Shortest Paths

Part Two

Recap from Last Time

Dijkstra's Algorithm

```
dijkstra's-algorithm() {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    if (that node isn't green) {
      color that node green.

      for (each neighboring node) {
        if (that node is not green) {
          color the node yellow.
          enqueue it at the new distance.
        }
      }
    }
  }
}
```

Use a priority queue rather than a standard queue to sort by distances, not number of hops.

Dijkstra's Algorithm

```
dijkstra's-algorithm() {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    if (that node isn't green) {
      color that node green.

      for (each neighboring node) {
        if (that node is not green) {
          color the node yellow.
          enqueue it at the new distance.
        }
      }
    }
  }
}
```

Allow nodes to be enqueued multiple times. The first time we find the node might not be the best option.

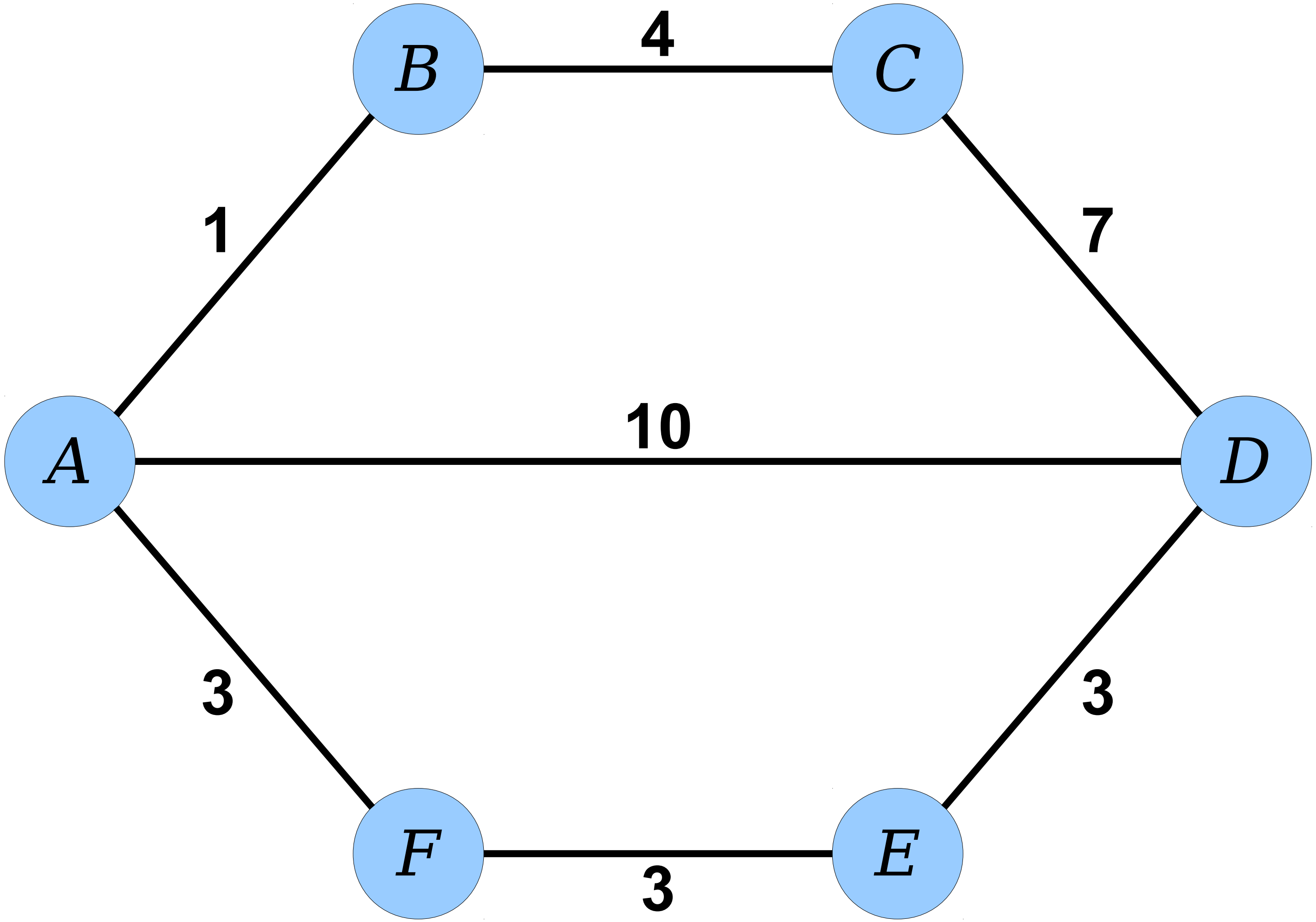
Dijkstra’s Algorithm

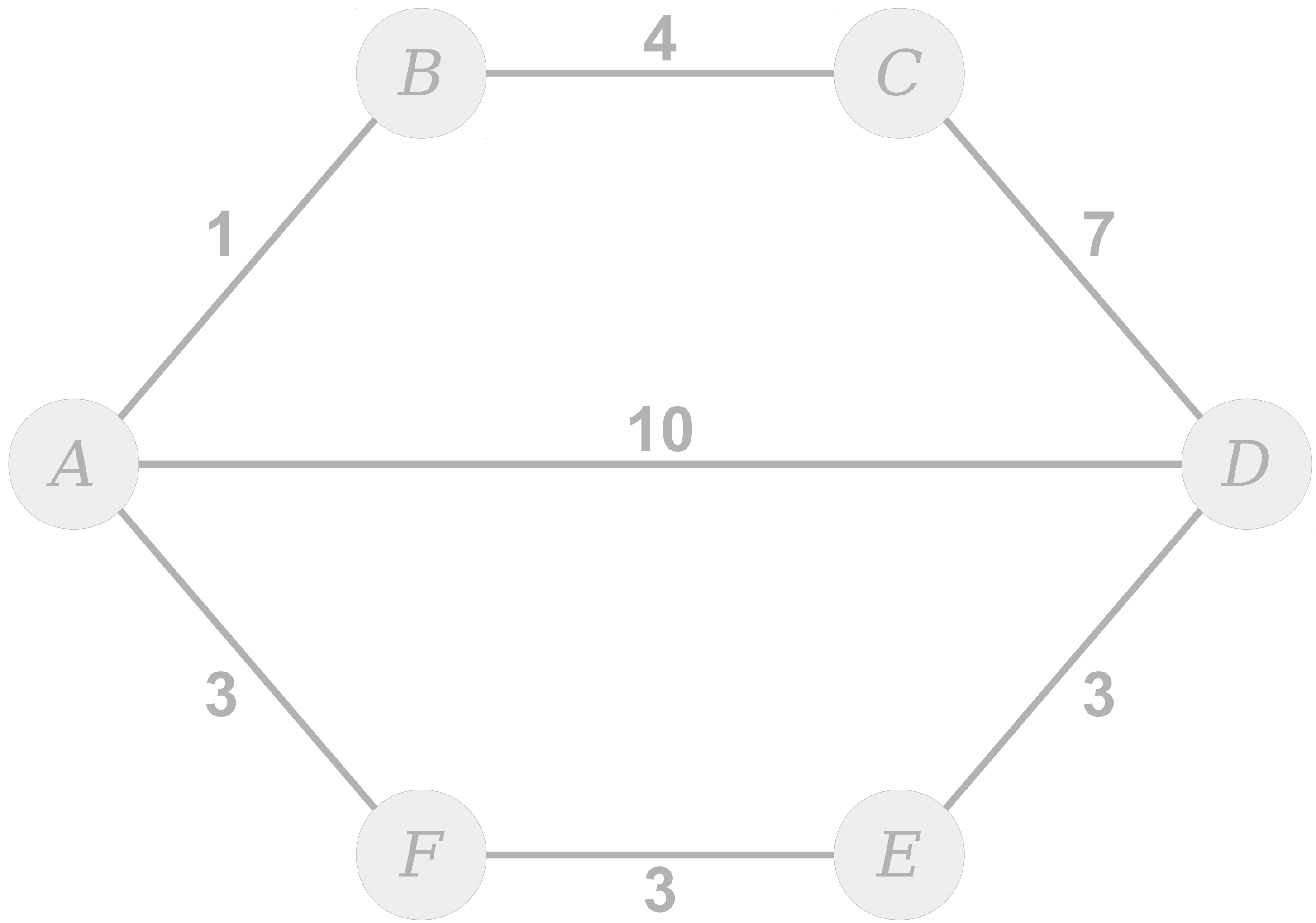
```
dijkstra's-algorithm() {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    if (that node isn't green) {
      color that node green.

      for (each neighboring node) {
        if (that node is not green) {
          color the node yellow.
          enqueue it at the new distance.
        }
      }
    }
  }
}
```

As a consequence, when dequeuing nodes, make sure we’re not visiting something we’ve already processed.

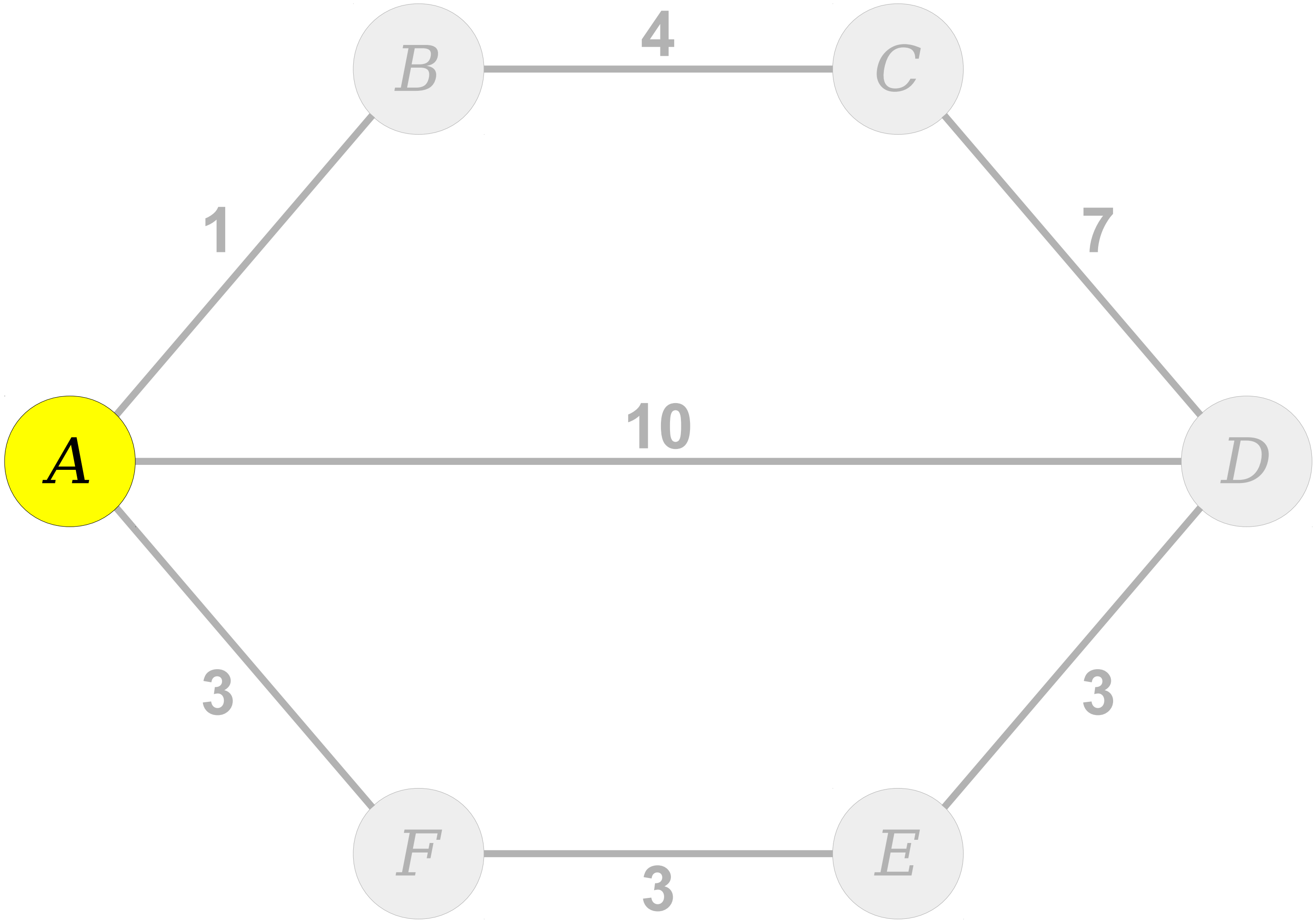
New Stuff!

Some Practical Concerns

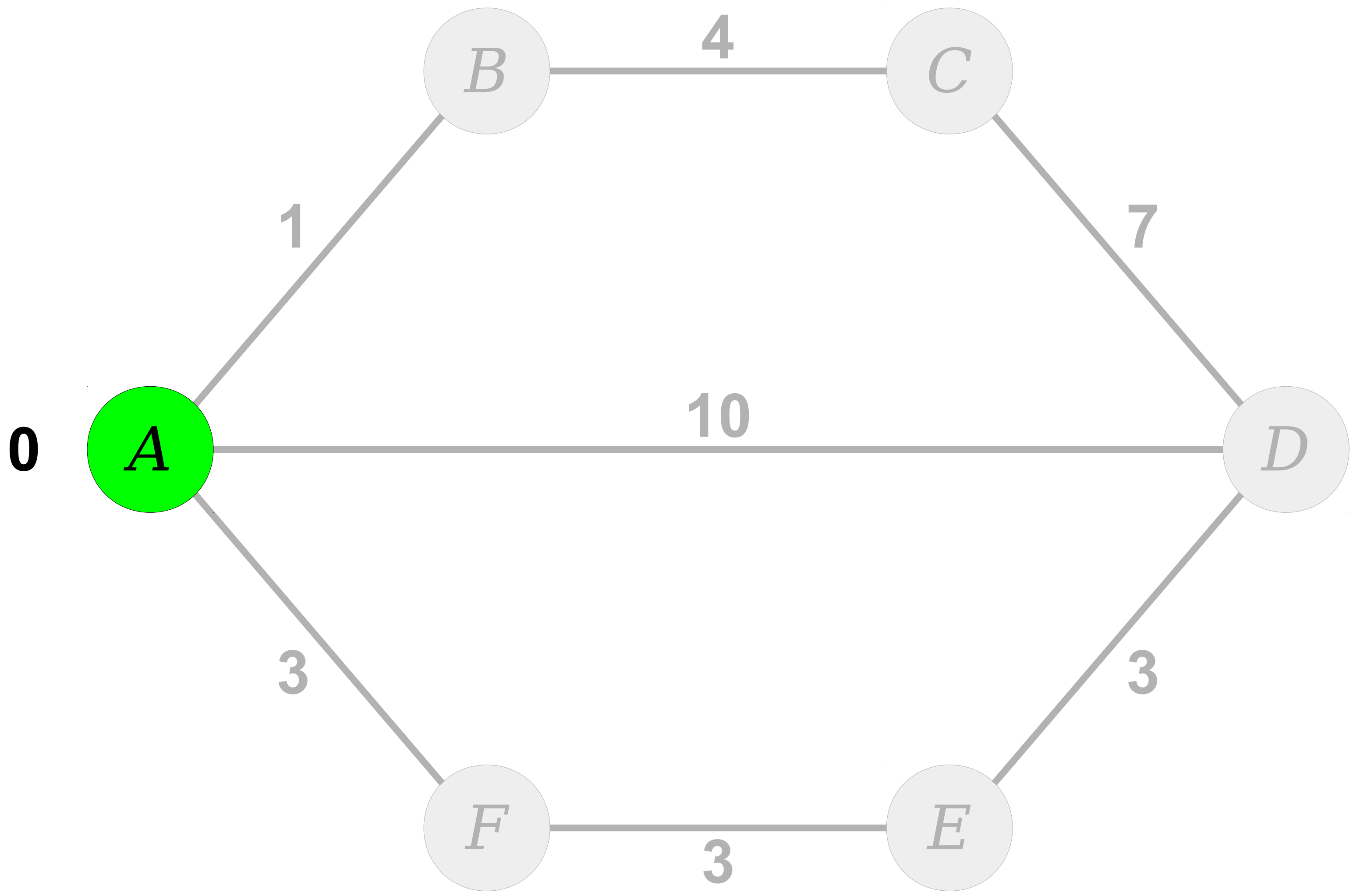

B
4
C
1
7
A
10
D
3
3
F
3
E

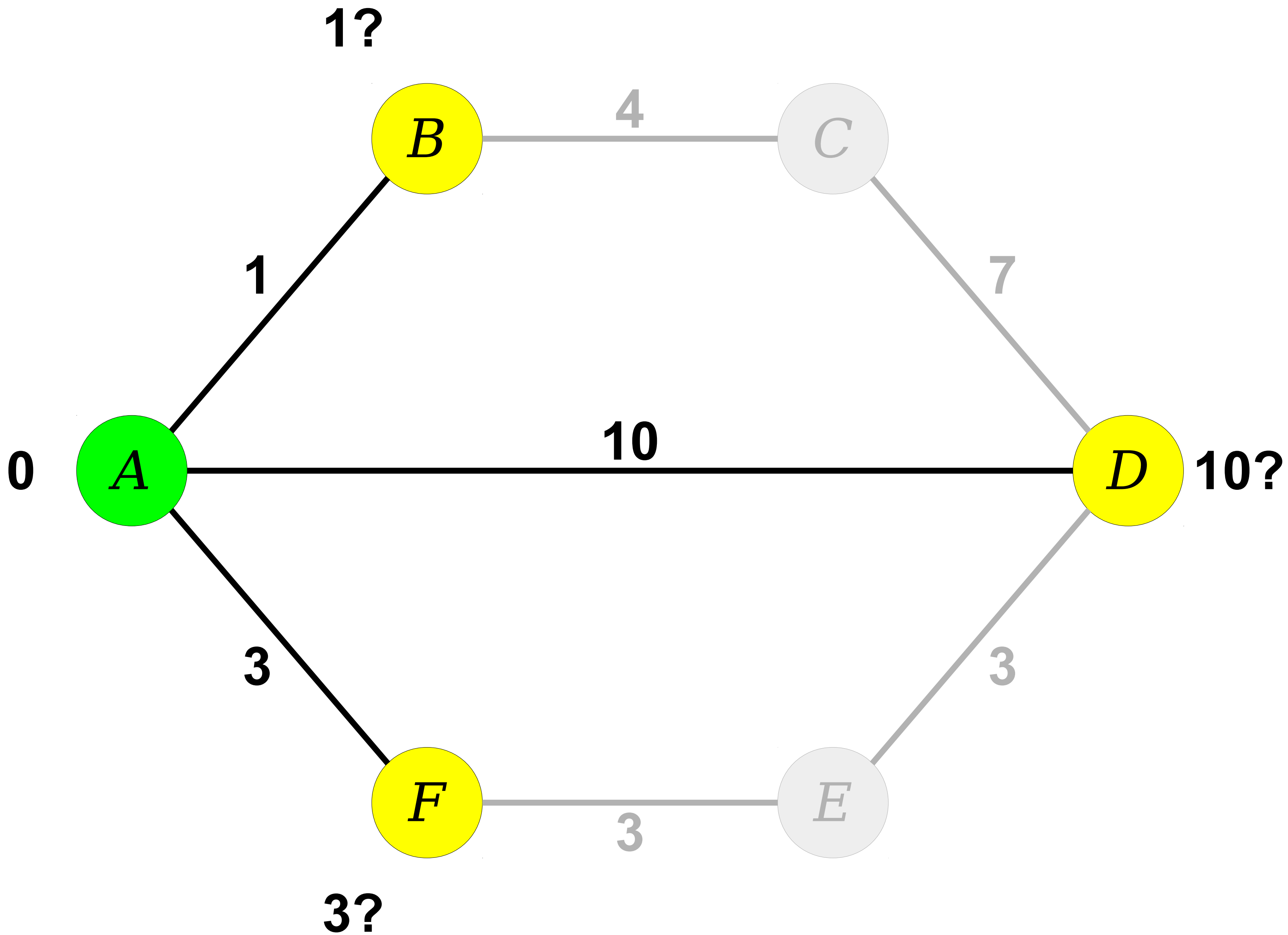


A
B
C
D
E
F
4
1
7
10
3
3
3

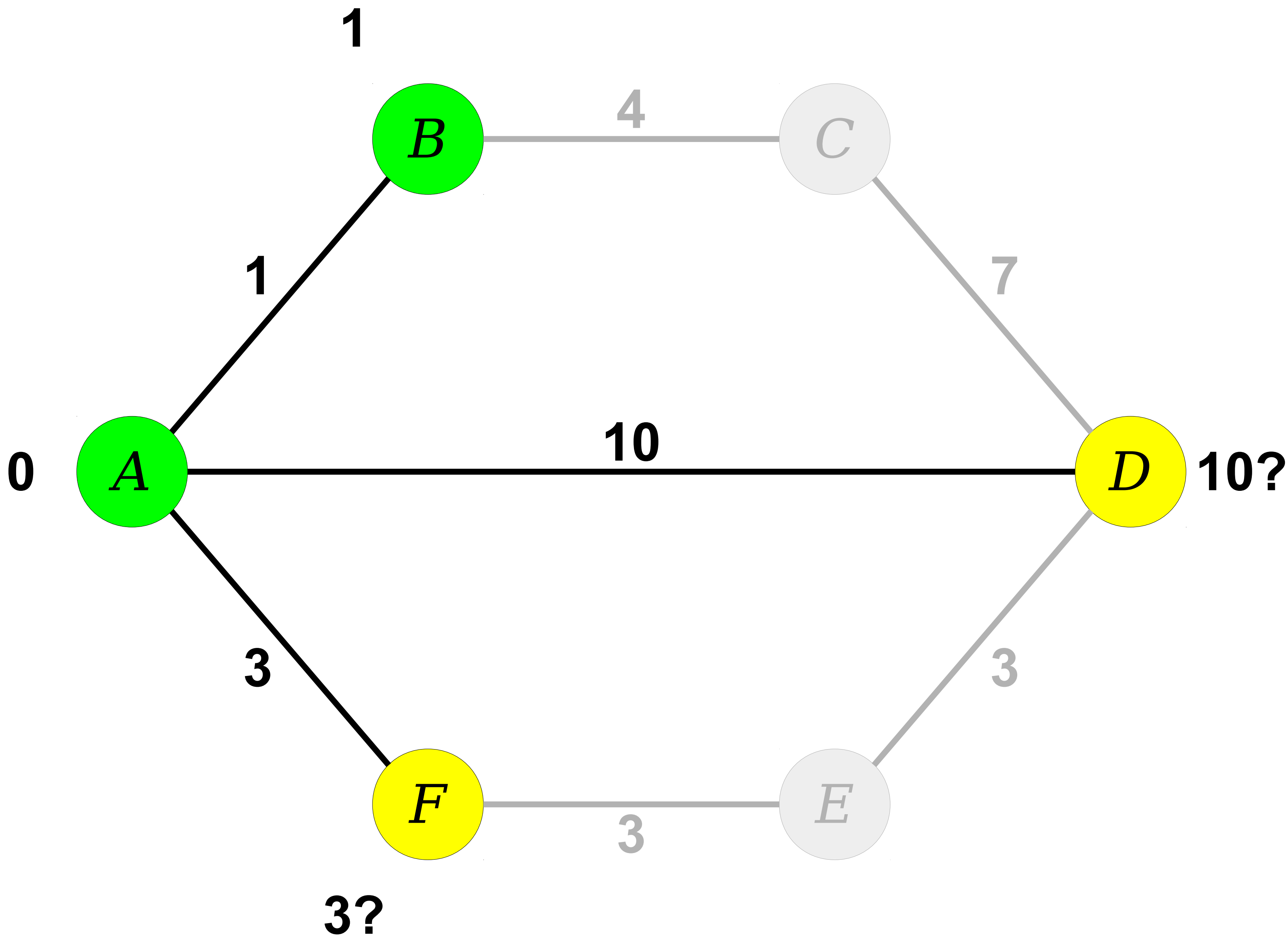


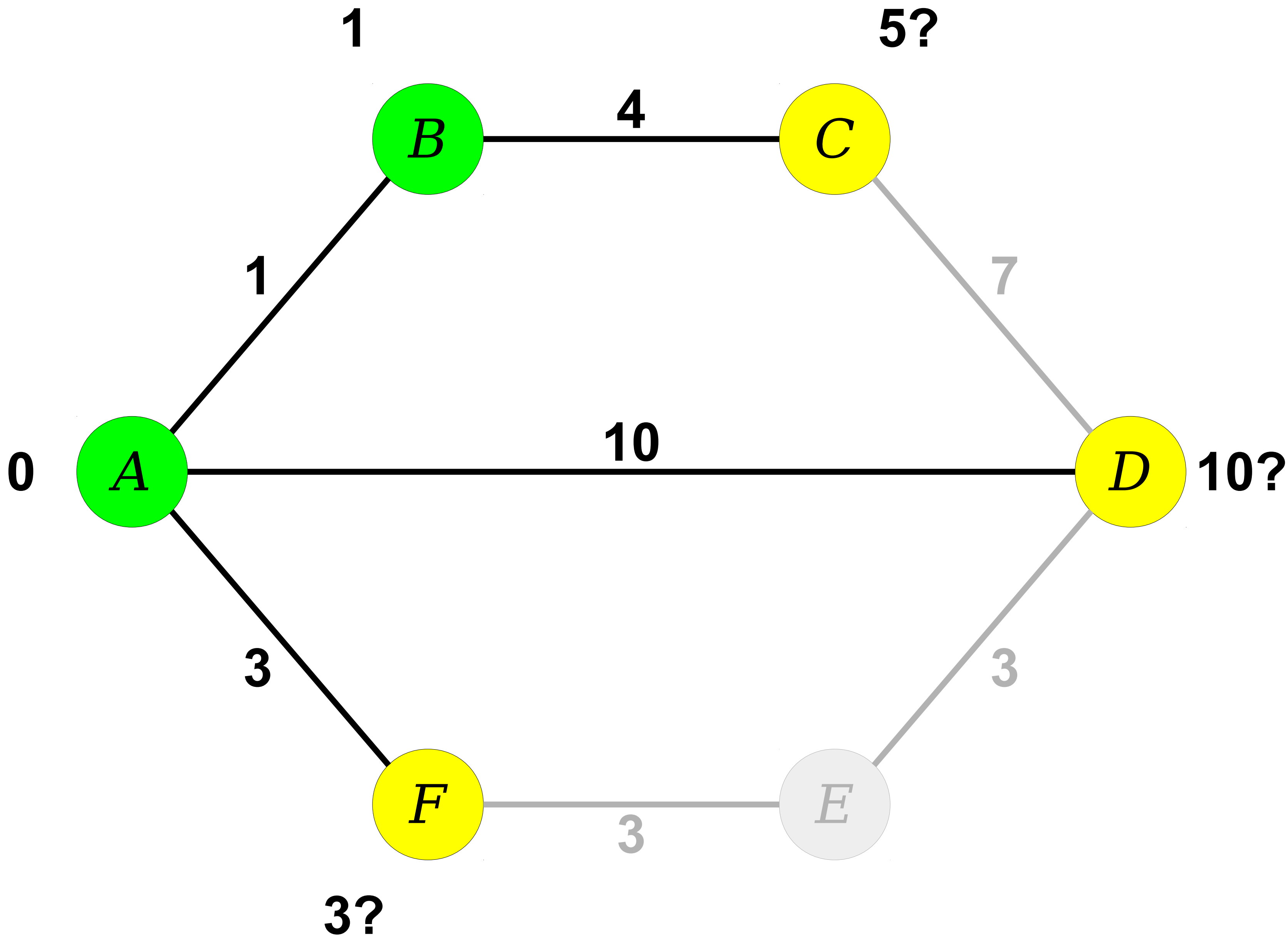
A
B
C
D
E
F
1
4
7
10
3
3
3

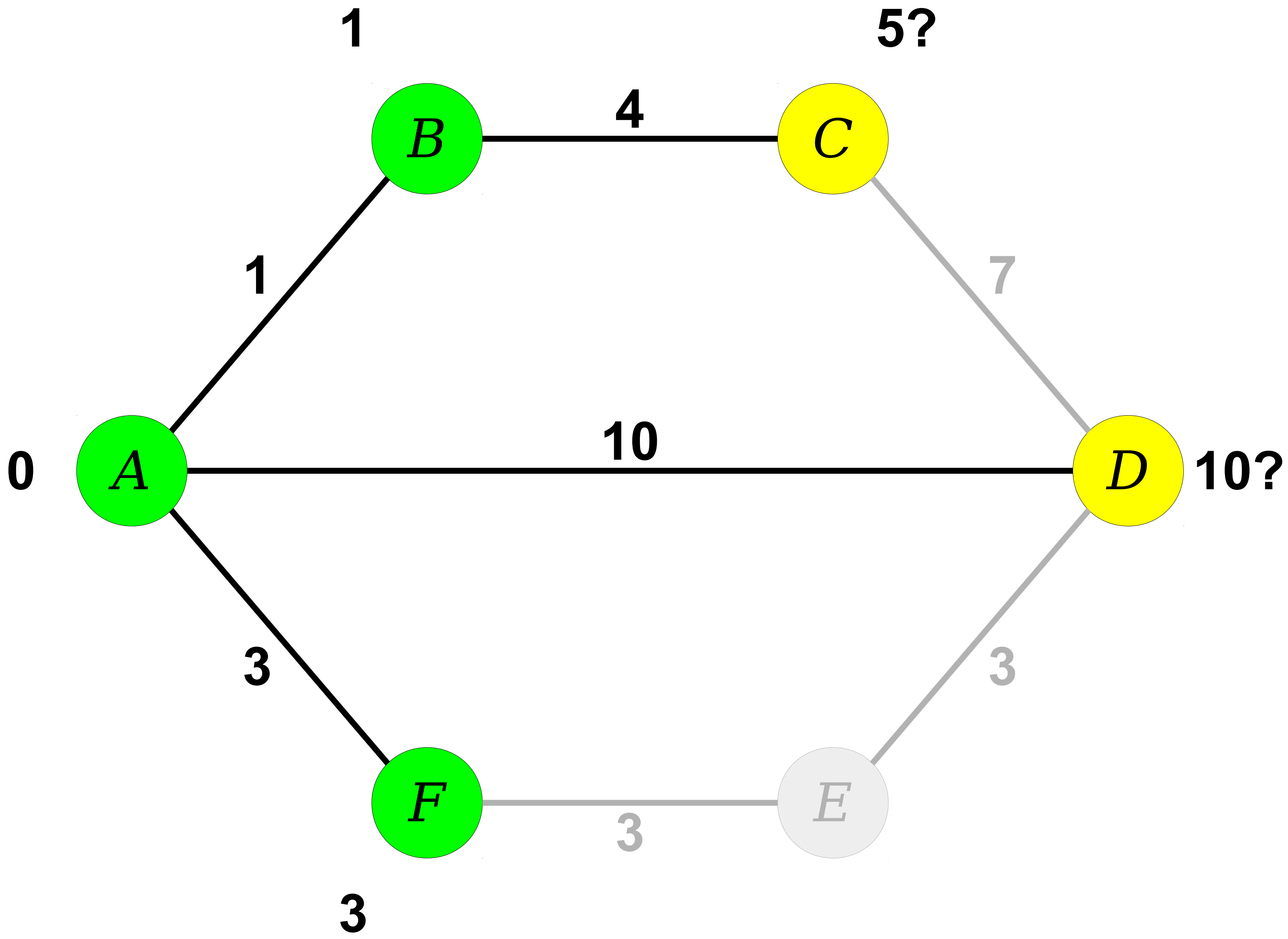

0
A
B
C
D
E
F
1
4
7
10
3
3
3

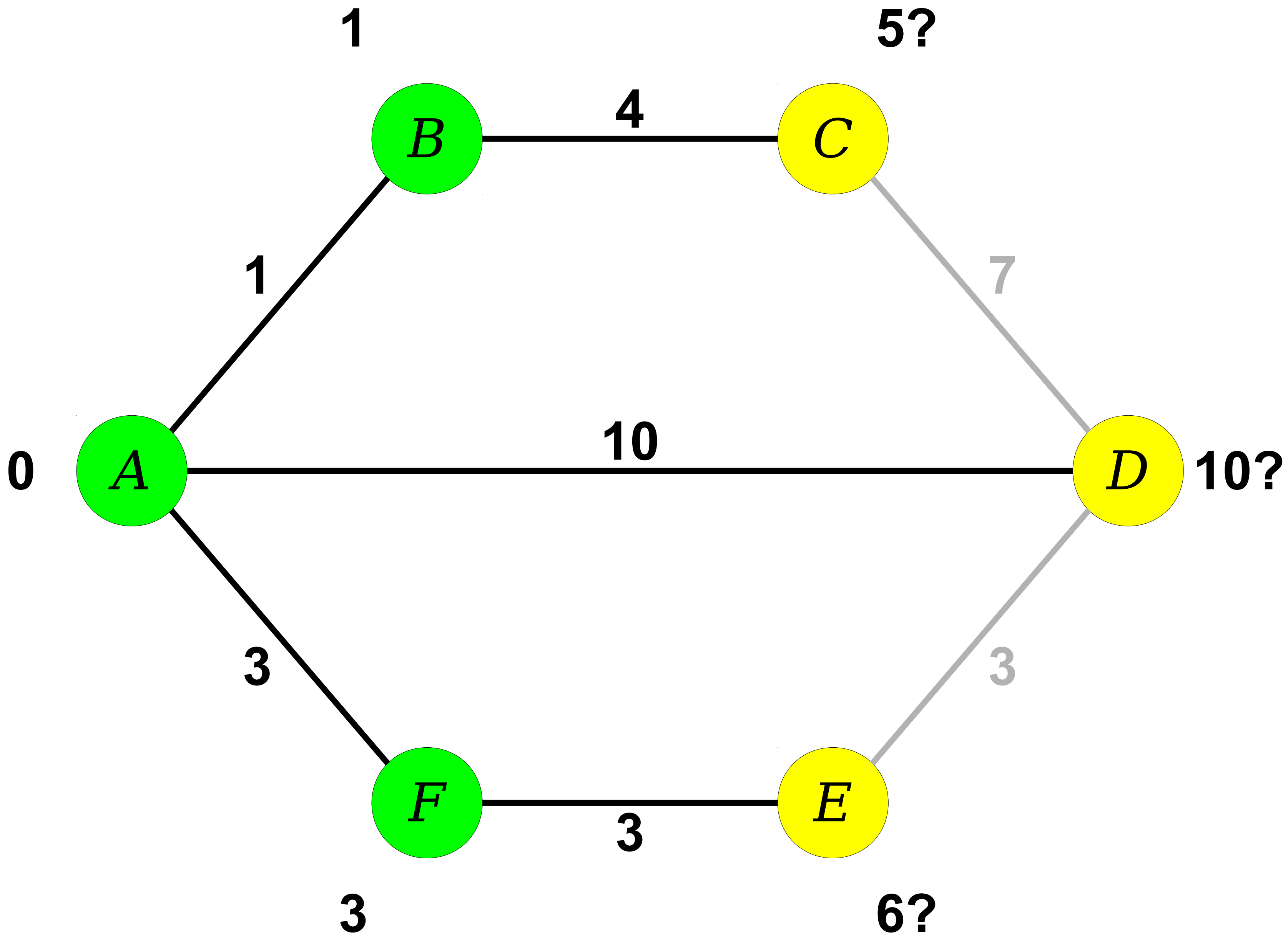


1?
B
4
C
1
7
0
A
10
D
10?
3
3
F
3
E
3?

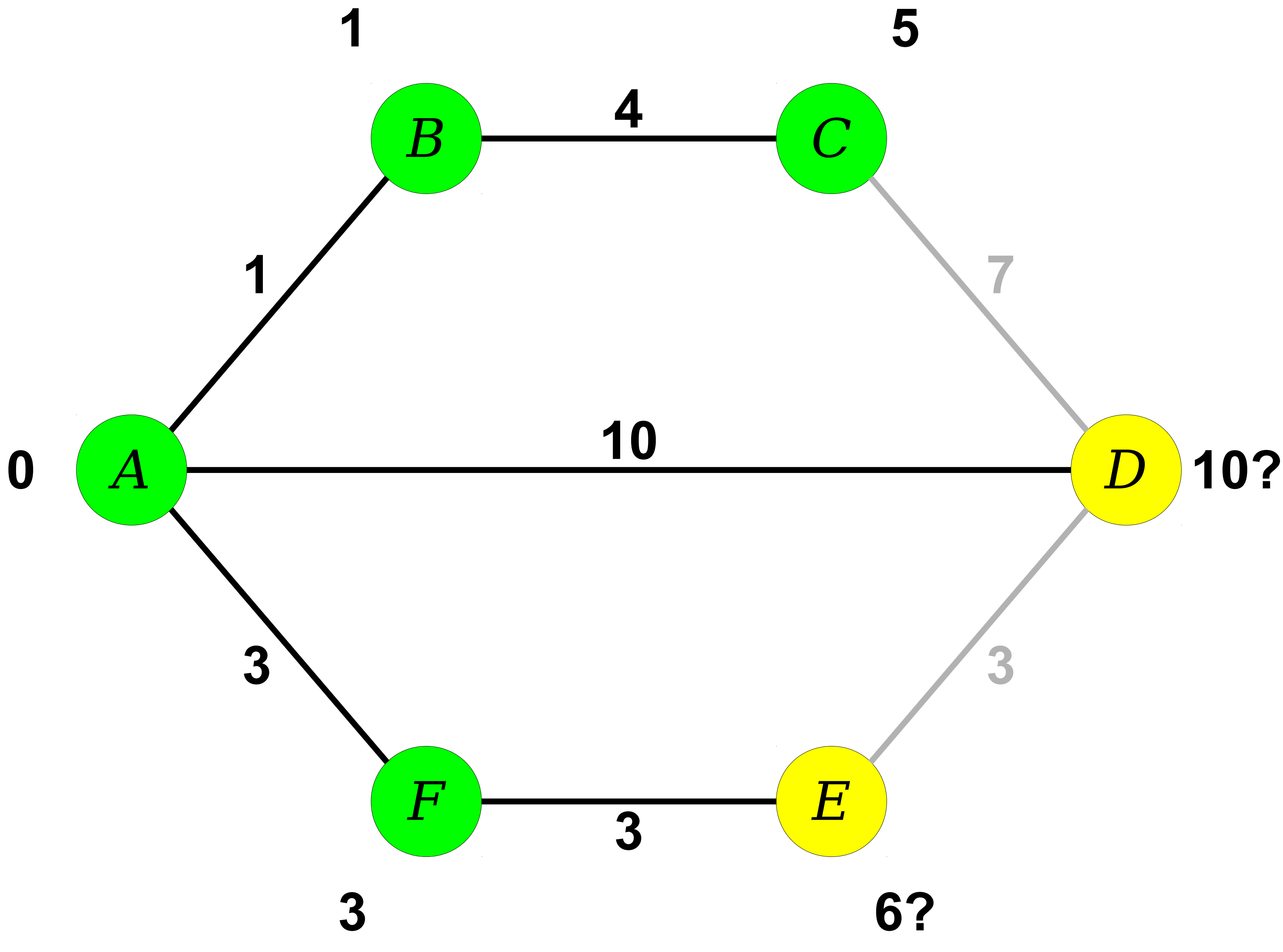

1
B
4
C
1
7
0
A
10
D
10?
3
3
F
3
E
3?

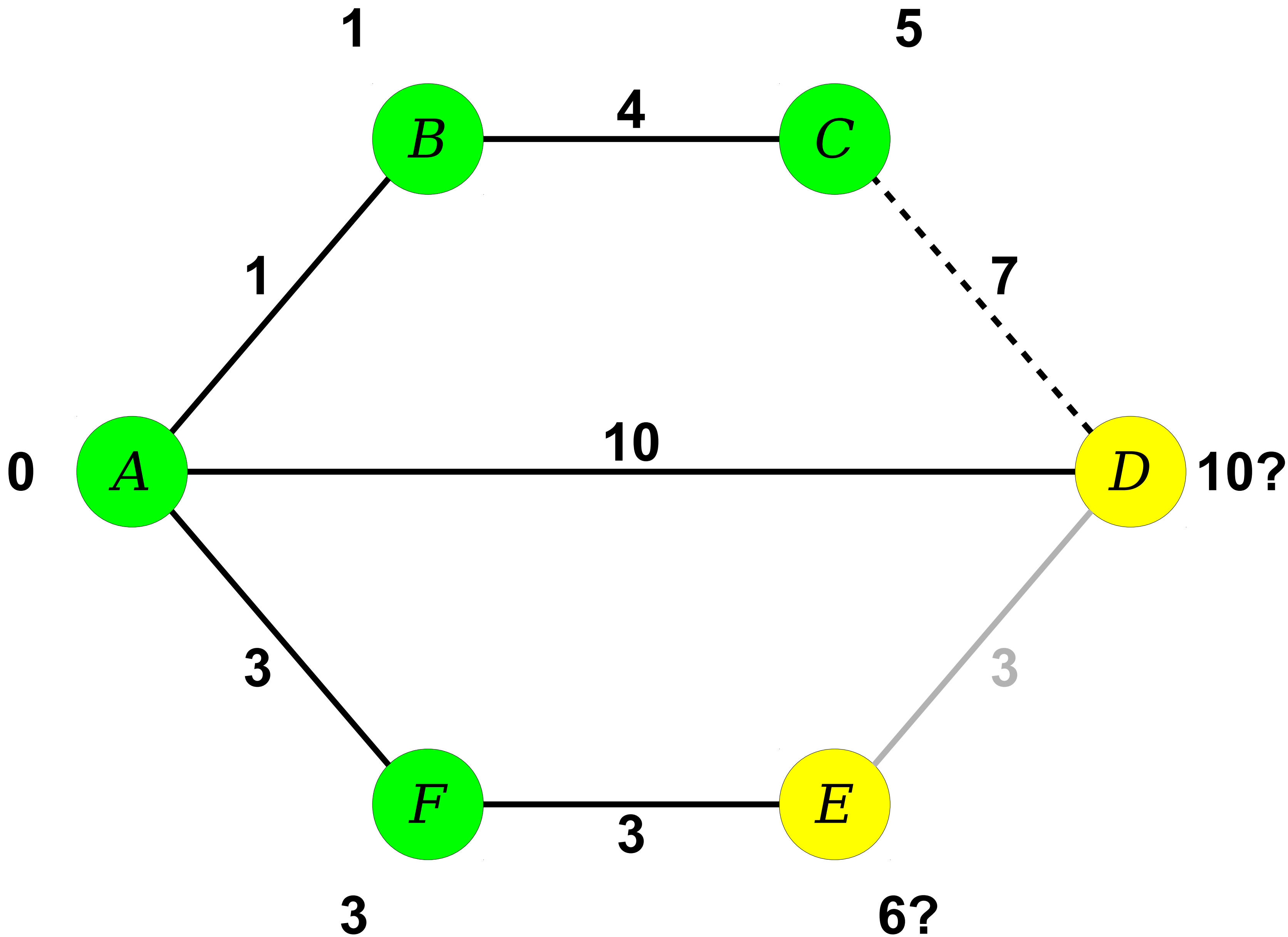

1
5?
4
B
C
1
7
0
A
10
D
10?
3
3
F
E
3
3?


1
5?
4
B
C
1
7
0
A
10
D
10?
3
3
F
E
3
3

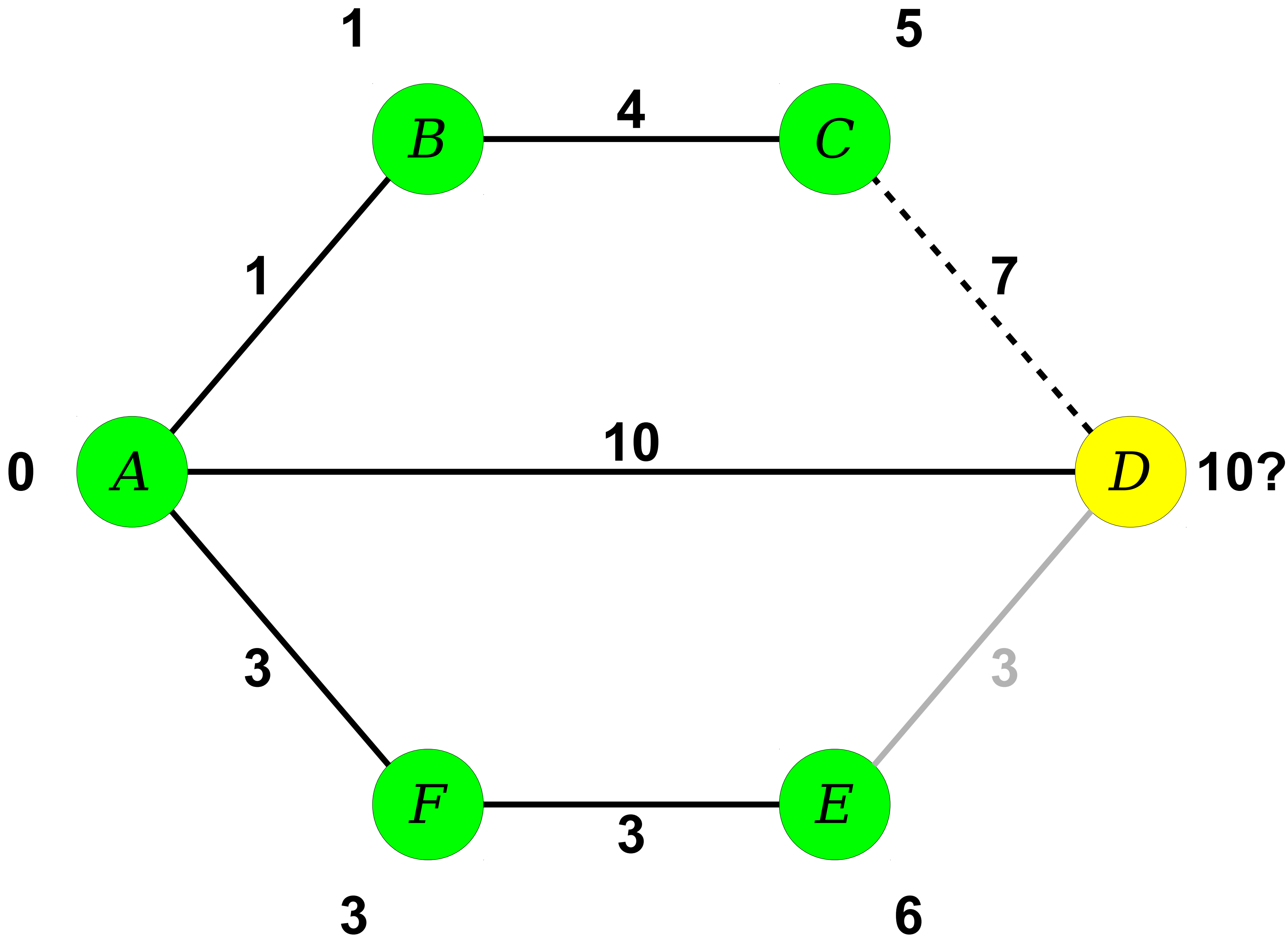


1
5?
4
B
C
1
7
0
A
10
D
10?
3
3
F
E
3
3
6?

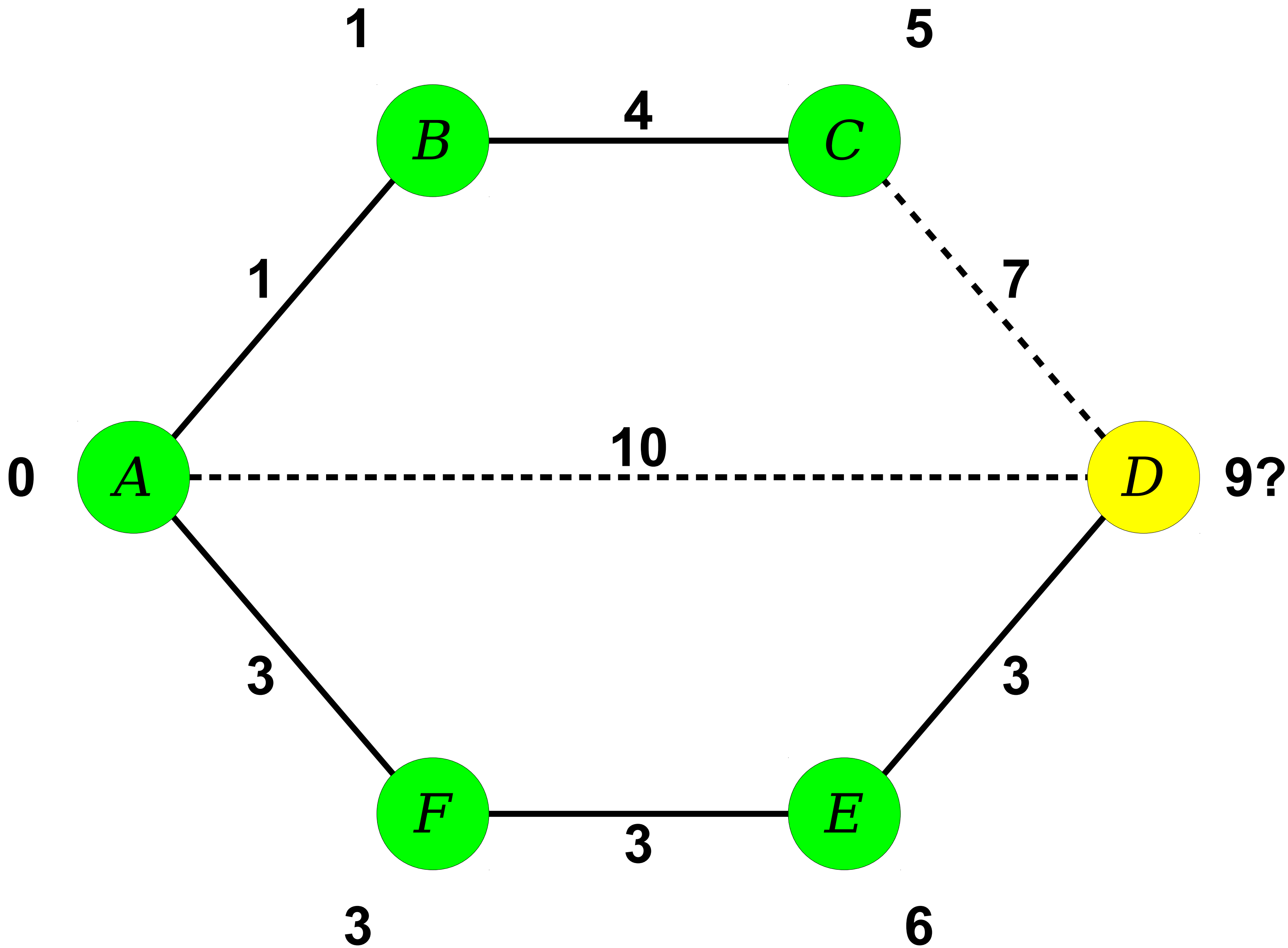

1
5
B
4
C
1
7
0
A
10
D
10?
3
3
F
3
E
3
6?

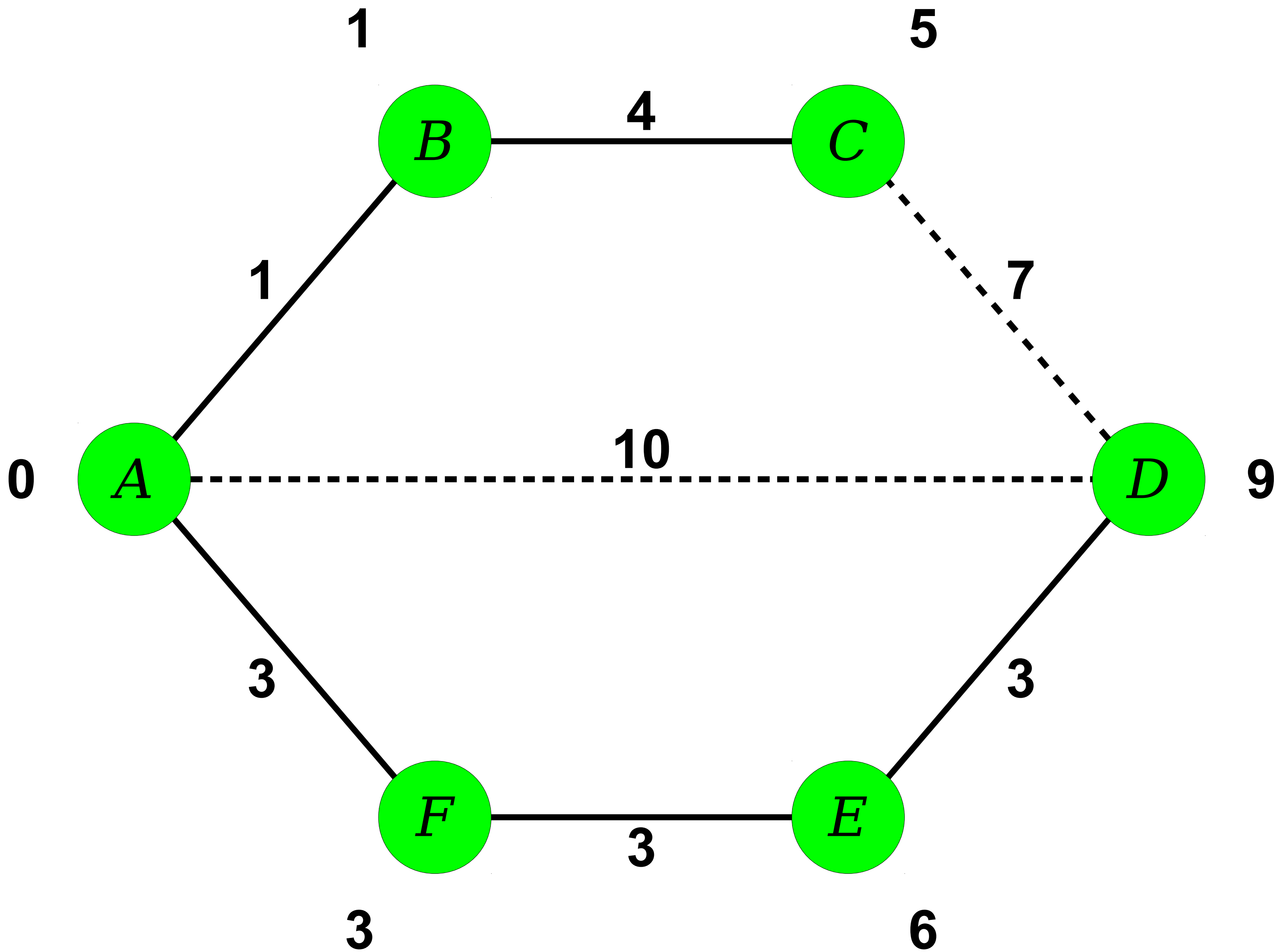


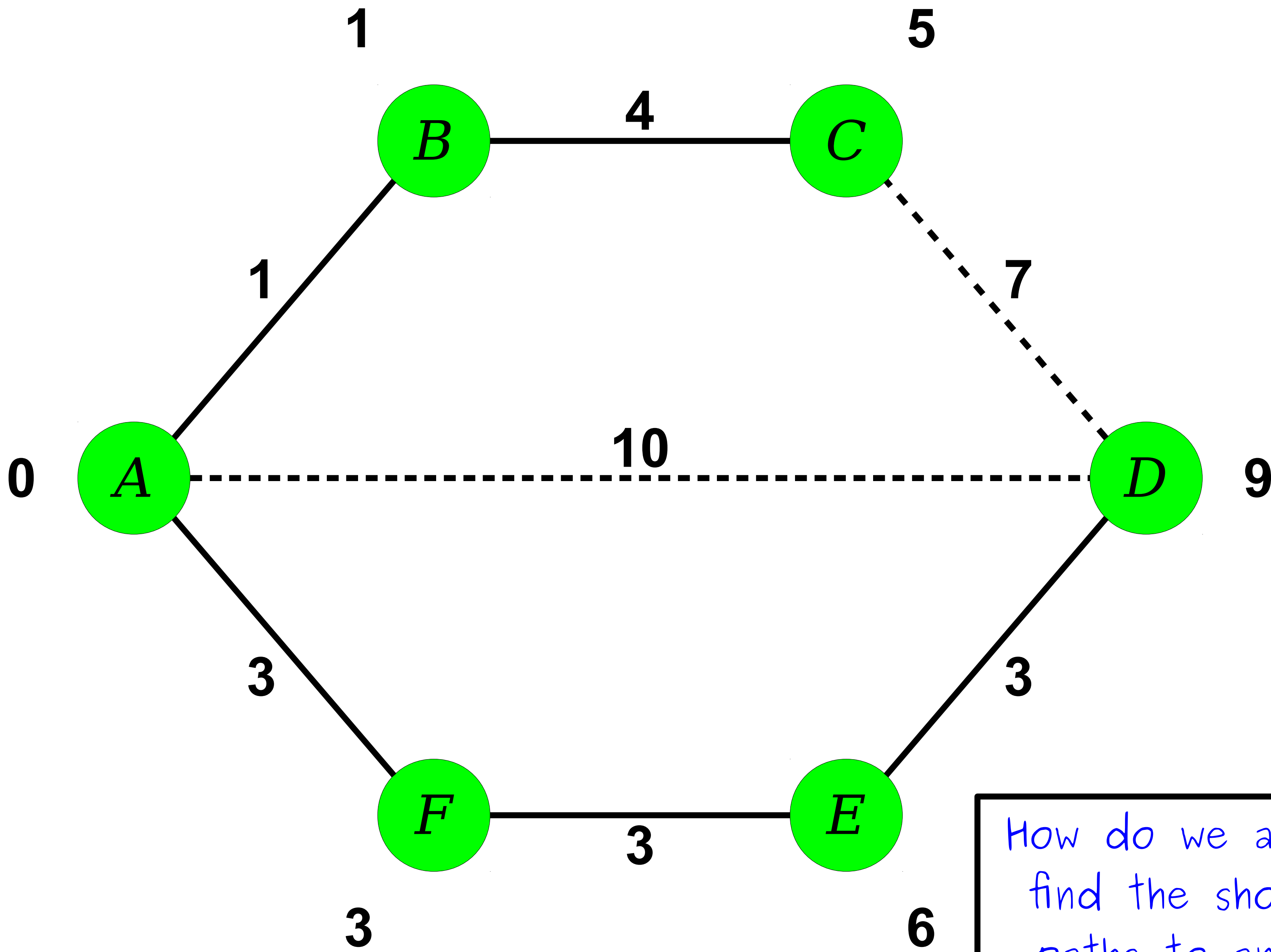
1
5
B
4
C
1
7
0
A
10
D
10?
3
3
F
3
E
3
6?



1
5
B
4
C
1
7
0
A
10
D
10?
3
3
F
3
E
3
6


1
5
B
4
C
1
7
0
A
10
D
9?
3
3
F
3
E
3
6


1
5
B
4
C
1
7
0
A
10
D
9
3
3
F
3
E
3
6



How do we actually find the shortest paths to any of these nodes?

Option 1: Explicitly Store Paths

(Easier, less efficient)

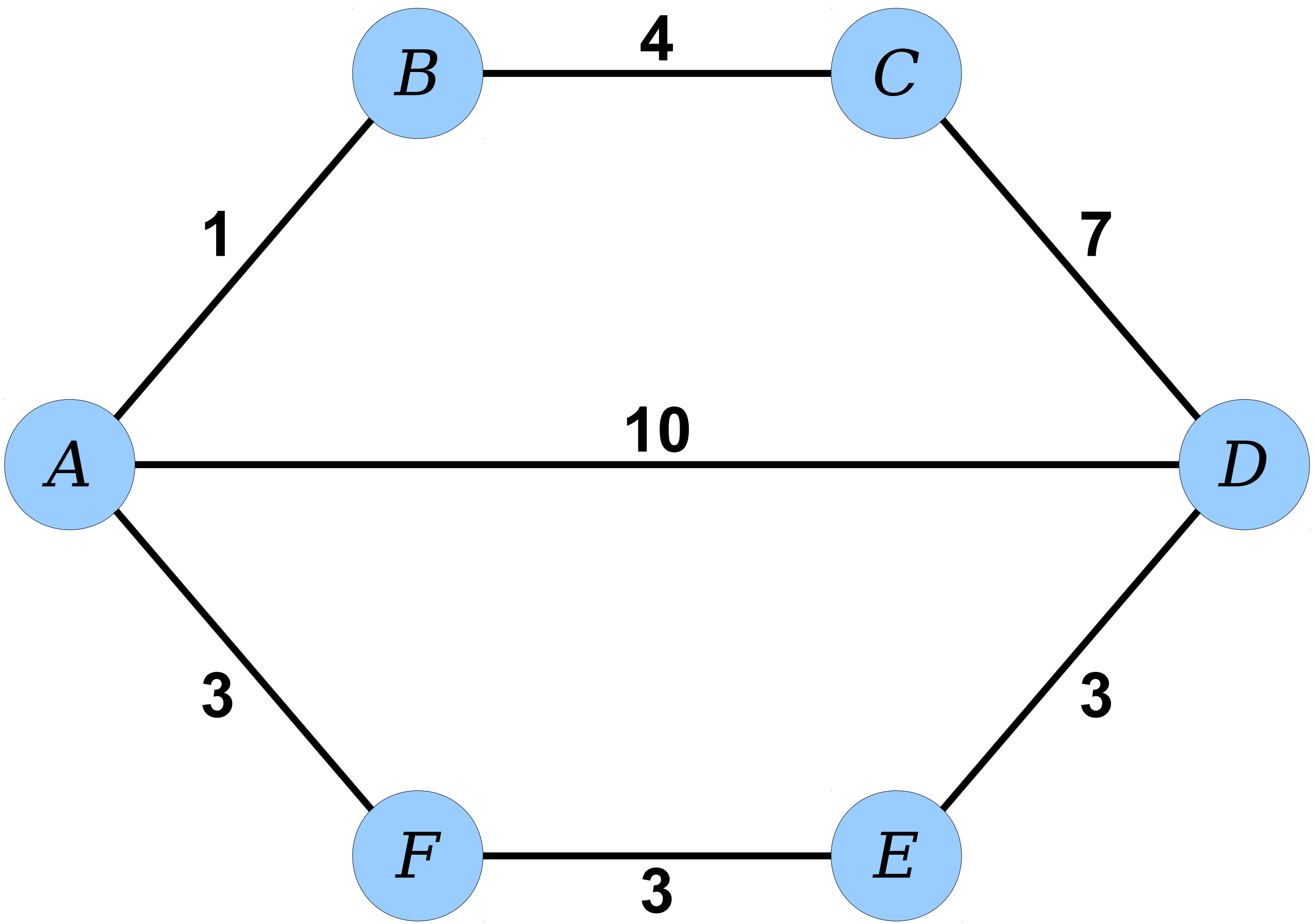
```
dijkstra's-algorithm() {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.

    if (that node isn't green) {
      color that node green.

      for (each neighboring node) {
        if (that node is not green) {
          color the node yellow.

          enqueue it at the new distance.
        }
      }
    }
  }
}
```

```
dijkstra's-algorithm() {
  make a priority queue of paths.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a path from the queue.
    look at the last node on that path.
    if (that node isn't green) {
      color that node green.

      for (each neighboring node) {
        if (that node is not green) {
          color the node yellow.
          extend the current path with that node.
          enqueue it at the new distance.
        }
      }
    }
  }
}
```

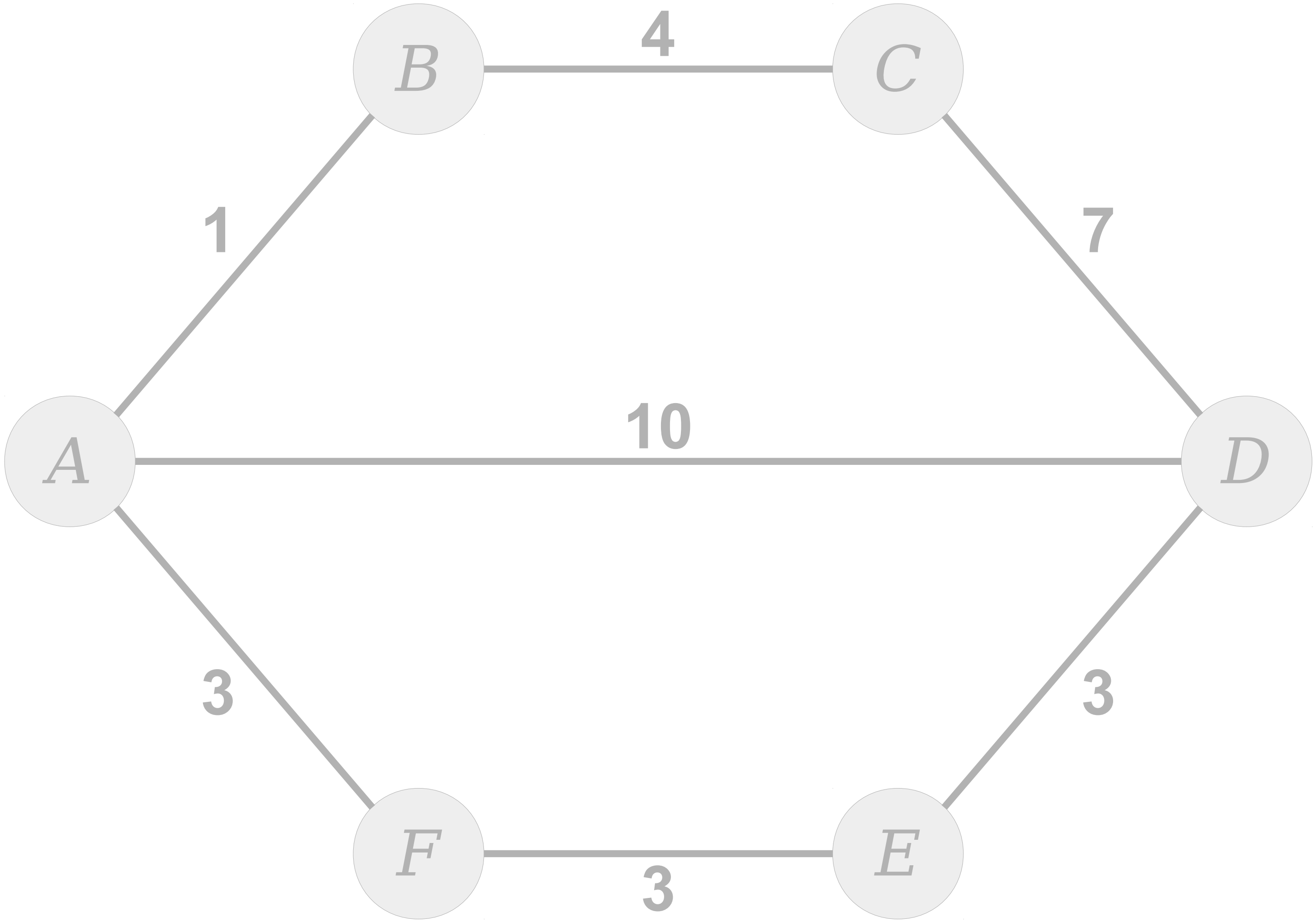
Just like in Word Ladders!

Option 2: Store Parent Pointers

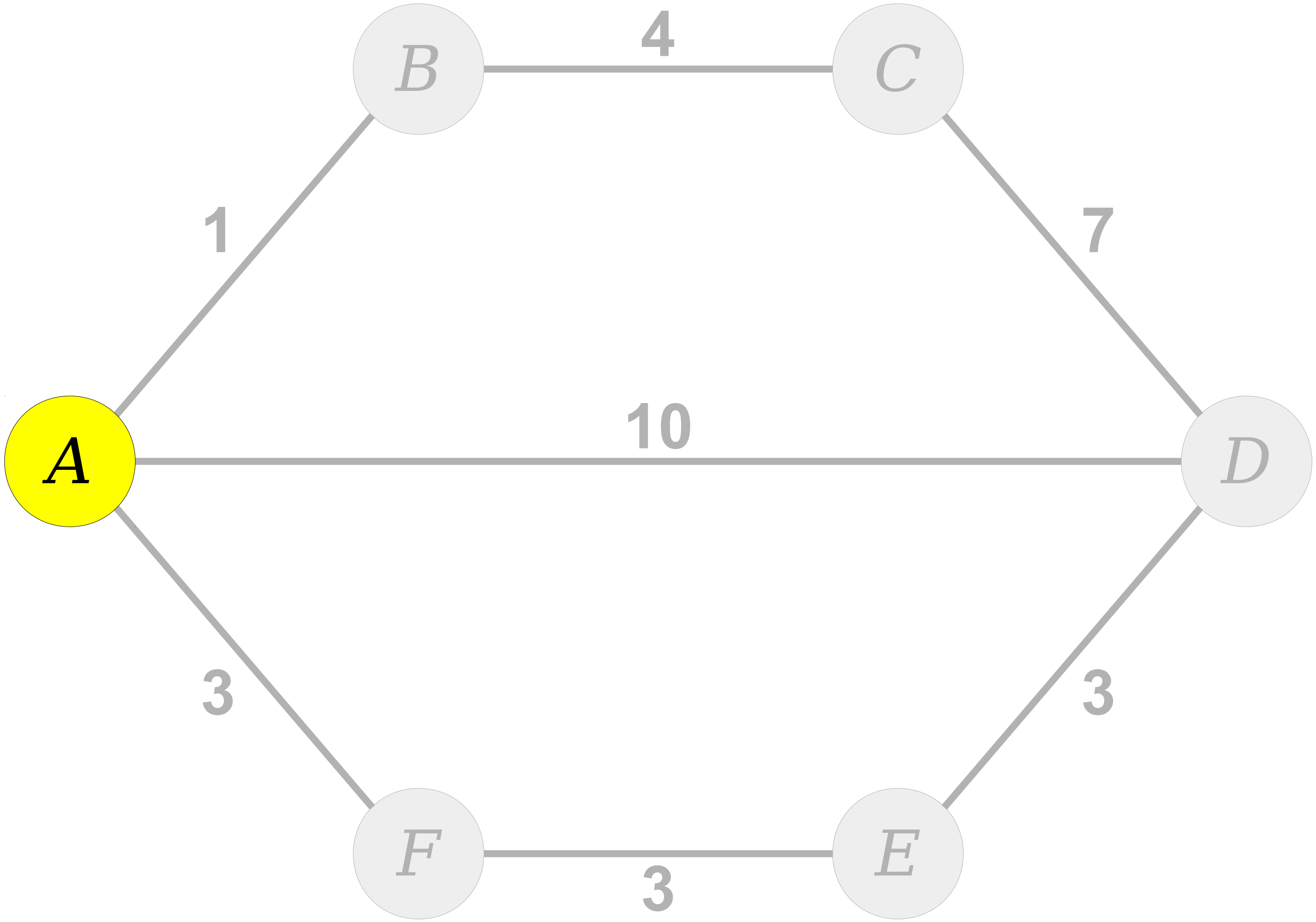
(Harder, faster)



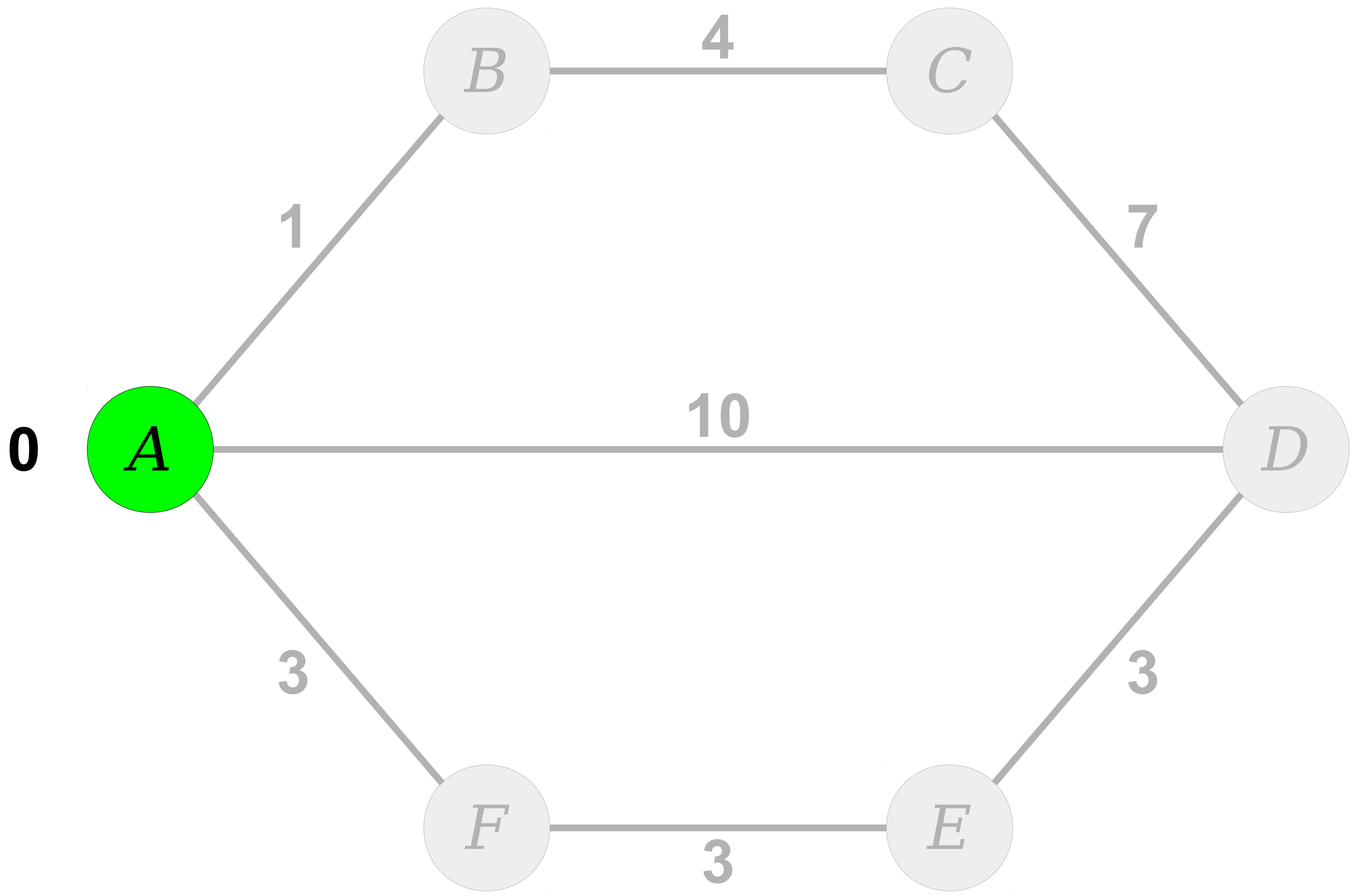
B
4
C
1
7
A
10
D
3
3
F
3
E



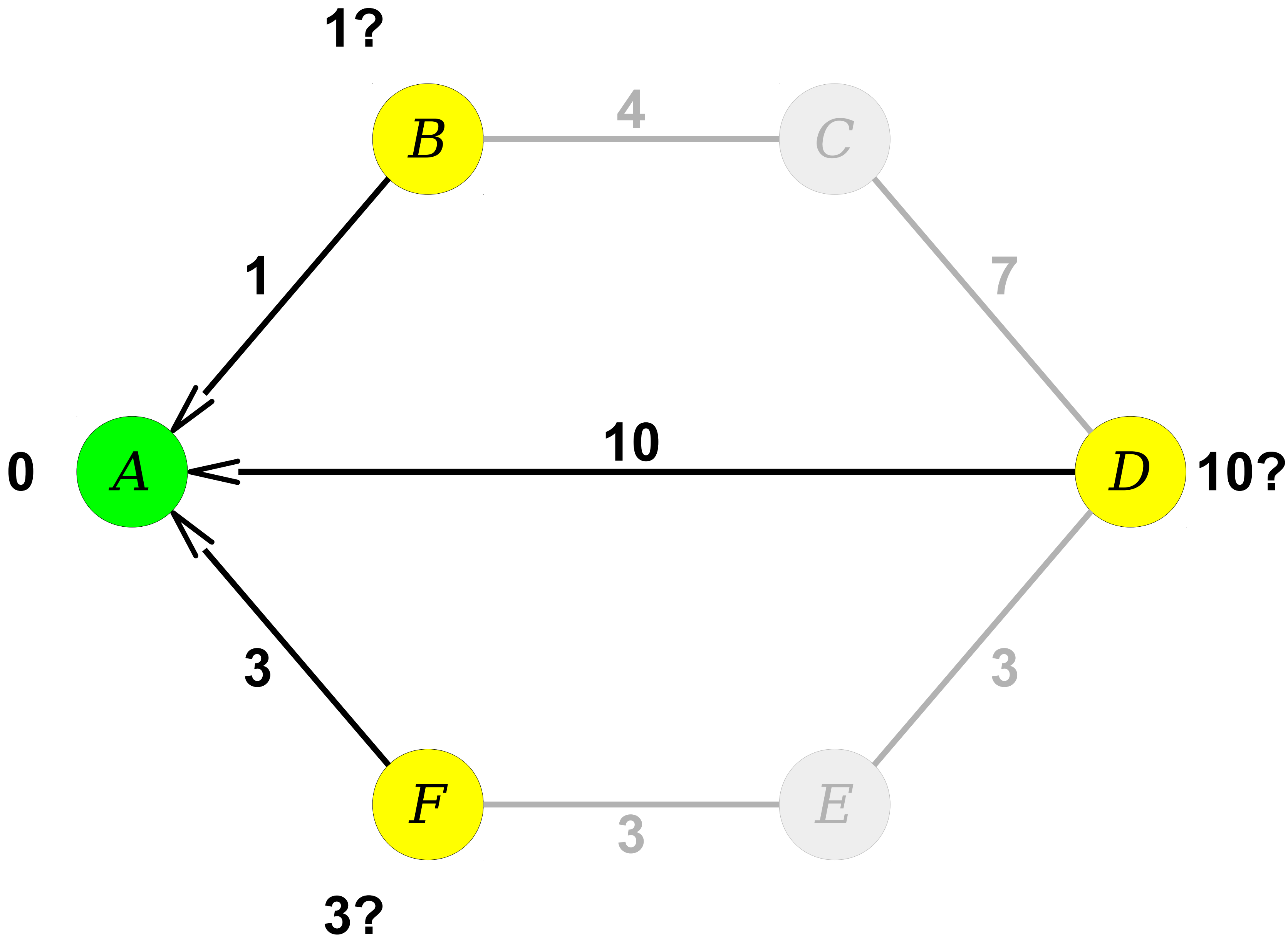
B
4
C
1
7
A
10
D
3
3
F
3
E



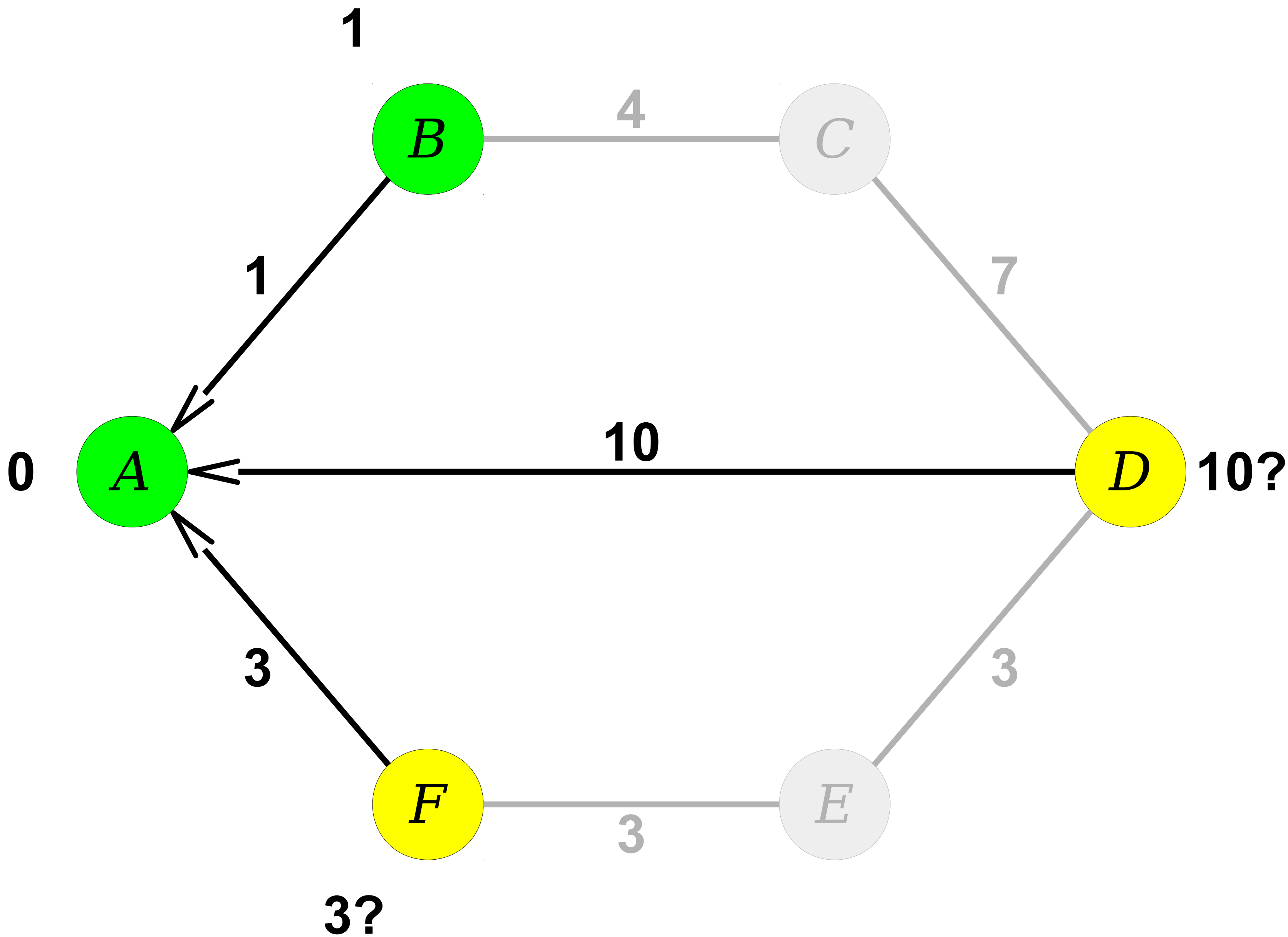
A
B
C
D
E
F
1
4
7
10
3
3
3

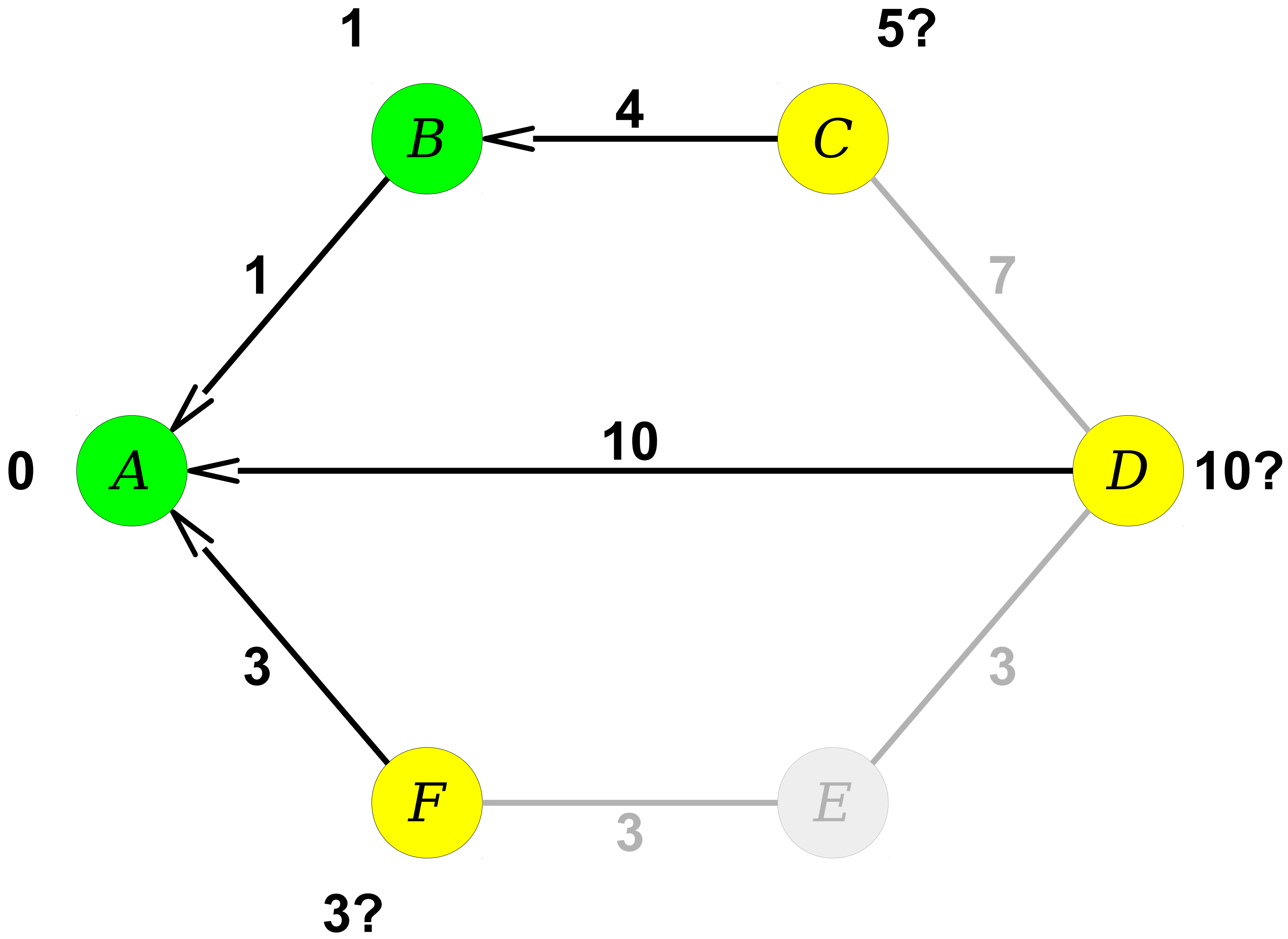


4
1
7
0
A
B
C
D
E
F
10
3
3
3

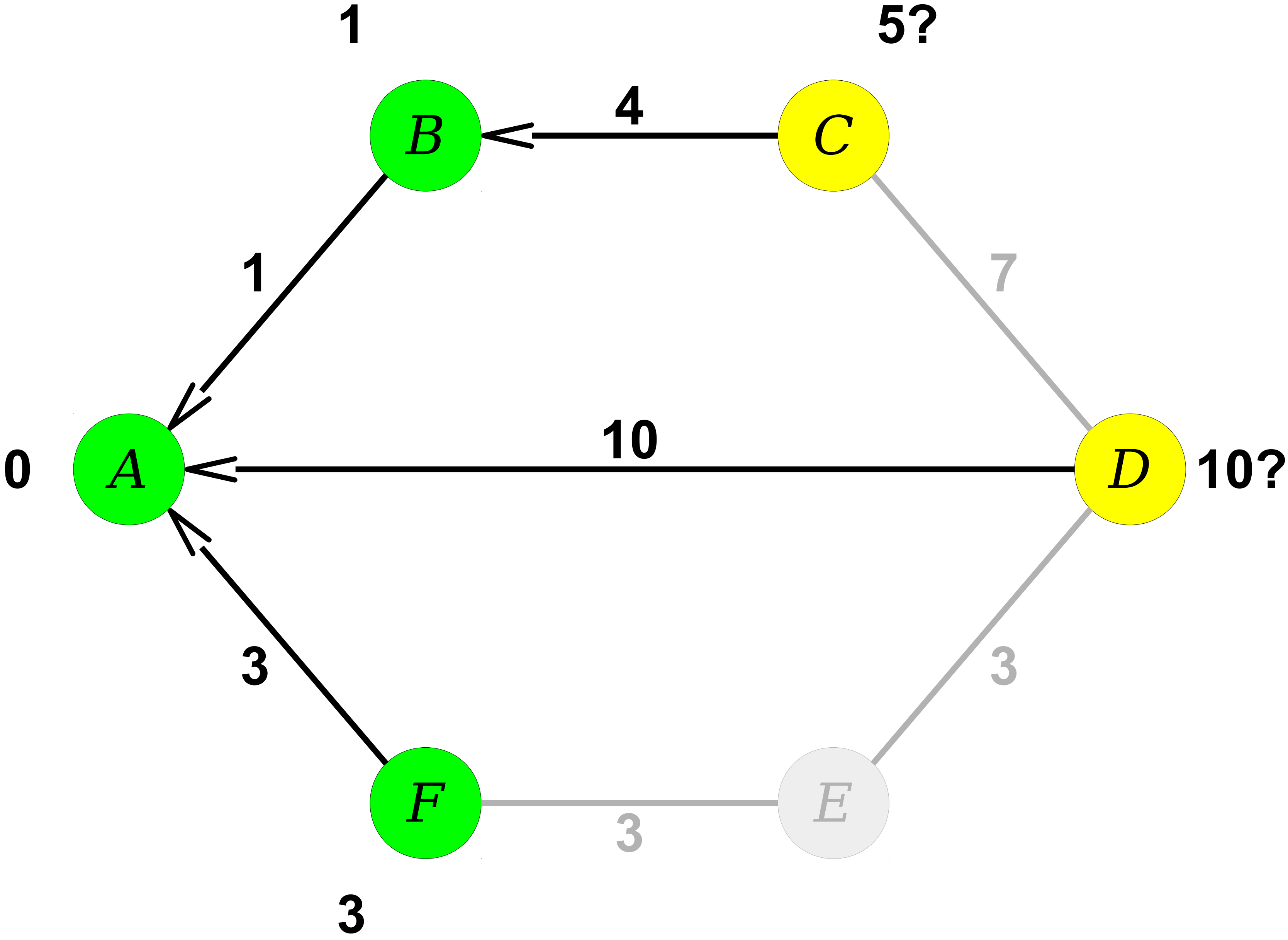


1?
B
4
C
1
7
0
A
10
D
10?
3
3
F
3
E
3?

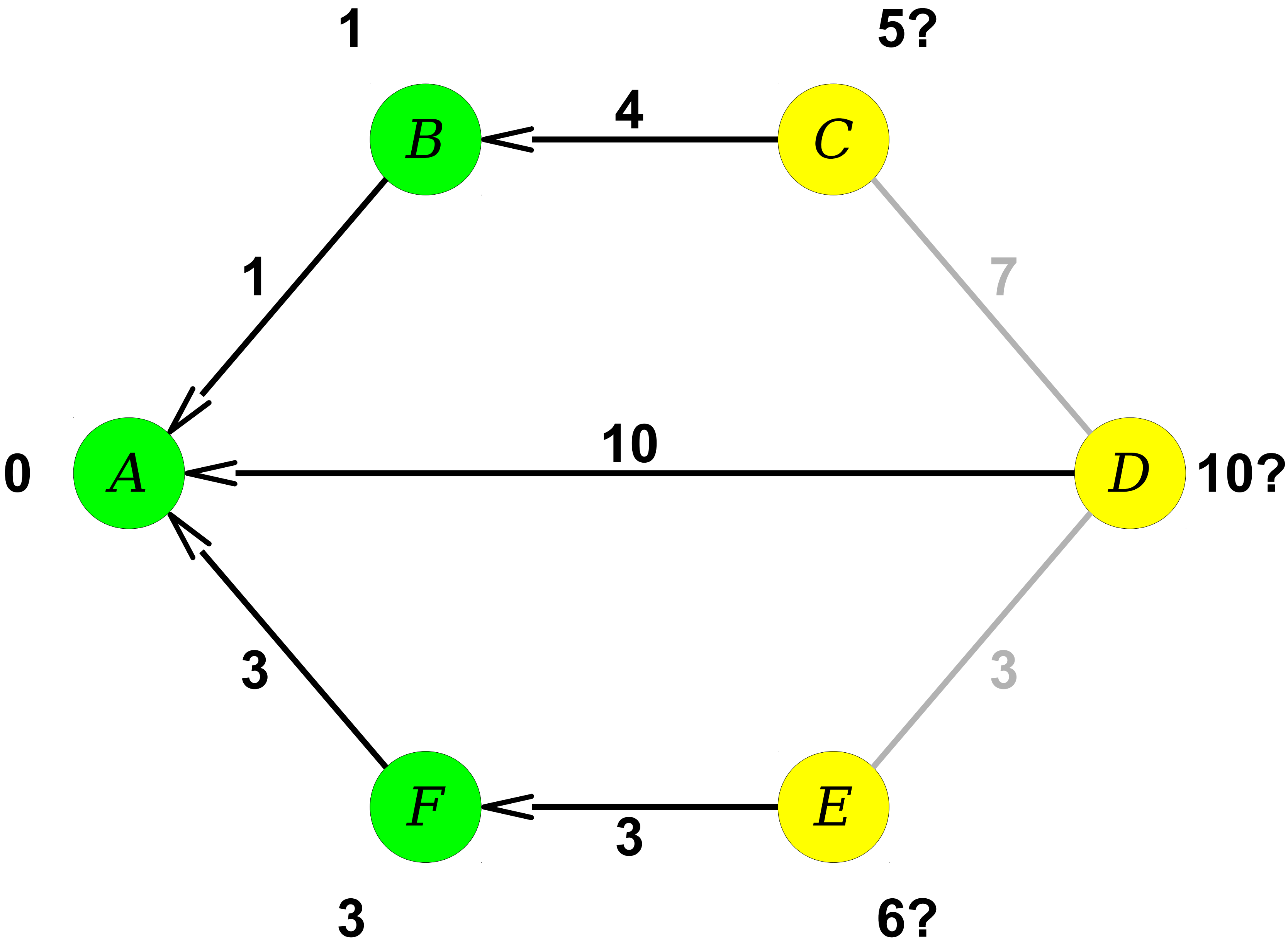

1
B
4
C
1
7
10
0
A
D
10?
3
3
F
E
3
3?

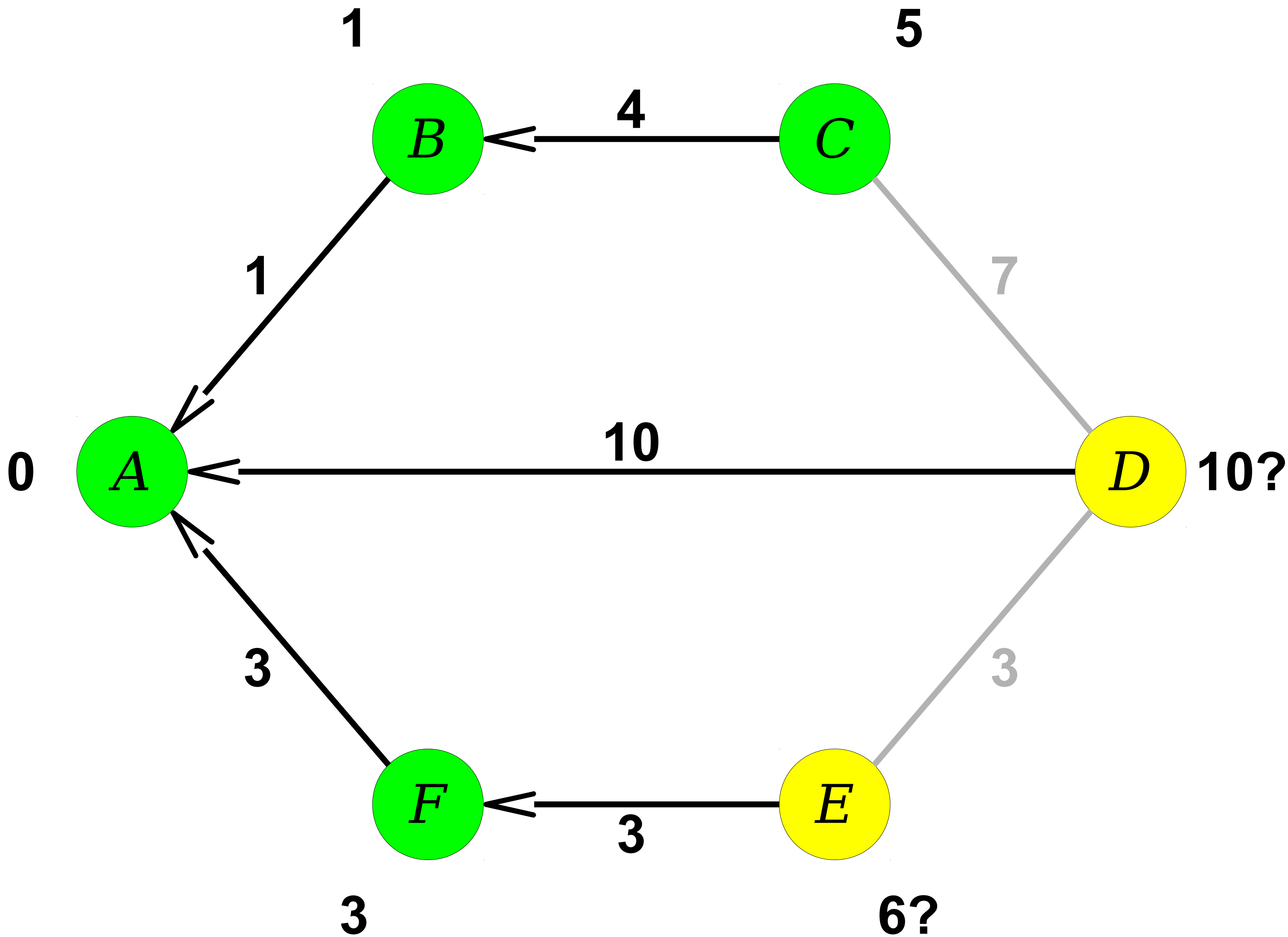


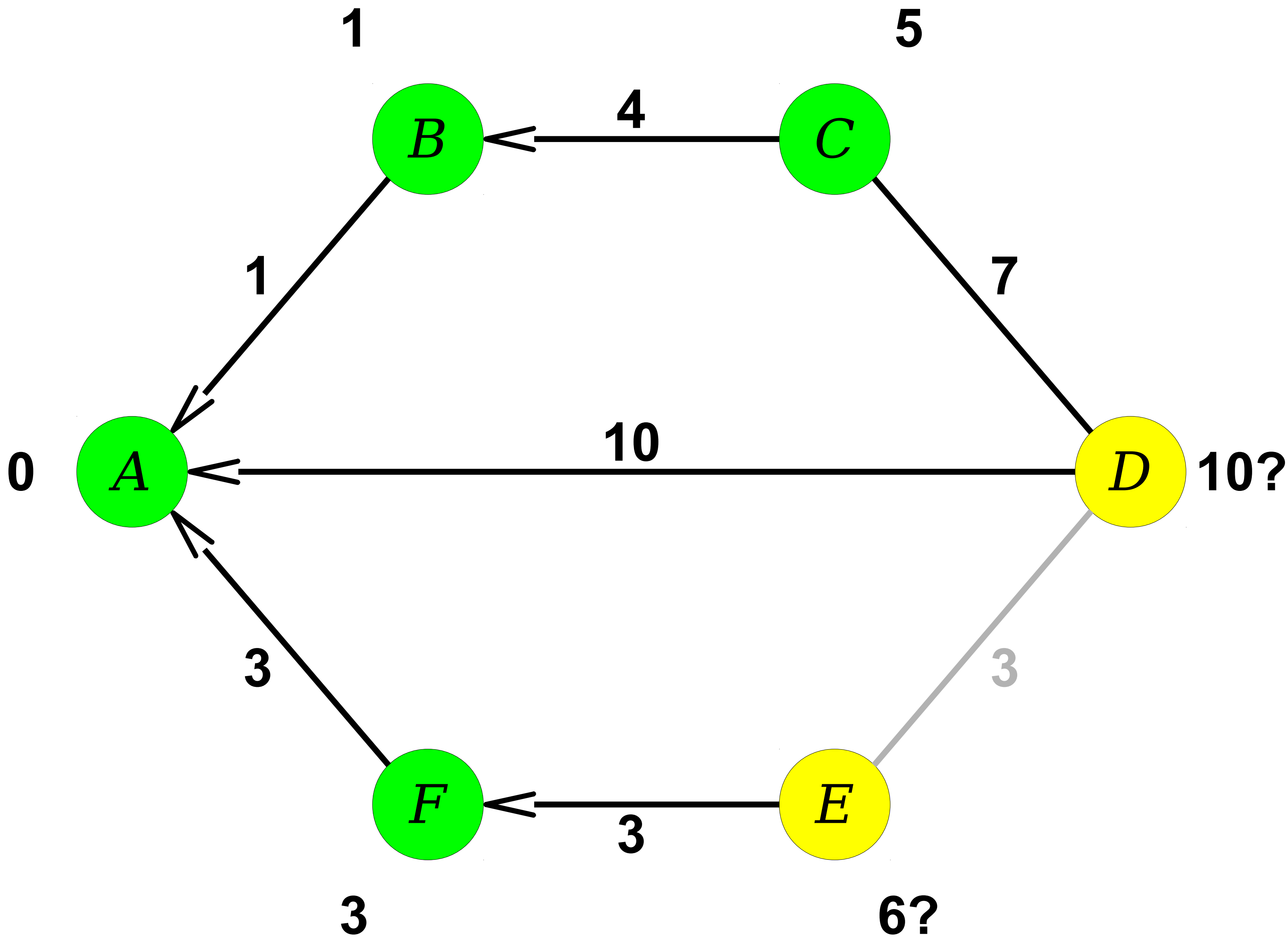
1
5?
4
A
B
C
D
E
F
1
7
0
10
10?
3
3
3
3?



