Graph Search

Fast Robots, ECE4160/5160, MAE 4190/5190

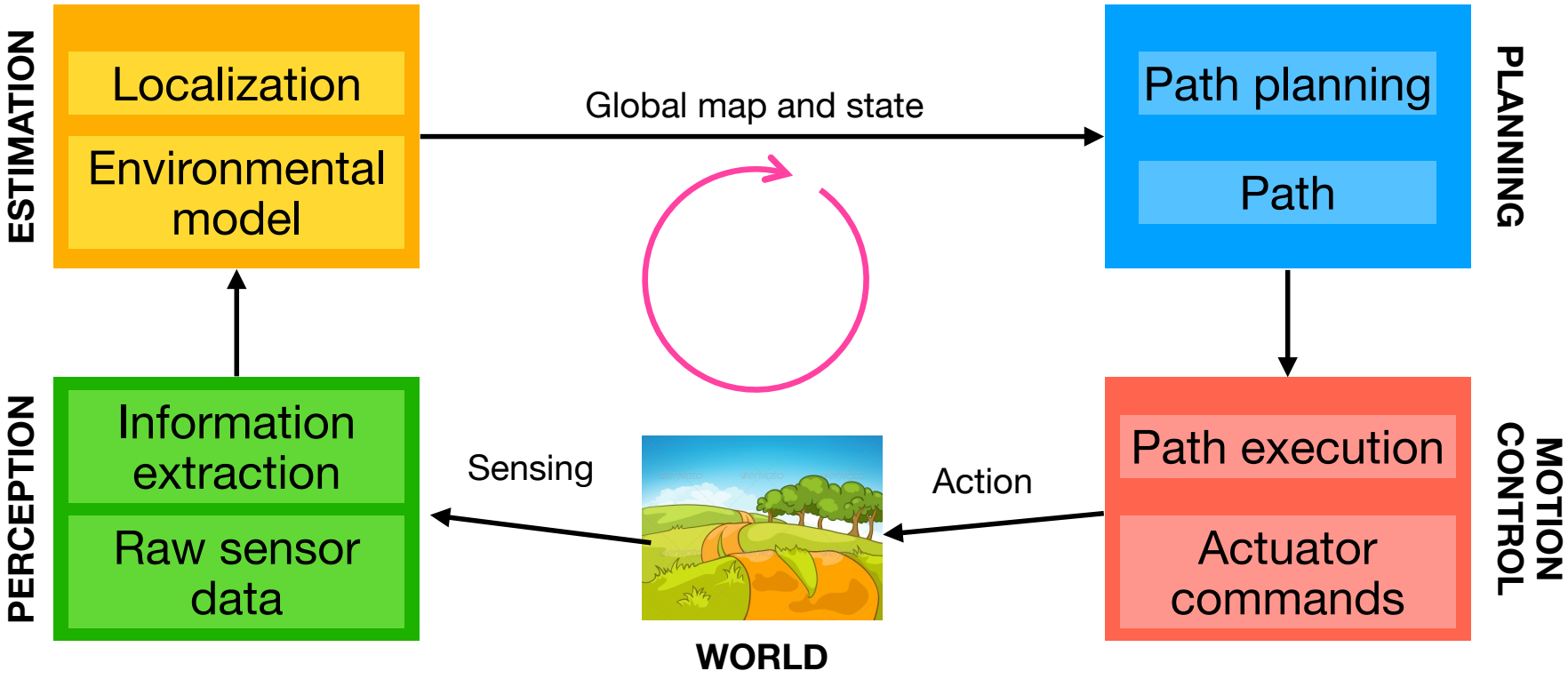
E. Farrell Helbling, 3/17/25

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!
- Lab 3 regrade requests will close on **Thursday midnight.** Submit requests in canvas.
- Lab 4 regrade requests will close next **Tuesday midnight.**

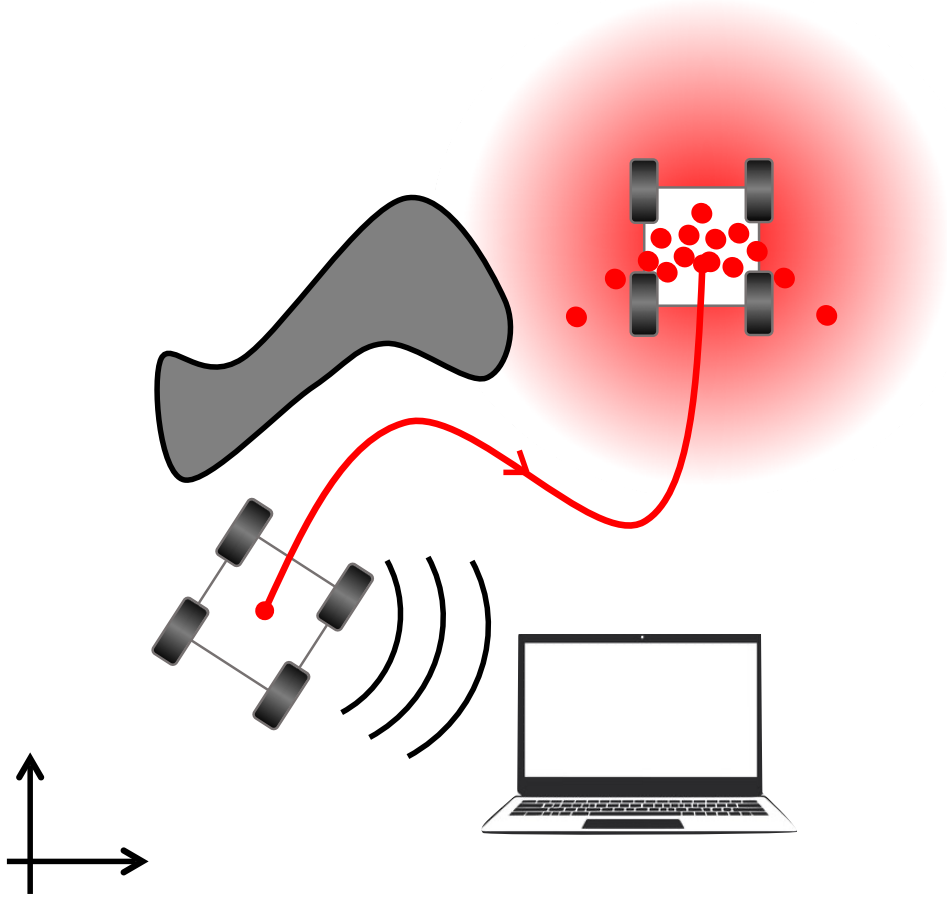
Navigation

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

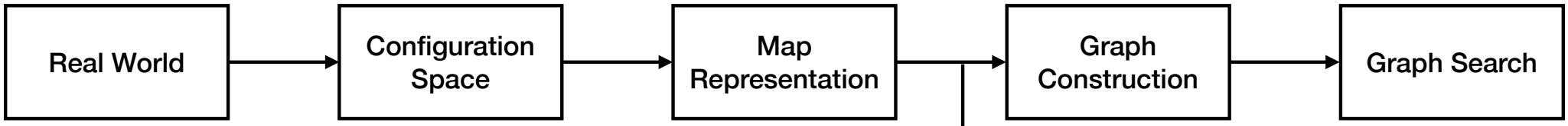


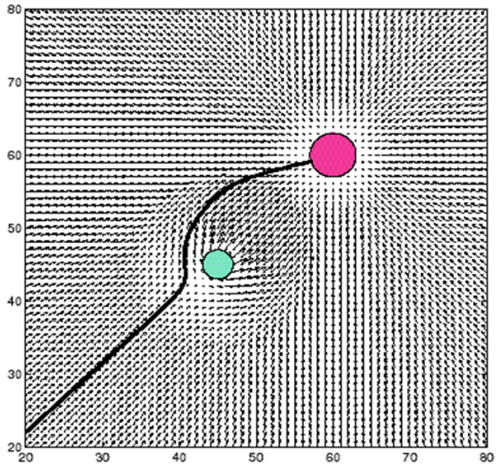
Navigation

- Local planners
- 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

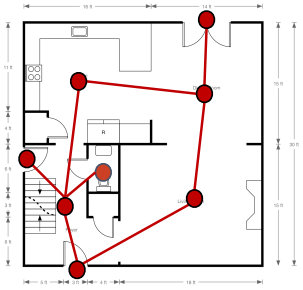


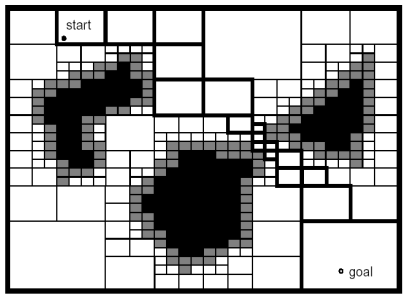


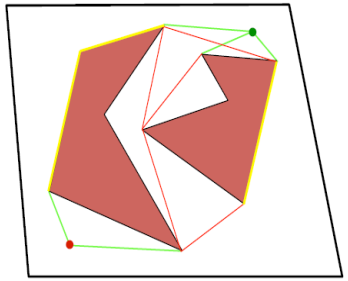
Common alternatives

- Optimal control
- Potential fields

Modeling path planning as a graph search problem







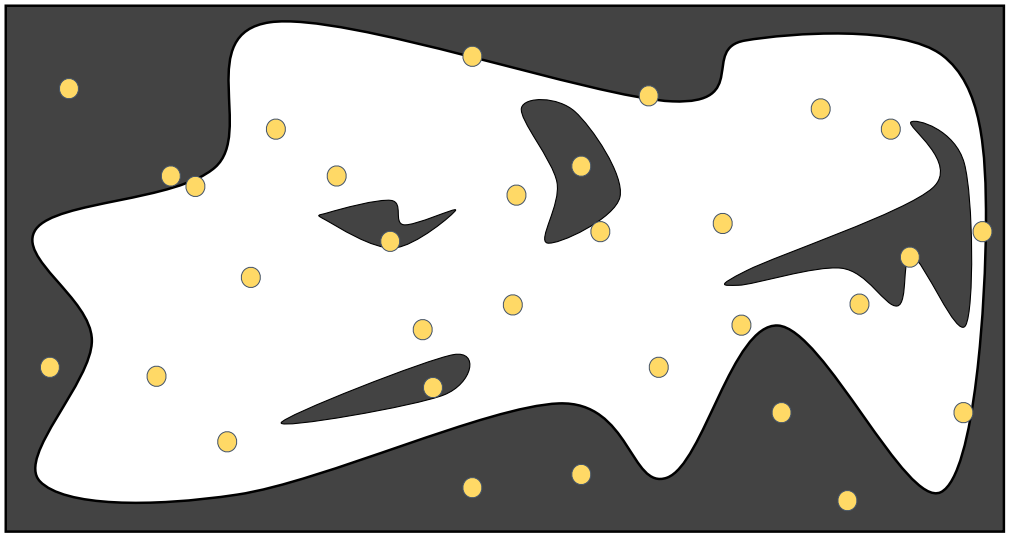
- Geometry-based graphs
  - Topological Graphs
  - Cell decomposition
  - Visibility Graphs
- Sampling-based graphs
  - RRT
  - PRM

Sampling-based planners

- Rather than computing the C-Space explicitly, we sample it
- Often efficient in high dimensional spaces
- Compute if a robot configuration has collisions
  - Just requires forward kinematics
  - (Local path plans between configurations)
- Examples
  - Probabilistic Roadmaps (PRM)
  - Rapidly Exploring Random Trees (RRT)

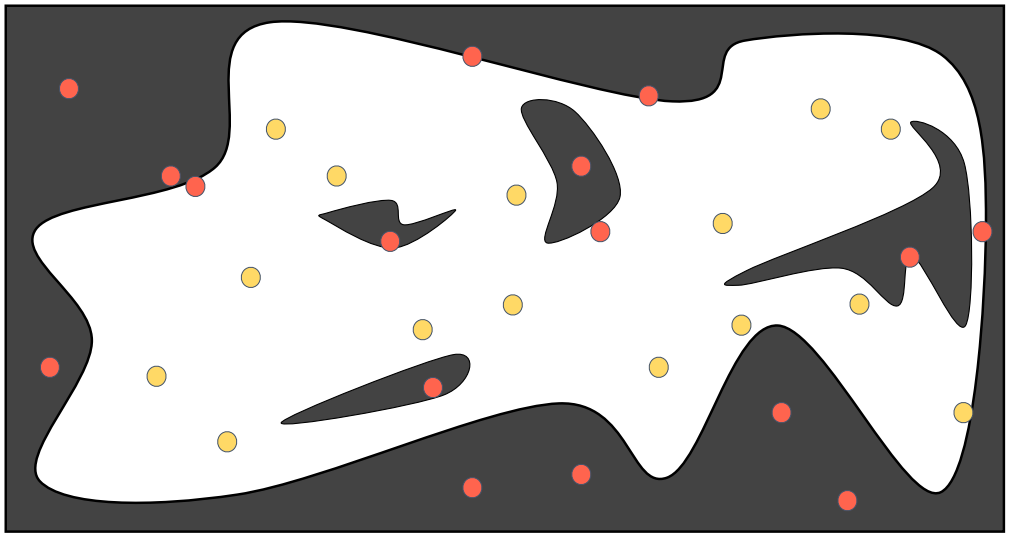
Probabilistic Roadmaps

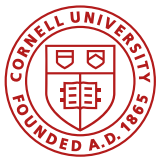
- Configurations are sampled by picking coordinates at random



Probabilistic Roadmaps

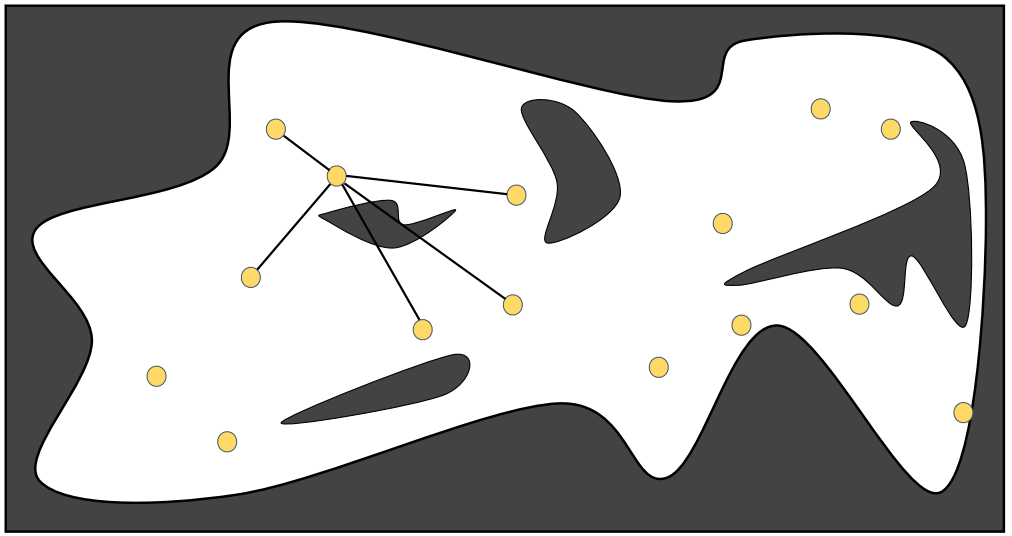
- Configurations are sampled by picking coordinates at random
- Sampled configurations are tested for collision

