FAI LAB 02b
Uninformed Search

Paolo Morettin

2024-25

Code available

You can find the **code** of the lab sessions on GitHub:

`https://github.com/paolomorettin/FAI-code`

Problem-solving agents

- a kind of **goal/utility-oriented** agents
- **atomic** representations of the states
- if the environment is
  - fully observable
  - deterministic
  - static

  ...we can plan once and execute the actions sequentially without checking their effect on the environment (**open-loop** system)

Search problems vs. search algorithms

Search problems:

- graphical representation of the **search space**:
  - nodes are the atomic states of the problem
  - edges represent actions (w/ costs)
  - an initial state
  - one or more goal states
- the search space is **implicit**, possibly **infinite**

Search problems vs. search algorithms

Search problems:

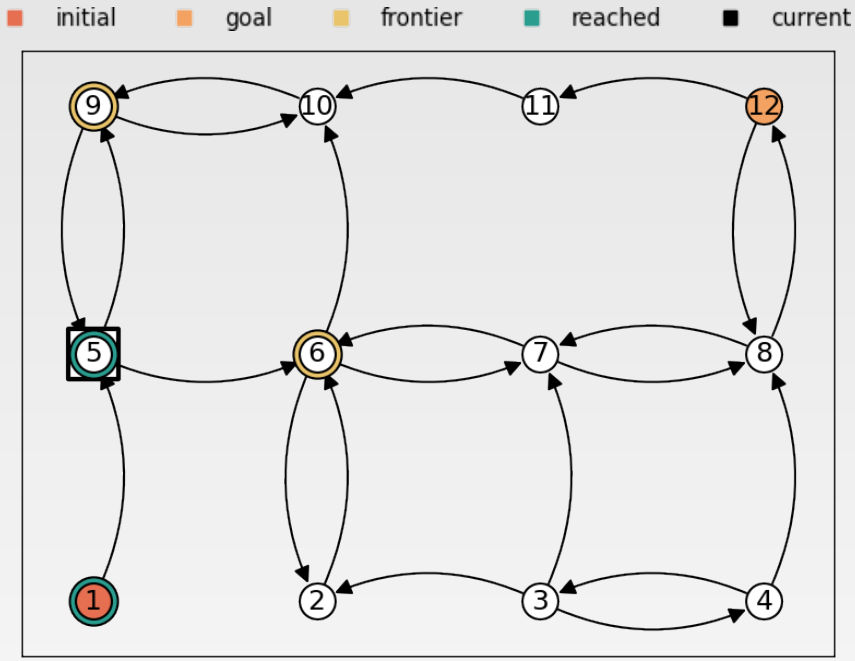
- graphical representation of the **search space**:
  - nodes are the atomic states of the problem
  - edges represent actions (w/ costs)
  - an initial state
  - one or more goal states
- the search space is **implicit**, possibly **infinite**

Search algorithms:

- computer programs that explore of the search space by building a **search tree/DAG**
- the nodes in this structure is **explicitly** encoded in memory
- ***REMEMBER!*** **state** $\neq$ **node** (in the search T/DAG)

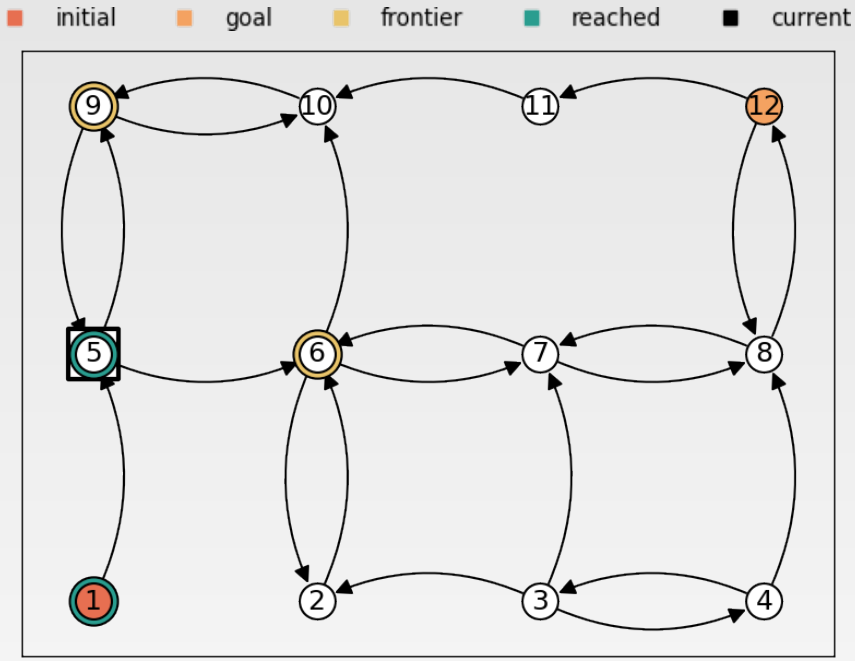
Uninformed search algorithms

uninformed = no prior knowledge when exploring the search space



Uninformed search algorithms

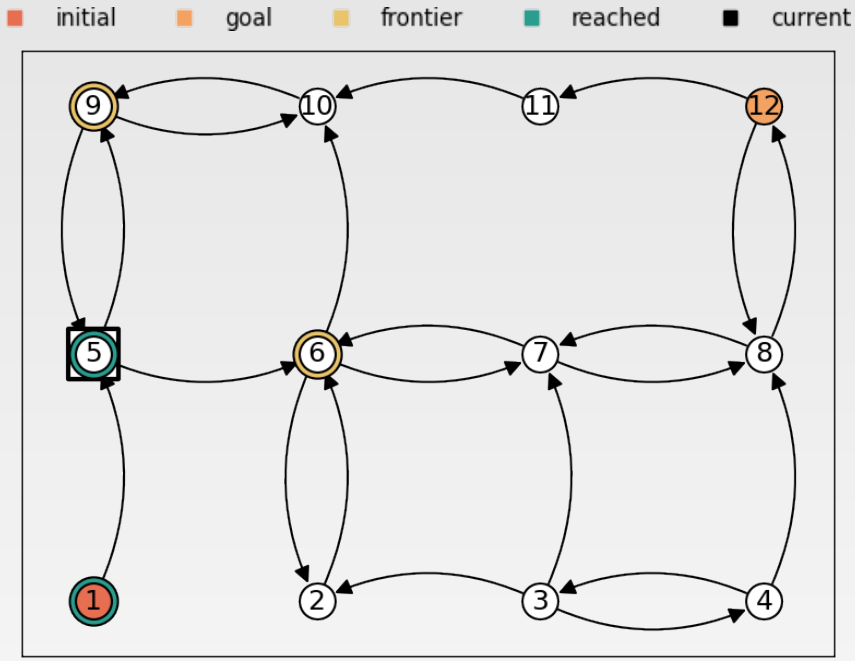
uninformed = no prior knowledge when exploring the search space



- **EXPANSION**: exploring the neighborhood of states reachable by a node
- **GENERATION**: allocating memory for a new node

Uninformed search algorithms

uninformed = no prior knowledge when exploring the search space



- **tree-like vs. graph-like search**: can states be visited multiple times? $\rightarrow$ extra memory cost!
- **FRONTIER**: partition states into explored/unexplored
- **REACHED**: states that were explored at some point

Best-first search

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
    node ← NODE(STATE=problem.INITIAL)
    frontier ← a priority queue ordered by f, with node as an element
    reached ← a lookup table, with one entry with key problem.INITIAL and value node
    while not IS-EMPTY(frontier) do
        node ← POP(frontier)
        if problem.IS-GOAL(node.STATE) then return node
        for each child in EXPAND(problem, node) do
            s ← child.STATE
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
                reached[s] ← child
                add child to frontier
    return failure

function EXPAND(problem, node) yields nodes
    s ← node.STATE
    for each action in problem.ACTIONS(s) do
        s' ← problem.RESULT(s, action)
        cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
        yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Best-first search as a template

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution node, or *failure*
  return BEST-FIRST-SEARCH(*problem*, PATH-COST)

function **BREADTH-FIRST SEARCH**(*problem*) **returns** a solution node, or *failure*
  return BEST-FIRST-SEARCH(*problem*, **DEPTH**)

function **DEPTH-FIRST SEARCH**(*problem*) **returns** a solution node, or *failure*
  return BEST-FIRST-SEARCH(*problem*, **- DEPTH**)

