Graph Search

Fast Robots, ECE4160/5160, MAE 4190/5190

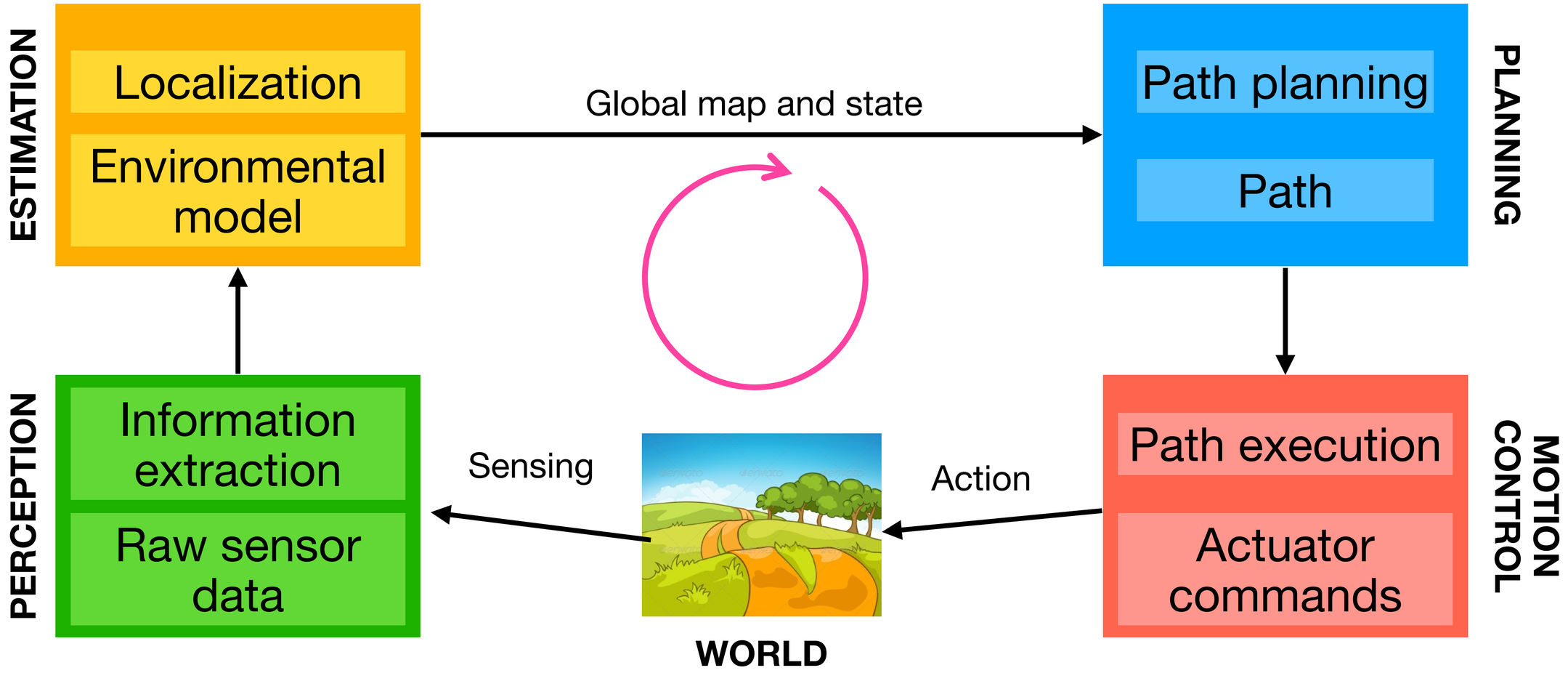
E. Farrell Helbling, 3/17/26

Class Action Items

- Lab 7: Kalman Filtering
  - The first three tasks were discussed in class last week, please review slides before you attend your lab section.
  - Getting the KF working in simulation is pretty easy, getting it to work on the robot is challenging. Do not leave this to the night before!
- Lab 8: Stunts, we will have test tracks set up this weekend for those that want to start this lab early. The lab is posted, you have two options, the flip or the drift. Please try to get this done next week before spring break (or do it after spring break). I don’t recommend taking the robot on a plane!

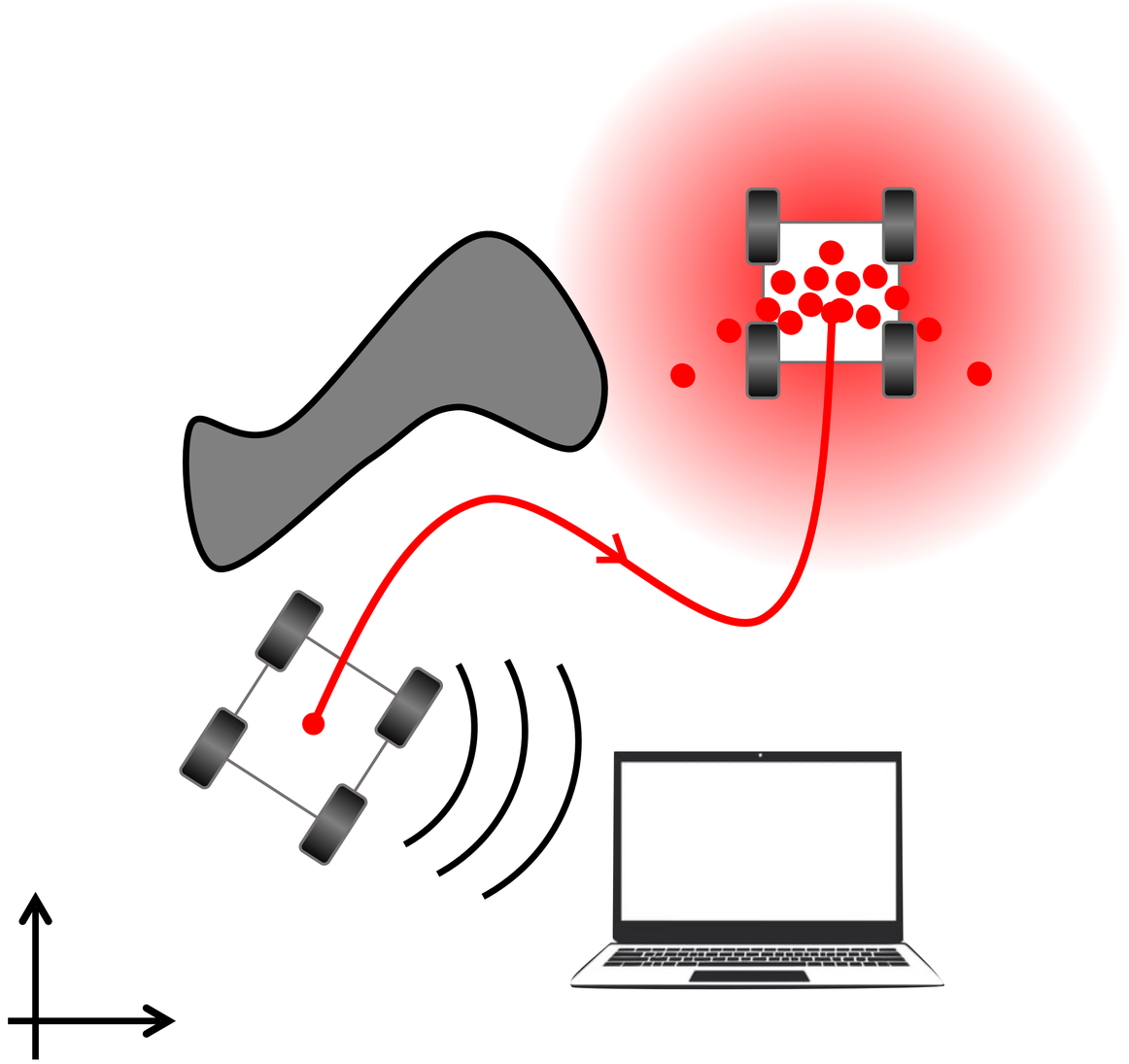
Navigation

- **Break the problem down:** localization, map building, path planning

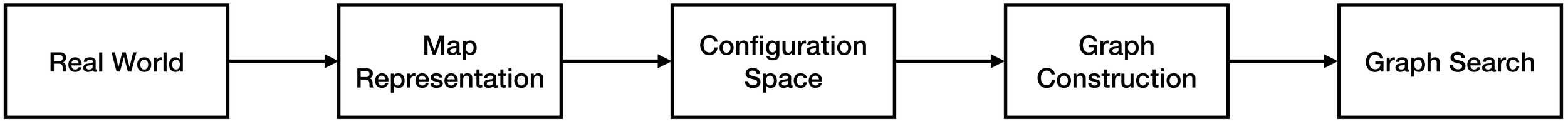


Navigation

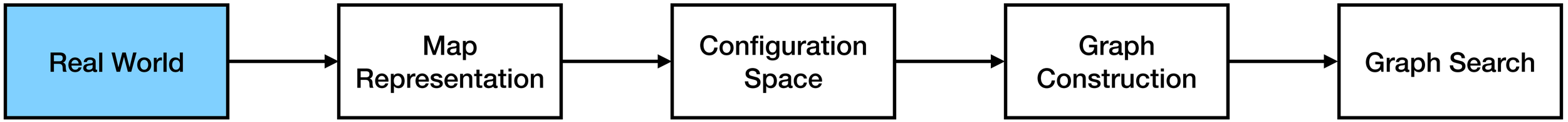
- Local planners (obstacle avoidance)
- Global localization and planning
  - Map representations
    - Continuous
    - Discrete
    - Topological
  - Maps as graphs
    - Graph search algorithms
      - Breadth first search
      - Depth first search
      - Dijkstras
      - A*

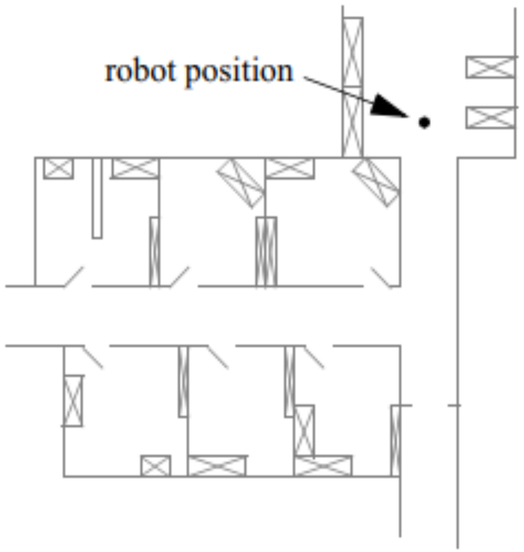


Modeling path planning as a graph search problem

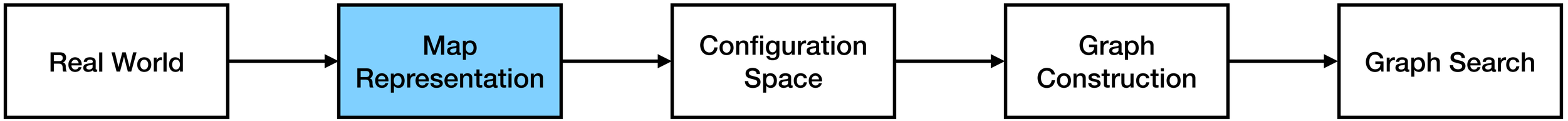


Modeling path planning as a graph search problem

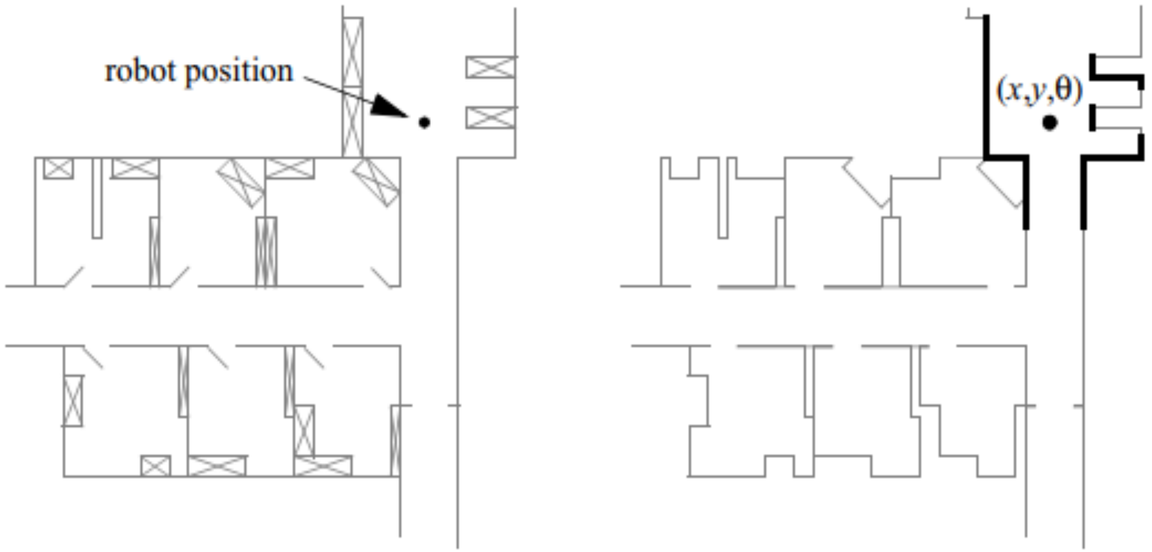




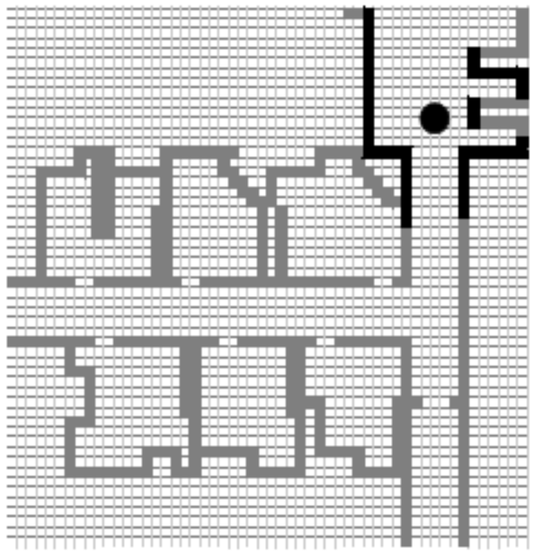
Modeling path planning as a graph search problem