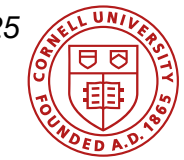




Probabilistic Roadmaps

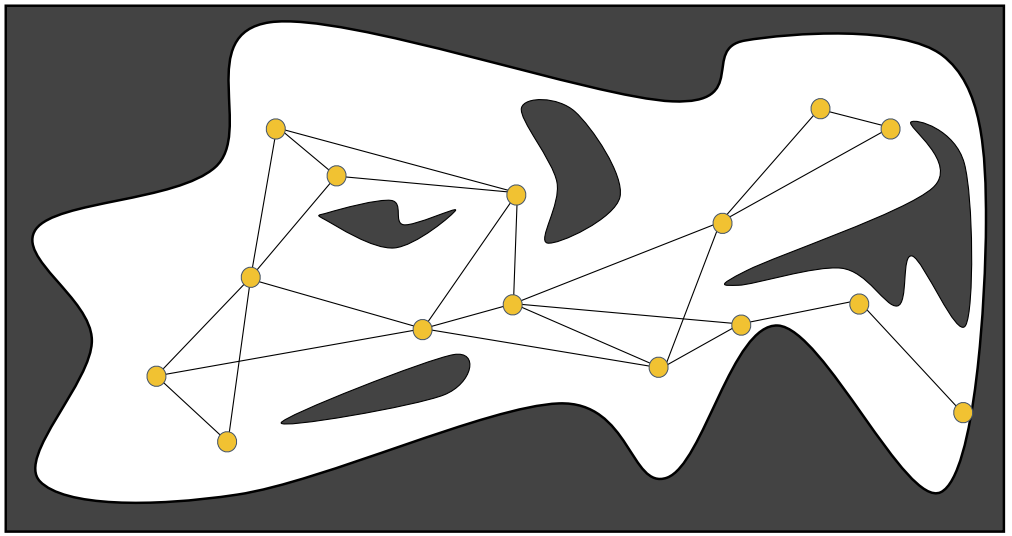
- Configurations are sampled by picking coordinates at random
- Sampled configurations are tested for collision
- Each configuration is linked by straight paths to its nearest neighbors





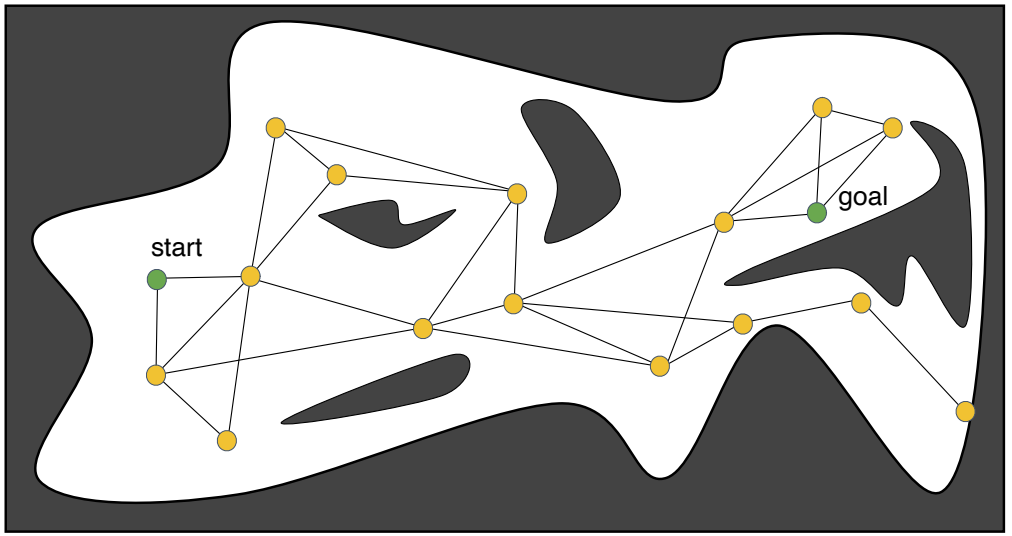
Probabilistic Roadmaps

- Configurations are sampled by picking coordinates at random
- Sampled configurations are tested for collision
- Each configuration is linked by straight paths to its nearest neighbors
- The collision-free links are retained as local paths to form the PRM



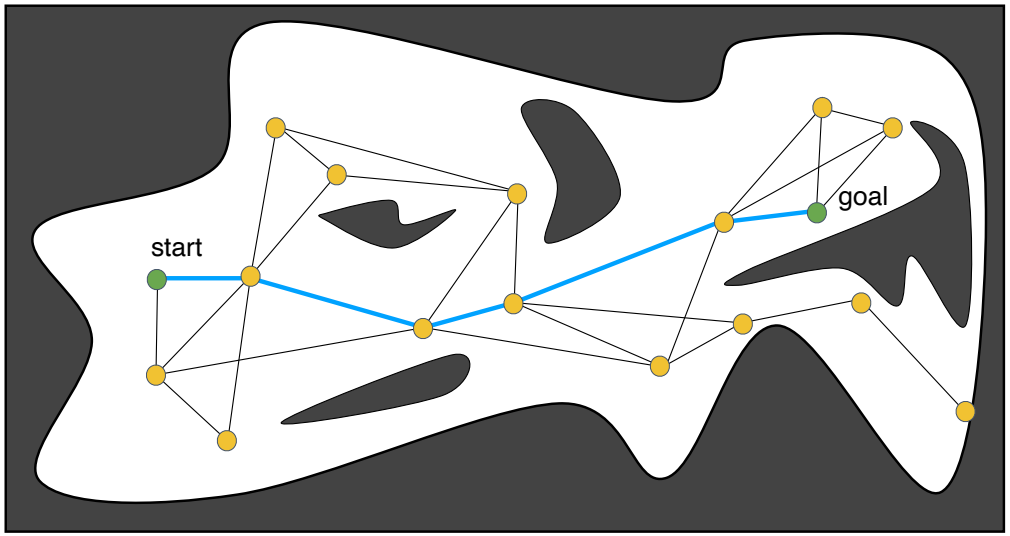
Probabilistic Roadmaps

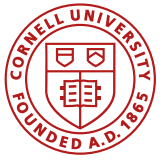
- Configurations are sampled by picking coordinates at random
- Sampled configurations are tested for collision
- Each configuration is linked by straight paths to its nearest neighbors
- The collision-free links are retained as local paths to form the PRM
- The start and goal configurations are included as milestones



Probabilistic Roadmaps

- Configurations are sampled by picking coordinates at random
- Sampled configurations are tested for collision
- Each configuration is linked by straight paths to its nearest neighbors
- The collision-free links are retained as local paths to form the PRM
- The start and goal configurations are included as milestones
- The PRM is searched for a path from start to goal





Probabilistic Roadmaps

Constructing the graph

- Initially empty Graph G
- A configuration q is randomly chosen
- If $q \in Q_{free}$ then add to G
- Repeat until N vertices chosen
- For each q, select k closest neighbors
- Local planner, Δ, connects q to neighbor q'
- If connection is collision free, add edge (q, q')

Algorithm 6 Roadmap Construction Algorithm

Input:

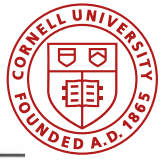
n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each configuration

Output:

A roadmap $G = (V, E)$

```
1:  V ← ∅
2:  E ← ∅
3:  while |V| < n do
4:     repeat
5:        q ← a random configuration in Q
6:     until q is collision-free
7:     V ← V ∪ {q}
8:  end while
9:  for all q ∈ V do
10:    N_q ← the k closest neighbors of q chosen from V according to dist
11:    for all q' ∈ N_q do
12:       if (q, q') ∉ E and Δ(q, q') ≠ NIL then
13:          E ← E ∪ {(q, q')}
14:       end if
15:    end for
16: end for
```



Probabilistic Roadmaps

Finding the Path

- Connect q_{init} and q_{goal} to the roadmap
- Find k closest neighbors of q_{init} and q_{goal} in roadmap, plan local path Δ
- Compute cost of path
- Repeat until graphs are connected
- Choose cheapest path

Algorithm 7 Solve Query Algorithm

Input:

q_{init}: the initial configuration
q_{goal}: the goal configuration
k: the number of closest neighbors to examine for each configuration
$G = (V, E)$: the roadmap computed by algorithm 6

Output:

A path from q_{init} to q_{goal} or failure

1: $N_{q_{\text{init}}} \leftarrow$ the k closest neighbors of q_{init} from V according to *dist*
2: $N_{q_{\text{goal}}} \leftarrow$ the k closest neighbors of q_{goal} from V according to *dist*
3: $V \leftarrow \{q_{\text{init}}\} \cup \{q_{\text{goal}}\} \cup V$
4: set q' to be the closest neighbor of q_{init} in $N_{q_{\text{init}}}$
5: **repeat**
6: **if** $\Delta(q_{\text{init}}, q') \neq$ `NIL` **then**
7: $E \leftarrow (q_{\text{init}}, q') \cup E$
8: **else**
9: set q' to be the next closest neighbor of q_{init} in $N_{q_{\text{init}}}$
10: **end if**
11: **until** a connection was succesful or the set $N_{q_{\text{init}}}$ is empty
12: set q' to be the closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
13: **repeat**
14: **if** $\Delta(q_{\text{goal}}, q') \neq$ `NIL` **then**
15: $E \leftarrow (q_{\text{goal}}, q') \cup E$
16: **else**
17: set q' to be the next closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
18: **end if**
19: **until** a connection was succesful or the set $N_{q_{\text{goal}}}$ is empty
20: $P \leftarrow$ shortest path$(q_{\text{init}}, q_{\text{goal}}, G)$
21: **if** P is not empty **then**
22: **return** P
23: **else**
24: **return** `failure`
25: **end if**

Probabilistic Roadmaps

Finding the Path

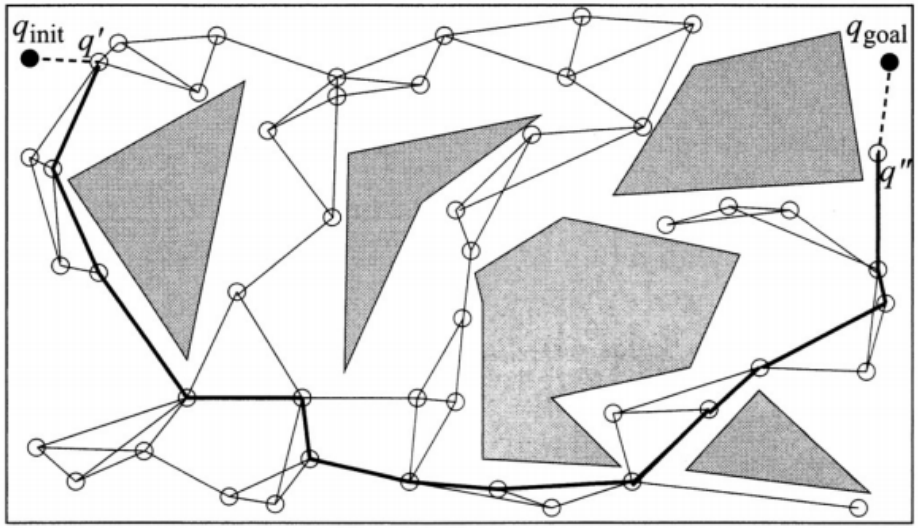


Figure 7.4 An example of how to solve a query with the roadmap from figure 7.3. The configurations q_{init} and q_{goal} are first connected to the roadmap through q' and q''. Then a graph-search algorithm returns the shortest path denoted by the thick black lines.

Probabilistic Roadmaps

Considerations

- Single query/ multi query
- How are nodes placed?
  - Uniform sampling strategies
  - Non-uniform sampling strategies
- How are local neighbors found?
- How is collision detection performed?
  - Dominates time consumption in PRMs

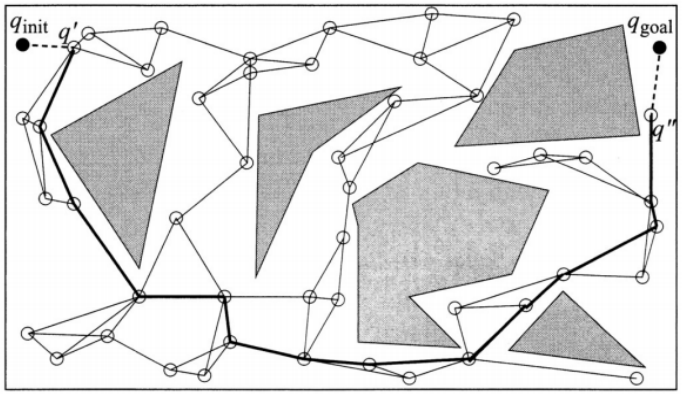


Figure 7.4 An example of how to solve a query with the roadmap from figure 7.3. The configurations q_{init} and q_{goal} are first connected to the roadmap through q' and q''. Then a graph-search algorithm returns the shortest path denoted by the thick black lines.