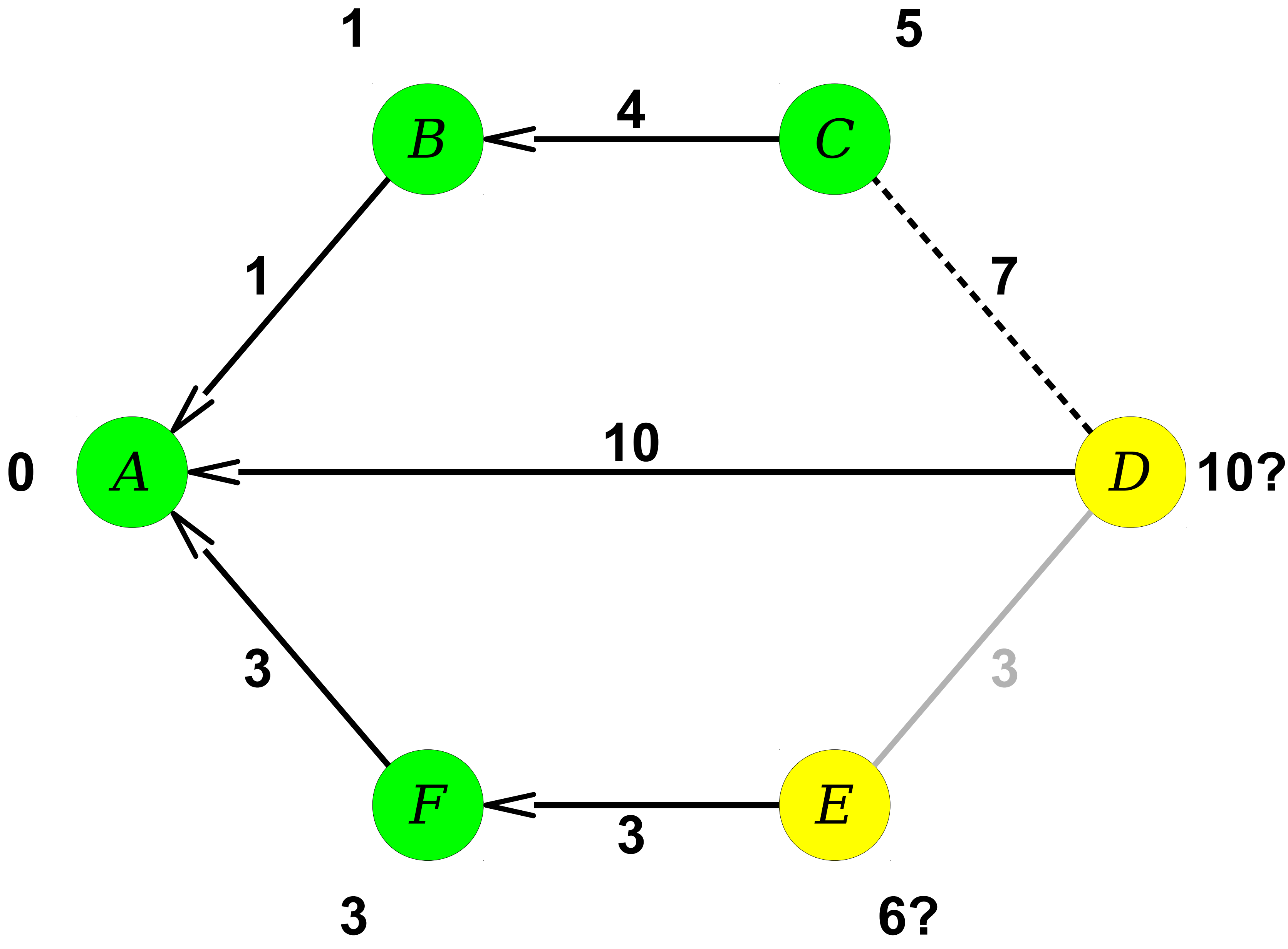
1
5?
4
B
C
1
7
0
A
10
D
10?
3
3
F
E
3
3

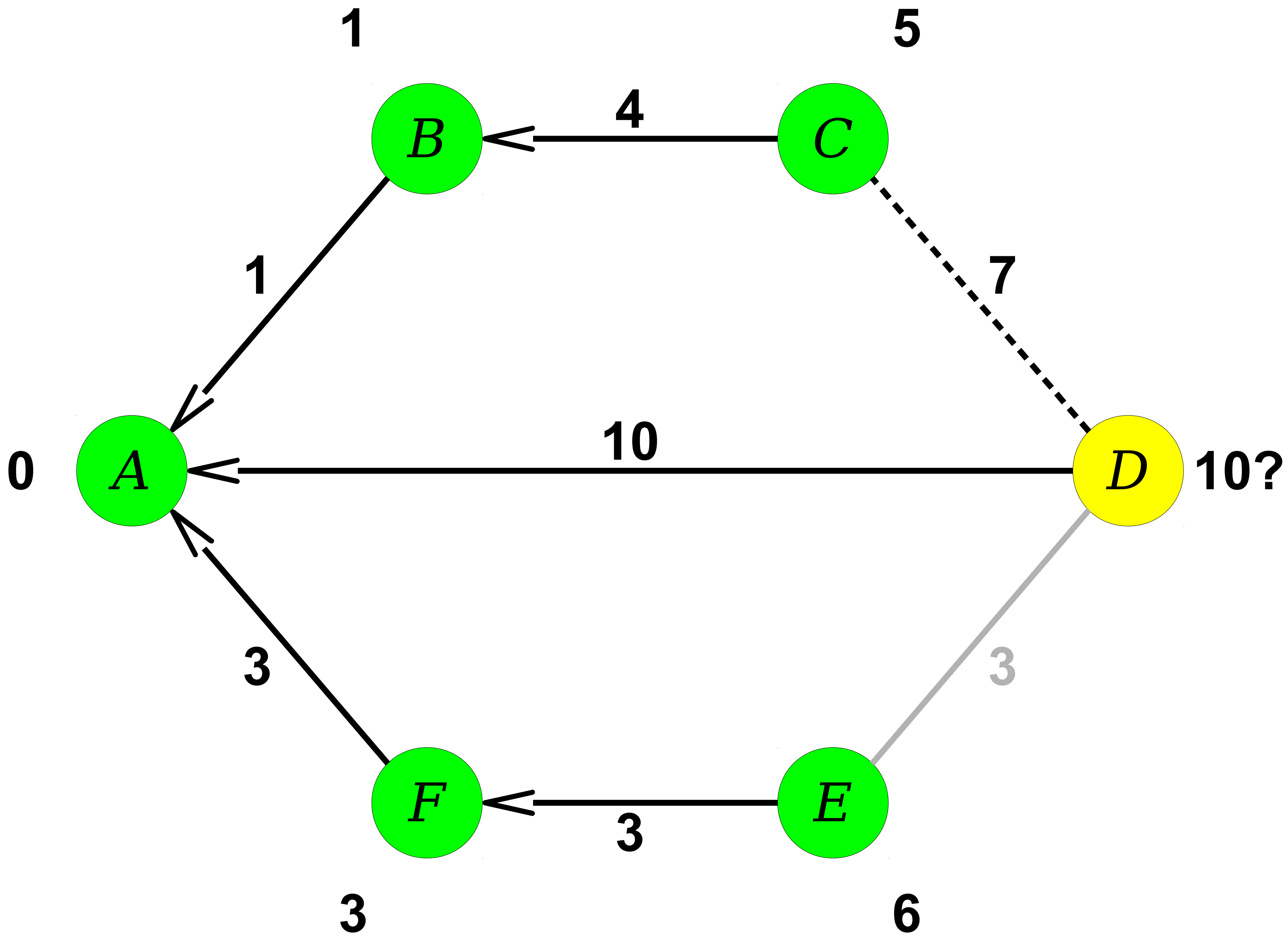

1
5?
B
4
C
1
7
0
A
10
D
10?
3
3
F
3
E
3
6?

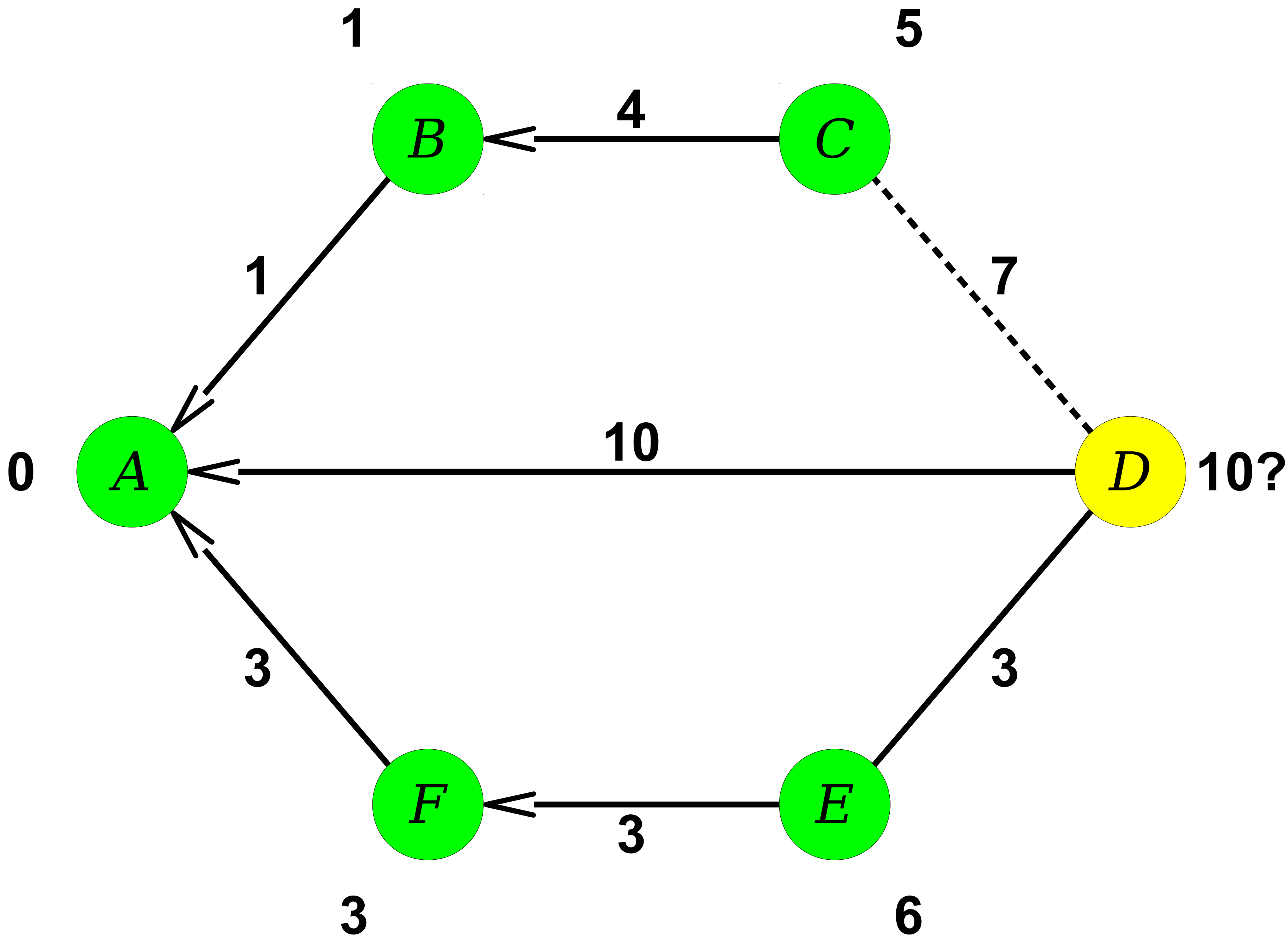

1
5
B
4
C
1
7
0
A
10
D
10?
3
3
F
3
E
3
6?

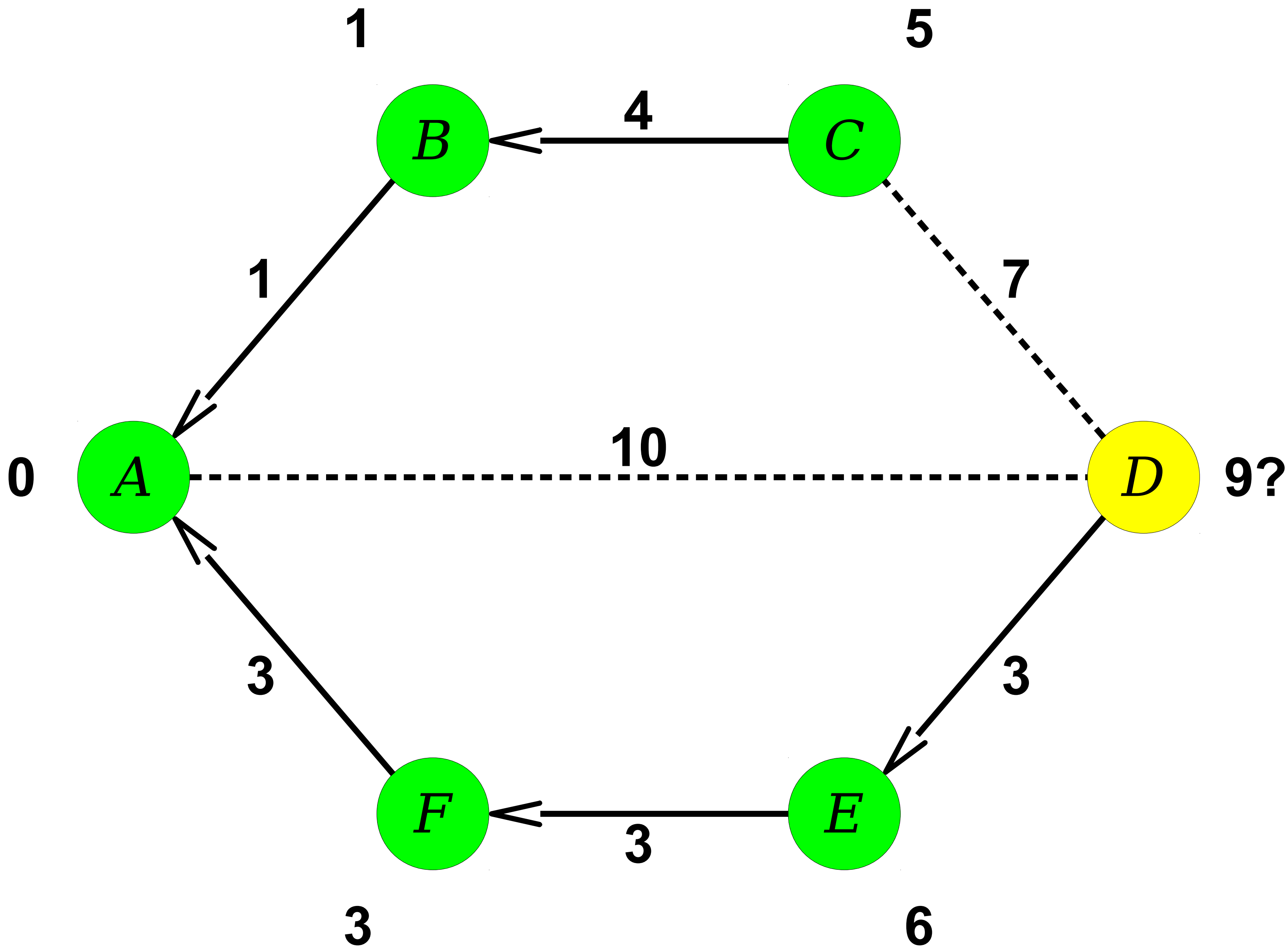


1
5
B
4
C
1
7
0
A
10
D
10?
3
3
F
3
E
3
6?

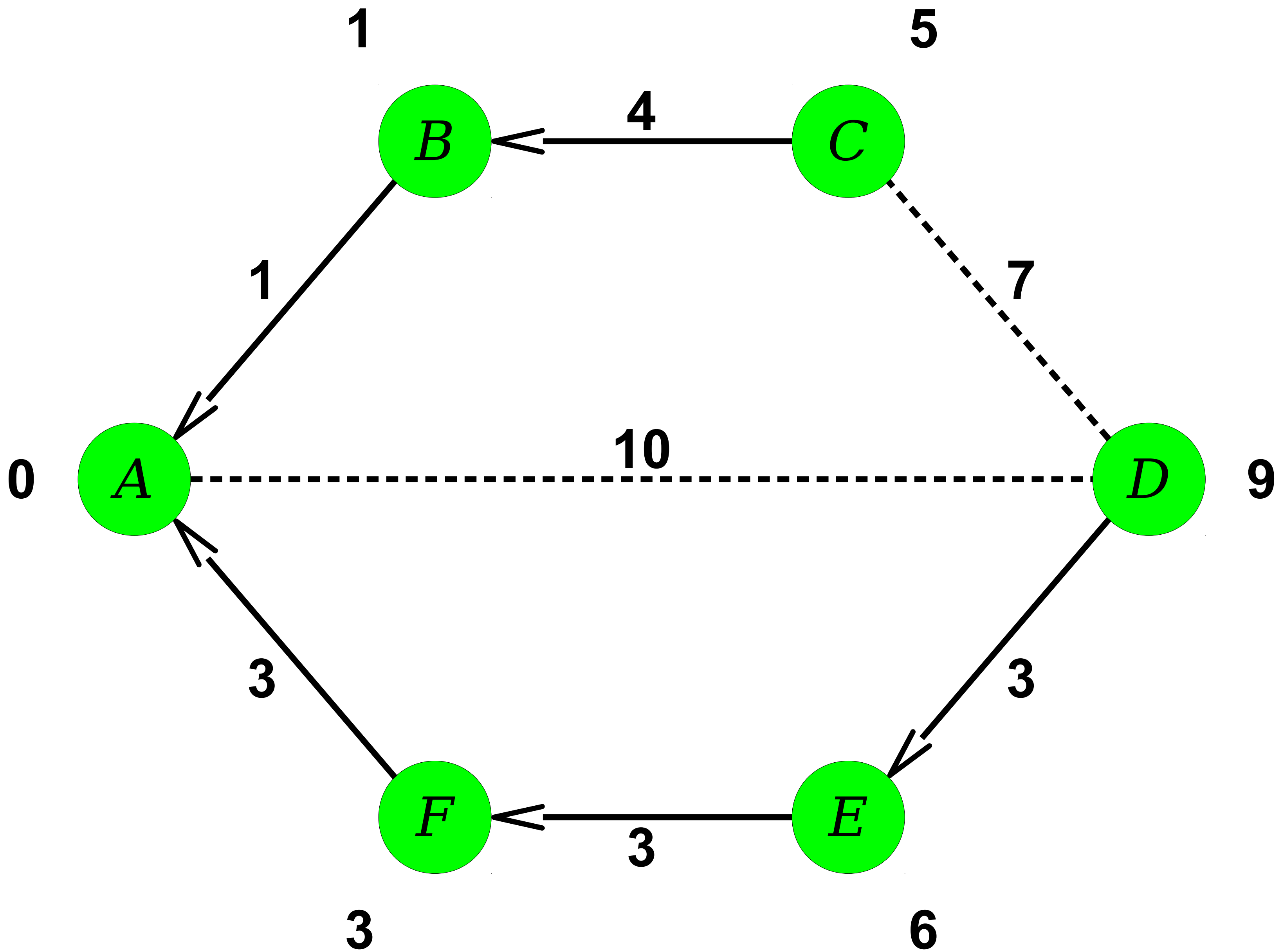

1
5
B
4
C
1
7
0
A
10
D
10?
3
3
F
3
E
3
6?

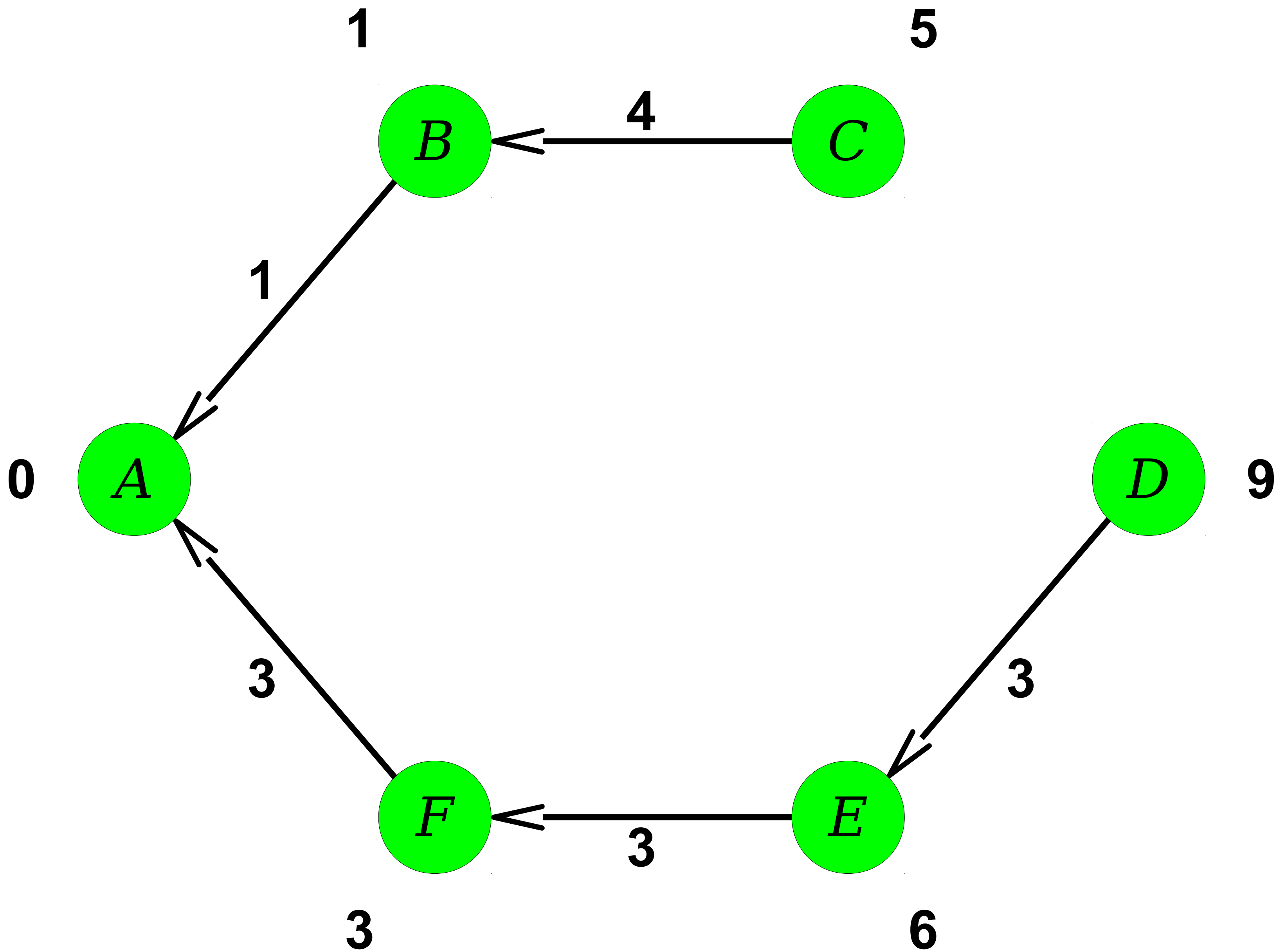

1
5
B
4
C
1
7
0
A
10
D
10?
3
3
F
3
E
3
6


1
5
B
4
C
1
7
0
A
10
D
10?
3
3
F
3
E
3
6



1
5
B
4
C
1
7
0
A
10
D
9?
3
3
F
3
E
3
6


1
5
B
4
C
1
7
0
A
10
D
9
3
3
F
3
E
3
6


1
5
B
4
C
1
0
A
D
9
3
3
F
3
E
3
6

Time-Out for Announcements!

Assignment 6

- Assignment 7 (***Trailblazer***) goes out today. It's due next Friday, March 17th, at the start of class.
  - Play around with BFS, Dijkstra's algorithm, and A* search (coming up!) in The Real World!
  - ***You are encouraged to complete this assignment in pairs***. You don't need to write much code, but you'll need to have a good conceptual handle on these algorithms.
  - ***No late days may be used, and no late submissions will be accepted***. This is a university policy (thanks, federalism!) and we don't have any wiggle room with it.
  - ***Recommendation:*** Complete BFS and Dijkstra's algorithm by Monday.
- Anton will be holding YEAH hours today from ***2:30PM – 3:30PM*** in ***370-370***.
- Assignment 6 was due at the start of class today.
  - ***Be strategic about taking late days on this assignment***. You'll be cutting into the time you need to spend on Assignment 7.

BLACK IN COMPUTER SCIENCE PRESENTS

BLACK IN CS
KICKBACK

5 - 7PM | FRIDAY, MARCH 10TH
A3C | LOUNGE

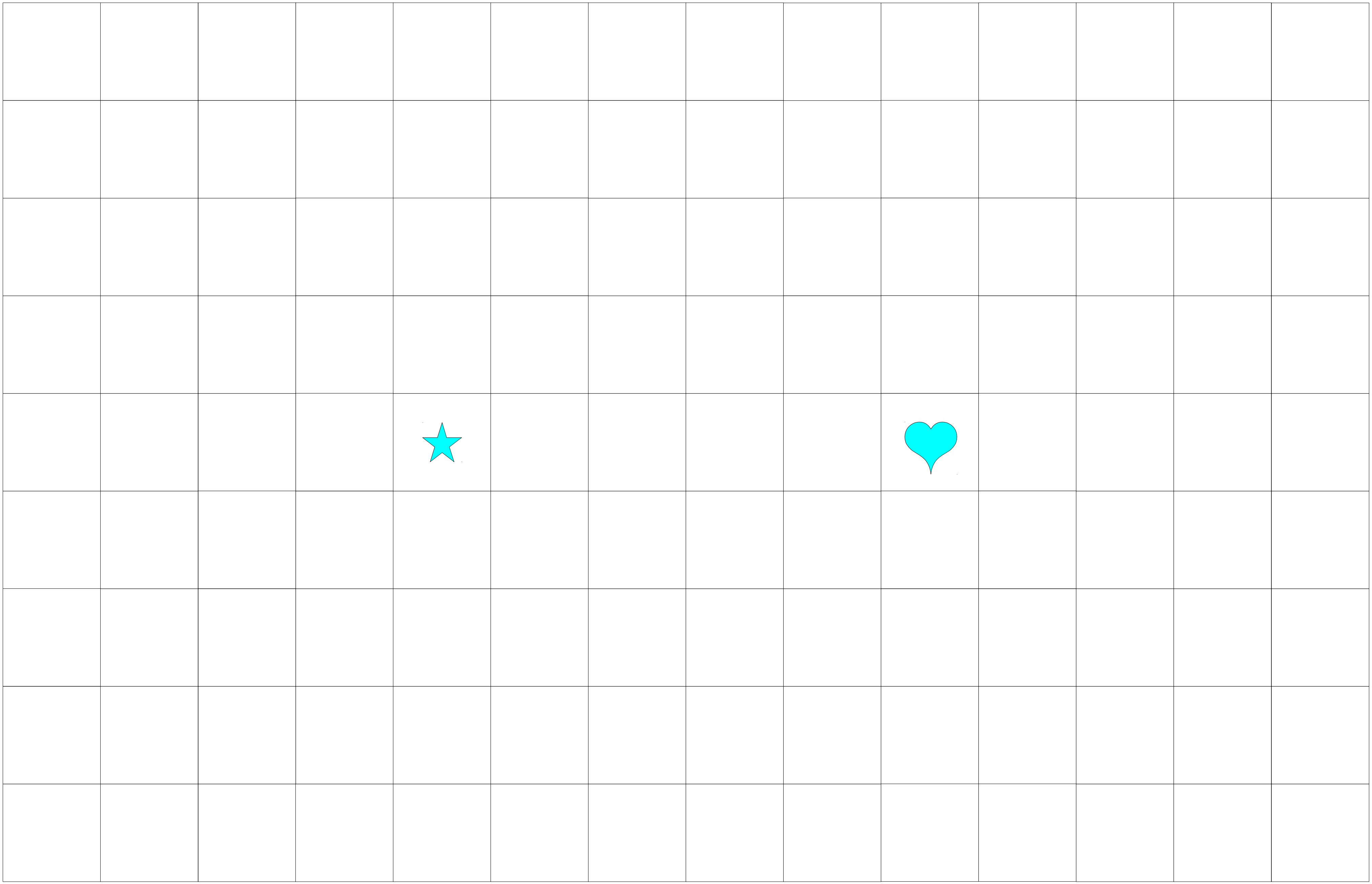
FOOD + FUN + FRIENDS
DINNER PROVIDED

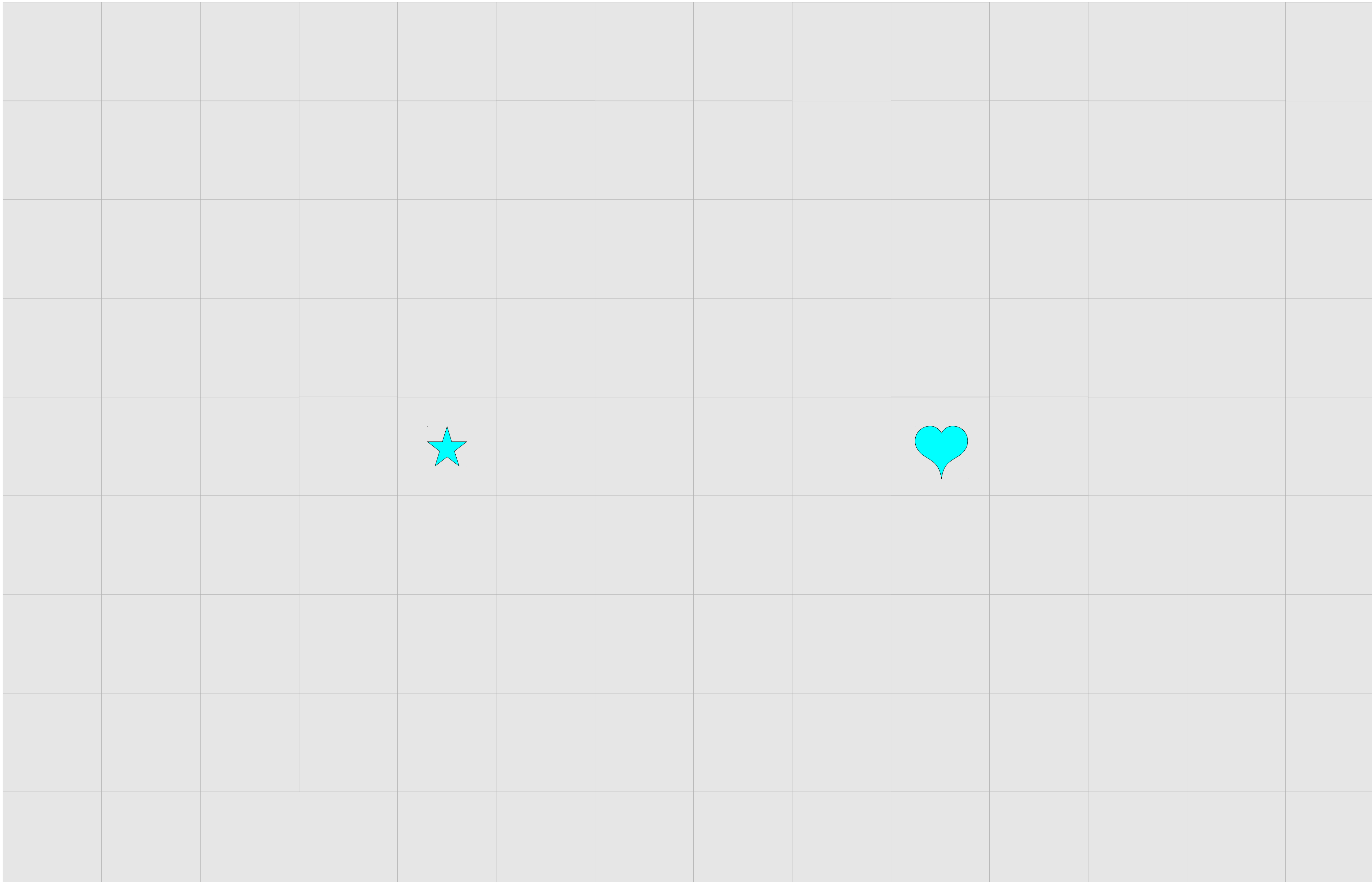
Final Exam Logistics

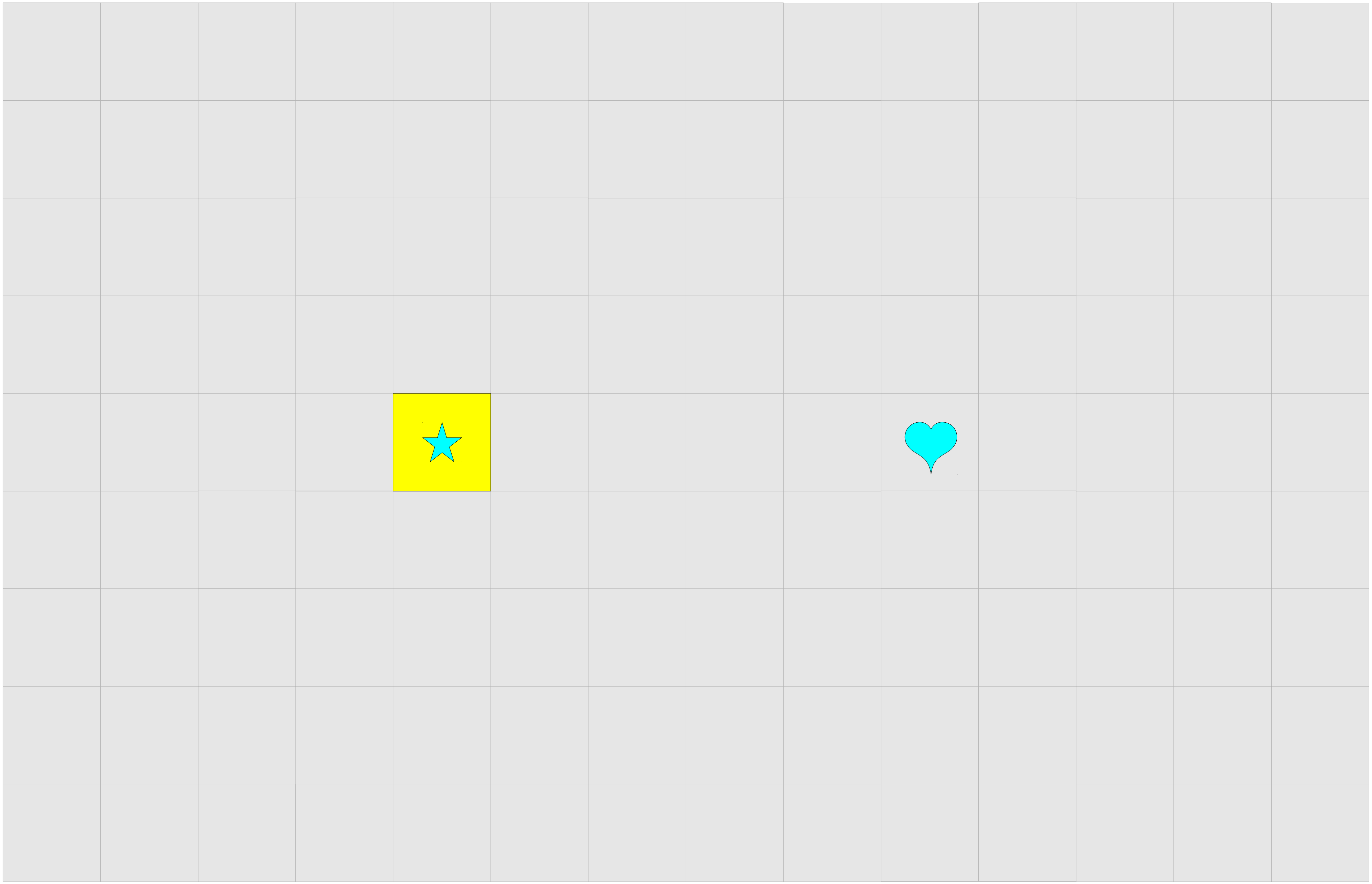
- Our final exam is on Monday, March 20th from 8:30AM – 11:30AM, location TBA.
  - Sorry about the timing! That was the registrar's decision.
- Format is same as the midterm: closed-book, closed-computer, limited-note. You get a single, double-sided sheet of 8.5" × 11" notes decorated however you'd like.
- Cumulative exam, slightly focused on the post-midterm topics.
  - Covers topics from all assignments from this quarter.
  - Covers topics from lectures up through and including this upcoming Monday.
- We will be holding a practice exam on ***Monday, March 13th*** from ***7:00PM – 10:00PM***, location TBA. Same deal as the practice midterm: I'm drafting two final exams, one which will be the practice, and one of which will be the main alternate.
- Have OAE accommodations? We'll reach out to you soon to coordinate alternate exams.

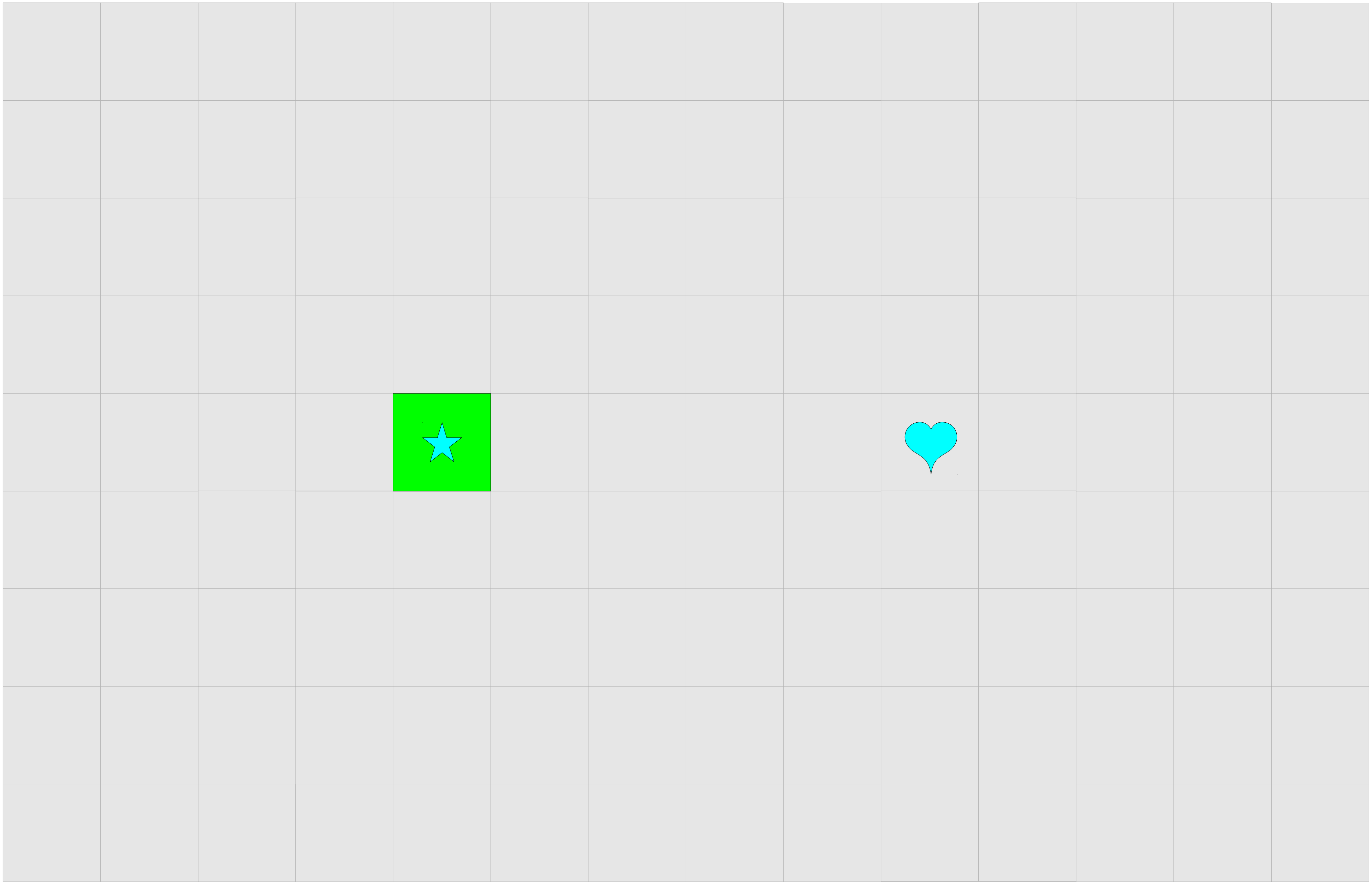
Back to CS106B!

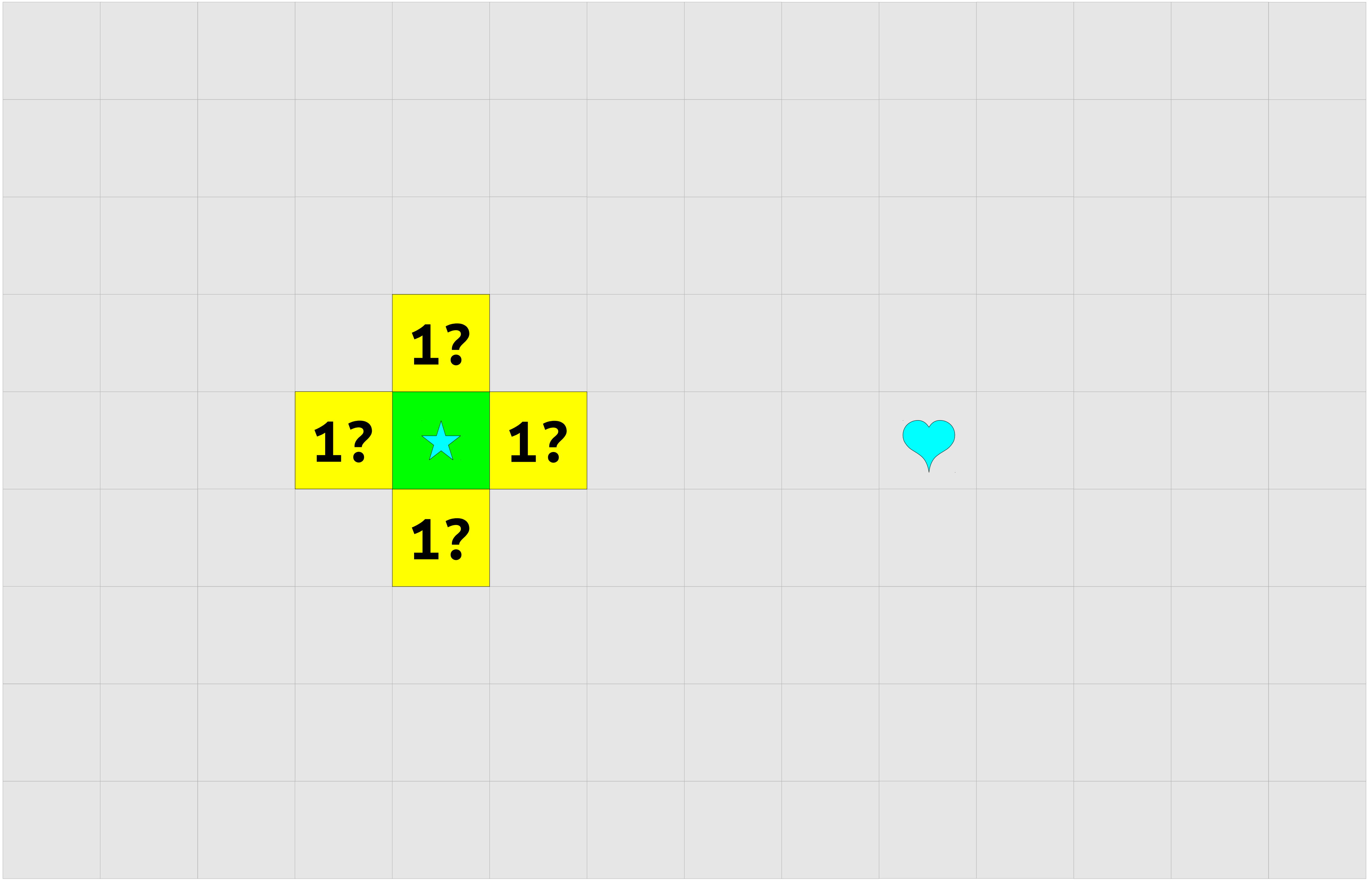
One Detail with Dijkstra's Algorithm

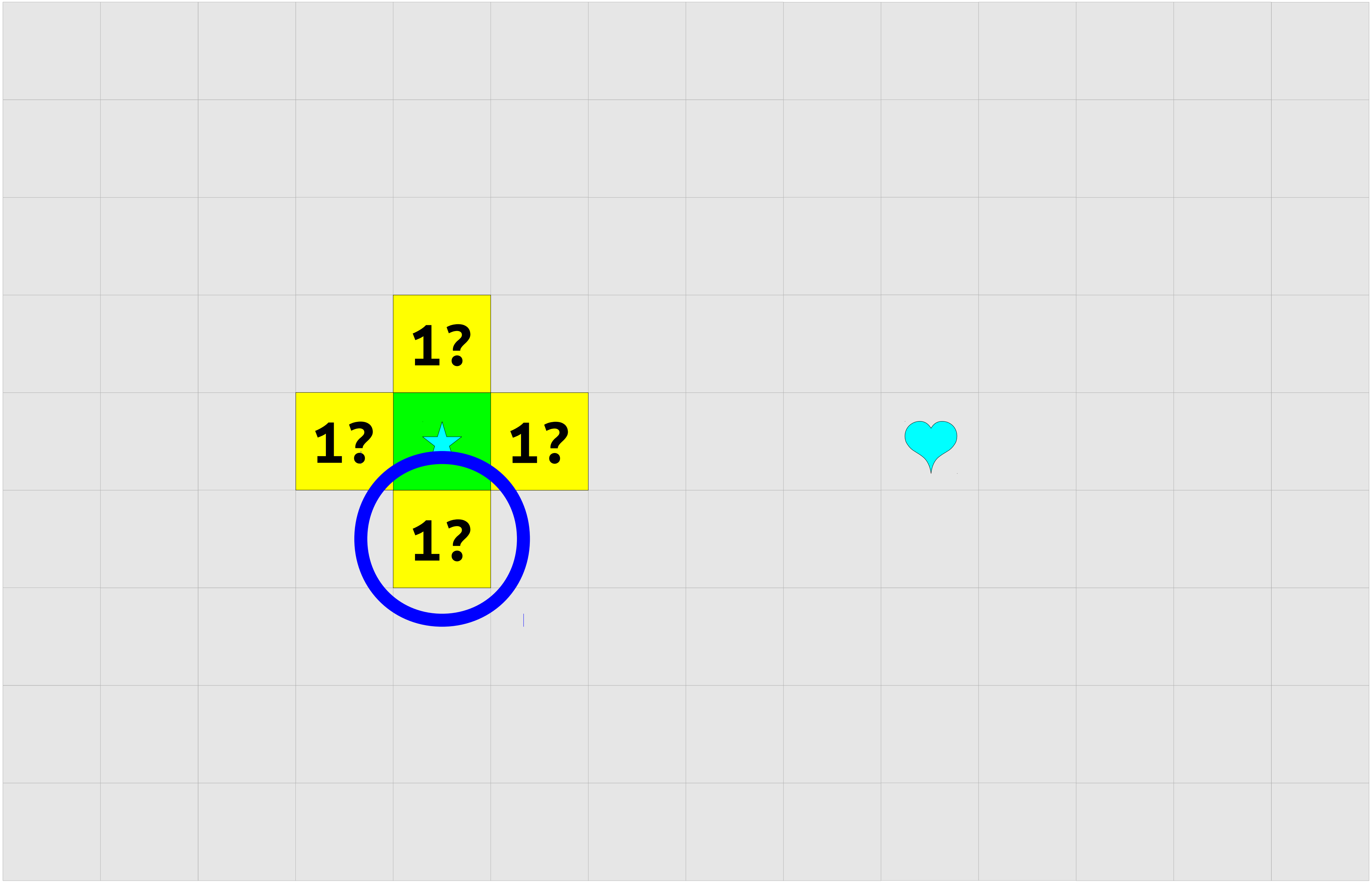


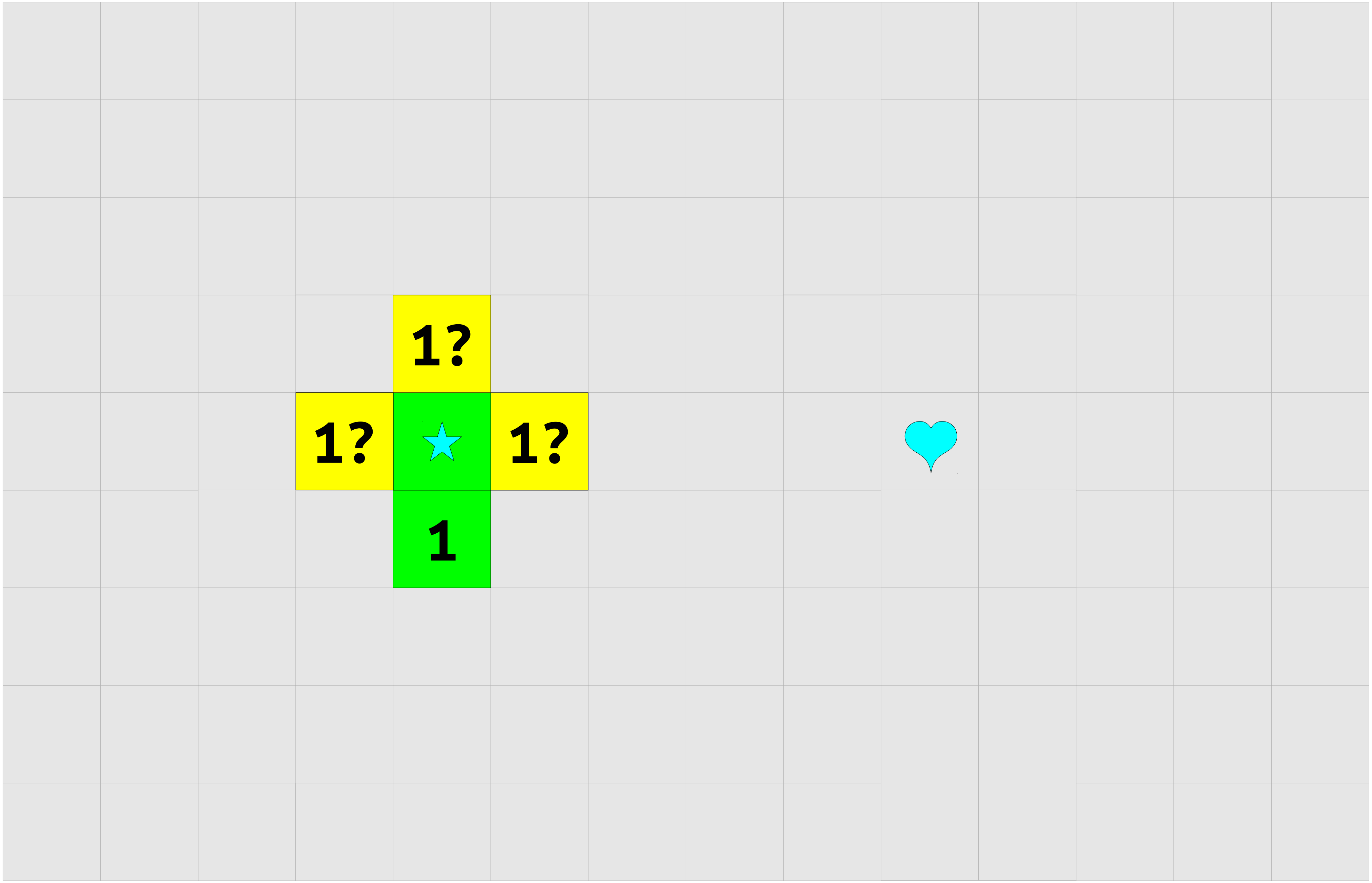


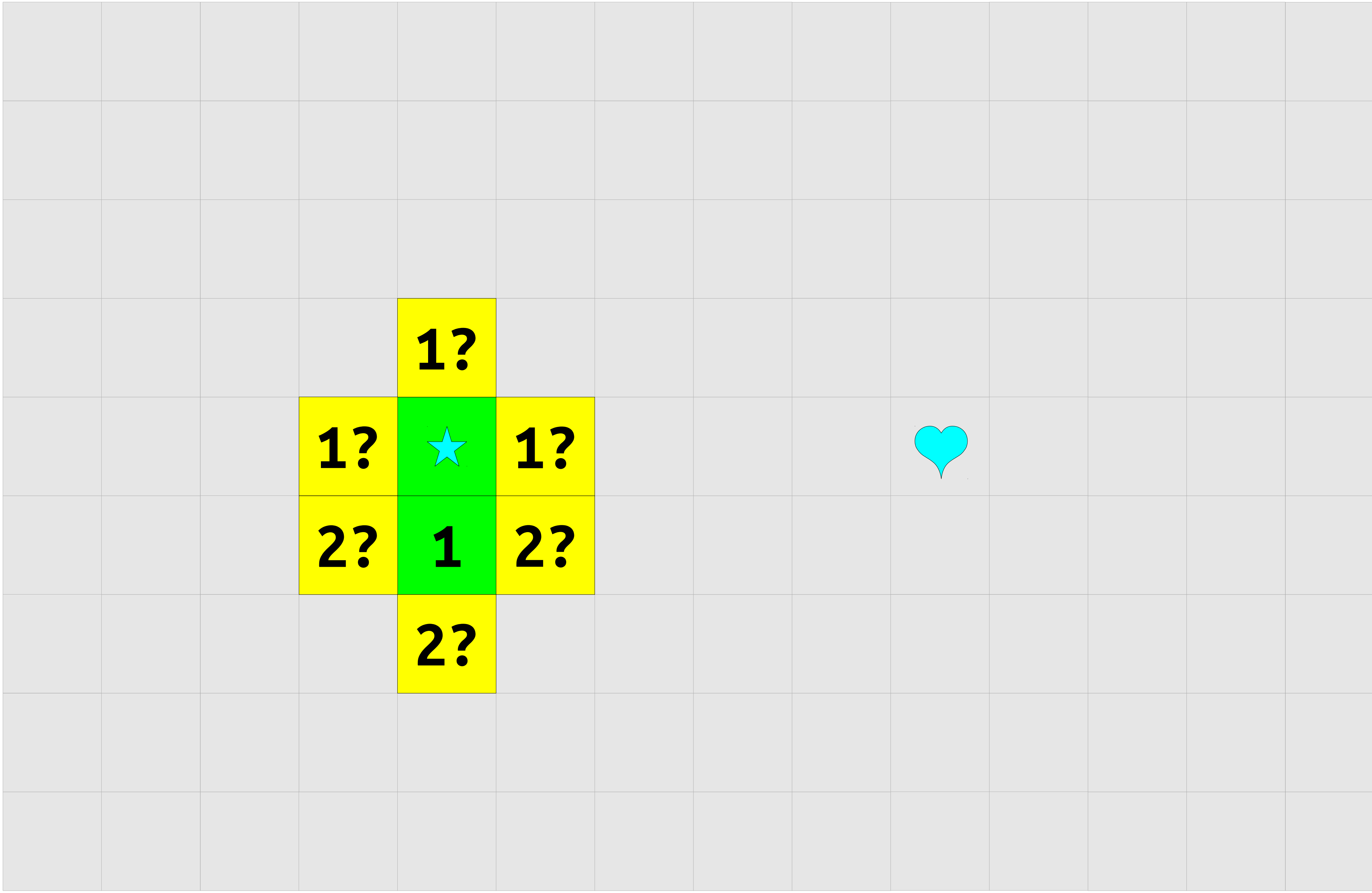


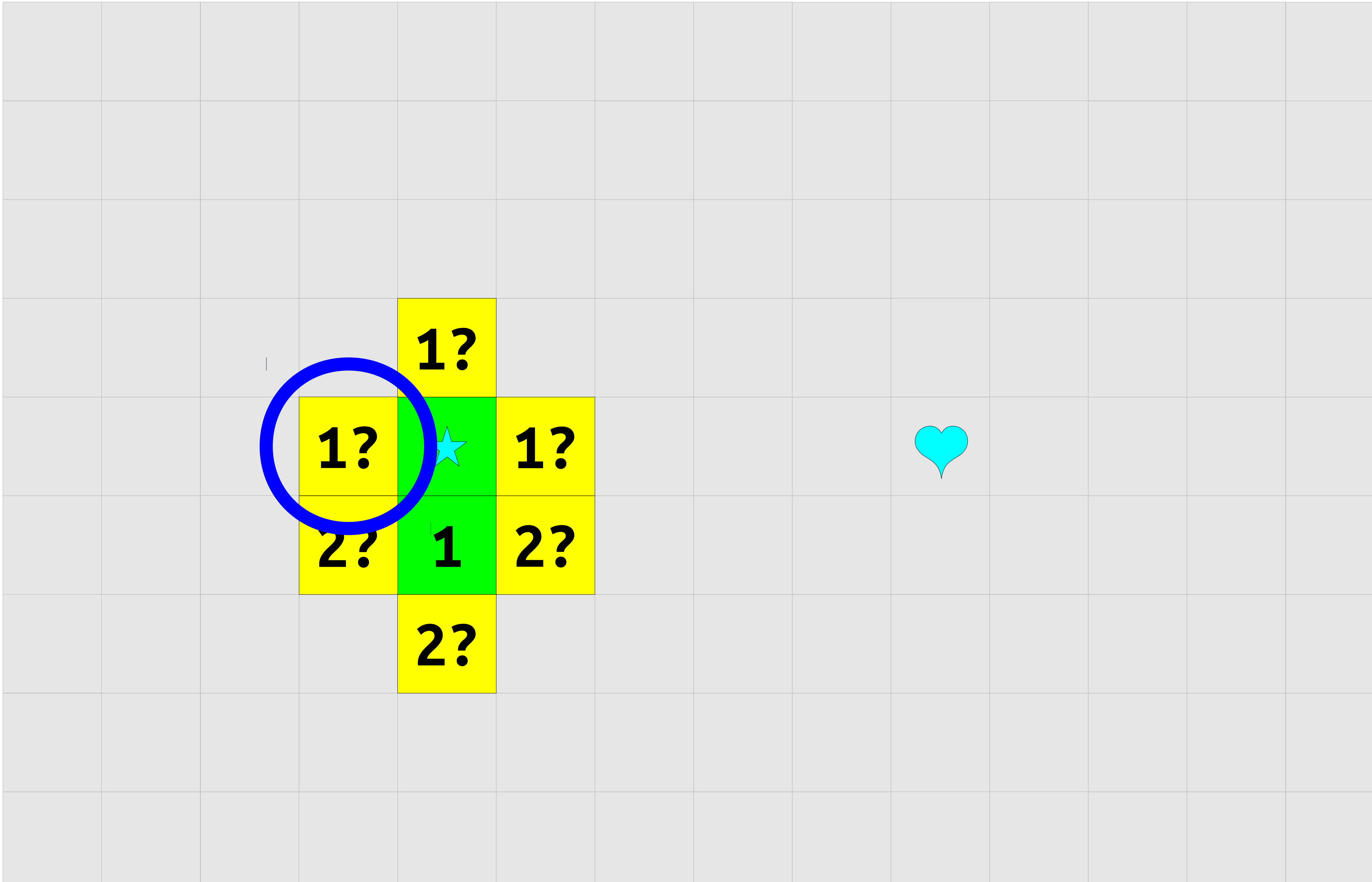


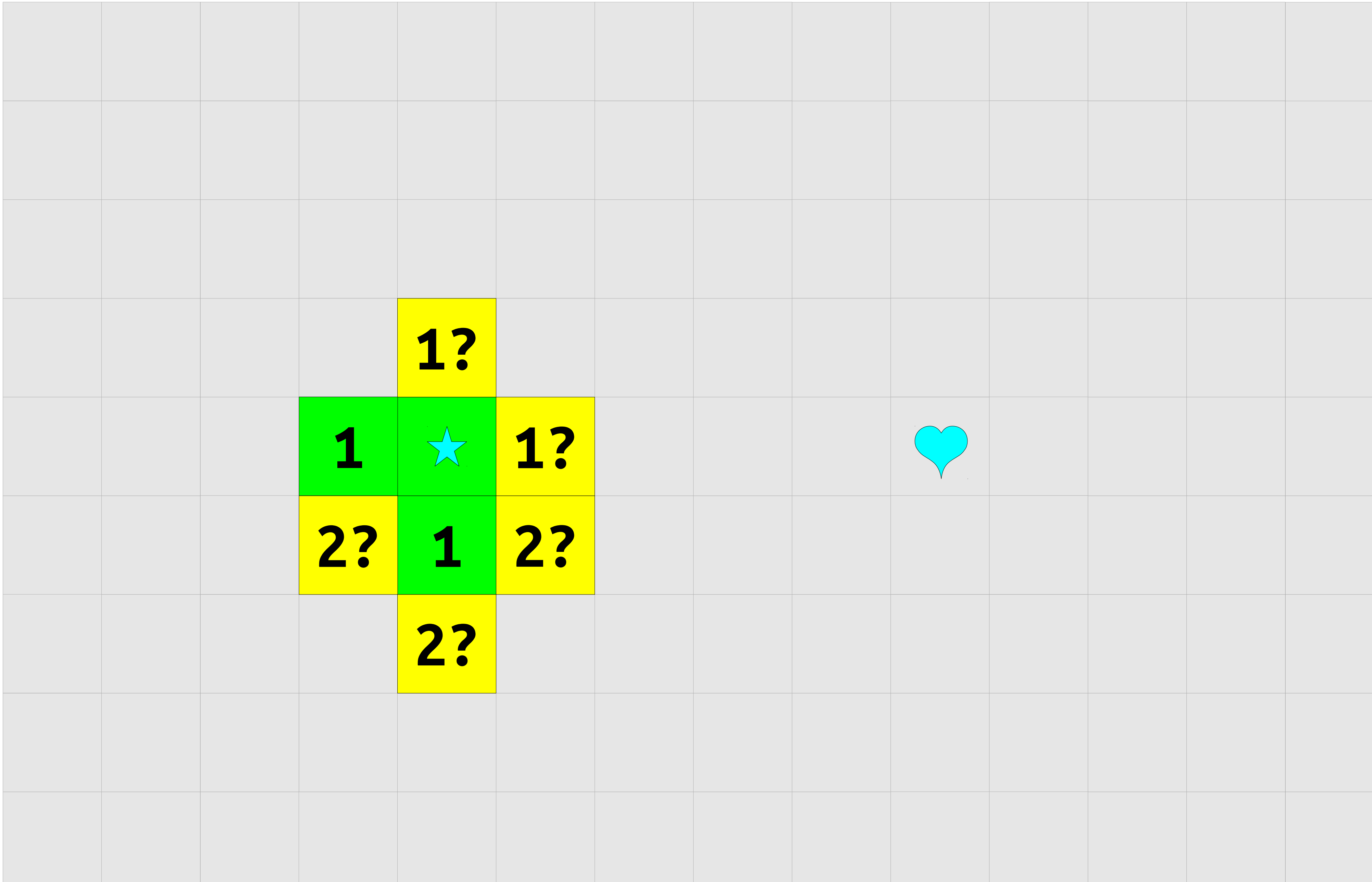

1?
1?
1?
1?

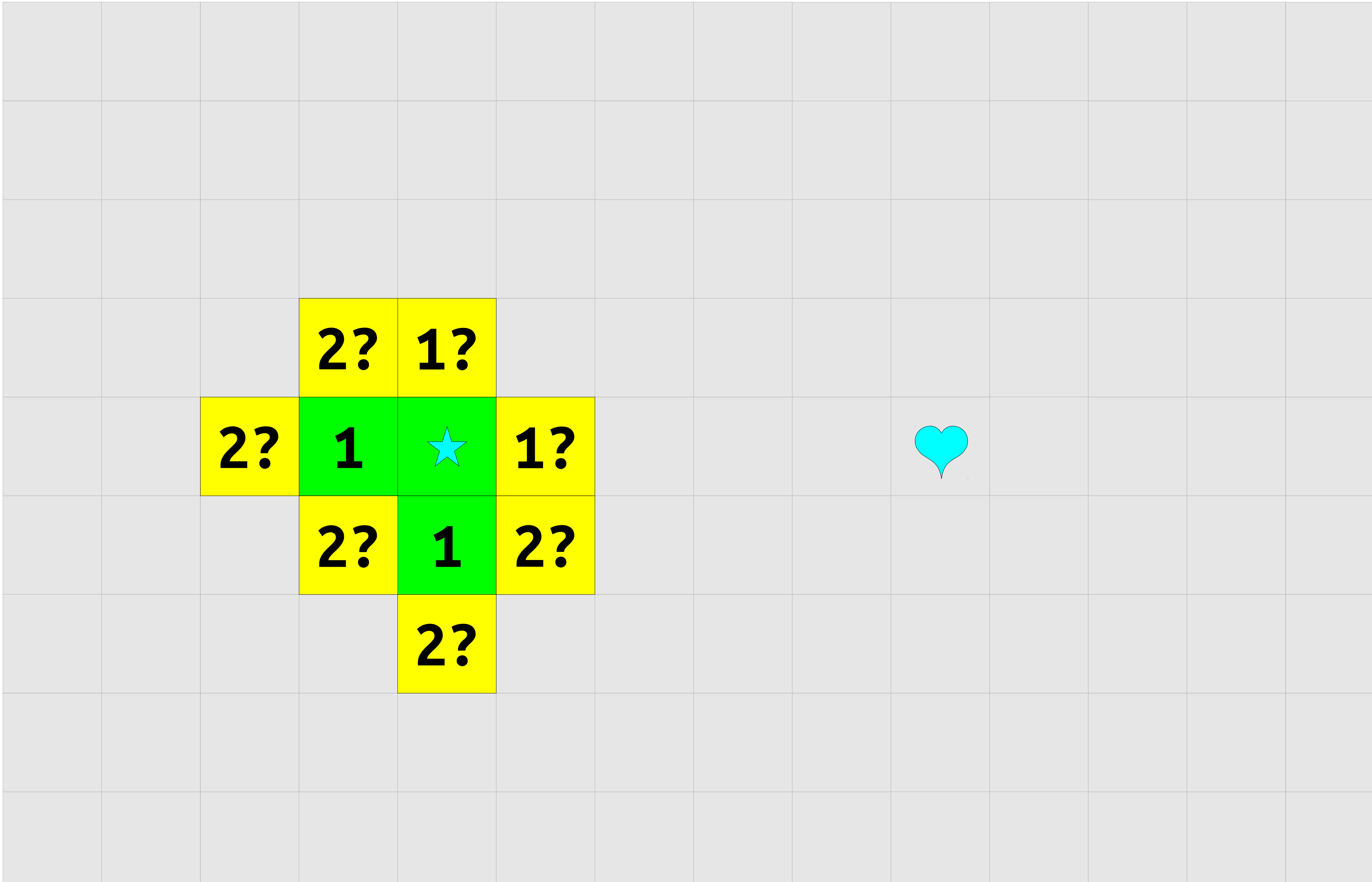

1?
1?
1?
1?

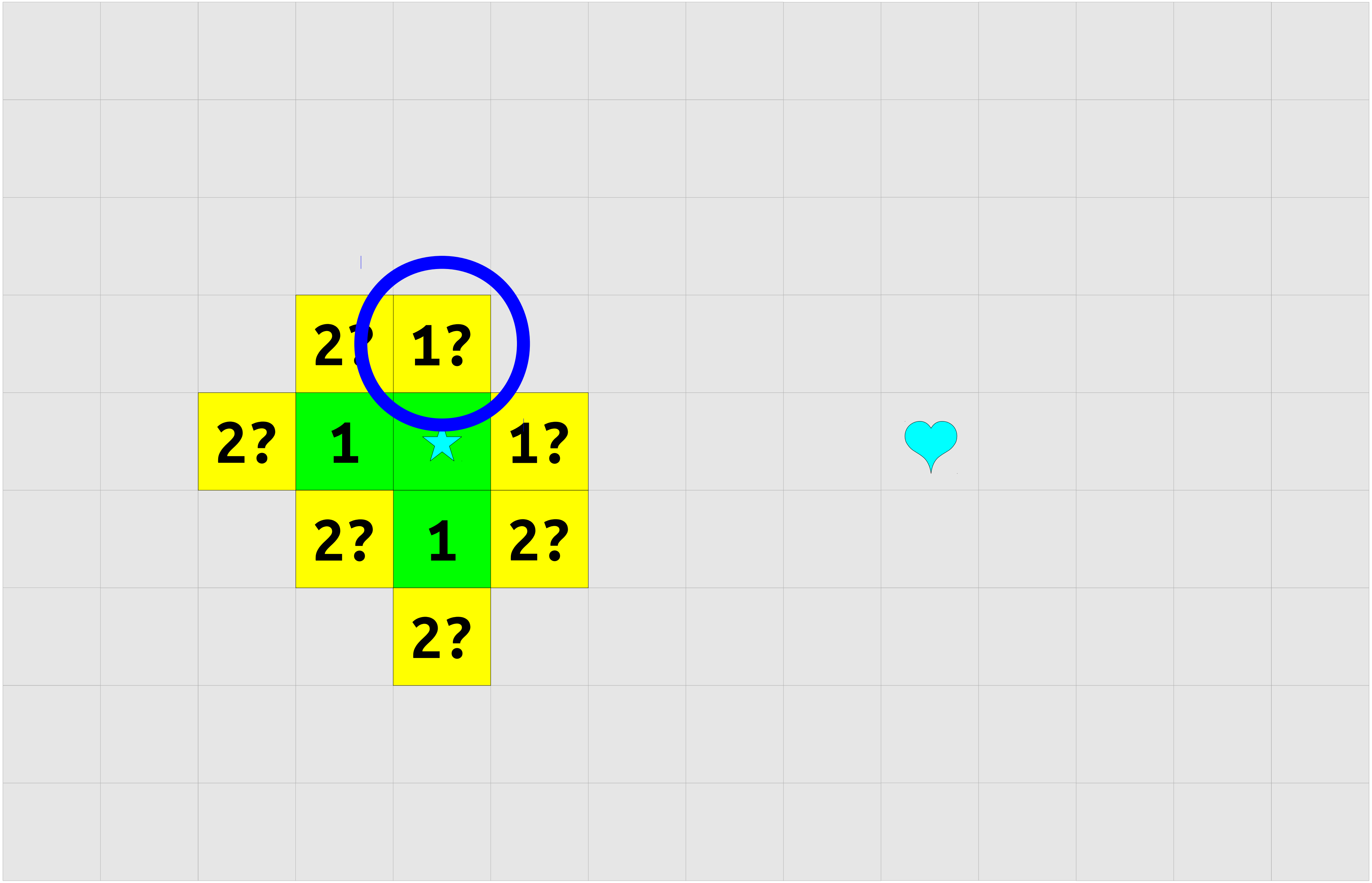

1?
1?
1?
1

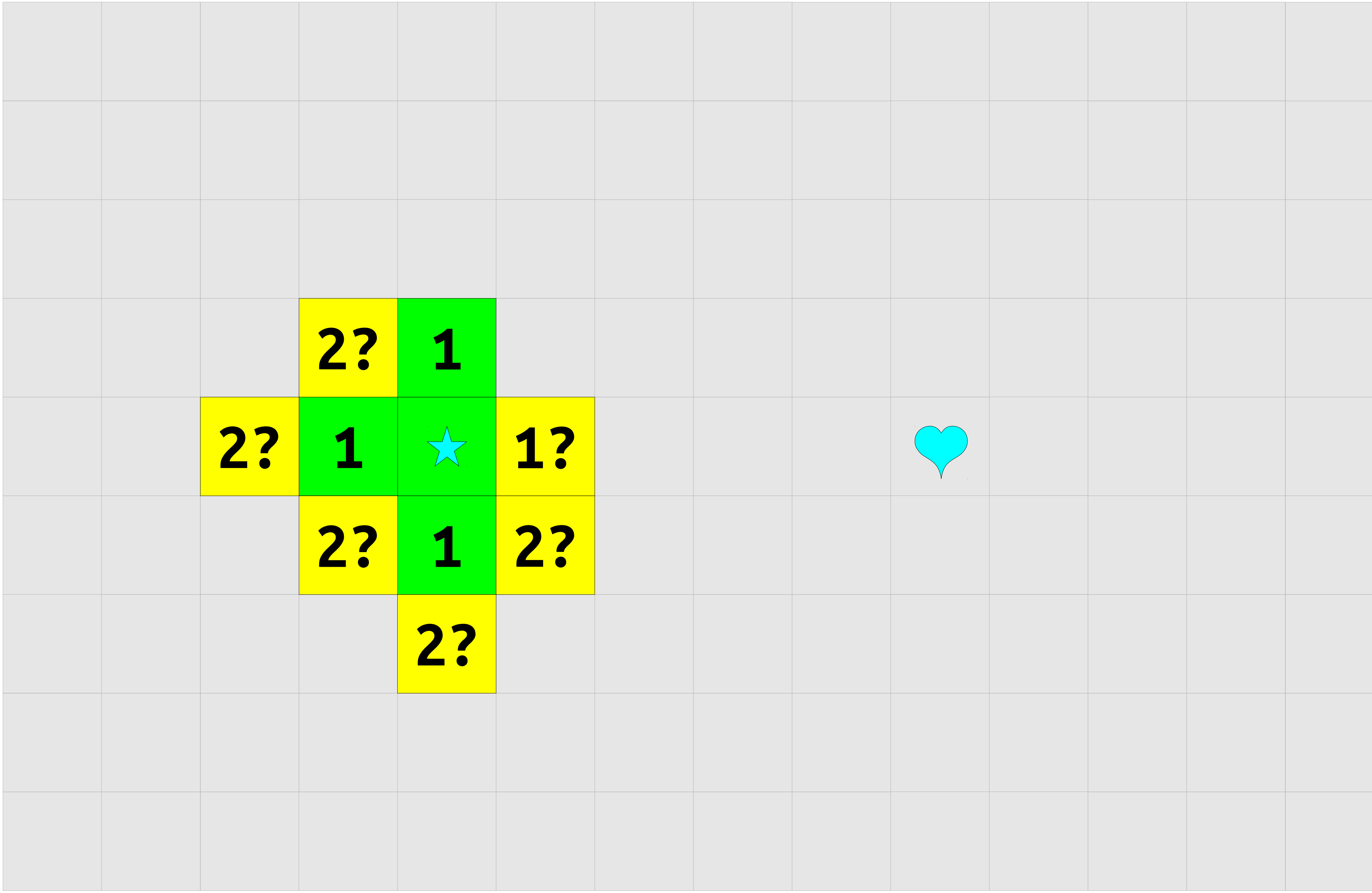

1?
1?
1?
2?
1
2?
2?

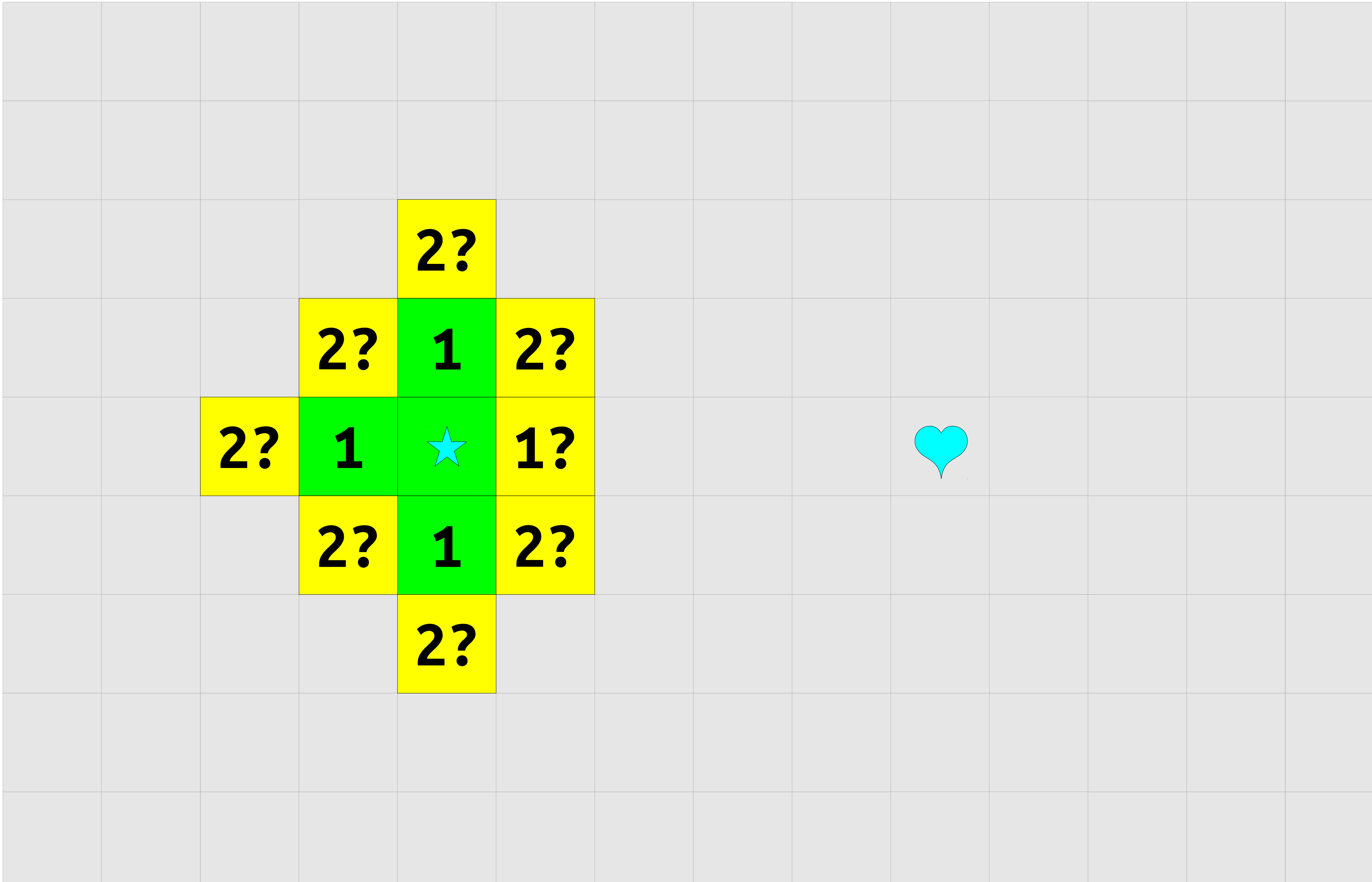

1?
1?
1?
2?
1
2?
2?

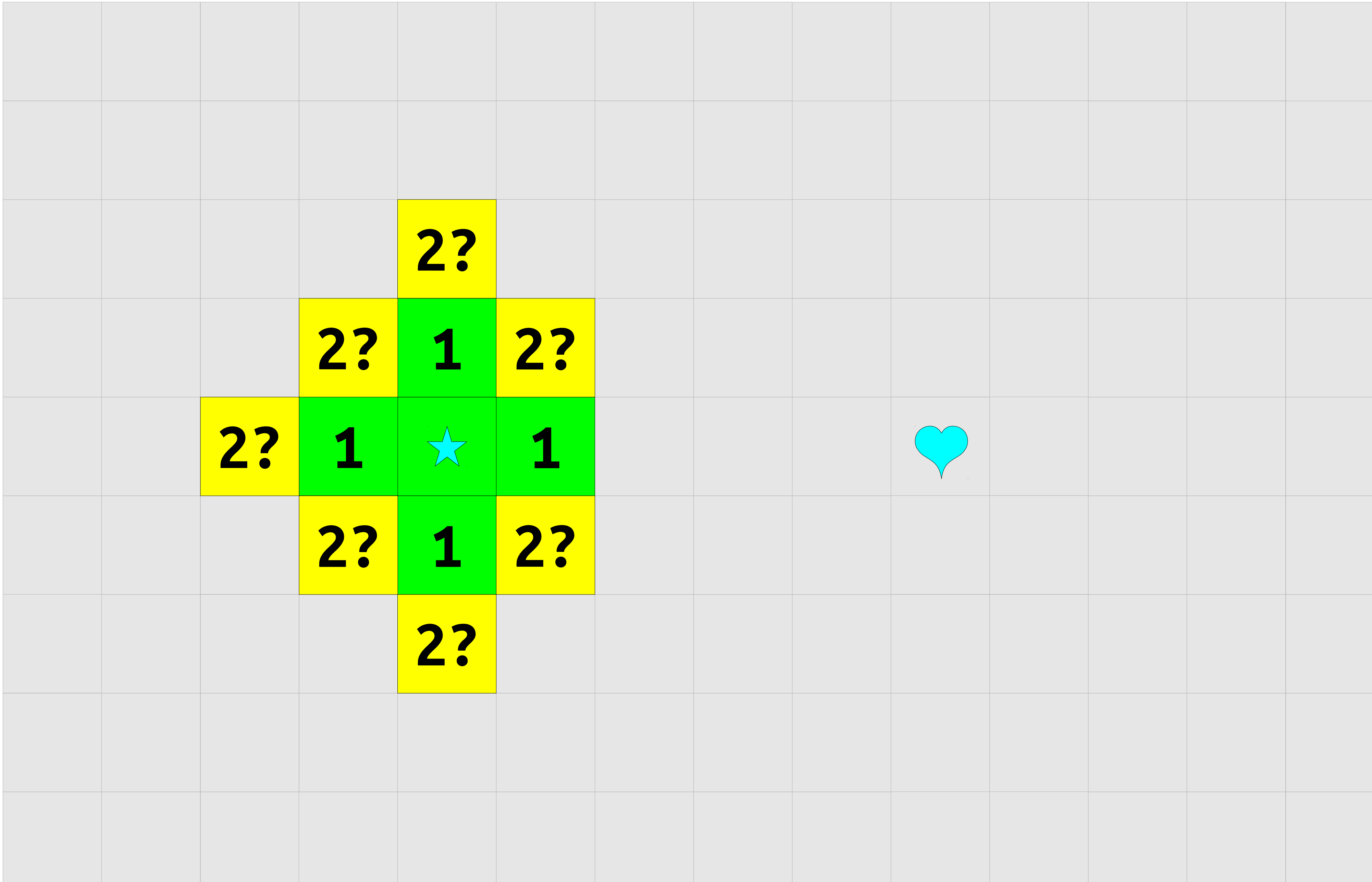

1?
1
1?
2?
1
2?
2?

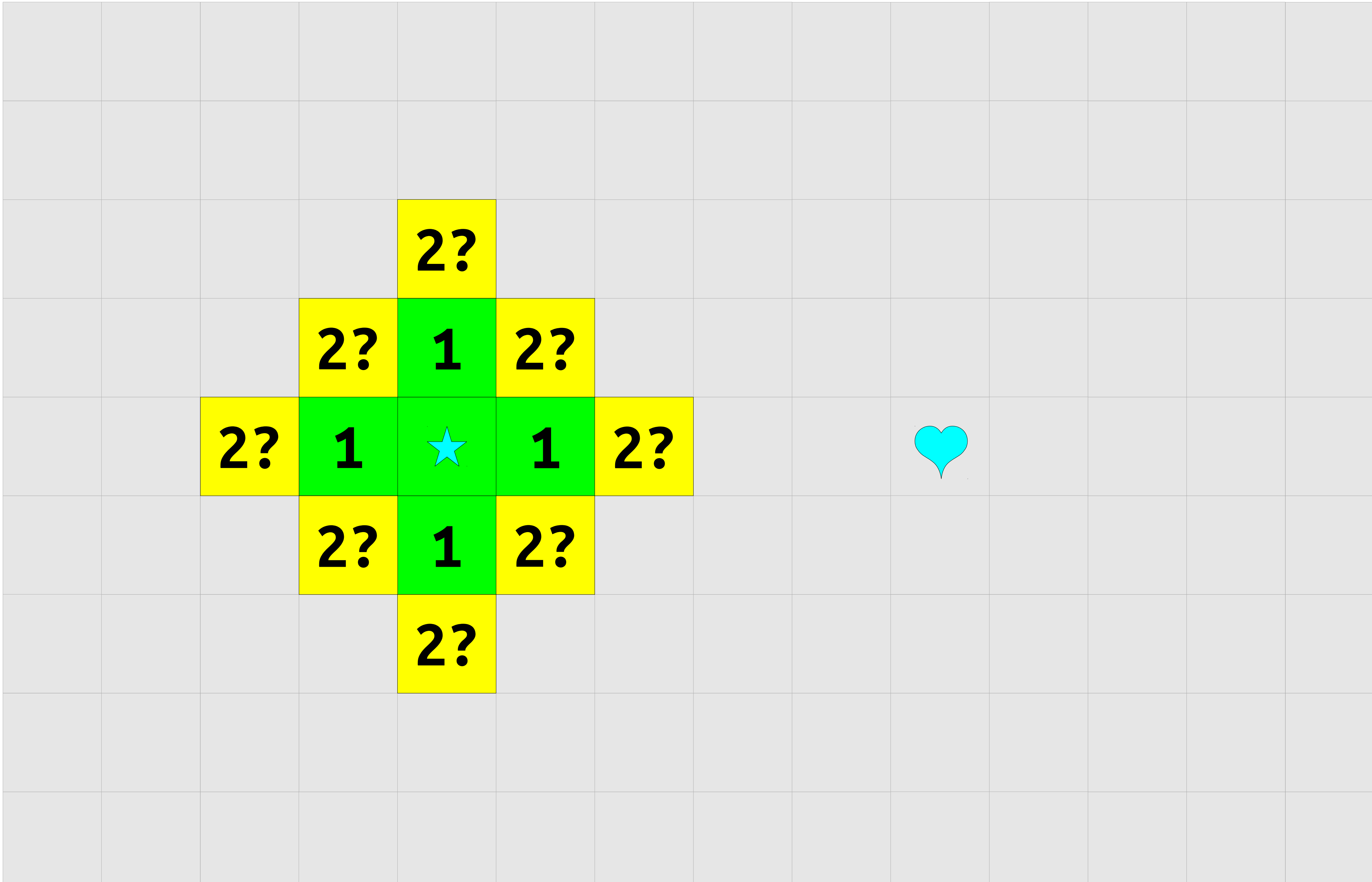

2?
1?
2?
1
1?
2?
1
2?
2?

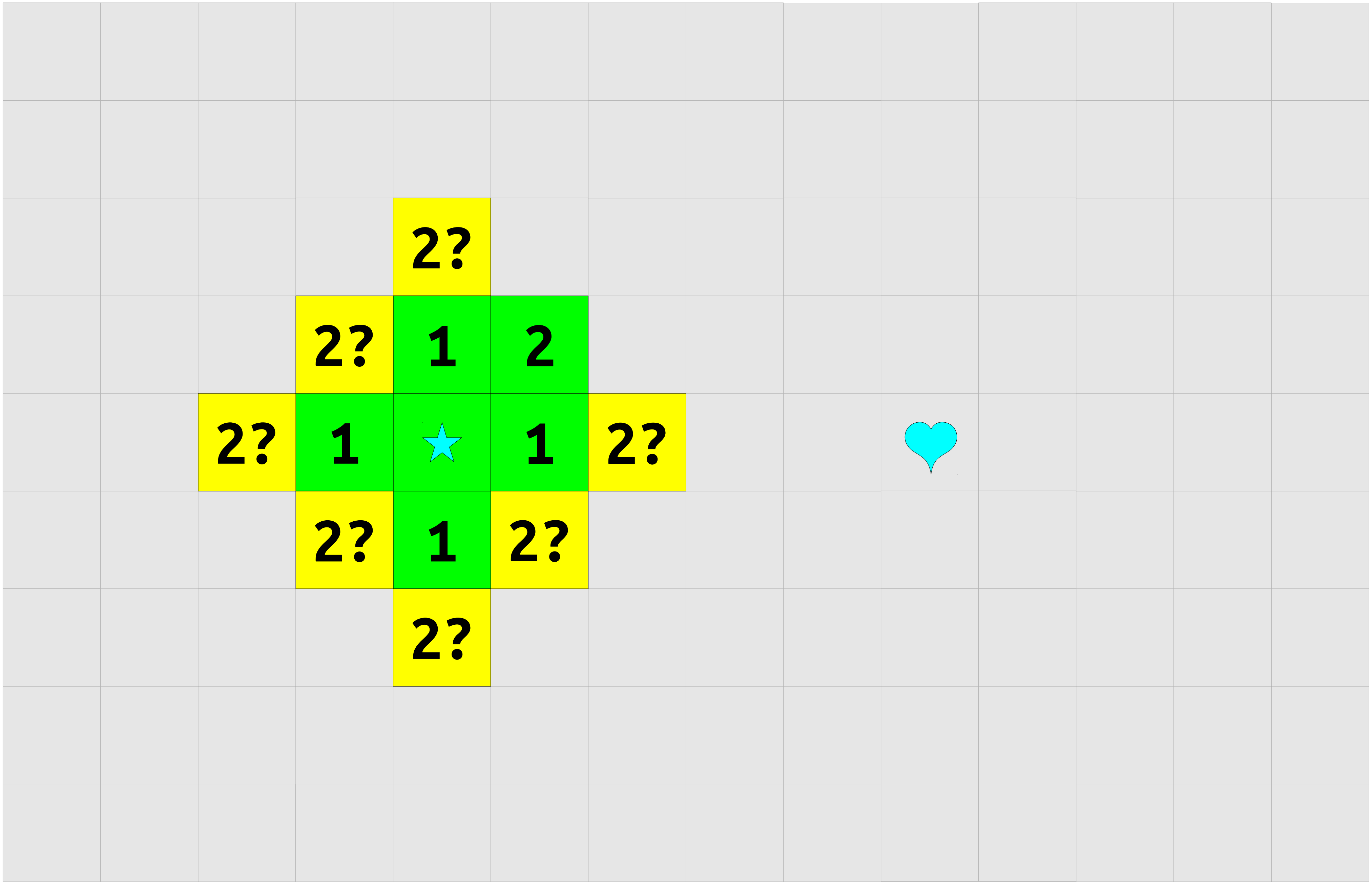

2?
1?
2?
1
1?
2?
1
2?
2?

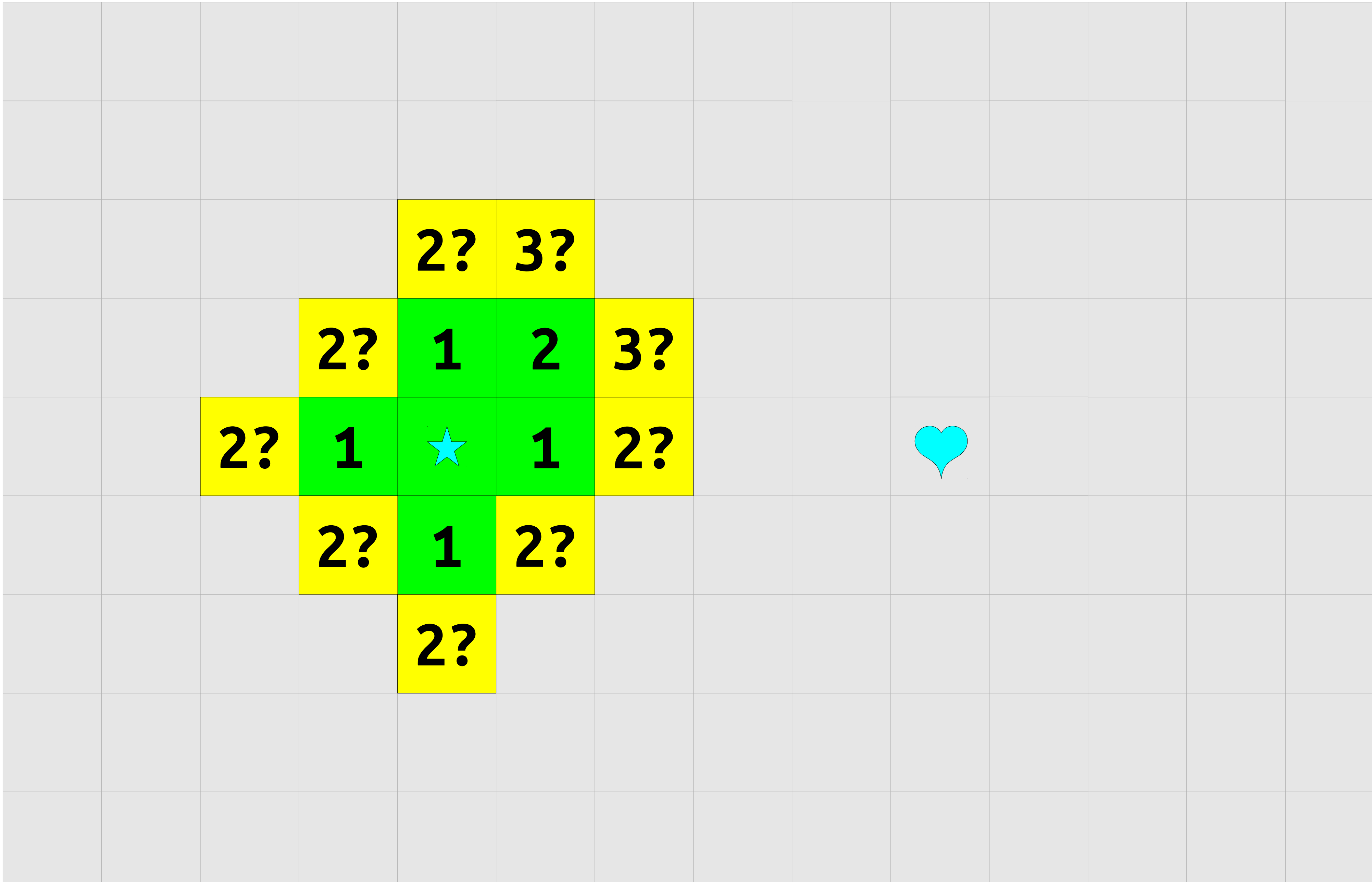

2?
1
2?
1
1?
2?
1
2?
2?