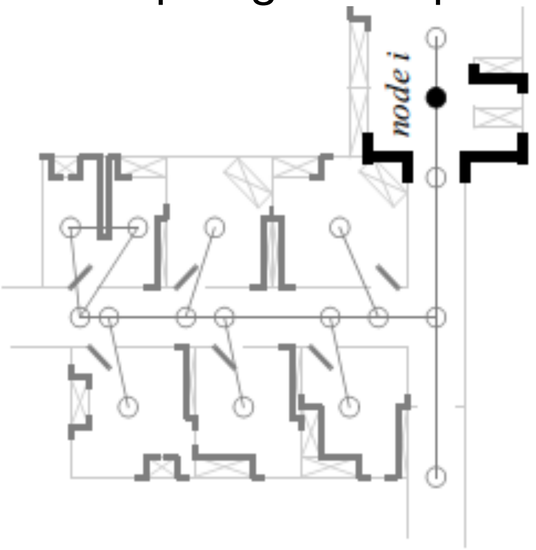
Line-based map



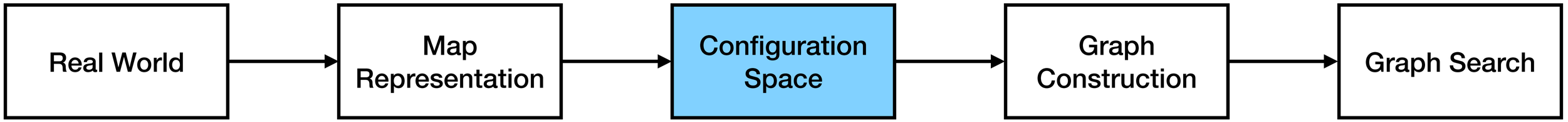
Occupancy Grid

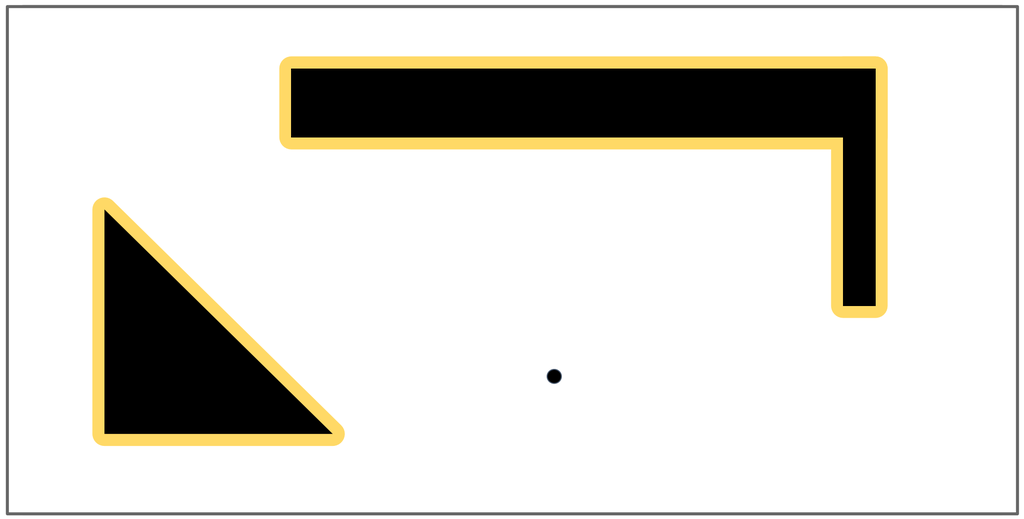


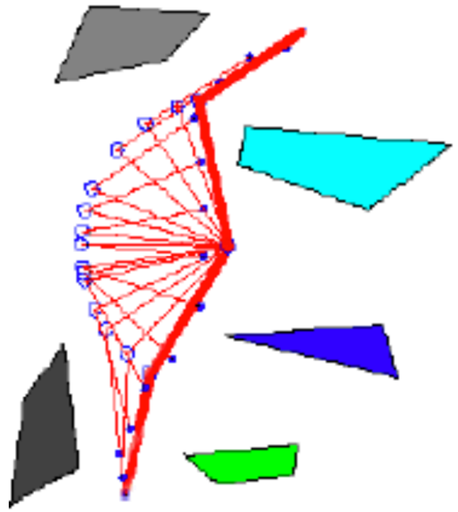
Topological Map

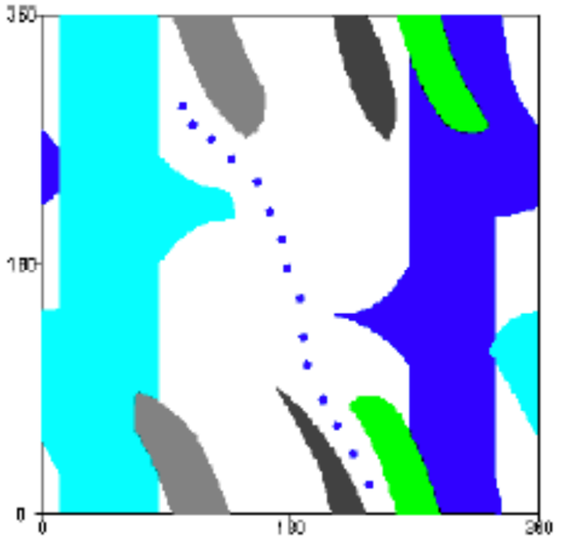


Modeling path planning as a graph search problem

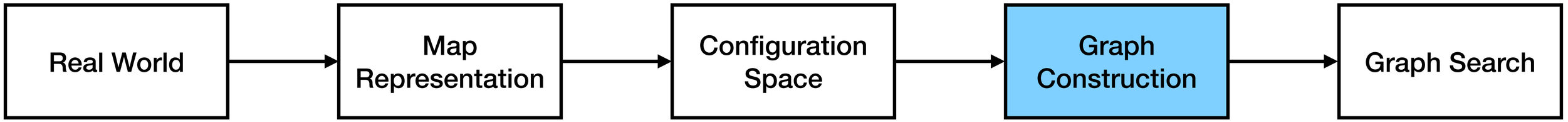


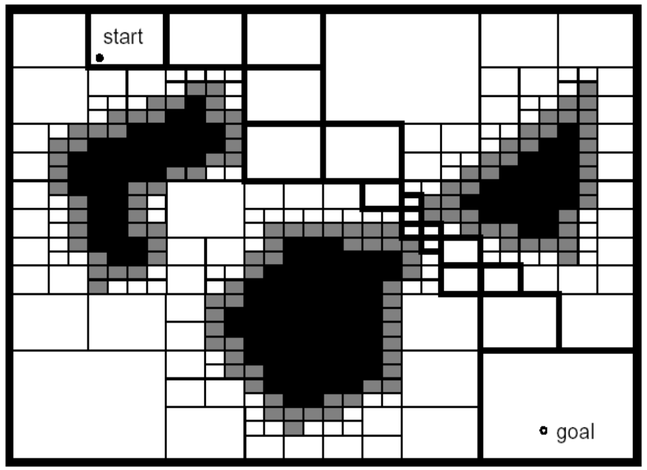


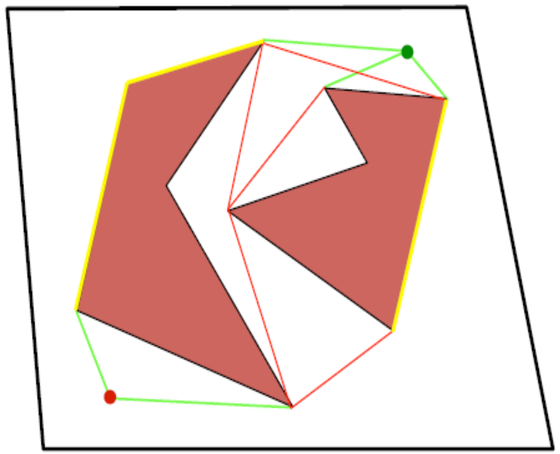




Modeling path planning as a graph search problem

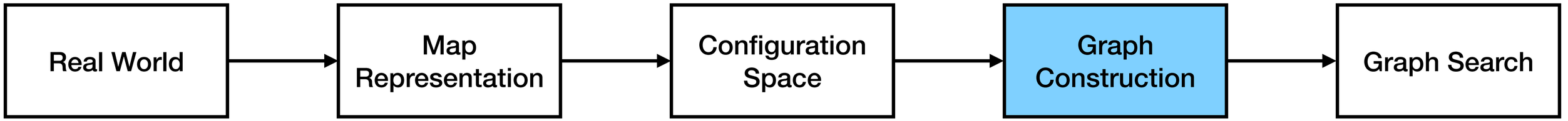


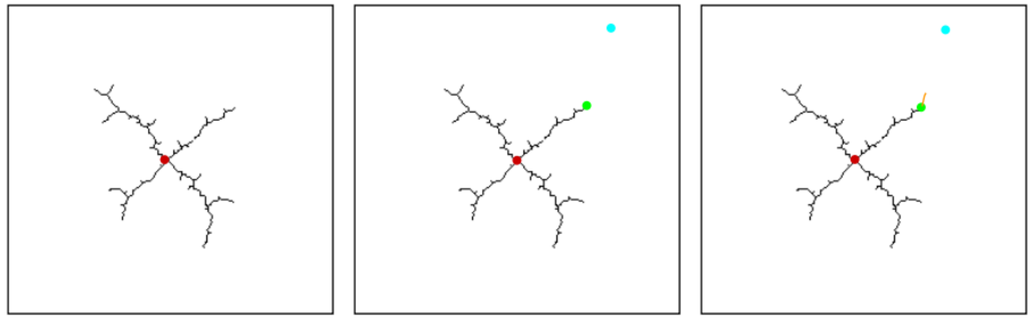


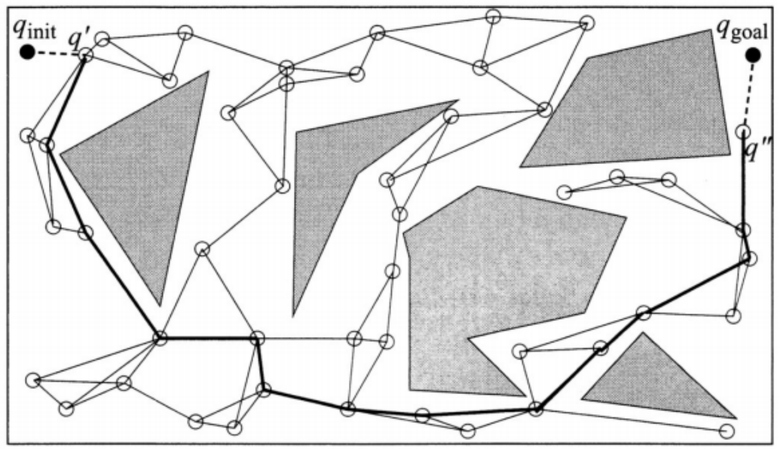


- Topological graphs
  - Cell decomposition
  - Visibility Graphs

Modeling path planning as a graph search problem



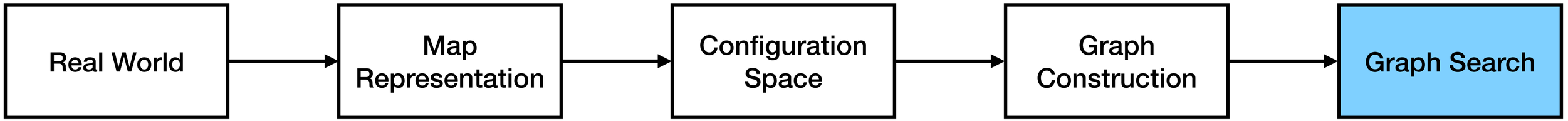




- Topological graphs
  - Cell decomposition
  - Visibility Graphs
  - RRT
  - PRM

https://atsushisakai.github.io/PythonRobotics/modules/5_path_planning/rrt/rrt.html

Modeling path planning as a graph search problem

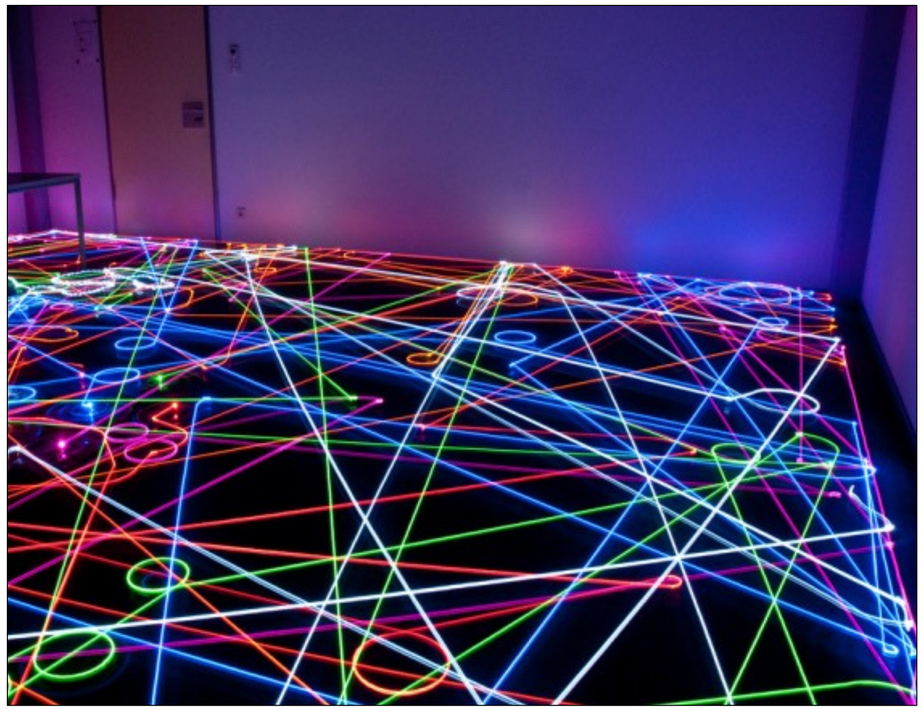


- Breadth First
- Depth First
- Dijkstra’s
- A*

Graph Search

Graph Search

- What is the simplest thing to do?
  - Random or brute force search

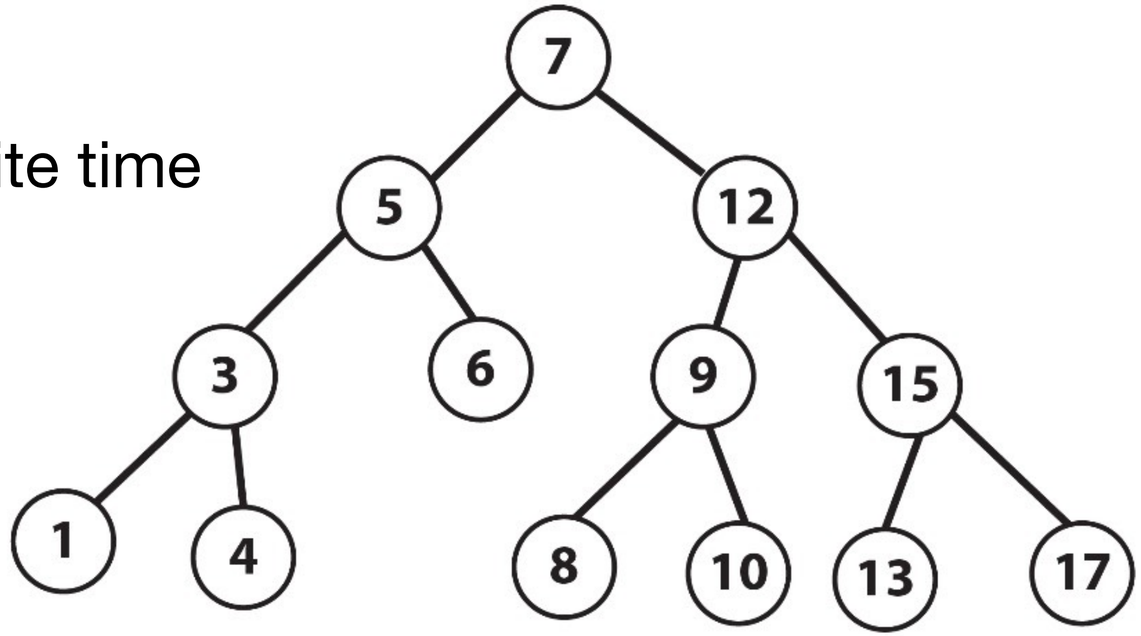


Graph Search

- What is the simplest thing to do?
  - Random or brute force search
- Other methods?
  - Uninformed search
    - Depth First Search (DFS)
    - Breadth First Search (BFS)
    - Dijkstra's Search (LCFS)
  - Informed Search
    - Greedy
    - A*
    - (and many more)

Comparing Search Algorithms

- Vocabulary: node, edge, parents/children, branching factor, depth
- Definitions
  - Complete
    - Guaranteed to find a solution in finite time
  - Time complexity
    - Worst-case run time
  - Space complexity
    - Worst-case memory
  - Optimality
    - A search is optimal if it is complete, and only returns cost-minimizing solutions

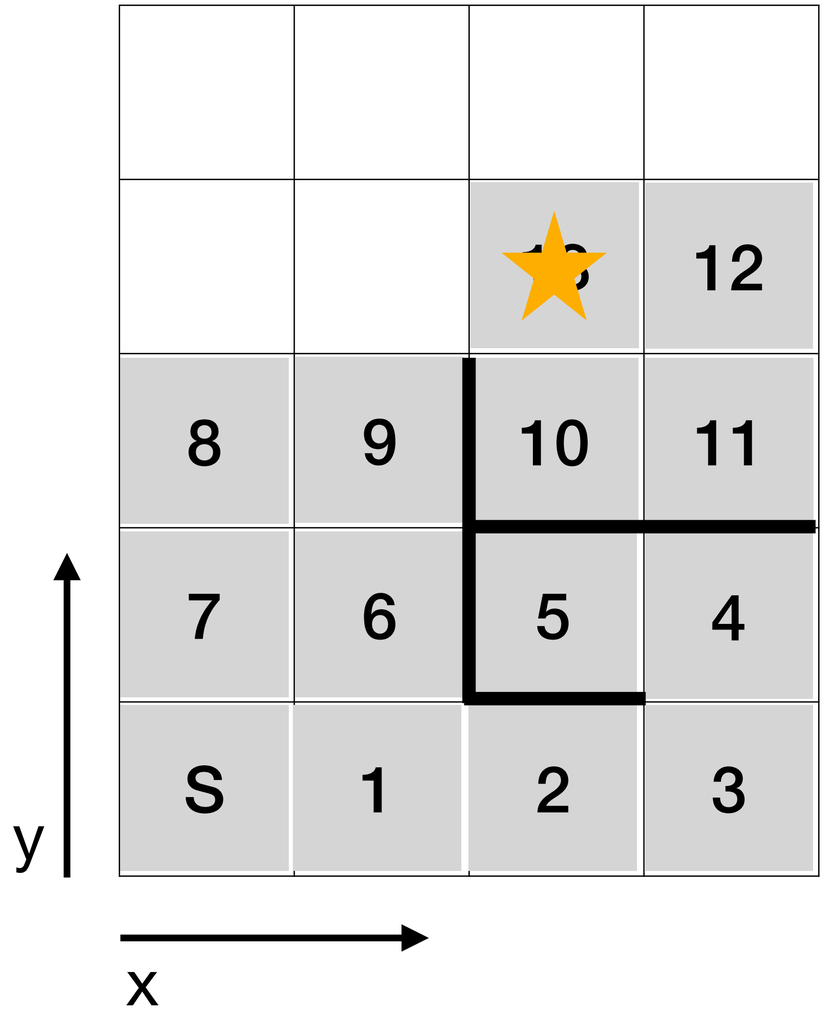


Algorithms and Search

- What is the simplest thing to do?
  - Random or brute force search
  - How many grid traversals will brute force take?
    - First establish a search order: N, E, S, W