Probabilistic Roadmaps

Robot Motion Planning on a Chip, Murray et al. RSS 2016

- PRM on an FPGA
- Collision detection circuits on each edge in logic gates for massive parallel operation
- 6DOF planning in <1ms

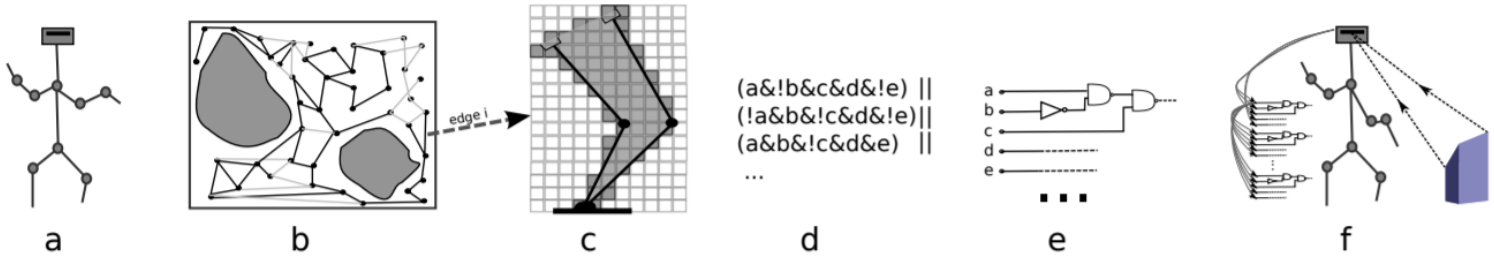
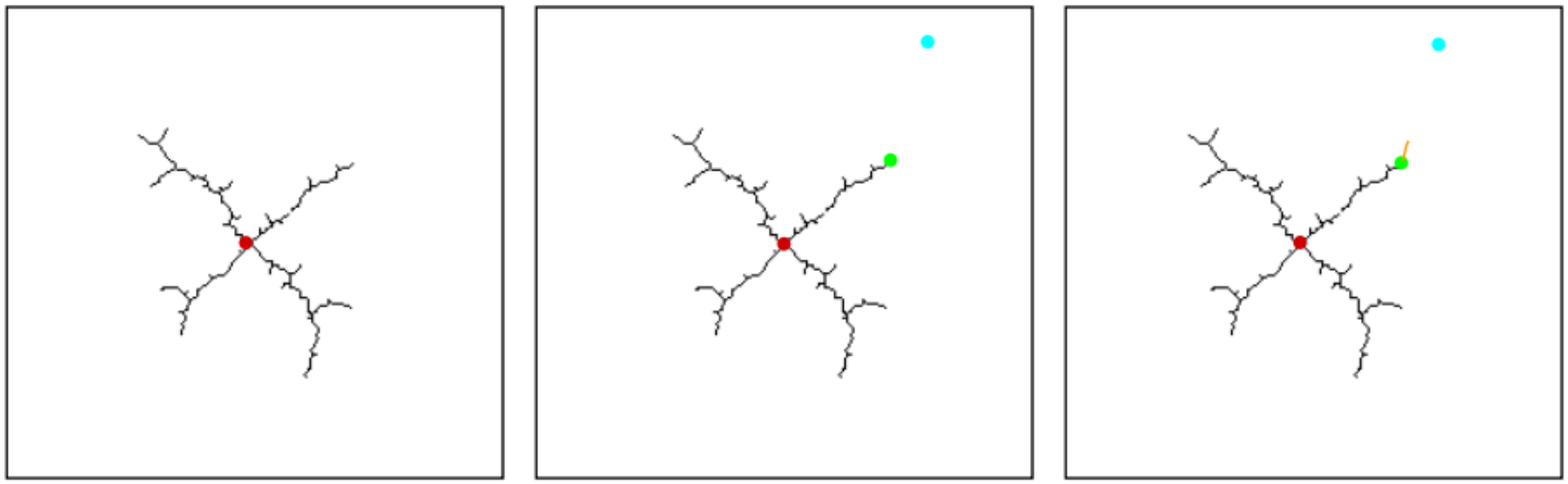


Fig. 3: Our process for producing robot-specific motion planning circuitry. Given a robot description (a), we construct a PRM (b), most likely subsampled for coverage from a much larger PRM. We discretize the robot's reachable space into depth pixels and, for each edge i on the PRM, precompute all the depth pixels that collide with the corresponding swept volume (c). We use these values to construct a logical expression that, given the coordinates of a depth pixel encoded in binary, returns `true` if that depth pixel collides with edge i (d); this logical expression is optimized and used to build a collision detection circuit (CDC) (e). For each edge in the PRM there is one such circuit. When the robot wishes to construct a motion plan, it perceives its environment, determines which depth pixels correspond to obstacles, and transmits their binary representations to every CDC (f). All CDCs perform collision detection *simultaneously, in parallel* for each depth pixel, storing a bit which indicates

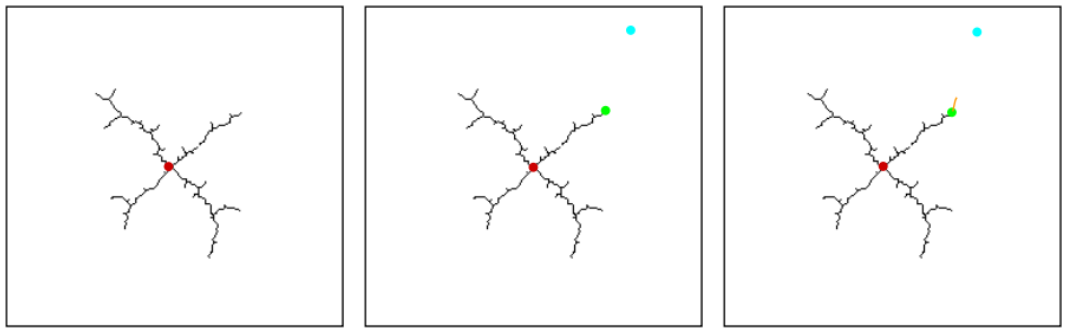
Rapidly Exploring Random Trees (RRT)



1. Maintain a tree rooted at the starting point
2. Choose a point at random from free space
3. Find the closest configuration already in the tree
4. Extend the tree in the direction of the new configuration

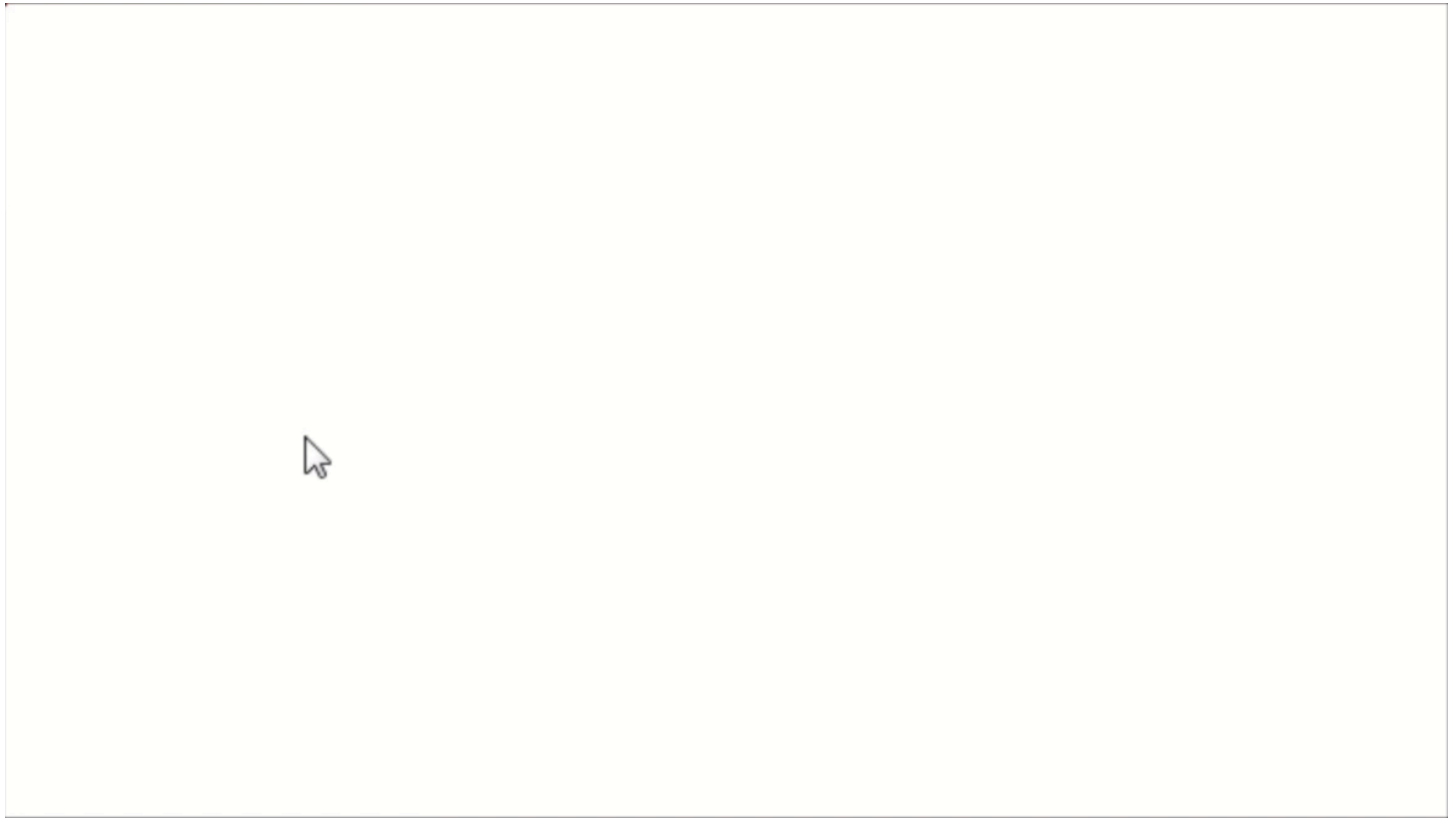
Rapidly Exploring Random Trees (RRT)

```
1.Algorithm BuildRRT
2.  Input: Initial configuration q_init, number of vertices K,
 incremental distance Δq)
3.  Output: RRT graph G
4.  G.init(q_init)
5.  for k = 1 to K
6.    qrand ← RAND_CONF()
7.    qnear ← NEAREST_VERTEX(qrand, G)
8.    qnew ← NEW_CONF(qnear, qrand, Δq)
9.    G.add_vertex(qnew)
10.   G.add_edge(qnear, qnew)
11. return G
```



Rapidly Exploring Random Trees (RRT)

Uniform/ biased sampling

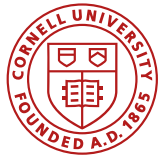


Aaron Becker, UH, Wolfram Player Example

Rapidly Exploring Random Trees (RRT)

Considerations

- Sensitive to step-size (Δq)
  - Small: many nodes, closely spaced, slowing down nearest neighbor computation
  - Large: Increased risk of suboptimal plans / not finding a solution
- How are samples chosen?
  - Uniform sampling may need too many samples to find the goal
  - Biased sampling towards goal can ease this problem
- How are closest neighbors found?
- How are local paths generated?
- Variations
  - RRT Connect, A*-RRT, Informed RRT*, Real-Time RRT*, Theta*-RRT, etc.



Graph Search

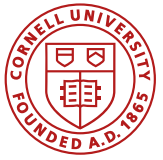
Modeling path planning as a graph search problem

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

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

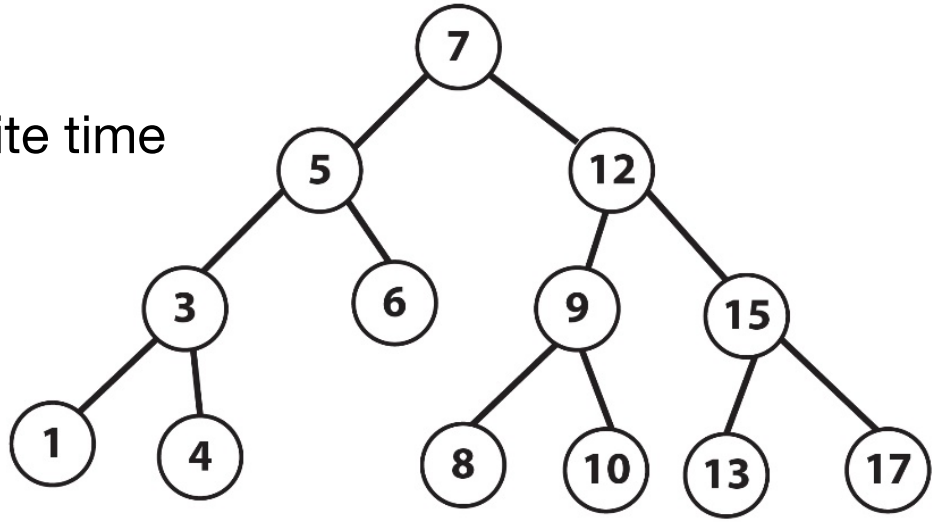
Graph Search

- 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



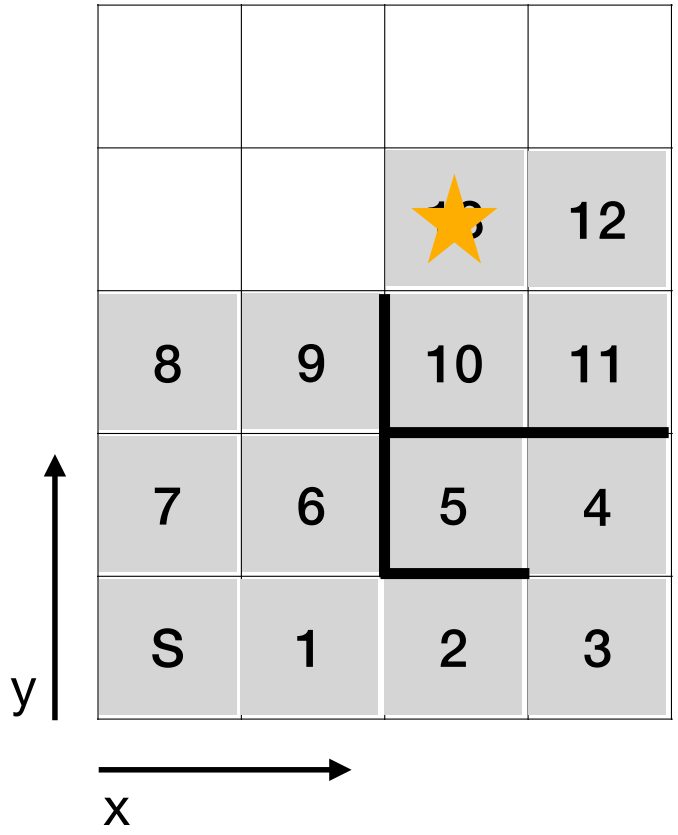
Algorithms and Search

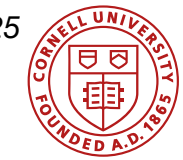
- What is the simplest thing to do?
  - Random or brute force search
  - How many grid traversals will brute force take?
    - First establish a search order: N, E, S, W

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

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.

