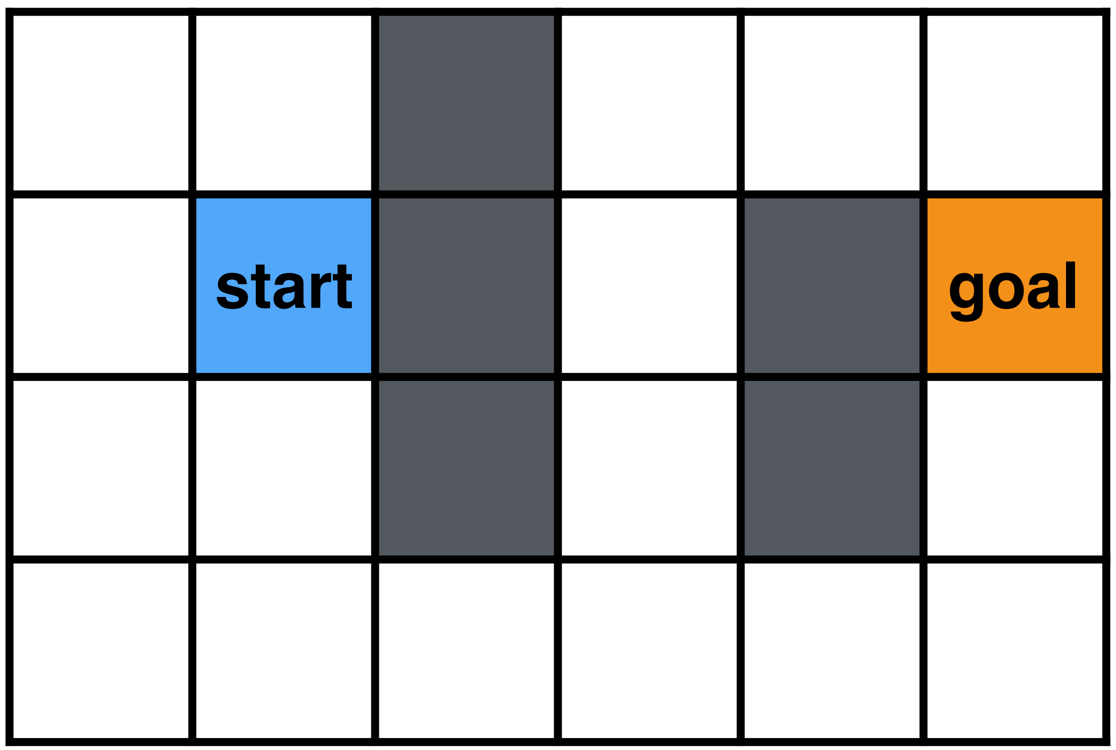
Robot Autonomy

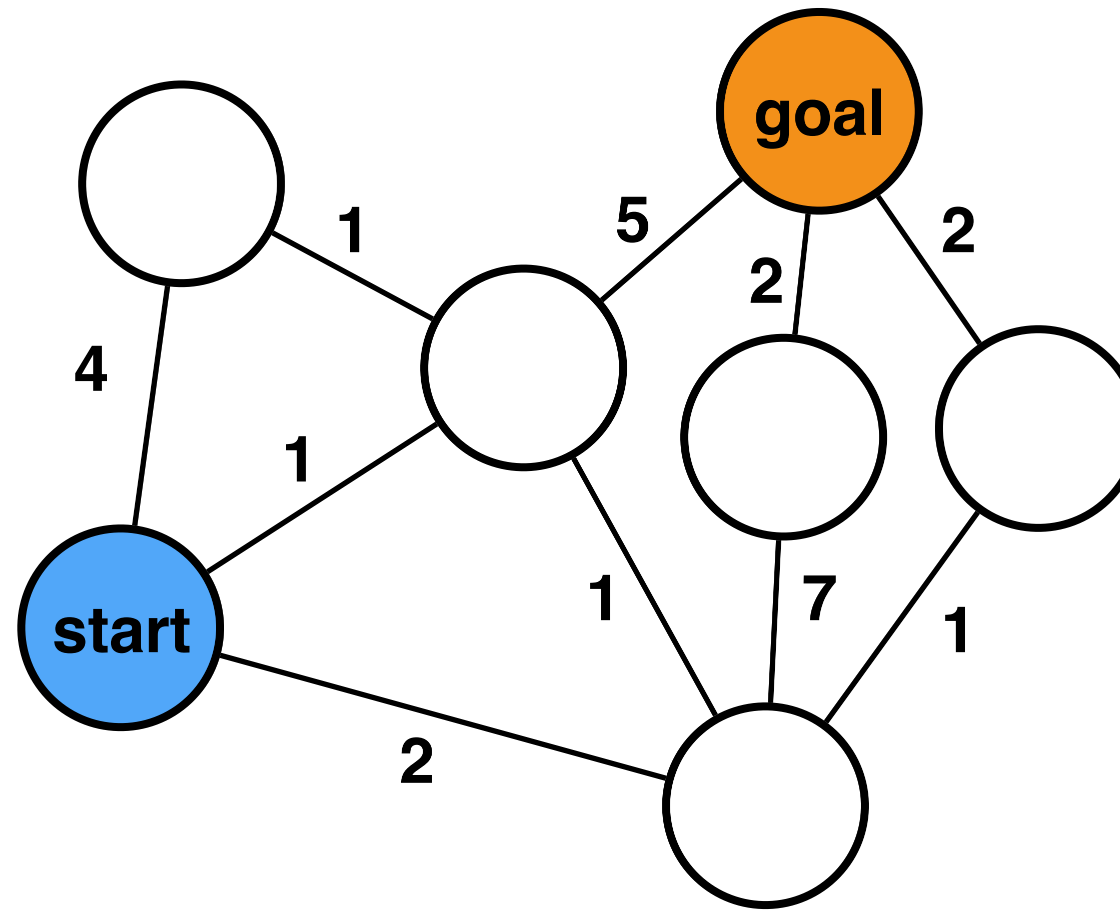
Lecture 09: Discrete Search

Oliver Kroemer

Motivation

- Discretized motion planning problems





  - Edges may have uniform or different distances/costs
- Want to find a (short/cheap) path from start to goal state

Discrete Piano Mover's Problem

- Discrete state

$$x \in X$$

- Discrete actions

$$u \in U(x)$$

- State transition function

$$f(x, u) = x'$$

  - traverse edges of graph or move to neighbouring grid cell

Forward Search with Priority Queue

Priority queue Q = empty

Q.insert(x_i)   mark x_i as visited

while Q is not empty

    $x \leftarrow Q$.GetFirst()

    if $x \in X_{goal}$

        return Success

    for all $u \in U(x)$

        $x' \leftarrow f(x, u)$

        if x' not visited

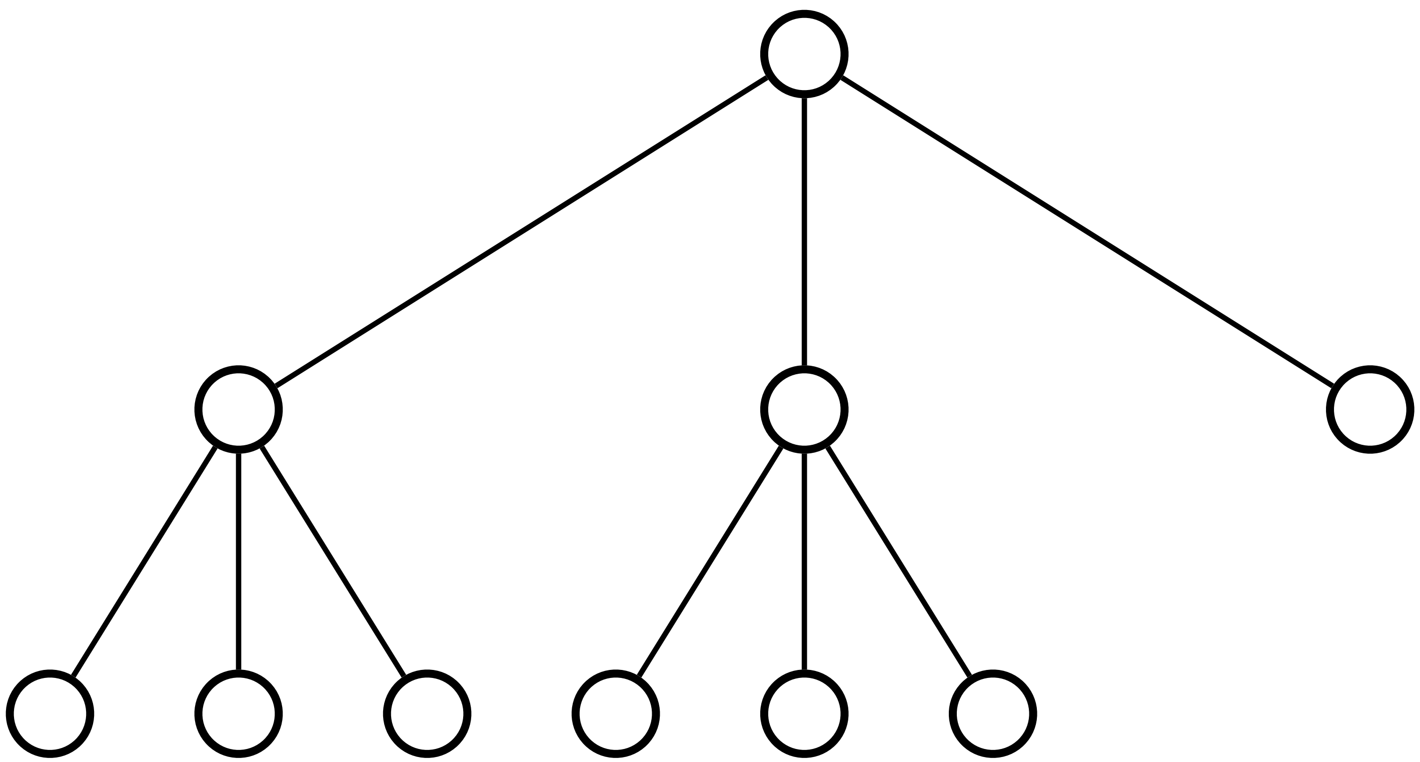
            mark x' as visited

            Q.insert(x')

return Failure

Breadth-First Search

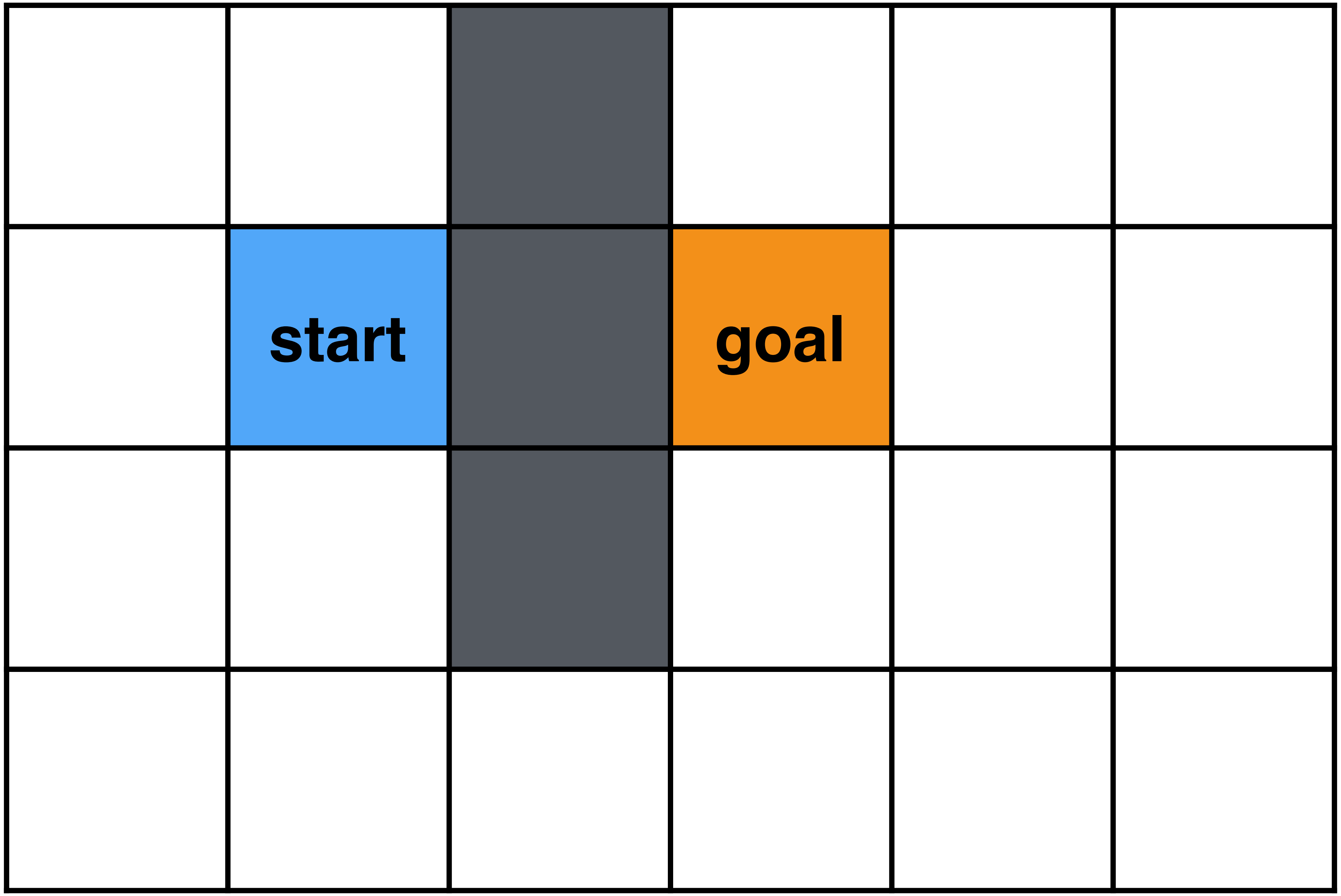
- Breadth-first search of states
- Expand states according to first-in-first-out queue



- Will find shortest path to goal
- Will expand to every state closer to the start

Example Grid Problem

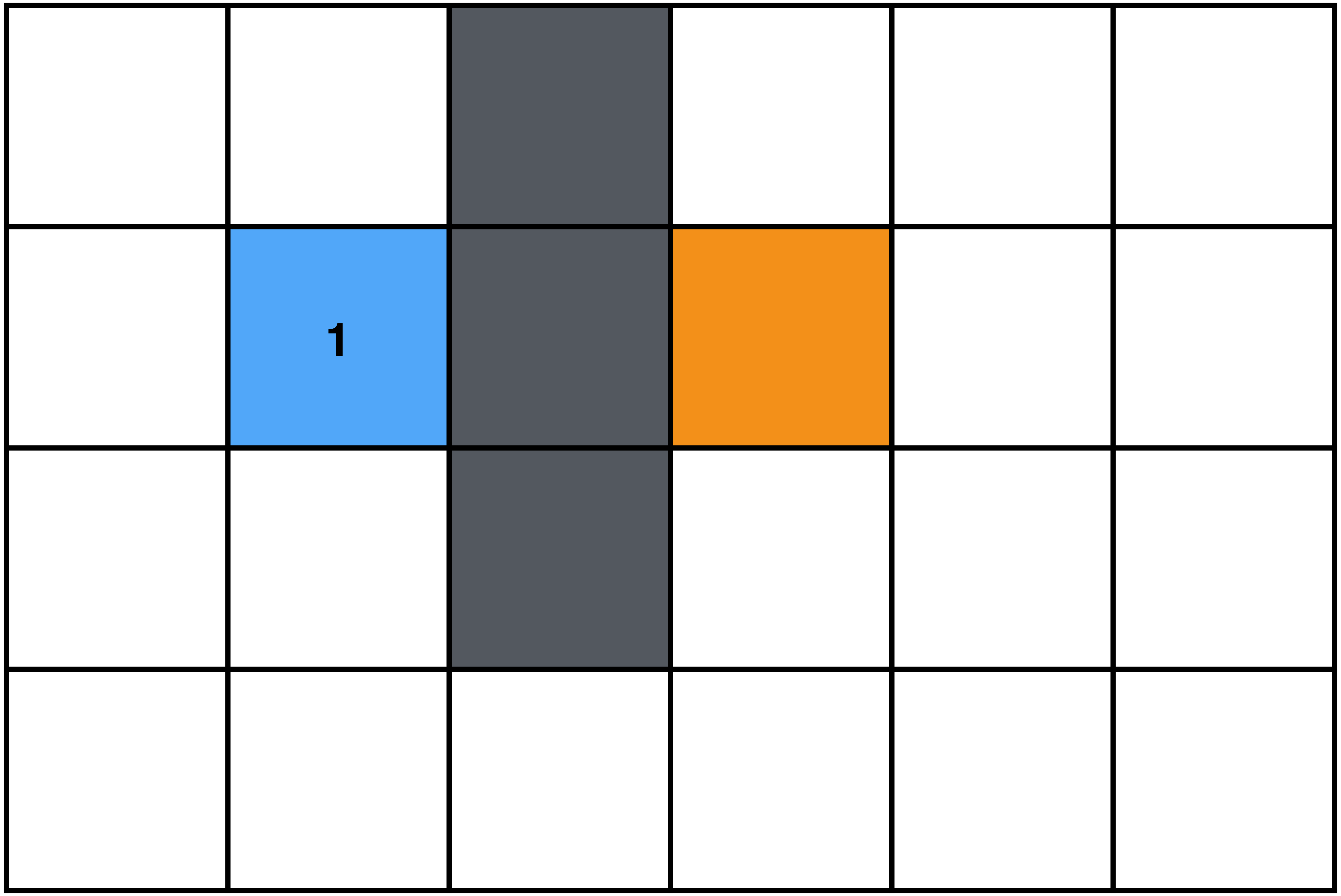
- Find start-to-goal path using down, left, right, up actions



- Compute obstacles online

Breadth-First Search

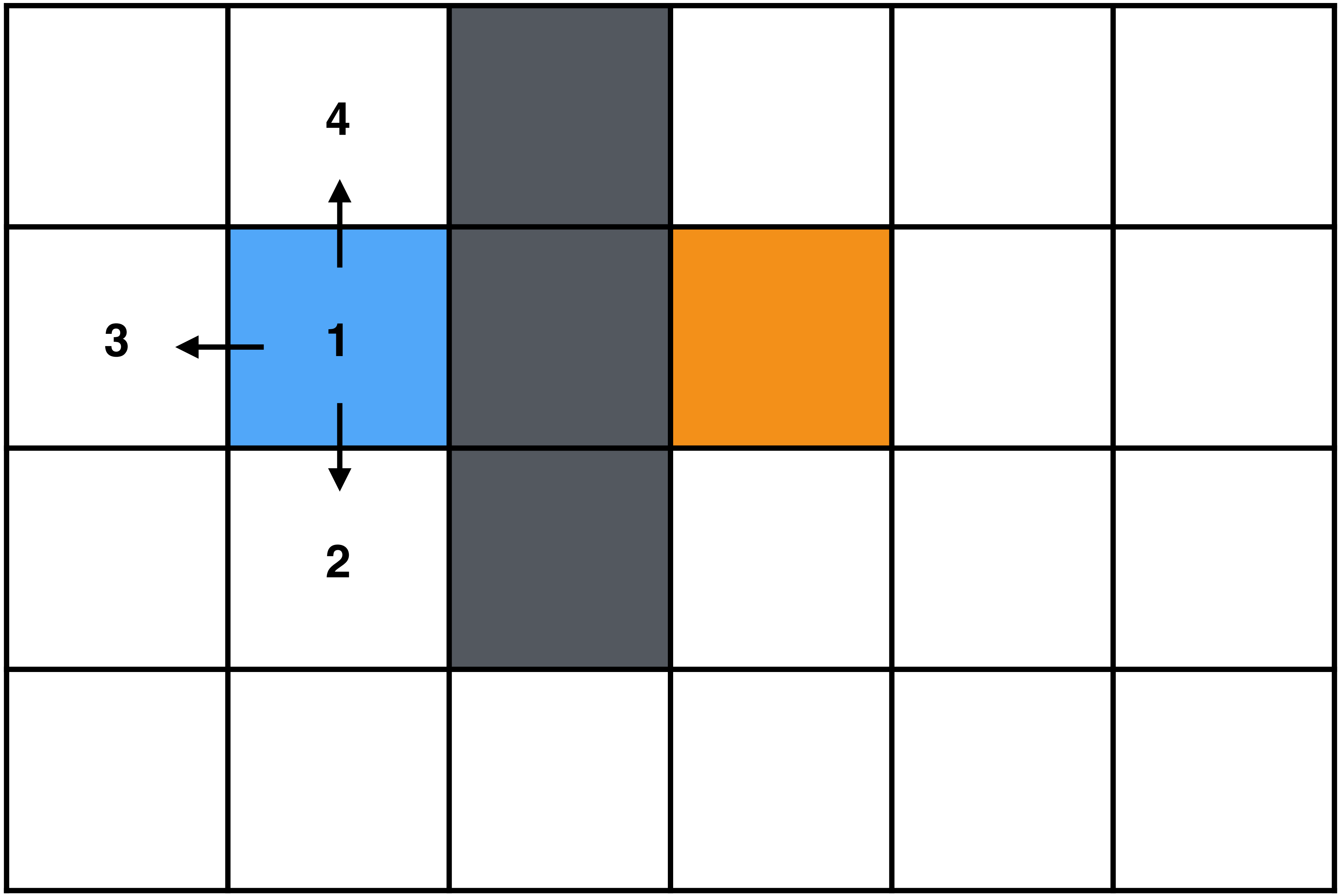
- Start at the start cell



- Q=[1]

Breadth-First Search

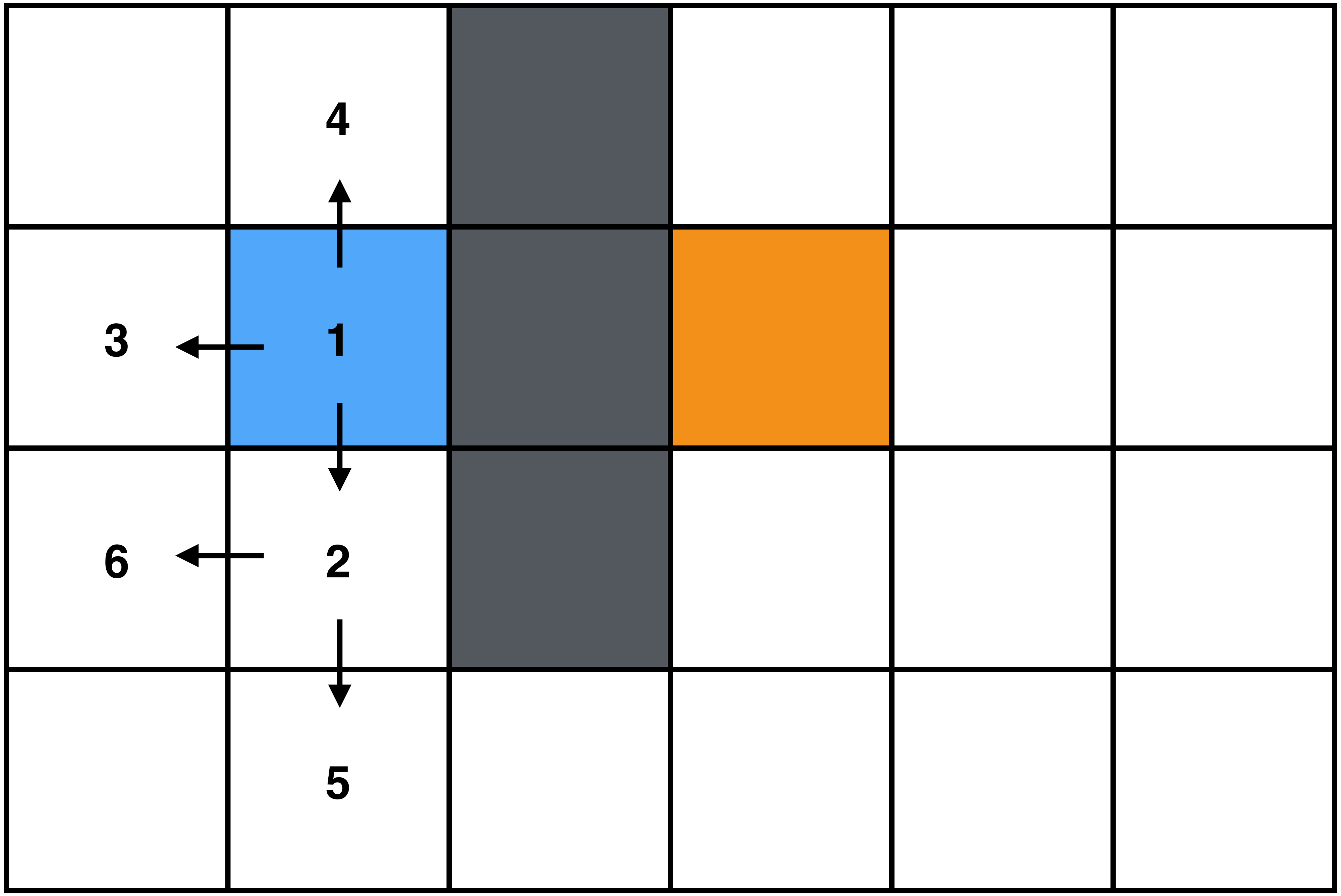
- Expand neighbours right ,down, left, and, up



- Q=[1,2,3,4]

Breadth-First Search

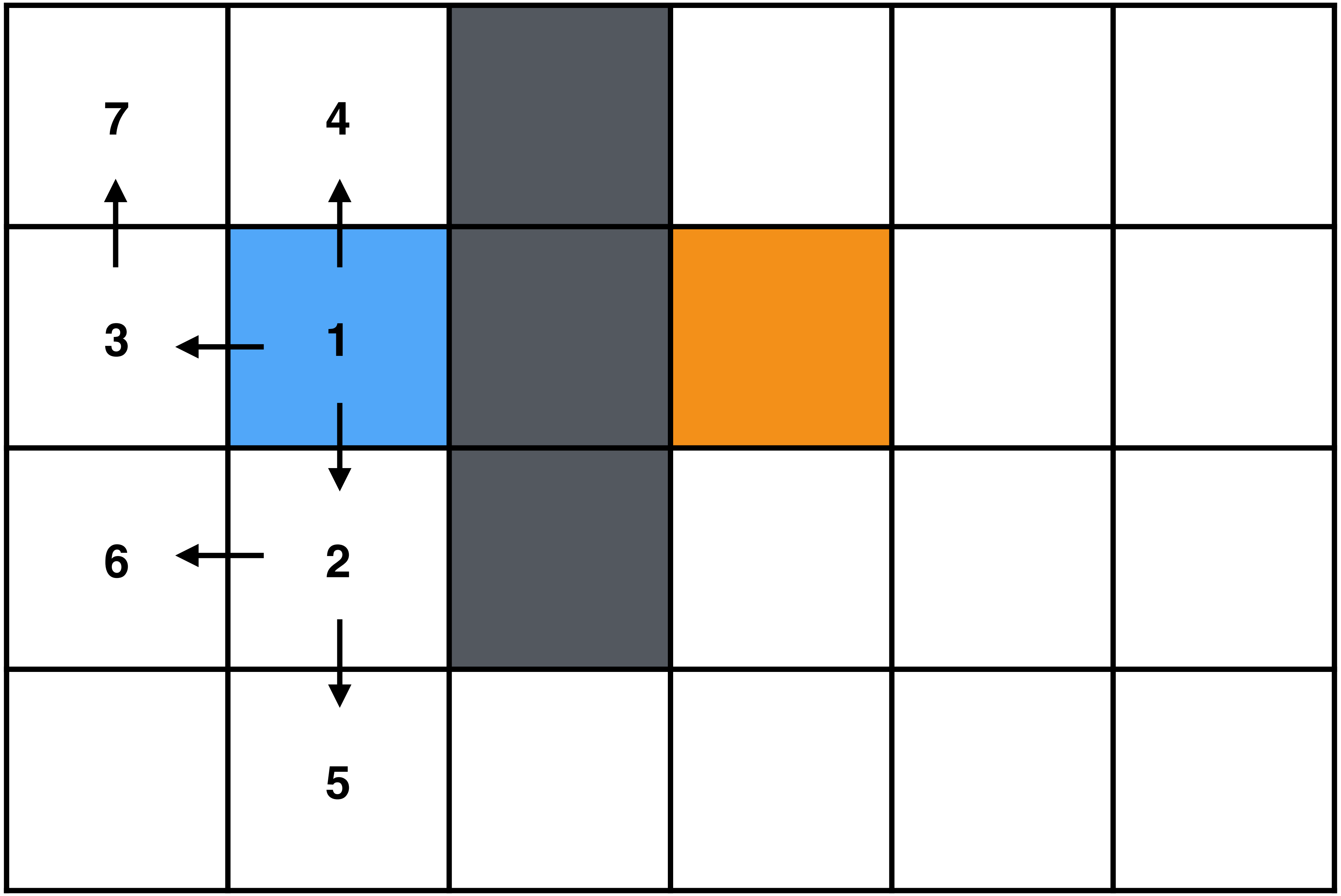
- Expand neighbours right ,down, left, and, up



- Q=[2,3,4,5,6]

