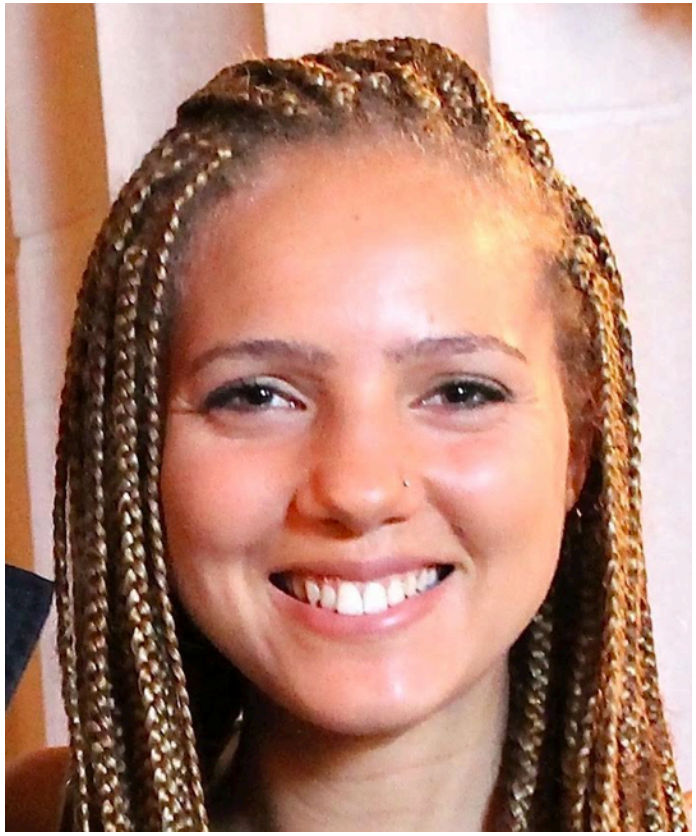
Blindingly Fast Search

Chris Piech

CS 106B
Lecture 21
Mar 2, 2016

Megan Teaches on Friday



* Chris is presenting research in Tennessee.
Office hours for today moved to next Friday

Assignment 7

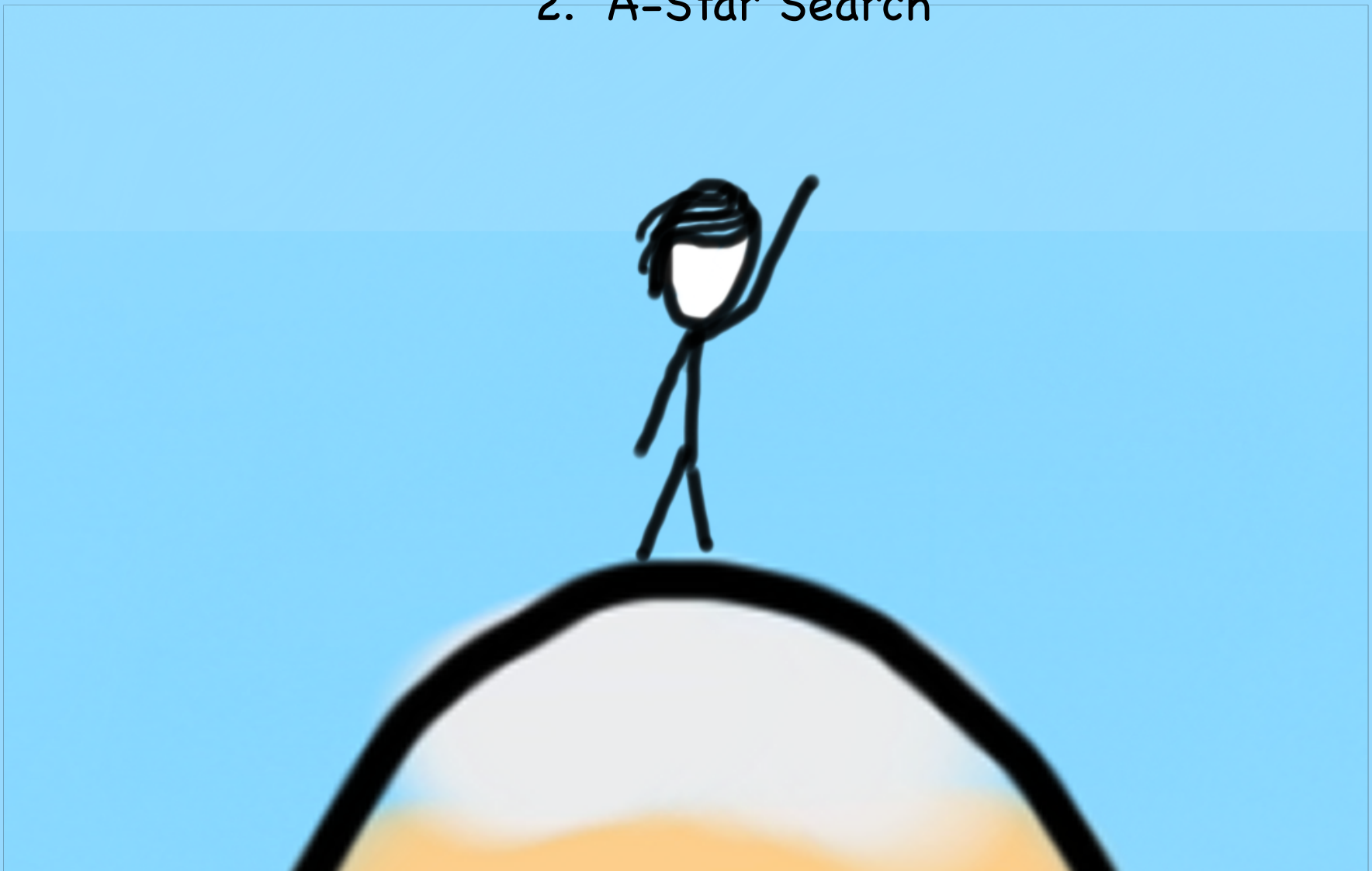
Due: March 11th

No Late Days

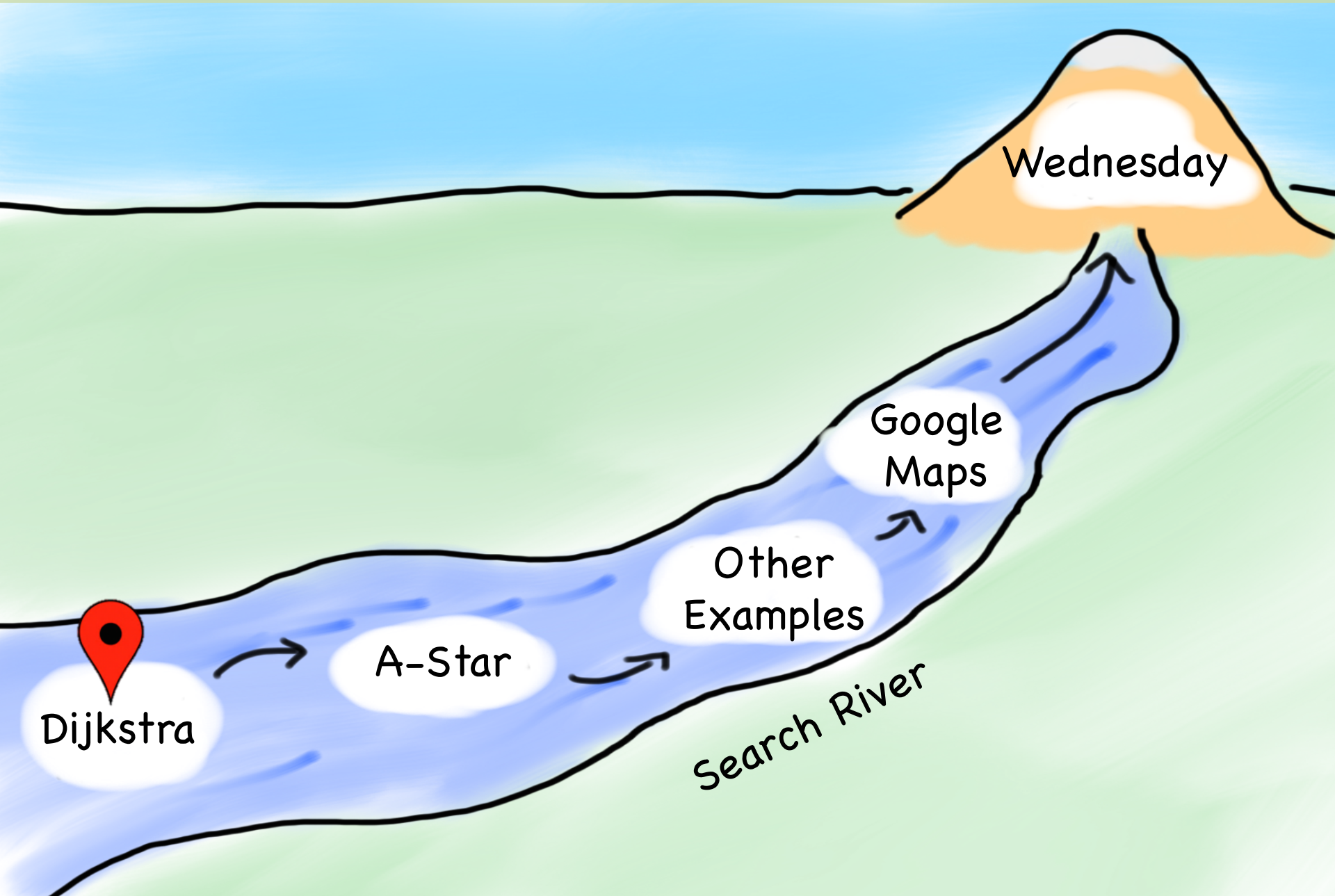
YEAH: TBD

Today's Goal

1. Dijkstra's Search
2. A-Star Search



Today's Route



How Does Google Maps Work?

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

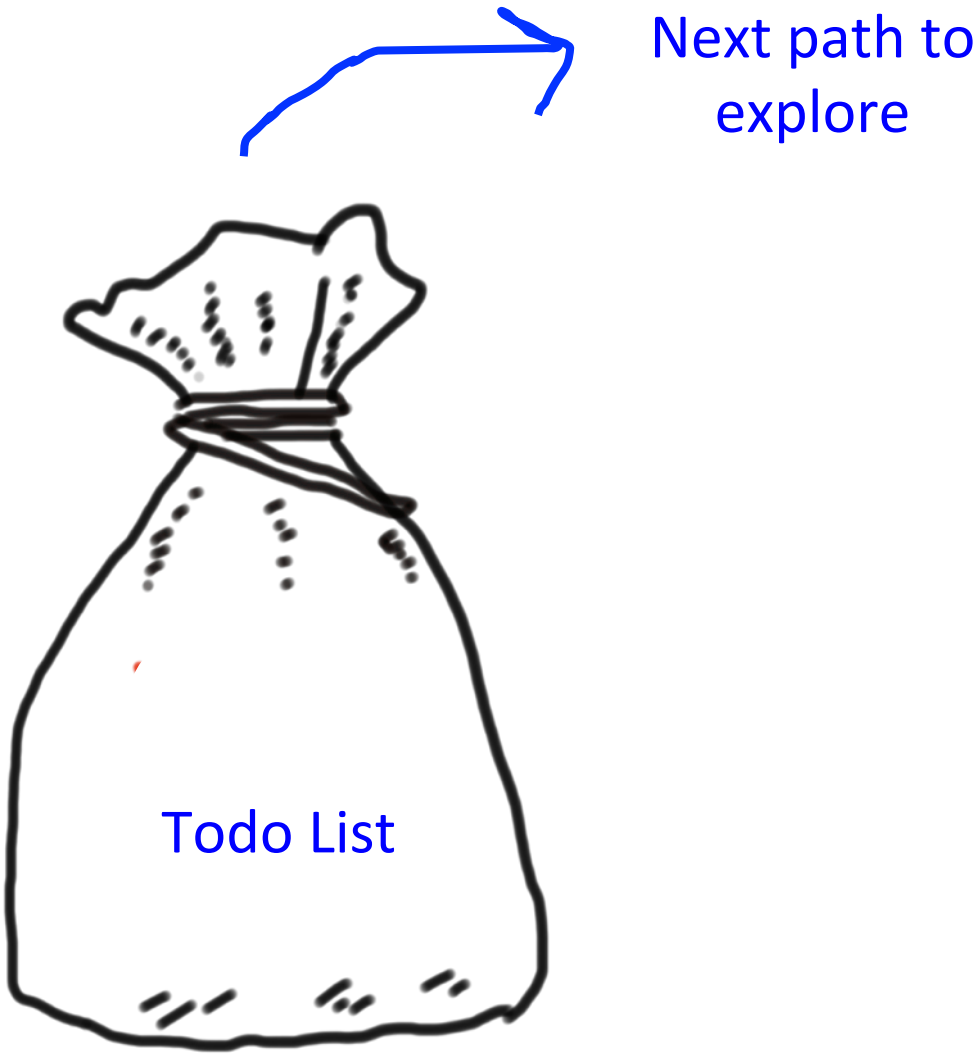
Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  Take a path out of the todo list
  Call the last node in the path "currNode"
  If "currNode" is the goal, you are done.
  If you have seen currNode before, skip it.
  for all neighbors of currNode
    Make a newPath = path + neighbor
    Add the new path to the todo list
  }
}
```

Todo list collection type



Depth First Search

```
stack = Stack()
stack.push(newPath(startNode))
seen = Set();
while !stack.isEmpty():
    currPath = stack.pop()
    currState = last(currPath);
    if(currState is goal) return currPath;
    if(seen contains currState) continue;
    seen.add(currState);
    for nextState in getNextStates(currState)
        path = newPath(currPath, nextState);
        stack.push(path);
    }
}
```

Depth First Search

```
stack = Stack()
stack.push(newPath(startNode))
seen = Set();
while !stack.isEmpty():
    currPath = stack.pop()
    currState = last(currPath);
    if(currState is goal) return currPath;
    if(seen contains currState) continue;
    seen.add(currState);
    for nextState in getNextStates(currState)
        path = newPath(currPath, nextState);
        stack.push(path);
    }
}
```

Breadth First Search

```
queue = Queue()
queue.enqueue(newPath(startNode))
seen = Set();
while !queue.isEmpty():
    currPath = queue.dequeue()
    currState = last(currPath);
    if(currState is goal) return currPath;
    if(seen contains currState) continue;
    seen.add(currState);
    for nextState in getNextStates(currState)
        path = newPath(currPath, nextState);
        queue.enqueue(path);
    }
}
```

Dijkstra's Algorithm

```
pQueue = PriorityQueue()
pQueue.enqueue(newPath(startNode), 0)
seen = Set();
while !pQueue.isEmpty():
    currPath = pQueue.dequeue()
    currState = last(currPath);
    if(currState is goal) return currPath;
    if(seen contains currState) continue;
    seen.add(currState);
    for nextState in getNextStates(currState)
        path = newPath(currPath, nextState);
        pQueue.enqueue(path, getCost(path));
    }
}
```

Represent a Path?

Paths

```
Vector<Vertex *> path;
```

Paths

```
Vector<Vertex *> path;
```

```
int getCost(Vector<Vertex *> path) {
  // add the cost of each edge
}
```

Paths

