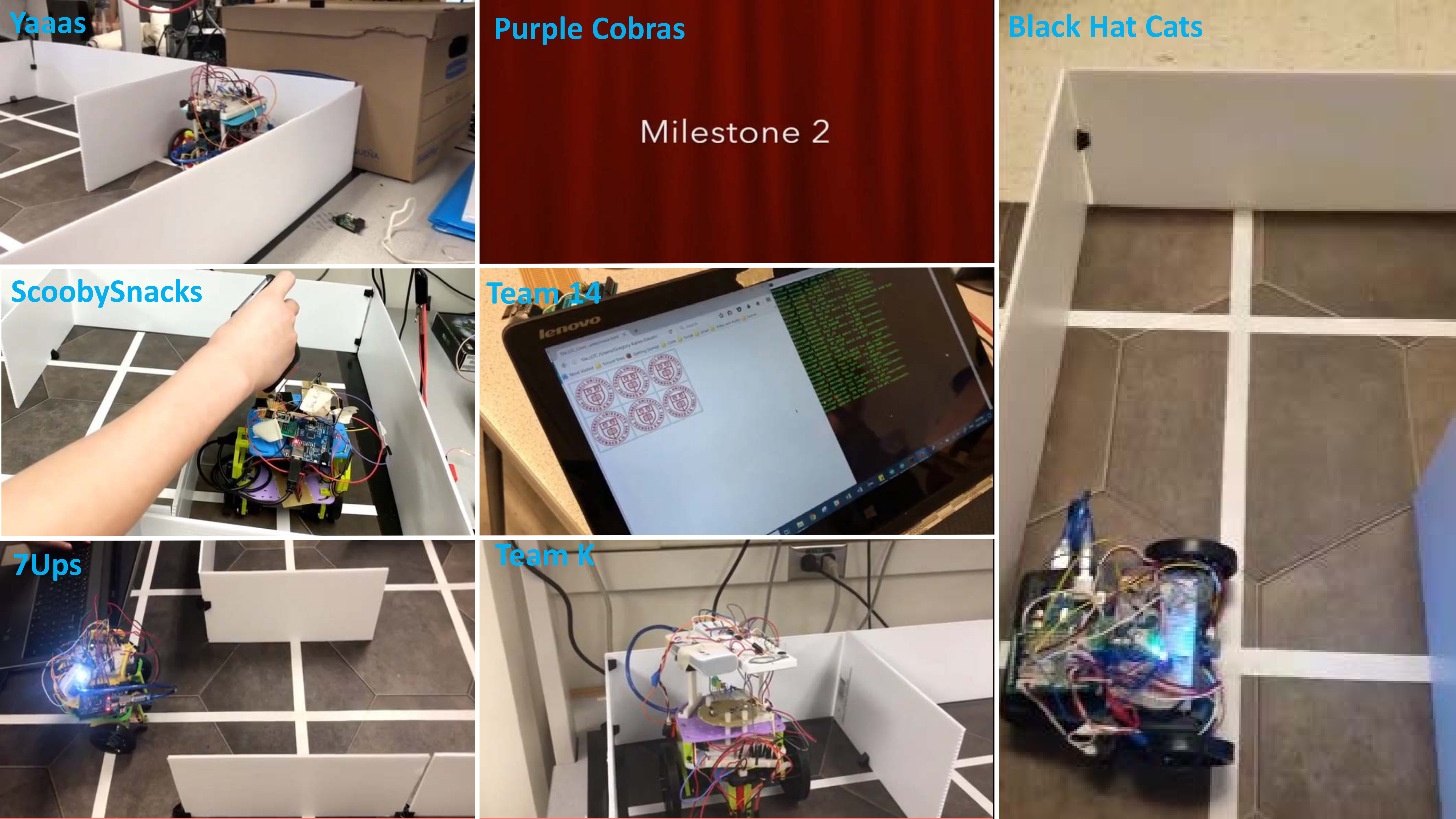
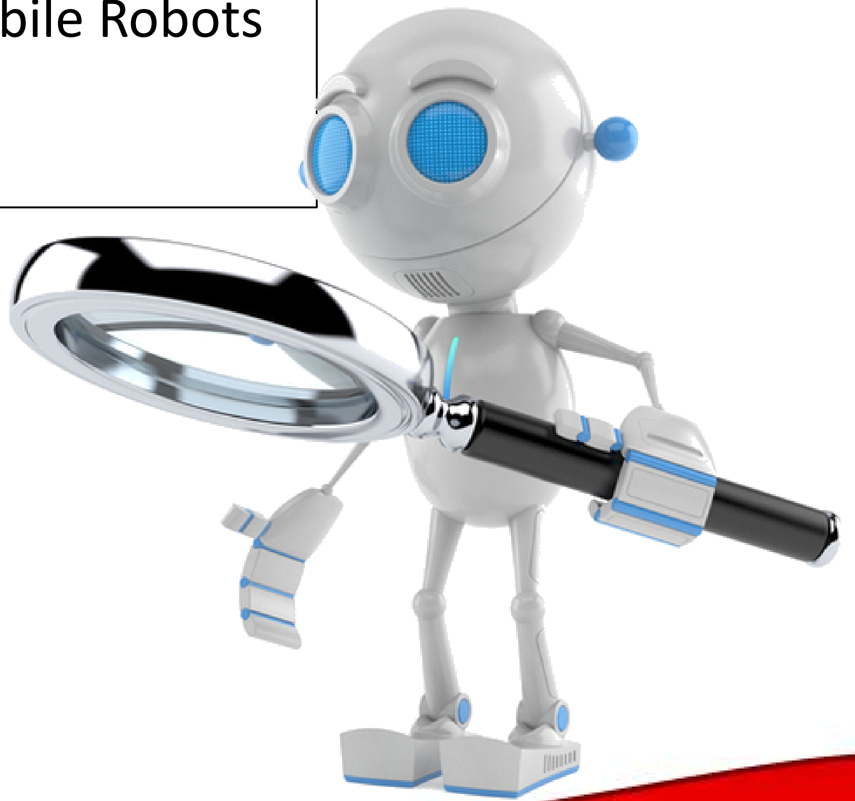

Yaaas
Purple Cobras
Milestone 2
Black Hat Cats
ScoobySnacks
Team 14
lenovo
7Ups
Team K

Algorithms and Path Planning

Topics
- Simple Search
- Depth First Search
- Breadth First Search
- Dijkstra’s Search
- Greedy Best First Search
- A* Search
- Adversarial Search

Classes of interest
- ECE2400: Computer Systems Programming
- CS4700: Foundations of Artificial Intelligence
- CS4701: Practicum of Artificial Intelligence
- CS3758/MAE4180: Autonomous Mobile Robots
- ECON4020: Game Theory
- ORIE 4350: Game Theory



ECE 3400:
Intelligent Physical Systems

Coverage

The full mazes will be 9 x 9 squares. The robot that maps the most of the maze correctly (wrt to walls and gaps) in a given round will receive 15 points. All other robots will receive scaled values thereof.

Treasures

- For every treasure which is located correctly: 1 point
- For every treasure that is located and color-identified correctly: 1 point
- For every treasure that is located and shape-identified correctly: 1 point
- For every discovery of a treasure that is not there: -1 point
- The minimum score per round is 0 points; the maximum is 20 points.

Can you explore the entire maze?

- 15s avg. for 6 squares
- 3.4min for 81 squares
- Unlikely, but possible.

Is there a reason to stop exploring?

Penalties

If your robot crashes into another robot you will receive -5 points per event (with the lowest score per round being 0 points).

Grading – last two milestones

Milestone 3 will be graded as follows:

- 8 points: Robot capable of maze exploration using DFS, BFS, Dijkstra, or A* (show that your robot can do different maze configurations, we expect at least one of them to be a minimum size of 4x5)
- 2 points: ..if the robot is also able to update the GUI

Milestone 4 will be graded as follows:

- 1 points: Robot which can detect when there are/are not treasures
- 4 points: Robot which can successfully distinguish between red and blue treasures
- 5 points: Robot which can successfully distinguish a square, triangle, and diamond shape

Search and Path Planning

- How do I get to my goal?
- No simple answers…
  - Can you see your goal?
  - Do you have a map?
  - Are obstacles unknown, or dynamic?
  - Does it matter how fast you get there?
  - Does it matter how smooth the path is?
  - How much computing power do you have?
  - How precise is your motion control?

Search and Path Planning

- *What is the simplest possible search?*
  - Random motion (no intelligence)
- Reactive path planning (purely local)
  - Visual homing
  - Wall following, etc…
- Bug-based Algorithms (mostly local)
  - Sense direction and distance to the goal
  - No knowledge of map and obstacles

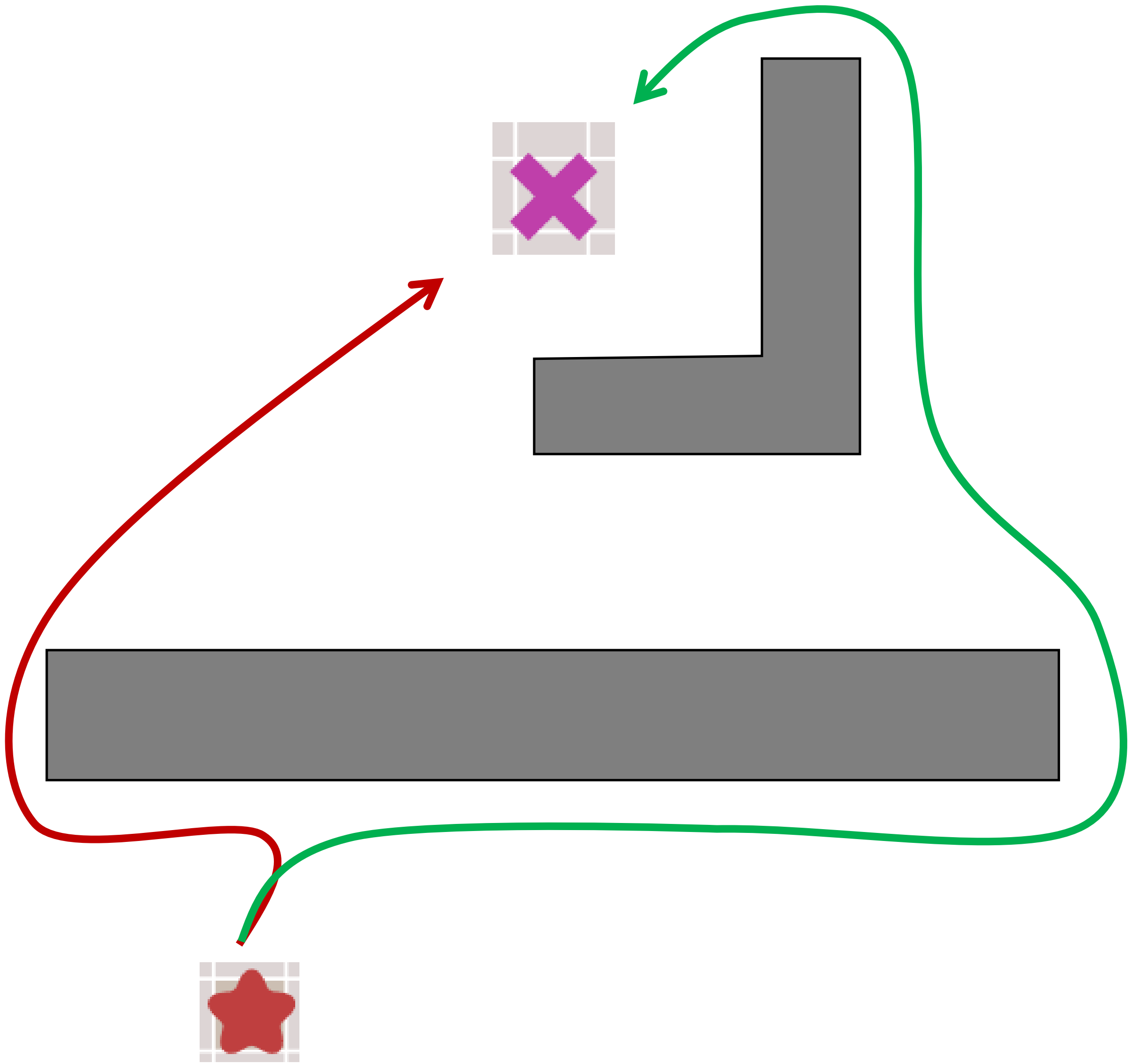
Bug 0

Sensor Assumptions:

- Direction to the goal
- Detect walls

Algorithm:

1. Go towards goal
2. Follow obstacles until you can go towards goal again
3. Loop



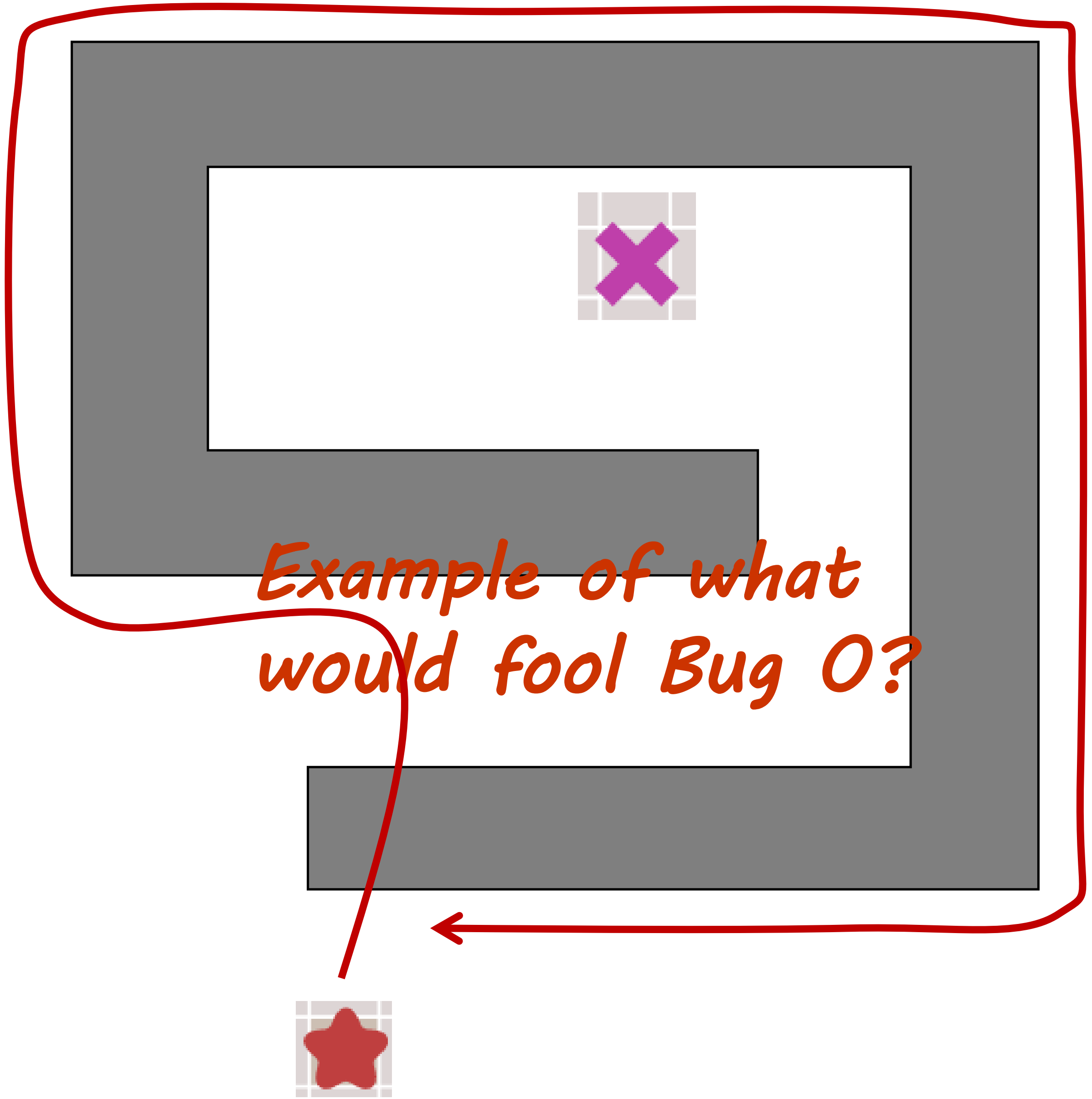
Bug 0

Sensor Assumptions:

- Direction to the goal
- Detect walls

Algorithm:

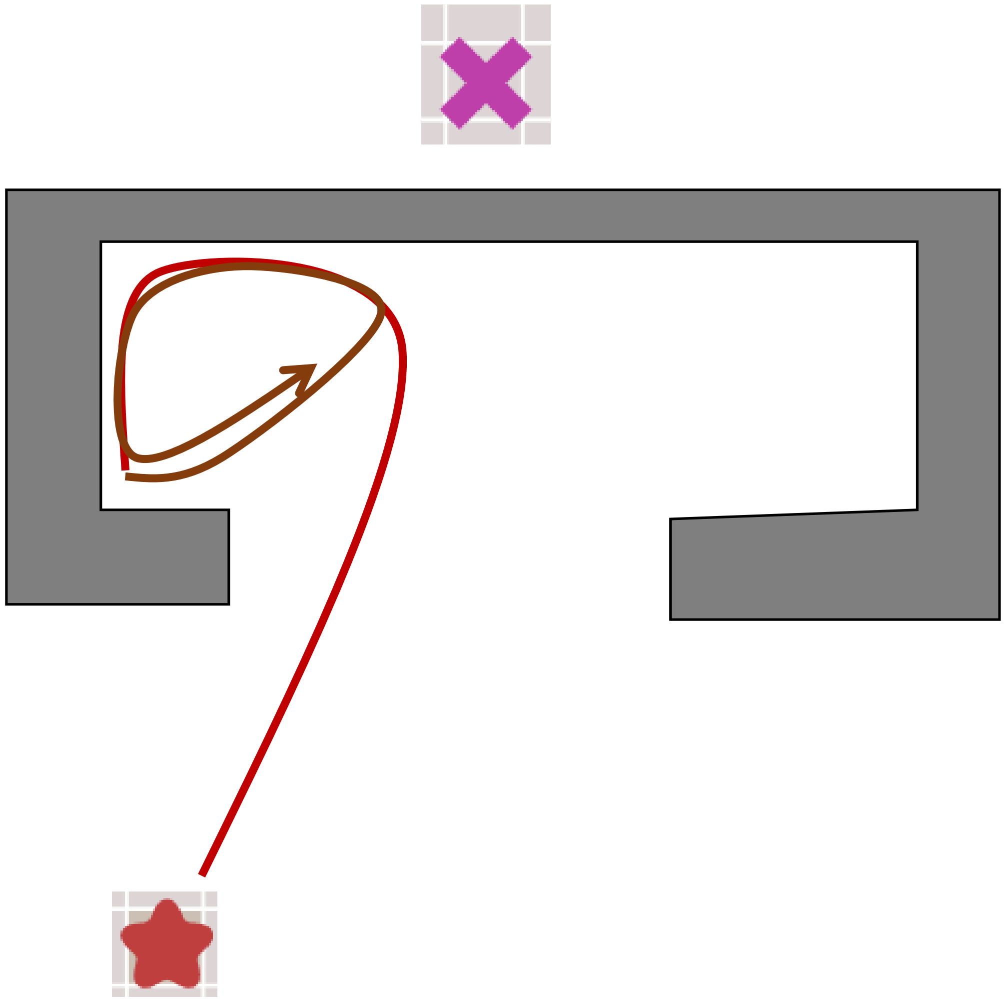
1. Go towards goal
2. Follow obstacles until you can go towards goal again
3. Loop



Bug 0

Sensor Assumptions:

- Direction to the goal
- Detect walls



Algorithm:

1. Go towards goal
2. Follow obstacles until you can go towards goal again
3. Loop

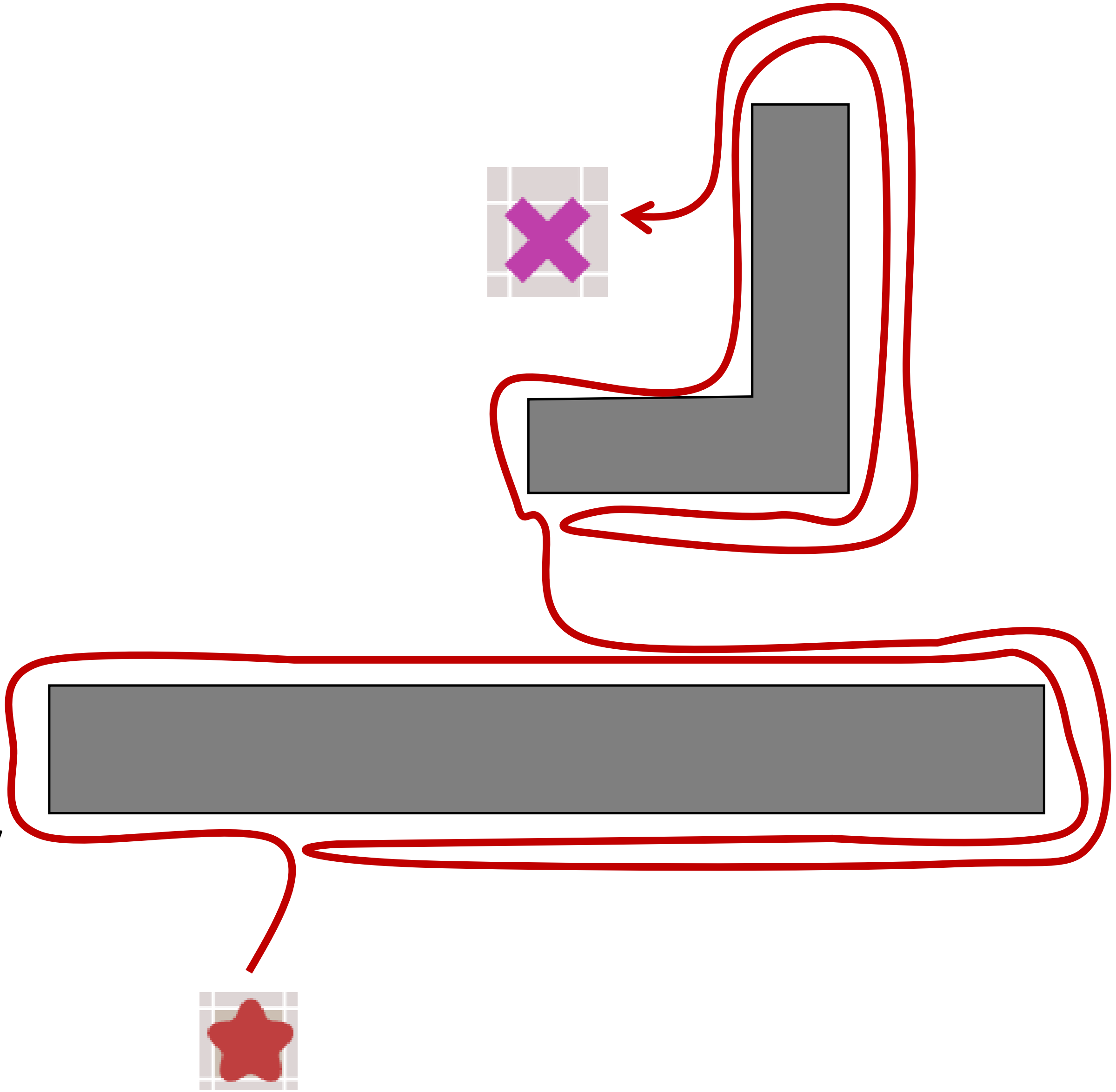
Bug 1

Sensor Assumptions:

- Direction to the goal
- Detect walls
- Odometry

Algorithm:

1. Go towards goal
2. Follow obstacles *and remember how close you got to the goal*
3. Return to the closest point, and loop



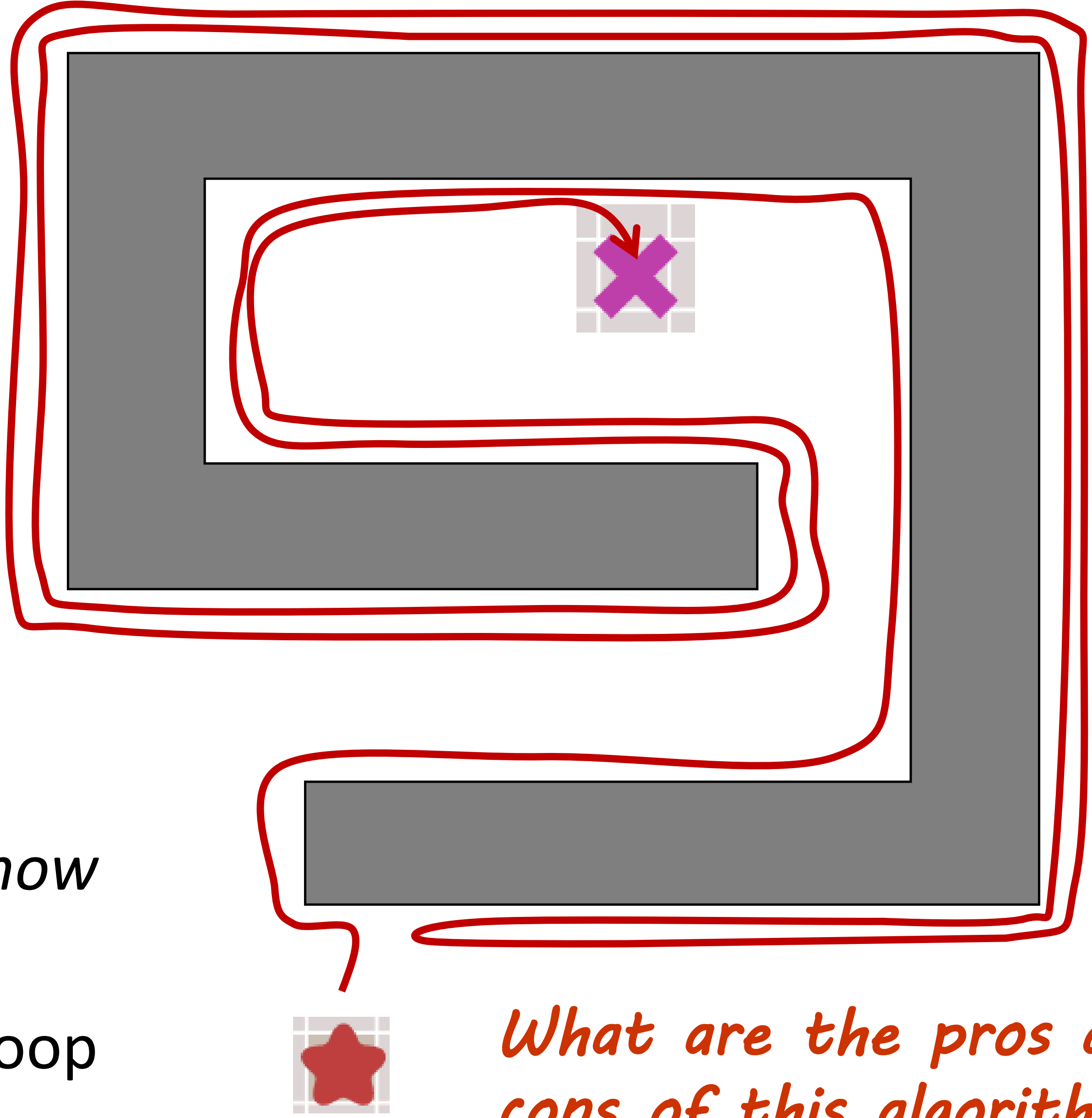
Bug 1

Sensor Assumptions:

- Direction to the goal
- Detect walls
- Odometry

Algorithm:

1. Go towards goal
2. Follow obstacles *and remember how close you got to the goal*
3. Return to the closest point, and loop



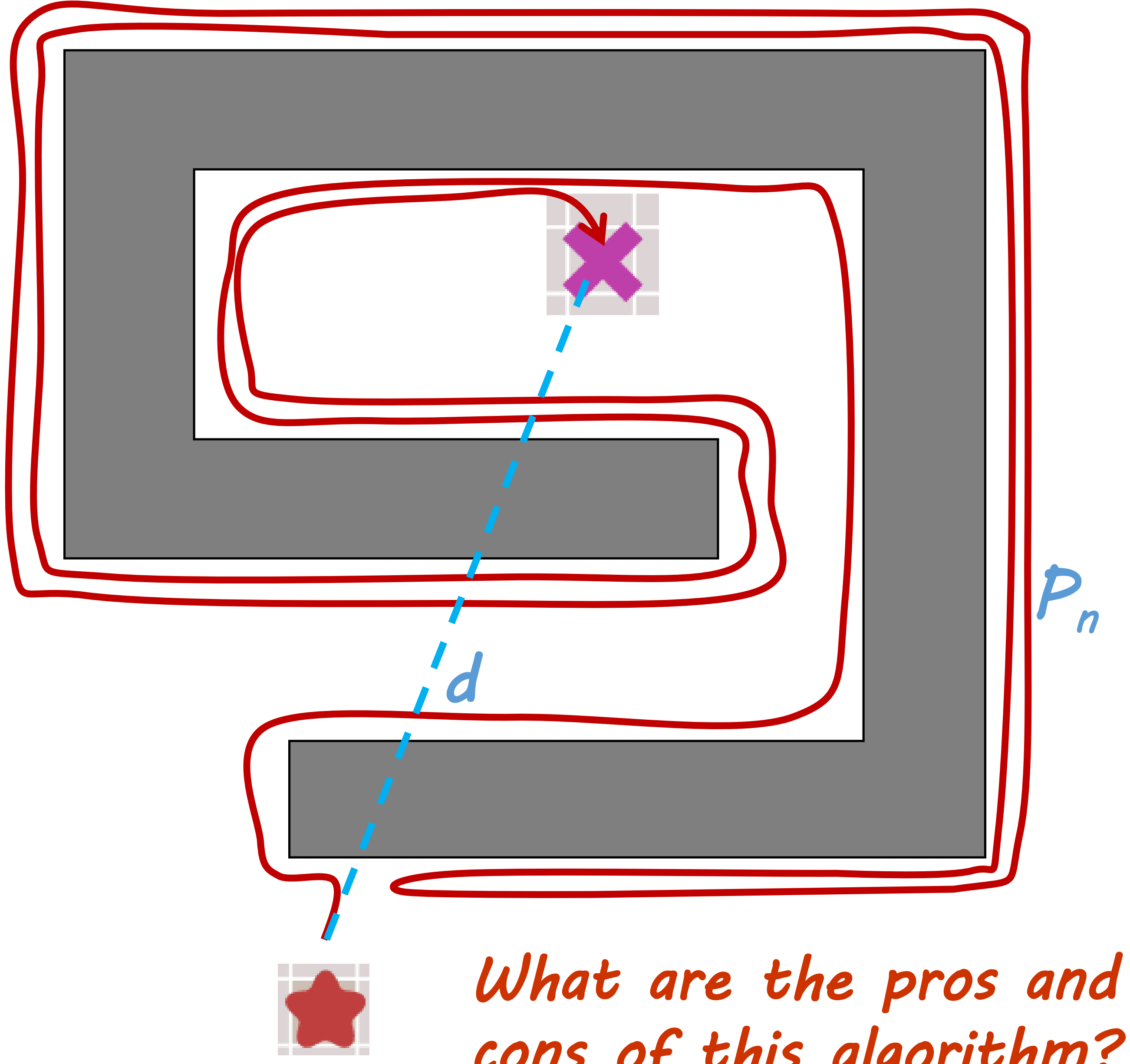
What are the pros and cons of this algorithm?

Bug 1 - formally

Sensor Assumptions:

- Direction to the goal
- Detect walls
- Odometry

- **Lower bound traversal?**
  - **d**
- **Upper bound traversal?**
  - **d + 1.5 · Sum(P)**



What are the pros and cons of this algorithm?

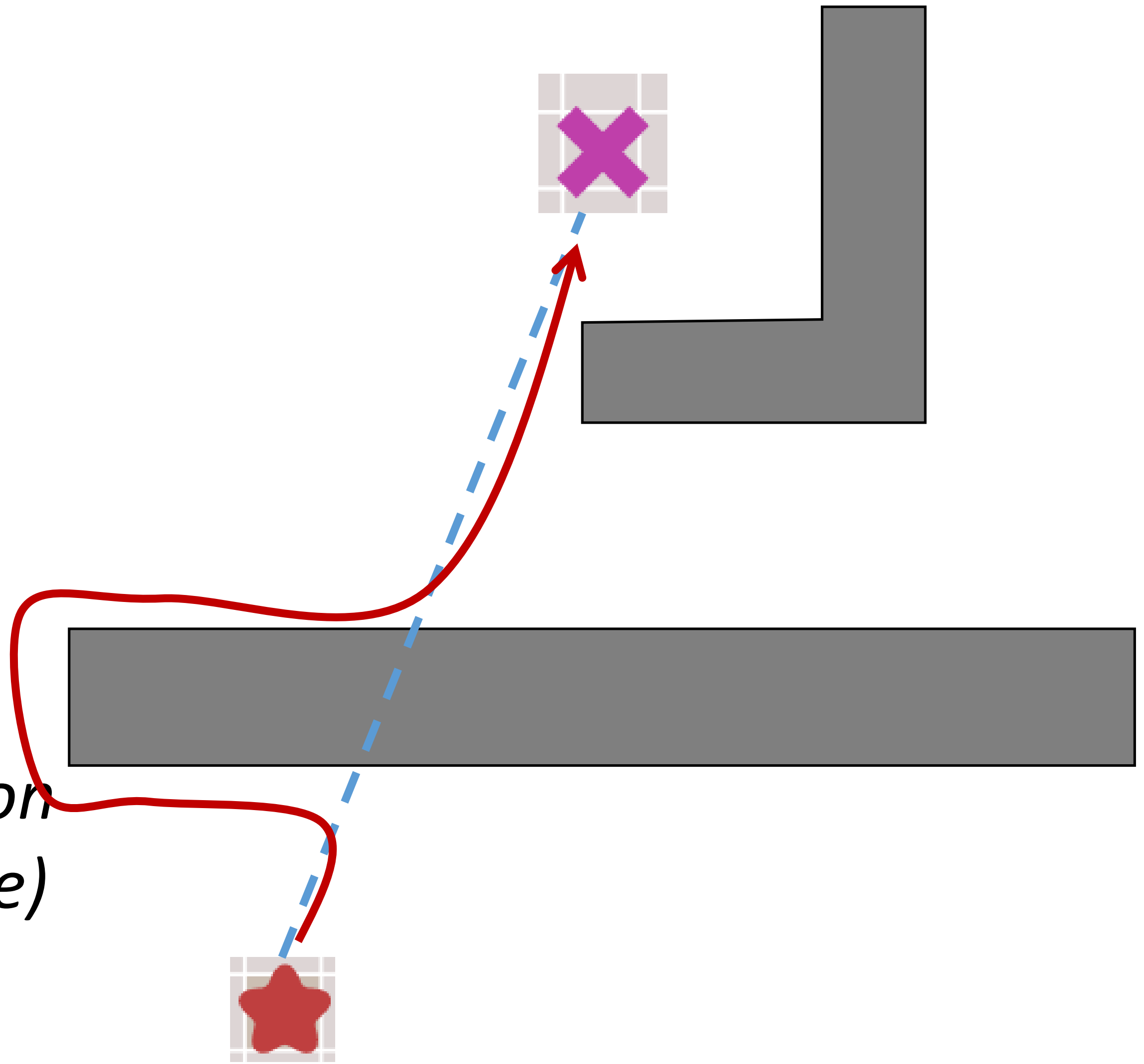
Bug 2

Sensor Assumptions:

- Direction to the goal
- Detect walls
- Odometry
- Original vector to the goal

Algorithm:

1. Go towards goal on the vector
2. Follow obstacles *until you are back on the vector (and closer to the obstacle)*
3. Loop



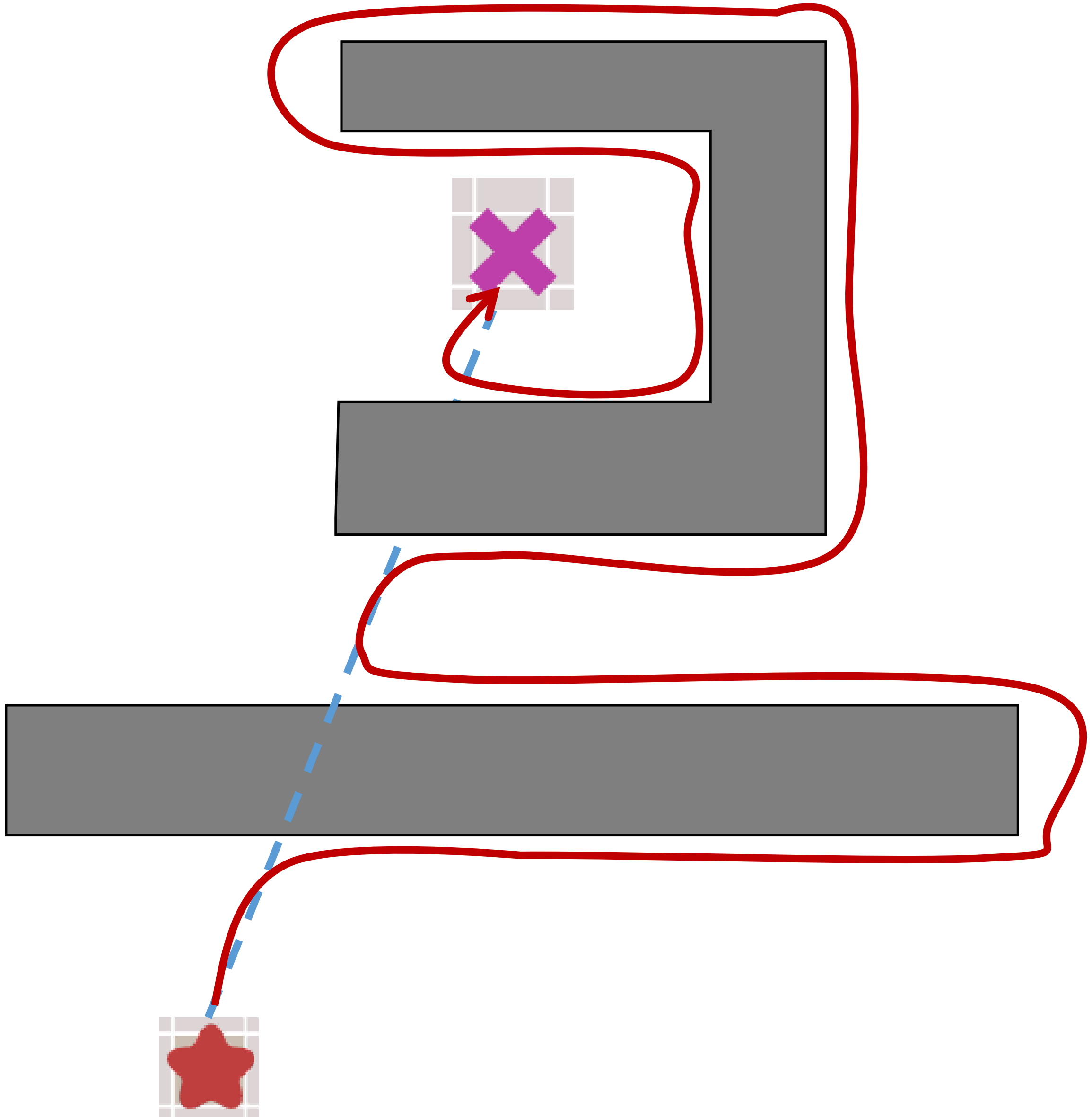
Bug 2

Sensor Assumptions:

- Direction to the goal
- Detect walls
- Odometry
- Original vector to the goal

Algorithm:

1. Go towards goal on the vector
2. Follow obstacles *until you are back on the vector*
3. Loop



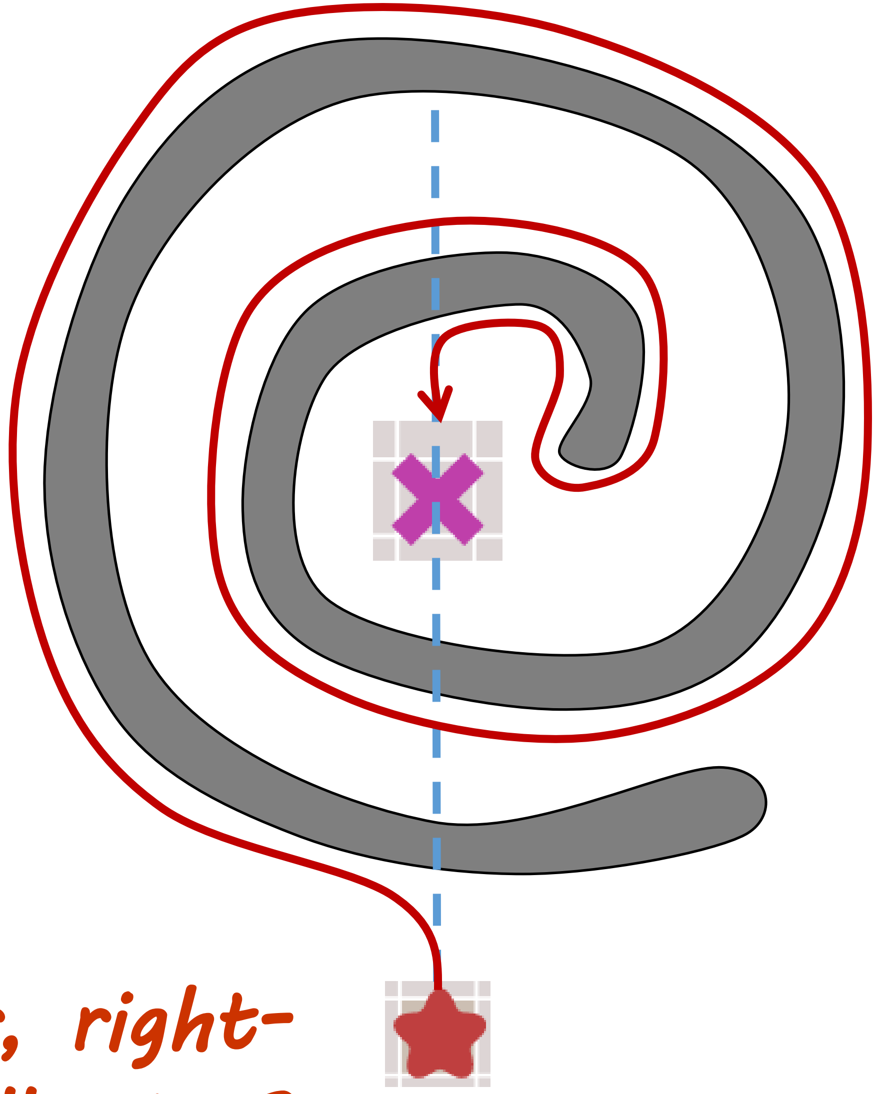
Bug 2

Sensor Assumptions:

- Direction to the goal
- Detect walls
- Odometry
- Original vector to the goal

Algorithm:

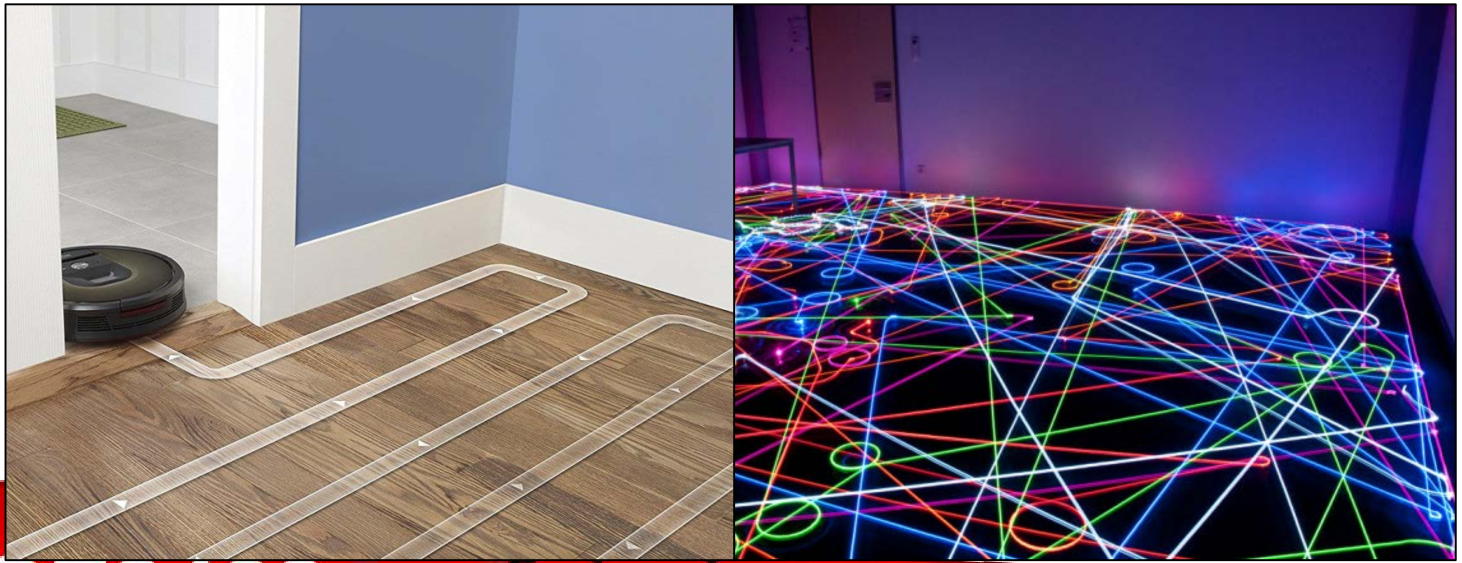
1. Go towards goal on the vector
2. Follow obstacles *until you are back on the vector (and doing something new)*
3. Loop



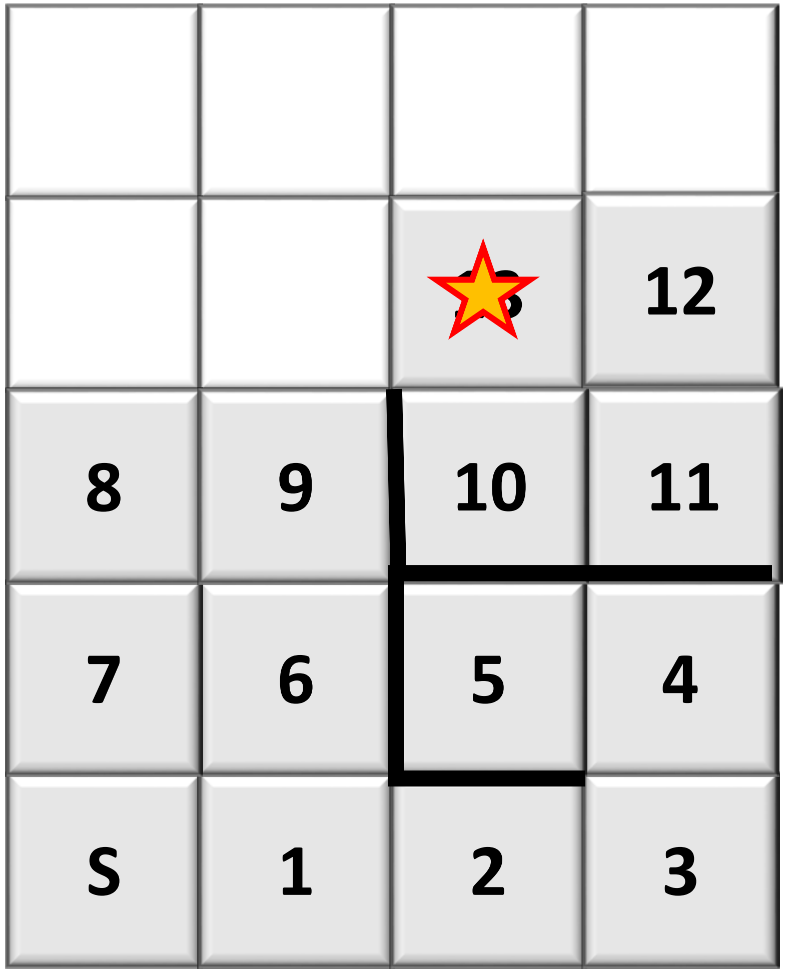
What is faster, right- or left wall following?

Algorithms and Search

- *What is the simplest thing to do?*
  - Brute force search
  - *How many grid traversals will it take?*
    - First establish a search order
    - Advance x first, then increment y and decrease x, etc.



Find a treasure



Search order: N, E, S, W

Algorithms and Search

- *What is the simplest thing to do?*
  - Brute force search
- *Other methods?*
  - Depth First Search (DFS)

Find a treasure

Algorithms and Search

Search order: N, E, S, W

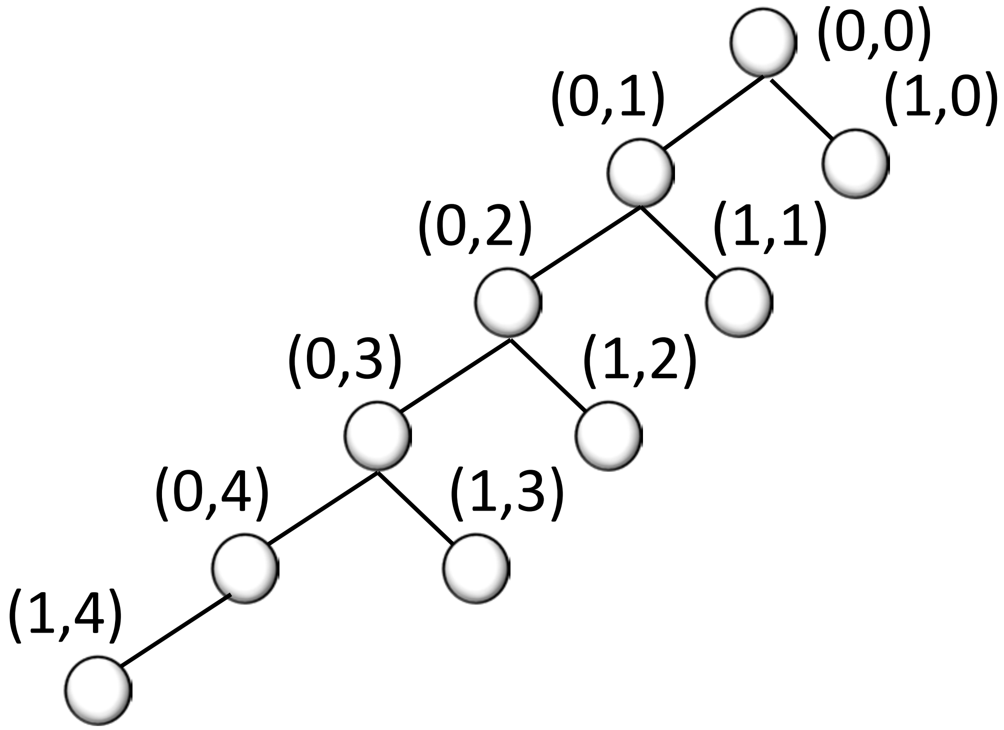
- *What is the simplest thing to do?*
  - Brute force search
- *Other methods?*
  - Depth First Search (DFS)
  - Breadth First Search (BFS)