Breadth-First Search

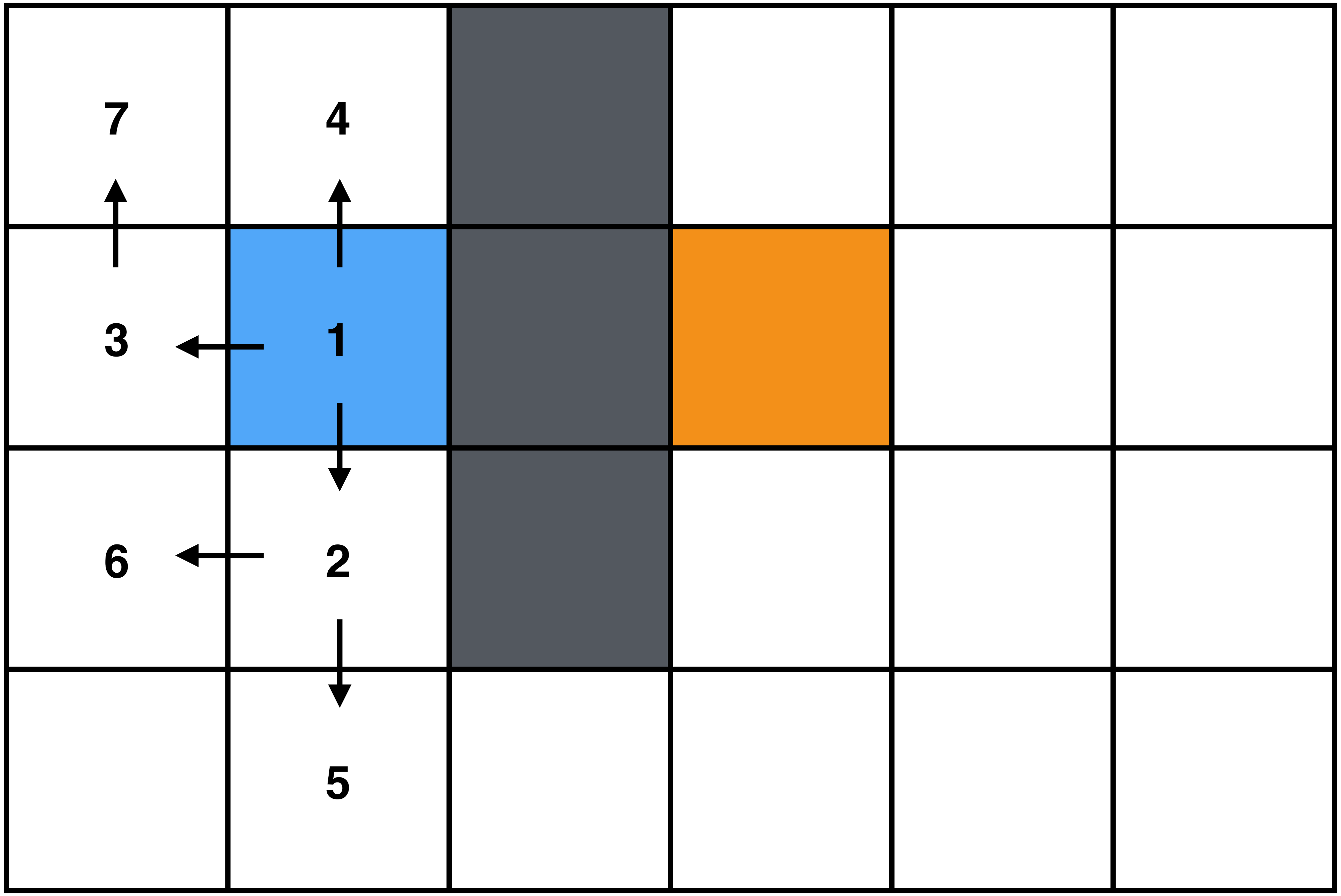
- Expand neighbours right ,down, left, and, up



- Q=[3,4,5,6,7]

Breadth-First Search

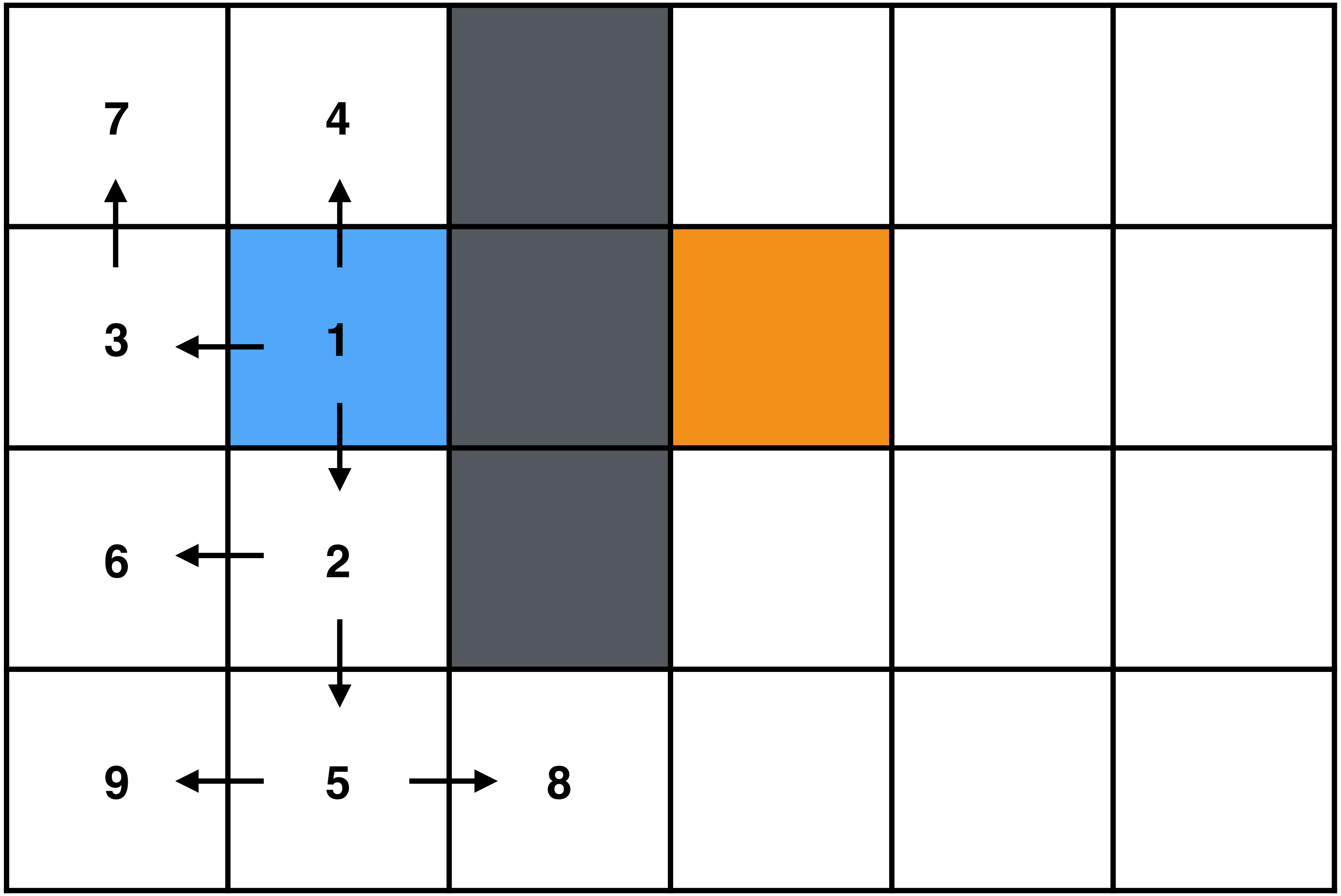
- Expand neighbours right ,down, left, and, up



- Q=[4,5,6,7]

Breadth-First Search

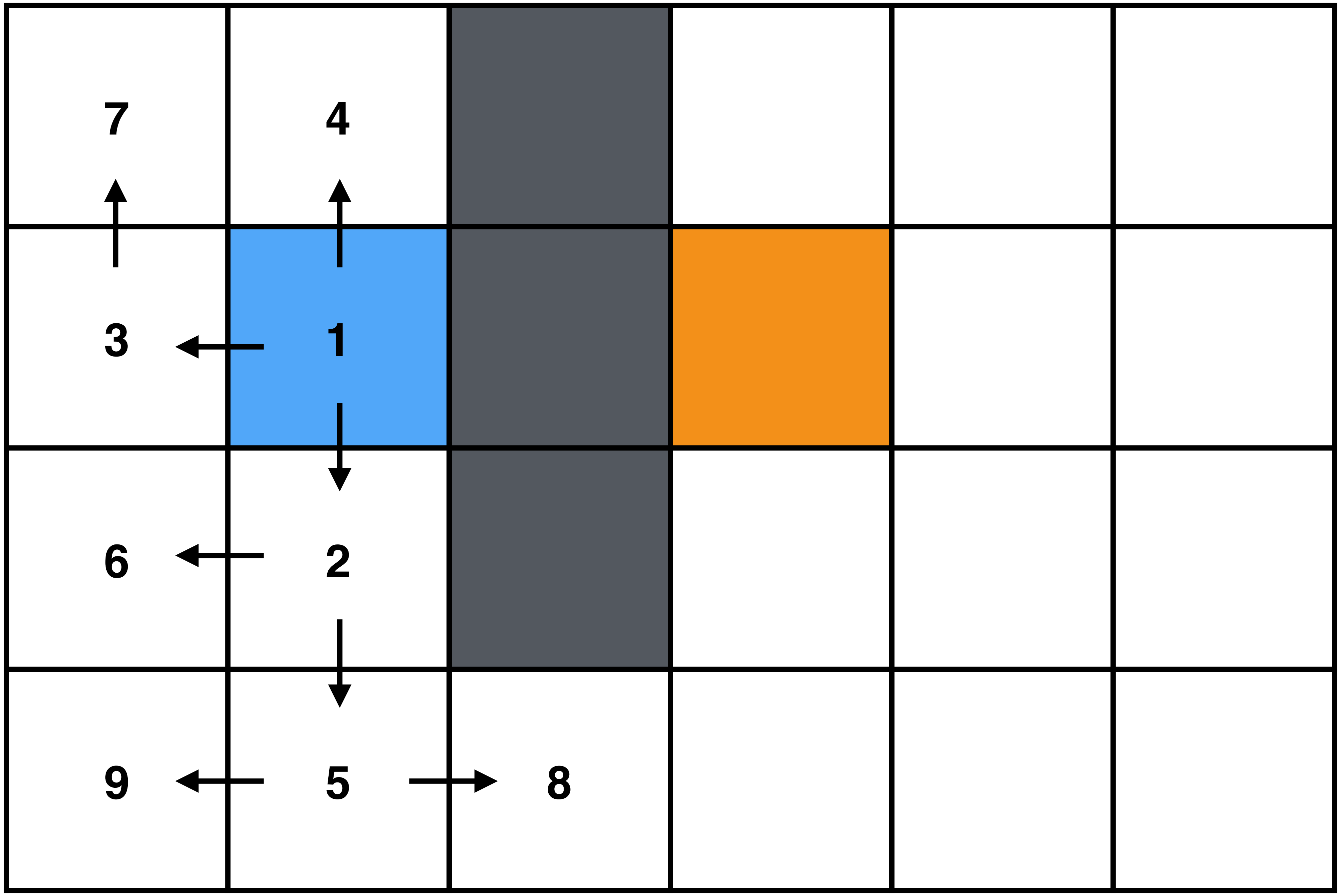
- Expand neighbours right ,down, left, and, up



- Q=[5,6,7,8,9]

Breadth-First Search

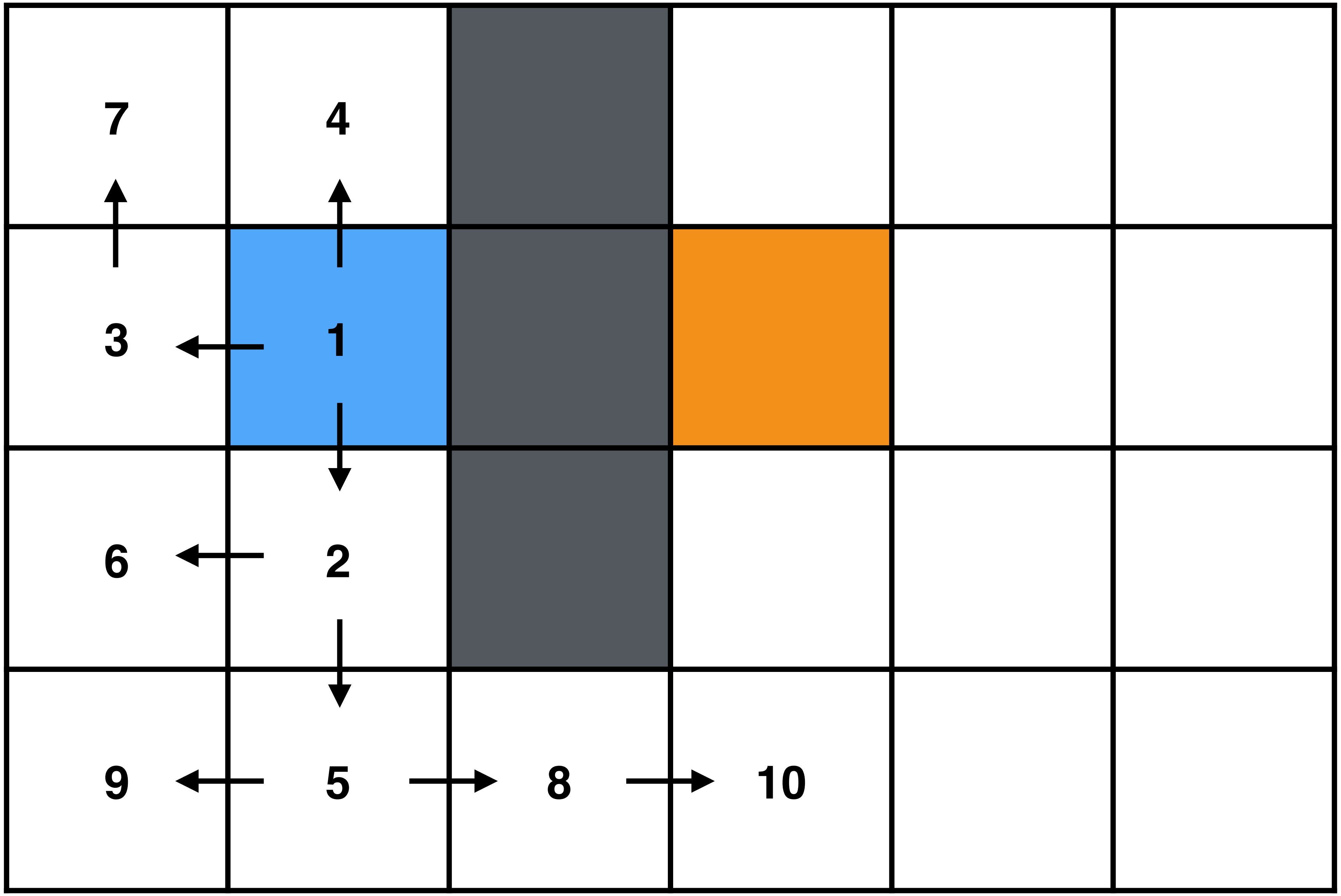
- Expand neighbours right ,down, left, and, up



- Q=[6,7,8,9]

Breadth-First Search

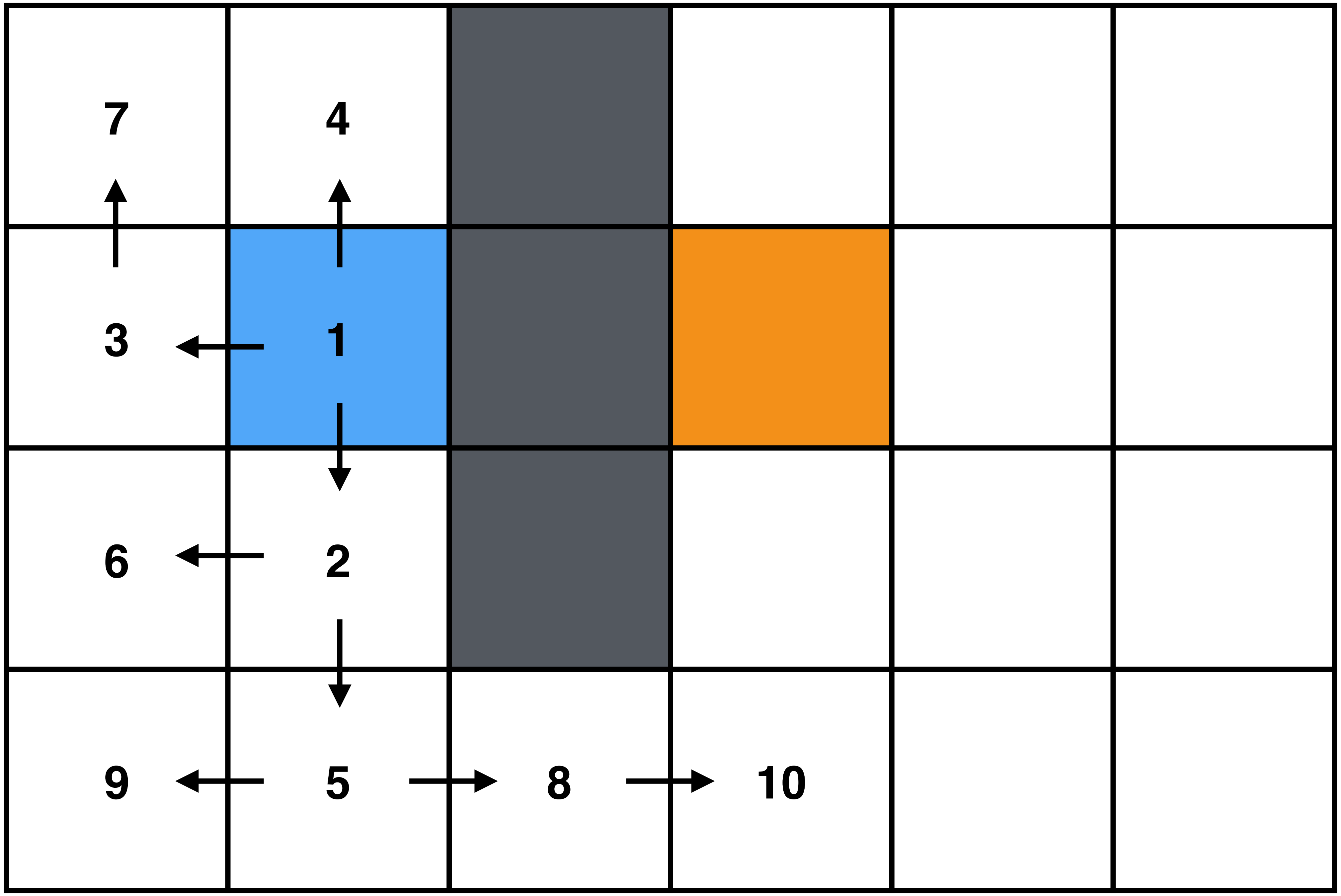
- Expand neighbours right ,down, left, and, up



- Q=[8,9,10]

Breadth-First Search

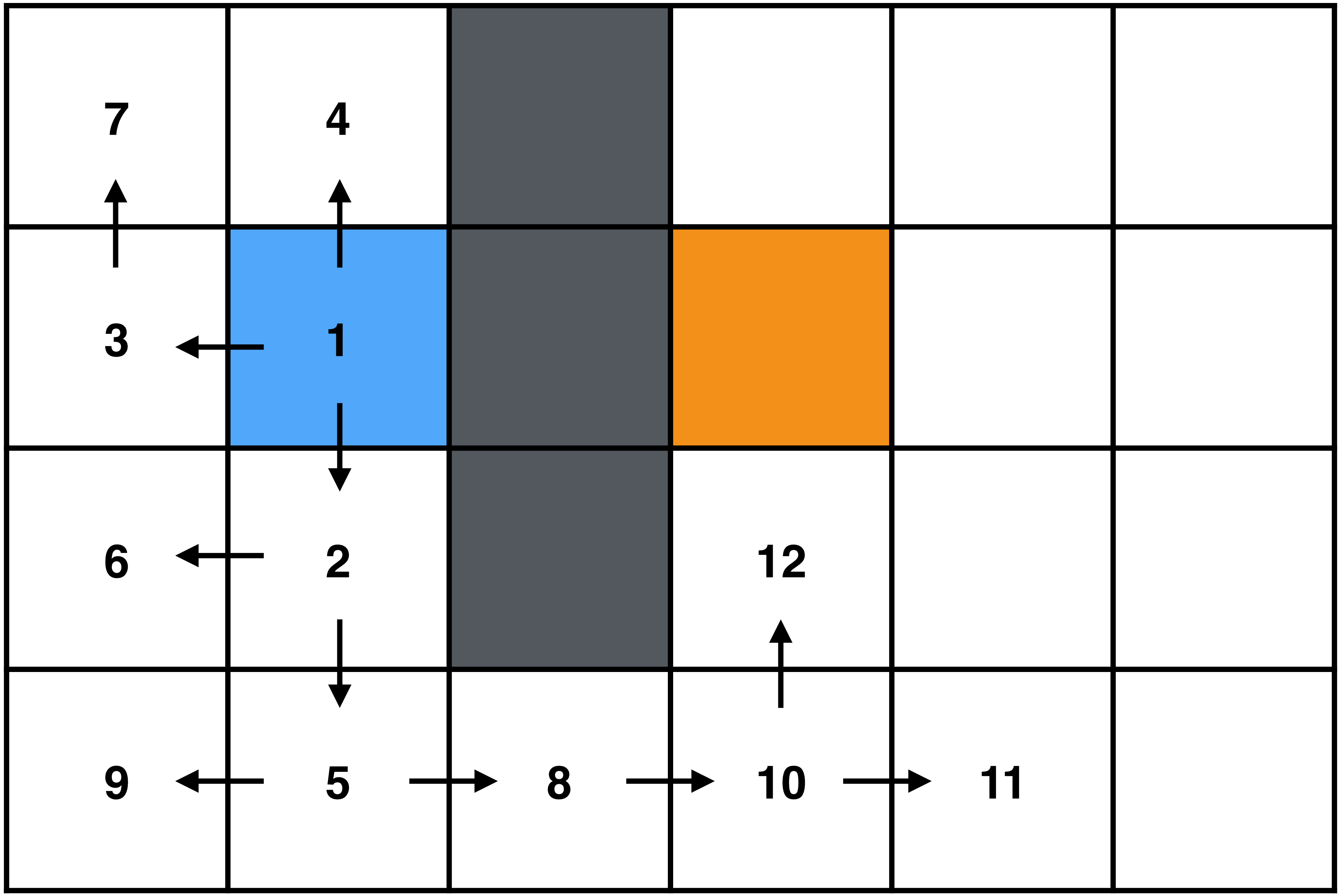
- Expand neighbours right ,down, left, and, up



- Q=[9,10]

Breadth-First Search

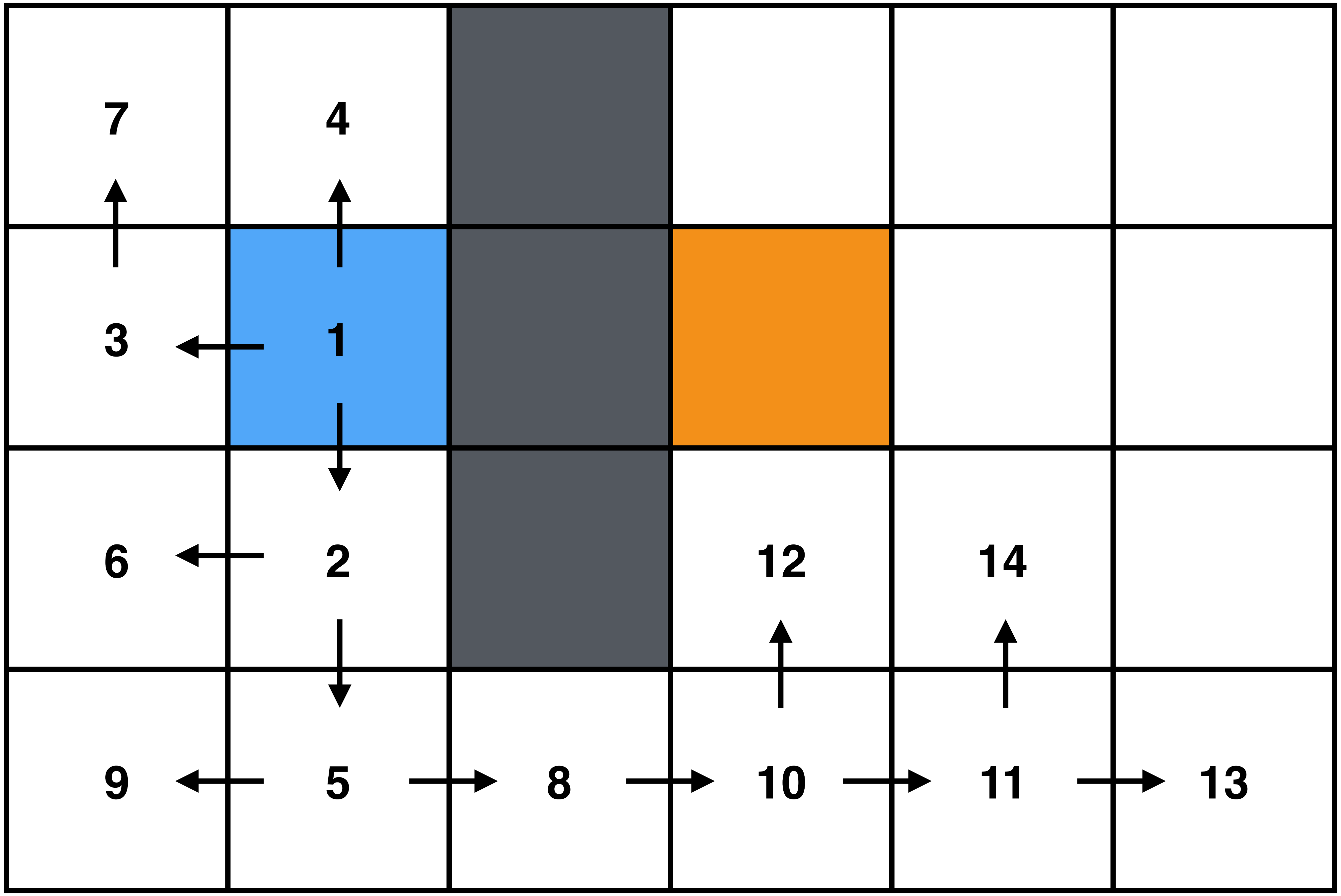
- Expand neighbours right ,down, left, and, up



- Q=[10,11,12]

Breadth-First Search

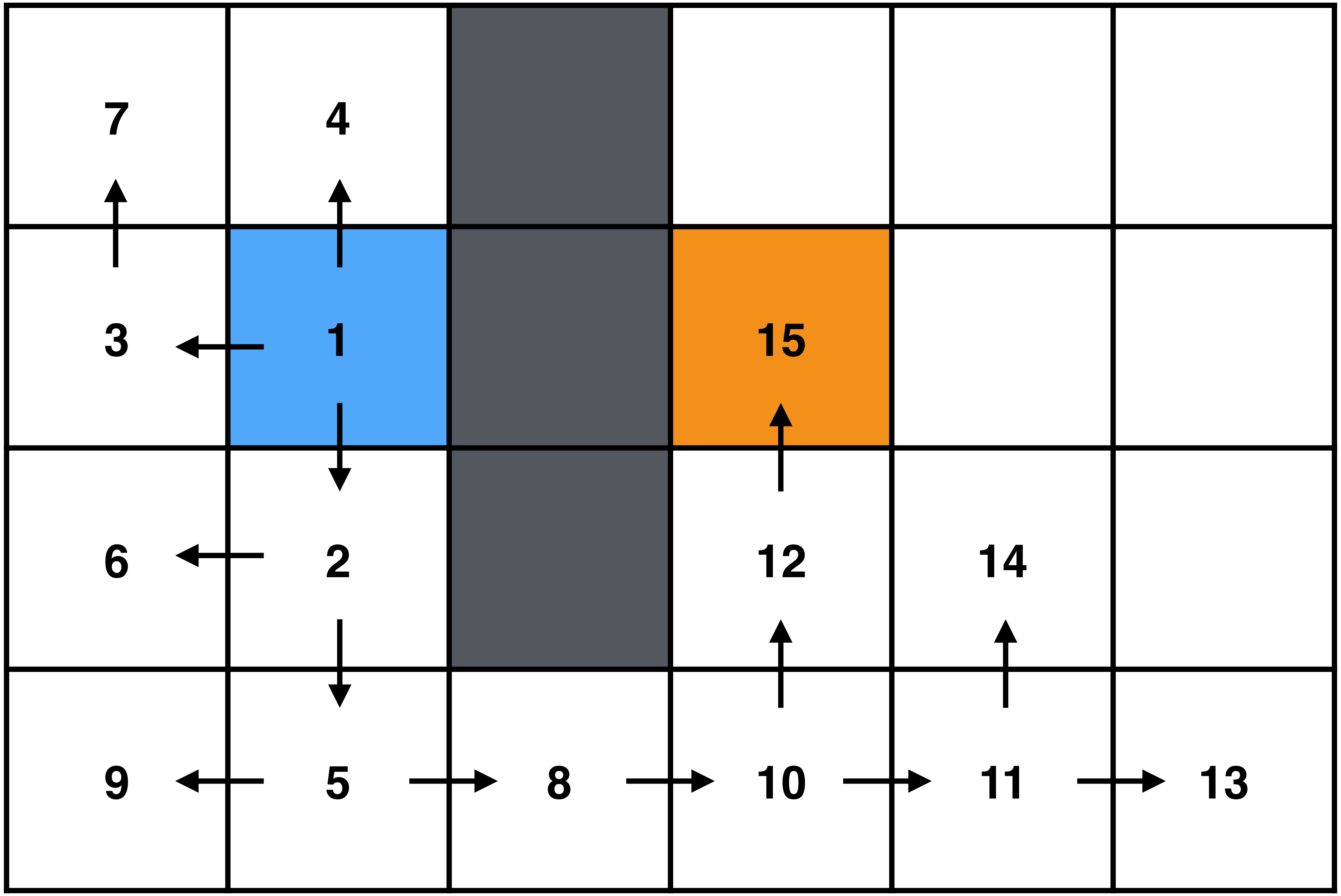
- Expand neighbours right ,down, left, and, up



