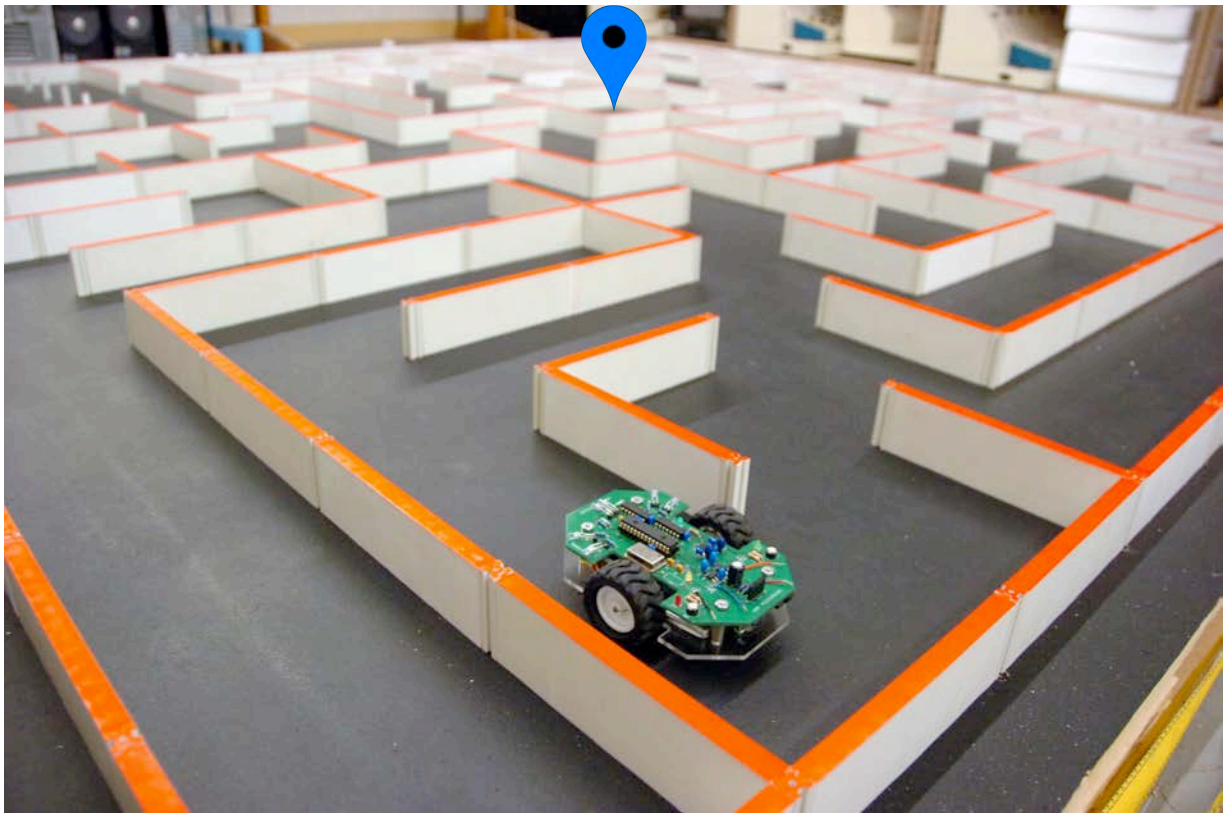
Autonomous Navigation: Global Search



https://app.emaze.com/@AIRRTROT/idea-2-the-robot-maze#1

Robotics 102

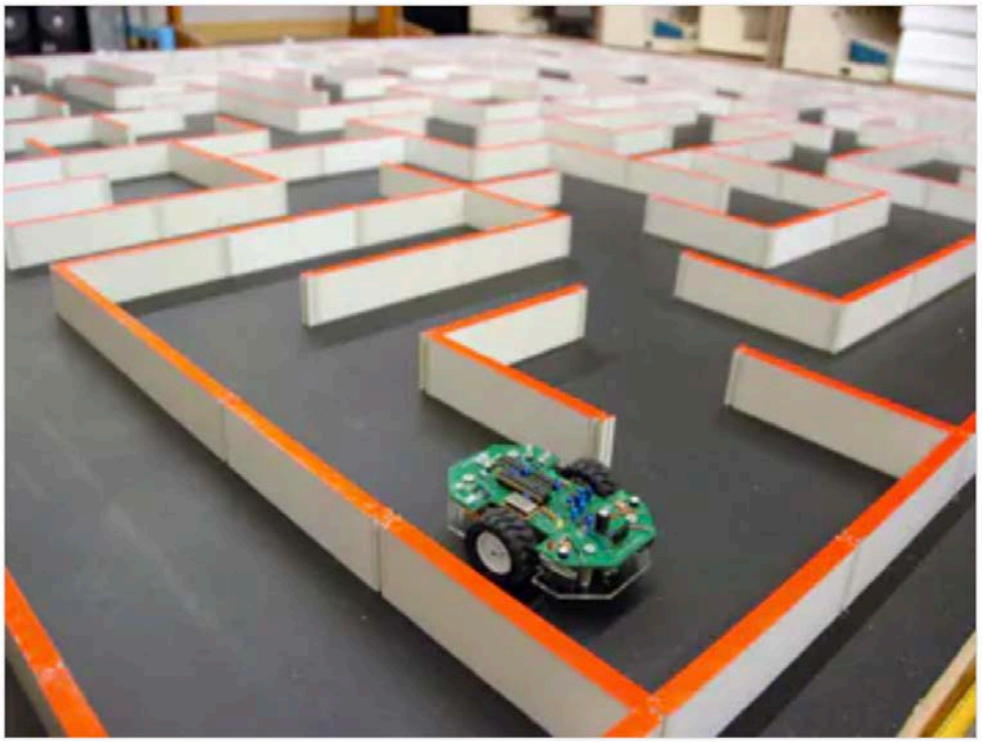
Introduction to AI and Programming
University of Michigan and Berea College
Fall 2021

Jordan Wong
Published June 19, 2015

IEEE Micromouse

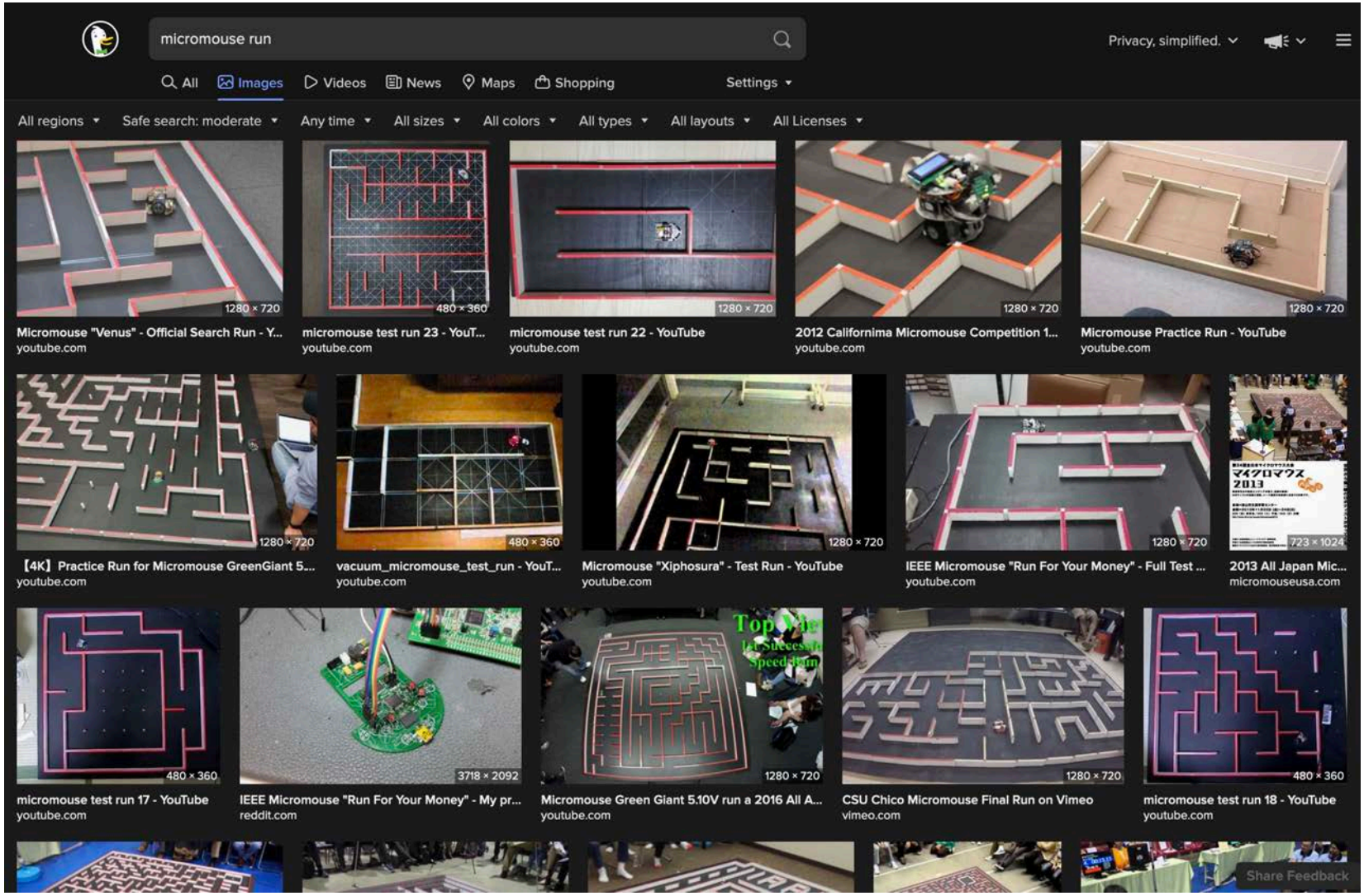
Autonomous mouse that can traverse any 16 by 16 maze.

Intermediate Full instructions provided 1,762



https://www.hackster.io/jordanjameswong/micromouse-83dab7

2011 All Japan micromouse contest: Ng BengKiat 4th Fast RUN
https://www.youtube.com/watch?v=CLwICJKV4dw


micromouse run
Privacy, simplified.
All
Images
Videos
News
Maps
Shopping
Settings
All regions
Safe search: moderate
Any time
All sizes
All colors
All types
All layouts
All Licenses
1280 × 720
Micromouse "Venus" - Official Search Run - Y...
youtube.com
480 × 360
micromouse test run 23 - YouT...
youtube.com
1280 × 720
micromouse test run 22 - YouTube
youtube.com
1280 × 720
2012 Californima Micromouse Competition 1...
youtube.com
1280 × 720
Micromouse Practice Run - YouTube
youtube.com
1280 × 720
【4K】Practice Run for Micromouse GreenGiant 5...
youtube.com
480 × 360
vacuum_micromouse_test_run - YouT...
youtube.com
1280 × 720
Micromouse "Xiphosura" - Test Run - YouTube
youtube.com
1280 × 720
IEEE Micromouse "Run For Your Money" - Full Test ...
youtube.com
マイクロマウス
2013
723 × 1024
2013 All Japan Mic...
micromouseusa.com
480 × 360
micromouse test run 17 - YouTube
youtube.com
3718 × 2092
IEEE Micromouse "Run For Your Money" - My pr...
reddit.com
1280 × 720
Micromouse Green Giant 5.10V run a 2016 All A...
youtube.com
1280 × 720
CSU Chico Micromouse Final Run on Vimeo
vimeo.com
480 × 360
micromouse test run 18 - YouTube
youtube.com
Share Feedback

Our goal

Our goal

Give you the power of autonomous navigation

Our goal

Give you the power of autonomous navigation

Our goal

Give you the power of autonomous navigation

Autonomous Navigation

Goal location

Start location

Autonomous Navigation
by global search

Goal location

Think of our robot's navigation as solving a maze

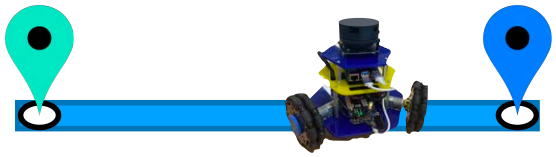
Start location

Autonomous Navigation
by global search

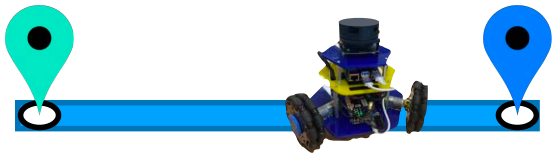
Think of our robot's navigation as solving a maze

Autonomous Navigation
by global search

Think of our robot's navigation as solving a maze

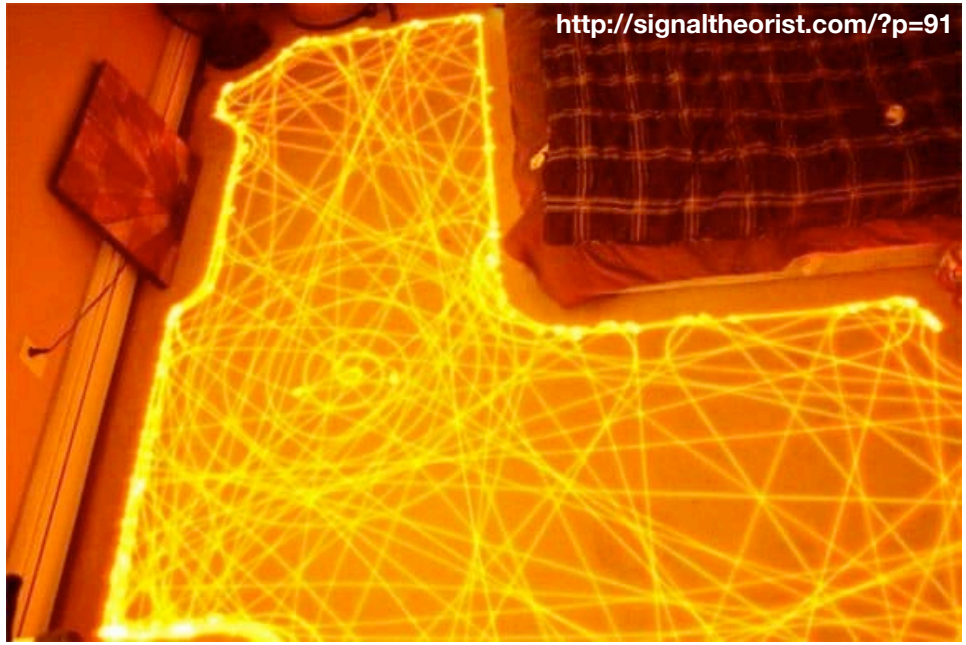


What options do we have for navigating our robot?

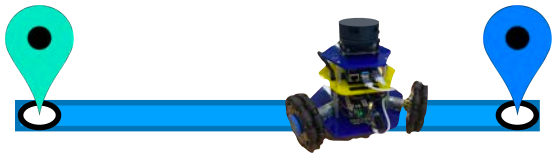


What options do we have for navigating our robot?

Just move randomly



Random walk algorithms

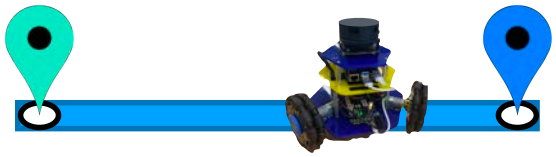


What options do we have for navigating our robot?

Just move randomly

Follow wall to goal

Bug algorithms



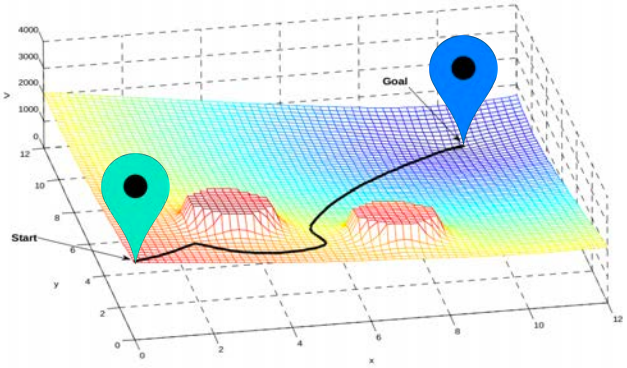
Just move randomly

Follow wall to goal

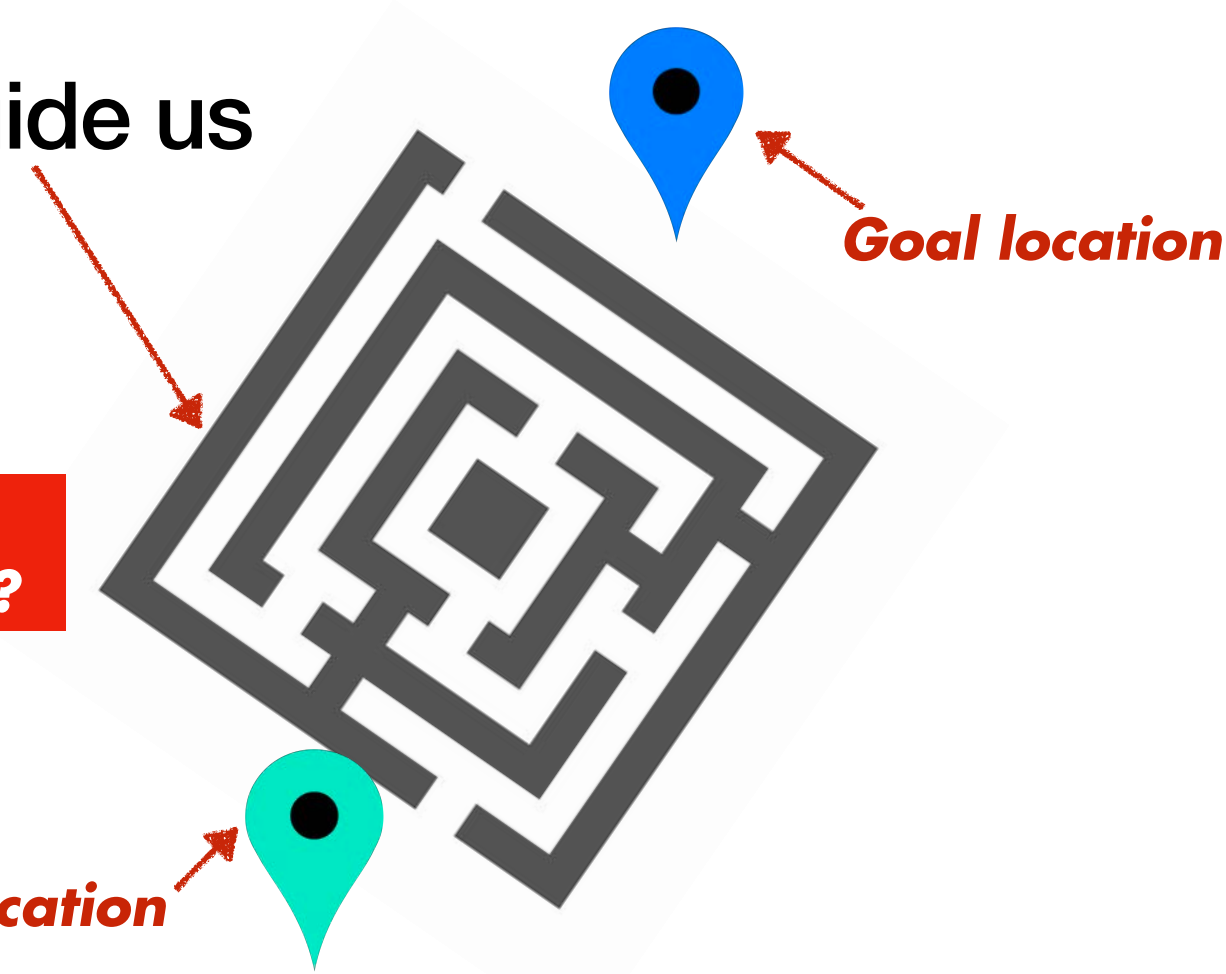
Build a map to guide us

Project 2: Potential Fields

Autonomous navigation to a goal location

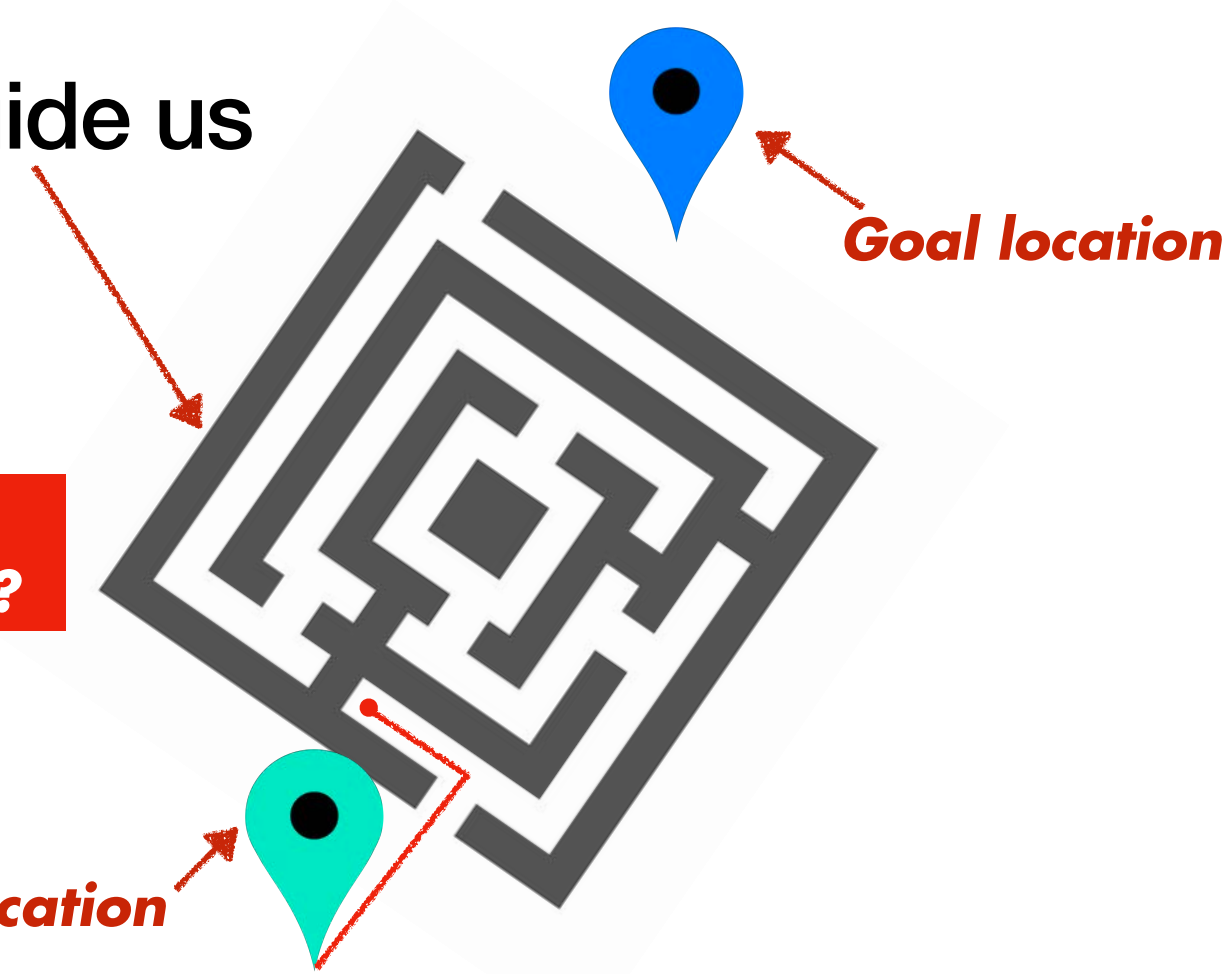


Build a map to guide us

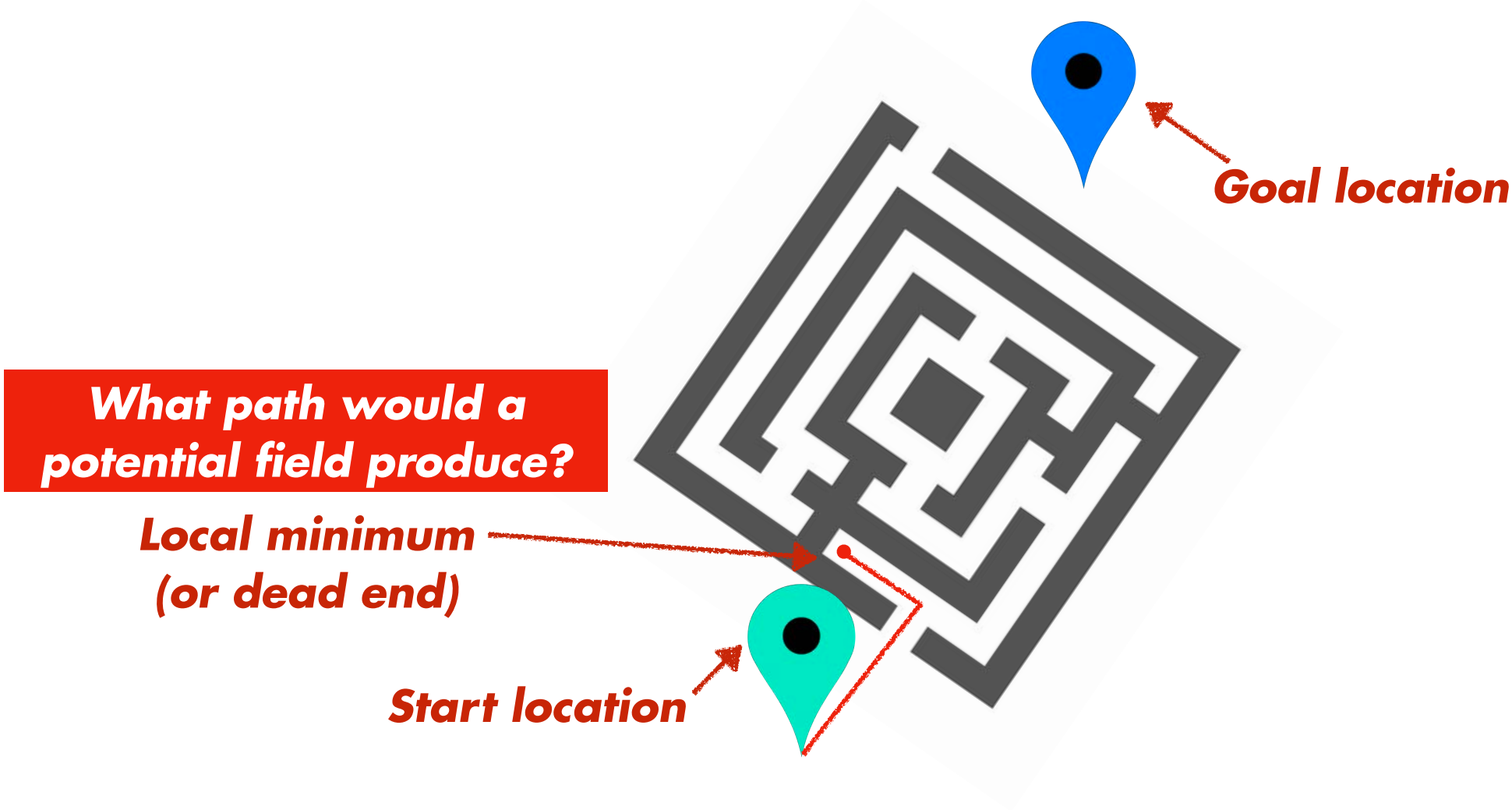


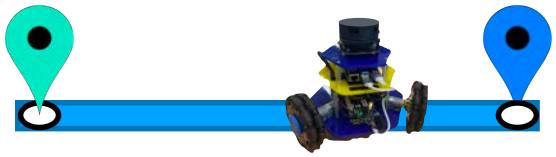
What path would a potential field produce?

Build a map to guide us



What path would a potential field produce?


Goal location
What path would a potential field produce?
Local minimum (or dead end)
Start location

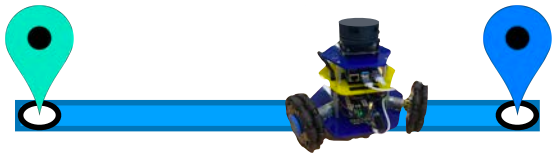


What options do we have for navigating our robot?

Just move randomly

Follow wall to goal

Build a map to guide us



Just move randomly

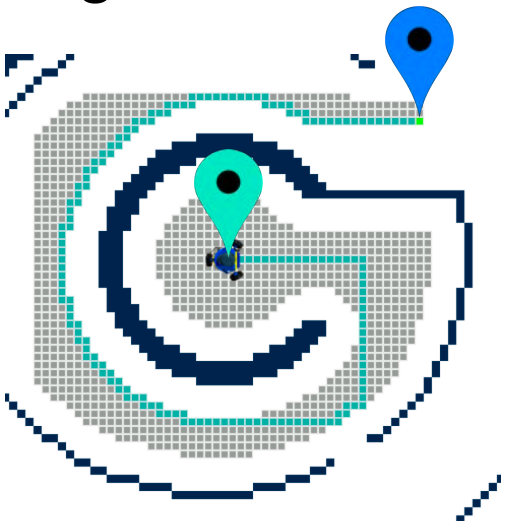
Follow wall to goal

Build a map to guide us

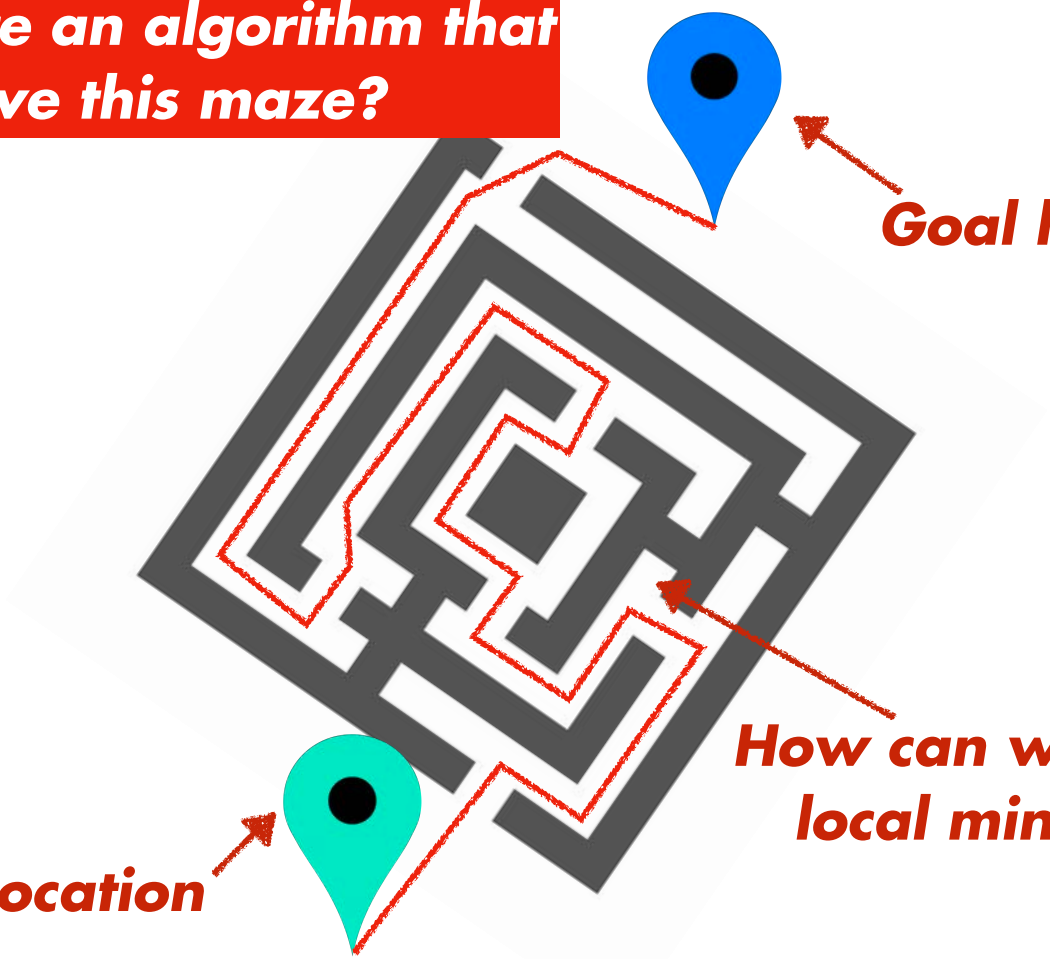
Consider all possible paths

Project 3: A* Pathfinding

Autonomous navigation to a goal location

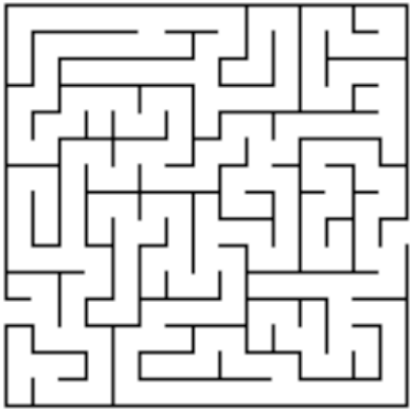


Can we create an algorithm that will solve this maze?



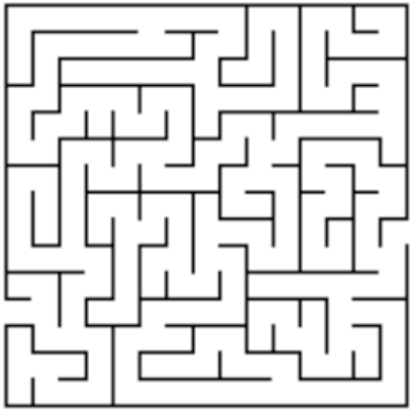
Can we create an algorithm that will solve this maze?

and this one?

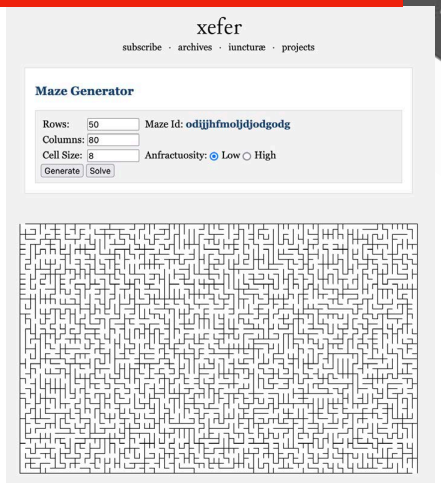


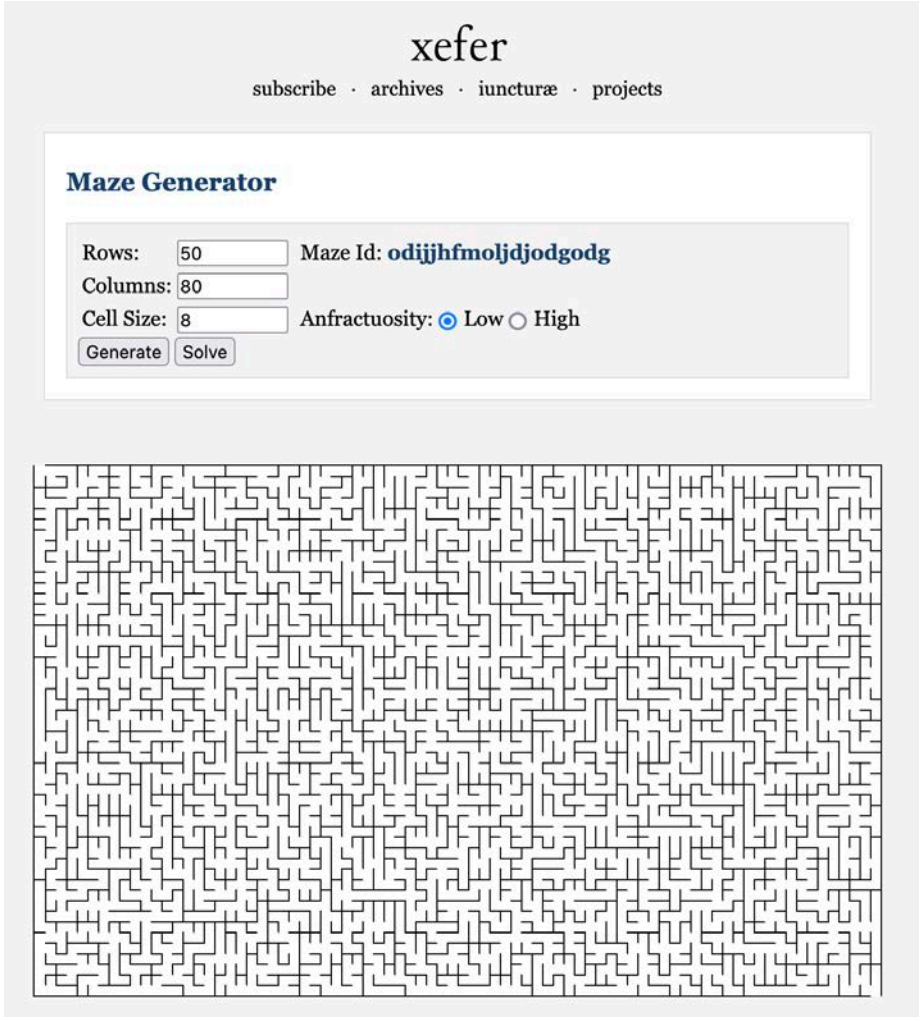
Can we create an algorithm that will solve this maze?

and this one?



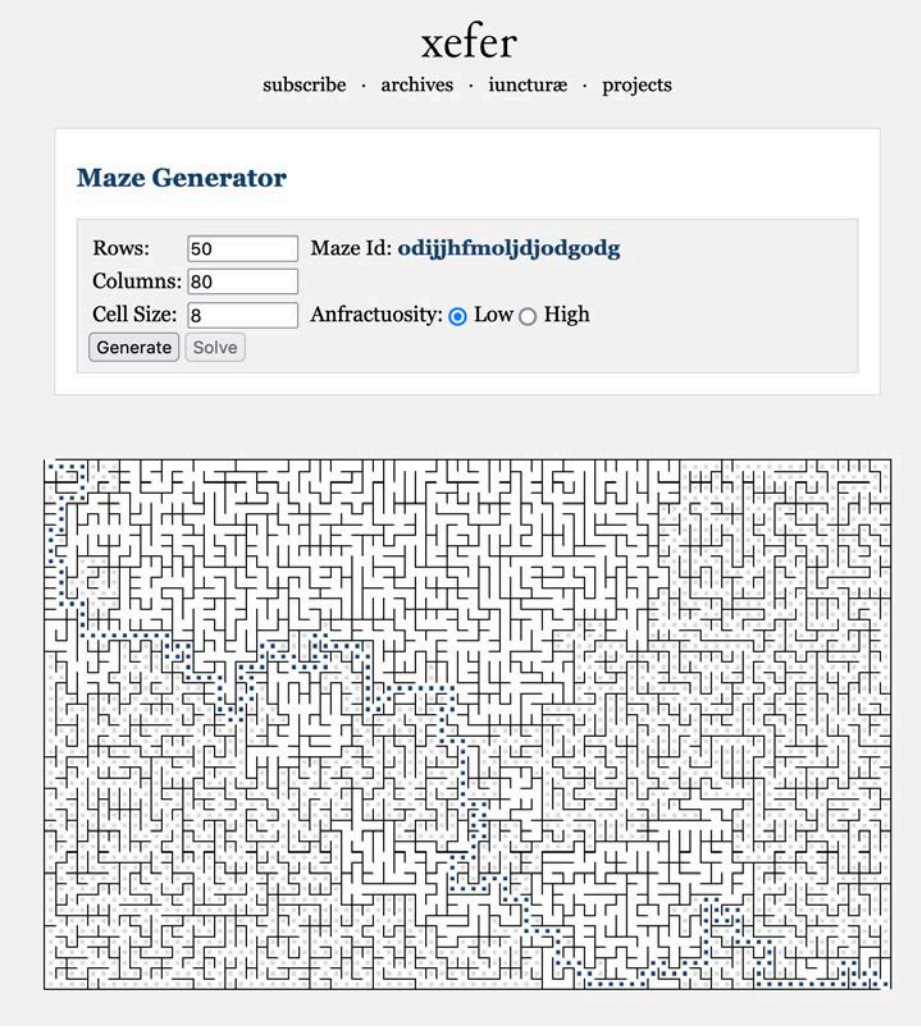
and this one?





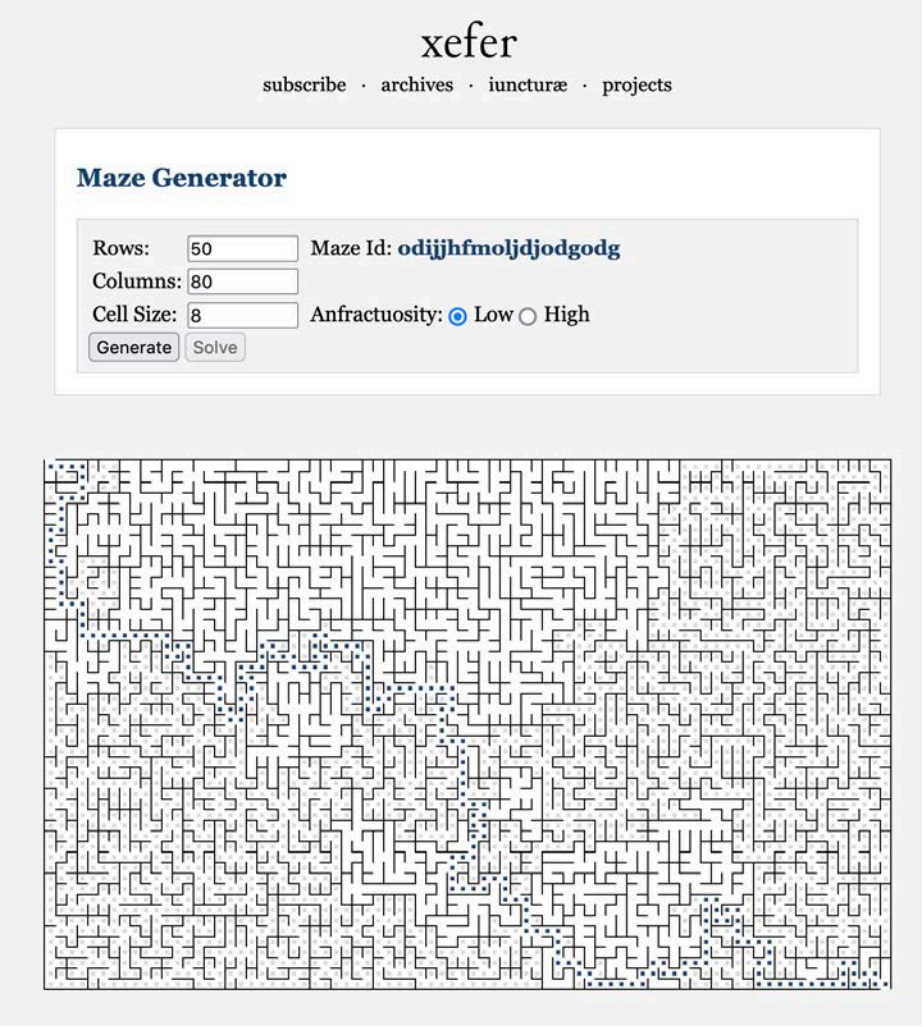
https://www.xefer.com/maze-generator

How does this algorithm work?



https://www.xefer.com/maze-generator

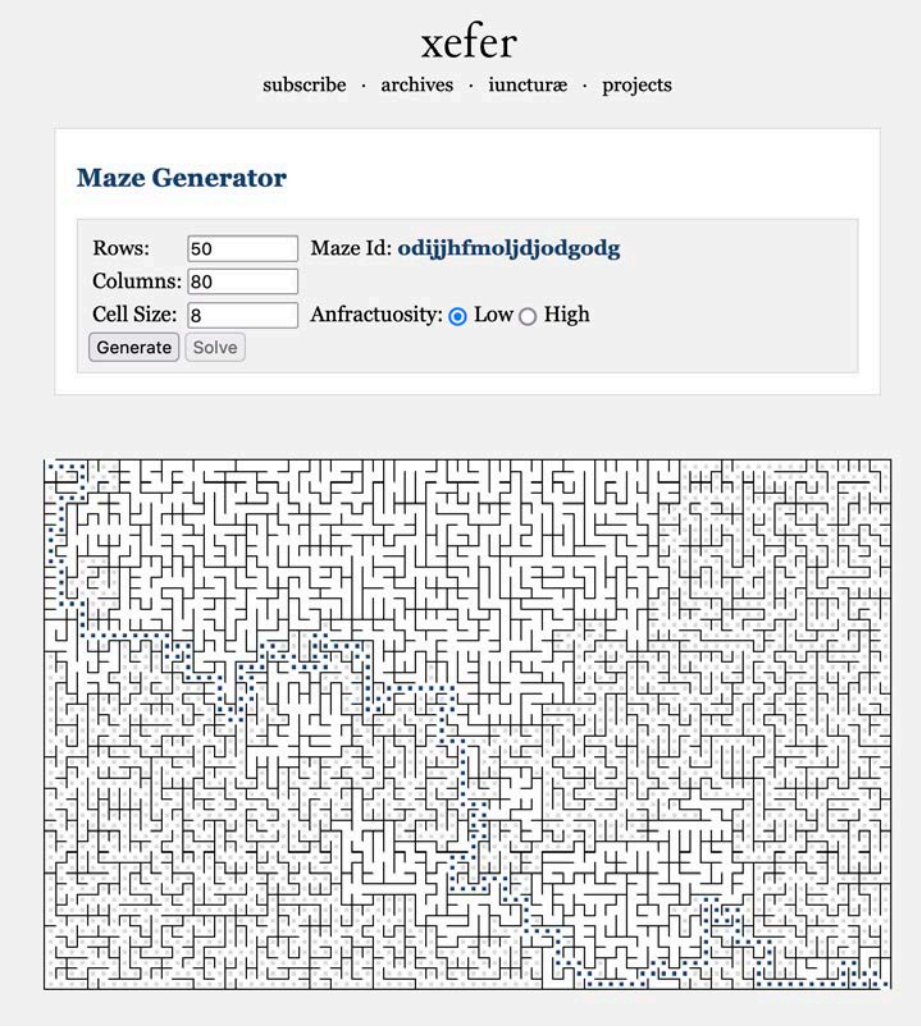
How does this algorithm work?



https://www.xefer.com/maze-generator

Represent the map as a graph

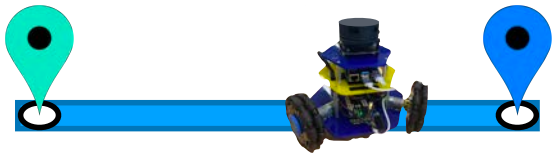
How does this algorithm work?



https://www.xefer.com/maze-generator

Represent the map as a graph

Search over all possible paths



Just move randomly

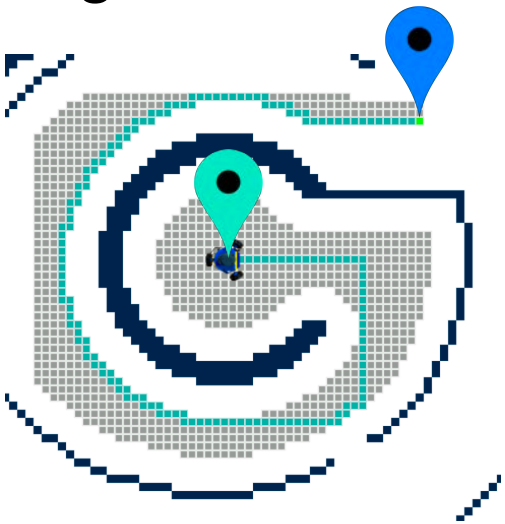
Follow wall to goal

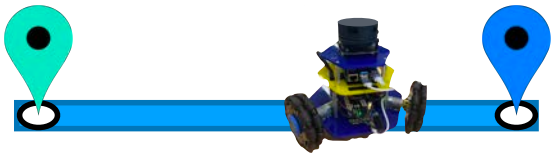
Build a map to guide us

Consic**Search over all possible paths**le paths

Project 3: A* Pathfinding

Autonomous navigation to a goal location





Just move randomly

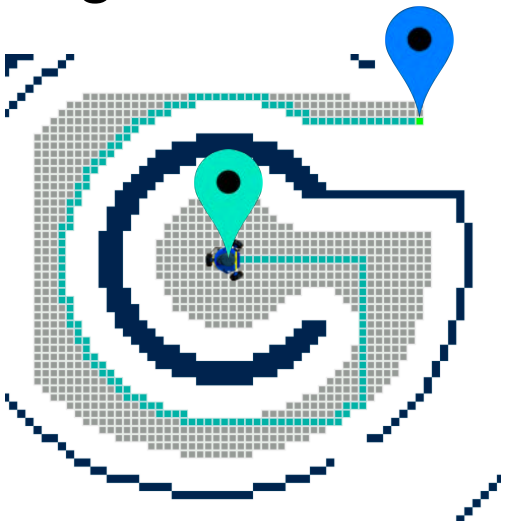
Follow wall to goal

Build a map to guide us

Search over all possible paths

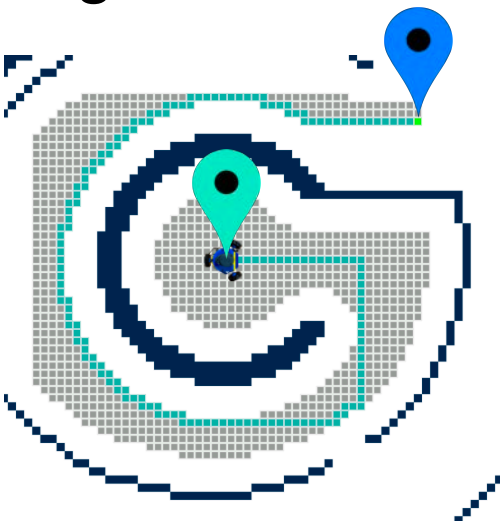
Project 3: A* Pathfinding

Autonomous navigation to a goal location



Project 3:
A*
Pathfinding

Autonomous navigation to a goal location

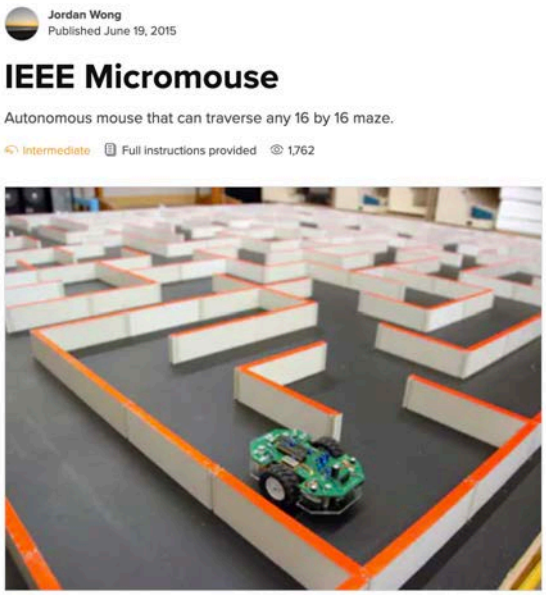


Search over all possible paths

Already done from Project 2

- [x] Build map of environment
- [x] Represent map as graph with a grid layout
- [ ] Store parent of each node along route to start location
- [ ] Store path distance at each node along route to start location
- [ ] Global search to find routing

Micromouse represents maze as a grid graph then performs “Floodfill” to find path to goal



https://medium.com/@austinxiao/ieee-micromouse-2016-software-design-496653ff104d

Micromouse represents maze as a grid graph then performs “Floodfill” to find path to goal

Result from pathfinding:

https://medium.com/@austinxiao/ieee-micromouse-2016-software-design-496653ff104d

Micromouse represents maze as a grid graph then performs “Floodfill” to find path to goal

Result from pathfinding:

Provides cell-to-cell routing along found path

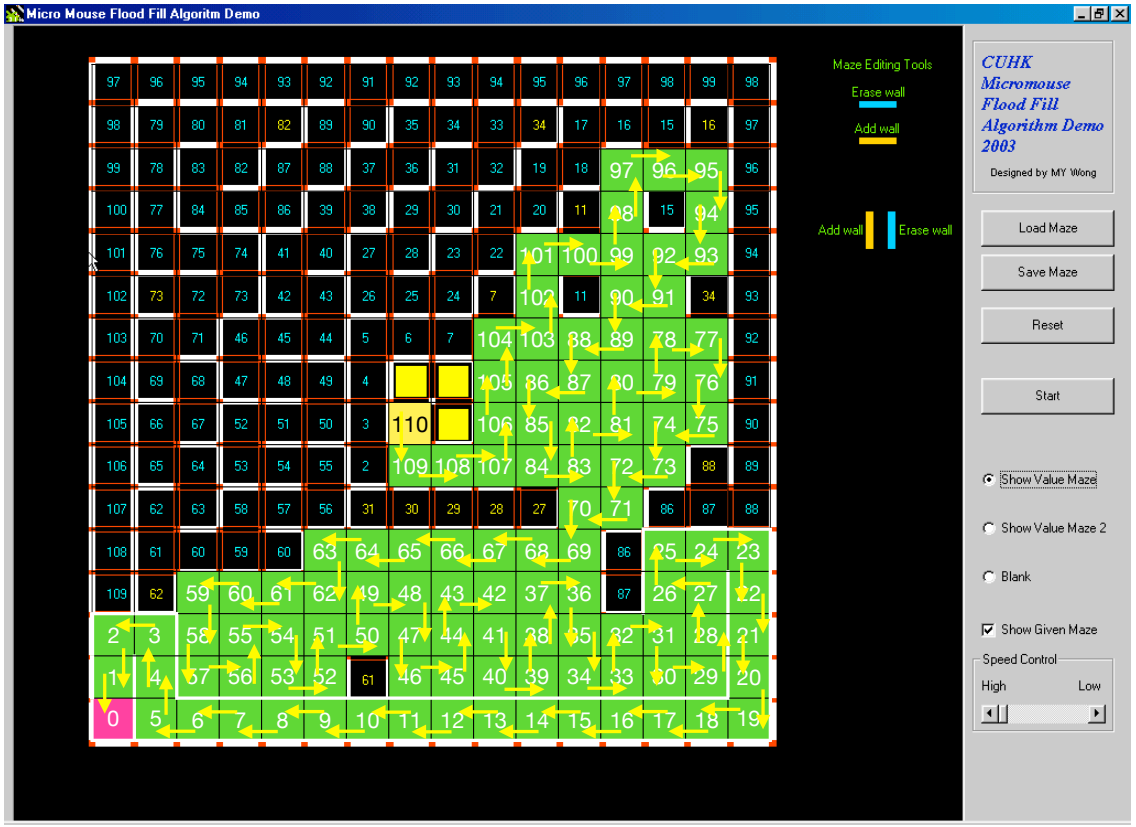
https://medium.com/@austinxiao/ieee-micromouse-2016-software-design-496653ff104d

Micromouse represents maze as a grid graph then performs “Floodfill” to find path to goal

Result from pathfinding:

Provides cell-to-cell routing along found path

Distance along path at each cell



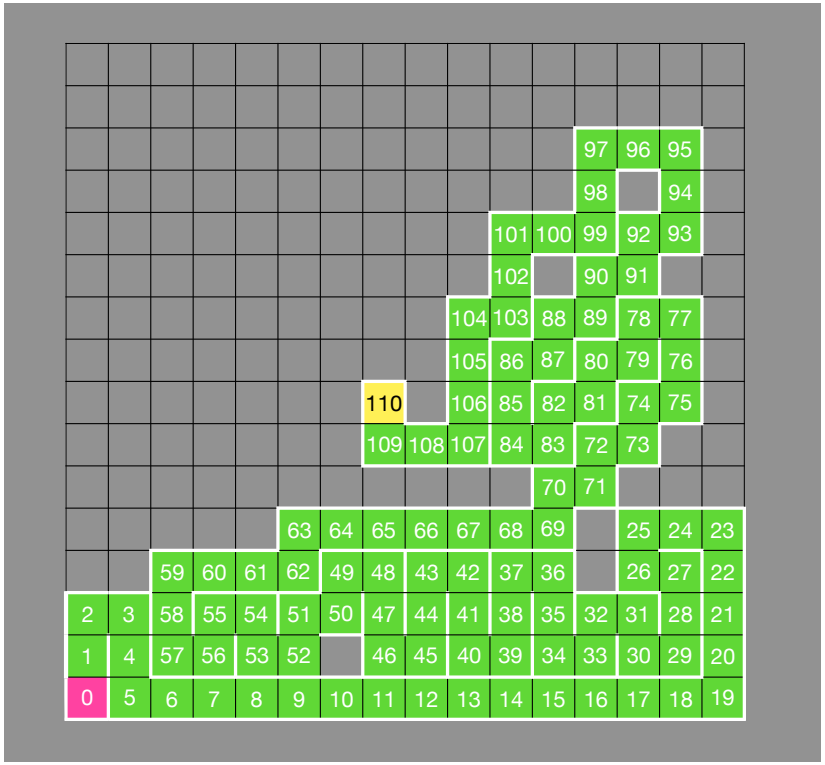
https://medium.com/@austinxiao/ieee-micromouse-2016-software-design-496653ff104d

Micromouse represents maze as a grid graph then performs “Floodfill” to find path to goal

Result from pathfinding:

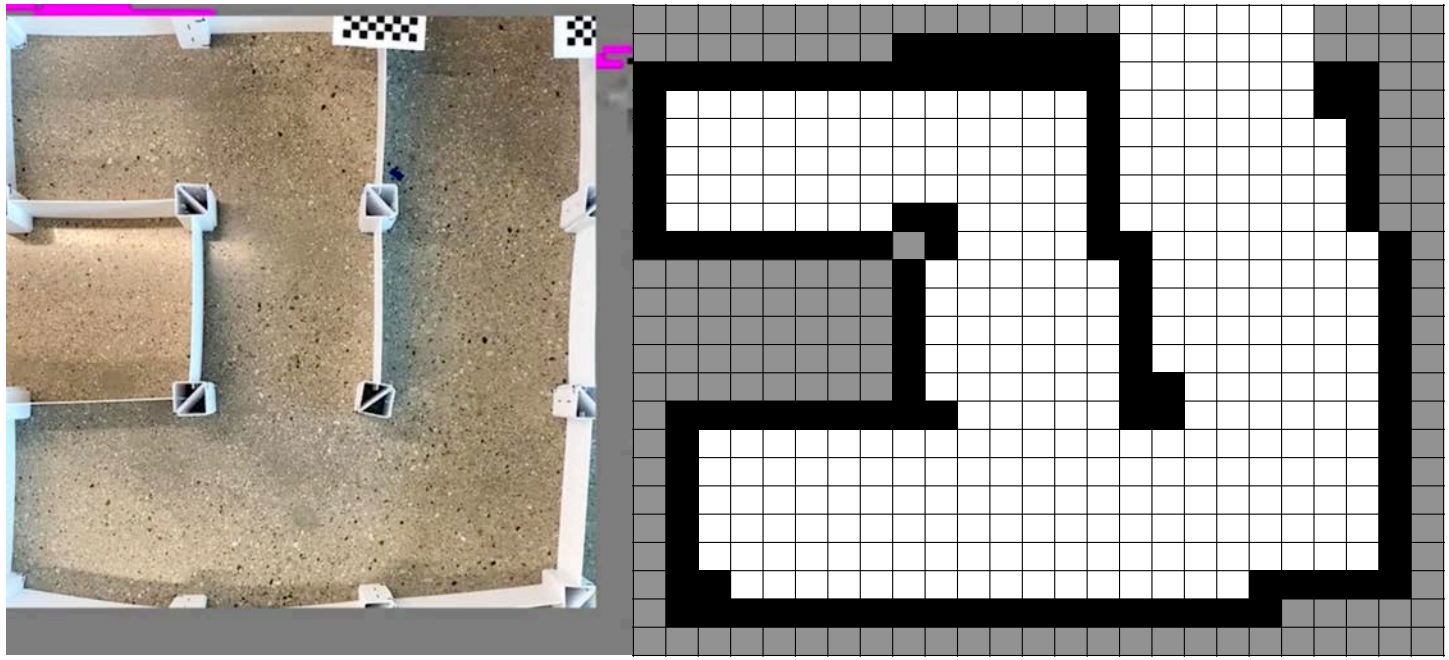
Provides cell-to-cell routing along found path

Distance along path at each cell



Remember what this graph looks like for our robot maps

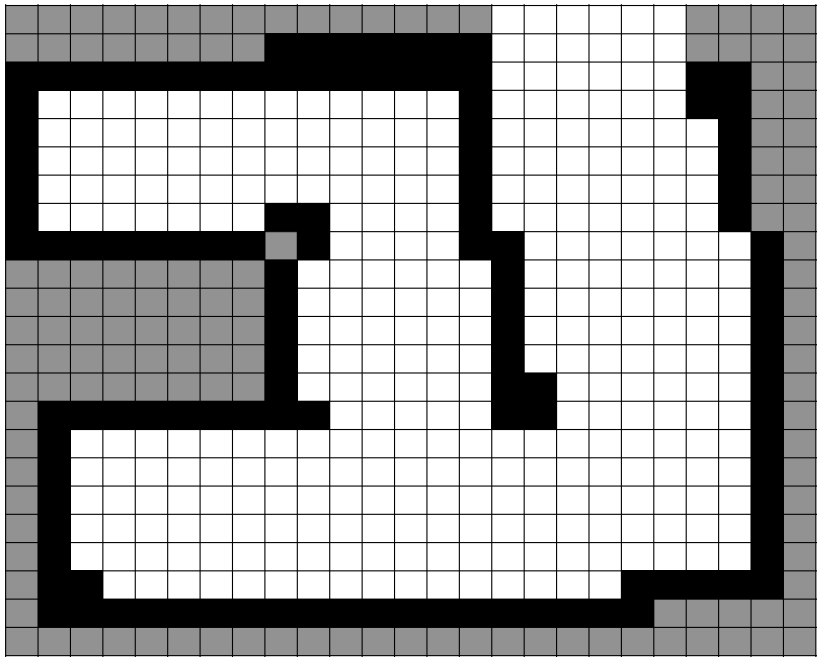
Robot map is stored as an image and represented as a graph



Real world *SLAM output*

Robot map is stored as an image and represented as a graph

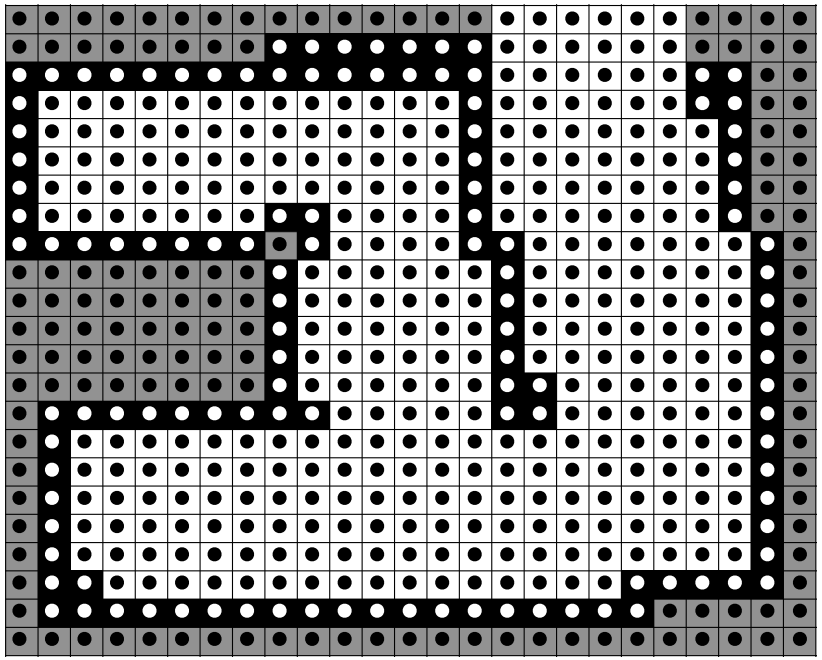
A vector of cells over robot locations



Robot map is stored as an image and represented as a graph

A vector of cells over robot locations

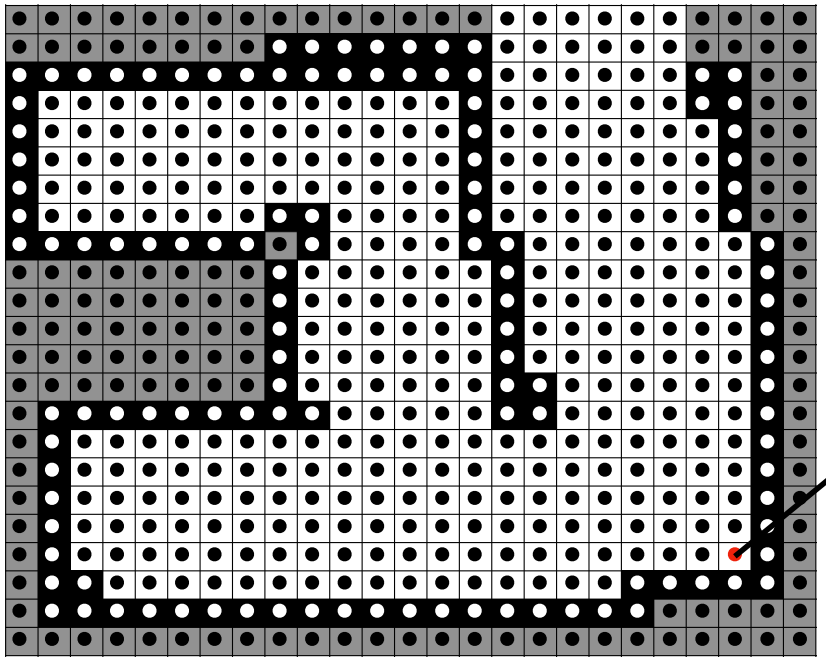
Every cell has a node in the graph



Robot map is stored as an image and represented as a graph

A vector of cells over robot locations

Every cell has a node in the graph



```
origin_x: 2.2
origin_y: 0.3
occupied: false
```

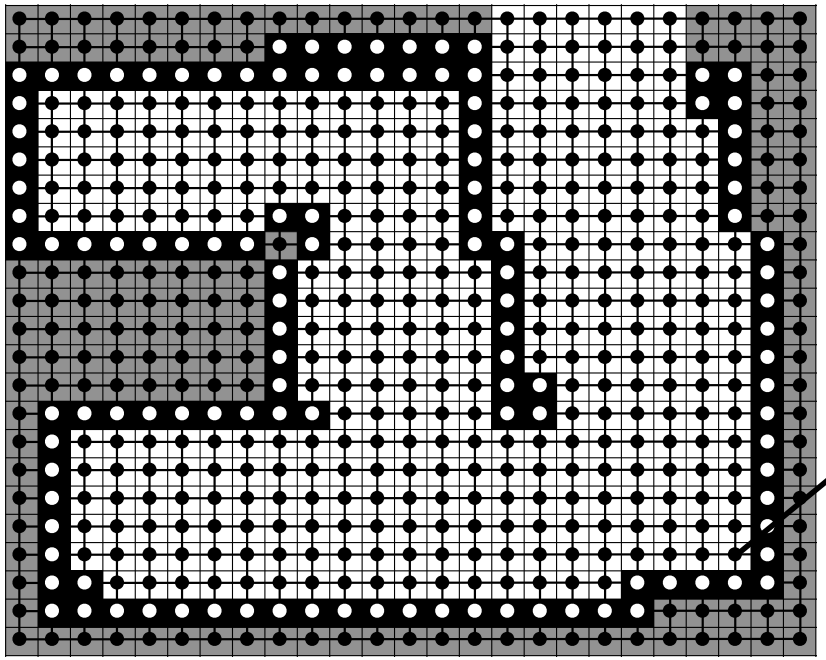
A graph node stores a struct of information about the cell

Robot map is stored as an image and represented as a graph

A vector of cells over robot locations

Every cell has a node in the graph

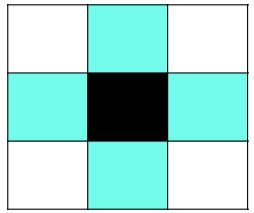
Every pair of neighboring cells shares an edge in the graph

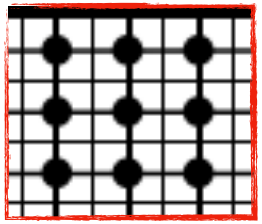


```
origin_x: 2.2
origin_y: 0.3
occupied: false
```

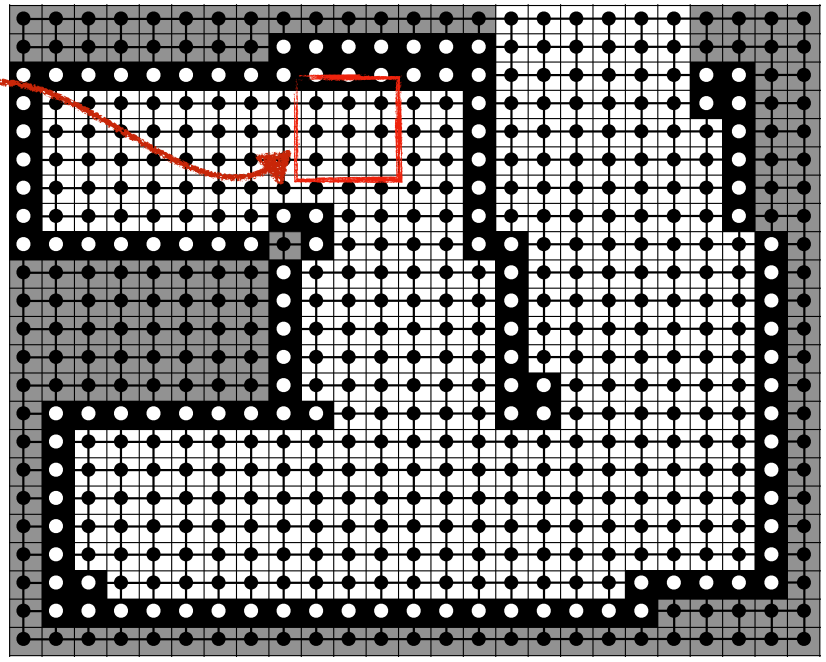
A graph node stores a struct of information about the cell

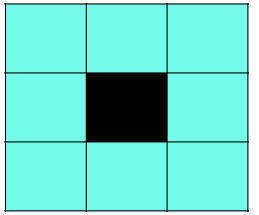
Grid graphs are typically either 4-connected or 8-connected

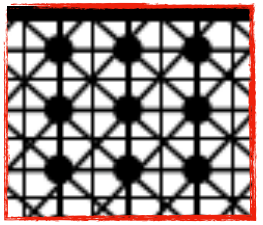




"NSEW"

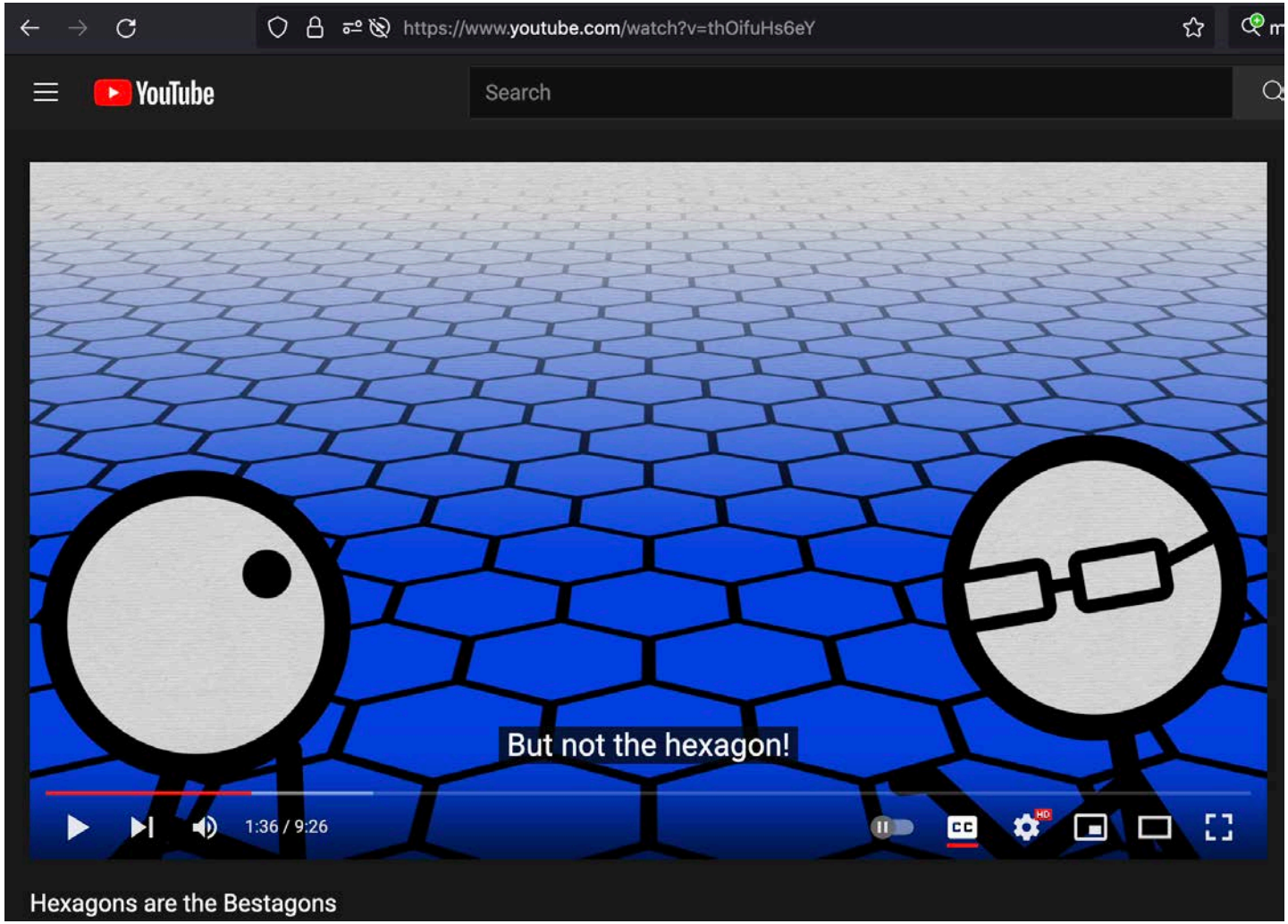






Diagonals included

Quick tangent: *Why are hexagons the bestagons ?*



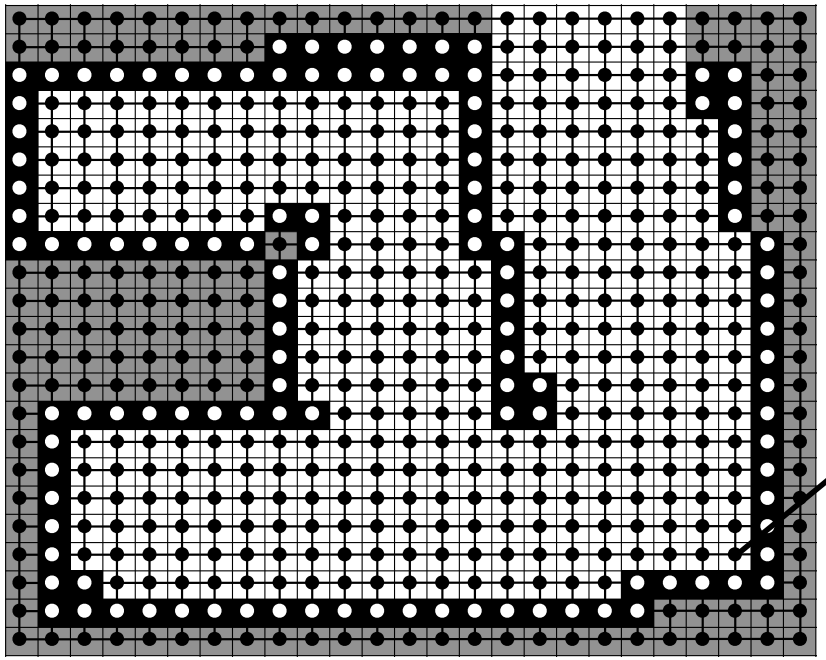
- [ ] Store parent of each node along route to start location
- [ ] Store path distance at each node along route to start location

