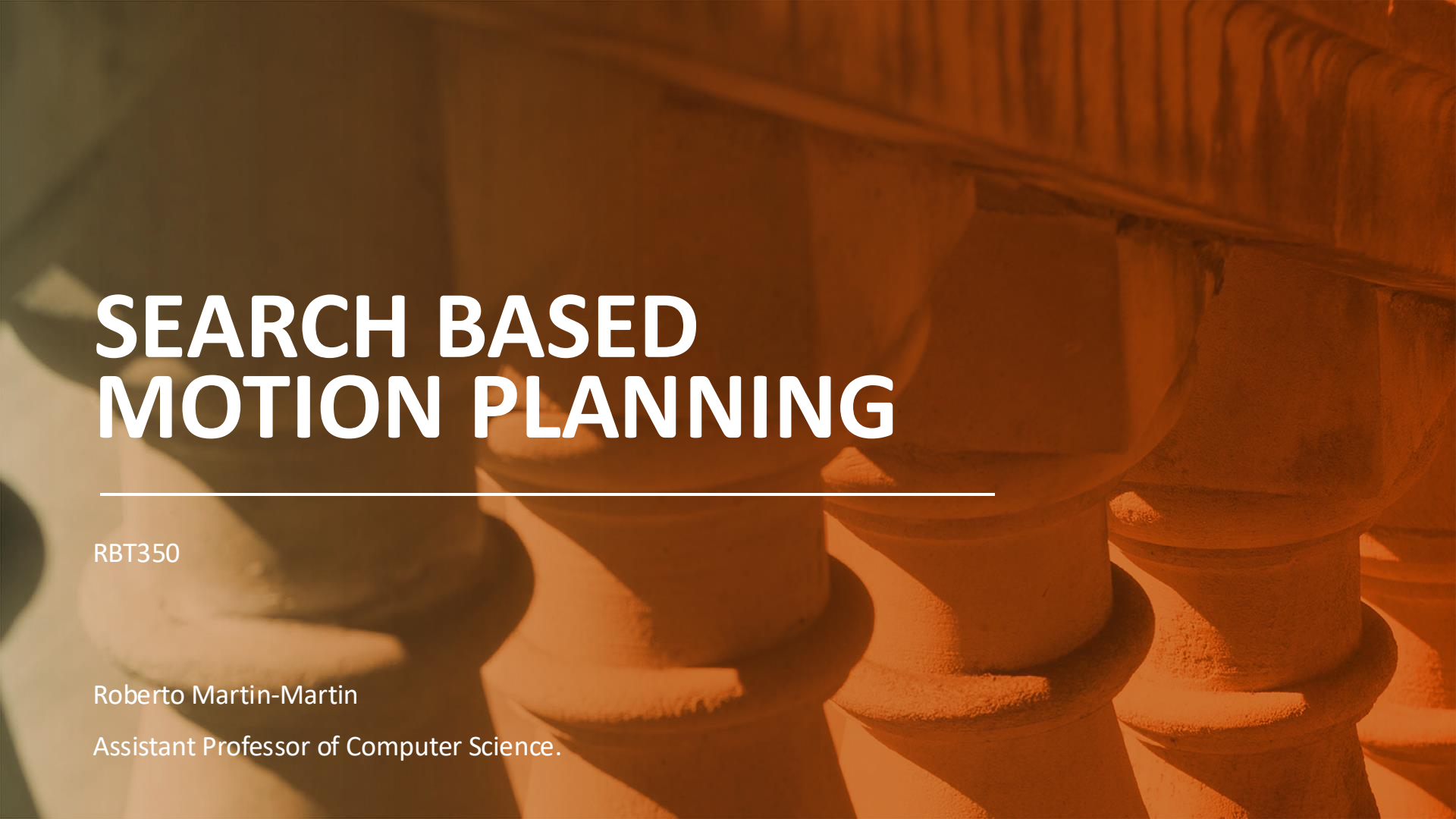


SEARCH BASED MOTION PLANNING

RBT350

Roberto Martin-Martin

Assistant Professor of Computer Science.

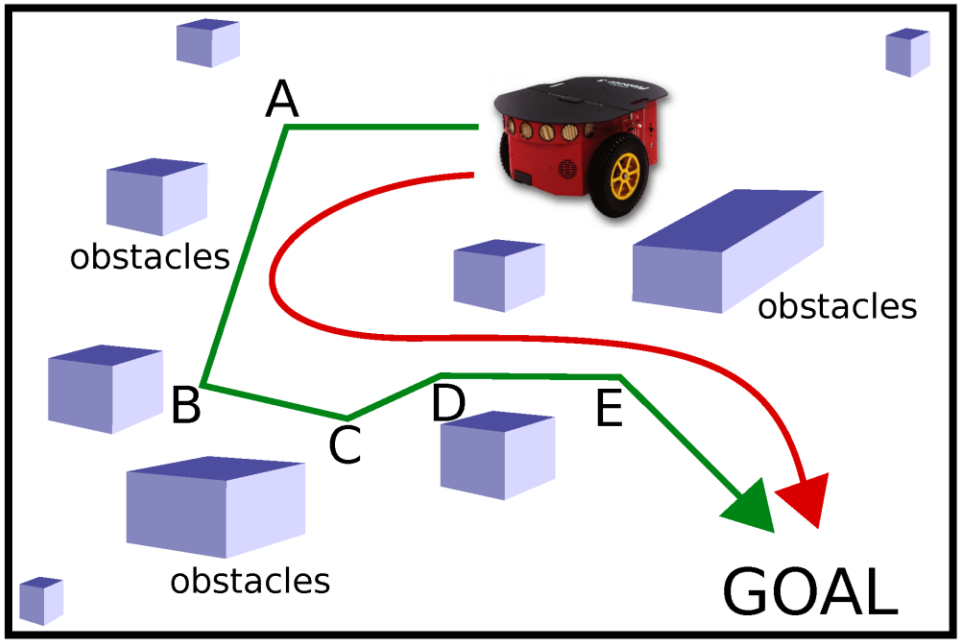
What do we have?

- Control
- Understanding of kinematics and dynamics
- Vision

What do we need?

A way to go from A to B without colliding!