```
Vector<Vertex *> path;
```

```
int getCost(Vector<Vertex *> path) {
  // add the cost of each edge
}
```

```
Vector<Vertex *> getPath(Vector<Vertex *> path,
                         Vertex * next) {
  Vector<Vertex *> newPath = path;
  newPath.add(next);
  return newPath;
}
```

Summary

Algorithm	Collection	Benefit
Depth First Search	Stack	Uses little memory
Breadth First Search	Queue	Finds shortest hops
Dijkstra's	Priority Queue	Finds shortest path

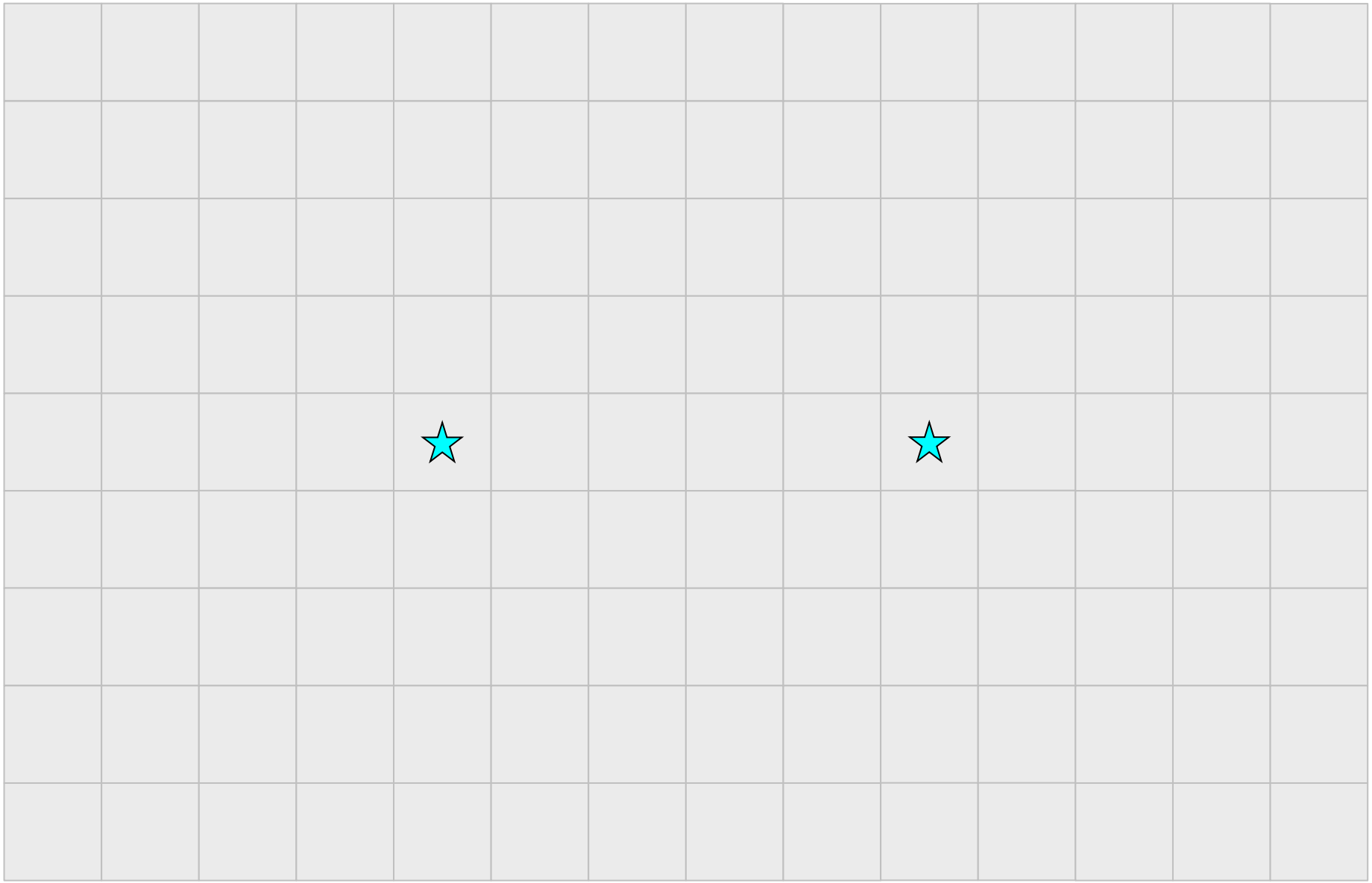
Try it live

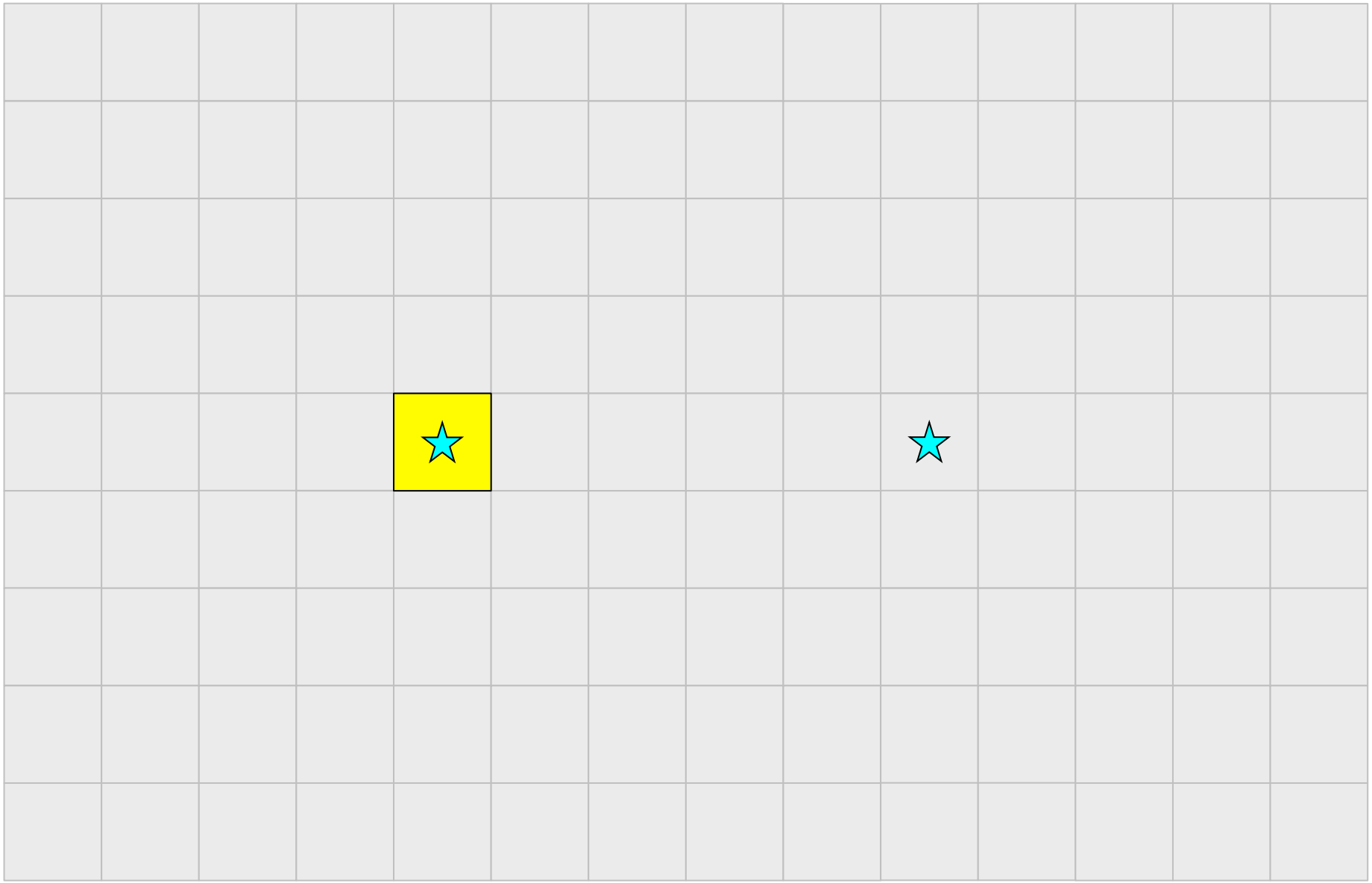
DFS is a mess

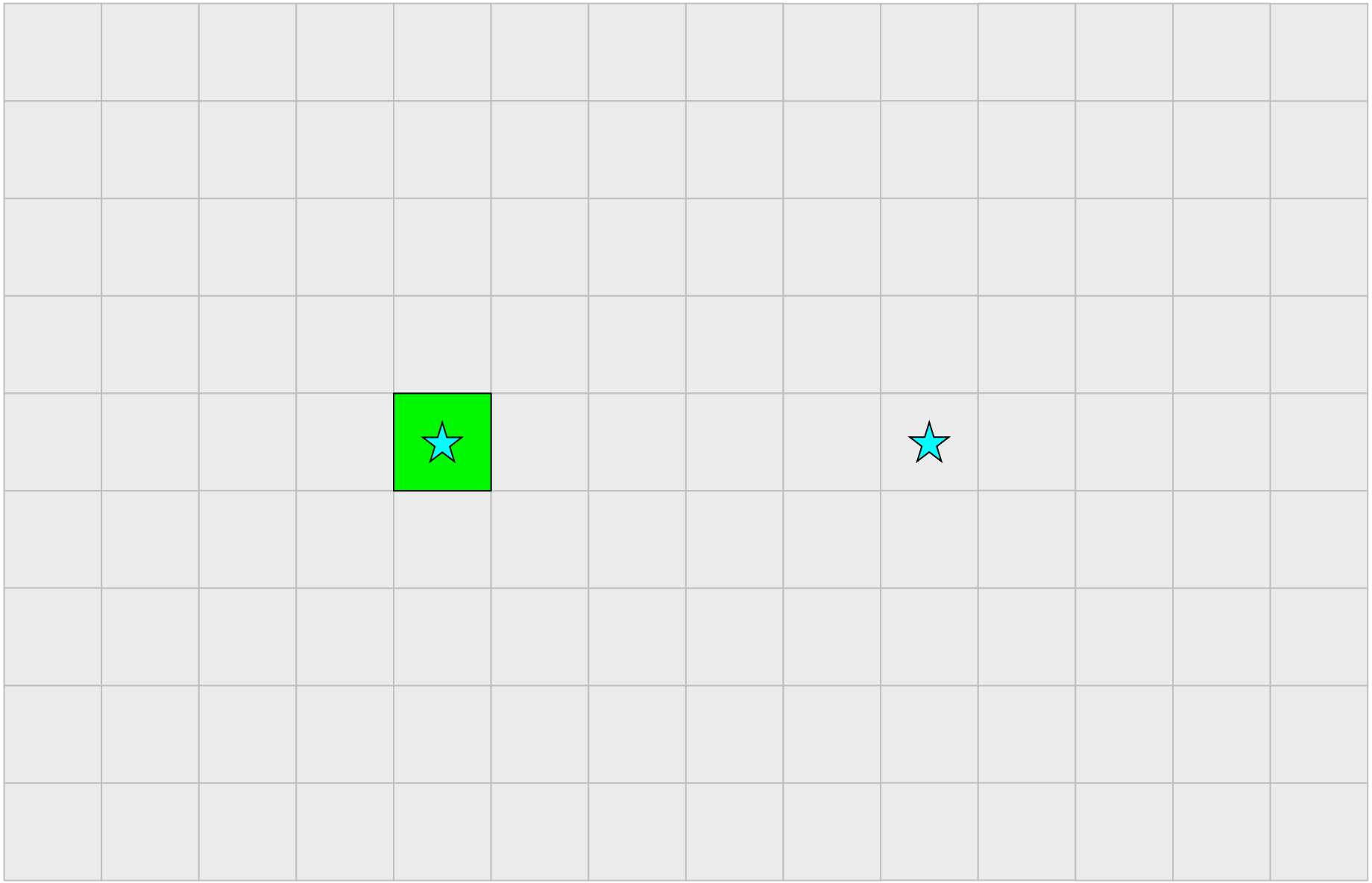
Dijkstra is terrific

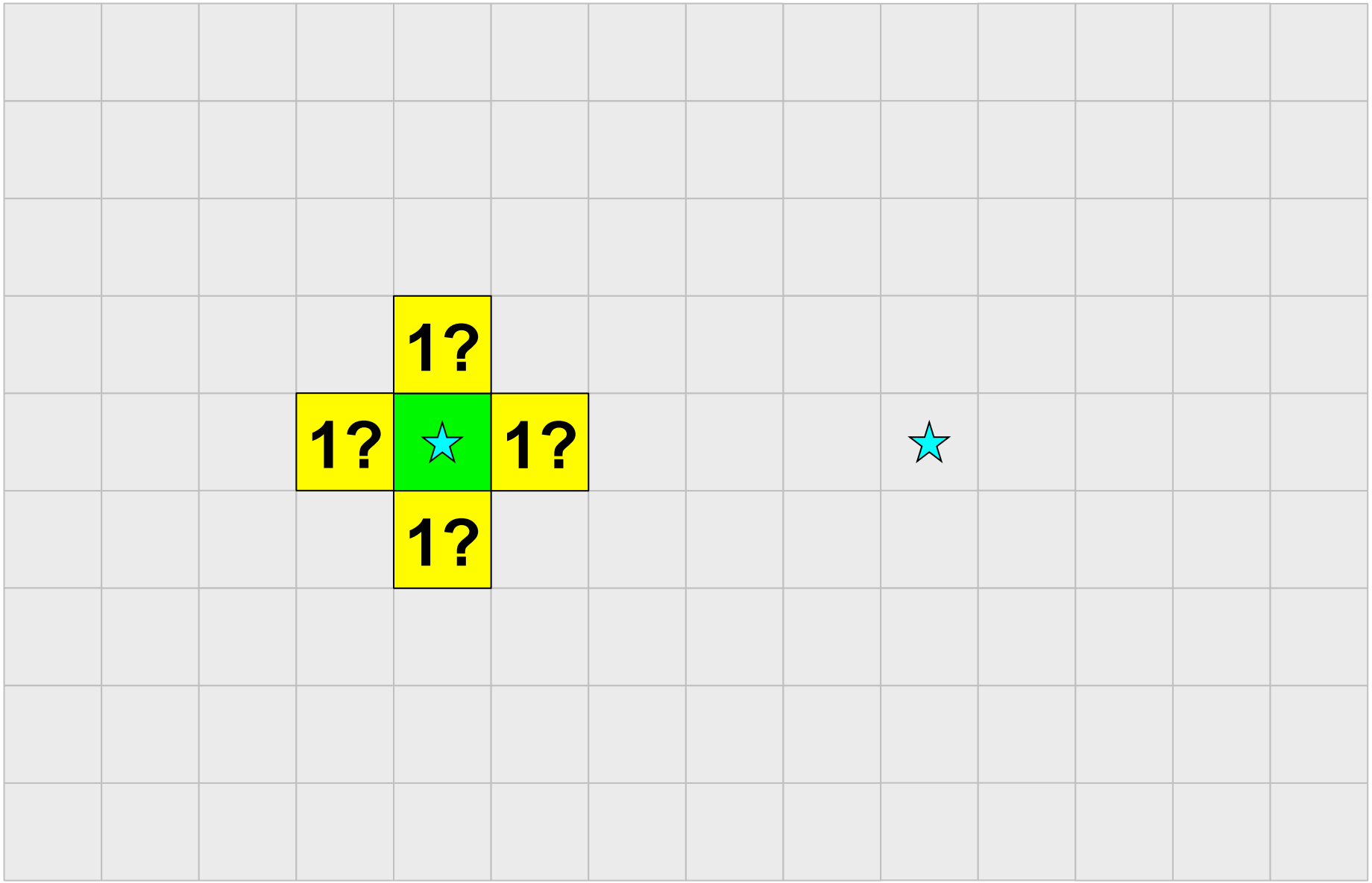
Dijkstra's Algorithm

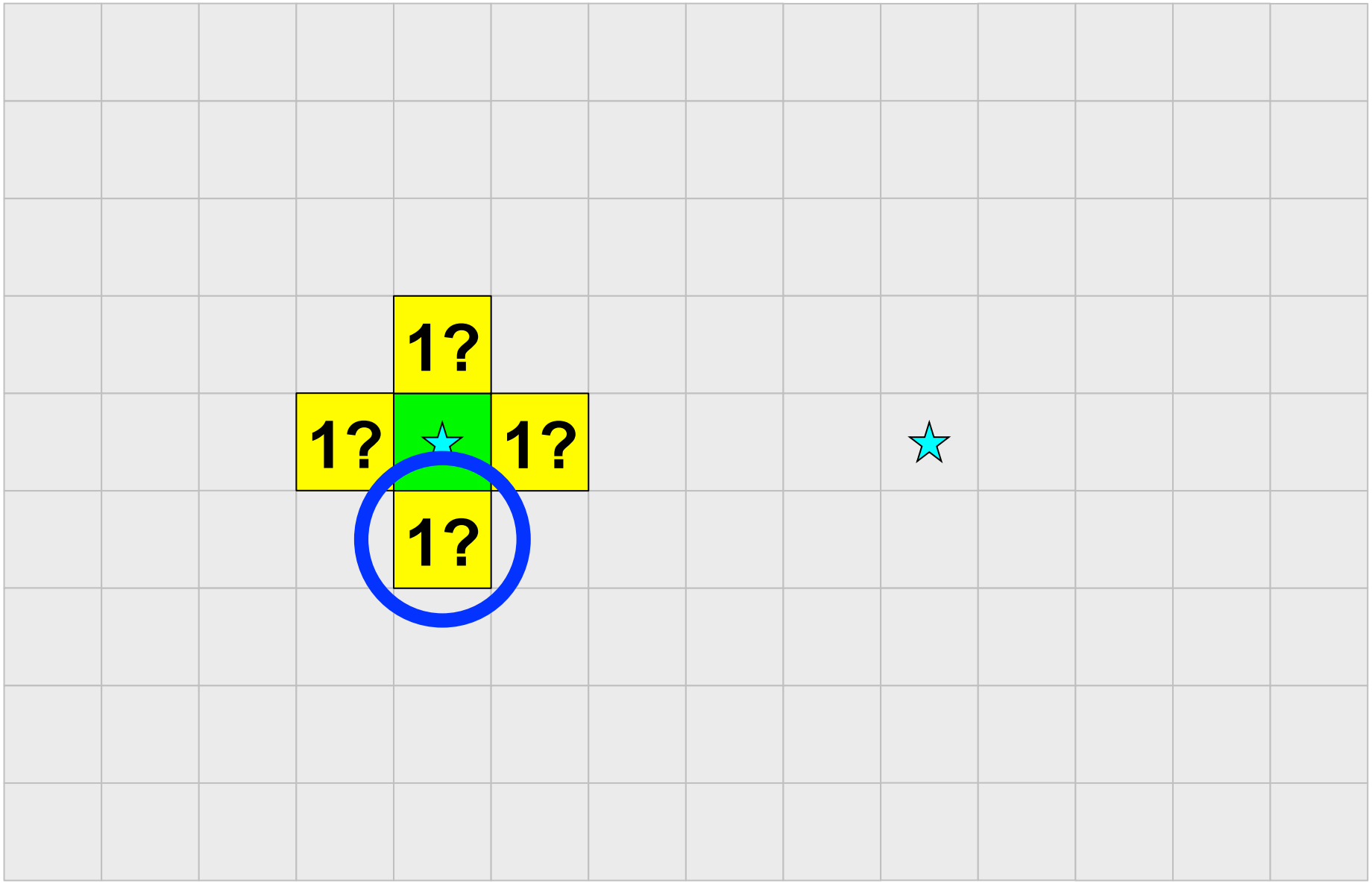
```
pQueue = PriorityQueue()
pQueue.enqueue(newPath(startNode), 0)
seen = Set();
while !pQueue.isEmpty():
    currPath = pQueue.dequeue()
    currState = last(currPath);
    if(currState is goal) return currPath;
    if(seen contains currState) continue;
    seen.add(currState);
    for nextState in getNextStates(currState)
        path = newPath(currPath, nextState);
        pQueue.enqueue(path, getCost(path));
    }
}
```

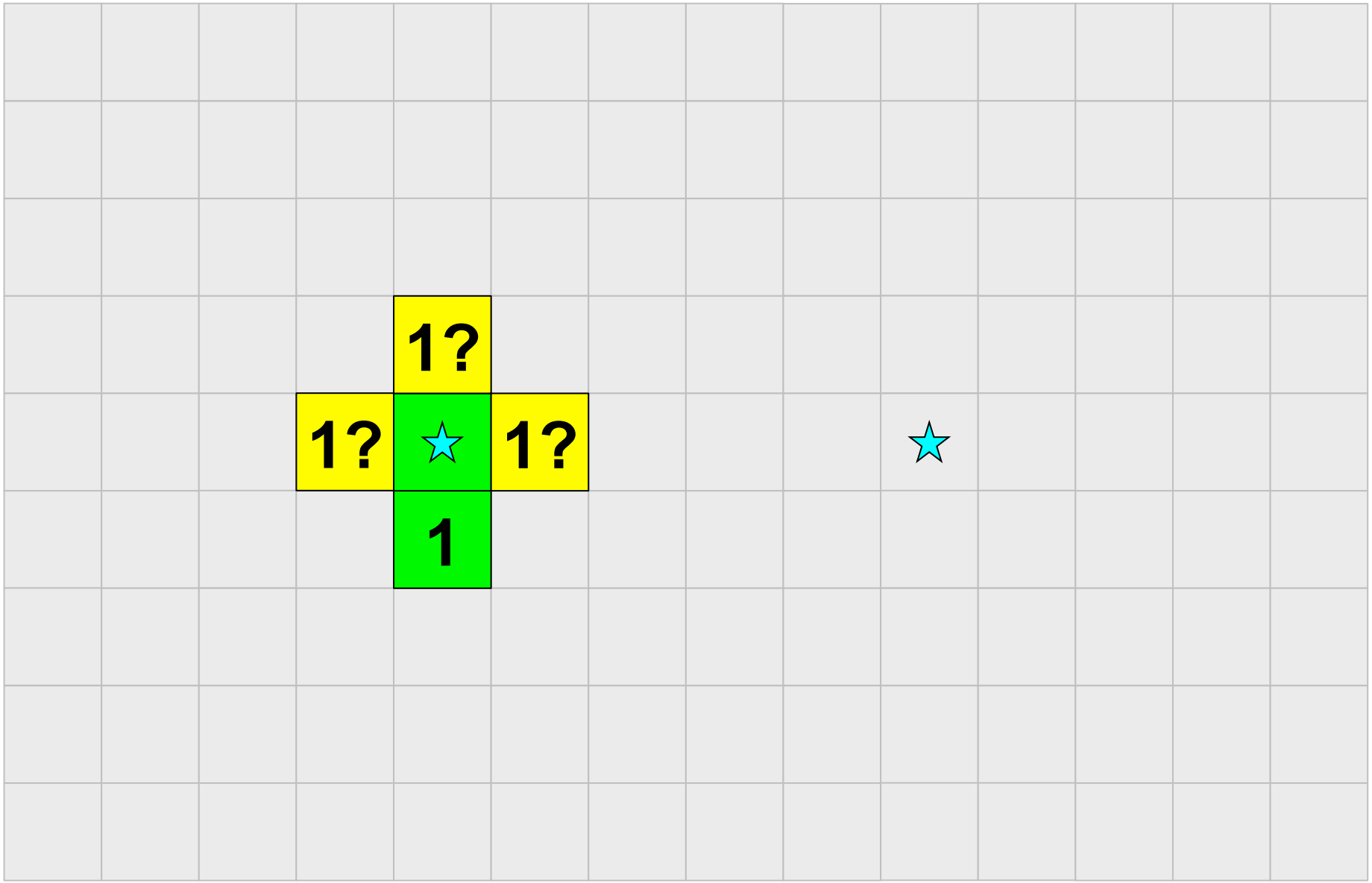


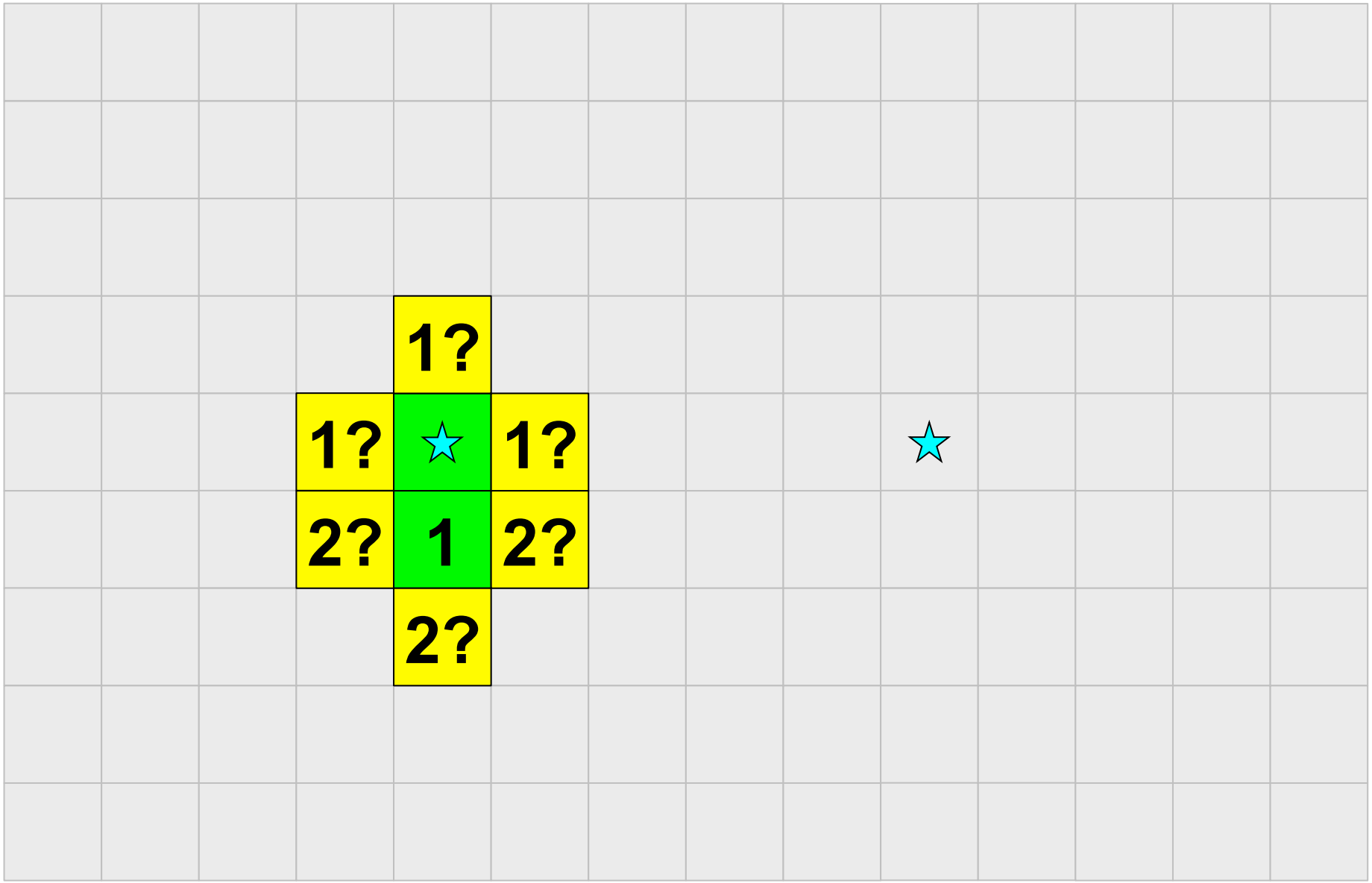


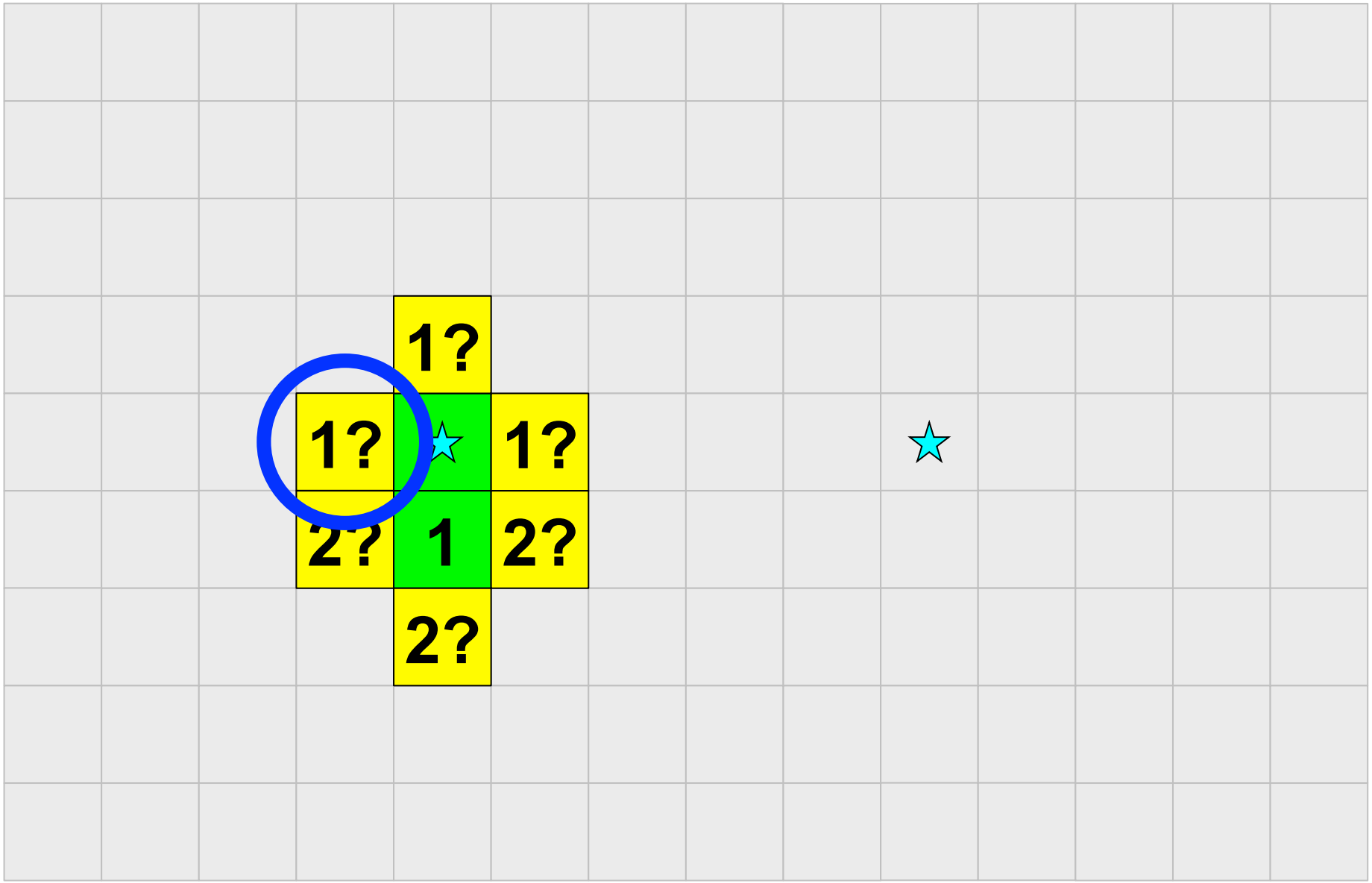


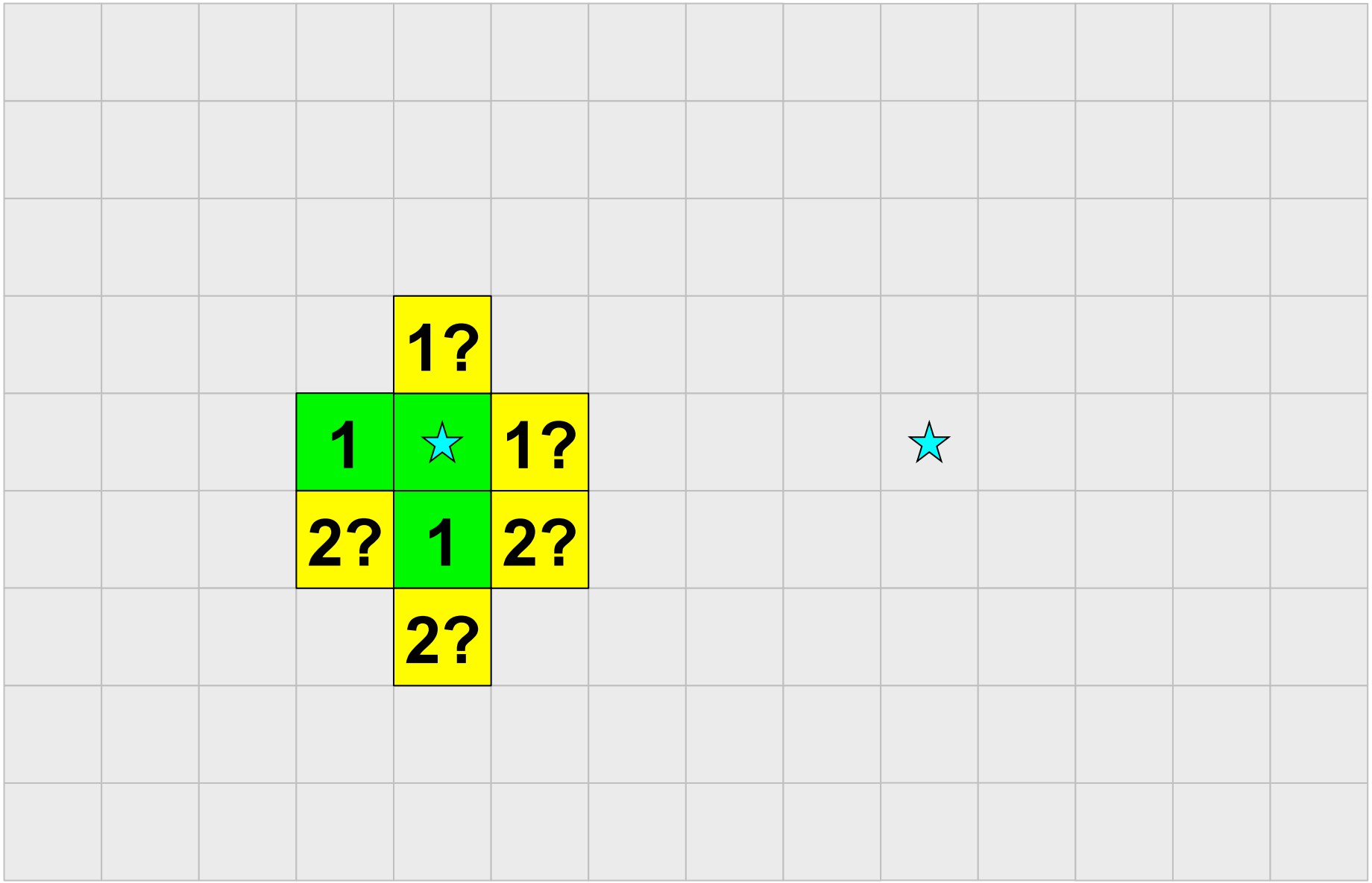

1?
1?
1?
1?

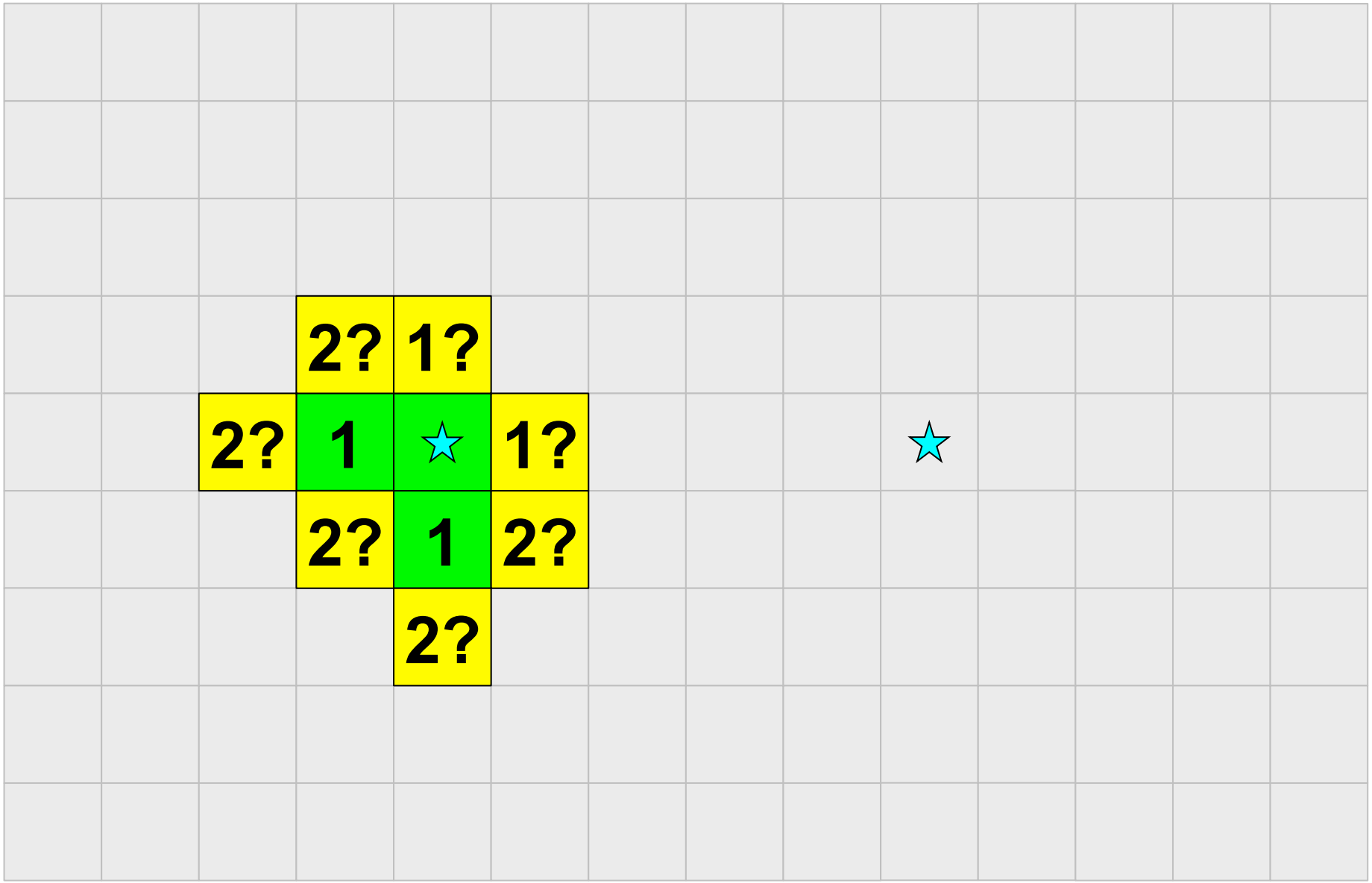

1?
1?
1?
1?

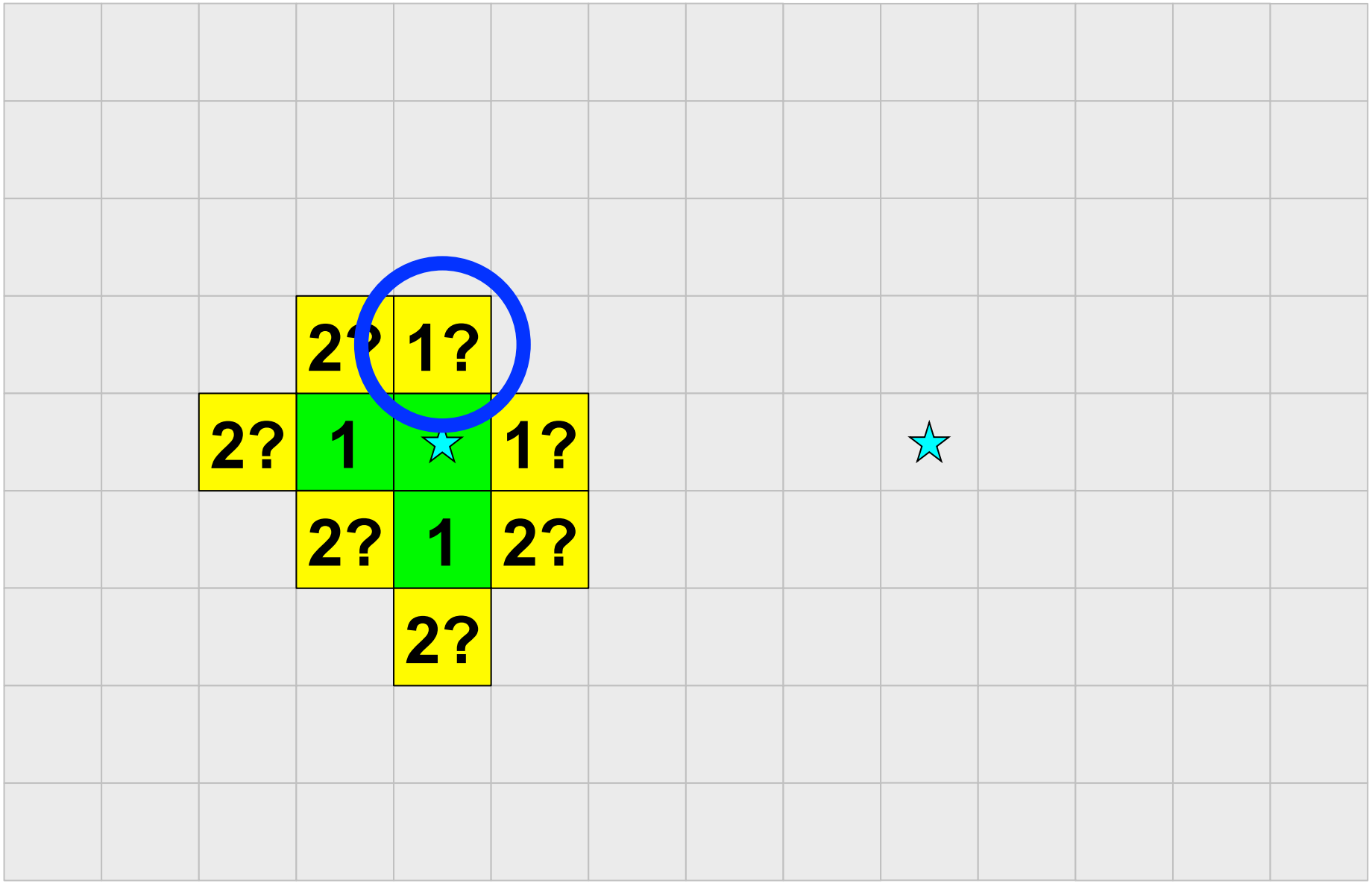

1?
1?
1?
1

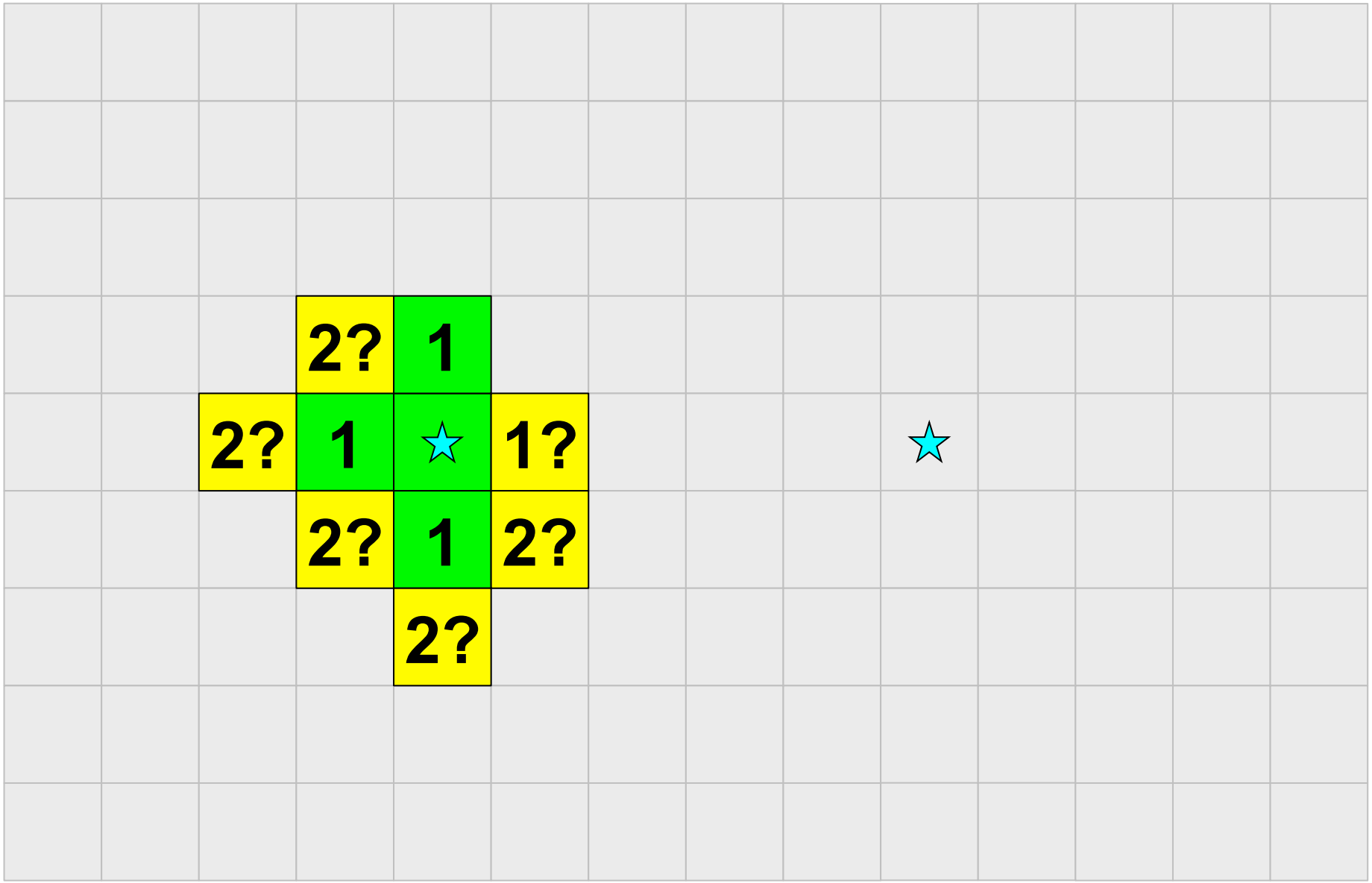

1?
1?
1?
2?
1
2?
2?

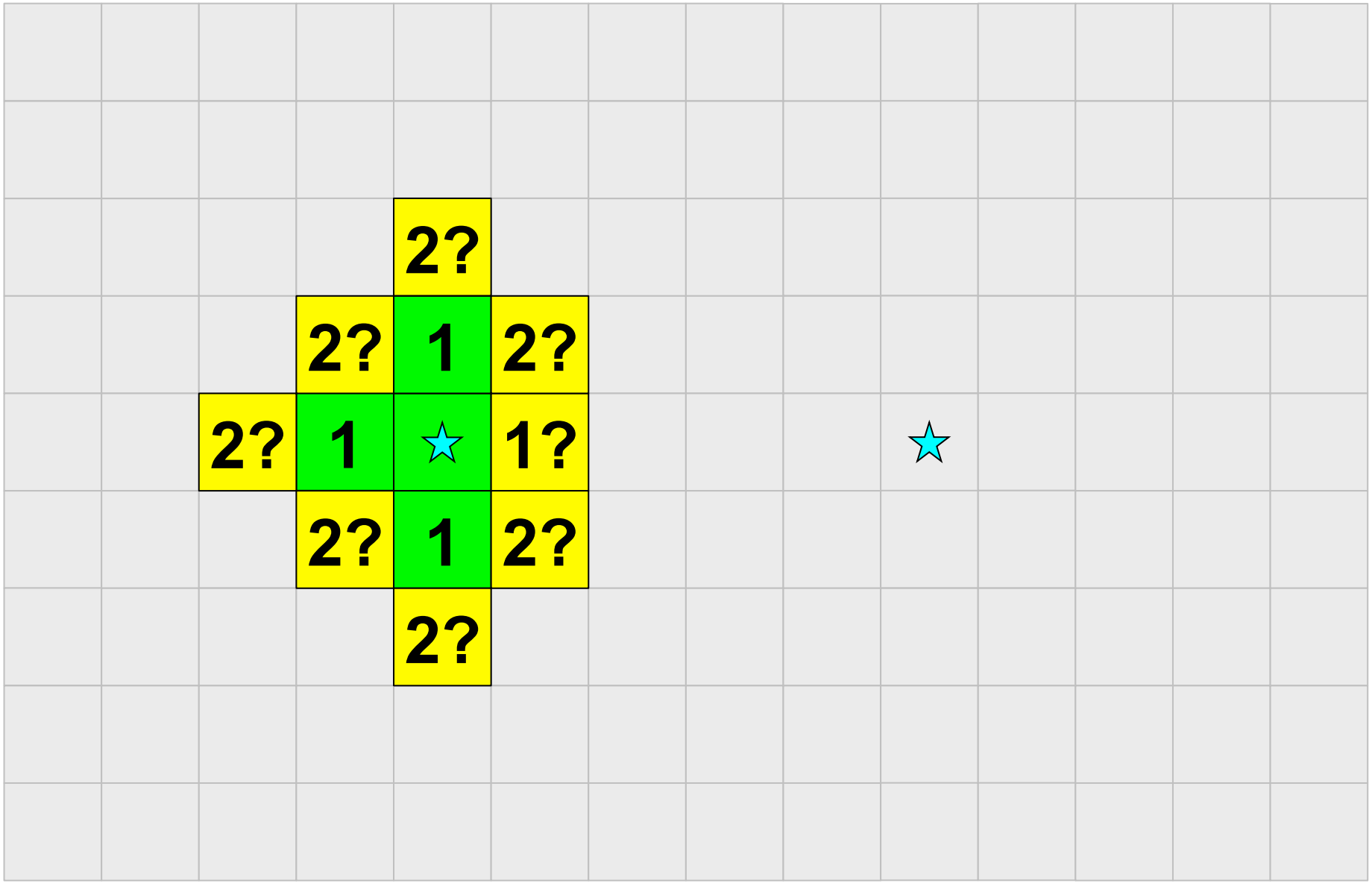

1?
1?
1?
2?
1
2?
2?

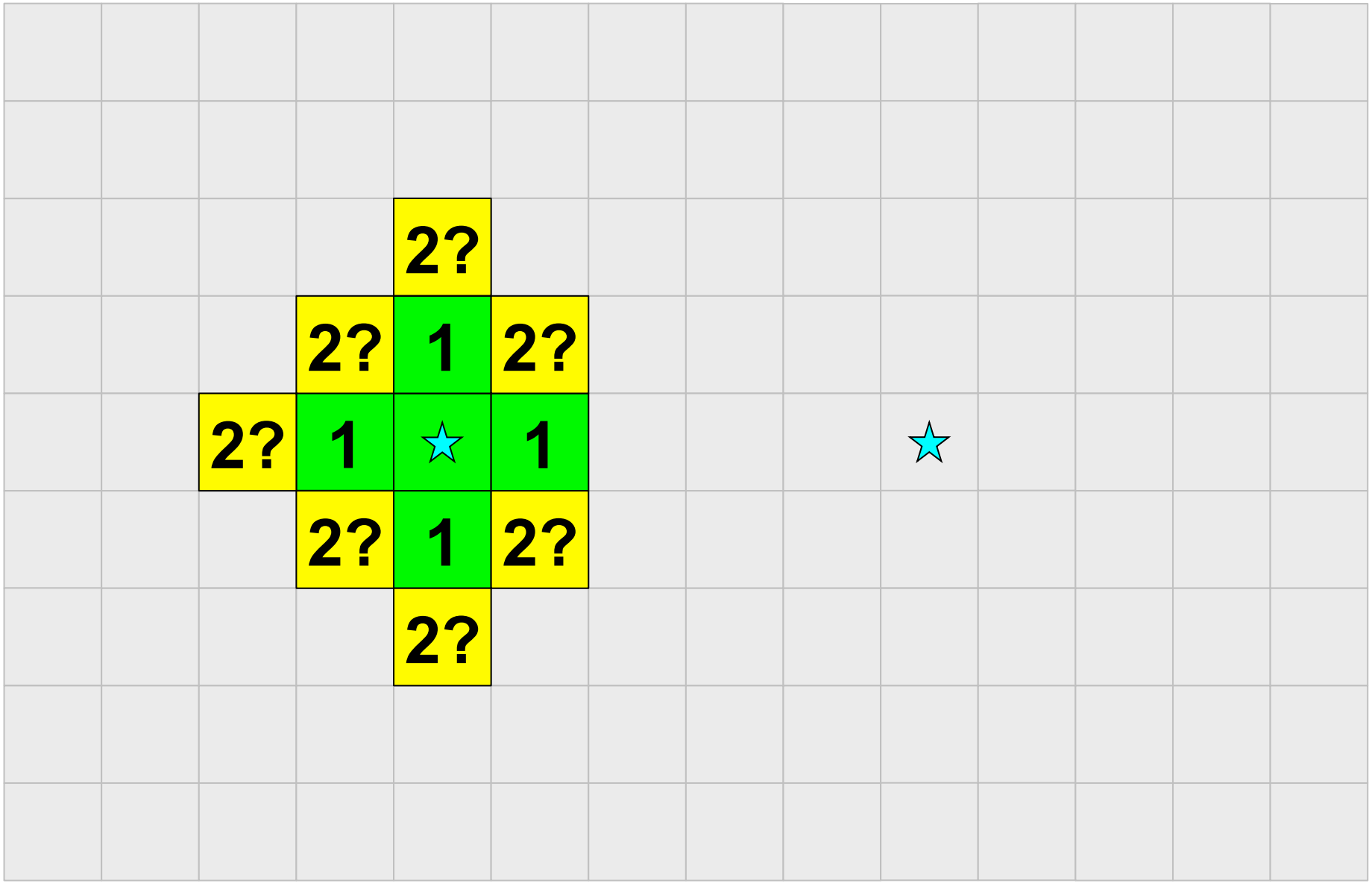

1?
1
1?
2?
1
2?
2?

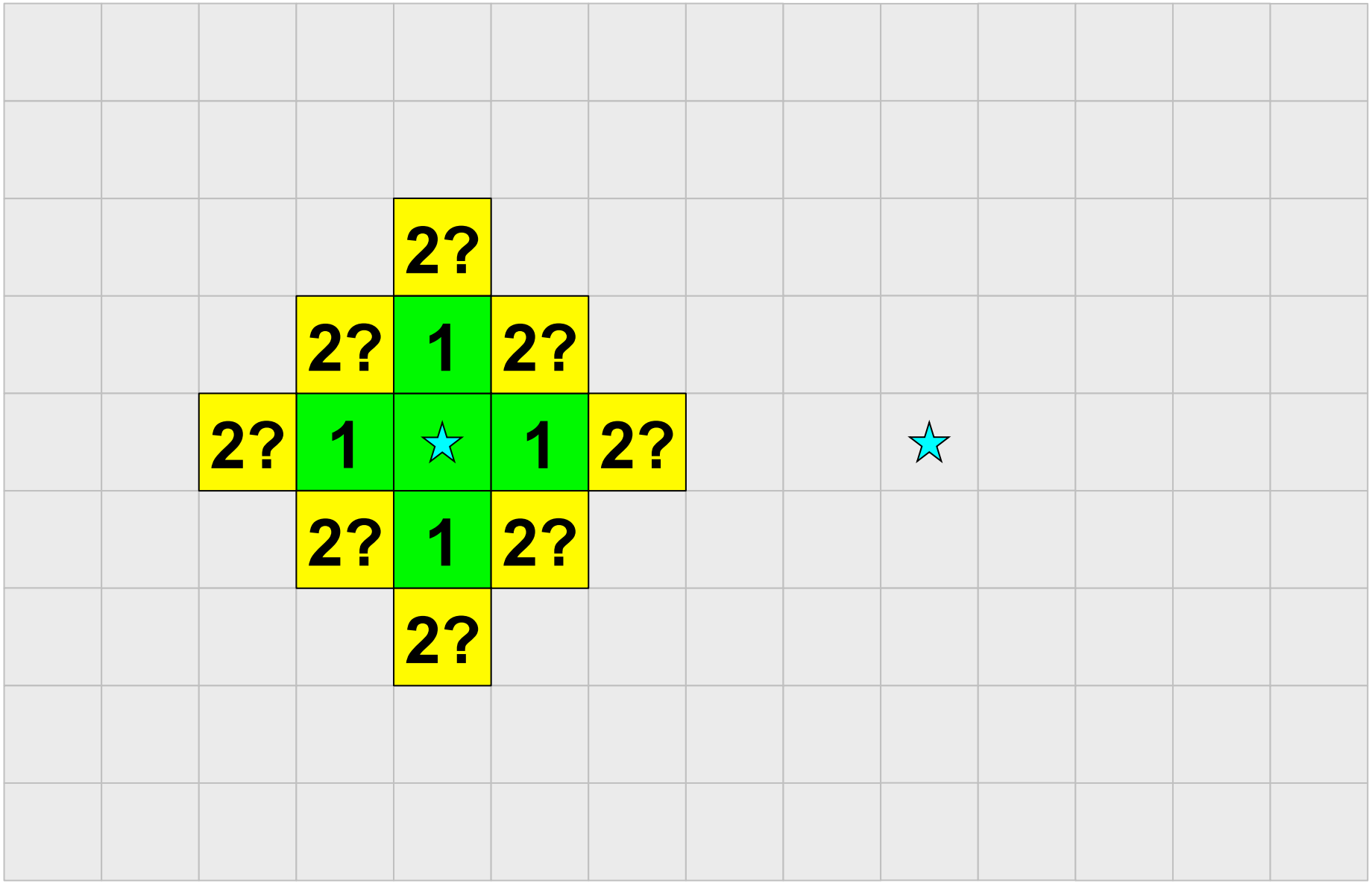

2?
1?
2?
1
1?
2?
1
2?
2?

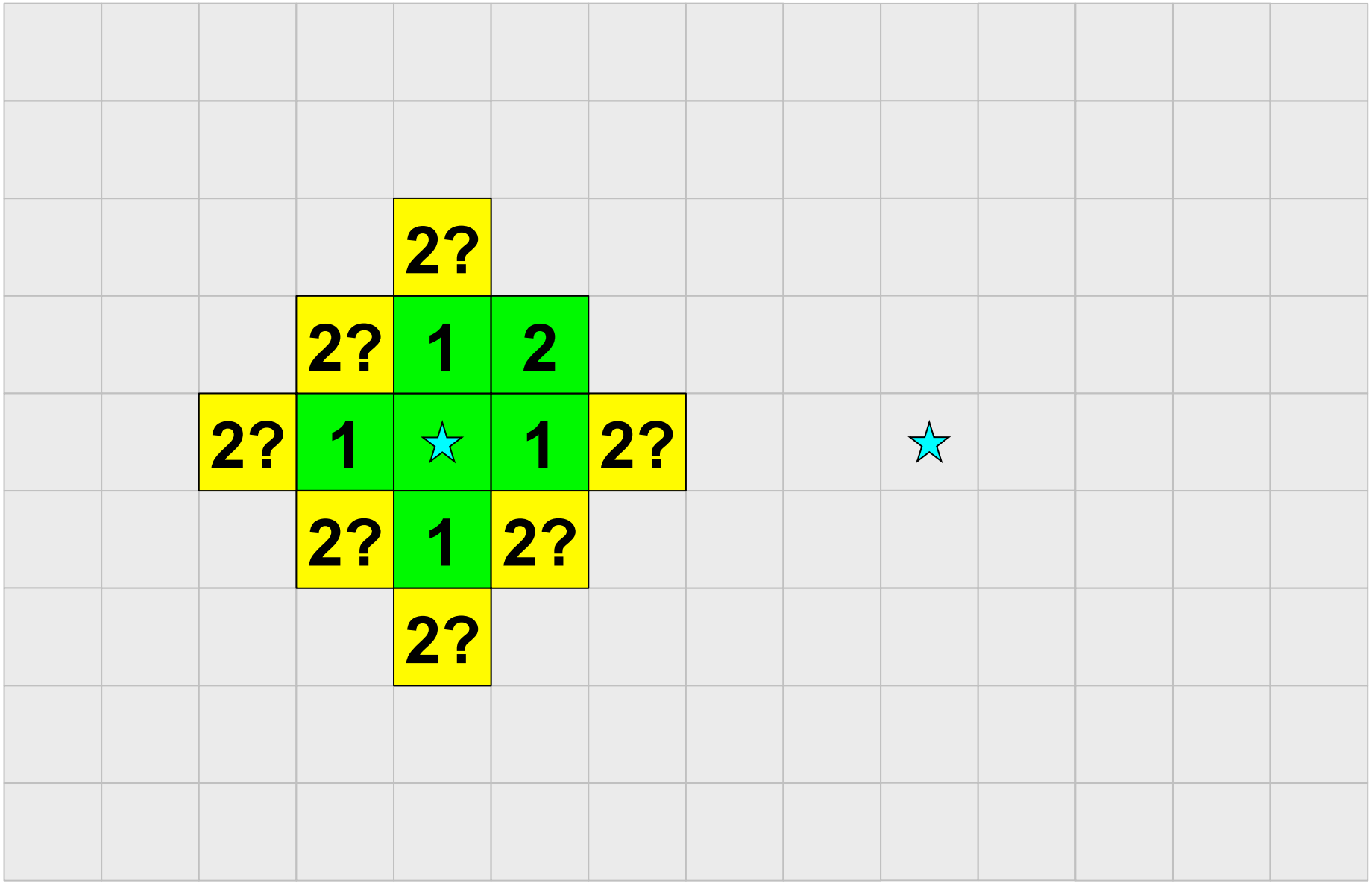

2?
1?
2?
1
1?
2?
1
2?
2?

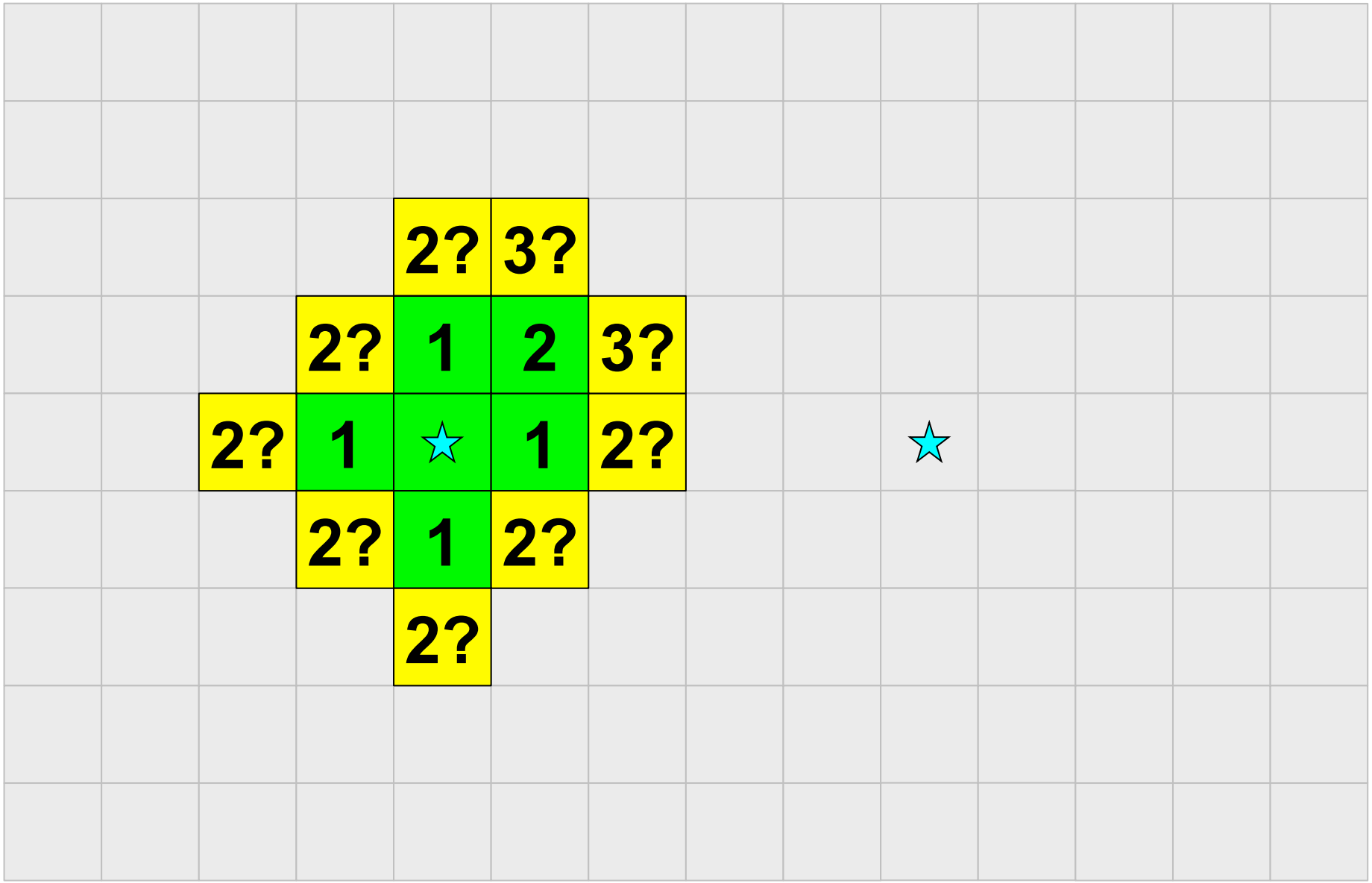

2?
1
2?
1
1?
2?
1
2?
2?

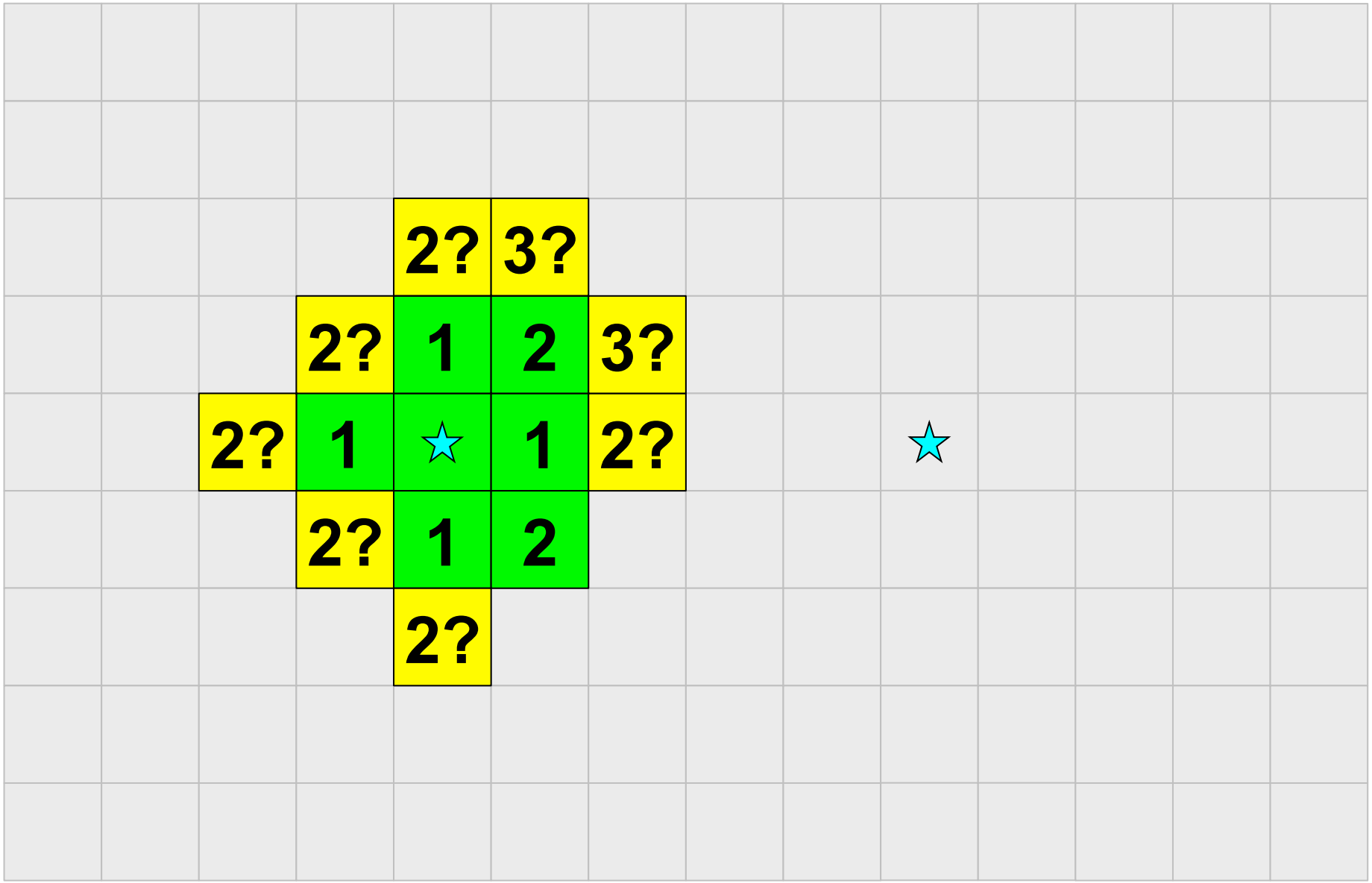

2?
2?
1
2?
2?
1
1?
2?
1
2?
2?

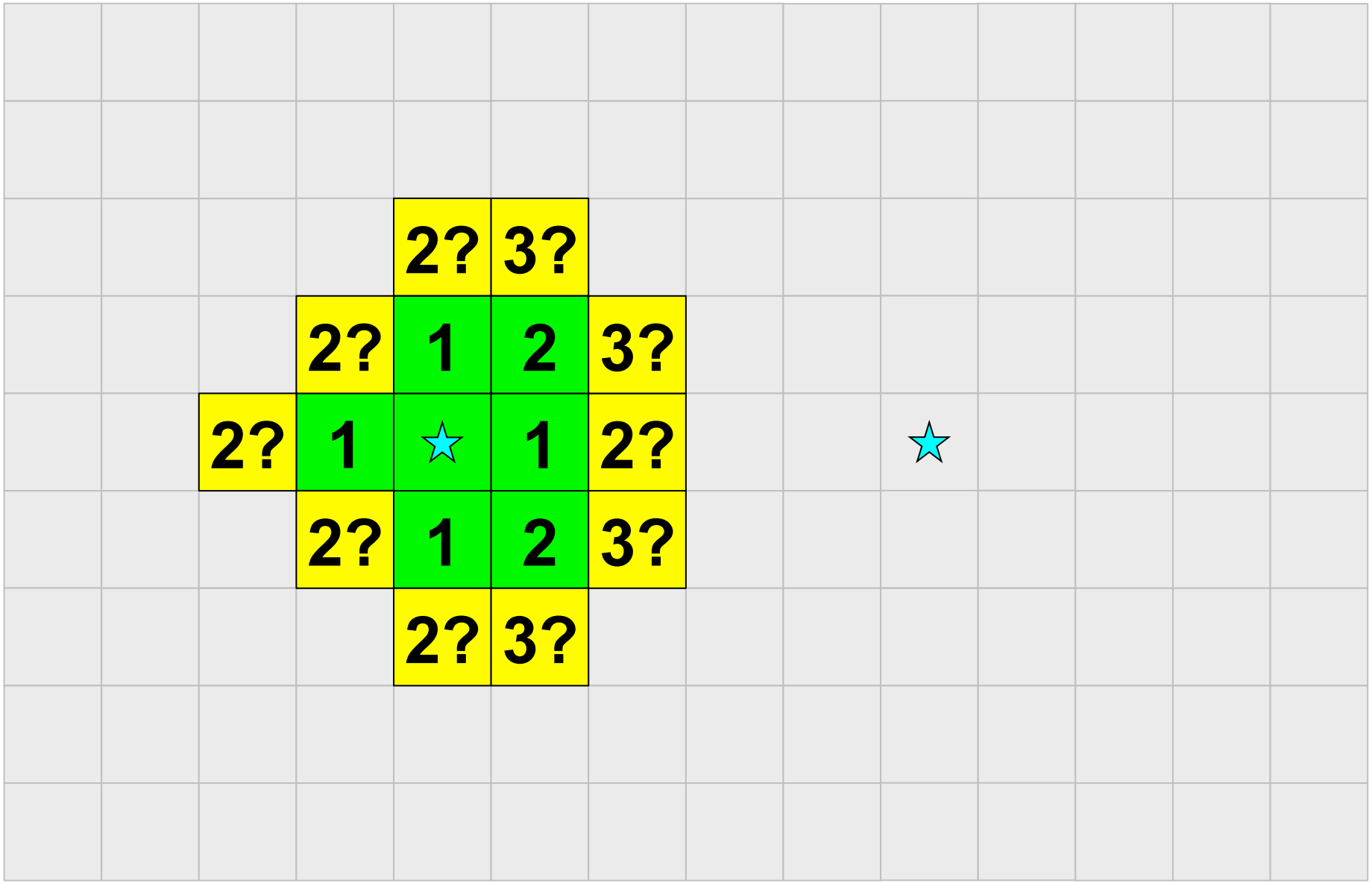

2?
2?
1
2?
2?
1
1
2?
1
2?
2?

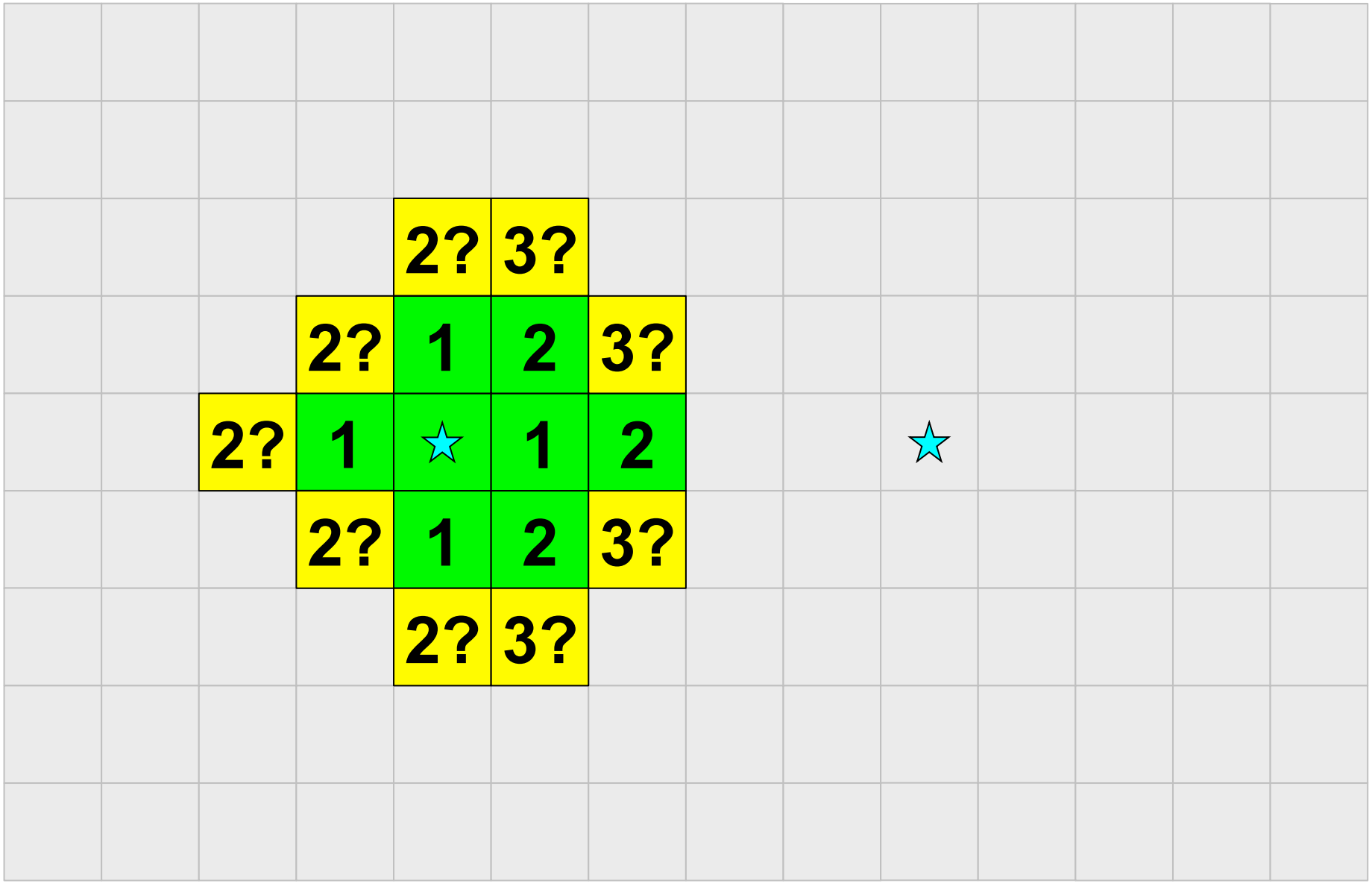

2?
2?
1
2?
2?
1
1
2?
2?
1
2?
2?

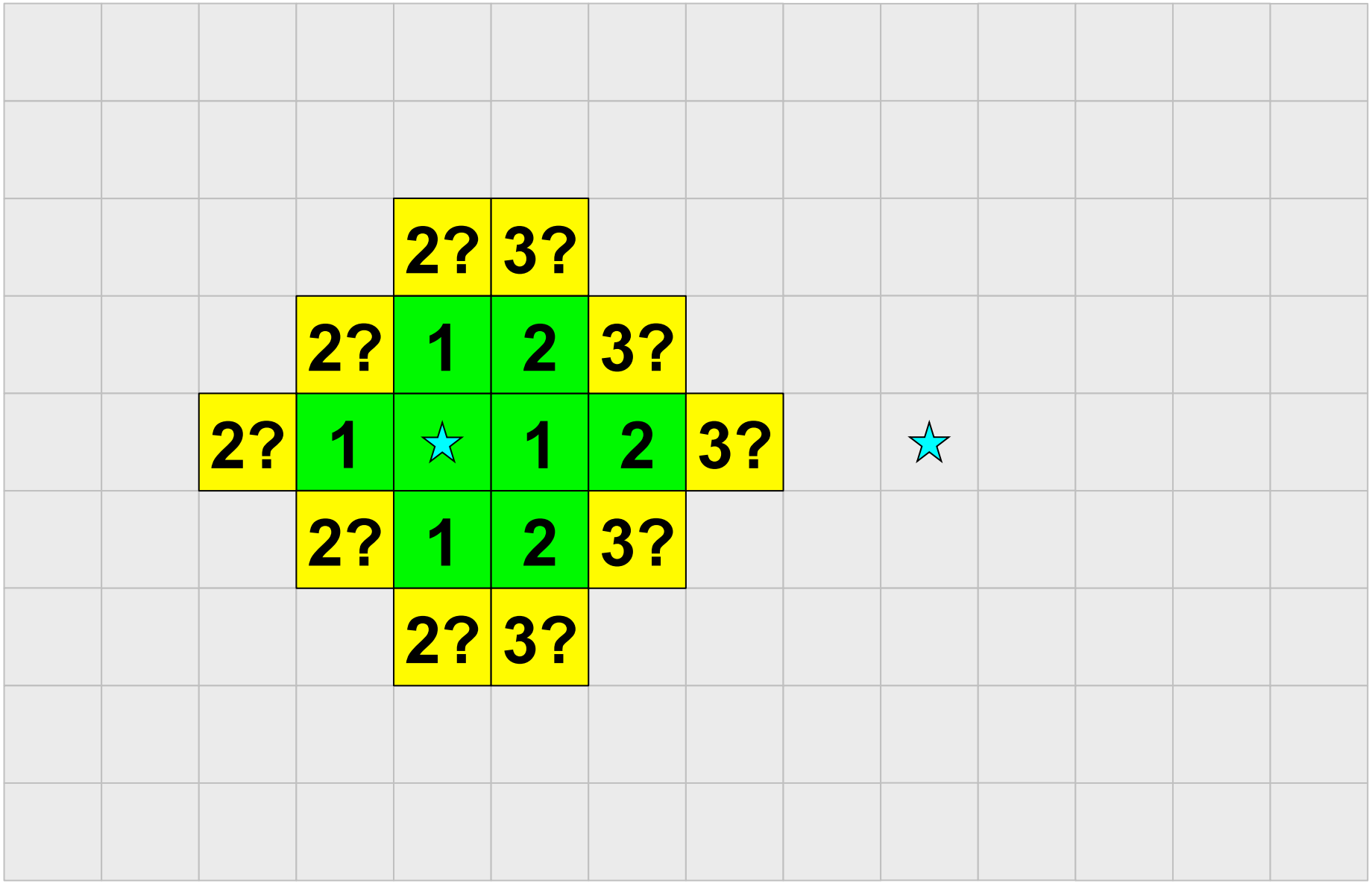

2?
2?
1
2
2?
1
1
2?
2?
1
2?
2?

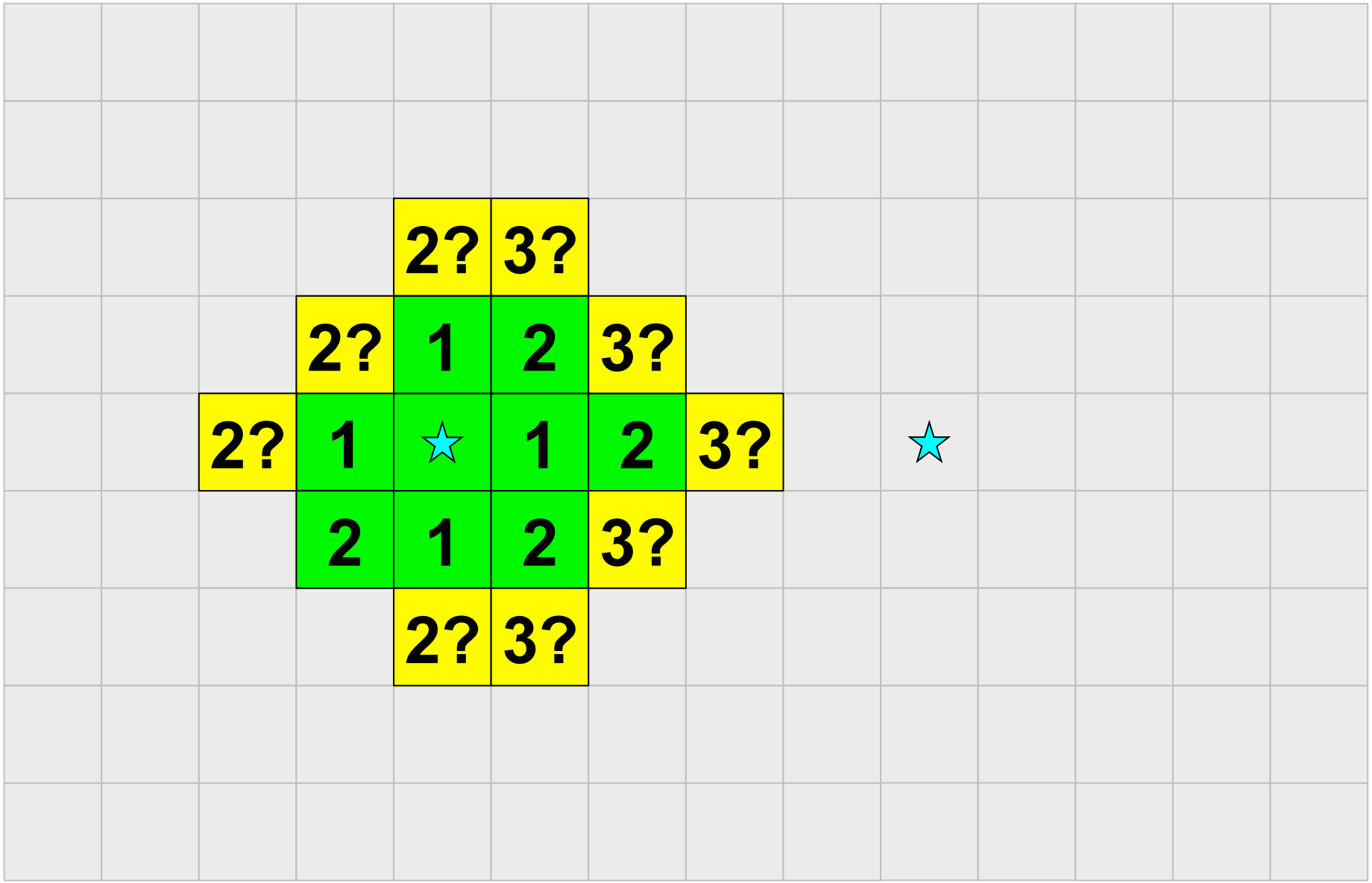

2?
3?
2?
1
2
3?
2?
1
1
2?
2?
1
2?
2?

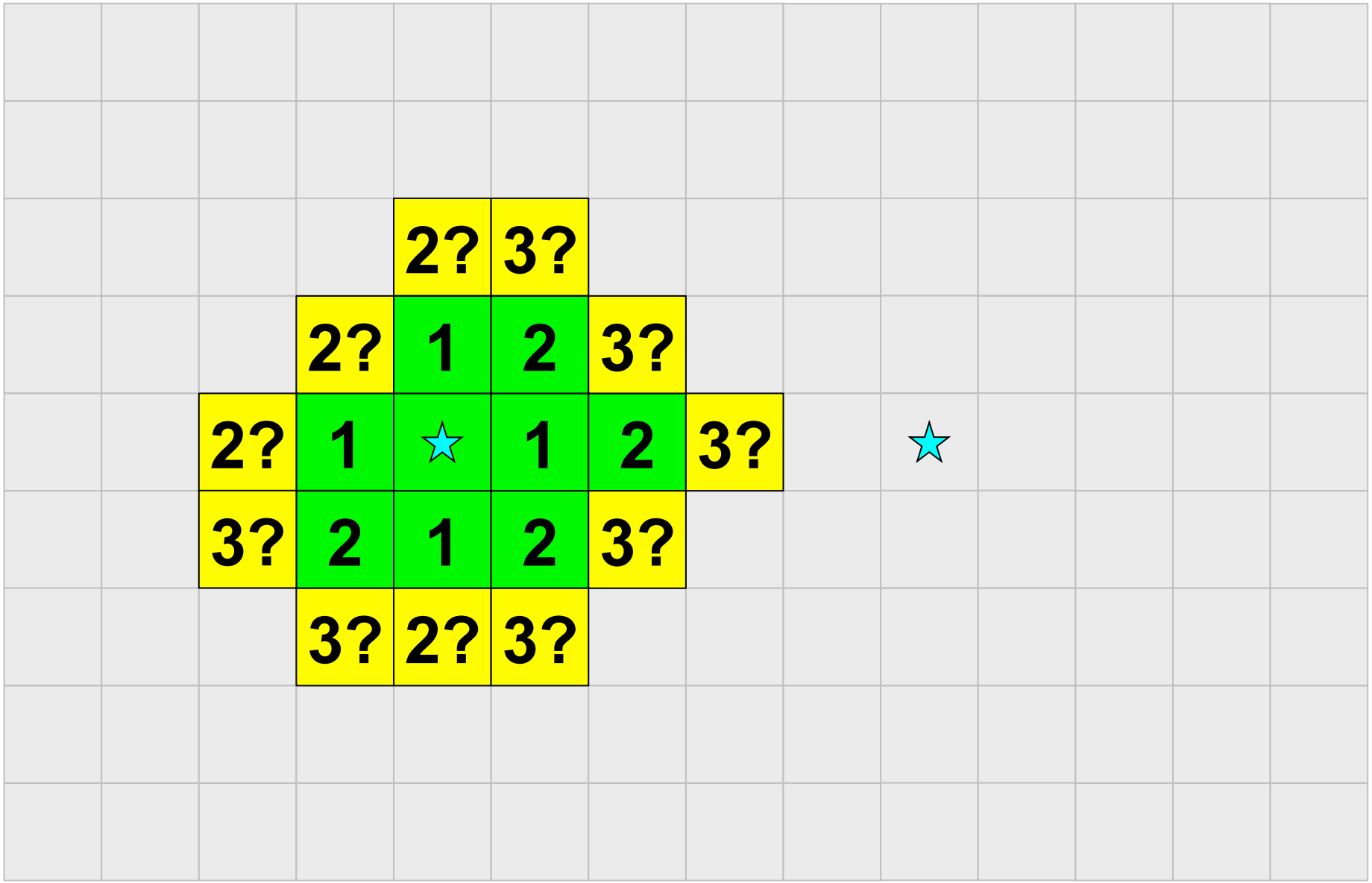

2?
3?
2?
1
2
3?
2?
1
1
2?
2?
1
2
2?

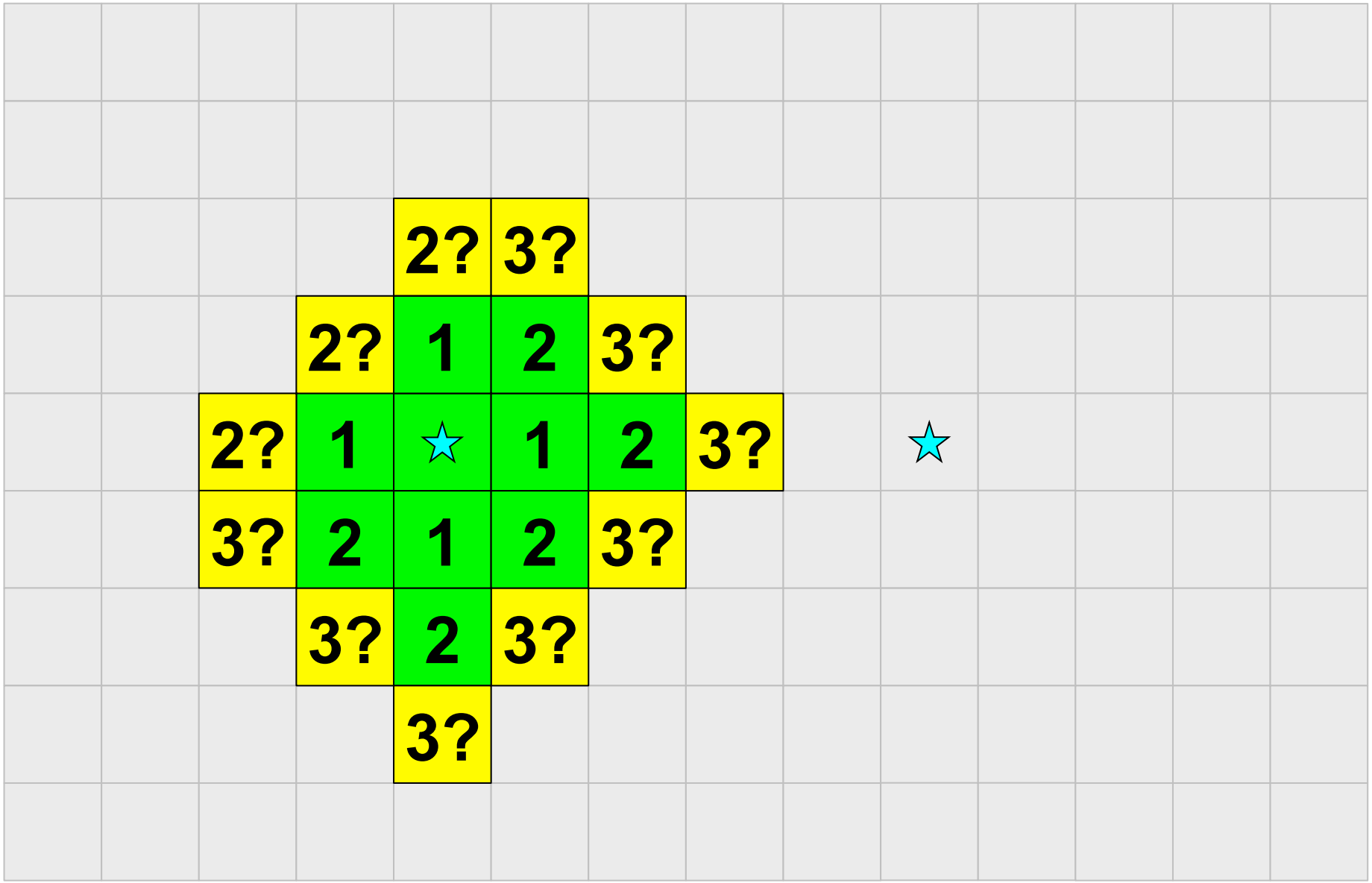

2? 3?
2? 1 2 3?
2? 1 1 2?
2? 1 2 3?
2? 3?

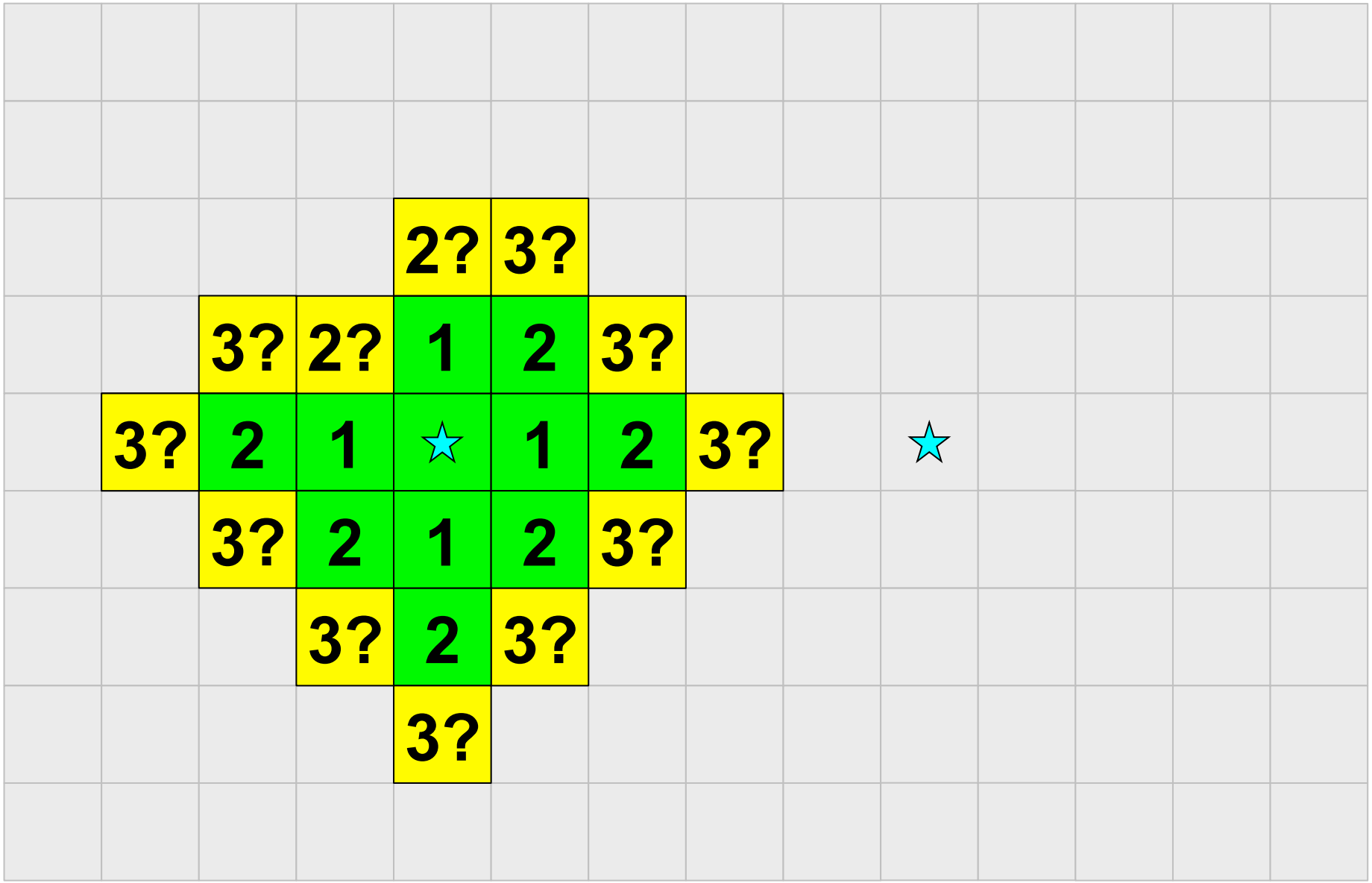

2? 3?
2? 1 2 3?
2? 1 1 2
2? 1 2 3?
2? 3?

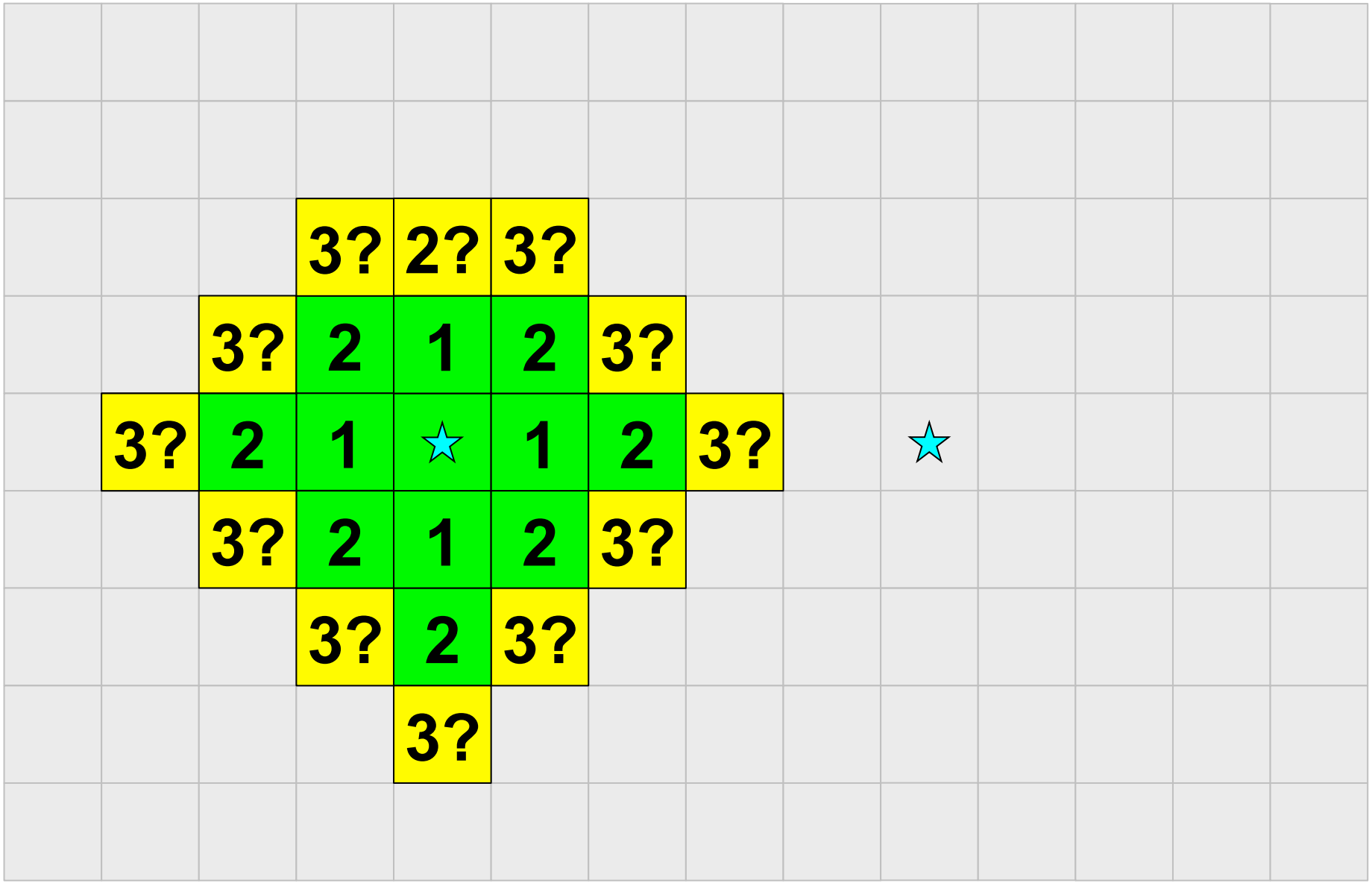

2? 3?
2? 1 2 3?
2? 1 1 2 3?
2? 1 2 3?
2? 3?

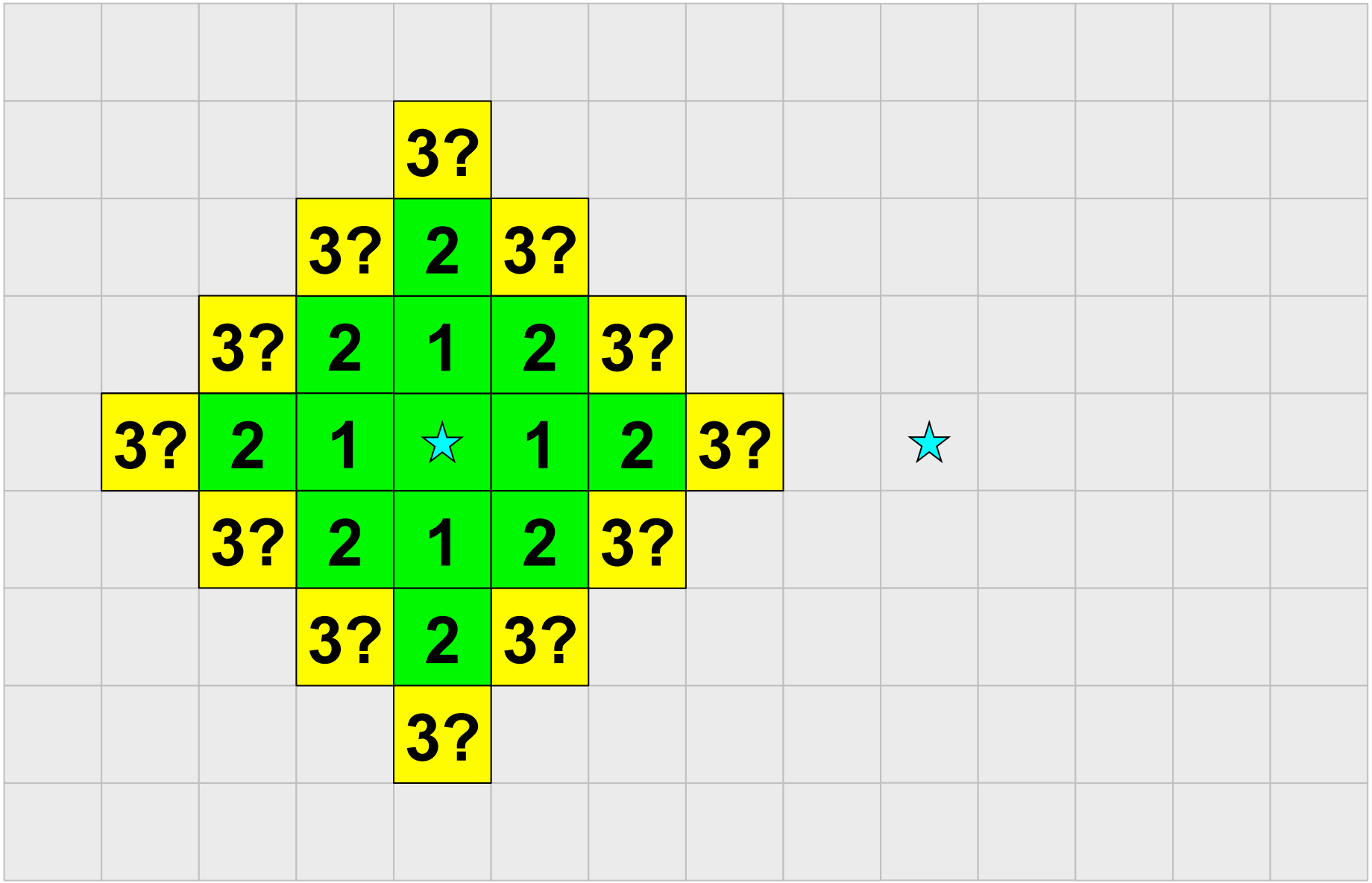

2? 3?
2? 1 2 3?
2? 1 1 2 3?
2 1 2 3?
2? 3?

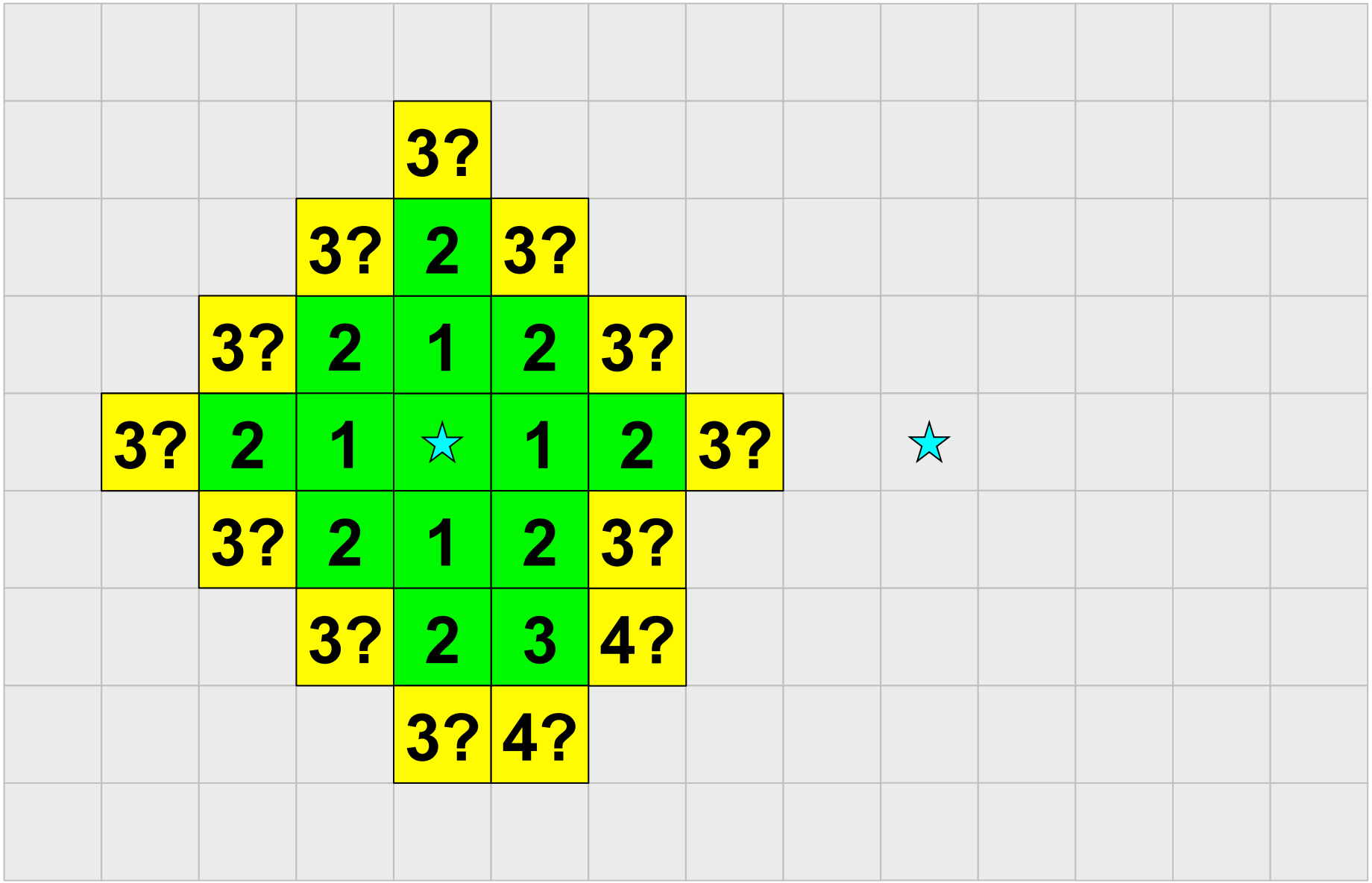

2? 3?
2? 1 2 3?
2? 1 1 2 3?
3? 2 1 2 3?
3? 2? 3?

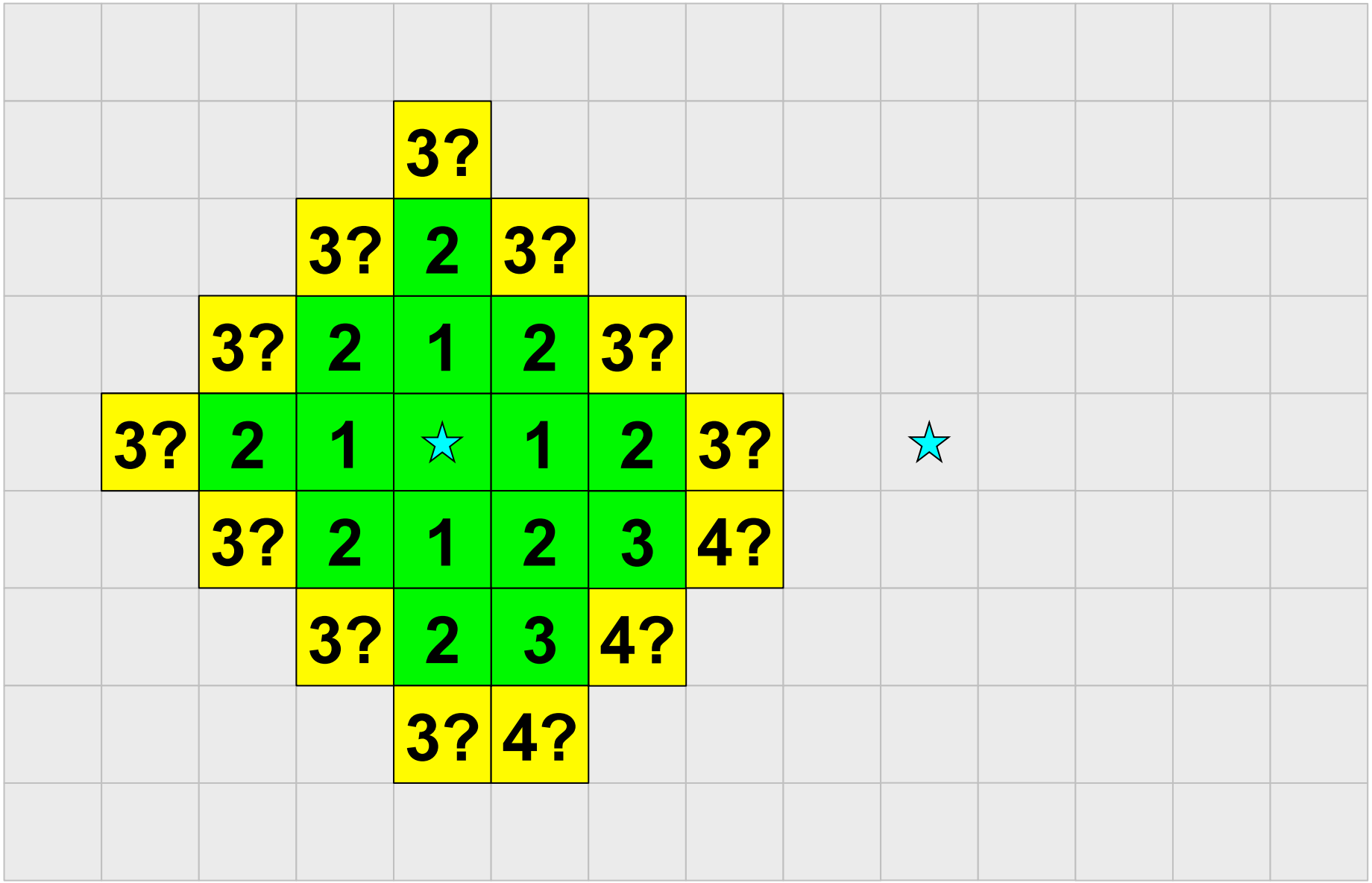

2? 3?
2? 1 2 3?
2? 1 1 2 3?
3? 2 1 2 3?
3? 2 3?
3?

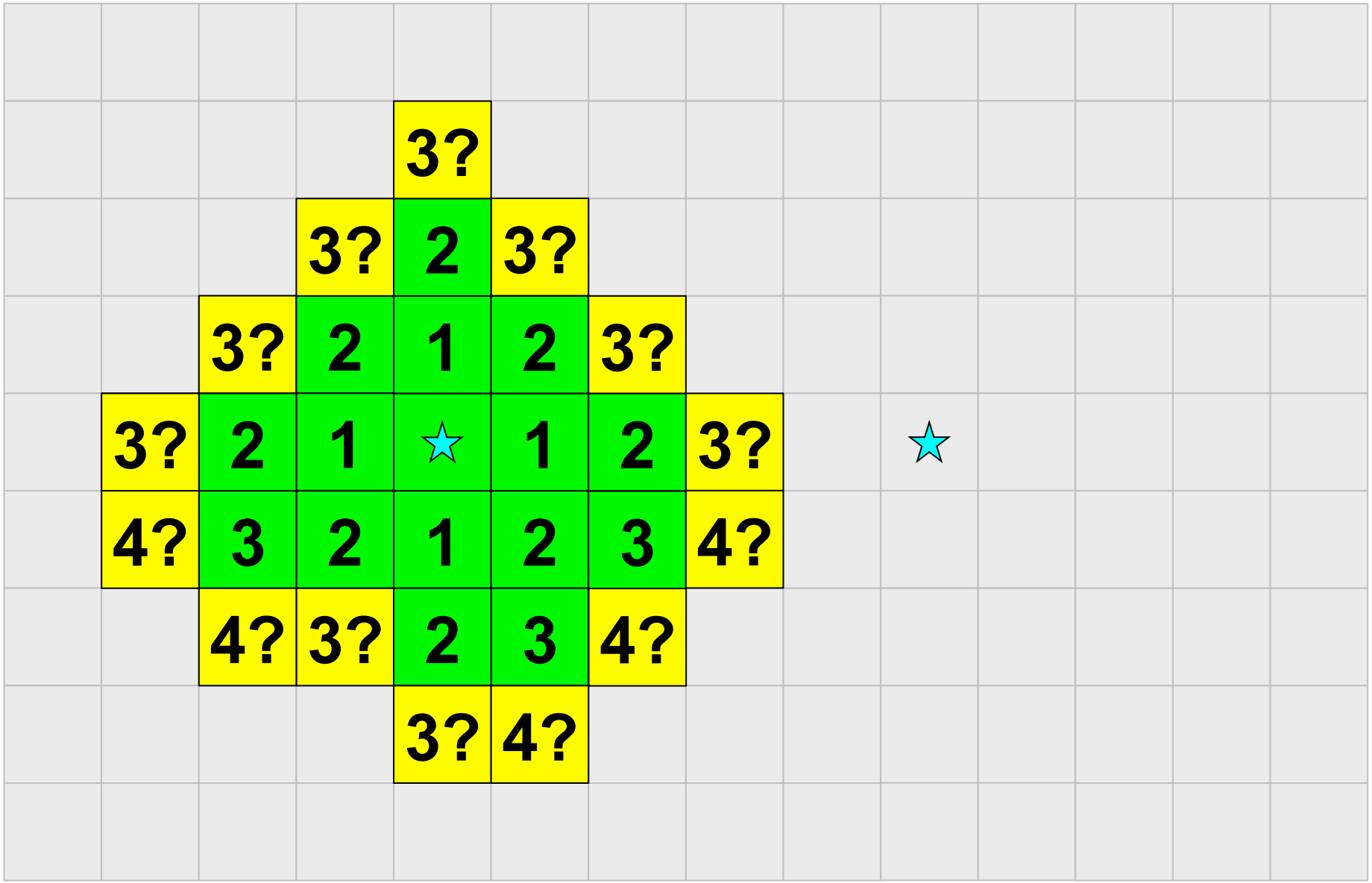

2? 3?
3? 2? 1 2 3?
3? 2 1 1 2 3?
3? 2 1 2 3?
3? 2 3?
3?

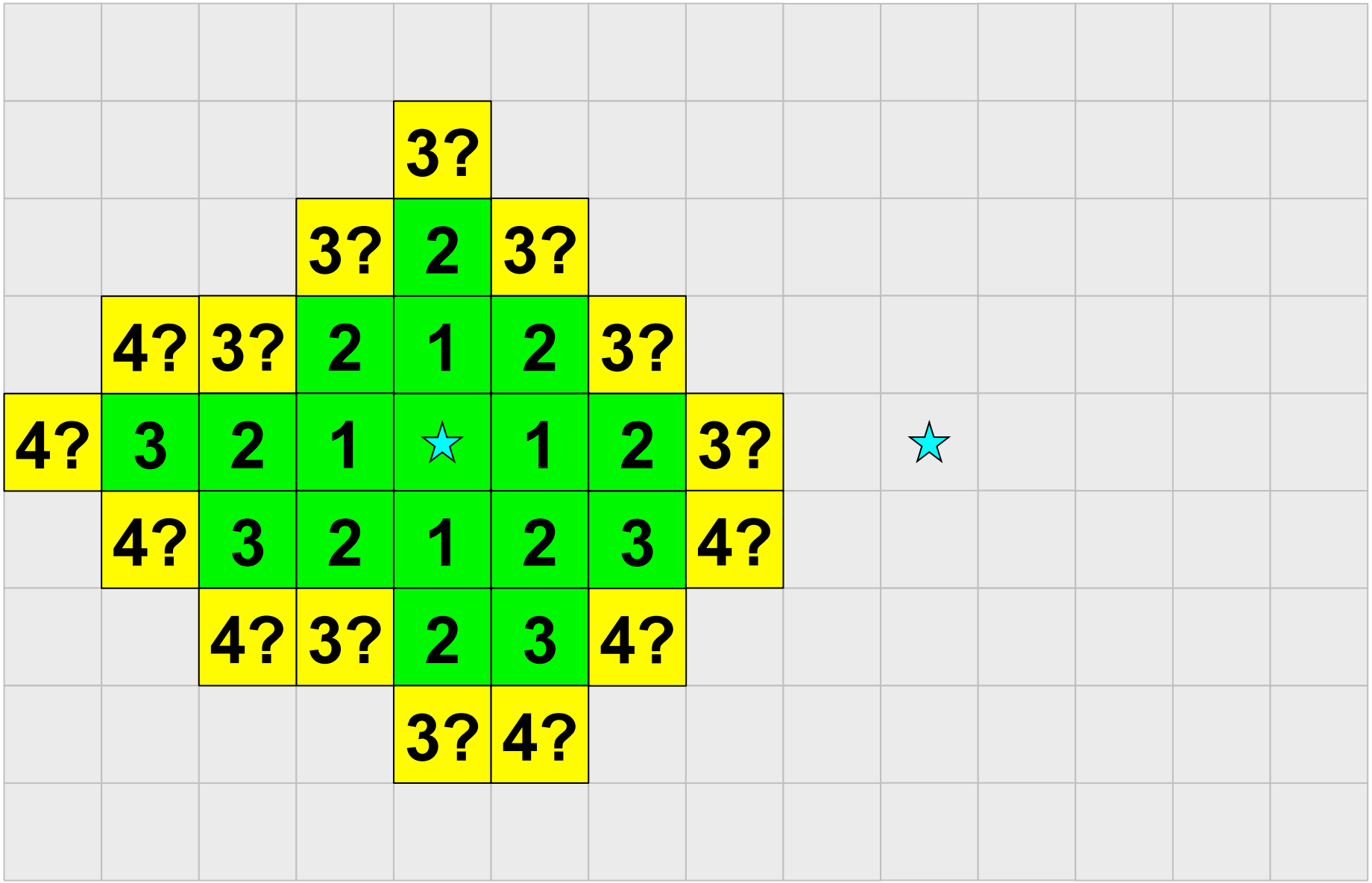

3? 2? 3?
3? 2 1 2 3?
3? 2 1 1 2 3?
3? 2 1 2 3?
3? 2 3?
3?

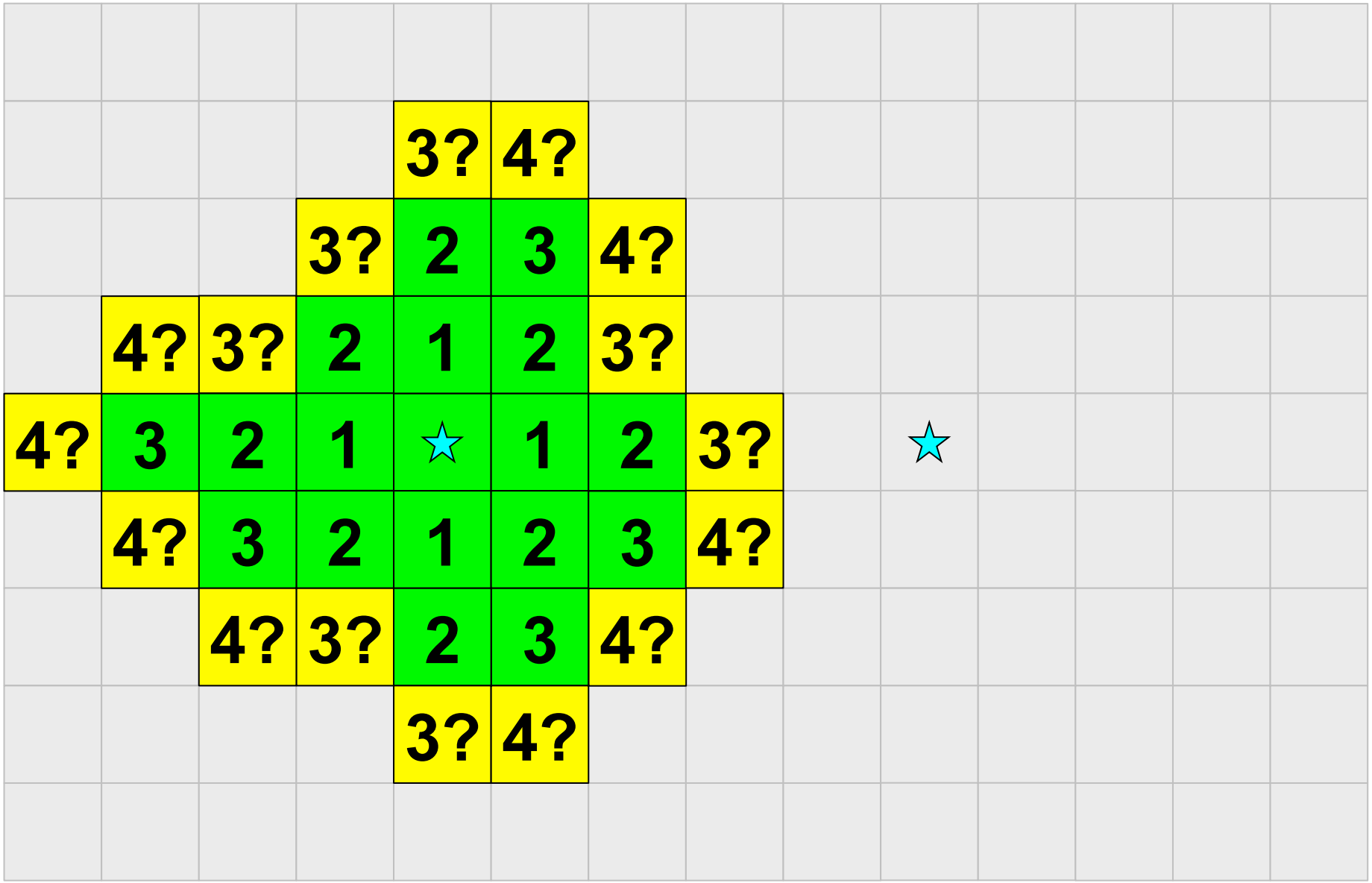

3?
3? 2 3?
3? 2 1 2 3?
3? 2 1 1 2 3?
3? 2 1 2 3?
3? 2 3?
3?

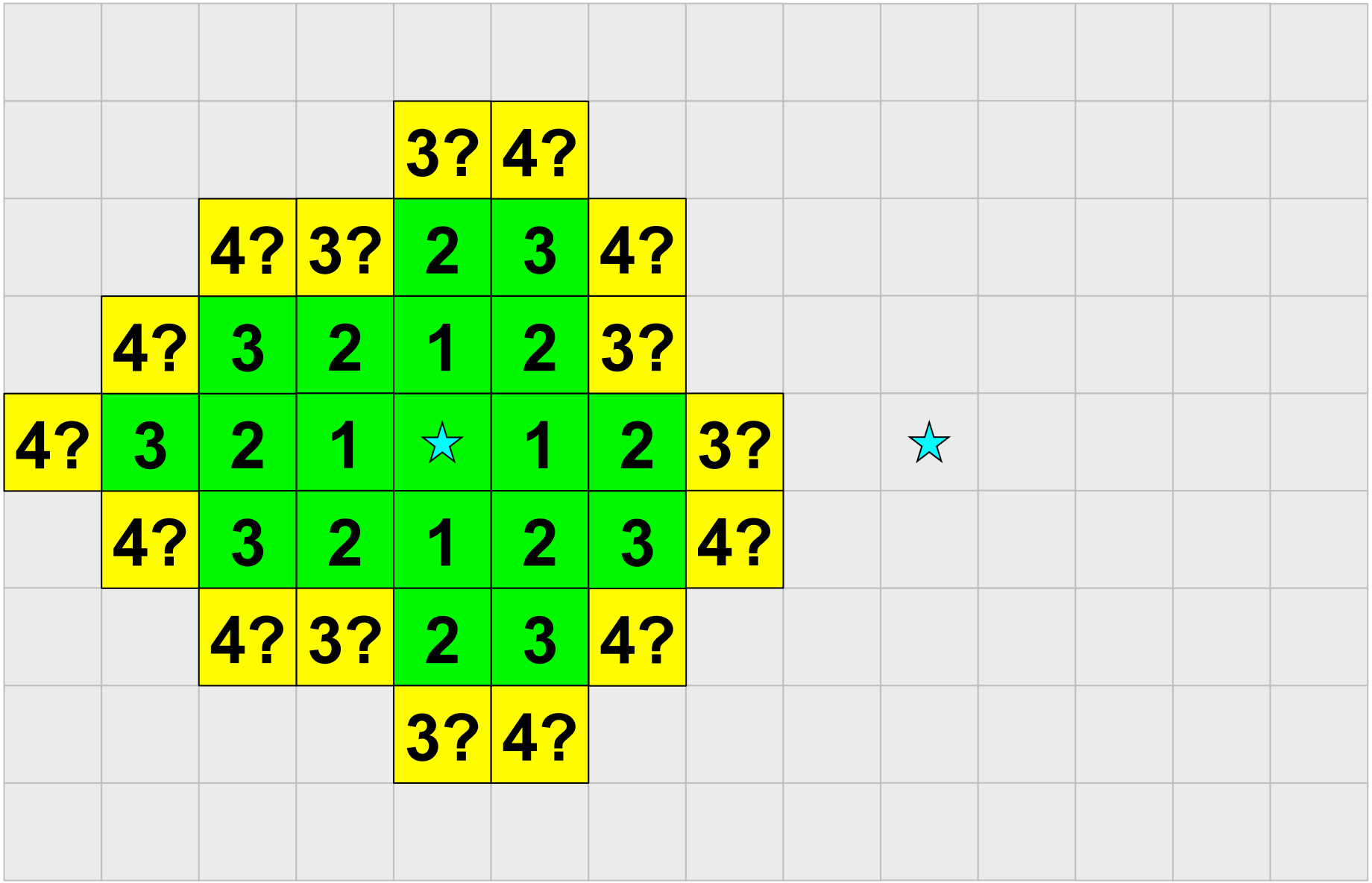

3?
3? 2 3?
3? 2 1 2 3?
3? 2 1 1 2 3?
3? 2 1 2 3?
3? 2 3 4?
3? 4?

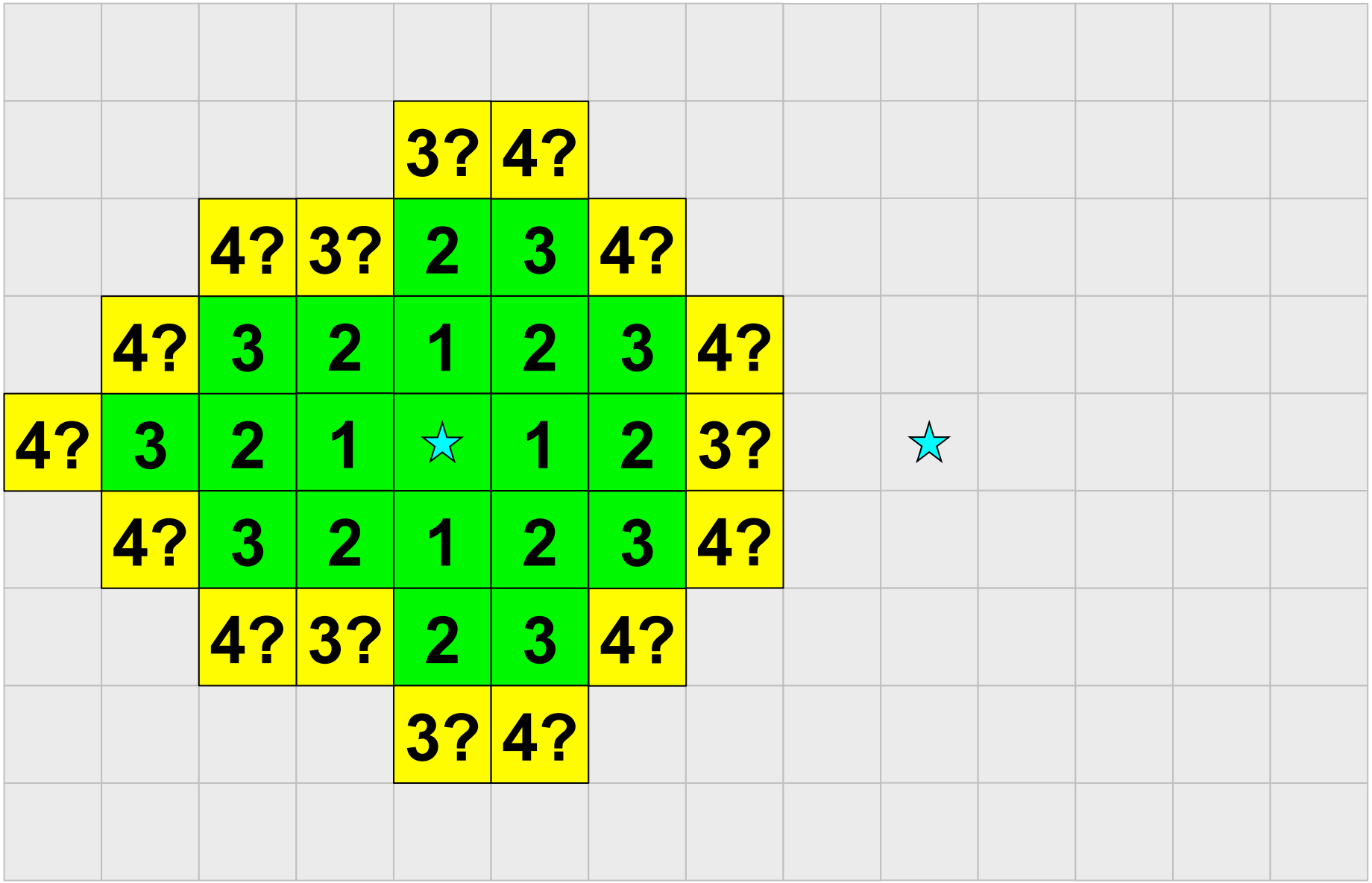

3?
3? 2 3?
3? 2 1 2 3?
3? 2 1 1 2 3?
3? 2 1 2 3 4?
3? 2 3 4?
3? 4?

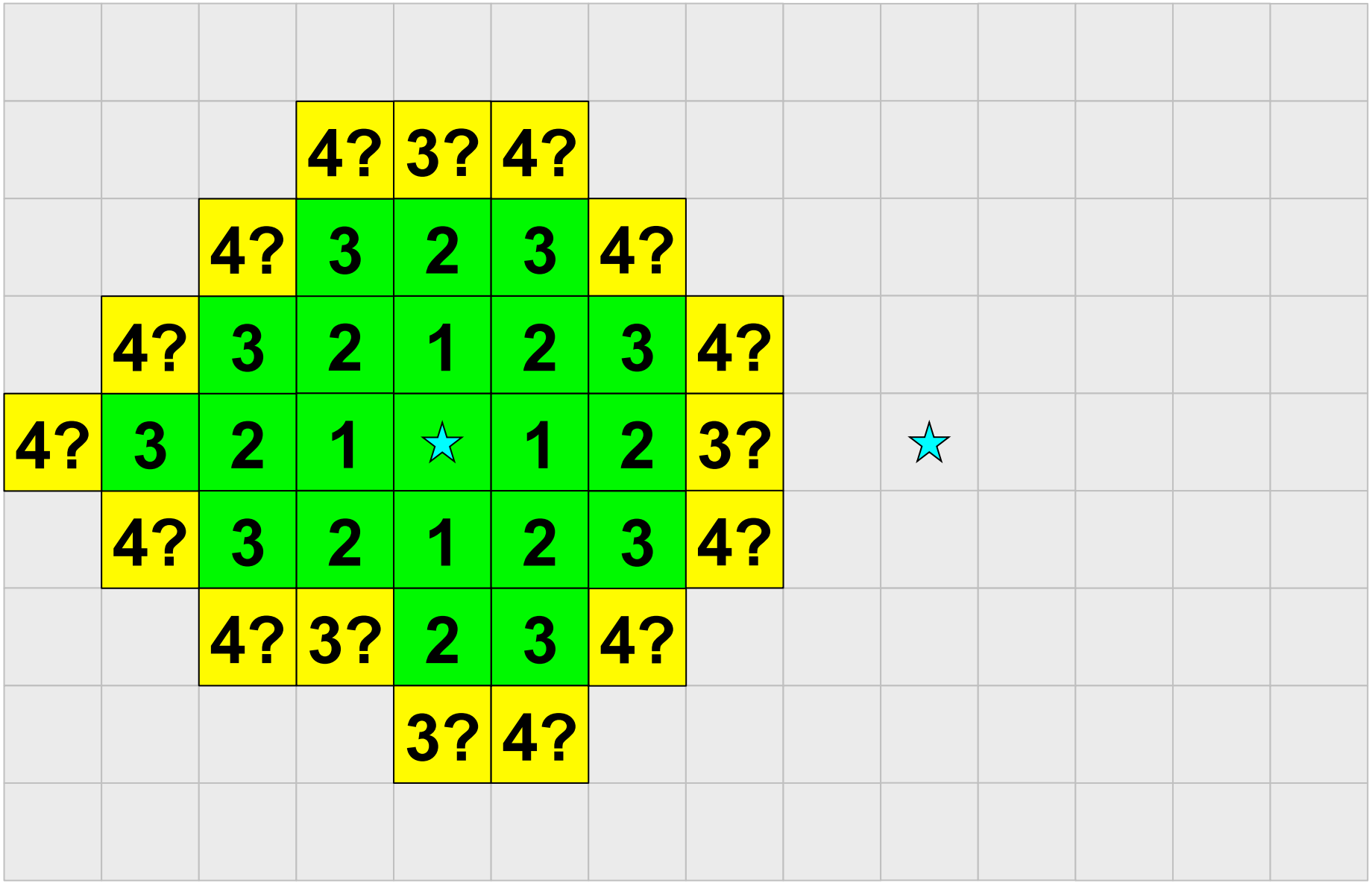

3?
3? 2 3?
3? 2 1 2 3?
3? 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

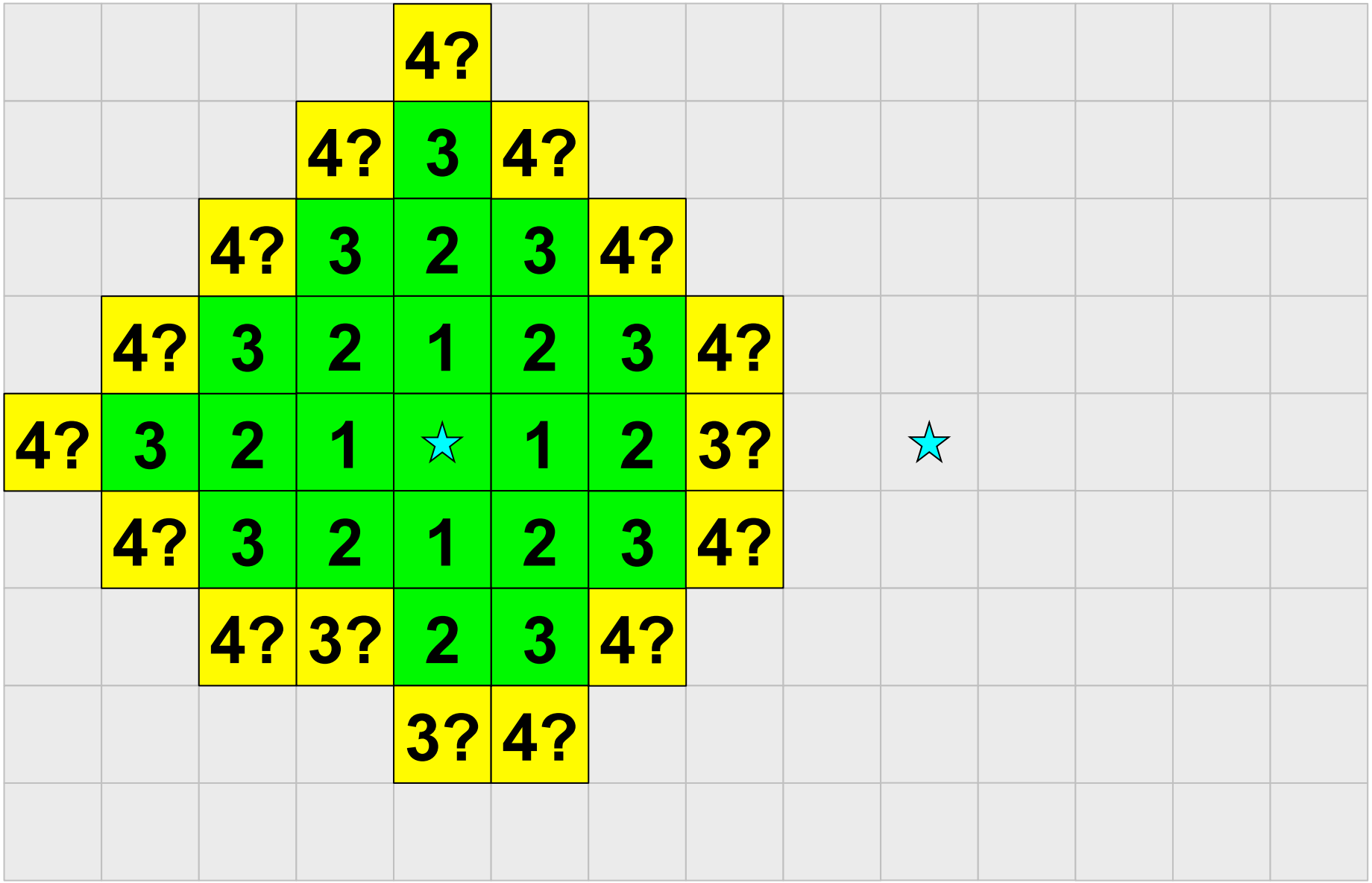

3?
3? 2 3?
4? 3? 2 1 2 3?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

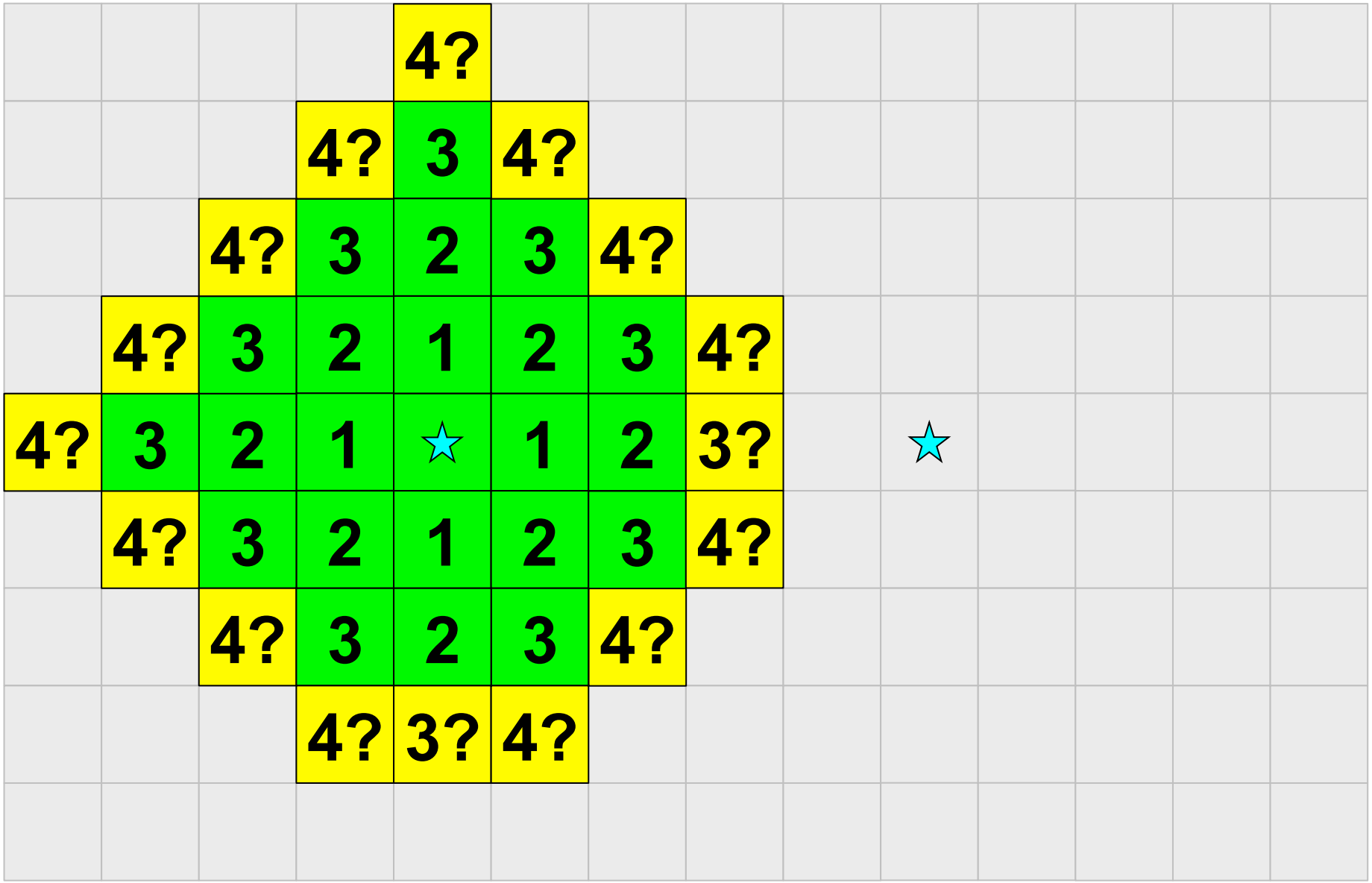

3? 4?
3? 2 3 4?
4? 3? 2 1 2 3?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

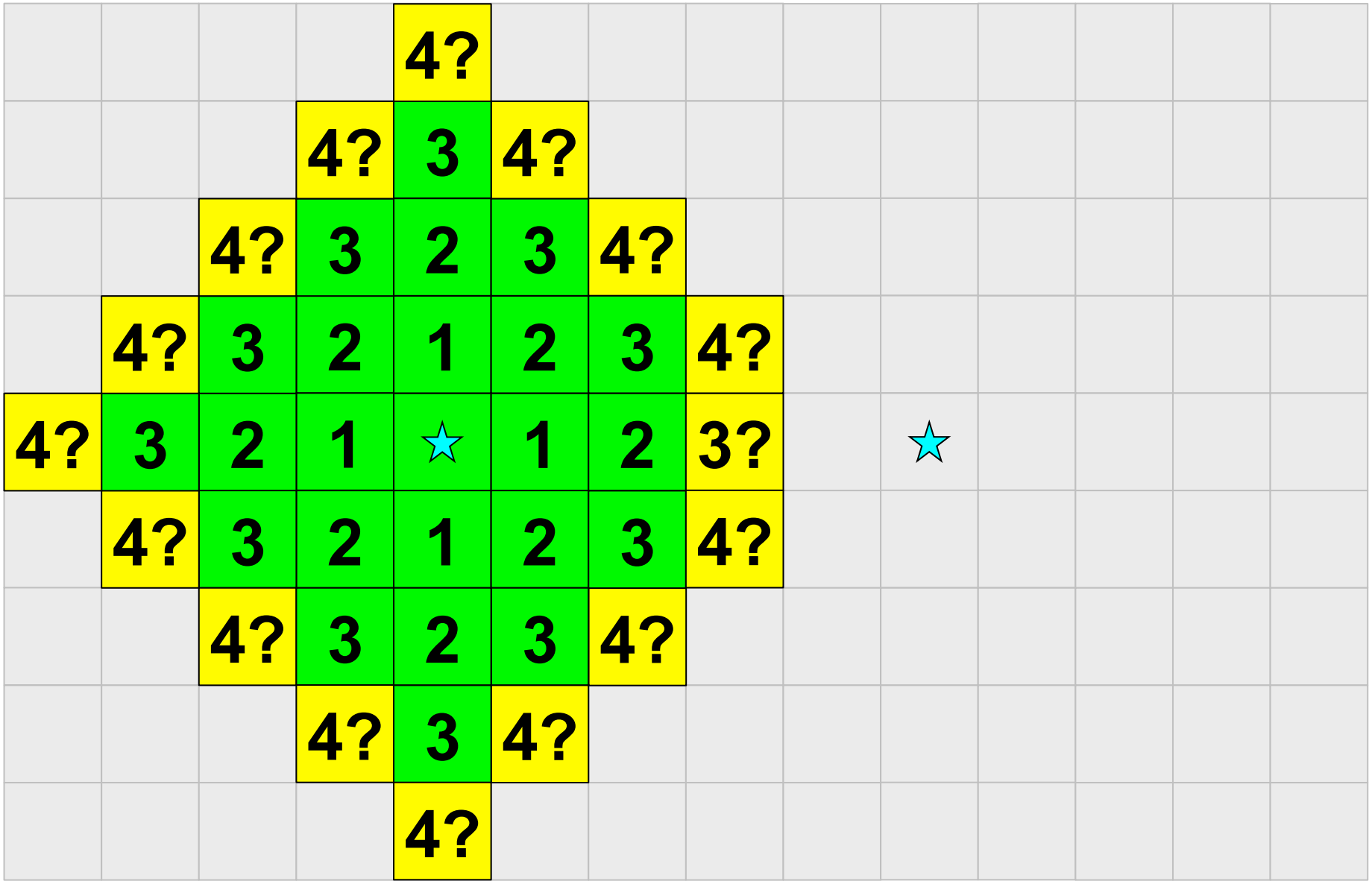

3? 4?
4? 3? 2 3 4?
4? 3 2 1 2 3?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

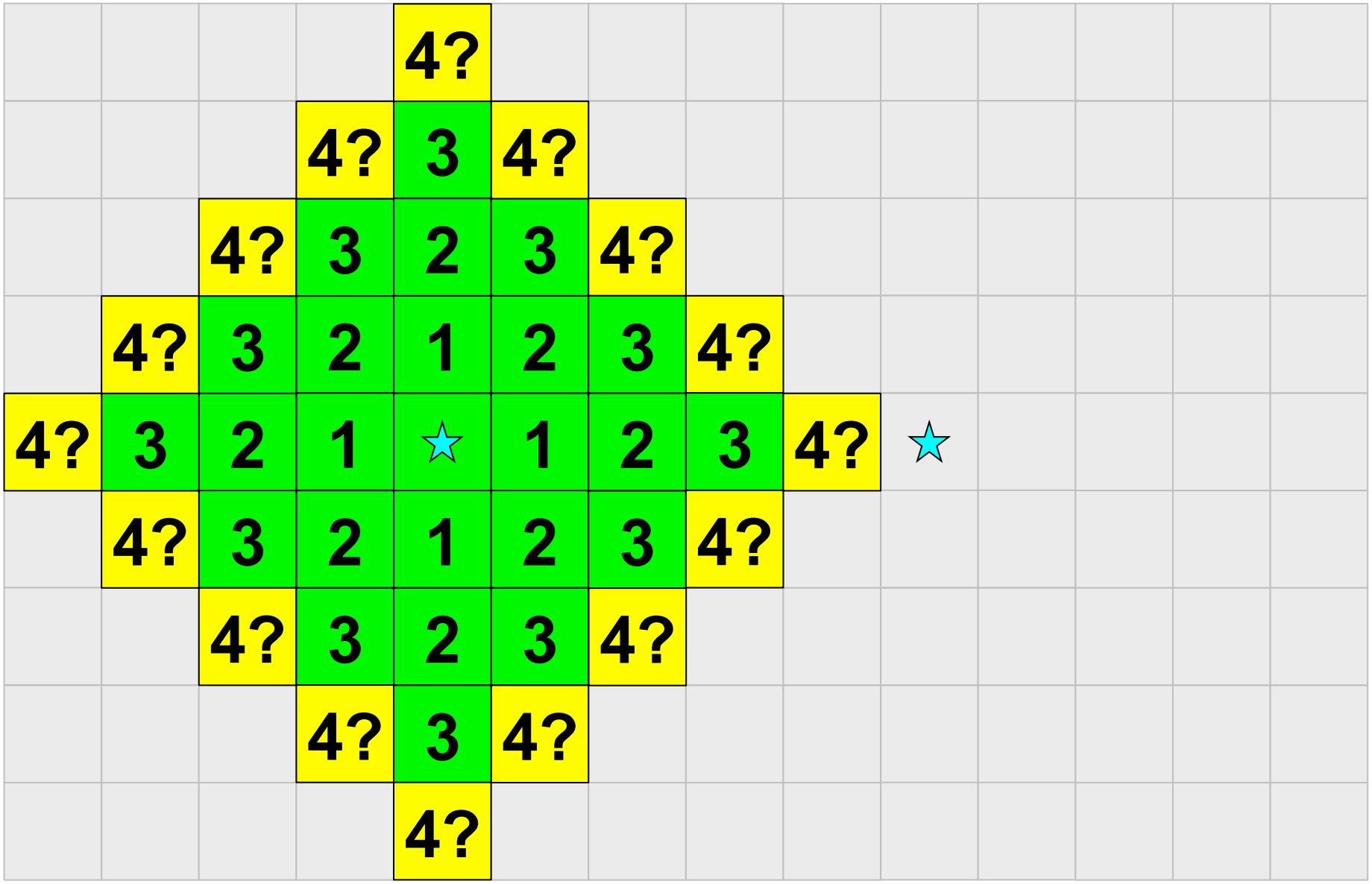

3? 4?
4? 3? 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

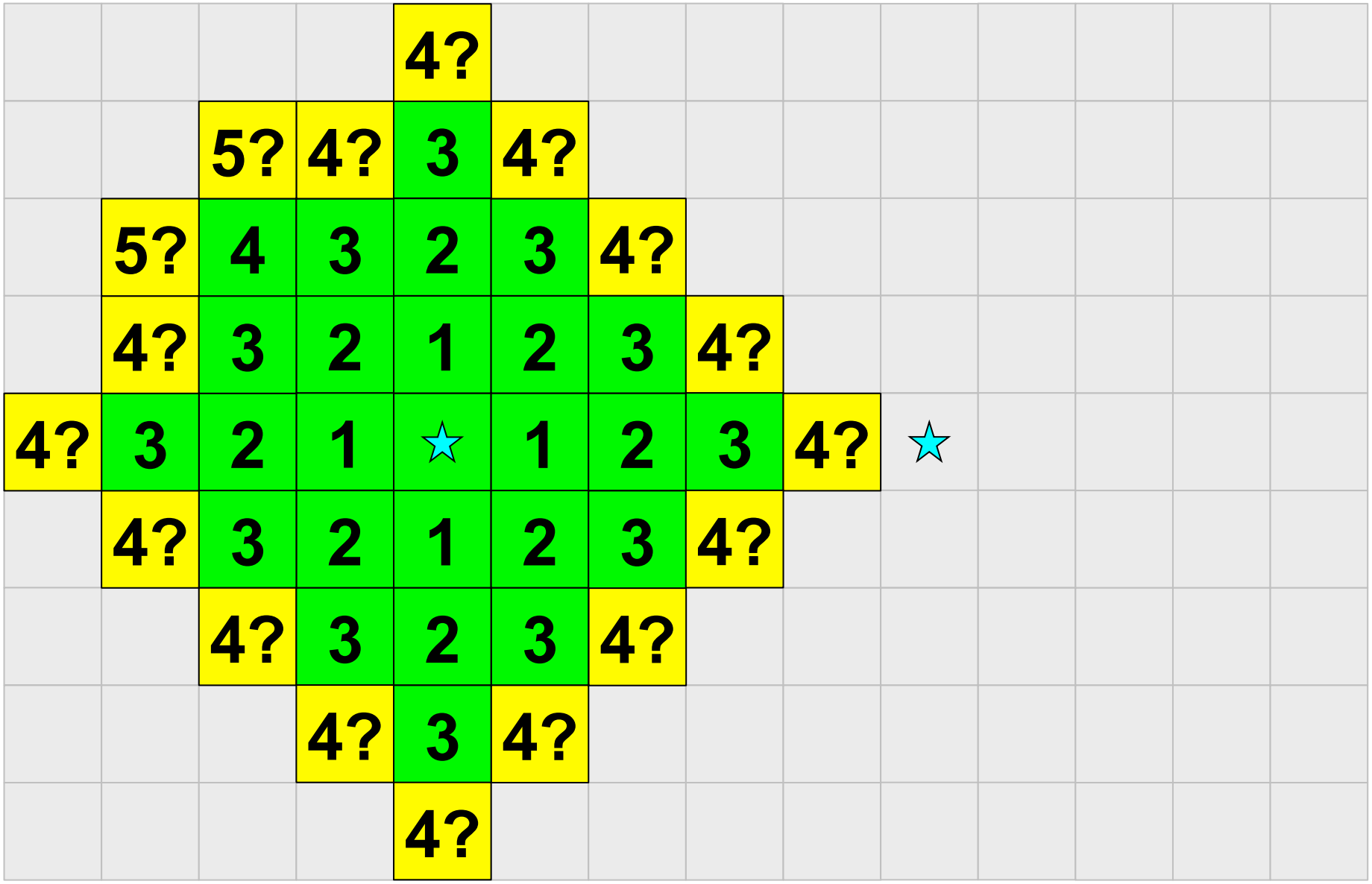

4? 3? 4?
4? 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

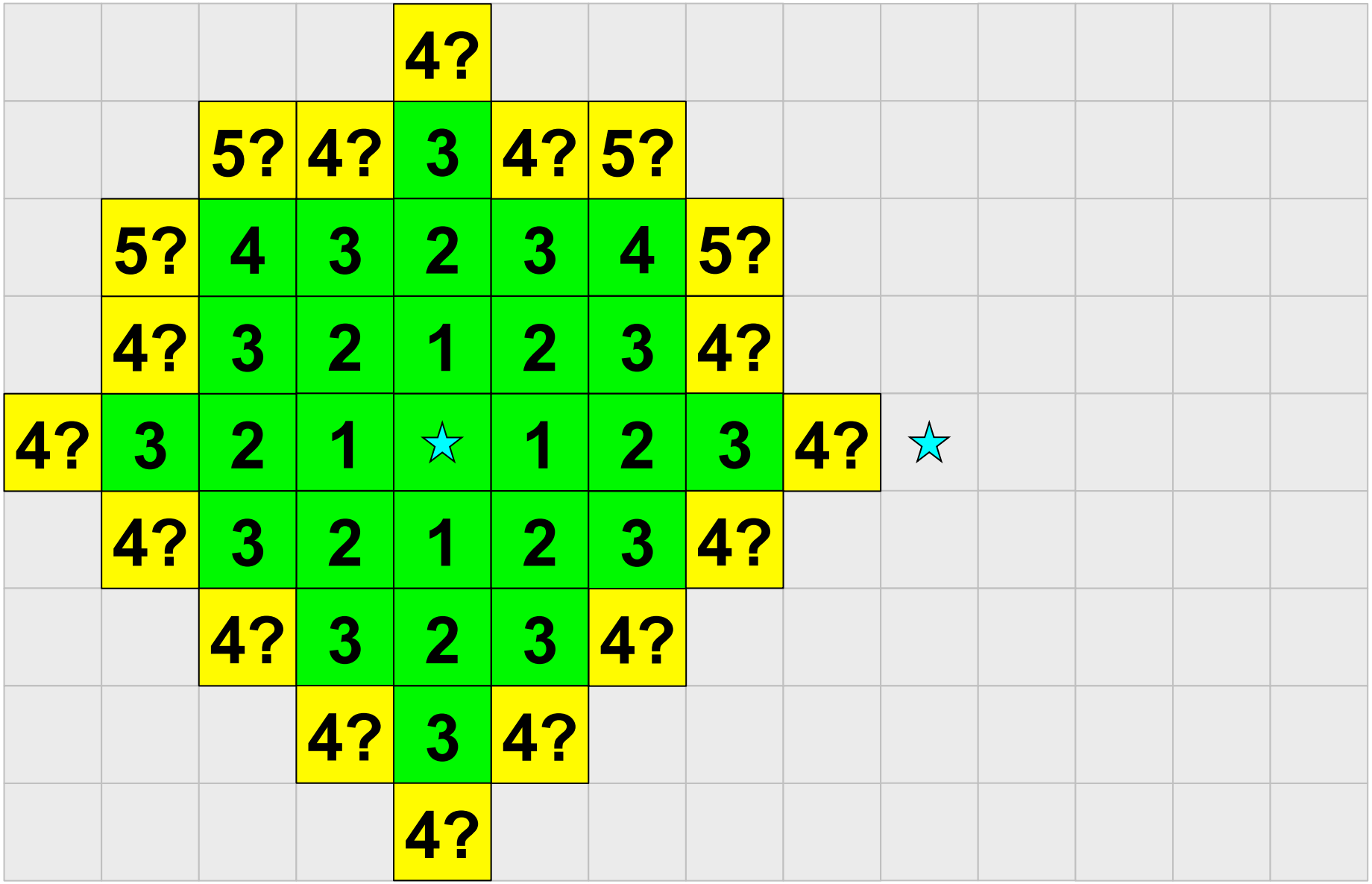

4?
4? 3 4?
4? 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

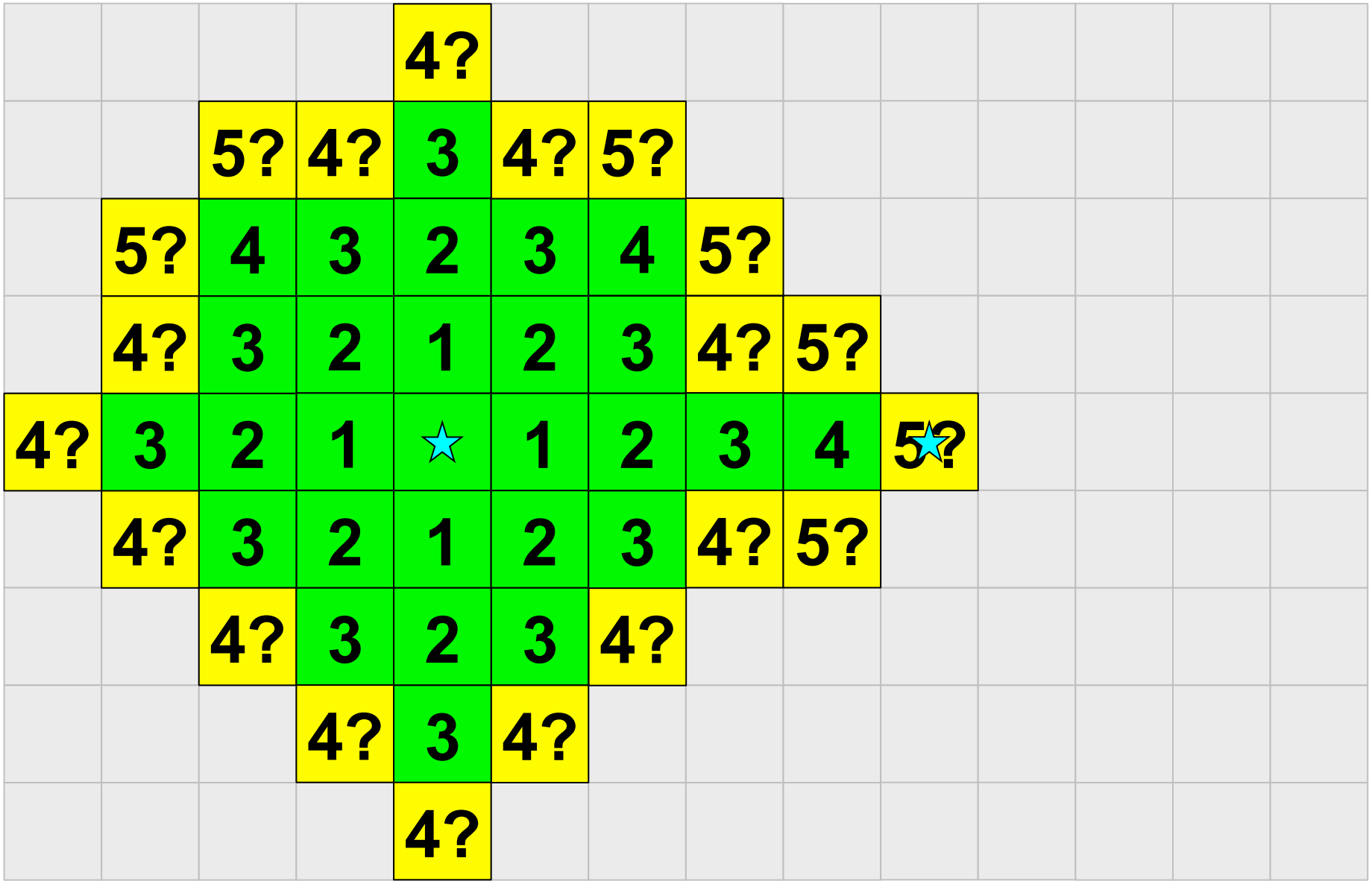

4?
4? 3 4?
4? 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3 2 3 4?
4? 3? 4?

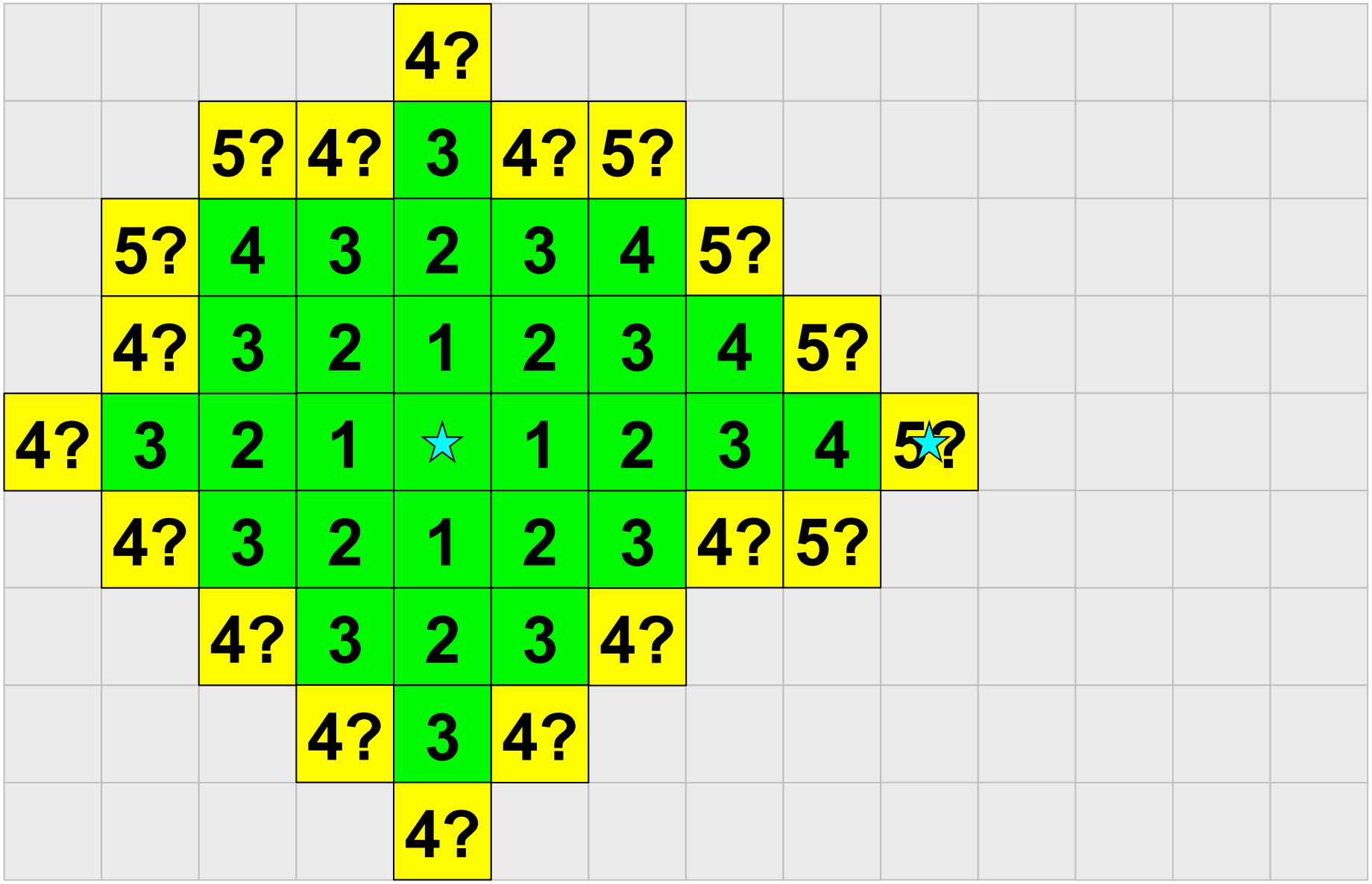

4?
4? 3 4?
4? 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3 2 3 4?
4? 3 4?
4?

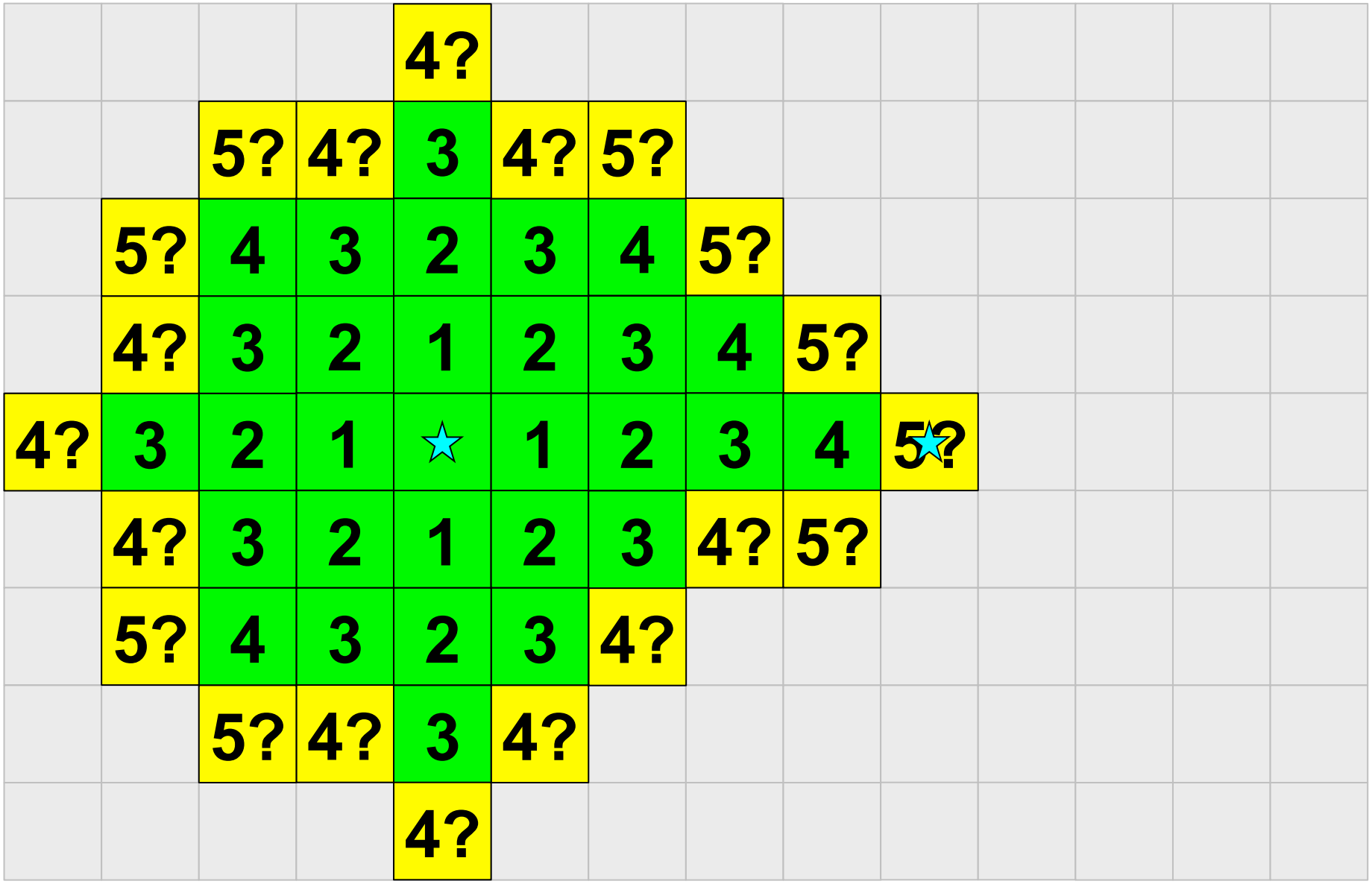

4?
4? 3 4?
4? 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 3 4?
4? 3 4?
4?

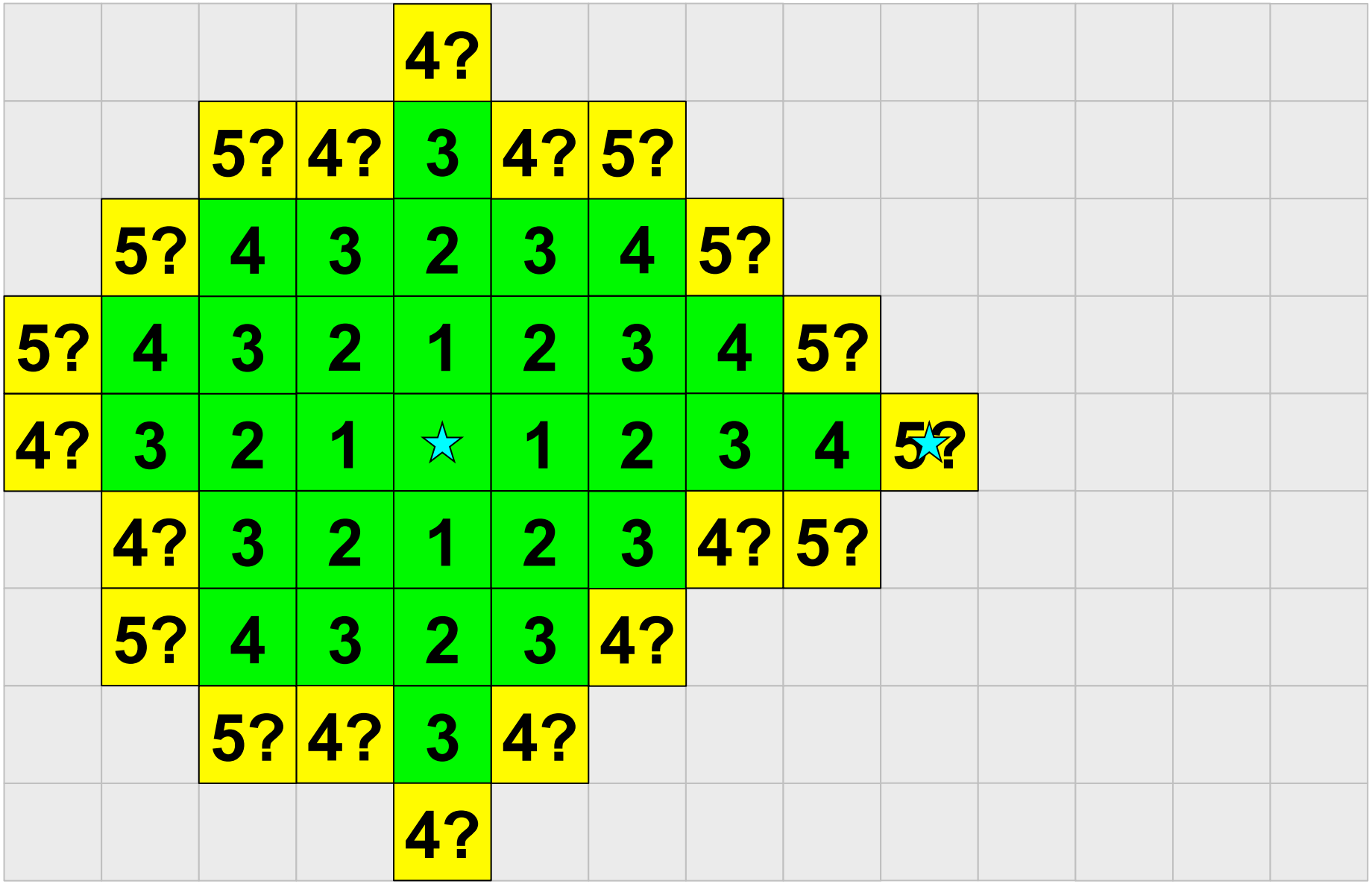

4?
5? 4? 3 4?
5? 4 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 3 4?
4? 3 4?
4?

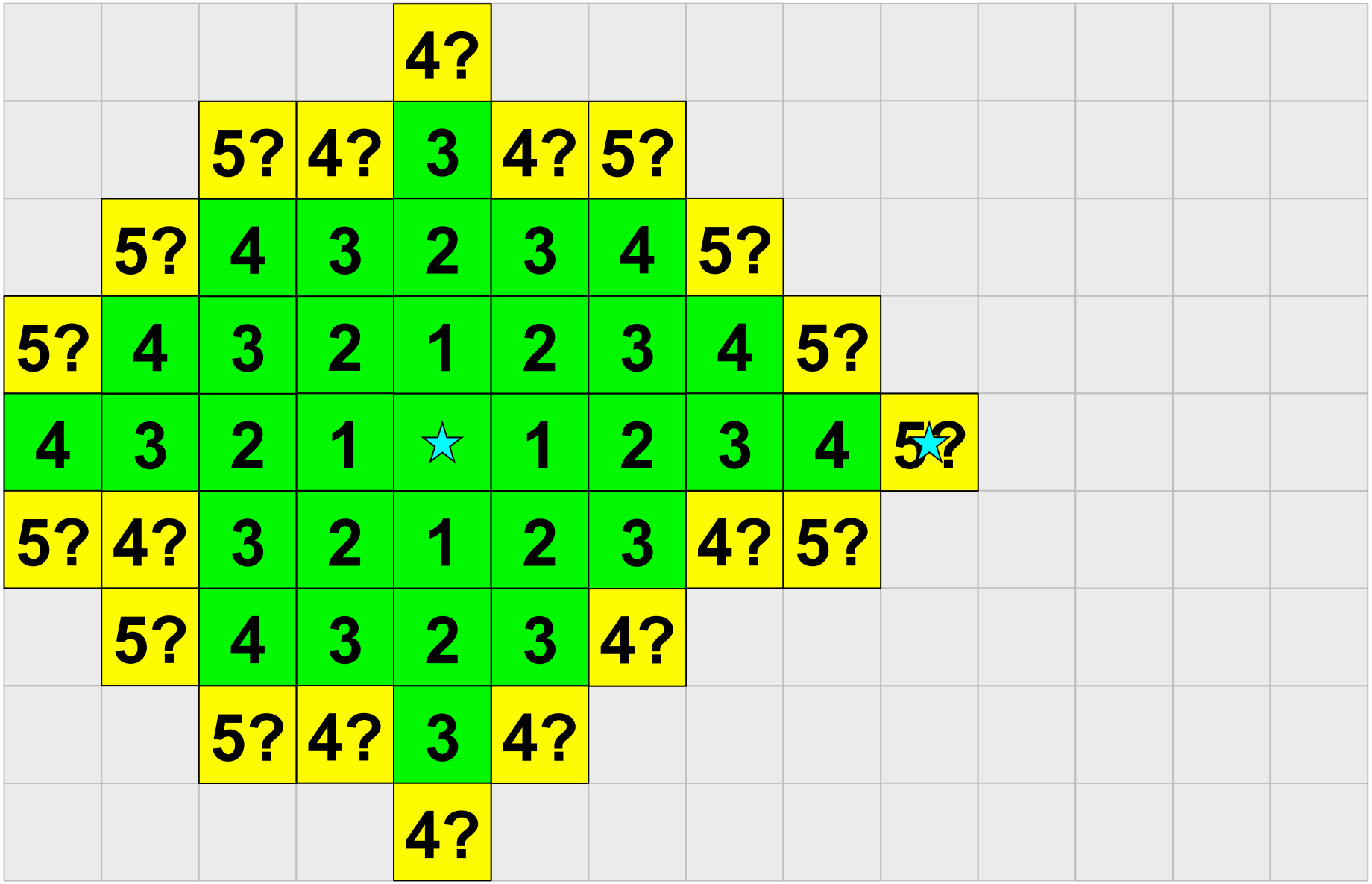

4?
5? 4? 3 4? 5?