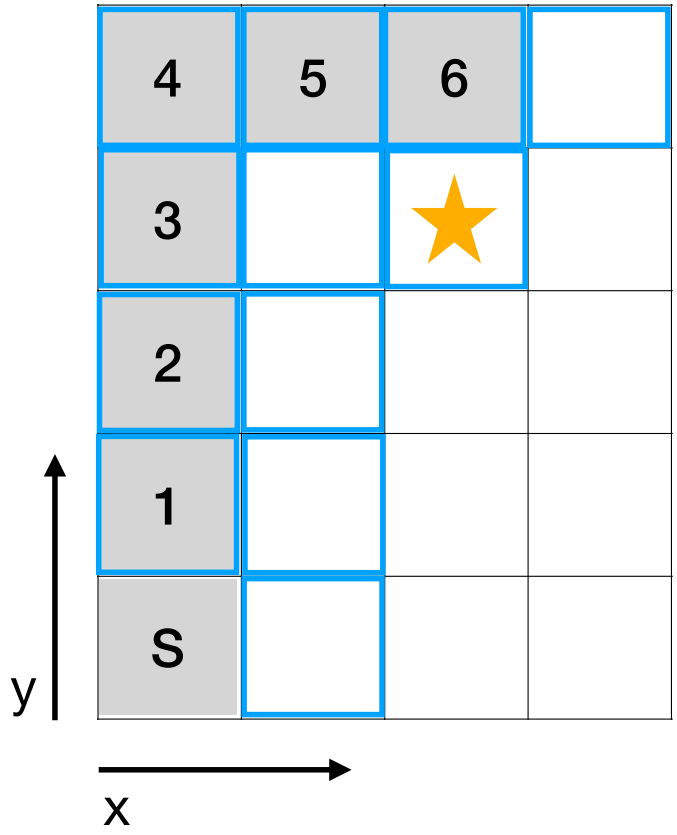


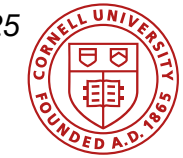


Algorithms and Search

- What is the simplest thing to do?
  - Random or brute force search
- Other methods?
  - Depth First Search (DFS)

Search Order: N, E, S, W

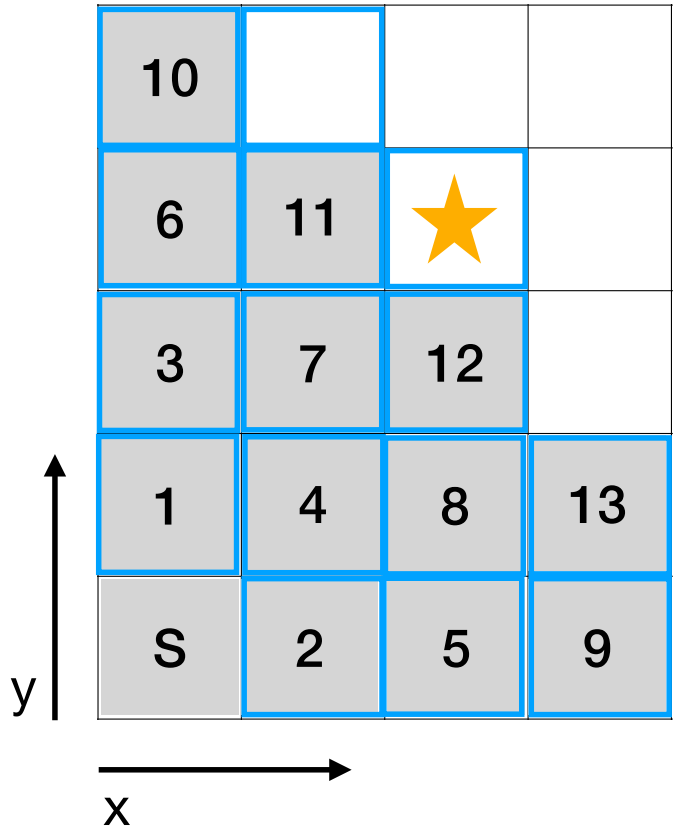




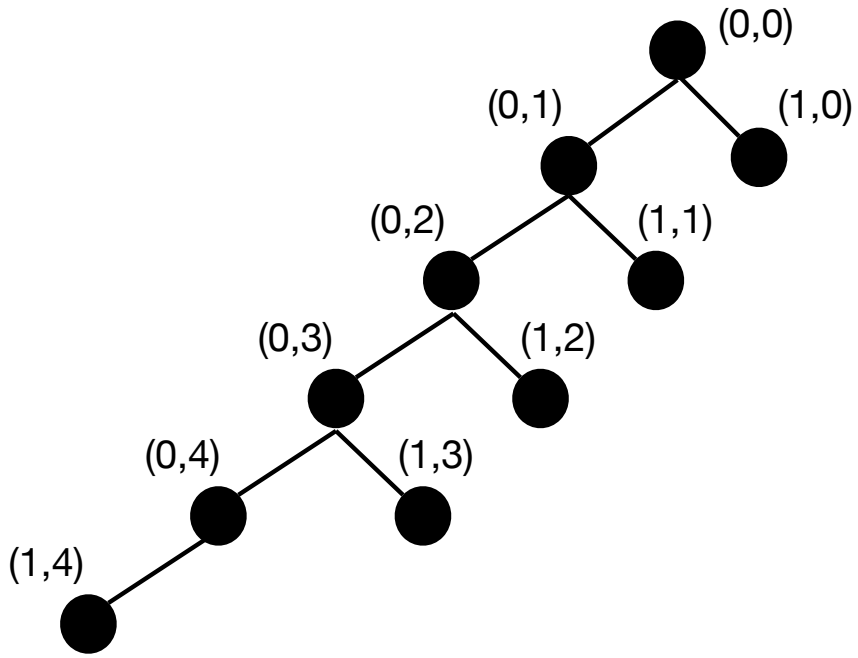
Algorithms and Search

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

Search Order: N, E, S, W

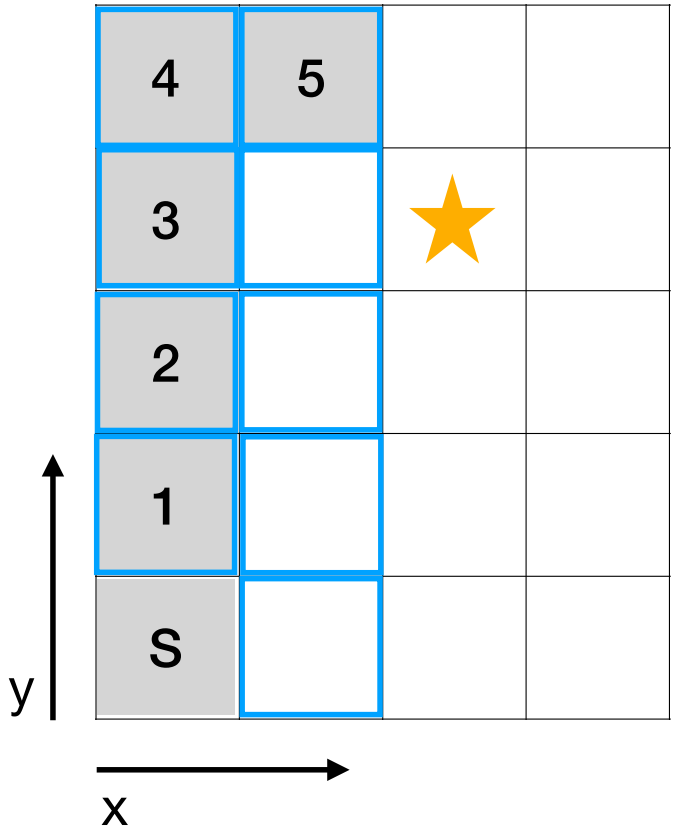


Depth First Search

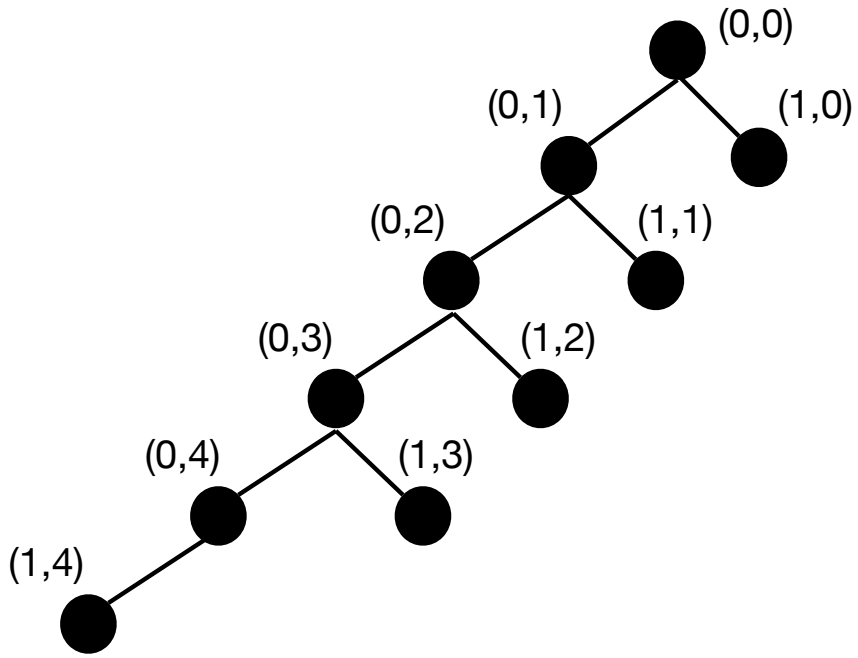


and so on…

Search Order: N, E, S, W

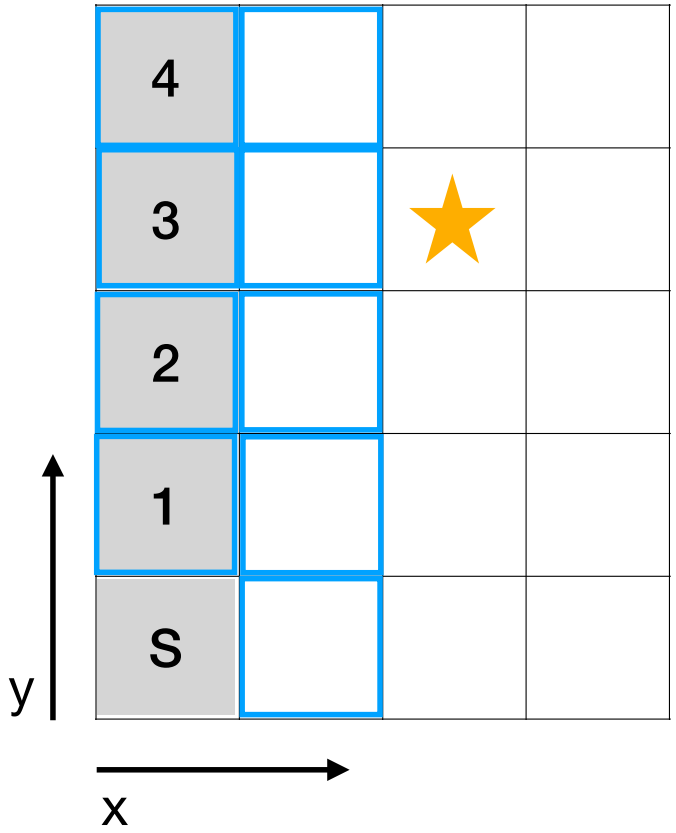


Depth First Search (DFS)

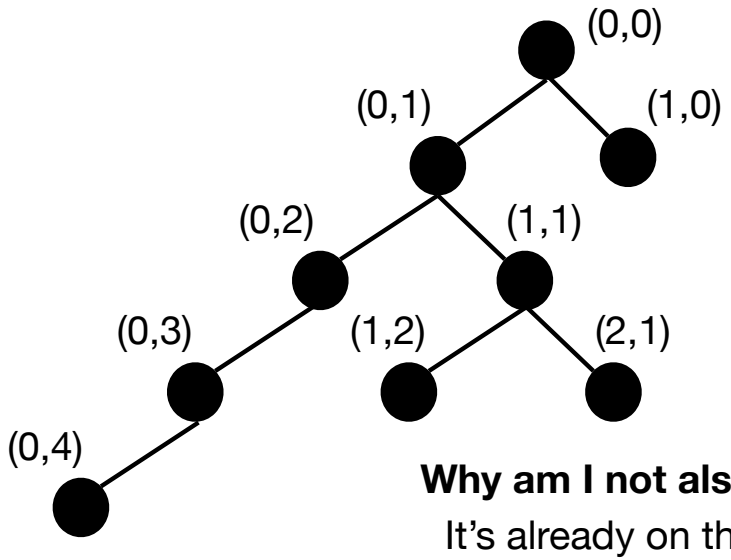


and so on…

Search Order: N, E, S, W



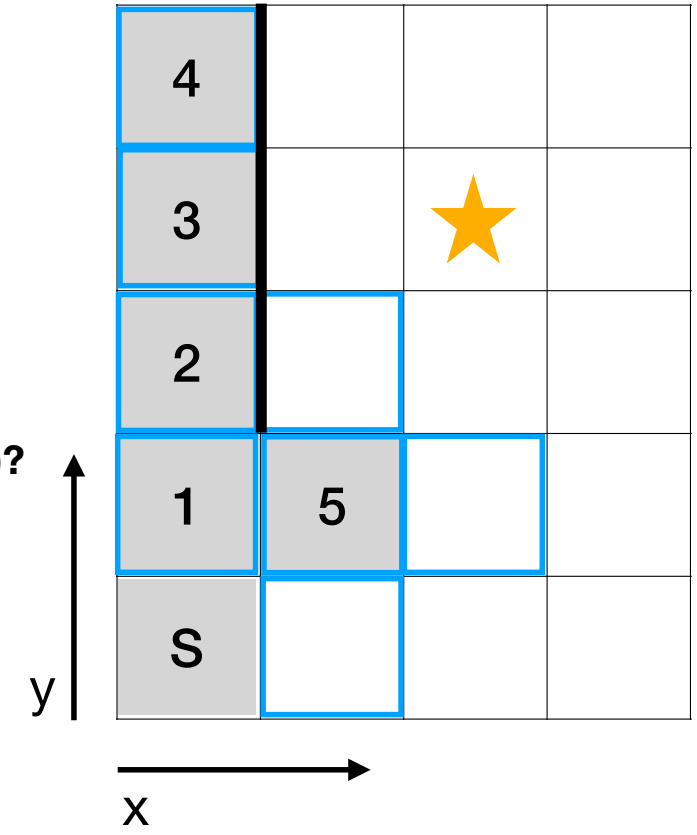
Depth First Search



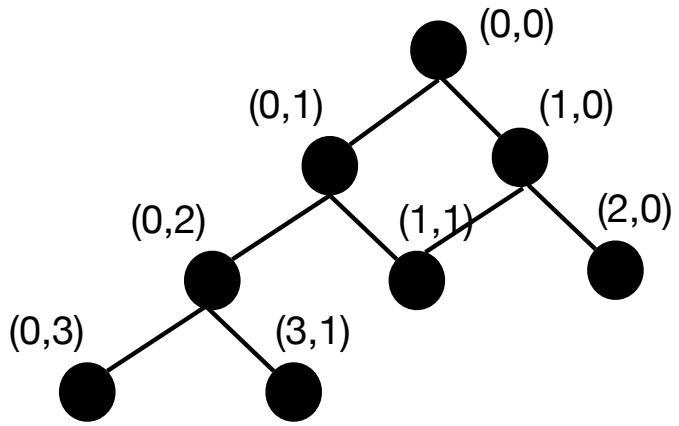
Why am I not also adding (1,0)?
It's already on the frontier