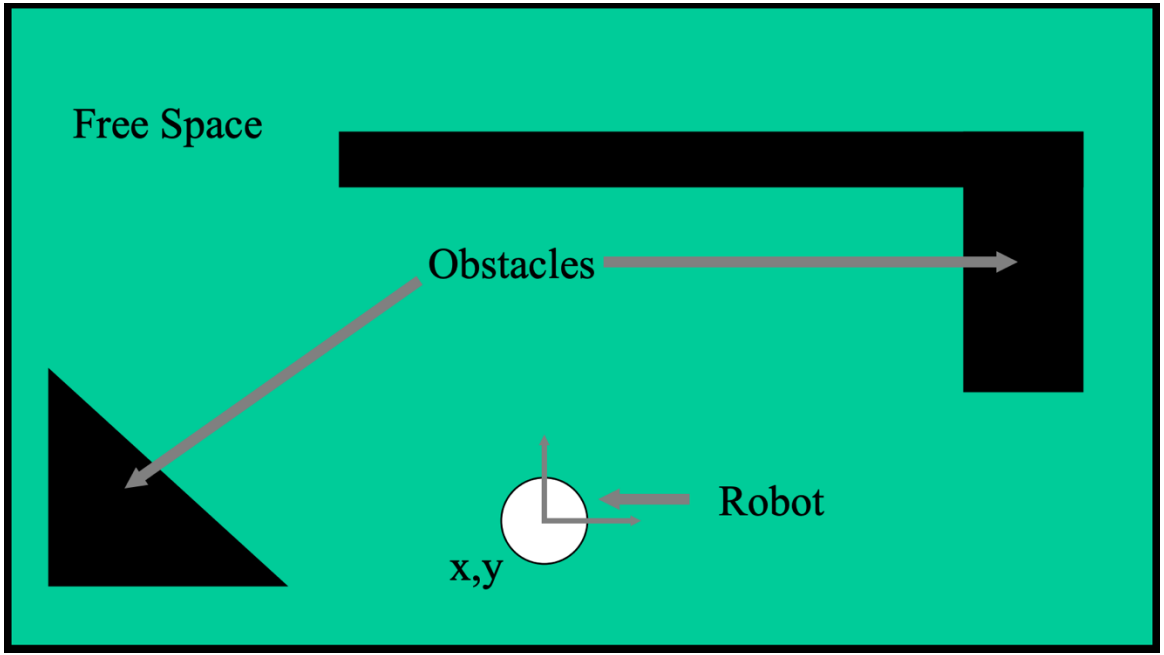
Another case



Motion Planning

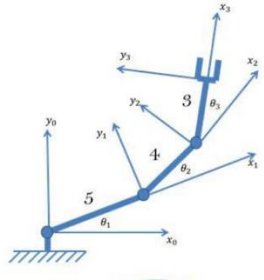
- Algorithm to find a path (sequence of states) to go from A to B without colliding
  - An algorithm is q procedure or formula for solving a problem, based on conducting a sequence of specified actions
- Our world for motion planning consist of:
  - Obstacles → Occupied space
    - Robot can't go there
  - Free space → Unoccupied space
    - Robot may be able to reach this

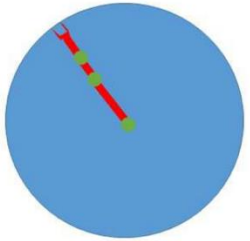
Example

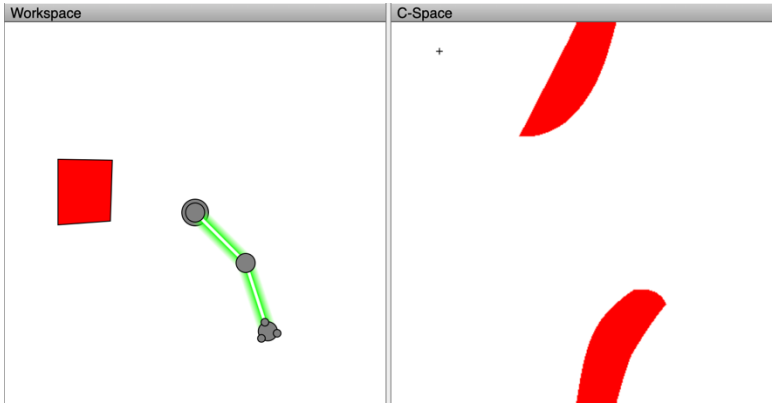


What does this look like for an arm?

- We will search for paths in the robot's configuration space
  - Configuration space, $q \in C$, where q is the vector of joint configurations
- Why configuration space and not directly Cartesian/Workspace?
- How do obstacles look in configuration space?