5? 4 3 2 3 4 5?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 3 4?
4? 3 4?
4?

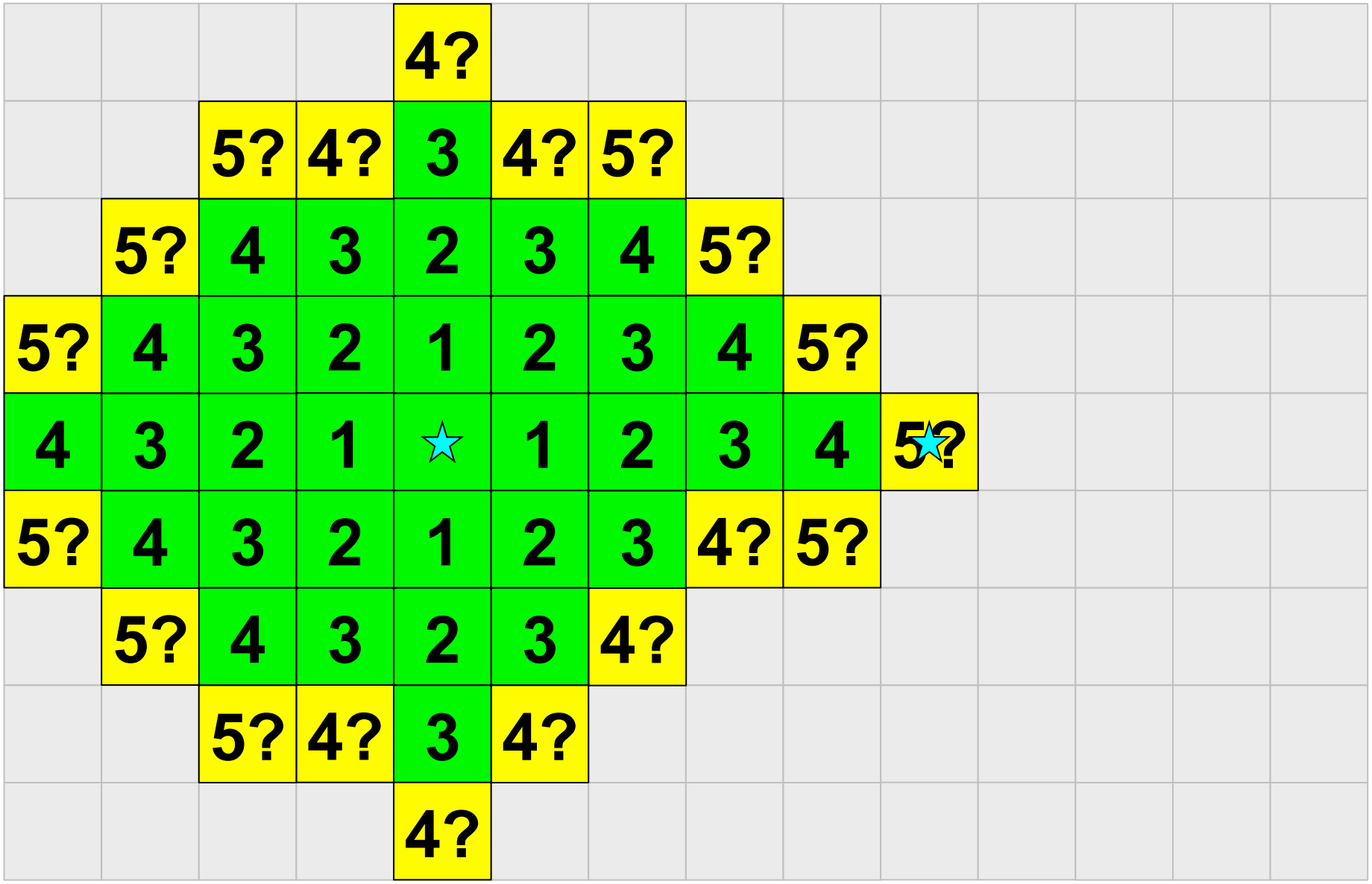

4?
5? 4? 3 4? 5?
5? 4 3 2 3 4 5?
4? 3 2 1 2 3 4? 5?
4? 3 2 1 1 2 3 4 5?
4? 3 2 1 2 3 4? 5?
4? 3 2 3 4?
4? 3 4?
4?

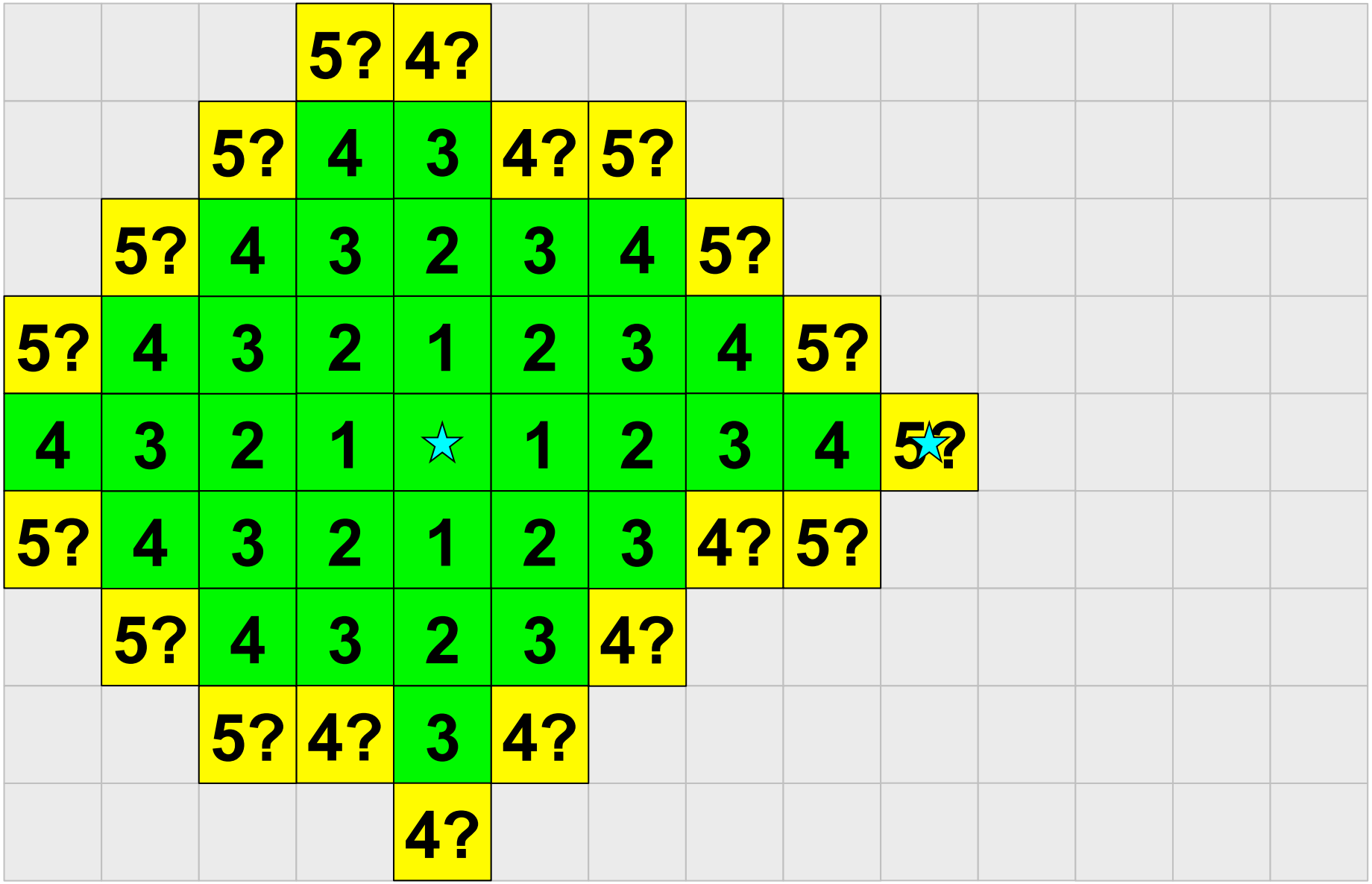

4?
5? 4? 3 4? 5?
5? 4 3 2 3 4 5?
4? 3 2 1 2 3 4 5?
4? 3 2 1 1 2 3 4 5?
4? 3 2 1 2 3 4? 5?
4? 3 2 3 4?
4? 3 4?
4?

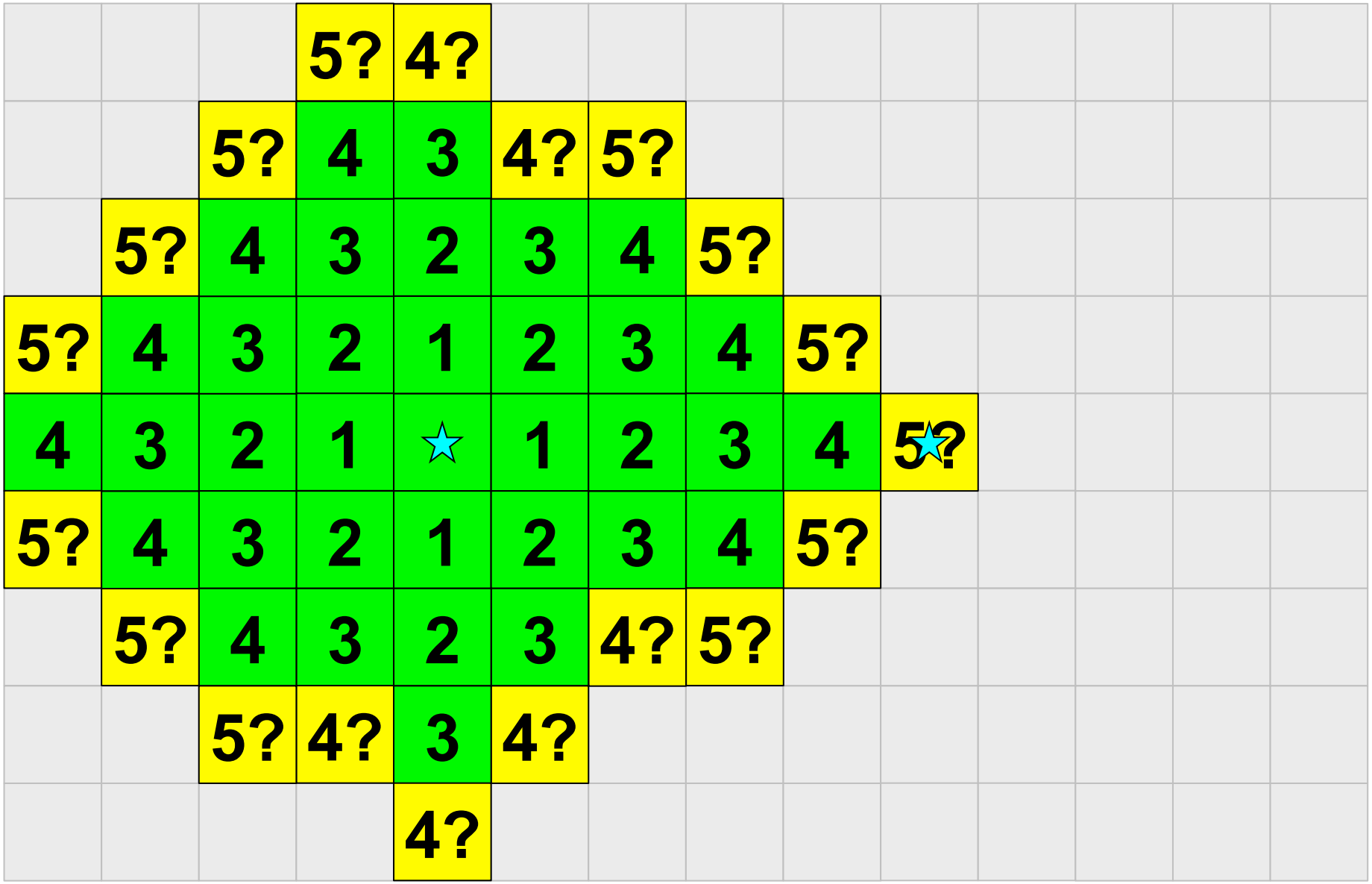

4?
5? 4? 3 4? 5?
5? 4 3 2 3 4 5?
4? 3 2 1 2 3 4 5?
4? 3 2 1 1 2 3 4 5?
4? 3 2 1 2 3 4? 5?
5? 4 3 2 3 4?
5? 4? 3 4?
4?

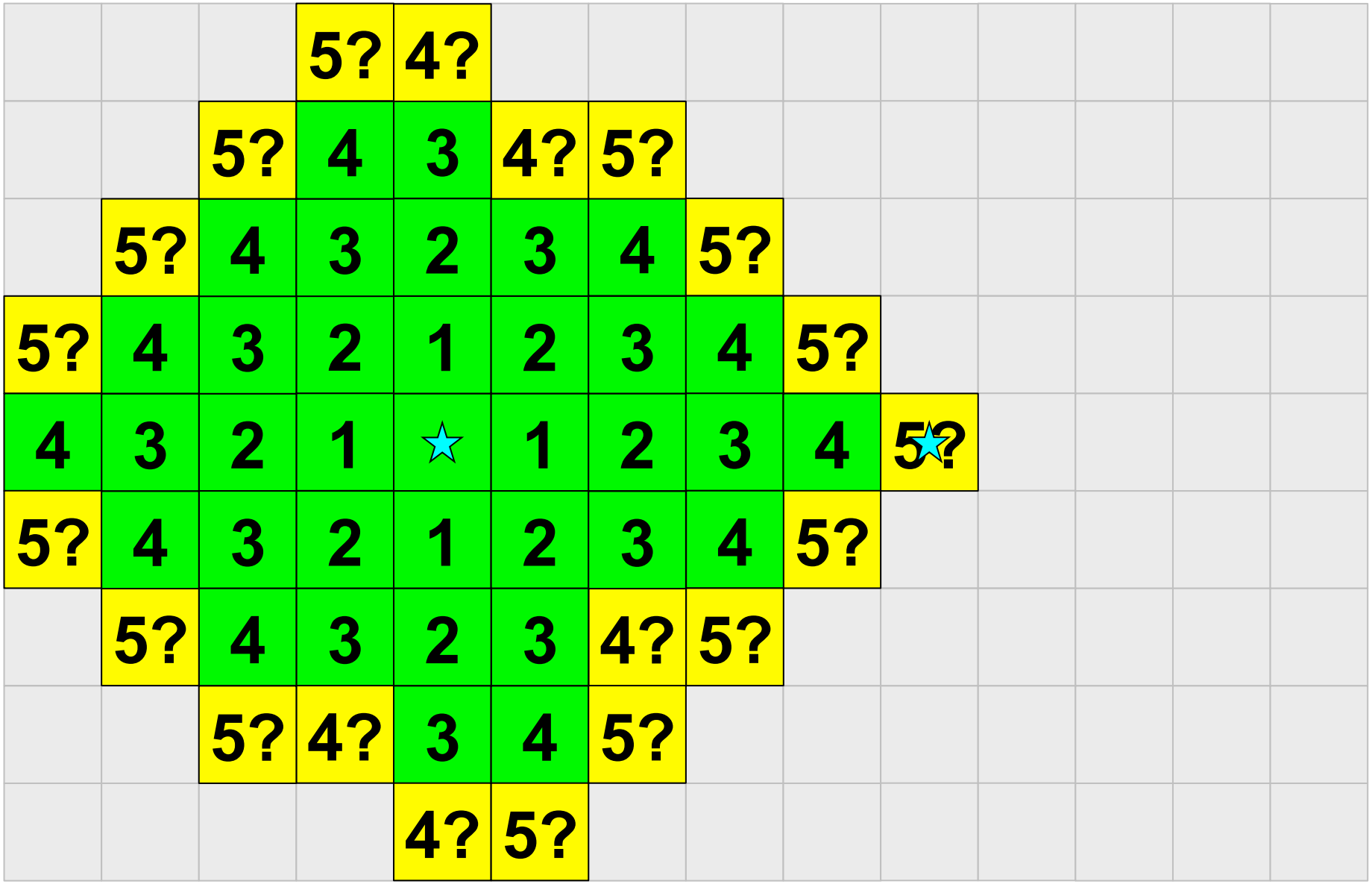

4?
5? 4? 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4? 3 2 1 1 2 3 4 5?
4? 3 2 1 2 3 4? 5?
5? 4 3 2 3 4?
5? 4? 3 4?
4?

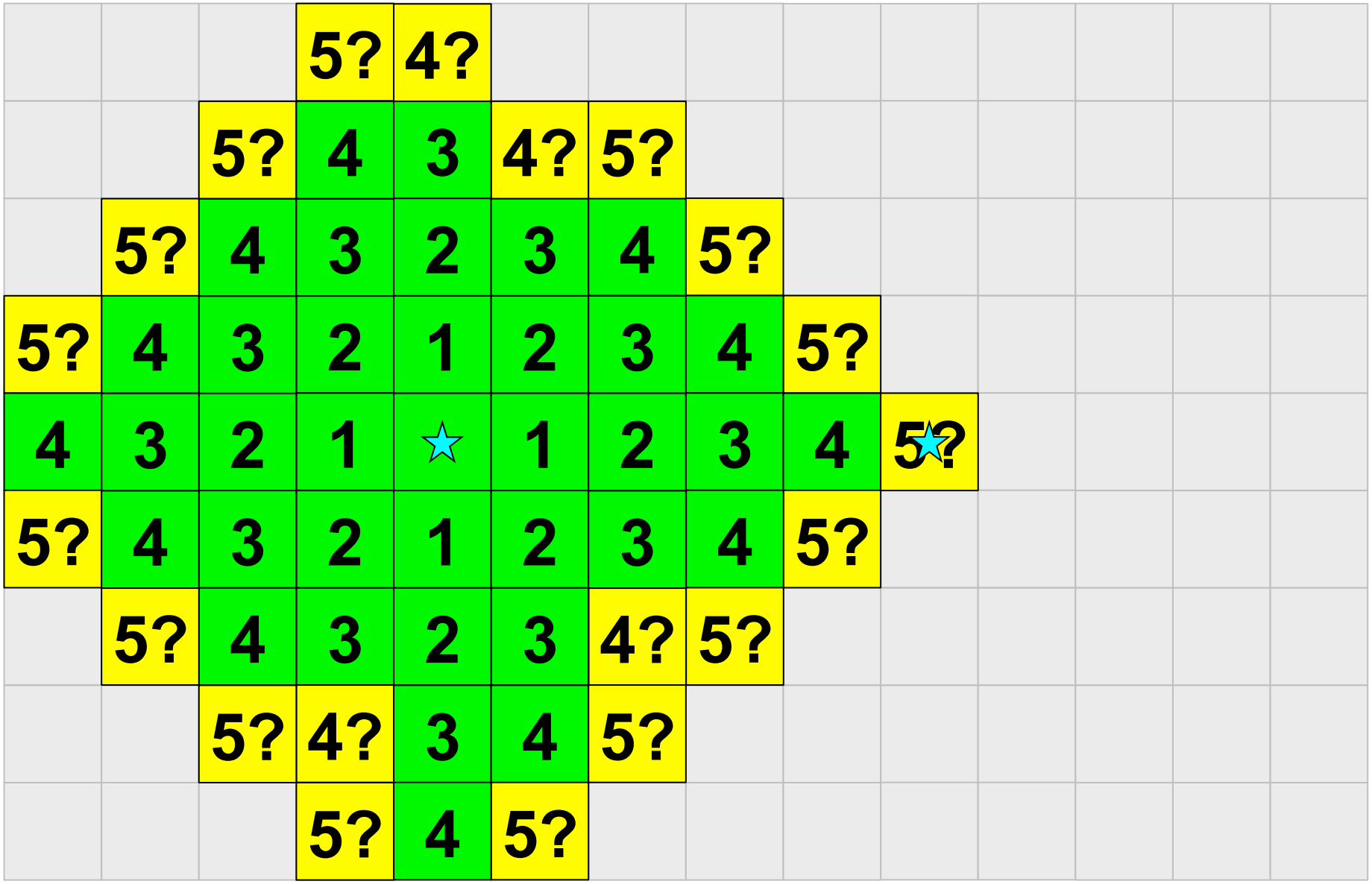

4?
5? 4? 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4 5?
5? 4? 3 2 1 2 3 4? 5?
5? 4 3 2 3 4?
5? 4? 3 4?
4?


4?
5? 4? 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4 5?
5? 4 3 2 1 2 3 4? 5?
5? 4 3 2 3 4?
5? 4? 3 4?
4?


5? 4?
5? 4 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4 5?
5? 4 3 2 1 2 3 4? 5?
5? 4 3 2 3 4?
5? 4? 3 4?
4?


5? 4?
5? 4 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4? 5?
5? 4? 3 4?
4?


5? 4?
5? 4 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4? 5?
5? 4? 3 4 5?
4? 5?


5? 4?
5? 4 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4? 5?
5? 4? 3 4 5?
5? 4 5?

5? 4?
5? 4 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4? 5?
5? 4 3 4 5?
5? 4 5?

5? 4?
5? 4 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4 5?
5? 4 3 4 5?
5? 4 5?

5? 4? 5?
5? 4 3 4 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4 5?
5? 4 3 4 5?
5? 4 5?

5? 4 5?
5? 4 3 4 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4 5?
5? 4 3 4 5?
5? 4 5?

5? 4 5? 6?
5? 4 3 4 5 6?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4 5?
5? 4 3 4 5?
5? 4 5?

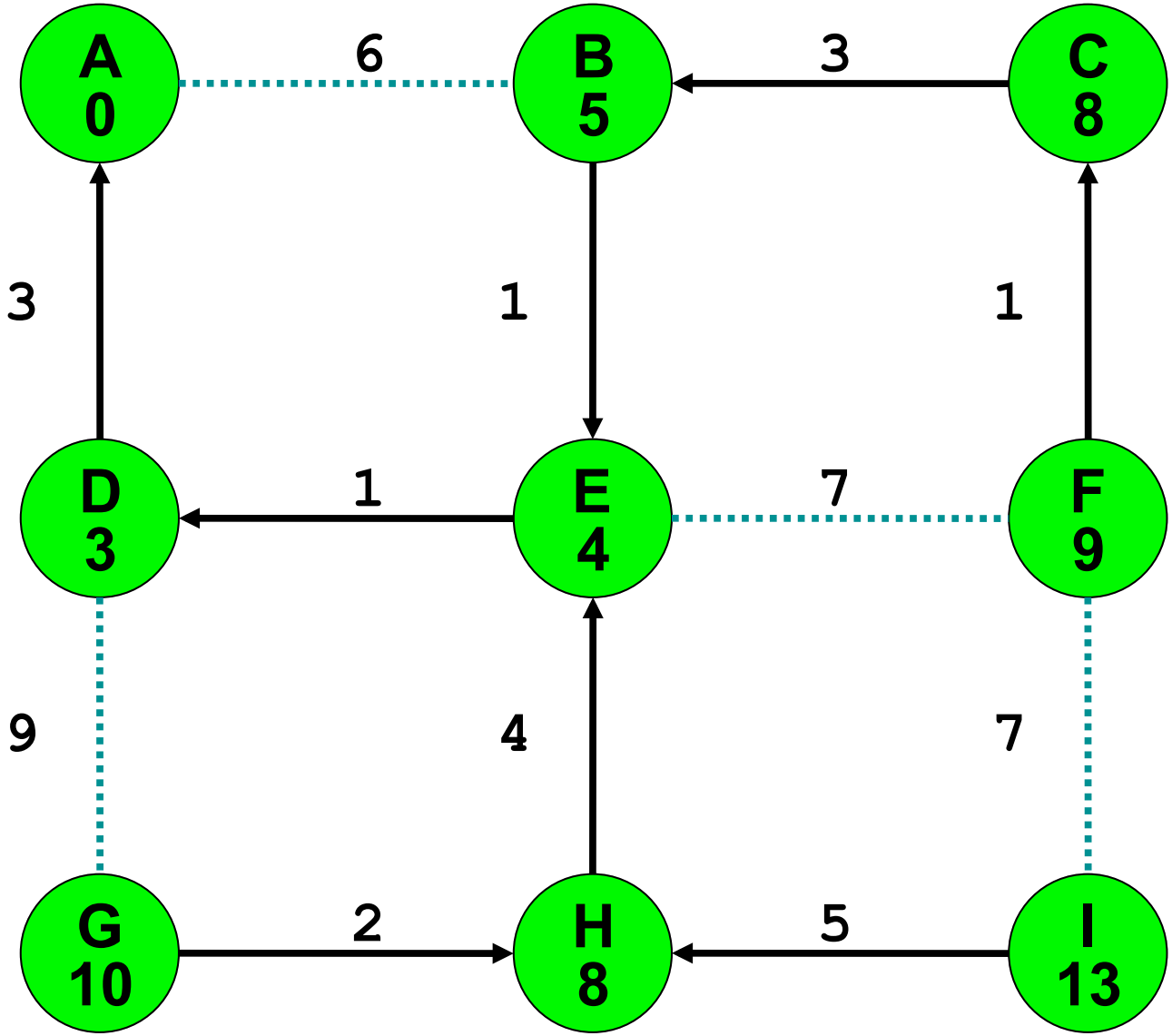
			5?	4	5?	6?							
		5?	4	3	4	5	6?						
	5?	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	☆	1	2	3	4	5?				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4	5	6?					
		5?	4	3	4	5?	6?						
			5?	4	5?								

5? 4 5? 6?
6? 5? 4 3 4 5 6?
6? 5 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4 5 6?
5? 4 3 4 5? 6?
5? 4 5?

			5?	4	5?	6?							
	6?	5?	4	3	4	5	6?						
6?	5	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	☆	1	2	3	4	5?				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4	5	6?					
	6?	5	4	3	4	5?	6?						
		6?	5?	4	5?								

			5?	4	5?	6?							
	6?	5?	4	3	4	5	6?						
6?	5	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	☆	1	2	3	4	☆				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4	5	6?					
	6?	5	4	3	4	5?	6?						
		6?	5?	4	5?								

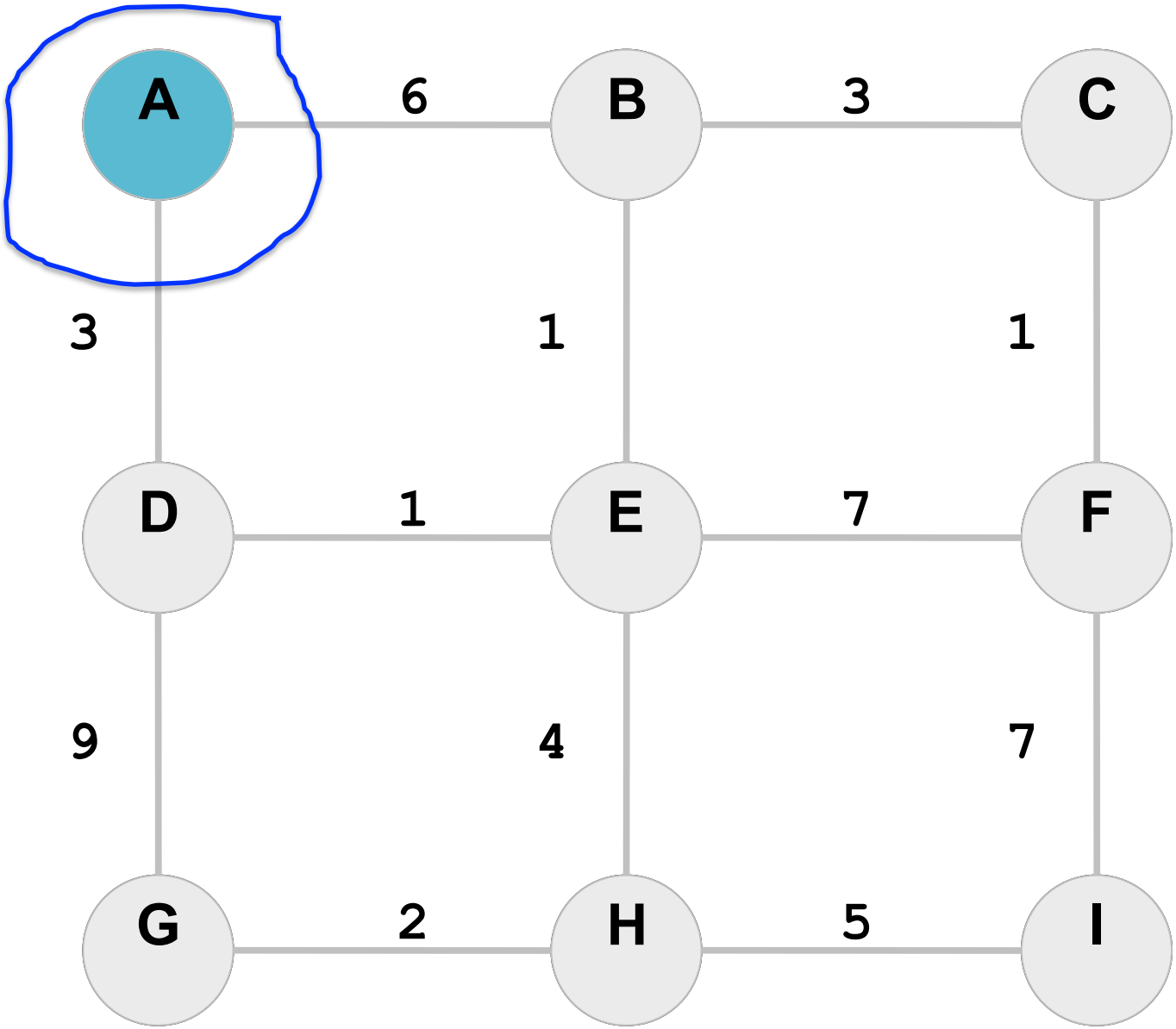
Works with Weights



Why does it work?

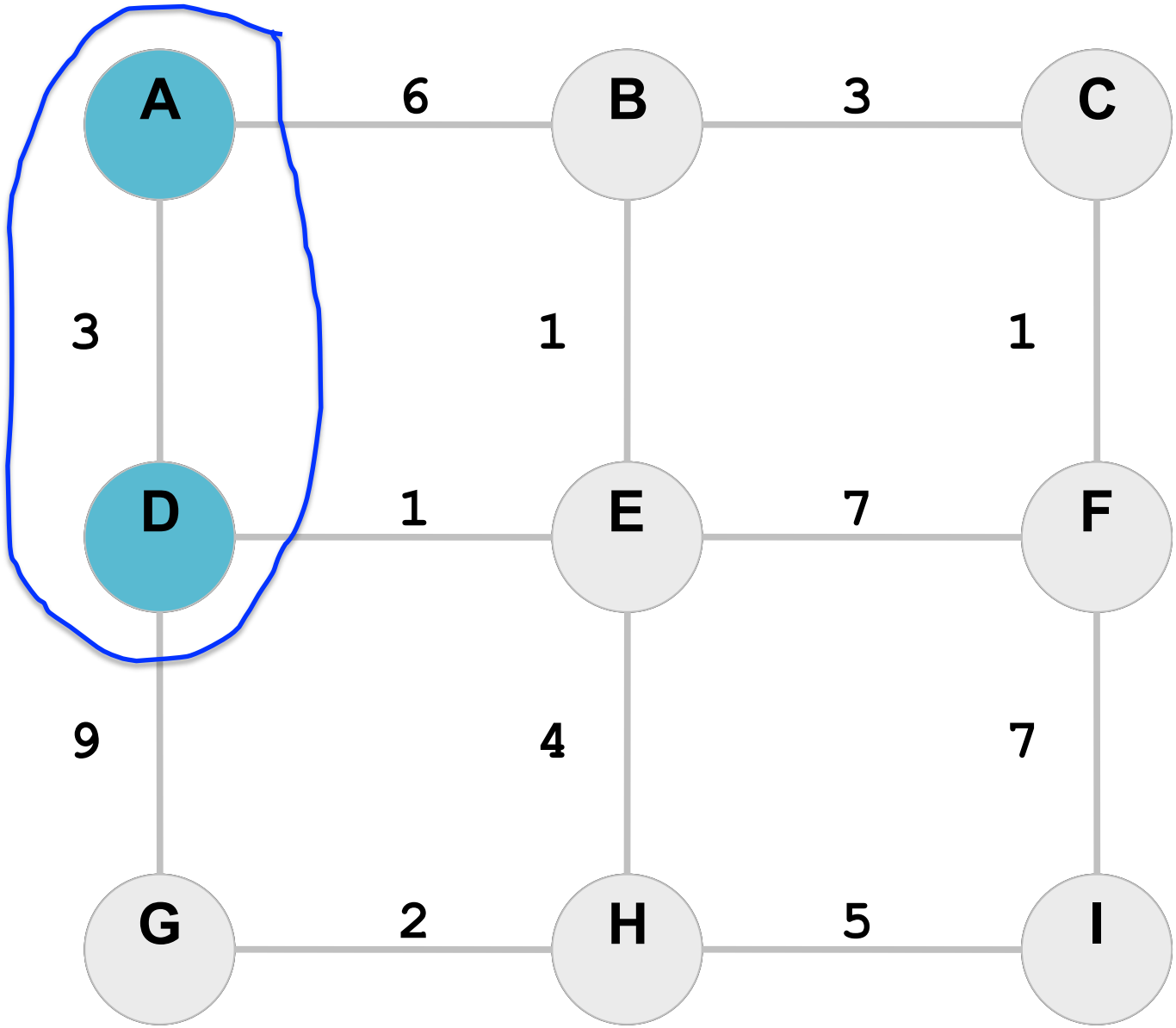
We look at paths in order of *increasing* distance from start

Works with Weights



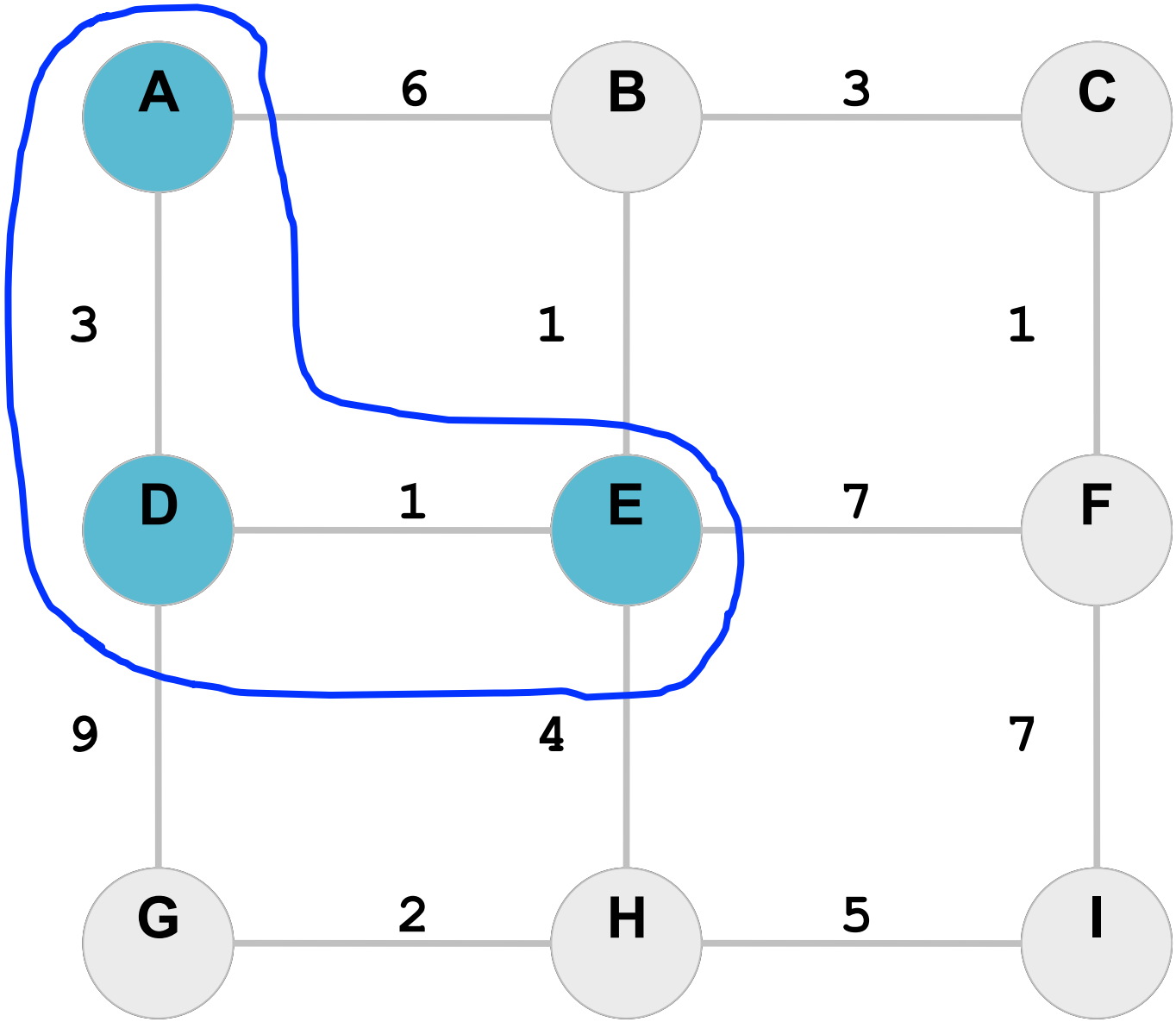
0

Works with Weights



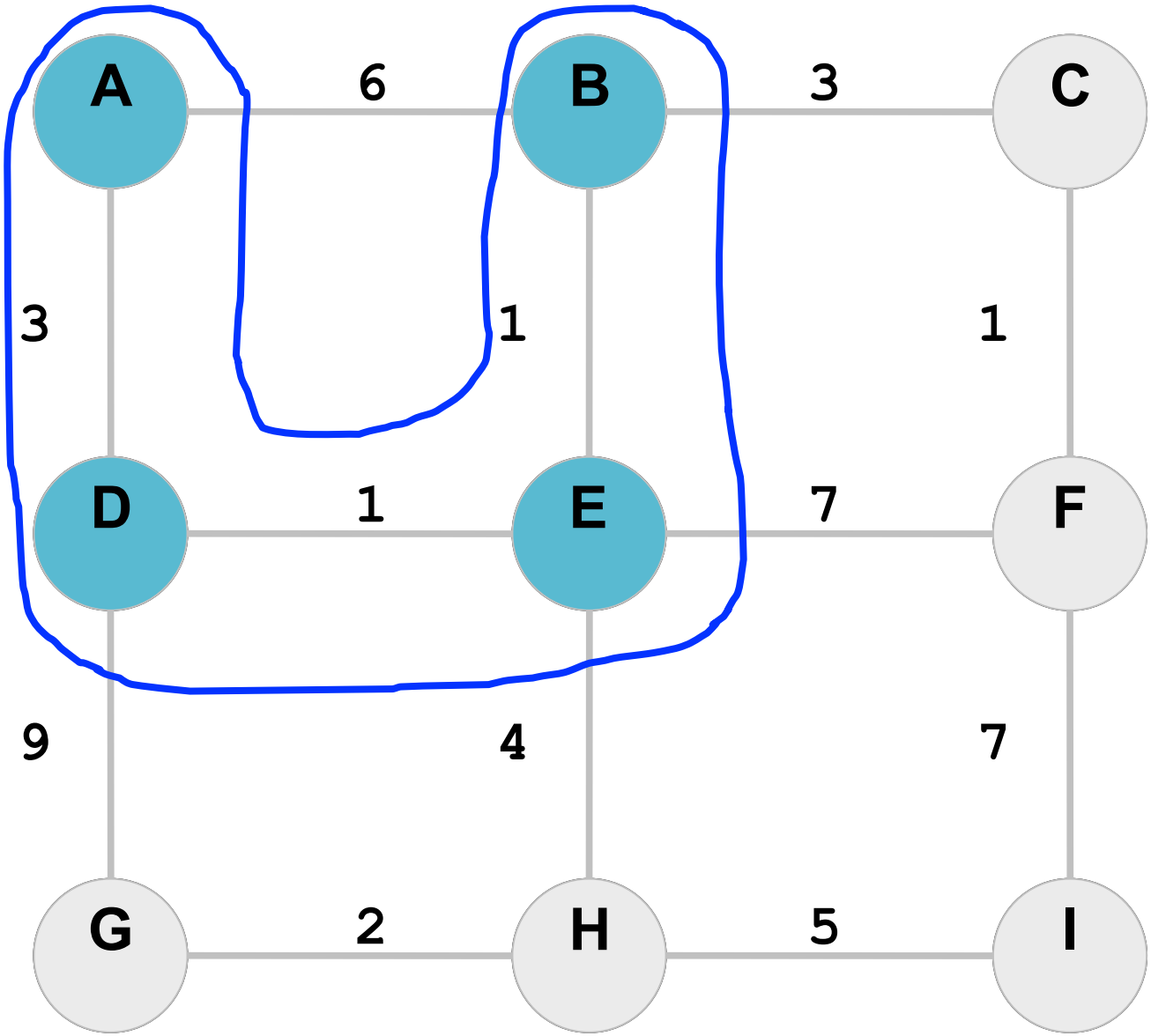
3

Works with Weights



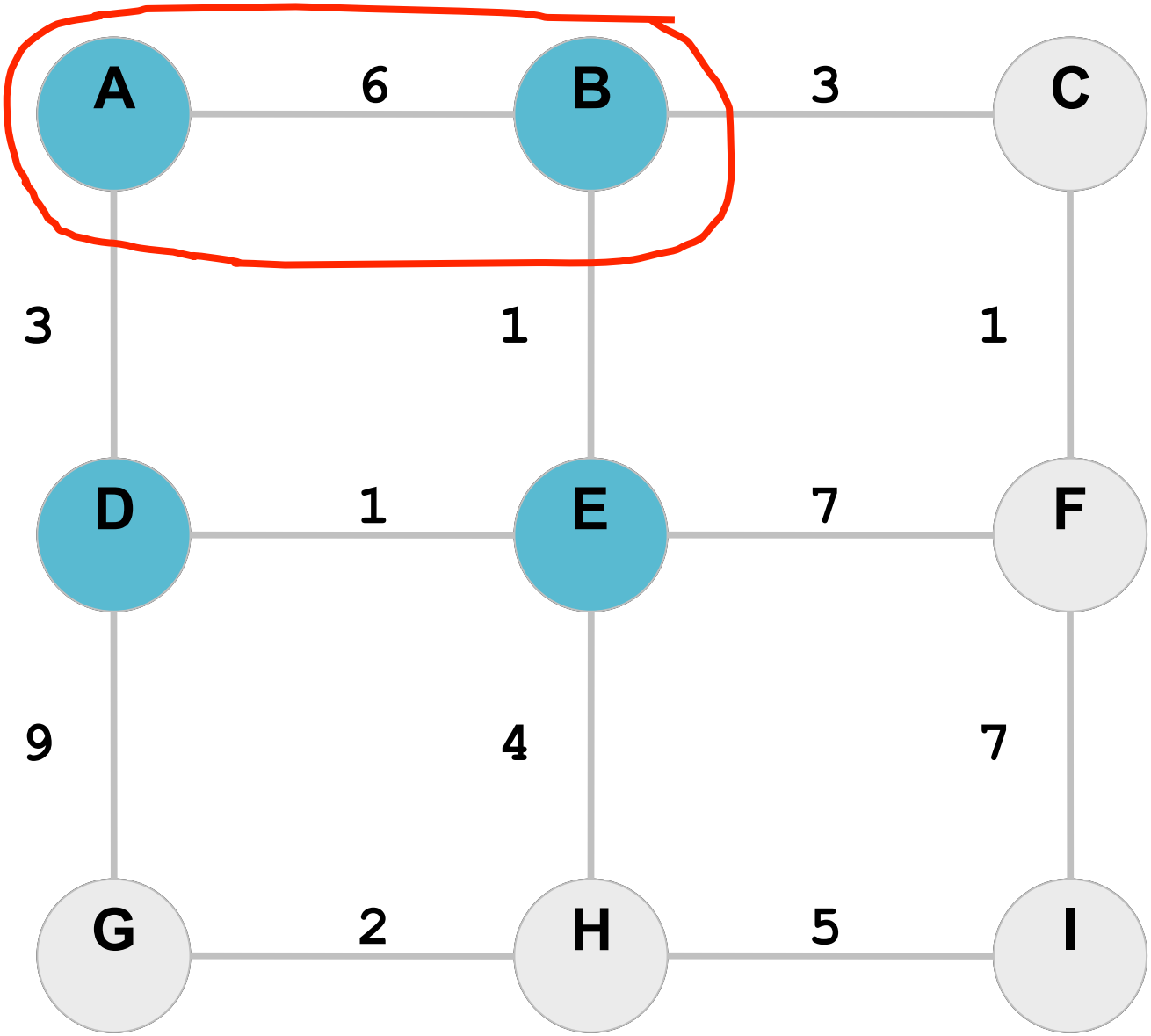
4

Works with Weights



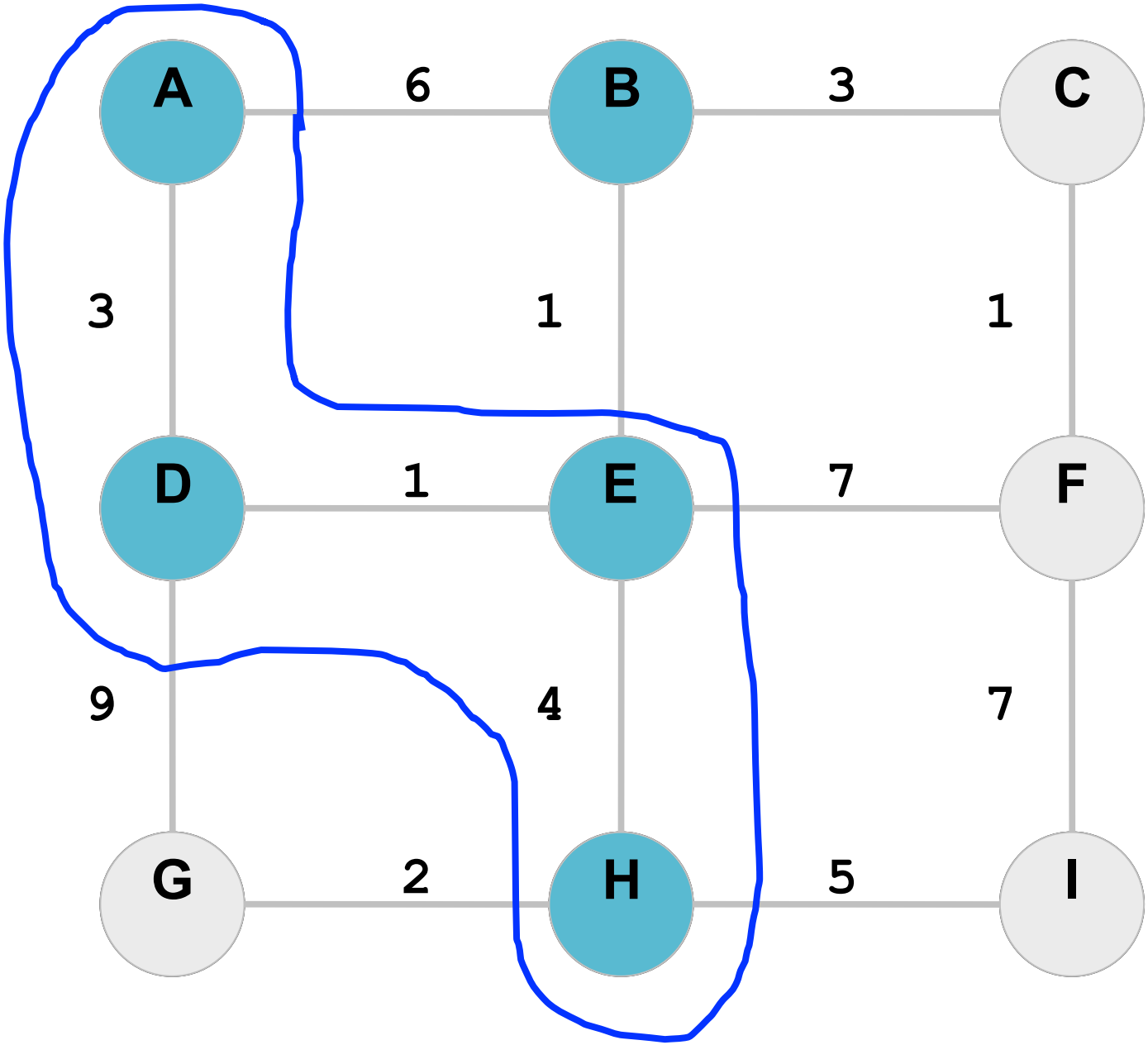
5

Works with Weights



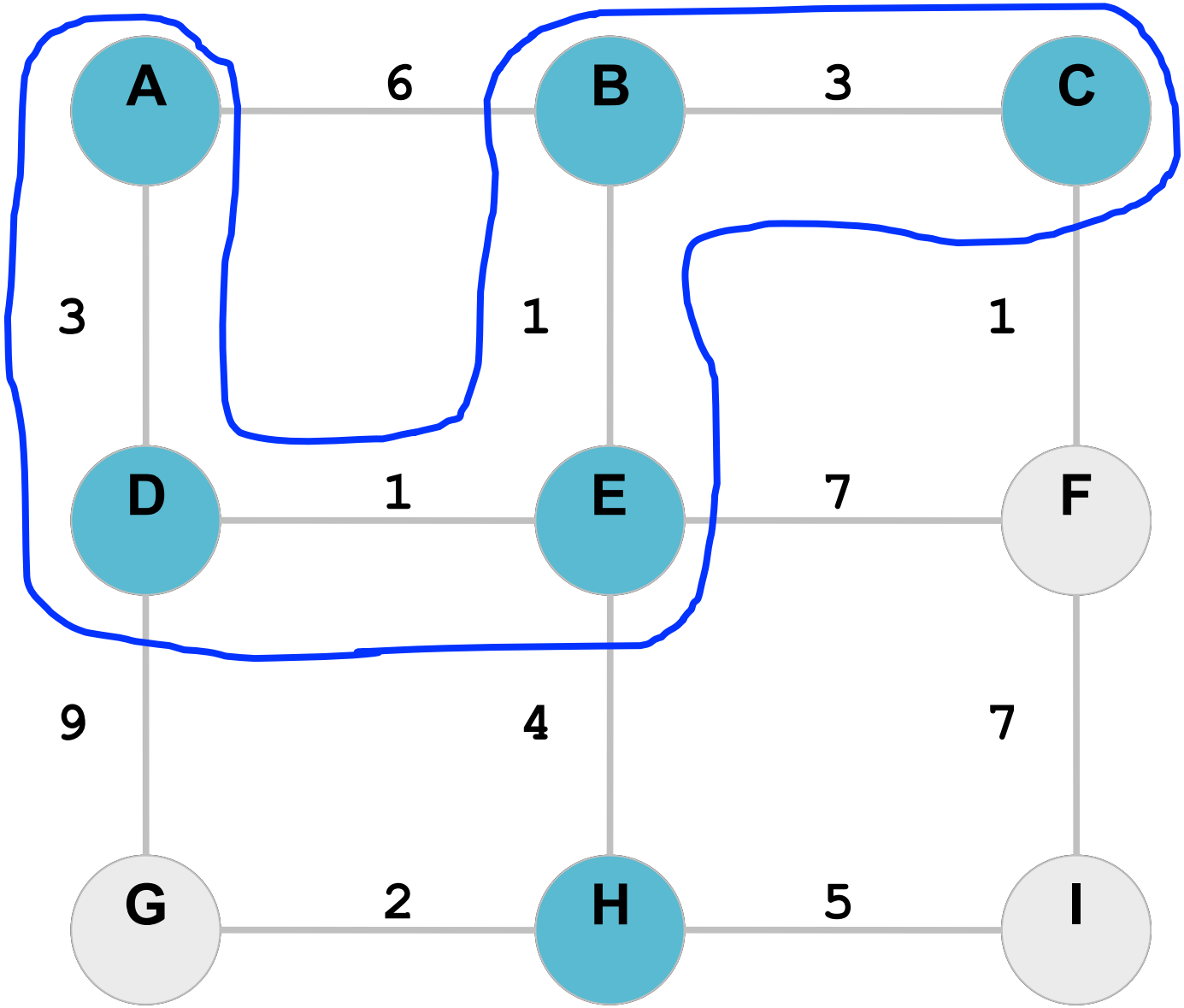
6

Works with Weights



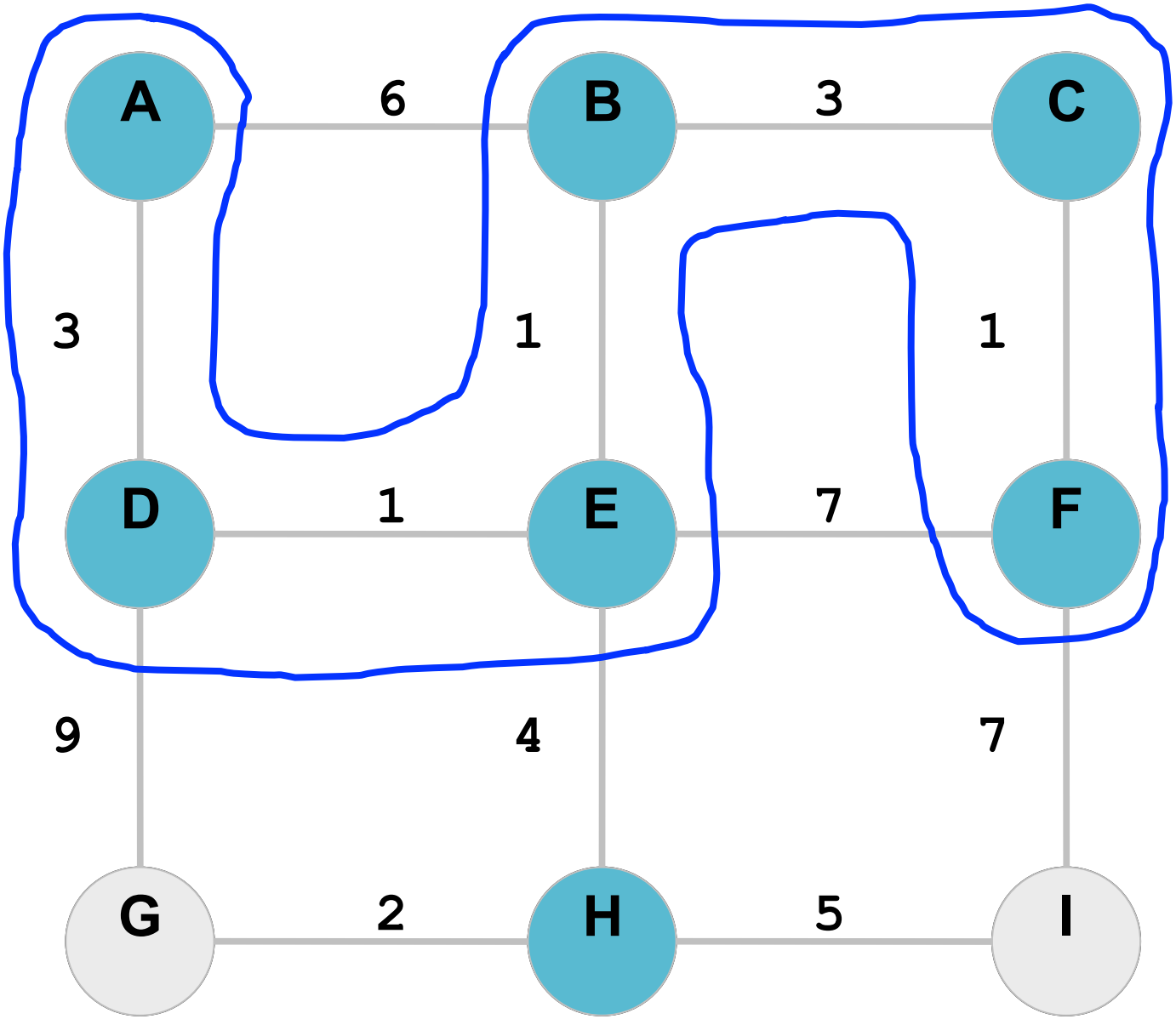
8

Works with Weights



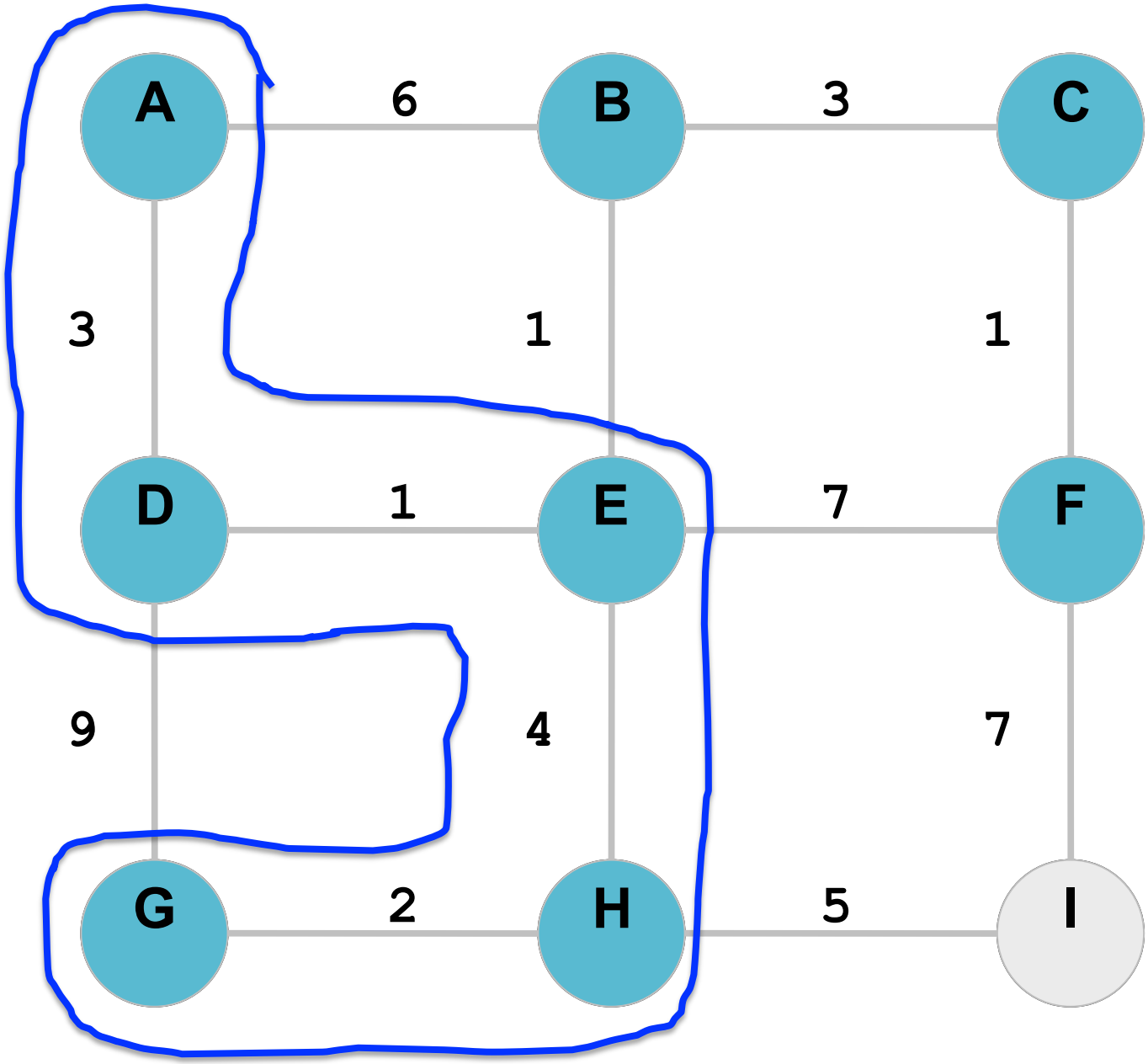
8

Works with Weights



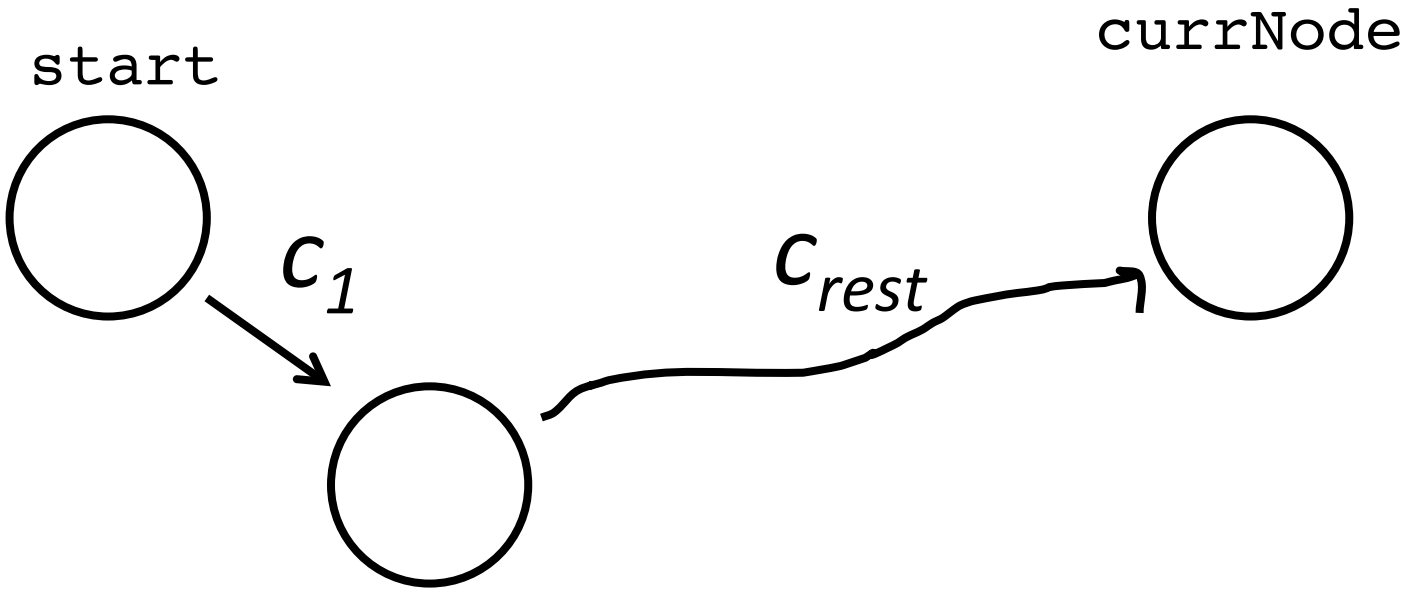
9

Works with Weights

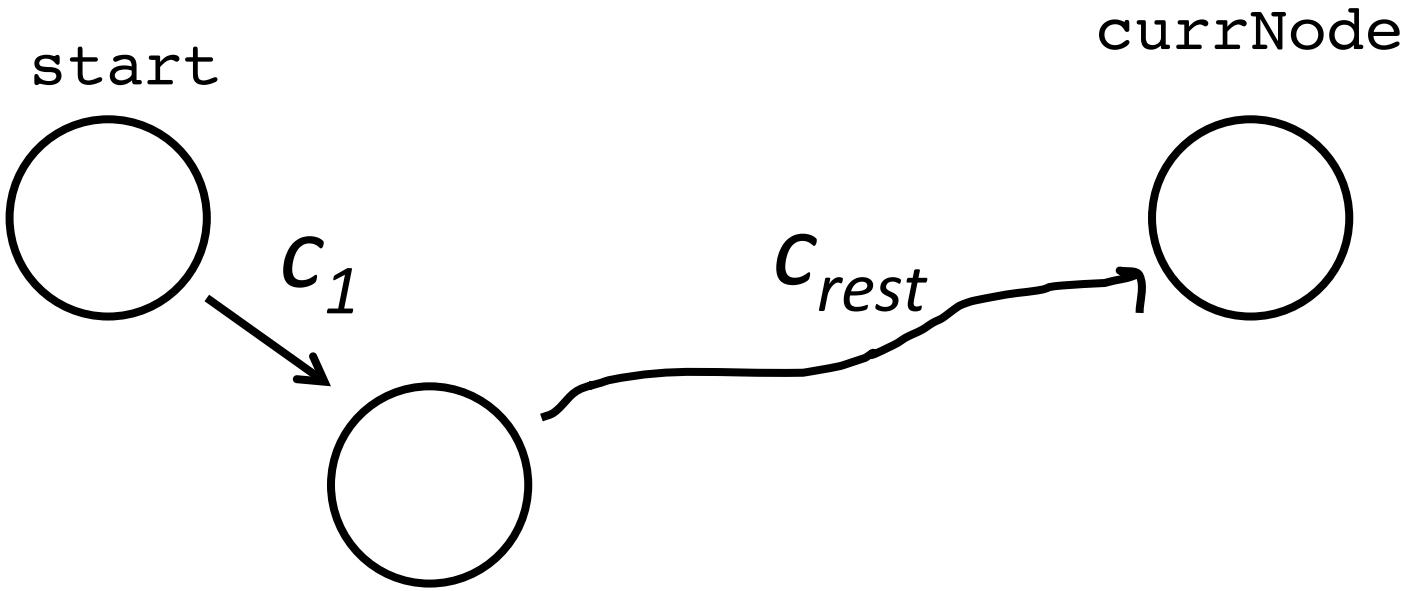


10

Imagine a Path From the Todo List

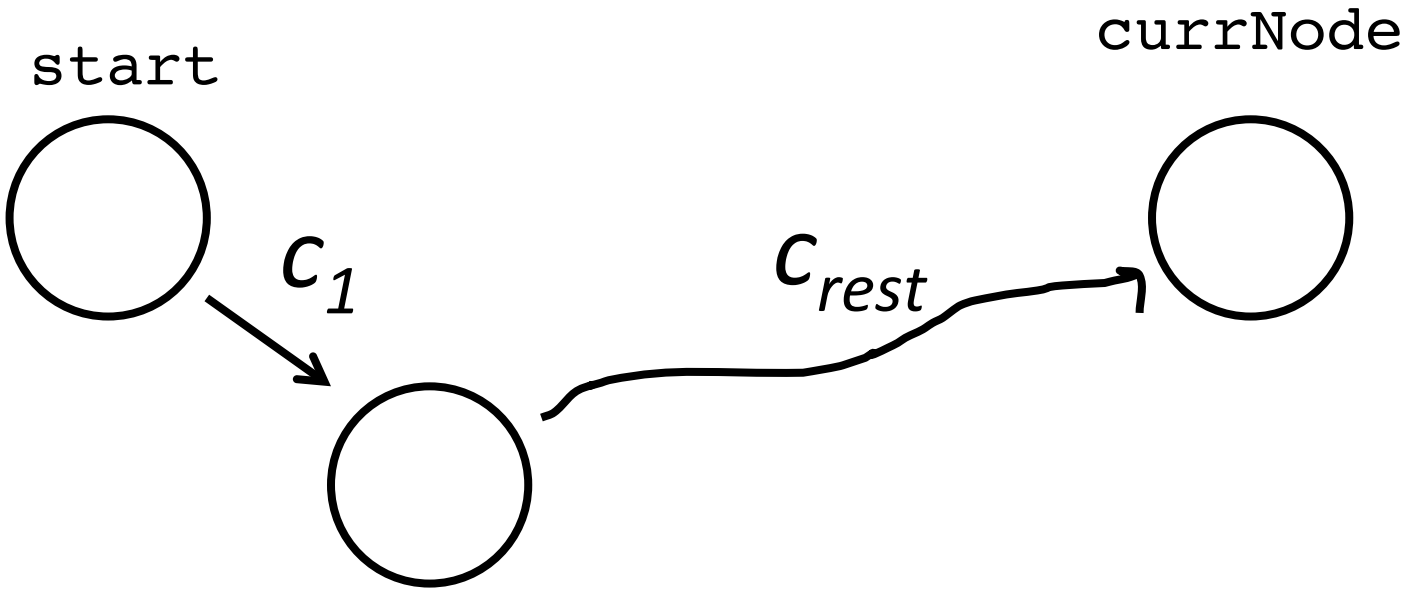


Imagine a Path From the Todo List



Is it possible that there is a path from start to currNode with lower cost?

Imagine a Path From the Todo List



Is it possible that there is a path from start to currNode with lower cost?

NO

Imagine a Path From the Todo List

Is it possible that there is a path from start to currNode with lower cost?

NO

Imagine a Path From the Todo List

Is it possible that there is a path from start to currNode with lower cost?

NO

We are looking at paths in order of increasing cost, and we haven't seen currNode yet.
Thus, this is the shortest path to currNode

Common Mistakes

Should we check goal
before putting into todo?

Dijkstra's Algorithm

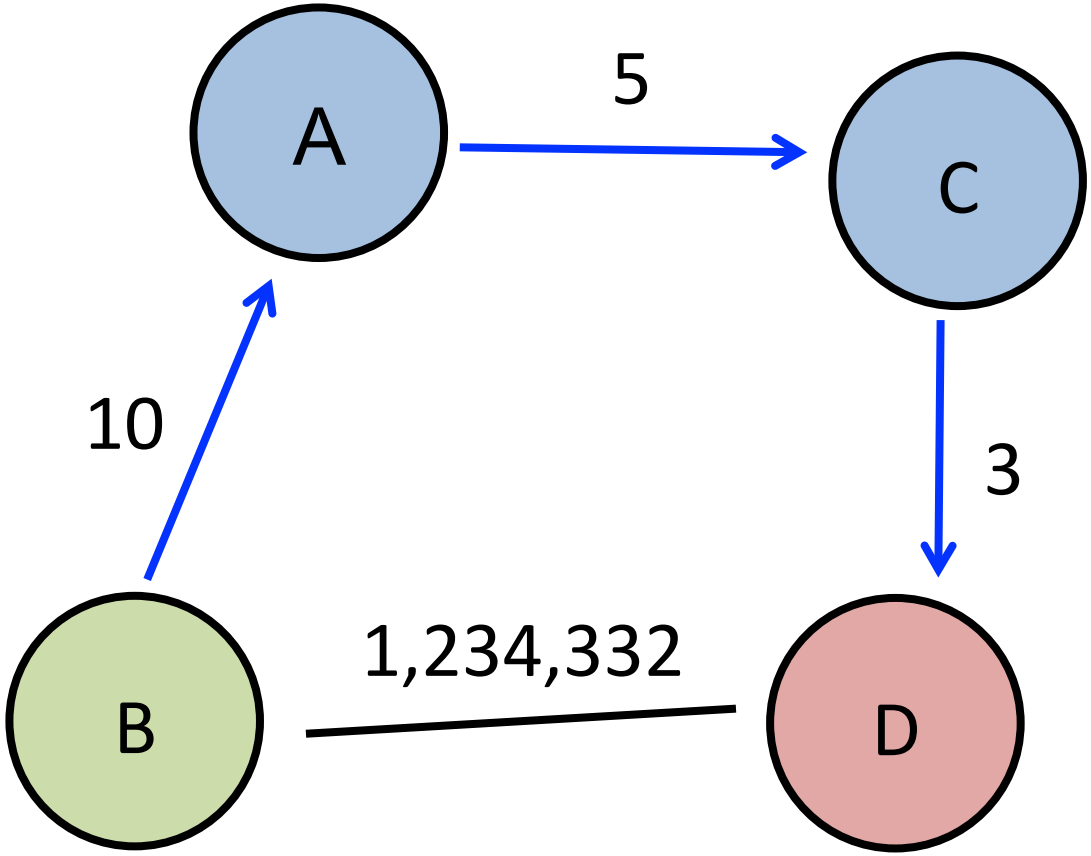
```
pQueue = PriorityQueue()
pQueue.enqueue(newPath(startNode), 0)
seen = Set();
while !pQueue.isEmpty():
    currPath = pQueue.dequeue()
    currState = last(currPath);
    if(currState is goal) return currPath;
    if(seen contains currState) continue;
    seen.add(currState);
    for nextState in getNextStates(currState)
        path = newPath(currPath, nextState);
        pQueue.enqueue(path, getCost(path));
    }
}
```

Dijkstra's Algorithm

