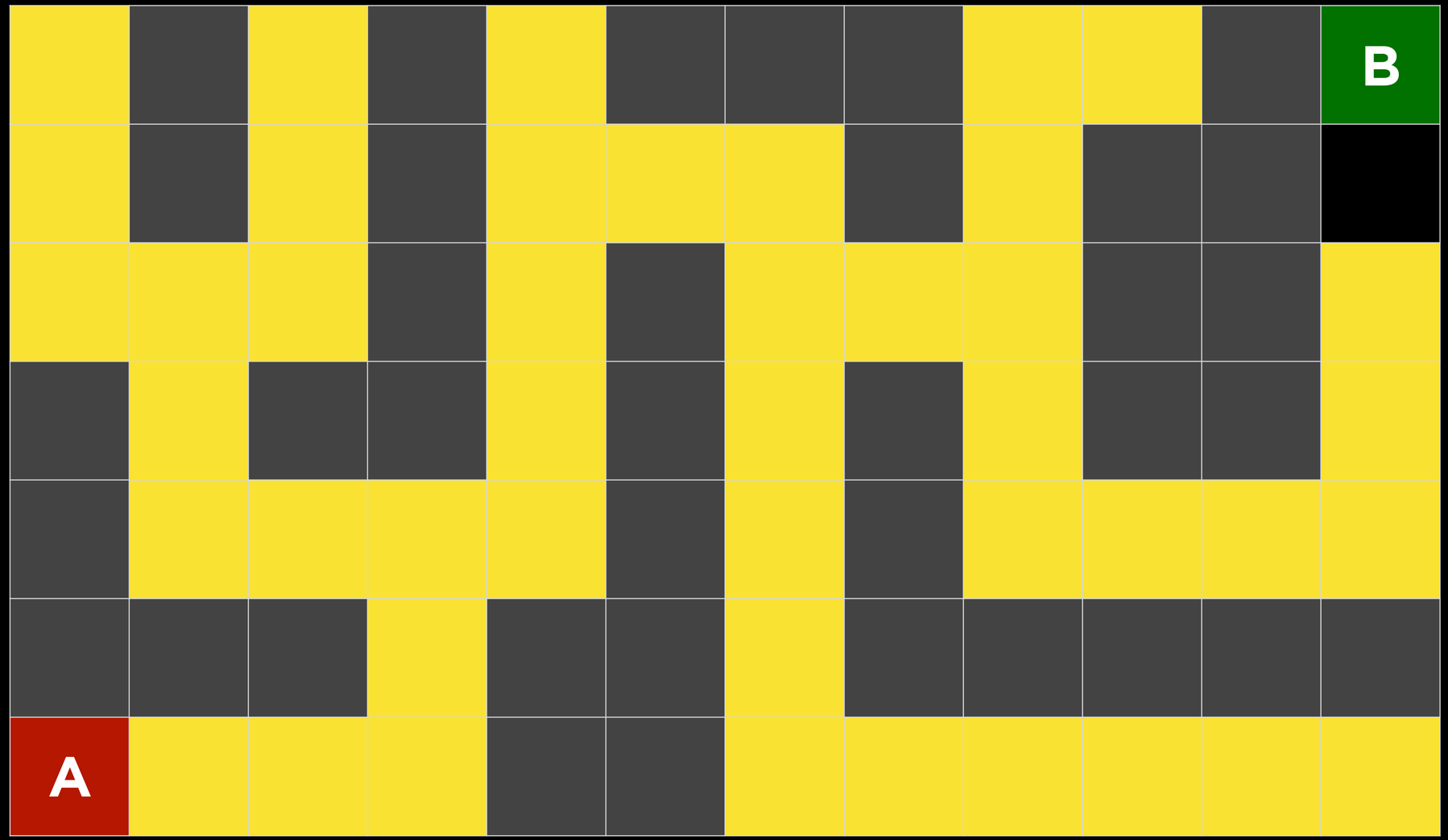
Breadth-First Search



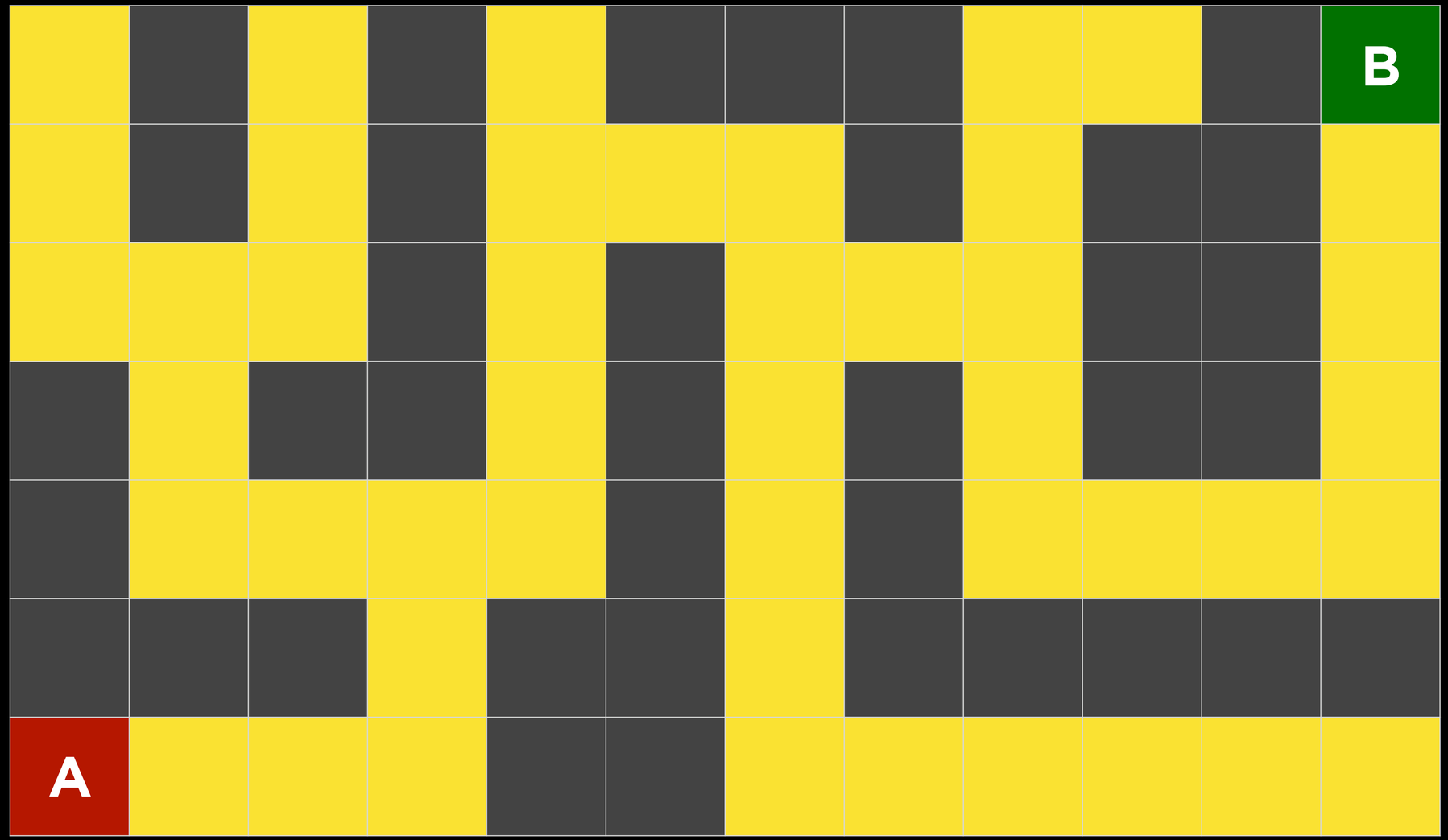
Breadth-First Search



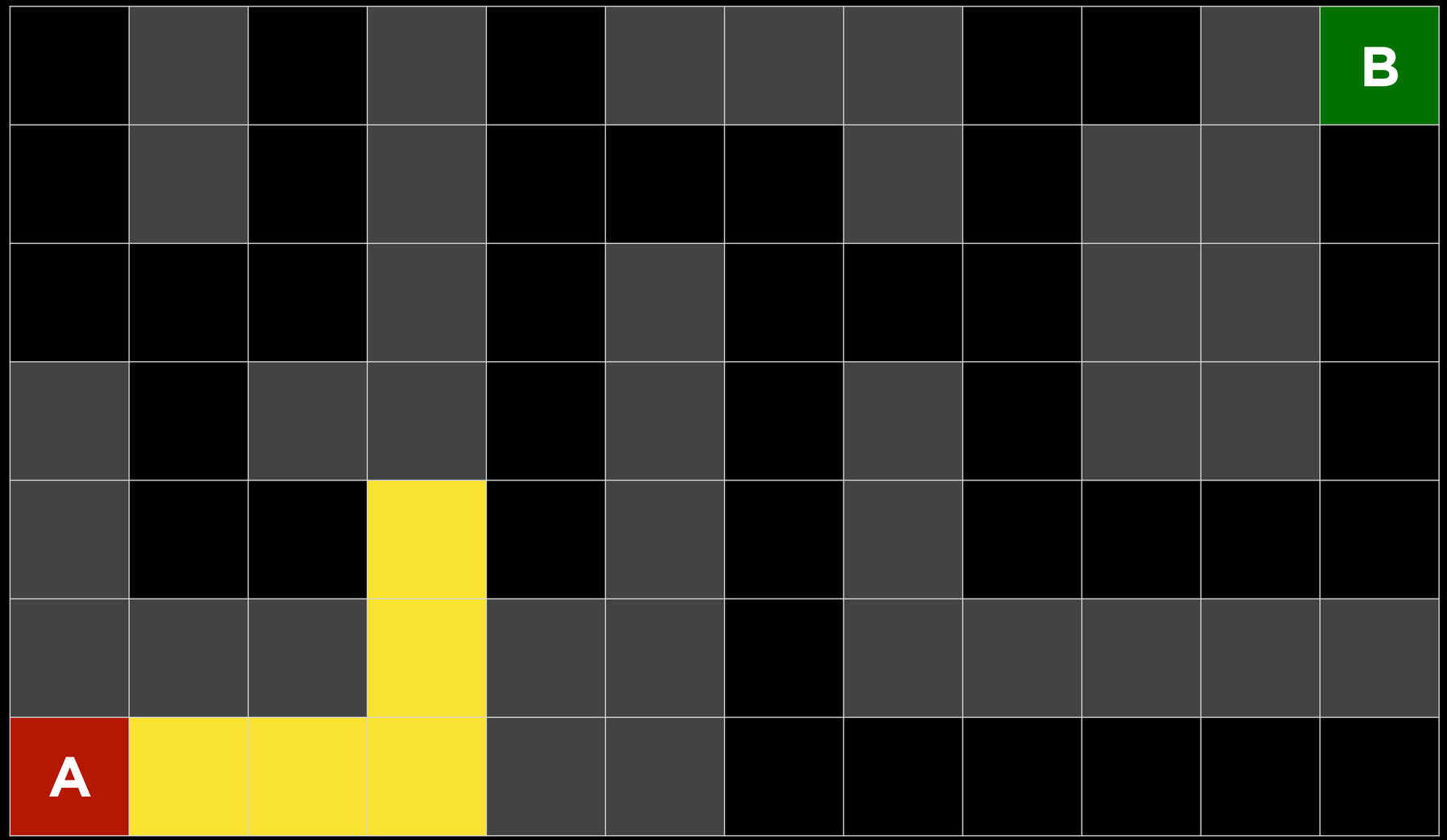
Breadth-First Search



Breadth-First Search



Breadth-First Search



uninformed search

search strategy that uses no problem-specific knowledge

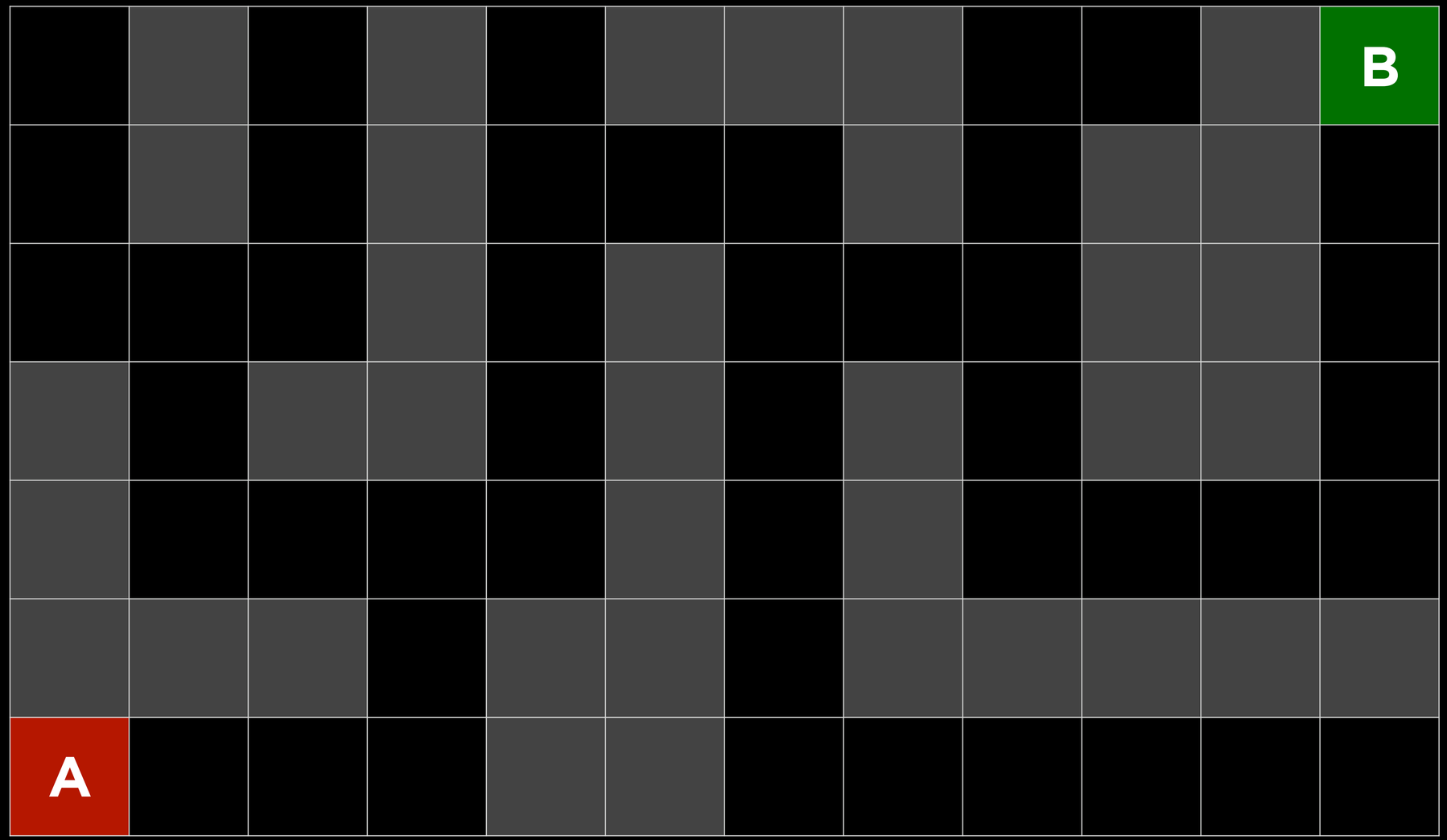