Breadth-first search

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

Depth-first and iterative deepening

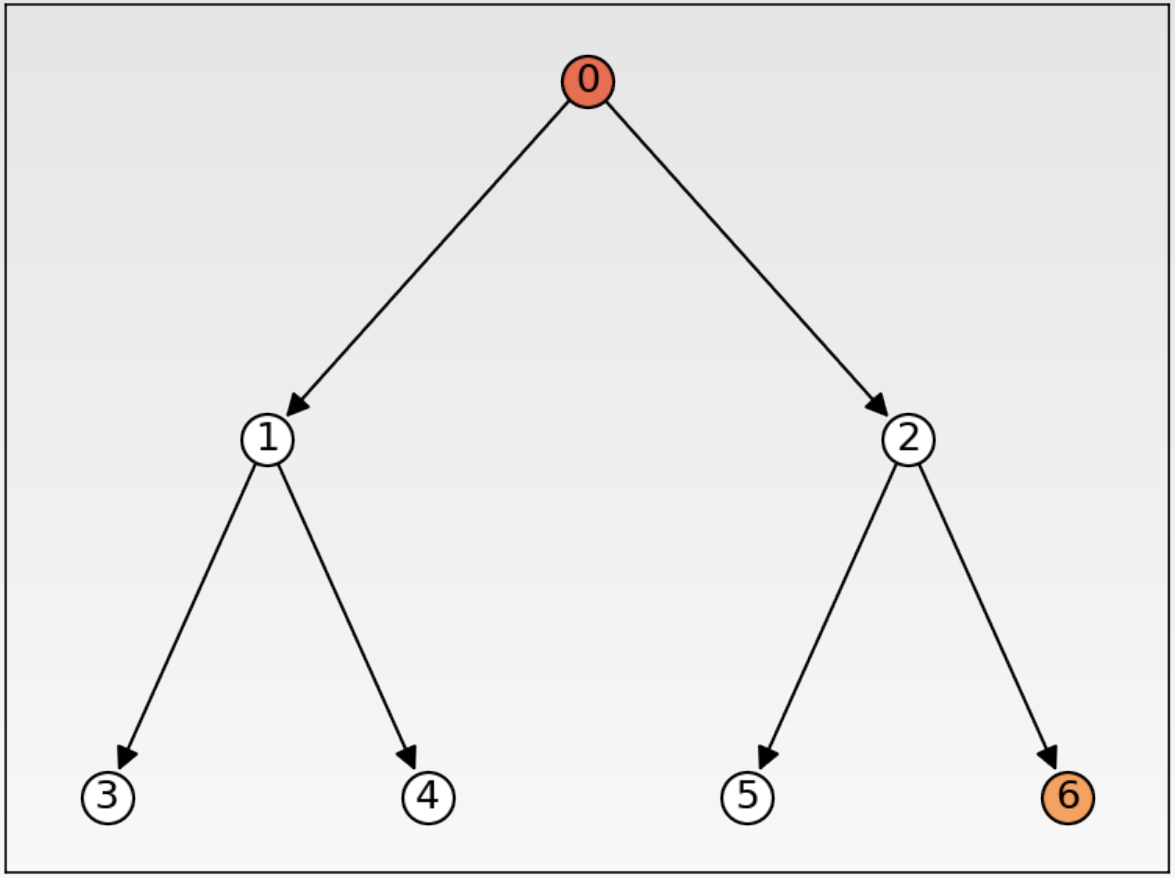
```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution node or failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result

function DEPTH-LIMITED-SEARCH(problem, ℓ) returns a node or failure or cutoff
  frontier ← a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```

R&N 4th ed. check *after* inserting nodes in *frontier*.

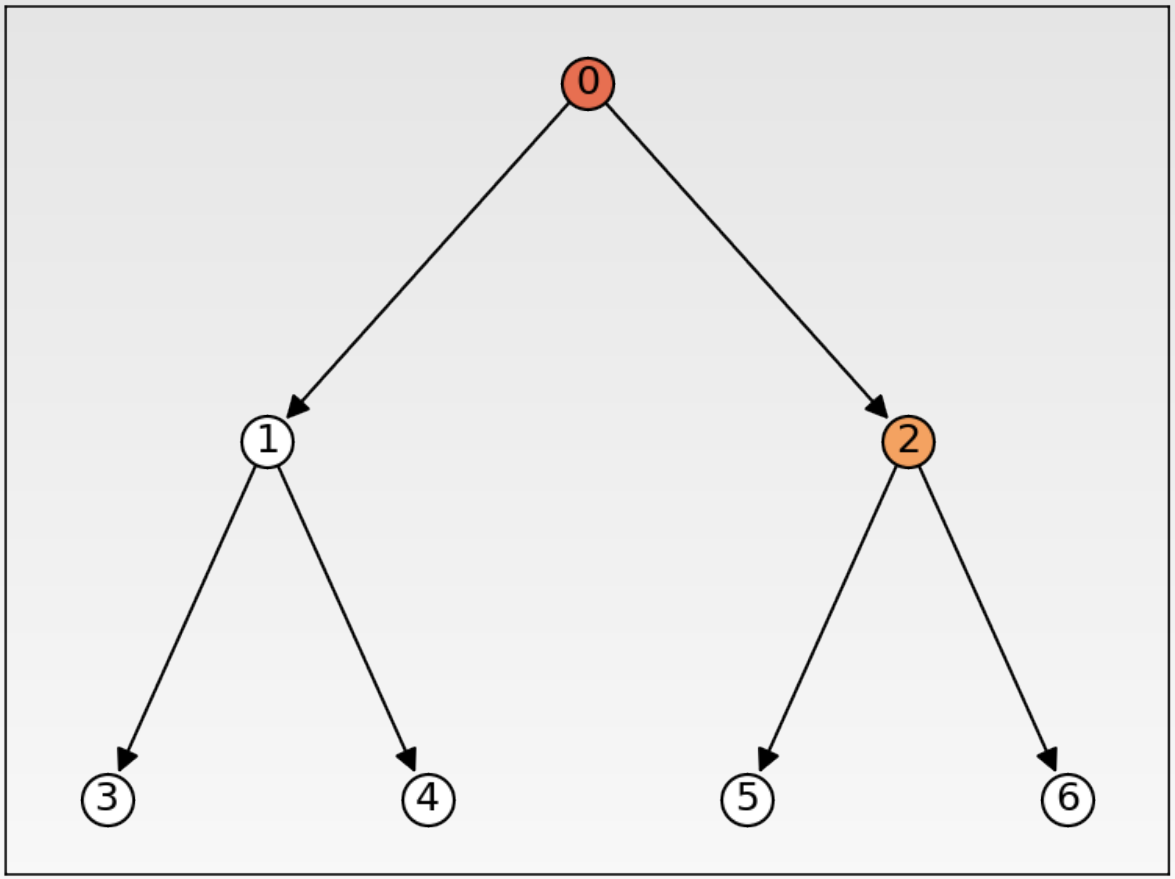
We stick to the previous version.

Quick questions 1



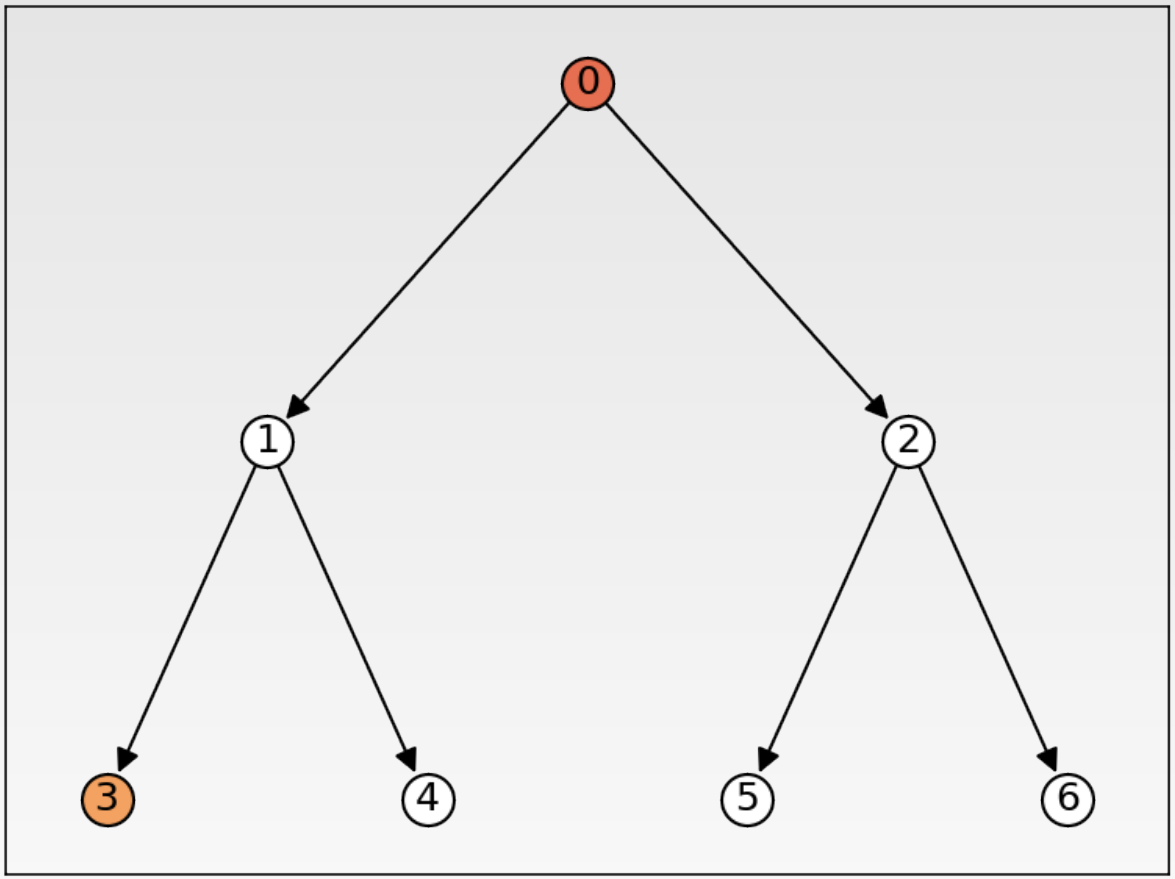
1.1) BFS? **1.2)** DFS? **1.3)** Uniform-cost?

Quick questions 2



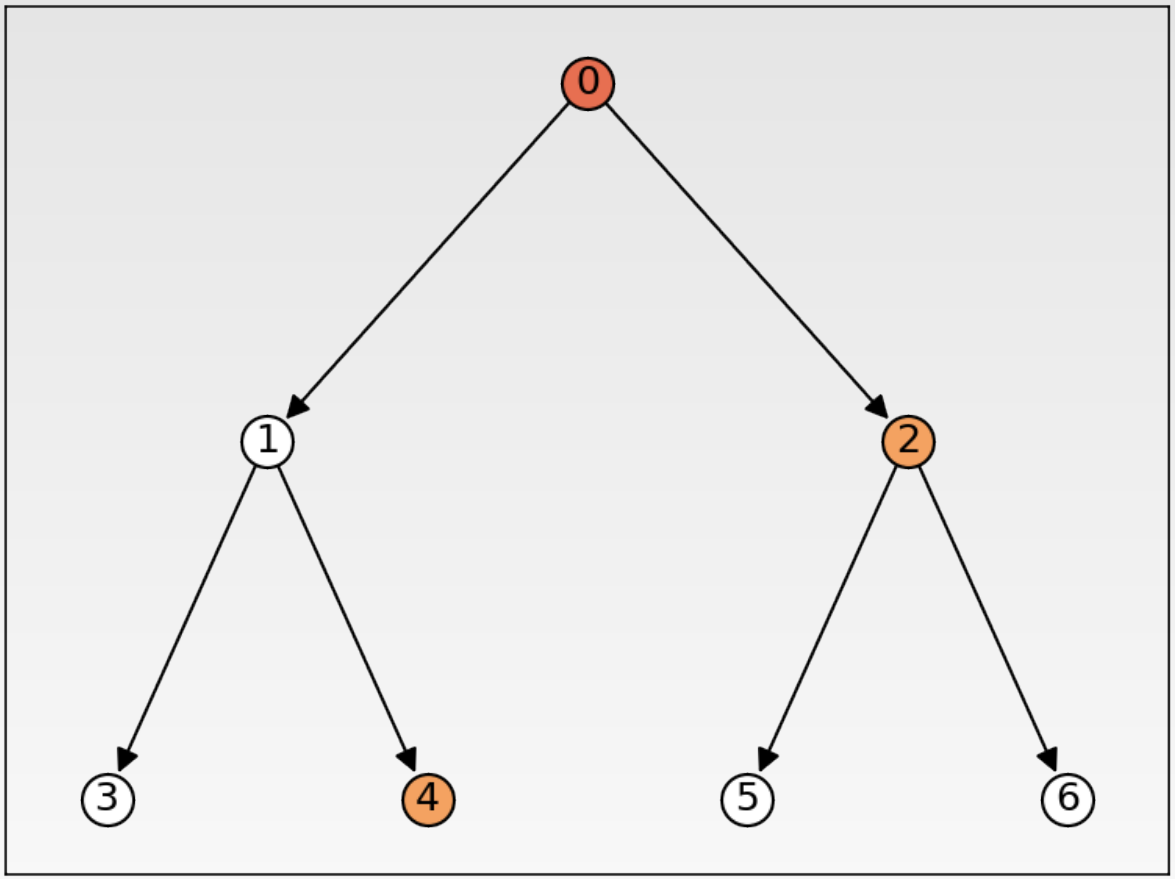
2.1) BFS? **2.2)** DFS? **2.3)** Uniform-cost?

Quick questions 3



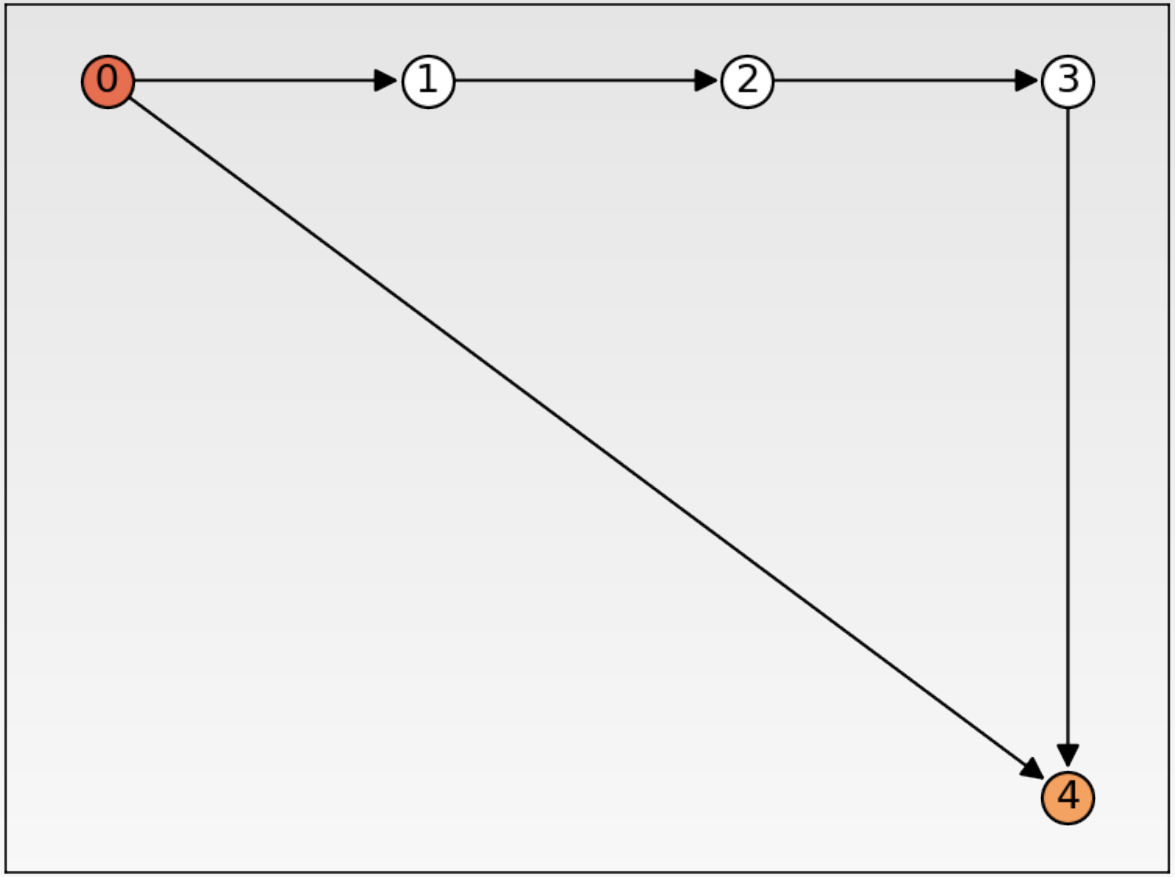
3.1) BFS? **3.2)** DFS? **3.3)** Uniform-cost?

Quick questions 4



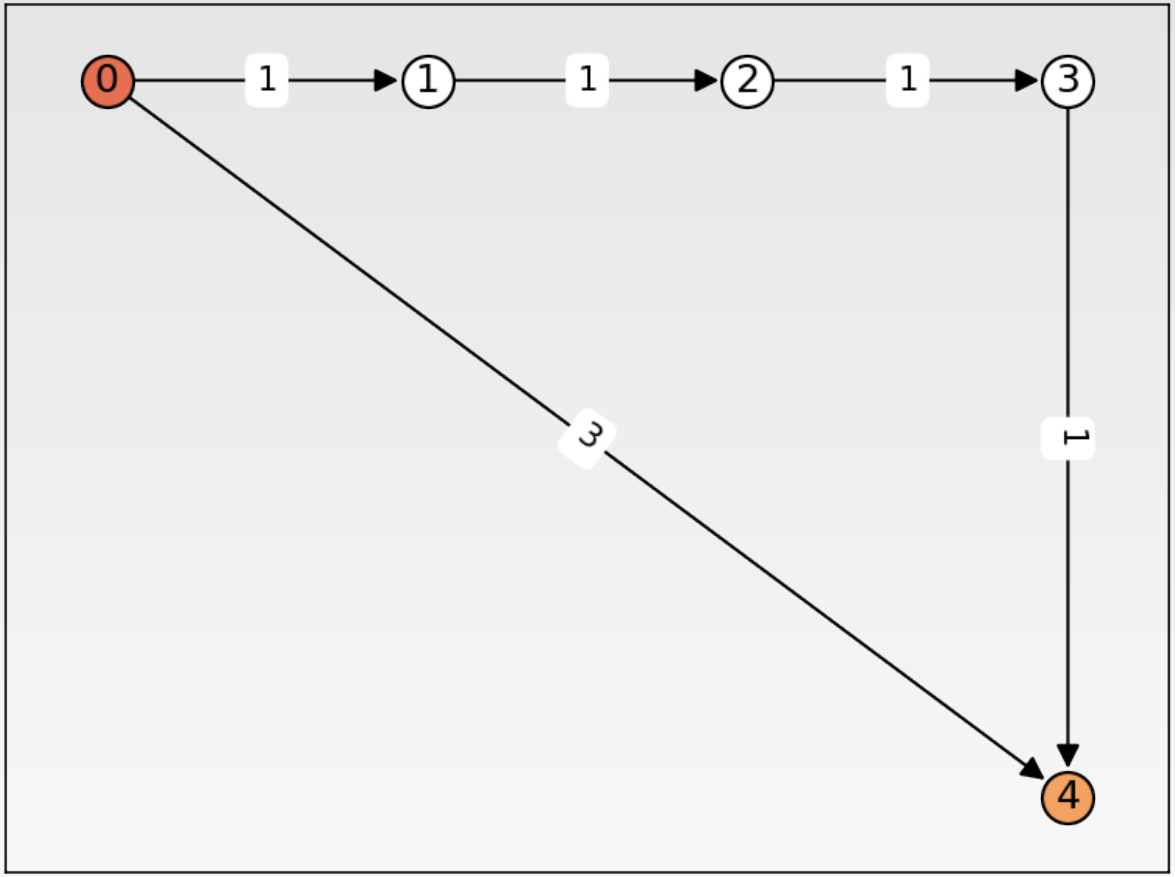
4.1) BFS? **4.2)** DFS? **4.3)** Uniform-cost?

Quick questions 5



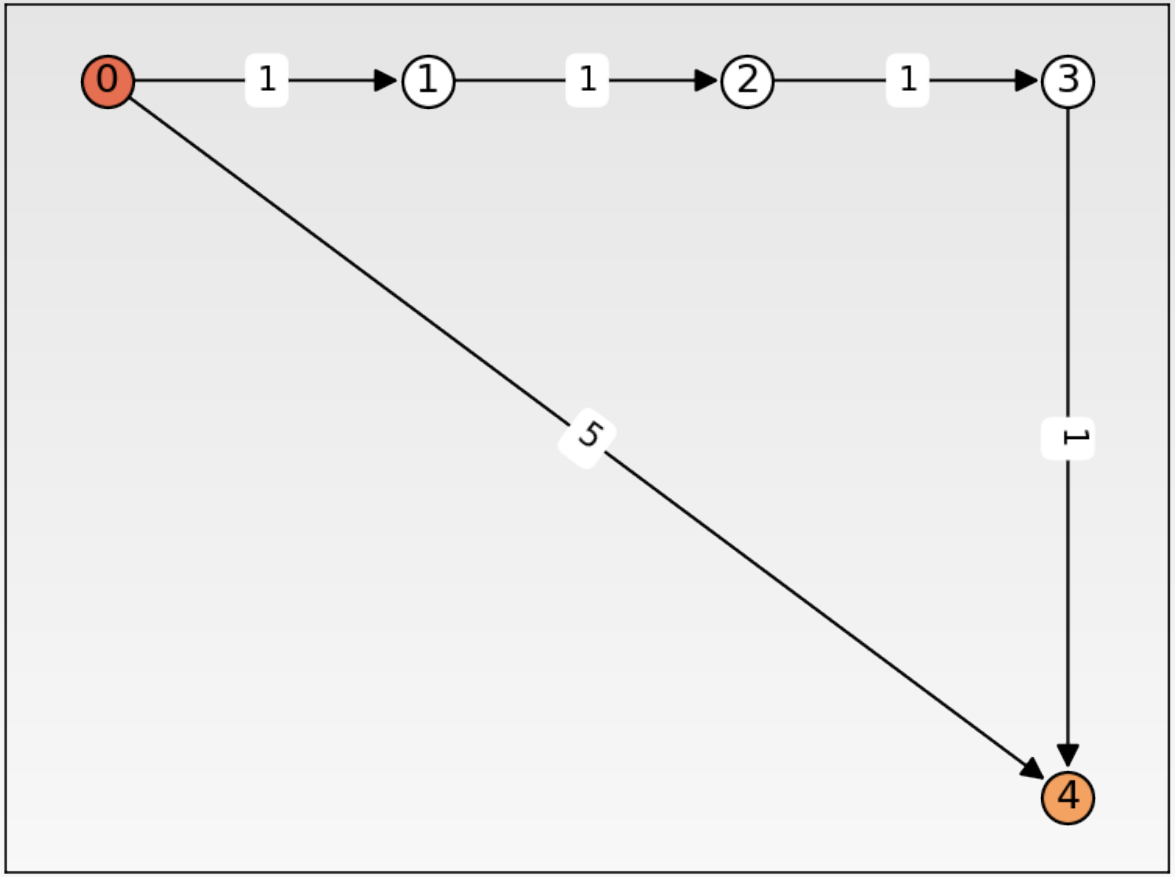
5.1) BFS? **5.2)** DFS? **5.3)** Uniform-cost?

Quick questions 6



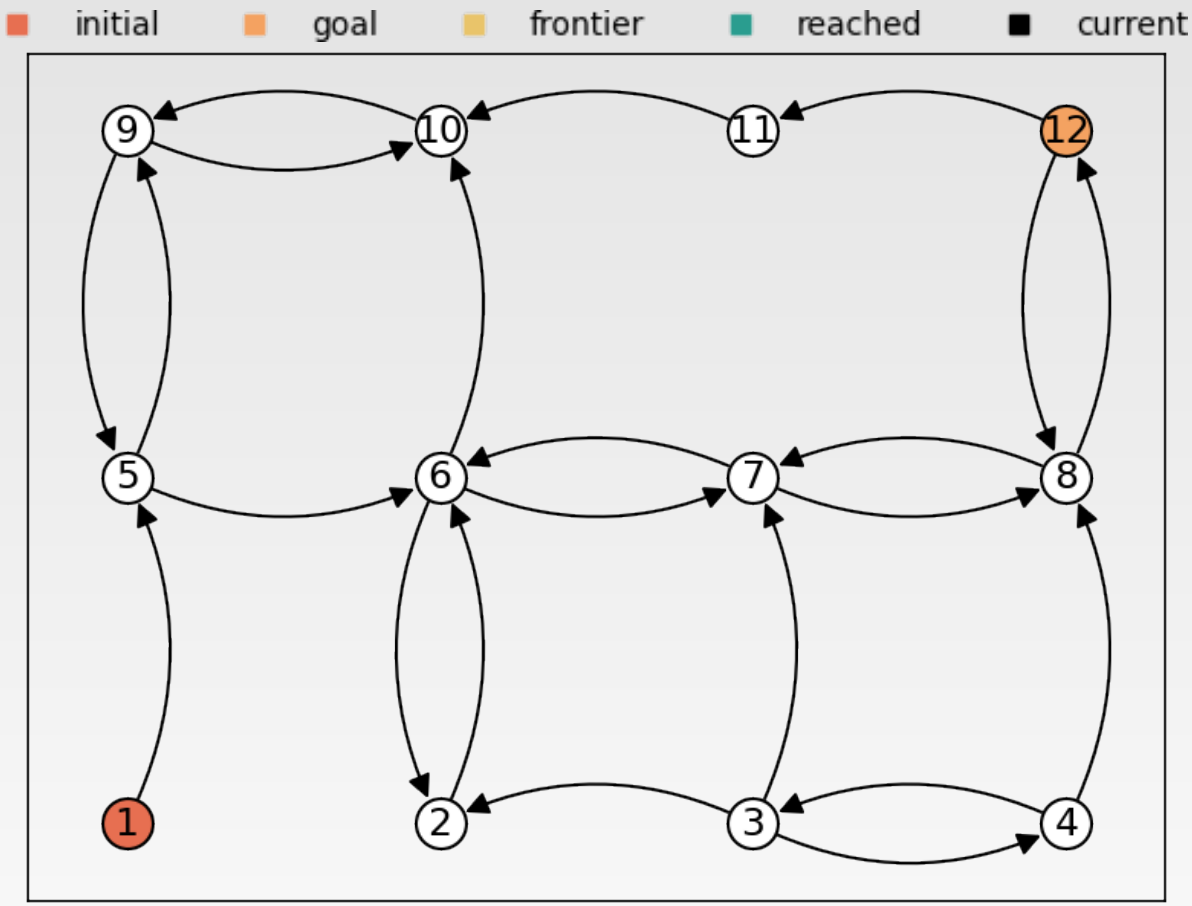
6.1) BFS? **6.2)** DFS? **6.3)** Uniform-cost?

Quick questions 7



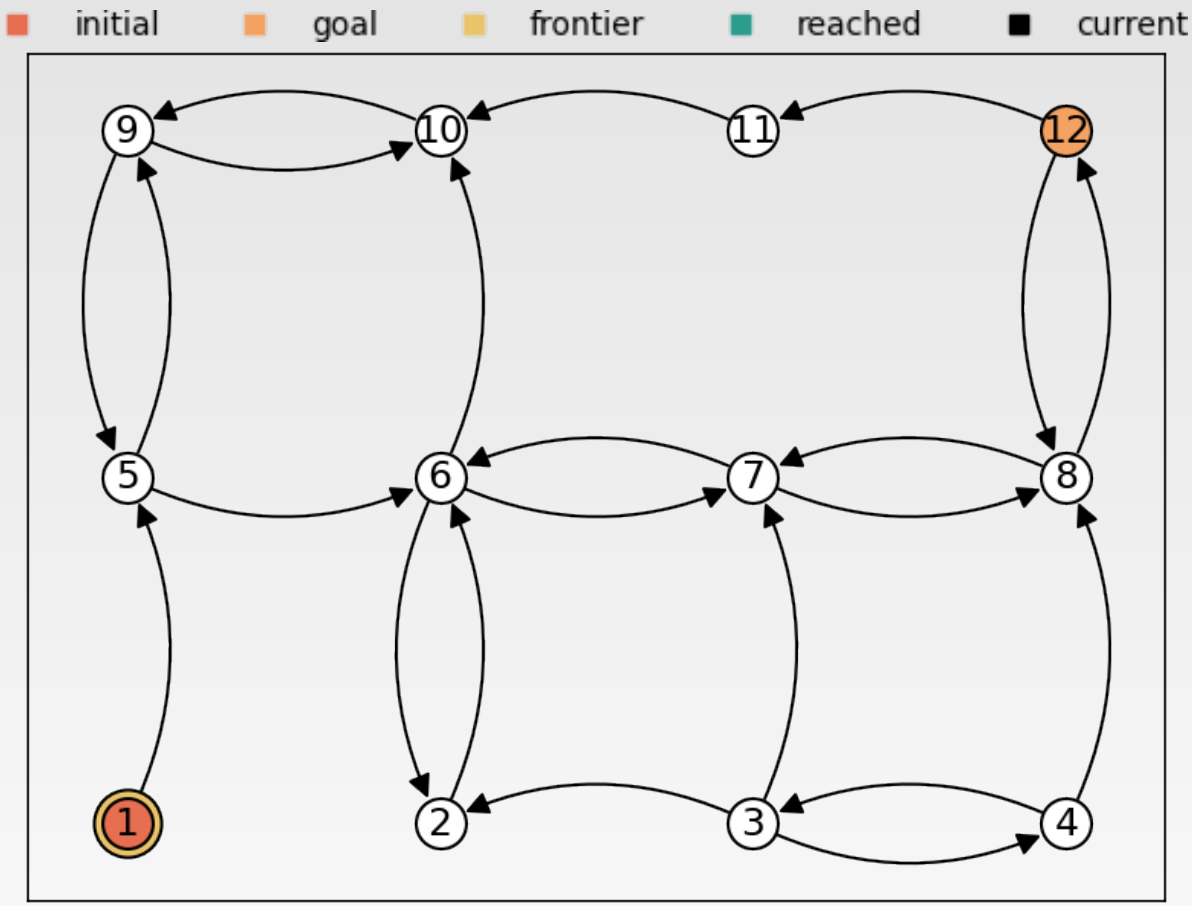
7.1) BFS? **7.2)** DFS? **7.3)** Uniform-cost?

Ex. 1 - Breadth-first



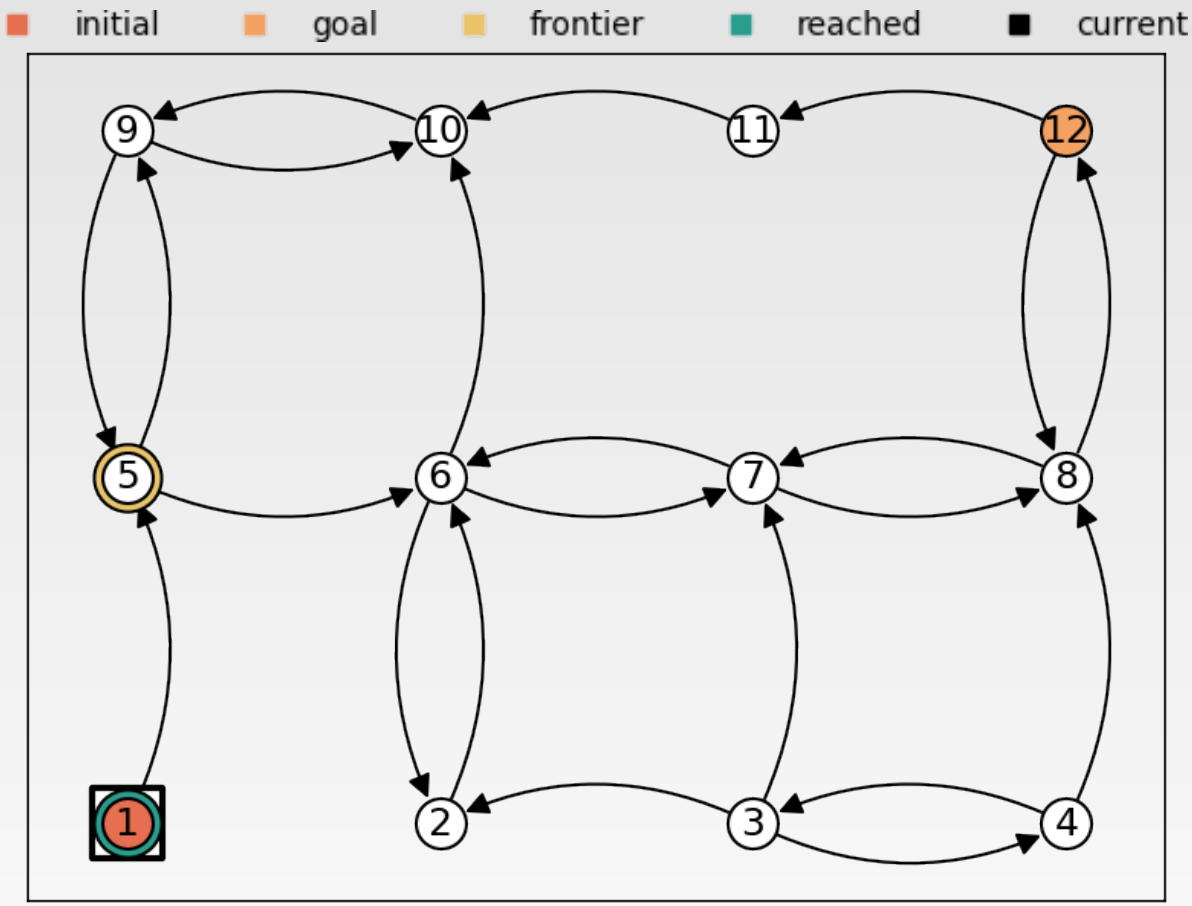
Iteration: -, Current:
Frontier: []
Reached: {}

