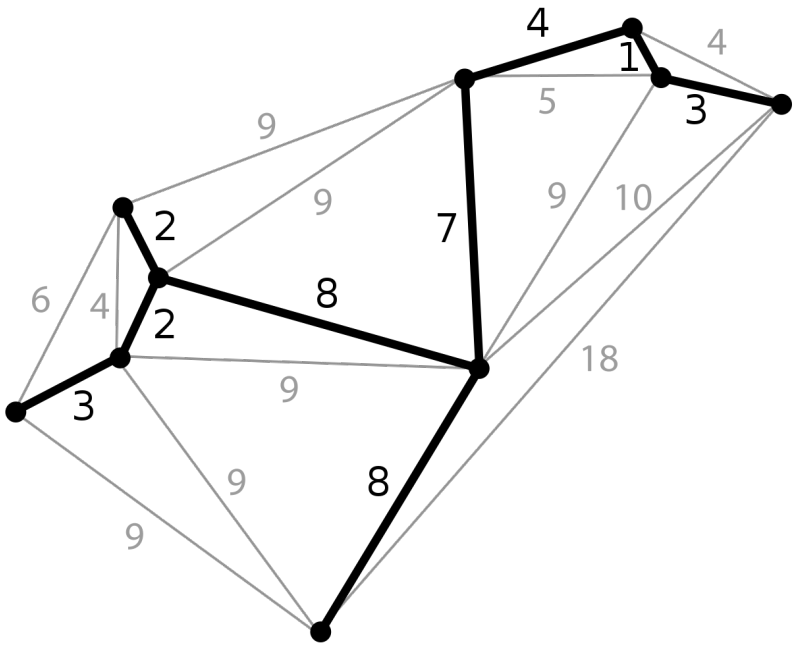
Data Structures and Algorithms (DSA) for AI

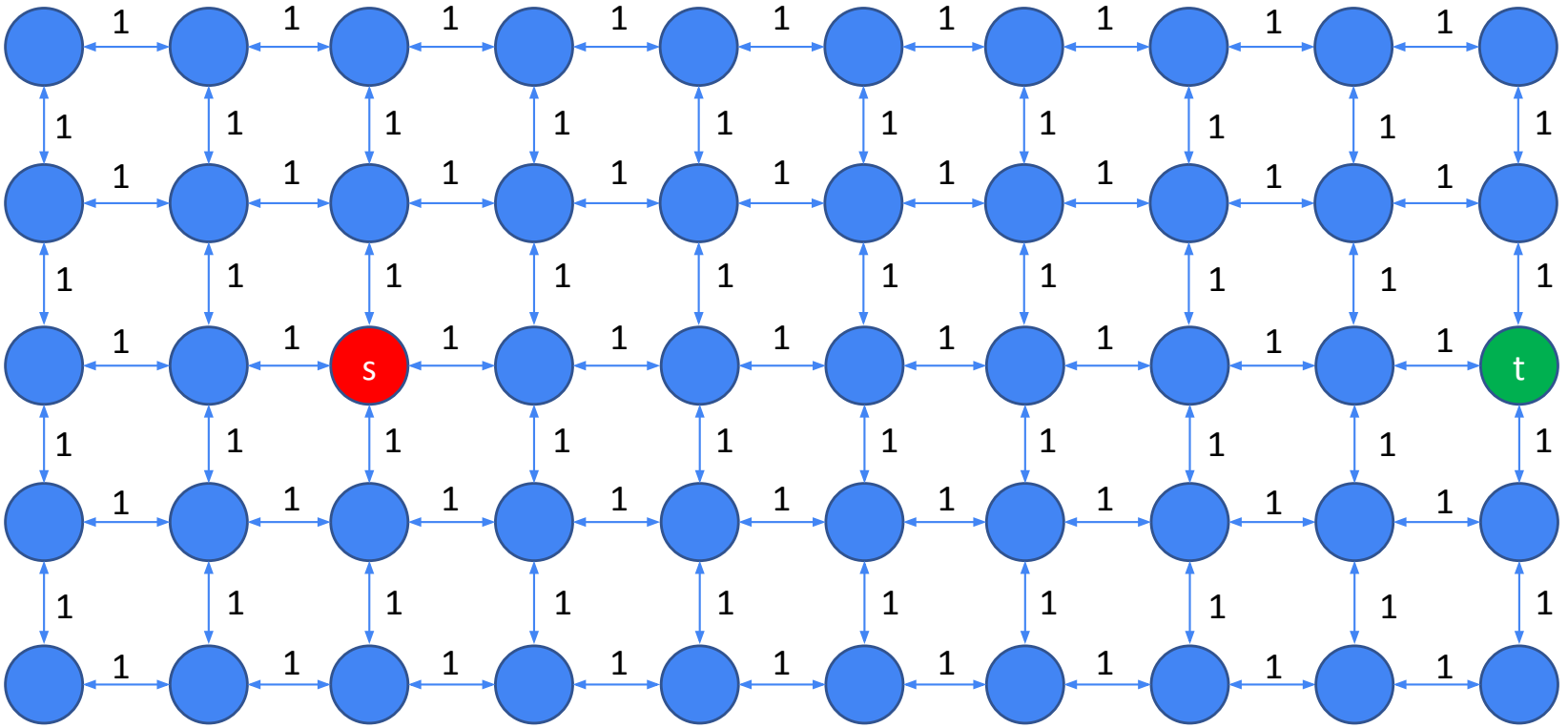
Katja Tuma



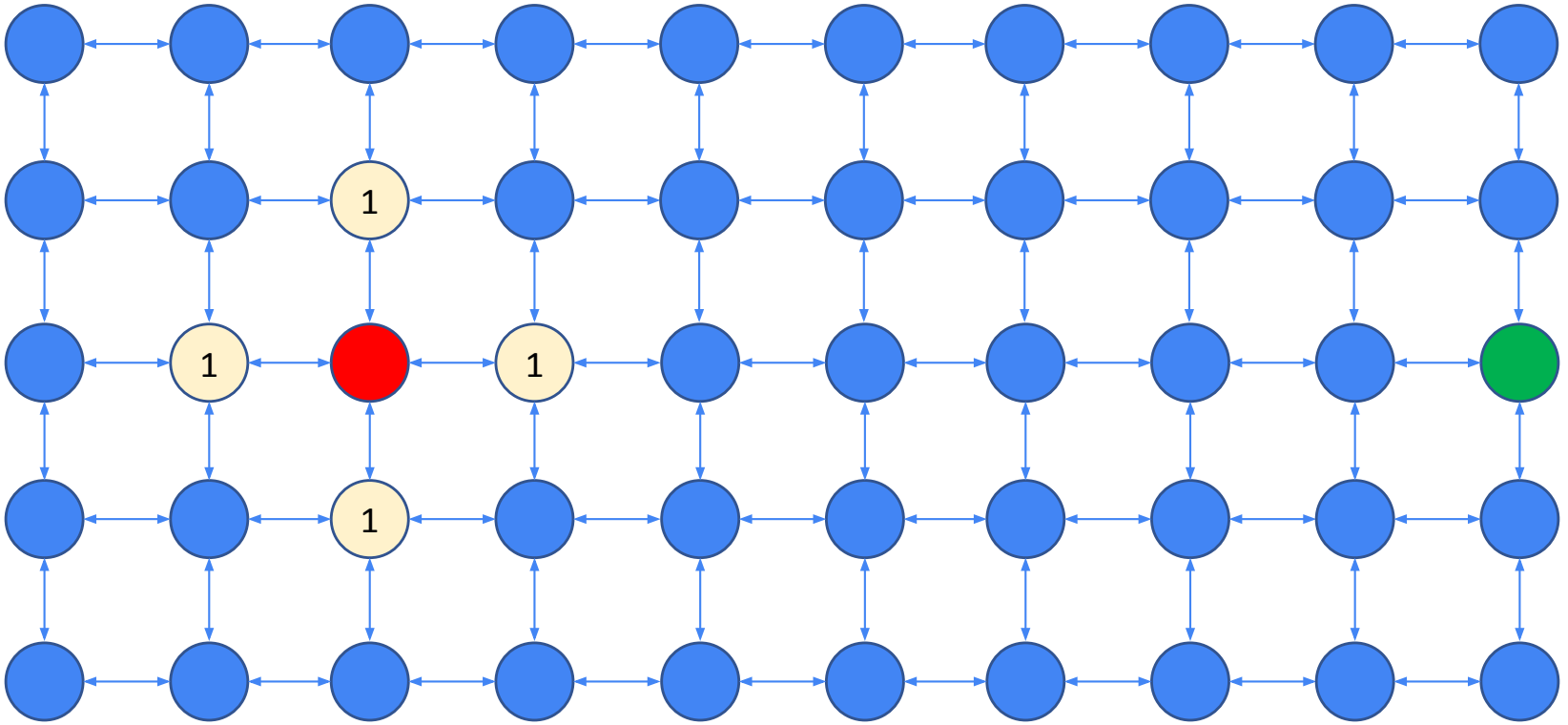
Dijkstra (recap)

Finds the shortest path from a vertex to every other vertex in the graph.