Path expressed as the route to navigate at a node

A vector of cells over robot locations

Every cell has a node in the graph

Every pair of neighboring cells shares an edge in the graph

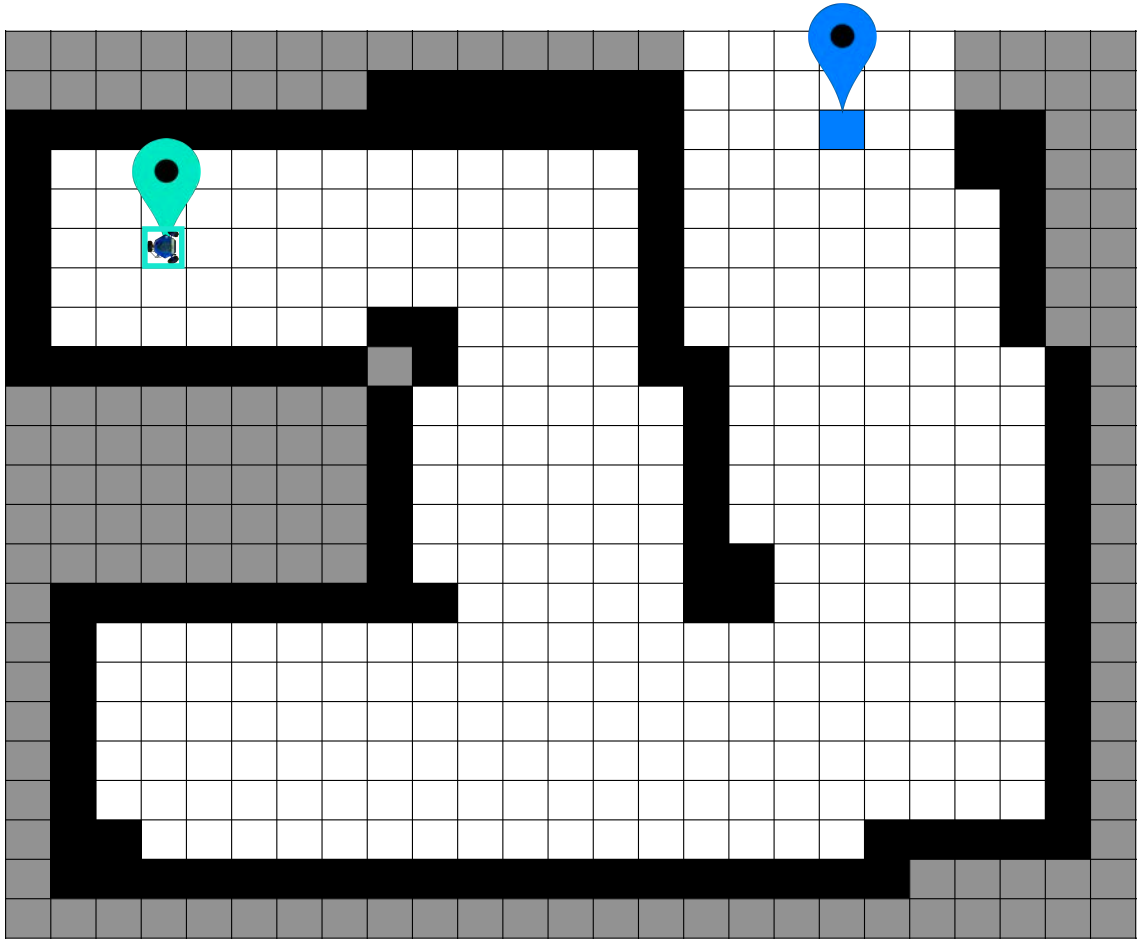


```
origin_x: 2.2
origin_y: 0.3
occupied: false
parent: ??
distance: ??
```

A graph node stores a struct of information about the cell

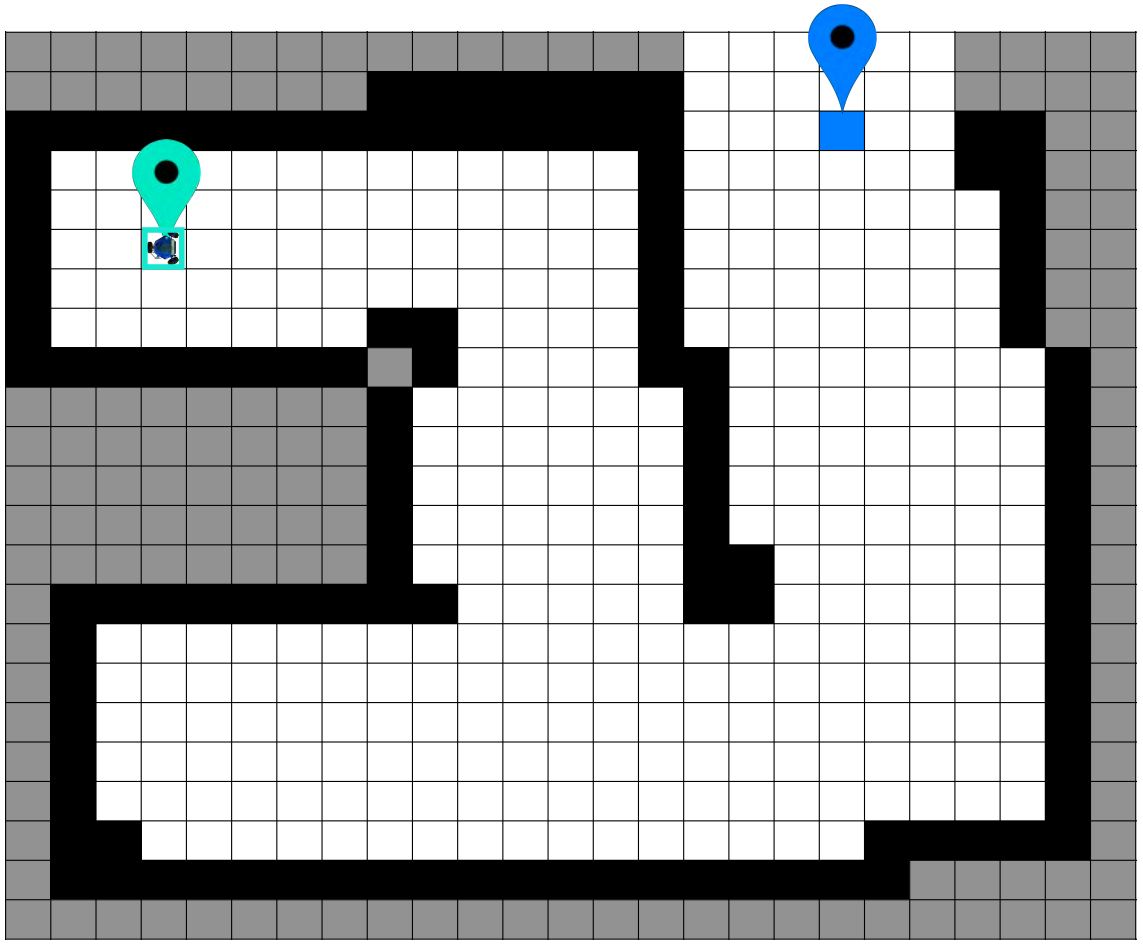
Assume robot planning a route from a given start location to a given goal location

How does Floodfill compute `.parent` *and* `.distance` *for each node?*



Assume robot planning a route from a given start location to a given goal location

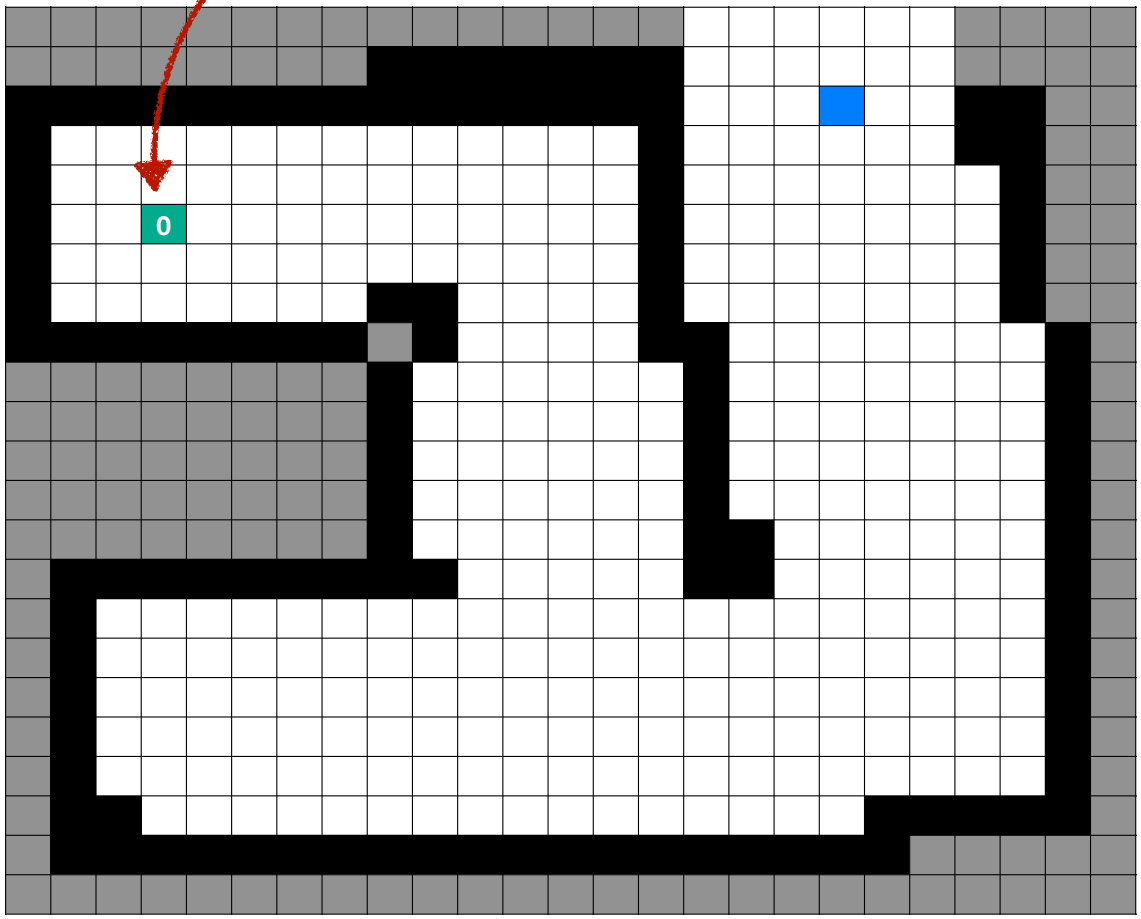
How does Floodfill compute `.parent` ***and*** `.distance` ***for each node?***



Grow outward from the node at the start location

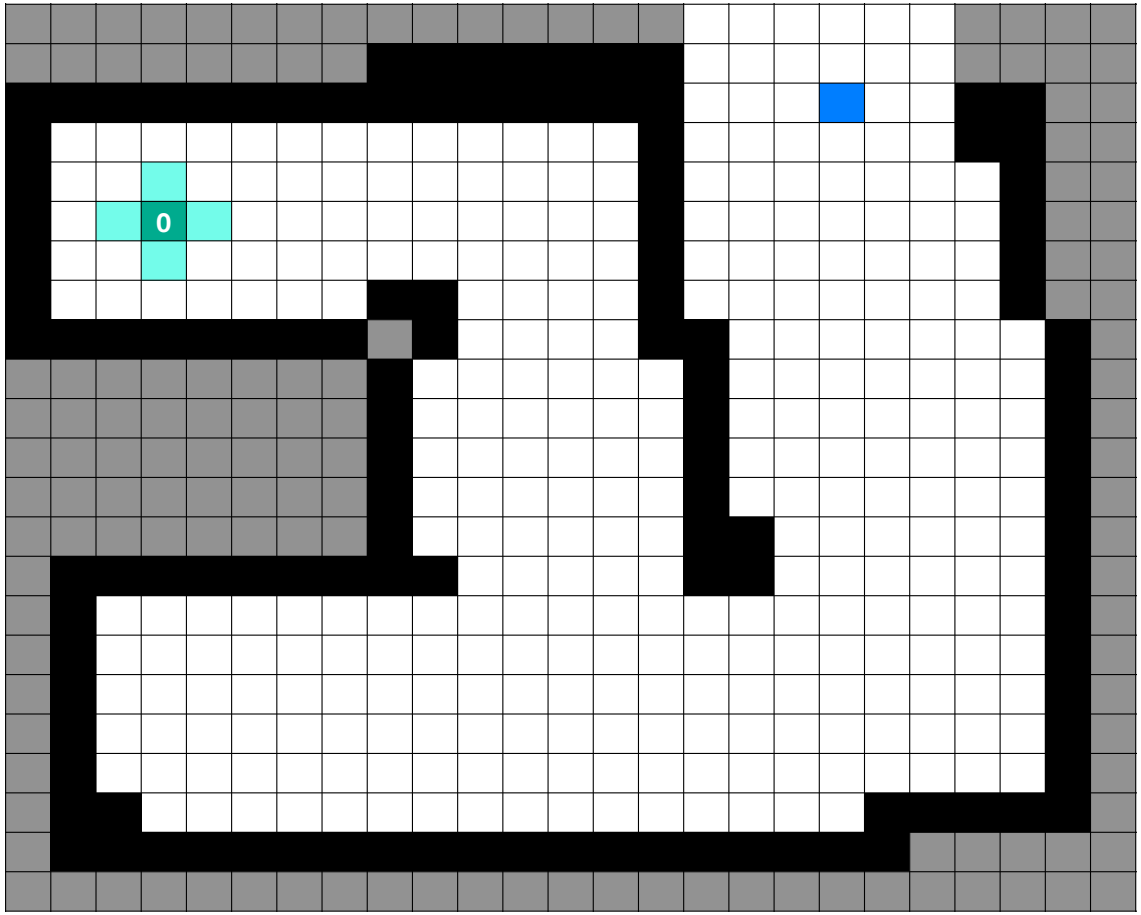
Begin from start node with no parent and zero distance

The start is the "root" of our search graph



Begin from start node with no parent and zero distance

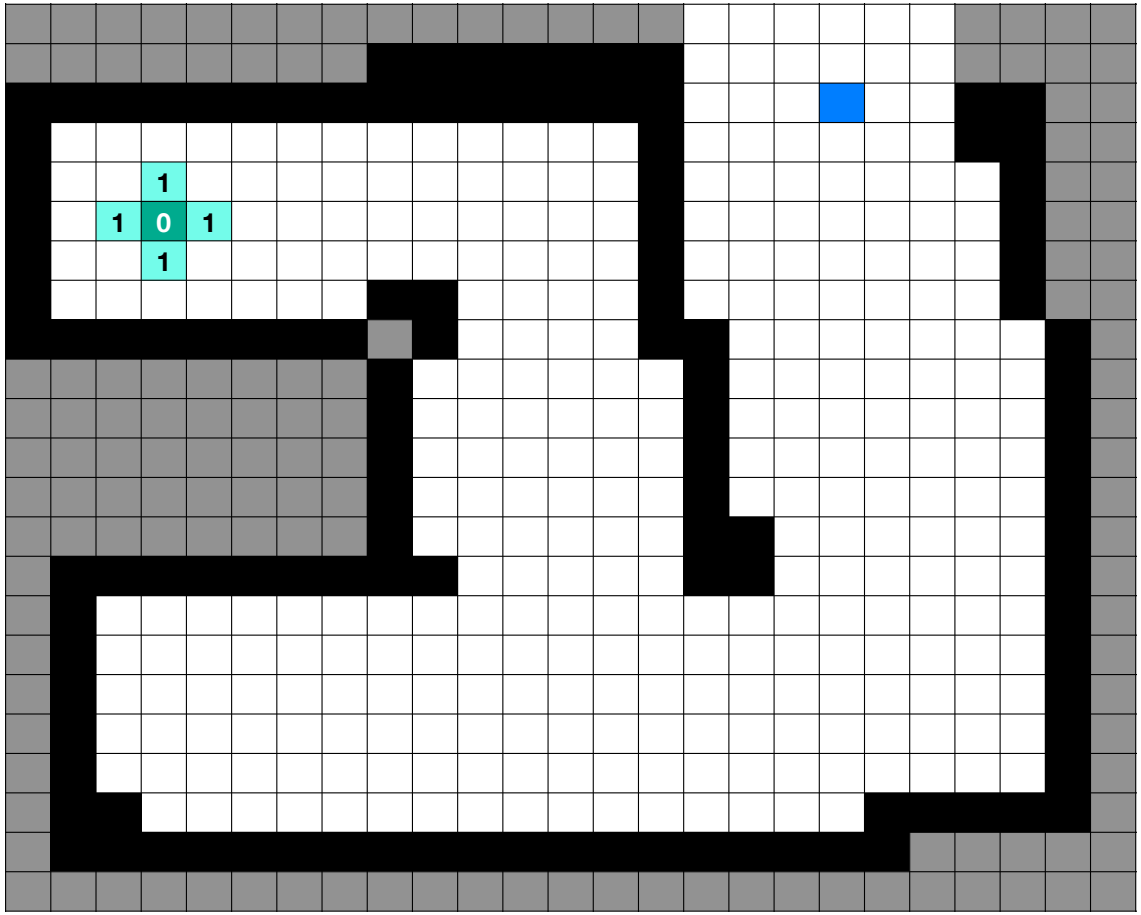
All neighbors of start are "visited"



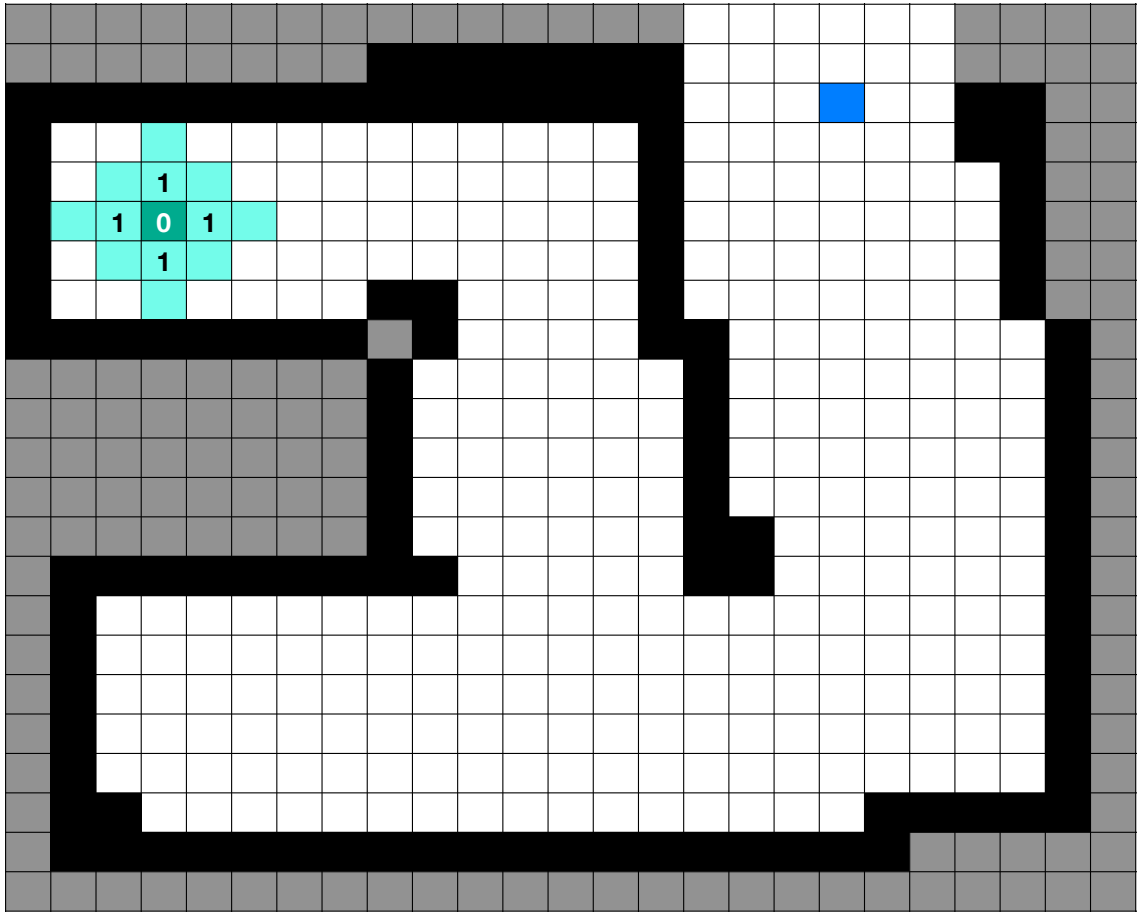
Begin from start node with no parent and zero distance

All neighbors of start are "visited"

And assigned one plus the smallest distance

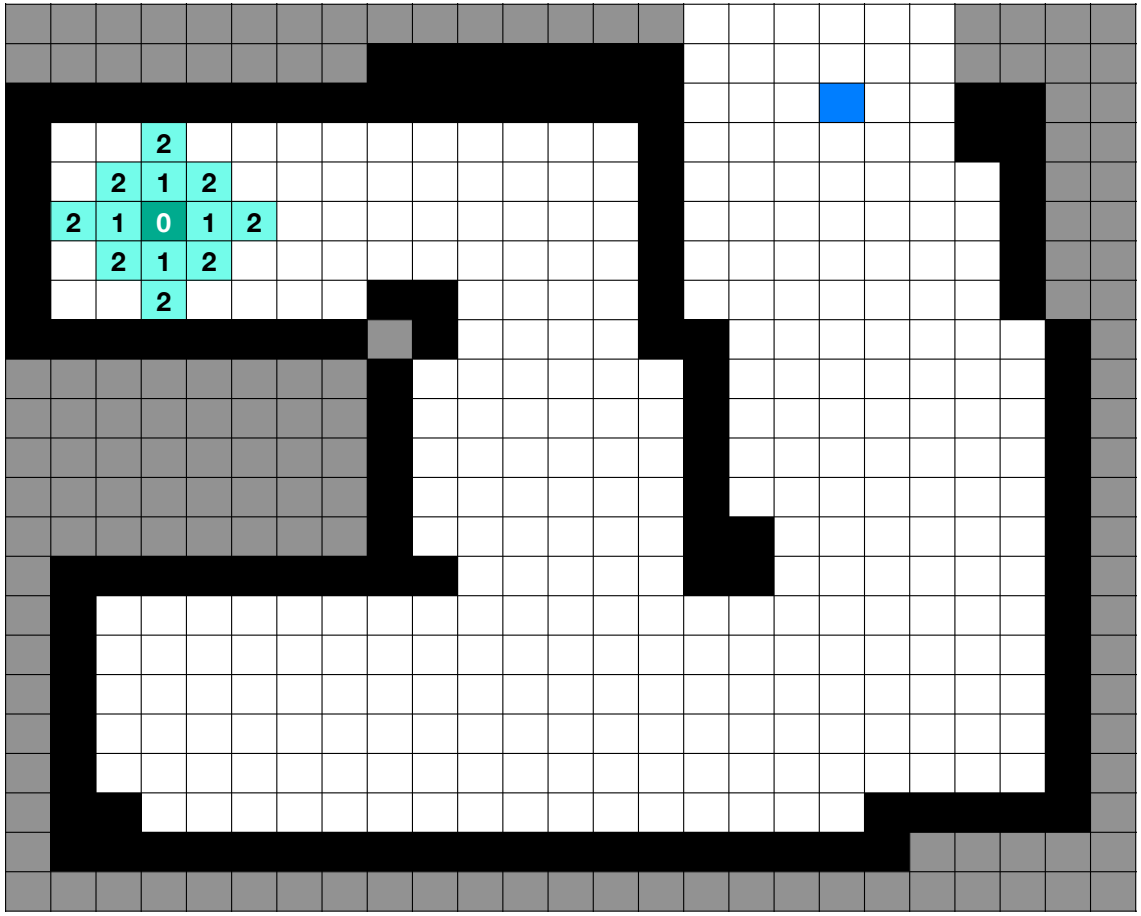


Then, visit the neighbors of nodes just visited



Then, visit the neighbors of nodes just visited

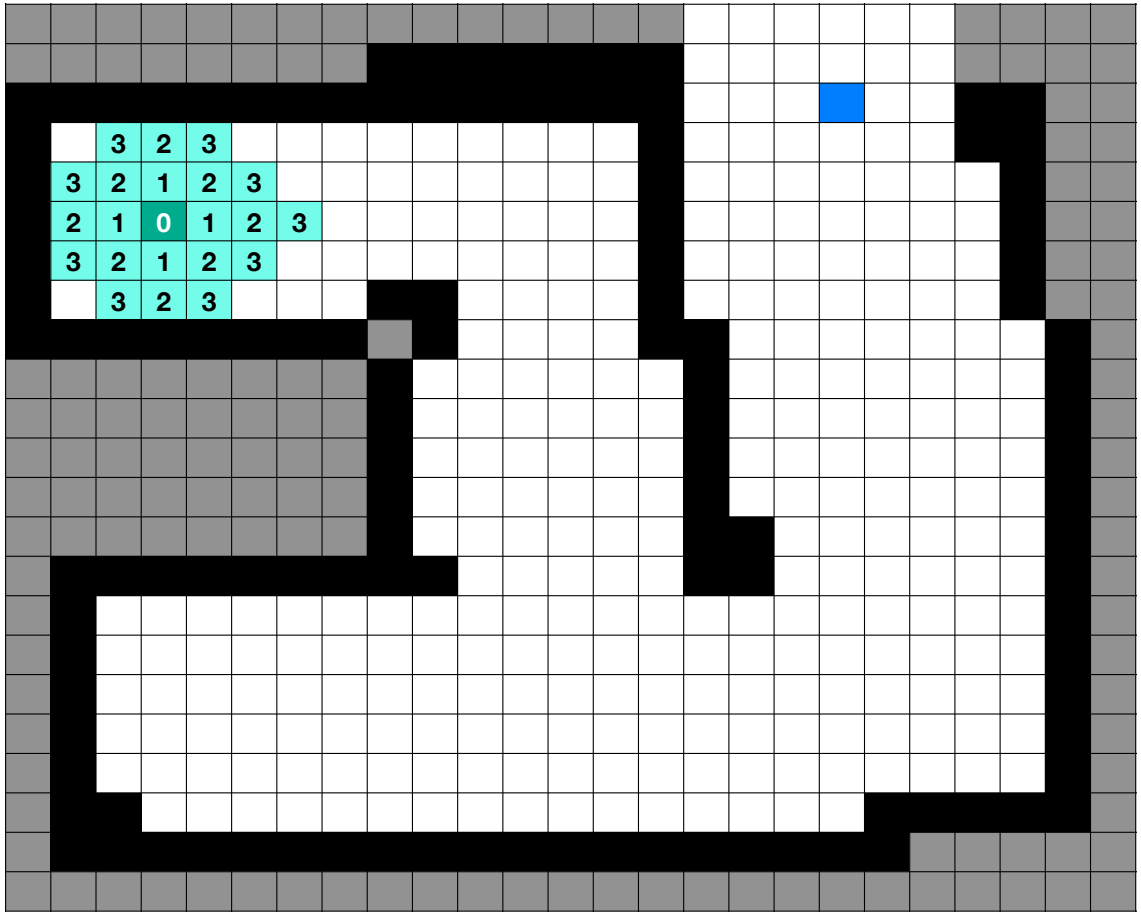
Assign each one plus the smallest distance



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

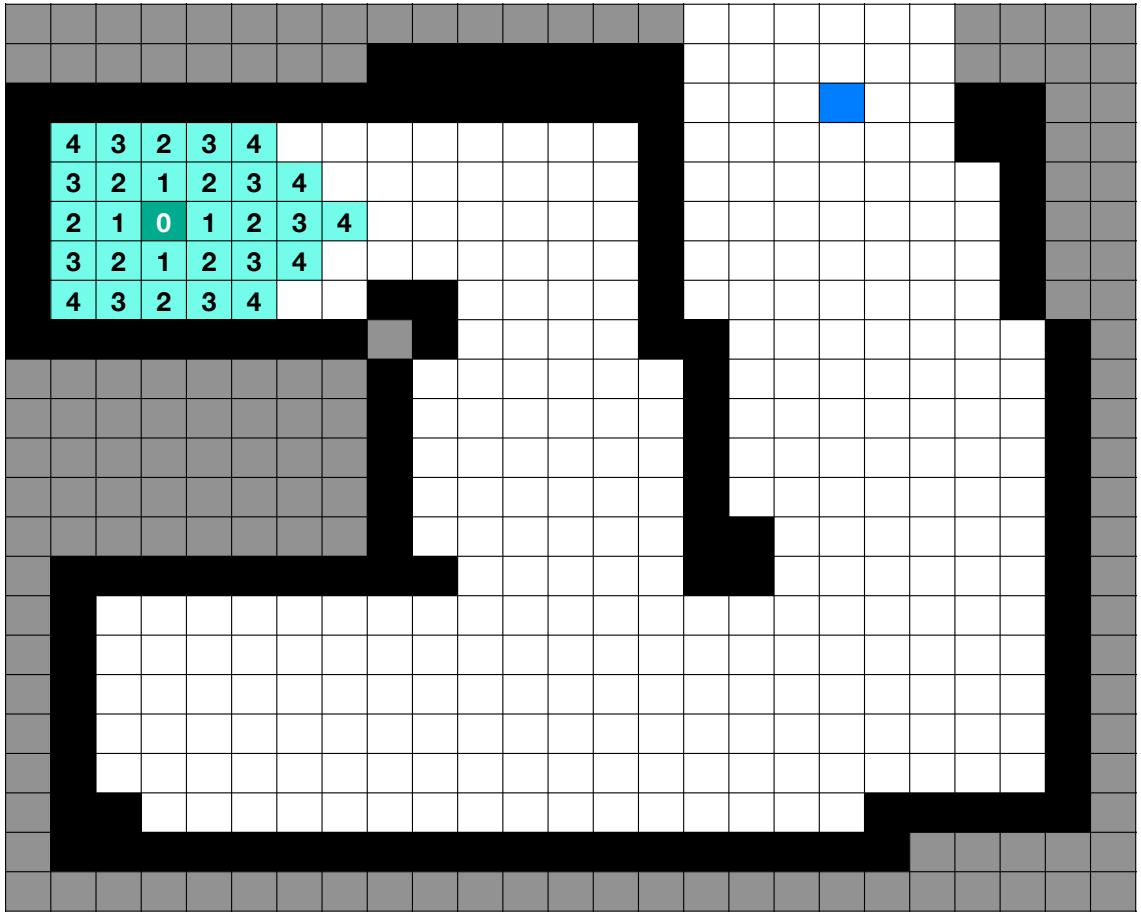
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

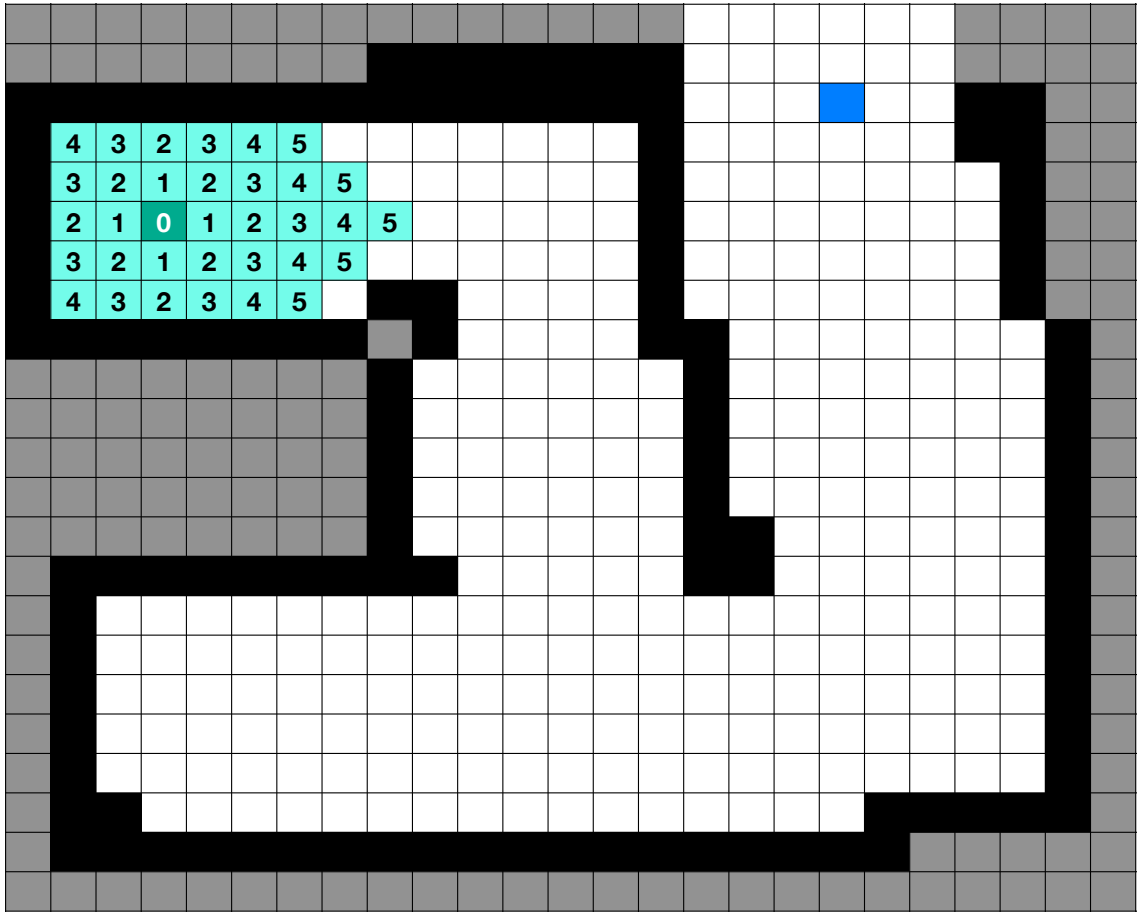
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

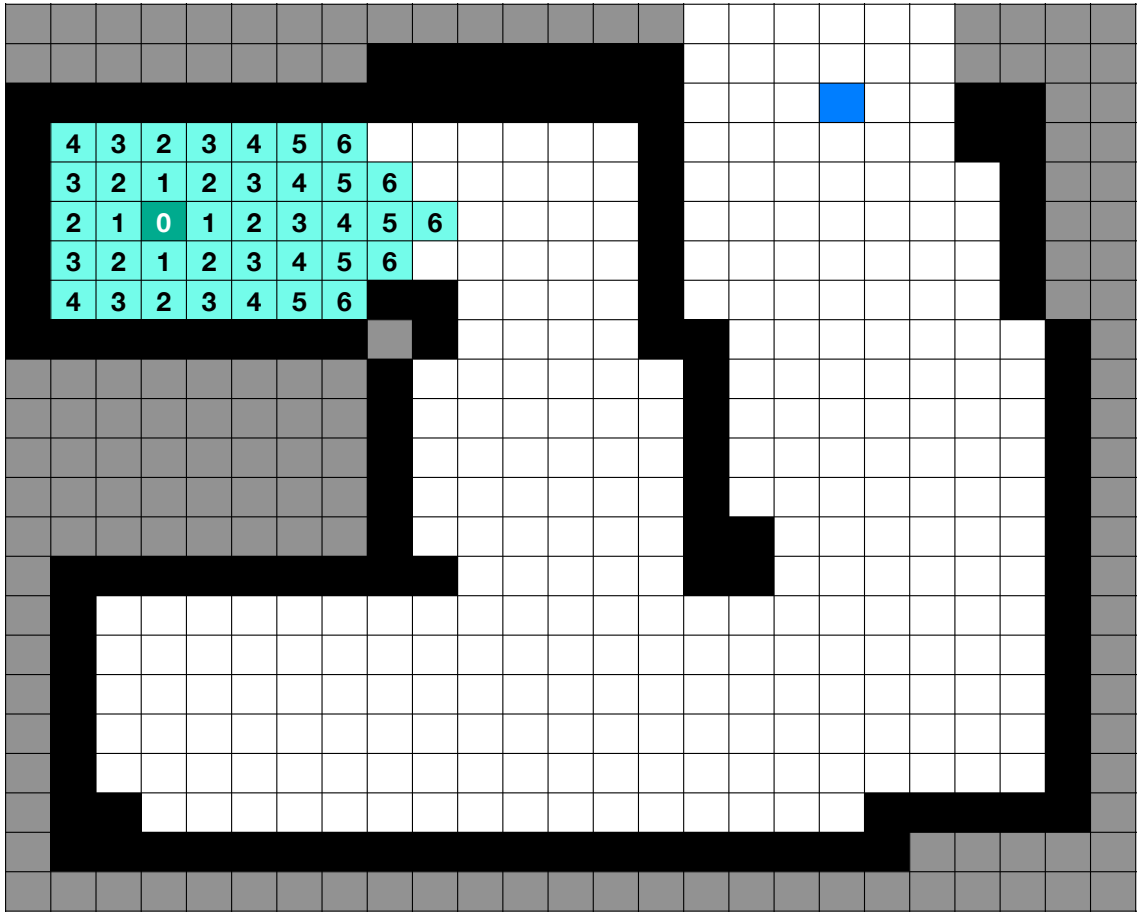
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

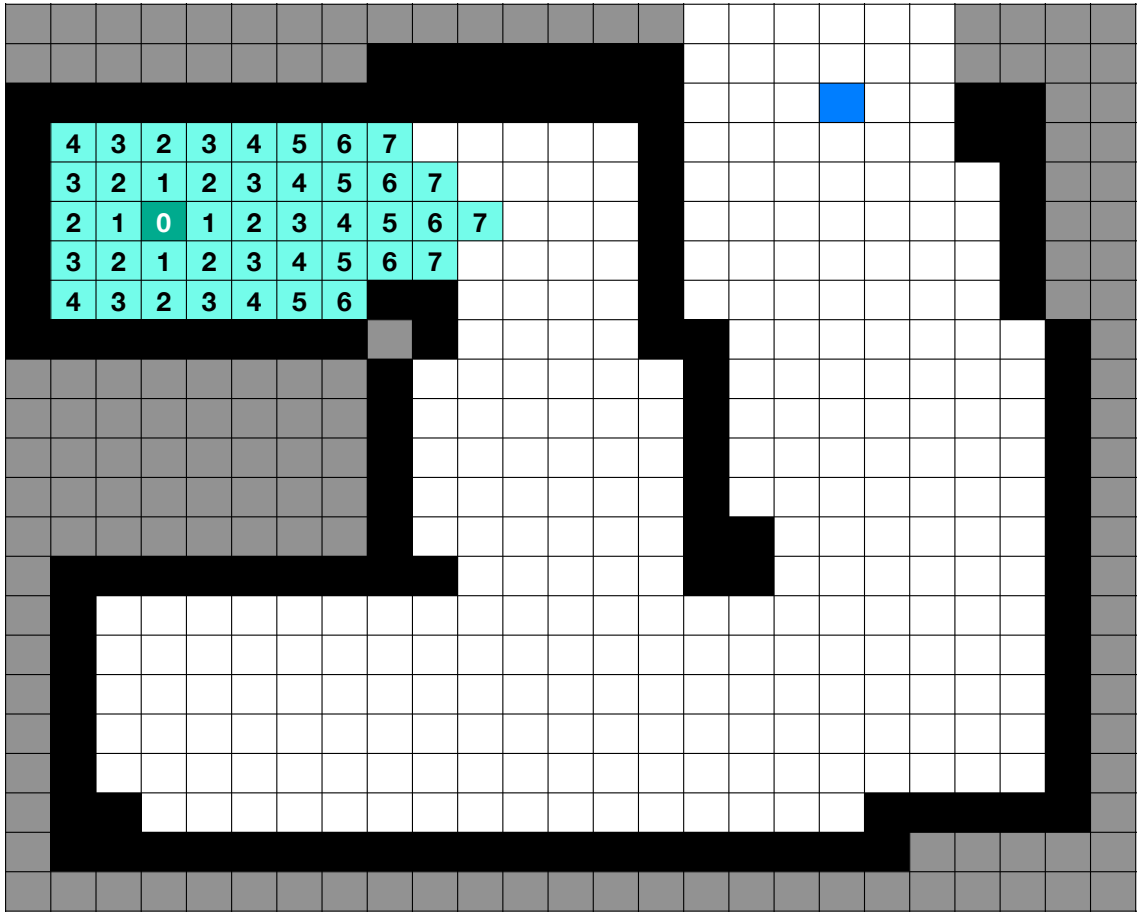
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

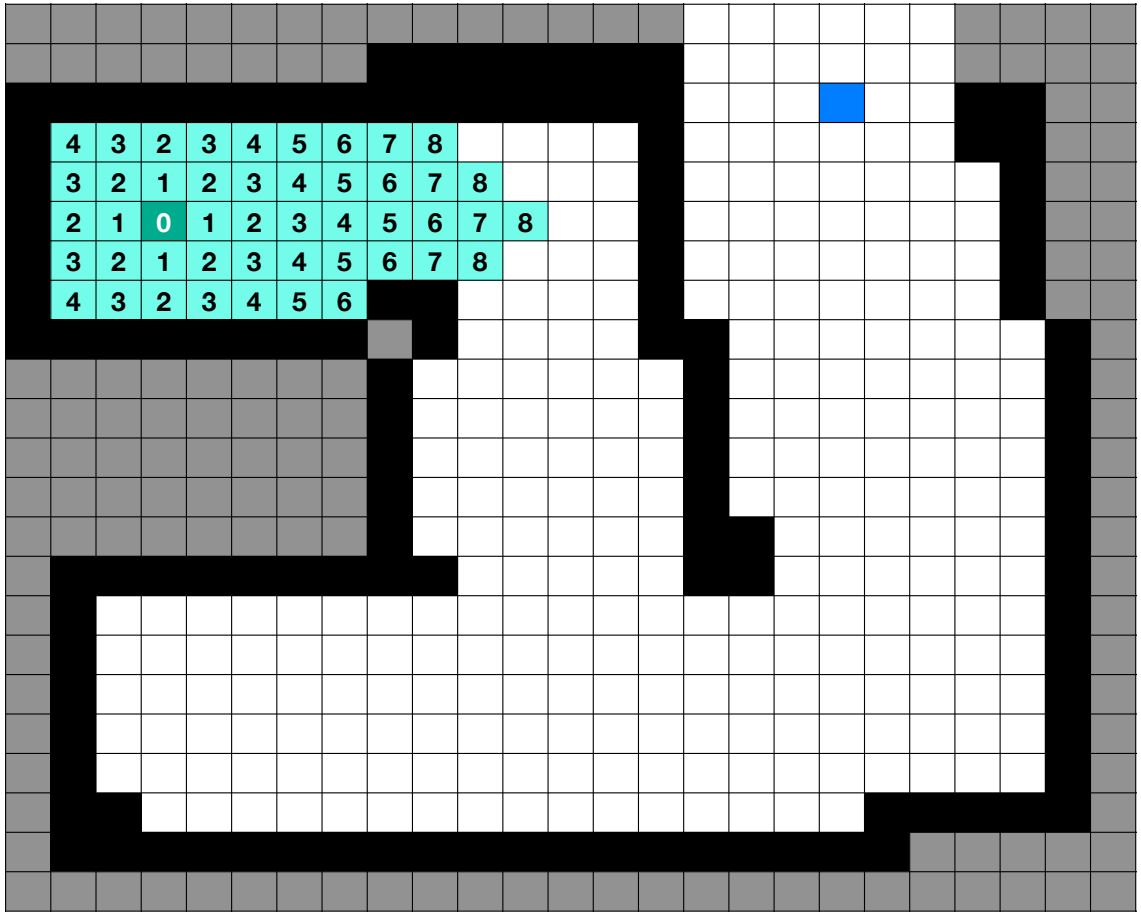
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

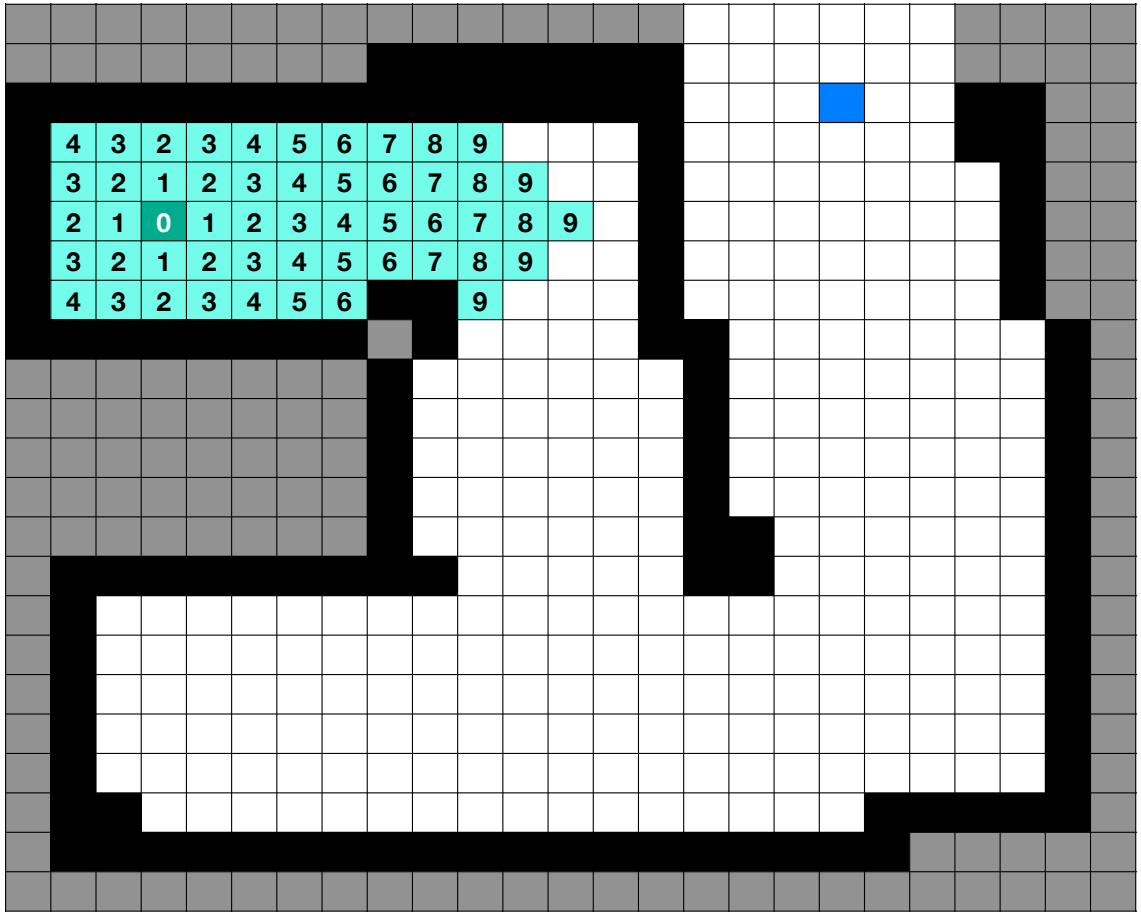
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

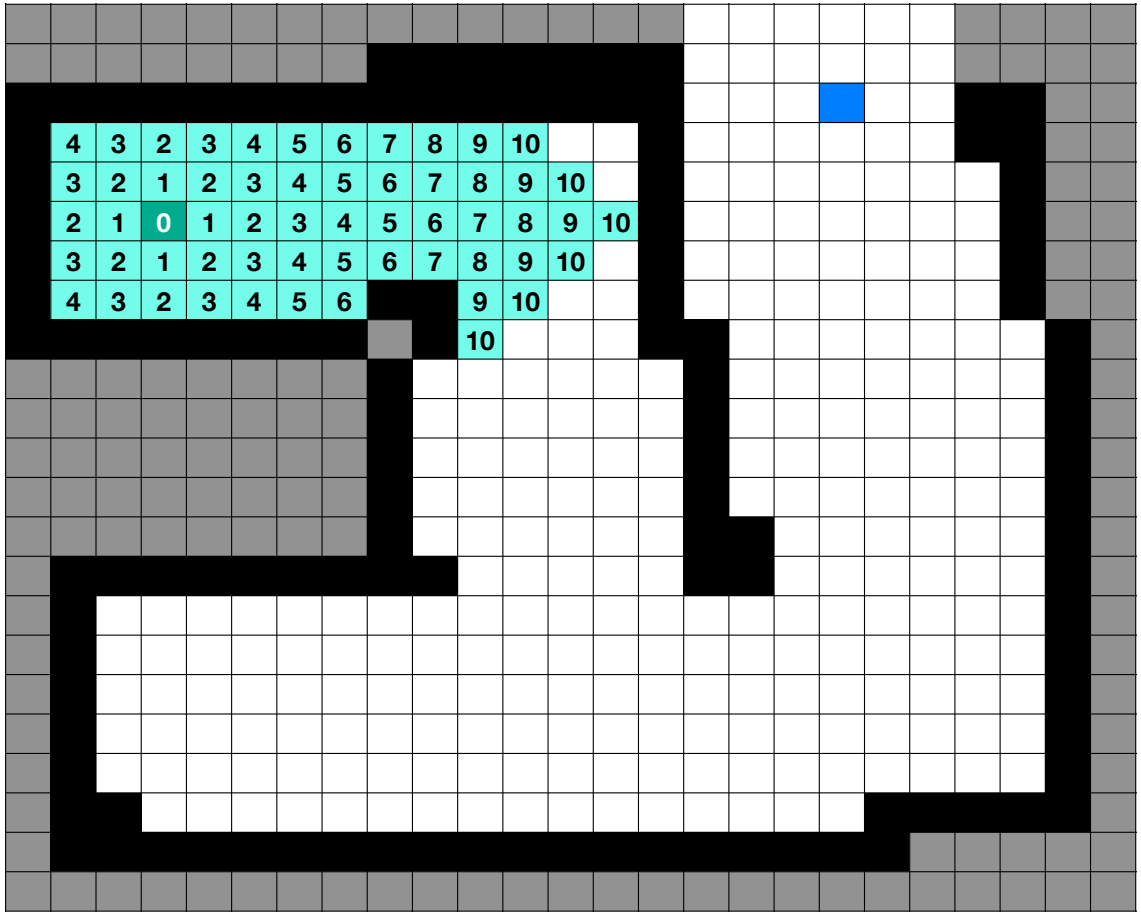
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

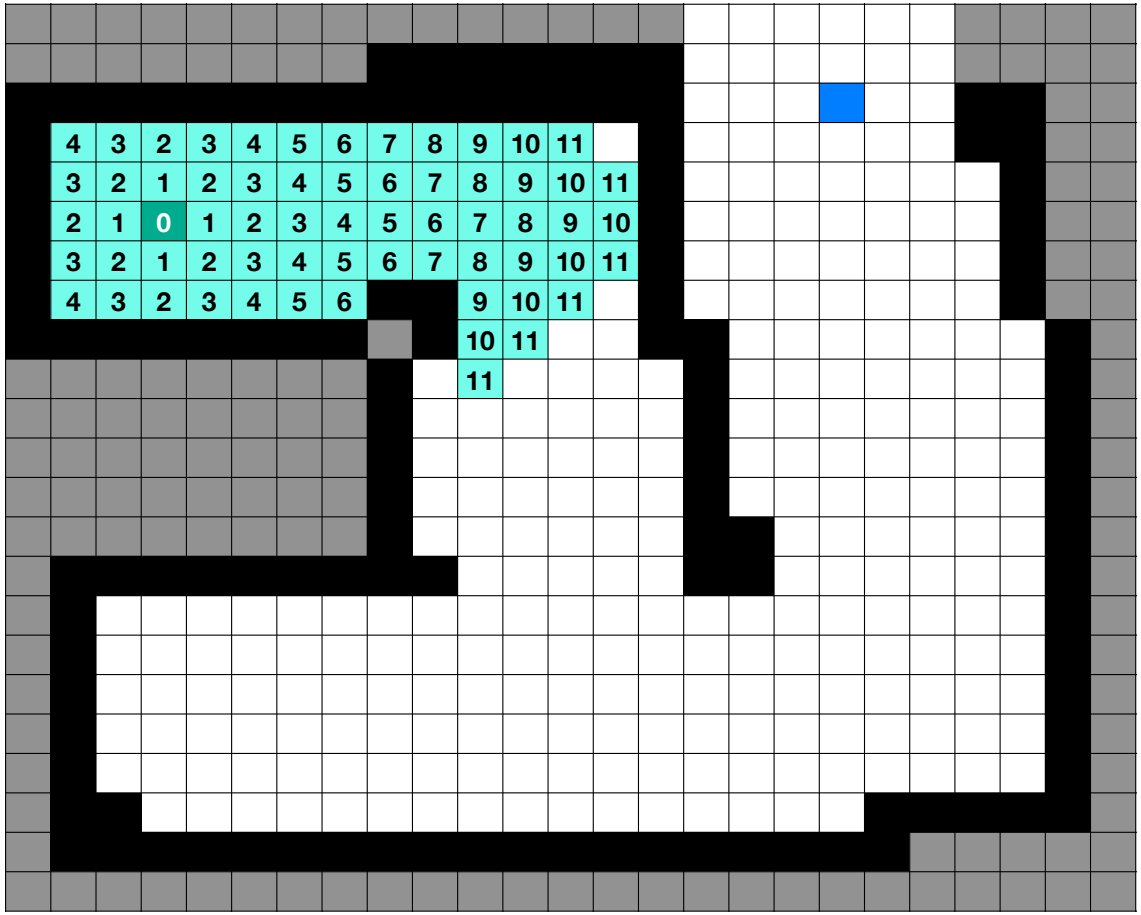
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

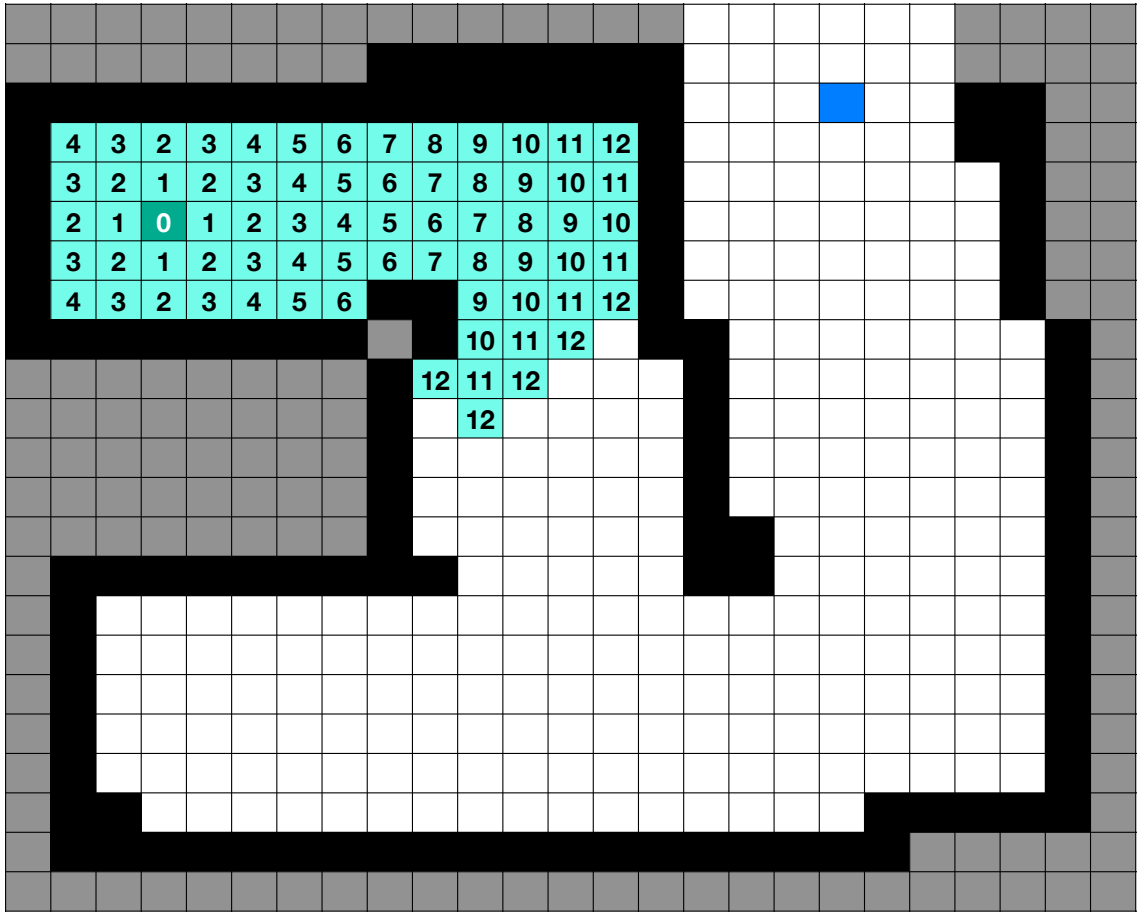
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

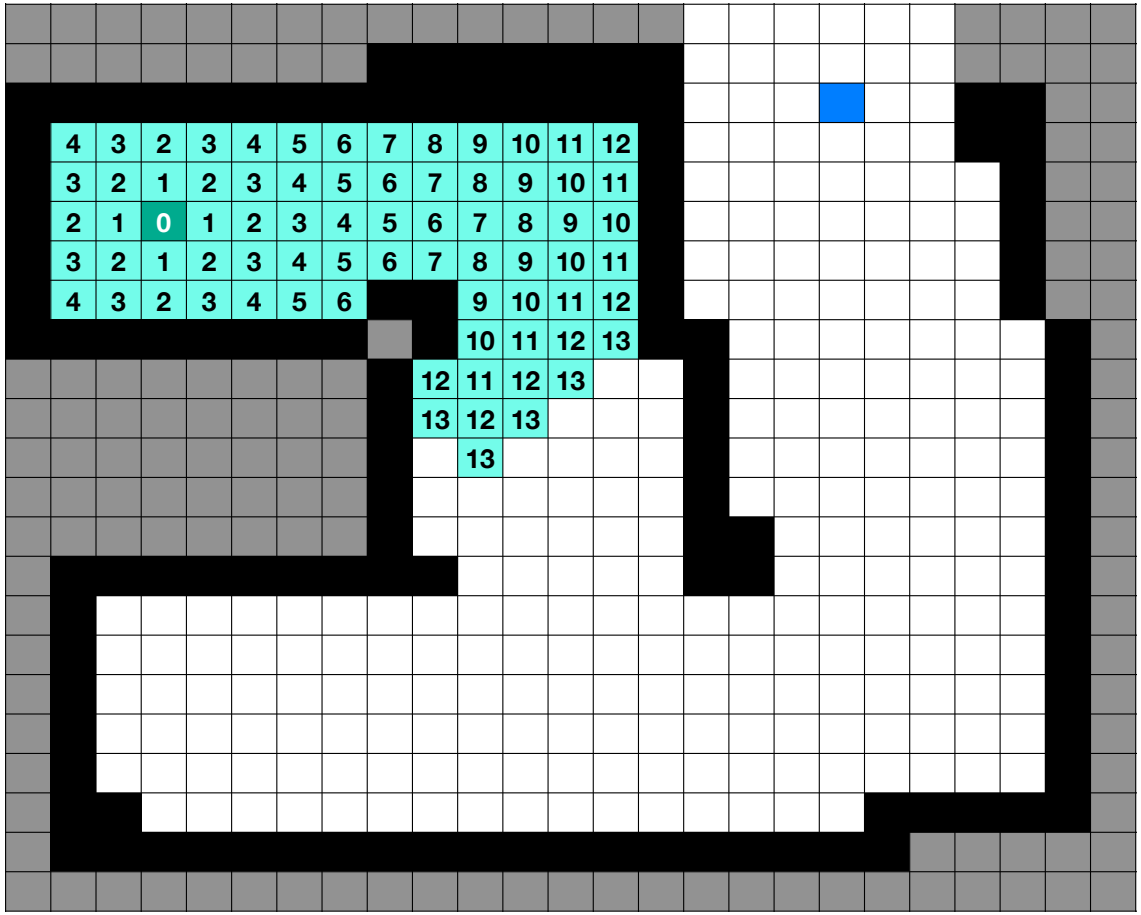
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

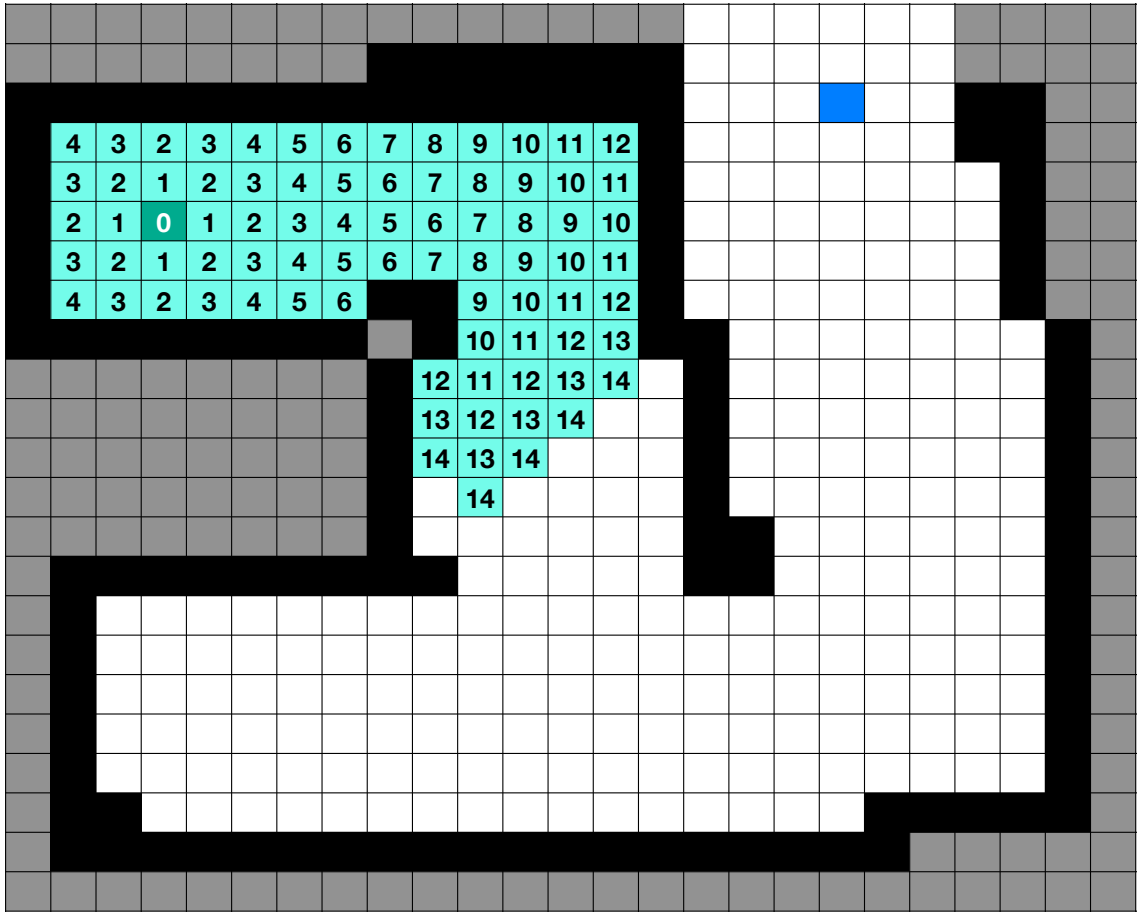
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

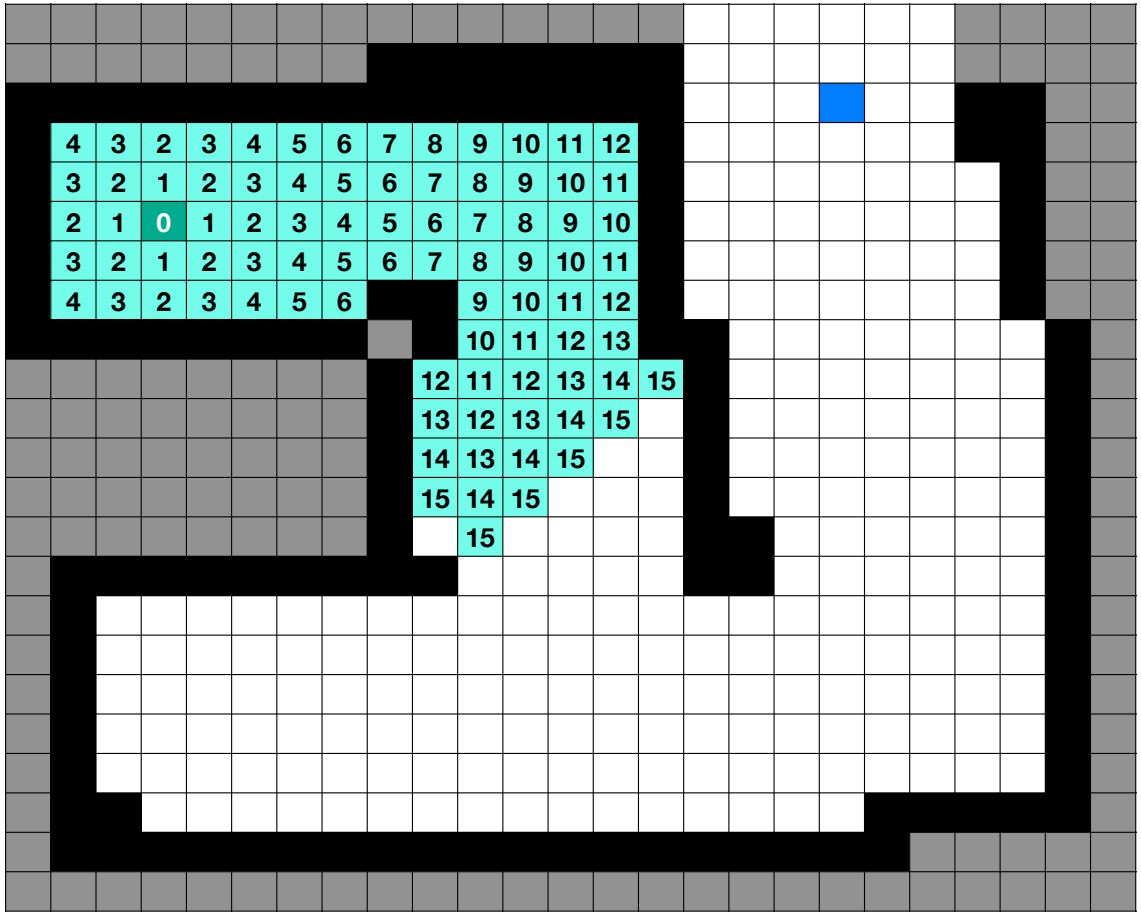
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

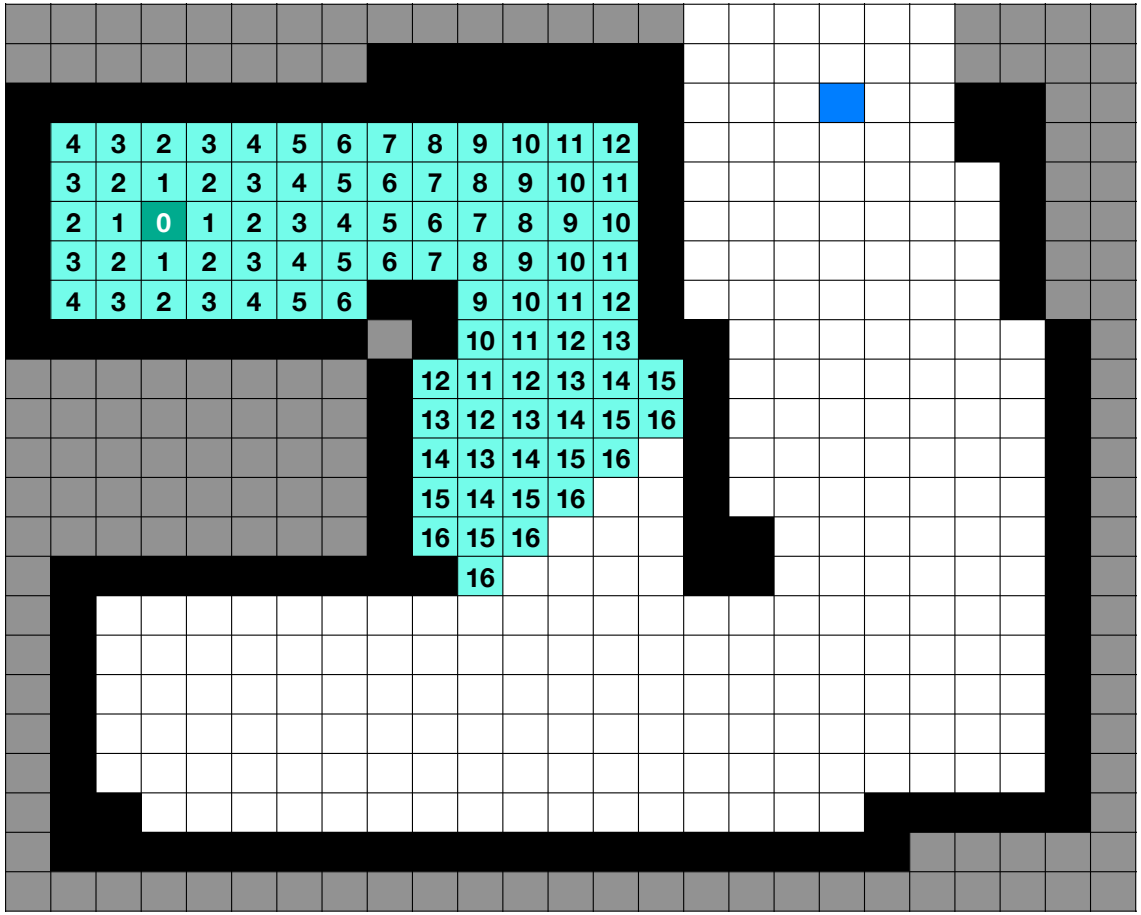
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

