COMPX216/Y05337 Artificial Intelligence

Informed search

Credits: Prof. Eibe Frank (UoW)

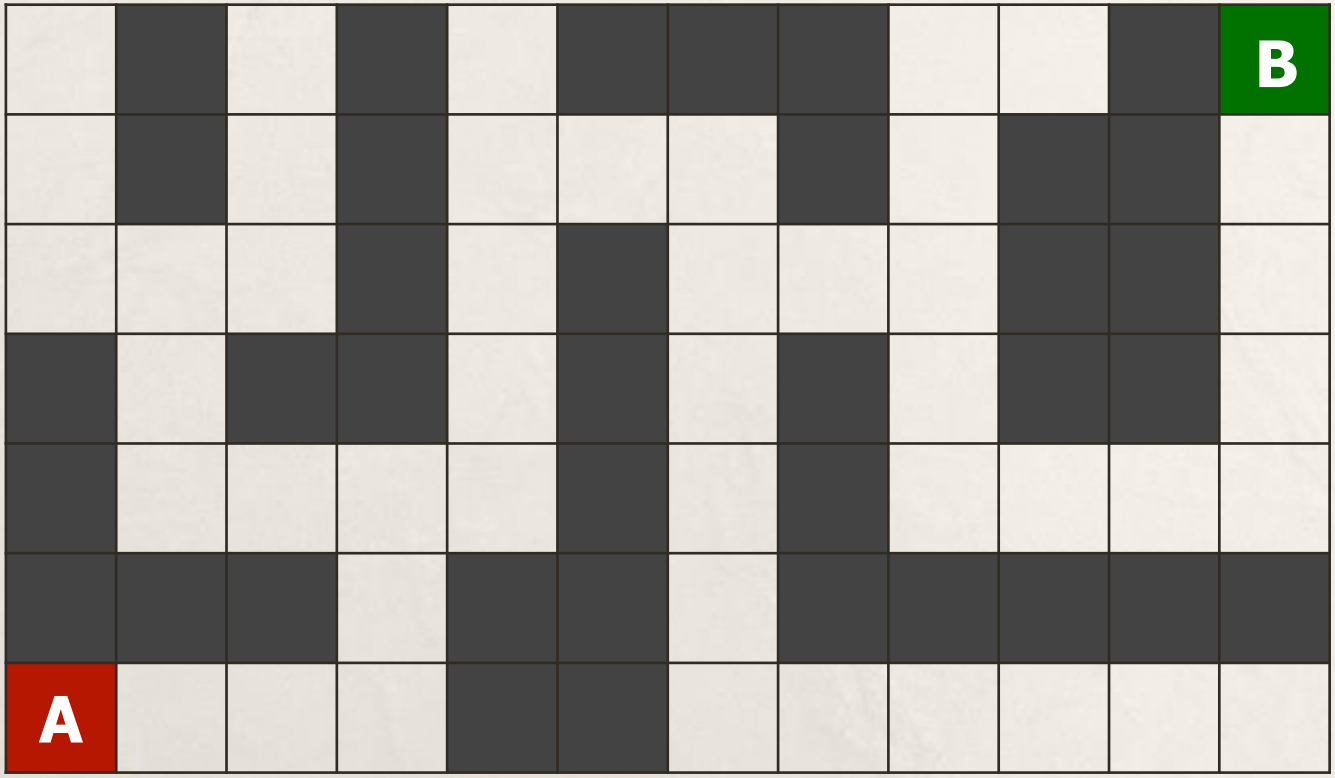
Today: Informed search

- Providing hints for the search using a heuristic
- Greedy best-first search
- A* search
- Admissible heuristics
- Consistent heuristics
- Search contours
- Memory-bounded search: beam search
- Heuristics for the 8-puzzle