informed search

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

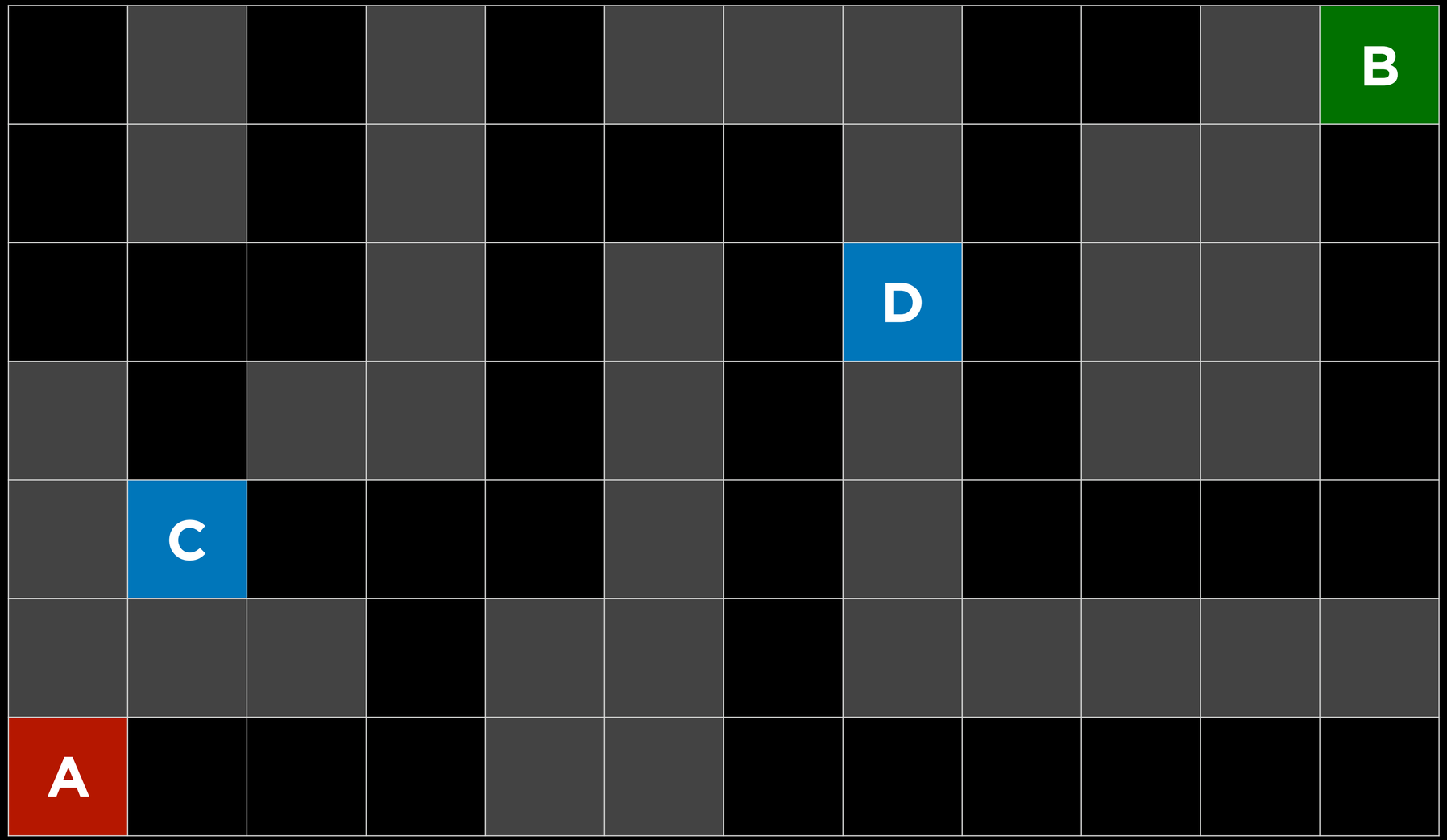
greedy best-first search

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

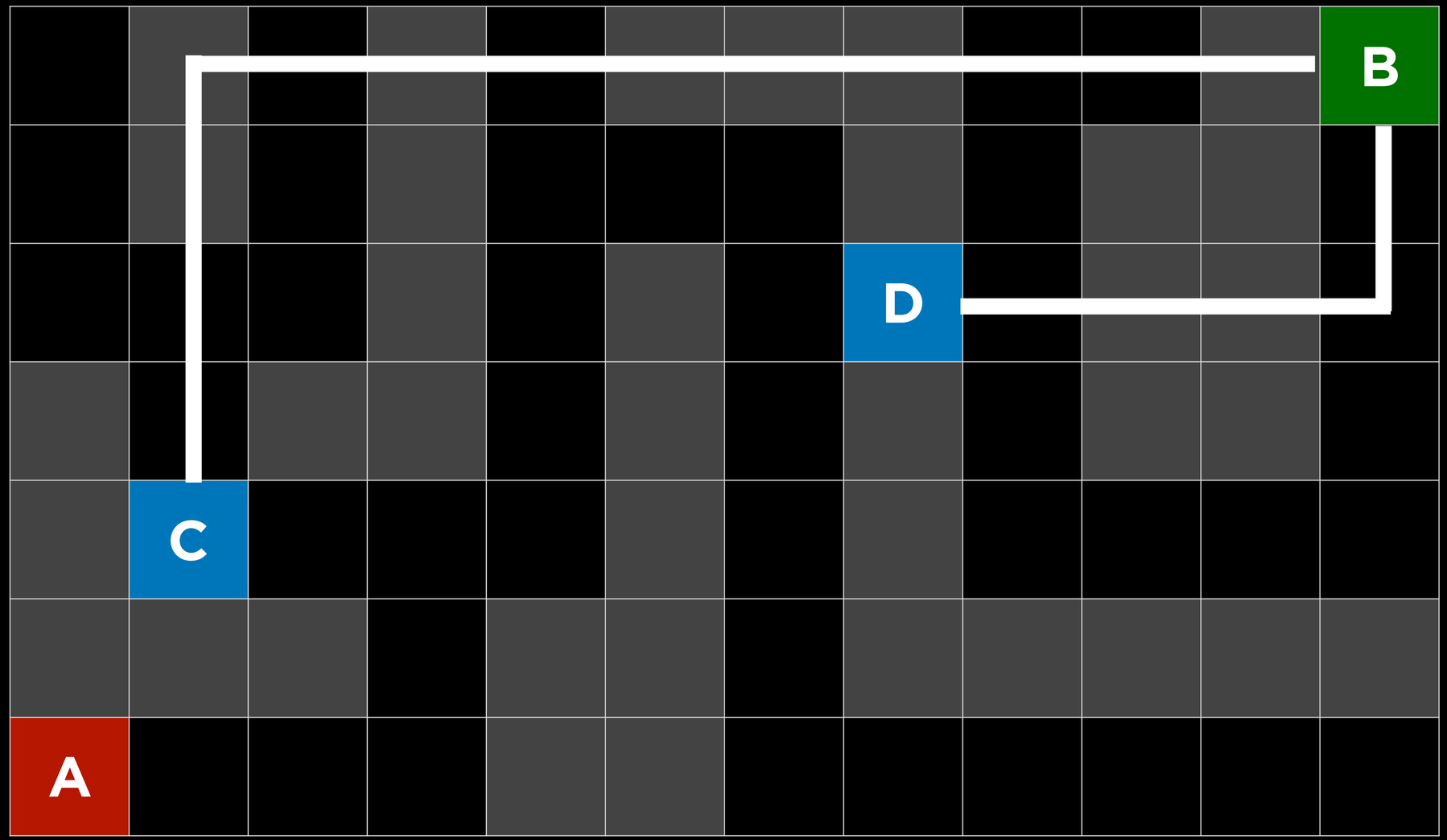
Heuristic function?



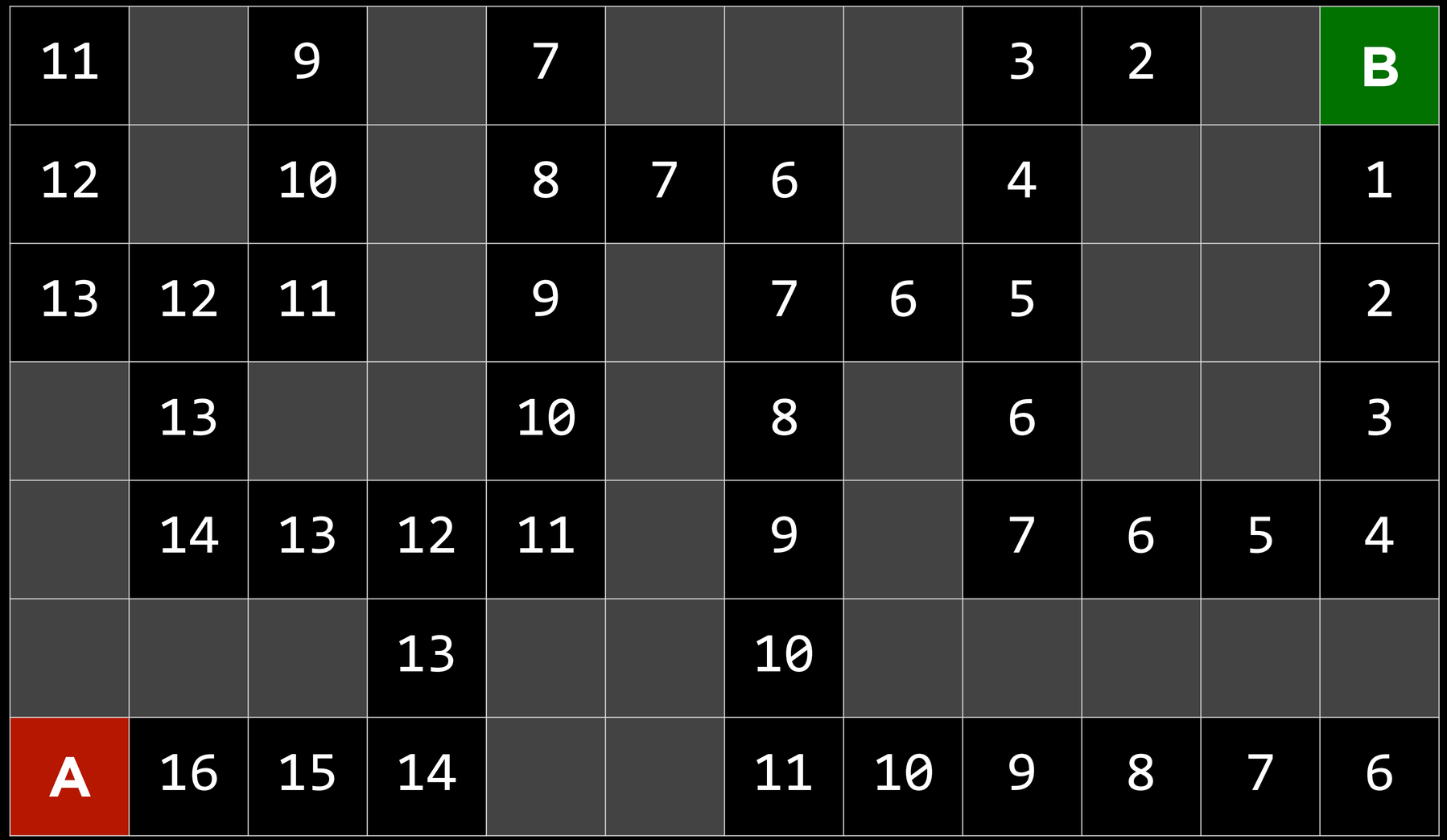
Heuristic function?