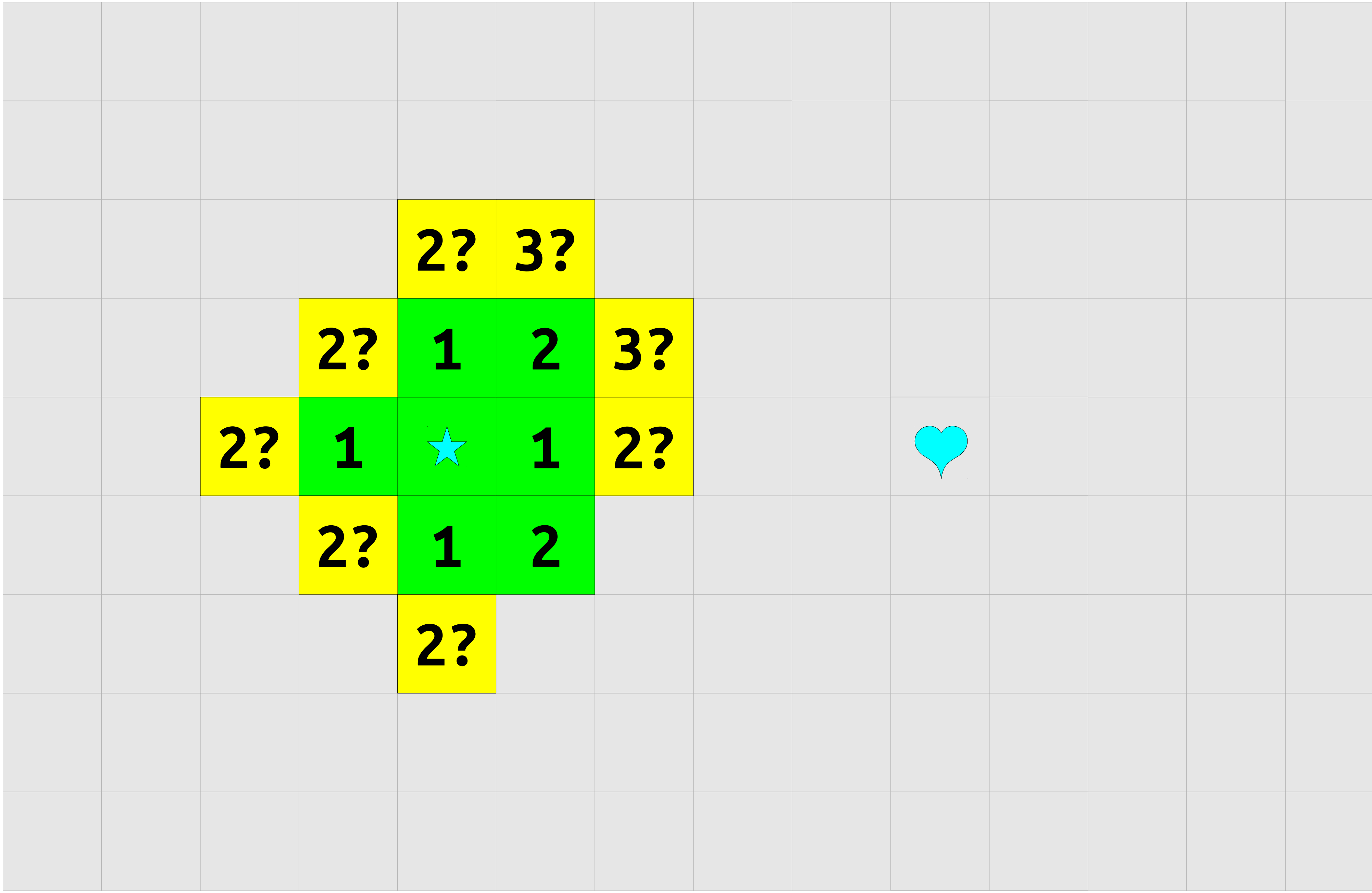

2?
2?
1
2?
2?
1
1?
2?
1
2?
2?

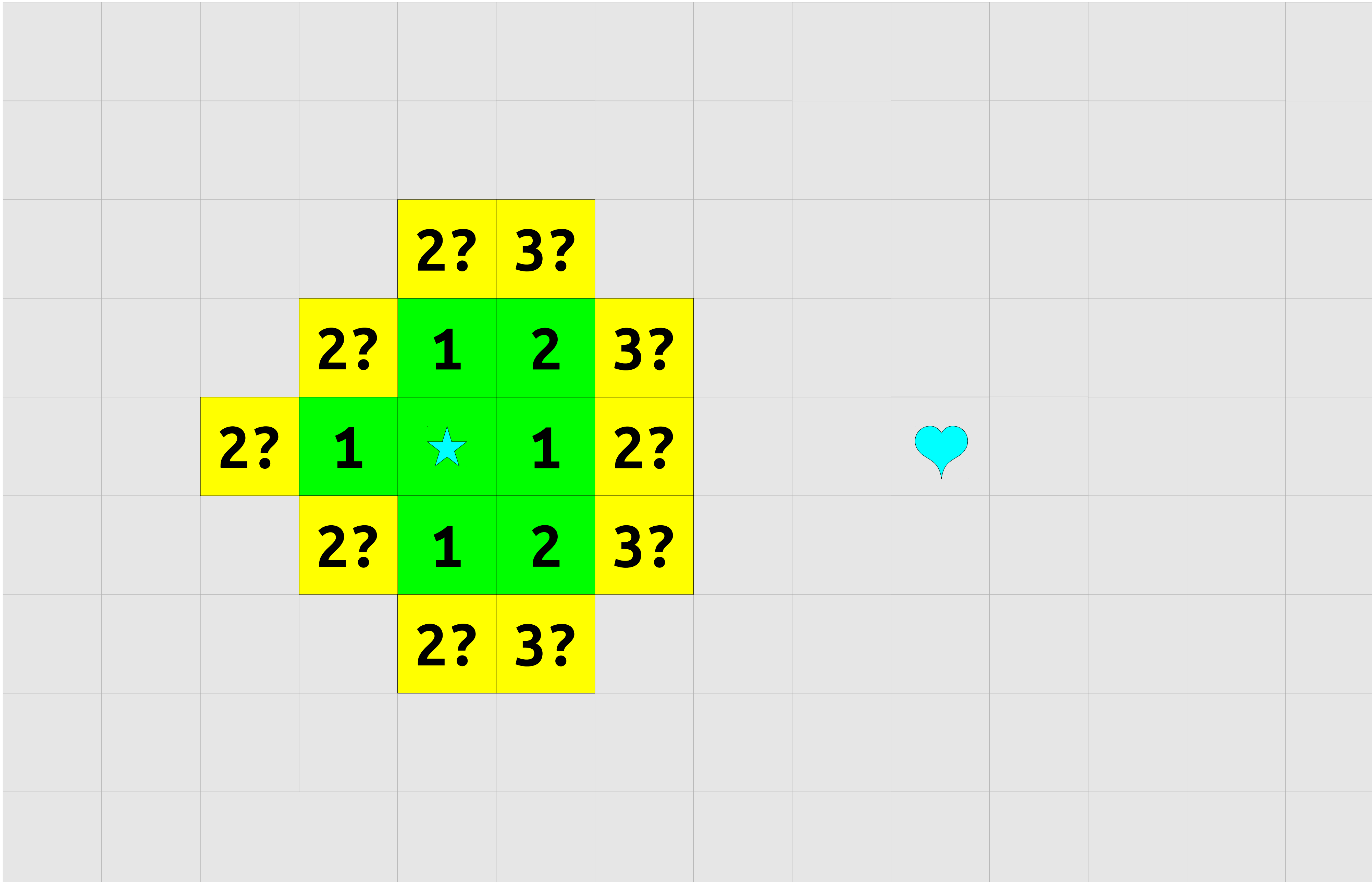

2?
2?
1
2?
2?
1
1
2?
1
2?
2?

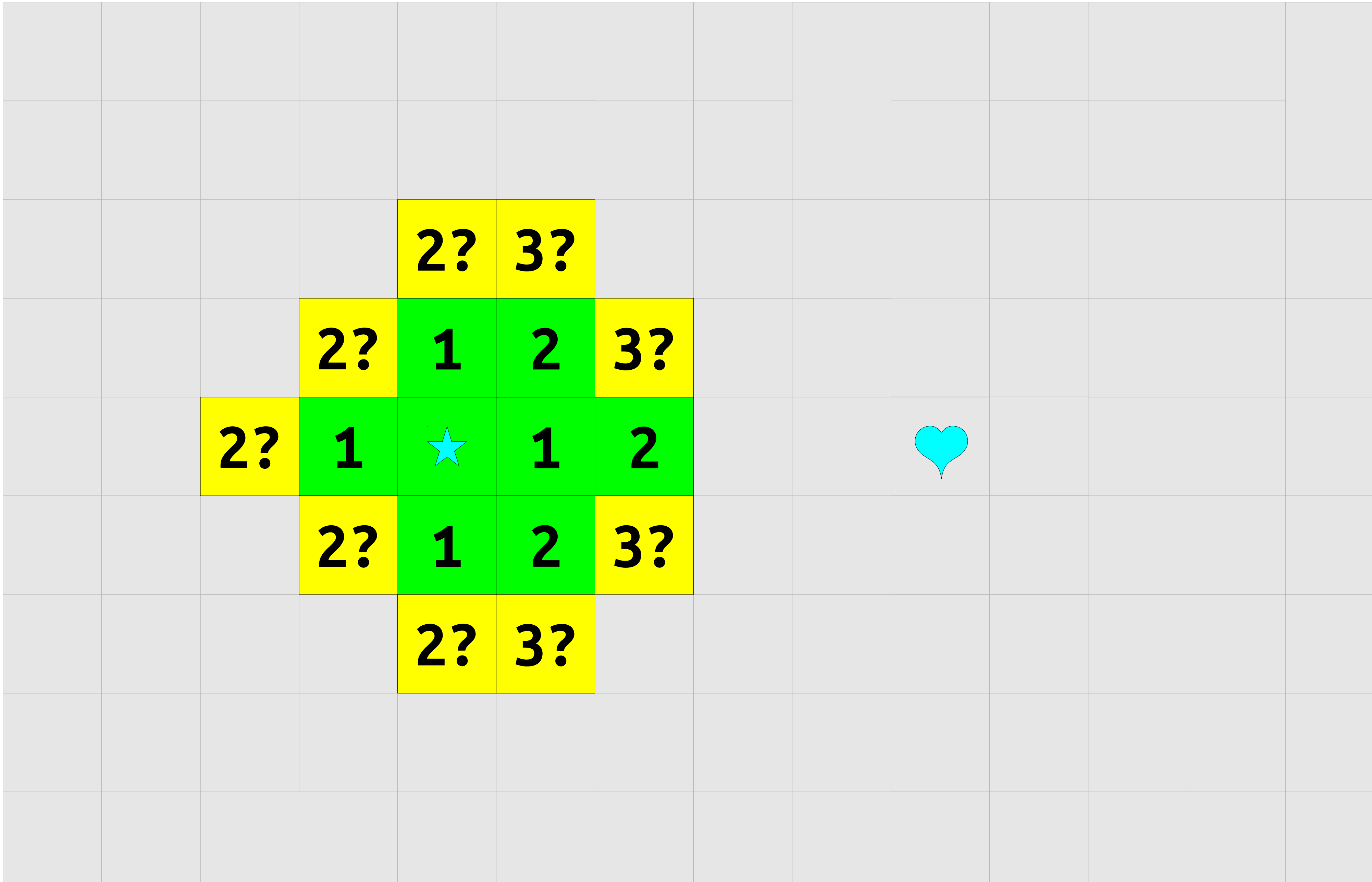

2?
2?
1
2?
2?
1
1
2?
2?
1
2?
2?

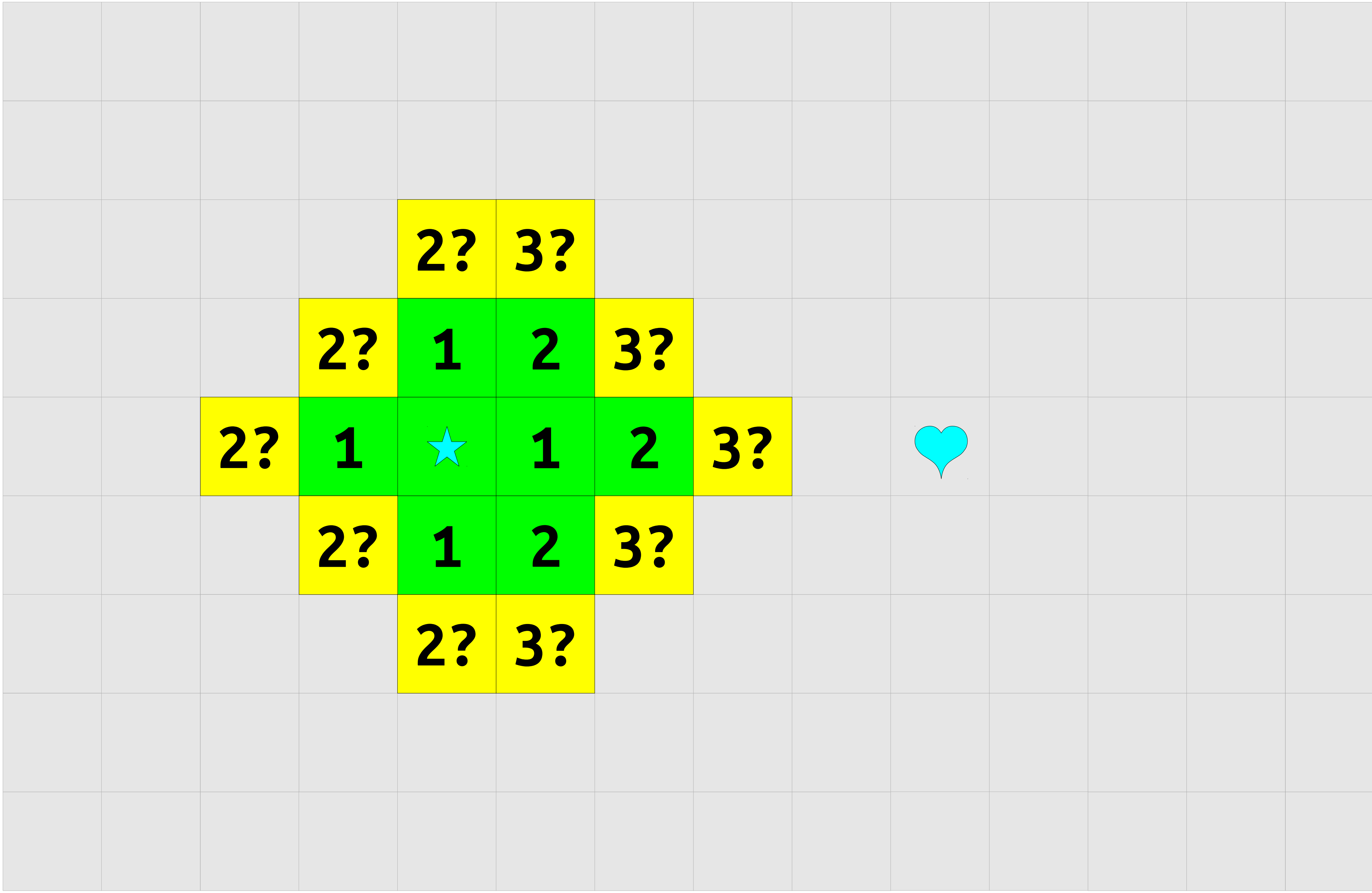

2?
2?
1
2
2?
1
1
2?
2?
1
2?
2?

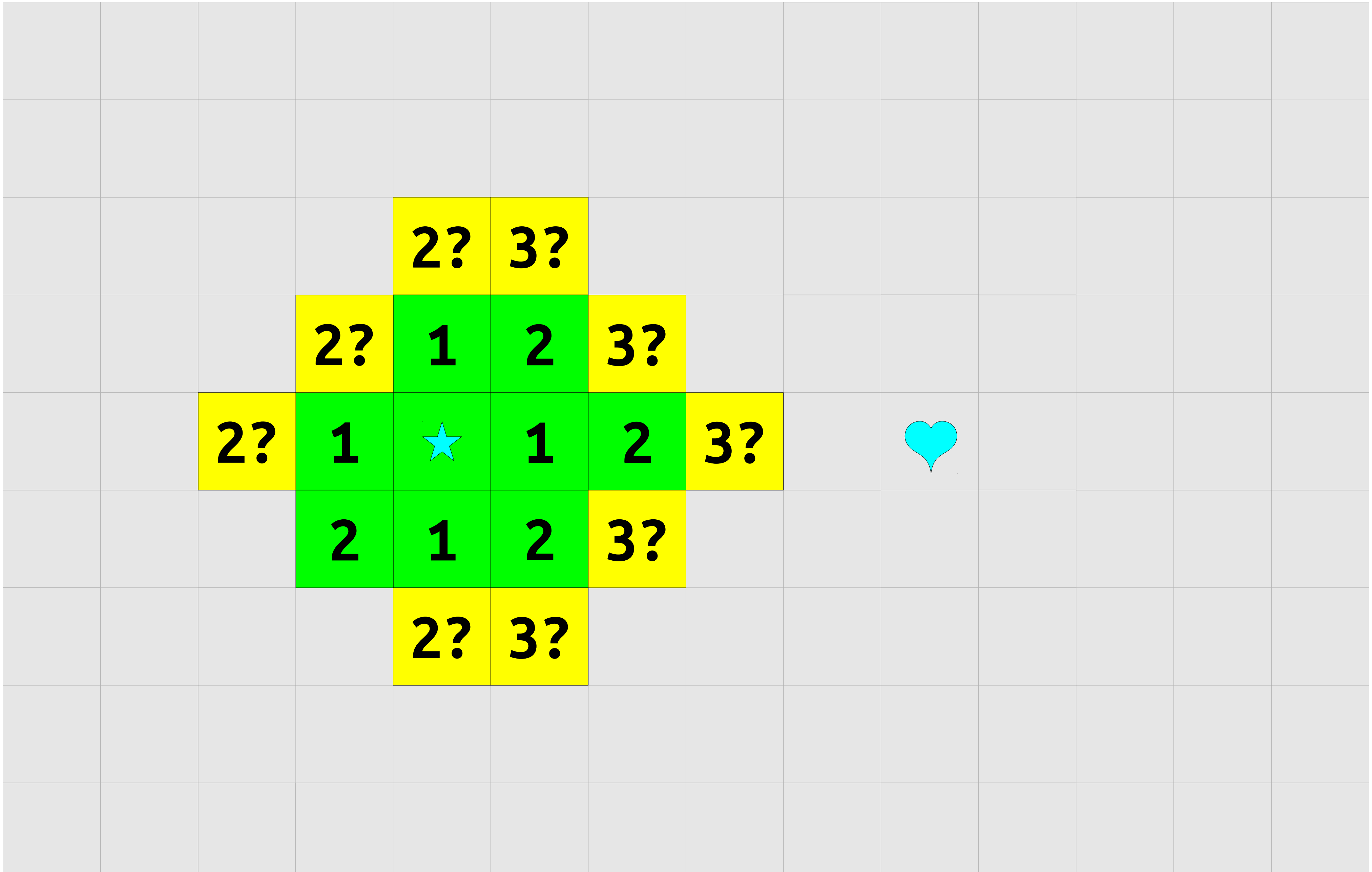

2? 3?
2? 1 2 3?
2? 1 1 2?
2? 1 2?
2?

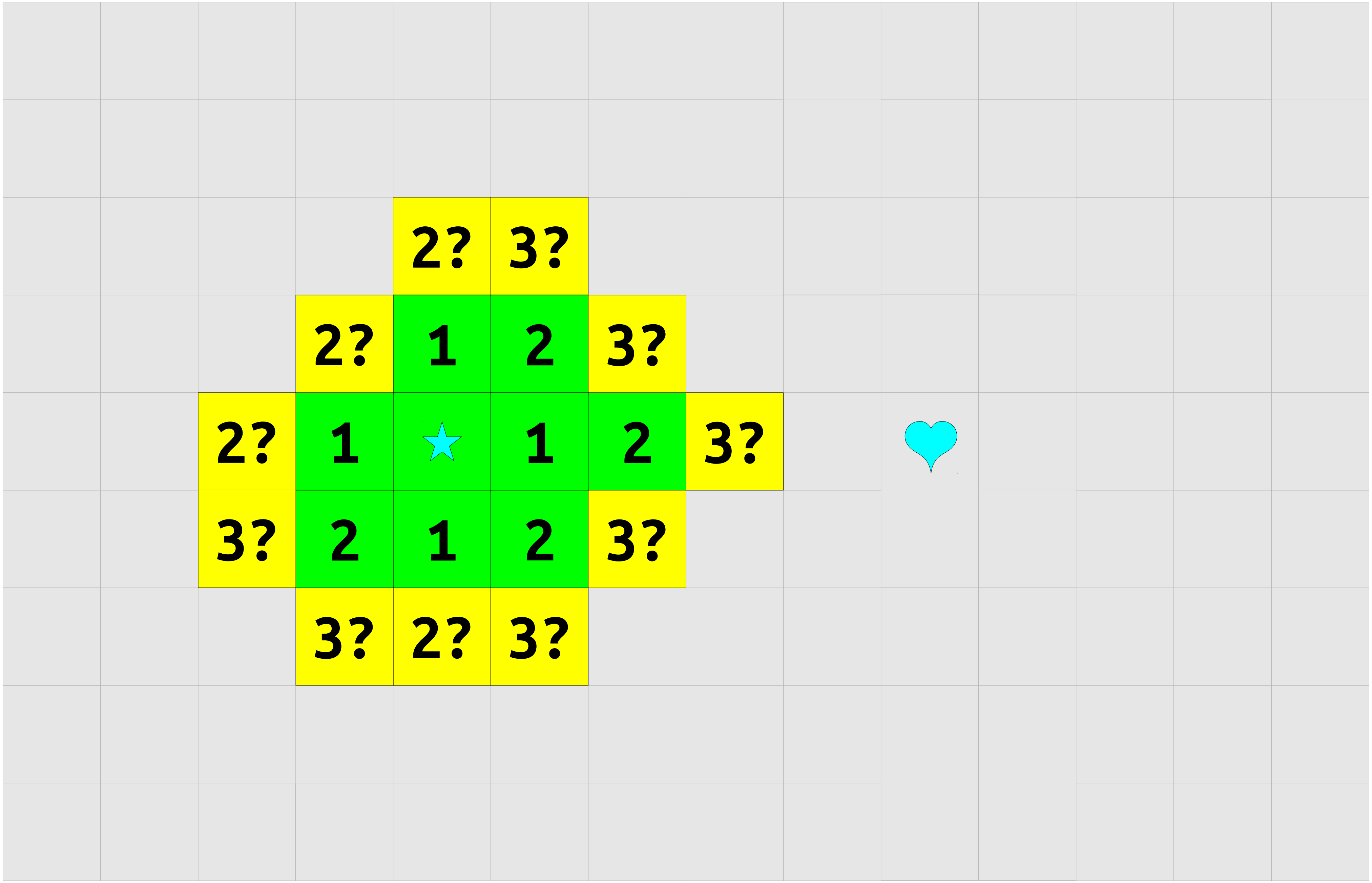

2?
3?
2?
1
2
3?
2?
1
1
2?
2?
1
2
2?

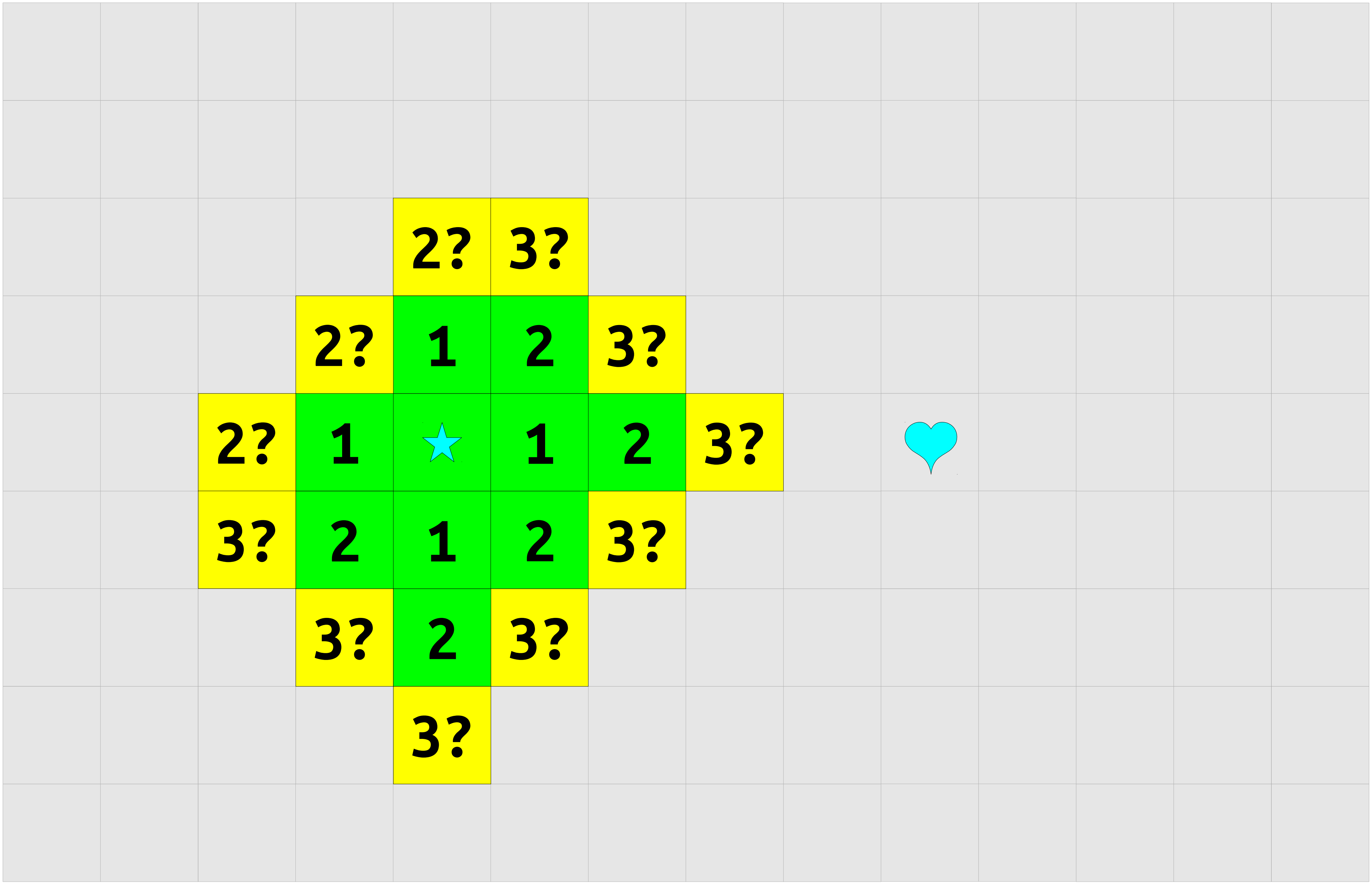

2? 3?
2? 1 2 3?
2? 1 1 2?
2? 1 2 3?
2? 3?

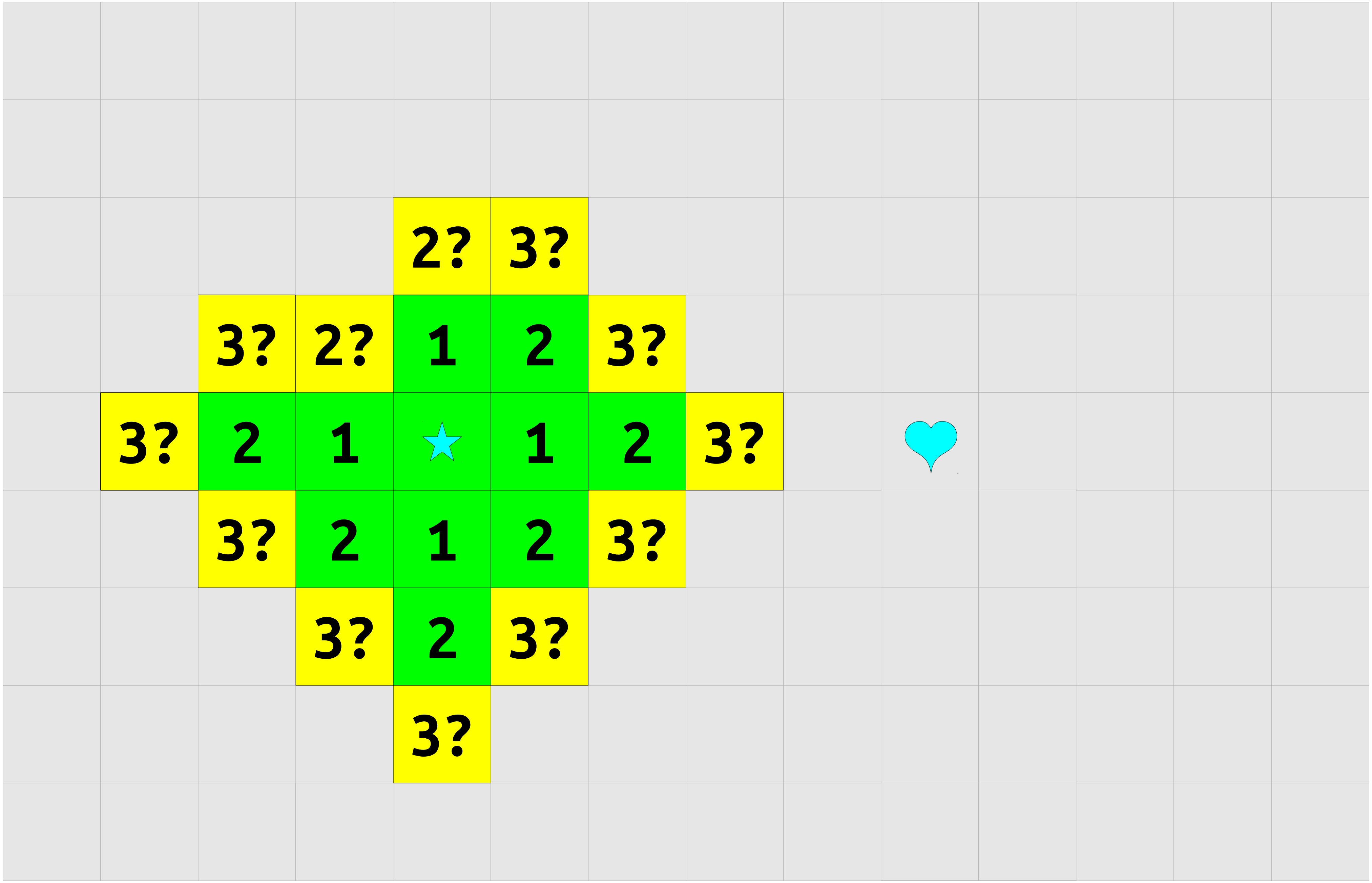

2?
3?
2?
1
2
3?
2?
1
1
2
2?
1
2
3?
2?
3?

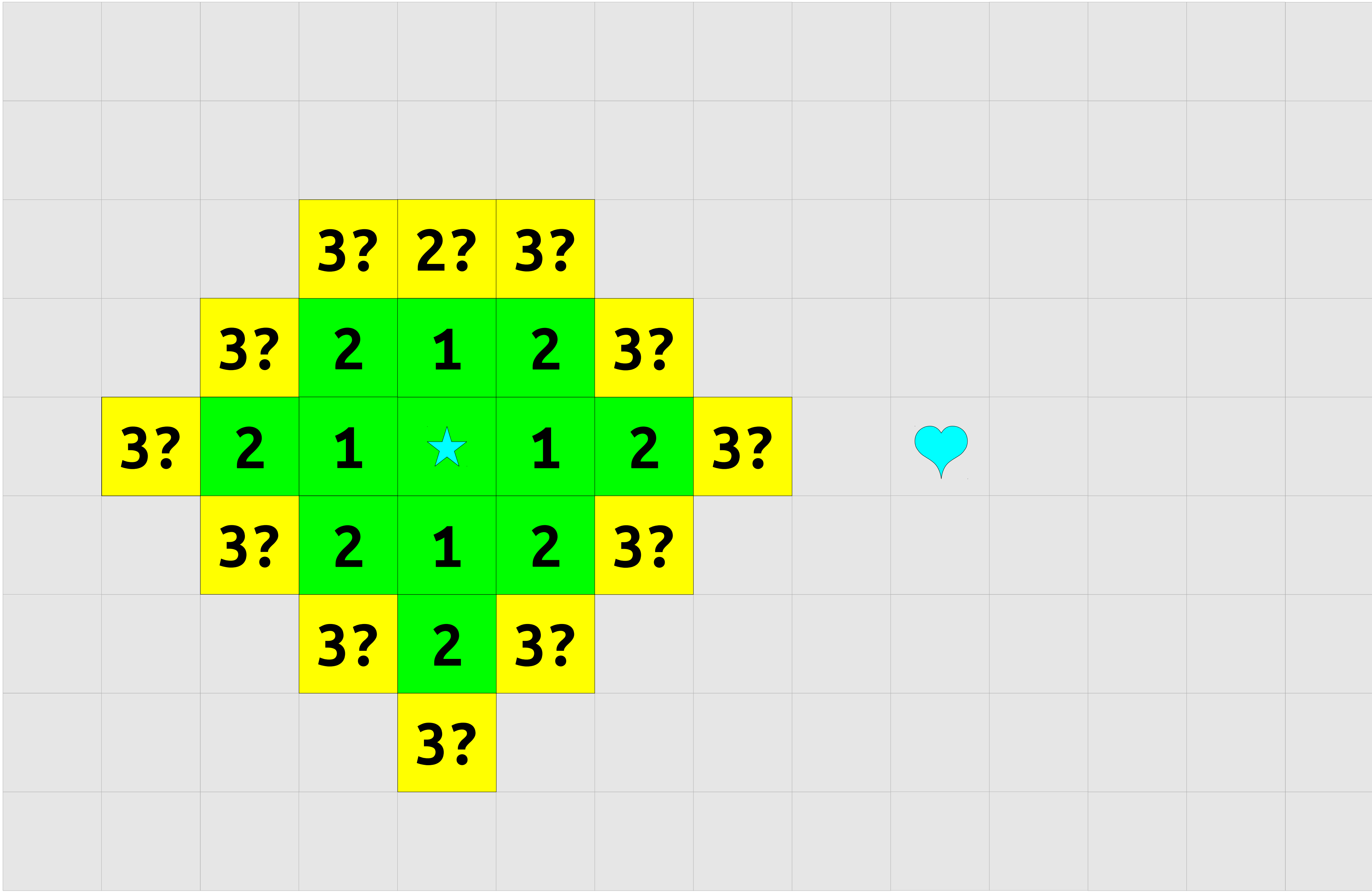

2? 3?
2? 1 2 3?
2? 1 1 2 3?
2? 1 2 3?
2? 3?

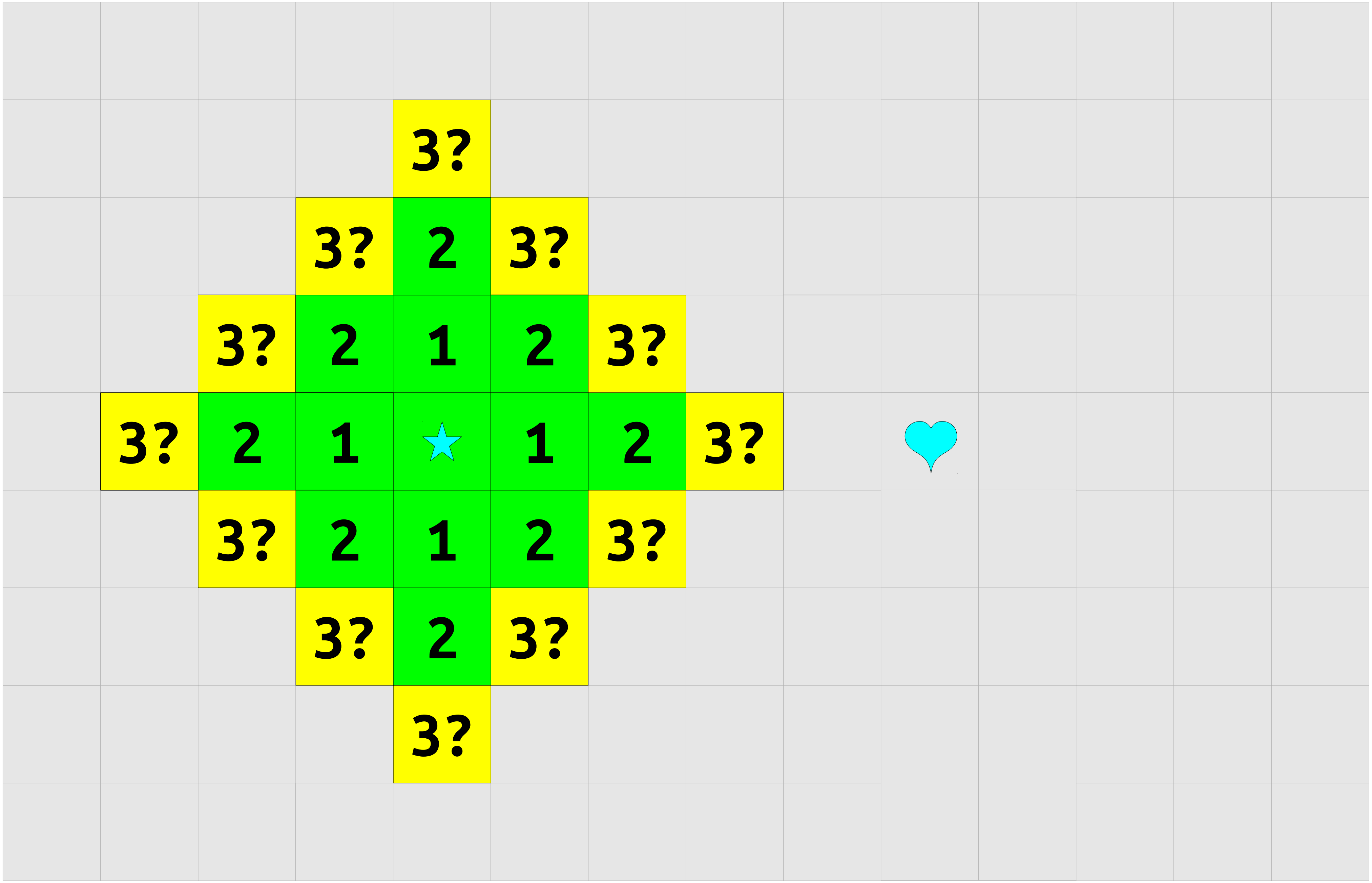

2? 3?
2? 1 2 3?
2? 1 1 2 3?
2 1 2 3?
2? 3?

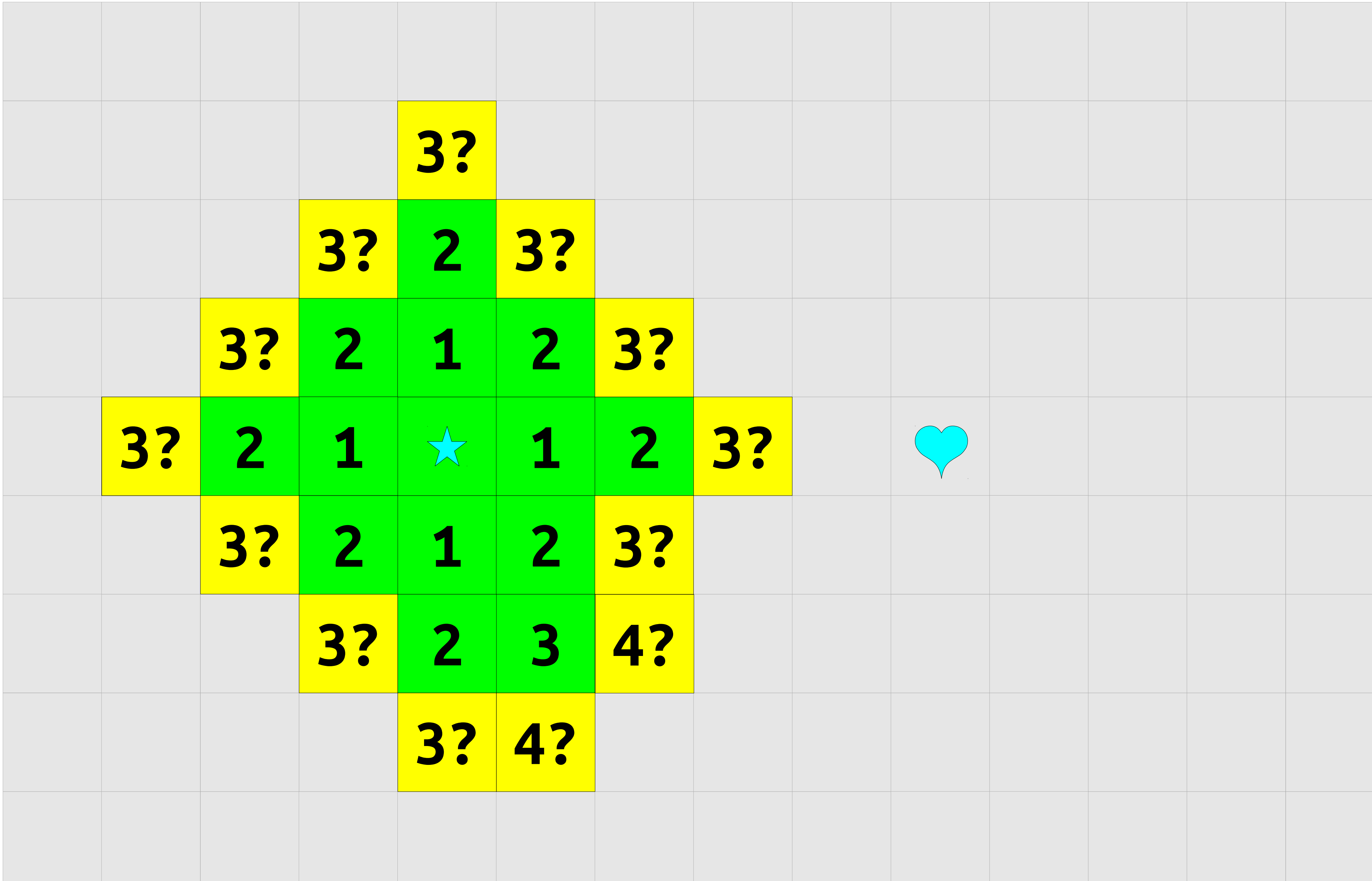

2?
3?
2?
1
2
3?
2?
1
1
2
3?
3?
2
1
2
3?
3?
2?
3?

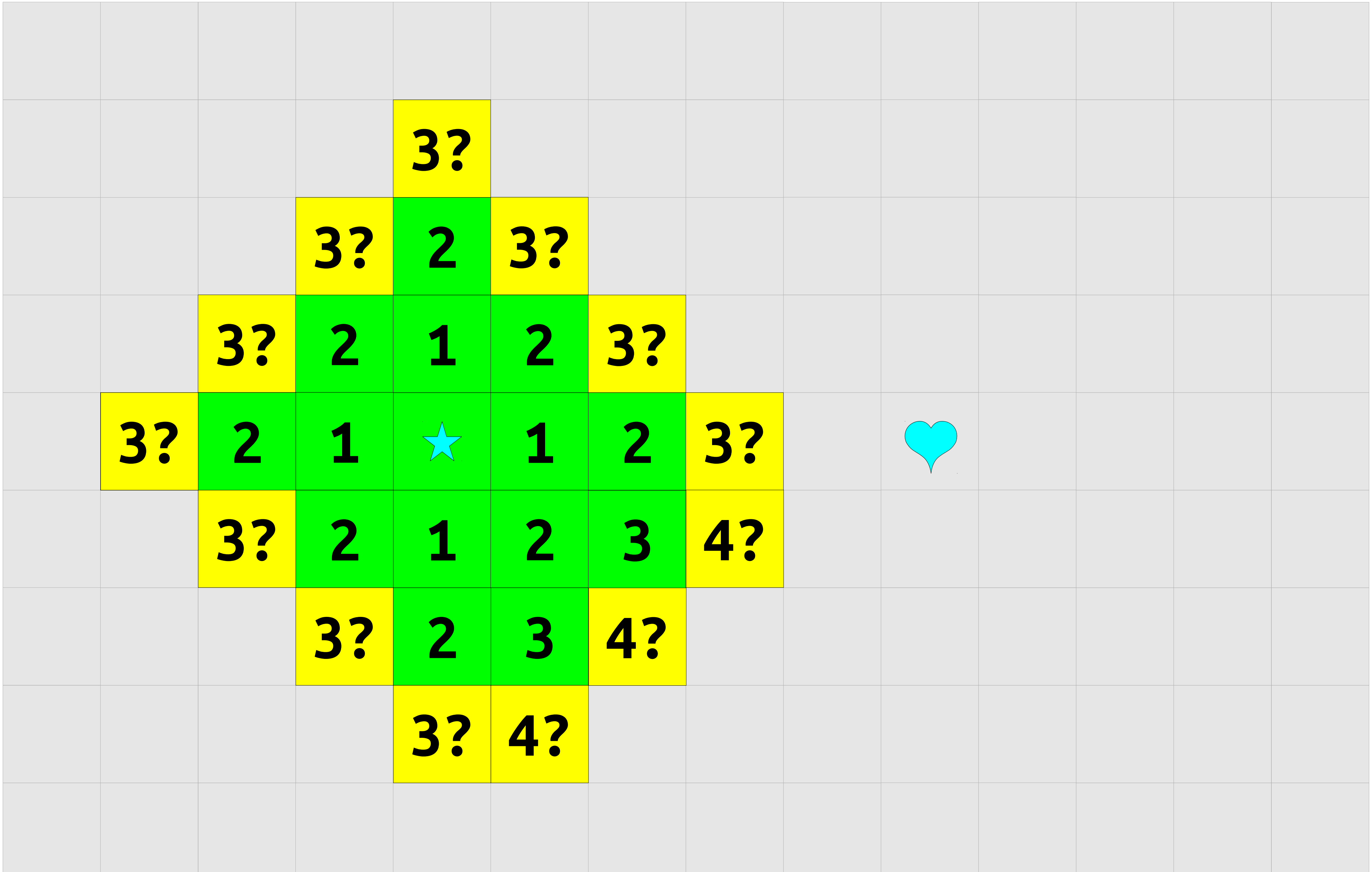

2? 3?
2? 1 2 3?
2? 1 1 2 3?
3? 2 1 2 3?
3? 2 3?
3?


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

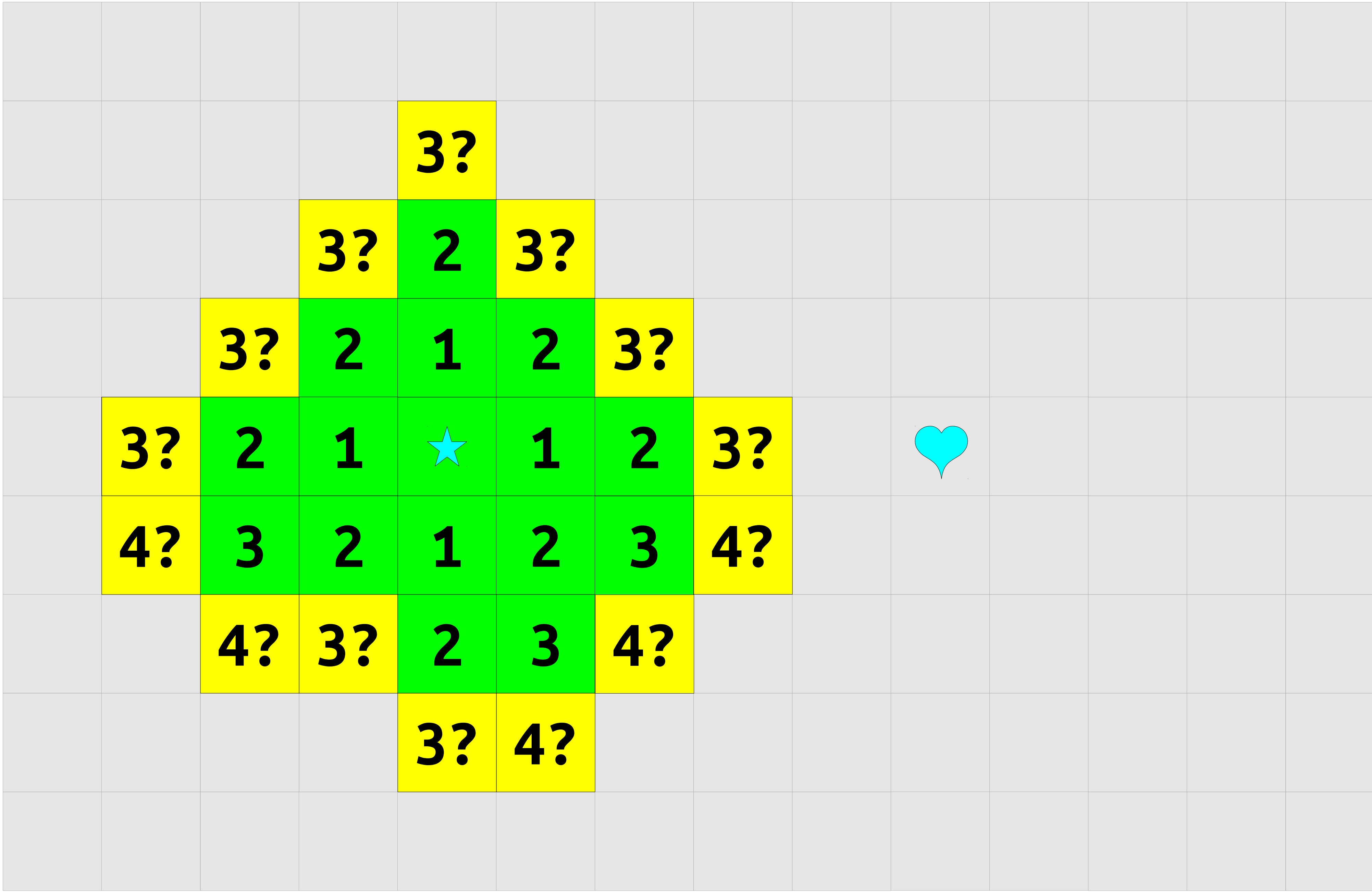

3? 2? 3?
3? 2 1 2 3?
3? 2 1 1 2 3?
3? 2 1 2 3?
3? 2 3?
3?

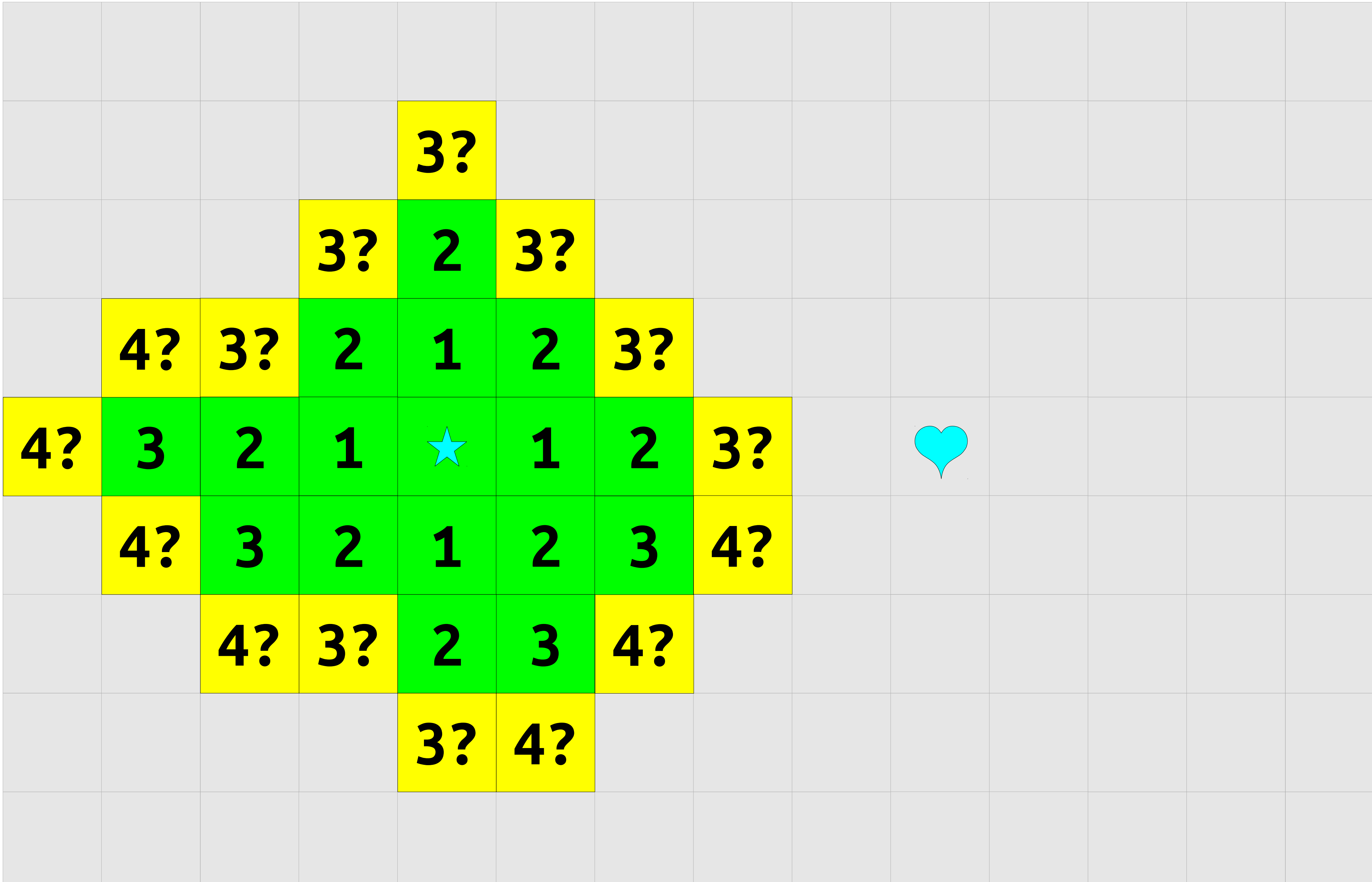

3?
3?
2
3?
3?
2
1
2
3?
3?
2
1
1
2
3?
3?
2
1
2
3?
3?
2
3?
3?

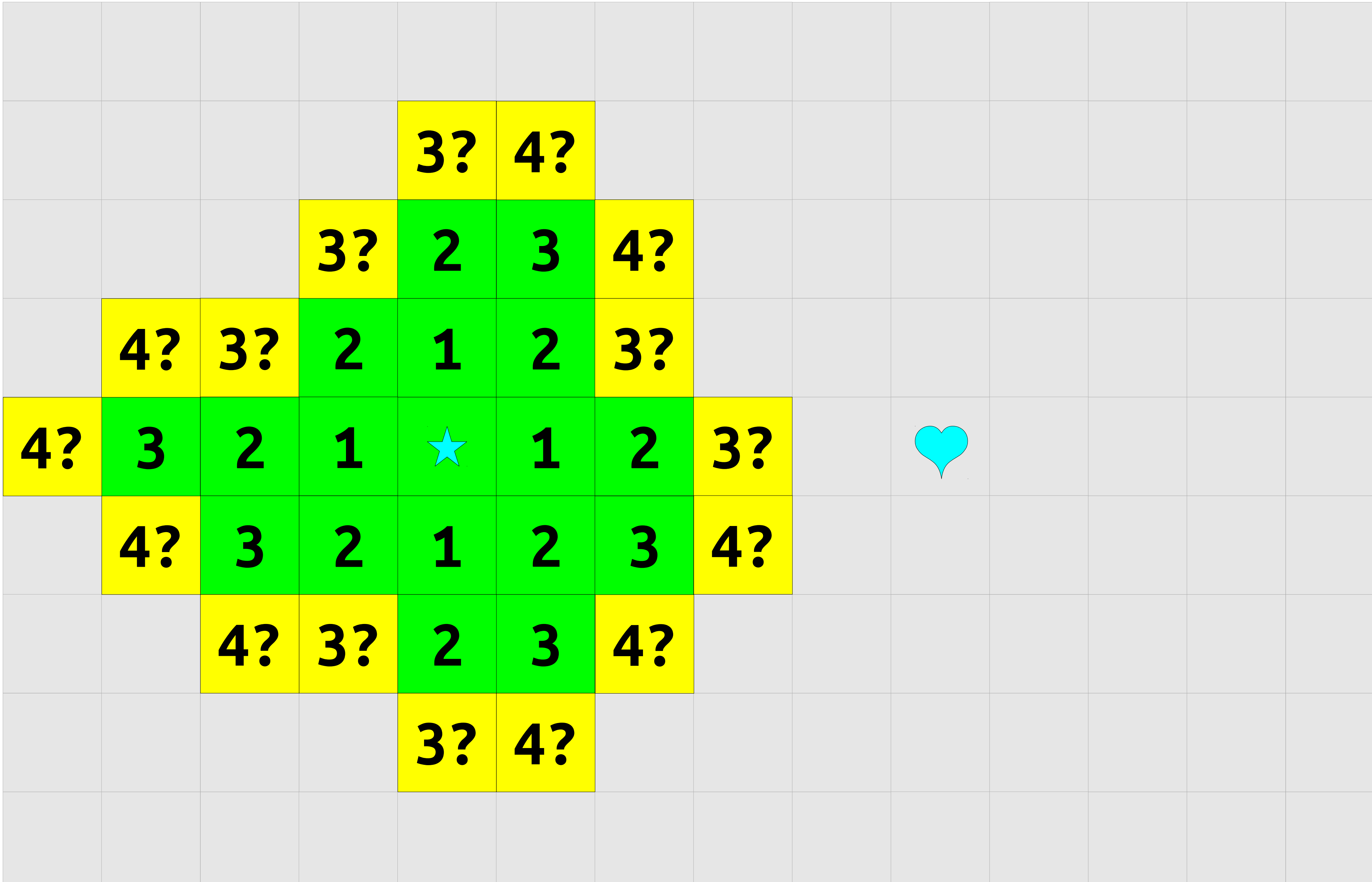

3?
3? 2 3?
3? 2 1 2 3?
3? 2 1 1 2 3?
3? 2 1 2 3?
3? 2 3 4?
3? 4?

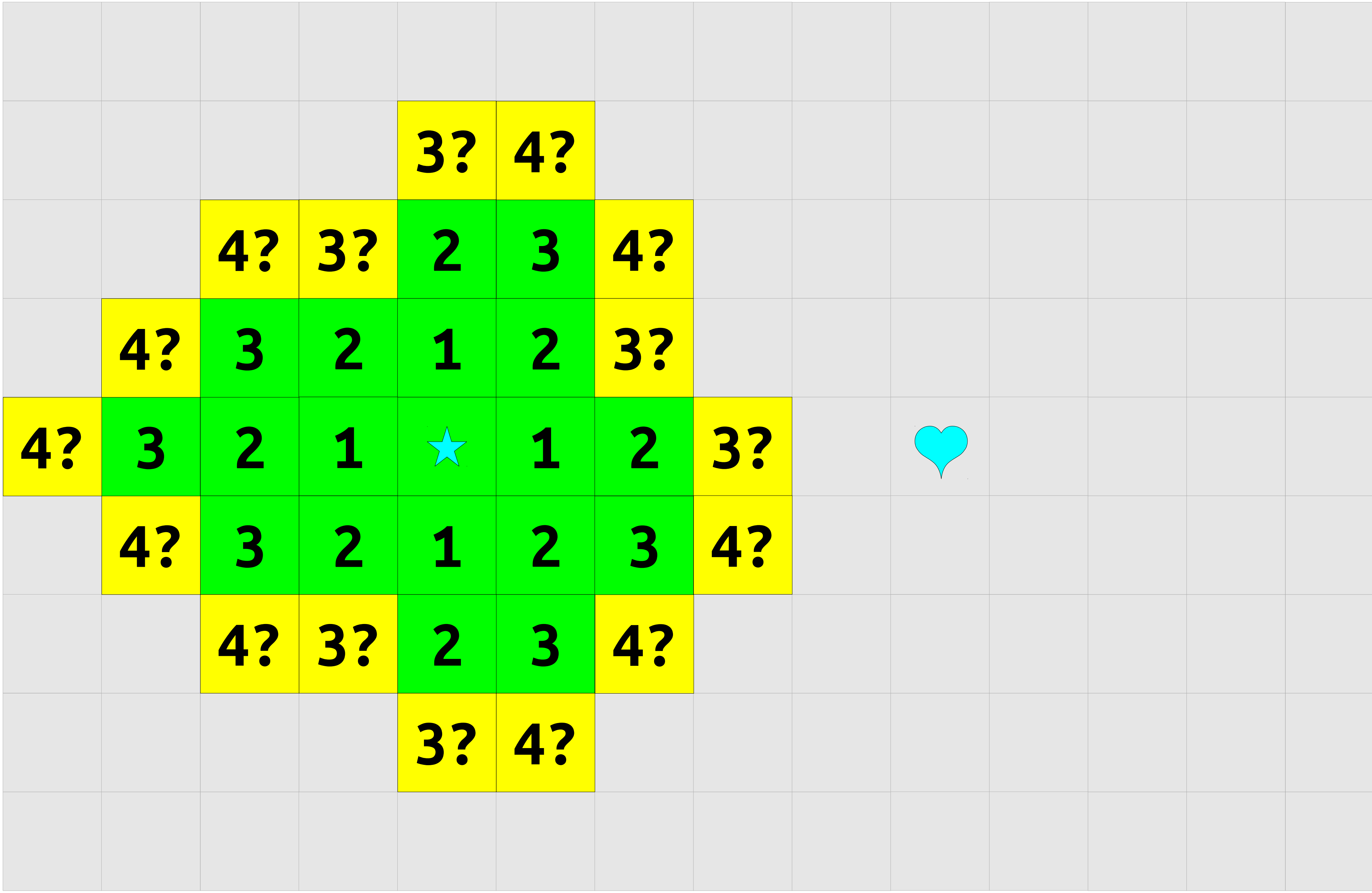


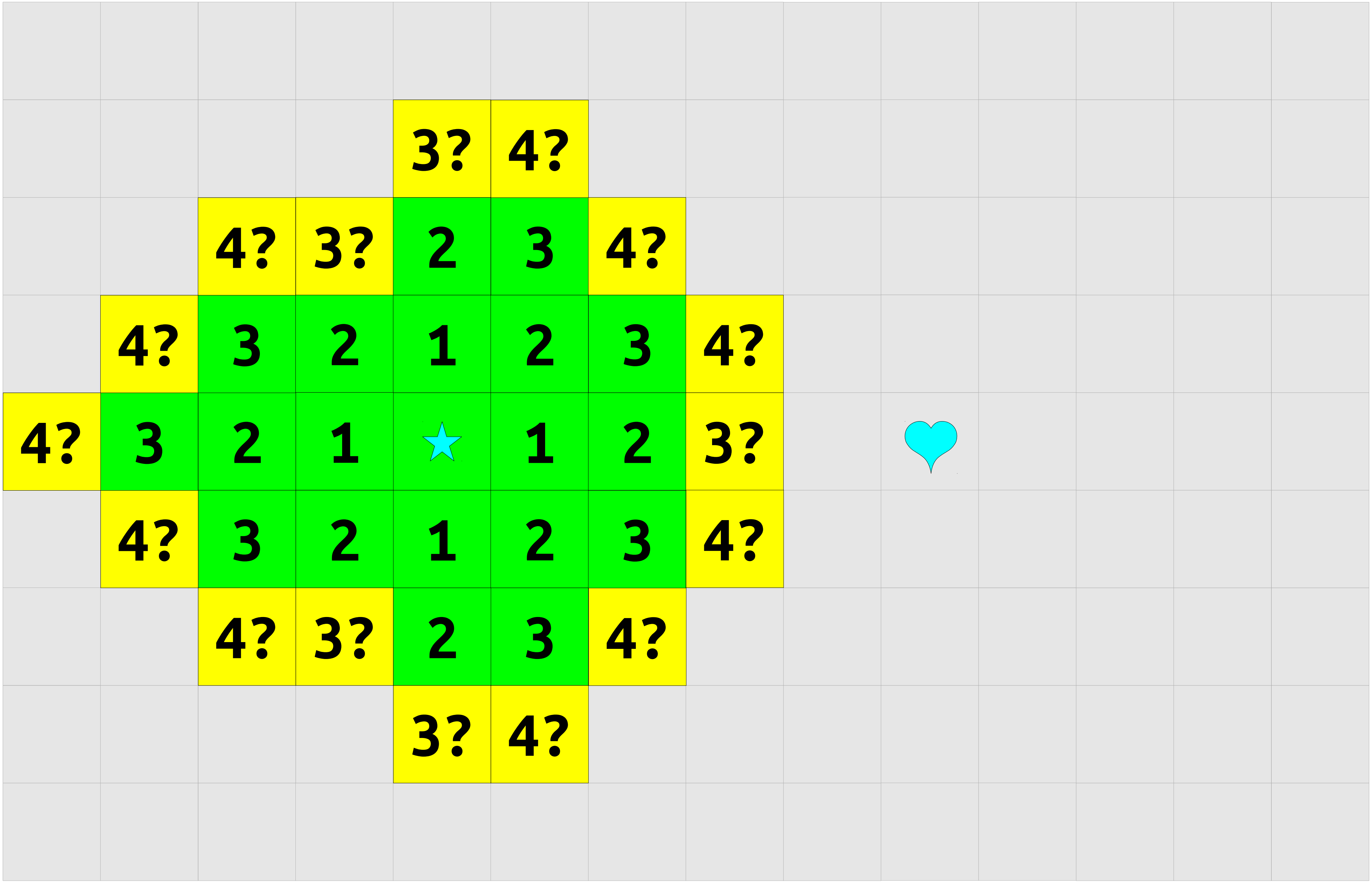
3?
3? 2 3?
3? 2 1 2 3?
3? 2 1 1 2 3?
3? 2 1 2 3 4?
3? 2 3 4?
3? 4?

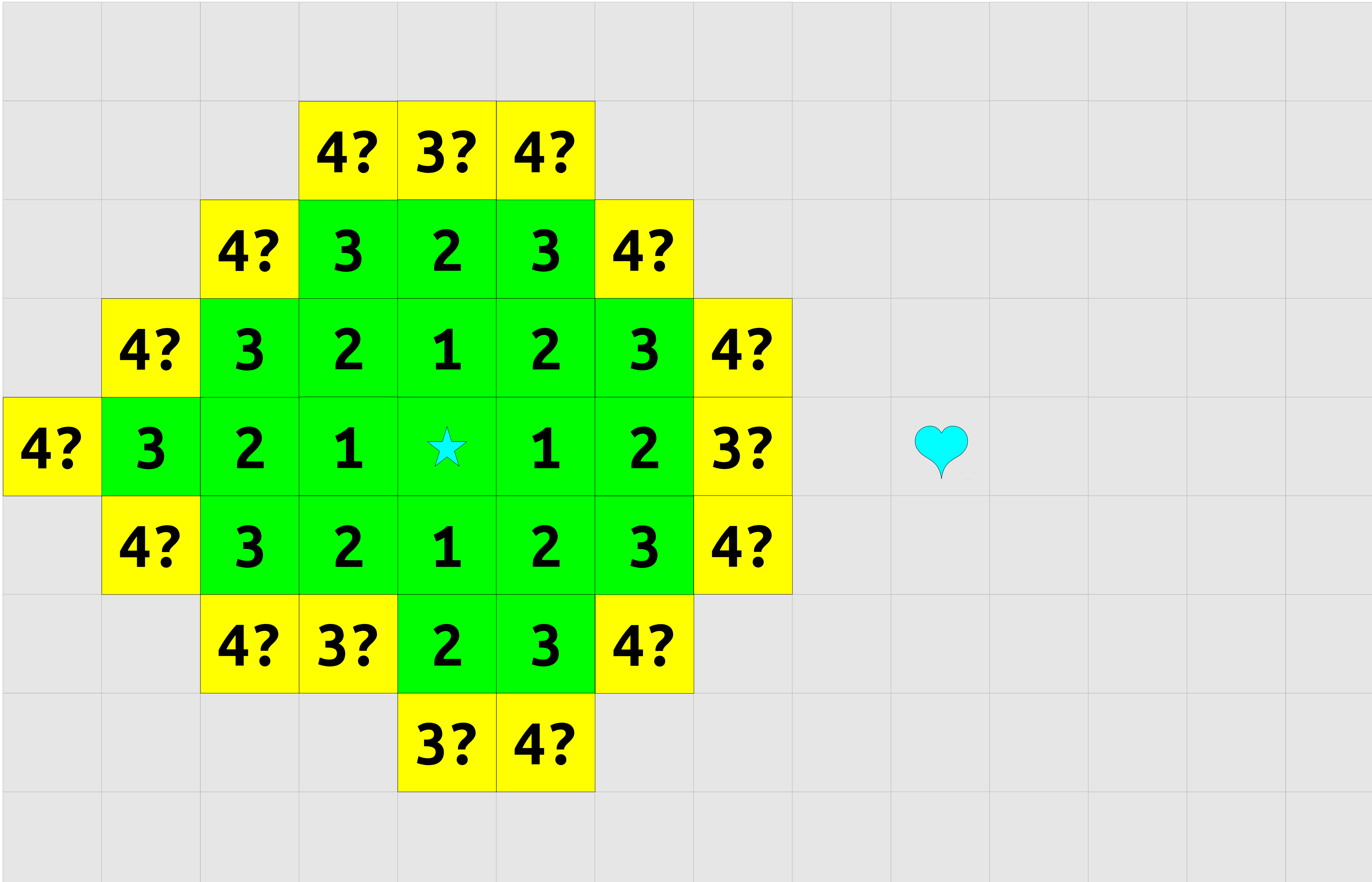

3?
3? 2 3?
3? 2 1 2 3?
3? 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

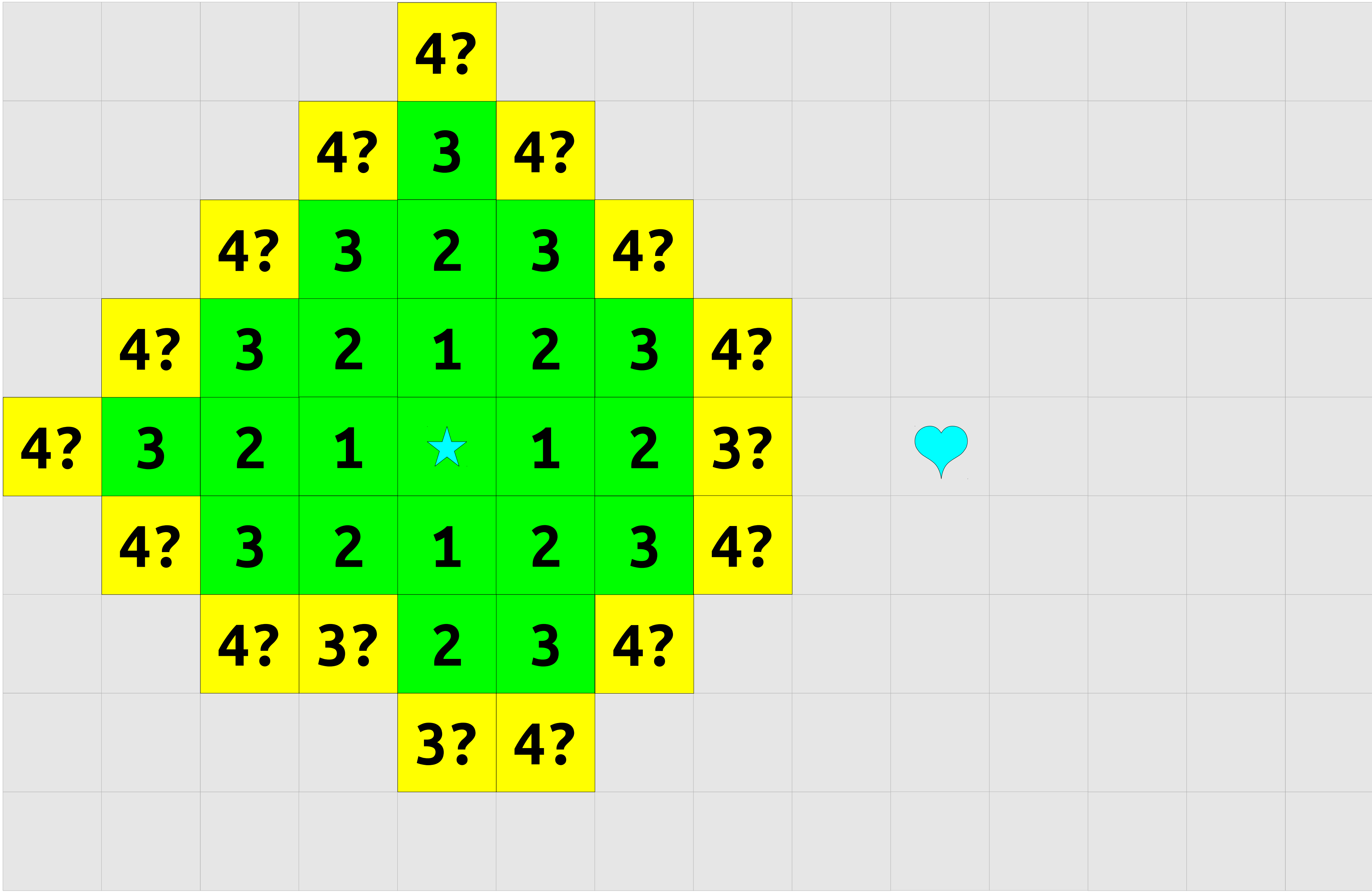

3?
3? 2 3?
4? 3? 2 1 2 3?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?


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?

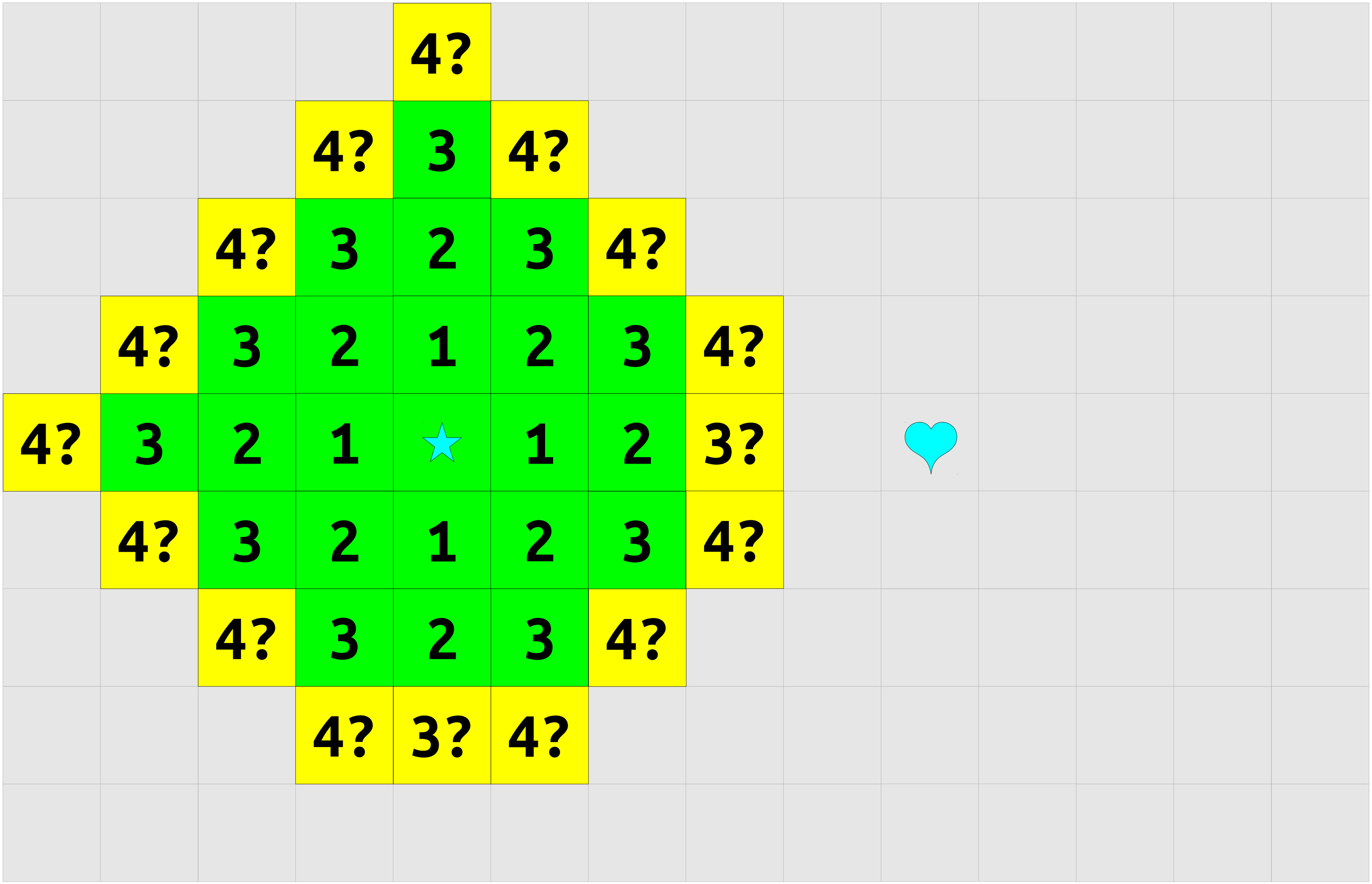

3? 4?
4? 3? 2 3 4?
4? 3 2 1 2 3?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

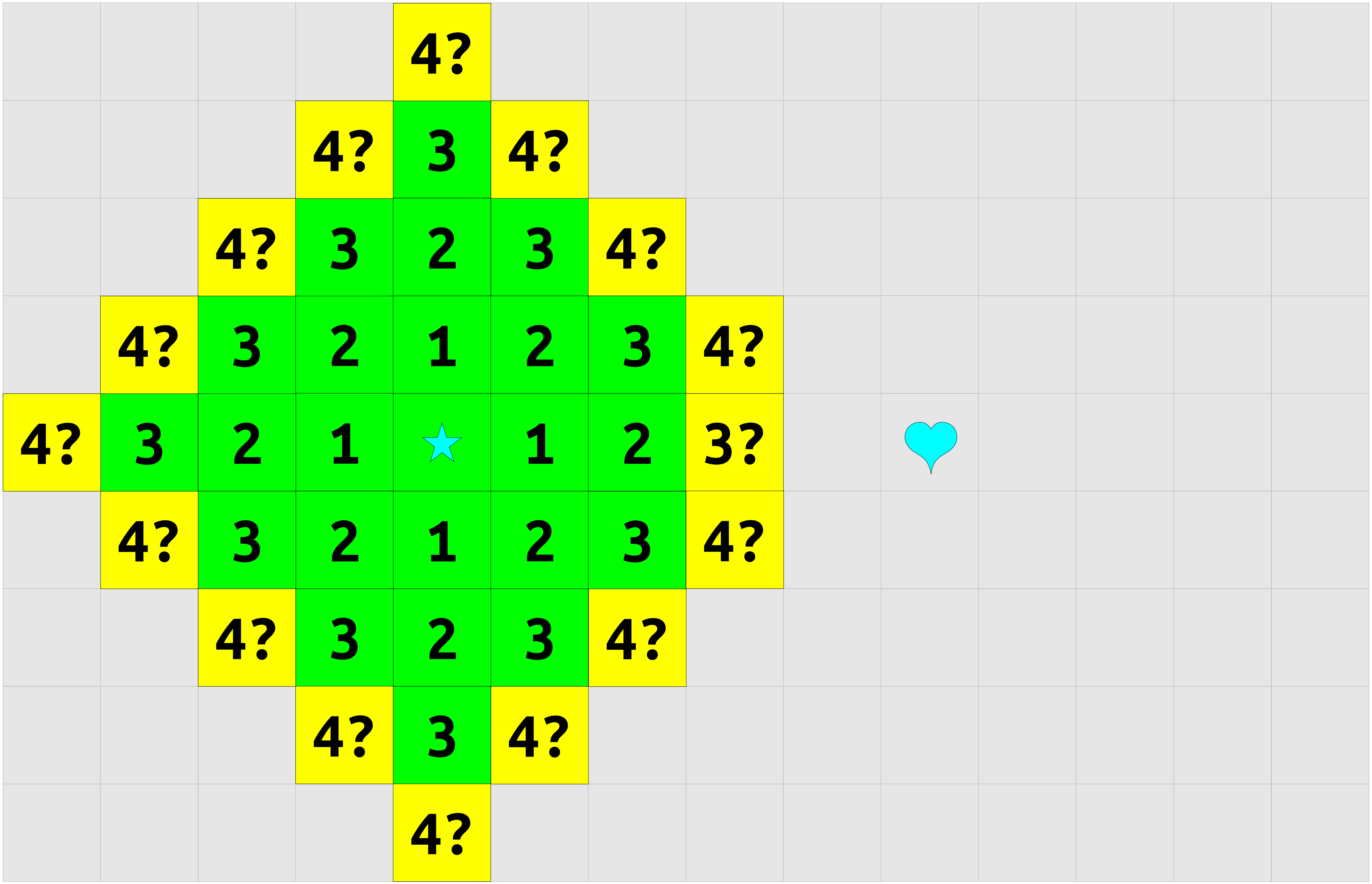

3? 4?
4? 3? 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

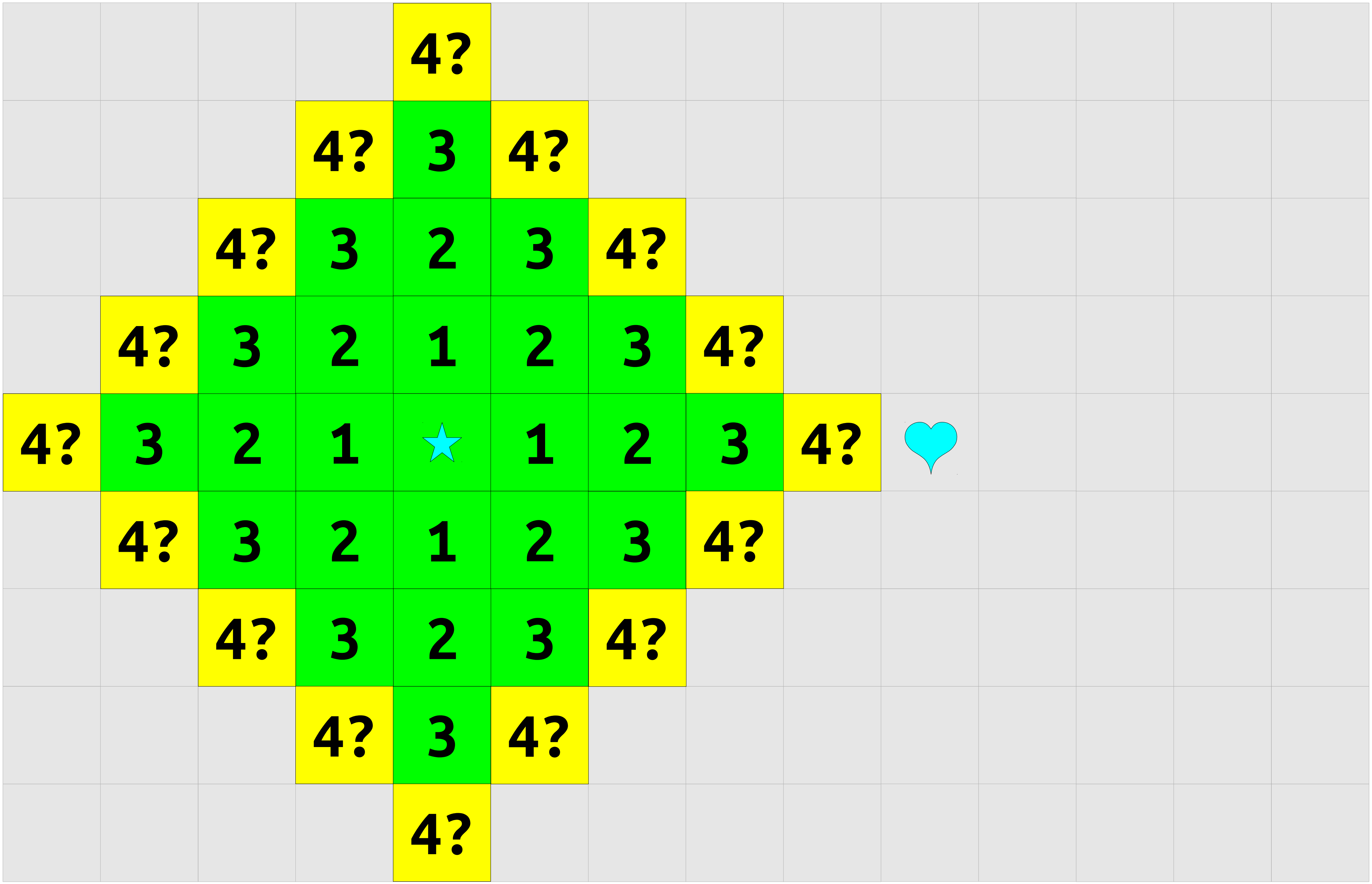

4? 3? 4?
4? 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

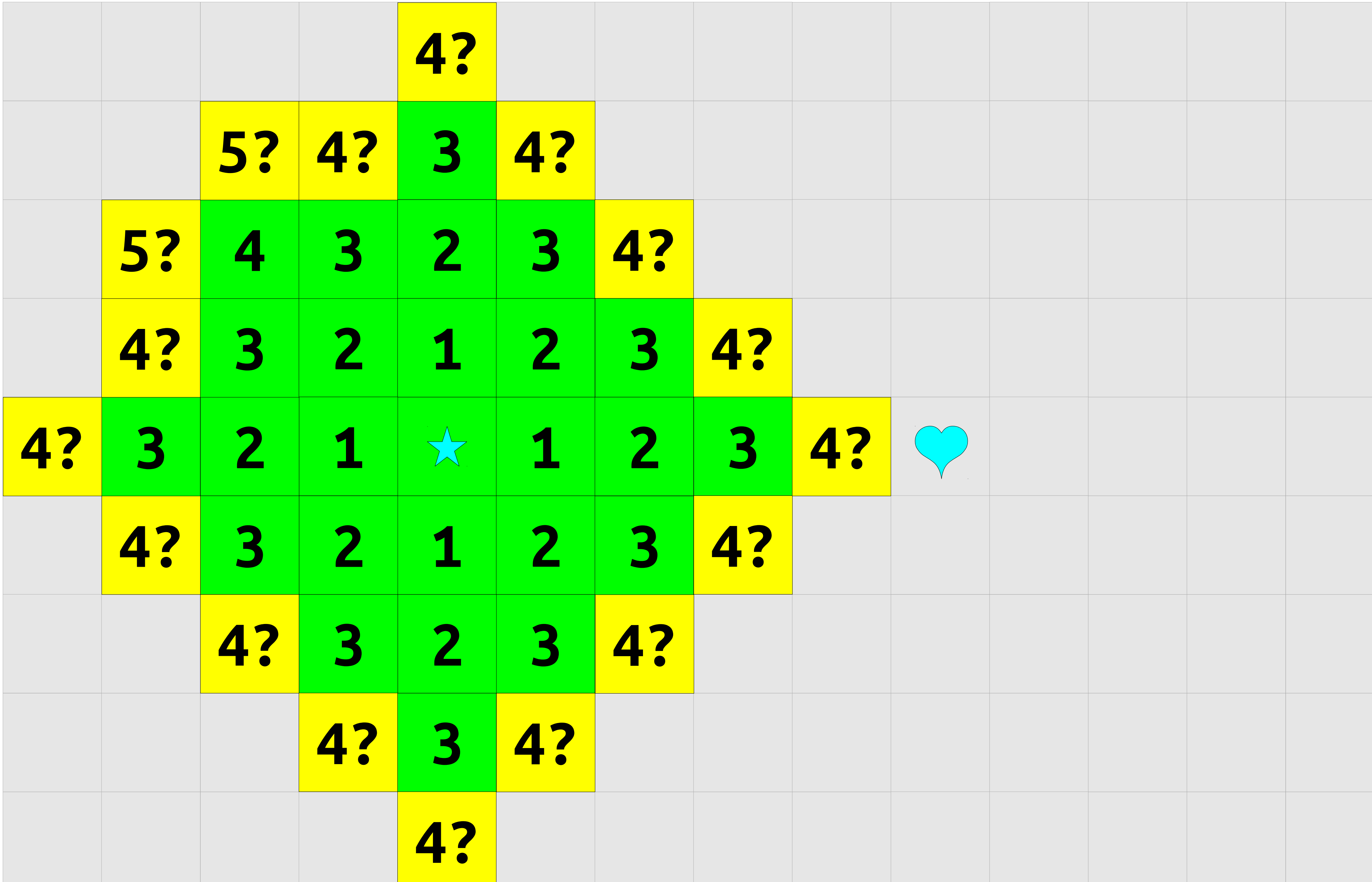


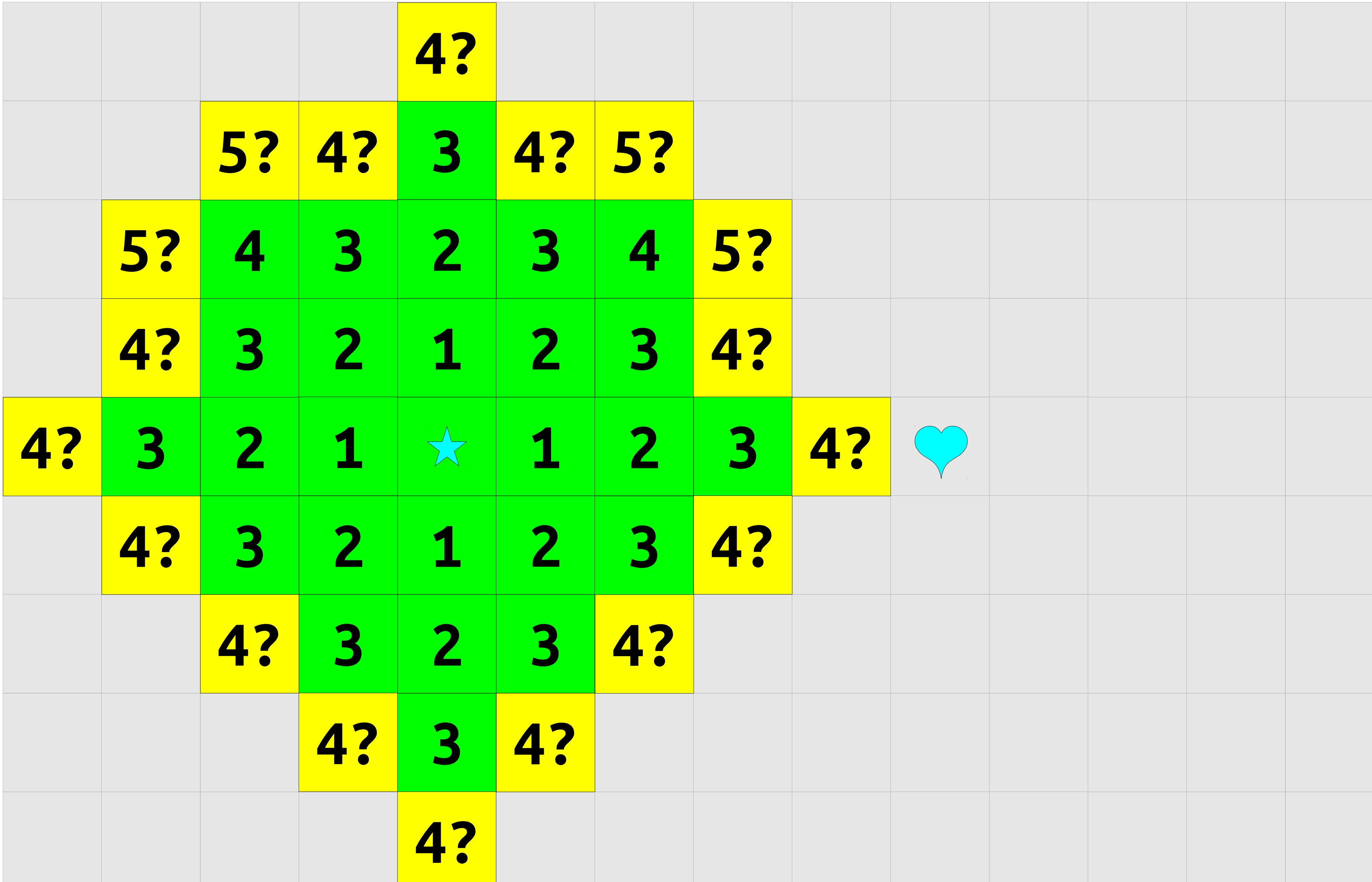
4?
4? 3 4?
4? 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

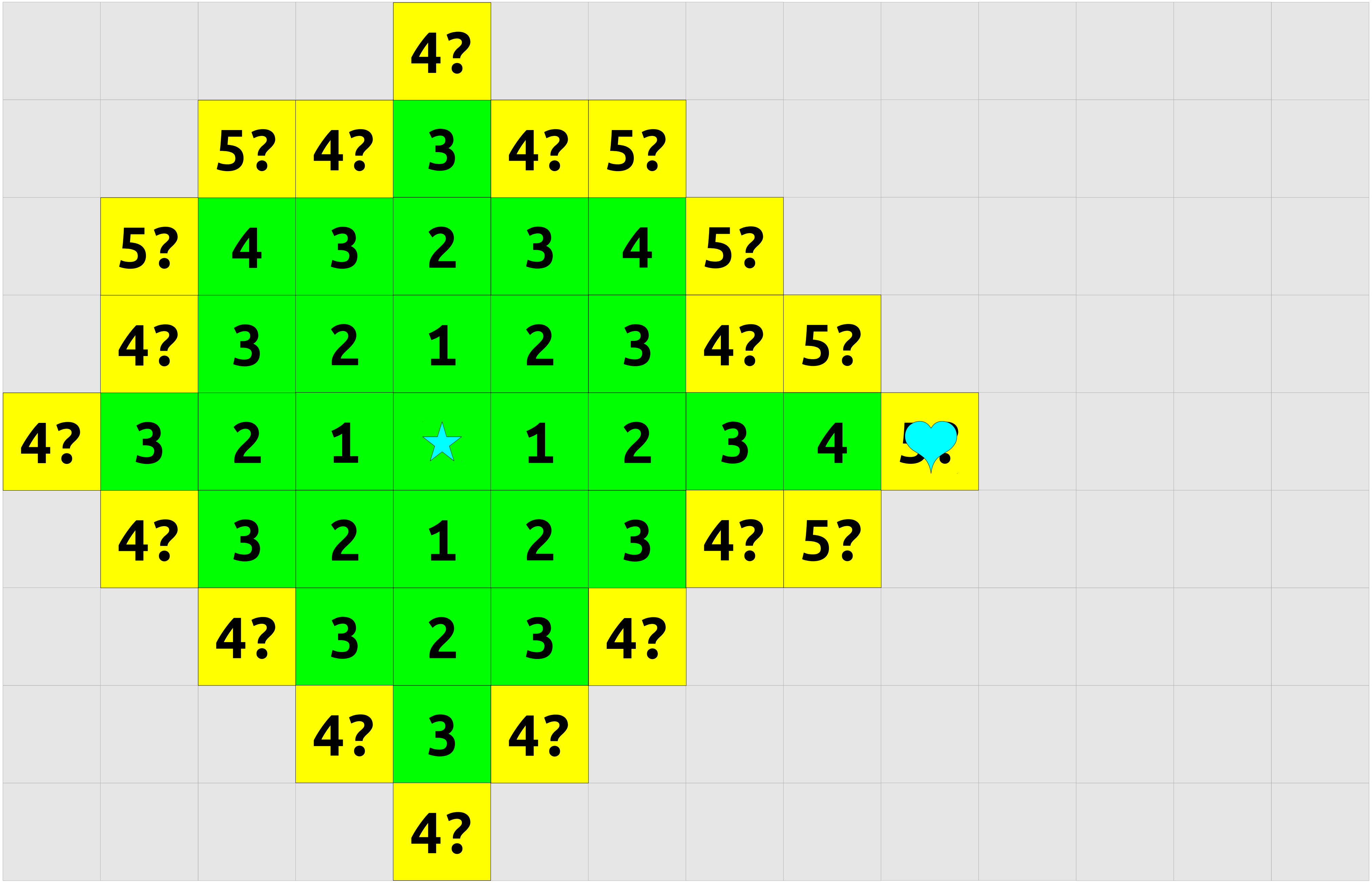

4?
4? 3 4?
4? 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3 2 3 4?
4? 3? 4?