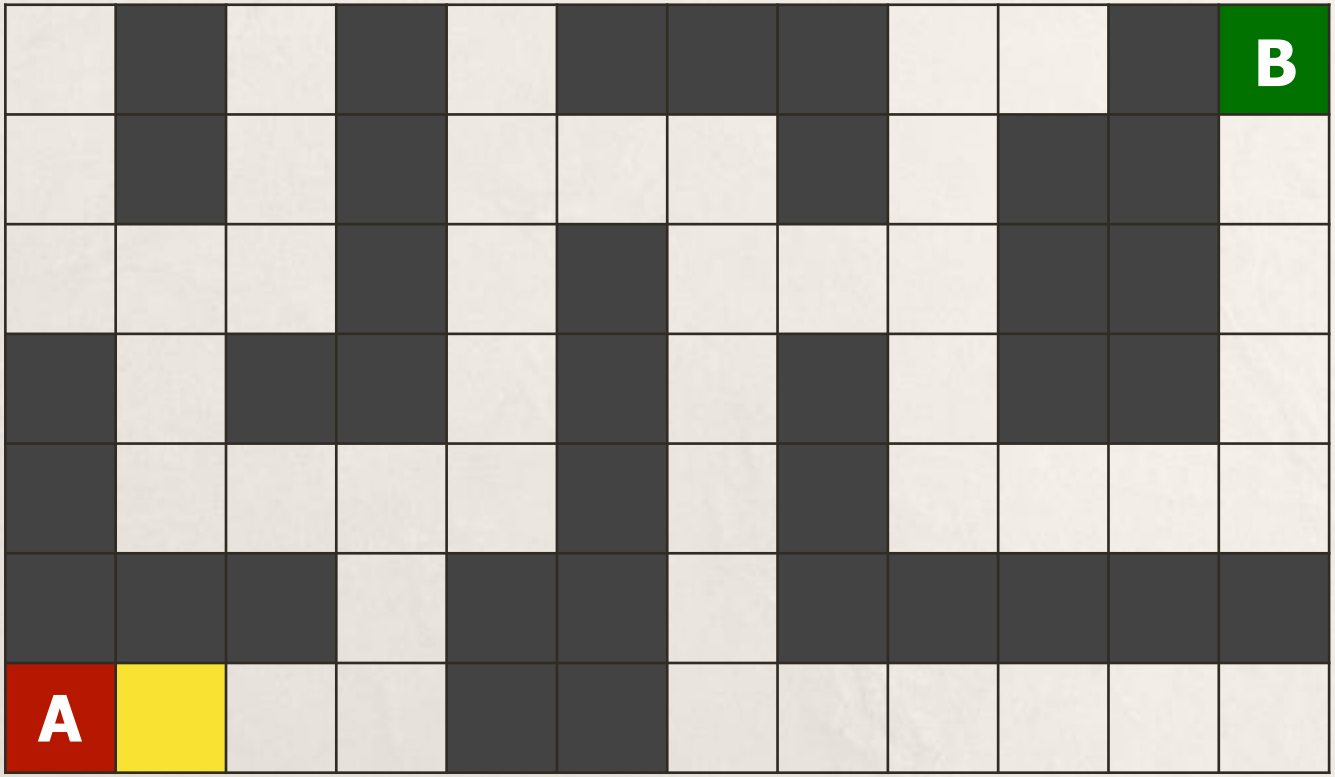
Uninformed search

- Search strategy that uses no problem-specific knowledge

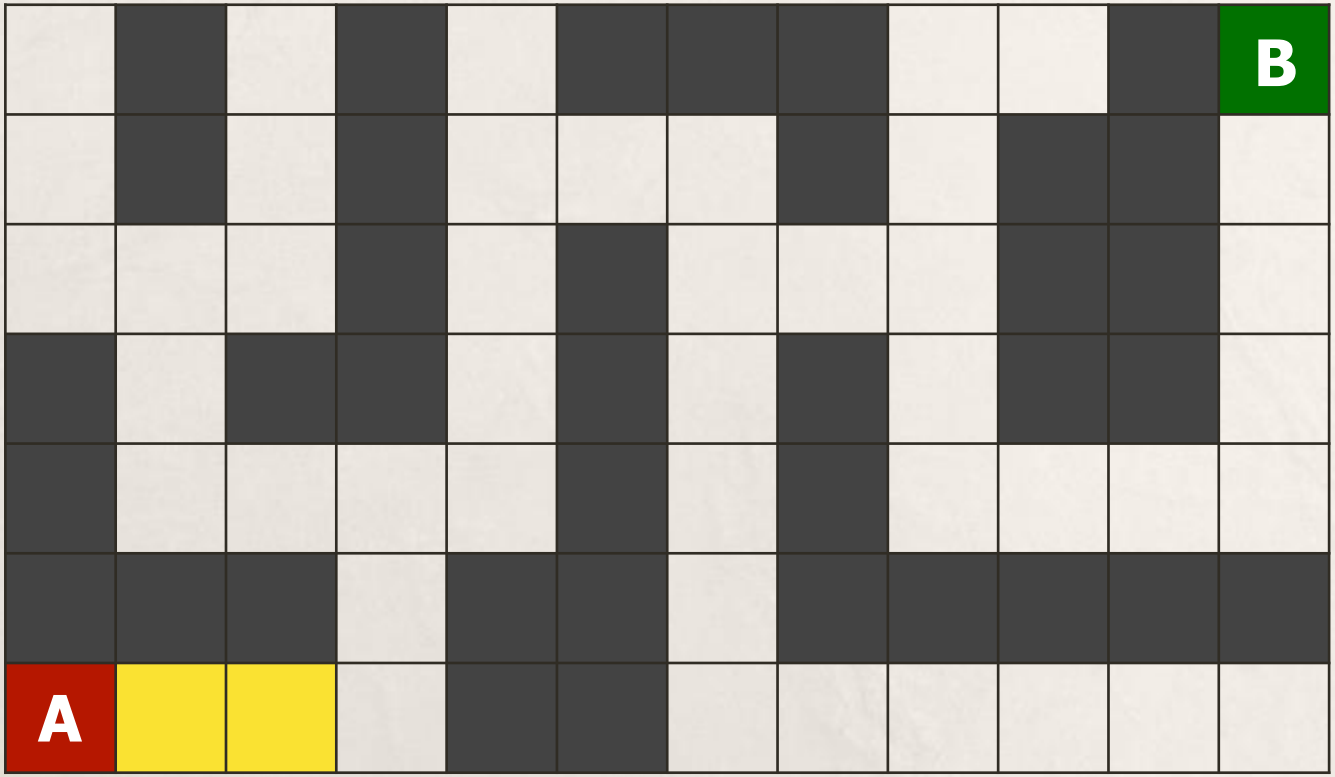
Depth-First Search



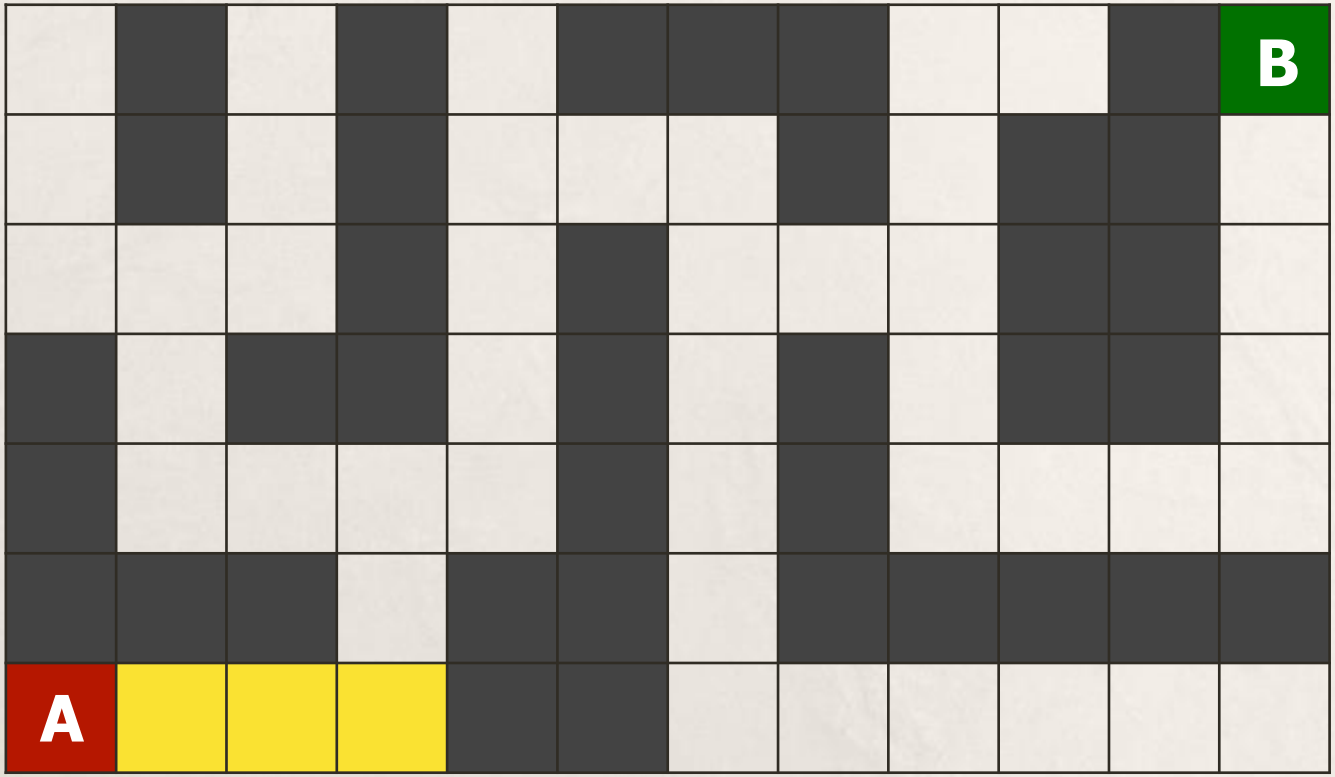
Depth-First Search



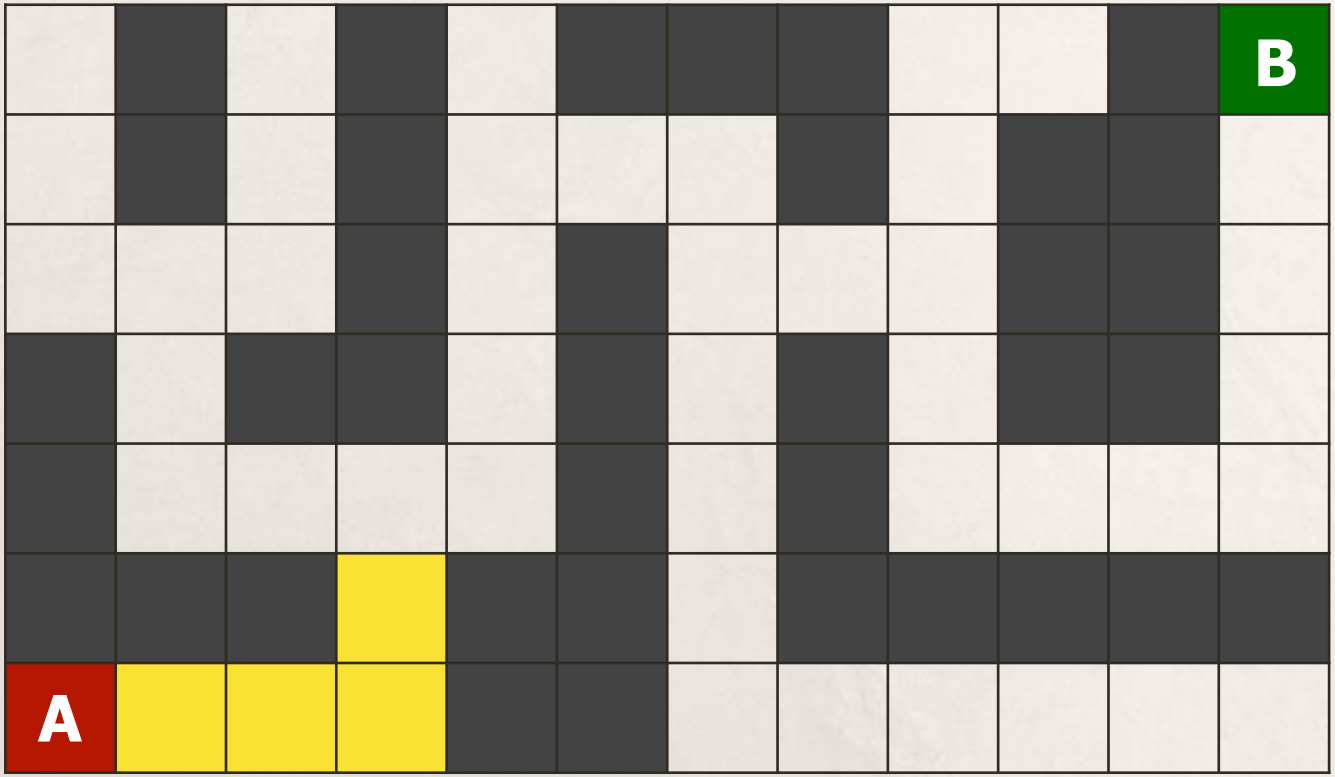
Depth-First Search



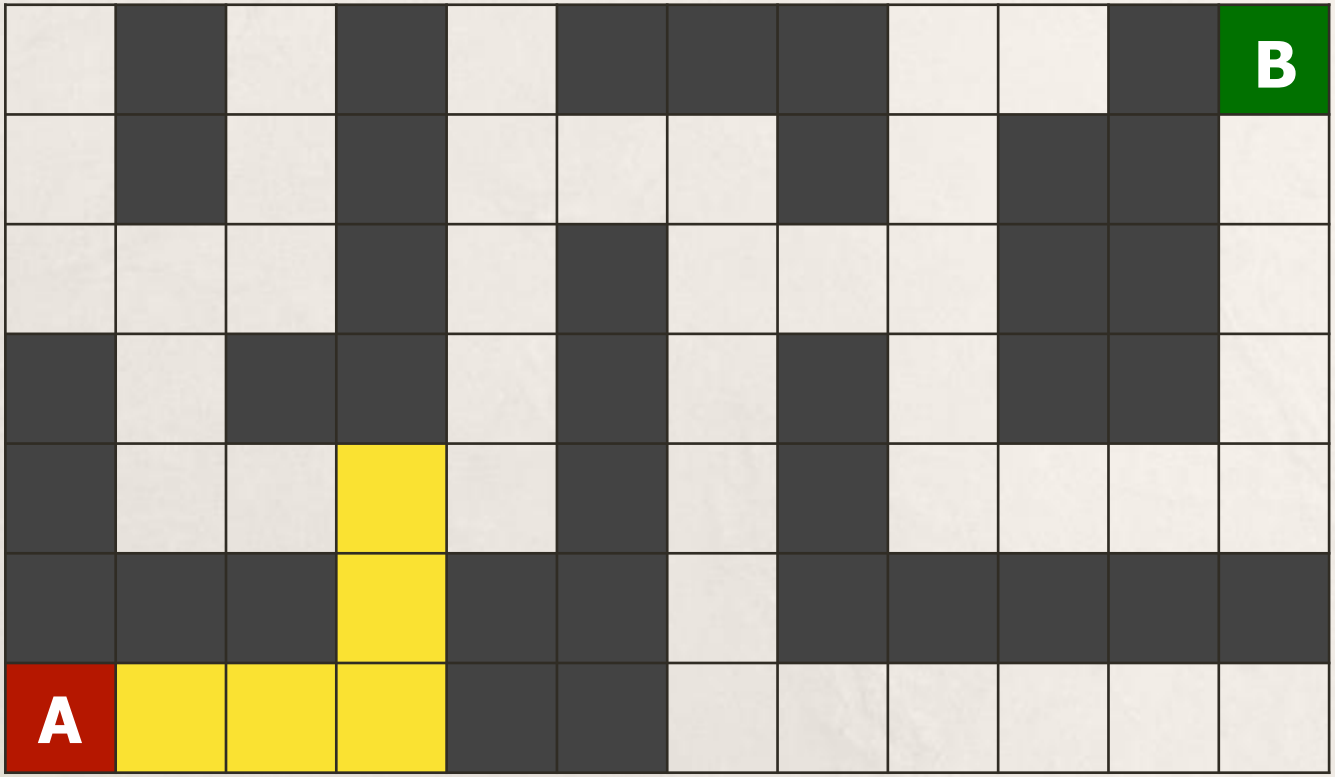
Depth-First Search



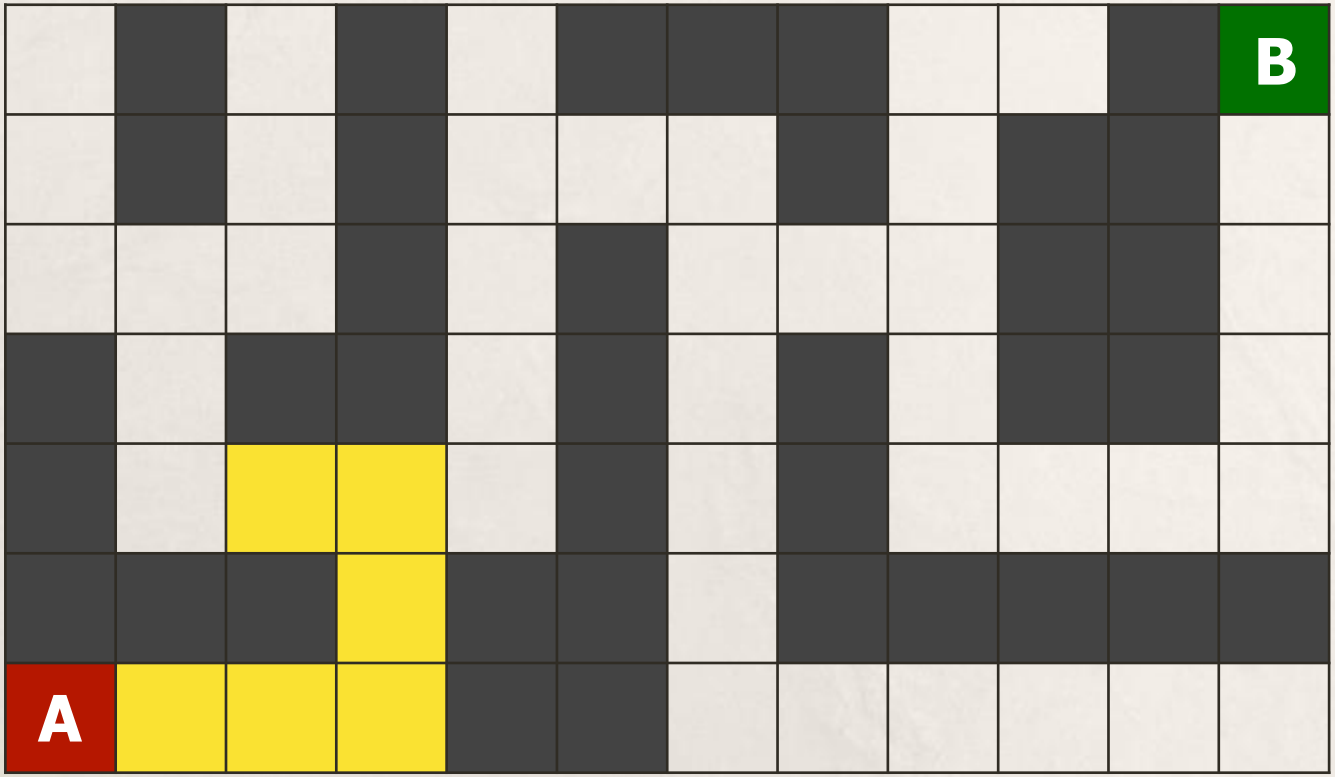
Depth-First Search



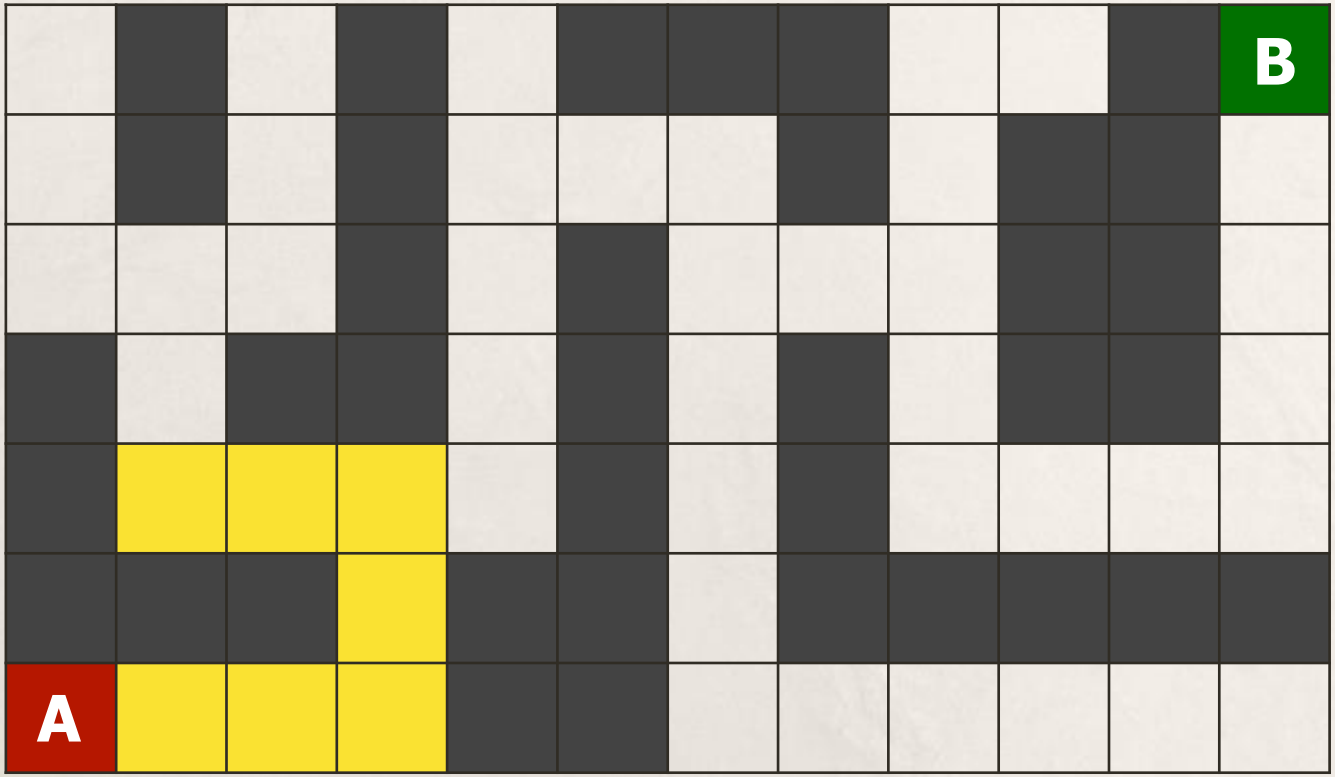
Depth-First Search



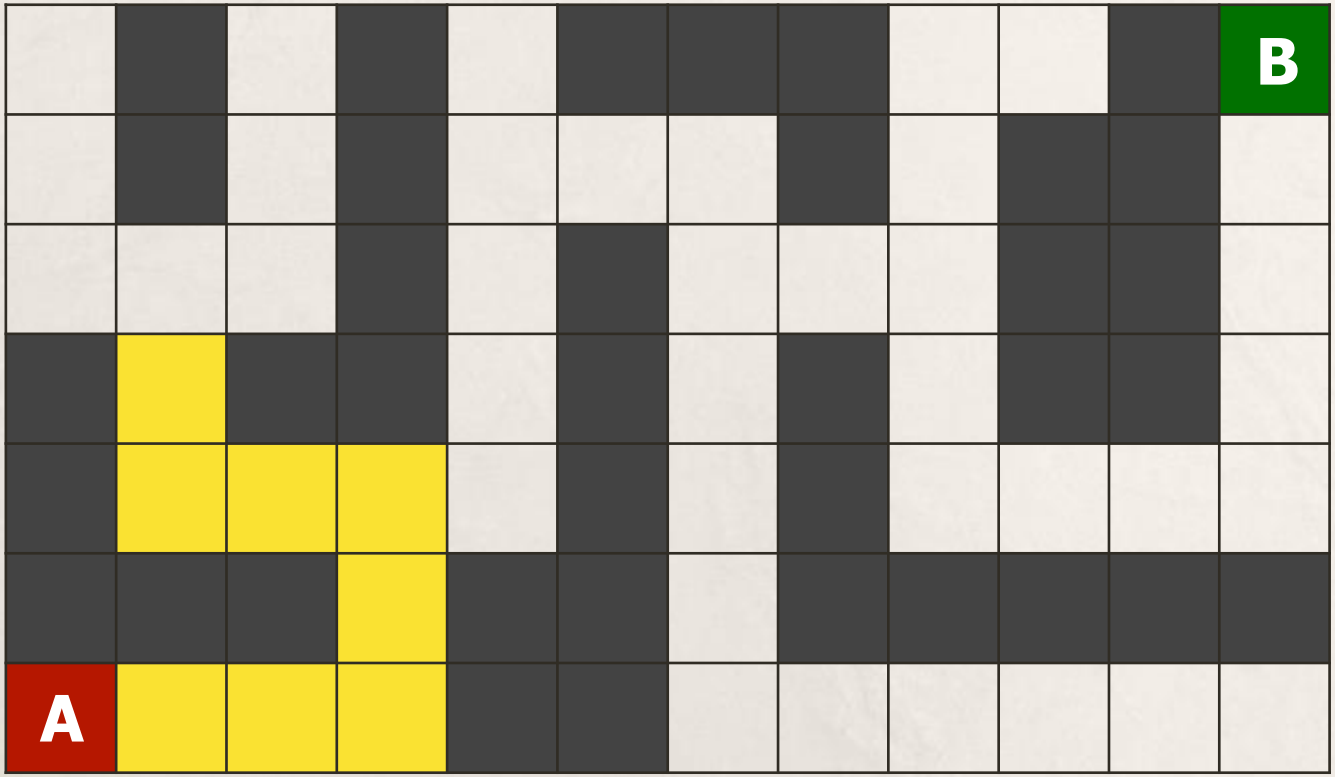
Depth-First Search



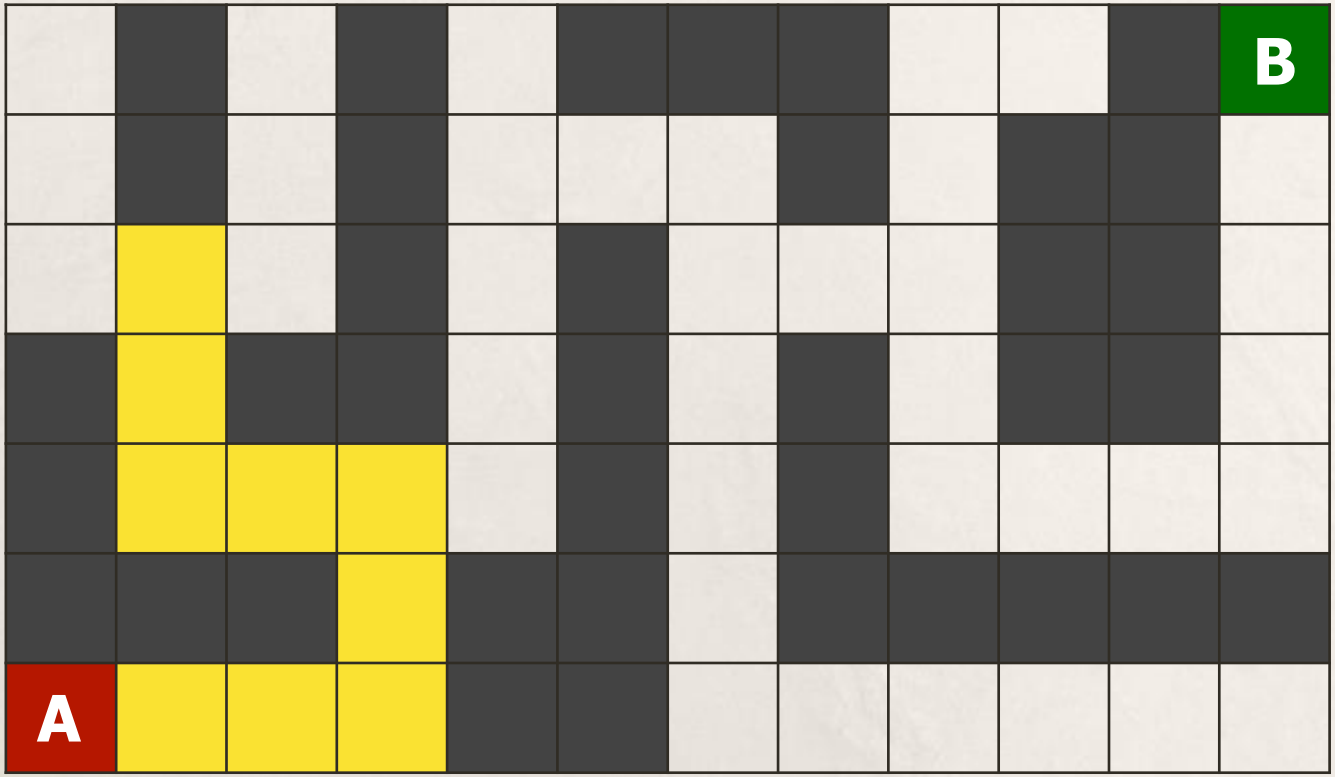
Depth-First Search



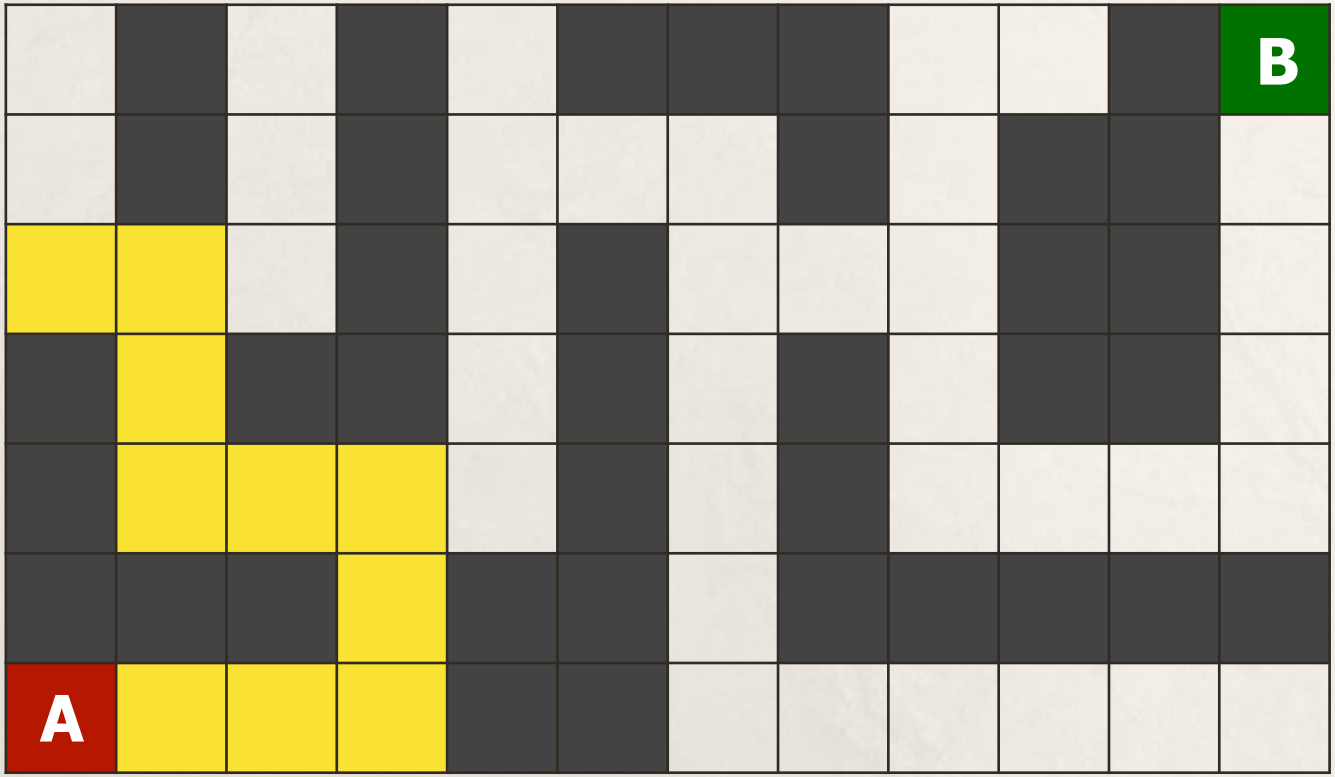
Depth-First Search



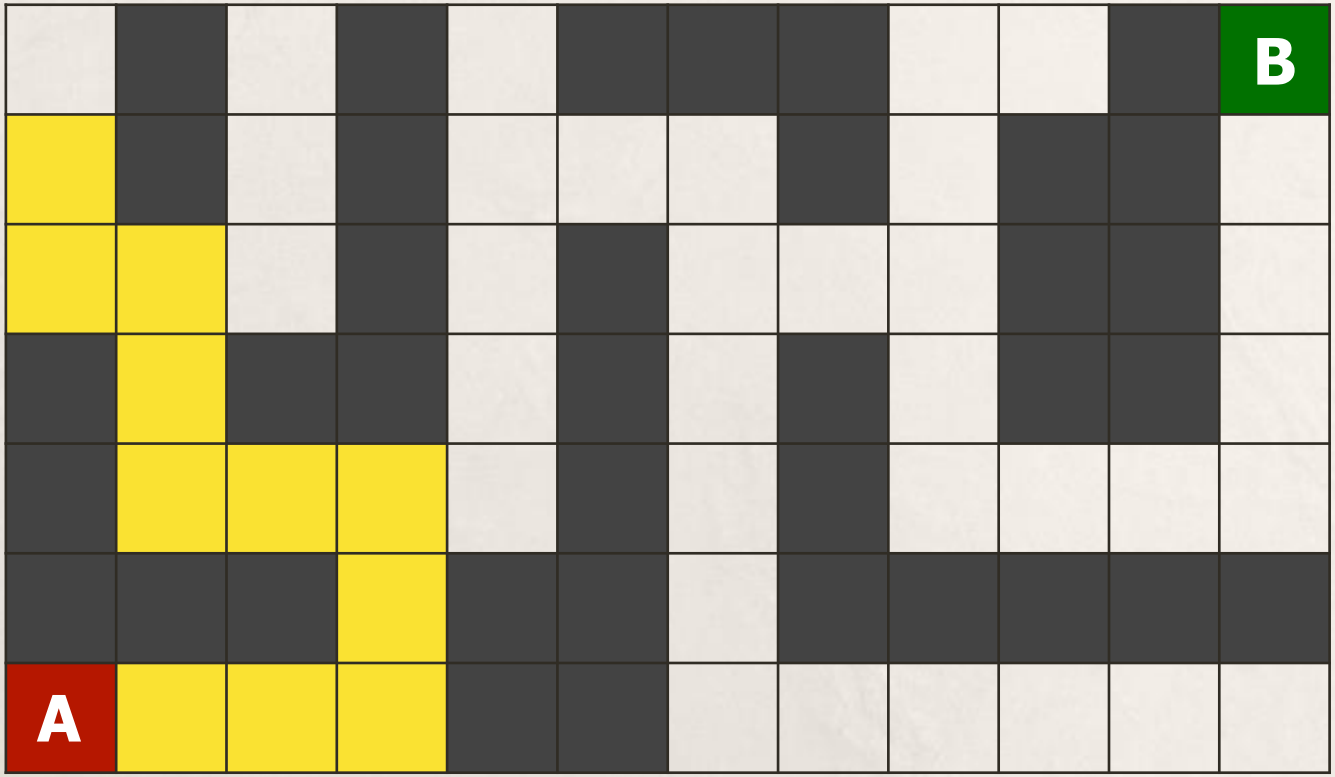
Depth-First Search



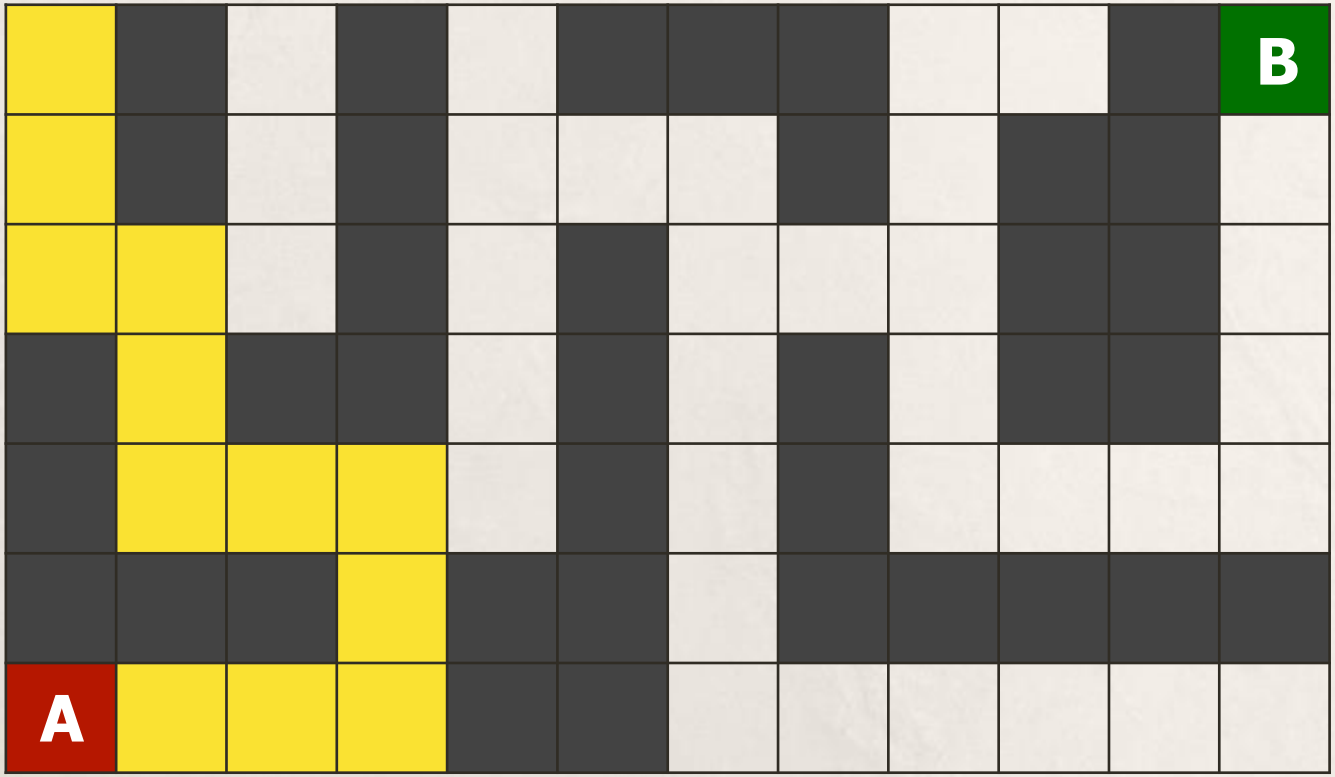
Depth-First Search



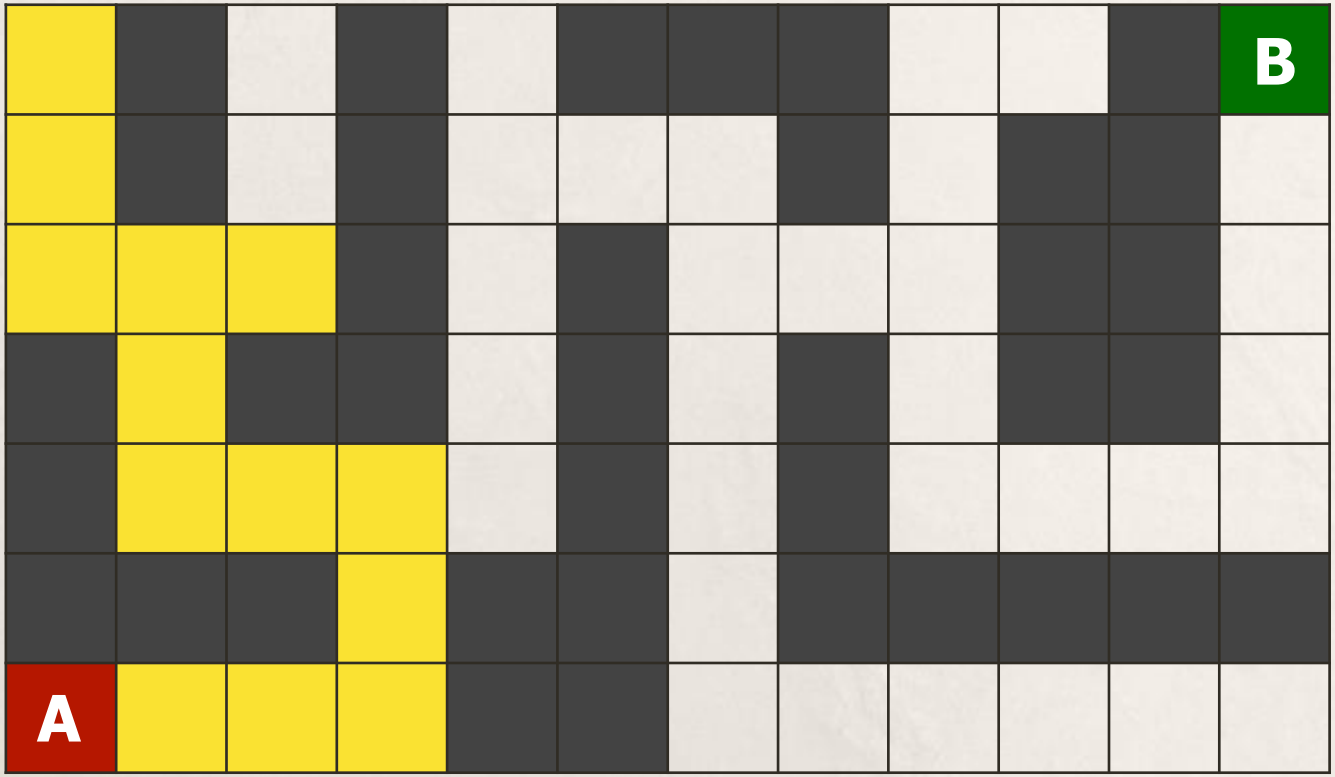
Depth-First Search



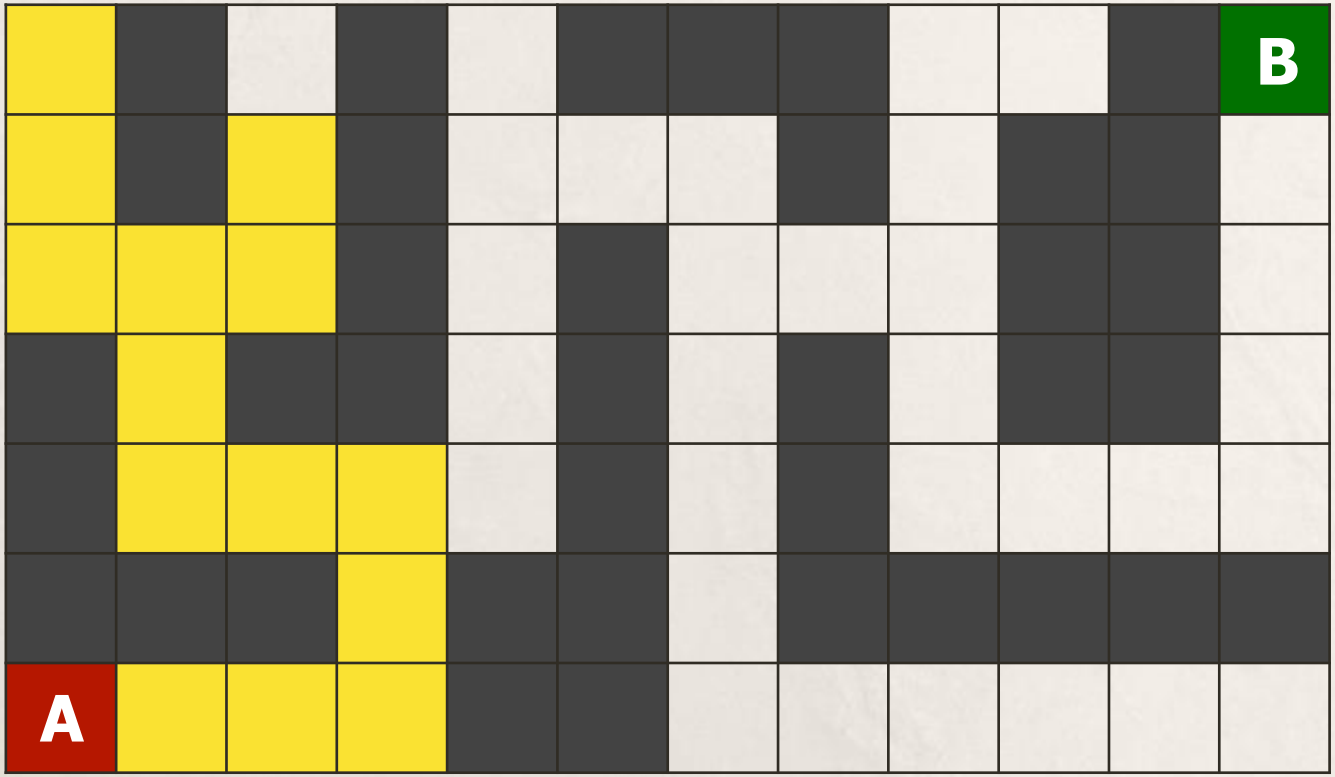
Depth-First Search



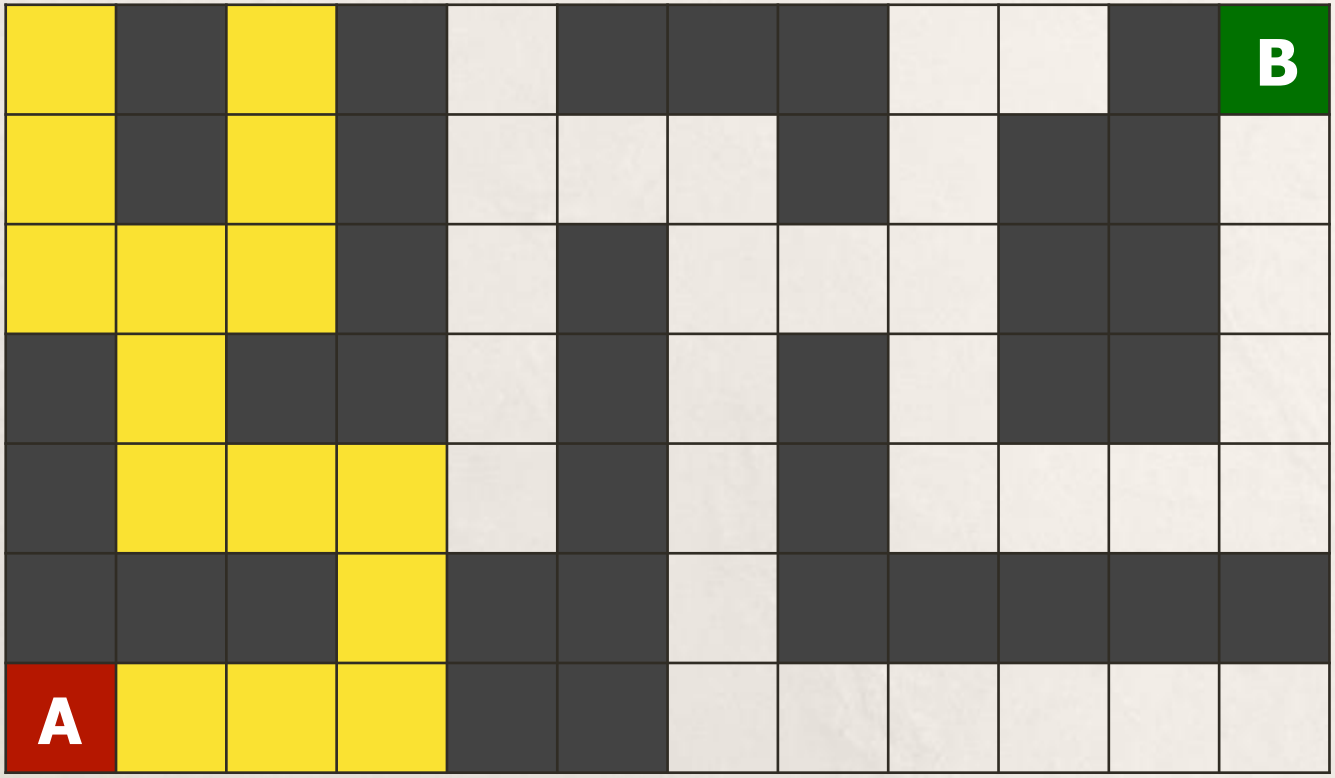
Depth-First Search



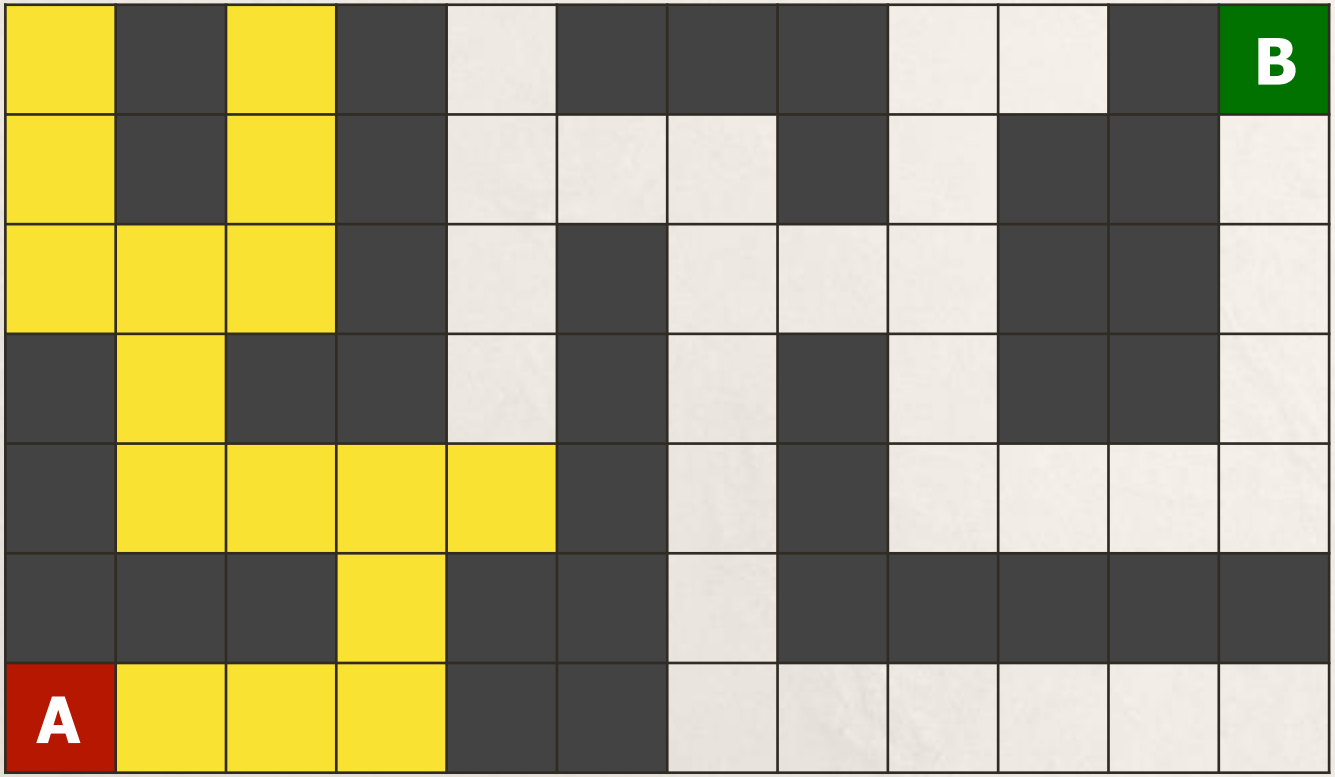
Depth-First Search



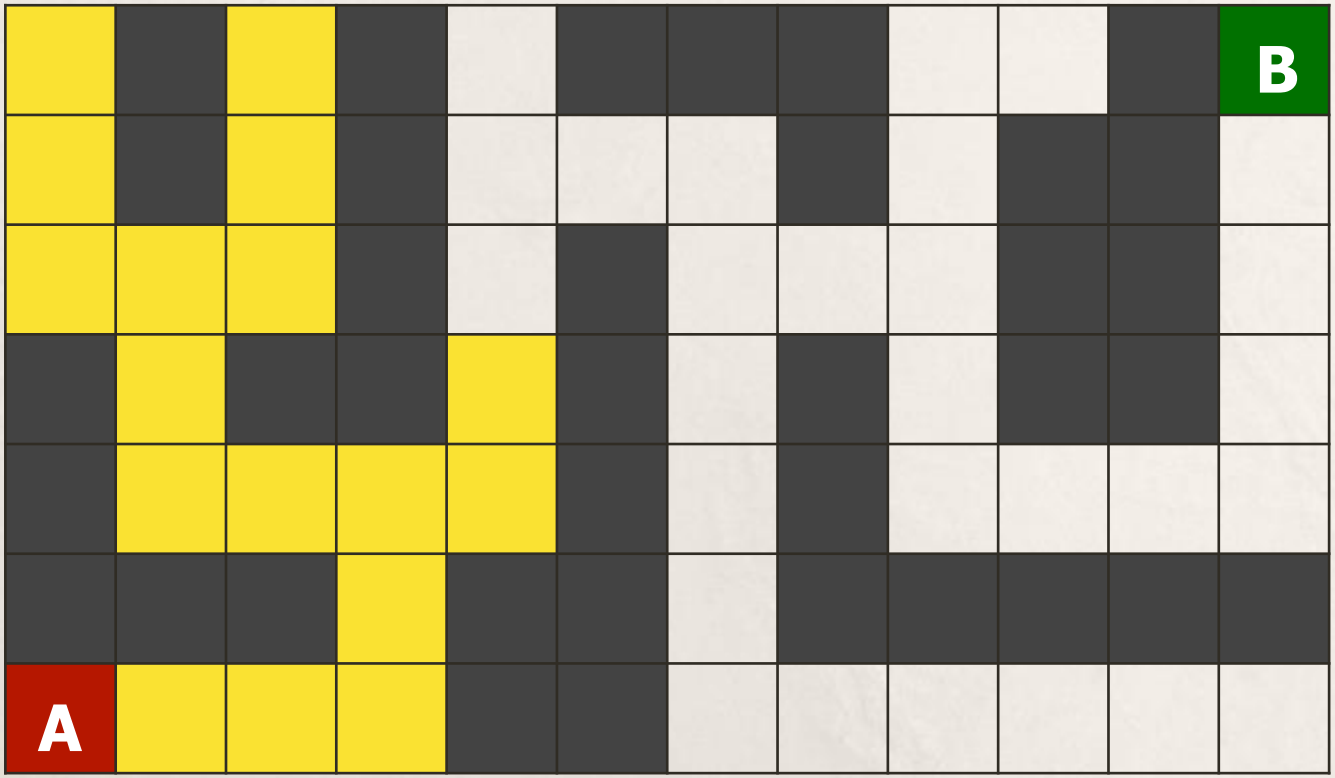
Depth-First Search



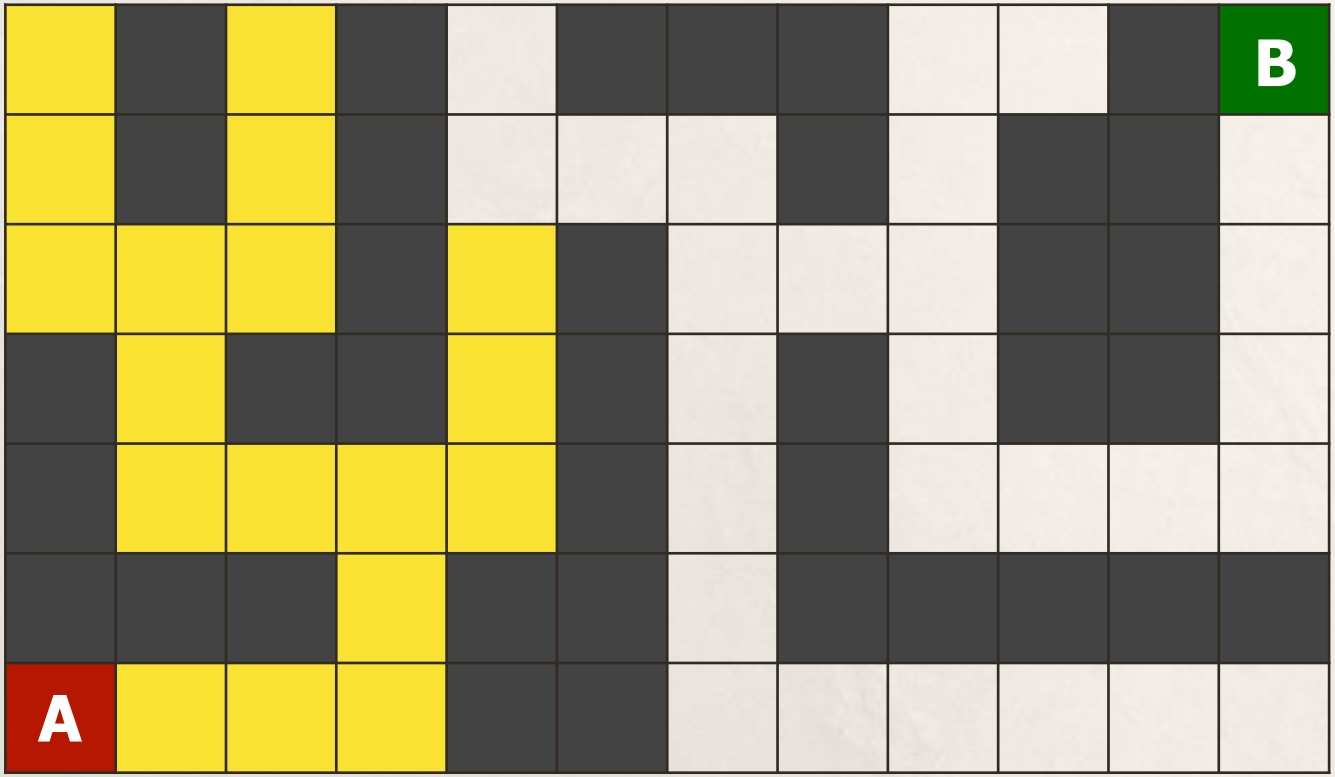
Depth-First Search



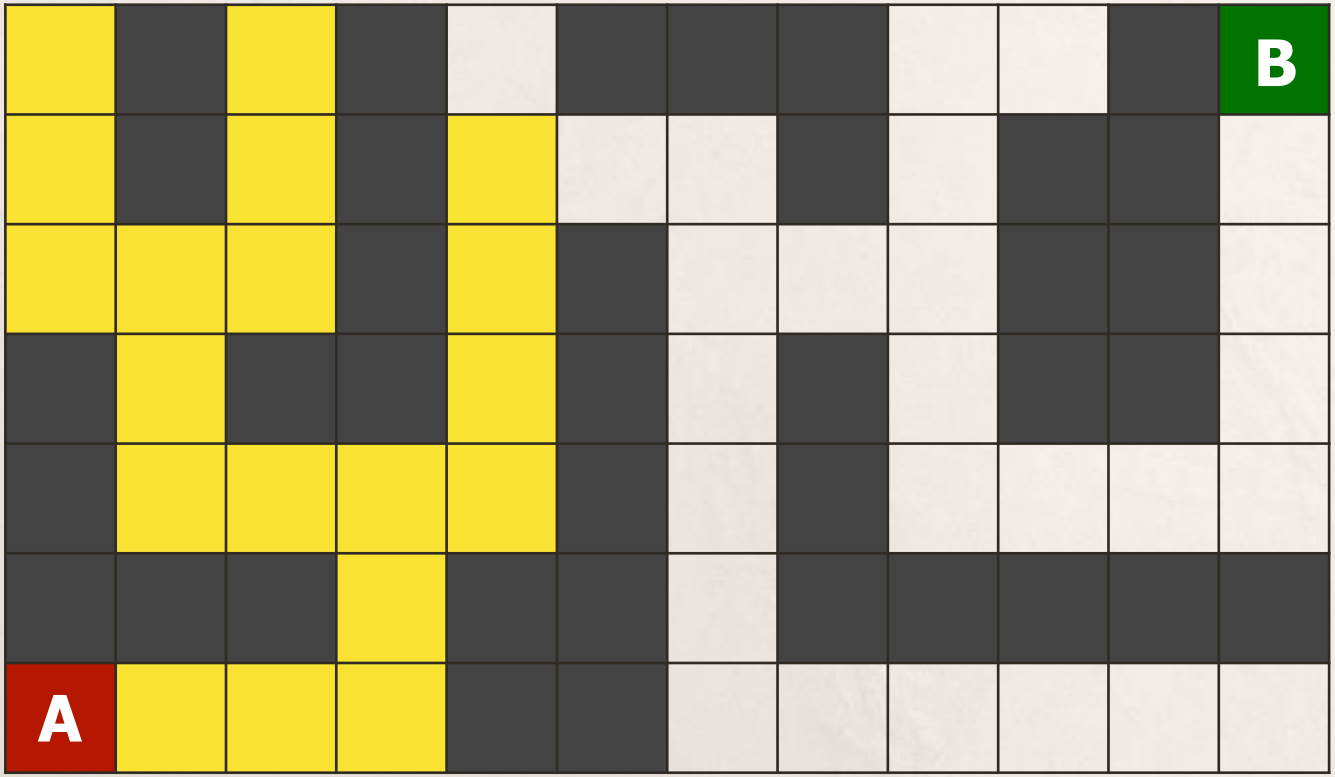
Depth-First Search



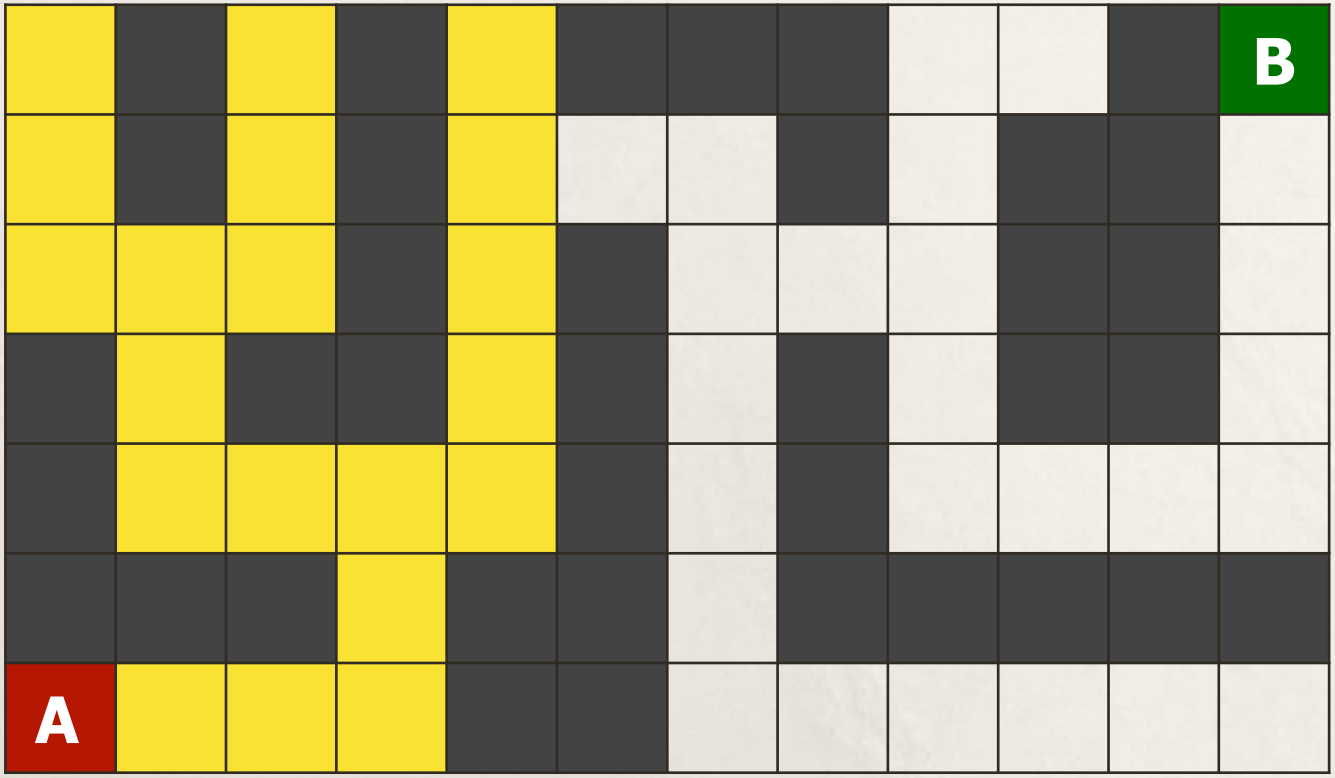
Depth-First Search



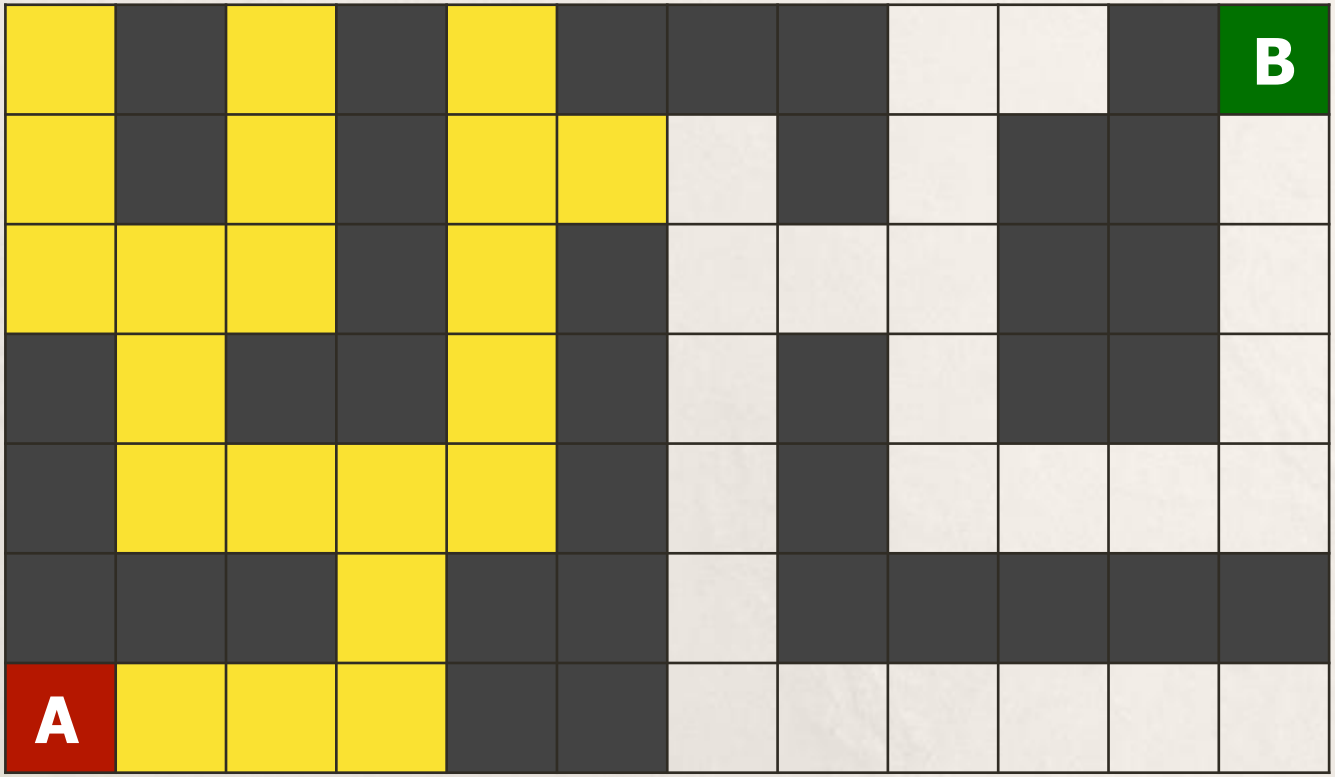
Depth-First Search



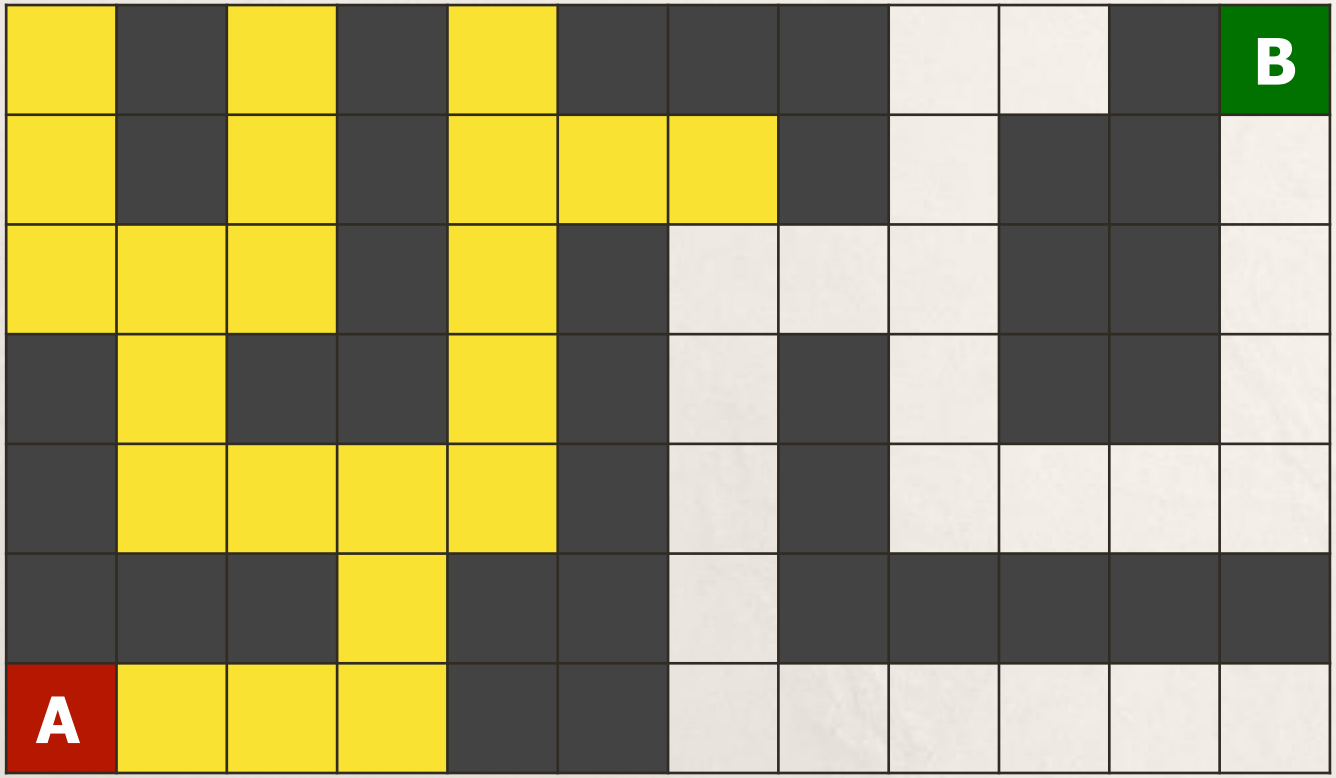
Depth-First Search



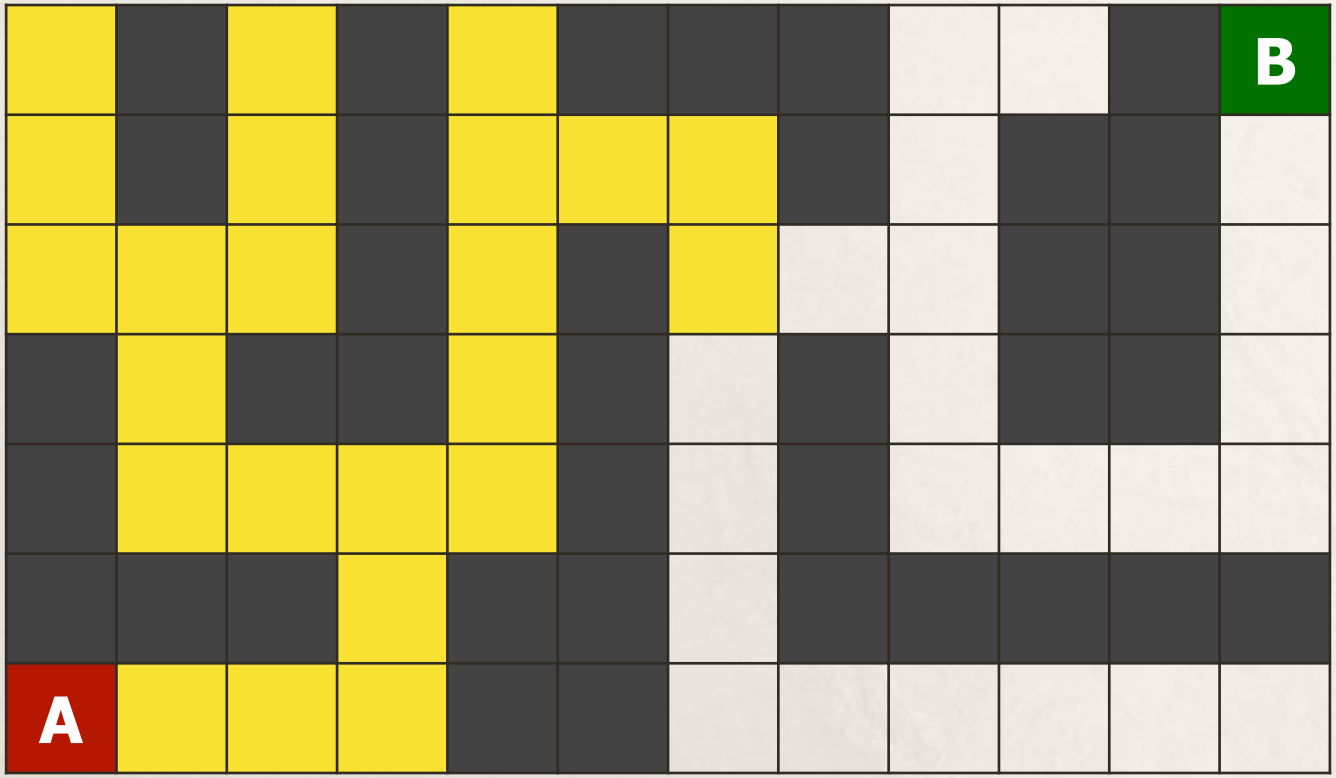
Depth-First Search



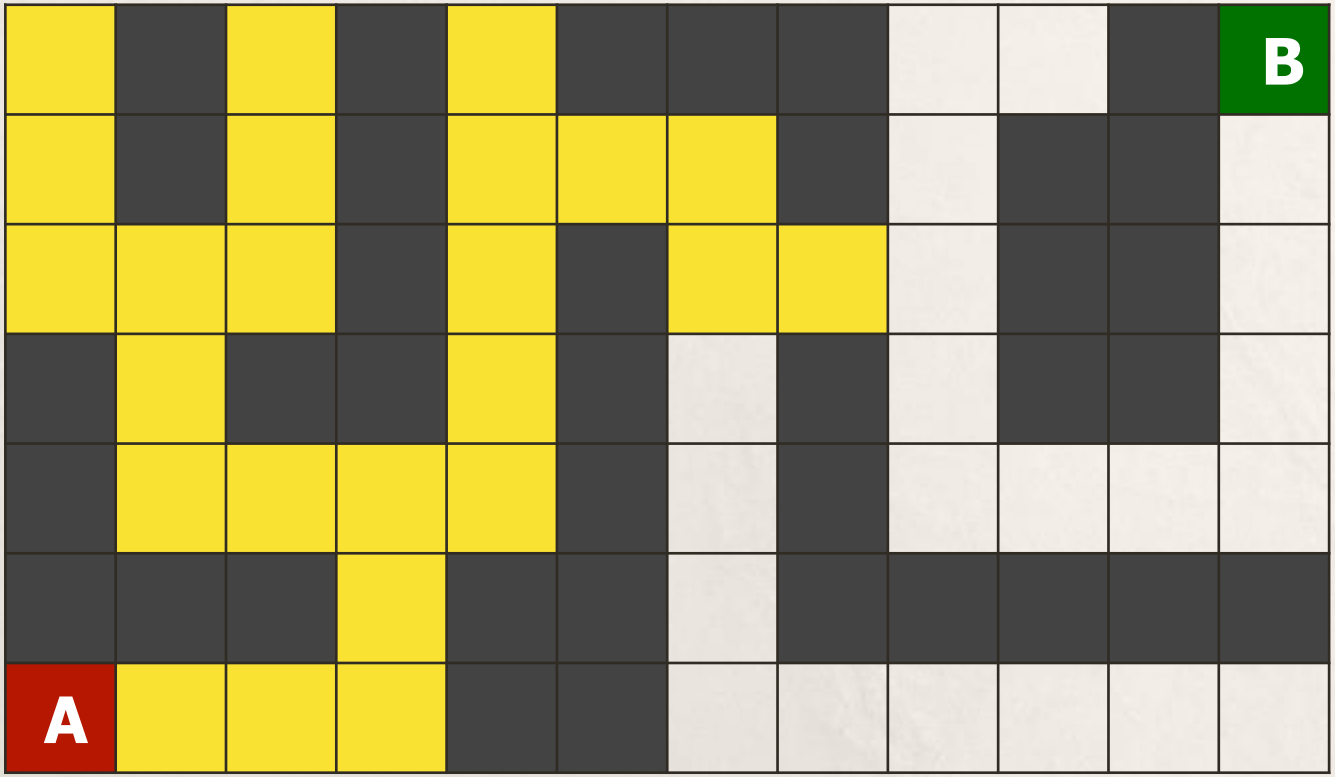
Depth-First Search



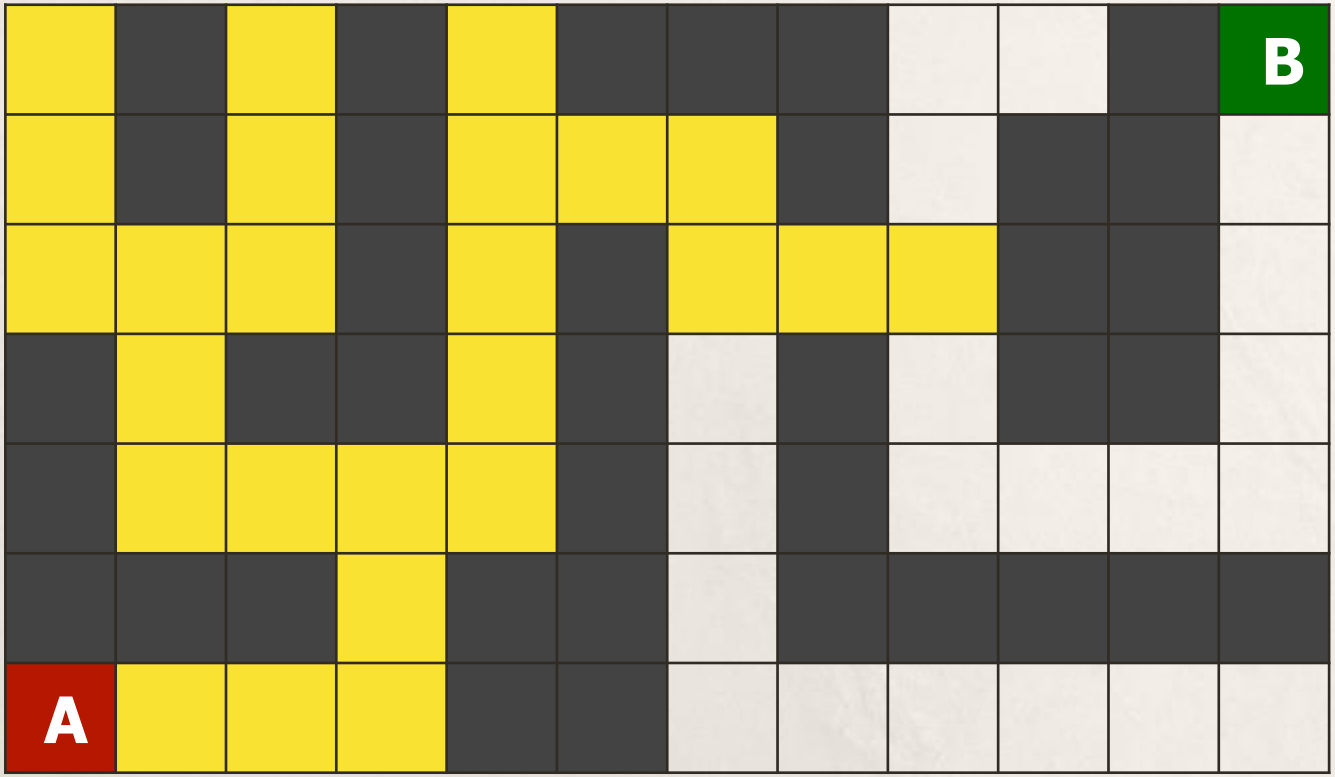
Depth-First Search



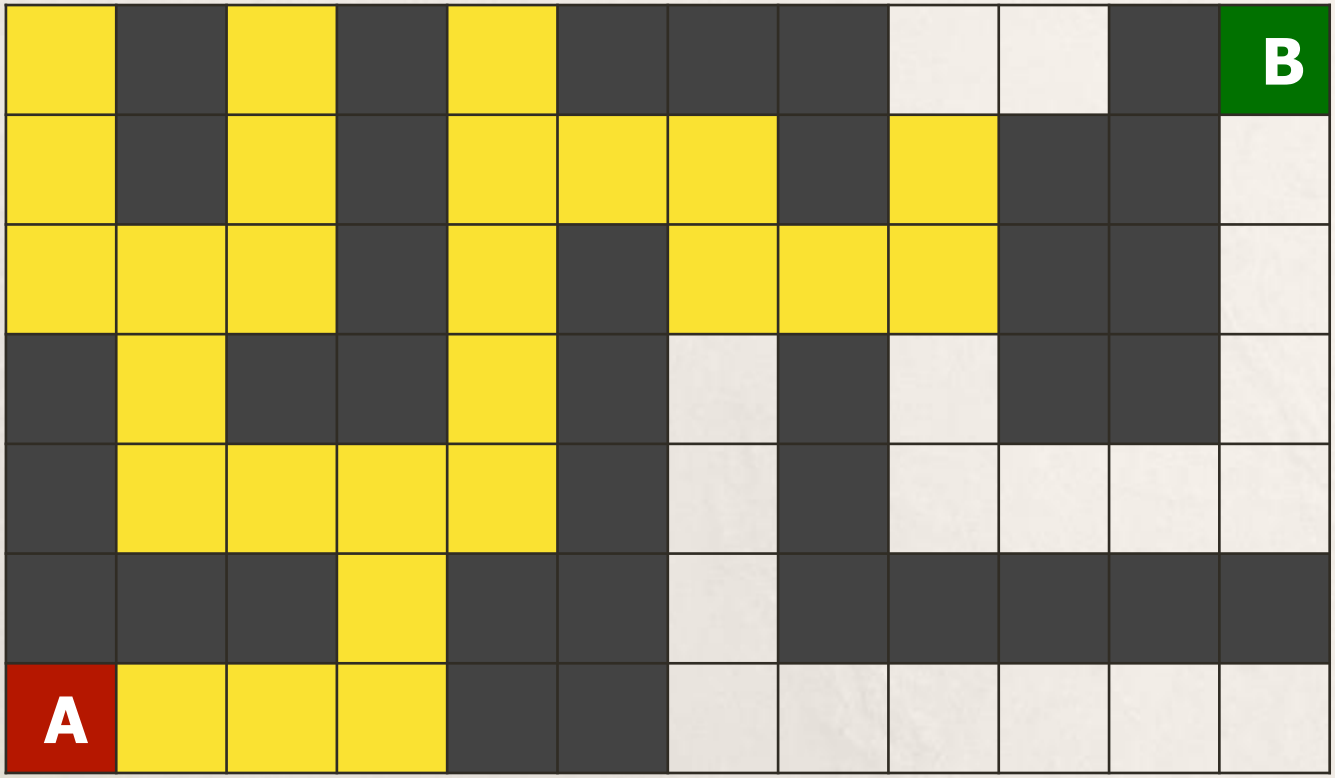
Depth-First Search



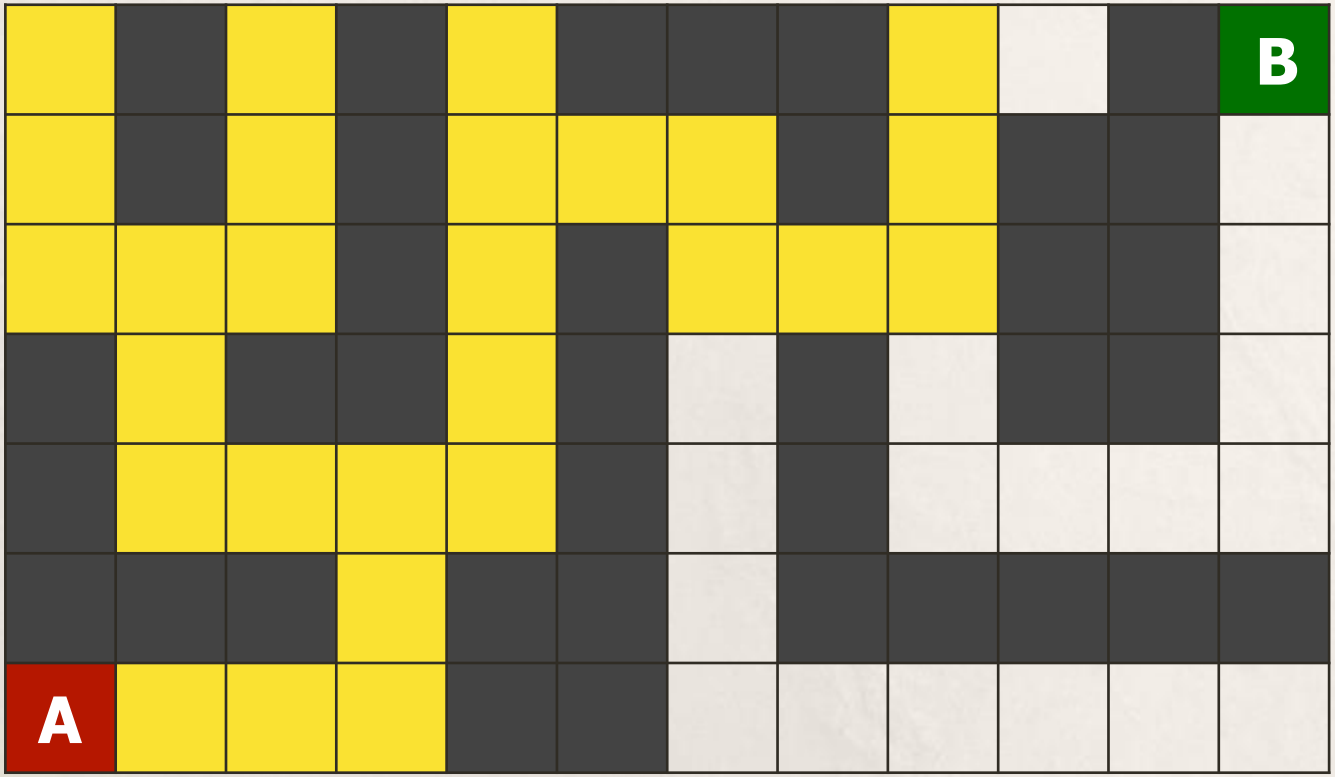
Depth-First Search



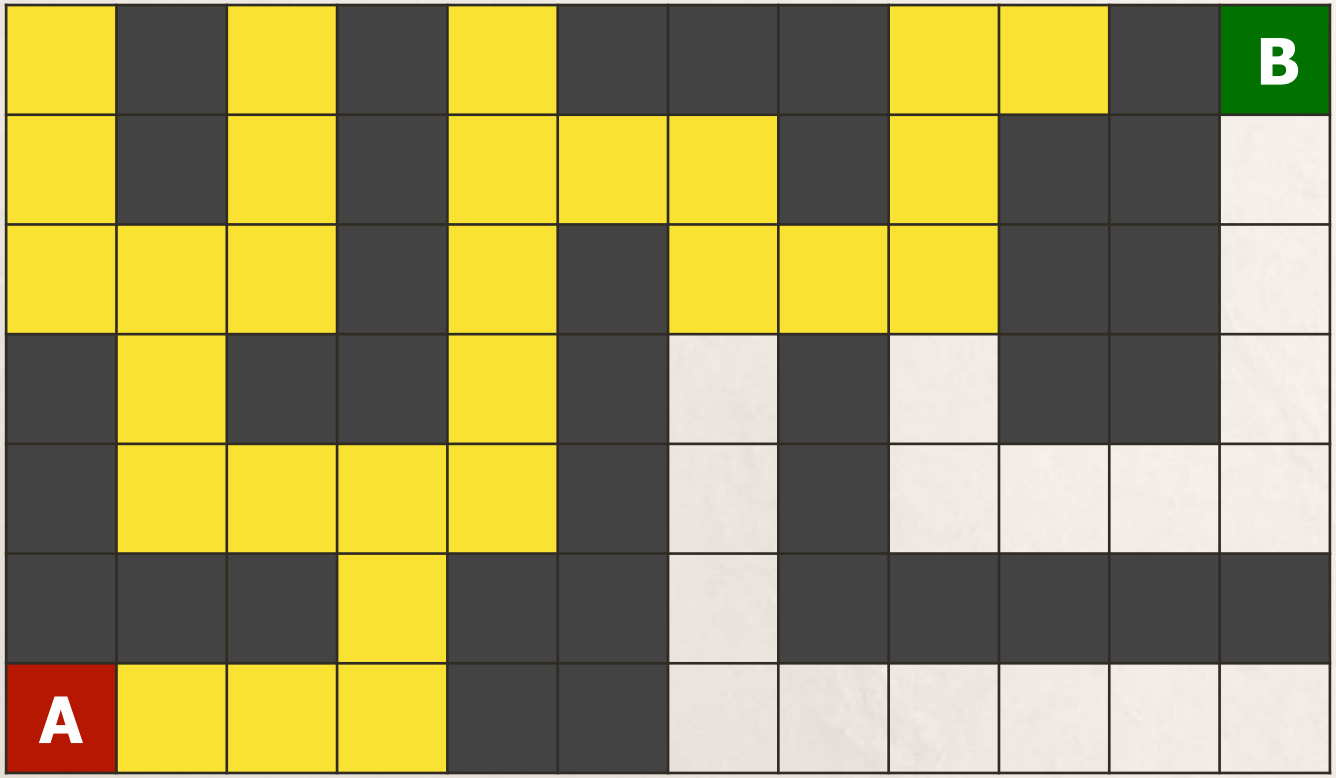
Depth-First Search