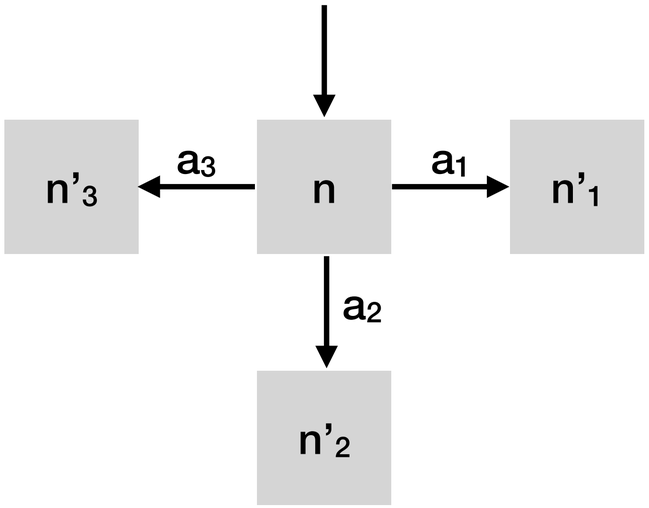
Algorithms and Search

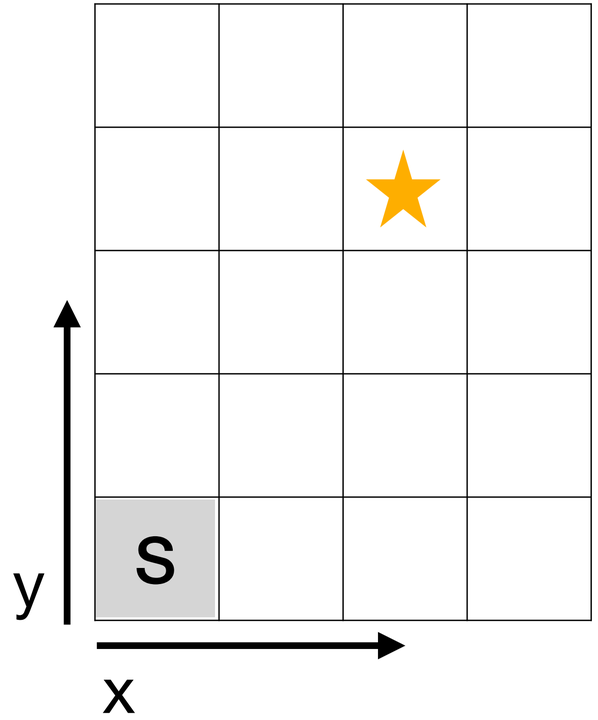
- What is the simplest thing to do?
  - Random or brute force search
  - How many grid traversals will brute force take?
    - First establish a search order: N, E, S, W
    - Advance x first, then increment y and decrease x, etc.



Search Algorithms, General

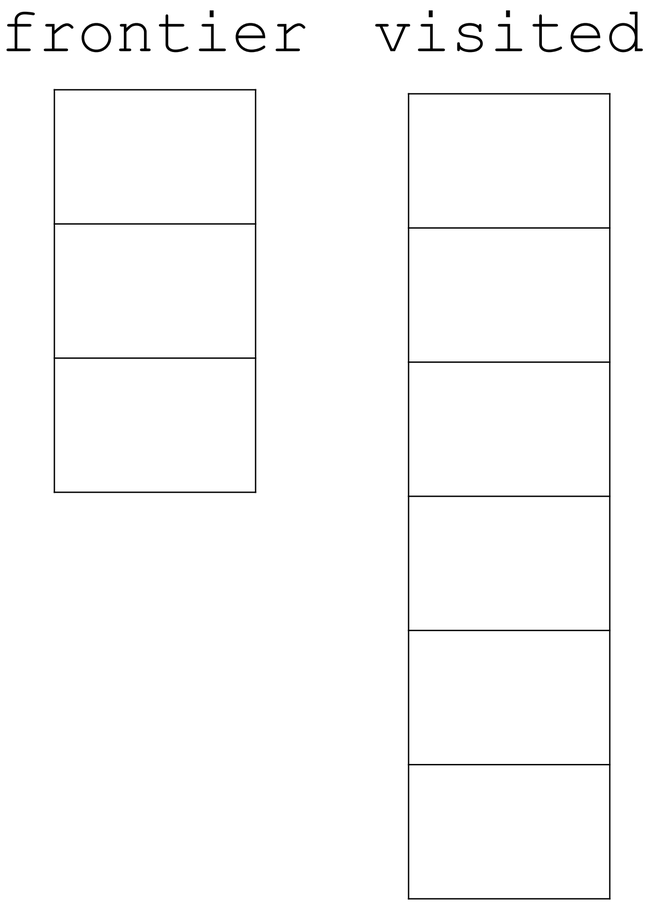
- For every node, n
- There is a set of actions, a
- That moves you to a new node, n’

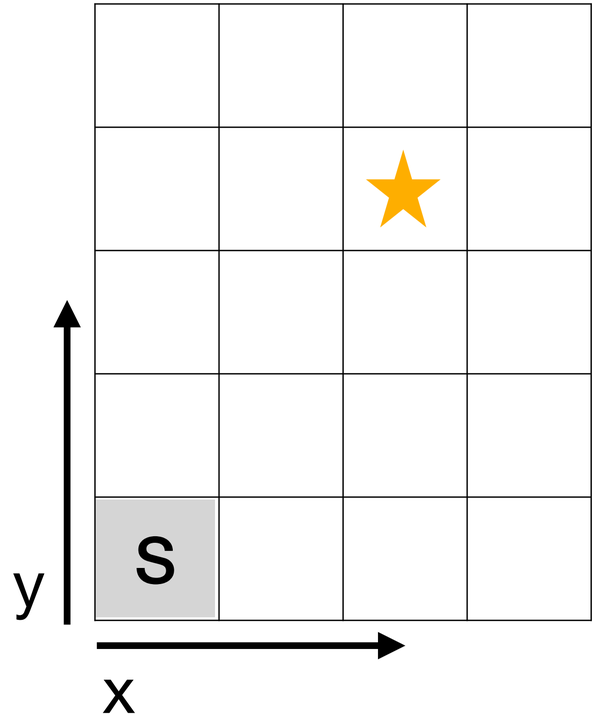




Search Algorithms, General

```
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
```

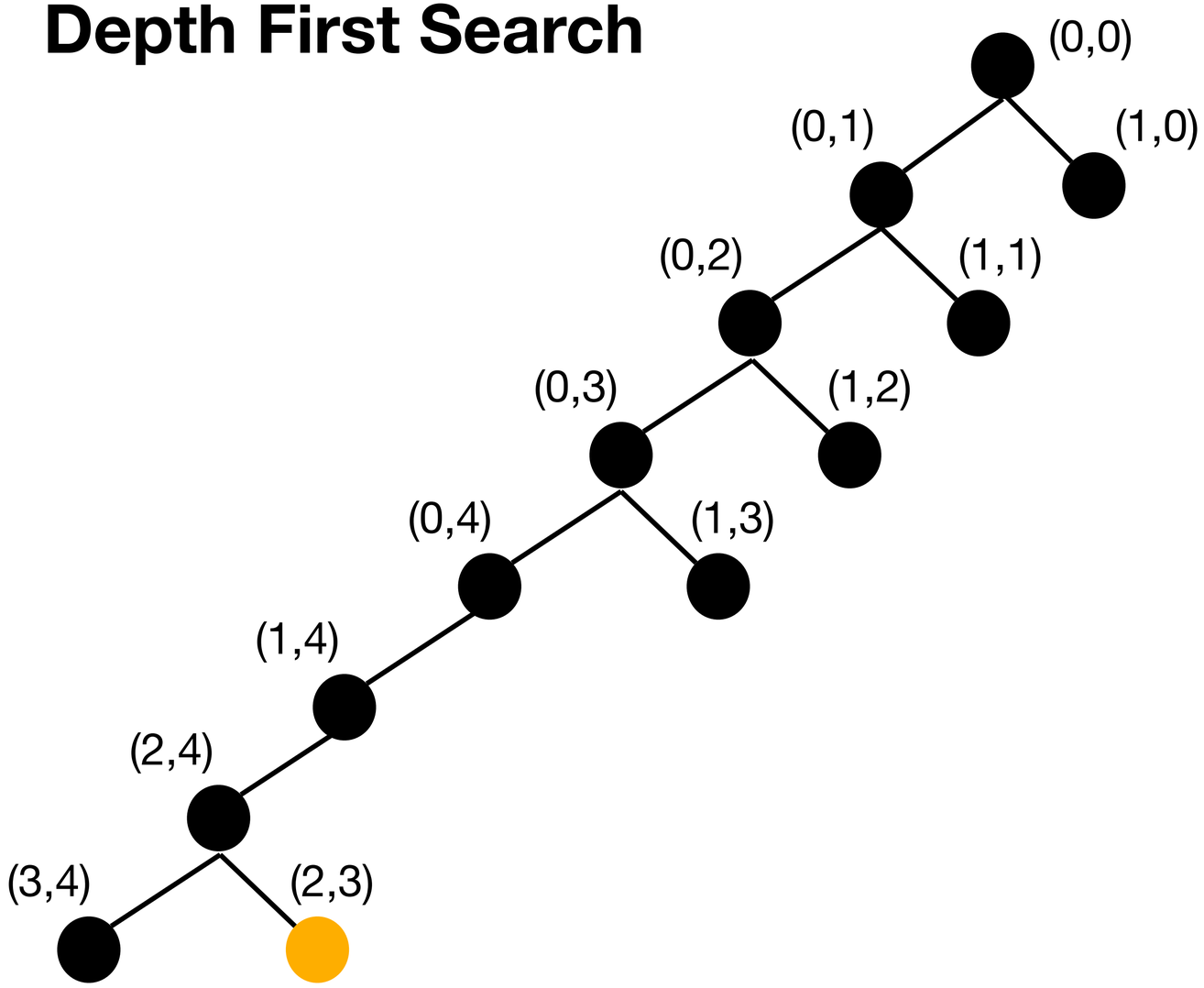




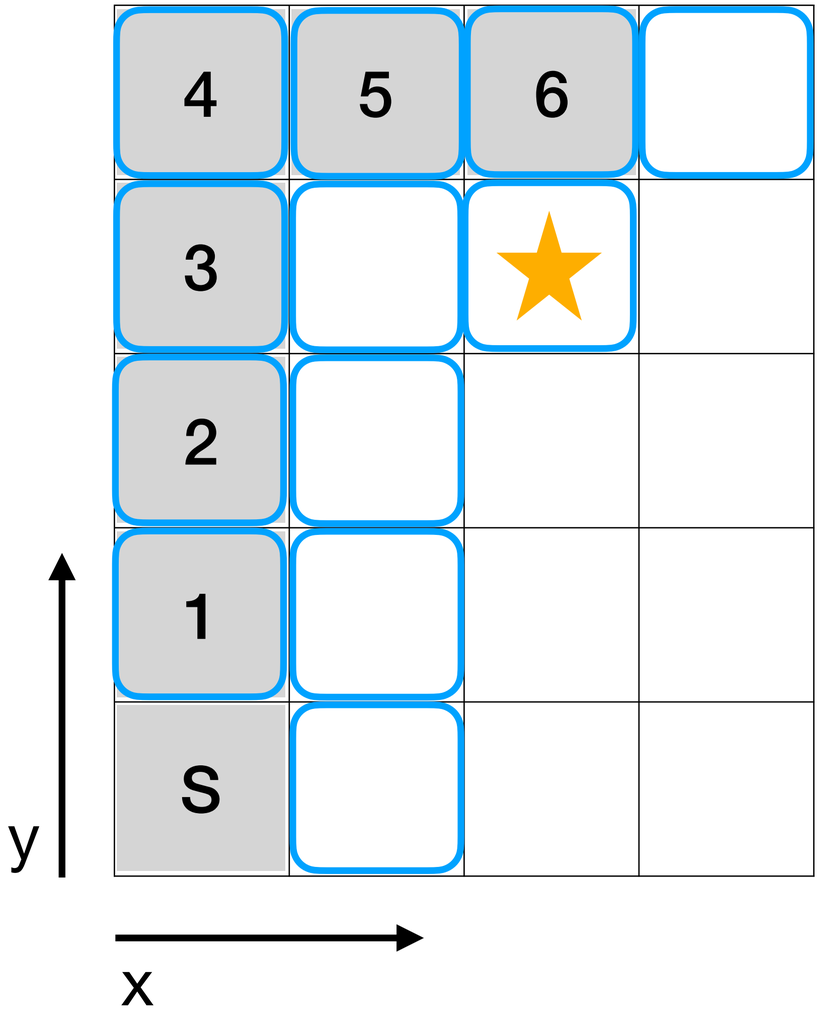
How much space do we allocate to the buffers?