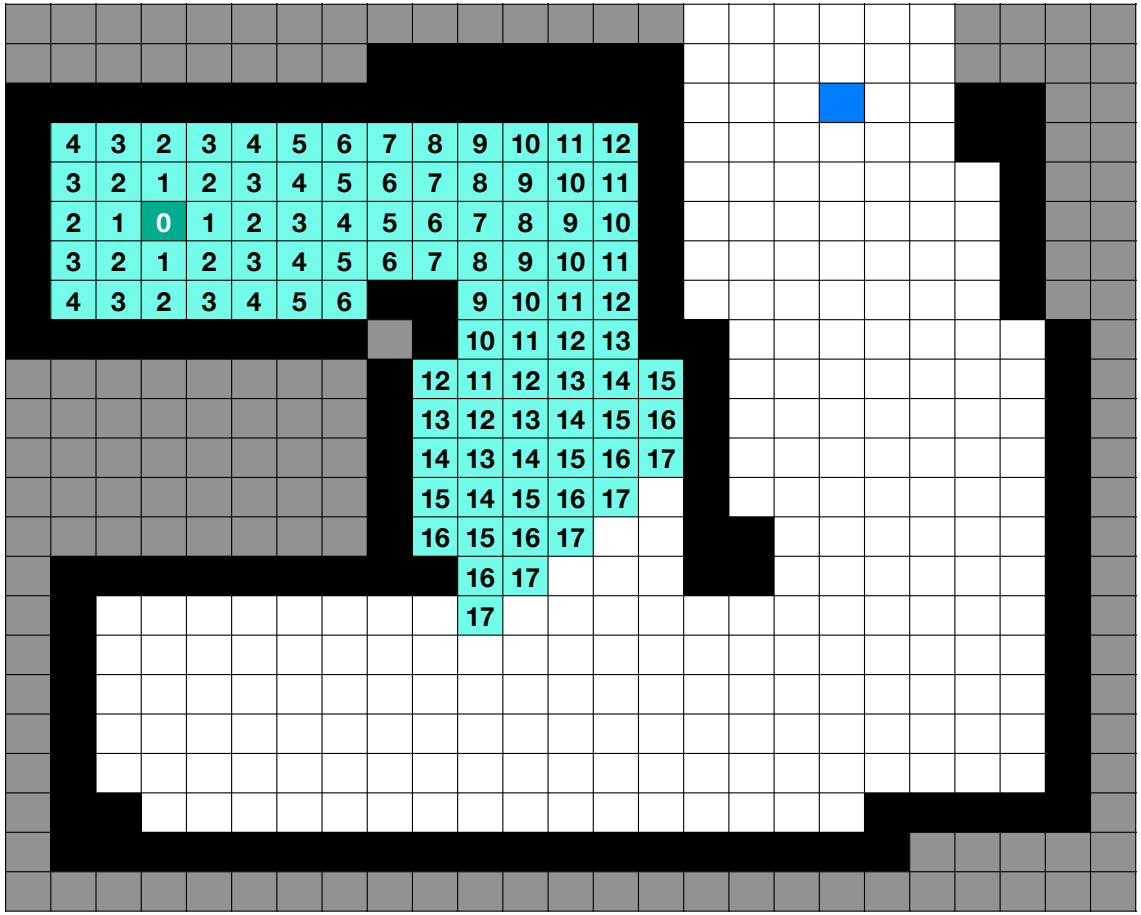
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

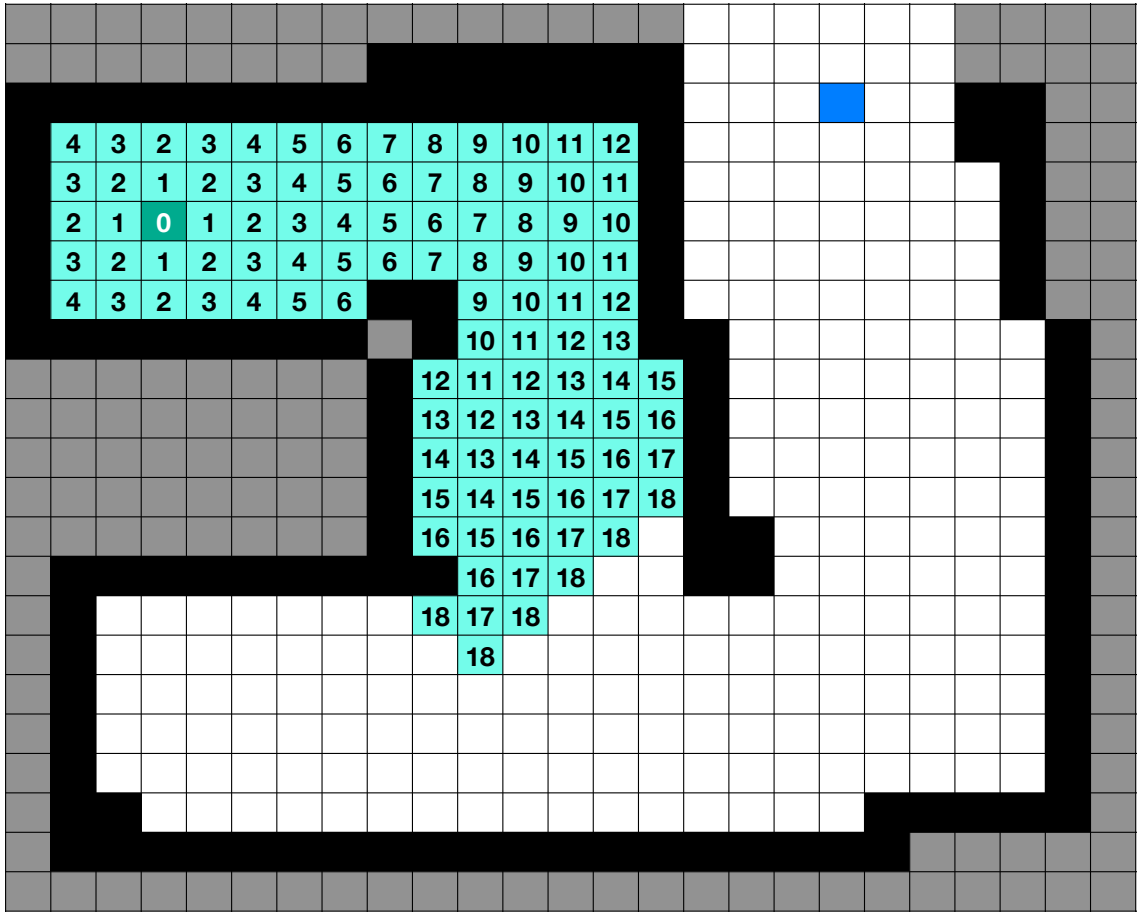
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

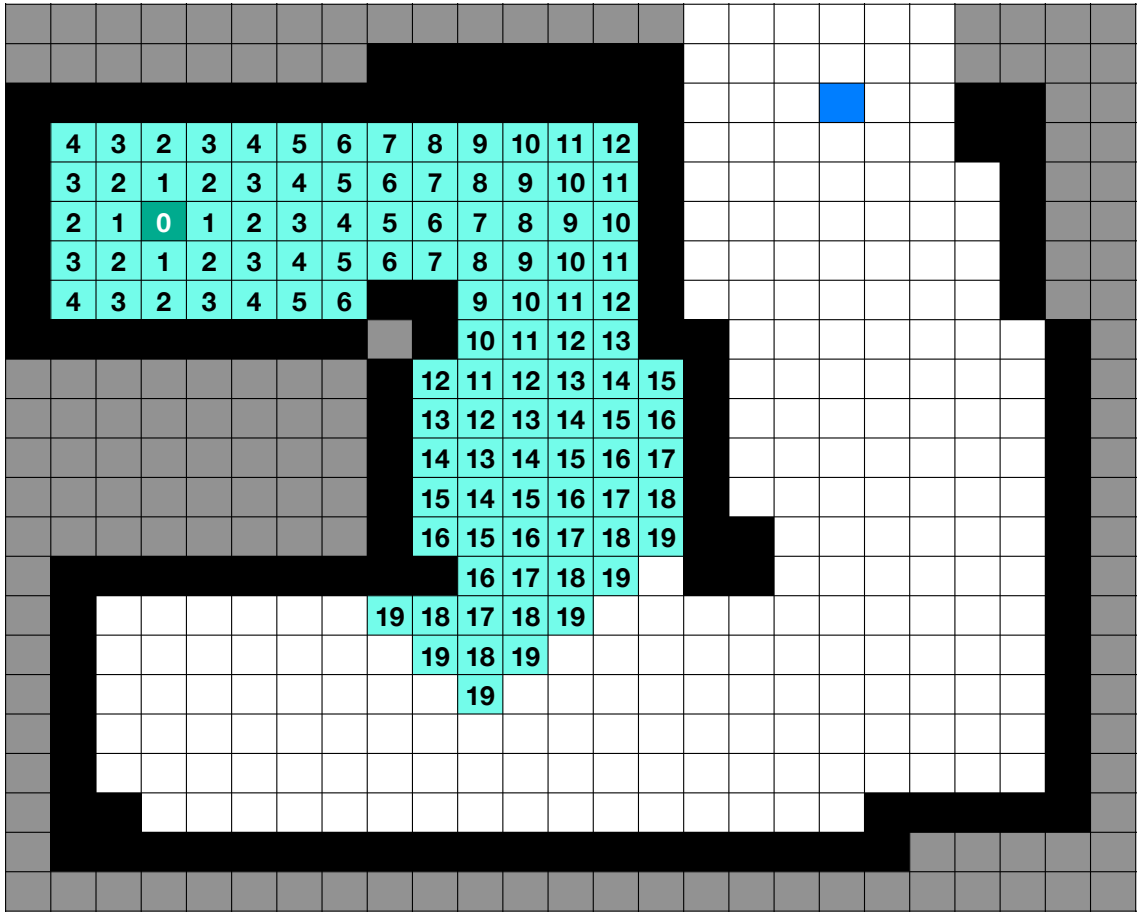
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

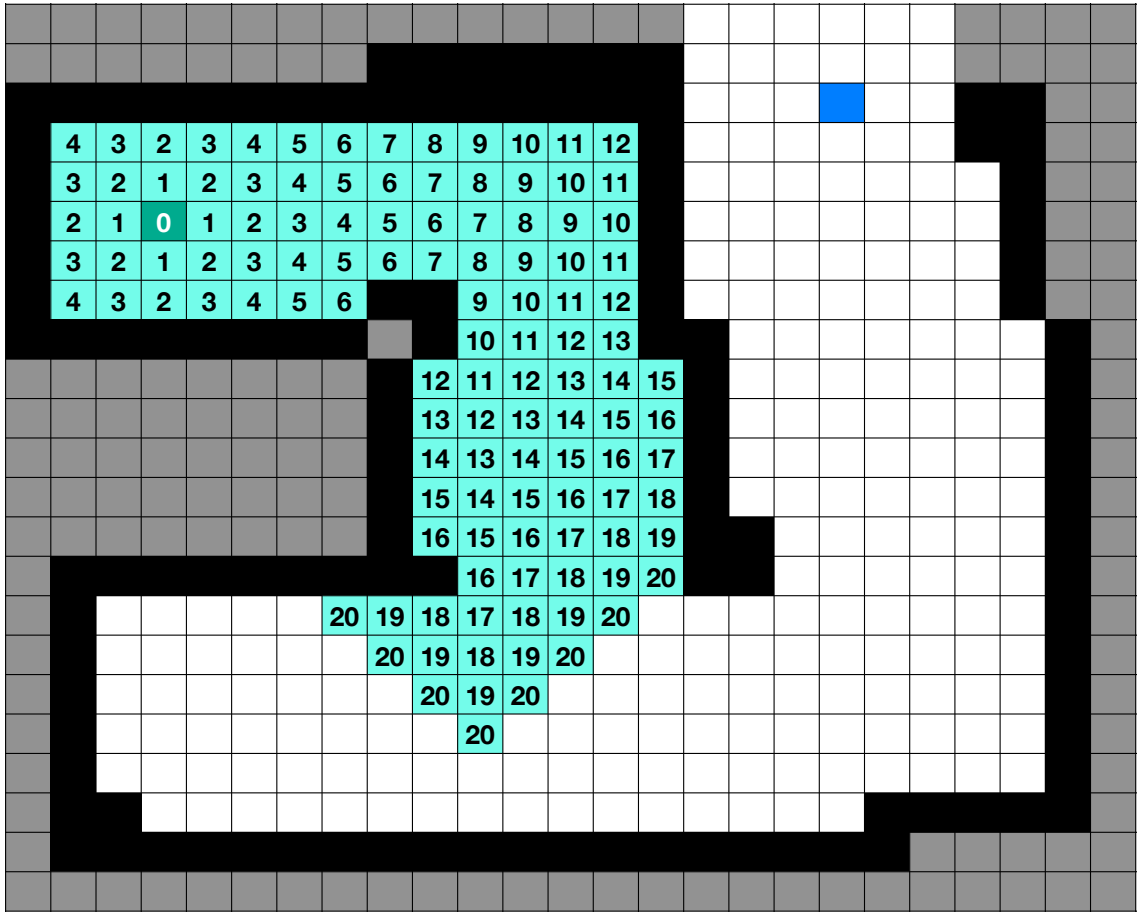
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

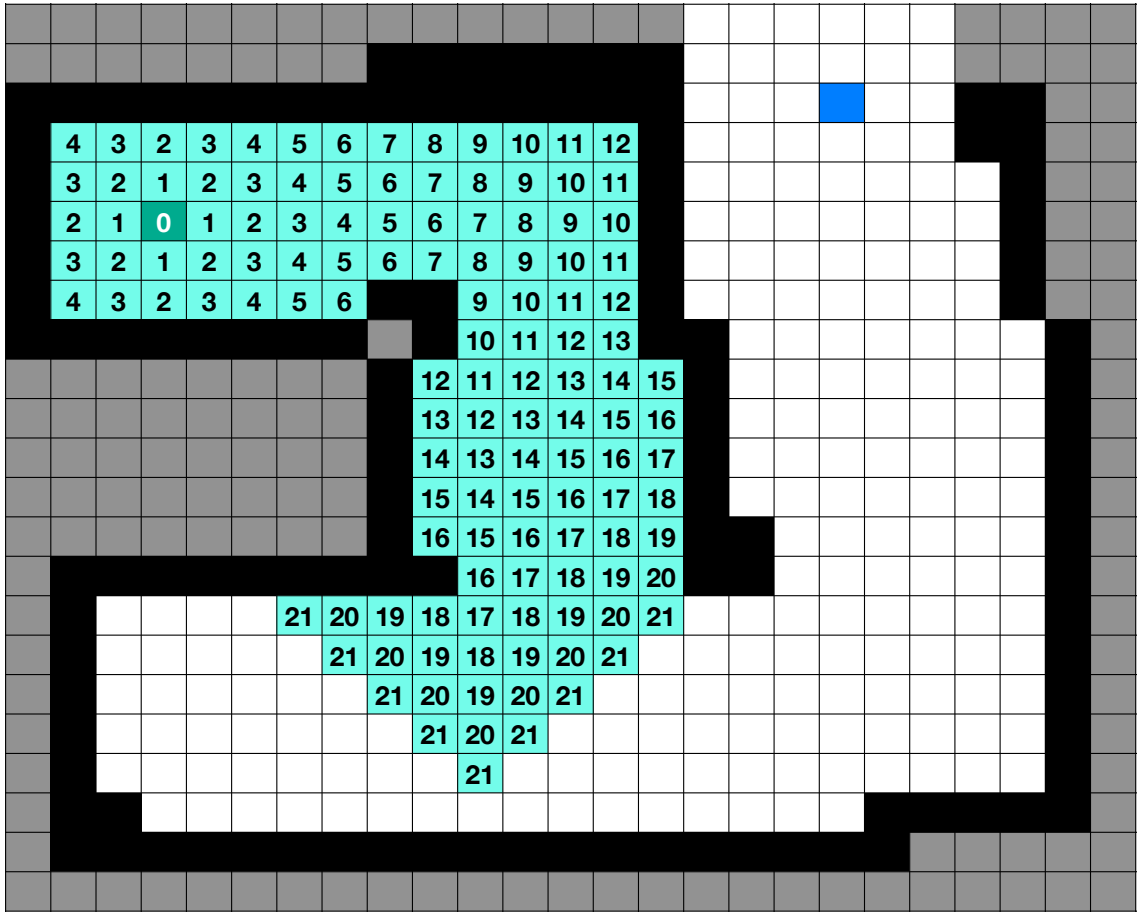
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

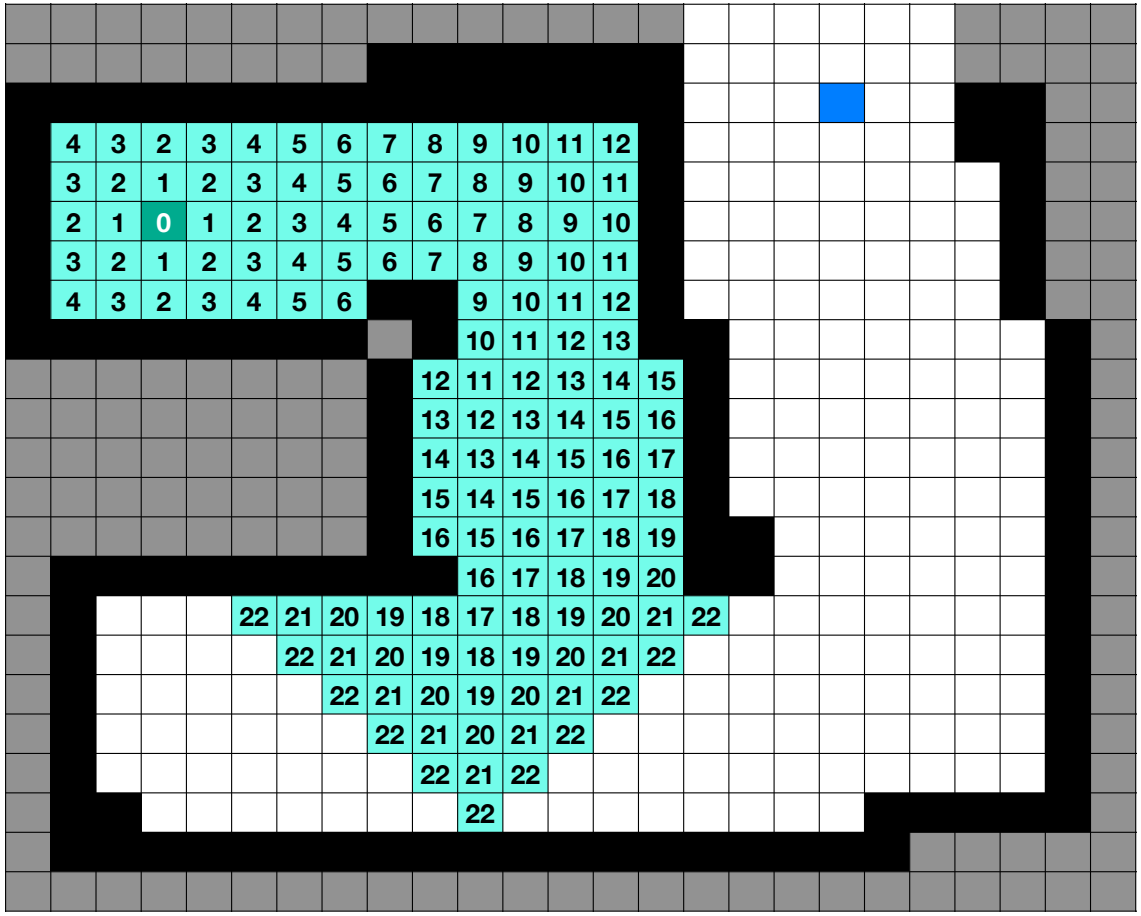
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

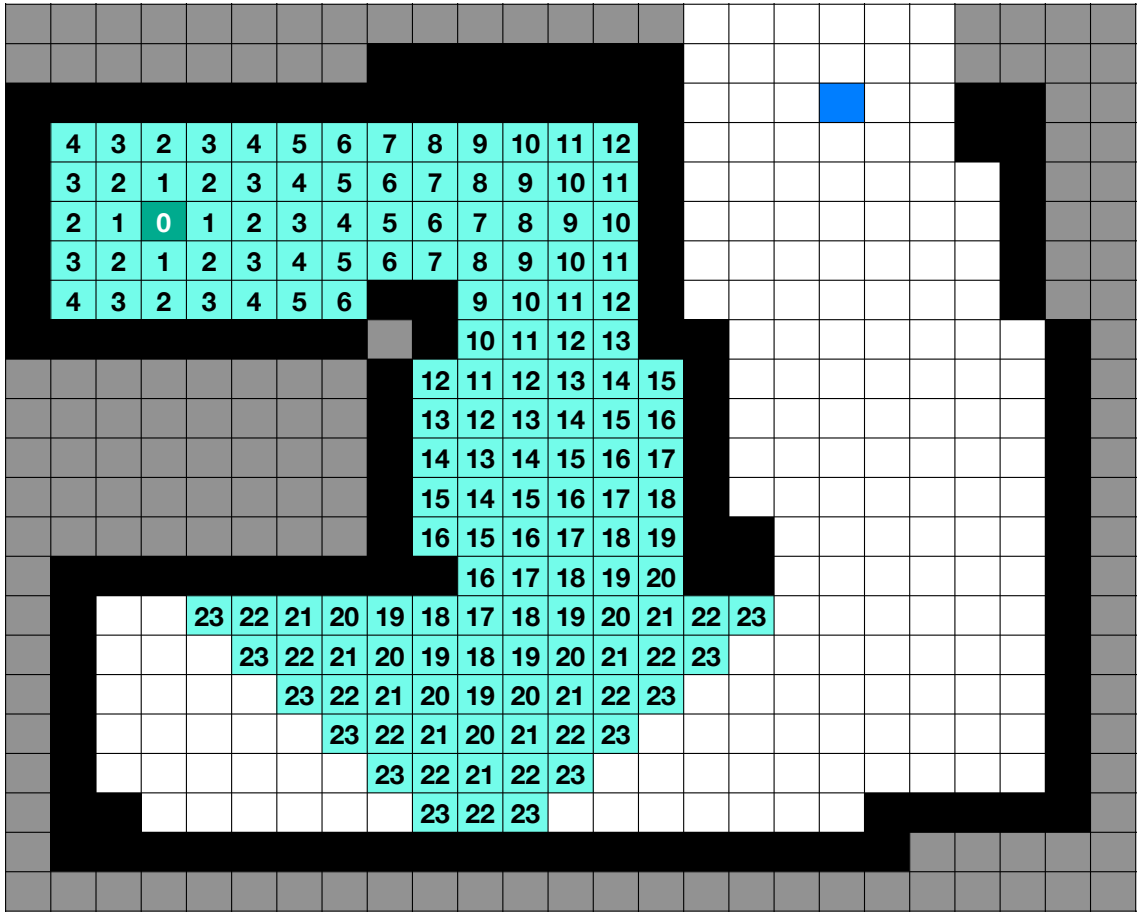
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

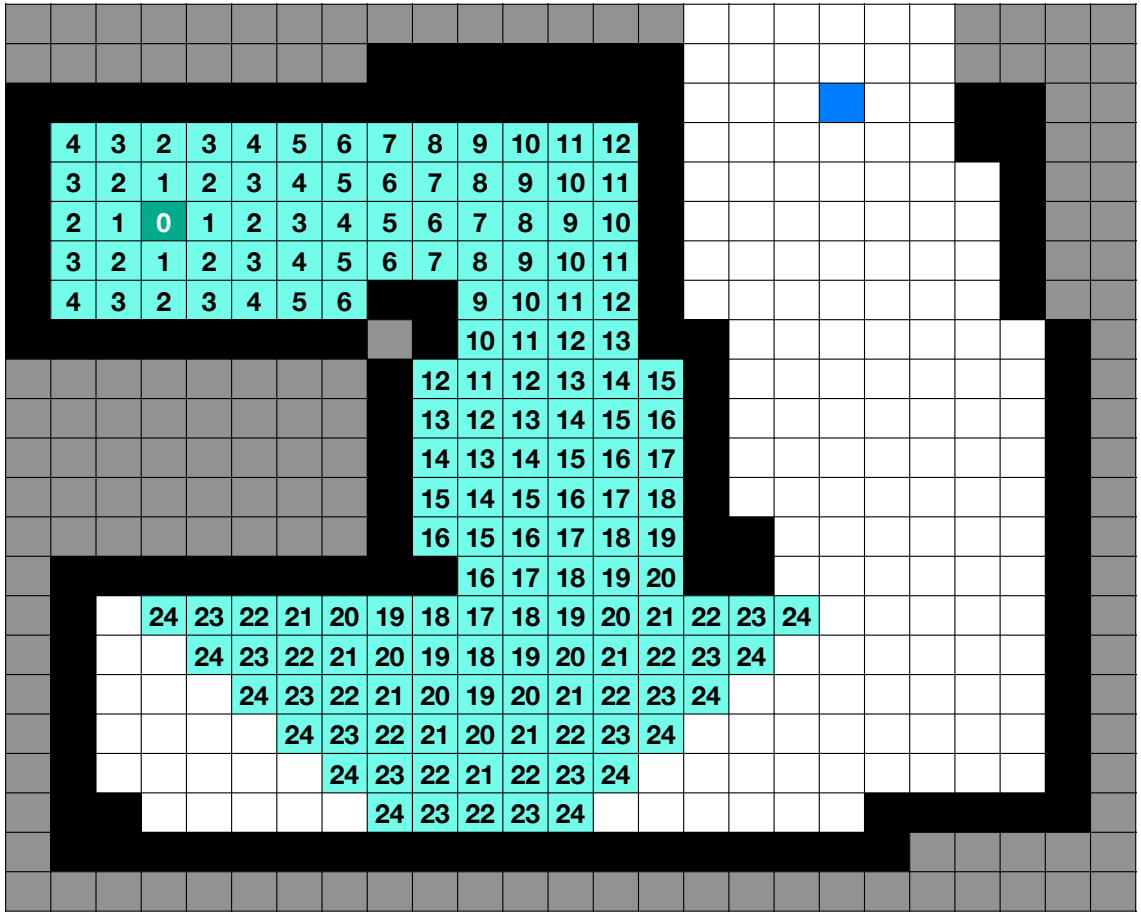
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

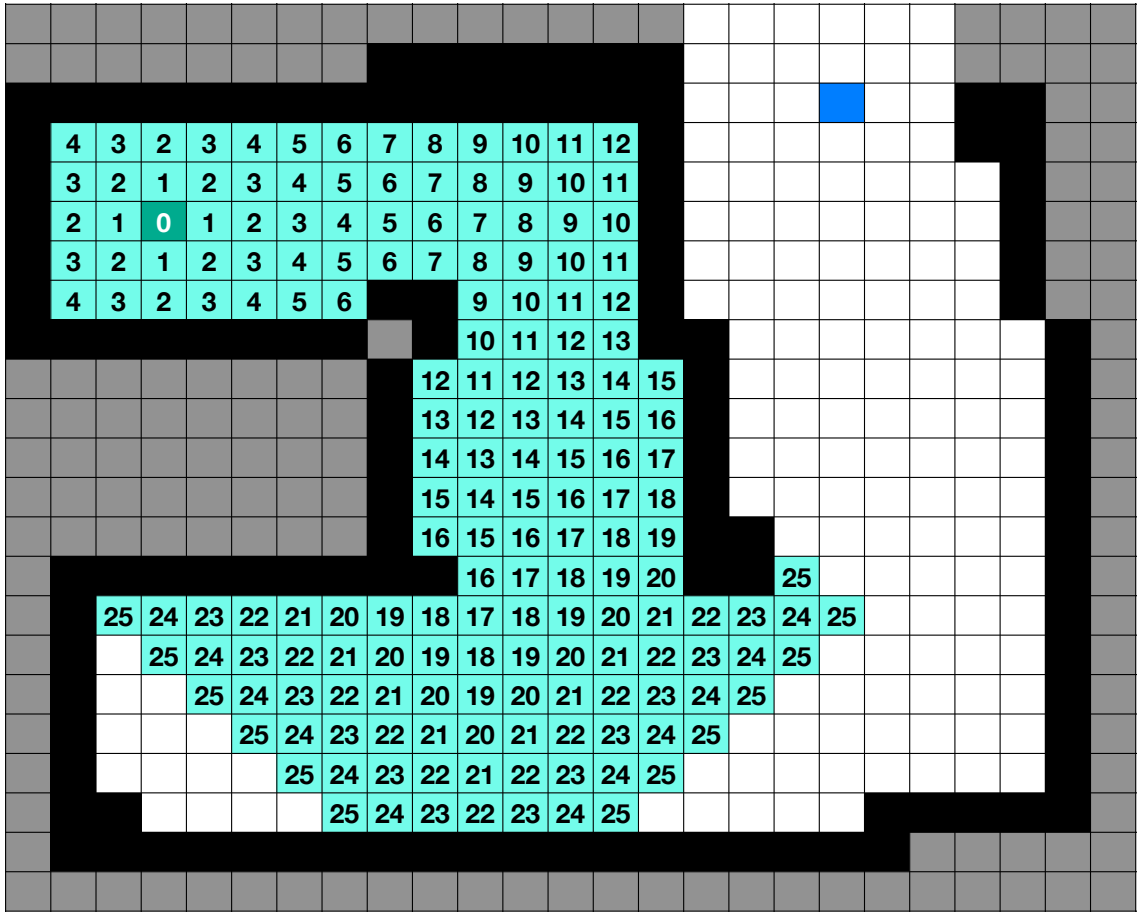
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

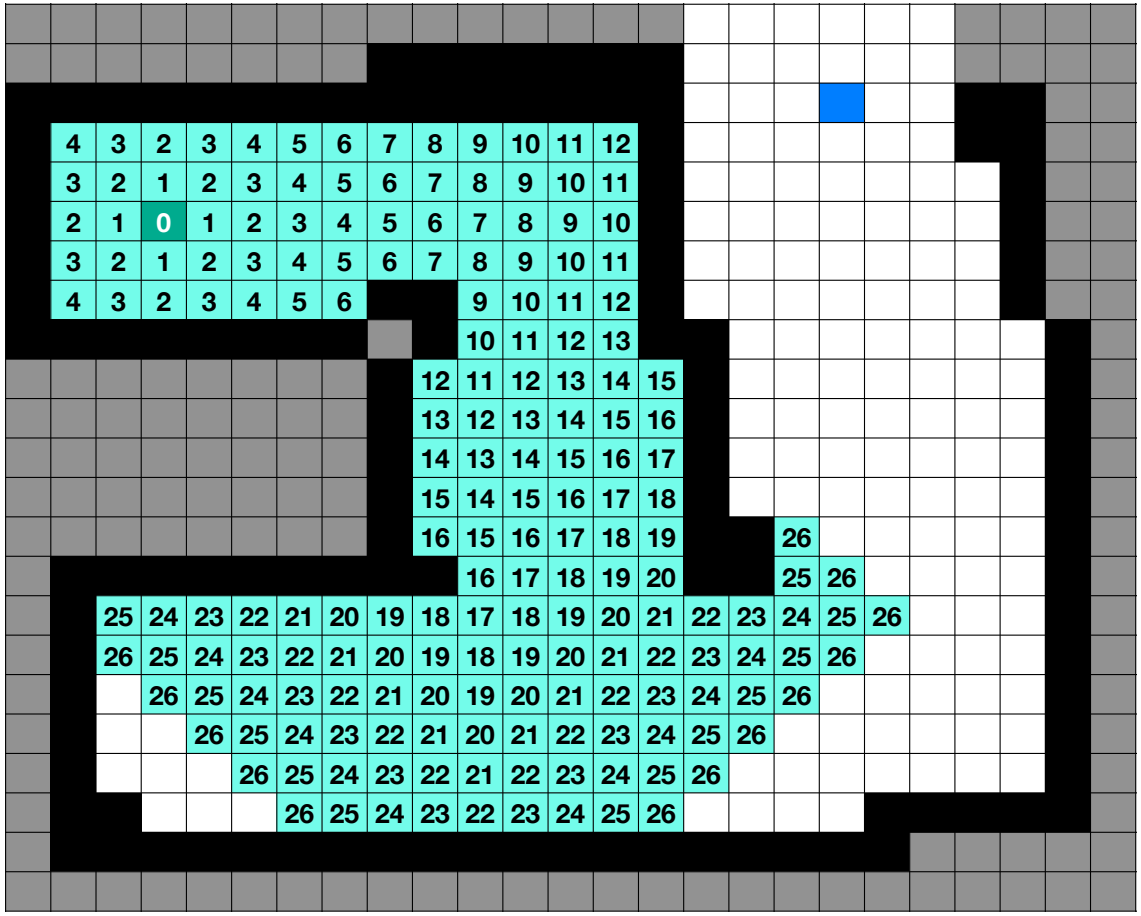
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

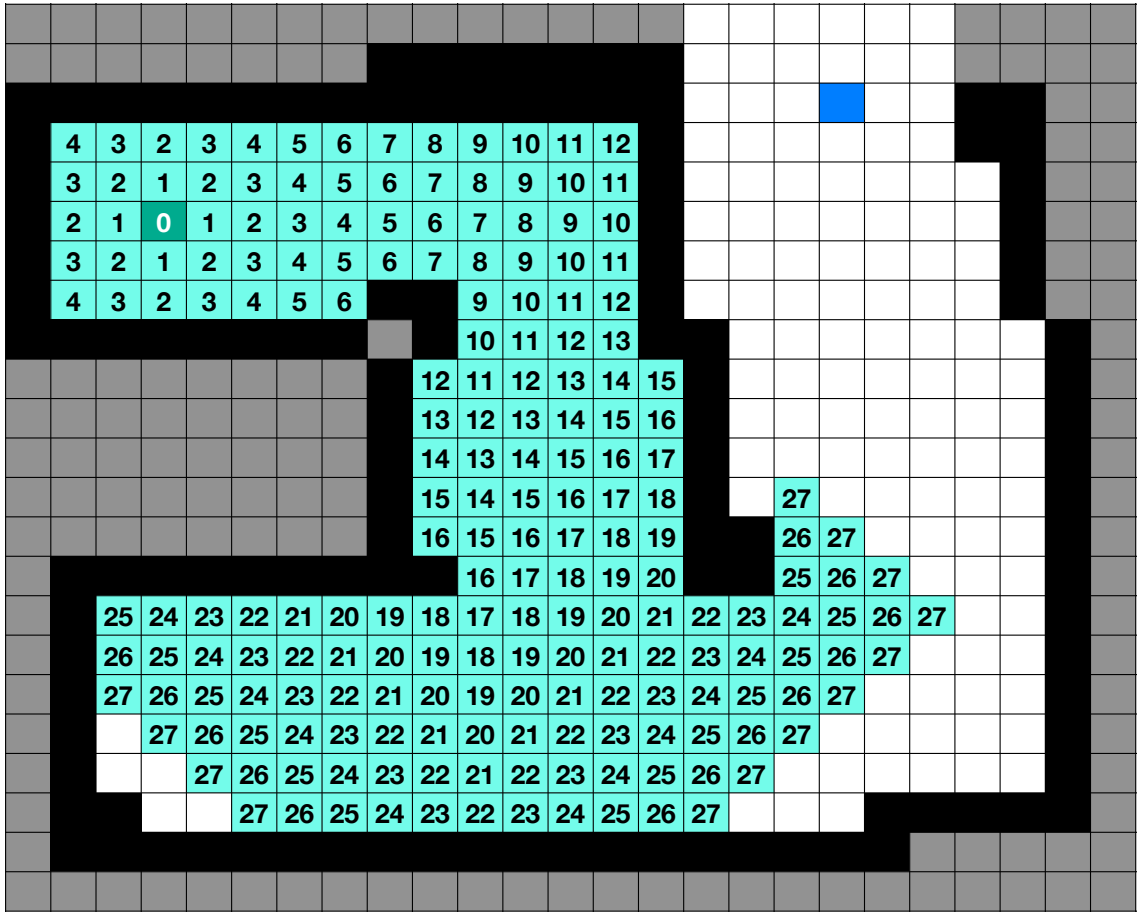
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

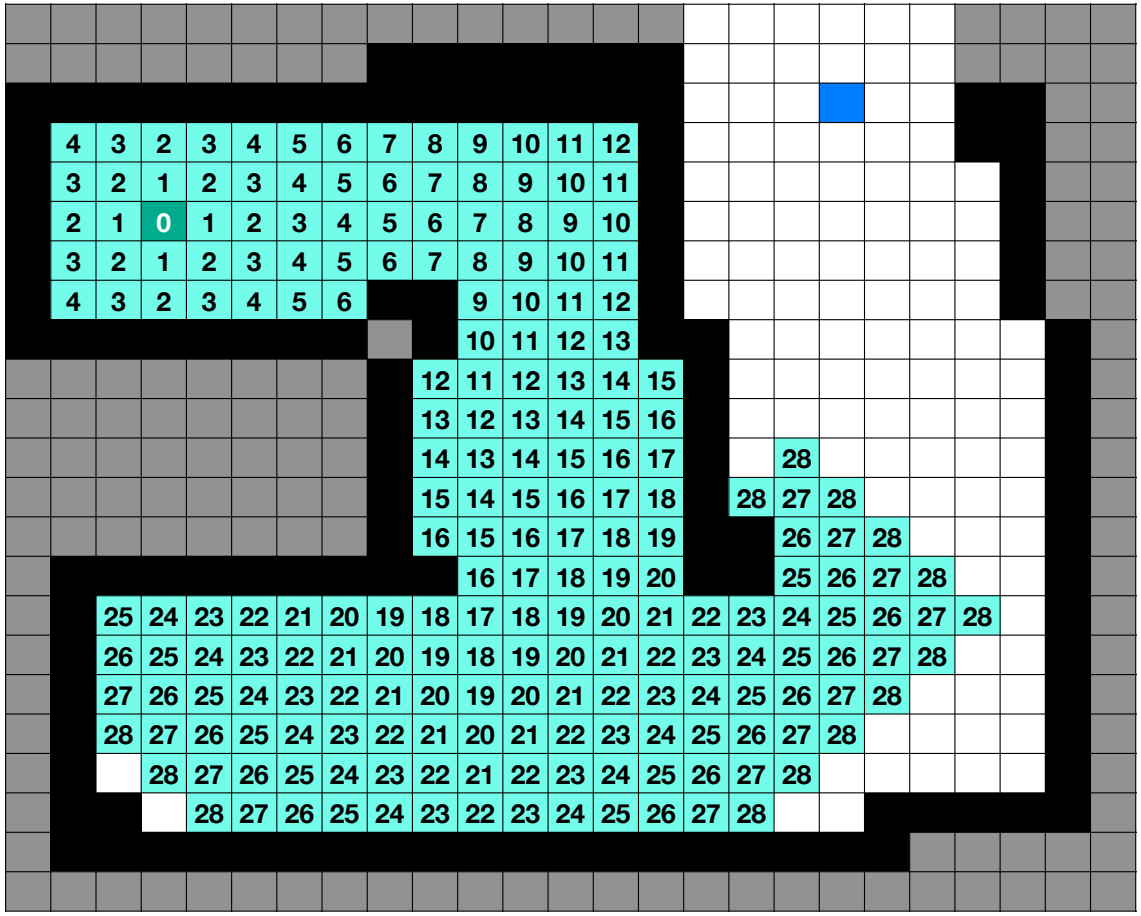
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

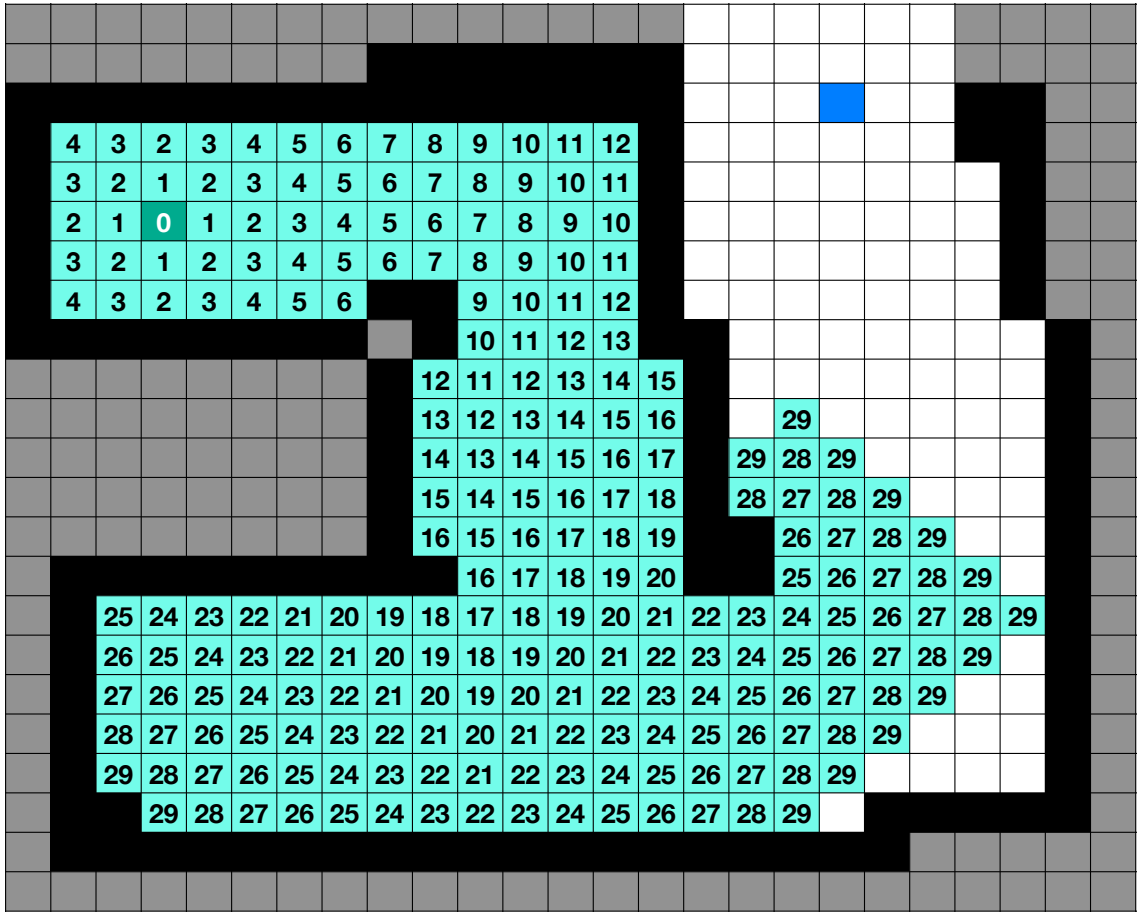
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

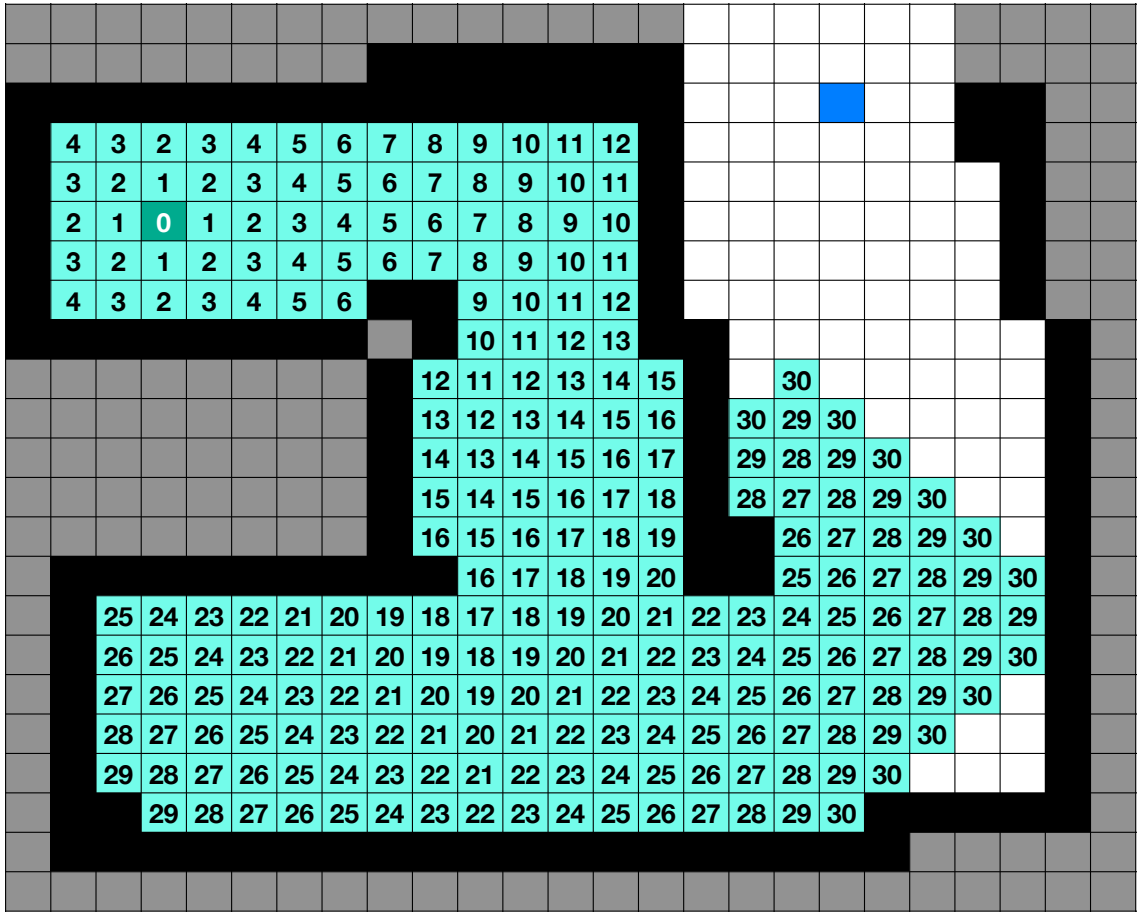
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

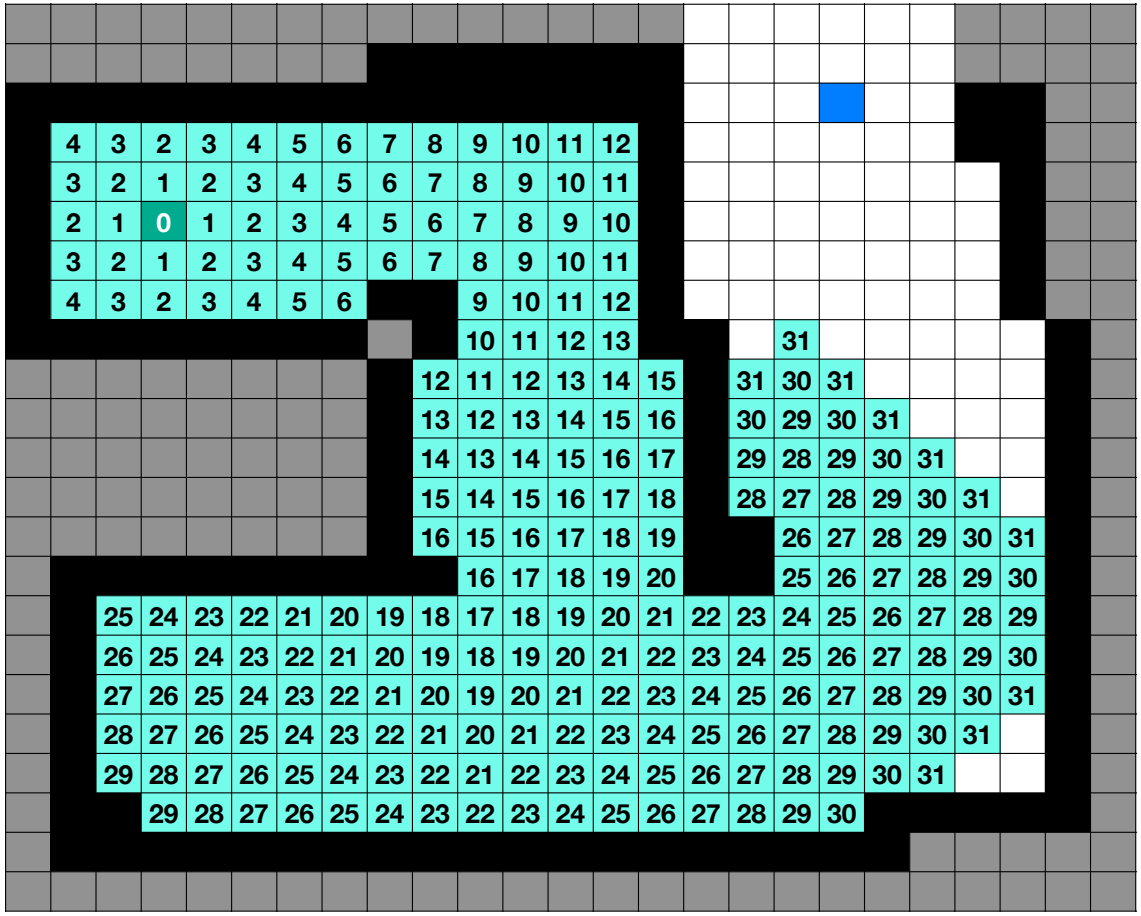
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

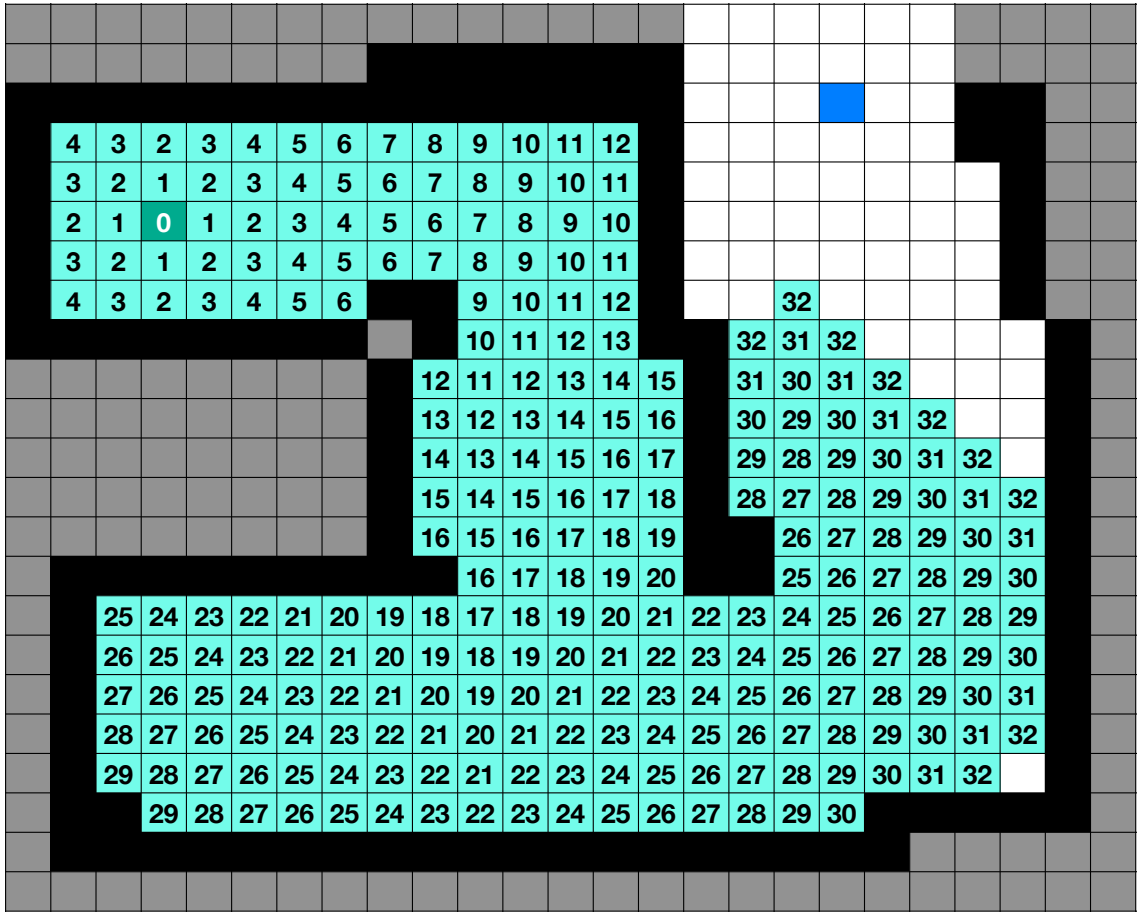
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

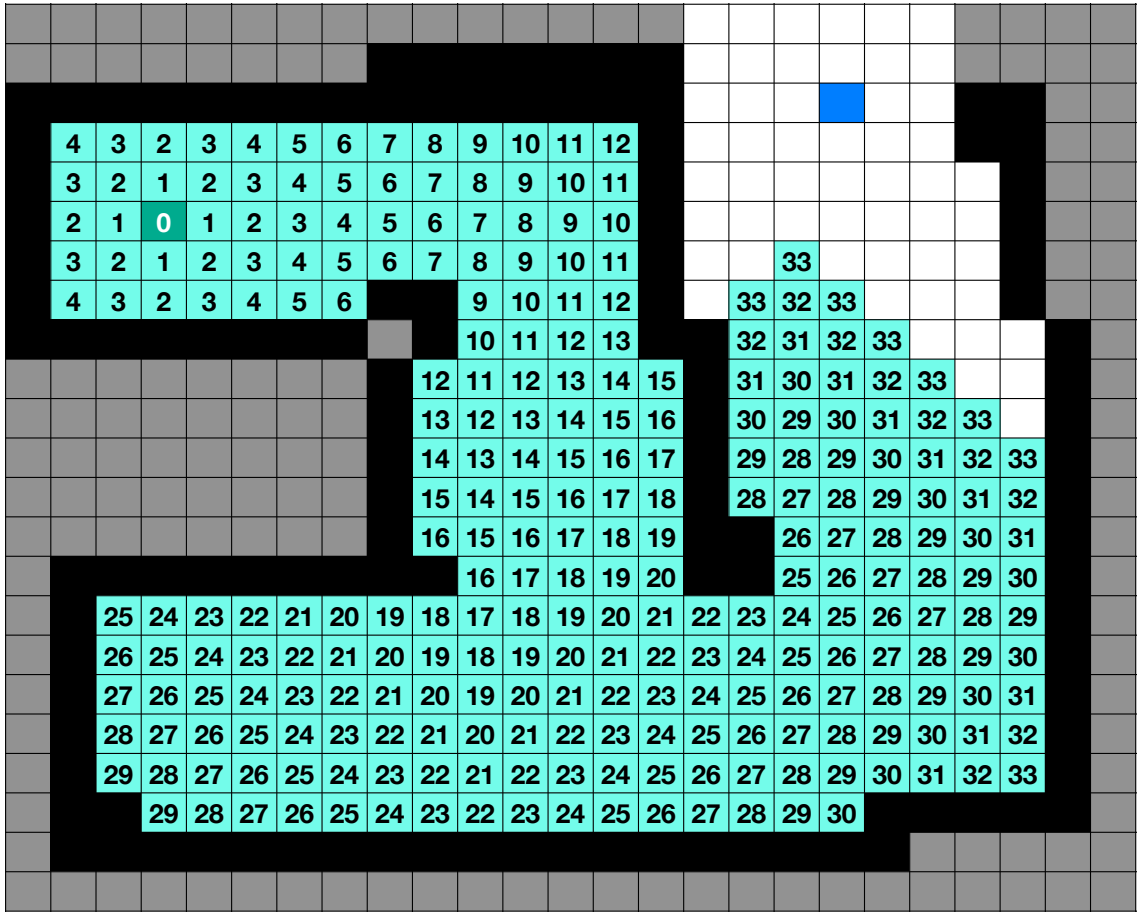
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

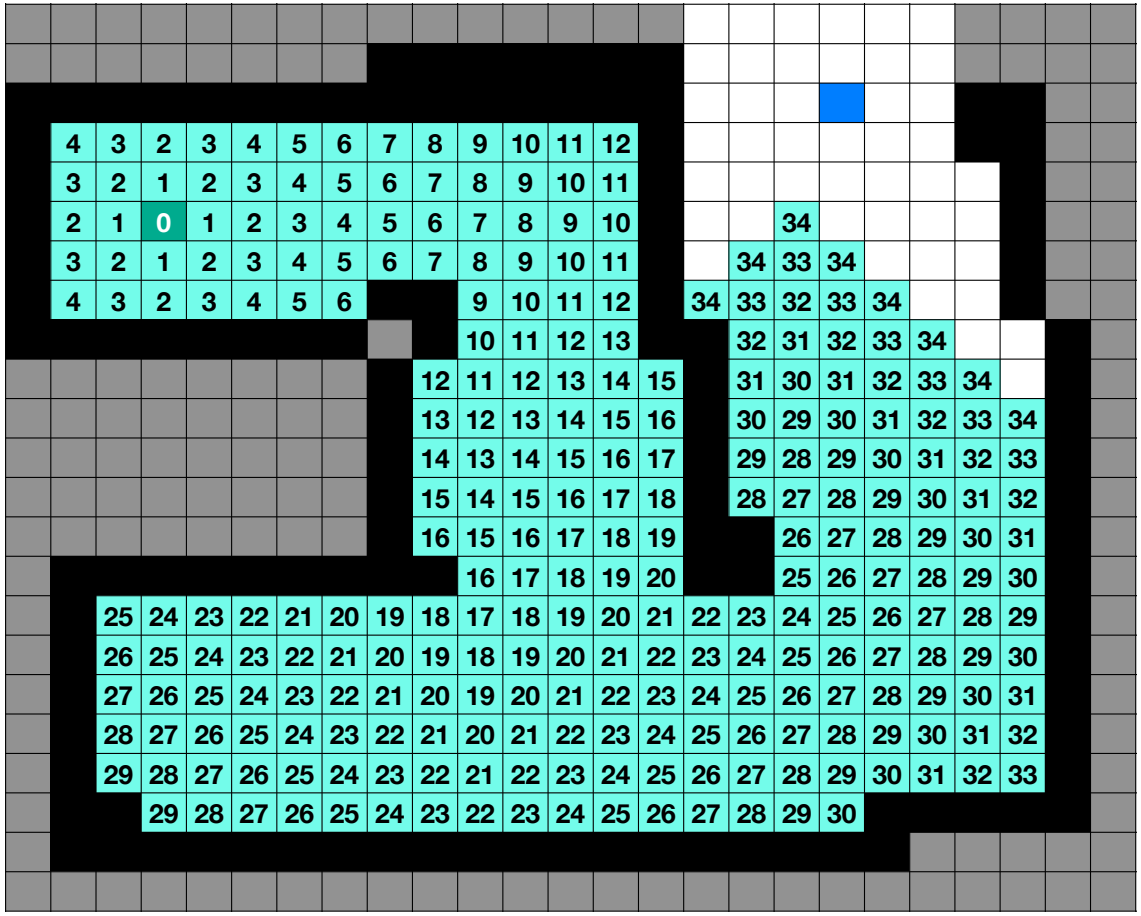
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

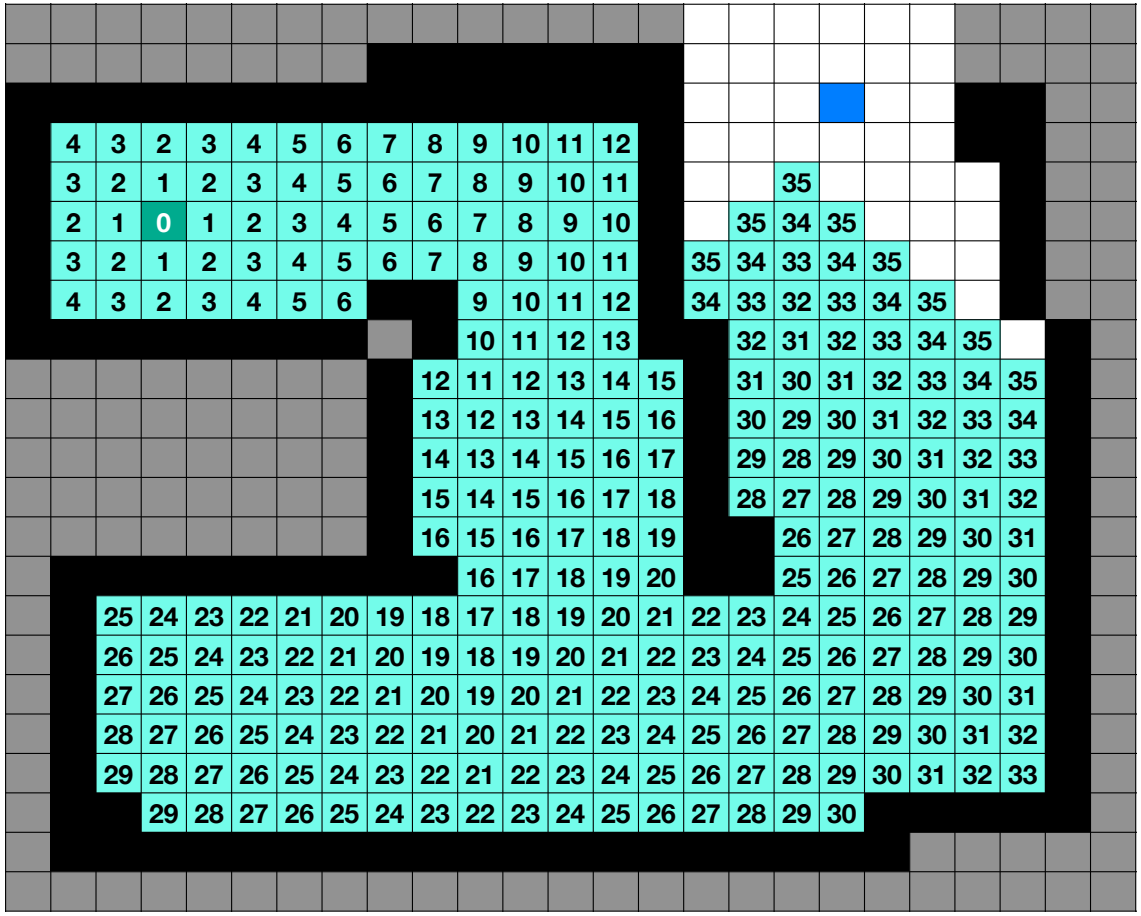
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

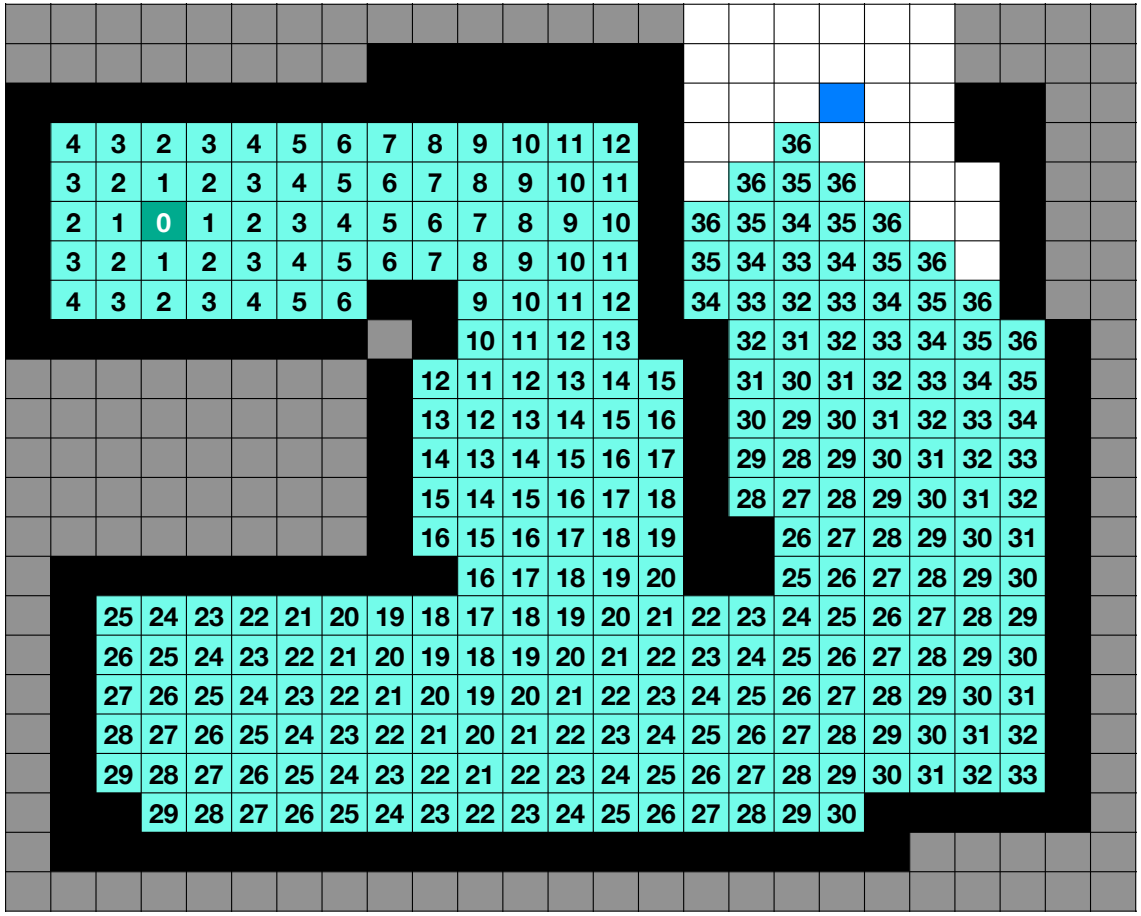
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

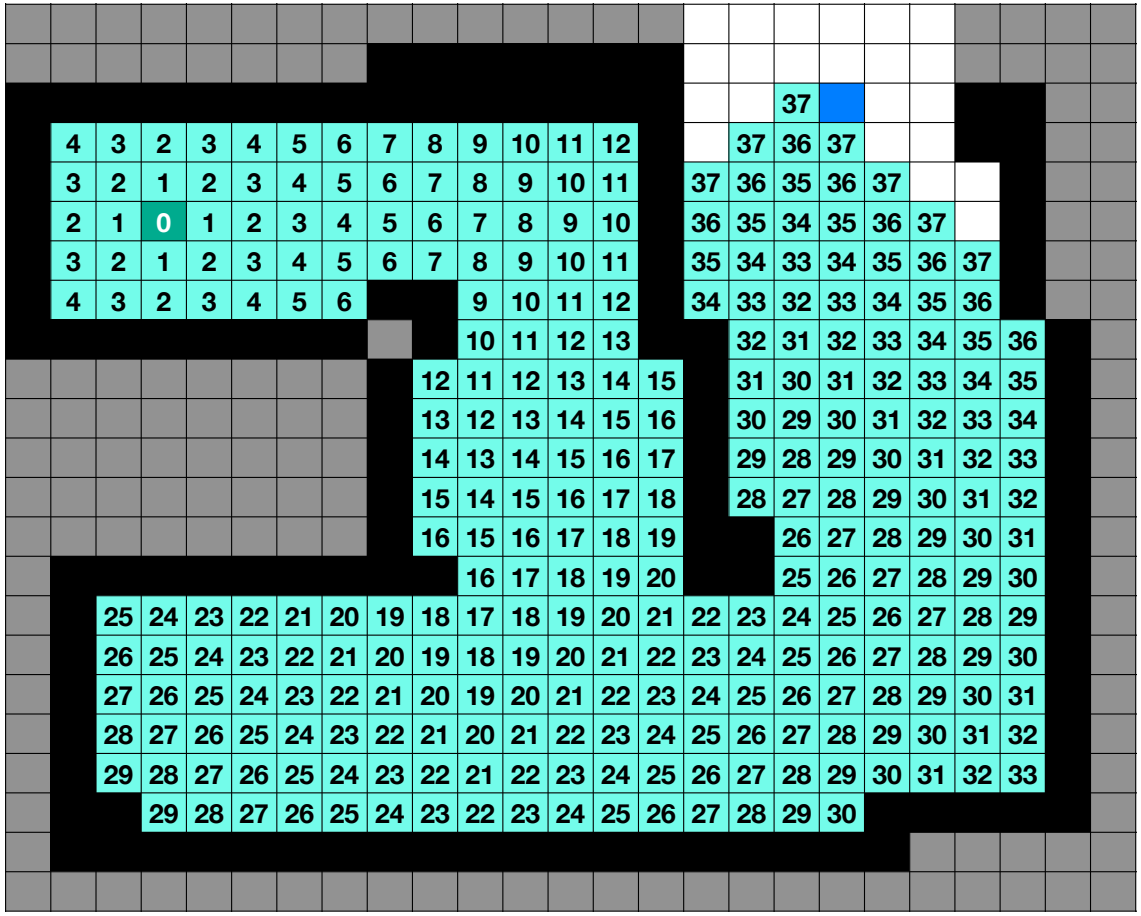
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