```
pQueue = PriorityQueue()
pQueue.enqueue(newPath(startNode), 0)
seen = Set();
while !pQueue.isEmpty():
    currPath = pQueue.dequeue()
    currState = last(currPath);
    if(currState is goal) return currPath;
    if(seen contains currState) continue;
    seen.add(currState);
    for nextState in getNextStates(currState)
        path = newPath(currPath, nextState);
        pQueue.enqueue(path, getCost(path));
    }
}
```

Check here if nextState is the goal

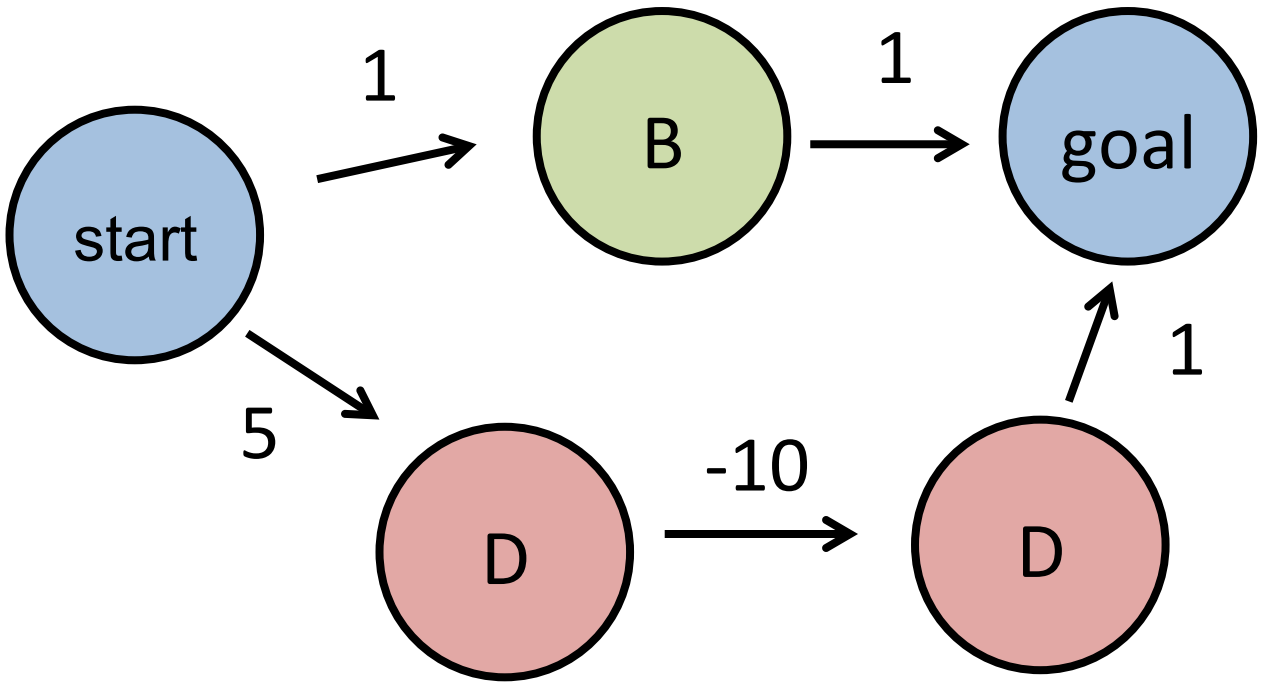
Check Goal After Dequeue



No

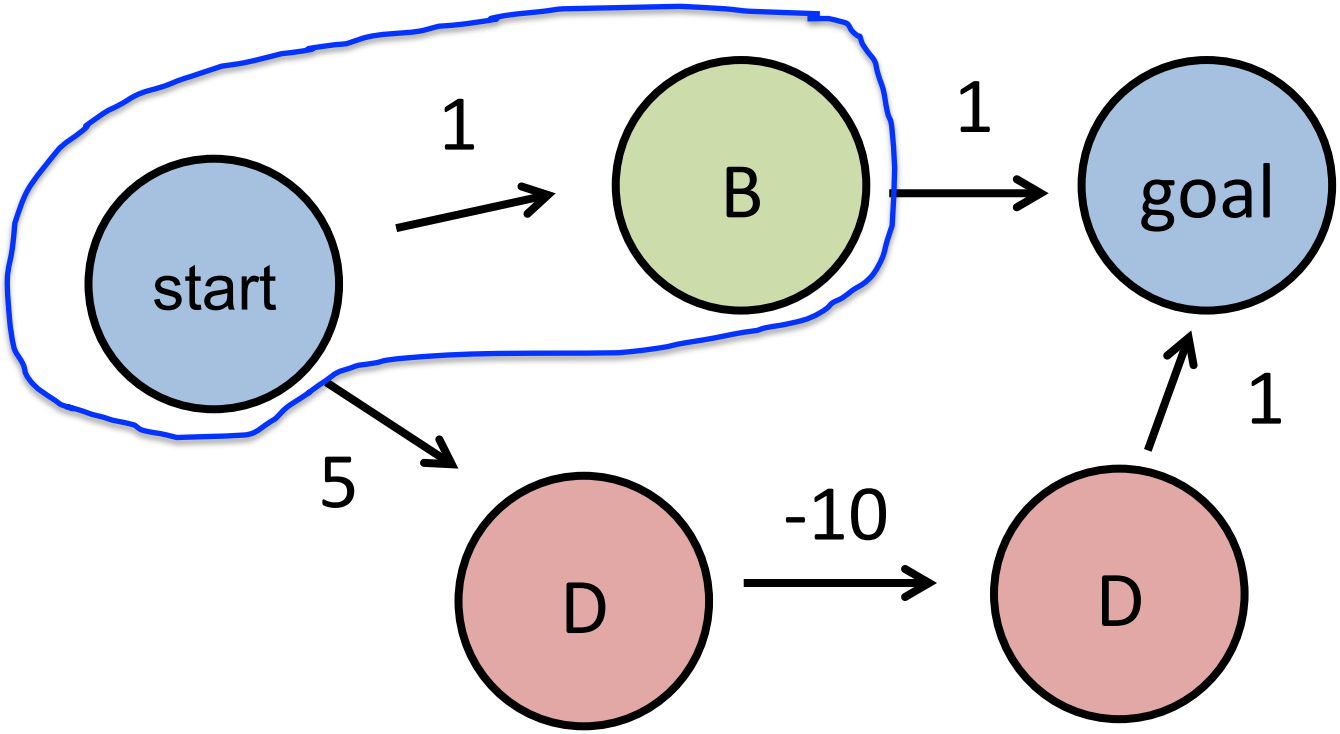
Does it work with negative edges?

Check Goal After Dequeue



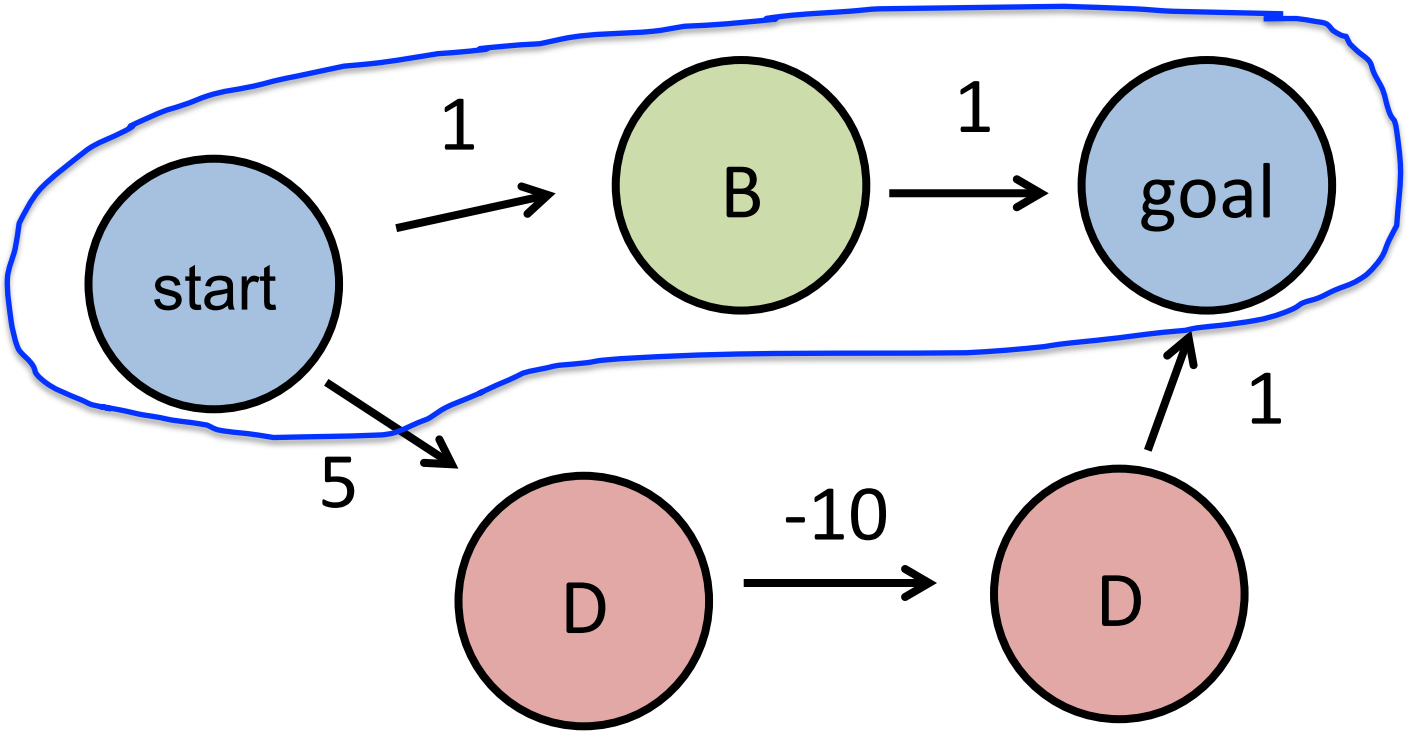
Dijkstra's checks paths in order of distance from start

Check Goal After Dequeue



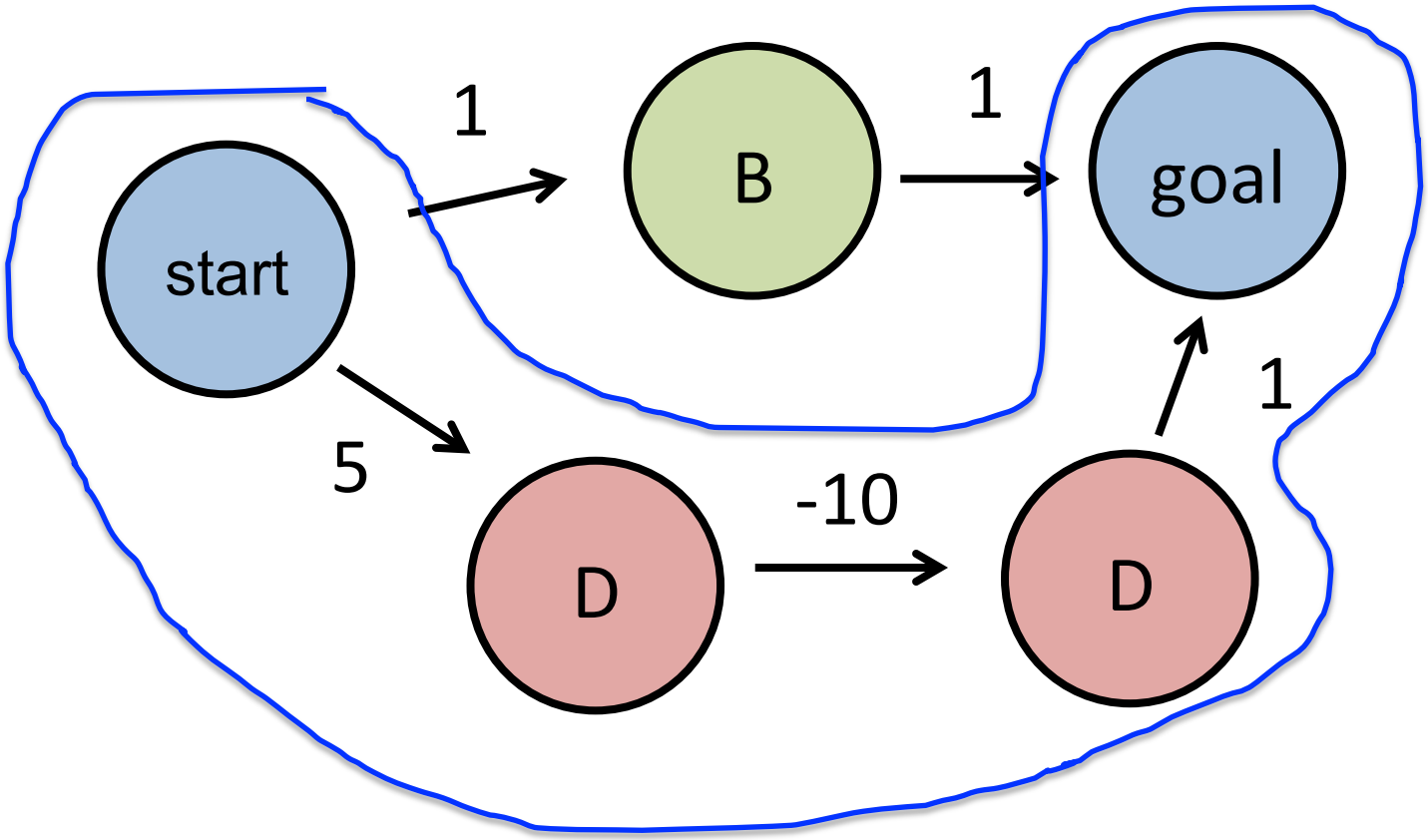
Dijkstra's checks paths in order of distance from start

Check Goal After Dequeue



Dijkstra's checks paths in order of distance from start

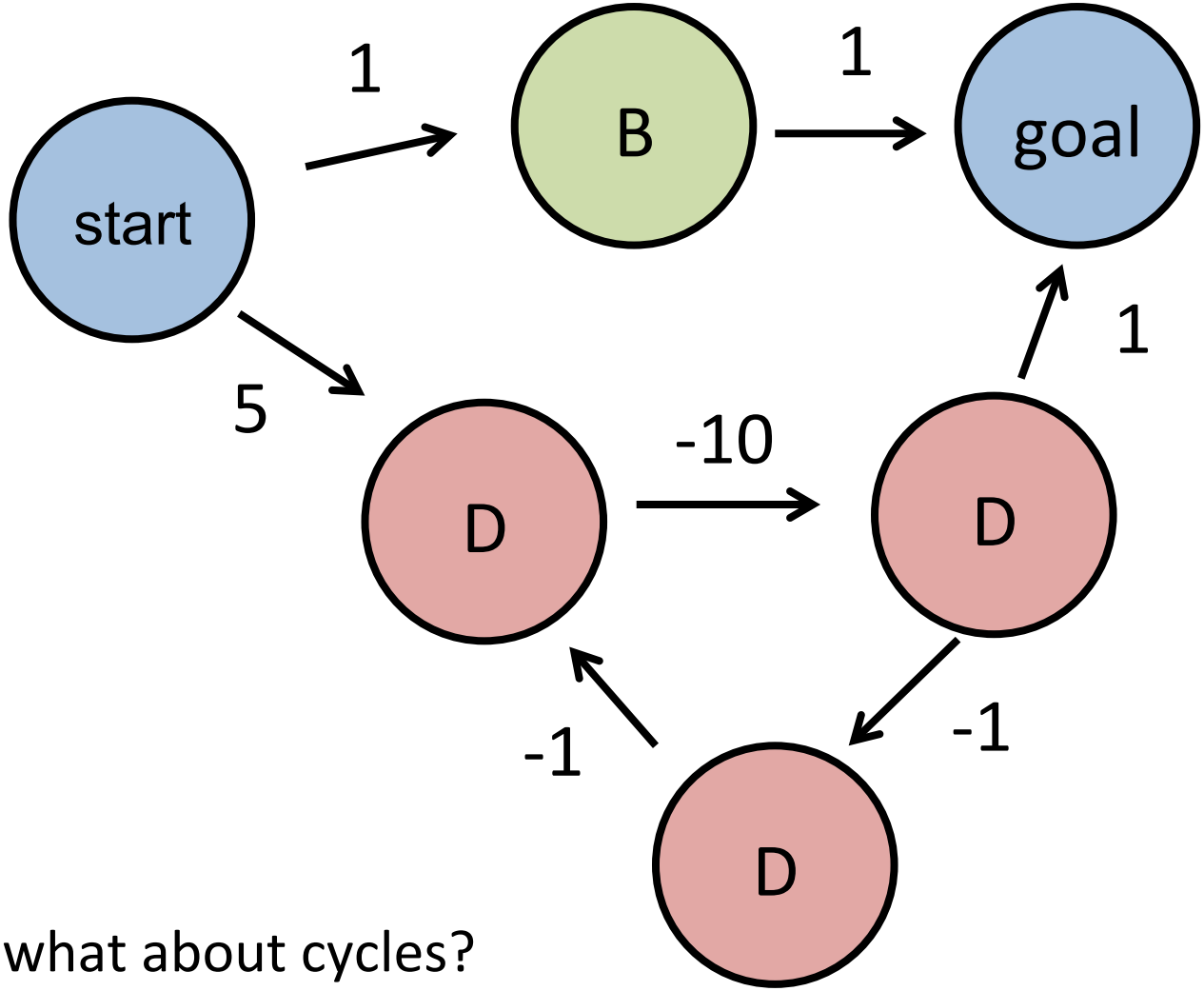
Check Goal After Dequeue



It will miss this path

No

Check Goal After Dequeue



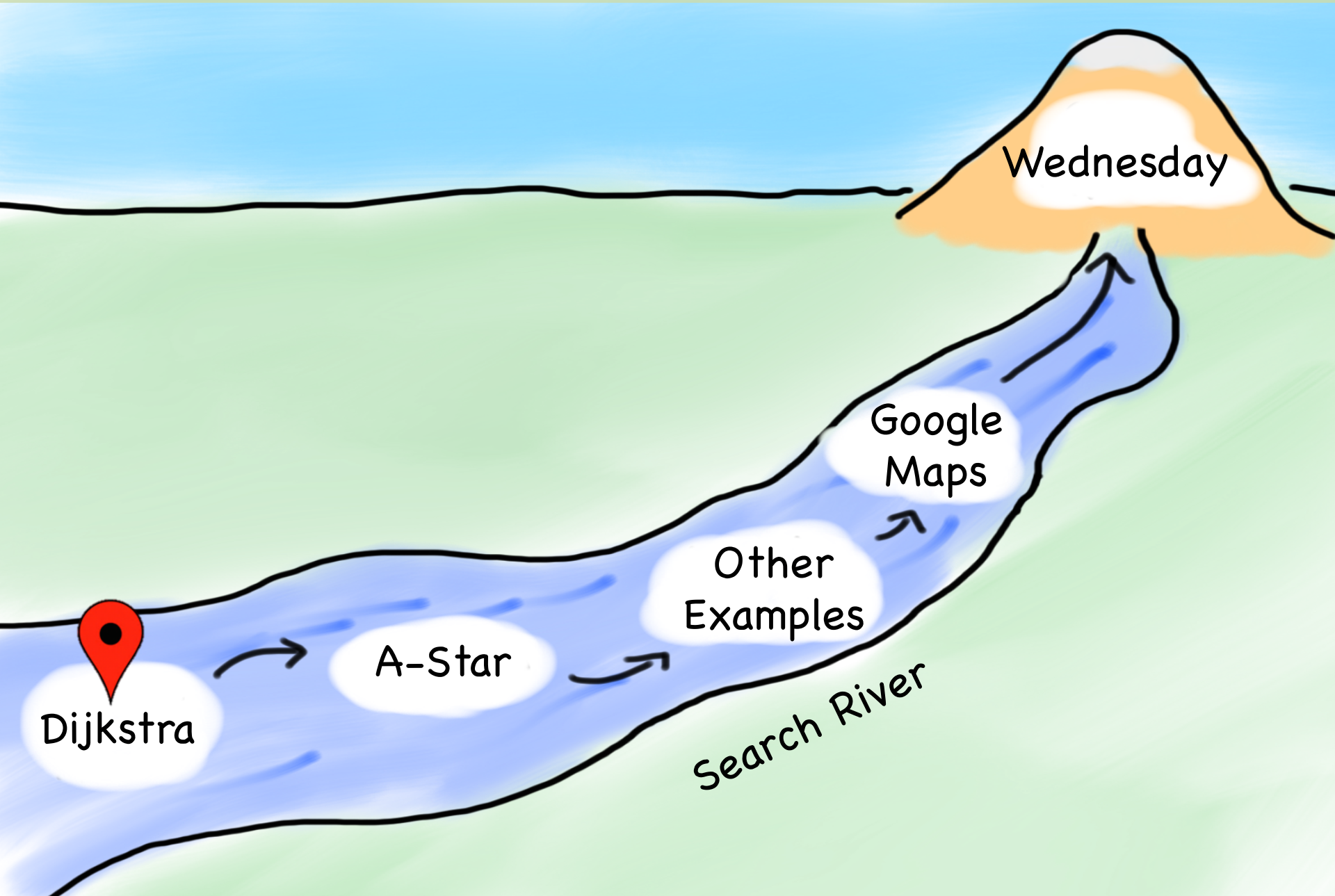
And what about cycles?

No

That’s all for Dijkstra’s

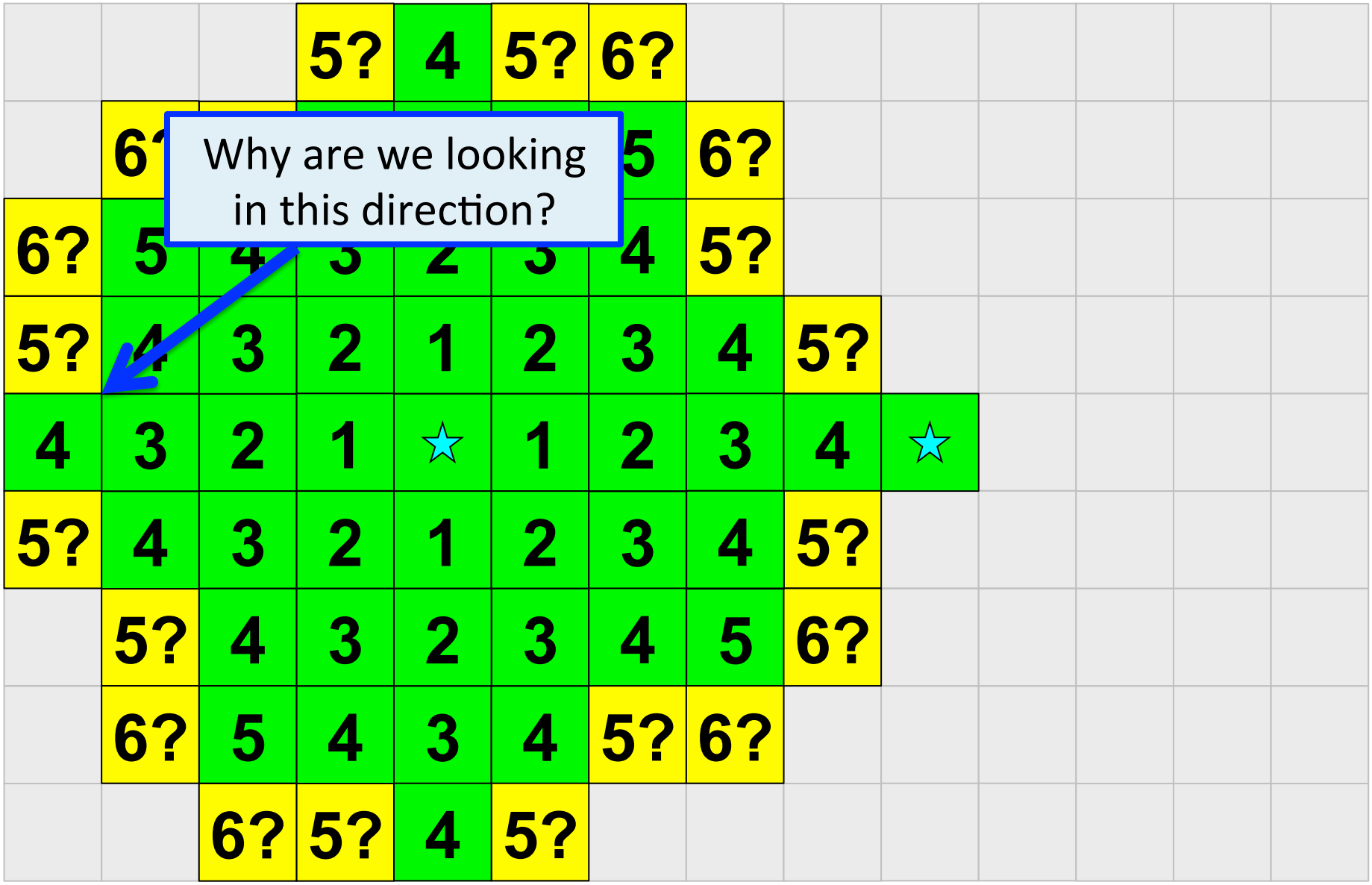
People make this sound hard.

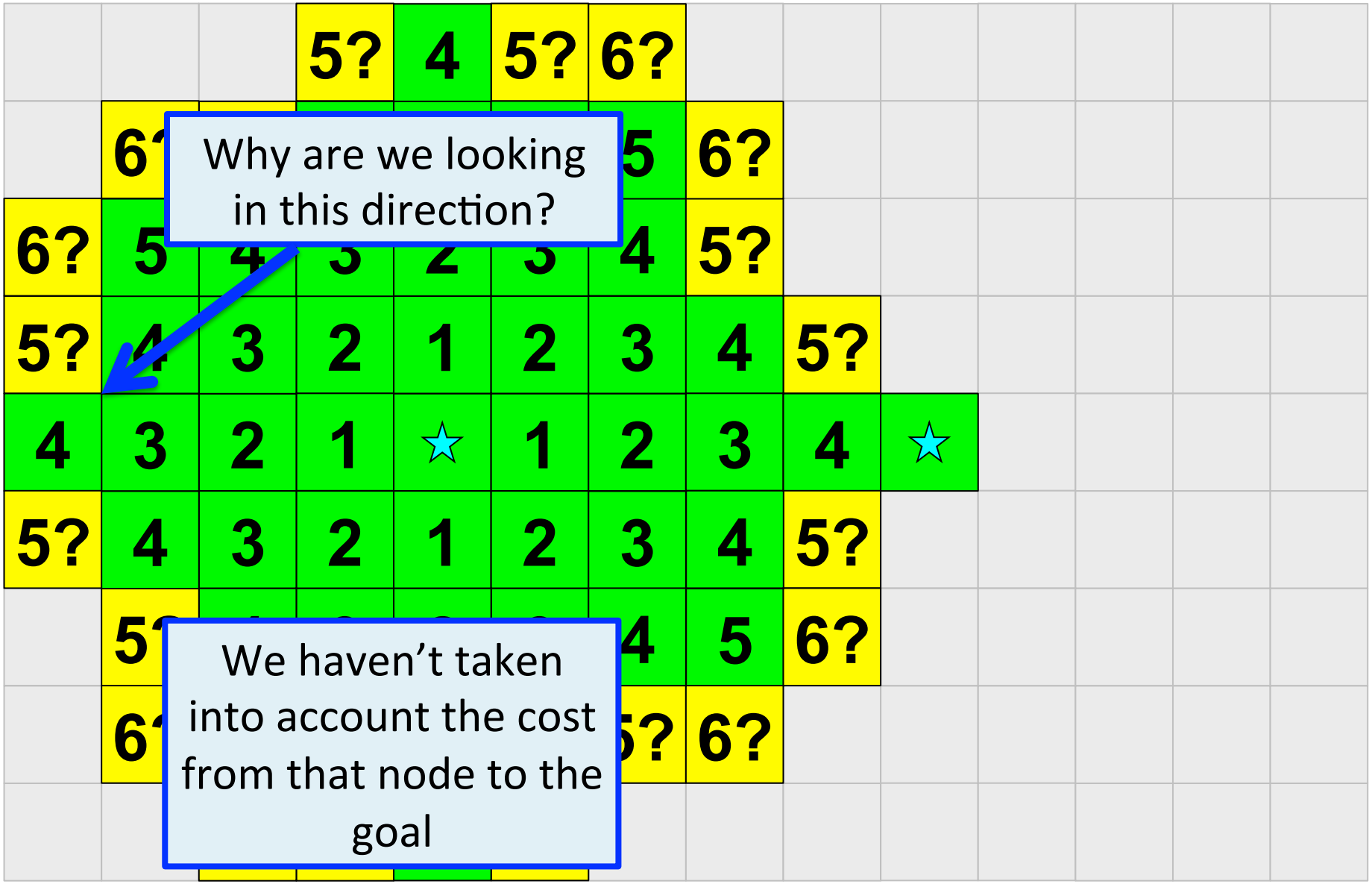
Today's Route



Today's Route

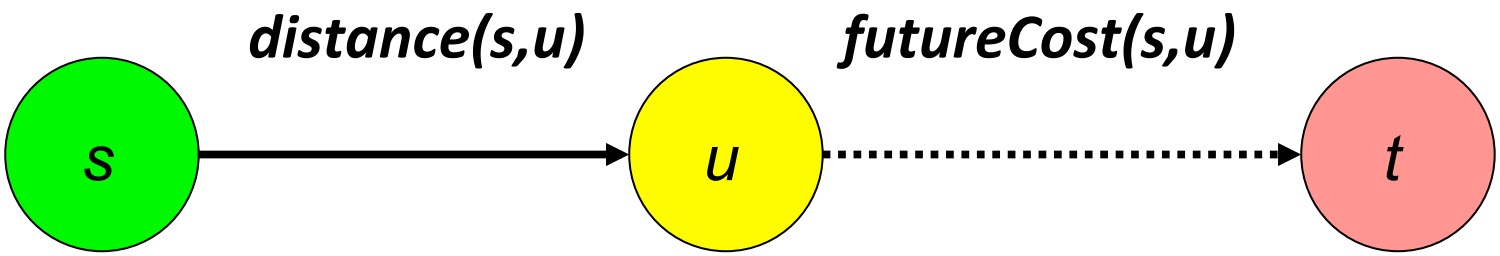
There was a very interesting question at the end of class…


Why are we looking in this direction?


Why are we looking in this direction?
We haven't taken into account the cost from that node to the goal

Dijkstra Priority

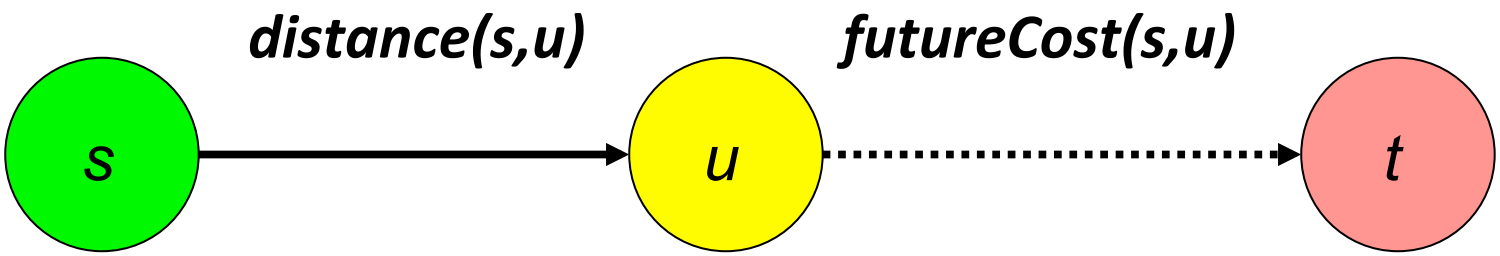
priority(u) = distance(s, u)



Priority of the path that ends in u

A-Star Priority

priority(u) = distance(s, u) + futureCost(u, t)



Priority of the path that ends in u

*note we will revise this slightly

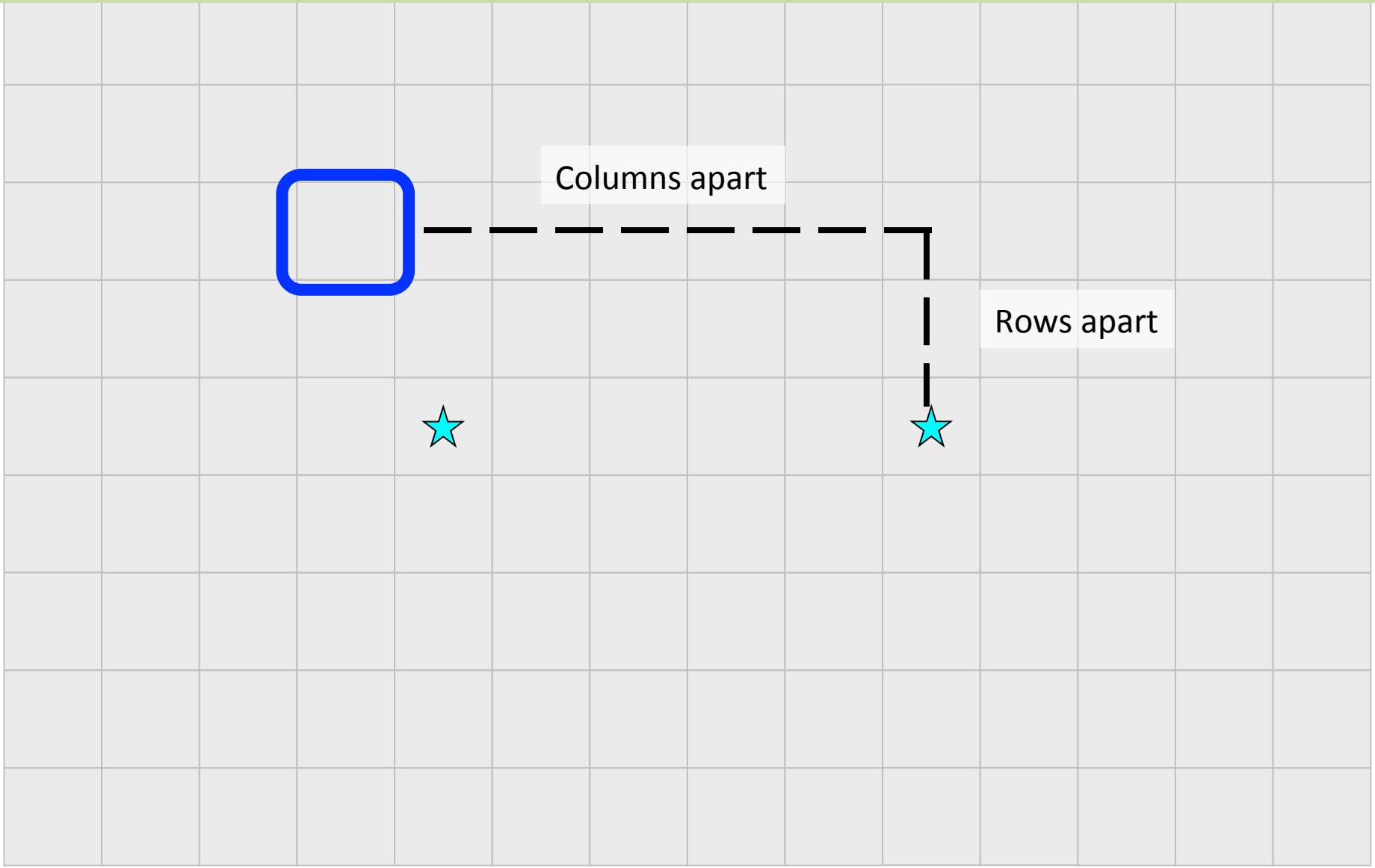
Recall: Dijkstra's