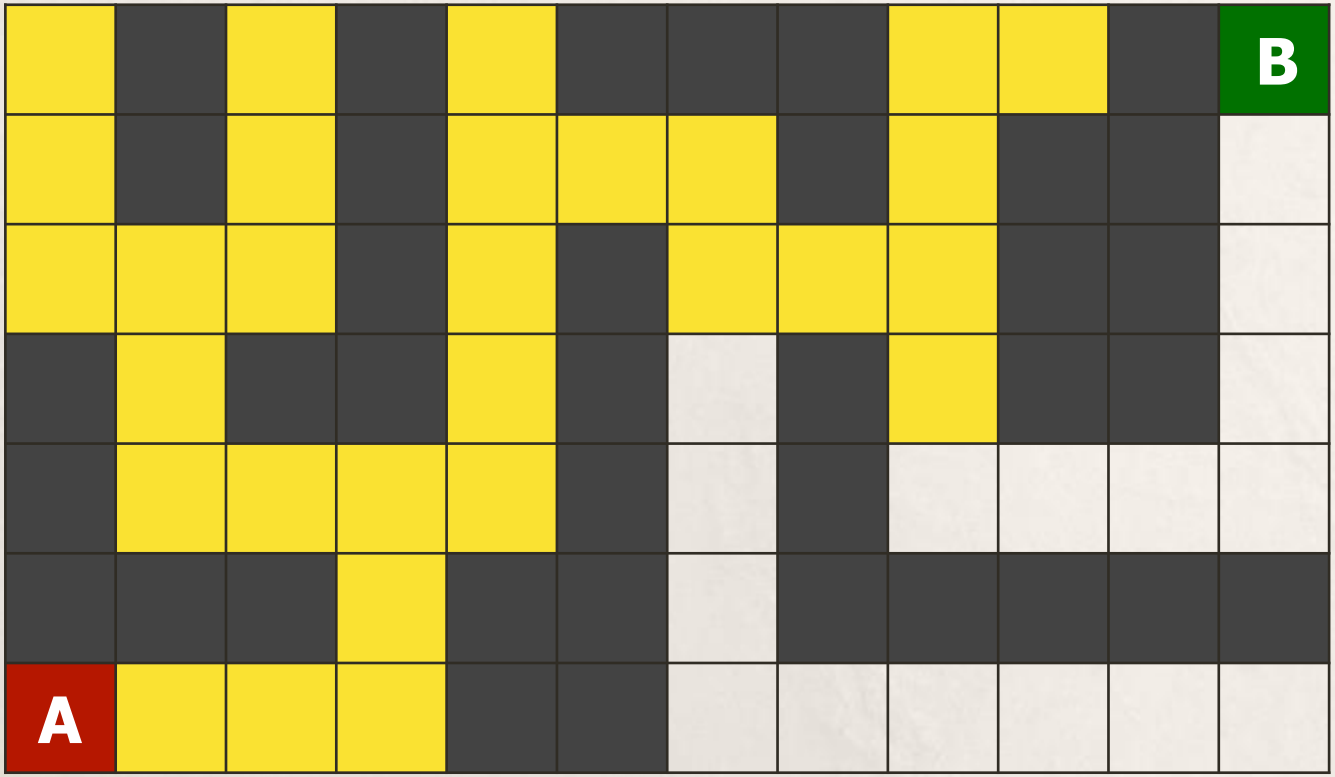
Depth-First Search



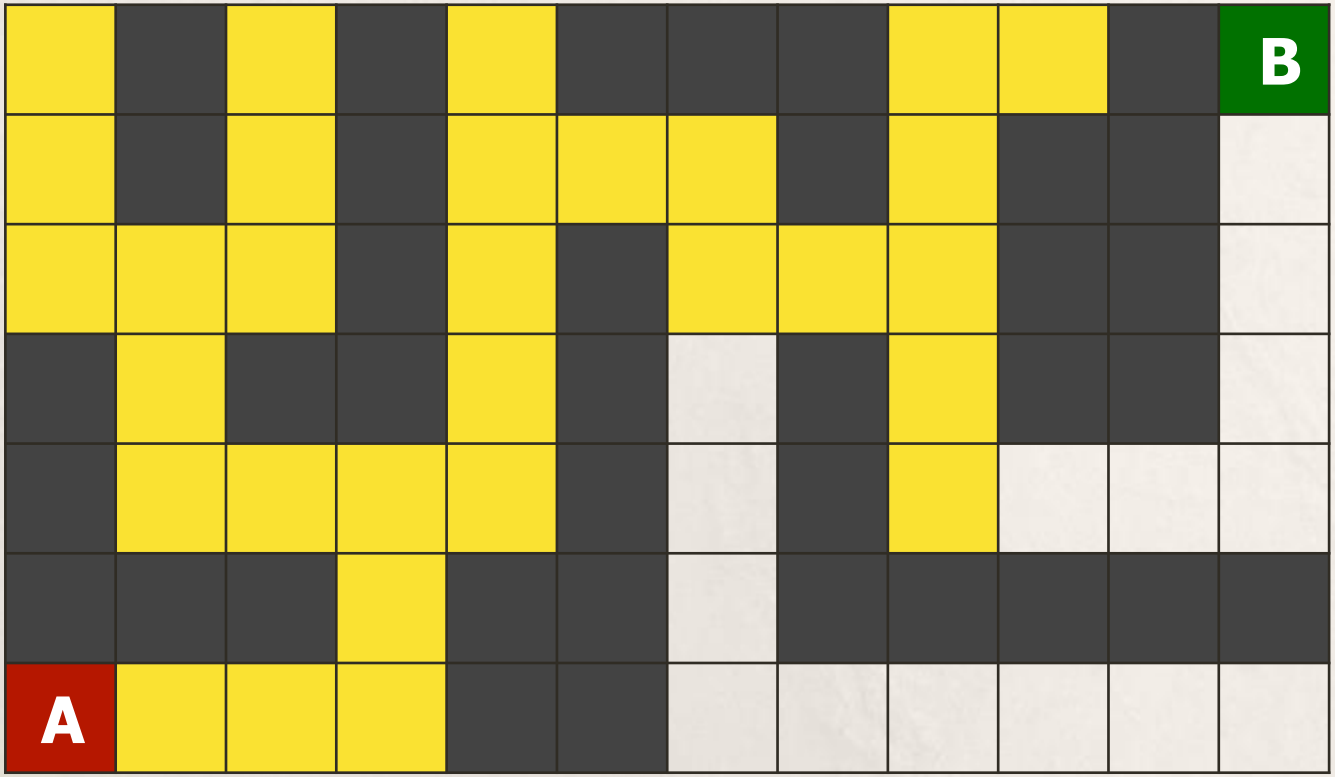
Depth-First Search



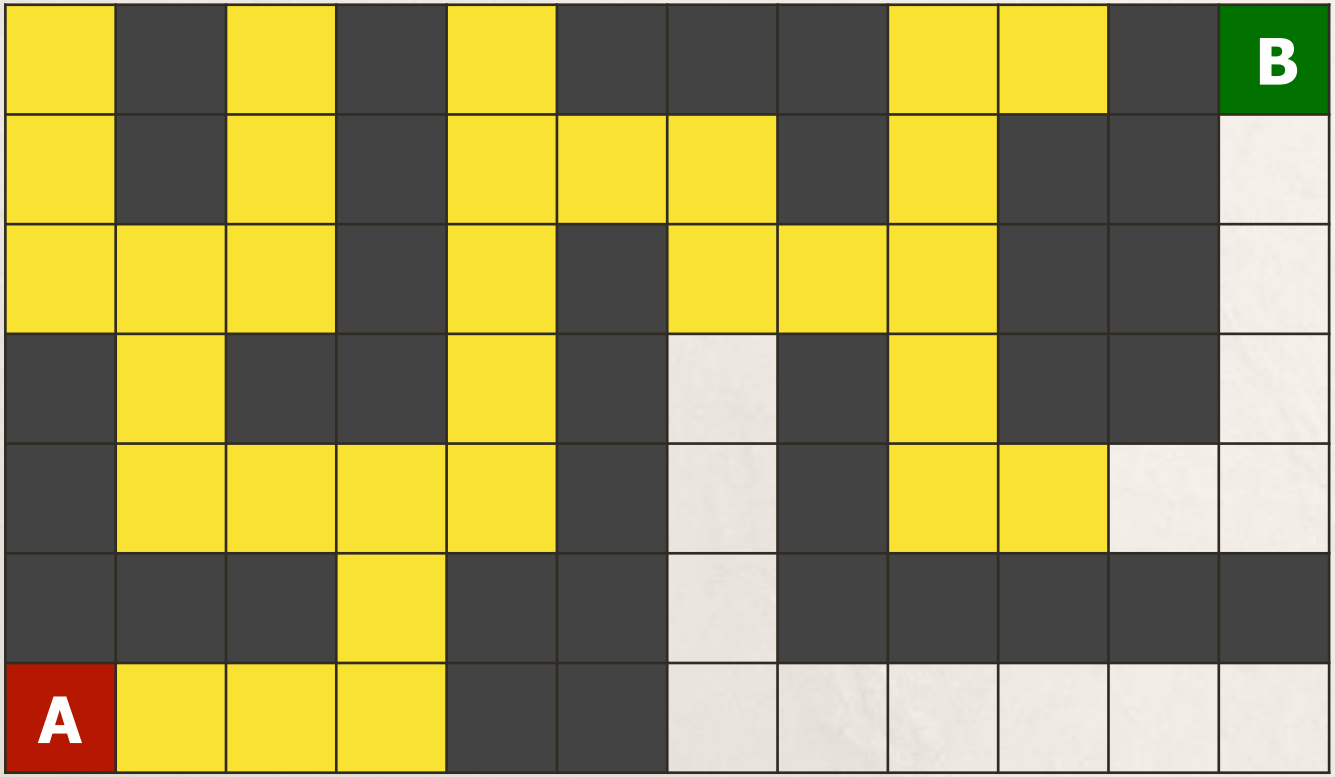
Depth-First Search



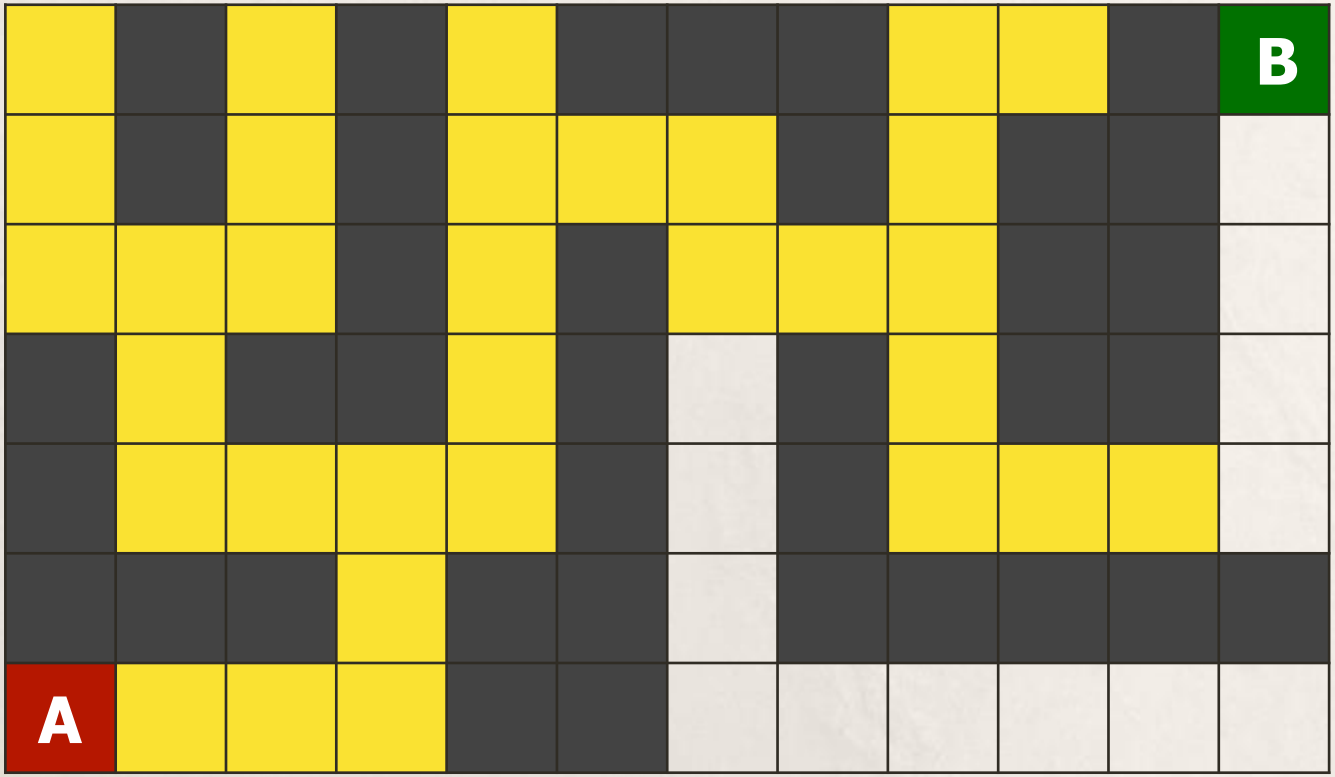
Depth-First Search



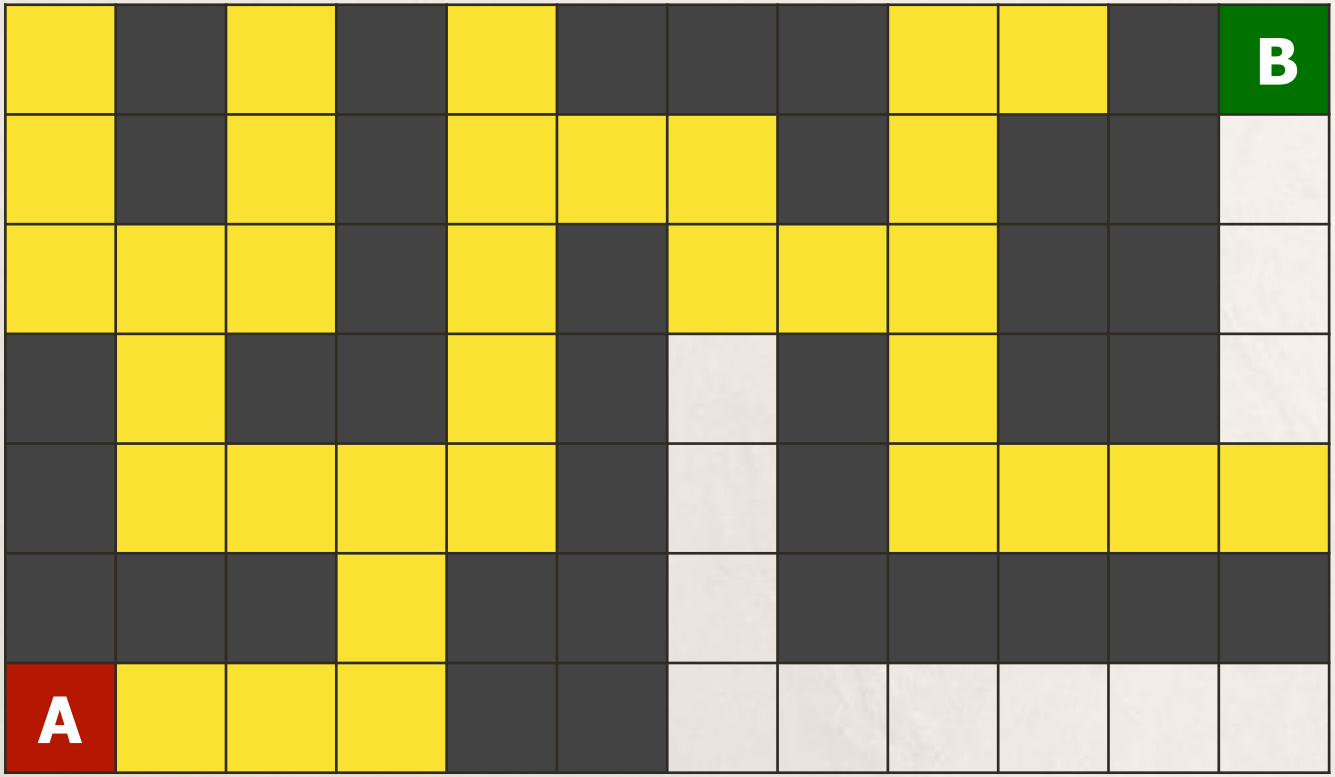
Depth-First Search



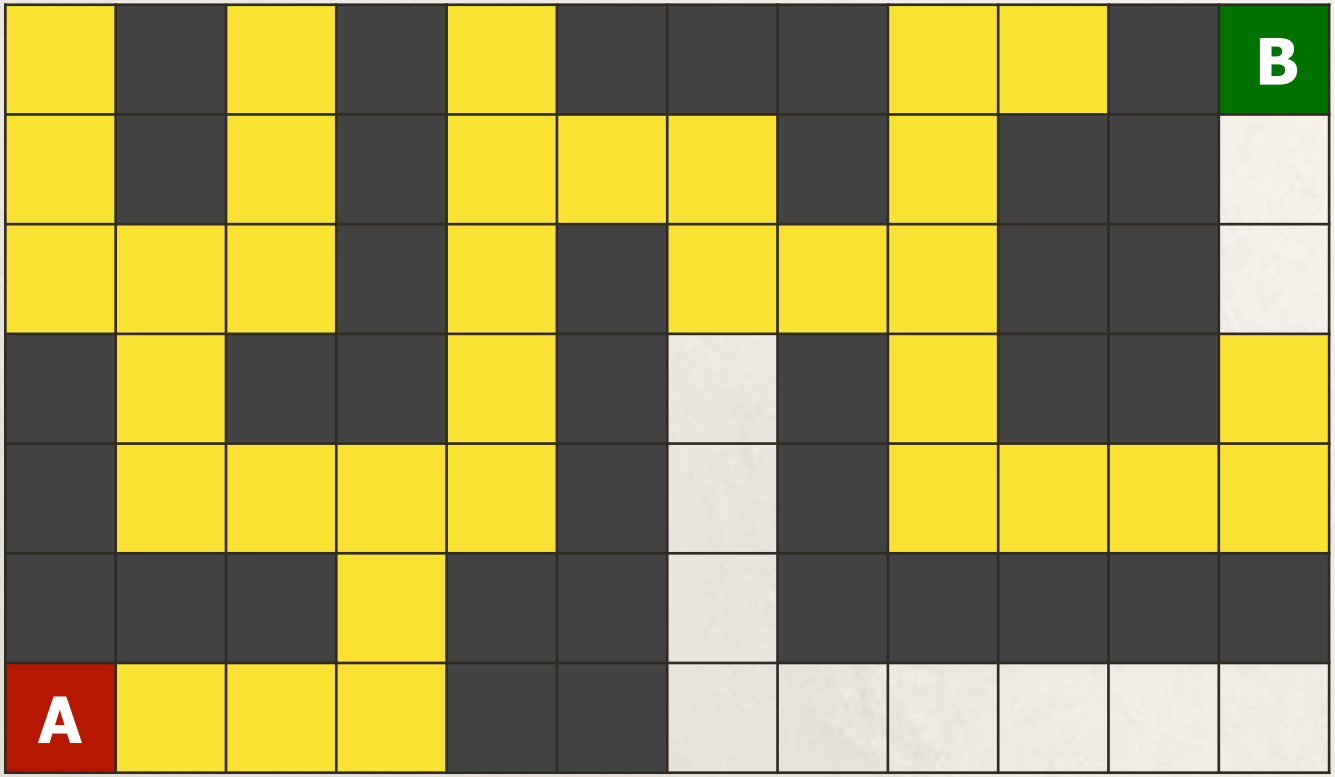
Depth-First Search



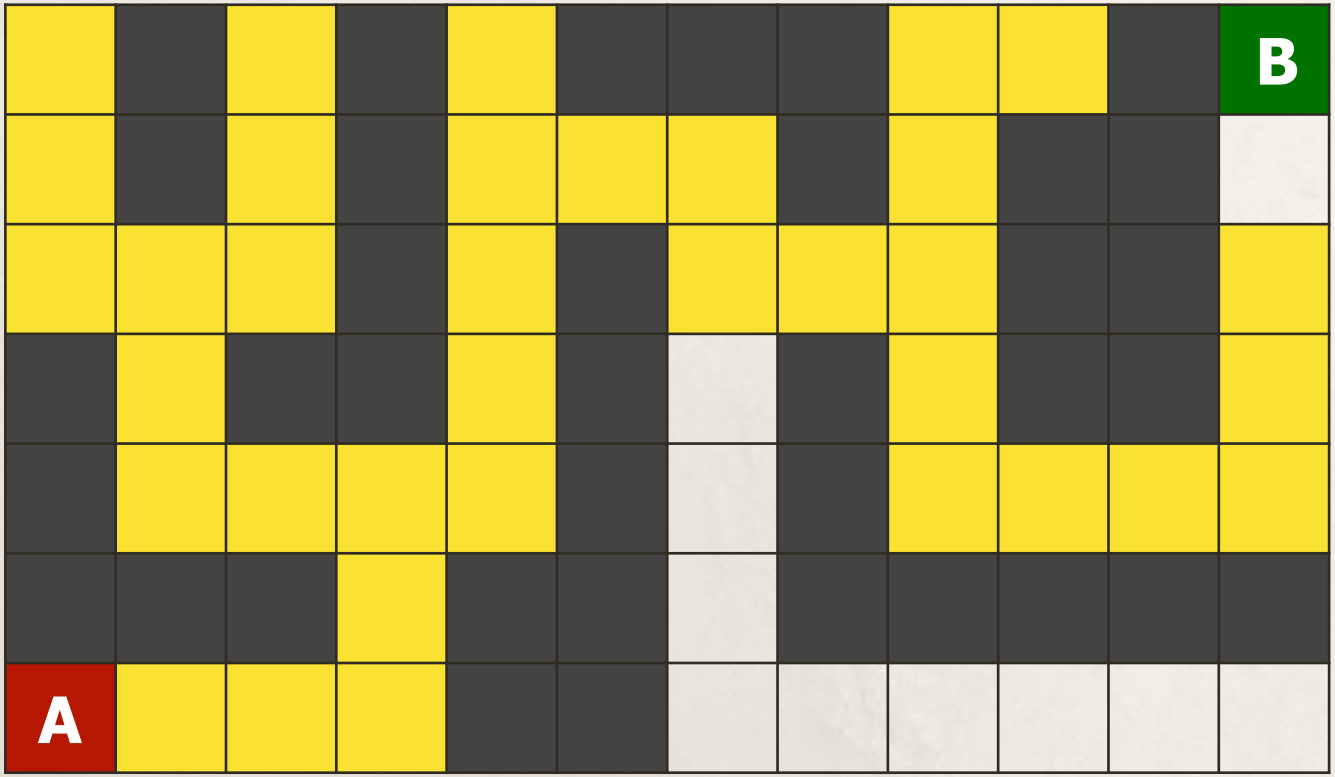
Depth-First Search



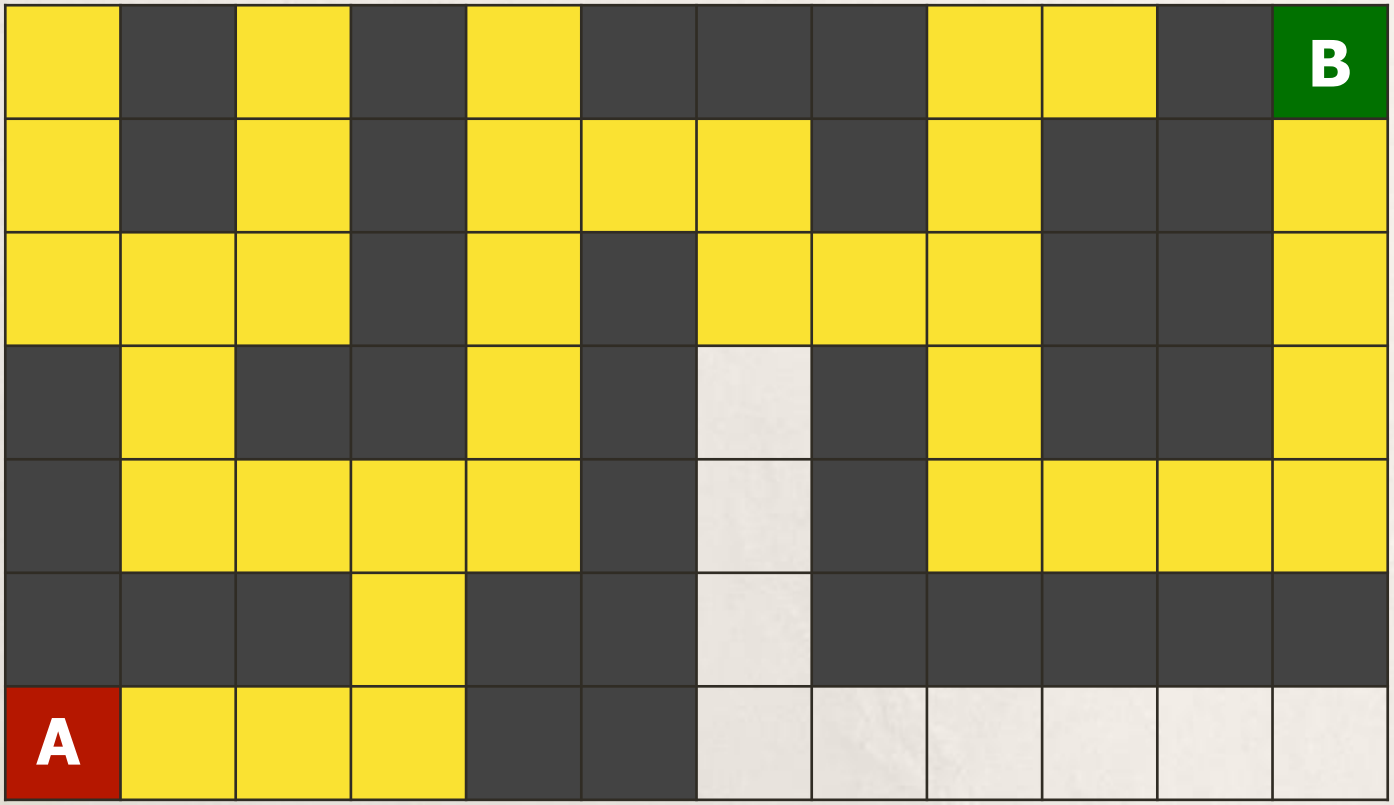
Depth-First Search



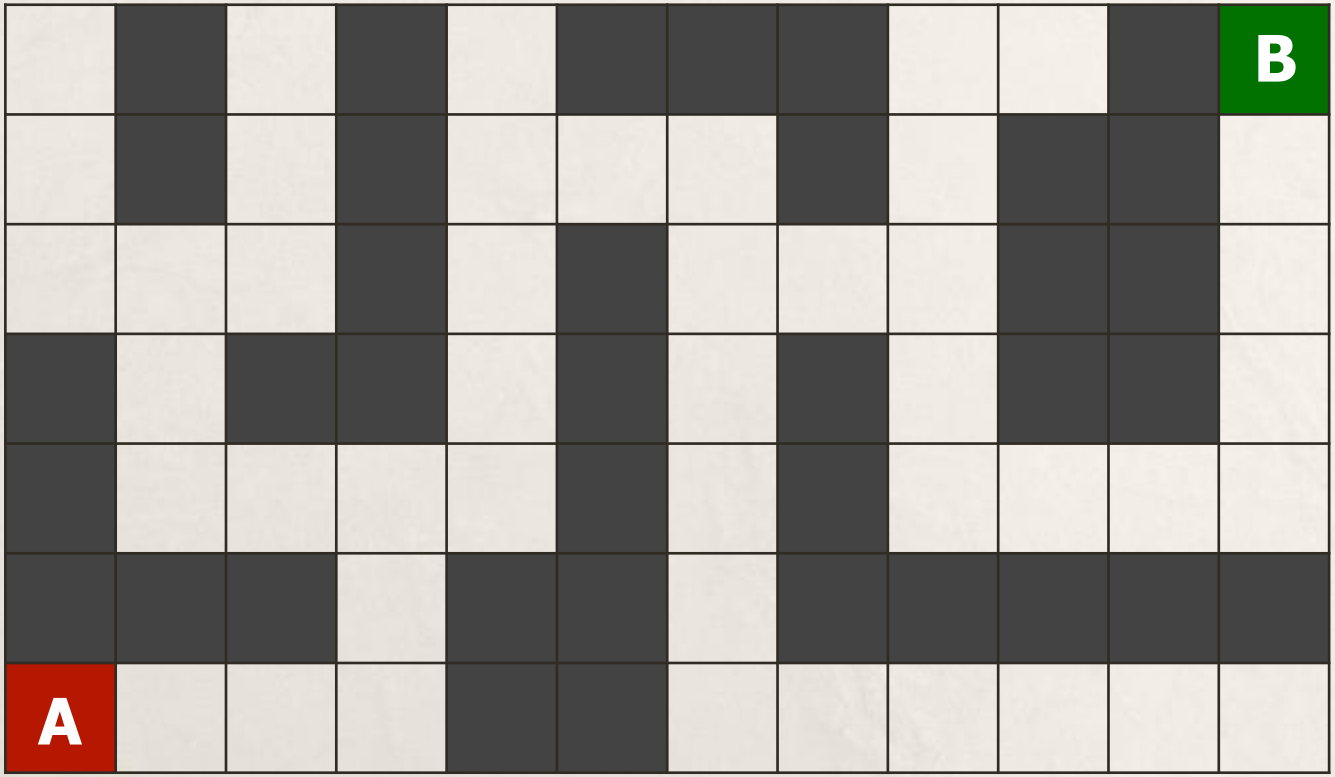
Depth-First Search



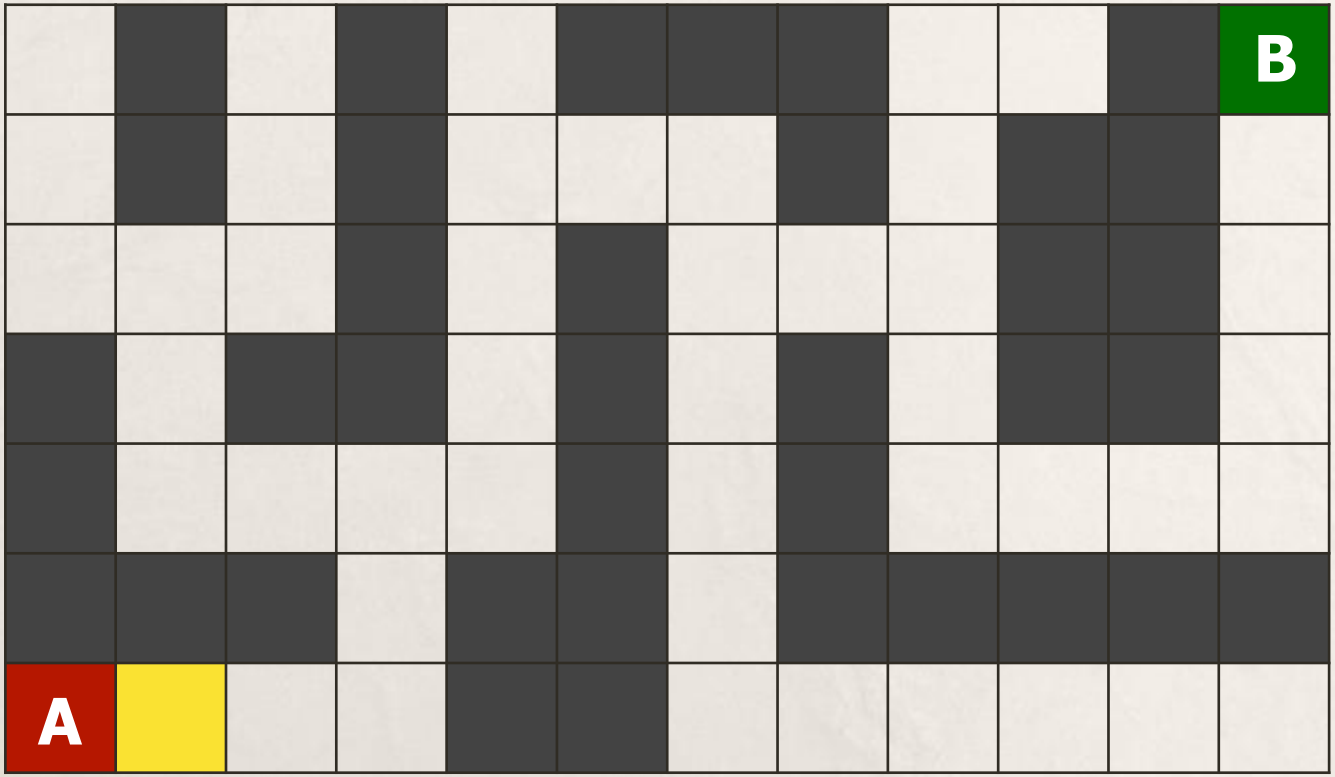
Depth-First Search



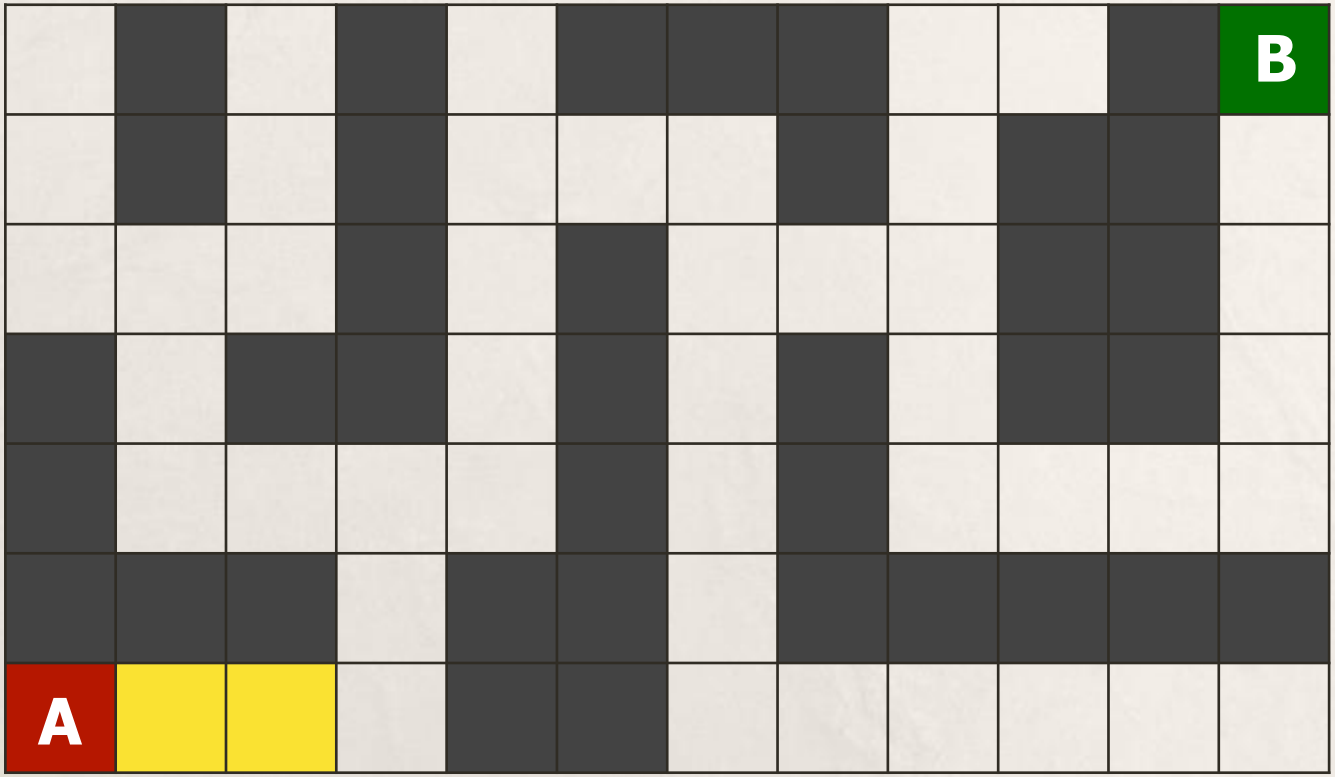
Breadth-First Search



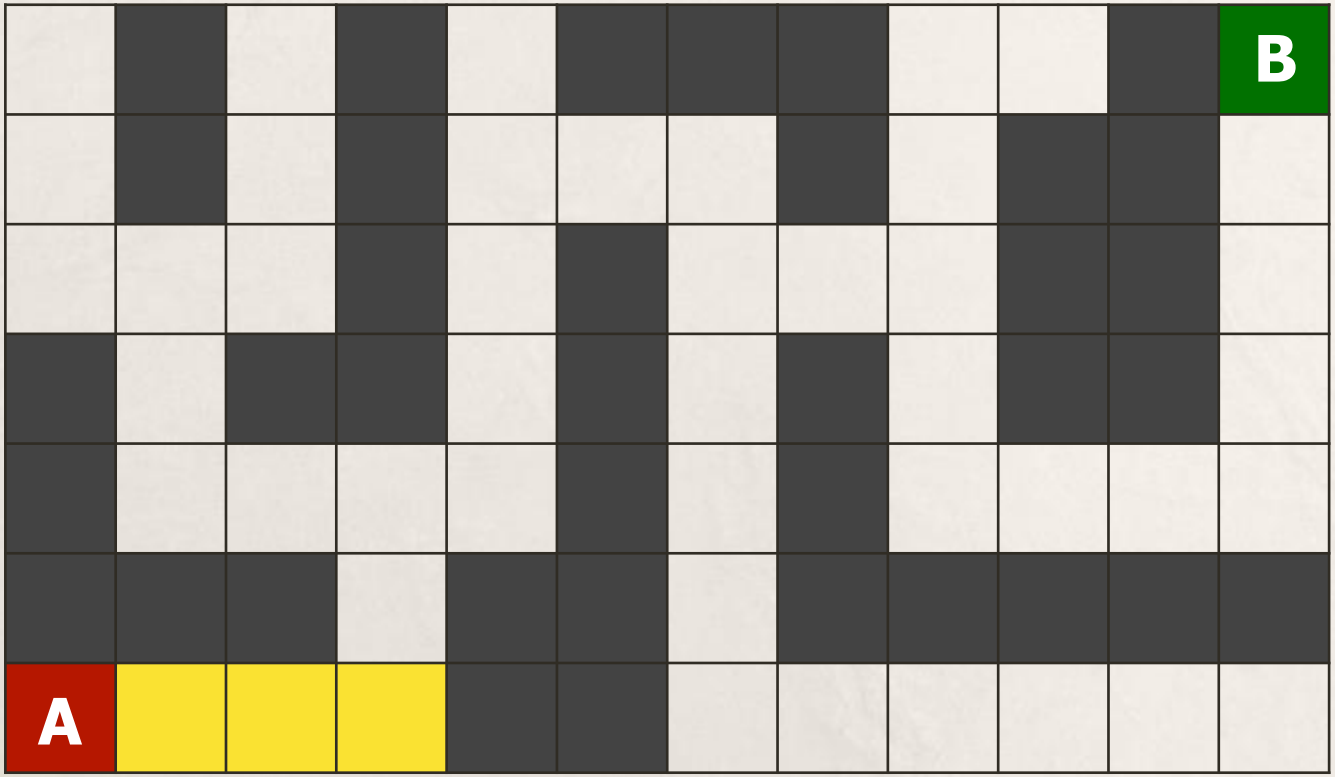
Breadth-First Search



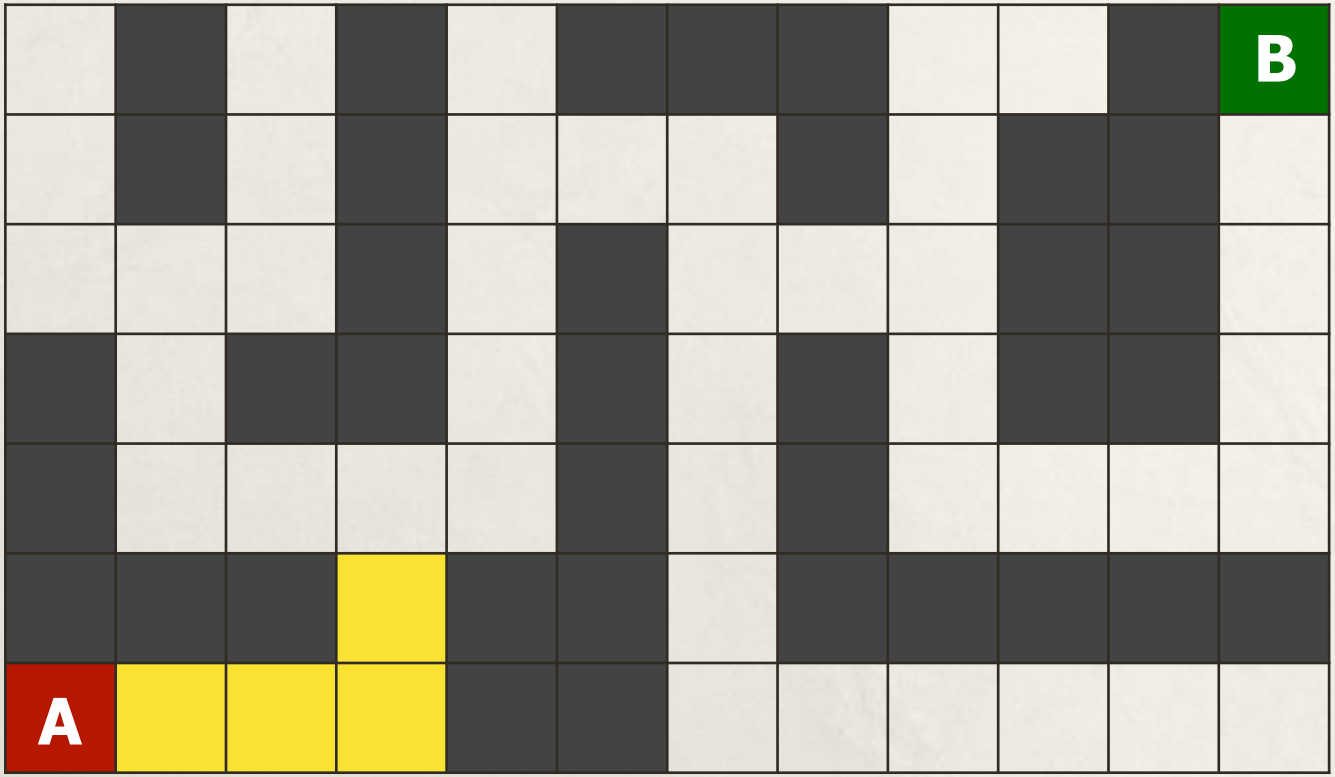
Breadth-First Search



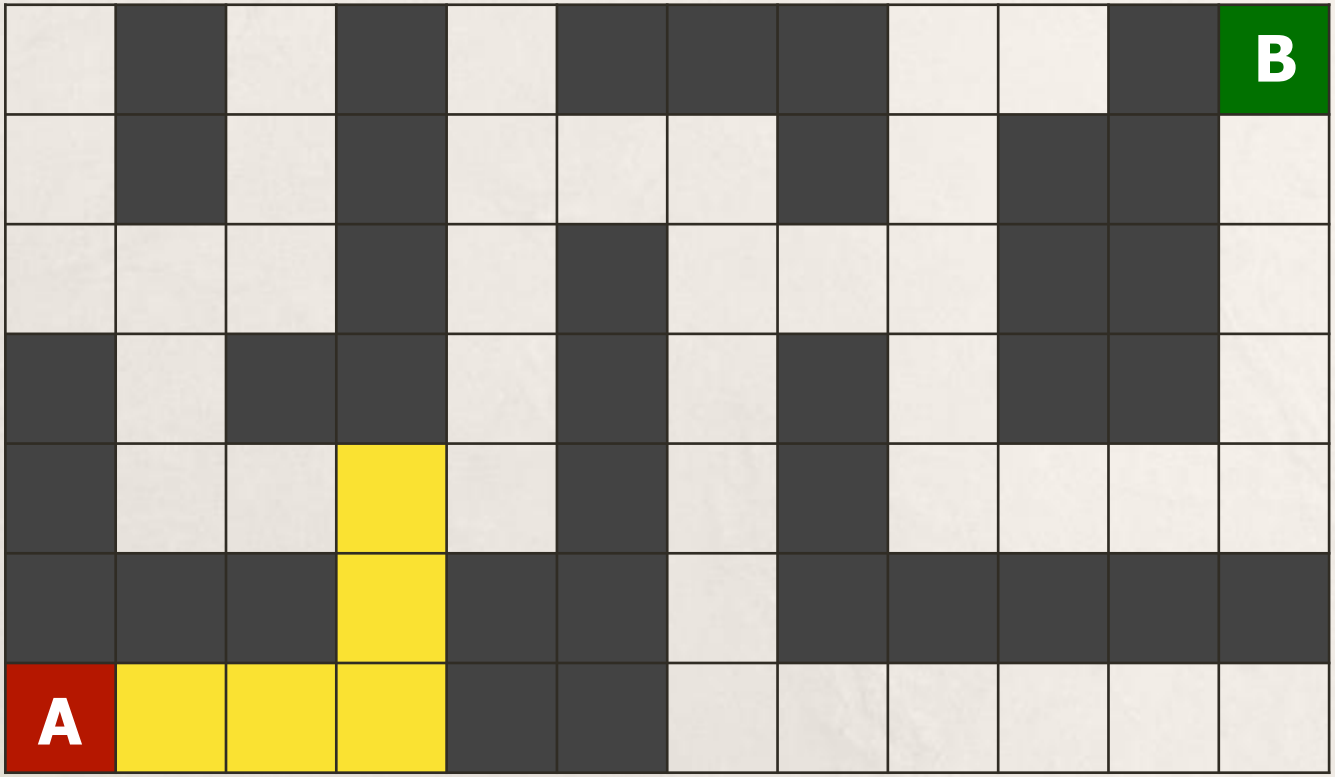
Breadth-First Search



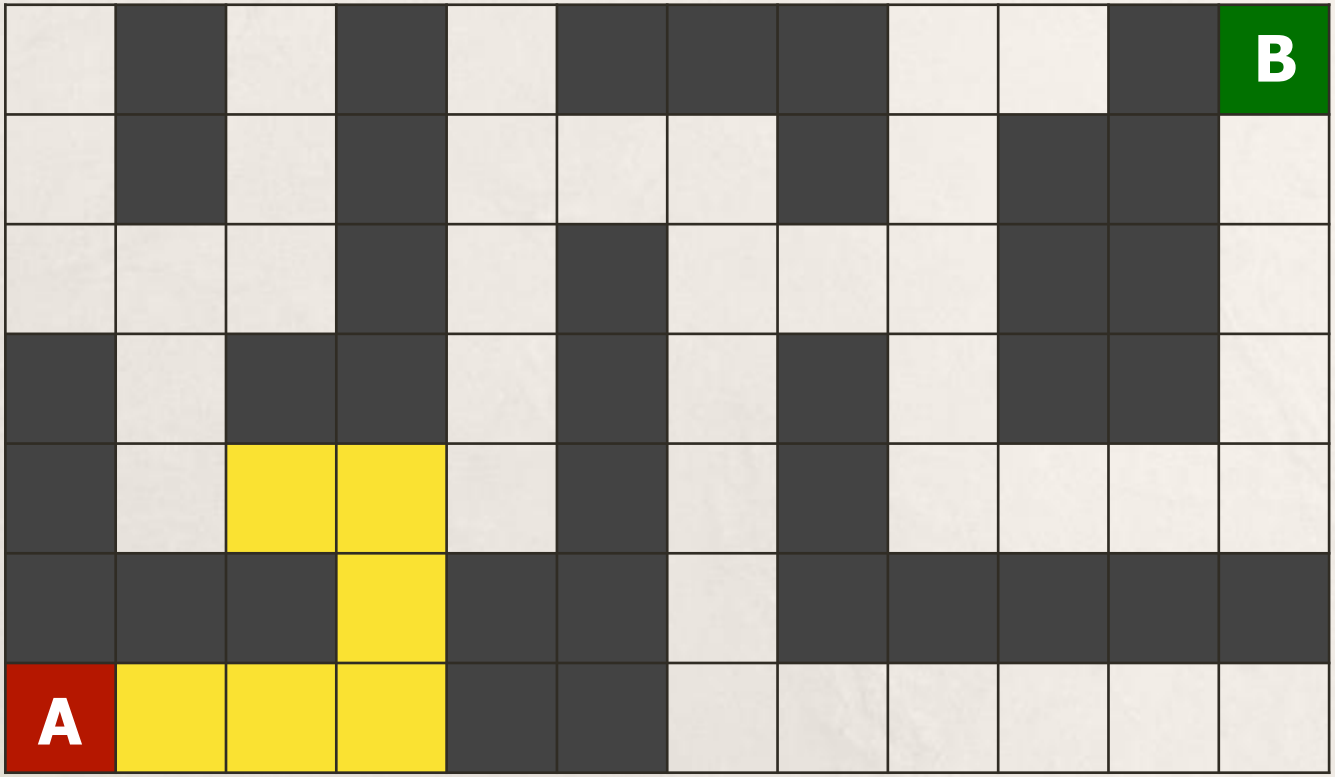
Breadth-First Search



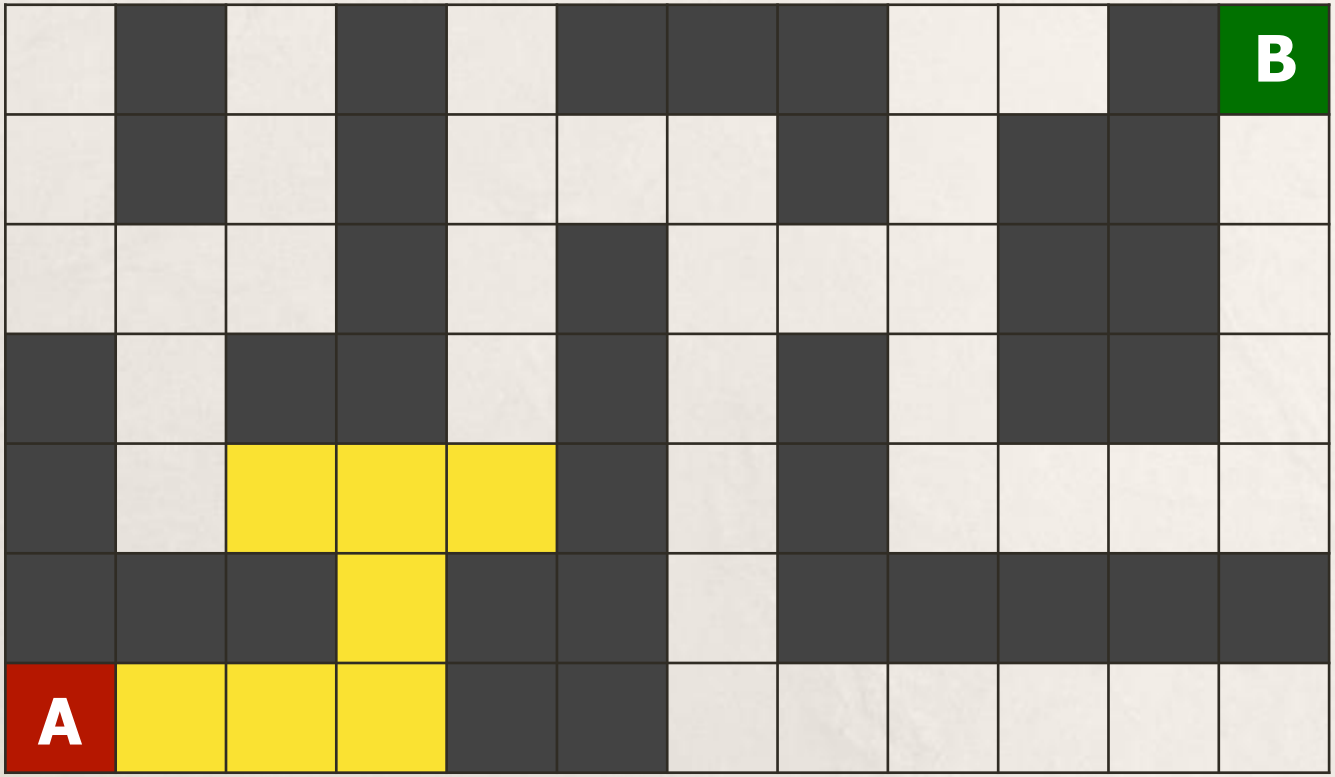
Breadth-First Search



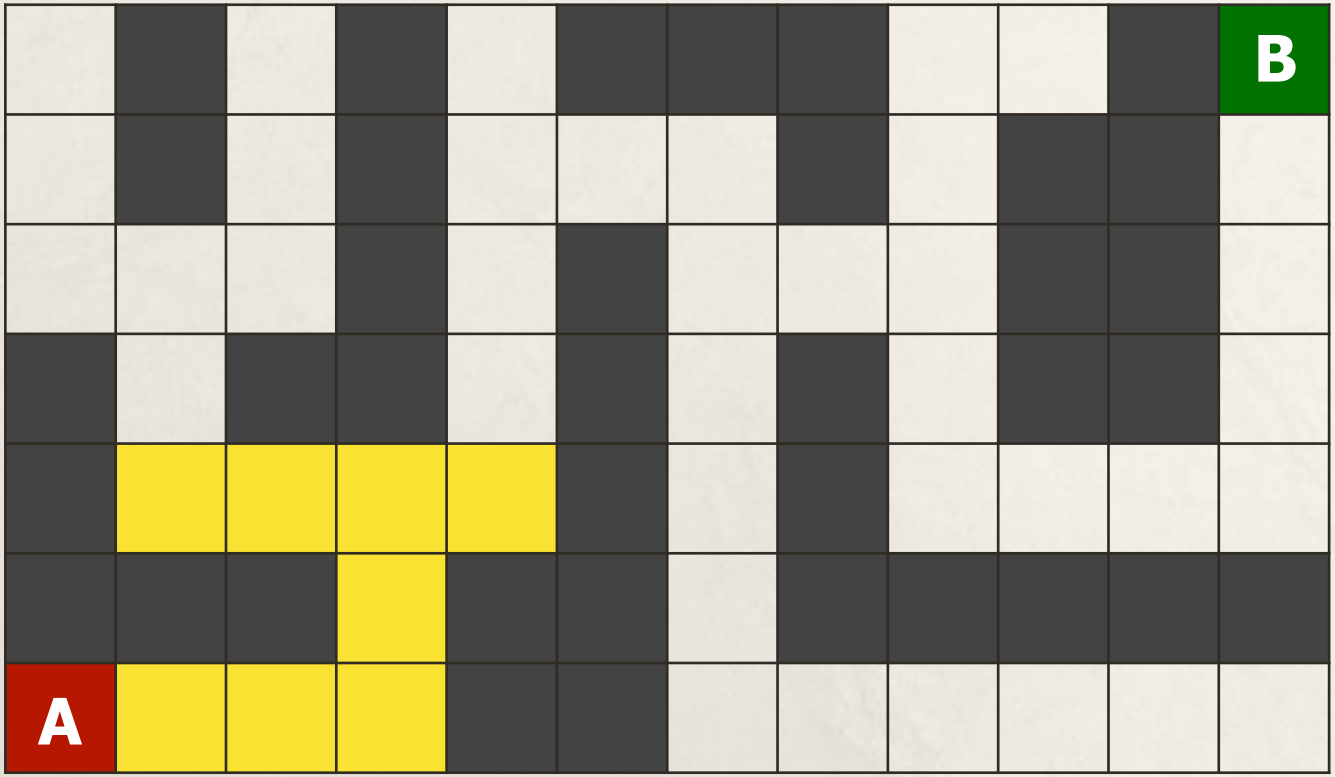
Breadth-First Search



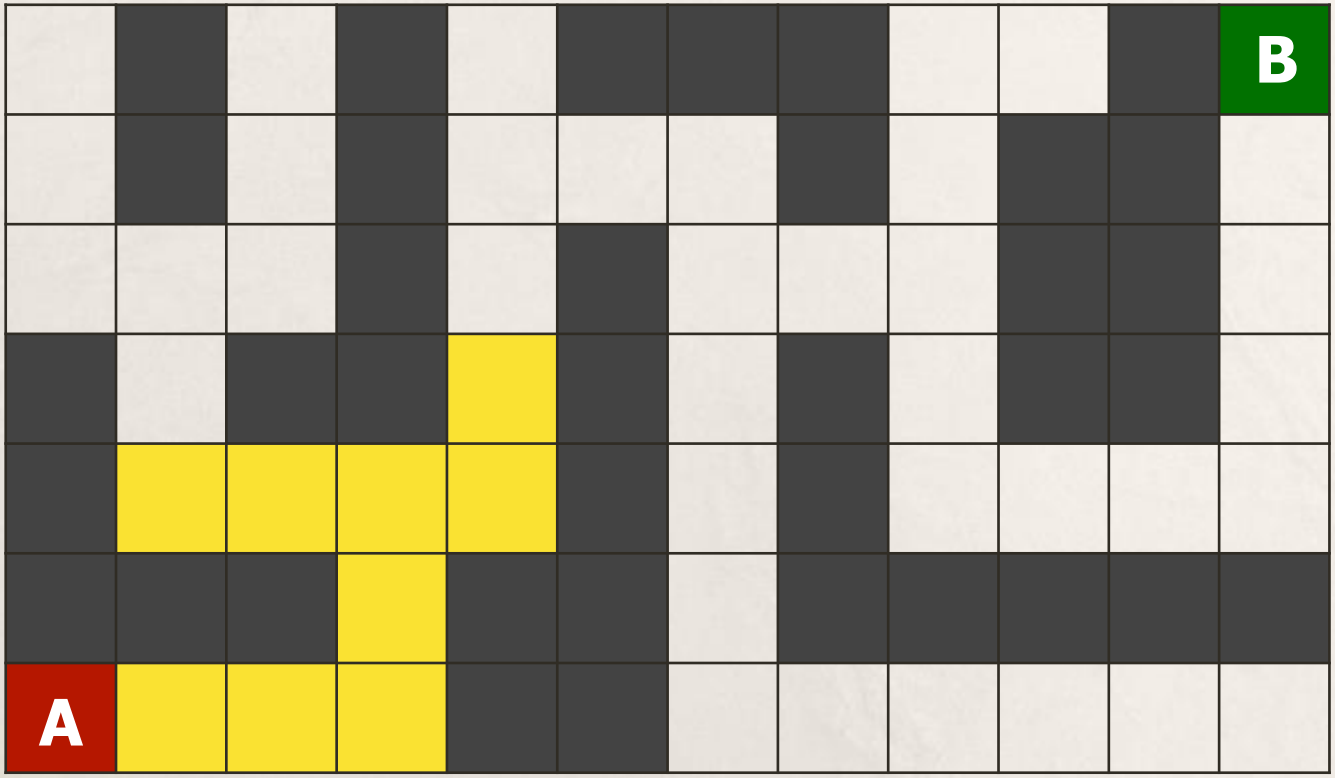
Breadth-First Search



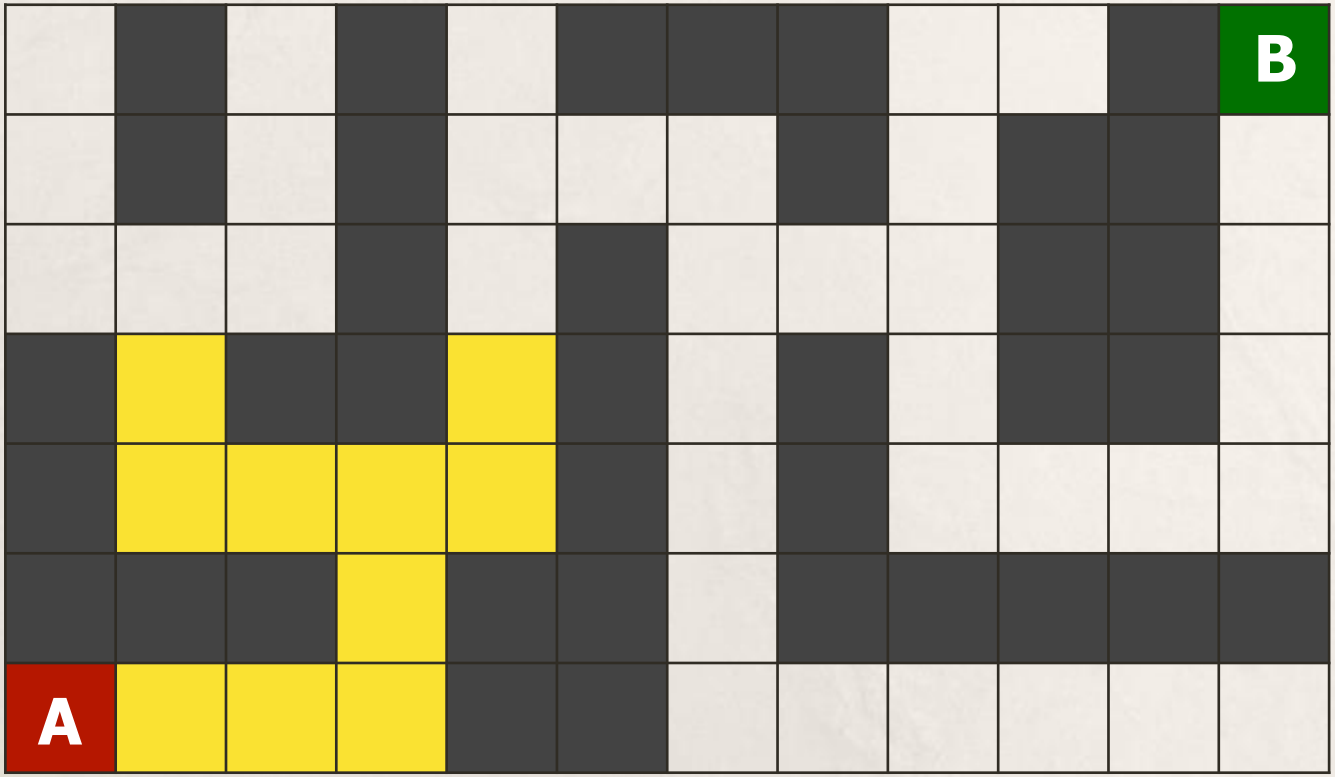
Breadth-First Search



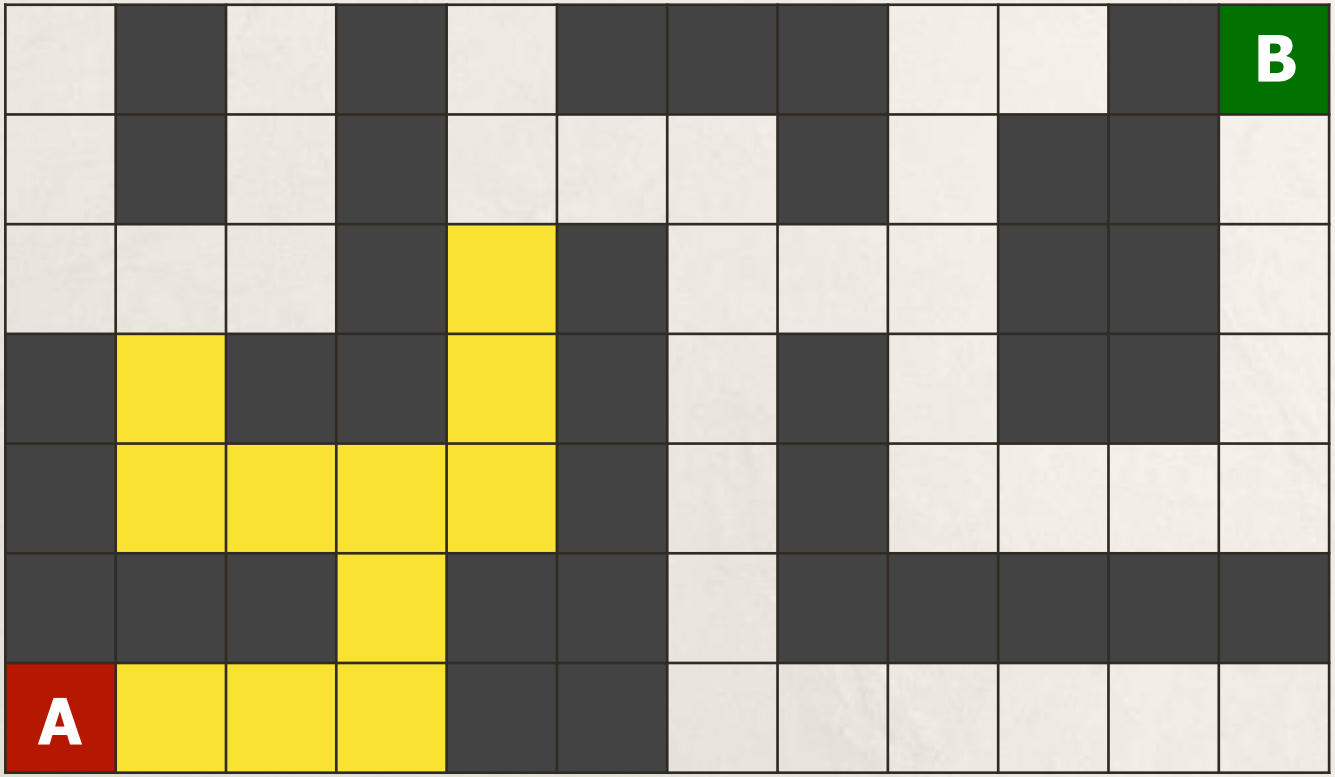
Breadth-First Search



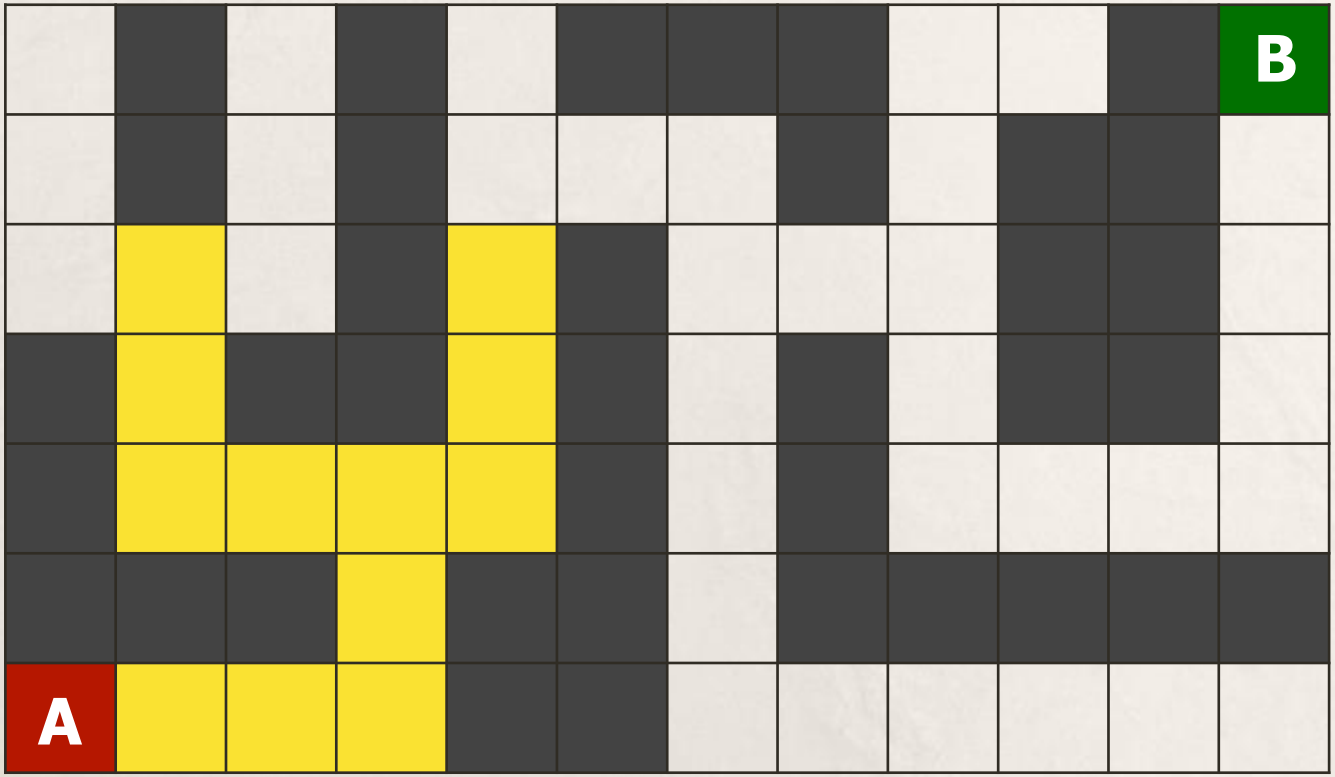
Breadth-First Search



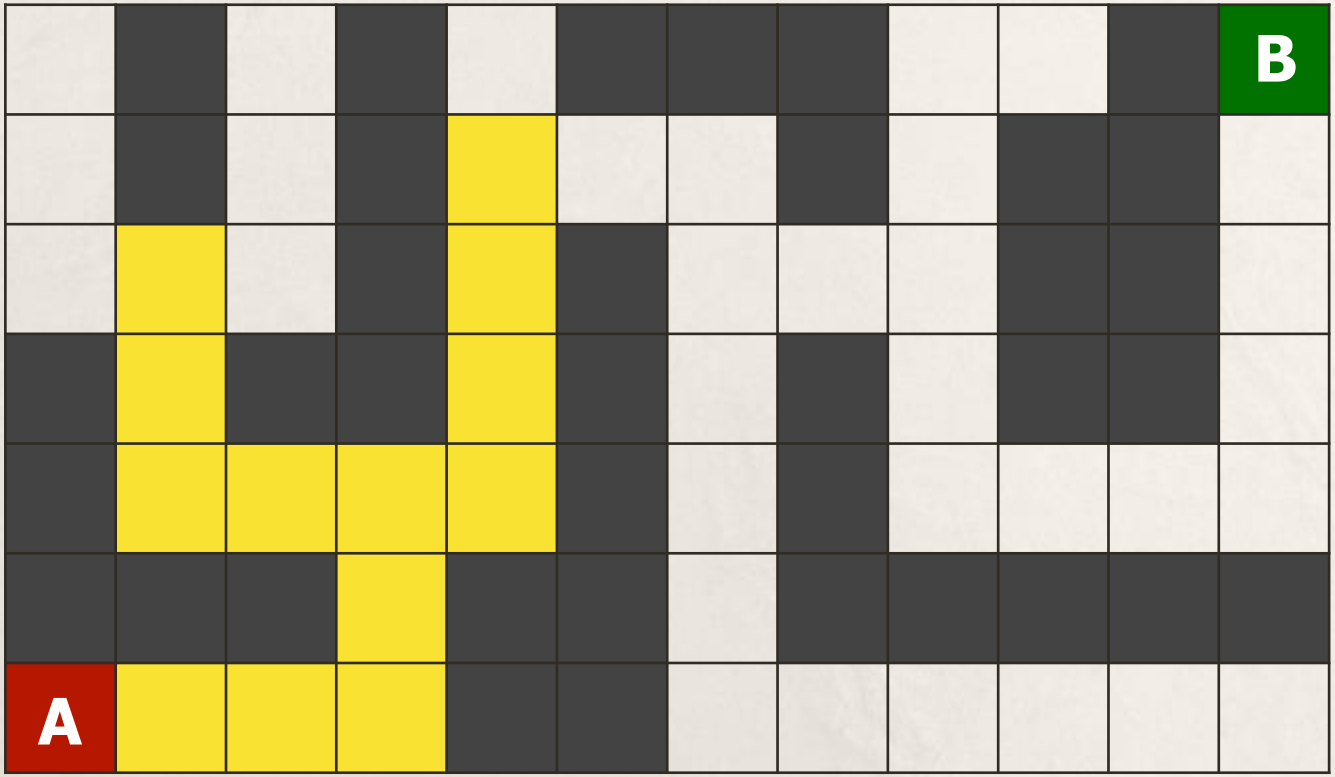
Breadth-First Search



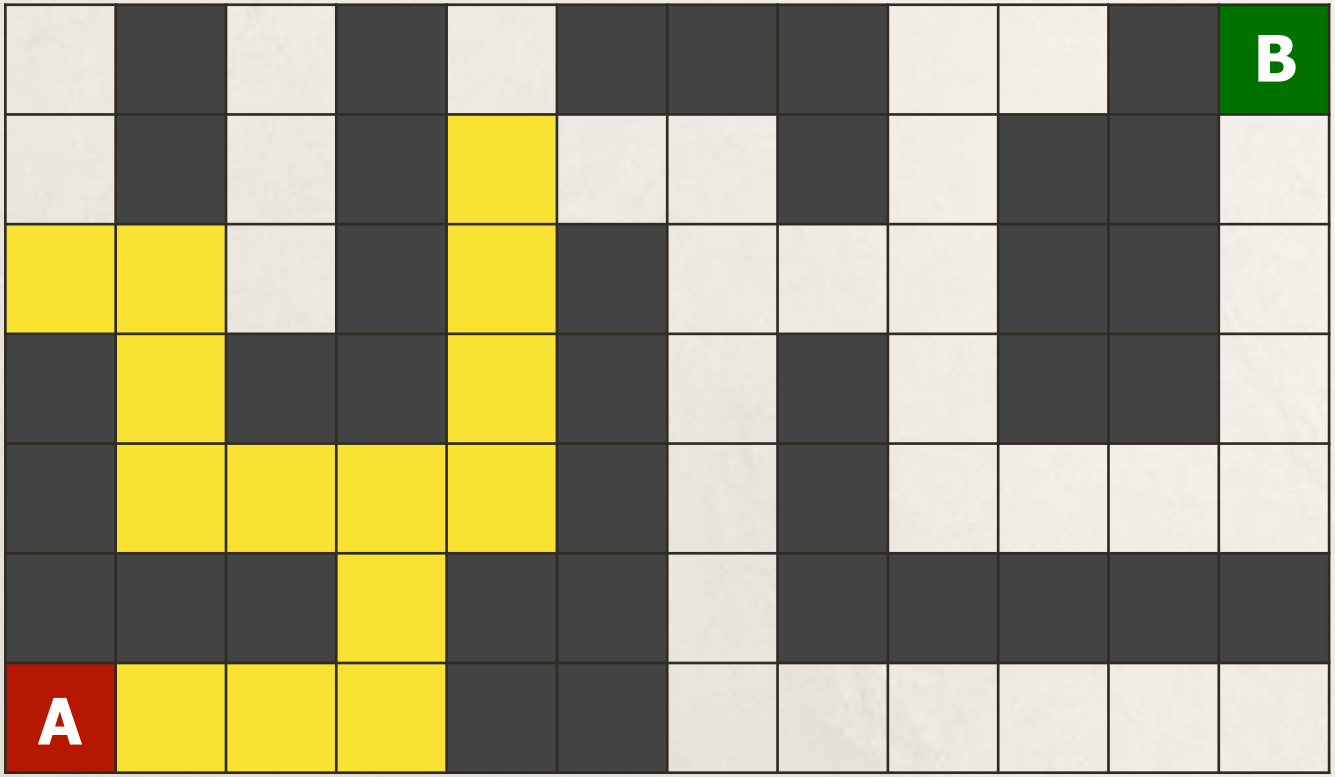
Breadth-First Search



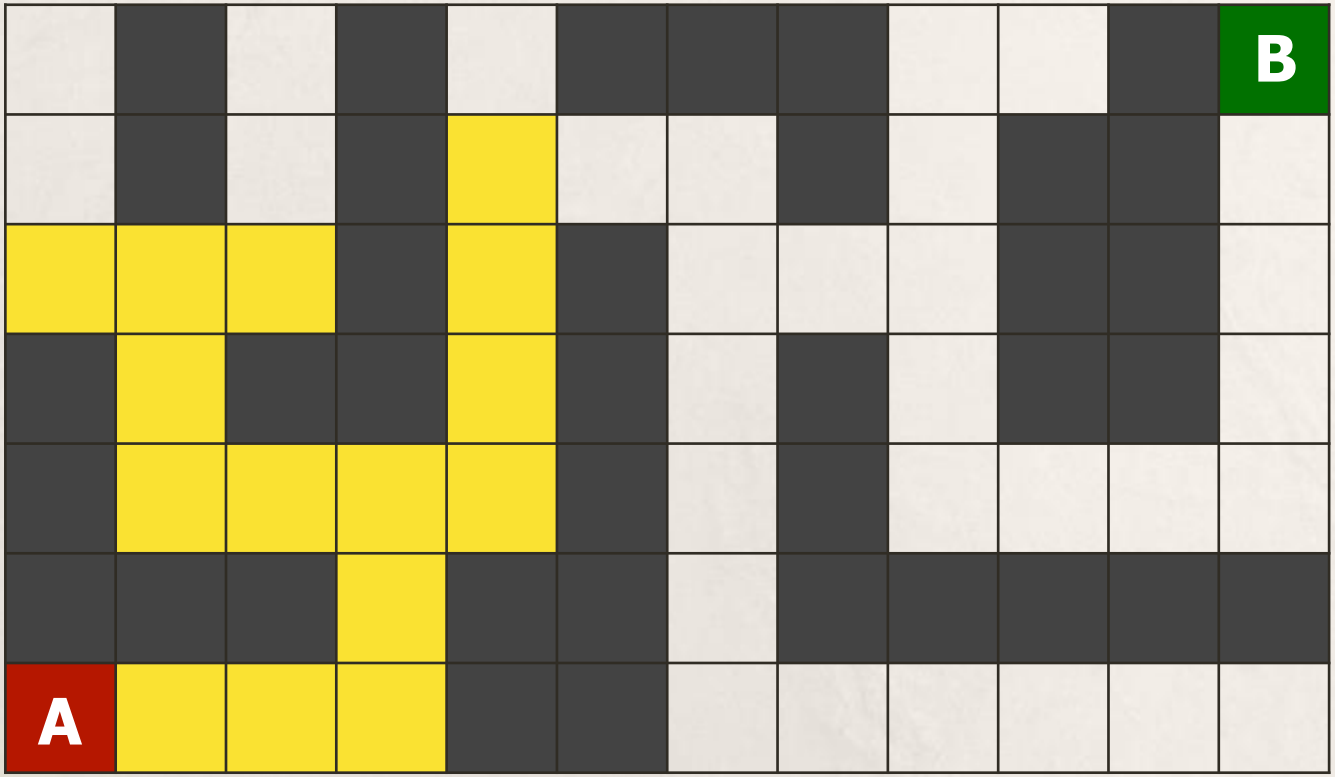
Breadth-First Search



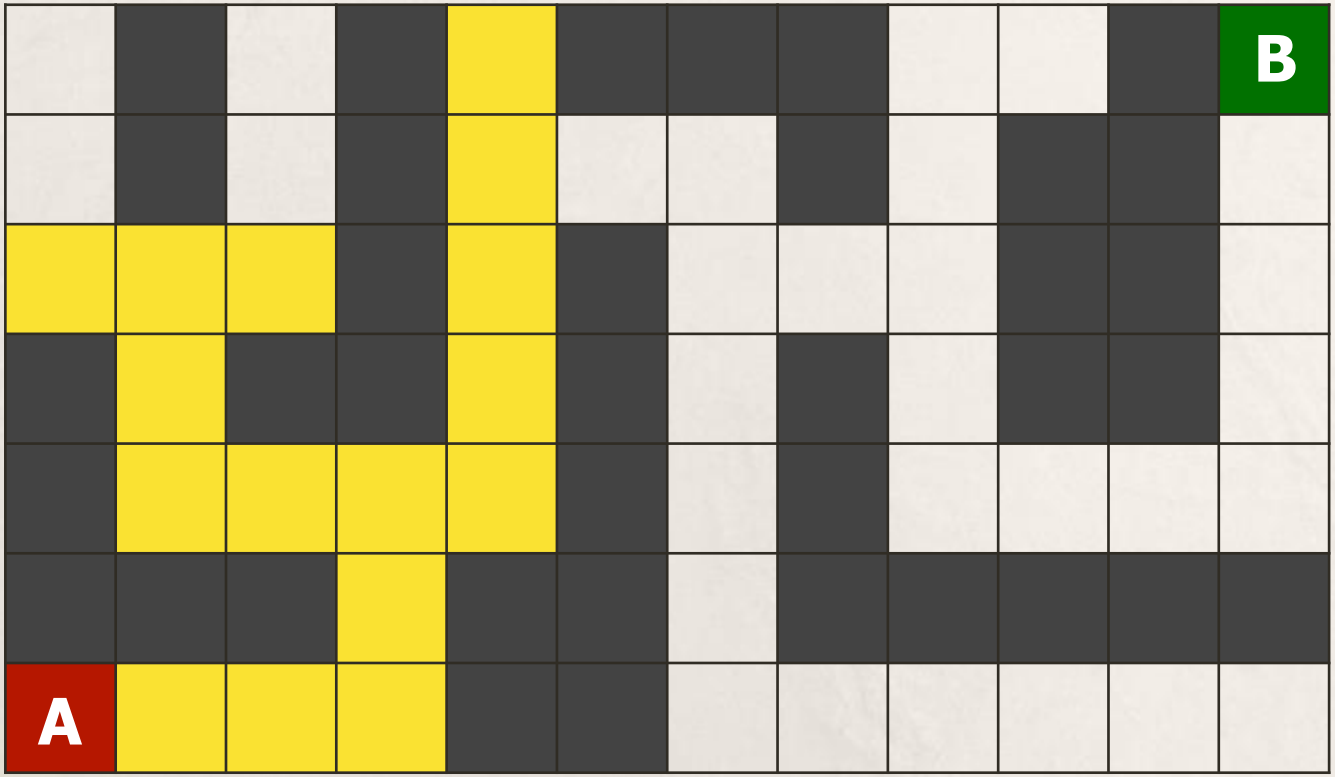
Breadth-First Search



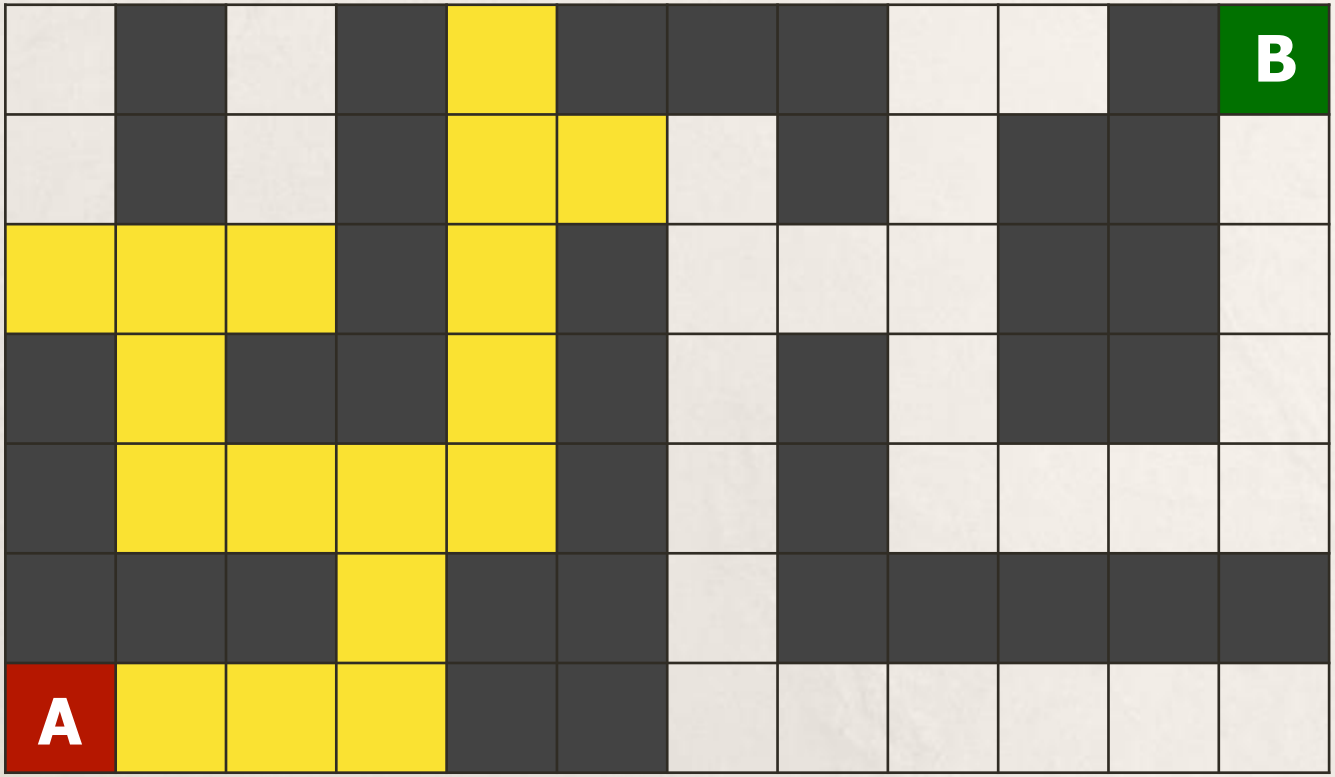
Breadth-First Search



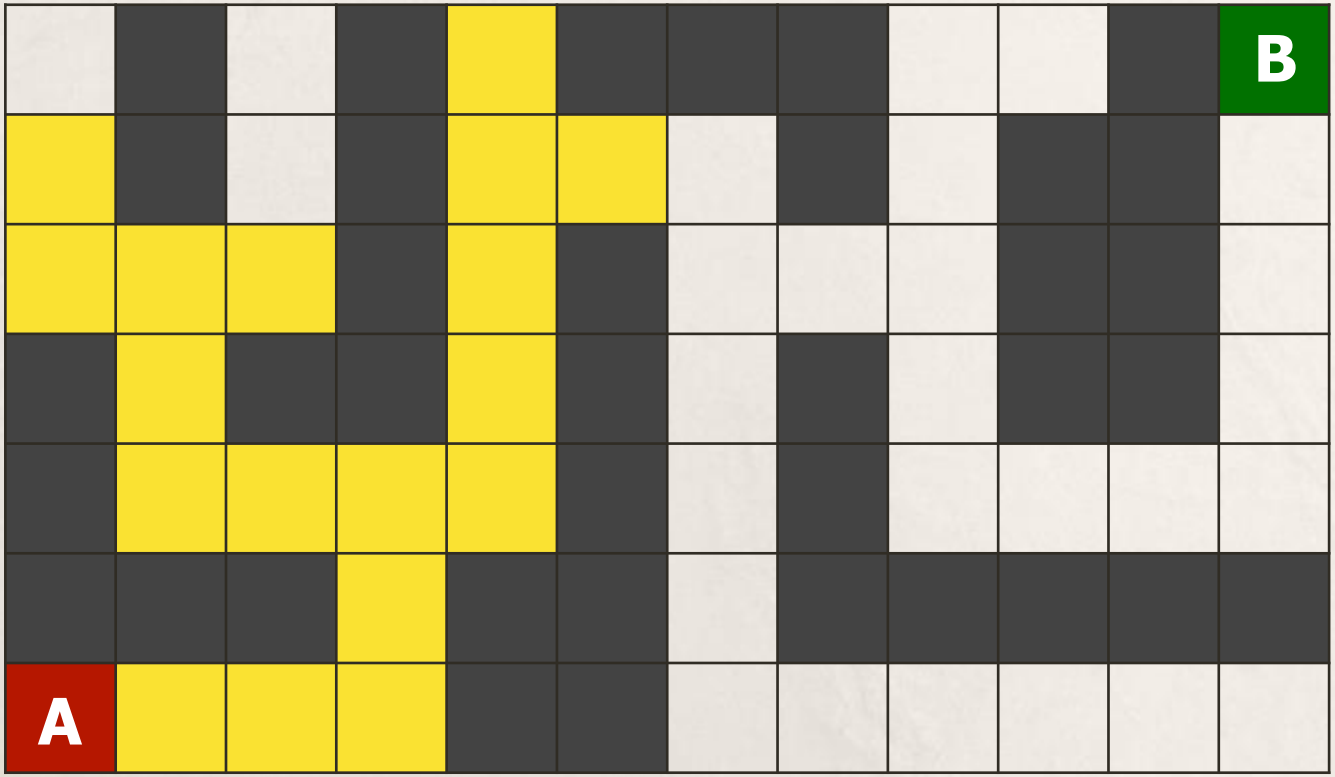
Breadth-First Search



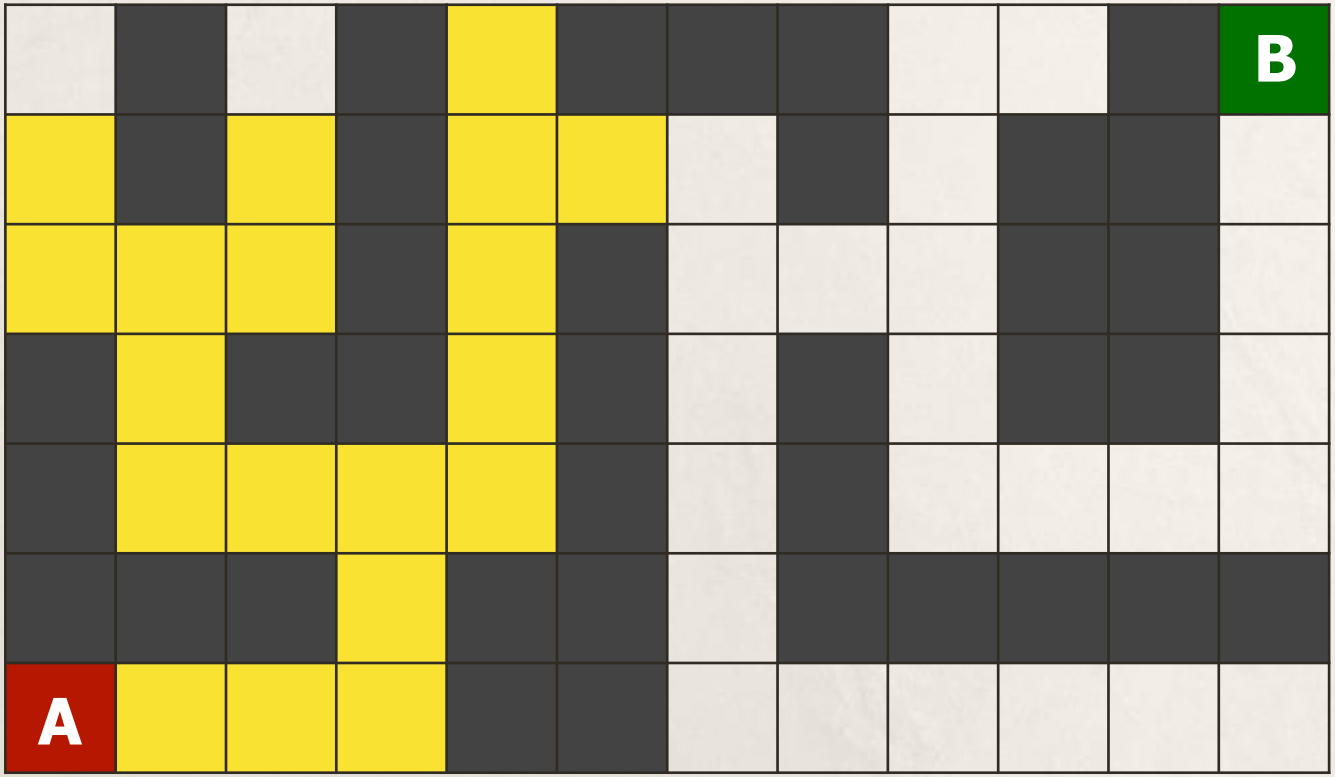
Breadth-First Search