- Q=[11,12,13,14]

Breadth-First Search

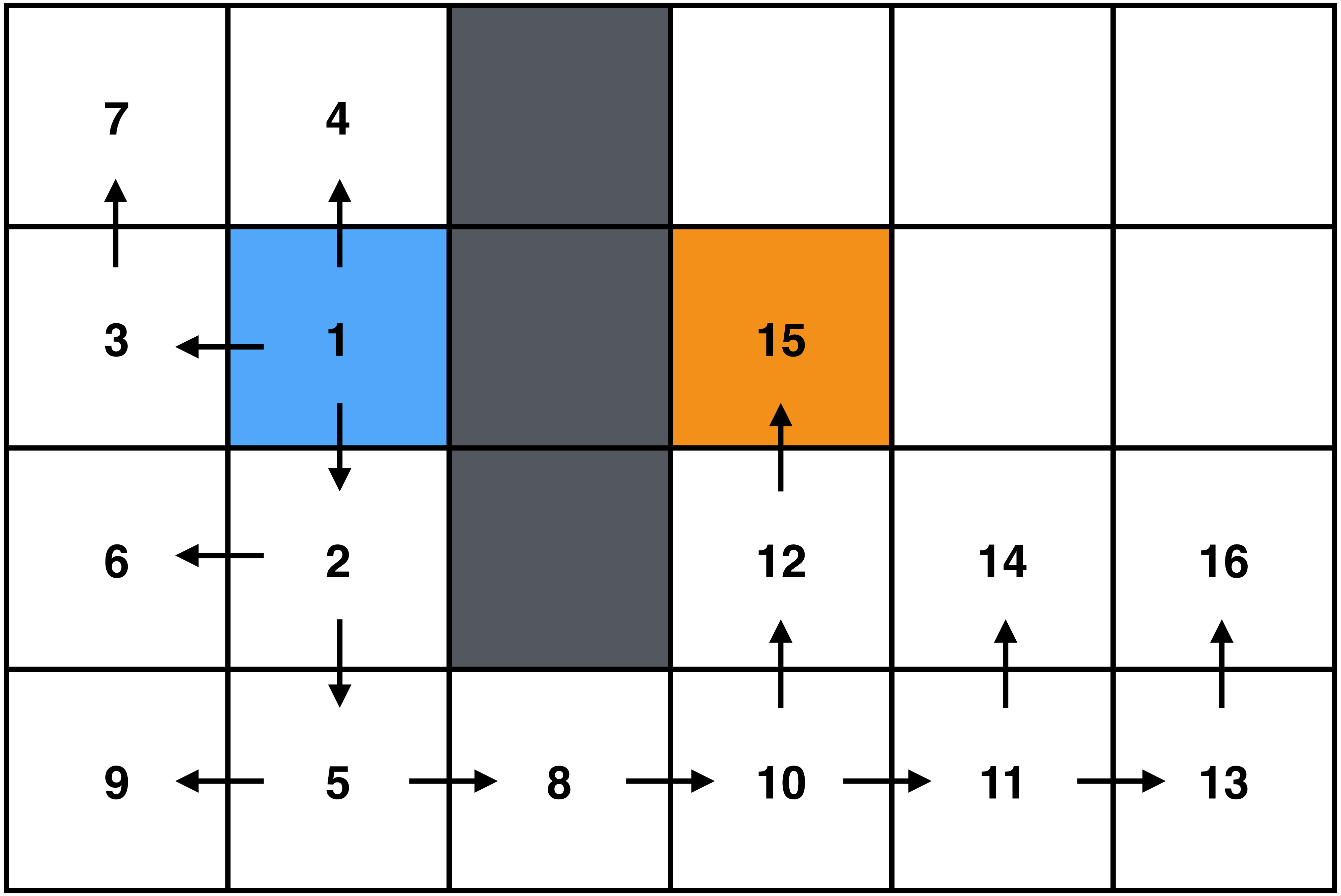
- Expand neighbours right ,down, left, and up



- Q=[12,13,14,15]

Breadth-First Search

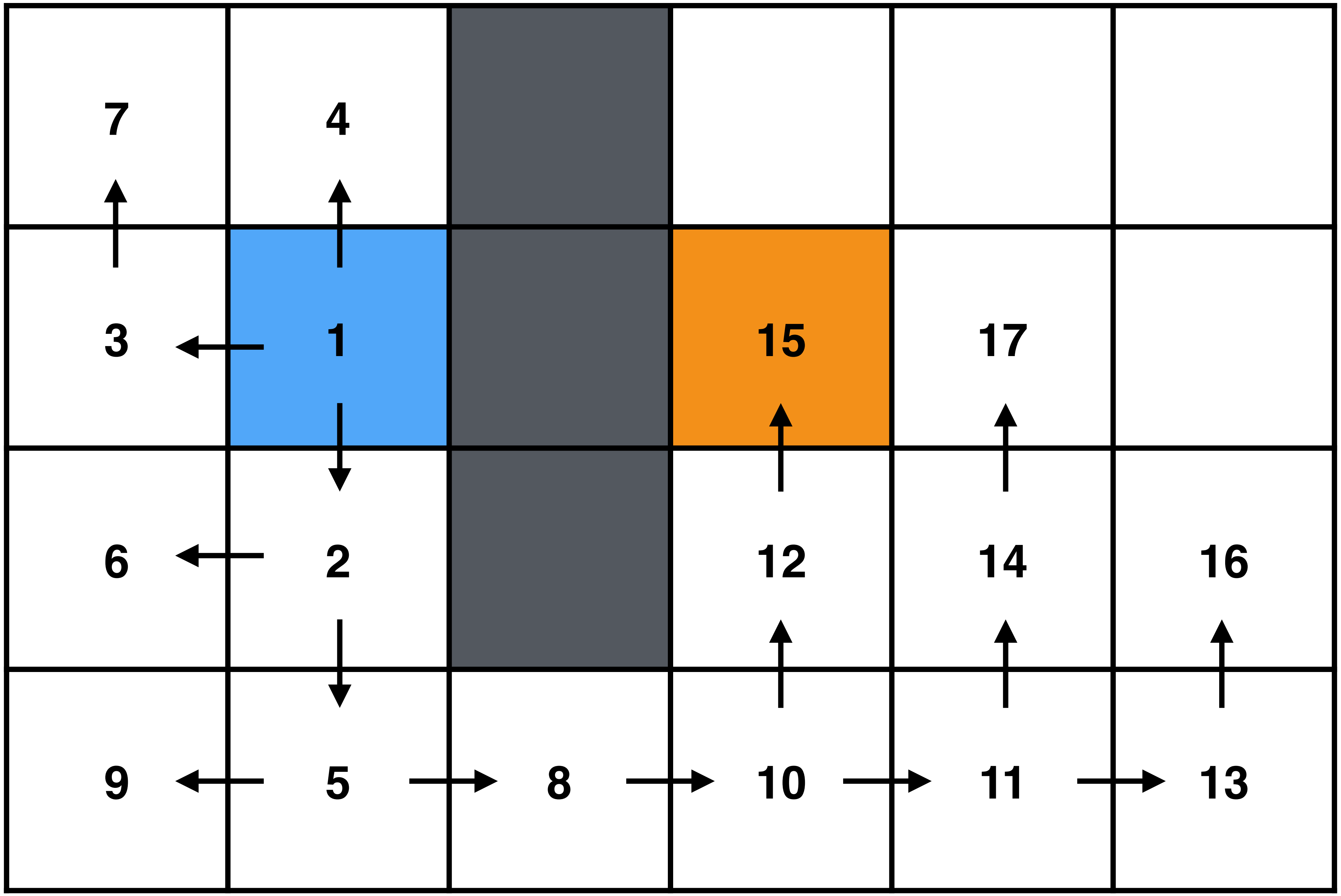
- Expand neighbours right ,down, left, and, up



- Q=[13,14,15,16]

Breadth-First Search

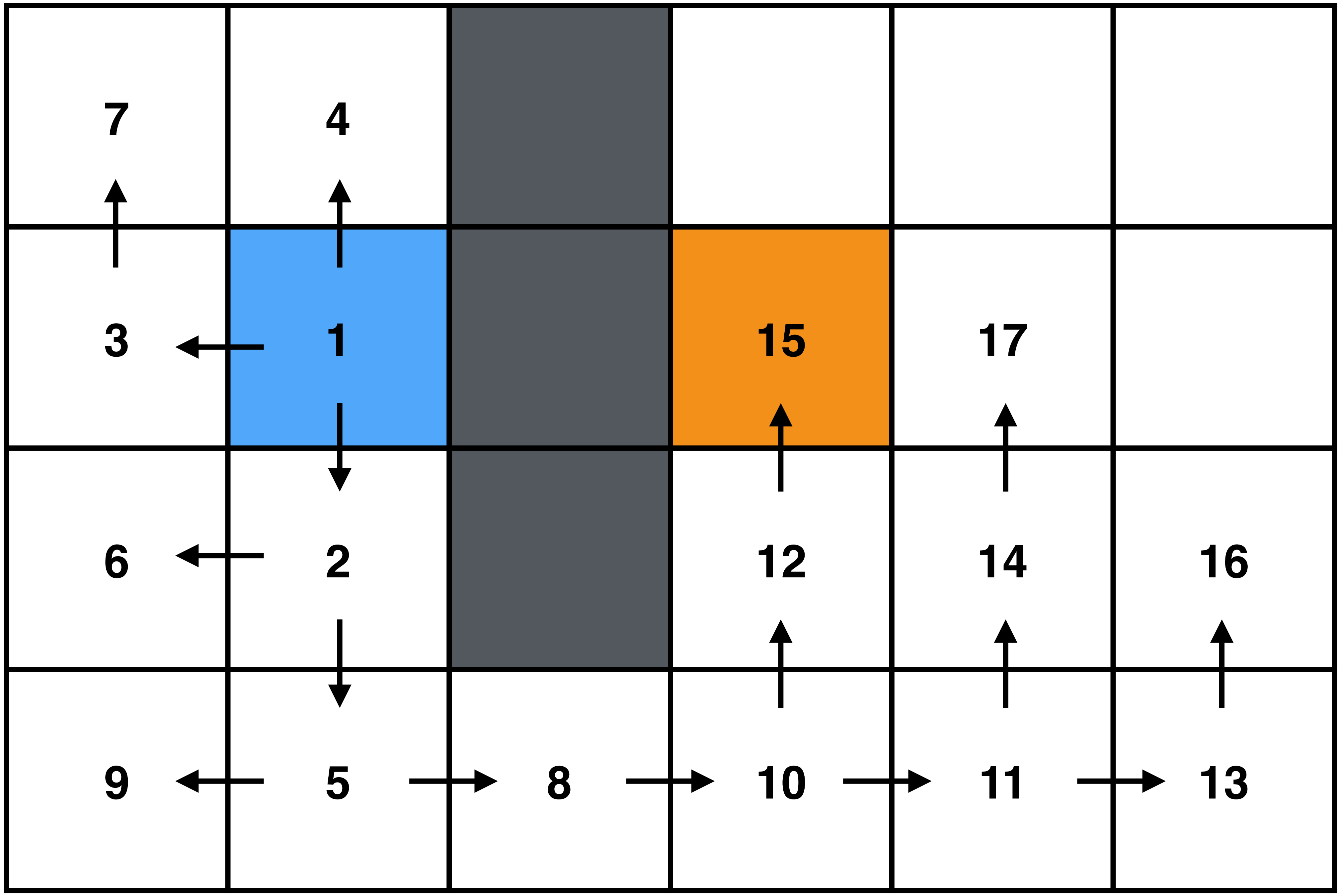
- Expand neighbours right ,down, left, and, up



- Q=[14,15,16]

Breadth-First Search

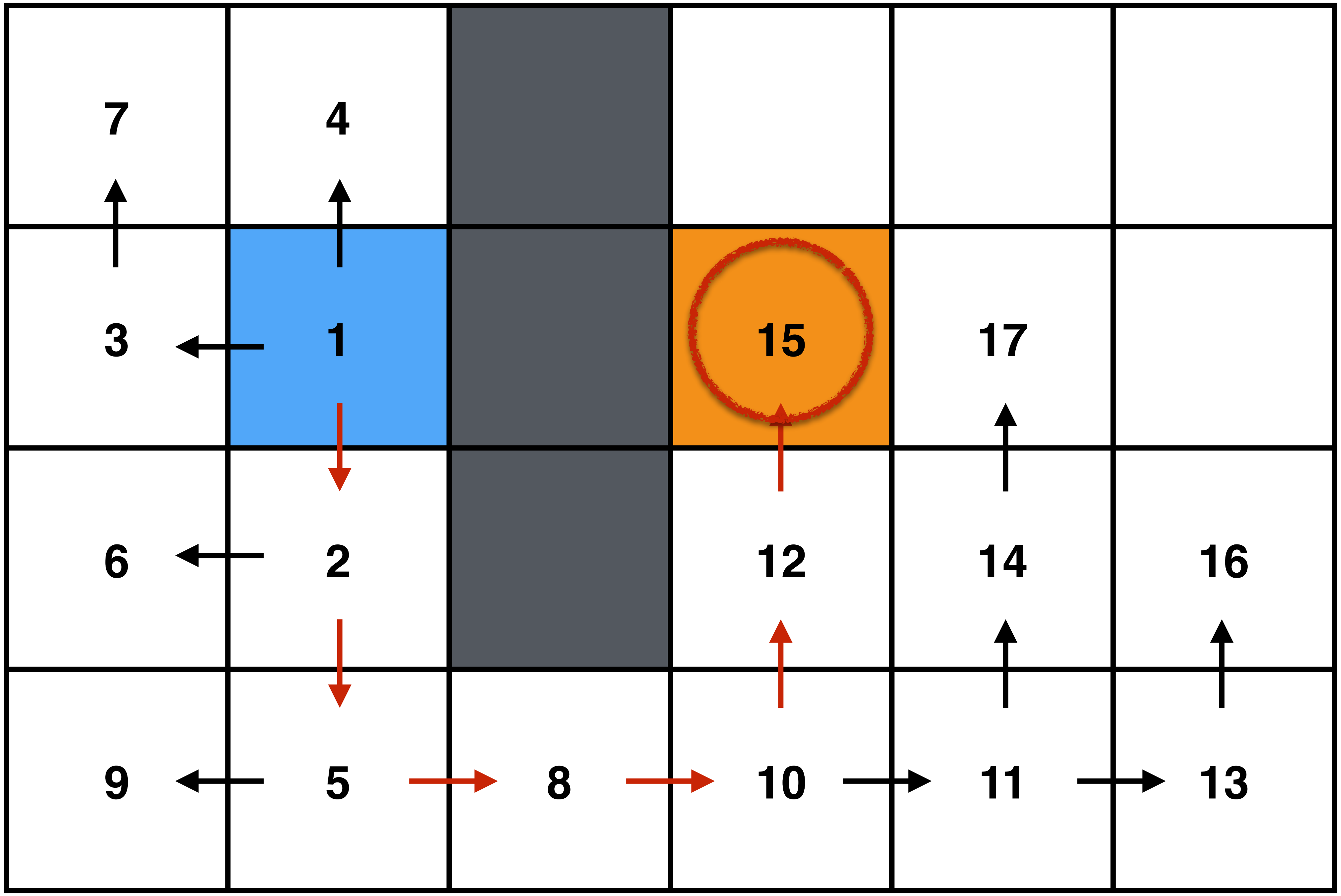
- Expand neighbours right ,down, left, and, up



- Q=[15,16] GOAL!!!

Breadth-First Search

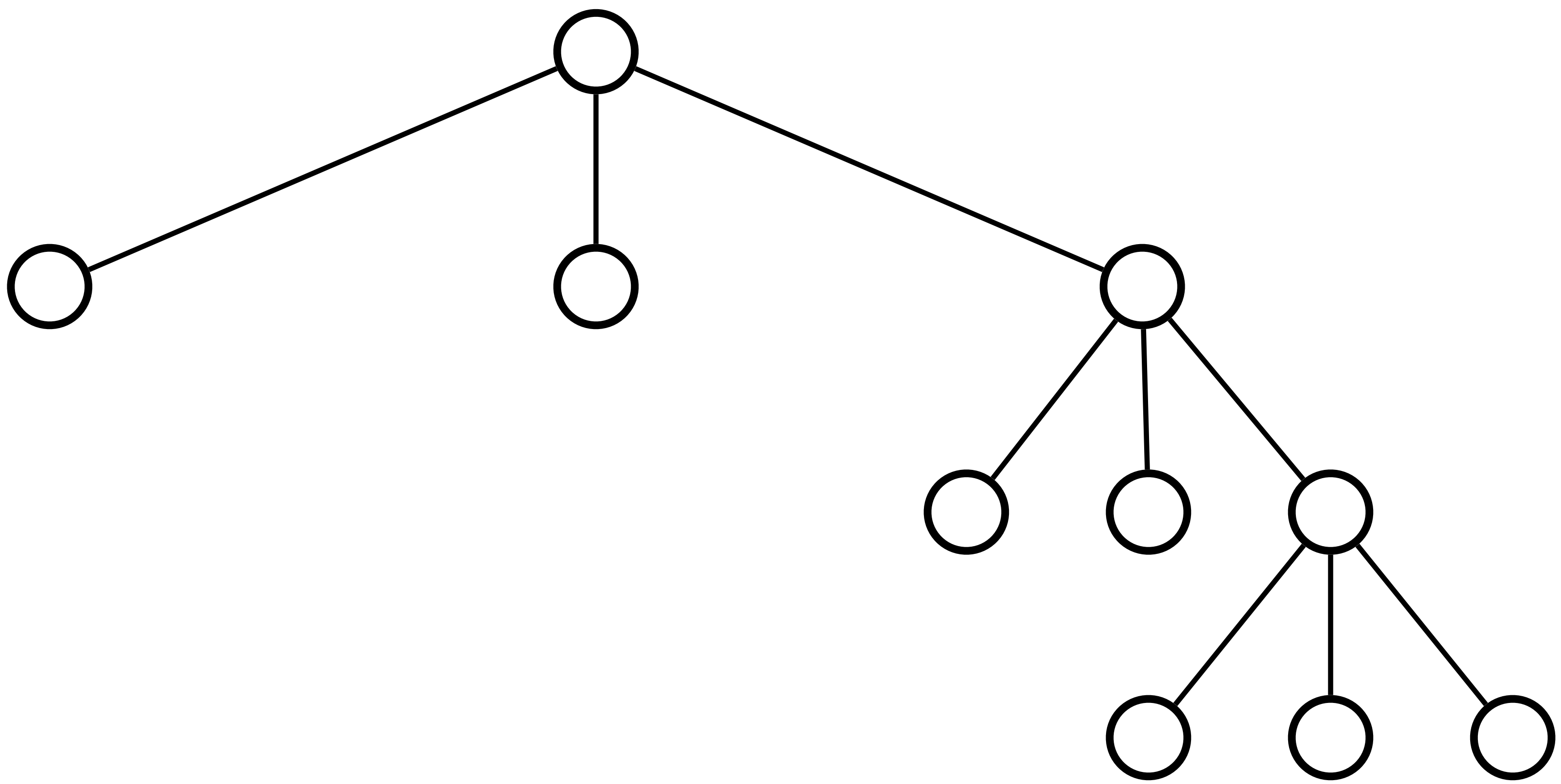
- Shortest path between start and goal



- lowest value neighbour: lower numbers closer or equal

Depth-First Search

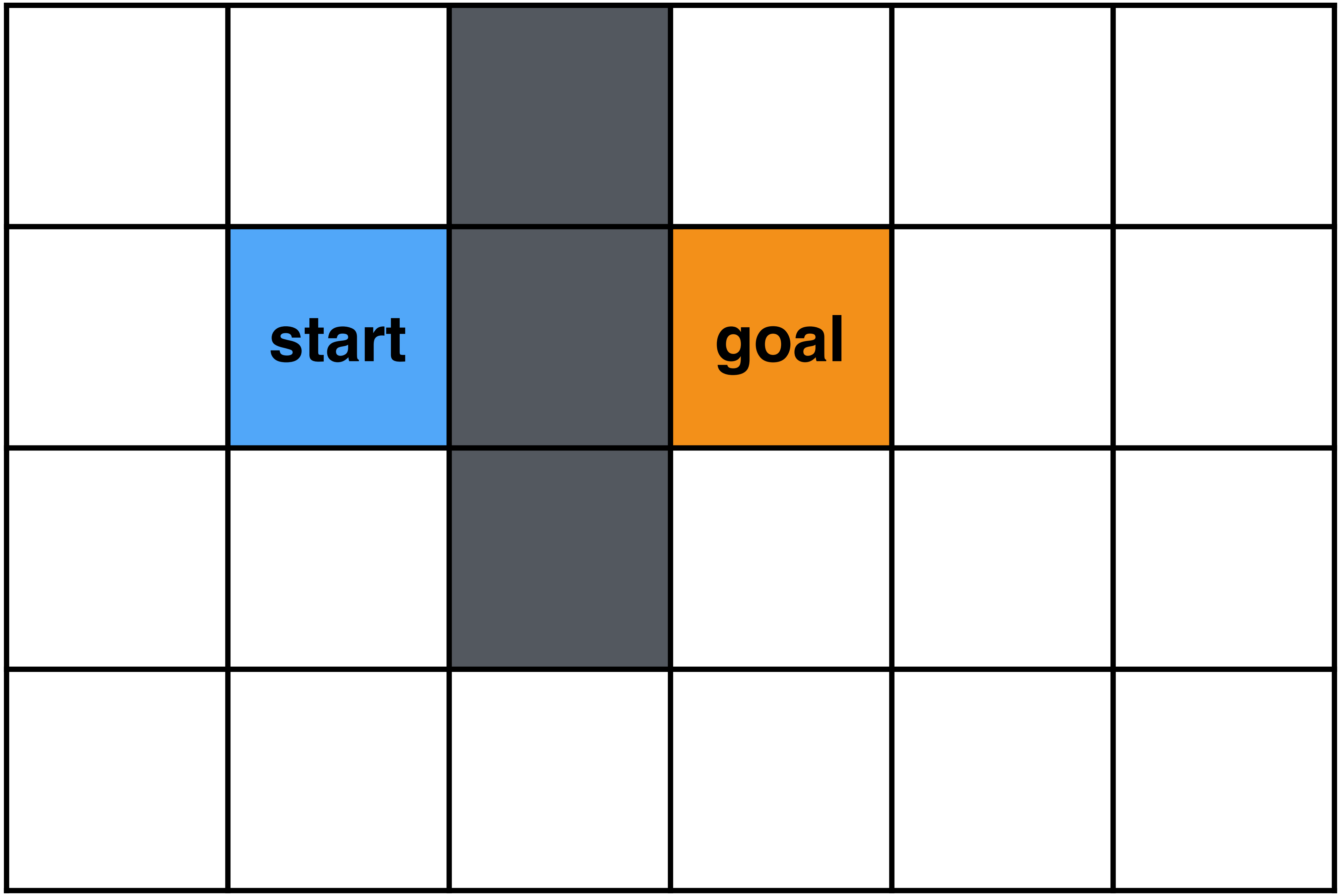
- Depth-first search of states
- Expand states according to last-in-first out queue



- Expand until a deadend or fixed depth (depth-limited)
- Quickly expands along one branch

Depth-First Search

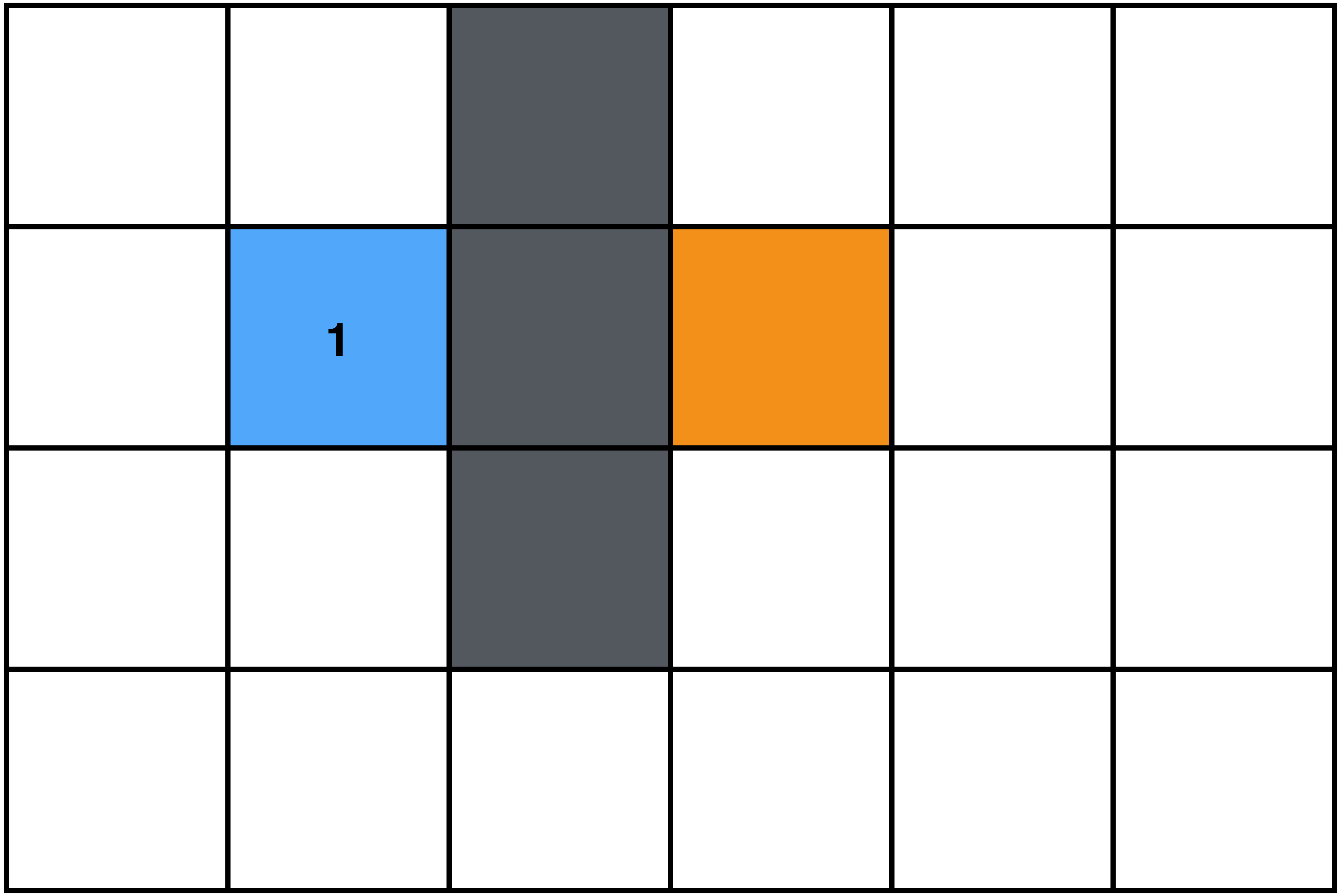
- Find start-to-goal path using down, left, right, up actions