Uninformed Search

Depth First Search

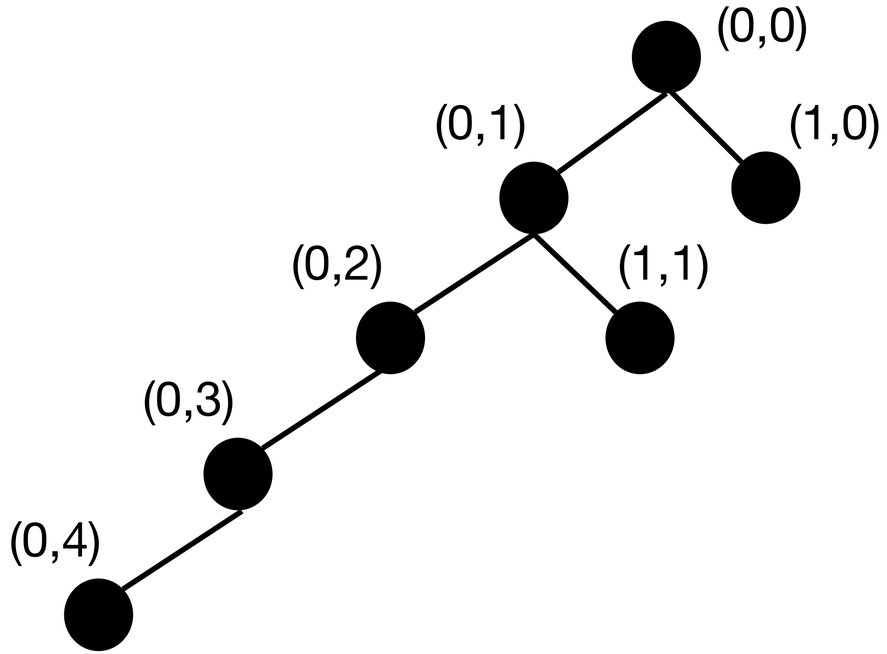


Search Order: N, E, S, W

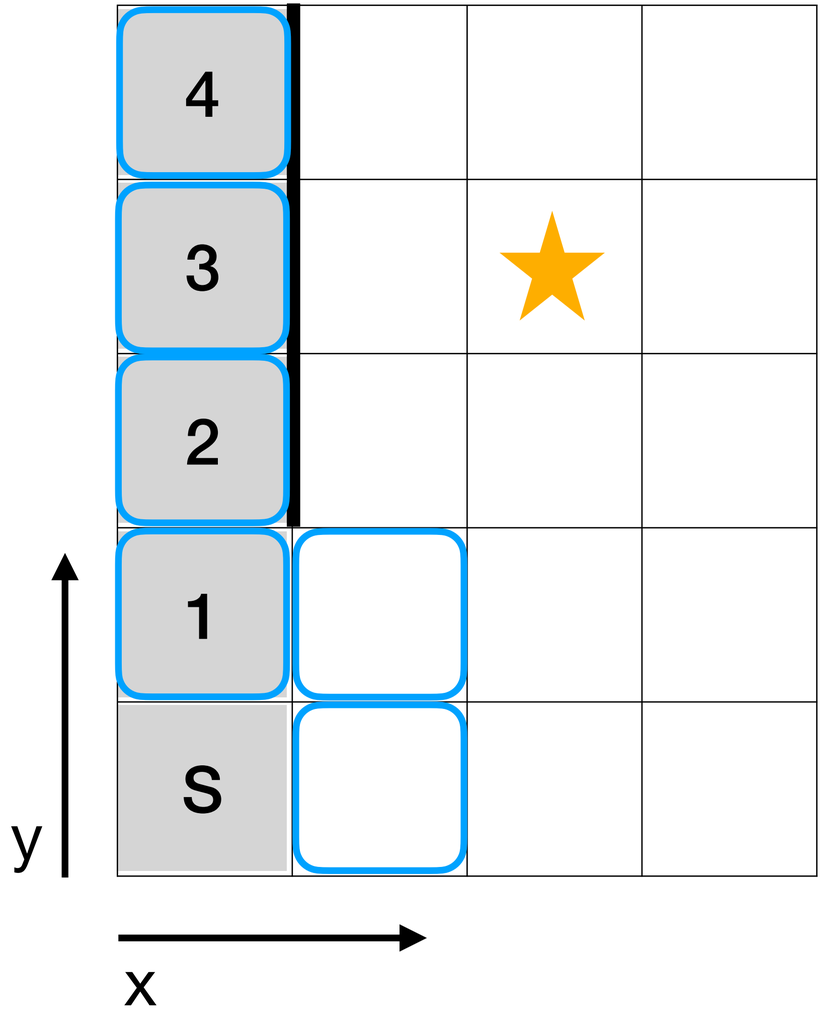


Uninformed Search

Depth First Search

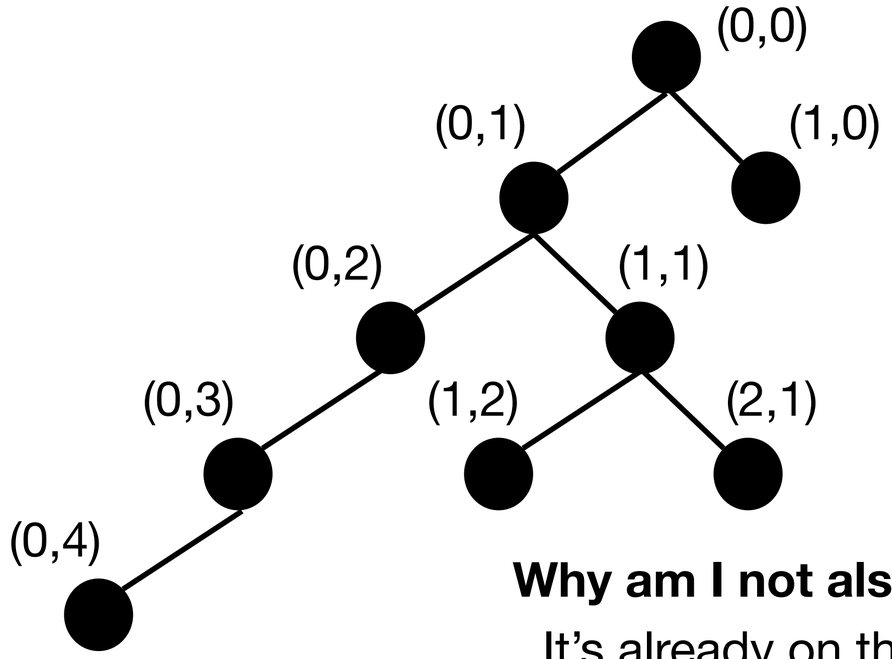


Search Order: N, E, S, W



Uninformed Search

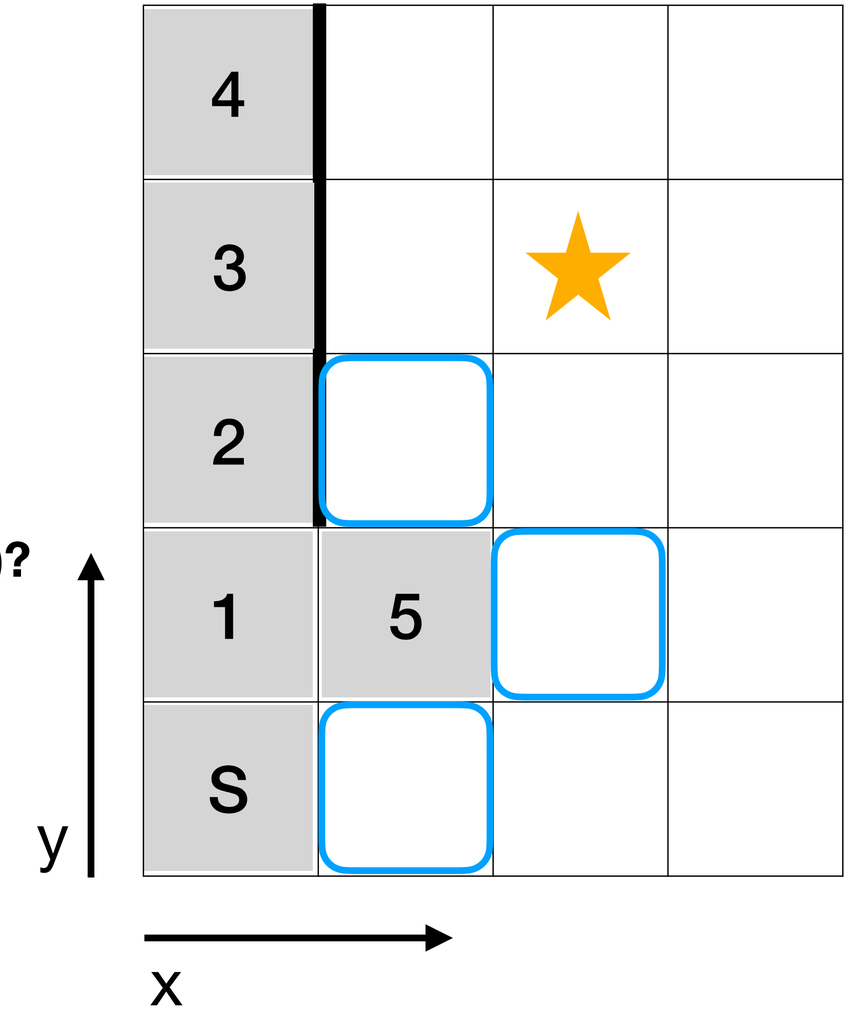
Depth First Search



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

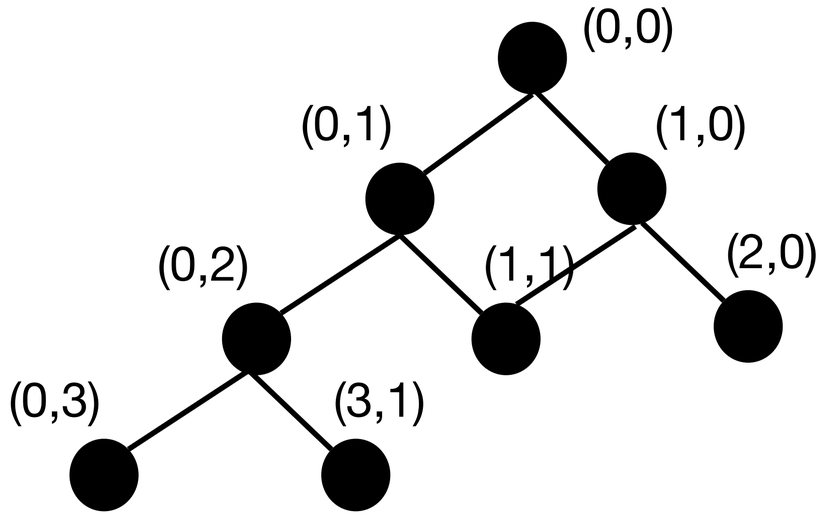
It's already on the frontier

Search Order: N, E, S, W



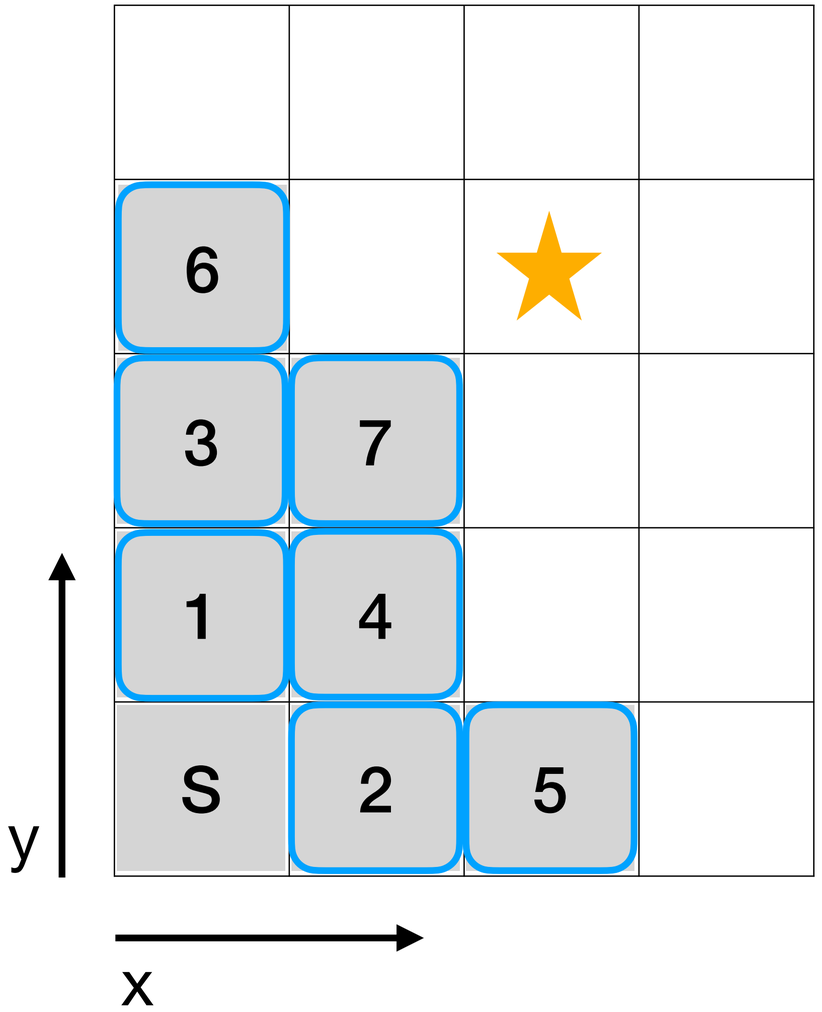
Uninformed Search

Breadth First Search



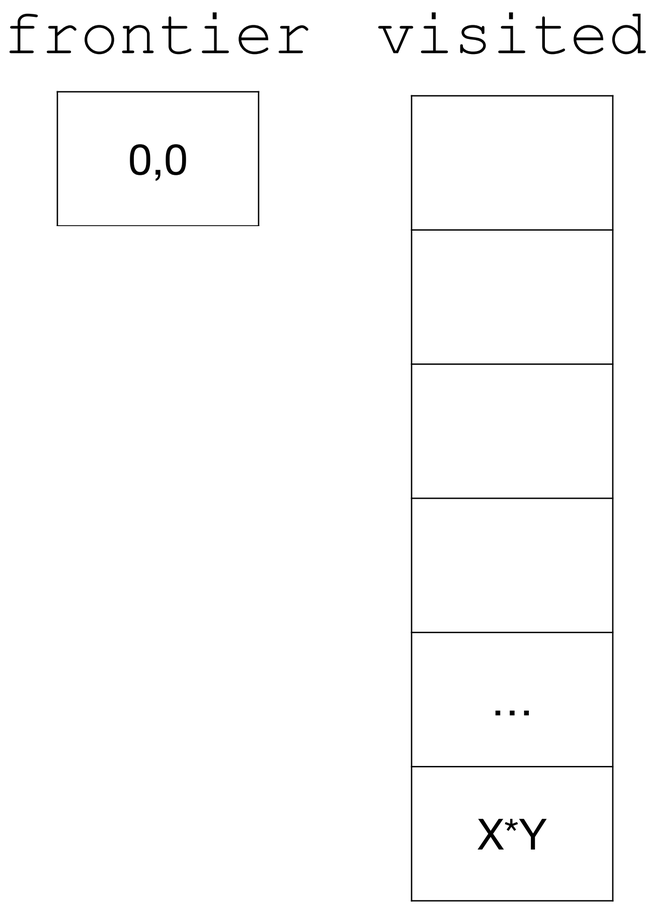
and so on…

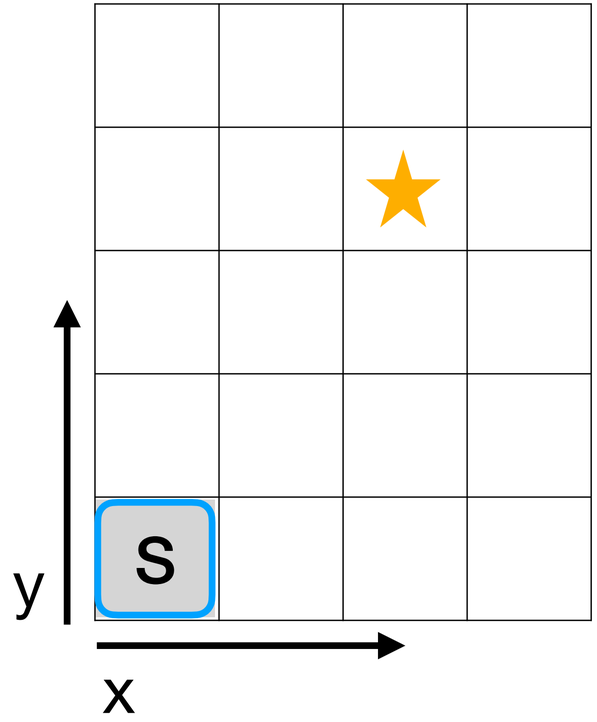
Search Order: N, E, S, W



Depth First Search (DFS)

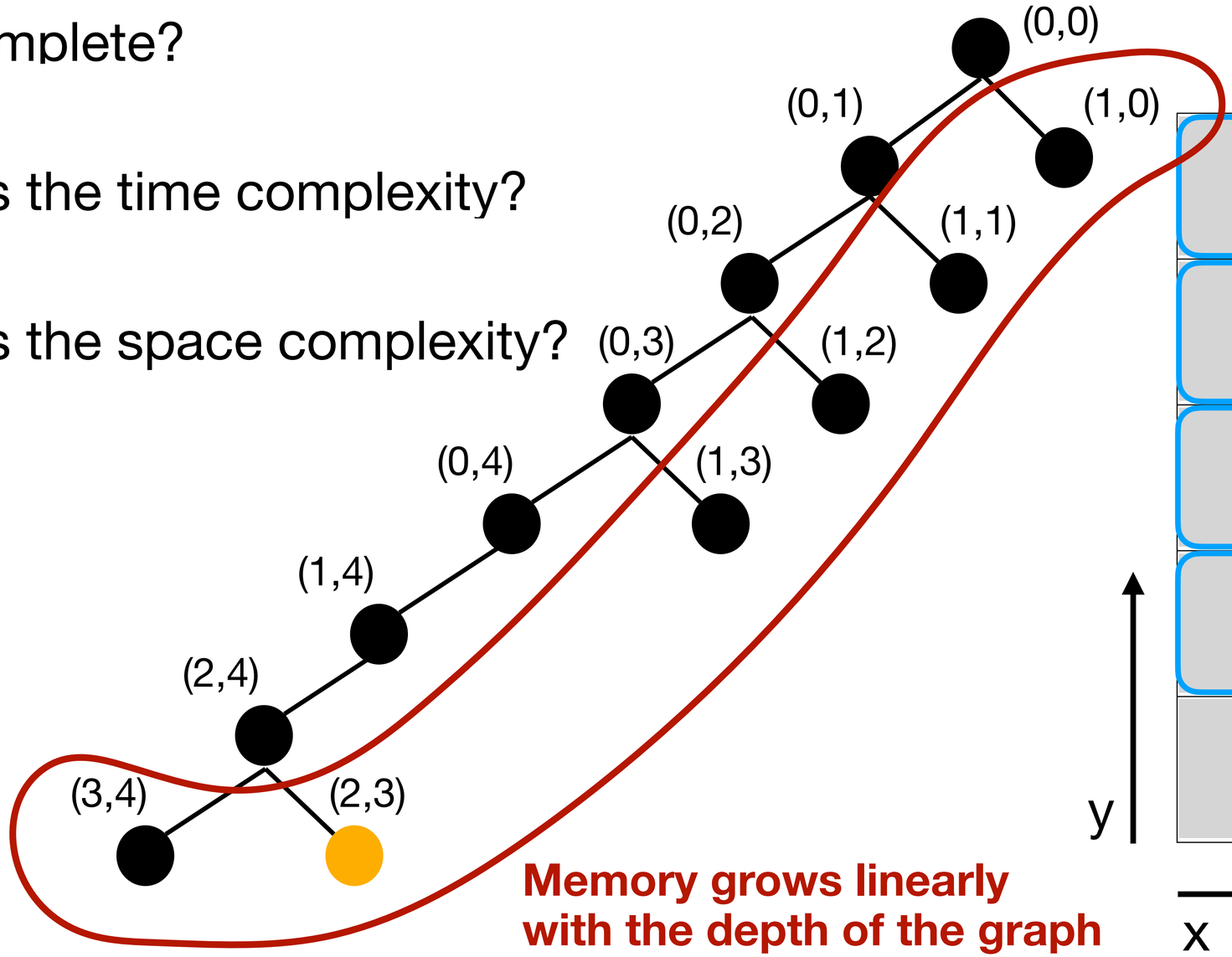
```
n = state(init)
frontier.append(n)
while(frontier not empty)
  n = pull state from frontier
  append n to visited
  if n = goal, return solution
  for all actions in n
    n' = a(n)
    if n' not visited
      append n' to frontier
```