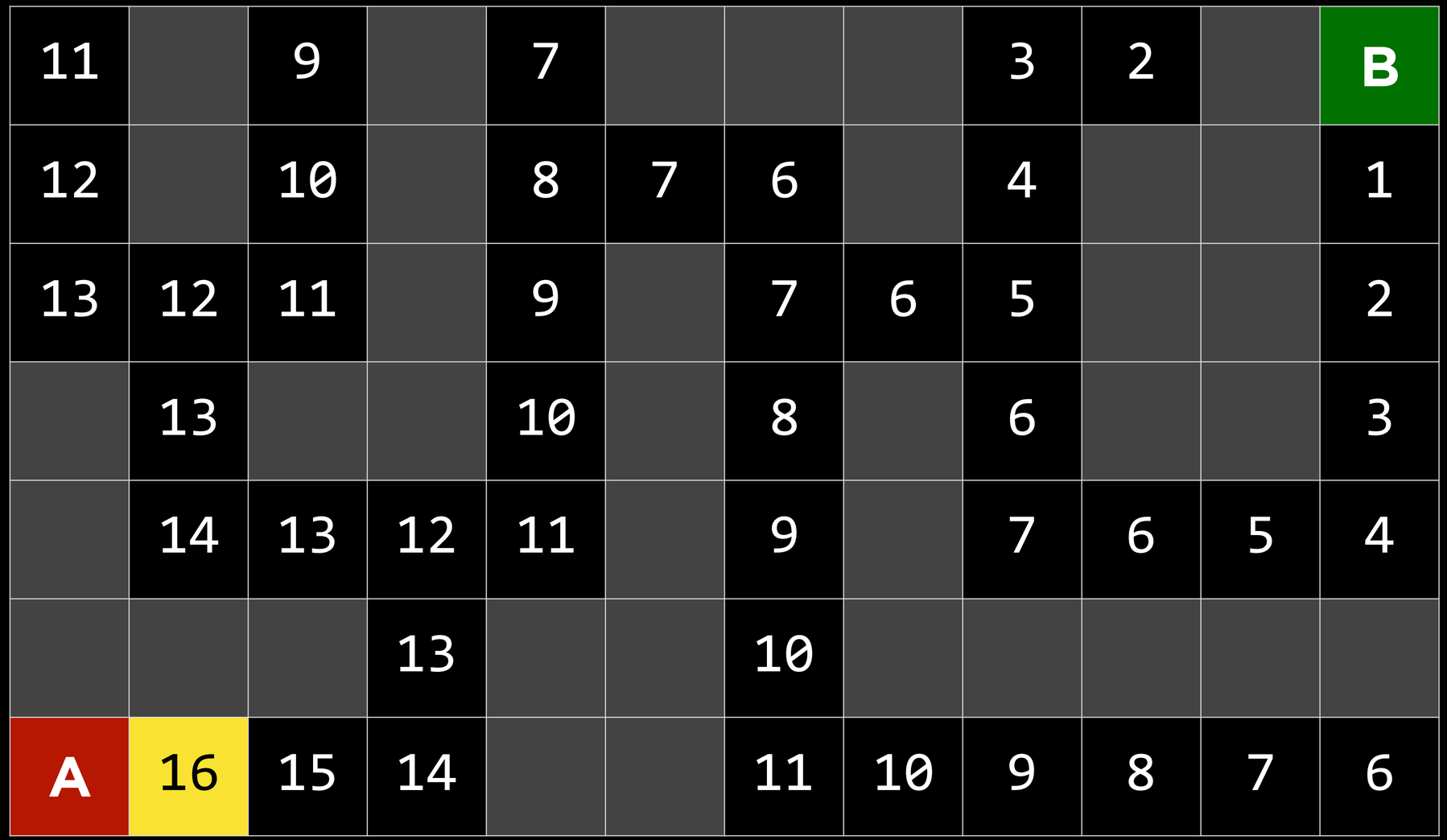
Heuristic function? Manhattan distance.