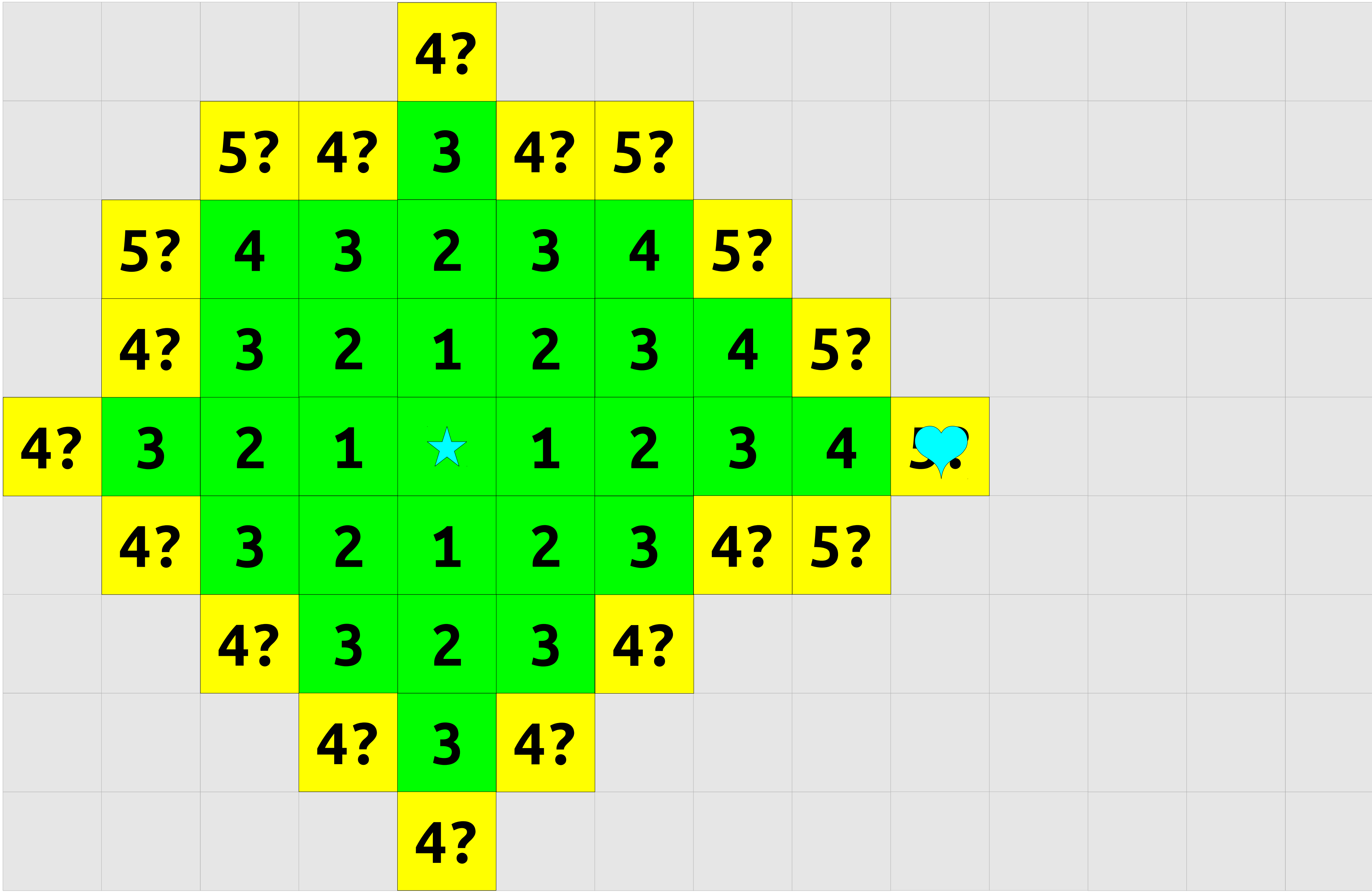

4?
4? 3 4?
4? 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3 2 3 4?
4? 3 4?
4?


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?

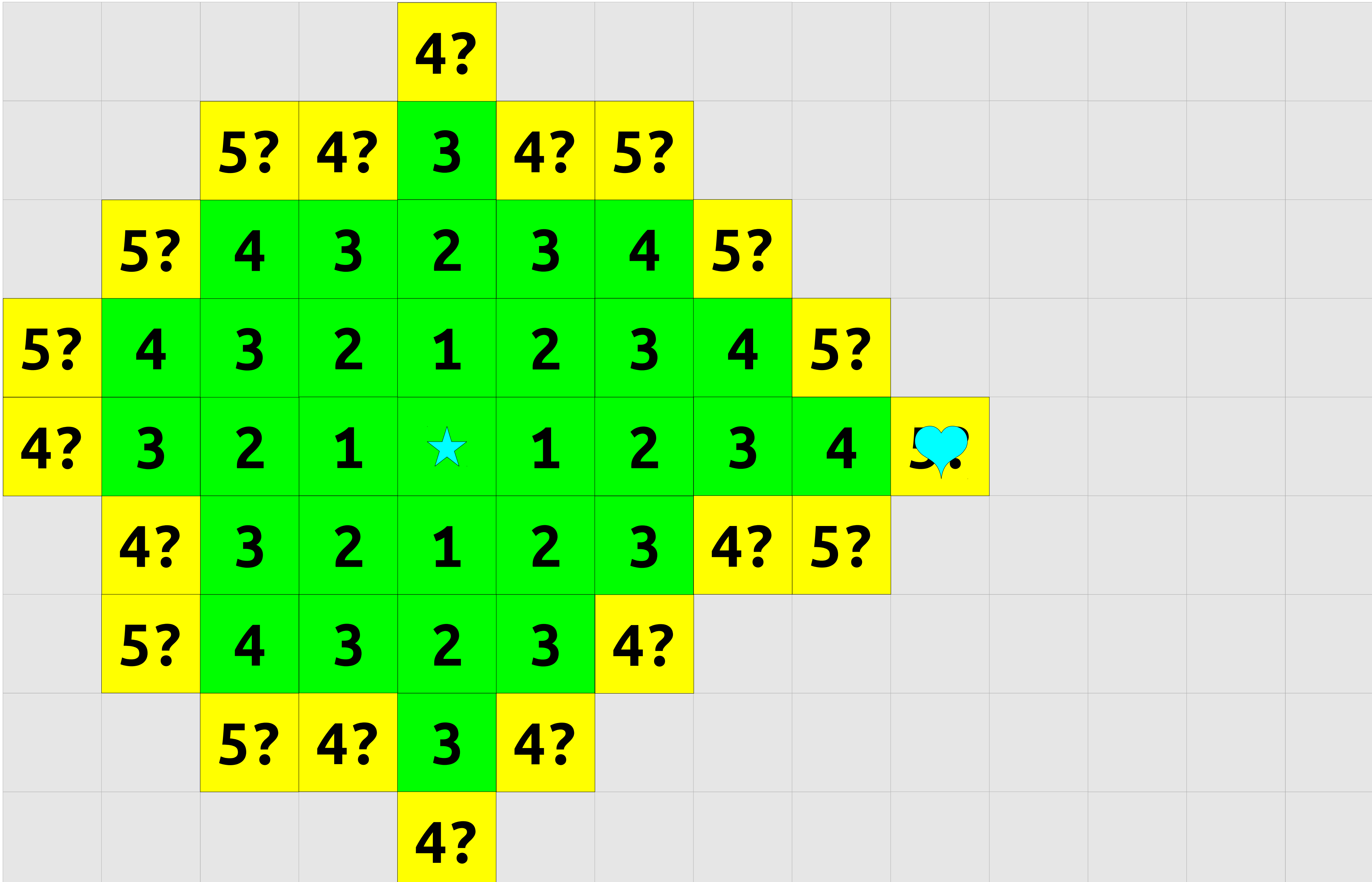

4?
5? 4? 3 4?
5? 4 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 3 4?
4? 3 4?
4?

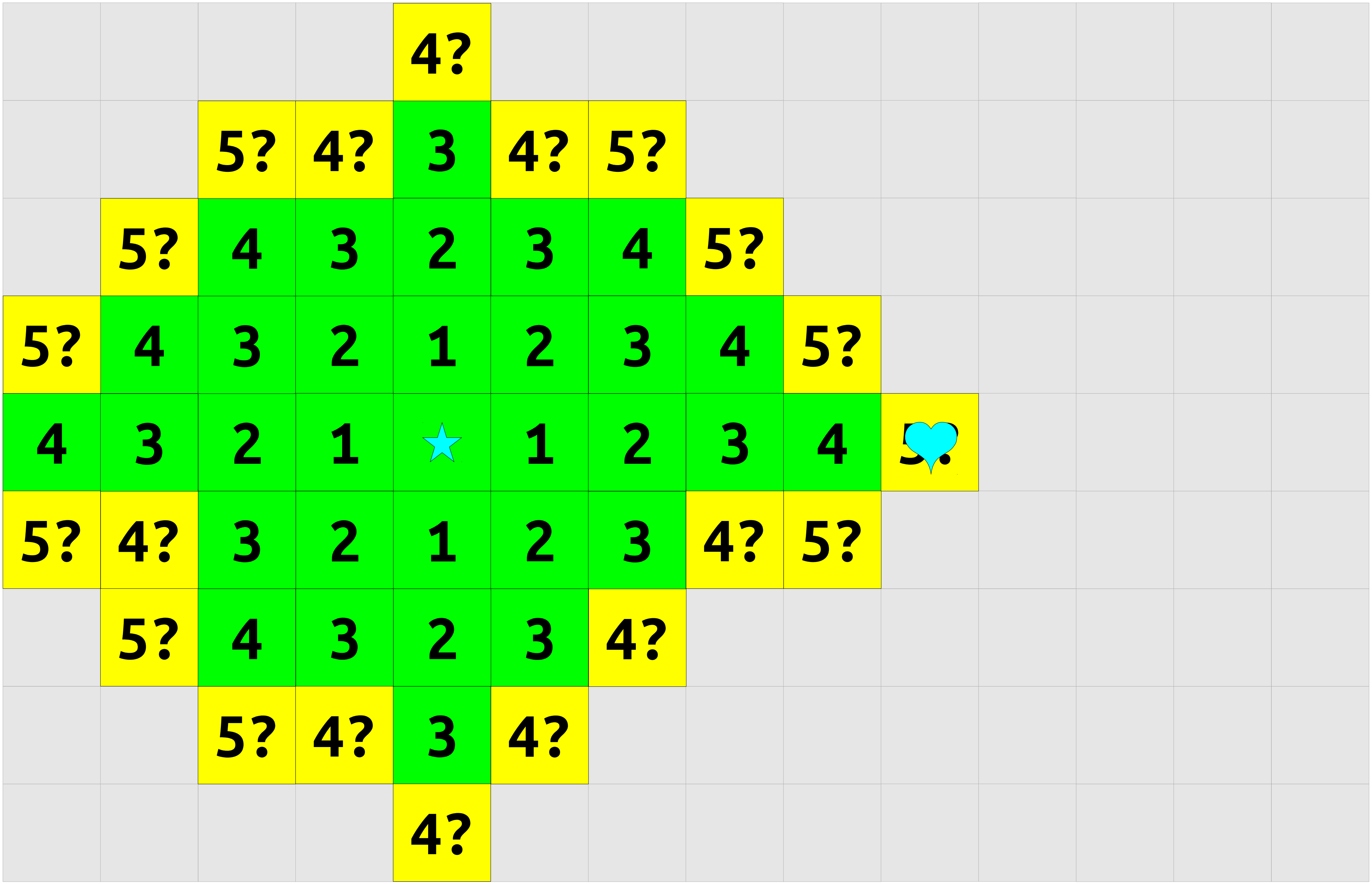

4?
5? 4? 3 4? 5?
5? 4 3 2 3 4 5?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 3 4?
4? 3 4?
4?

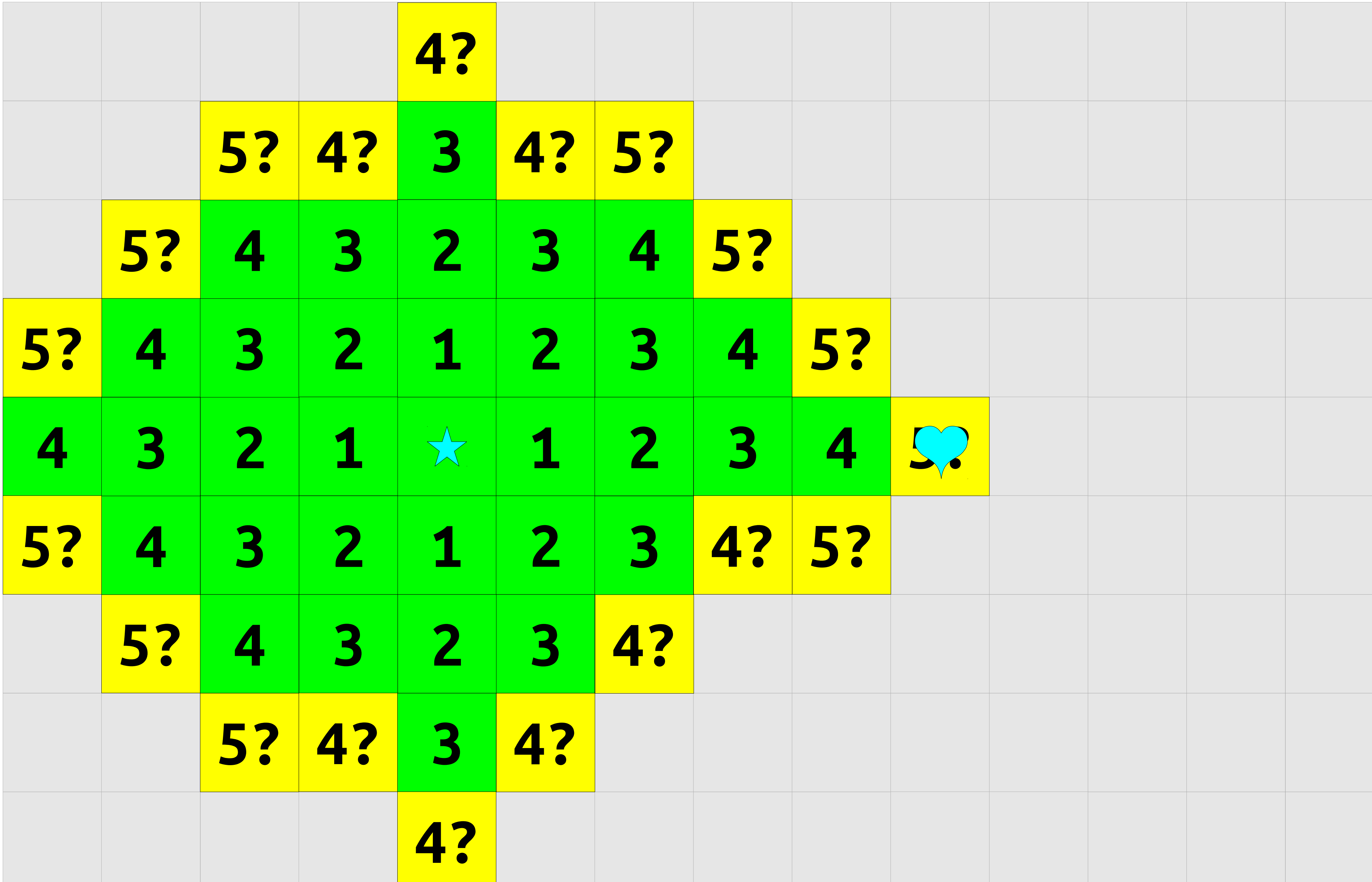

4?
5? 4? 3 4? 5?
5? 4 3 2 3 4 5?
4? 3 2 1 2 3 4? 5?
4? 3 2 1 1 2 3 4
4? 3 2 1 2 3 4? 5?
4? 3 2 3 4?
4? 3 4?
4?


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

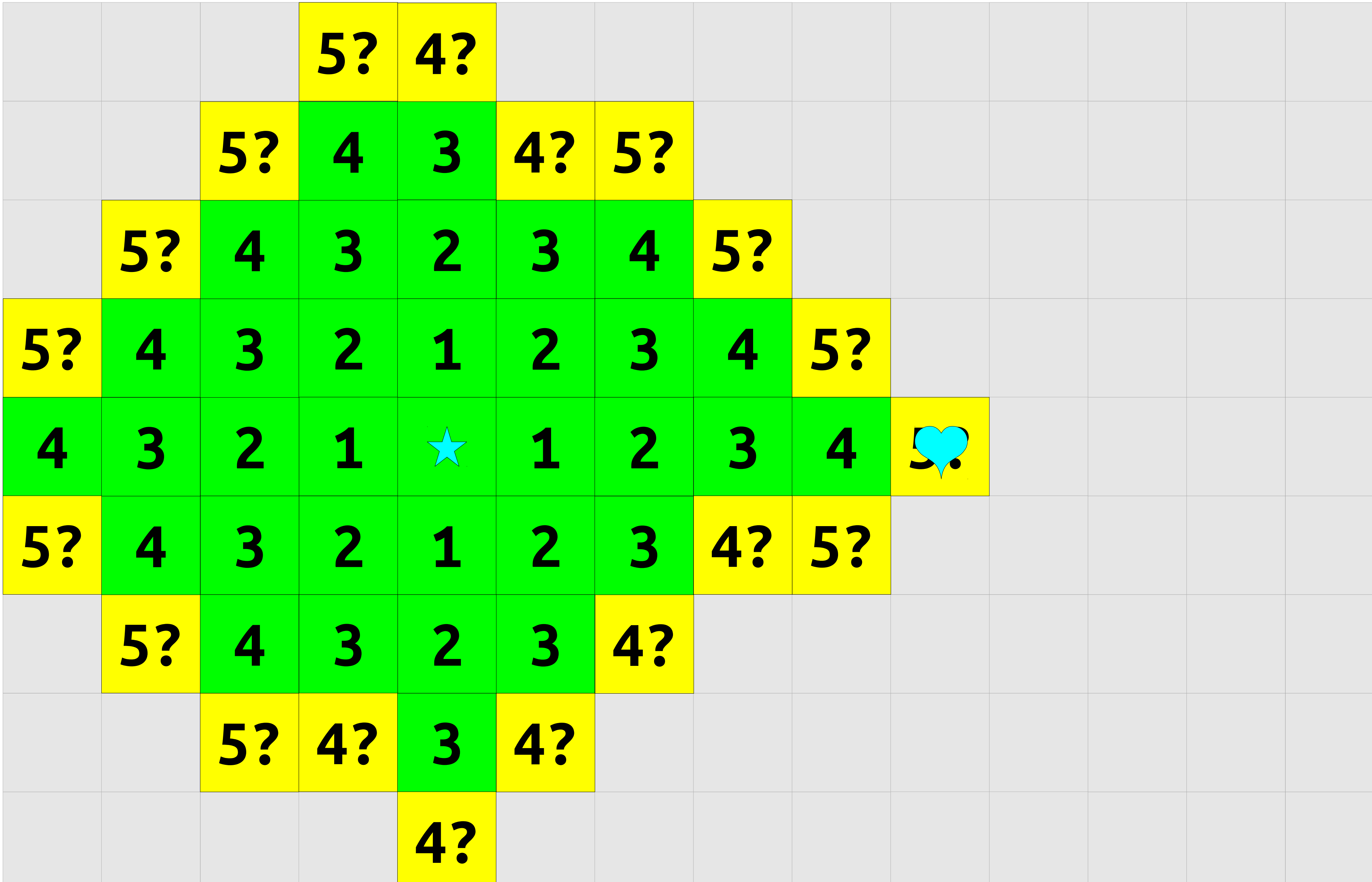
				4?									
		5?	4?	3	4?	5?							
	5?	4	3	2	3	4	5?						
	4?	3	2	1	2	3	4	5?					
4?	3	2	1	★	1	2	3	4	5?				
	4?	3	2	1	2	3	4?	5?					
	5?	4	3	2	3	4?							
		5?	4?	3	4?								
				4?									

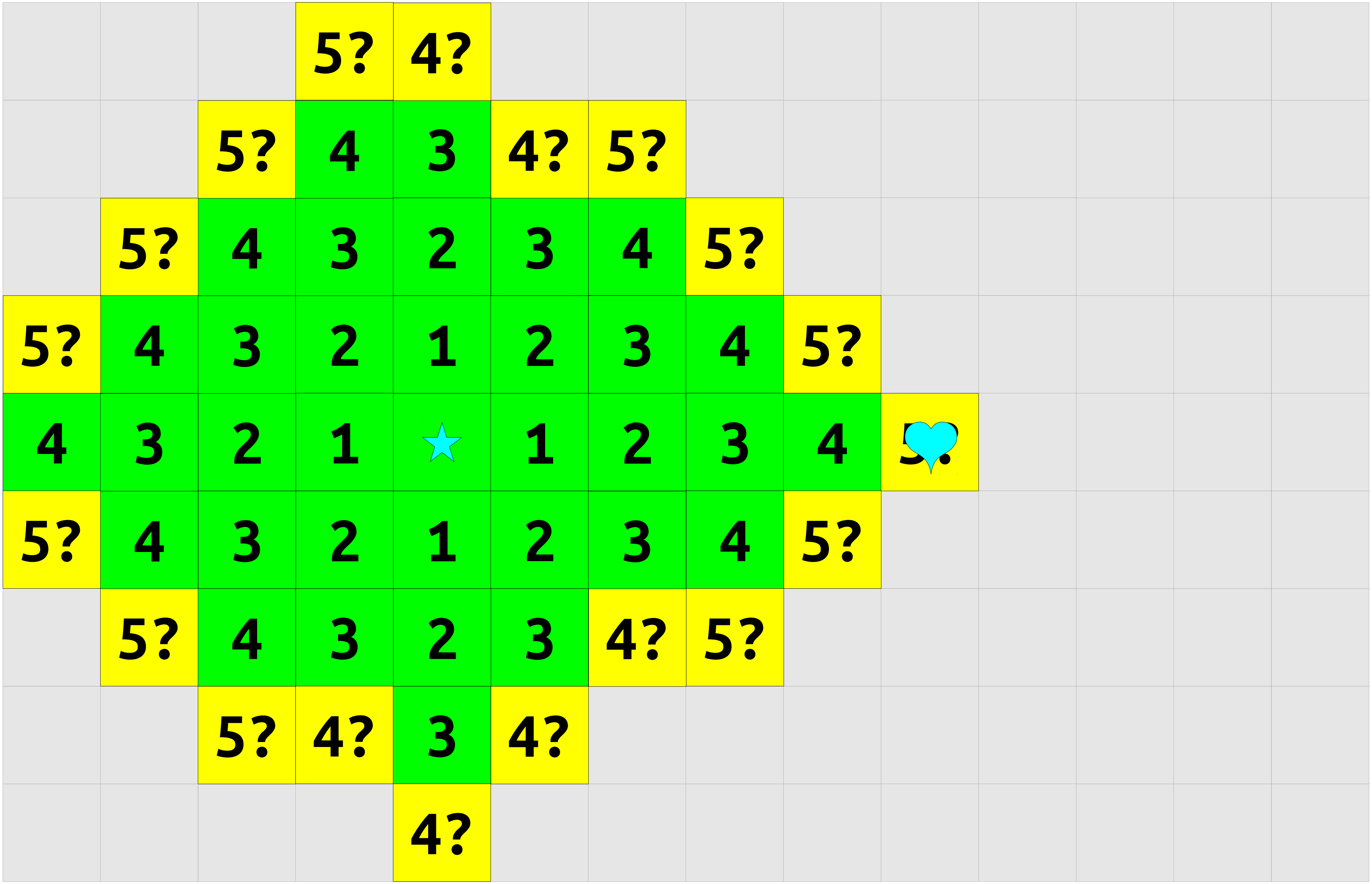

4?
5? 4? 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4? 3 2 1 1 2 3 4
4? 3 2 1 2 3 4? 5?
5? 4 3 2 3 4?
5? 4? 3 4?
4?

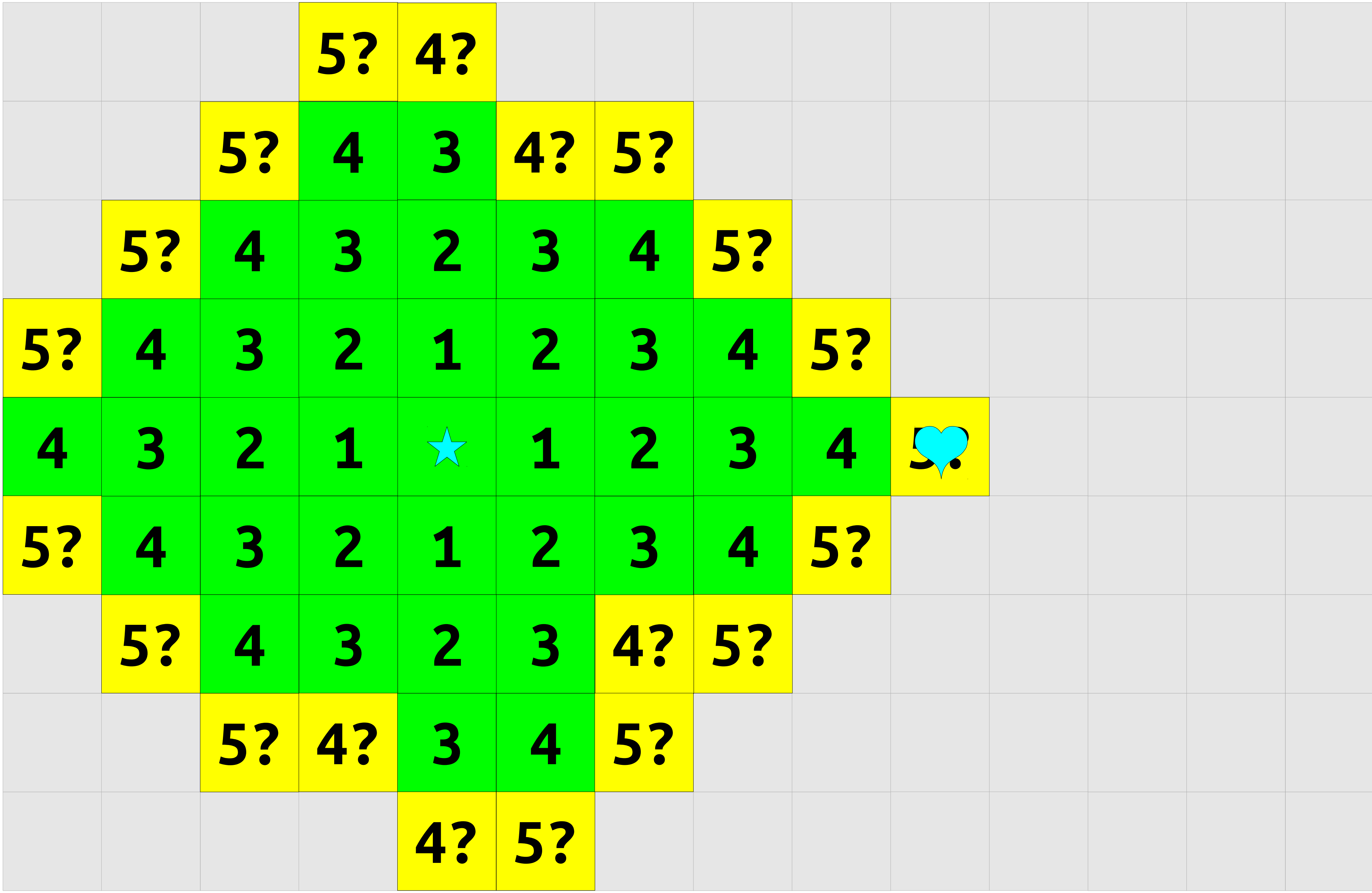

4?
5? 4? 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4
5? 4? 3 2 1 2 3 4? 5?
5? 4 3 2 3 4?
5? 4? 3 4?
4?

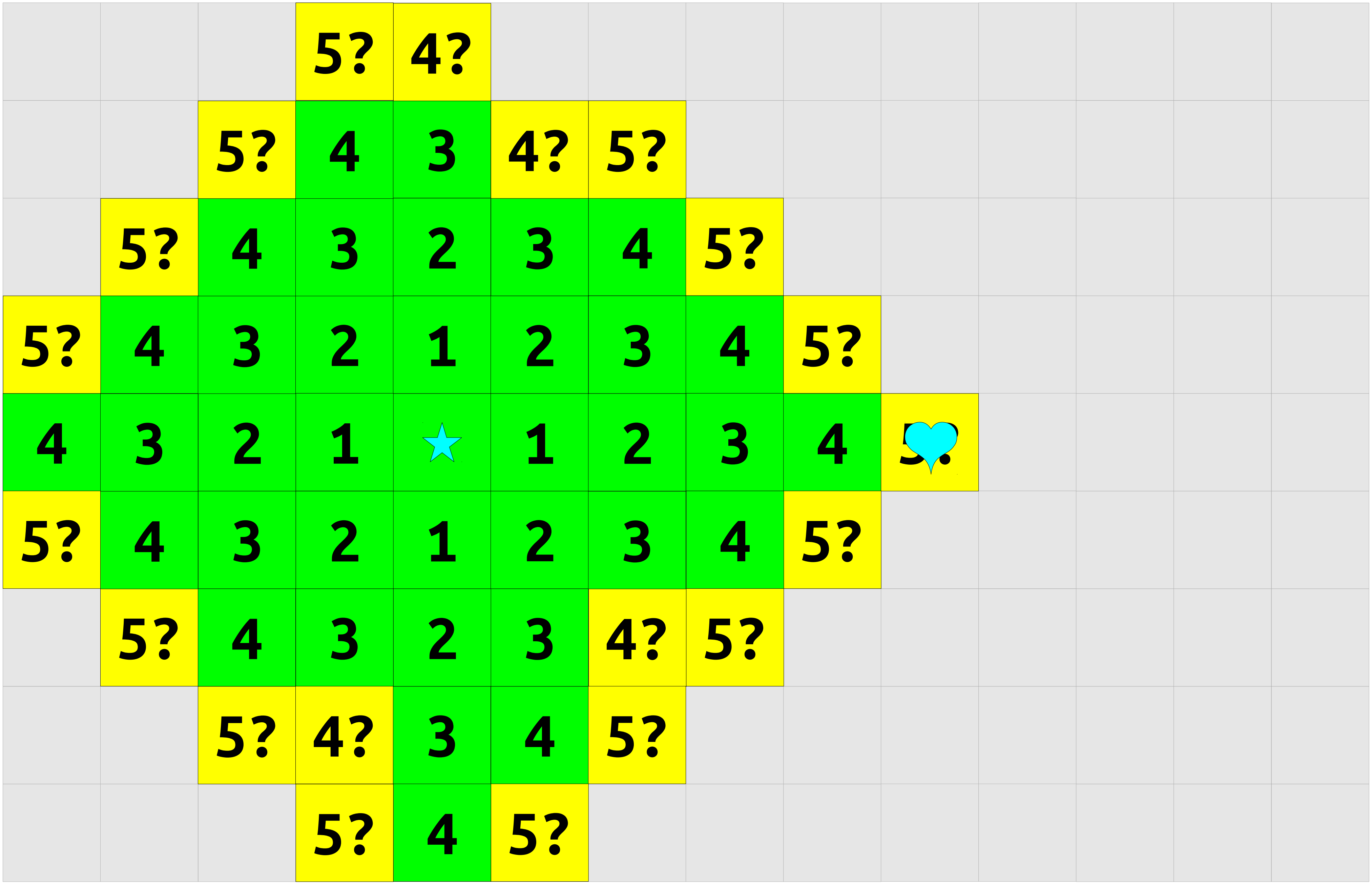


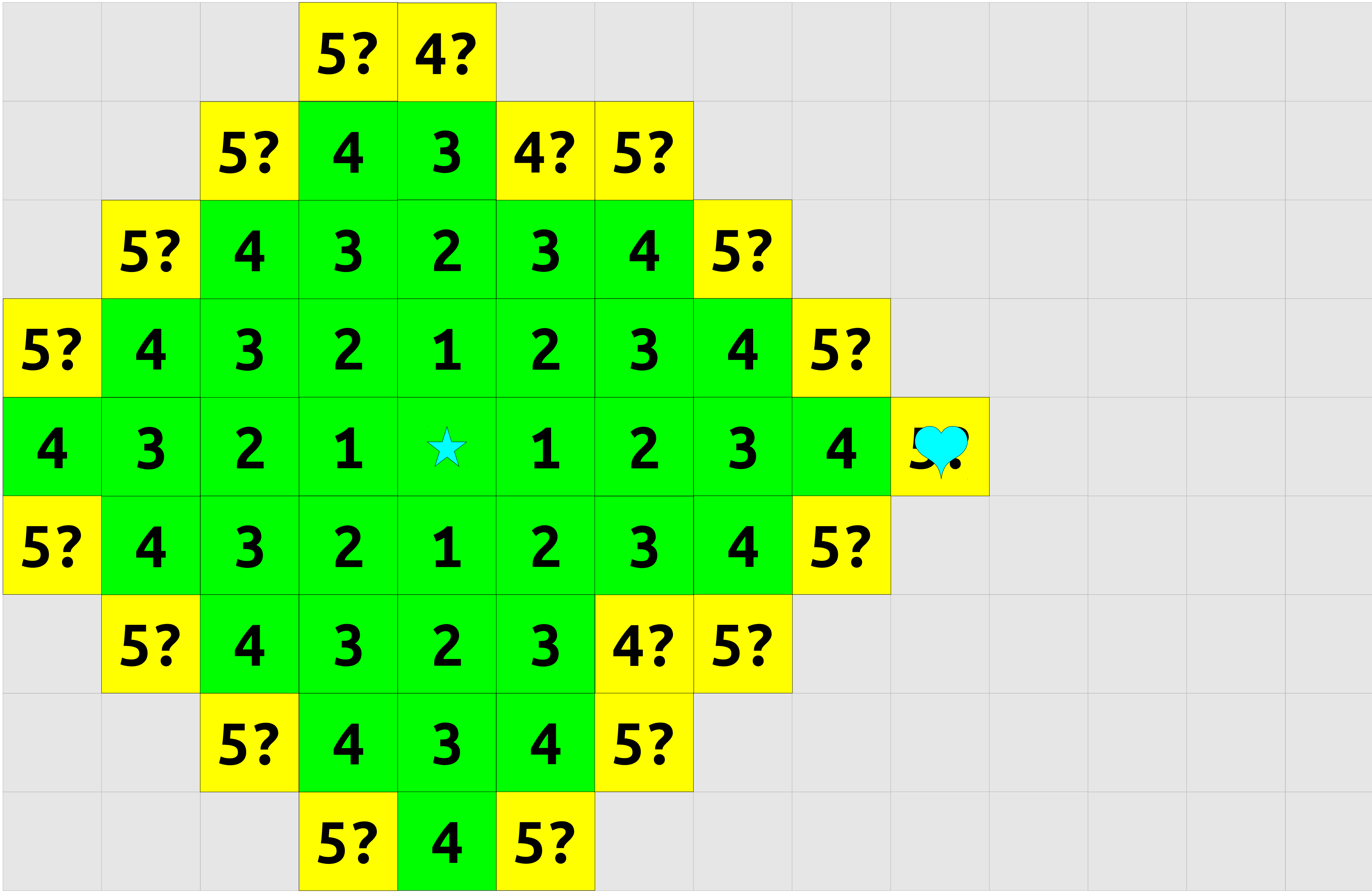
4?
5? 4? 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4
5? 4 3 2 1 2 3 4? 5?
5? 4 3 2 3 4?
5? 4? 3 4?
4?

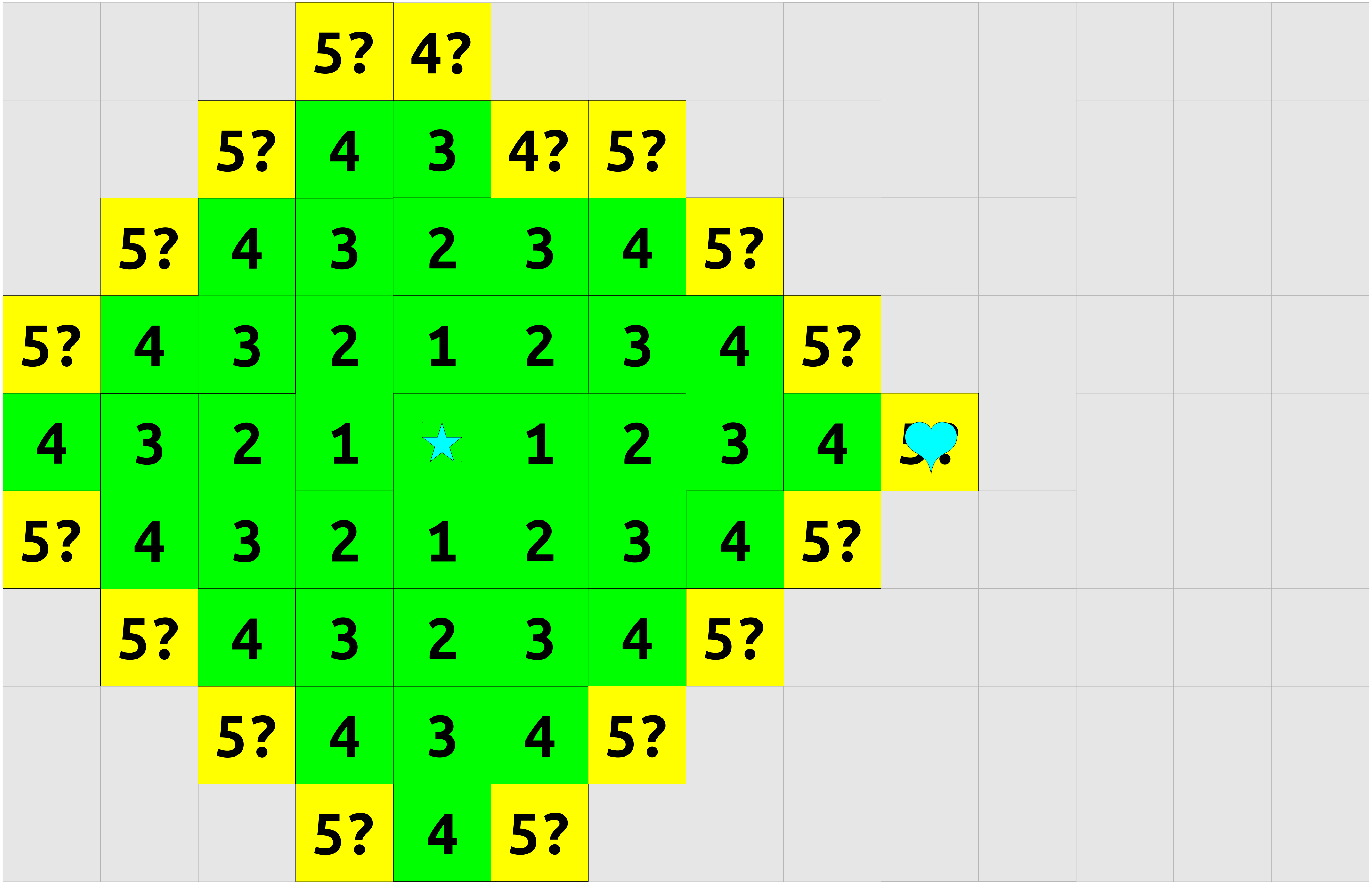

5? 4?
5? 4 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4
5? 4 3 2 1 2 3 4? 5?
5? 4 3 2 3 4?
5? 4? 3 4?
4?

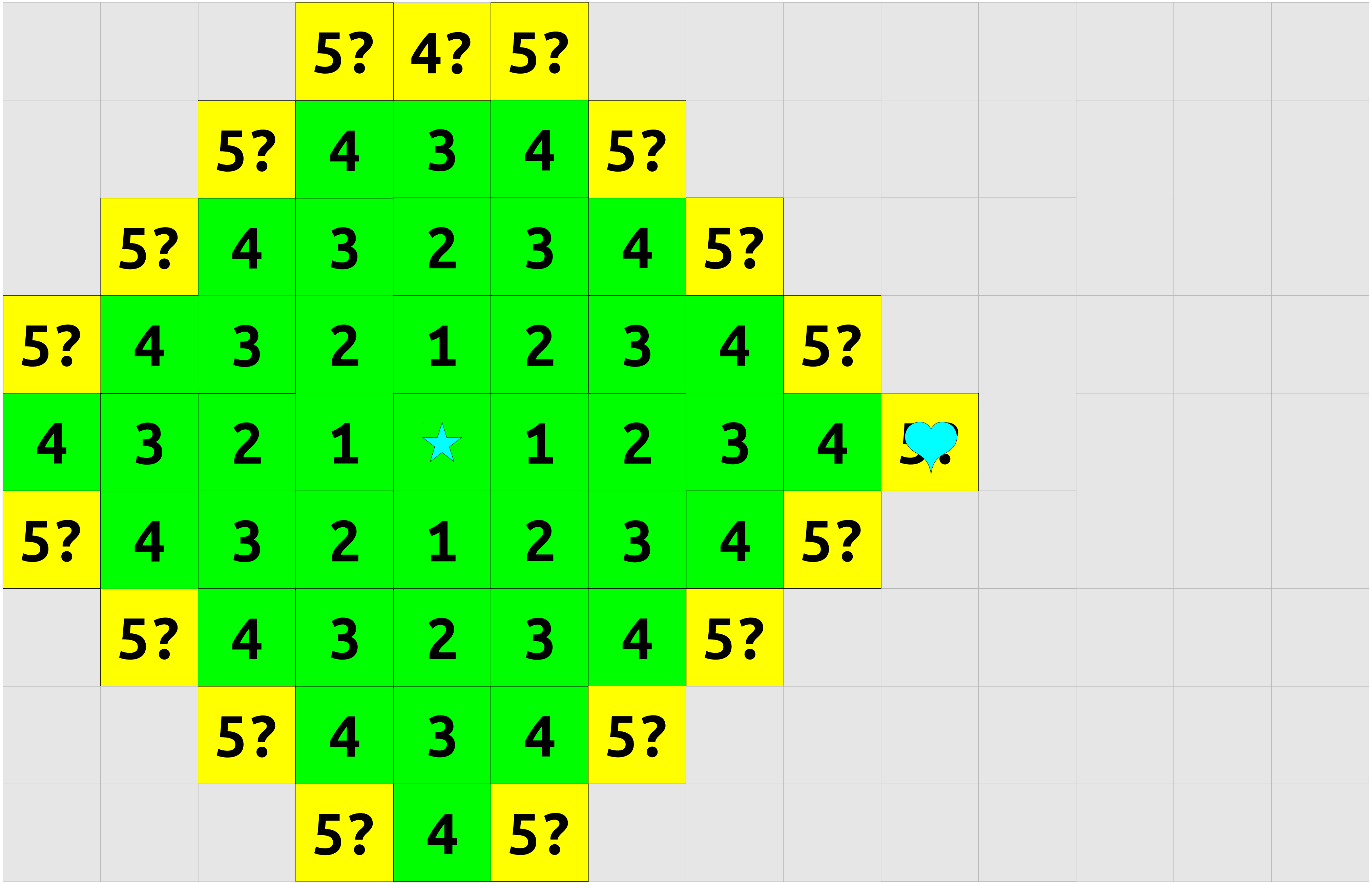

5? 4?
5? 4 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4? 5?
5? 4? 3 4?
4?

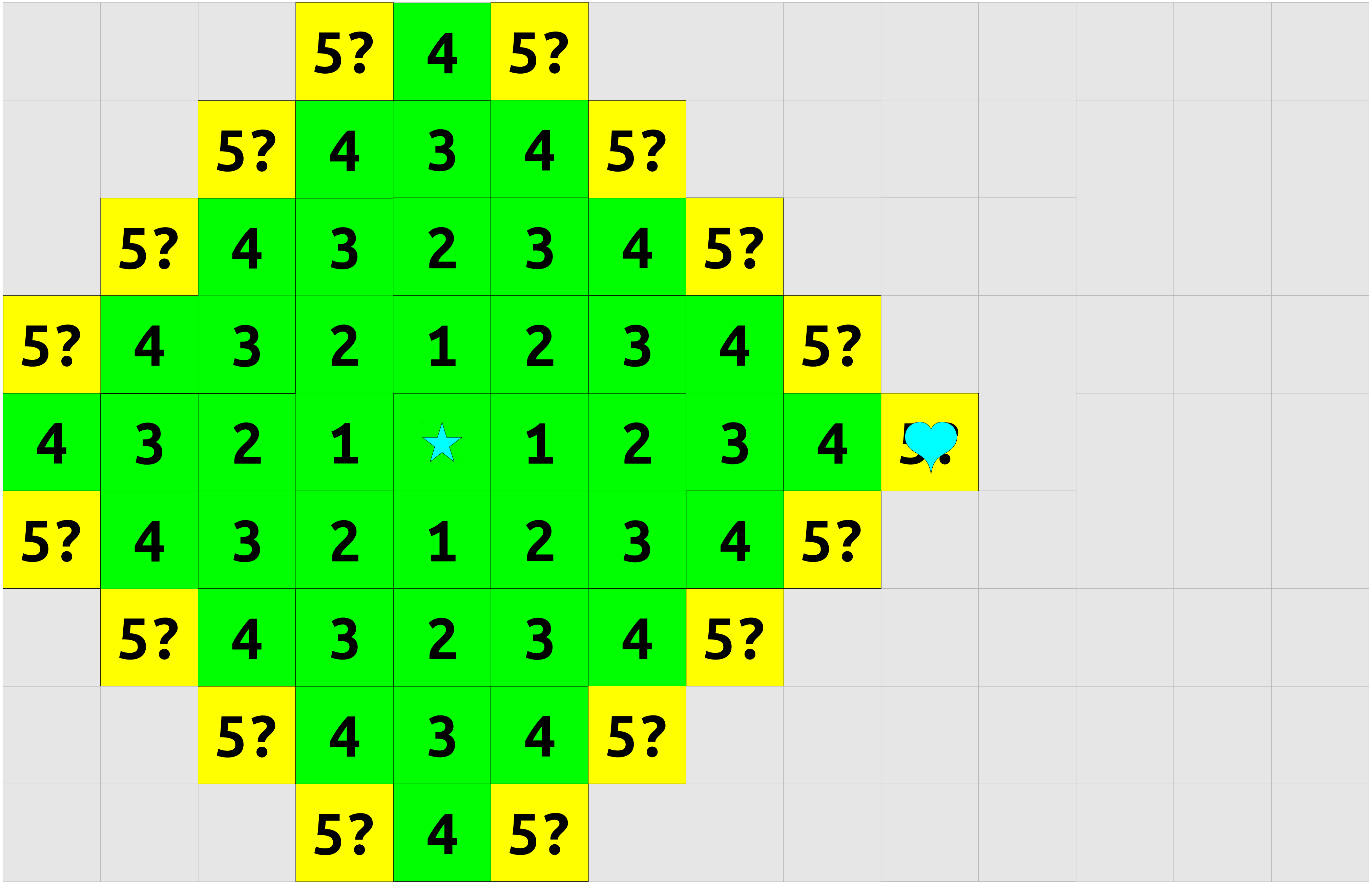

5? 4?
5? 4 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4? 5?
5? 4? 3 4 5?
4? 5?

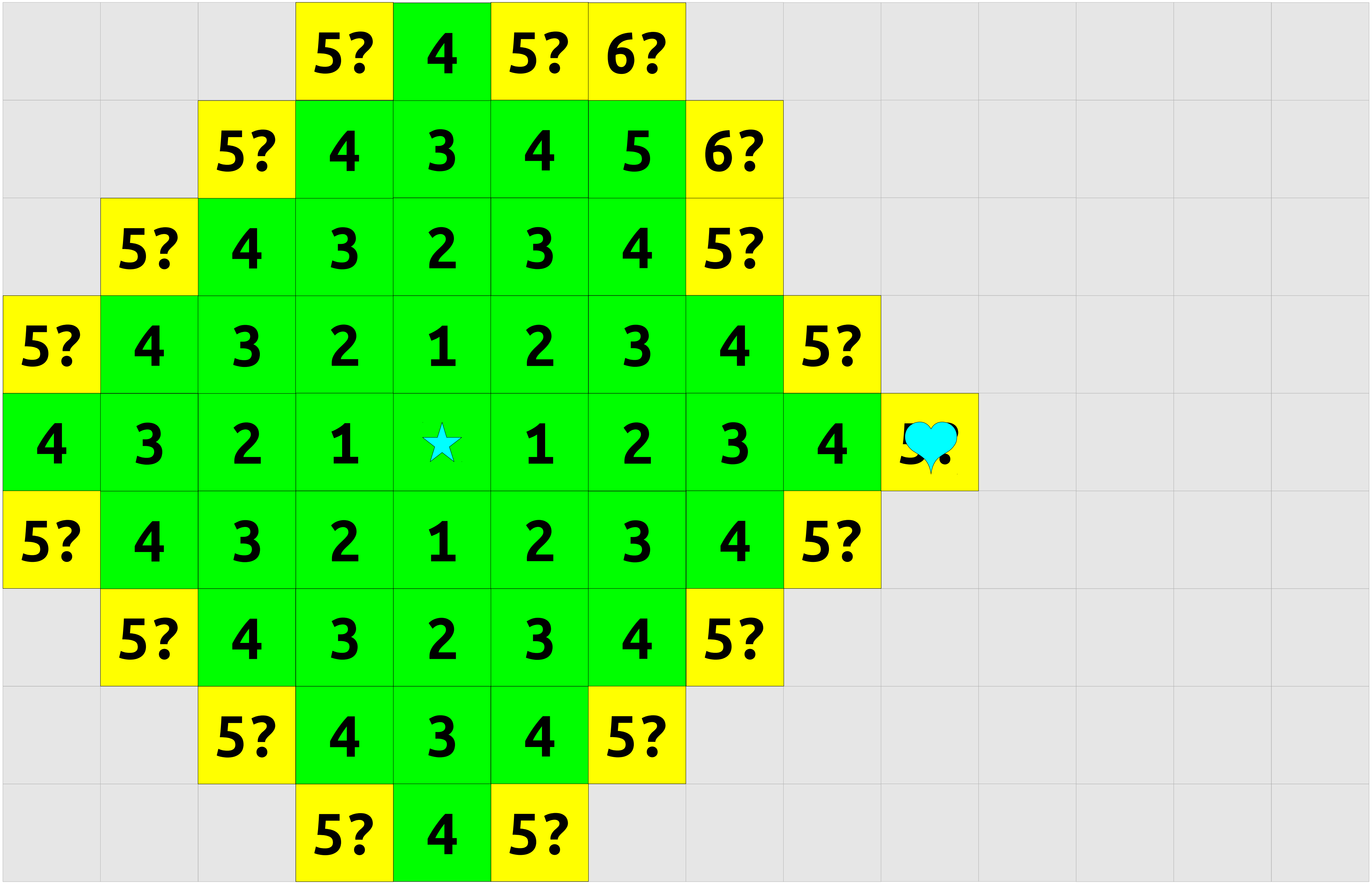

5? 4?
5? 4 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4? 5?
5? 4? 3 4 5?
5? 4 5?


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

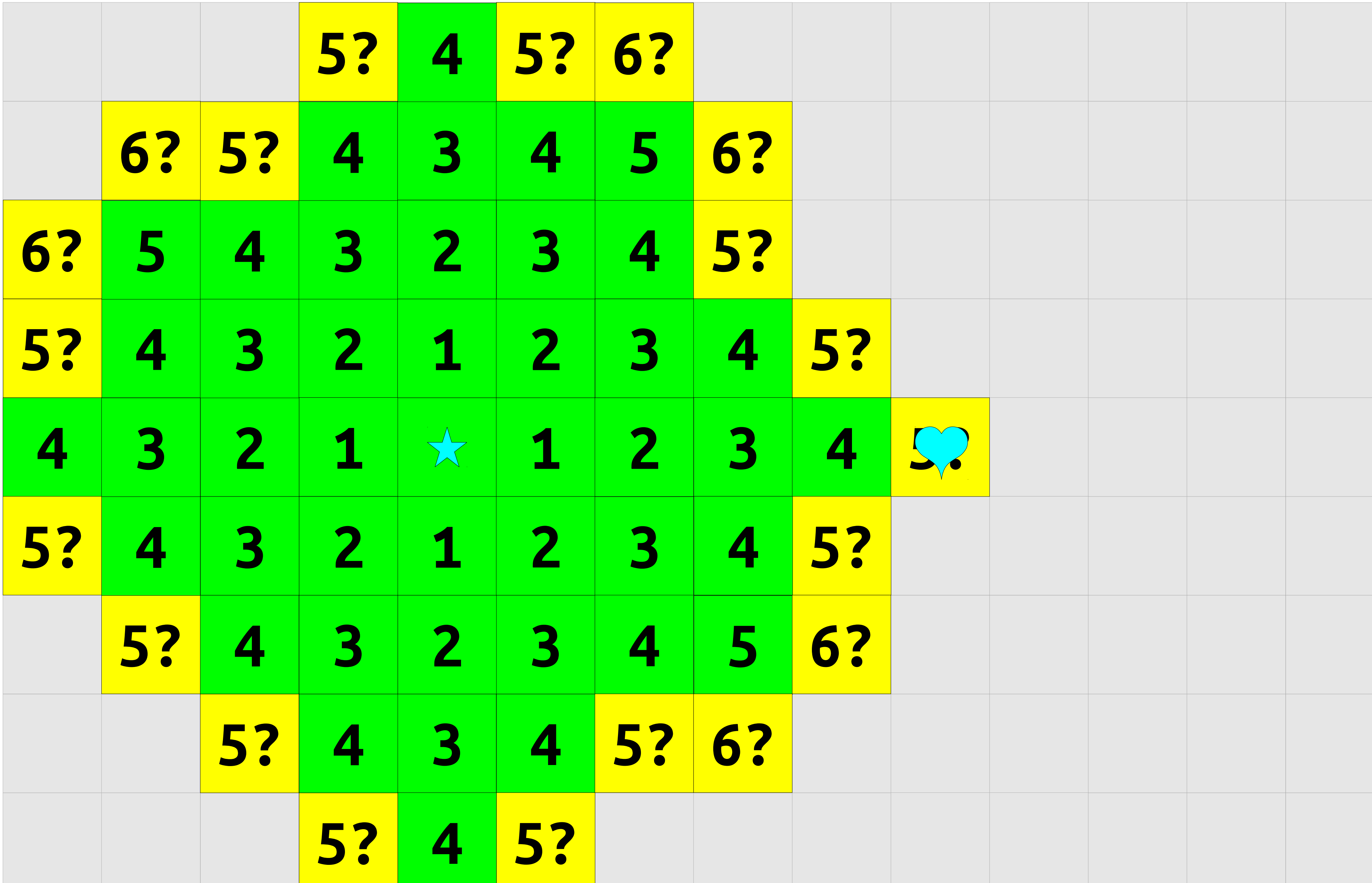

5? 4?
5? 4 3 4? 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4 5?
5? 4 3 4 5?
5? 4 5?


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

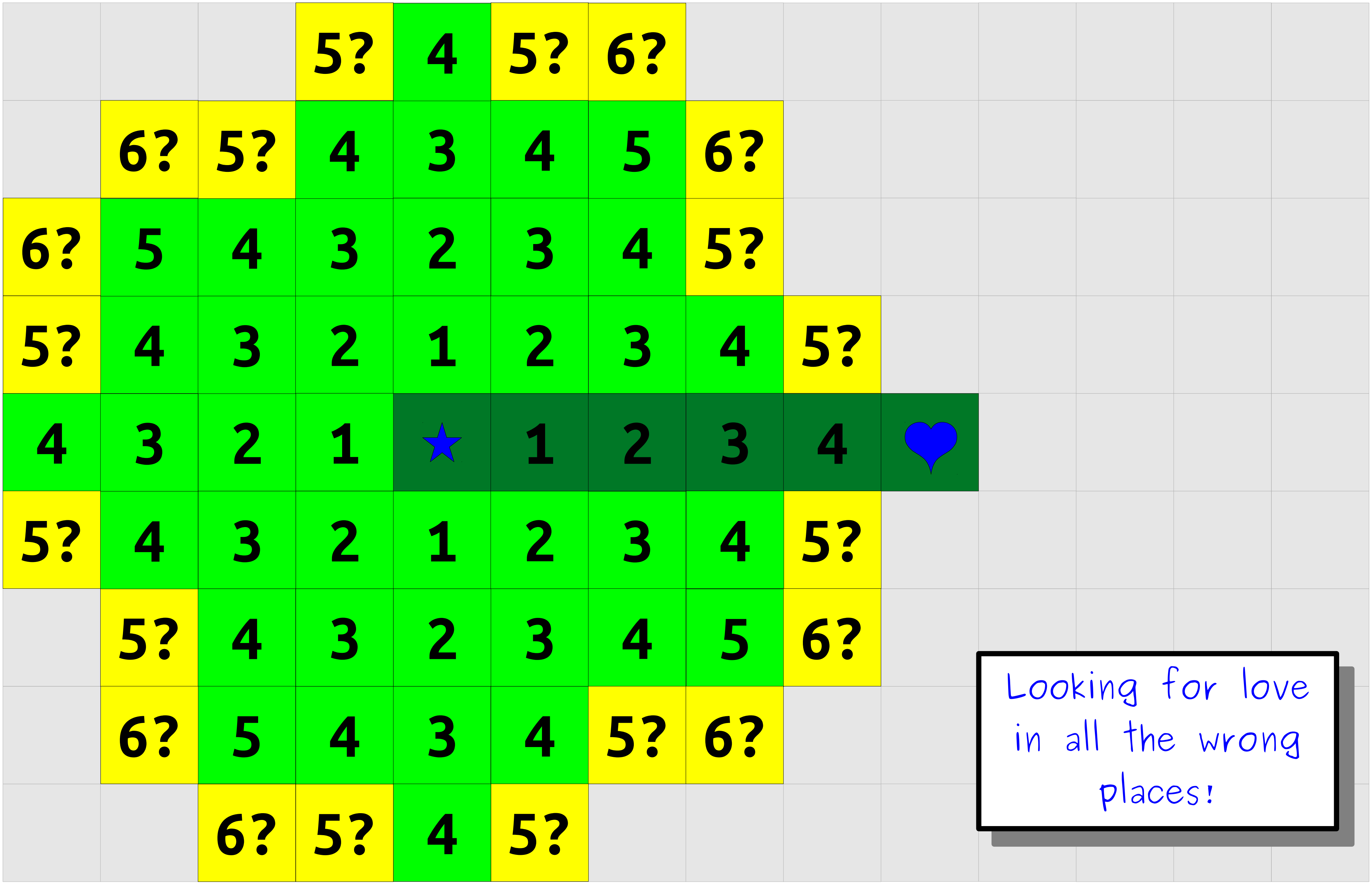

5? 4 5?
5? 4 3 4 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4 5?
5? 4 3 4 5?
5? 4 5?


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? 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?	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?								

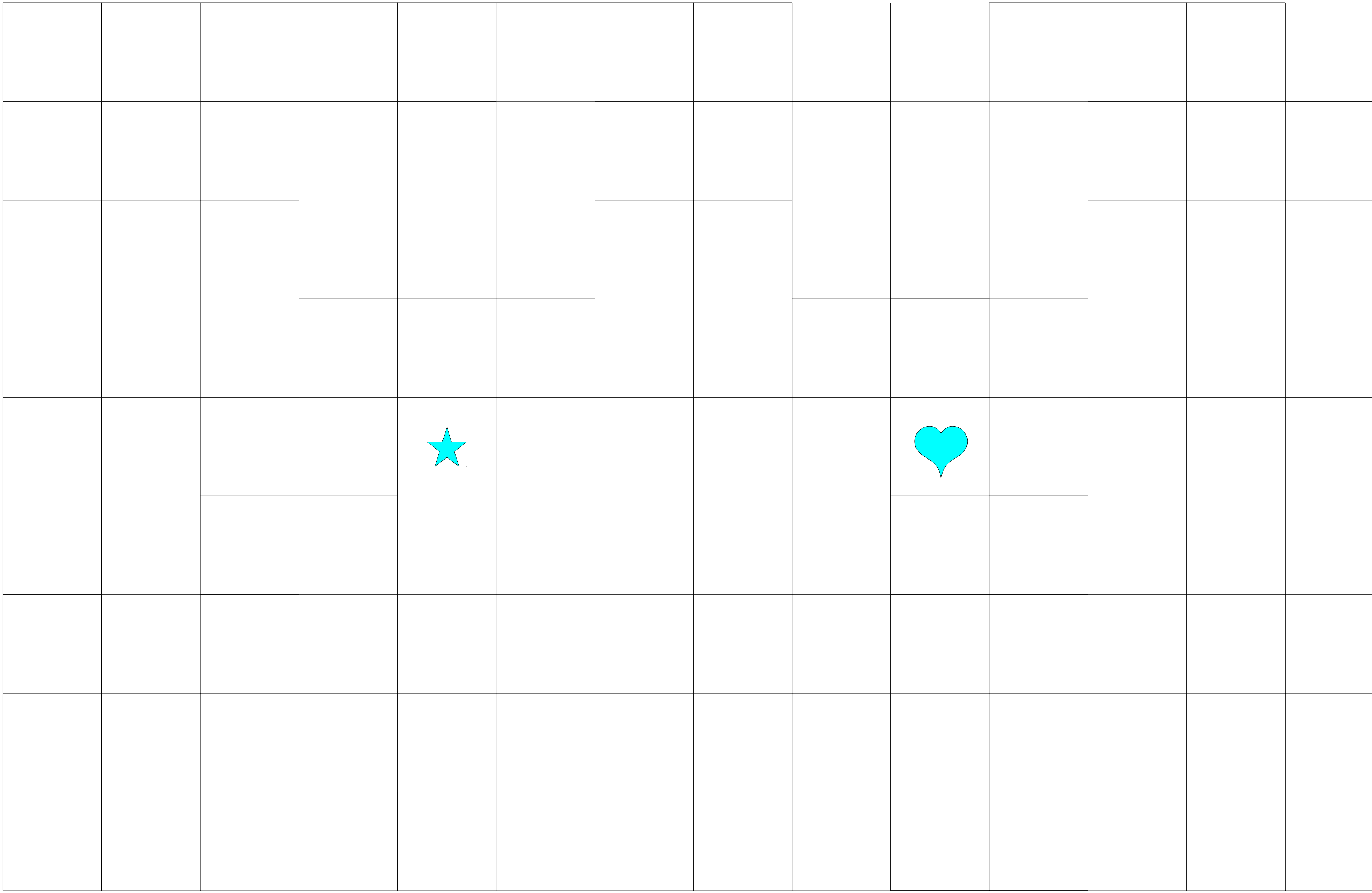

5? 4 5? 6?
6? 5? 4 3 4 5 6?
6? 5 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4 5 6?
5? 4 3 4 5? 6?
5? 4 5?

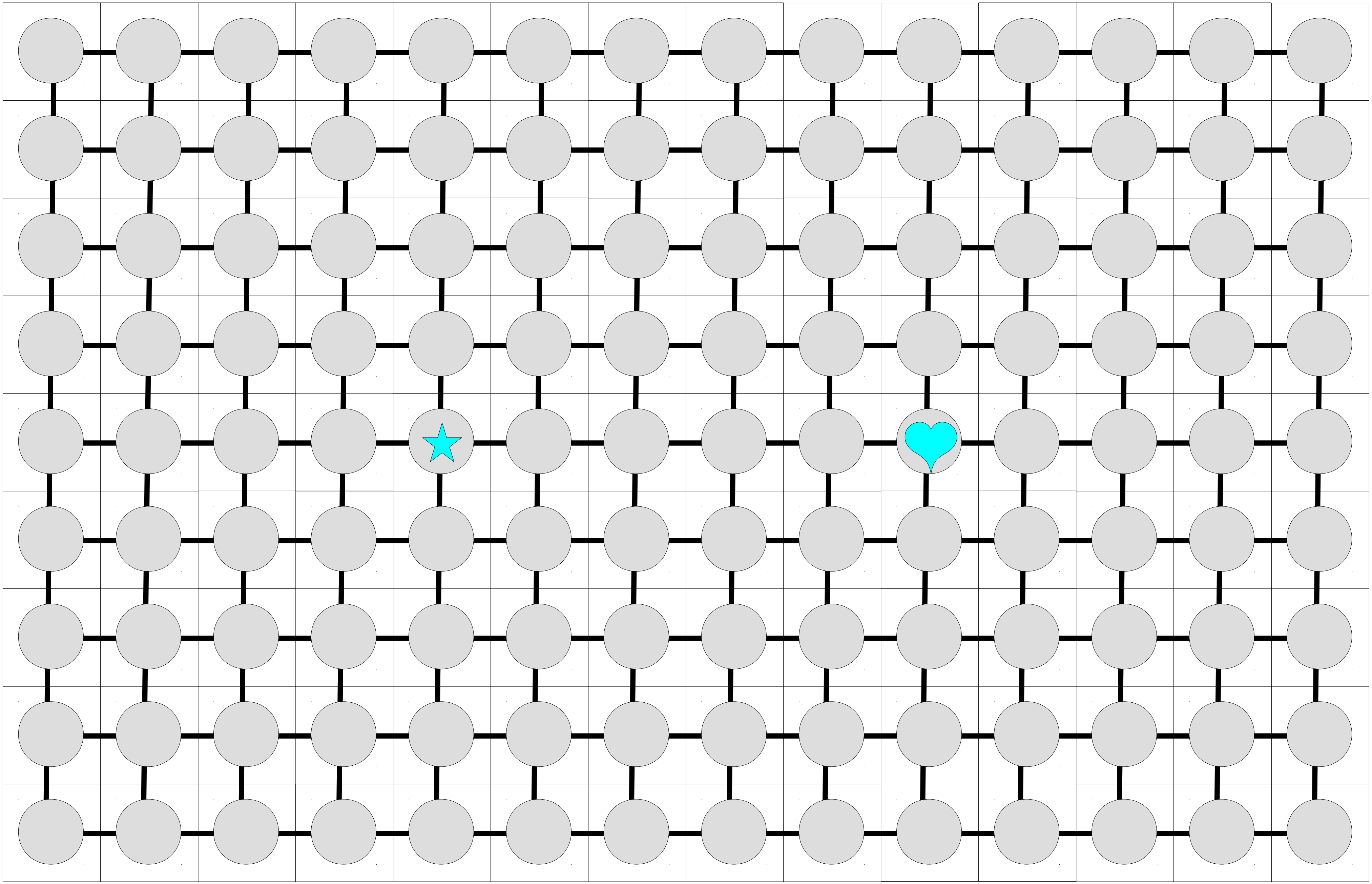
			5?	4	5?	6?							
	6?	5?	4	3	4	5	6?						
6?	5	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	★	1	2	3	4	♥				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4	5	6?					
	6?	5	4	3	4	5?	6?						
		6?	5?	4	5?								

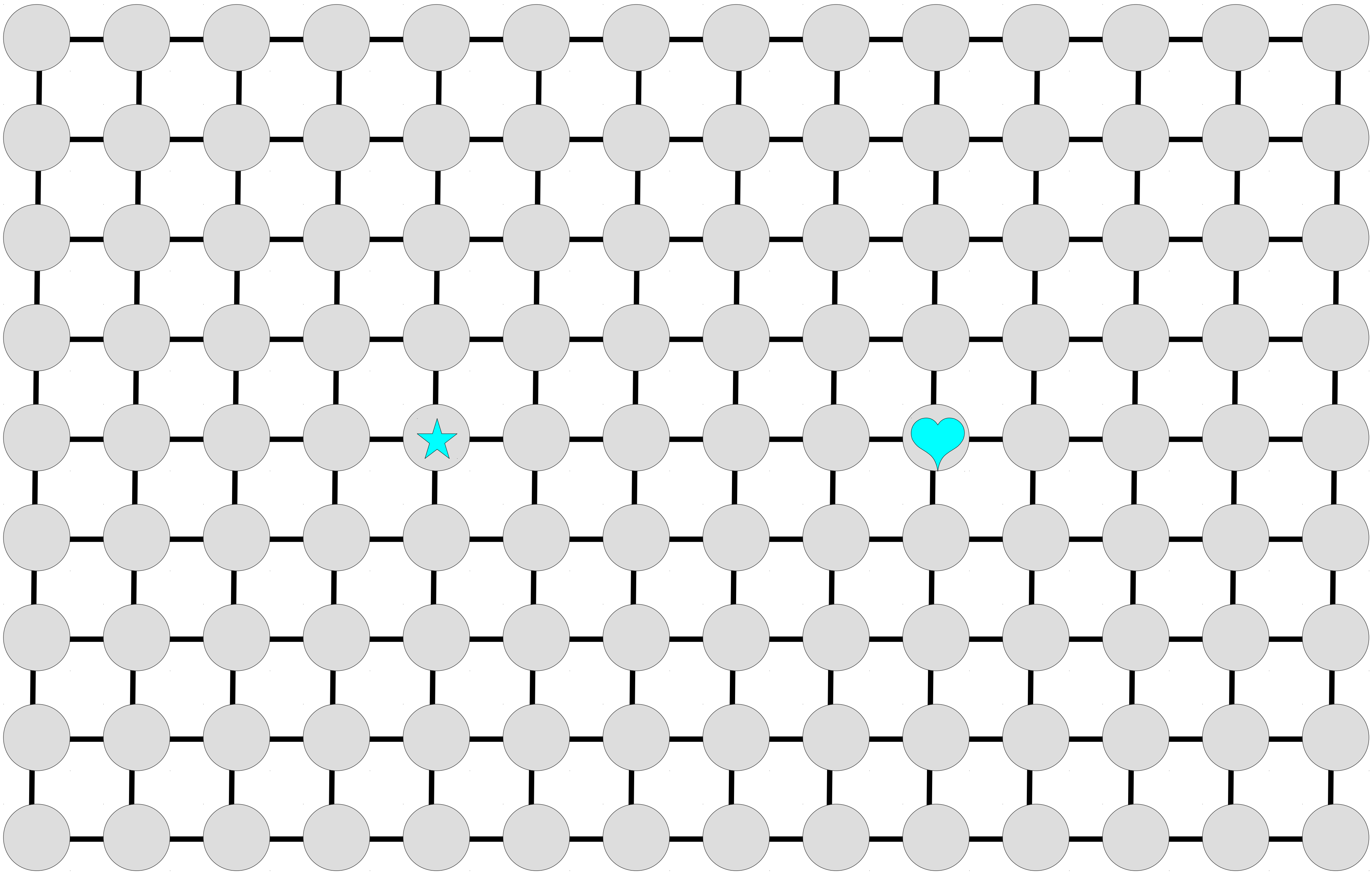

5? 4 5? 6?
6? 5? 4 3 4 5 6?
6? 5 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4 5 6?
6? 5 4 3 4 5? 6?
6? 5? 4 5?
Looking for love in all the wrong places!

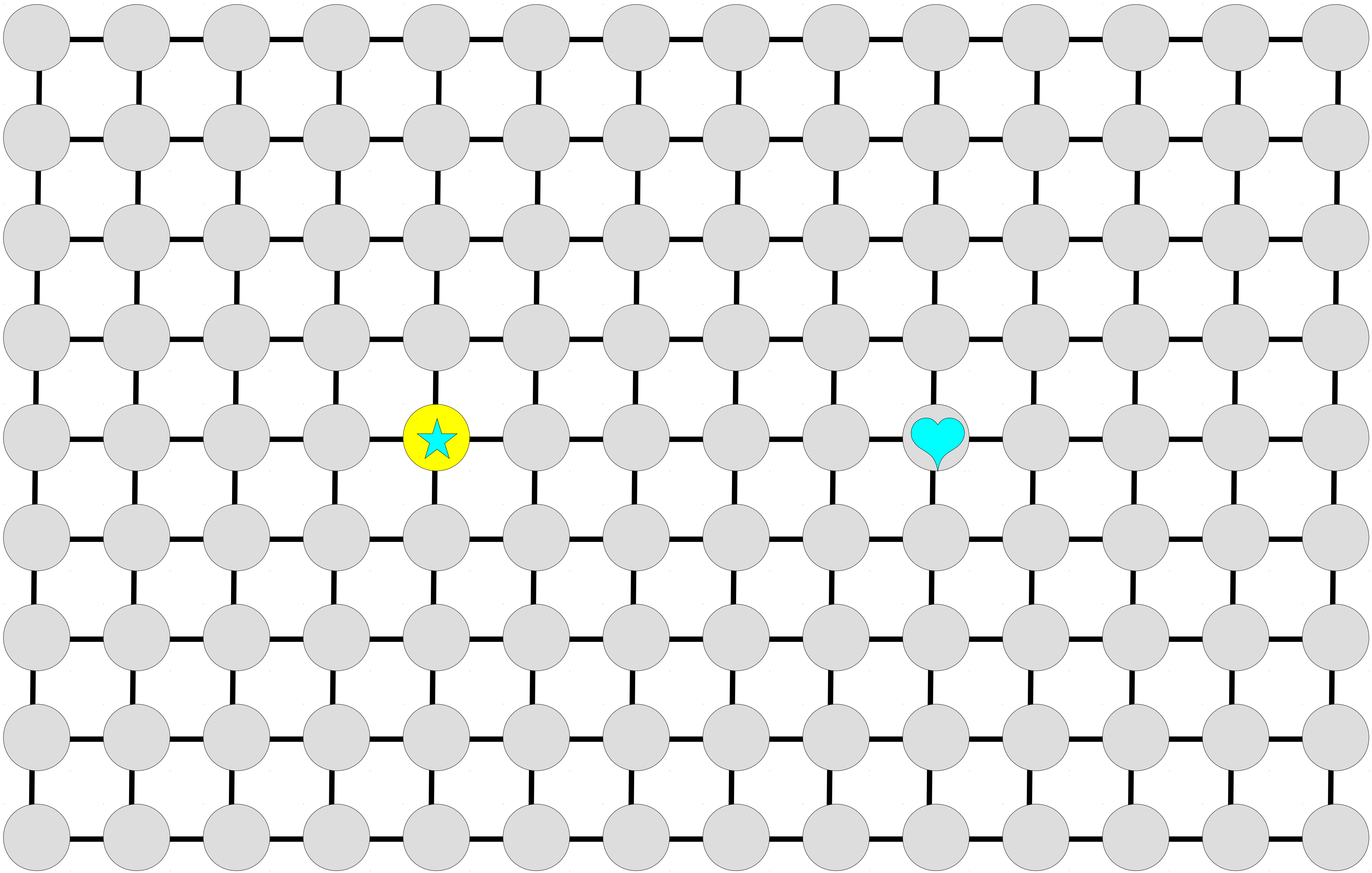
How Dijkstra's Works

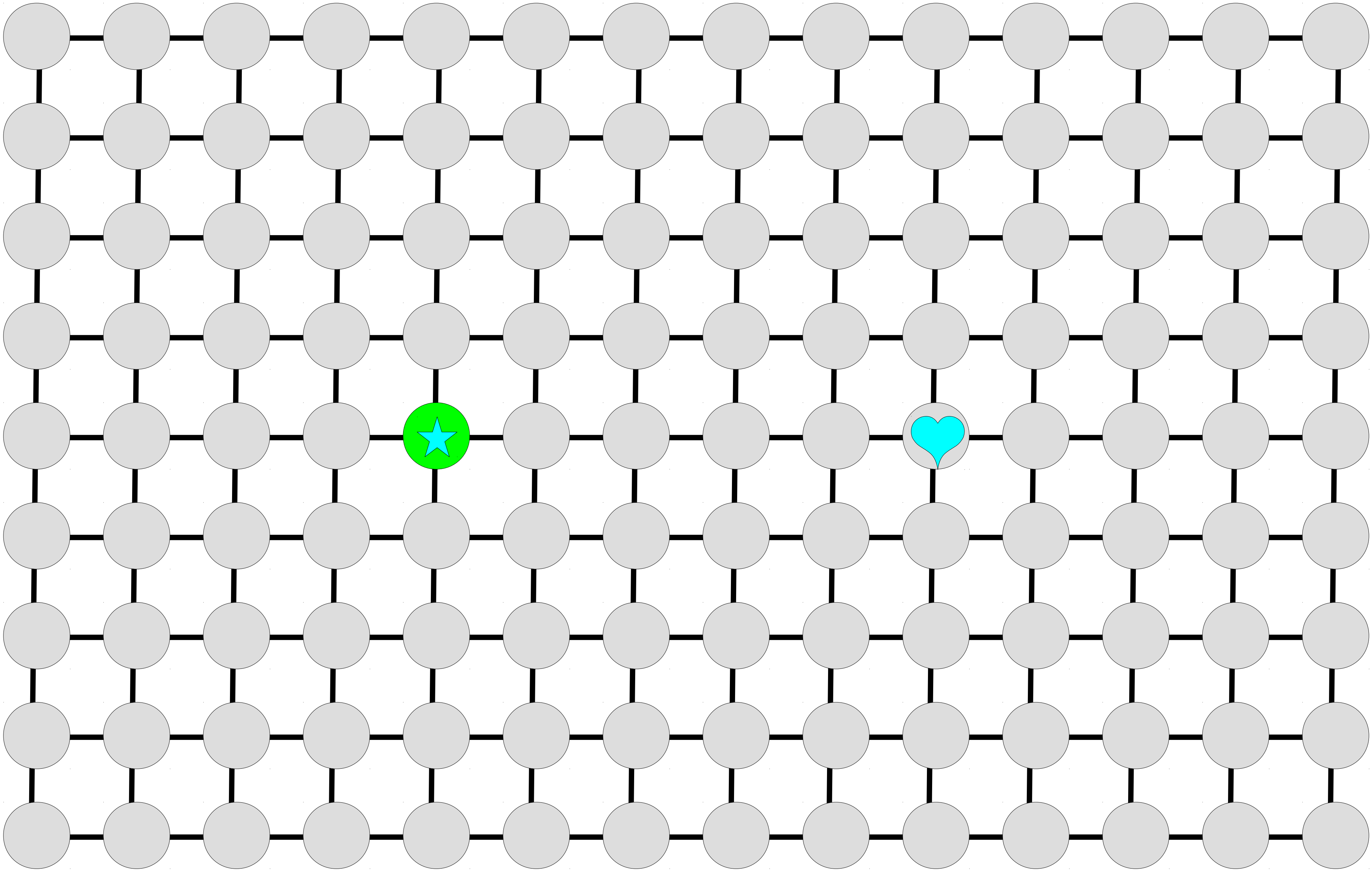
- Dijkstra's algorithm works by computing the shortest paths to lots of intermediary nodes in case they prove to be useful.
- Most of these nodes are in the completely wrong direction.
- Two questions:
  - What is Dijkstra thinking when it does this?
  - Can we get Dijkstra to change its mind?

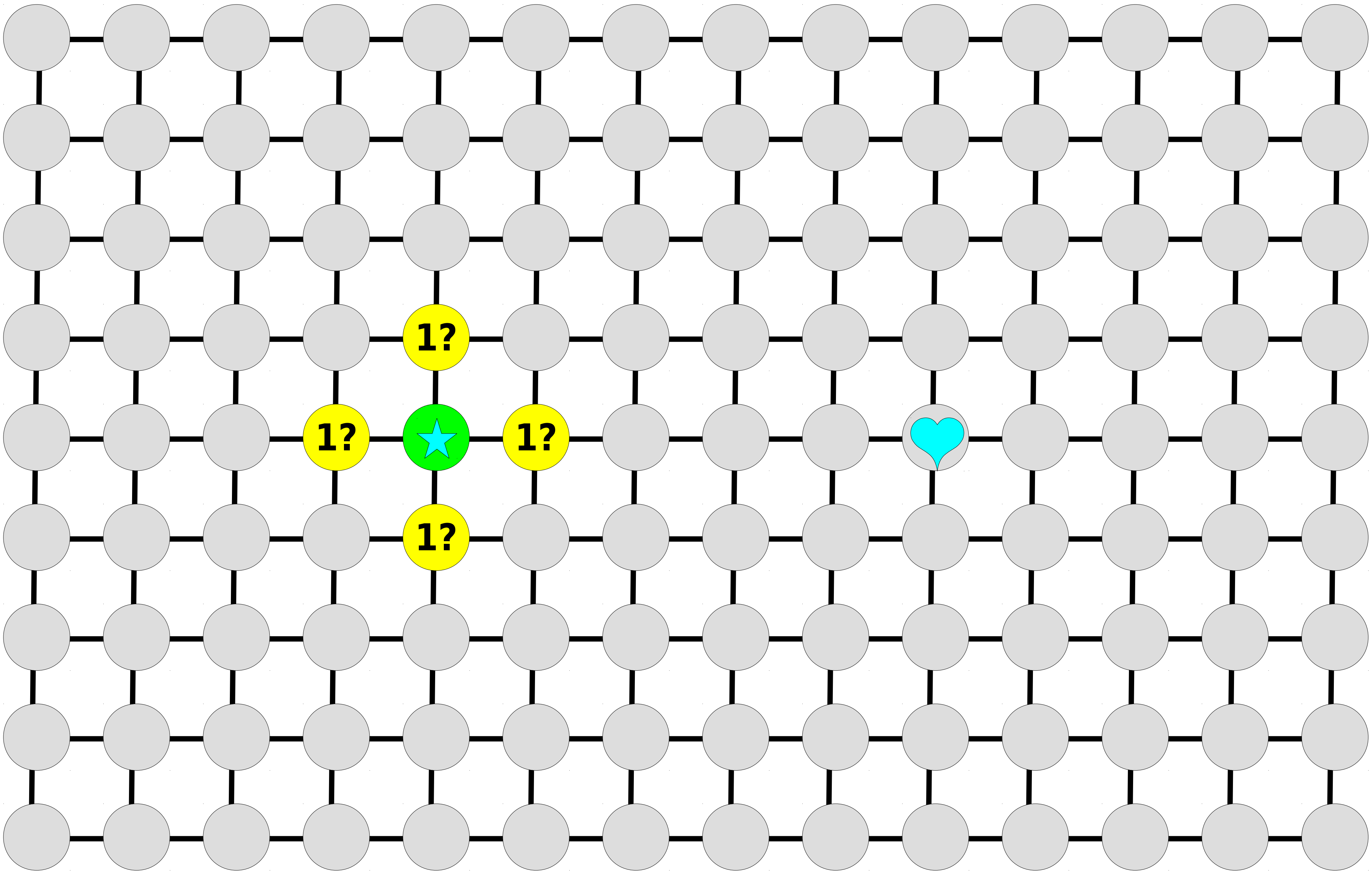


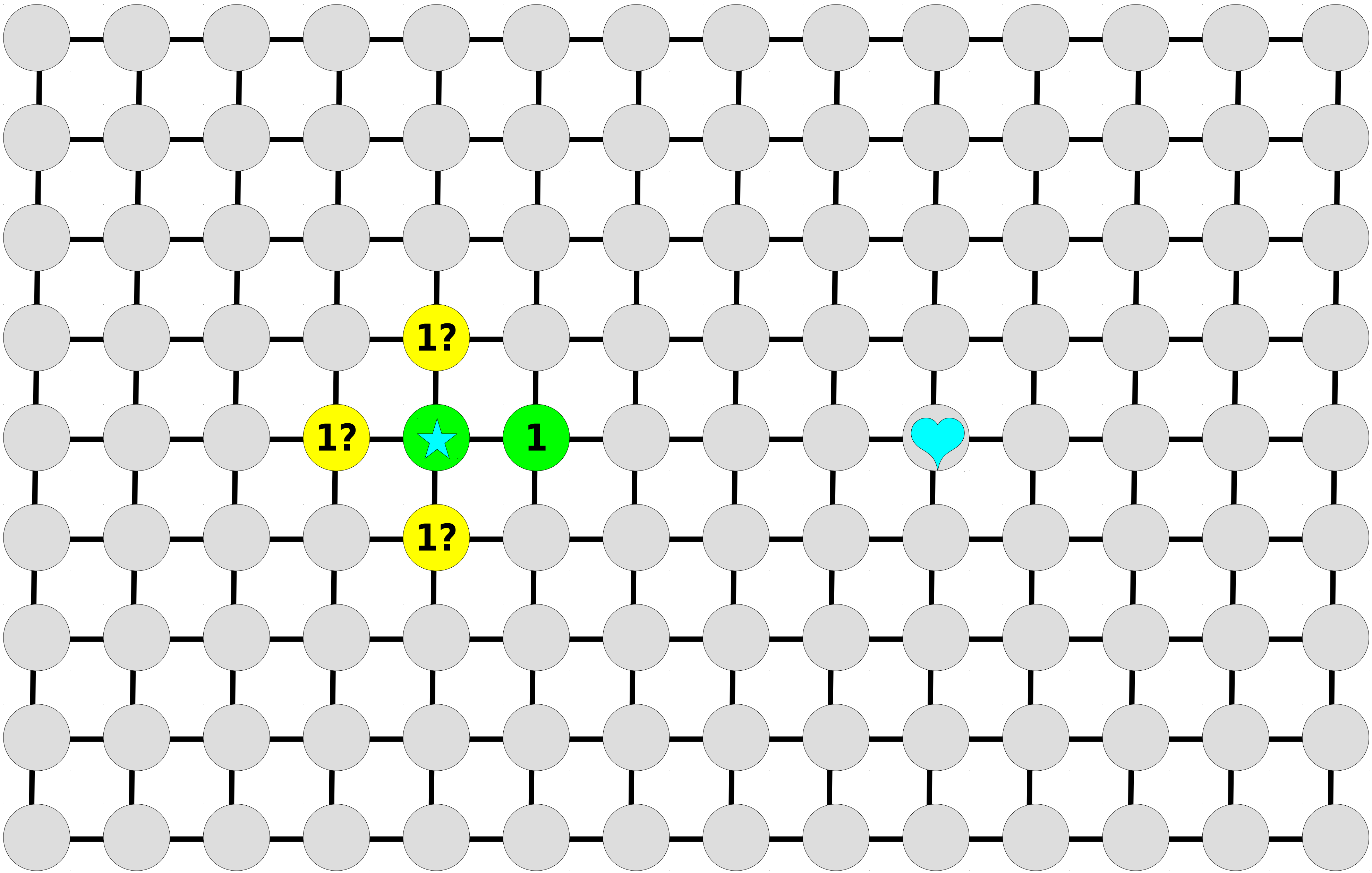


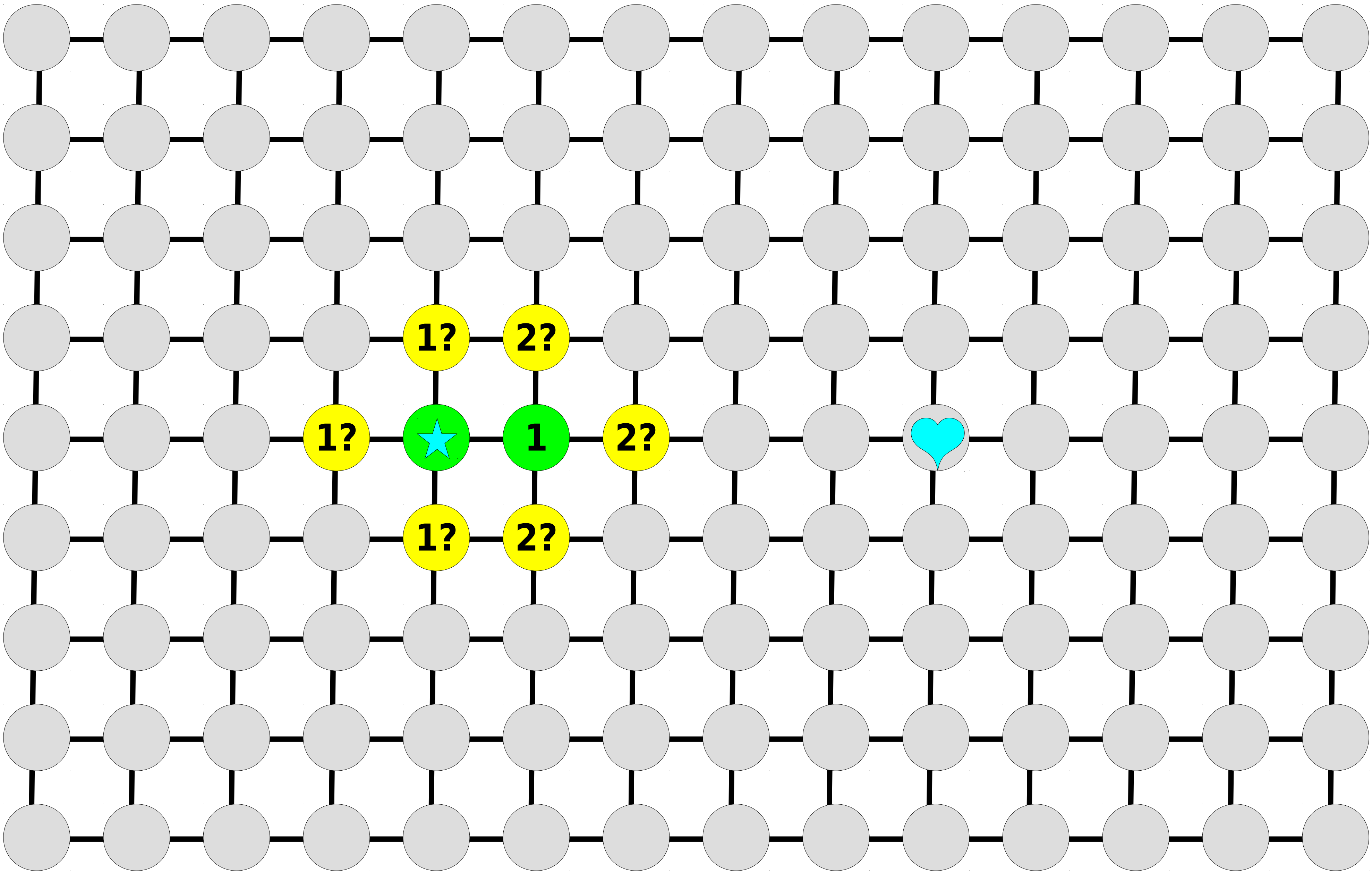


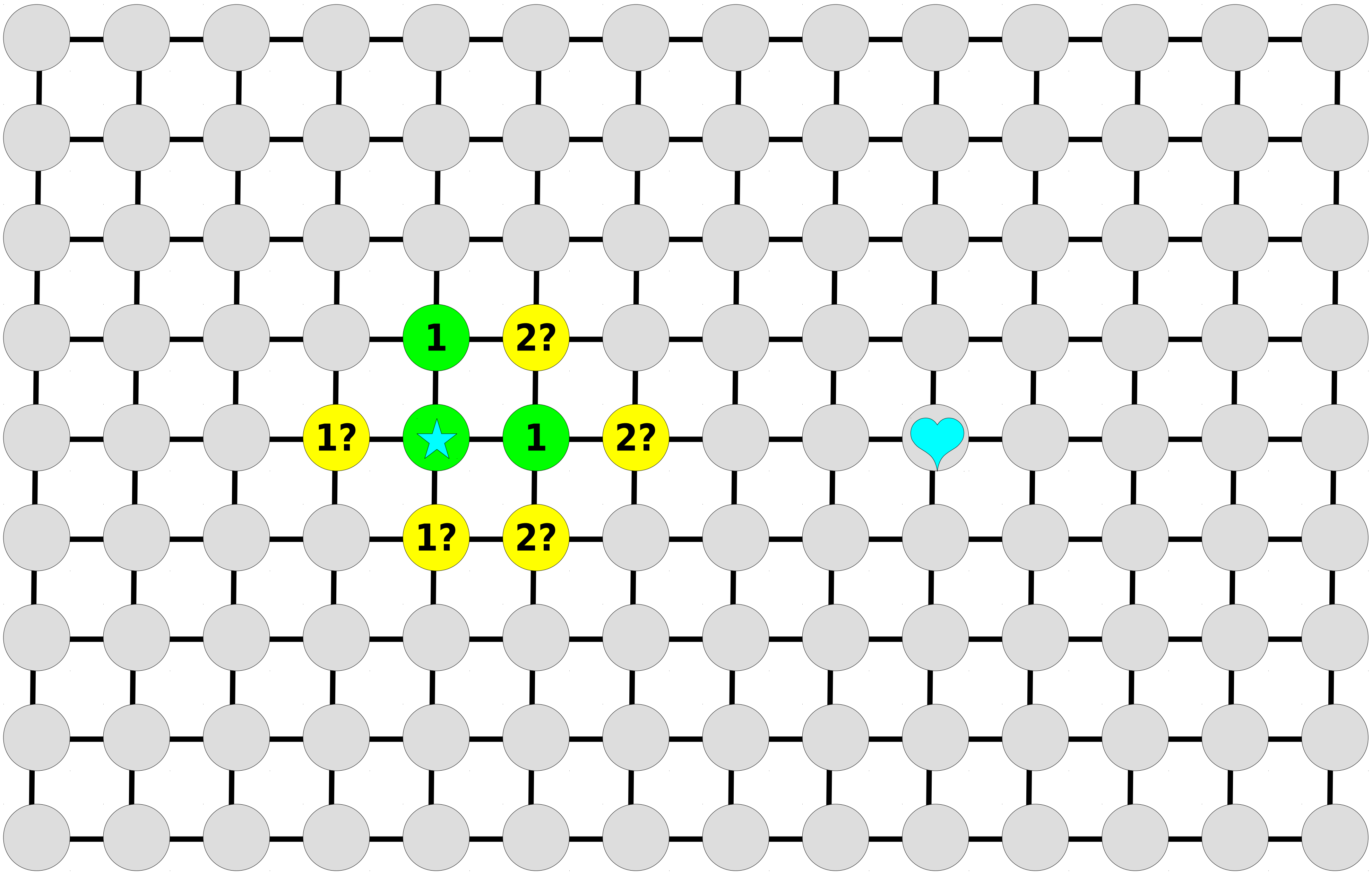


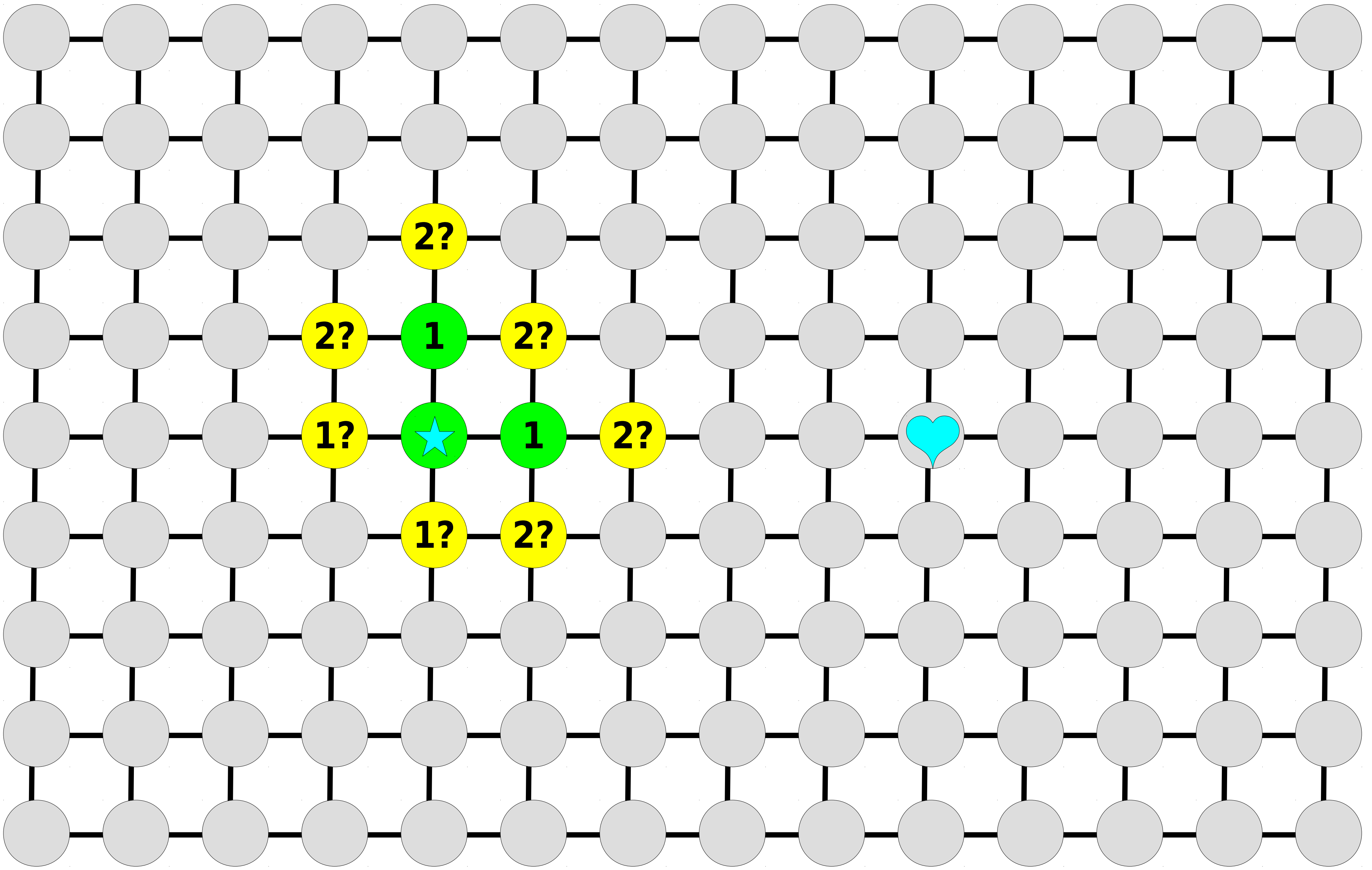


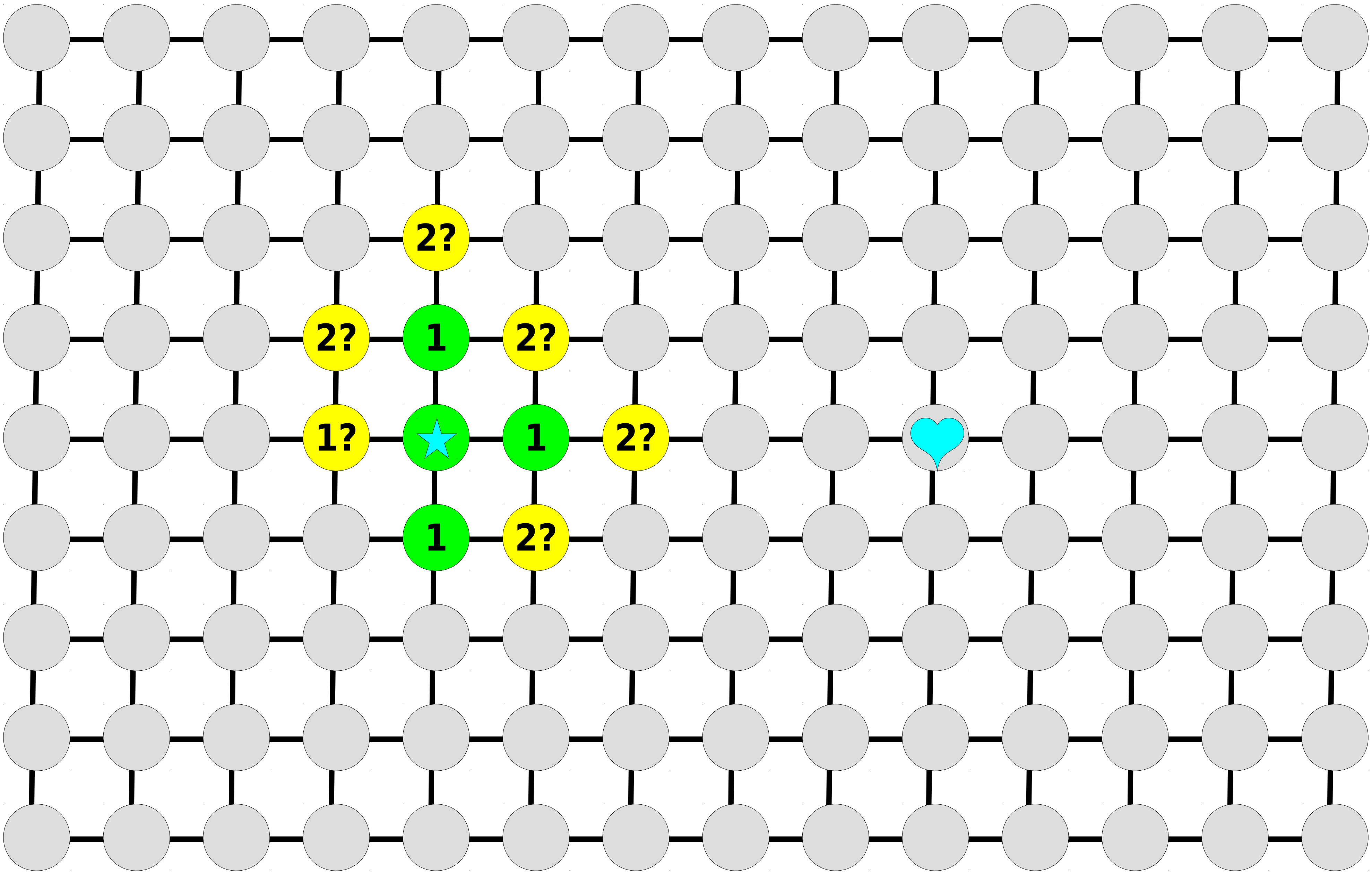

1?
1?
1?
1?