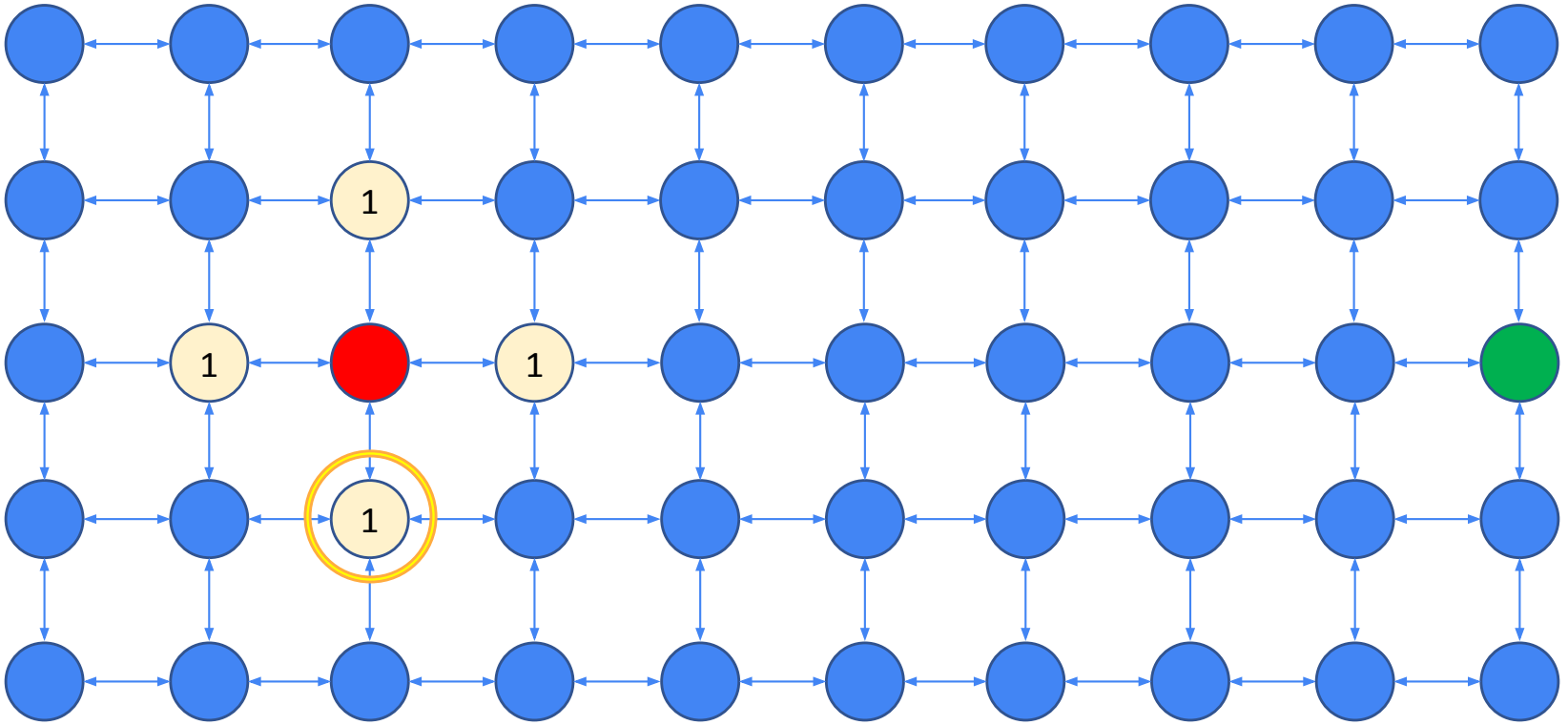
Bad graph example for Dijkstra



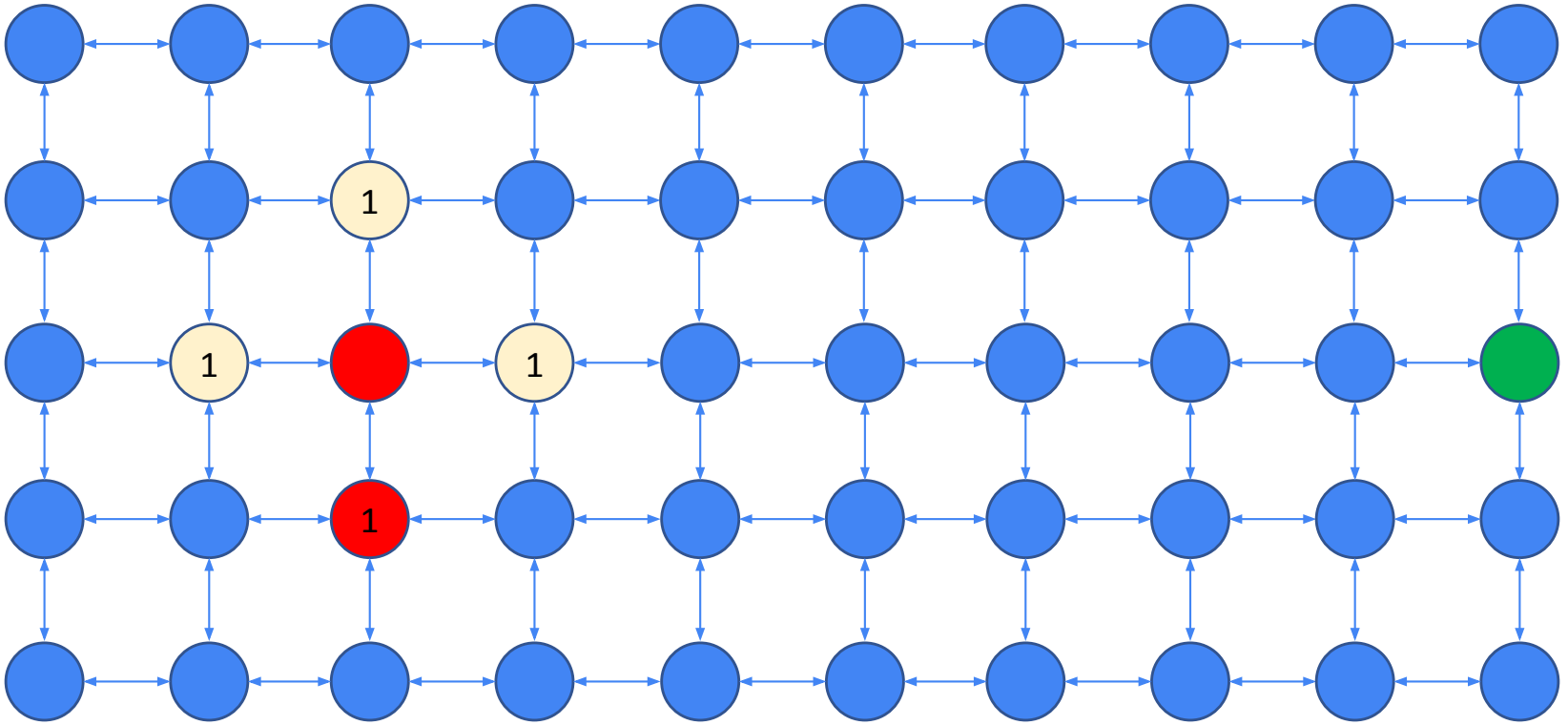
Bad graph example for Dijkstra



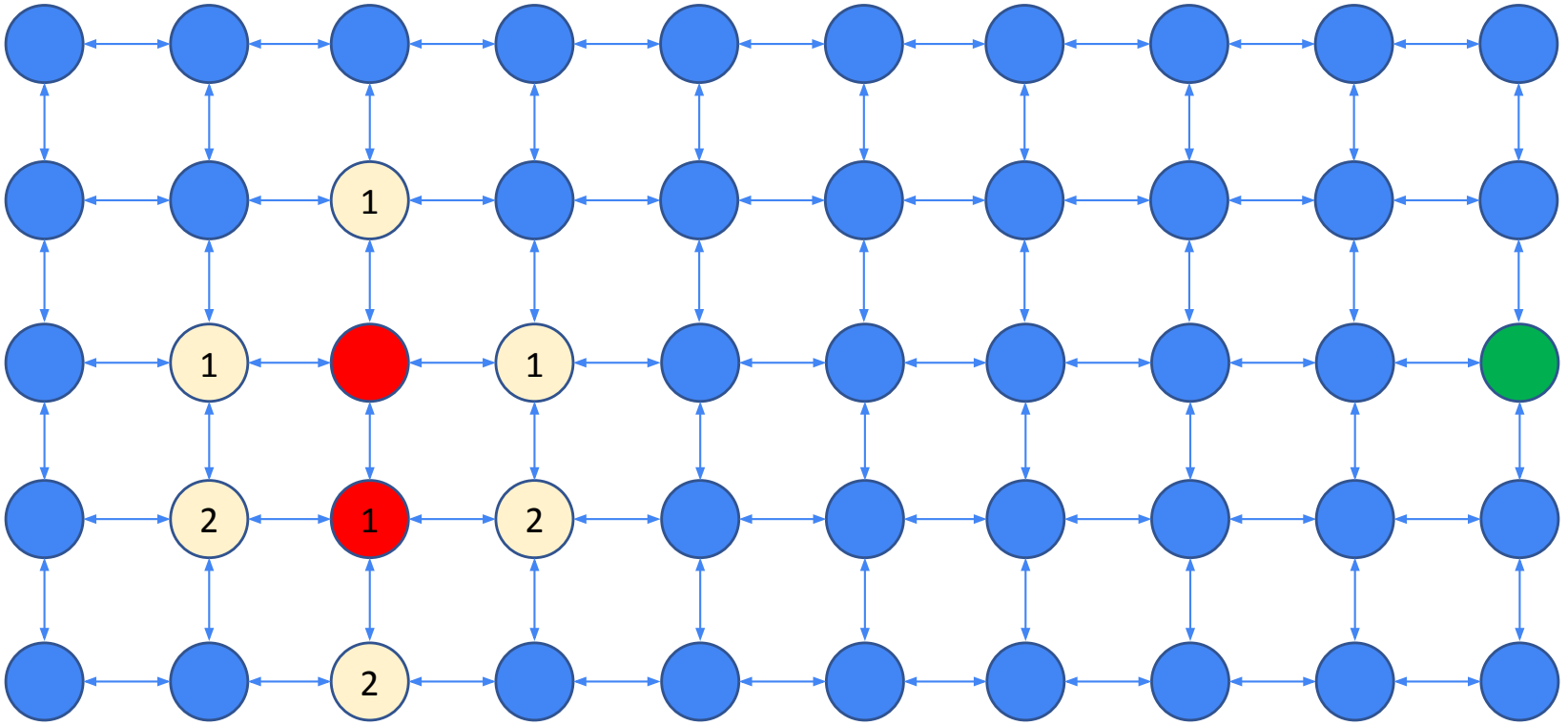
Bad graph example for Dijkstra



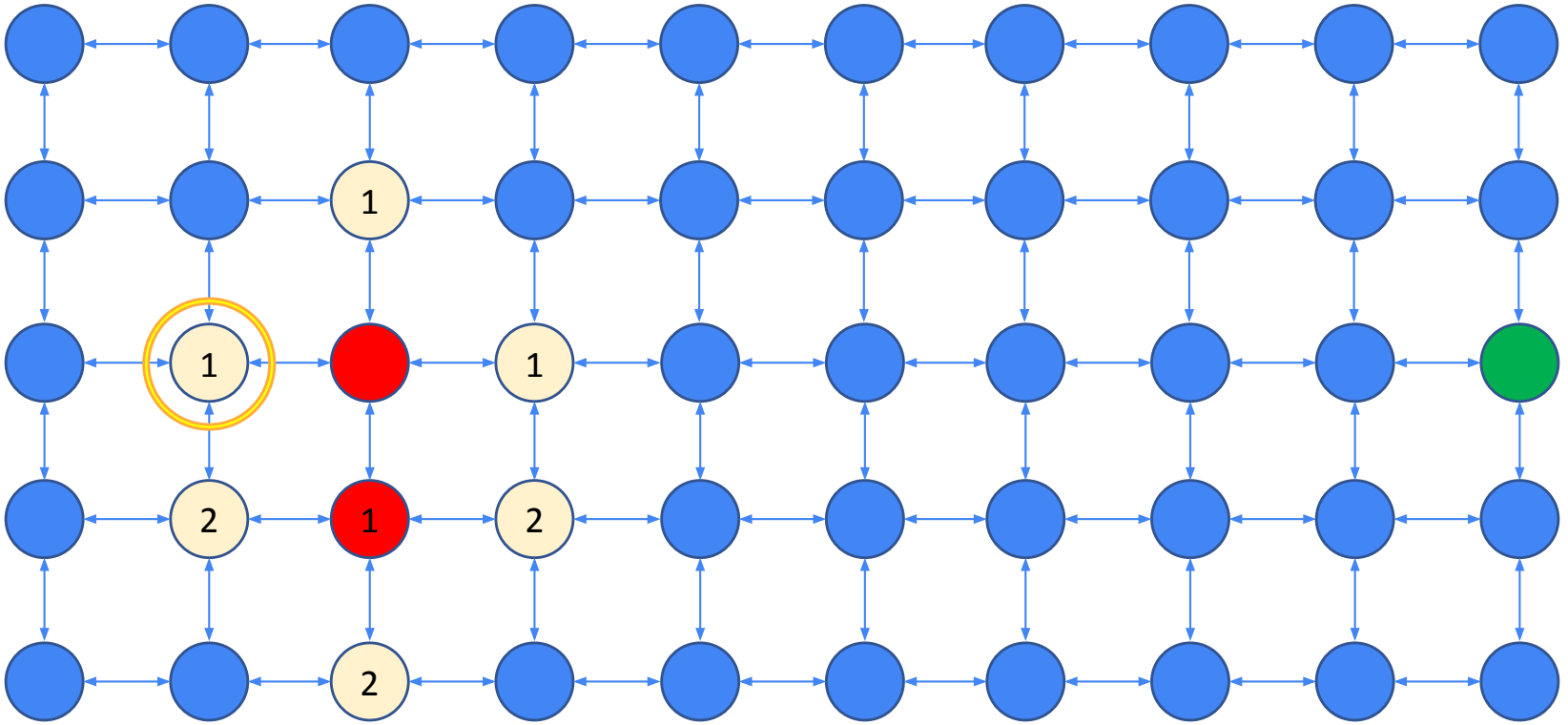
Bad graph example for Dijkstra



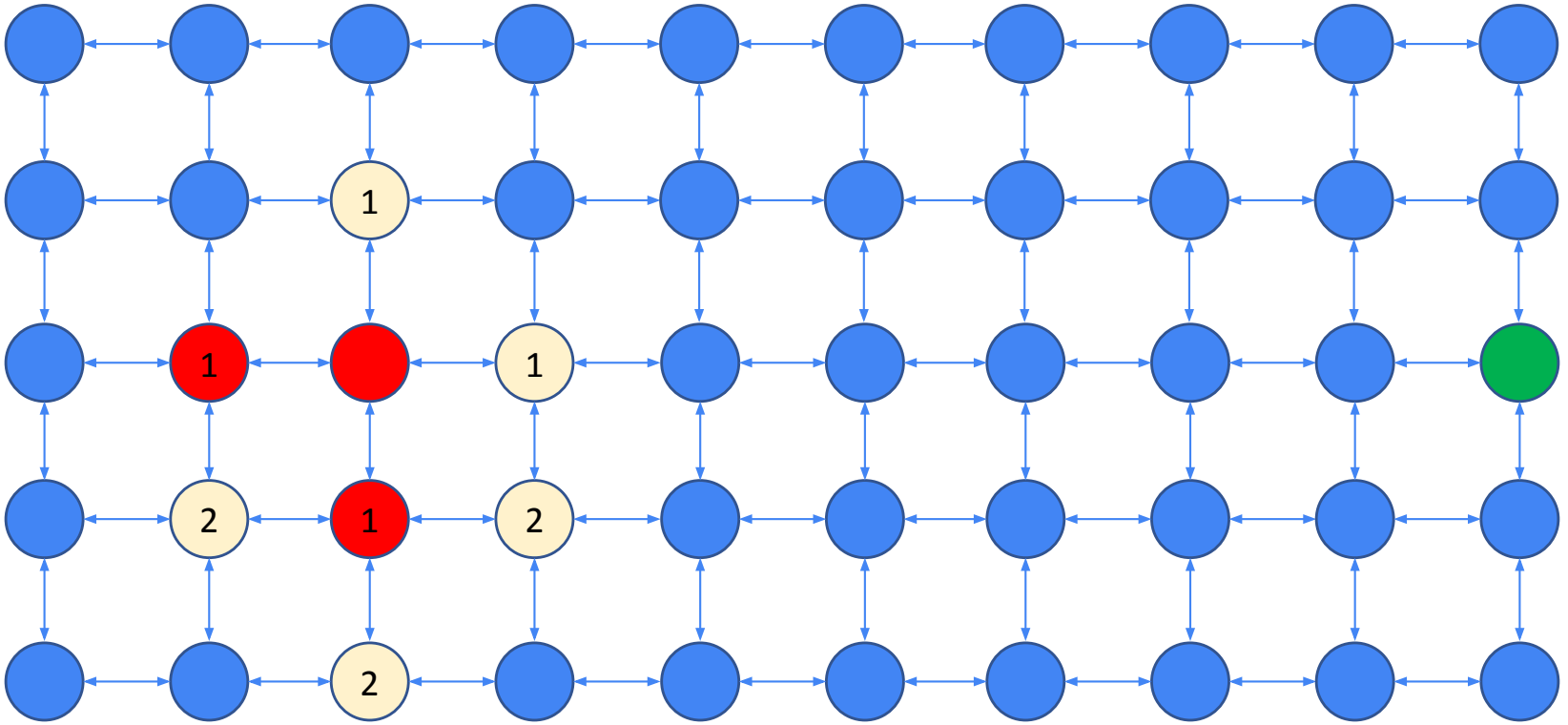
Bad graph example for Dijkstra



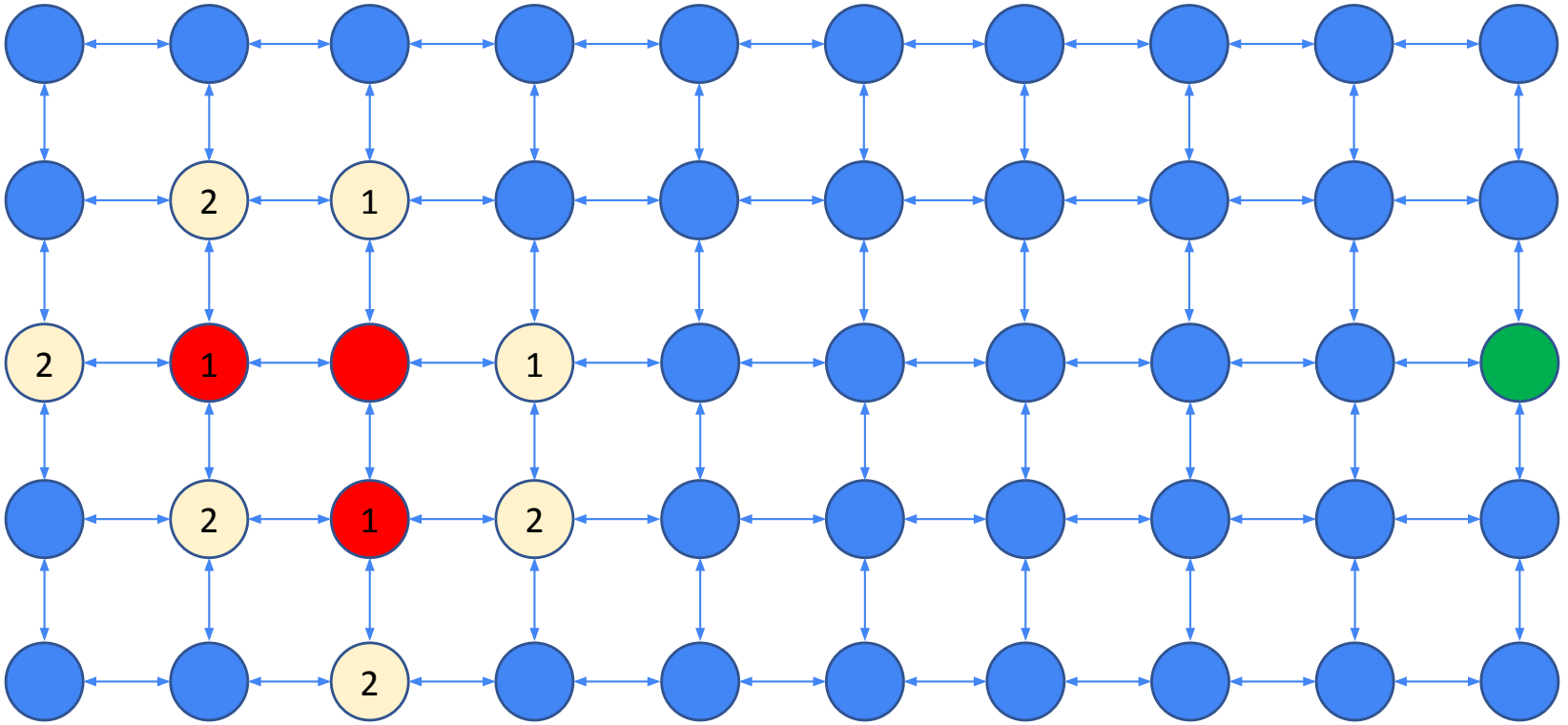
Bad graph example for Dijkstra



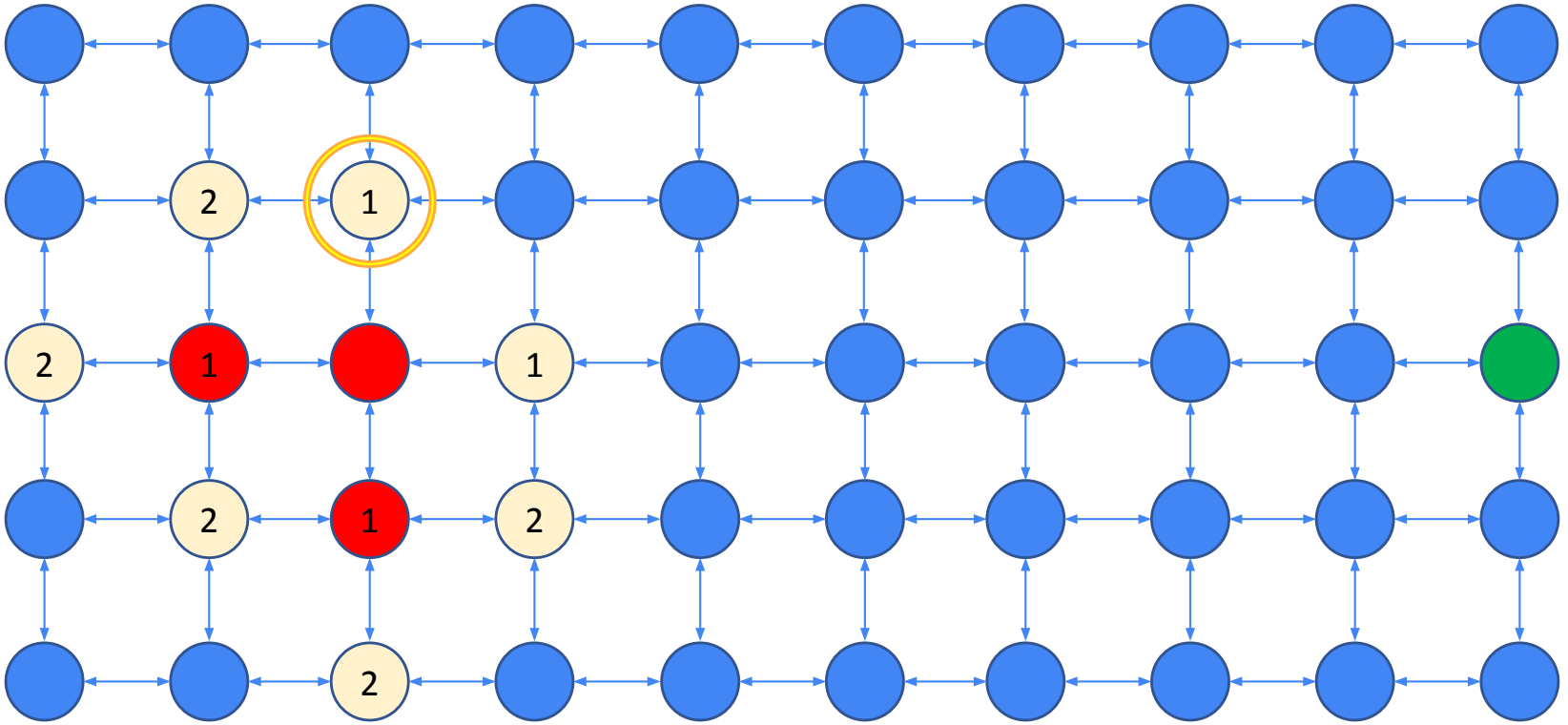
Bad graph example for Dijkstra



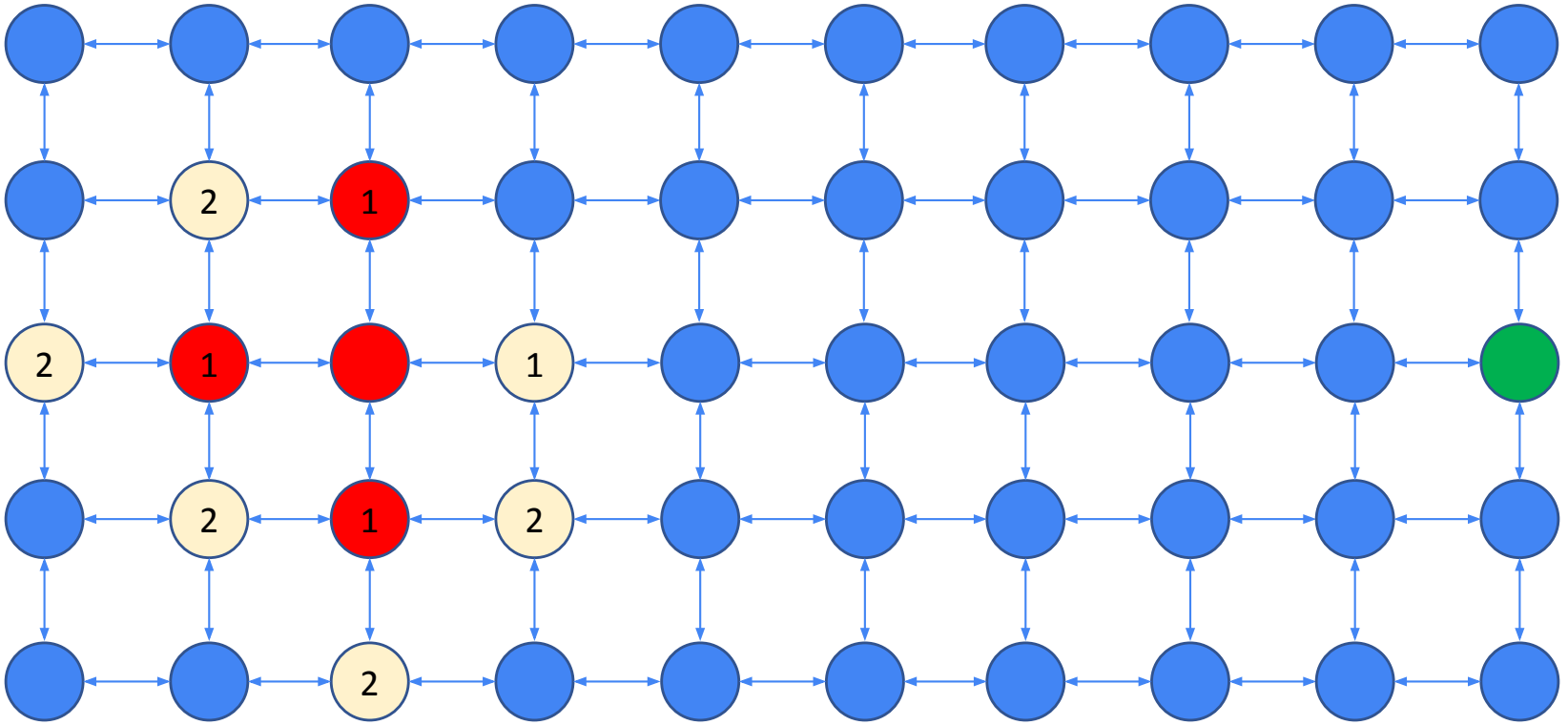
Bad graph example for Dijkstra



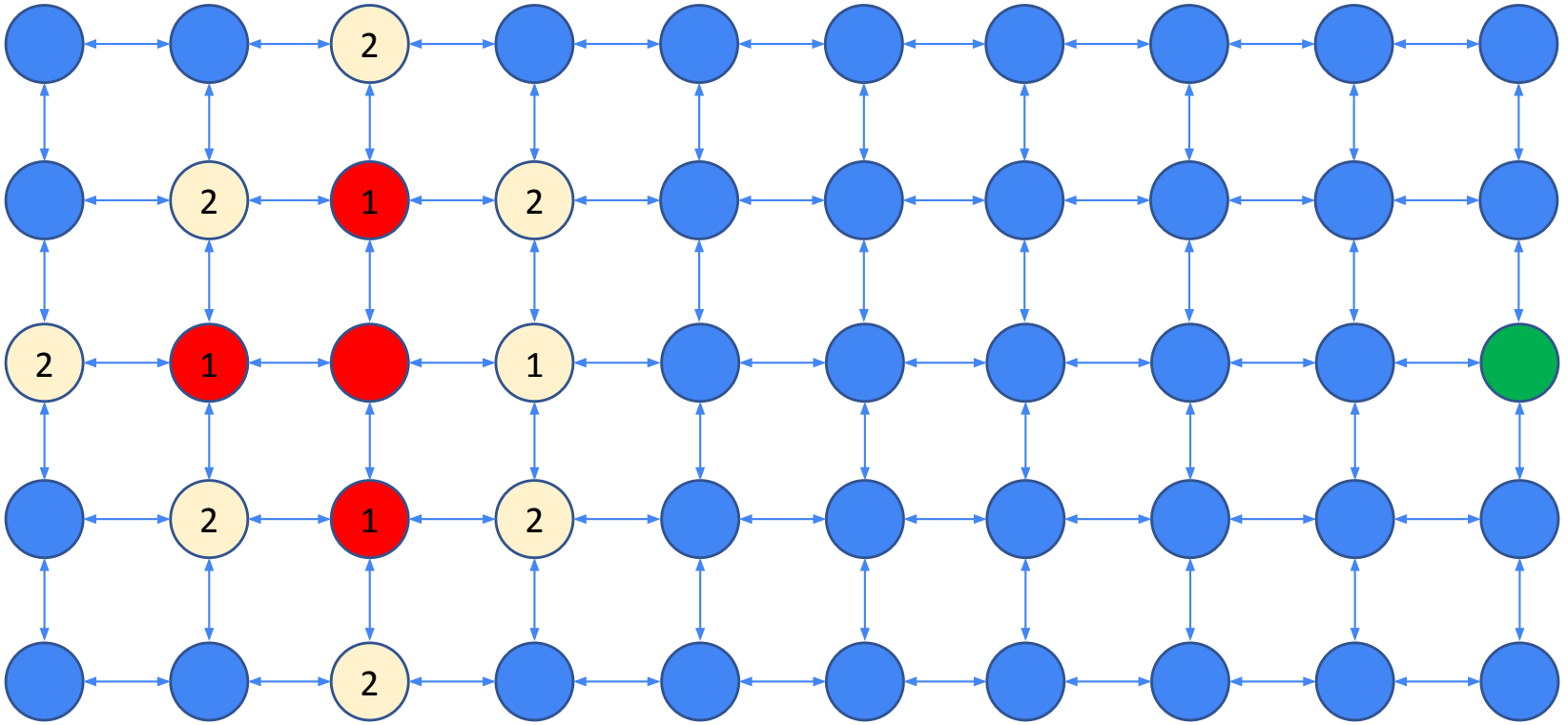