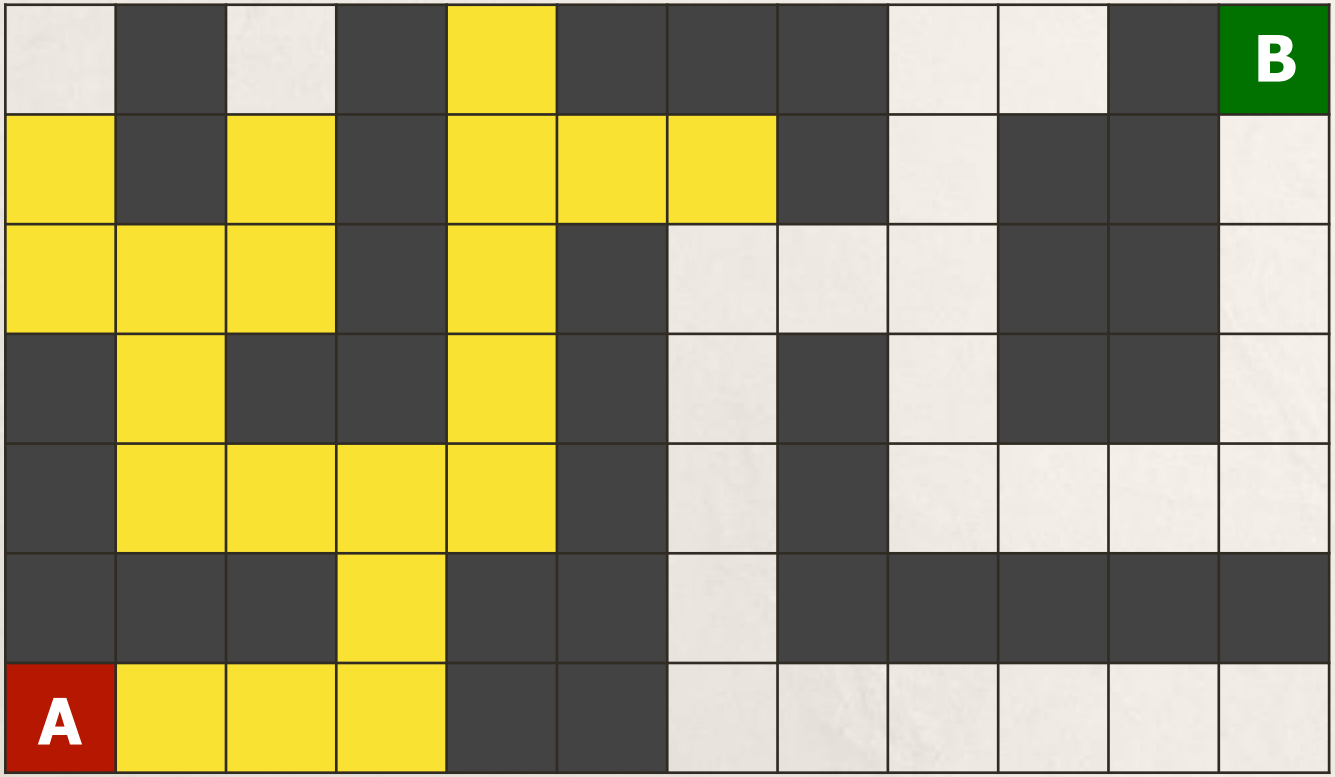
Breadth-First Search



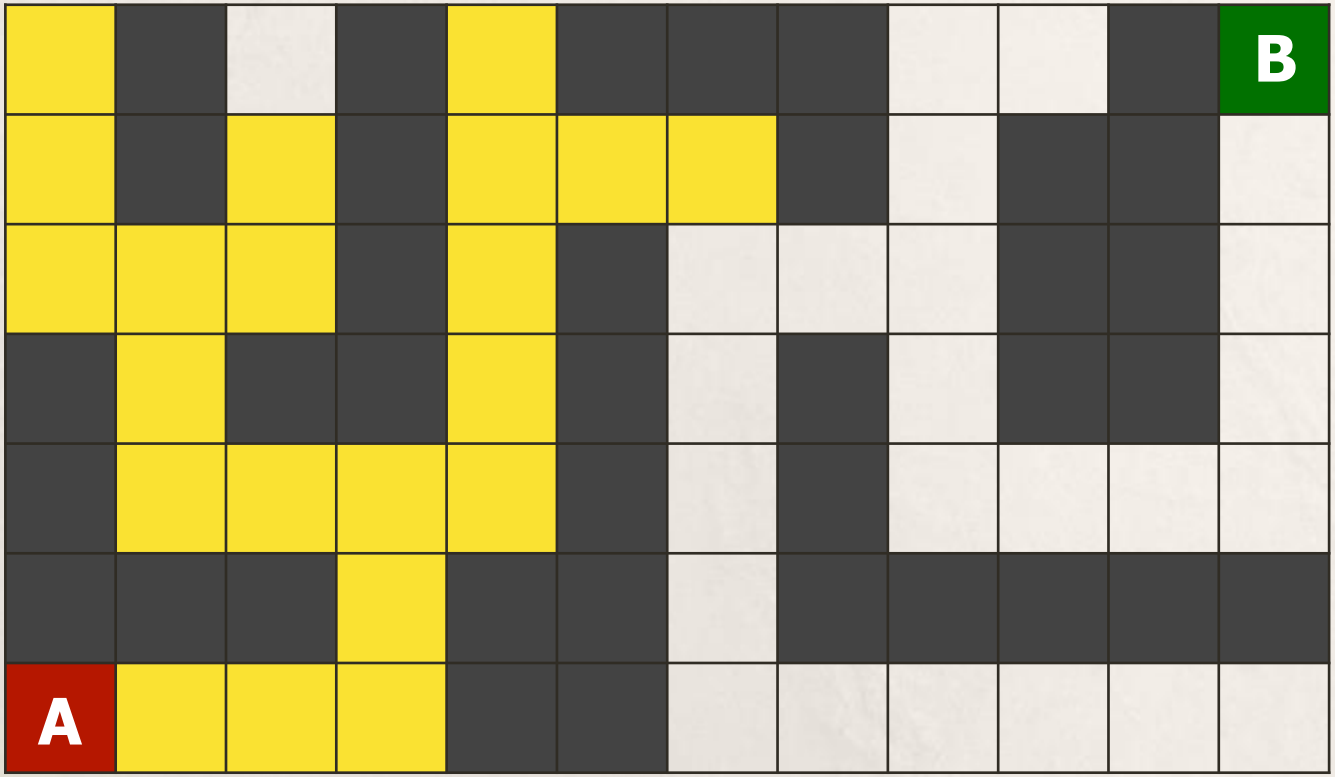
Breadth-First Search



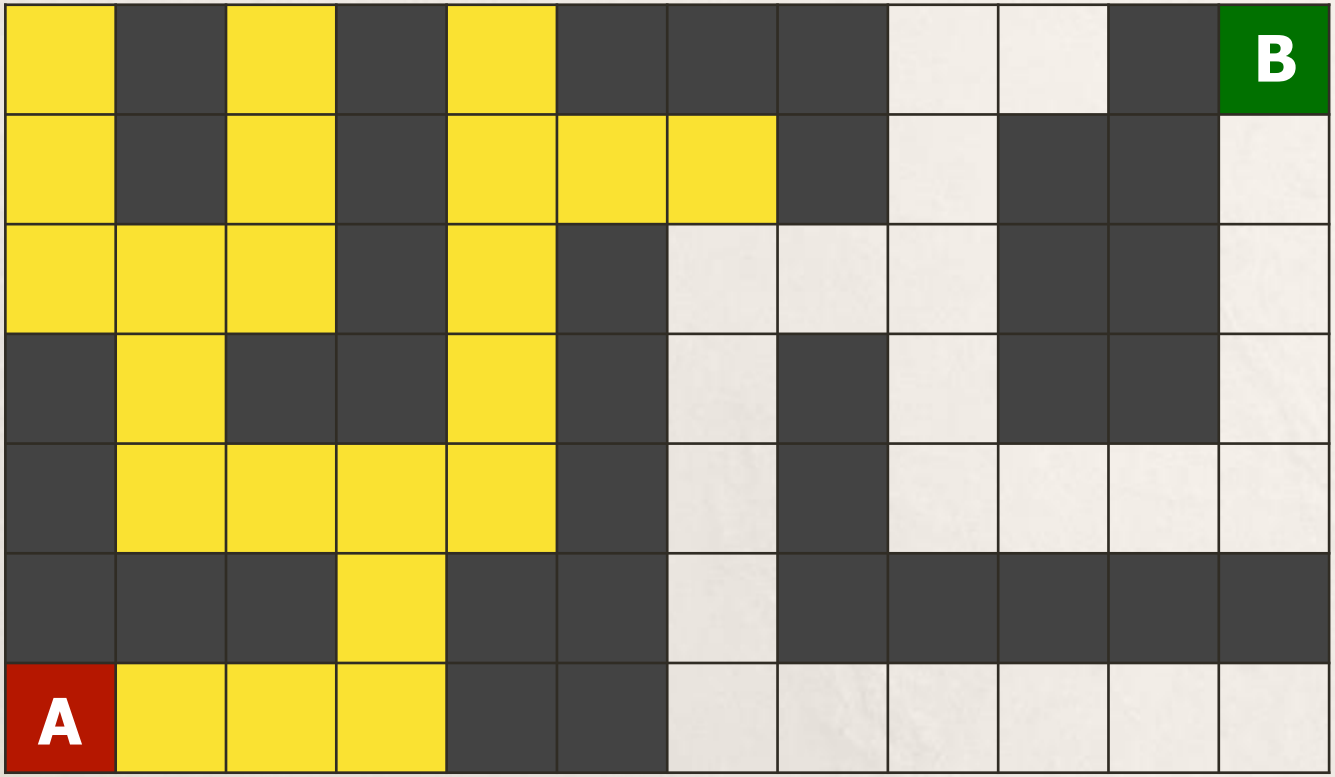
Breadth-First Search



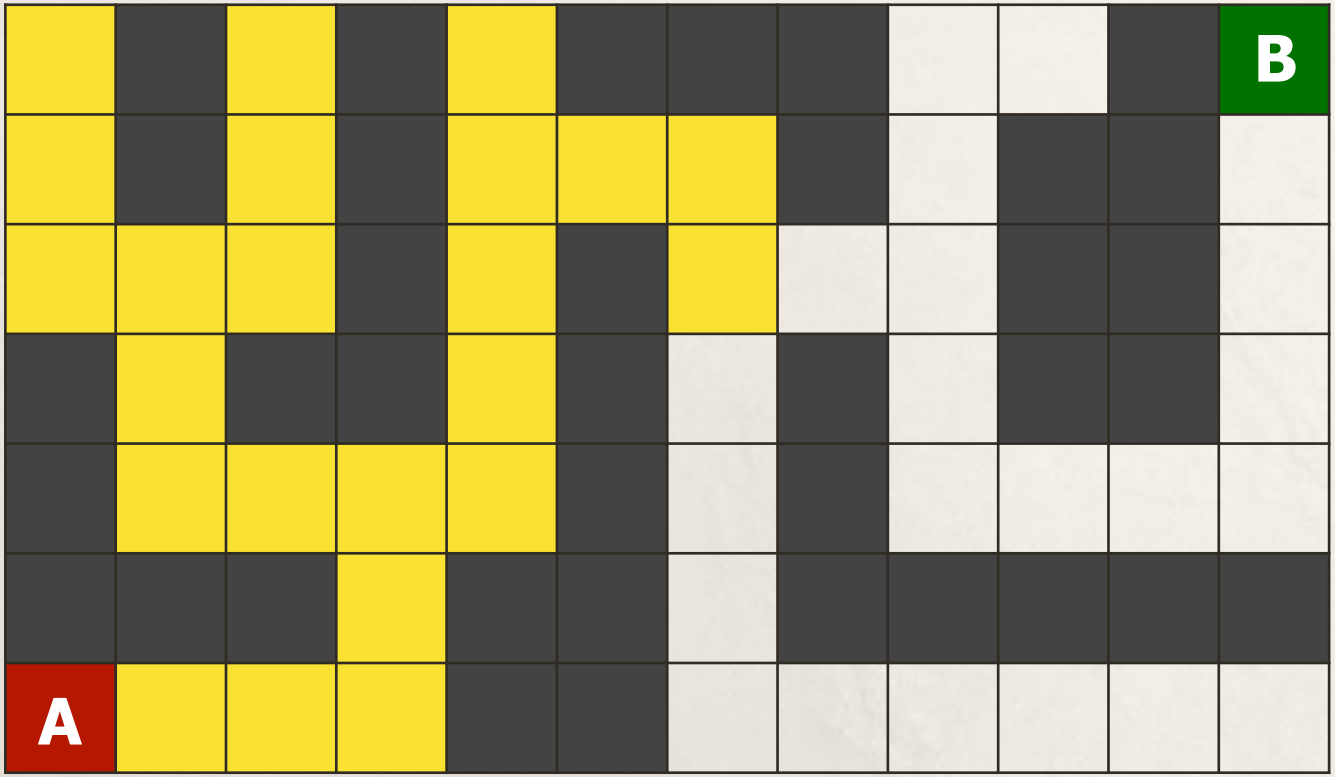
Breadth-First Search



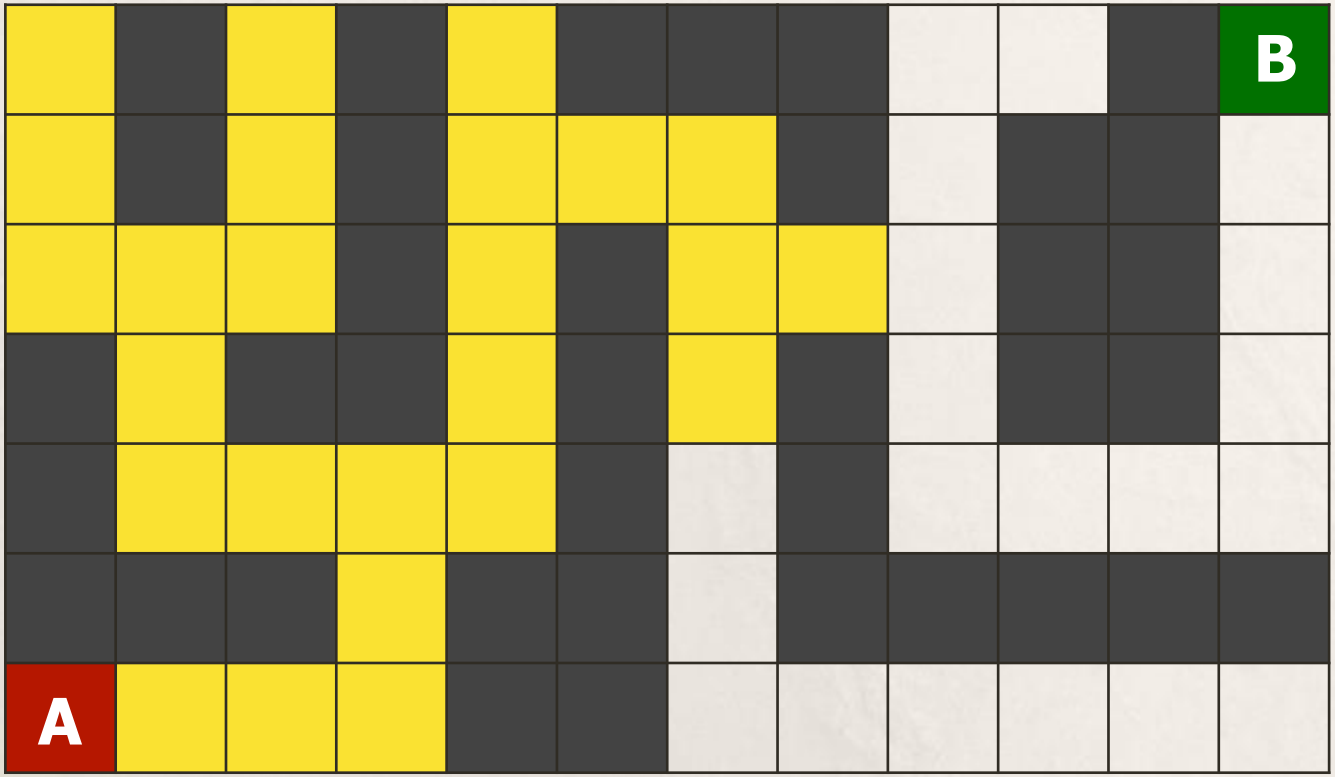
Breadth-First Search



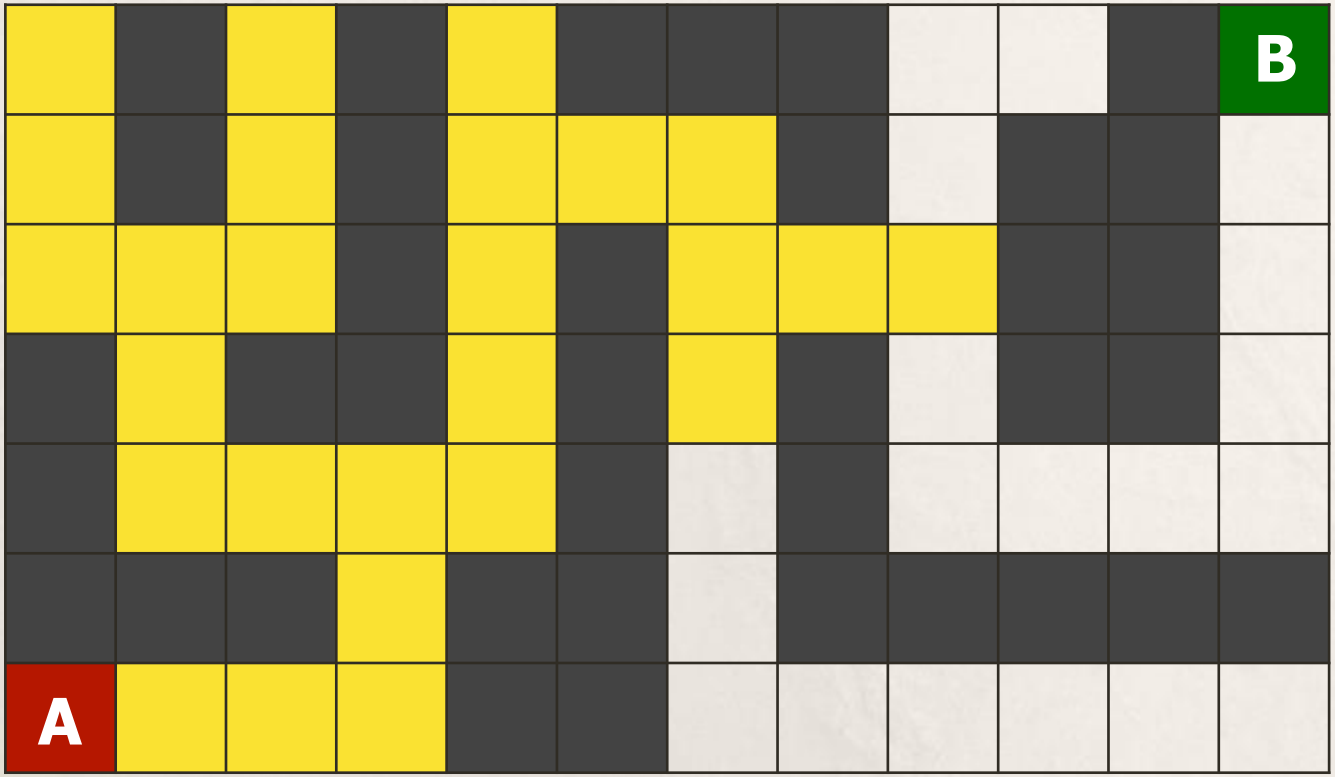
Breadth-First Search



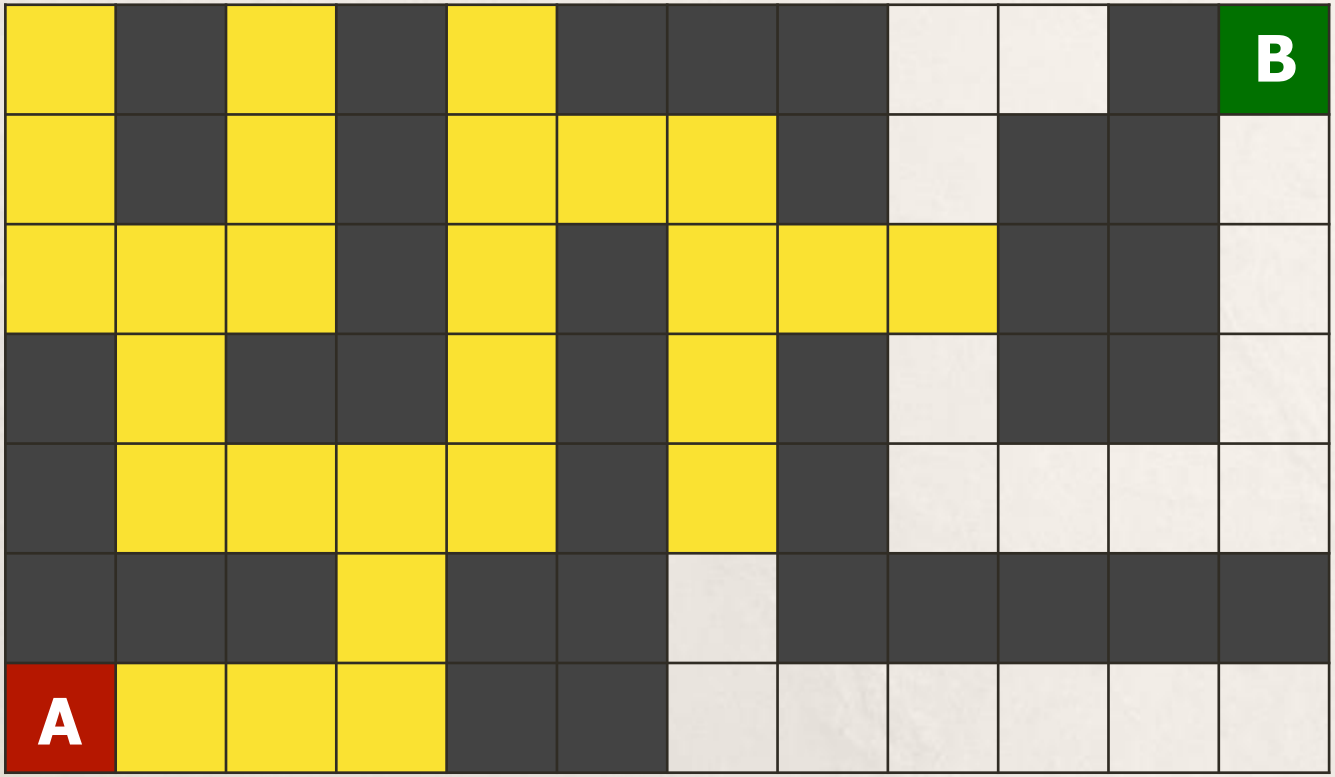
Breadth-First Search



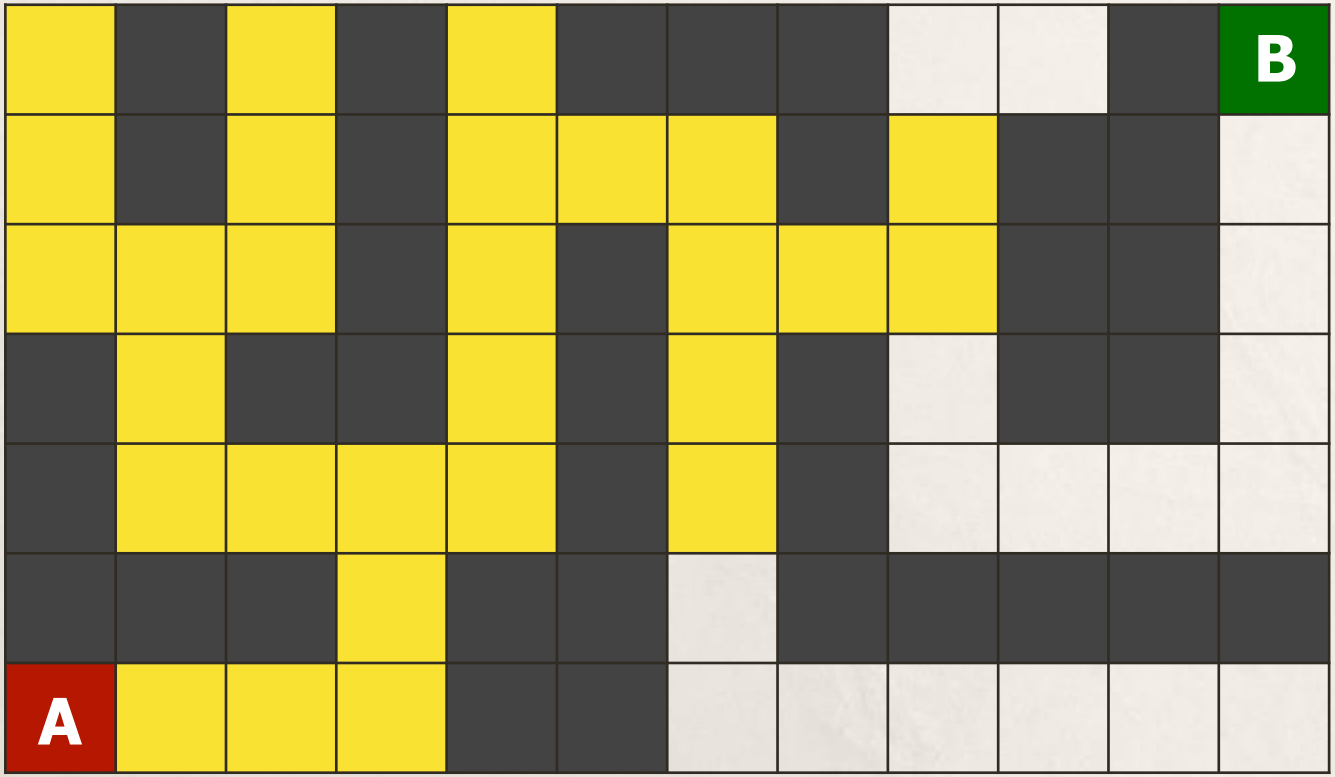
Breadth-First Search



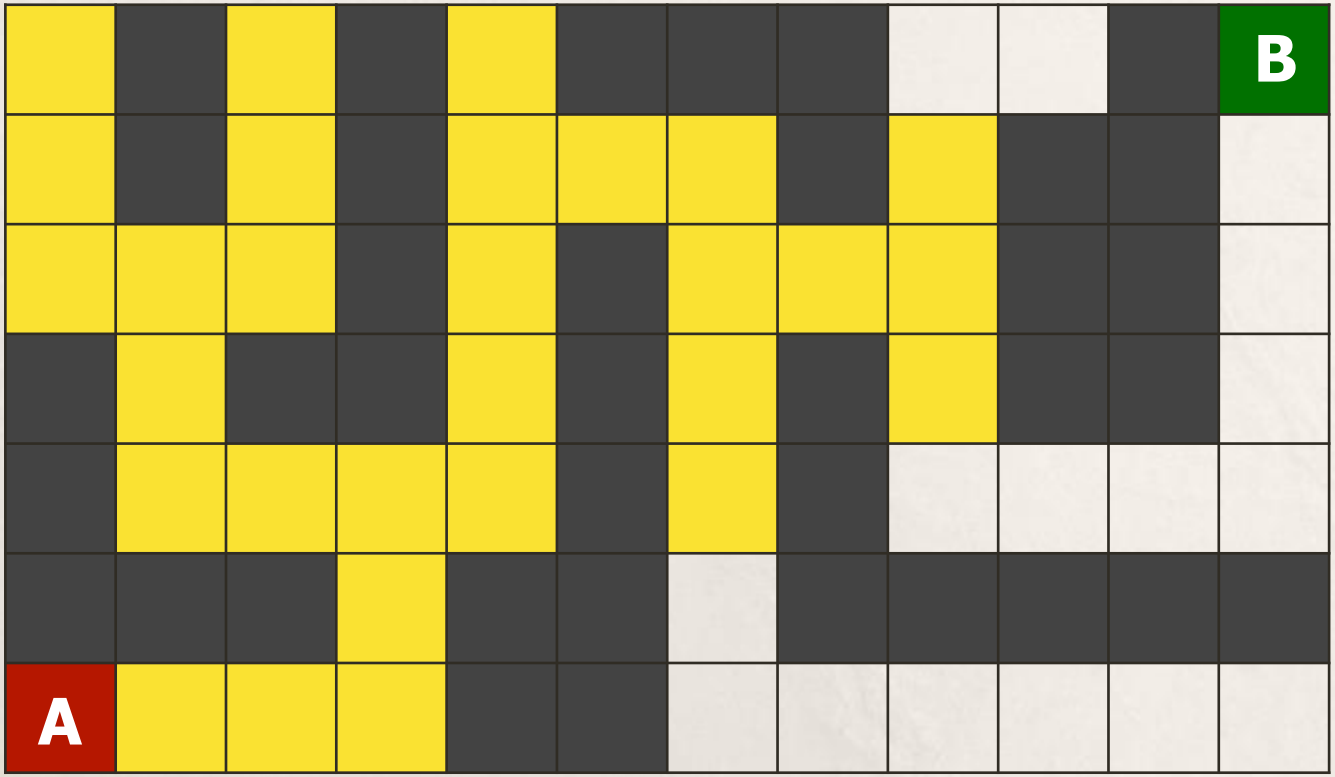
Breadth-First Search



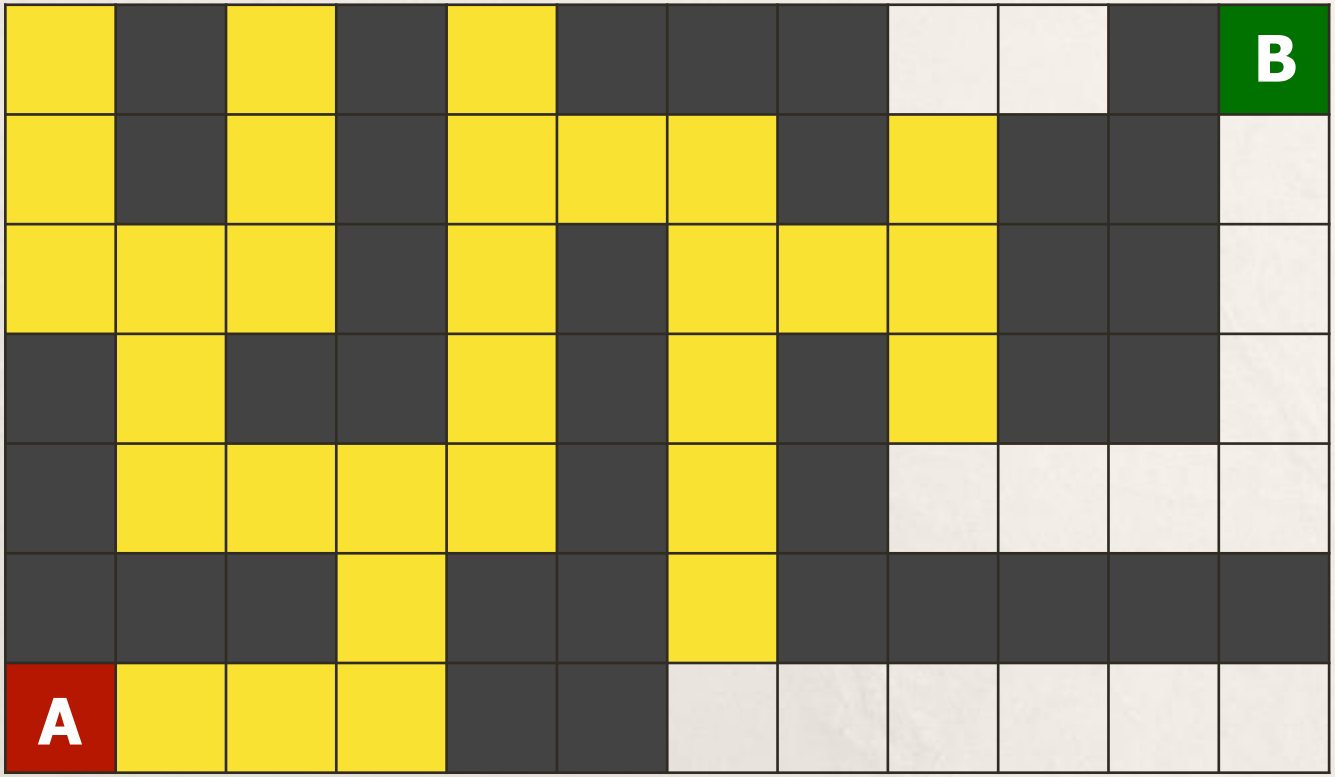
Breadth-First Search



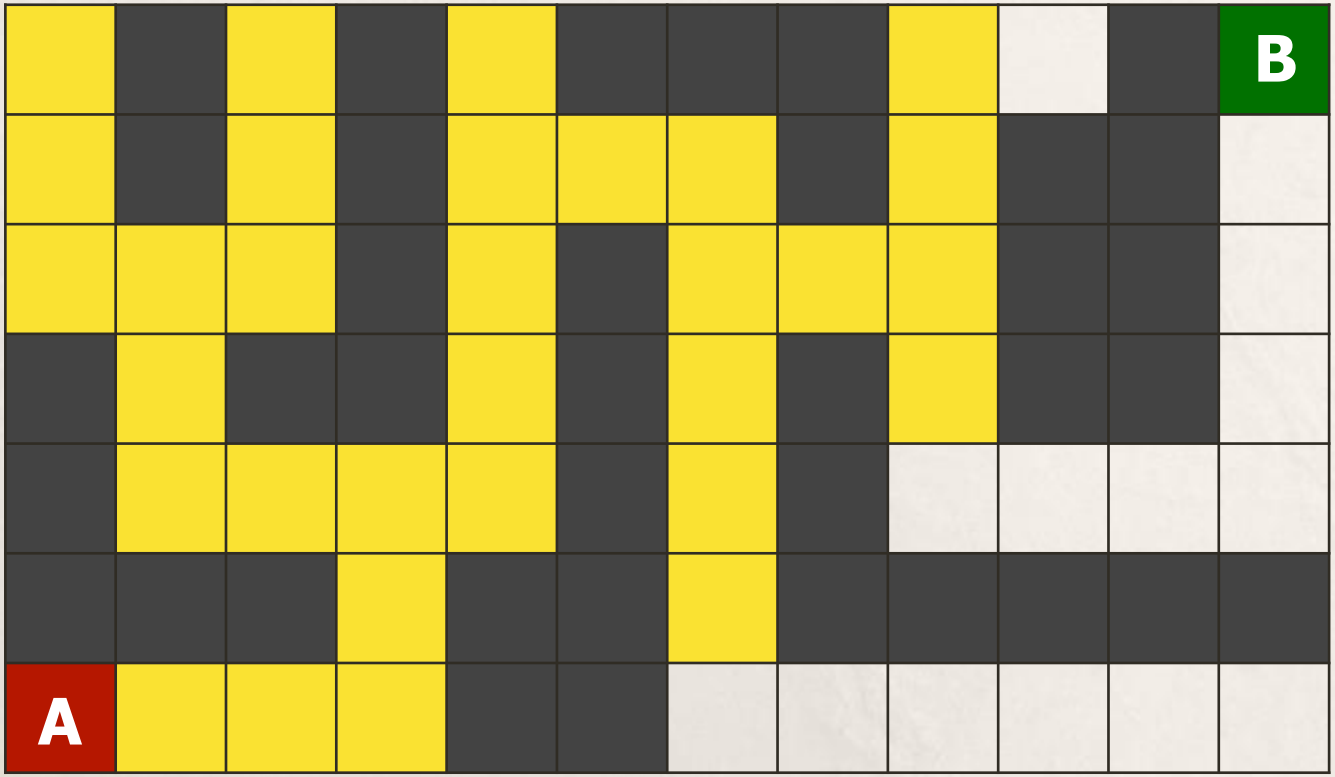
Breadth-First Search



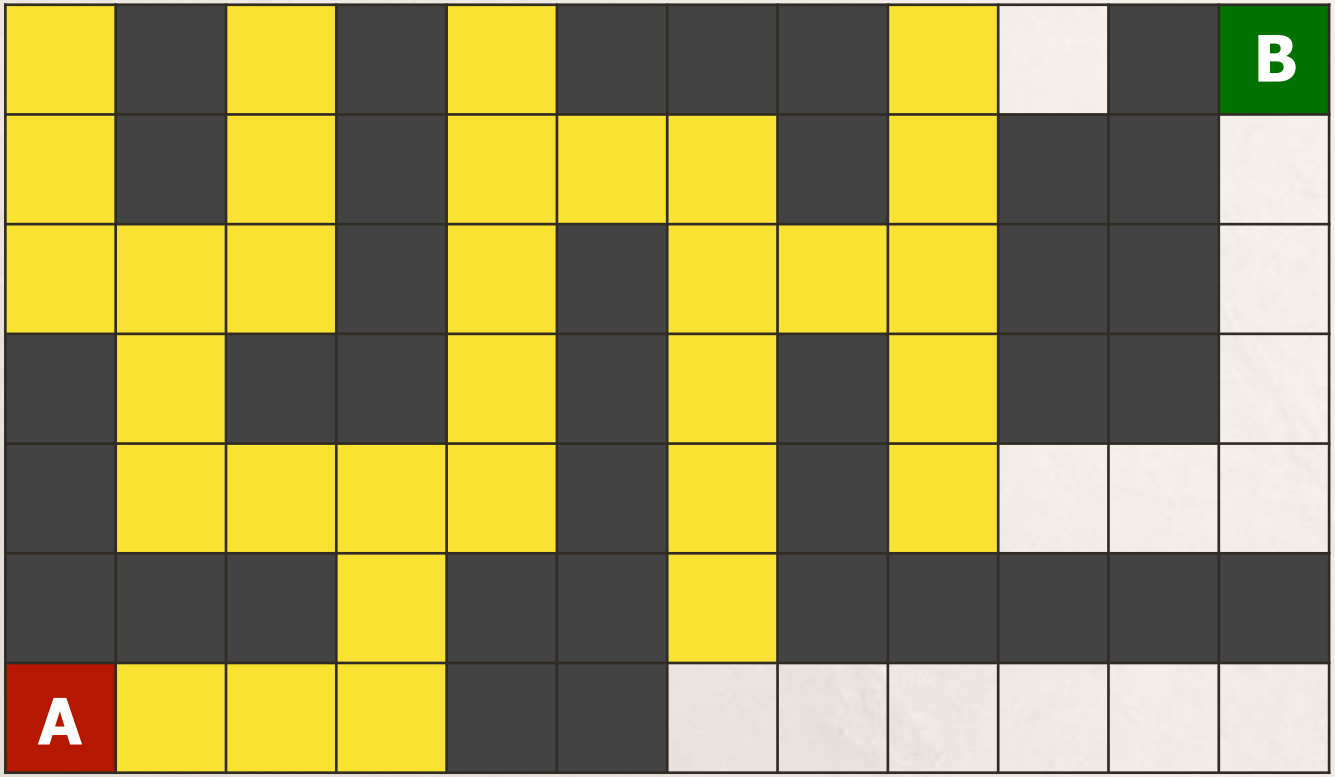
Breadth-First Search



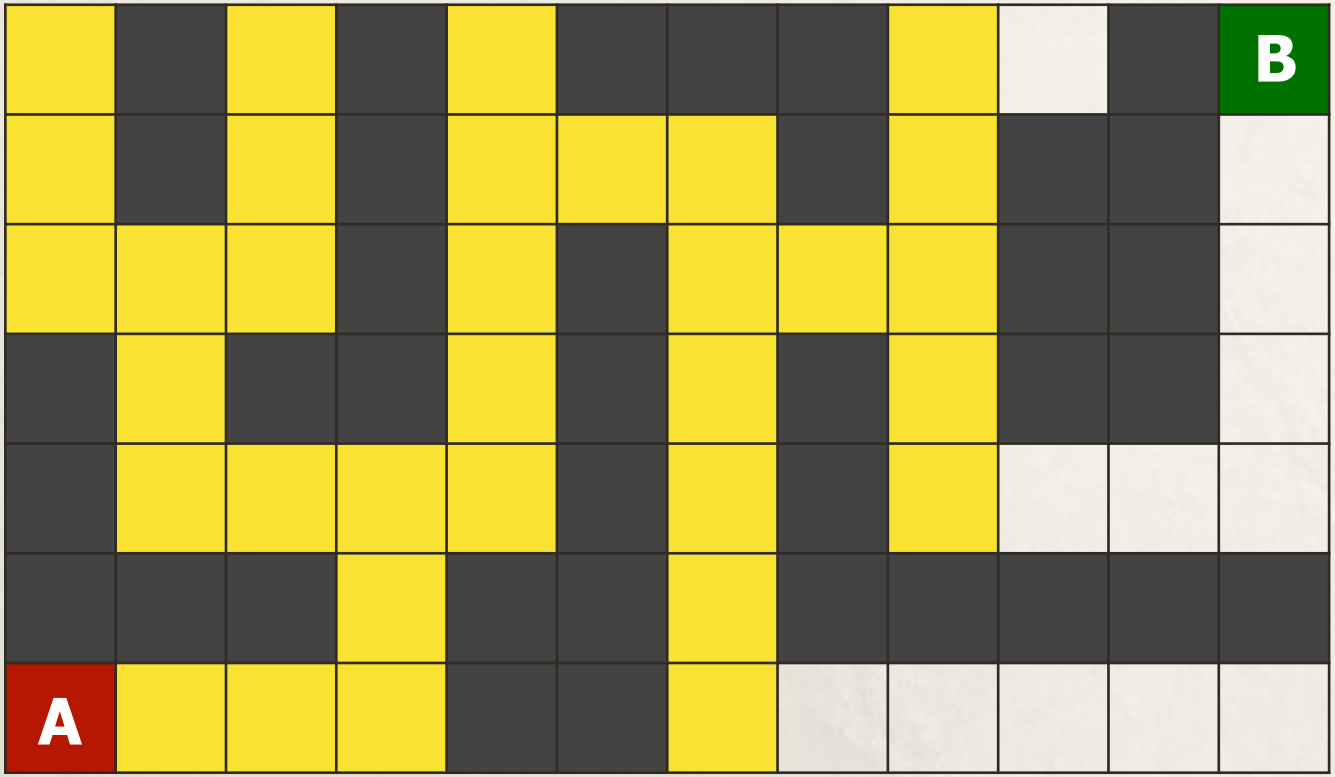
Breadth-First Search



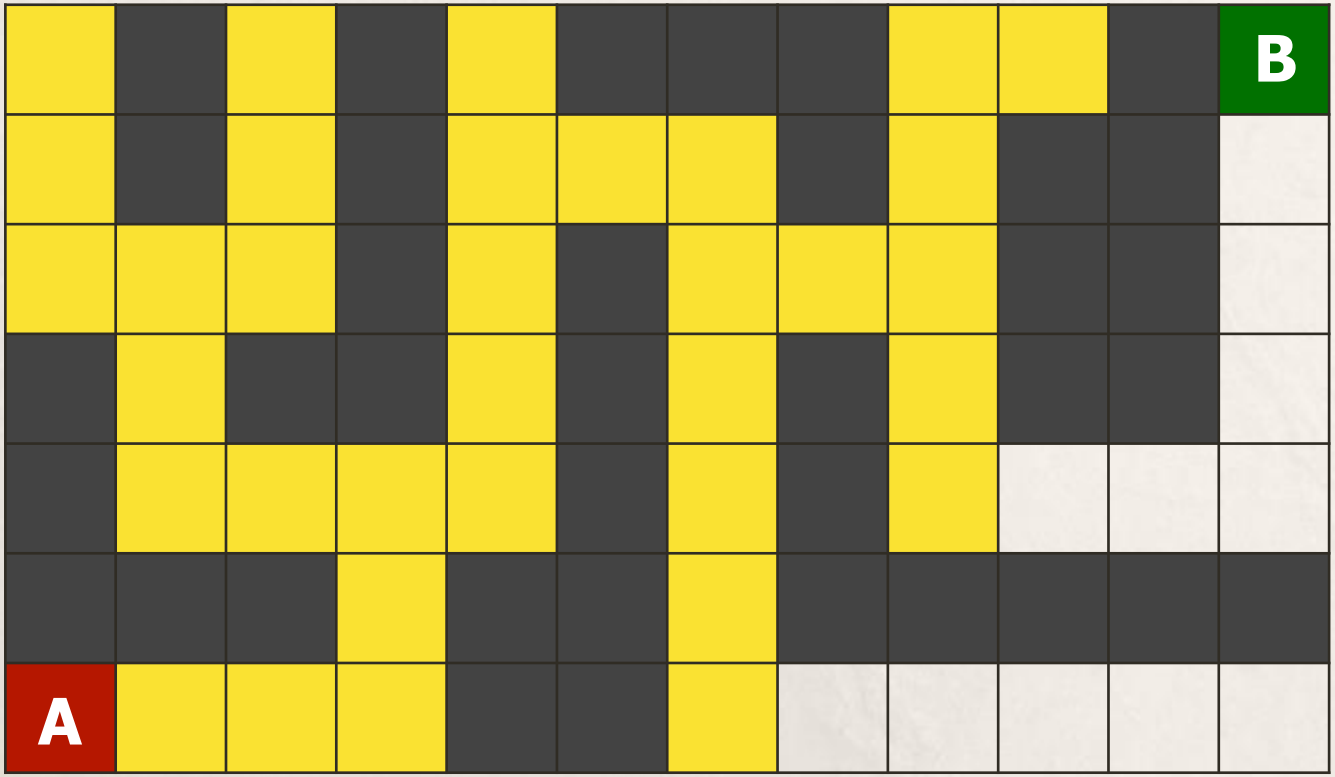
Breadth-First Search



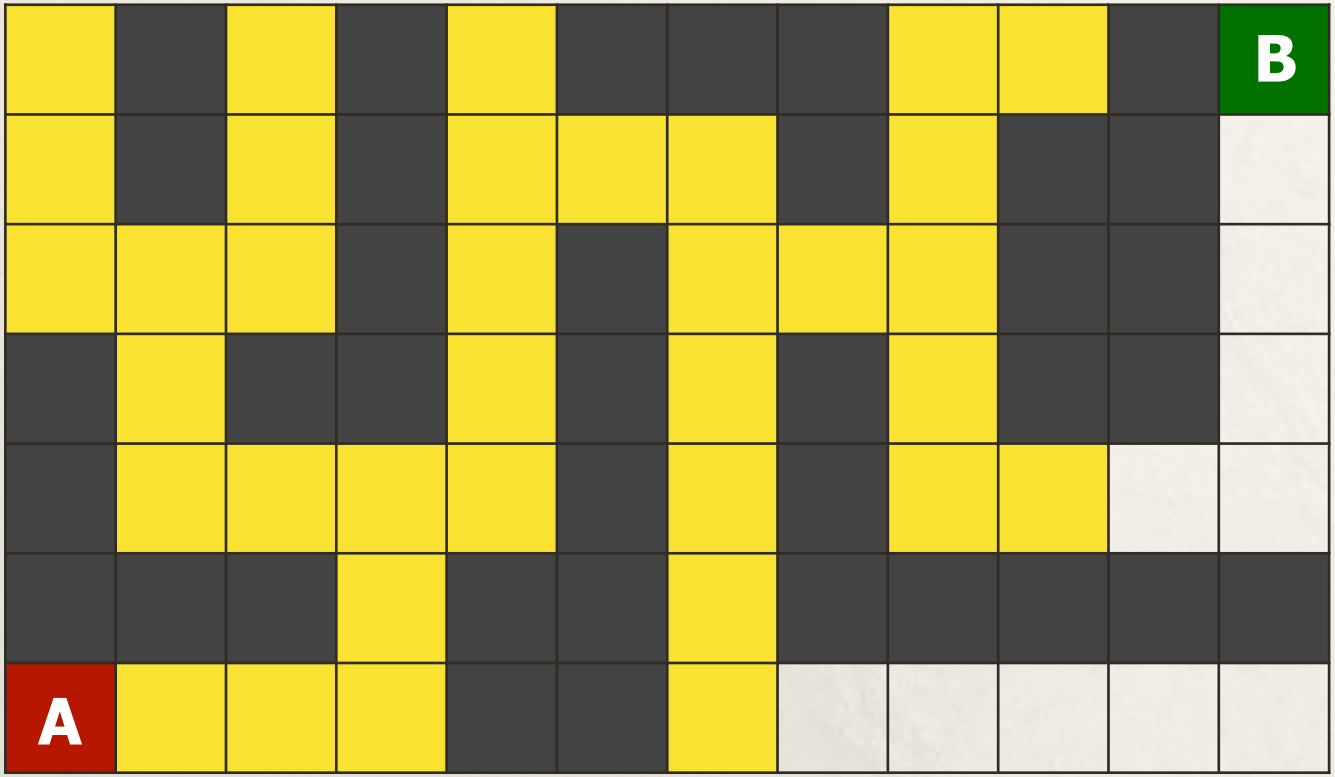
Breadth-First Search



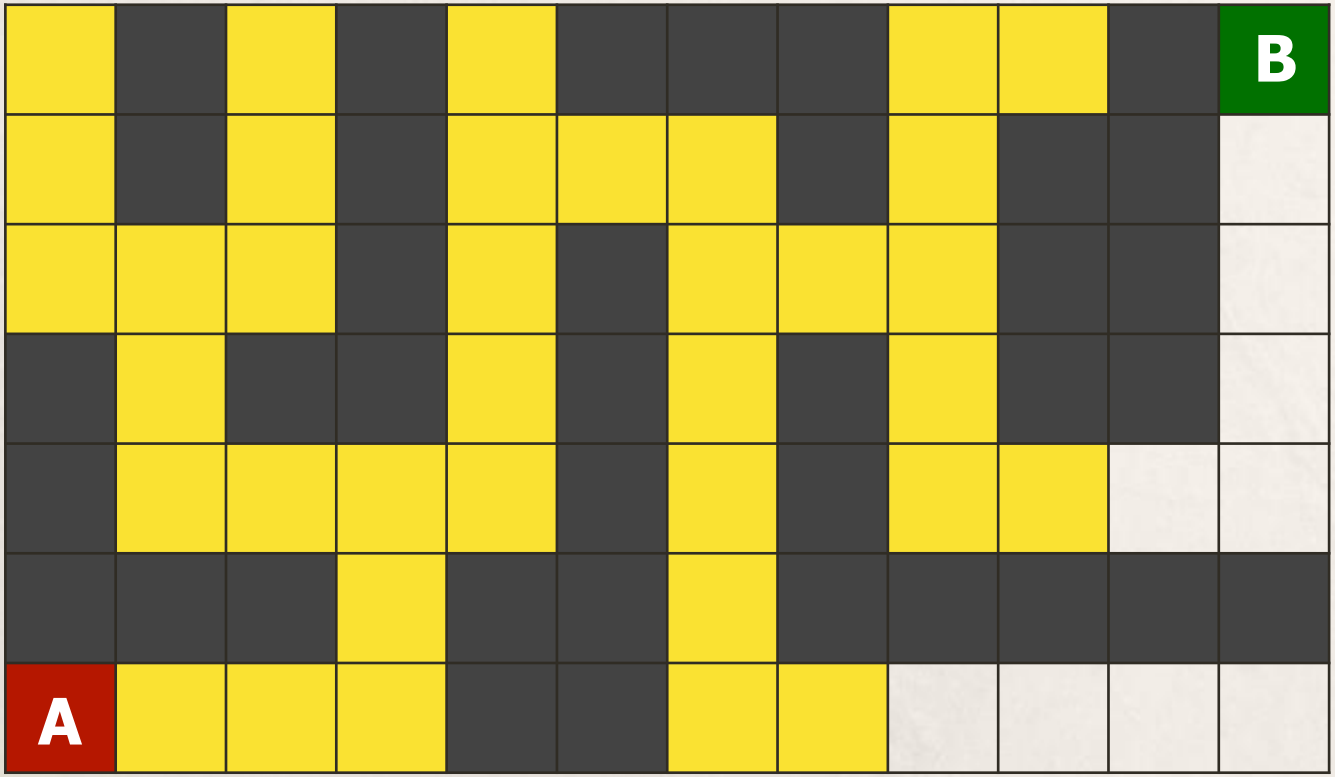
Breadth-First Search



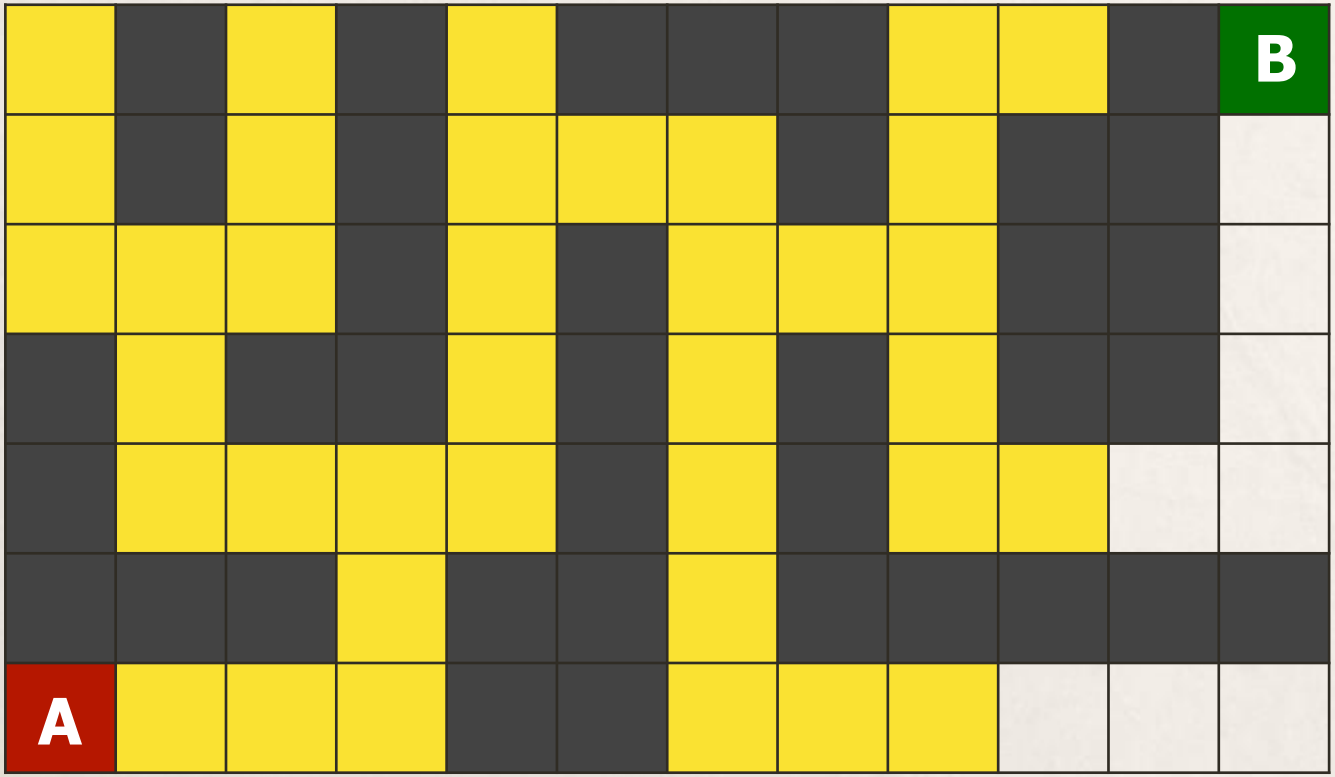
Breadth-First Search



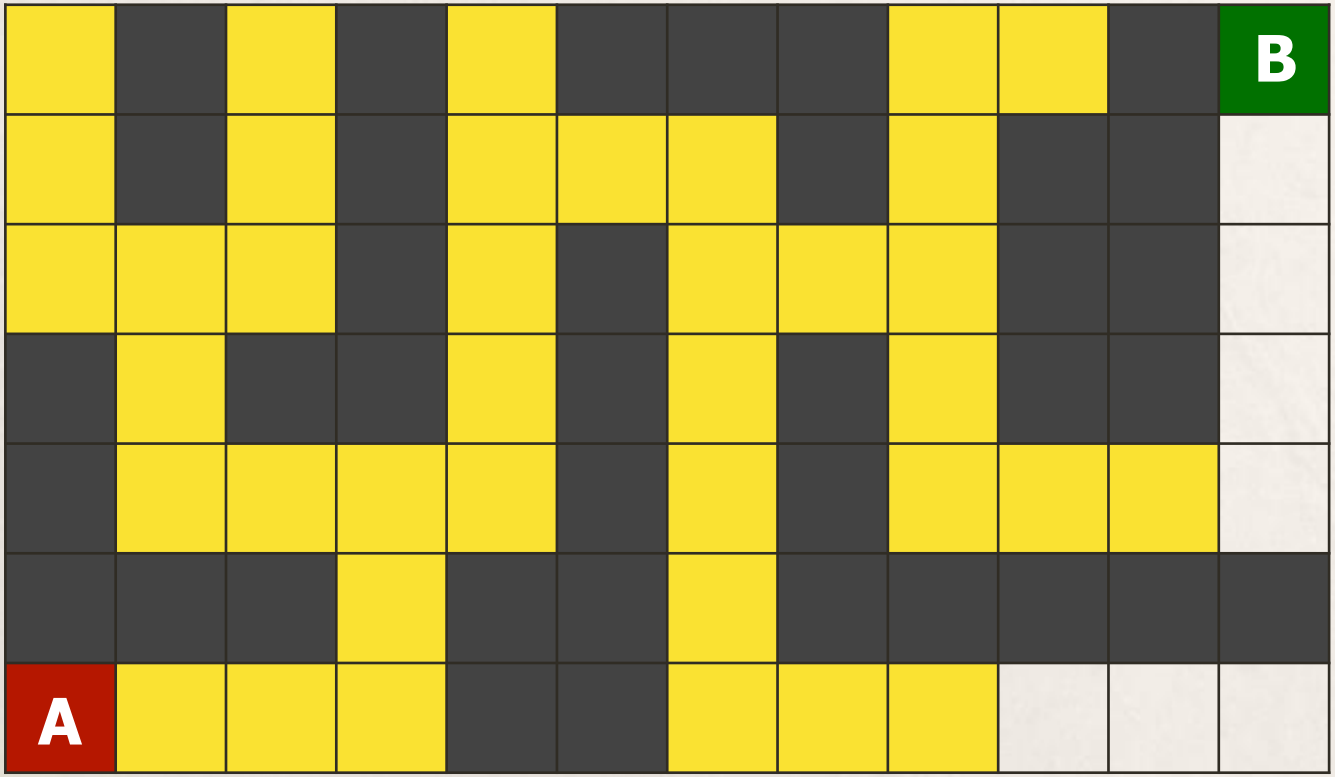
Breadth-First Search



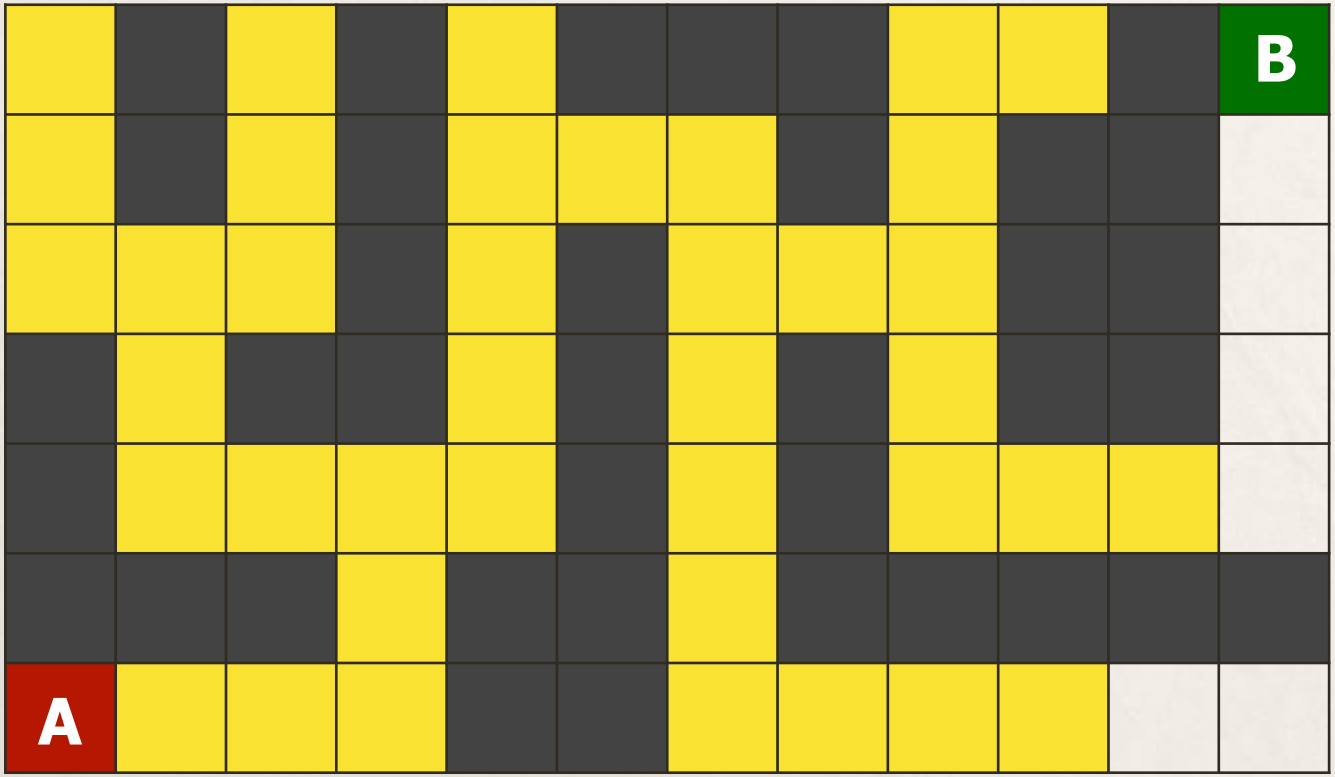
Breadth-First Search



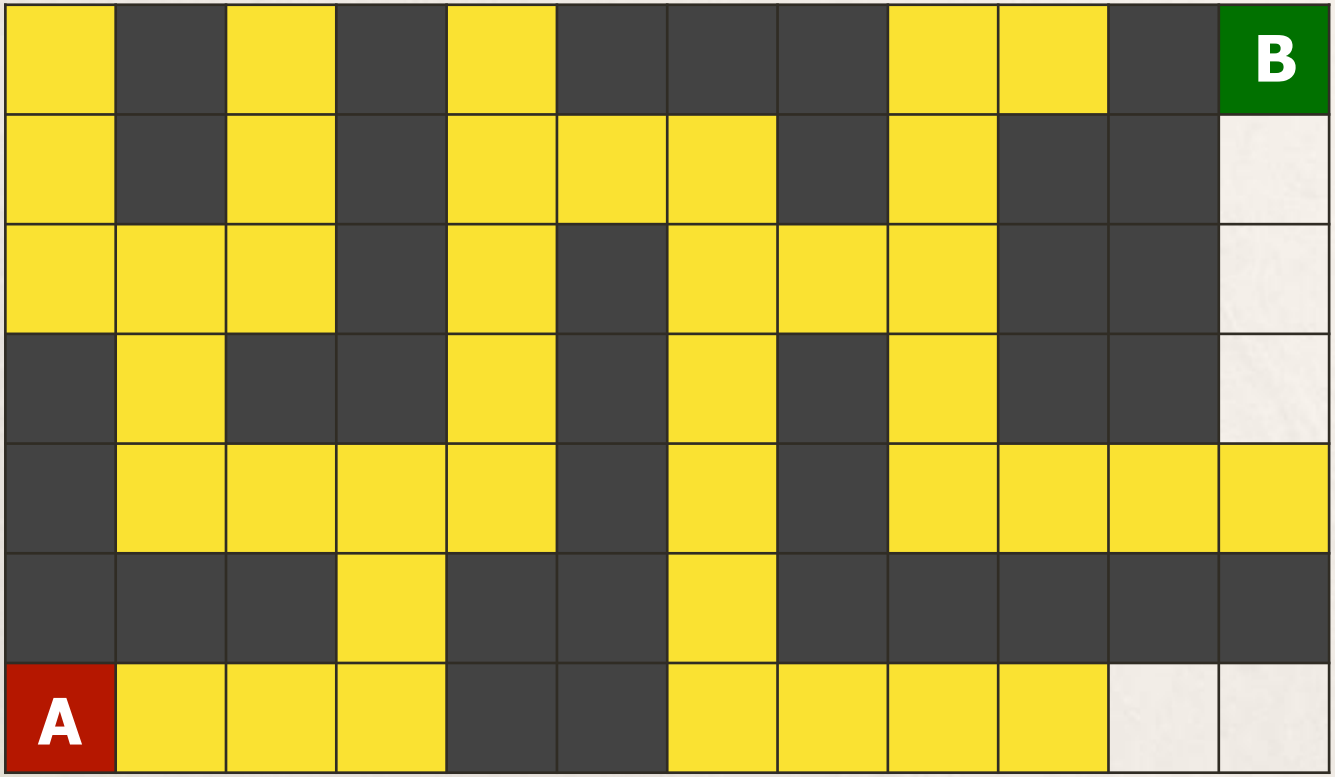
Breadth-First Search



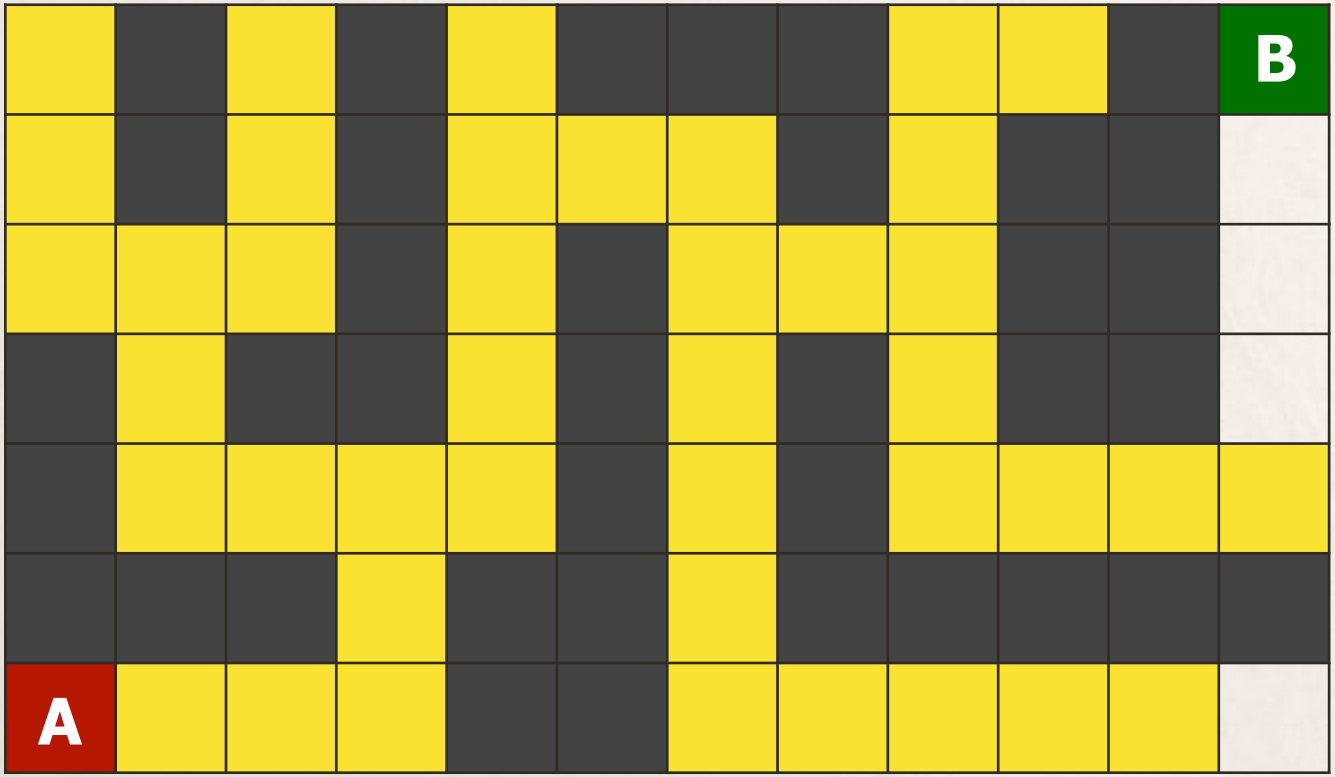
Breadth-First Search



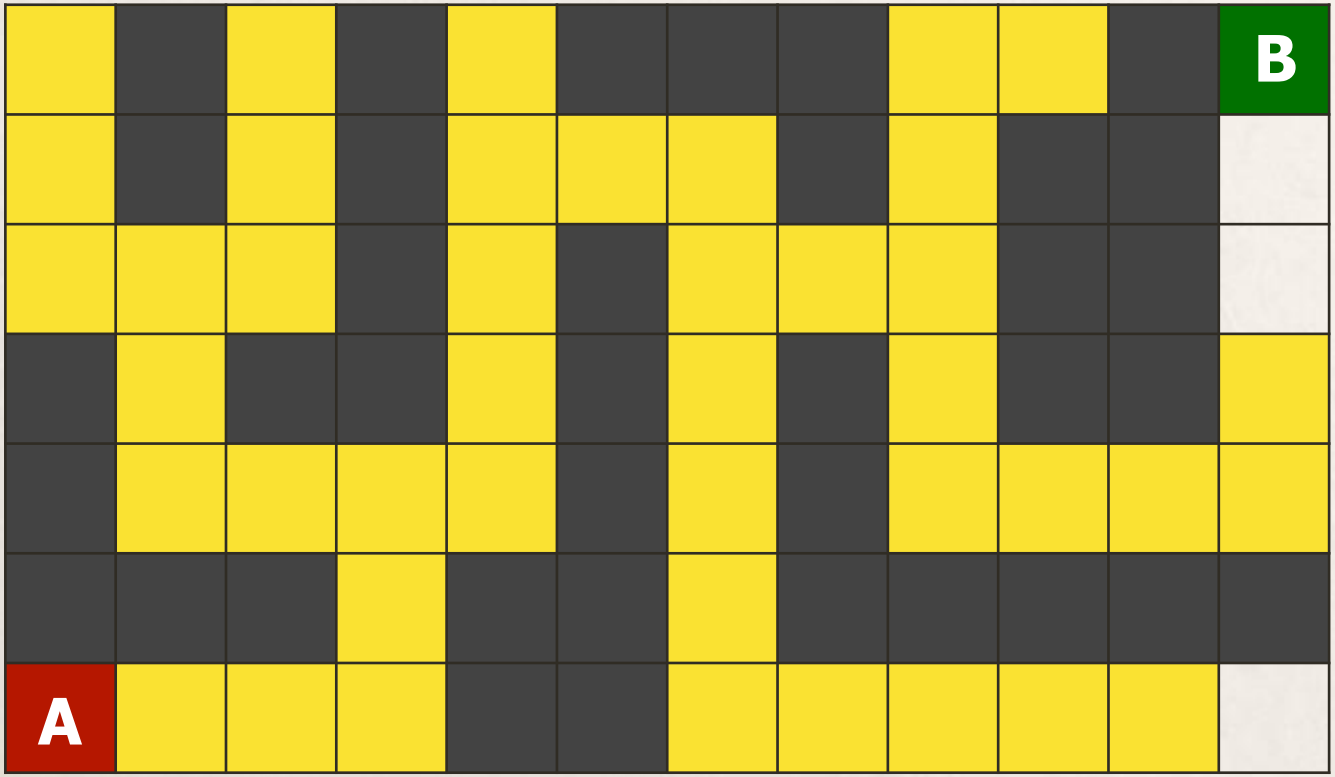
Breadth-First Search



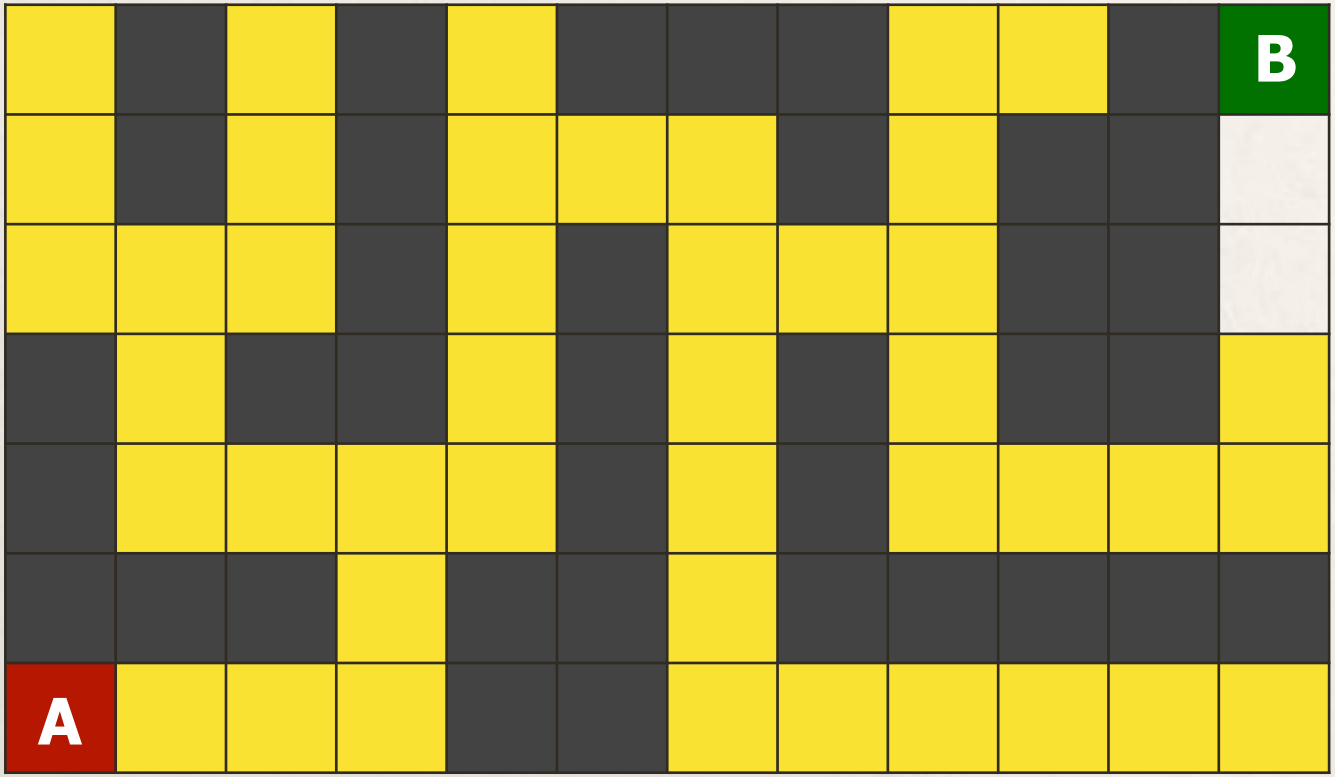
Breadth-First Search



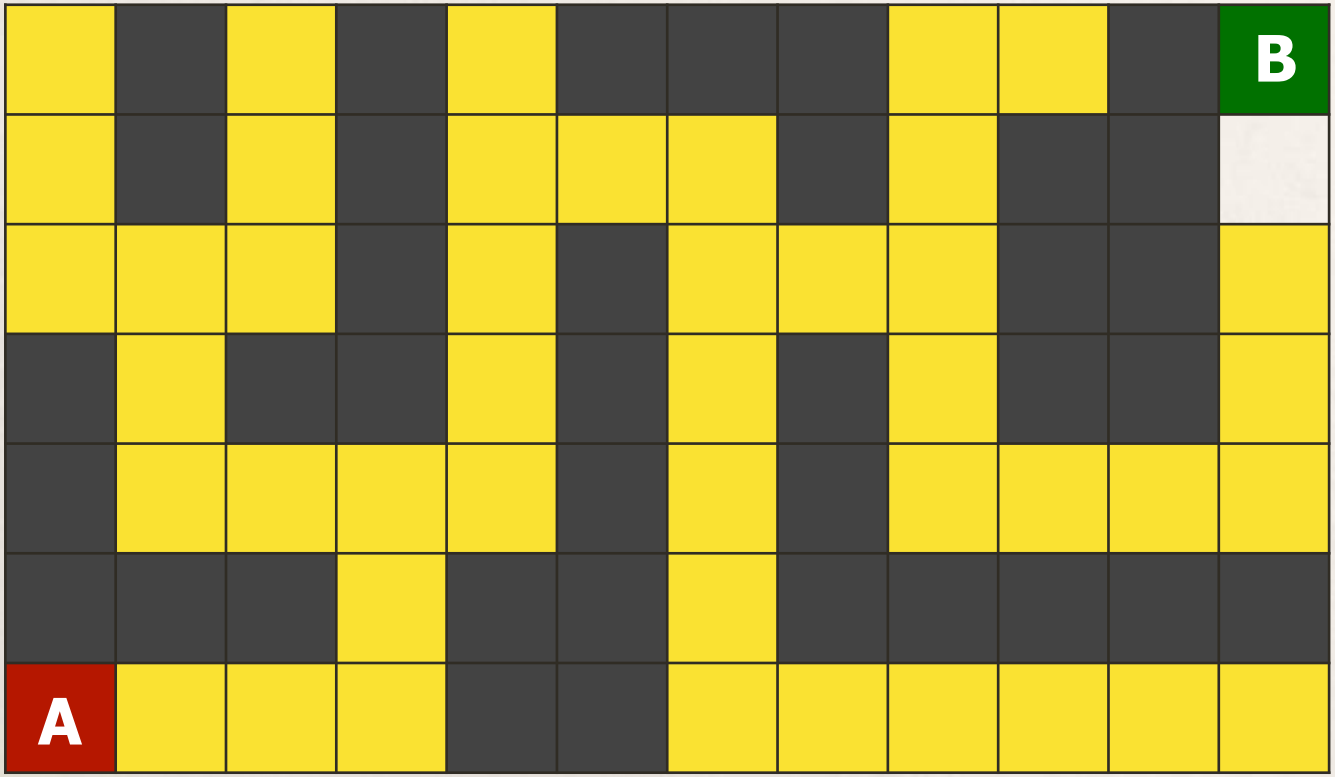
Breadth-First Search



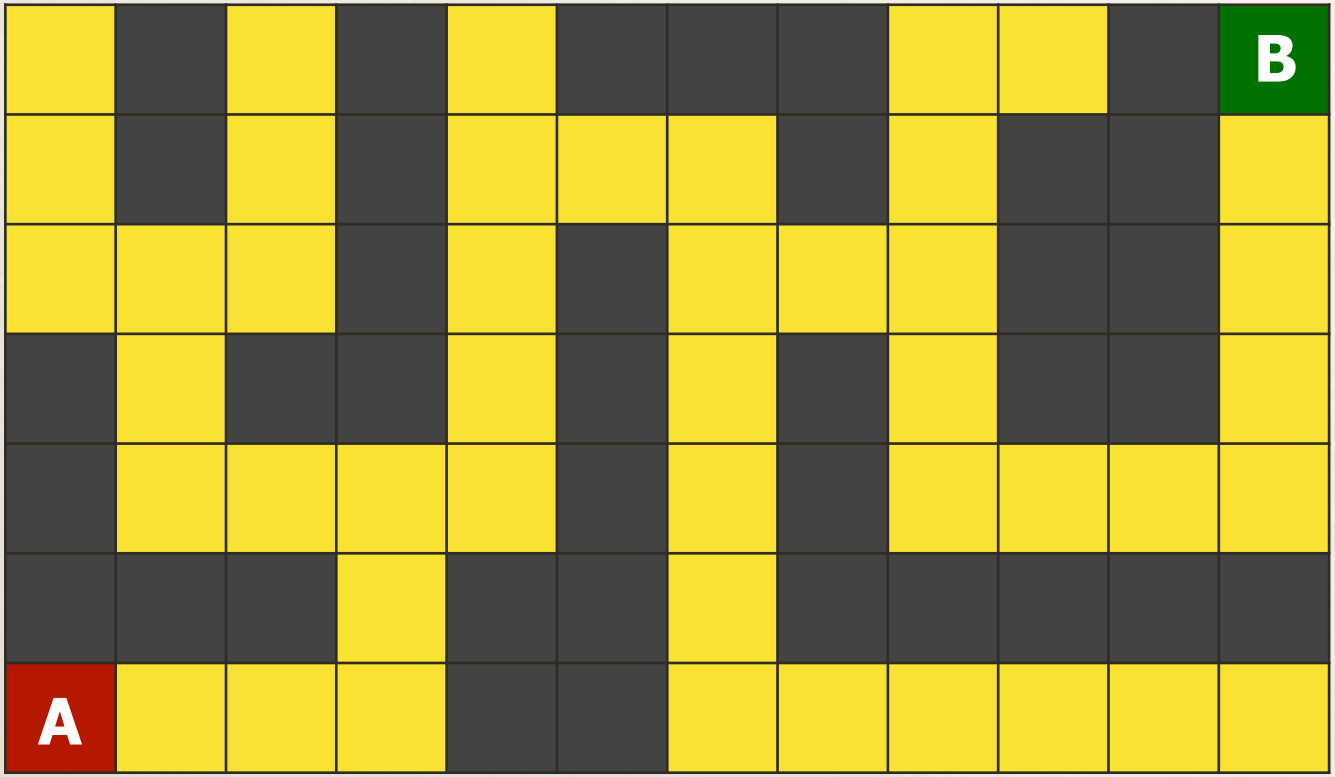
Breadth-First Search



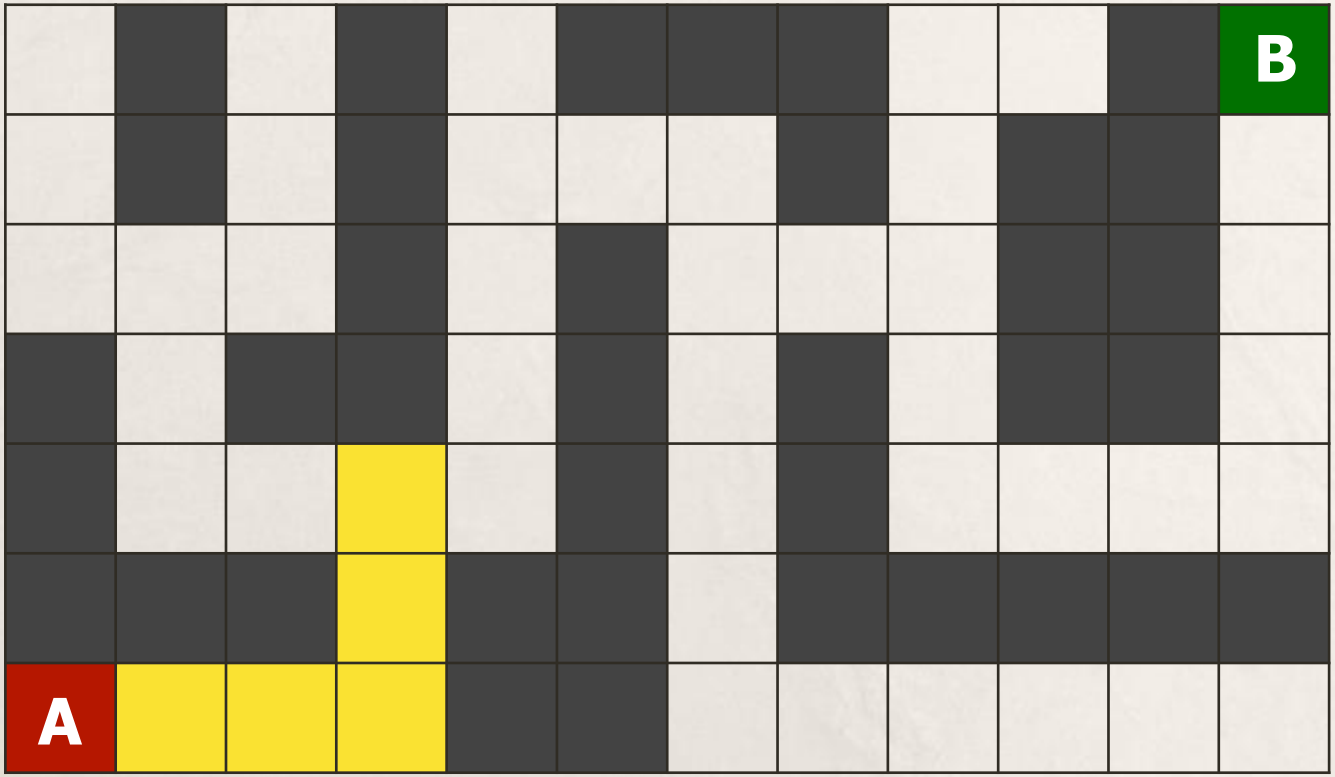
Breadth-First Search



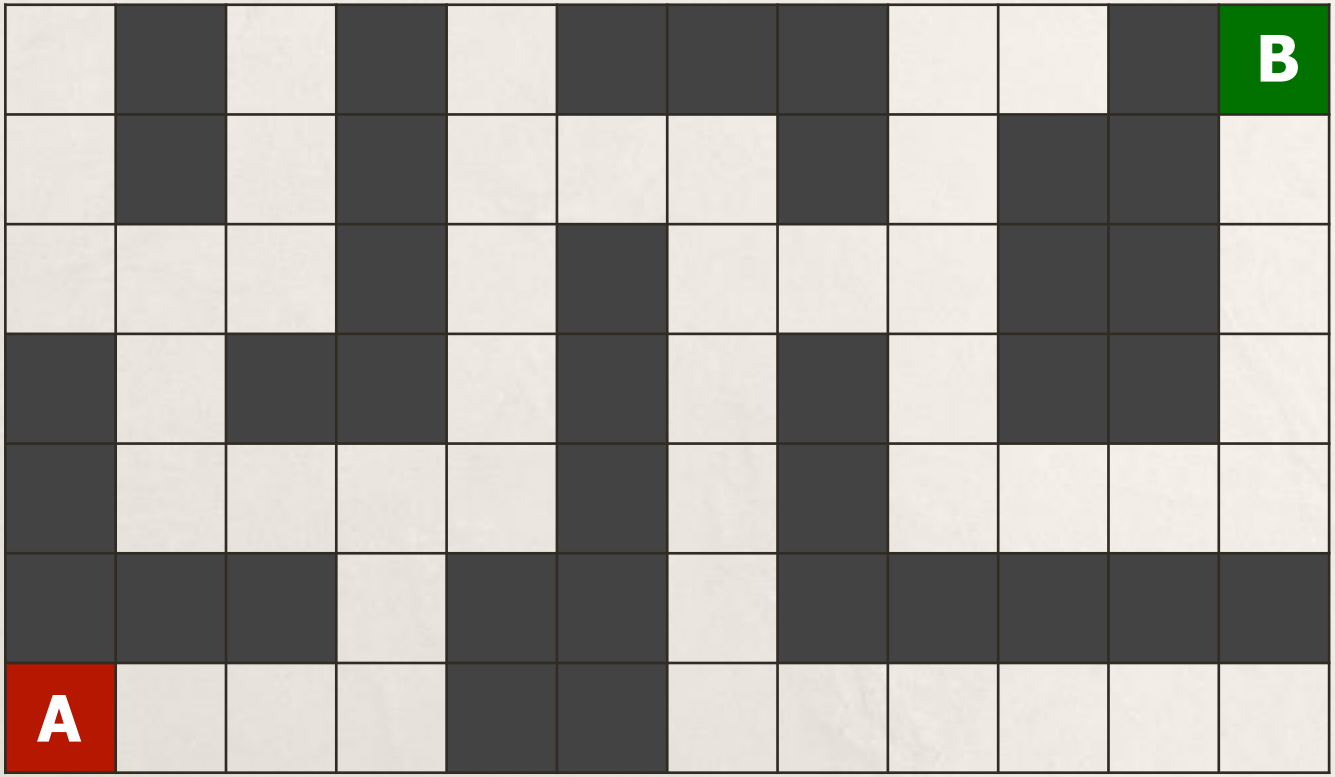
Breadth-First Search



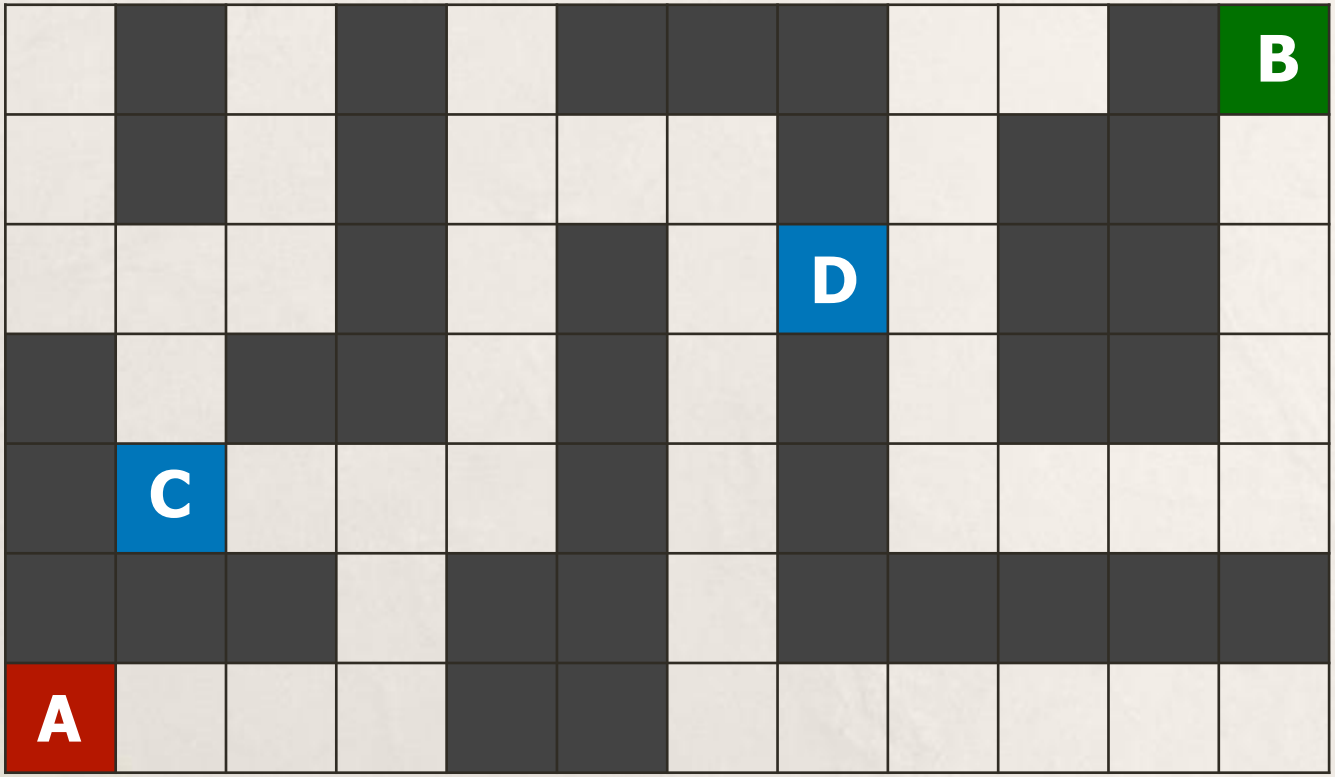
Breadth-First Search