1?
1?
1
1?

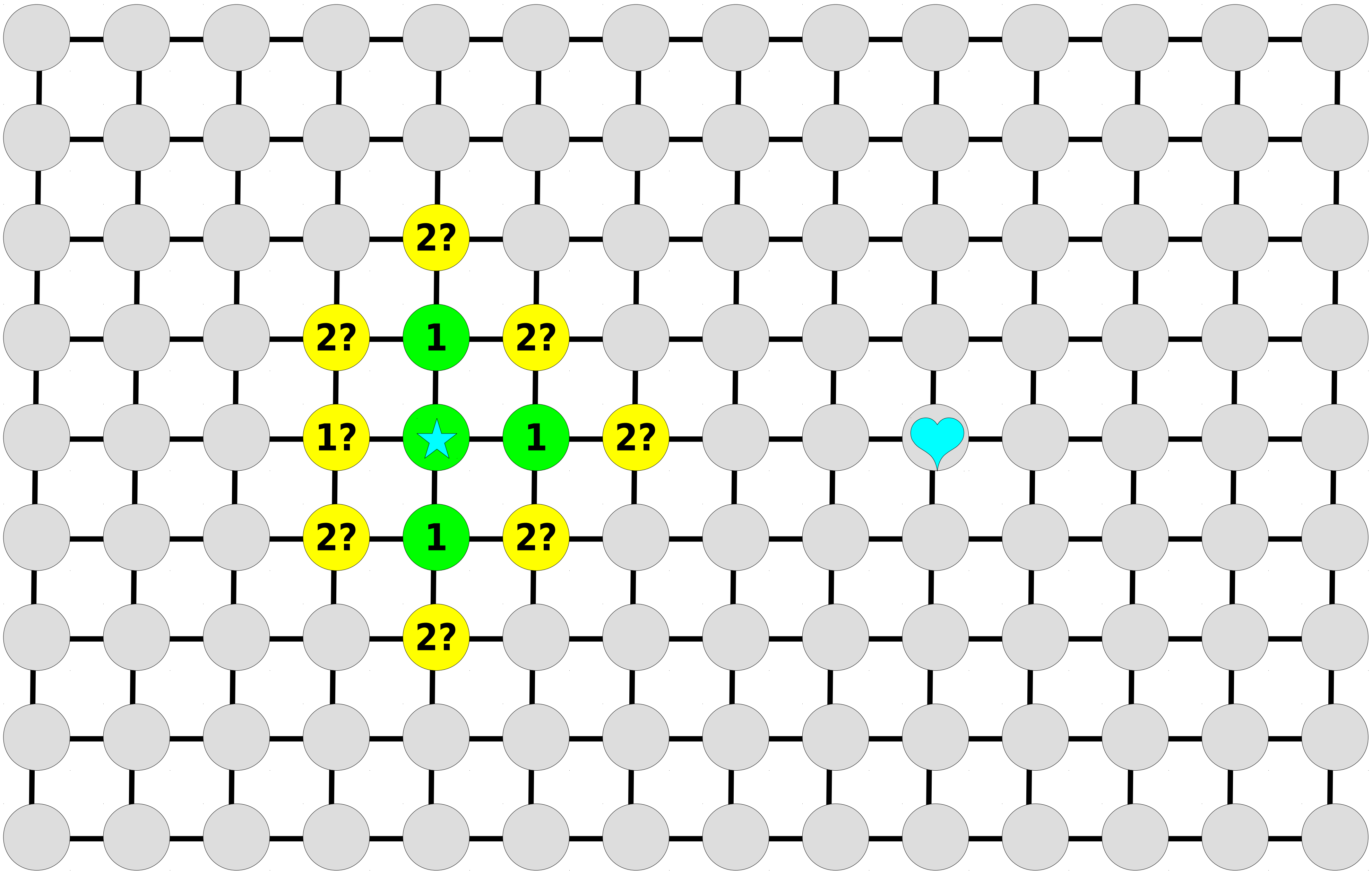

1?
2?
1?
1
2?
1?
2?


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

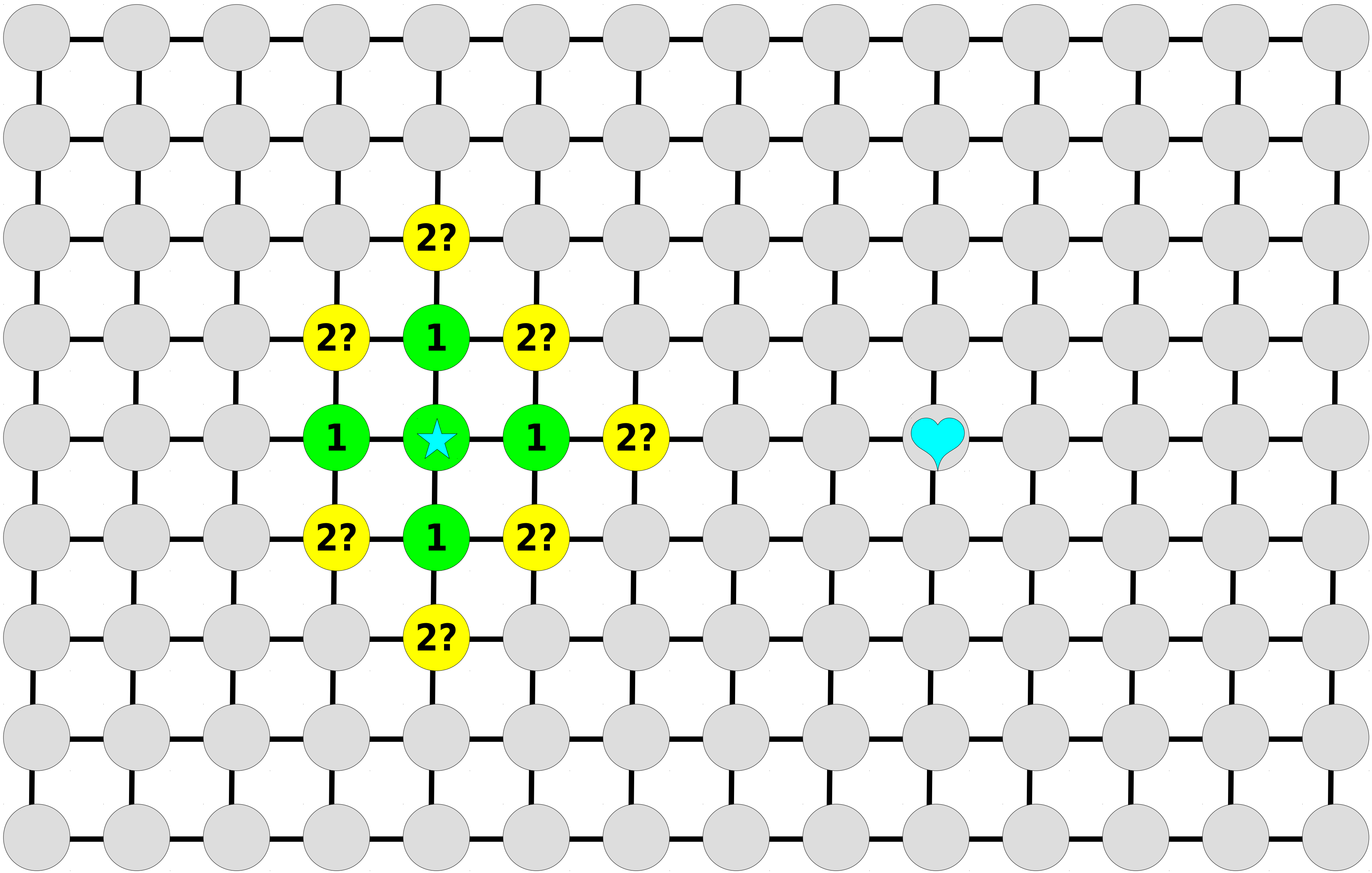

2?
2?
1
2?
1?
1
2?
1?
2?

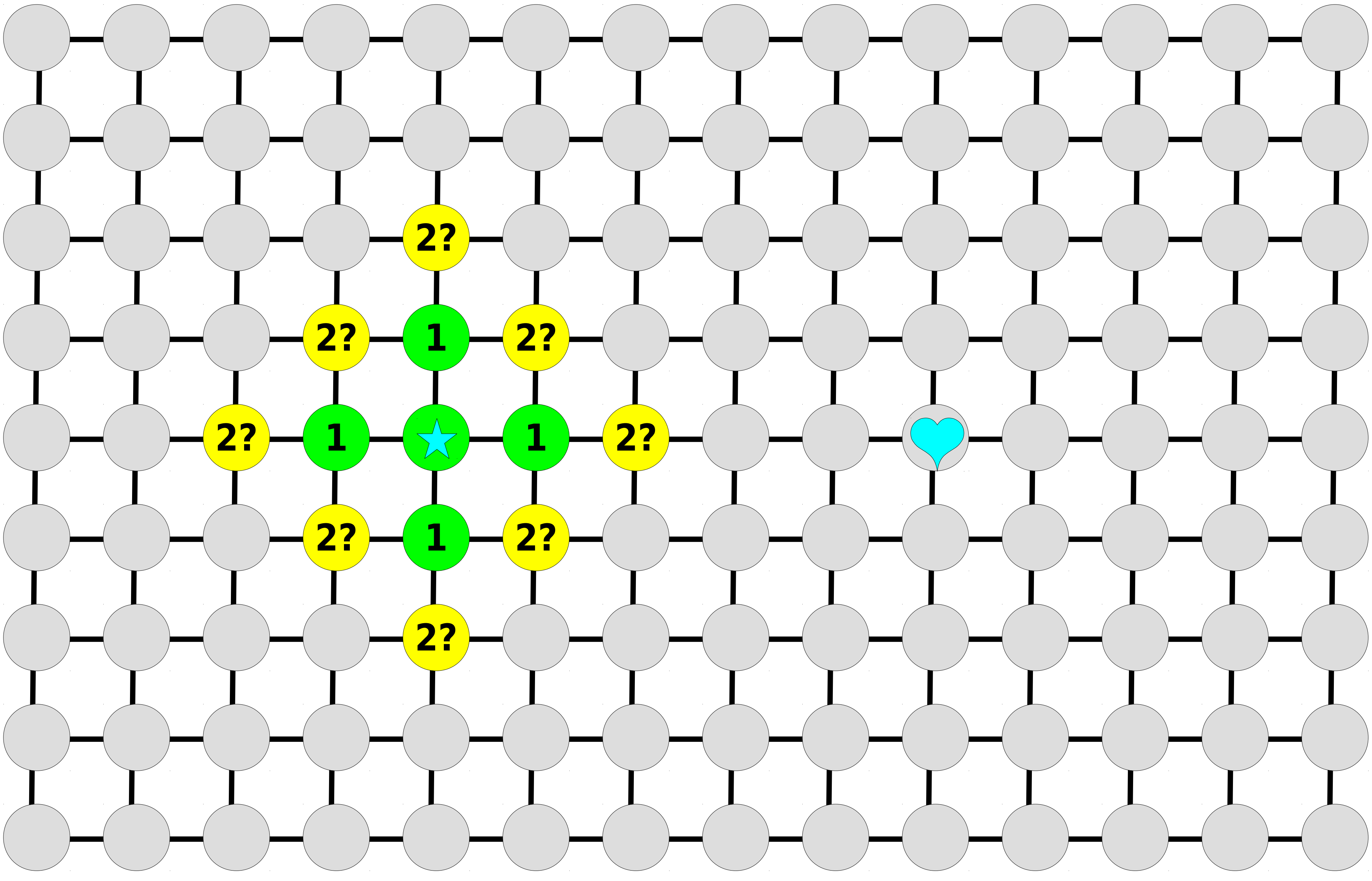


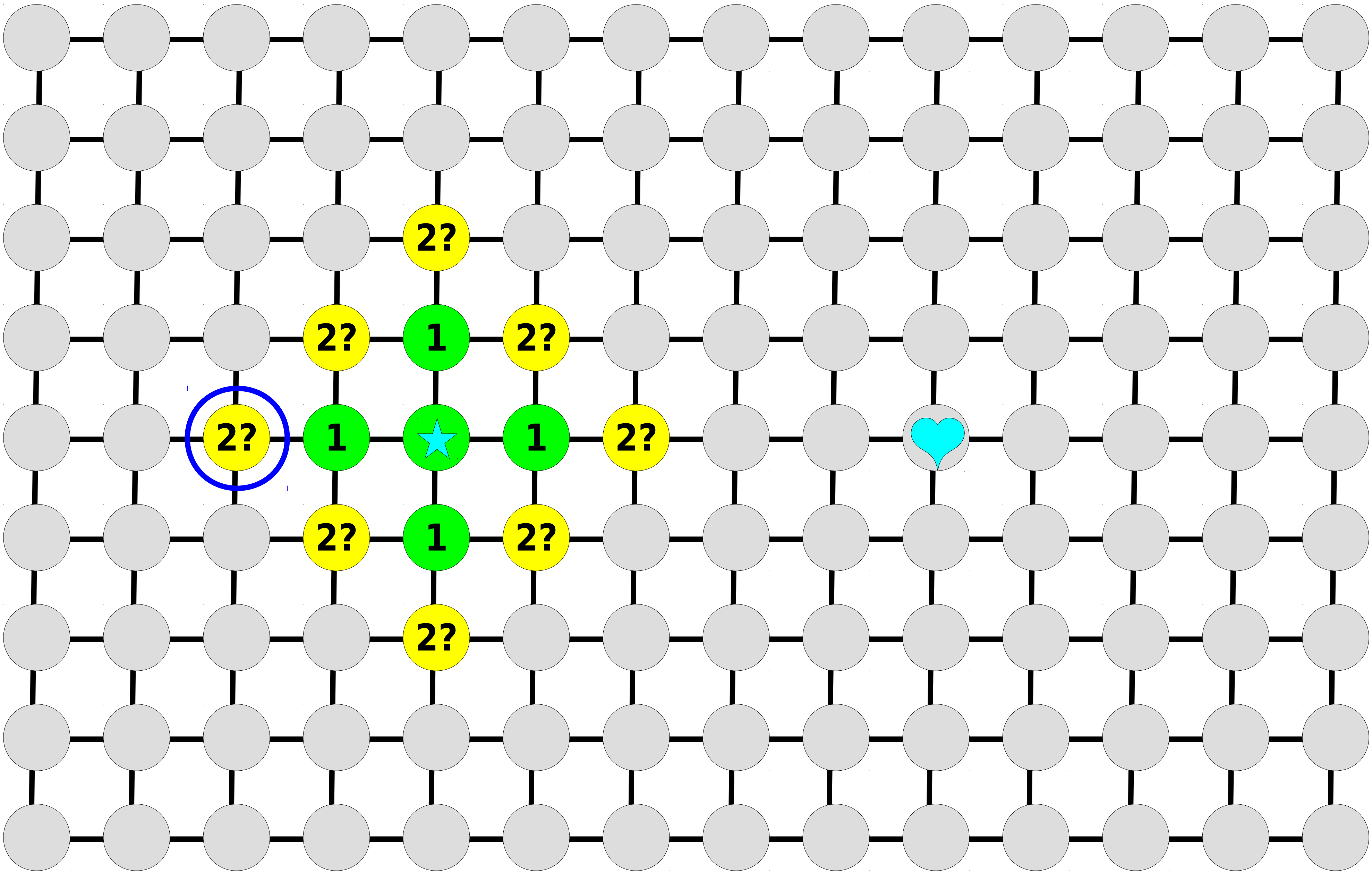
2?
2?
1
2?
1?
1
2?
1
2?



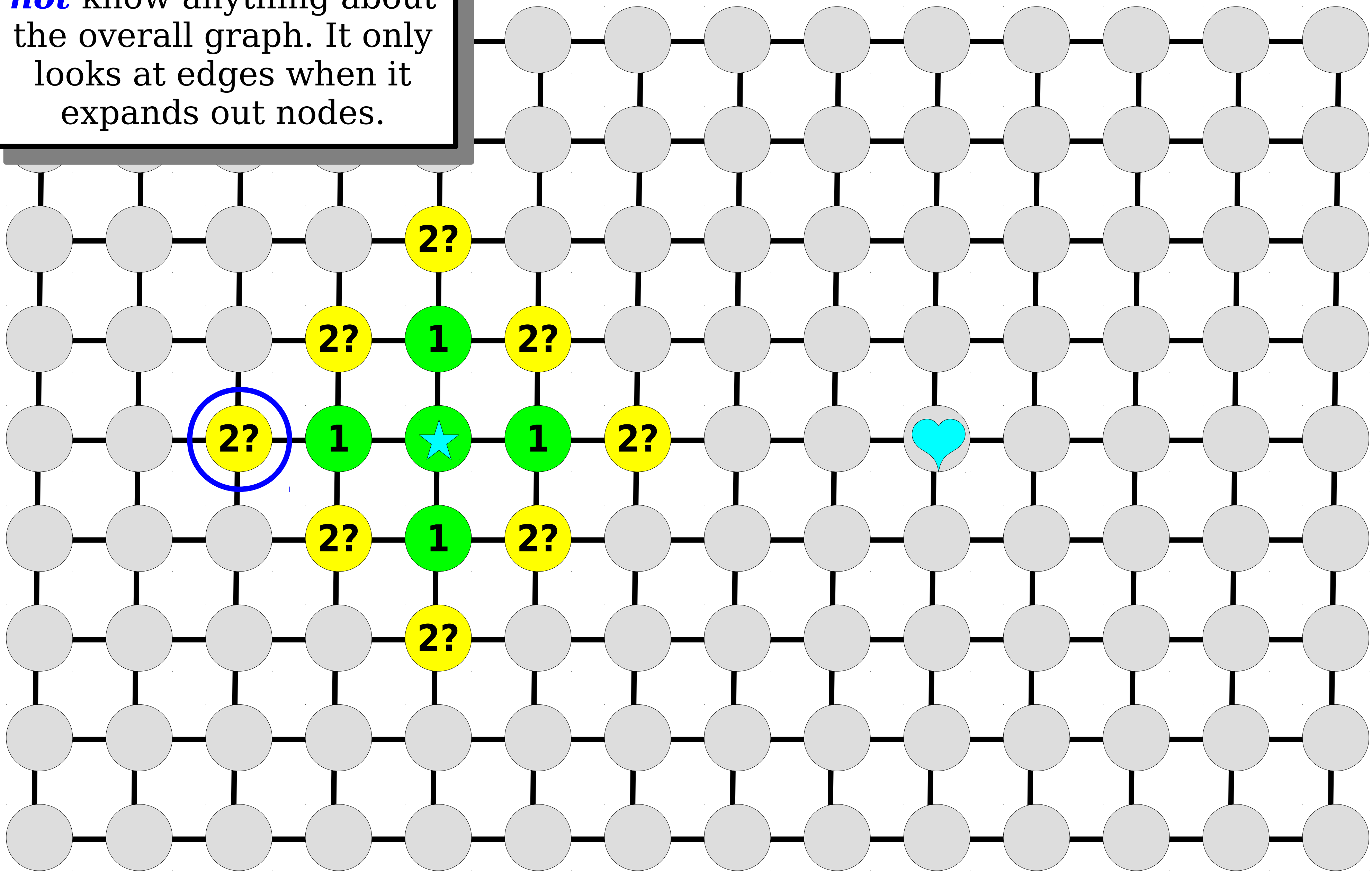
2?
2?
1
2?
1?
1
2?
2?
1
2?
2?

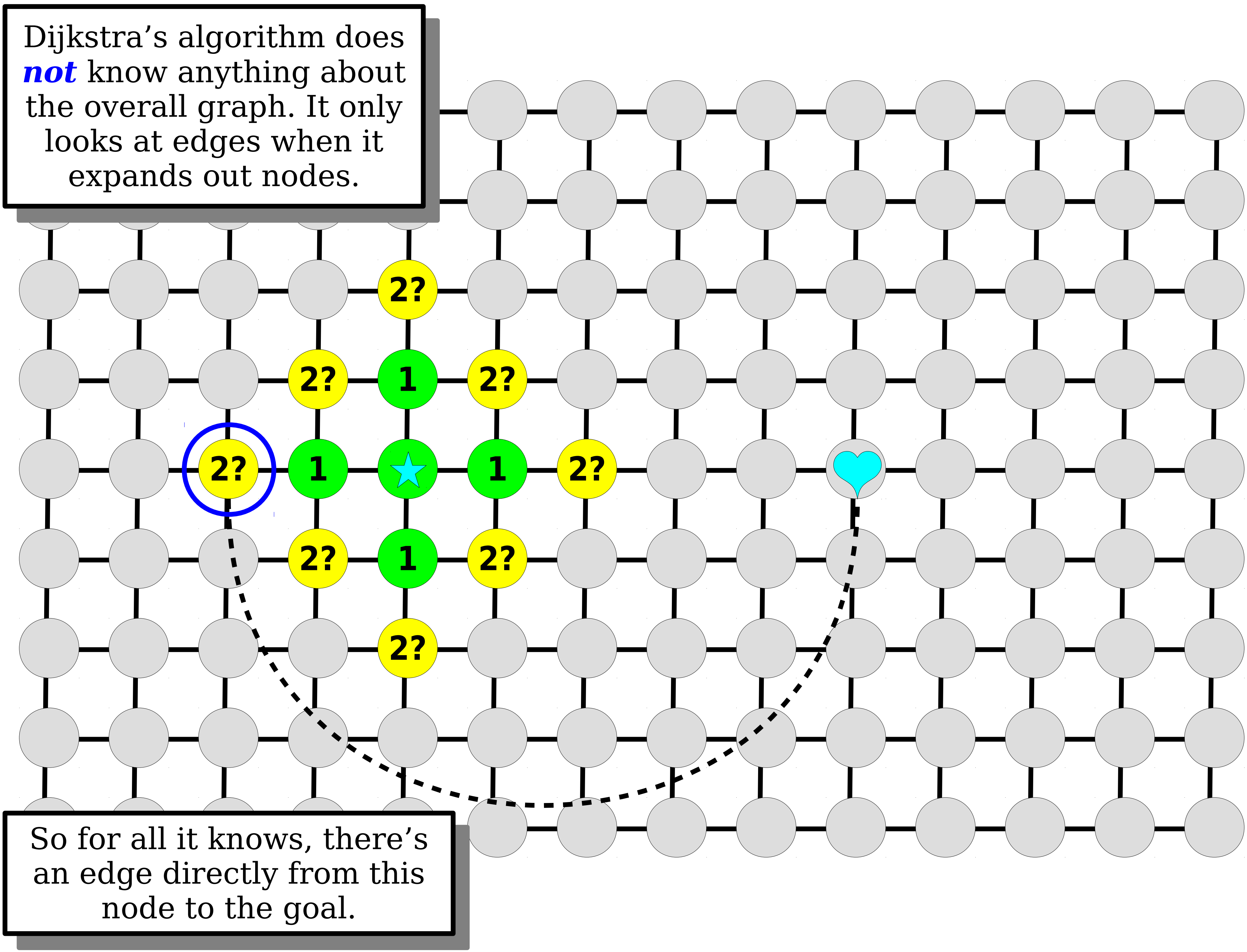

2?
2?
1
2?
1
1
2?
2?
1
2?
2?

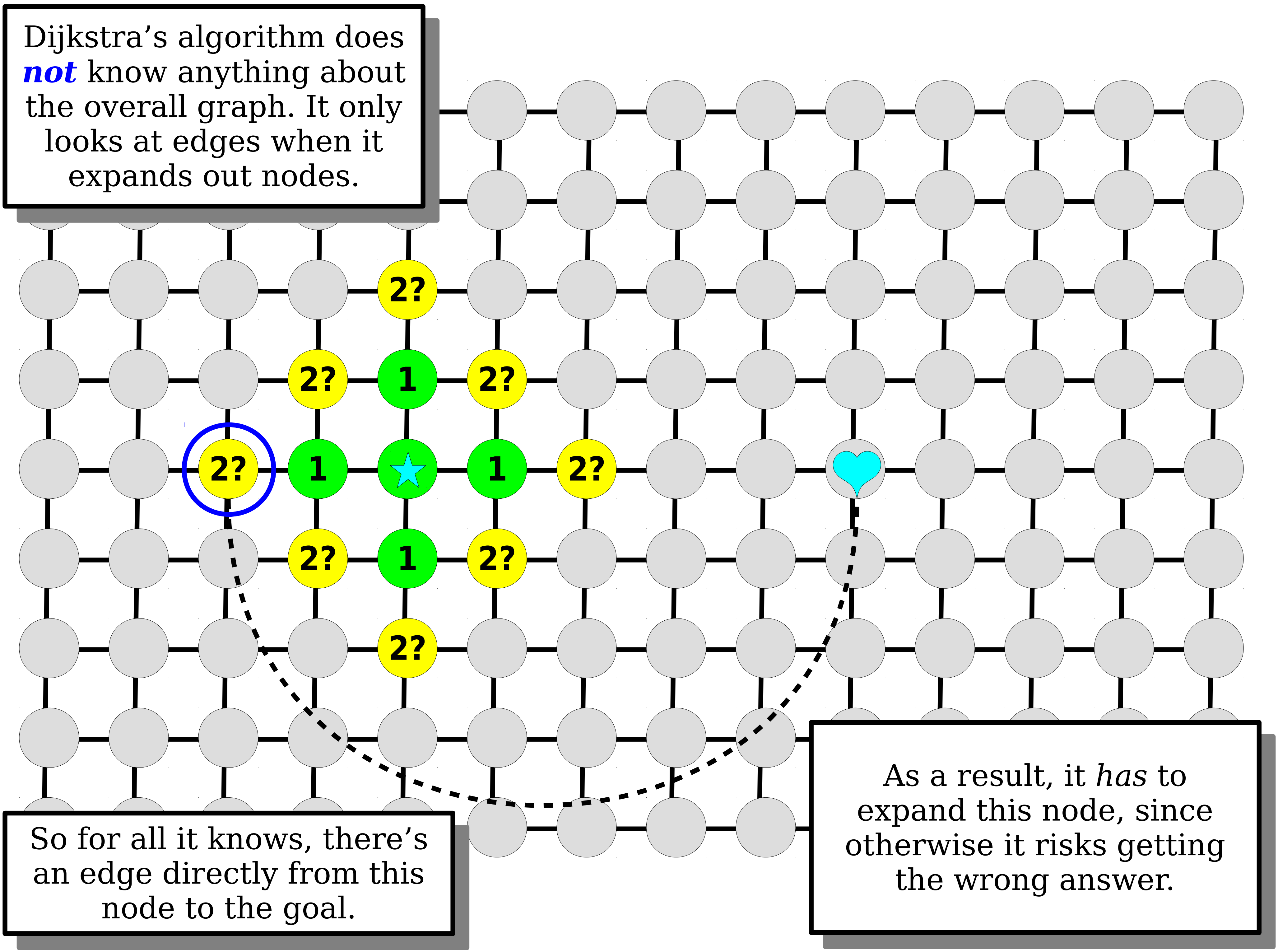

2?
2?
1
2?
2?
1
1
2?
2?
1
2?
2?

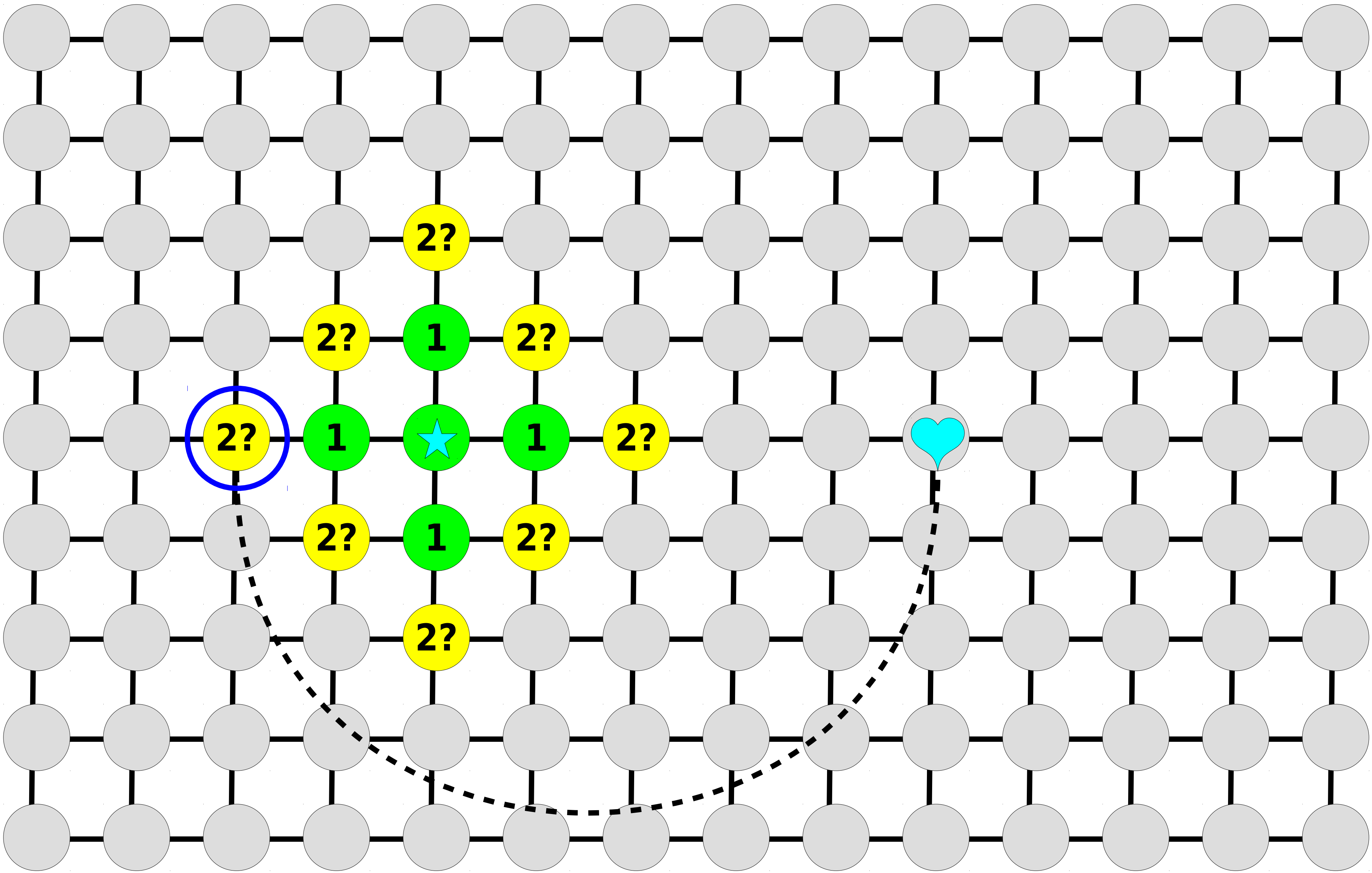

2?
2?
1
2?
2?
1
1
2?
2?
1
2?
2?

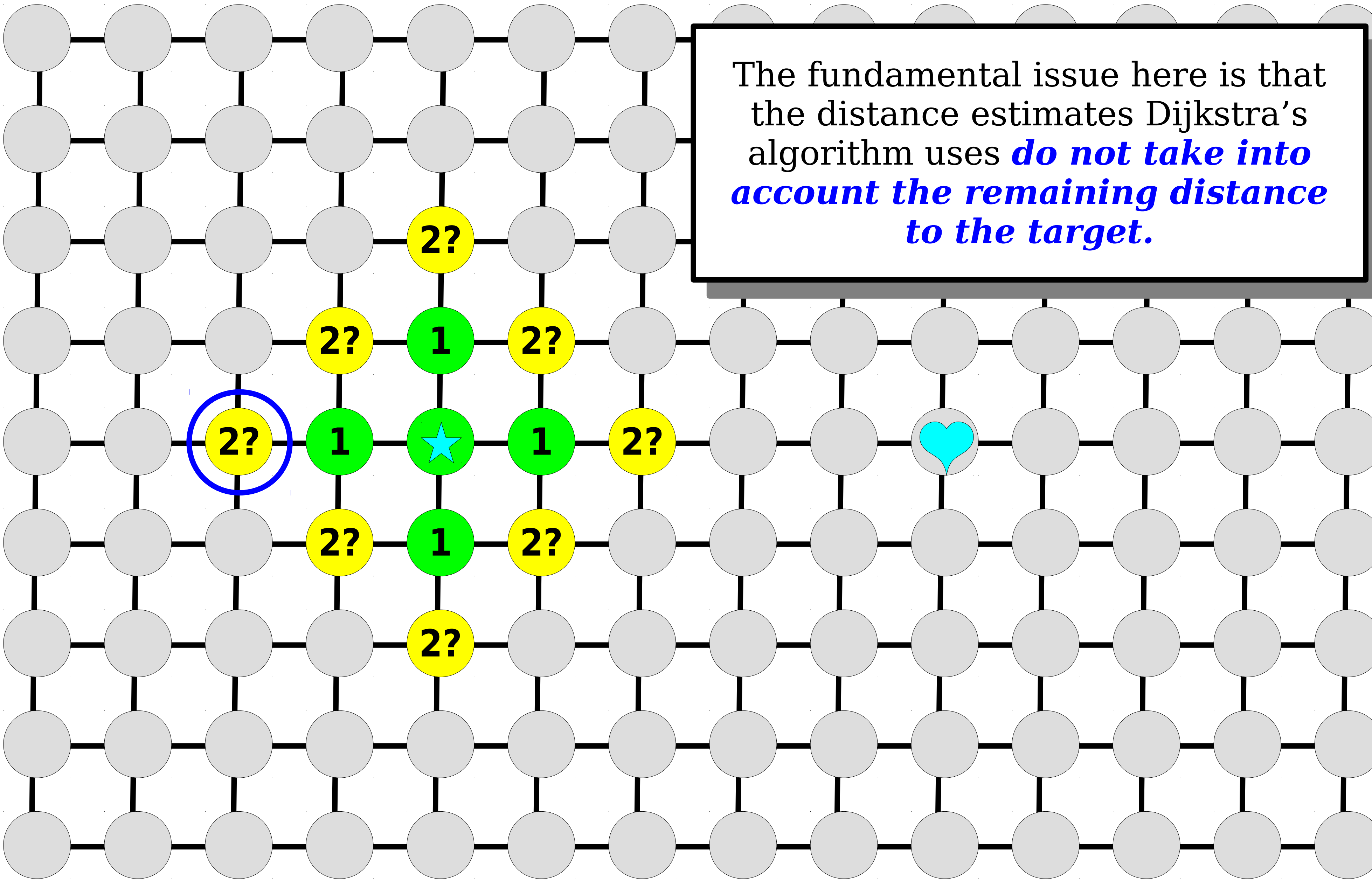
Dijkstra's algorithm does ***not*** know anything about the overall graph. It only looks at edges when it expands out nodes.




Dijkstra’s algorithm does *not* know anything about the overall graph. It only looks at edges when it expands out nodes.
2?
2?
1
2?
2?
1
1
2?
2?
1
2?
2?
So for all it knows, there’s an edge directly from this node to the goal.

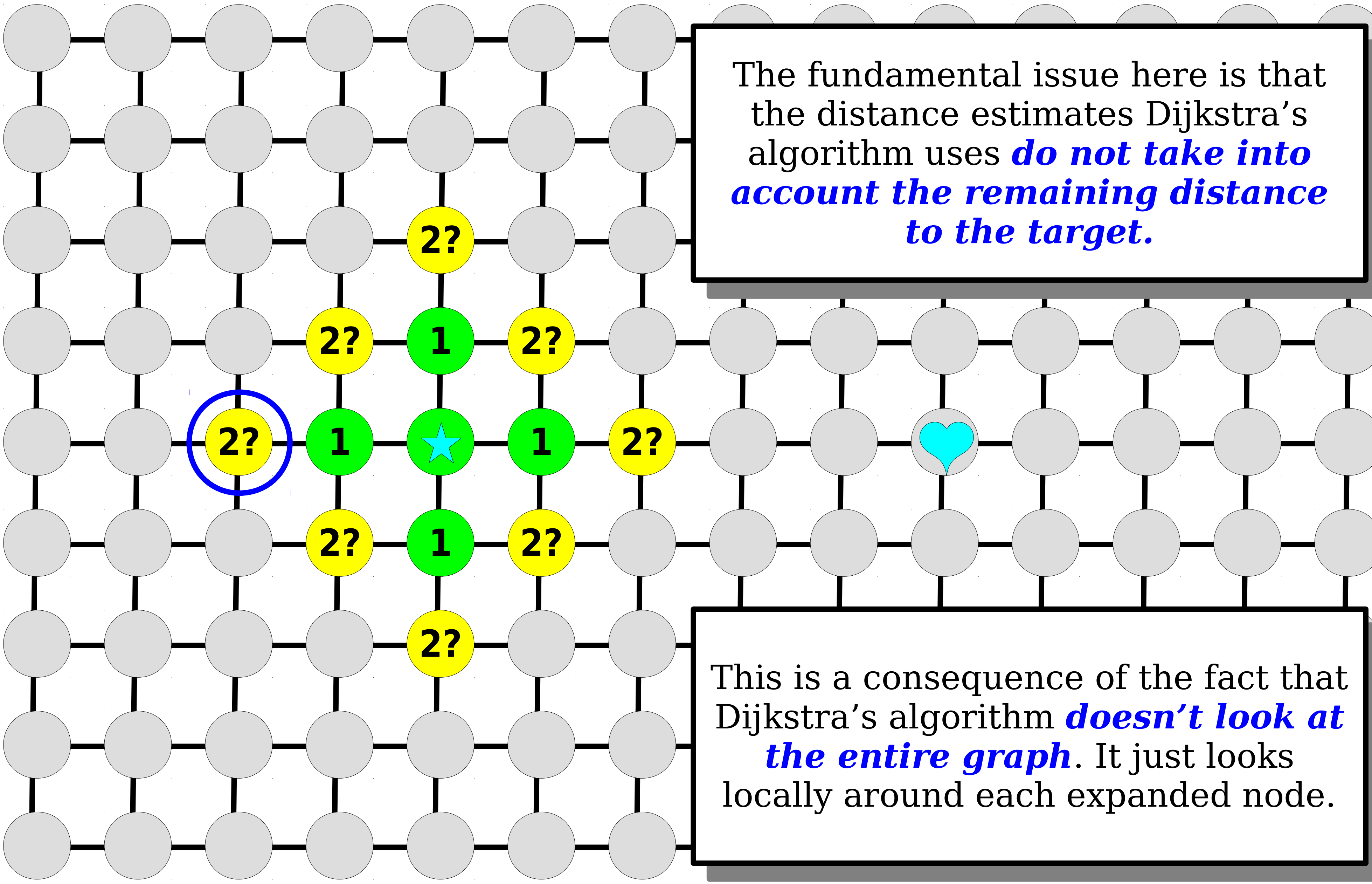

Dijkstra's algorithm does *not* know anything about the overall graph. It only looks at edges when it expands out nodes.
2?
2?
1
2?
2?
1
1
2?
2?
1
2?
2?
So for all it knows, there's an edge directly from this node to the goal.
As a result, it *has* to expand this node, since otherwise it risks getting the wrong answer.


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



The fundamental issue here is that the distance estimates Dijkstra’s algorithm uses ***do not take into account the remaining distance to the target.***

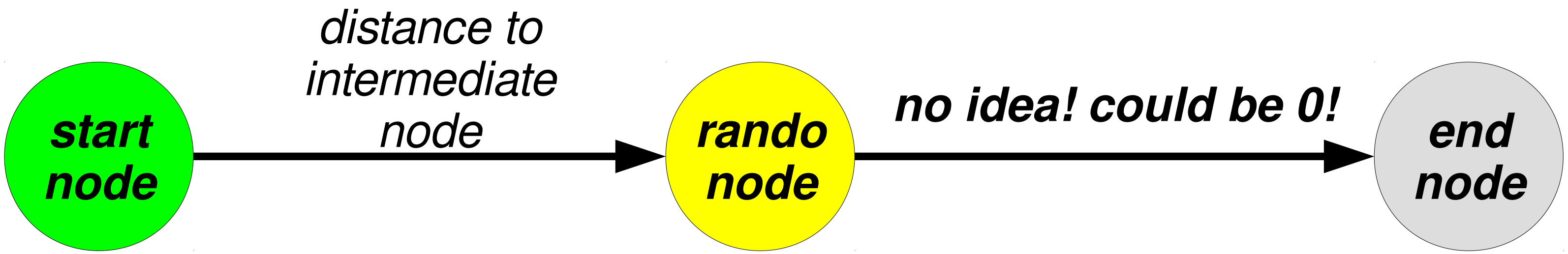
The fundamental issue here is that the distance estimates Dijkstra’s algorithm uses ***do not take into account the remaining distance to the target.***



This is a consequence of the fact that Dijkstra’s algorithm ***doesn’t look at the entire graph***. It just looks locally around each expanded node.

To Recap

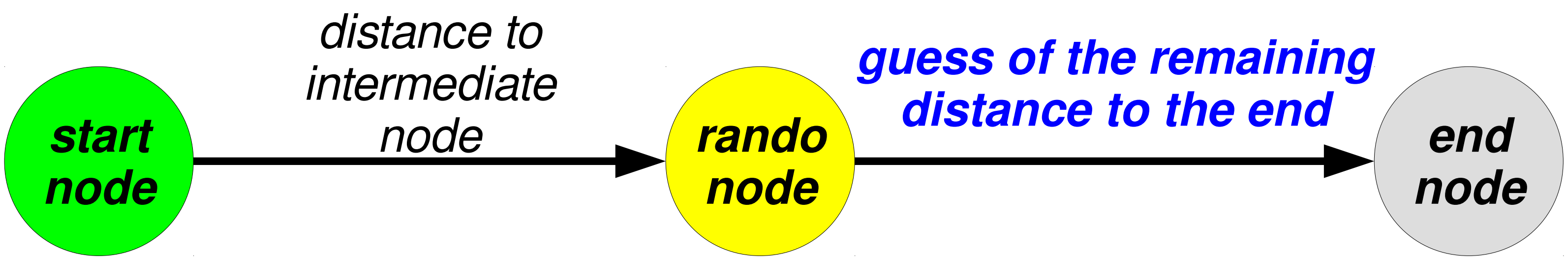
- When Dijkstra's algorithm sees a yellow node, it's "nervous" that there might be a free path from there to the destination:



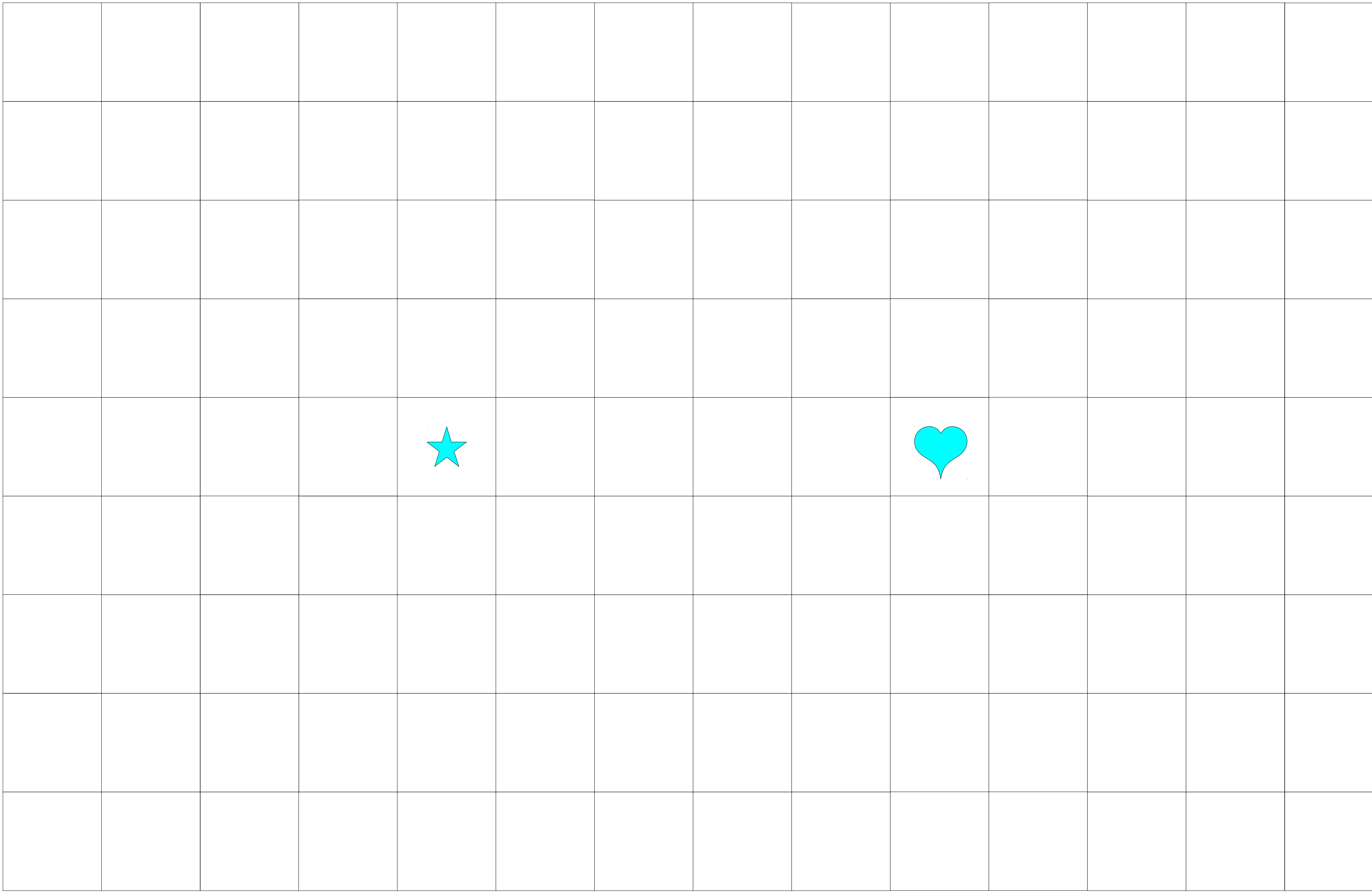
- ***Idea:*** What if we gave the algorithm some more information about how far away the end node really is?

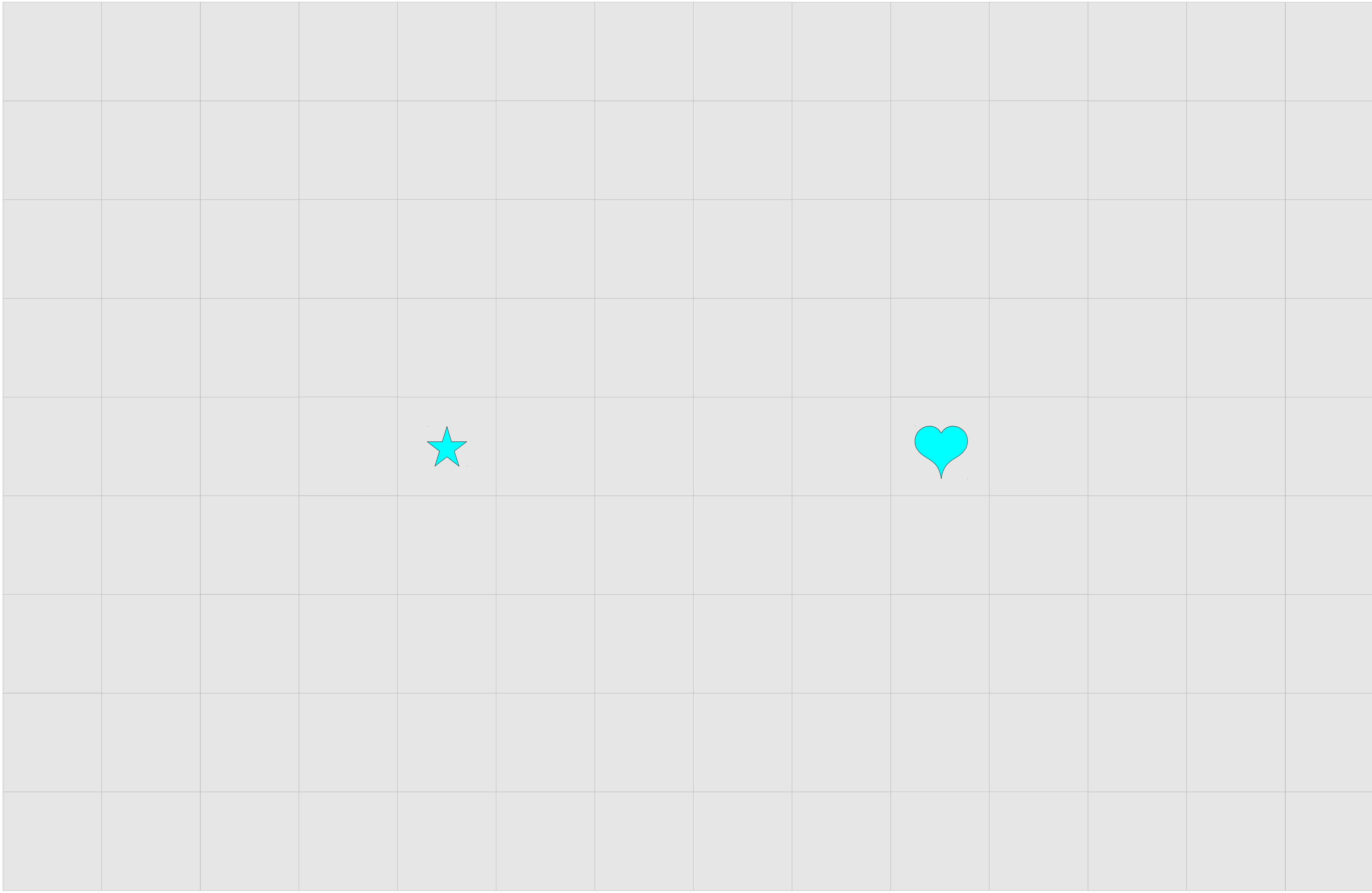
To Recap

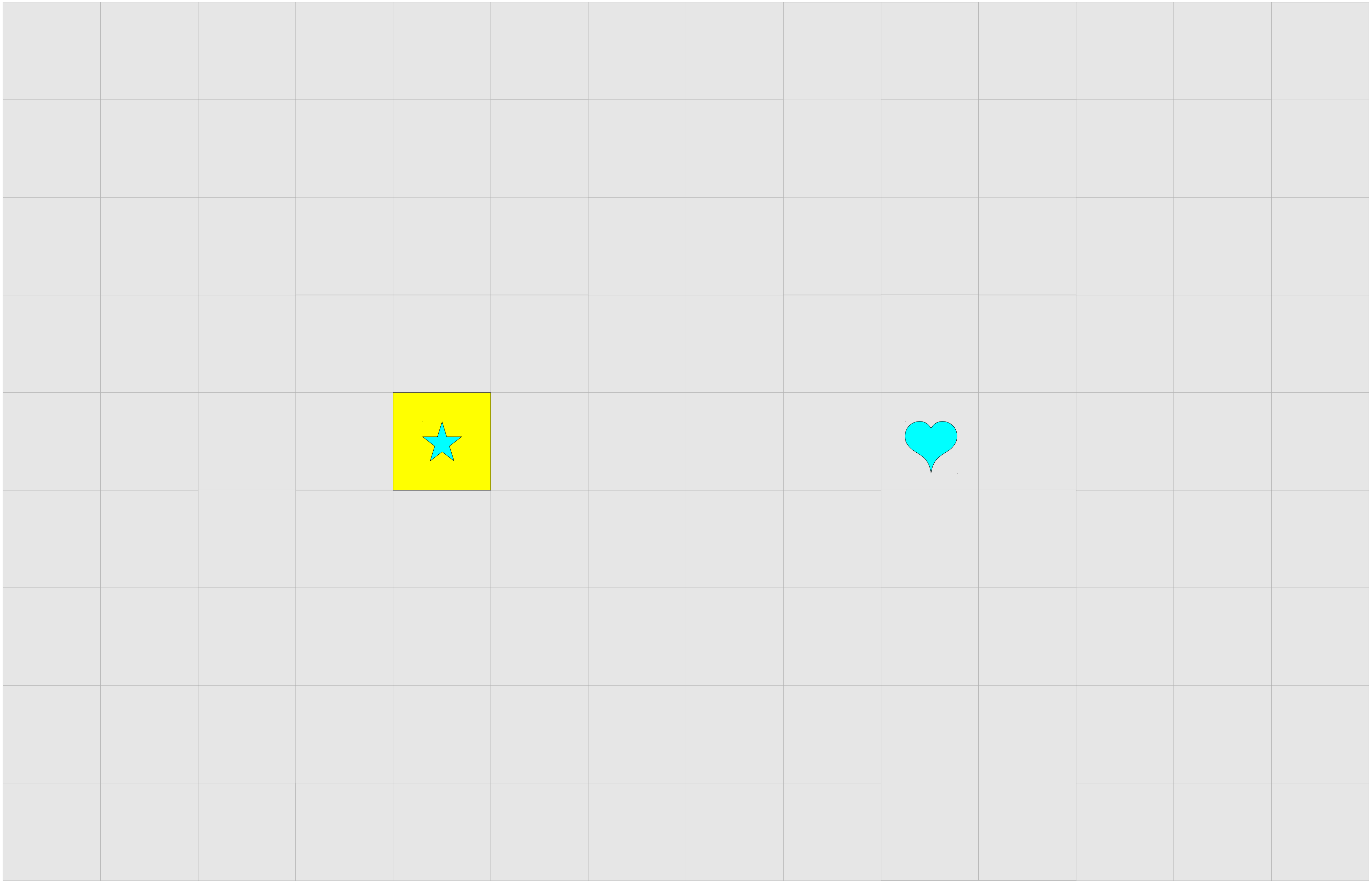
- When Dijkstra's algorithm sees a yellow node, it's "nervous" that there might be a free path from there to the destination:

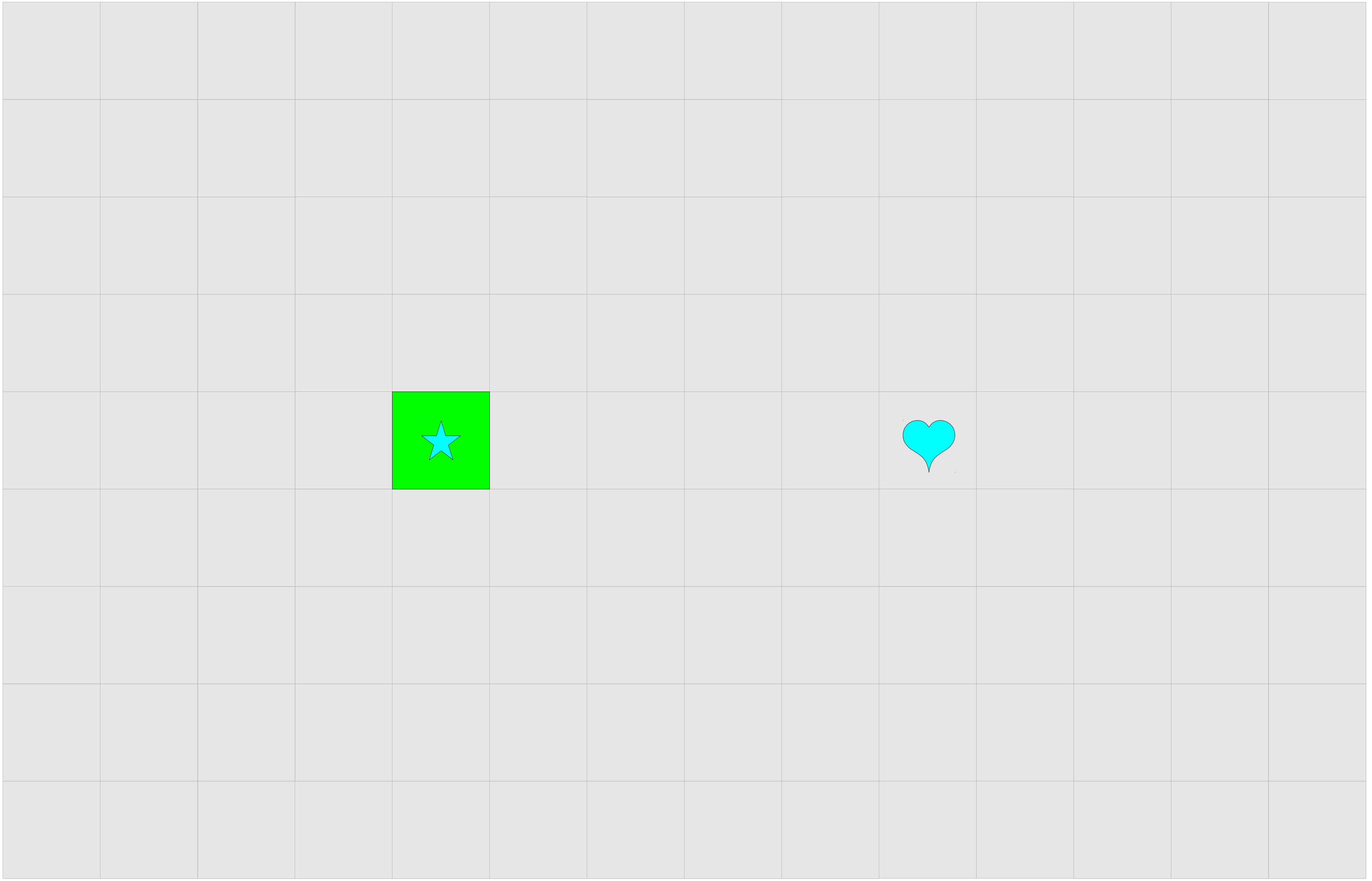


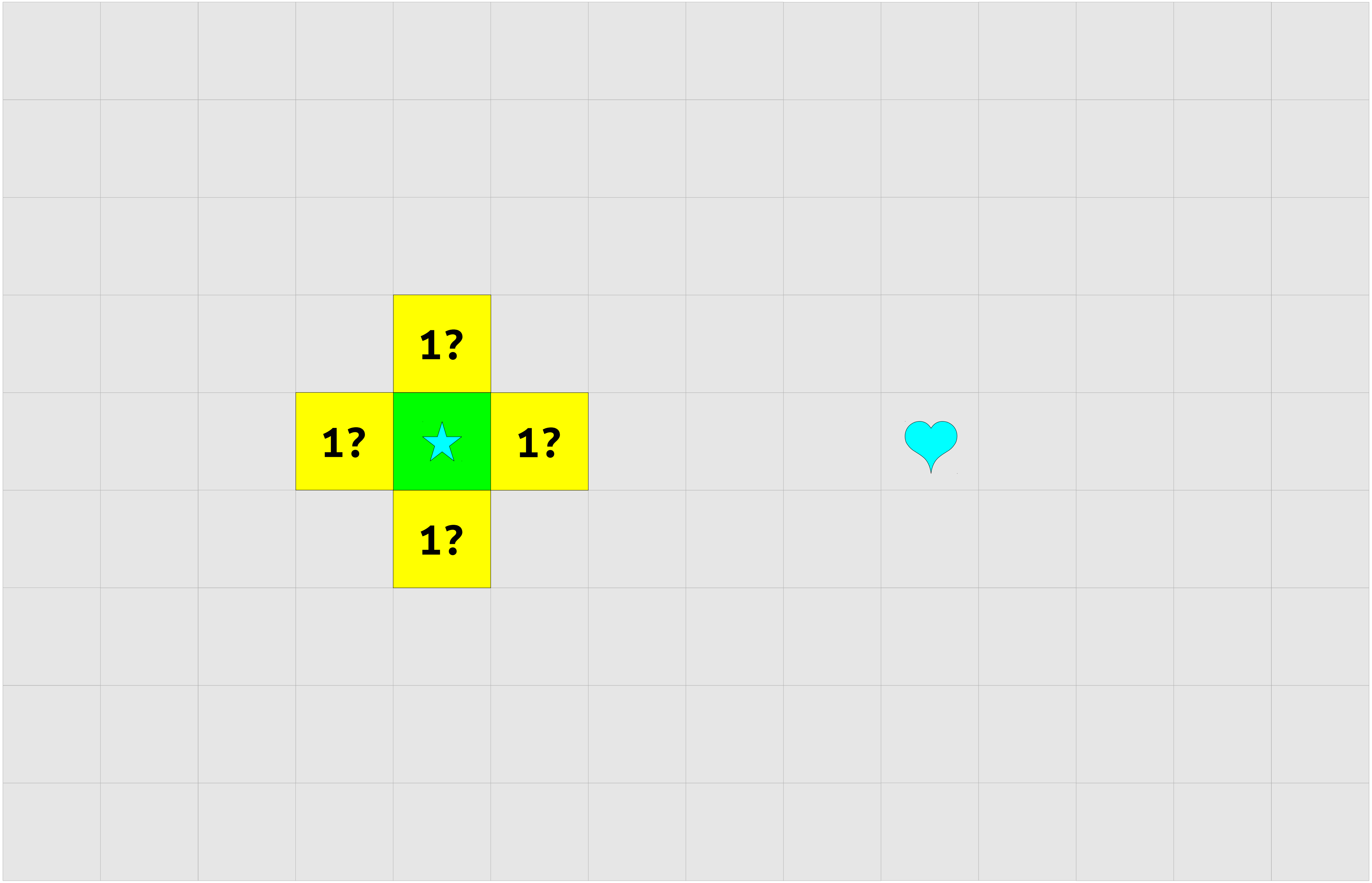
- ***Idea:*** What if we gave the algorithm some more information about how far away the end node really is?

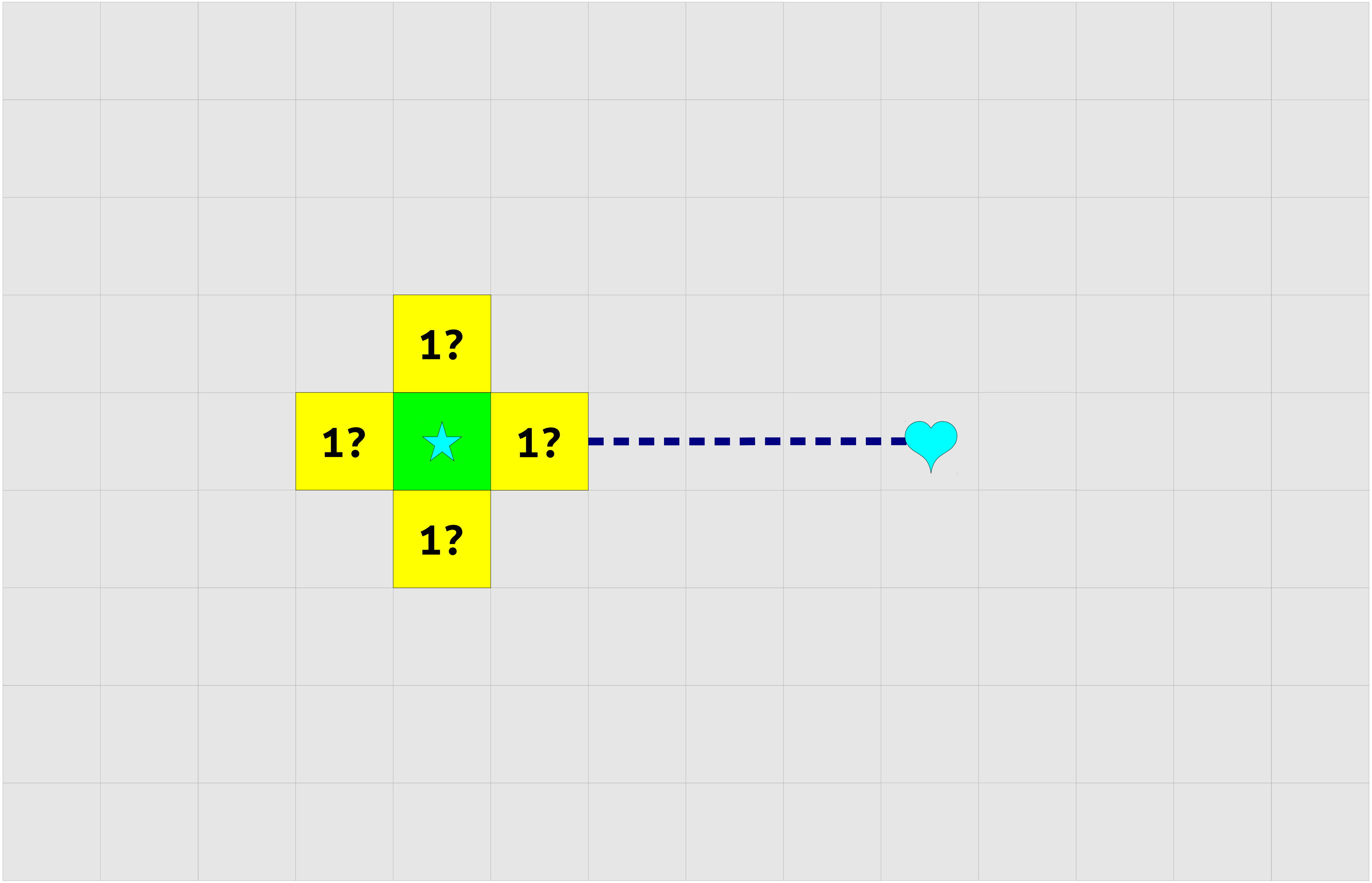


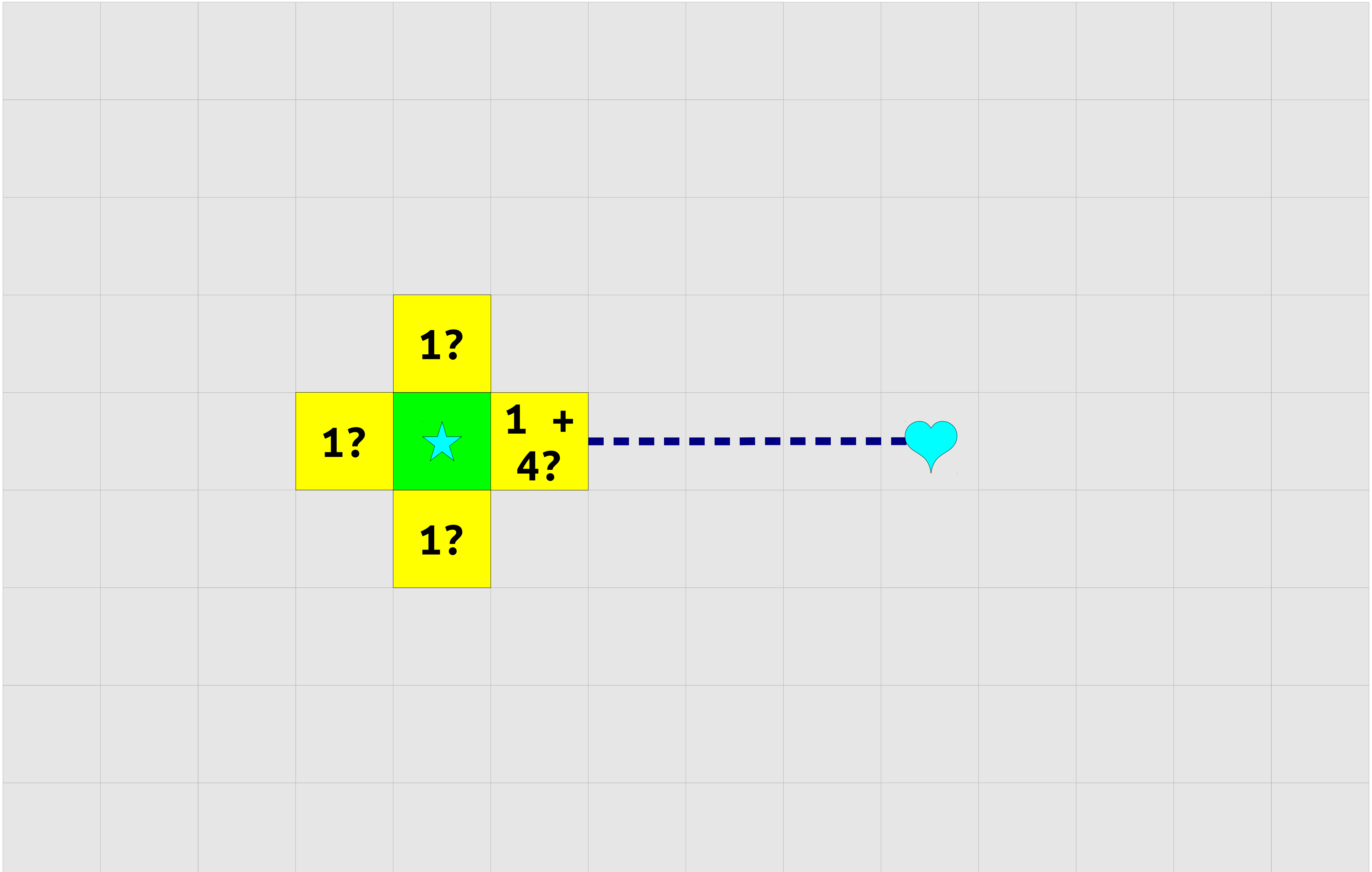


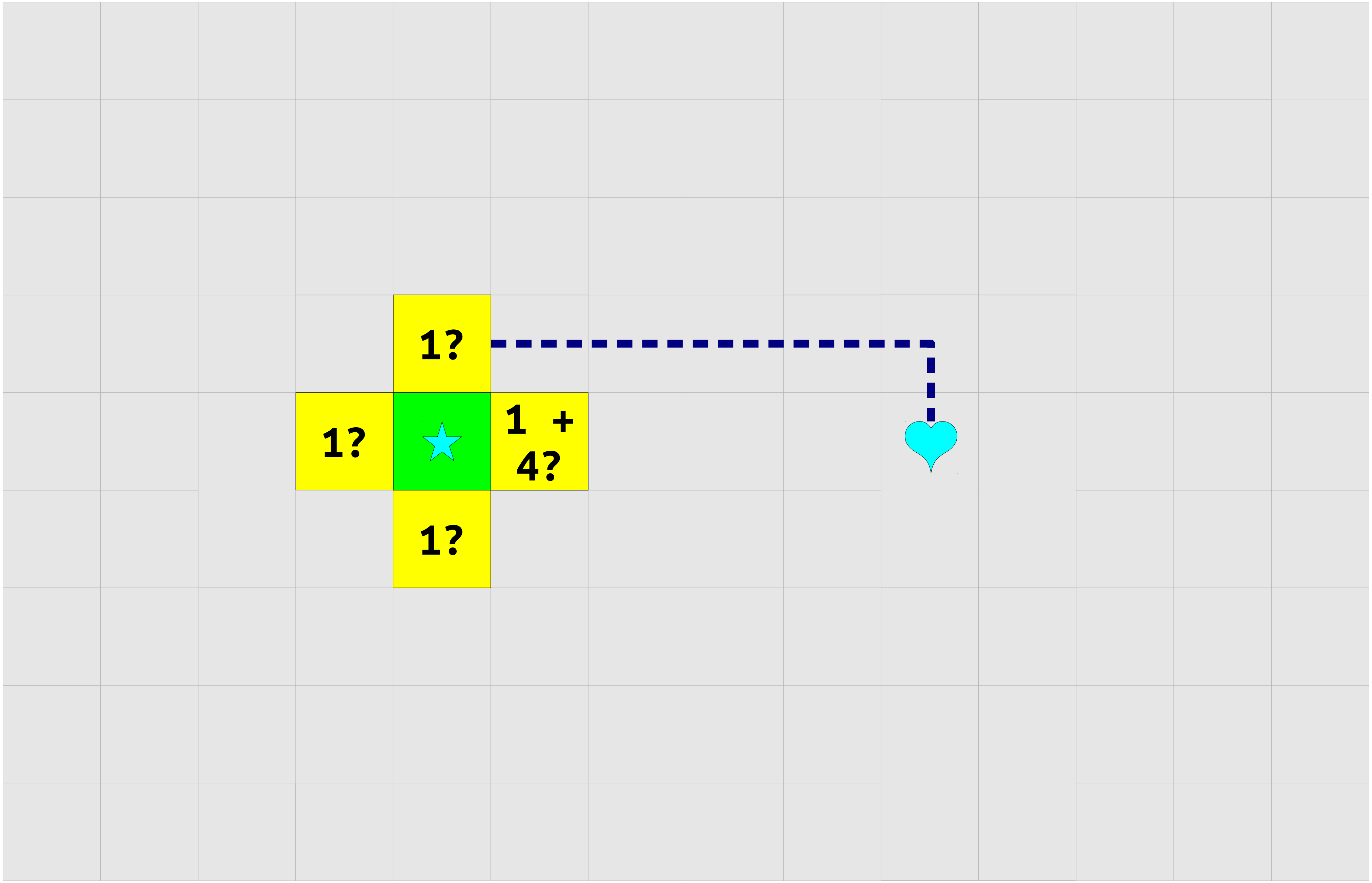


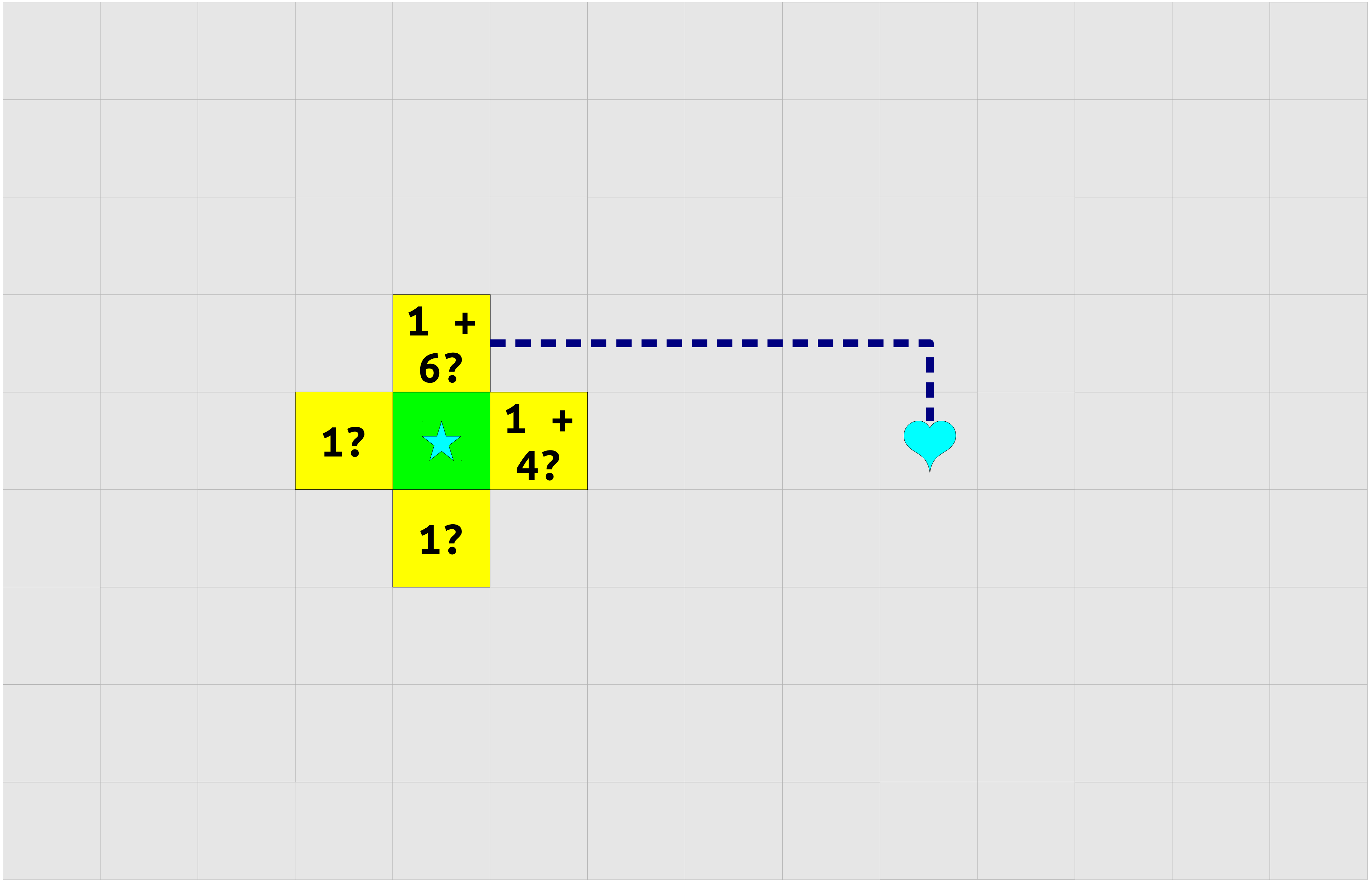


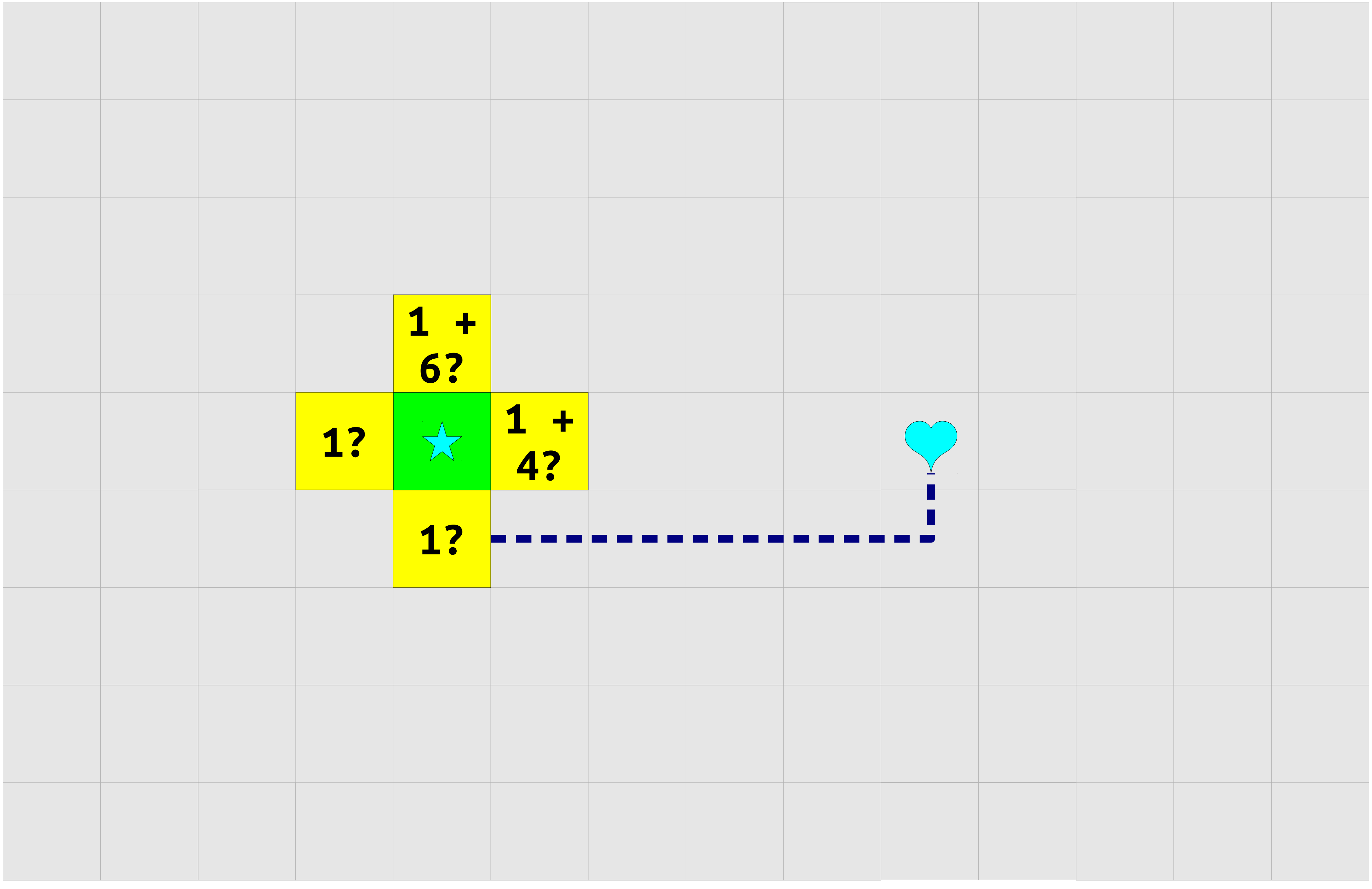

1?
1?
1?
1?

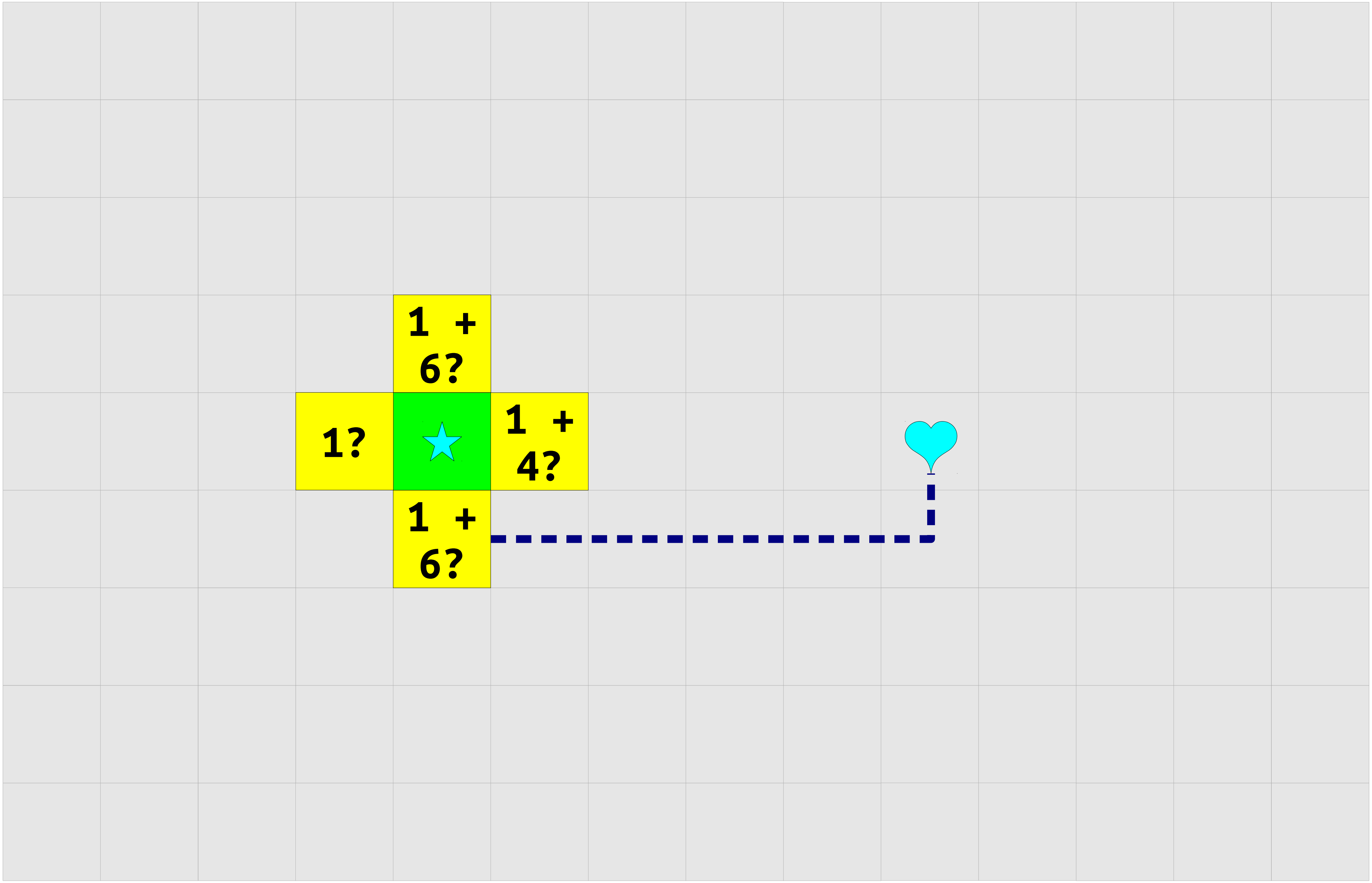

1?
1?
1?
1?


1?
1?
1 + 4?
1?

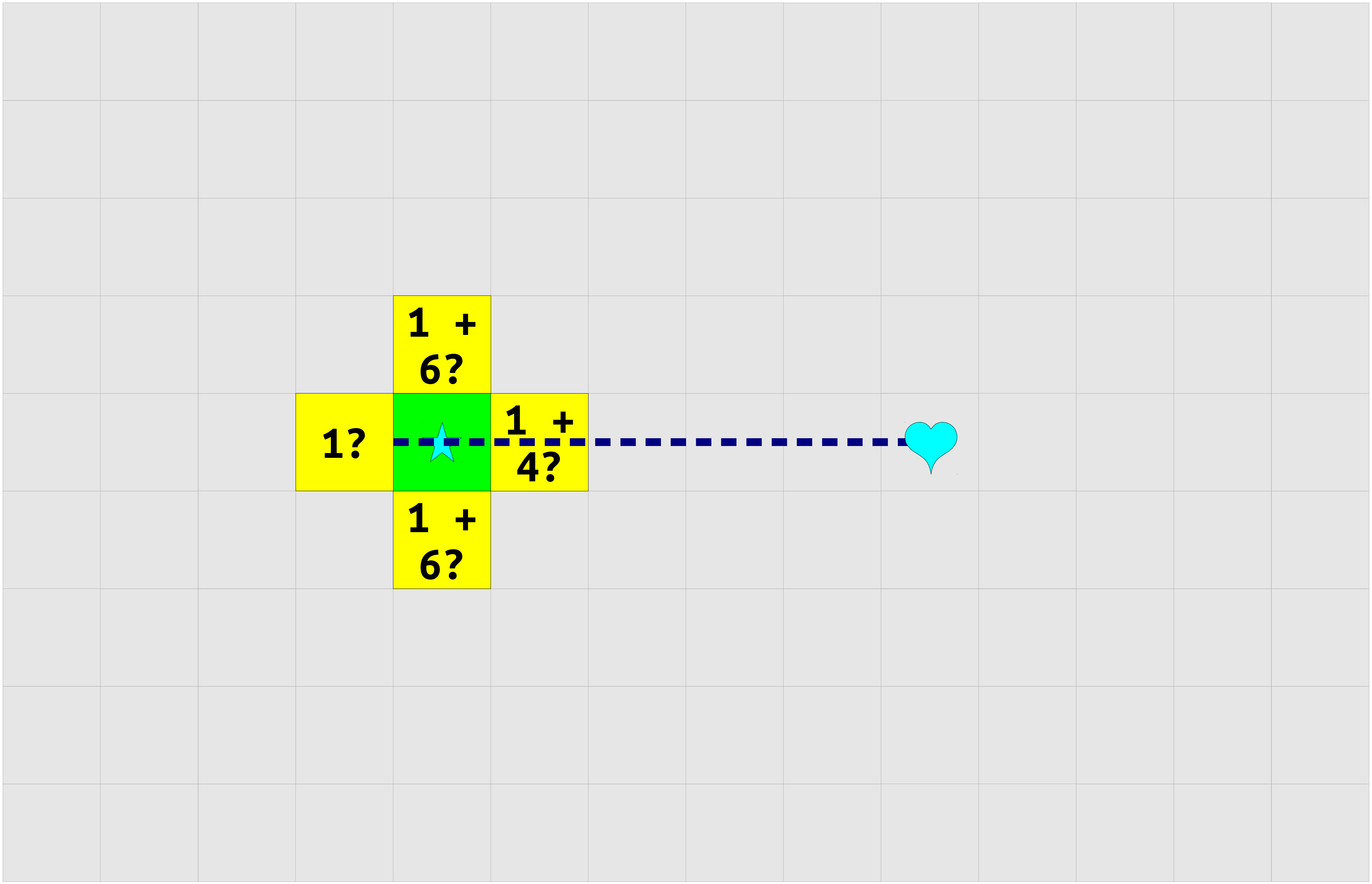

1?
1?
1 +
4?
1?

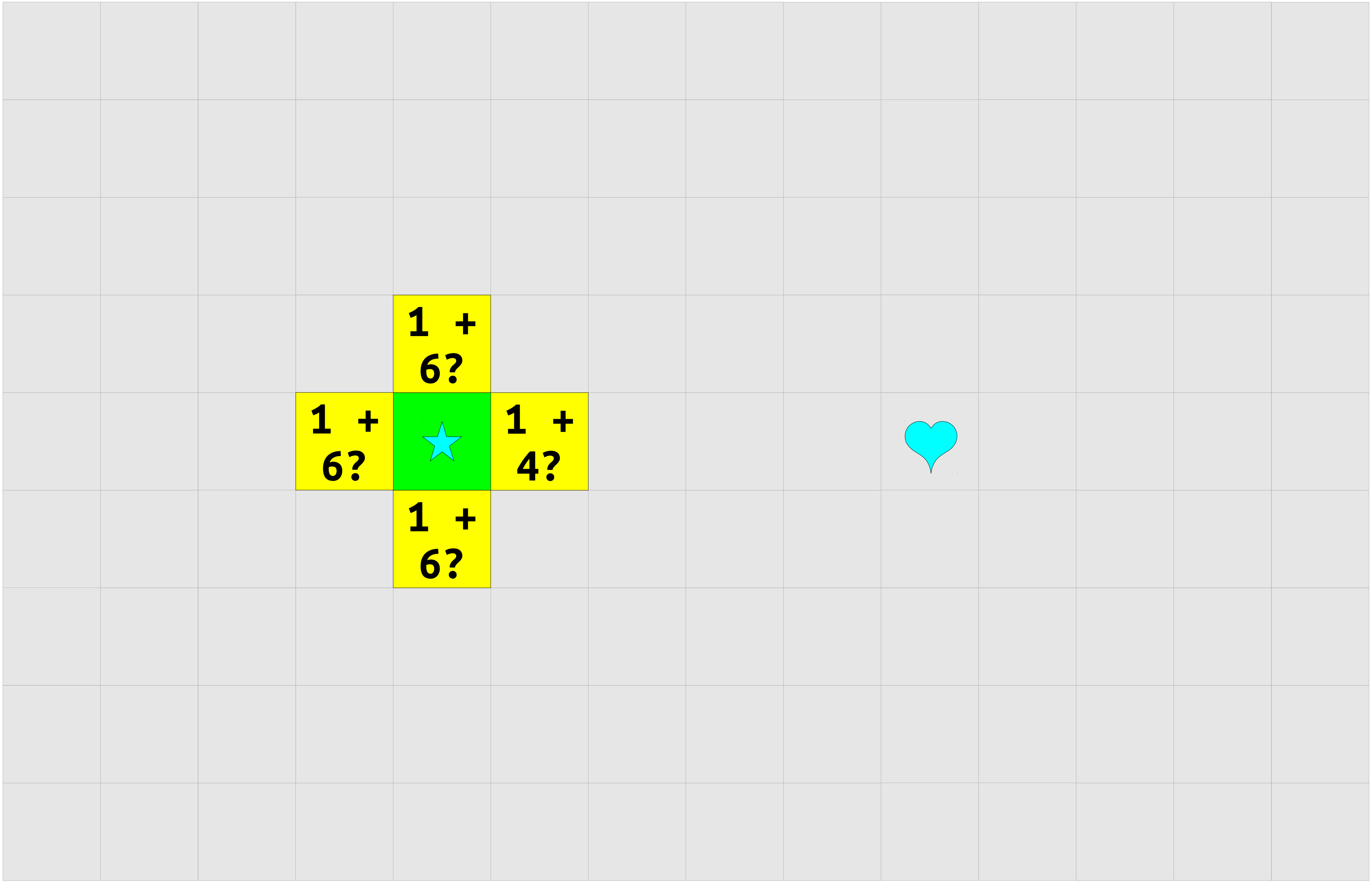

1 + 6?
1?
1 + 4?
1?

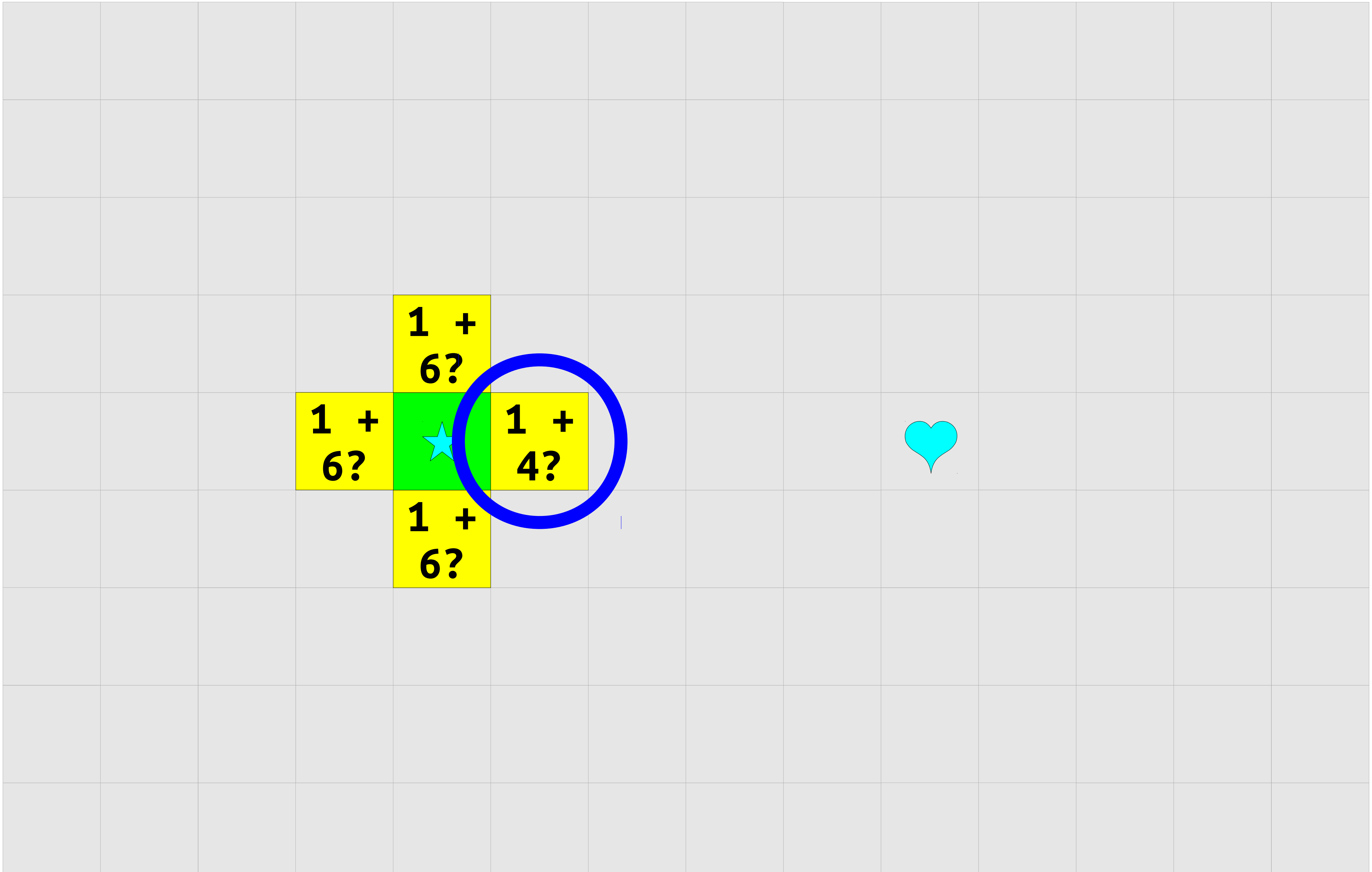

1 + 6?
1?
1 + 4?
1?