Find a treasure

Search order: N, E, S, W

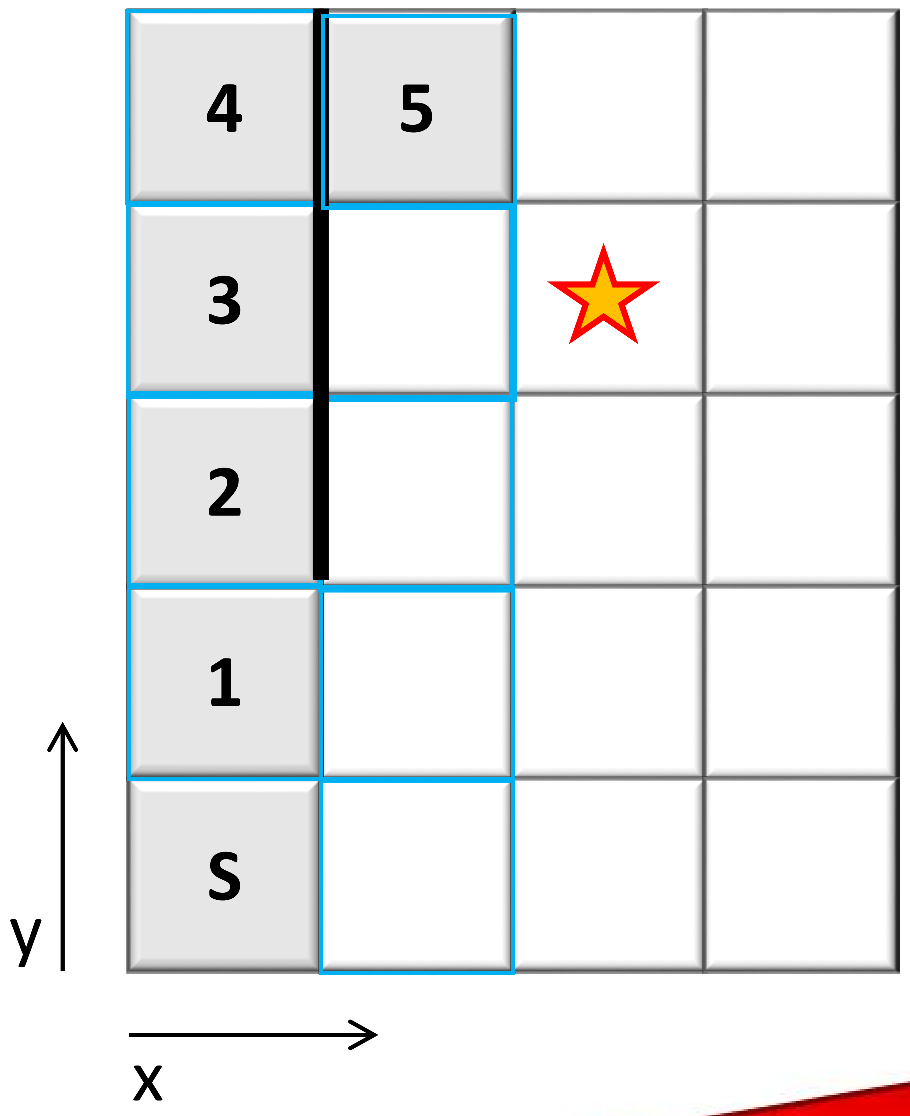
Algorithms and Search

- Depth First Search (DFS)



and so on…

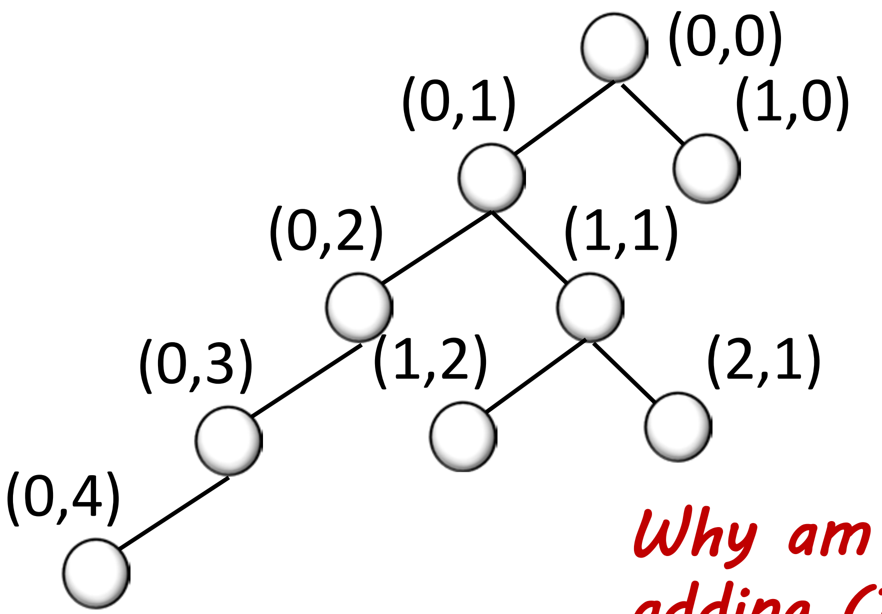
Find a treasure



Algorithms and Search

Search order: N, E, S, W

- Depth First Search (DFS)

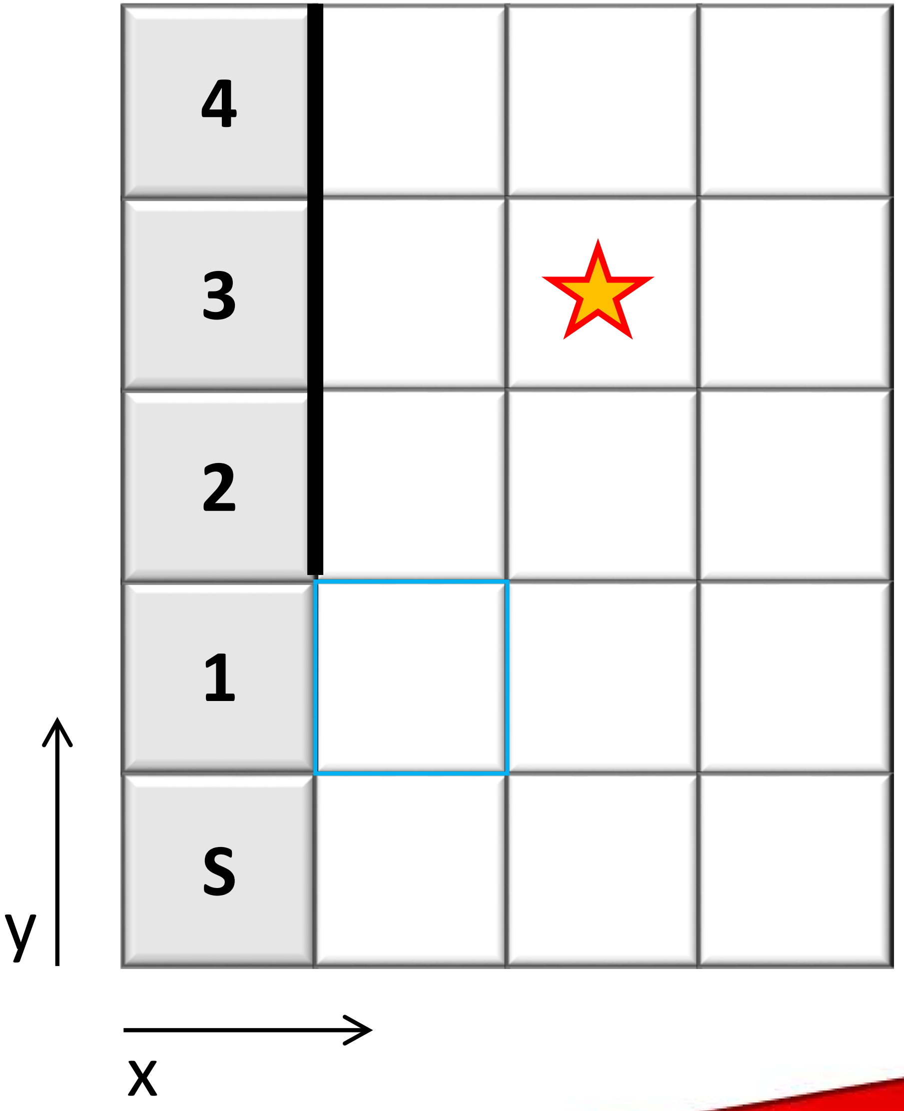


and so on…

Why am I not also adding (1,0)?

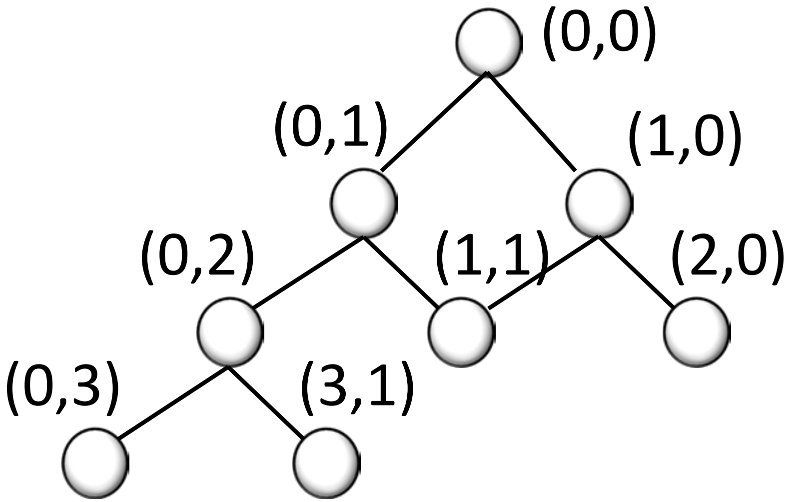
- Keep track of what is already on the frontier

Find a treasure



Algorithms and Search

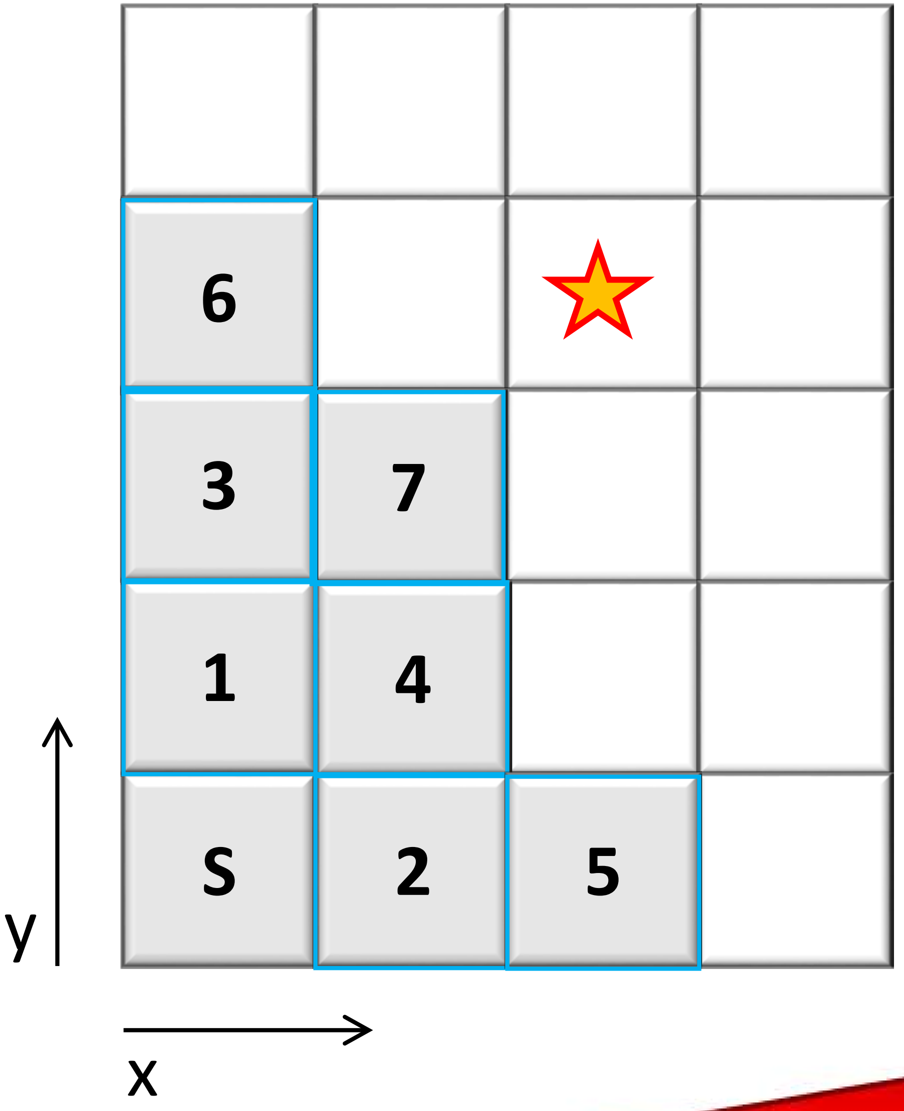
- Depth First Search (DFS)
- Breadth First Search (BFS)