Informing the search using heuristics

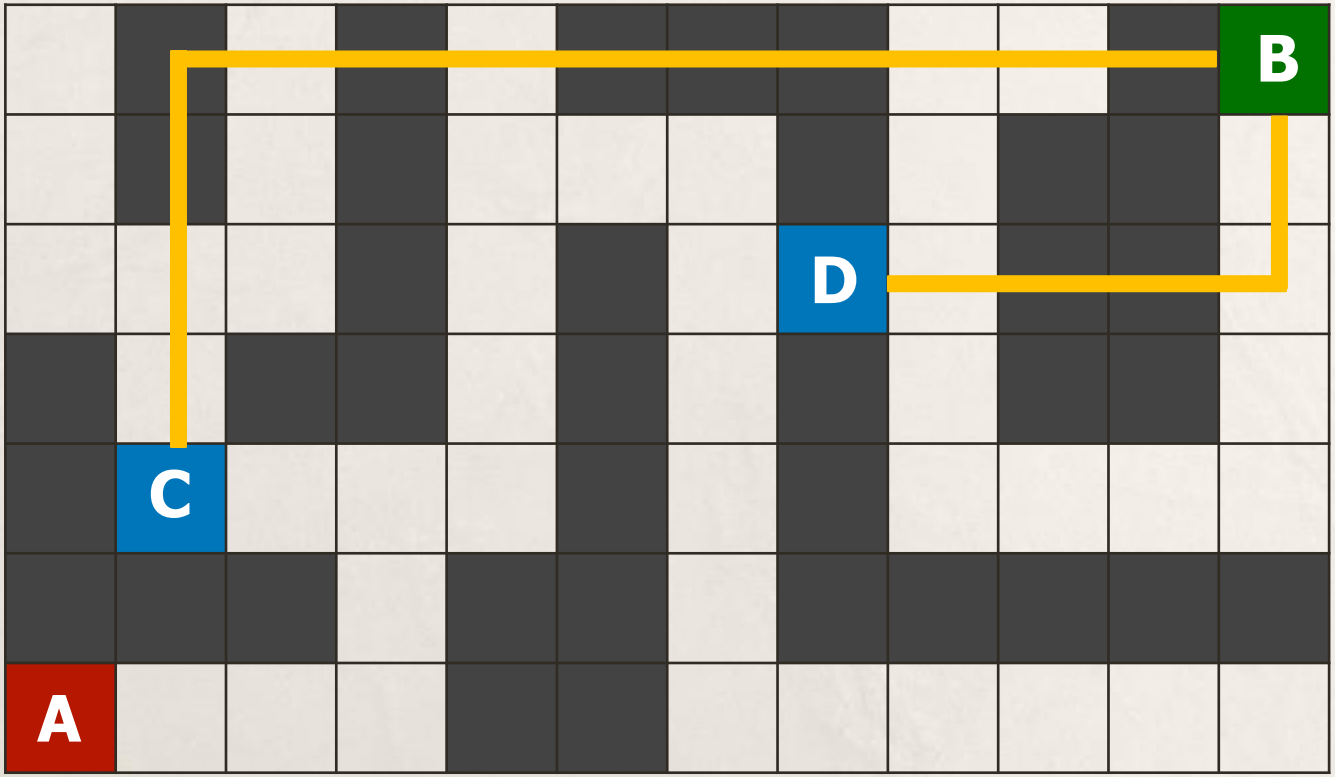


Informing the search using heuristics



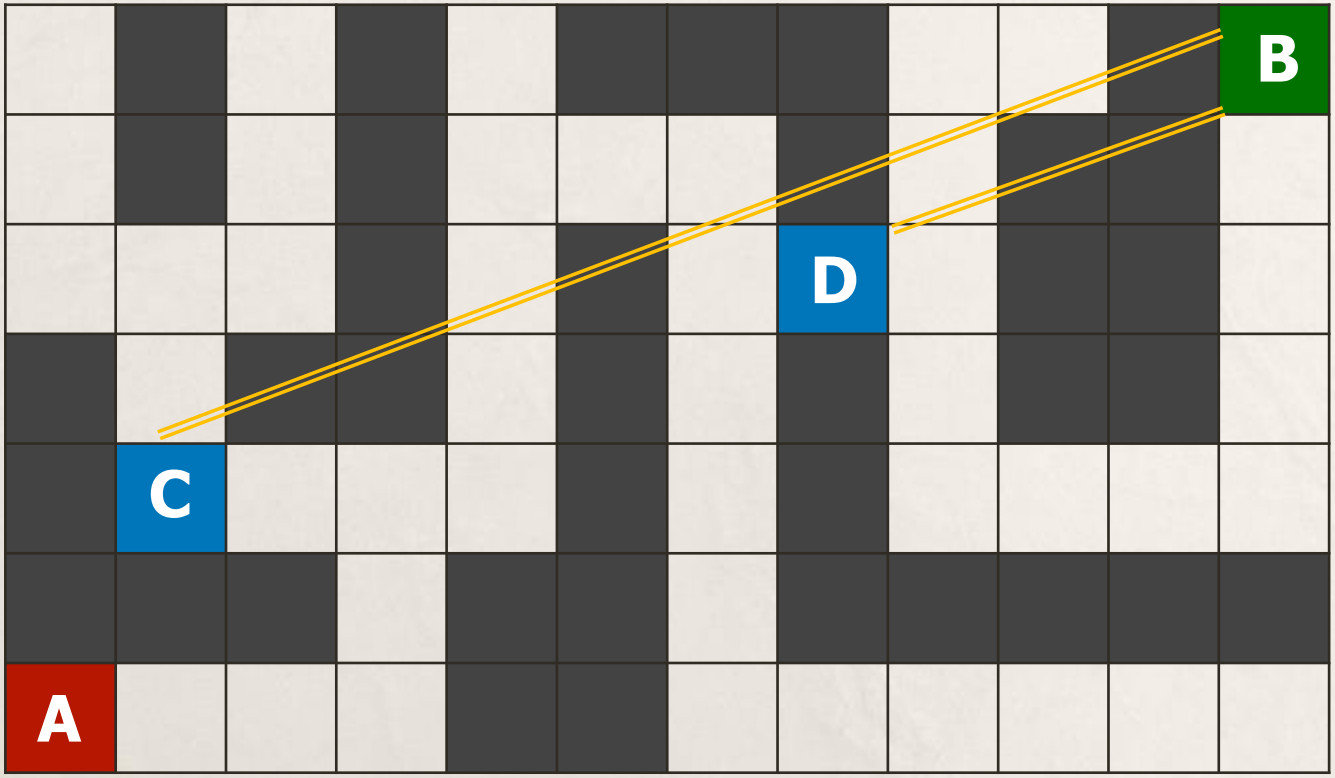
Informing the search using heuristics

- Manhattan distance



Informing the search using heuristics

- Straight-line distance



Providing hints for search using heuristics

- **Choosing the Fastest Checkout Line**
- **Question**: You walk into a supermarket and see multiple checkout lines. You want to check out quickly. Would you:
  - Count every item in each customer's cart to calculate exactly which line will move fastest?
  - Estimate which line might be faster by looking at the number of people, cart sizes, and how fast the cashier is working?
- **What factors help you make a quick but effective decision?**

Providing hints for search using heuristics

- **Choosing the Fastest Checkout Line**
- **Question**: You walk into a supermarket and see multiple checkout lines. You want to check out quickly. Would you:
  - Count every item in each customer's cart to calculate exactly which line will move fastest?
  - Estimate which line might be faster by looking at the number of people, cart sizes, and how fast the cashier is working?
- **What factors help you make a quick but effective decision?**
- **Expected Heuristic**: Choose the shortest line with the fewest items and the fastest-moving cashier.

Providing hints for search using heuristics

- **Guessing a Word in a Crossword Puzzle**
- **Question**: You are solving a crossword puzzle, and a missing word looks like this: **“S_N_”**. Would you:
  - Go through the dictionary, checking every possible word
  - Think of common words that fit the pattern, like **"SAND" or "SONG"**?
- **How does your brain quickly filter out unlikely words?**

Providing hints for search using heuristics