and so on…

Search Order: N, E, S, W

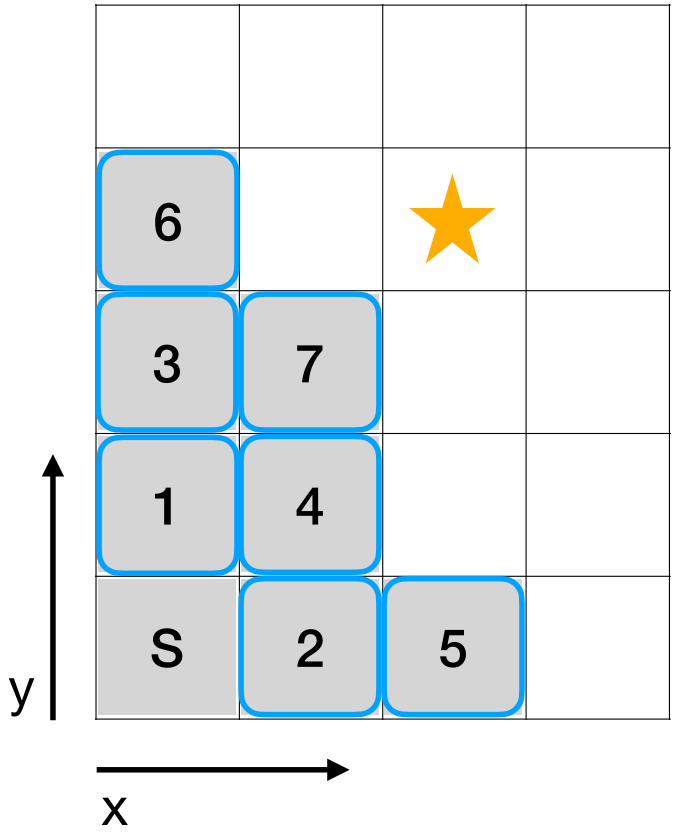


Breadth First Search



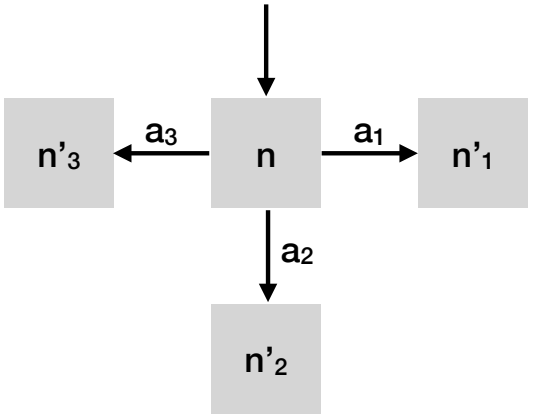
and so on…

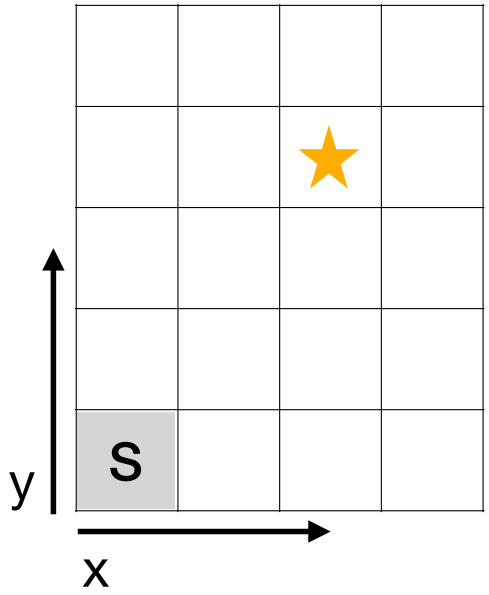
Search Order: N, E, S, W



Search Algorithms, General

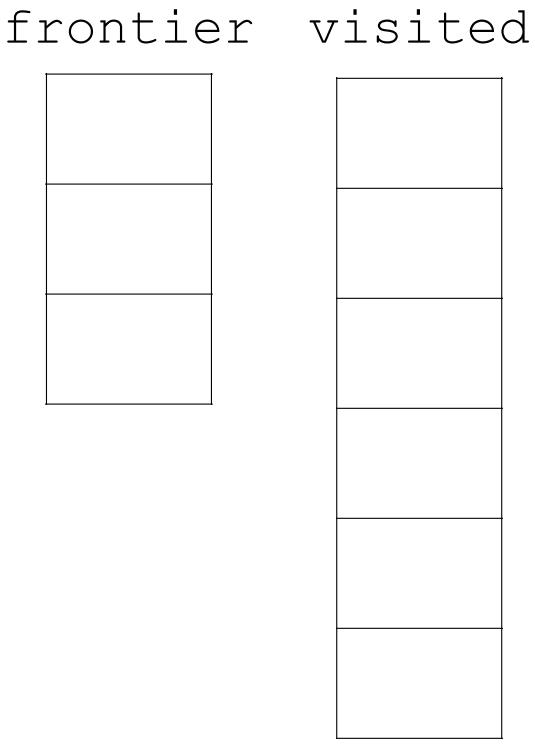
- For every node, n
- There is a set of actions, a
- That moves you to a new node, n’

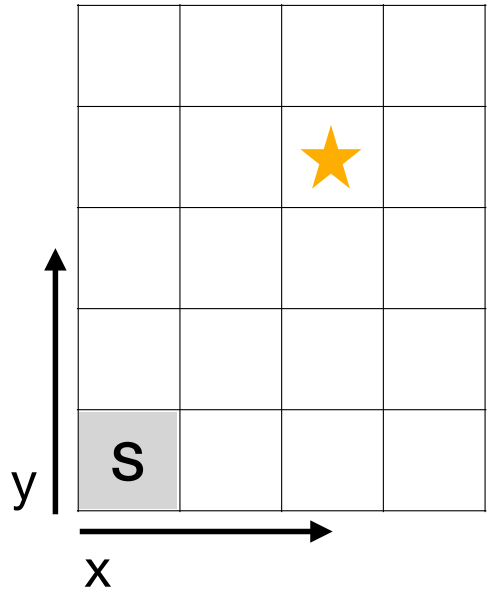




Search Algorithms, General

```
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
```

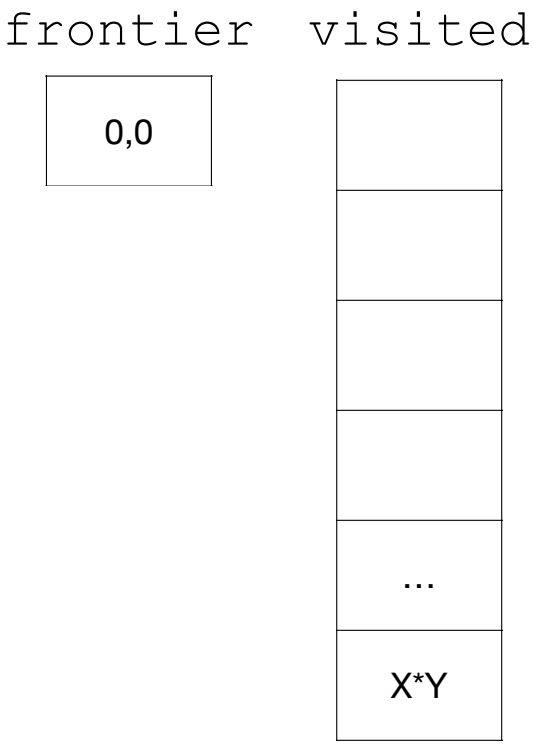


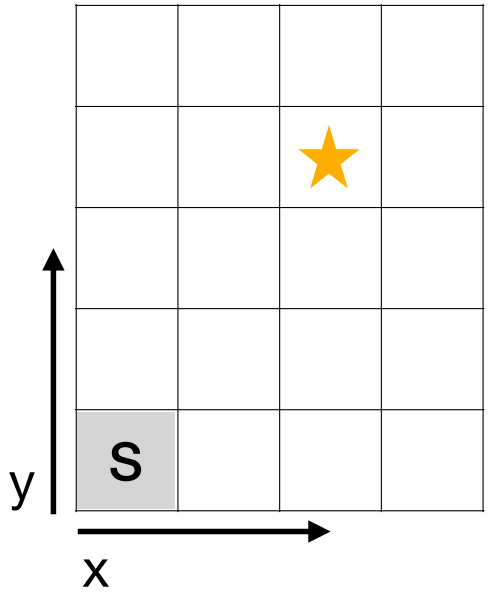


How much space do we allocate to the buffers?

Depth First Search (DFS)

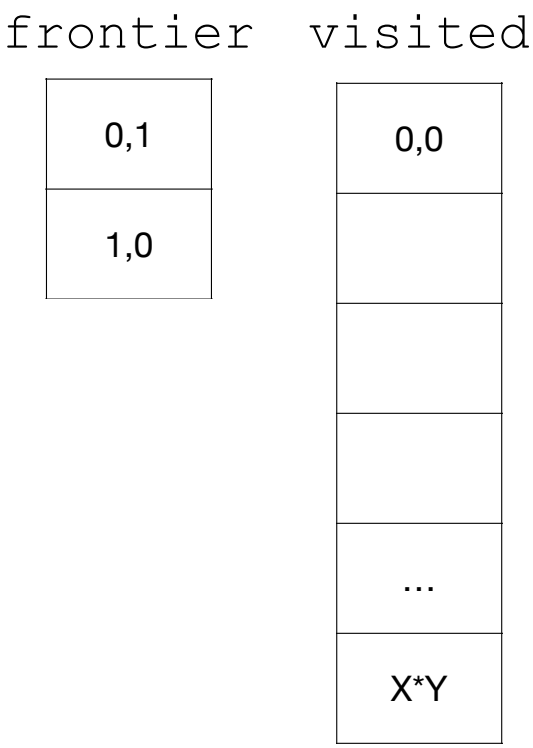
```
n = state(init)
frontier.append(n)
while(frontier not empty)
  n = pull state from frontier
  append n to visited
  if n = goal, return solution
  for all actions in n
    n' = a(n)
    if n' not visited
      append n' to frontier
```

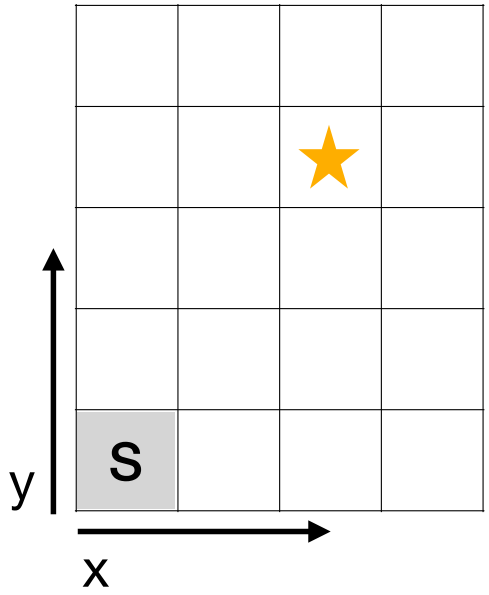




Depth First Search (DFS)

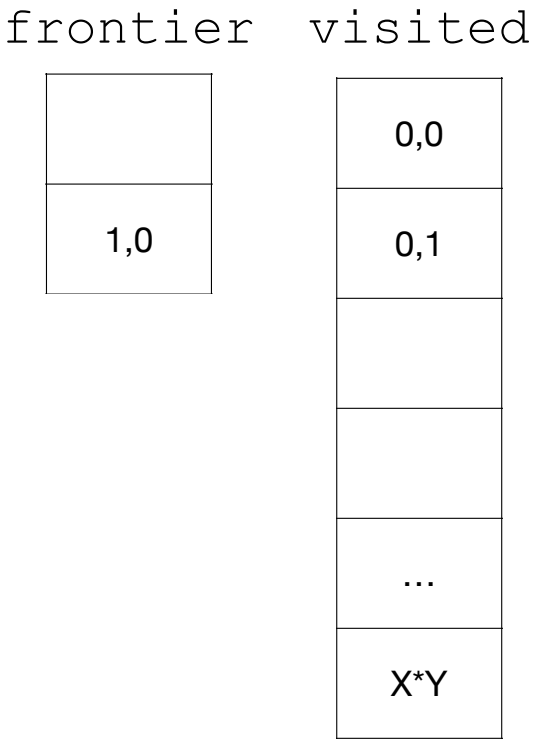
```
n = state(init)
frontier.append(n)
while(frontier not empty)
  n = pull state from frontier
  append n to visited
  if n = goal, return solution
  for all actions in n
    n' = a(n)
    if n' not visited
      append n' to frontier
```

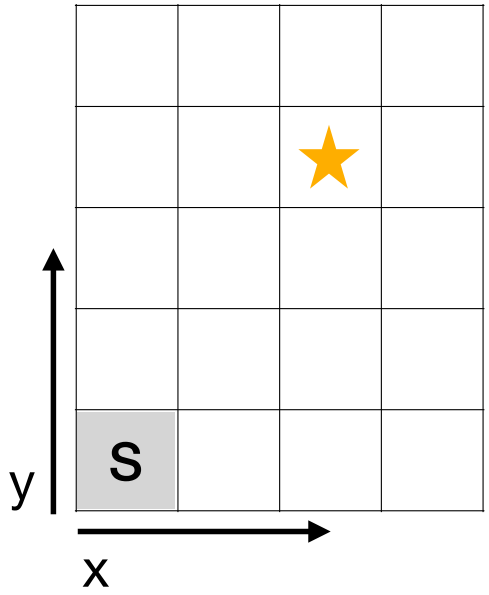




Depth First Search (DFS)

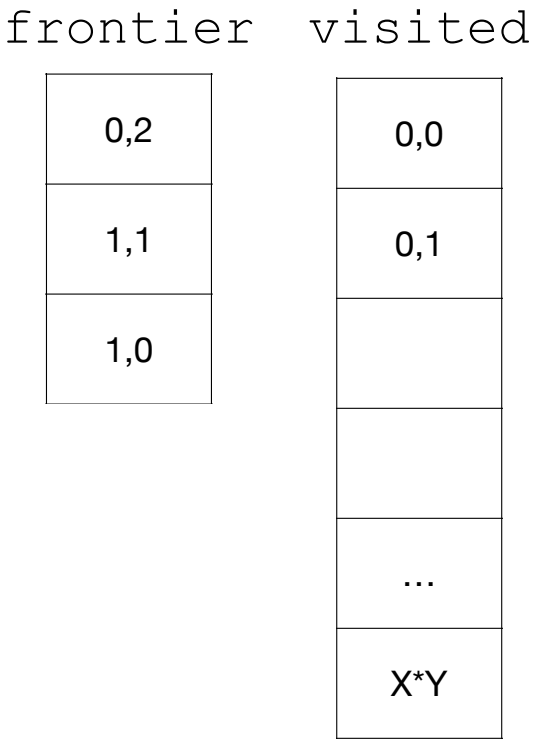
```
n = state(init)
frontier.append(n)
while(frontier not empty)
  n = pull state from frontier
  append n to visited
  if n = goal, return solution
  for all actions in n
    n' = a(n)
    if n' not visited
      append n' to frontier
```

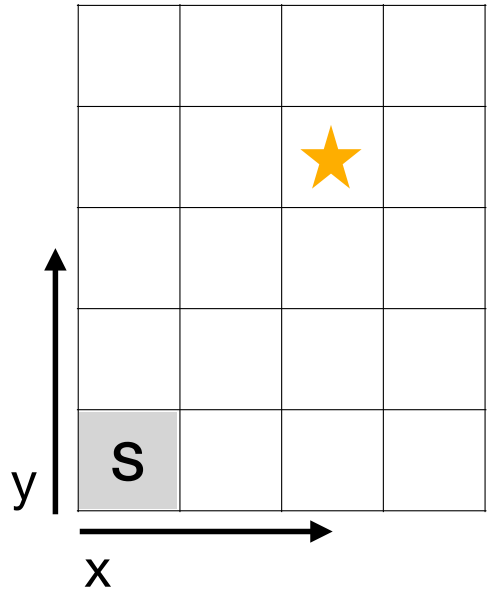




Depth First Search (DFS)

```
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
```