Bad graph example for Dijkstra



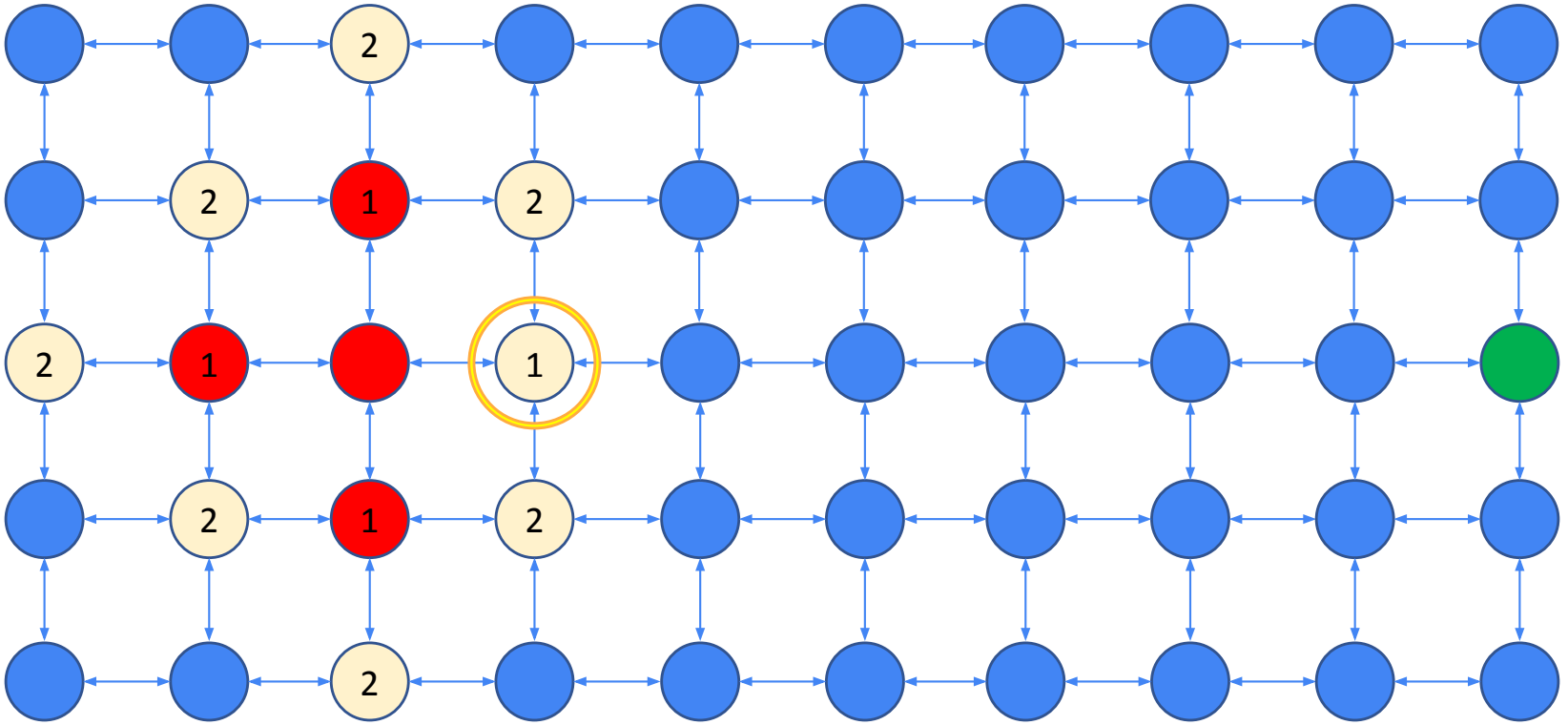
Bad graph example for Dijkstra



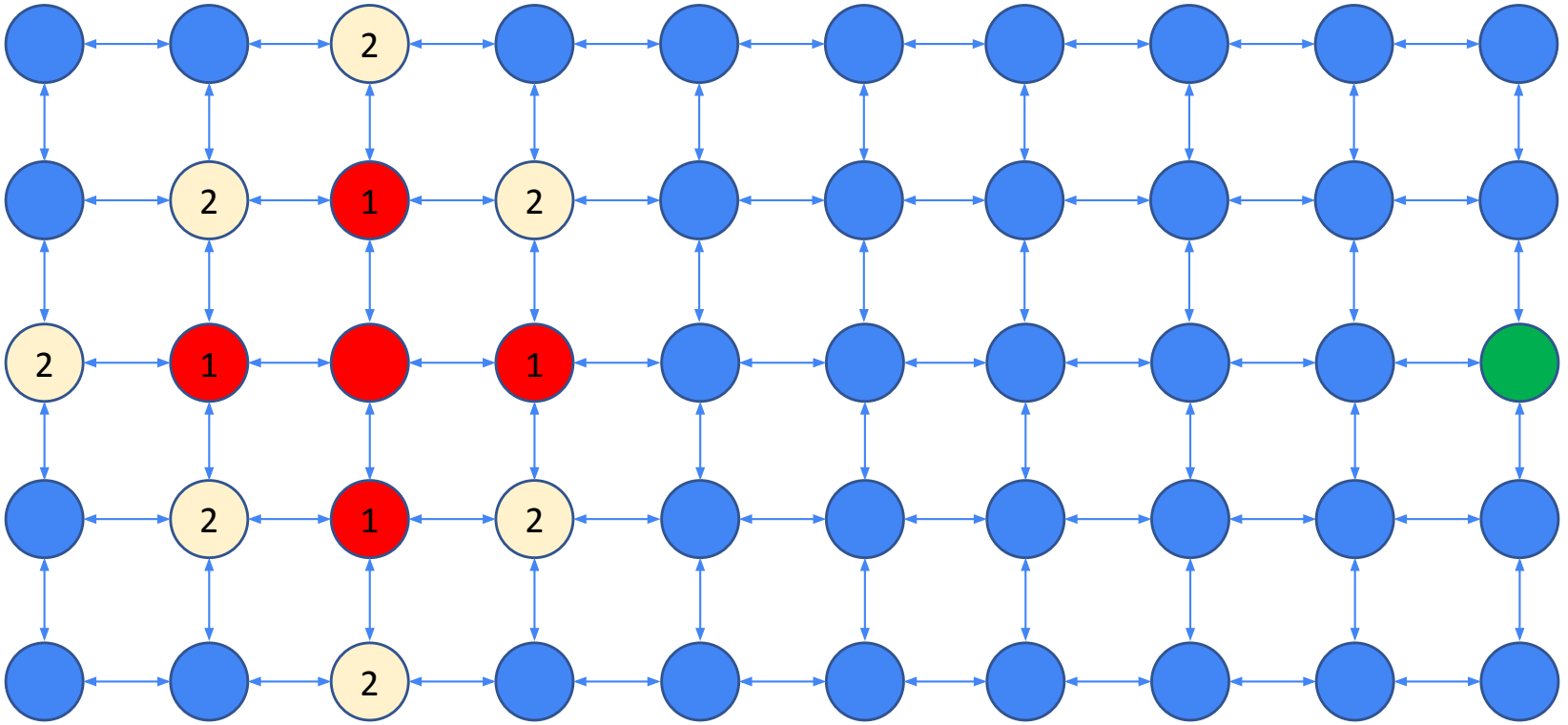
Bad graph example for Dijkstra



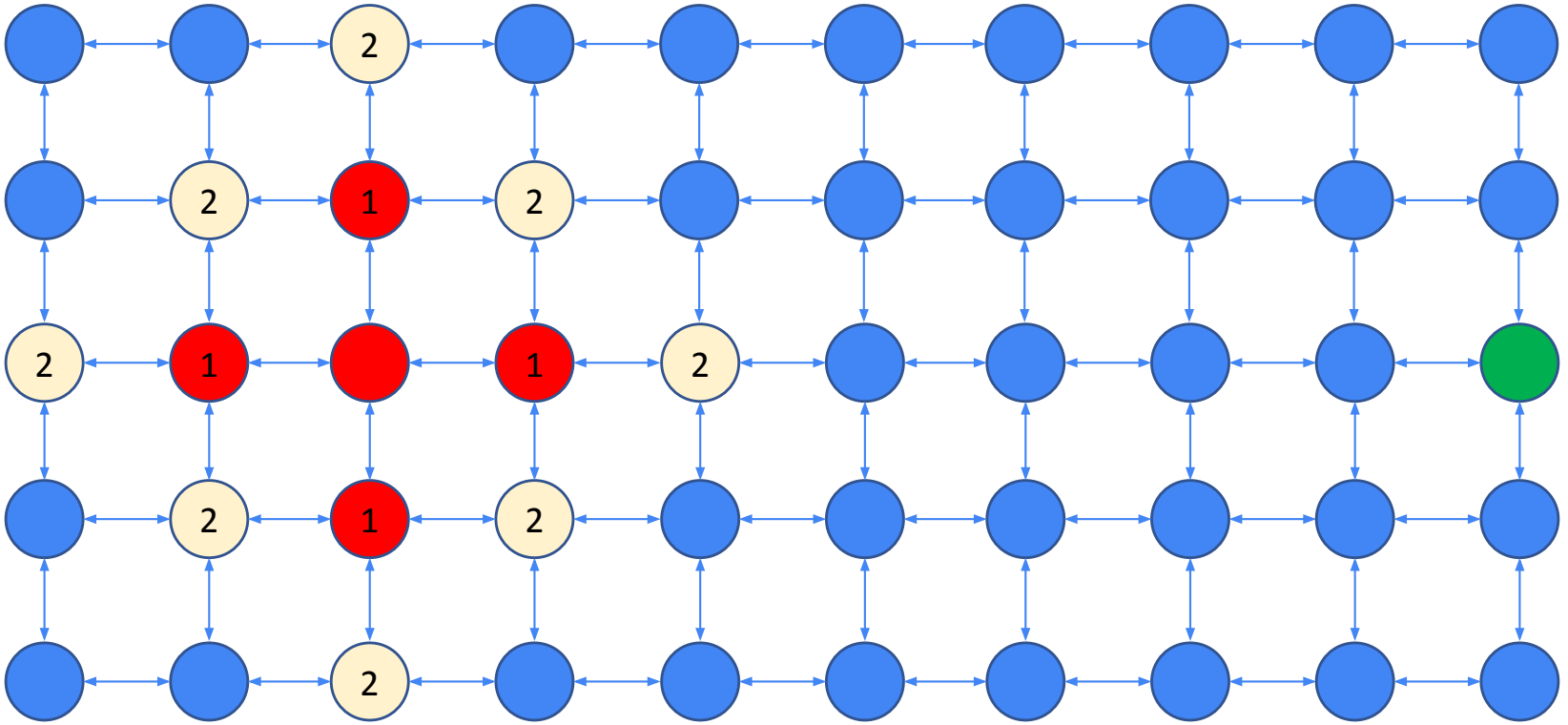
Bad graph example for Dijkstra



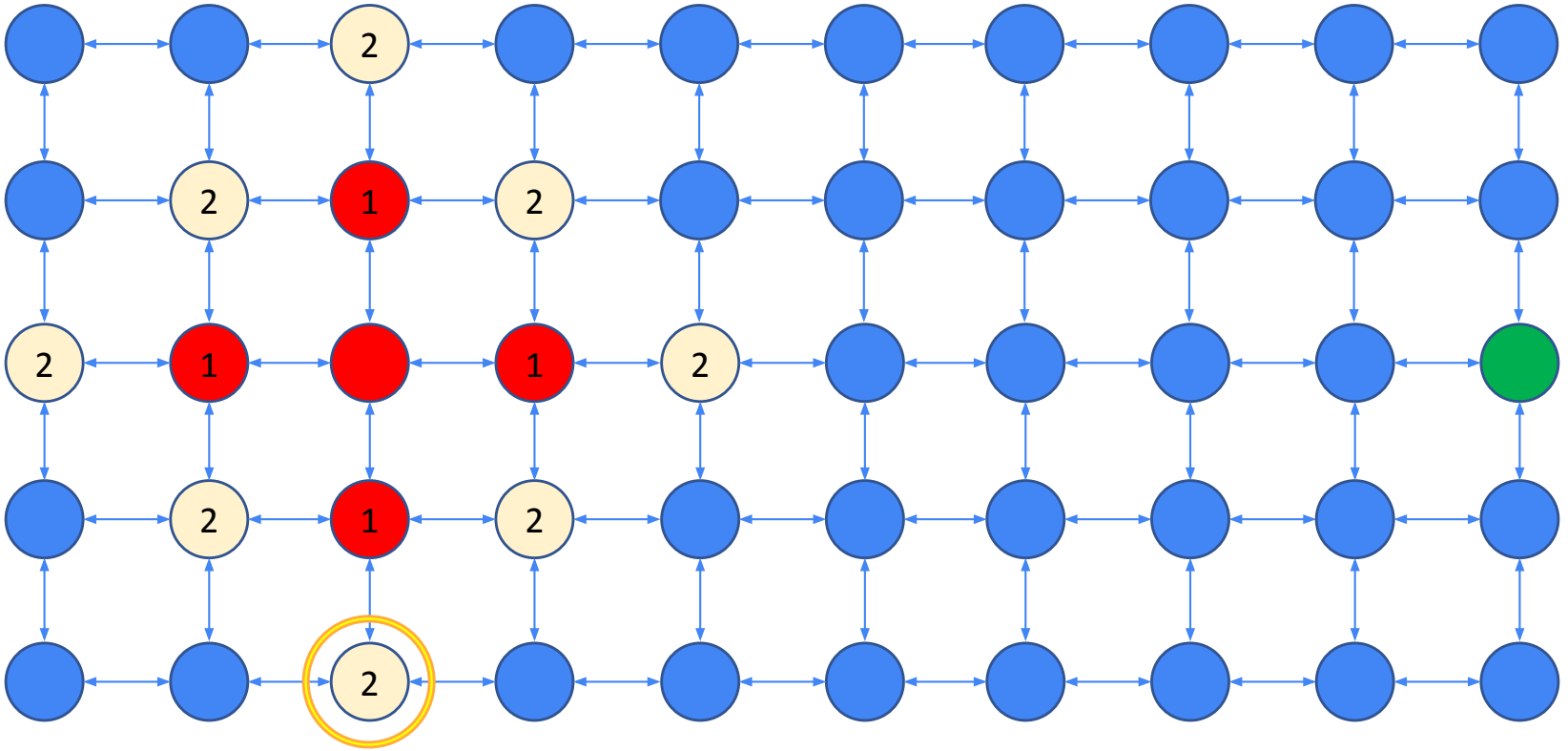
Bad graph example for Dijkstra



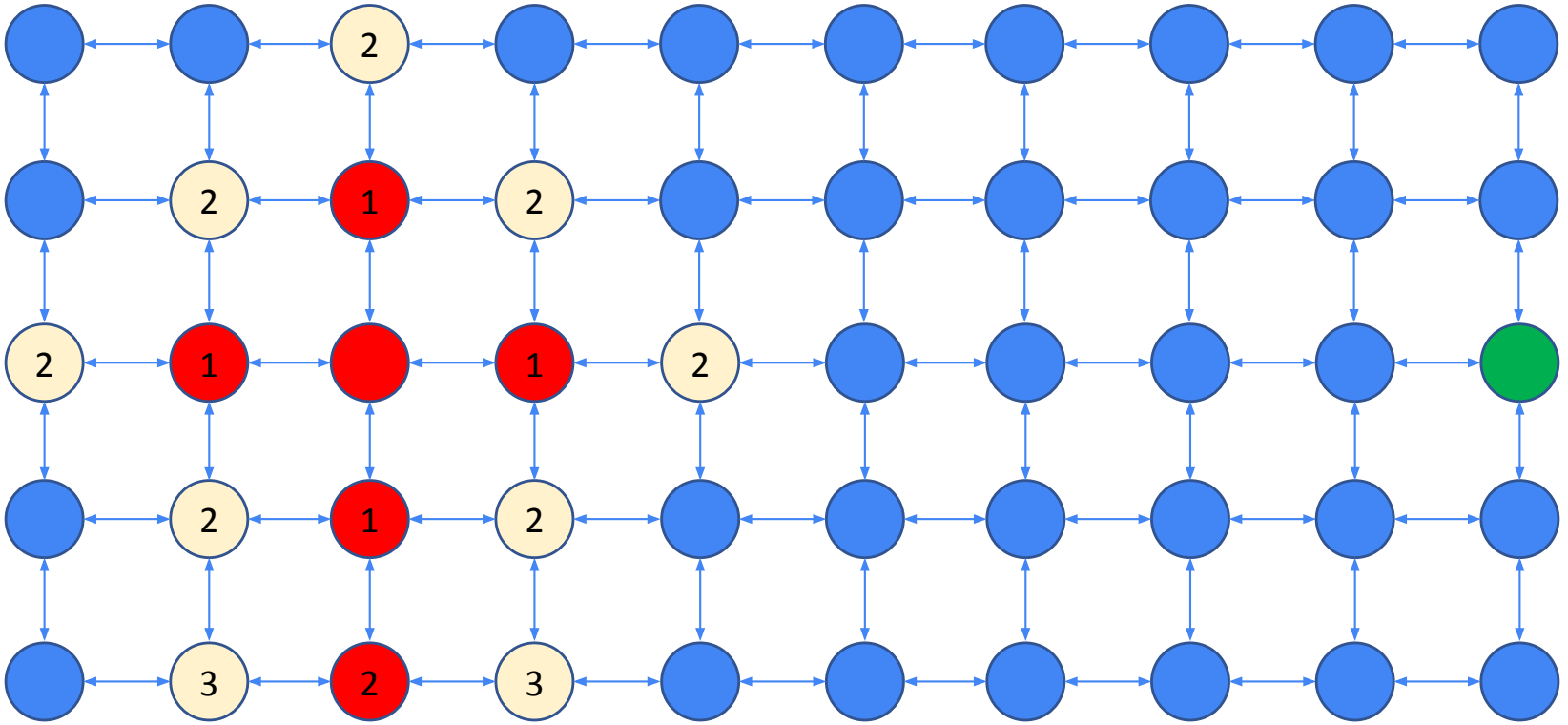
Bad graph example for Dijkstra



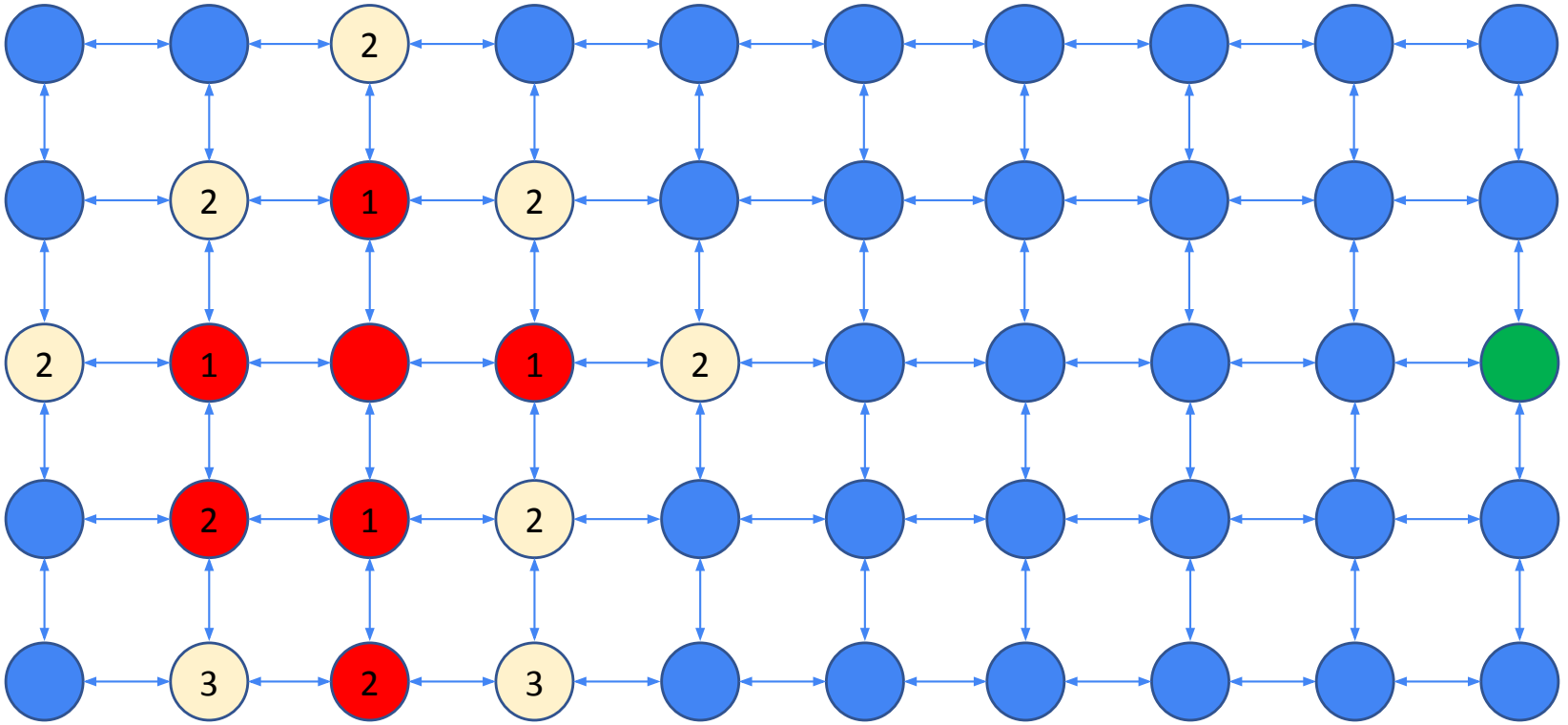
Bad graph example for Dijkstra



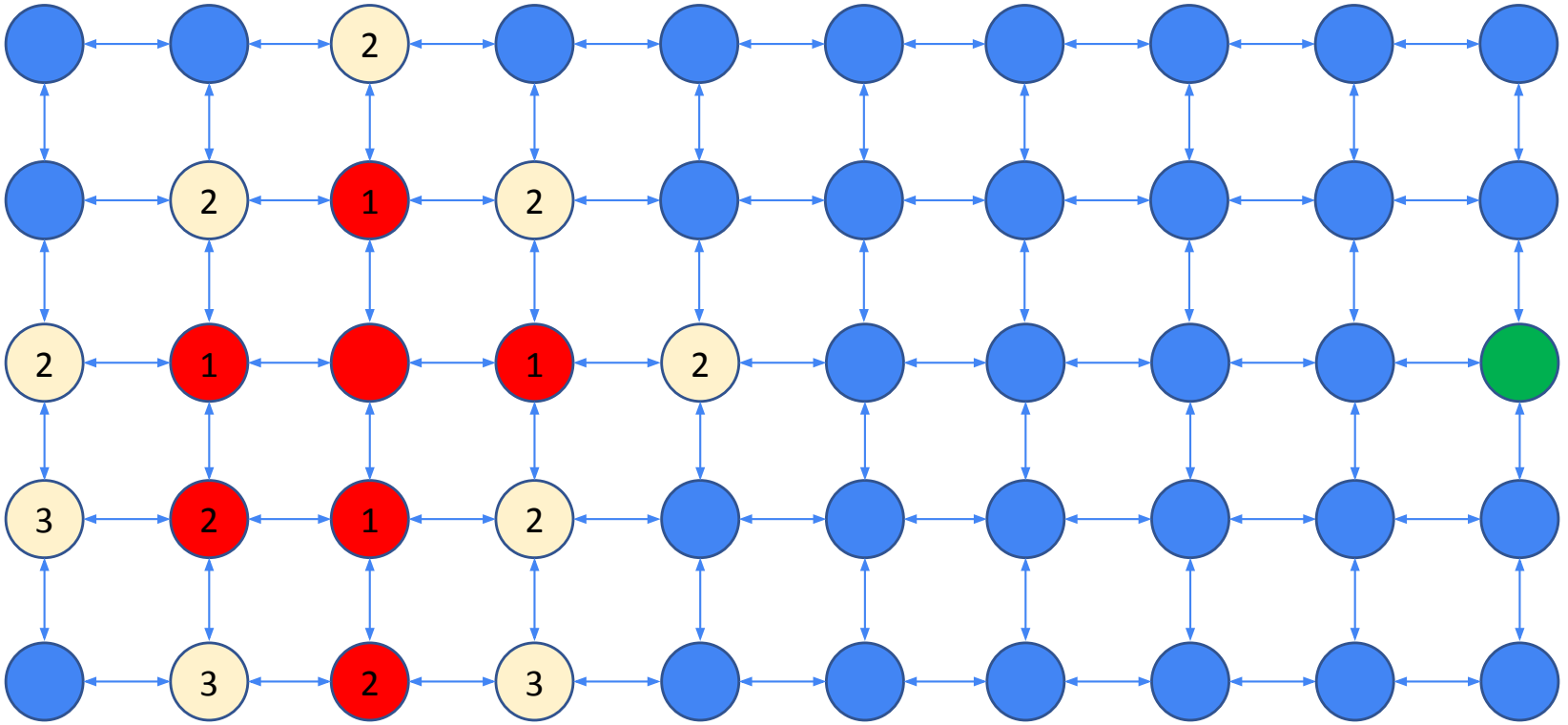
Bad graph example for Dijkstra



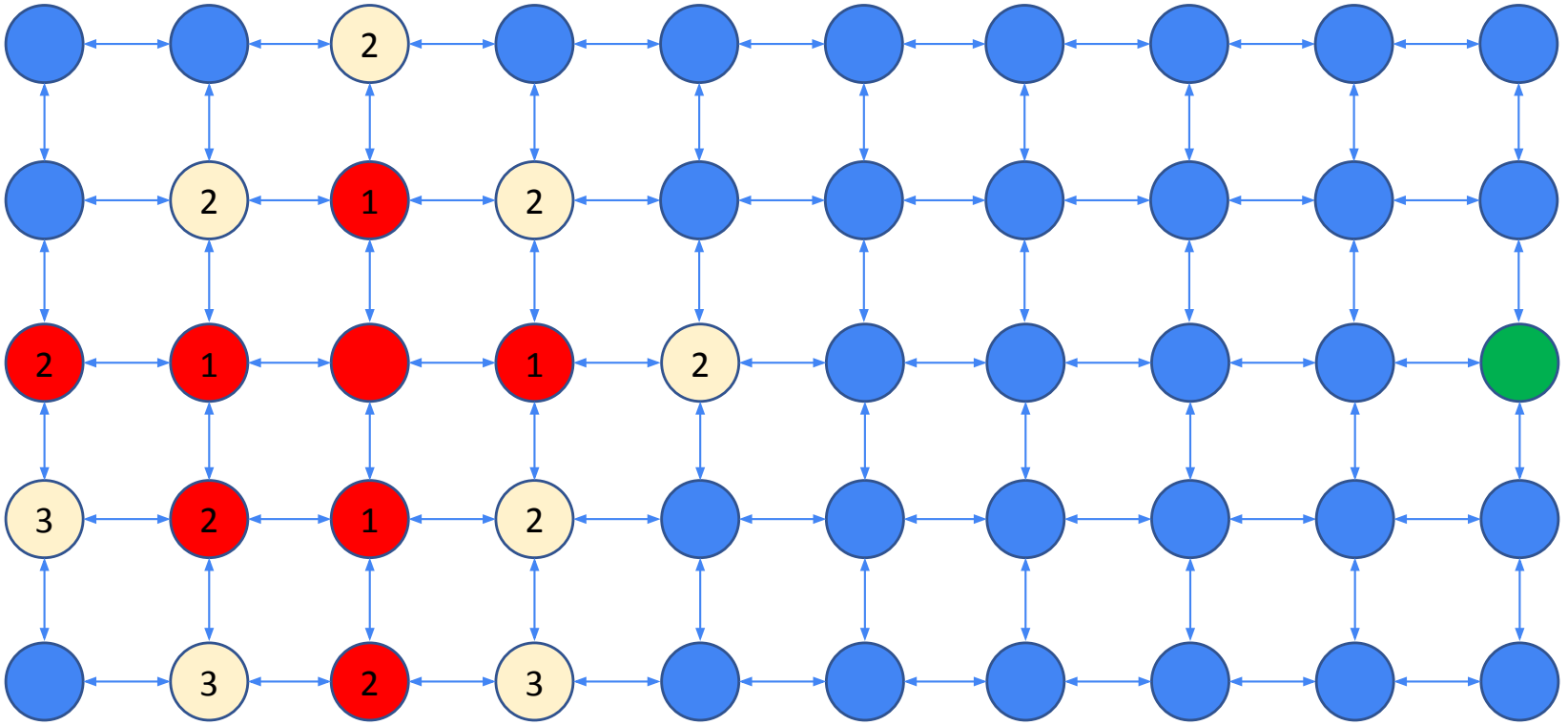