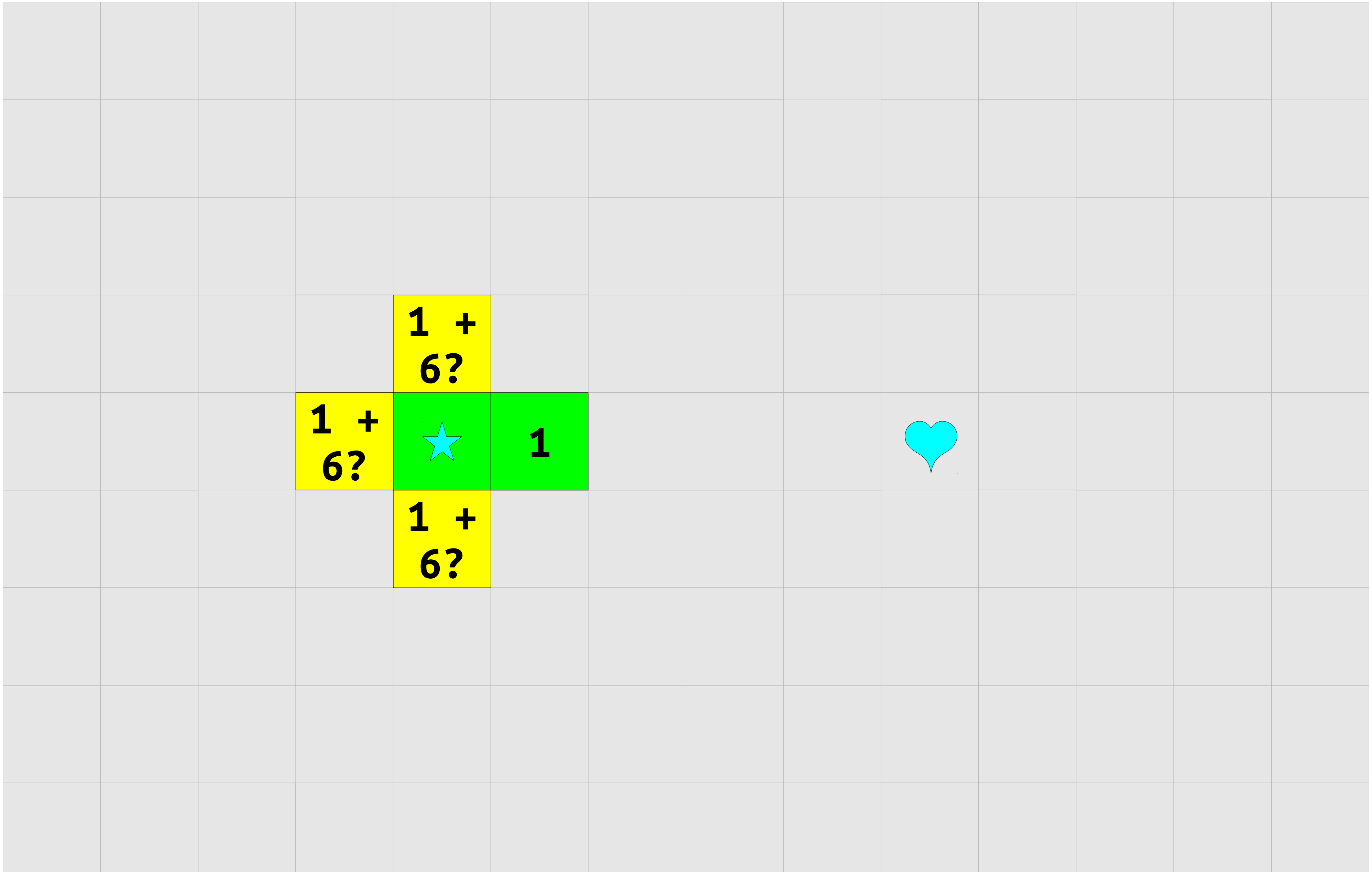


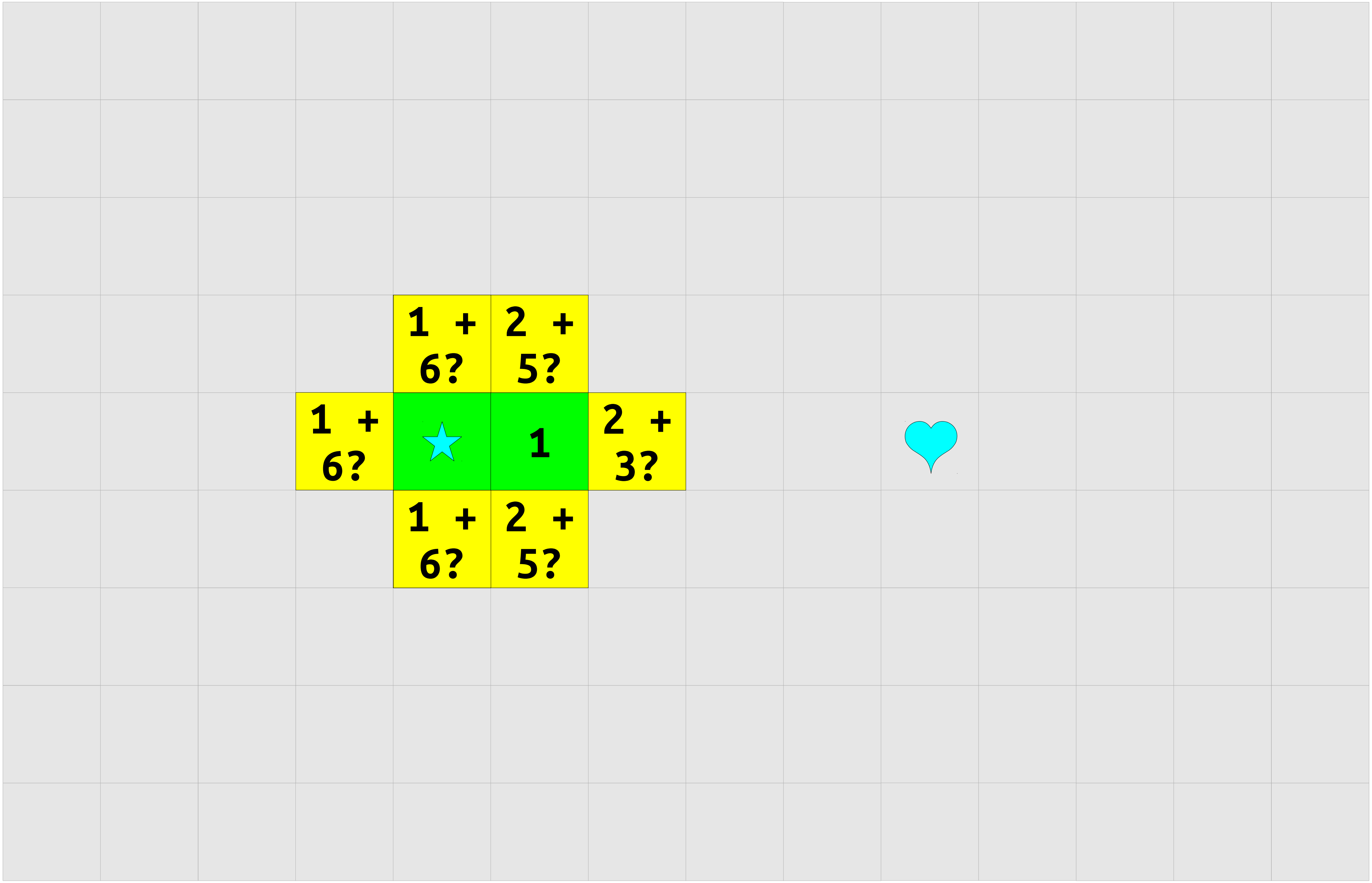
1 + 6?
1?
1 + 4?
1 + 6?

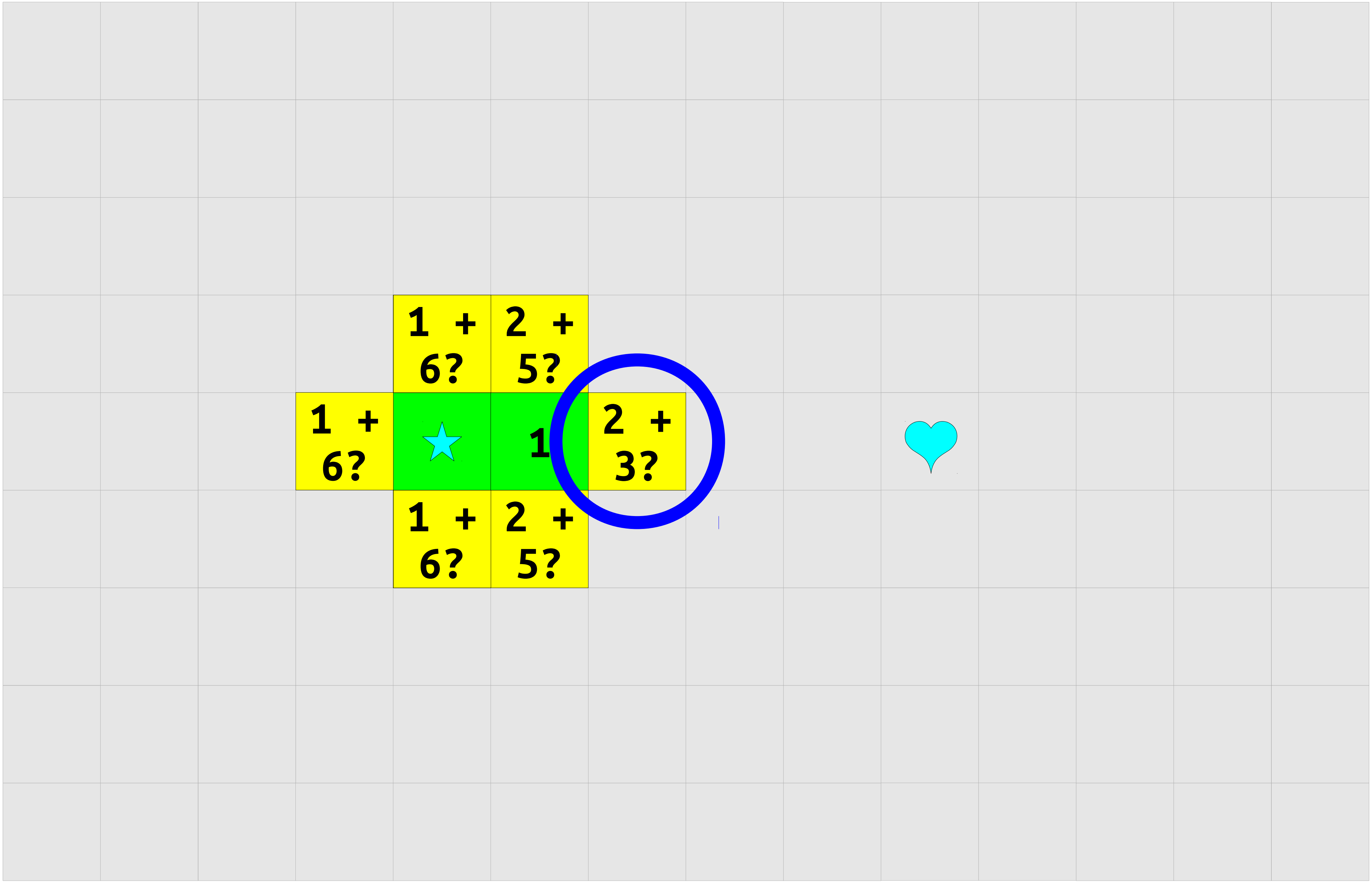

1 + 6?
1?
1 + 4?
1 + 6?

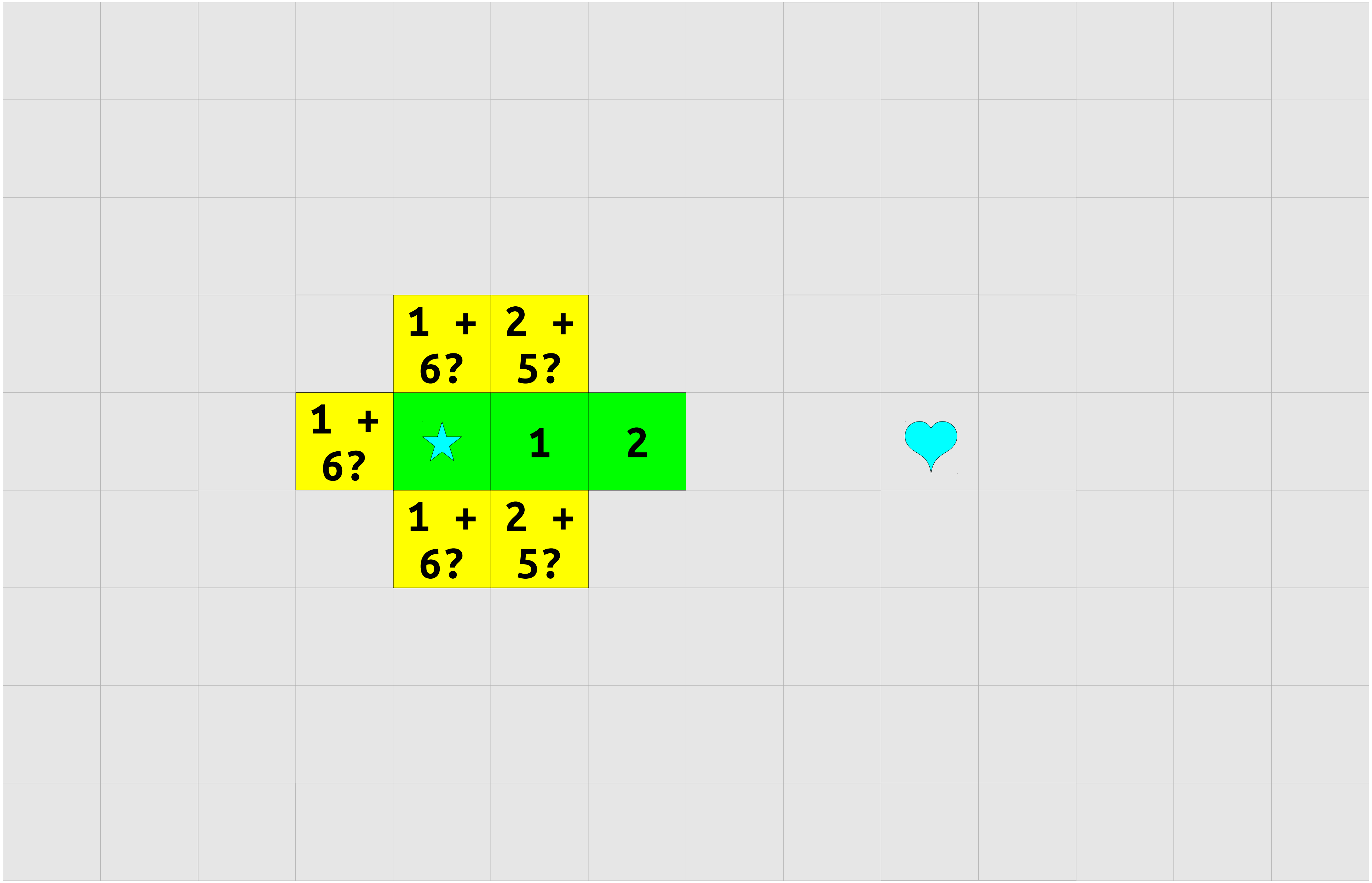

1 + 6?
1 + 6?
1 + 4?
1 + 6?

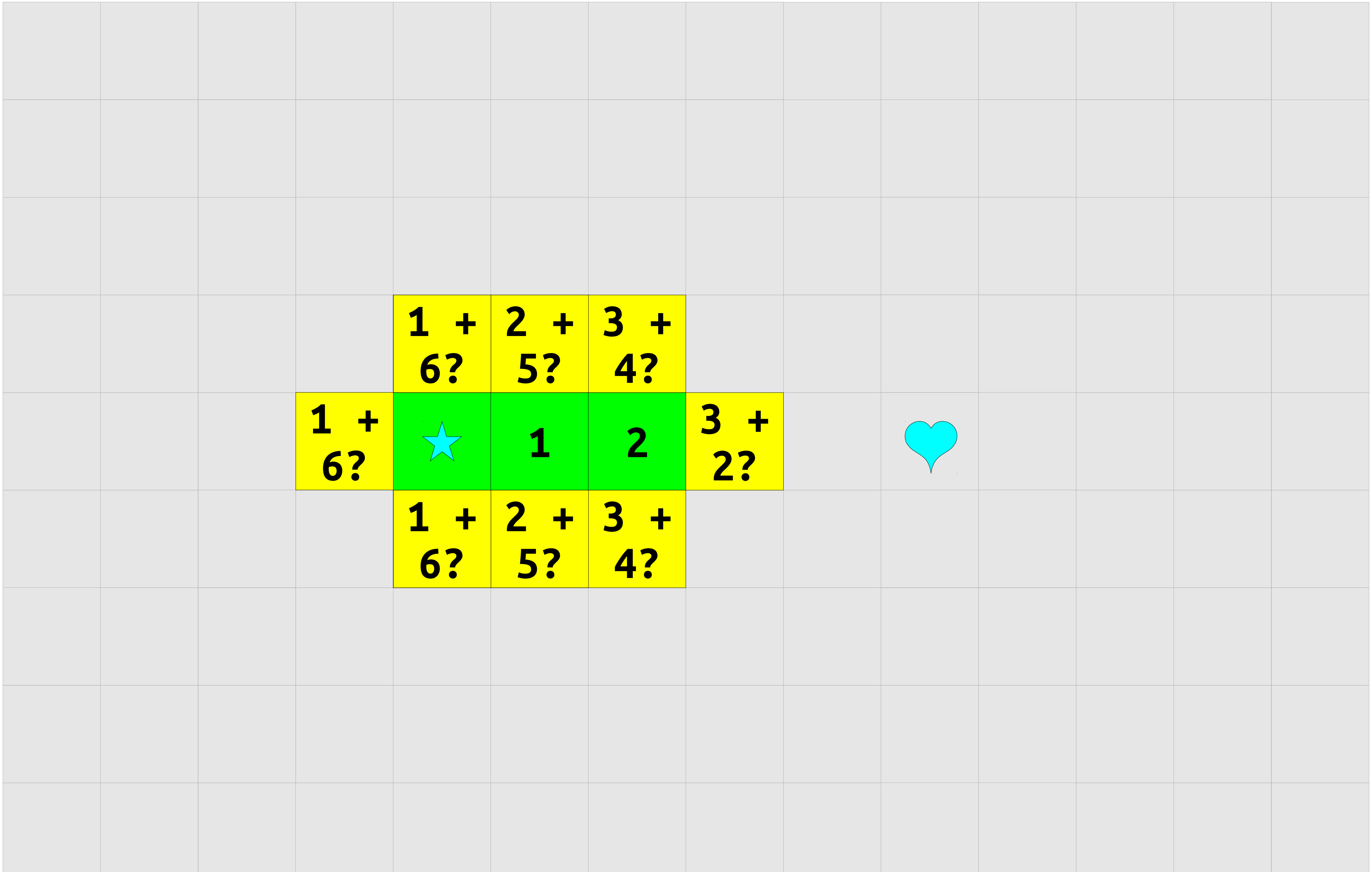

1 + 6?
1 + 6?
1 + 4?
1 + 6?

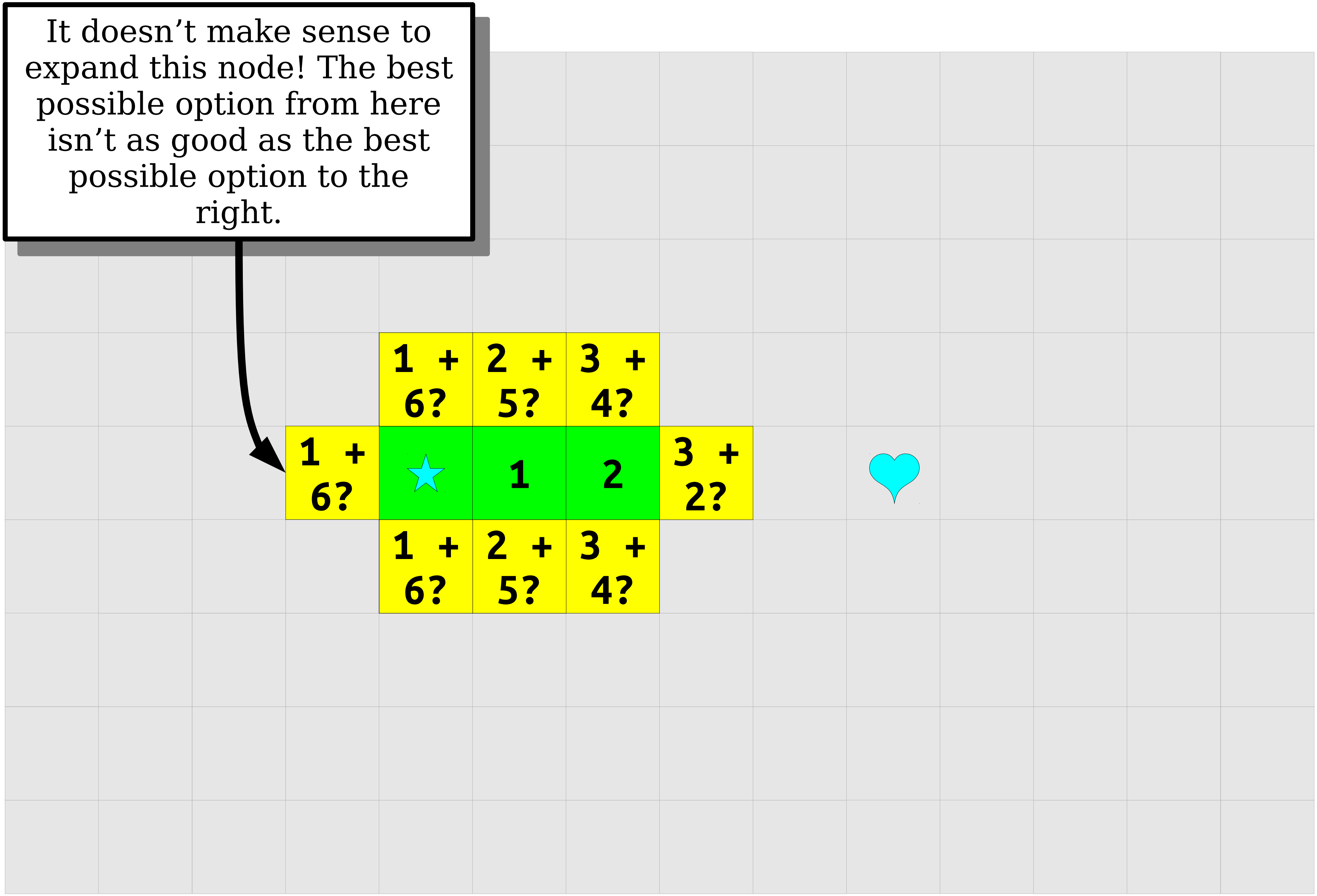

1 + 6?
1 + 6?
1
1 + 6?

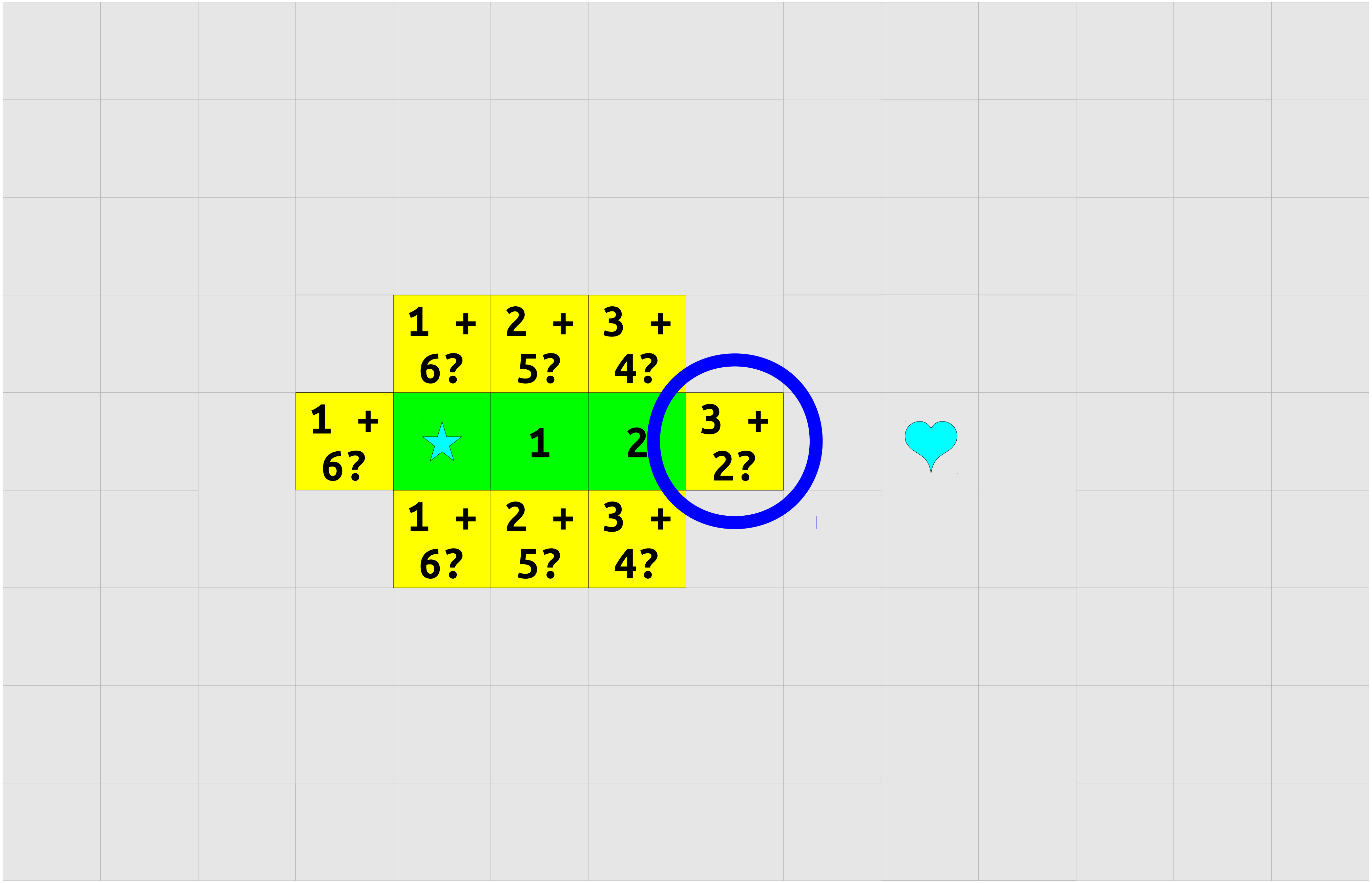

1 + 6?
2 + 5?
1 + 6?
1
2 + 3?
1 + 6?
2 + 5?


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

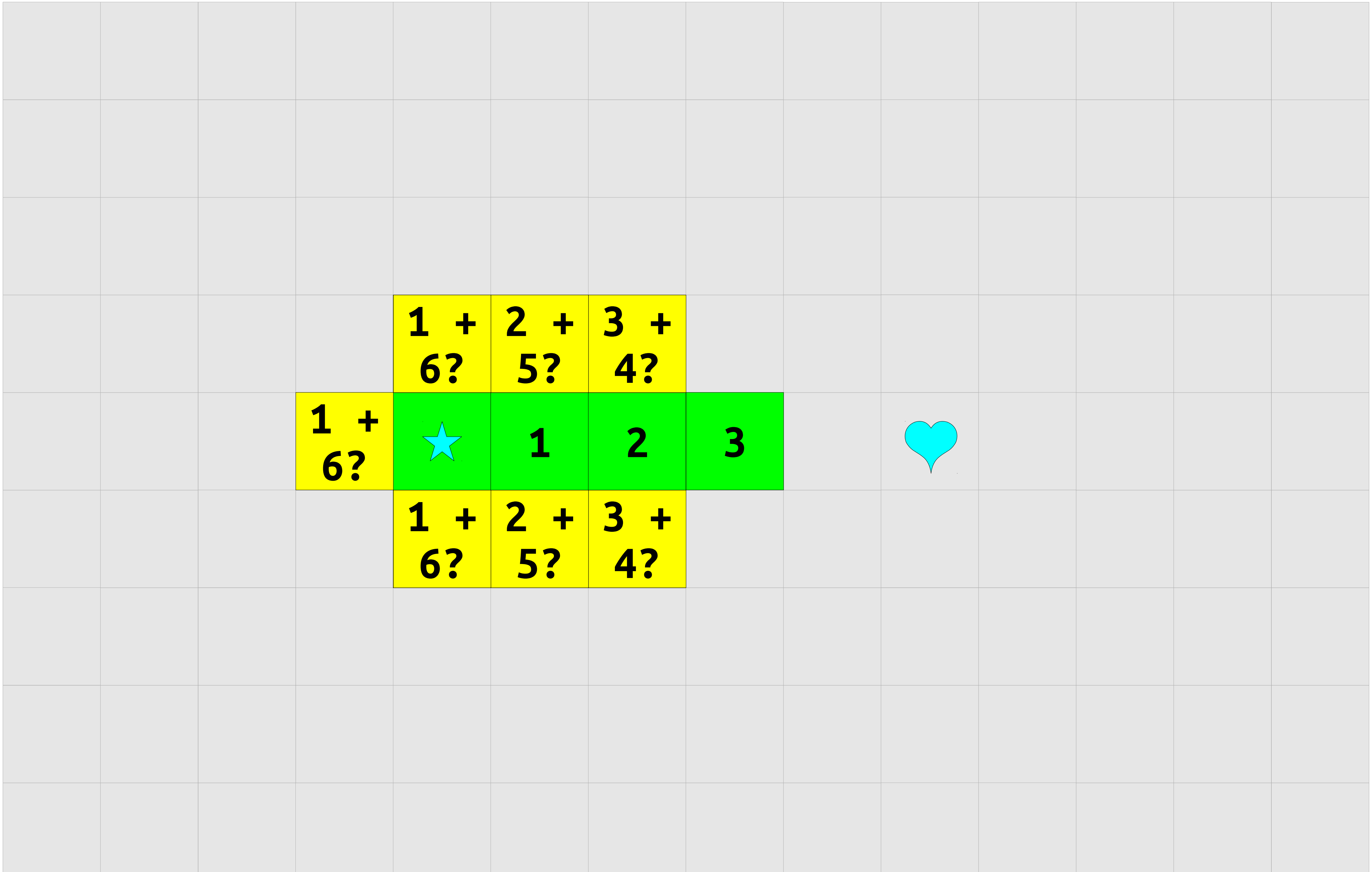

1 + 6?
2 + 5?
1 + 6?
1
2
1 + 6?
2 + 5?

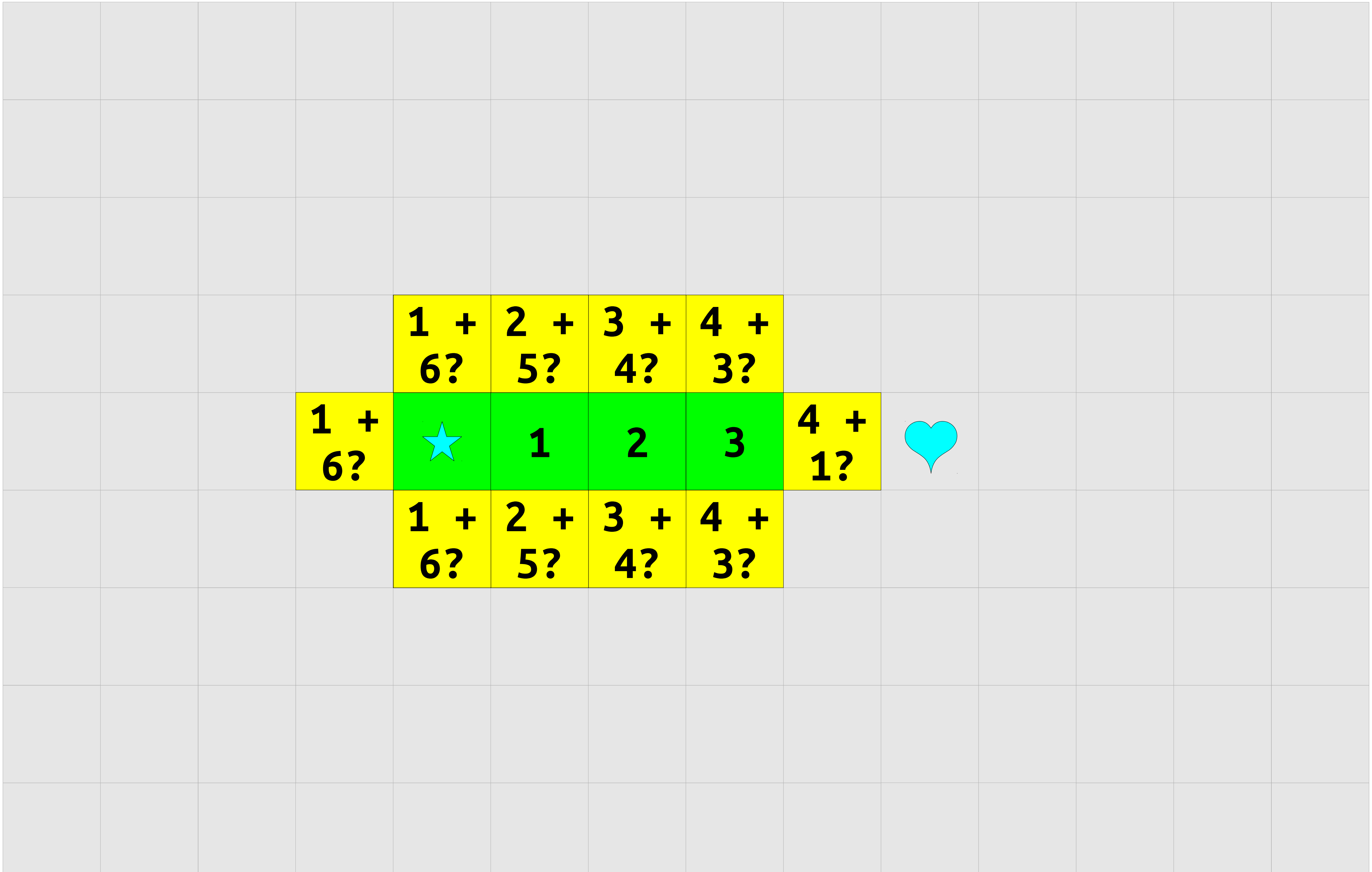

1 + 6?
2 + 5?
3 + 4?
1 + 6?
1
2
3 + 2?
1 + 6?
2 + 5?
3 + 4?

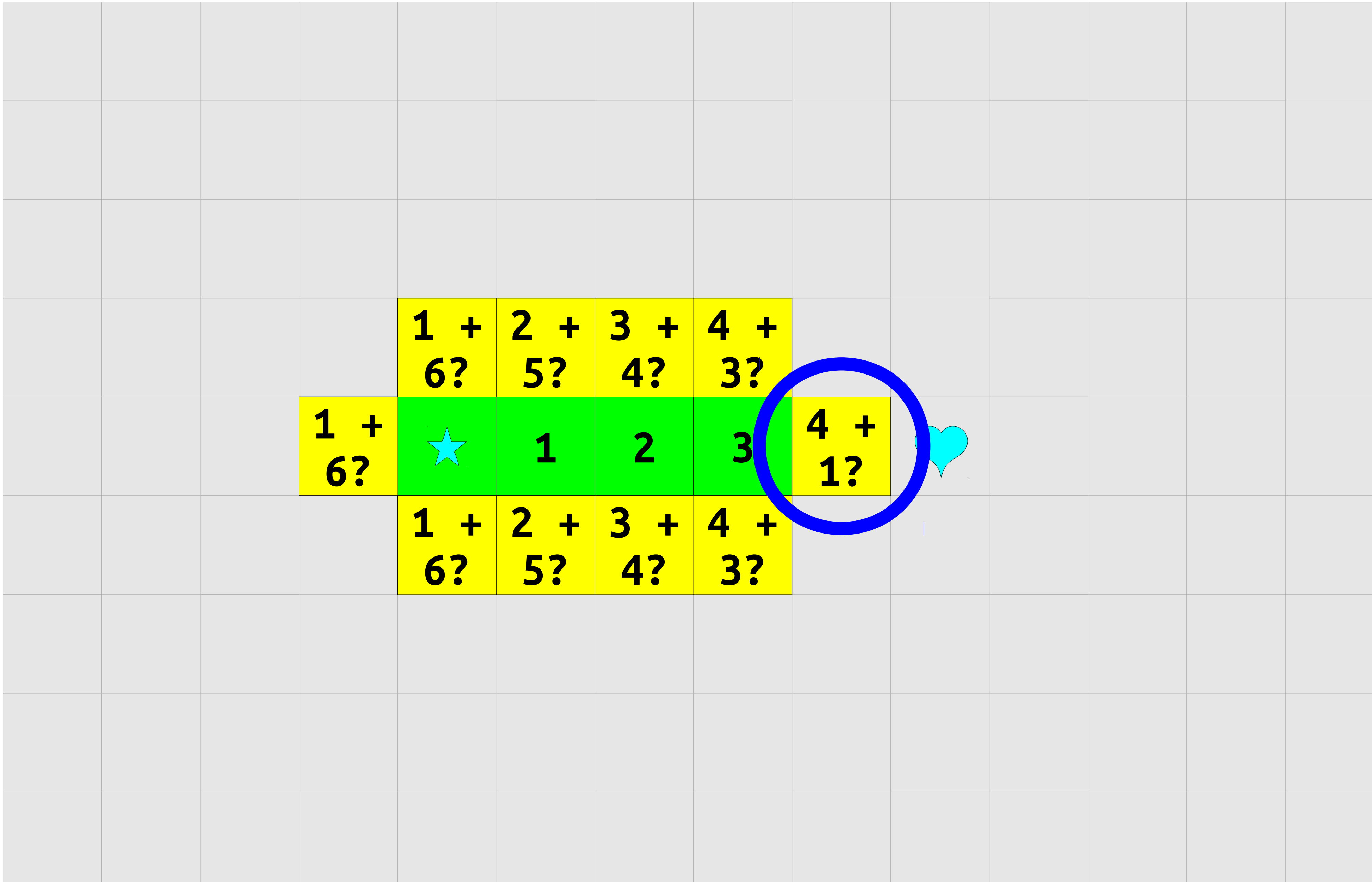

It doesn't make sense to expand this node! The best possible option from here isn't as good as the best possible option to the right.
1 + 6?
2 + 5?
3 + 4?
1 + 6?
1
2
3 + 2?
1 + 6?
2 + 5?
3 + 4?

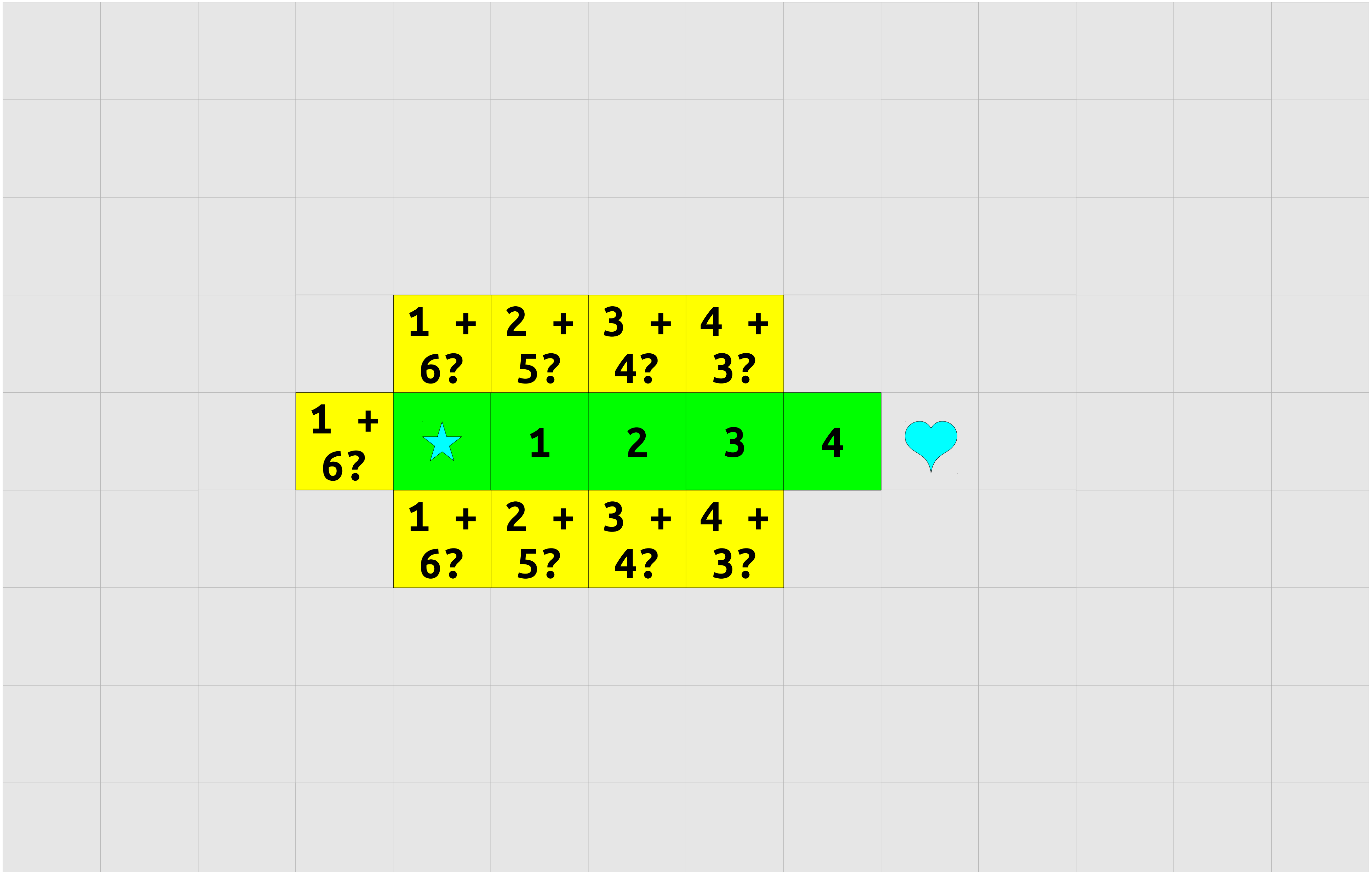


1 + 6?
2 + 5?
3 + 4?
1 + 6?
1
2
3 + 2?
1 + 6?
2 + 5?
3 + 4?


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

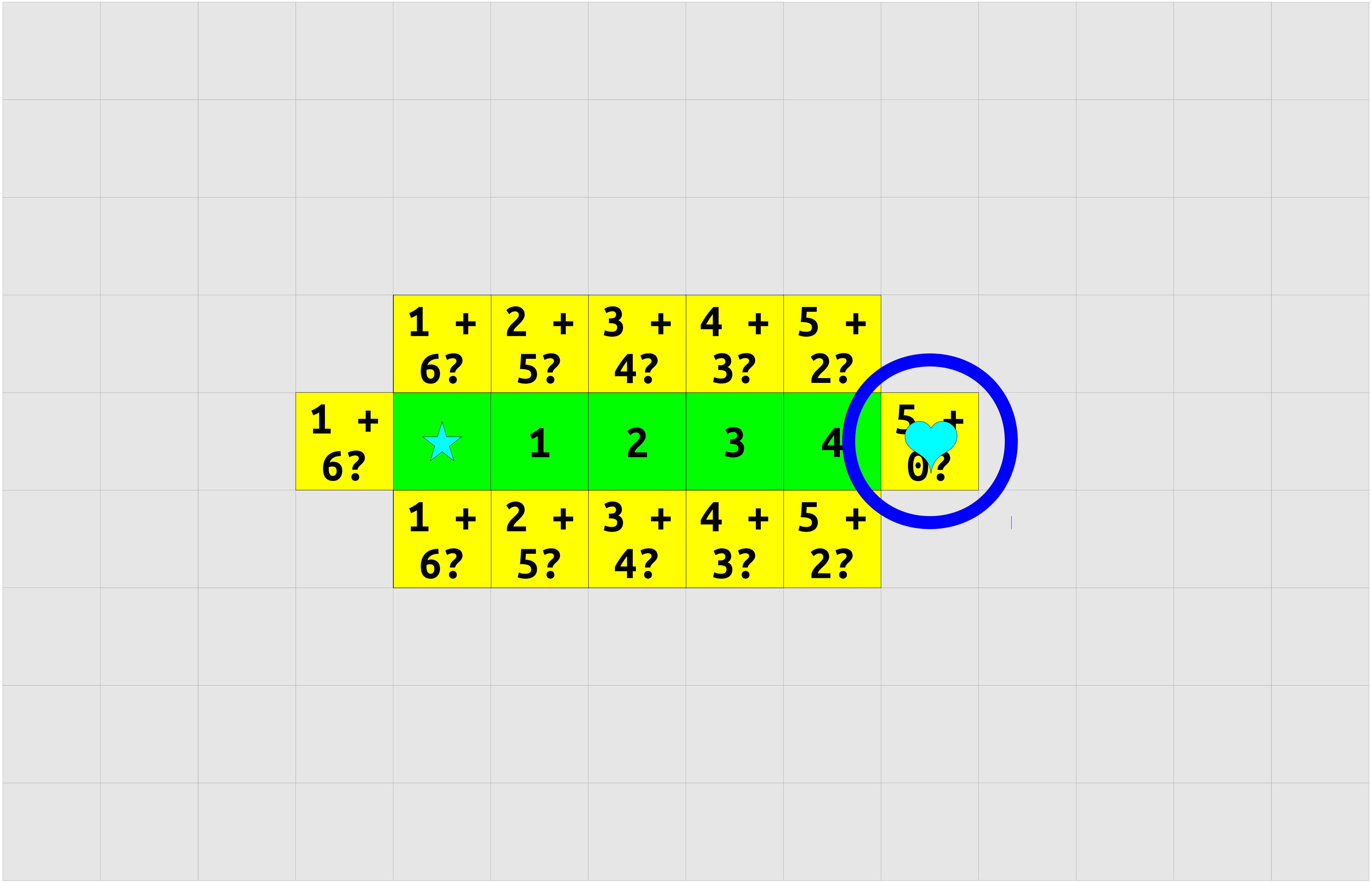

1 + 6?
2 + 5?
3 + 4?
4 + 3?
1 + 6?
1
2
3
4 + 1?
1 + 6?
2 + 5?
3 + 4?
4 + 3?


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

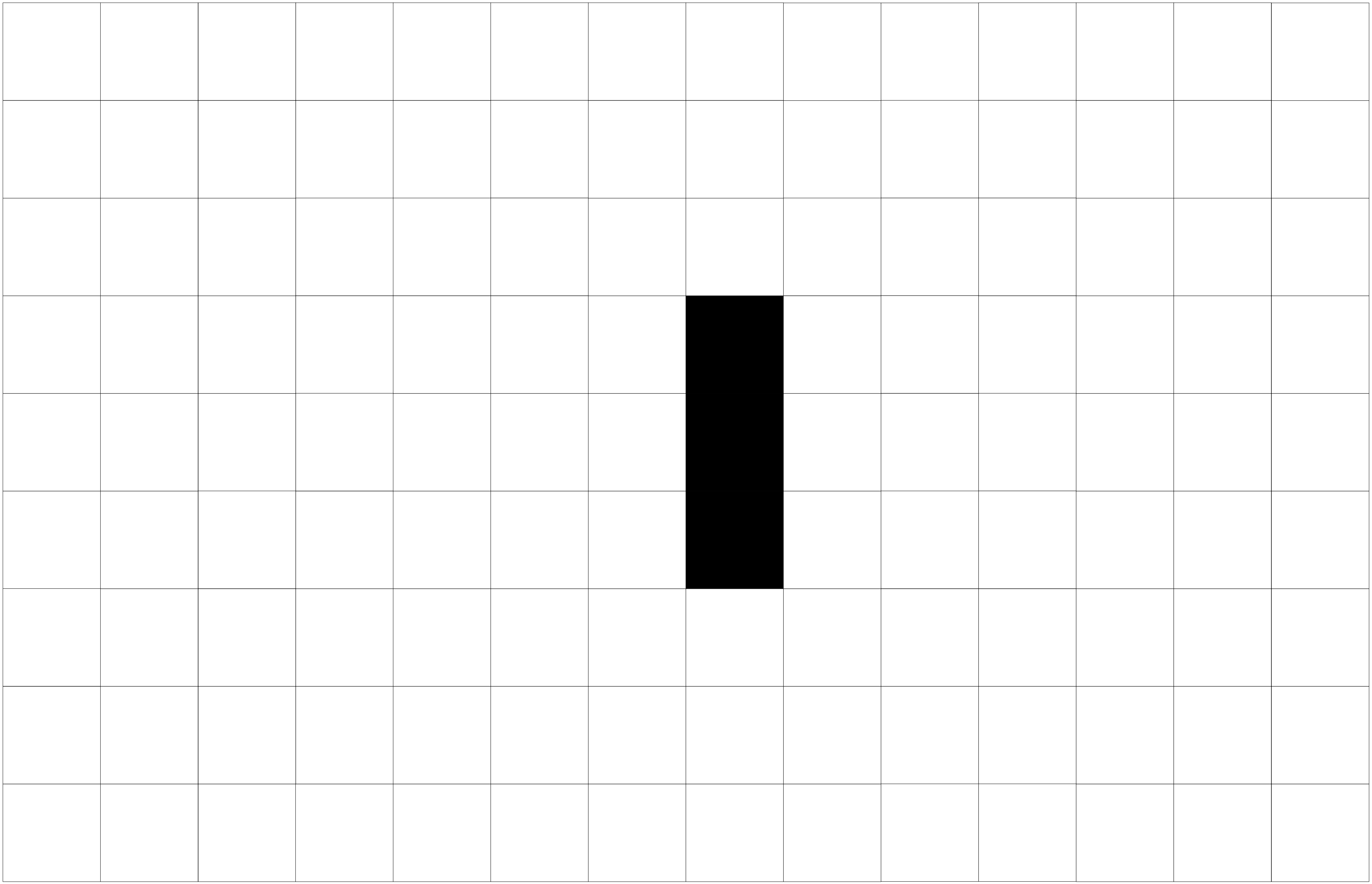


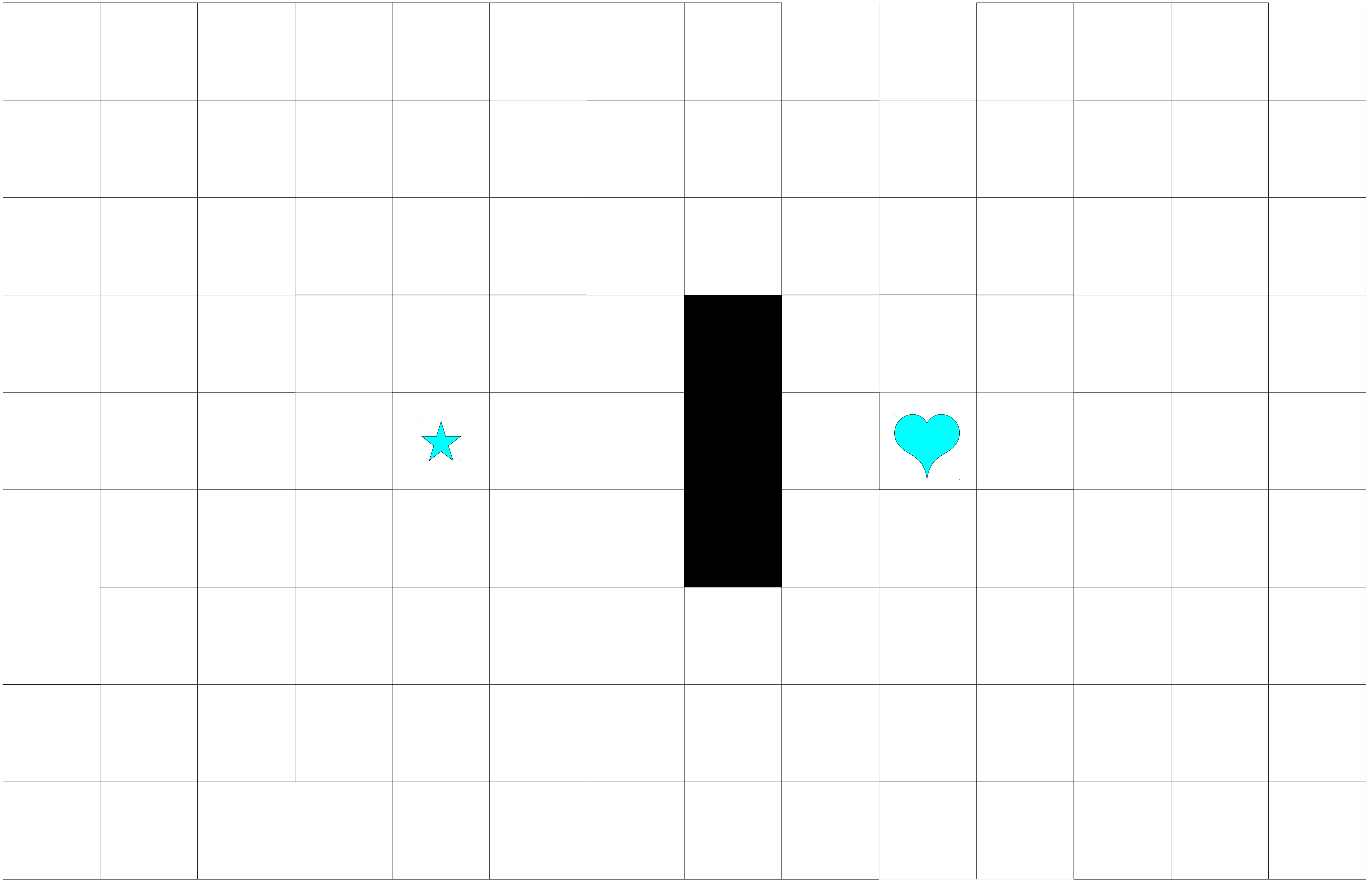
1 + 6?
2 + 5?
3 + 4?
4 + 3?
1 + 6?
1
2
3
4
1 + 6?
2 + 5?
3 + 4?
4 + 3?

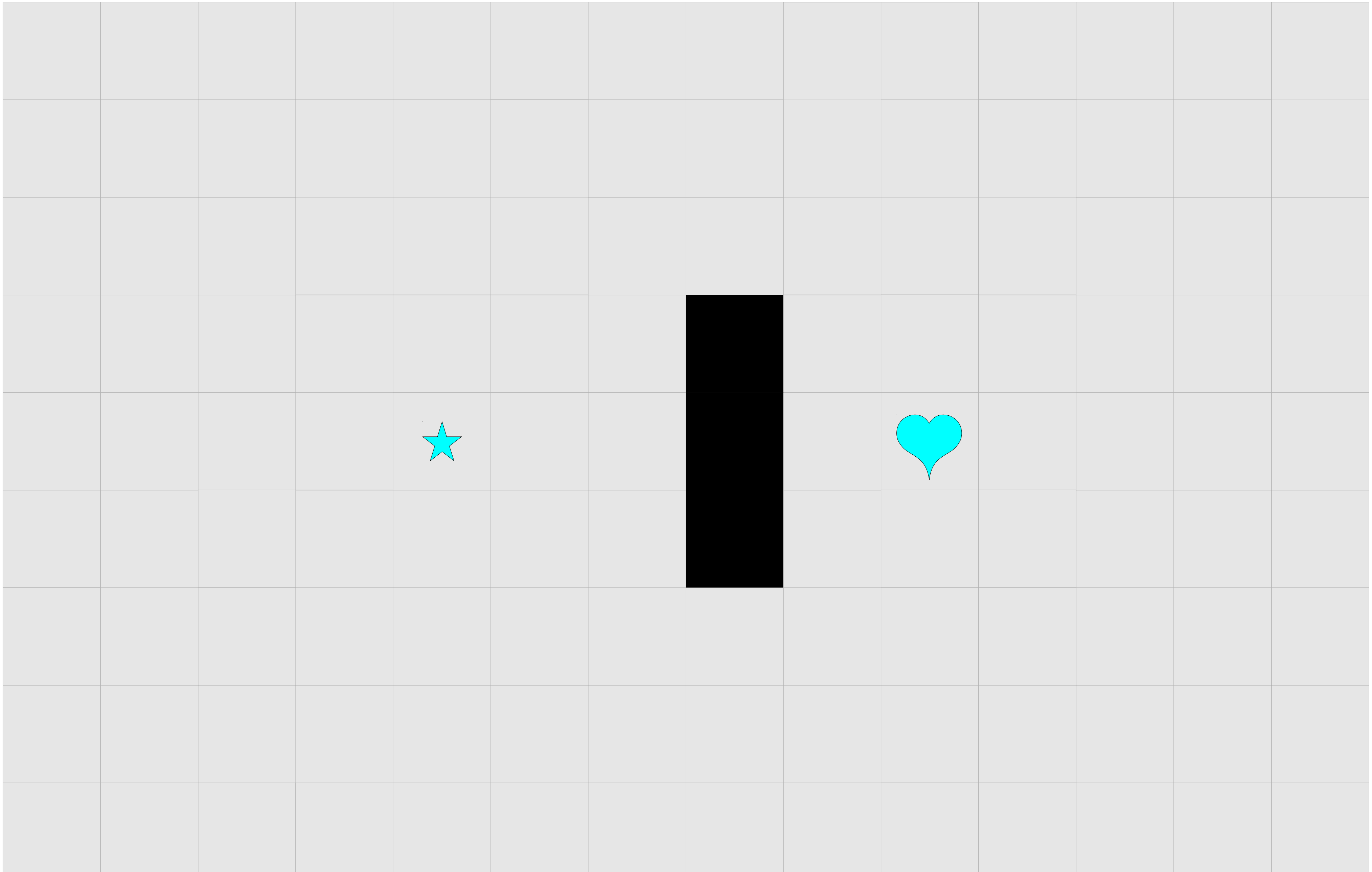
	1 + 6?	2 + 5?	3 + 4?	4 + 3?	5 + 2?	
1 + 6?	★	1	2	3	4	5 + 0? ♥
	1 + 6?	2 + 5?	3 + 4?	4 + 3?	5 + 2?	

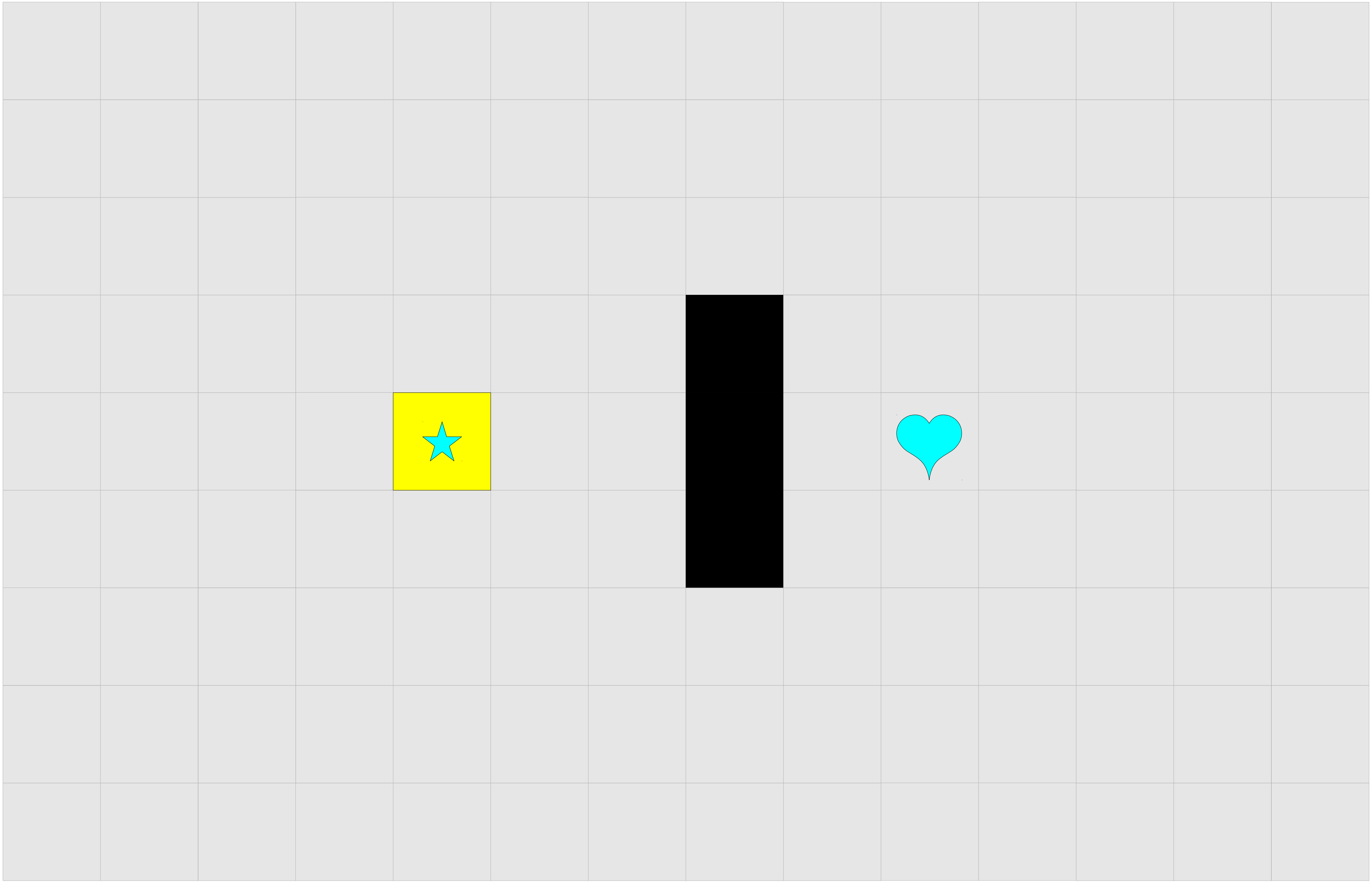

1 + 6?
2 + 5?
3 + 4?
4 + 3?
5 + 2?
1 + 6?
1
2
3
4
5 + 0?
1 + 6?
2 + 5?
3 + 4?
4 + 3?
5 + 2?

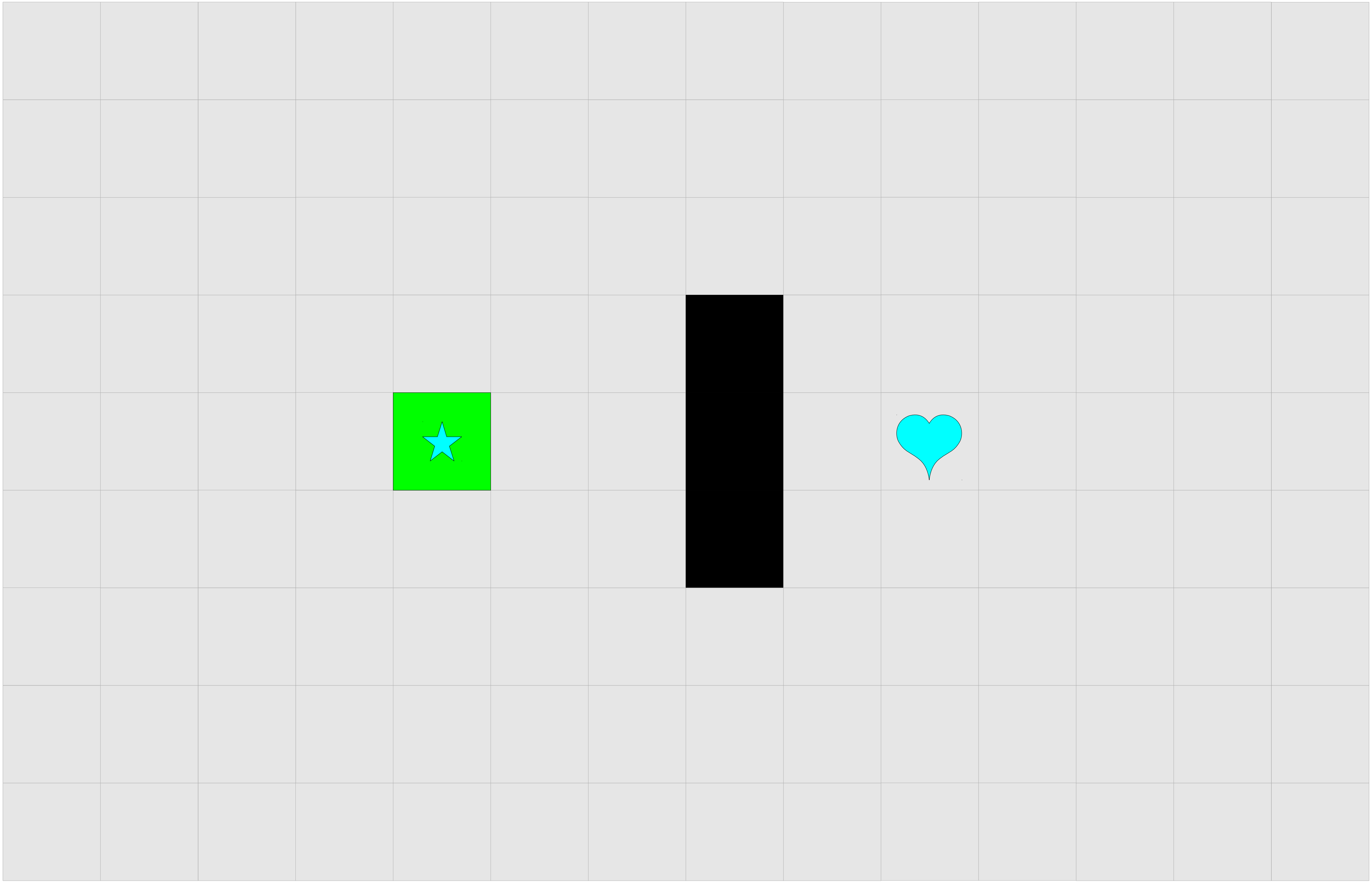
	1 + 6?	2 + 5?	3 + 4?	4 + 3?	5 + 2?	
1 + 6?		1	2	3	4	
	1 + 6?	2 + 5?	3 + 4?	4 + 3?	5 + 2?	

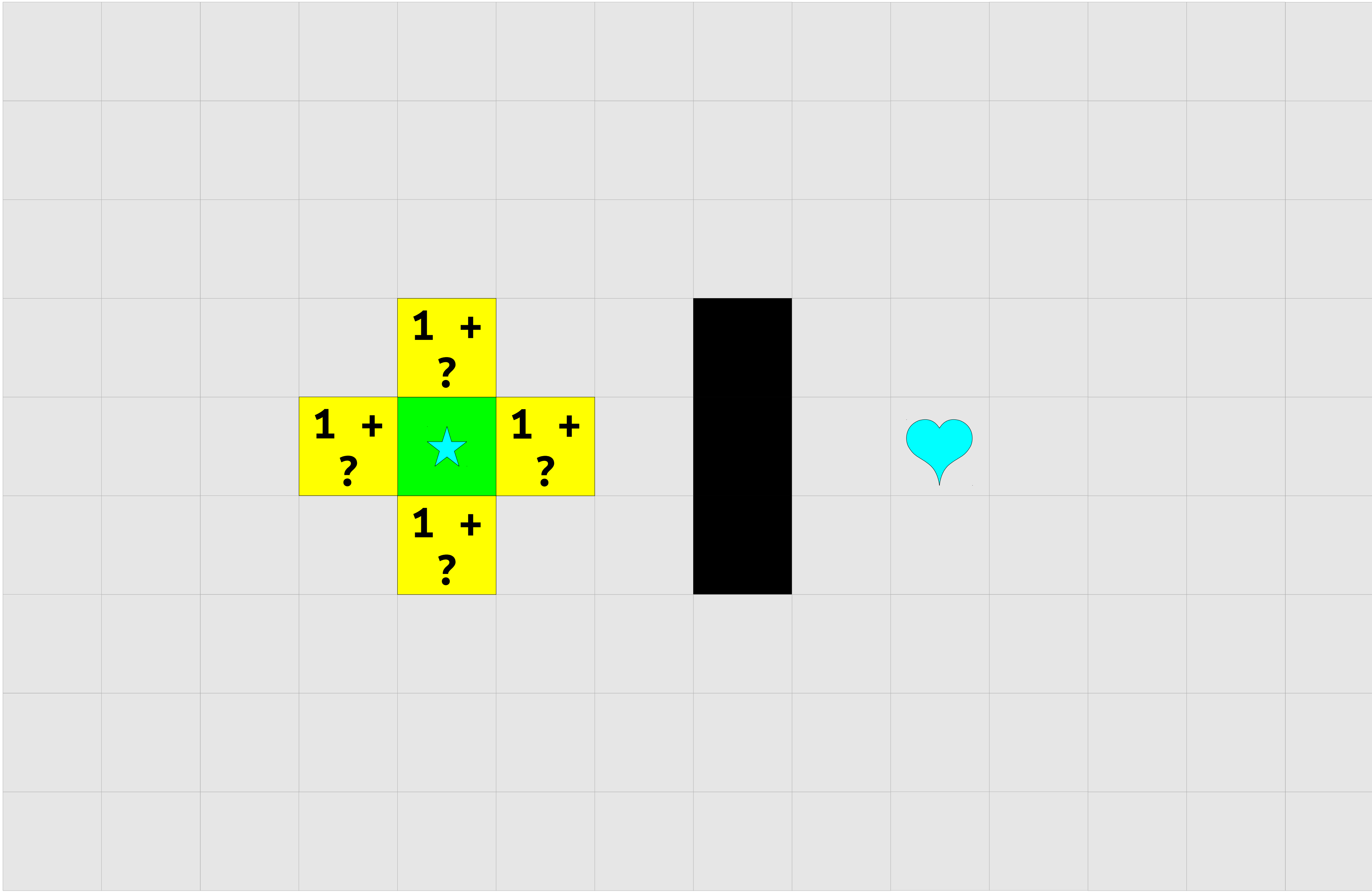


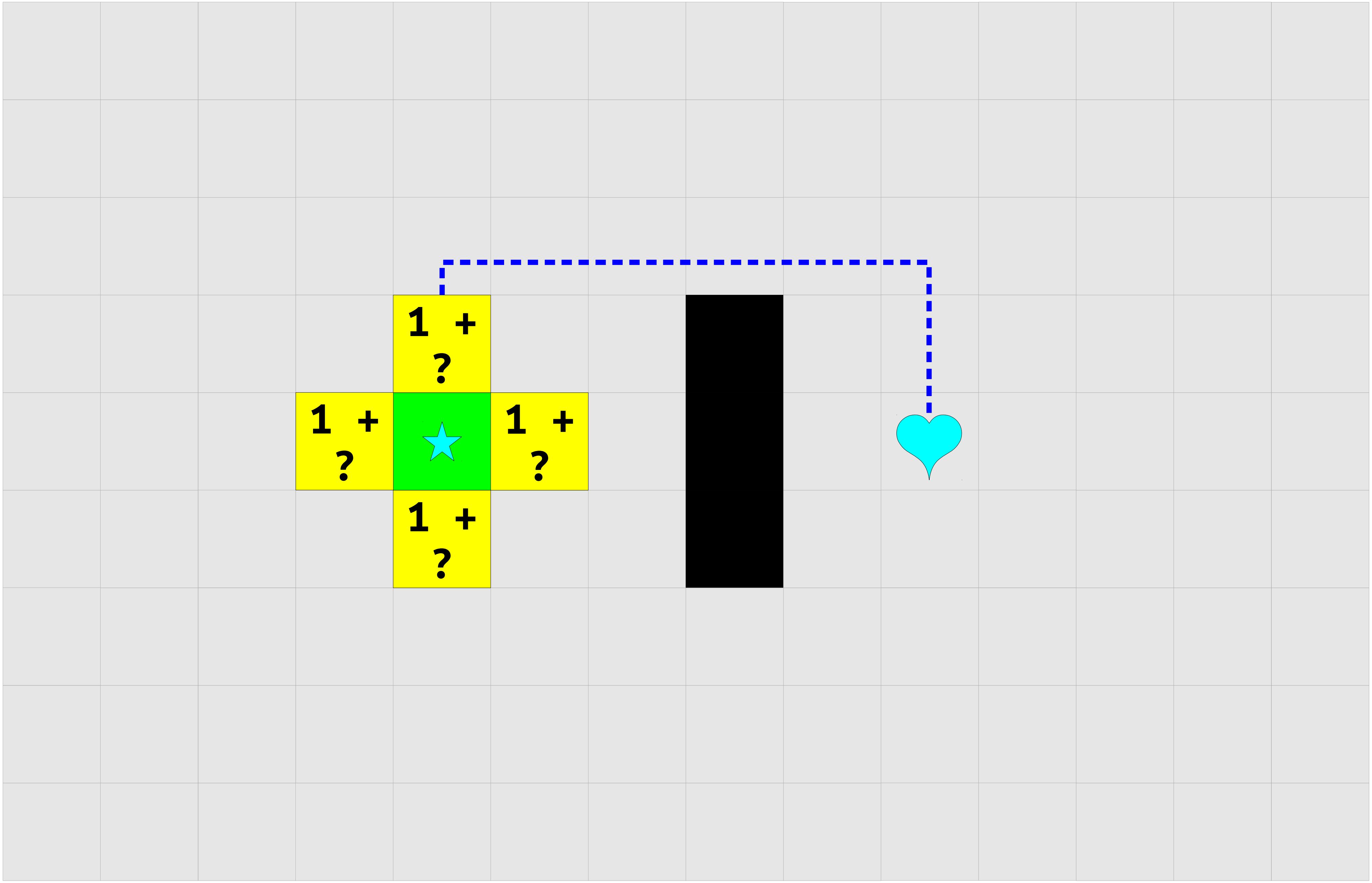


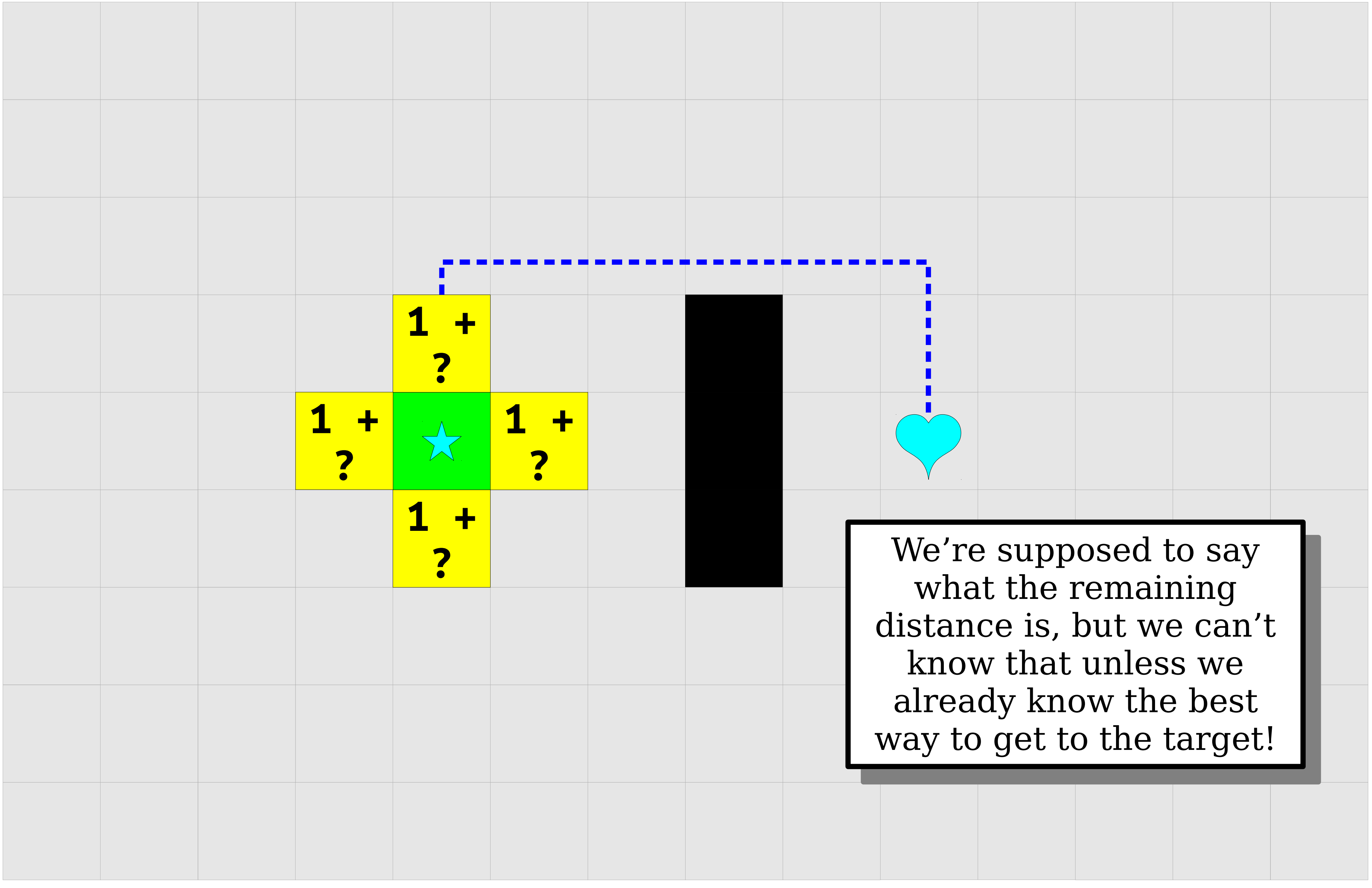







1 + ?
1 + ?
1 + ?
1 + ?

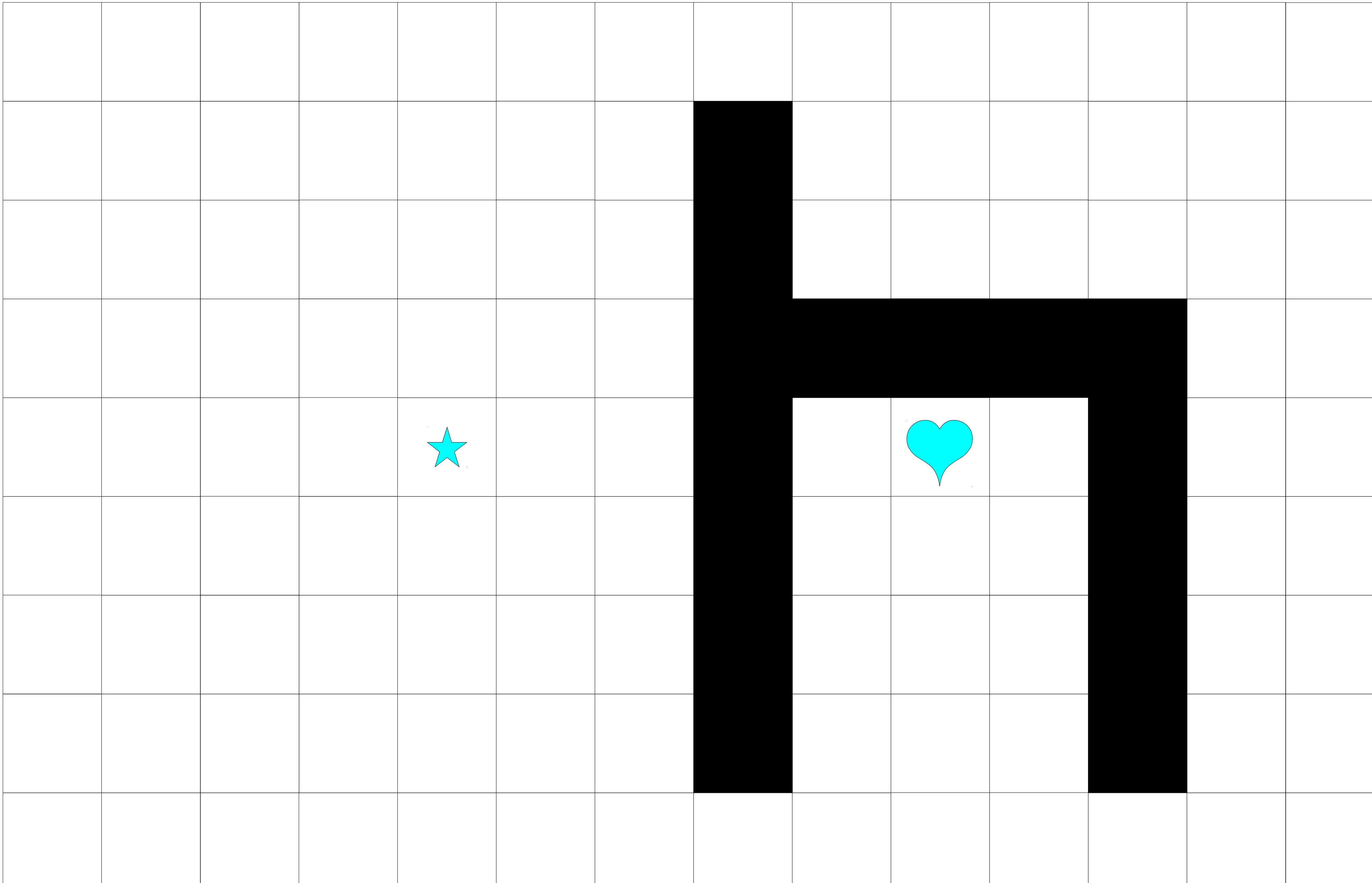

1 + ?
1 + ?
1 + ?
1 + ?

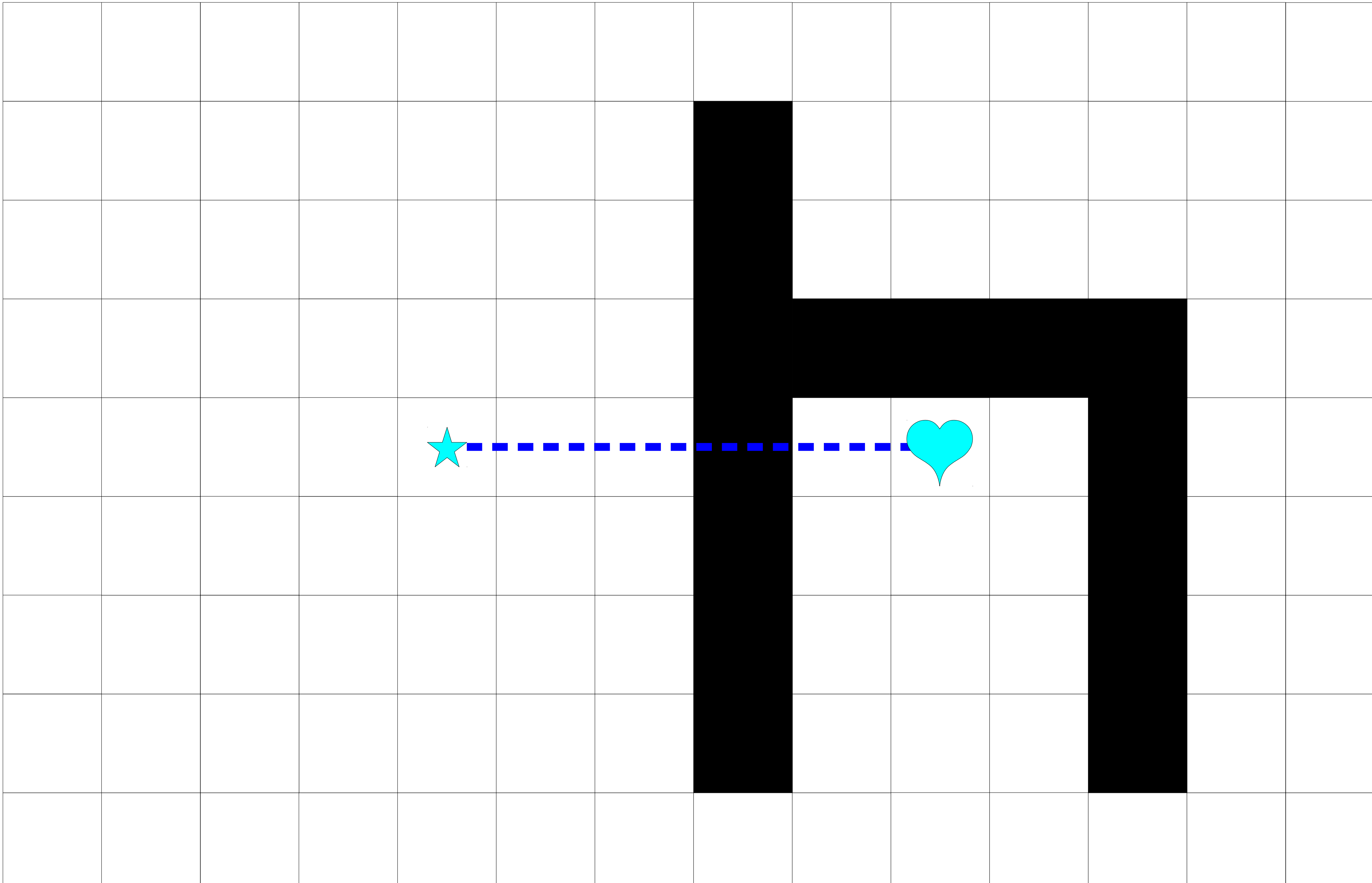


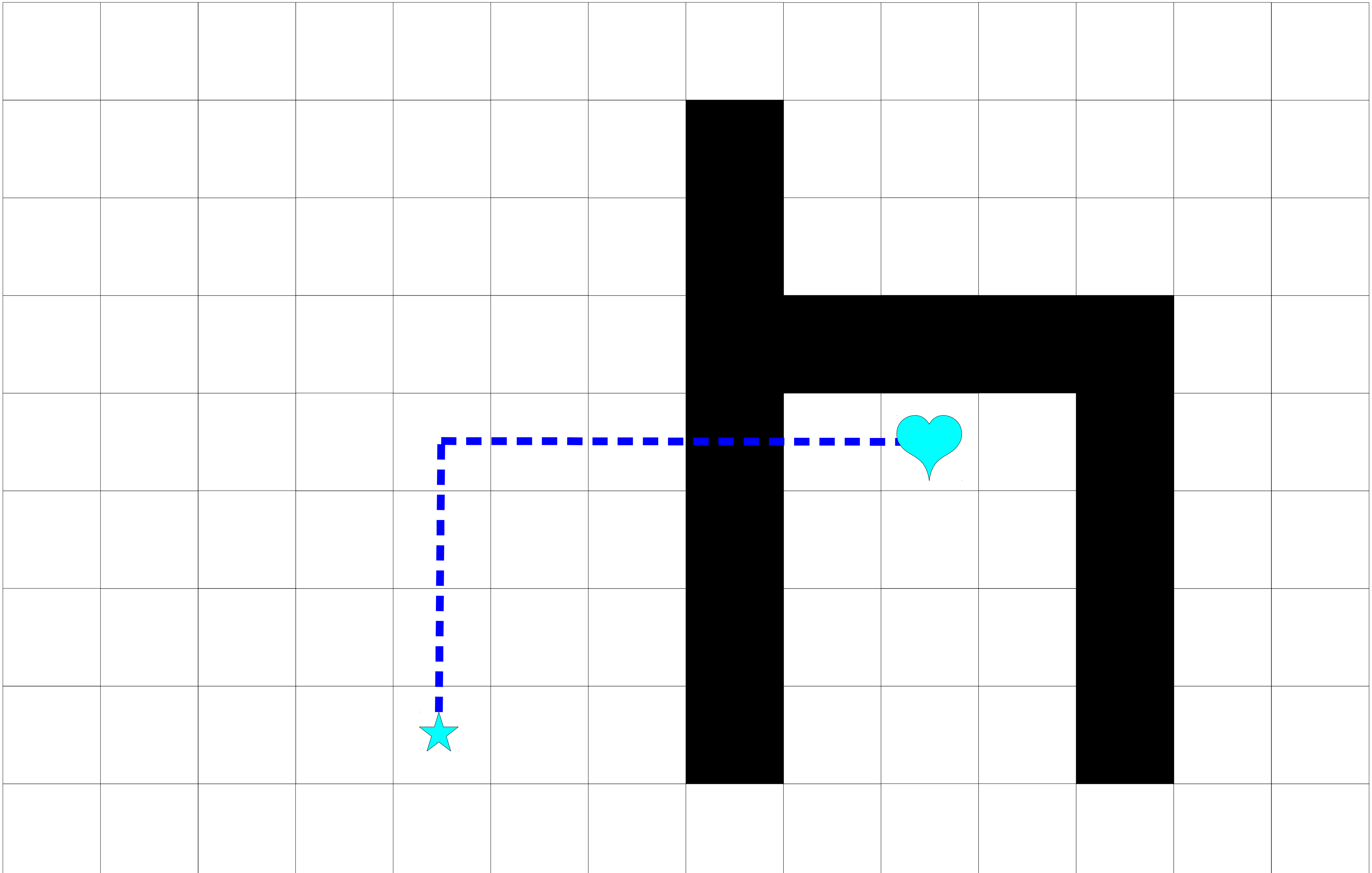
1 + ?
1 + ?
1 + ?
1 + ?
We're supposed to say what the remaining distance is, but we can't know that unless we already know the best way to get to the target!

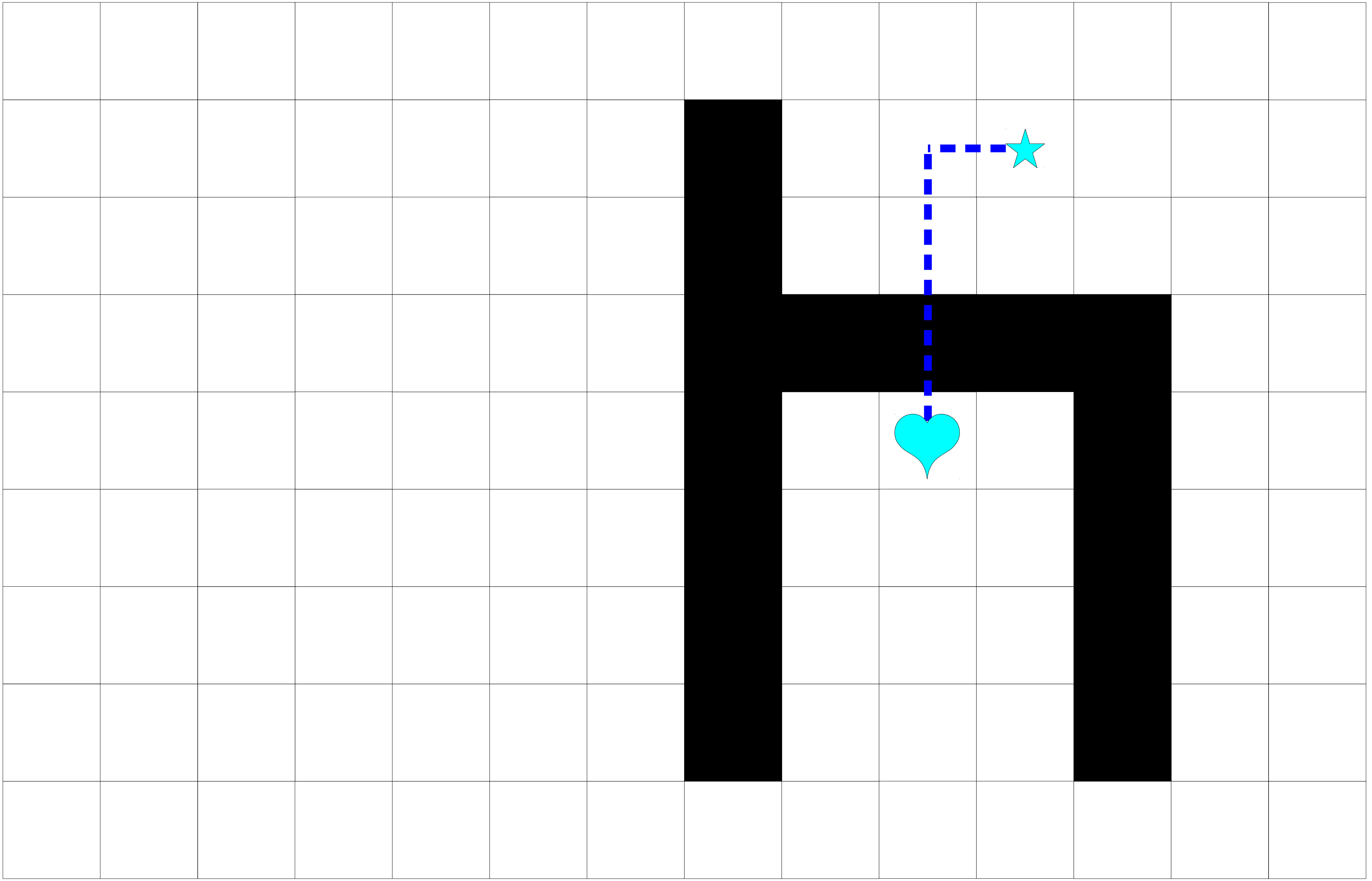
The Golden Mean

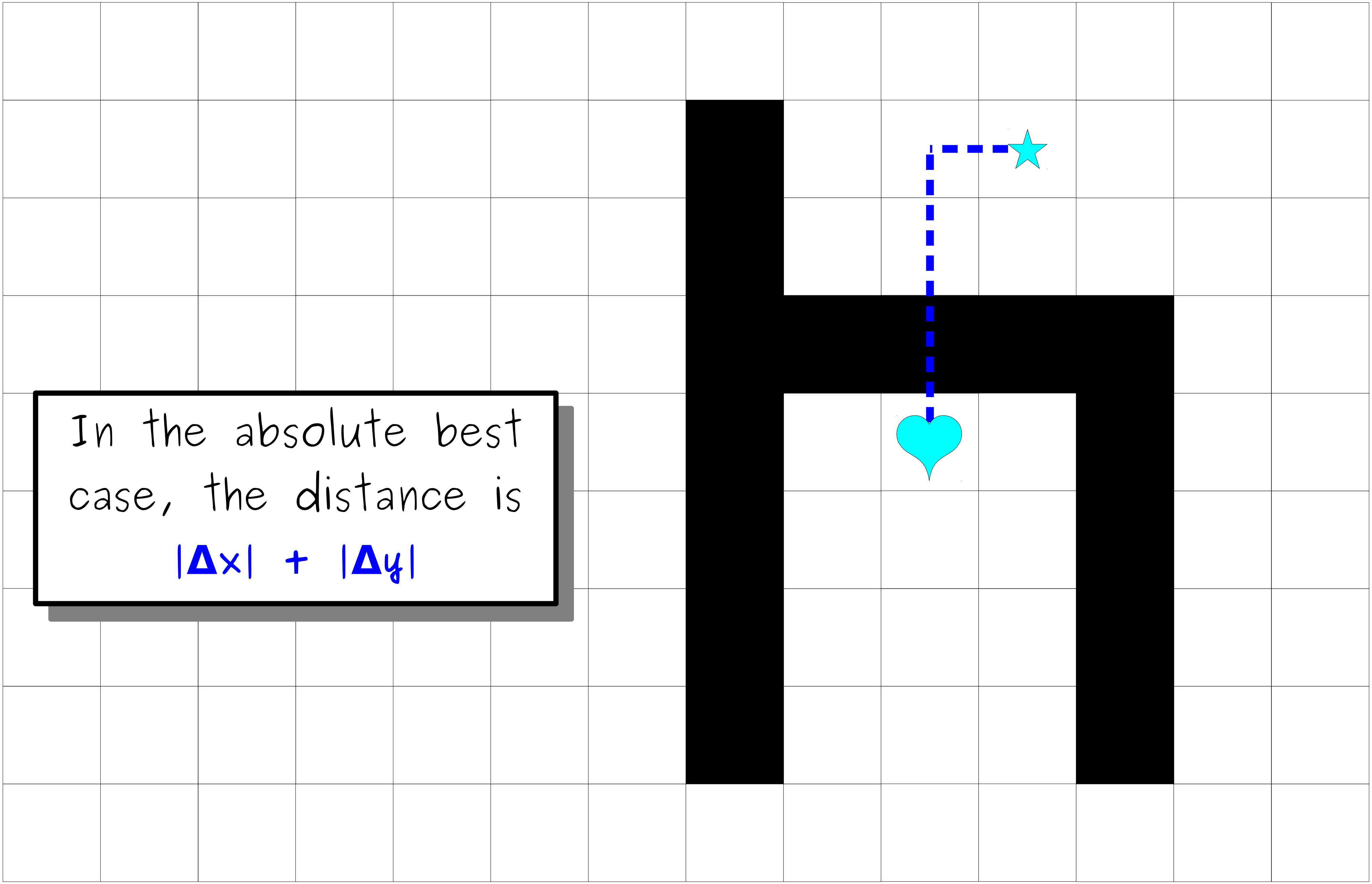
- We're looking for a virtuous golden mean between two extremes:
  - The vice of deficiency: making no assumptions whatsoever about the graph structure.
  - The vice of excess: having to know everything about the graph in order to provide assistance.
- ***Idea:*** Look for some kind of compromise between the two.