and so on…

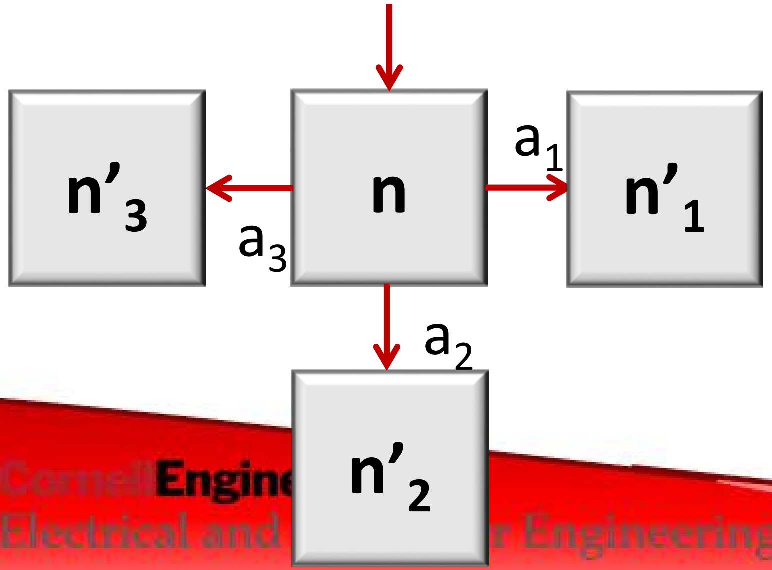
Search order: N, E, S, W

Find a treasure

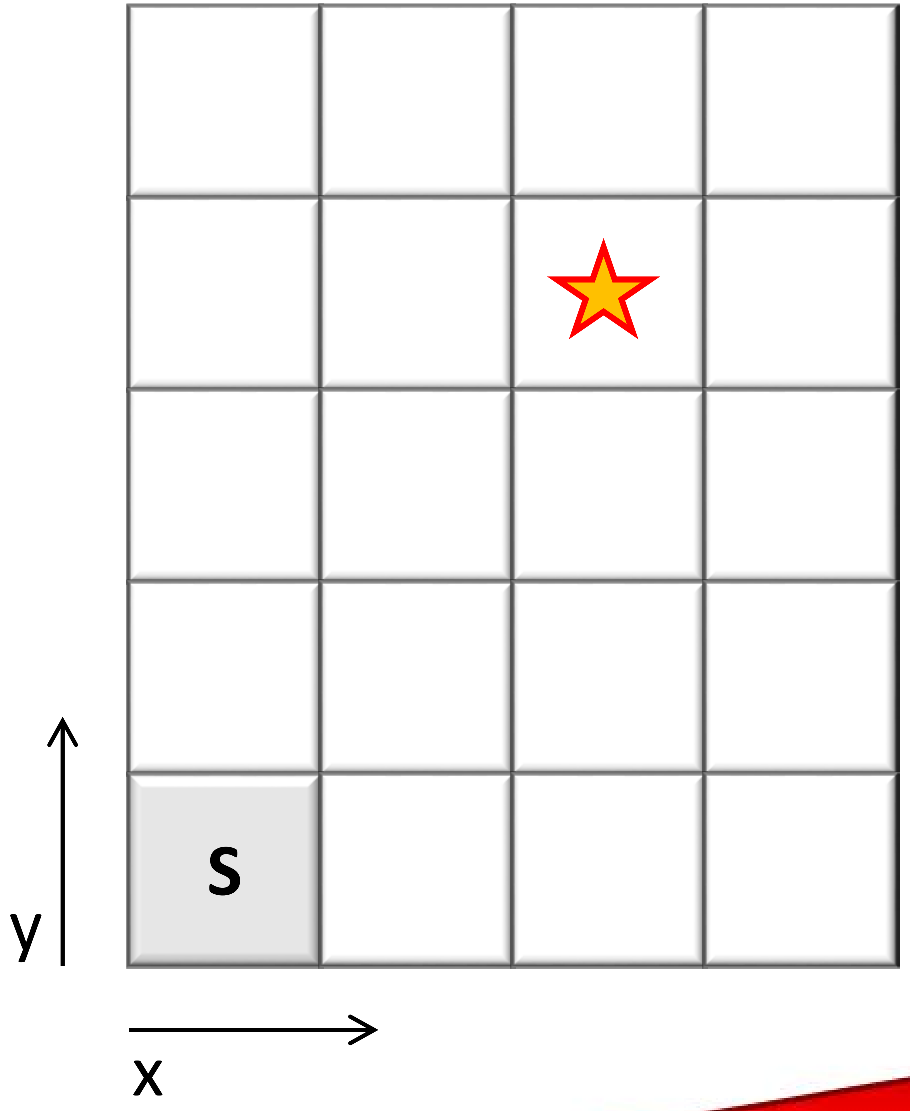


Algorithms and Search

- Depth First Search (DFS)
- Breadth First Search (BFS)
- Common graph structure
  - For every node, n
  - you have a set of actions, a
  - that moves you to a new node, n’

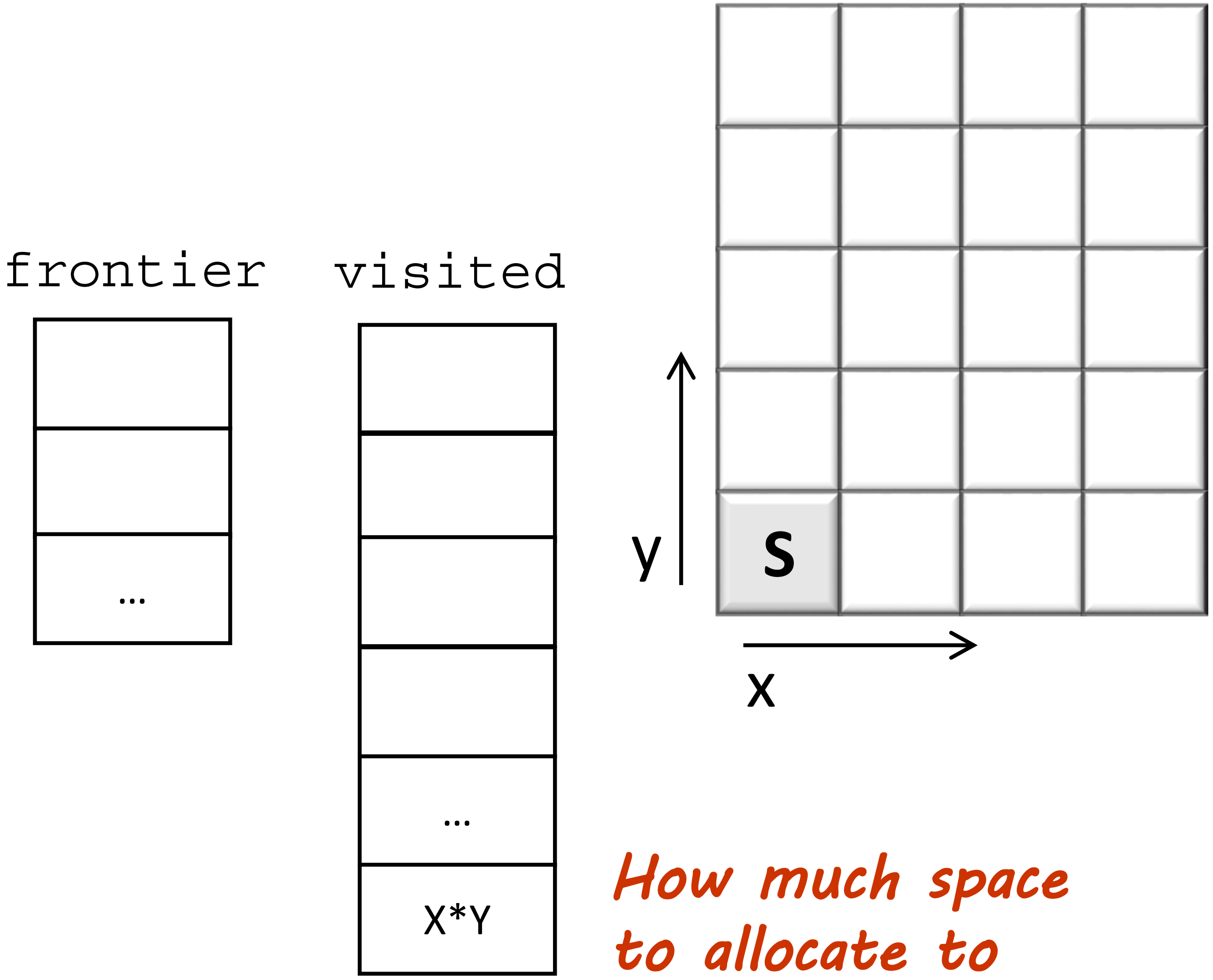


Find a treasure



General Search Algorithm

```
n = state(init)
frontier.append(n)
while(frontier not empty)
   n = pull state from frontier
   append n to visited
   if n = goal, return solution
   for all actions in n
      n’ = a(n)
      if n’ not visited
            append n’ to frontier
```



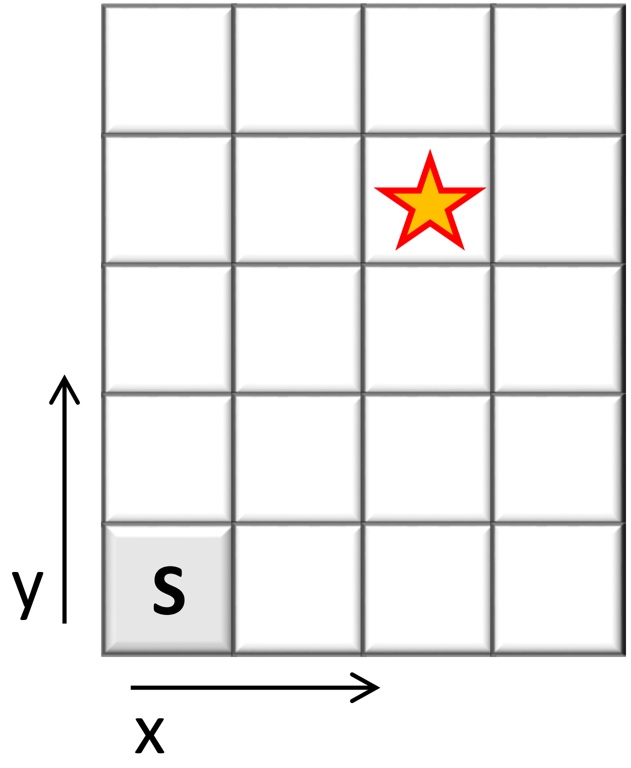
How much space to allocate to visited?

General Search Algorithm

- **Depth First Search (DFS)**

```
n = state(init)
frontier.append(n)
while(frontier not empty)
   n = pull state from frontier
   append n to visited
   if n = goal, return solution
   for all actions in n
      n' = a(n)
      if n' not visited
            append n' to frontier
```

frontier	visited
0,4	0,0
1,3	0,1
1,2	0,2
1,1	0,3
1,0	…
etc…	X*Y



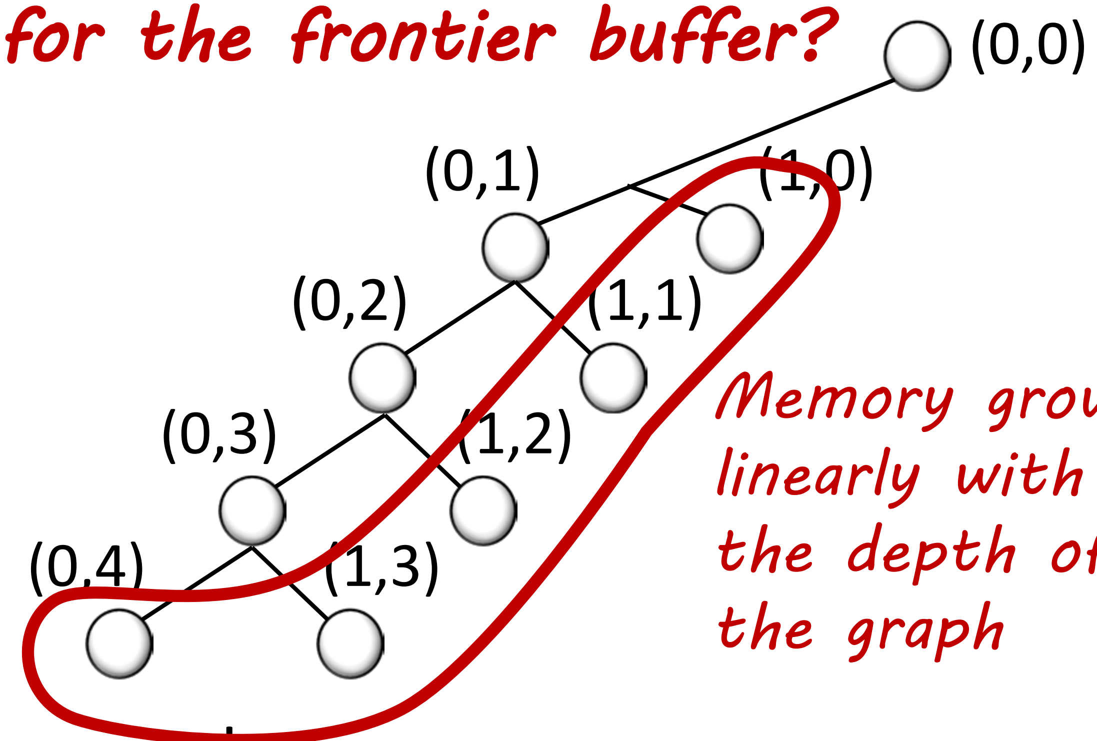
Type of Buffer?