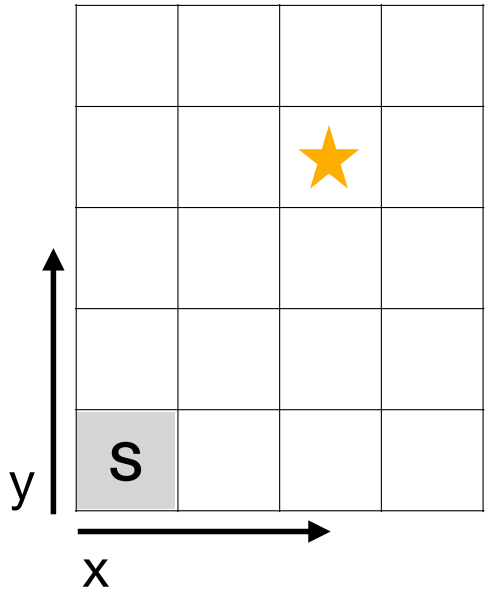


Depth First Search (DFS)

```
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
0,3
1,2
1,1
1,0

visited
0,0
0,1
0,2
...
X*Y

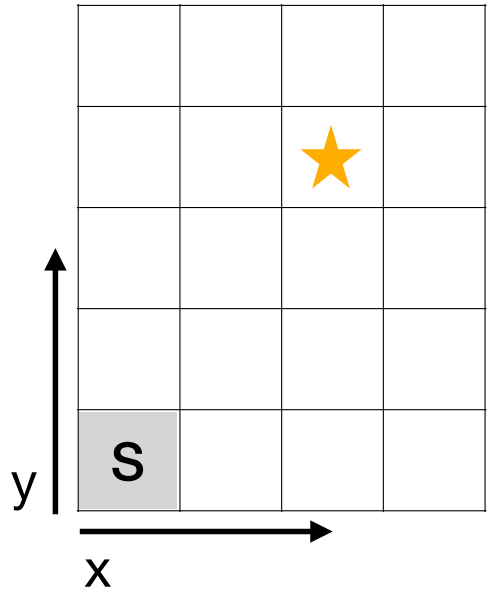


Depth First Search (DFS)

```
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
```

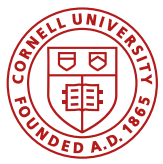
frontier
0,4
1,3
1,2
1,1
1,0

visited
0,0
0,1
0,2
0,3
...
X*Y



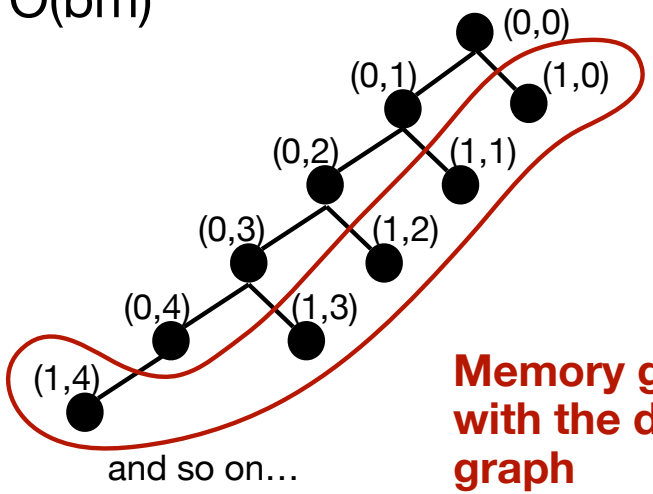
What is the frontier buffer?

Last-In First-Out (LIFO)



Depth First Search (DFS)

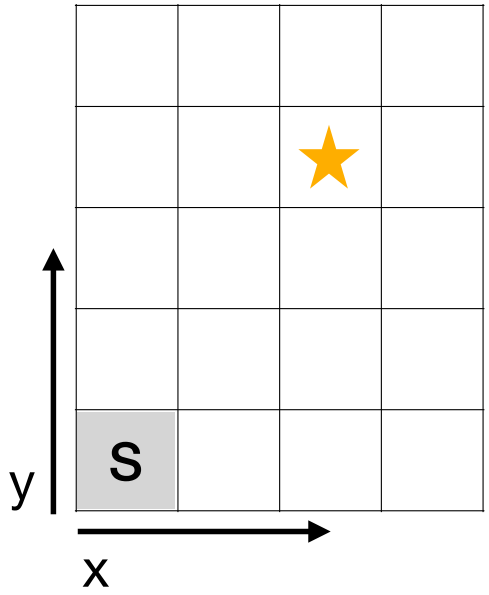
- Is it complete?
  - Yes, but only on finite graphs
- What is the time complexity?
  - $O(b^m)$
- What is the space complexity?
  - O(bm)



Memory grows linearly with the depth of the graph

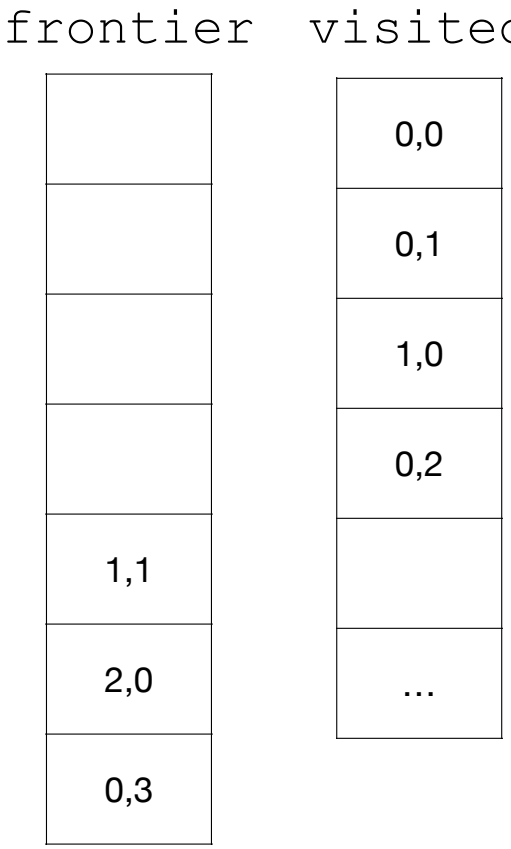
frontier
0,4
1,3
1,2
1,1
1,0

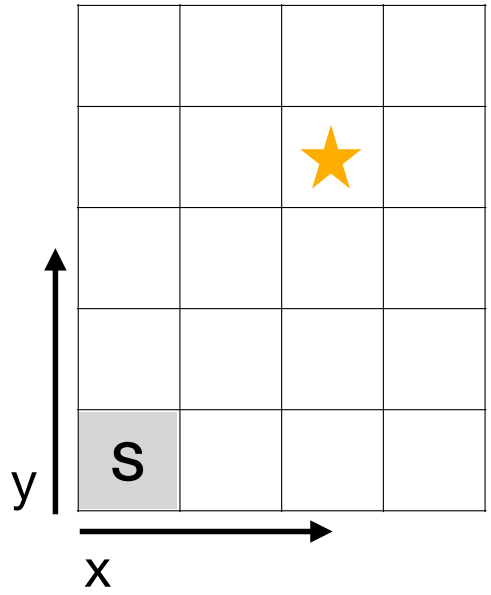
visited
0,0
0,1
0,2
0,3
…
X*Y

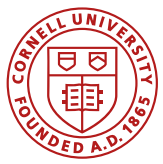


Breadth First Search (BFS)

```
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
      append n' to frontier
```

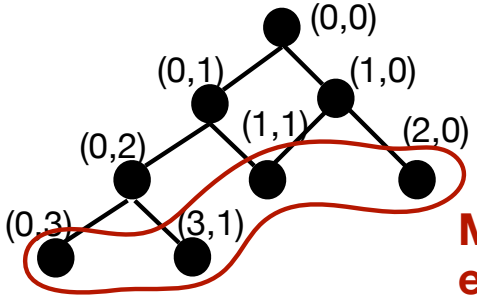






Breadth First Search (BFS)

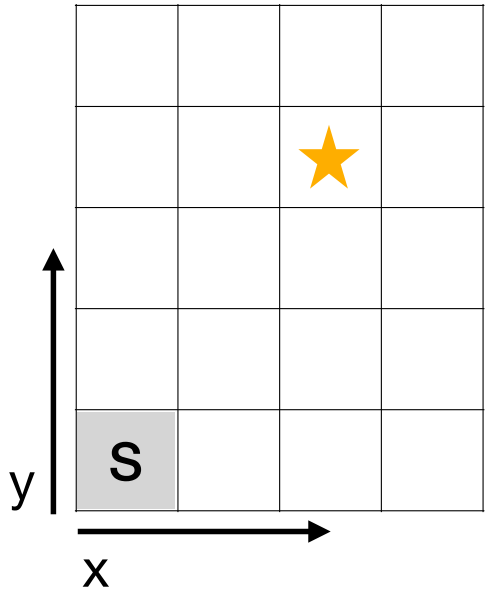
- Is it complete?
  - Yes, as long as b is finite
- Is it optimal?
  - Yes
- What is the time complexity?
  - $O(b^m)$
- What is the space complexity?
  - $O(b^m)$



Memory grows exponentially with the depth of the graph

frontier
1,1
2,0
0,3

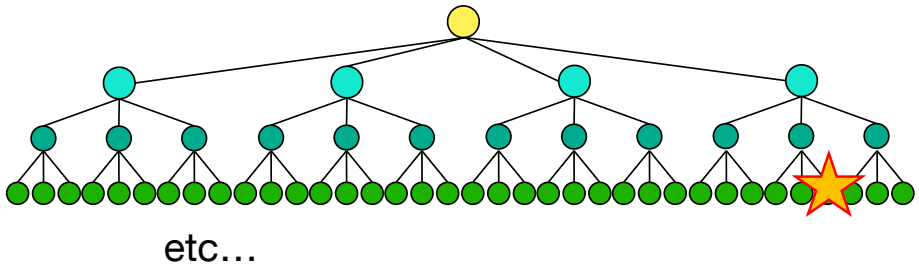
visited
0,0
0,1
1,0
0,2
…

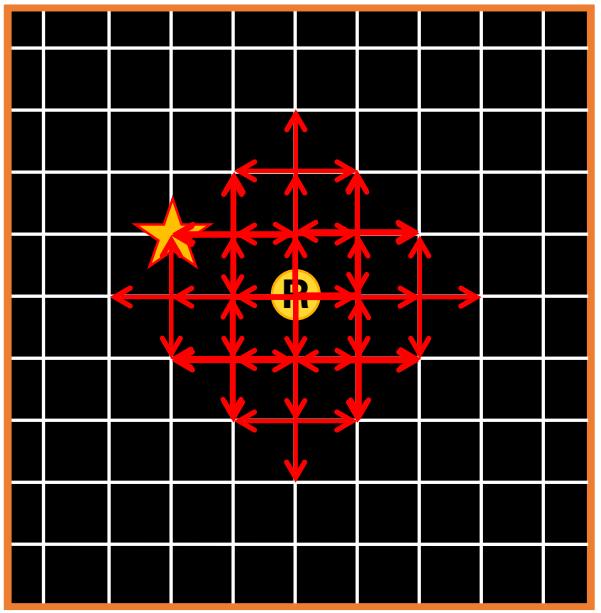


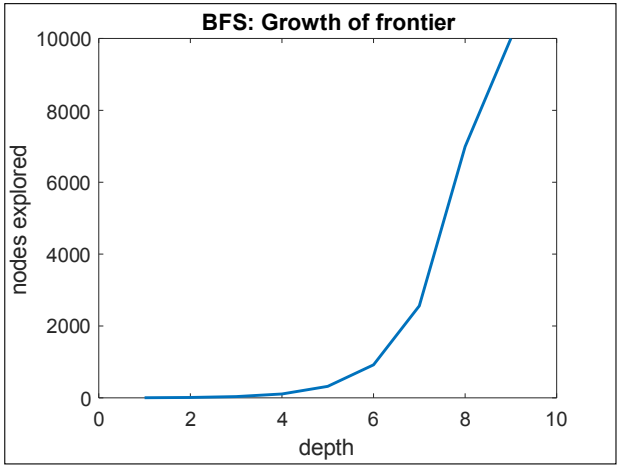
BFS: Memory and Computation

Frontier size:

- 4
- 12
- 36

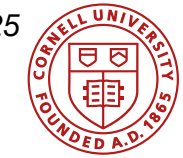






Uninformed Search Algorithms

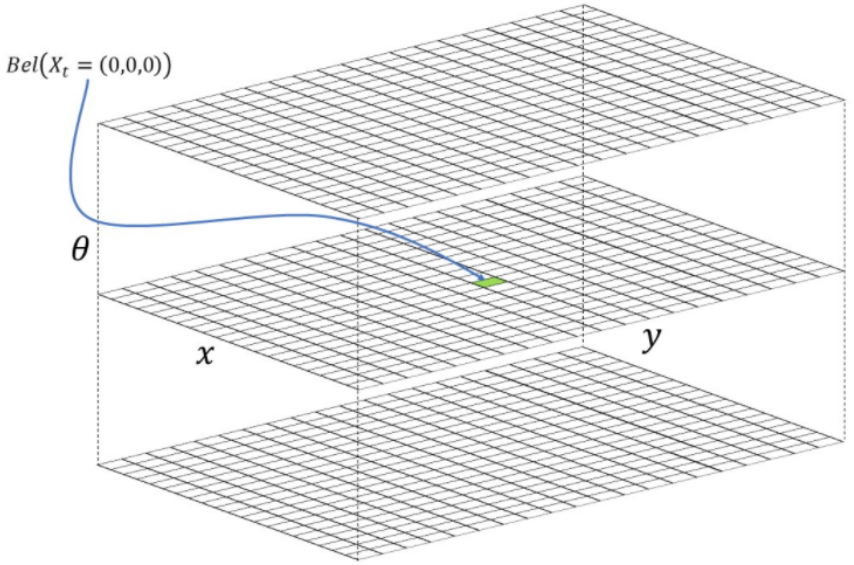
- When is DFS appropriate?
  - If the memory is restricted
  - If solutions tend to occur at the same depth in the tree
- When is DFS inappropriate?
  - If some paths have infinite length / if the graph contains cycles
  - If some solutions are very deep, while others are very shallow
- When is BFS appropriate?
  - If you need to find the shortest path
  - If memory is not a problem
  - If some solutions are shallow
  - If there might be infinite paths
- When is BFS inappropriate?
  - If memory is limited / if the branching factor is very large
  - If solutions tend to be located deep in the tree



Applications in Fast Robots

Is BFS/DFS possible on the Artemis?

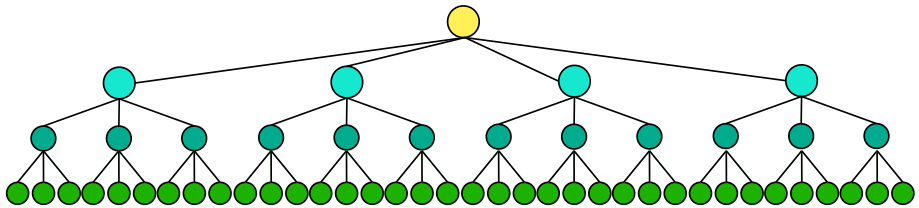
- What is the maximum branching factor?
  - b = 4
- What is the longest path?
  - m ~ 20*20 = 400
- Depth First Search
  - Frontier: O(bm) = 1,600 nodes
  - Float -> 6.4kB
  - Artemis memory?
    - 1MB flash and 384k RAM
- Breadth First Search
  - Frontier: O(bm) = 420*20 = 6.7e240 nodes

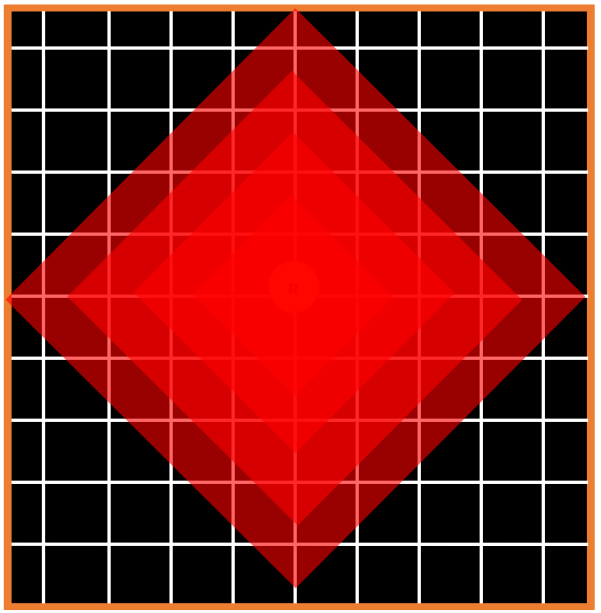


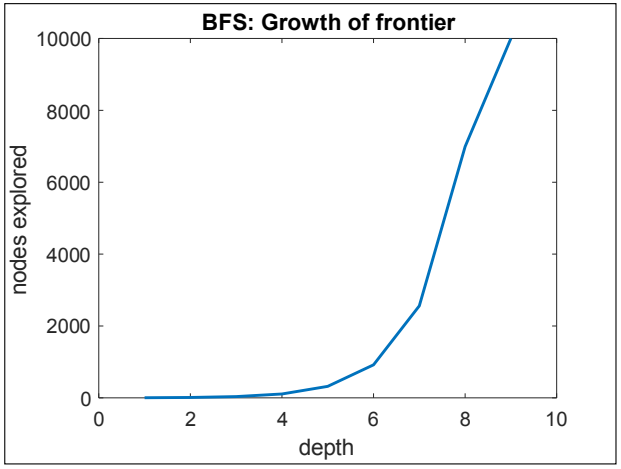
BFS: Memory and Computation

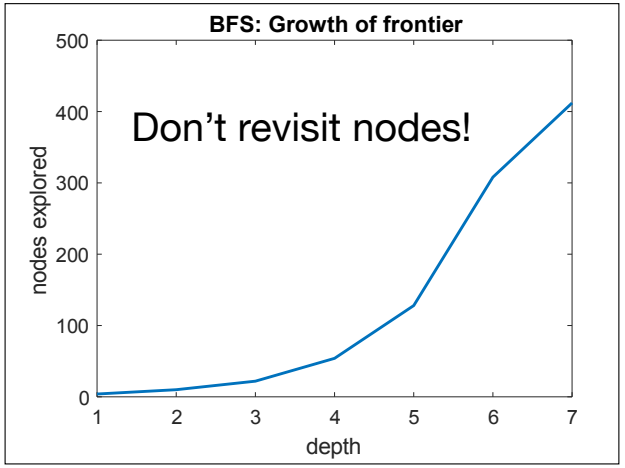
Frontier size:

- 4
- 12
- 36



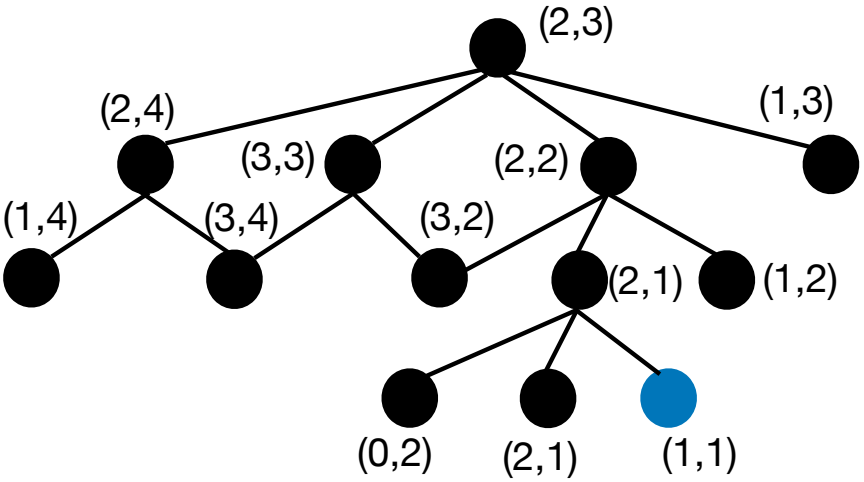






Lowest-Cost First Search (LCFS)

Consider parent cost!



What node to expand next?

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

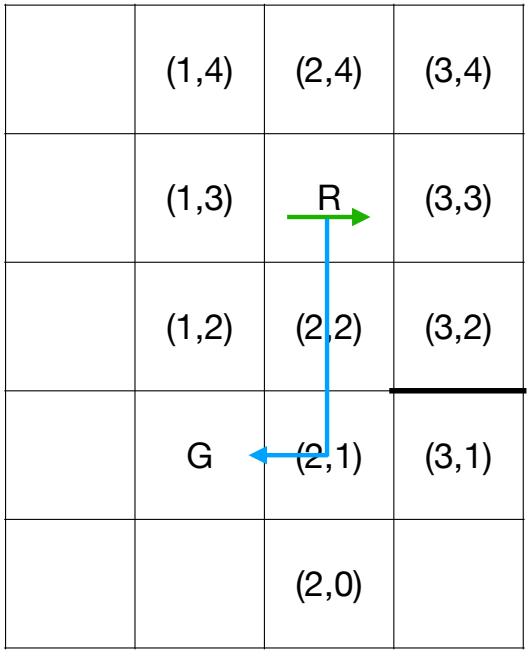
Lowest-Cost First Search (LCFS)

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(goal,n')
      frontier.append(priority)
```