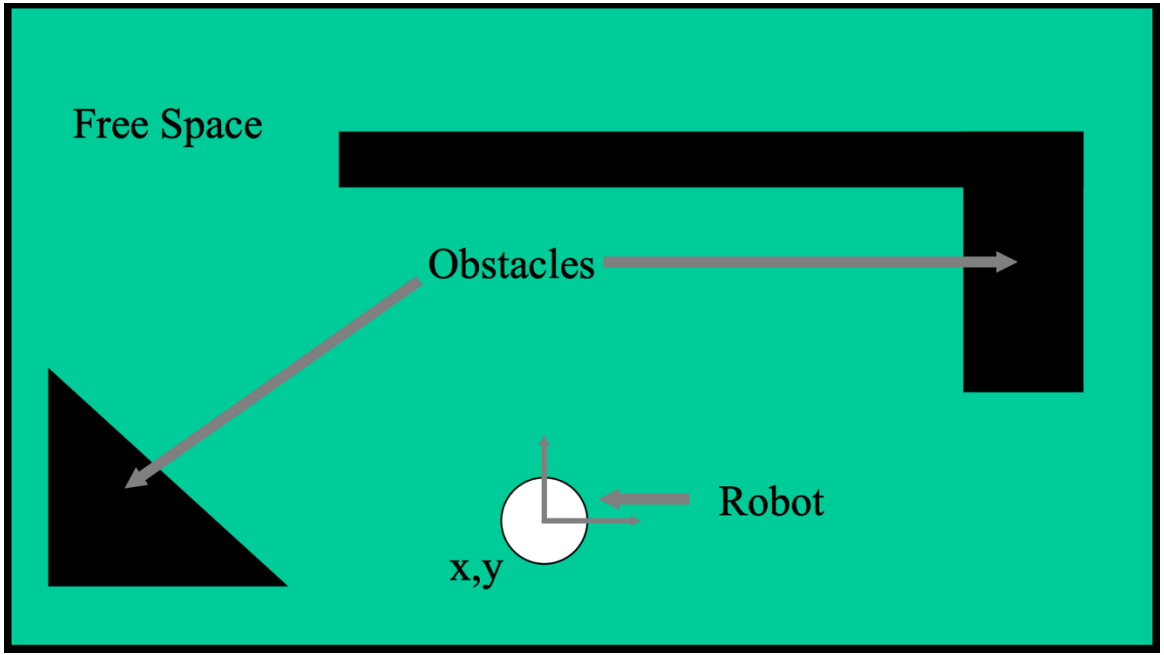




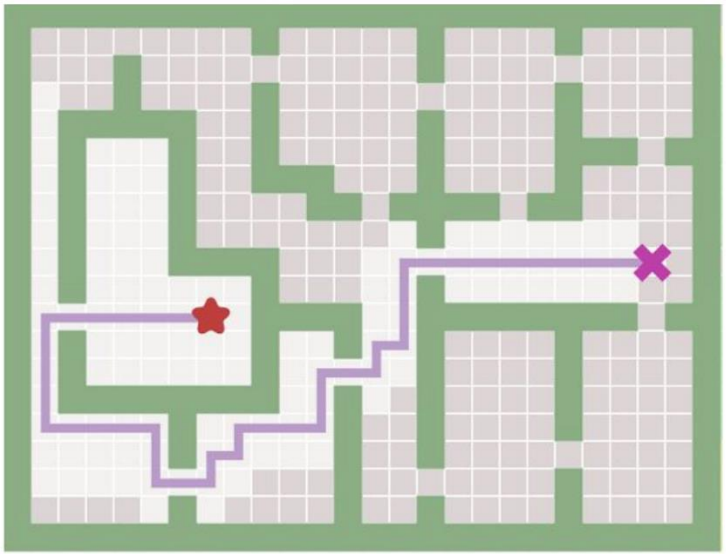


https://www.cs.unc.edu/~jeffi/c-space/robot.xhtml

Two families of solutions



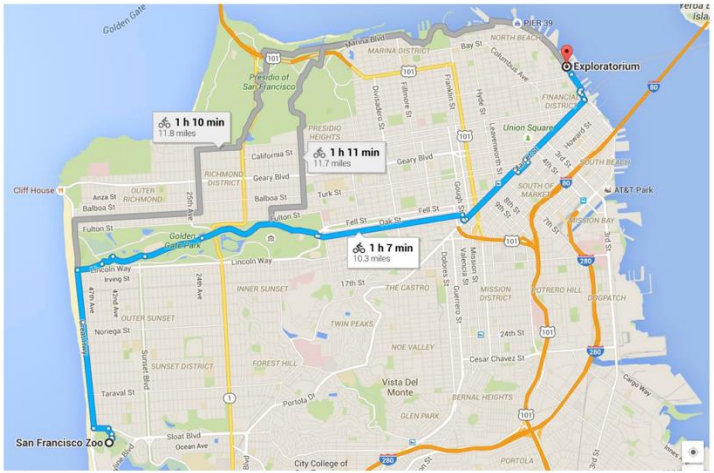
Motion planning as graph search



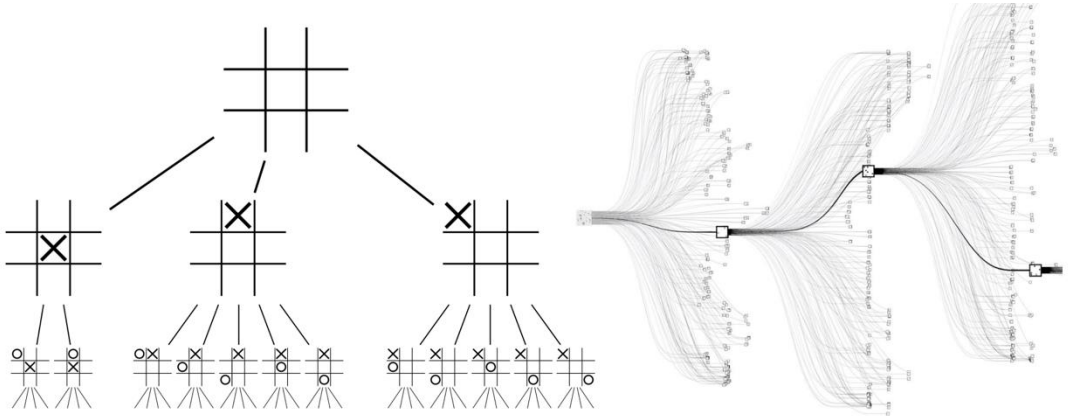
- We will transform the problem of finding a path from a starting point to a goal location as a problem to find a connection between two nodes in a graph
- The solution involves then two steps:
  - Discretize the problem → construct the graph
  - Find a path → search in the graph

[based on Mussmann's and Biswas' slides]