```
pQueue = PriorityQueue()
pQueue.enqueue(newPath(start), 0)
seen = Set();
while !pQueue.isEmpty():
  currPath = pQueue.dequeue()
  currState = last(currPath);
  if(currState is goal) return currPath;
  if(seen contains currState) continue;
  seen.add(currState);
  for nextState in getNextStates(currState)
    path = newPath(currPath, nextState);
    cost = getCost(path);
    pQueue.enqueue(path, getCost(path)+ cost);
  }
}
```

A-Star

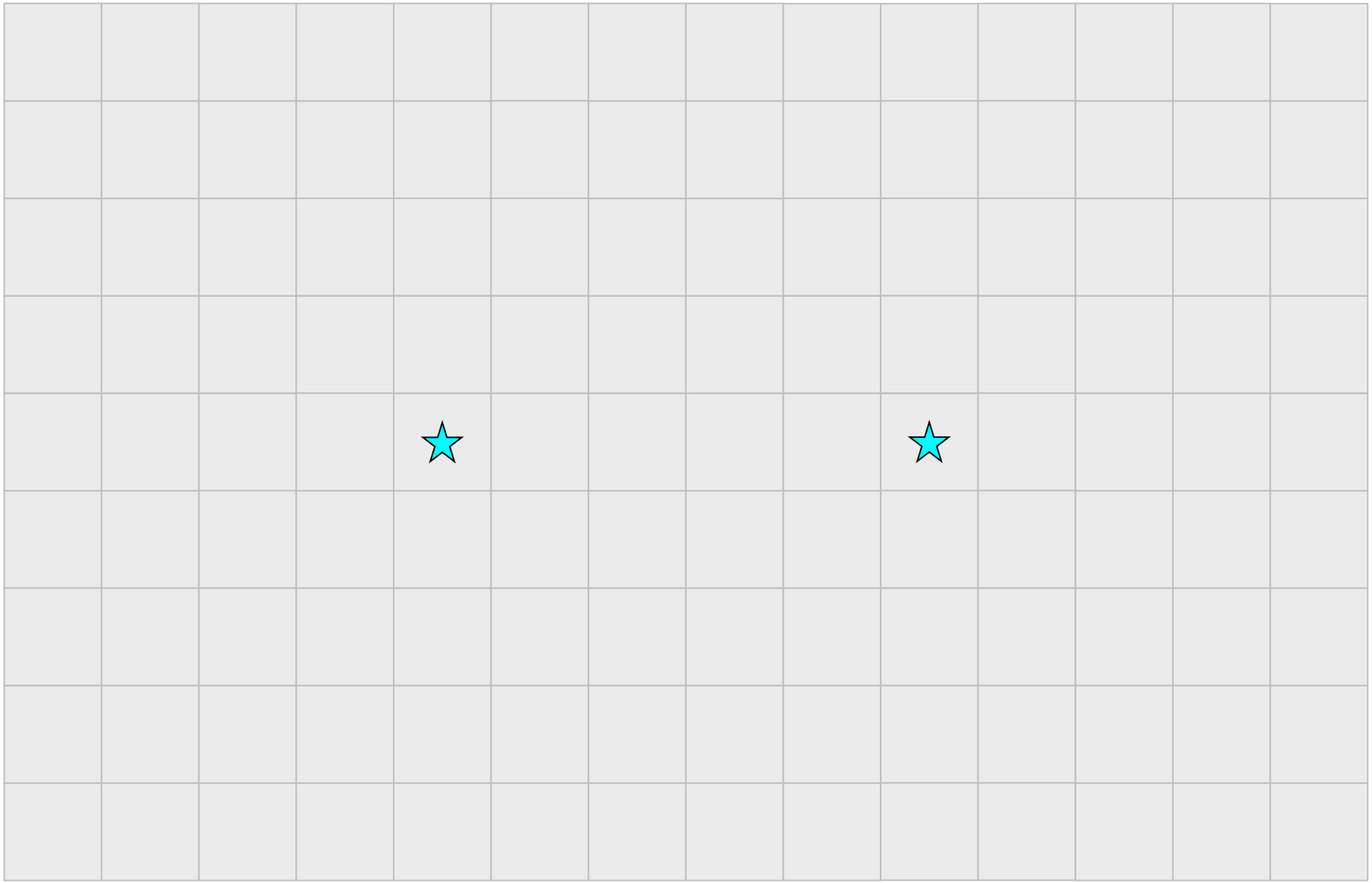
```
pQueue = PriorityQueue()
pQueue.enqueue(newPath(start), h(start,goal))
seen = Set();
while !pQueue.isEmpty():
  currPath = pQueue.dequeue()
  currState = last(currPath);
  if(currState is goal) return currPath;
  if(seen contains currState) continue;
  seen.add(currState);
  for nextState in getNextStates(currState)
    path = newPath(currPath, nextState);
    cost = getCost(path) + h(currState, goal);
    pQueue.enqueue(path, getCost(path)+ cost);
  }
}
```

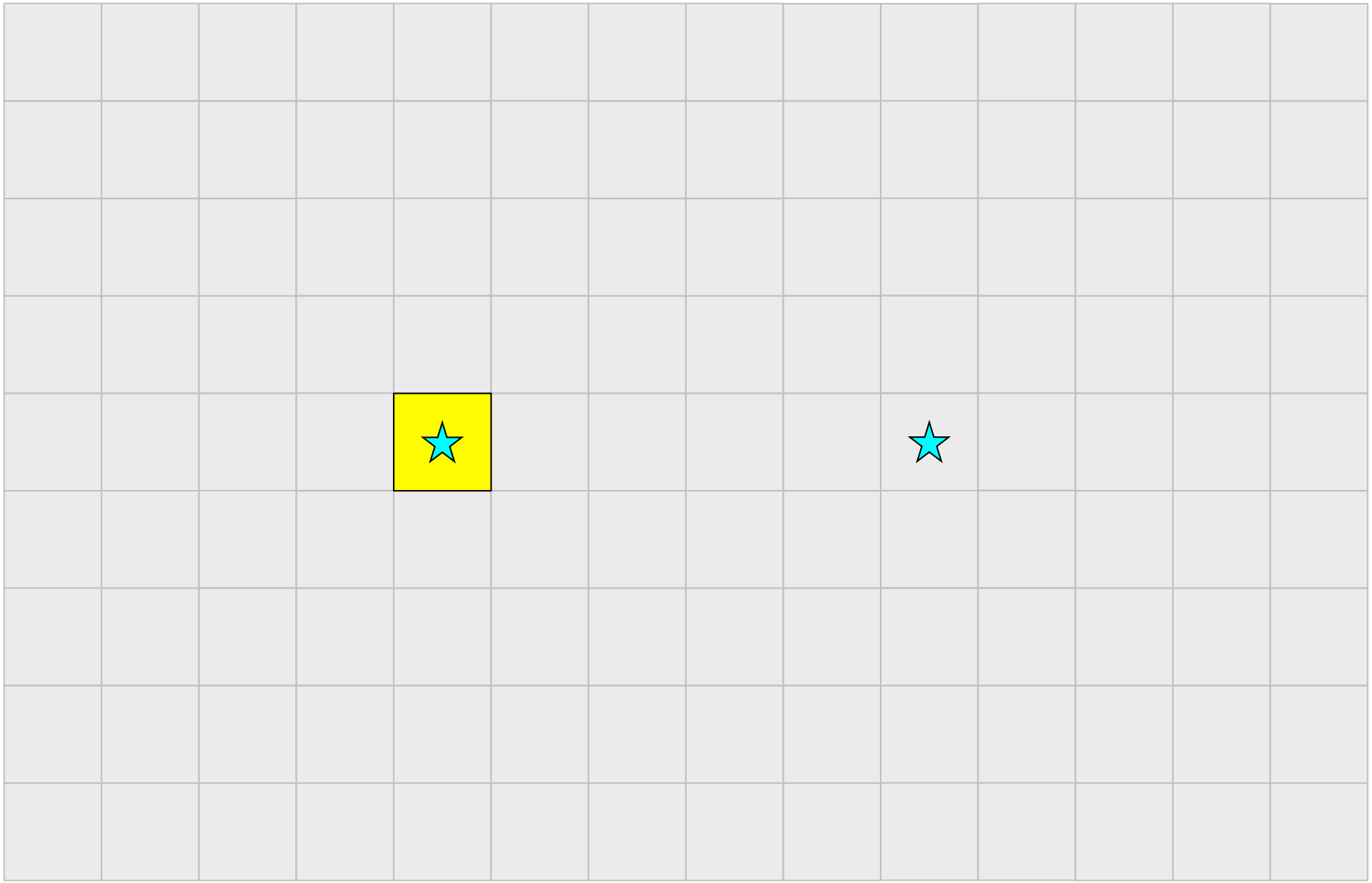
Future Cost?

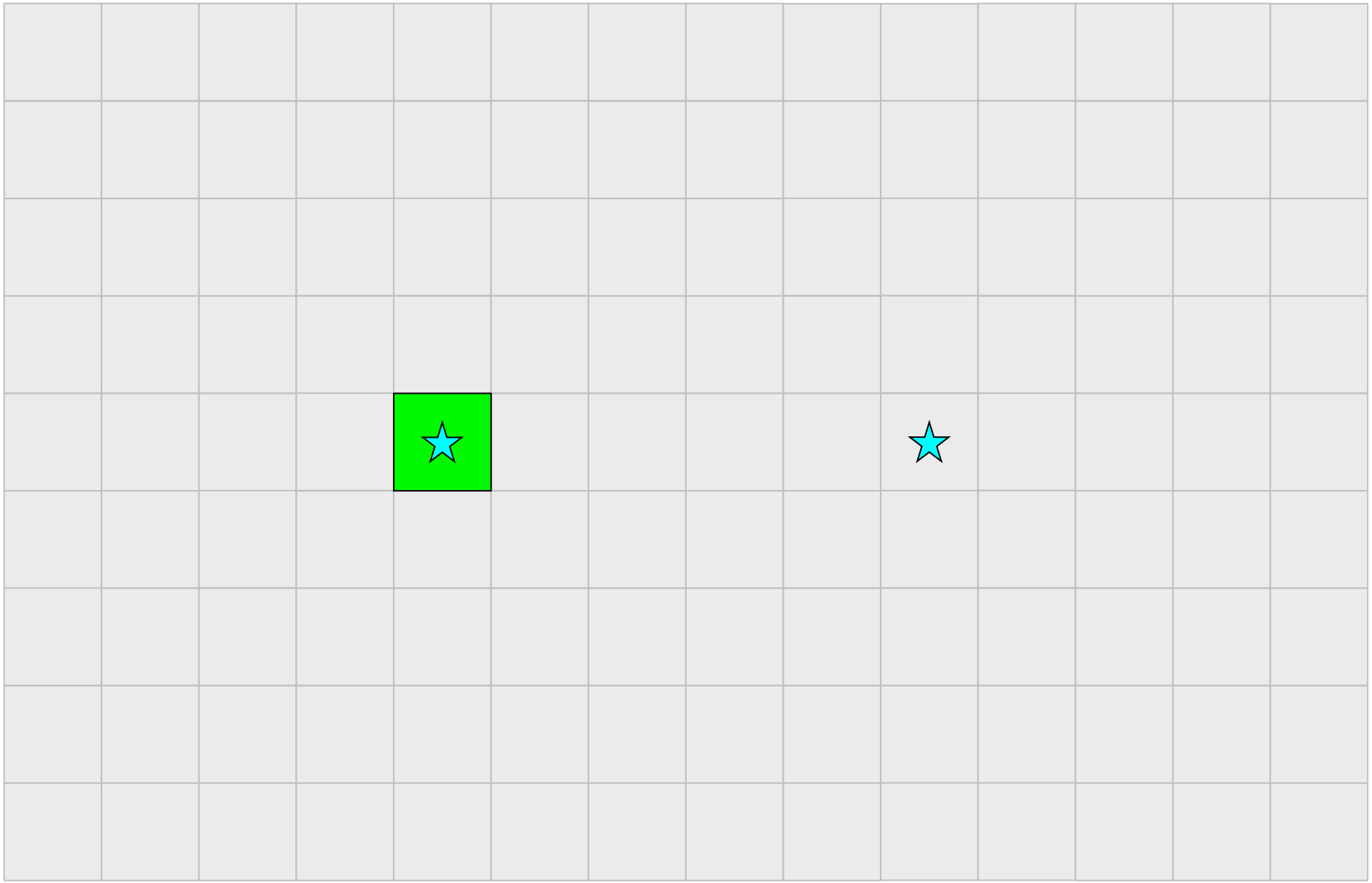


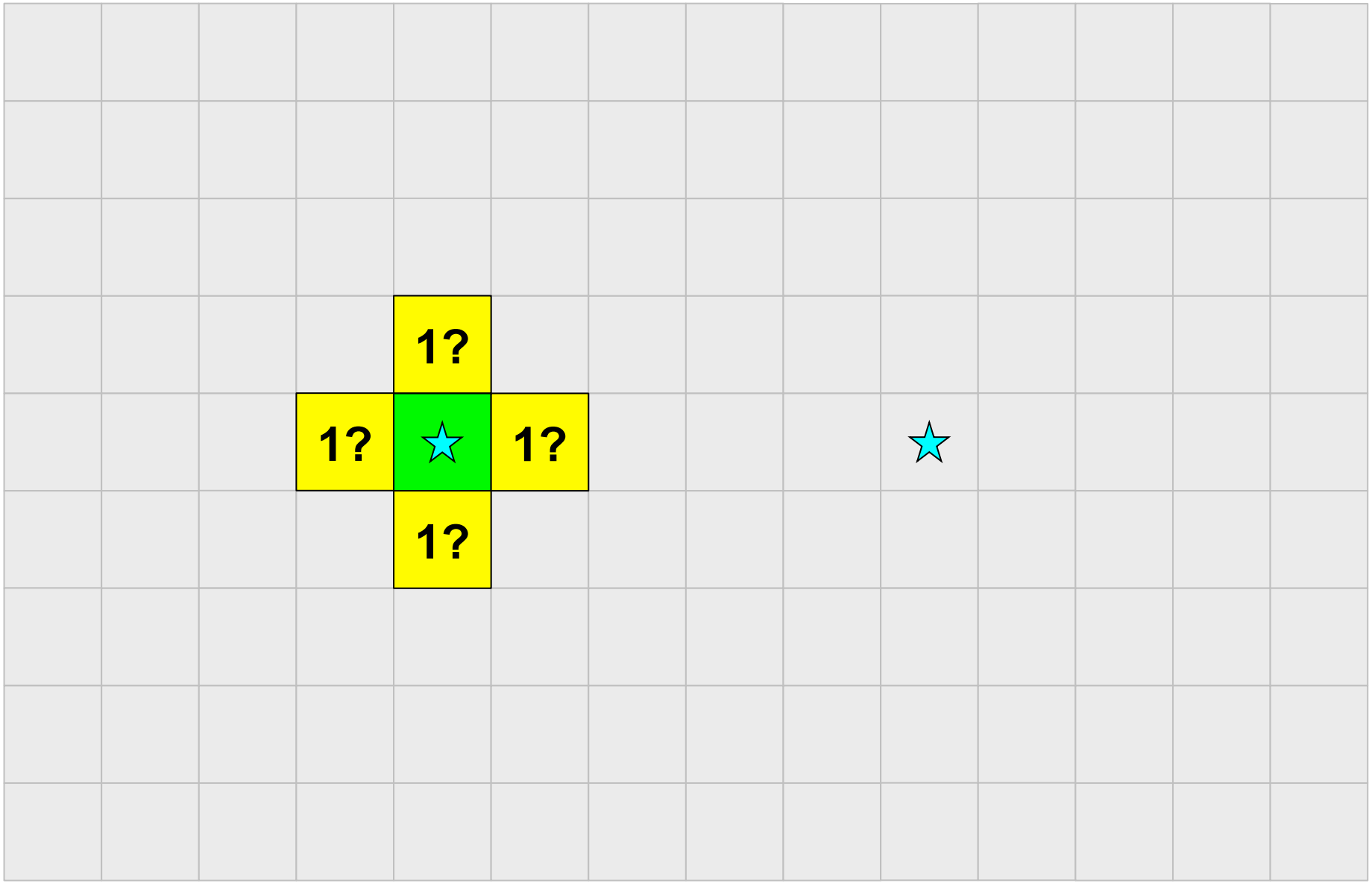
Manhattan Distance

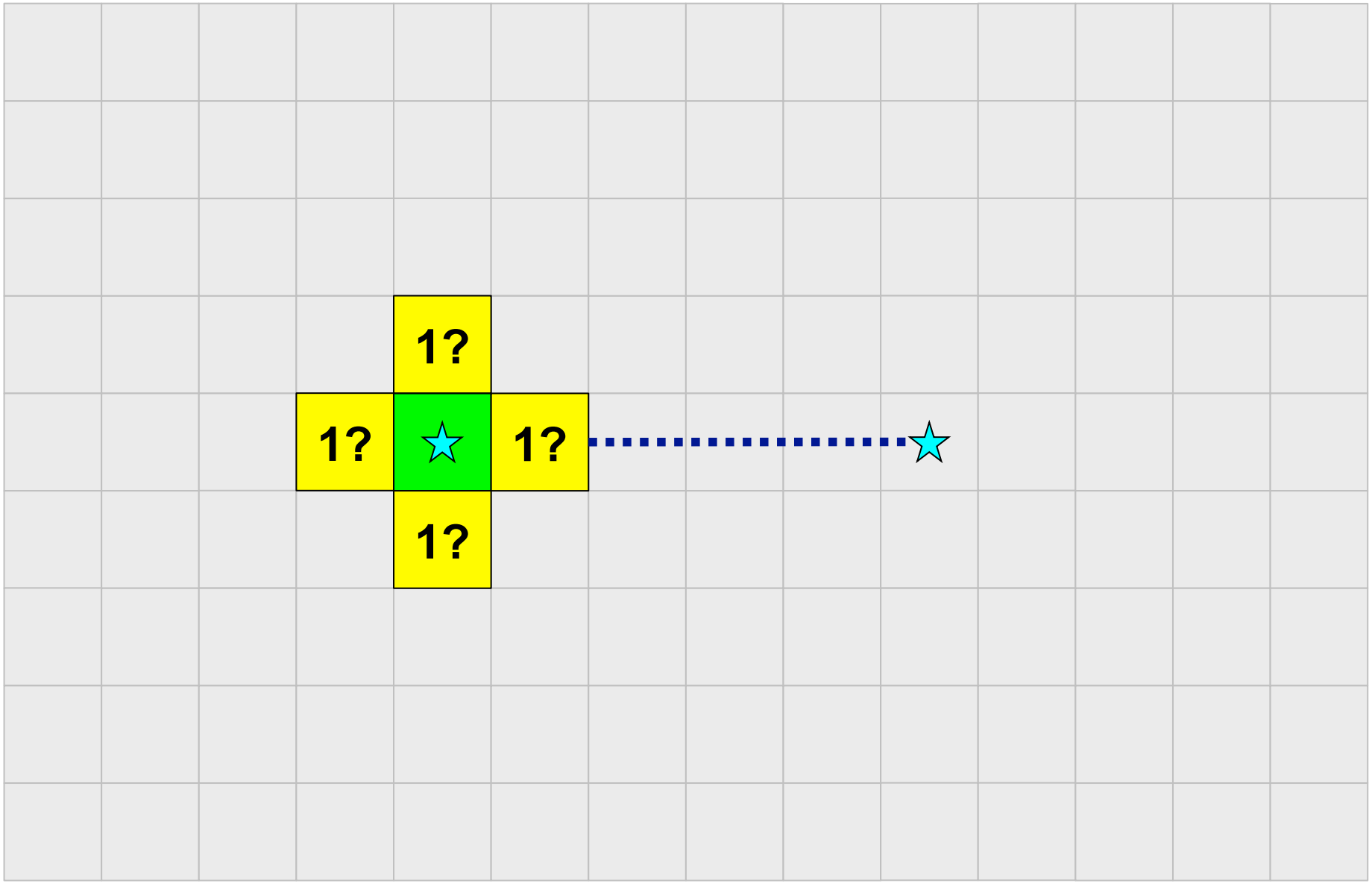
```
function h(start,goal) {
  dRows = abs(start.row - goal.row);
  dCols = abs(start.col - goal.col);
  return dRows + dCols
}
```

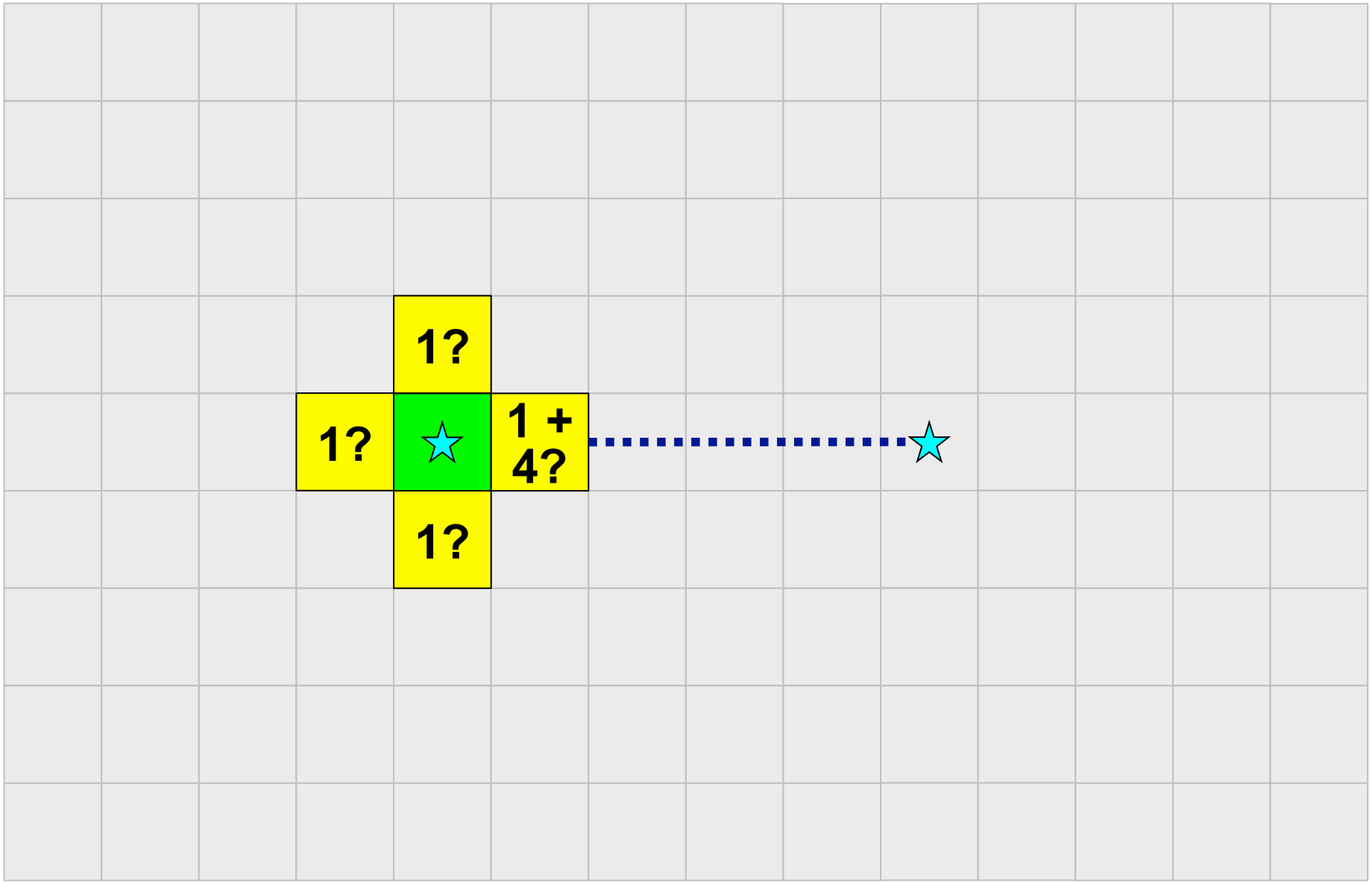


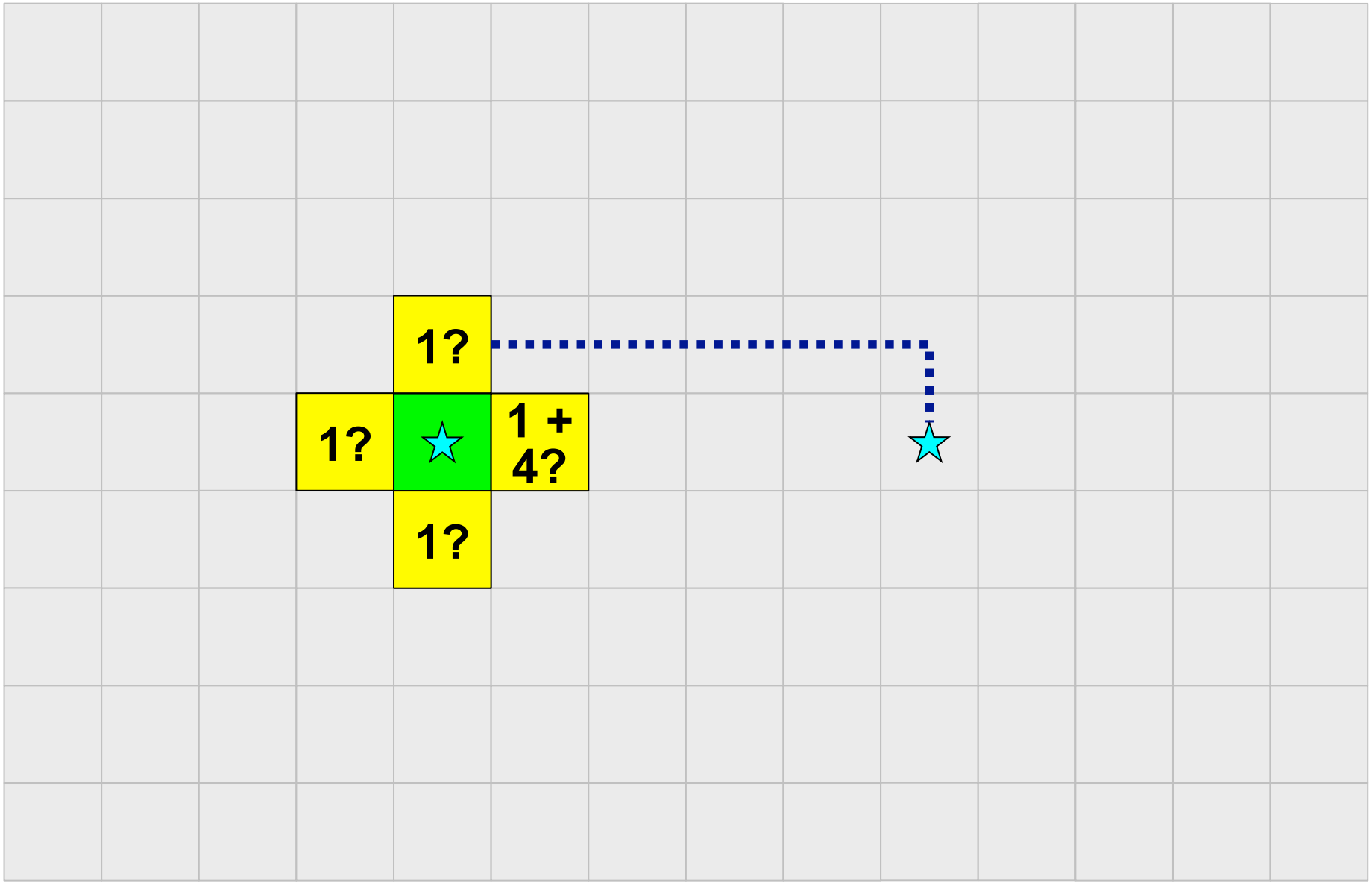


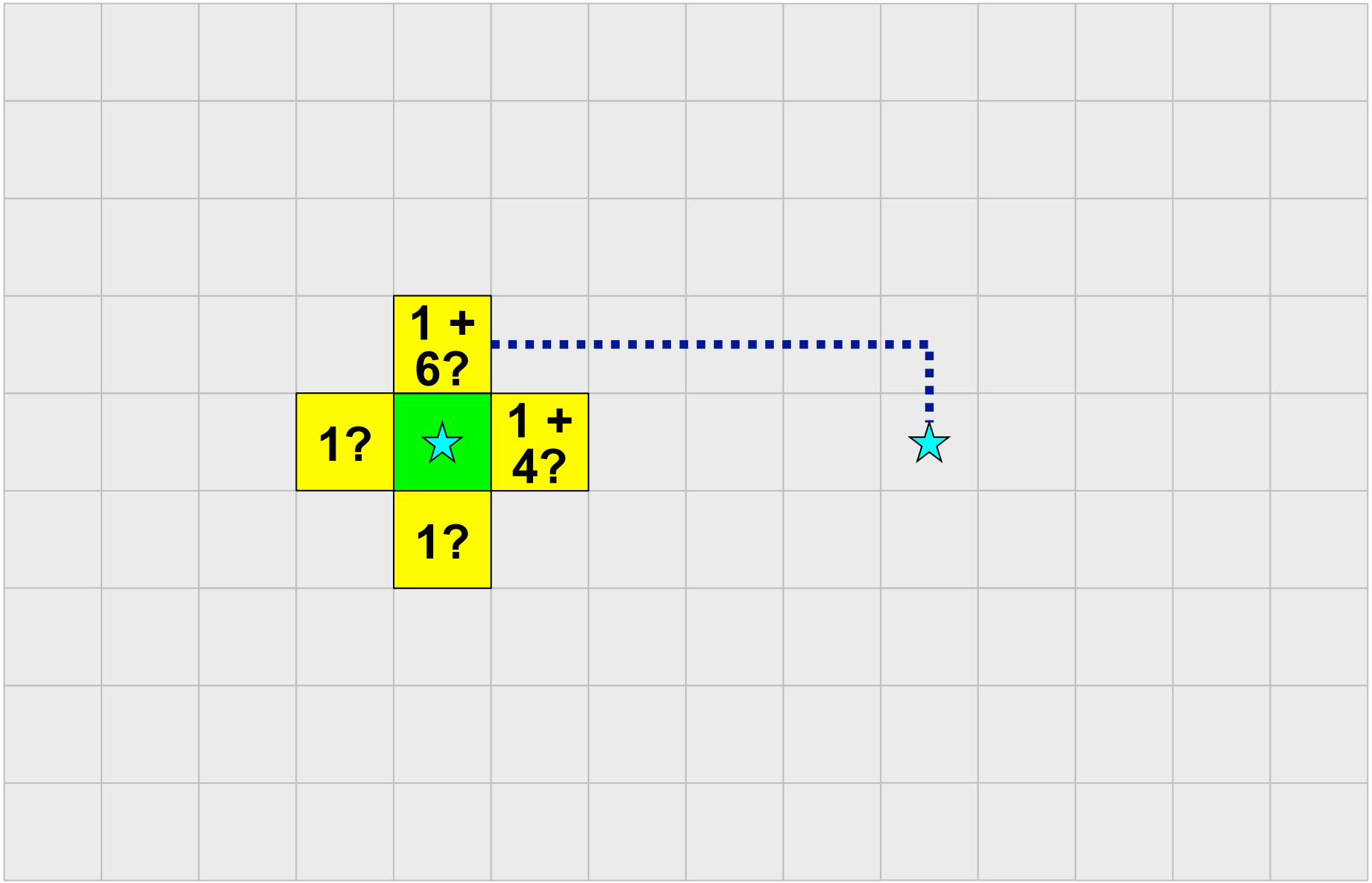


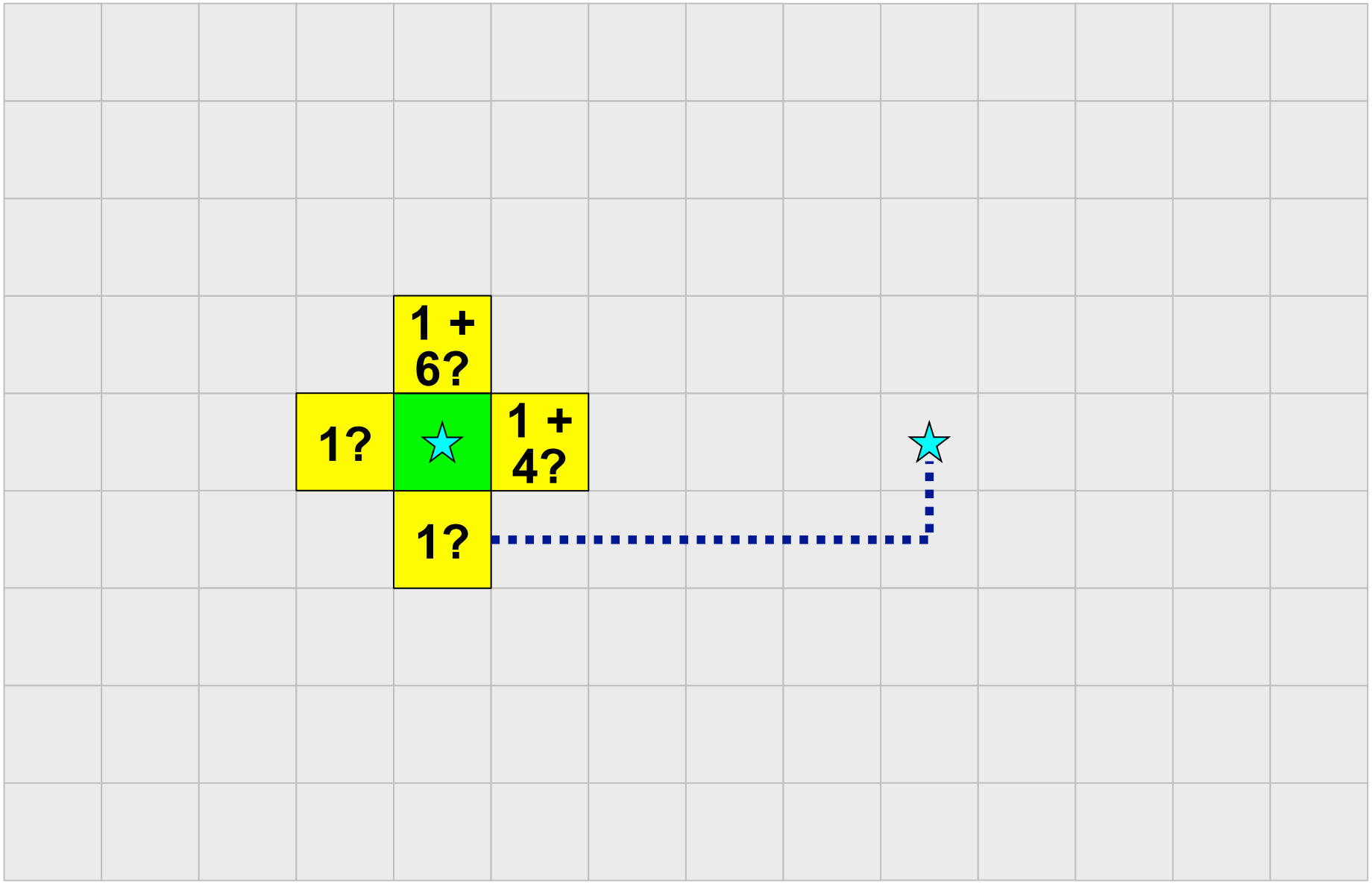

1?
1?
1?
1?

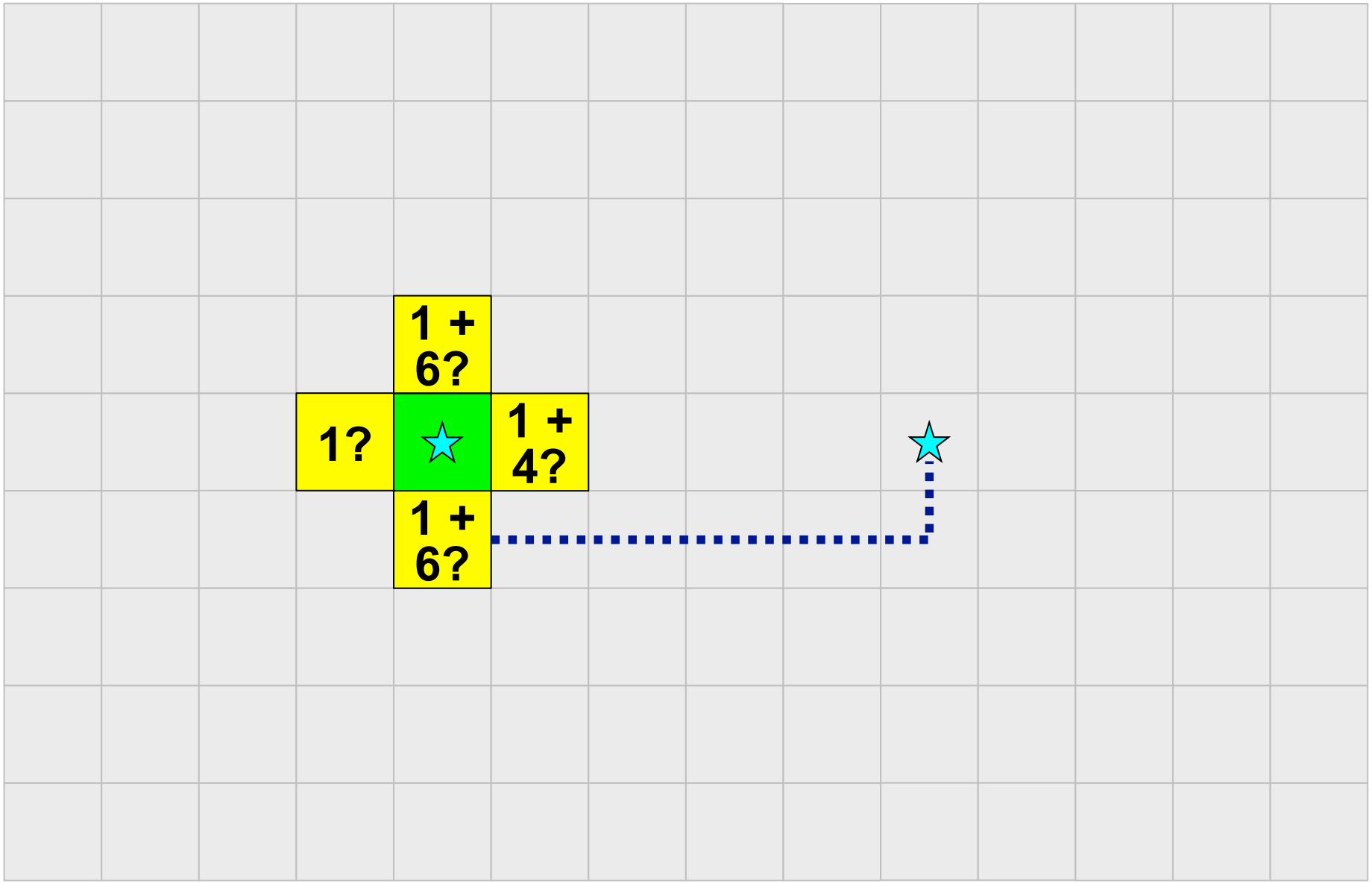

1?
1?
1?
1?

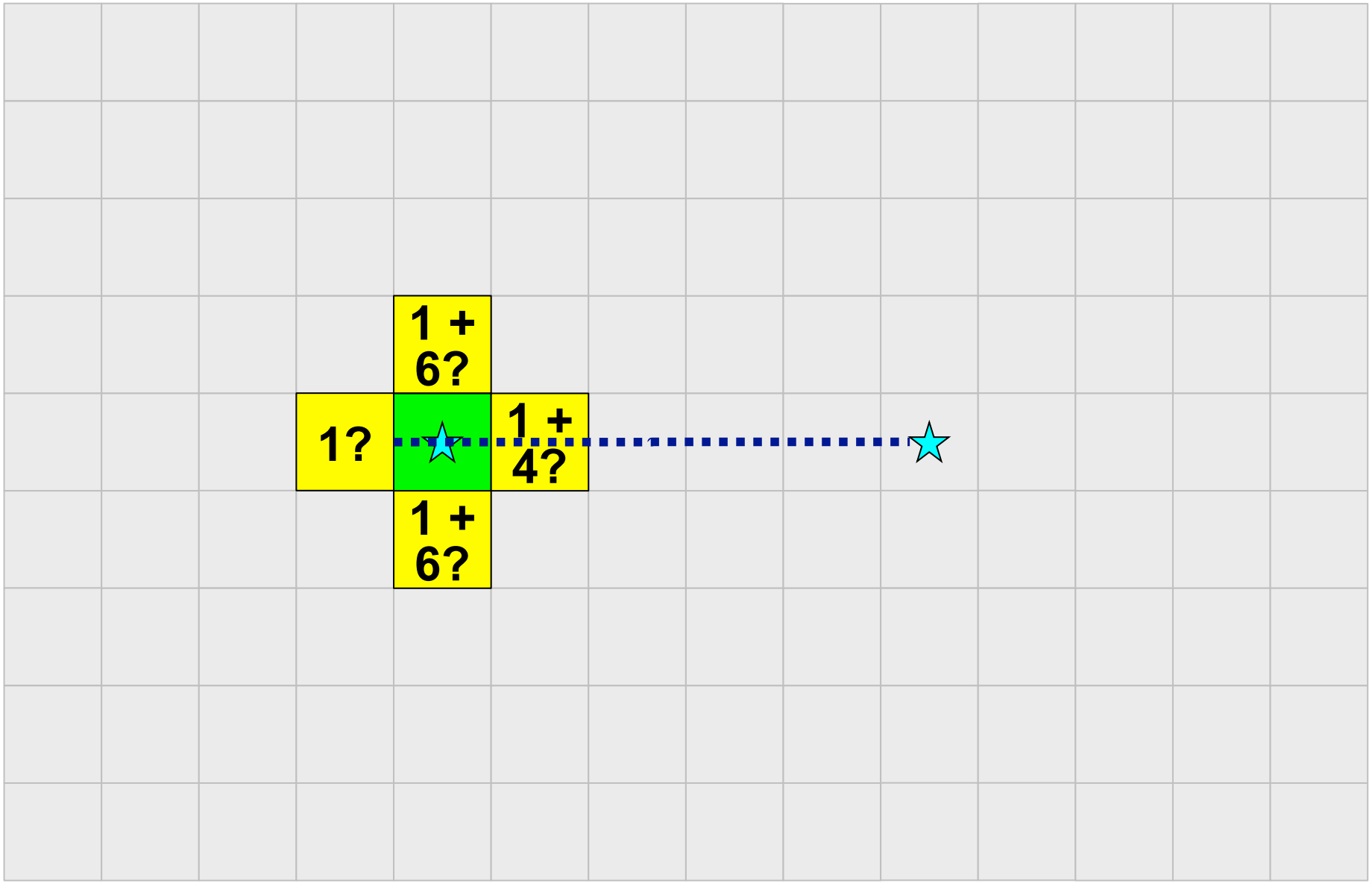

1?
1?
1 +
4?
1?

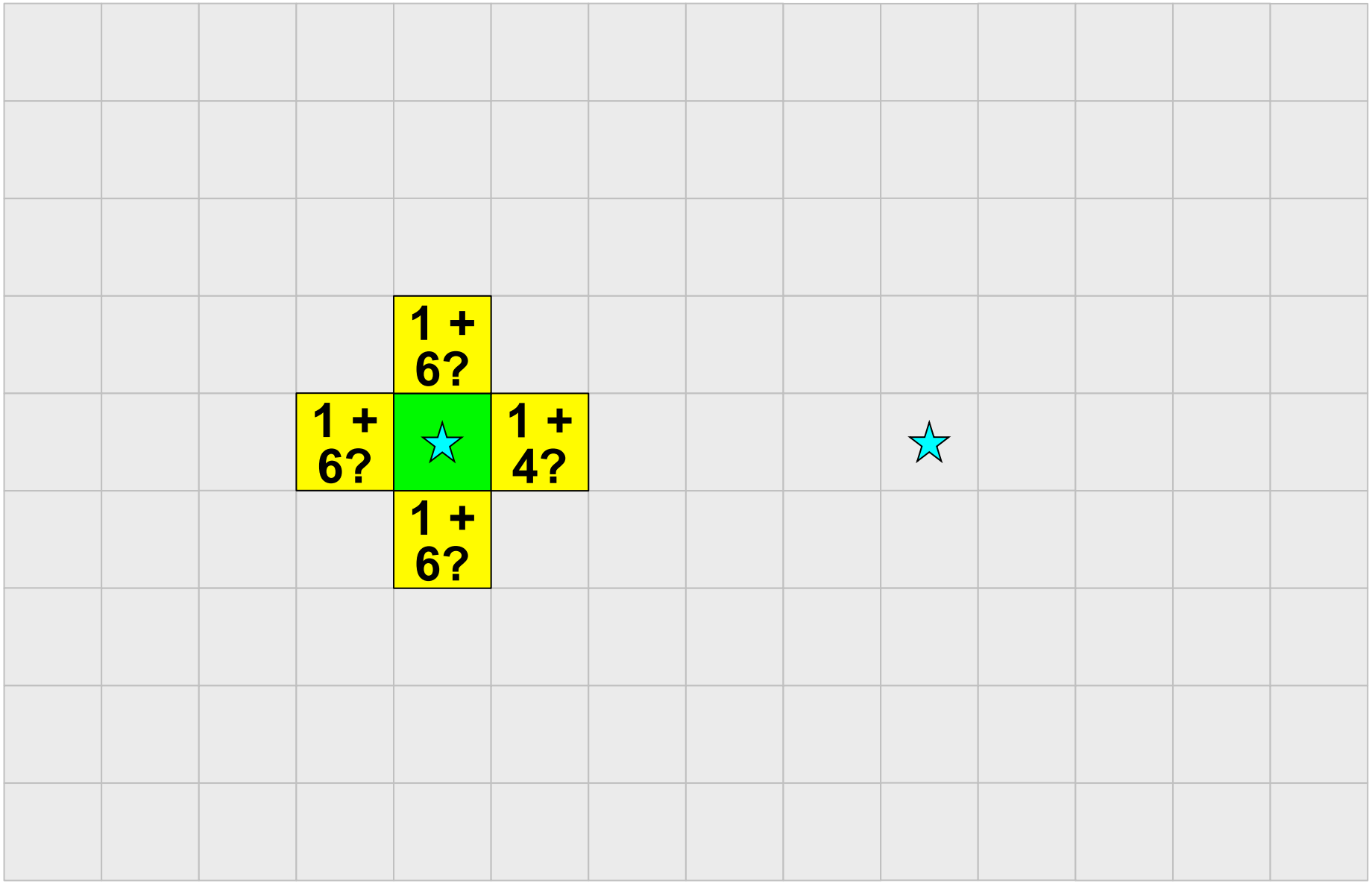

1?
1?
1 +
4?
1?

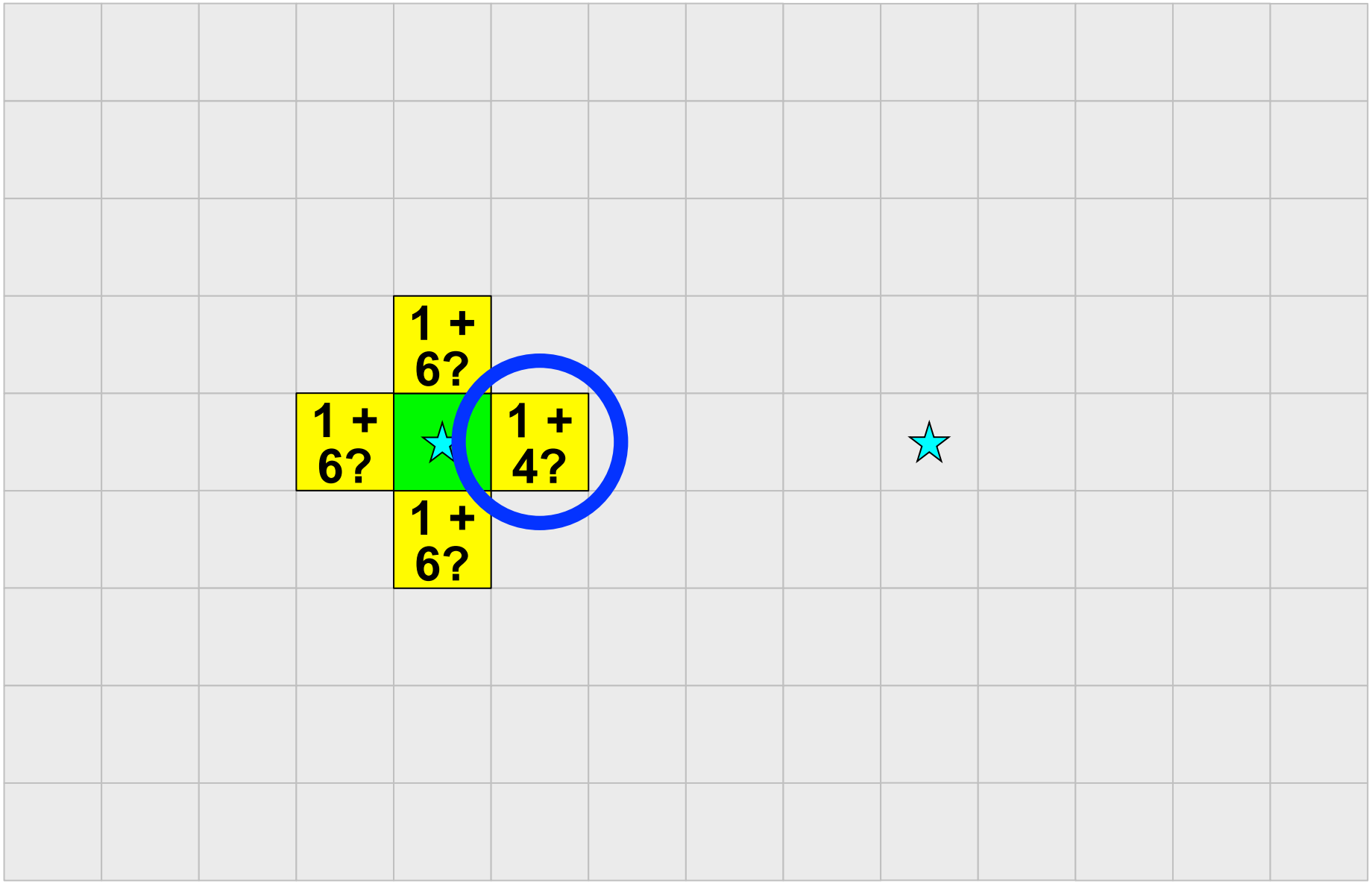

1 +
6?
1?
1 +
4?
1?

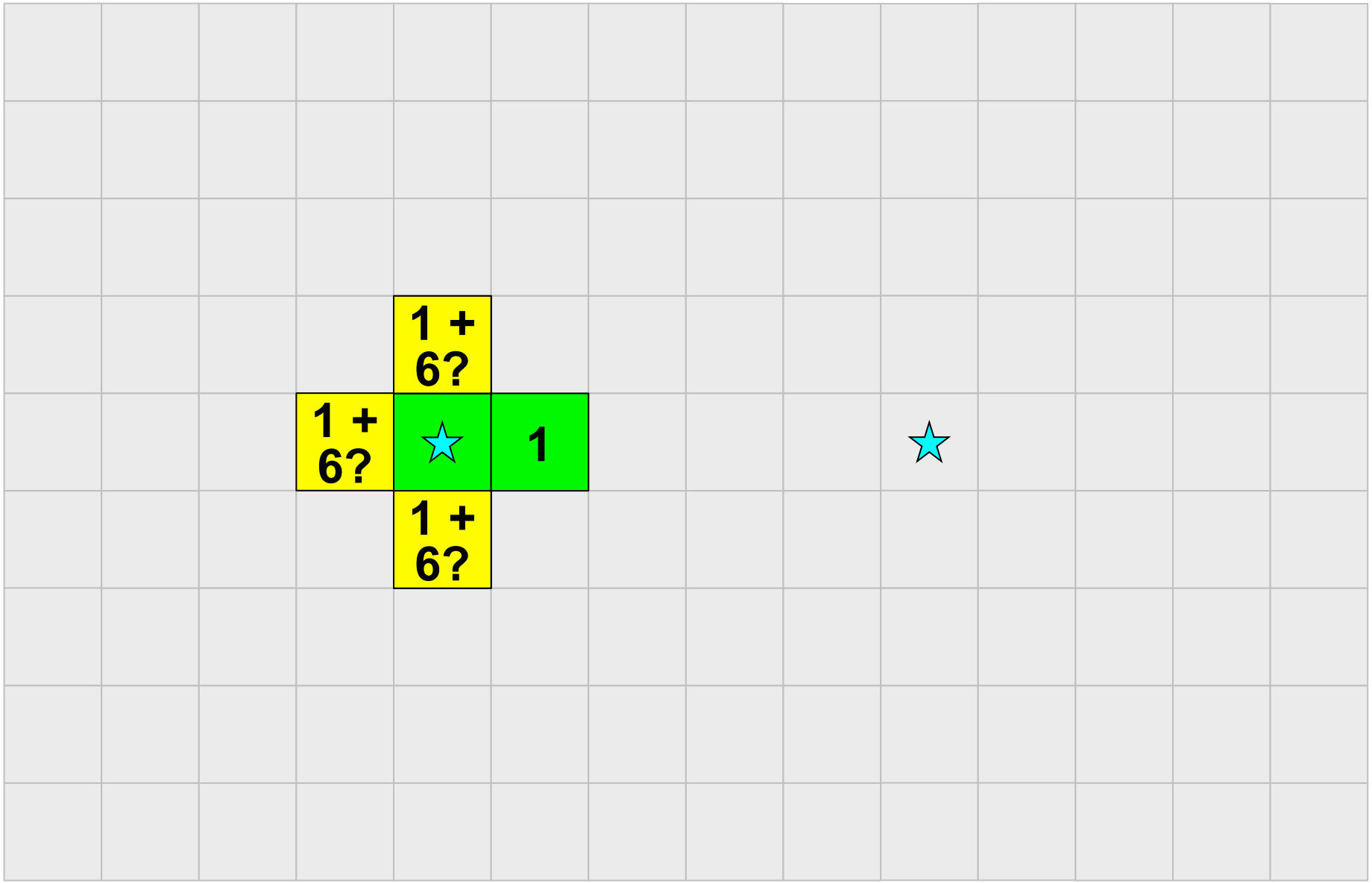

1 +
6?
1?
1 +
4?
1?

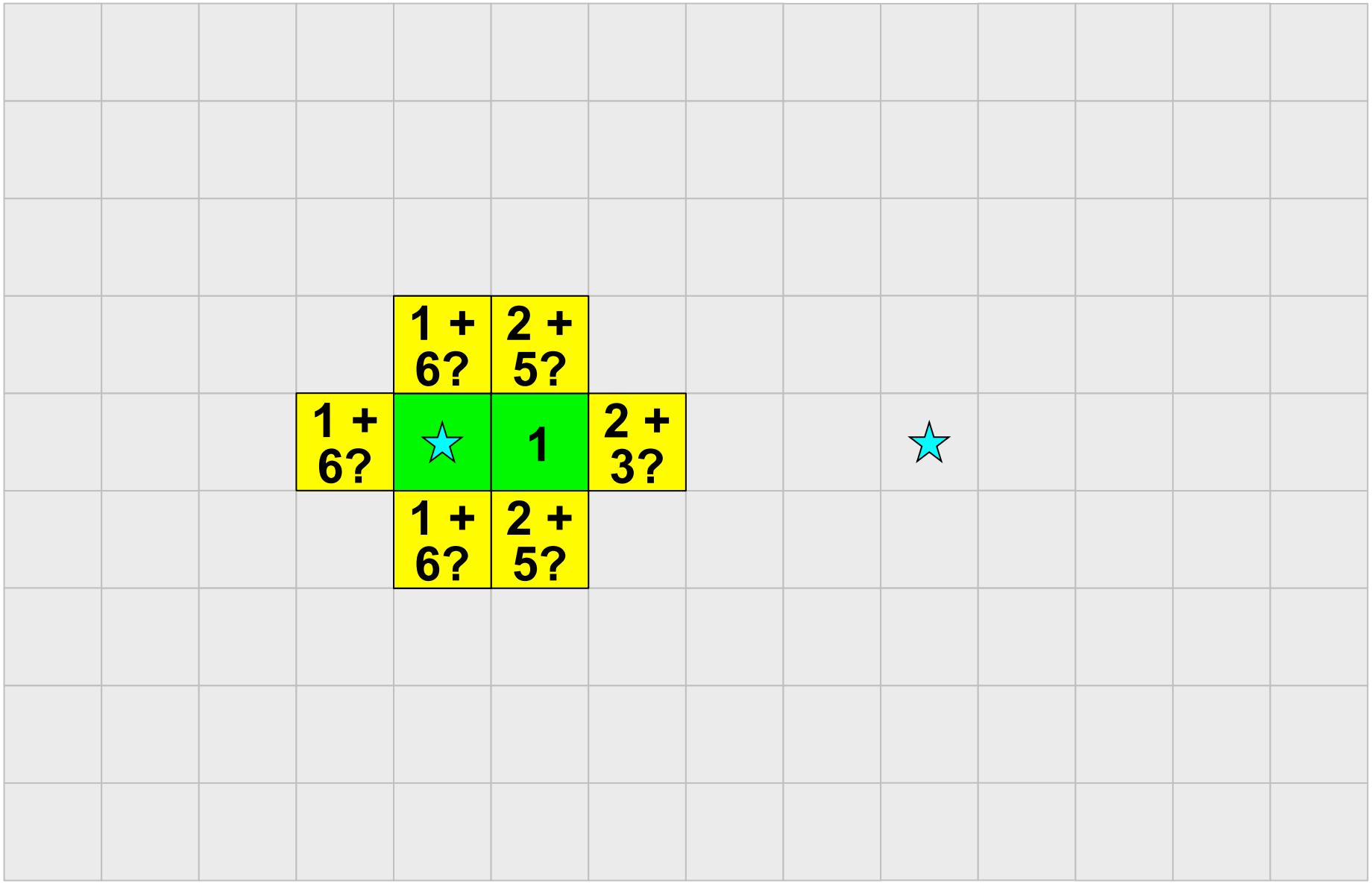

1 +
6?
1?
1 +
4?
1 +
6?

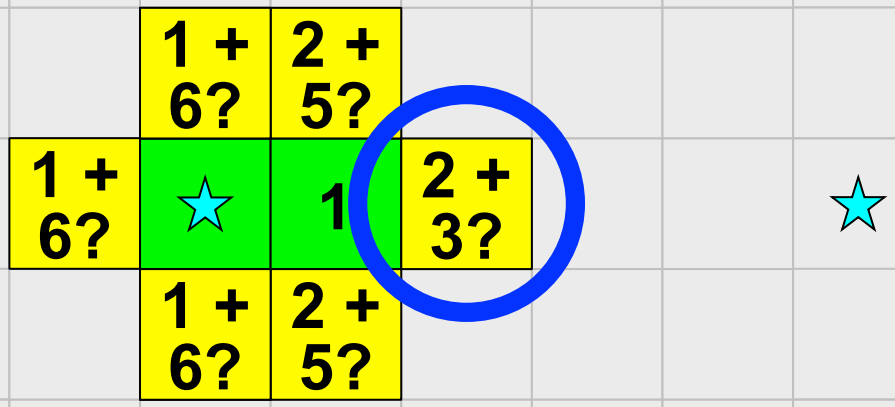

1 +
6?
1?
1 +
4?
1 +
6?

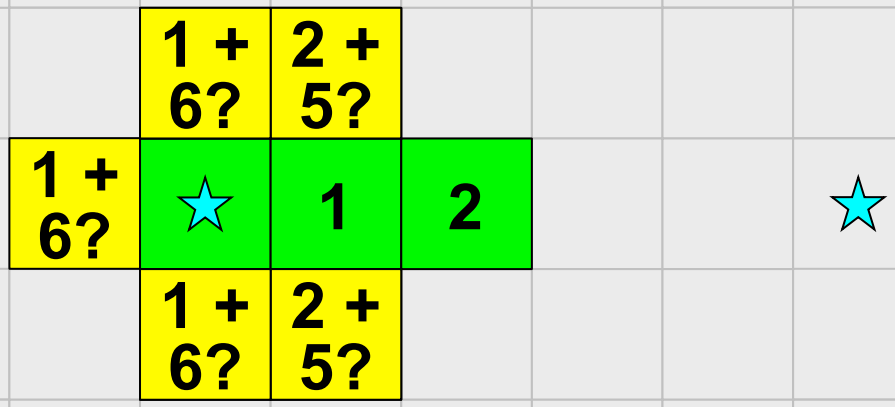

1 +
6?
1 +
6?
1 +
4?
1 +
6?

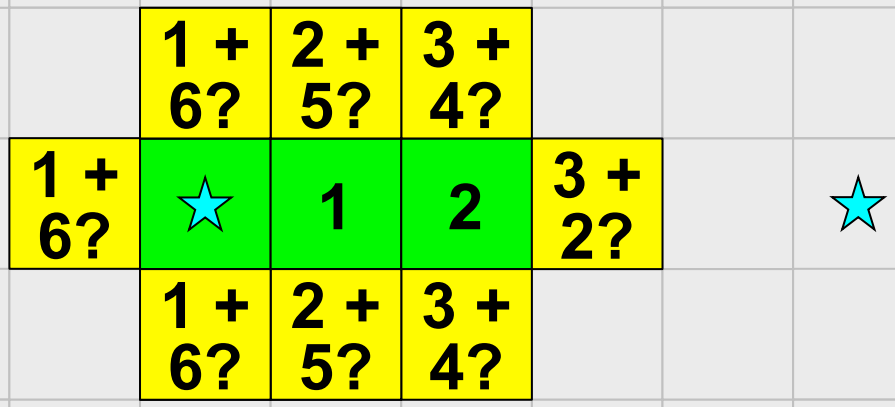

1 +
6?
1 +
6?
1 +
4?
1 +
6?

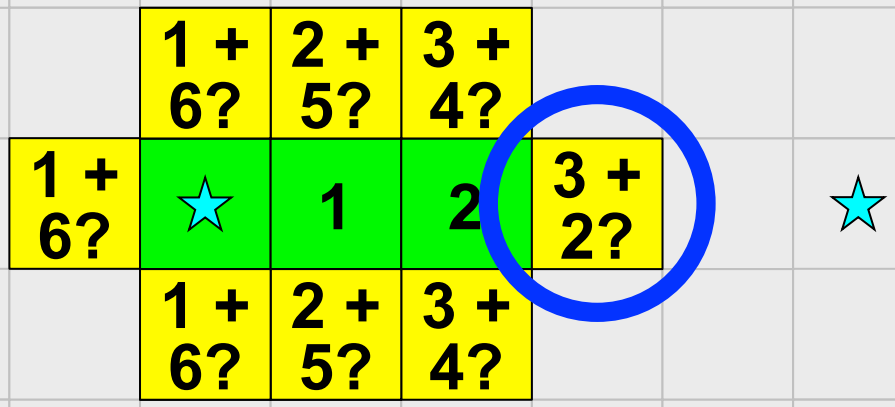

1 +
6?
1 +
6?
1
1 +
6?

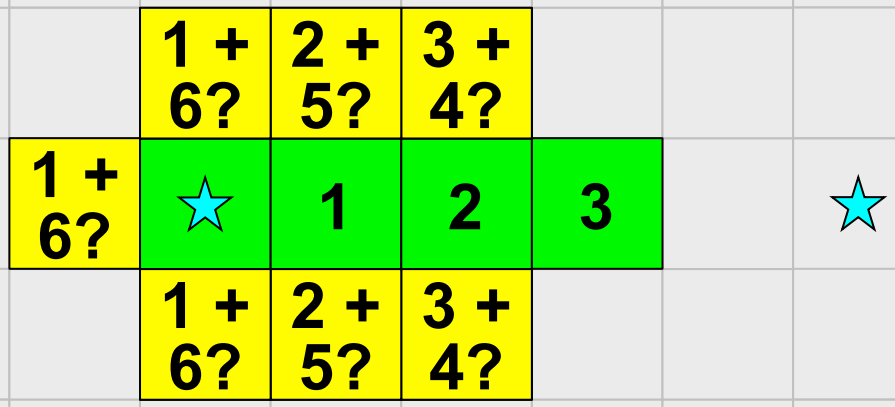

1 +
6?
2 +
5?
1 +
6?
1
2 +
3?
1 +
6?
2 +
5?


1 +
6?
2 +
5?
1 +
6?
1
2 +
3?
1 +
6?
2 +
5?

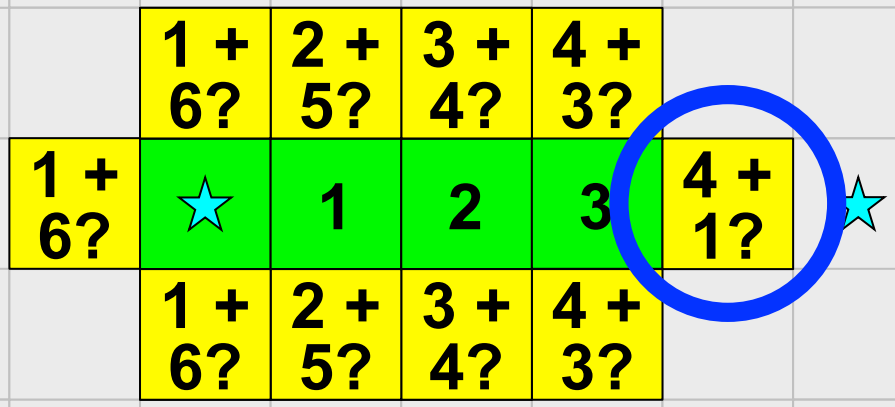

1 +
6?
2 +
5?
1 +
6?
1
2
1 +
6?
2 +
5?


1 + 6?
2 + 5?
3 + 4?
1 + 6?
1
2
3 + 2?
1 + 6?
2 + 5?
3 + 4?


1 + 6?
2 + 5?
3 + 4?
1 + 6?
1
2
3 + 2?
1 + 6?
2 + 5?
3 + 4?


1 + 6?
2 + 5?
3 + 4?
1 + 6?
1
2
3
1 + 6?
2 + 5?
3 + 4?

				1 + 6?	2 + 5?	3 + 4?	4 + 3?						
			1 + 6?	★	1	2	3	4 + 1?	★				
				1 + 6?	2 + 5?	3 + 4?	4 + 3?						


1 + 6?
2 + 5?
3 + 4?
4 + 3?
1 + 6?
1
2
3
4 + 1?
1 + 6?
2 + 5?
3 + 4?
4 + 3?

	1 + 6?	2 + 5?	3 + 4?	4 + 3?		
1 + 6?	☆	1	2	3	4	☆
	1 + 6?	2 + 5?	3 + 4?	4 + 3?		

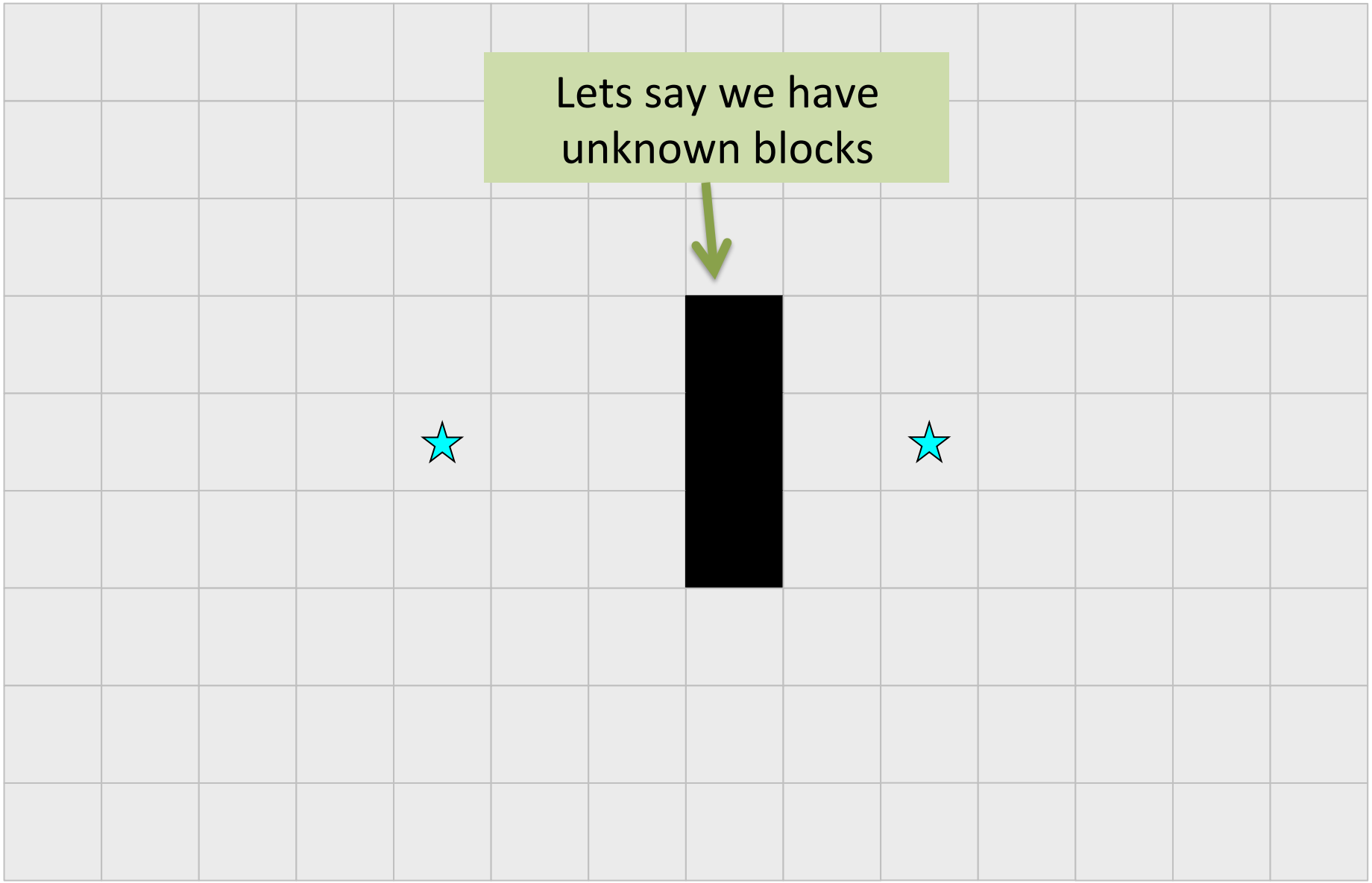
	1 + 6?	2 + 5?	3 + 4?	4 + 3?	5 + 2?	
1 + 6?	☆	1	2	3	4	5 + 0? ☆
	1 + 6?	2 + 5?	3 + 4?	4 + 3?	5 + 2?	

	1 + 6?	2 + 5?	3 + 4?	4 + 3?	5 + 2?	
1 + 6?	☆	1	2	3	4	5 + 0? ☆
	1 + 6?	2 + 5?	3 + 4?	4 + 3?	5 + 2?	

	1 + 6?	2 + 5?	3 + 4?	4 + 3?	5 + 2?	
1 + 6?	☆	1	2	3	4	☆
	1 + 6?	2 + 5?	3 + 4?	4 + 3?	5 + 2?	

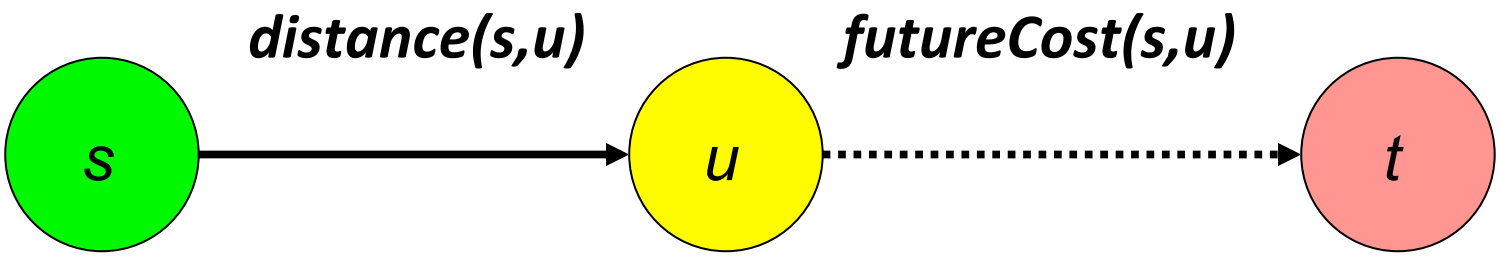
That was easy…

… a little too easy


Lets say we have
unknown blocks

A-Star Priority

priority(u) = distance(s, u) + futureCost(u, t)

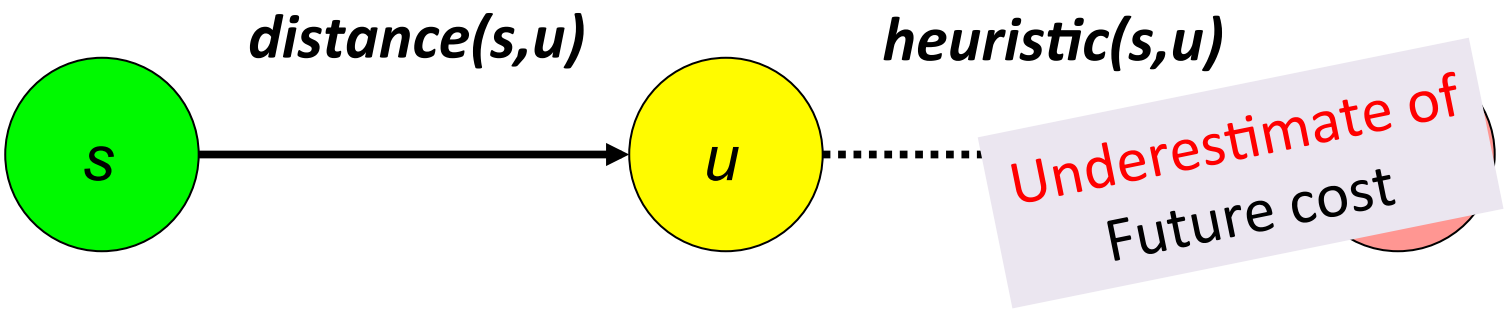


Priority of the path that ends in u

*note we will revise this slightly

A-Star Insight

priority(u) = distance(s, u) + heuristic(u, t)



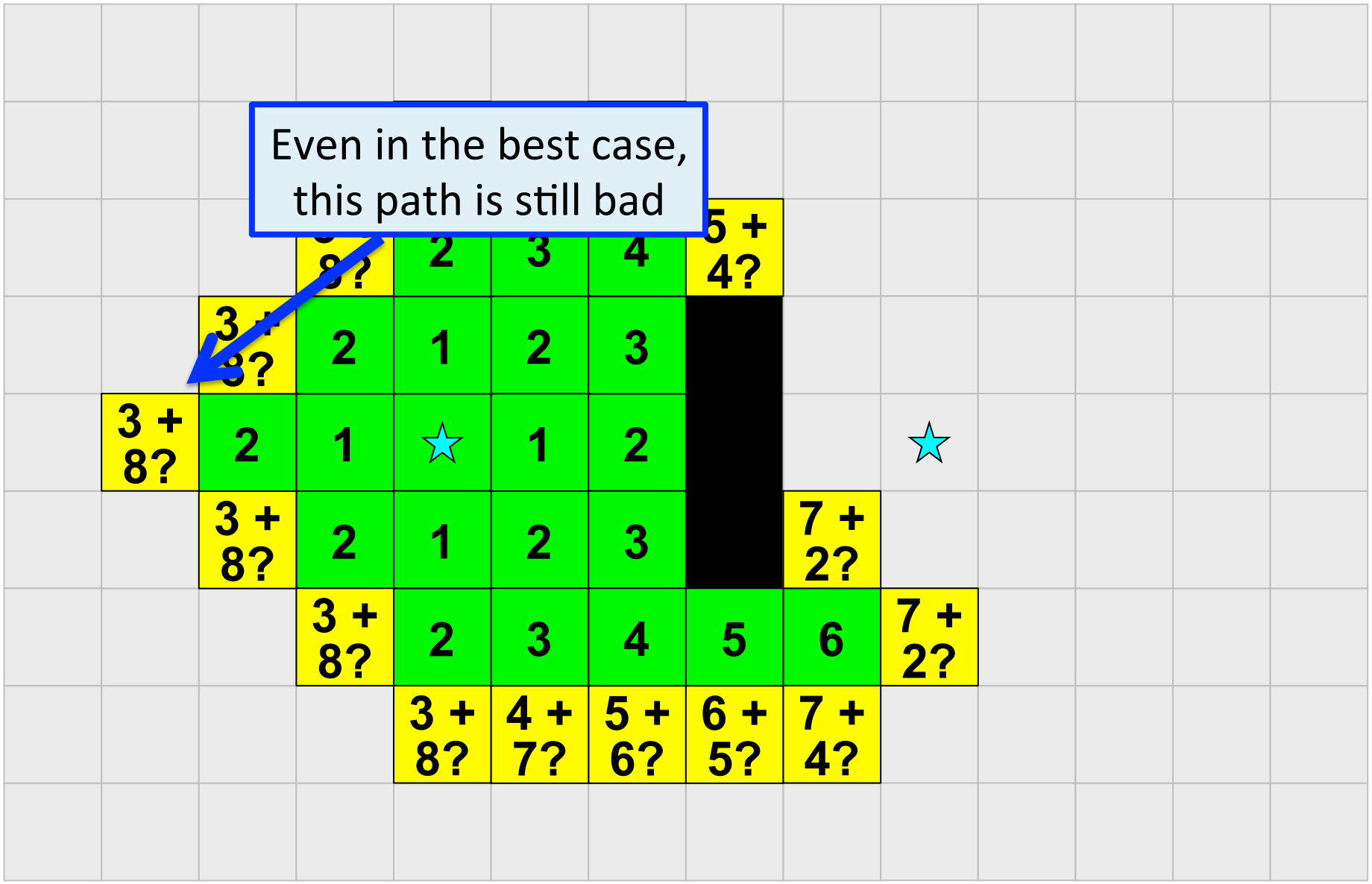
Priority of the path that ends in u

Admissible Heuristic

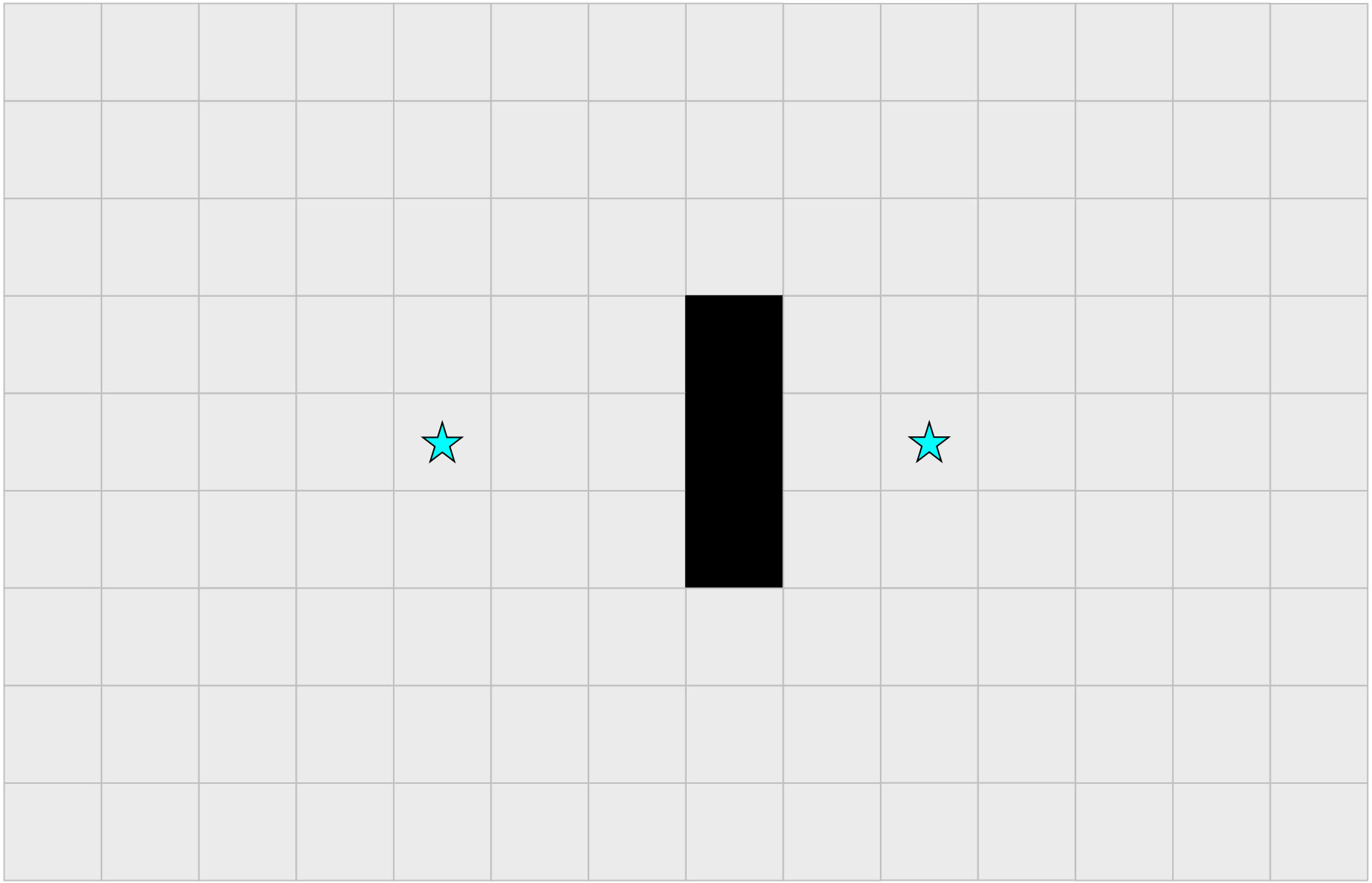
Definition: An admissible heuristic always underestimates the true cost.

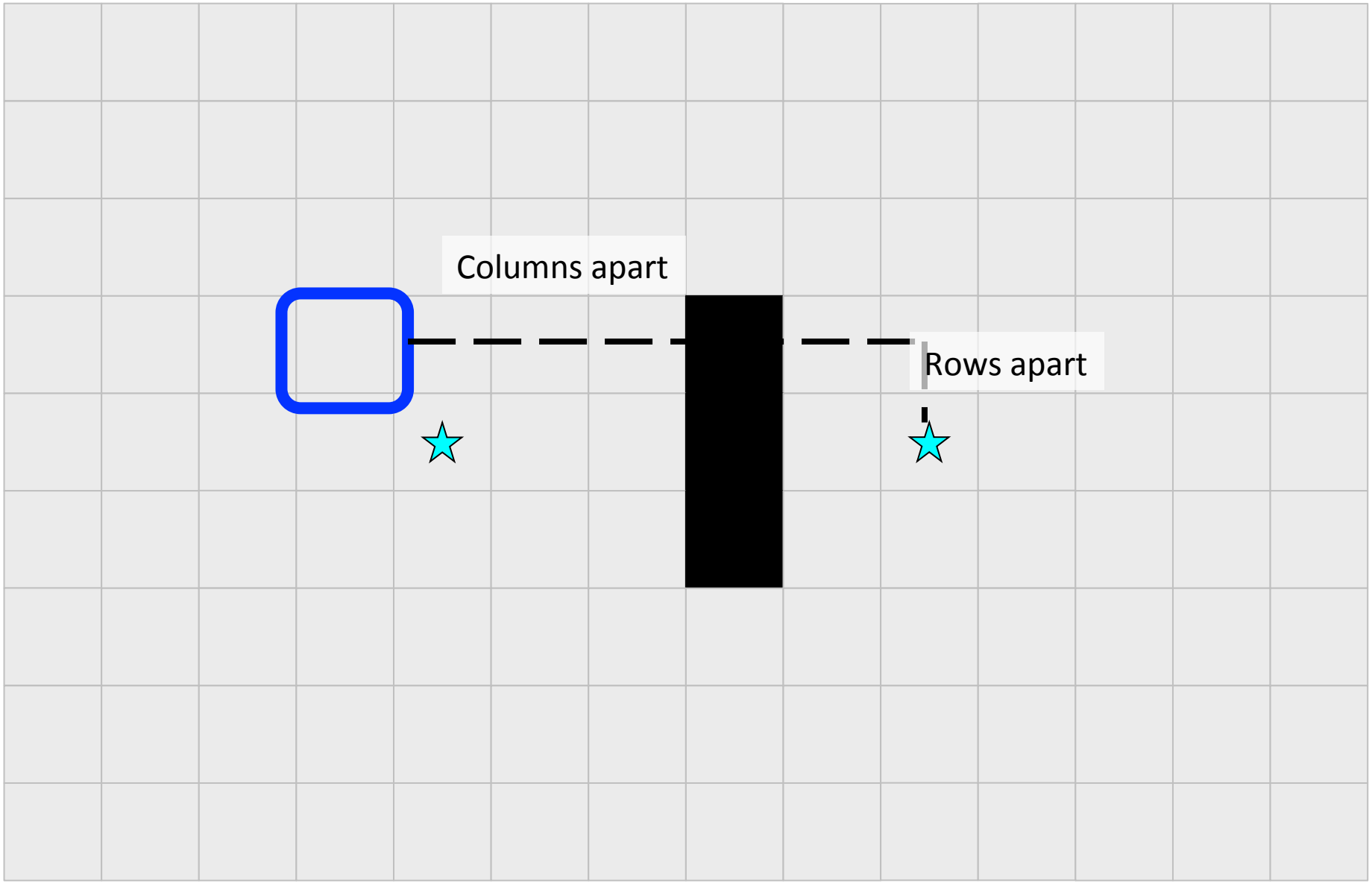
Thus: "even in the best case scenario, this path is still terrible…"

Admissible Heuristic



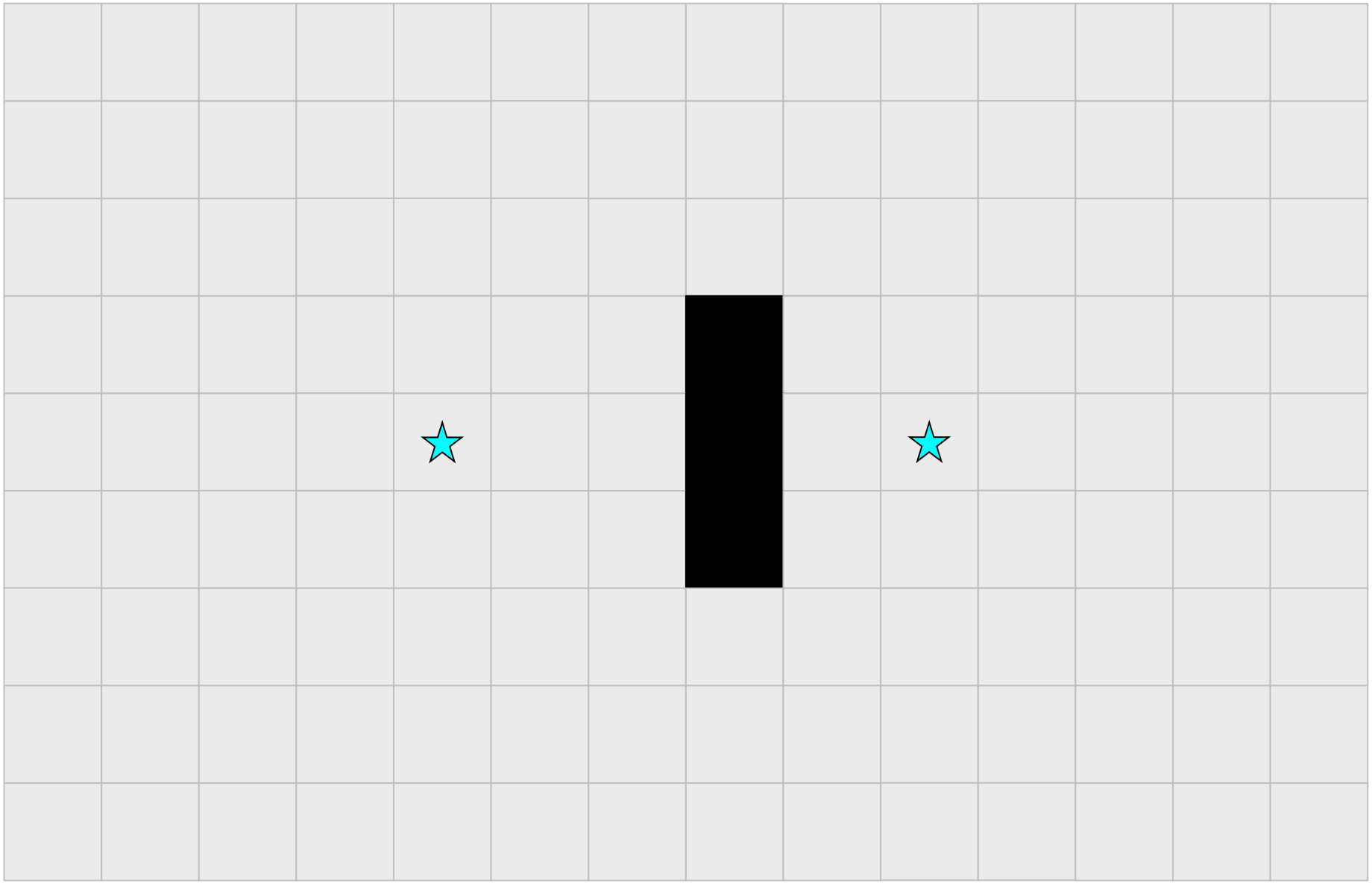
Just Relax

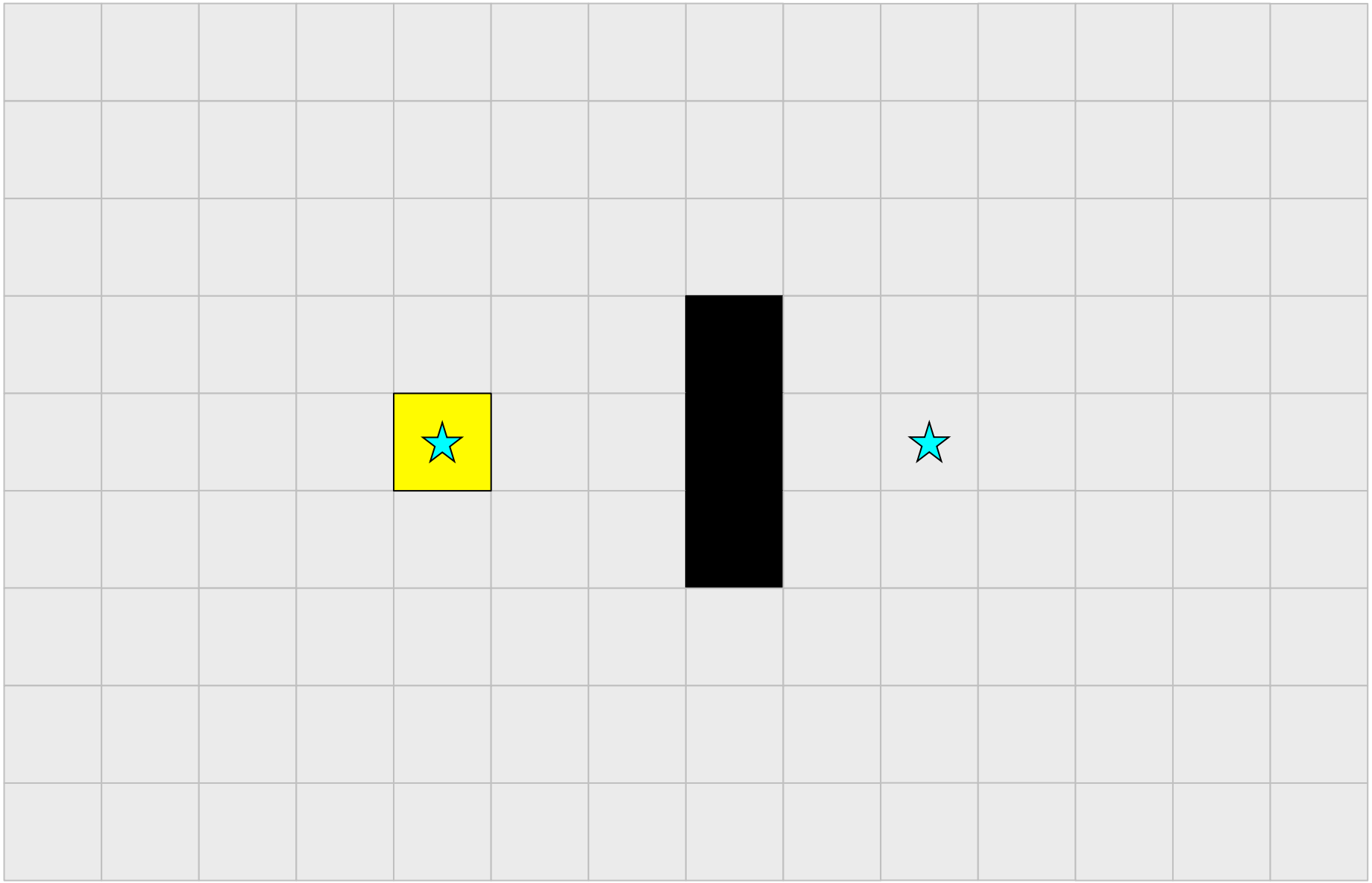


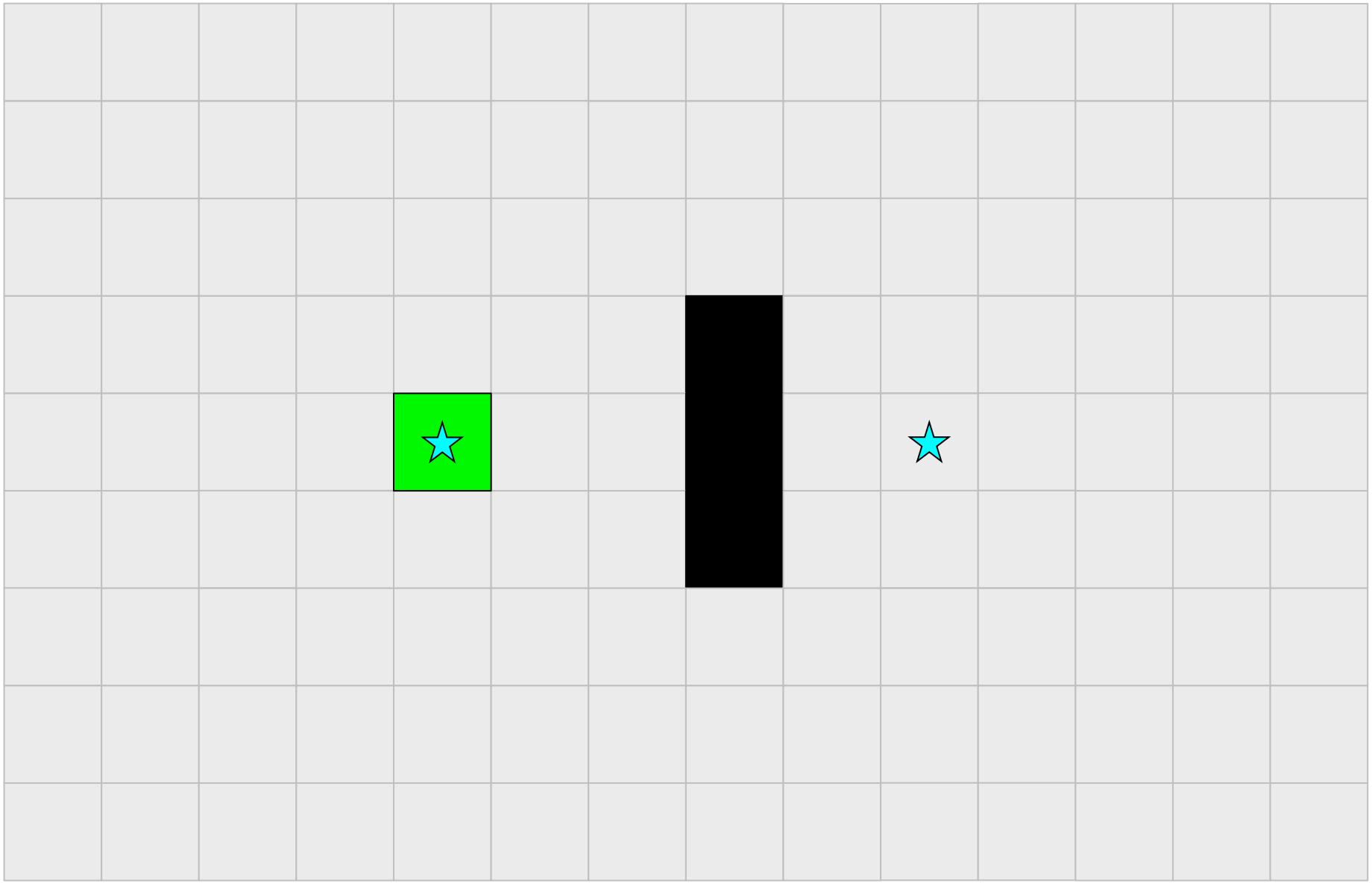

Columns apart
Rows apart

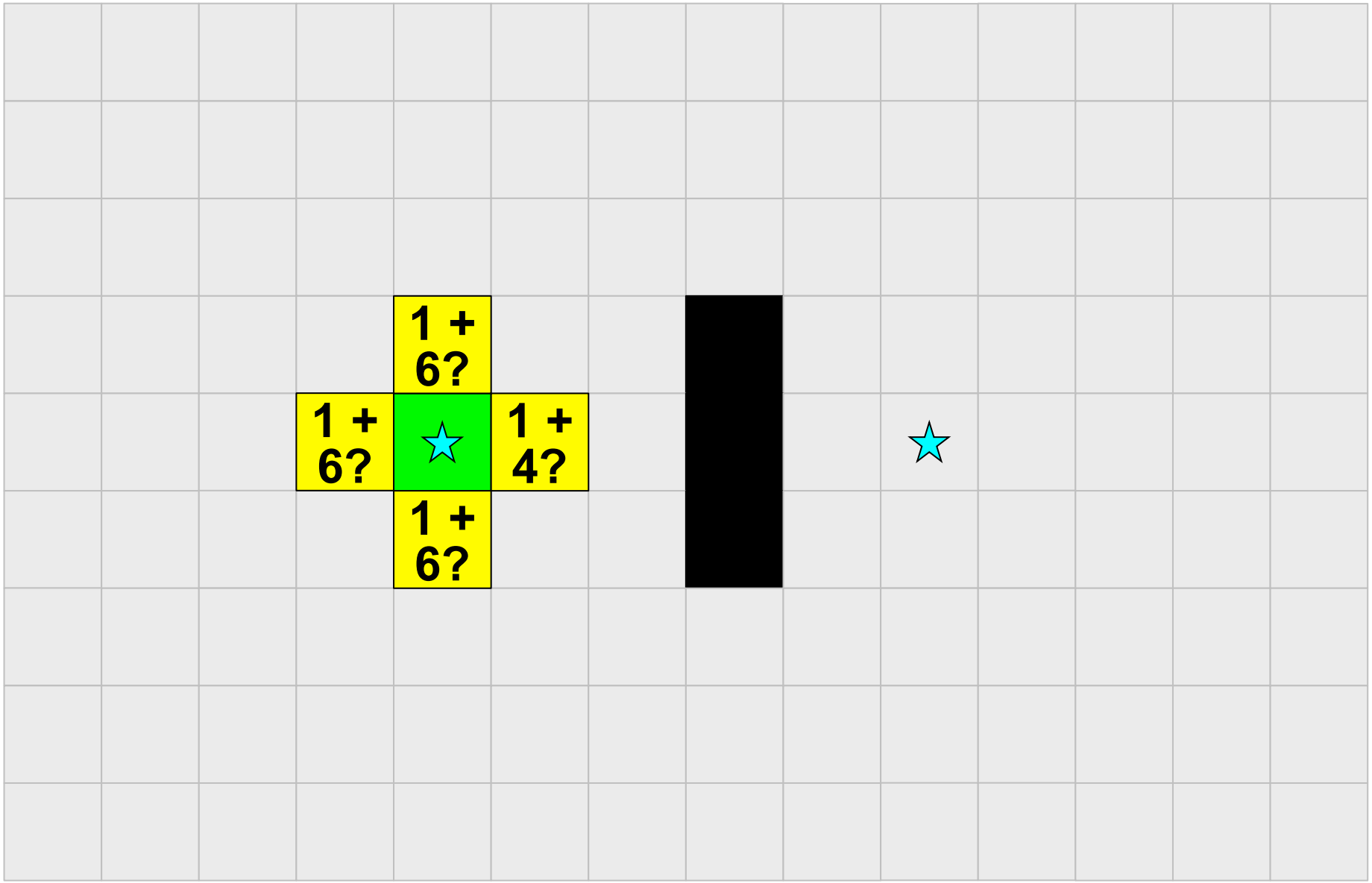
Manhattan Distance

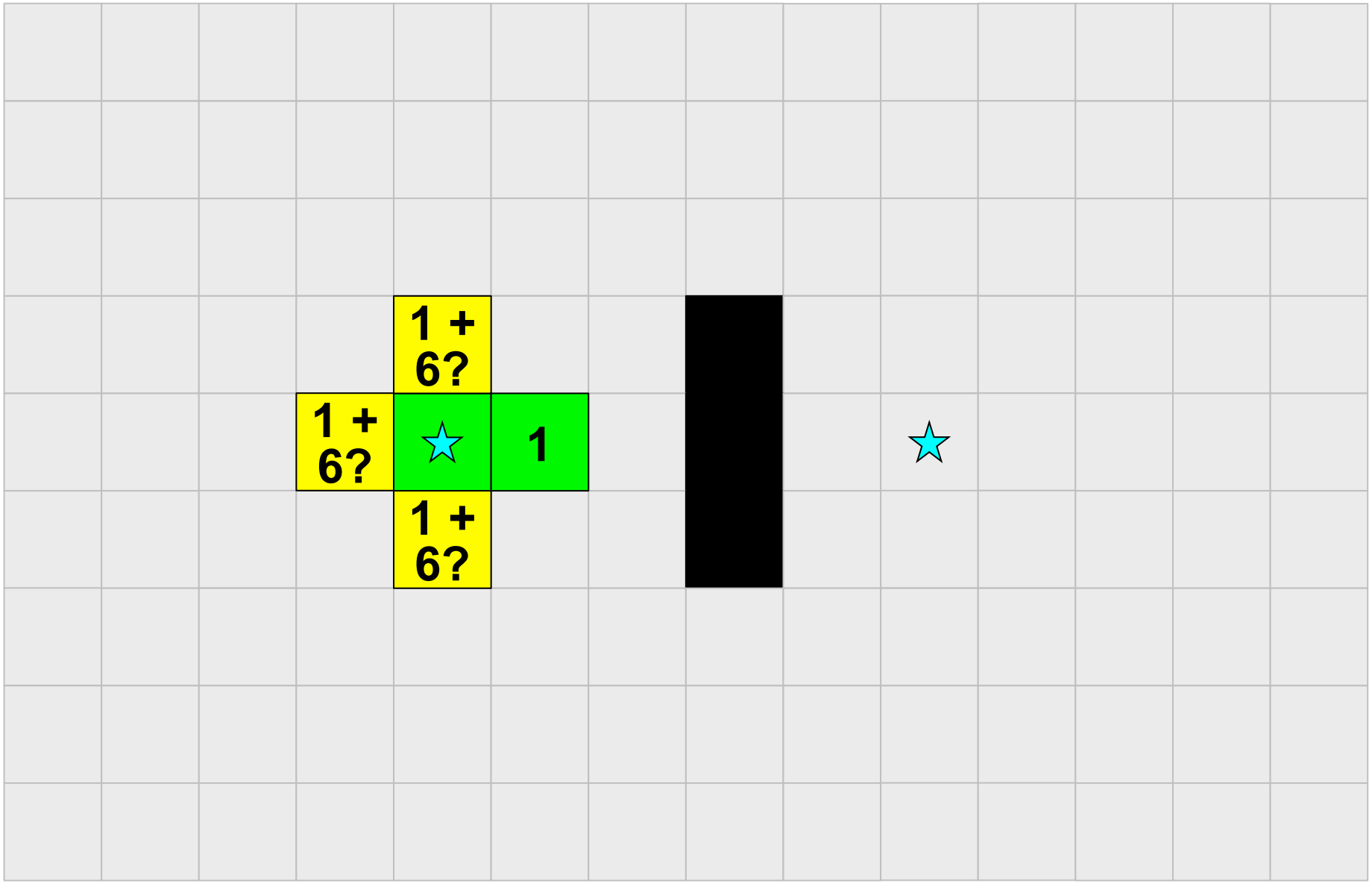
```
function h(start,goal) {