- Compute X_obs online

Depth-First Search

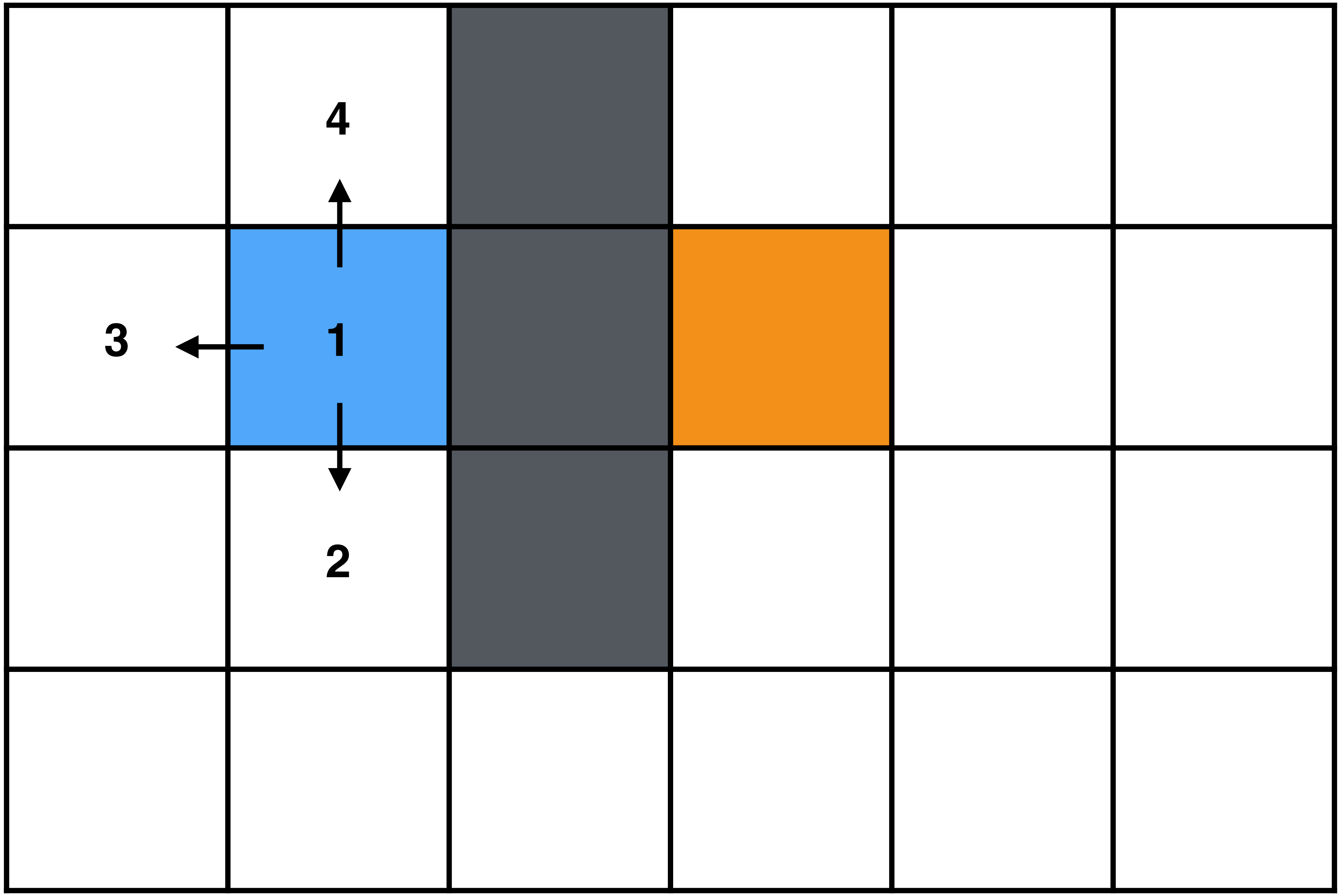
- Start at the start cell



- Q=[1]

Depth-First Search

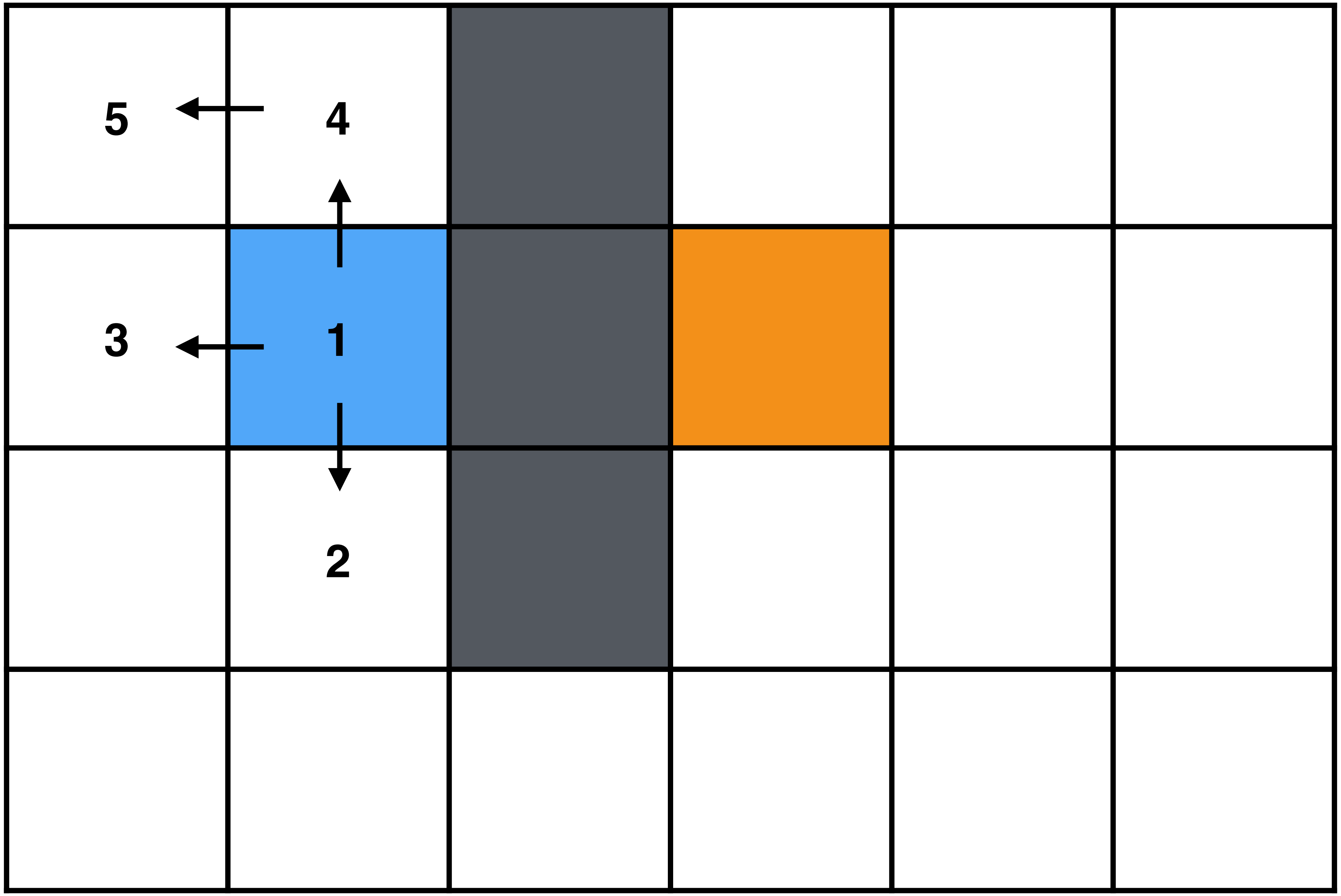
- Expand neighbours right ,down, left, and, up



- Q=[1,2,3,4]

Depth-First Search

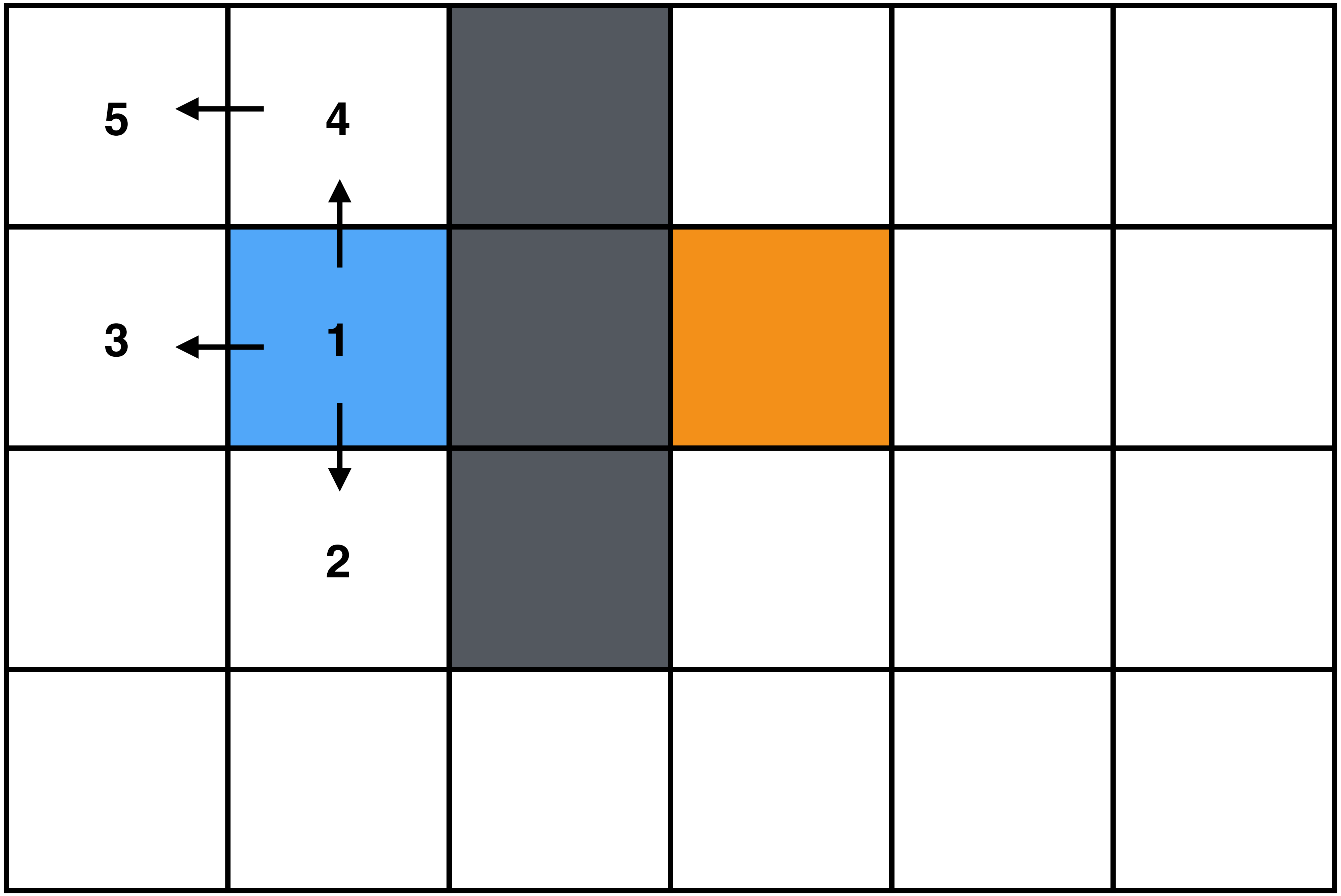
- Expand neighbours right ,down, left, and, up



- Q=[2,3,4,5]

Depth-First Search

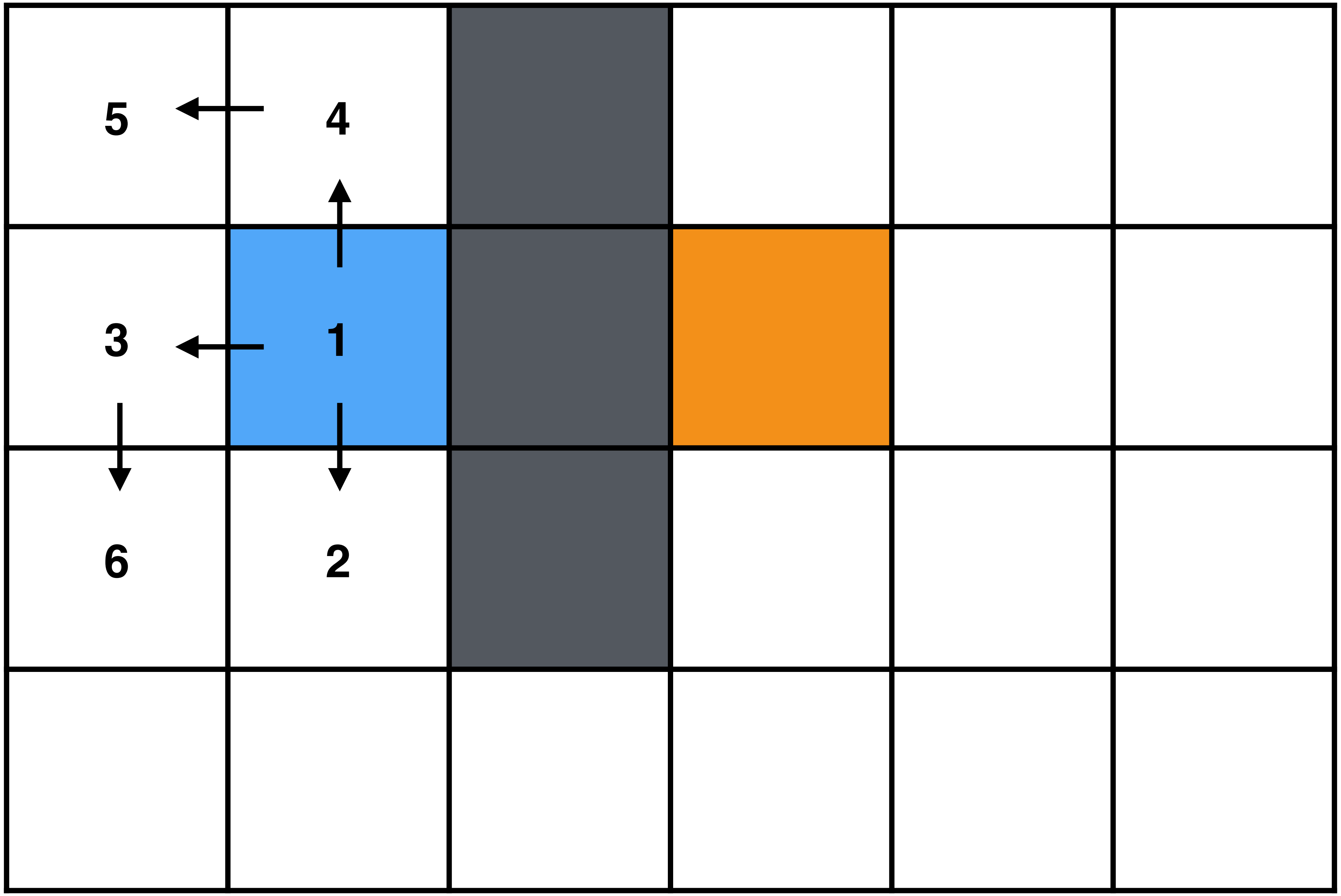
- Expand neighbours right ,down, left, and, up



- Q=[2,3,5]

Depth-First Search

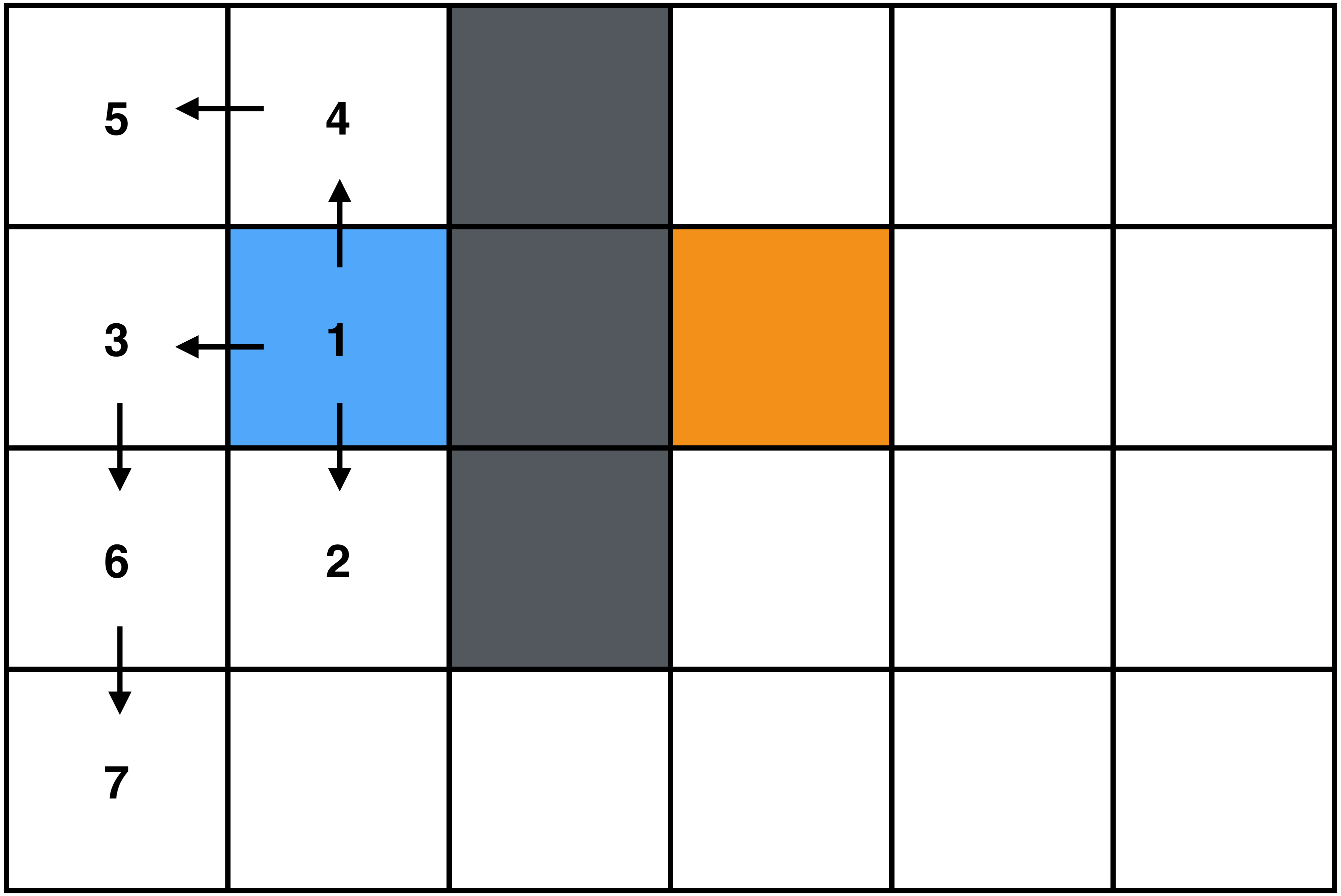
- Expand neighbours right ,down, left, and, up



- Q=[2,3,6]

Depth-First Search

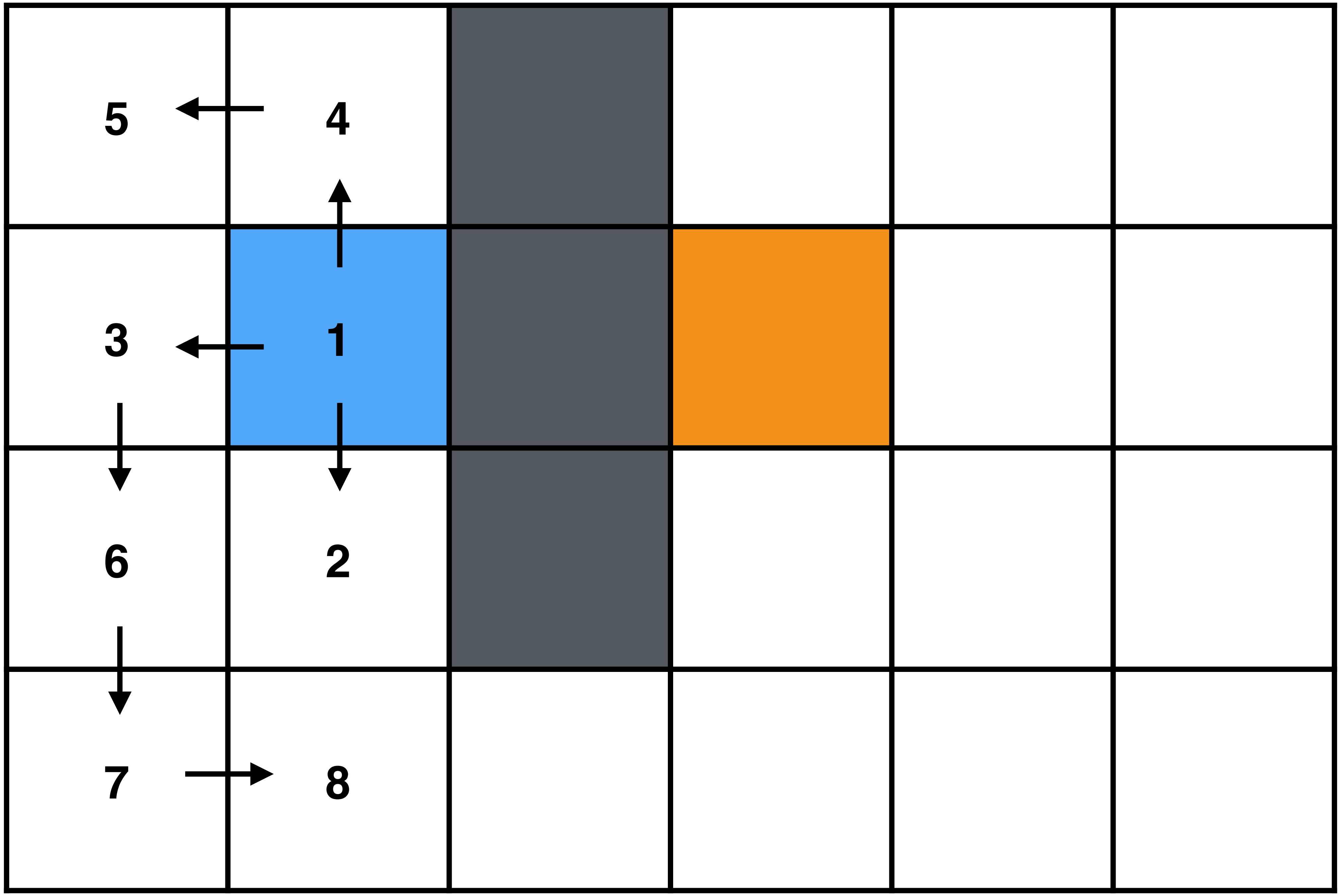
- Expand neighbours right ,down, left, and, up



- Q=[2,6,7]

Depth-First Search

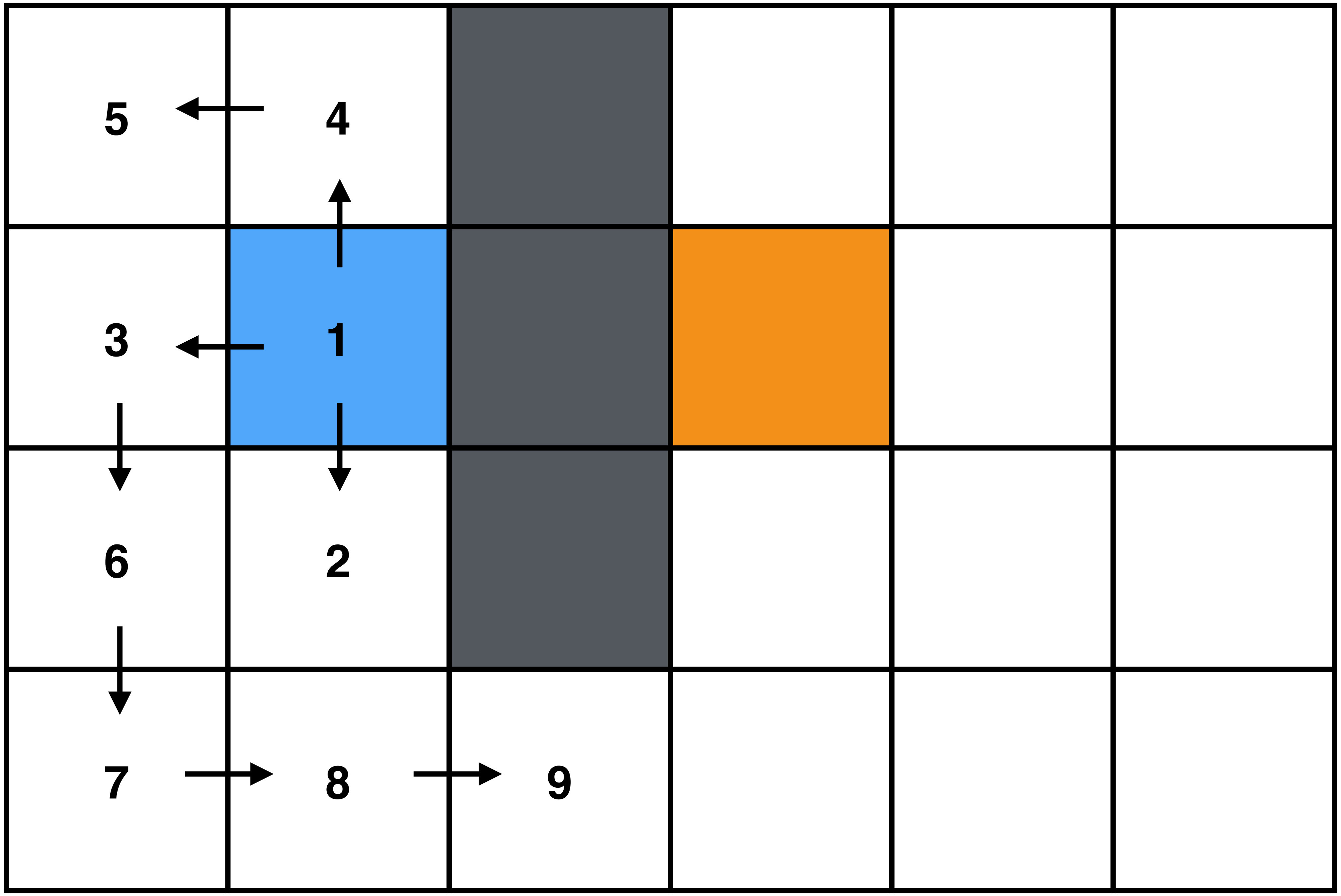
- Expand neighbours right ,down, left, and, up



- Q=[2,7,8]

Depth-First Search

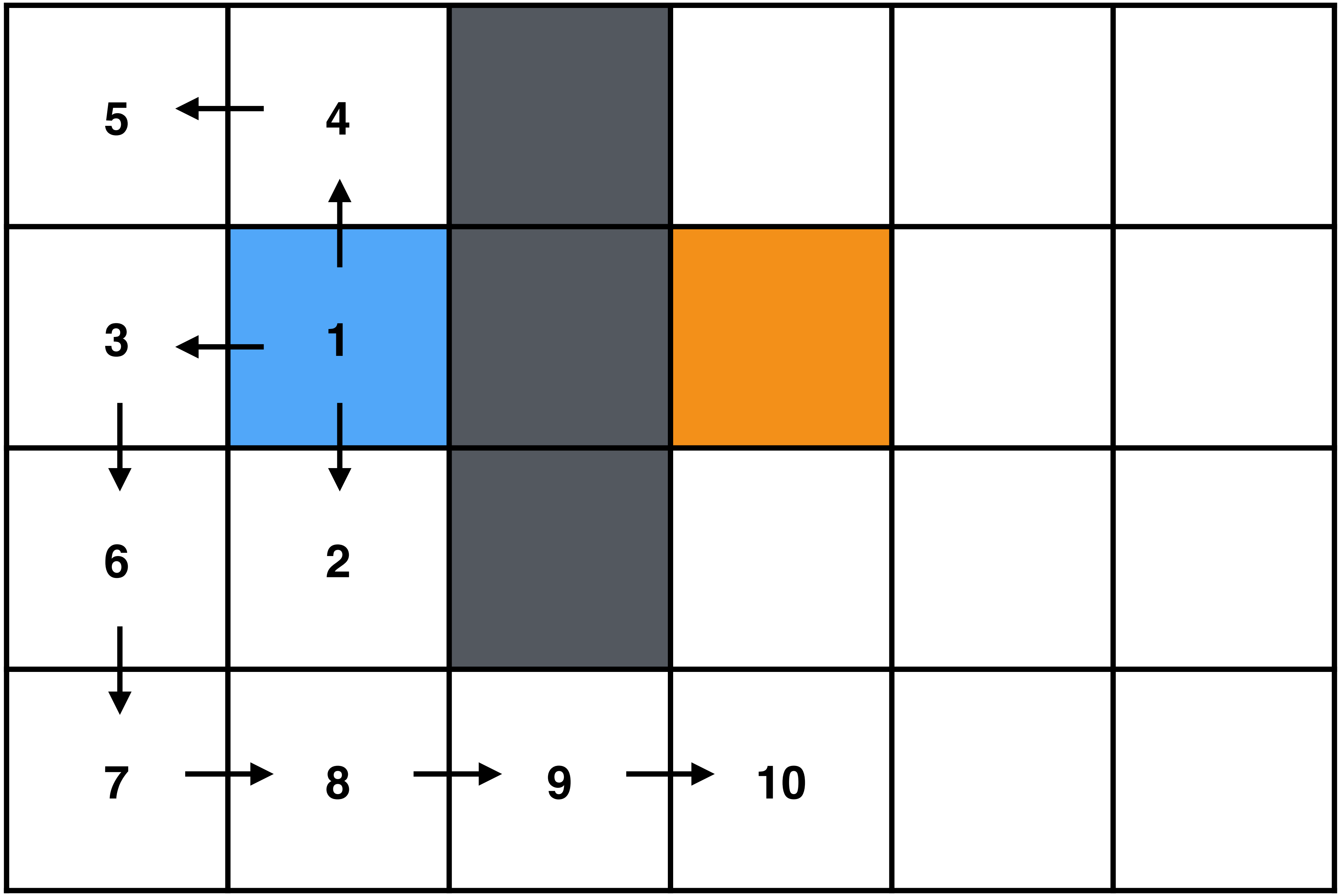
- Expand neighbours right ,down, left, and up



- Q=[2,8,9]