- **Guessing a Word in a Crossword Puzzle**
- **Question**: You are solving a crossword puzzle, and a missing word looks like this: **“S_N_”**. Would you:
  - Go through the dictionary, checking every possible word
  - Think of common words that fit the pattern, like **"SAND" or "SONG"**?
- **How does your brain quickly filter out unlikely words?**
- **Expected Heuristic**: Use letter patterns and context to prioritize likely words.

Informed search

- Search strategy that uses problem-specific knowledge (heuristics) to find solutions more efficiently

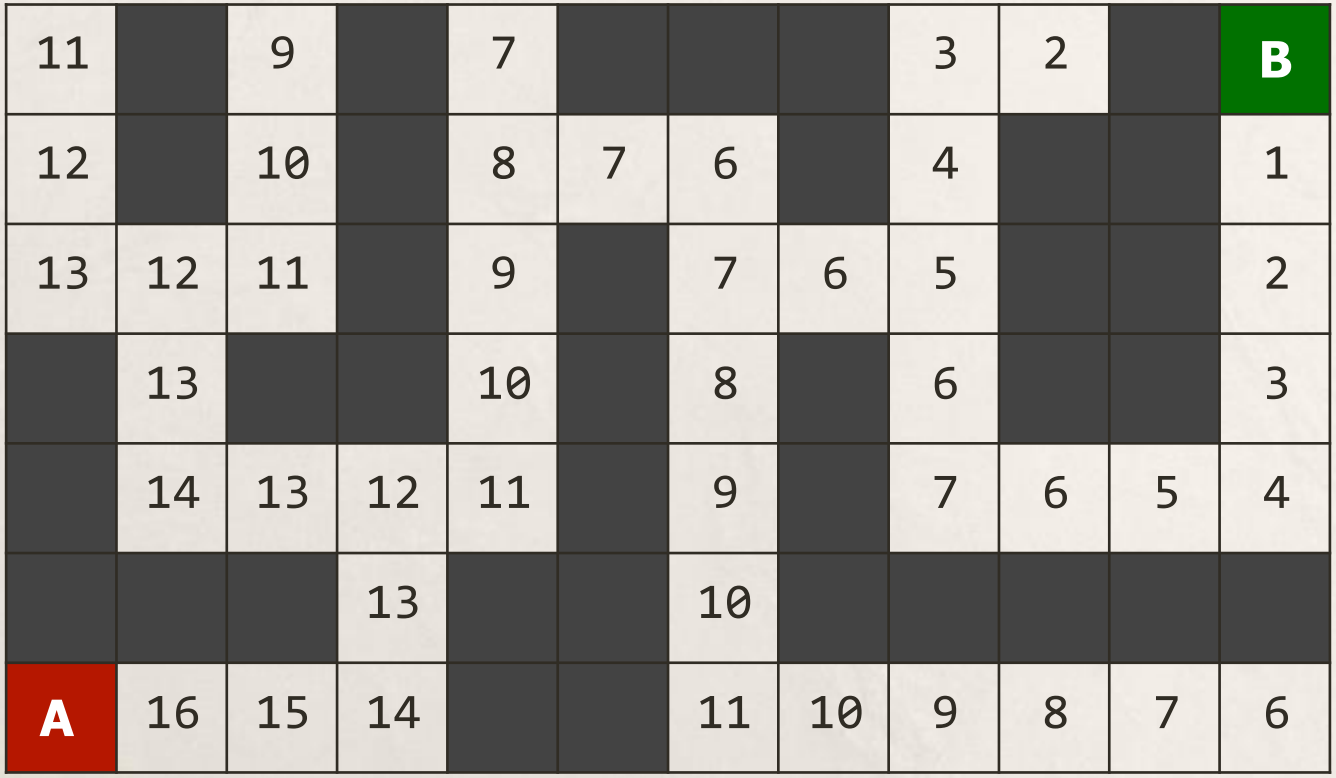
Greedy best-first search

- Let $g(n)$ represent a simple function that returns a path cost for a node n
- Dijkstra's algorithm uses the evaluation function $f(n) = g(n)$ in the priority queue for the search frontier to choose the node to expand.
  - UCS is Dijkstra's algorithm for a single goal
- Potential problem: it may be cheap to get to the candidate node's state but expensive to get from that state to the goal!
- Idea: estimate the cost of the cheapest path from the candidate node's state to a goal state and use this to modify the value returned by $f(n)$
- Greedy best-first search algorithm expands the node that is closest to the goal state, as estimated by a heuristic function $h(n)$
- It ignores how costly it was to get to n and *greedily* expands the node that is gauged to be closest to a goal according to $h(n)$
- Greedy best-first search: $f(n) = h(n)$

Greedy best-first search

```
1   Function GBFS(problem, h)
2     frontier = PQ{f=h, Node(problem.initial_state)}
3     while (true)
4       if (frontier.empty) return fail
5       node = pop_min_f(frontier)
6       if (node.state is goal) return solution
7       for c in Expand(node, problem)
8           frontier.add(c)
```