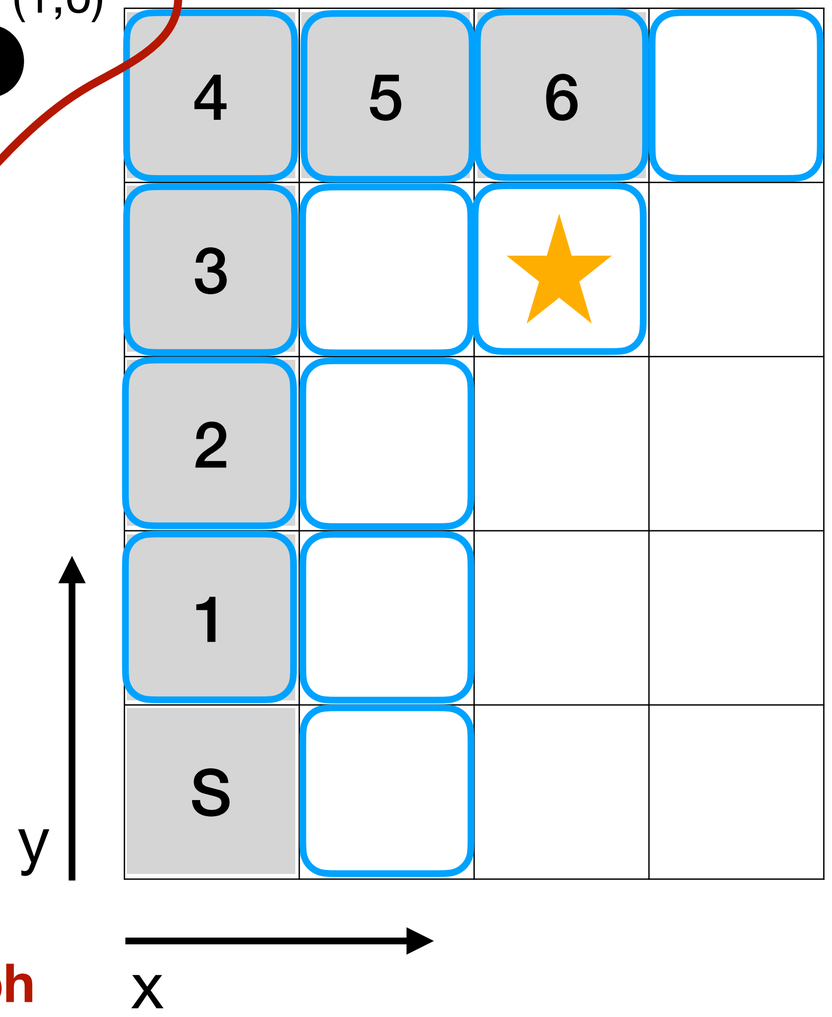


Depth First Search

- Is it complete?
- What is the time complexity?
- What is the space complexity?



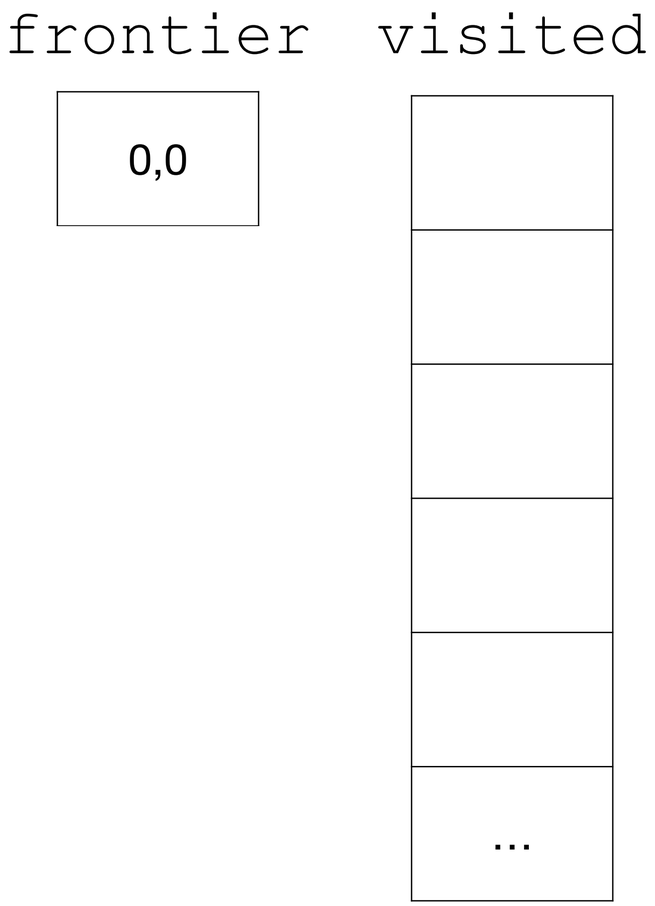
Search Order: N, E, S, W

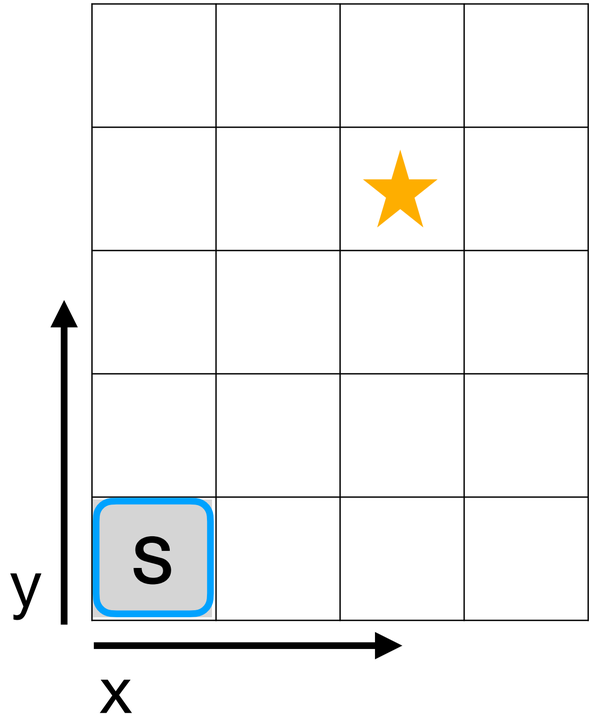


Memory grows linearly with the depth of the graph

Breadth First Search (BFS)

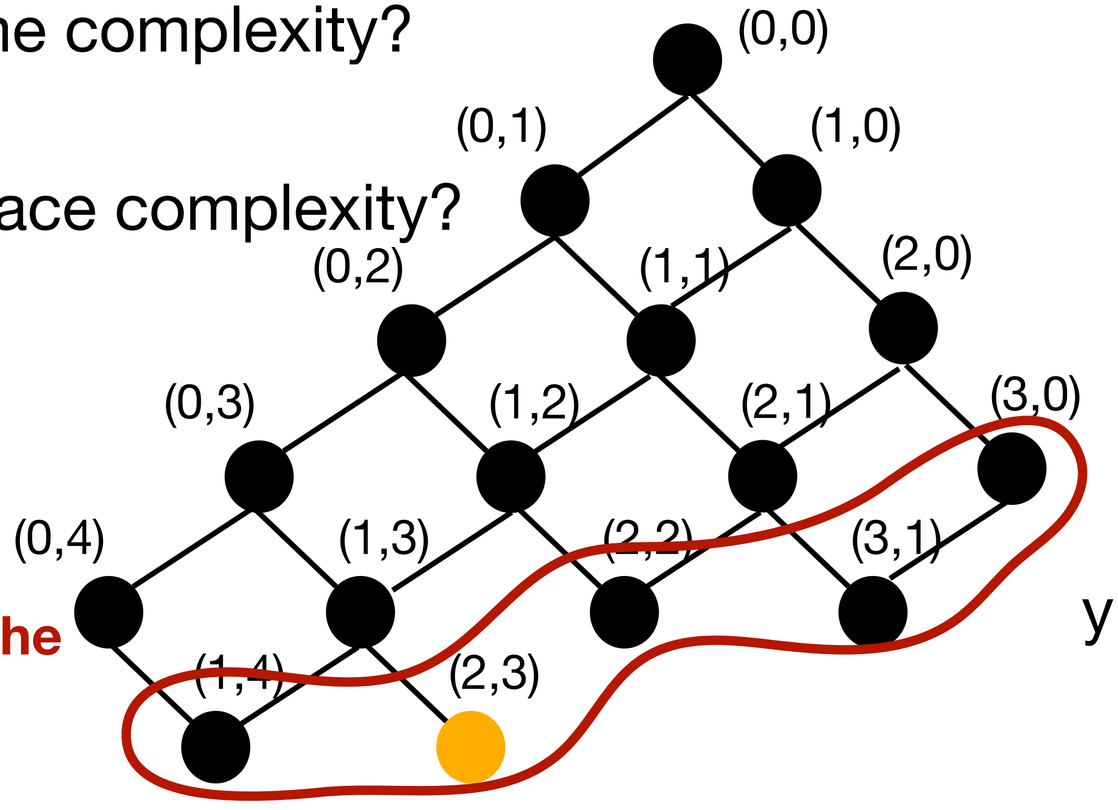
```
n = state(init)
frontier.append(n)
while(frontier not empty)
  n = pull state from frontier
  append n to visited
  if n = goal, return solution
  for all actions in n
    n' = a(n)
    if n' not visited
      append n' to frontier
```





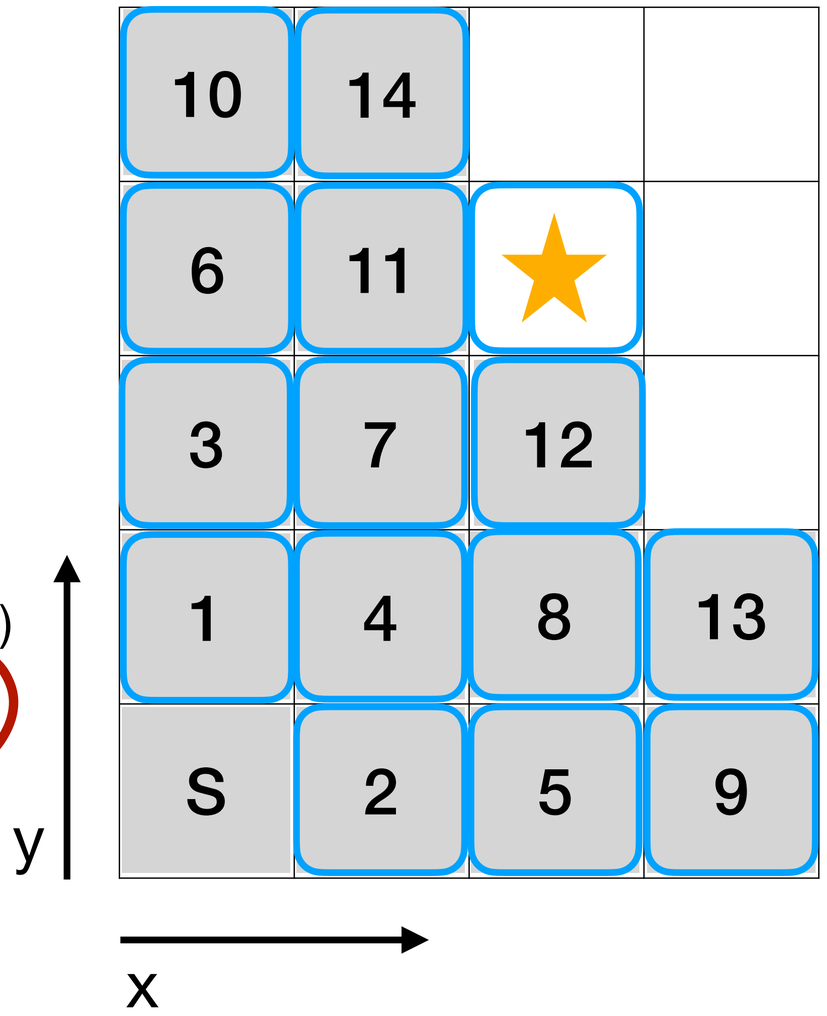
Breadth First Search

- Is it complete?
- Is it optimal?
- What is the time complexity?
- What is the space complexity?



Memory grows exponentially with the depth of the graph

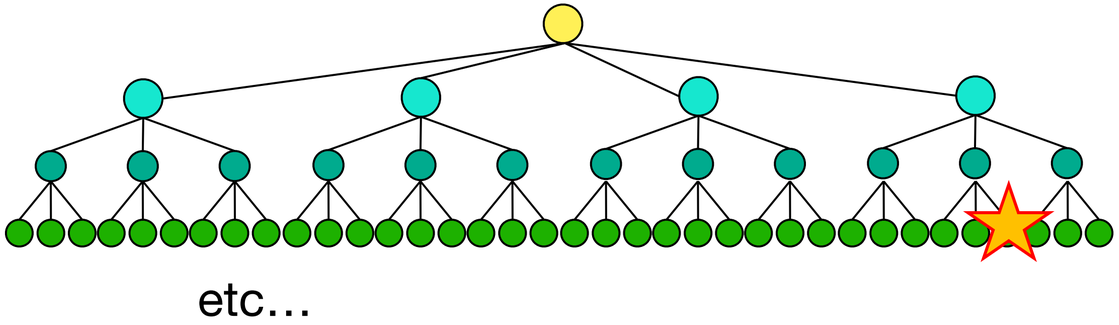
Search Order: N, E, S, W

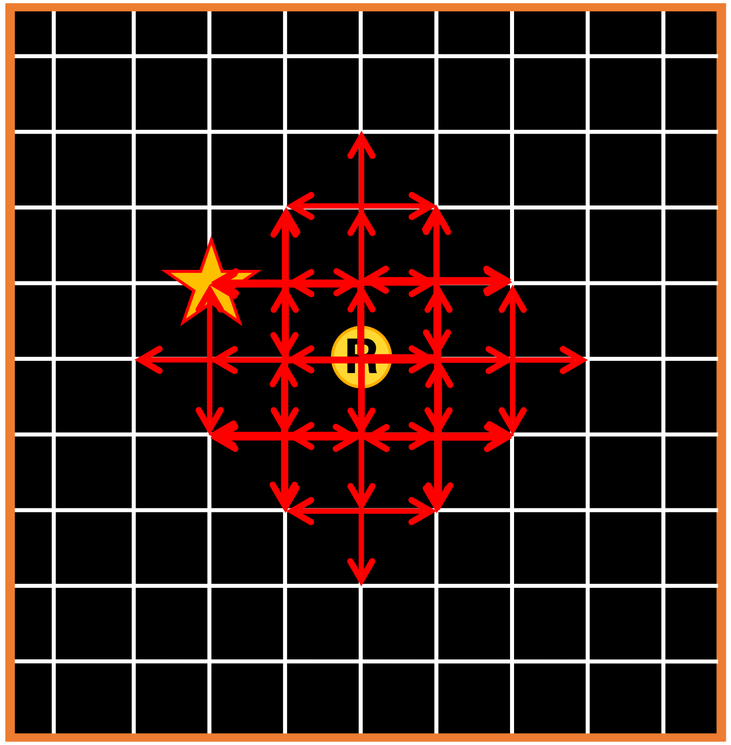


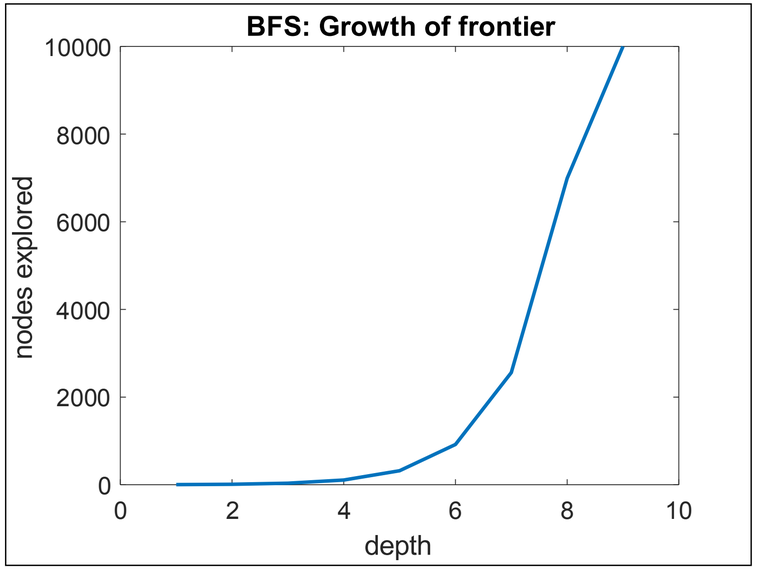
BFS: Memory and Computation

Frontier size:

- 4
- 12
- 36





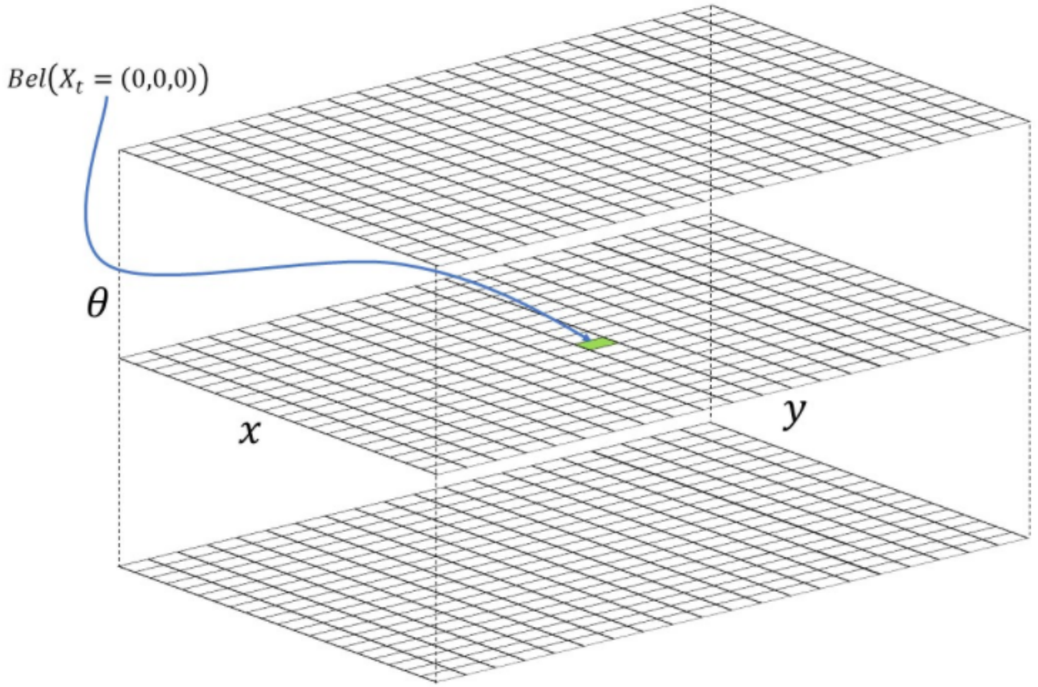


Uninformed Search Algorithms

- When is DFS appropriate?
- When is DFS inappropriate?
- When is BFS appropriate?
- When is BFS inappropriate?

Applications in Fast Robots

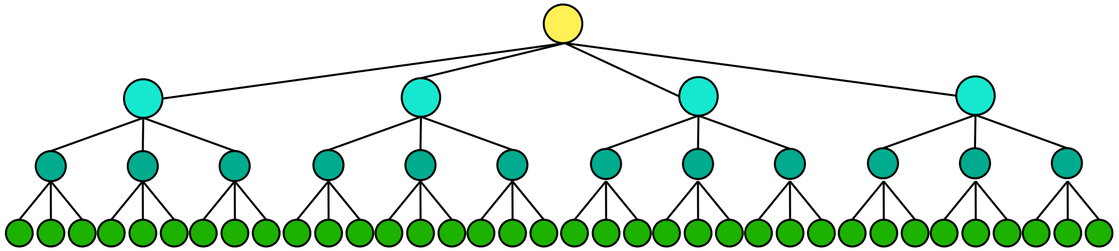
Is BFS/DFS possible on the Artemis?

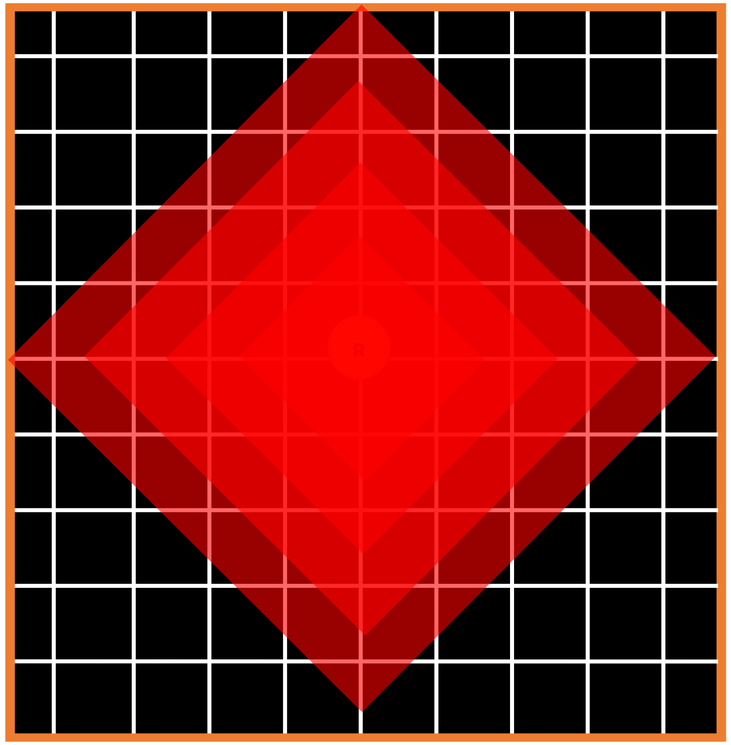


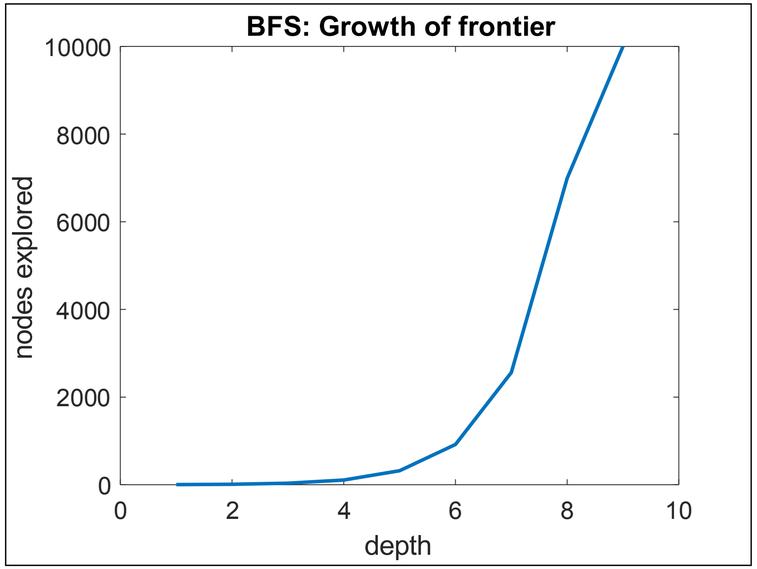
BFS: Memory and Computation

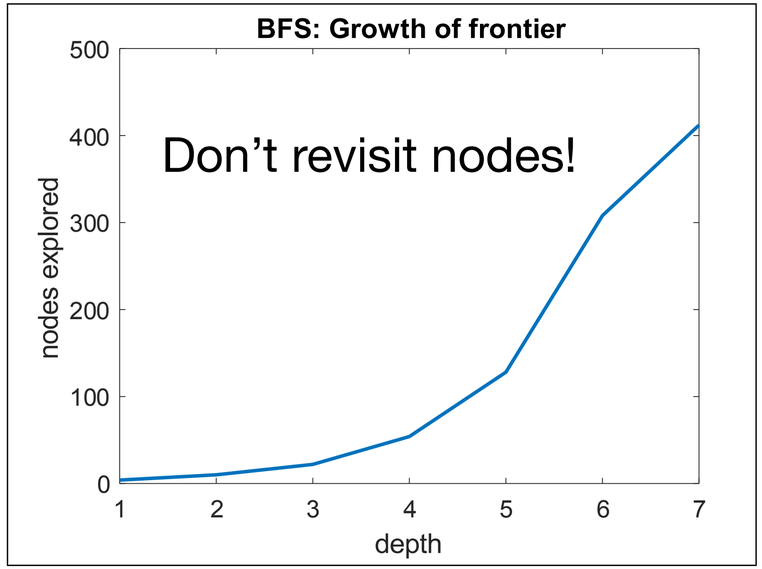
Frontier size:

- 4
- 12
- 36



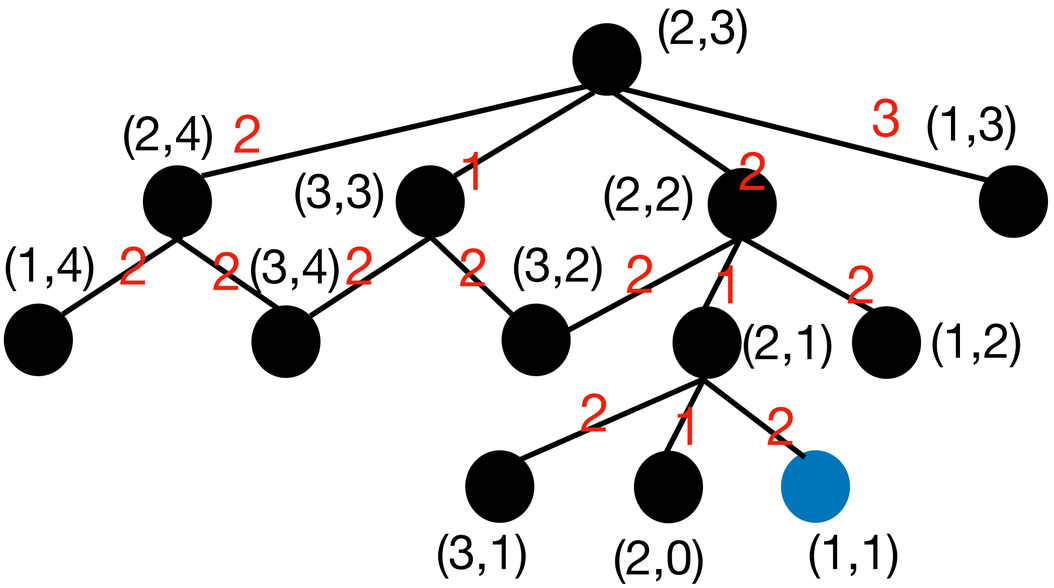






Lowest-Cost First Search (LCFS)

Consider parent cost!



What cost heuristic could we add?

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

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

Data structure

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

What node to expand next?

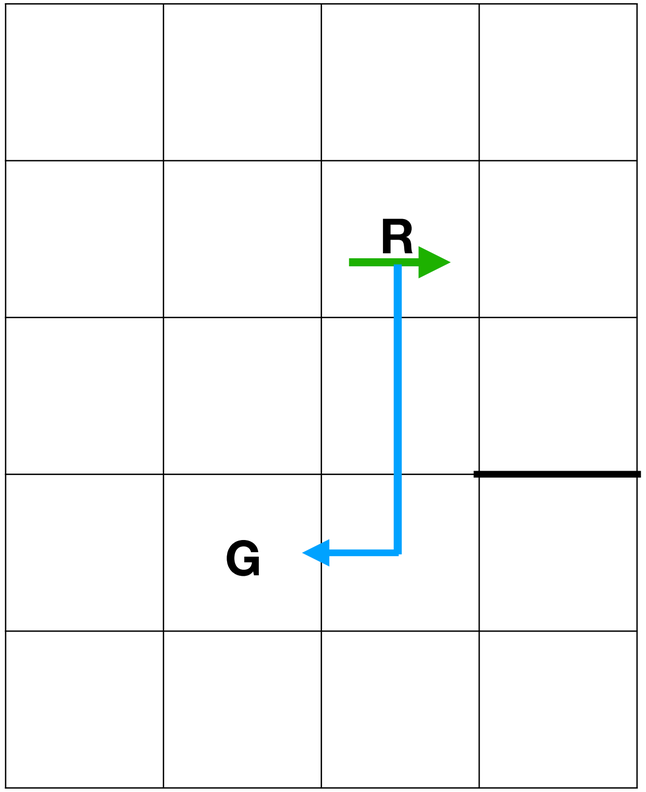
Lowest-Cost First Search (LCFS)

Consider parent cost!

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

Data structure

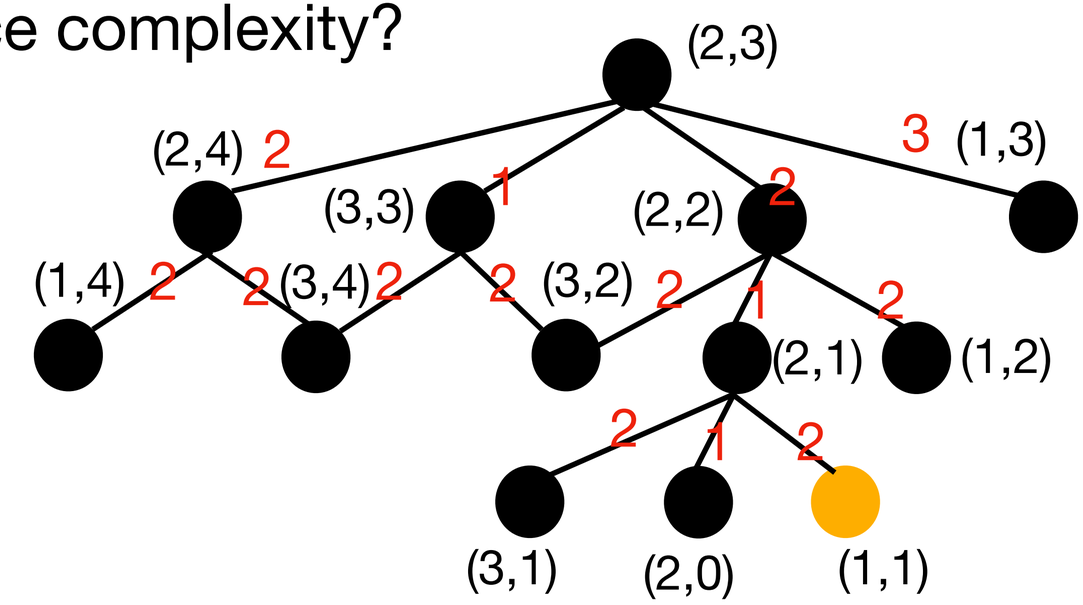
- n.state
- n.parent
- n.cost
- n.action



Lowest-Cost First Search (LCFS)

Consider parent cost!

- Is it complete?
- What is the time complexity?
- What is the space complexity?