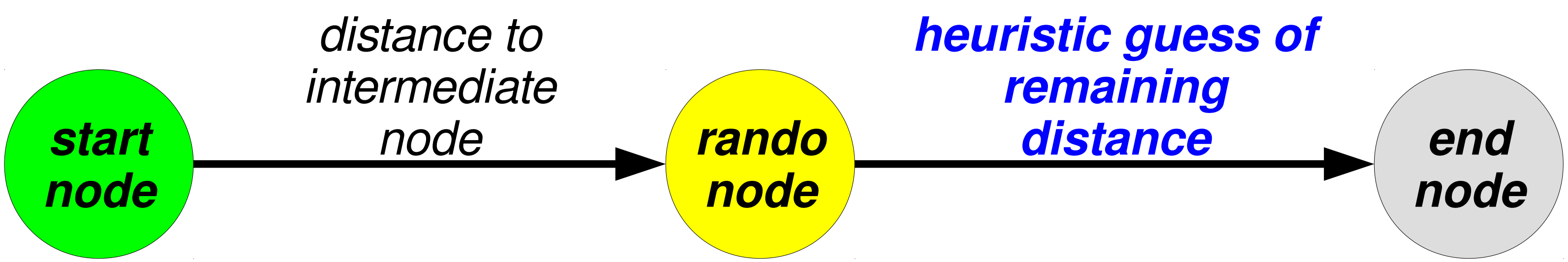




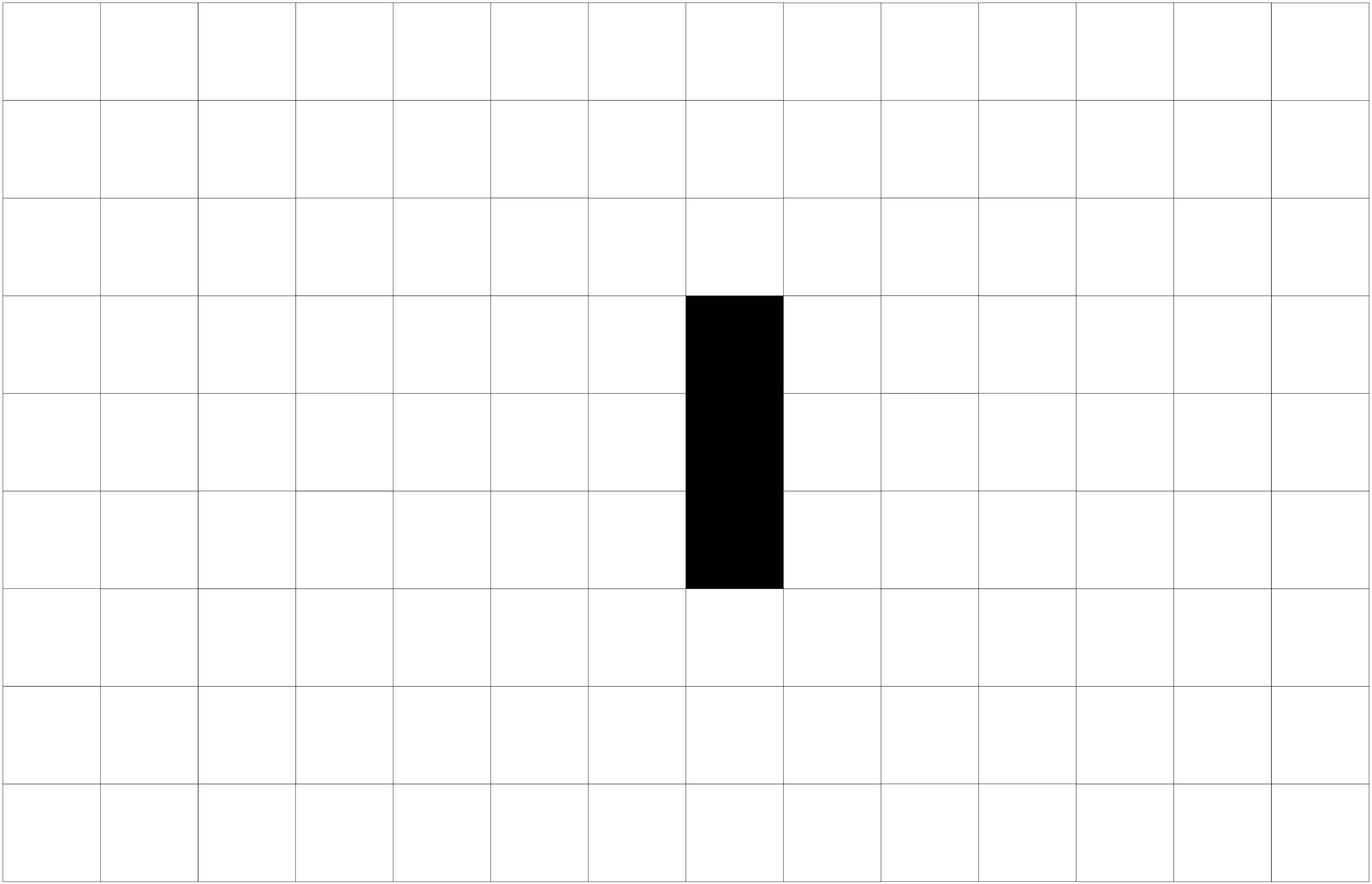

In the absolute best case, the distance is
|Δx| + |Δy|

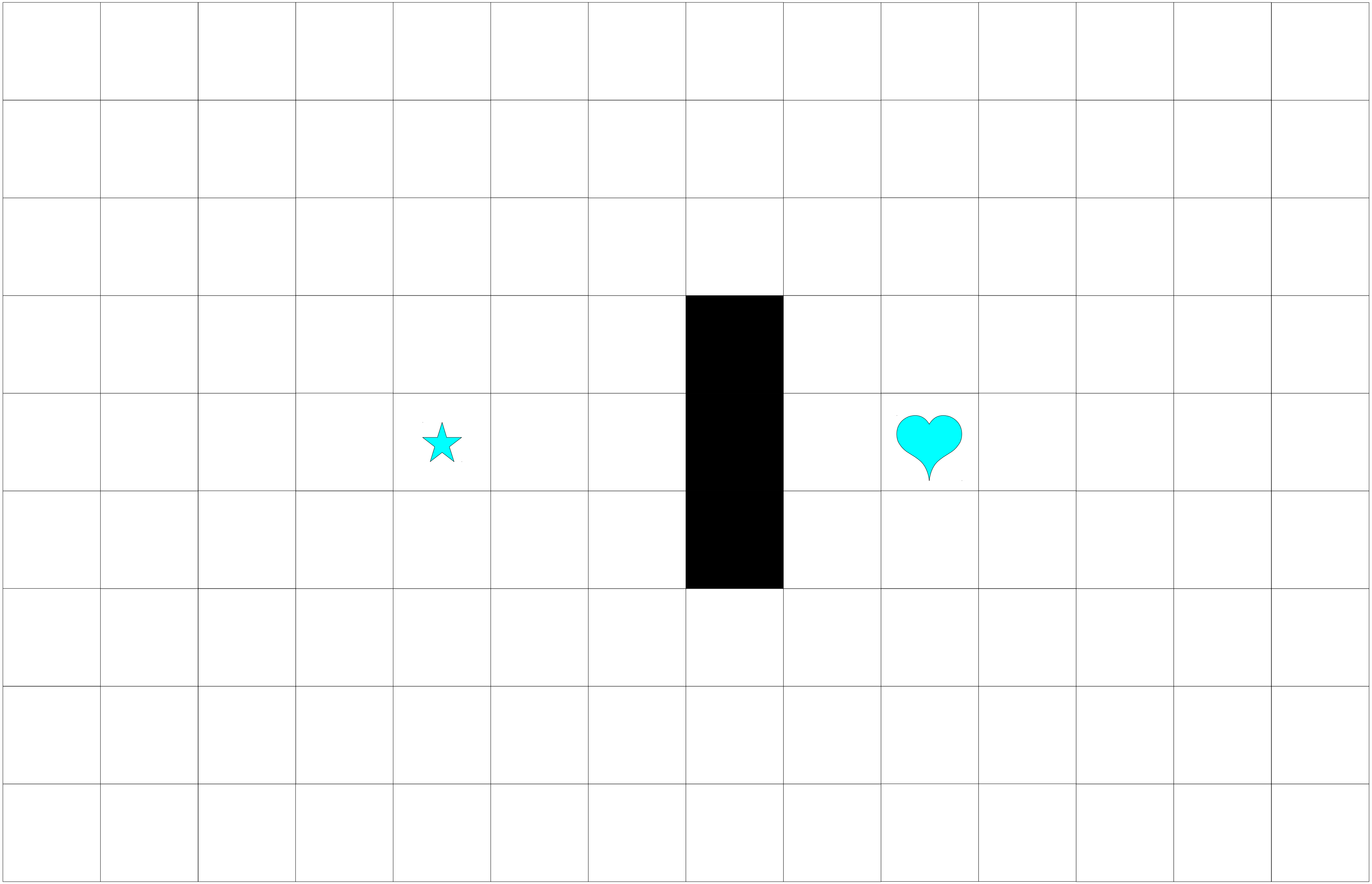
Heuristic Functions

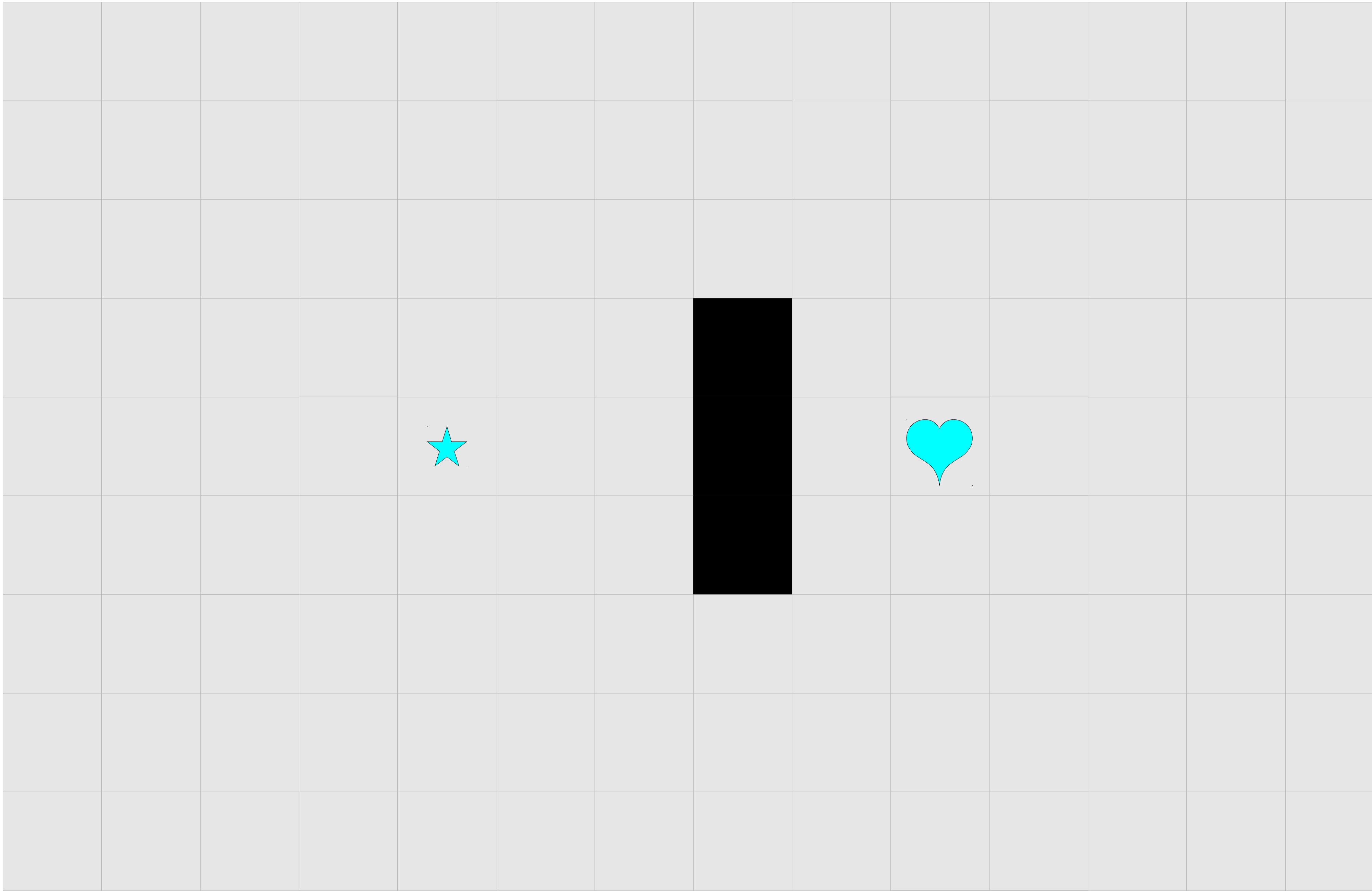
- In the context of a graph search, a ***heuristic function*** is a function that makes a "guess" of the distance from a node to the destination node.

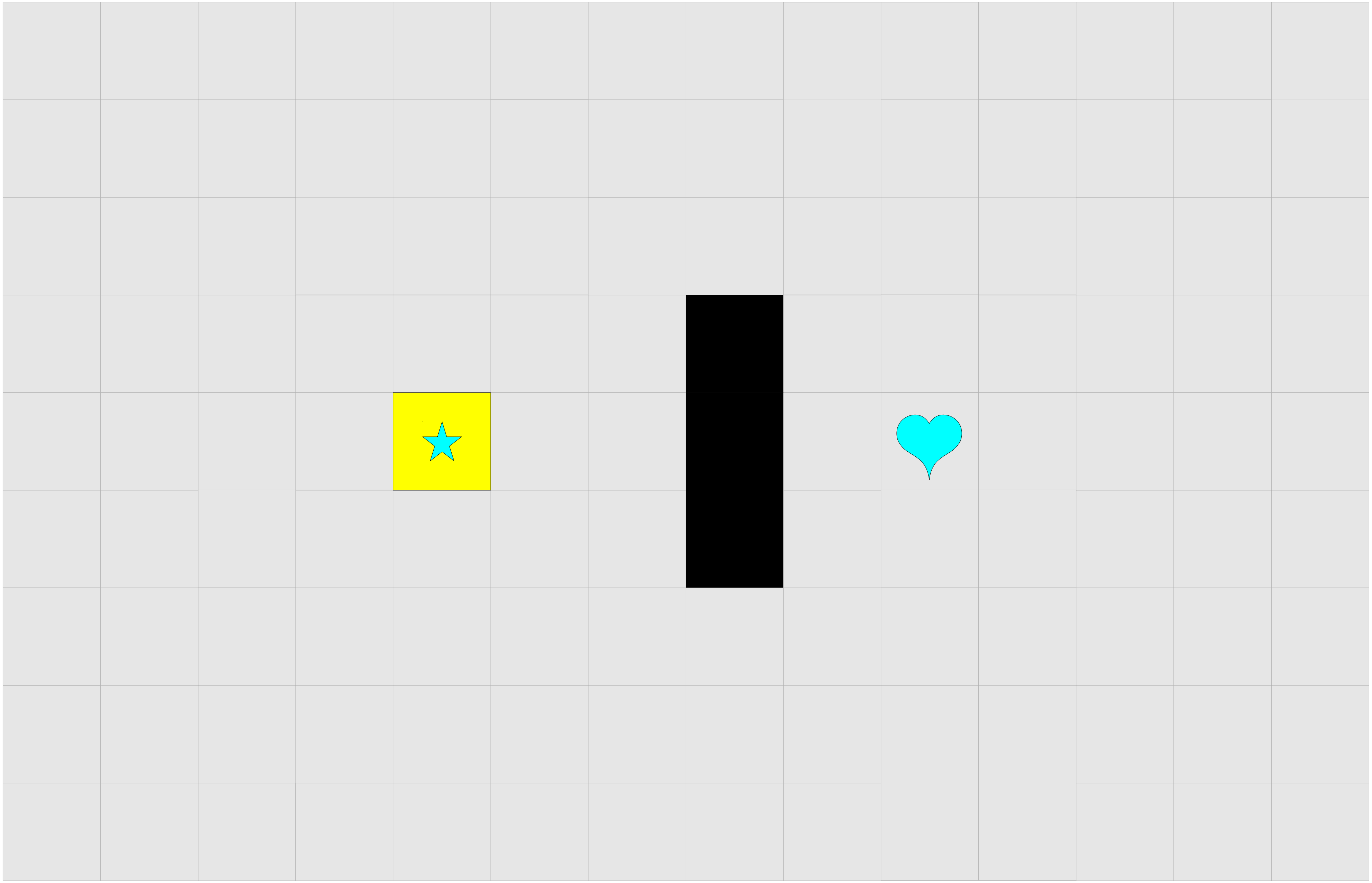


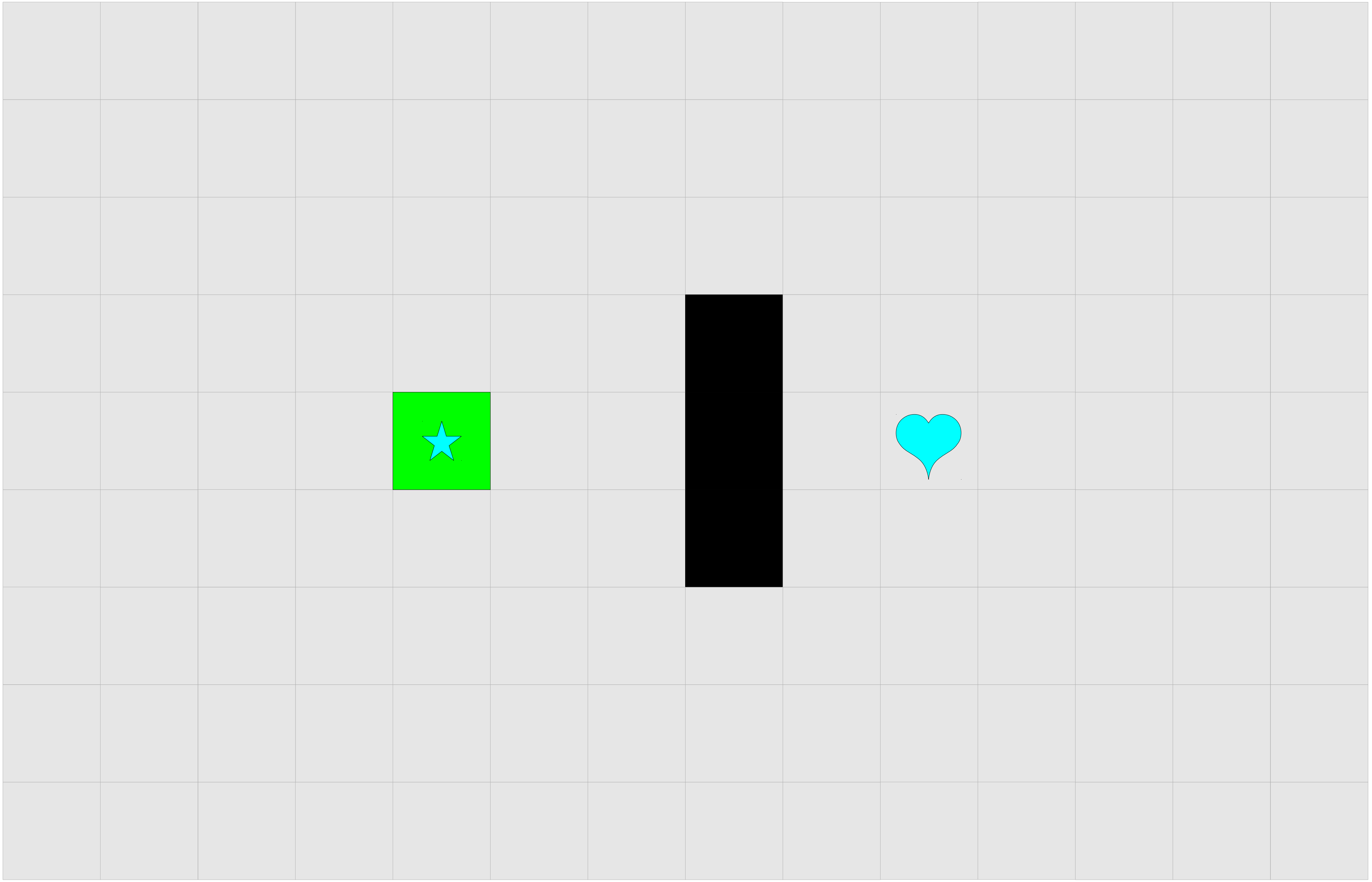
- An ***admissible*** heuristic is one that never overestimates the true distance.

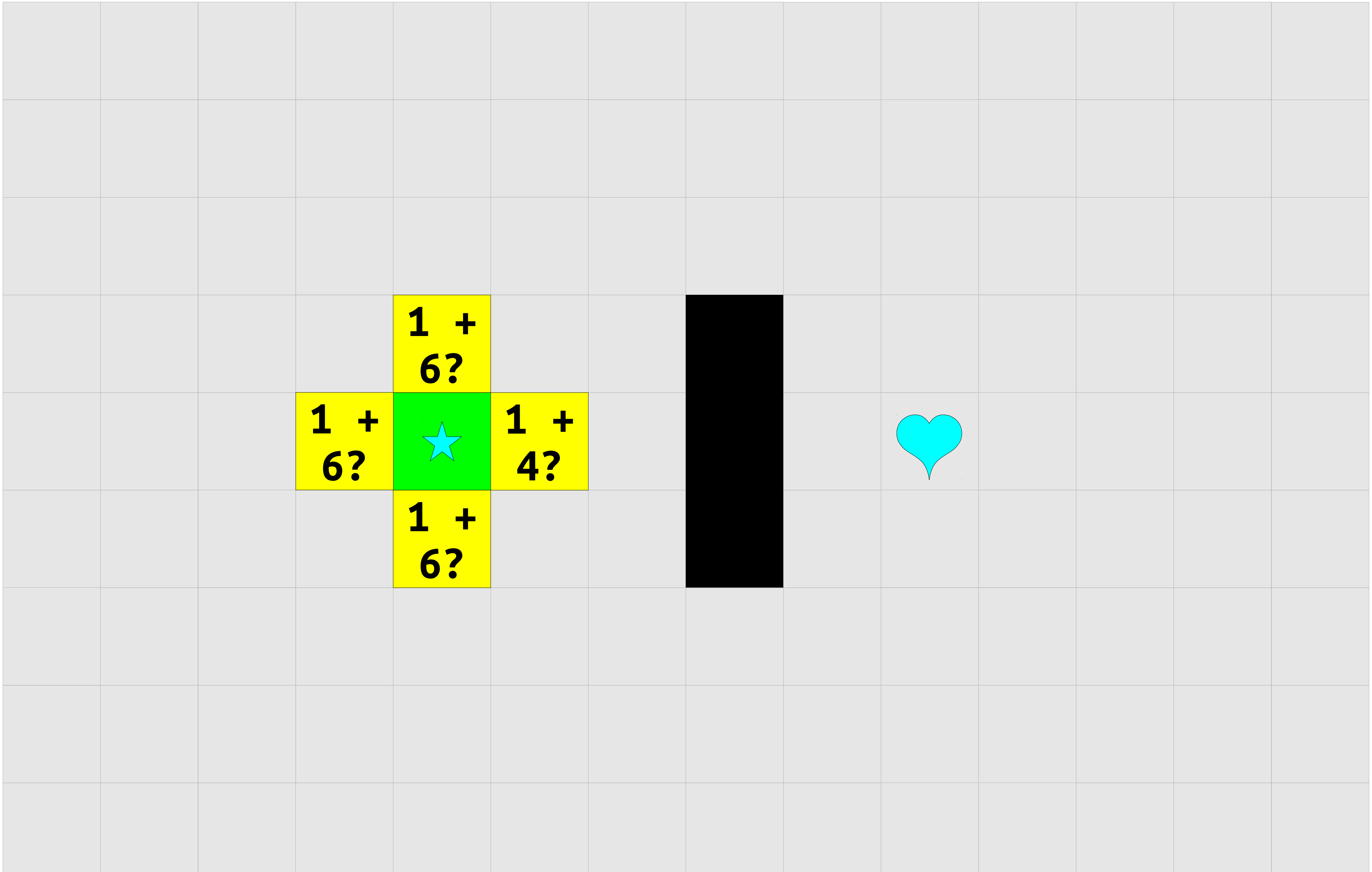


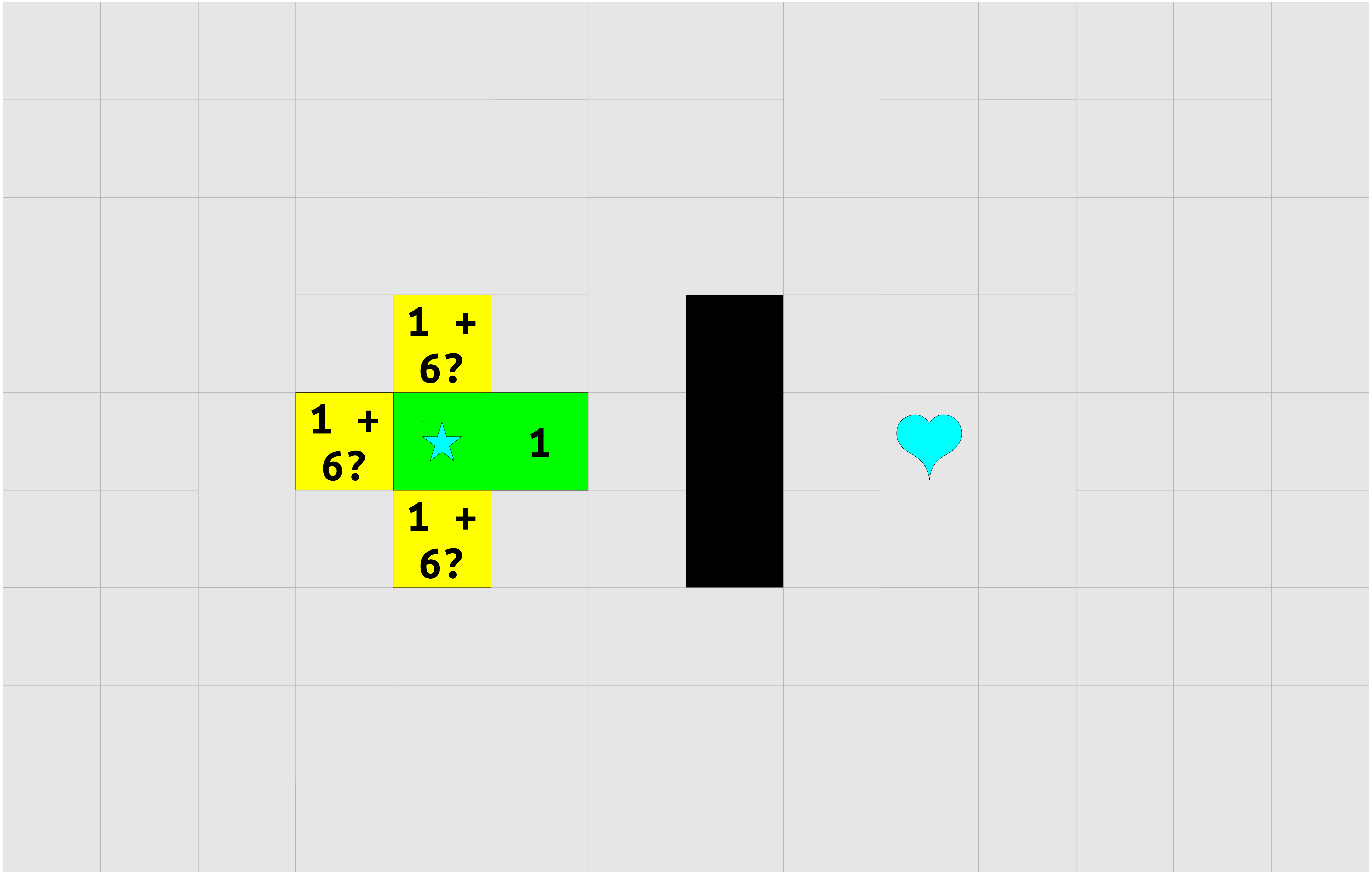


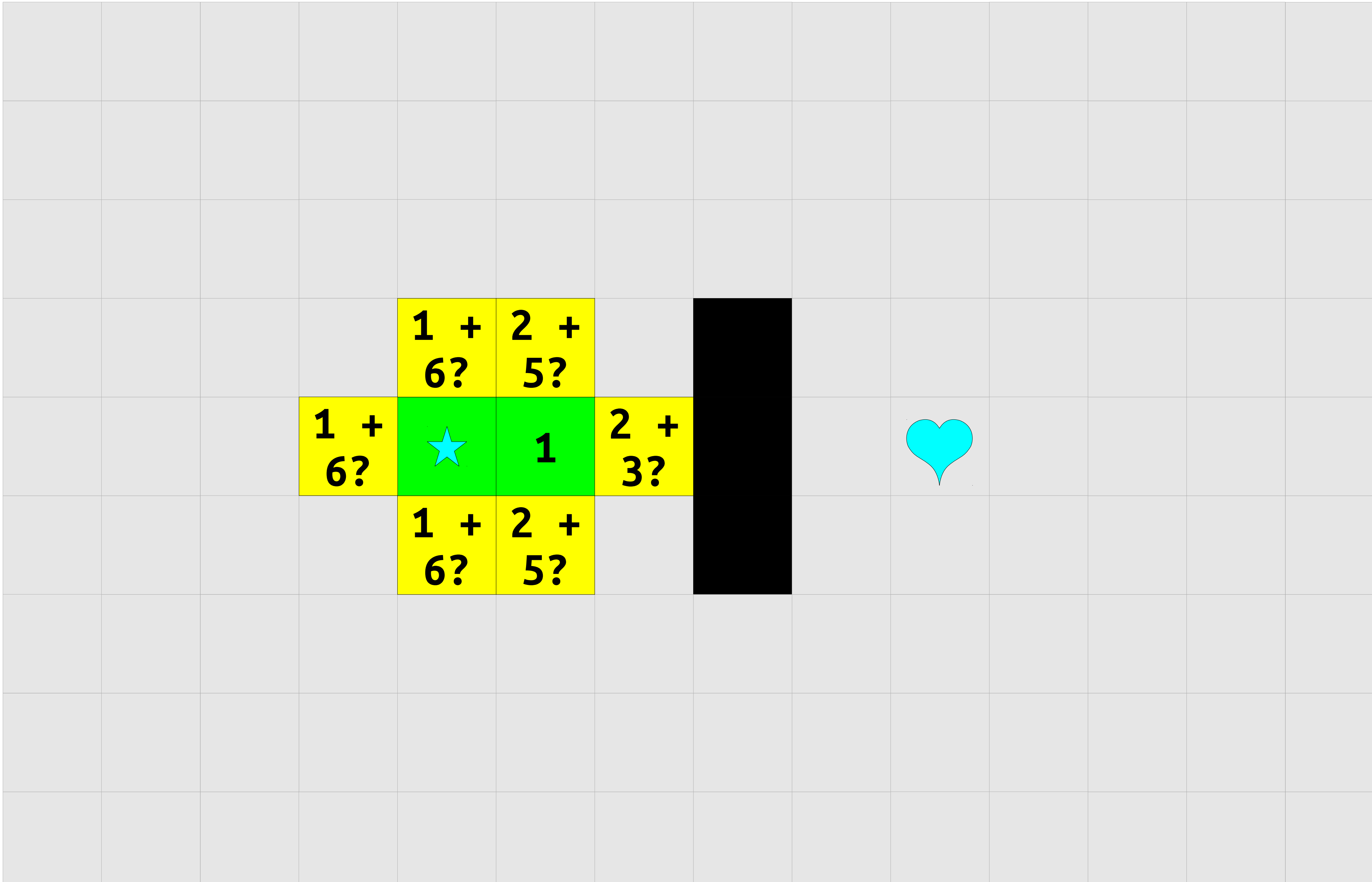




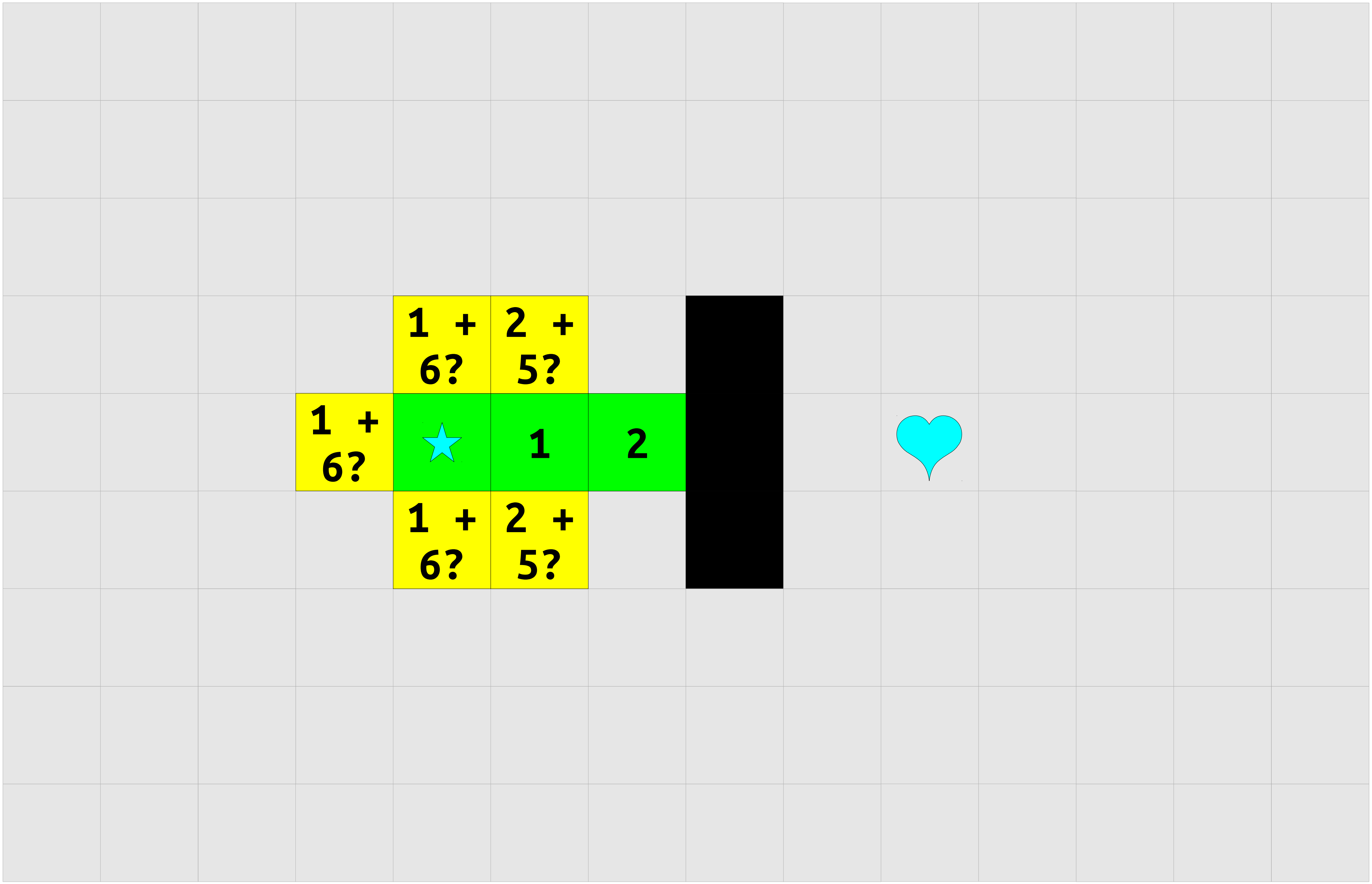


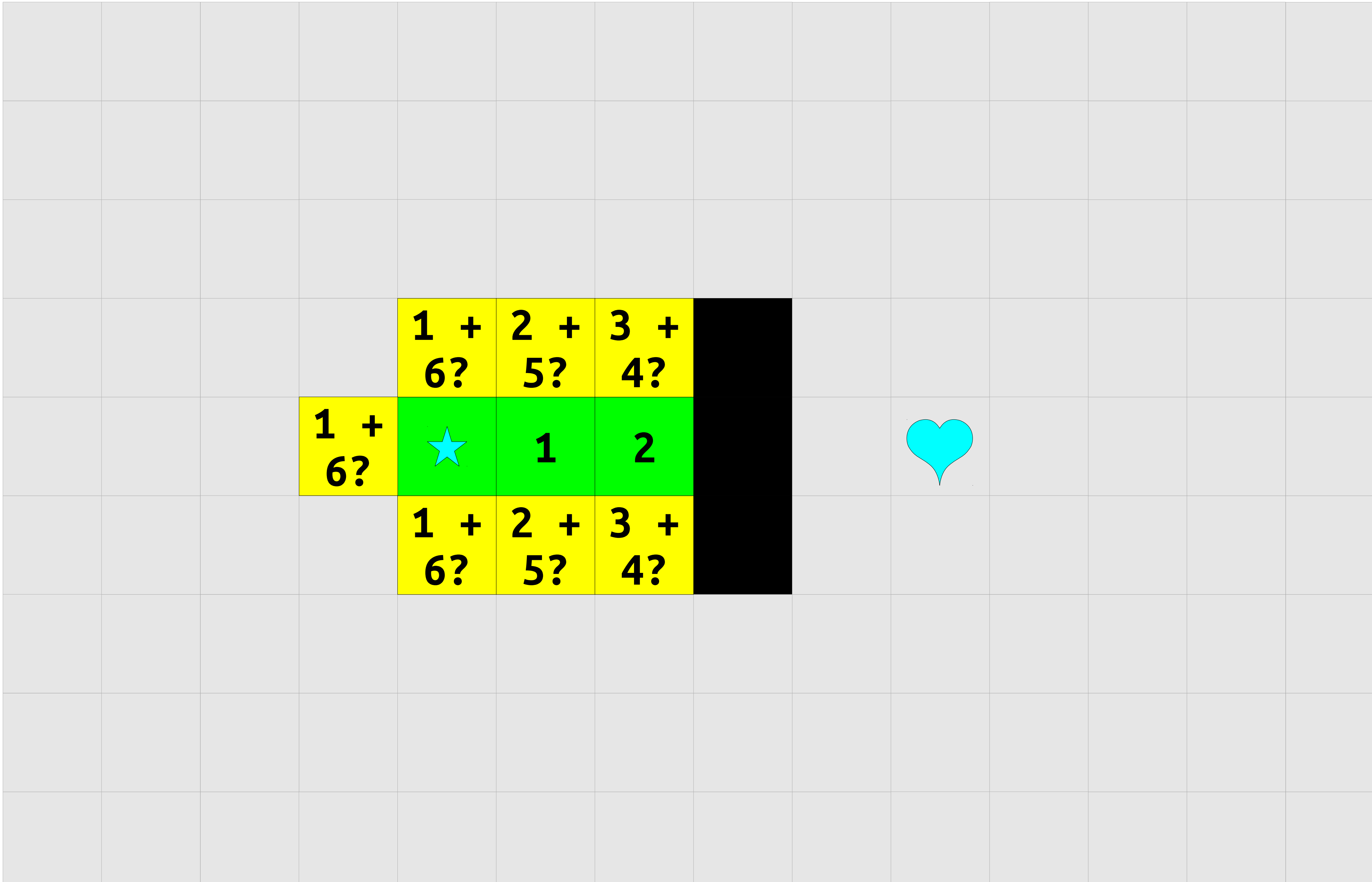

1 + 6?
1 + 6?
1 + 4?
1 + 6?

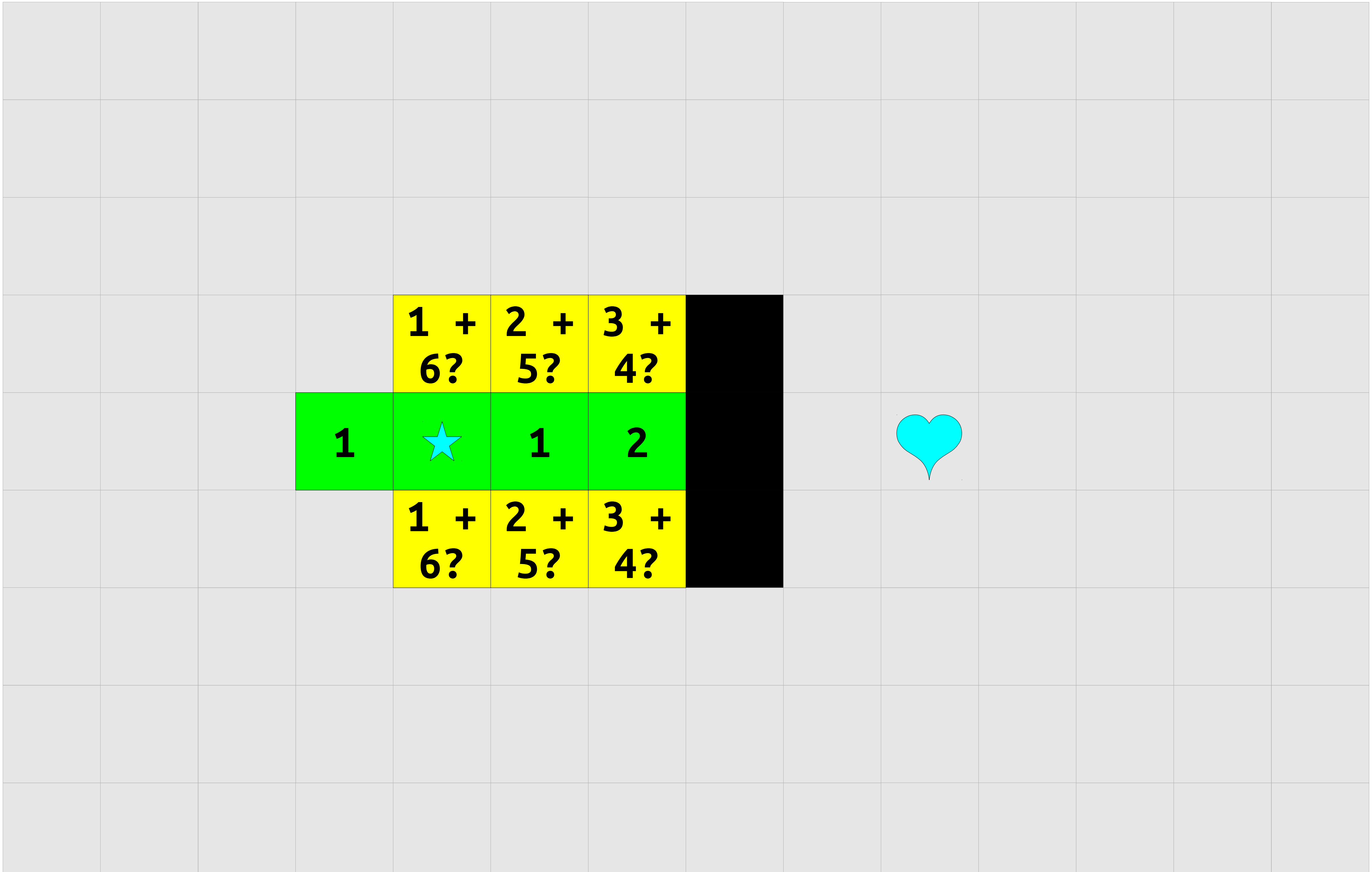

1 + 6?
1 + 6?
1
1 + 6?

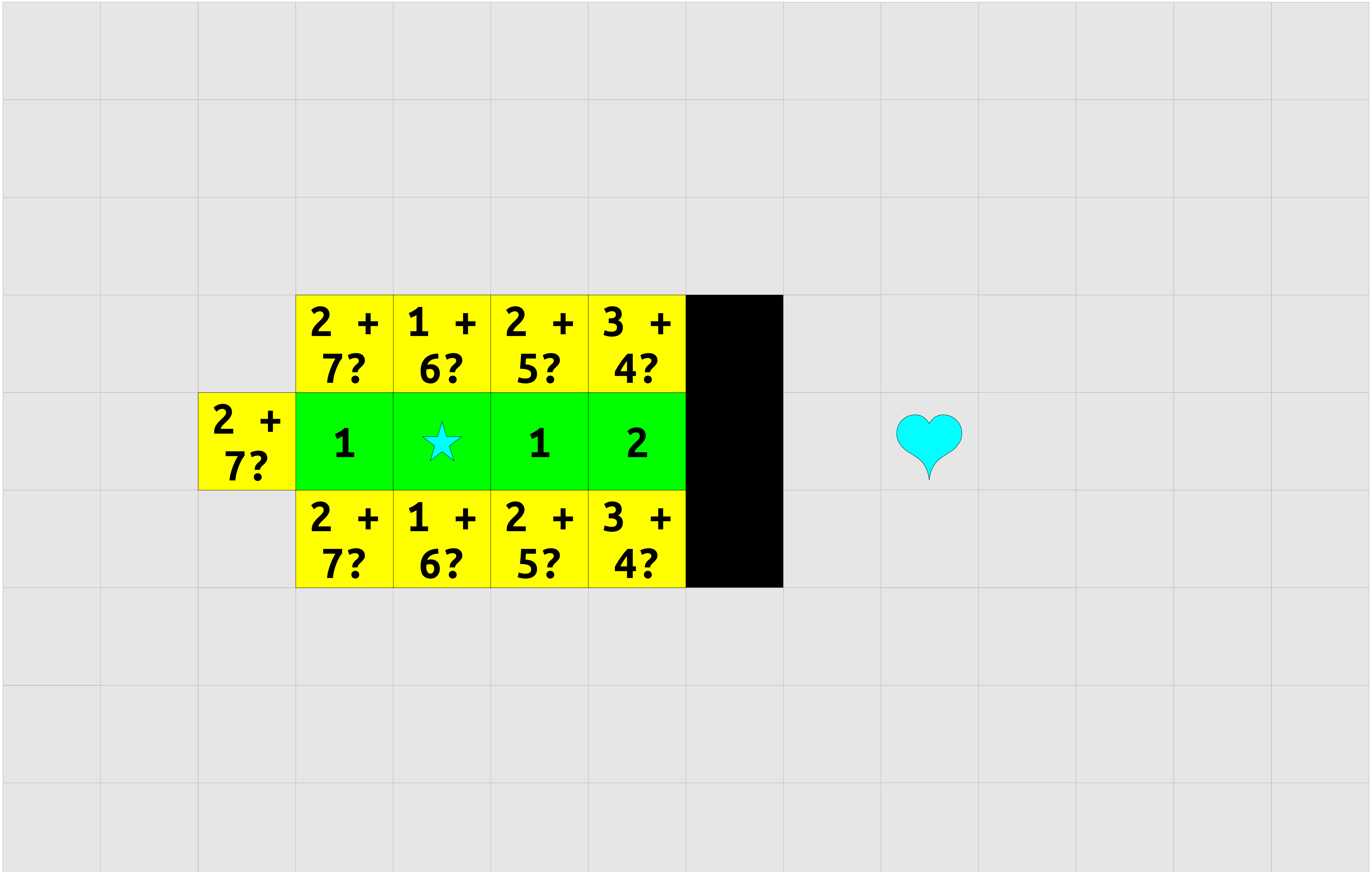


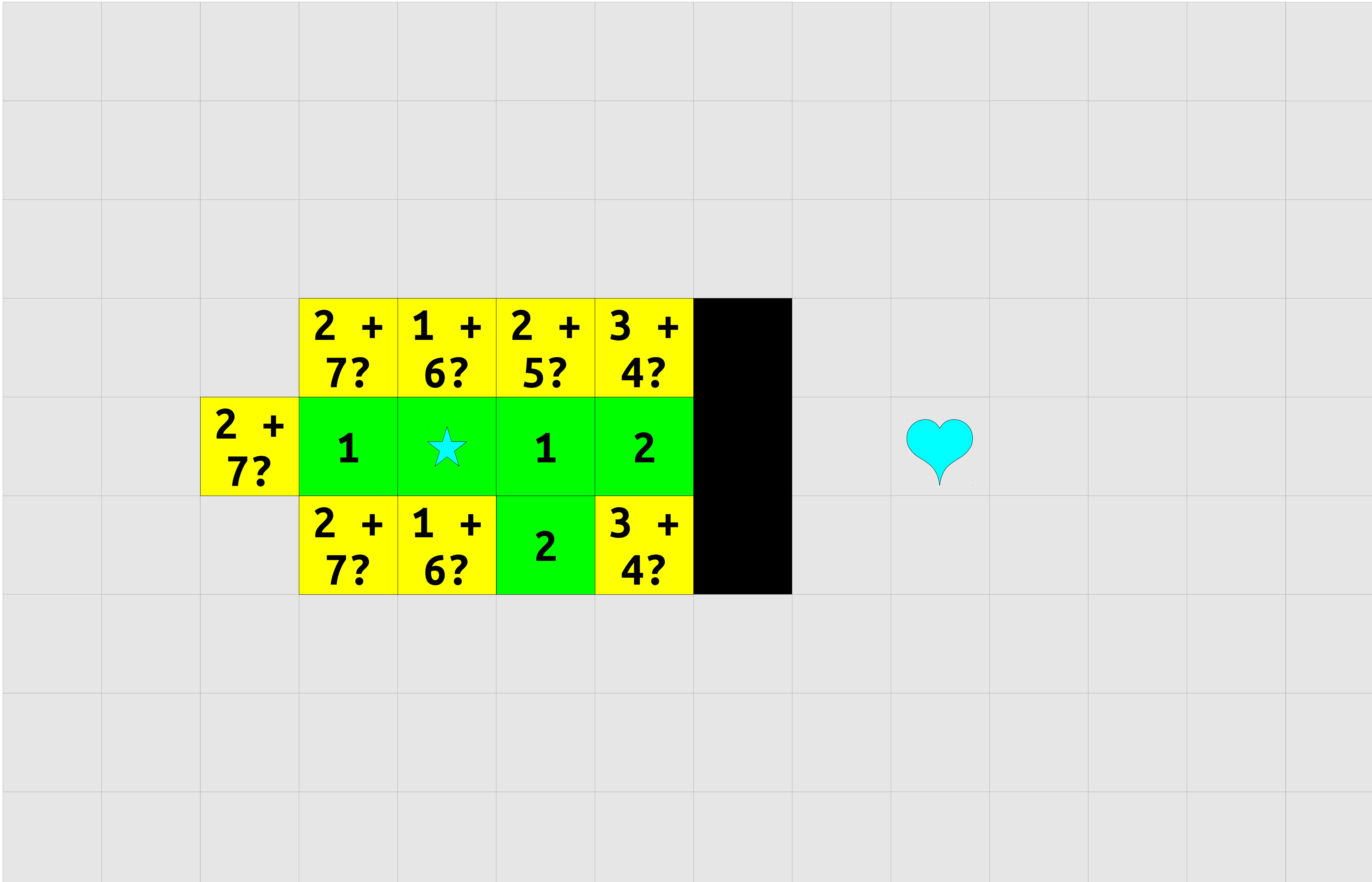
1 + 6?
2 + 5?
1 + 6?
1
2 + 3?
1 + 6?
2 + 5?

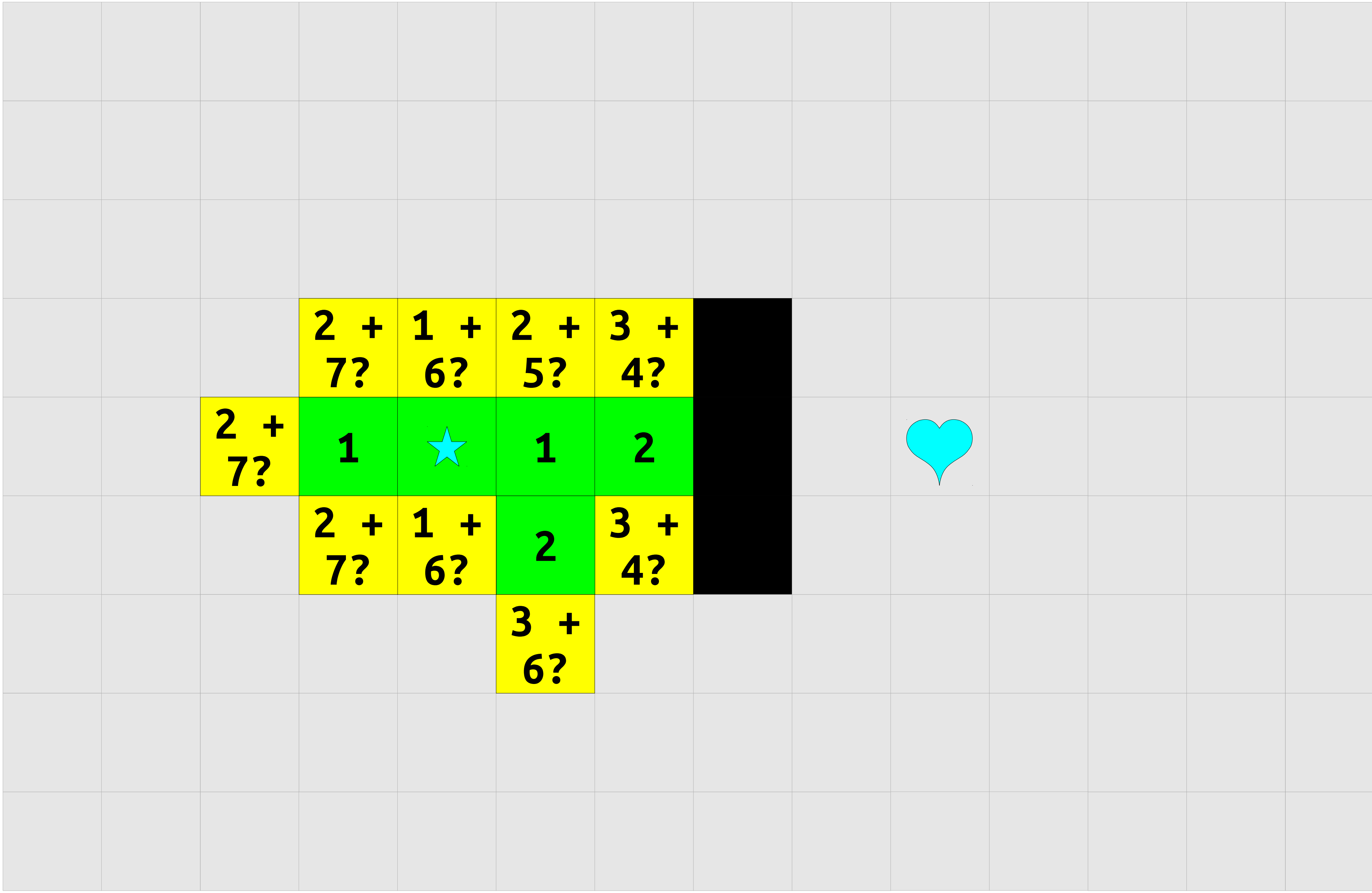

1 + 6?
2 + 5?
1 + 6?
1
2
1 + 6?
2 + 5?

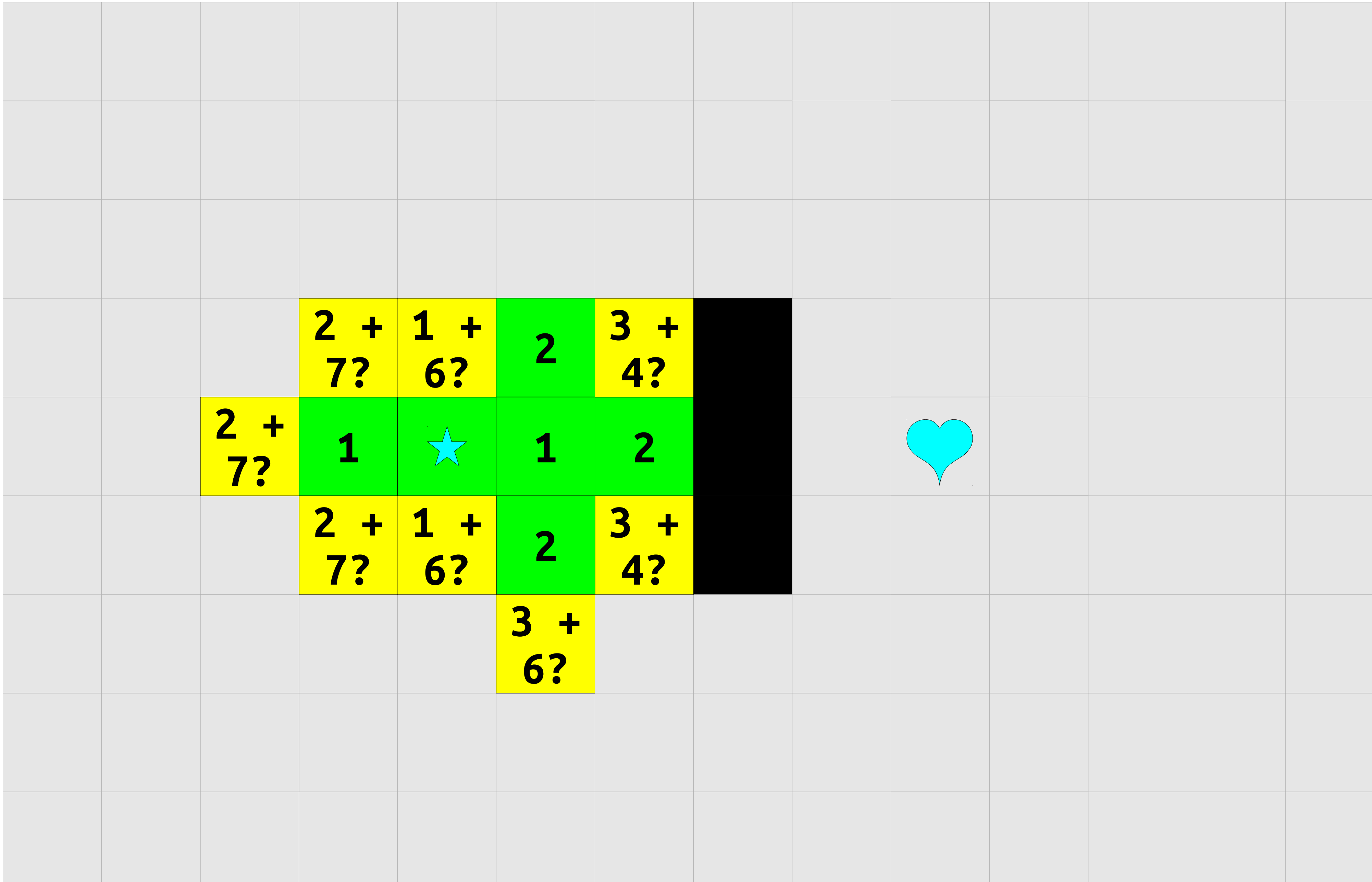

1 + 6?
2 + 5?
3 + 4?
1 + 6?
1
2
1 + 6?
2 + 5?
3 + 4?


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

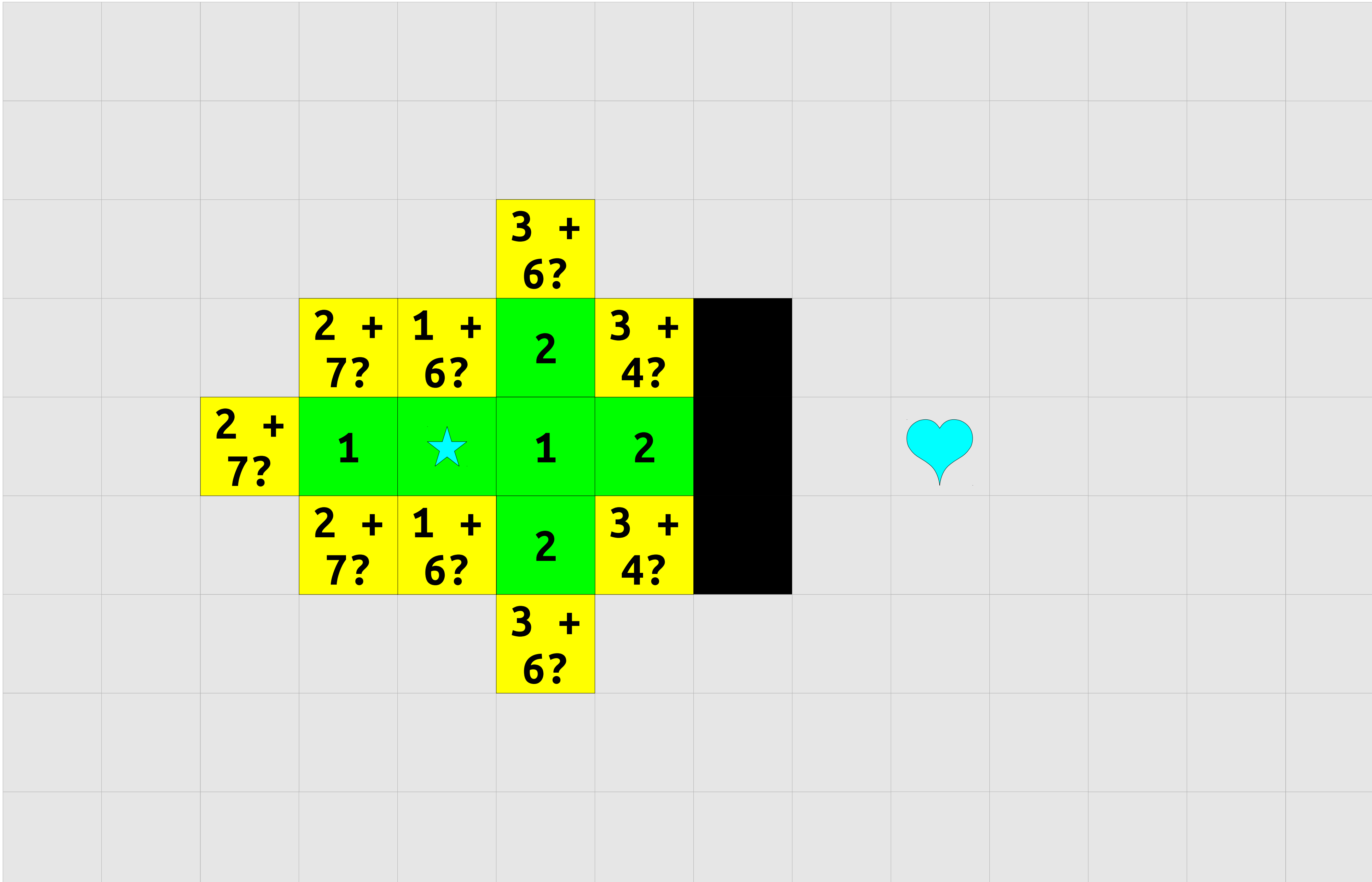

2 + 7?
1 + 6?
2 + 5?
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2 + 5?
3 + 4?

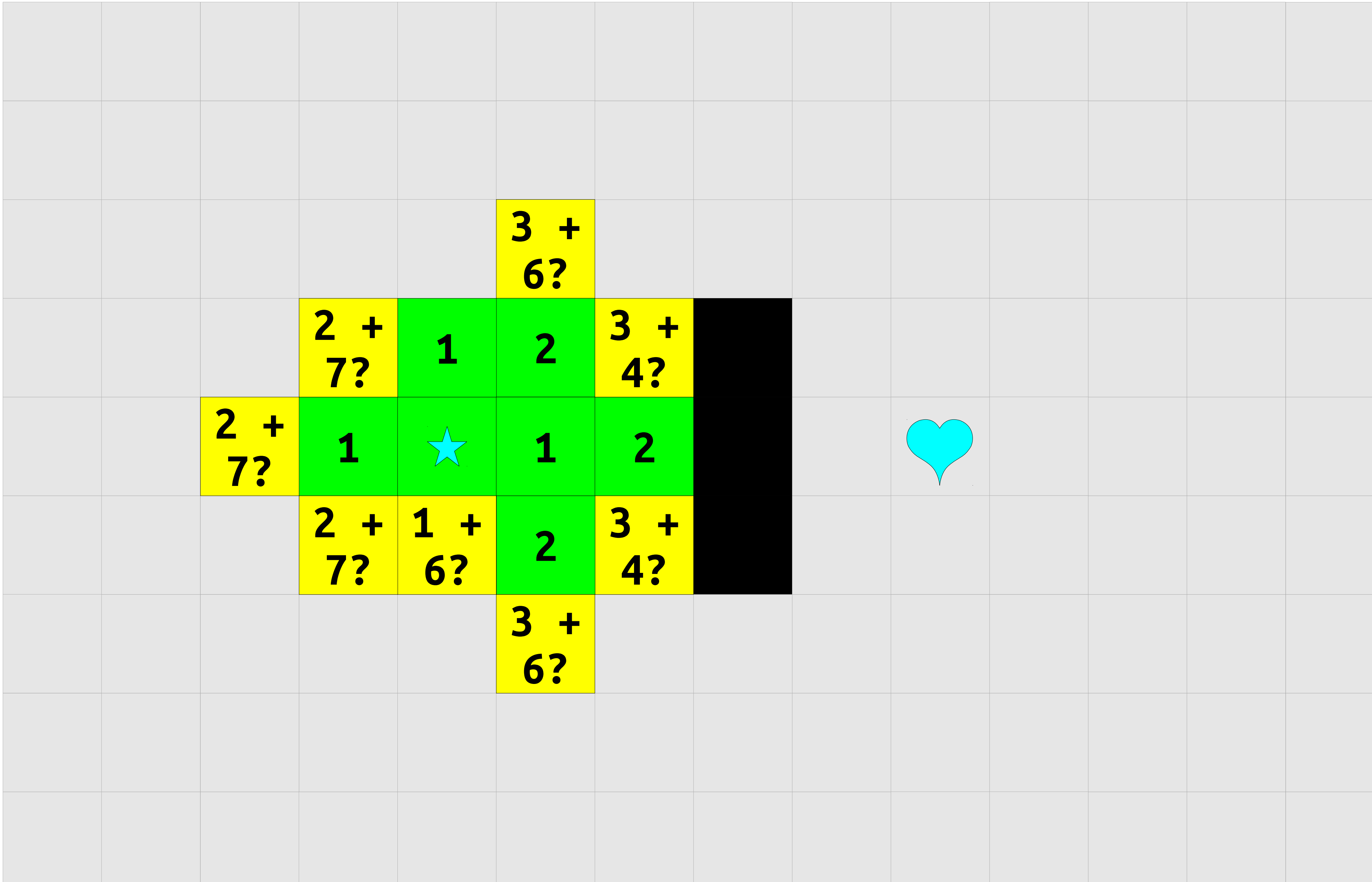

2 + 7?
1 + 6?
2 + 5?
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?


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

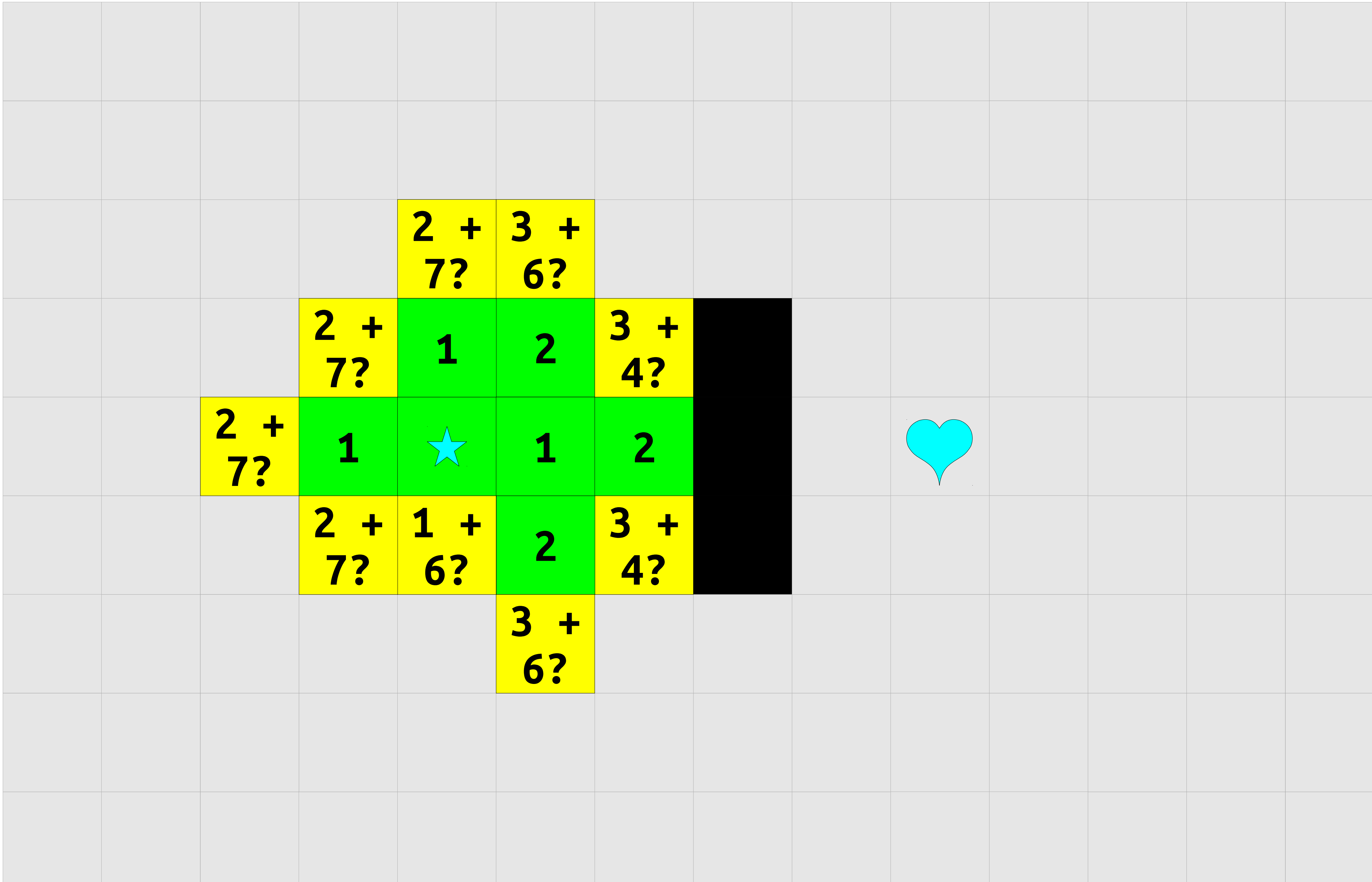


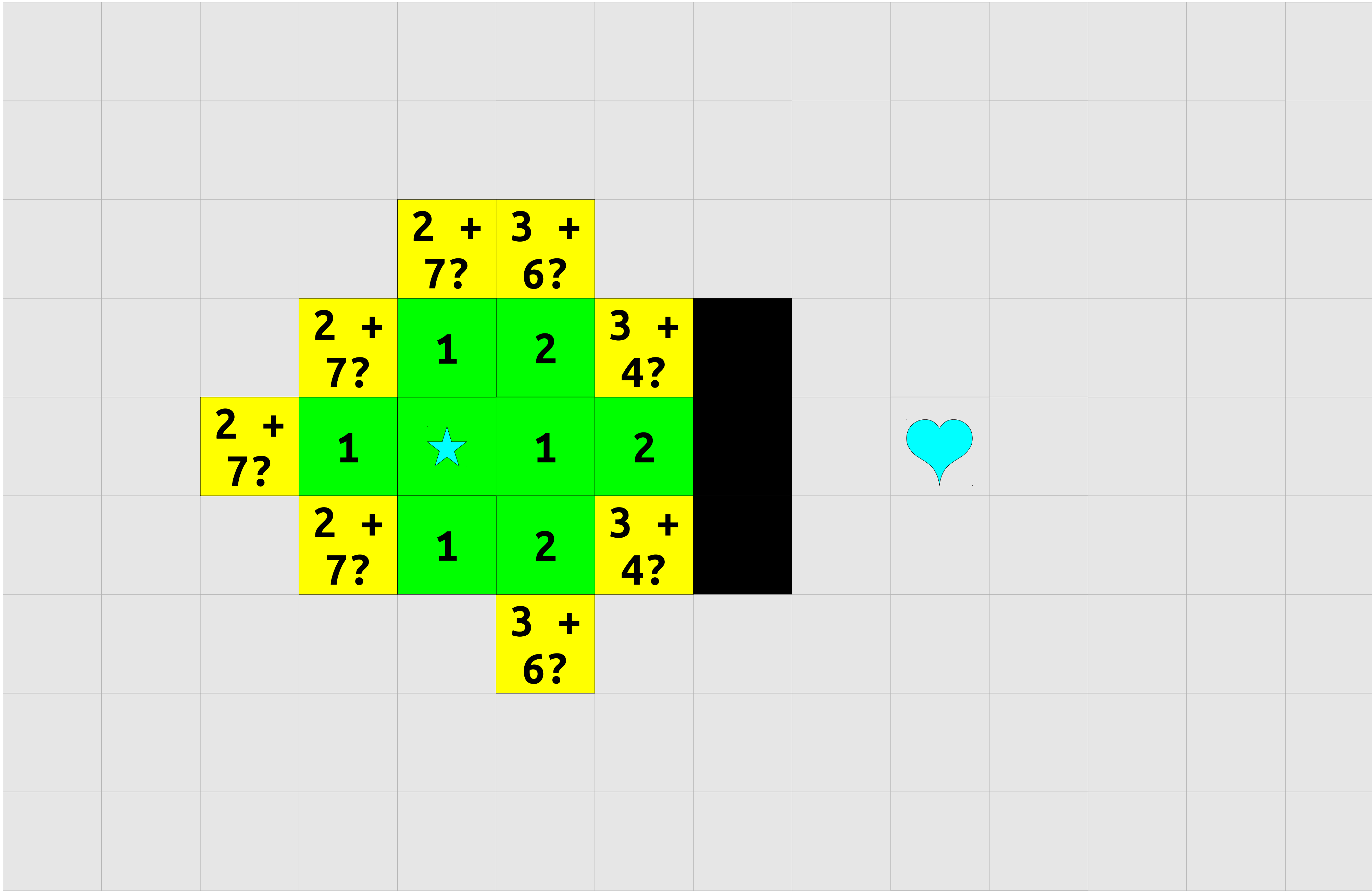
2 + 7?
1 + 6?
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?
3 + 6?

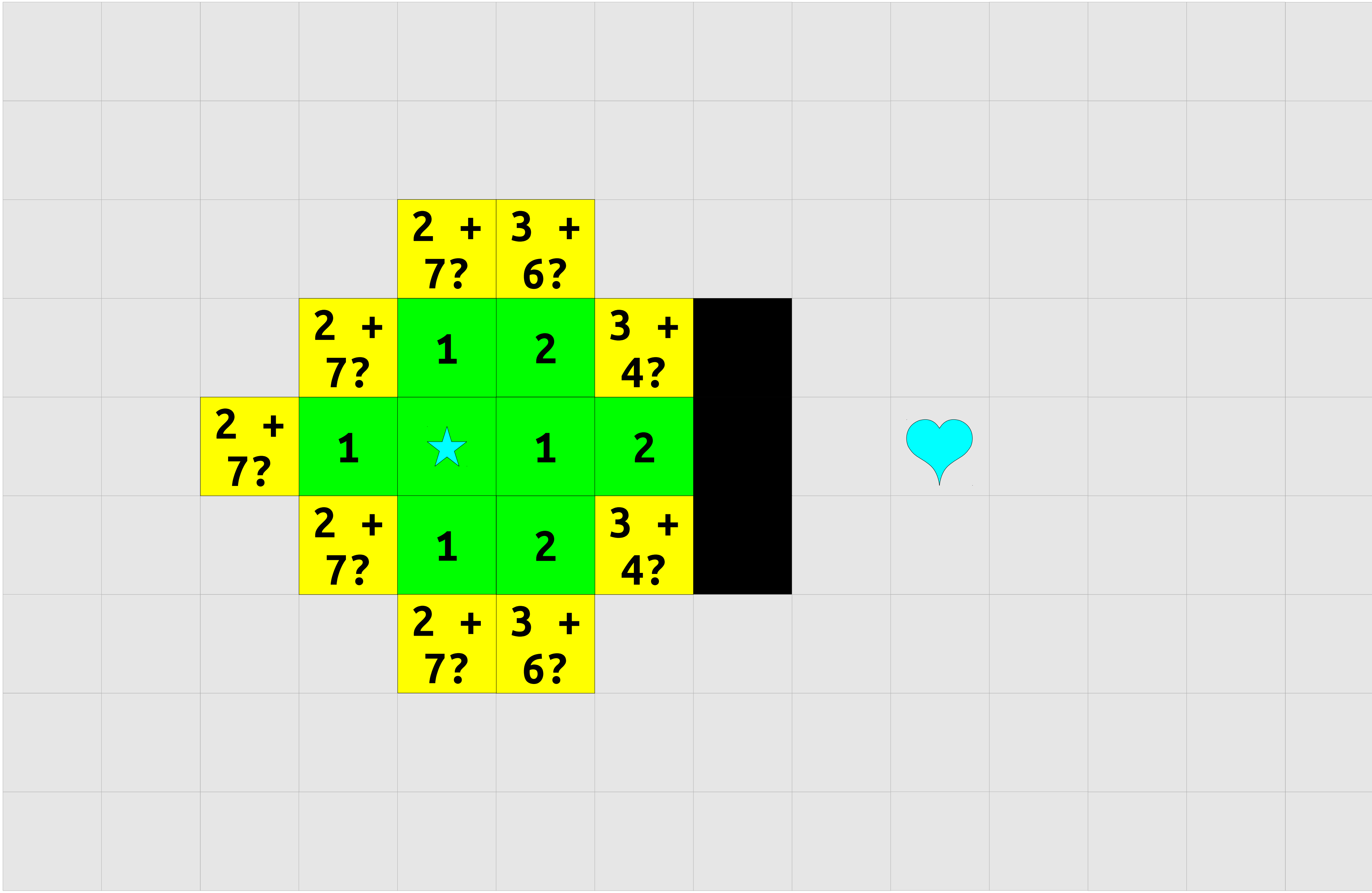

3 + 6?
2 + 7?
1 + 6?
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?
3 + 6?

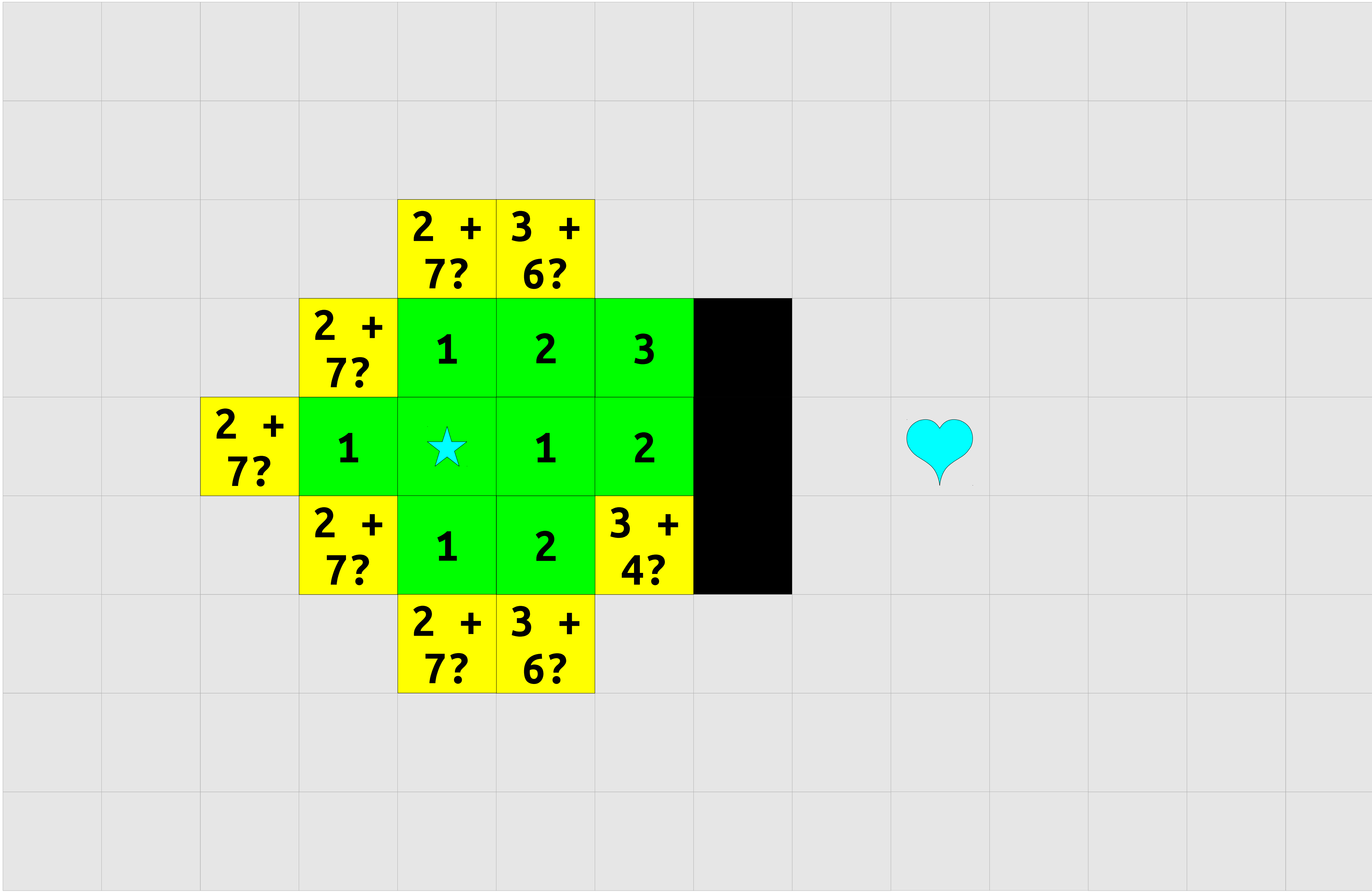


3 + 6?
2 + 7?
1
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?
3 + 6?

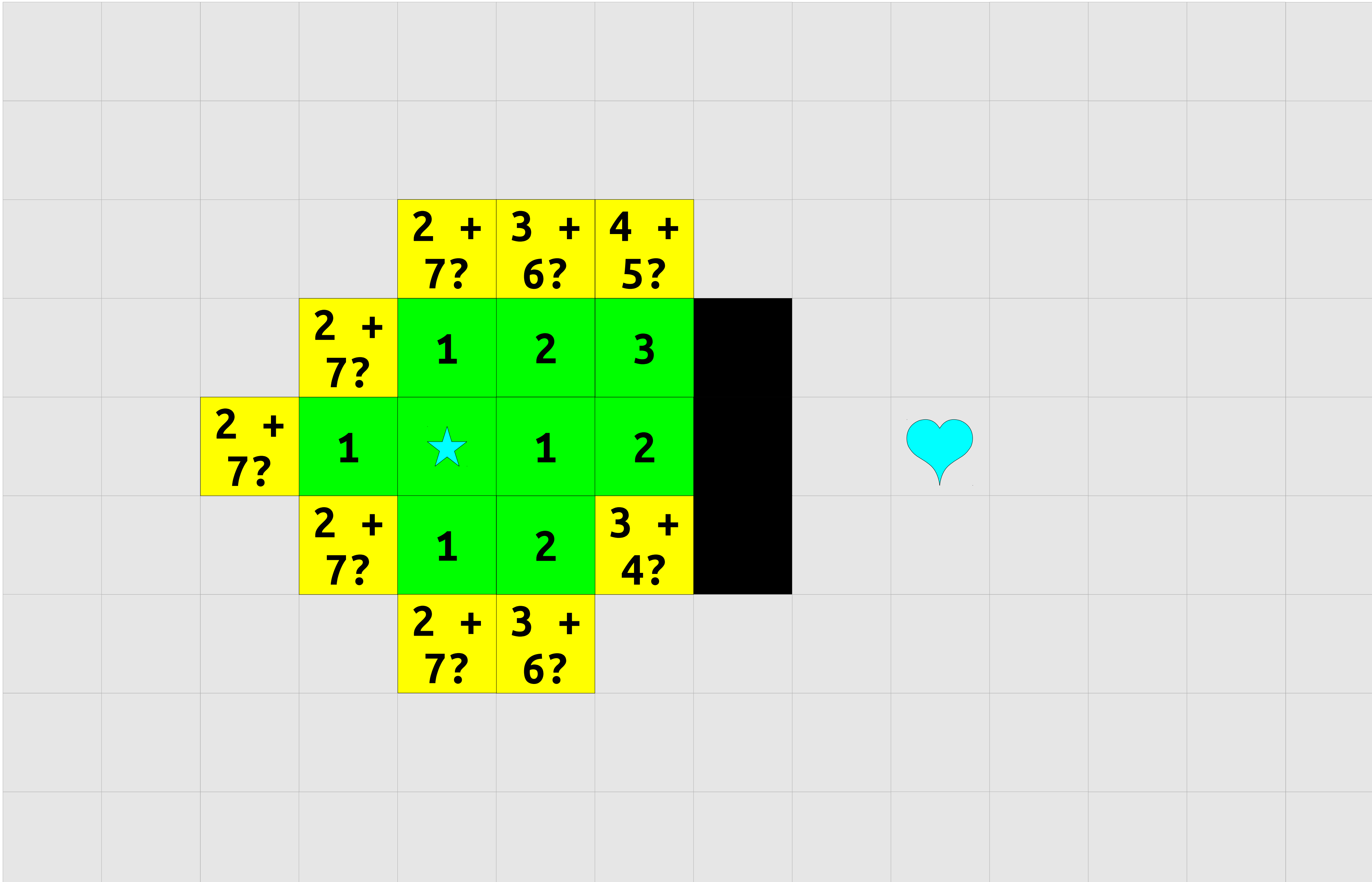

2 + 7?
3 + 6?
2 + 7?
1
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?
3 + 6?

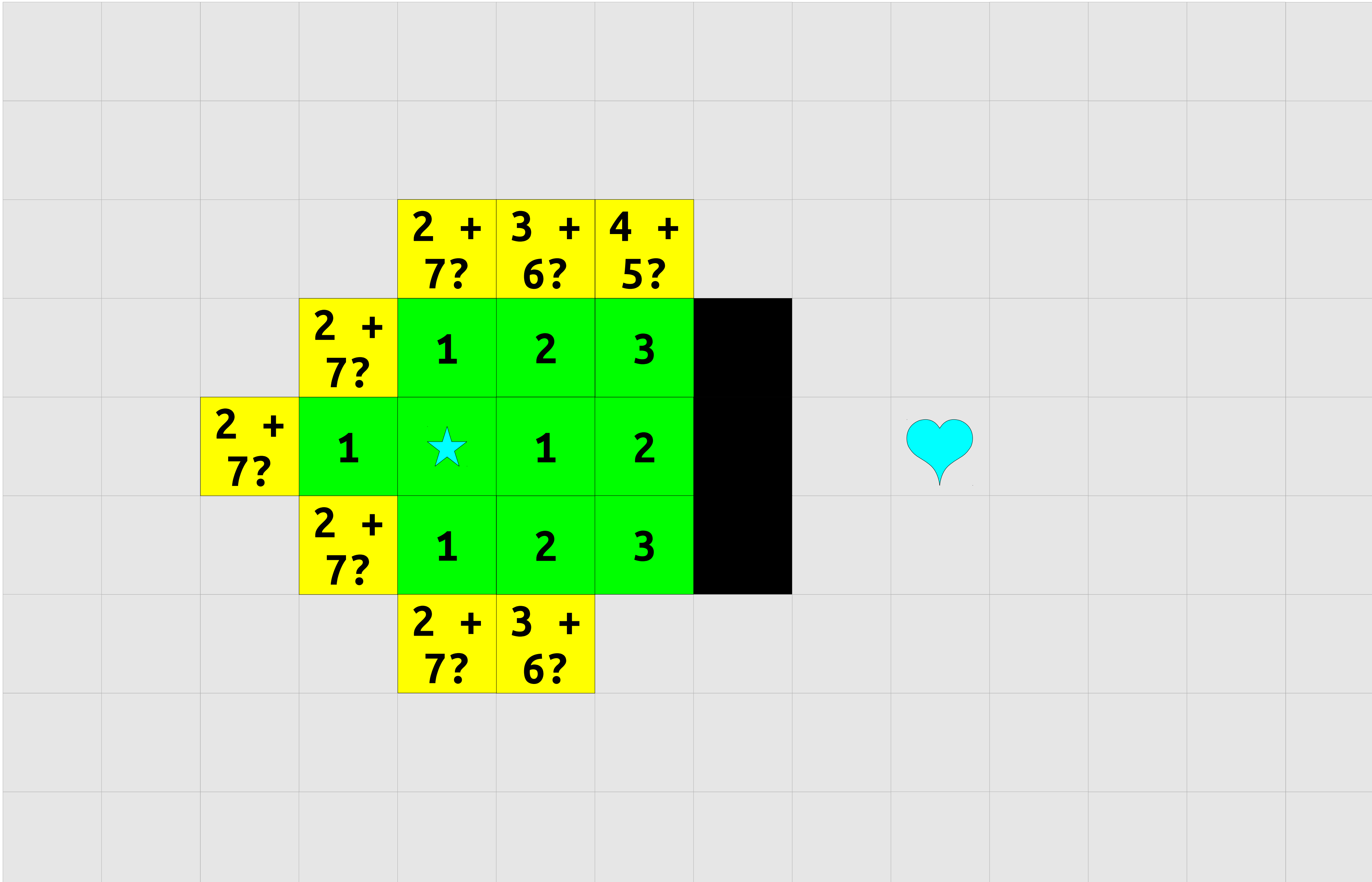

2 + 7?
3 + 6?
2 + 7?
1
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1
2
3 + 4?
3 + 6?

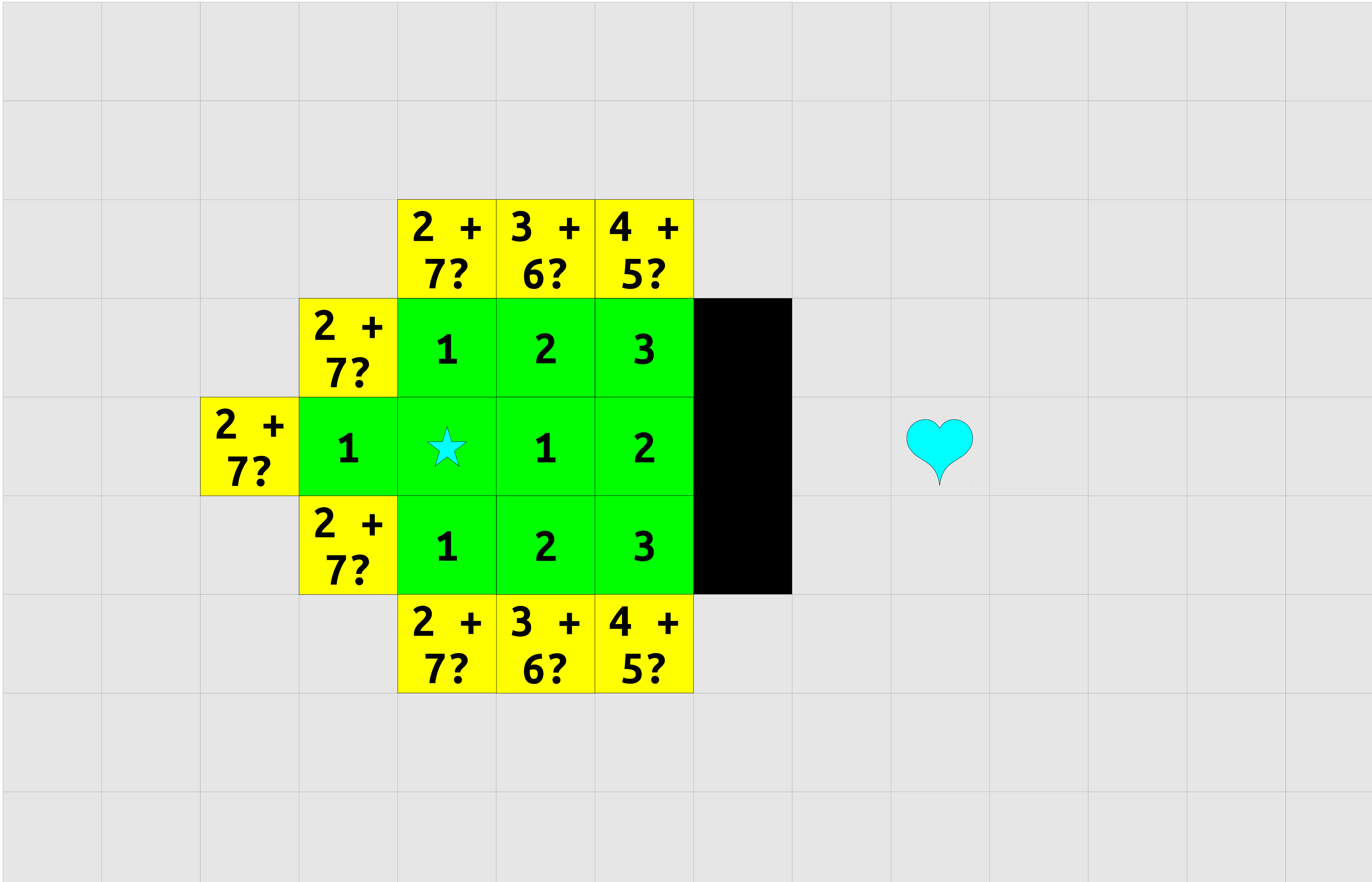

2 + 7?
3 + 6?
2 + 7?
1
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1
2
3 + 4?
2 + 7?
3 + 6?

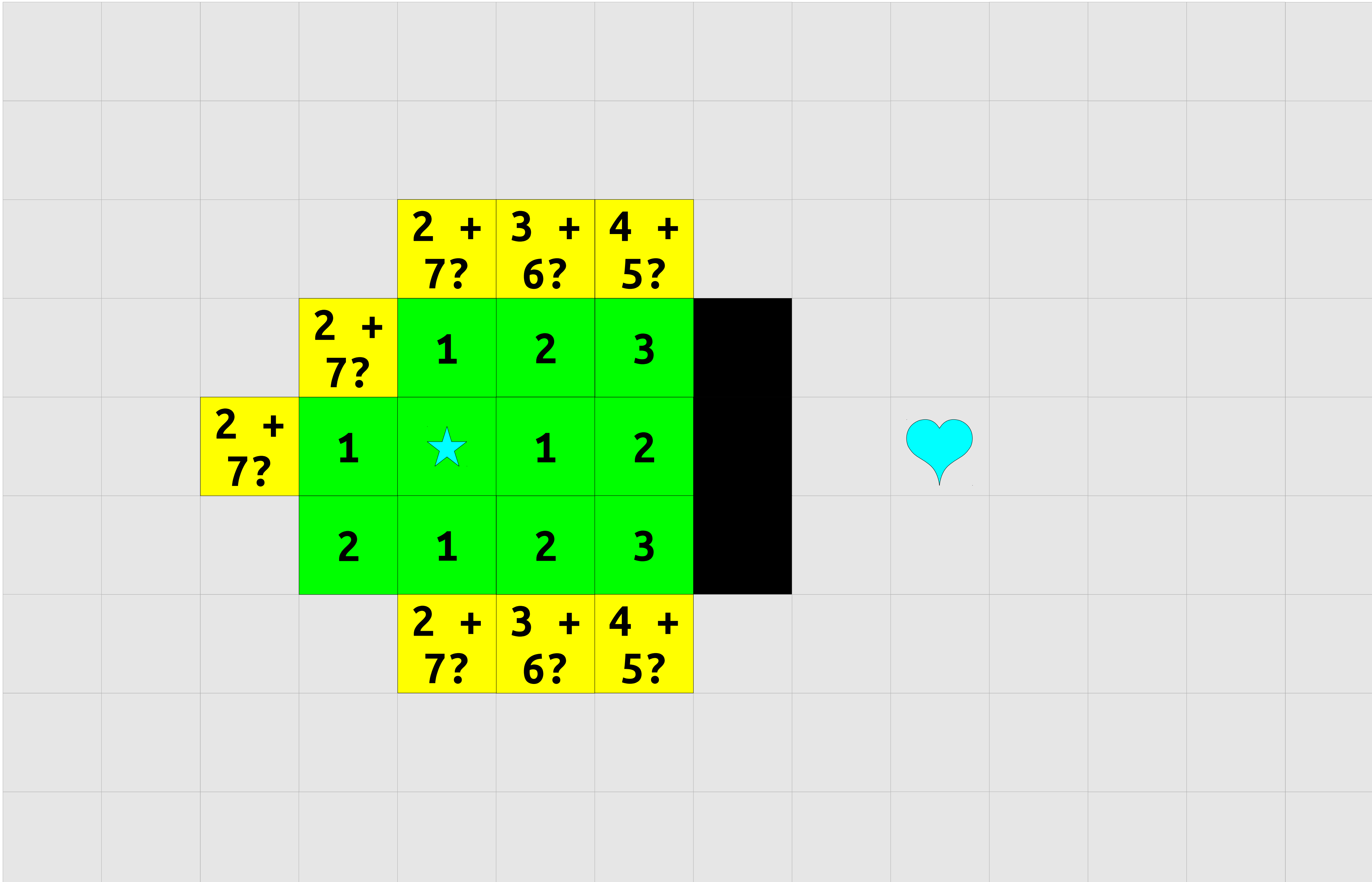


2 + 7?
3 + 6?
2 + 7?
1
2
3
2 + 7?
1
1
2
2 + 7?
1
2
3 + 4?
2 + 7?
3 + 6?