Greedy Best-First Search



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

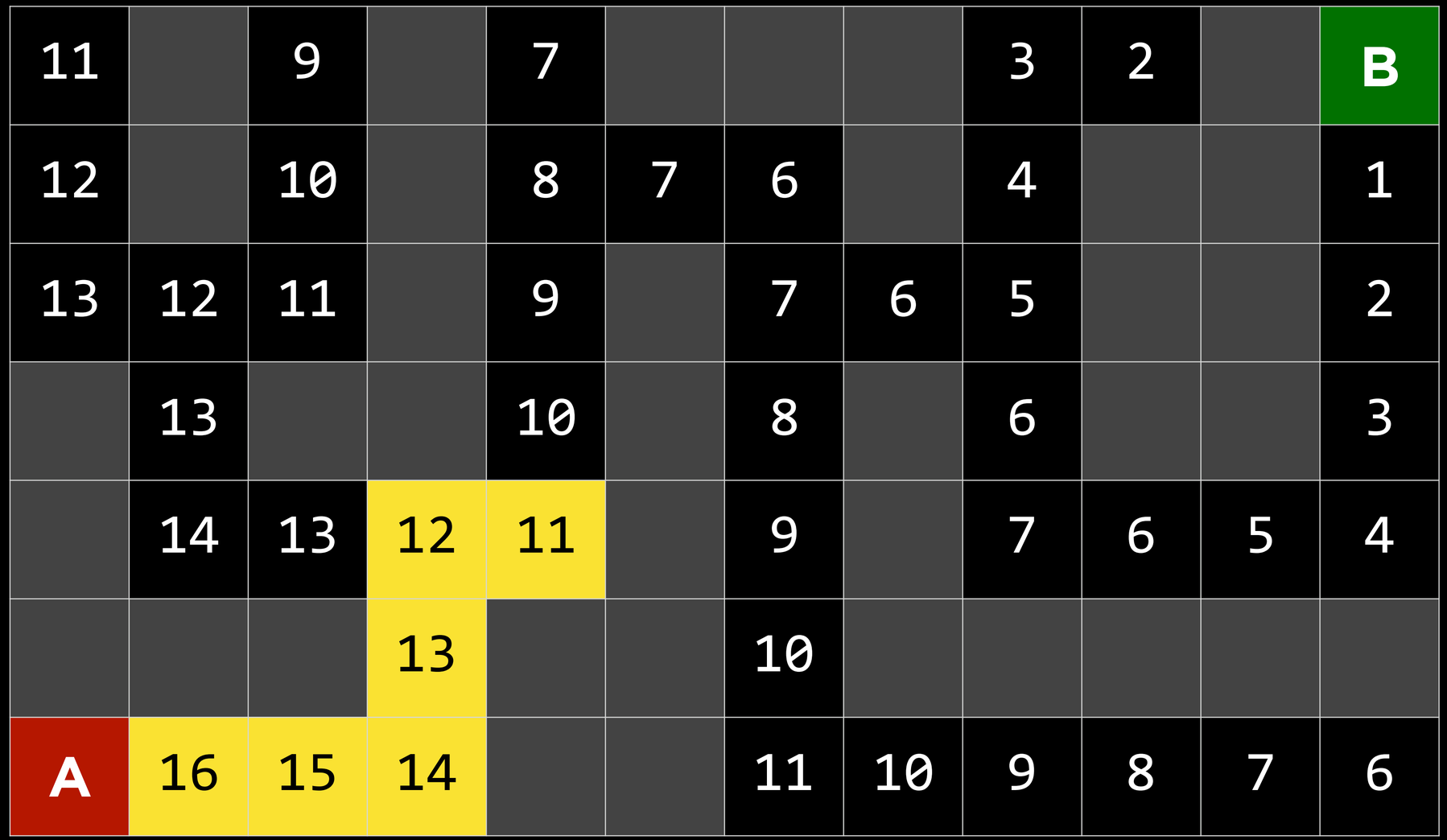
Greedy Best-First Search

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

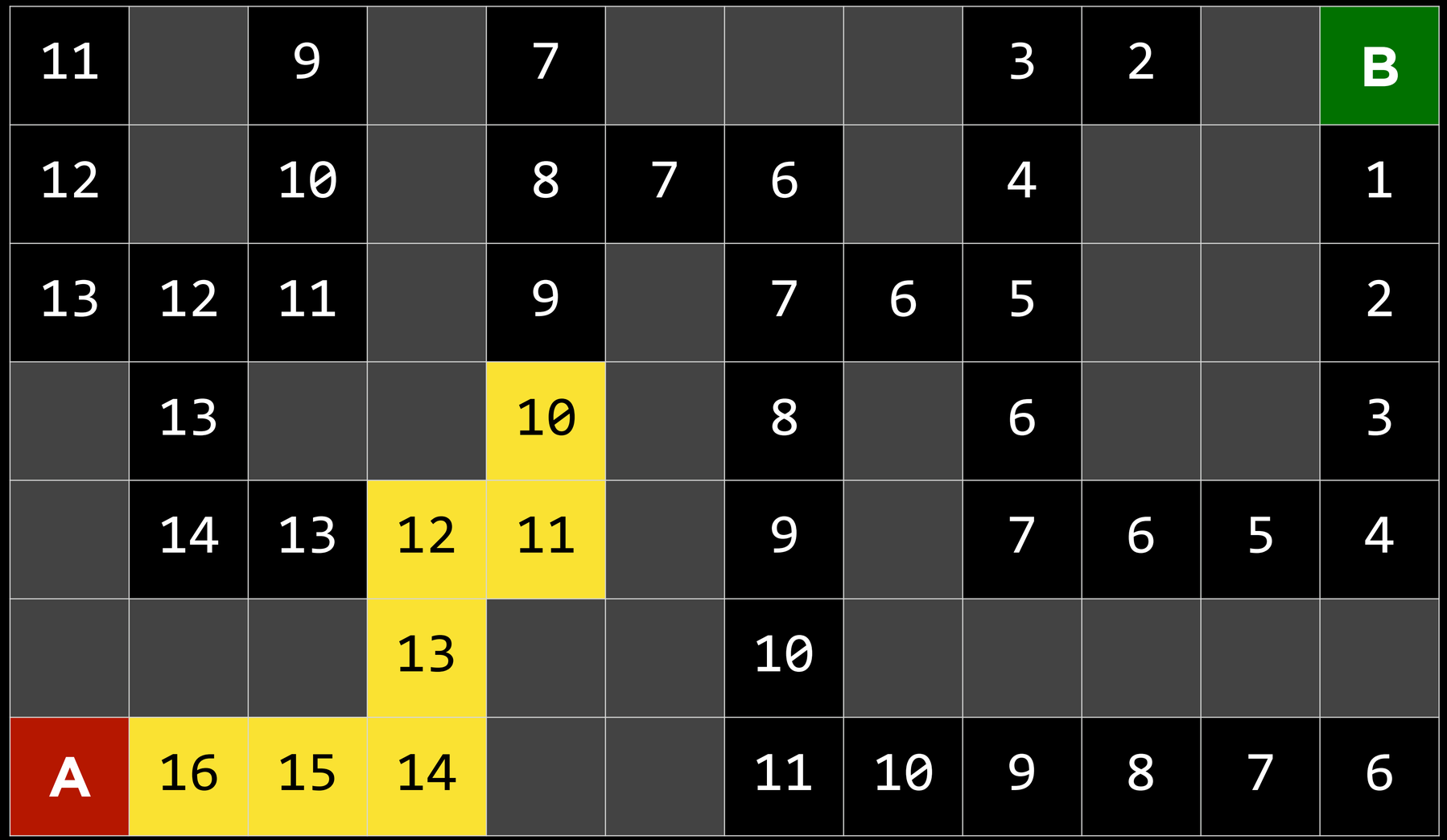
Greedy Best-First Search

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

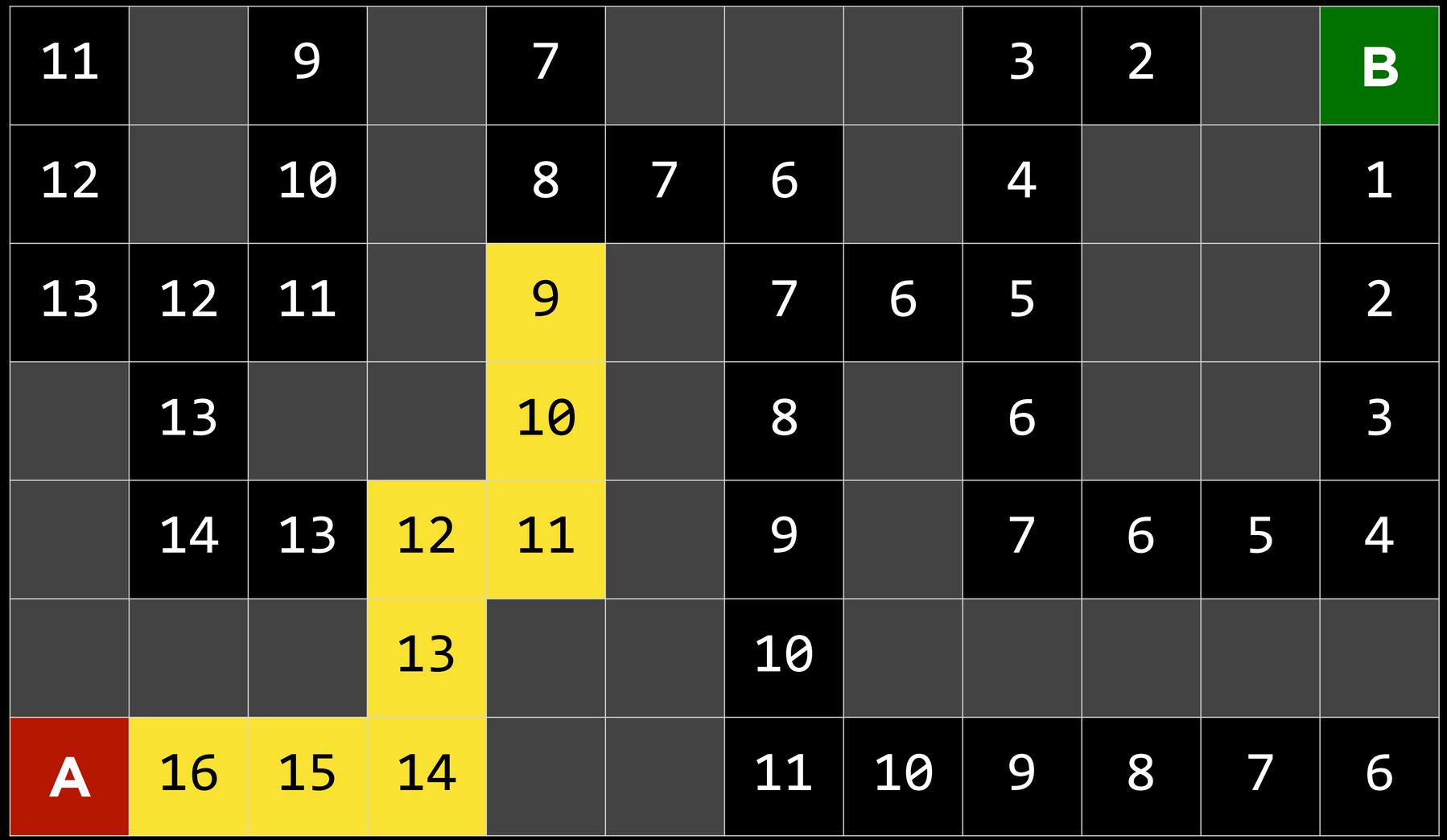