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

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

Uninformed Search Algorithms

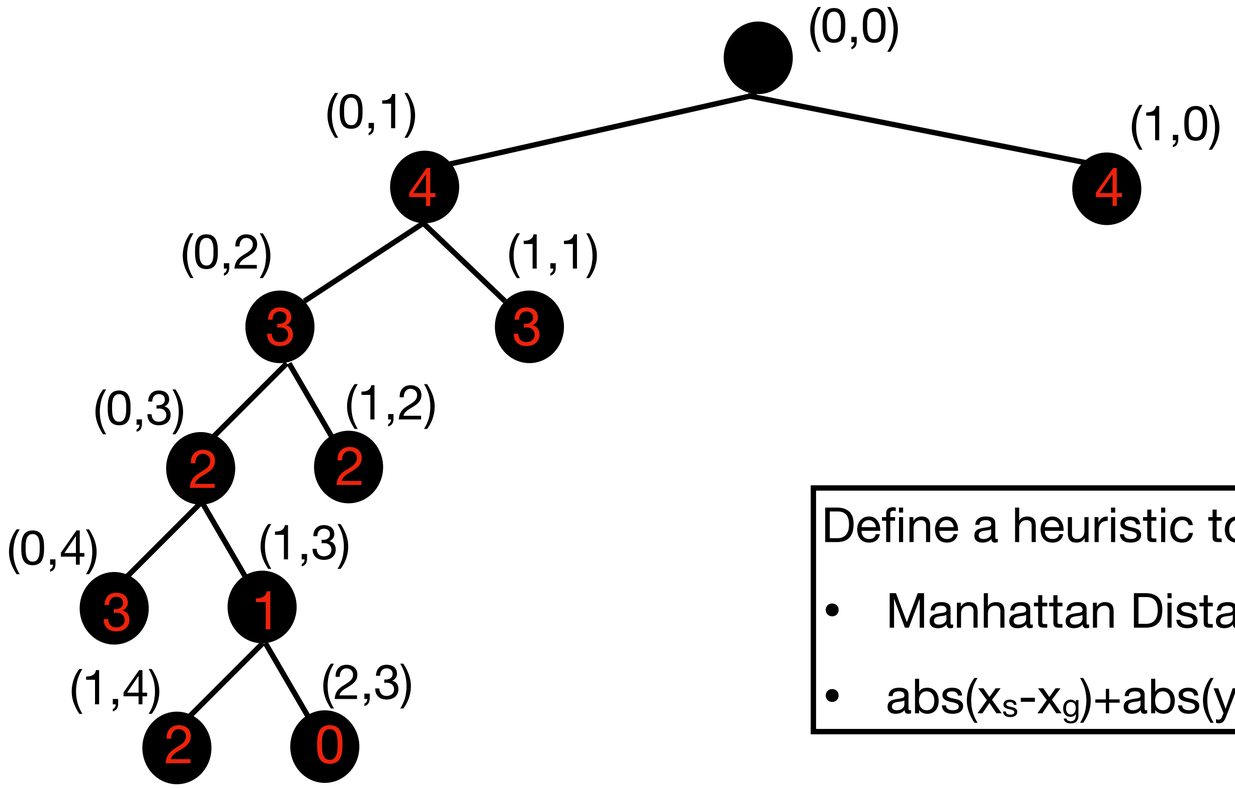
Criterion	BFS	DFS	LCFS
Complete	Yes (finite)	No (finite)	Yes (positive cost)
Time	$O(b^m)$	$O(b^m)$	$O(b^{1+C/c})$
Space	$O(b^m)$	$O(bm)$	$O(b^{1+C/c})$
Optimal	Yes (identical cost)	No	Yes
When to use	• Memory is a nonissue • Shallow solutions • Minimal branching factors • Shortest path needed	• Memory is restricted • Deep solutions	• Care about cost over length of path

Could we be smarter?

- Sure, you know the graph and you know the goal!
- Informed search
  - Consider the parent cost, and…
  - Estimate the shortest path to the “goal”
- Assign a value to the frontier
  - Pick a frontier closest to the goal (minimize distance)

Informed Search

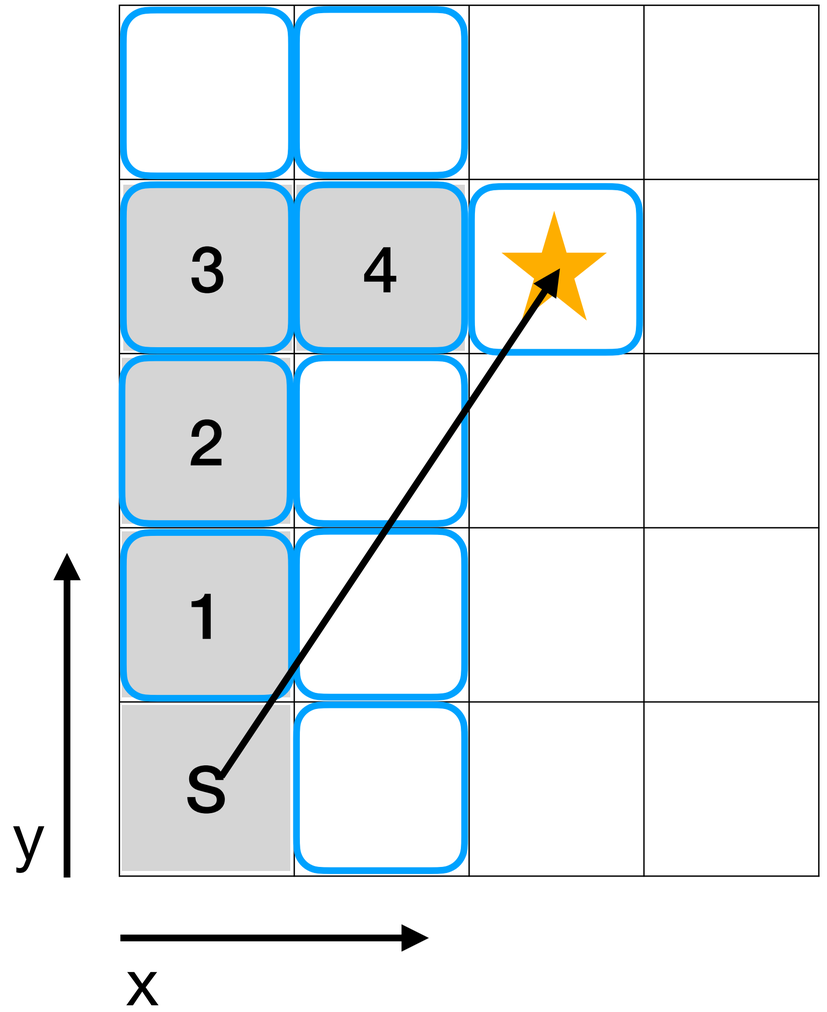
Greedy Search



Define a heuristic to target:

- Manhattan Distance
- $abs(x_s-x_g)+abs(y_s-y_g)$

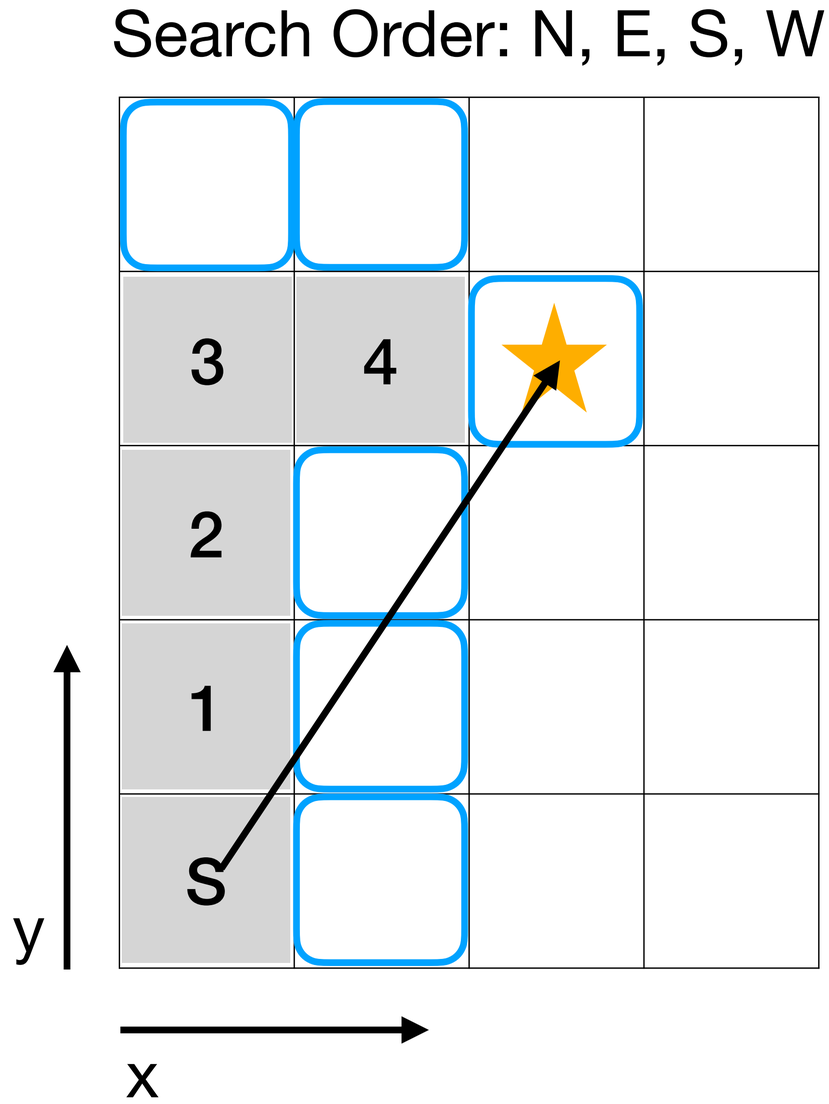
Search Order: N, E, S, W



Informed Search

Greedy Search

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

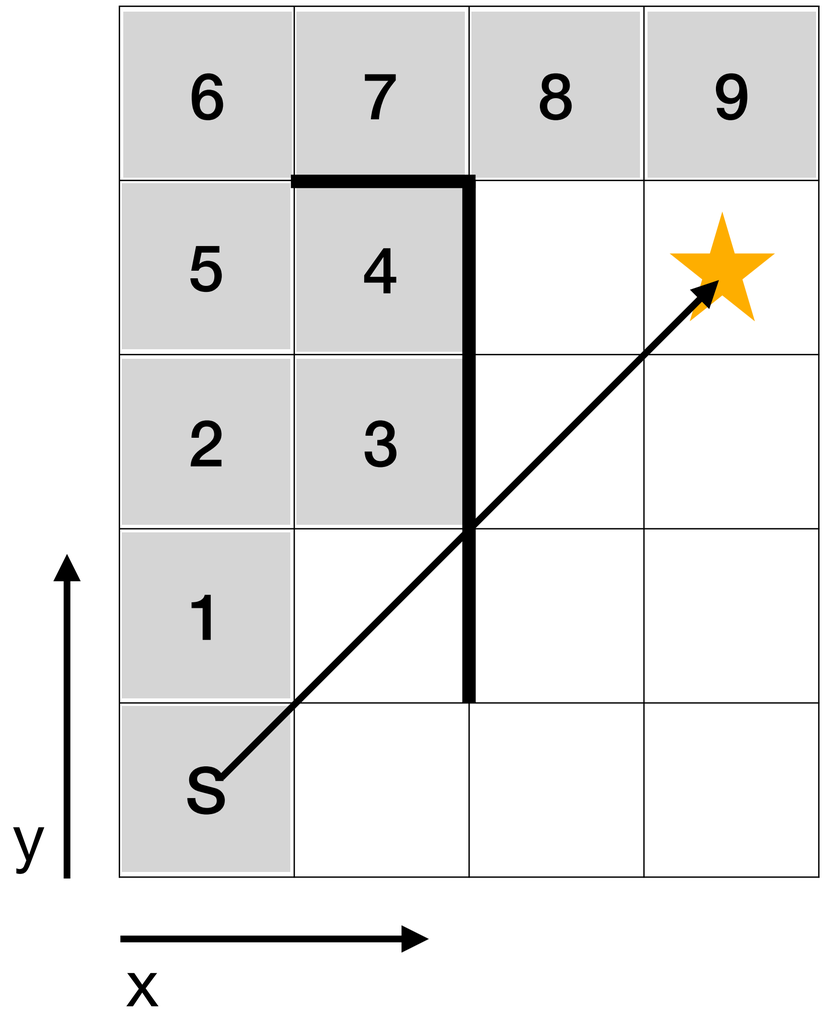


Informed Search

Greedy Search

- Is it complete?
- What is the time complexity?
- What is the space complexity?
- Optimal?

Search Order: N, E, S, W



Search Algorithms, general

- Breadth First Search
  - Complete and optimal
  - …but searches everything

Can we do better? A*

- Lowest-Cost First Algorithm
  - Complete and optimal
  - … but it wastes time exploring in directions that aren't promising

Considers parent cost

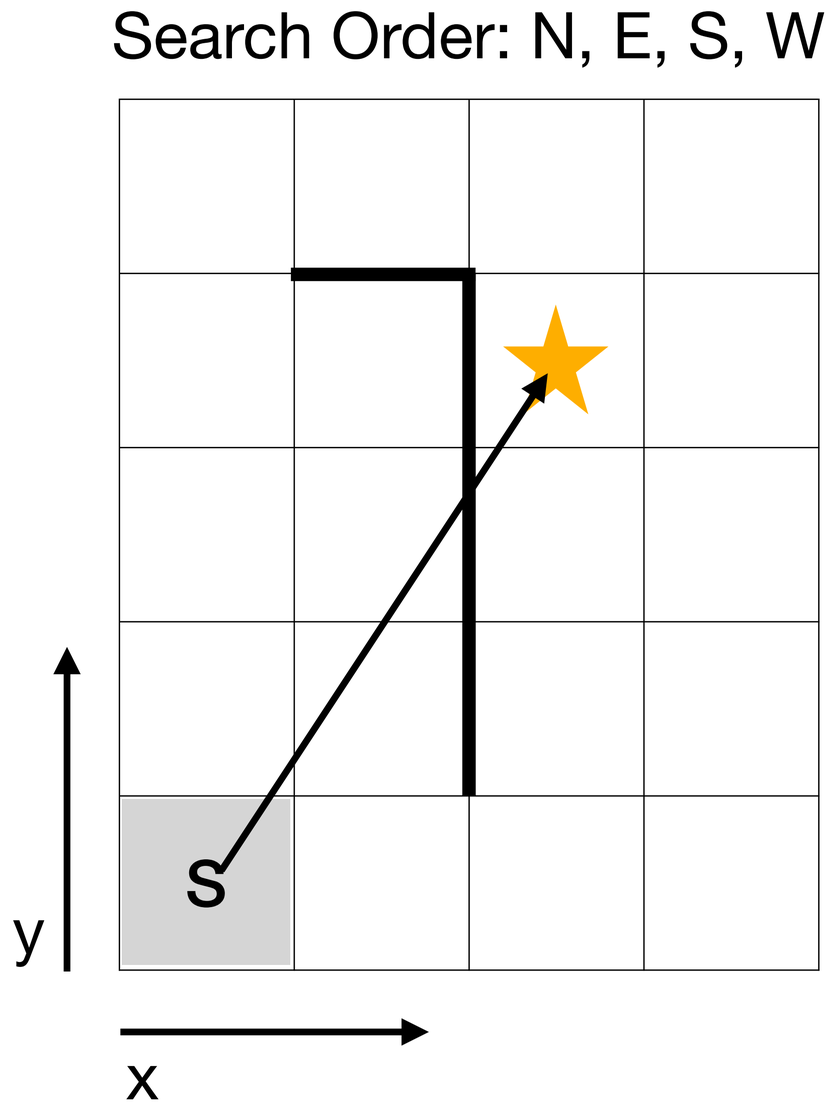
- Greedy Search
  - Complete (in most cases)
  - … only explores promising directions