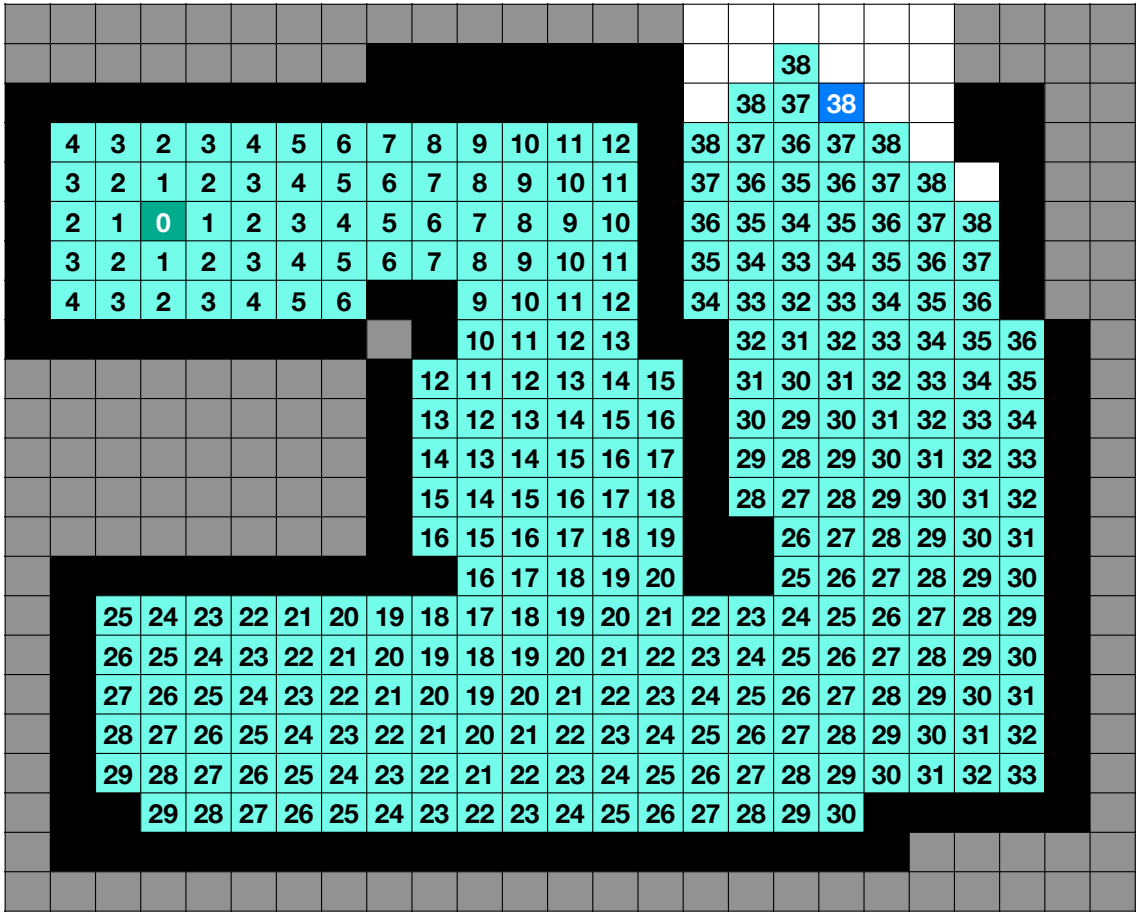
Repeat for next set of neighbors



Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

Repeat for next set of neighbors

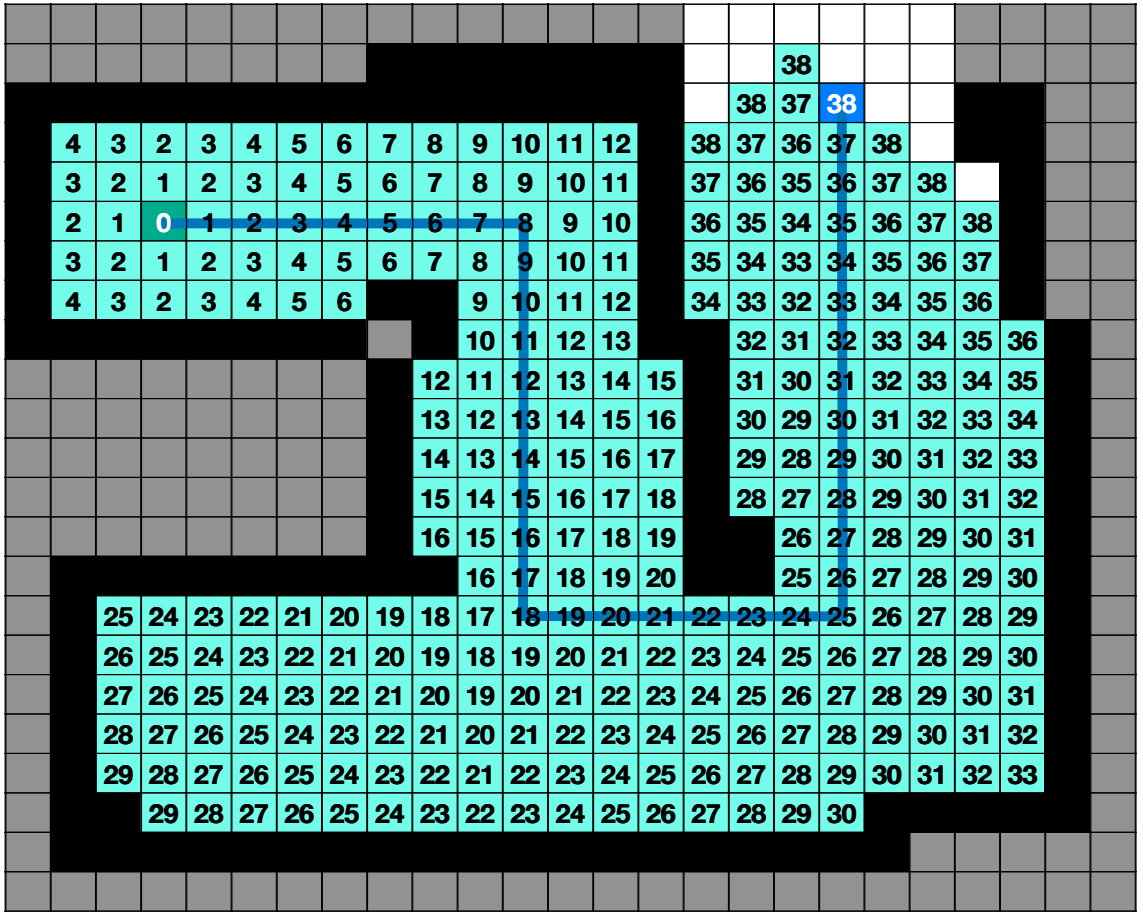


Once goal node reached, perform local search back to start

Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

Repeat for next set of neighbors

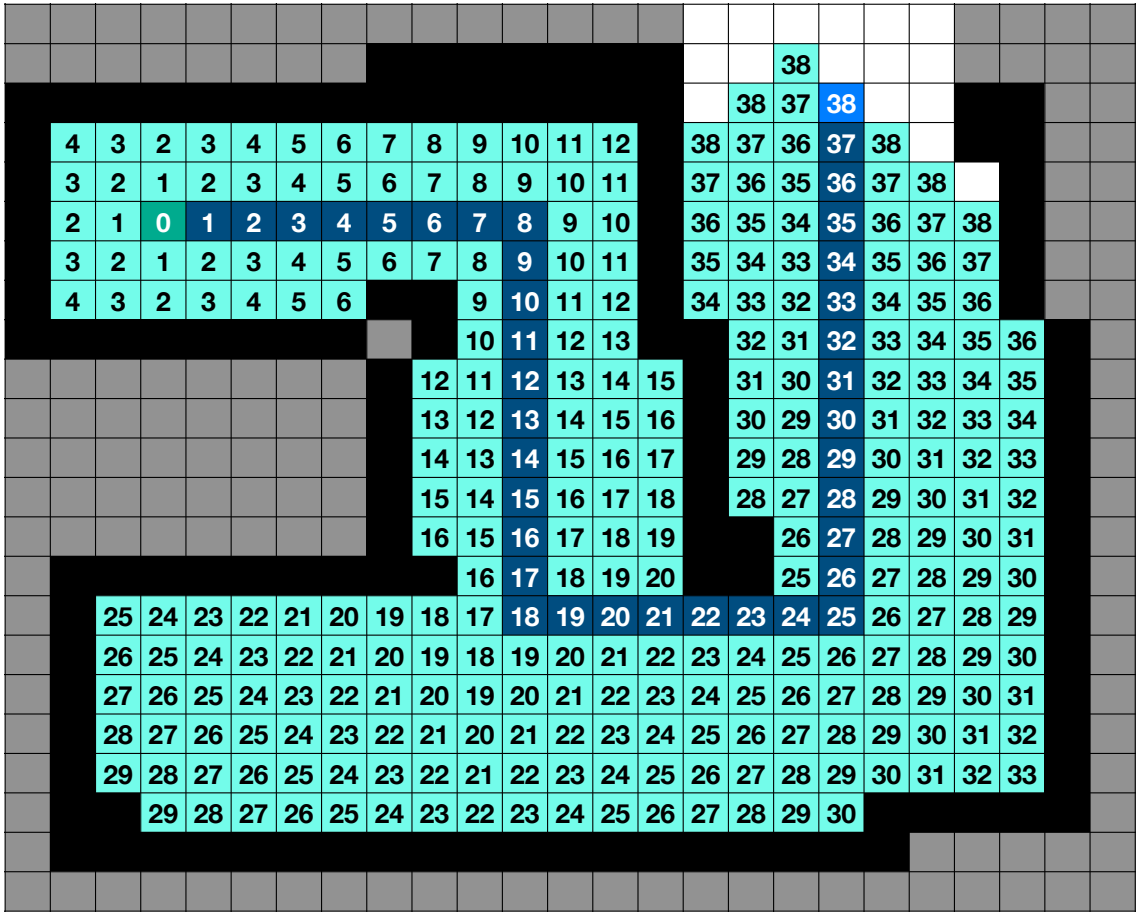


Once goal node reached, perform local search back to start

Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

Repeat for next set of neighbors



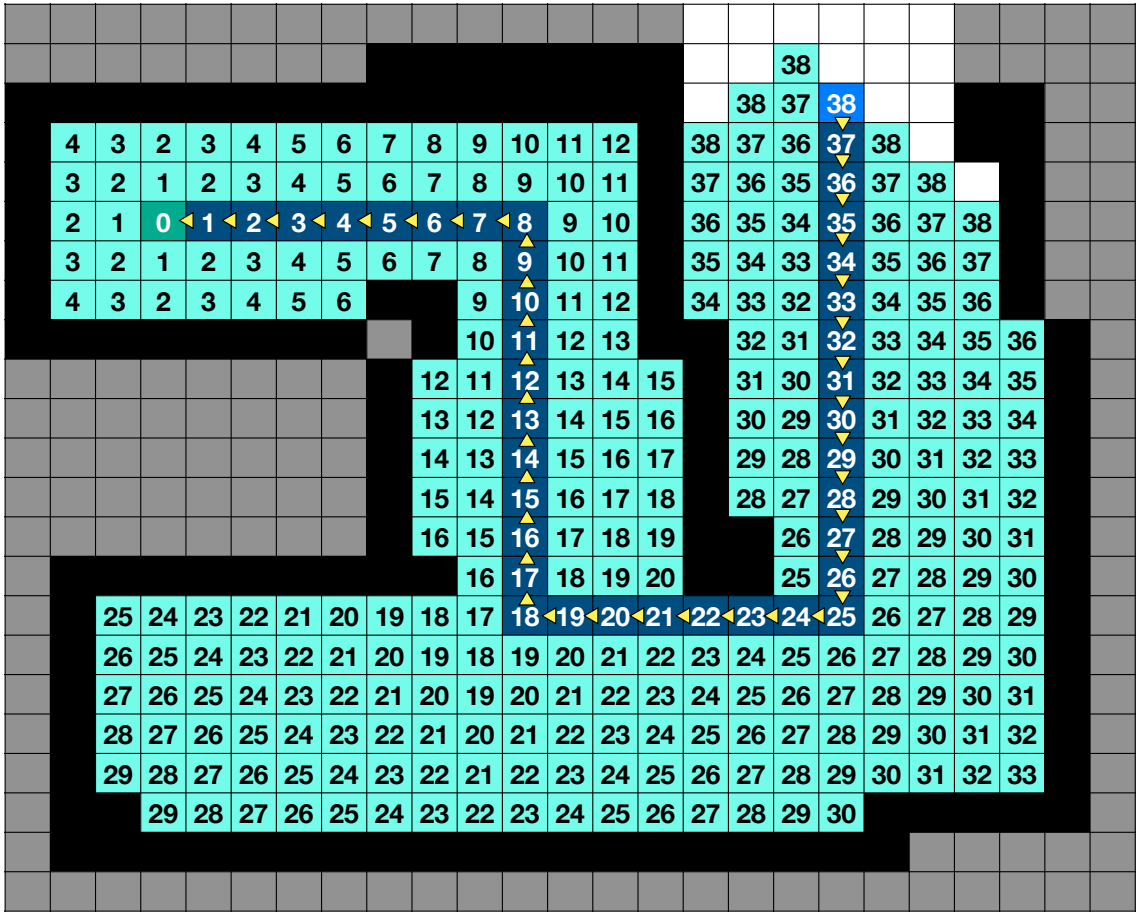
Once goal node reached, perform local search back to start

Assign parents along path in decreasing order

Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

Repeat for next set of neighbors



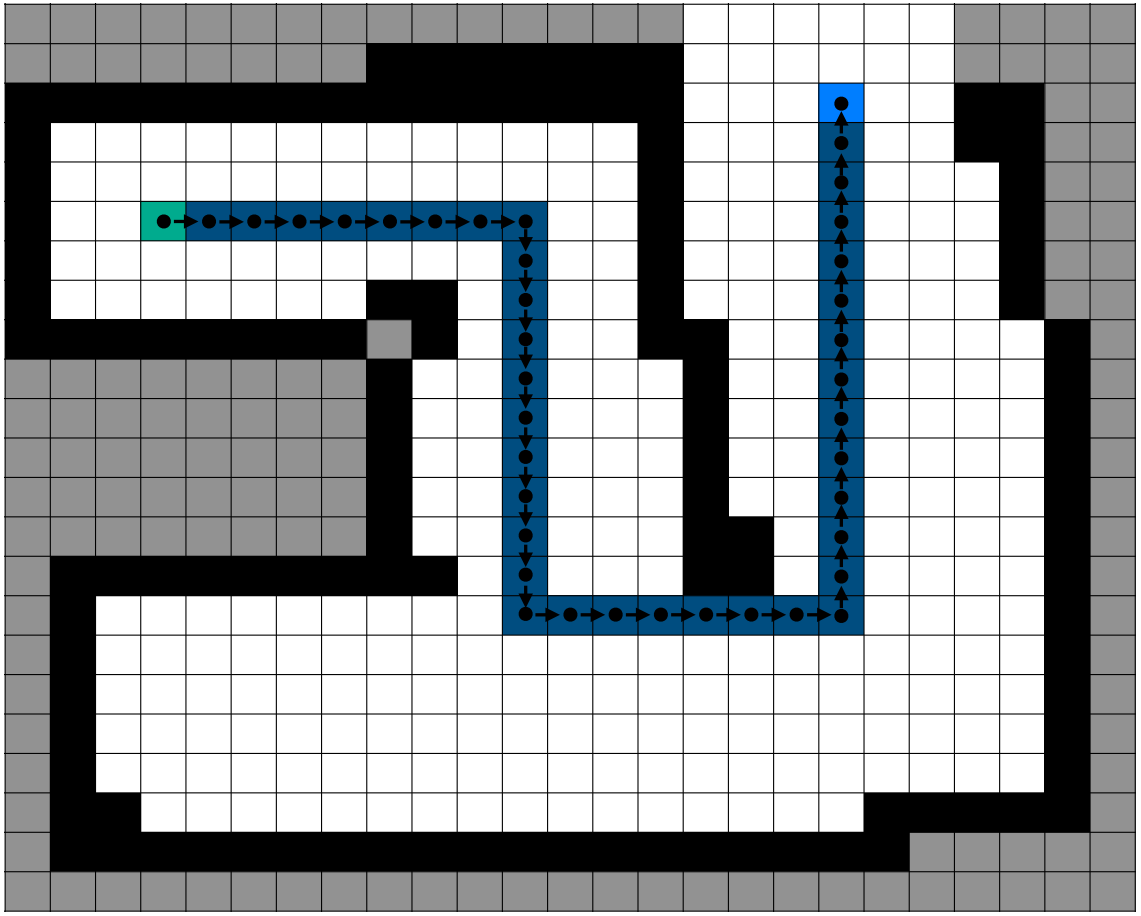
Once goal node reached, perform local search back to start

Assign parents along path in decreasing order

Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

Repeat for next set of neighbors



Once goal node reached, perform local search back to start

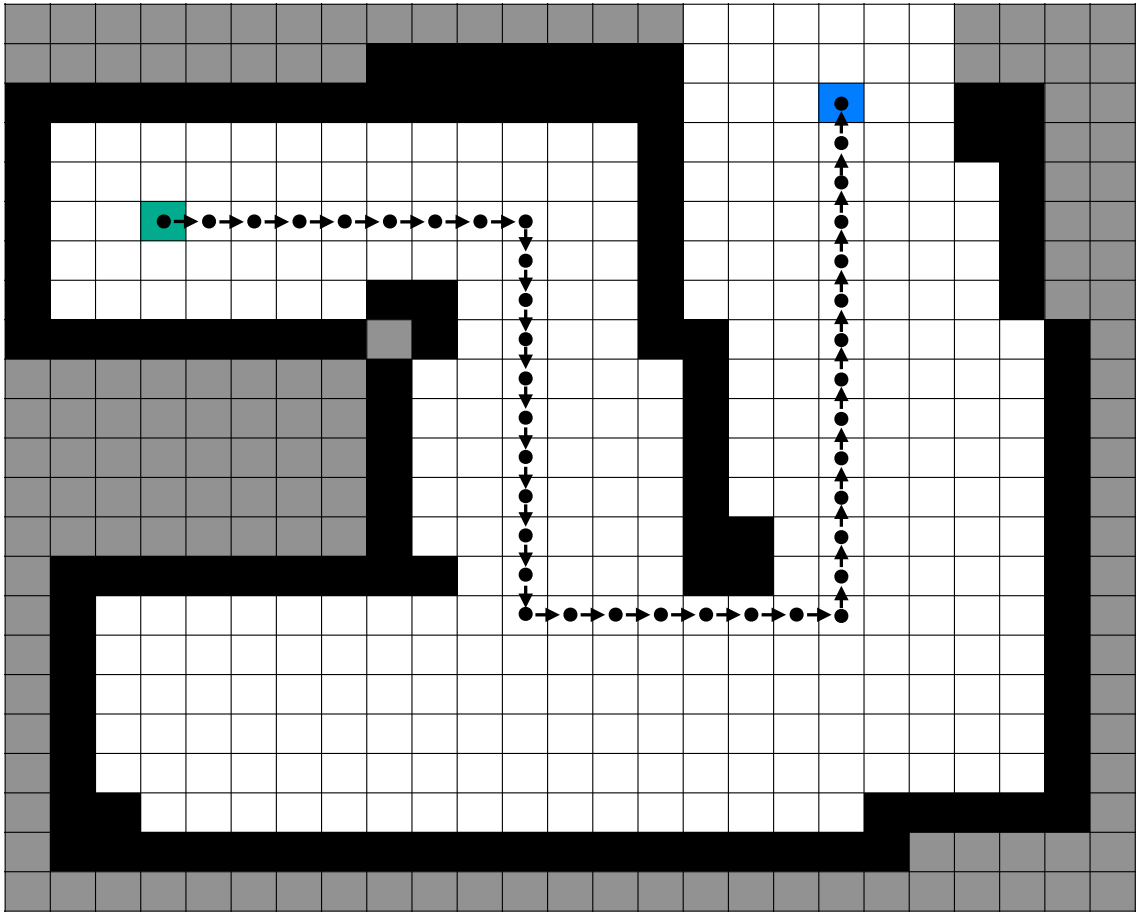
Assign parents along path in decreasing order

Form list of navigation waypoints

Then, visit the neighbors of nodes just visited

Assign each one plus the smallest distance

Repeat for next set of neighbors



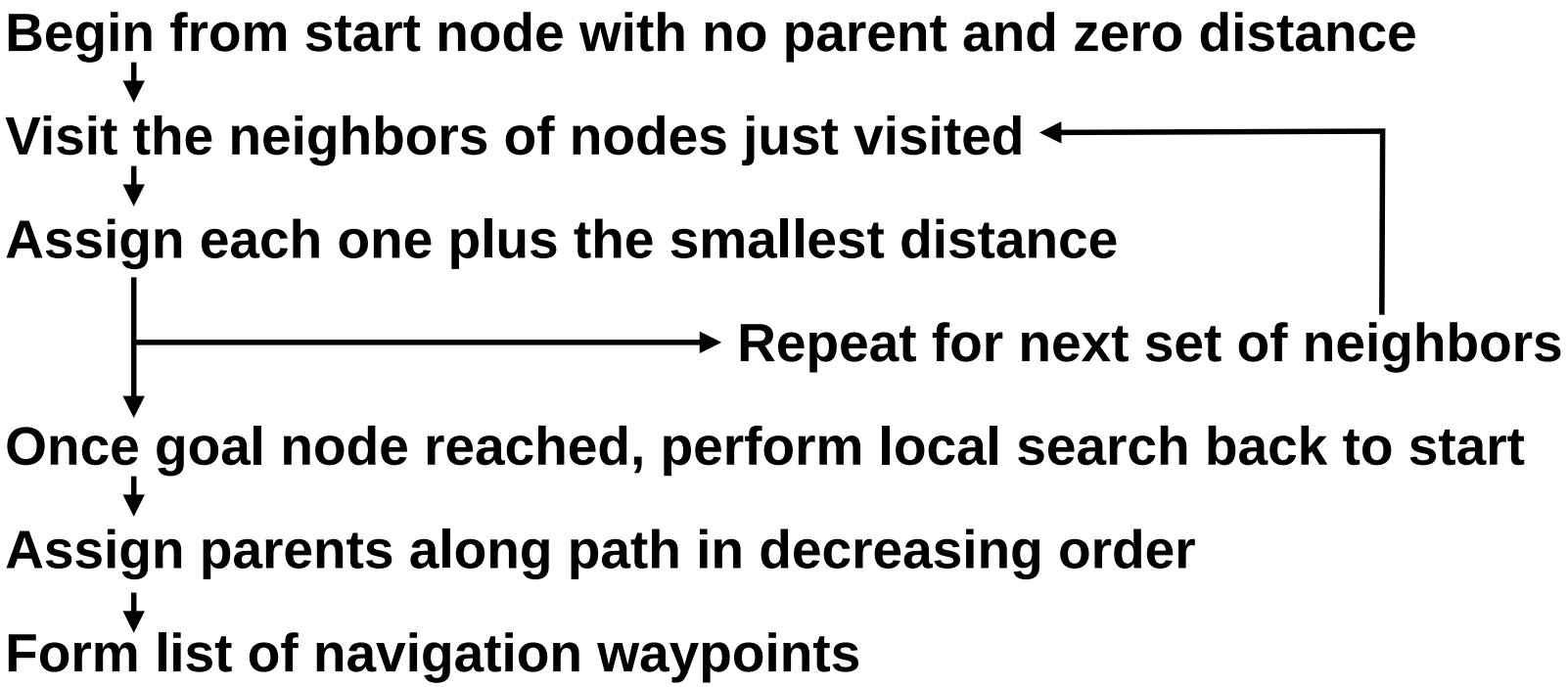
Once goal node reached, perform local search back to start

Assign parents along path in decreasing order

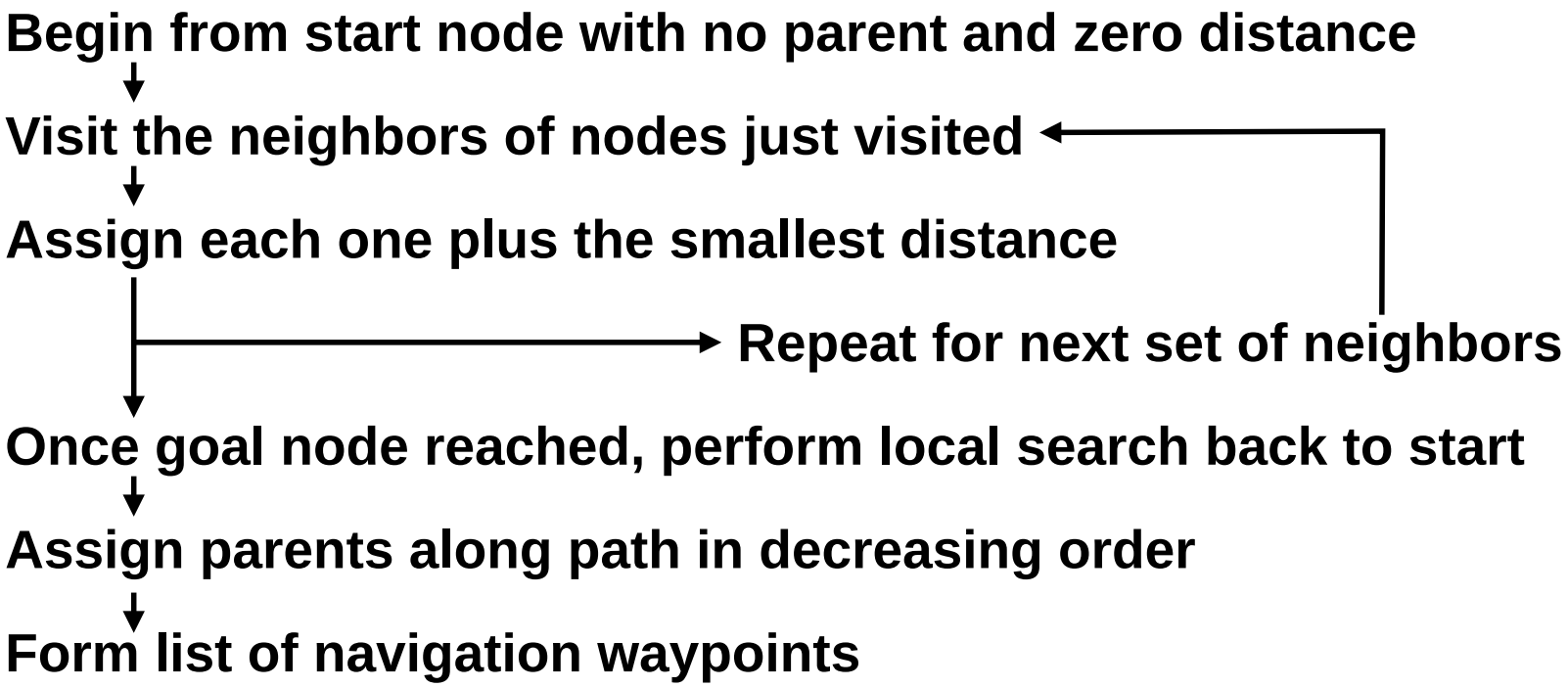
Form list of navigation waypoints

☐ Global search to find routing

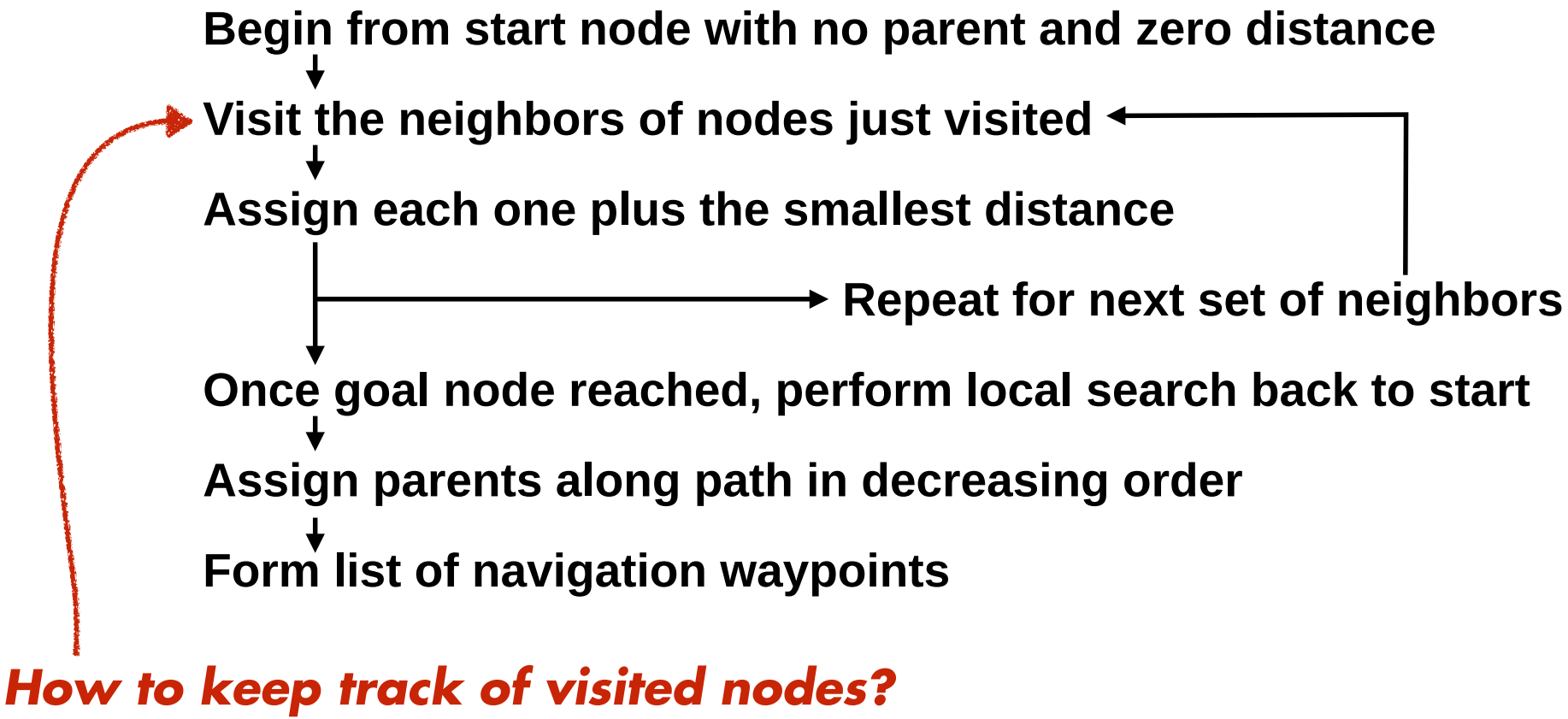
Does this search algorithm have a name ?



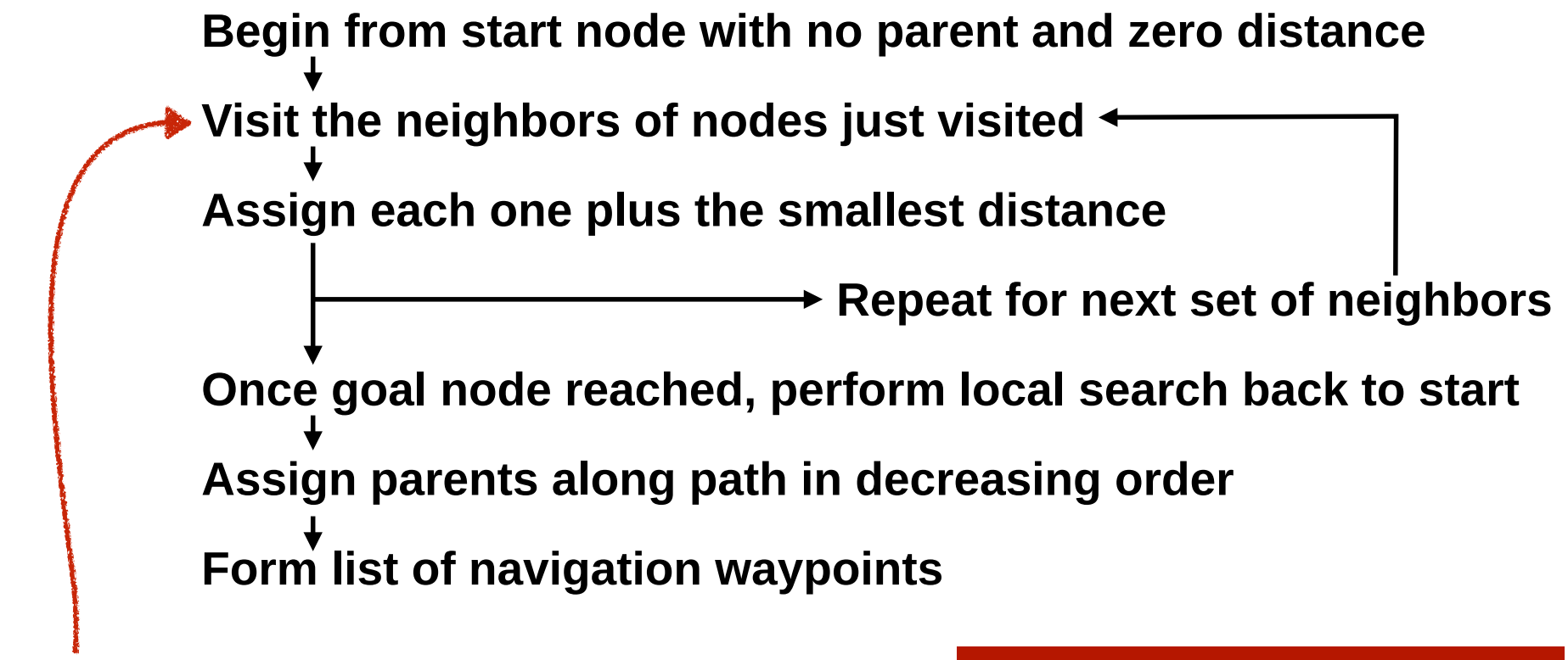
Brushfire Algorithm



Brushfire Algorithm



Brushfire Algorithm

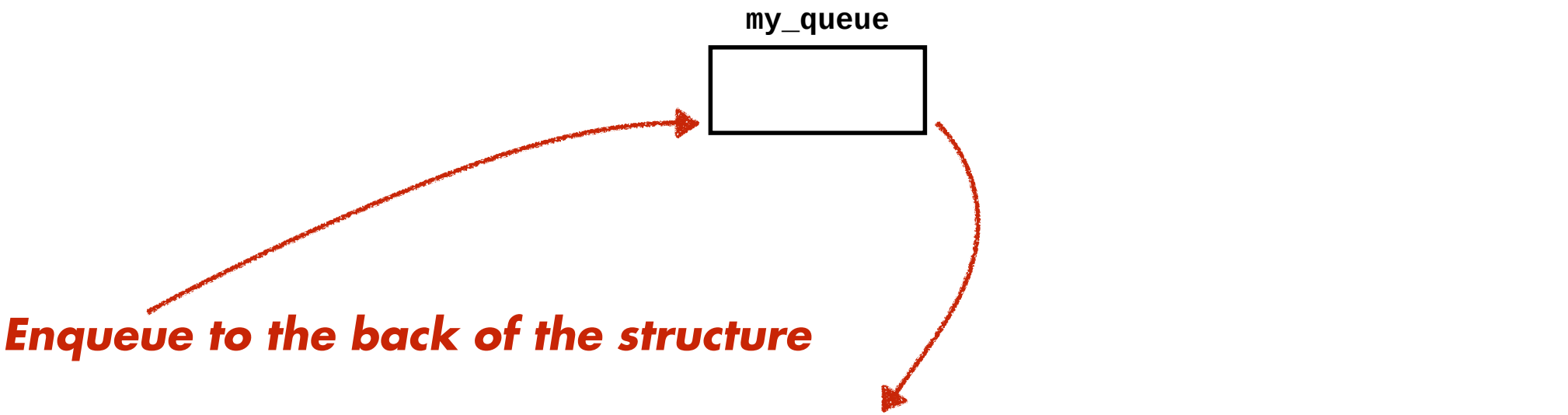


How to keep track of visited nodes?

Queue data structure

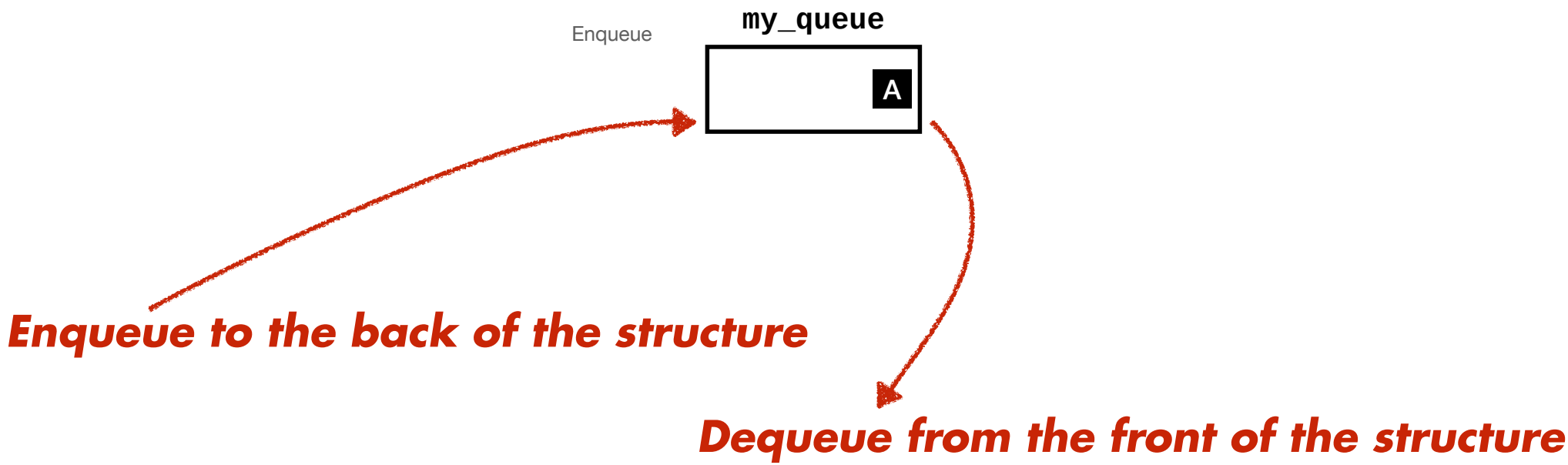
Queue data structure

"First in, first out" (FIFO) data structure



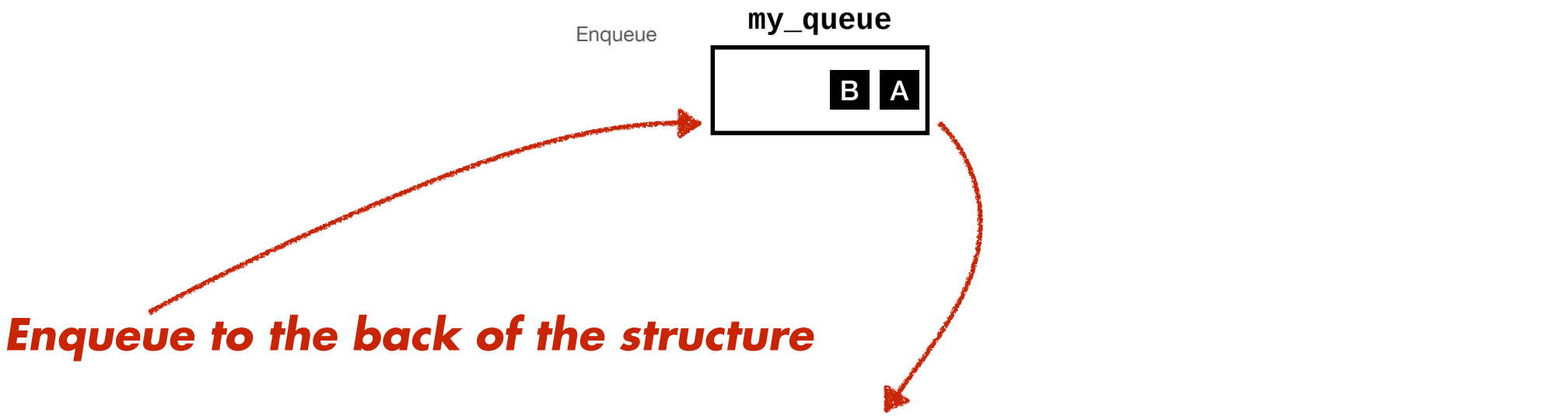
Queue data structure

"First in, first out" (FIFO) data structure



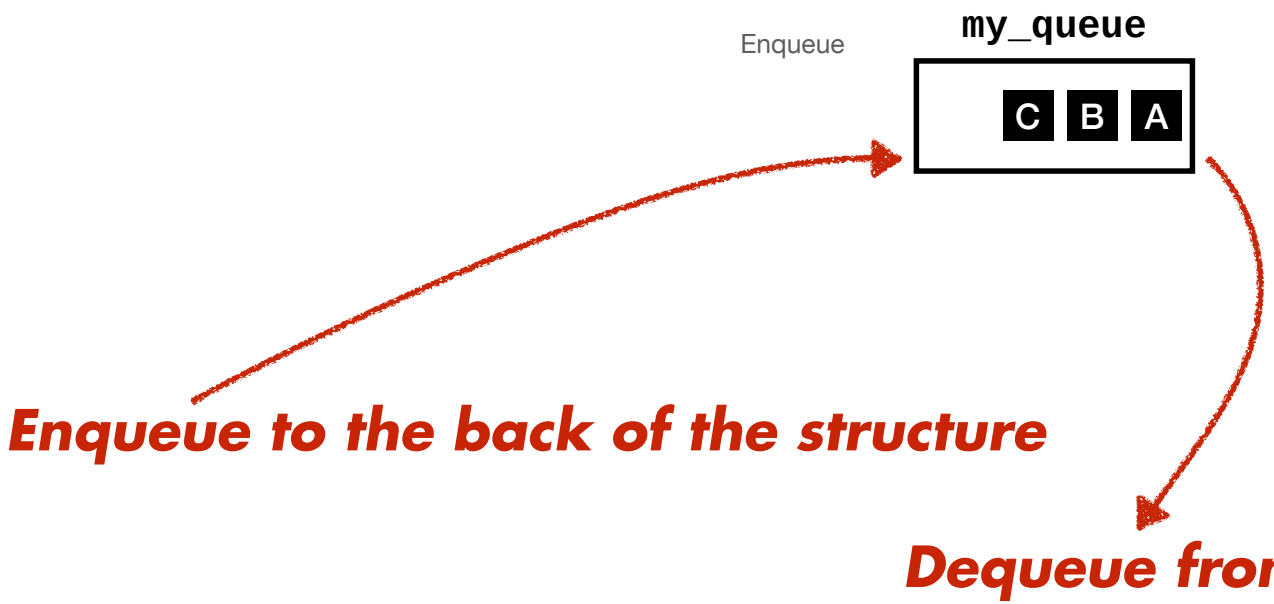
Queue data structure

"First in, first out" (FIFO) data structure



Queue data structure

"First in, first out" (FIFO) data structure

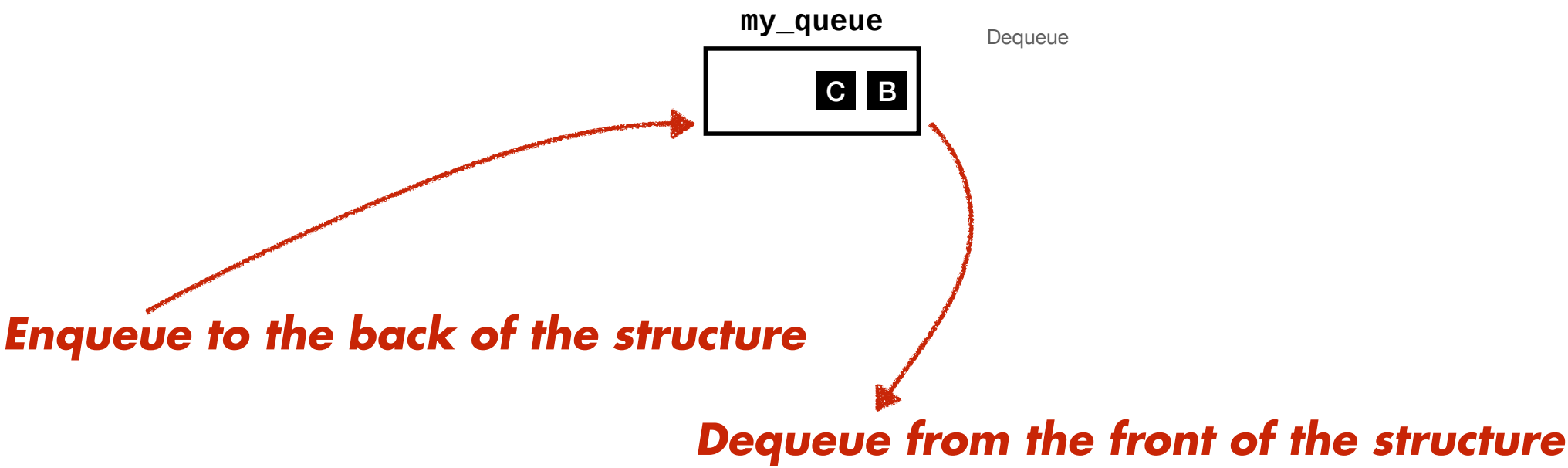


Enqueue to the back of the structure

Dequeue from the front of the structure

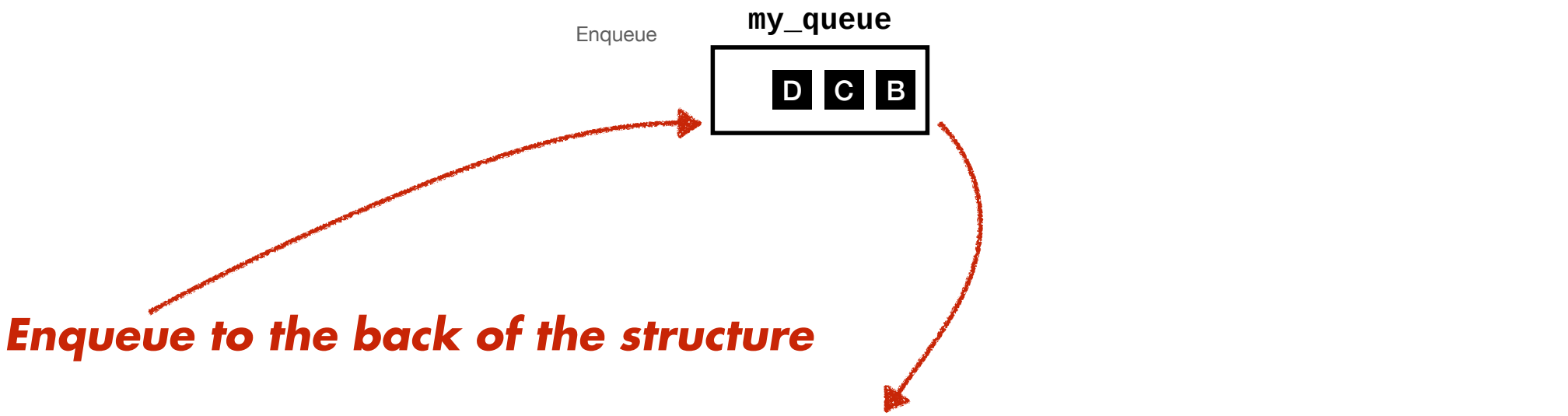
Queue data structure

"First in, first out" (FIFO) data structure



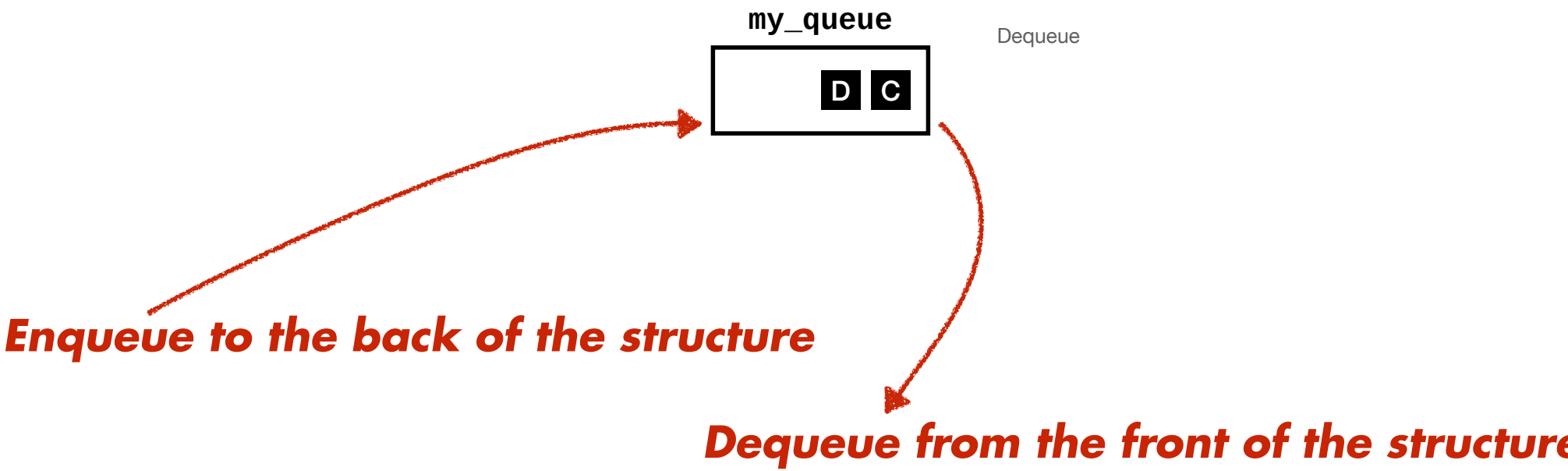
Queue data structure

"First in, first out" (FIFO) data structure



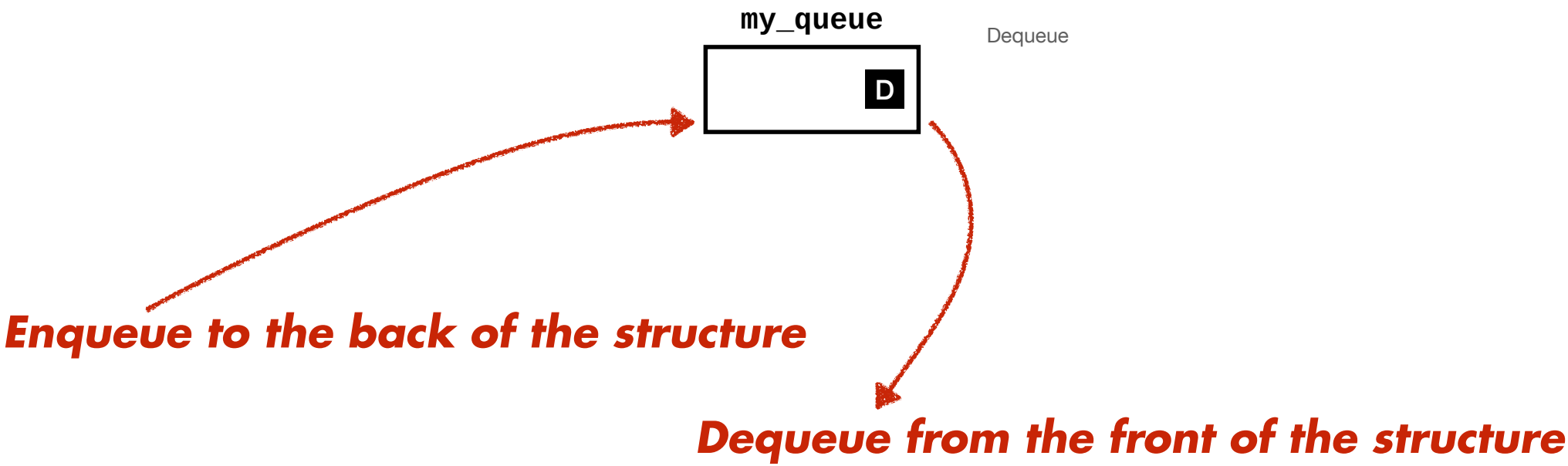
Queue data structure

"First in, first out" (FIFO) data structure



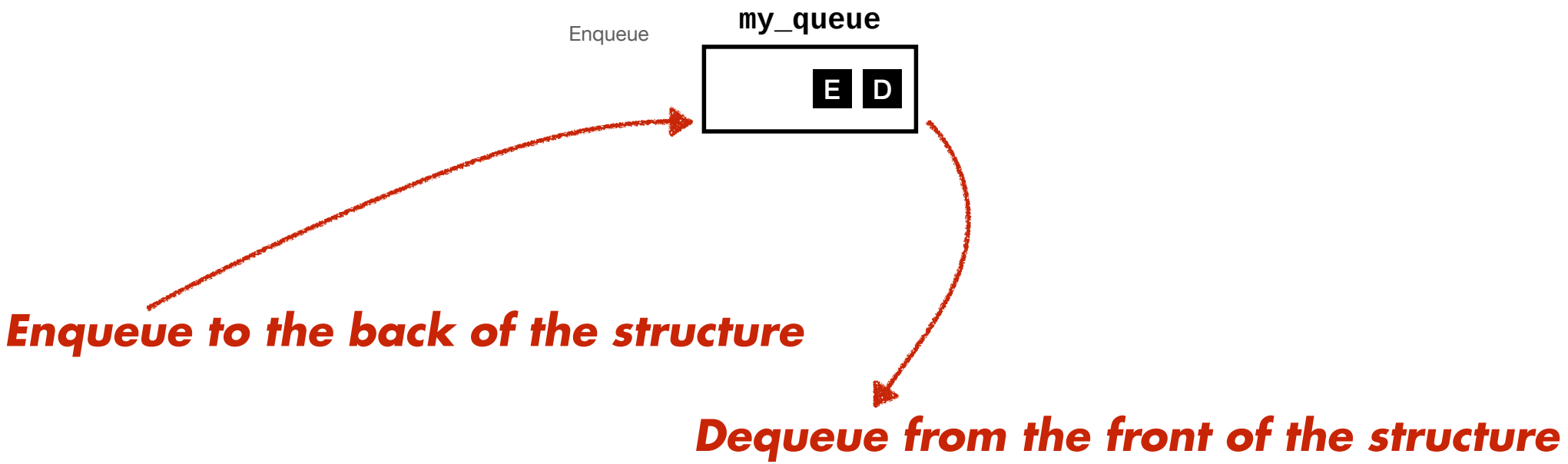
Queue data structure

"First in, first out" (FIFO) data structure



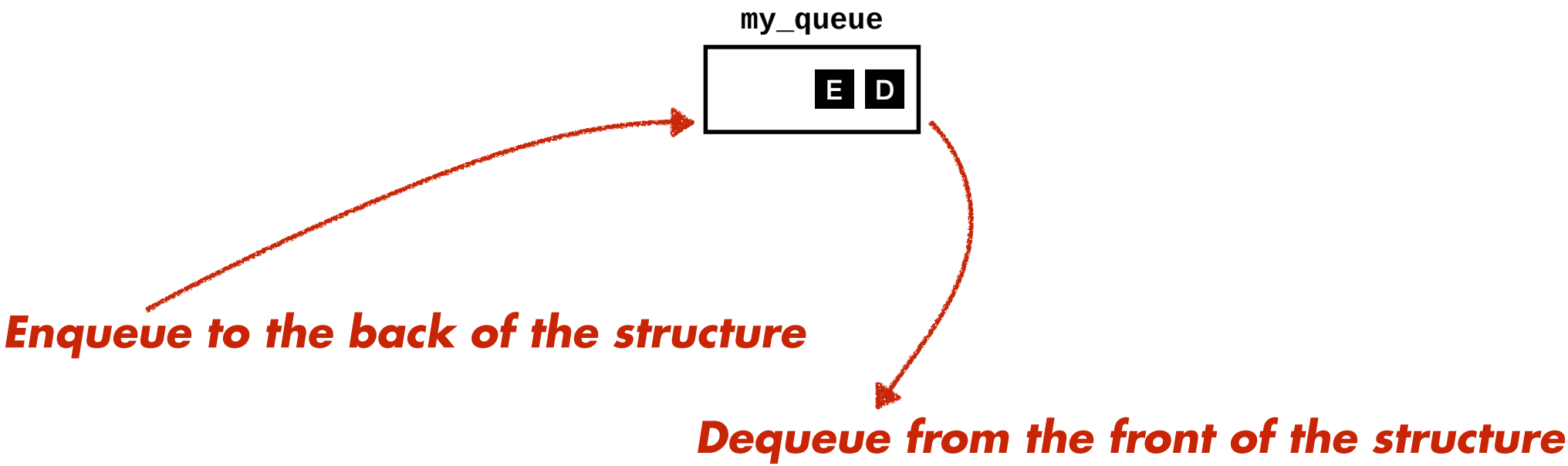
Queue data structure

"First in, first out" (FIFO) data structure



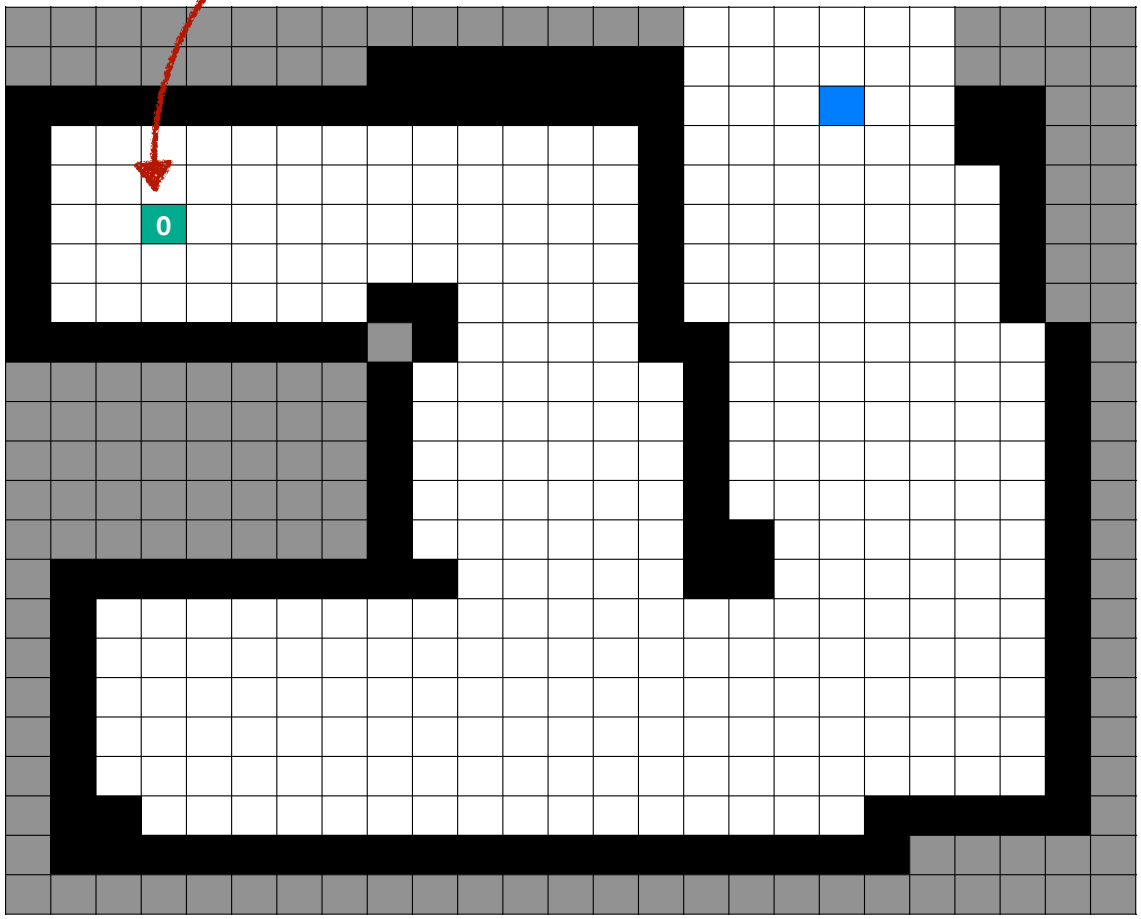
Queue data structure

"First in, first out" (FIFO) data structure



Begin from start node with no parent and zero distance

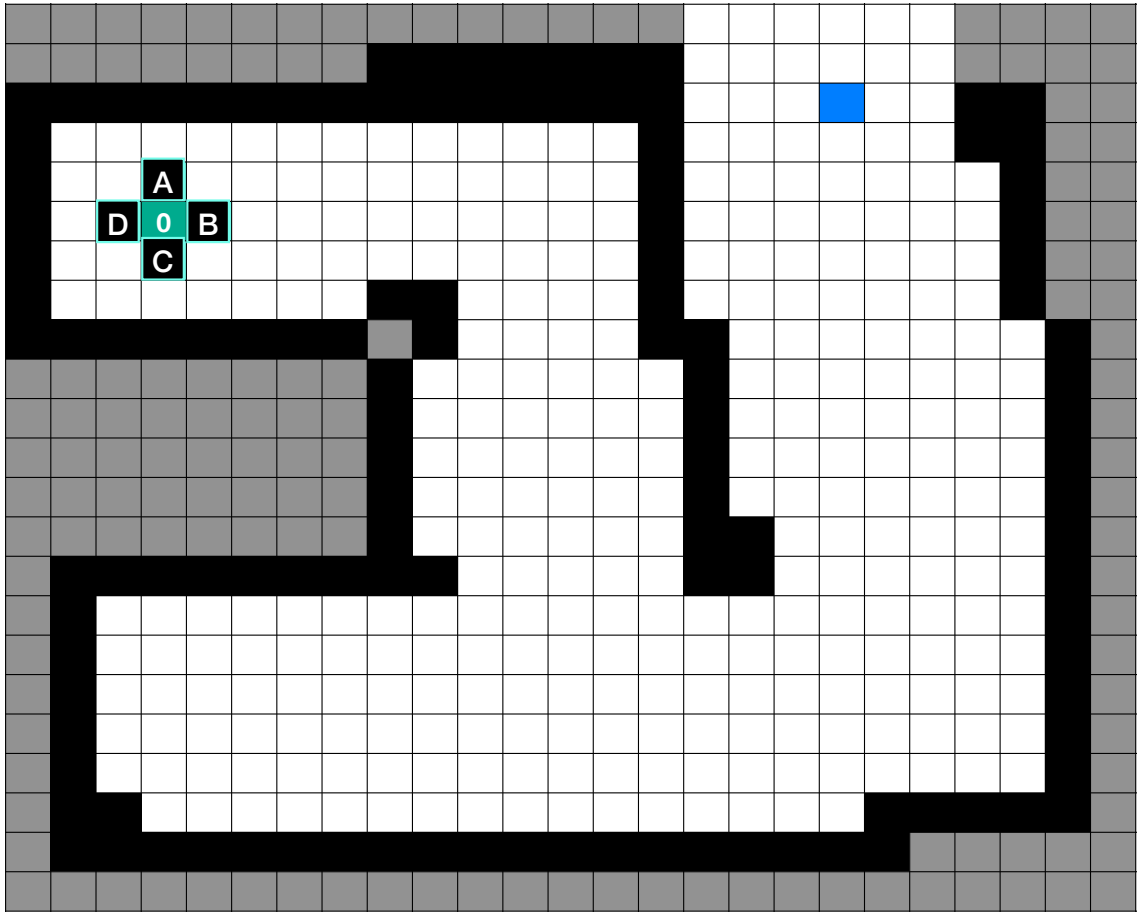
Going back to our example from the root node



`visit_queue`

Begin from start node with no parent and zero distance

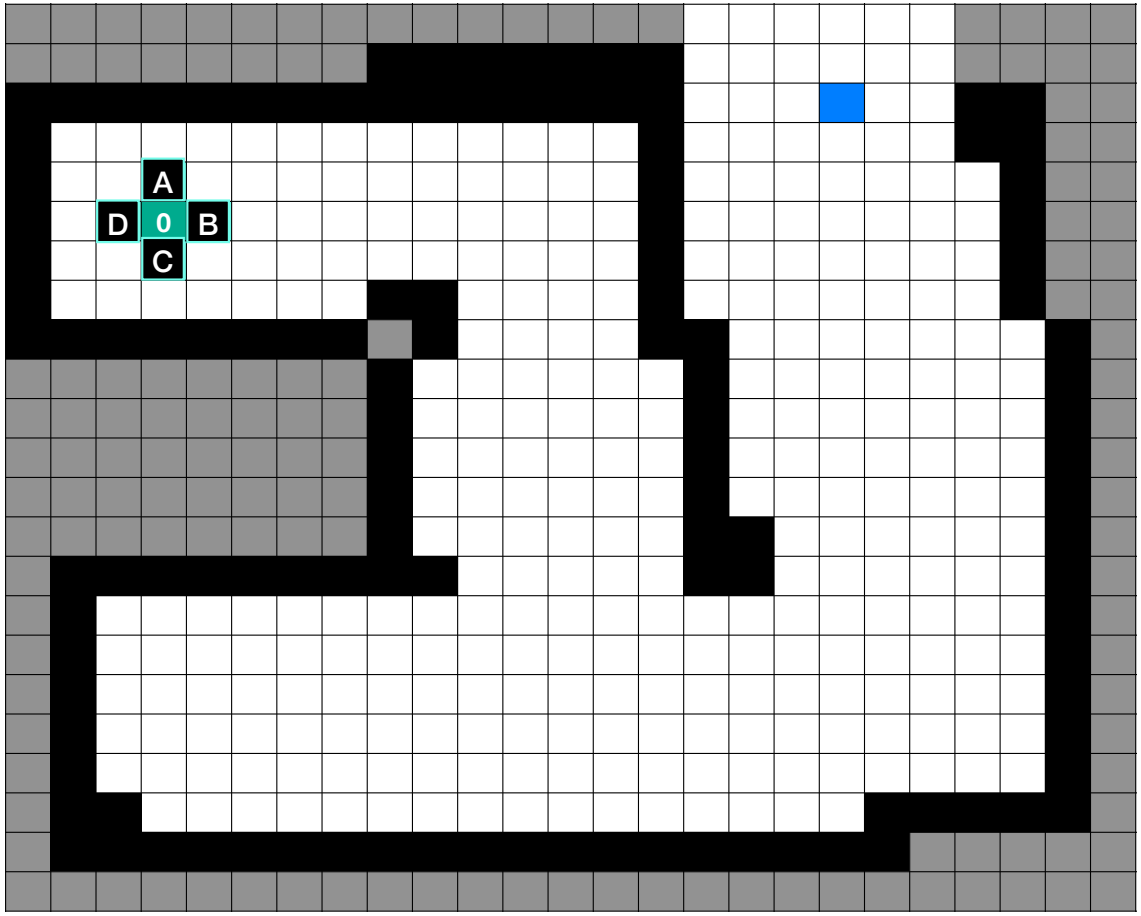
All neighbors of start are "queued"



`visit_queue`

Begin from start node with no parent and zero distance

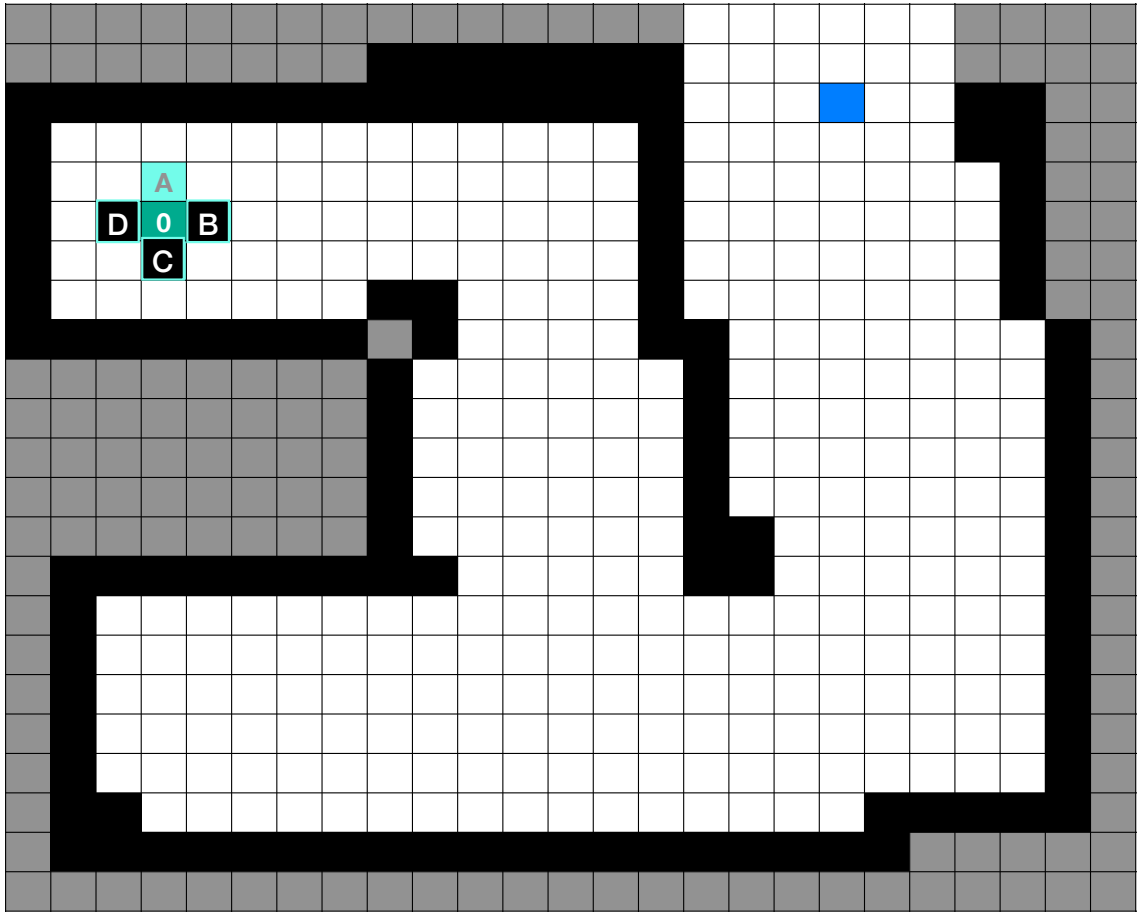
All neighbors of start are "queued"



`visit_queue`

Begin from start node with no parent and zero distance

All neighbors of start are "queued"

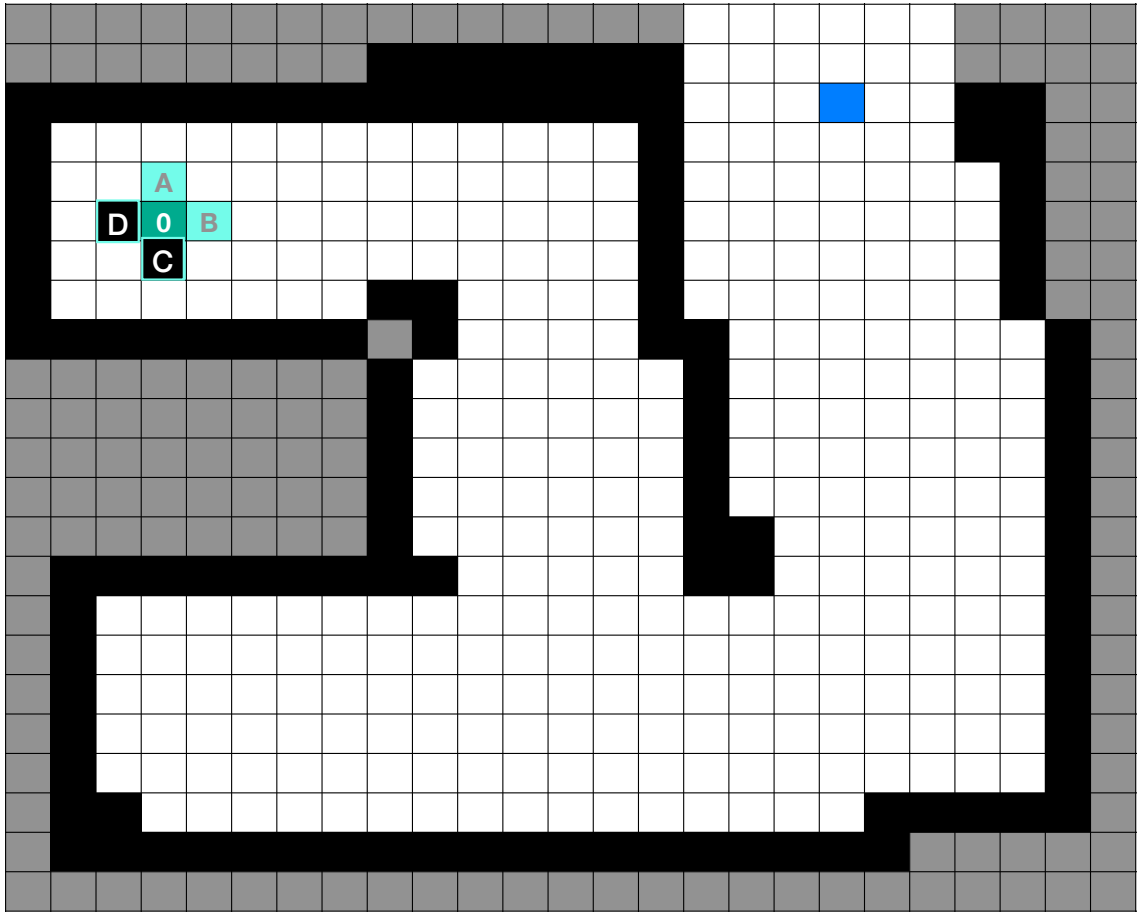


visit_queue

Enqueue

Begin from start node with no parent and zero distance

All neighbors of start are "queued"

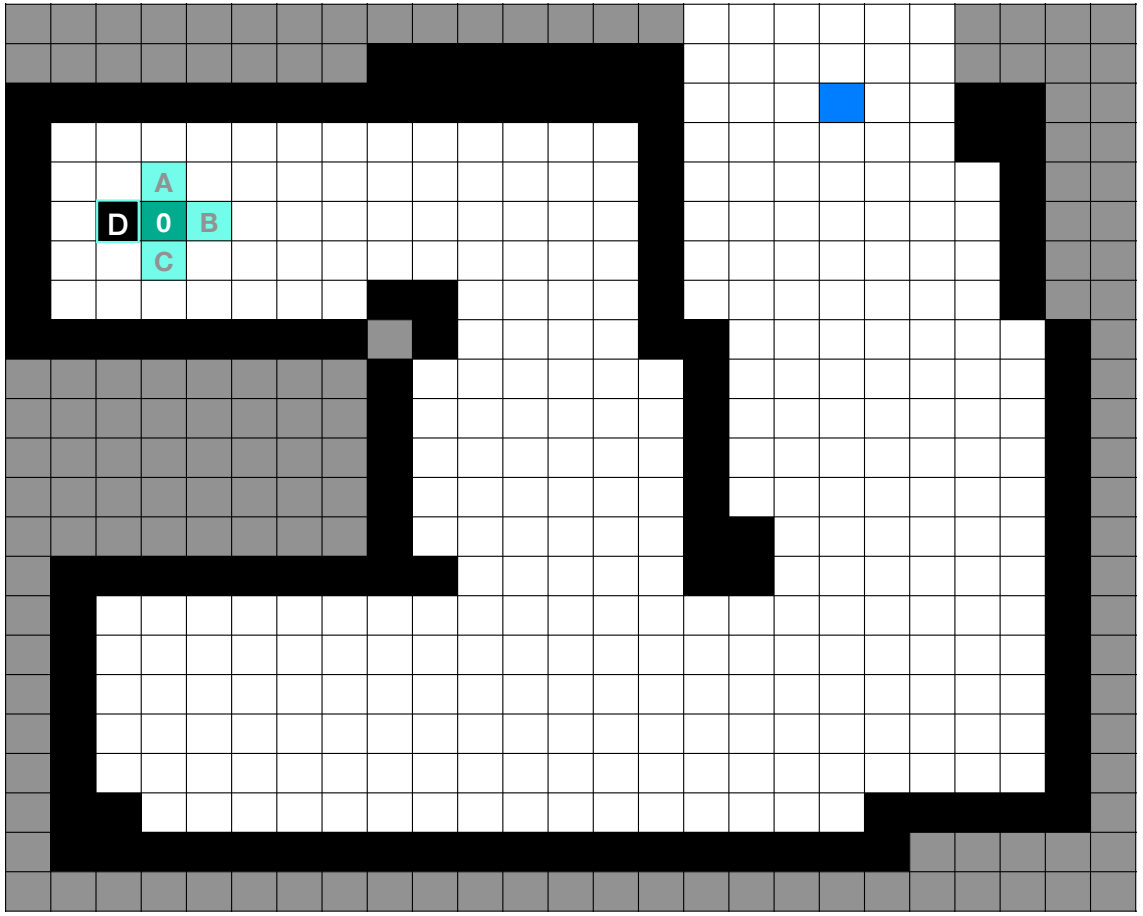


`visit_queue`

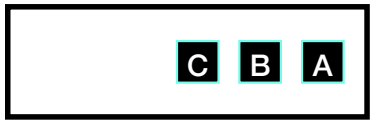
Enqueue

Begin from start node with no parent and zero distance

All neighbors of start are "queued"



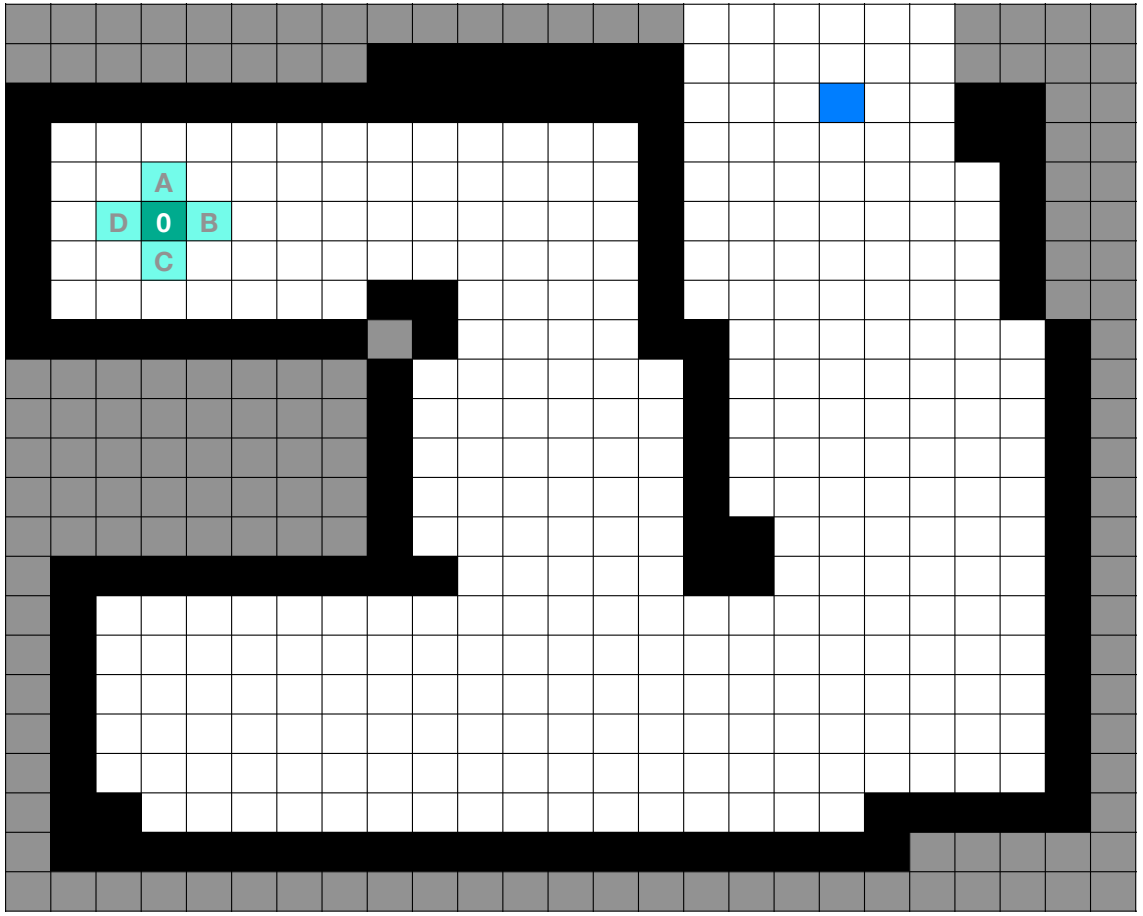
`visit_queue`



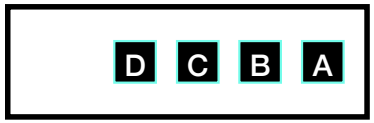
Enqueue

Begin from start node with no parent and zero distance

All neighbors of start are "queued"