Last-In First-Out (LIFO) Buffer

General Search Algorithm

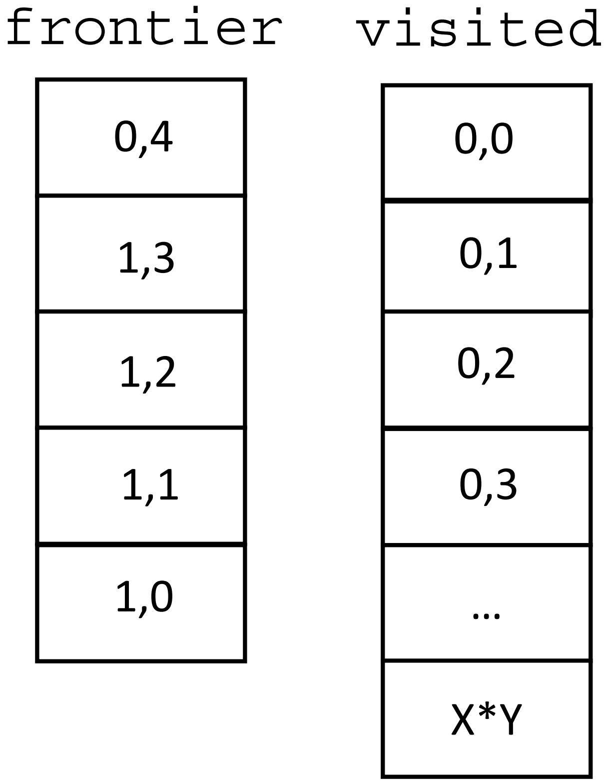
- **Depth First Search (DFS)**

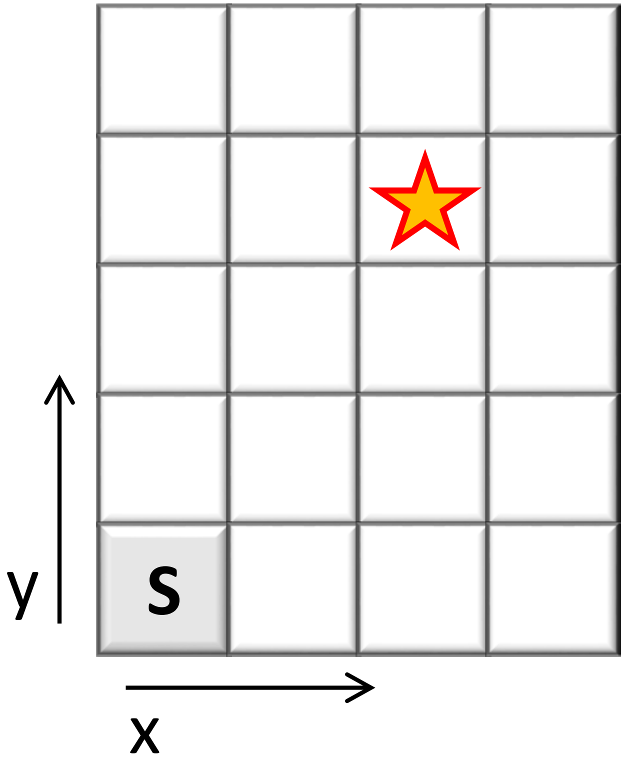
How much memory to allocate for the frontier buffer?



Memory grows linearly with the depth of the graph

and so on…





Type of Buffer?

Last-In First-Out (LIFO) Buffer

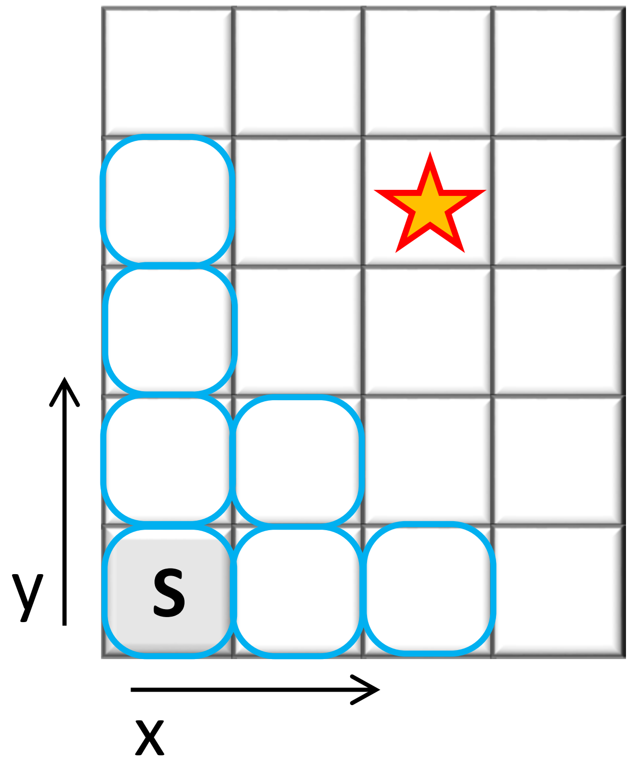
General Search Algorithm

- Depth First Search (DFS)
- **Breadth First Search (BFS)**

```
n = state(init)
frontier.append(n)
while(frontier not empty)
    n = pull state from frontier
    if n is goal, return solution
    for all actions in n
        n' = a(n)
        if n' not visited
            append n' to visited
            append n' to frontier
```

frontier
0,0
0,1
1,0
0,2
1,1
2,0
0,3

visited
0,0
0,1
1,0
0,2
…



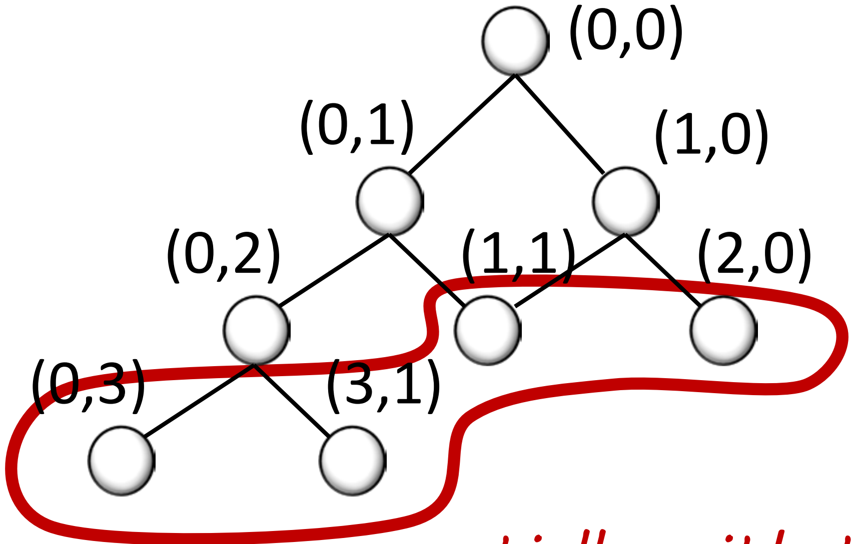
Type of Buffer?

First-In First-Out (FIFO) Buffer

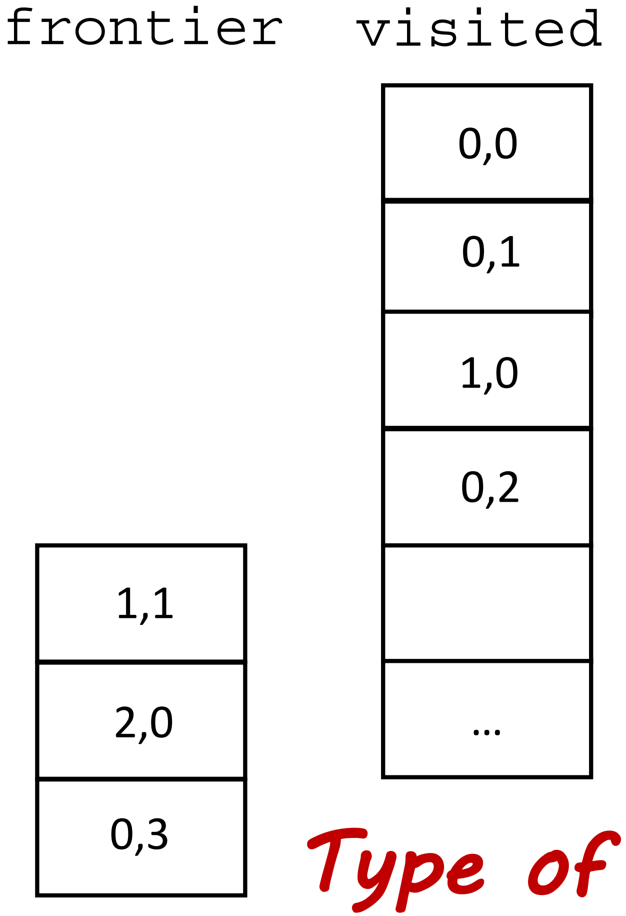
General Search Algorithm

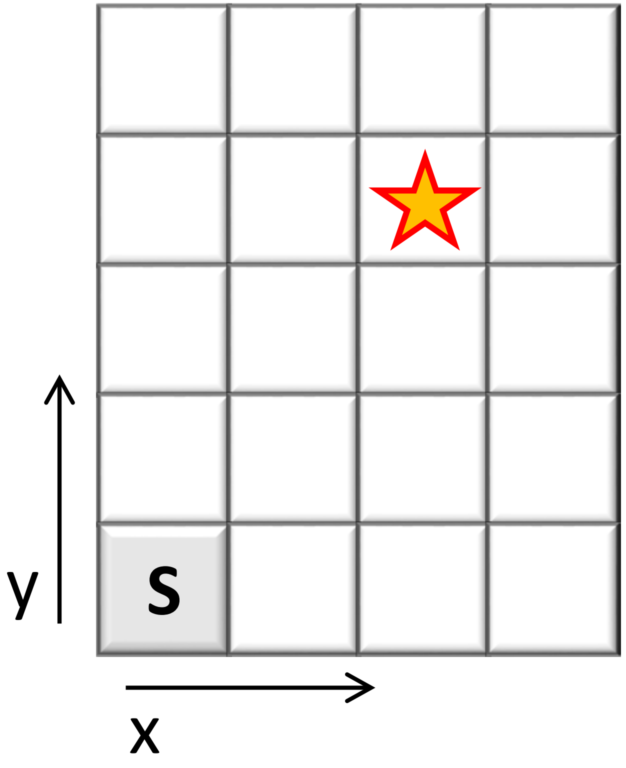
- Depth First Search (DFS)
- **Breadth First Search (BFS)**

How much memory to allocate for the frontier buffer?



Memory grows exponentially with the depth of the graph





Type of Buffer?

First-In First-Out (FIFO) Buffer

Maze Exploration

- What is the most efficient way to explore a maze with obstacles?
  - Hint: Your robot takes time to move!
  - Double hint: Your robot takes time to turn!

S			

DFS

4	5	6	7
3	16	17	8
2	15	18	9
1	14	19	10
S	13	12	11

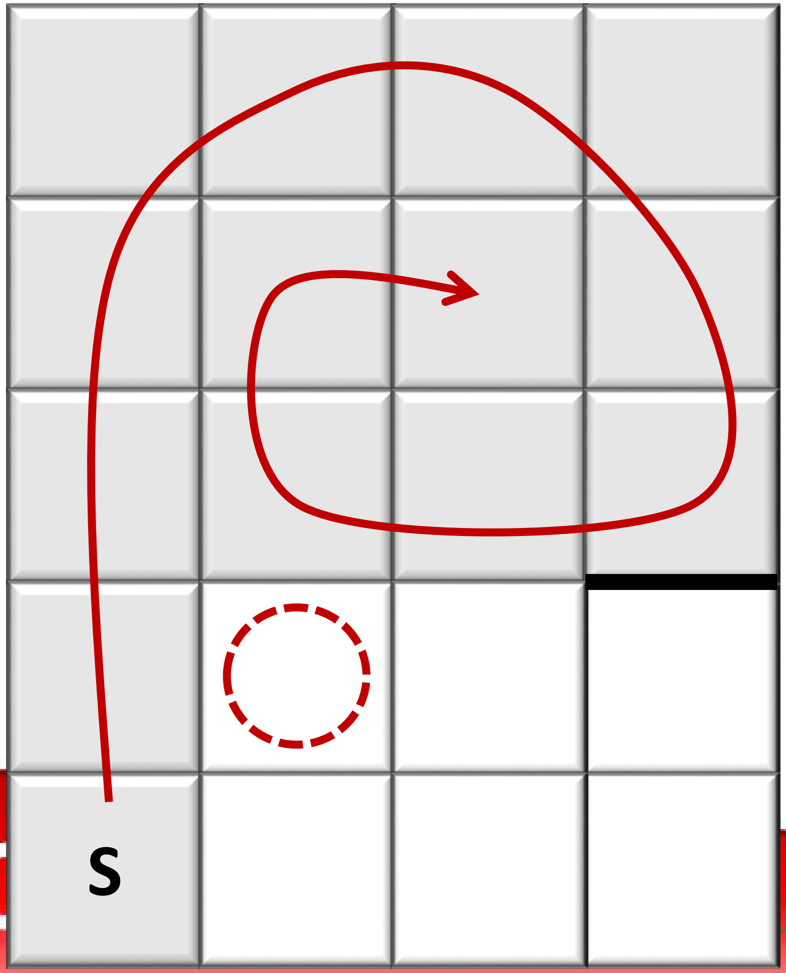
BFS

13	14	18	19
6	12	15	17
3	7	11	16
1	4	8	10
S	2	5	9

EC

Maze Exploration with Depth First Search

- *Can we be done already?*
  - *What path is the robot going to take?*
  - *What is the next frontier?*