Bad graph example for Dijkstra



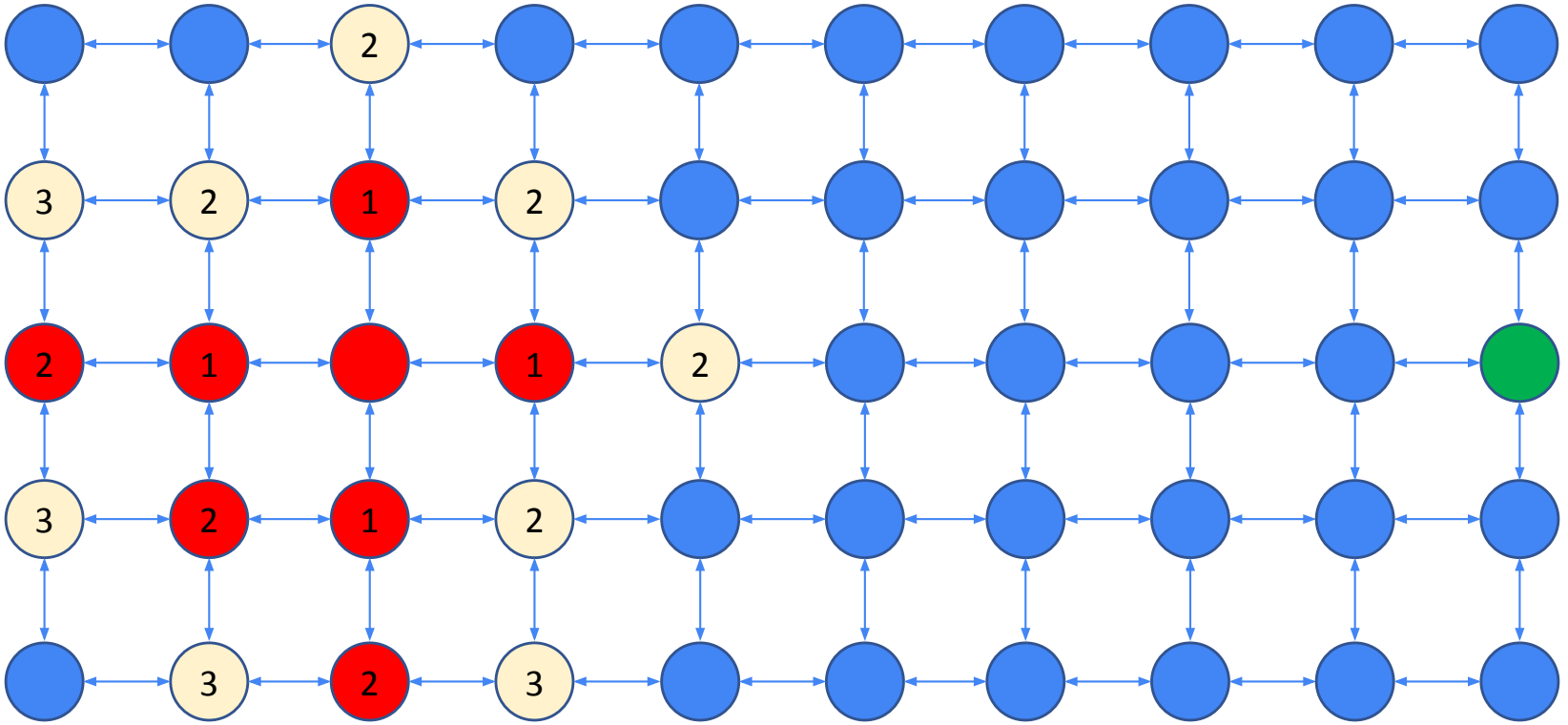
Bad graph example for Dijkstra



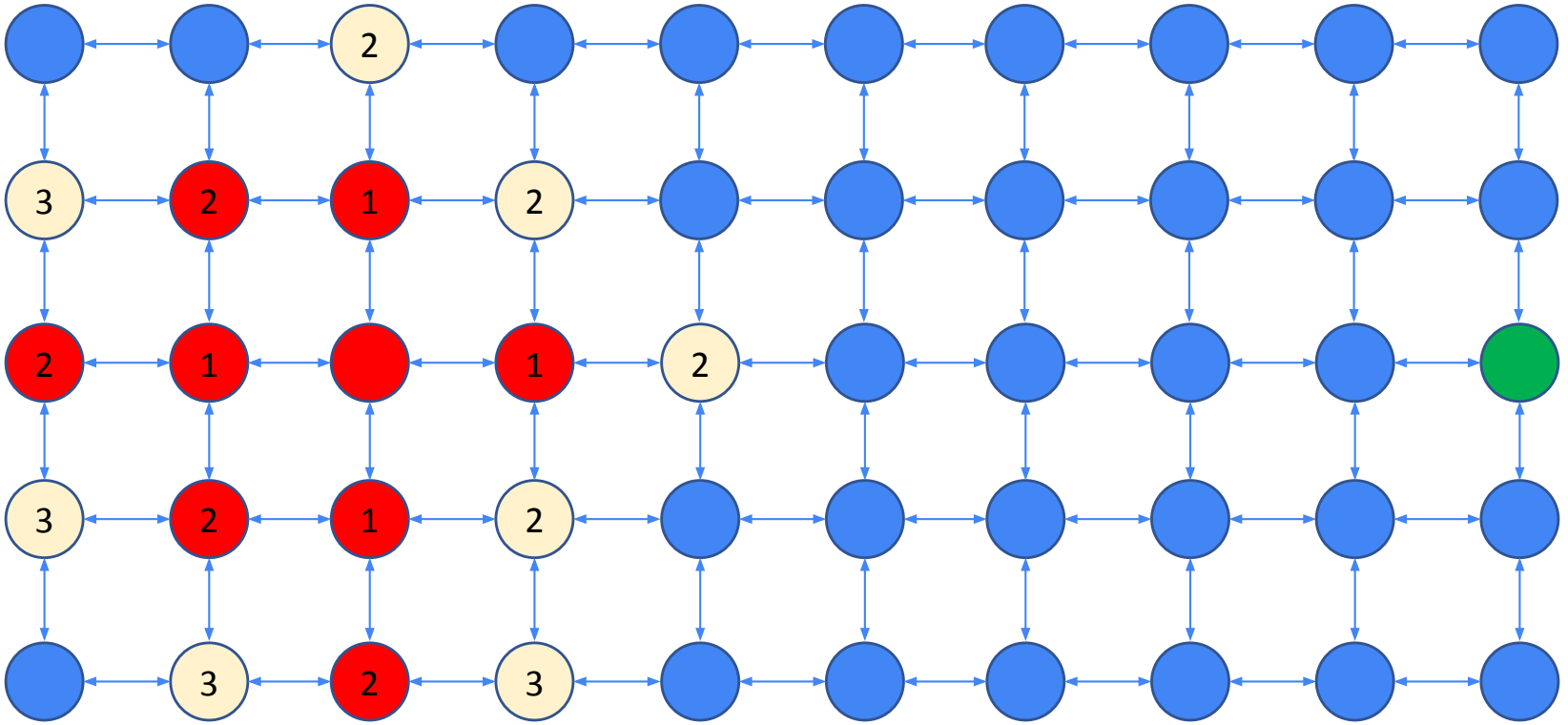
Bad graph example for Dijkstra



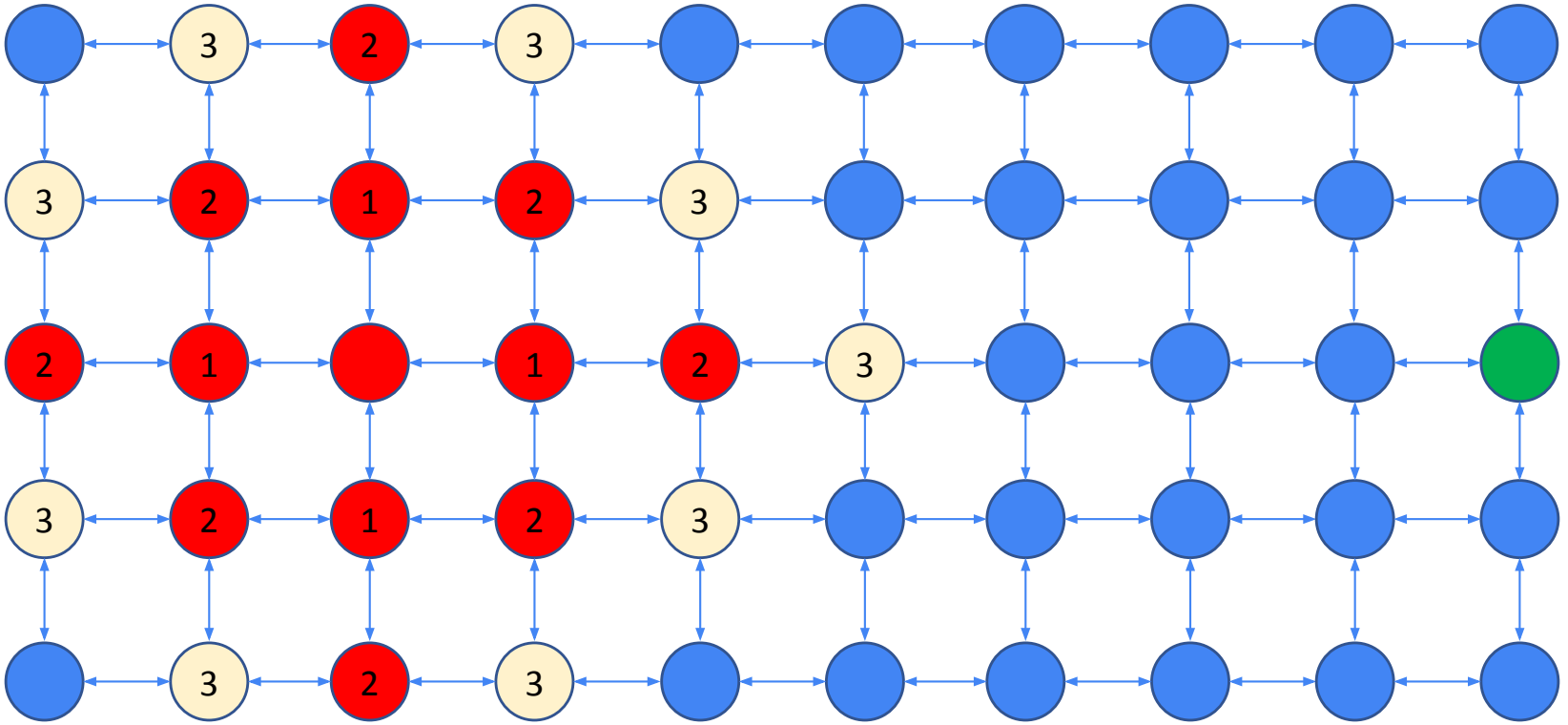
Bad graph example for Dijkstra



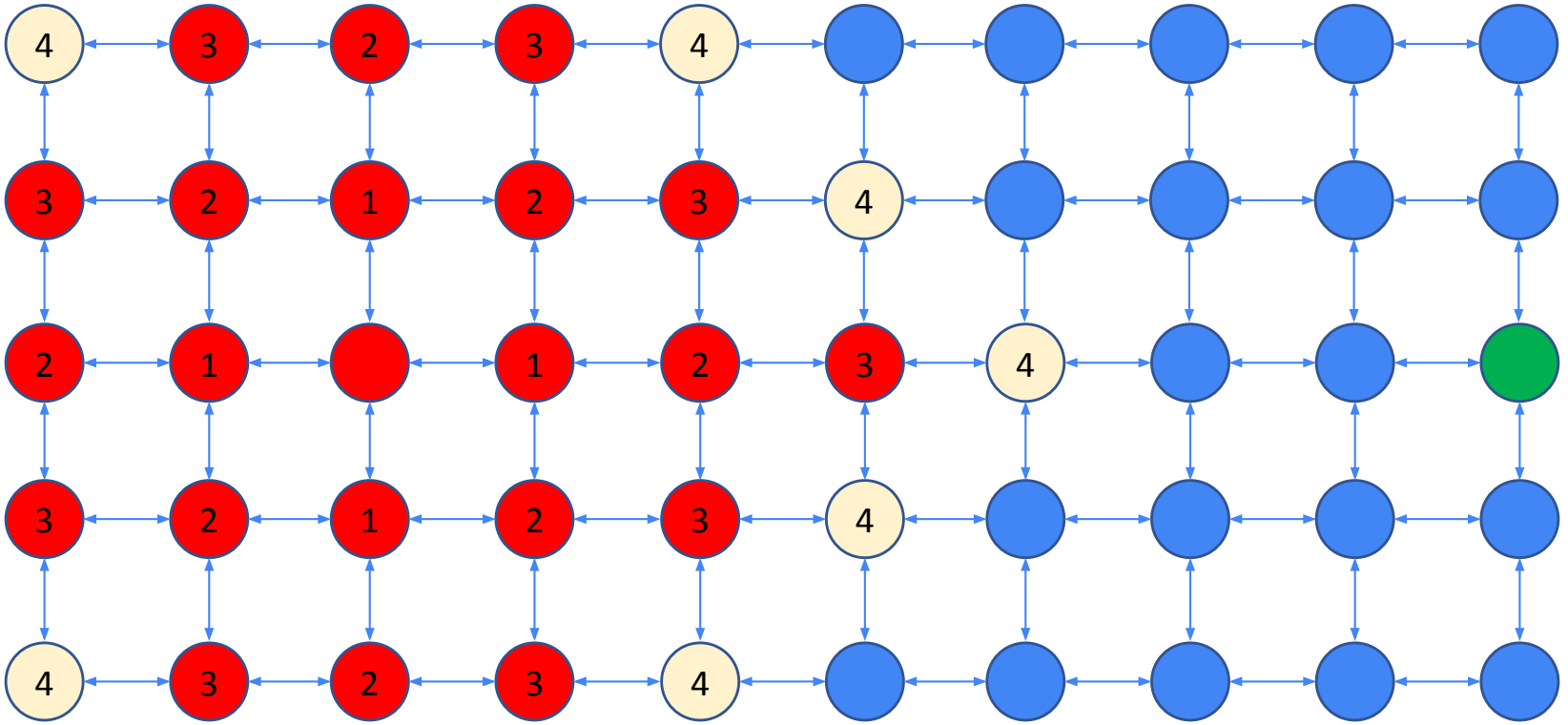
Bad graph example for Dijkstra



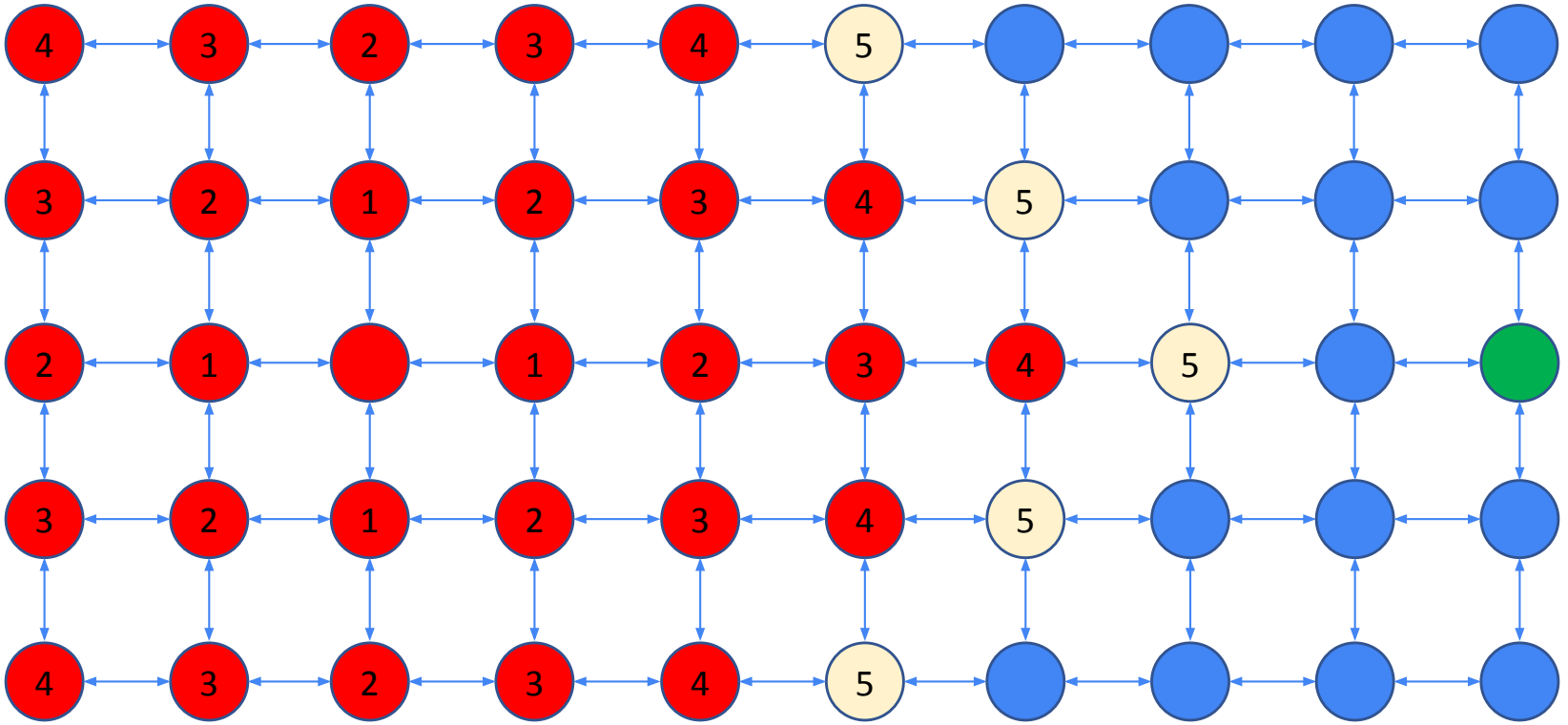
Bad graph example for Dijkstra



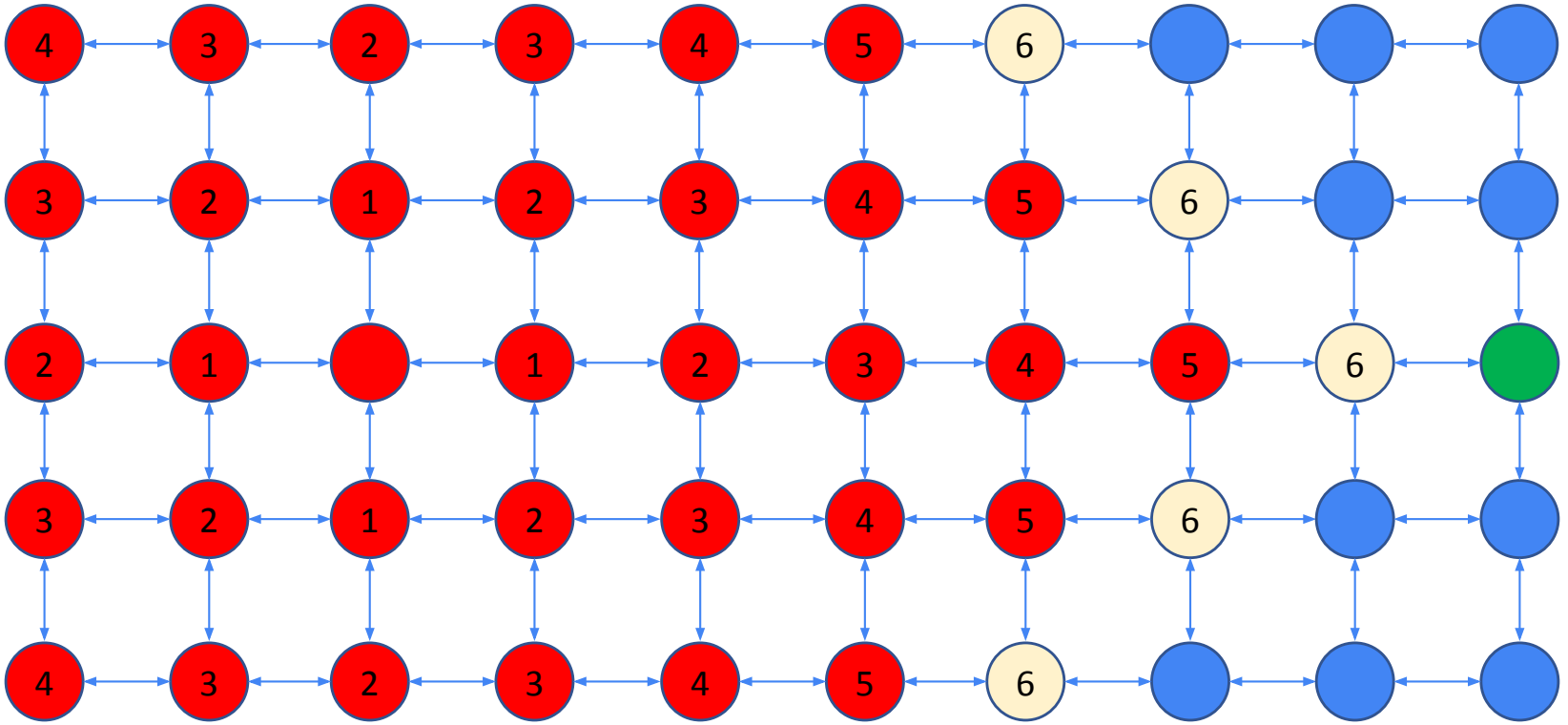
Bad graph example for Dijkstra



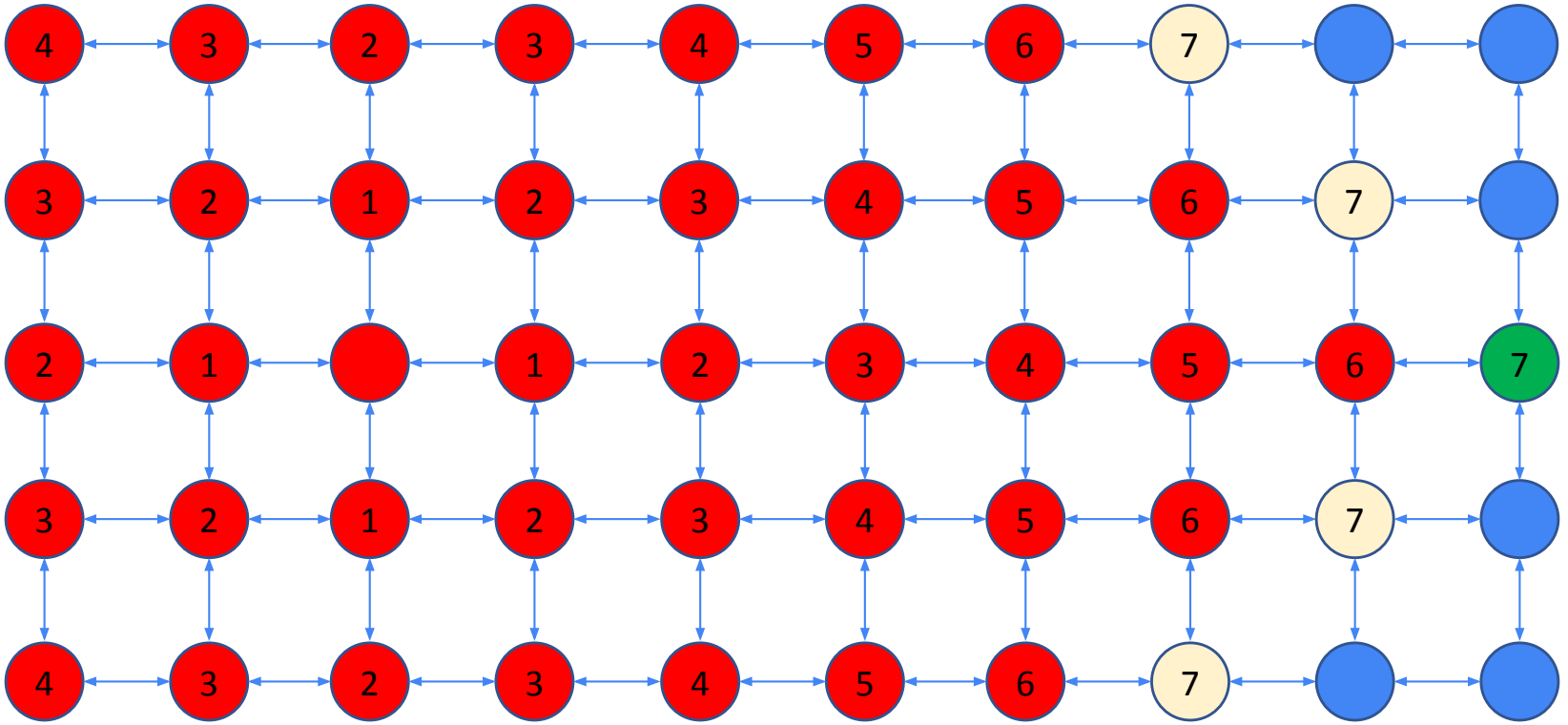
Bad graph example for Dijkstra



Bad graph example for Dijkstra



Bad graph example for Dijkstra



The A* algorithm

This algorithm is used in graph traversal to find the shortest path from a single source to **a single destination**.

Input : $G = (V, E)$ where, s $\in V, t \in V$ for each edge $e \in E$

Post-condition: we want the shortest path from s to t

$X = \{ s \}$

len(s) = 0, len(v) = ∞ (for all vertices)

while there is an edge (v,w), such that:

v $\in X$, w $w \notin X$ do:

if $v = t$, then exit

(a, b) = minimise len(v) + h(w, t)

add b to X

len(b) = len(a) + h(b, A)

The A* algorithm

This algorithm is used in graph traversal to find the shortest path from a single source to **a single destination**.