Applications beyond robotics



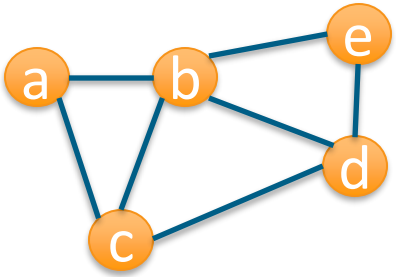
paths in maps

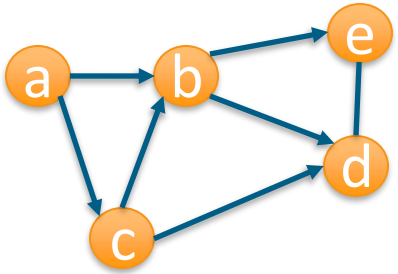


games / sequential decision making

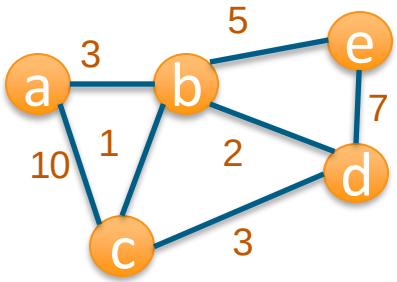
Graphs

- $G = (V, E)$



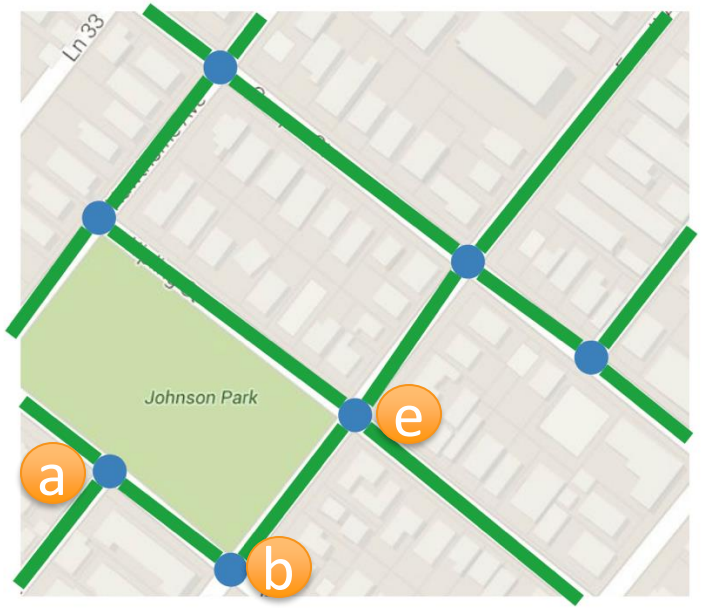


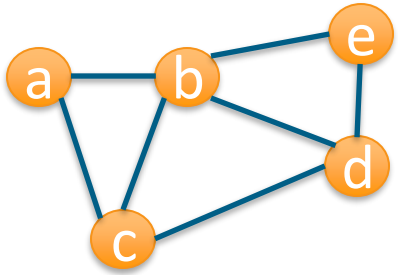
directed



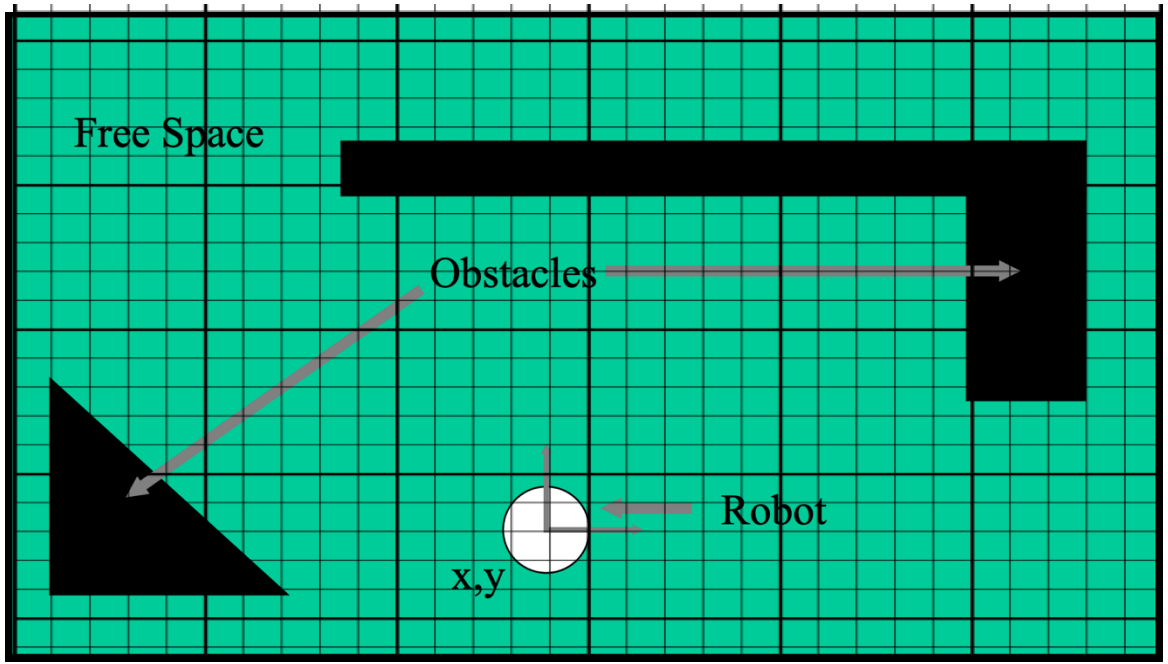
weighted

Casting problems as graph search

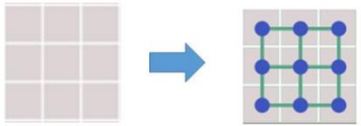




Casting problems as graph search



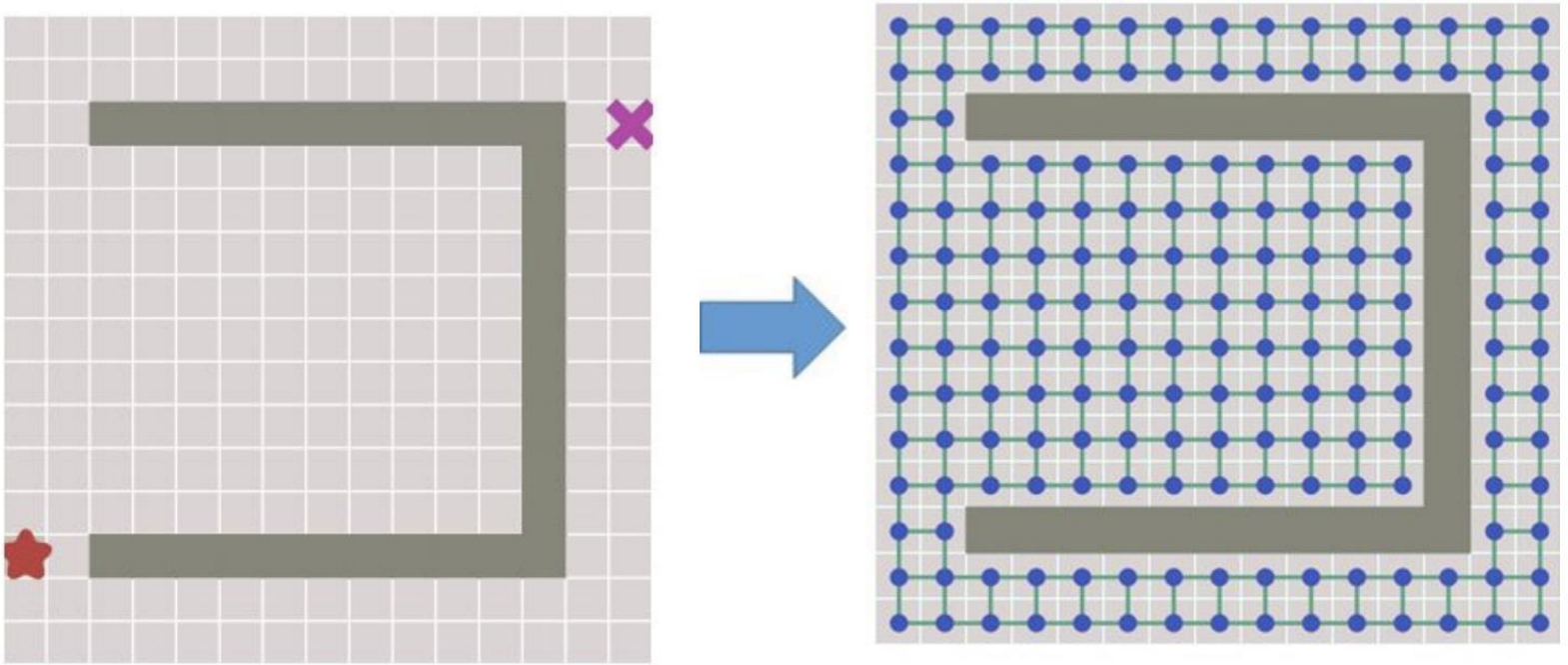
Each cell is a node



Edges connect adjacent cells.

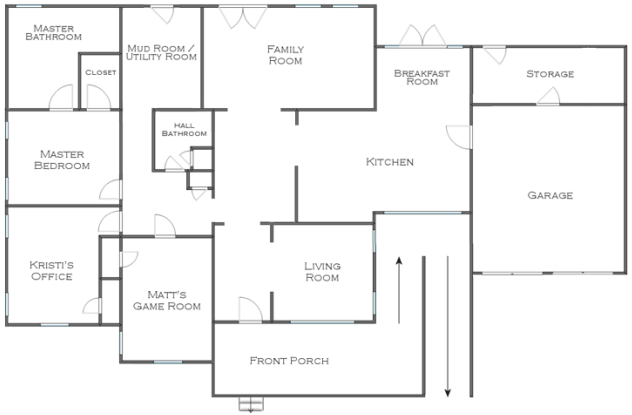
Obstacles have no edges

Casting problems as graph search



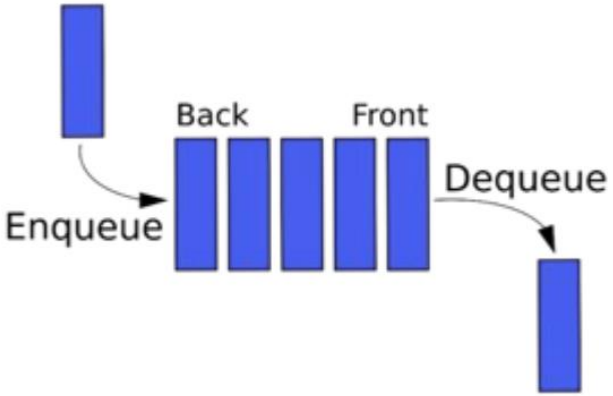
Graph Traversal Algorithms

- We have transformed the original problem into a graph equivalent problem
  - Find a collision free path from A to B →
  - → Find a sequence of nodes/edges connecting node A and node B
- To find the path in the graph, we need to "traverse" the graph, searching for the connection between A and B
- Graph traversal algorithms specify the order in which we search the nodes in the graph
  - Start at the source node (starting point)
  - The frontier of the exploration: nodes that we have seen but we have NOT explored fully yet
    - Exploring a node: seeing all its connected nodes
  - At each iteration of a graph traversal algorithm, we explore a node of the frontier: we take a node off the frontier and add its neighbors to the frontier



Breath-First Search

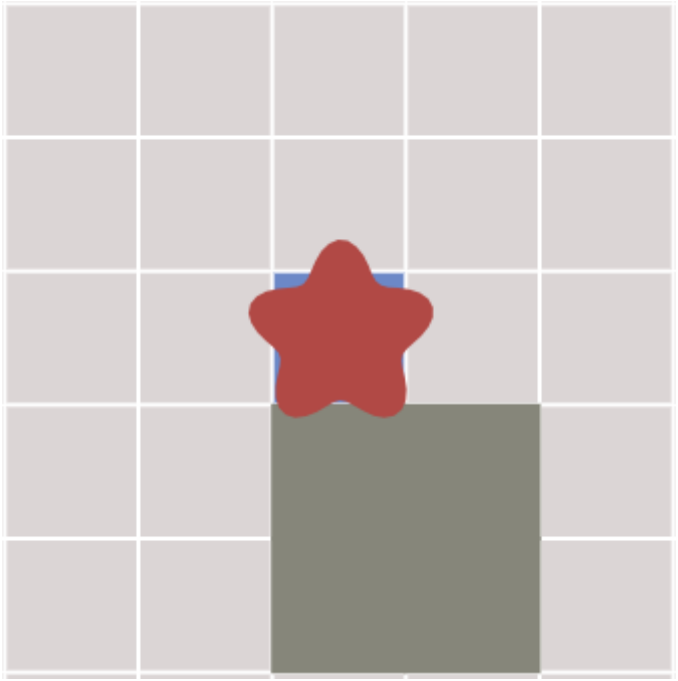
- We search first all the nodes connected to a same “level”
- We use a “queue” to control the order in which we search nodes
  - First in, first out (FIFO)
- BFS finds the shortest path



queue: first-in, first-out

Breath-First Search

```
frontier = Queue()  // FIFO
frontier.push(start)
parent = {}, parent[start] = Null
visited = {}
visited[start] = True

while not frontier.empty():
  current = frontier.get()
  if current == goal:
    break
  for next in neighbors(current):
    if visited[next]  = False:
      frontier.push(next)
      visited[next] = True
      parent[next] = current
```