Depth-First Search

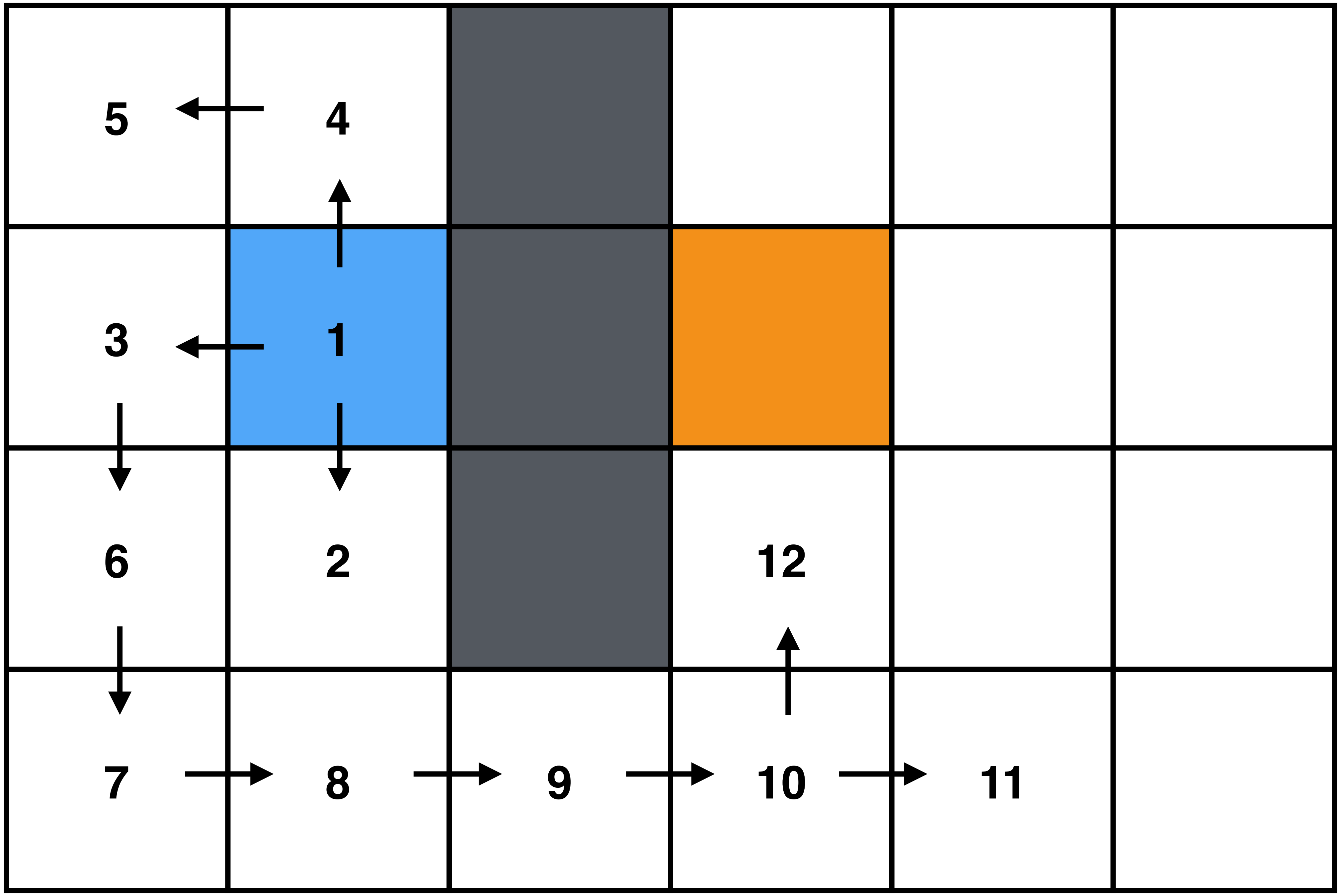
- Expand neighbours right ,down, left, and, up



- Q=[2,9,10]

Depth-First Search

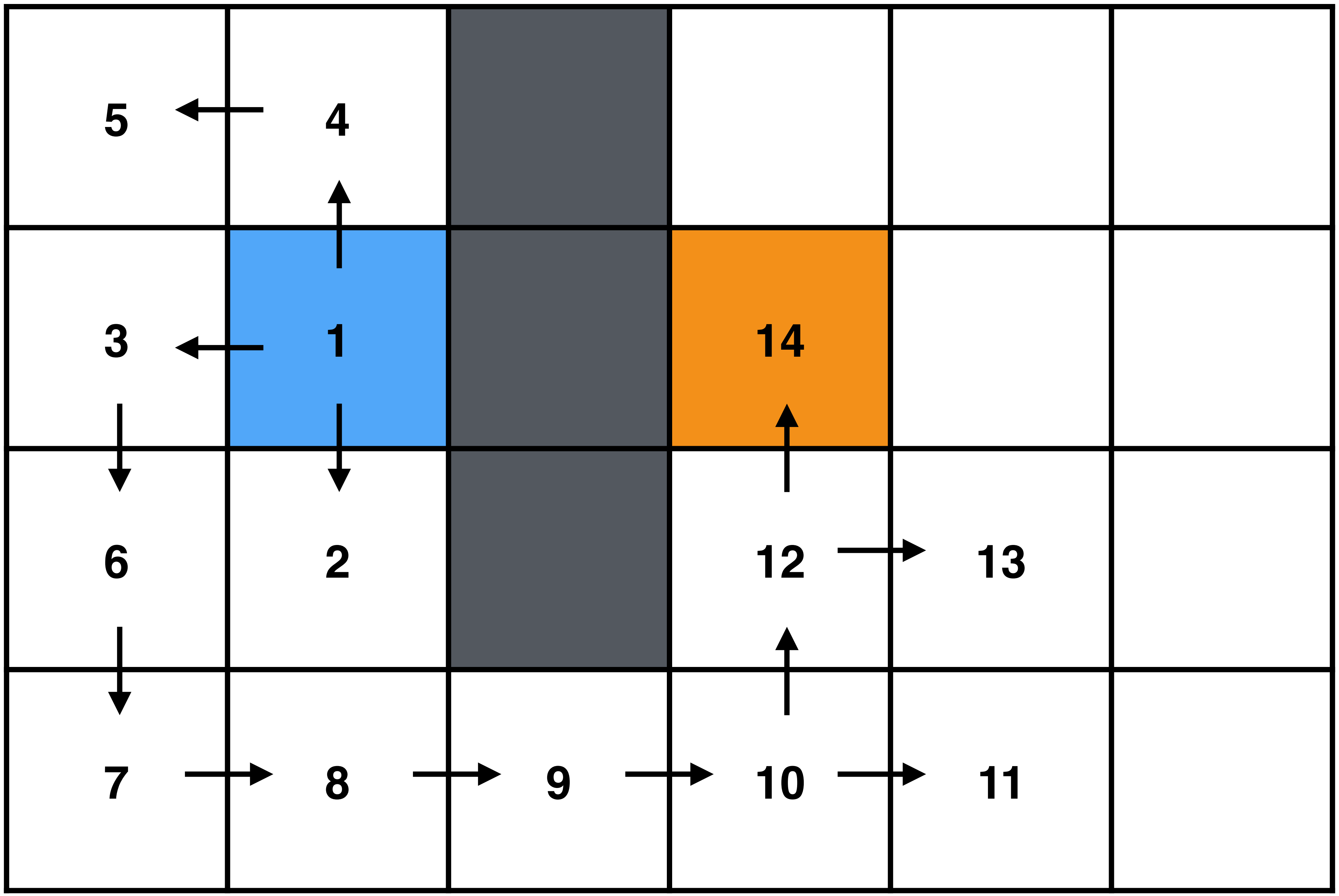
- Expand neighbours right ,down, left, and, up



- Q=[2,10,11,12]

Depth-First Search

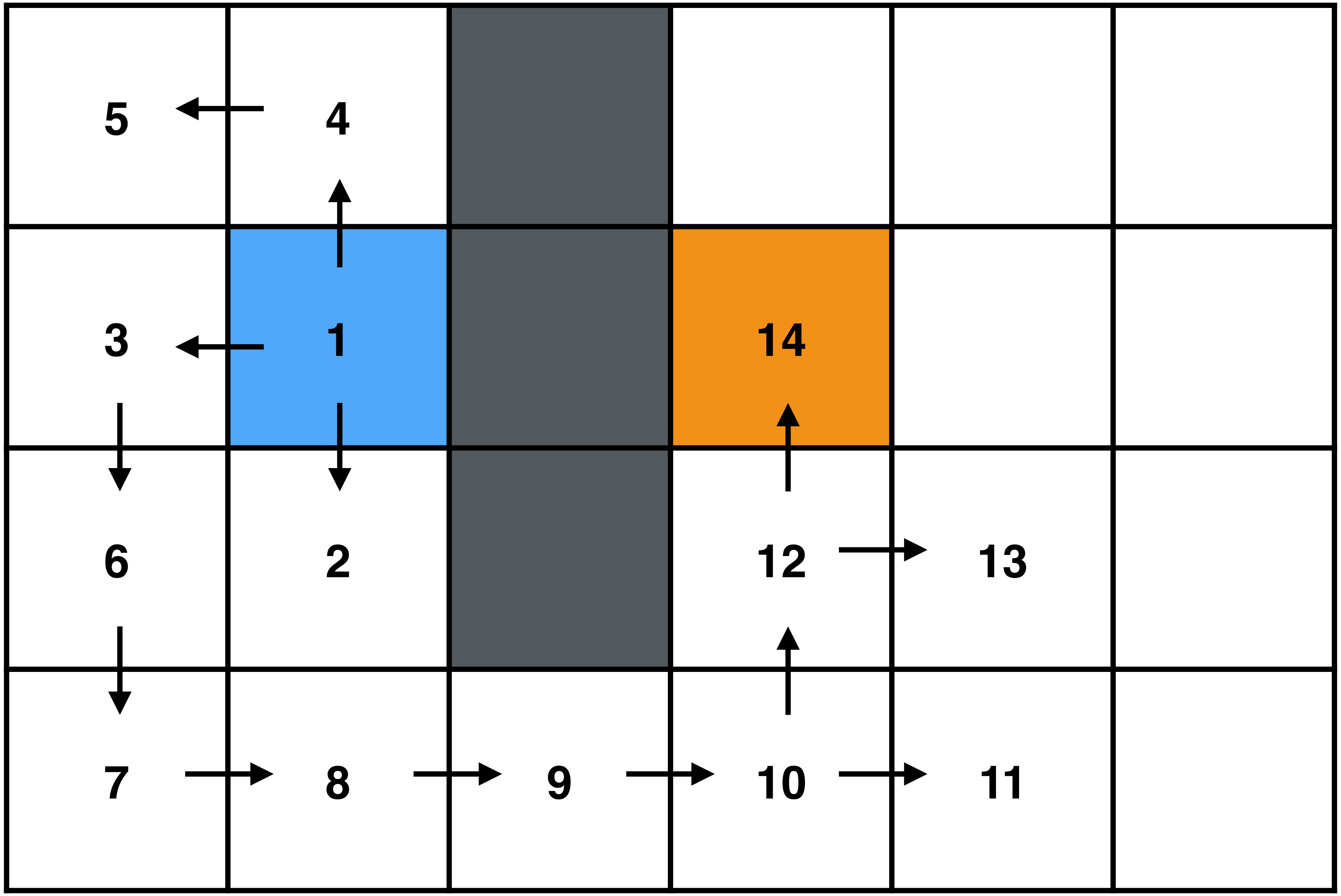
- Expand neighbours right ,down, left, and, up



- Q=[2,11,12,13,14]

Depth-First Search

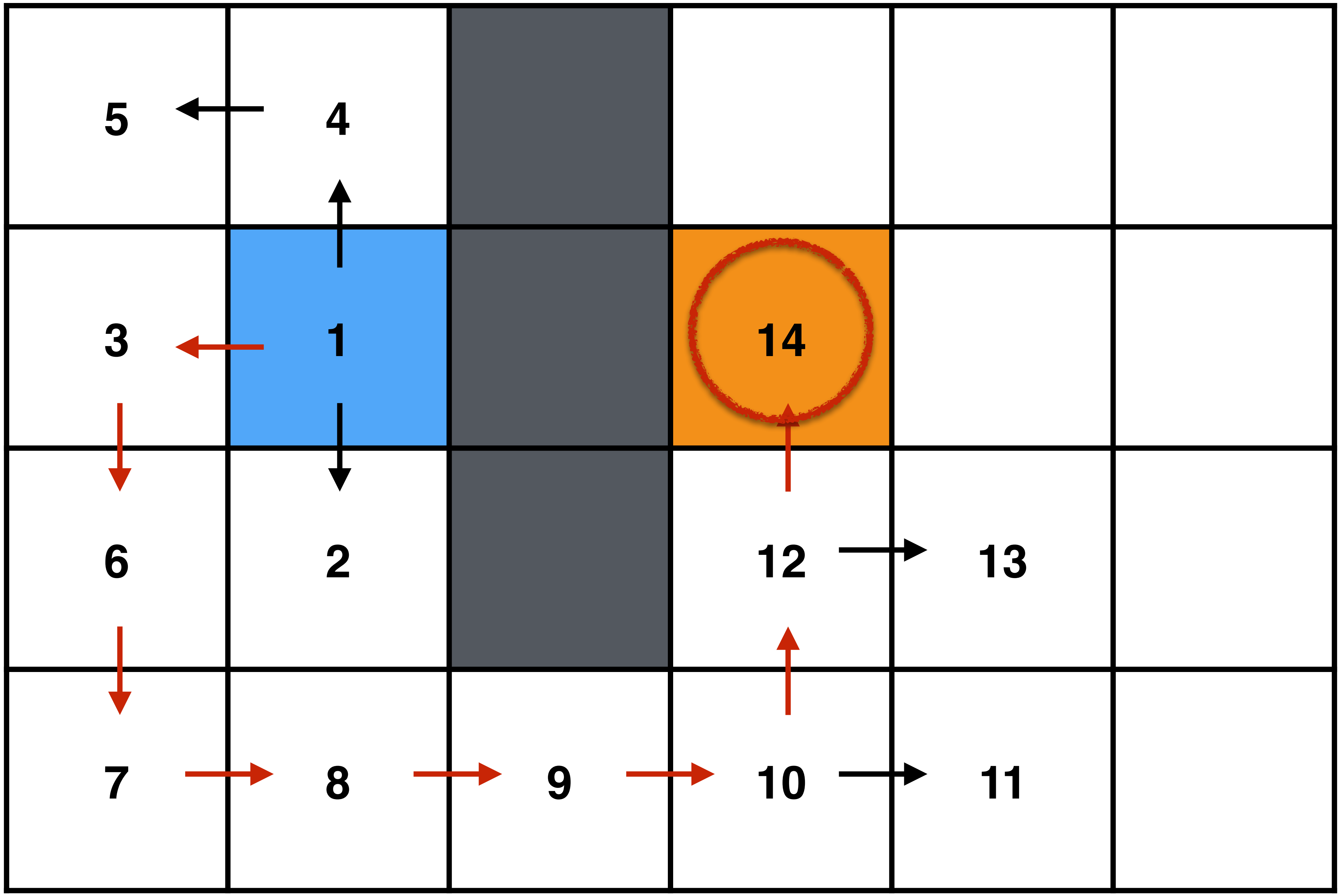
- Expand neighbours right ,down, left, and, up



- Q=[2,11,13,14] GOAL!!!

Depth-First Search

- Not the shortest path, can be very winding



- So… no advantage over breadth-first… whats the point?

Tree Search vs. Graph Search

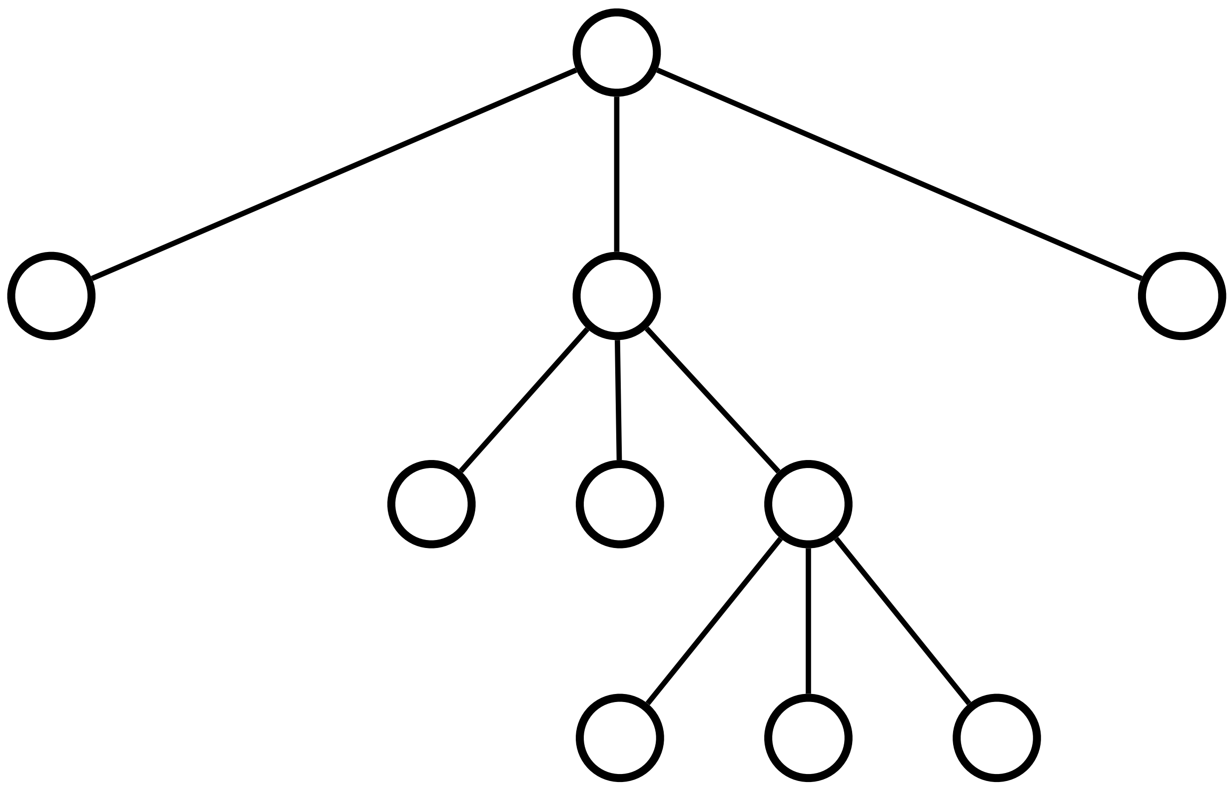
- Tree Search - Expand to <u>all</u> neighbours
- Graph Search - Expand to neighbours <u>not</u> yet visited

- We employed graph search in the examples

- Tree search can have same state multiple times in tree
- Graph search avoids redundant paths and infinite loops
  - Infinite loops make search incomplete even for finite states
  - Tree can avoid loops by comparing to states along path

Depth-First Tree Search

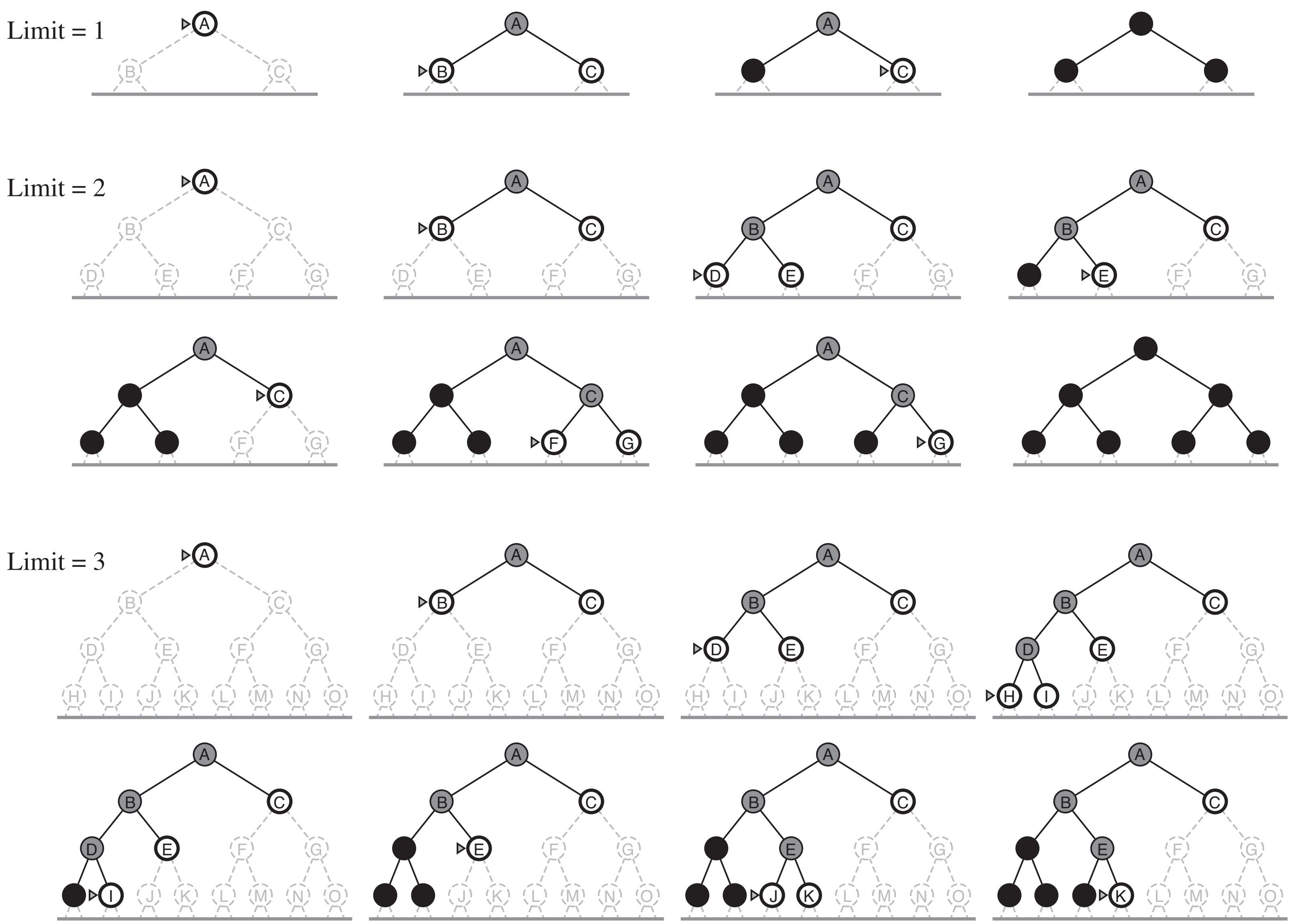
- Depth-first tree search is memory efficient



- Only need to store current path and sibling nodes
- Avoid having to store huge tree

Iterative Deepening

- Use depth-first tree with incremental max depth



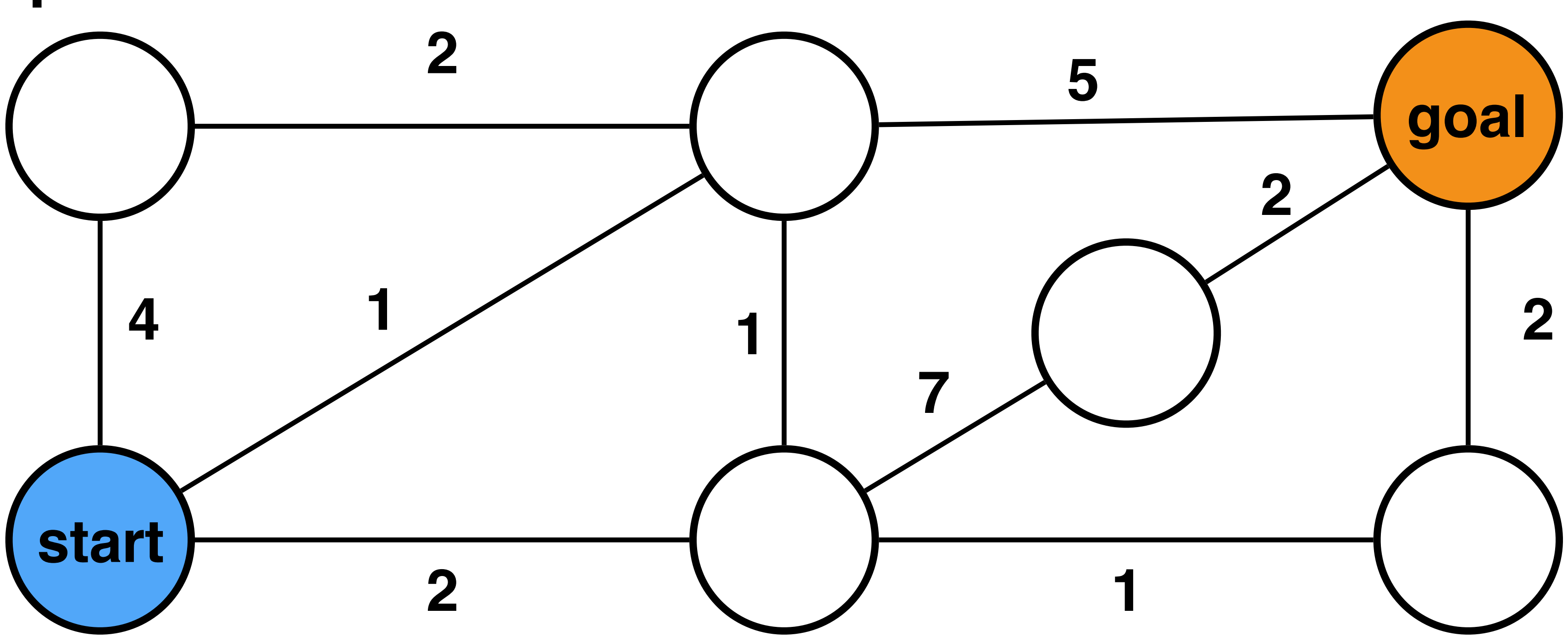
- Useful for problems with large branching factor
- Low memory usage and finds shortest path

Optimal Planning

- Not all actions or paths may have equal cost
- Loss function $l(x, u)$
- Want to find a path that minimises loss

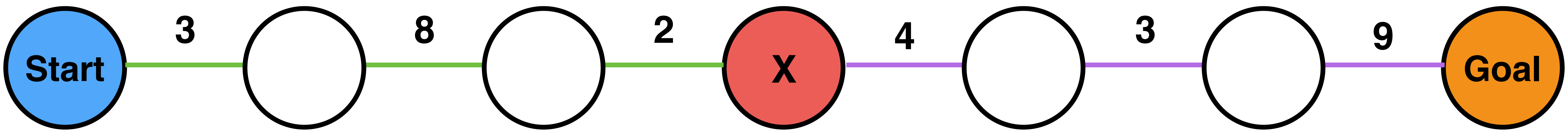
$$\arg\min_{\tau} \sum_{\tau} l(x, u)$$

- Example:



Cost-to-come and Cost-to-go

- Divide total cost into two parts:



Cost-to-come

$$g(x) = 13$$

Cost-to-go

$$h^*(x) = 16$$

- Cost-to-come is computed as we expand our search
- Cost-to-go is generally not available
- Approximate cost-to-go with a heuristic function

$$h(x)$$

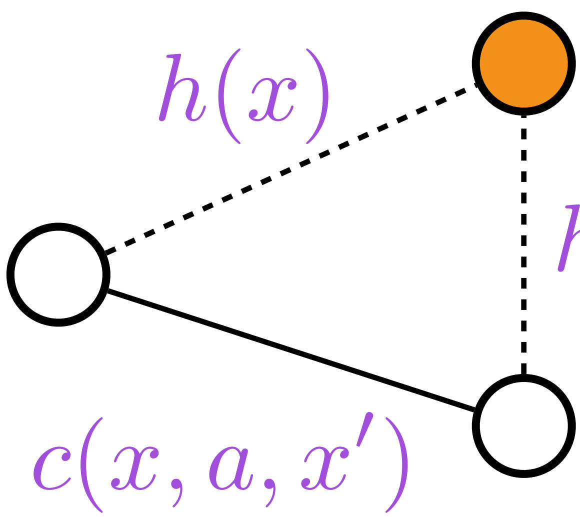
What Makes a Good Heuristic?

- The heuristic is admissible

$$h(x) \leq h^*(x)$$

  - Never overestimate the true cost-to-go
  - Avoids skipping over the shortest path

- It is consistent/monotonic, follows the triangle inequality



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

  - Needed for finding optimal paths with graph search (or do some extra bookkeeping)

What Makes a Good Heuristic?