Ex. 1 - Breadth-first



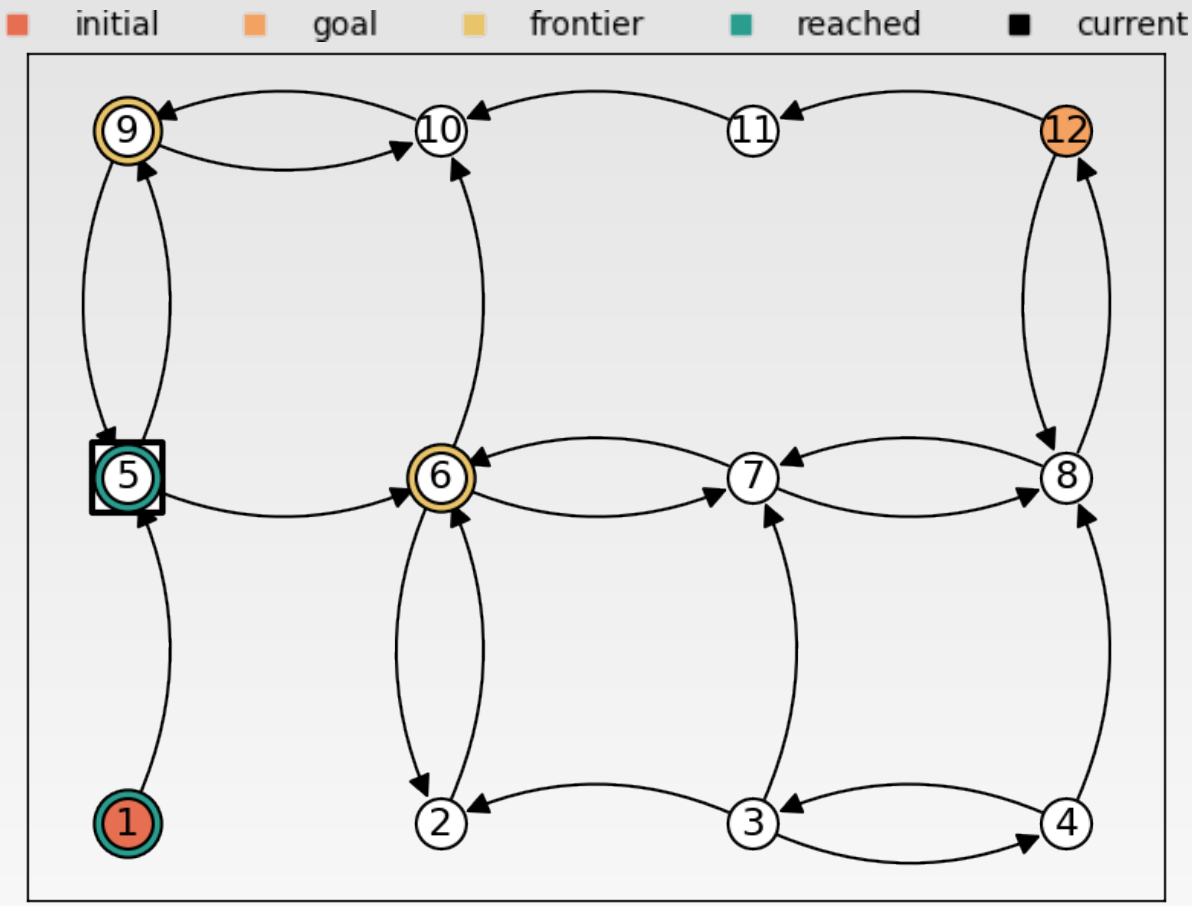
Iteration: 1, Current:
Frontier: [1]
Reached: {1}

Ex. 1 - Breadth-first



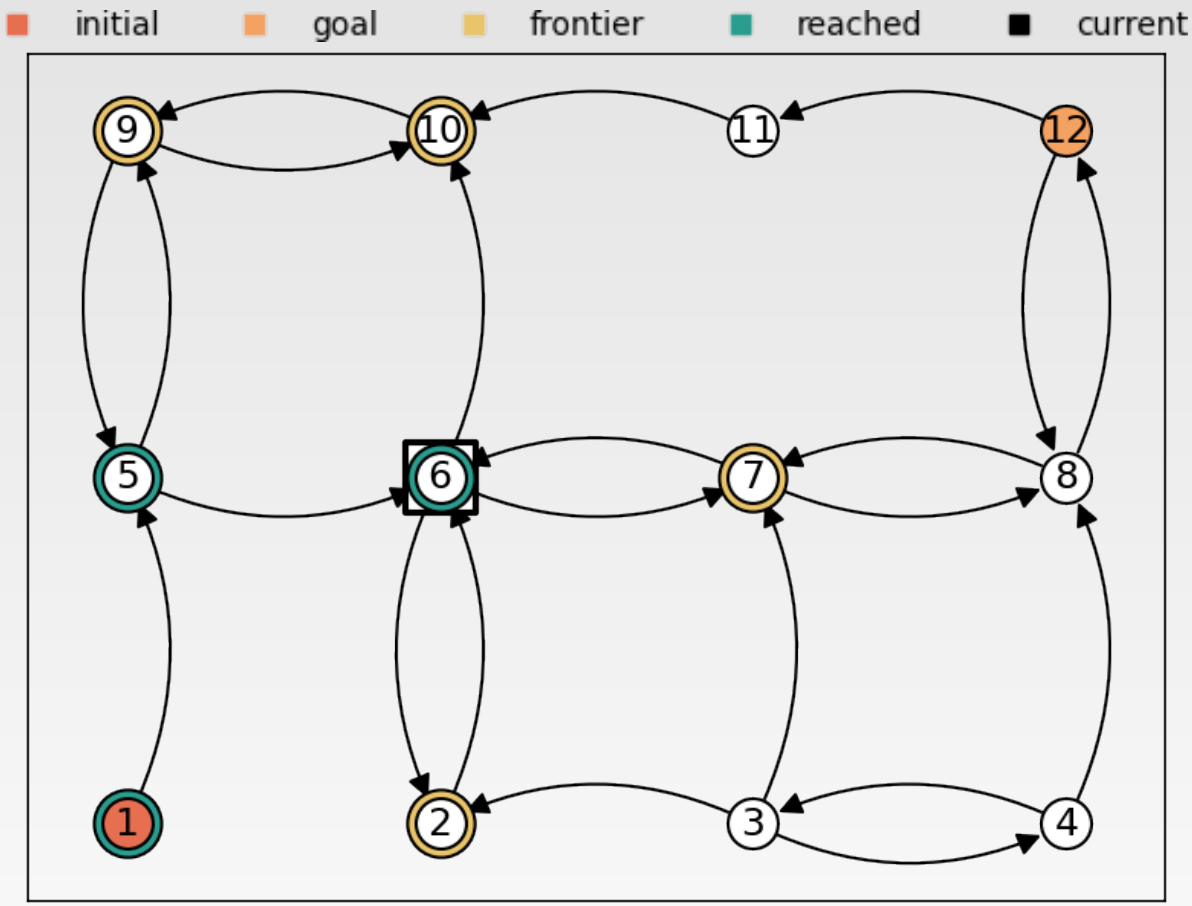
Iteration: 2, Current: 1
Frontier: [5]
Reached: {1, 5}

Ex. 1 - Breadth-first



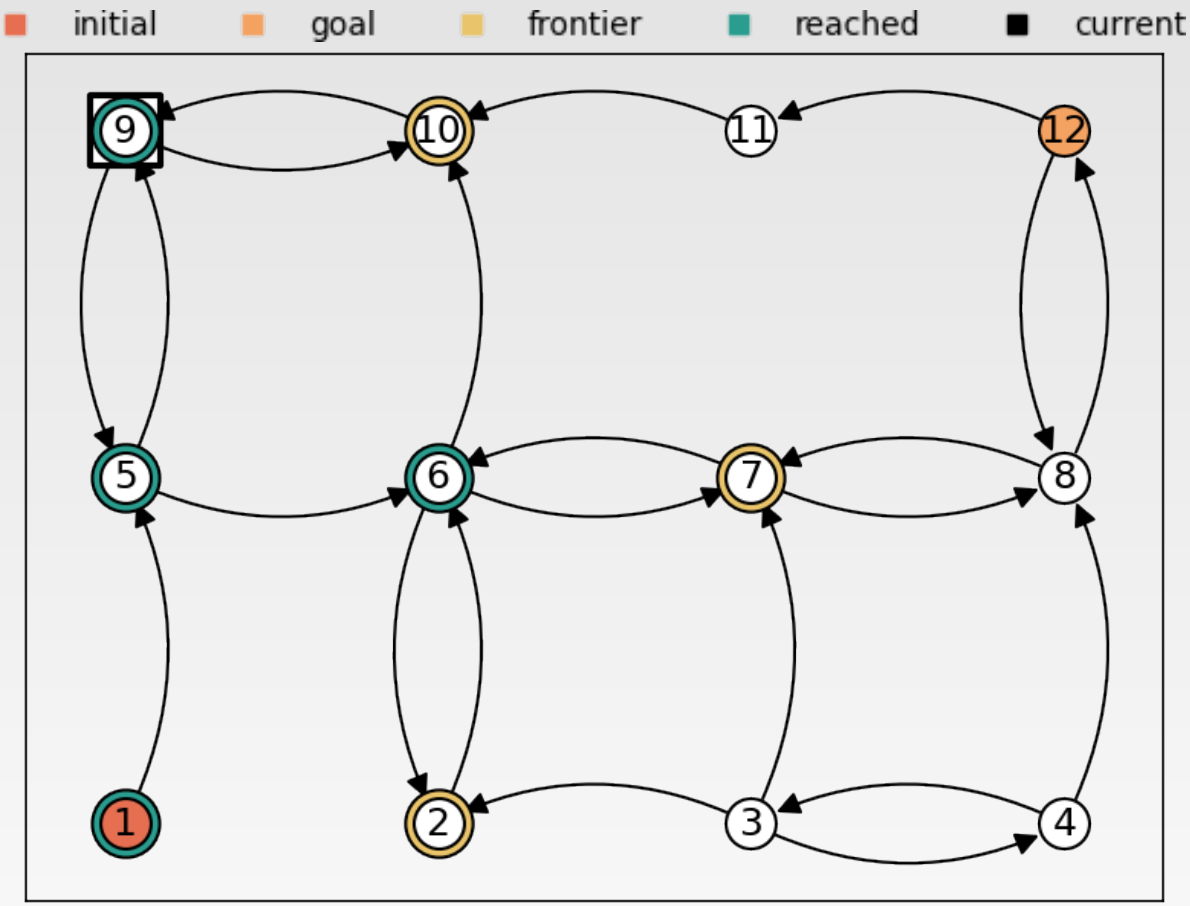
Iteration: 3, Current: 5
Frontier: [6, 9]
Reached: {1, 5, 6, 9}

Ex. 1 - Breadth-first



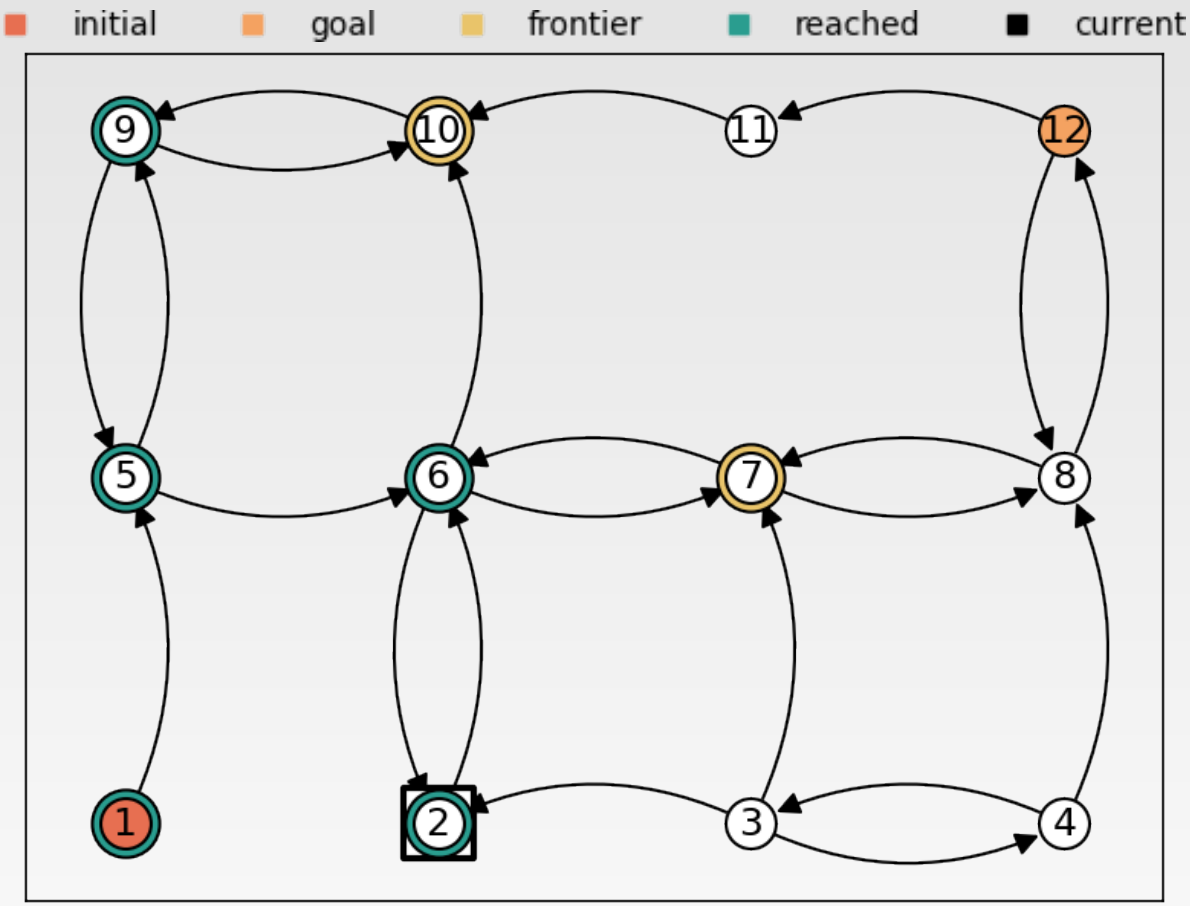
Iteration: 4, Current: 6
Frontier: [9, 2, 7, 10]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Breadth-first



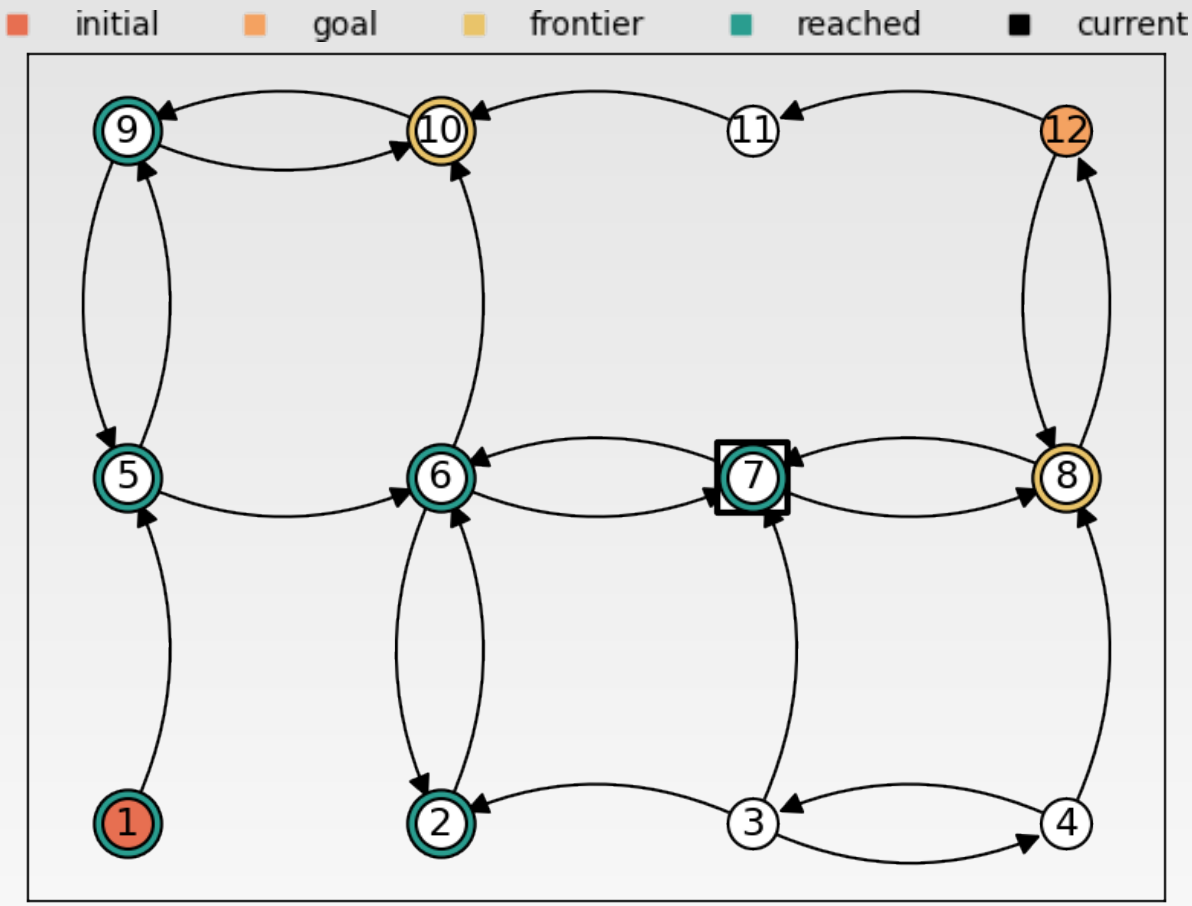
Iteration: 5, Current: 9
Frontier: [2, 7, 10]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Breadth-first



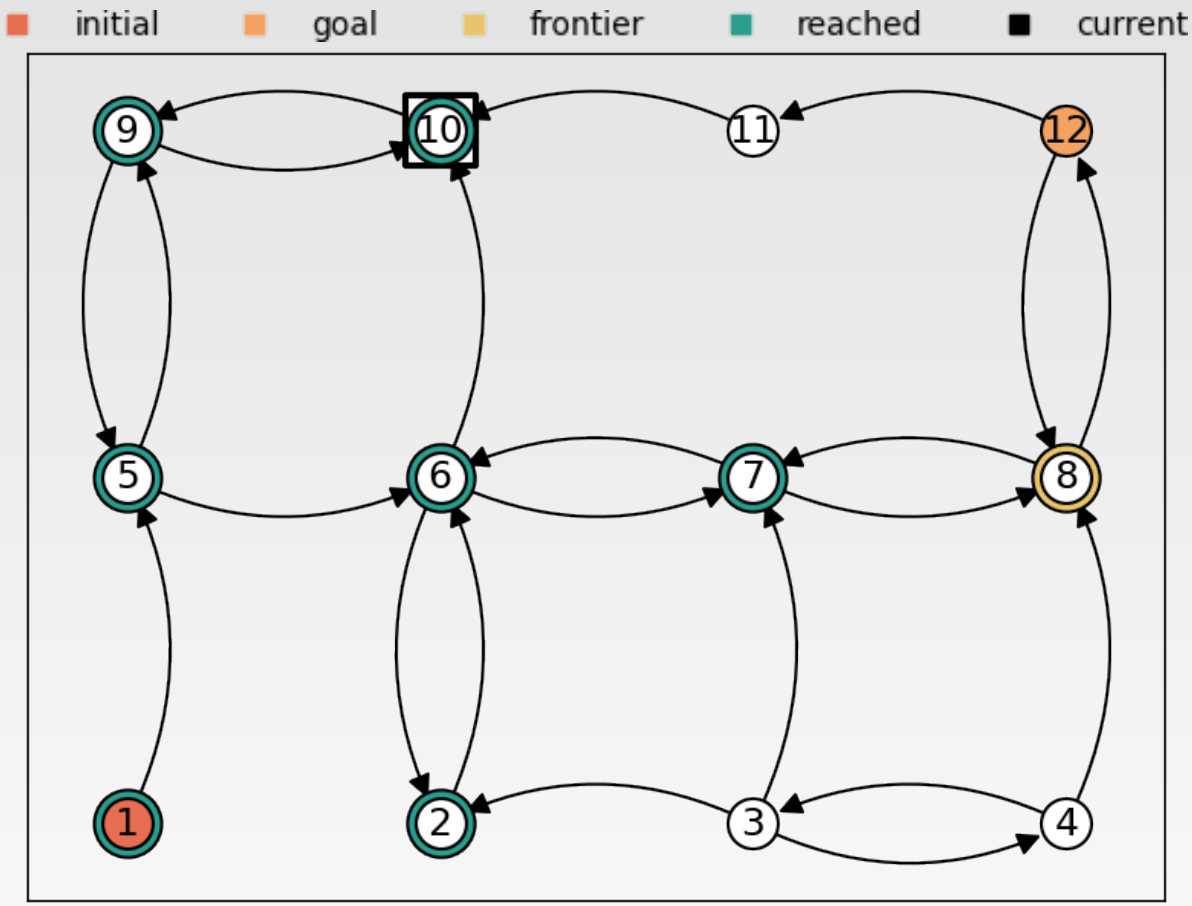
Iteration: 6, Current: 2
Frontier: [7, 10]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Breadth-first



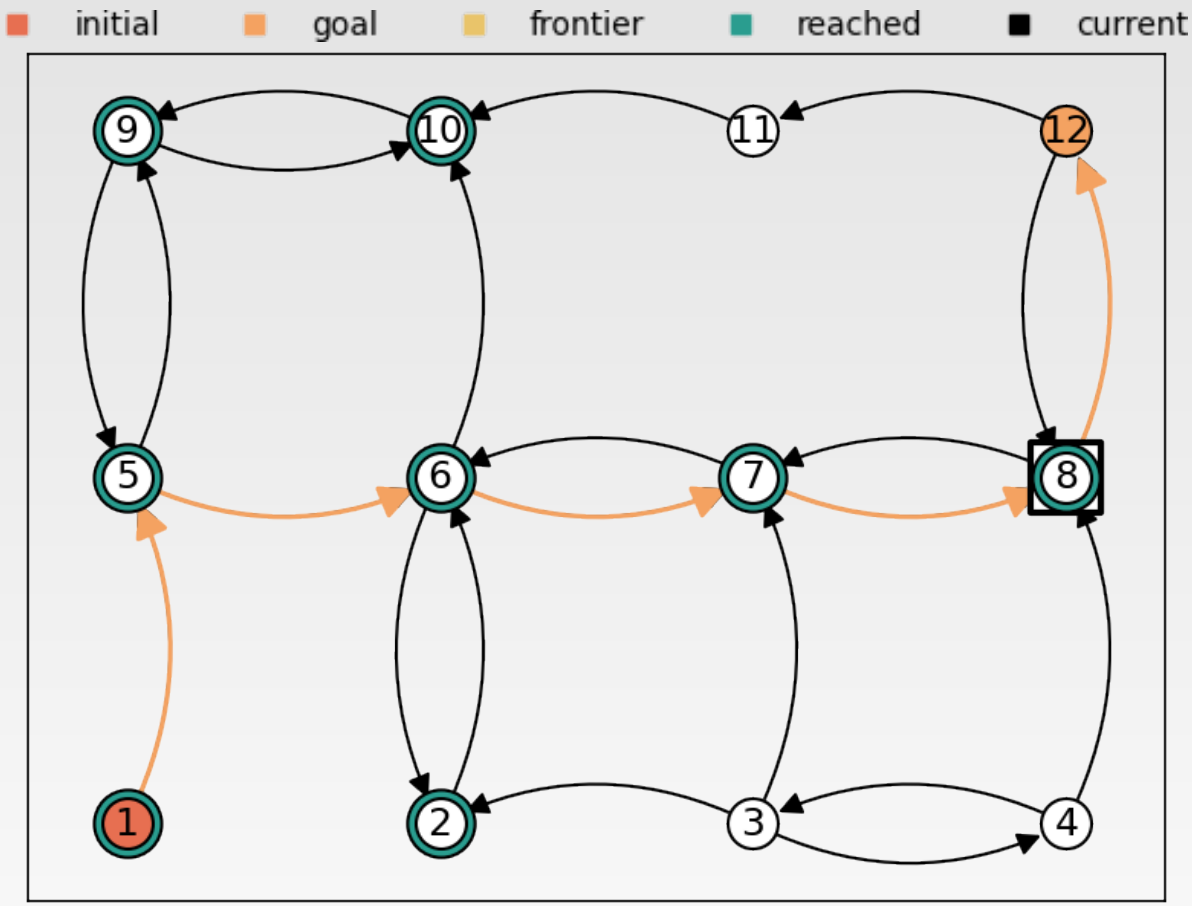
Iteration: 7, Current: 7
Frontier: [10, 8]
Reached: {1, 2, 5, 6, 7, 8, 9, 10}

Ex. 1 - Breadth-first



Iteration: 8, Current: 10
Frontier: [8]
Reached: {1, 2, 5, 6, 7, 8, 9, 10}

Ex. 1 - Breadth-first

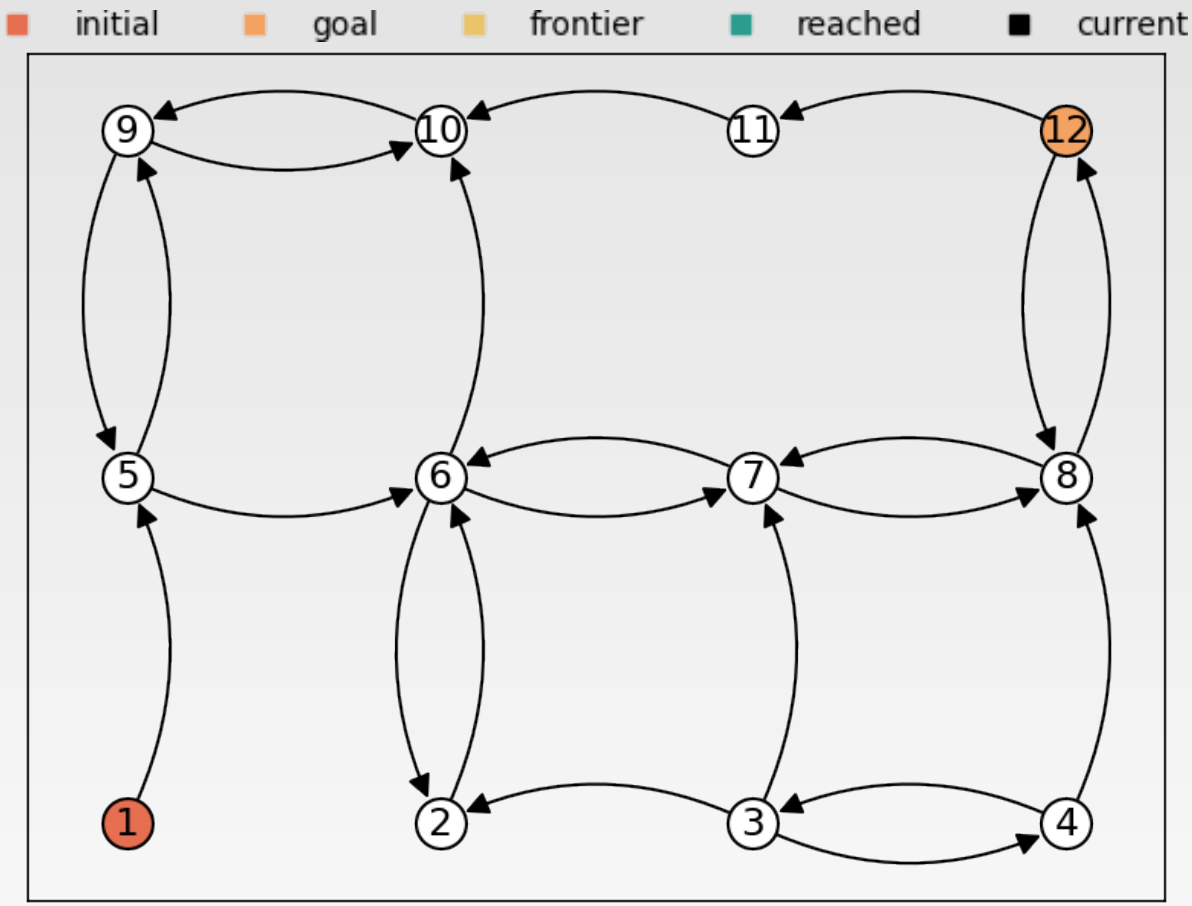


Iteration: 9, Current: 8
Frontier: []
Reached: {1, 2, 5, 6, 7, 8, 9, 10}

Ex. 1 - Breadth-first

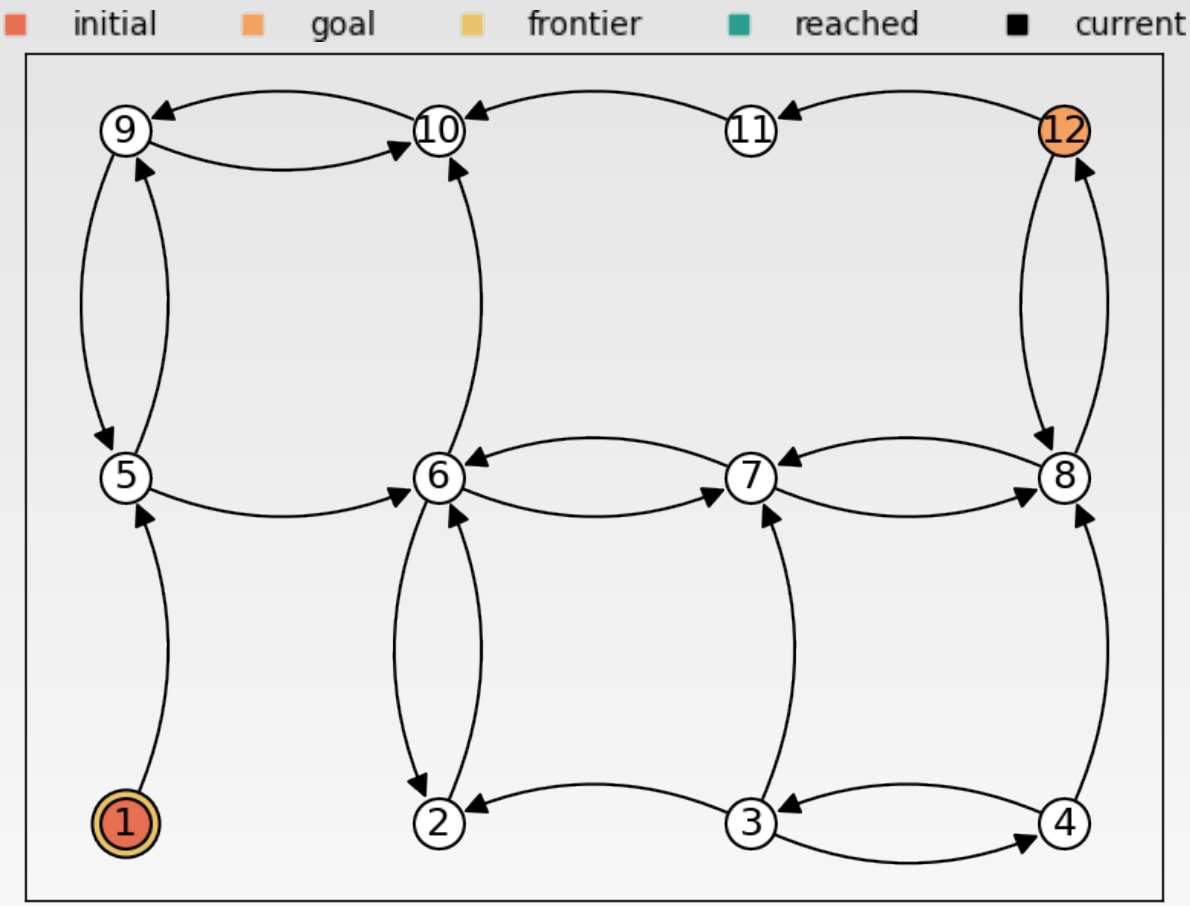
Done! *(we explored all the reachable nodes though..)*

Ex. 1 - Depth-first (tree-like)



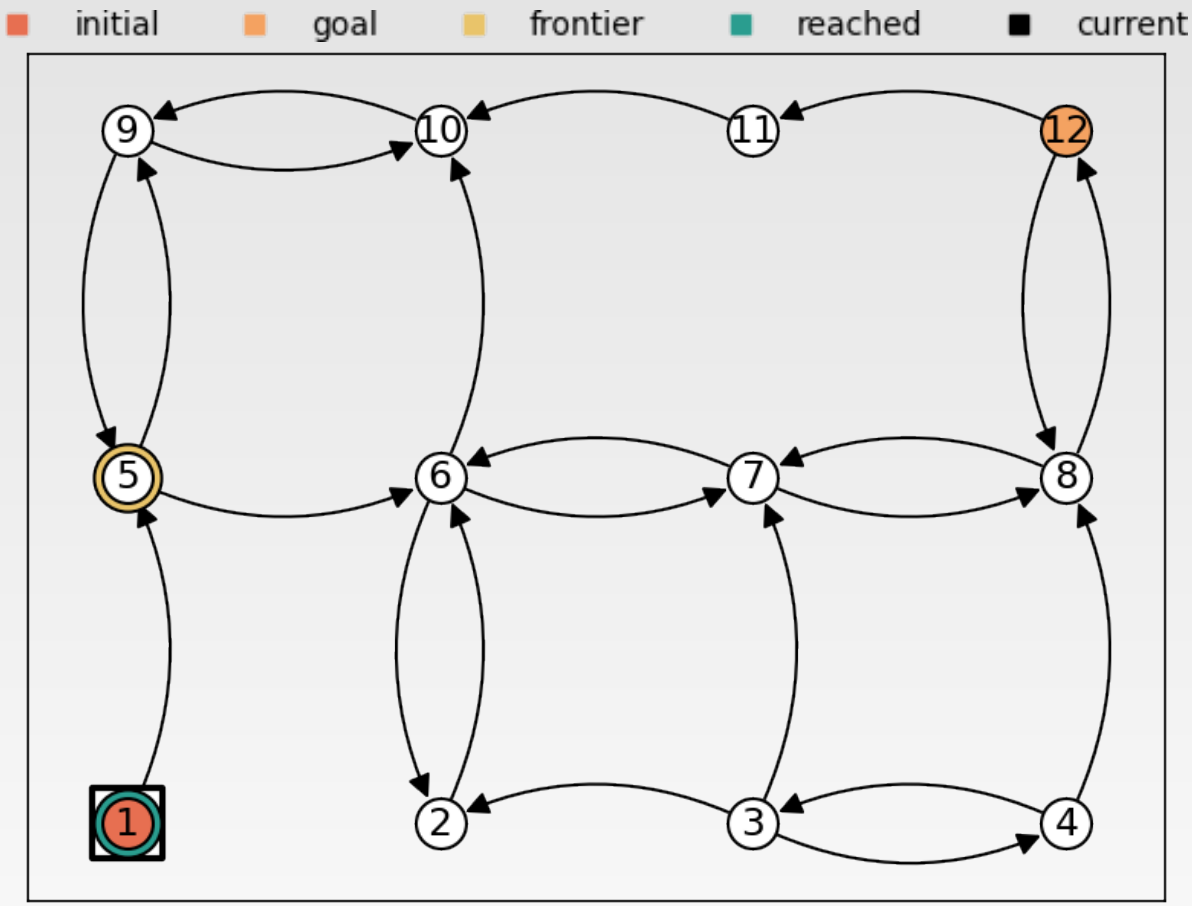
Iteration: -, Current:
Frontier: []
Reached: {}