`visit_queue`

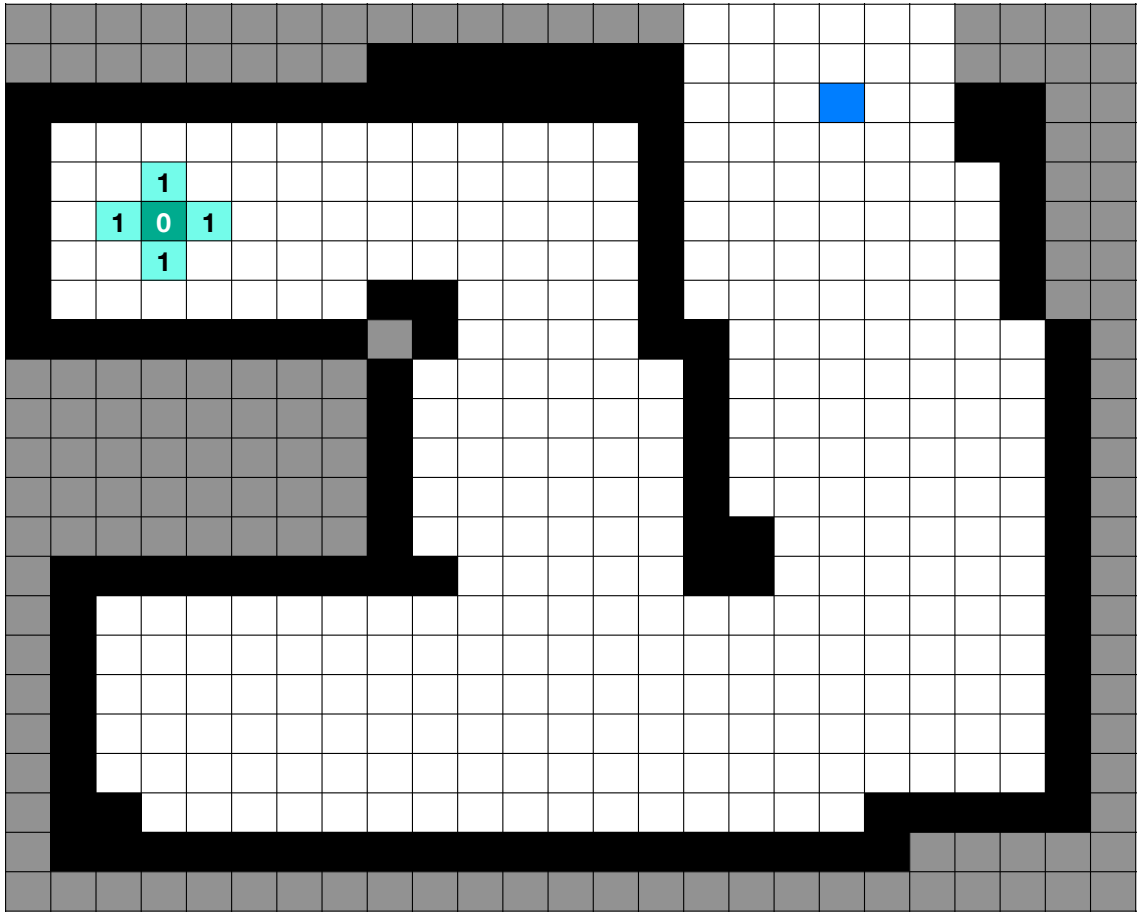


Enqueue

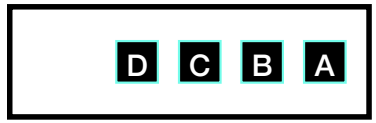
Begin from start node with no parent and zero distance

All neighbors of start are "queued"

and assigned distance as one plus start node distance



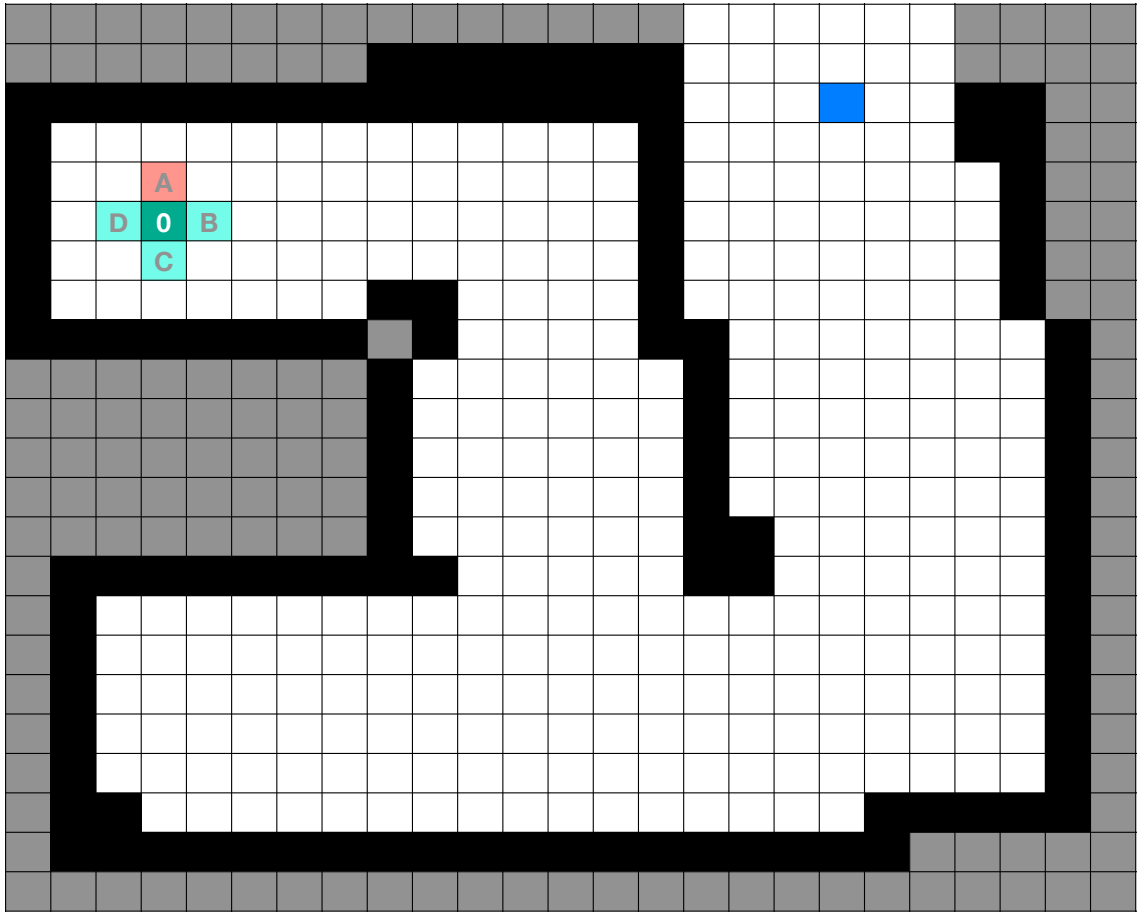
`visit_queue`



Repeat for next node in the queue

All neighbors of current node are "queued"

and assigned distance as one plus current node distance



`visit_queue`

Dequeue

A

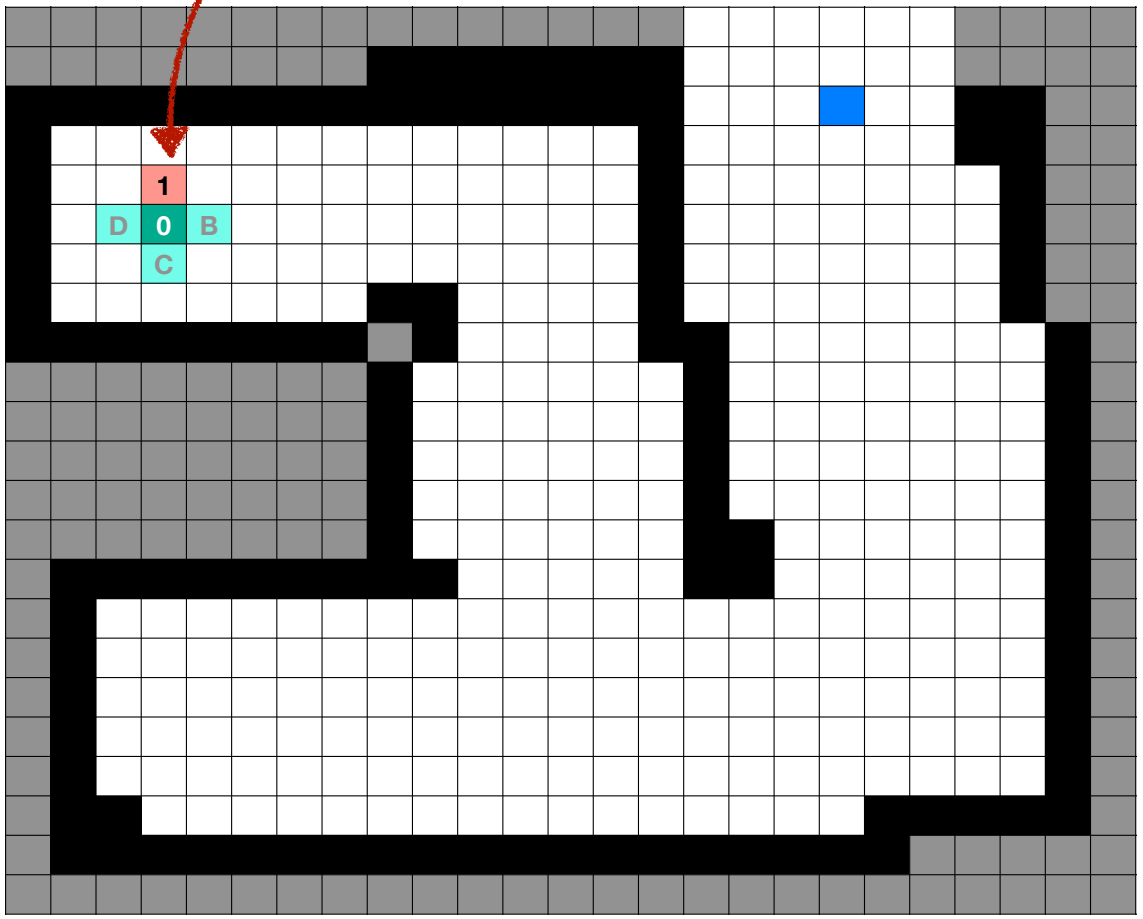
`current_node`

Path distance at current node was already assigned in previous iteration

Repeat for next node in the queue

All neighbors of current node are "queued"

and assigned distance as one plus current node distance



`visit_queue`

D C B

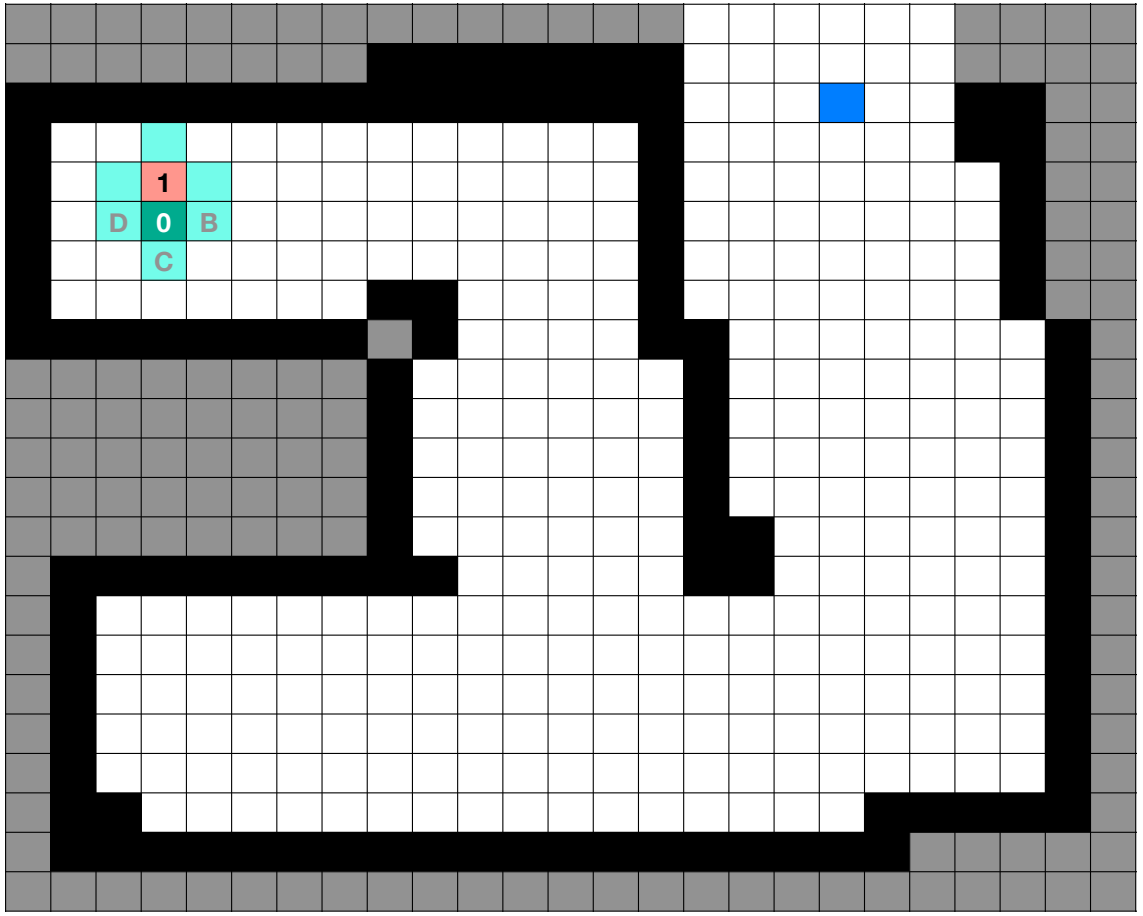
A

`current_node`

Repeat for next node in the queue

All neighbors of current node are "queued"

and assigned distance as one plus current node distance



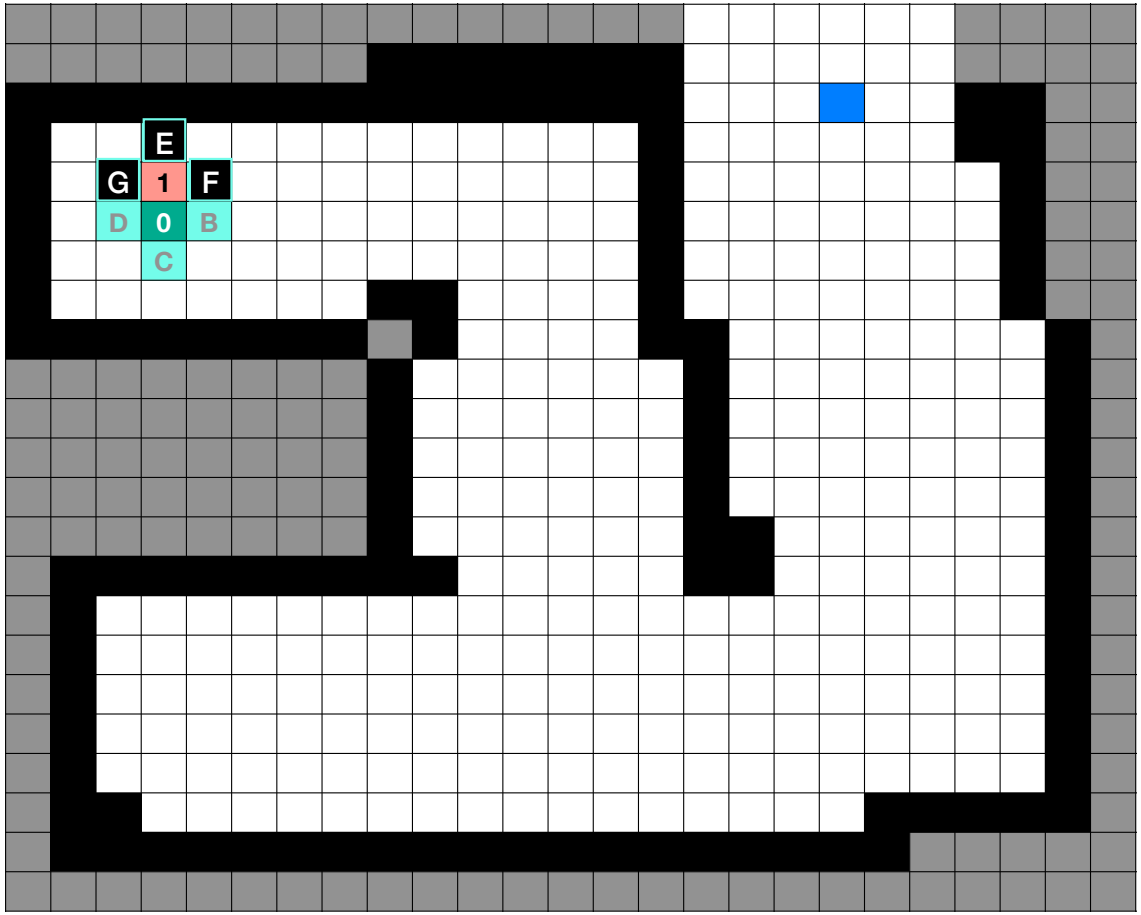
`visit_queue`

`current_node`

Repeat for next node in the queue

All neighbors of current node are "queued"

and assigned distance as one plus current node distance



`visit_queue`

D C B

A

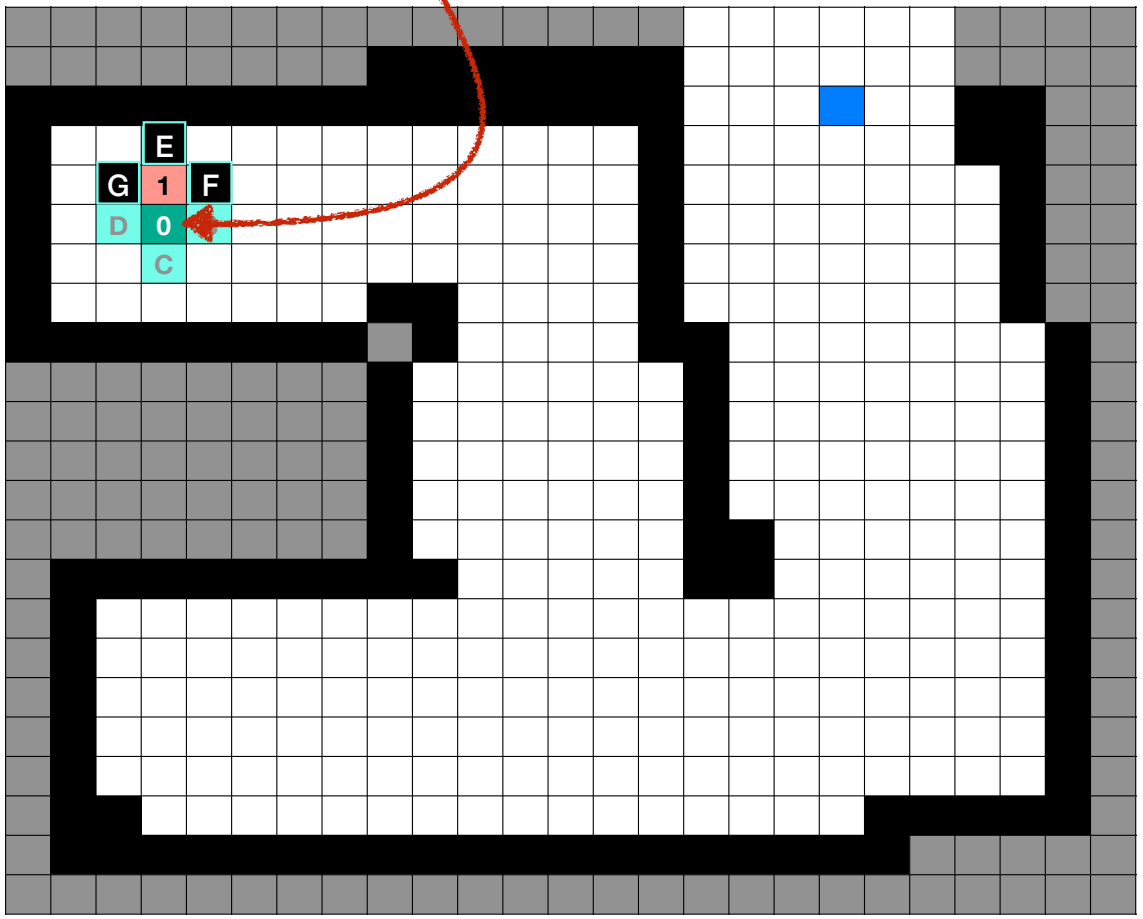
`current_node`

Repeat for next node in the queue

All neighbors of current node are "queued"

and assigned distance as one plus current node distance

Do not revisit or re-enqueue nodes



`visit_queue`

D C B

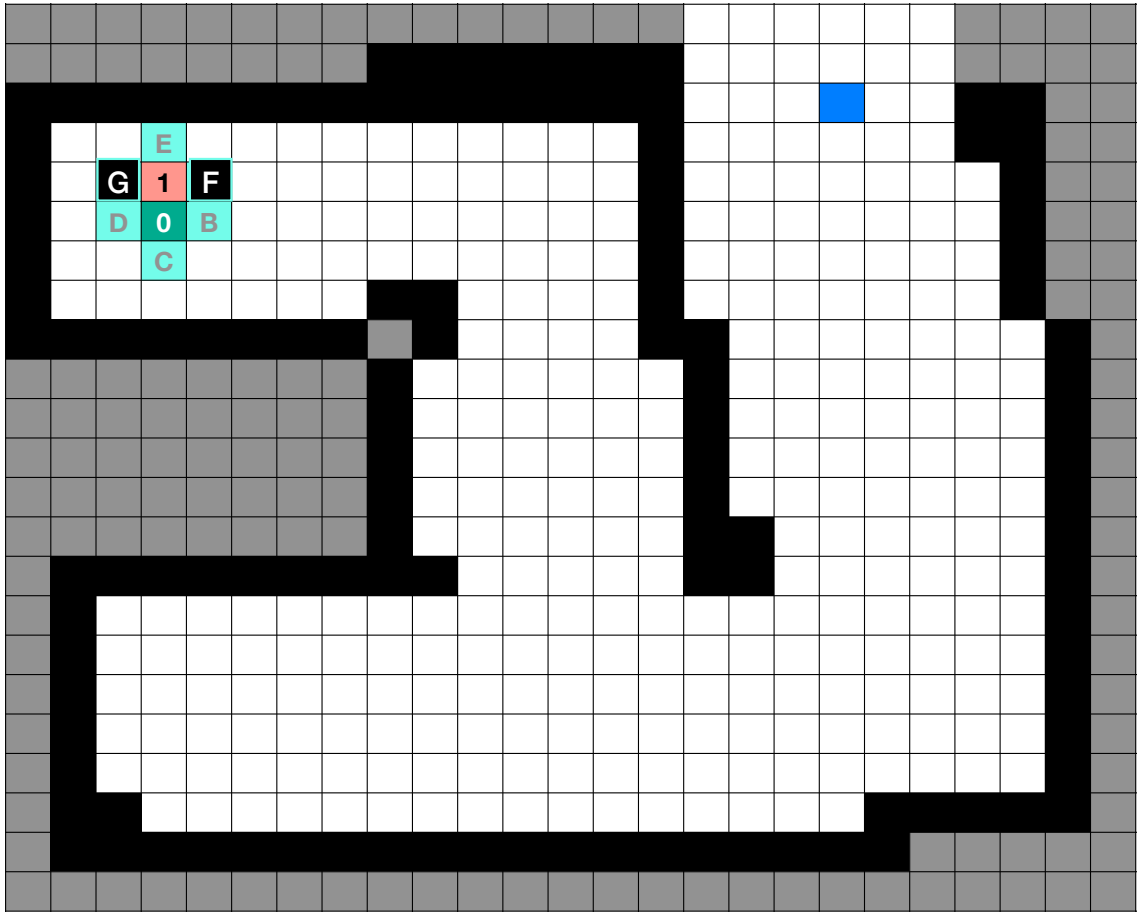
A

`current_node`

Repeat for next node in the queue

All neighbors of current node are "queued"

and assigned distance as one plus current node distance



`visit_queue`

Enqueue

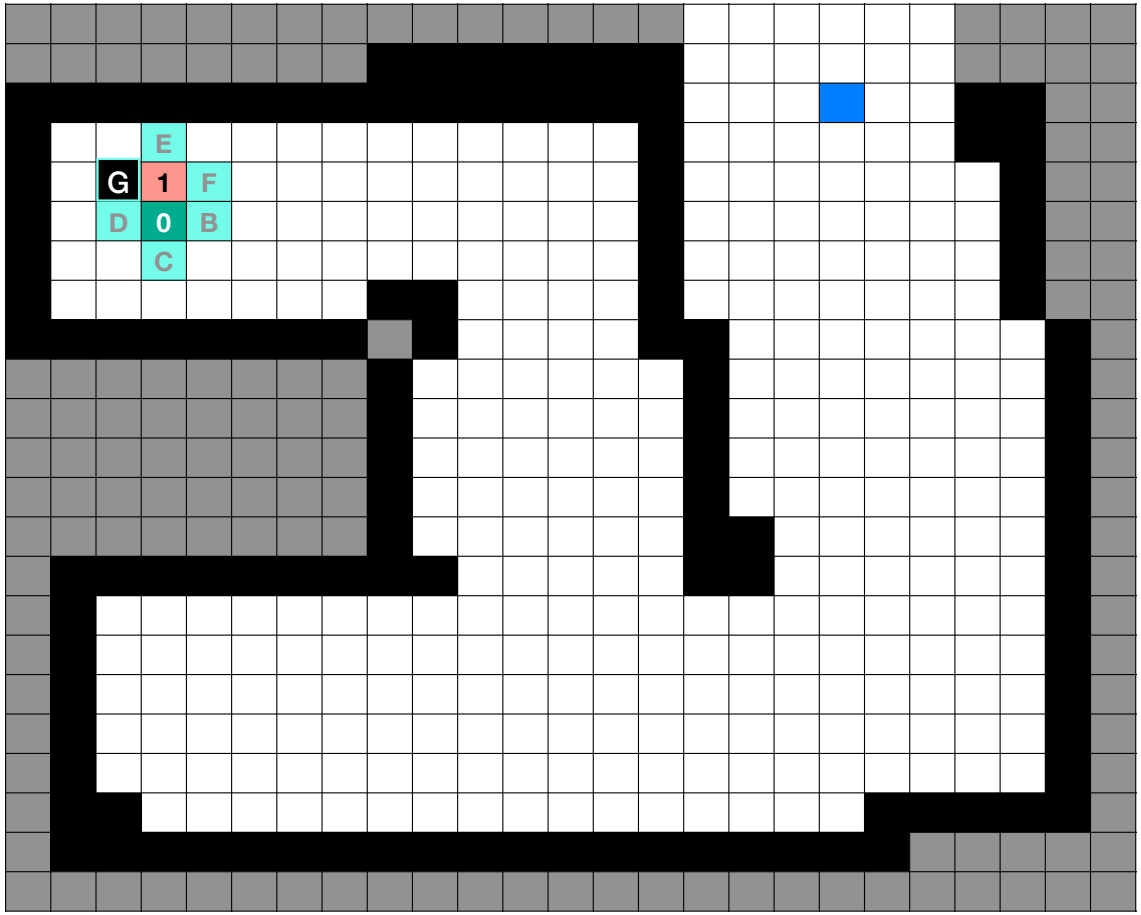
A

`current_node`

Repeat for next node in the queue

All neighbors of current node are "queued"

and assigned distance as one plus current node distance



`visit_queue`

Enqueue

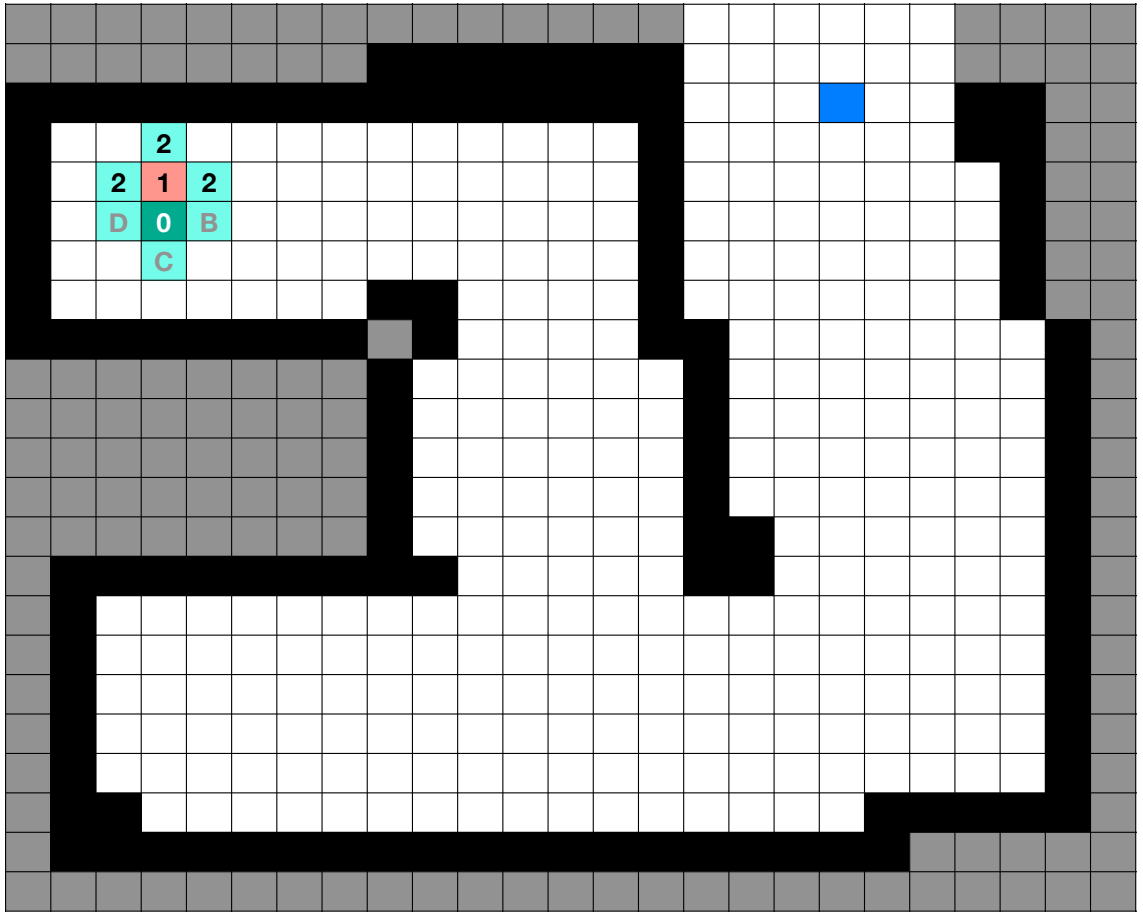
A

`current_node`

Repeat for next node in the queue

All neighbors of current node are "queued"

and assigned distance as one plus current node distance



`visit_queue`

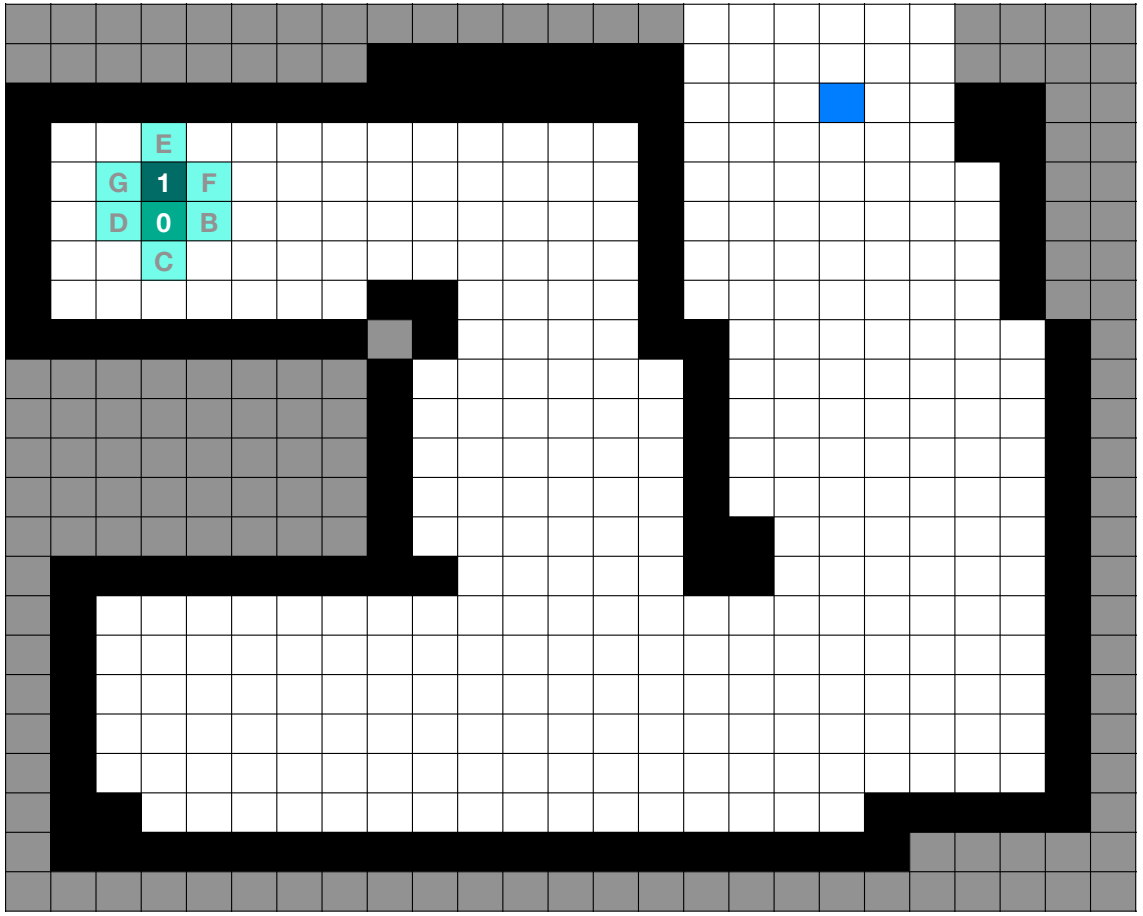
Enqueue

`current_node`

Repeat for next node in the queue

All neighbors of current node are "queued"

and assigned distance as one plus current node distance



`visit_queue`

G	F	E	D	C	B

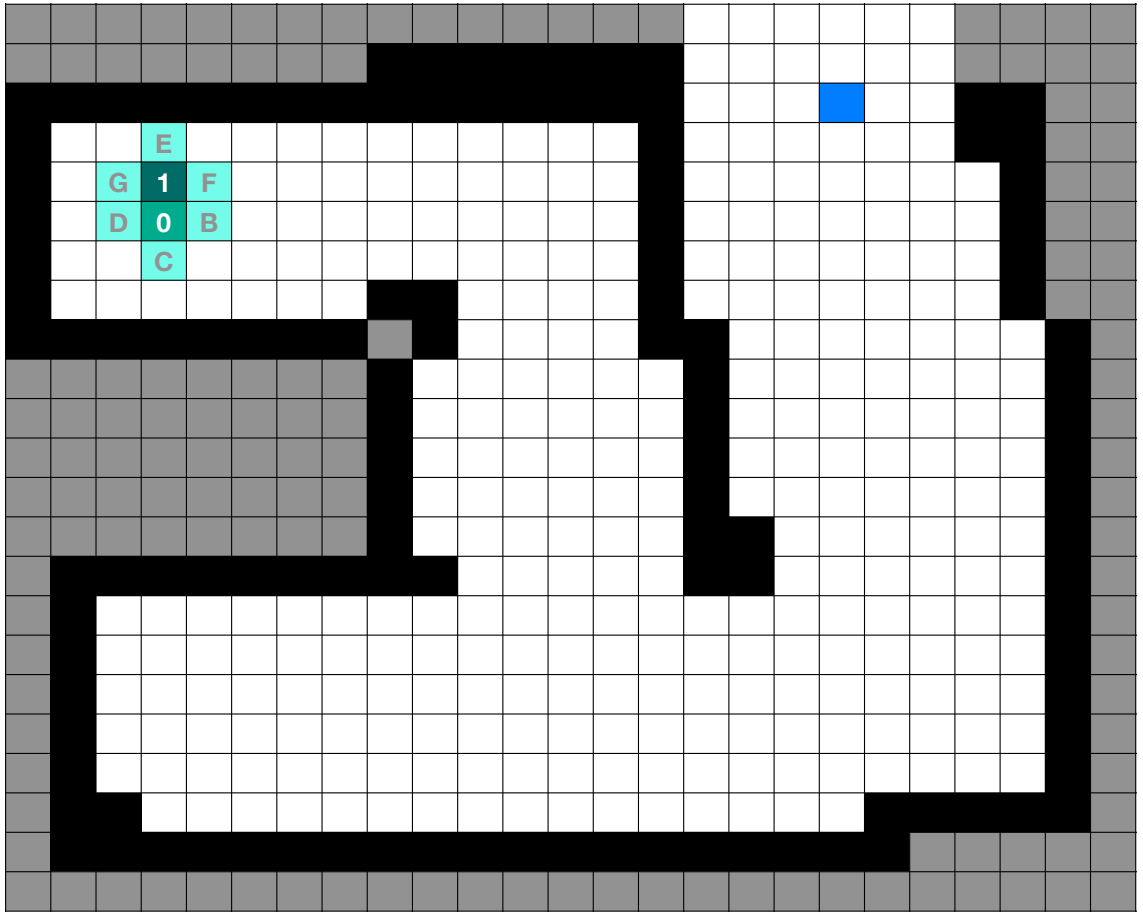
`current_node`

Node A done being processed

Repeat for next node in the queue

All neighbors of current node are "queued"

and assigned distance as one plus current node distance



`visit_queue`

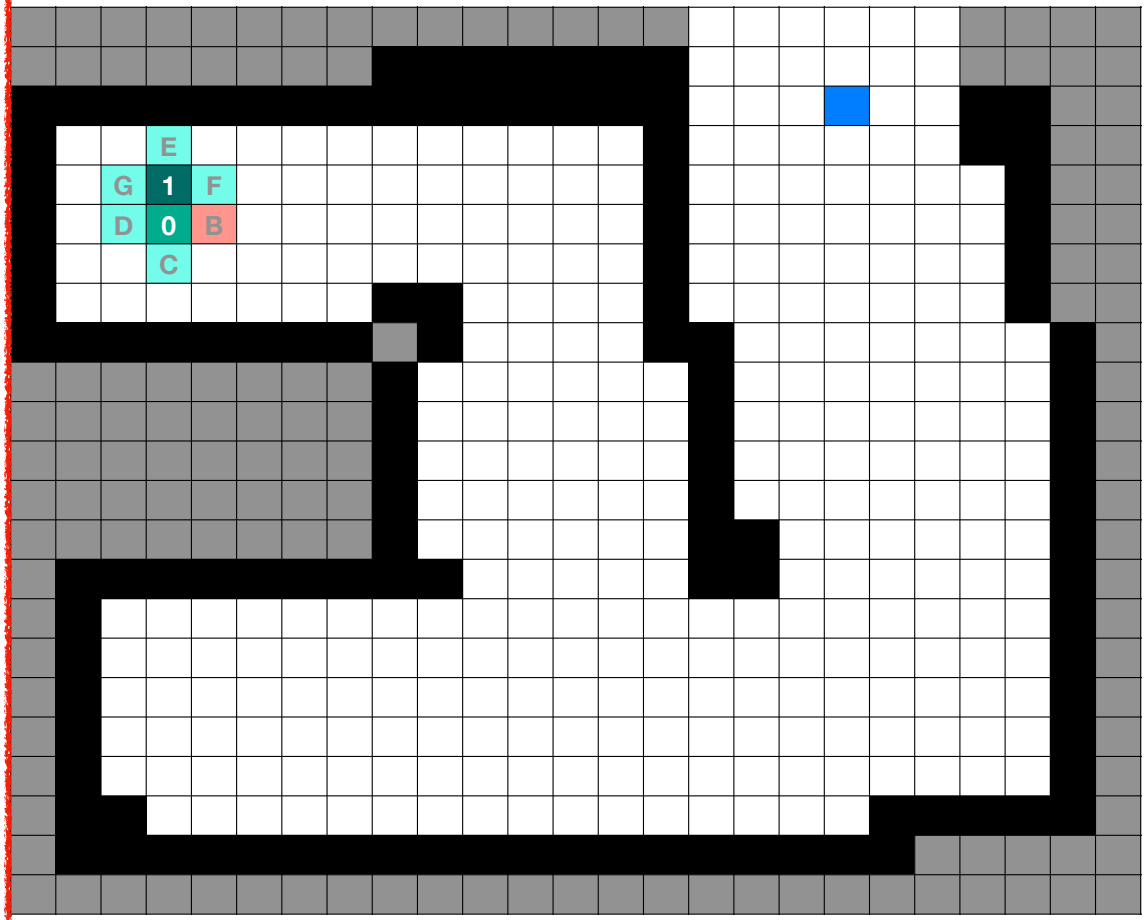
`current_node`

Continue until goal reached or queue is empty

Repeat for next node in the queue

All neighbors of current node are "queued"

and assigned distance as one plus current node distance



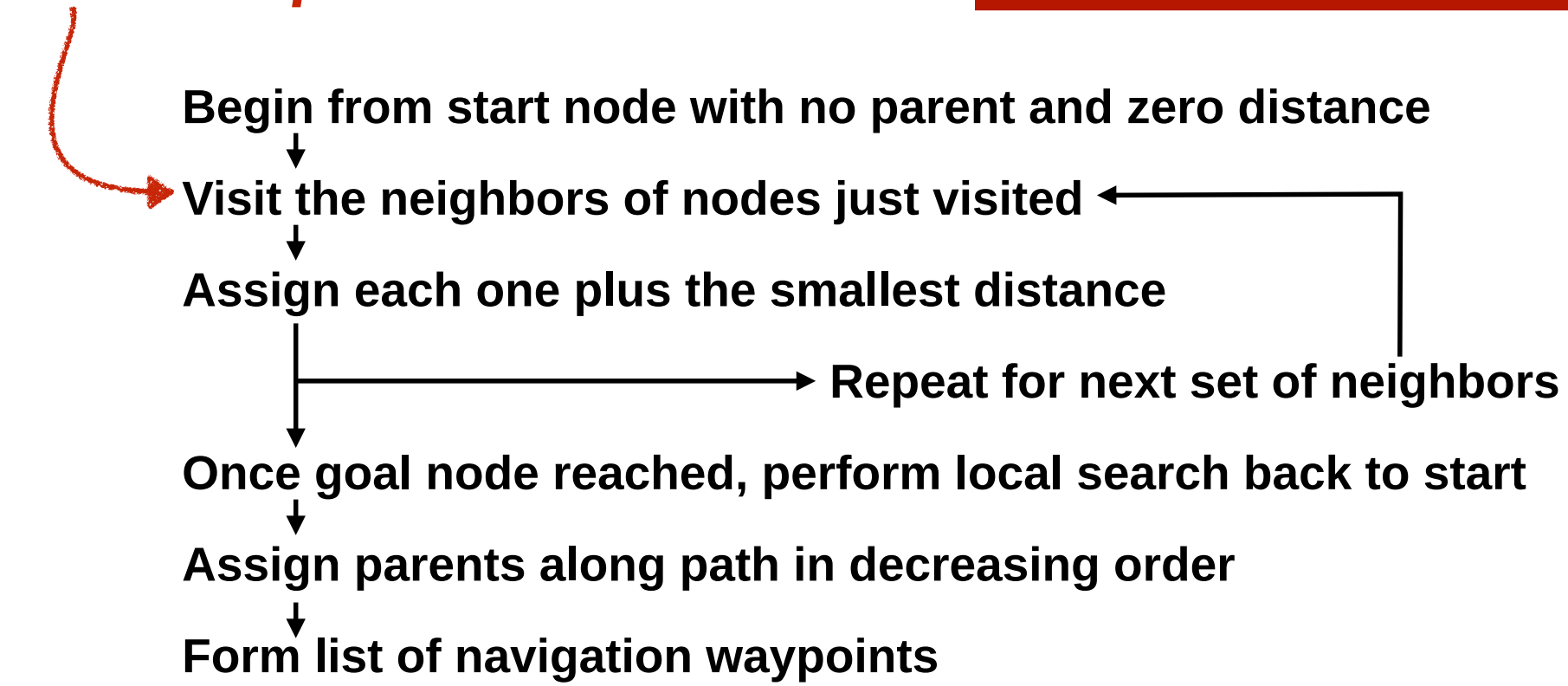
visit_queue

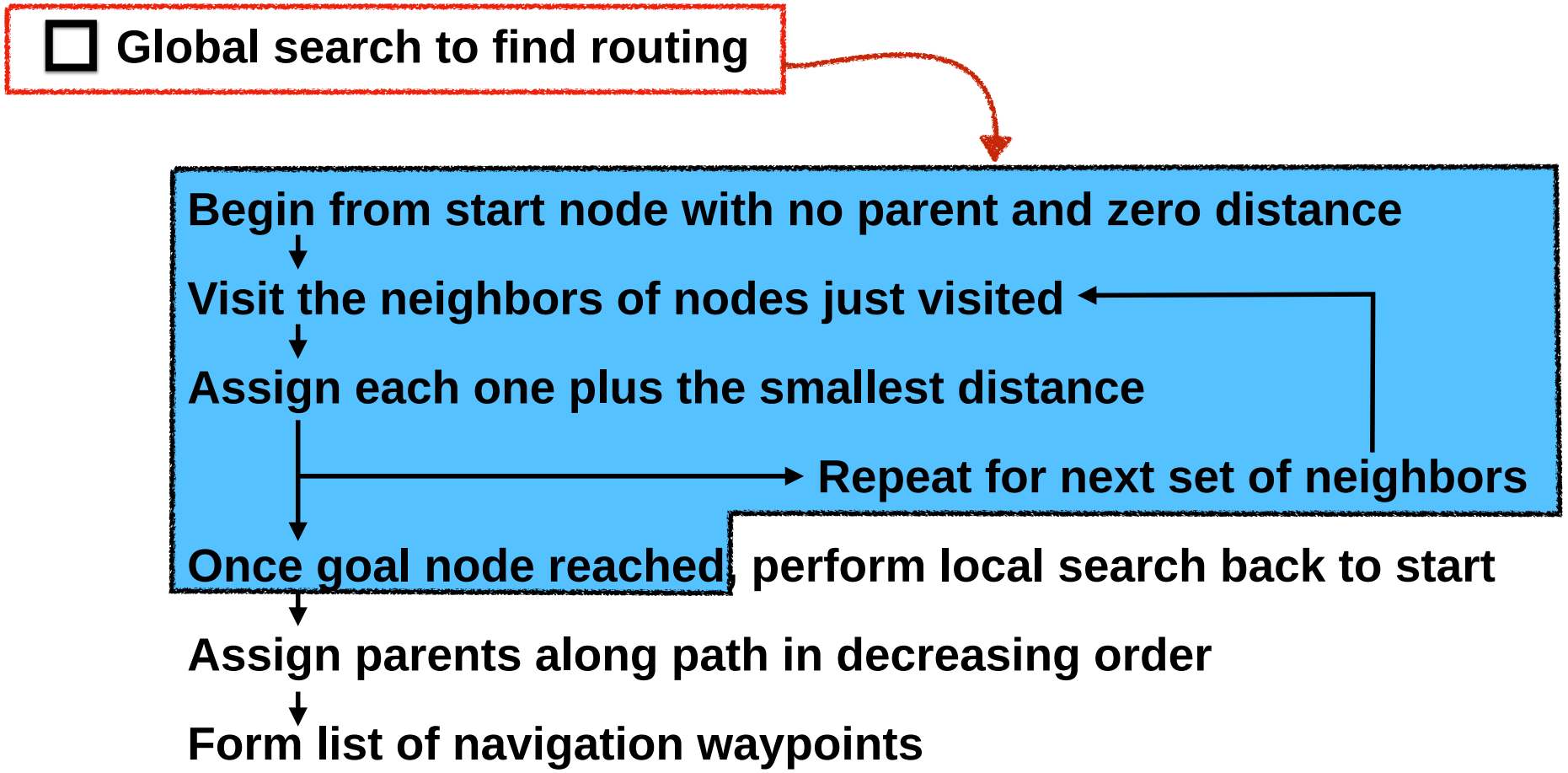
Dequeue

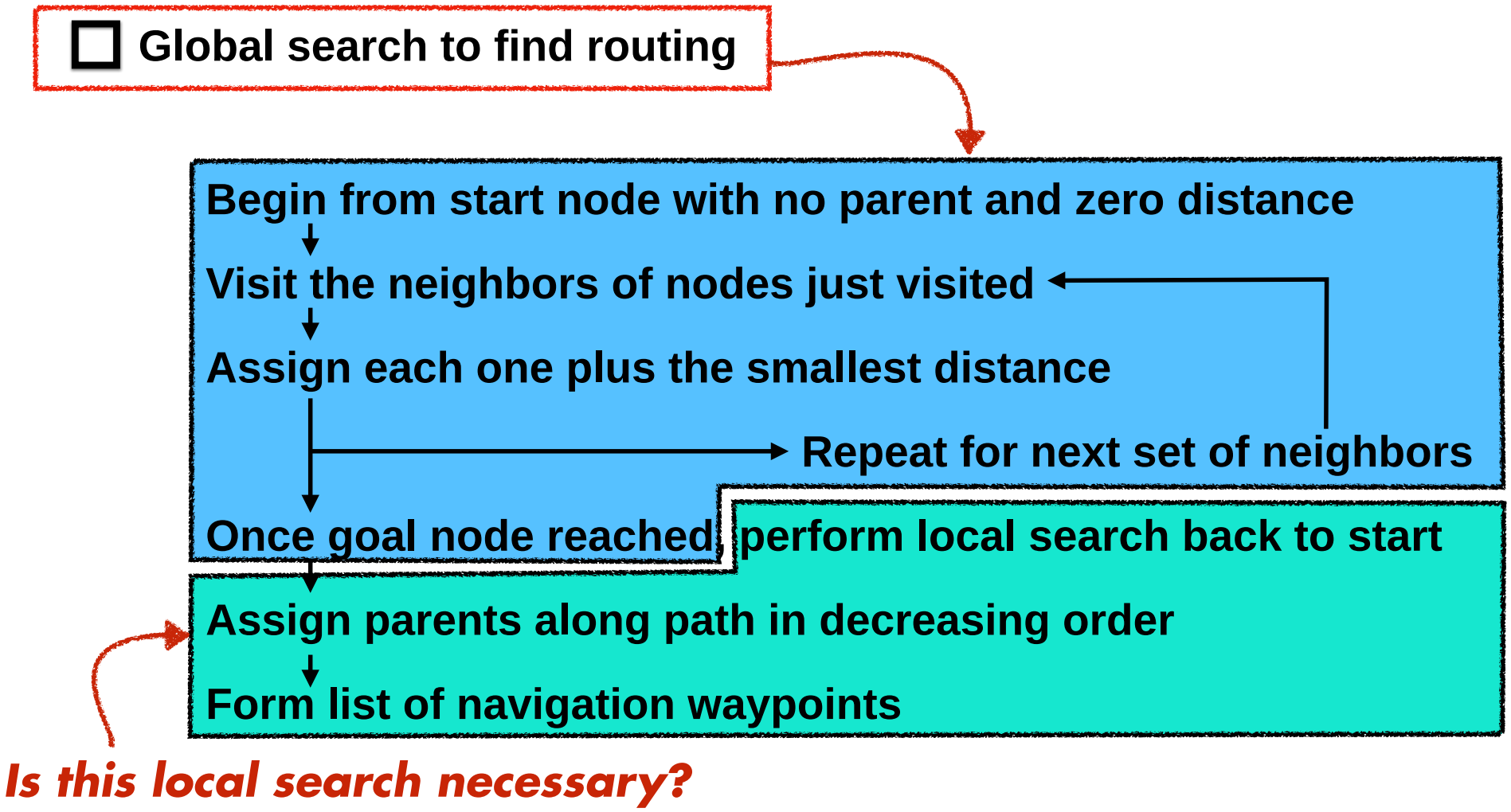
current_node

How to keep track of visited nodes?

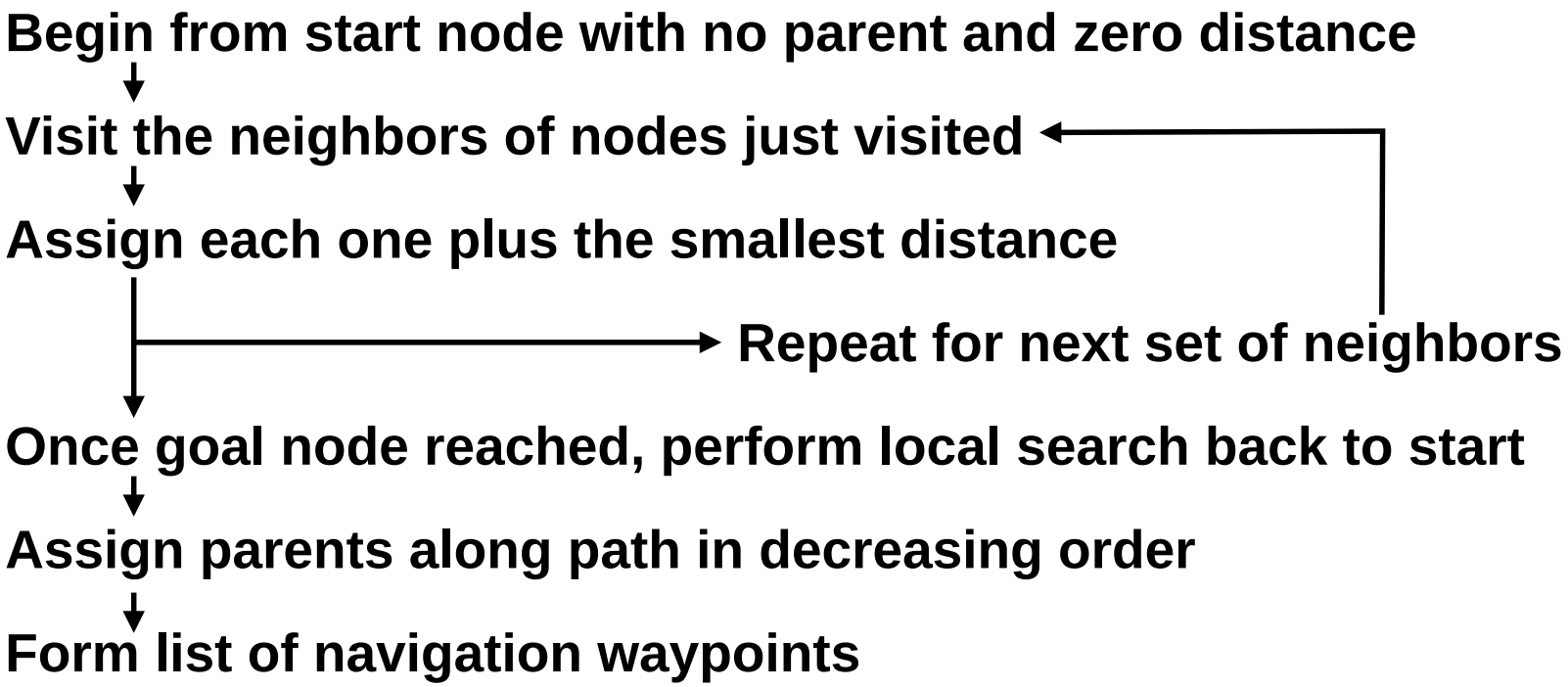
Queue data structure




Global search to find routing
Begin from start node with no parent and zero distance
Visit the neighbors of nodes just visited
Assign each one plus the smallest distance
Repeat for next set of neighbors
Once goal node reached, perform local search back to start
Assign parents along path in decreasing order
Form list of navigation waypoints

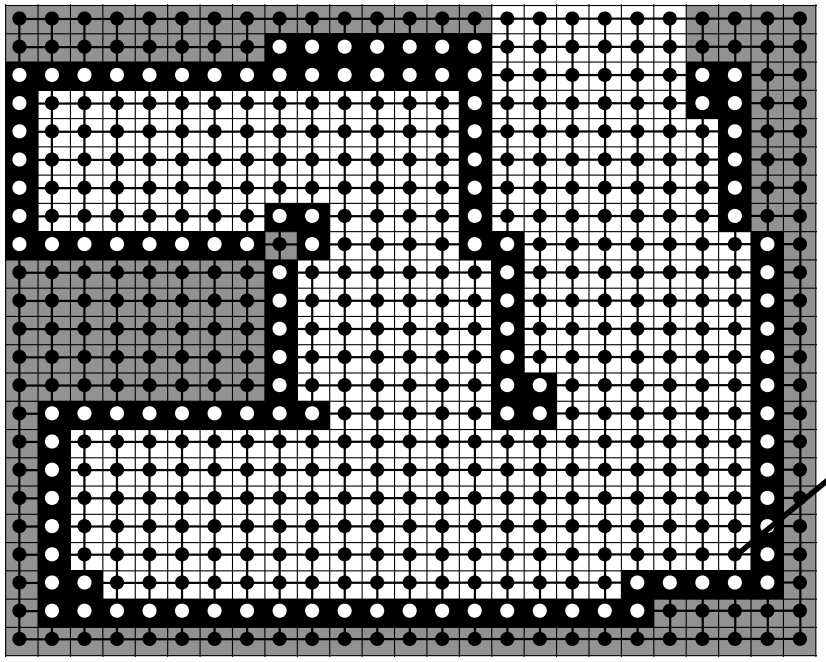

Global search to find routing
Begin from start node with no parent and zero distance
Visit the neighbors of nodes just visited
Assign each one plus the smallest distance
Repeat for next set of neighbors
Once goal node reached, perform local search back to start
Assign parents along path in decreasing order
Form list of navigation waypoints
Is this local search necessary?

Brushfire is one of several algorithms that perform a "floodfill"



Breadth-first Search

For Breadth-first Search, each cell needs to keep track of whether it has been visited and queued



```
origin_x: 2.2
origin_y: 0.3
occupied: false
parent: ??
distance: ??
visited: false
queued: false
```

A graph node stores a struct of information about the cell

Breadth-first Search

Initialize :

All nodes to have no parent, max distance, and as unvisited

Start node to have no parent and zero distance

Visit queue with start node as its only enqueued element

***Iterate* : While visit list not empty and currently visited node is not the goal**

Dequeue new current node to visit and mark it as visited

For each neighbor :

Add to visit queue, if not previously visited or queued

If Distance of neighbor > Distance of current node + …

Cost to move from current node to neighbor

Then Update neighbor's parent to be current node and …

distance to be distance of current node + cost to move

Breadth-first Search

Initialize :

All nodes to have no parent, max distance, and as unvisited

Start node to have no parent and zero distance

Visit queue with start node as its only enqueued element

Iterate : While visit list not empty and currently visited node is not the goal

Dequeue new current node to visit and mark it as visited

For each neighbor :

Add to visit queue, if not previously visited or queued

If Distance of neighbor > Distance of current node + …

Cost to move from current node to neighbor

Then Update neighbor's parent to be current node and …

distance to be distance of current node + cost to move

Initialize :

All nodes to have no parent, max distance, and as unvisited

Start node to have no parent and zero distance

Visit queue with start node as its only enqueued element

***Iterate* : While visit list not empty and currently visited node is not the goal**

Dequeue new current node to visit and mark it as visited

For each neighbor :

Add to visit queue, if not previously visited or queued

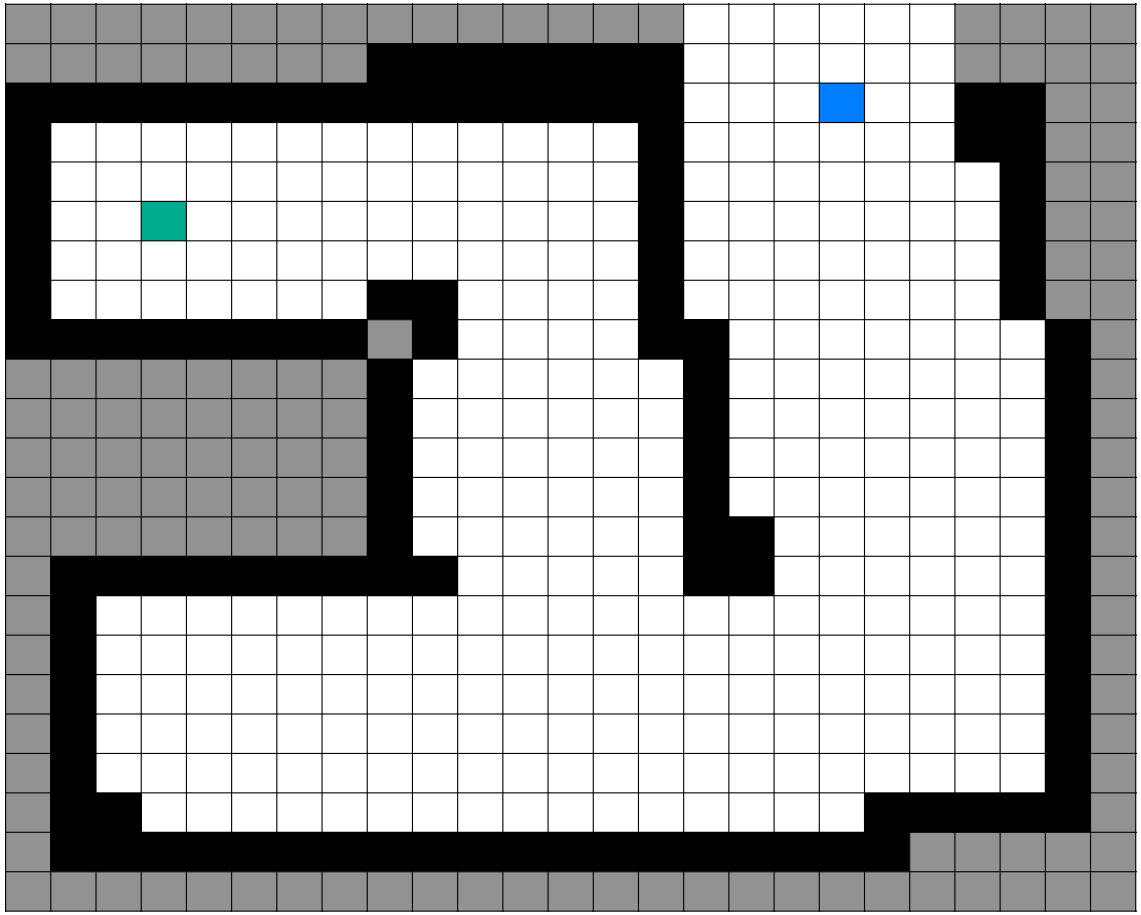
If Distance of neighbor > Distance of current node + …

Cost to move from current node to neighbor

Then Update neighbor's parent to be current node and …

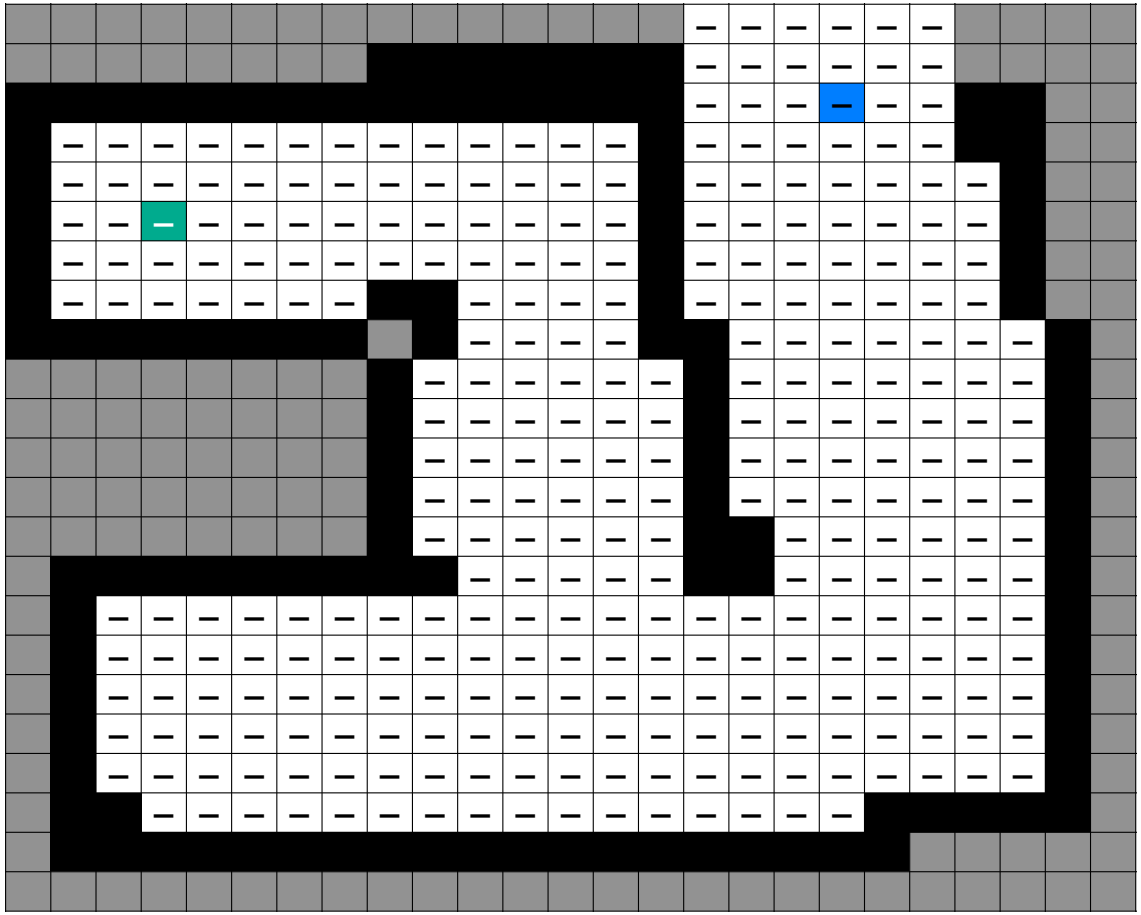
distance to be distance of current node + cost to move

All nodes to have no parent, max distance, and as unvisited



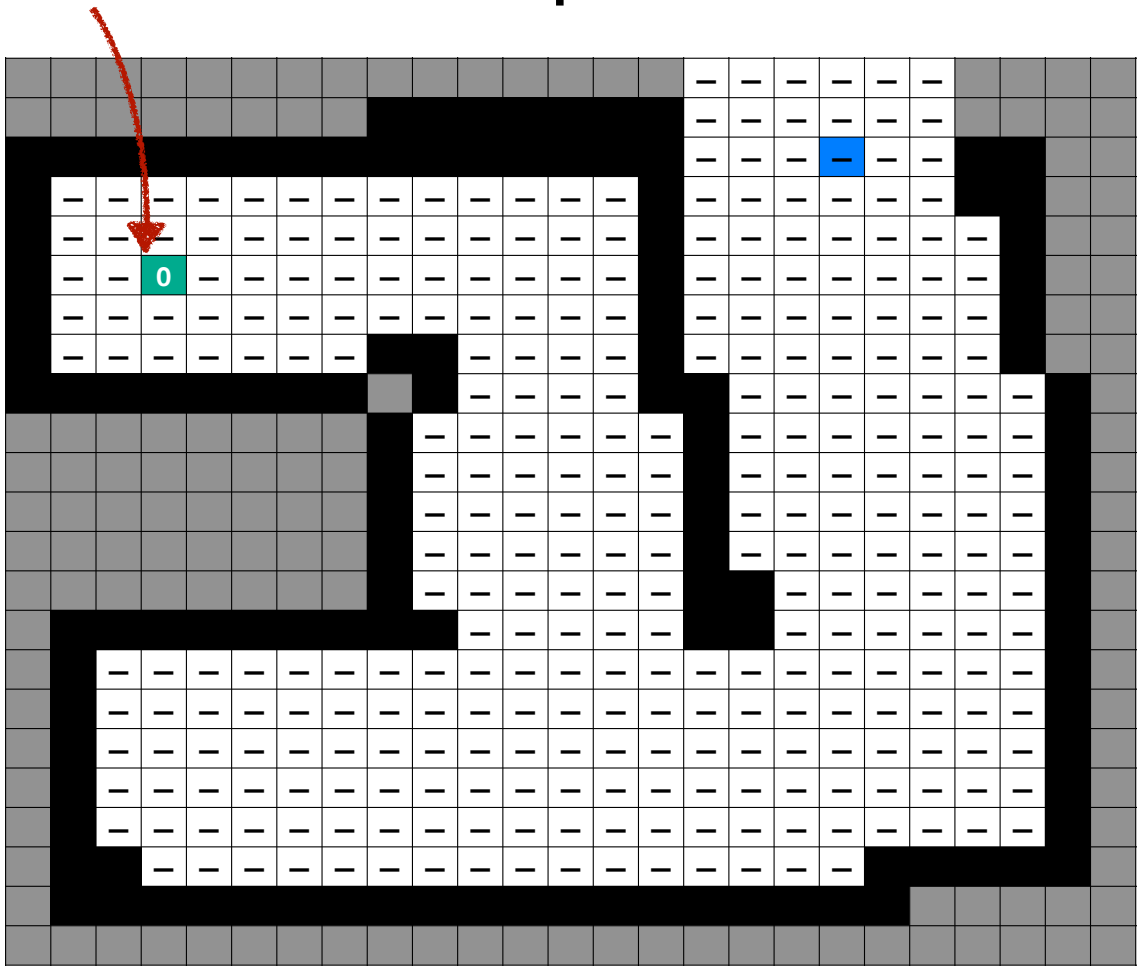
– is some very large number

All nodes to have no parent, max distance, and as unvisited

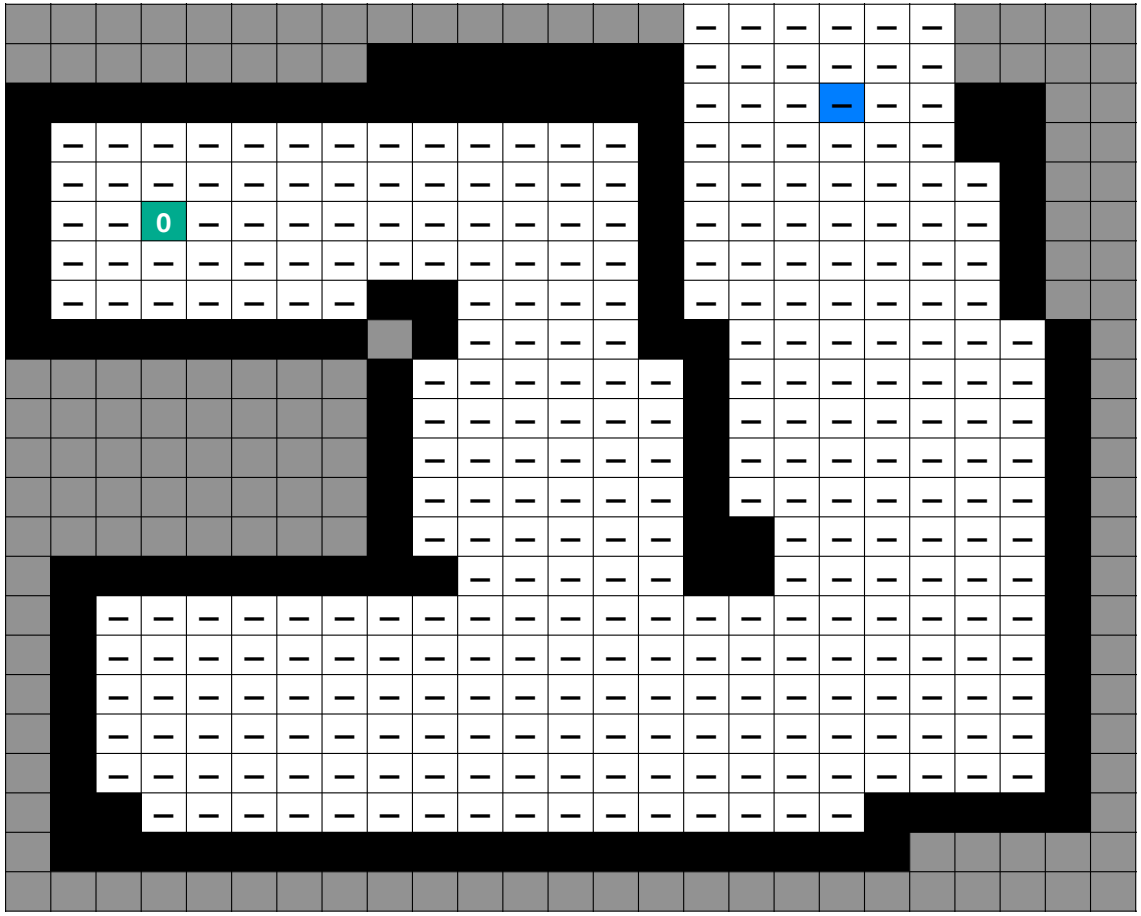


– is some very large number

Start node to have no parent and zero distance



Visit queue with start node as its only enqueued element



visit_queue

Initialize :