Considers goal

Informed Search

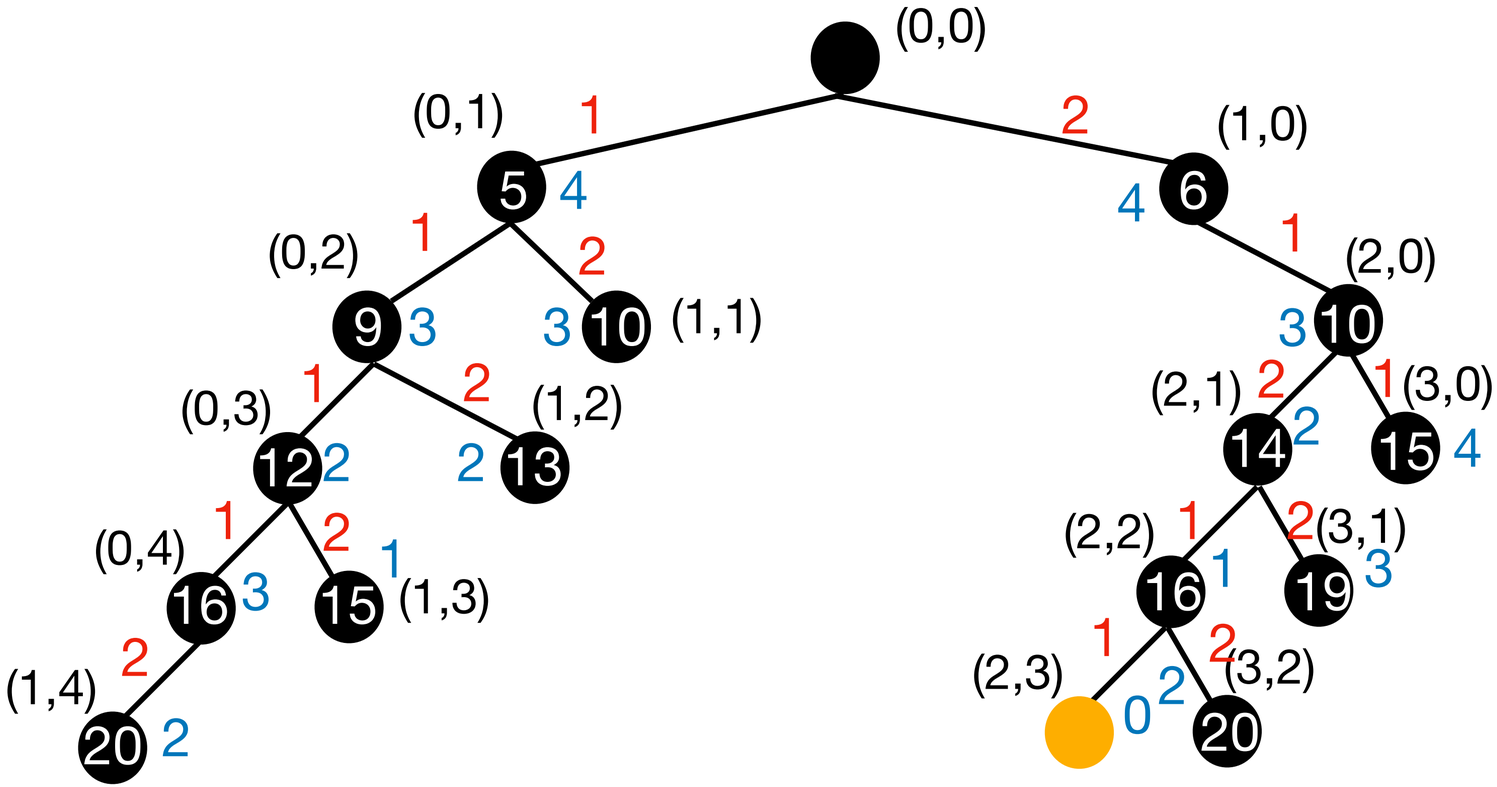
A* (A-star)

```
n = state(init)
frontier.append(n)
while(frontier not empty)
  n = pull state from frontier
  if n = goal, return solution
  for all actions in n
    n' = a(n)
    if (n' not visited)
        priority = heuristic(goal,n')+cost
        frontier.append(priority)
    if (visited and n'.cost < n_old.cost)
        visited.append(n')
```

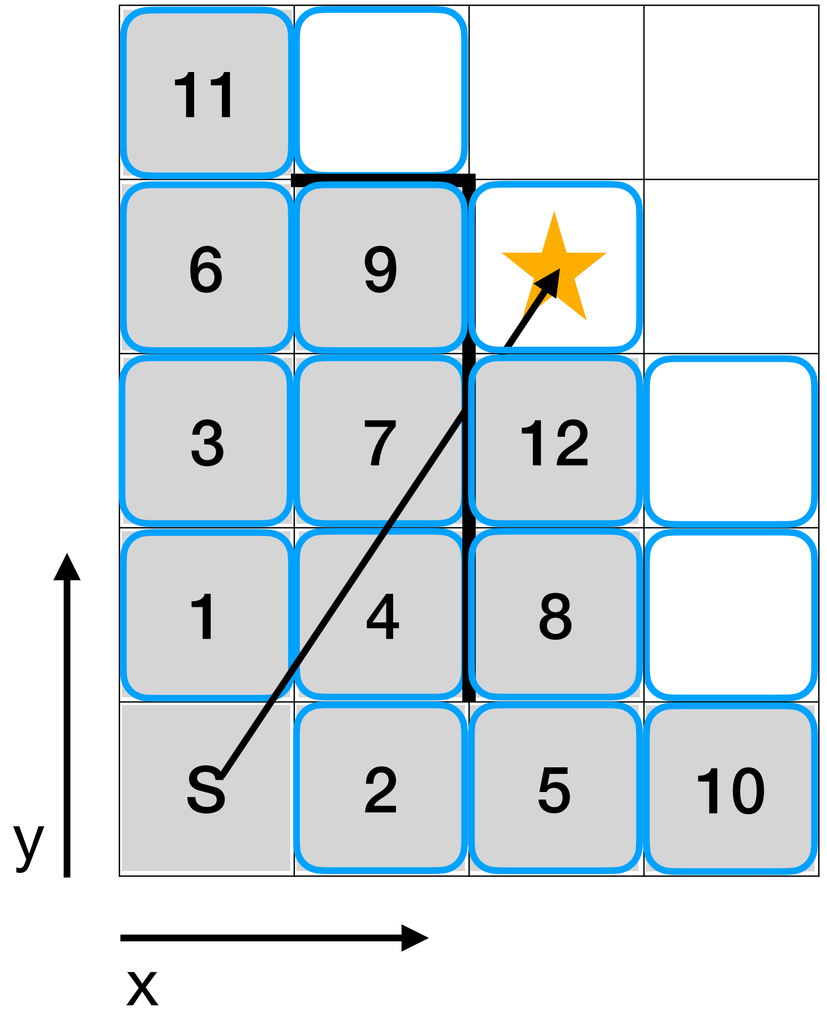


Informed Search

A* (A-star)



Search Order: N, E, S, W



A* Search

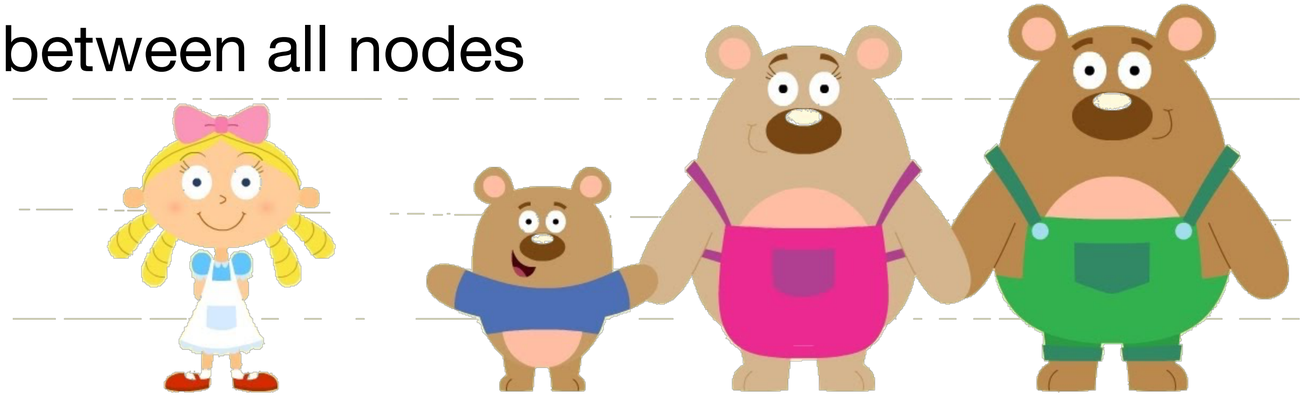
- What if the heuristic is too optimistic?
  - Estimated cost to goal < true cost to goal

Admissible heuristic

- What if the heuristic is too pessimistic?
  - Estimated cost to goal > true cost to goal
  - No longer guaranteed to be optimal

Inadmissible heuristic

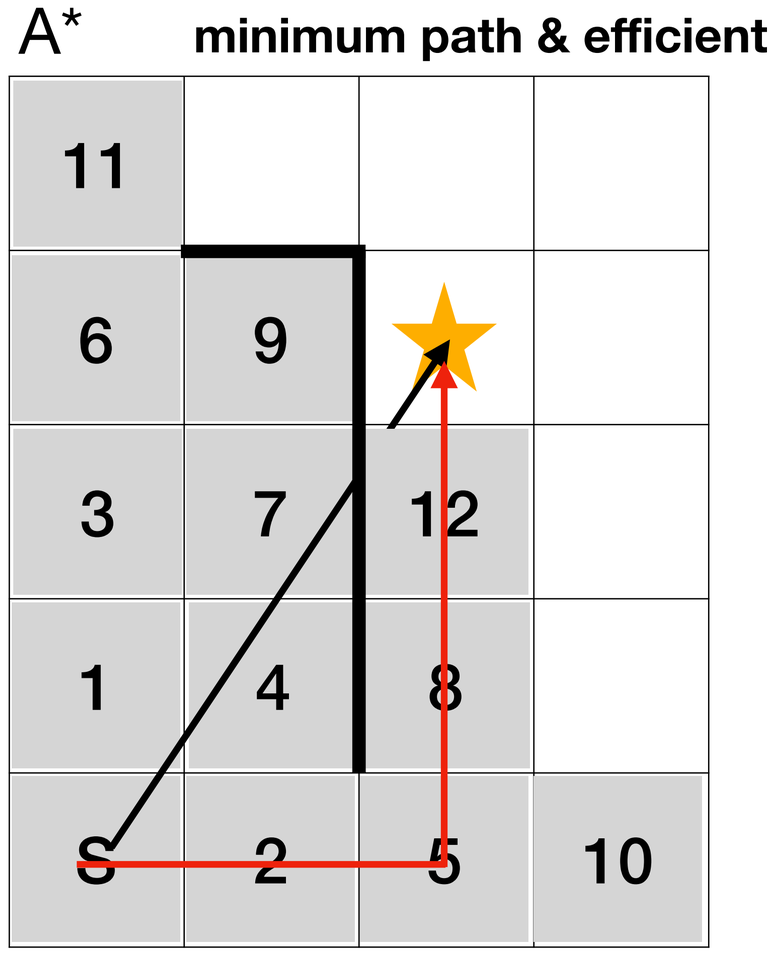
- What if the heuristic is just right?
  - Pre-compute the cost to the goal between all nodes
  - Feasible for you?



Informed Search

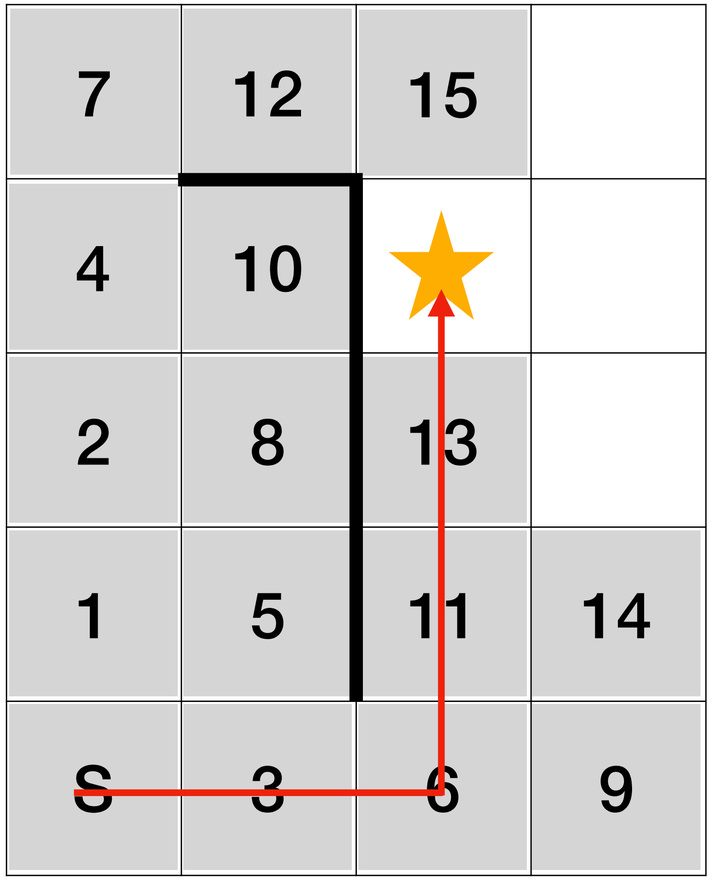
A* (A-star)

- Is it complete?
  - Yes!
- What is the time complexity?
  - $O(b^m)$
- What is the space complexity?
  - $O(b^m)$
- Optimal?
  - Yes, if the heuristic is admissible!

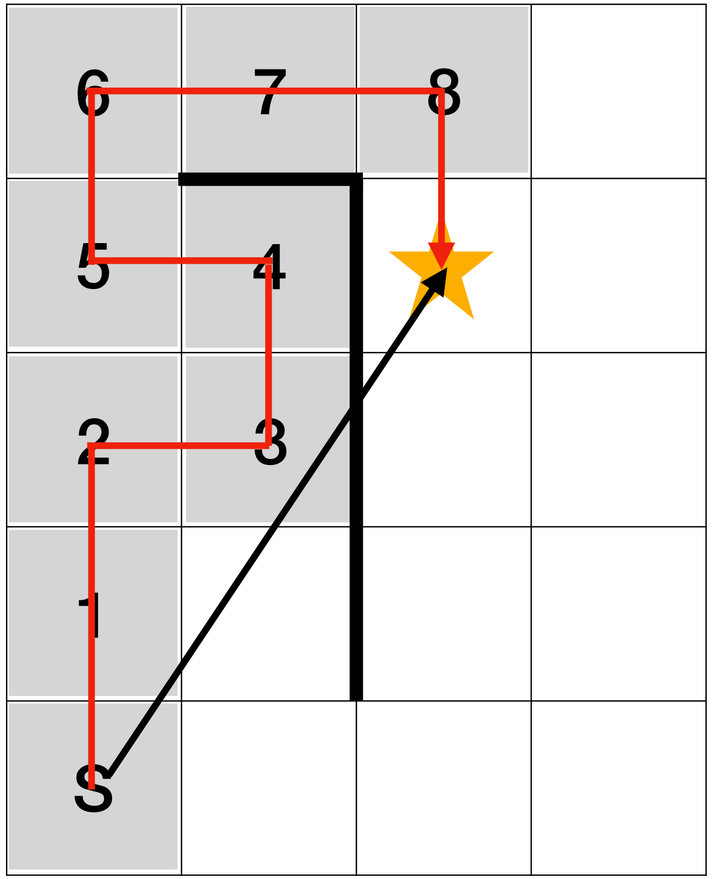


Summary

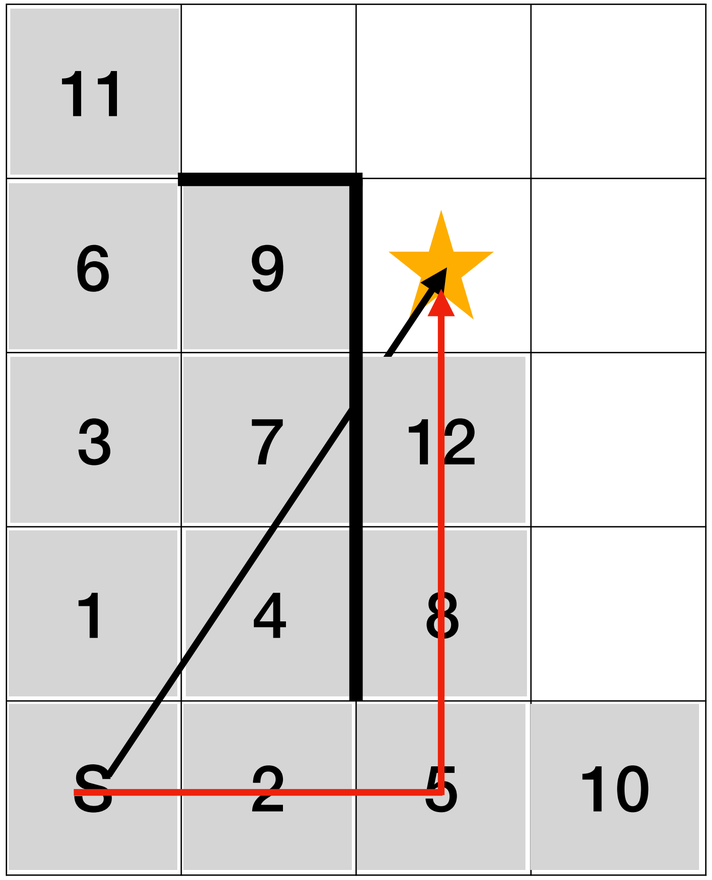
LCFS **minimum path**



Greedy



A* **minimum path & efficient**



Next class… Markov and Bayes!