Greedy Best-First Search



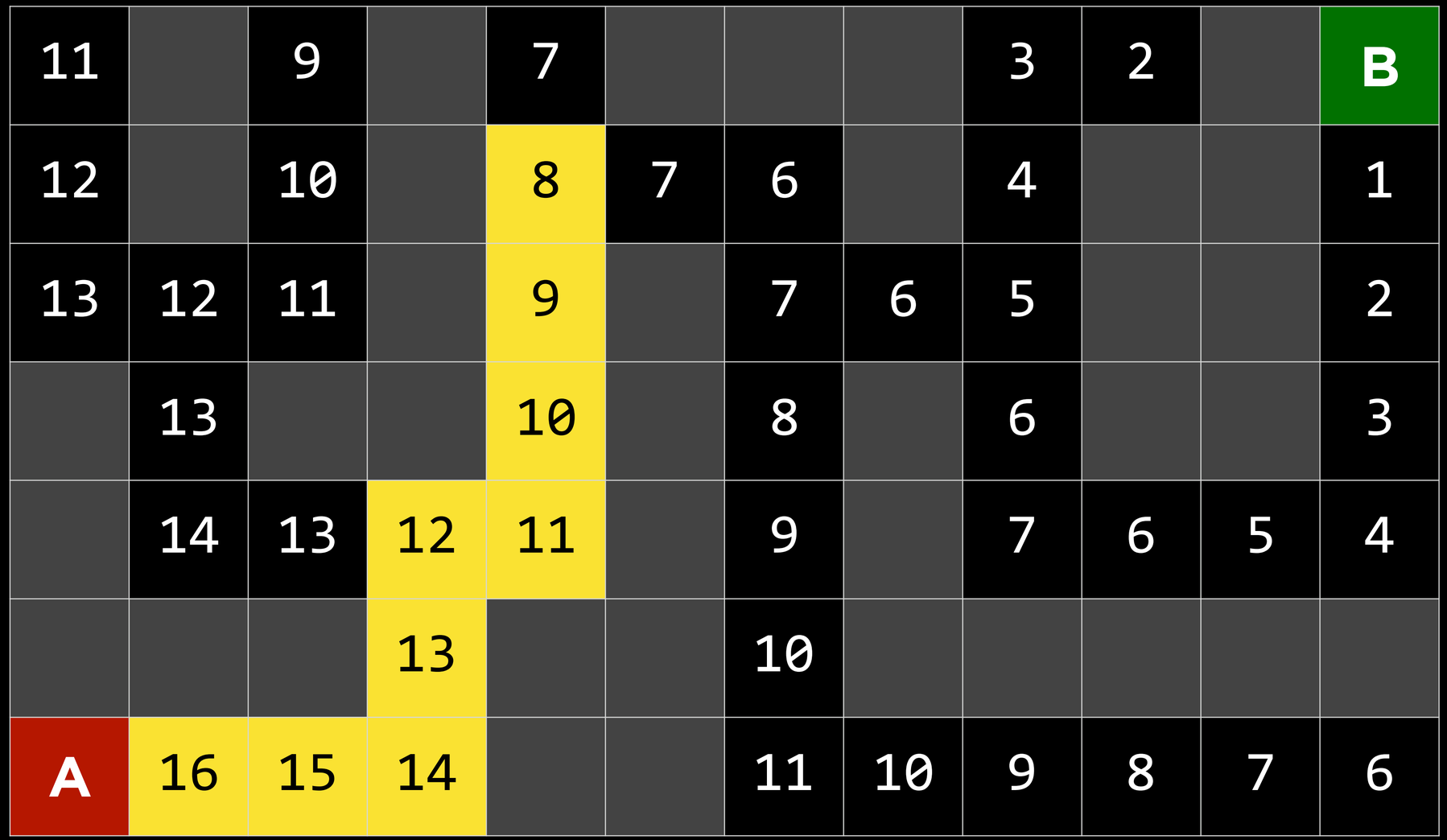
Greedy Best-First Search



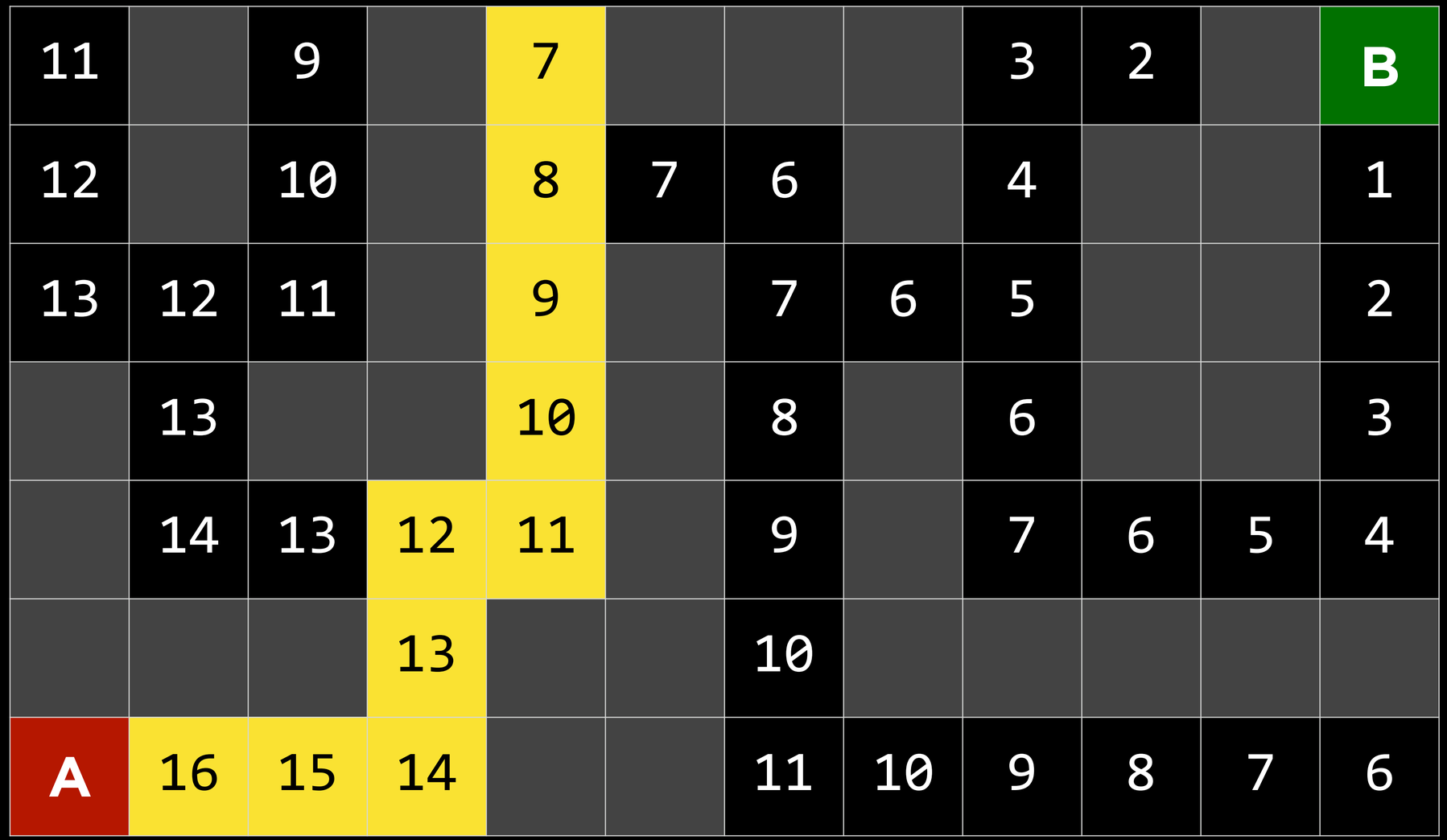
Greedy Best-First Search



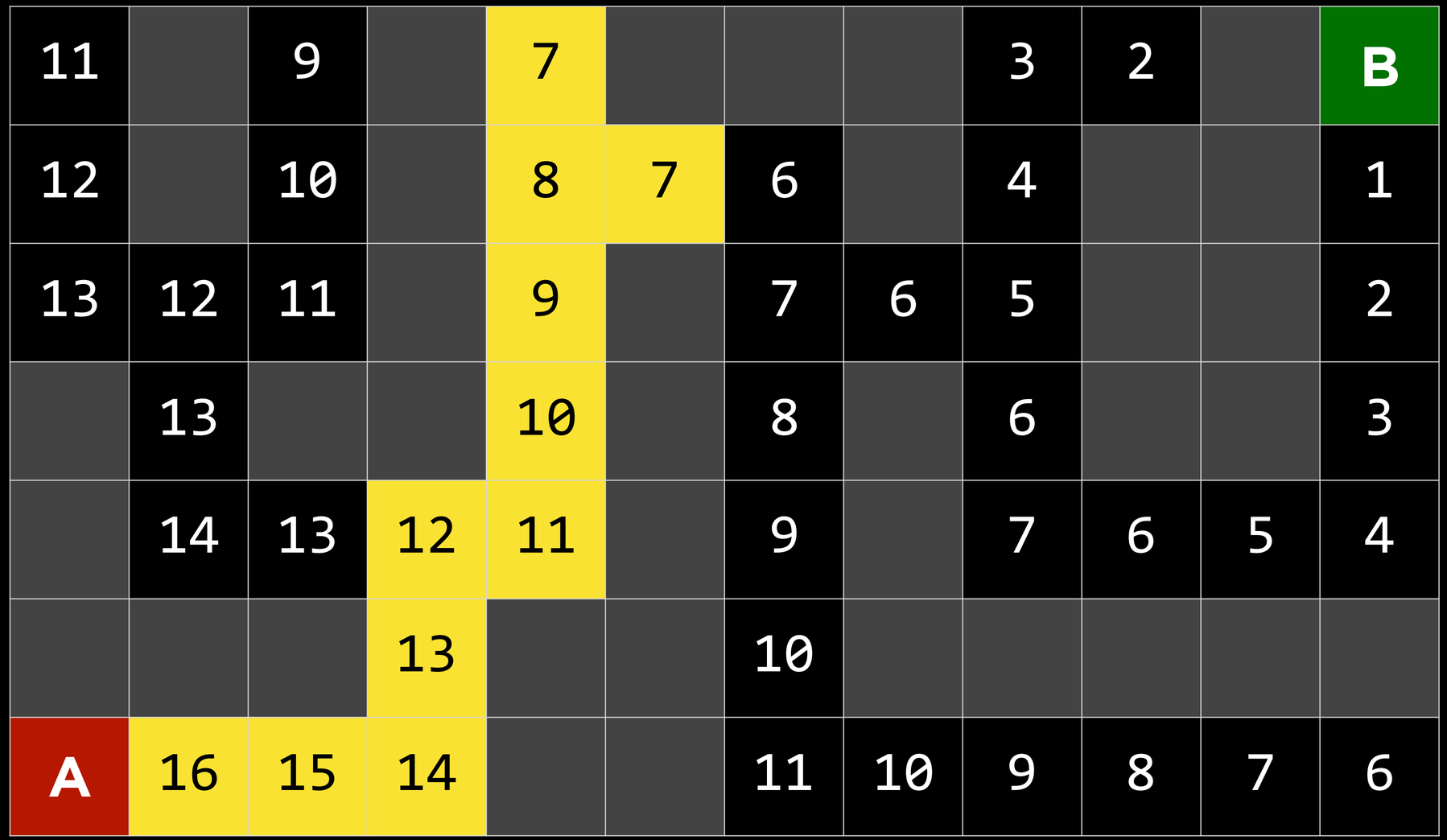
Greedy Best-First Search



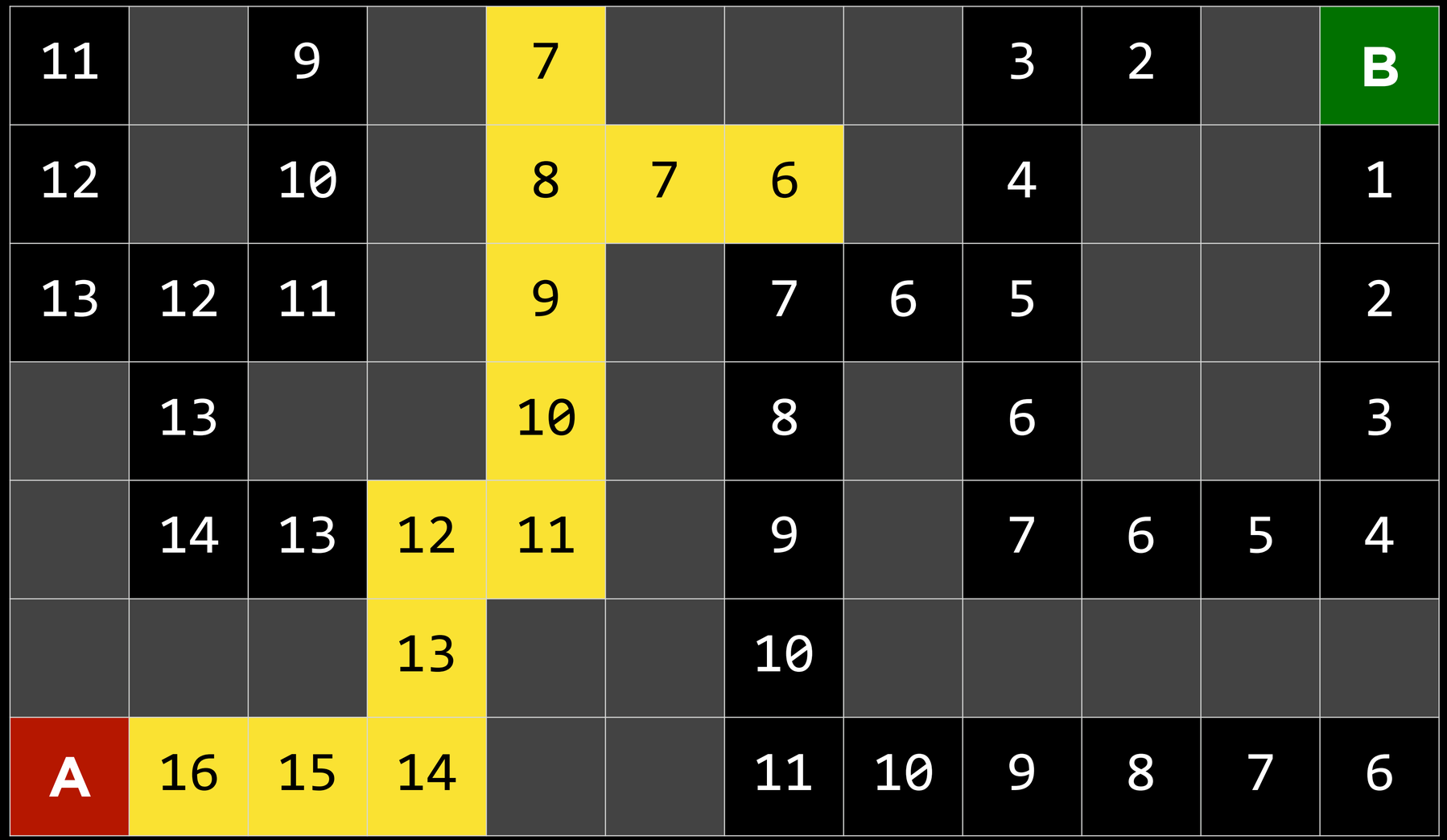
Greedy Best-First Search



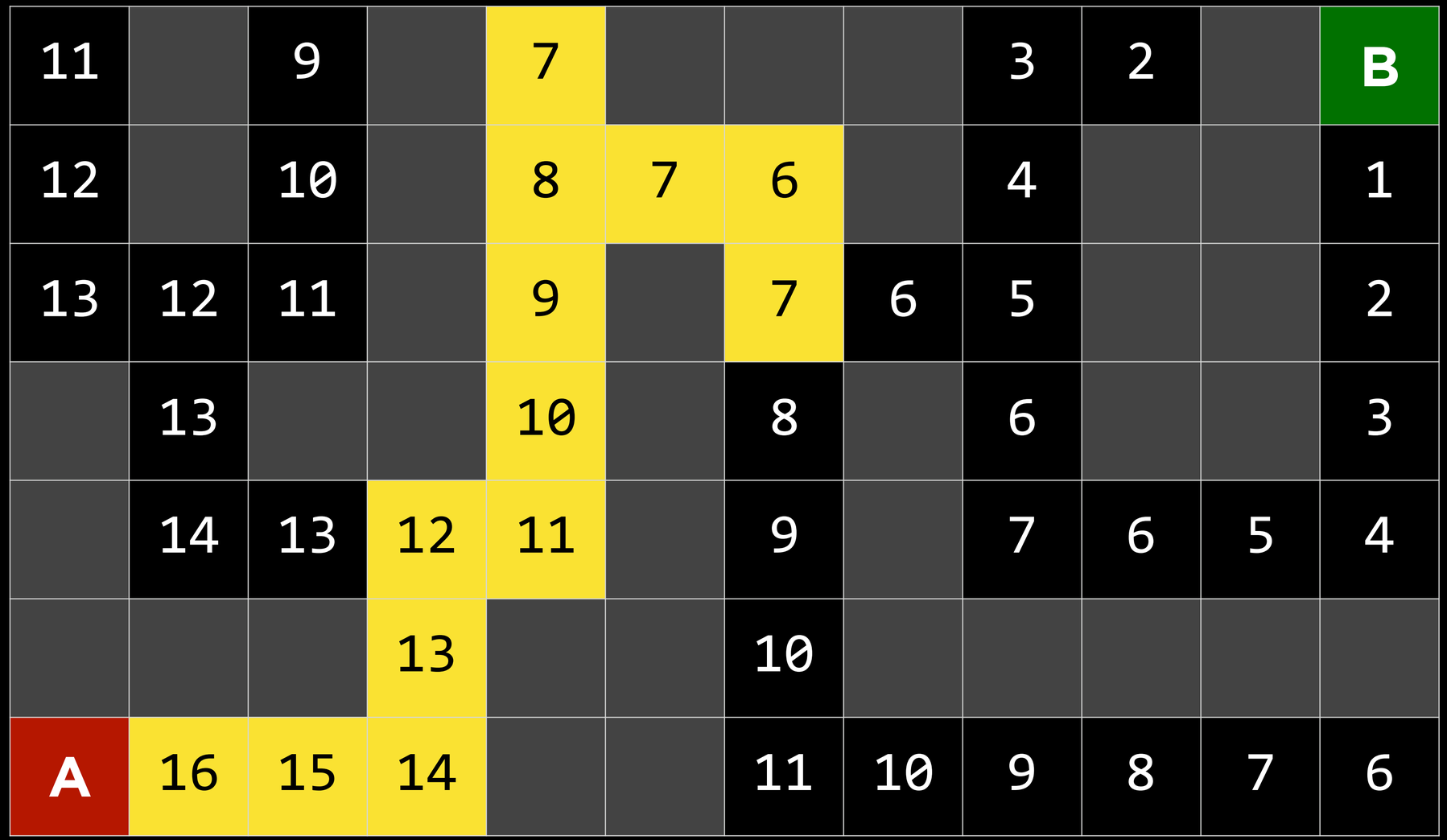
Greedy Best-First Search