Greedy Best-First Search



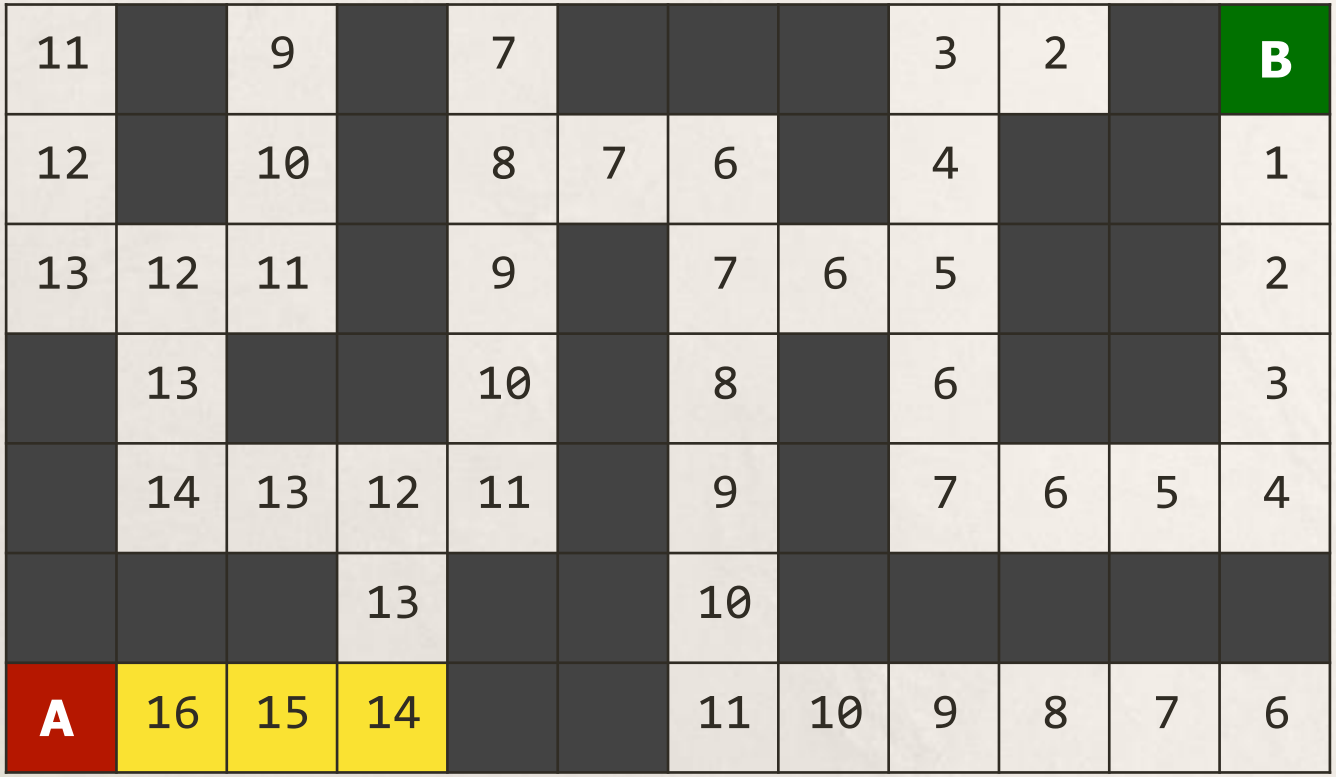
Greedy Best-First Search

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

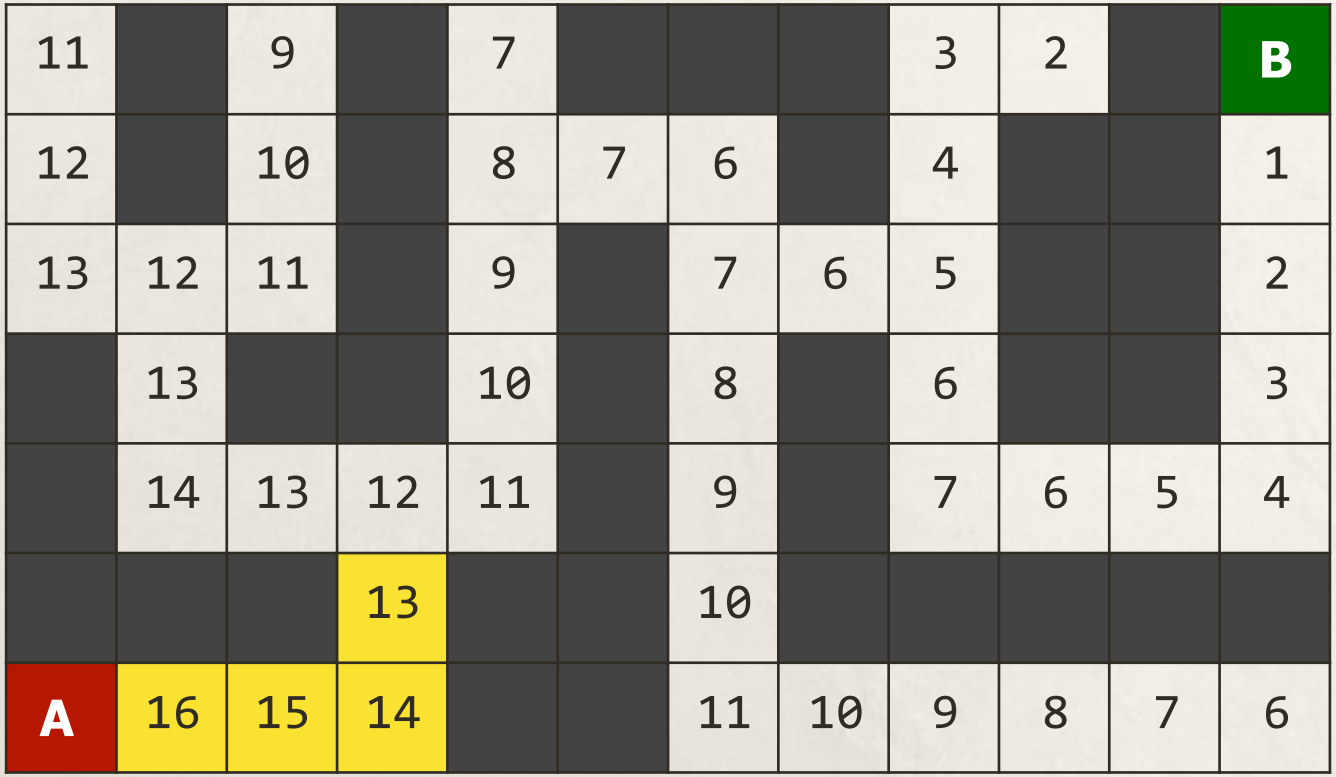
Greedy Best-First Search

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

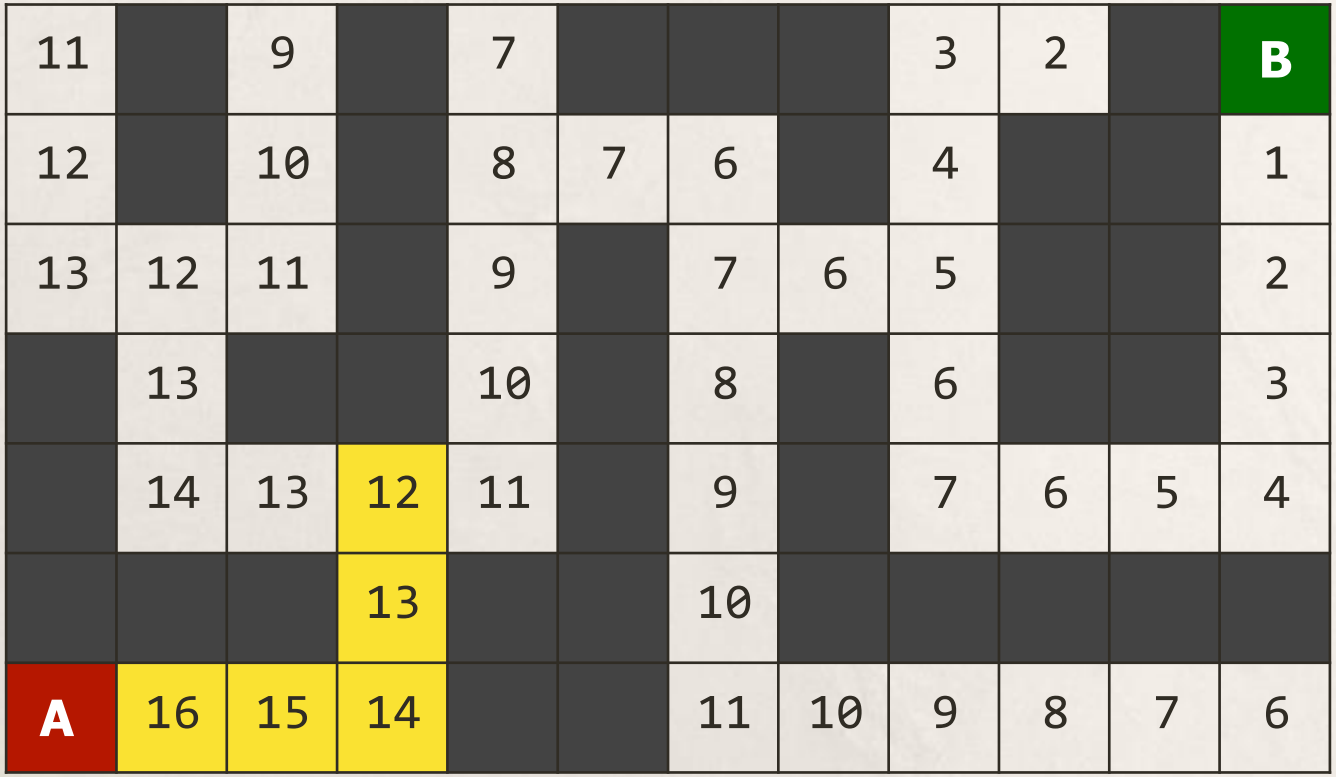
Greedy Best-First Search



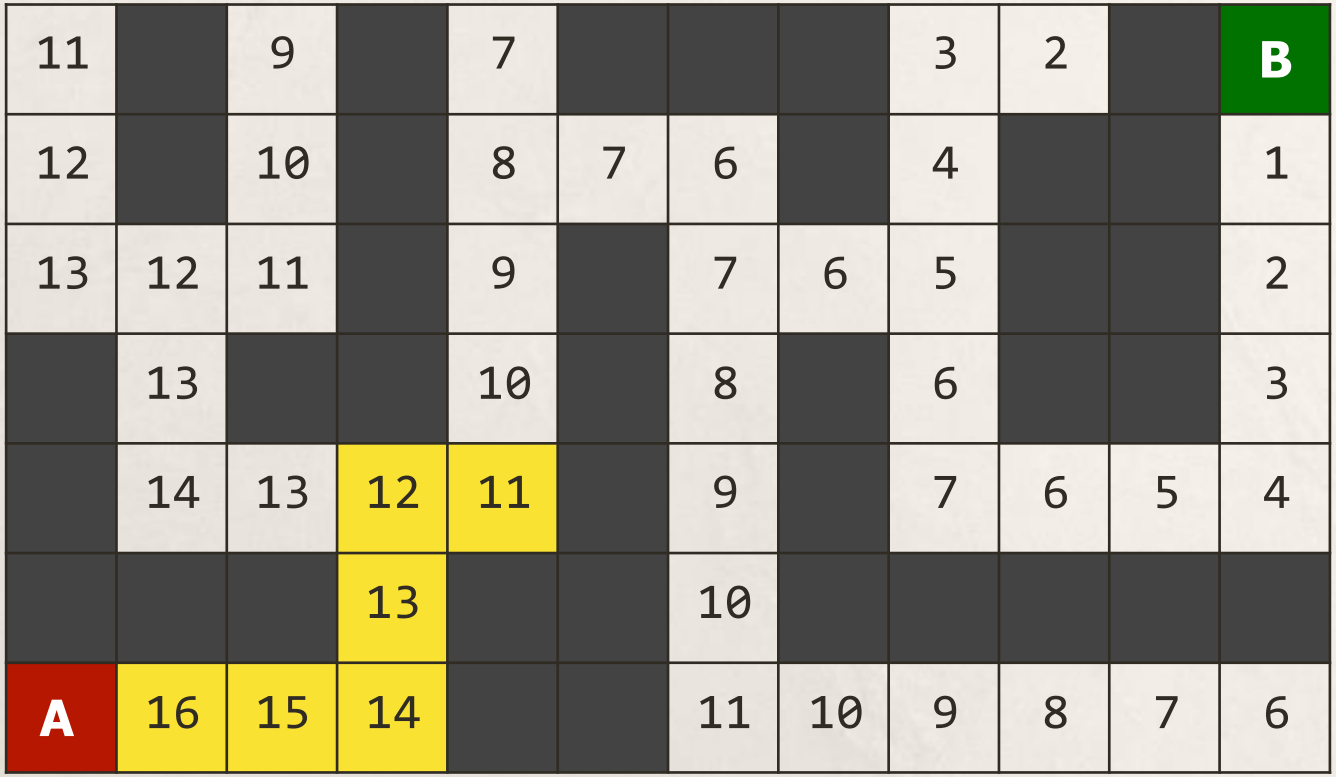
Greedy Best-First Search



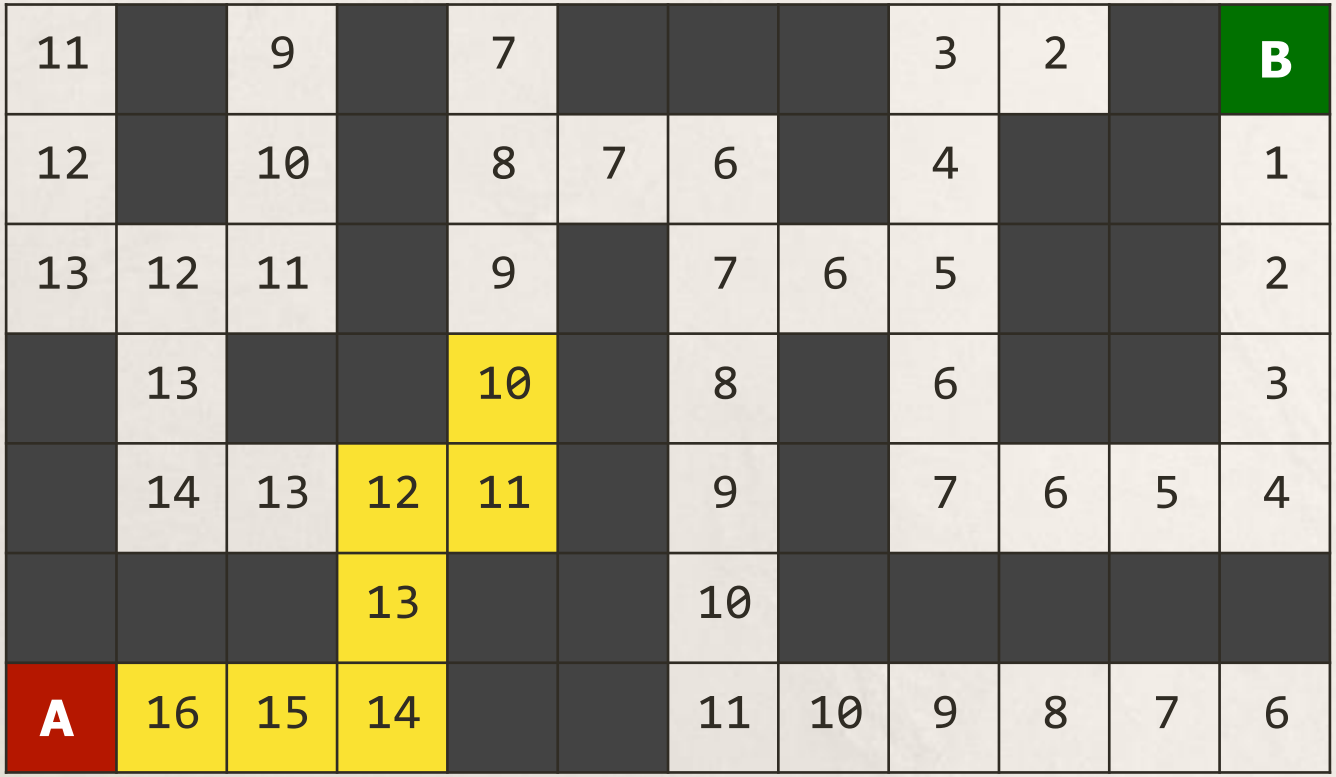
Greedy Best-First Search



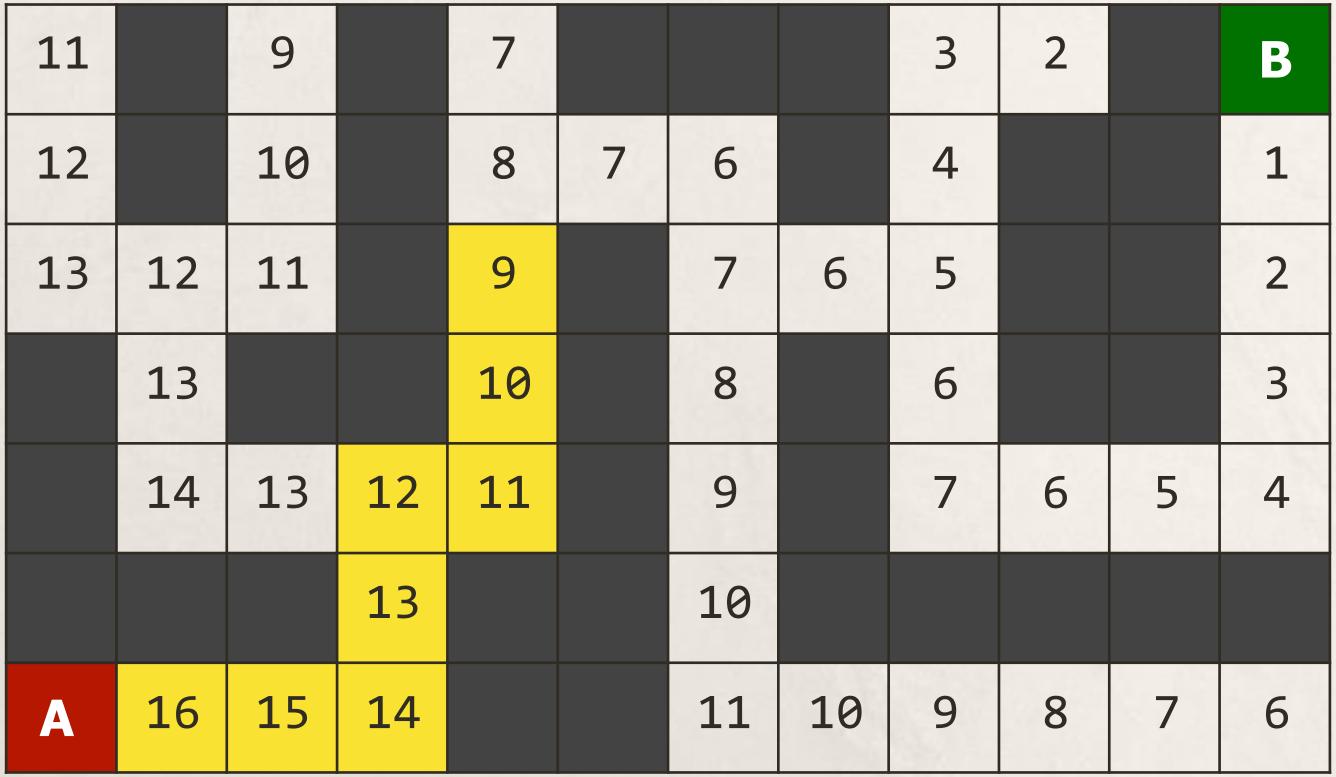
Greedy Best-First Search



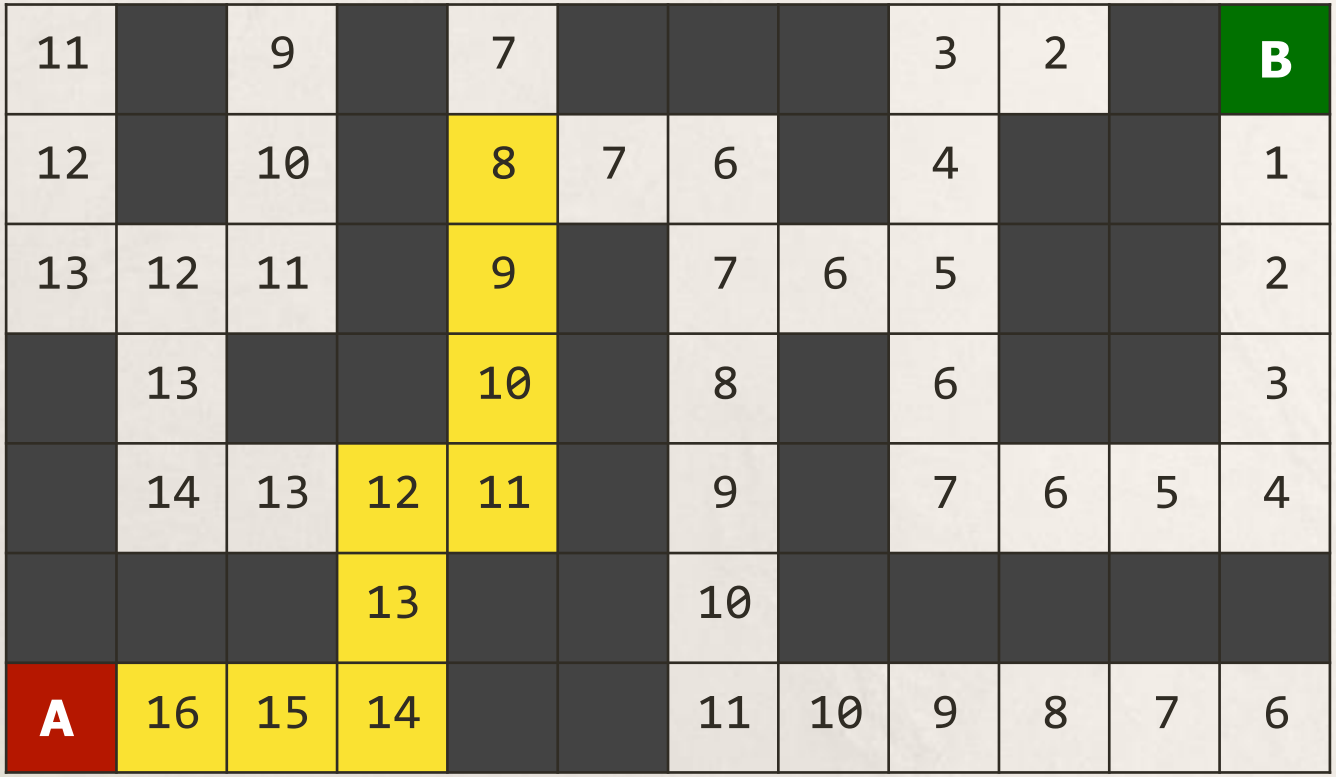
Greedy Best-First Search



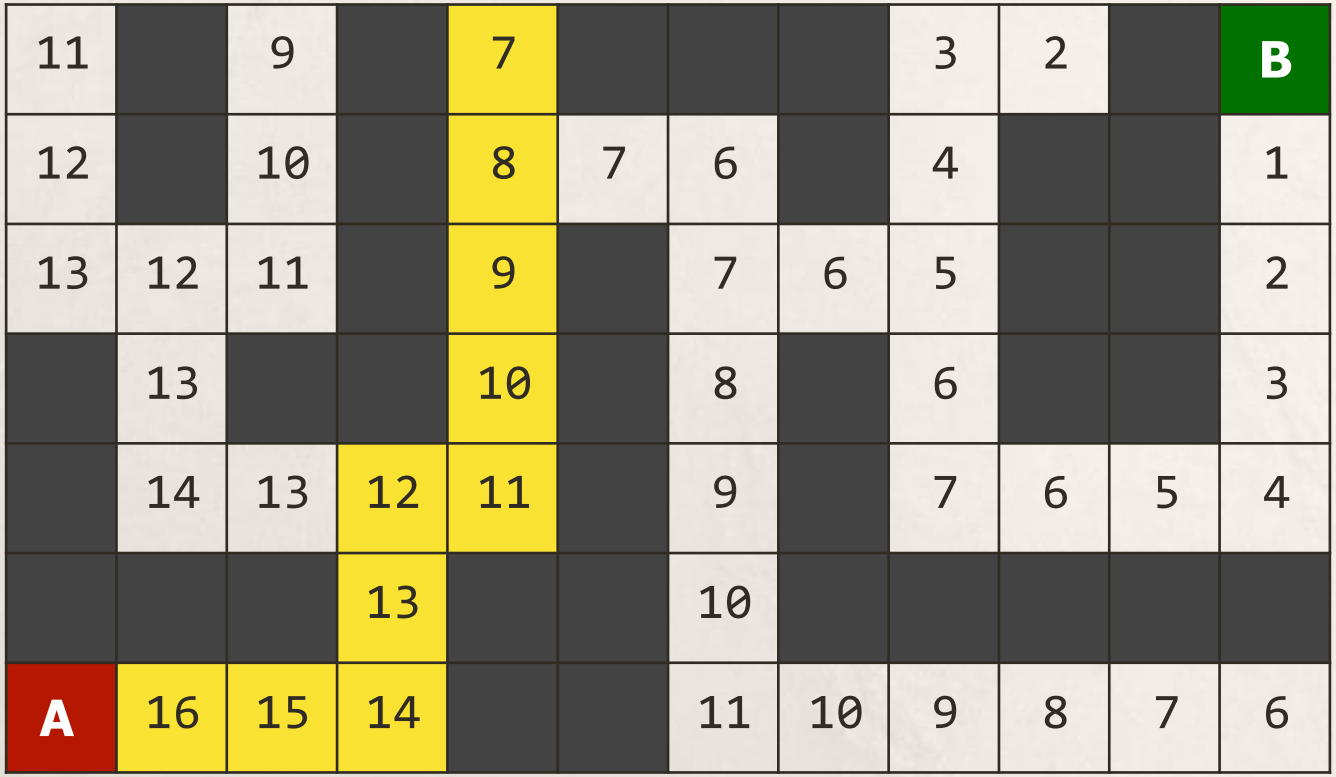
Greedy Best-First Search



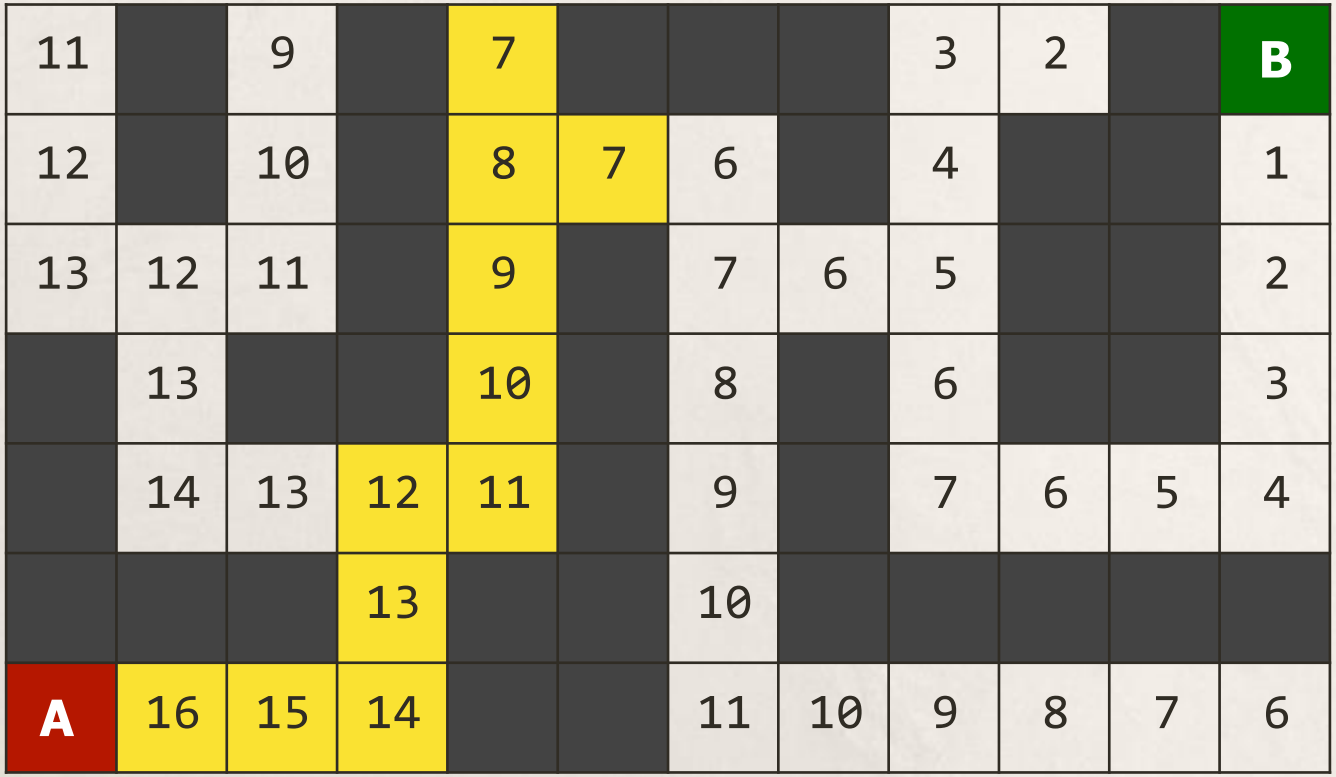
Greedy Best-First Search



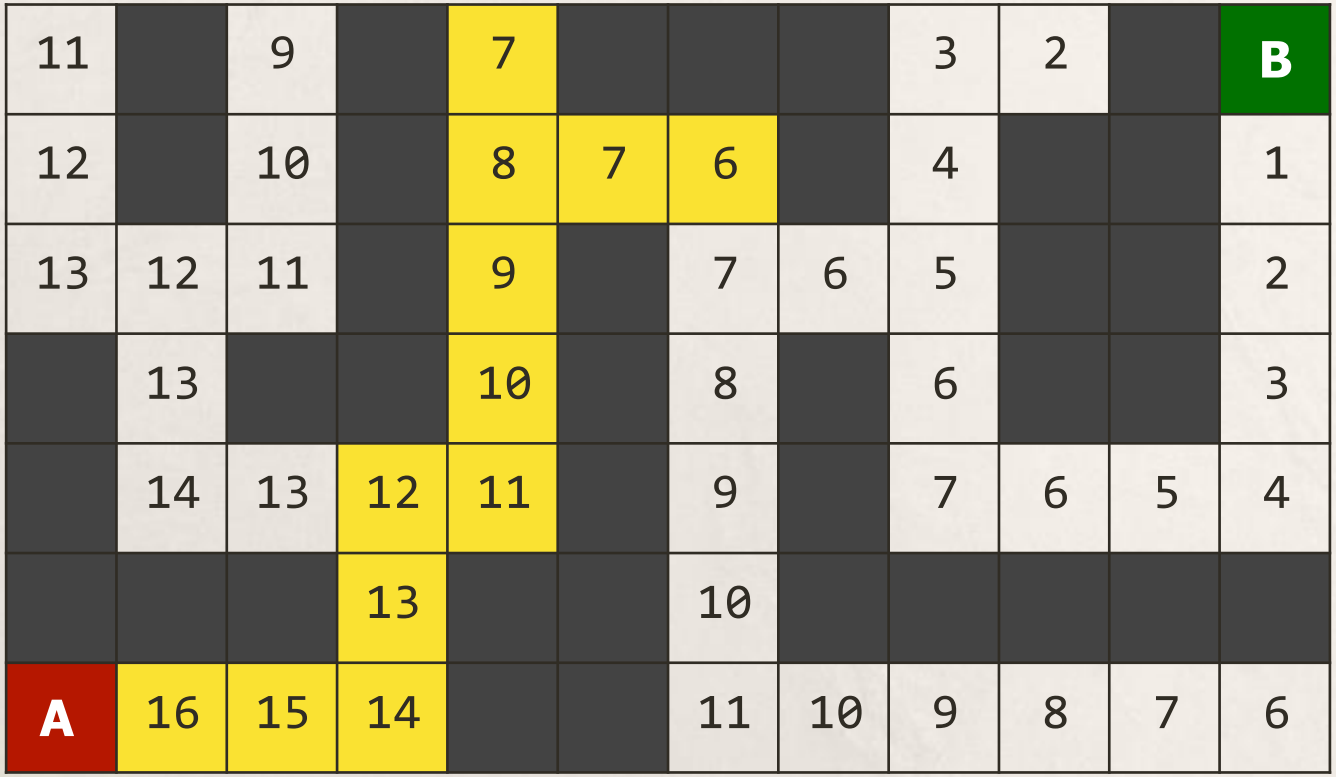
Greedy Best-First Search



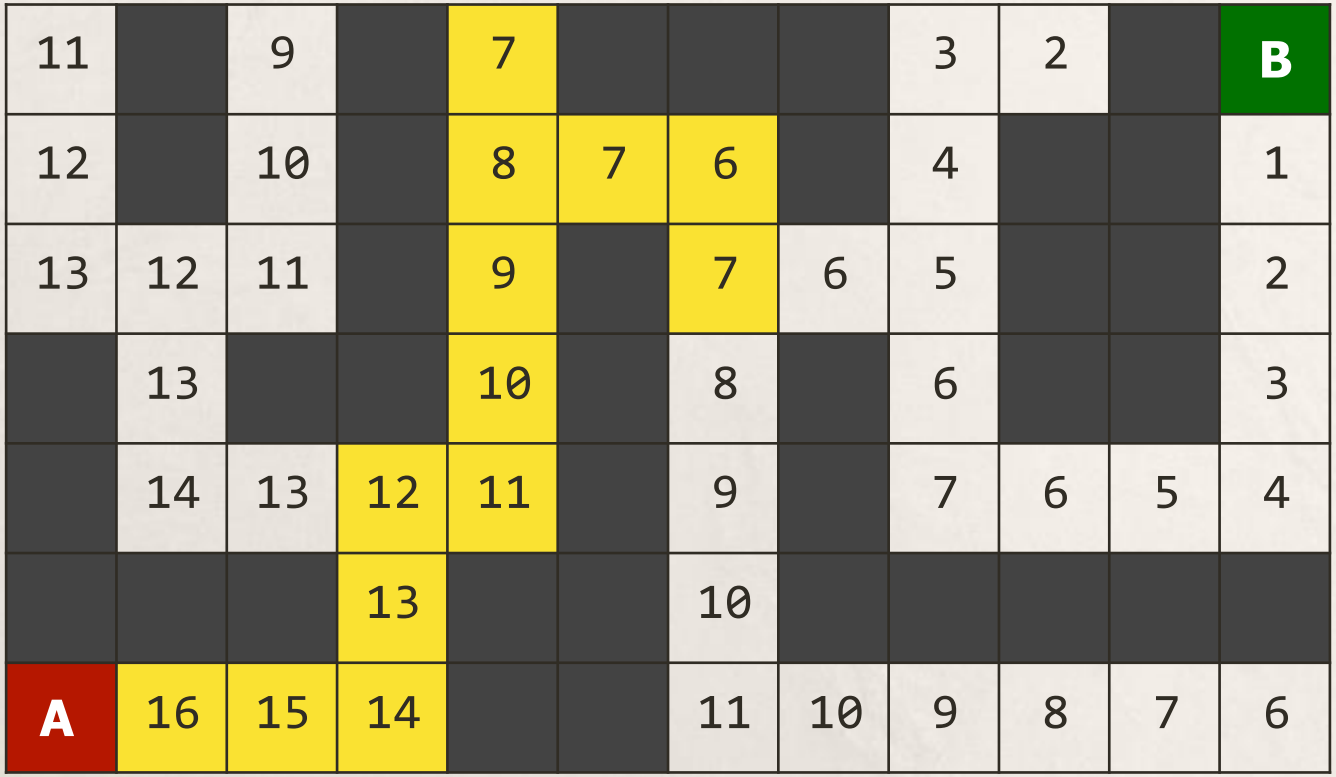
Greedy Best-First Search



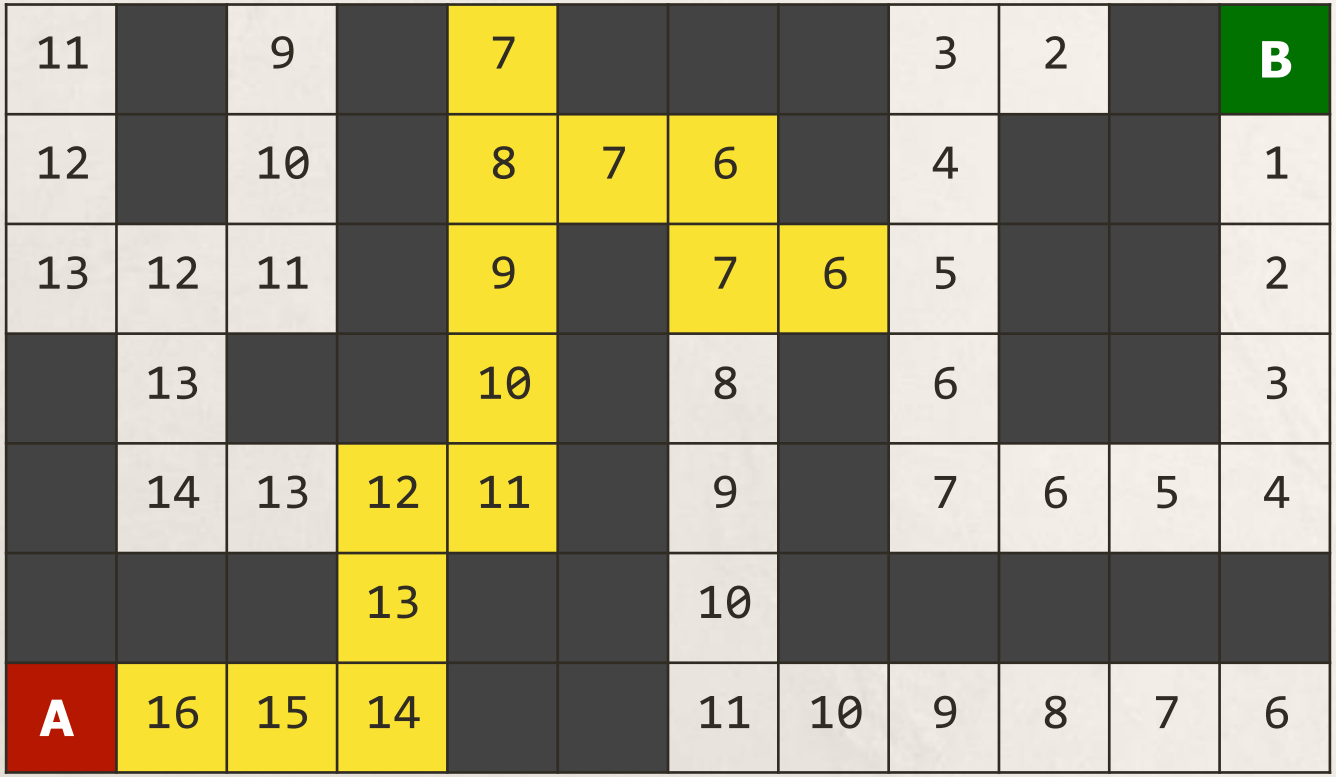
Greedy Best-First Search



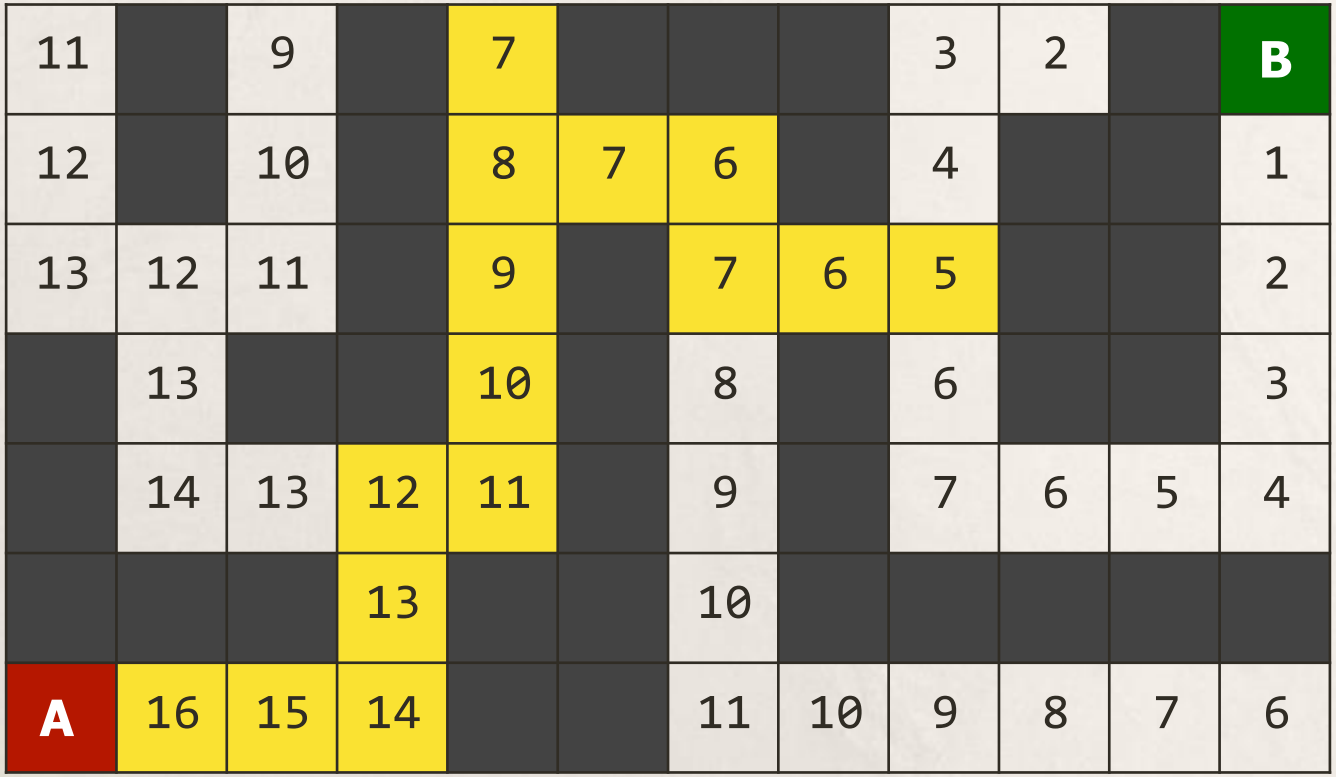
Greedy Best-First Search



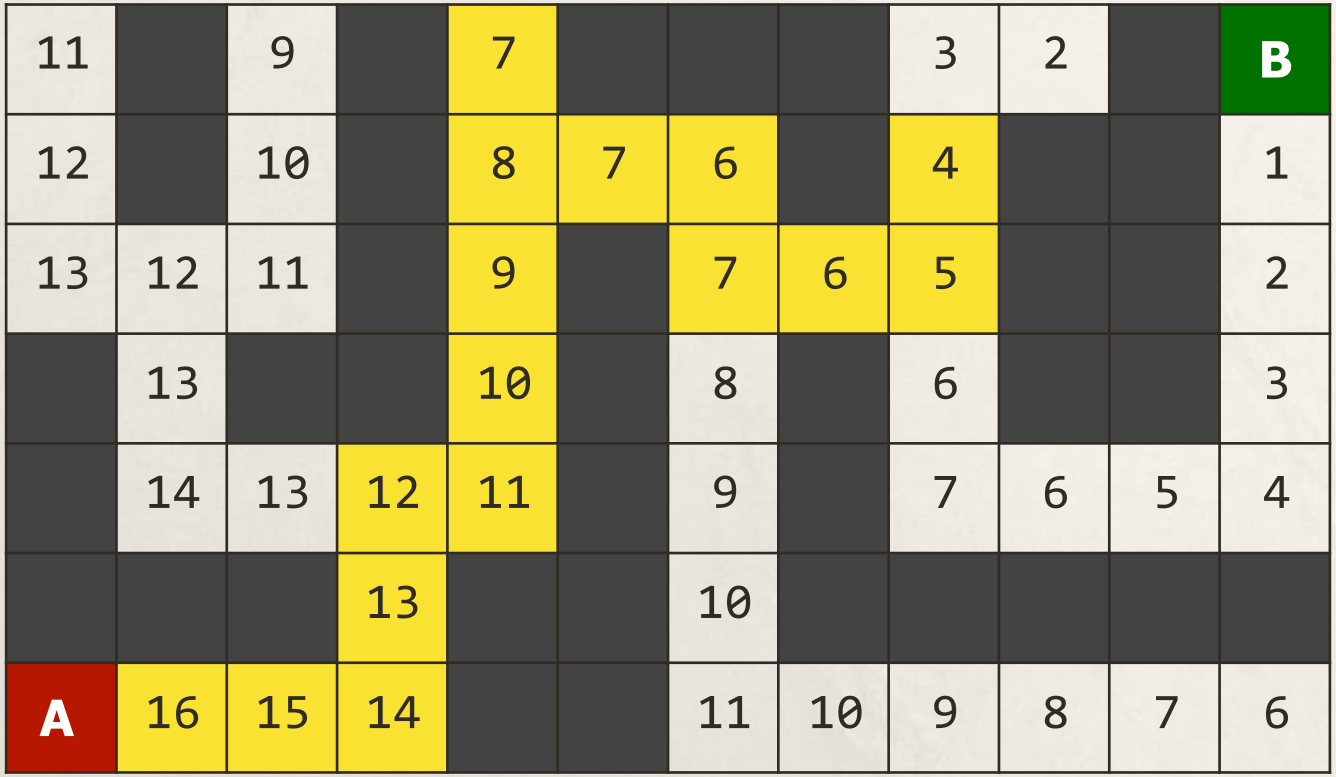
Greedy Best-First Search



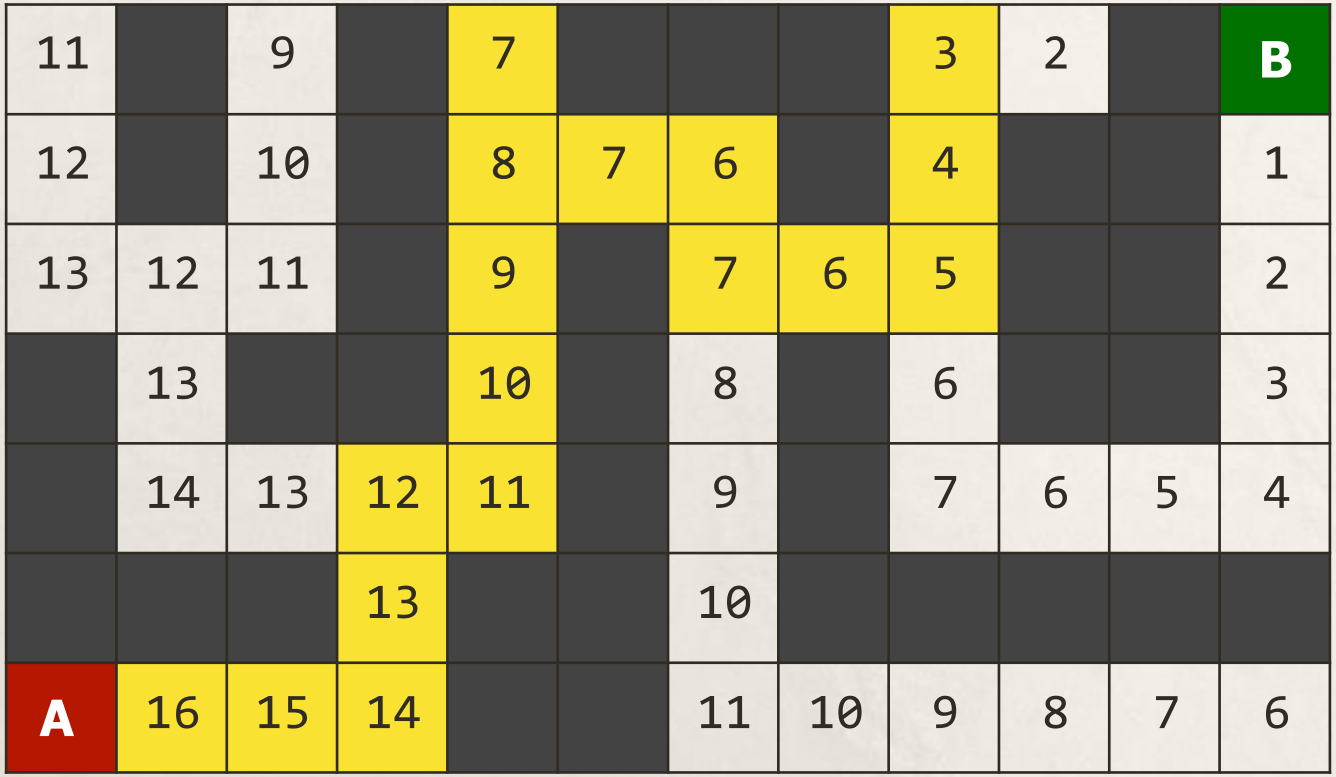
Greedy Best-First Search



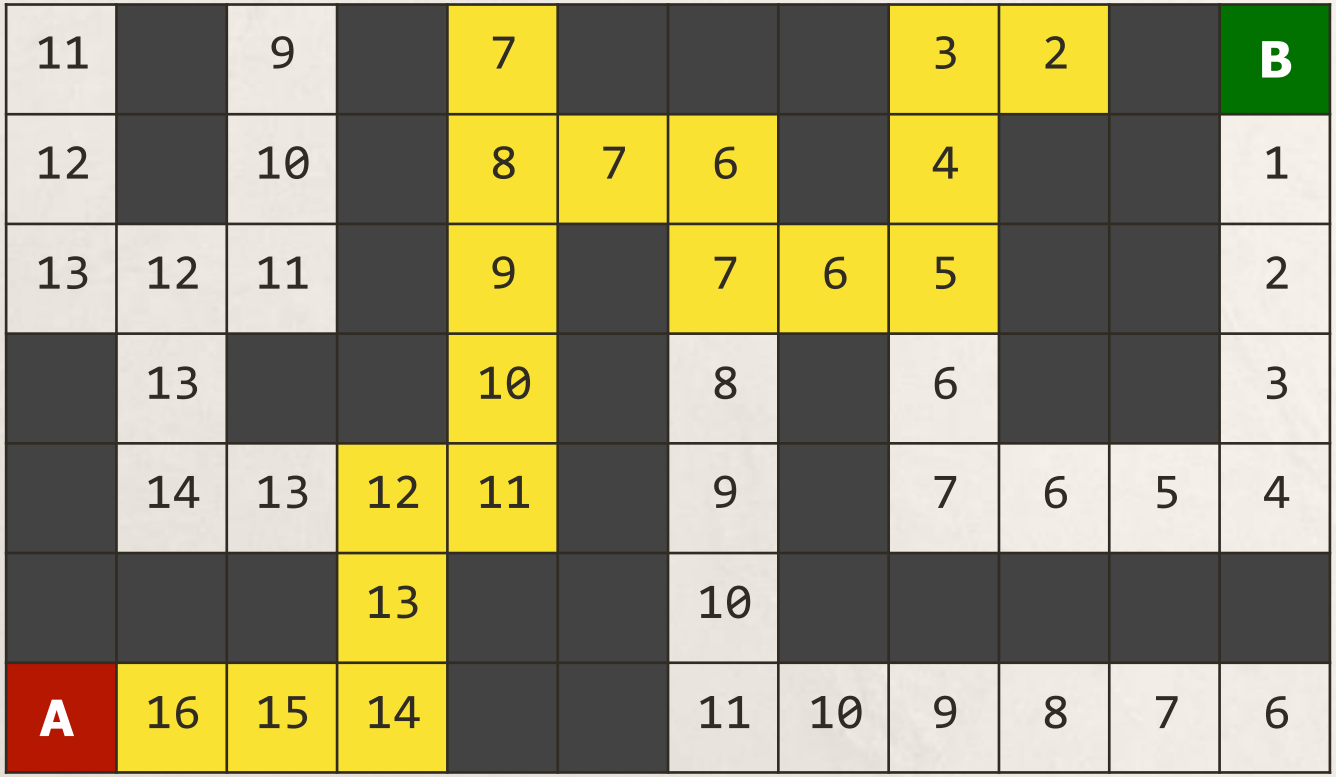
Greedy Best-First Search



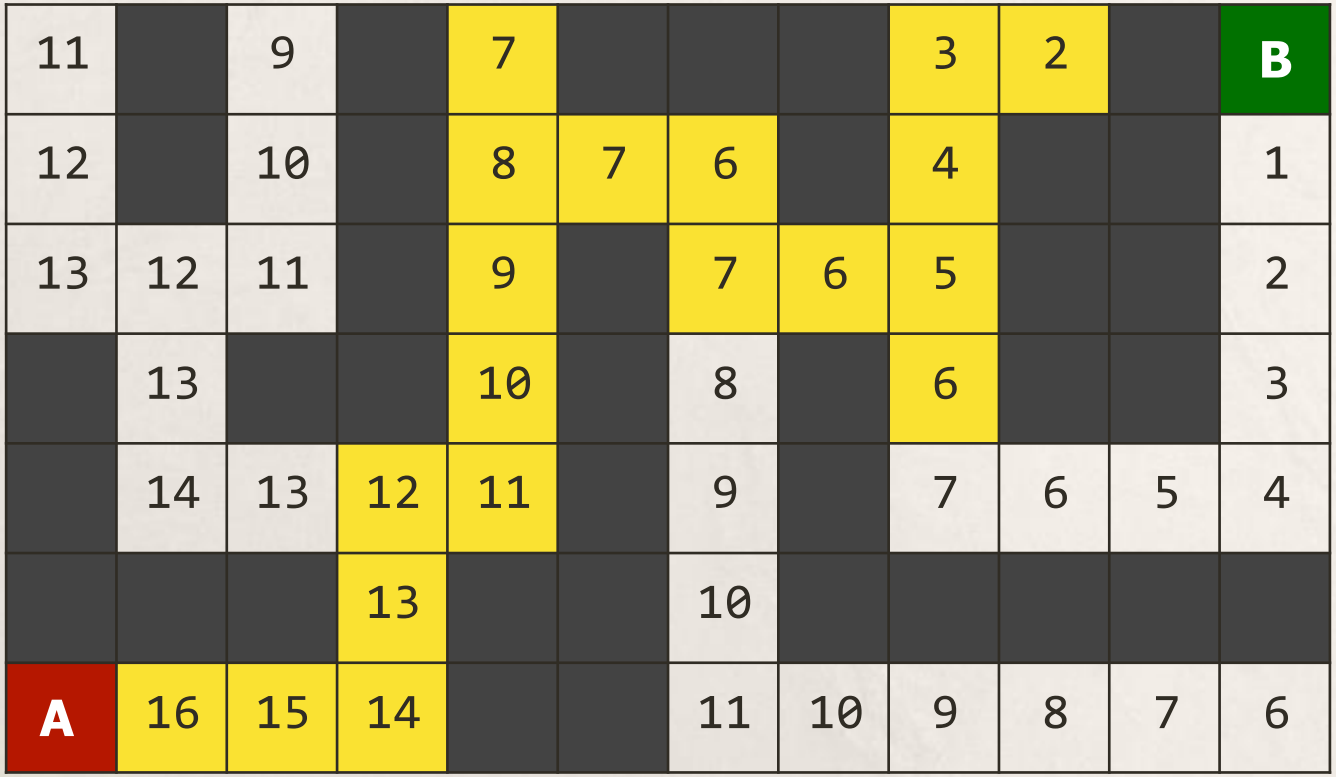
Greedy Best-First Search



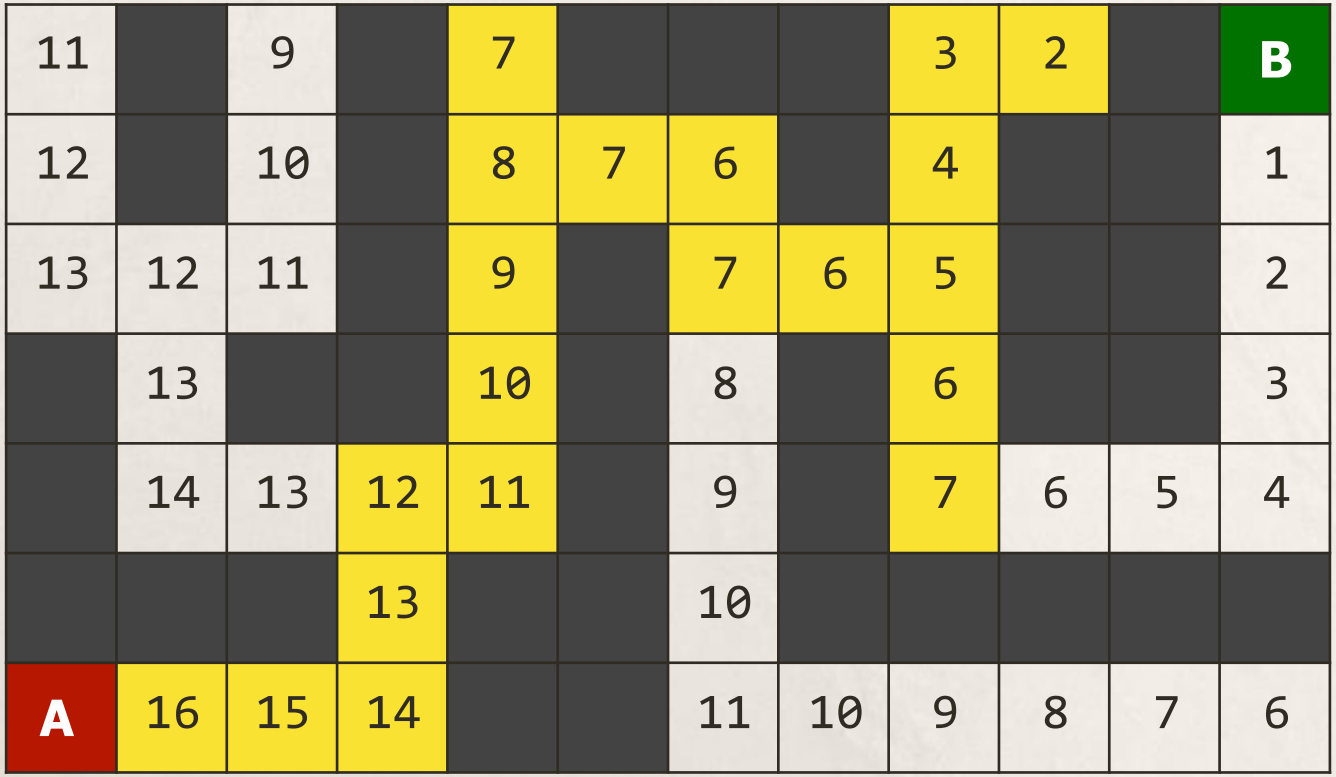
Greedy Best-First Search



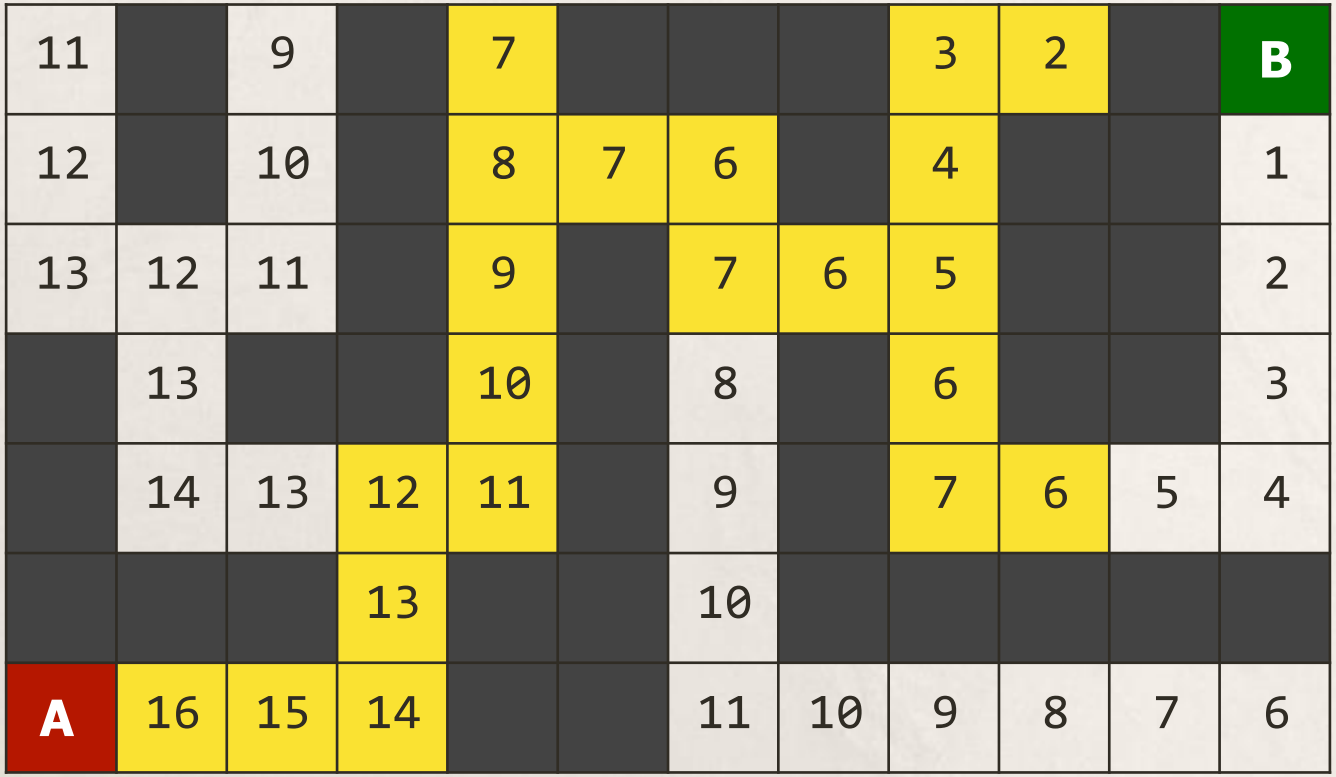
Greedy Best-First Search



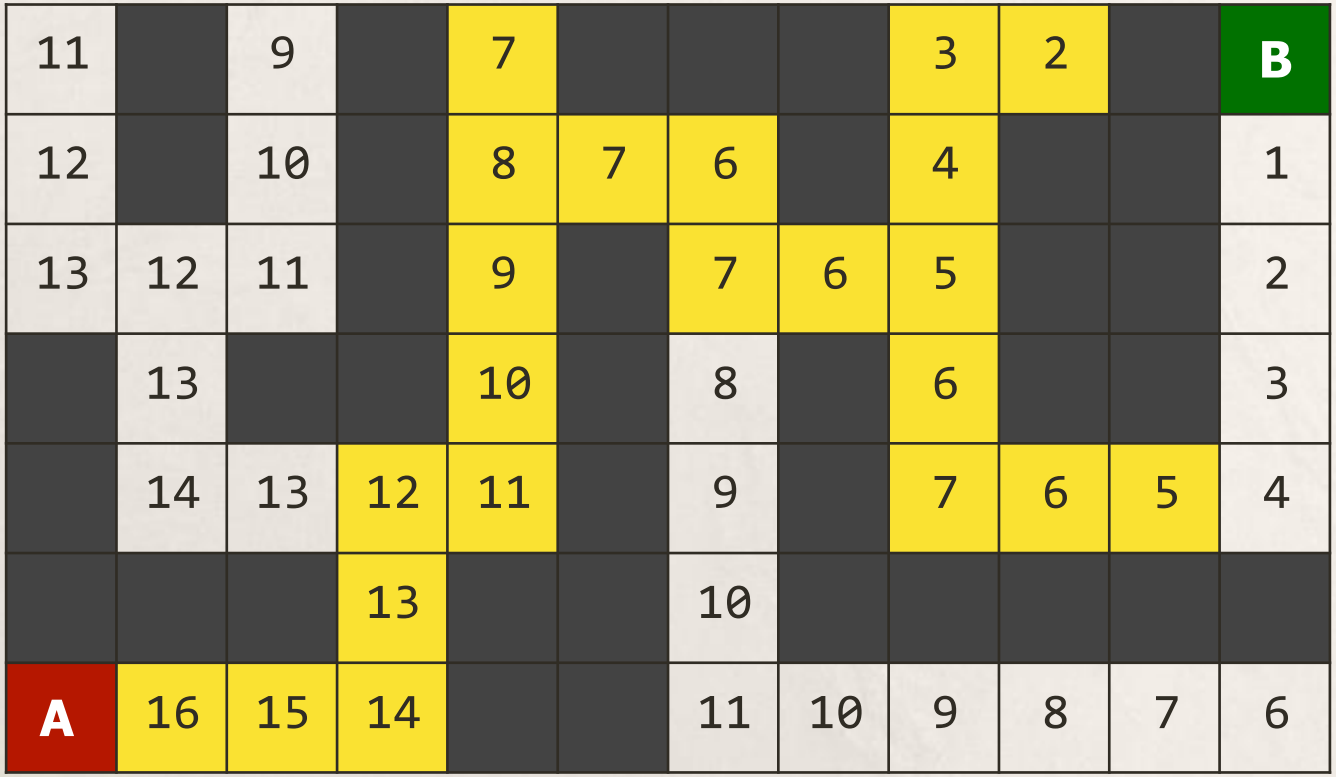
Greedy Best-First Search



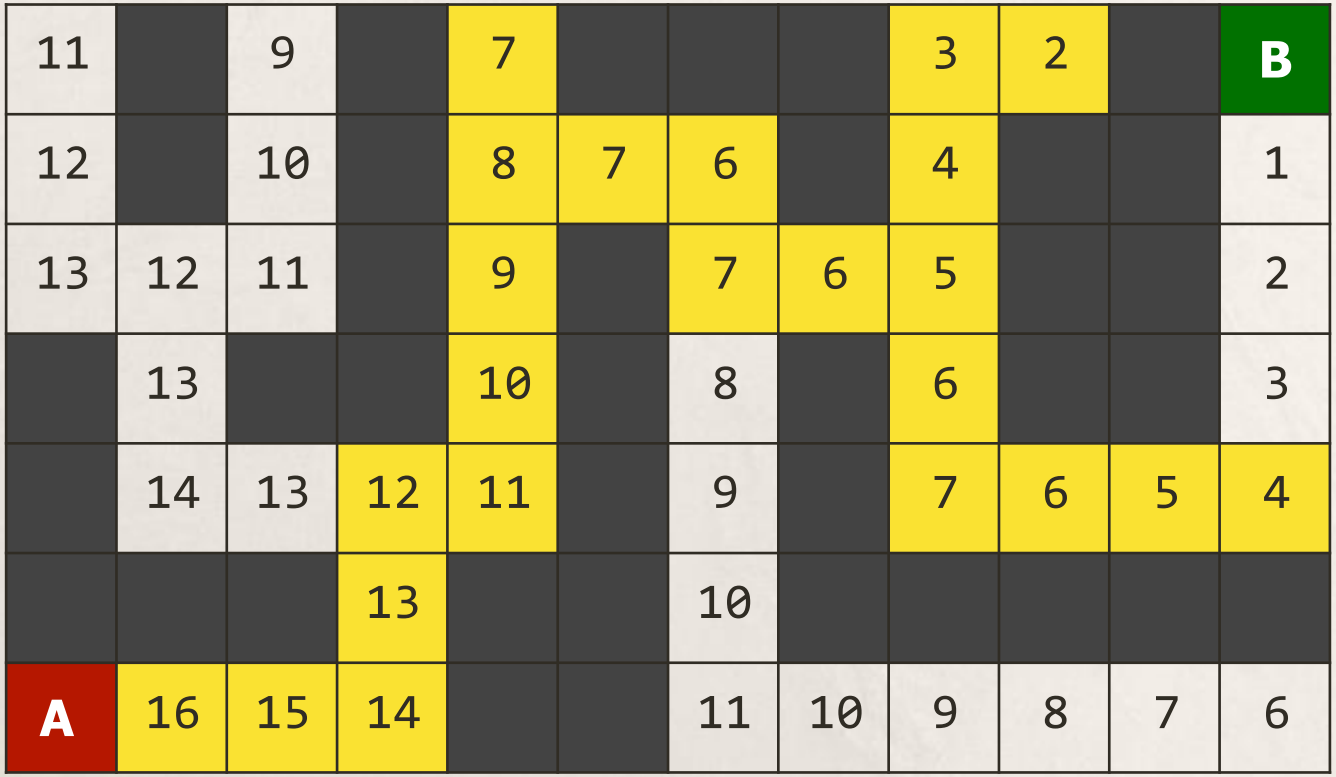
Greedy Best-First Search



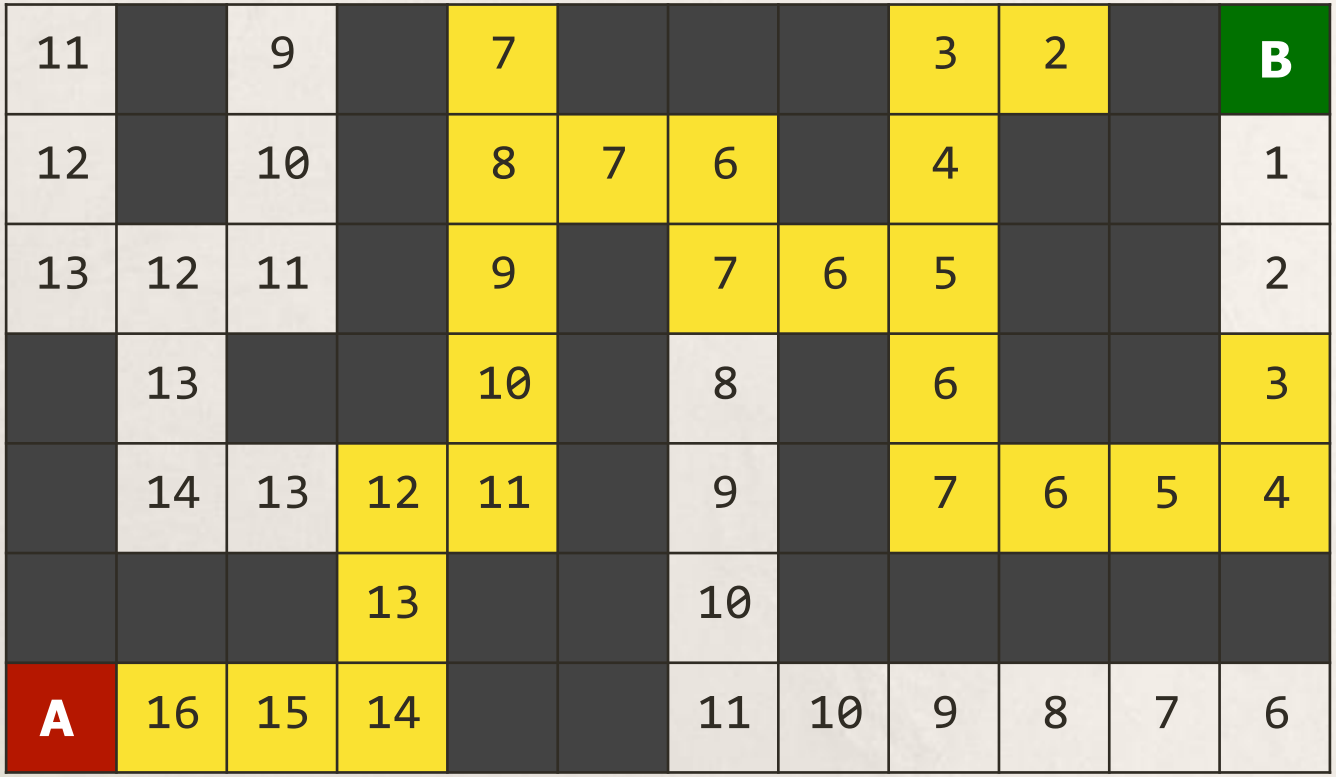
Greedy Best-First Search



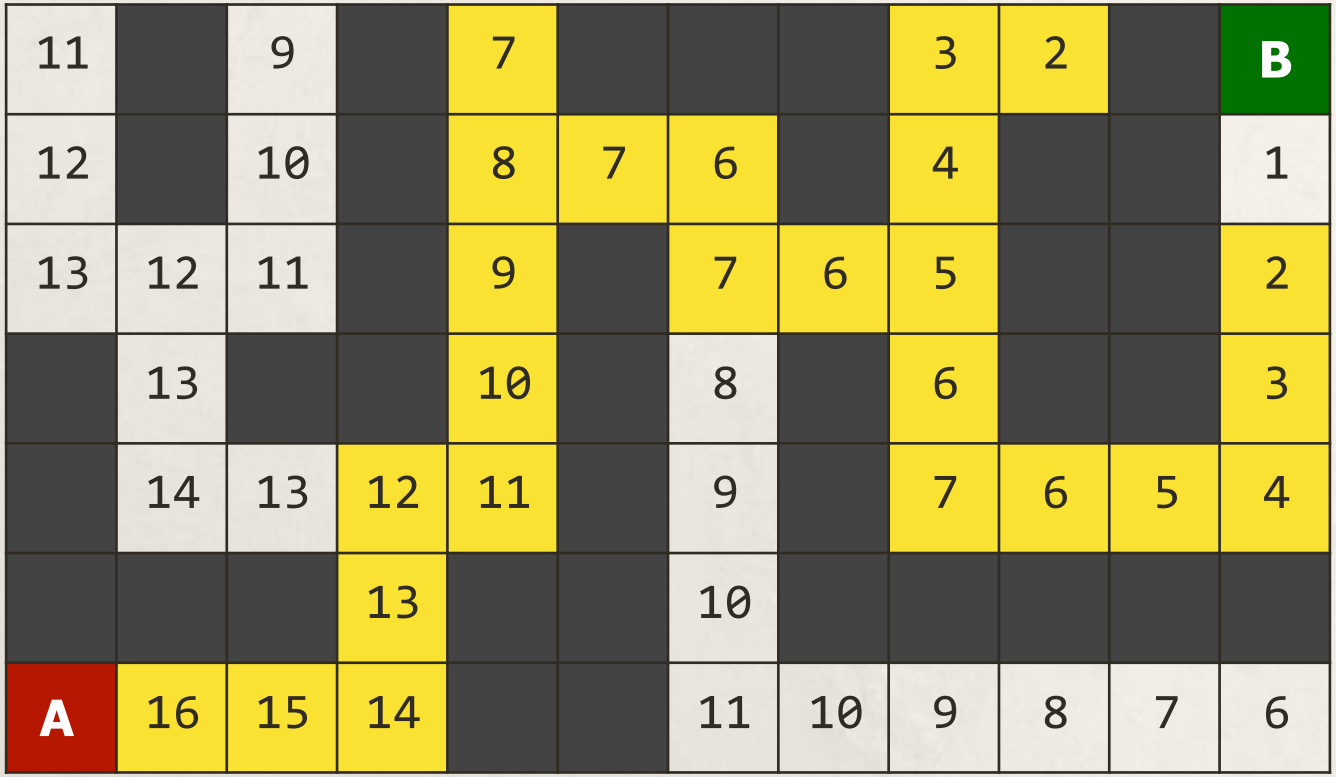
Greedy Best-First Search



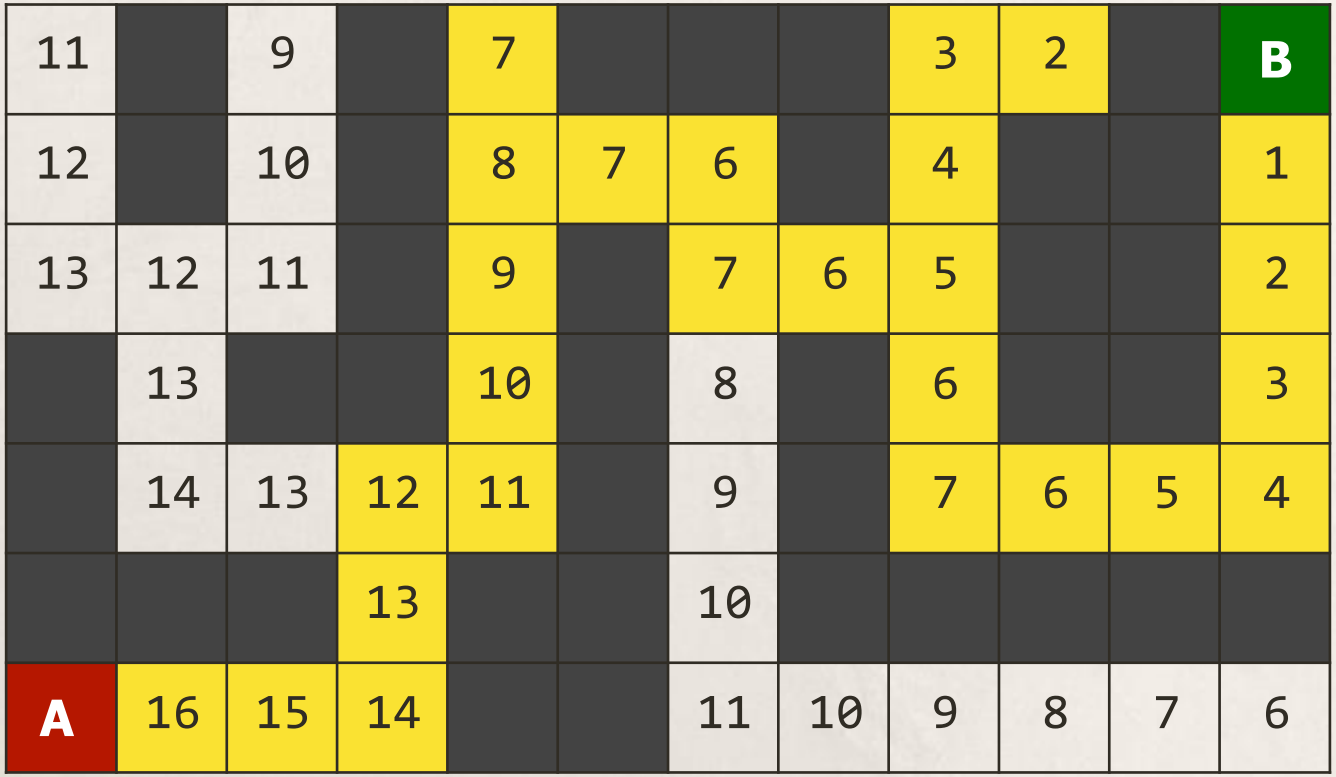
Greedy Best-First Search



Greedy Best-First Search



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

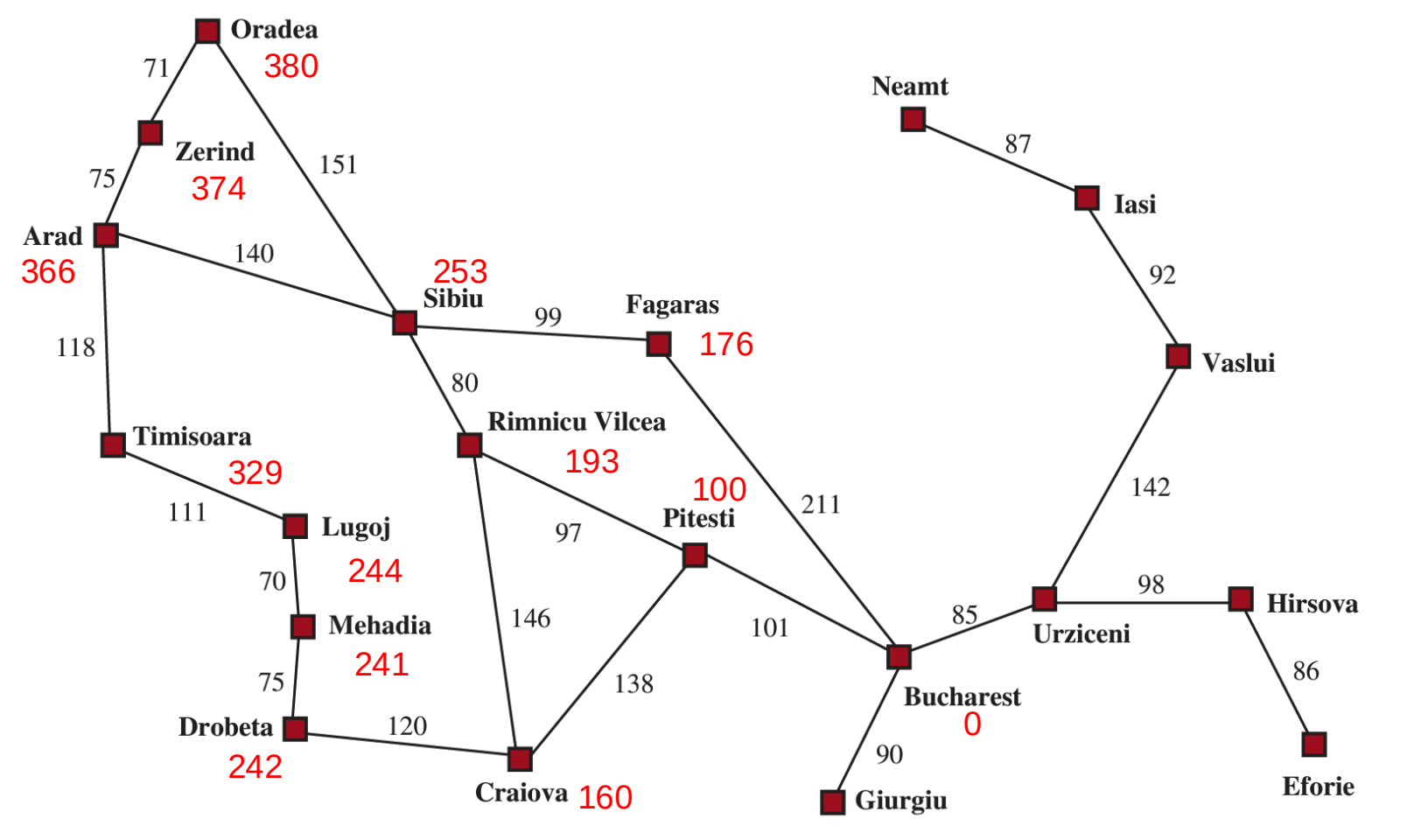
	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		10	9	8	7	6	5	4		2
	13		11						5		3
	14	13	12		10	9	8	7	6		4
			13		11						5
A	16	15	14		12	11	10	9	8	7	6

A heuristic for route finding (in Romania)

- Assuming Bucharest is the goal state, the following table gives values for a suitable heuristic function $h(n)$
  - The values are the straight-line distances from the relevant town ("state") to the goal state, i.e., we use the **straight-line distance**

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

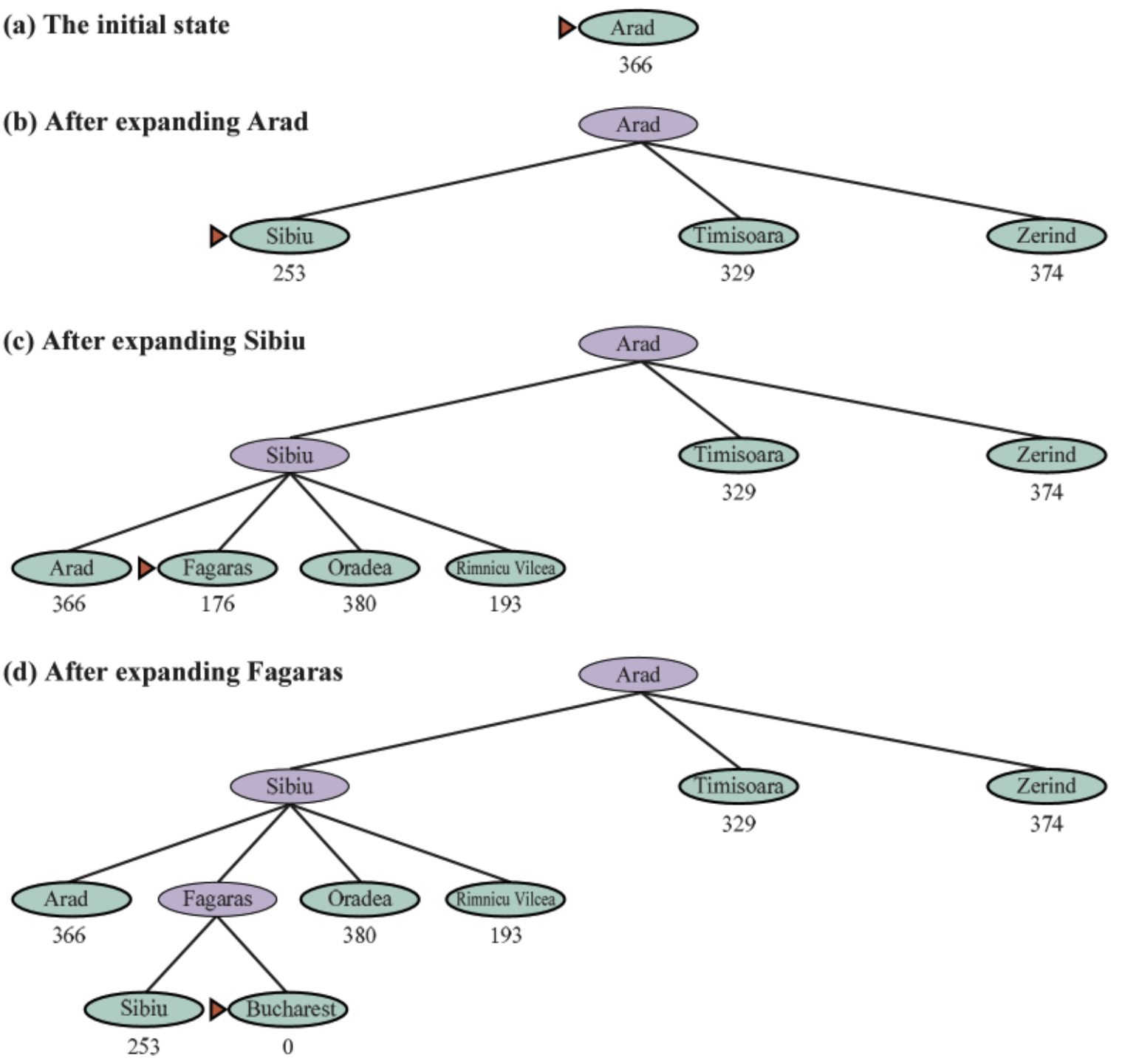
Route finding from Arad to Bucharest



Greedy best-first search

- Starting with the town Arad as the initial state, greedy best-first search will expand the search tree as shown on the next slide
- It finds a solution (guaranteed for finite search spaces), but this solution is not necessarily optimal
- In this example, the route found, via Sibiu and Fagaras, is 33 units longer than the optimum one

The search tree



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 state
- Given an estimate to a goal state $h(n)$ we can use $f(n) = g(n) + h(n)$ in the best-first search algorithm to get a **heuristic** (i.e., **informed**) **search** known as **A* search**
- A* search: $f(n) = g(n) + h(n)$

Greedy best-first search

```
1   Function GBFS(problem, h)
2     frontier = PQ{f=h, Node(problem.initial_state)}
3     while (true)
4       if (frontier.empty) return fail
5       node = pop_min_f(frontier)
6       if (node.state is goal) return solution
7       for c in Expand(node, problem)
8           frontier.add(c)
```

A* search

```
1  Function AStar(problem, h)
2    frontier = PQ{f=g+h, Node(problem.initial_state)}
3    while (true)
4      if (frontier.empty) return fail
5      node = pop_min_f(frontier)
6      if (node.state is goal) return solution
7      for c in Expand(node, problem)
8          frontier.add(c)
```

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

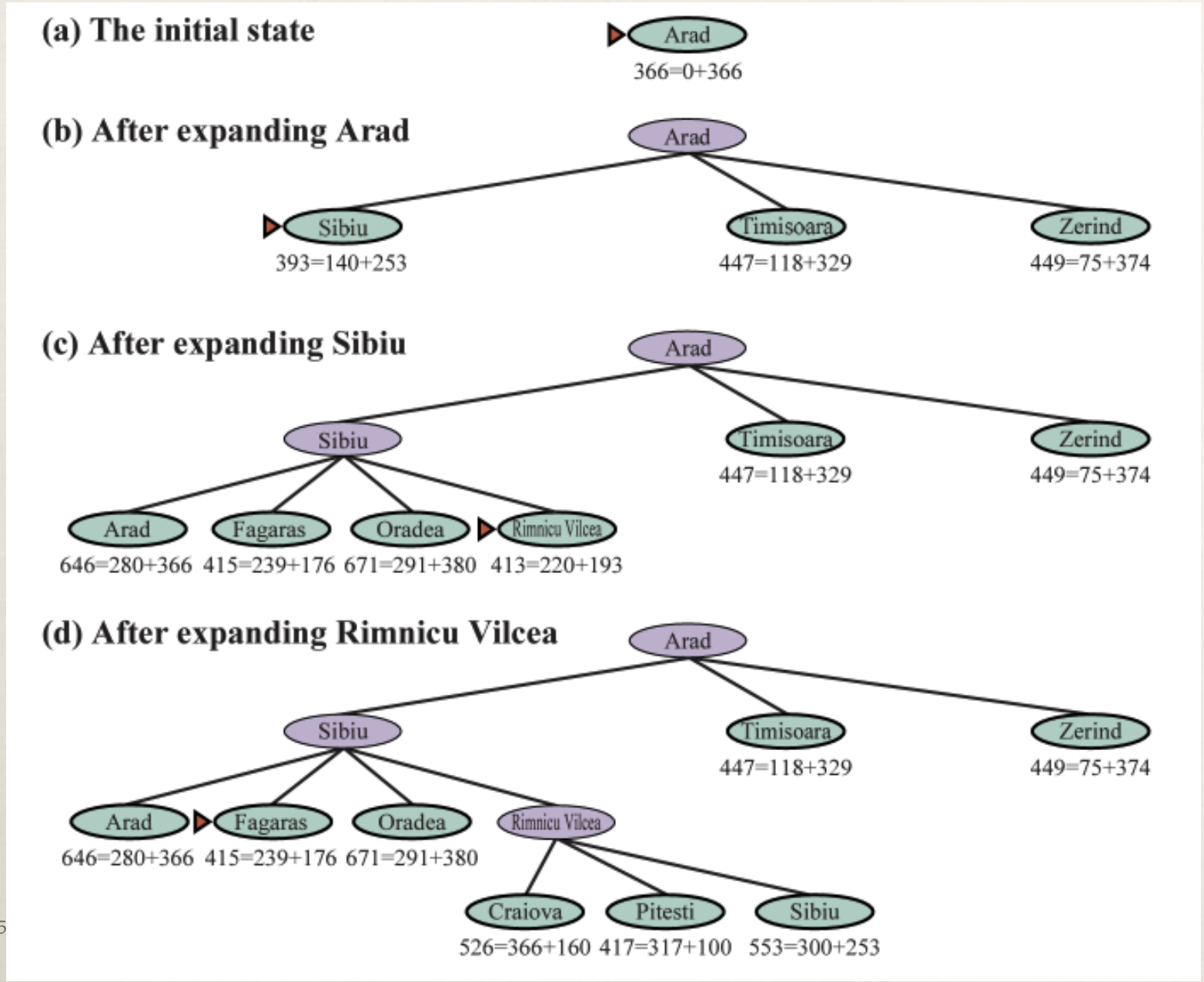
A* search

	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

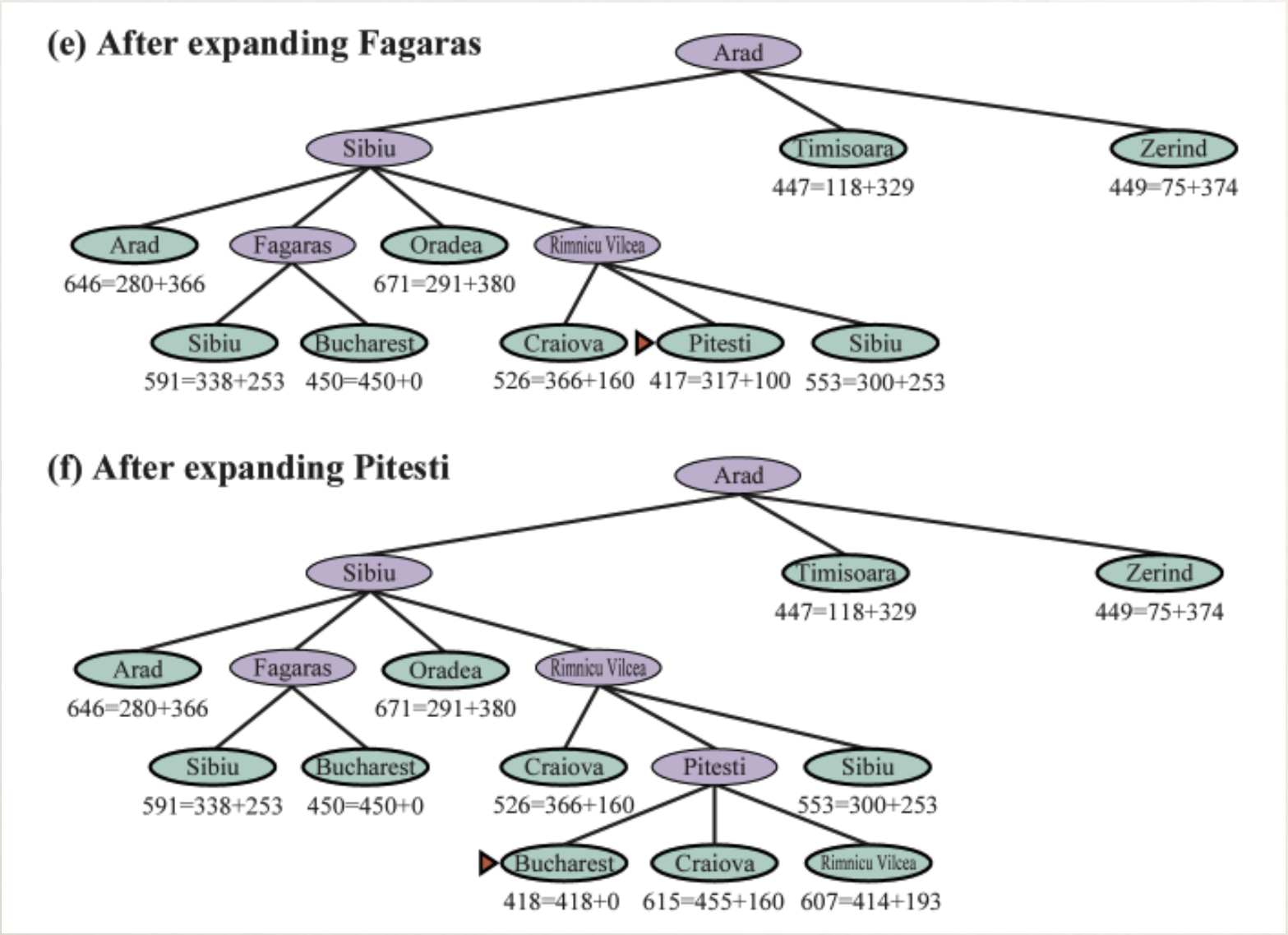
A* search

	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: using $f(n) = g(n) + h(n)$



A* search continued



Uniform-cost search updated

```
1  Function UCS(problem)
2    reached = {}
3    frontier = PQ{f=g, Node(problem.initial_state)}
4    while (true)
5      if (frontier.empty) return fail
6      node = pop_min_f(frontier)
7      if (node.state is goal) return solution
8      if (node.state in reached) continue
9      reached.add(node.state)
10     for c in Expand(node, problem)
11        frontier.add(c)
```

Greedy best-first search updated

```
1  Function GBFS-Updated(problem)
2    reached = {}
3    frontier = PQ{f=h, Node(problem.initial_state)}