  dRows = abs(start.row - goal.row);
  dCols = abs(start.col - goal.col);
  return dRows + dCols
}
```

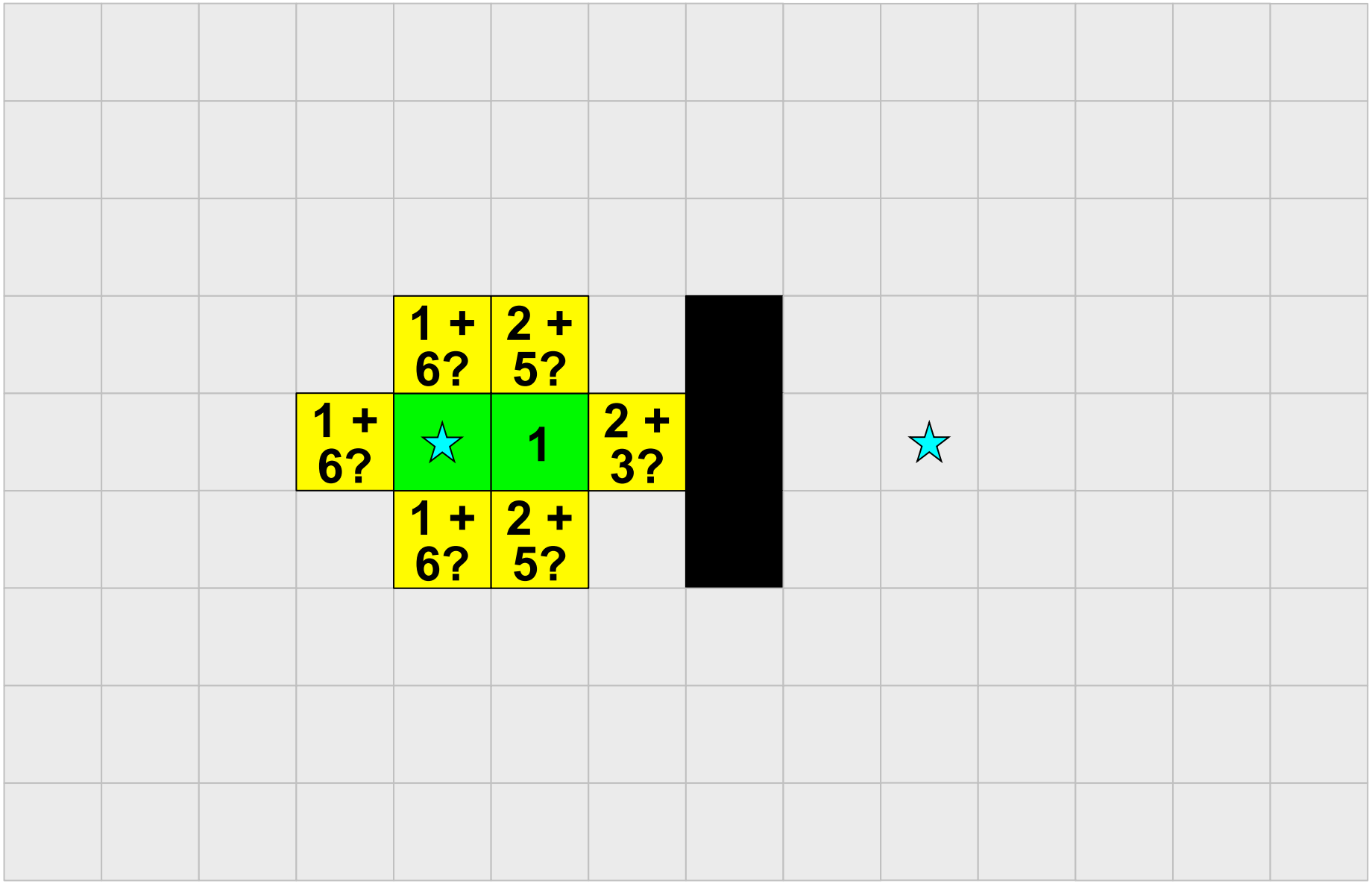


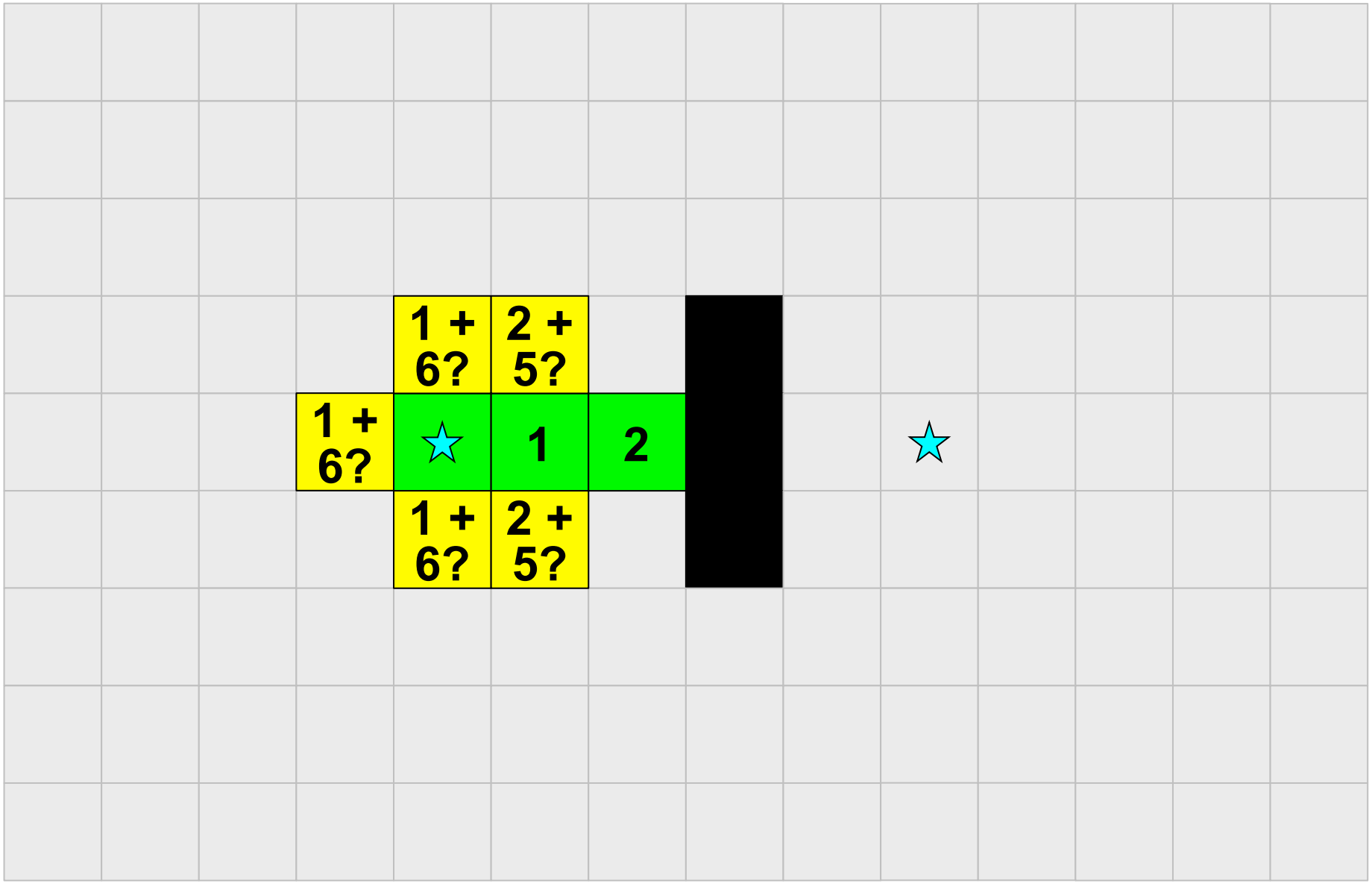


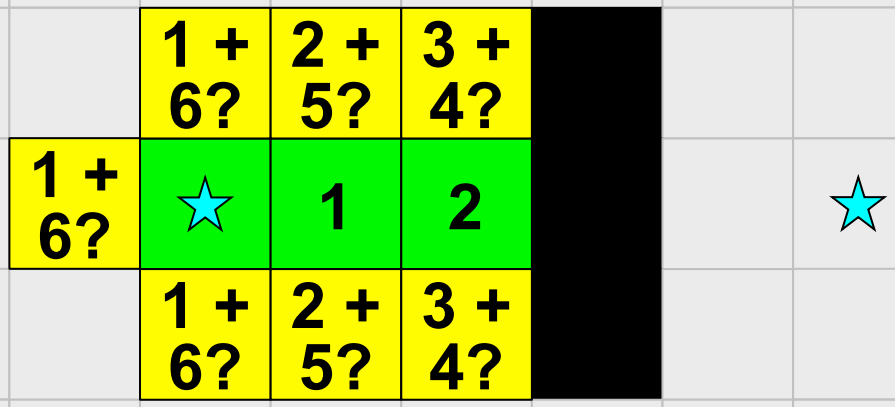


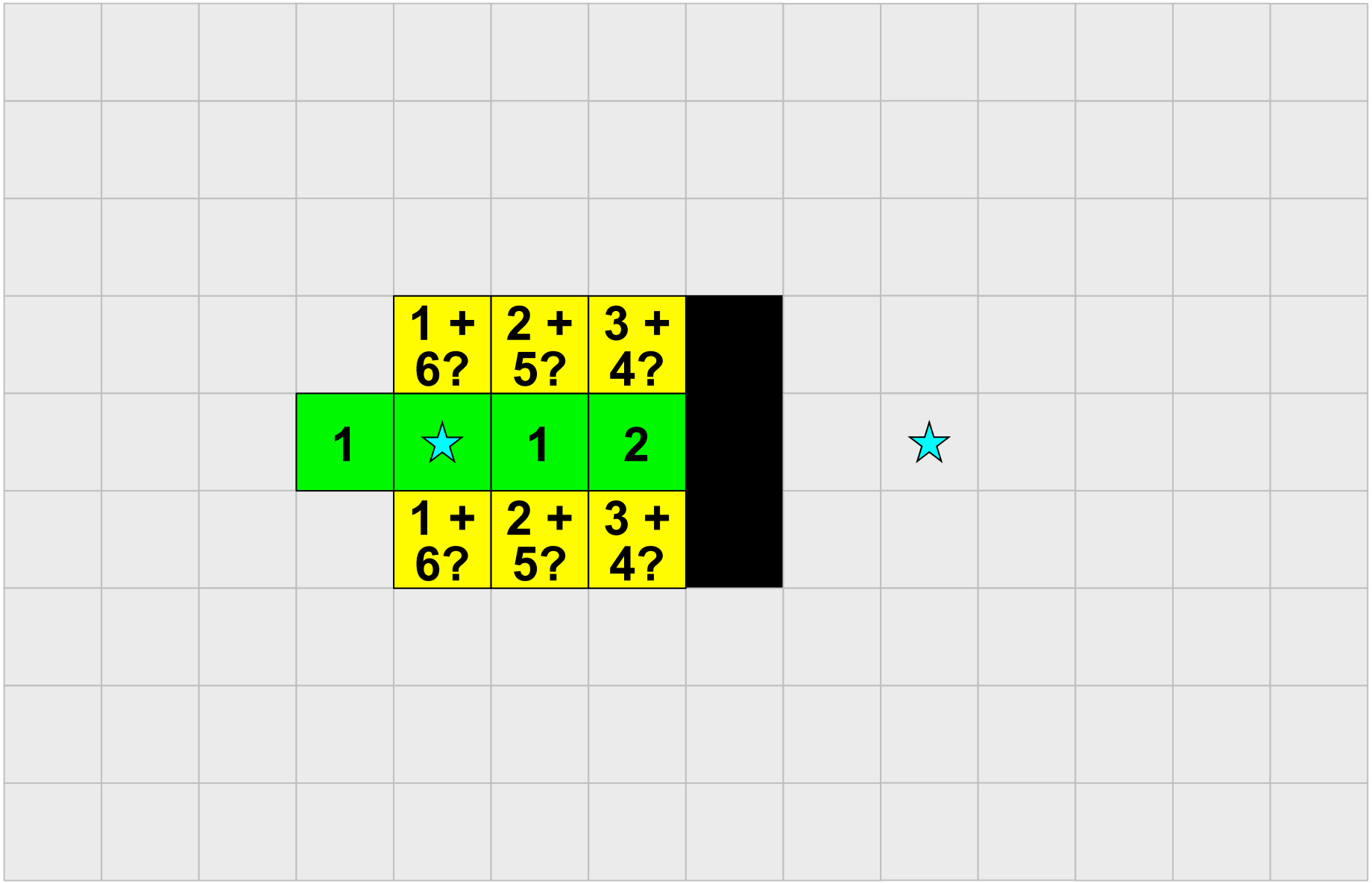

1 +
6?
1 +
6?
1 +
4?
1 +
6?

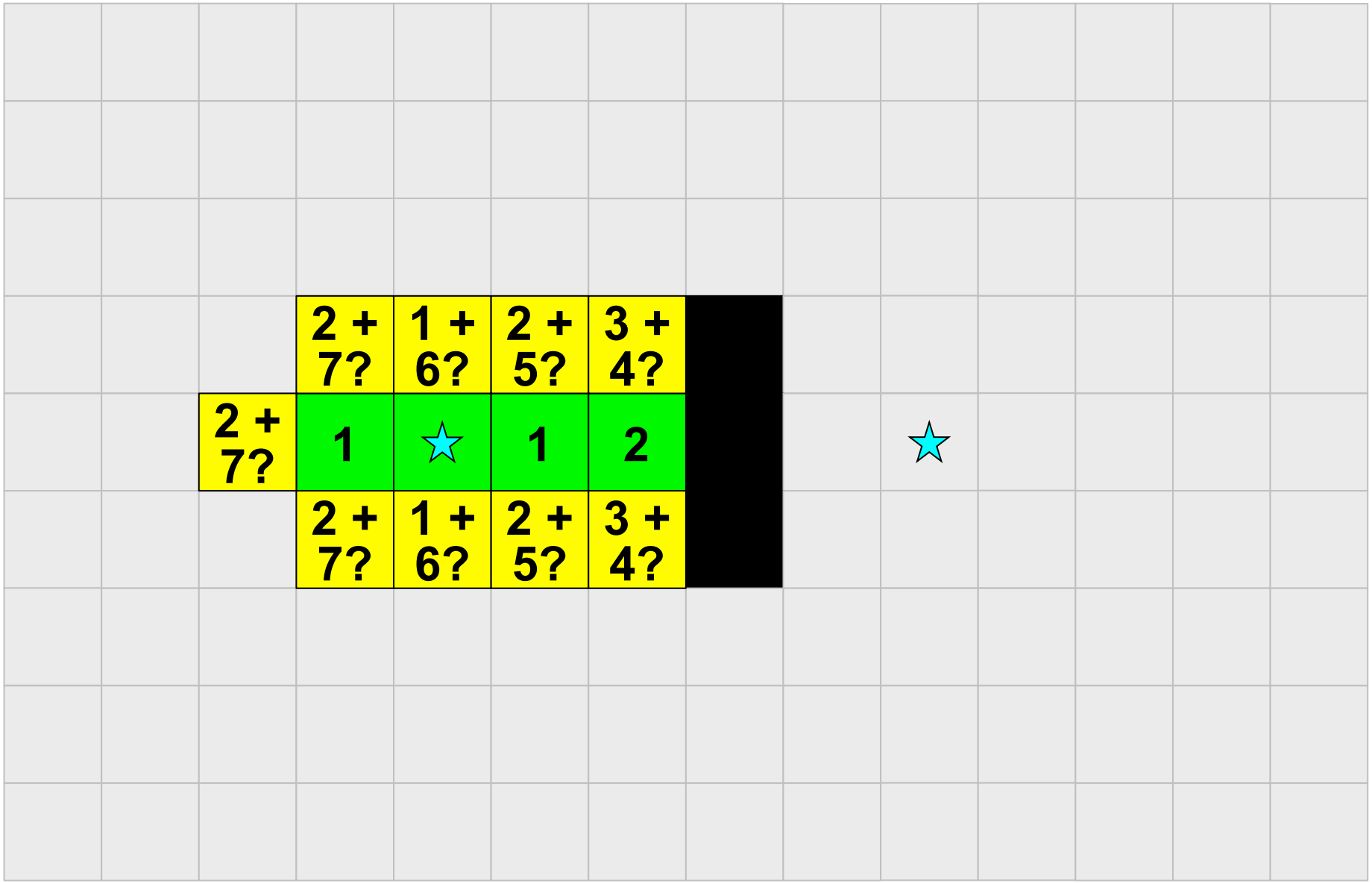

1 +
6?
1 +
6?
1
1 +
6?

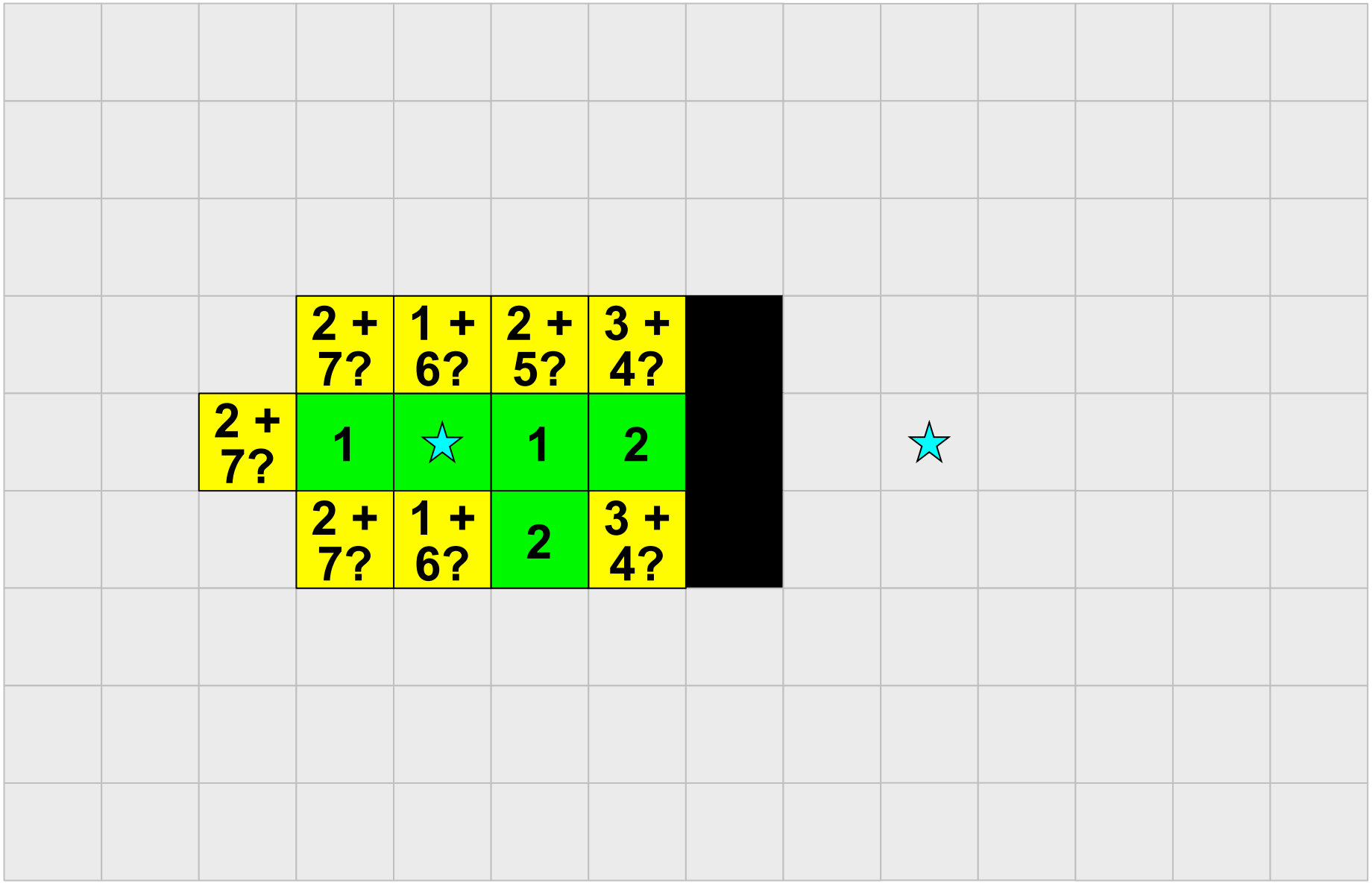

1 +
6?
2 +
5?
1 +
6?
1
2 +
3?
1 +
6?
2 +
5?

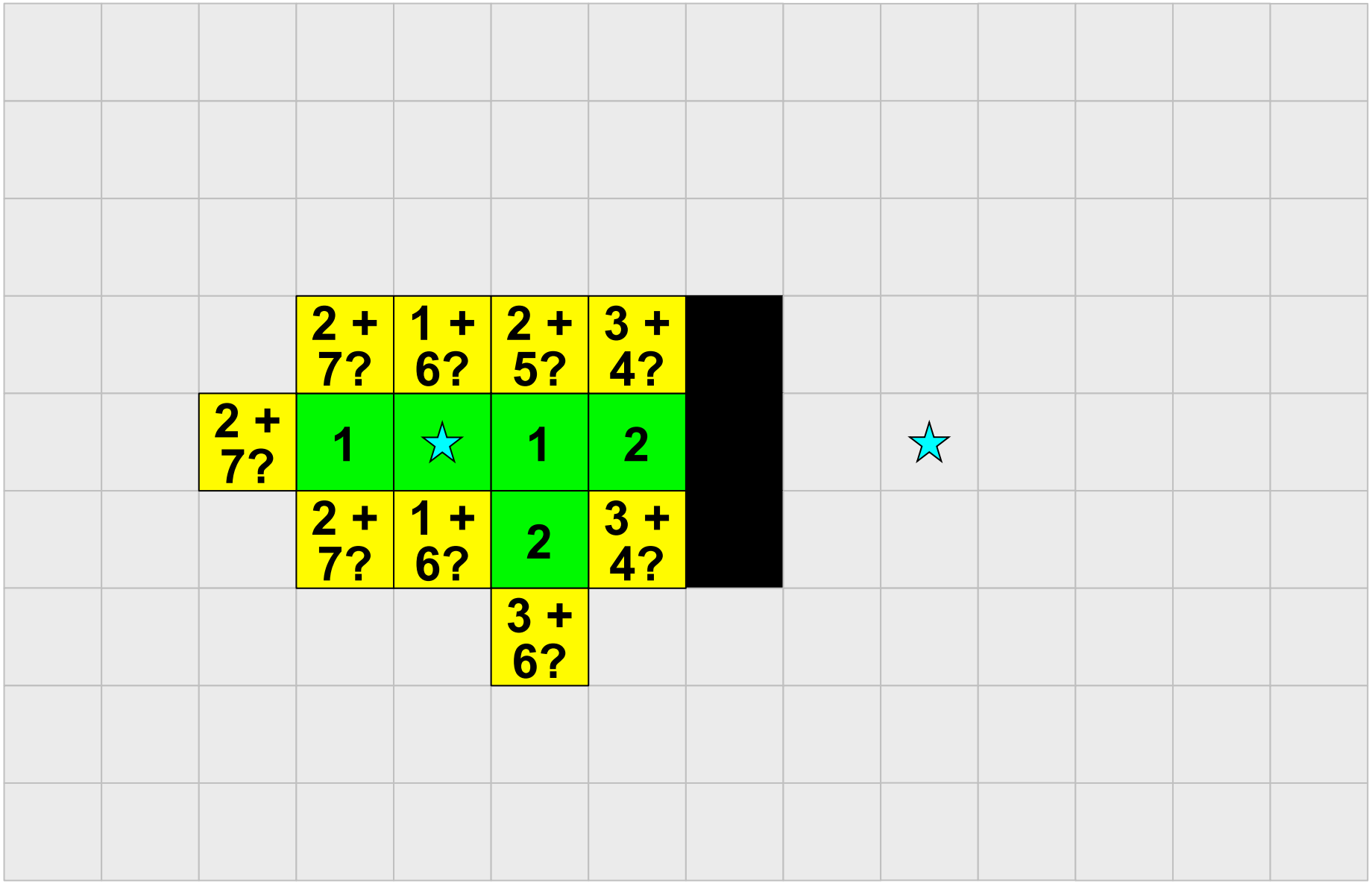

1 + 6?
2 + 5?
1 + 6?
1
2
1 + 6?
2 + 5?

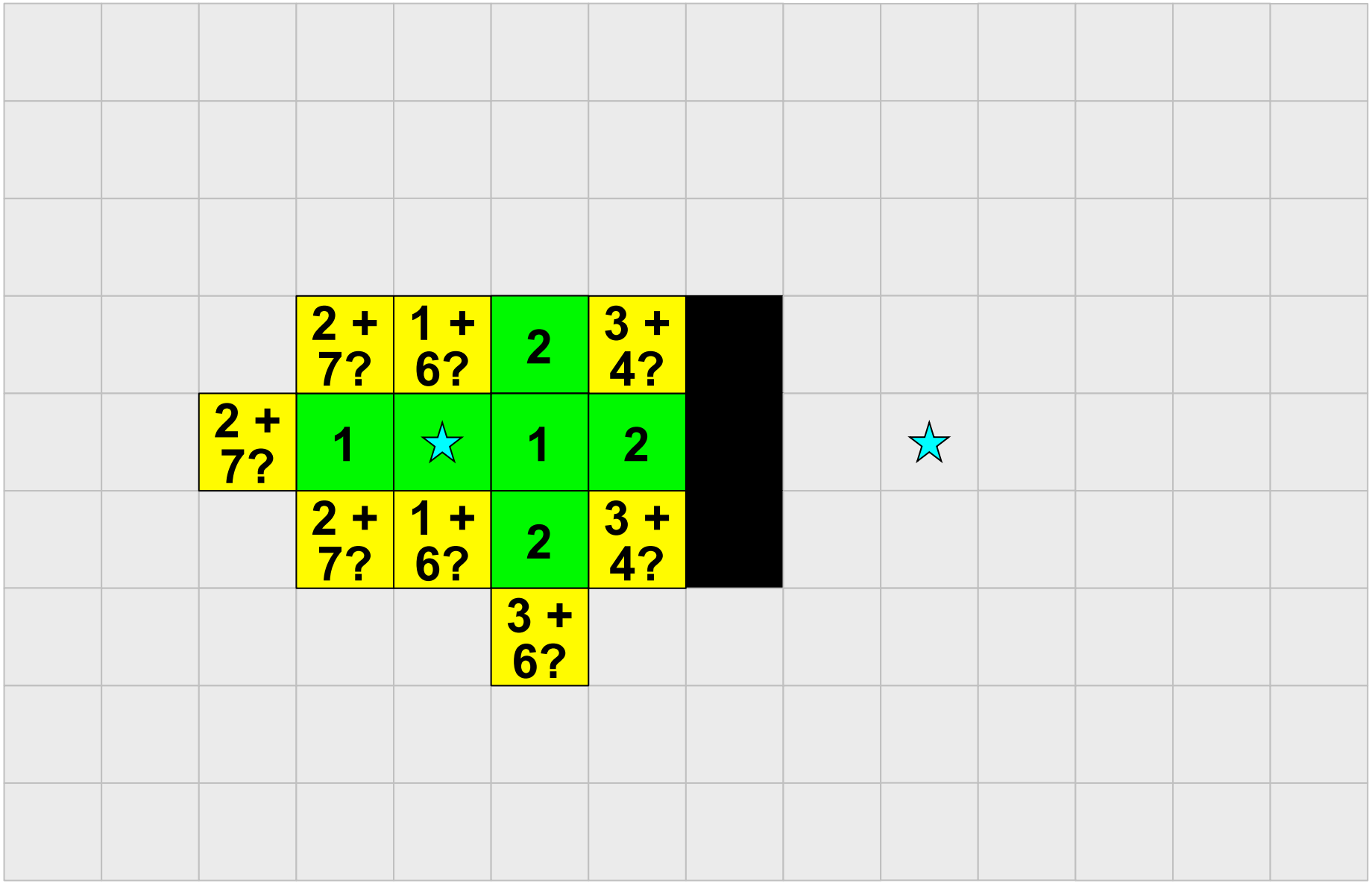

1 + 6?
2 + 5?
3 + 4?
1 + 6?
1
2
1 + 6?
2 + 5?
3 + 4?

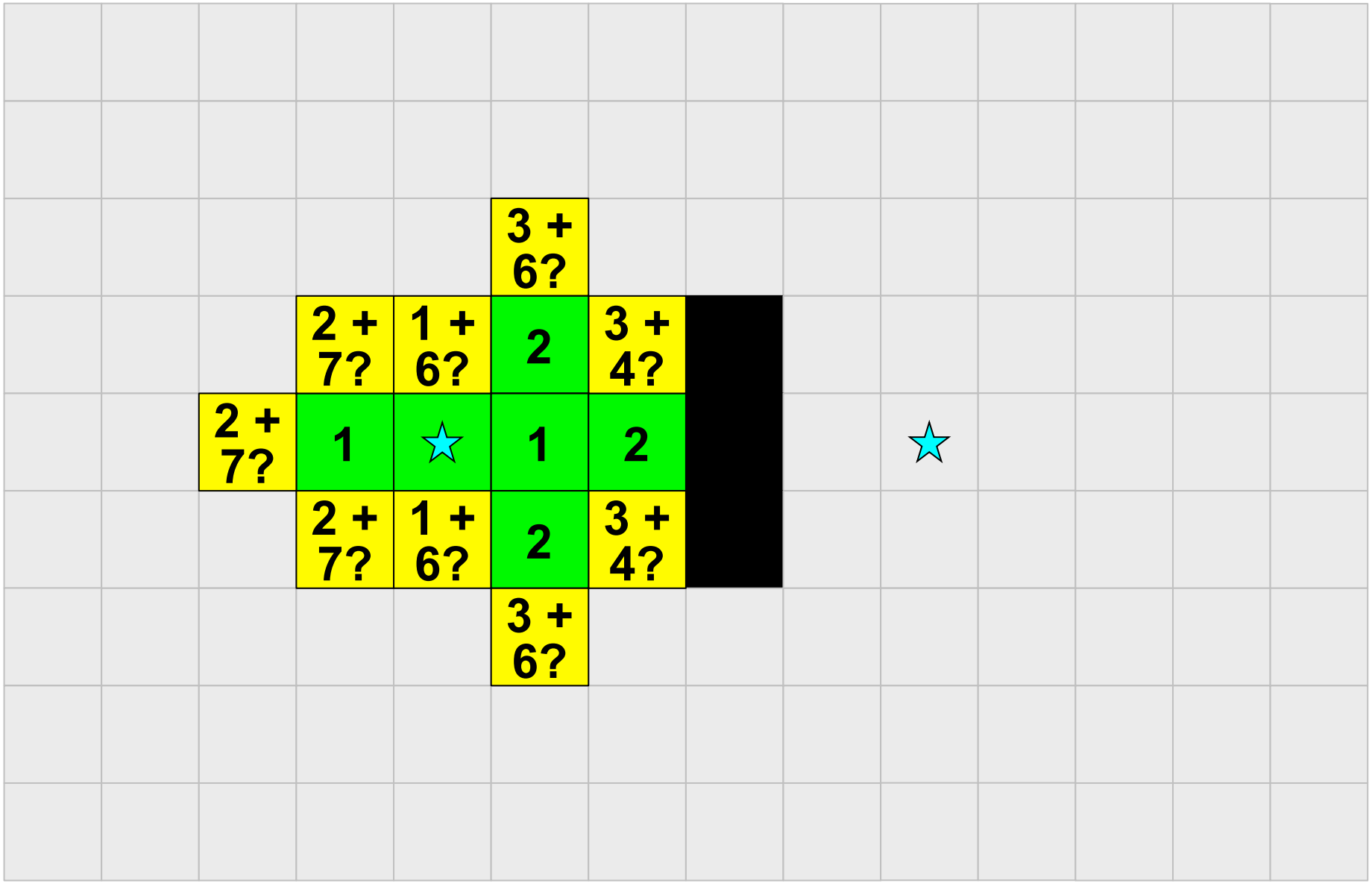

1 + 6?
2 + 5?
3 + 4?
1
1
2
1 + 6?
2 + 5?
3 + 4?

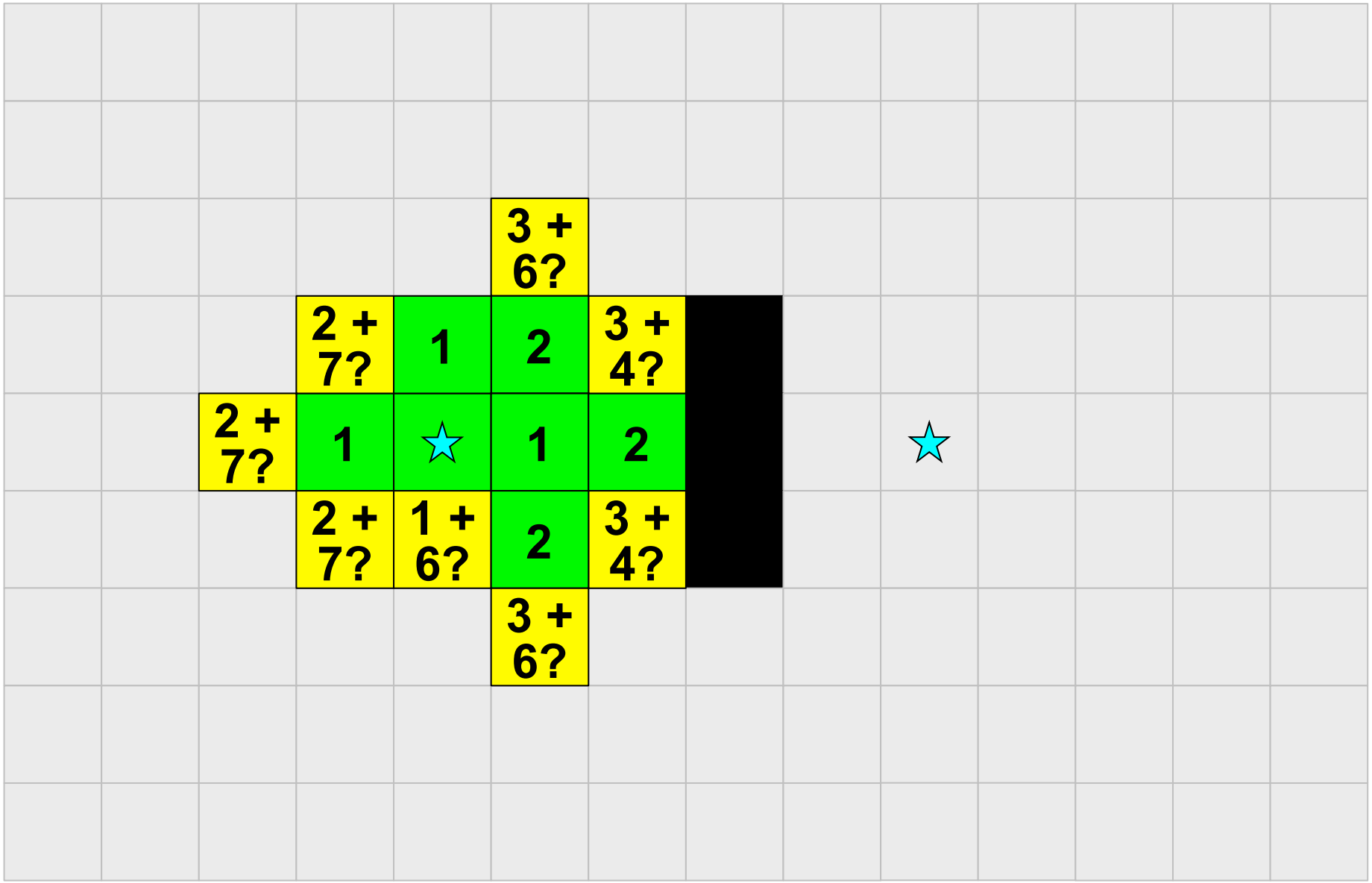

2 + 7?
1 + 6?
2 + 5?
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2 + 5?
3 + 4?

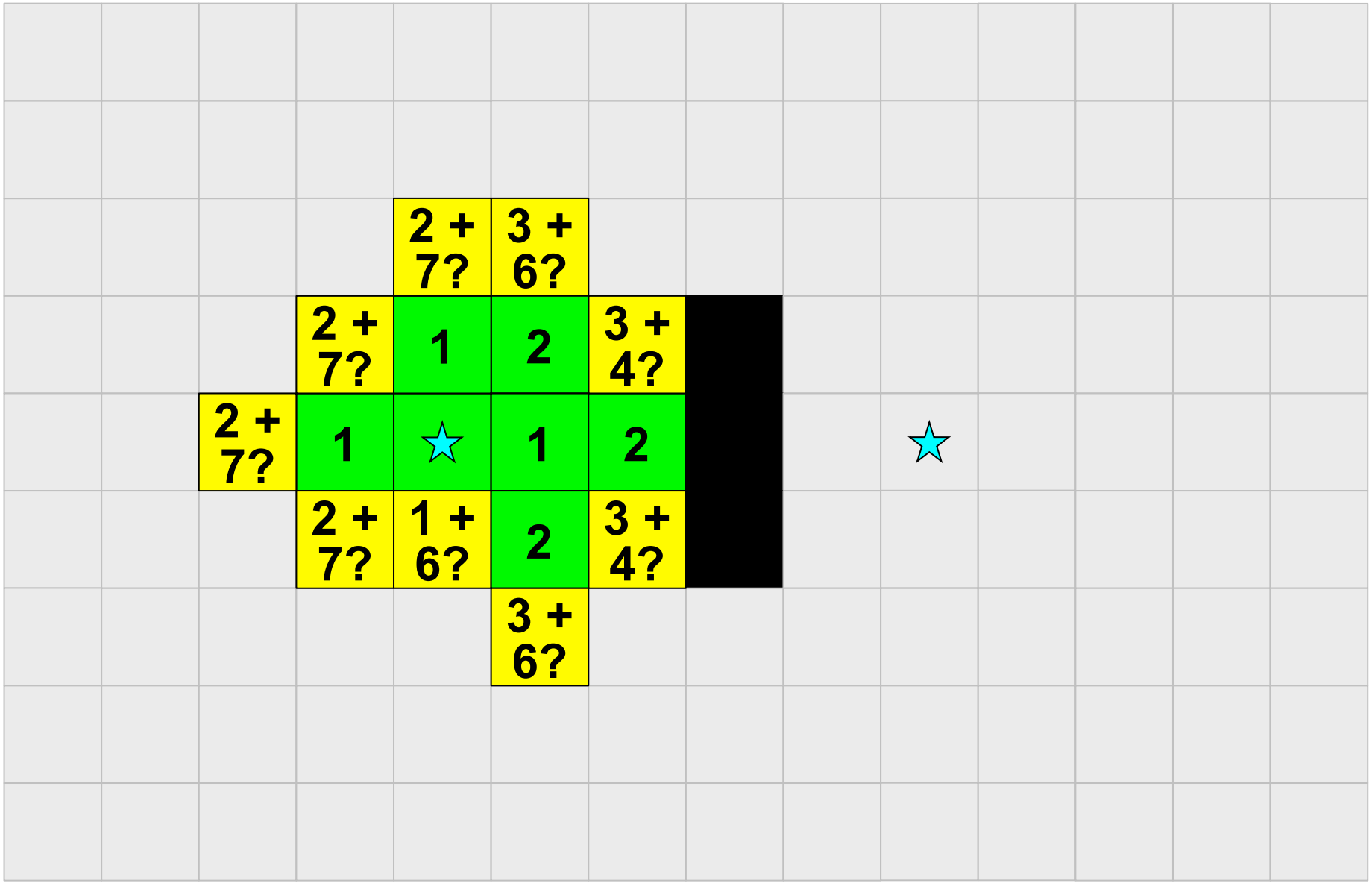

2 + 7?
1 + 6?
2 + 5?
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?

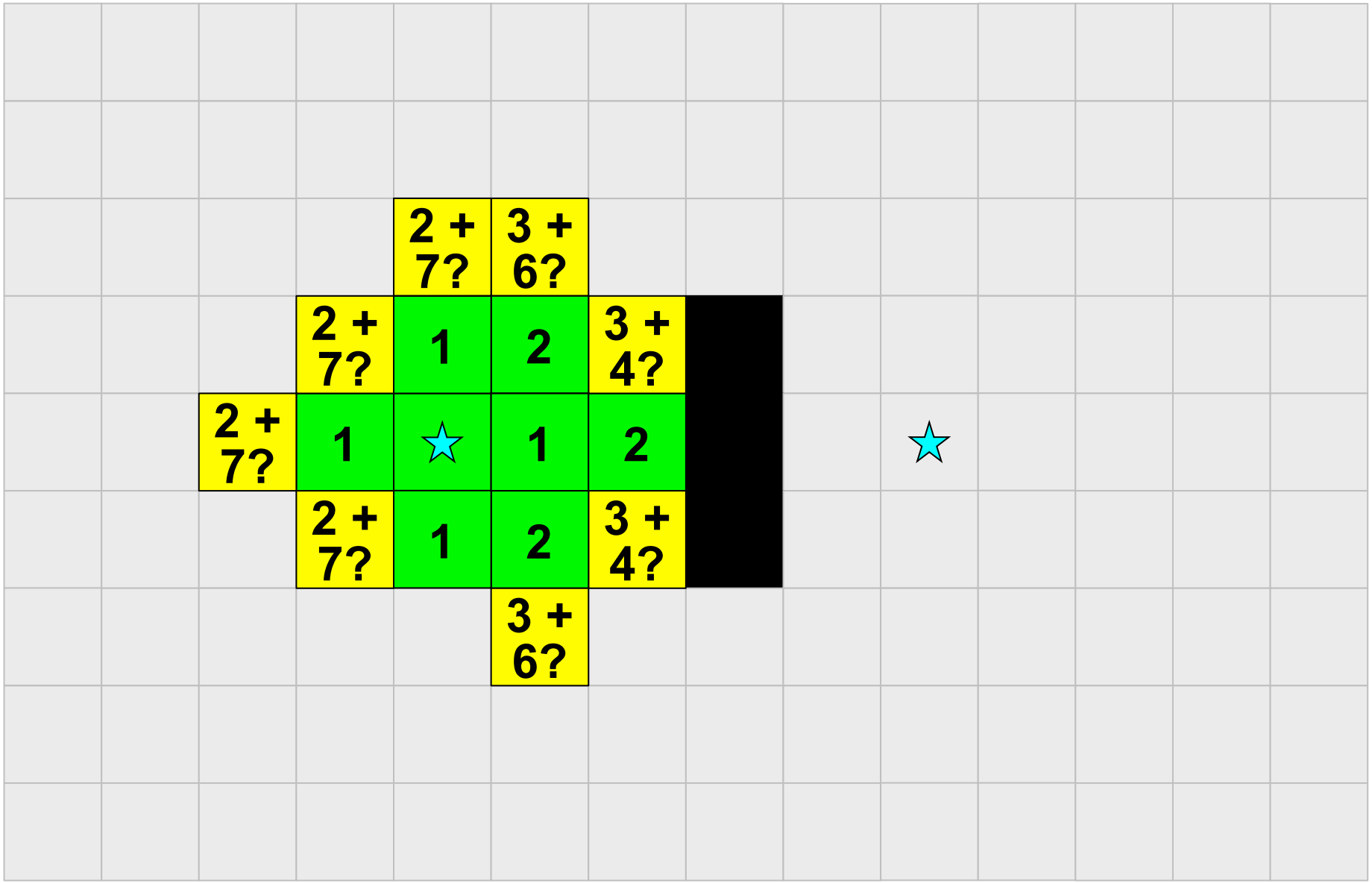

2 + 7?
1 + 6?
2 + 5?
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?
3 + 6?

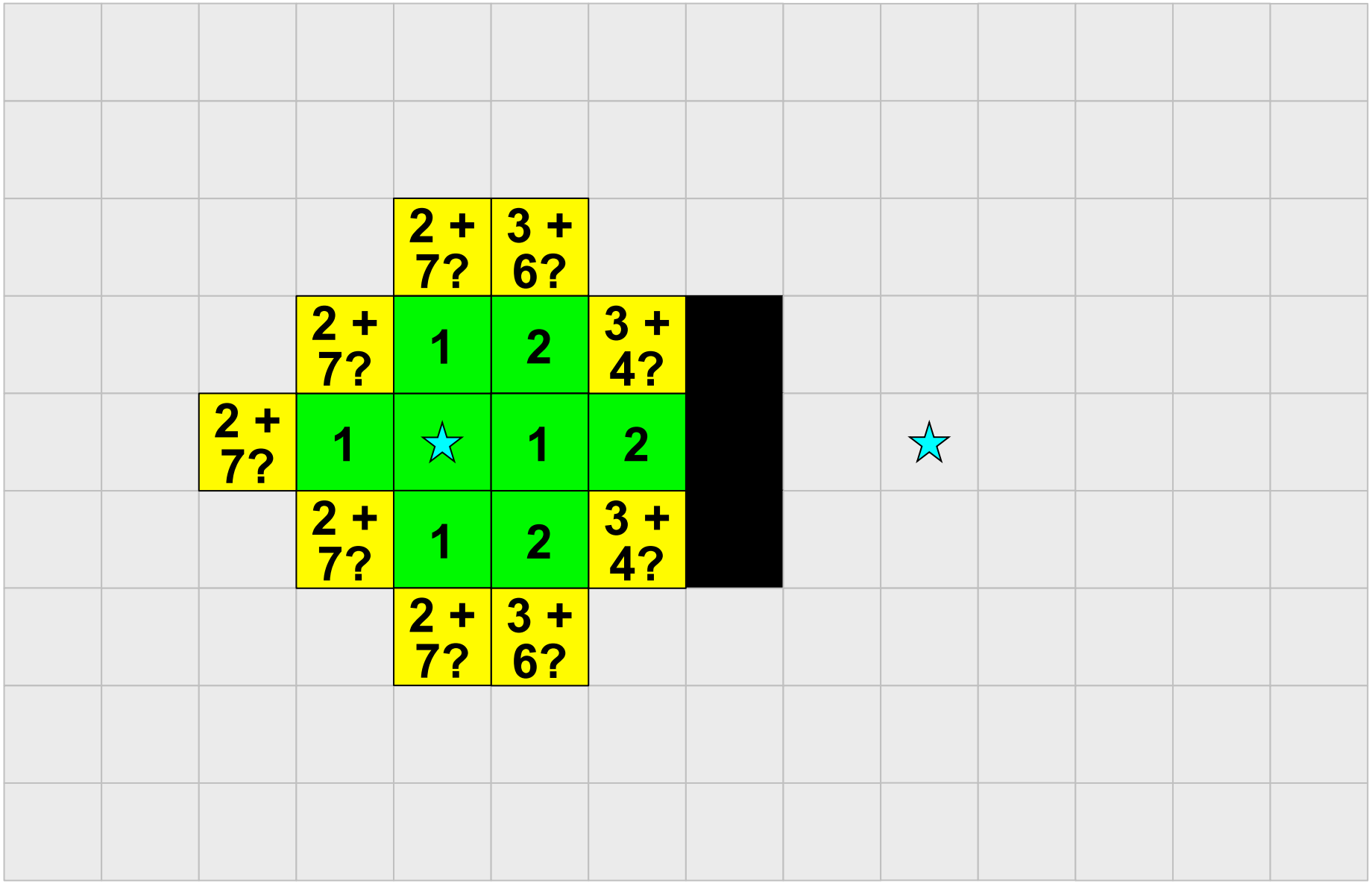

2 + 7?
1 + 6?
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?
3 + 6?

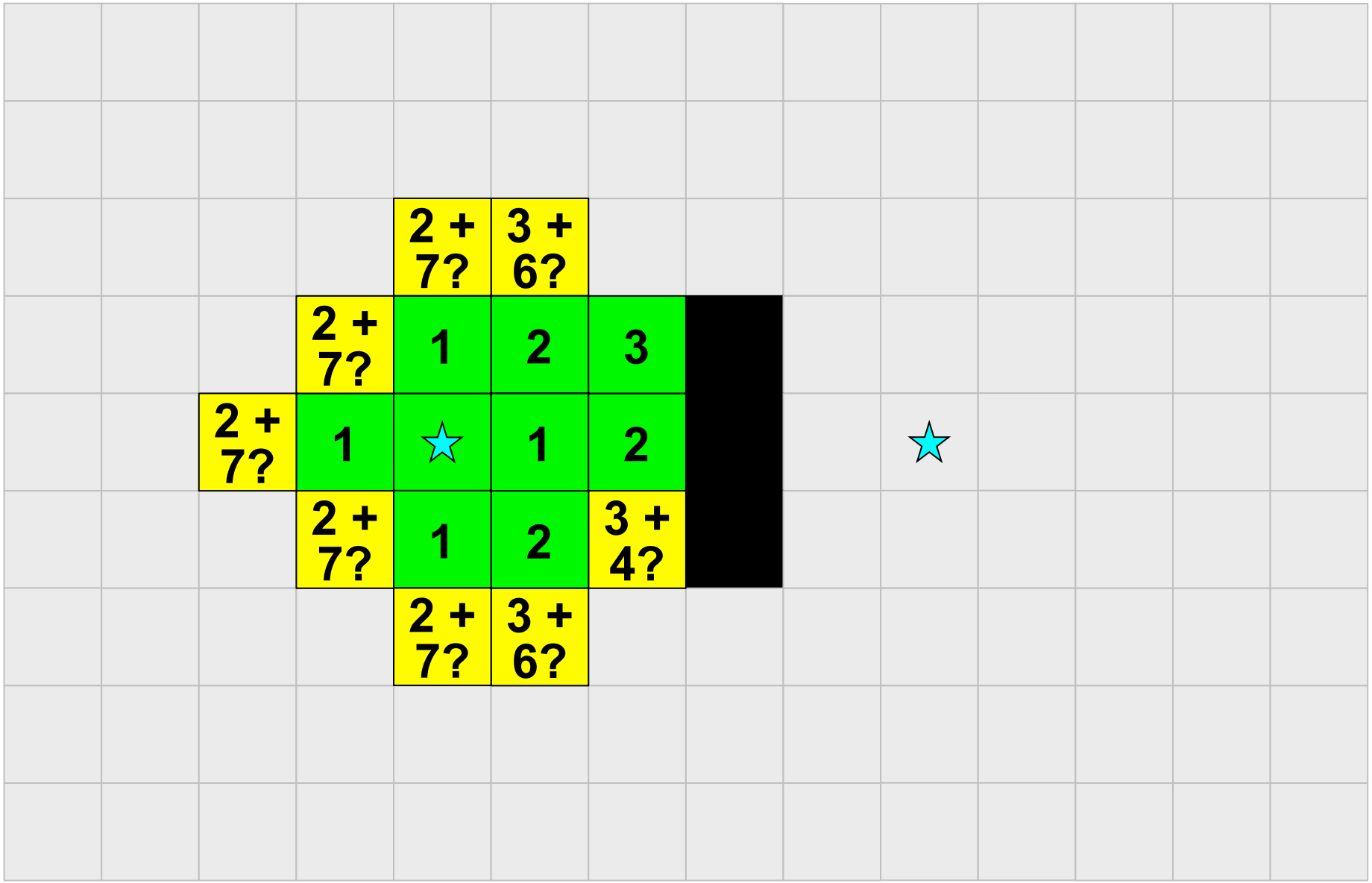

3 + 6?
2 + 7?
1 + 6?
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?
3 + 6?

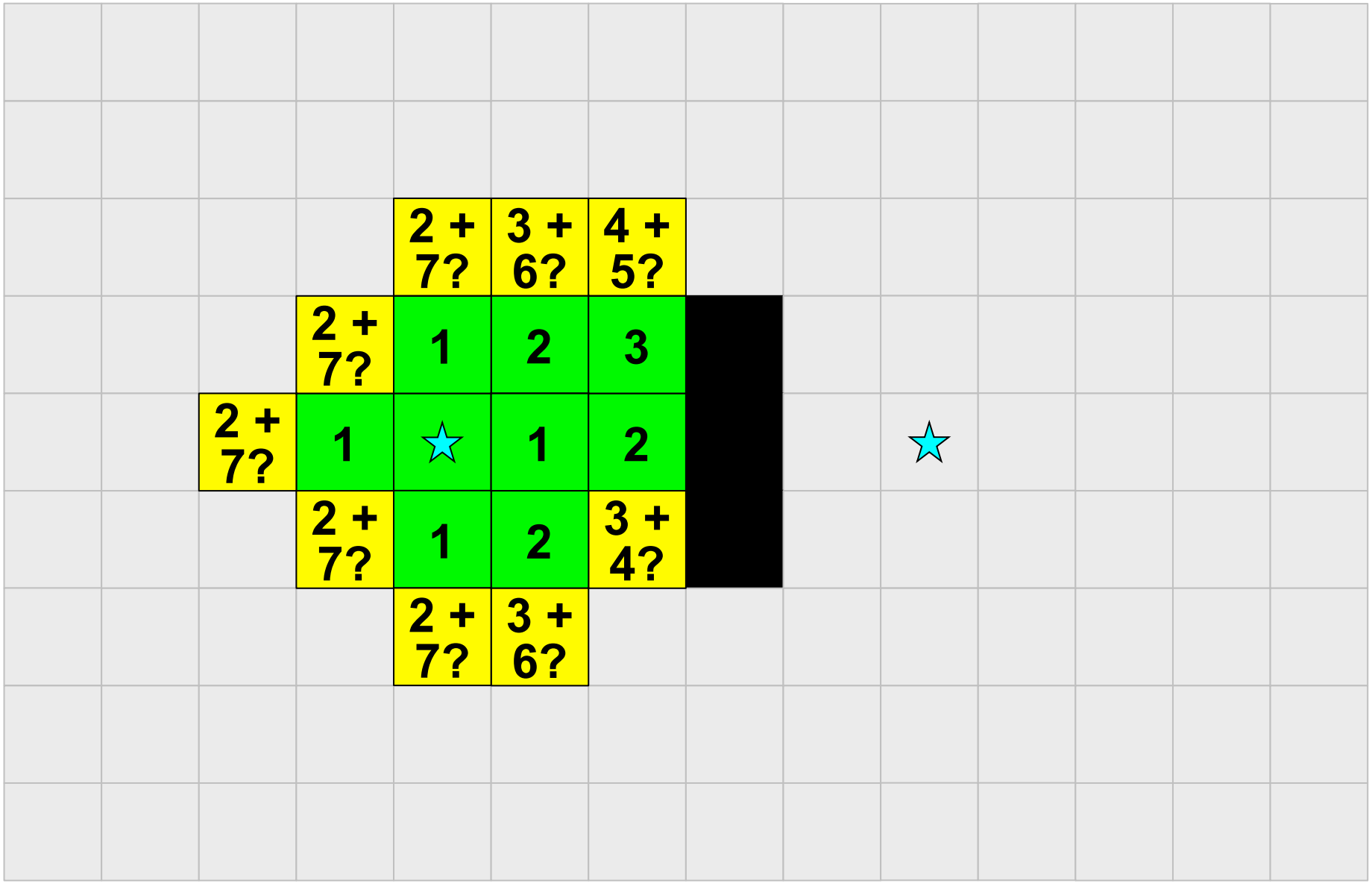

3 + 6?
2 + 7?
1
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?
3 + 6?

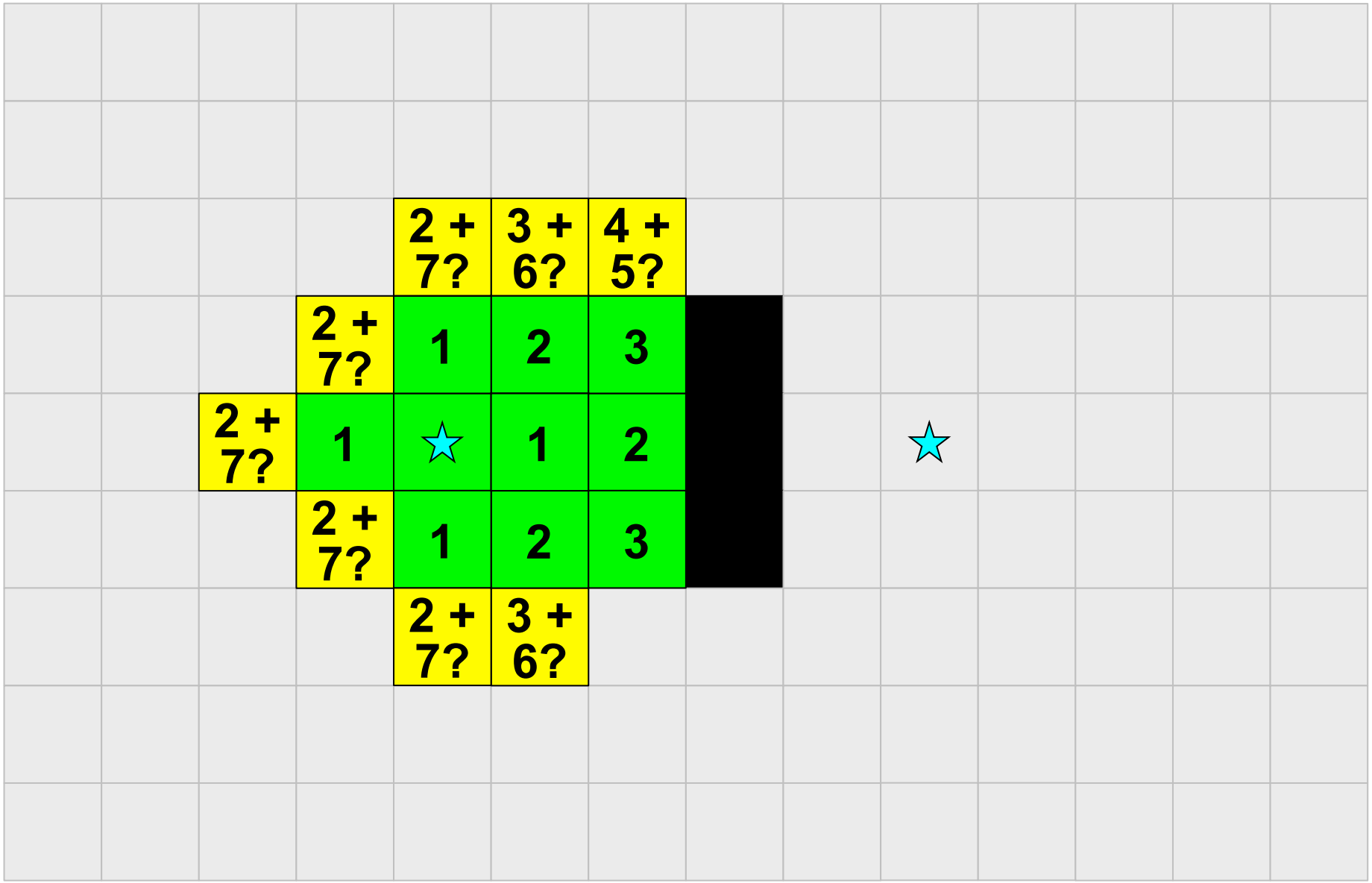

2 + 7?
3 + 6?
2 + 7?
1
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?
3 + 6?

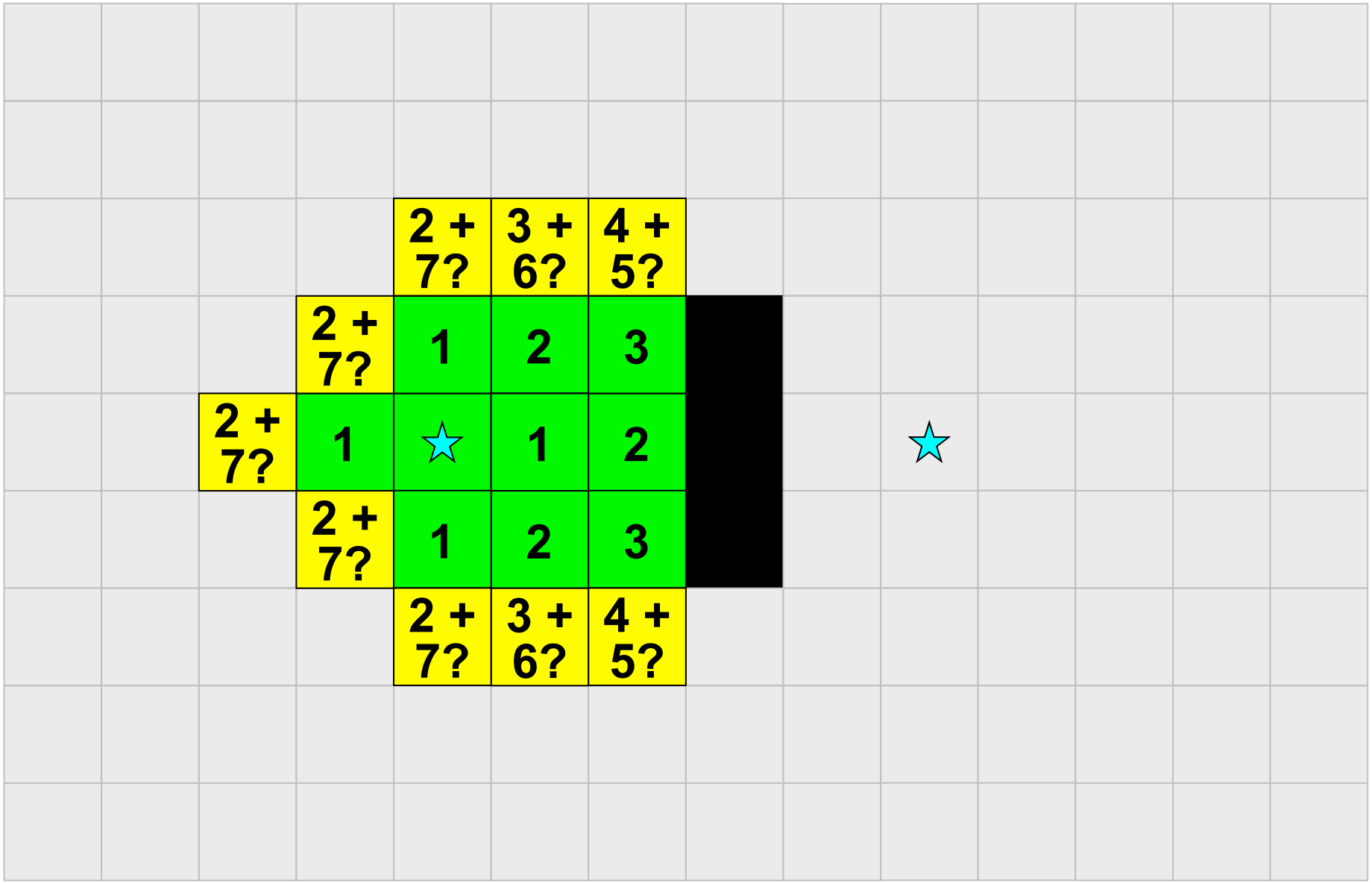

2 + 7?
3 + 6?
2 + 7?
1
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1
2
3 + 4?
3 + 6?

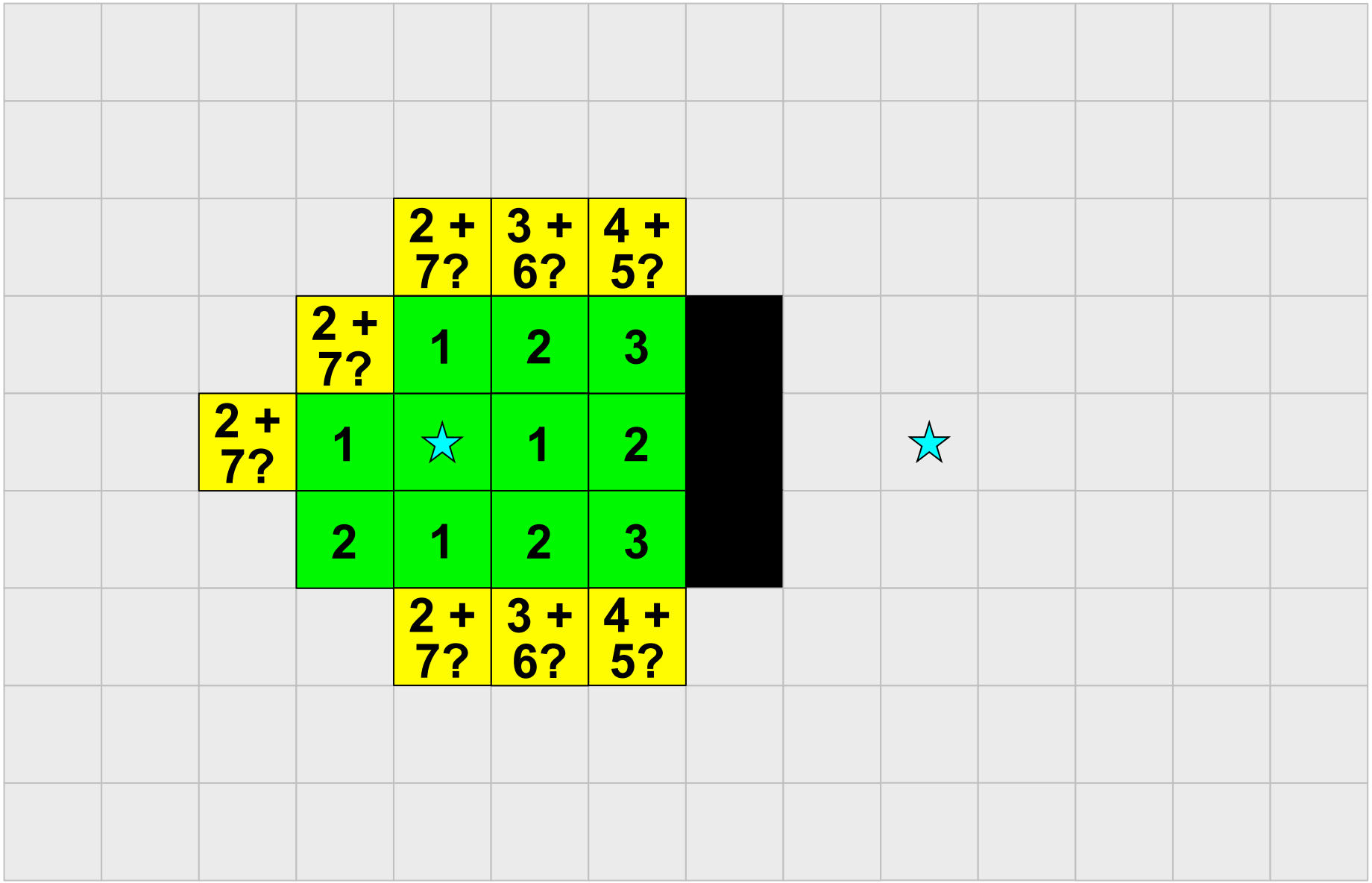

2 + 7?
3 + 6?
2 + 7?
1
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1
2
3 + 4?
2 + 7?
3 + 6?

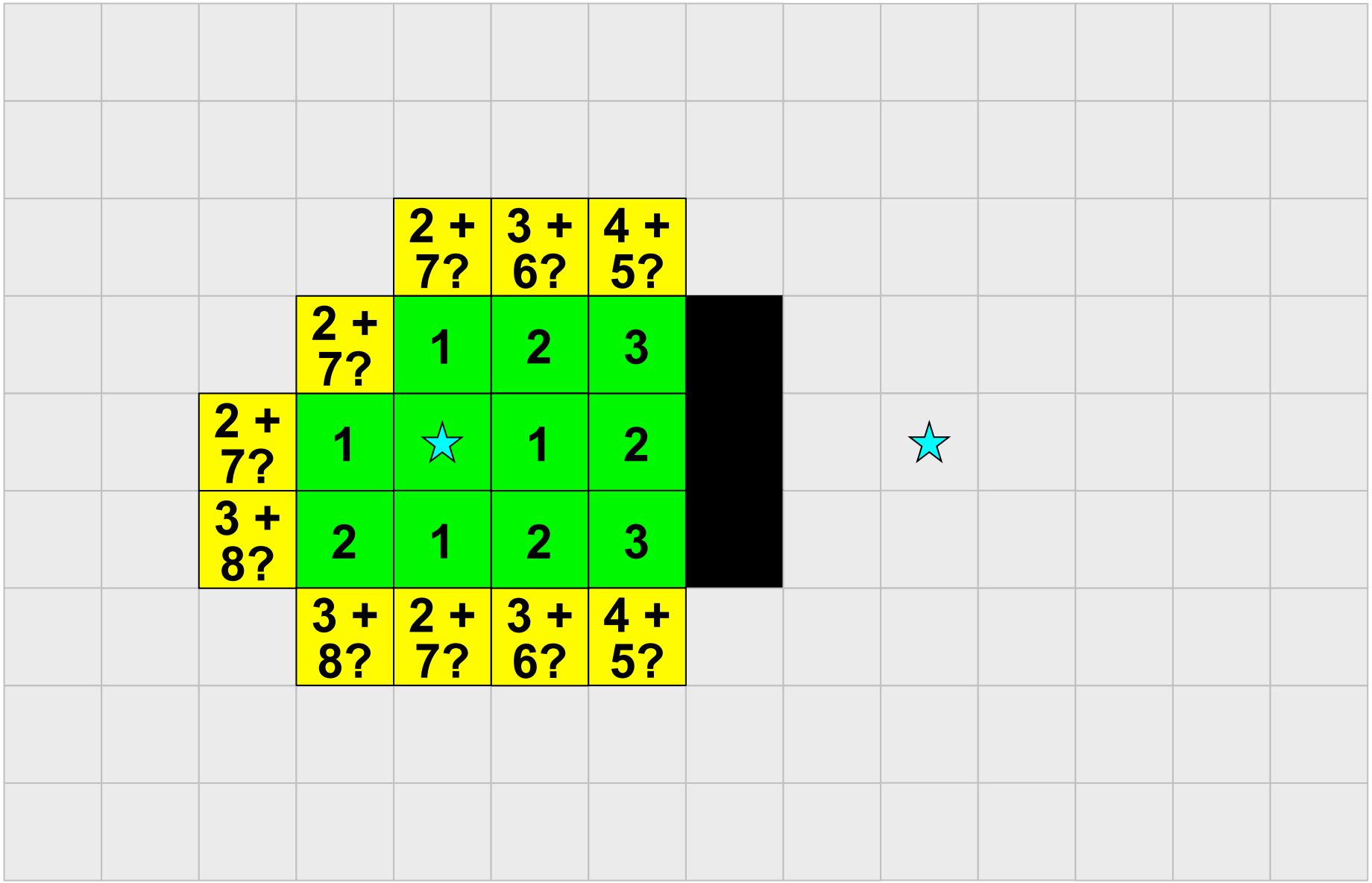

2 + 7?
3 + 6?
2 + 7?
1
2
3
2 + 7?
1
1
2
2 + 7?
1
2
3 + 4?
2 + 7?
3 + 6?

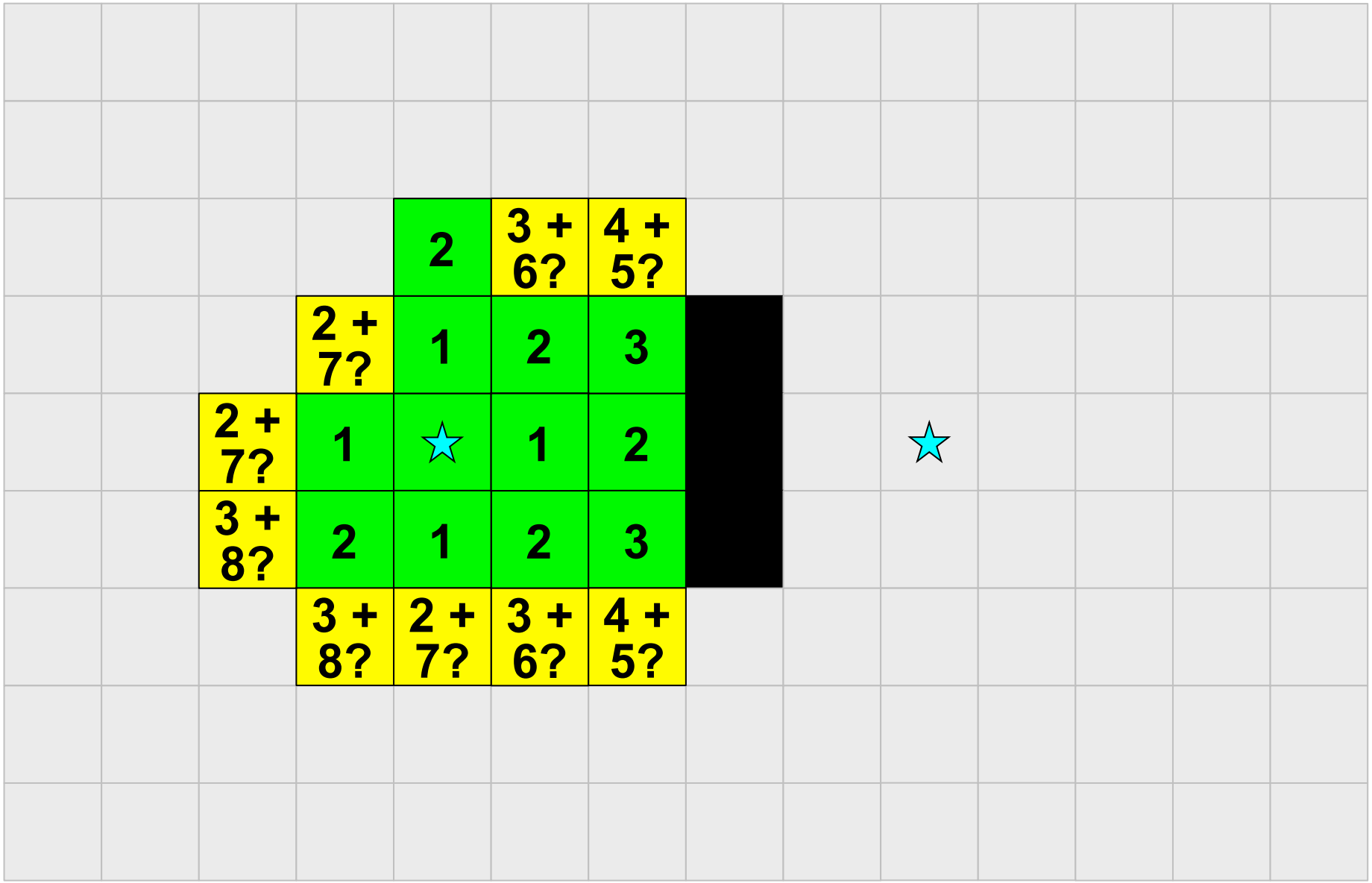

2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
2 + 7?
1
2
3 + 4?
2 + 7?
3 + 6?

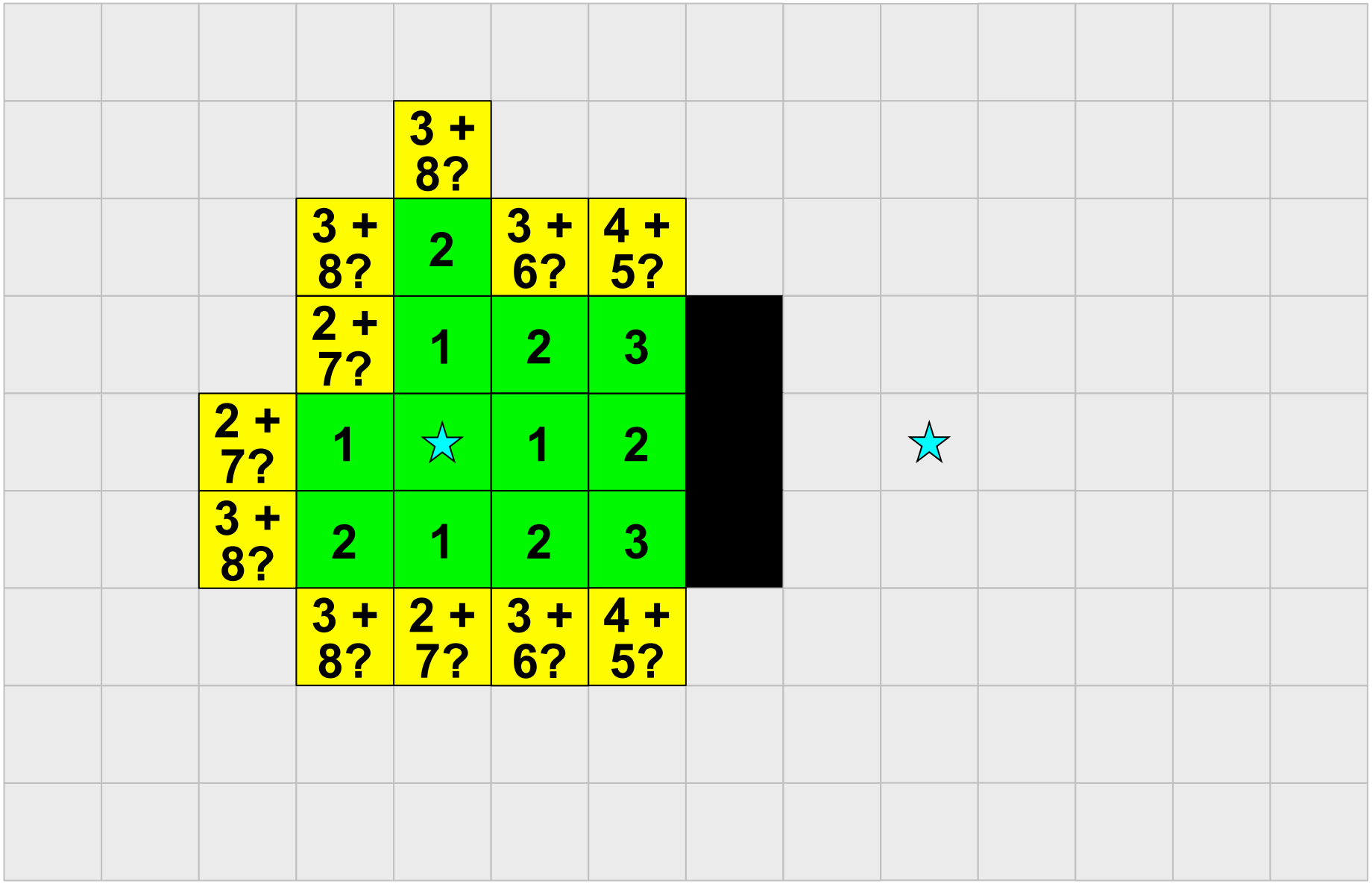

2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
2 + 7?
1
2
3
2 + 7?
3 + 6?

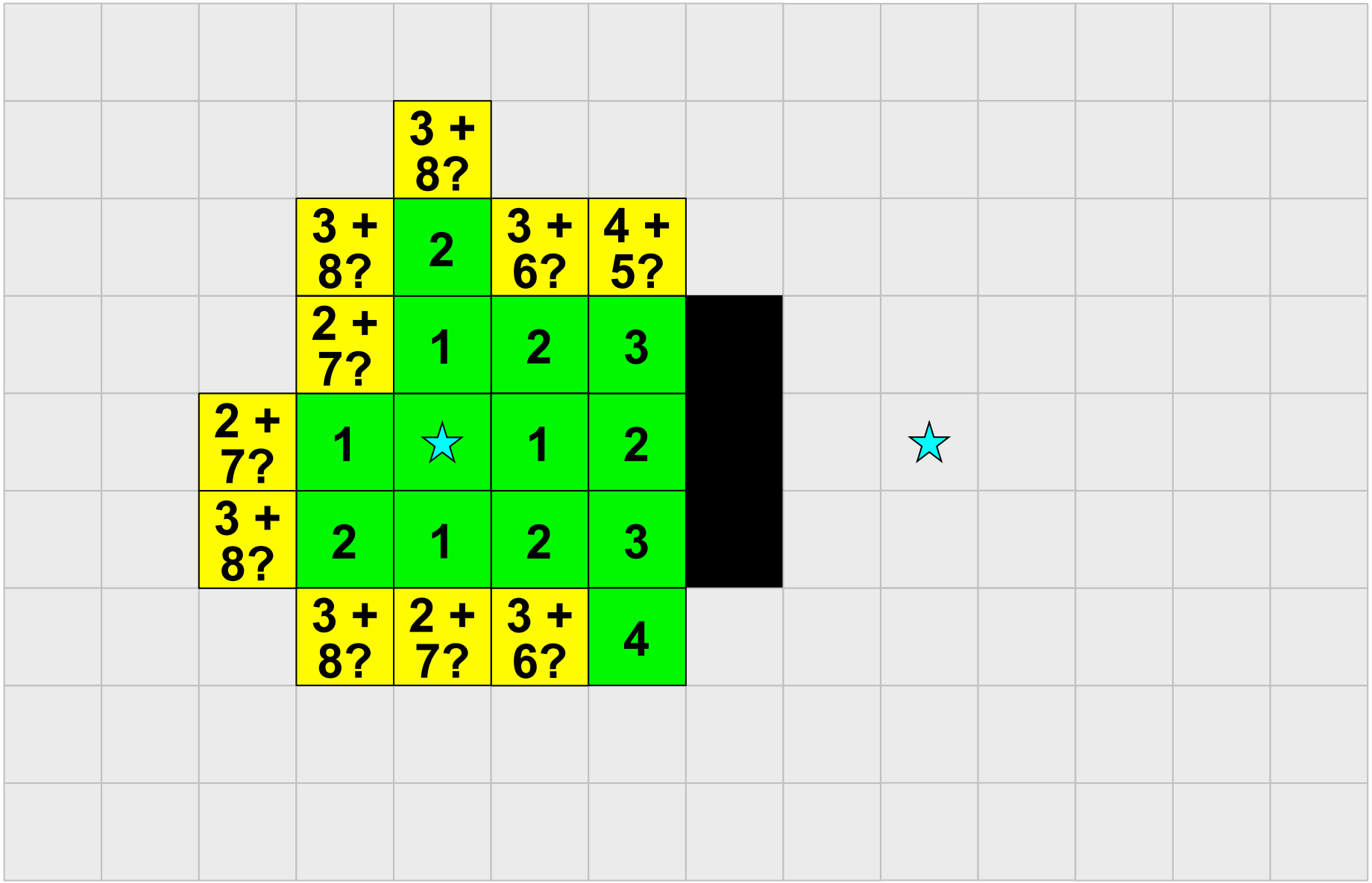

2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
2 + 7?
1
2
3
2 + 7?
3 + 6?
4 + 5?

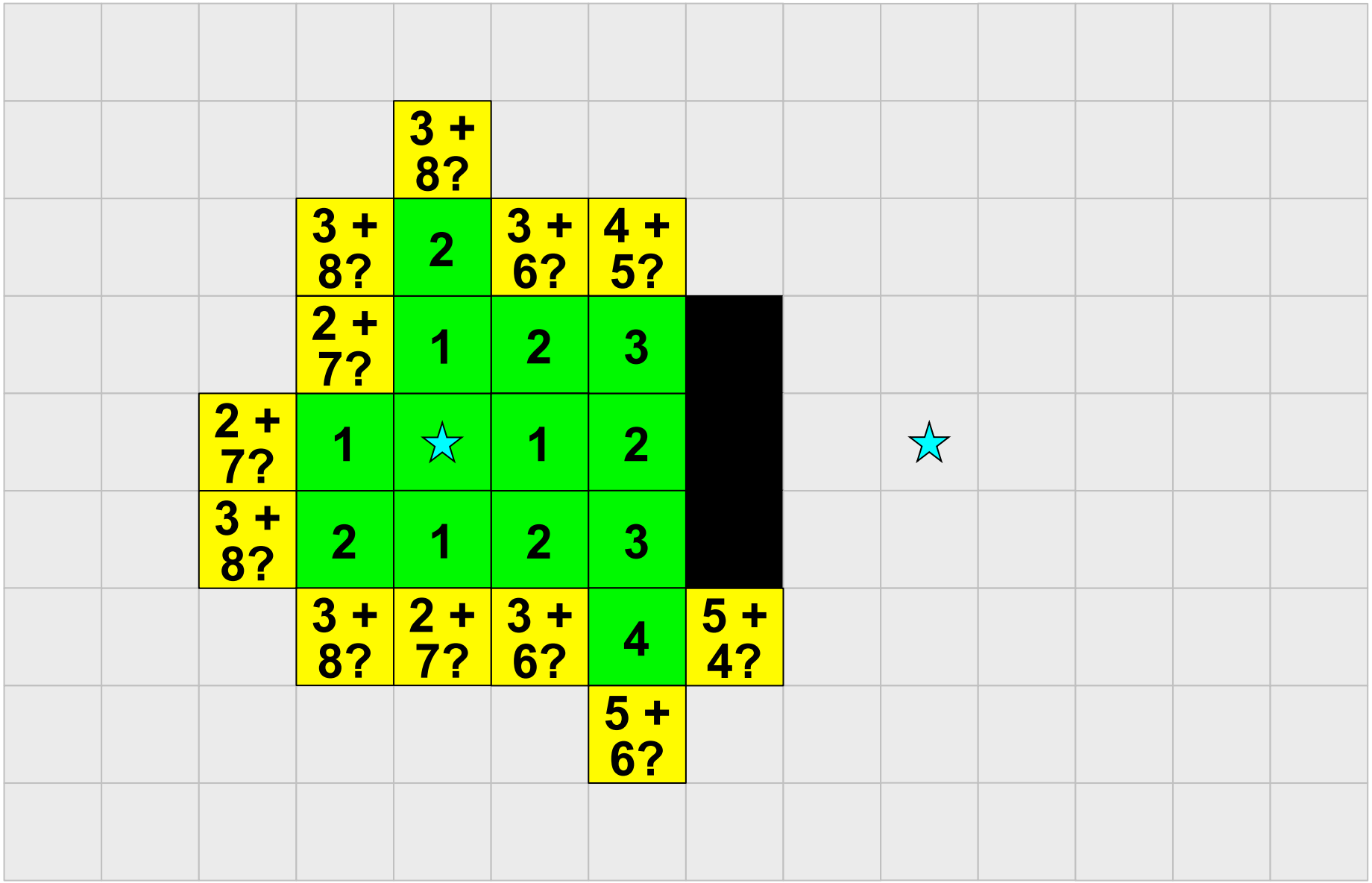

2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
2
1
2
3
2 + 7?
3 + 6?
4 + 5?

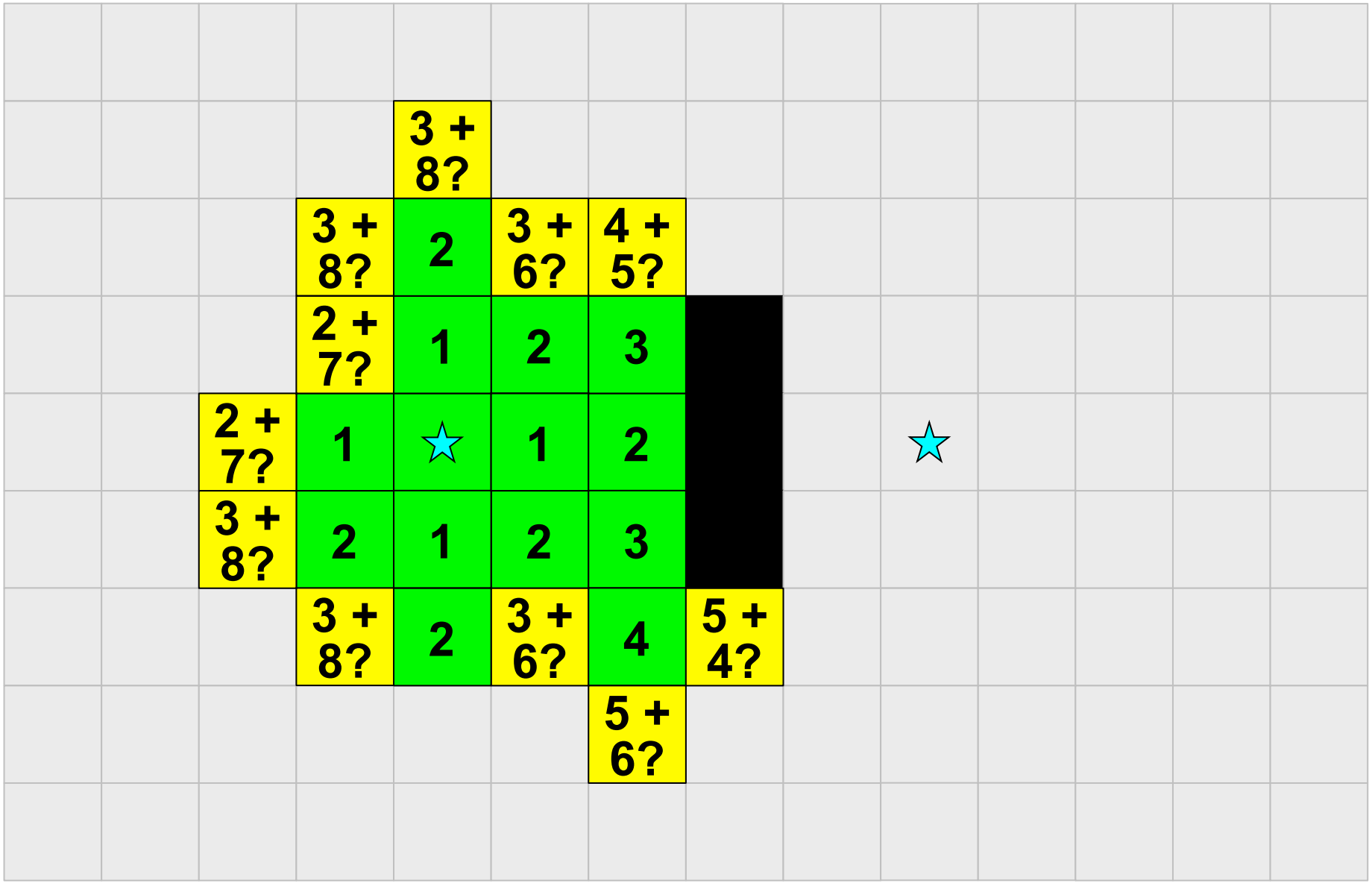

2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2 + 7?
3 + 6?
4 + 5?

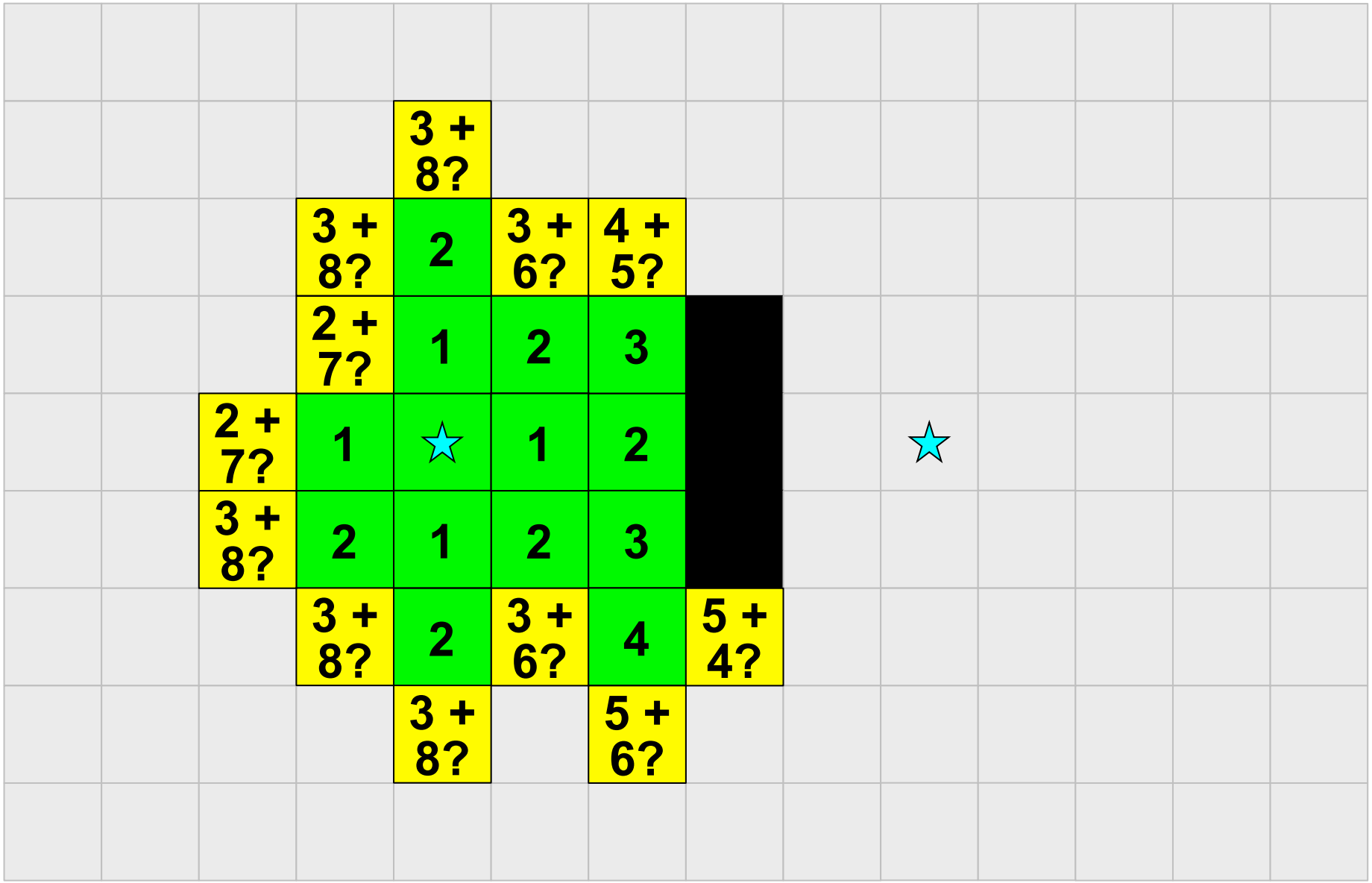

2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2 + 7?
3 + 6?
4 + 5?

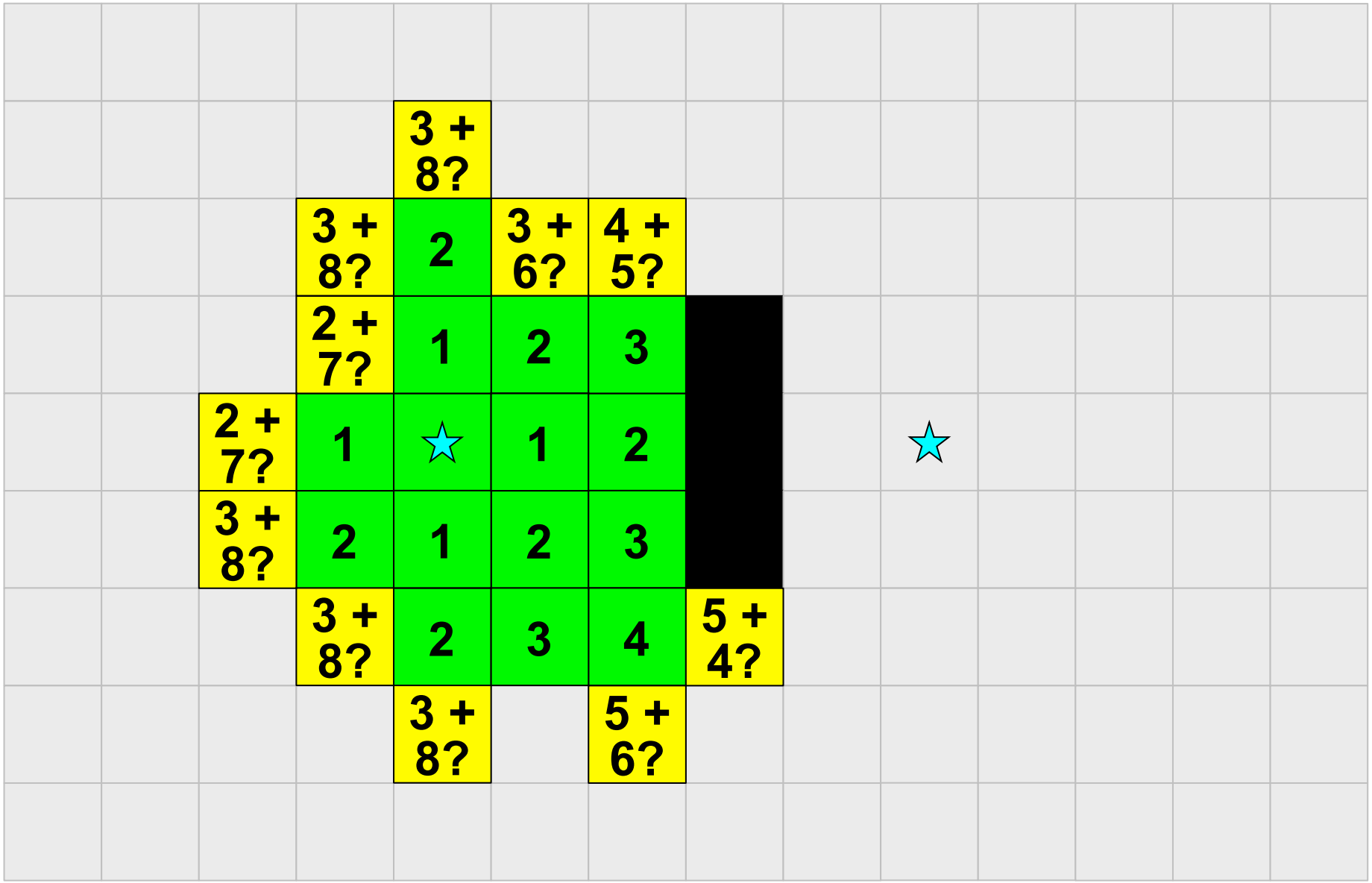

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2 + 7?
3 + 6?
4 + 5?

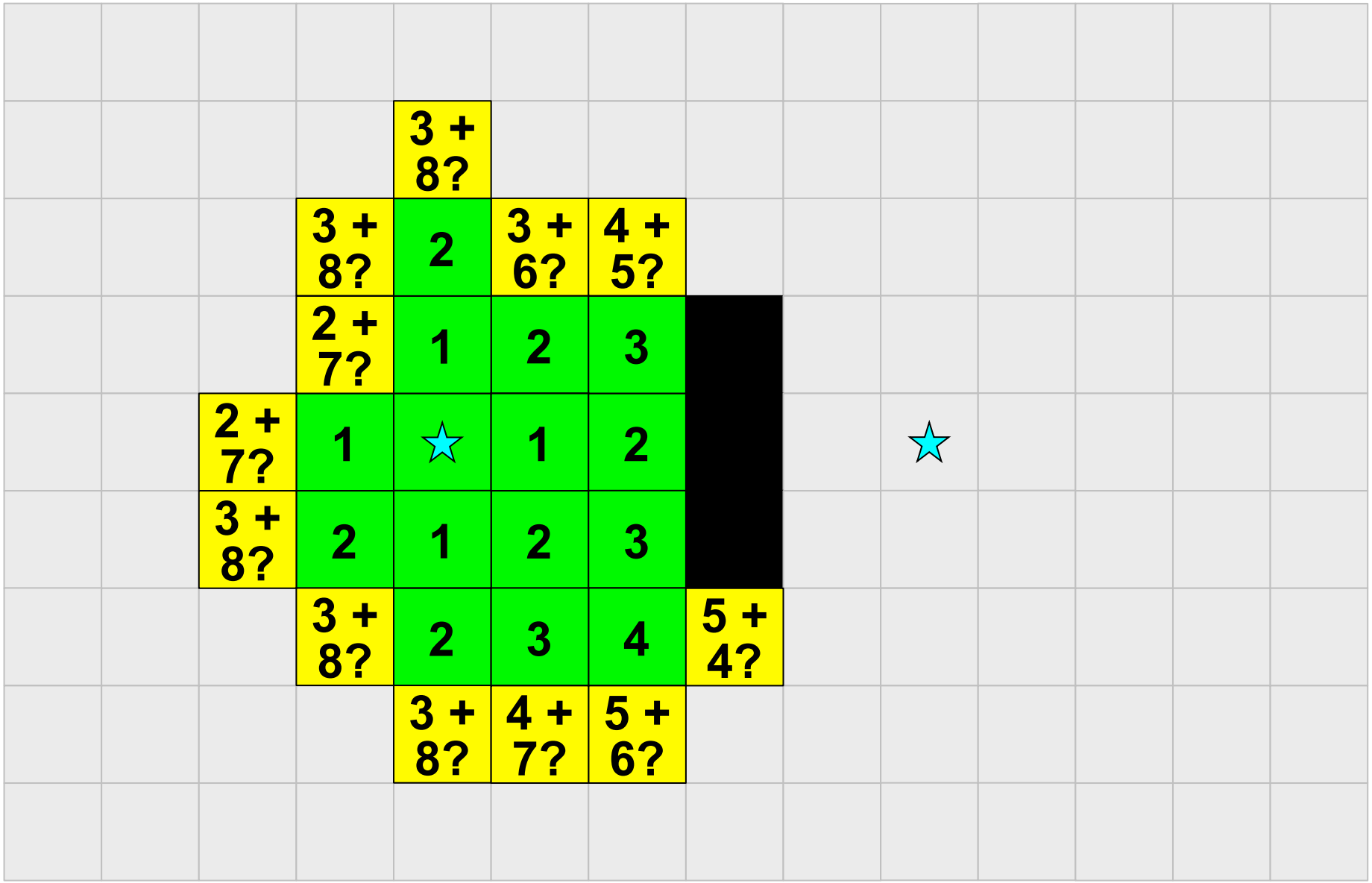

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2 + 7?
3 + 6?
4

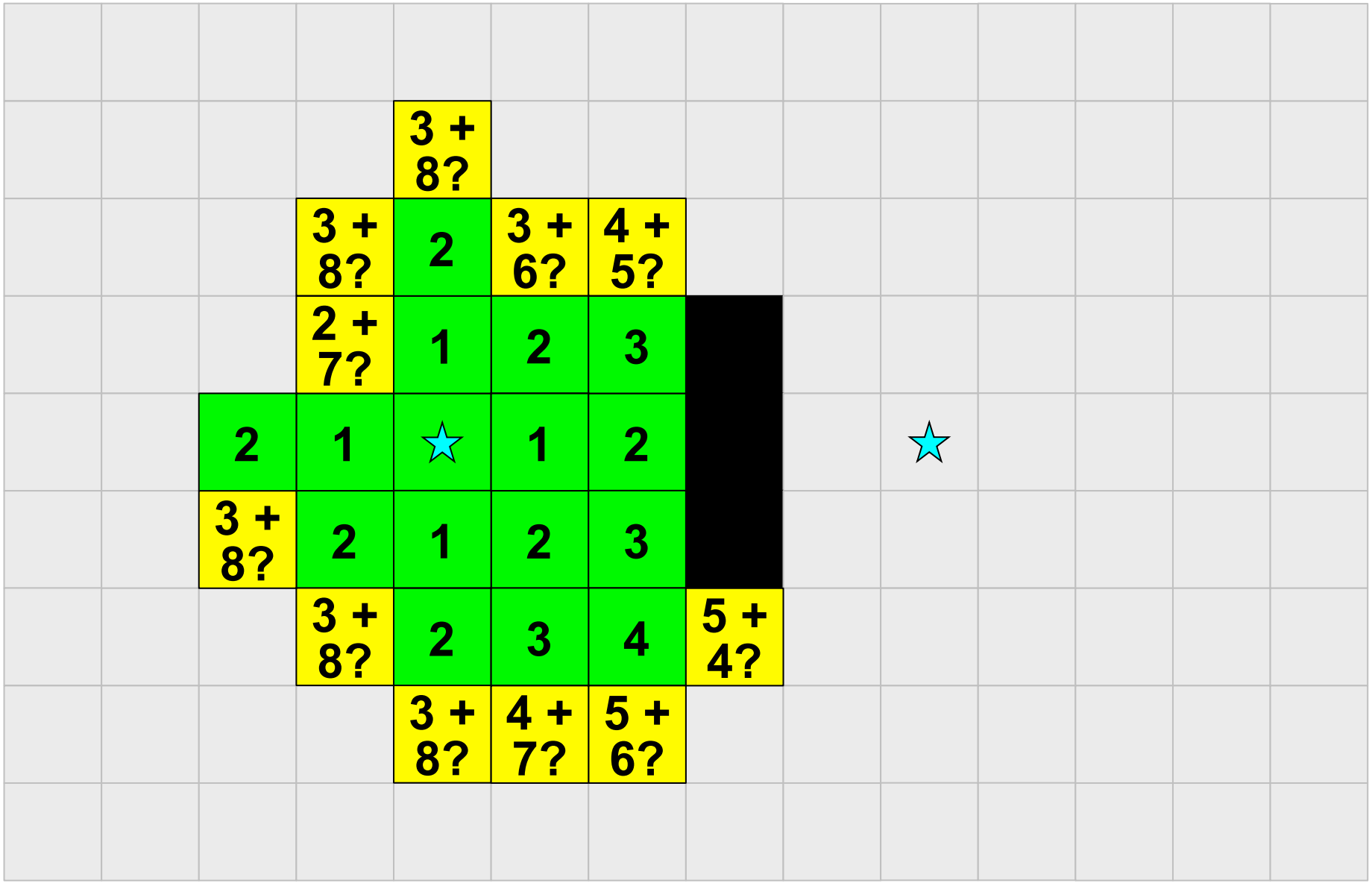

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2 + 7?
3 + 6?
4
5 + 4?
5 + 6?

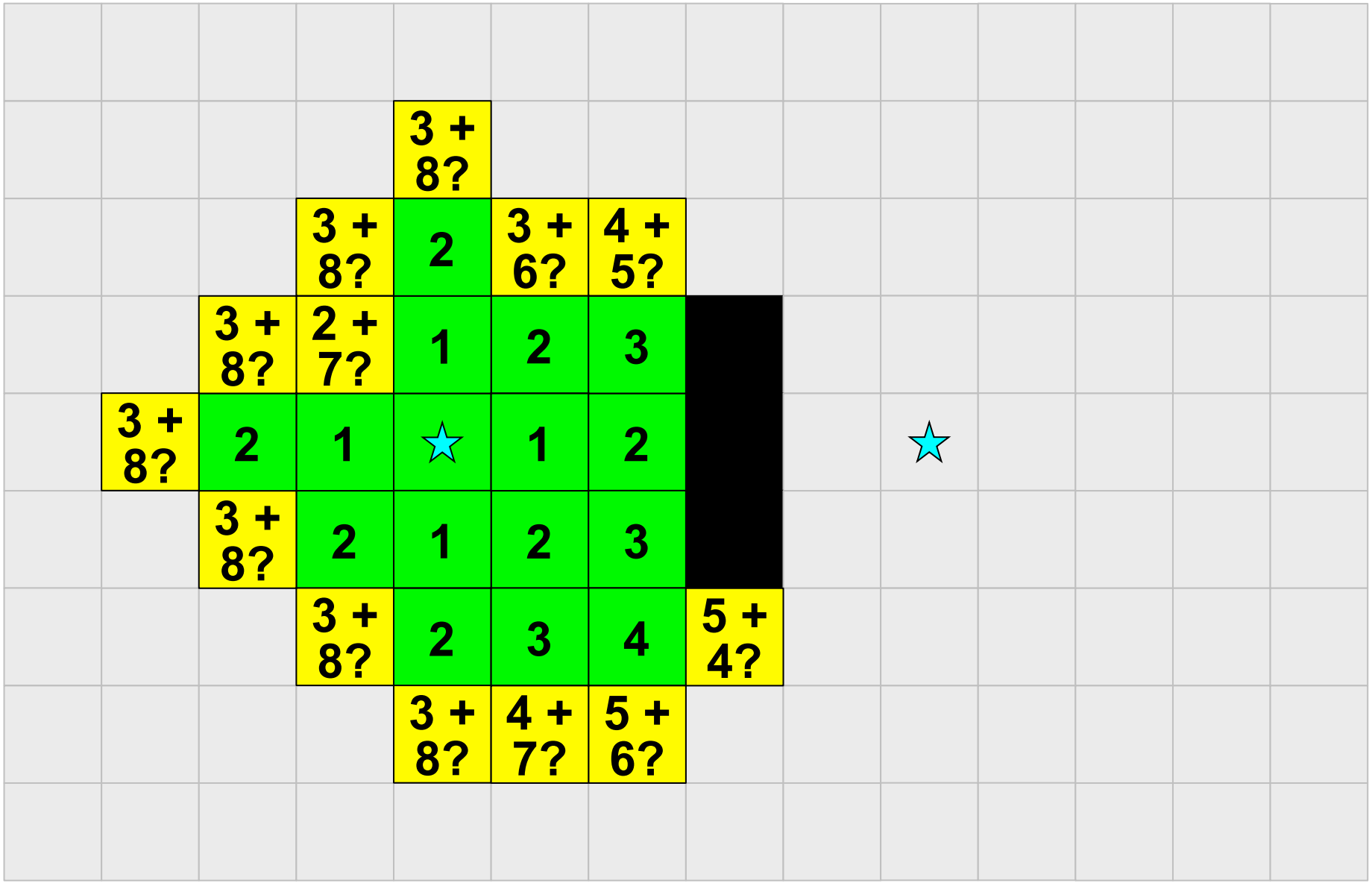

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3 + 6?
4
5 + 4?
5 + 6?

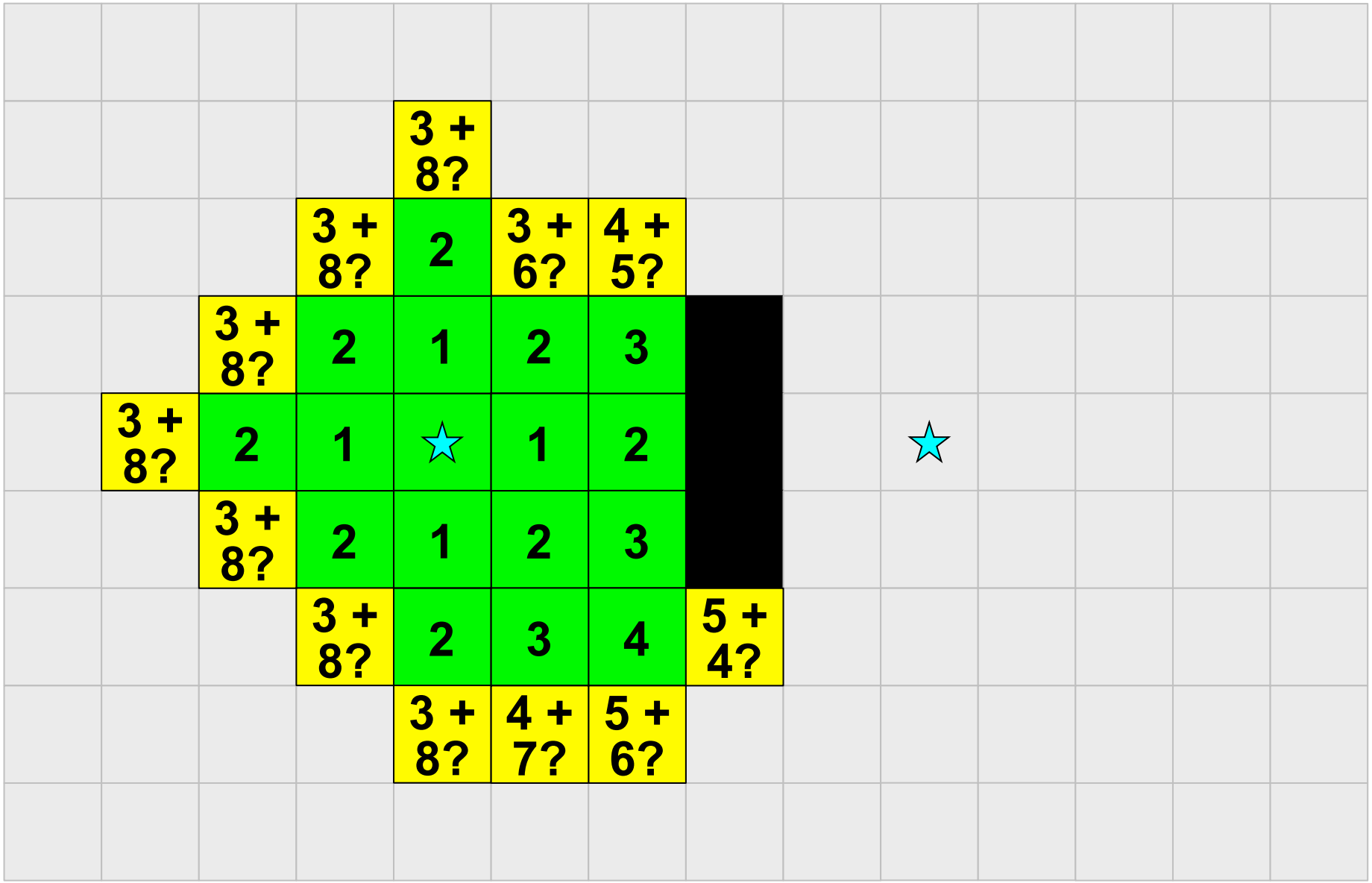

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3 + 6?
4
5 + 4?
3 + 8?
5 + 6?

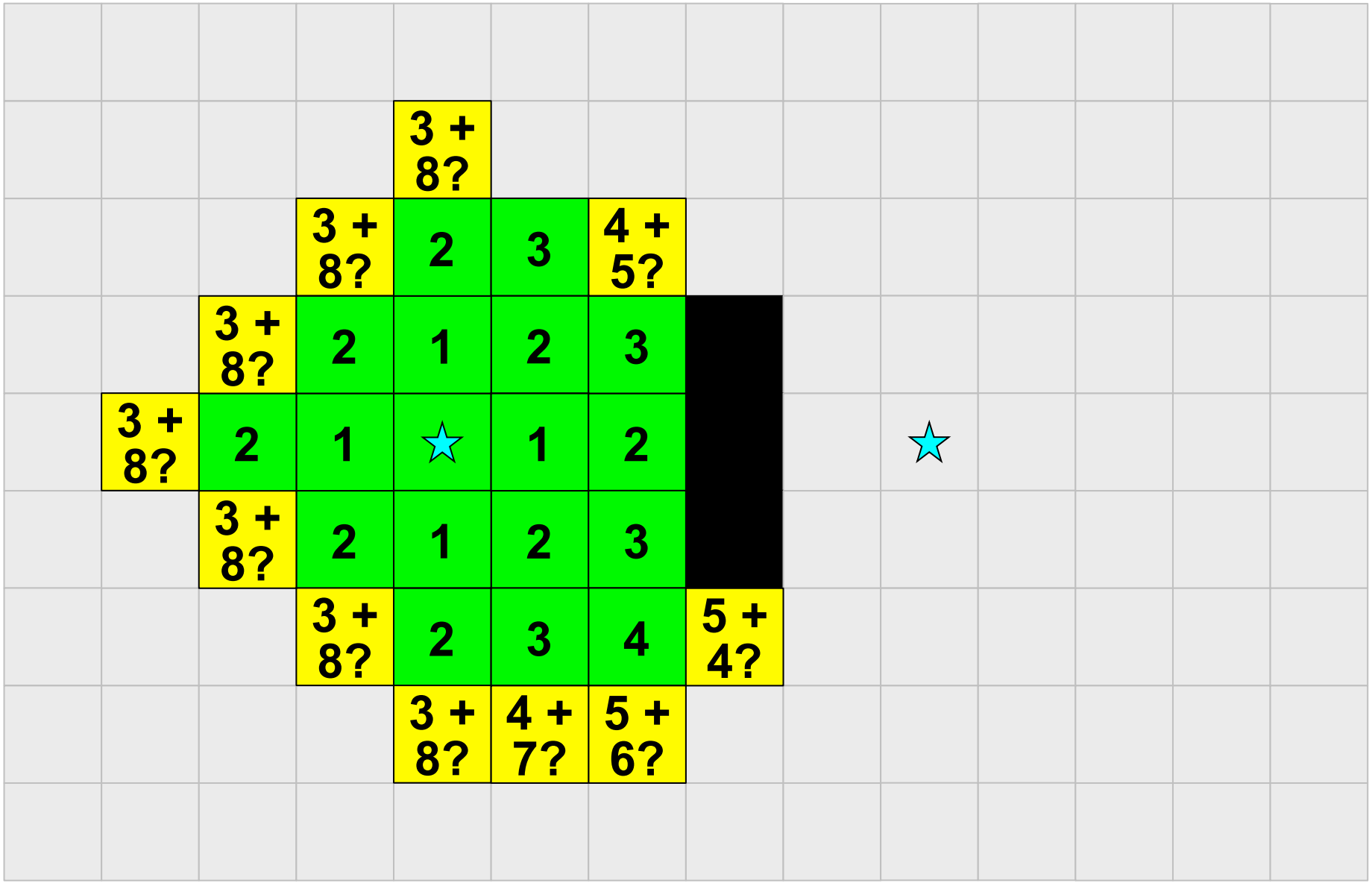

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
5 + 6?

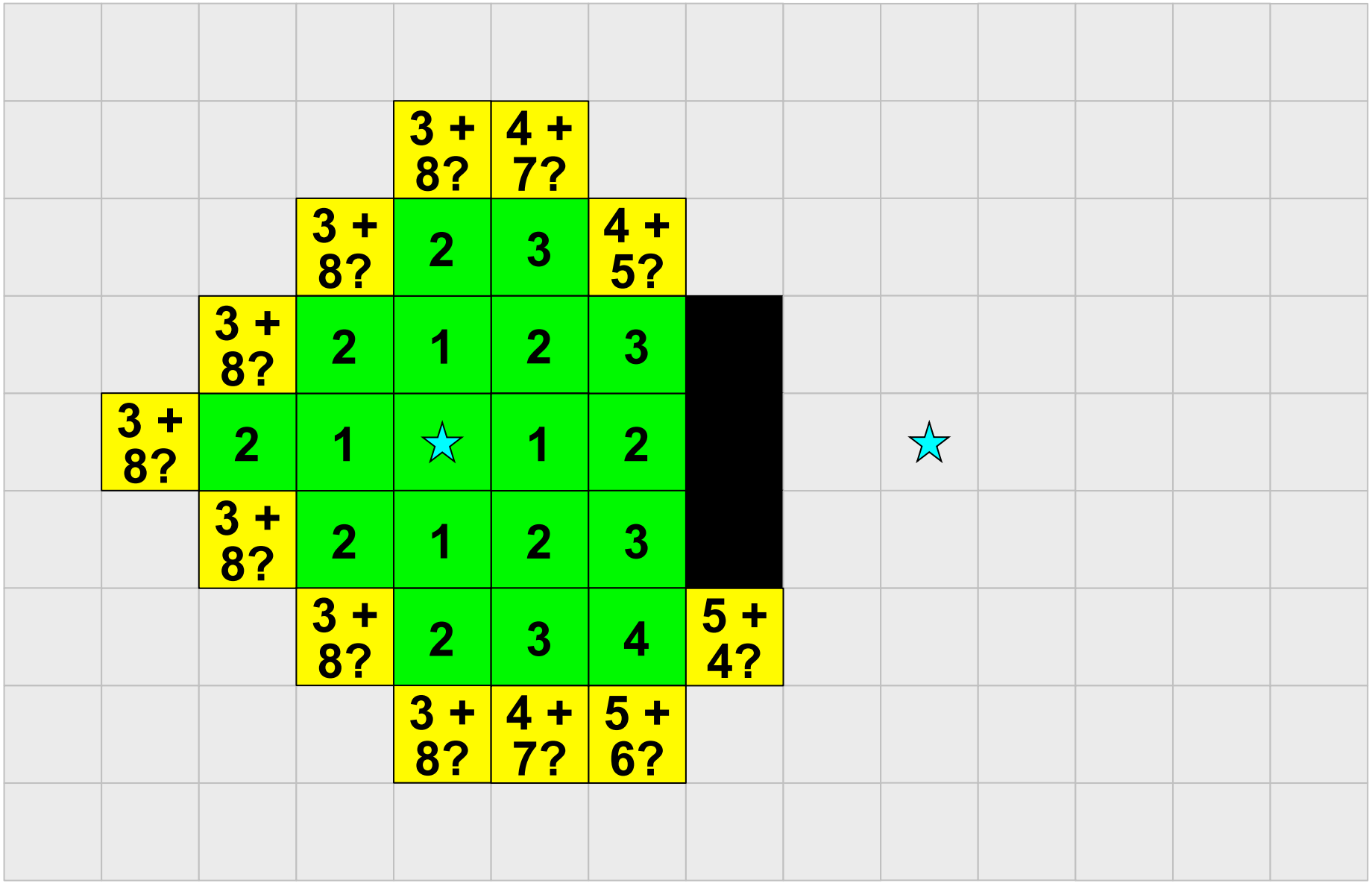

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

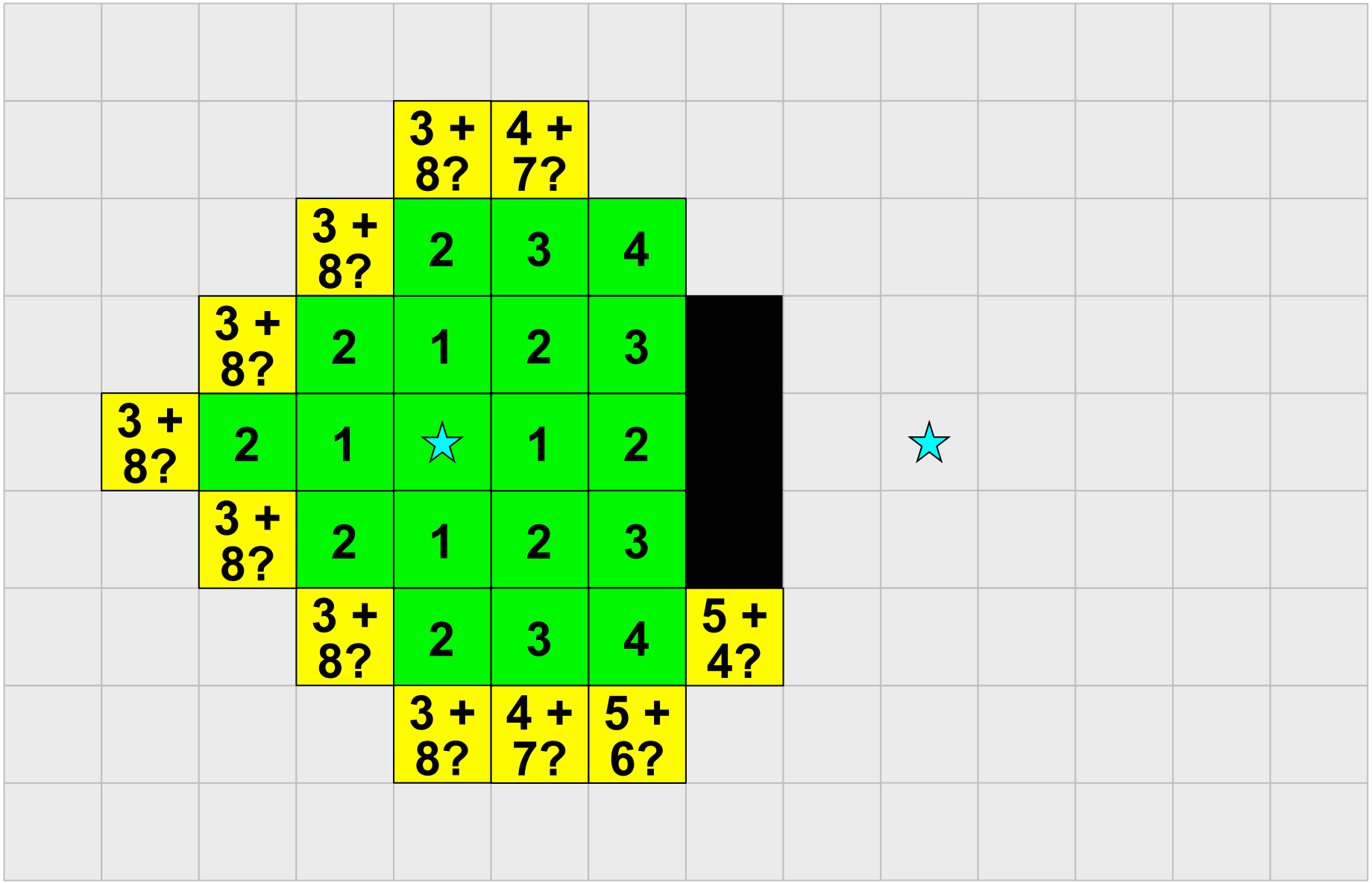

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

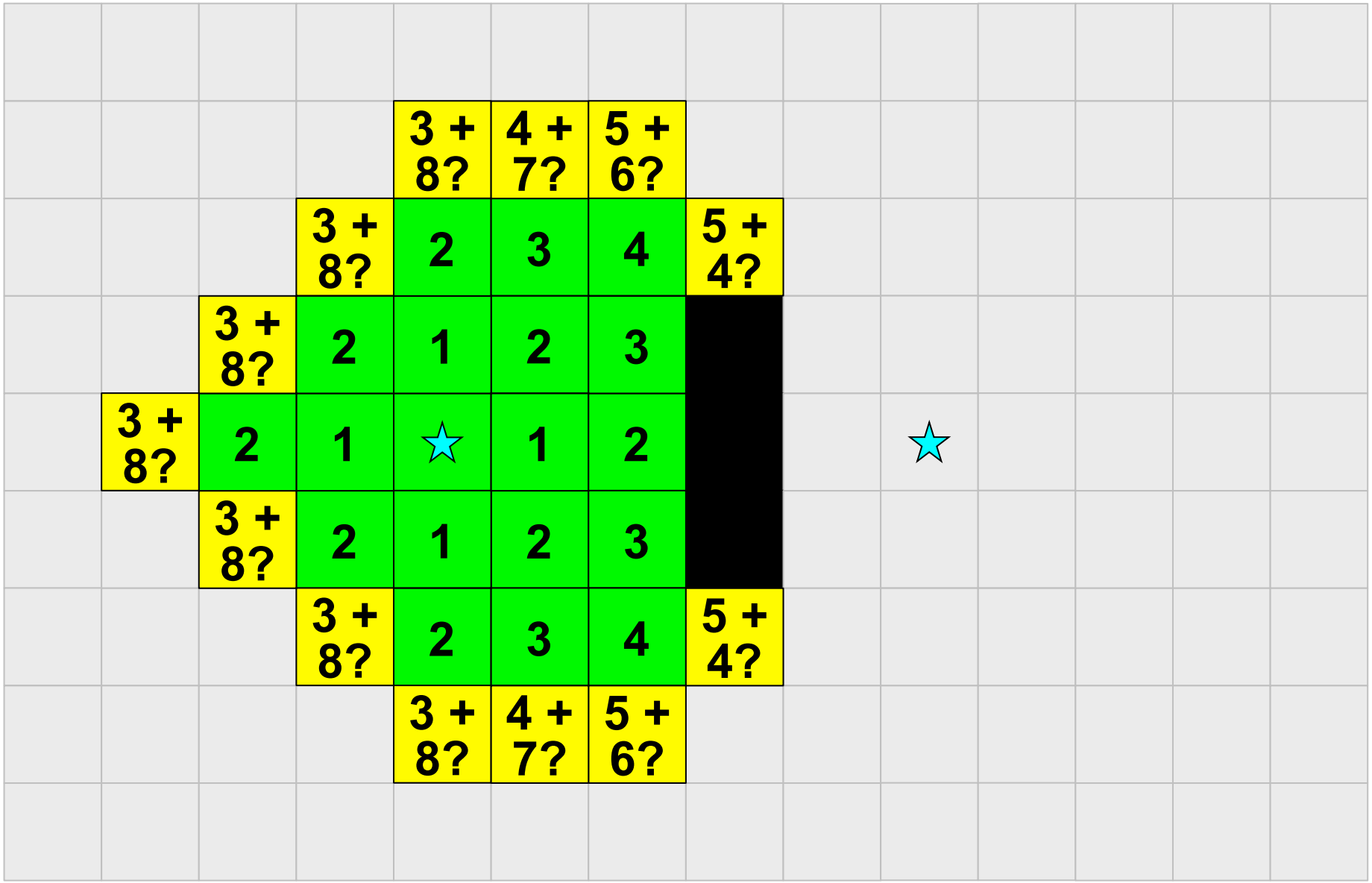

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
3 + 8?
2 + 7?
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

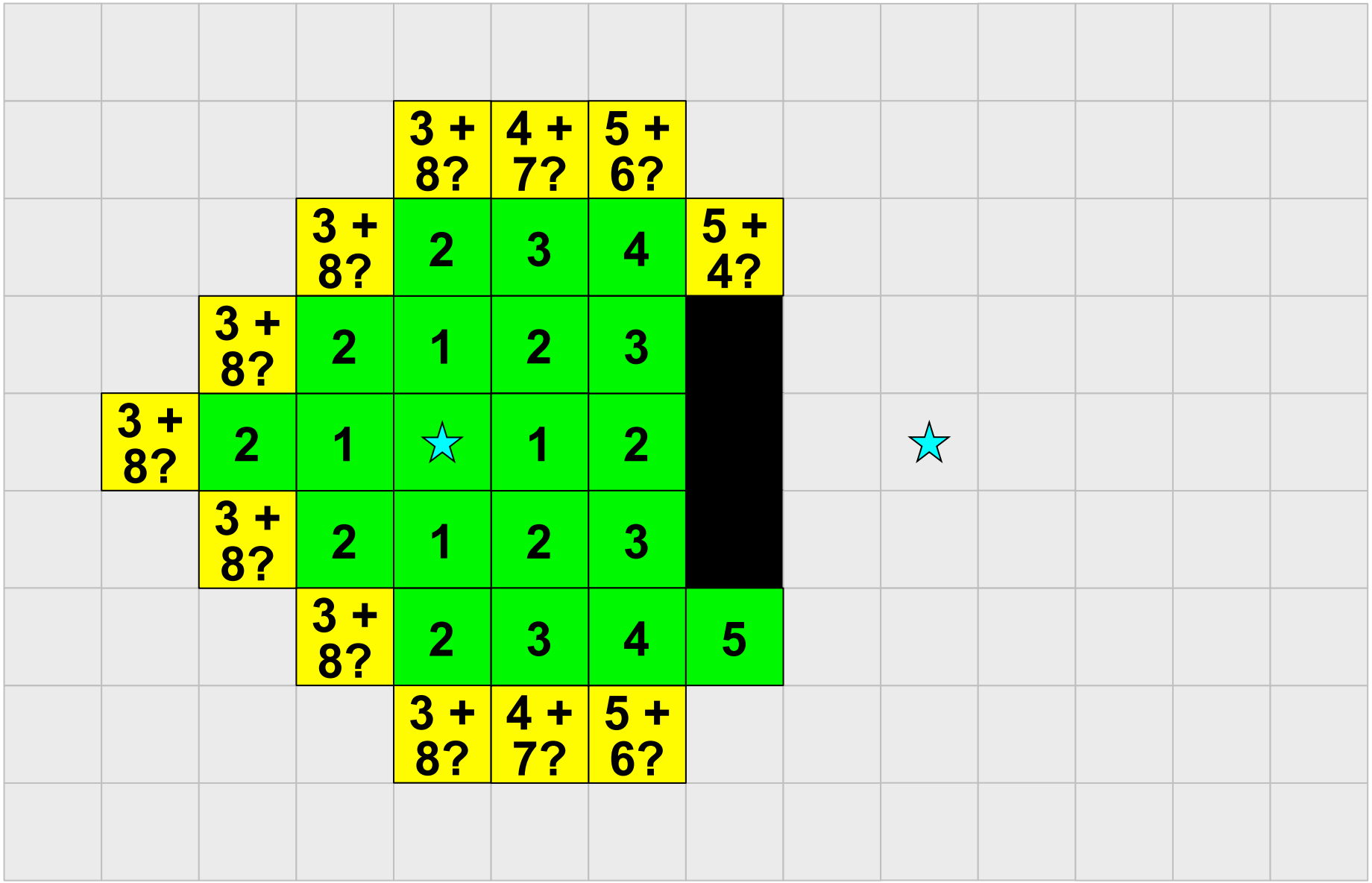

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

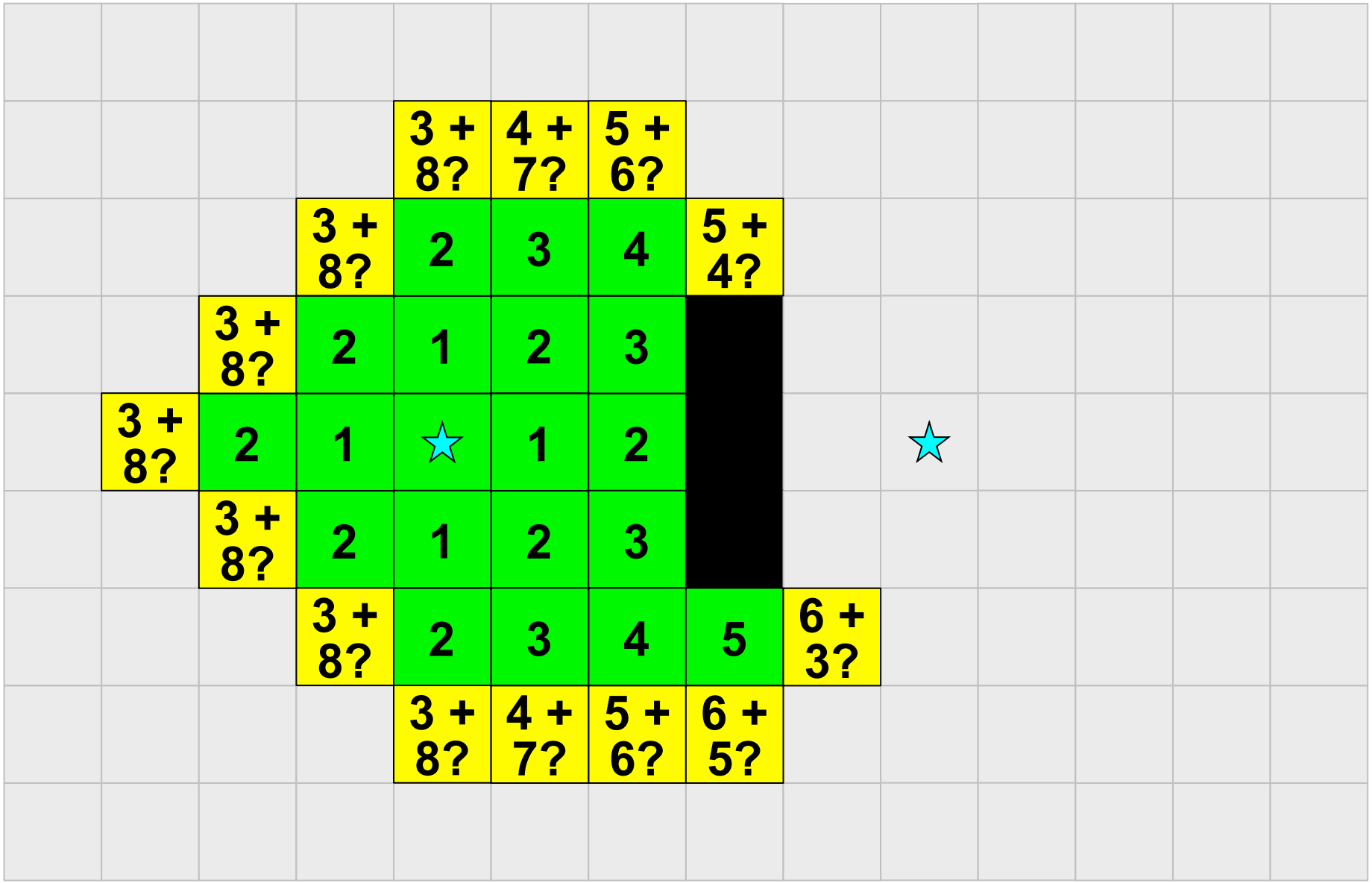

3 + 8?
3 + 8?
2
3
4 + 5?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

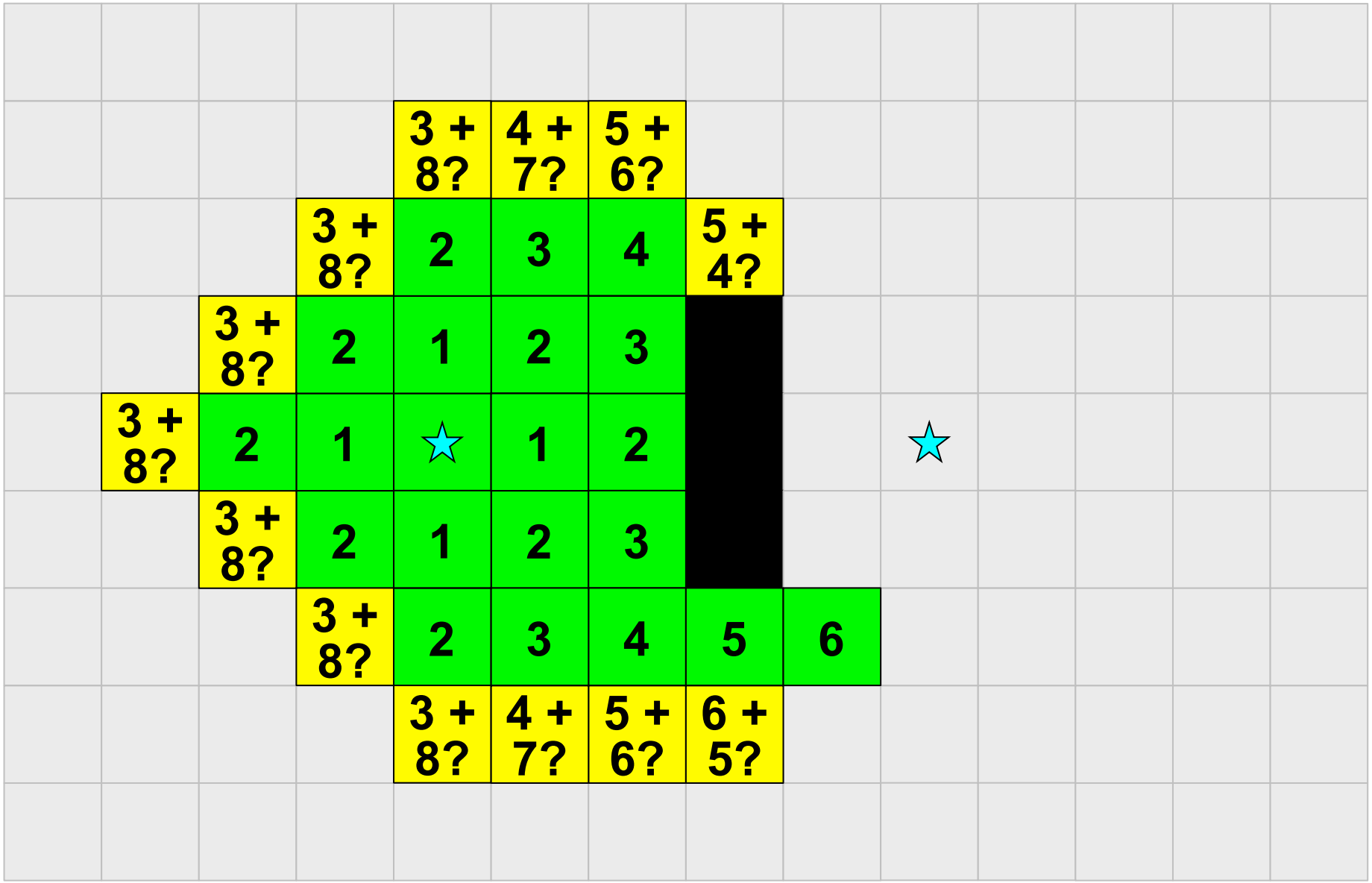

3 + 8?
4 + 7?
3 + 8?
2
3
4 + 5?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