Ex. 1 - Depth-first (tree-like)



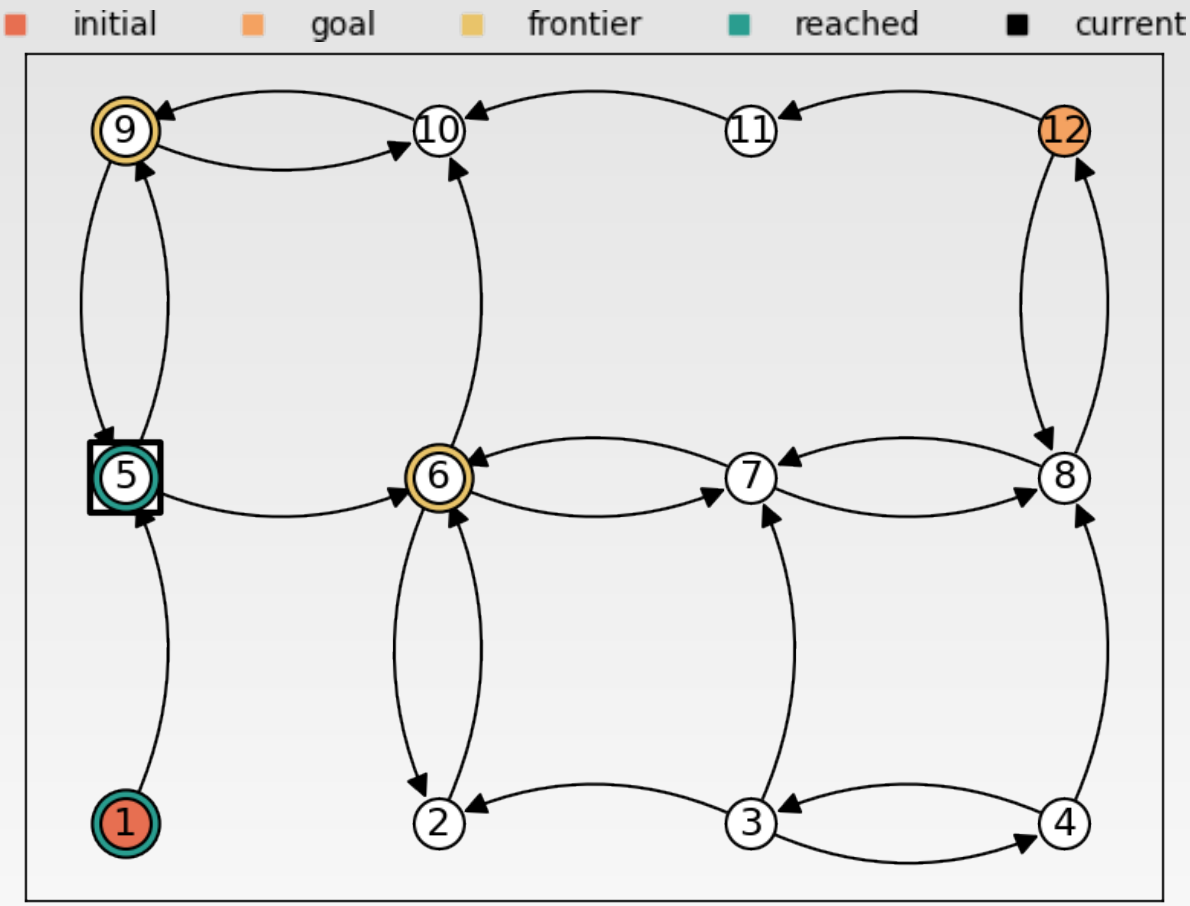
Iteration: 1, Current:
Frontier: [1]
Reached: {1}

Ex. 1 - Depth-first (tree-like)



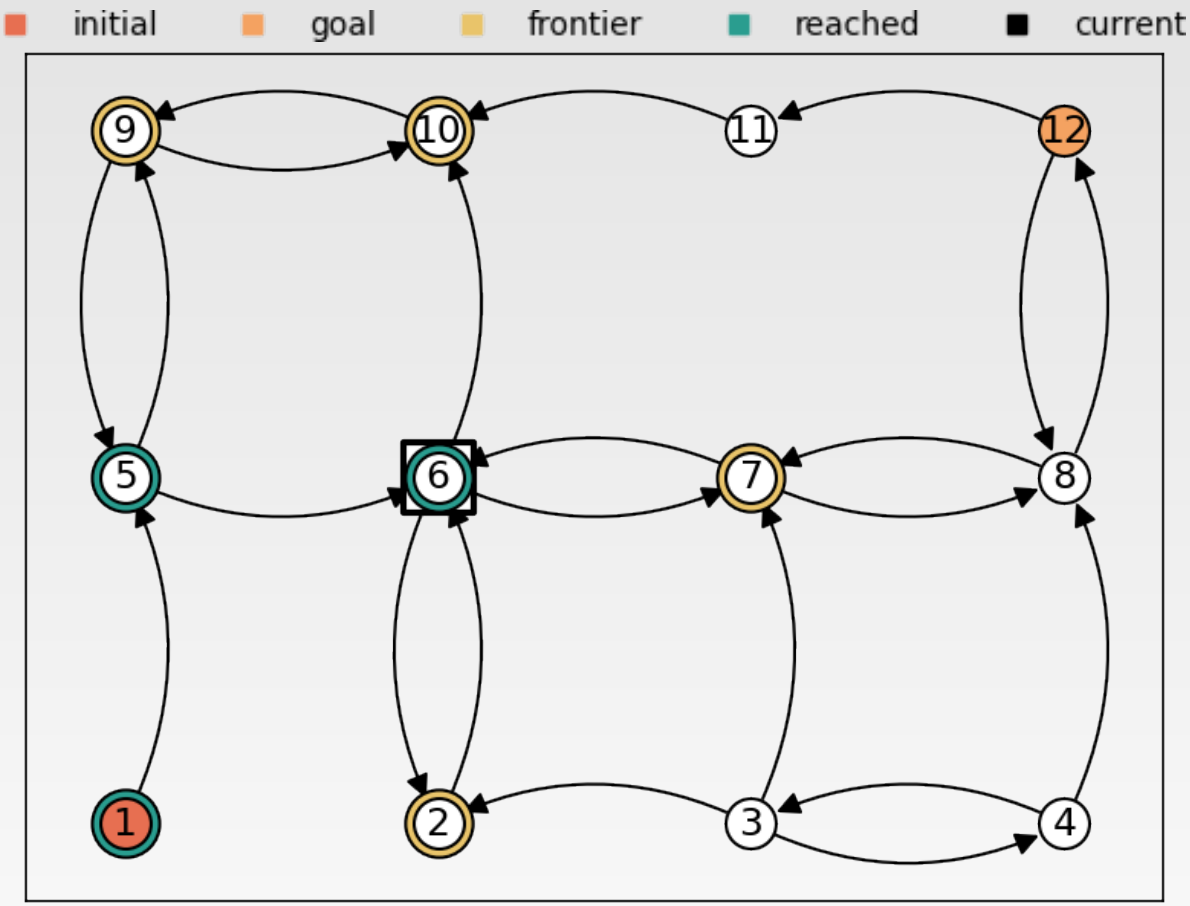
Iteration: 2, Current: 1
Frontier: [5]
Reached: {1, 5}

Ex. 1 - Depth-first (tree-like)



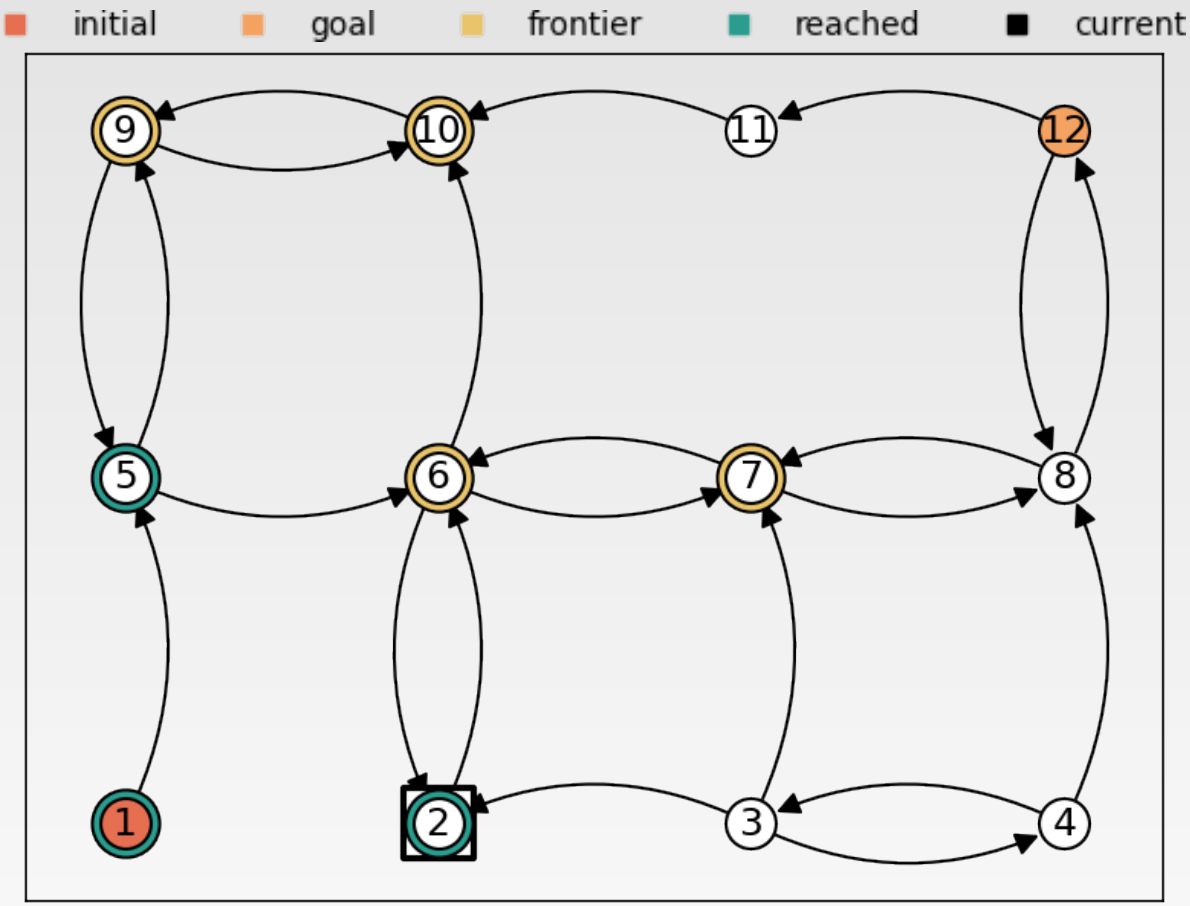
Iteration: 3, Current: 5
Frontier: [9, 6]
Reached: {1, 5, 9, 6}

Ex. 1 - Depth-first (tree-like)



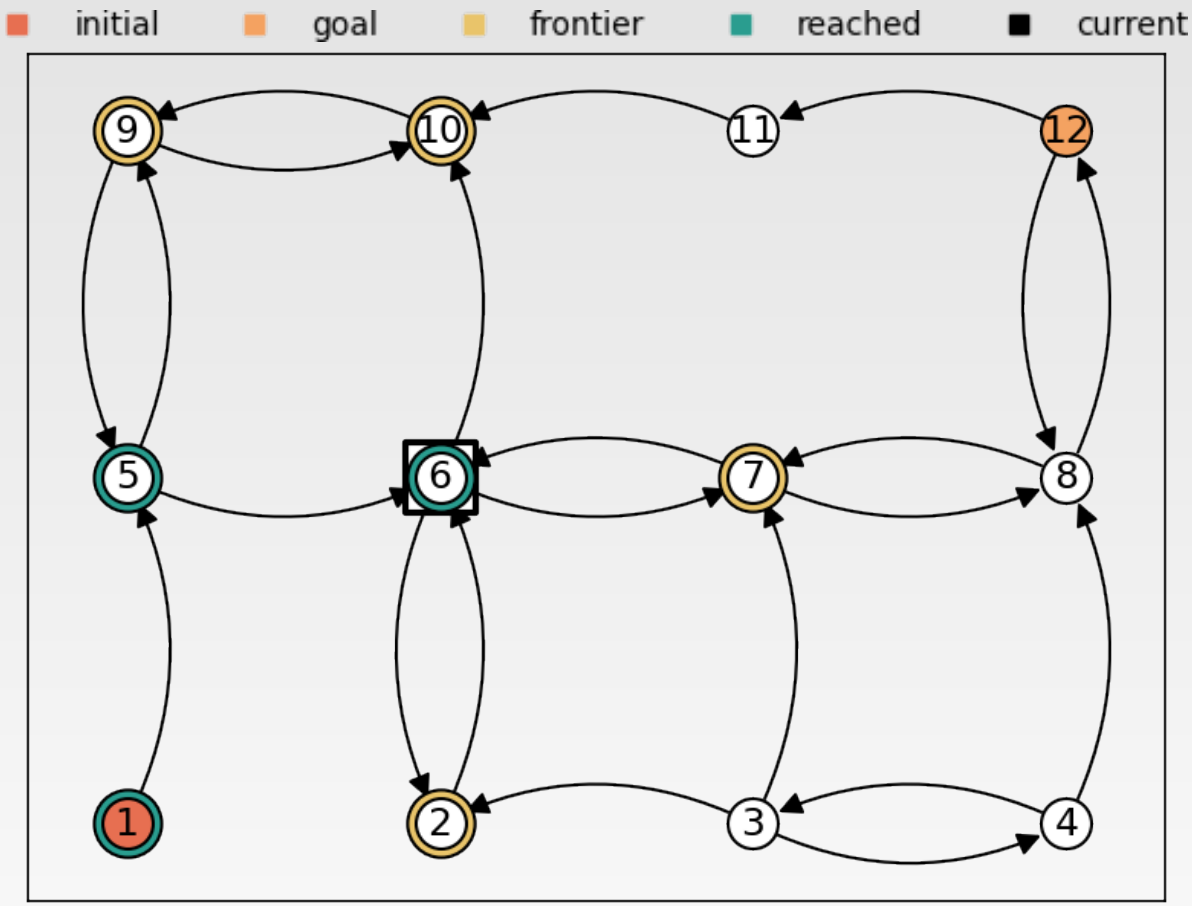
Iteration: 4, Current: 6
Frontier: [9, 10, 7, 2]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Depth-first (tree-like)



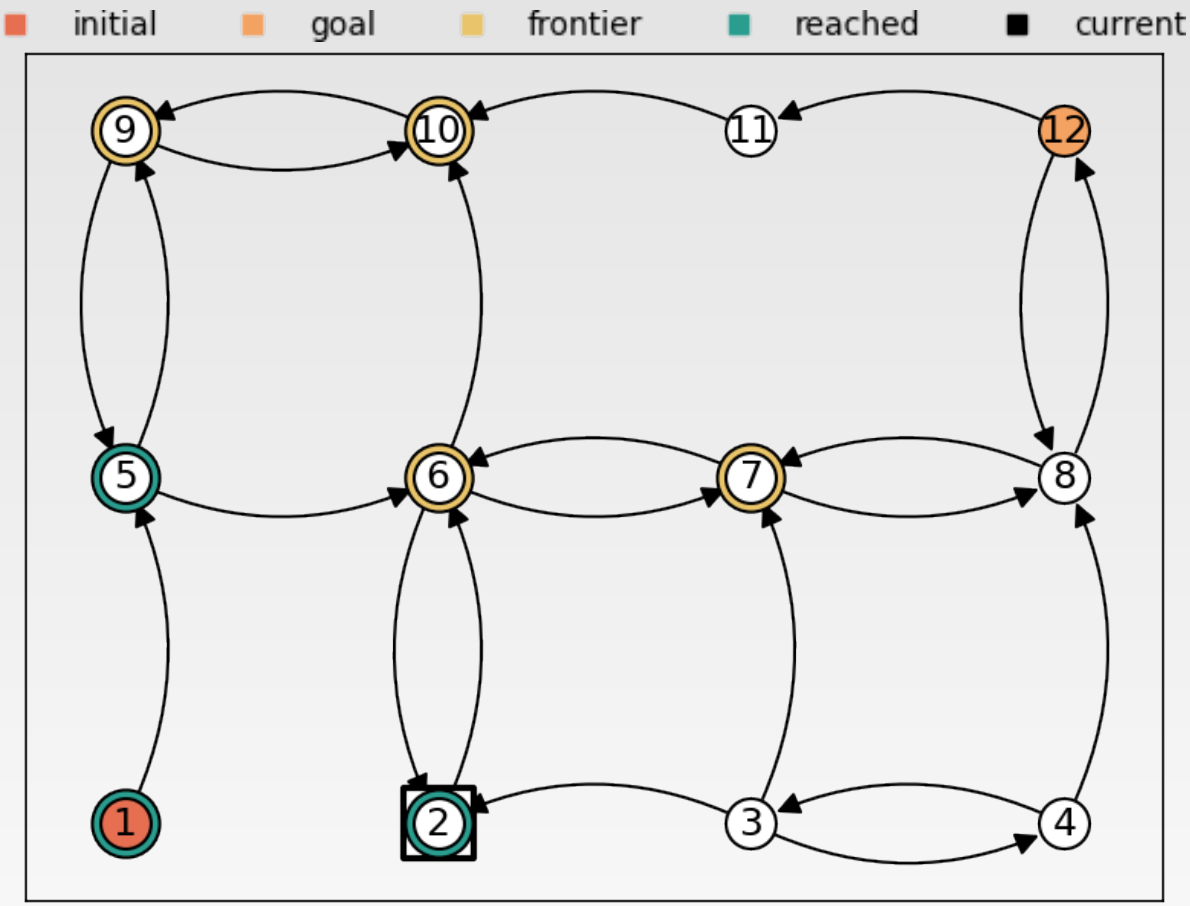
Iteration: 5, Current: 2
Frontier: [9, 10, 7, 6]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Depth-first (tree-like)



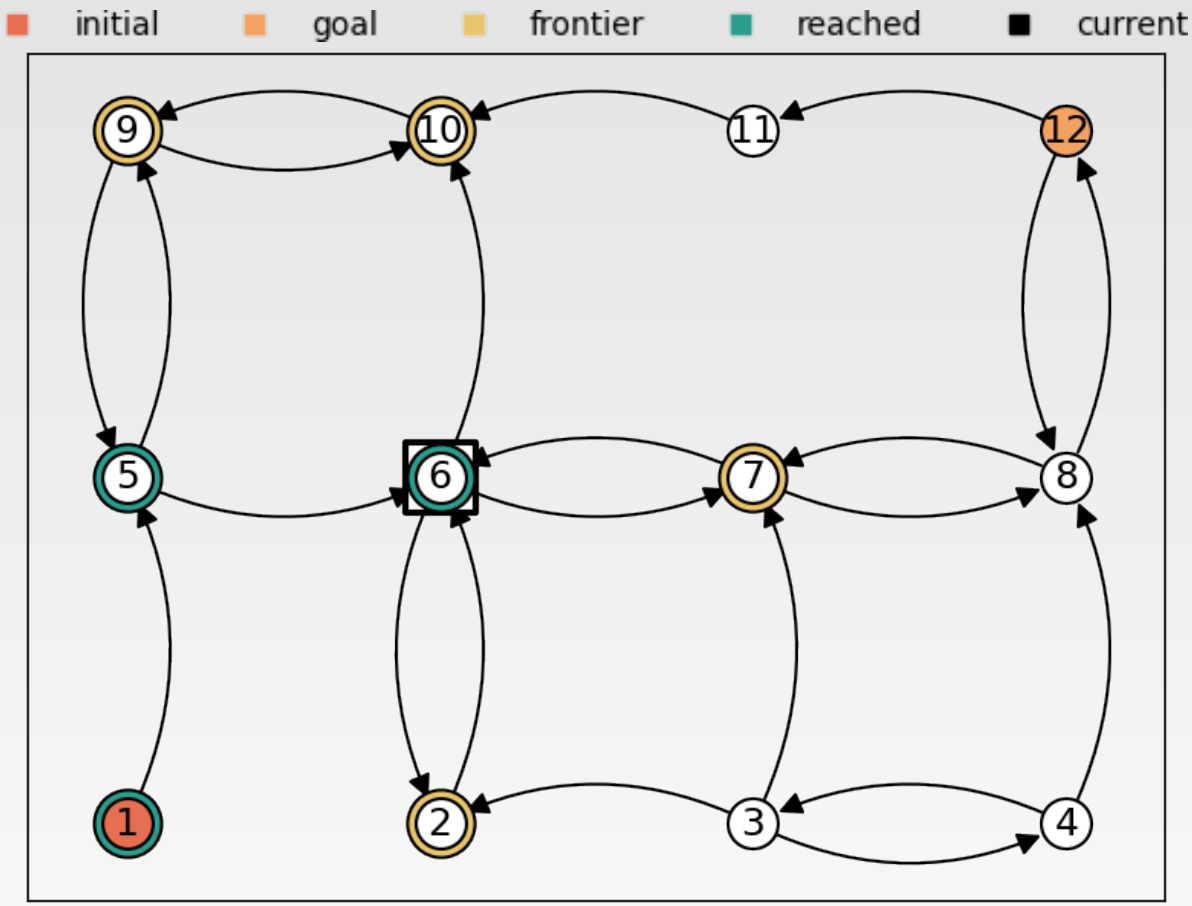
Iteration: 6, Current: 6
Frontier: [9, 10, 7, 10, 7, 2]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Depth-first (tree-like)



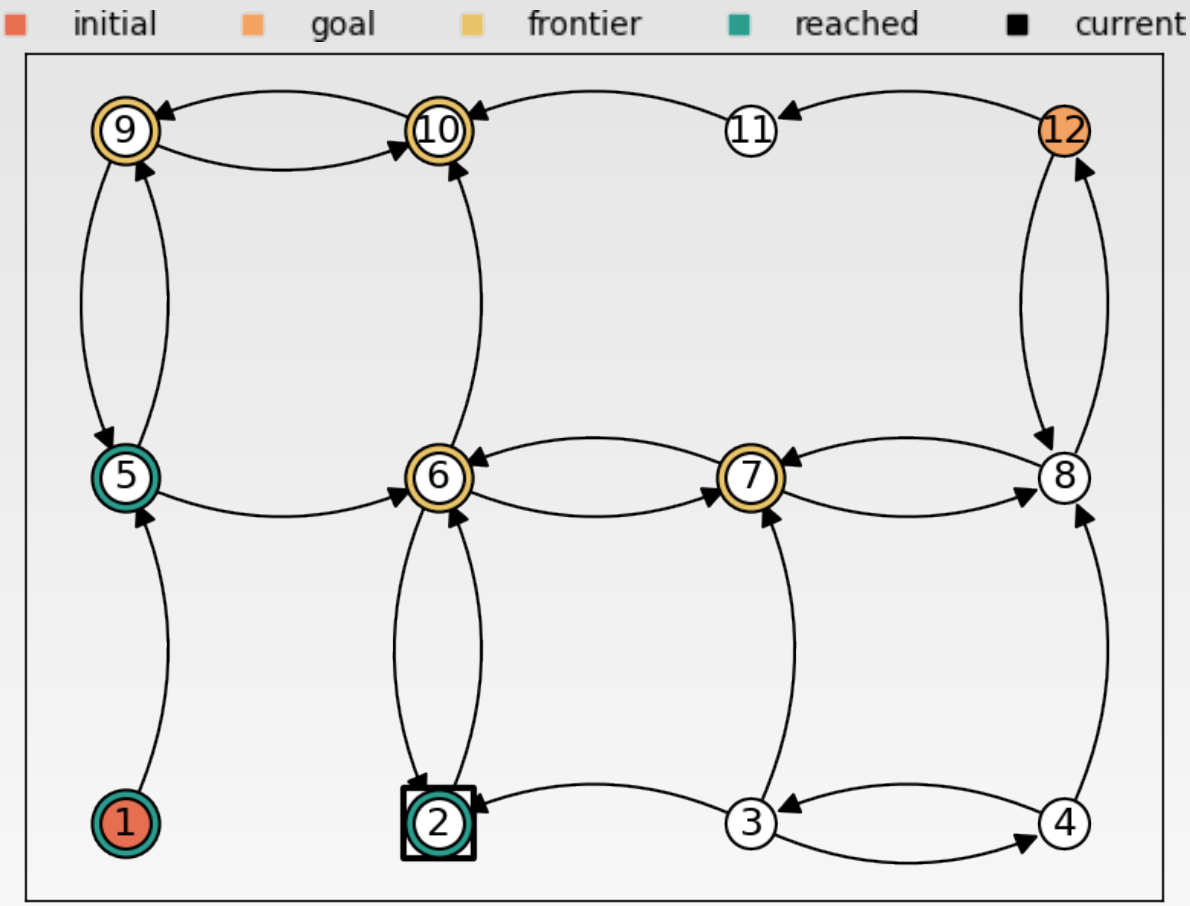
Iteration: 7, Current: 2
Frontier: [9, 10, 7, 10, 7, 6]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Depth-first (tree-like)



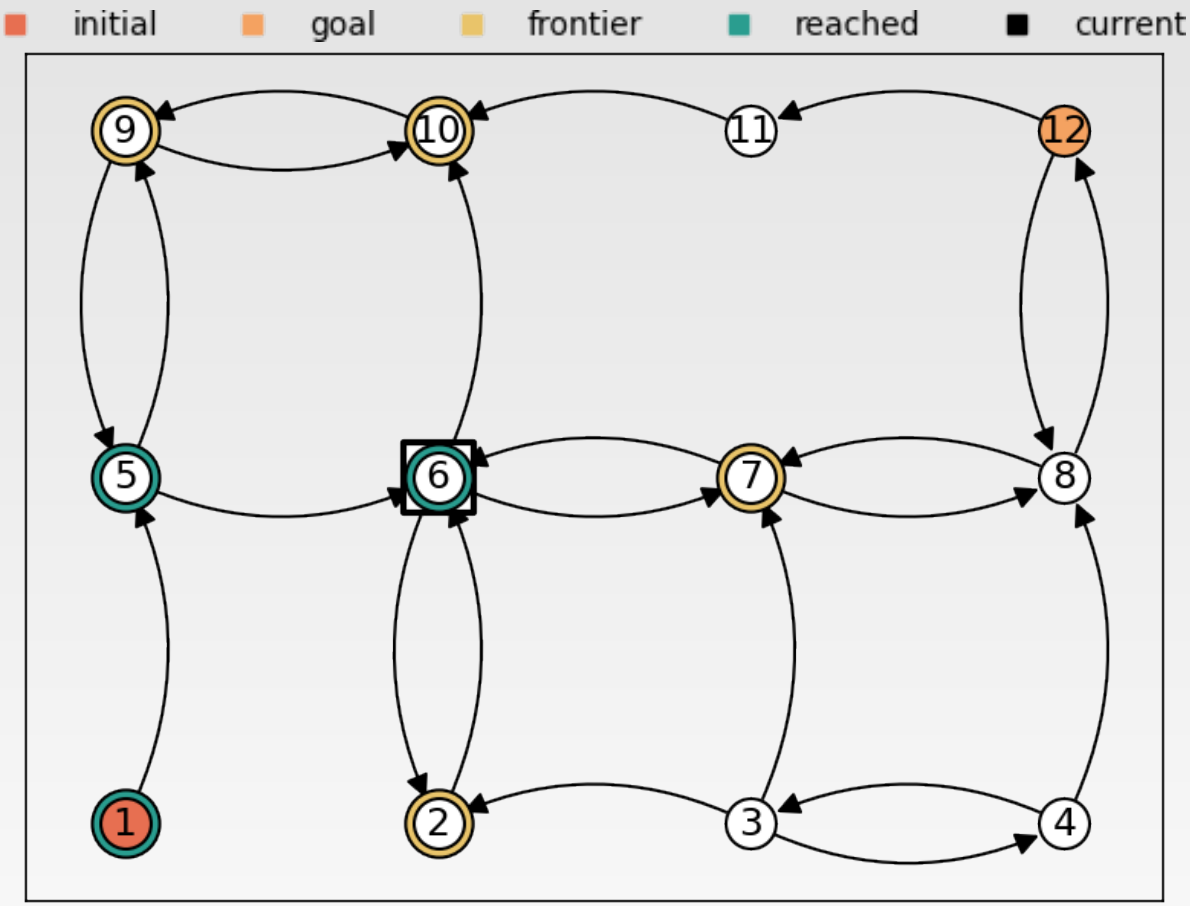
Iteration: 8, Current: 6
Frontier: [9, 10, 7, 10, 7, 10, 7, 2]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Depth-first (tree-like)



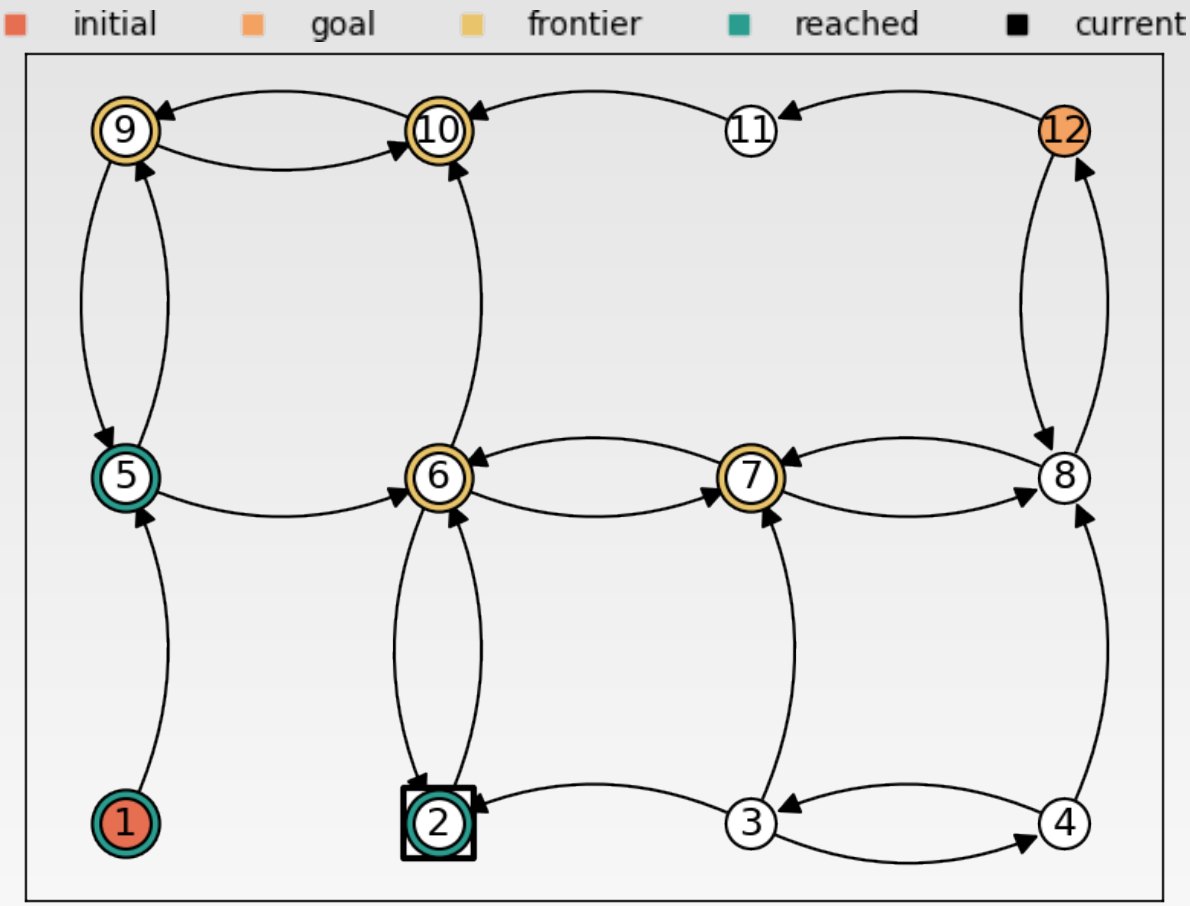
Iteration: 9, Current: 2
Frontier: [9, 10, 7, 10, 7, 10, 7, 6]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Depth-first (tree-like)