All nodes to have no parent, max distance, and as unvisited

Start node to have no parent and zero distance

Visit queue with start node as its only enqueued element

***Iterate* : While visit list not empty and currently visited node is not the goal**

Dequeue new current node to visit and mark it as visited

For each neighbor :

Add to visit queue, if not previously visited or queued

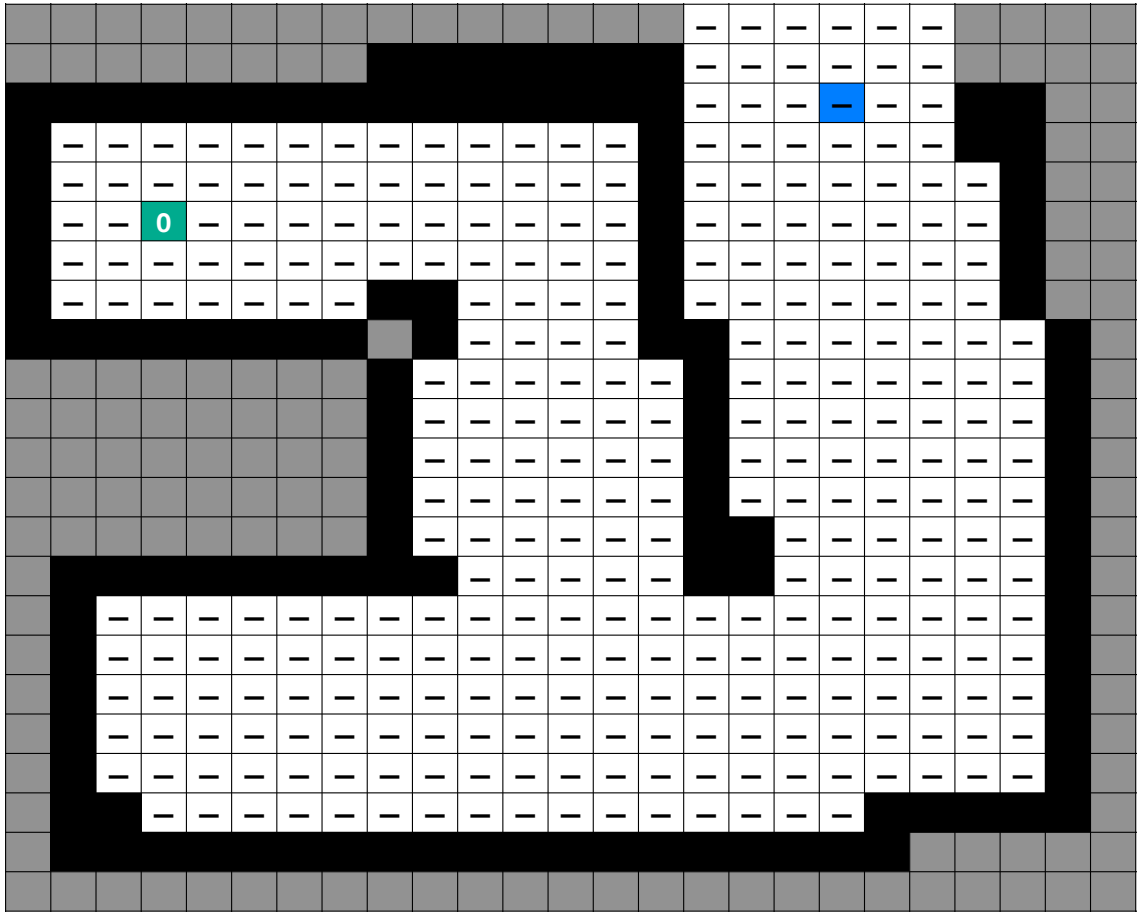
If Distance of neighbor > Distance of current node + …

Cost to move from current node to neighbor

Then Update neighbor's parent to be current node and …

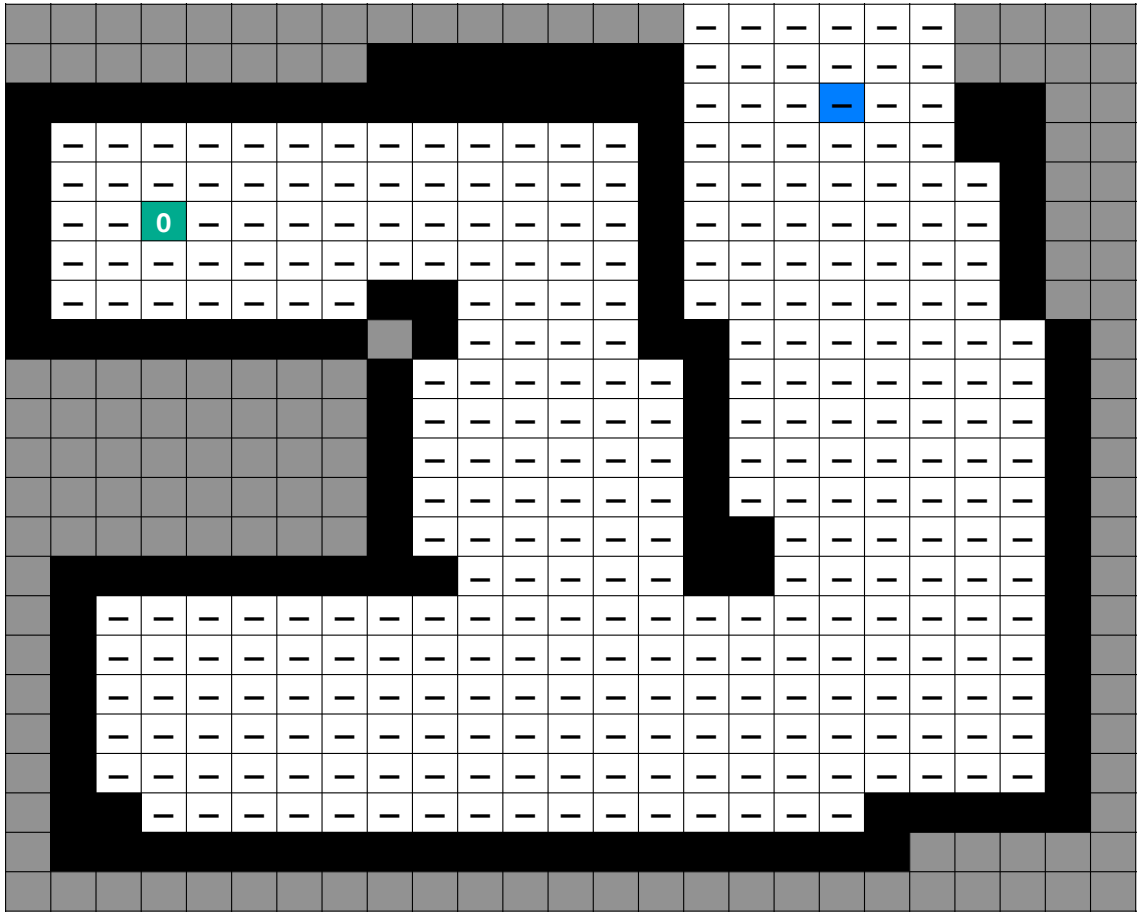
distance to be distance of current node + cost to move

Dequeue new current node to visit and mark it as visited



visit_queue

Dequeue new current node to visit and mark it as visited



visit_queue

current_node

Initialize :

All nodes to have no parent, max distance, and as unvisited

Start node to have no parent and zero distance

Visit queue with start node as its only enqueued element

Iterate : While visit list not empty and currently visited node is not the goal

Dequeue new current node to visit and mark it as visited

For each neighbor :

Add to visit queue, if not previously visited or queued

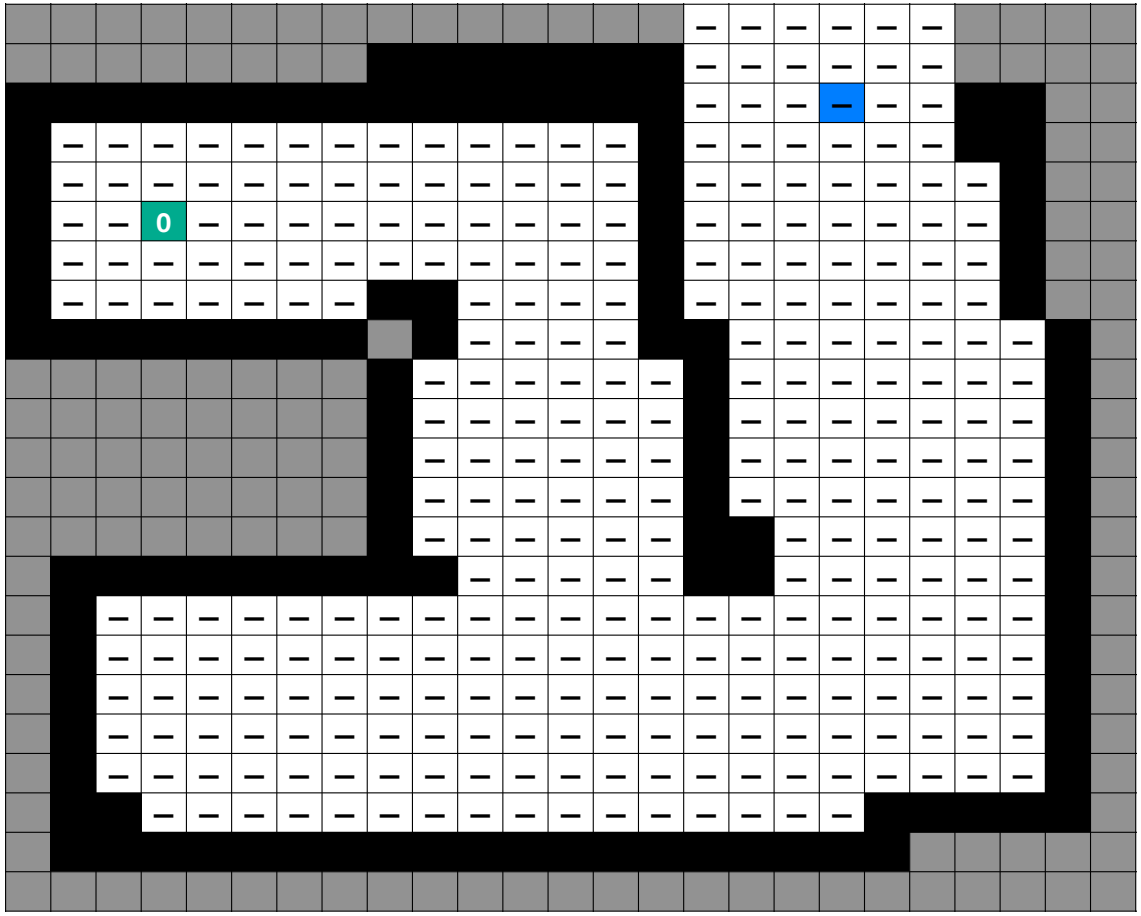
If Distance of neighbor > Distance of current node + …

Cost to move from current node to neighbor

Then Update neighbor's parent to be current node and …

distance to be distance of current node + cost to move

Add to visit queue, if not previously visited or queued

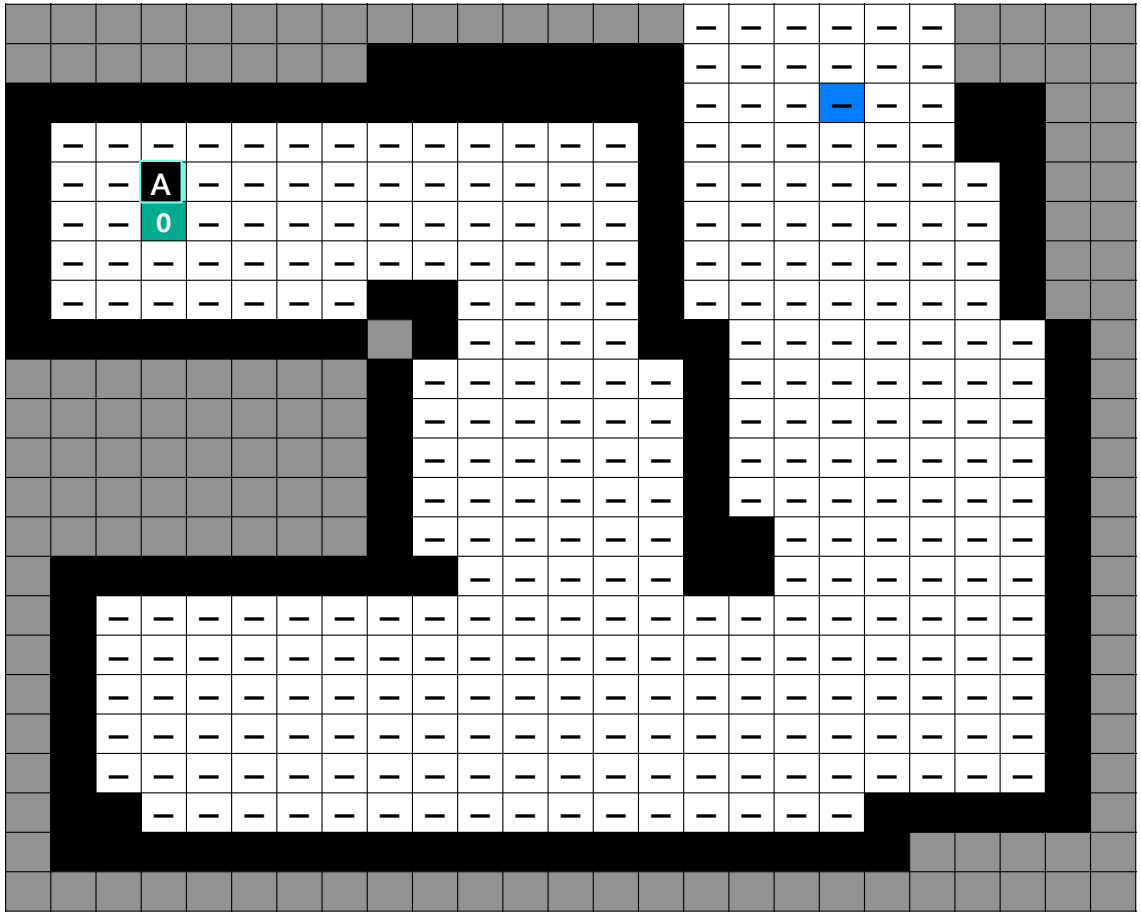


visit_queue

current_node

Add to visit queue, if not previously visited or queued

Queue first neighbor

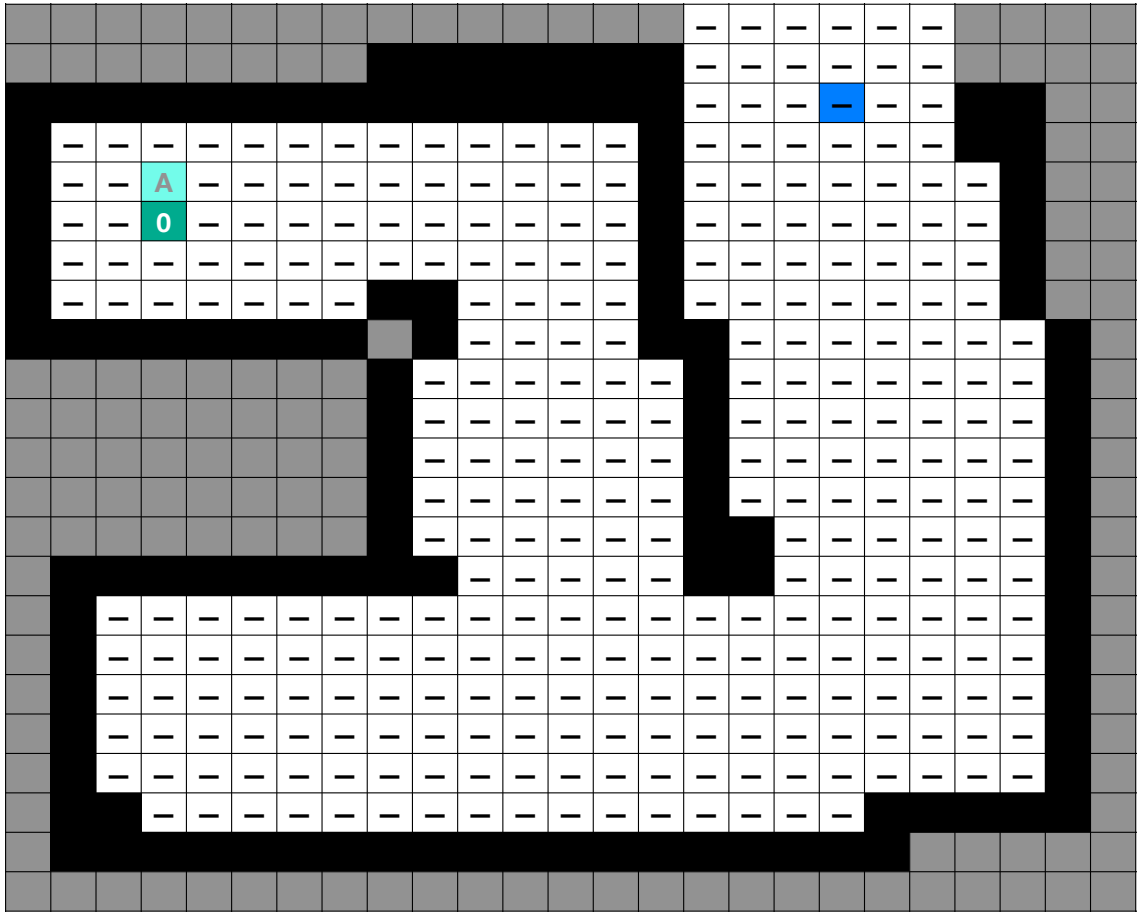


visit_queue

current_node

Add to visit queue, if not previously visited or queued

Queue first neighbor



`visit_queue`

`current_node`

Initialize :

All nodes to have no parent, max distance, and as unvisited

Start node to have no parent and zero distance

Visit queue with start node as its only enqueued element

Iterate : While visit list not empty and currently visited node is not the goal

Dequeue new current node to visit and mark it as visited

For each neighbor :

Add to visit queue, if not previously visited or queued

If Distance of neighbor > Distance of current node + …

Cost to move from current node to neighbor

Then Update neighbor's parent to be current node and …

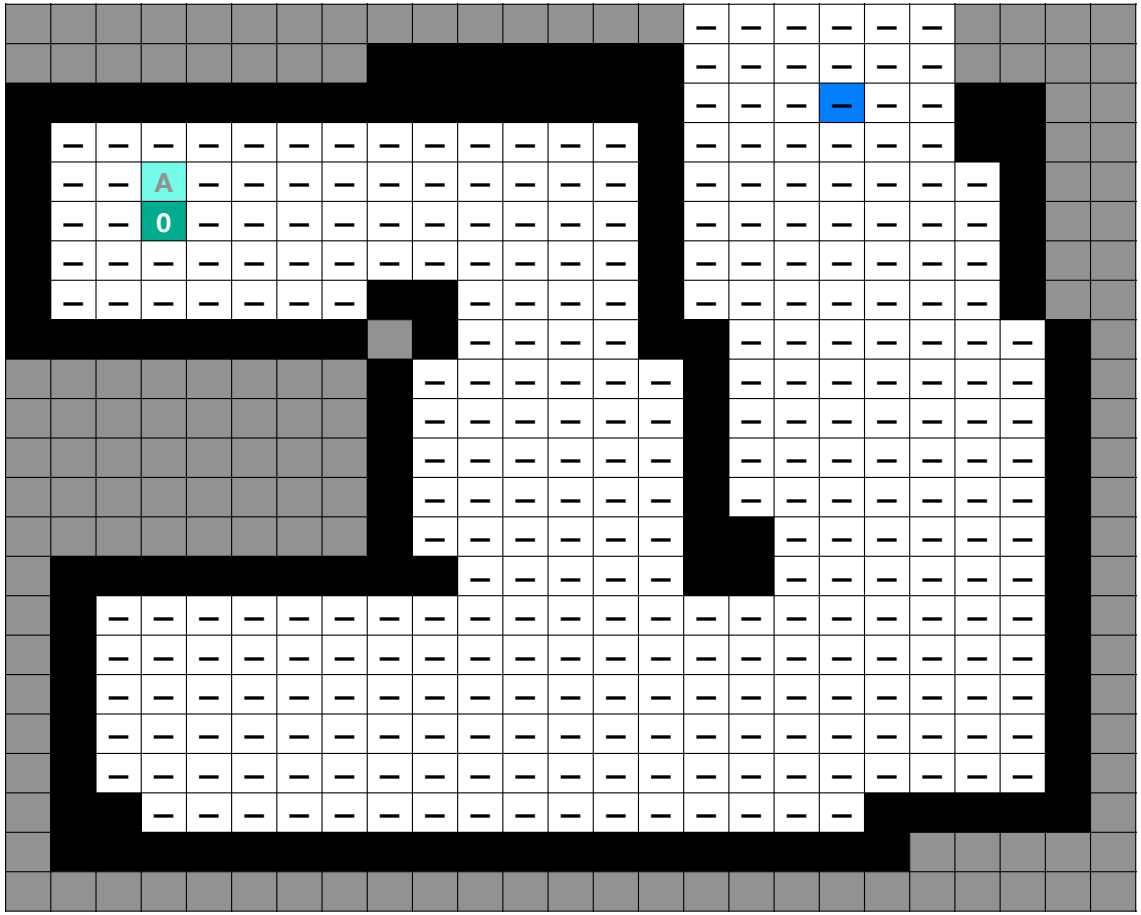
distance to be distance of current node + cost to move

If Distance of neighbor > Distance of current node + …

–

0

Cost to move from current node to neighbor

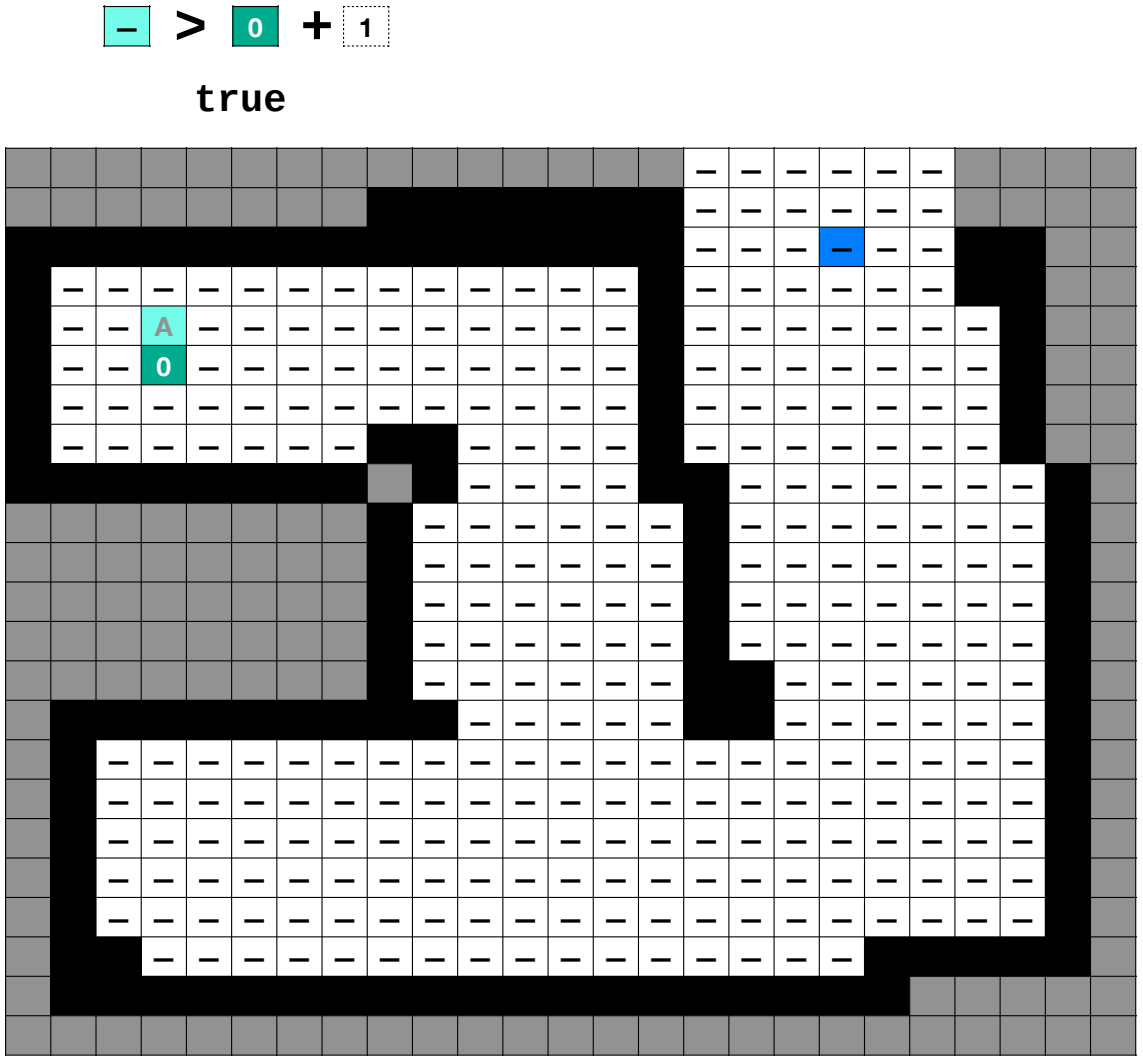


1

visit_queue

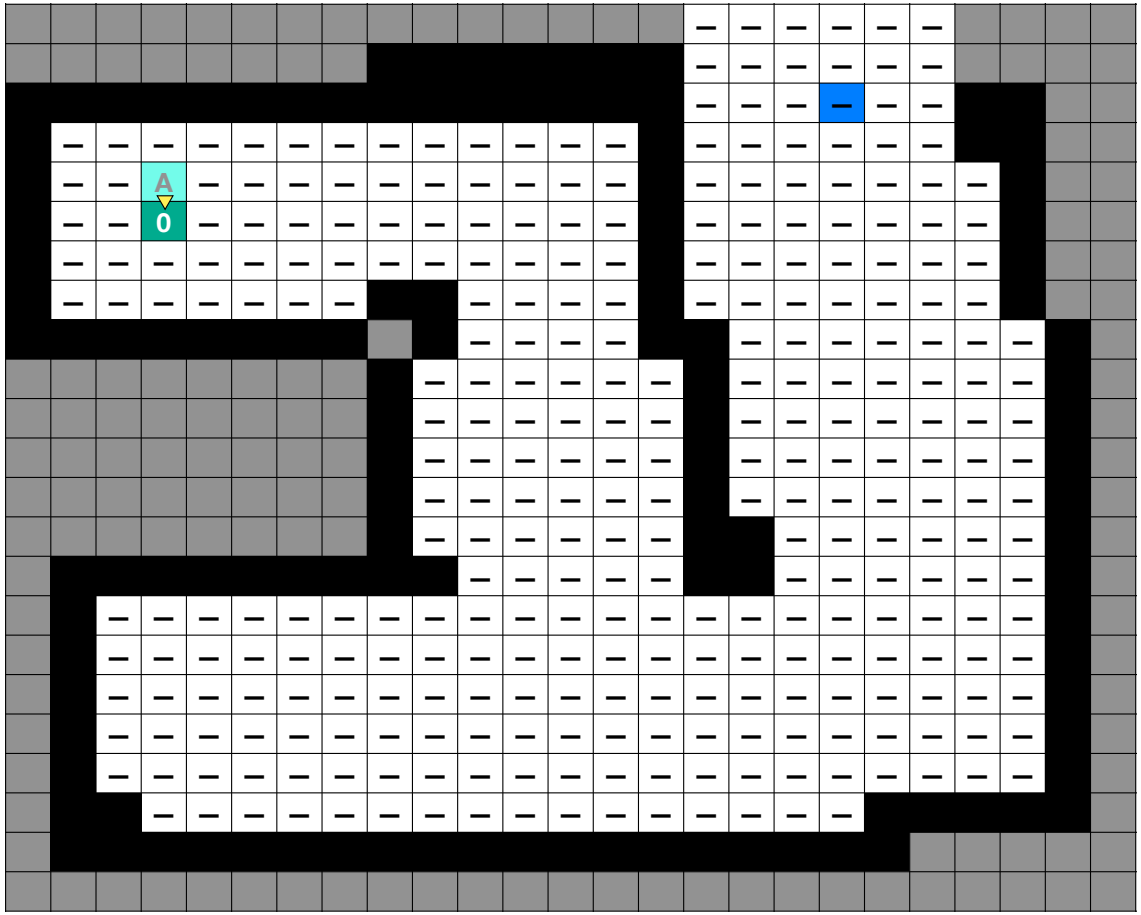
A

current_node


–
>
0
+
1
true
A
0
visit_queue
A
current_node

Then Update neighbor's parent to be current node and …

true

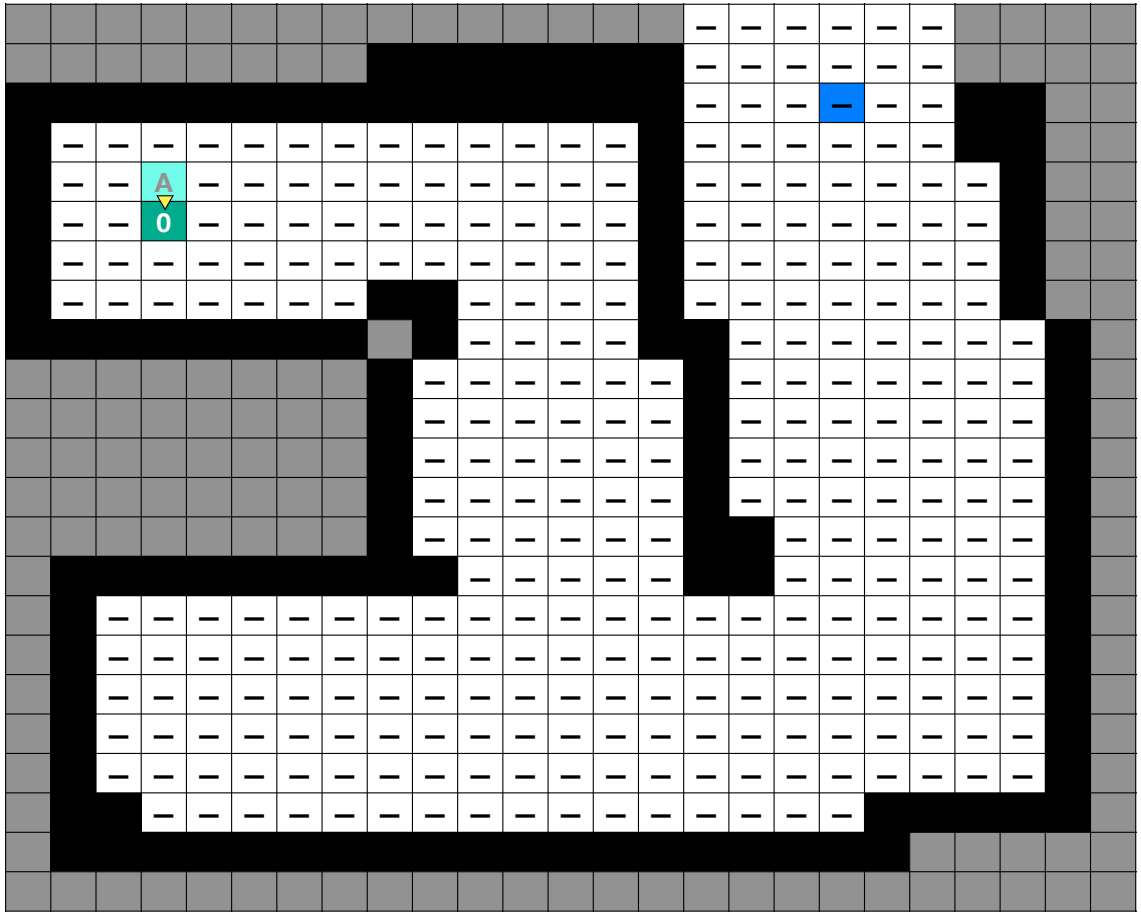


visit_queue

current_node

update neighbor’s distance to be distance of current node + cost to move

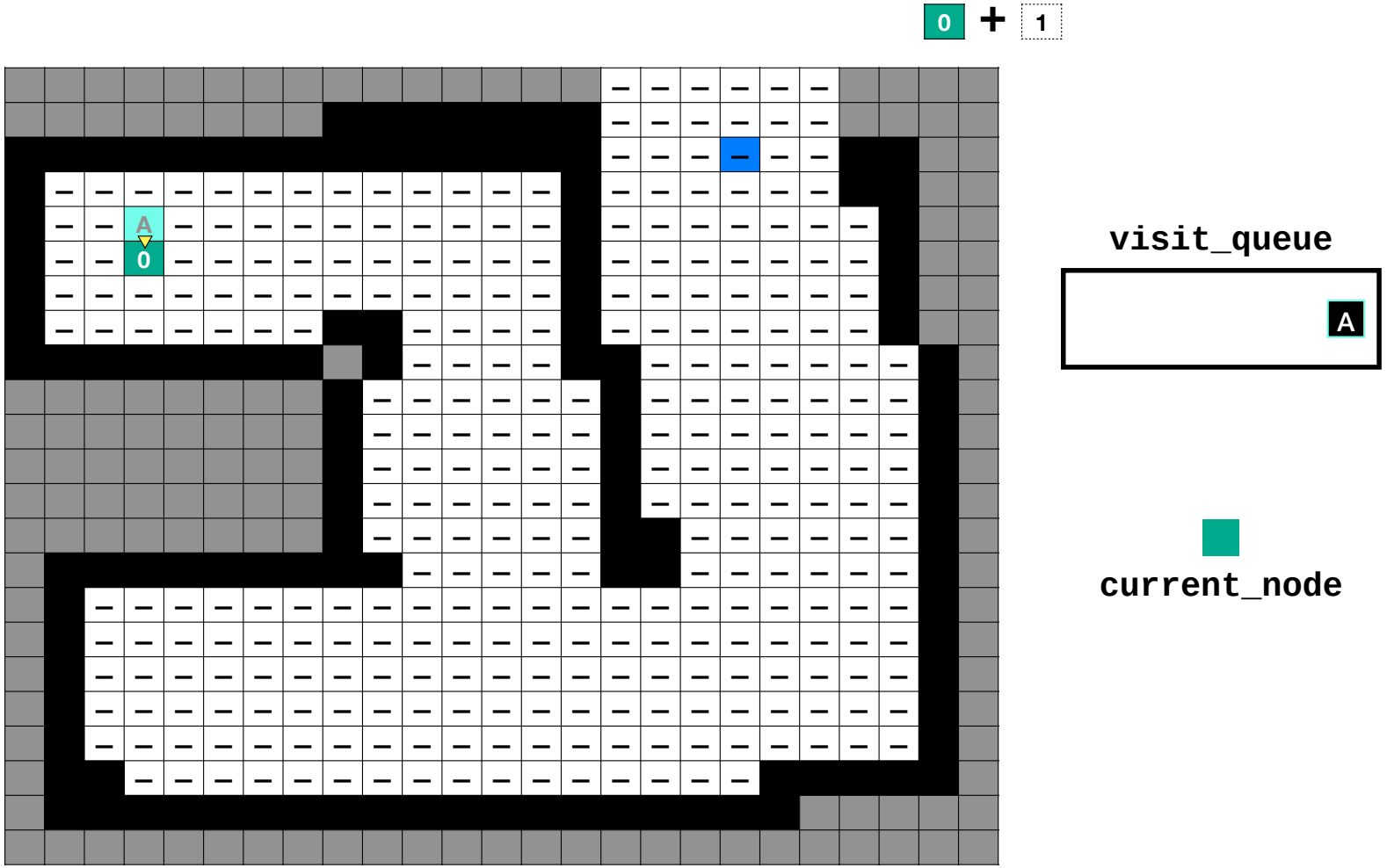
true



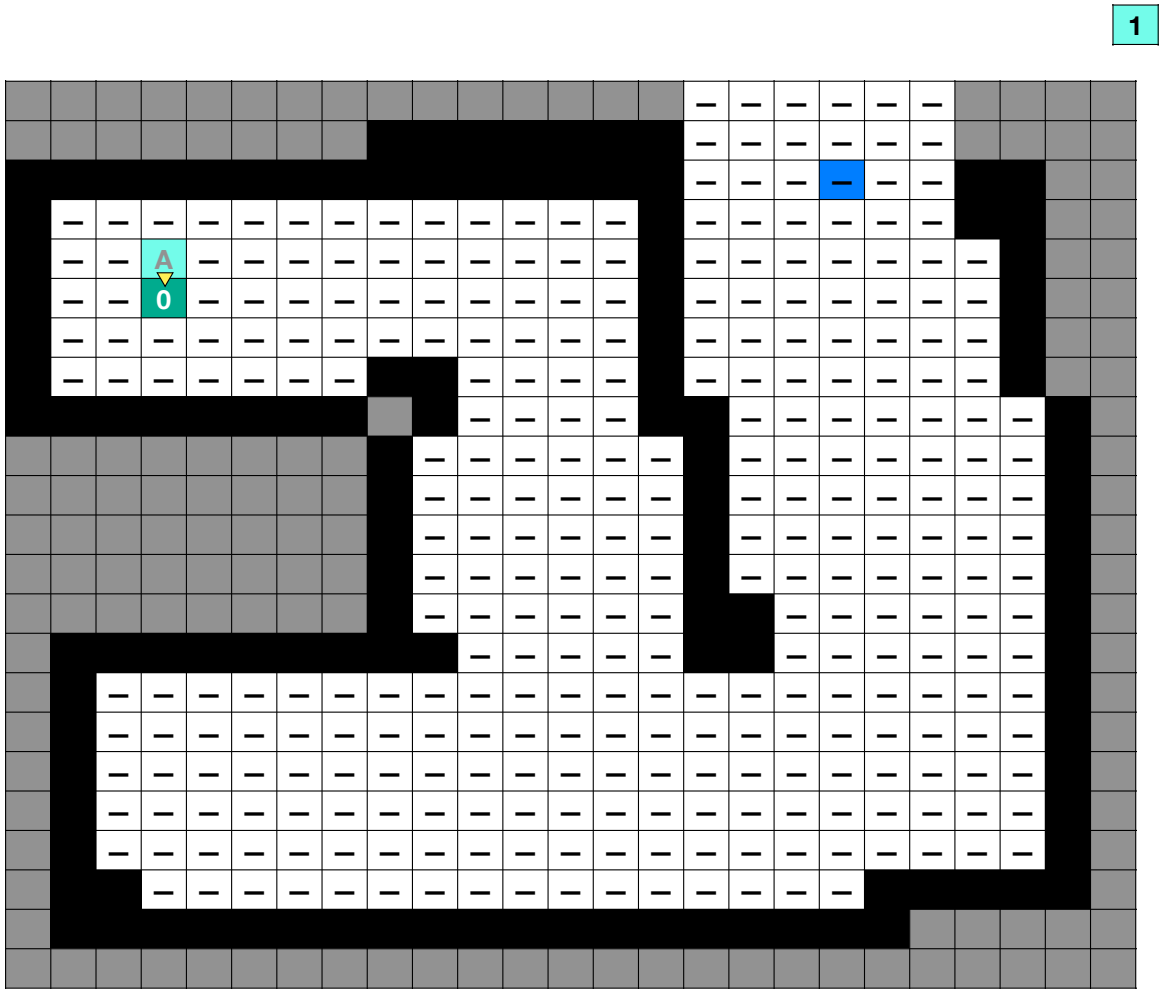
visit_queue

current_node

update neighbor’s distance to be distance of current node + cost to move



update neighbor’s distance to be distance of current node + cost to move

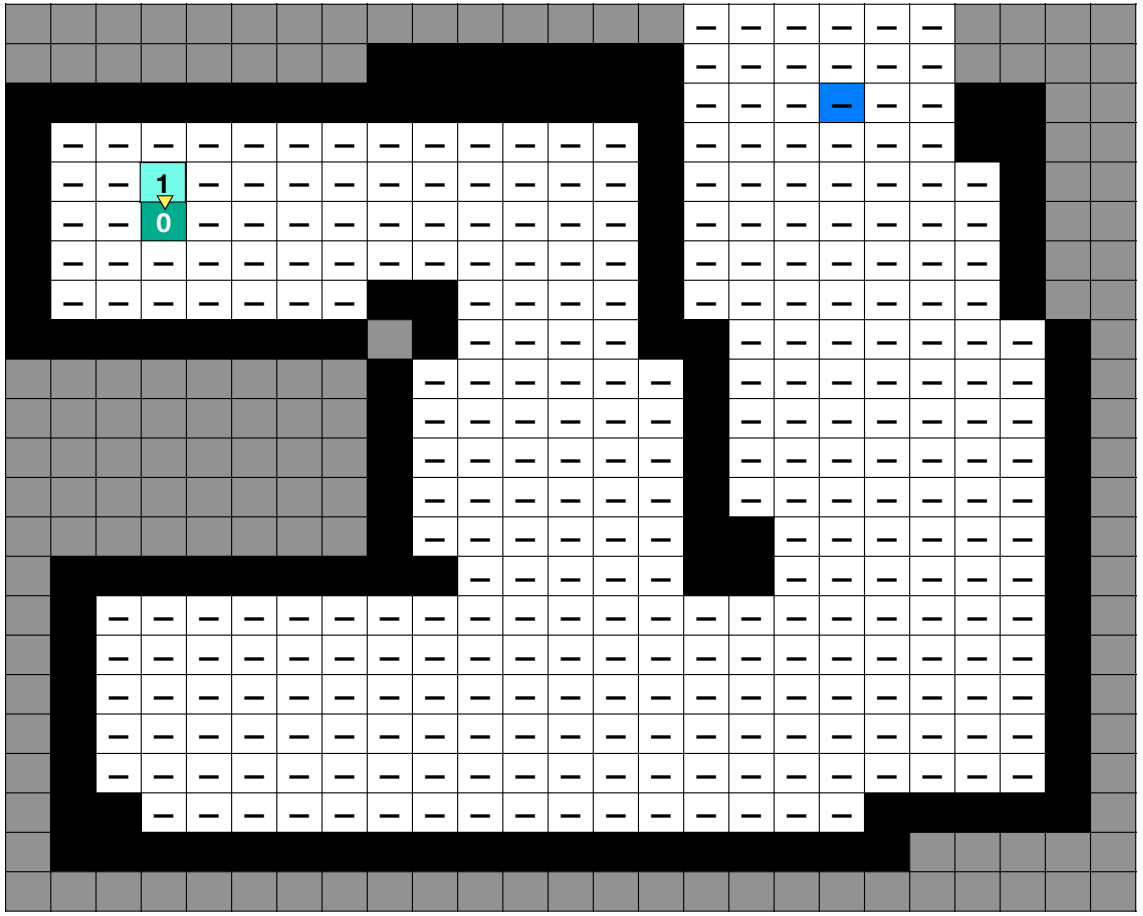


visit_queue

A

current_node

update neighbor’s distance to be distance of current node + cost to move



visit_queue

A

current_node

Initialize :

All nodes to have no parent, max distance, and as unvisited

Start node to have no parent and zero distance

Visit queue with start node as its only enqueued element

Iterate : While visit list not empty and currently visited node is not the goal

Dequeue new current node to visit and mark it as visited

For each neighbor :

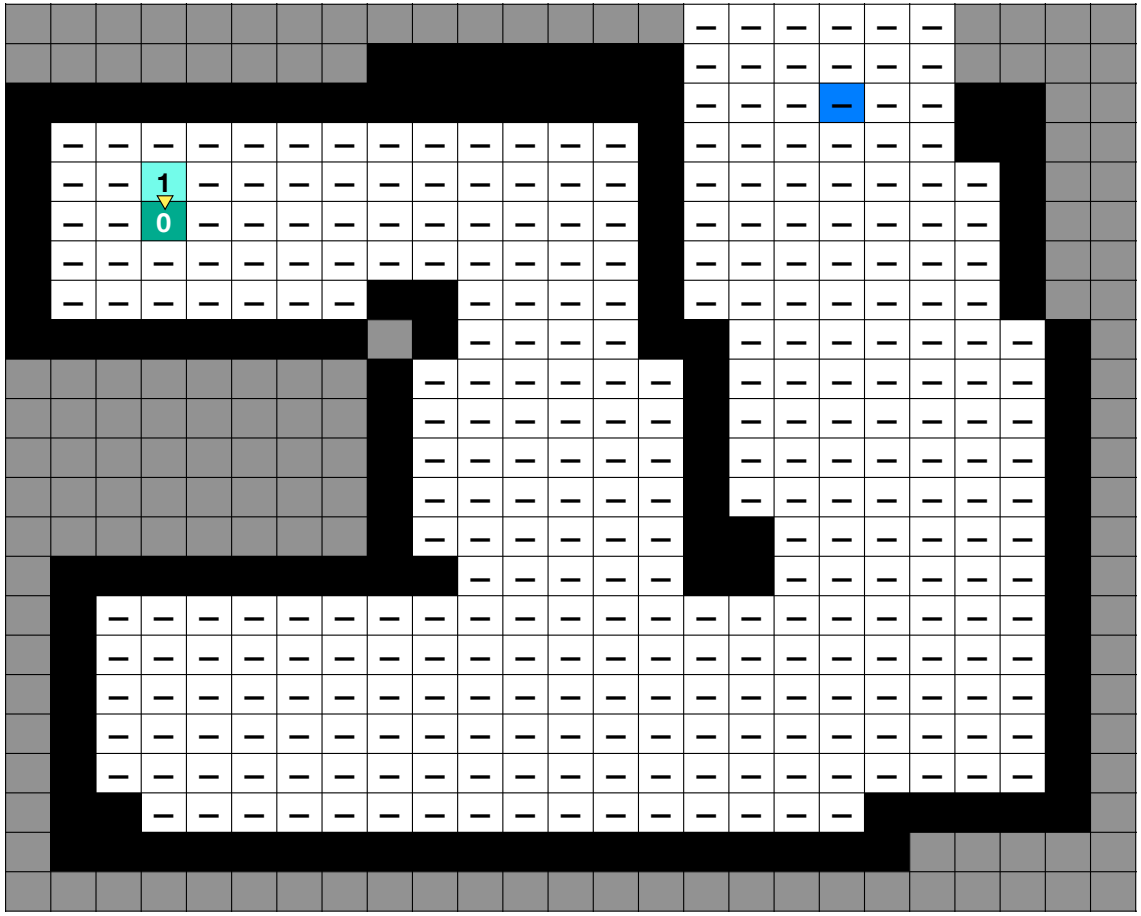
Add to visit queue, if not previously visited or queued