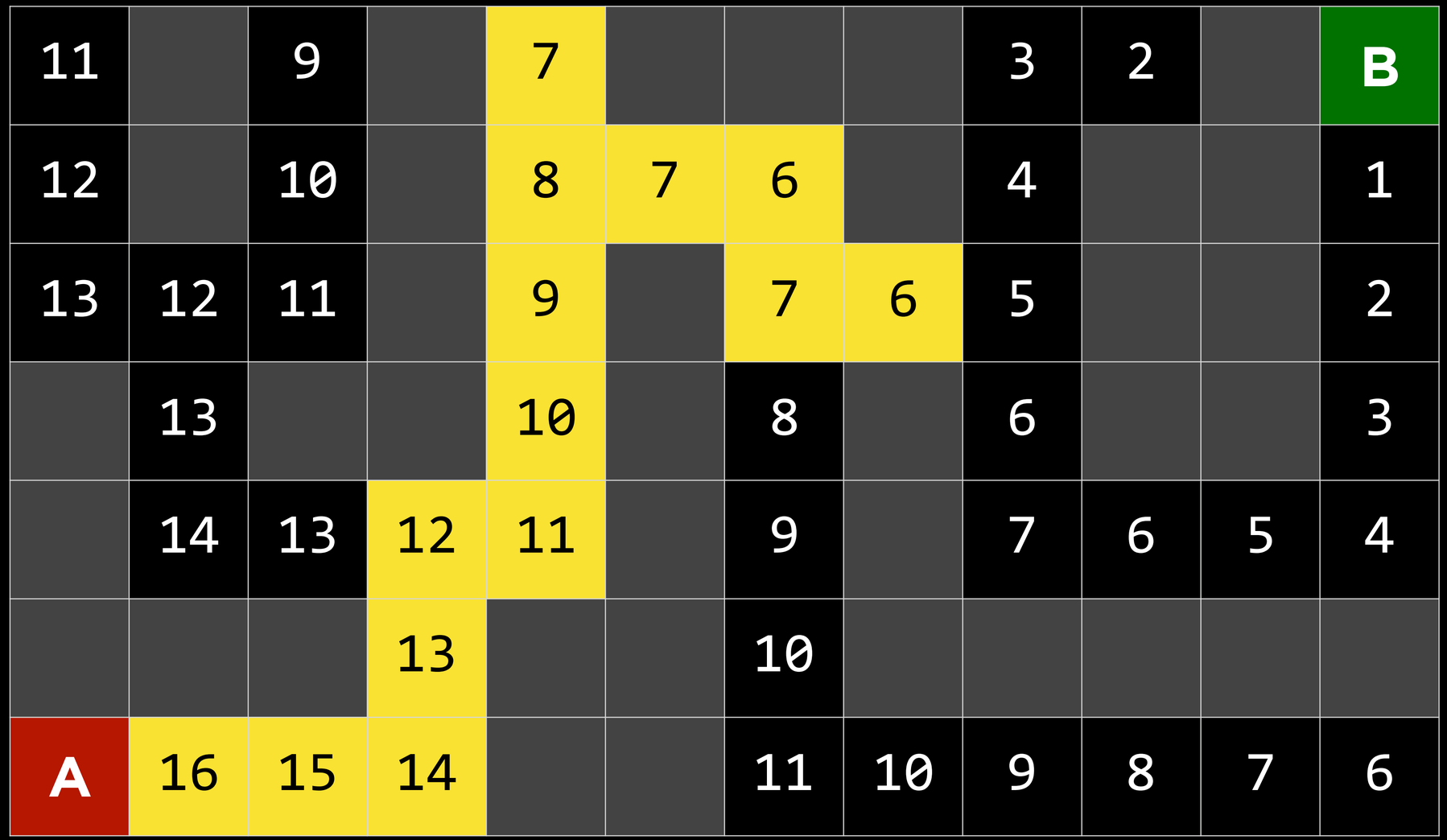
Greedy Best-First Search



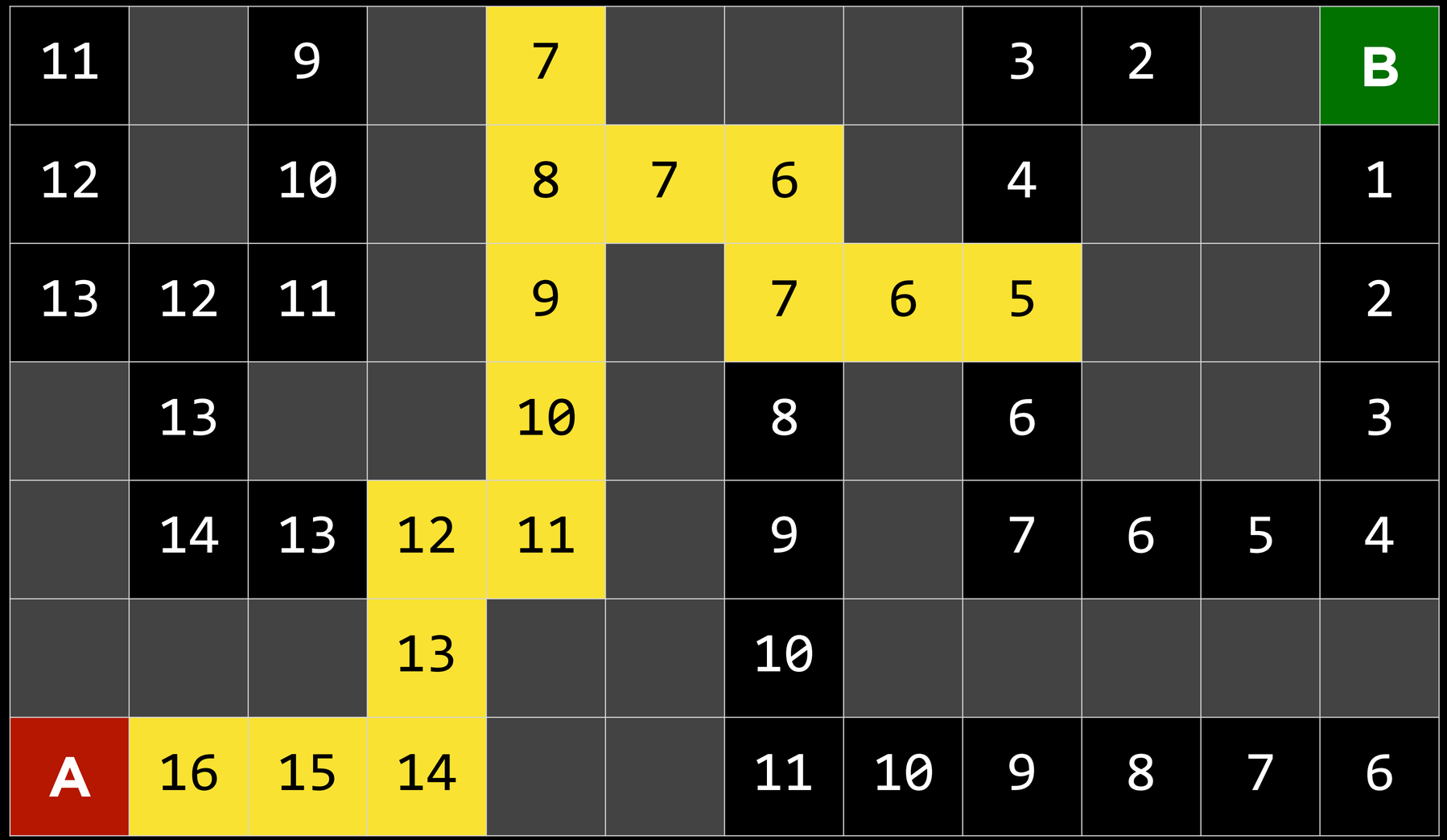
Greedy Best-First Search



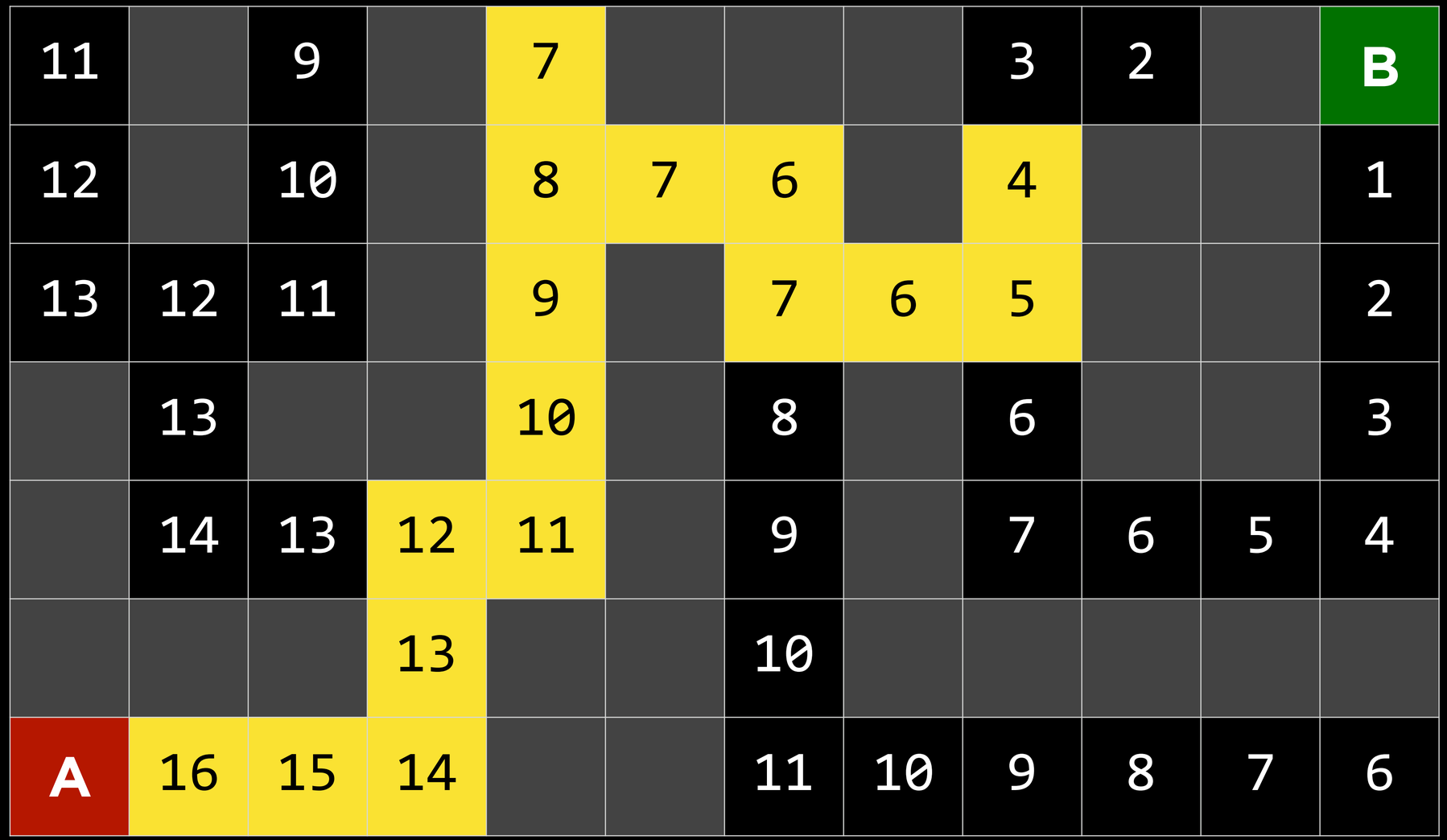
Greedy Best-First Search



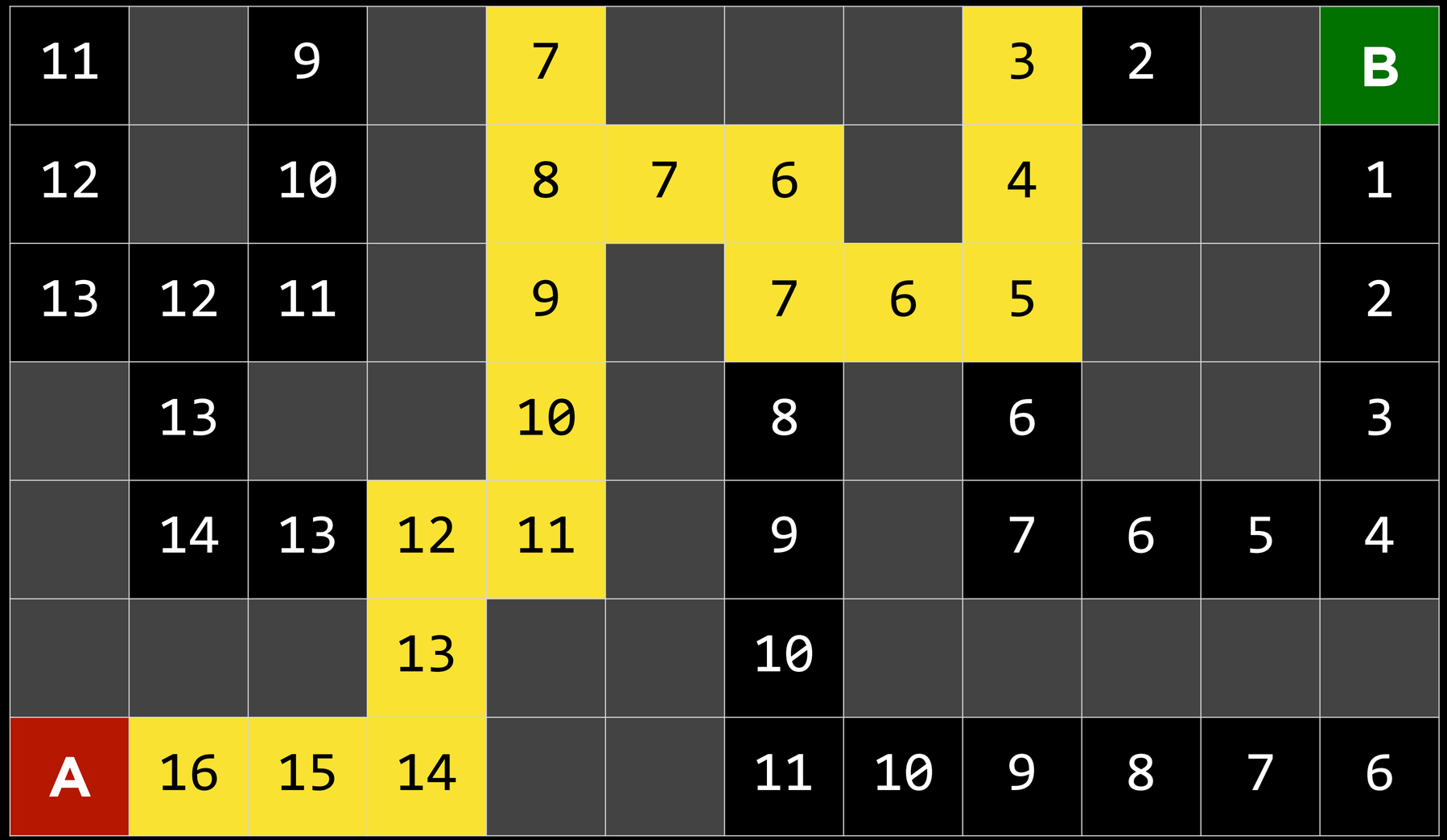
Greedy Best-First Search



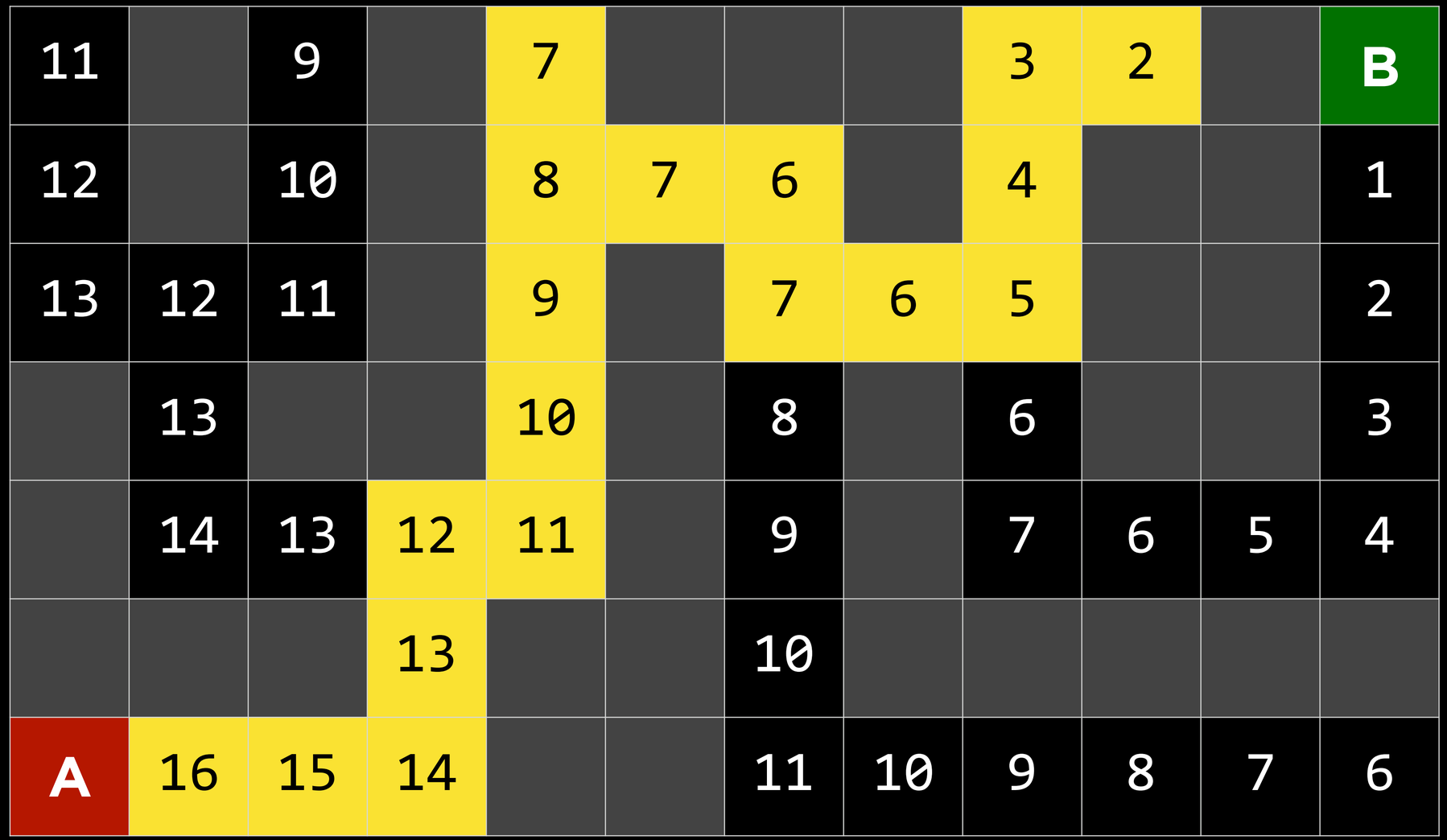
Greedy Best-First Search



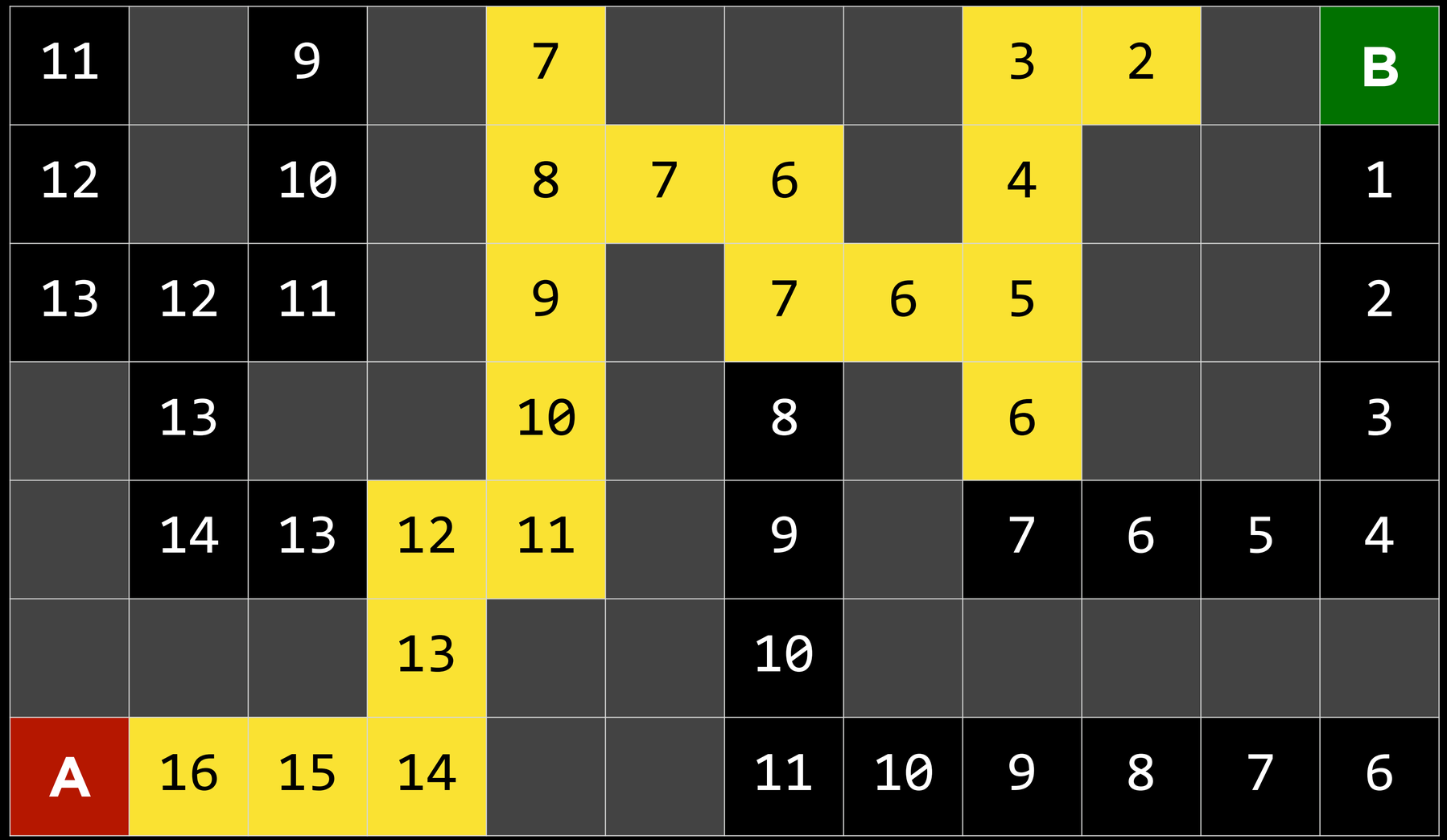