Iteration: 10, Current: 6
Frontier: [9, 10, 7, 10, 7, 10, 7, 10, 7, 2]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Depth-first (tree-like)

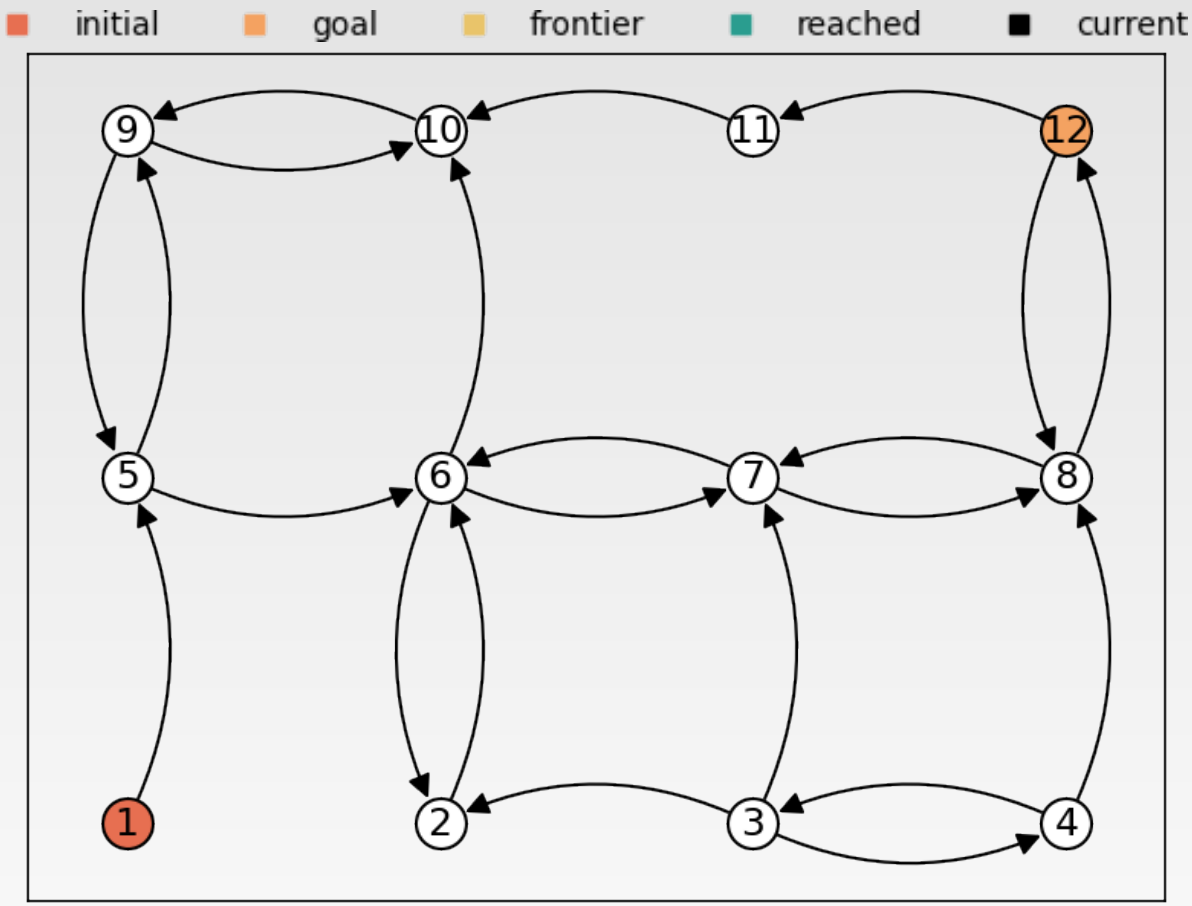


Iteration: 11, Current: 2
Frontier: [9, 10, 7, 10, 7, 10, 7, 10, 7, 6]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Depth-first (tree-like)

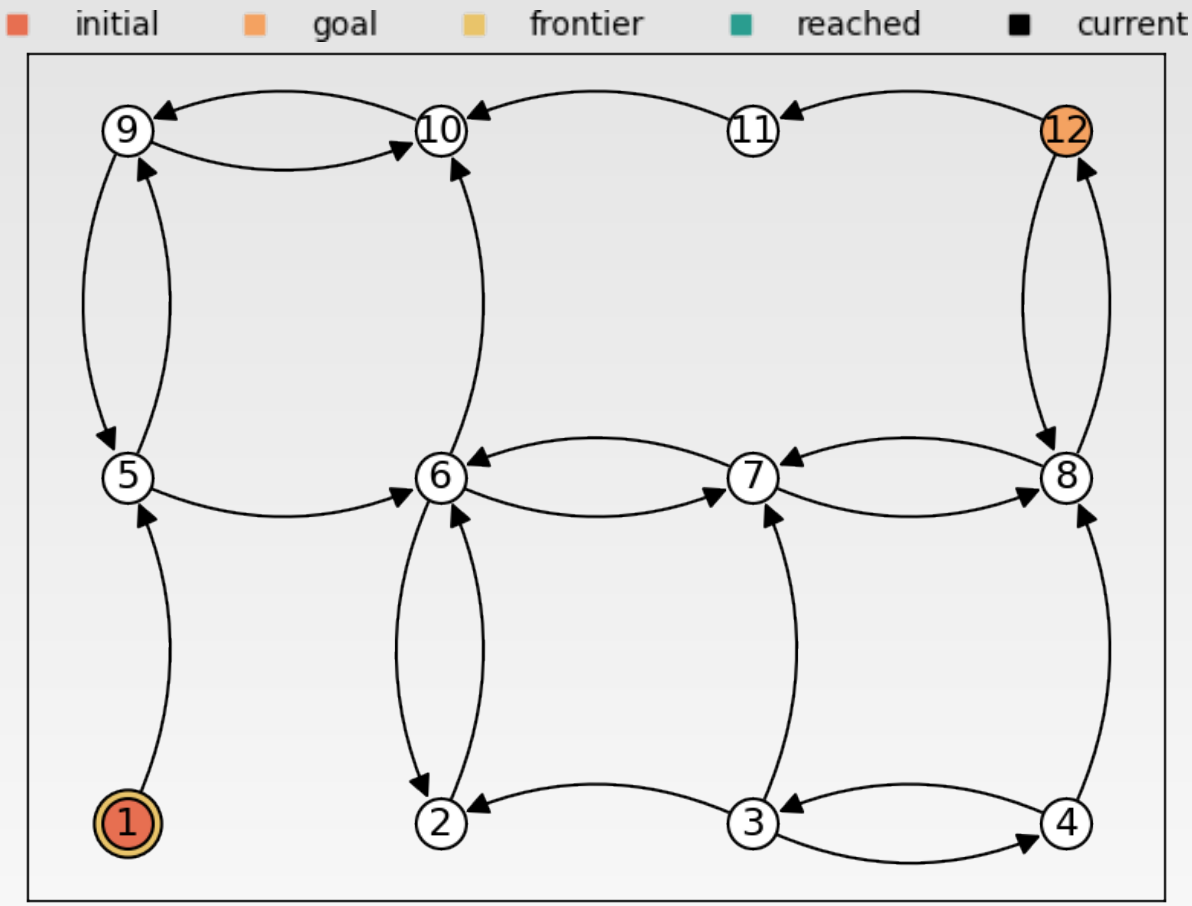
Ouch! *(this could have been easily avoided with cycle detection)*

Ex. 1 - Depth-first (graph-like)



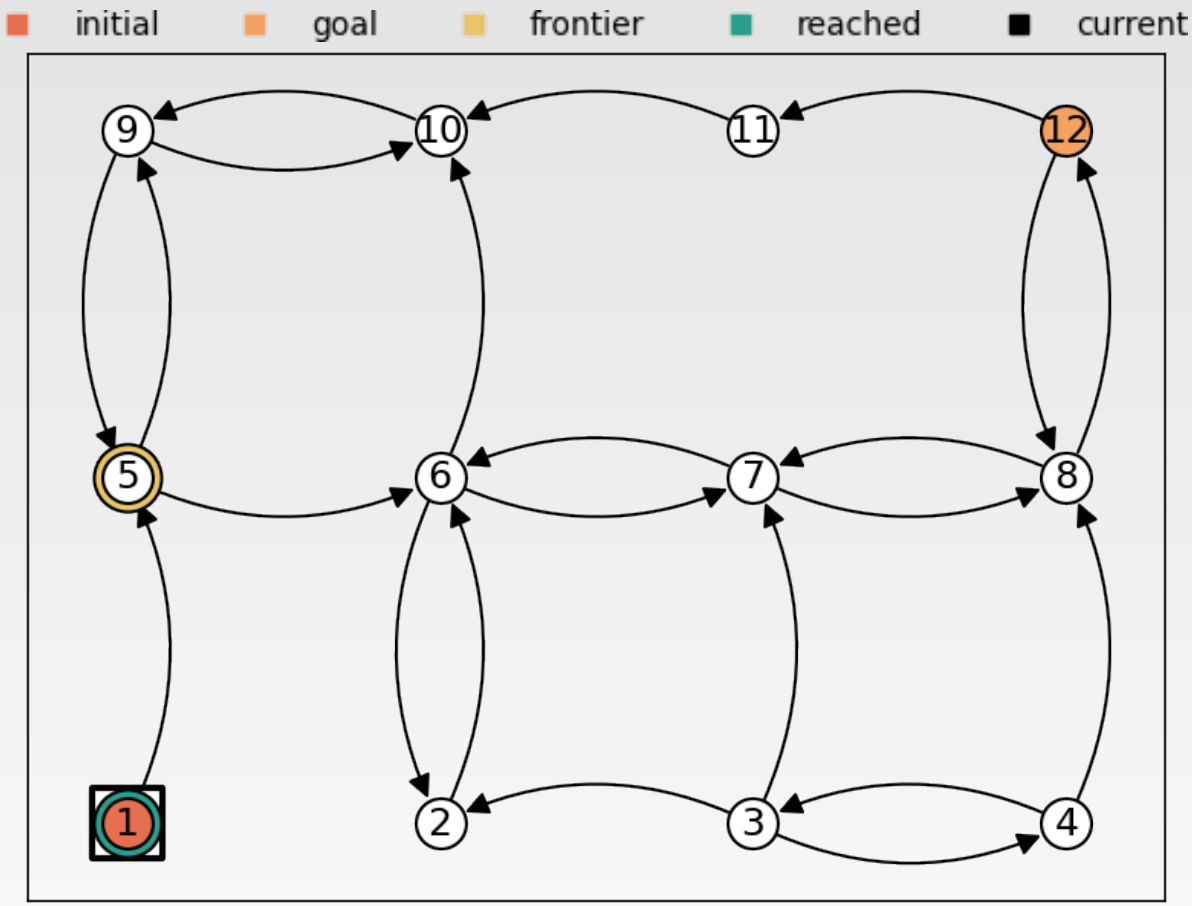
Iteration: -, Current:
Frontier: []
Reached: {}

Ex. 1 - Depth-first (graph-like)



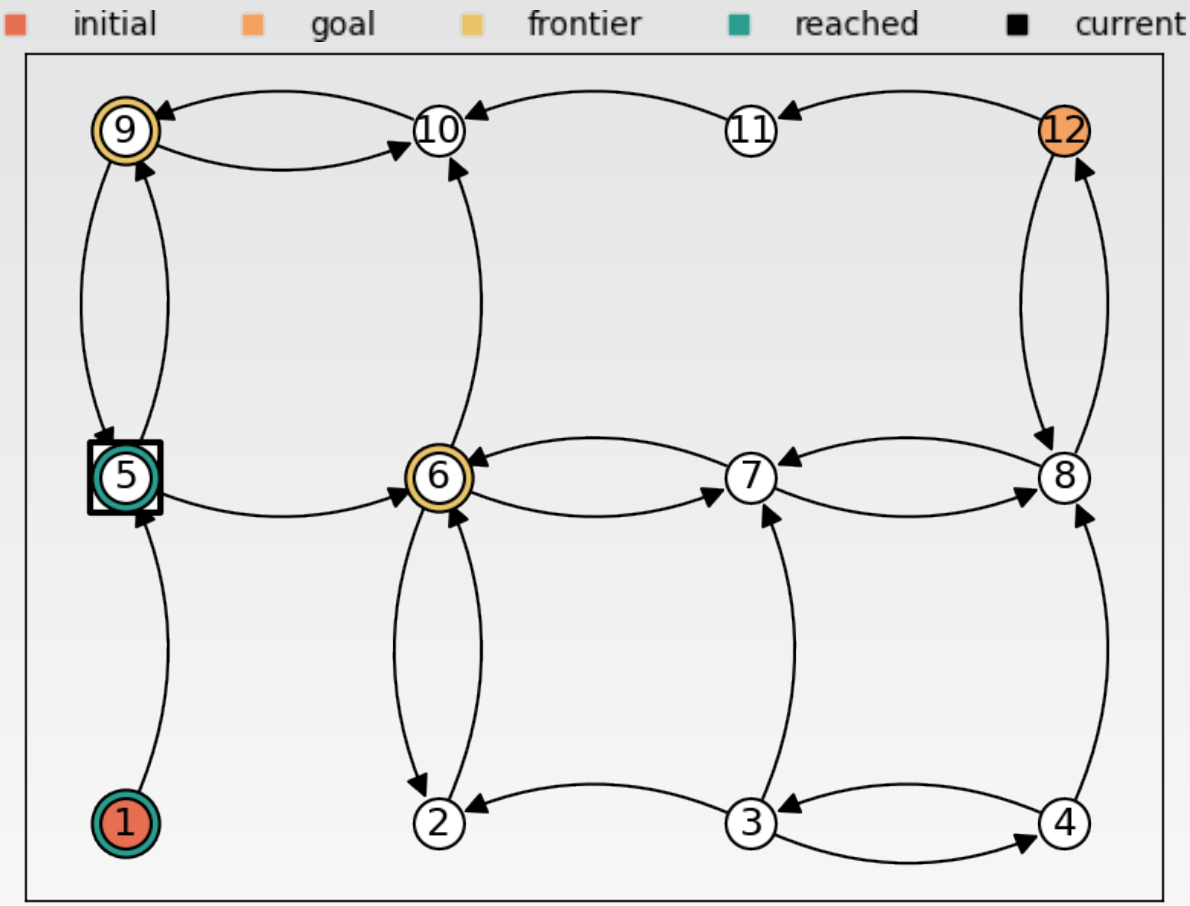
Iteration: 1, Current:
Frontier: [1]
Reached: {1}

Ex. 1 - Depth-first (graph-like)



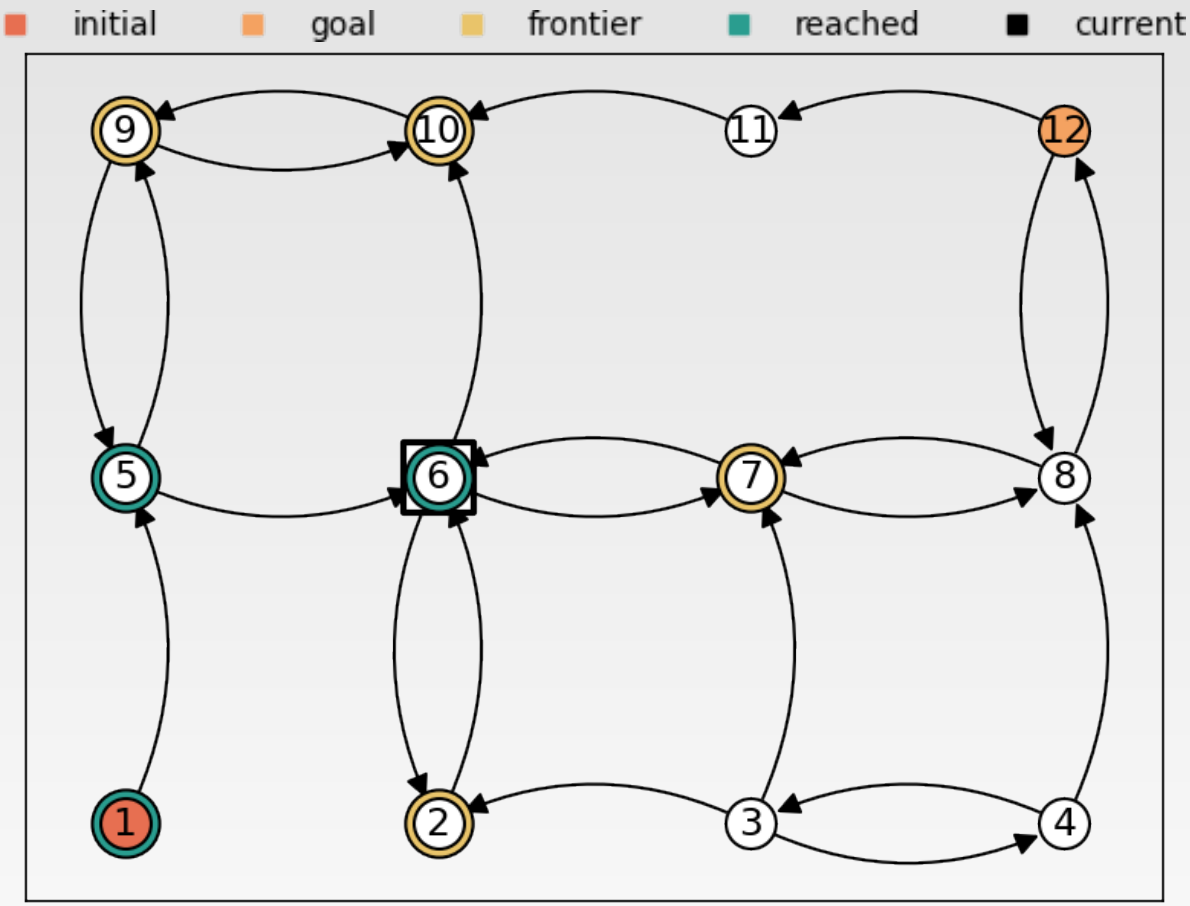
Iteration: 2, Current: 1
Frontier: [5]
Reached: {1, 5}

Ex. 1 - Depth-first (graph-like)



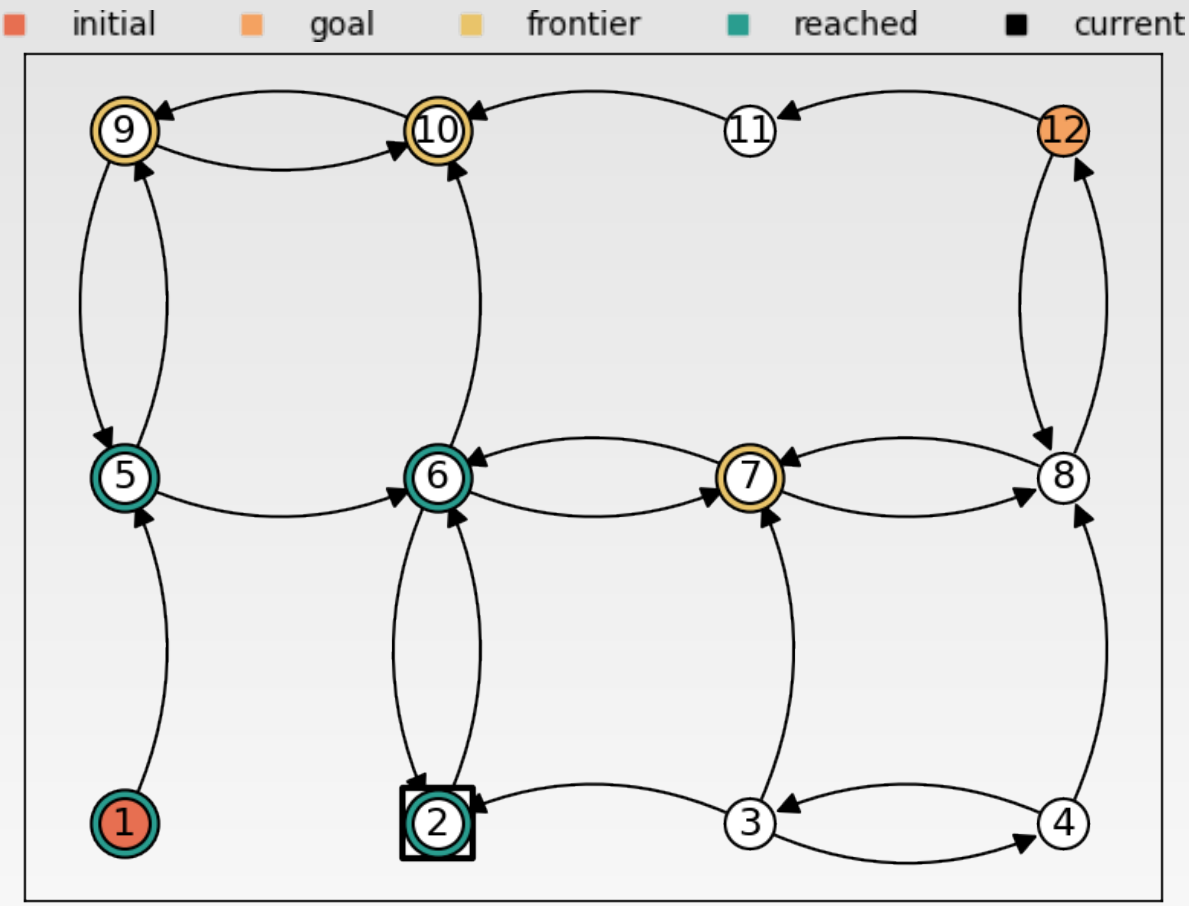
Iteration: 3, Current: 5
Frontier: [9, 6]
Reached: {1, 5, 9, 6}

Ex. 1 - Depth-first (graph-like)



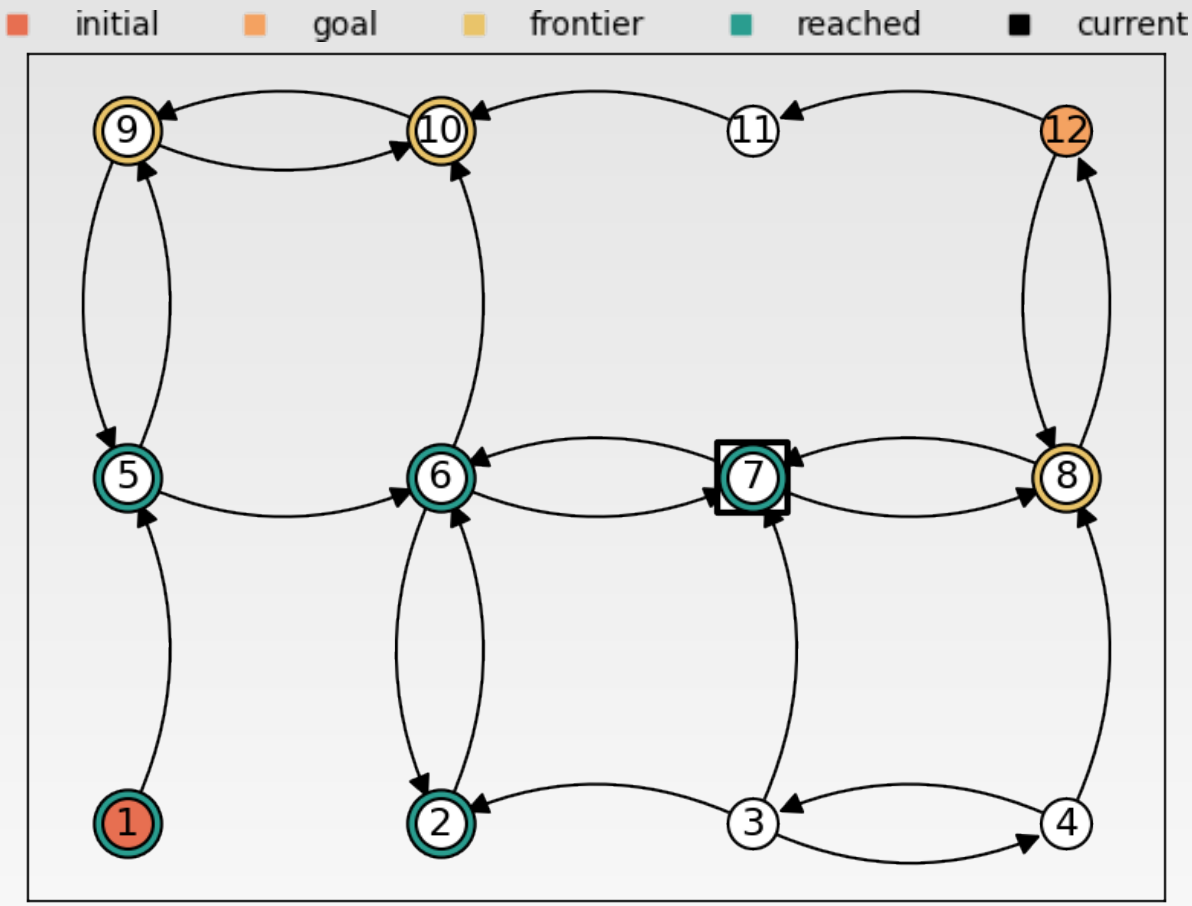
Iteration: 4, Current: 6
Frontier: [9, 10, 7, 2]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Depth-first (graph-like)



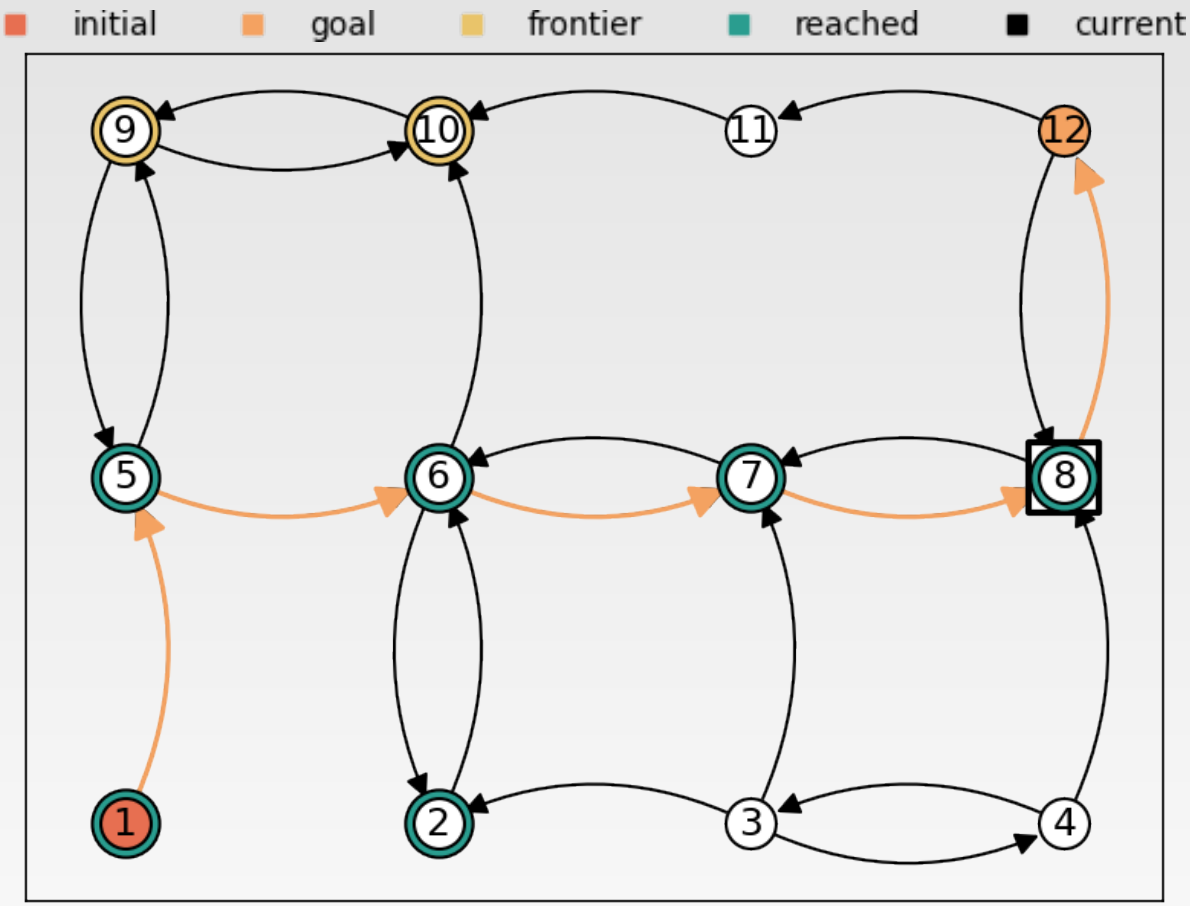
Iteration: 5, Current: 2
Frontier: [9, 10, 7]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Depth-first (graph-like)



Iteration: 6, Current: 7
Frontier: [9, 10, 8]
Reached: {1, 2, 5, 6, 7, 8, 9, 10}