- Heuristic is often computed by solving an easier problem
  - Additional challenges/constraints only increase distance/cost

- Often use Euclidean distance to goal for shortest path
  - Next node always has a shortcut action for going to the goal
  - Ignore obstacles along the way

Search Algorithms

- Organize priority Q according to different values
- Dijkstra's Algorithm

$$g(x)$$

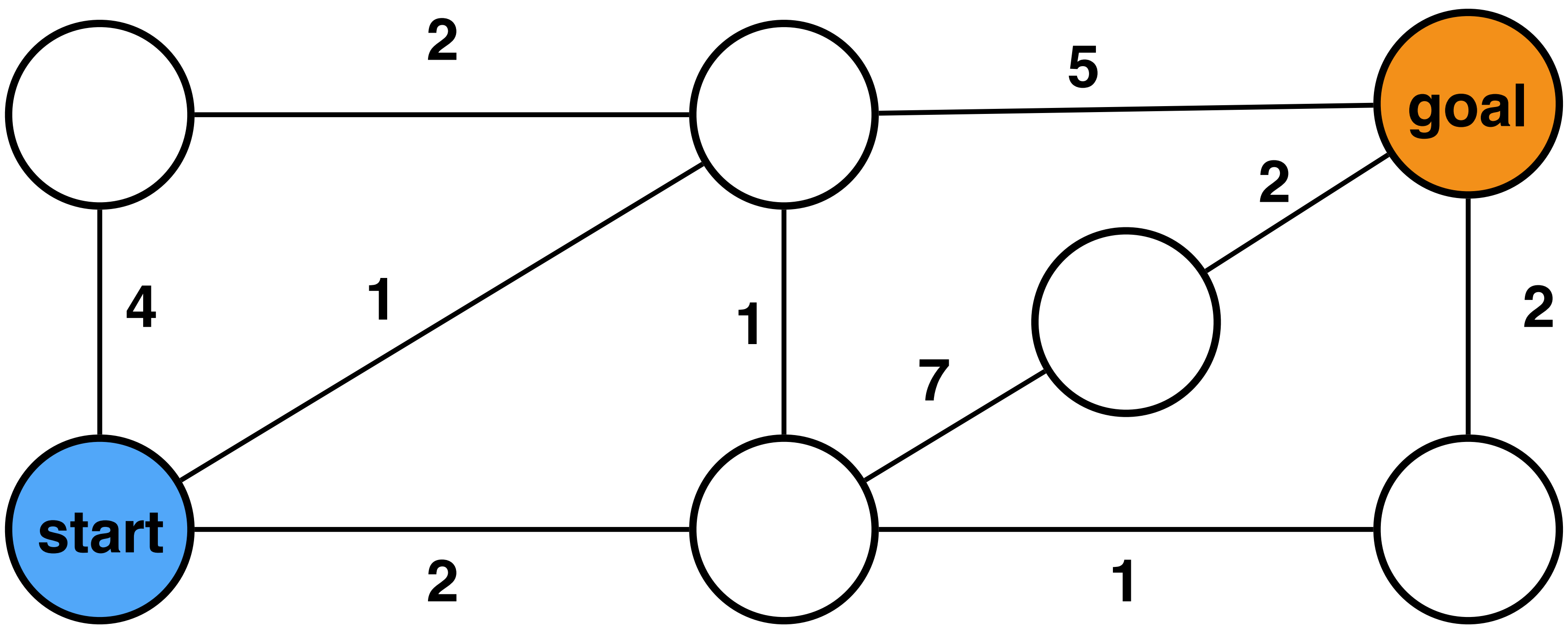
- Best-First Search Algorithm

$$h(x)$$

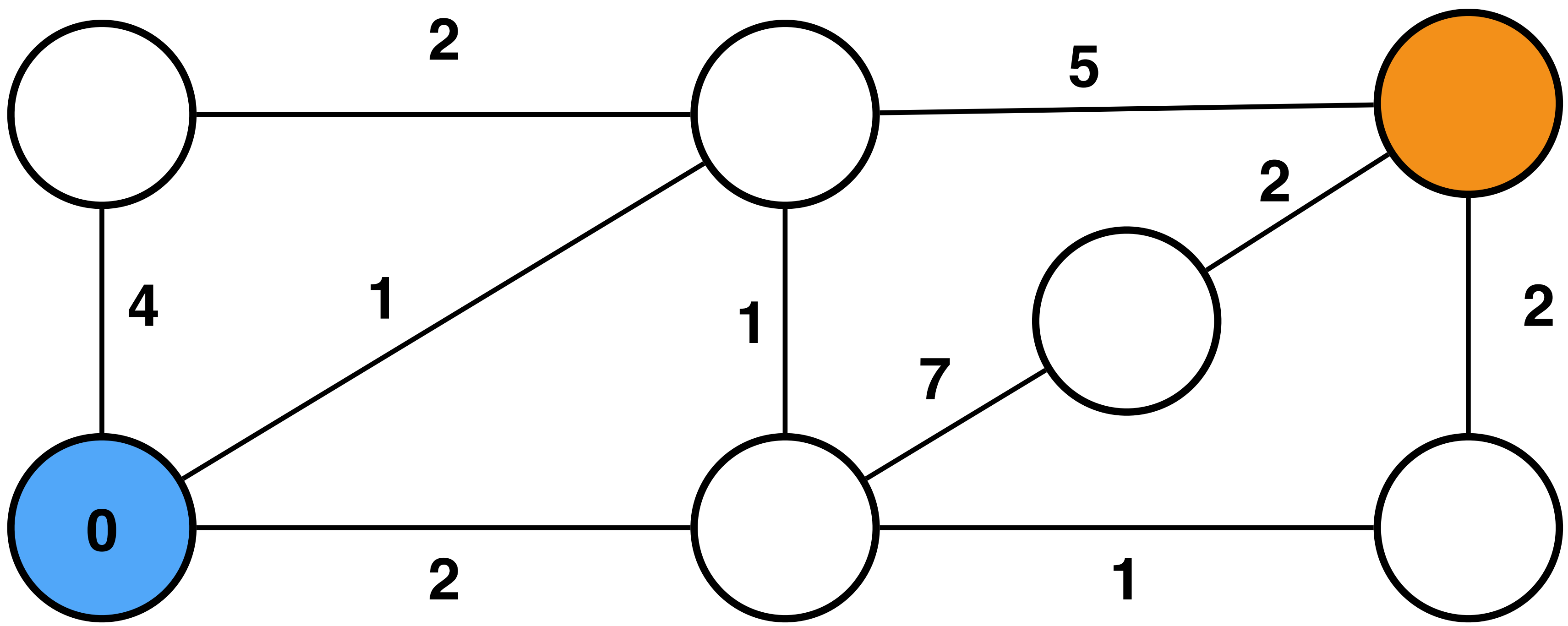
- A* Algorithm ("A star")

$$g(x) + h(x)$$

Djikstra’s Algorithm

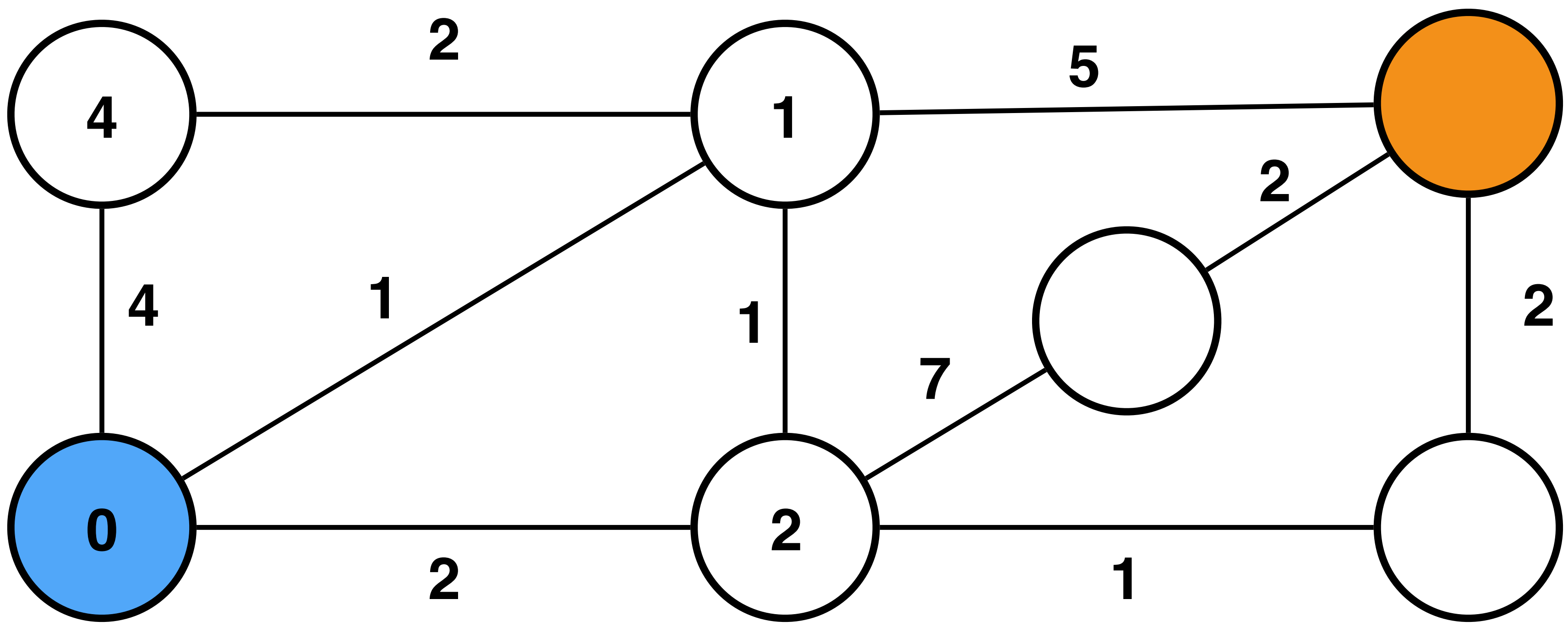


Djikstra's Algorithm



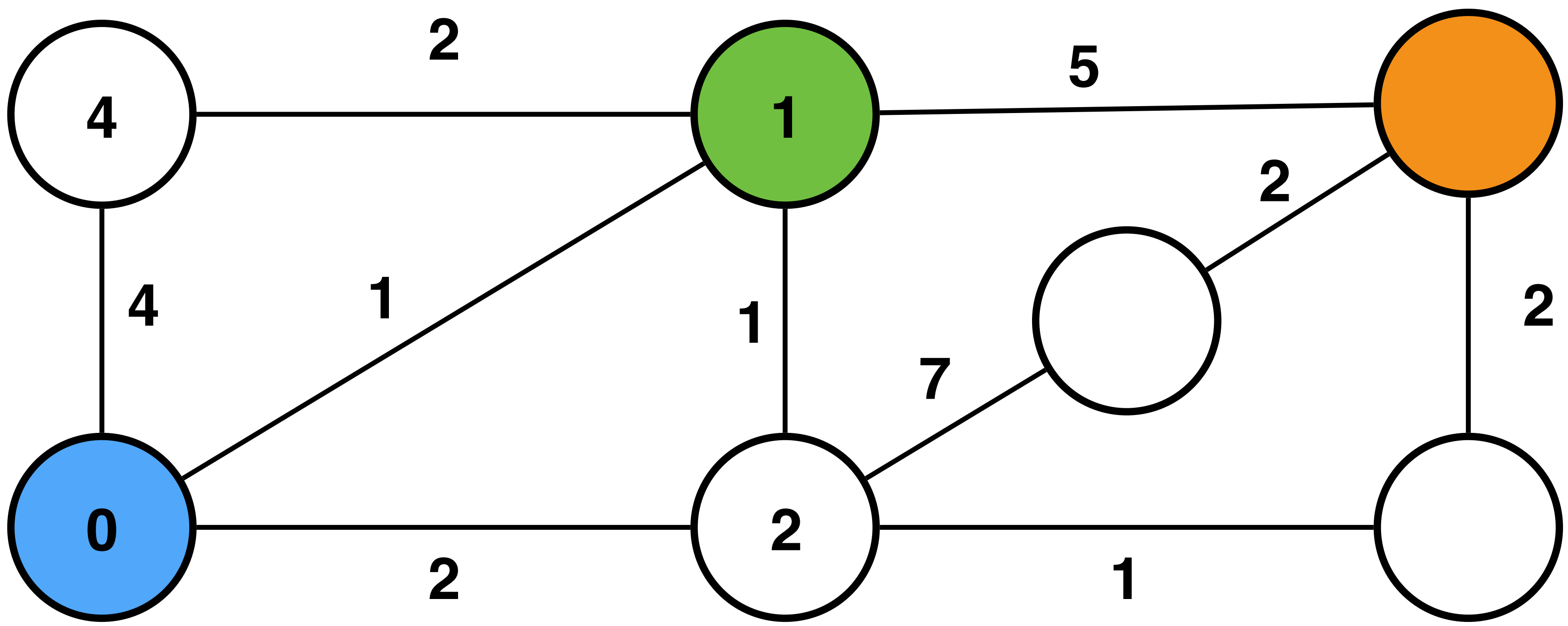
Djikstra's Algorithm

- Estimate cost-to-come of neighbours



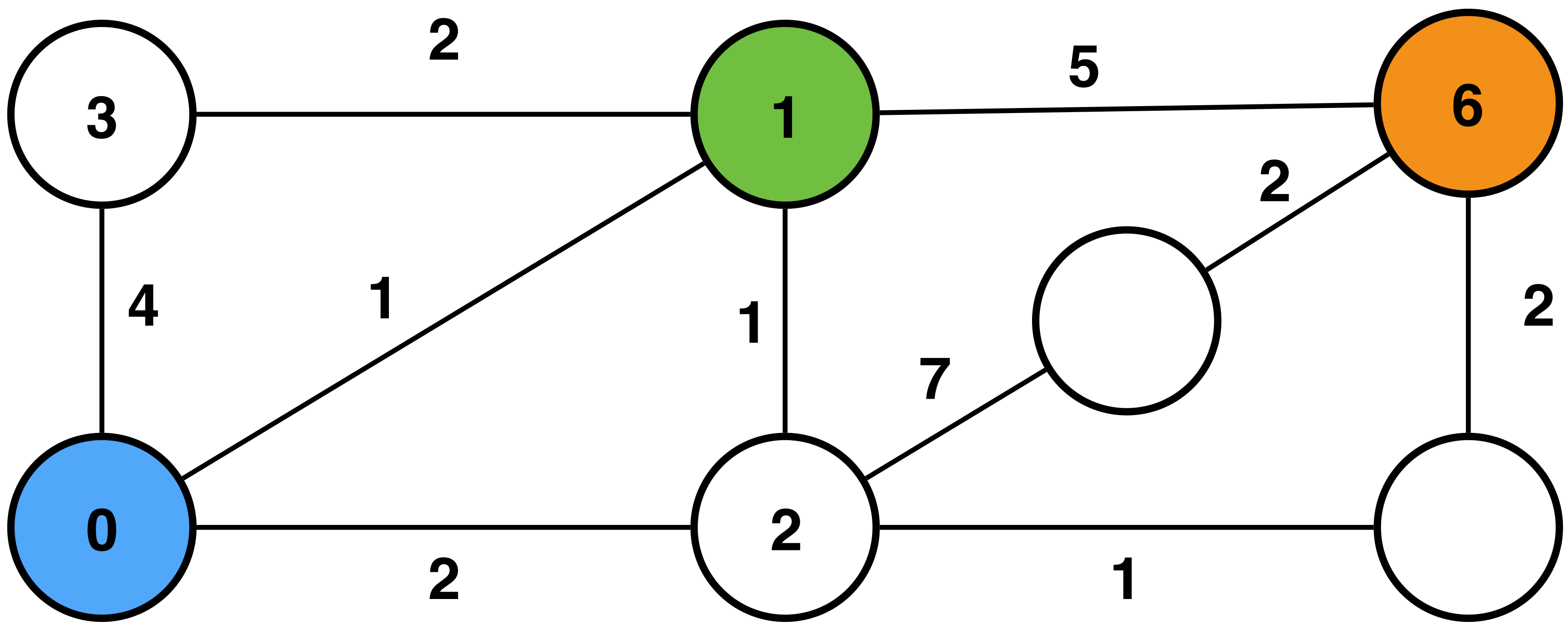
Djikstra's Algorithm

- Visit node with lowest cost-to-come (true cost-to-come)



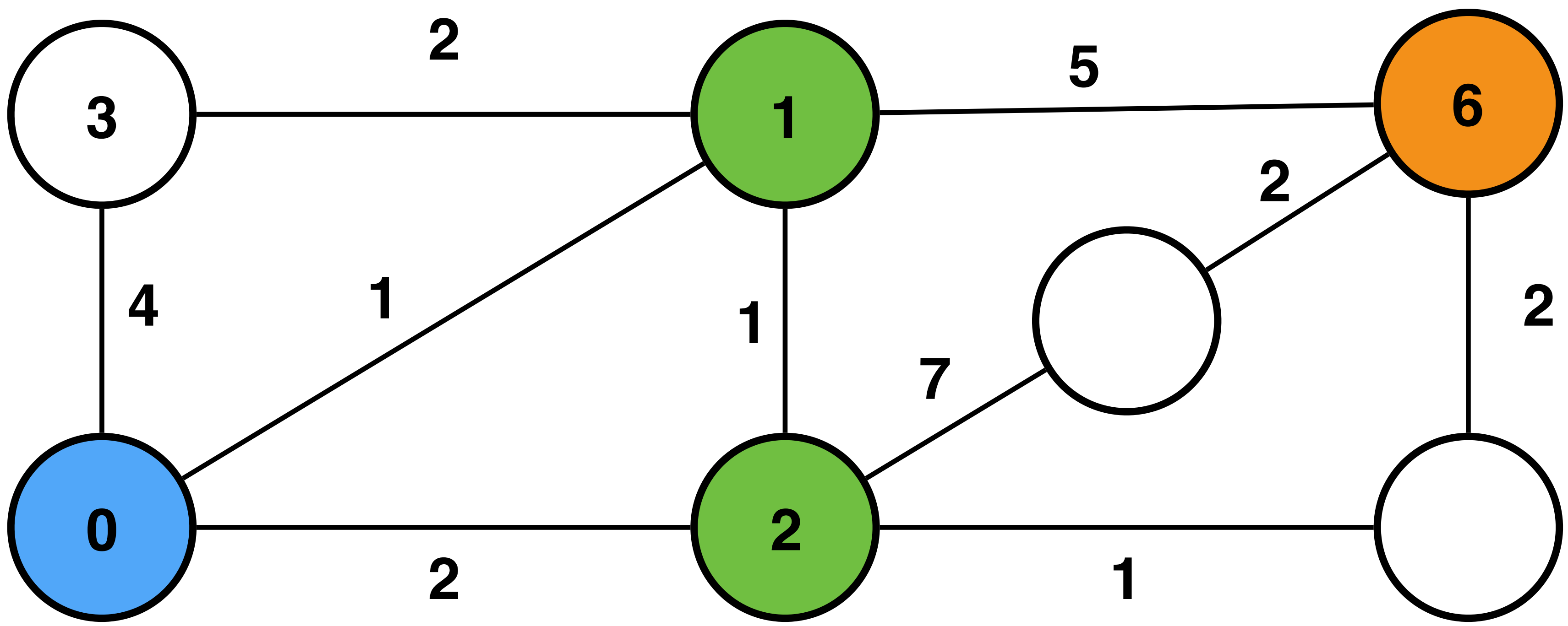
Djikstra's Algorithm

- Estimate cost-to-come of neighbours



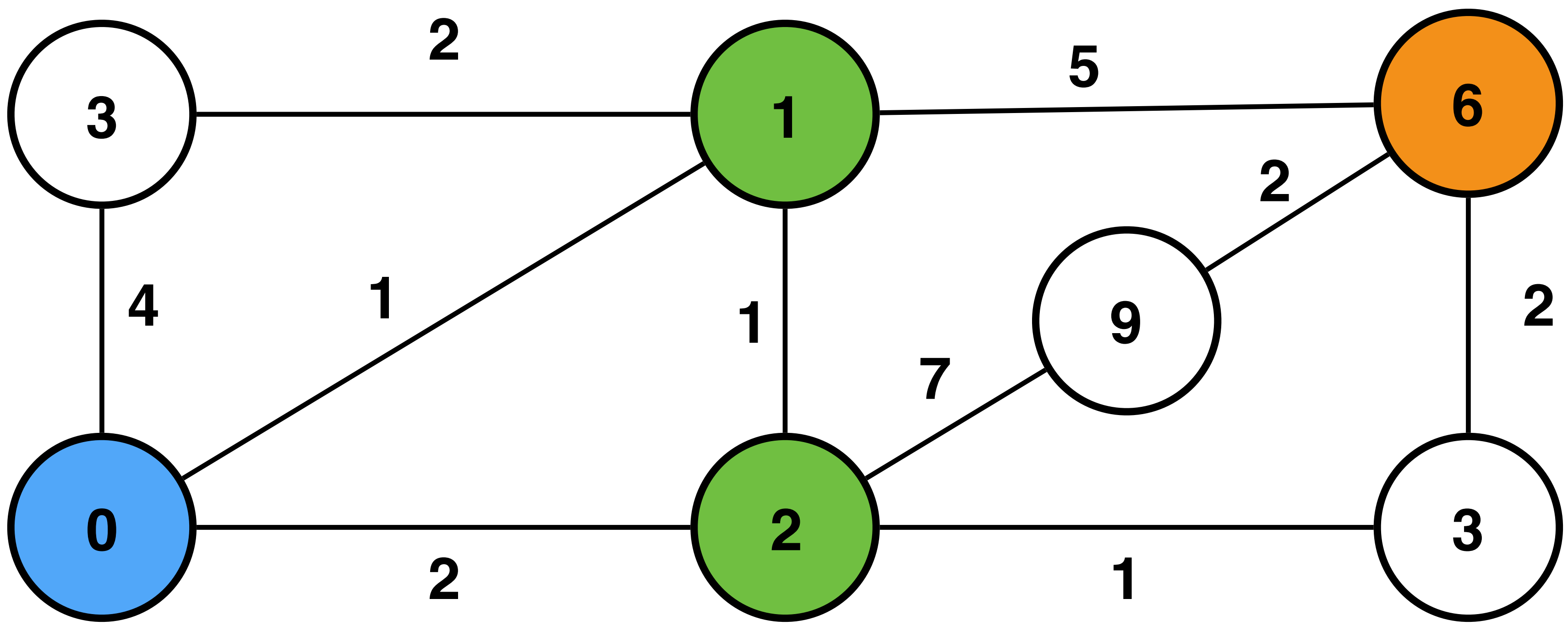
Djikstra's Algorithm

- Visit node with lowest cost-to-come



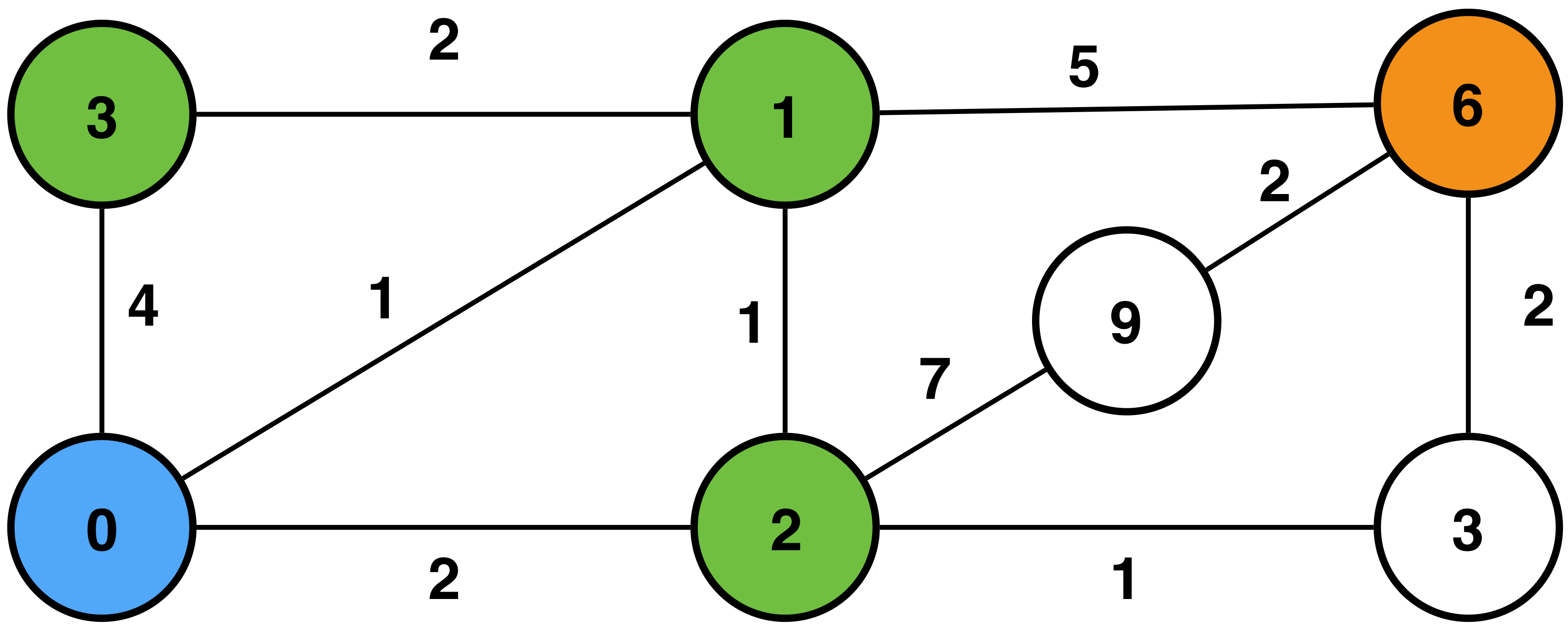
Djikstra’s Algorithm

- Estimate cost-to-come of neighbours



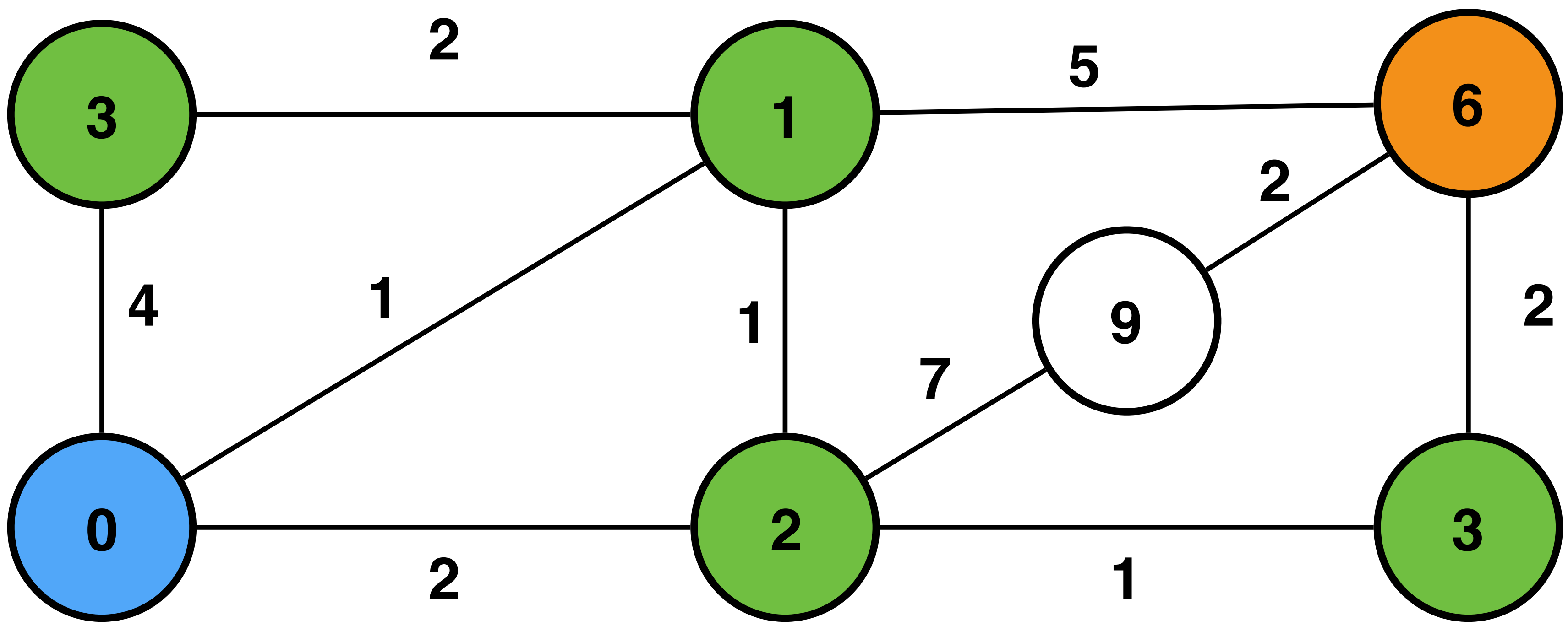
Djikstra's Algorithm

- Visit node with lowest cost-to-come (no unvisited neighbours)