Input : $G = (V, E)$ where, s $\in V, t \in V$ for each edge $e \in E$

Post-condition: we want the shortest path from s to t

$X = \{ s \}$

len(s) = 0, len(v) = ∞ (for all vertices)

while there is an edge (v,w), such that:

v $\in X$, w $w \notin X$ do:

if $v = t$, then exit

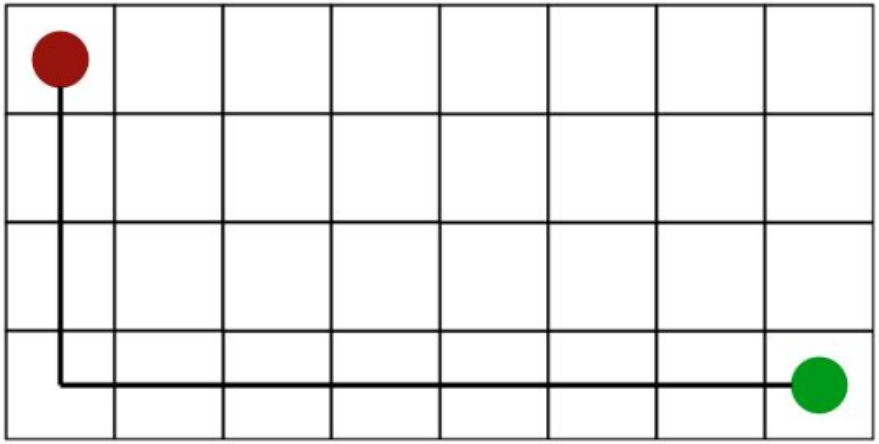
(a, b) = minimise len(v) + h(w, t)

add b to X

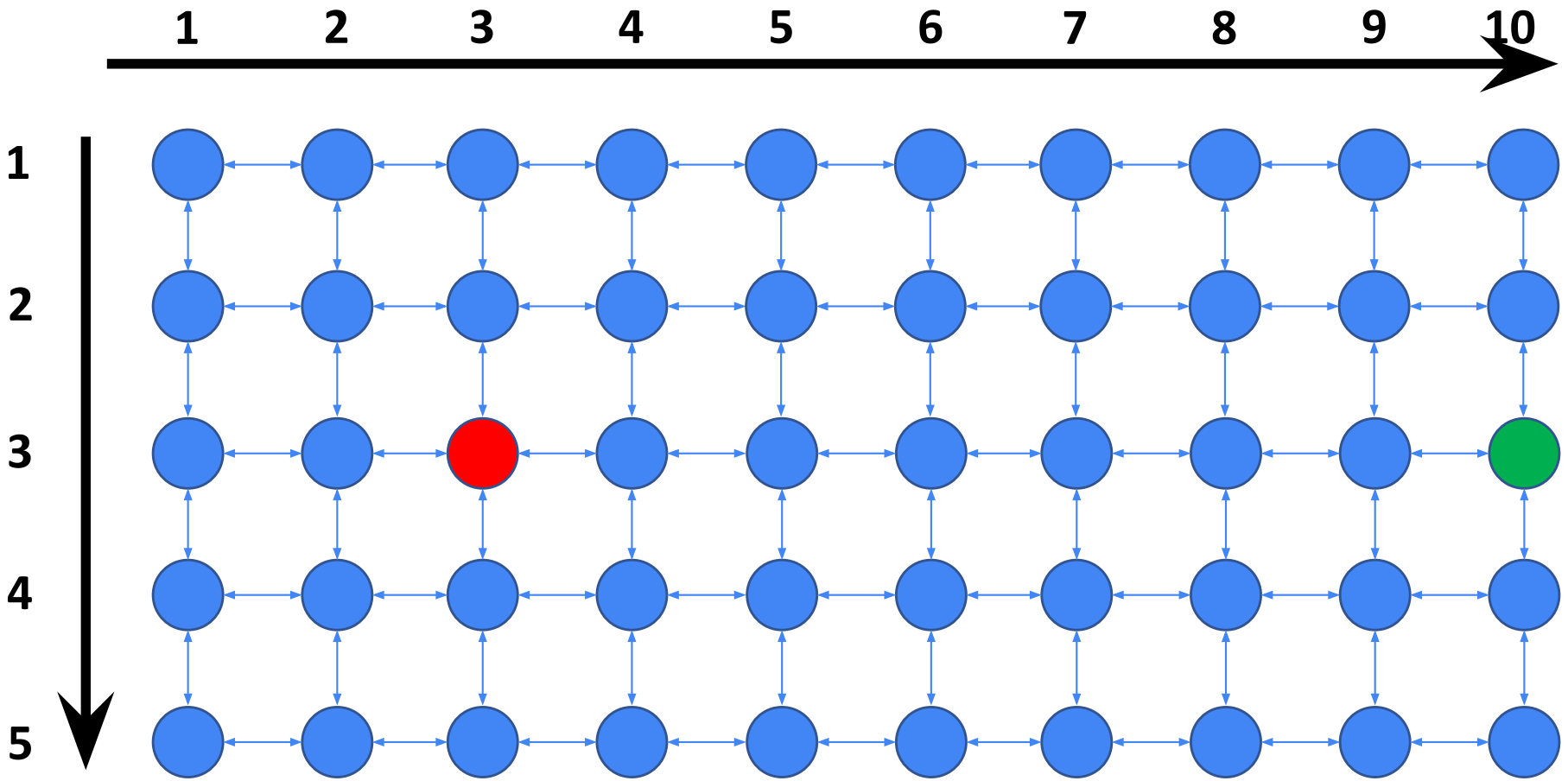
len(b) = len(a) + h(b, A)

A* with Manhattan Distance (for grid-like graphs)

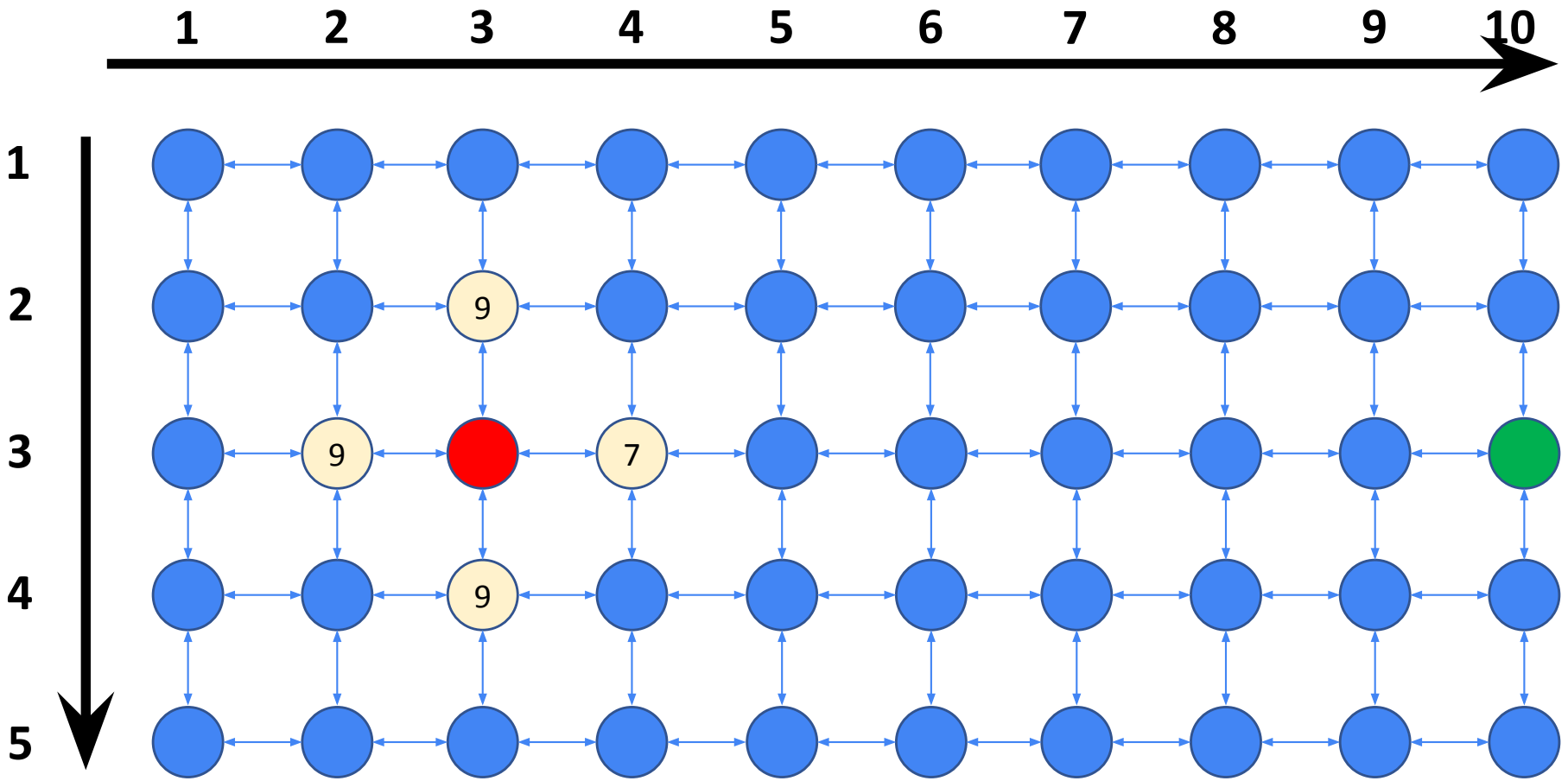
```
function h(current, target) {
    return abs(current.row - target.row) +
           abs(current.col - target.col)
}
```