Ex. 1 - Depth-first (graph-like)

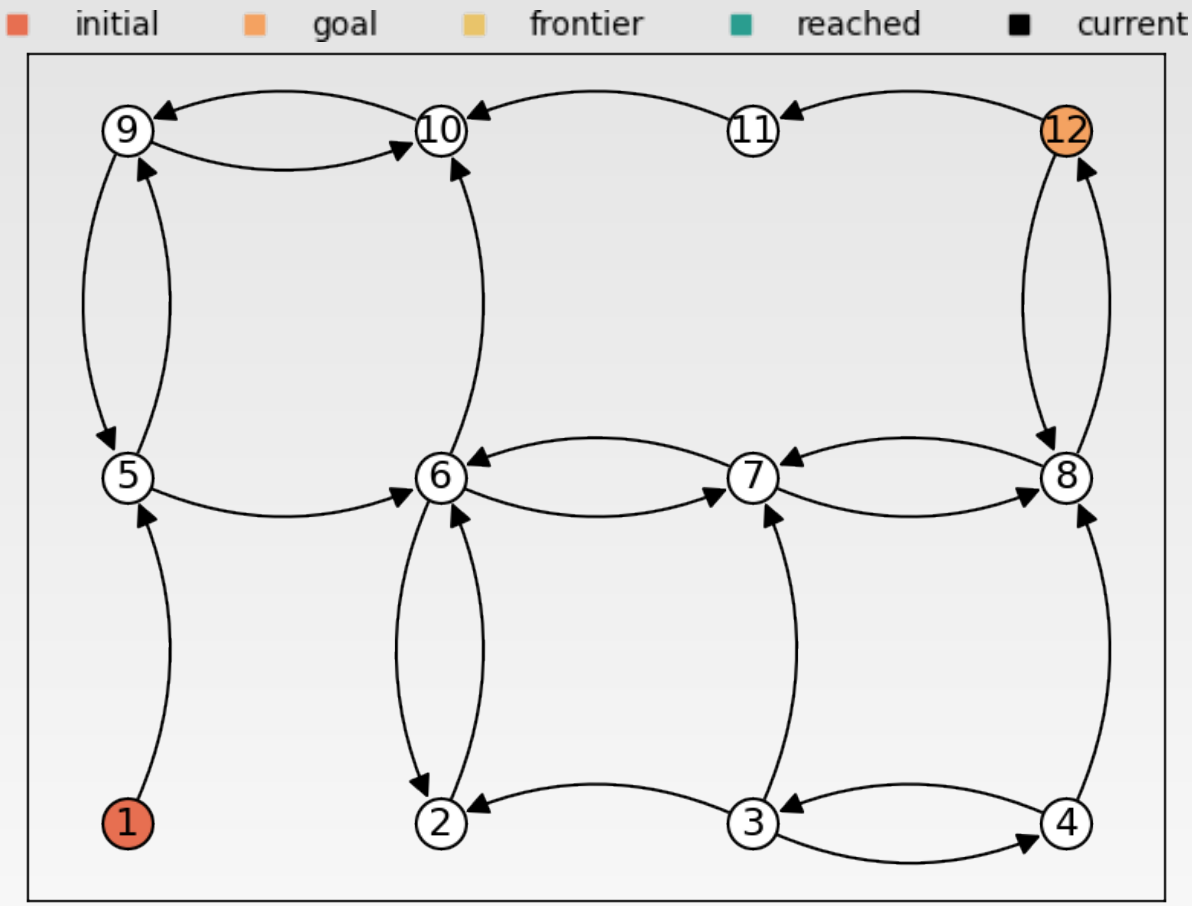


Iteration: 7, Current: 8
Frontier: [9, 10]
Reached: {1, 2, 5, 6, 7, 8, 9, 10}

Ex. 1 - Depth-first (graph-like)

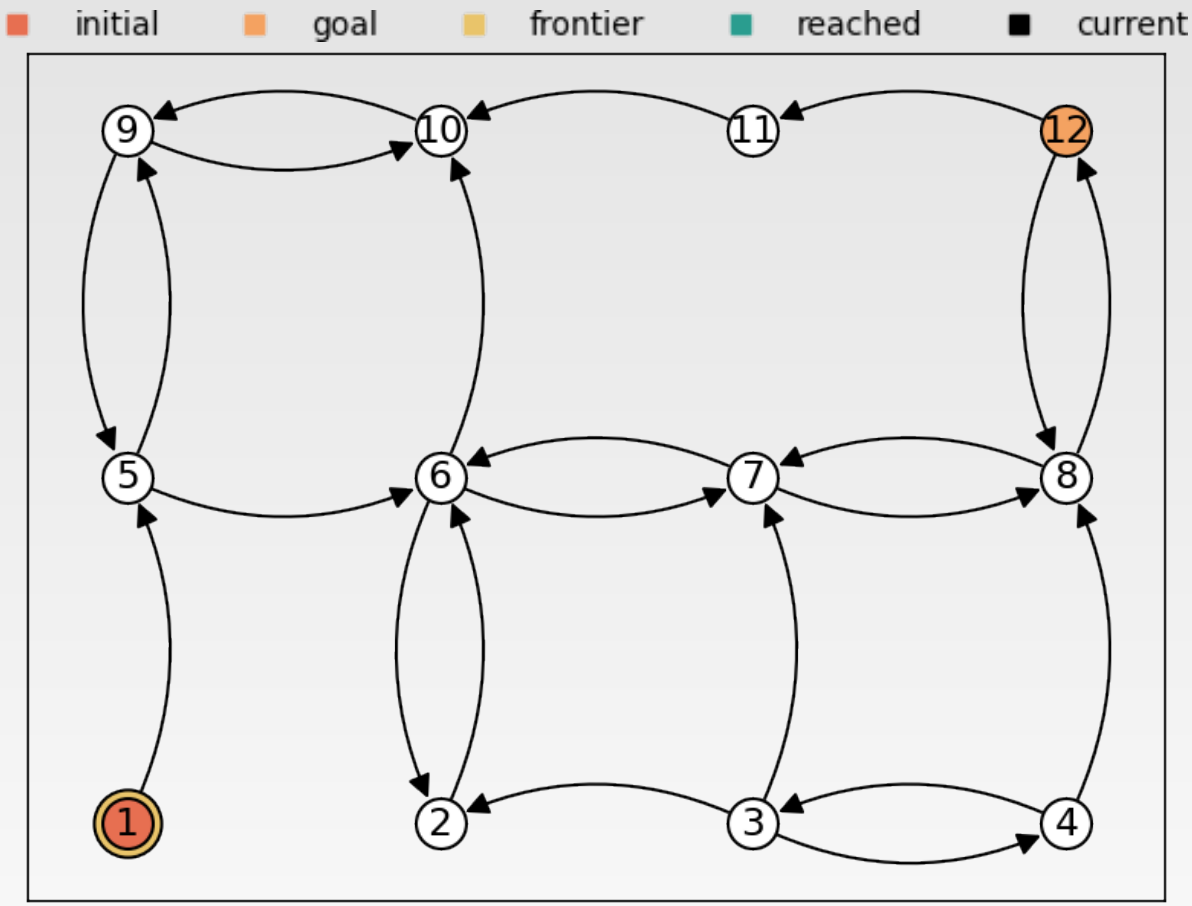
Done! *(at least we avoided expanding 9, 10)*

Ex. 1 - Iterative deepening



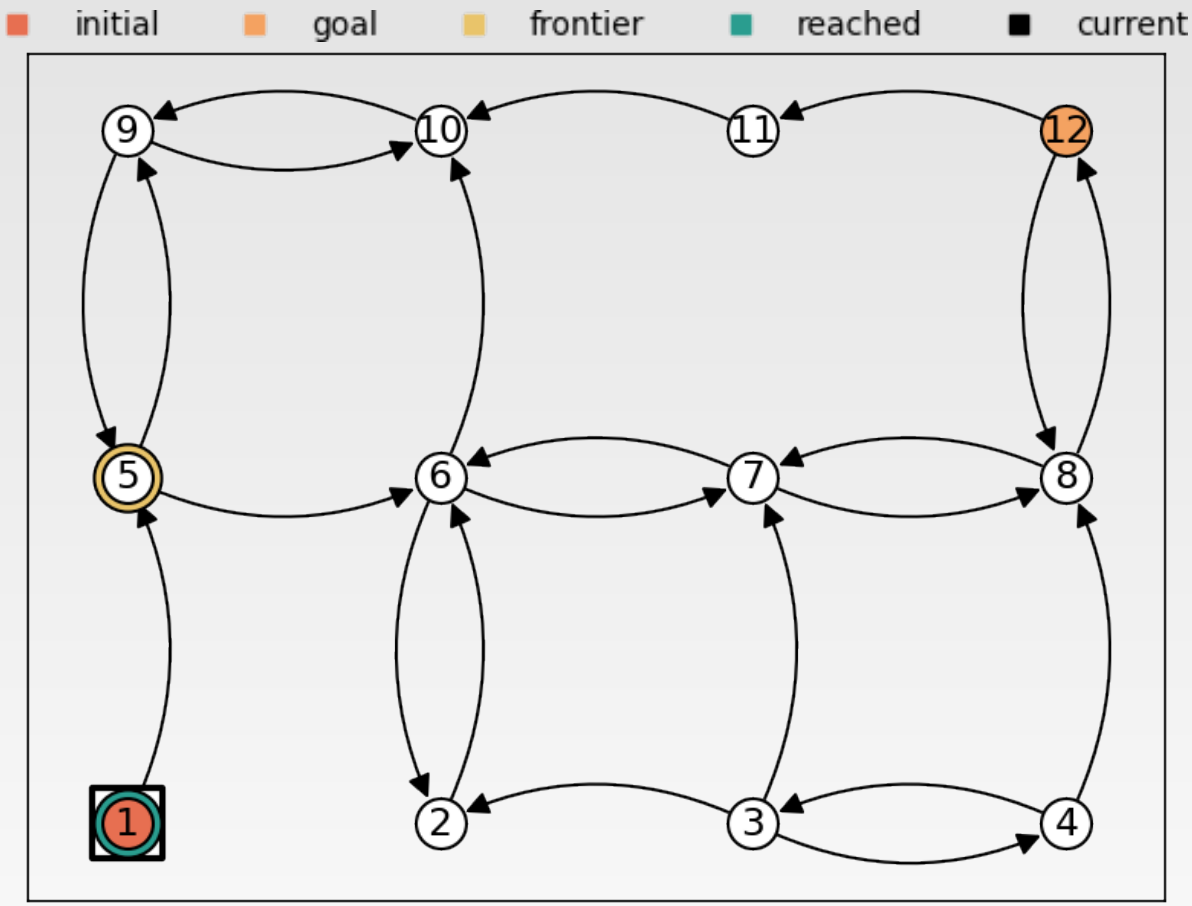
Iteration: -, Current:
Frontier: []
Reached: {}

Ex. 1 - Iterative deepening



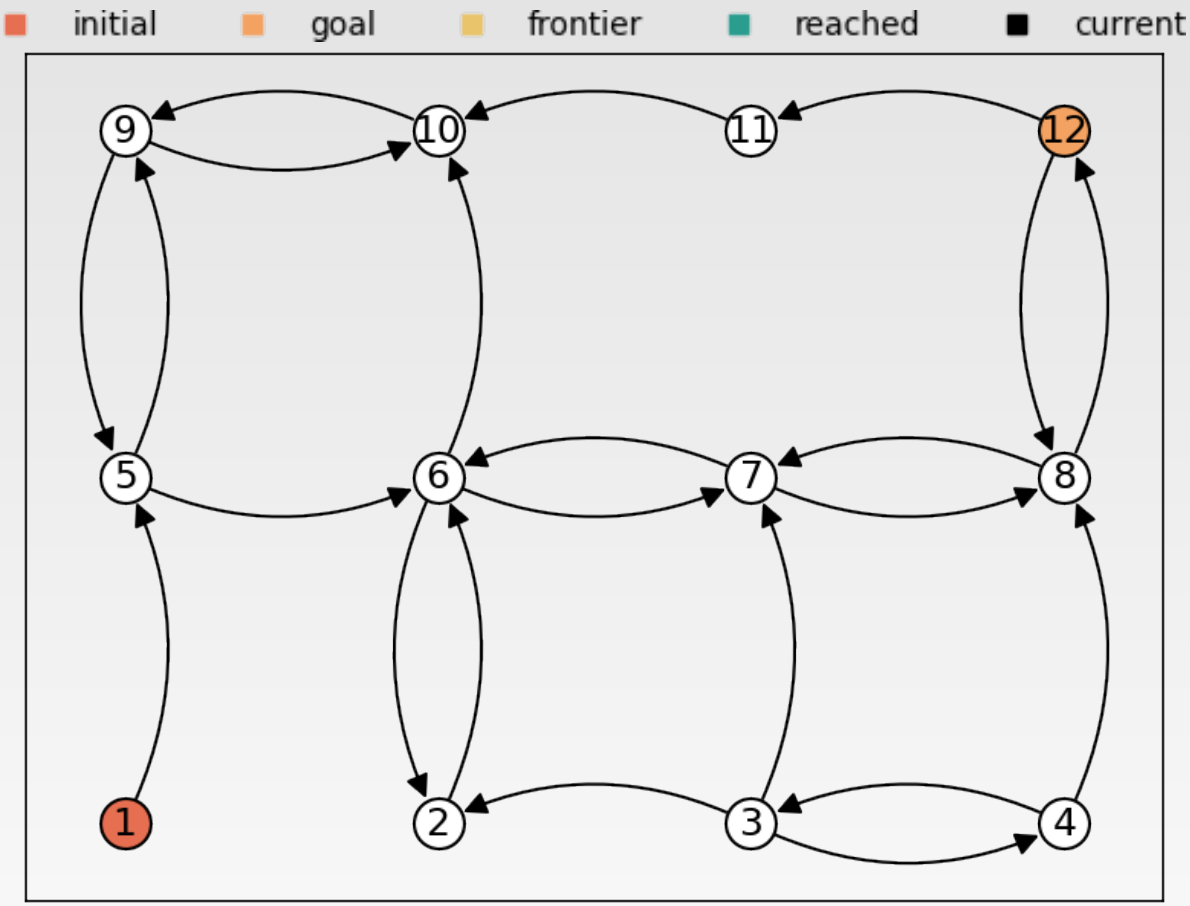
Iteration: 1, Current:
Frontier: [1]
Reached: {1}

Ex. 1 - Iterative deepening



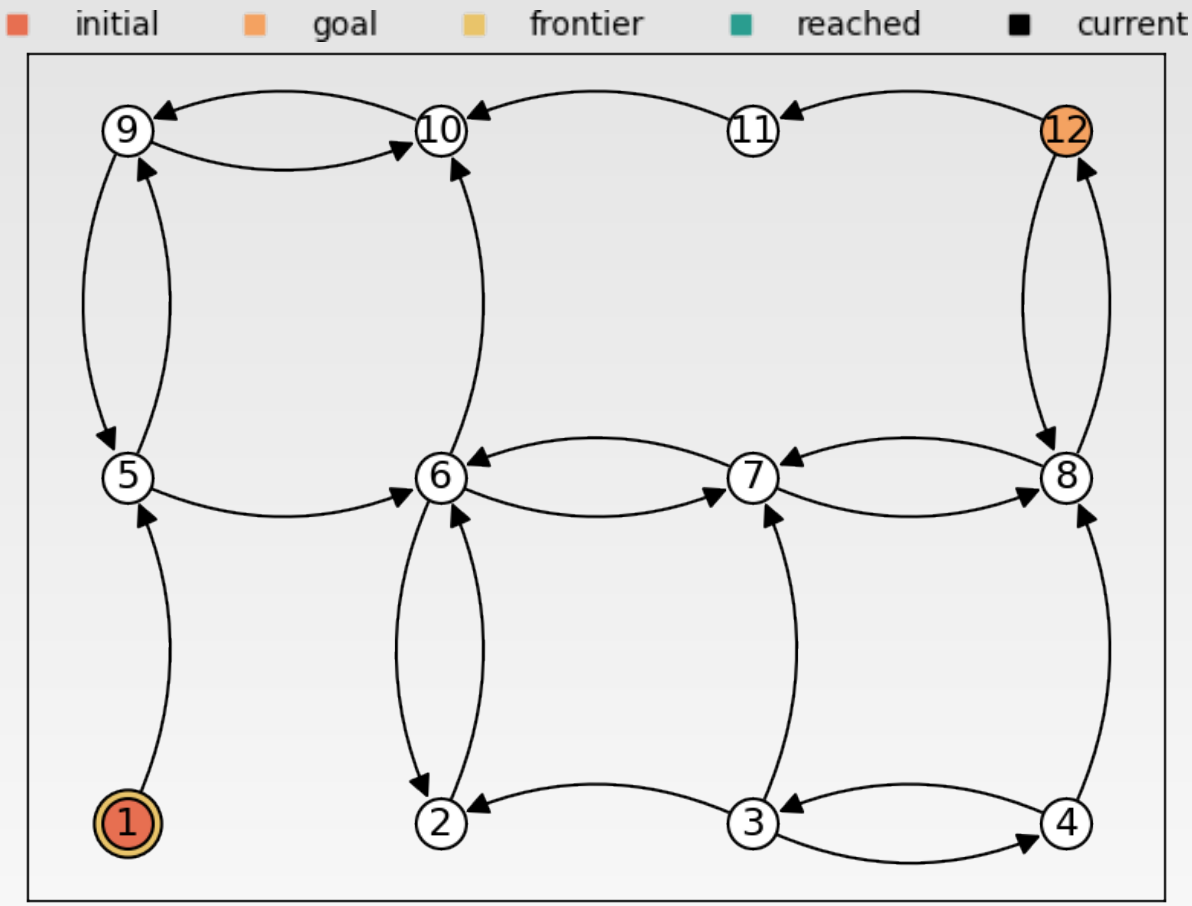
Iteration: 2, Current: 1
Frontier: [5]
Reached: {1, 5}

Ex. 1 - Iterative deepening



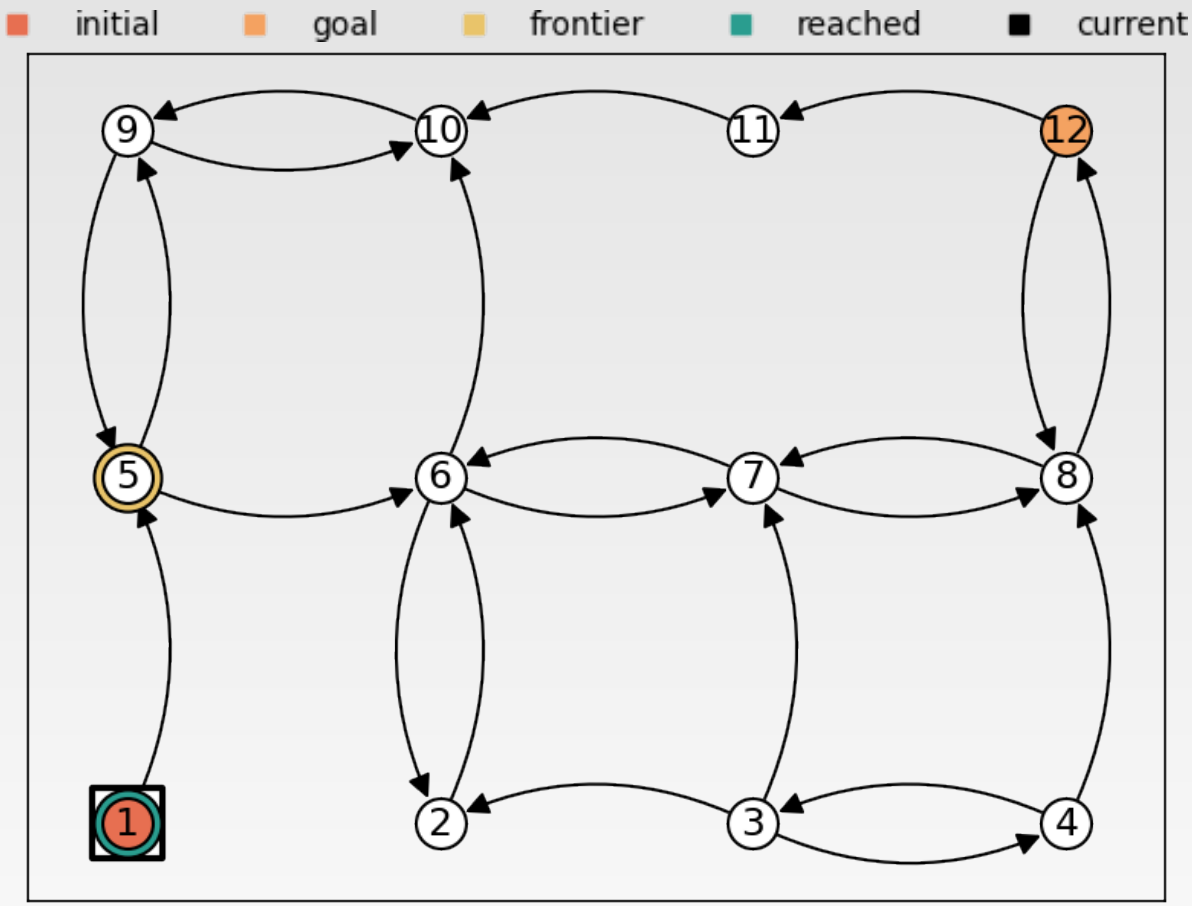
Iteration: 3, Current:
Frontier: []
Reached: {}

Ex. 1 - Iterative deepening



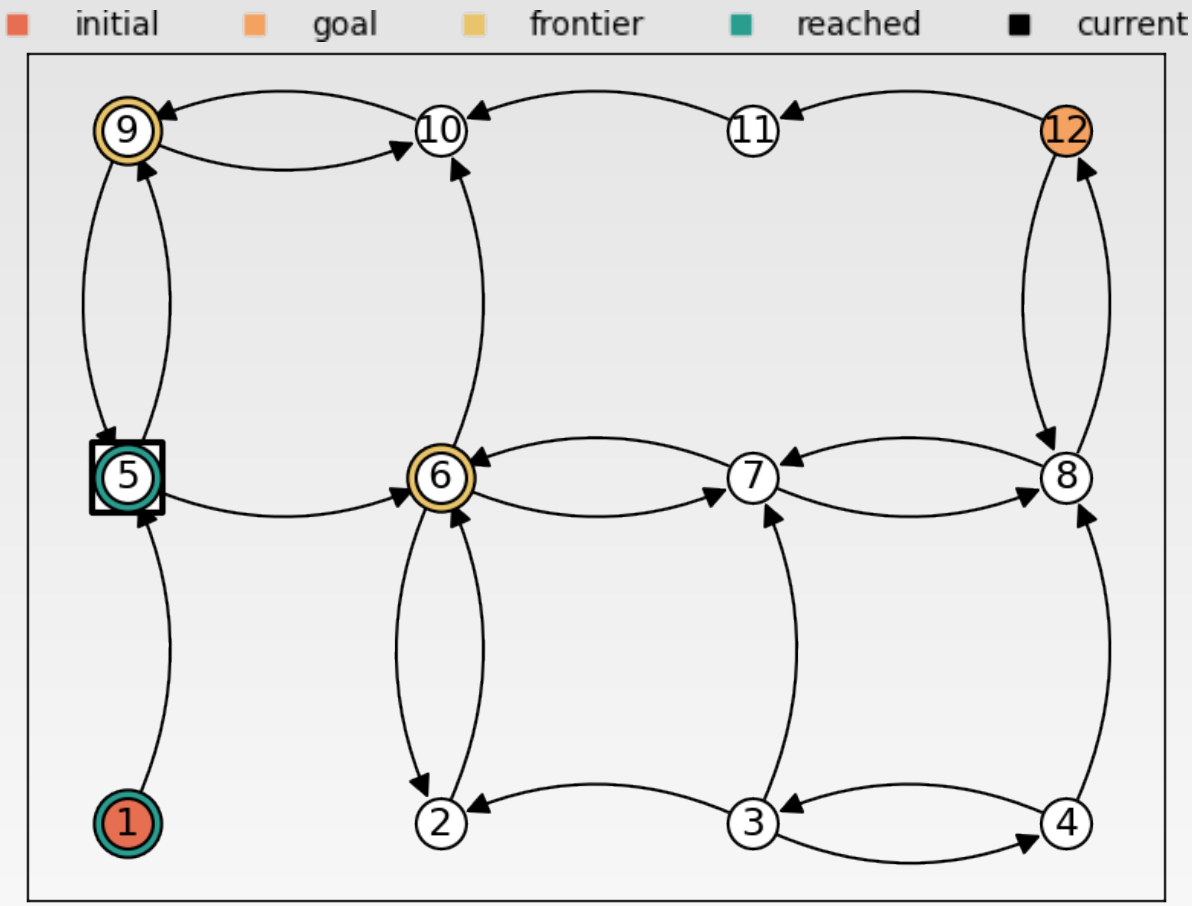
Iteration: 4, Current:
Frontier: [1]
Reached: {1}

Ex. 1 - Iterative deepening



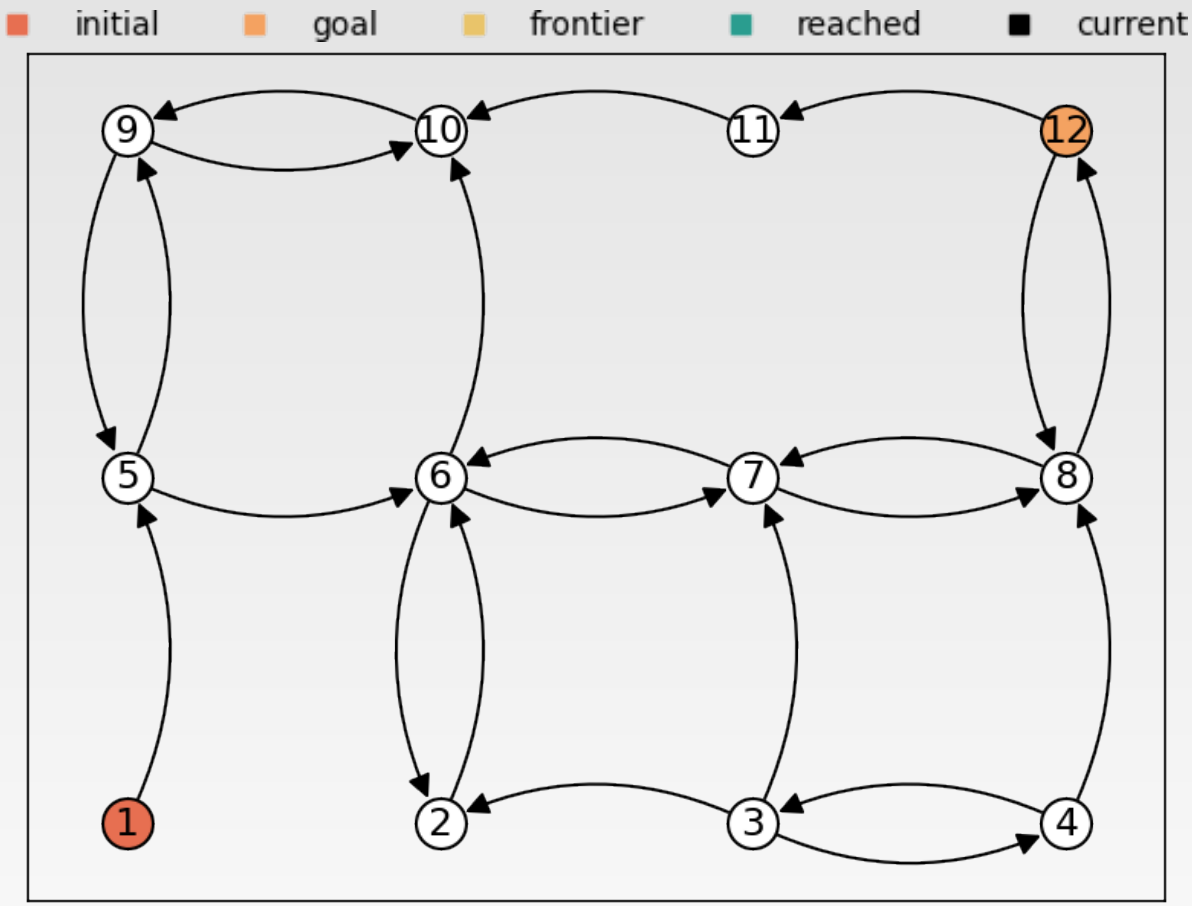
Iteration: 5, Current: 1
Frontier: [5]
Reached: {1, 5}

Ex. 1 - Iterative deepening



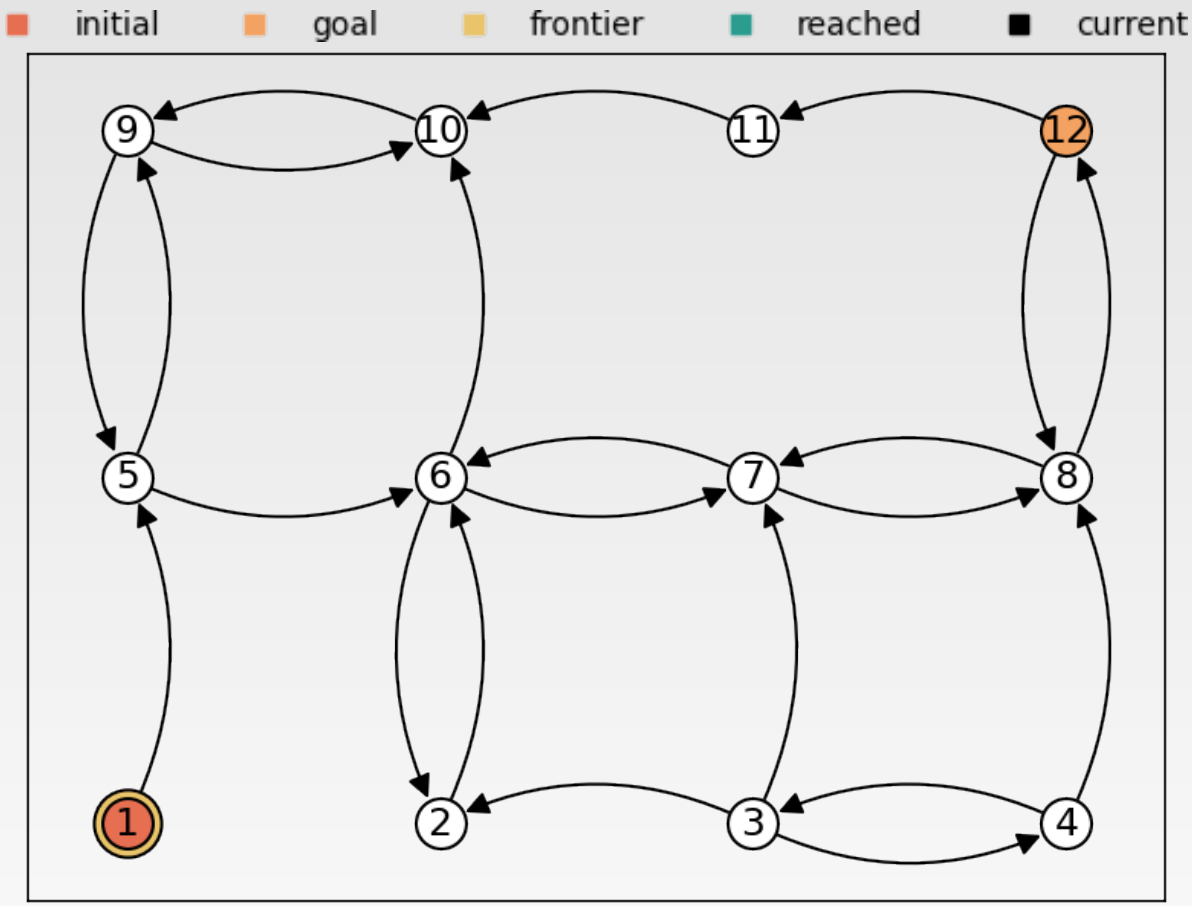
Iteration: 6, Current: 5
Frontier: [9, 6]
Reached: {1, 5, 9, 6}

Ex. 1 - Iterative deepening



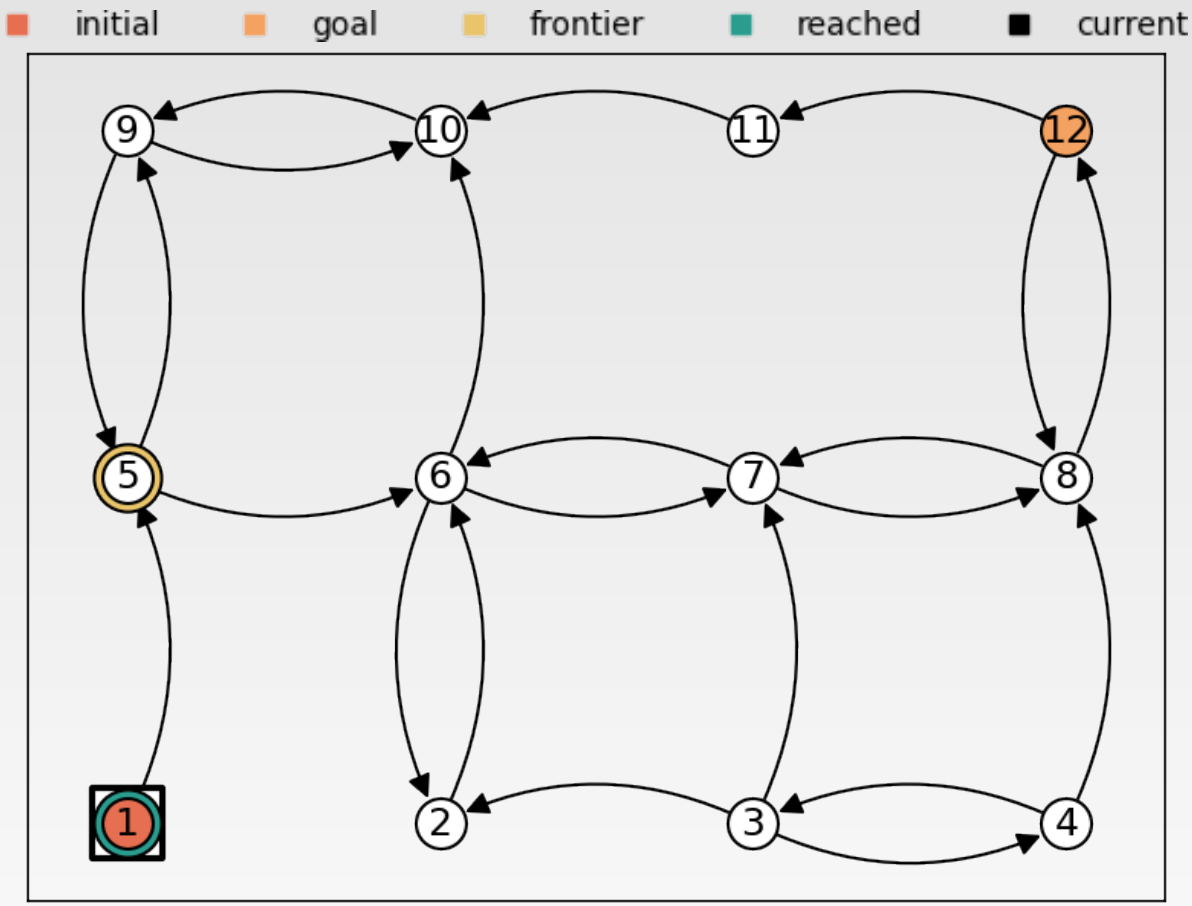
Iteration: 8, Current:
Frontier: []
Reached: {}

Ex. 1 - Iterative deepening



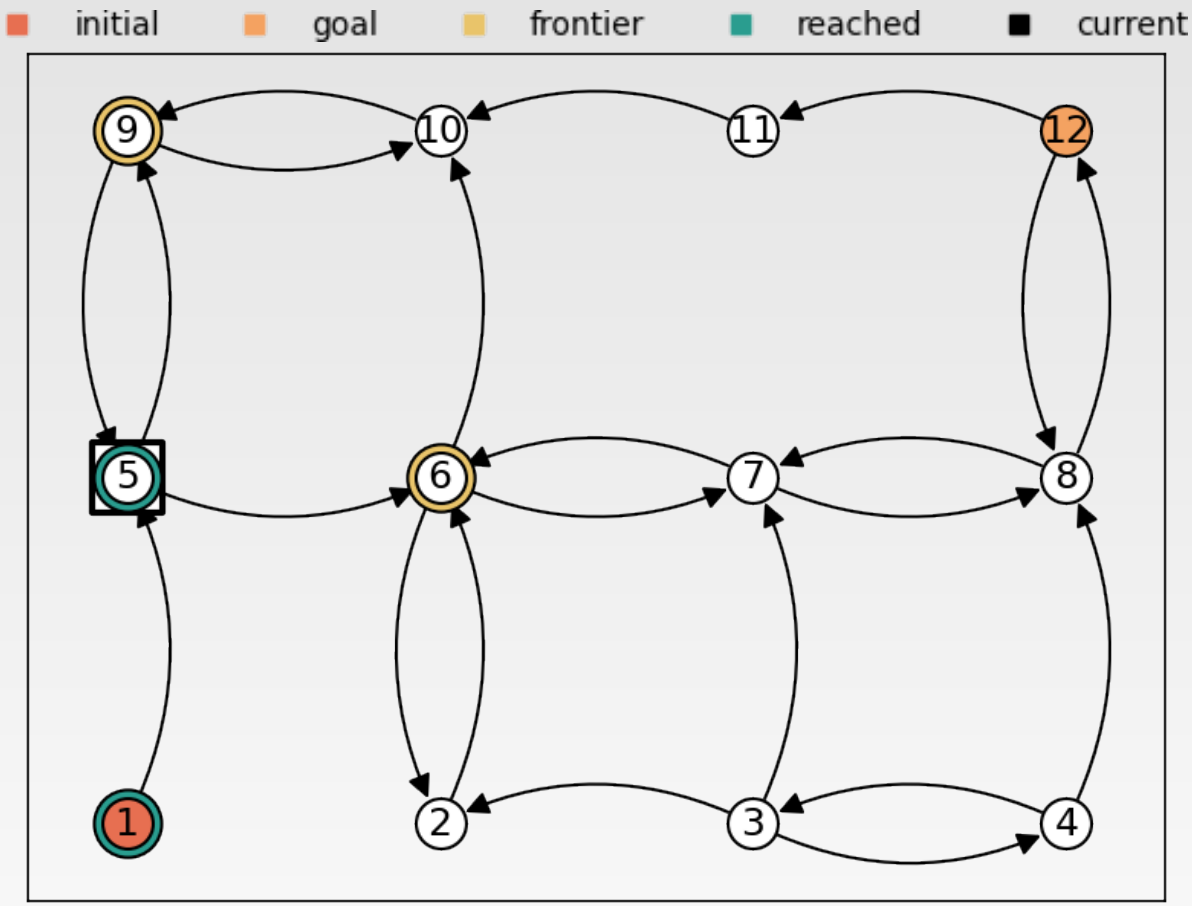
Iteration: 9, Current:
Frontier: [1]
Reached: {1}

Ex. 1 - Iterative deepening



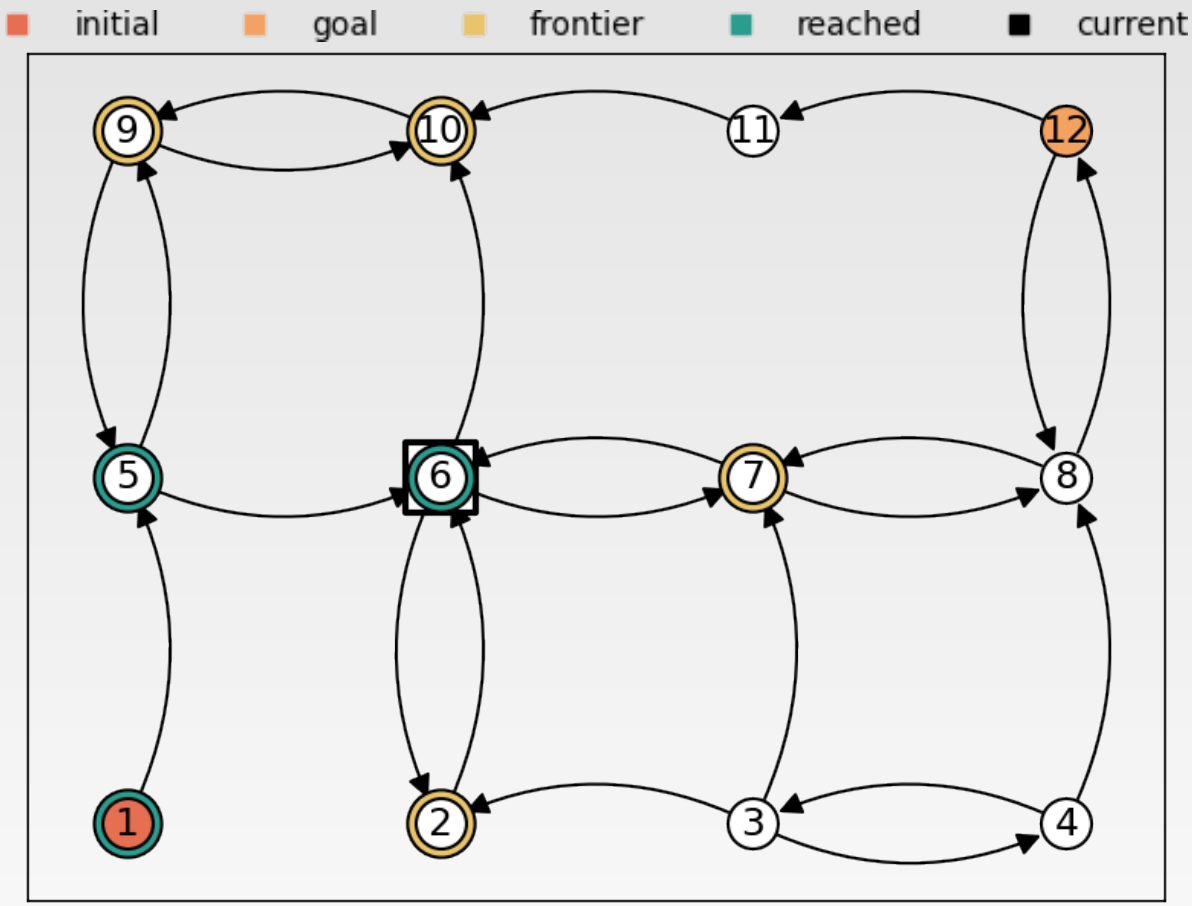