4    while (true)
5      if (frontier.empty) return fail
6      node = pop_min_f(frontier)
7      if (node.state is goal) return solution
8      if (node.state in reached) continue
9      reached.add(node.state)
10     for c in Expand(node, problem)
11        frontier.add(c)
```

A* search updated

```
1  Function UCS(problem)
2    reached = {}
3    frontier = PQ{f=g+h, Node(problem.initial_state)}
4    while (true)
5      if (frontier.empty) return fail
6      node = pop_min_f(frontier)
7      if (node.state is goal) return solution
8      if (node.state in reached) continue
9      reached.add(node.state)
10     for c in Expand(node, problem)
11        frontier.add(c)
```

Admissible heuristics

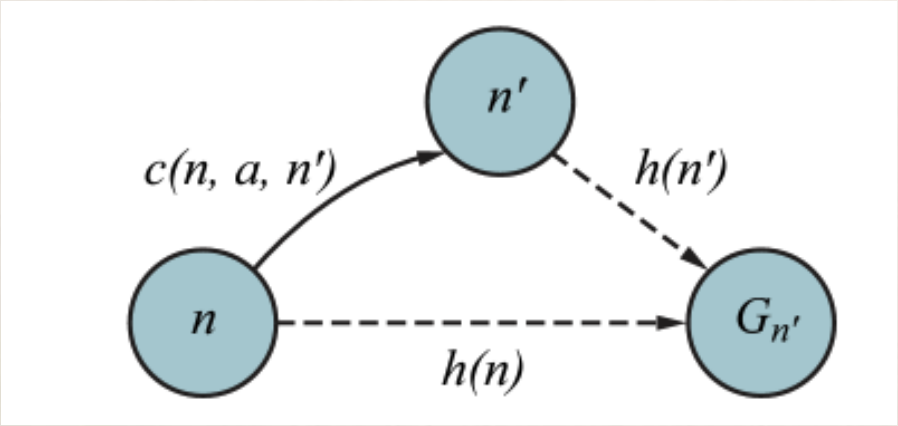
- We want the estimate of the cost provided by $h(n)$ to be as accurate as possible (while being cheap to compute!)
  - Ideally, the heuristic function would give us the true cost of the shortest path from the candidate node's state to a goal state
- Is A* search guaranteed to give use the lowest cost path?
- Yes, assuming the heuristic function is optimistic and *never overestimates* the true cost of getting to a goal state
  - Optimality can be shown using a proof by contradiction
- The straight-line distance is a suitable optimistic heuristic in the route-finding problem: it will never be greater than the actual distance
  - $0 <= h(n) <= h^*(n)$; where $h^*(n)$ is the true cost to the nearest goal
- This kind of optimistic heuristic is called **admissible**
- We want admissible heuristics whose estimates are as close as possible to the true cost of the path!

Consistent heuristics

- For search efficiency, it would be nice to be able to guarantee the following condition during A* search:
  - The first time we reach a state (i.e., create a node for a state in the search tree), this will be on an optimal path to the goal
  - Then, the state will only ever occur once in the frontier, and we will never update the best node in the dictionary of reached states
- This can be achieved by using a heuristic that is not just **admissible** but also **consistent:**

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

where $c(n, a, n')$ is the cost of taking action a

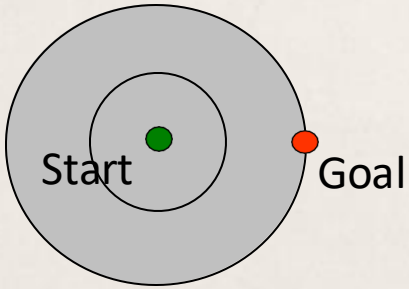


A* search

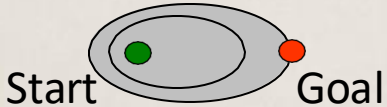
- Optimal if
  - $h(n)$ is admissible (never overestimates the true cost), and
  - $h(n)$ is consistent (for every node n and successor n' with step cost c, $h(n) \leq c(n, a, n') + h(n')$)

Search contours: UCS vs A*

- Uniform-cost expands equally in all "directions"



- A* expands mainly toward the goal, but does hedge its bets to ensure optimality

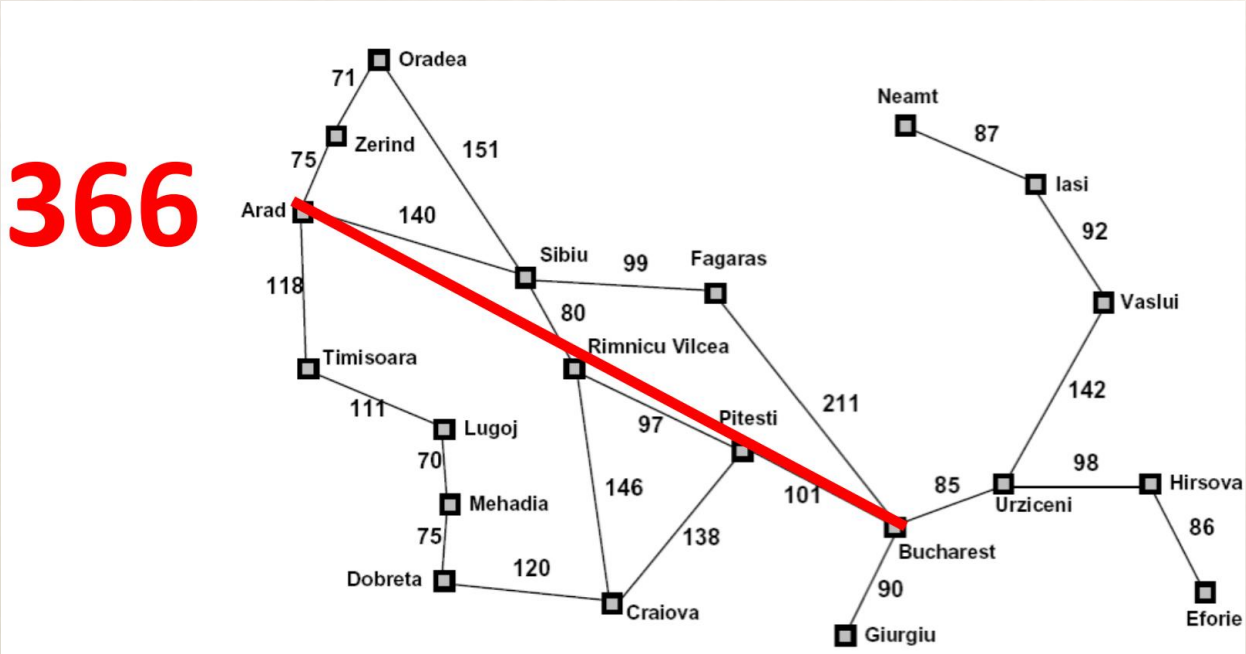


Memory-bounded search: beam search

- A* search with consistent heuristics is **optimally efficient**
- However, if the search problem is hard, it may require too much storage space or computation to complete
- We can sacrifice completeness and optimality of the search algorithm by limiting the size of the frontier
  - A search algorithm is **complete** if it is guaranteed to find a solution in cases where a solution exists
- Limiting the size of the frontier gives us **beam search**
- Assuming the size of the frontier is limited to *k* nodes, the simplest approach is to keep only the *k* best nodes
  - An efficient way to do this is to use a min-max heap to implement a double-ended priority queue as the frontier

Creating admissible heuristics

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to relaxed problems, where new actions are available
- Inadmissible heuristics are often useful too



Heuristics for the 8-puzzle

- One admissible heuristic: the number of misplaced tiles
- Better: the sum of the (Manhattan) distances between each tile's current location and its location in the goal state
- These heuristics are both obtained by considering appropriately **relaxed** versions of the original problem
  - Relaxation in the first case: a tile can move anywhere in one move
  - Relaxation in the second case: a tile can move by one square even if that square is already occupied
- Both heuristics are admissible and in fact even consistent (because they provide exact costs for the relaxed problems)
- Another option for getting heuristics is to create a **pattern database** with pre-computed solutions to subproblems

References

- Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley (ai.berkeley.edu)
- Dr. David Churchill, Department of Computer Science Memorial University of Newfoundland (http://www.cs.mun.ca/~dchurchill)
- CS50's Introduction to Artificial Intelligence with Python 2020 (https://cs50.harvard.edu/ai/2024/)