If Distance of neighbor > Distance of current node + …

Cost to move from current node to neighbor

Then Update neighbor's parent to be current node and …

distance to be distance of current node + cost to move

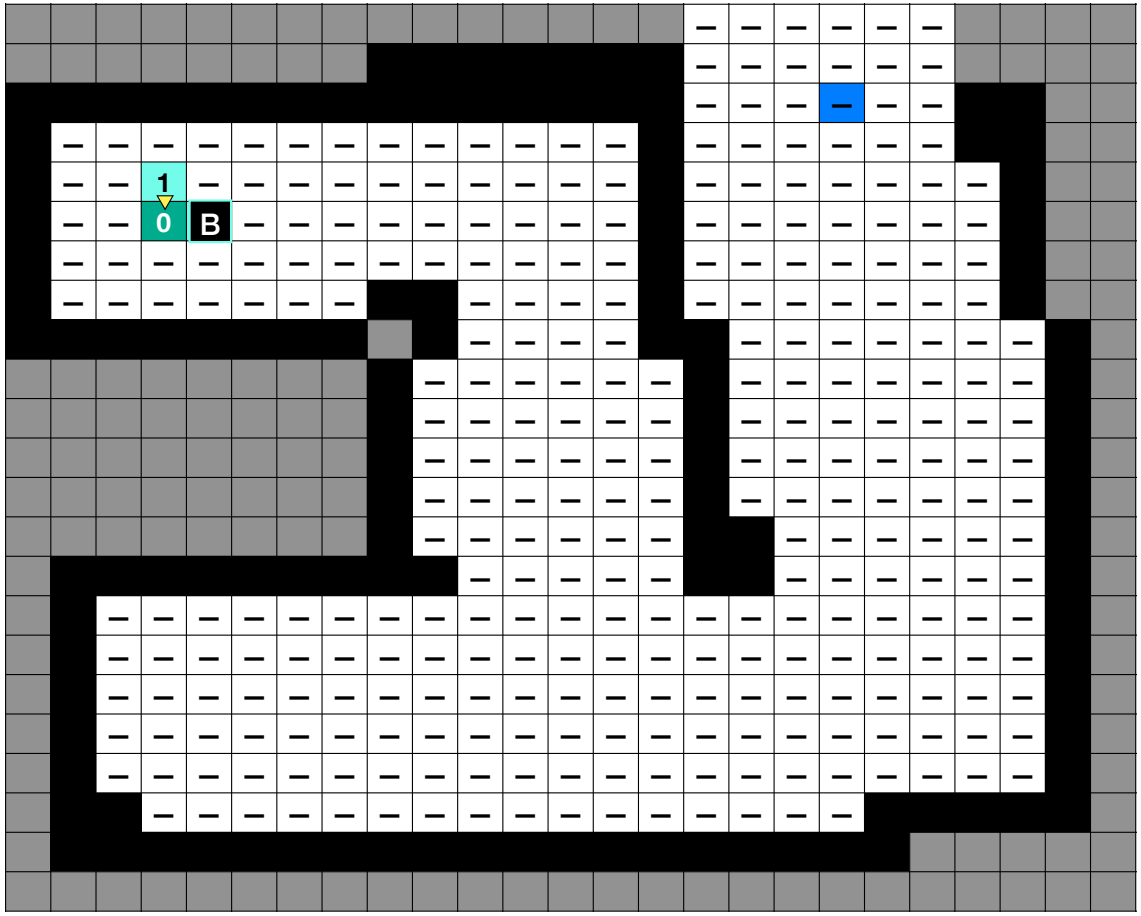


visit_queue

A

current_node

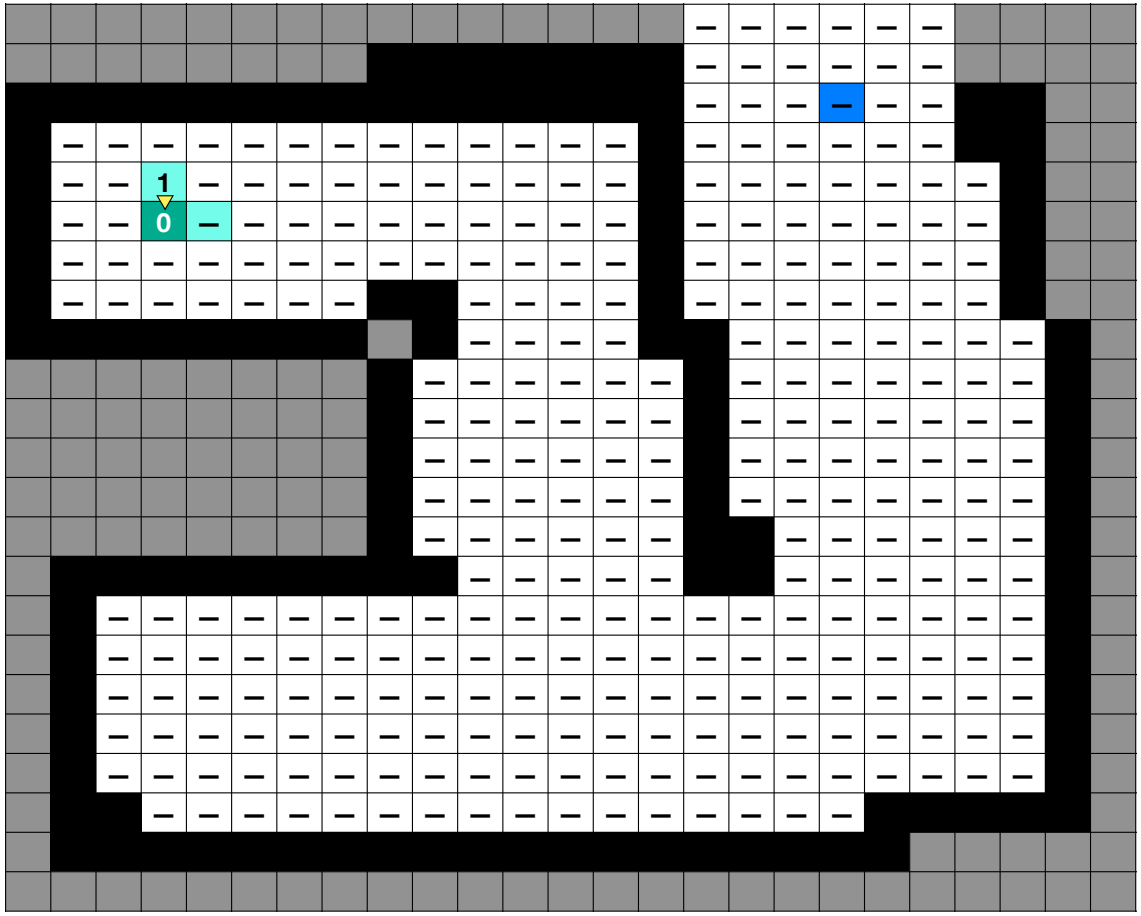
Queue second neighbor



`visit_queue`

`current_node`

Queue second neighbor

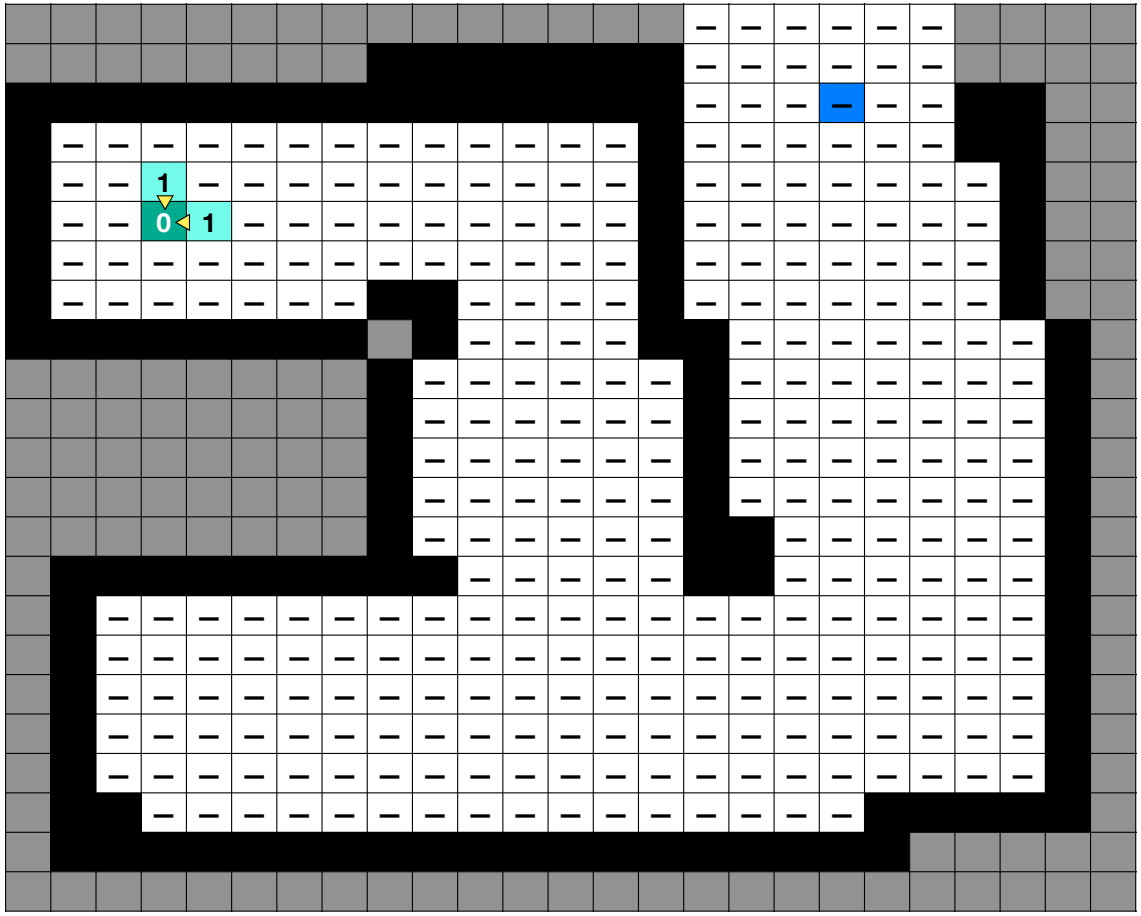


visit_queue

B A

current_node

Assign distance and parent

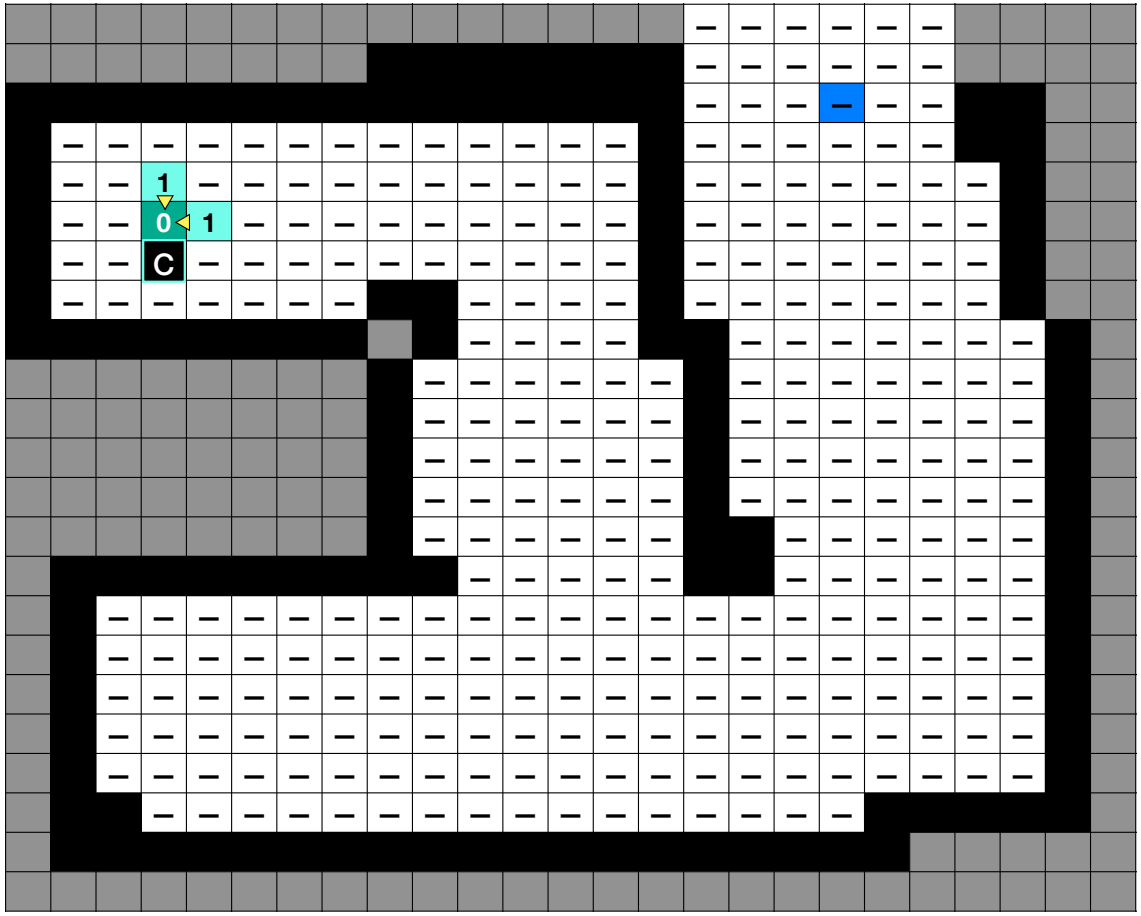


visit_queue

B A

current_node

Queue third neighbor

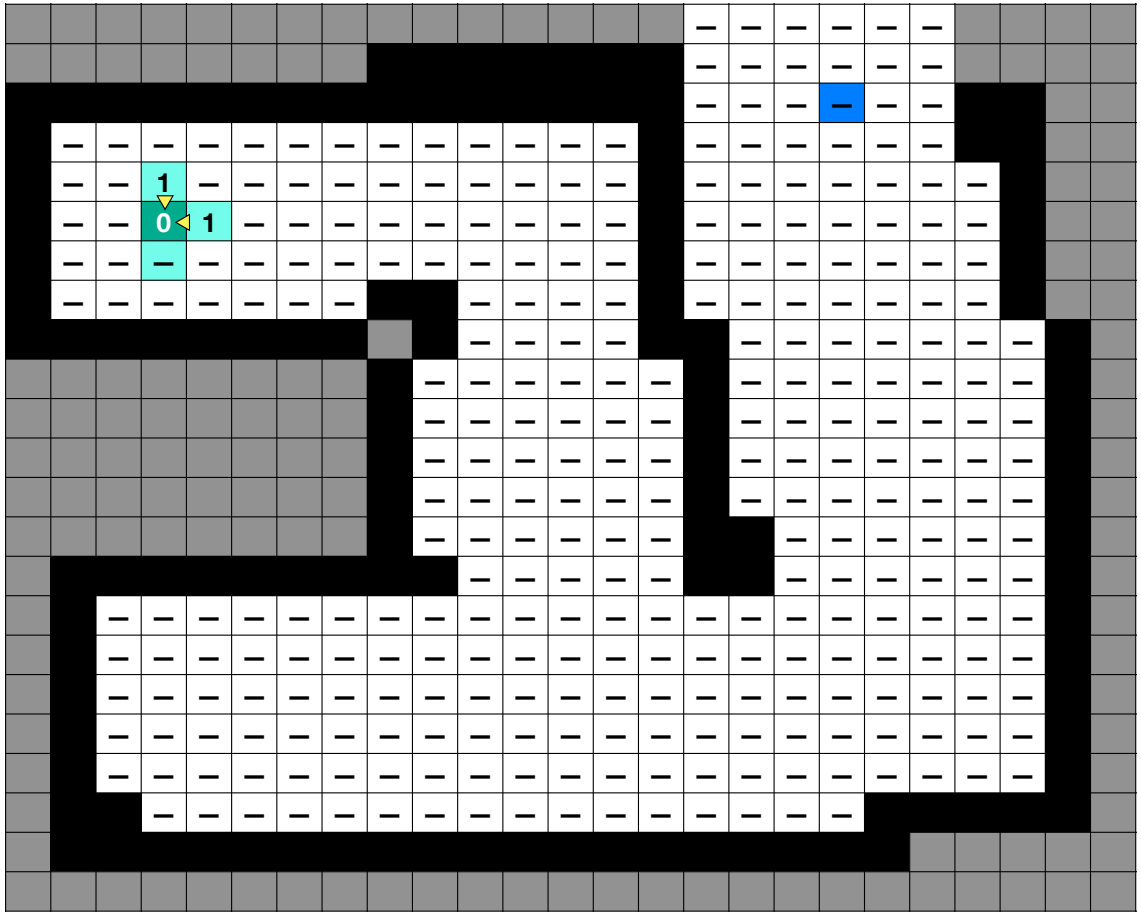


visit_queue

B A

current_node

Queue third neighbor

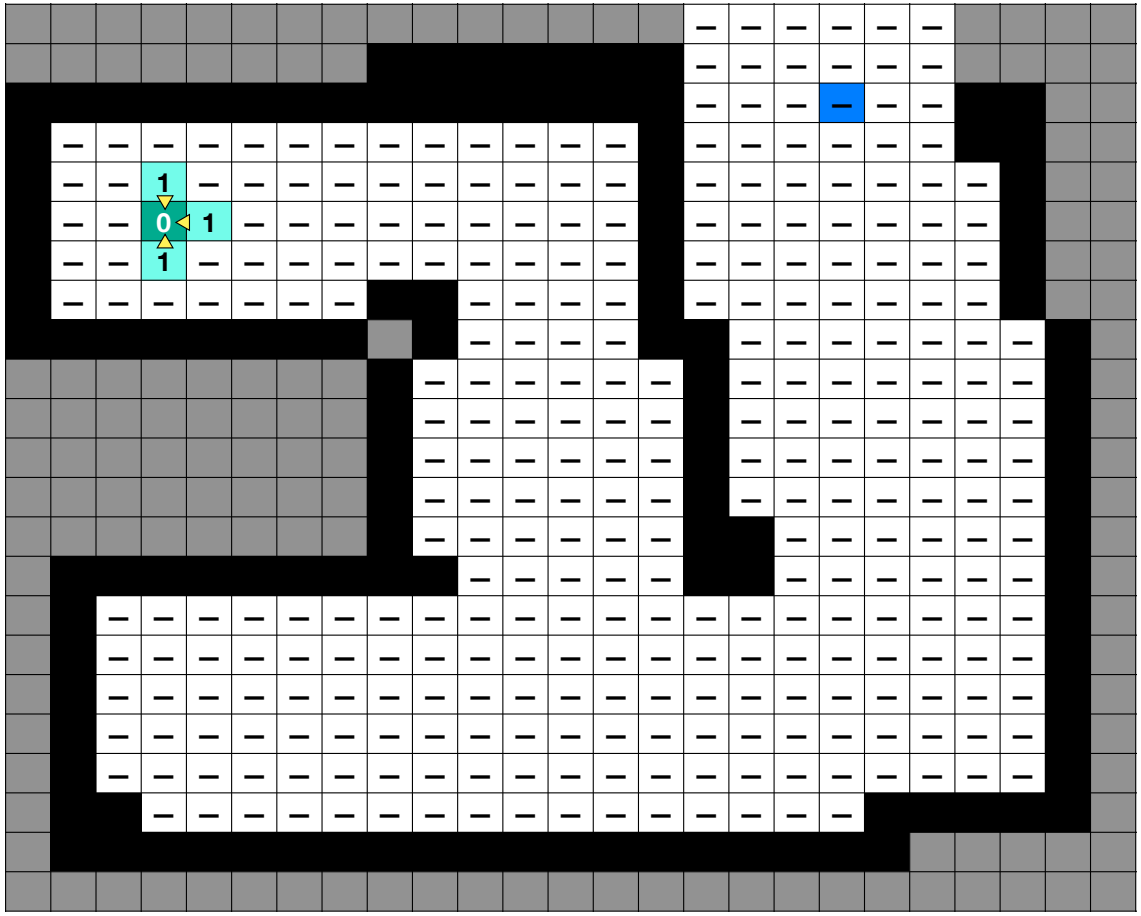


visit_queue

C B A

current_node

Assign distance and parent

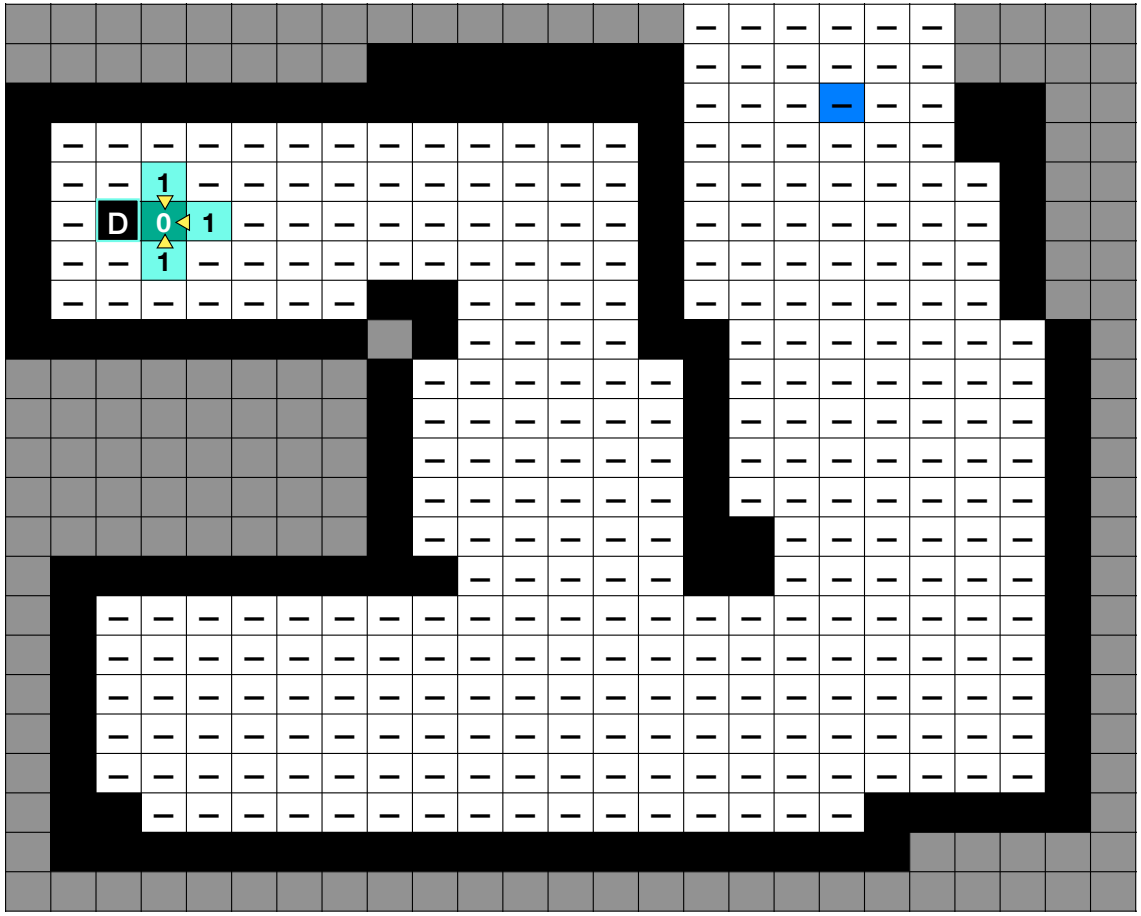


visit_queue

C B A

current_node

Queue last neighbor

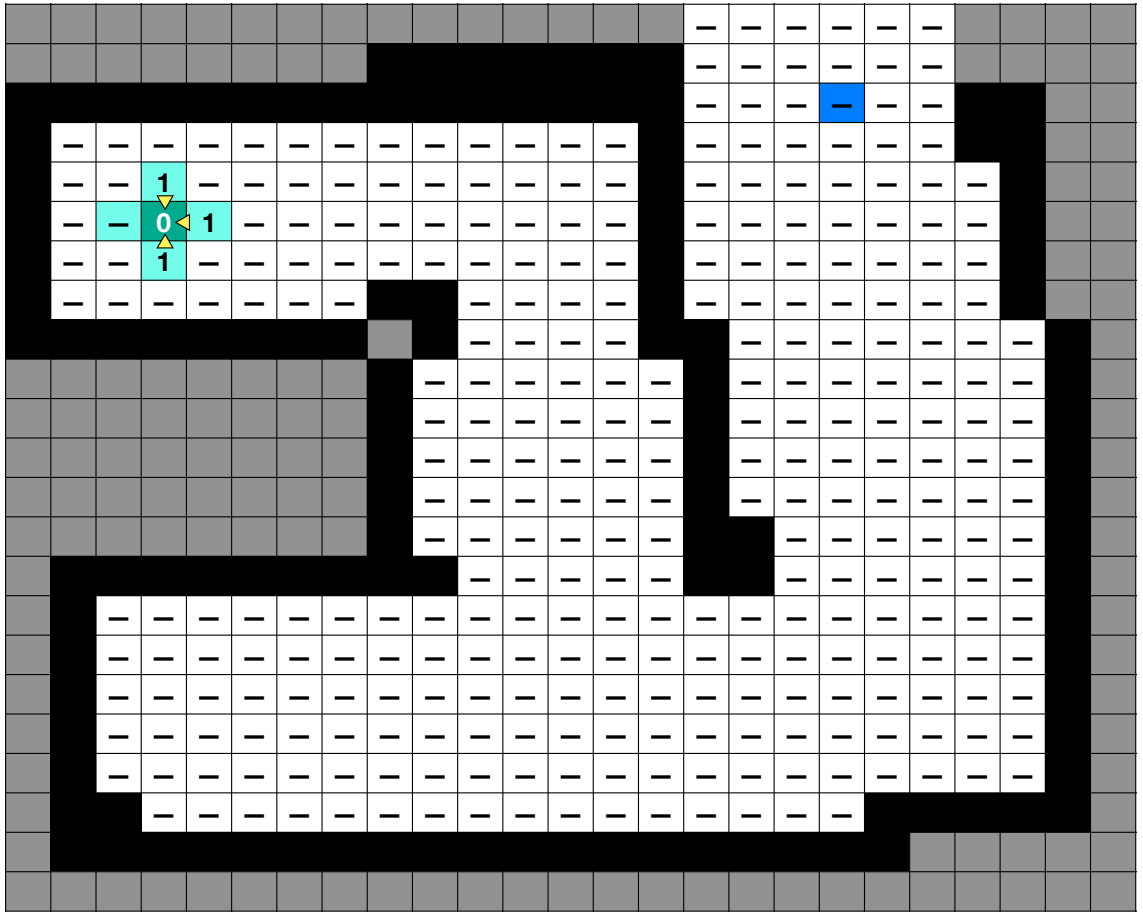


visit_queue

C B A

current_node

Queue last neighbor

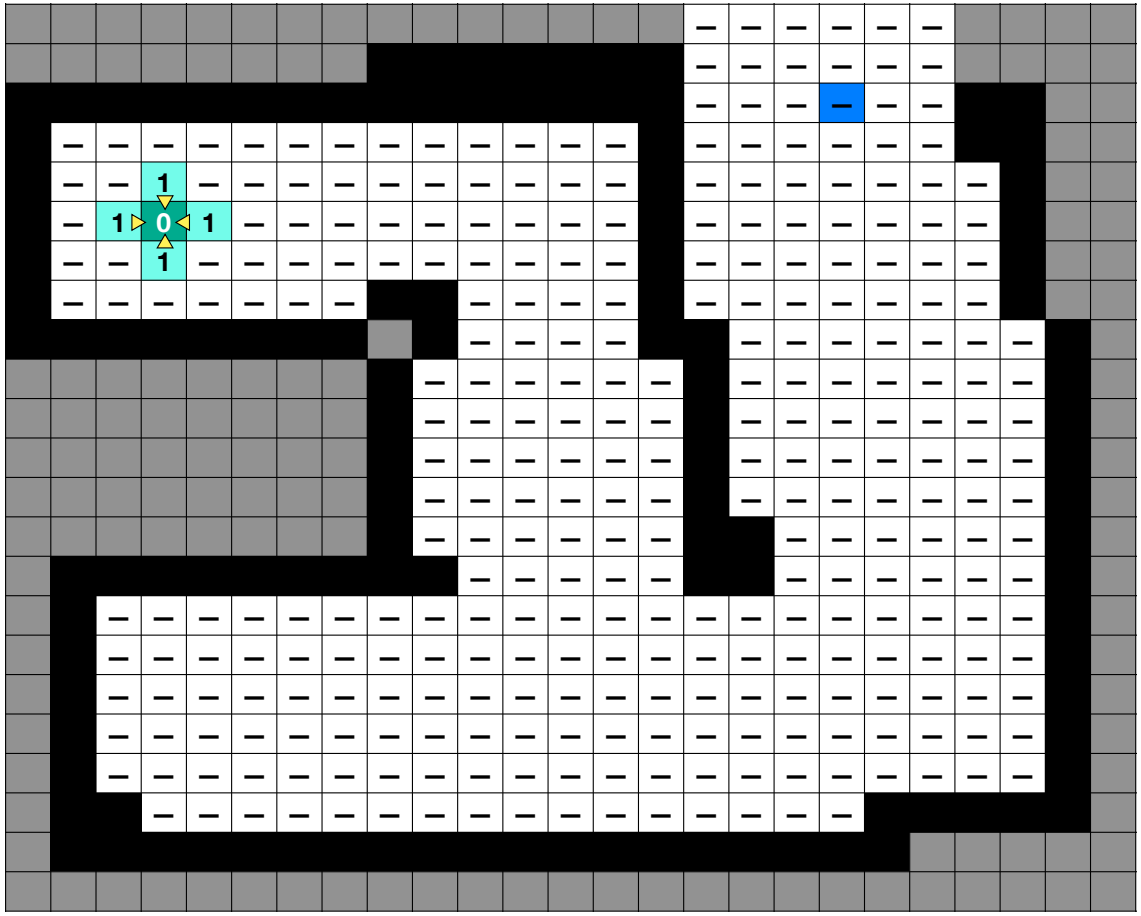


visit_queue

D C B A

current_node

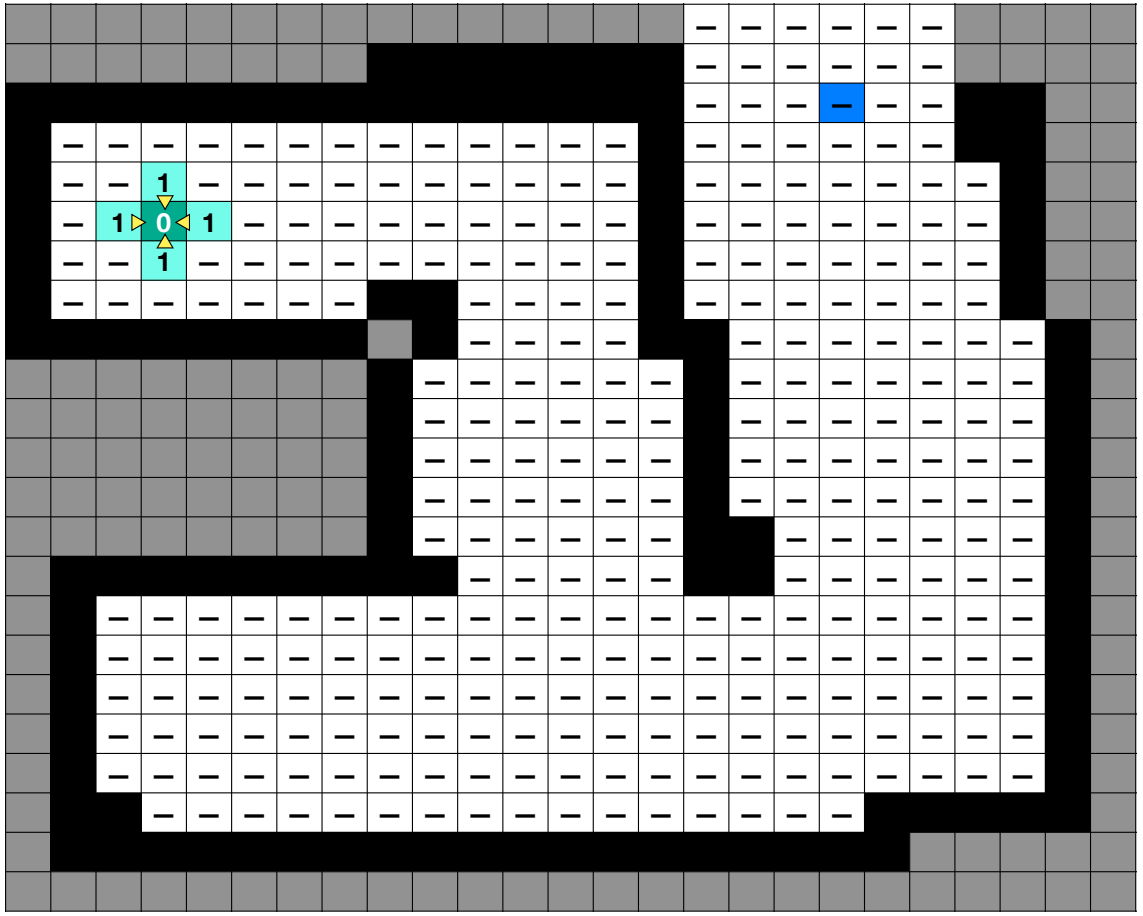
Assign distance and parent



visit_queue

D C B A

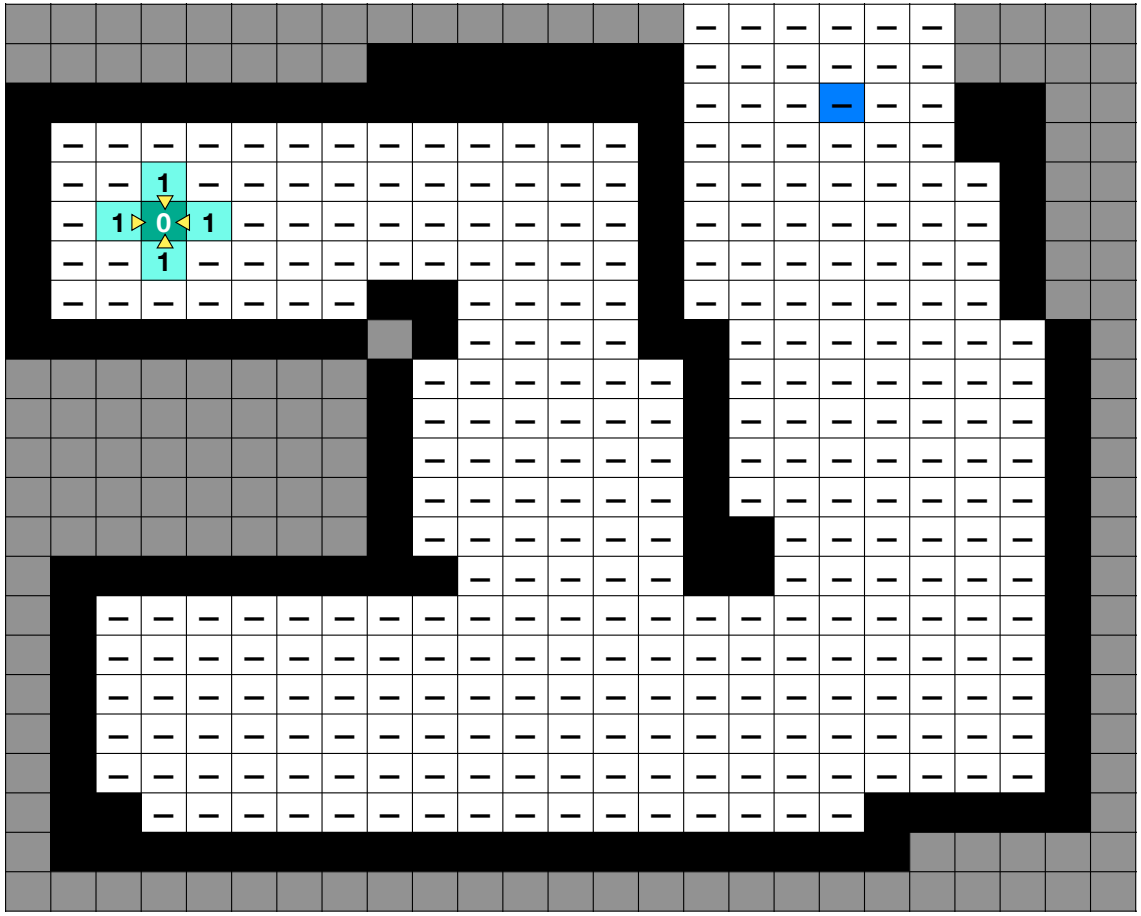
current_node



visit_queue

current_node

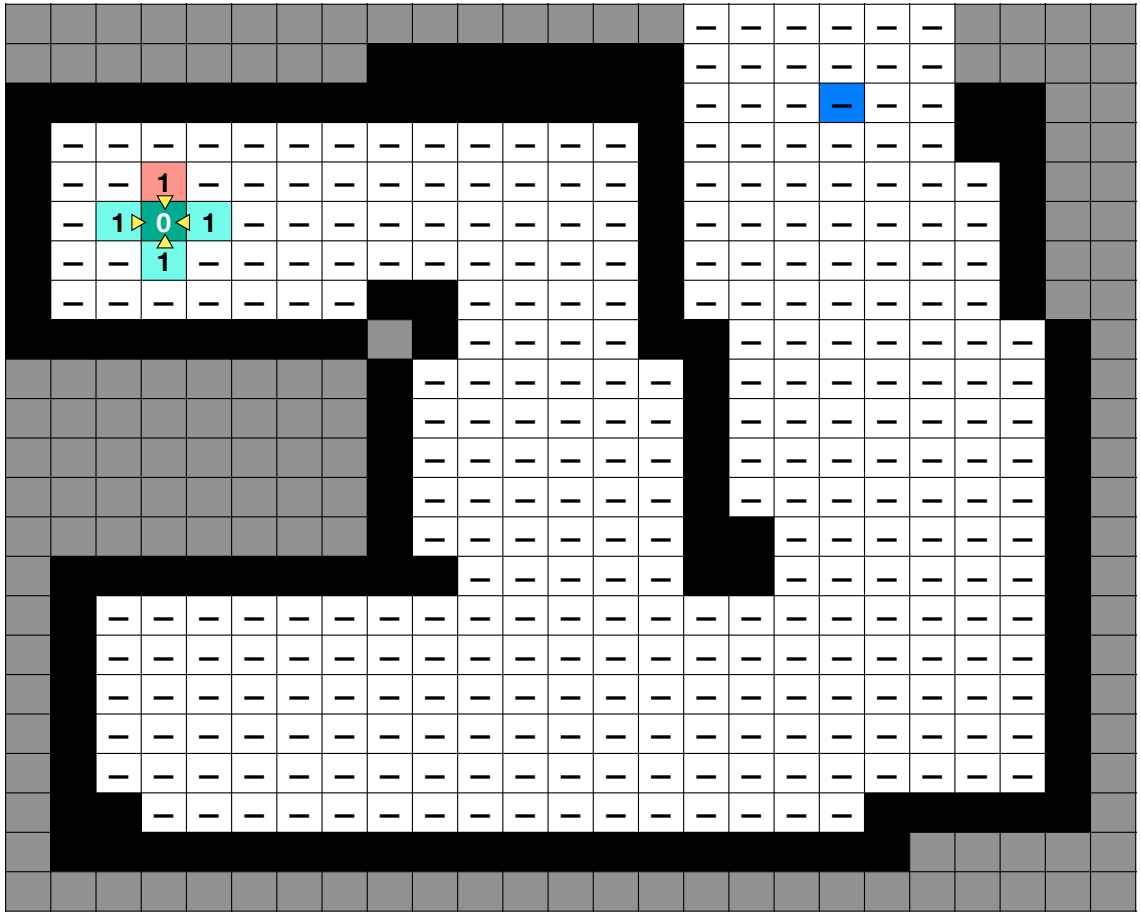
Current node fully processed



visit_queue

D C B A

current_node



visit_queue

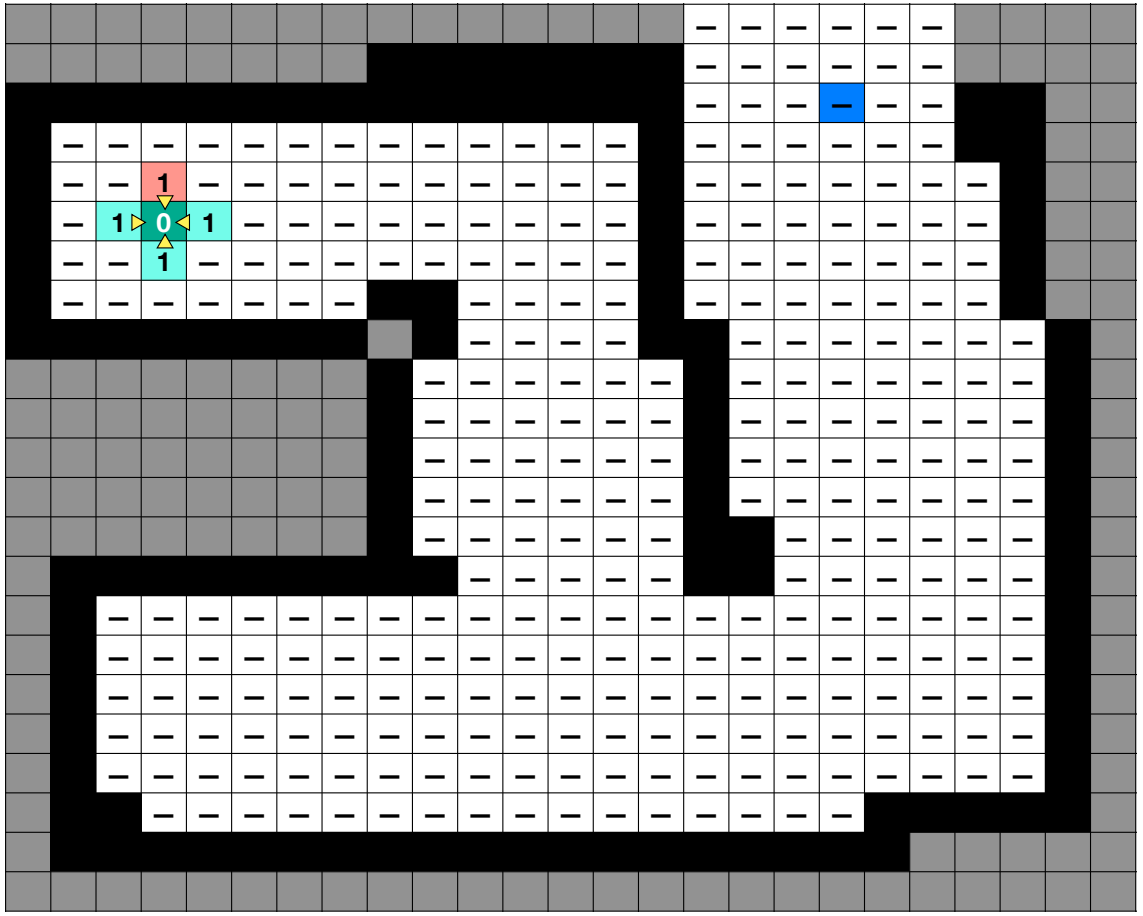
D C B

A

current_node

Dequeue from visit queue

Process current node



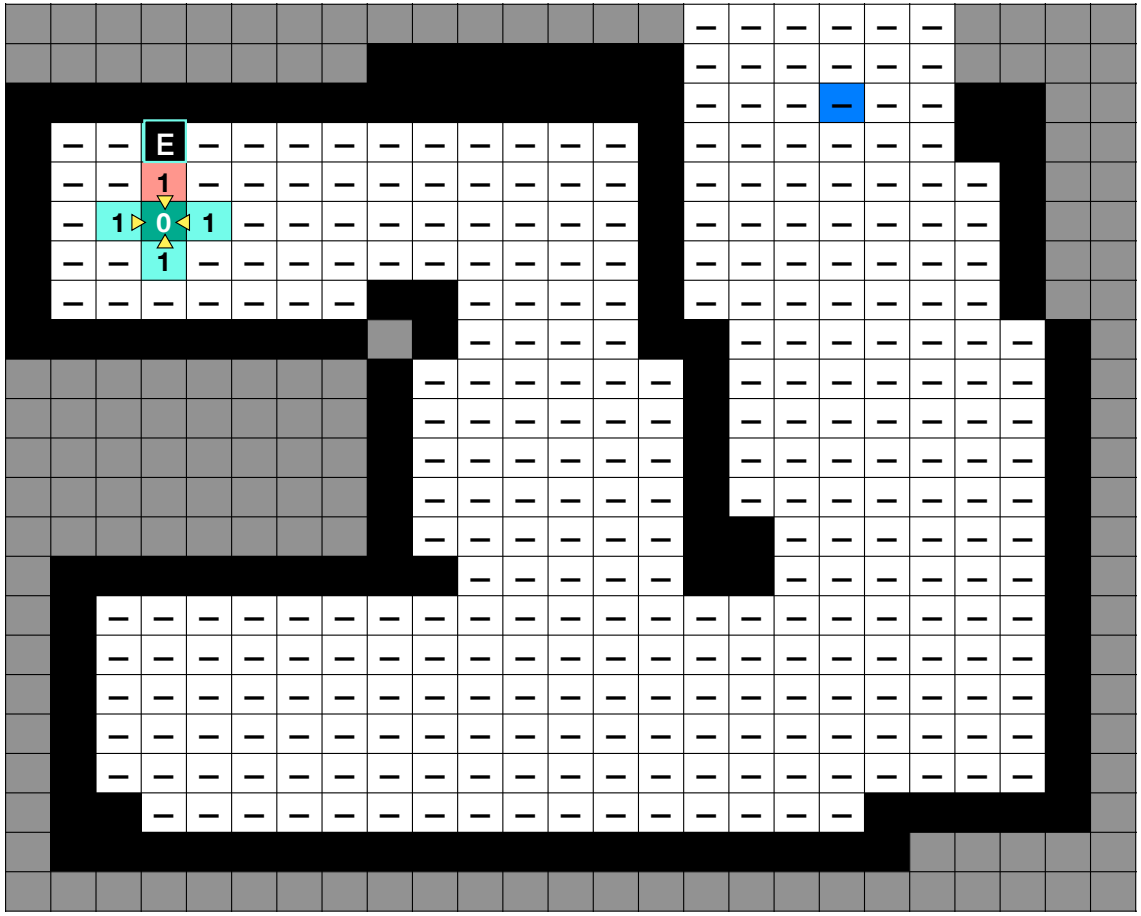
visit_queue

current_node

Dequeue from visit queue

Process current node

Queue first neighbor

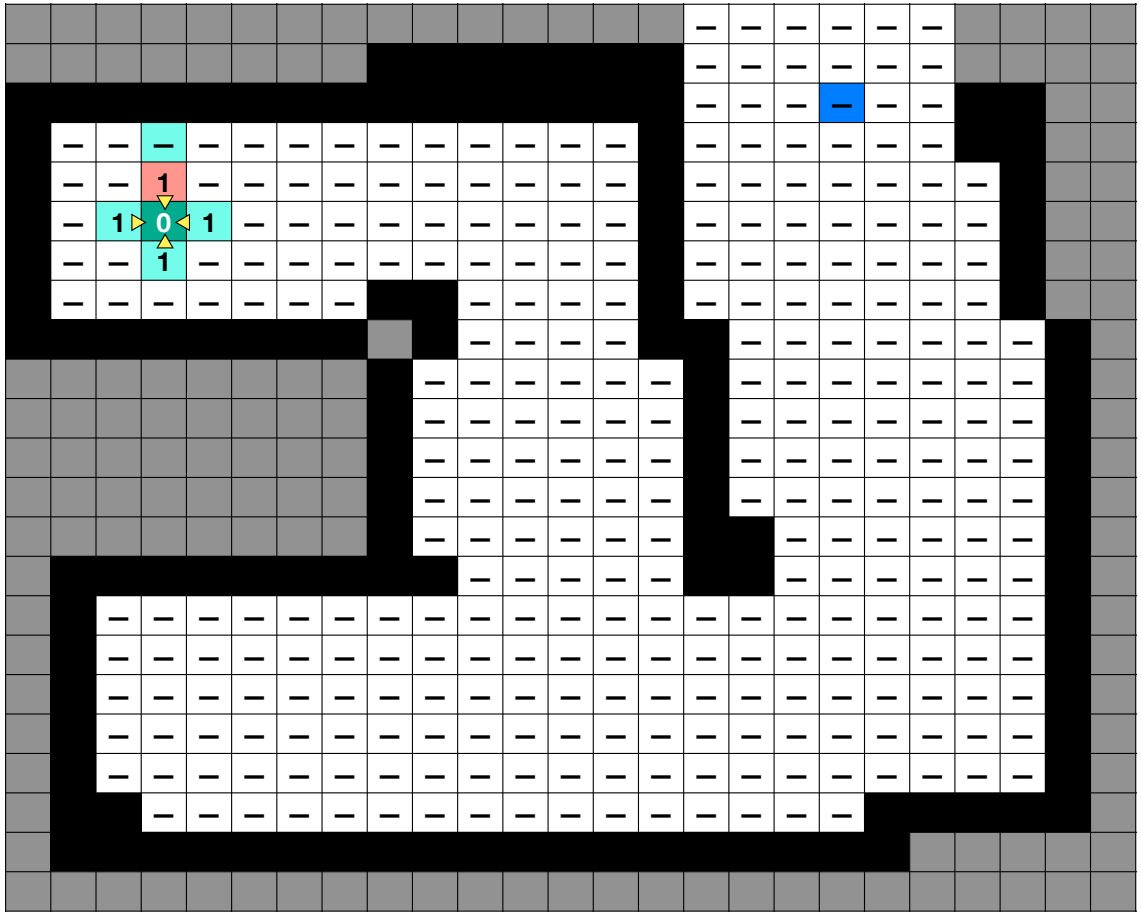


`visit_queue`

`current_node`

Process current node

Queue first neighbor



`visit_queue`

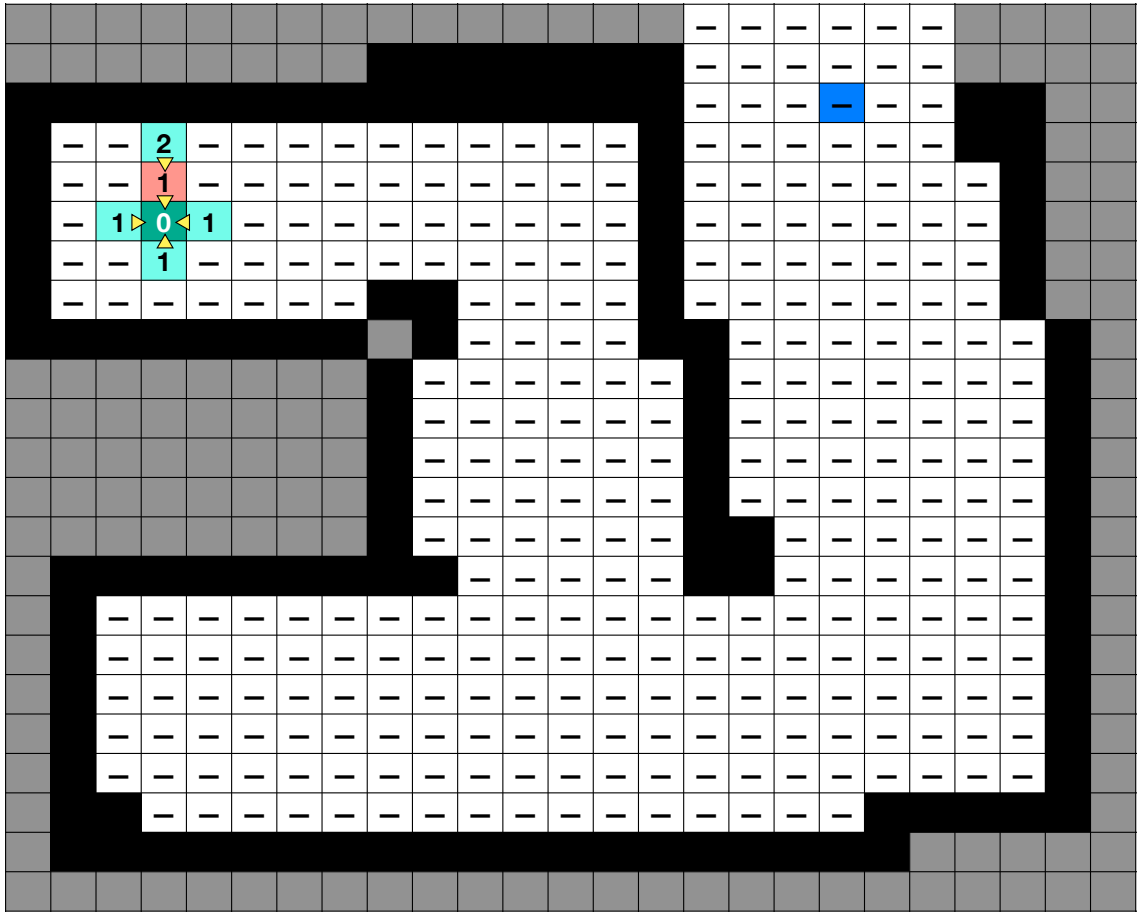
E D C B

A
`current_node`

Process current node

Queue first neighbor

Assign distance and parent

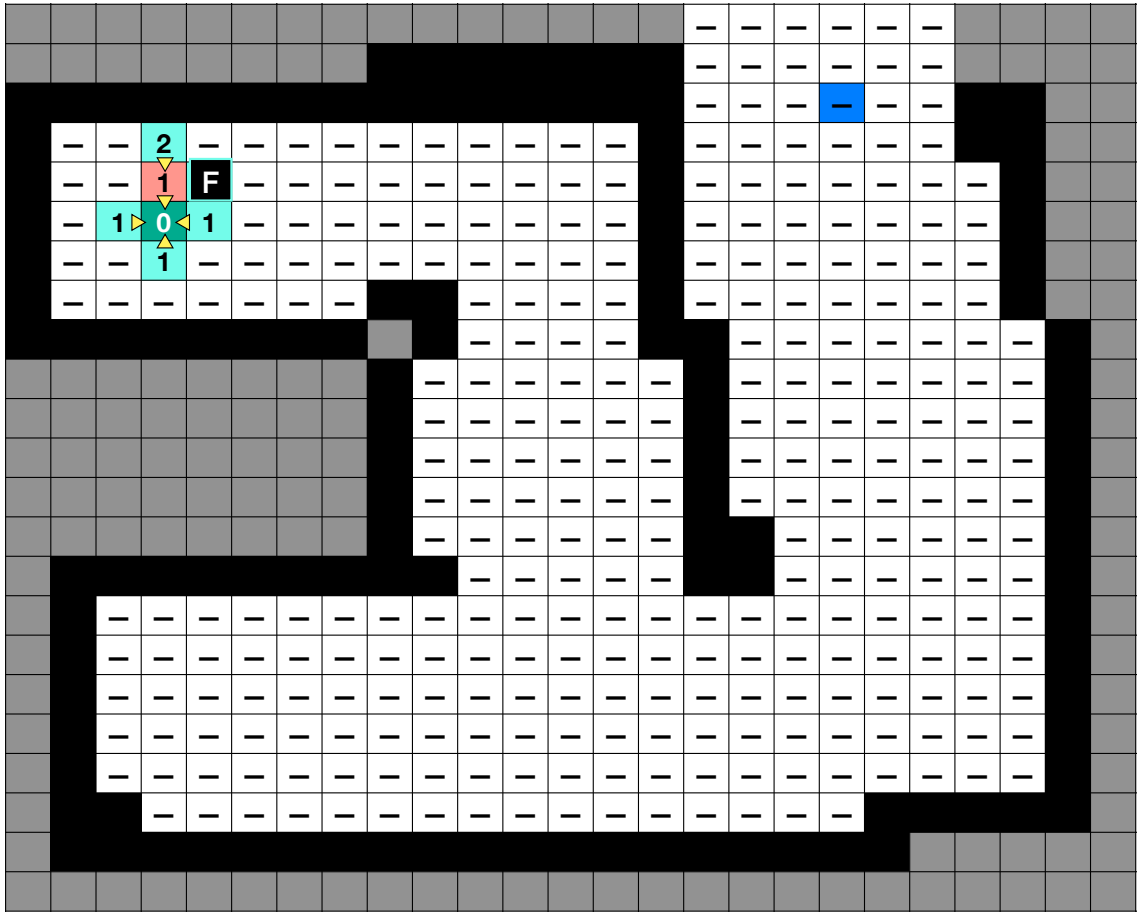


`visit_queue`

`current_node`

Process current node

Queue second neighbor



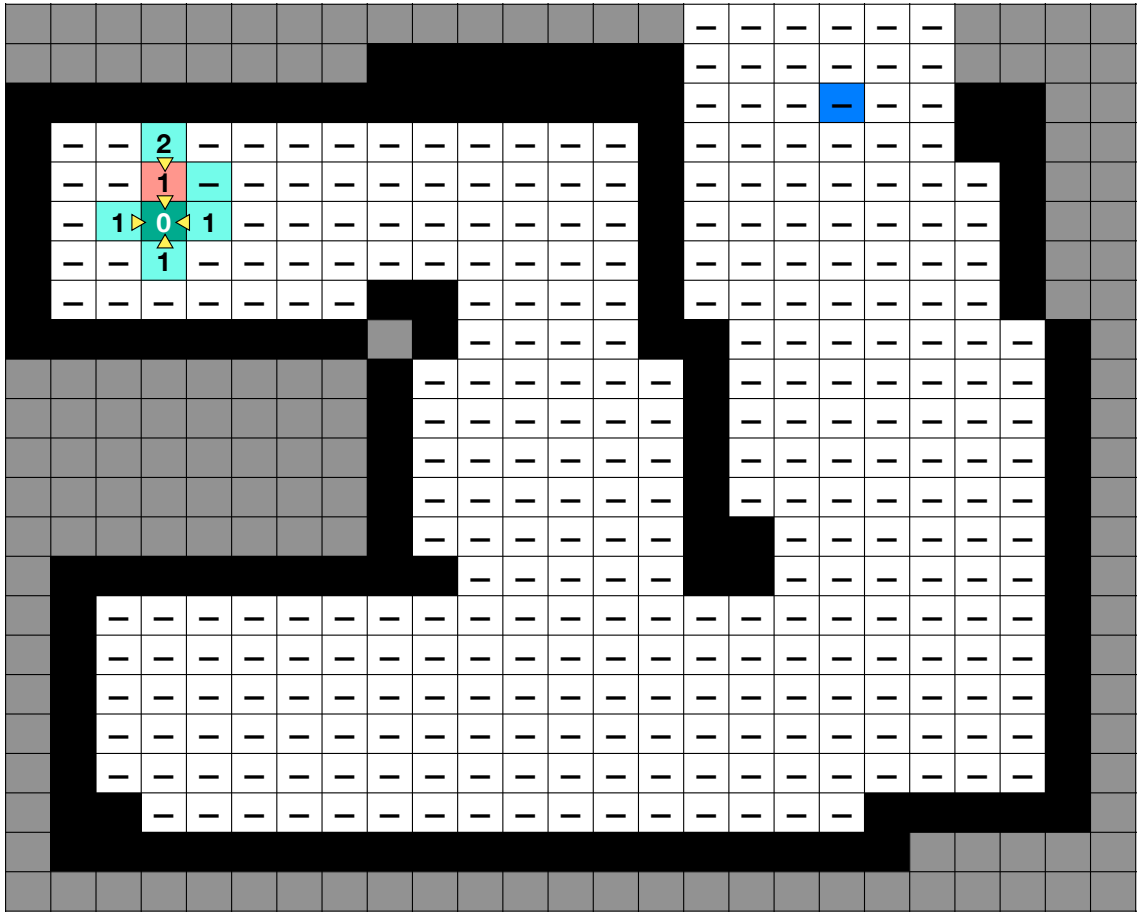
`visit_queue`

E D C B

A
`current_node`

Process current node

Queue second neighbor



`visit_queue`

F E D C B

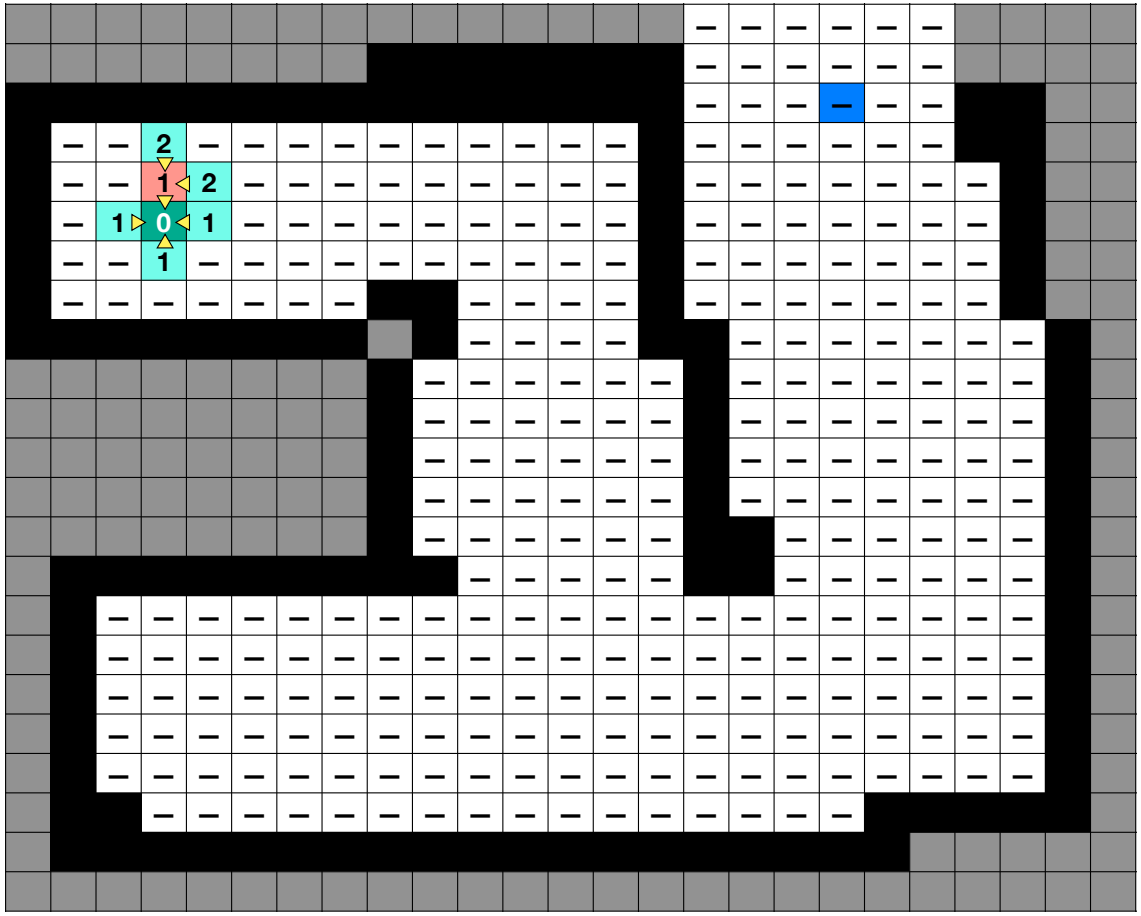
A

`current_node`

Process current node

Queue second neighbor

Assign distance and parent



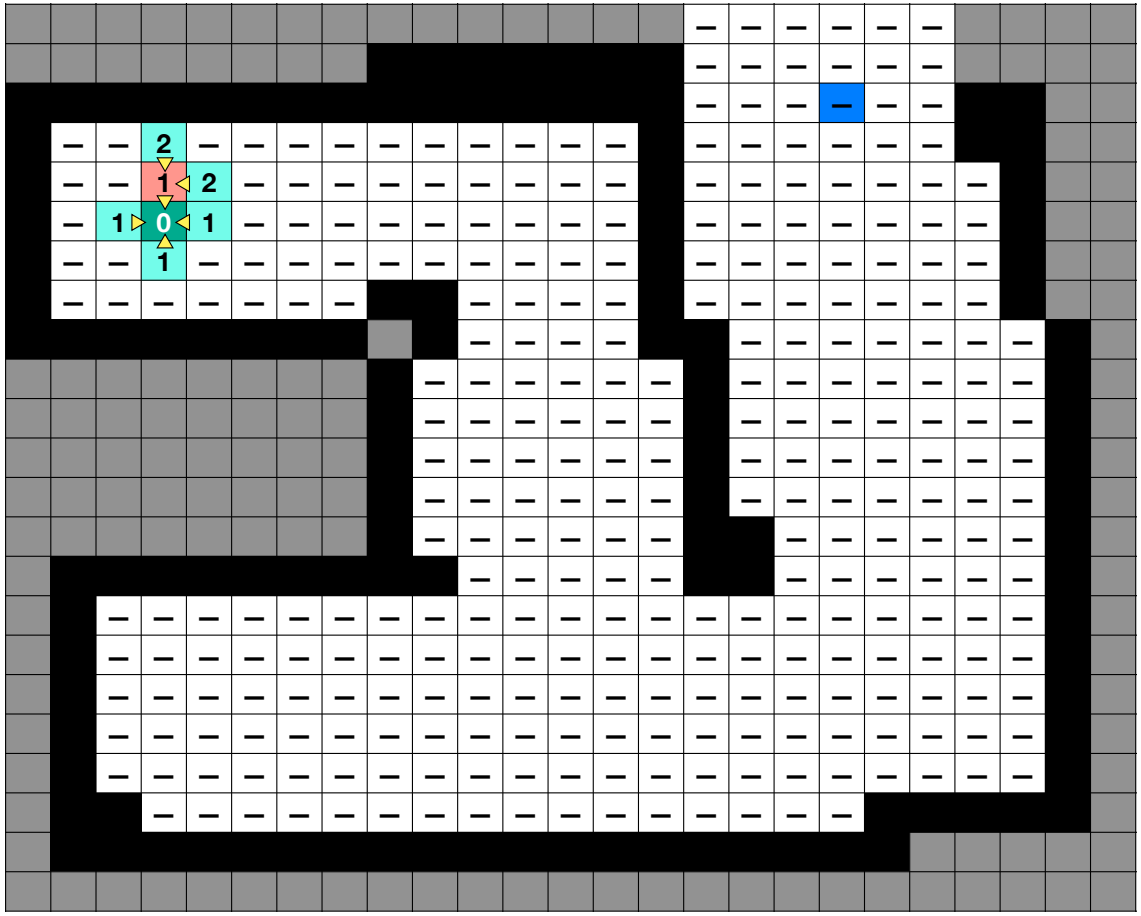
`visit_queue`

F E D C B

A
`current_node`

Process current node

Do not queue third neighbor (already visited)



visit_queue

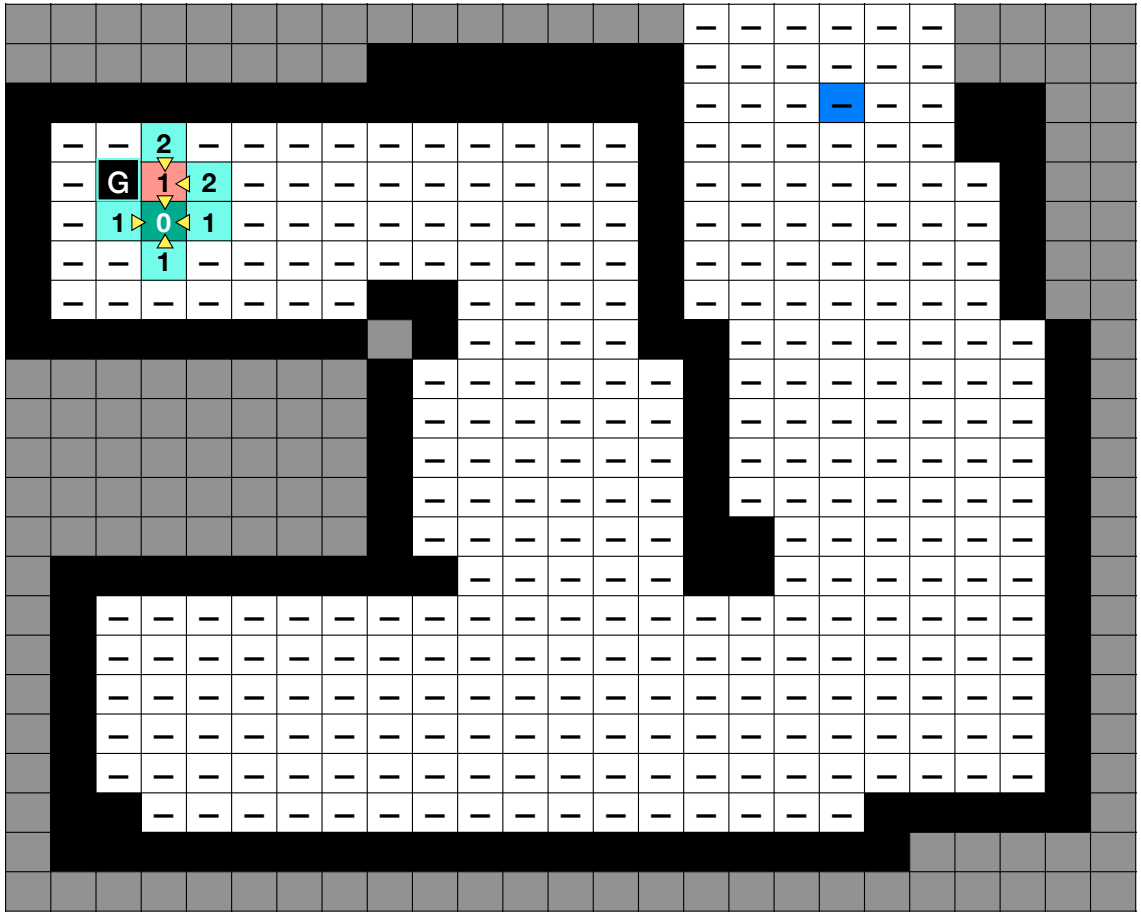
F E D C B

A

current_node

Process current node

Queue last neighbor



`visit_queue`

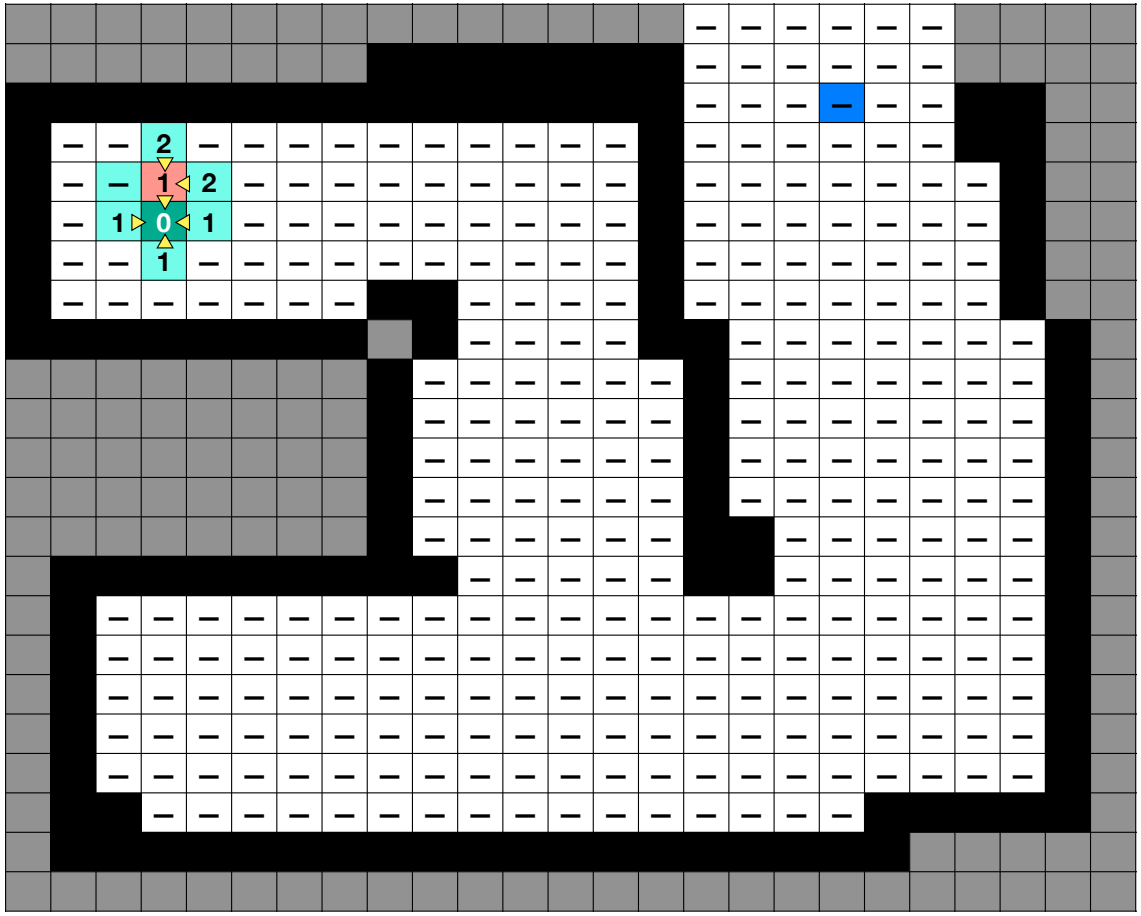
F E D C B

A

`current_node`

Process current node

Queue last neighbor



`visit_queue`

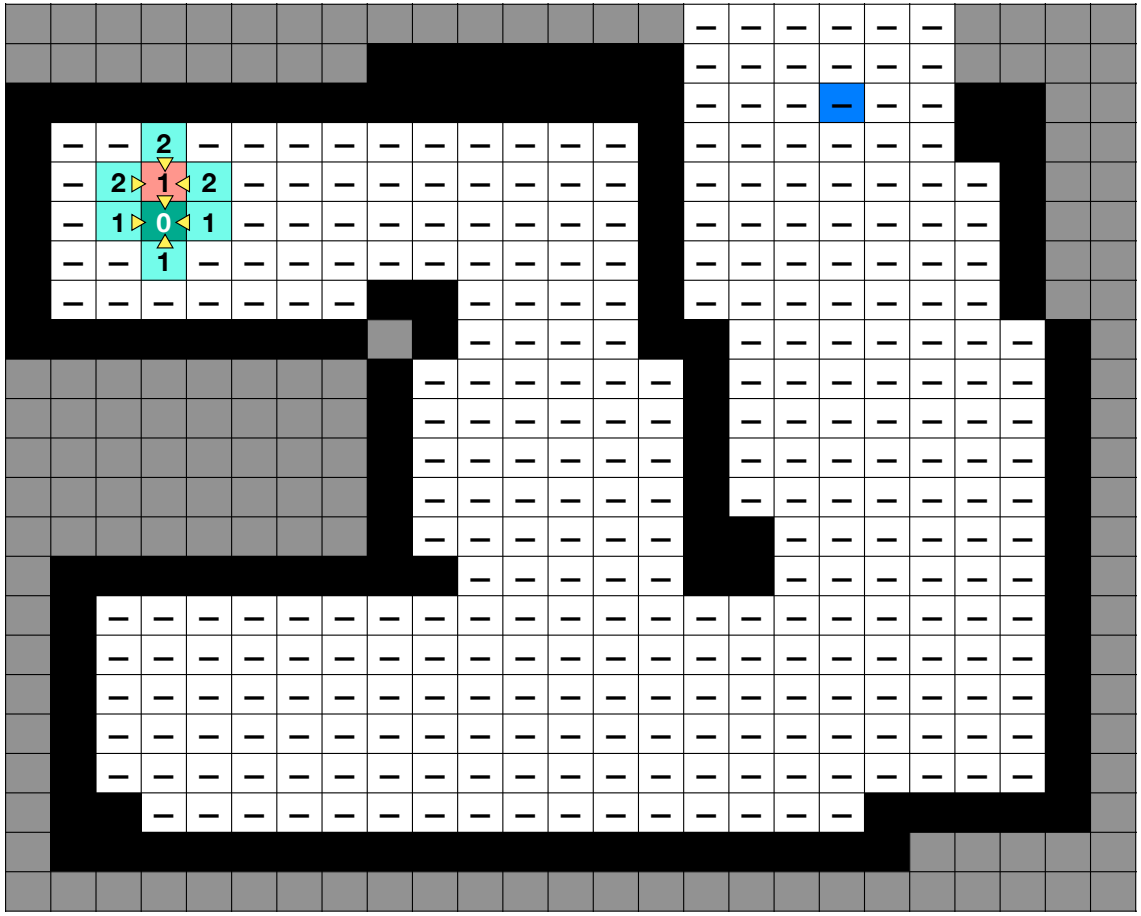
A

`current_node`

Process current node

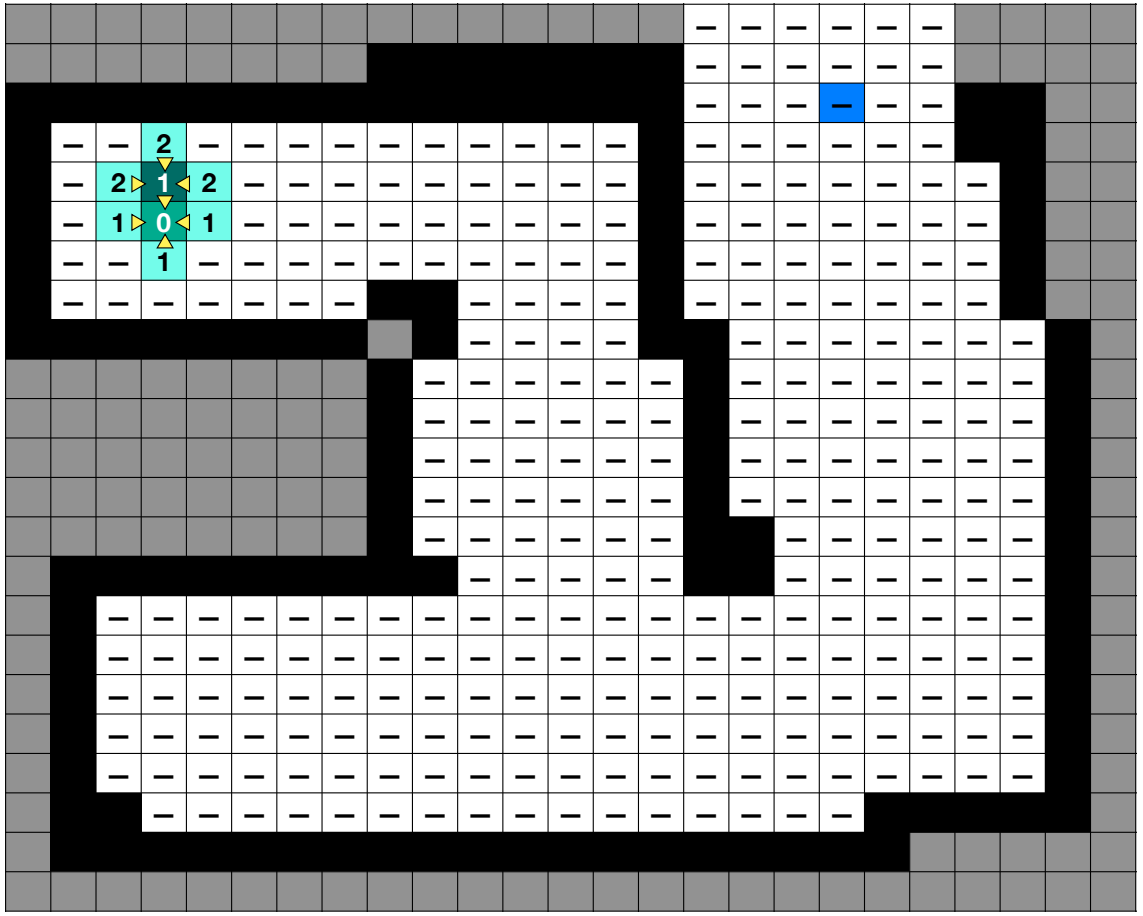
Queue last neighbor

Assign distance and parent



`visit_queue`

A
`current_node`

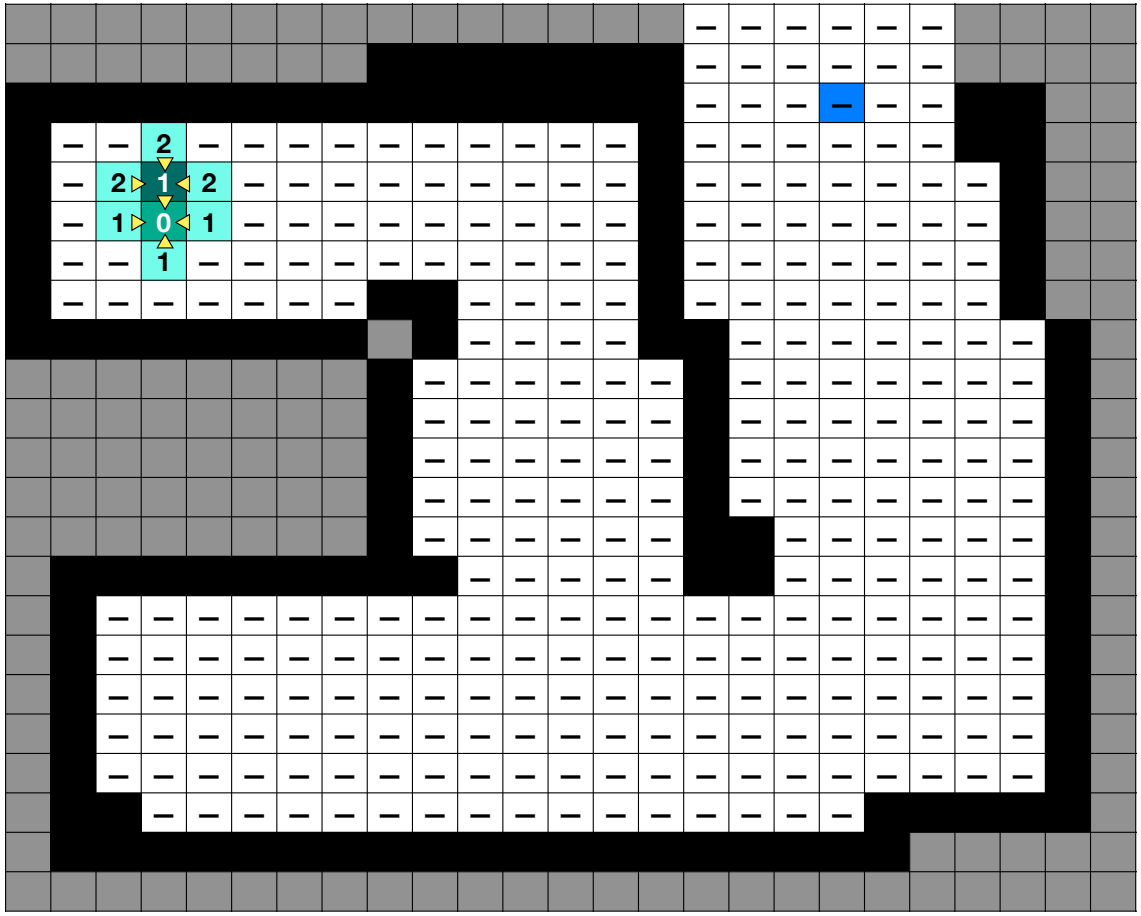


visit_queue

G F E D C B

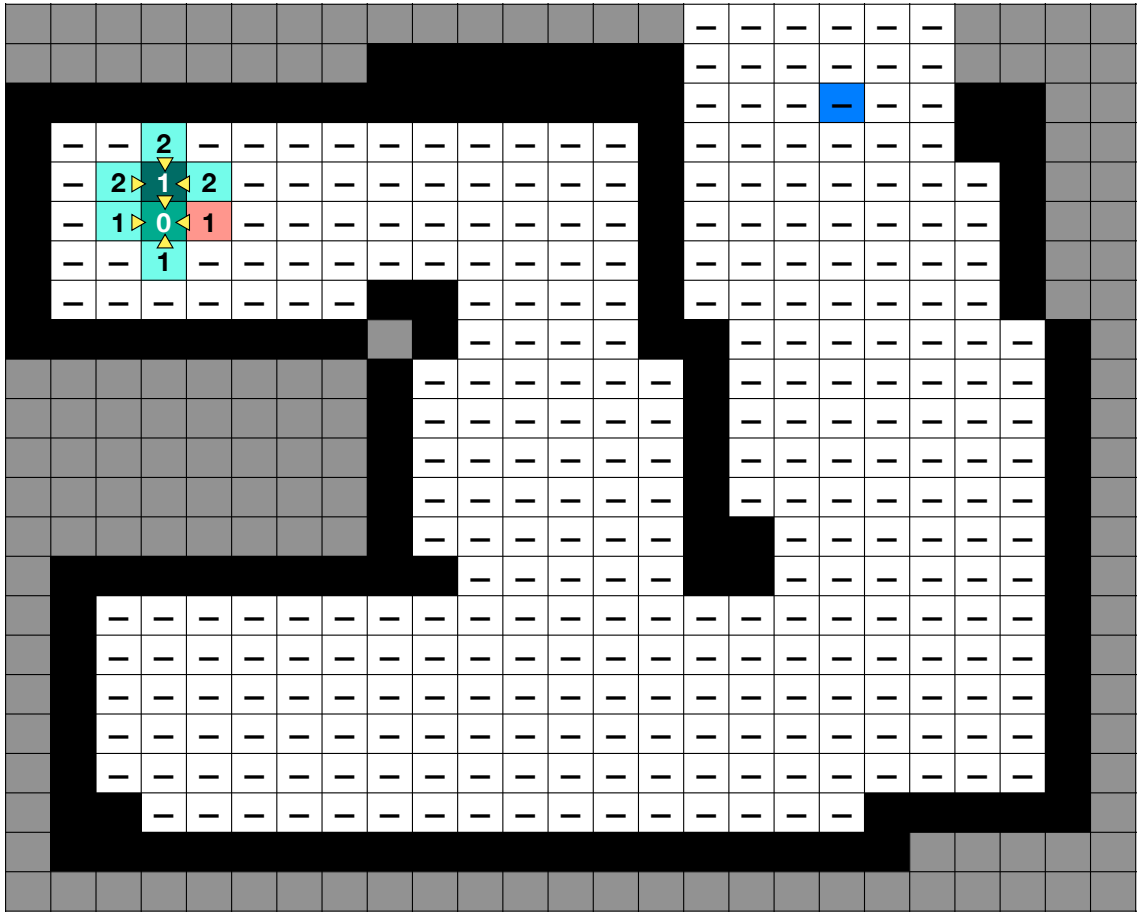
A
current_node

Current node fully processed



visit_queue

current_node



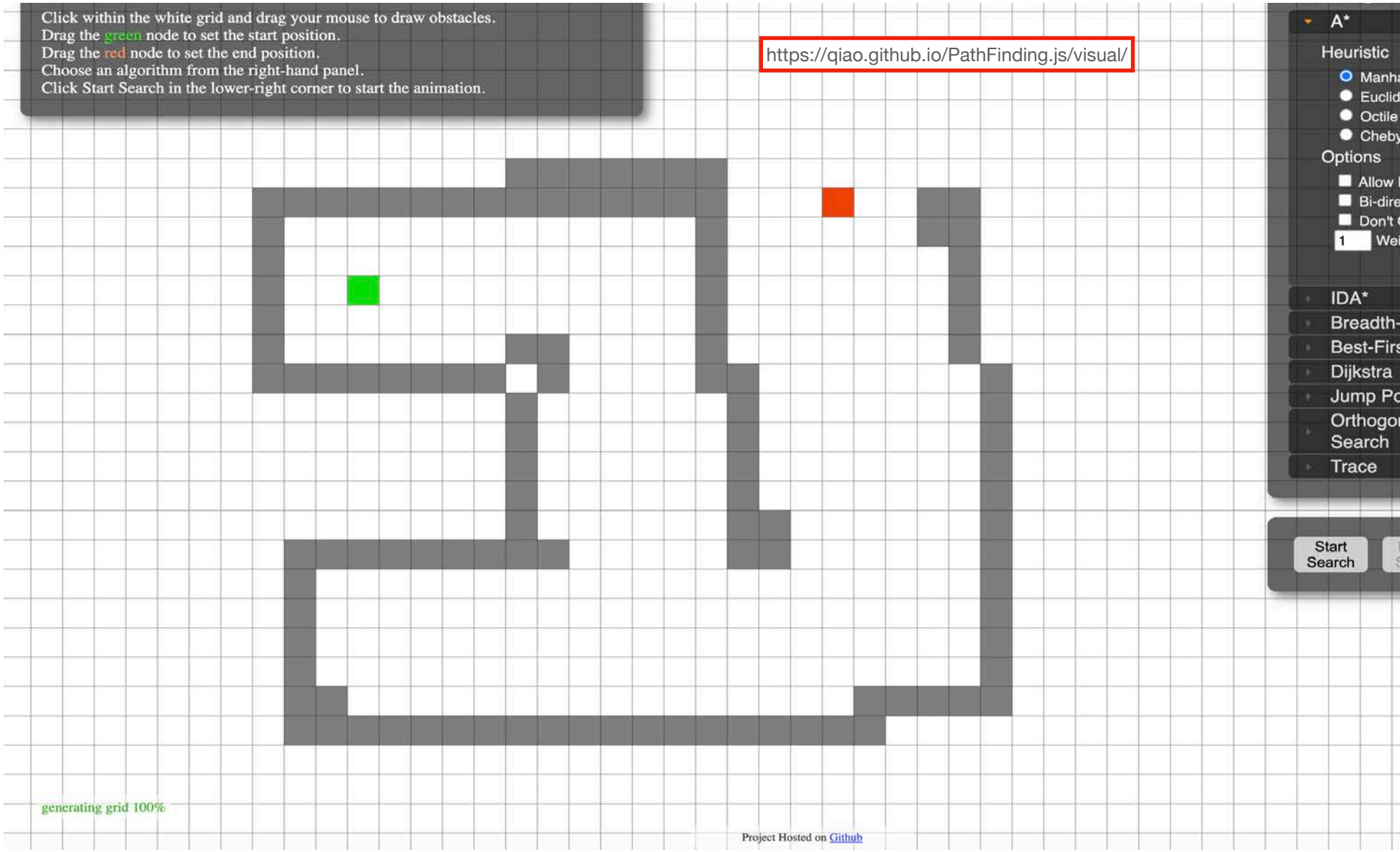
visit_queue

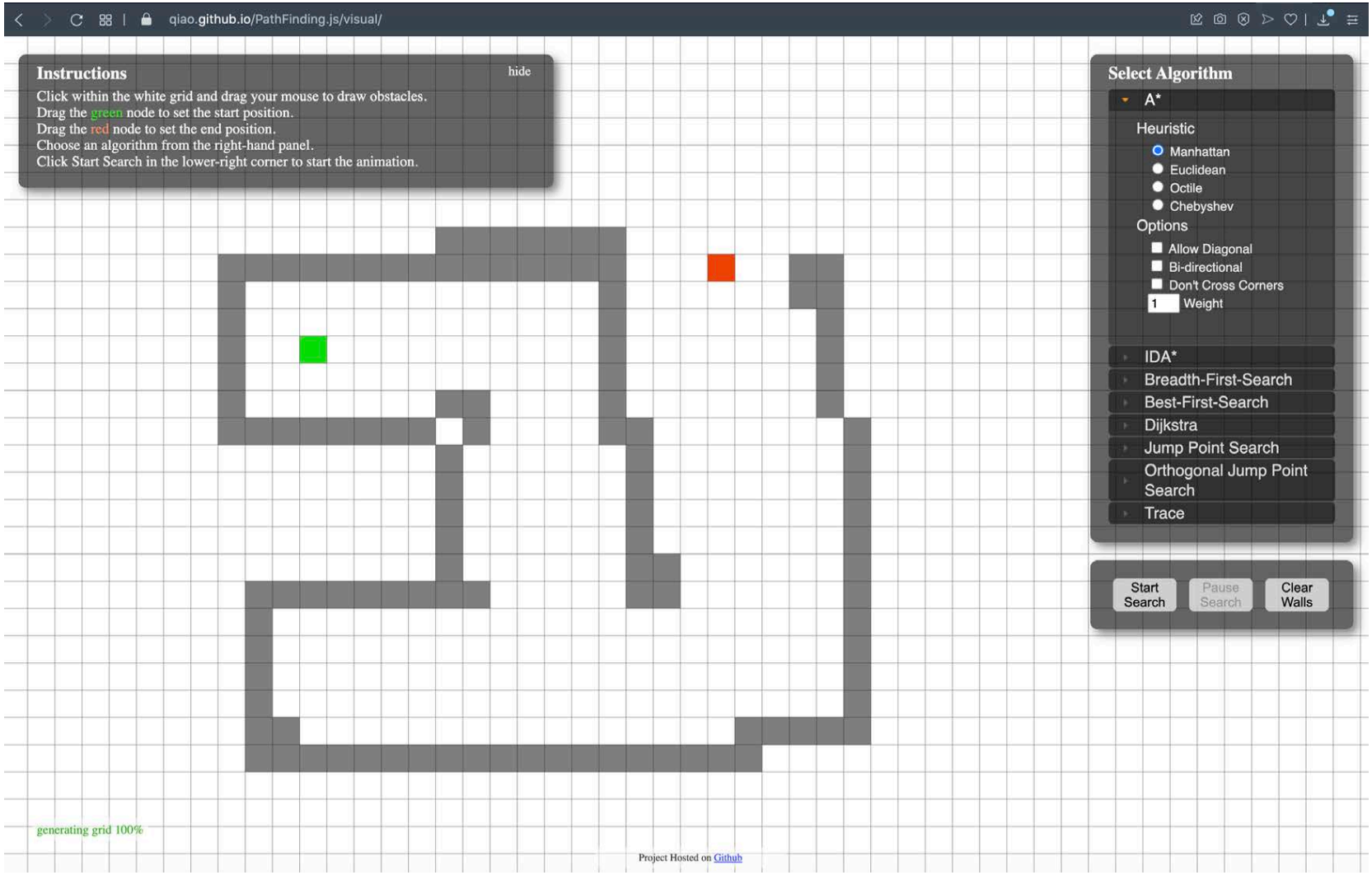
G F E D C

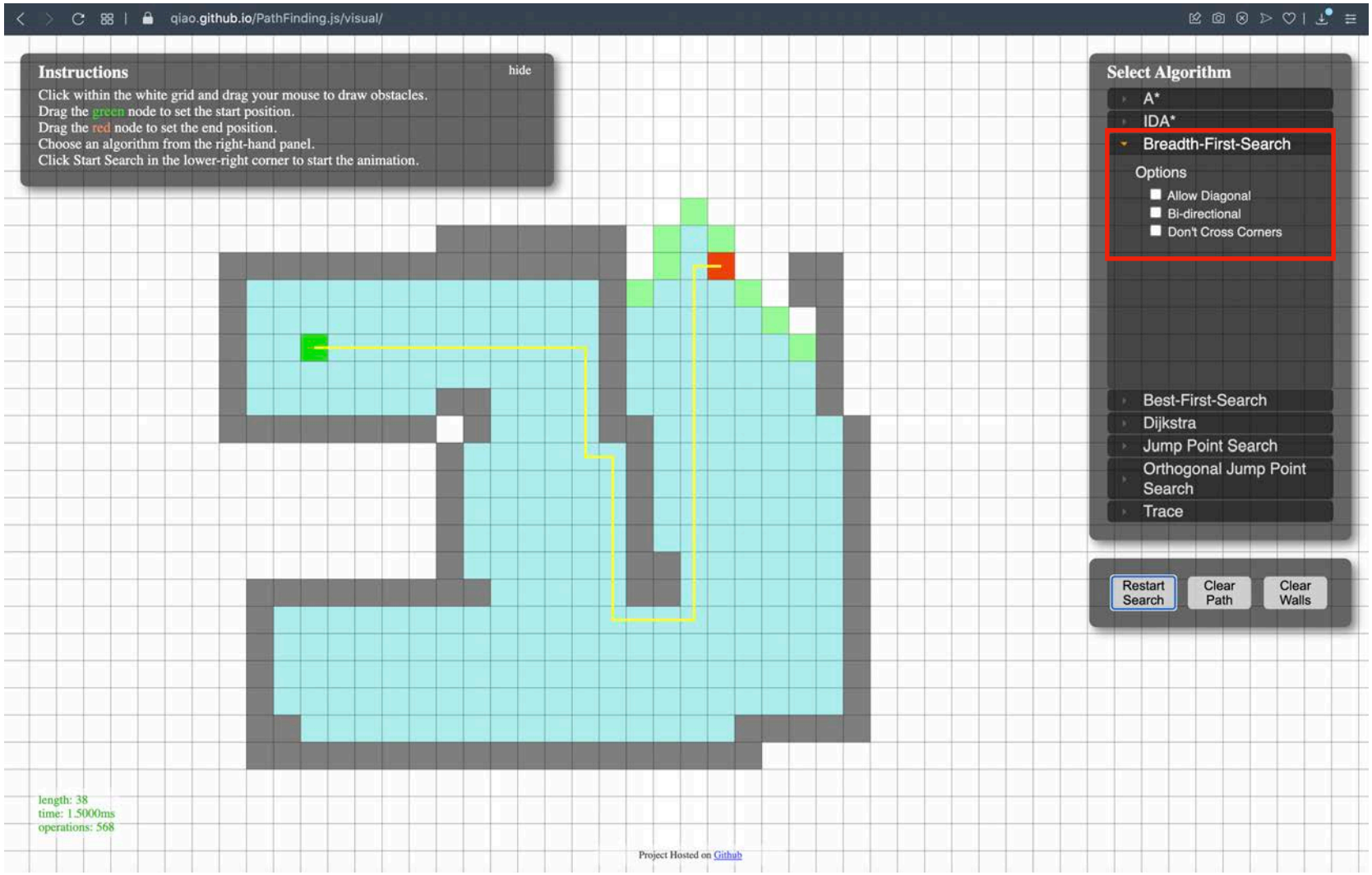
B

current_node

Dequeue from visit queue ... process continues


Click within the white grid and drag your mouse to draw obstacles.
Drag the green node to set the start position.
Drag the red node to set the end position.
Choose an algorithm from the right-hand panel.
Click Start Search in the lower-right corner to start the animation.
https://qiao.github.io/PathFinding.js/visual/
A*
Heuristic
Manha
Euclid
Octile
Cheby
Options
Allow
Bi-dire
Don't
1
Wei
IDA*
Breadth-
Best-Firs
Dijkstra
Jump Po
Orthogo
Search
Trace
Start
Search
generating grid 100%
Project Hosted on Github


qiao.github.io/PathFinding.js/visual/
Instructions
hide
Click within the white grid and drag your mouse to draw obstacles.
Drag the green node to set the start position.
Drag the red node to set the end position.
Choose an algorithm from the right-hand panel.
Click Start Search in the lower-right corner to start the animation.
Select Algorithm
A*
Heuristic
Manhattan
Euclidean
Octile
Chebyshev
Options
Allow Diagonal
Bi-directional
Don't Cross Corners
1
Weight
IDA*
Breadth-First-Search
Best-First-Search
Dijkstra
Jump Point Search
Orthogonal Jump Point Search
Trace
Start Search
Pause Search
Clear Walls
generating grid 100%
Project Hosted on Github