Iteration: 10, Current: 1
Frontier: [5]
Reached: {1, 5}

Ex. 1 - Iterative deepening



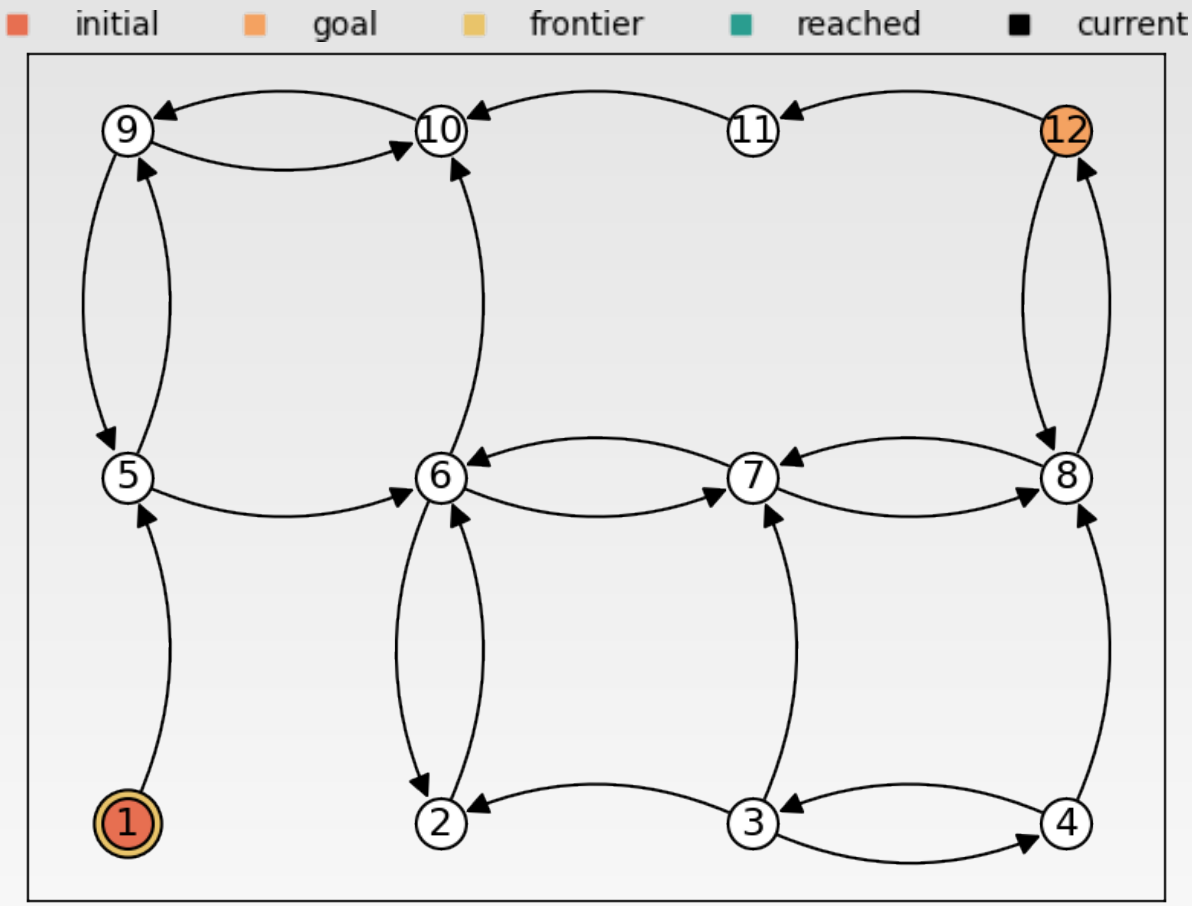
Iteration: 11, Current: 5
Frontier: [9, 6]
Reached: {1, 5, 9, 6}

Ex. 1 - Iterative deepening



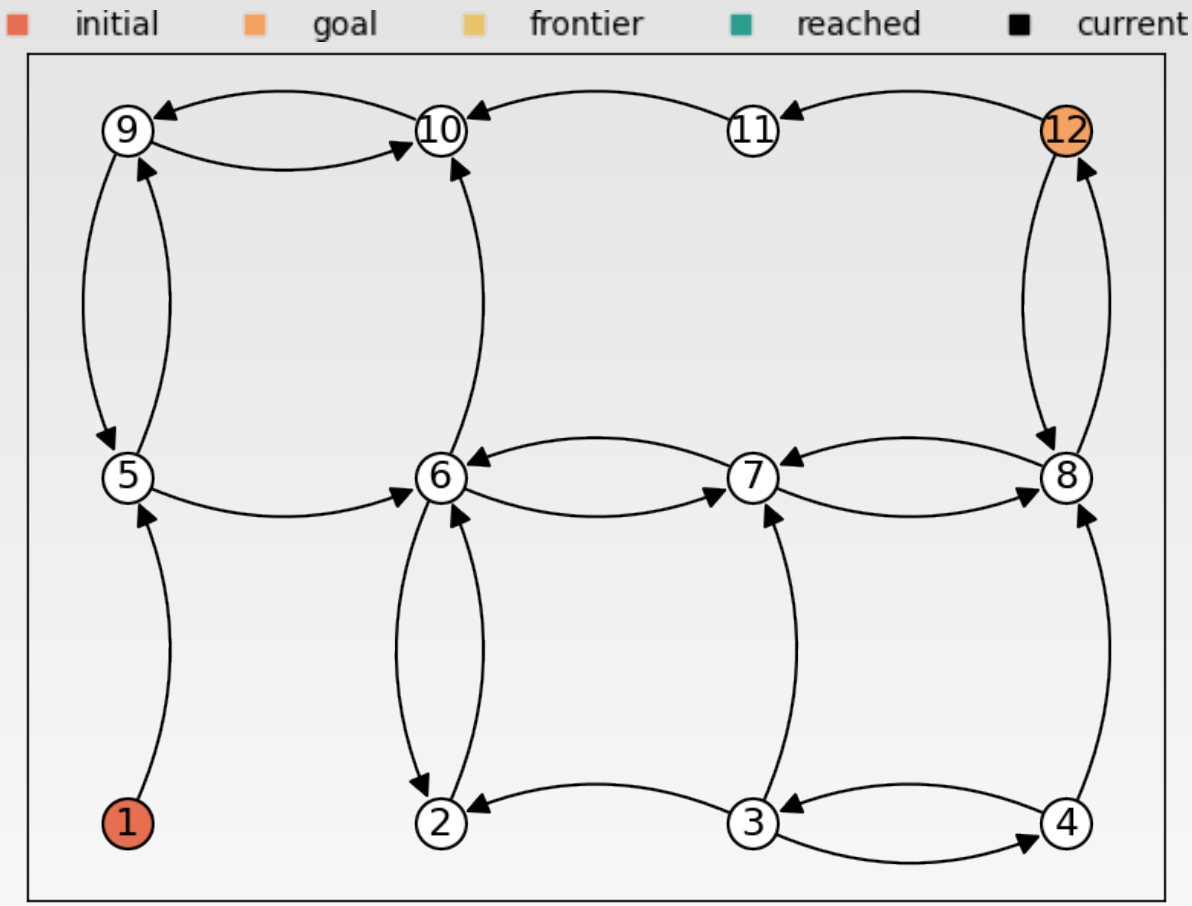
Iteration: 12, Current: 6
Frontier: [9, 10, 7, 2]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Iterative deepening



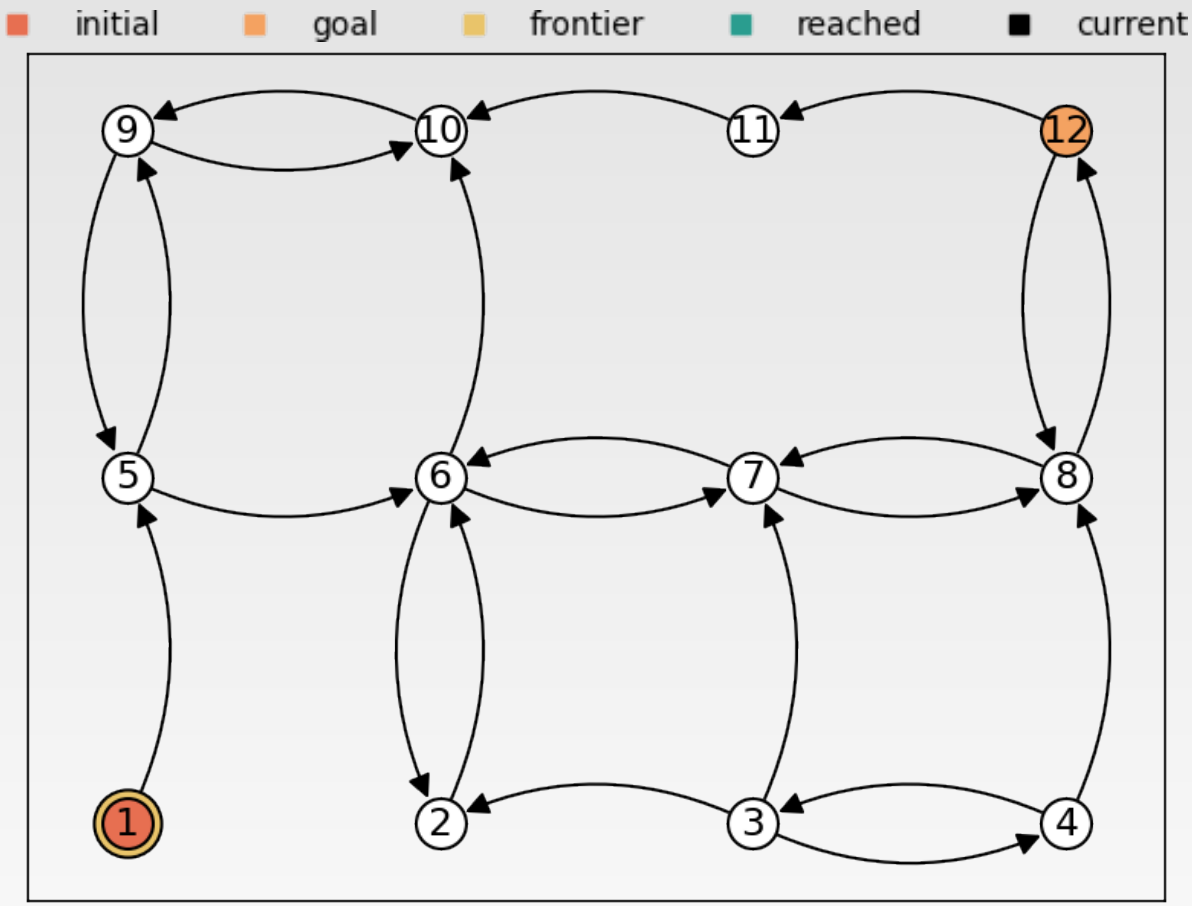
Iteration: 16, Current: 9
Frontier: []
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Iterative deepening



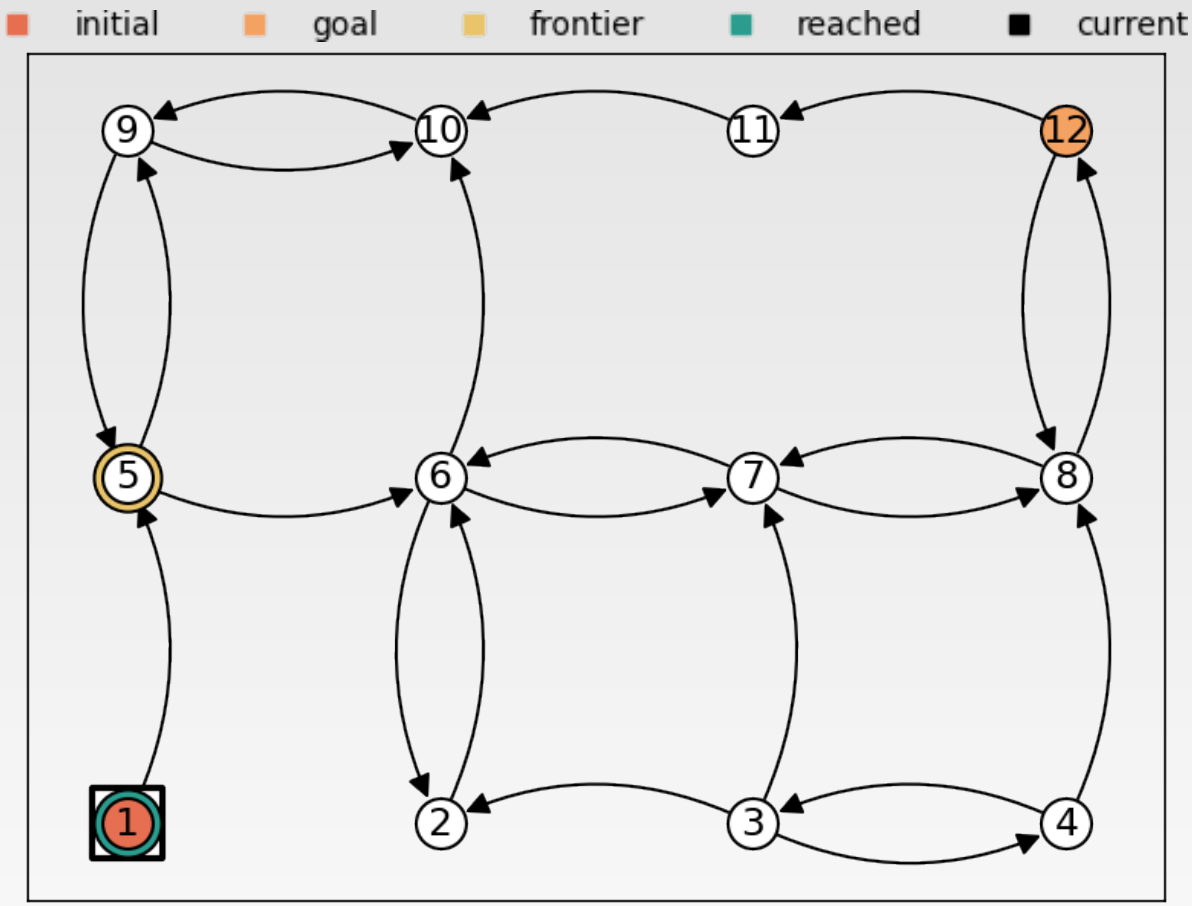
Iteration: 15, Current:
Frontier: []
Reached: {}

Ex. 1 - Iterative deepening



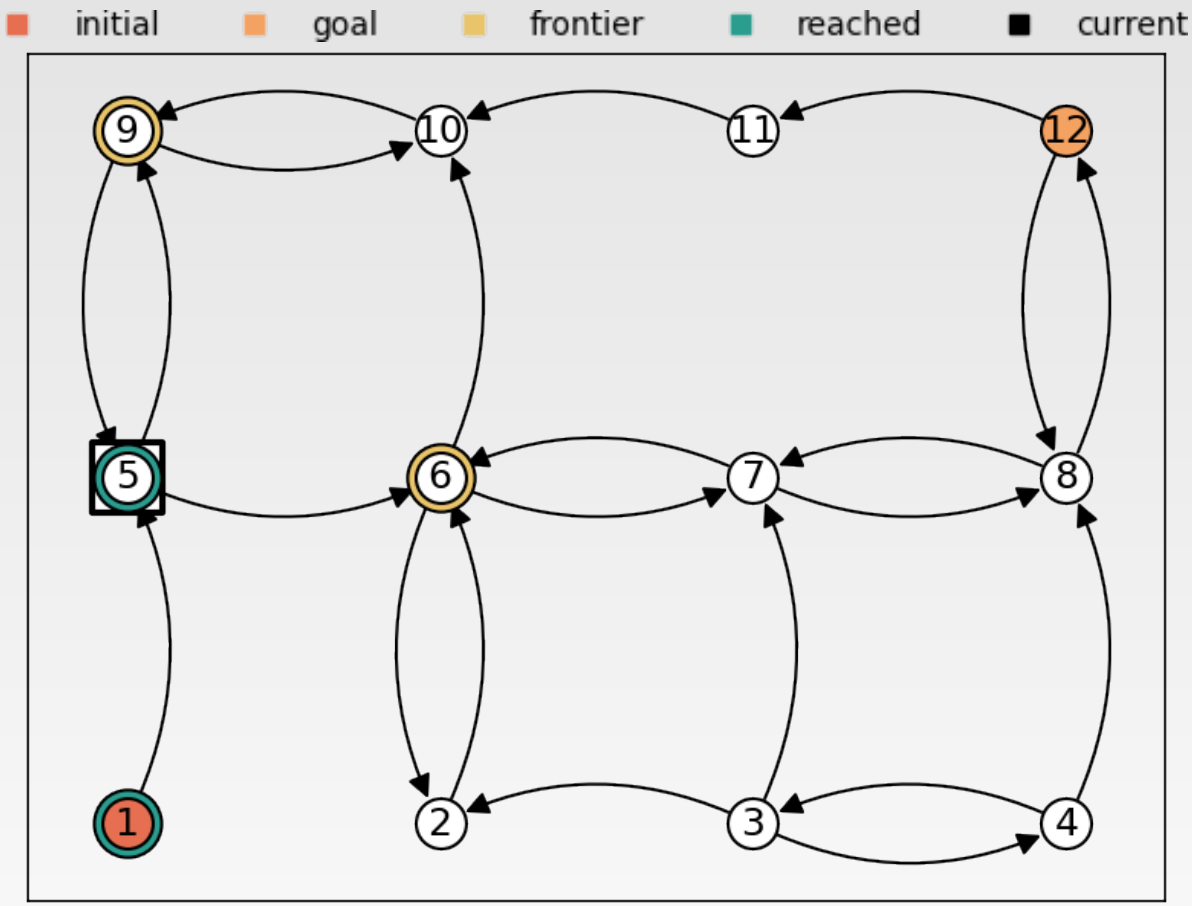
Iteration: 16, Current:
Frontier: [1]
Reached: {1}

Ex. 1 - Iterative deepening



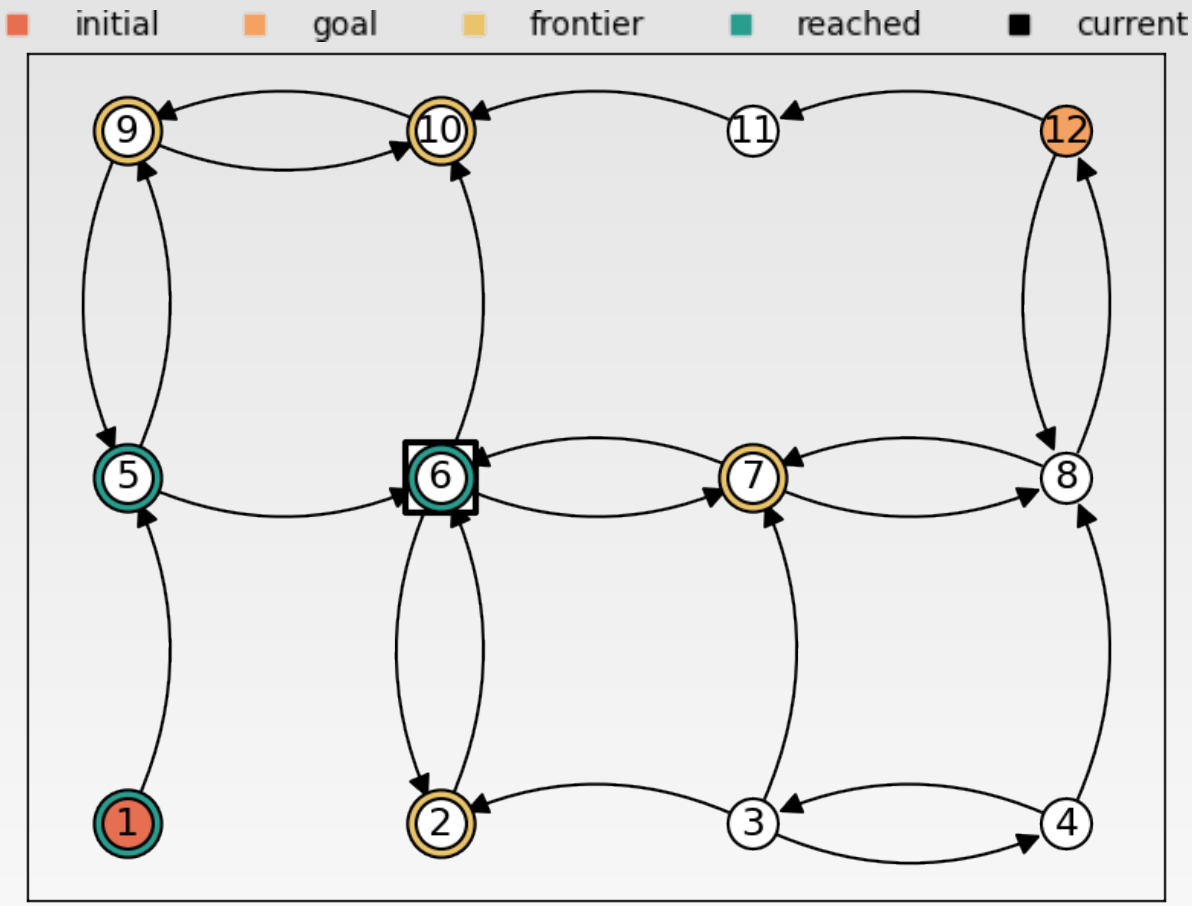
Iteration: 17, Current: 1
Frontier: [5]
Reached: {1, 5}

Ex. 1 - Iterative deepening



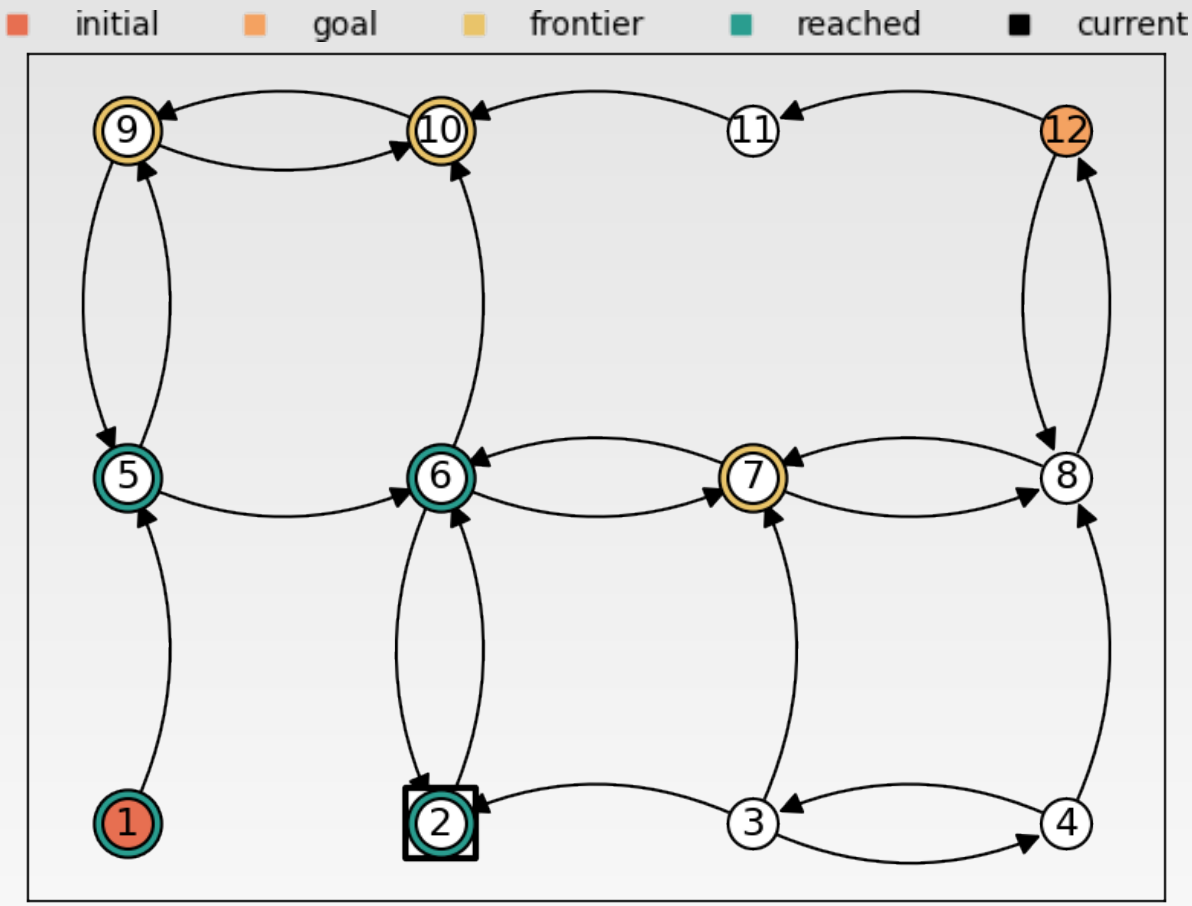
Iteration: 18, Current: 5
Frontier: [9, 6]
Reached: {1, 5, 9, 6}

Ex. 1 - Iterative deepening



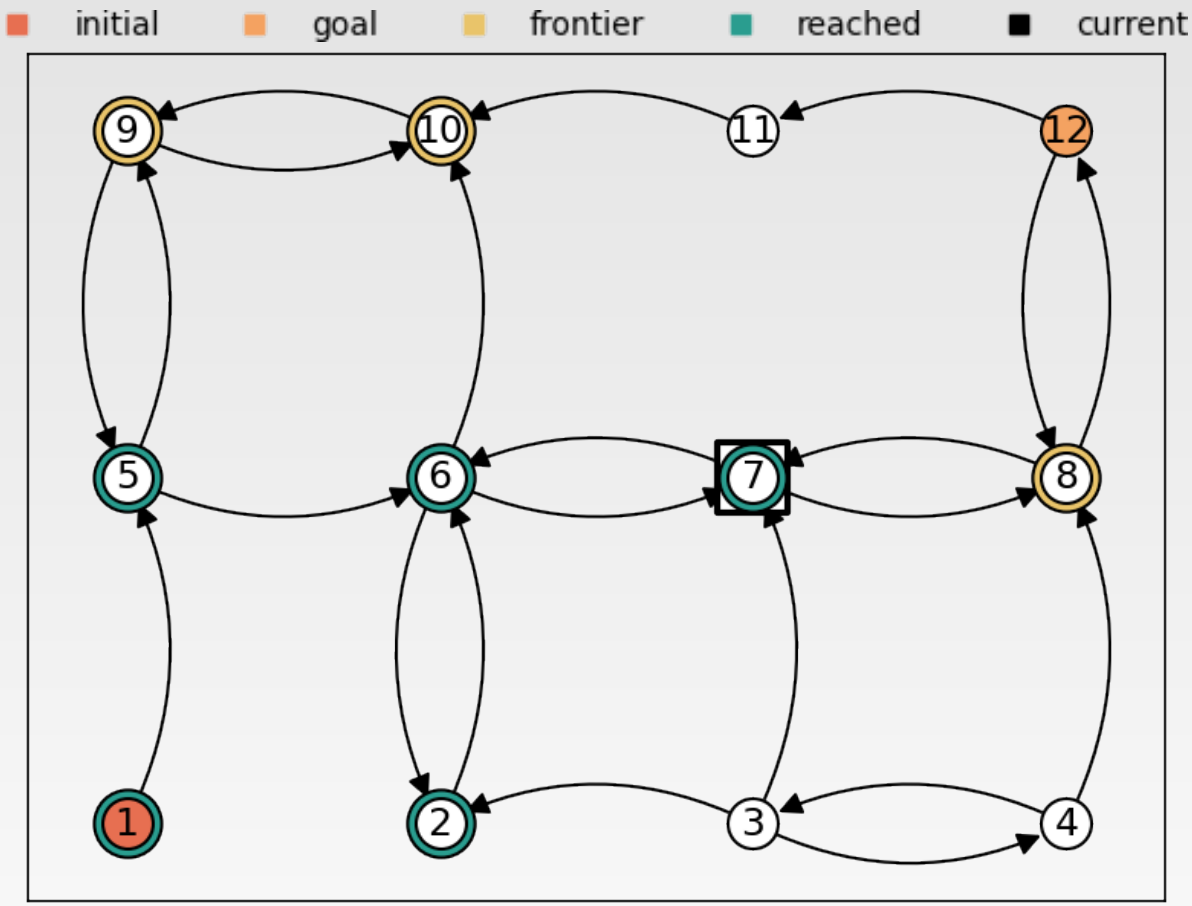
Iteration: 19, Current: 6
Frontier: [9, 10, 7, 2]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Iterative deepening



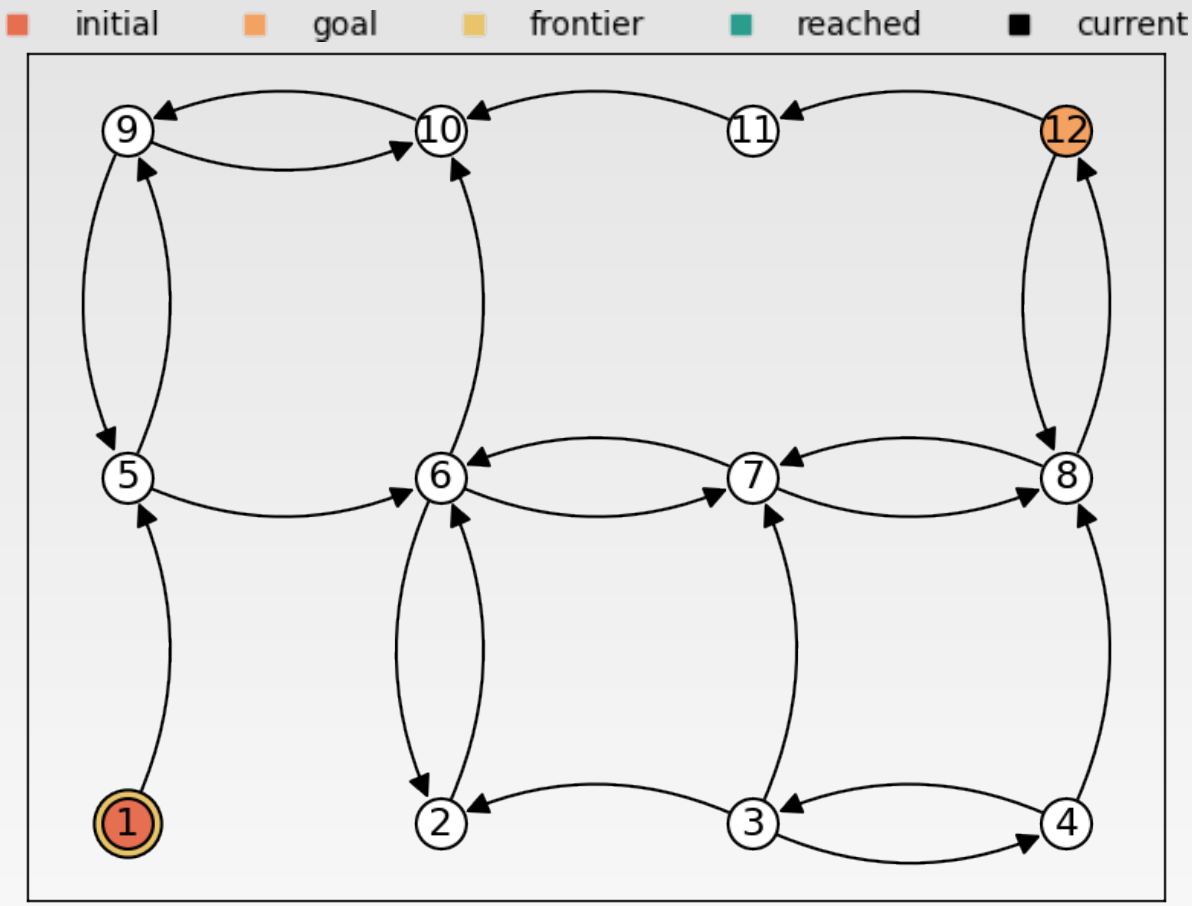
Iteration: 20, Current: 2
Frontier: [9, 10, 7]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Iterative deepening



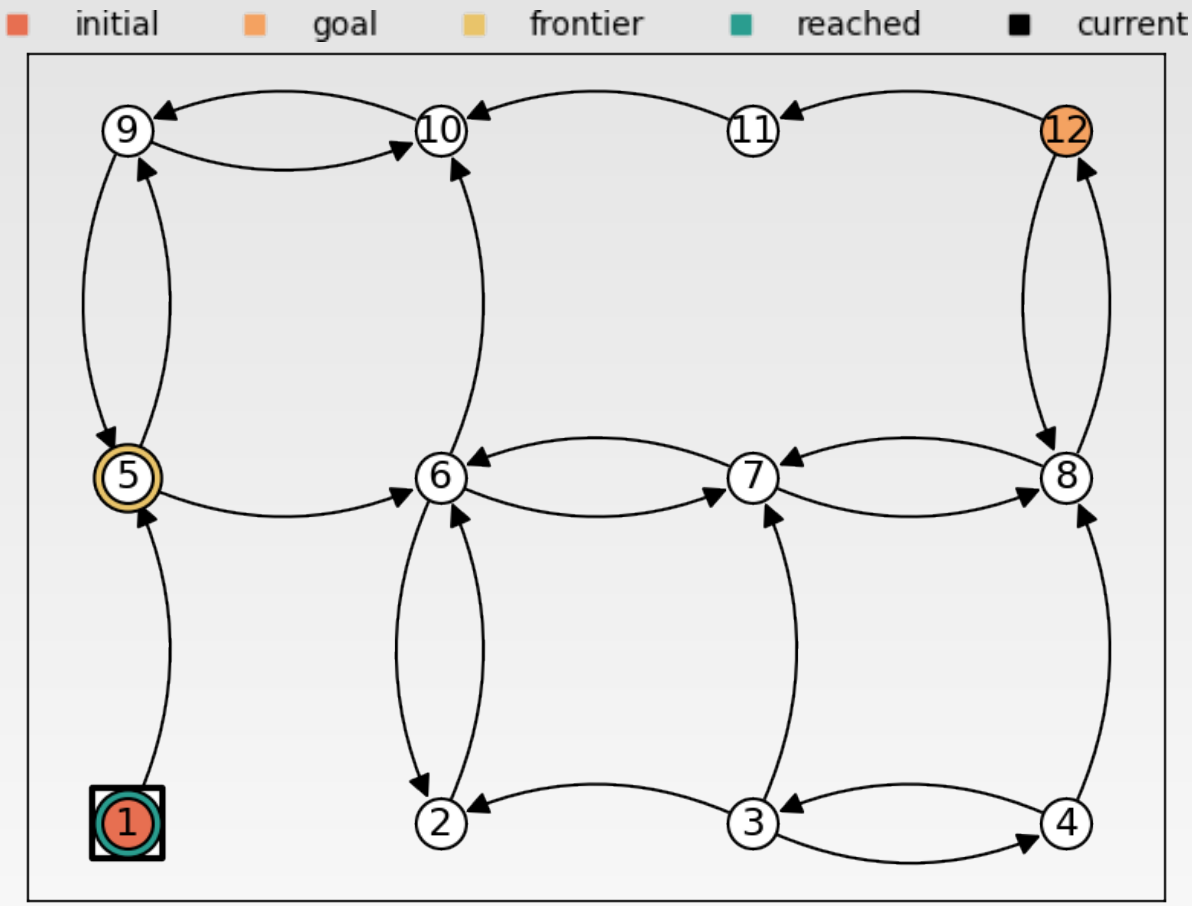
Iteration: 21, Current: 7
Frontier: [9, 10, 8]