Queue

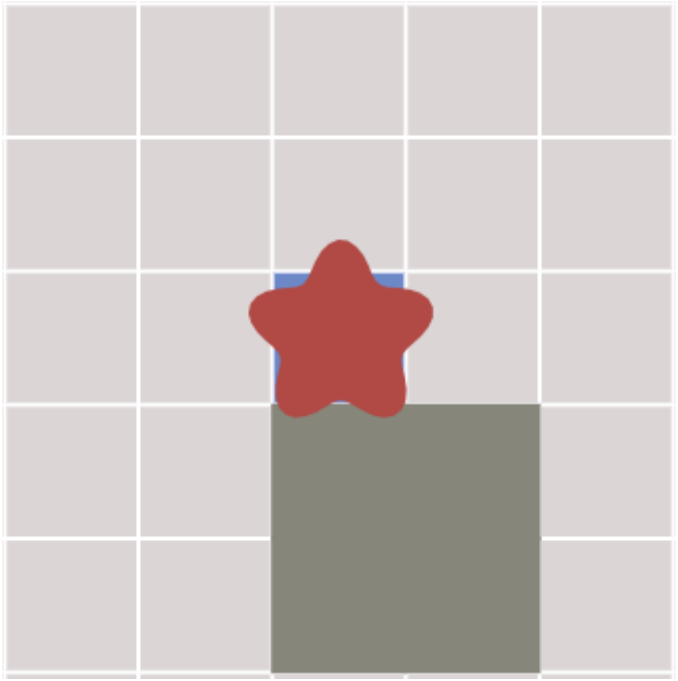
```
frontier = Queue()  // FIFO
frontier.put(start)
parent = {}, parent[start] = Null
visited = {}
visited[start] = True

while not frontier.empty():
  current = frontier.get()
  if current == goal:
    break
  for next in neighbors(current):
    if visited[next] == False:
      frontier.put(next)
      visited[next] = True
      parent[next] = current
```

Dictionary

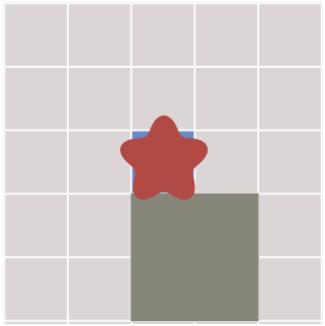
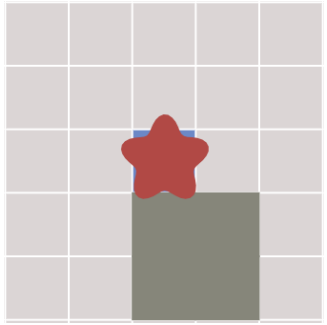
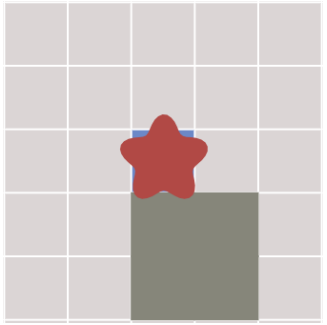
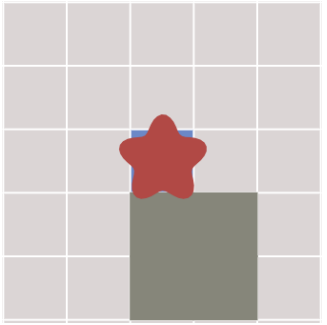
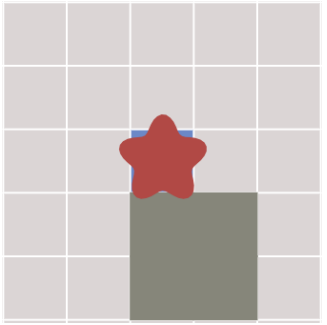
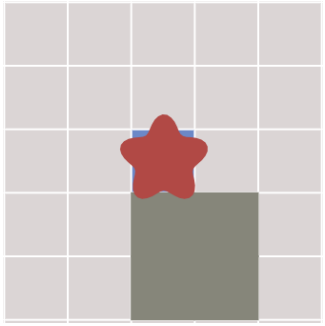
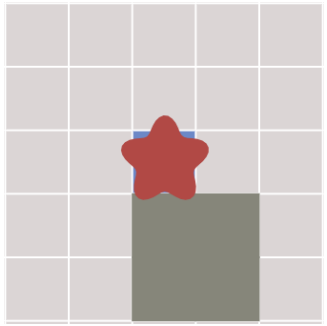
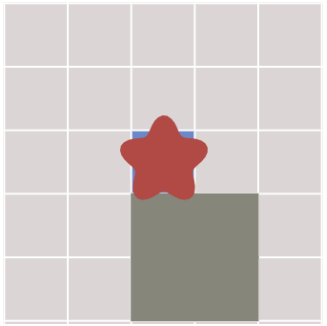
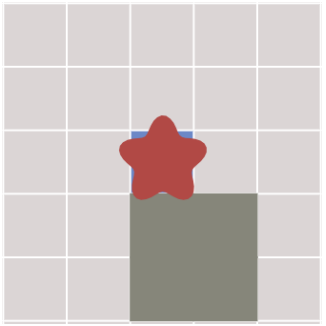
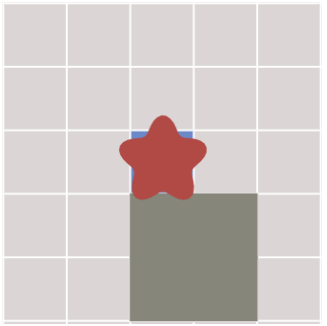
```
frontier = Queue()  // FIFO
frontier.put(start)
parent = {}, parent[start] = Null
visited = {}
visited[start] = True

while not frontier.empty():
  current = frontier.get()
  if current == goal:
    break
  for next in neighbors(current):
    if visited[next] == False:
      frontier.put(next)
      visited[next] = True
      parent[next] = current
```

Breath-First Search

```
frontier = Queue()  // FIFO
frontier.push(start)
parent = {}, parent[start] = Null
visited = {}
visited[start] = True

while not frontier.empty():
  current = frontier.get()
  if current == goal:
    break
  for next in neighbors(current):
    if visited[next]  = False:
      frontier.push(next)
      visited[next] = True
      parent[next] = current
```