Greedy Best-First Search



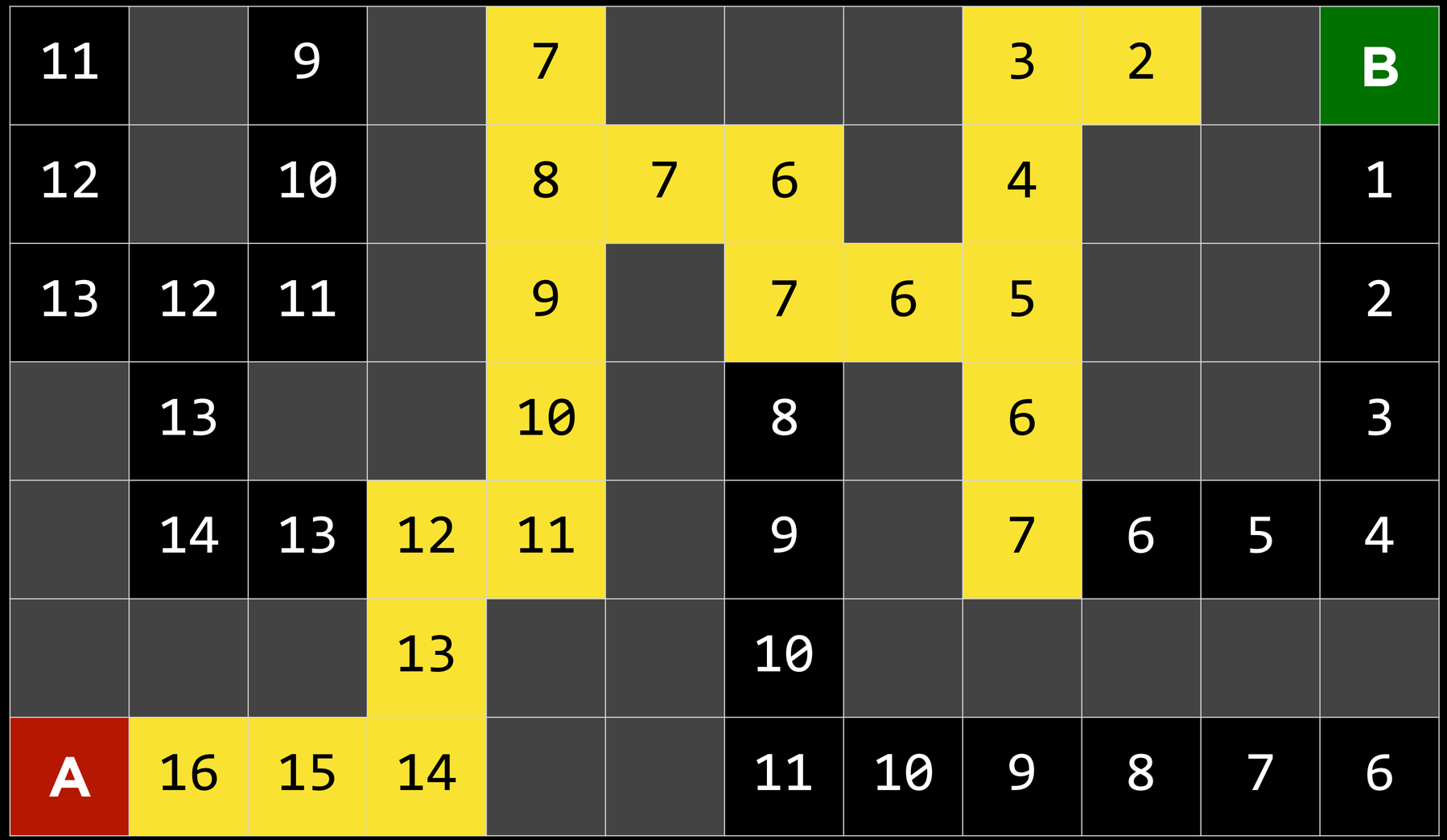
Greedy Best-First Search



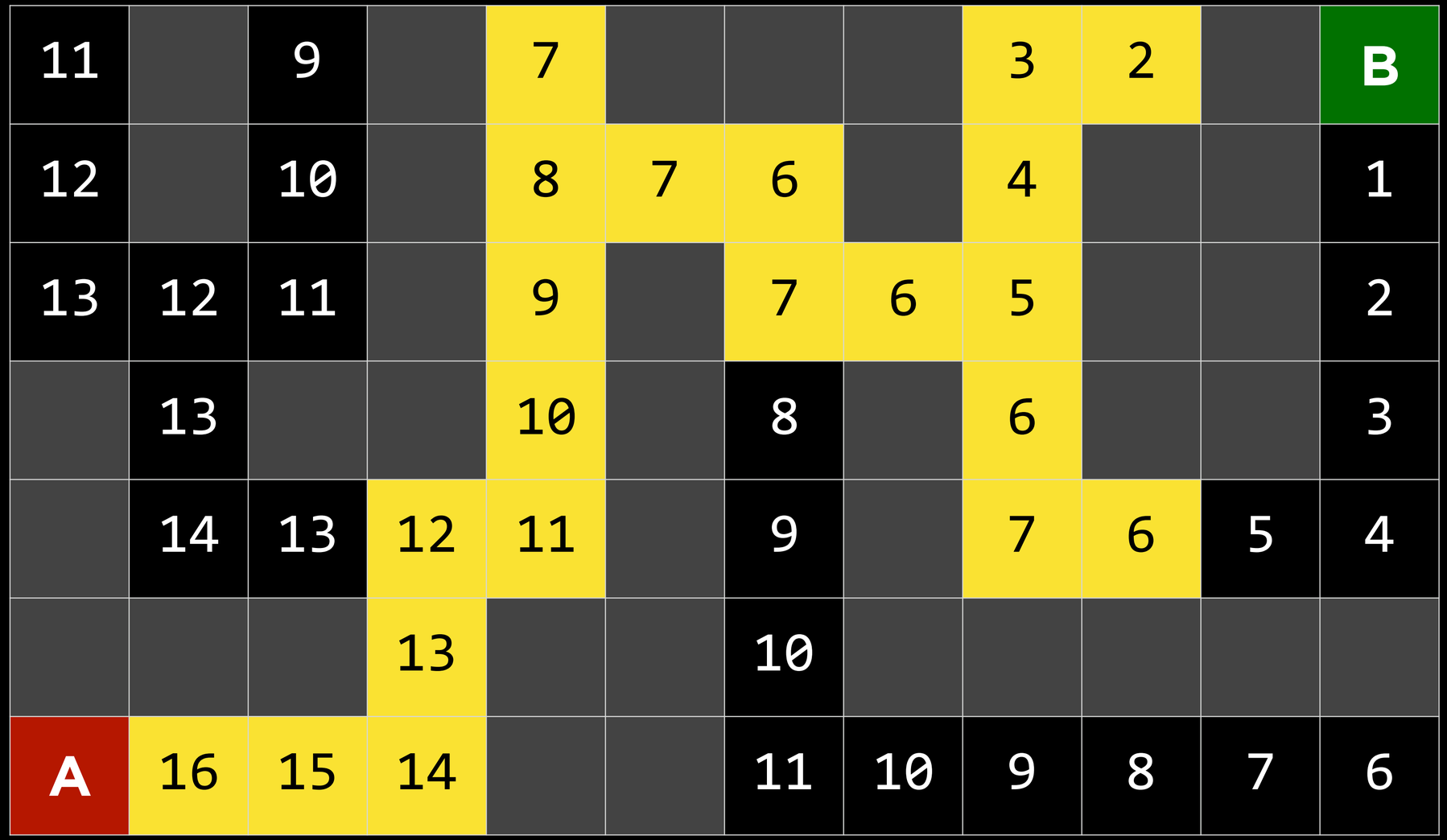
Greedy Best-First Search



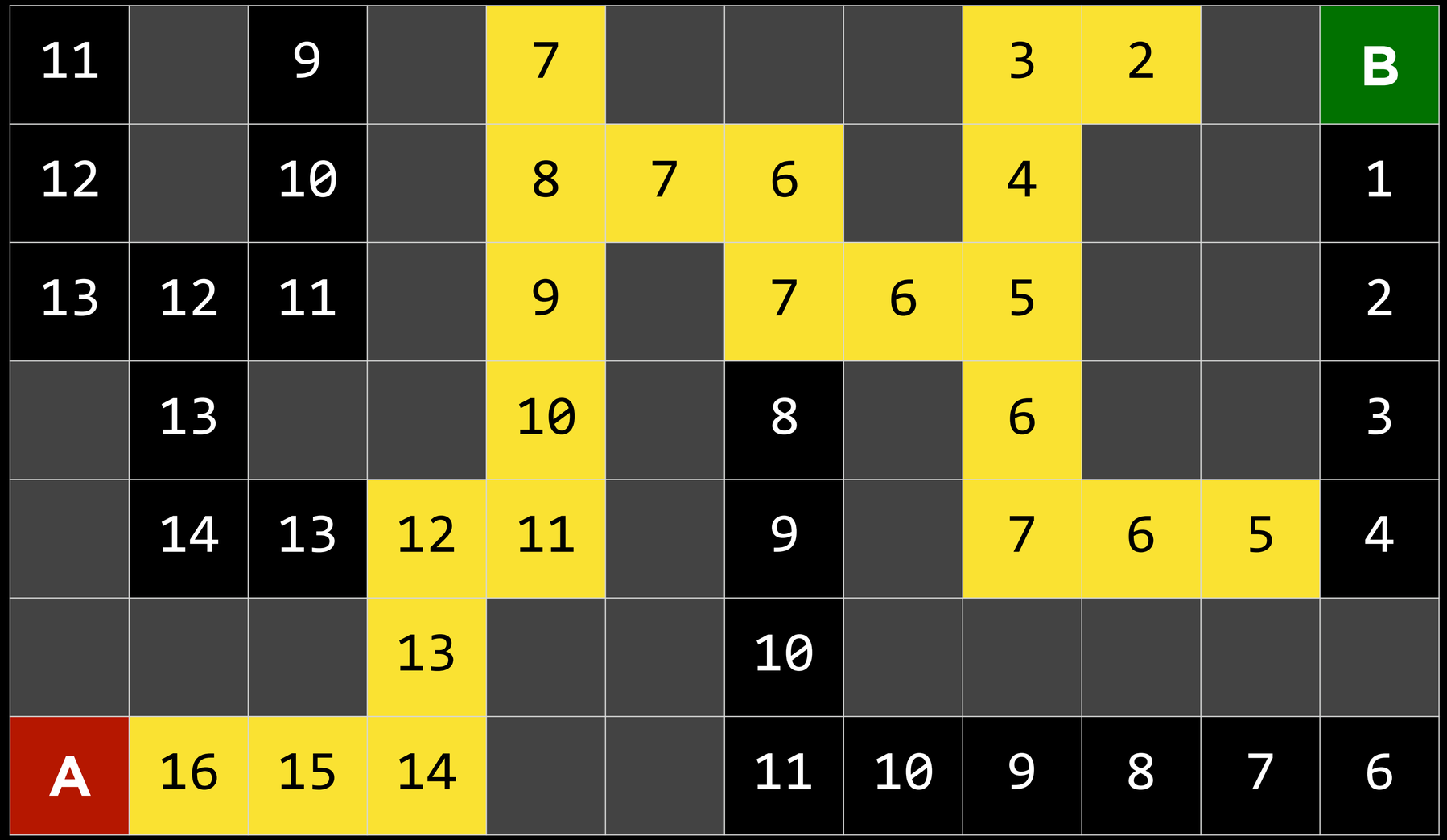
Greedy Best-First Search



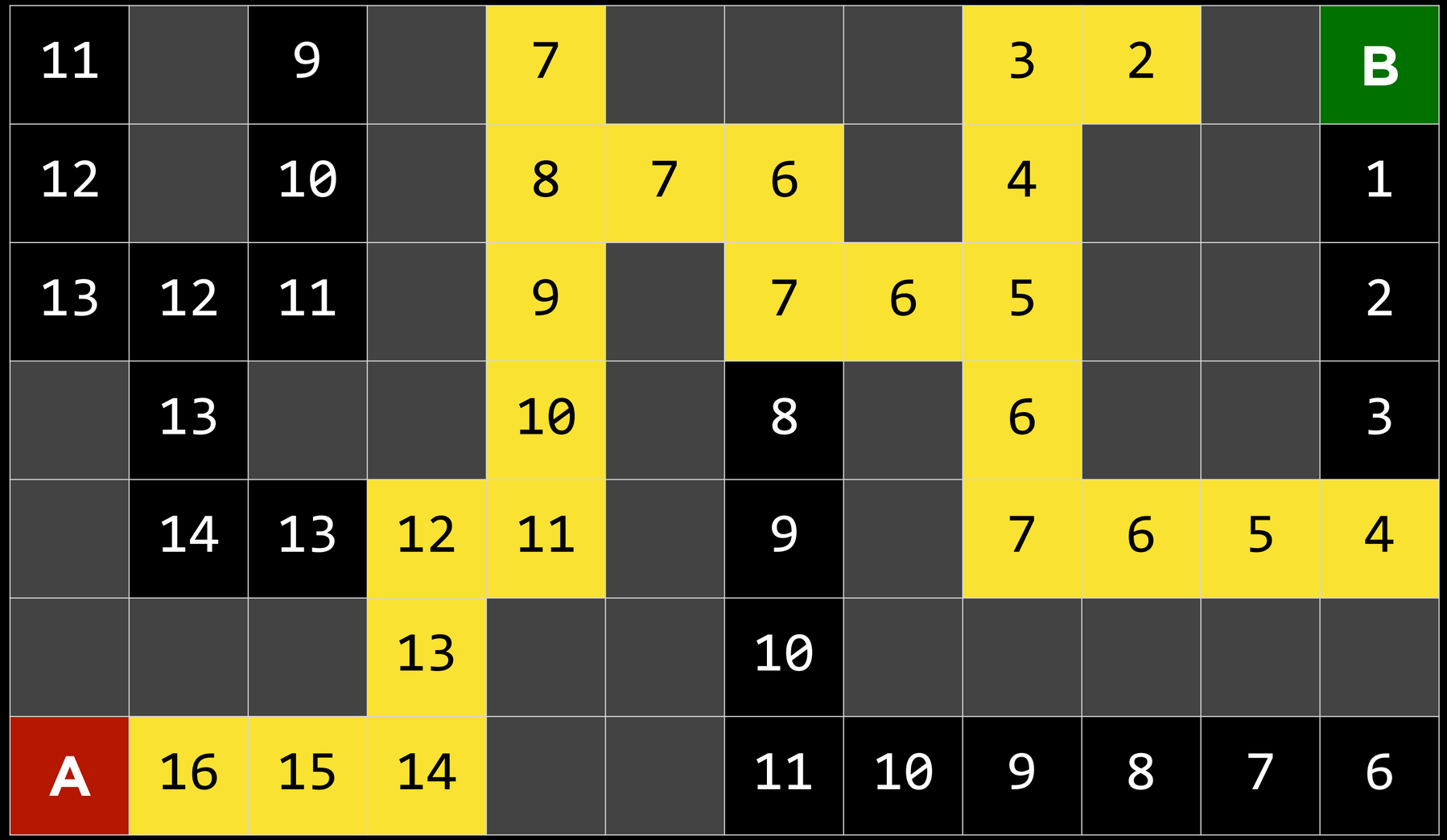
Greedy Best-First Search



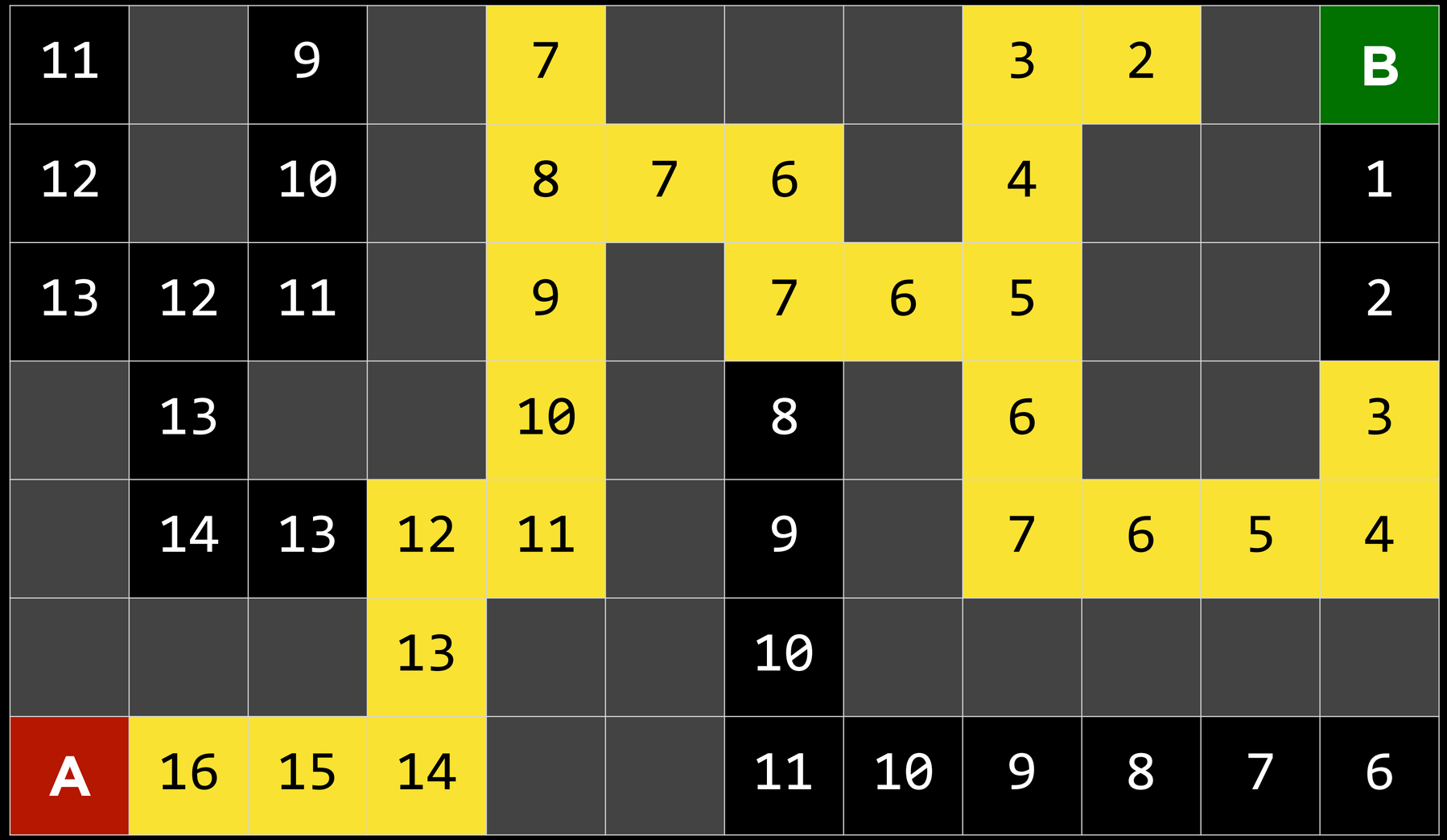
Greedy Best-First Search