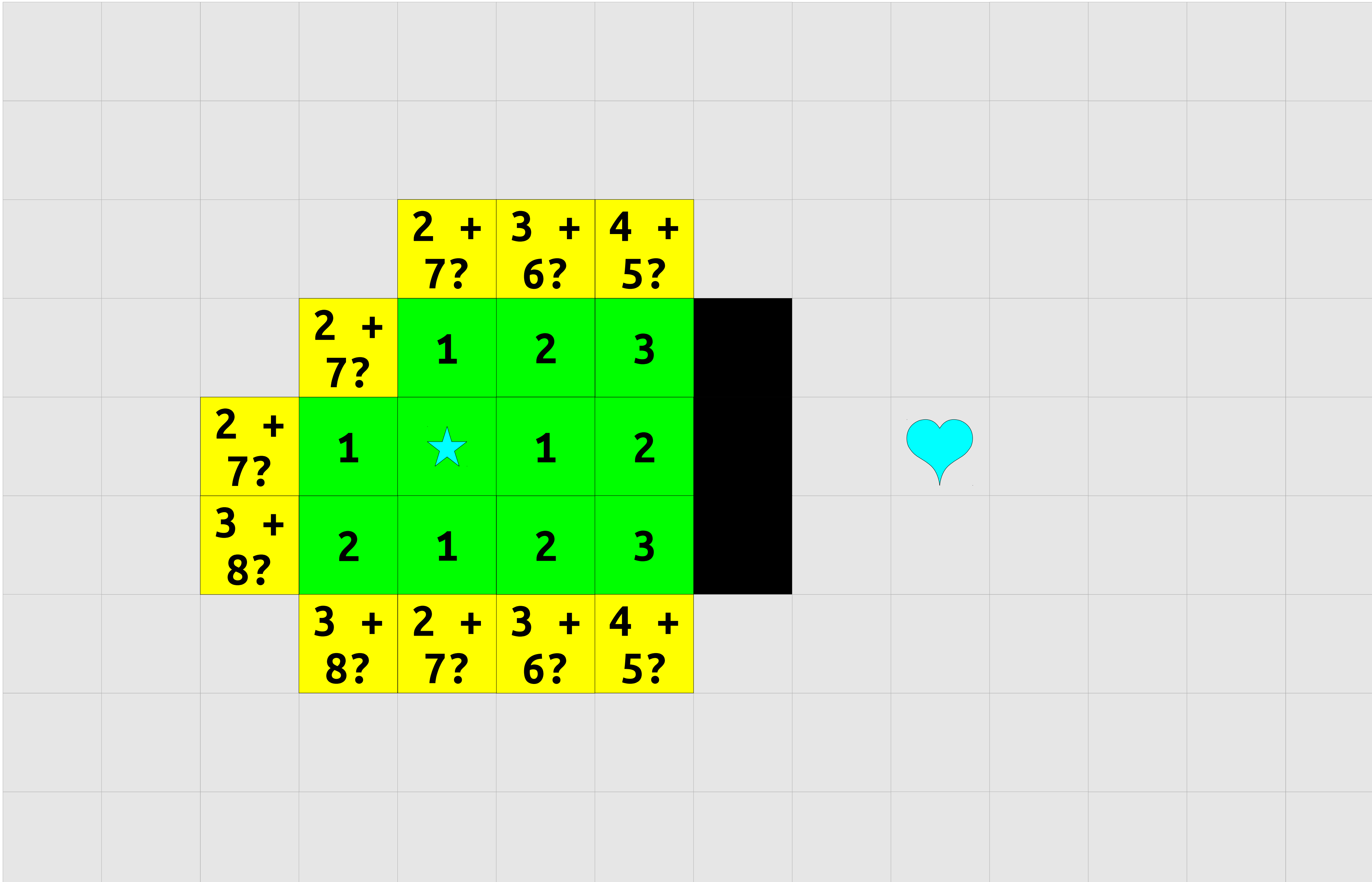

2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
2 + 7?
1
2
3 + 4?
2 + 7?
3 + 6?

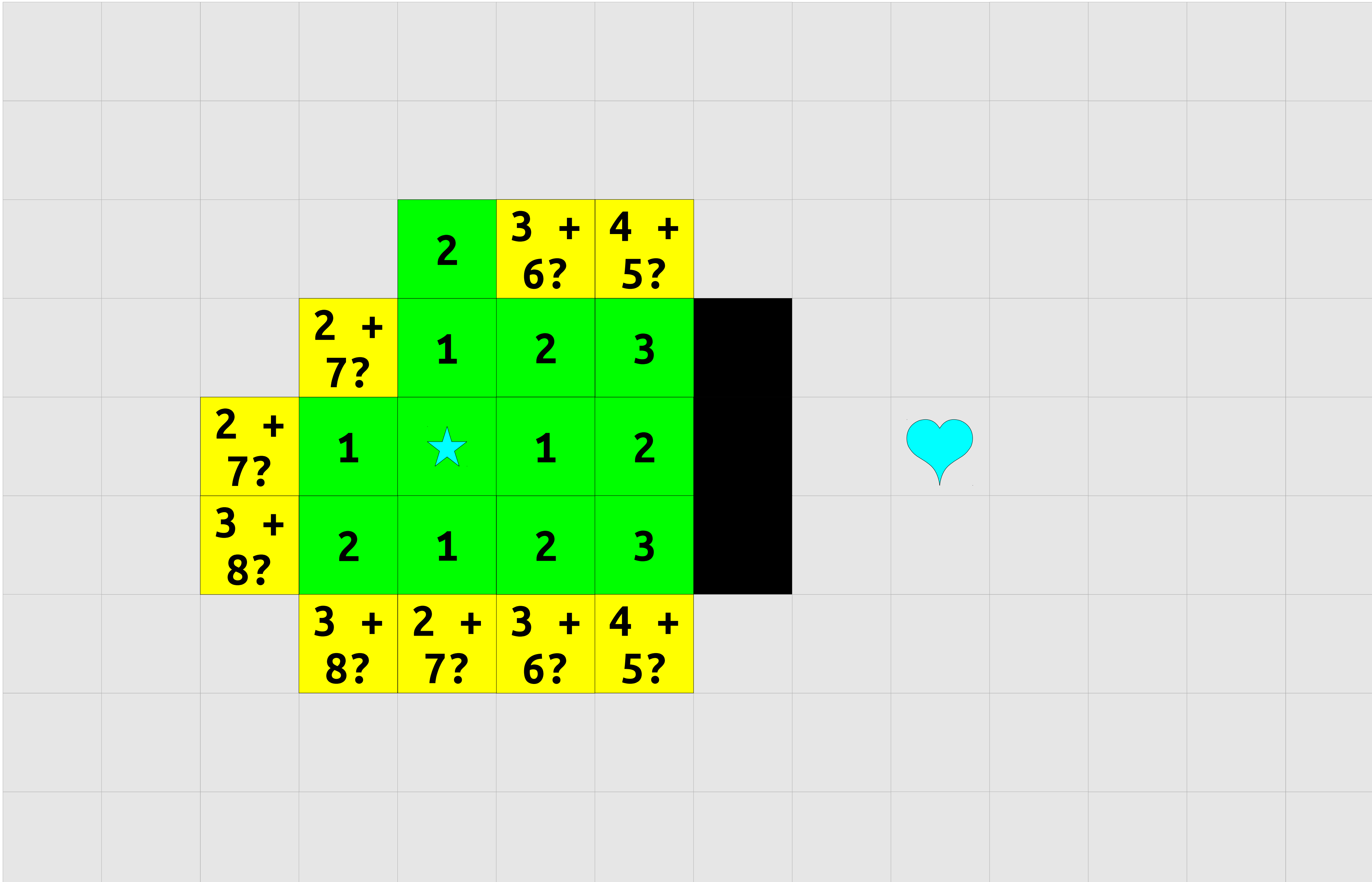

2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
2 + 7?
1
2
3
2 + 7?
3 + 6?

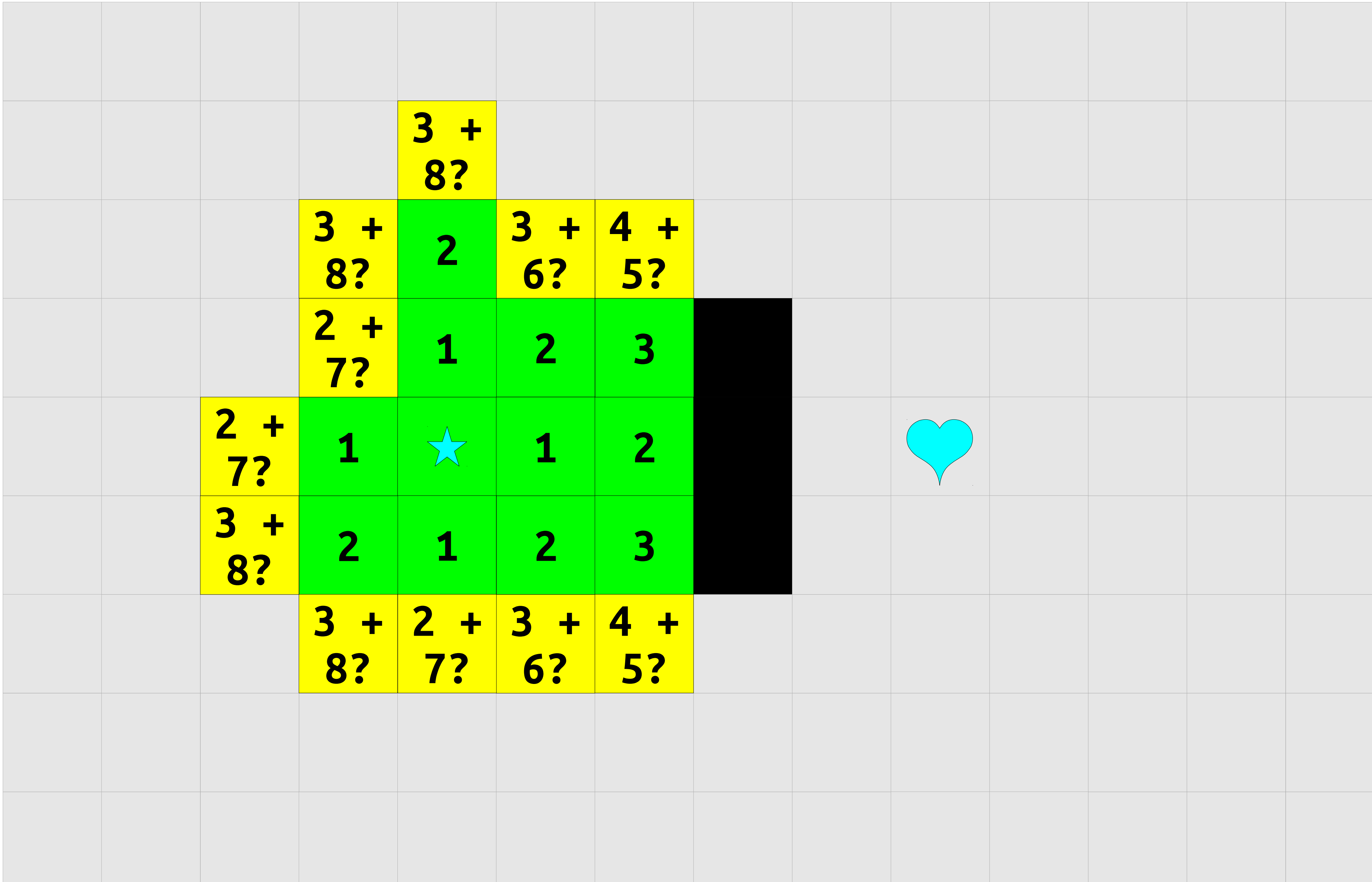

2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
2 + 7?
1
2
3
2 + 7?
3 + 6?
4 + 5?

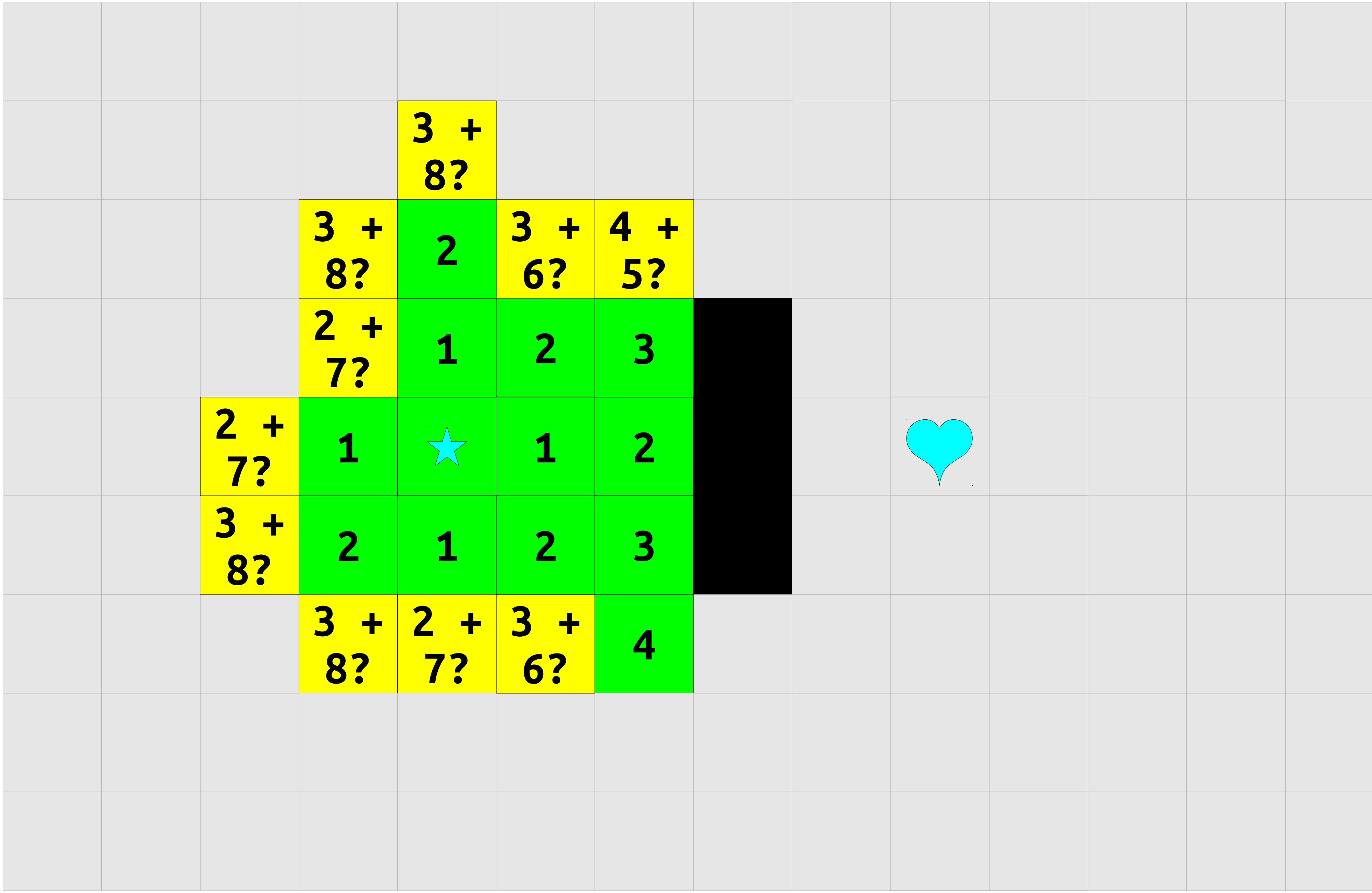


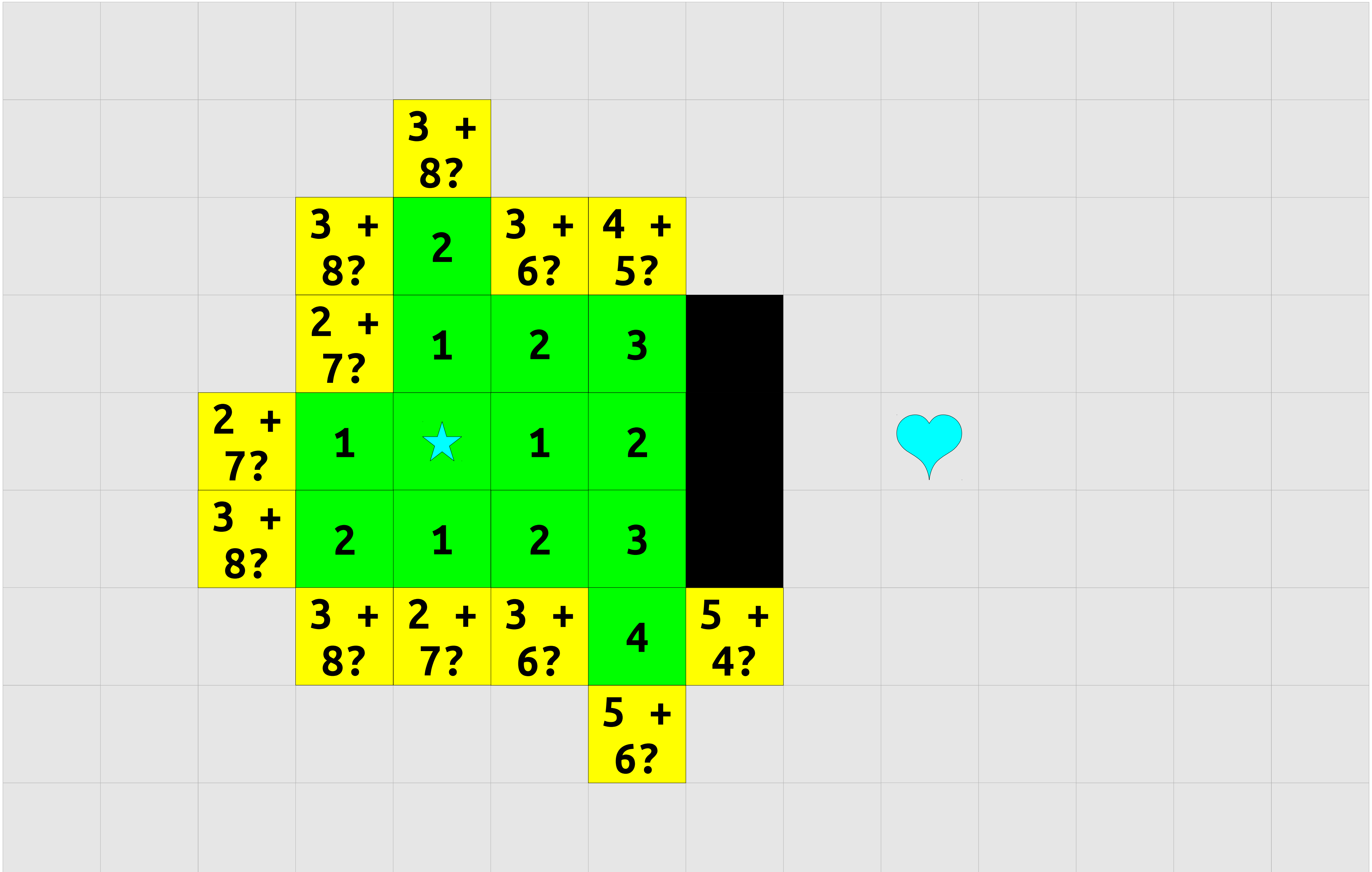
2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
2
1
2
3
2 + 7?
3 + 6?
4 + 5?

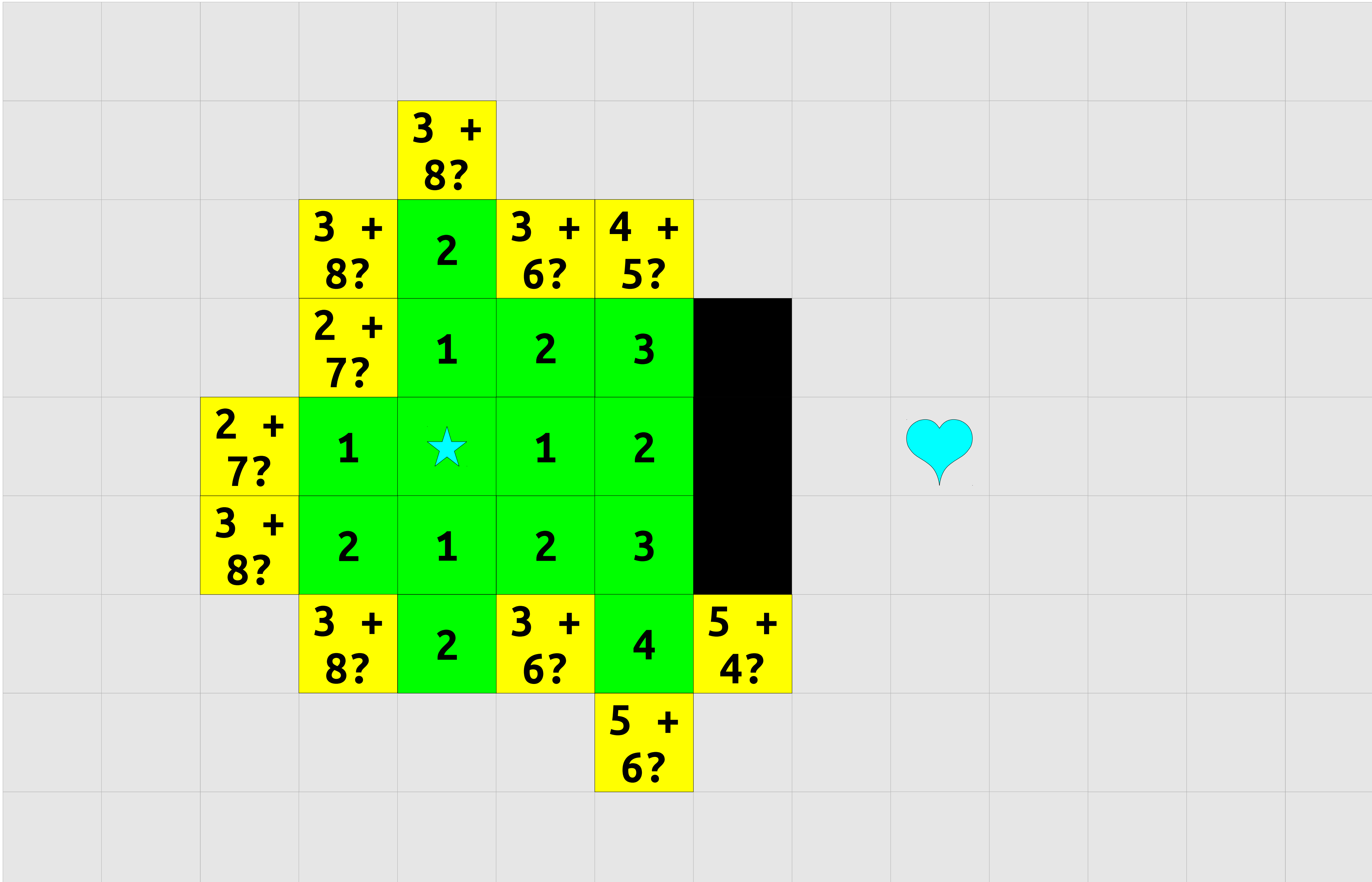

2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2 + 7?
3 + 6?
4 + 5?

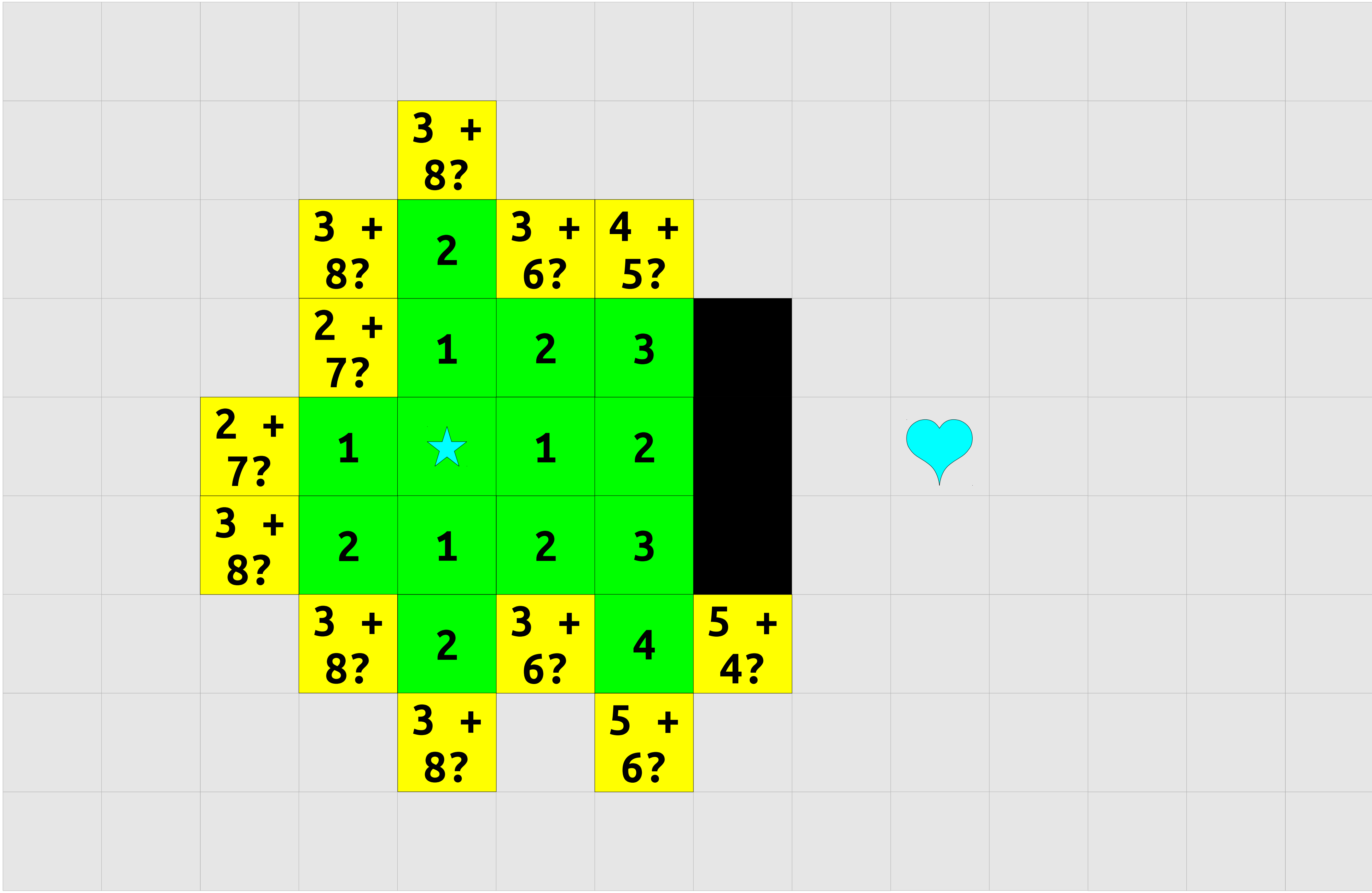

2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2 + 7?
3 + 6?
4 + 5?

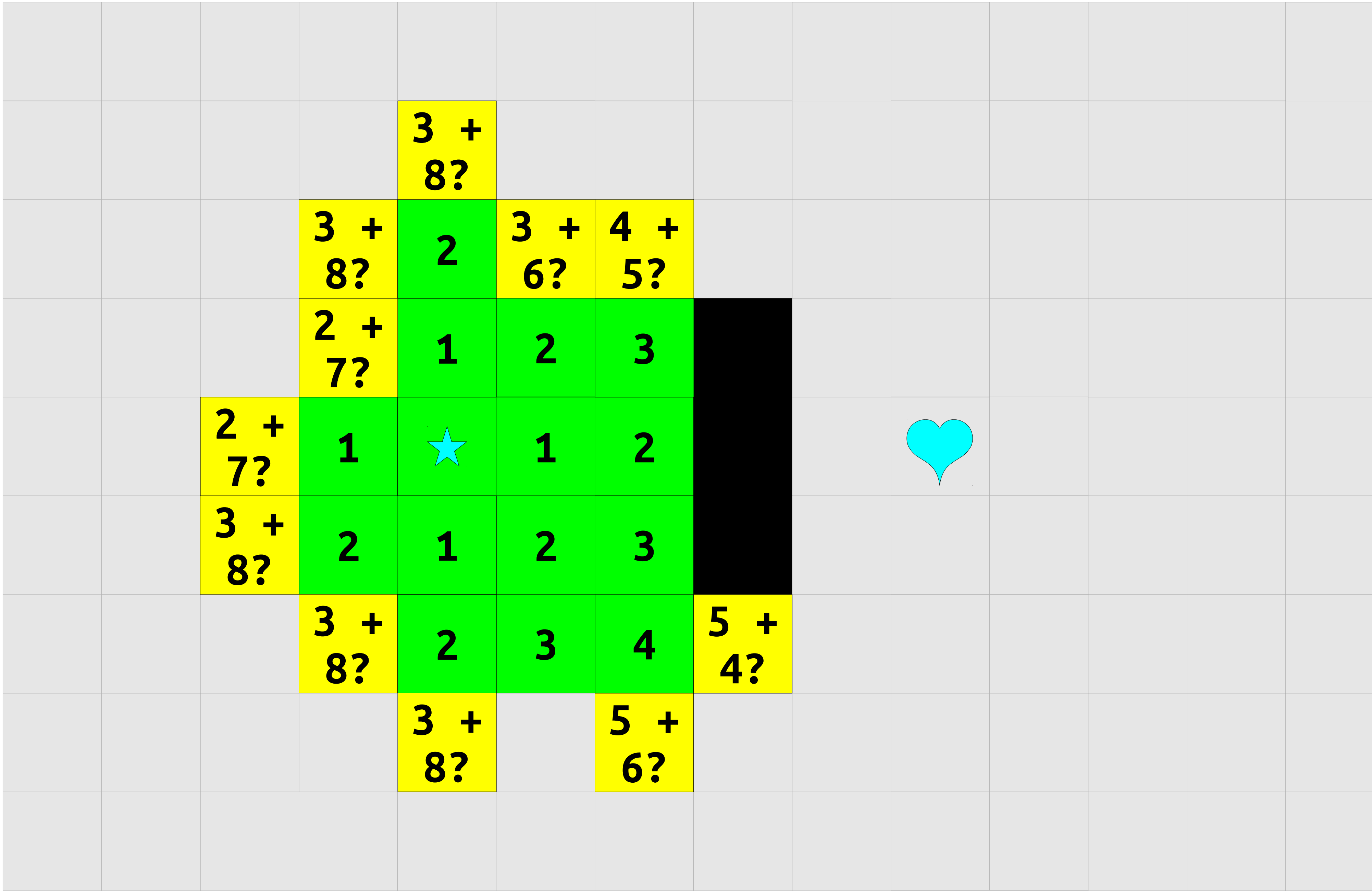

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2 + 7?
3 + 6?
4 + 5?


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

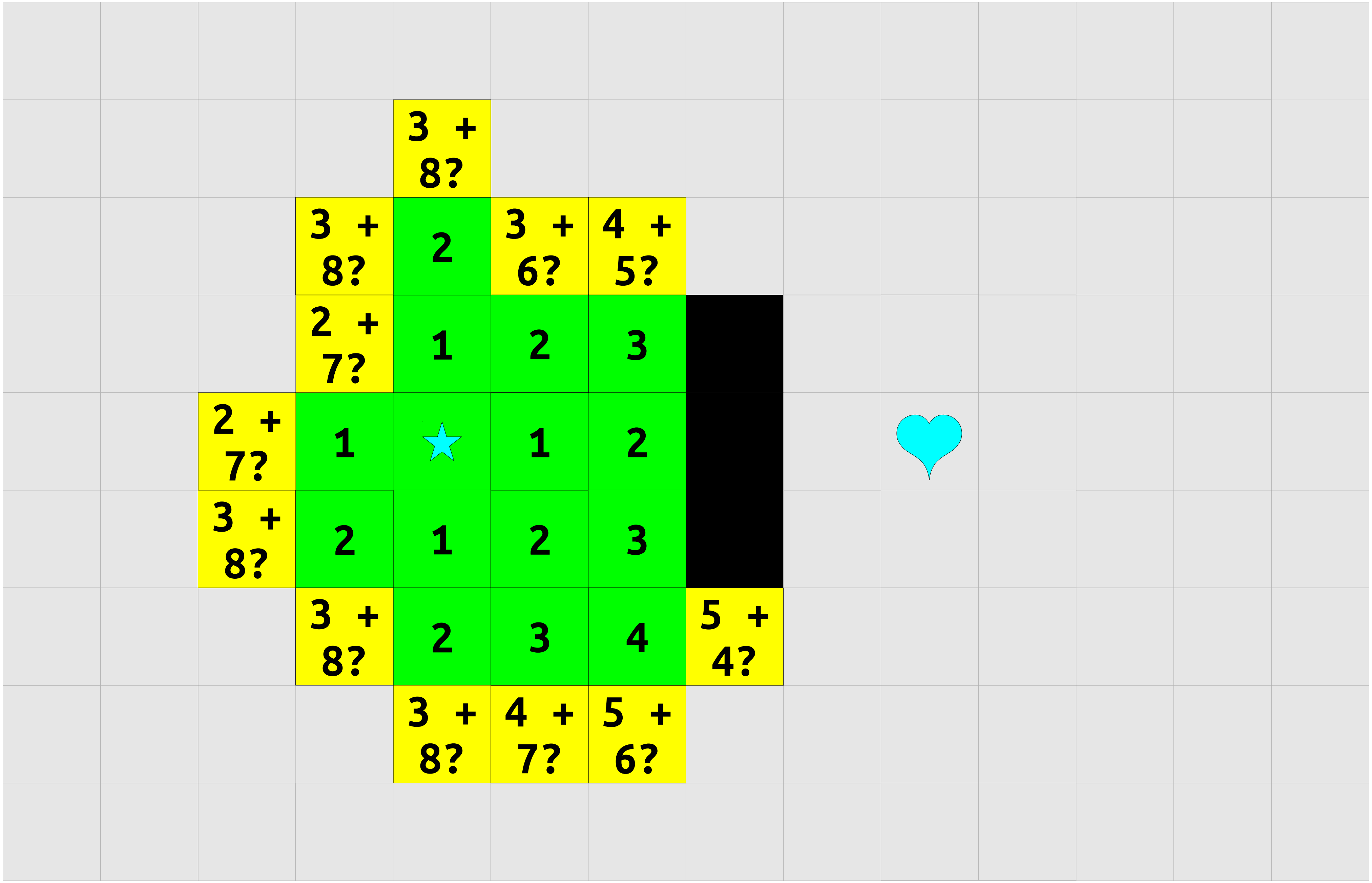

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2 + 7?
3 + 6?
4
5 + 4?
5 + 6?

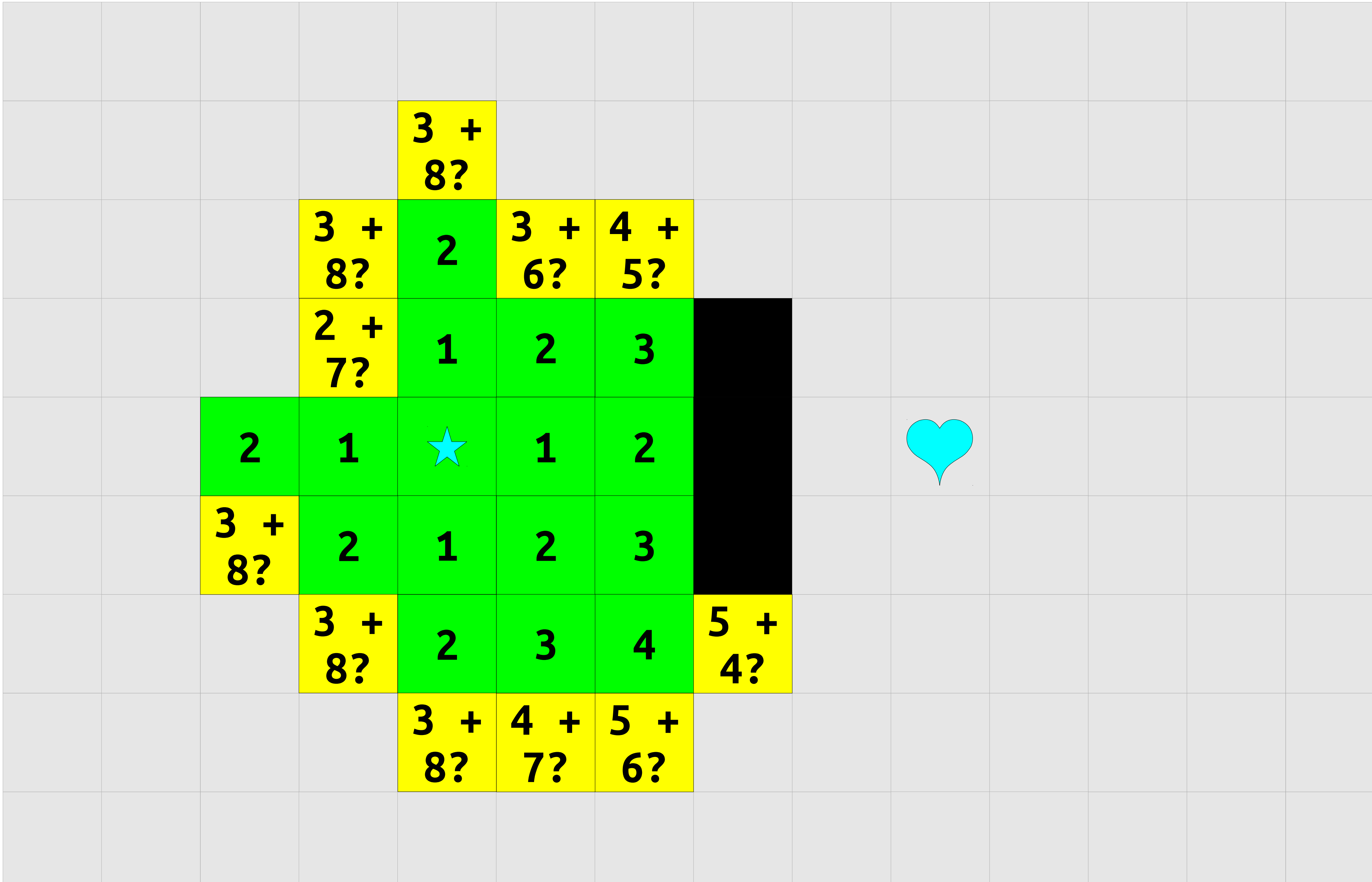

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3 + 6?
4
5 + 4?
5 + 6?

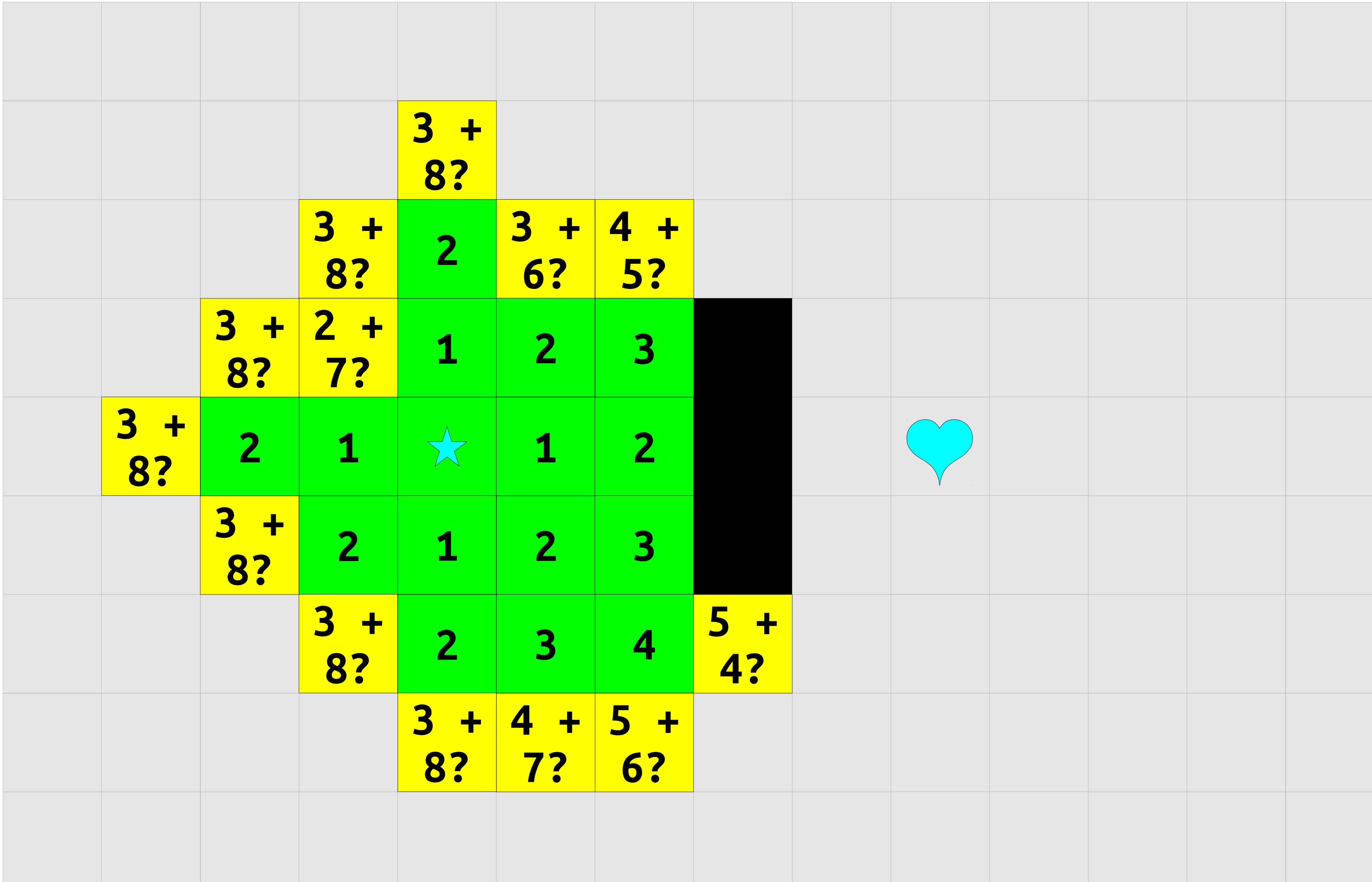

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3 + 6?
4
5 + 4?
3 + 8?
5 + 6?

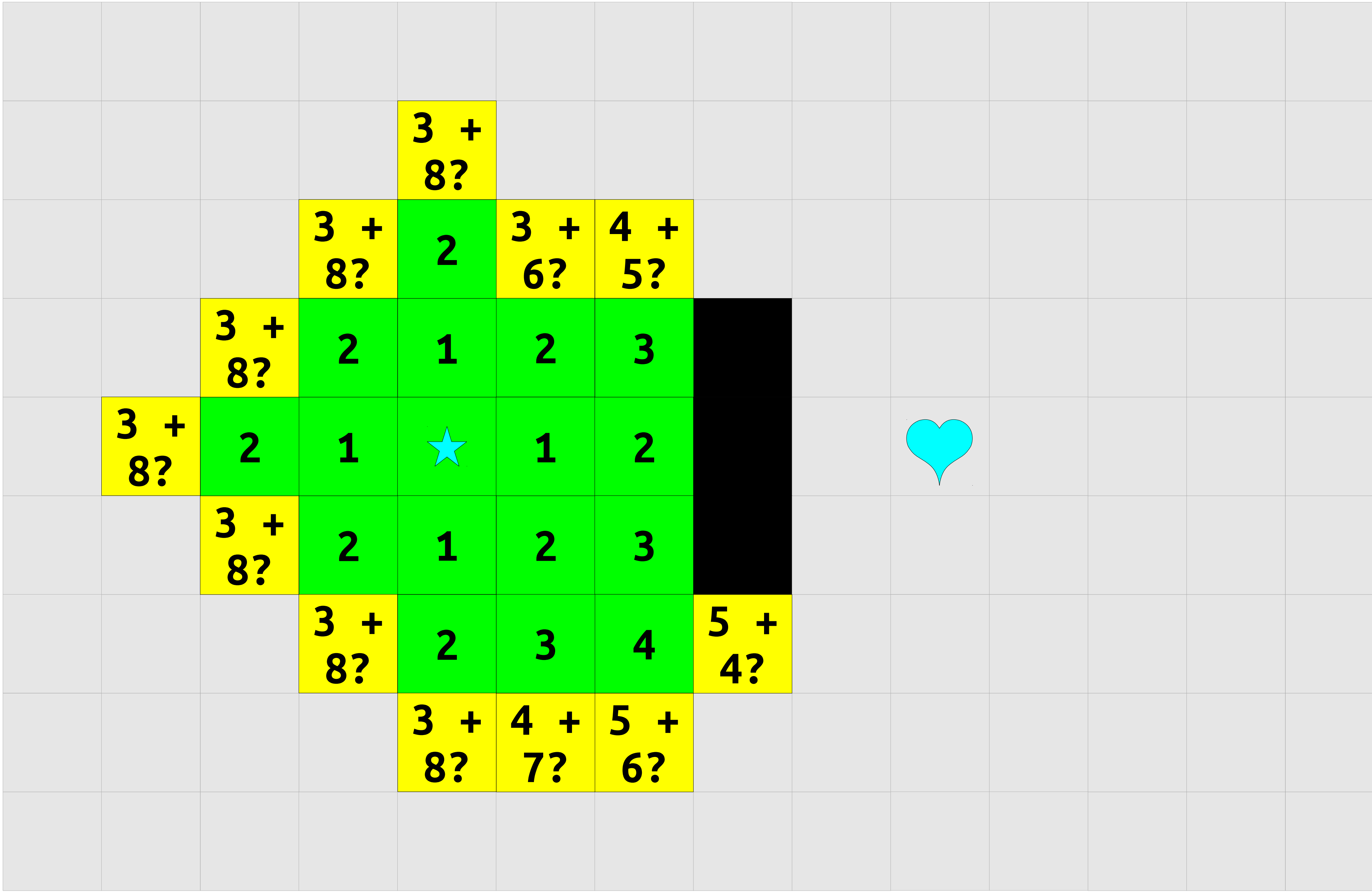


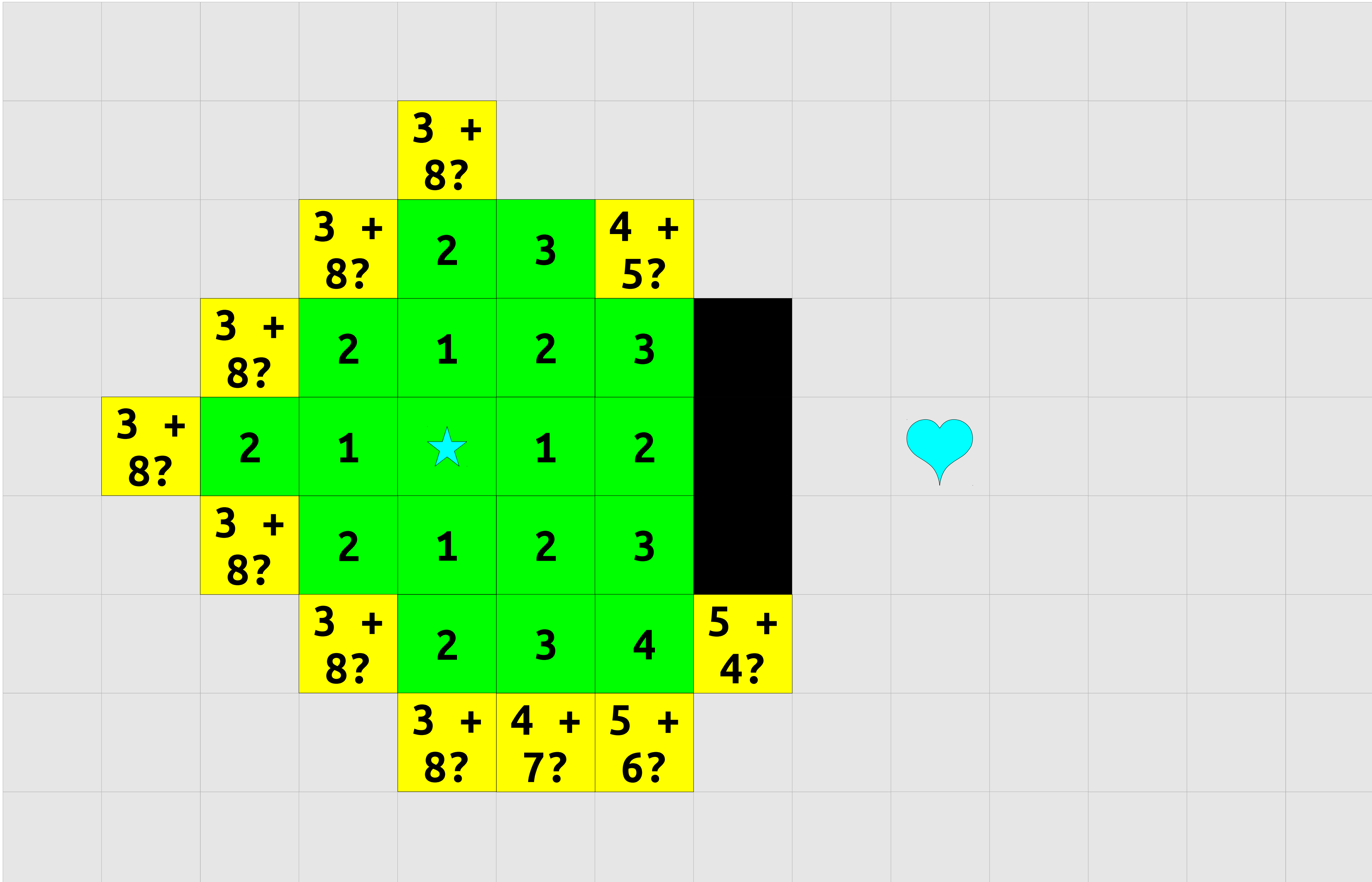
3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
5 + 6?

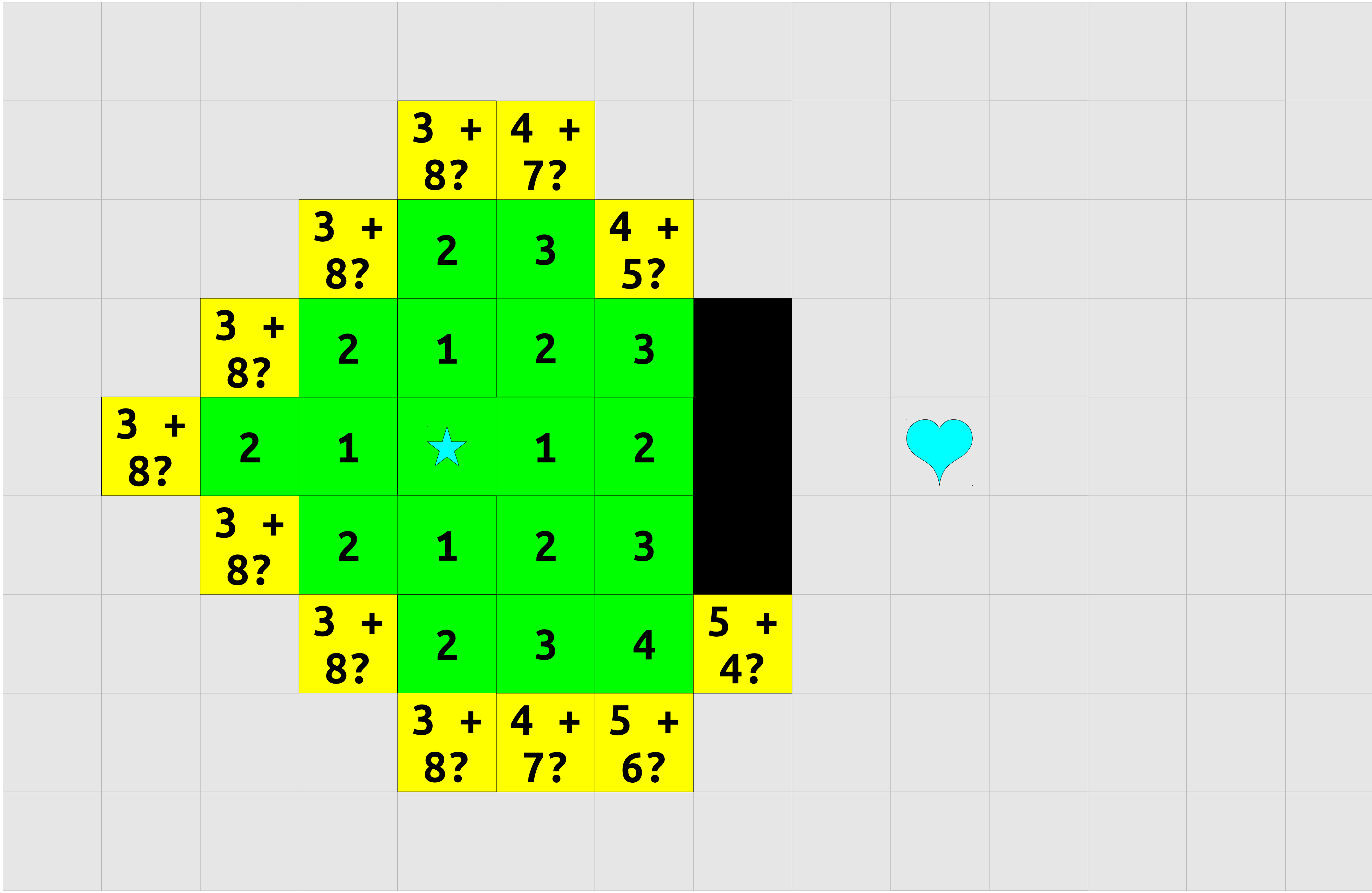

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

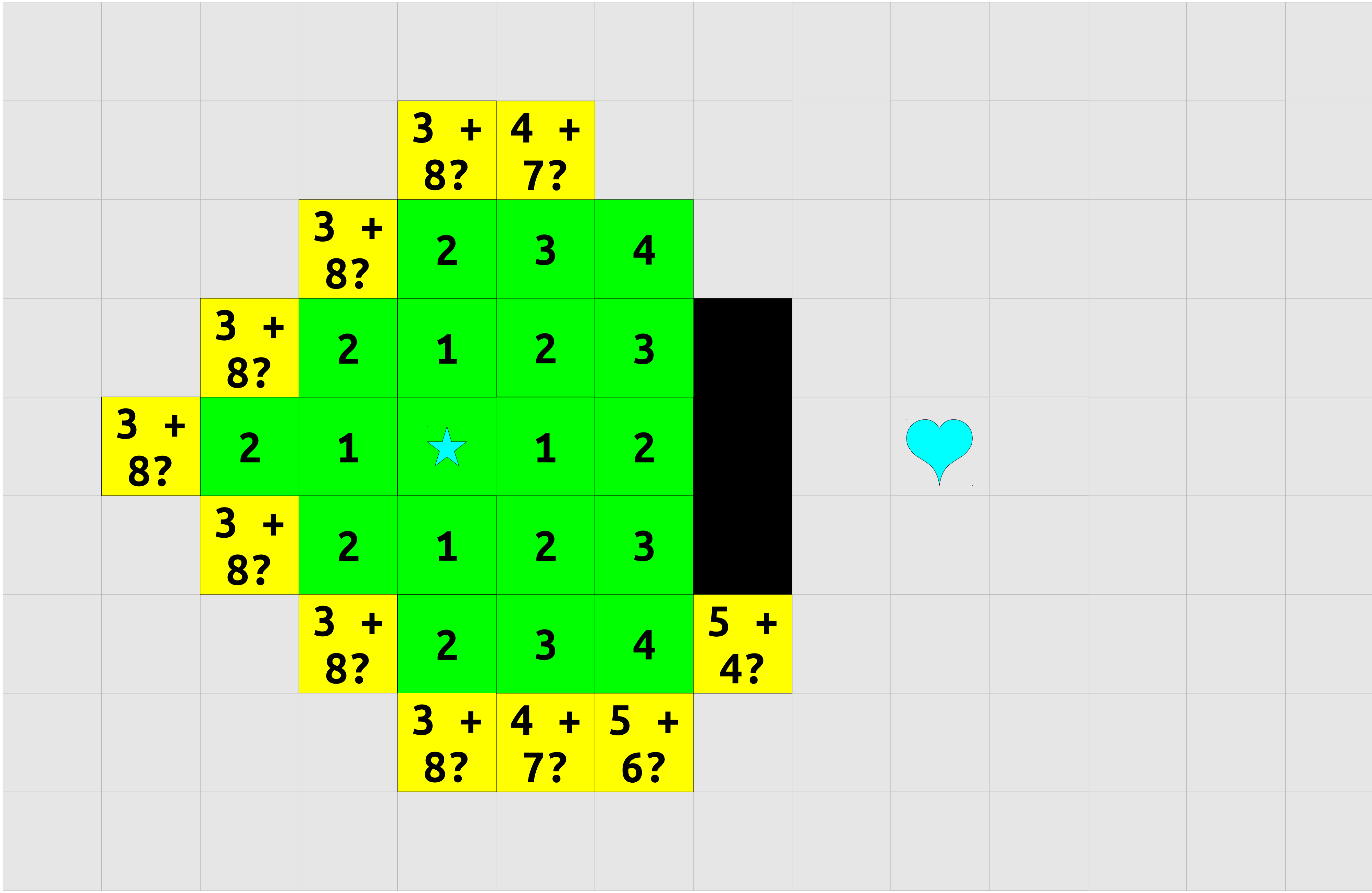

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?


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?

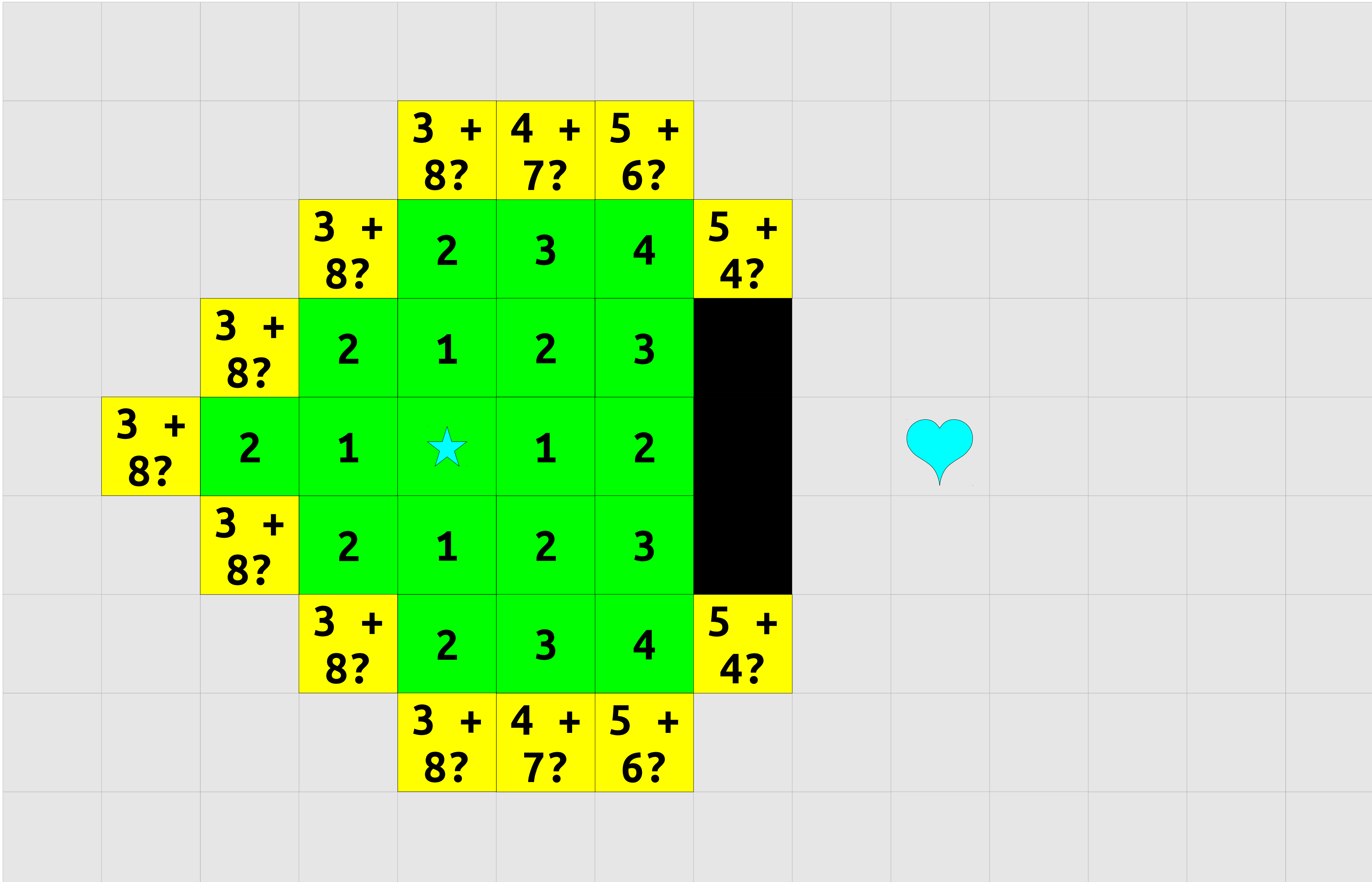

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

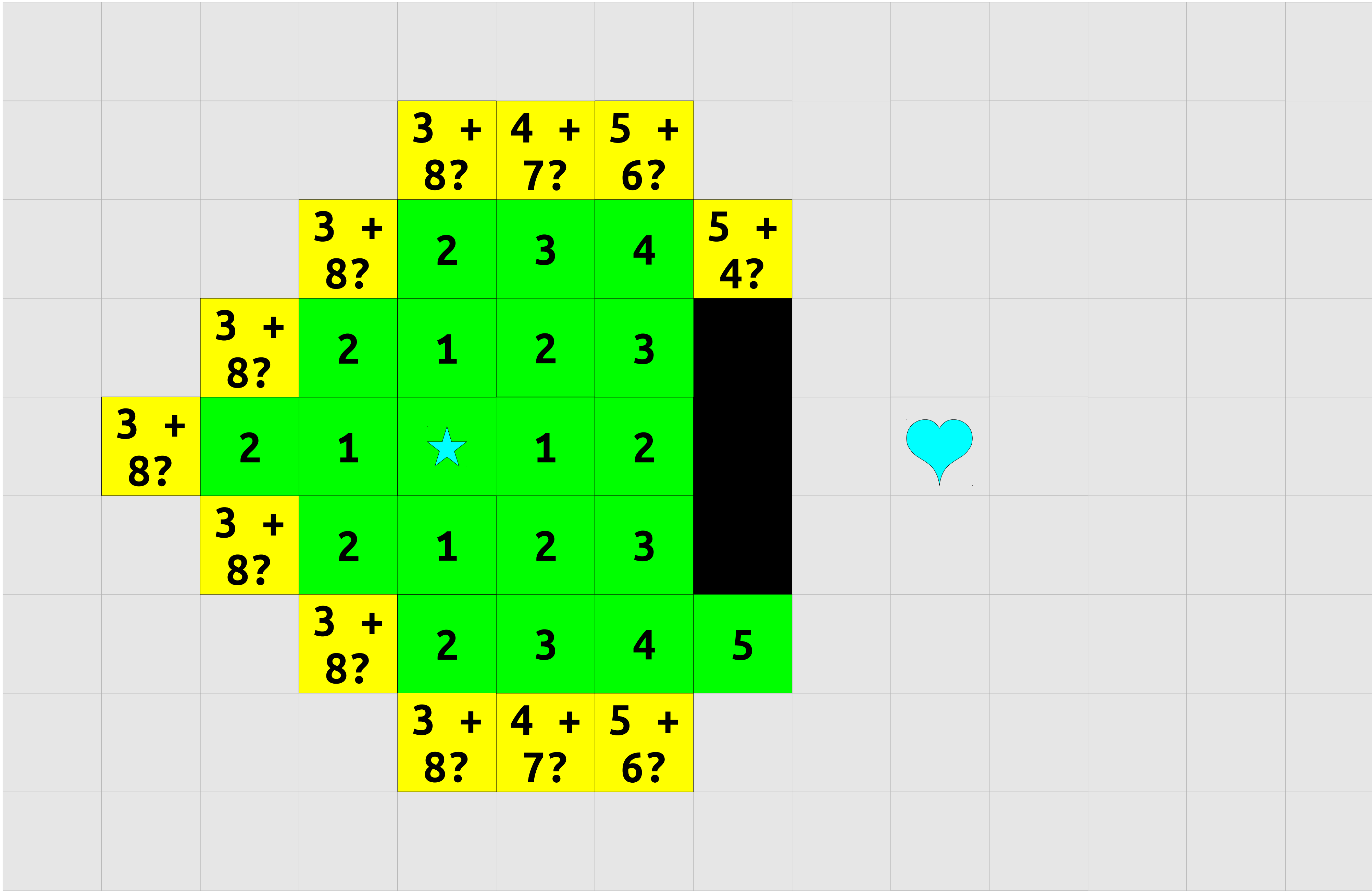

3 + 8?
3 + 8?
2
3
4 + 5?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

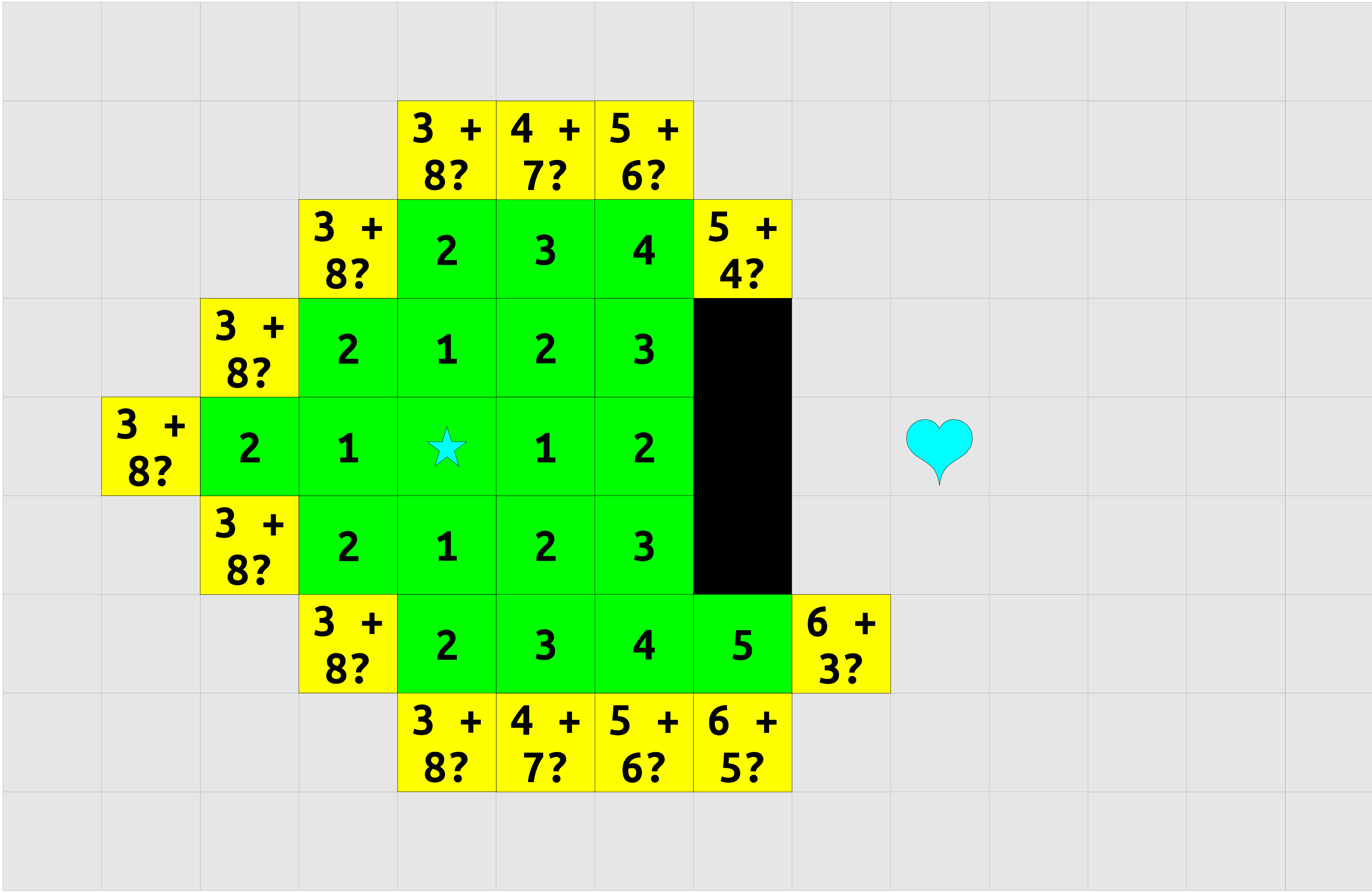

3 + 8?
4 + 7?
3 + 8?
2
3
4 + 5?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?



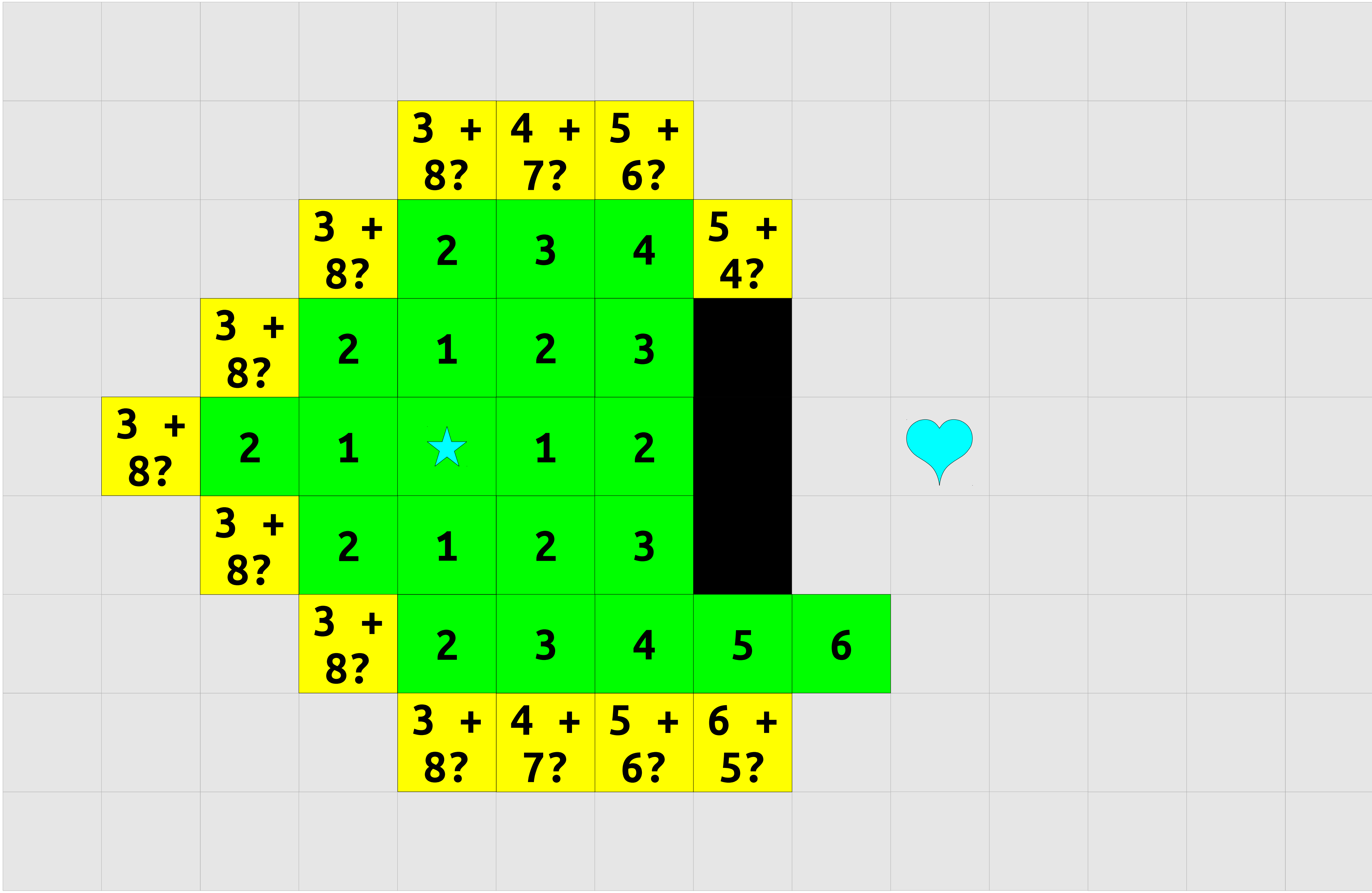
3 + 8?
4 + 7?
3 + 8?
2
3
4
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

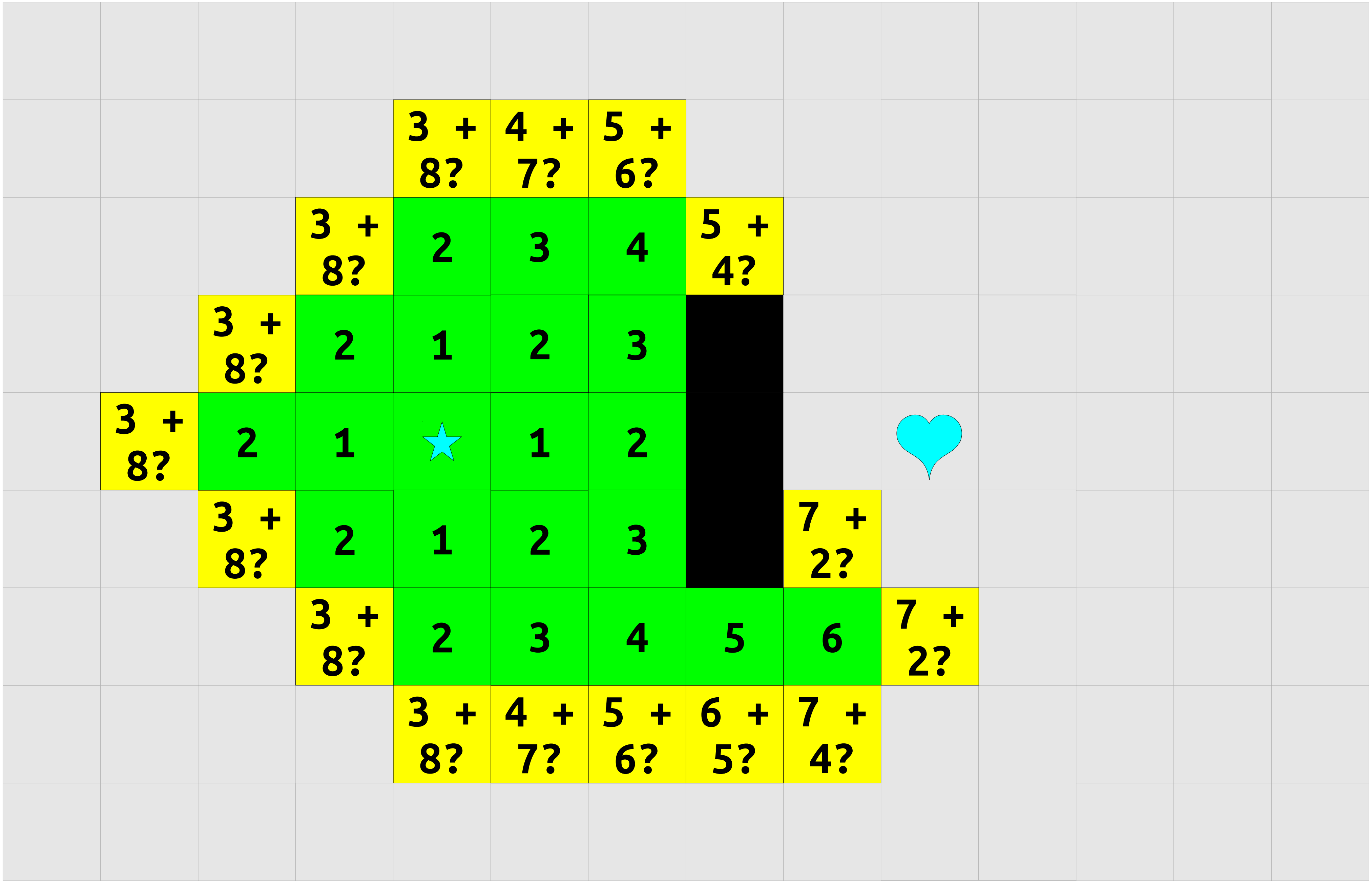

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5 + 4?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

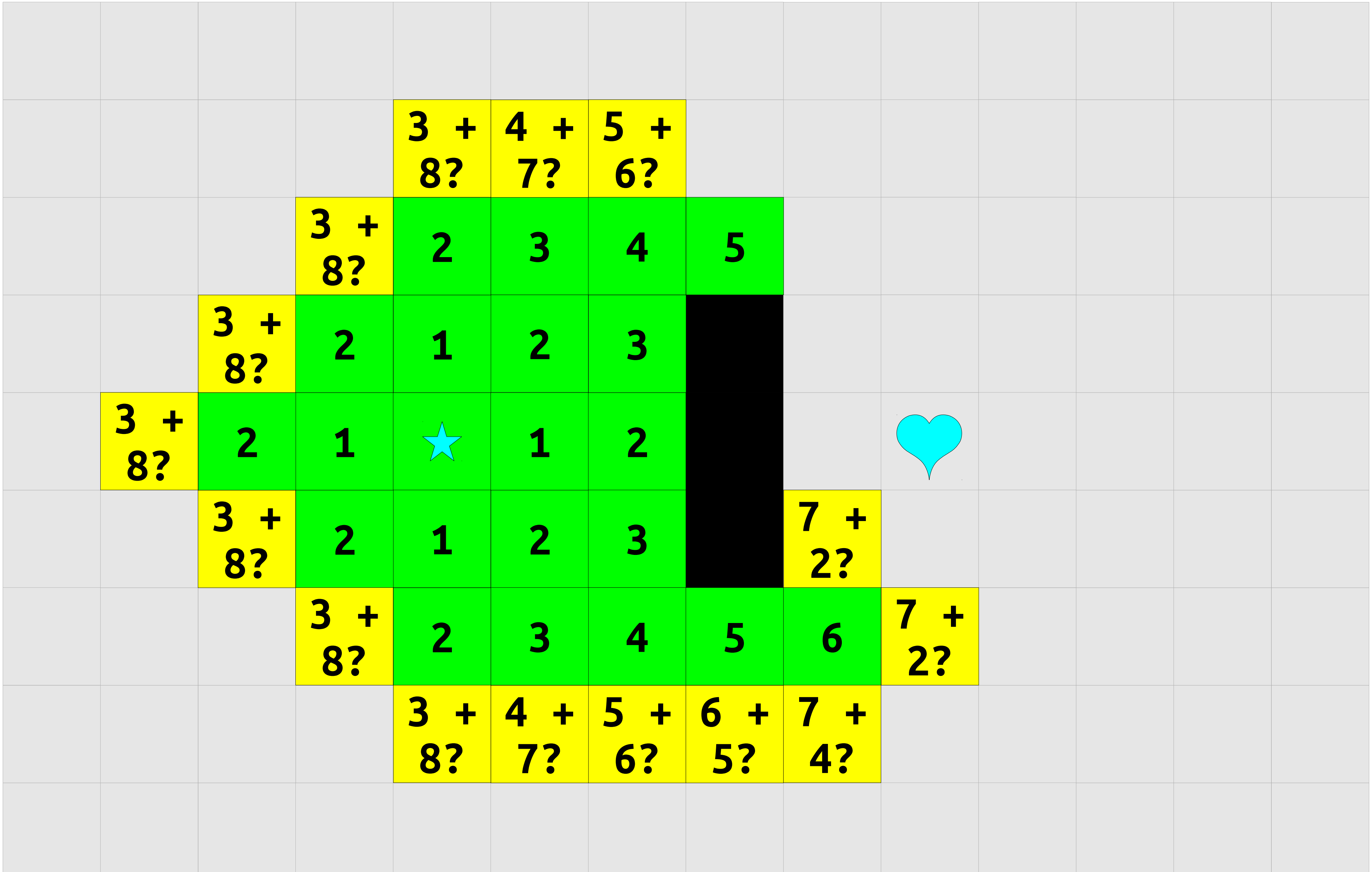

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5 + 4?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5
3 + 8?
4 + 7?
5 + 6?



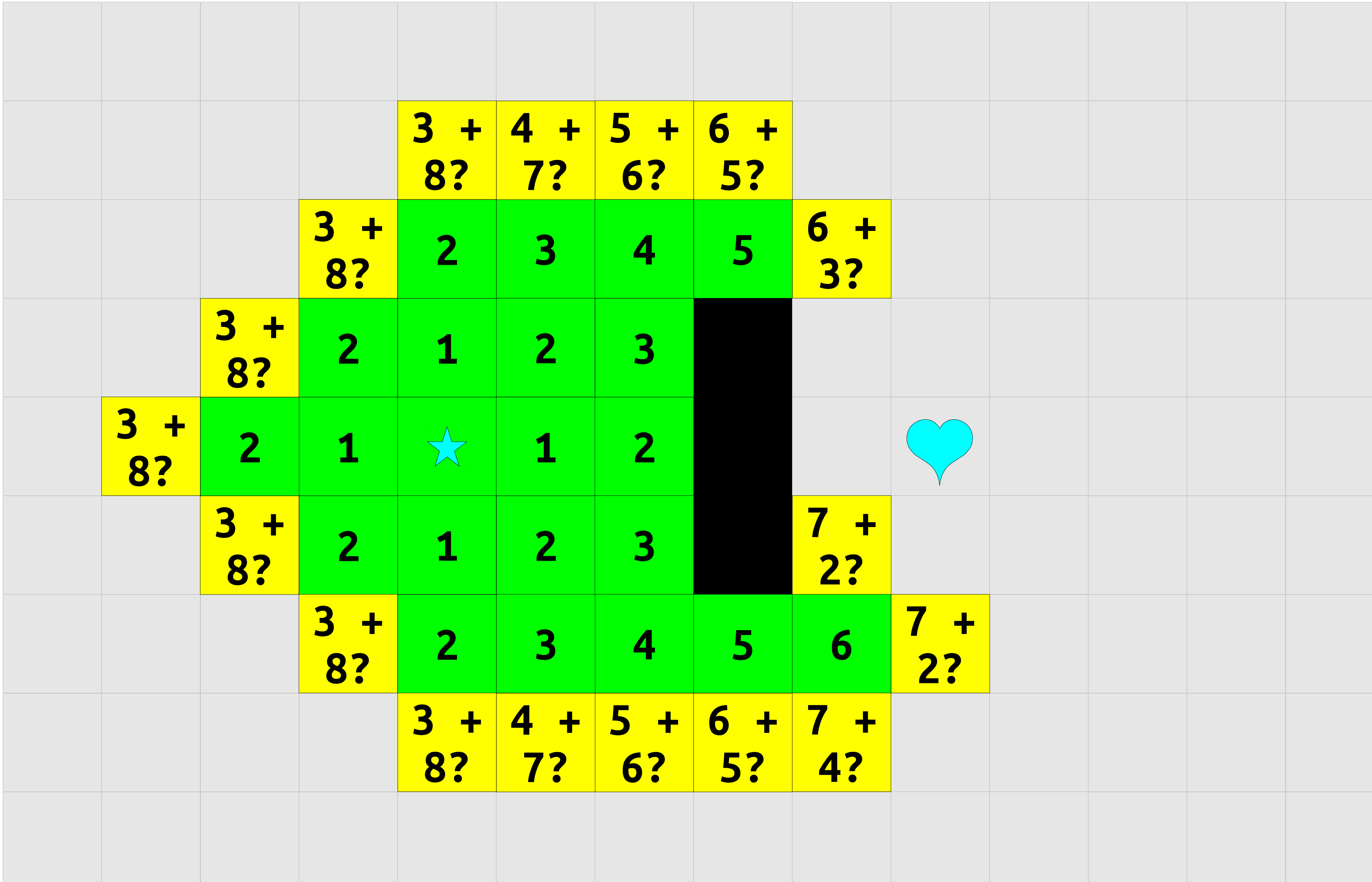
3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5 + 4?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5
6 + 3?
3 + 8?
4 + 7?
5 + 6?
6 + 5?


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?

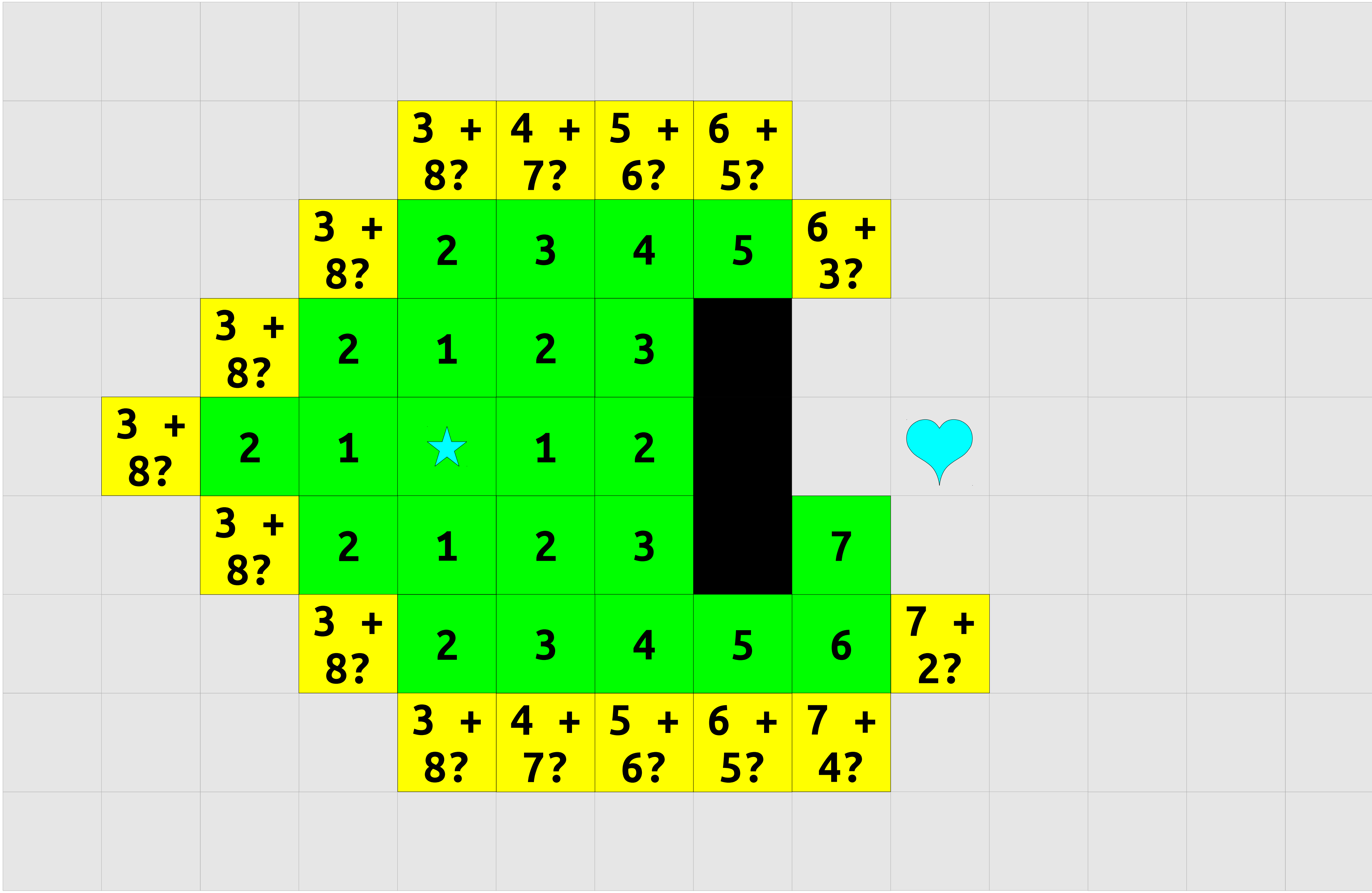

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5 + 4?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
7 + 2?
3 + 8?
2
3
4
5
6
7 + 2?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?



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?



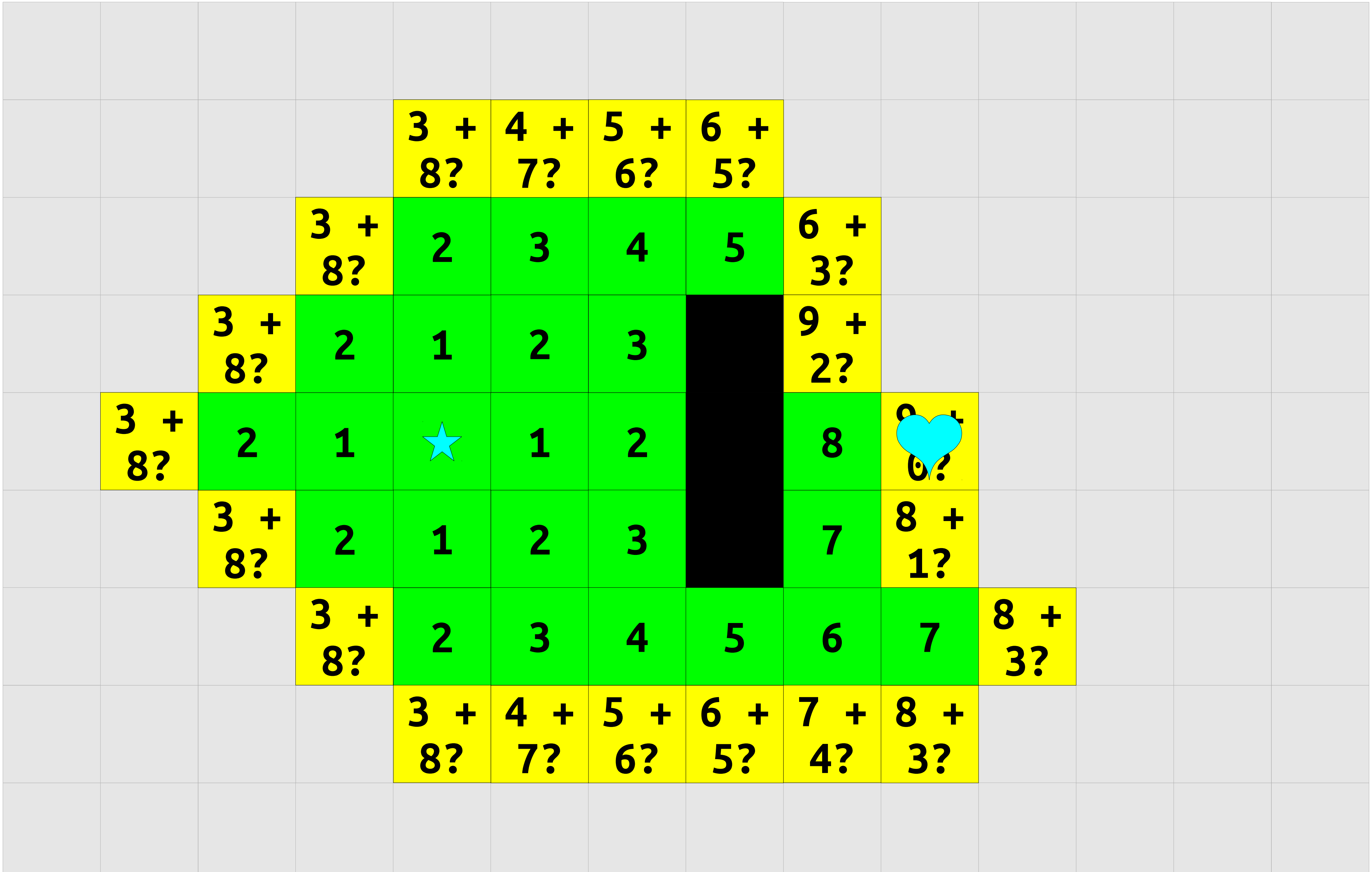
3 + 8?
4 + 7?
5 + 6?
6 + 5?
3 + 8?
2
3
4
5
6 + 3?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
7
3 + 8?
2
3
4
5
6
7 + 2?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?

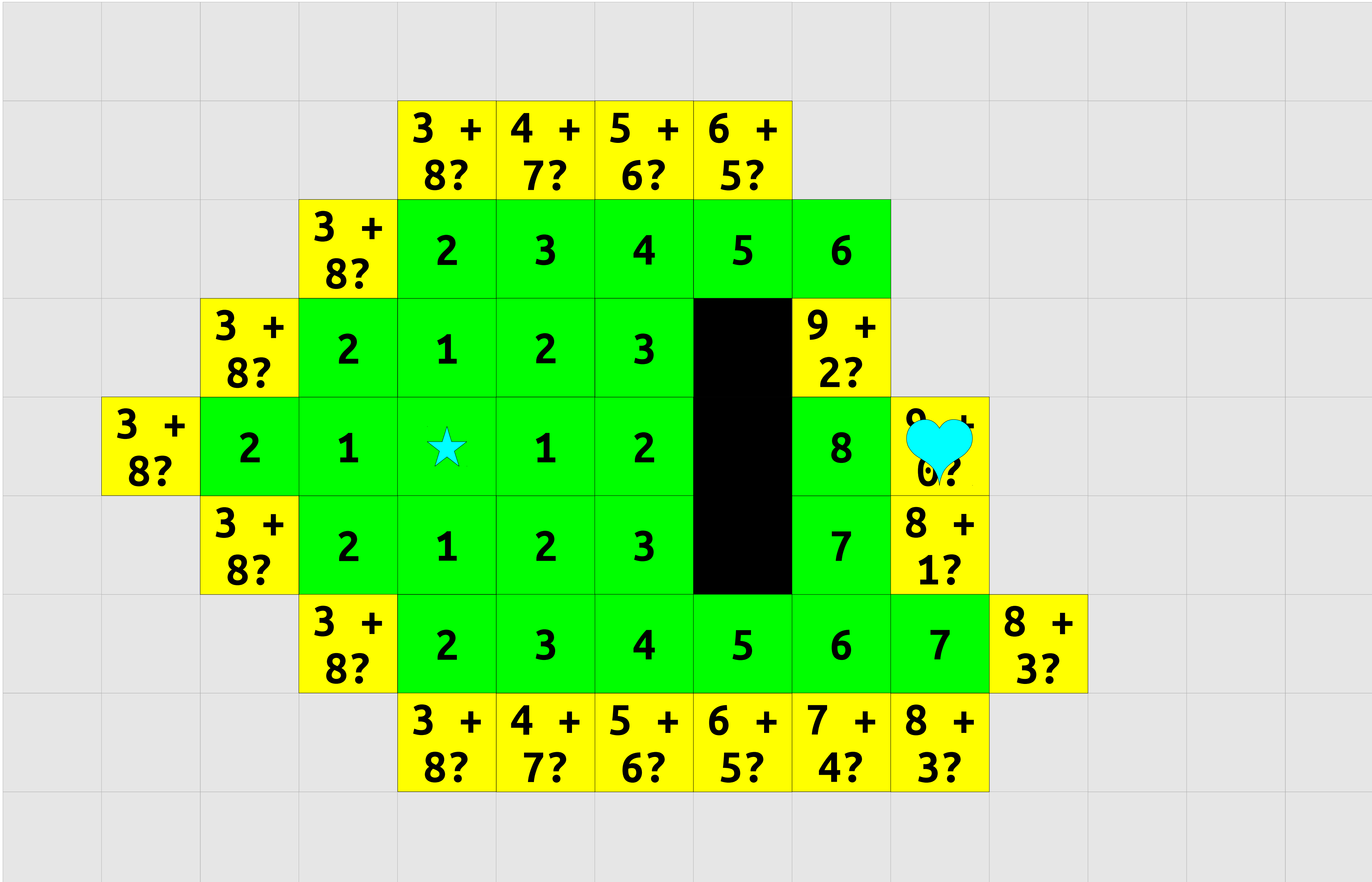
				3 + 8?	4 + 7?	5 + 6?	6 + 5?						
			3 + 8?	2	3	4	5	6 + 3?					
		3 + 8?	2	1	2	3							
	3 + 8?	2	1		1	2		8 + 1?					
		3 + 8?	2	1	2	3		7	8 + 1?				
			3 + 8?	2	3	4	5	6	7 + 2?				
				3 + 8?	4 + 7?	5 + 6?	6 + 5?	7 + 4?					

				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
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
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?				

A* Search

- The approach described here (using not just the estimated distance to each node, but also the ***heuristic distance*** to the target) is called ***A* search***.
- Provided you have an admissible heuristic, A* can be dramatically faster than Dijkstra’s algorithm.
- Oh, and the code is a *trivial* modification on Dijkstra’s algorithm…

Dijkstra’s Algorithm

```
dijkstra's-algorithm() {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    if (that node isn't green) {
      color that node green.

      for (each neighboring node) {
        if (that node is not green) {
          color the node yellow.
          enqueue it at the new distance.
        }
      }
    }
  }
}
```

A* Search

```
a*-search() {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    if (that node isn't green) {
      color that node green.

      for (each neighboring node) {
        if (that node is not green) {
          color the node yellow.
          enqueue it at the new distance plus the heuristic.
        }
      }
    }
  }
}
```

Questions to Ponder

- Why must the heuristic never overestimate the distance to the target?
  - ***Hint:*** Think about the reason why Dijkstra's algorithm is correct in the first place.
- A heuristic of zero is always admissible, since it never overestimates distances. What do you get if you run A* search with a zero heuristic function?

Next Time

- Minimum Spanning Trees
- Data Clustering
- Applications to Computational Biology