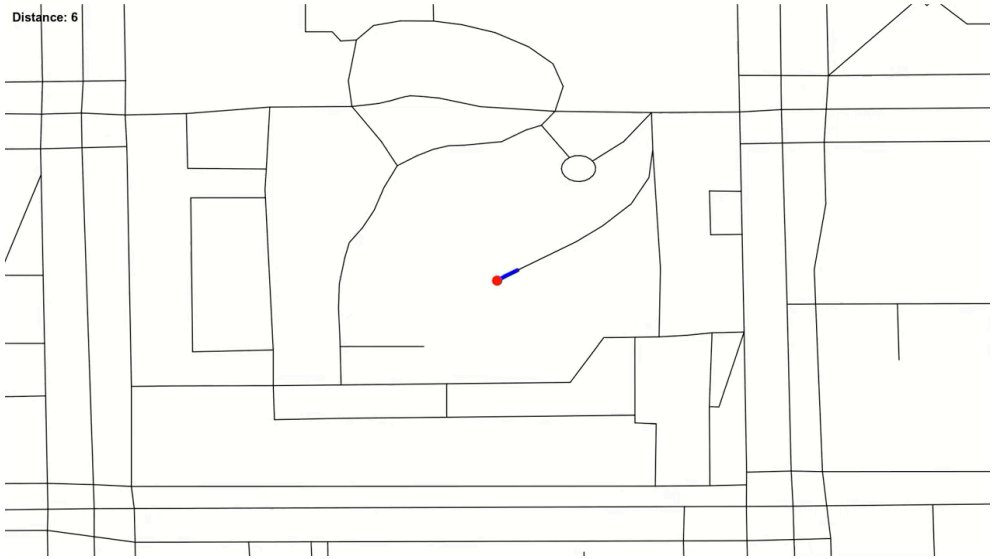
What cost heuristic could we add?

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



Lowest-Cost First Search (LCFS)

- Is it complete?
  - Yes, as long as path costs are positive
- What is the time complexity?
  - $O(b^m)$
- What is the space complexity?
  - $O(b^m)$



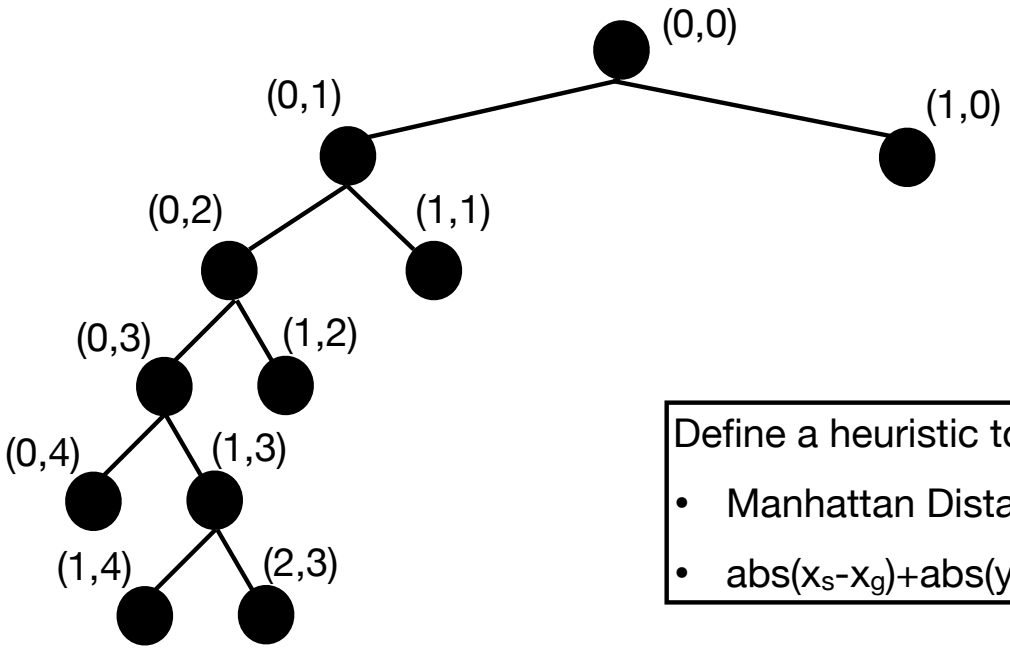
https://www.youtube.com/watch?v=t7UjtzqIXSA

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

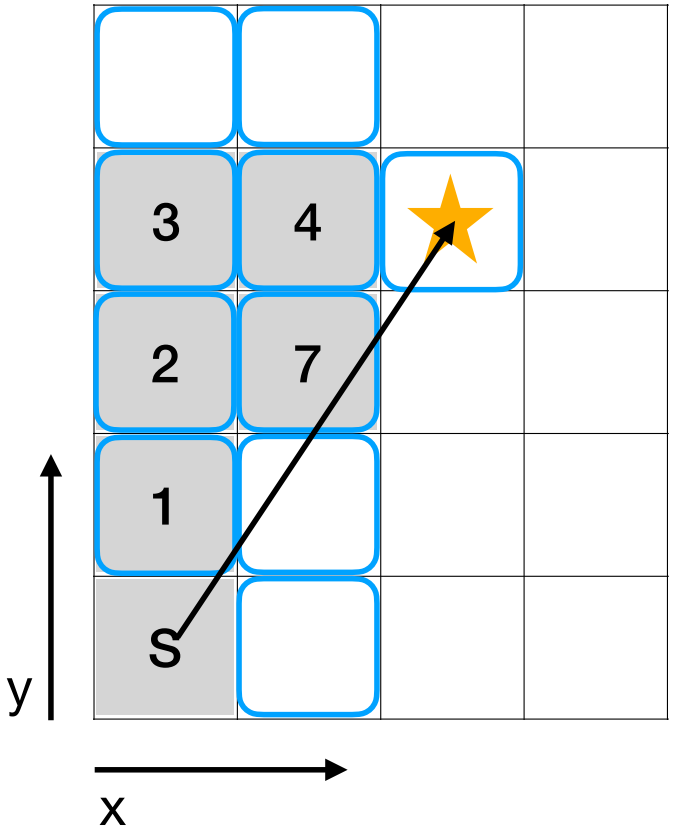
Greedy Search



Define a heuristic to target:

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

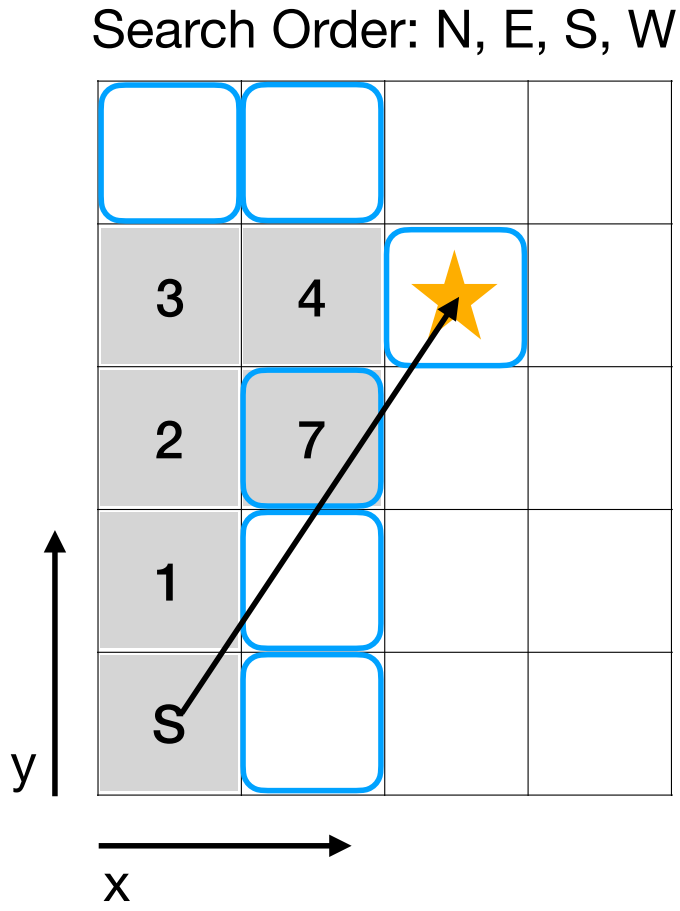
Search Order: N, E, S, W



Informed Search

Greedy Search

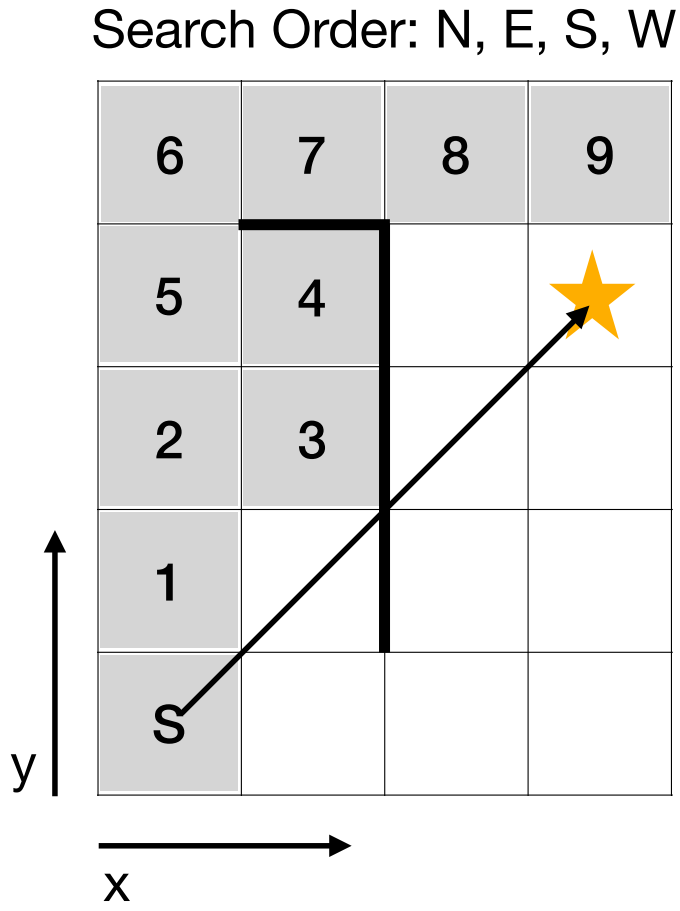
```
n = state(init)
frontier.append(n)
while(frontier not empty)
  n = pull state from frontier
  visited.append(n)
  if n = goal, return solution
  for all actions in n
    n' = a(n)
    if n' not visited
        priority = heuristic(goal,n')
        frontier.append(priority)
```