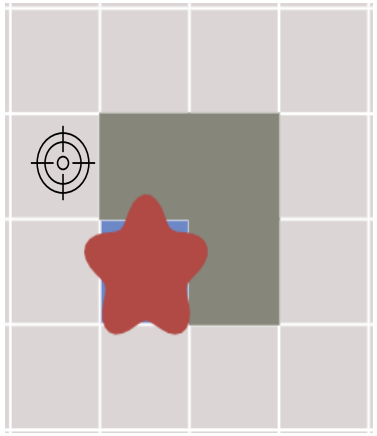
Breath-First Search

https://pollev.com/robertomartinmartin739

Exercise – Breath-First Search

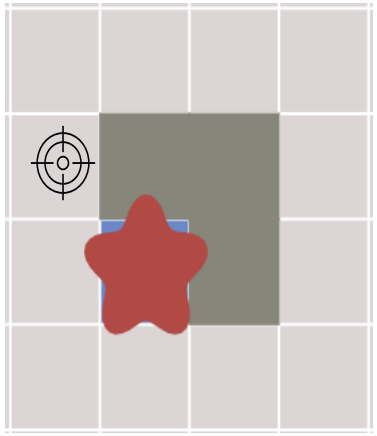
- Given the problem of the map
- Assume the robot can only move
  - up/down
  - left/right



https://pollev.com/robertomartinmartin739

Exercise – Breath-First Search

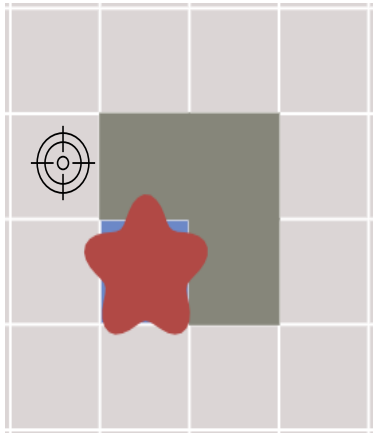
- Given the problem of the map
- Assume the robot can only move
  - up/down
  - left/right



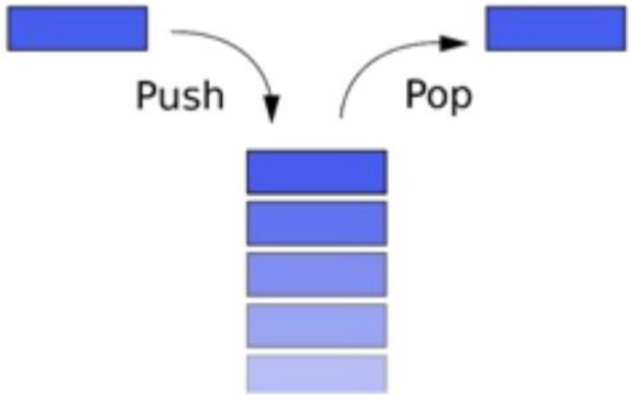
https://pollev.com/robertomartinmartin739

Exercise – Breath-First Search

- Given the problem of the map
- Assume the robot can only move
  - up/down
  - left/right



Depth-First Search

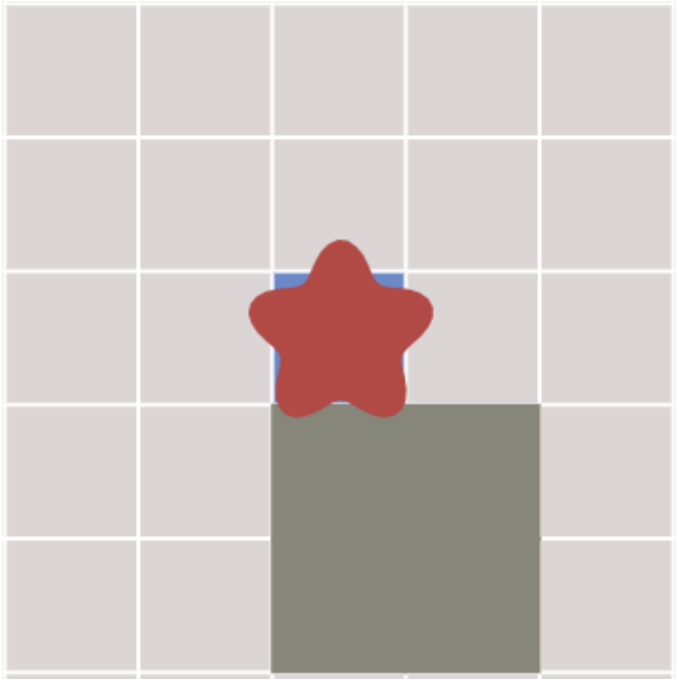


- We pick one direction and we go “deep” until we can’t go further
- We use a “stack” to control the order in which we search nodes
  - Last in, first out (LIFO)
- DFS finds a path

stack: last-in, first-out

Depth-First Search

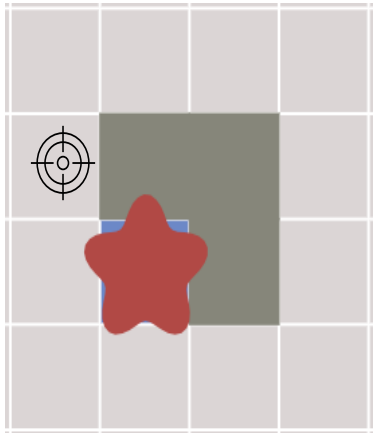
```
frontier = Stack()  // LIFO
frontier.push(start)
parent = {}, parent[start] = Null
visited = {}
visited[start] = True

while not frontier.empty():
  current = frontier.get()
  if current == goal:
    break
  for next in neighbors(current):
    if visited[next]  = False:
      frontier.put(next)
      visited[next] = True
      parent[next] = current
```



https://pollev.com/robertomartinmartin739

Exercise – Depth-First Search

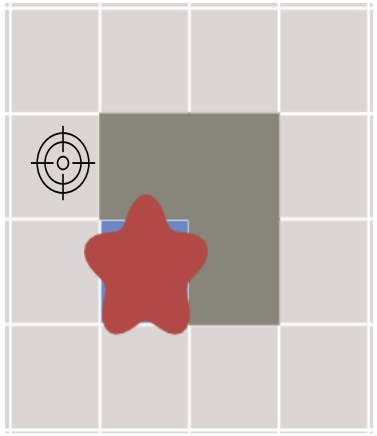
- Given the problem of the map
- Assume the robot can only move
  - up/down
  - left/right



https://pollev.com/robertomartinmartin739

Exercise – Depth-First Search

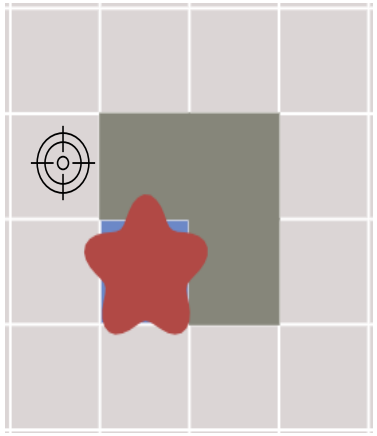
- Given the problem of the map
- Assume the robot can only move
  - up/down
  - left/right



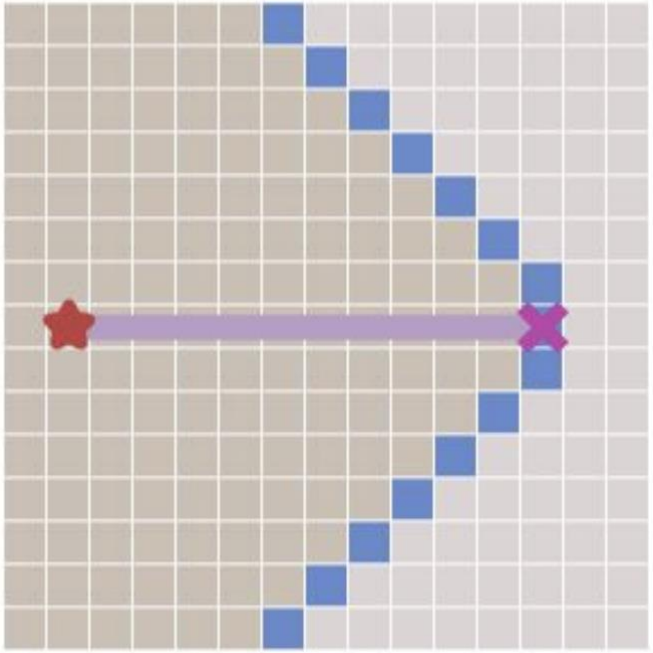
https://pollev.com/robertomartinmartin739