Reached: {1, 2, 5, 6, 7, 8, 9, 10}

Ex. 1 - Iterative deepening



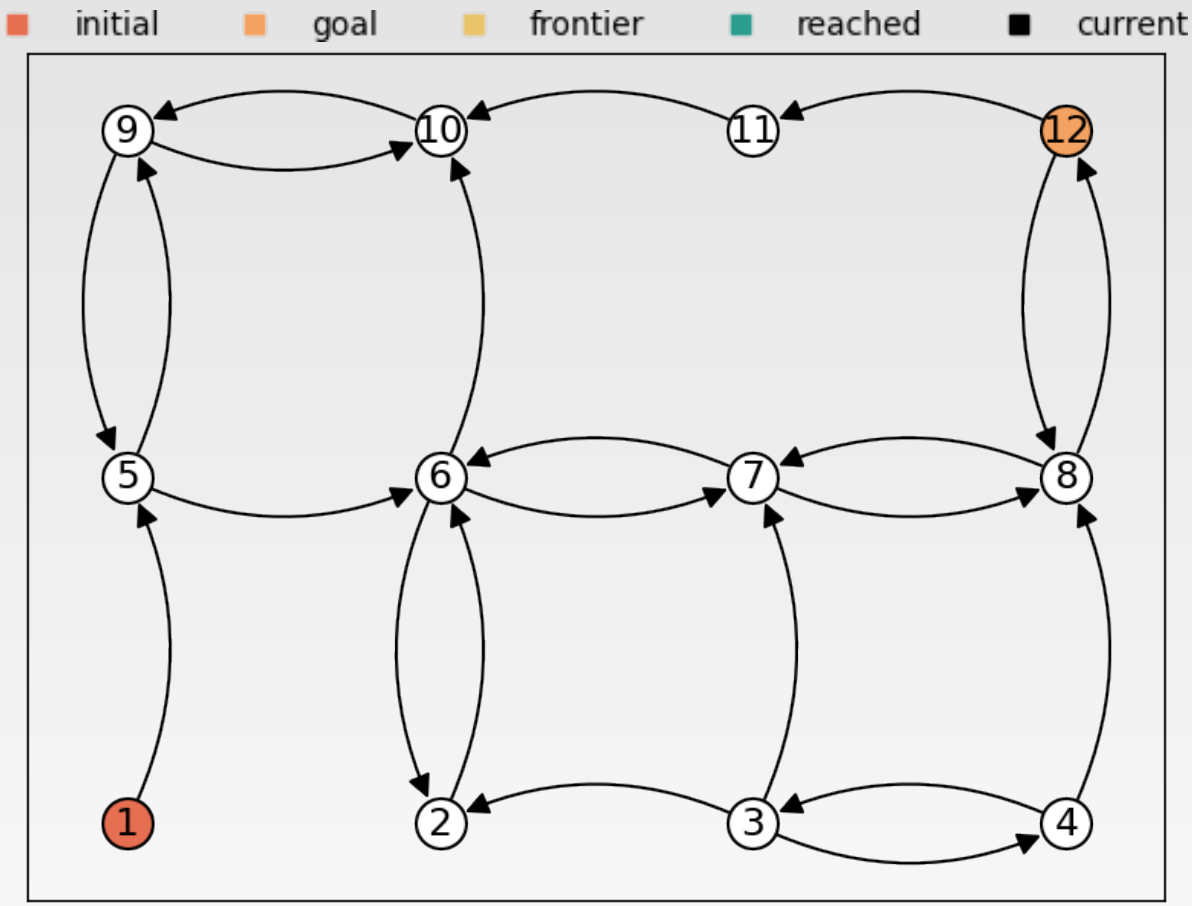
Iteration: 23, Current: 10
Frontier: [9]
Reached: {1, 2, 5, 6, 7, 8, 9, 10}

Ex. 1 - Iterative deepening



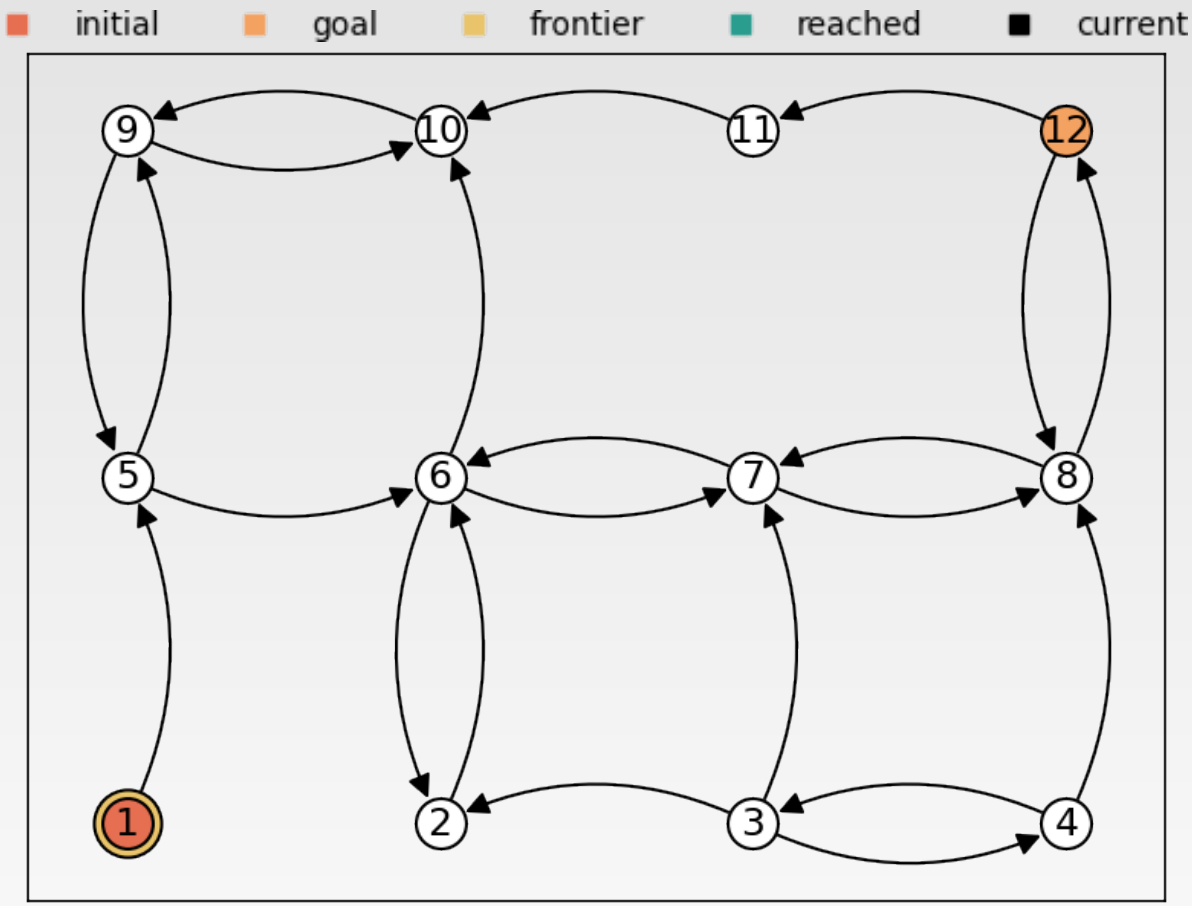
Iteration: 24, Current: 9
Frontier: []
Reached: {1, 2, 5, 6, 7, 8, 9, 10}

Ex. 1 - Iterative deepening



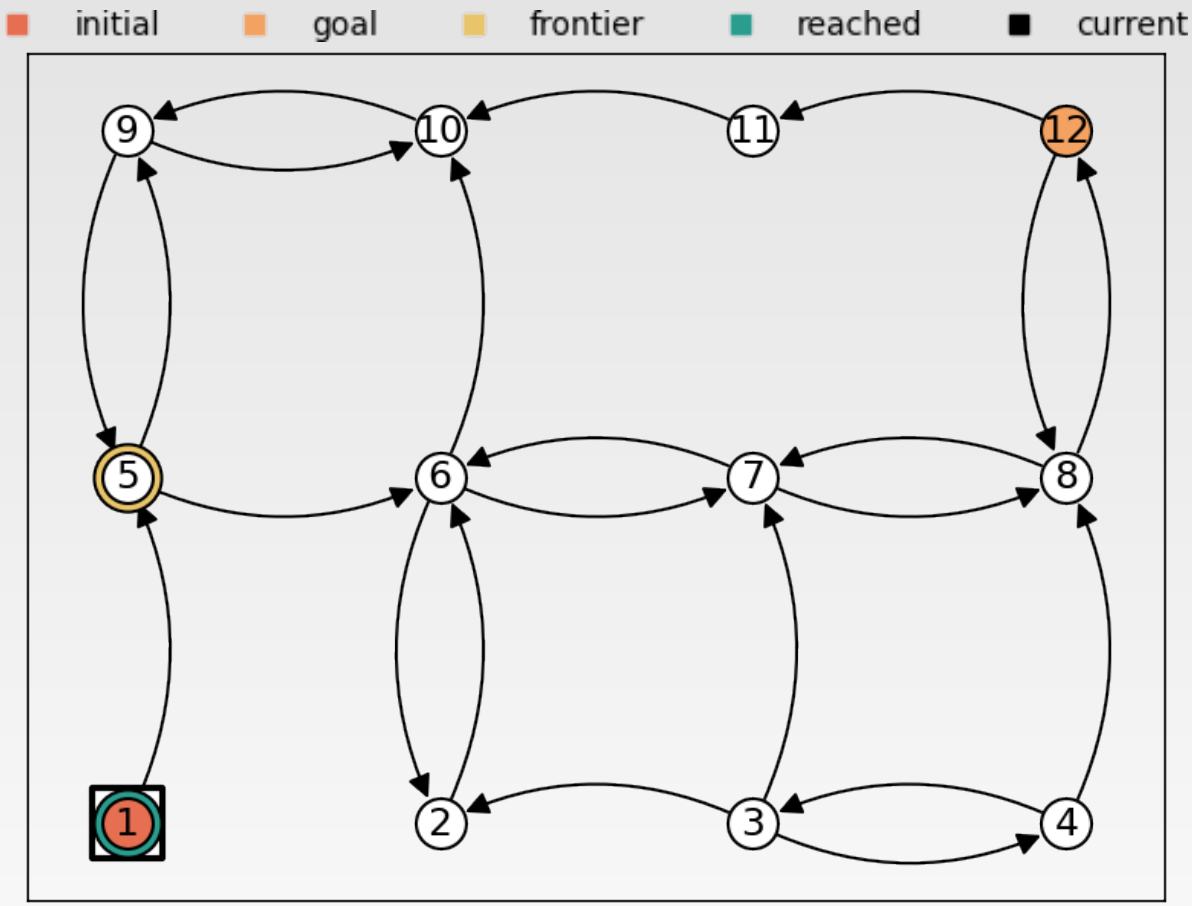
Iteration: 22, Current:
Frontier: []
Reached: {}

Ex. 1 - Iterative deepening



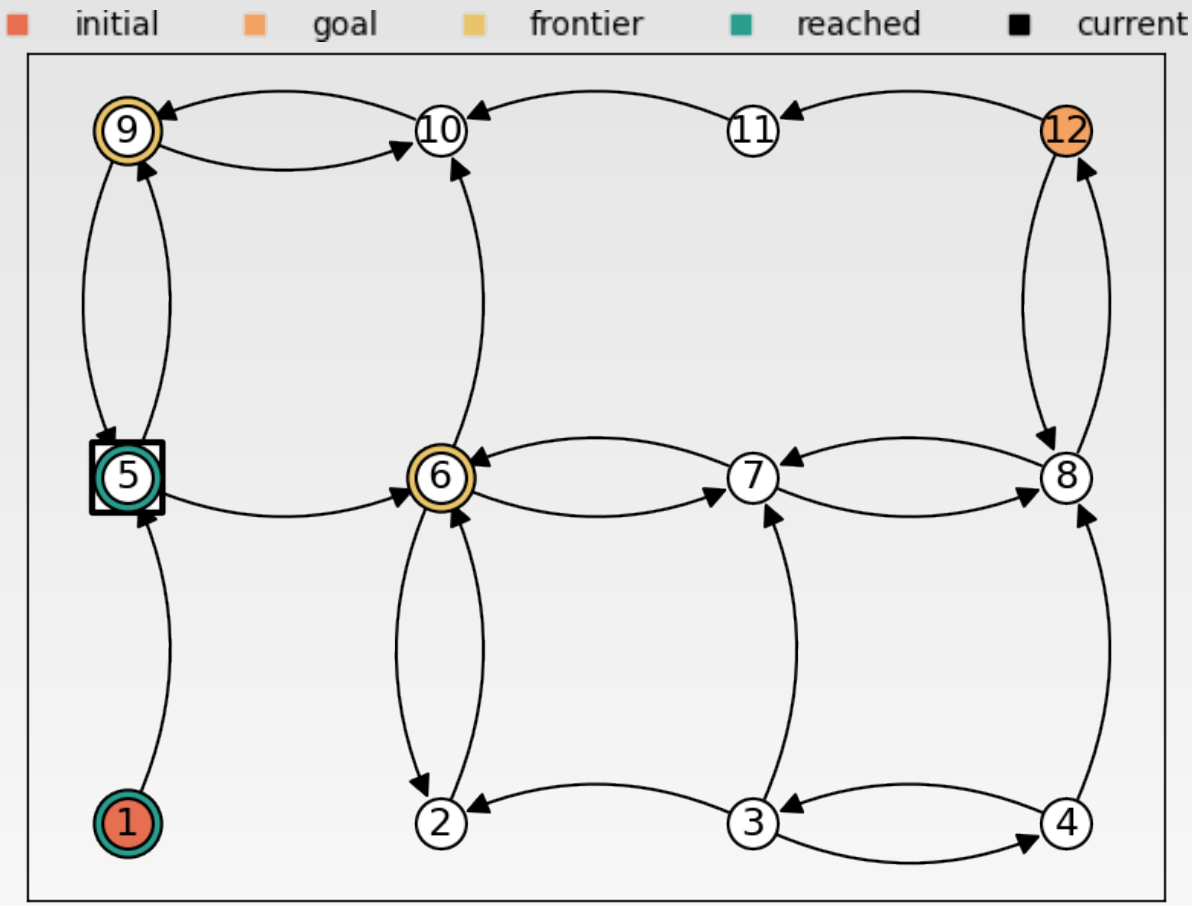
Iteration: 23, Current:
Frontier: [1]
Reached: {1}

Ex. 1 - Iterative deepening



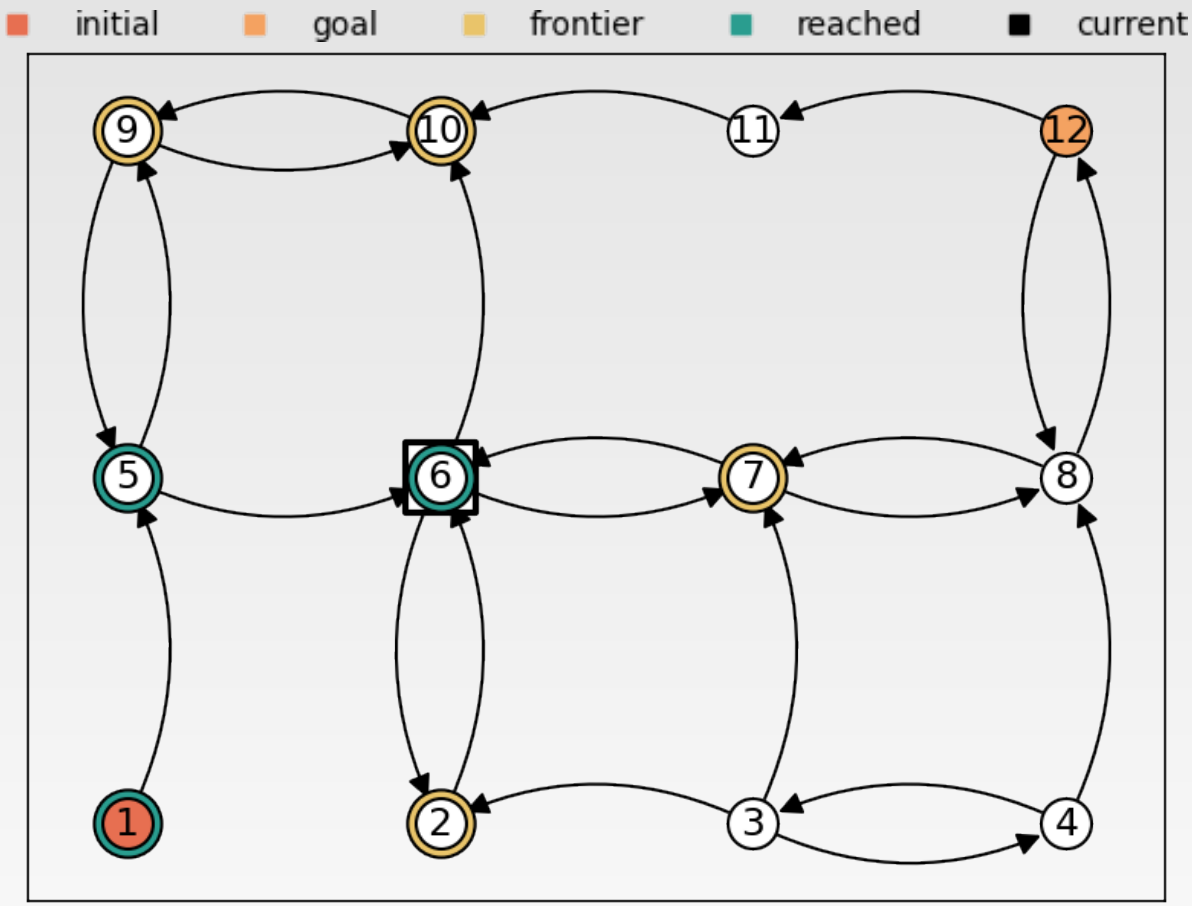
Iteration: 24, Current: 1
Frontier: [5]
Reached: {1, 5}

Ex. 1 - Iterative deepening



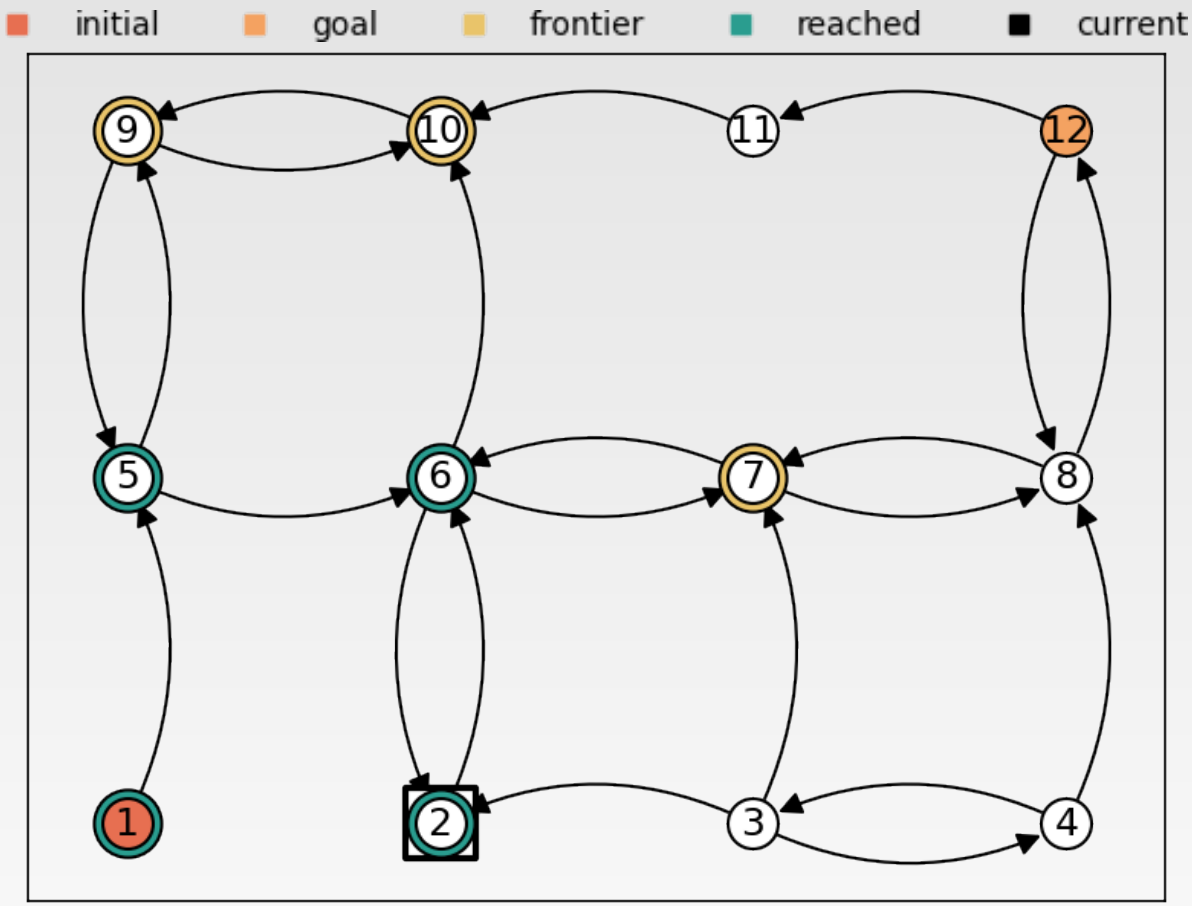
Iteration: 25, Current: 5
Frontier: [9, 6]
Reached: {1, 5, 9, 6}

Ex. 1 - Iterative deepening



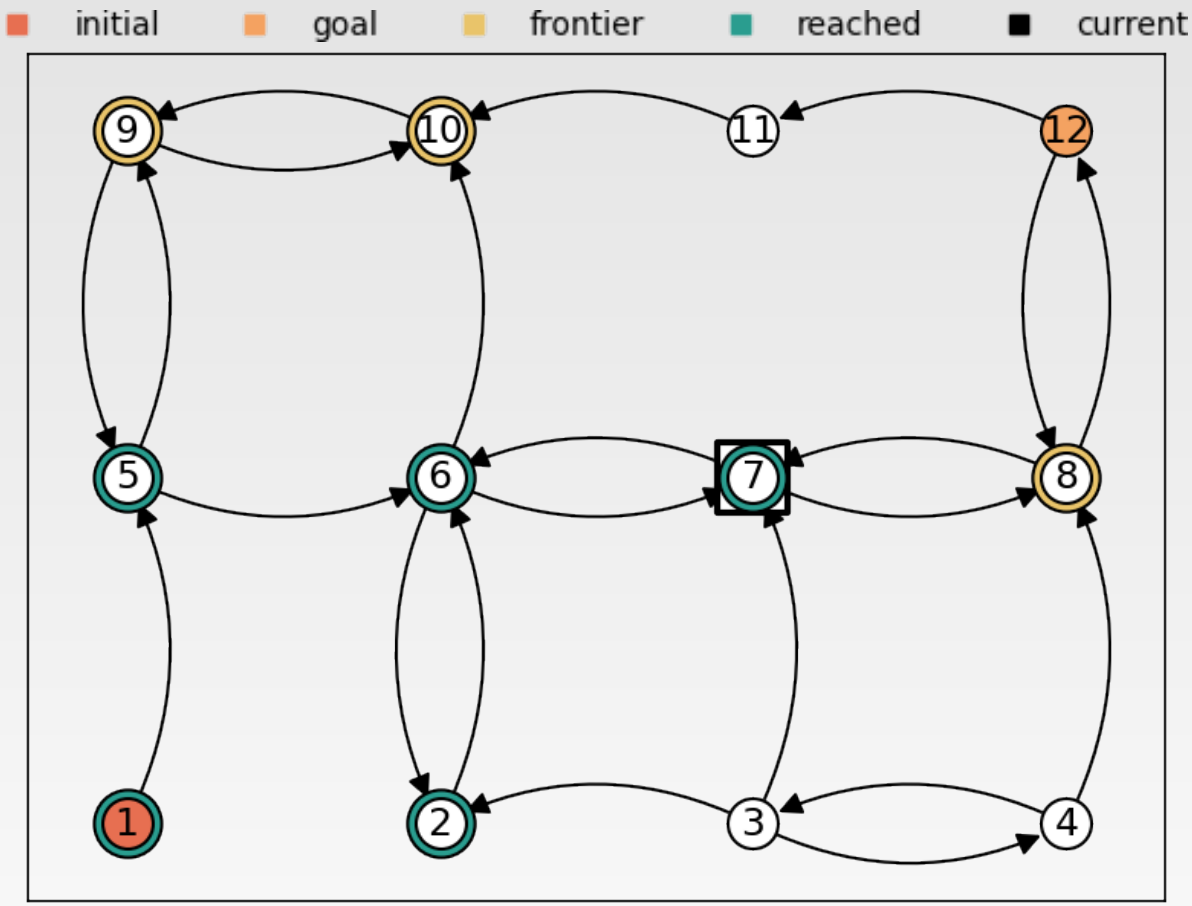
Iteration: 26, Current: 6
Frontier: [9, 10, 7, 2]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Iterative deepening



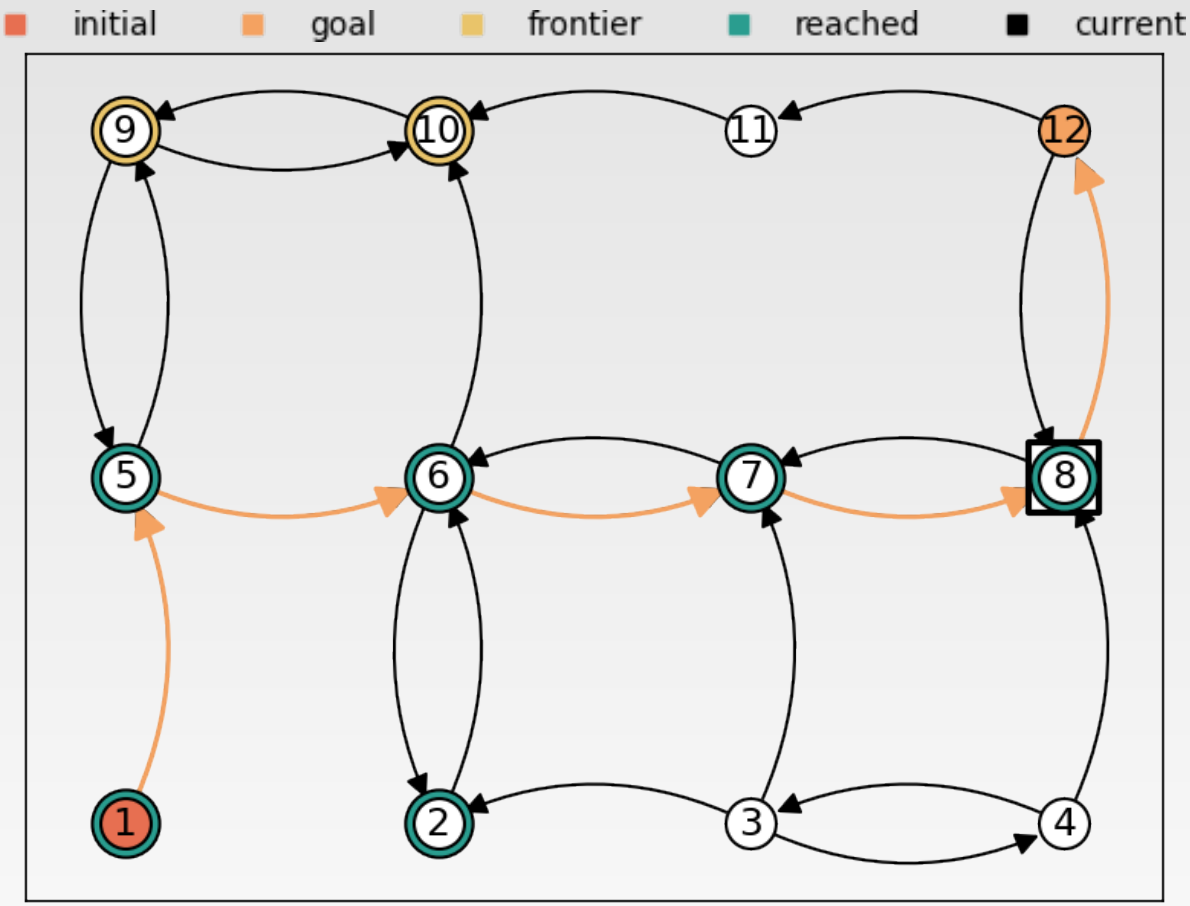
Iteration: 27, Current: 2
Frontier: [9, 10, 7]
Reached: {1, 2, 5, 6, 7, 9, 10}

Ex. 1 - Iterative deepening



Iteration: 28, Current: 7
Frontier: [9, 10, 8]
Reached: {1, 2, 5, 6, 7, 8, 9, 10}

Ex. 1 - Iterative deepening



Iteration: 29, Current: 8
Frontier: [9, 10]
Reached: {1, 2, 5, 6, 7, 8, 9, 10}

Ex. 1 - Iterative deepening

Done! *(For these problems, not worth it)*