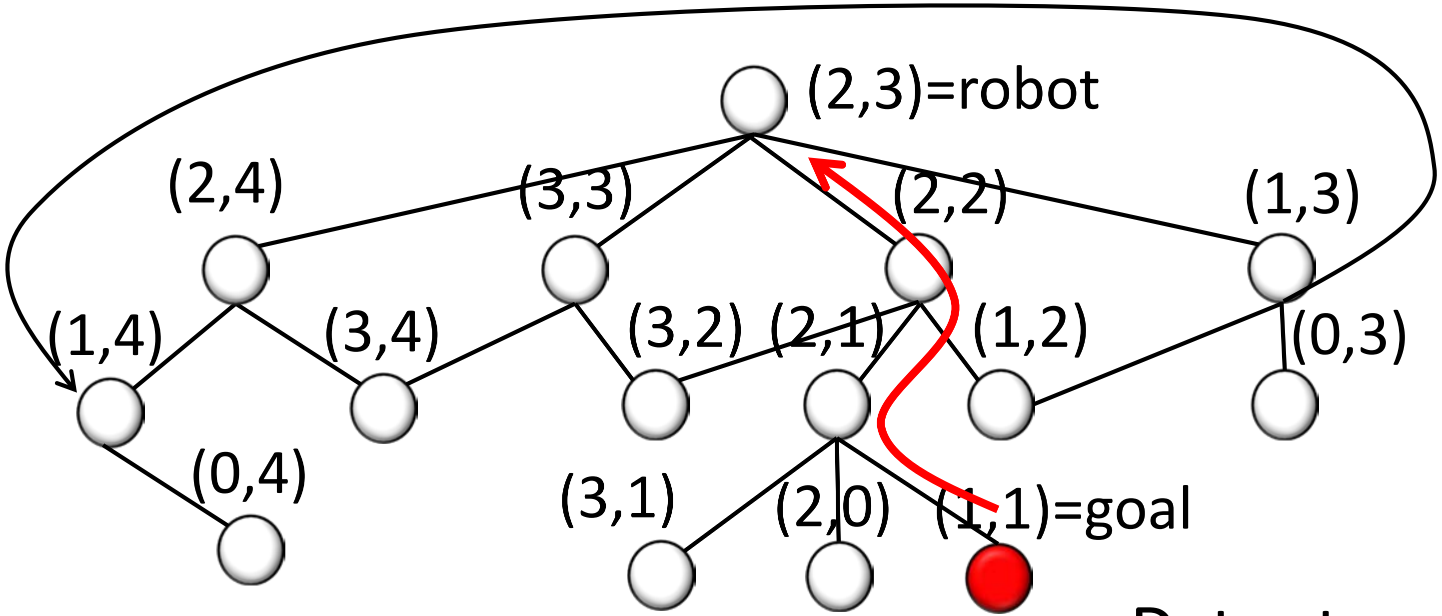
- *How should the robot plan how to get there?*

Procedure:

- Depth First exploration
- Breadth First Search to find the shortest path to the frontier
- (sequence of actions to get to the frontier)

Search order: N, E, S, W

Breadth First Search



What is the shortest path to the goal?

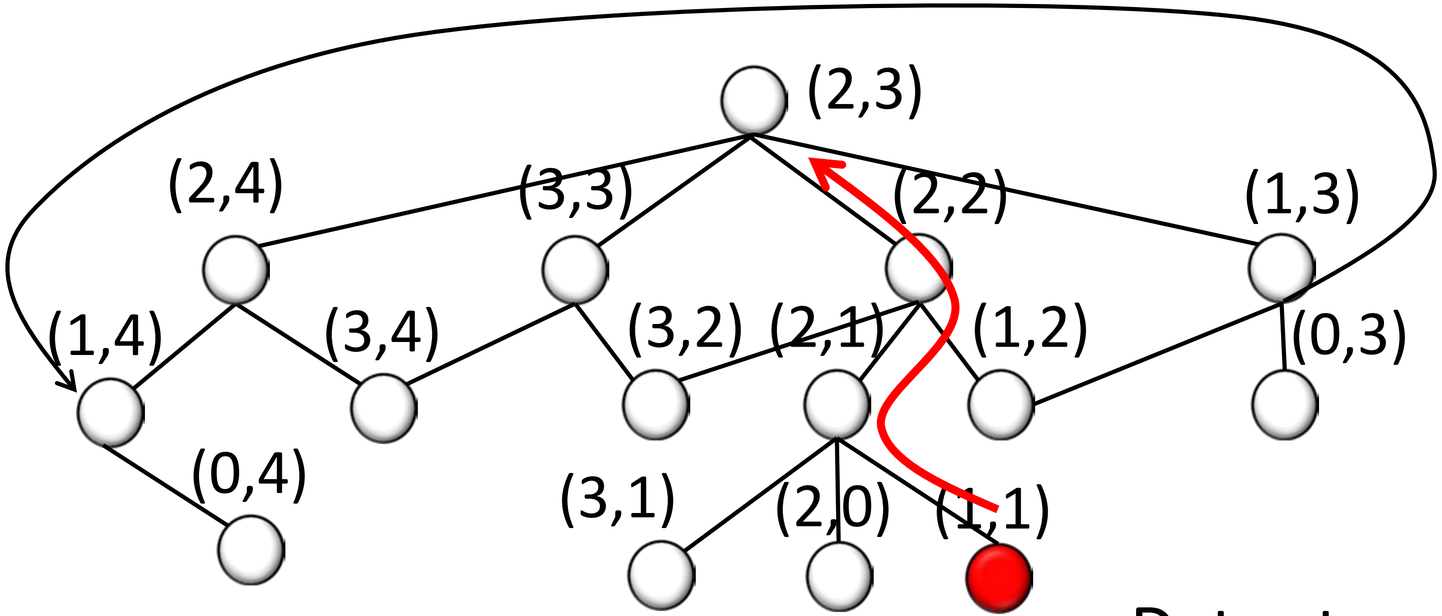
Data structure

- n.state
- n.parent

(0,4)	(1,4)	(2,4)	(3,4)
(0,3)	(1,3)	R	(3,3)
	(1,2)	(2,2)	(3,2)
	G	(2,1)	(3,1)
		(2,0)	

Breadth First Search

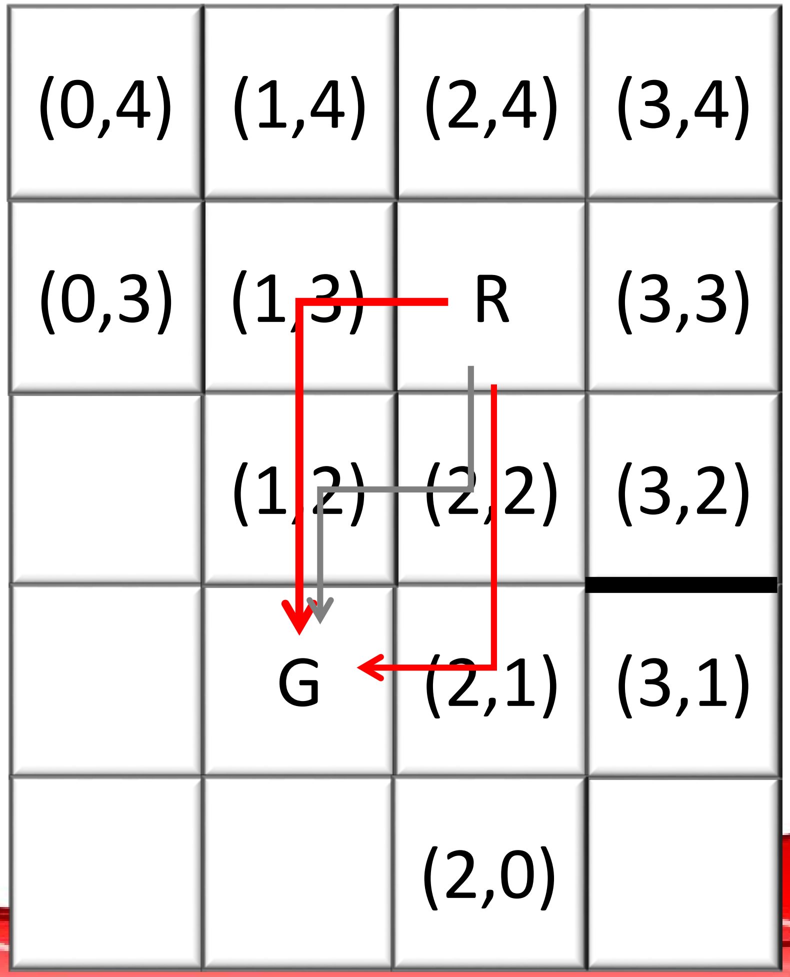
Search order: N, E, S, W



Does not include the cost to get there…

Data structure

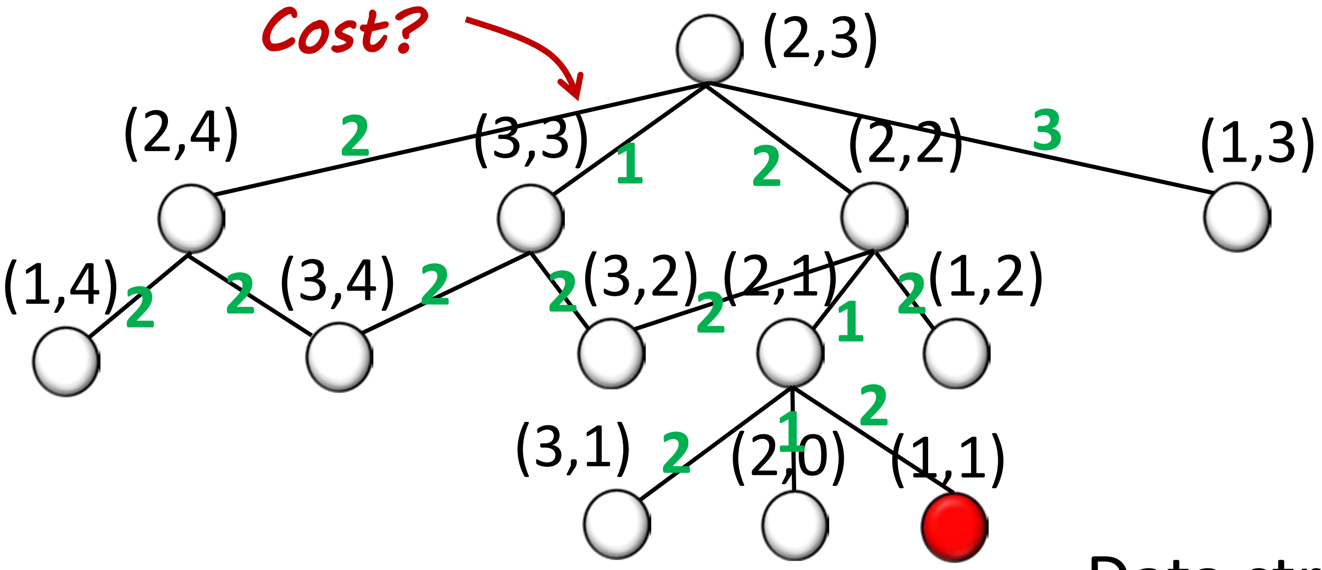
- n.state
- n.parent



Breadth First Search and Dijkstra's Algorithm

Search order: N, E, S, W

- Dijkstra's Algorithm: consider parent cost



What node to expand next?

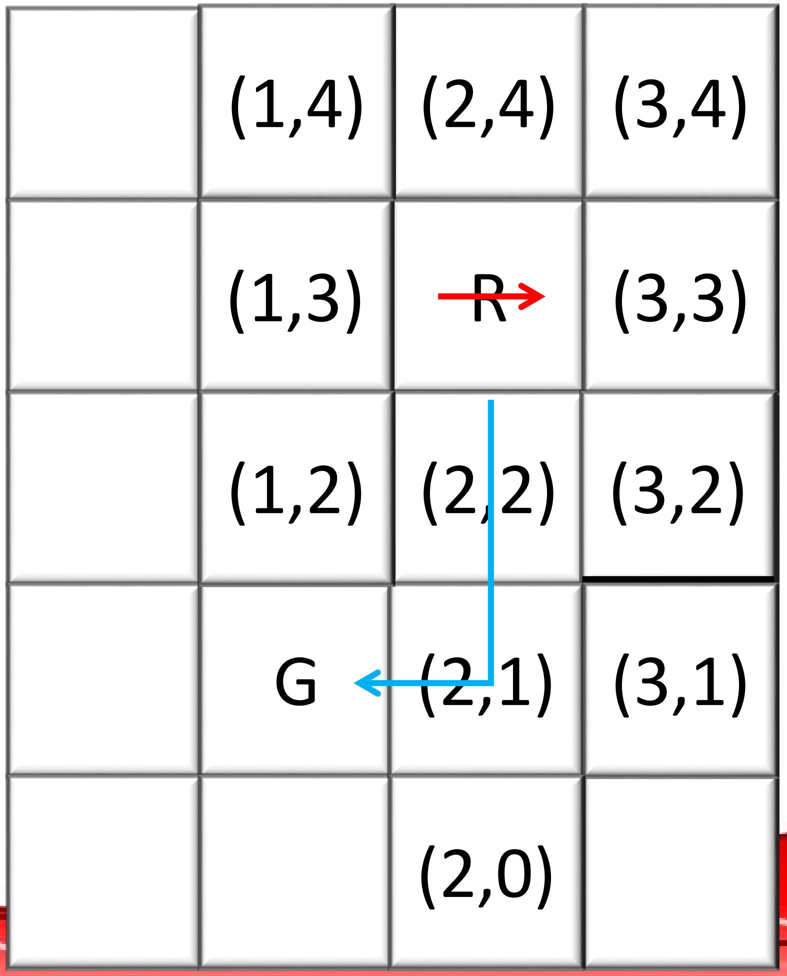
...may save some computation!

Data structure

- n.state
- n.parent
- n.cost
- n.action

What cost heuristic could we add?

- Go straight, cost 1
- Turn quadrant, cost 1



Go Build Robots!

Class website: https://cei-lab.github.io/ece3400/
Piazza: https://piazza.com/cornell/fall2017/ece3400/home