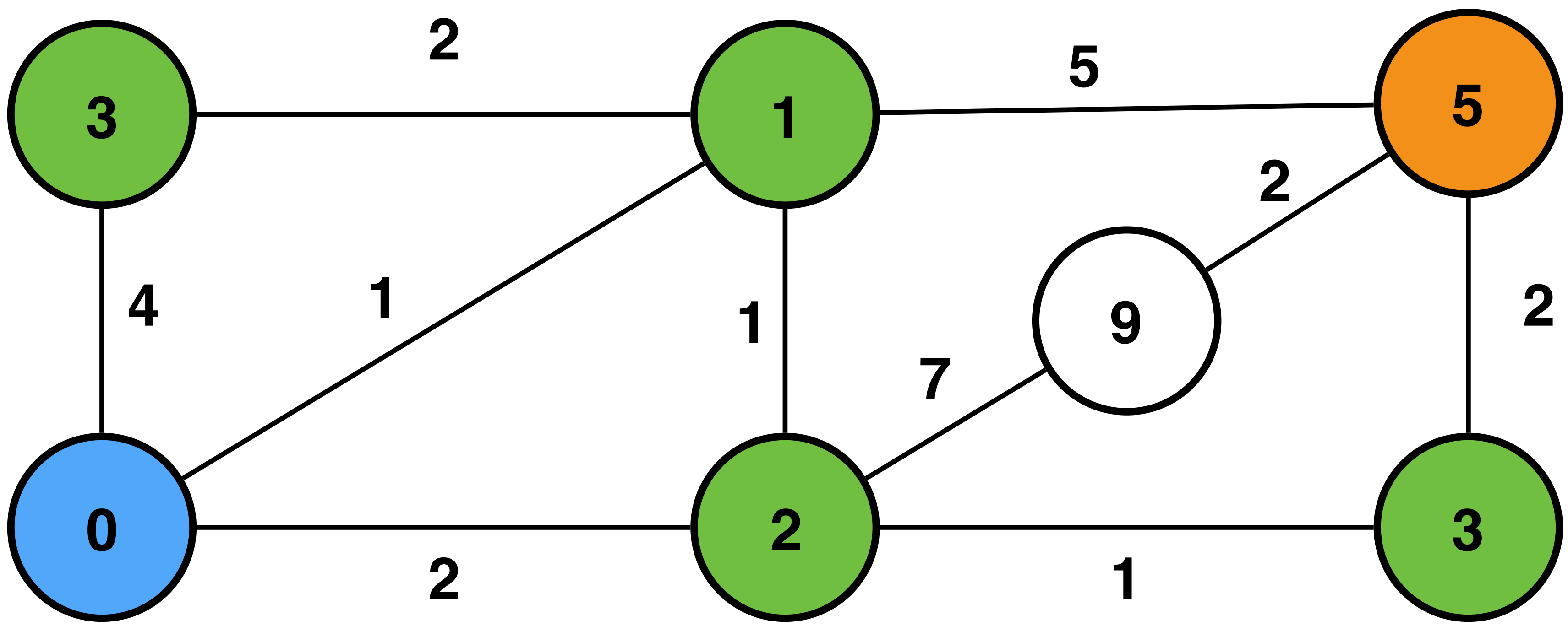
Djikstra’s Algorithm

- Visit node with lowest cost-to-come



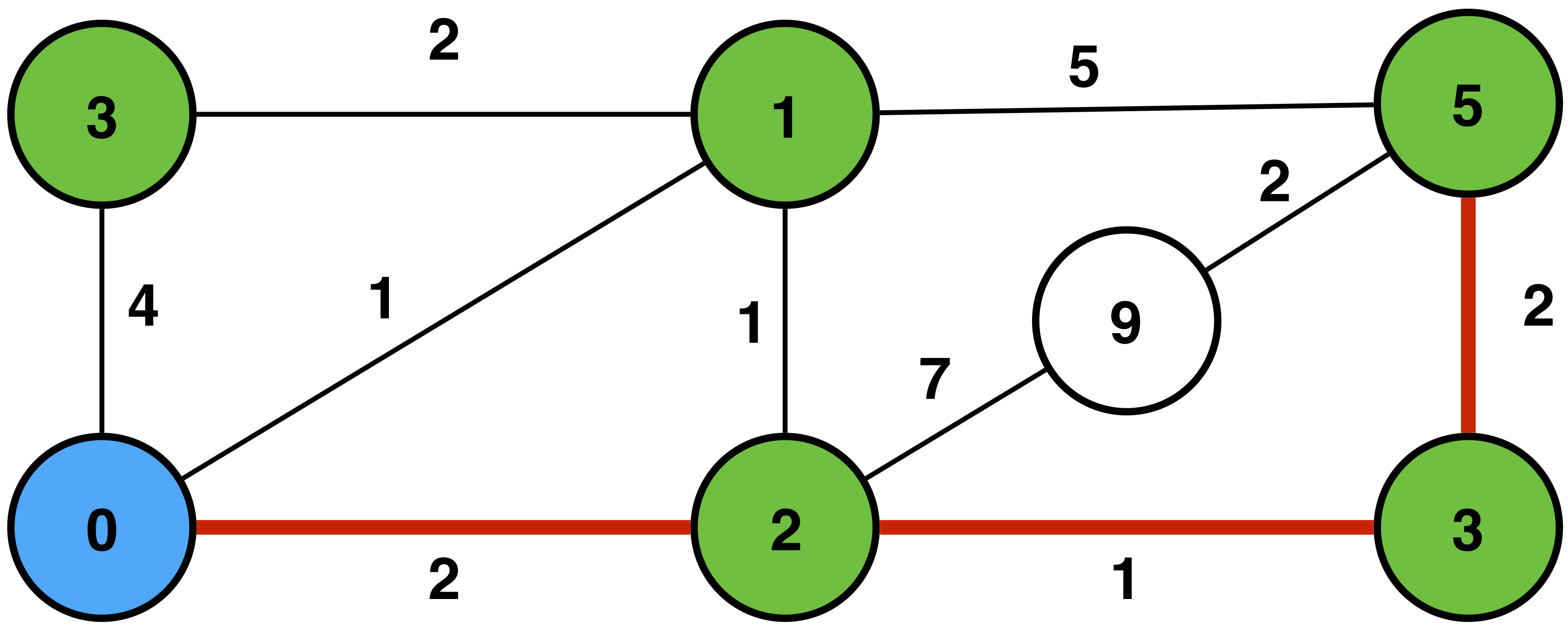
Djikstra's Algorithm

- Estimate cost-to-come of neighbours

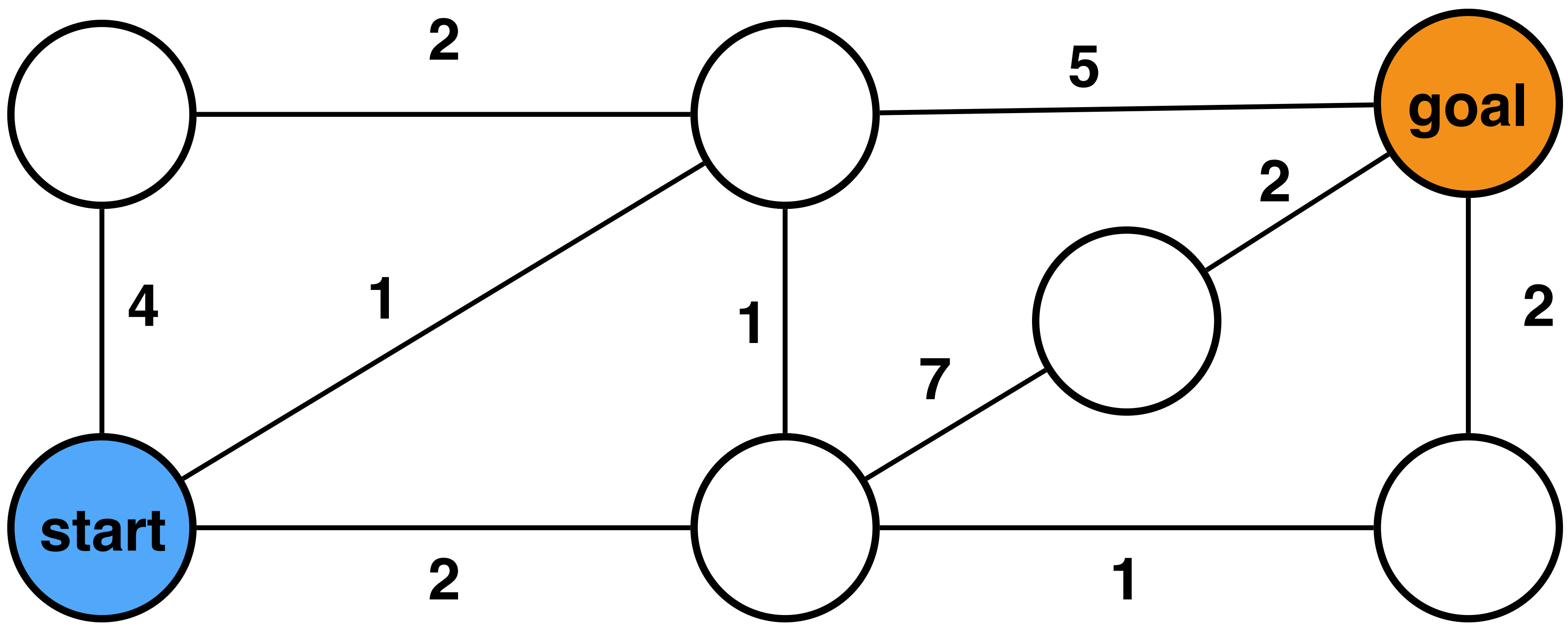


Djikstra's Algorithm

- Visit node with lowest cost-to-come GOAL!!!

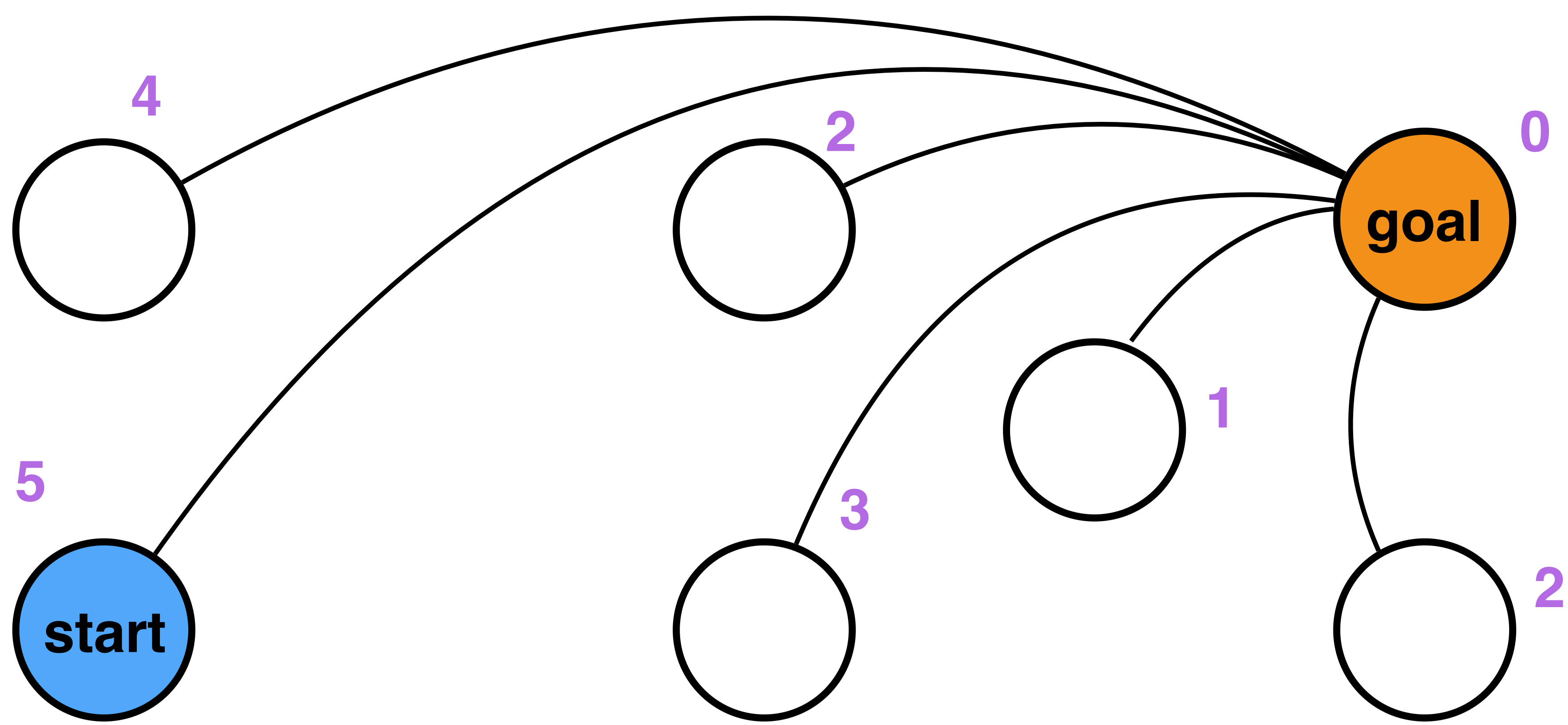


Best-First Algorithm

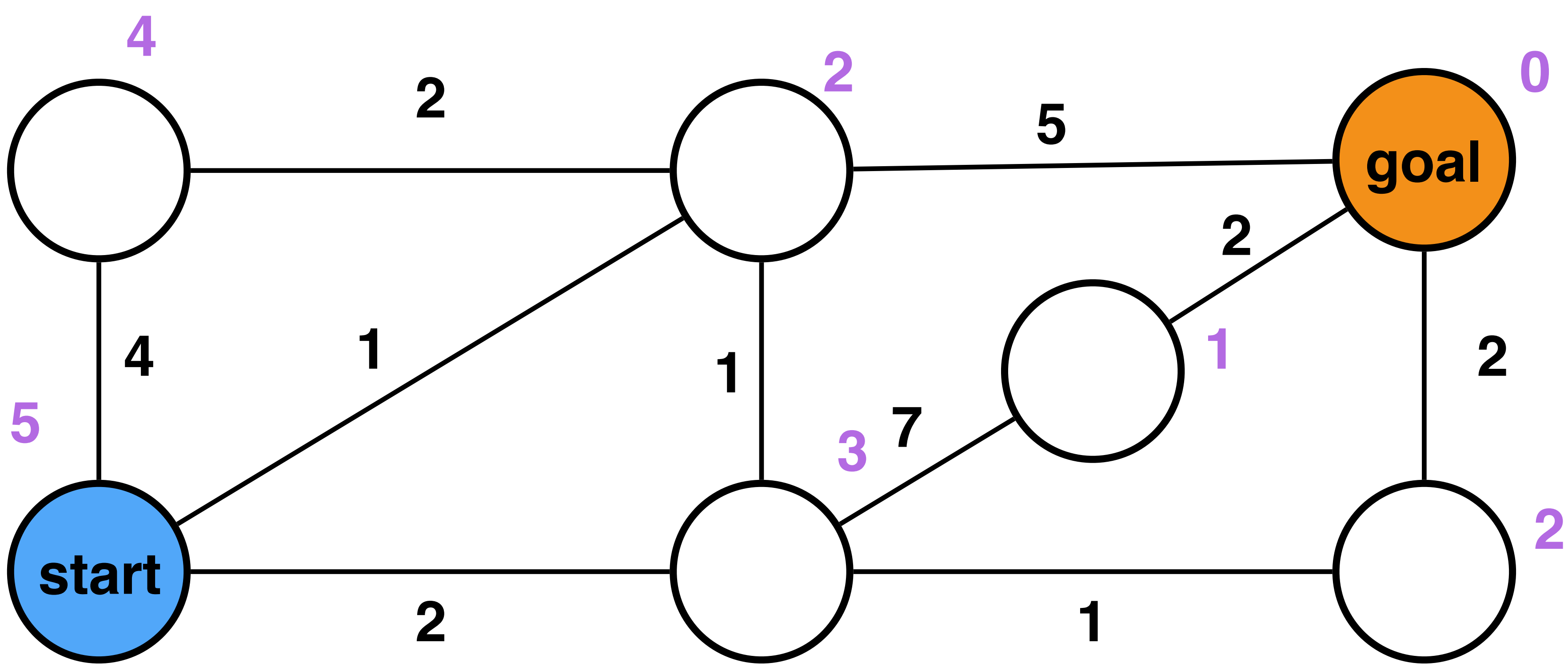


Best-First Algorithm

- Estimate the cost-to-go (can do this online)

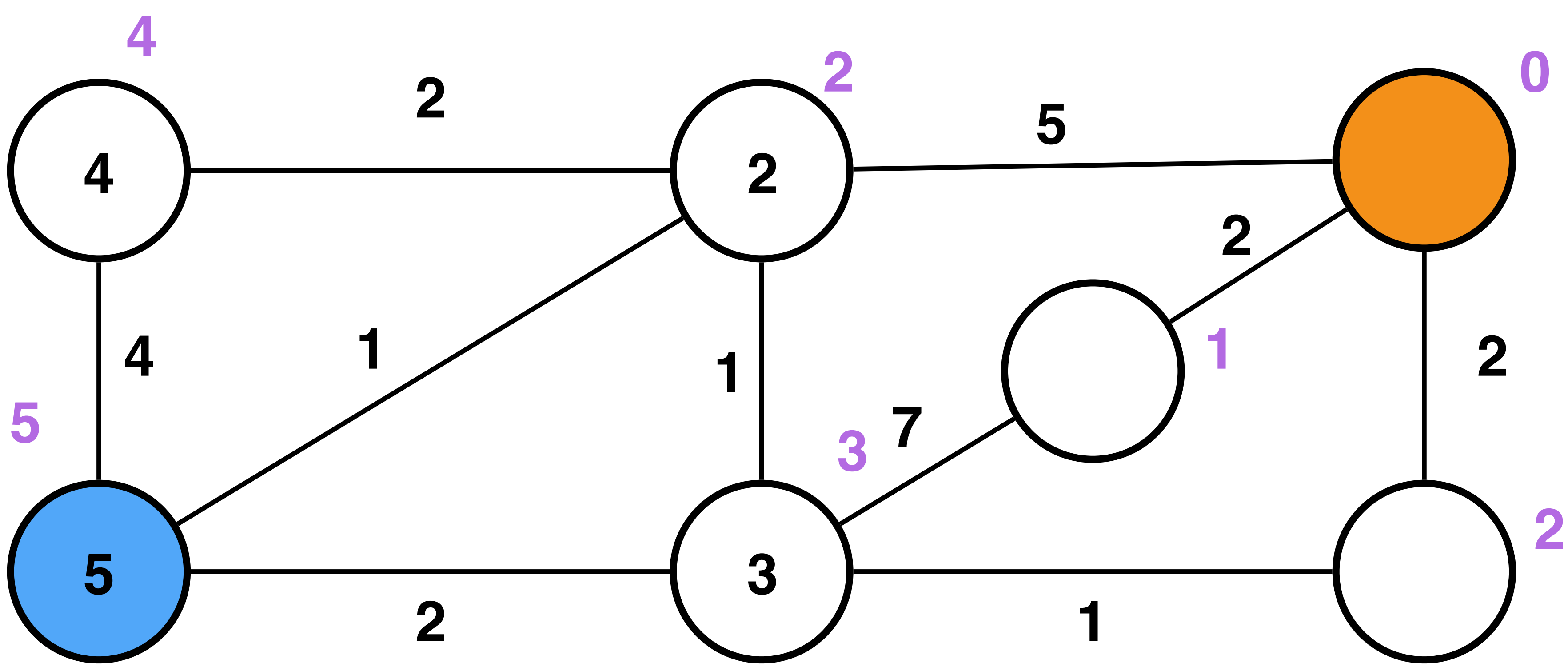


Best-First Algorithm



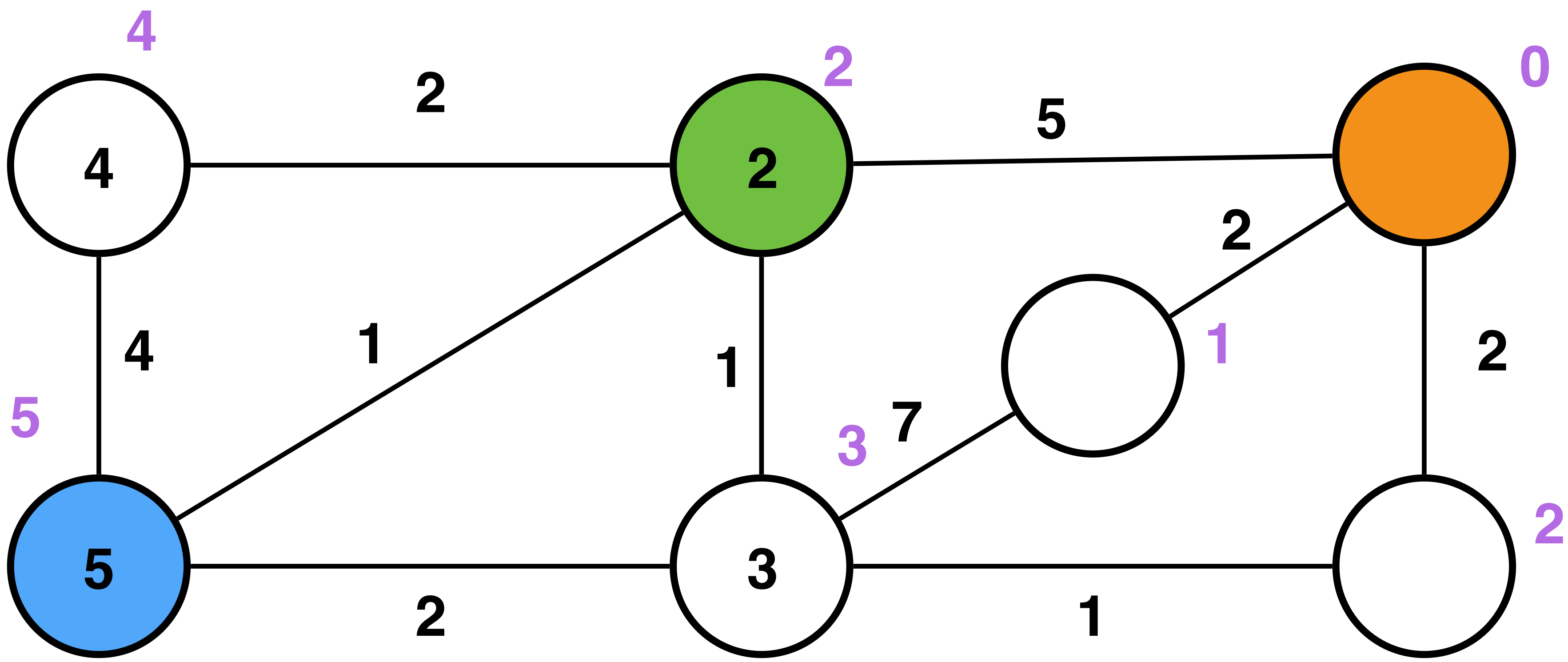
Best-First Algorithm

- Compute cost-to-go estimates of neighbouring states



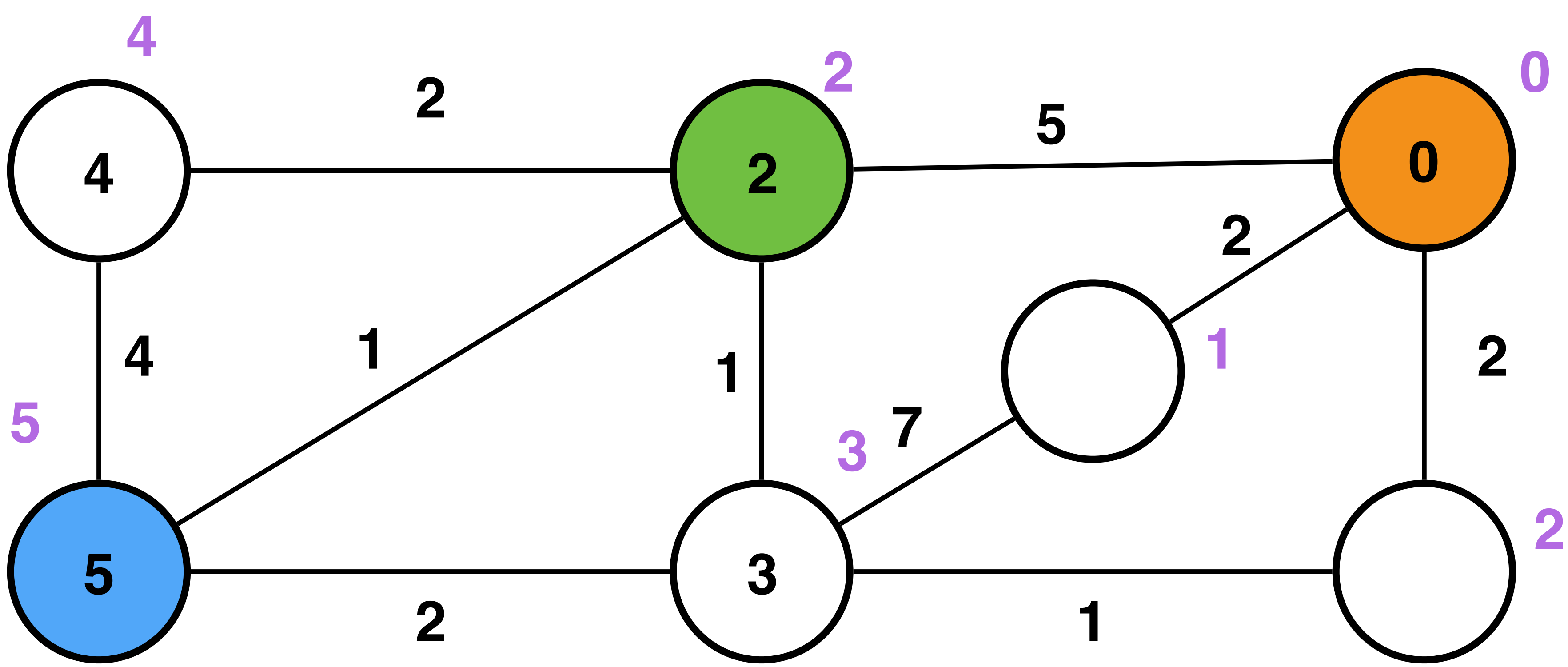
Best-First Algorithm

- Expand to node closest to goal



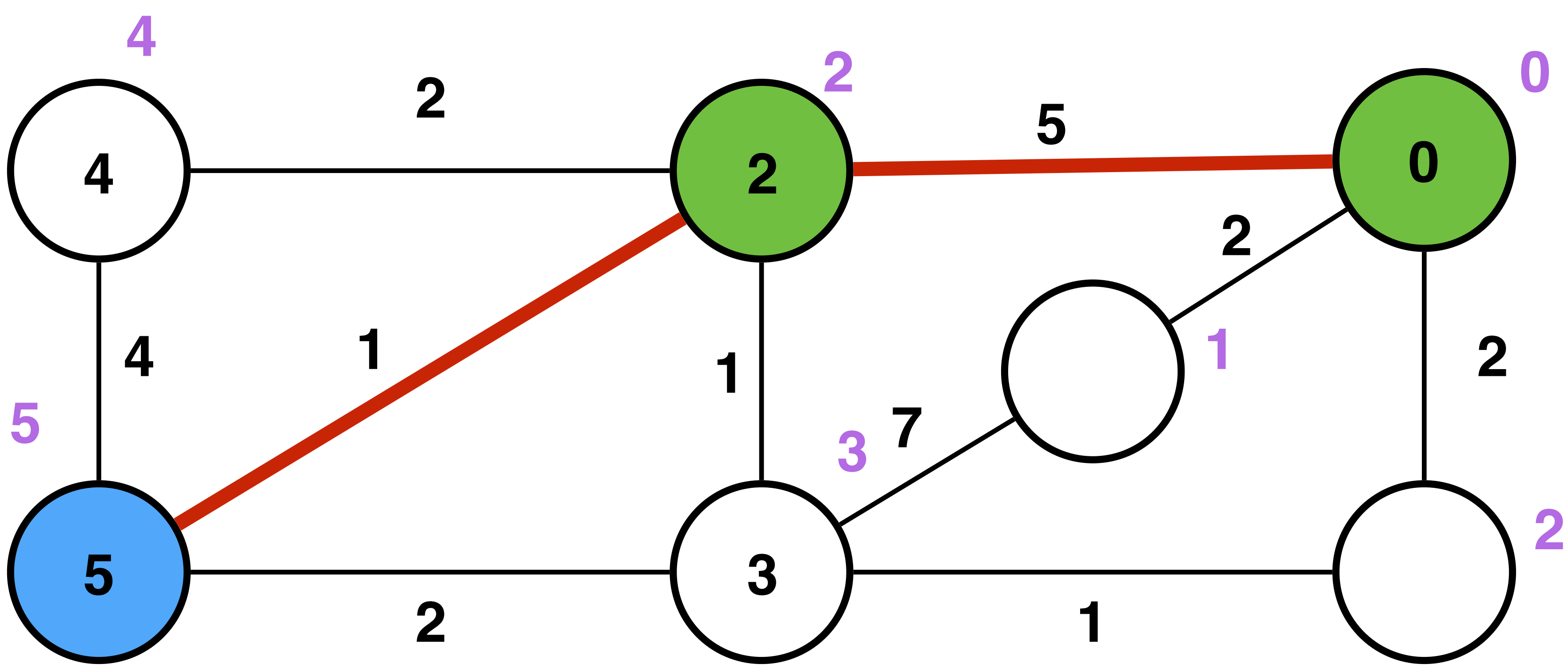
Best-First Algorithm

- Compute cost-to-go estimates

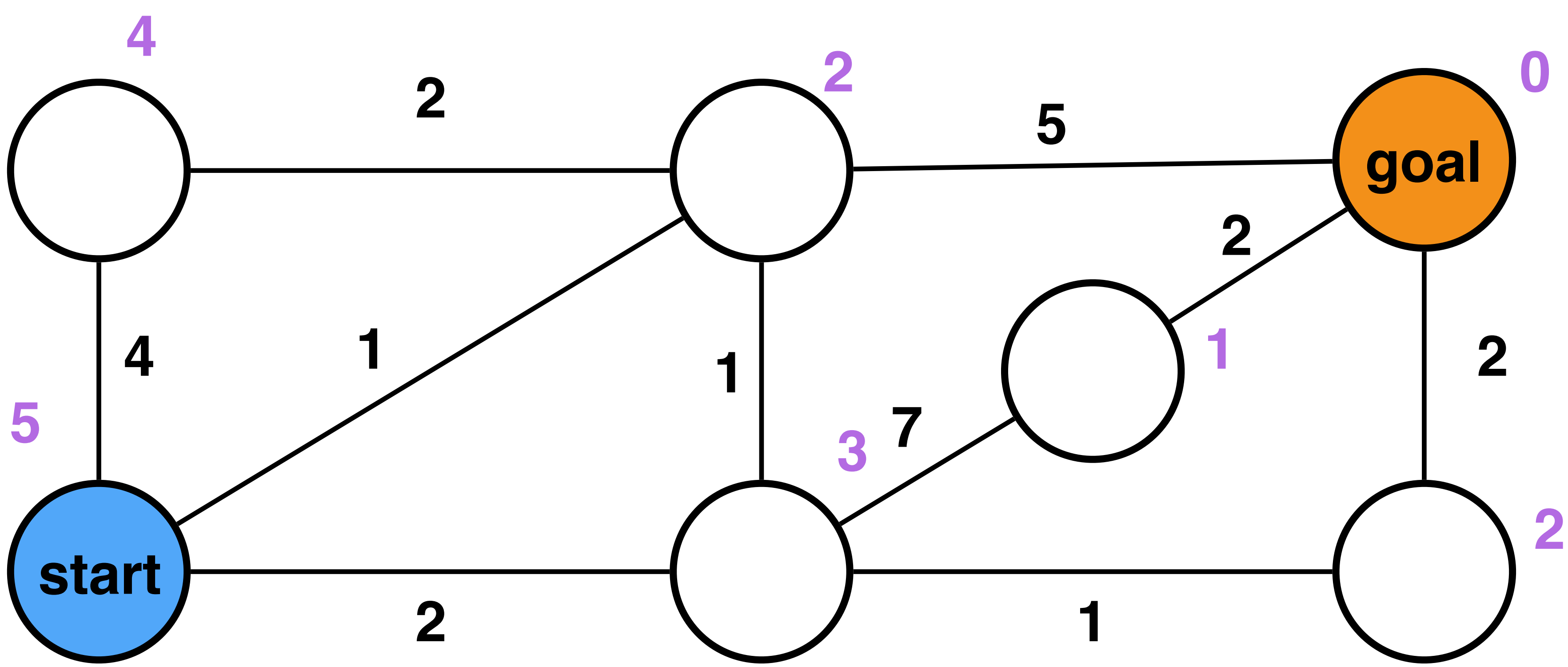


Best-First Algorithm

- Expand to node closest to goal. GOAL!!! (not cheapest)