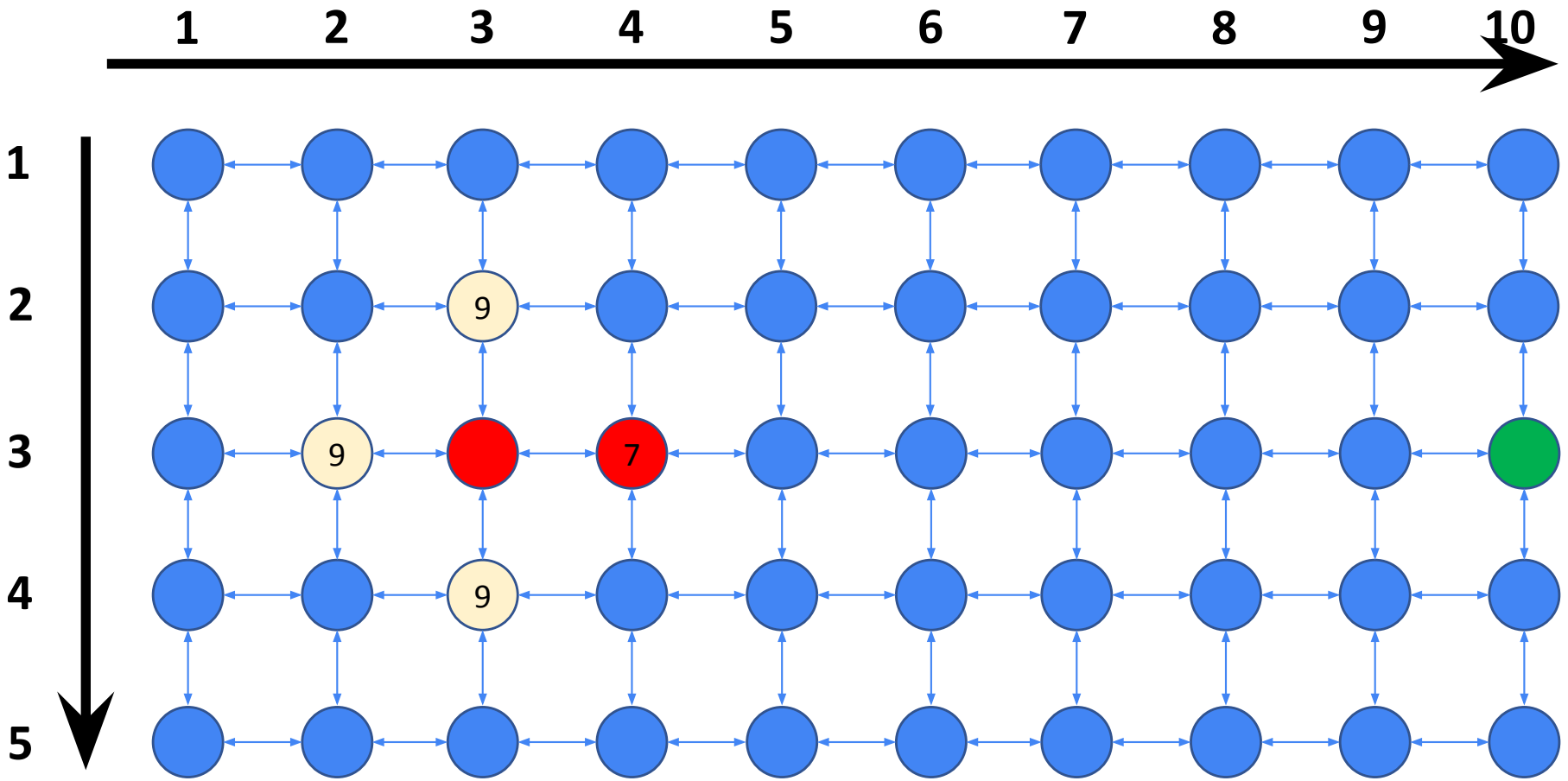
Using A* with Manhattan Distance



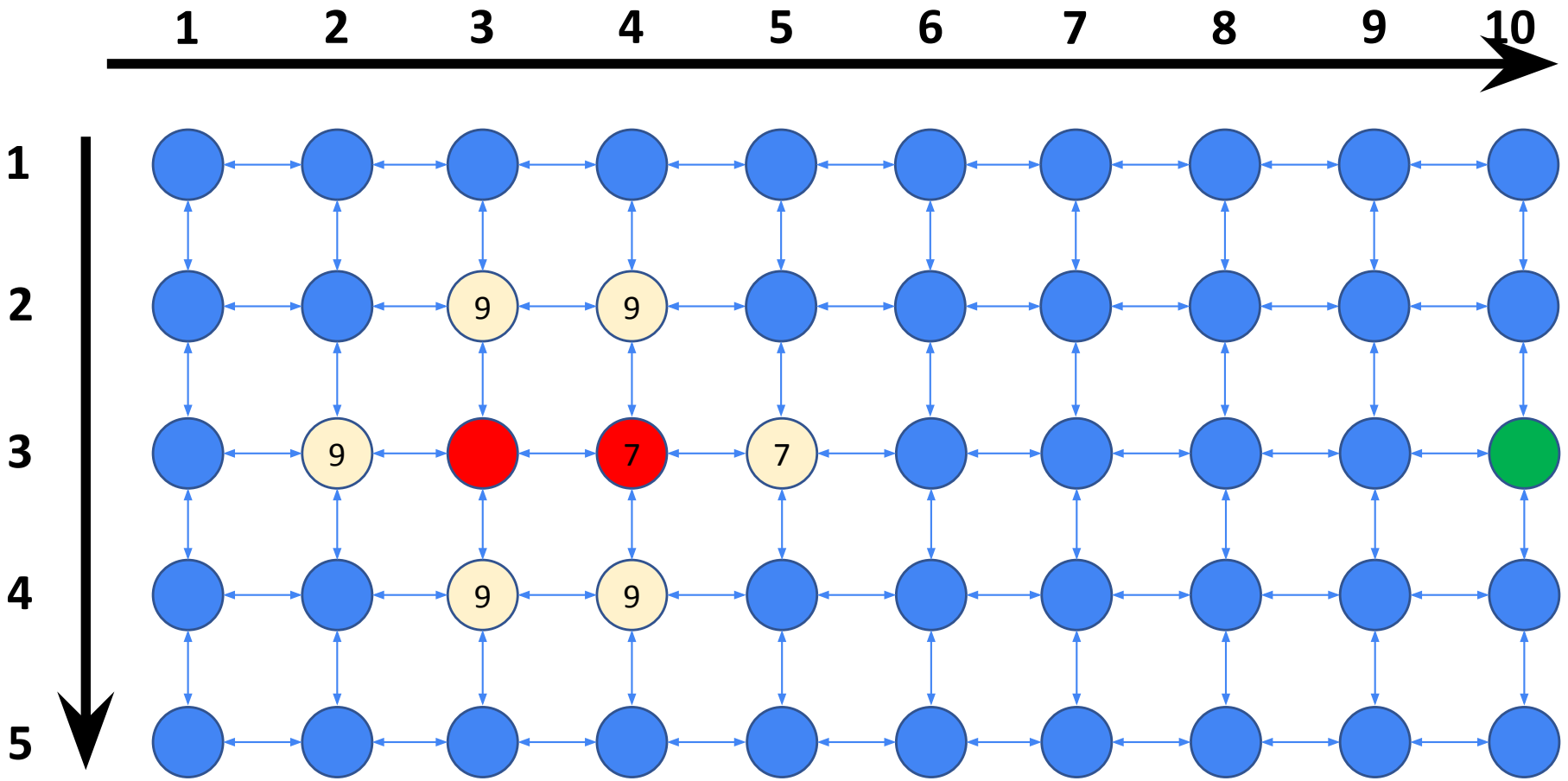
Using A* with Manhattan Distance



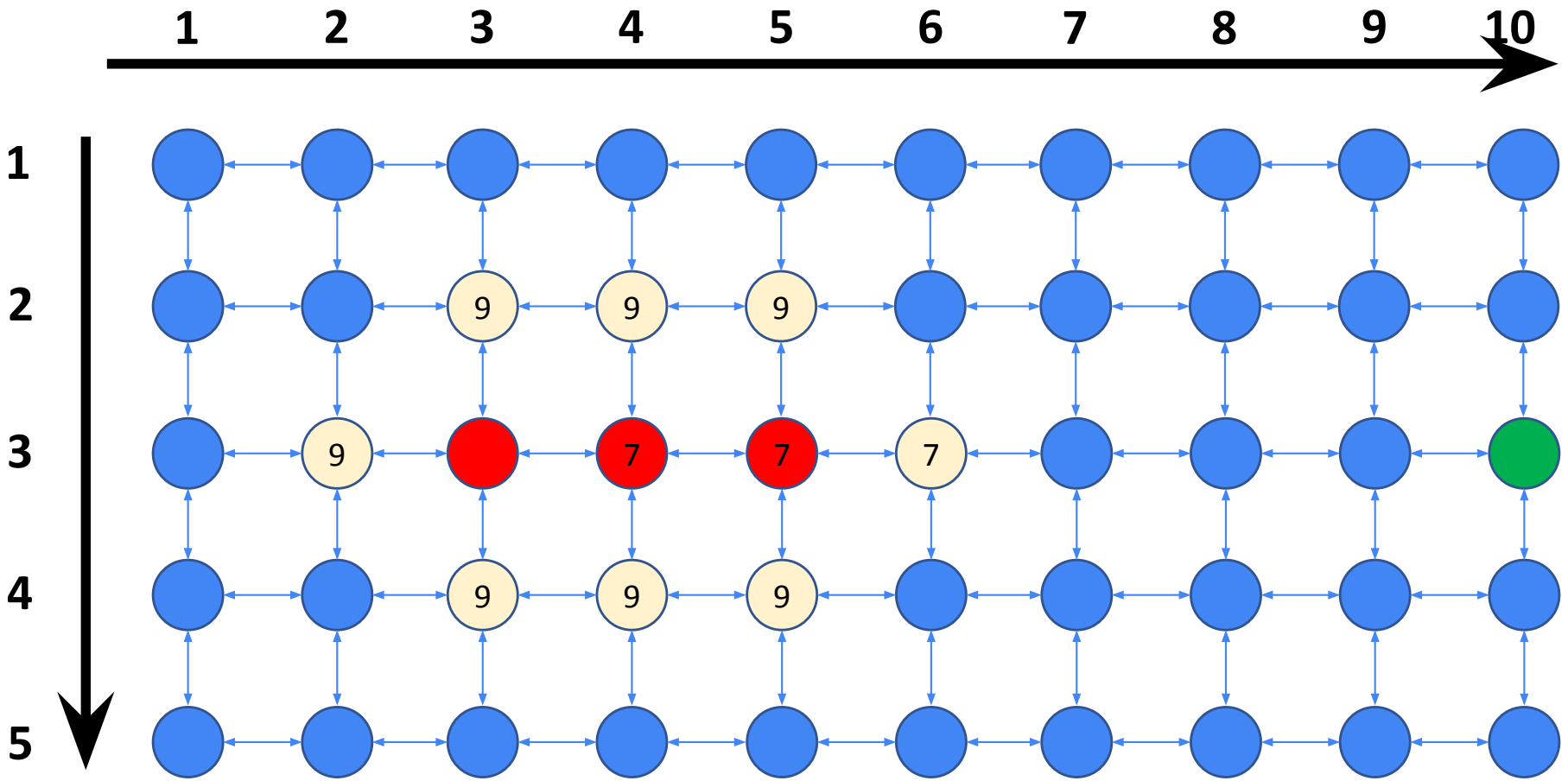
Using A* with Manhattan Distance



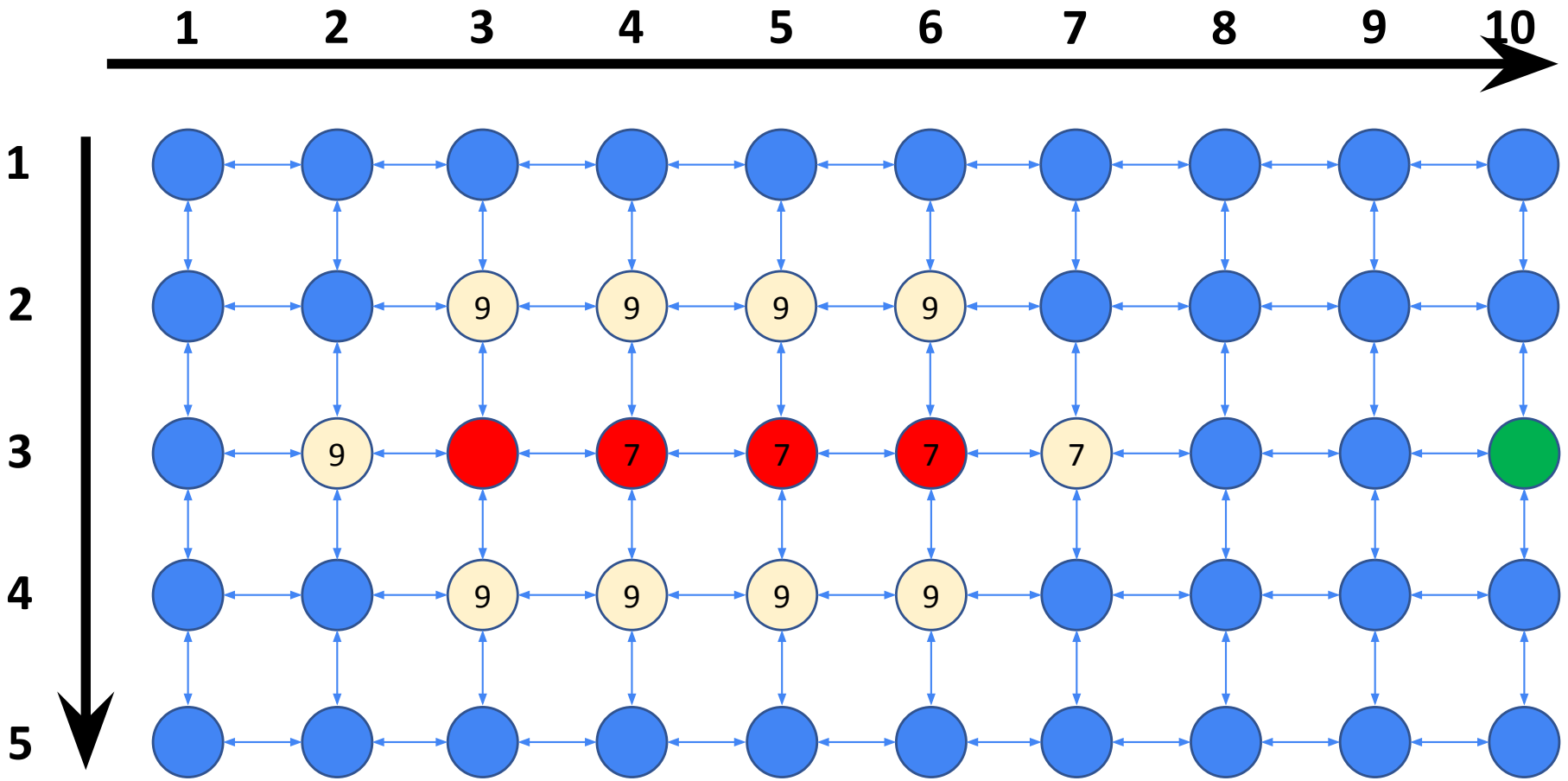
Using A* with Manhattan Distance



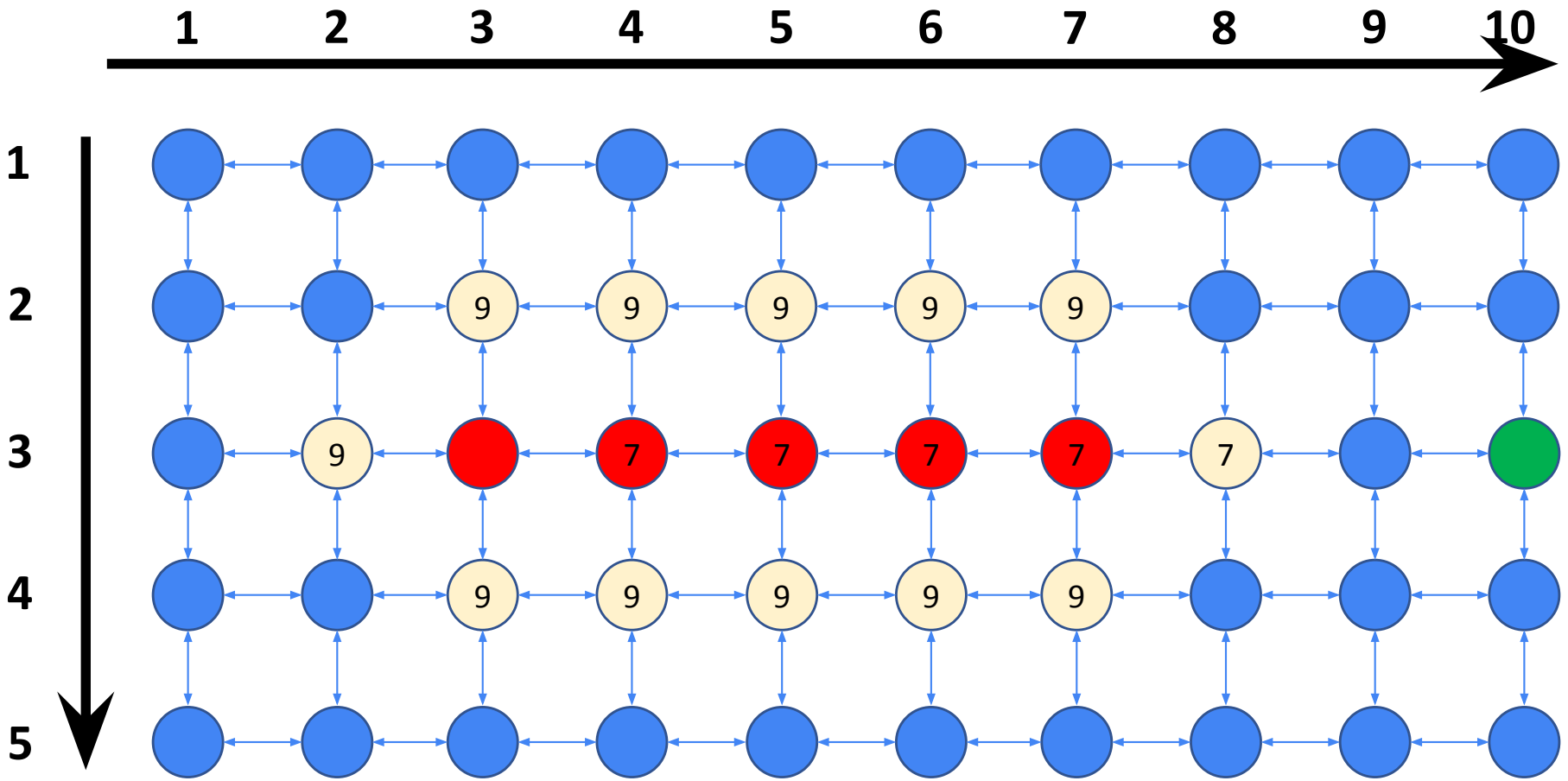
Using A* with Manhattan Distance



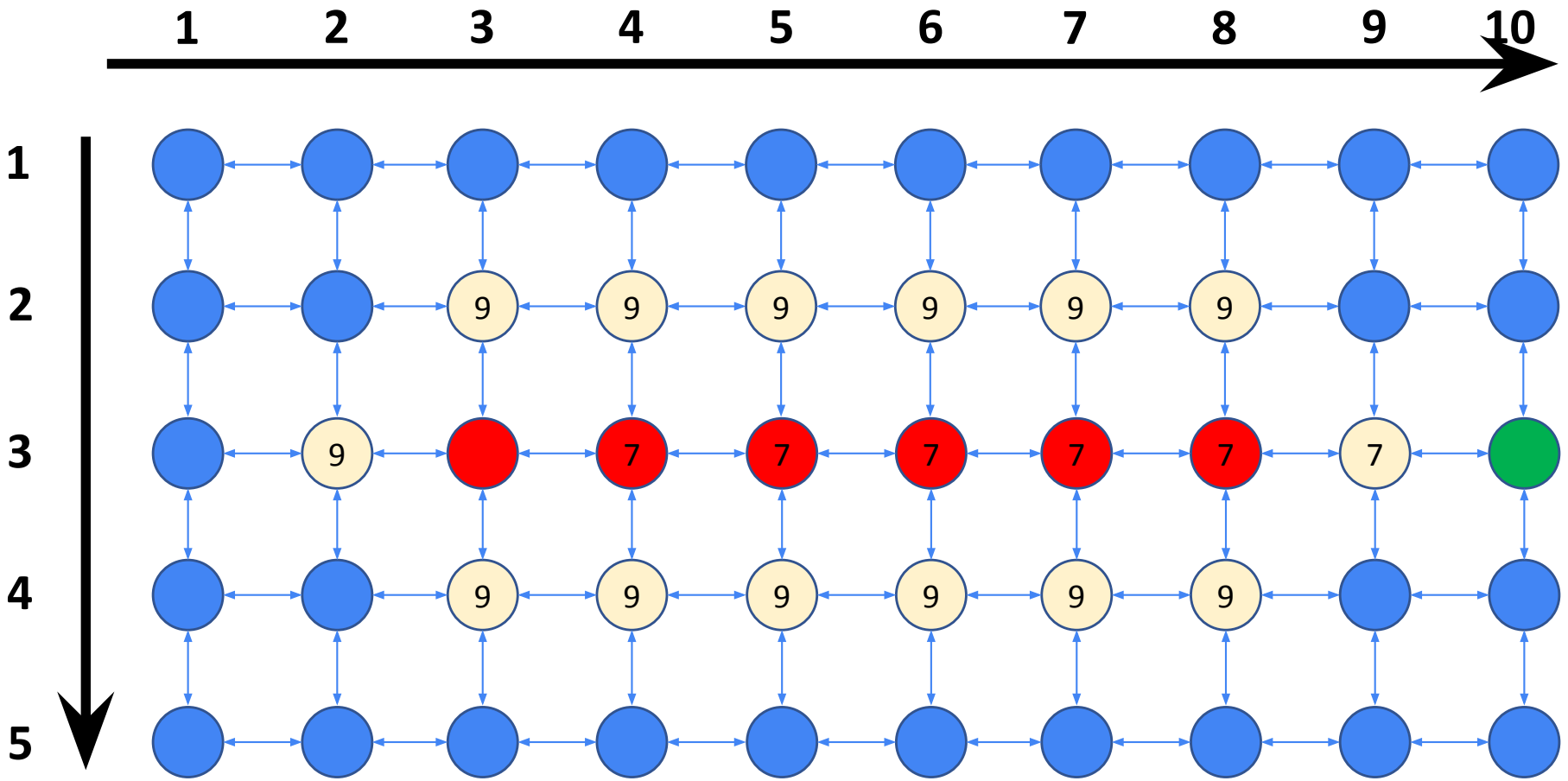
Using A* with Manhattan Distance



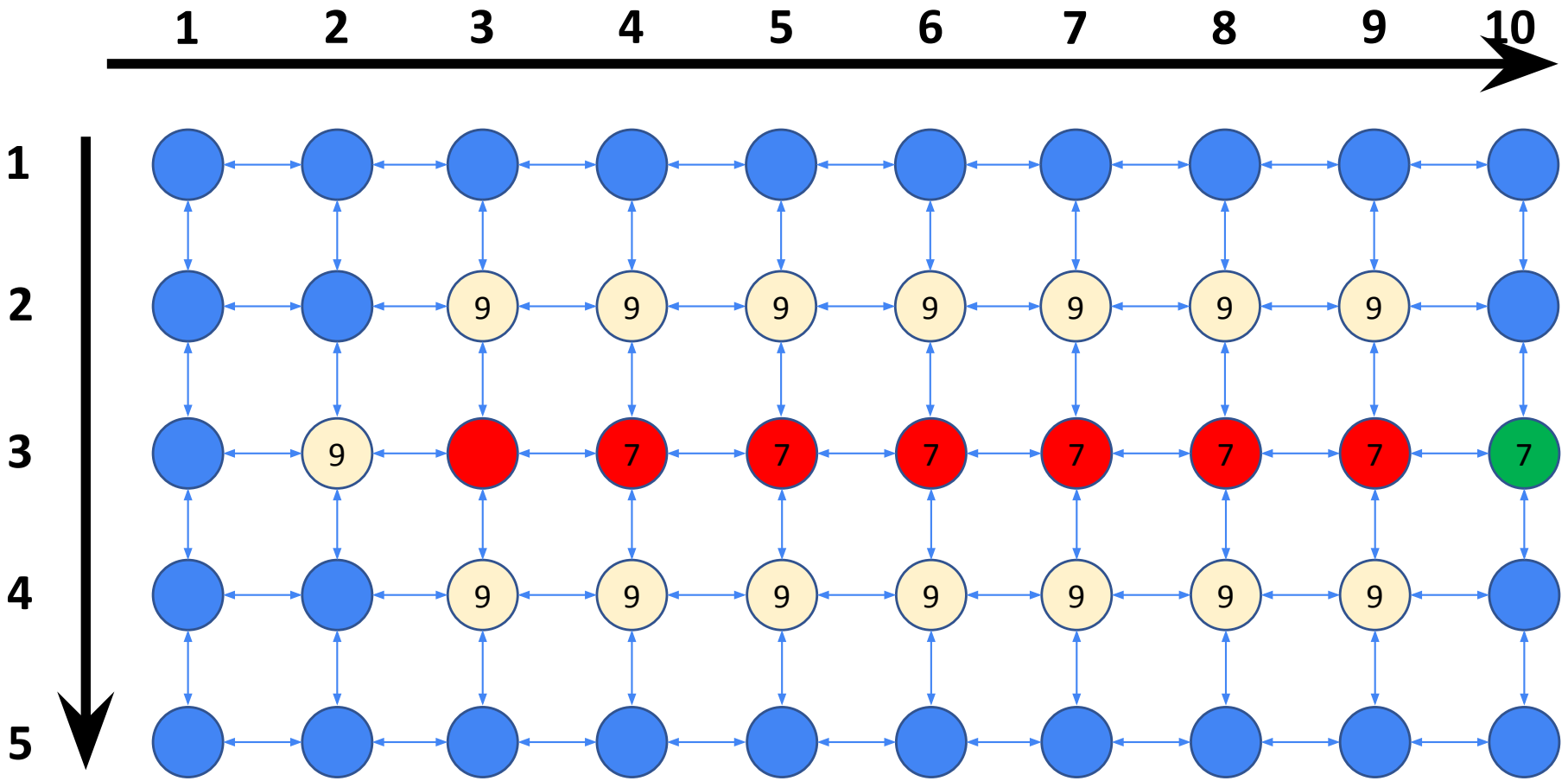
Using A* with Manhattan Distance



Using A* with Manhattan Distance



Using A* with Manhattan Distance



A* with priority queue

The priority queue will help in having the next best vertex always at the front.

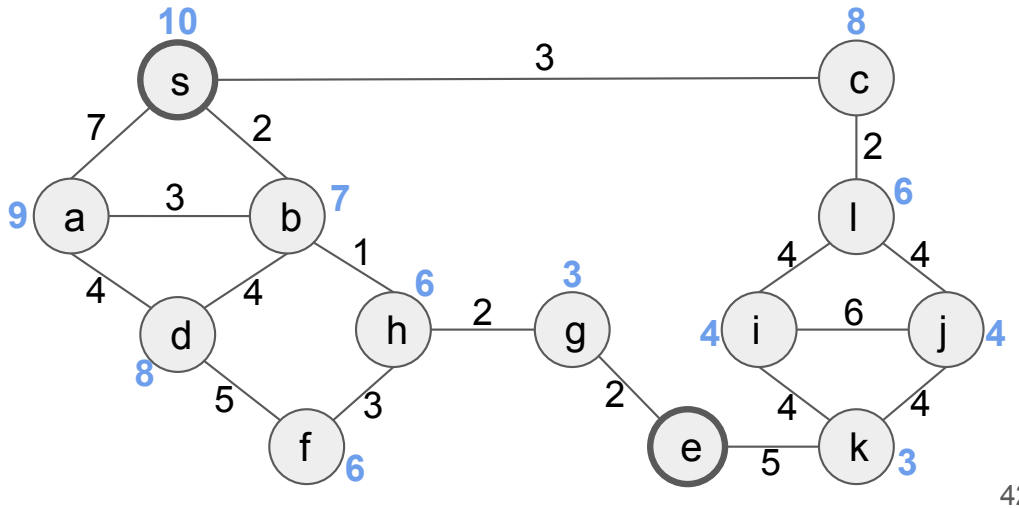
The numbers in blue are estimations of the actual distance between each node and the target node e.

s = start

e = end

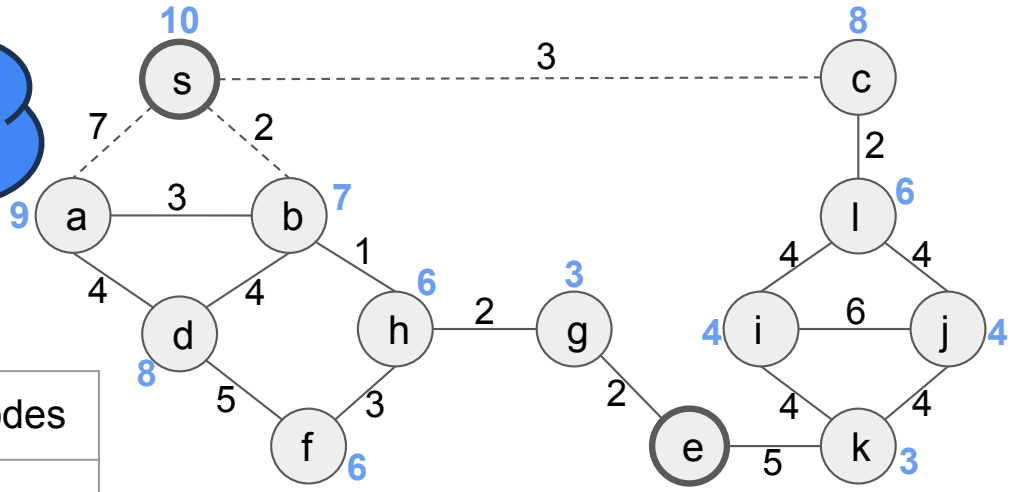
numbers at the top of the edges are the edge costs

len(s) = 0; len(all other) = ∞



Cost of the edge leading to vertex a (7)

Combined cost including the heuristic estimate of a (7+9)