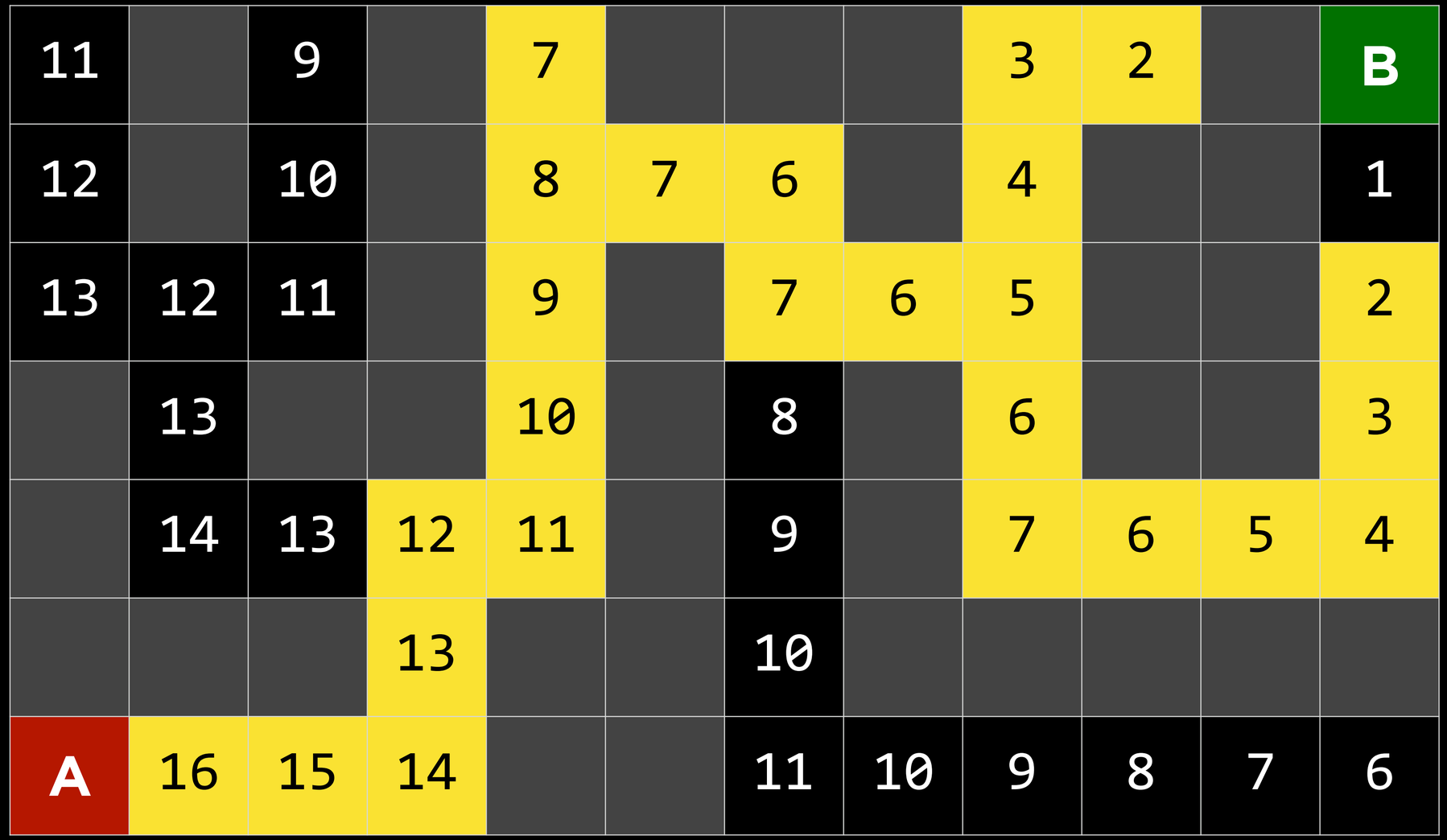
Greedy Best-First Search



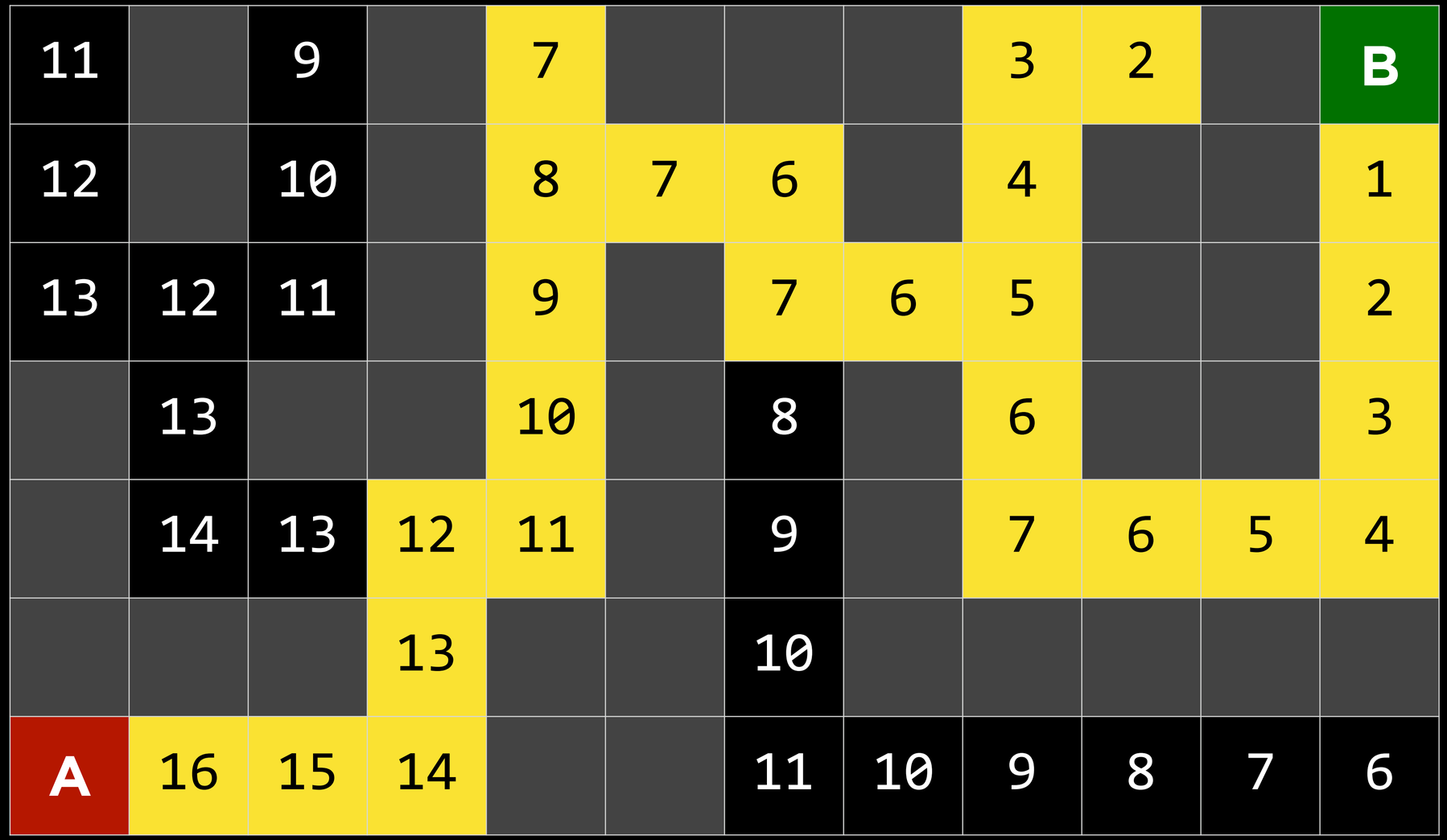
Greedy Best-First Search



Greedy Best-First Search



Greedy Best-First Search



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

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						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, $h(n) \leq h(n') + c$)