Exercise – Depth-First Search

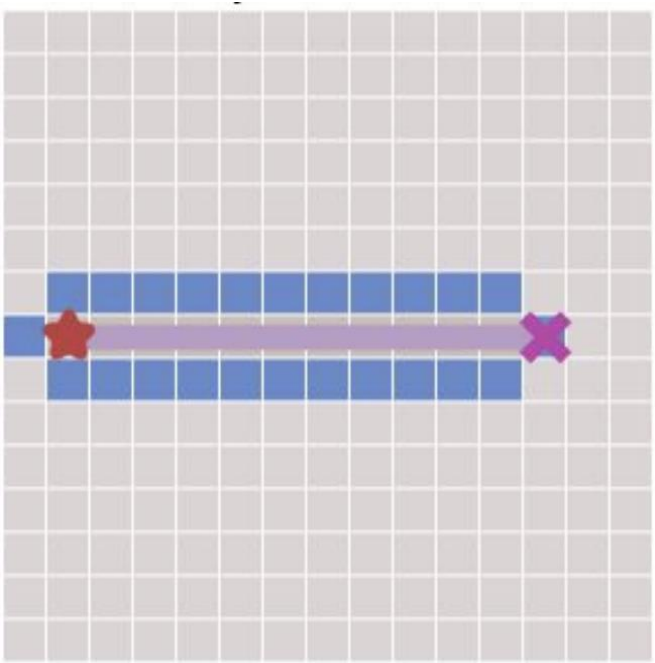
- Given the problem of the map
- Assume the robot can only move
  - up/down
  - left/right



Greedy Best First Search



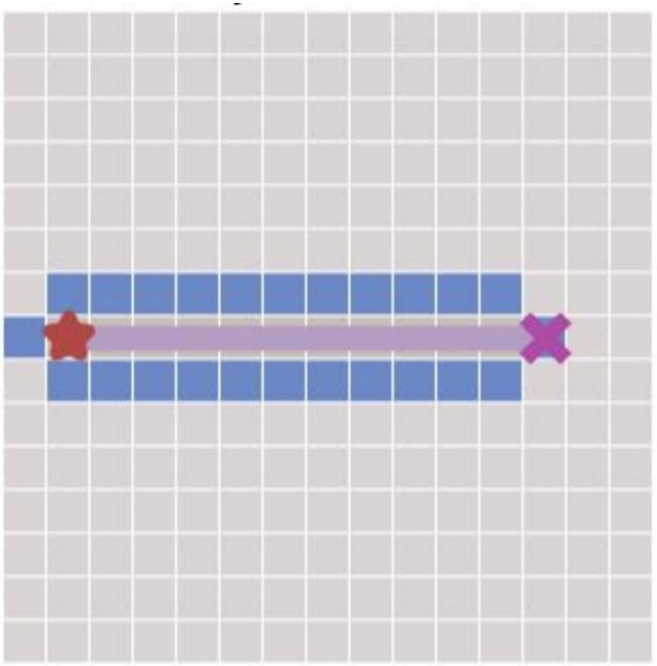
BFS



Greedy First Search

Greedy Best First Search

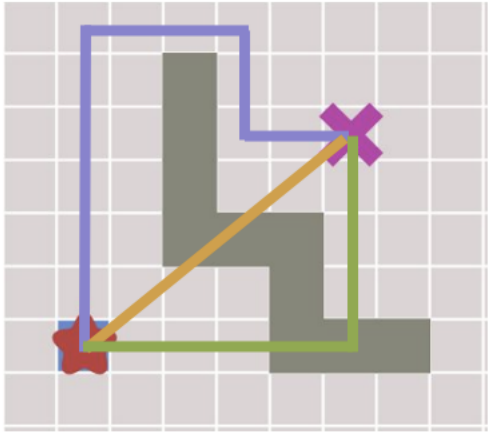
- At every iteration, Greedy Best First moves “in the direction of the target”
  - Tries to “shorten” the distance
- It is “greedy” because it takes the best-looking direction at each step
- Greedy Best First picks the "best" node according to, some rule of thumb, called a heuristic
  - A heuristic gives AN APPROXIMATE measure of the distance to the goal



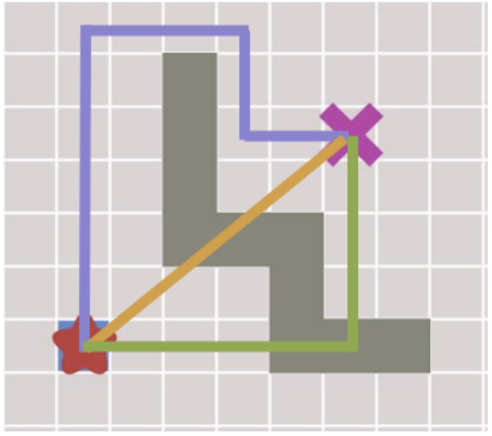
Greedy First Search

Greedy Best First Search - Heuristics

- Heuristic should be easy to compute at each step
- Candidates
  - L2 distance
  - Manhattan distance



Greedy Best First Search - Heuristics

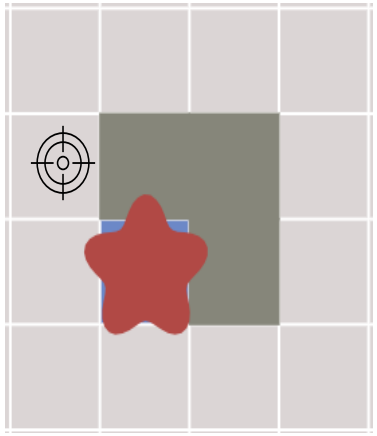


- Heuristic should be easy to compute at each step
- Candidates
  - L2/Euclidean distance
  - Manhattan distance

https://pollev.com/robertomartinmartin739

Exercise – Greedy-First Search

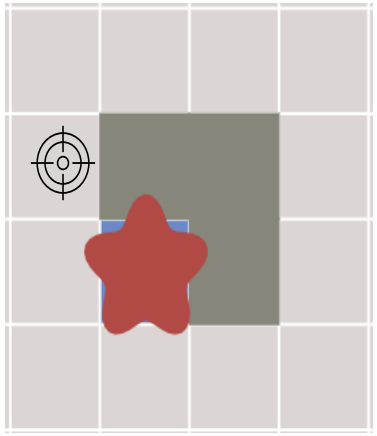
- Given the problem of the map
- Assume the robot can only move
  - up/down
  - left/right
- Euclidean distance



https://pollev.com/robertomartinmartin739

Exercise – Greedy-First Search

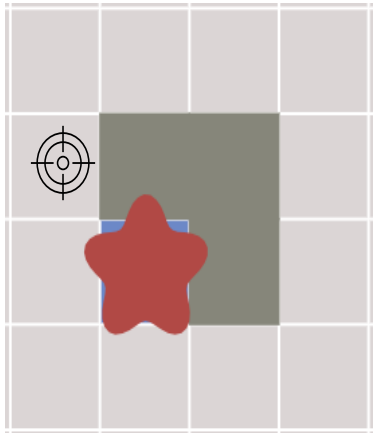
- Given the problem of the map
- Assume the robot can only move
  - up/down
  - left/right