Informed Search

Greedy Search

- Is it complete?
  - No
- What is the time complexity?
  - $O(b^m)$
- What is the space complexity?
  - $O(b^m)$
- Optimal?
  - No…



Search Algorithms, general

- Breadth First Search
  - Complete and optimal
  - …but searches everything
- Lowest-Cost First Algorithm **Considers parent cost**
  - Complete and optimal
  - … but it wastes time exploring in directions that aren't promising
- Greedy Search **Considers goal**
  - Complete (in most cases)
  - … only explores promising directions

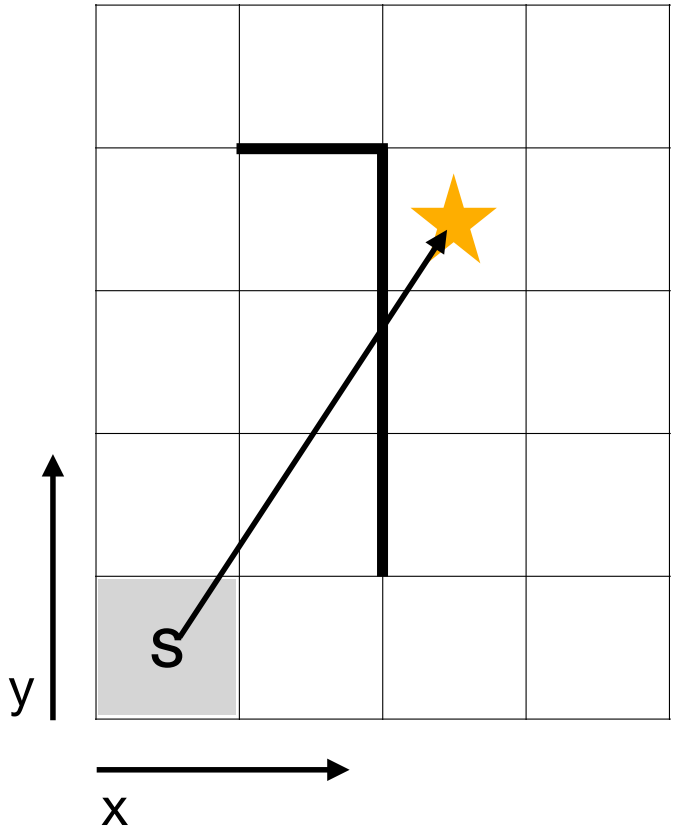
Can we do better? A*

Informed Search

A* (A-star)

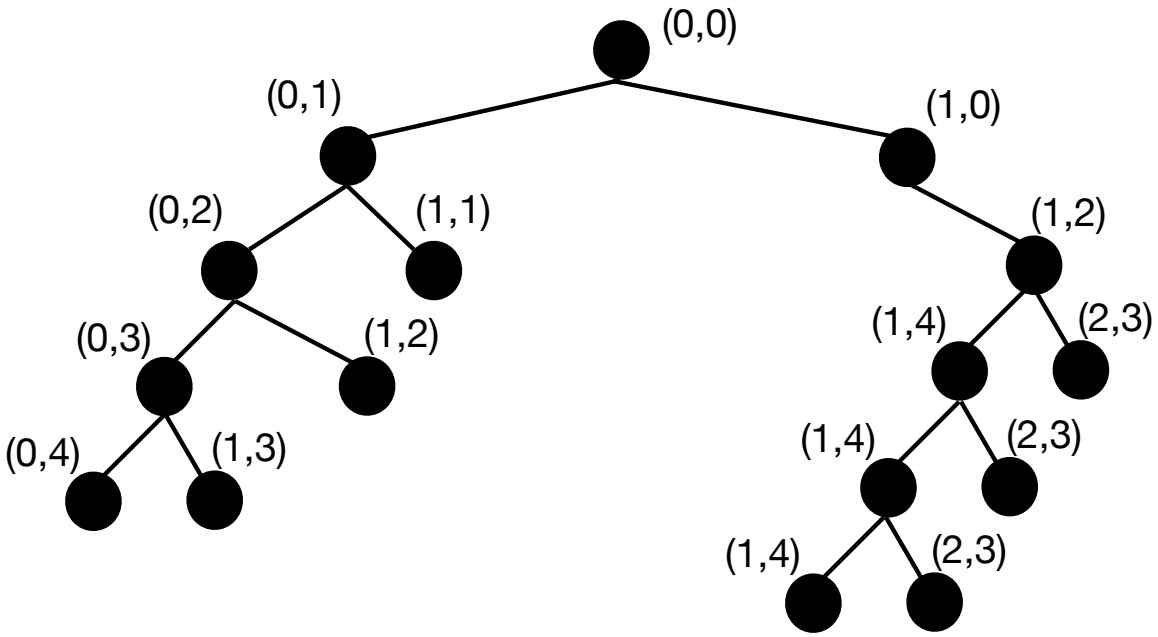
```
n = state(init)
frontier.append(n)
while(frontier not empty)
  n = pull state from frontier
  if n = goal, return solution
  for all actions in n
    n' = a(n)
    if ((n' not visited or
        (visited and n'.cost < n_old.cost))
        priority = heuristic(goal,n')+cost
        frontier.append(priority)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

Search Order: N, E, S, W

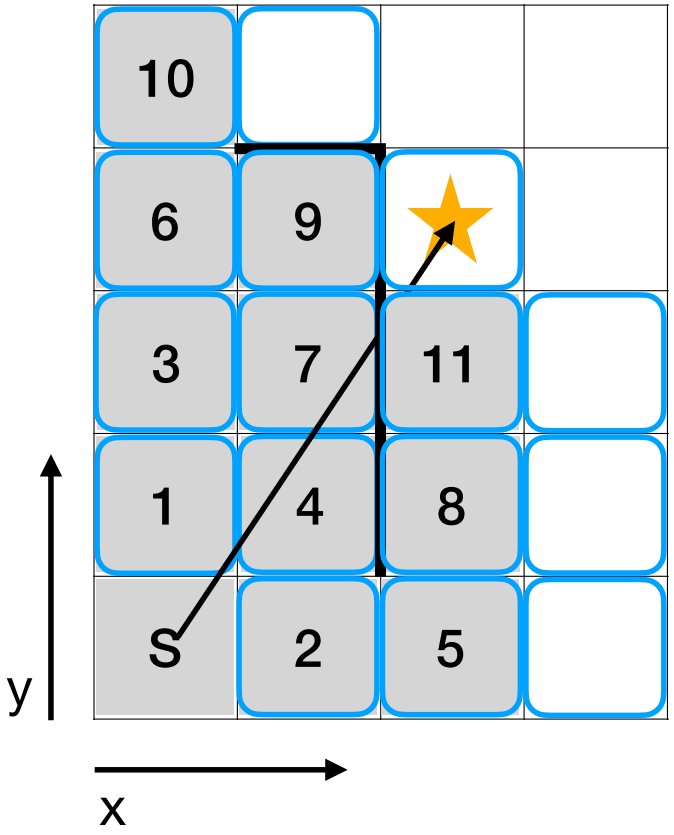


Informed Search

A* (A-star)



Search Order: N, E, S, W



A* Search

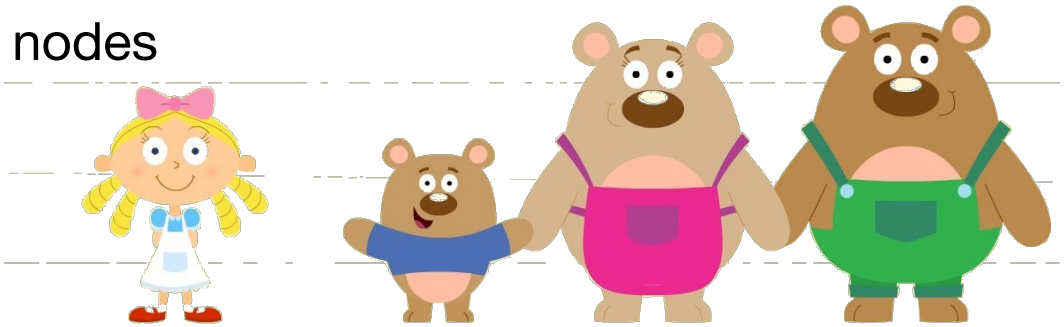
- What if the heuristic is too optimistic?
  - Estimated cost < true cost

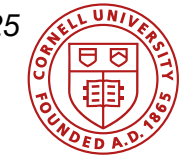
Admissible heuristic

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

Inadmissible heuristic

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

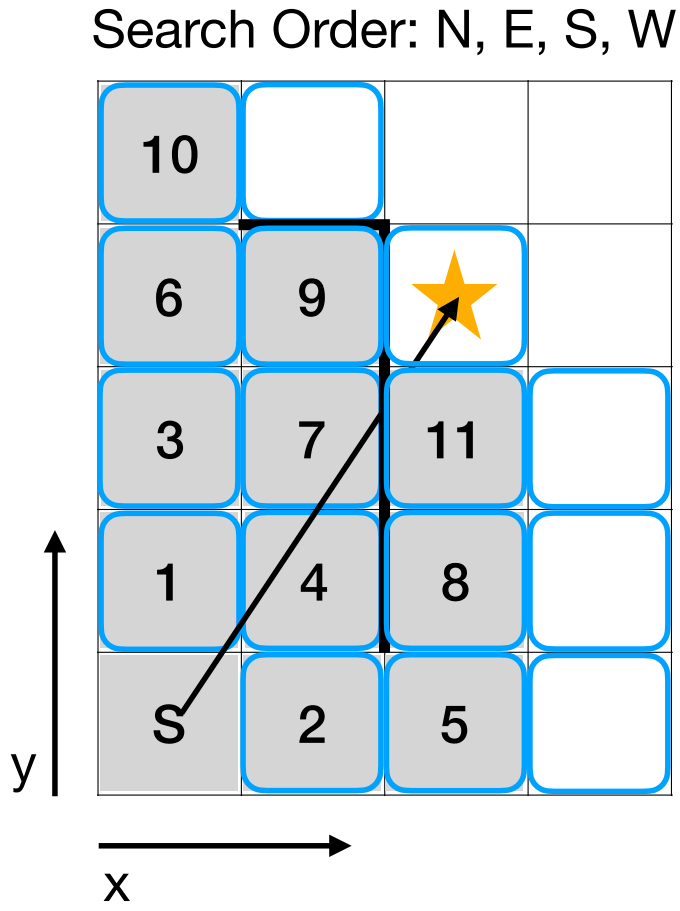




Informed Search

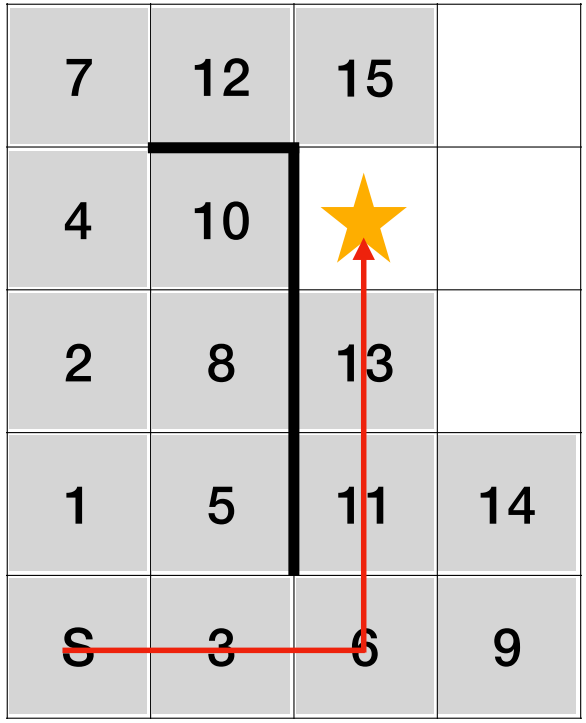
A* (A-star)

- 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!

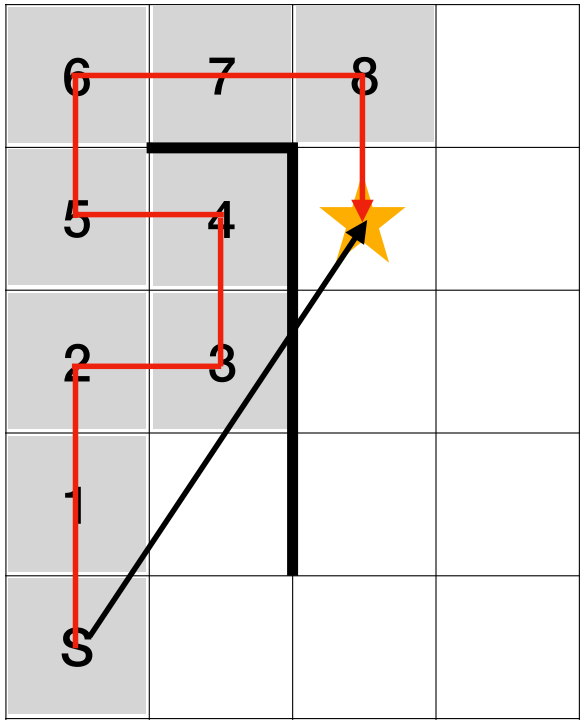


Summary

LCFS **minimum path**



Greedy



A* **minimum path & efficient**