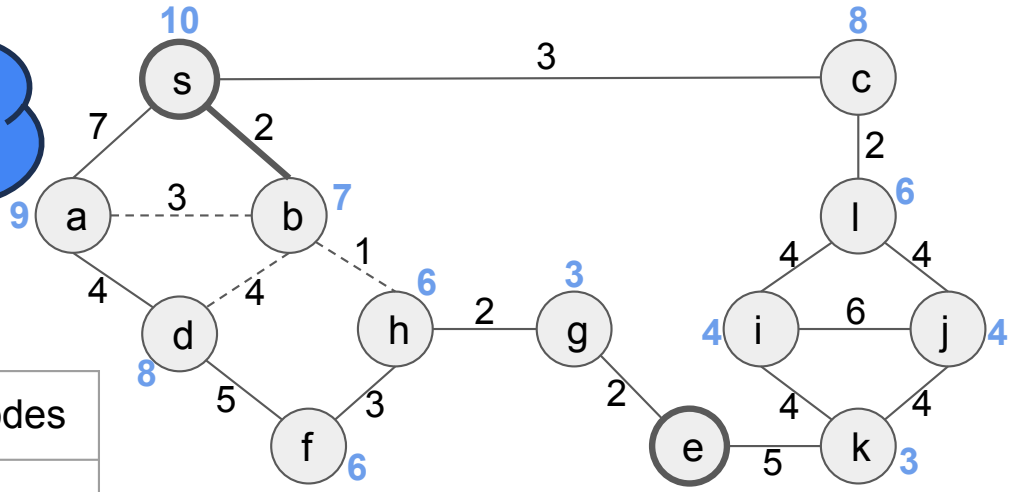


Priority queue	Cost calculation	Visited nodes
s; 0; 0+10 = 10 All other vertices with ∞	Expand s: a; 7; 16 b; 2; 9 c; 3; 11	s

All vertices with combined cost calculations

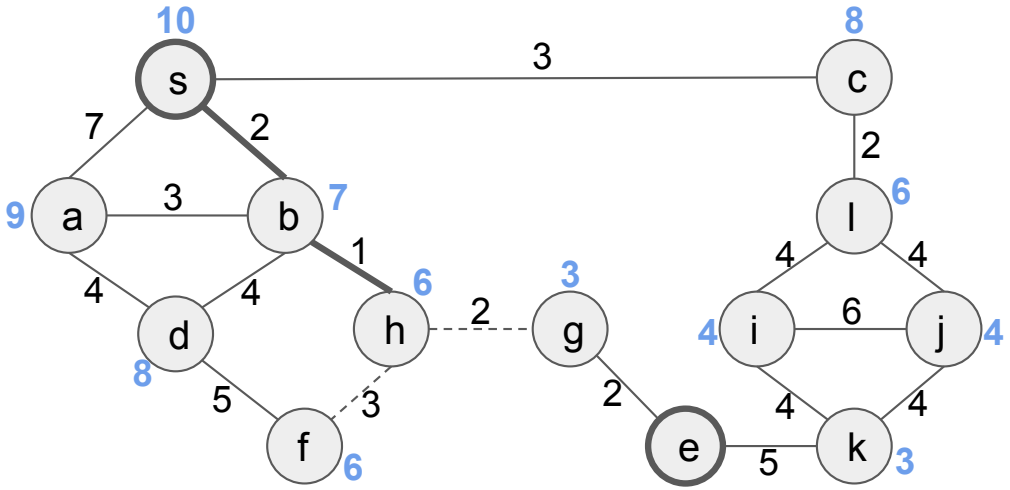
Cost of the edge leading to vertex a (7)

Combined cost including the heuristic estimate of a (7+9)

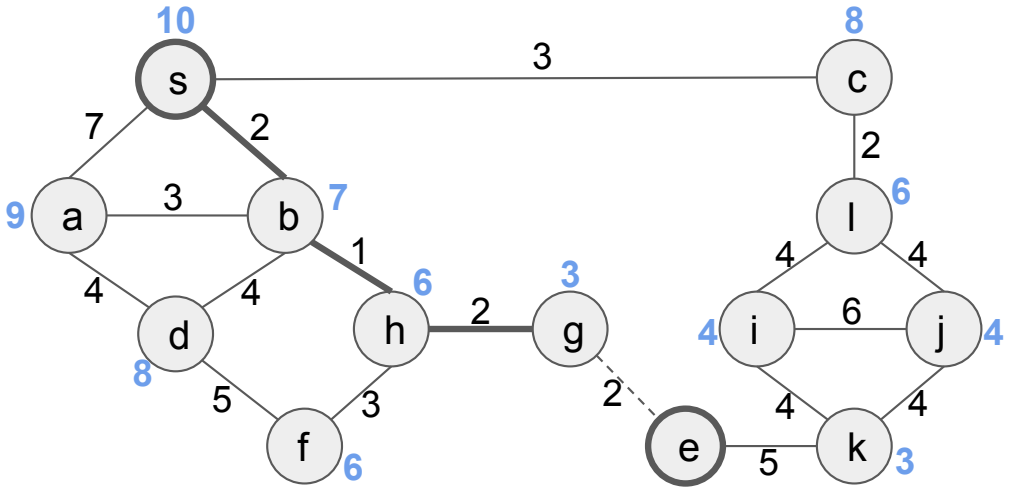


Priority queue	Cost calculation	Visited nodes
s; 0; 0+10 = 10 All other vertices with ∞	Expand s: a; 7; 16 b; 2; 9 c; 3; 11	s
b; 2; 9 c; 3; 11 a; 7; 16 Other vertices…	Expand b: a(via b); 5; 14 h(via b); 3; 9 d(via b); 6; 14	s, b

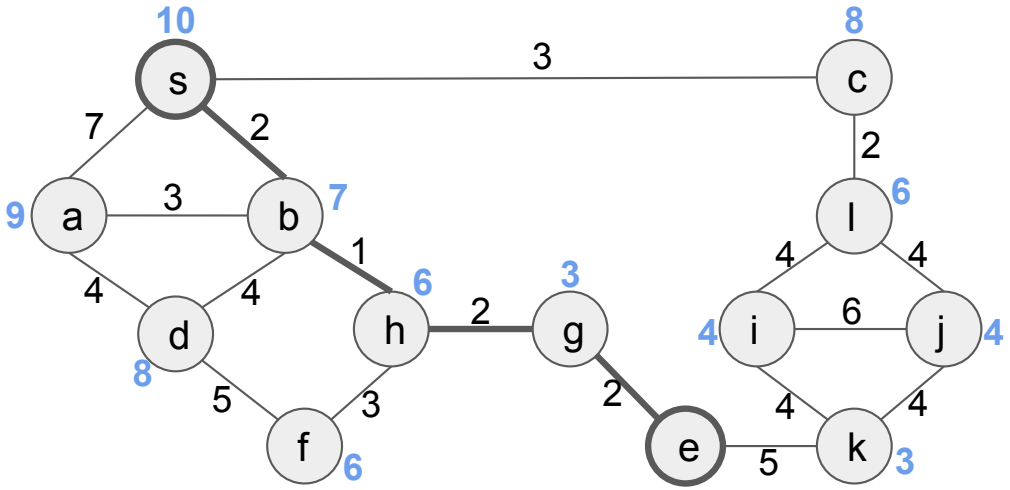
Priority queue	Cost calculation	Visited nodes
s; 0; 0+10 = 10 All other vertices with ∞	Expand s: a; 7; 16 b; 2; 9 c; 3; 11	s
b; 2; 9 c; 3; 11 a; 7; 16 Other vertices…	Expand b: a(via b); 5; 14 h(via b); 3; 9 d(via b); 6; 14	s, b
h(via b); 3; 9 c; 3; 11 a(via b); 5; 14 d(via b); 6; 14	Expand h: g; 5; 8 f; 6; 12	s, b, h



Priority queue	Cost calculation	Visited nodes
s; 0; 0+10 = 10 All other vertices with ∞	Expand s: a; 7; 16 b; 2; 9 c; 3; 11	s
b; 2; 9 c; 3; 11 a; 7; 16 Other vertices…	Expand b: a(via b); 5; 14 h(via b); 3; 9 d(via b); 6; 14	s, b
h(via b); 3; 9 c; 3; 11 a(via b); 5; 14 d(via b); 6; 14	Expand h: g; 5; 8 f; 6; 12	s, b, h
g; 5; 8 c; 3; 11 f; 6; 12 a(via b); 5; 14 d(via b); 6; 14	Expand g: e; 7; 7	s, b, h, g



Priority queue	Cost calculation	Visited nodes
s; 0; 0+10 = 10 All other vertices with ∞	Expand s: a; 7; 16 b; 2; 9 c; 3; 11	s
b; 2; 9 c; 3; 11 a; 7; 16 Other vertices…	Expand b: a(via b); 5; 14 h(via b); 3; 9 d(via b); 6; 14	s, b
h(via b); 3; 9 c; 3; 11 a(via b); 5; 14 d(via b); 6; 14	Expand h: g; 5; 8 f; 6; 12	s, b, h
g; 5; 8 c; 3; 11 f; 6; 12 a(via b); 5; 14 d(via b); 6; 14	Expand g: e; 7; 7	s, b, h, g
e; 7; 7 …	Expand e: e is the target	s, b, h, g, e return

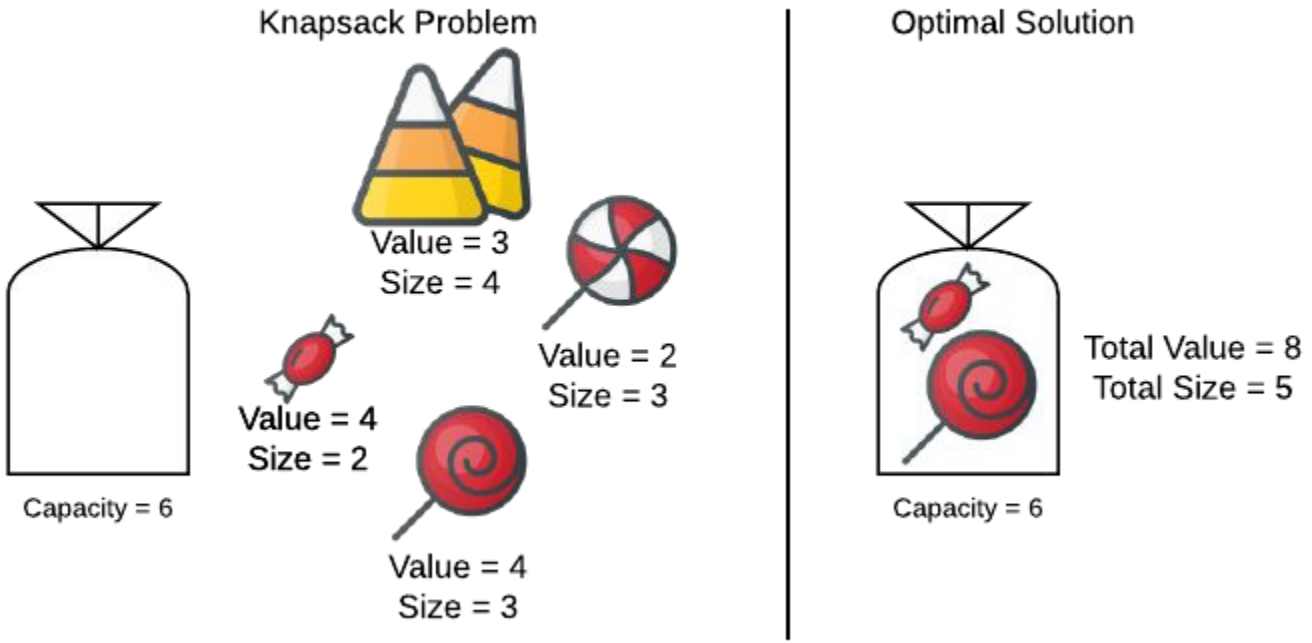


Correctness of A*

If the heuristic function is admissible (meaning that it never overestimates the actual cost to get to the goal), A* is guaranteed to return a least-cost path from start to goal.

The 0/1 Knapsack problem

Given a knapsack of a certain capacity, how to select the items in such a way that we maximize the total value and not exceed the capacity?



The 0/1 Knapsack problem

Problem: Knapsack

Input: Item values $v_1, v_2, \ldots, v_n$, item sizes $s_1, s_2, \ldots, s_n$, and a knapsack capacity C. (All positive integers.)

Output: A subset $S \subseteq \{1, 2, \ldots, n\}$ of items with the maximum-possible sum $\sum_{i \in S} v_i$ of values, subject to having total size $\sum_{i \in S} s_i$ at most C.

The 0/1 Knapsack problem

First, we create a table of n + 1 rows and w + 1 columns (n is the number of items and w is the weight units).

A row number i represents the set of all the items from rows 1— i. For instance, the values in row 3 assumes that we only have items 1, 2, and 3.

A column number j represents the weight capacity of our knapsack. Therefore, the values in column 5, for example, assumes that our knapsack can hold 5 weight units.

Max capacity = 7

Knapsack weight units

items

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0							
2	2	1	0							
3	4	3	0							
4	5	4	0							
5	3	2	0							

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0							
2	2	1	0							
3	4	3	0							
4	5	4	0							
5	3	2	0							

Item 1, when capacity is 1.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0						
2	2	1	0							
3	4	3	0							
4	5	4	0							
5	3	2	0							

Item 1, when capacity is 1.
Item 1 doesn't fit.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0					
2	2	1	0							
3	4	3	0							
4	5	4	0							
5	3	2	0							

Item 1, when capacity is 2.
Item 1 doesn't fit.

Max capacity = 7

item	value	weight	0	1	2	**3**	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	**3**	0	0	0	2				
2	2	1	0							
3	4	3	0							
4	5	4	0							
5	3	2	0							

Item 1, when capacity is 3.
Item 1 fits!

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0							
3	4	3	0							
4	5	4	0							
5	3	2	0							

Since item 1 is the only item at the moment,
it will fit in the other capacities.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2						
3	4	3	0							
4	5	4	0							
5	3	2	0							

Item 2, when capacity is 1.
Item 2 fits!

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2					
3	4	3	0							
4	5	4	0							
5	3	2	0							

Item 2, when capacity is 2.
Item 2 fits!

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2				
3	4	3	0							
4	5	4	0							
5	3	2	0							

Item 2, when capacity is 3.
There is no space for both items.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	**2**	3	0	0	0	2	2	2	2	2
2	**2**	1	0	2	2	2	4			
3	4	3	0							
4	5	4	0							
5	3	2	0							

Item 2, when capacity is 4.
The maximum value is 2 + 2, because there is space for both items!

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	**2**	3	0	0	0	2	2	2	2	2
2	**2**	1	0	2	2	2	4	4	4	4
3	4	3	0							
4	5	4	0							
5	3	2	0							

The maximum value is 2 + 2, because there is space for both items!

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2						
4	5	4	0							
5	3	2	0							

Item 3, when capacity is 1.
Only item 2 fits, thus repeat previous value.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2					
4	5	4	0							
5	3	2	0							

Item 3, when capacity is 2.
Only item 2 fits, thus repeat previous value.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4				
4	5	4	0							
5	3	2	0							

Item 3, when capacity is 3.
The maximum value is 4, which is the value of item 3.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6			
4	5	4	0							
5	3	2	0							

Item 3, when capacity is 4.
The maximum value is 4 (item 3) + 2 (item 2) = 6.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	
4	5	4	0							
5	3	2	0							

The maximum value is 4 (item 3) + 2 (item 2) = 6.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0							
5	3	2	0							

Item 3, when capacity is 7.
All items fit (weights 3 + 1 + 3 = 7).
The maximum value is 4 (item 3) + 2 (item 2) + 2 (item 1) = 8.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2						
5	3	2	0							

Item 4, when capacity is 1.
Item 4 doesn't fit, thus repeat previous value.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4				
5	3	2	0							

Item 4 doesn't fit, thus repeat previous value.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6			
5	3	2	0							

Item 4, when capacity is 4.
The maximum value is 4 (item 3) + 2 (item 2) = 6.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7		
5	3	2	0							

Item 4, when capacity is 5.
The maximum value is 5 (item 4) + 2 (item 2) = 7.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7	7	
5	3	2	0							

Item 4, when capacity is 6.
The maximum value is 5 (item 4) + 2 (item 2) = 7.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7	7	9
5	3	2	0							

Item 4, when capacity is 7.
The maximum value is 5 (item 4) + 4 (item 3) = 9.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7	7	9
5	3	2	0	2						

Item 5, when capacity is 1.
Item 5 doesn't fit, thus repeat previous value.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7	7	9
5	3	2	0	2	3					

Item 5, when capacity is 2.
The maximum value is 3 (item 5).

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7	7	9
5	3	2	0	2	3	5				

Item 5, when capacity is 3.
The maximum value is 3 (item 5) + 2 (item 2) = 5.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7	7	9
5	3	2	0	2	3	5	6			

Item 5, when capacity is 4.
The maximum value is 4 (item 3) + 2 (item 2) = 6.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7	7	9
5	3	2	0	2	3	5	6	7		

Item 5, when capacity is 5.
The maximum value is 3 (item 5) + 4 (item 3) = 7.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7	7	9
5	3	2	0	2	3	5	6	7	9	

Item 5, when capacity is 6.
The maximum value is 3 (item 5) + 4 (item 3) + 2 (item 2) = 9.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7	7	9
5	3	2	0	2	3	5	6	7	9	10

Item 5, when capacity is 7.
The maximum value is 3 (item 5) + 5 (item 4) + 2 (item 2) = 10.

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7	7	9
5	3	2	0	2	3	5	6	7	9	10

How do we know which items to include, at the end?

Max capacity = 7

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7	7	9
5	3	2	0	2	3	5	6	7	9	10

Start from the bottom right up to the top left.
10≠9, so include item 5 (because the value is greater).
Shift by weight of item 5 (2) to left.

Max capacity = 7 (item included: 5)

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7	7	9
5	3	2	0	2	3	5	6	7	9	10

Start from the bottom right up to the top left.
7≠6, so include item 4 (because the value is greater).
Shift by weight of item 4 (4) to left.

Max capacity = 7 (item included: 5, 4)

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7	7	9
5	3	2	0	2	3	5	6	7	9	10

Start from the bottom right up to the top left.
2=2, so skip item 3 and select item 2 (why?)
Shift by weight of item 2 (1) to left.

Max capacity = 7 (item included: 5, 4, 2)

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	**2**	**1**	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	**5**	**4**	0	2	2	4	6	7	7	9
5	**3**	**2**	0	2	3	5	6	7	9	10

No item to be added. The process stop.
Total value: 2+5+3=10
Total weight: 1+4+2=7

Generalization of executed steps

item	value	weight	0	1	2	3	4	5	6	7
			0	0	0	0	0	0	0	0
1	2	3	0	0	0	2	2	2	2	2
2	2	1	0	2	2	2	4	4	4	4
3	4	3	0	2	2	4	6	6	6	8
4	5	4	0	2	2	4	6	7	7	9
5	3	2	0	2	3	5	6	7	9	10

$$ITEMS[i,c] = \max\{ITEMS[i-1,c], ITEMS[i-1,c-w[i]] + v[i]\}$$

Case 1: when "we **do not** include current item"

Case 2: when "we include current item"

Dynamic programming

At each step, we make use of our solutions to previous sub-problems.

Solution to the 0/1 Knapsack problem complexity

- A naive approach would be to list all possible combinations. If order doesn't matter, then 2^n.

- Using **dynamic programming**, for every instance of the Knapsack problem, the Knapsack algorithm returns the total value of an optimal solution and runs in time O(n*w), where n is the number of items and w is the Knapsack capacity.