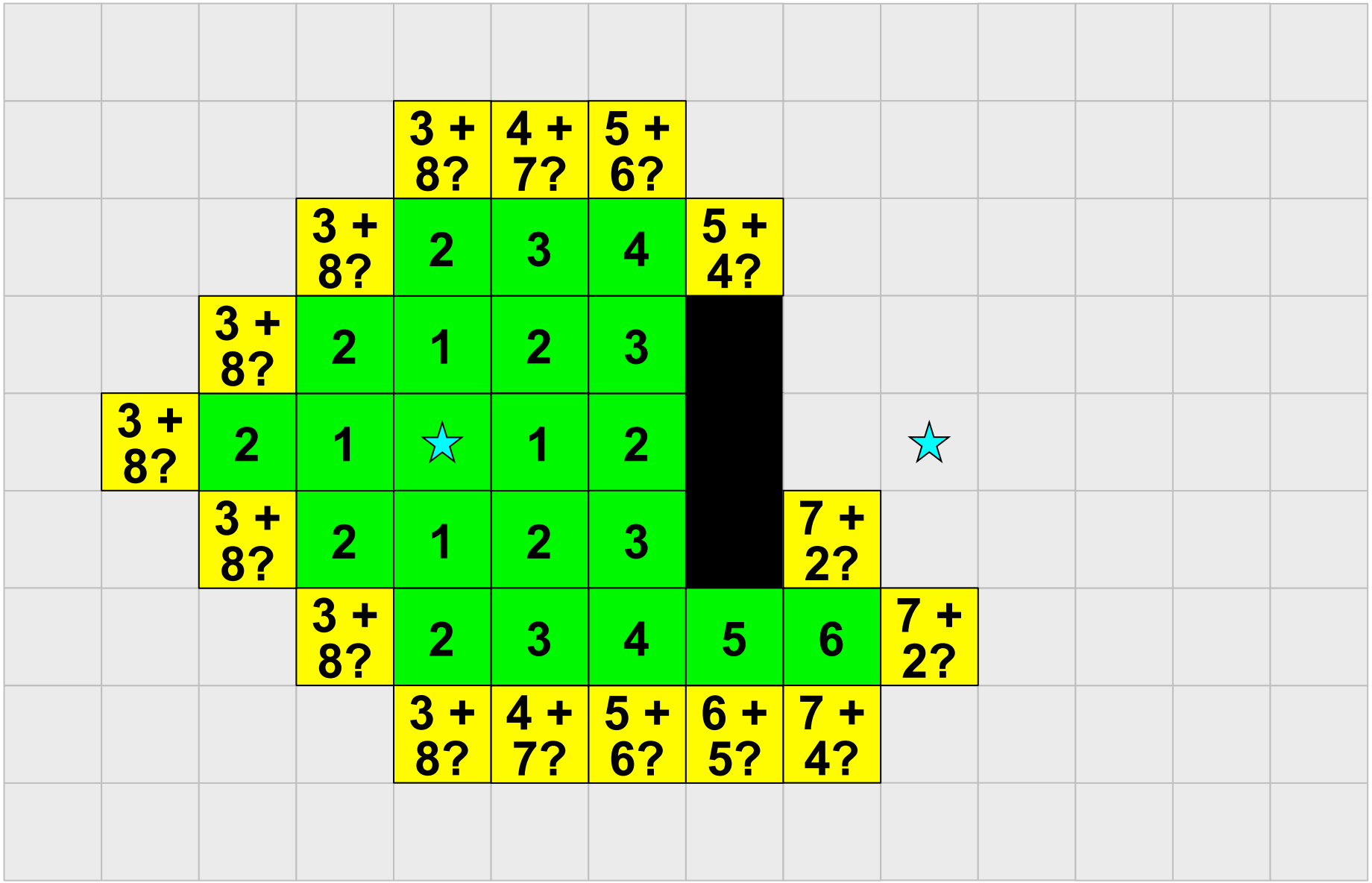

3 + 8?
4 + 7?
3 + 8?
2
3
4
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

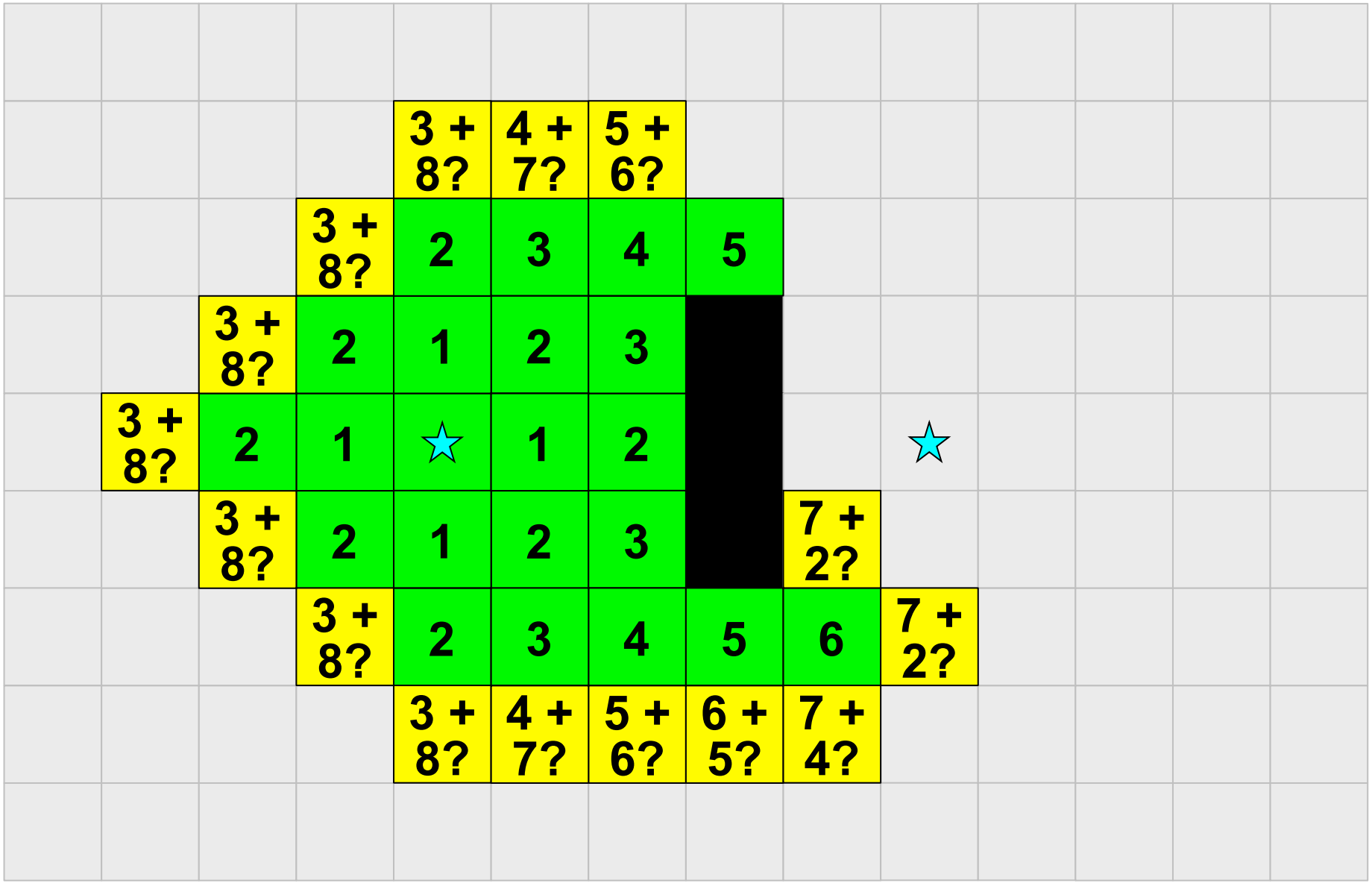

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5 + 4?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

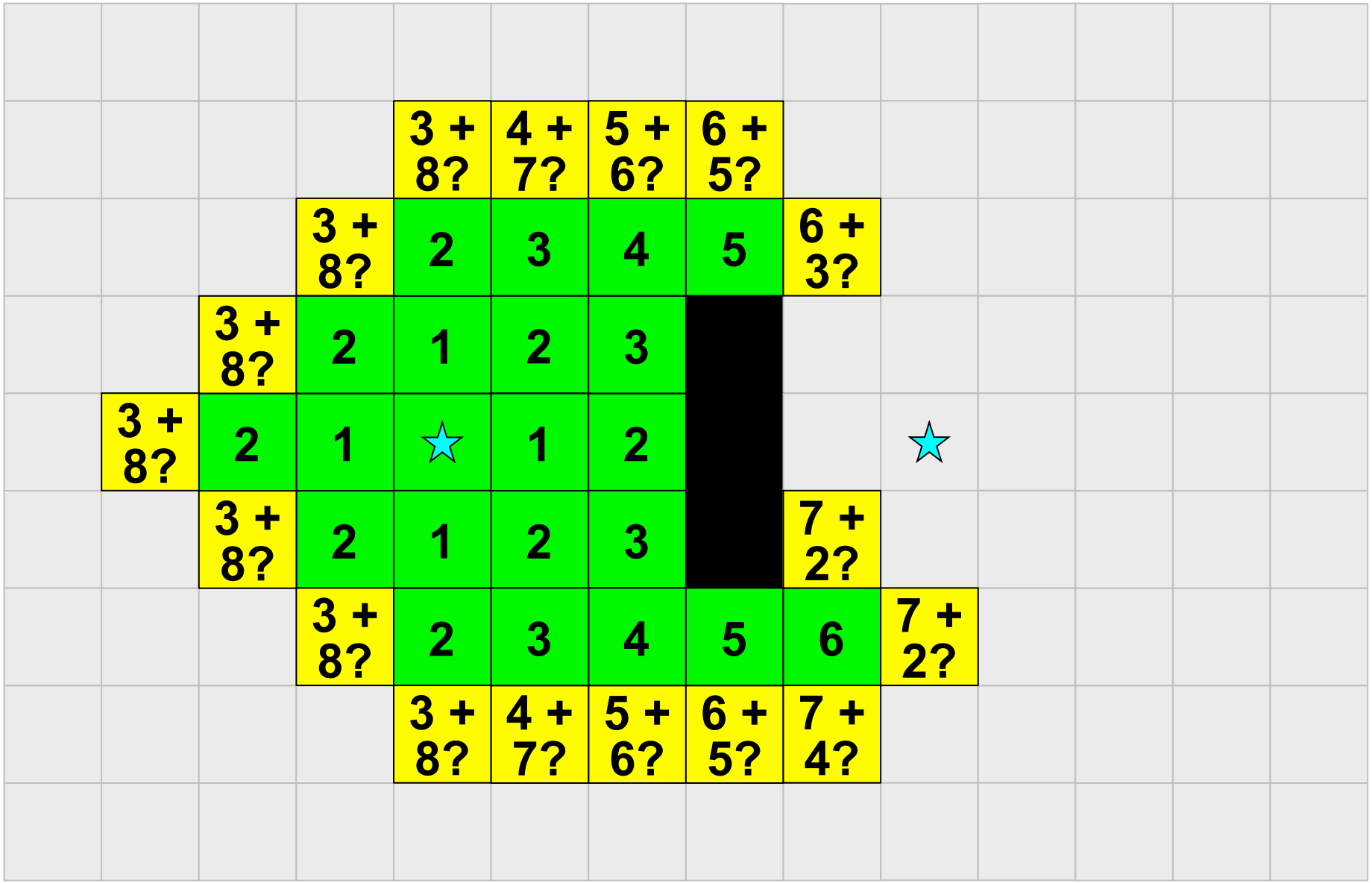

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5 + 4?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5
3 + 8?
4 + 7?
5 + 6?

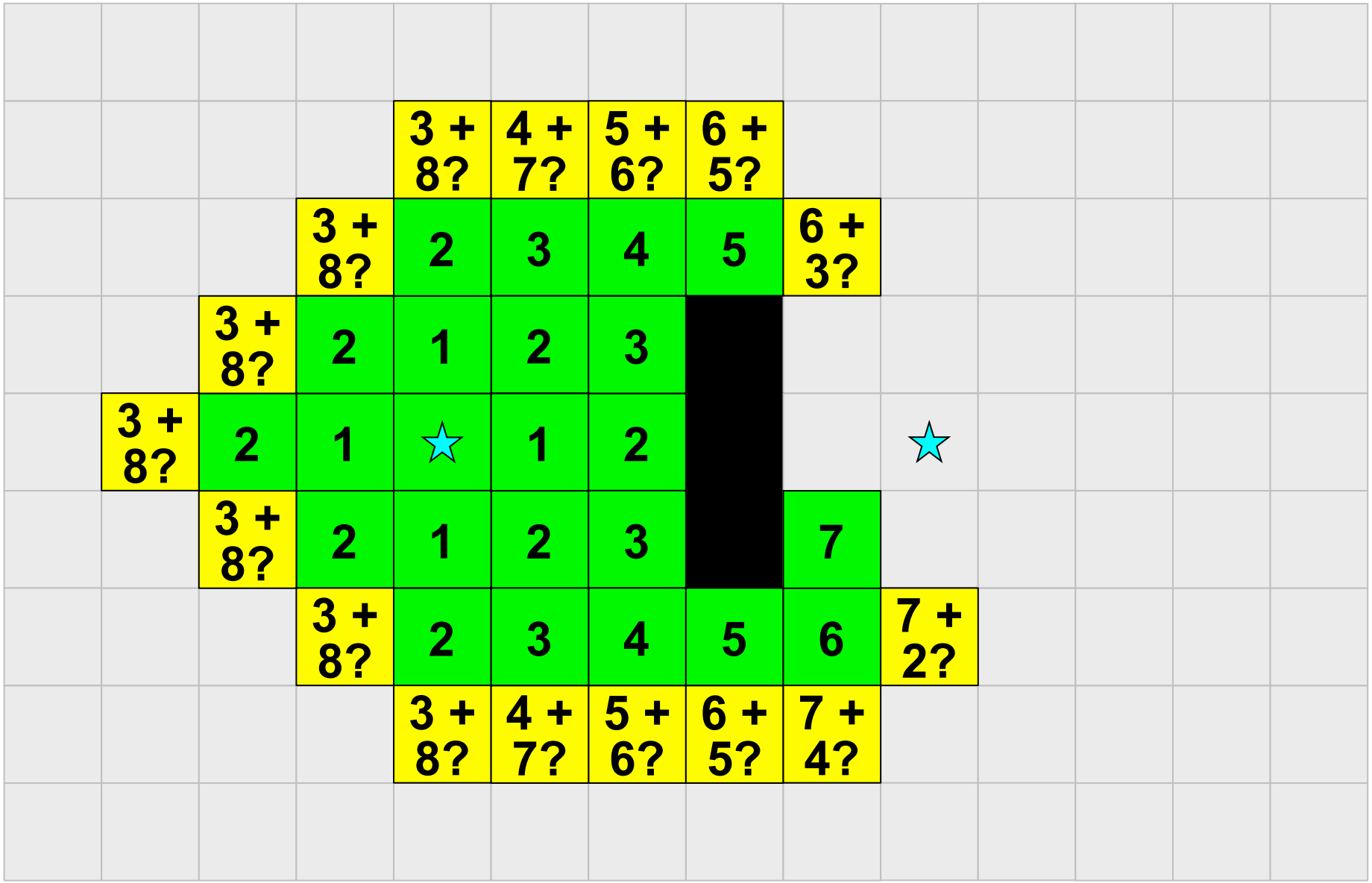

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5 + 4?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5
6 + 3?
3 + 8?
4 + 7?
5 + 6?
6 + 5?

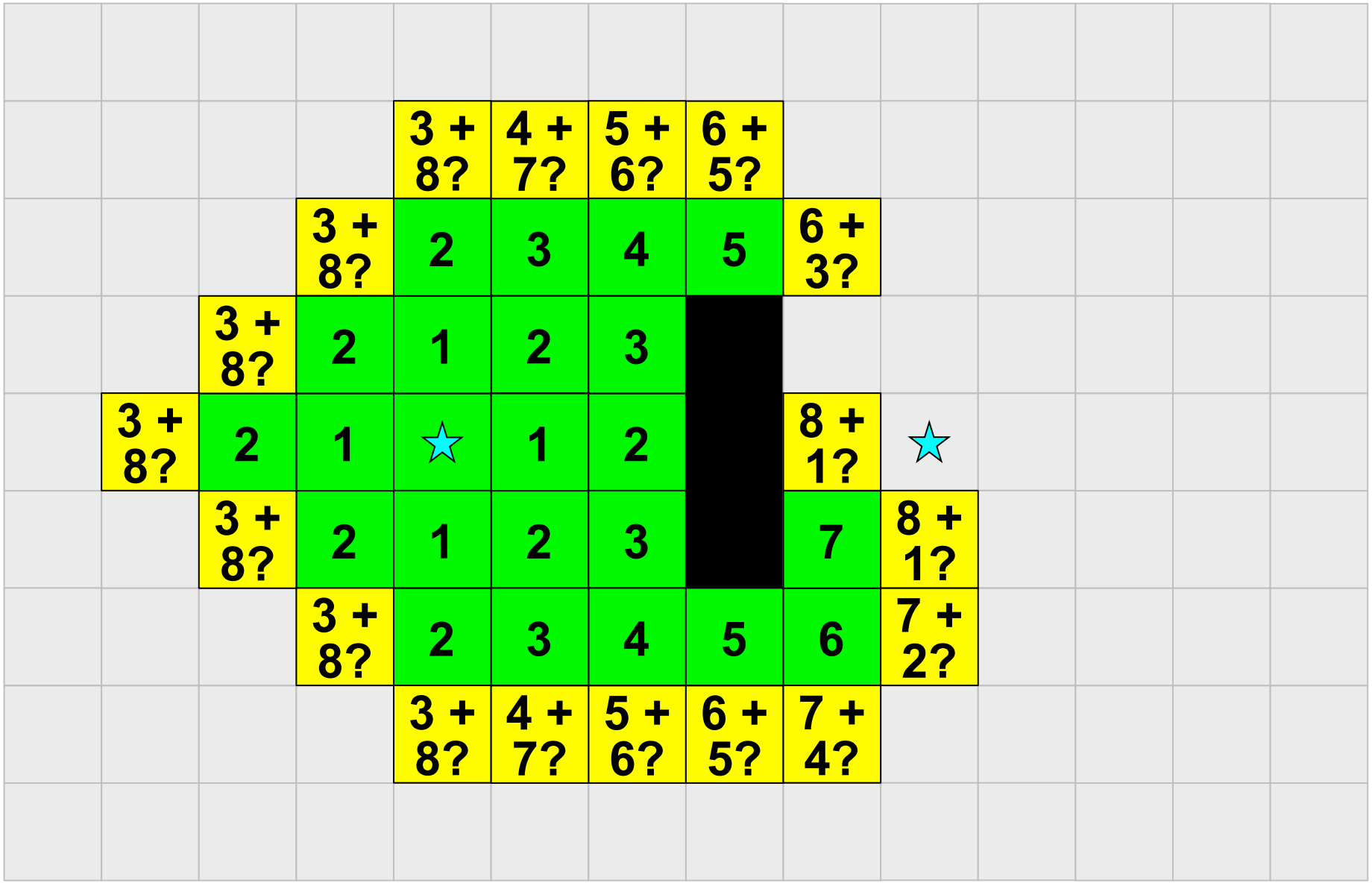

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5 + 4?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5
6
3 + 8?
4 + 7?
5 + 6?
6 + 5?

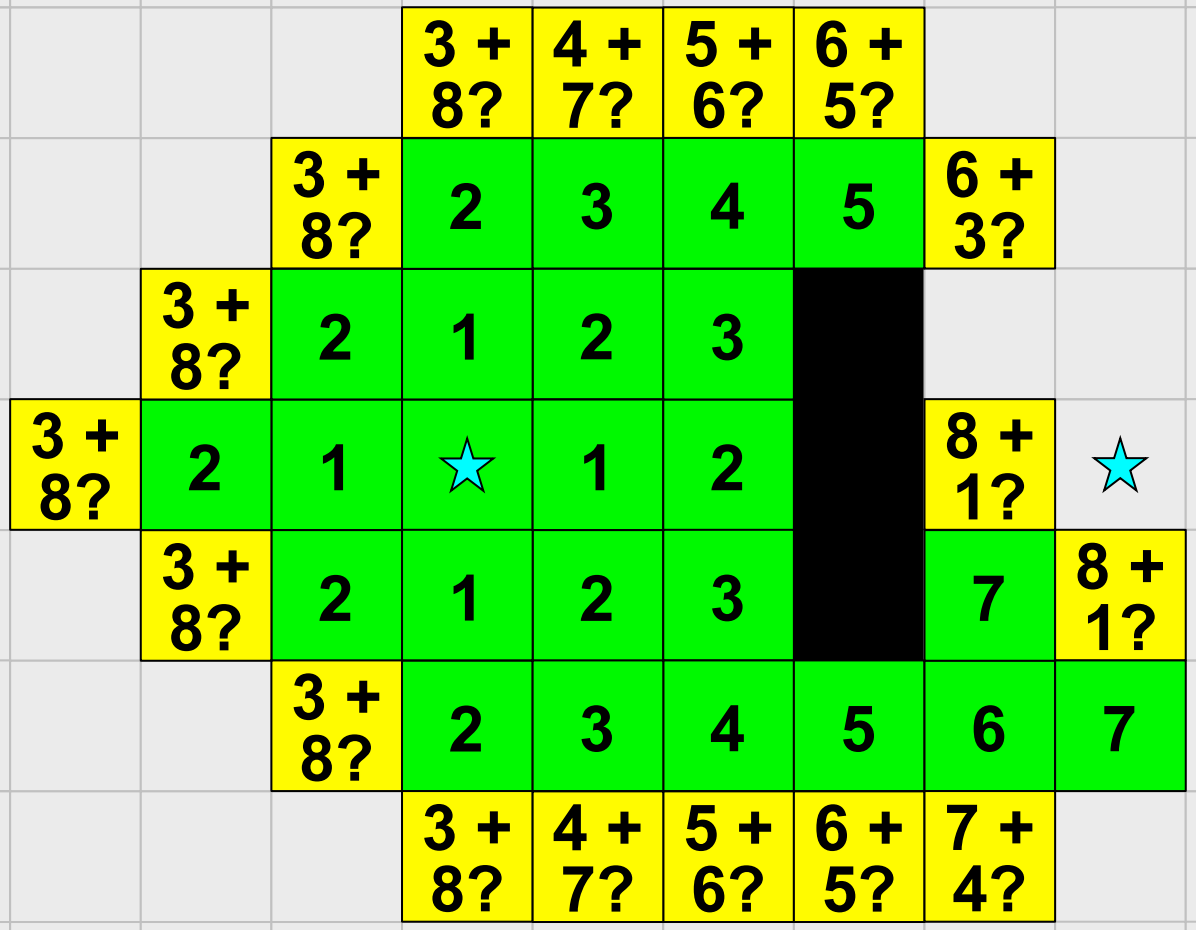

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5 + 4?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
7 + 2?
3 + 8?
2
3
4
5
6
7 + 2?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?

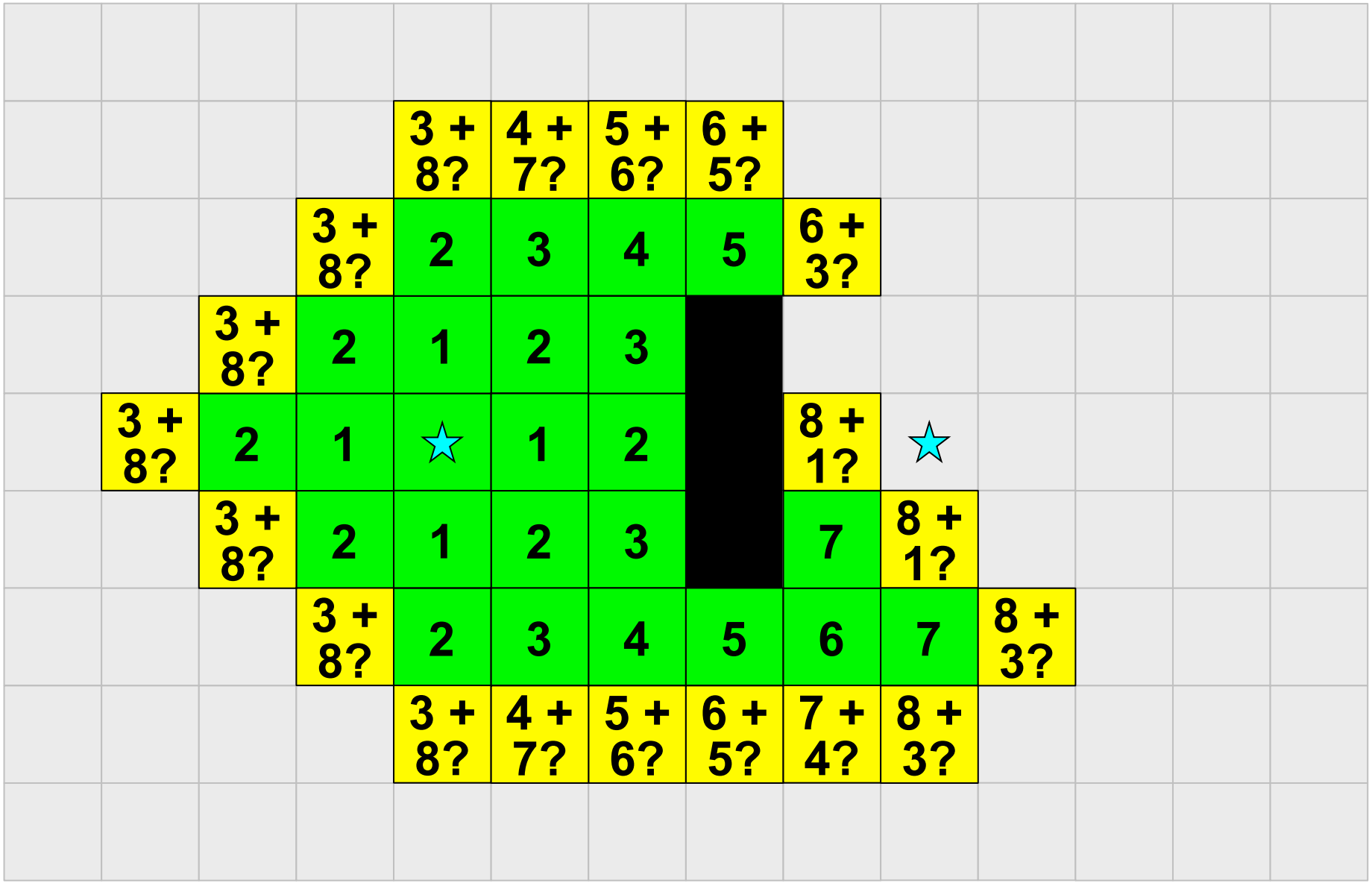

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
7 + 2?
3 + 8?
2
3
4
5
6
7 + 2?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?

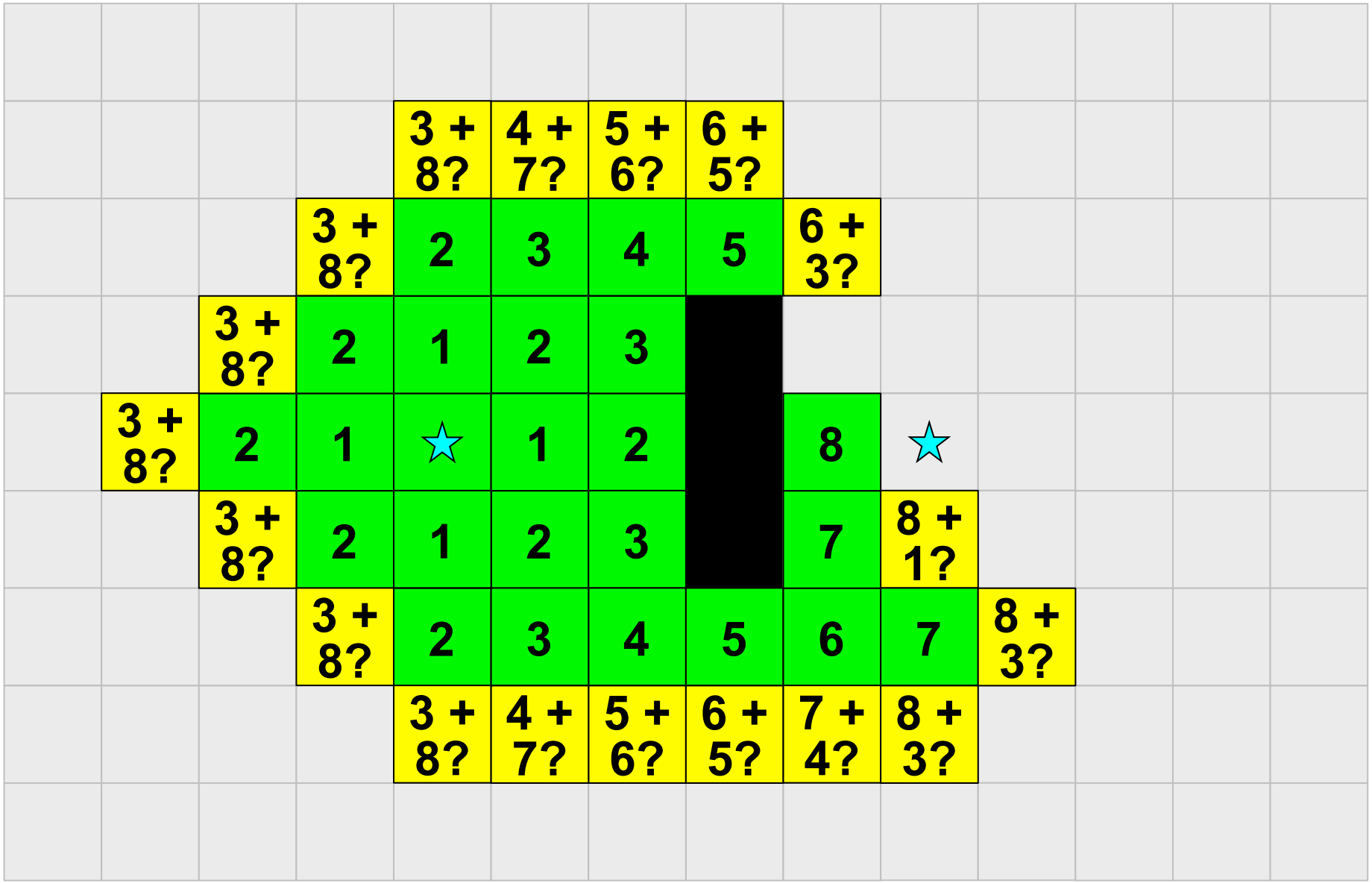

3 + 8?
4 + 7?
5 + 6?
6 + 5?
3 + 8?
2
3
4
5
6 + 3?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
7 + 2?
3 + 8?
2
3
4
5
6
7 + 2?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?

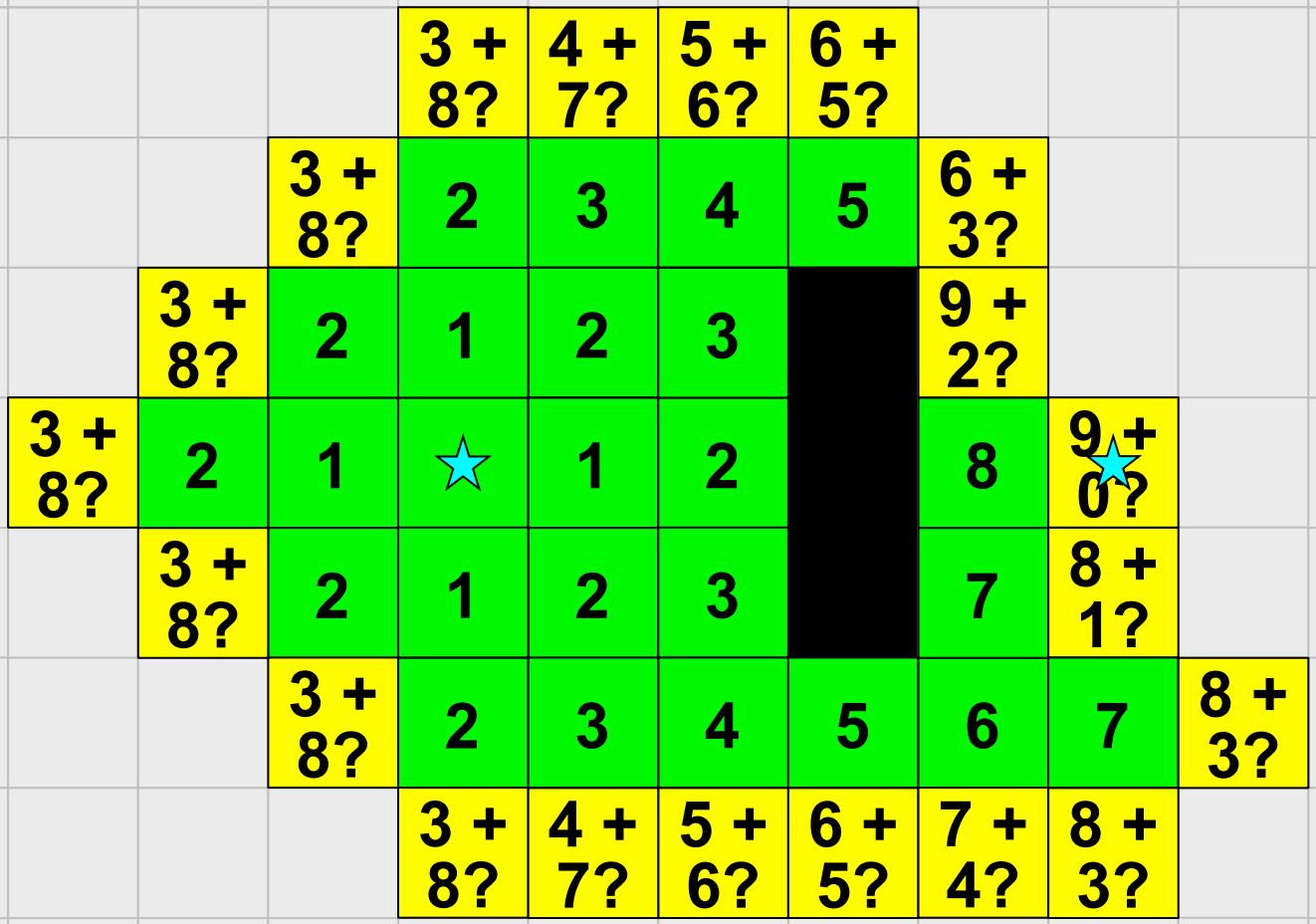

3 + 8?
4 + 7?
5 + 6?
6 + 5?
3 + 8?
2
3
4
5
6 + 3?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
7
3 + 8?
2
3
4
5
6
7 + 2?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?

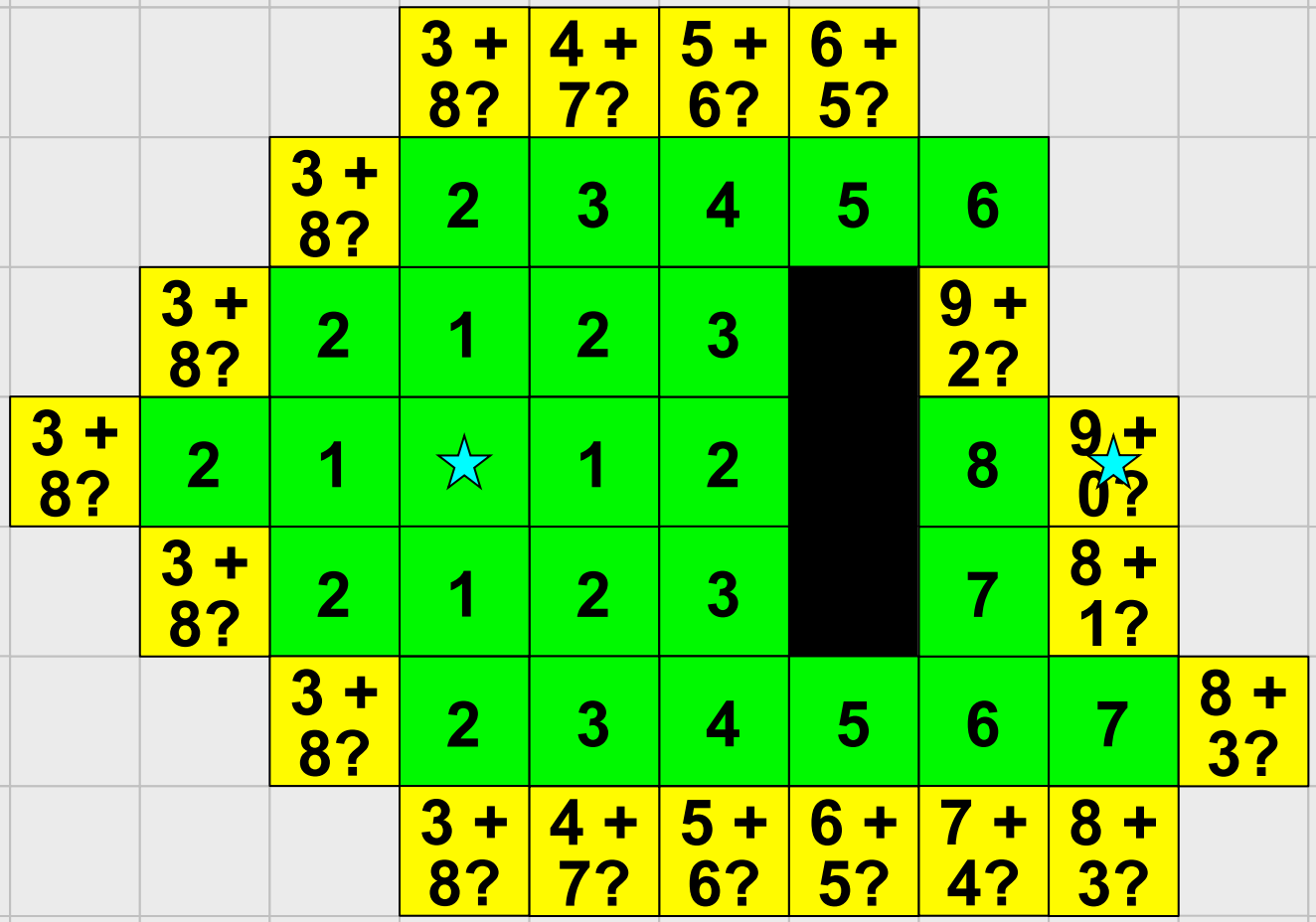

3 + 8?
4 + 7?
5 + 6?
6 + 5?
3 + 8?
2
3
4
5
6 + 3?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
8 + 1?
3 + 8?
2
1
2
3
7
8 + 1?
3 + 8?
2
3
4
5
6
7 + 2?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?

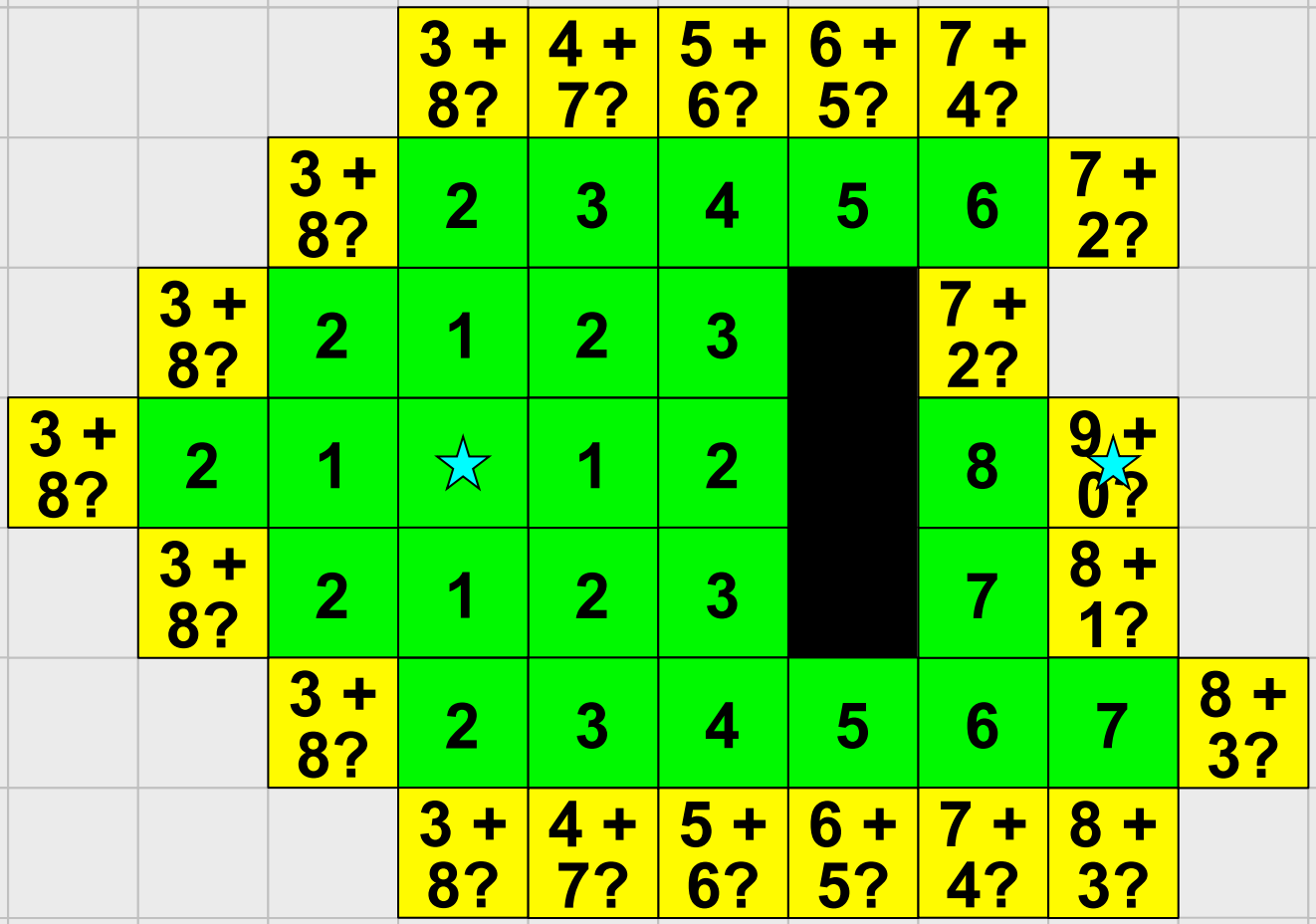

3 + 8?
4 + 7?
5 + 6?
6 + 5?
3 + 8?
2
3
4
5
6 + 3?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
8 + 1?
3 + 8?
2
1
2
3
7
8 + 1?
3 + 8?
2
3
4
5
6
7
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?

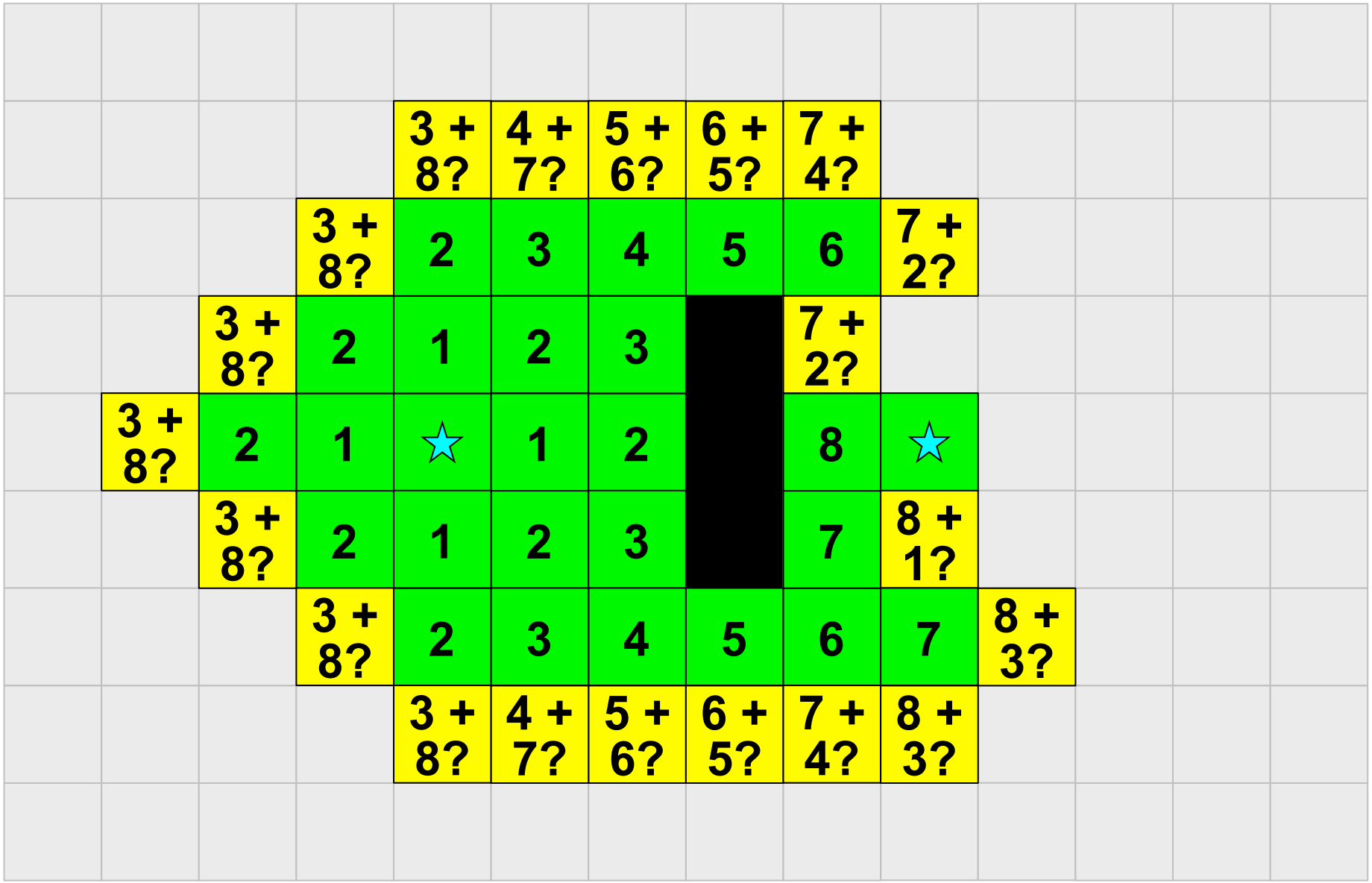

3 + 8?
4 + 7?
5 + 6?
6 + 5?
3 + 8?
2
3
4
5
6 + 3?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
8 + 1?
3 + 8?
2
1
2
3
7
8 + 1?
3 + 8?
2
3
4
5
6
7
8 + 3?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?
8 + 3?


3 + 8?
4 + 7?
5 + 6?
6 + 5?
3 + 8?
2
3
4
5
6 + 3?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
8
3 + 8?
2
1
2
3
7
8 + 1?
3 + 8?
2
3
4
5
6
7
8 + 3?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?
8 + 3?


3 + 8?
4 + 7?
5 + 6?
6 + 5?
3 + 8?
2
3
4
5
6 + 3?
3 + 8?
2
1
2
3
9 + 2?
3 + 8?
2
1
1
2
8
9 + 0?
3 + 8?
2
1
2
3
7
8 + 1?
3 + 8?
2
3
4
5
6
7
8 + 3?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?
8 + 3?


3 + 8?
4 + 7?
5 + 6?
6 + 5?
3 + 8?
2
3
4
5
6
3 + 8?
2
1
2
3
9 + 2?
3 + 8?
2
1
1
2
8
9 + 0?
3 + 8?
2
1
2
3
7
8 + 1?
3 + 8?
2
3
4
5
6
7
8 + 3?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?
8 + 3?


3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?
3 + 8?
2
3
4
5
6
7 + 2?
3 + 8?
2
1
2
3
7 + 2?
3 + 8?
2
1
1
2
8
9 + 0?
3 + 8?
2
1
2
3
7
8 + 1?
3 + 8?
2
3
4
5
6
7
8 + 3?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?
8 + 3?


3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?
3 + 8?
2
3
4
5
6
7 + 2?
3 + 8?
2
1
2
3
7 + 2?
3 + 8?
2
1
1
2
8
3 + 8?
2
1
2
3
7
8 + 1?
3 + 8?
2
3
4
5
6
7
8 + 3?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?
8 + 3?

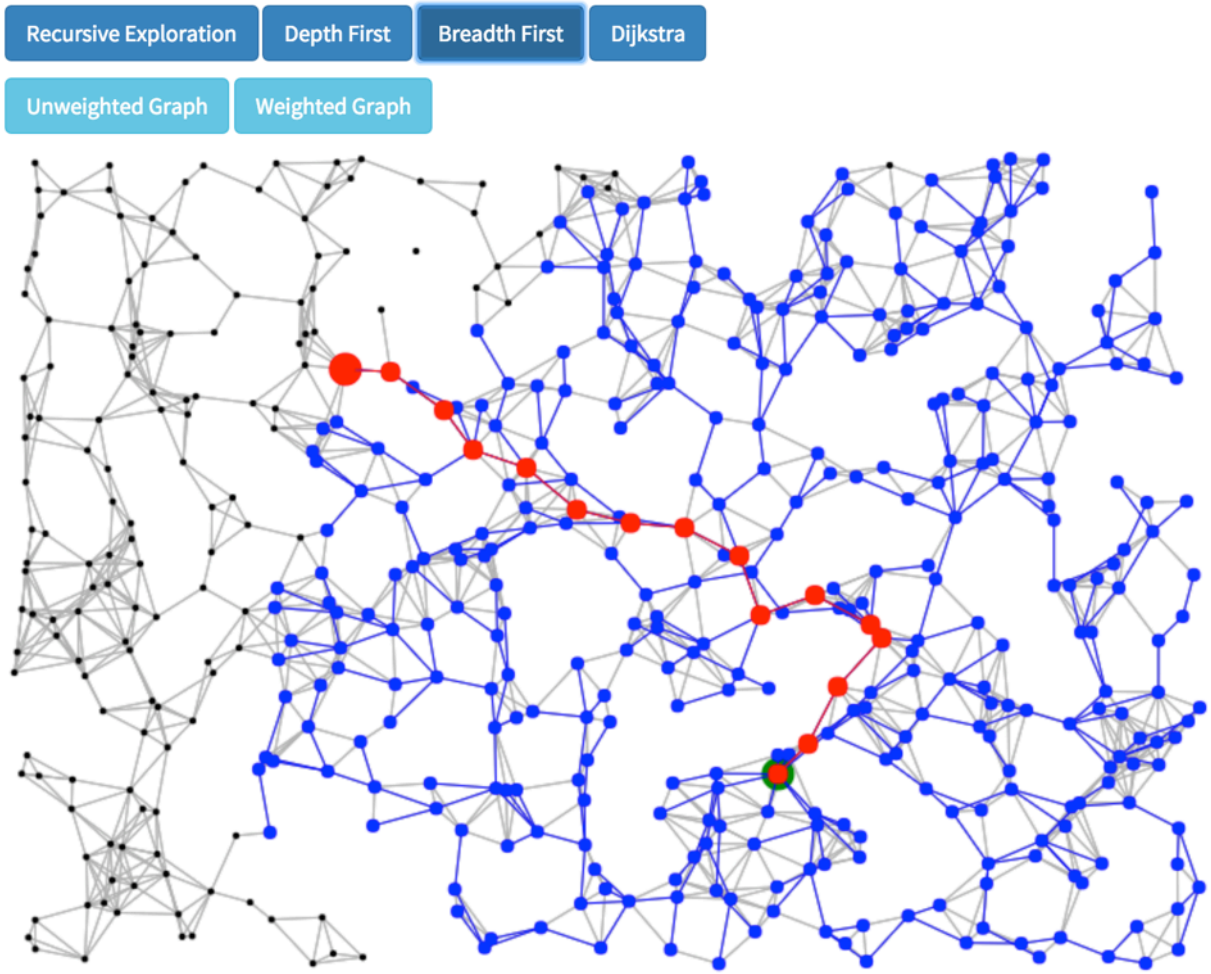
For Comparison: What Dijkstra's Algorithm Would Have Searched

8	7	6	5	4	5	6	7	8	9?				
7	6	5	4	3	4	5	6	7	8	9?			
6	5	4	3	2	3	4	5	6	7	8	9?		
5	4	3	2	1	2	3		7	8	9?			
4	3	2	1	★	1	2		8	★				
5	4	3	2	1	2	3		7	8	9?			
6	5	4	3	2	3	4	5	6	7	8	9?		
7	6	5	4	3	4	5	6	7	8	9?			
8	7	6	5	4	5	6	7	8	9?				

Why underestimate?

You only ignore paths that in the best case are worse than your current path.

See It Live



Path hops: 16

Path cost: 16

h(start,goal) = 0?

Admissible!

A-Star