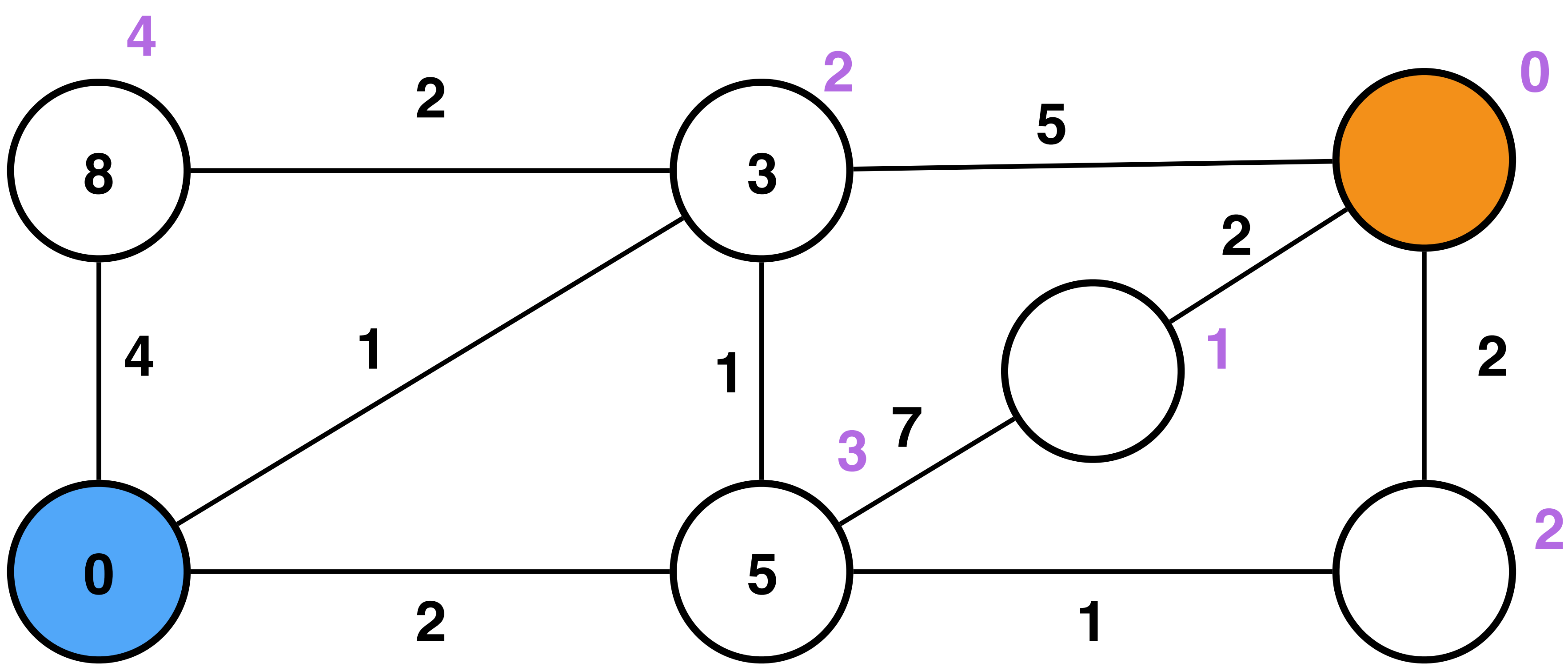


A* Algorithm



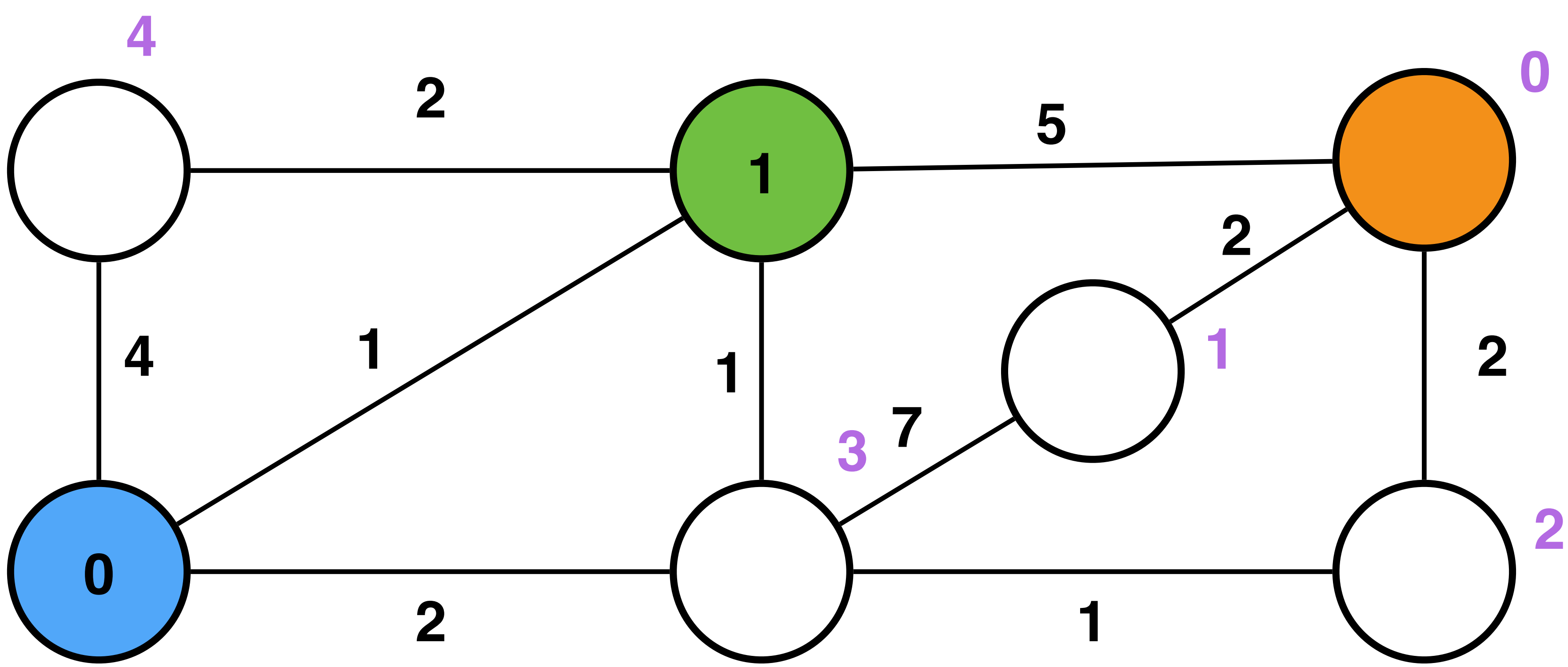
A* Algorithm

- Estimate cost-to-come + cost-to-go



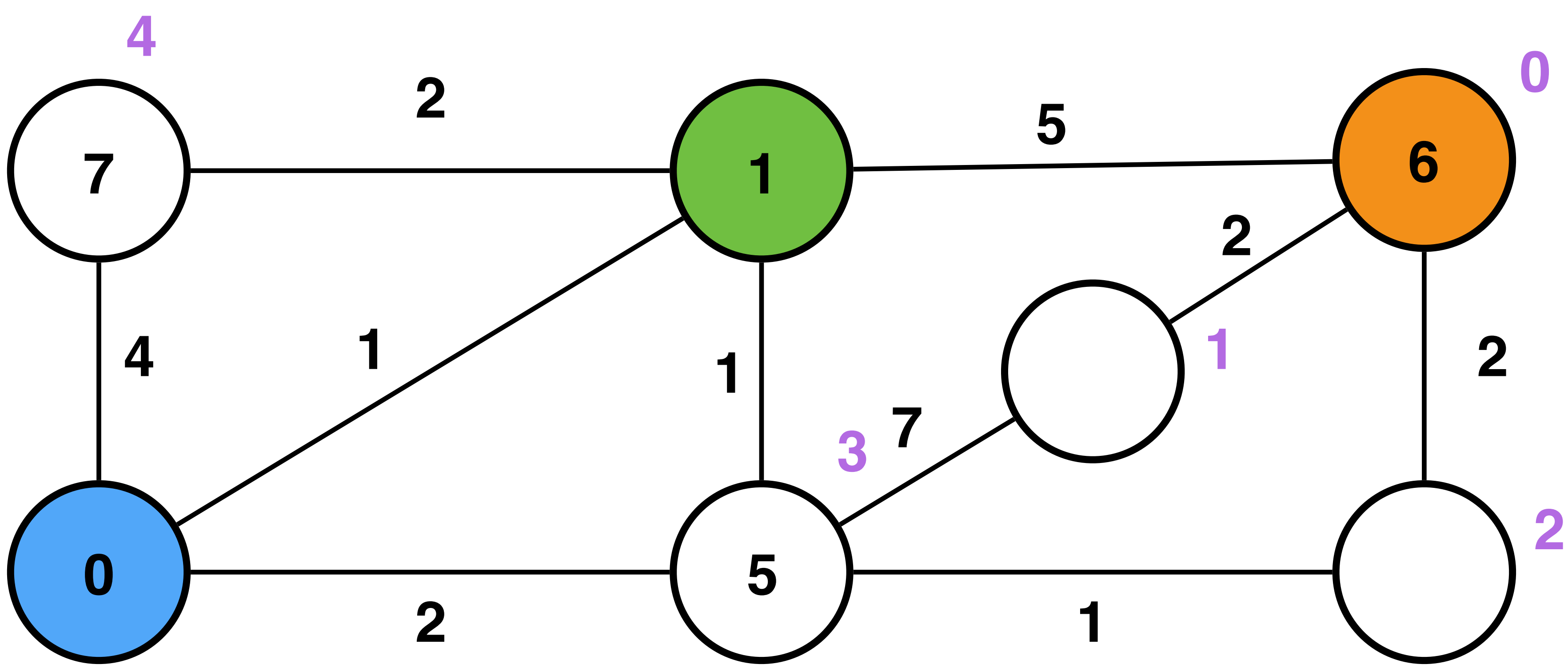
A* Algorithm

- Expand to lowest-value node



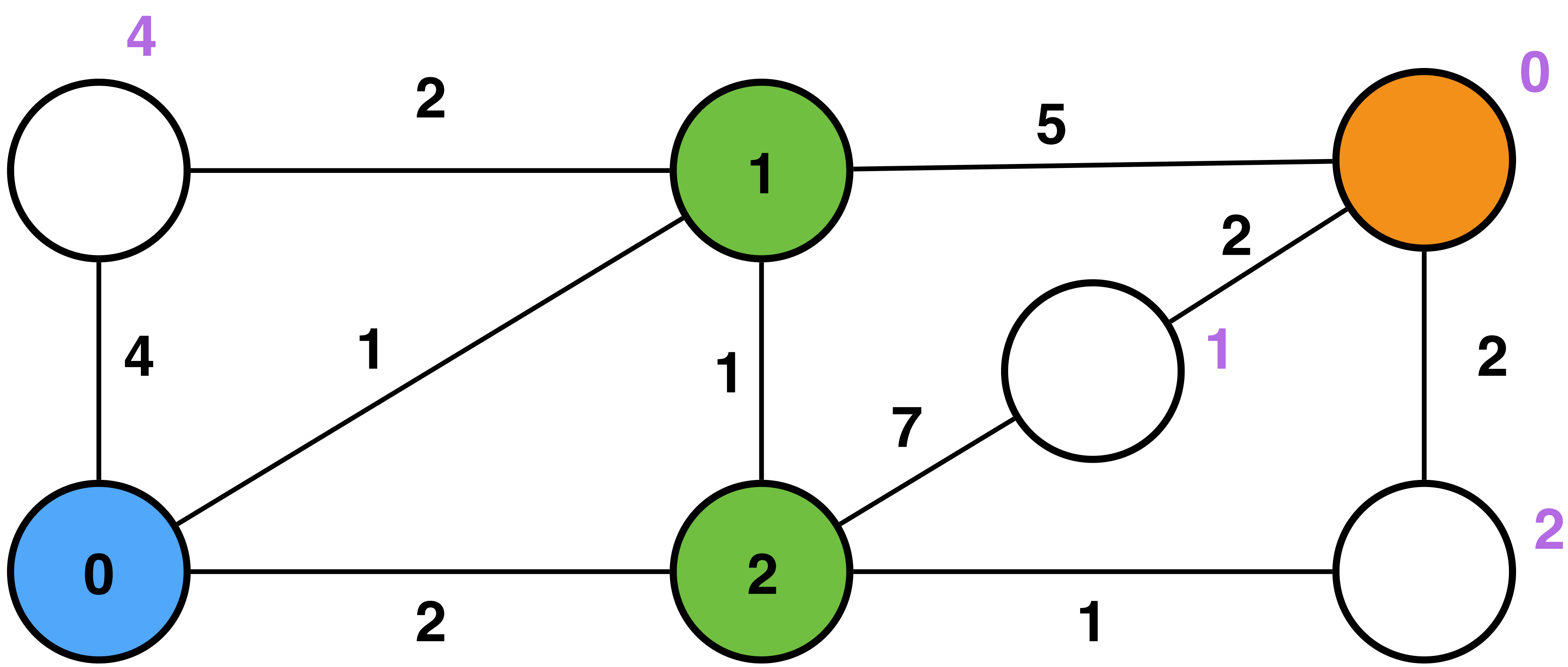
A* Algorithm

- Estimate cost-to-come + cost-to-go



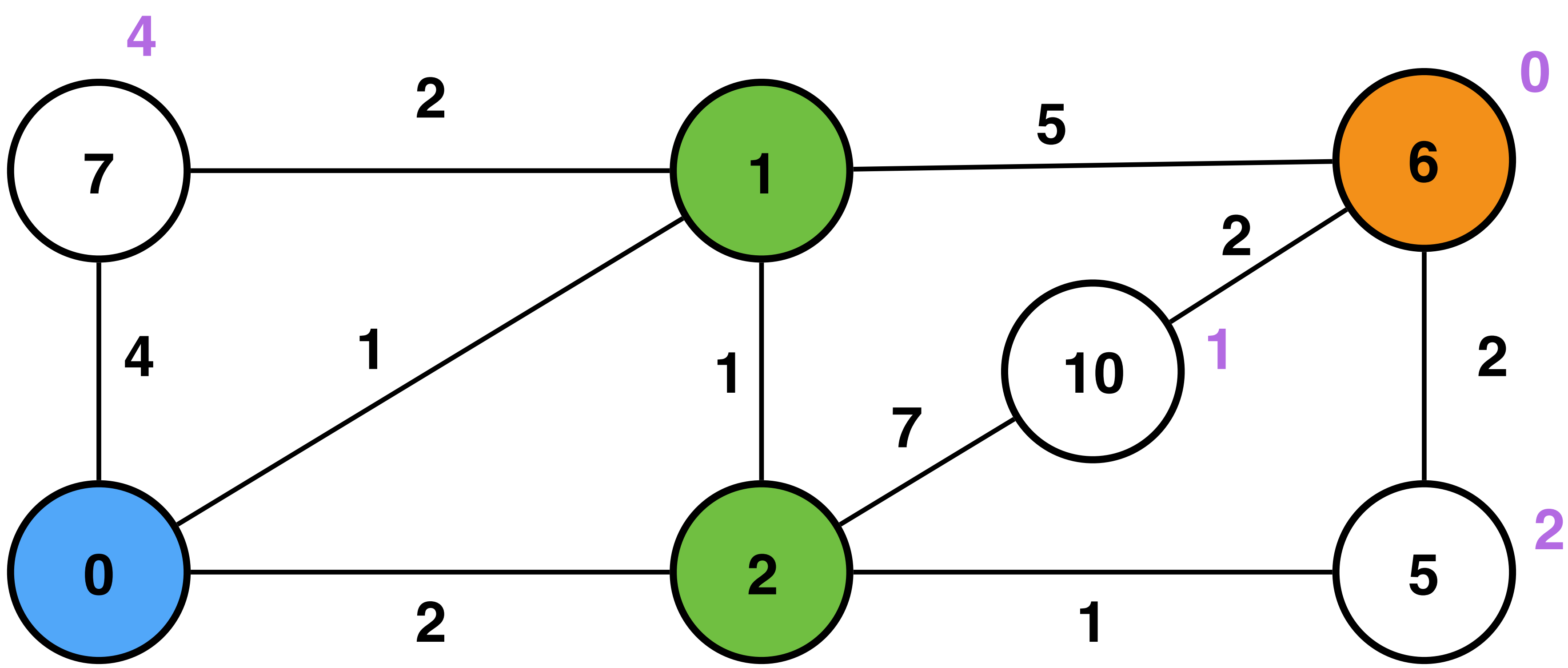
A* Algorithm

- Expand to lowest-value node



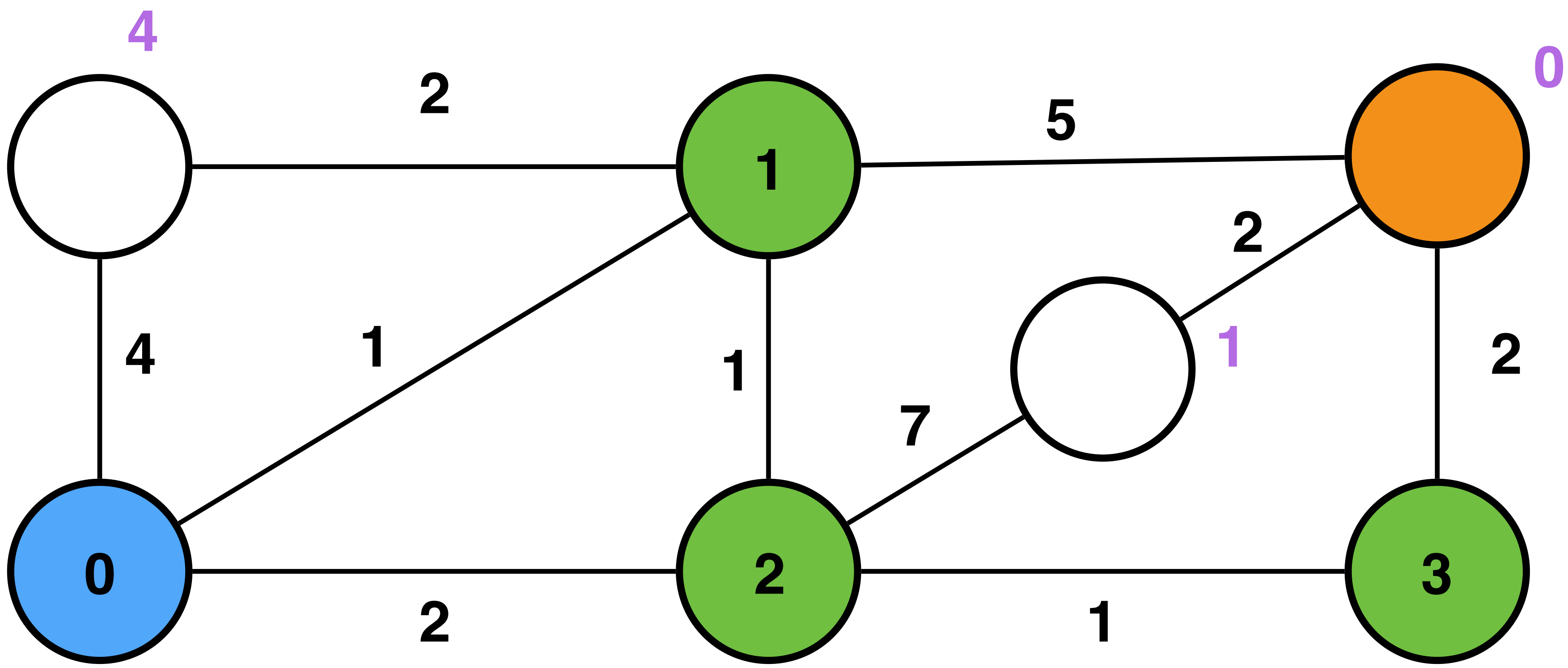
A* Algorithm

- Estimate cost-to-come + cost-to-go



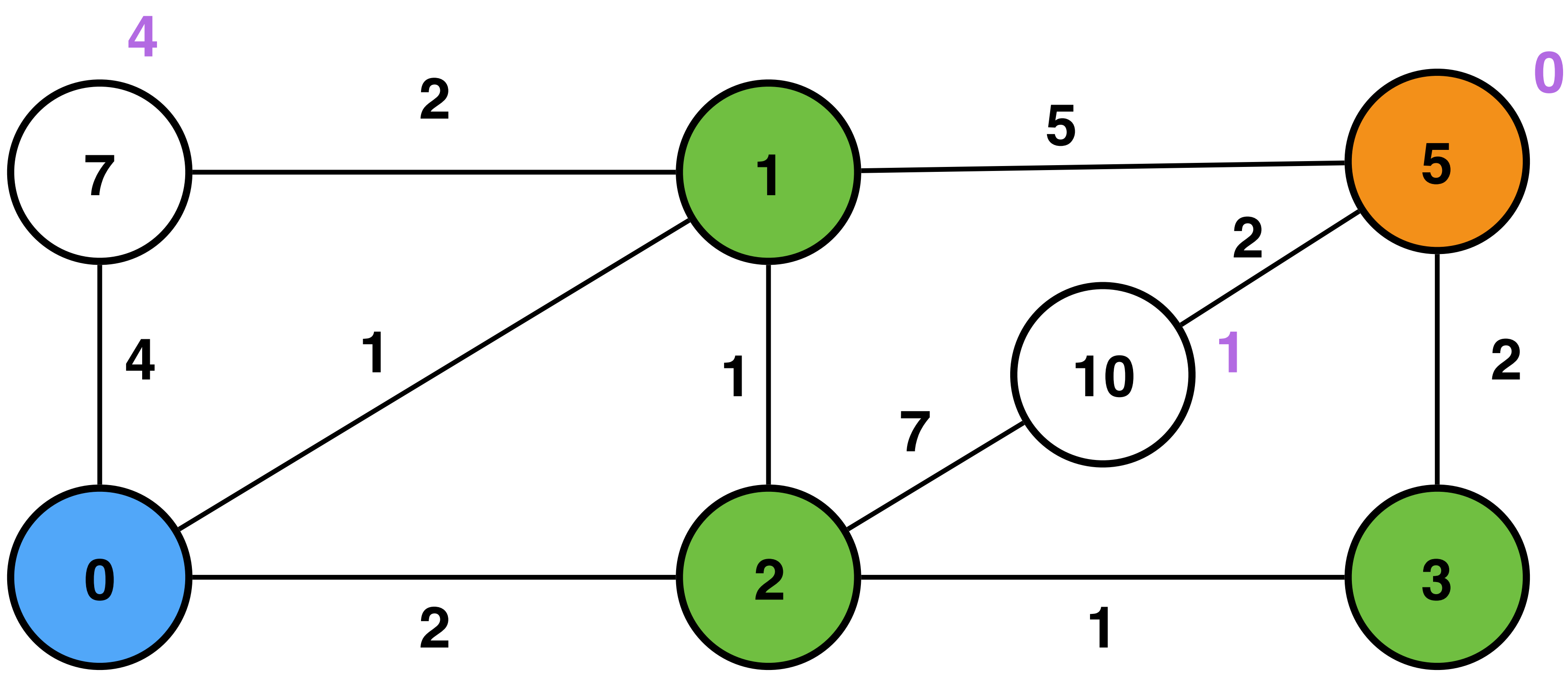
A* Algorithm

- Expand to lowest-value node



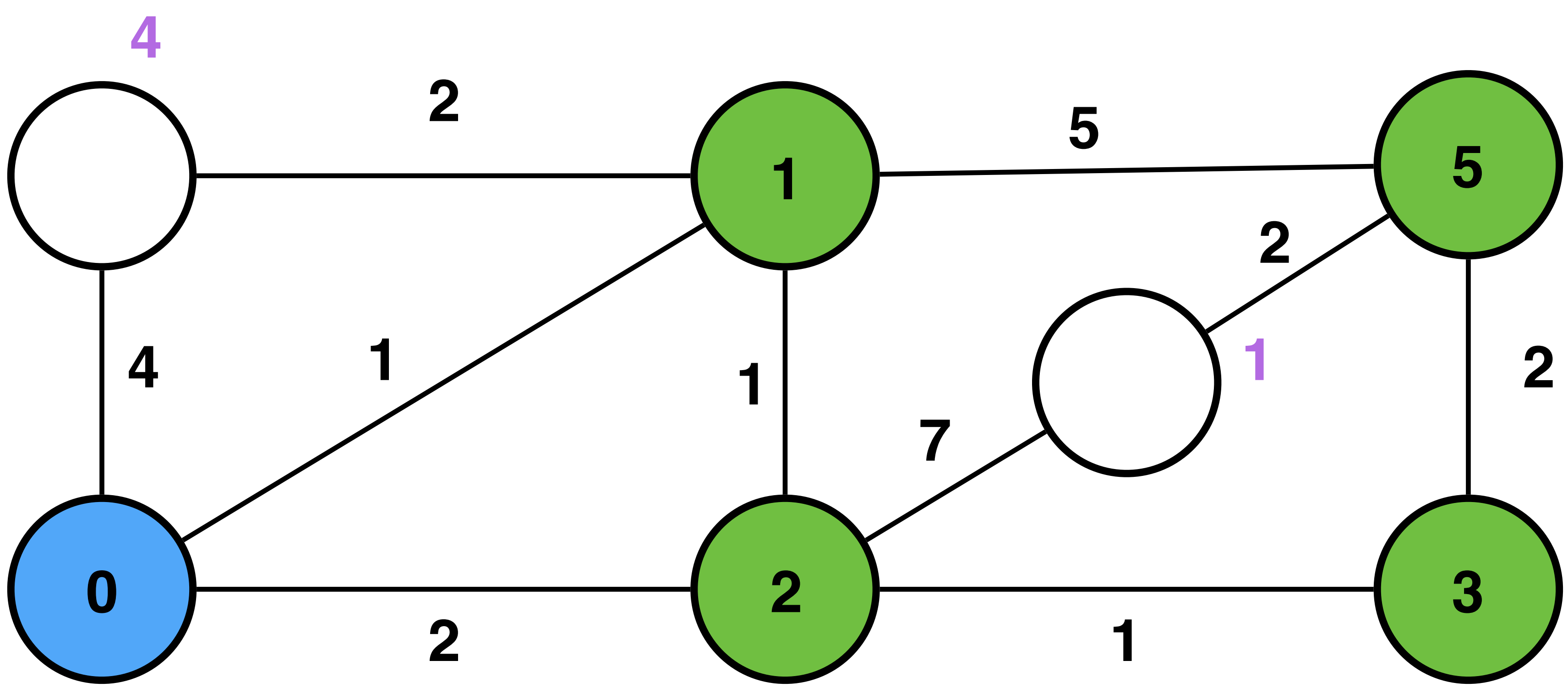
A* Algorithm

- Estimate cost-to-come + cost-to-go



A* Algorithm

- Expand to lowest-value node GOAL!!! Shortest path



Search Algorithms

- Dijkstra's Algorithm

$$g(x)$$

- Best-First Search Algorithm

$$h(x)$$

- A* Algorithm ("A star")

$$g(x) + h(x)$$

Questions?