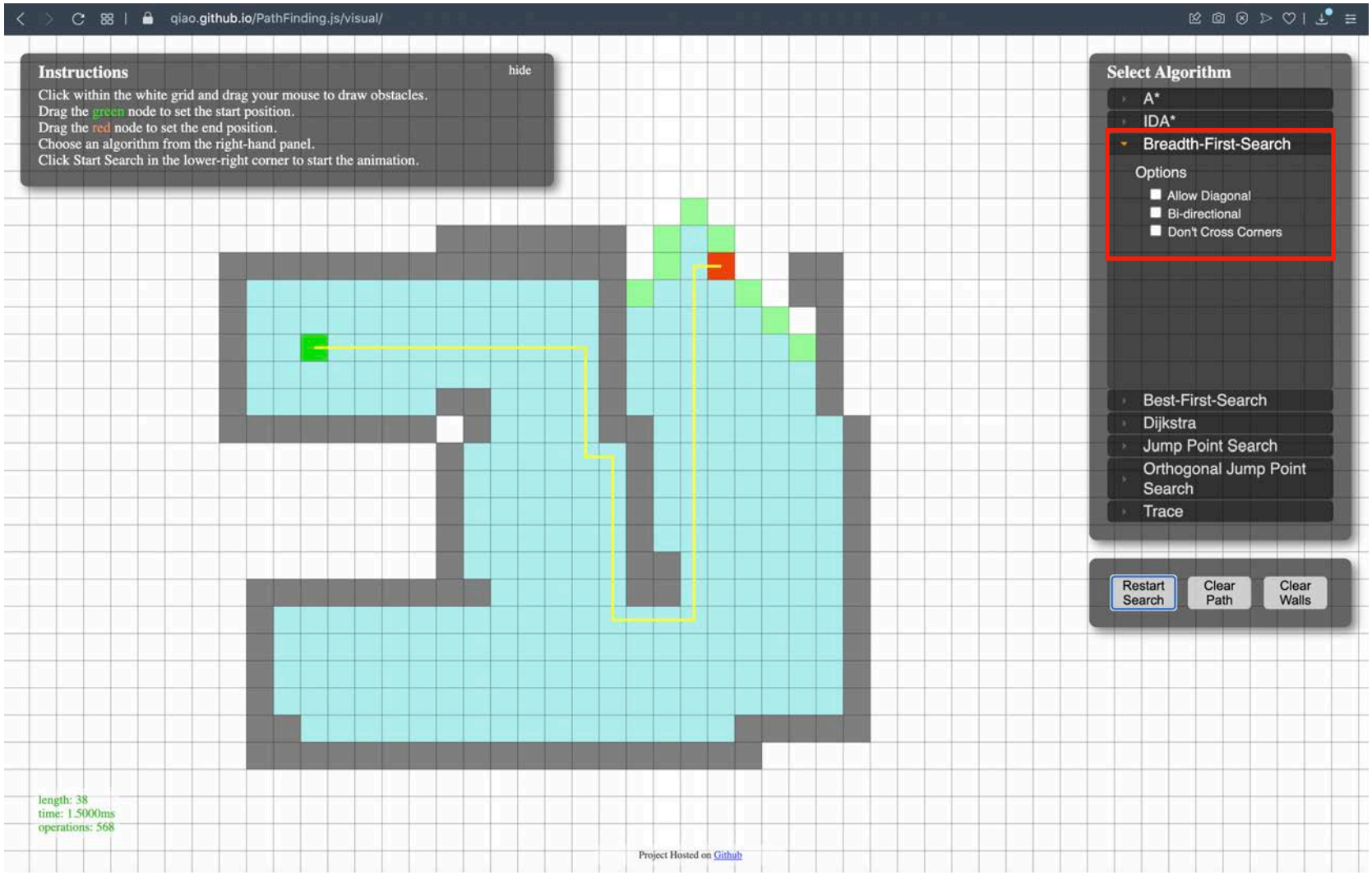

qiao.github.io/PathFinding.js/visual/
Instructions
hide
Click within the white grid and drag your mouse to draw obstacles.
Drag the green node to set the start position.
Drag the red node to set the end position.
Choose an algorithm from the right-hand panel.
Click Start Search in the lower-right corner to start the animation.
Select Algorithm
A*
IDA*
Breadth-First-Search
Options
Allow Diagonal
Bi-directional
Don't Cross Corners
Best-First-Search
Dijkstra
Jump Point Search
Orthogonal Jump Point Search
Trace
Restart Search
Clear Path
Clear Walls
length: 38
time: 1.5000ms
operations: 568
Project Hosted on Github

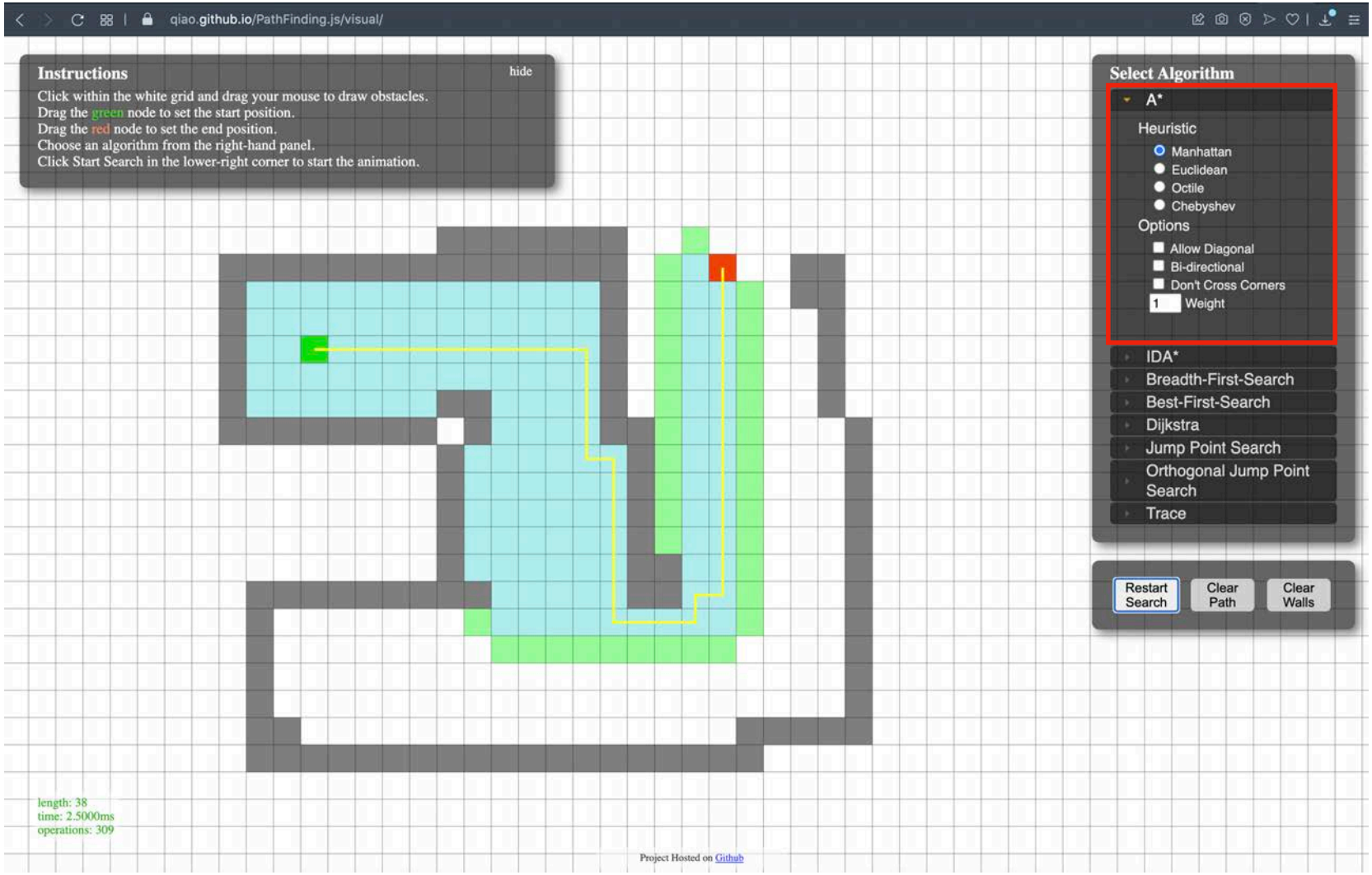
Considerations for BFS

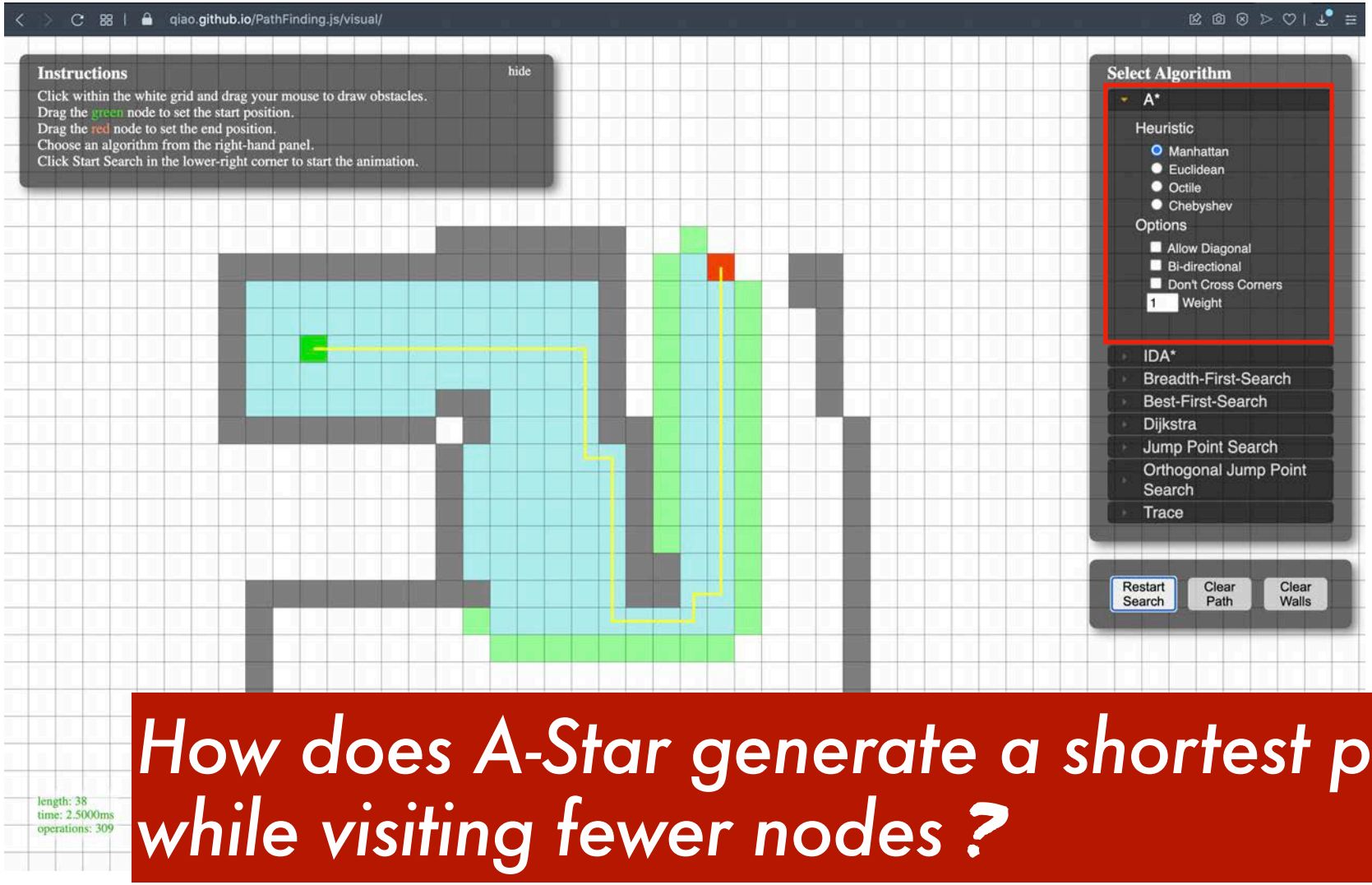
Is any path that reaches the goal a good path?

How many cells do we need to visit?

Could we use another visit strategy?


qiao.github.io/PathFinding.js/visual/
Instructions
hide
Click within the white grid and drag your mouse to draw obstacles.
Drag the green node to set the start position.
Drag the red node to set the end position.
Choose an algorithm from the right-hand panel.
Click Start Search in the lower-right corner to start the animation.
Select Algorithm
A*
IDA*
Breadth-First-Search
Options
Allow Diagonal
Bi-directional
Don't Cross Corners
Best-First-Search
Dijkstra
Jump Point Search
Orthogonal Jump Point Search
Trace
Restart Search
Clear Path
Clear Walls
length: 38
time: 1.5000ms
operations: 568
Project Hosted on Github


qiao.github.io/PathFinding.js/visual/
Instructions
hide
Click within the white grid and drag your mouse to draw obstacles.
Drag the green node to set the start position.
Drag the red node to set the end position.
Choose an algorithm from the right-hand panel.
Click Start Search in the lower-right corner to start the animation.
Select Algorithm
A*
Heuristic
Manhattan
Euclidean
Octile
Chebyshev
Options
Allow Diagonal
Bi-directional
Don't Cross Corners
1
Weight
IDA*
Breadth-First-Search
Best-First-Search
Dijkstra
Jump Point Search
Orthogonal Jump Point Search
Trace
Restart Search
Clear Path
Clear Walls
length: 38
time: 2.5000ms
operations: 309
Project Hosted on Github



How does A-Star generate a shortest path while visiting fewer nodes ?

Breadth-first Search

Initialize :

All nodes to have no parent, max distance, and as unvisited

Start node to have no parent and zero distance

Visit queue with start node as its only enqueued element

***Iterate* : While visit list not empty and currently visited node is not the goal**

Dequeue new current node to visit and mark it as visited

For each neighbor :

Add to visit queue, if not previously visited or queued

If Distance of neighbor > Distance of current node + …

Cost to move from current node to neighbor

Then Update neighbor's parent to be current node and …

distance to be distance of current node + cost to move

A-Star Search

Initialize :

All nodes to have no parent, max distance, and as unvisited

Start node to have no parent and zero distance

Visit queue with start node as its only enqueued element

***Iterate* : While visit list not empty and currently visited node is not the goal**

Dequeue new current node to visit and mark it as visited

For each neighbor :

Add to visit priority queue, if not previously visited or queued

If Distance of neighbor > Distance of current node + …

Cost to move from current node to neighbor

Then Update neighbor's parent to be current node and …

distance to be distance of current node + cost to move

A-Star Search

Initialize :

All nodes to have no parent, max distance, and as unvisited

Start node to have no parent and zero distance

Visit queue with start node as its only enqueued element

Iterate : While visit list not empty and currently visited node is not the goal

Dequeue new current node to visit and mark it as visited

For each neighbor :

Add to visit priority queue, if not previously visited or queued

If Distance of neighbor > Distance of current node + …

How do we implement a priority queue?

Then Up… r

What is a node's priority for A-Star?

distance to be distance of current node + cost to move

How do we implement a priority queue?

Topic for later courses in data structures (e.g., Michigan EECS 280)

Foreshadowing: use a binary heap

Considered an advanced extension for Project 3

What is a node's priority for A-Star?

How do we implement a priority queue?

Topic for later courses in data structures (e.g., Michigan EECS 280)

Foreshadowing: use a binary heap

Considered an advanced extension for Project 3

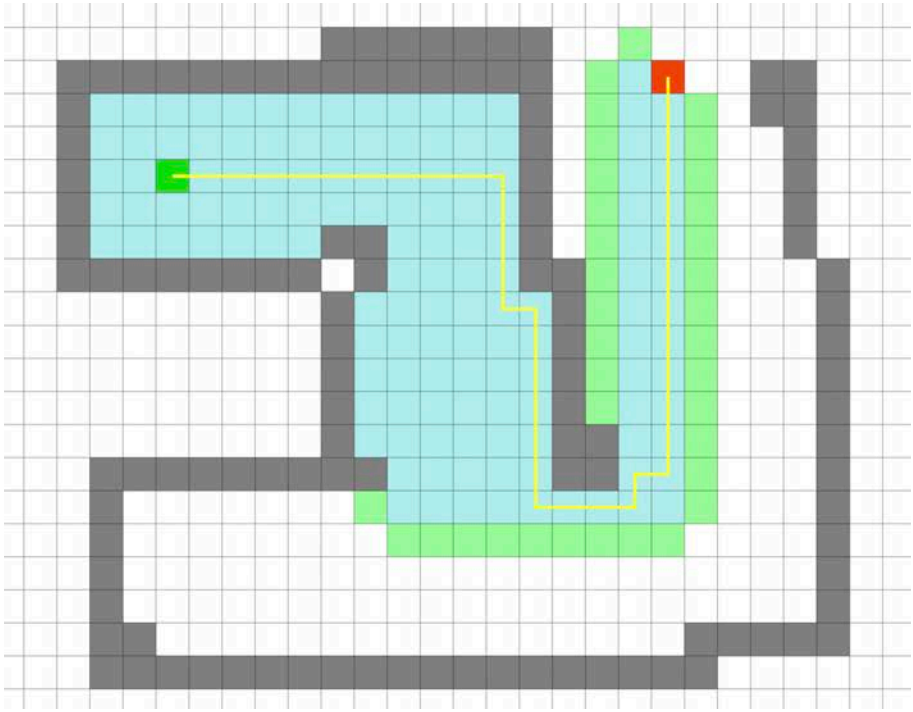
What is a node's priority for A-Star?

Use an optimistic heuristic for the best possible outcome

Each node's priority is distance along path route to start +
best possible distance to goal

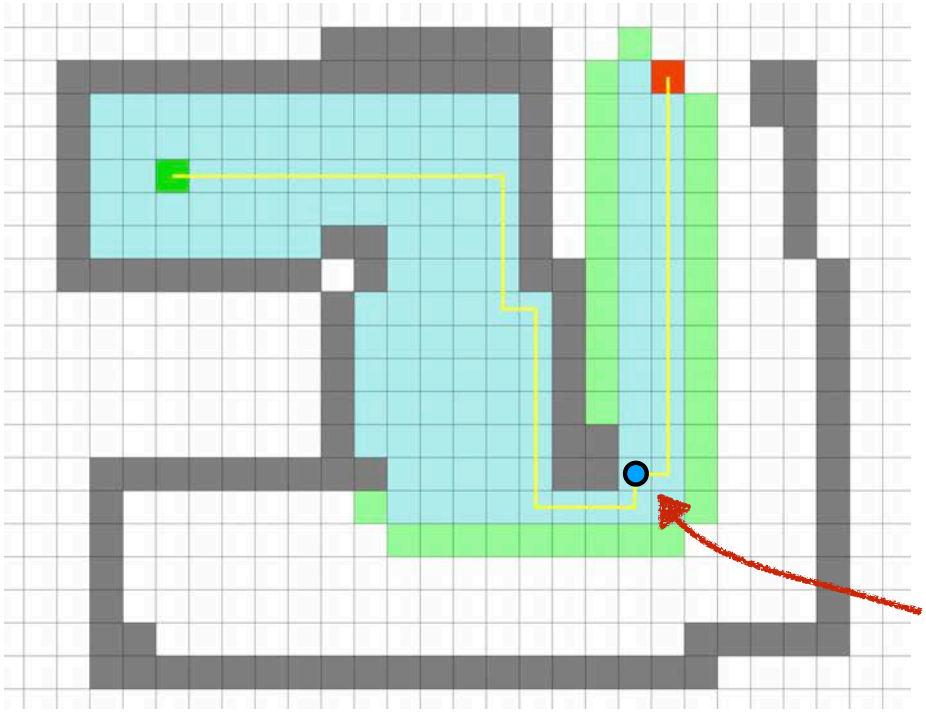
Each node's priority is distance along path route to start +

best possible distance to goal



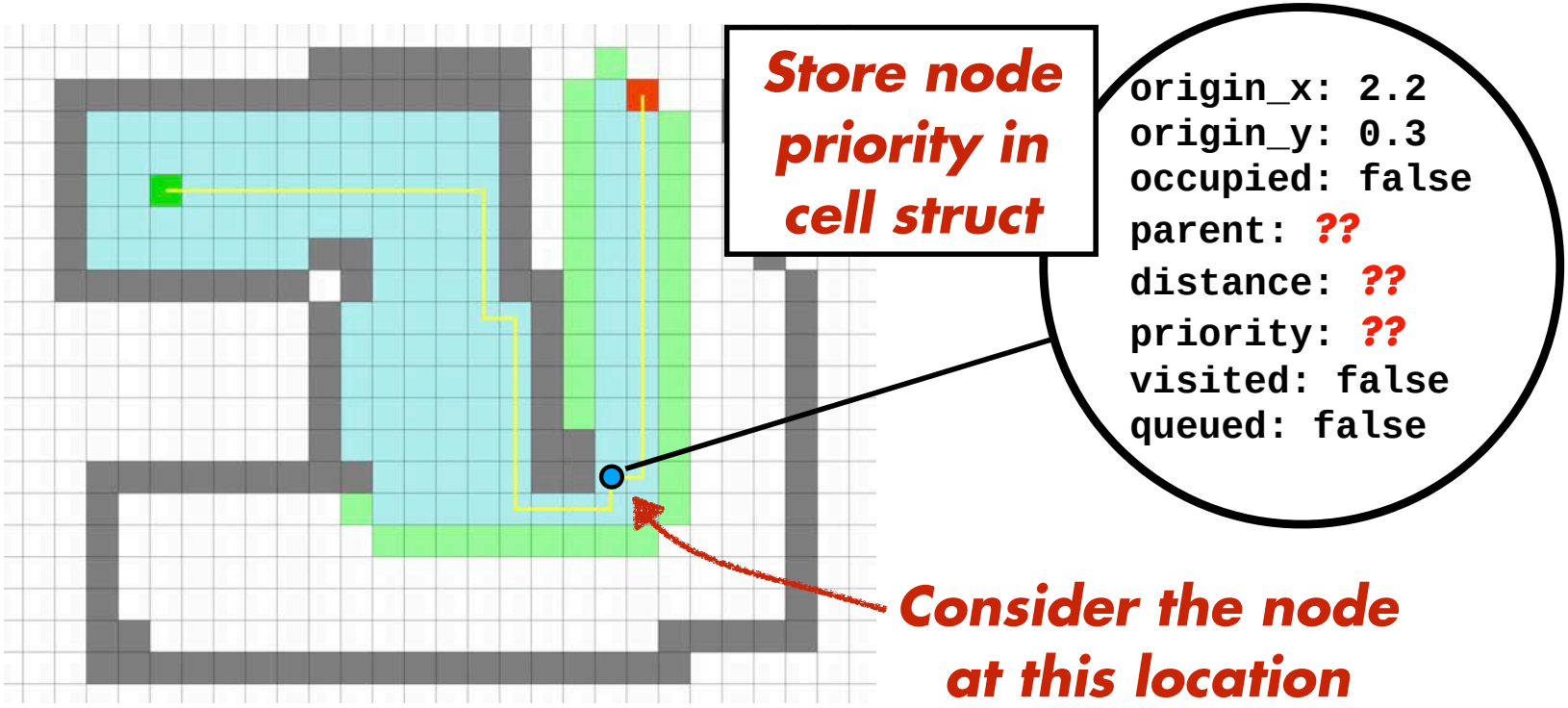
Each node's priority is distance along path route to start +

best possible distance to goal

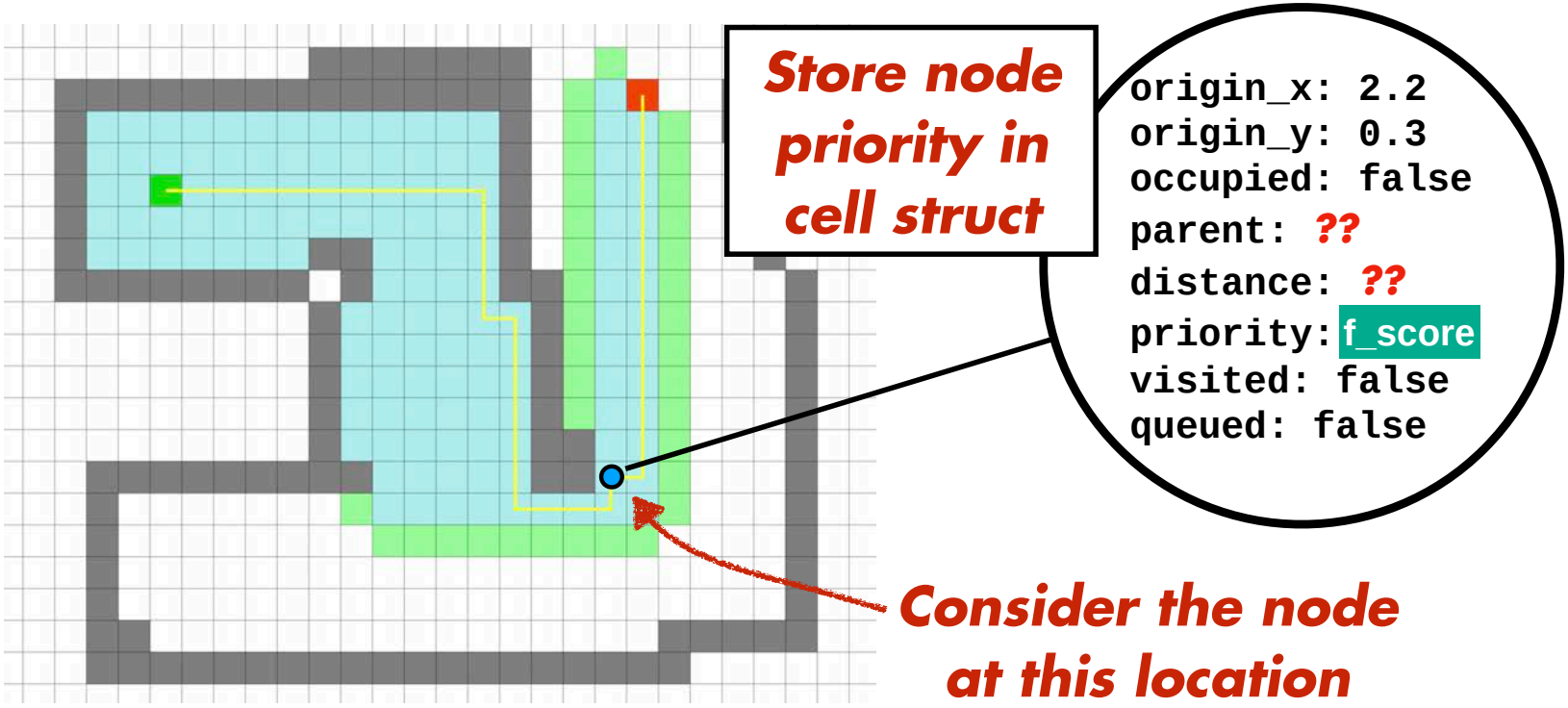


Consider the node at this location

Each node's priority is distance along path route to start + best possible distance to goal

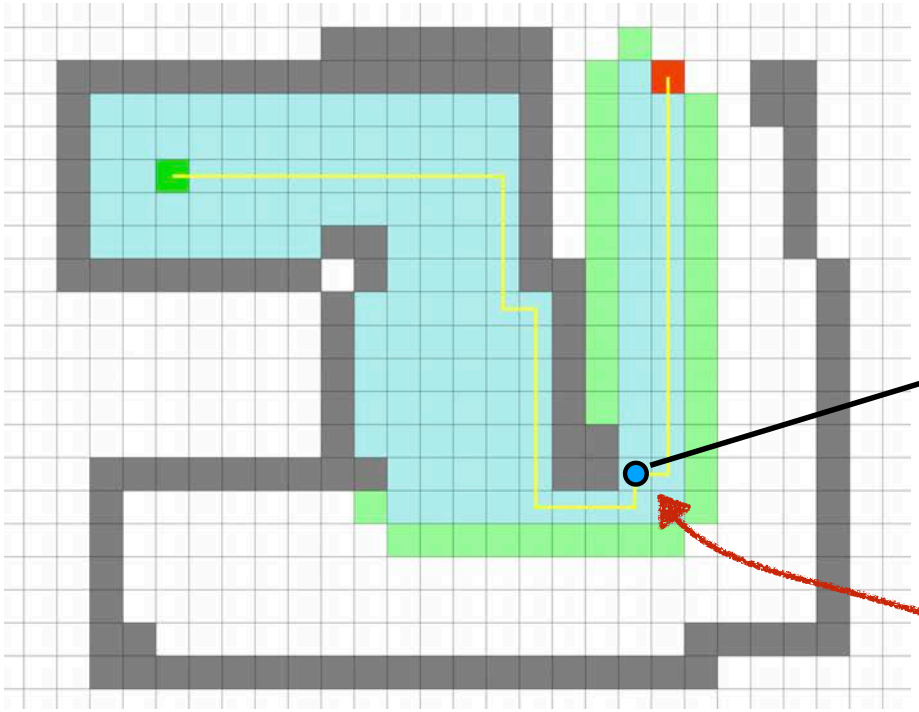


Each node's priority is distance along path route to start + best possible distance to goal



Each node's priority is distance along path route to start +

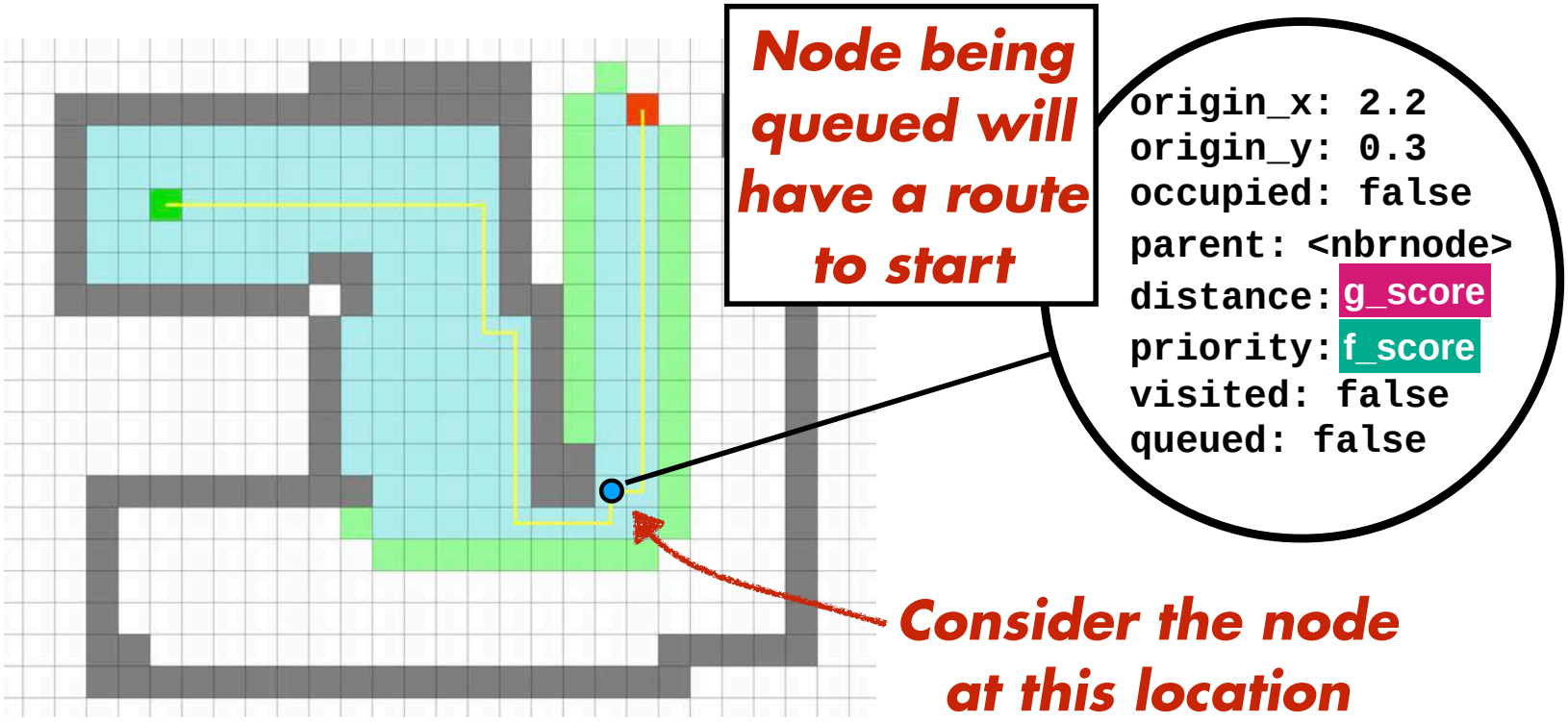
f_score **best possible distance to goal**



```
origin_x: 2.2
origin_y: 0.3
occupied: false
parent: ??
distance: ??
priority: f_score
visited: false
queued: false
```

Consider the node at this location

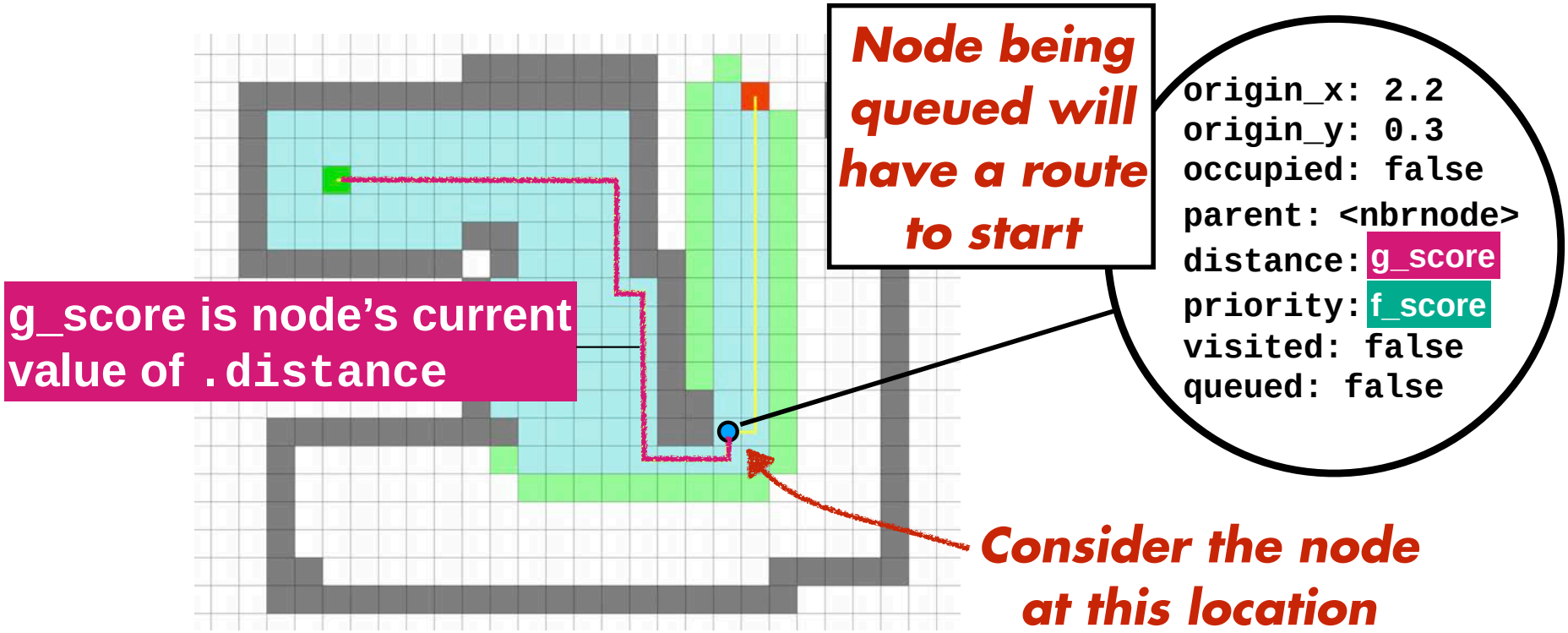
Each node's priority is distance along path route to start + f_score best possible distance to goal



Each node's priority is distance along path route to start + best possible distance to goal

g_score

f_score



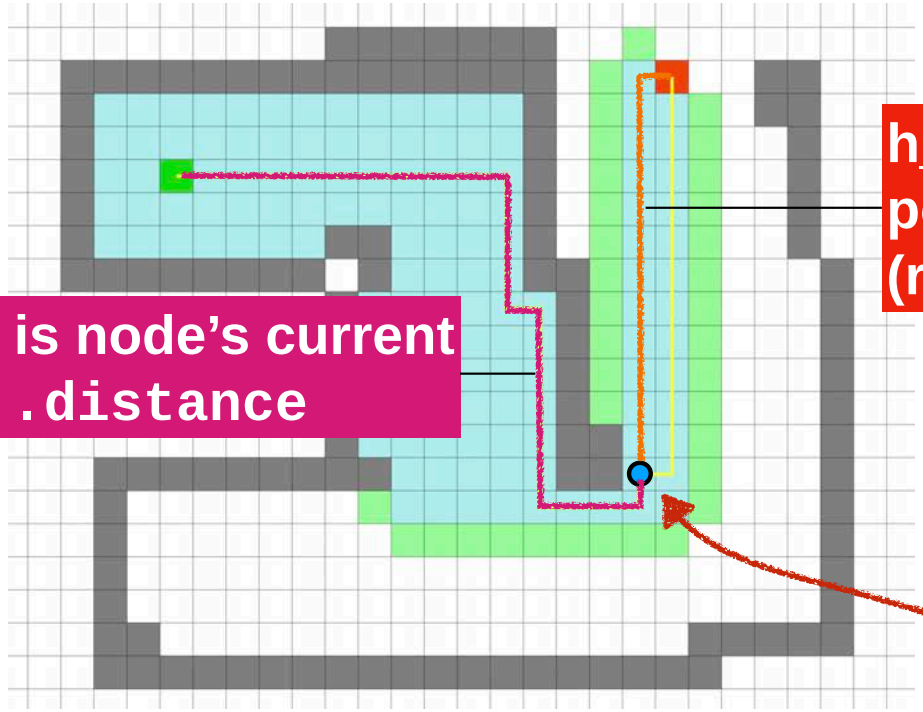
g_score

Each node's priority is distance along path route to start +

f_score

best possible distance to goal

h_score



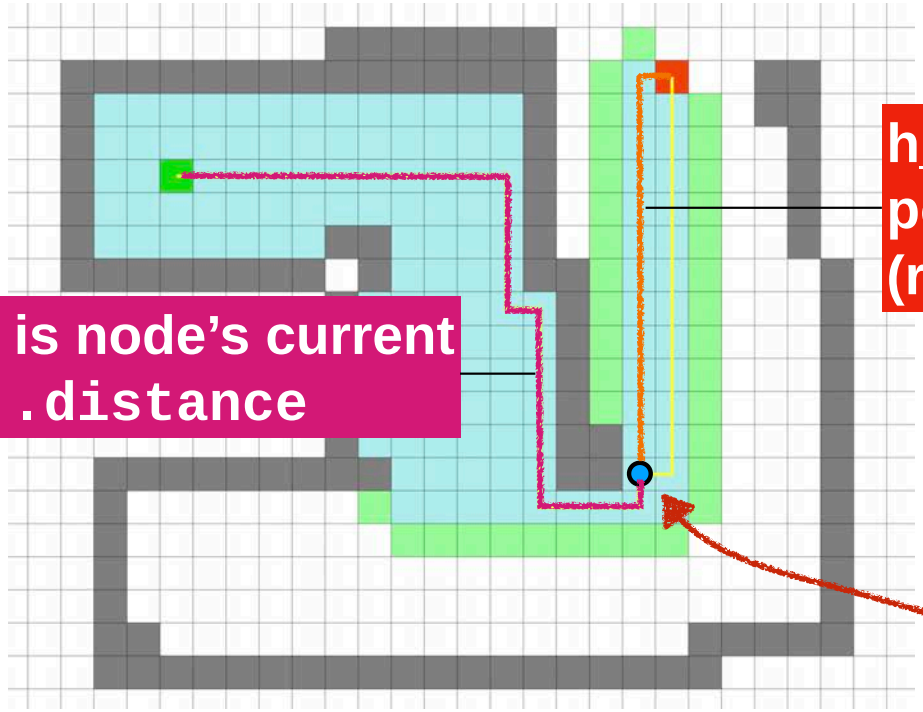
h_score is lowest possible distance to goal (regardless of collisions)

g_score is node's current value of `.distance`

Consider the node at this location

`.priority` `.distance`

f_score = g_score + h_score



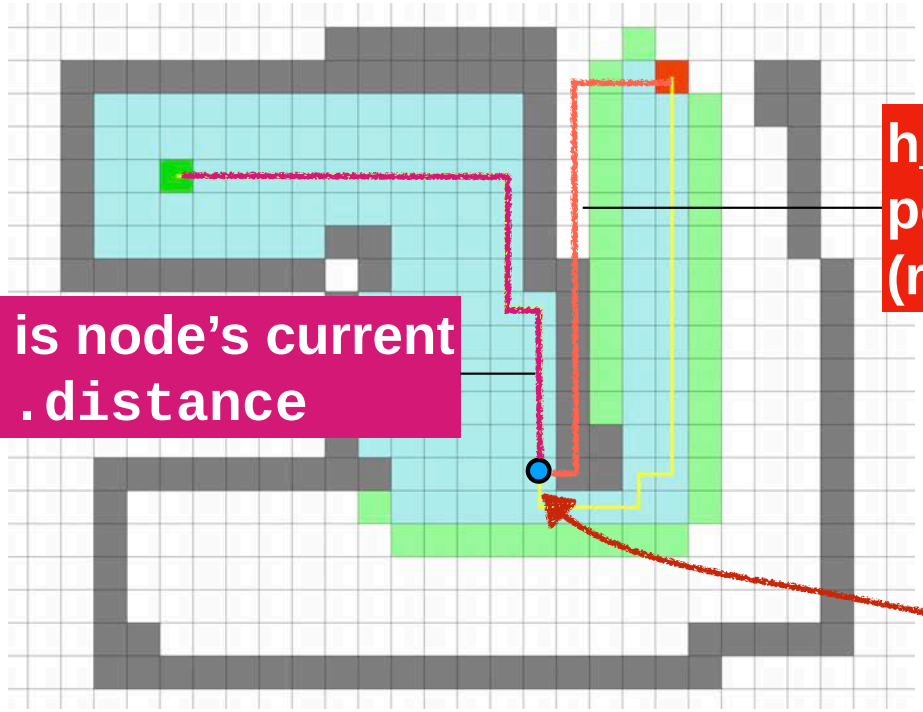
h_score is lowest possible distance to goal (regardless of collisions)

g_score is node's current value of `.distance`

Consider the node at this location

`.priority` `.distance`

f_score = g_score + h_score



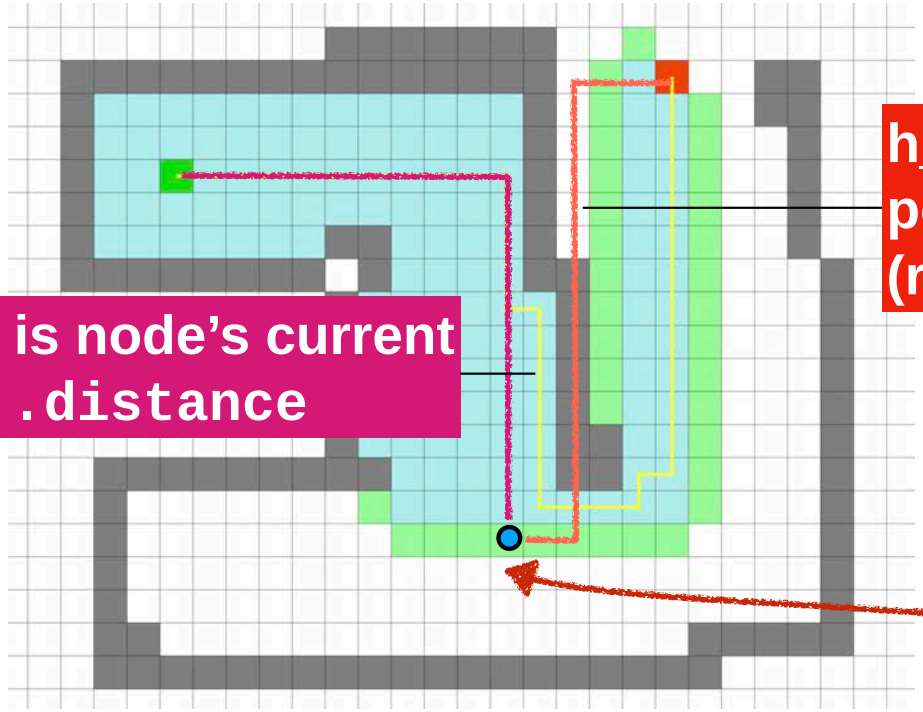
h_score is lowest possible distance to goal (regardless of collisions)

g_score is node's current value of `.distance`

Consider the node at this location

`.priority` `.distance`

f_score = g_score + h_score



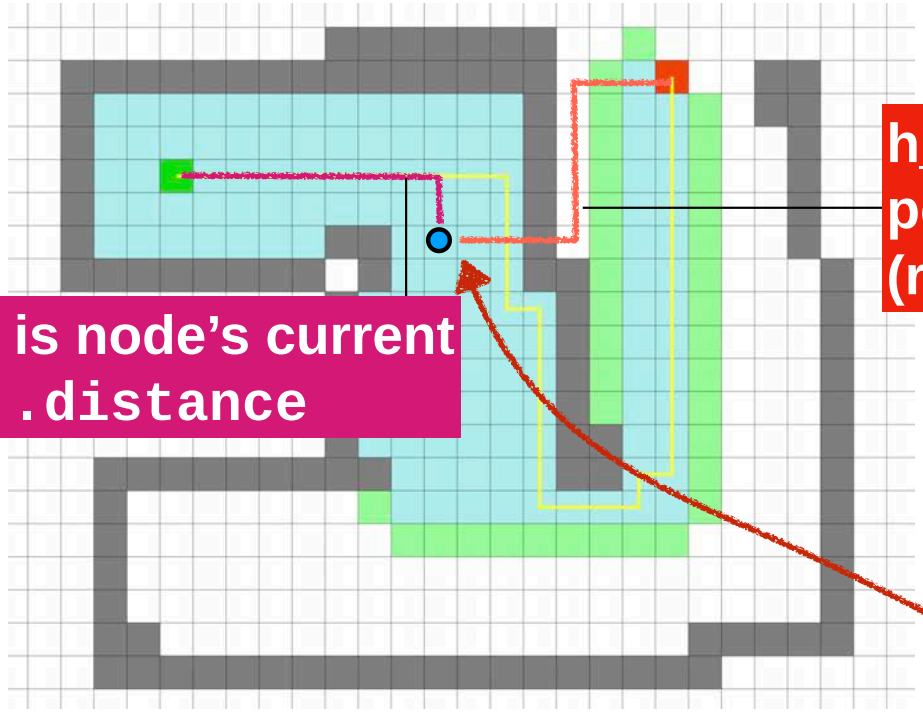
h_score is lowest possible distance to goal (regardless of collisions)

g_score is node's current value of `.distance`

Consider the node at this location

`.priority` `.distance`

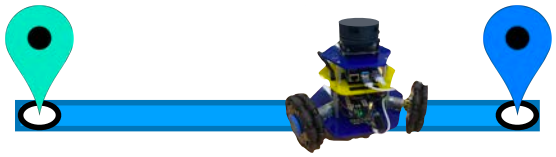
f_score = g_score + h_score



h_score is lowest possible distance to goal (regardless of collisions)

g_score is node's current value of `.distance`

Consider the node at this location



What options do we have for navigating our robot?

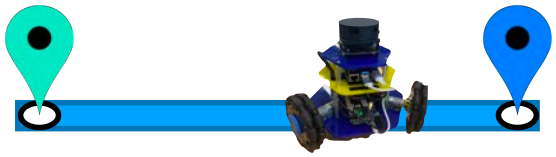
Just move randomly

Follow wall to goal

Build a map to guide us

Robot that plans paths using global search ?

Search over all possible paths

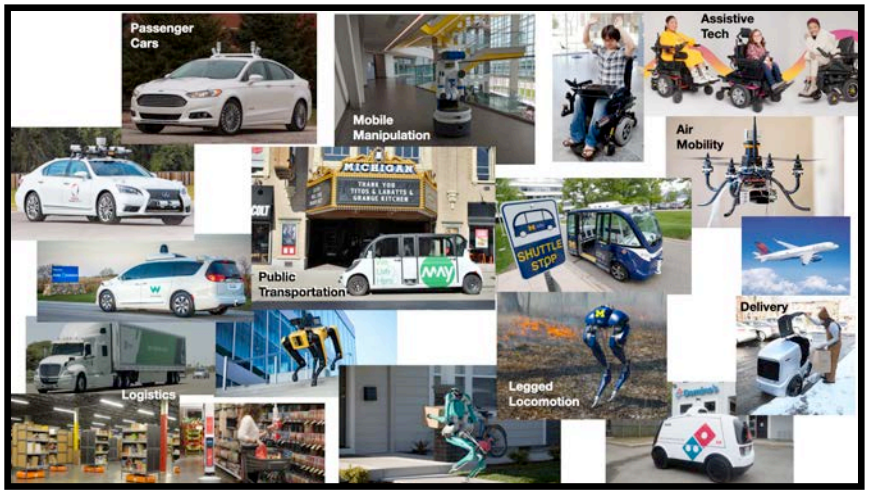


What options do we have for navigating our robot?

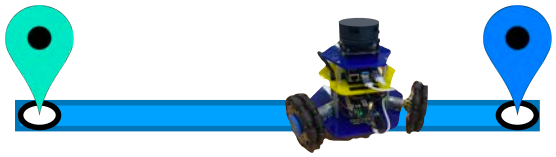
Just move randomly

Follow wall to goal

Build a map to guide us



Search over all possible paths



What options do we have for navigating our robot?

Just move randomly

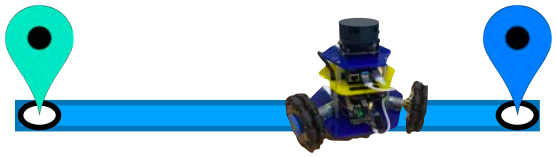
Follow wall to goal

Build a map to guide us

+ Complete algorithm
  (guarantees correct answer)
+ Optimal path

– Expensive
– Responsiveness

Search over all possible paths



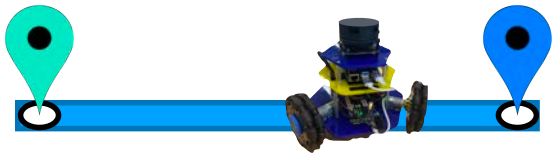
What options do we have for navigating our robot?

Just move randomly

Follow wall to goal

Build a map to guide us

Search over all possible paths



What options do we have for navigating our robot?

Random Walk

Bug Algorithm

Local Search

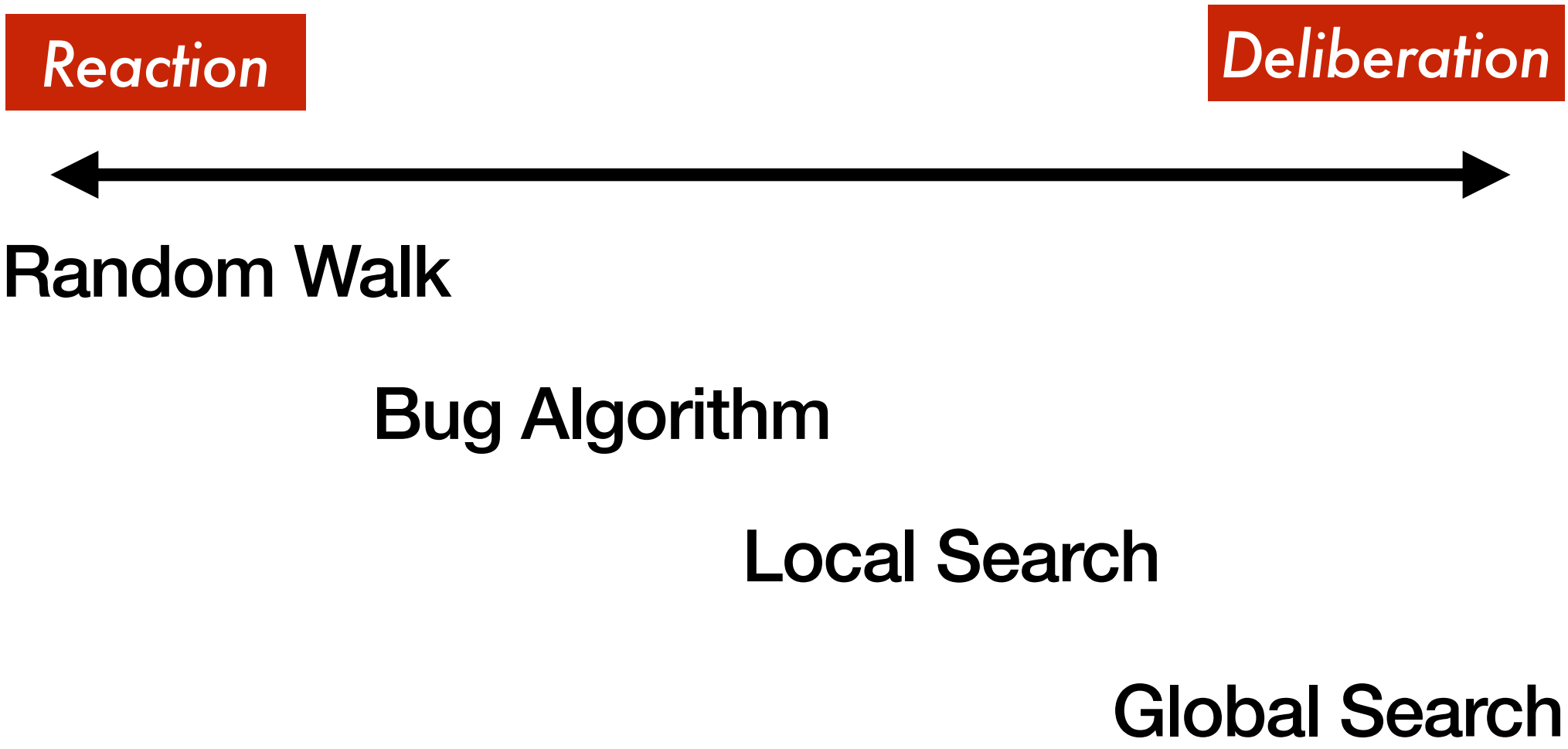
Global Search

Random Walk

Bug Algorithm

Local Search

Global Search


Reaction
Deliberation
Random Walk
Bug Algorithm
Local Search
Global Search

Reaction

Deliberation

Random Walk

+ Cheap
+ Simple
+ Robust
– Slow

Bug Algorithm

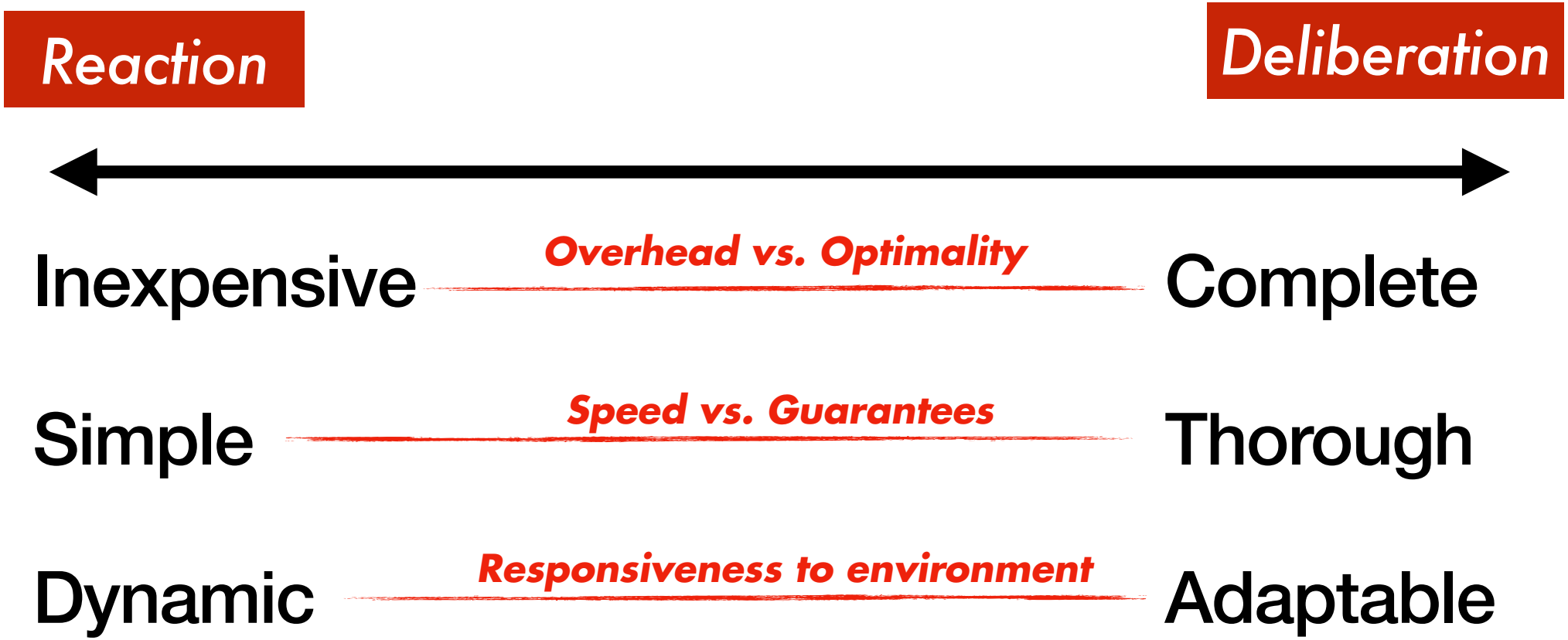
+ Simple
+ Reliable
– Known goal location
– Forgetful

Local Search

+ Adaptable
– Requires SLAM
– Gets stuck

Global Search

+ Complete
+ Optimal
– Expensive
– Responsive


Reaction
Deliberation
Inexpensive
Overhead vs. Optimality
Complete
Simple
Speed vs. Guarantees
Thorough
Dynamic
Responsiveness to environment
Adaptable

There is no one right algorithm.

There is a larger world of possibilities.

Inexpensive — ***Overhead vs. Optimality*** — Complete

Simple — ***Speed vs. Guarantees*** — Thorough

Dynamic — ***Responsiveness to environment*** — Adaptable

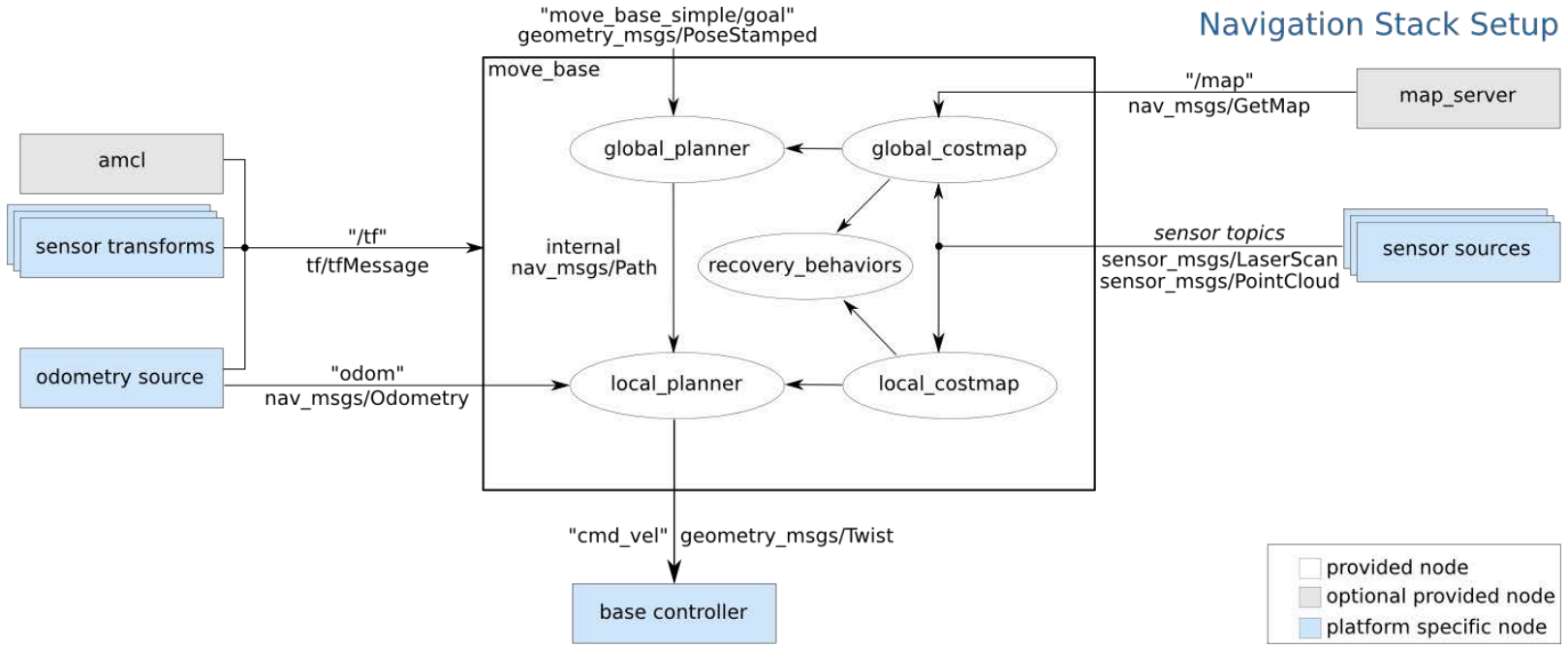
Reaction ***Deliberation***

http://wiki.ros.org/navigation/Tutorials/RobotSetup

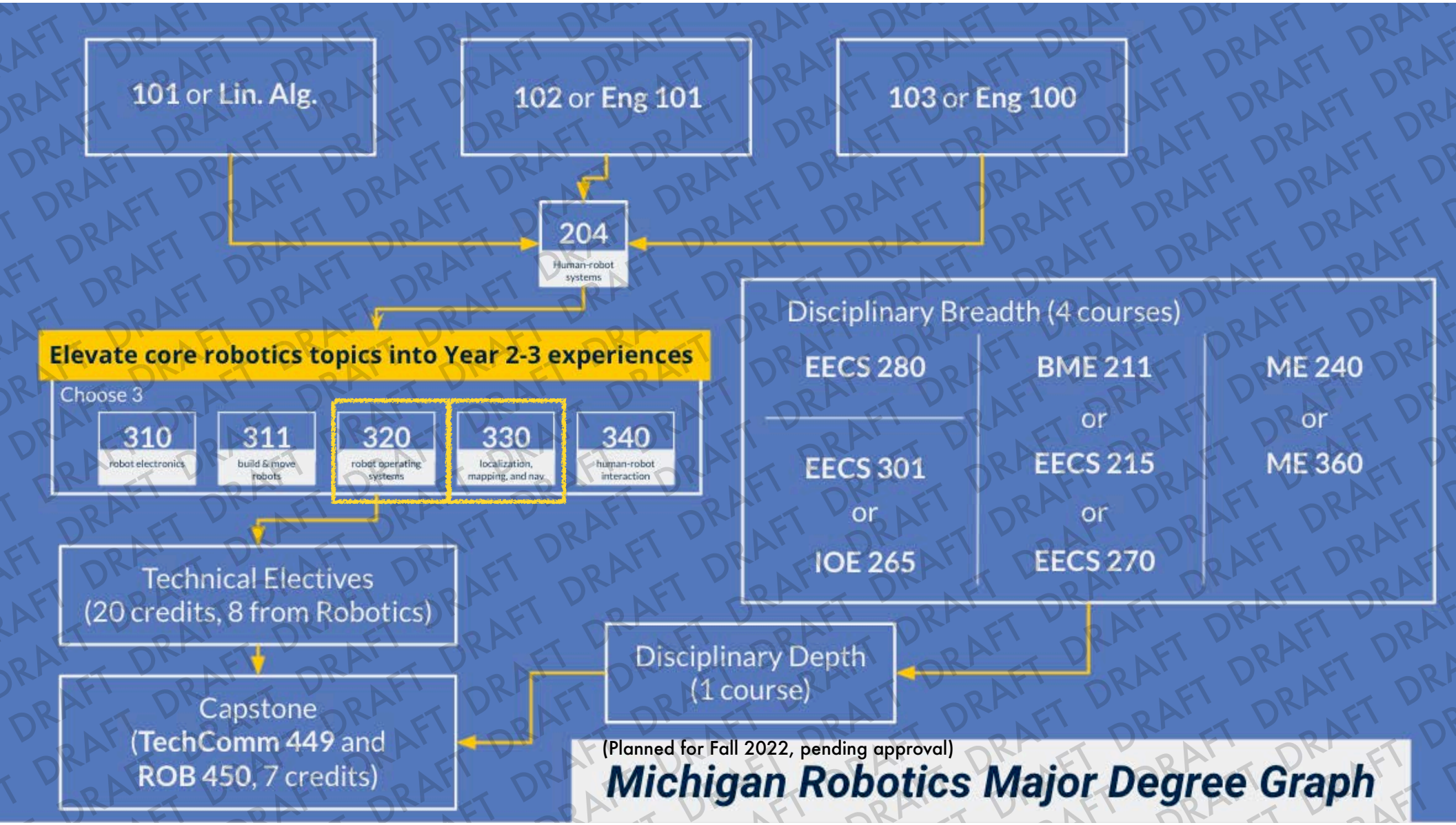
ROS.org About | Support | Discussion Forum | Service Status | Q&A answers.ros.org Search: Submit

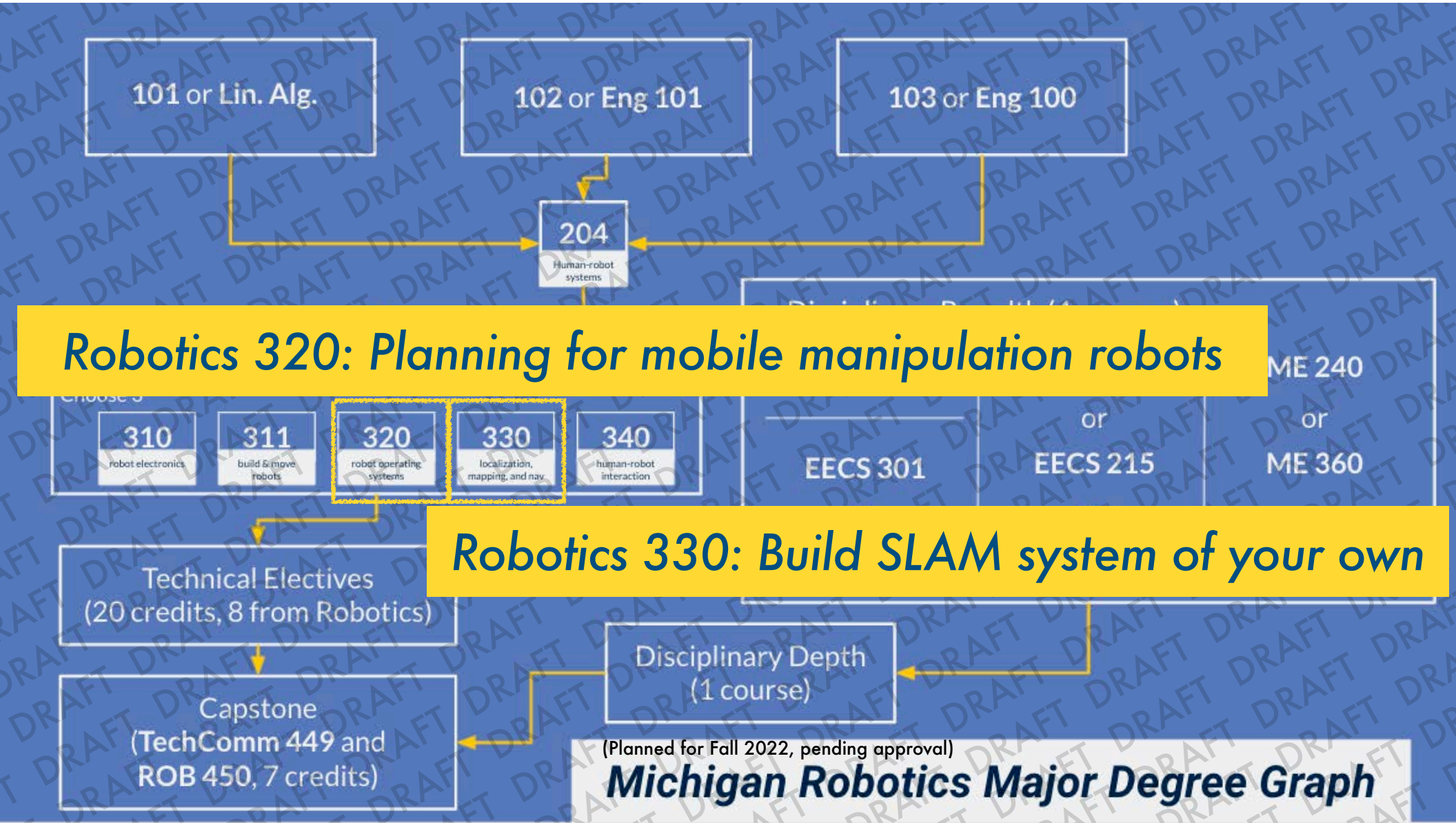
Documentation Browse Software News Download

navigation/ Tutorials/ RobotSetup

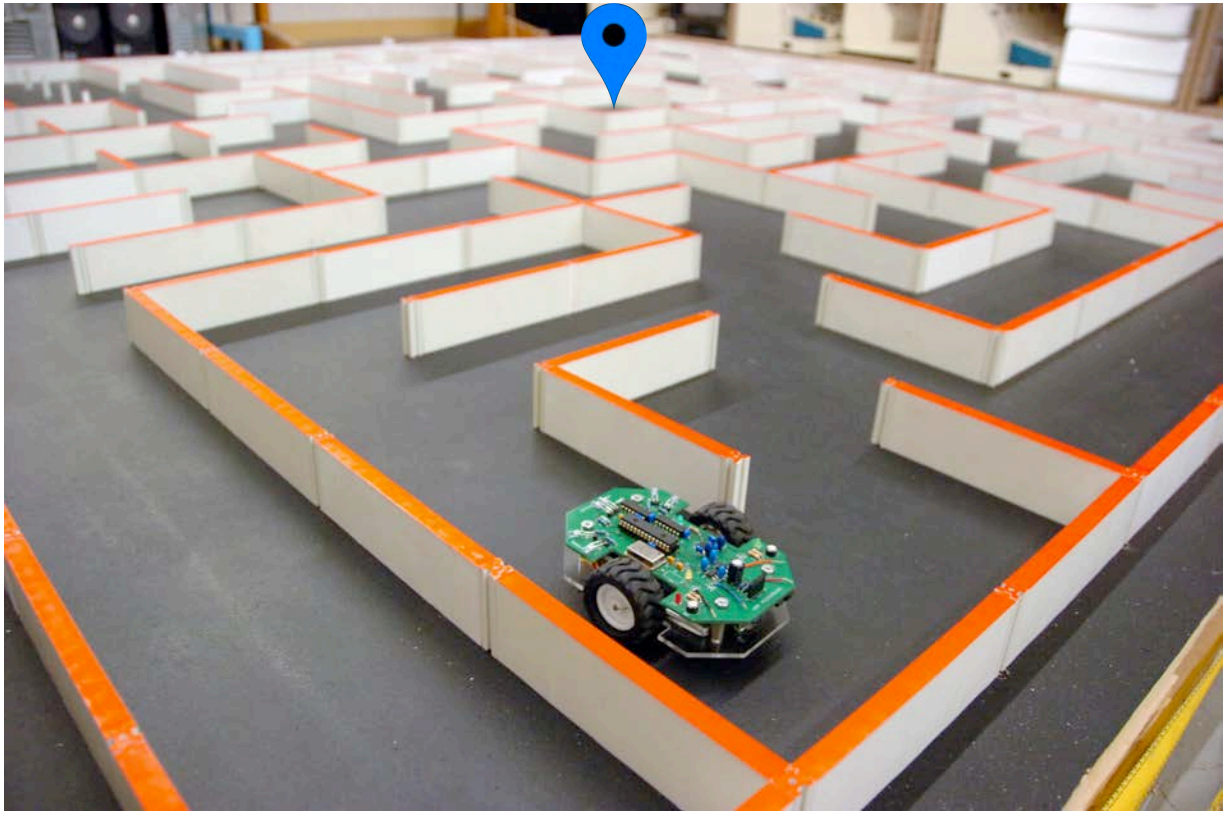


Navigation in Robot Operating Systems


101 or Lin. Alg.
102 or Eng 101
103 or Eng 100
204
Human-robot systems
Elevate core robotics topics into Year 2-3 experiences
Choose 3
310
robot electronics
311
build & move robots
320
robot operating systems
330
localization, mapping, and nav.
340
human-robot interaction
Technical Electives
(20 credits, 8 from Robotics)
Capstone
(TechComm 449 and ROB 450, 7 credits)
Disciplinary Breadth (4 courses)
EECS 280
or
EECS 301
or
IOE 265
BME 211
or
EECS 215
or
EECS 270
ME 240
or
ME 360
Disciplinary Depth
(1 course)
(Planned for Fall 2022, pending approval)
Michigan Robotics Major Degree Graph


101 or Lin. Alg.
102 or Eng 101
103 or Eng 100
204
Human-robot systems
Robotics 320: Planning for mobile manipulation robots
310
robot electronics
311
build & move robots
320
robot operating systems
330
localization, mapping, and nav.
340
human-robot interaction
EECS 301
or
EECS 215
or
ME 360
ME 240
Robotics 330: Build SLAM system of your own
Technical Electives
(20 credits, 8 from Robotics)
Disciplinary Depth
(1 course)
Capstone
(TechComm 449 and
ROB 450, 7 credits)
(Planned for Fall 2022, pending approval)
Michigan Robotics Major Degree Graph

Autonomous Navigation: Global Search



https://app.emaze.com/@AIRRTROT/idea-2-the-robot-maze#1

Robotics 102

Introduction to AI and Programming
University of Michigan and Berea College
Fall 2021