```
pQueue = PriorityQueue()
pQueue.enqueue(newPath(start), h(start,goal))
seen = Set();
while !pQueue.isEmpty():
  currPath = pQueue.dequeue()
  currState = last(currPath);
  if(currState is goal) return currPath;
  if(seen contains currState) continue;
  seen.add(currState);
  for nextState in getNextStates(currState)
    path = newPath(currPath, nextState);
    cost = getCost(path) + h(currState, goal);
    pQueue.enqueue(path, getCost(path)+ cost);
  }
}
```

A-Star

```
pQueue = PriorityQueue()
pQueue.enqueue(newPath(start), 0)
seen = Set();
while !pQueue.isEmpty():
  currPath = pQueue.dequeue()
  currState = last(currPath);
  if(currState is goal) return currPath;
  if(seen contains currState) continue;
  seen.add(currState);
  for nextState in getNextStates(currState)
    path = newPath(currPath, nextState);
    cost = getCost(path) + 0;
    pQueue.enqueue(path, getCost(path)+ cost);
  }
}
```

Dijkstra's

```
pQueue = PriorityQueue()
pQueue.enqueue(newPath(start), 0)
seen = Set();
while !pQueue.isEmpty():
  currPath = pQueue.dequeue()
  currState = last(currPath);
  if(currState is goal) return currPath;
  if(seen contains currState) continue;
  seen.add(currState);
  for nextState in getNextStates(currState)
    path = newPath(currPath, nextState);
    cost = getCost(path);
    pQueue.enqueue(path, getCost(path)+ cost);
  }
}
```

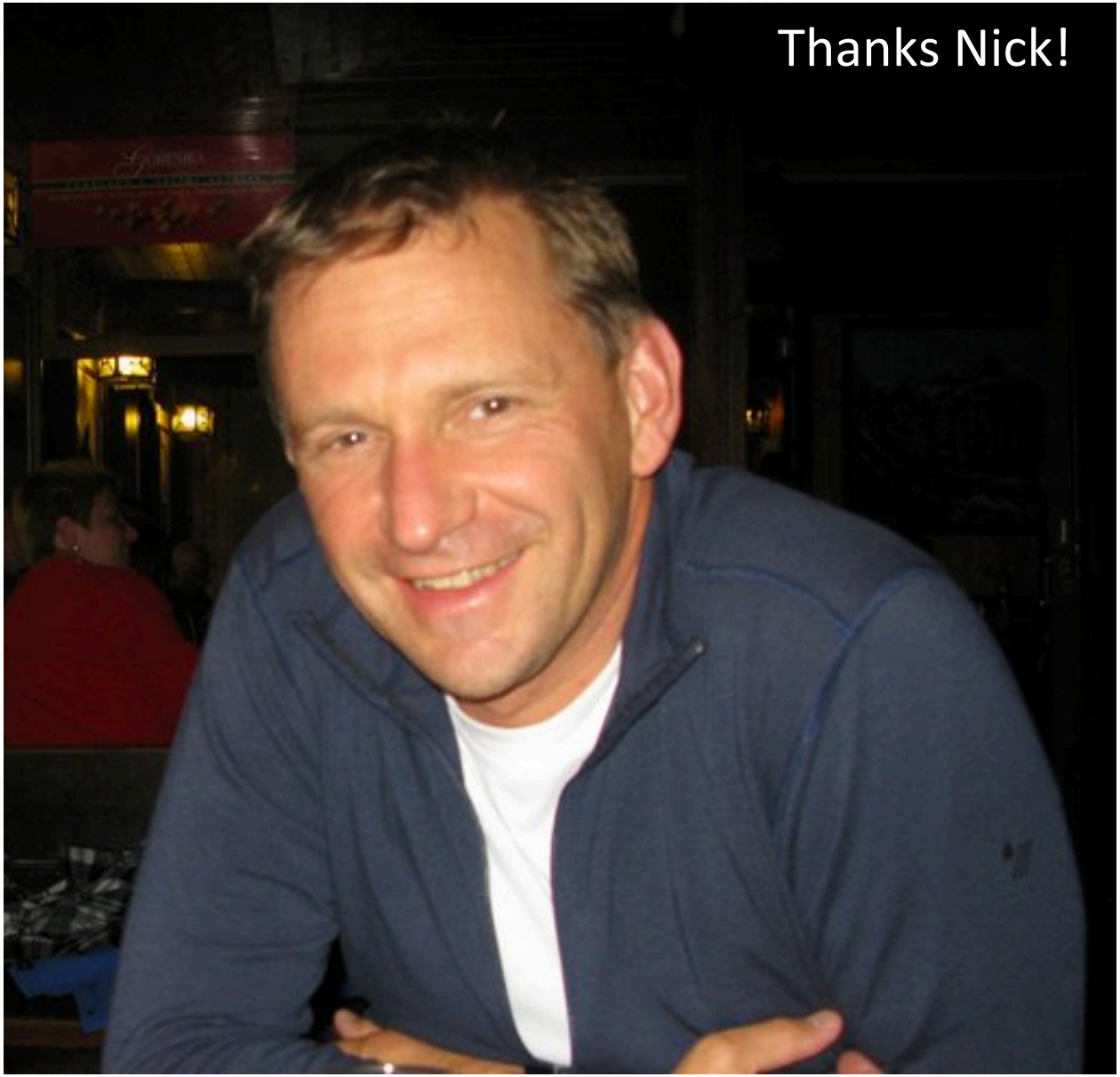
Want the largest admissible heuristic possible

Today's Route

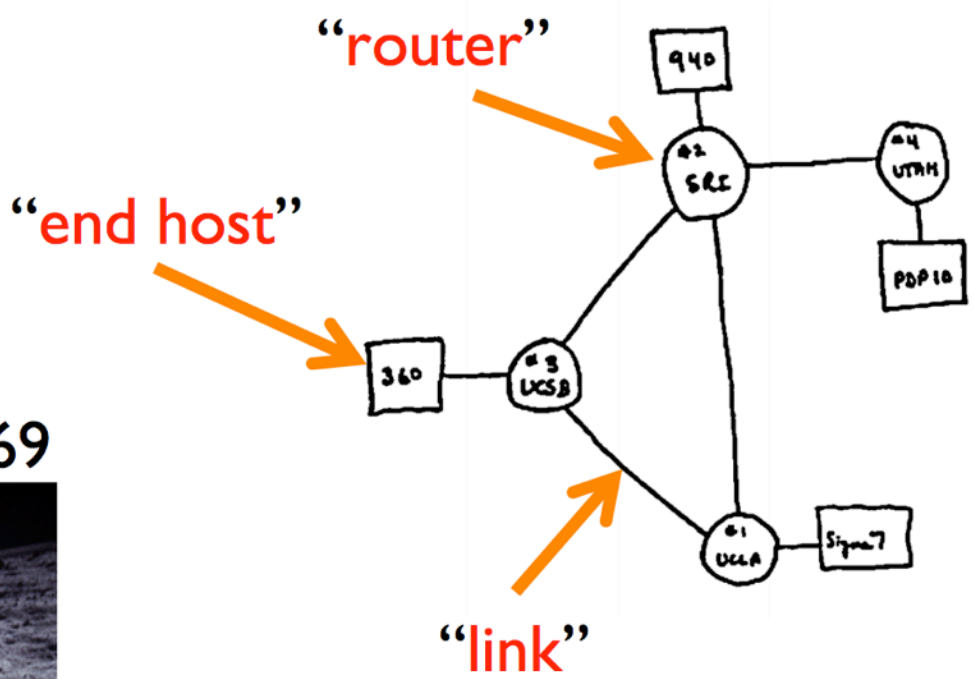
Today's Route

Largest graph you can think of?

Nick McKeown CS144



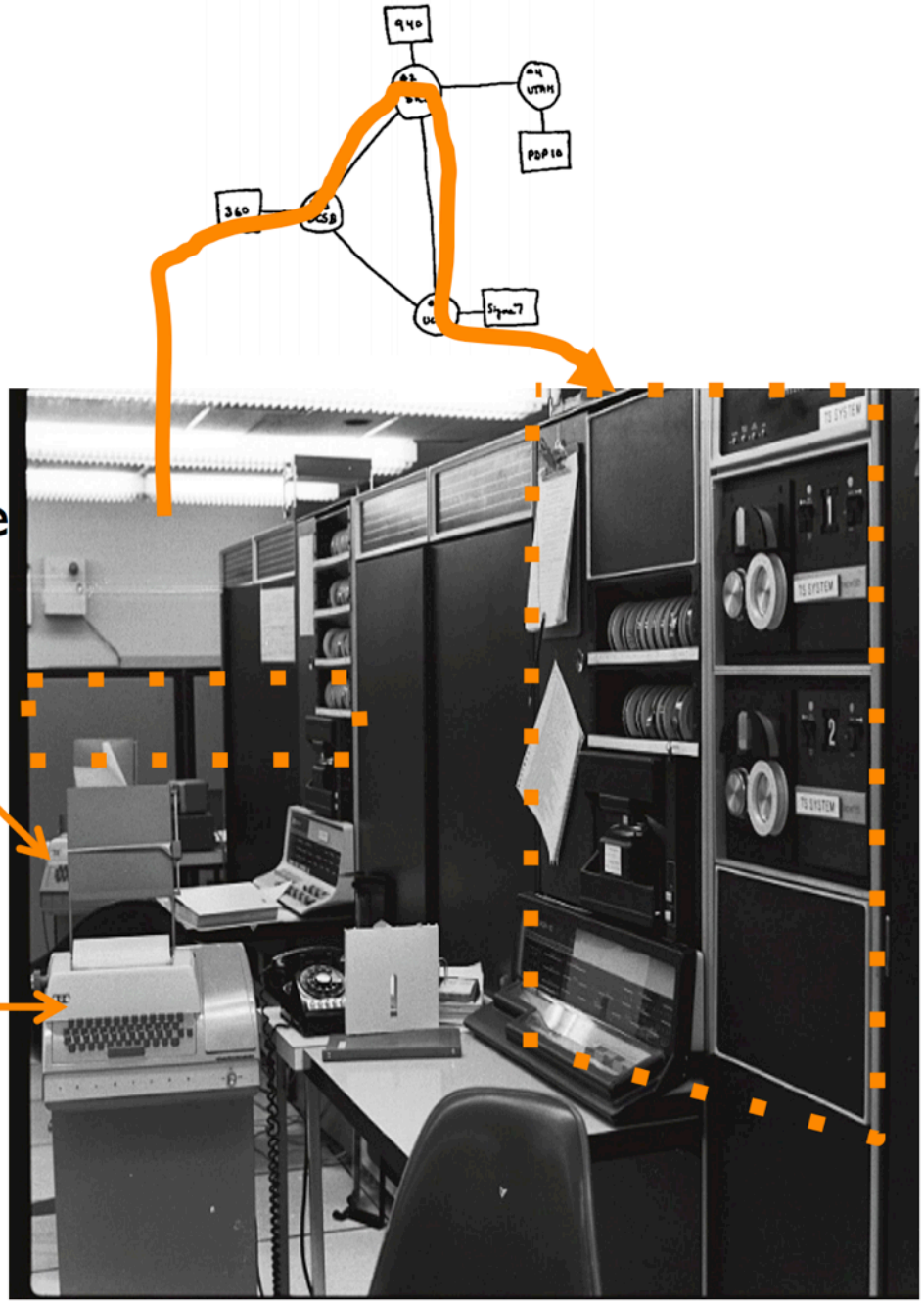
Internet in 1969



Also in 1969

But not this…

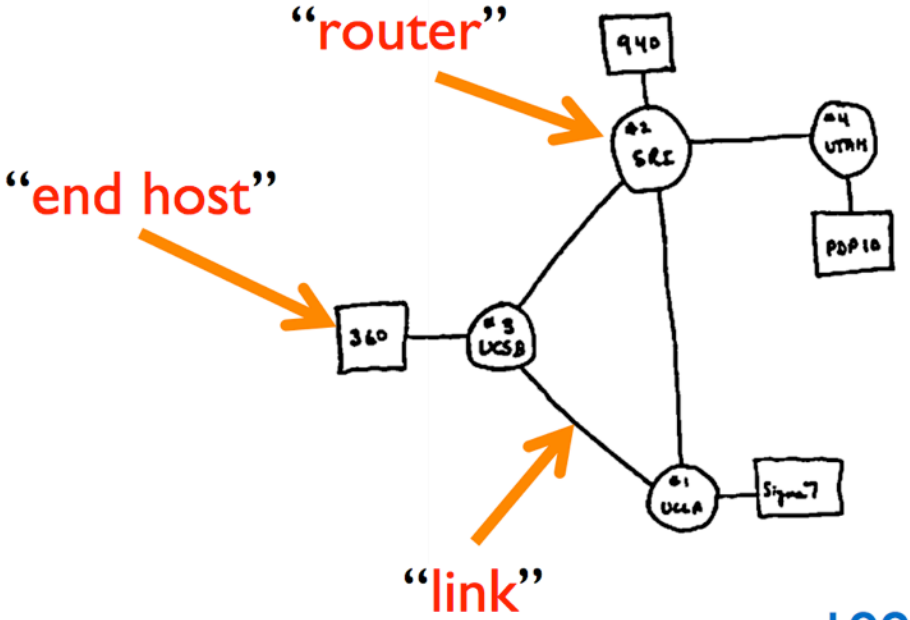
1971



First email typed here

"QWERTYUIOP"

…and printed here



1969

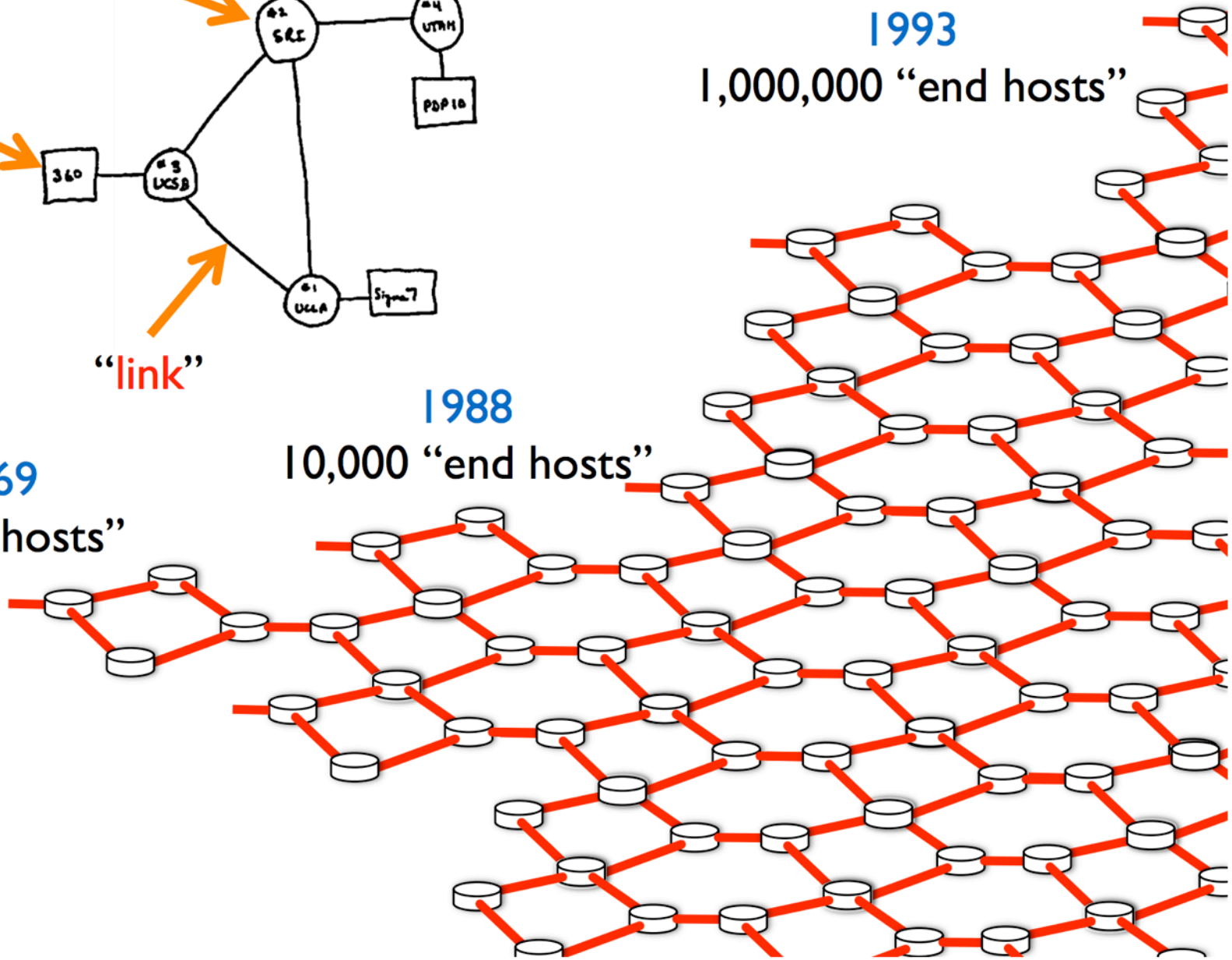
4 "end hosts"

1988

10,000 "end hosts"

1993

1,000,000 "end hosts"



Then in 1993 something really HUGE happened!!!

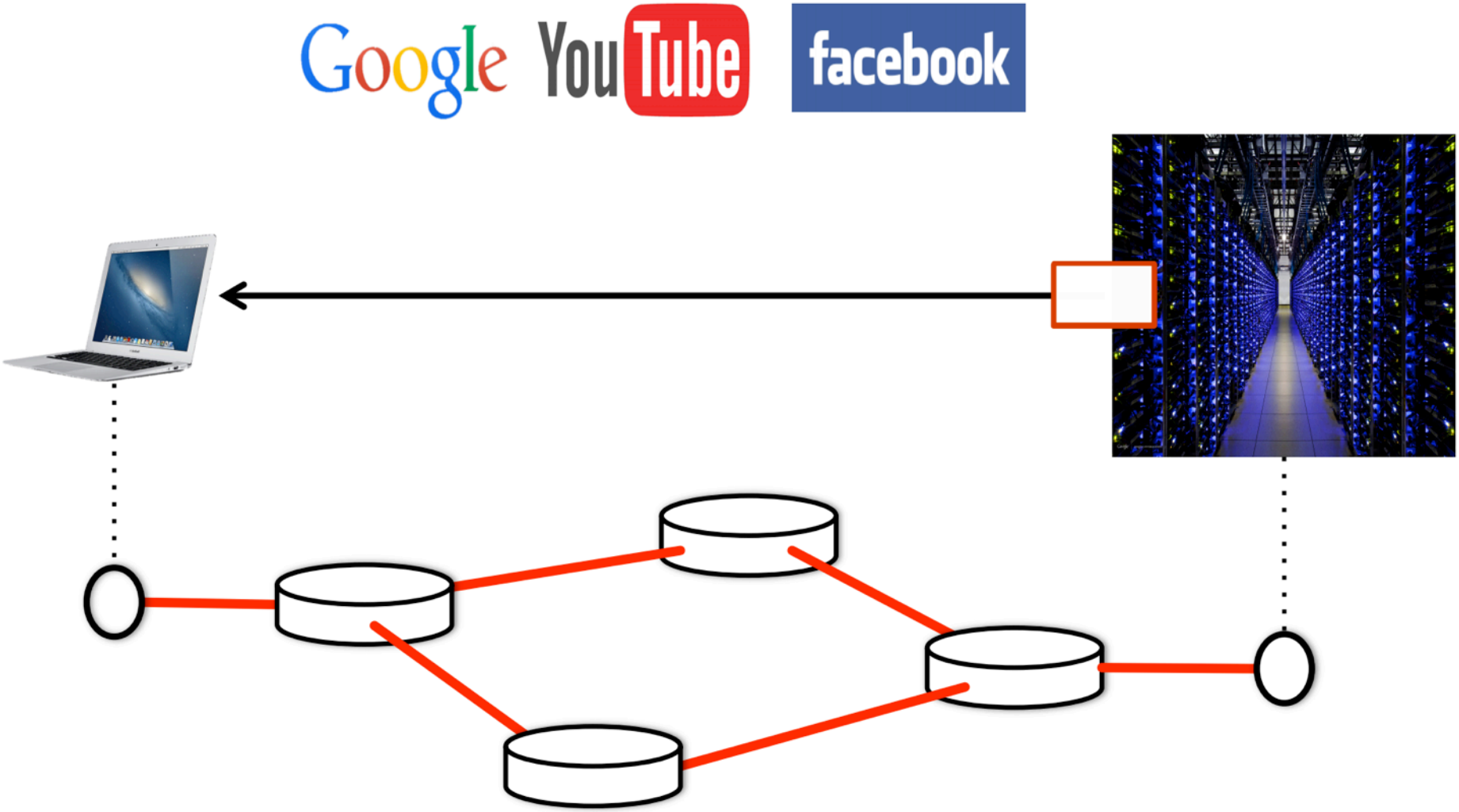
1993: The First Web Browser (Mosaic)

Marc Andreessen

1993

President
Bill Clinton

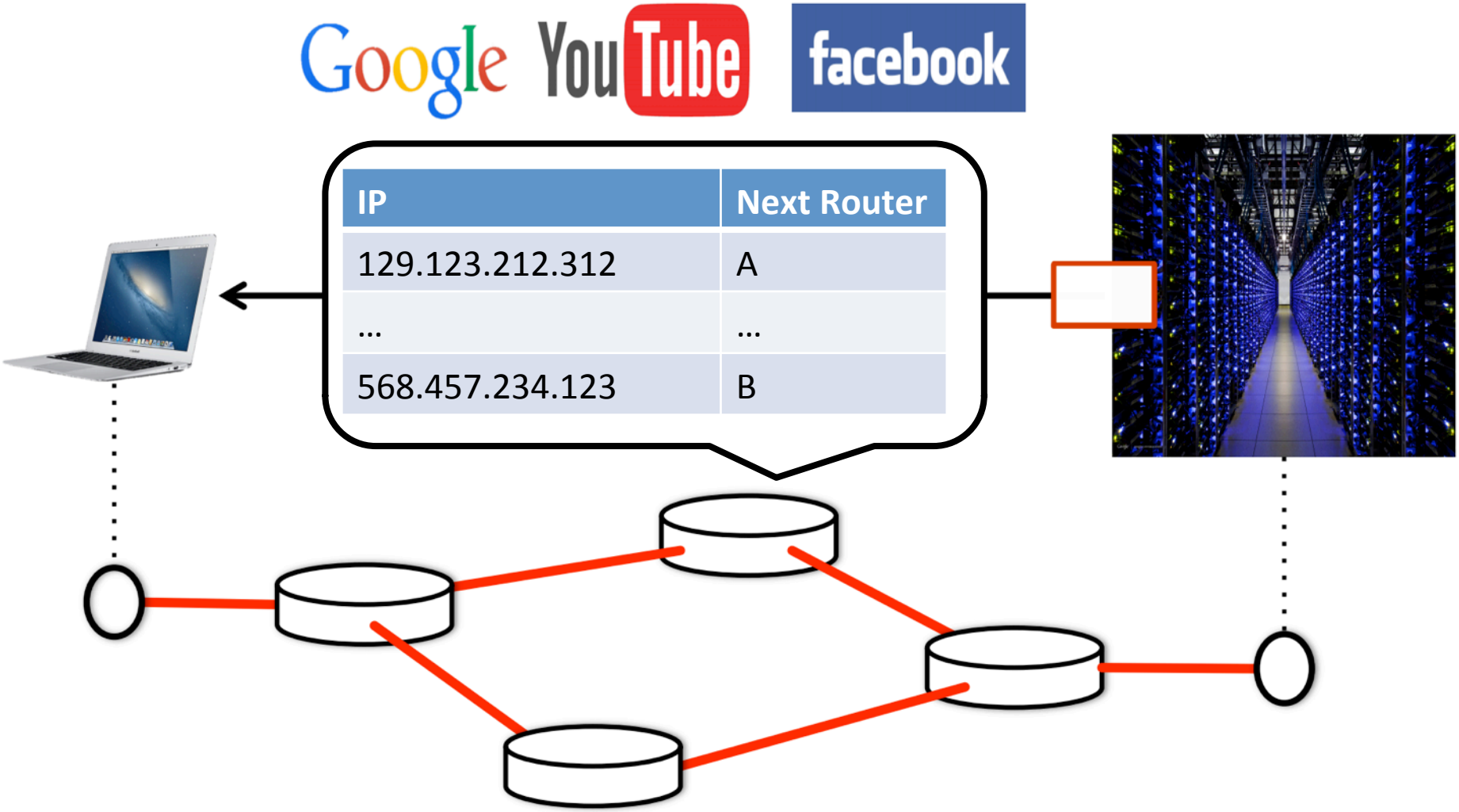
Modern Internet Graph



It's Fast

15,000km (120ms round-trip)
16 h 20 min
University Of Cambridge
Stanford University

Modern Internet Graph

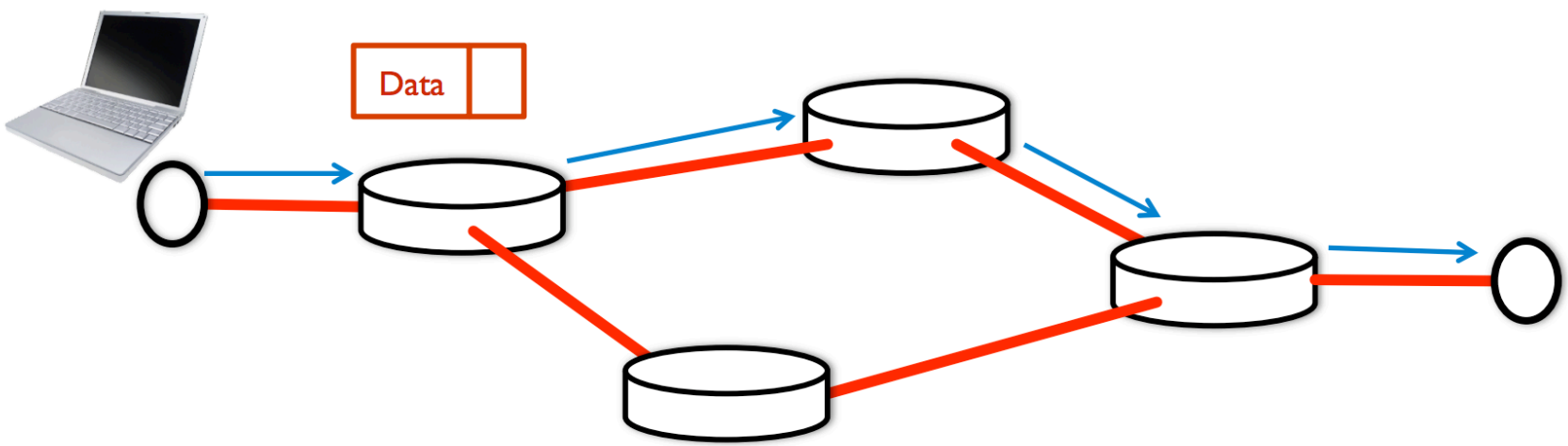


Each Router Runs Dijkstras

Can't be Trusted

The Internet doesn't promise to deliver packets in order.
It doesn't promise to deliver packets quickly, or on time.
It doesn't even promise to deliver them at all!

It just makes a "best-effort" attempt.



Today's Route

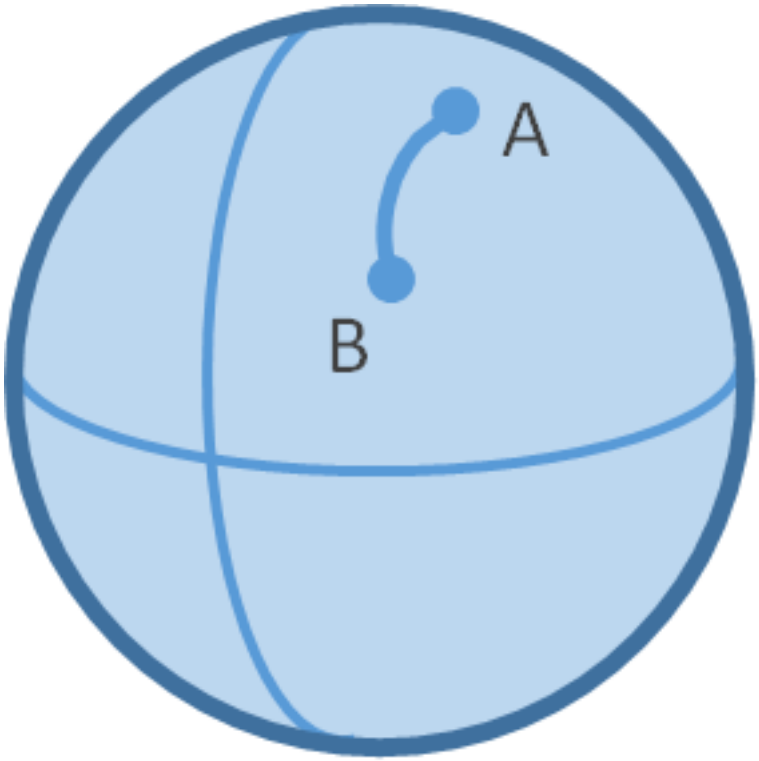
Today's Route

How Does Google Maps Work?

Can you think of a heuristic?

Direct Highway

$$\text{Heuristic} = \frac{\text{Distance on surface of earth}}{\text{Speed on fastest highway}}$$



Could Google Just Precompute?

How many nodes in google maps graph?

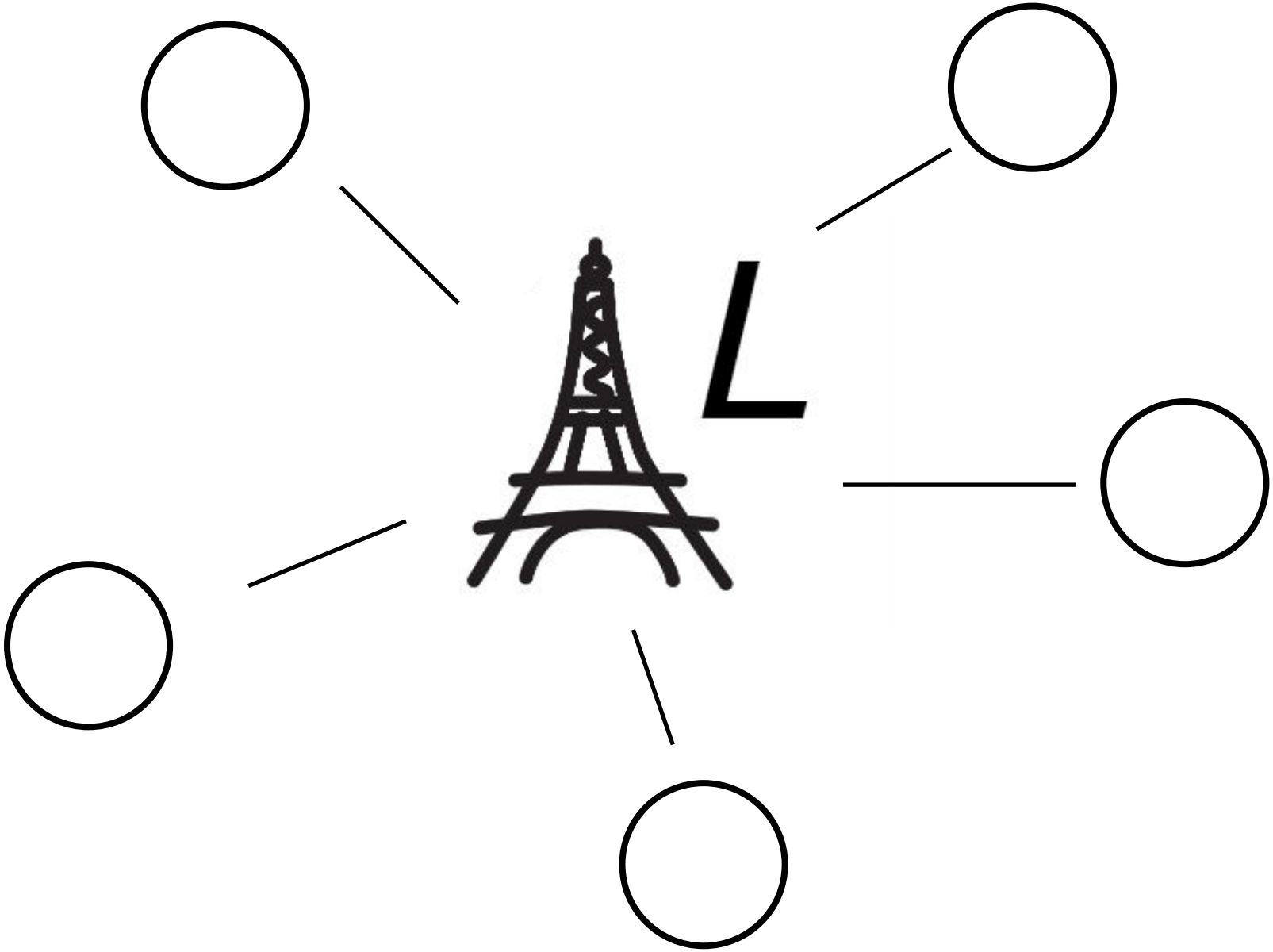
~ 75 million

n^2

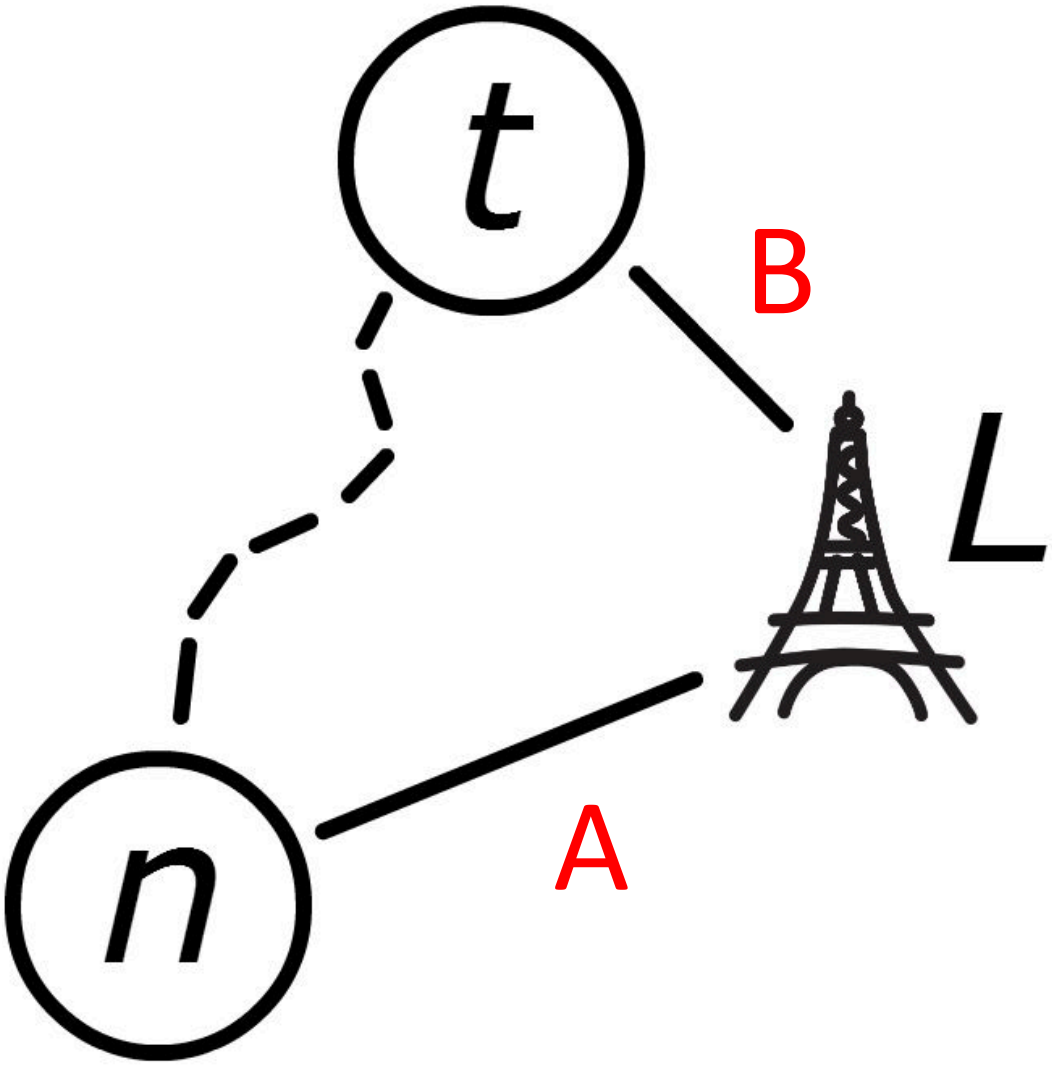
6×10^{15}

1 petasecond = 31.7 million years

Distance to Landmarks



Landmark Heuristic



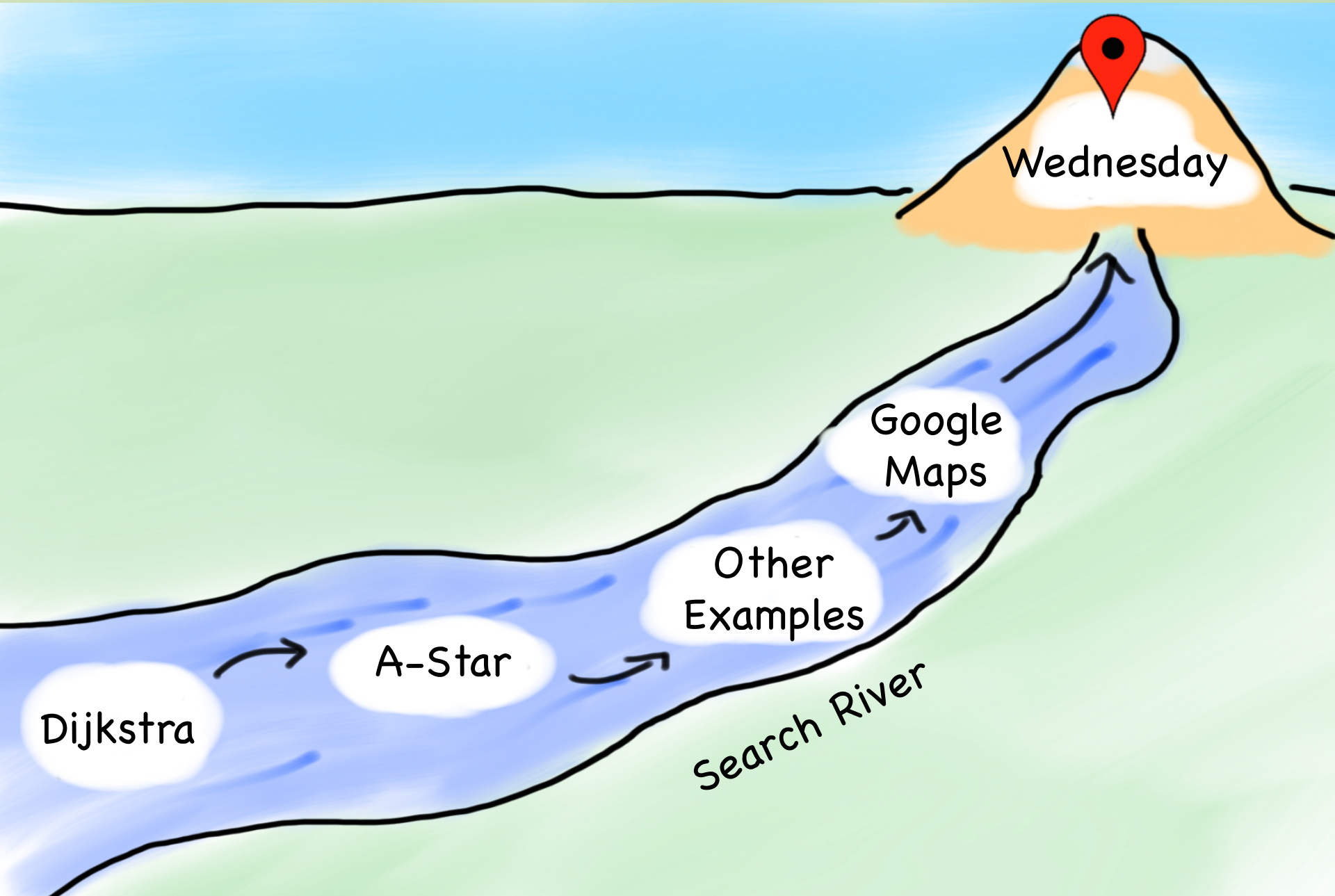
Distance < abs(A – B)

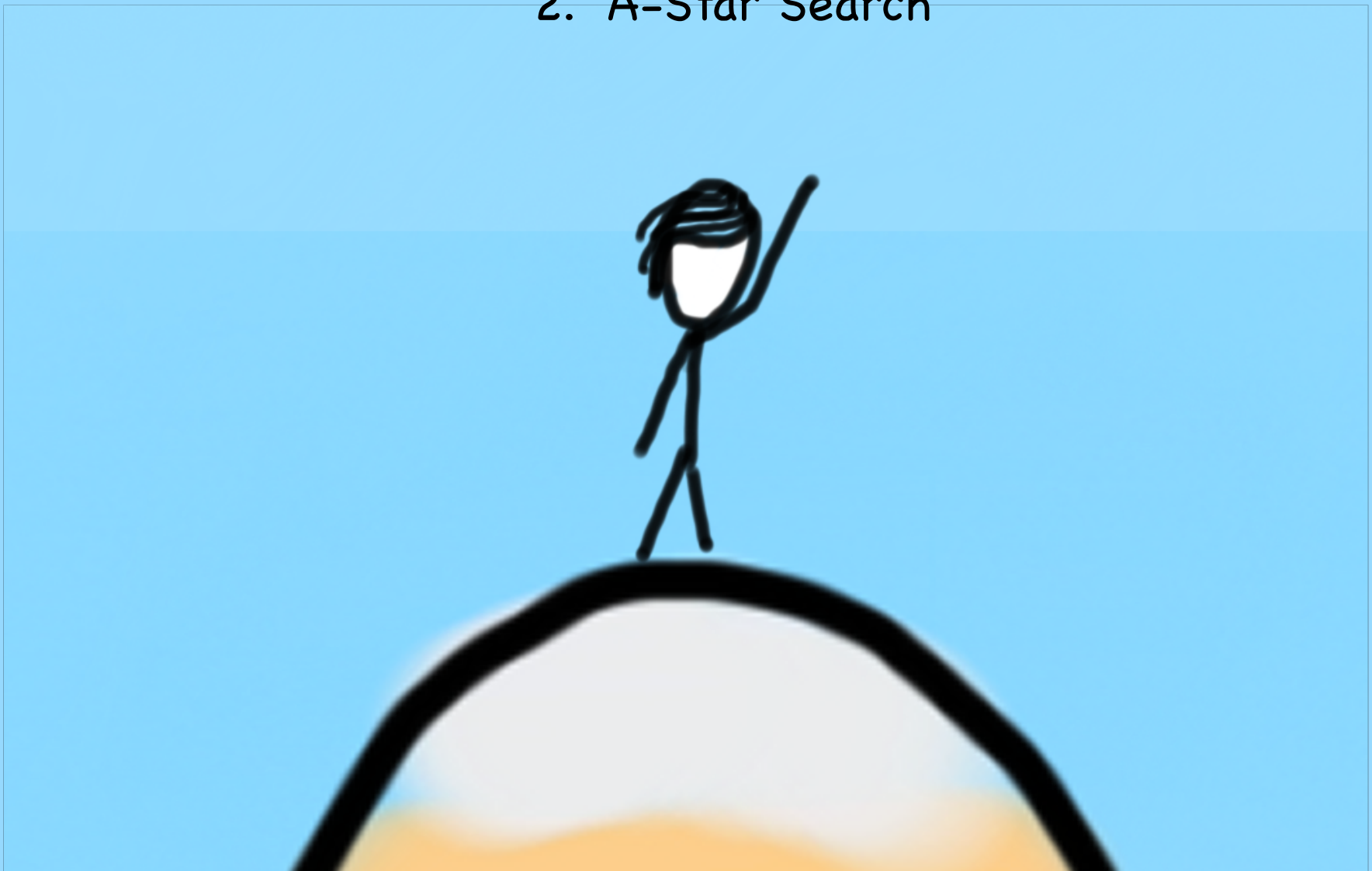
Best of All Heuristics

$$h = \max(h_1, h_2, ..., h_n)$$

Today's Route

Today's Route



Today's Goal

1. Dijkstra's Search
2. A-Star Search