https://pollev.com/robertomartinmartin739

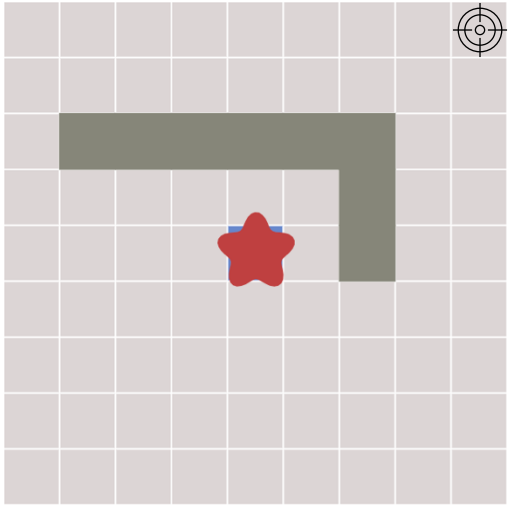
Exercise – Greedy-First Search

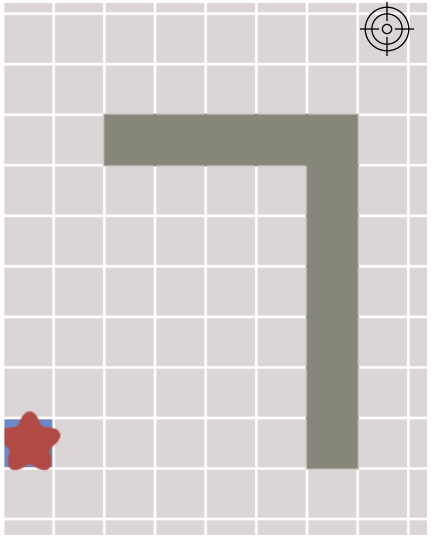
- Given the problem of the map
- Assume the robot can only move
  - up/down
  - left/right



Exercise – Optimality

- Optimal algorithm:
  - If the algorithm finds a path, the path found is ALWAYS the shortest





A*

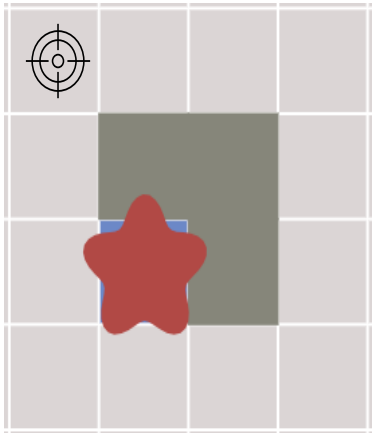
- Combines the strengths of BFS and GFS
  - Optimal: finds the shortest path
  - Fast
- At each iteration, A chooses to explore the node of the frontier that minimizes:

N = steps from source to node + approximate steps to target

https://pollev.com/robertomartinmartin739

Exercise – A* Search

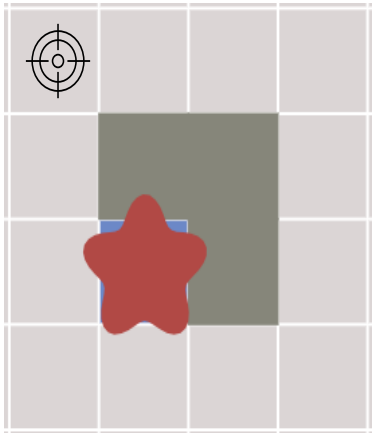
- Given the problem of the map
- Assume the robot can only move
  - up/down
  - left/right
- Euclidean distance



https://pollev.com/robertomartinmartin739

Exercise – A* Search

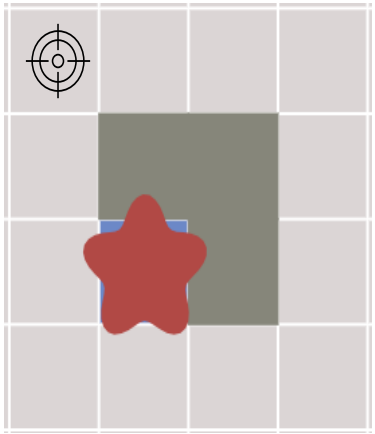
- Given the problem of the map
- Assume the robot can only move
  - up/down
  - left/right
- Euclidean distance



https://pollev.com/robertomartinmartin739

Exercise – A* Search

- Given the problem of the map
- Assume the robot can only move
  - up/down
  - left/right
- Euclidean distance



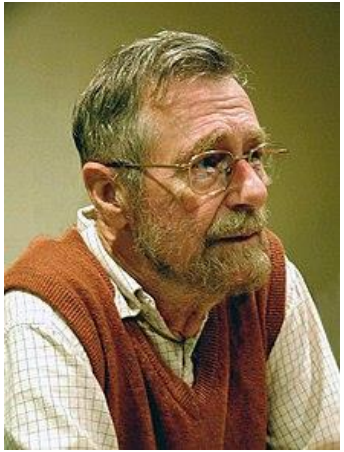
Dijkstra's Algorithm

- So far, traversing from one node to another (the cost of one step, the cost of one edge) was always =1
- In some problems, moving between two nodes can have different costs
- Our graph becomes a weighted graph
- We want to find the shortest path: the path where the sum of costs over transitions is the smallest

A weighted graph in navigation

Dijkstra's Algorithm

- Like BFS for weighted graphs
  - If all edge costs are 1, Dijkstra=BFS
- Explore nodes in increasing order of cost from source



Edsger W. Dijkstra
1930-2002

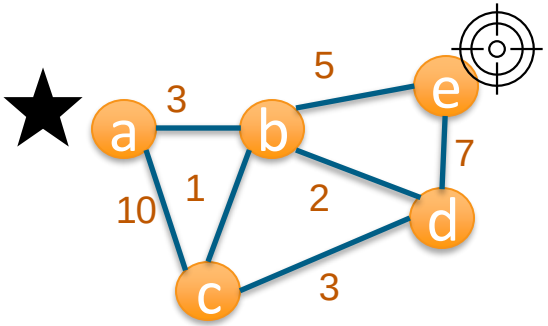
Dijkstra's Algorithm

```
frontier = PriorityQueue()  // Note priority!
frontier.push(start, 0)     // 0 = cost to go
parent = {}, parent[start] = Null
cost = {}
cost[start] = 0             // Cost from start

while not frontier.empty():
  current = frontier.get()  // Get by priority!
  if current == goal:
    break
  for next in neighbors(current):
    new_cost = cost[current] + EdgeCost[current, next]
    if next not in cost or new_cost < cost[next]:
      cost[next] = new_cost  // Insertion or edit
      frontier.put(next, new_cost)
      parent[next] = current
```

https://pollev.com/robertomartinmartin739

Exercise – Dijkstra's

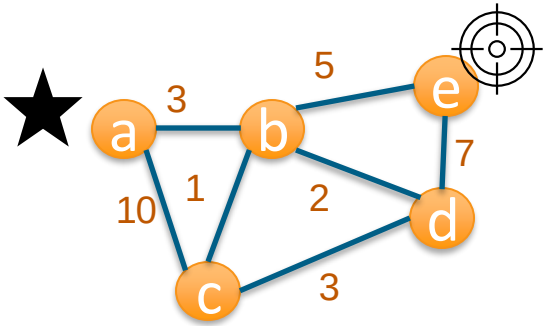
- Given the problem of the graph



https://pollev.com/robertomartinmartin739

Exercise – Dijkstra's

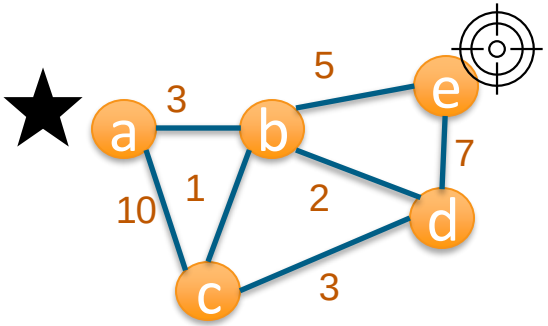
- Given the problem of the graph



https://pollev.com/robertomartinmartin739

Exercise – Dijkstra's

- Given the problem of the graph



Back to our robot