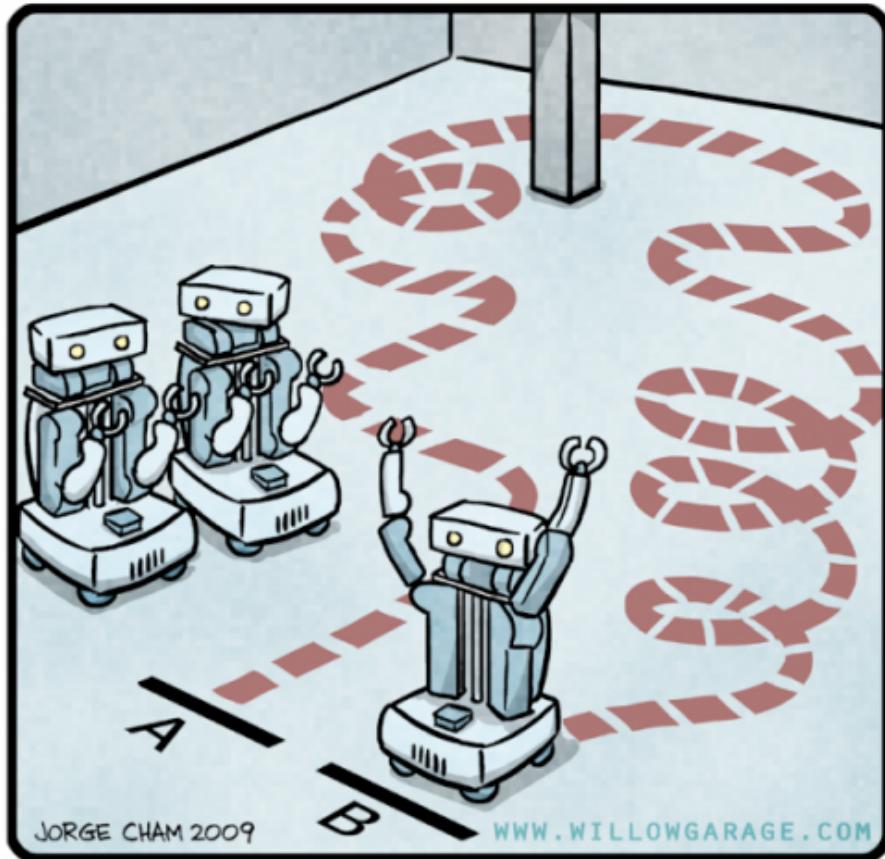


R.O.B.O.T. Comics



"HIS PATH-PLANNING MAY BE
SUB-OPTIMAL, BUT IT'S GOT FLAIR."

Path planning

the best way to get from A to B

EECS 398
Intro. to Autonomous Robotics

ME/EECS 567 ROB 510
Robot Modeling and Control

Fall 2018

autorob.org

Administrivia

- Assignment 1 (Path Planning) has been released
 - due 1:20pm, Monday, September 24, 2018
 - Send me your email, picture, and pointer to git repository
- Anyone still need permission to move from waitlist?
 - Please see me if you are not added by the end of Wednesday's lecture
- Did you see May Mobility on the TODAY show this morning!?

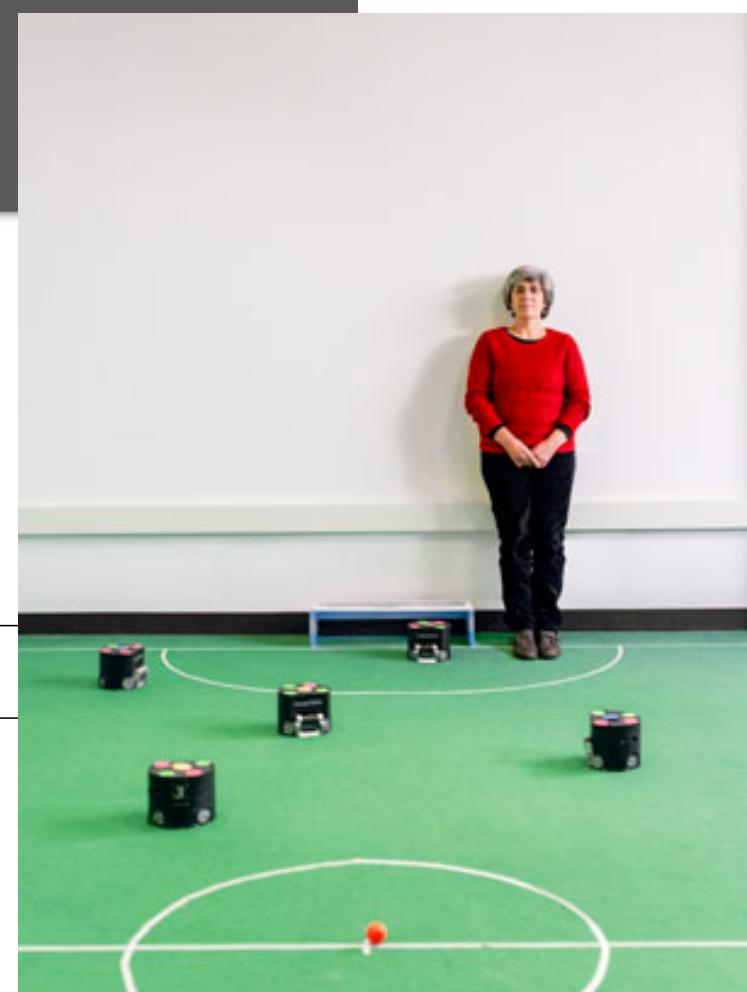
Feature | Geek Life | Profiles

Manuela Veloso: RoboCup's Champion

This roboticist has transformed robot soccer into a global phenomenon

Posted 28 Feb 2015 | 17:00 GMT

By [PRACHI PATEL](#)



CMDragons'06

Carnegie Mellon



CMDragons
RoboCup Small
2006

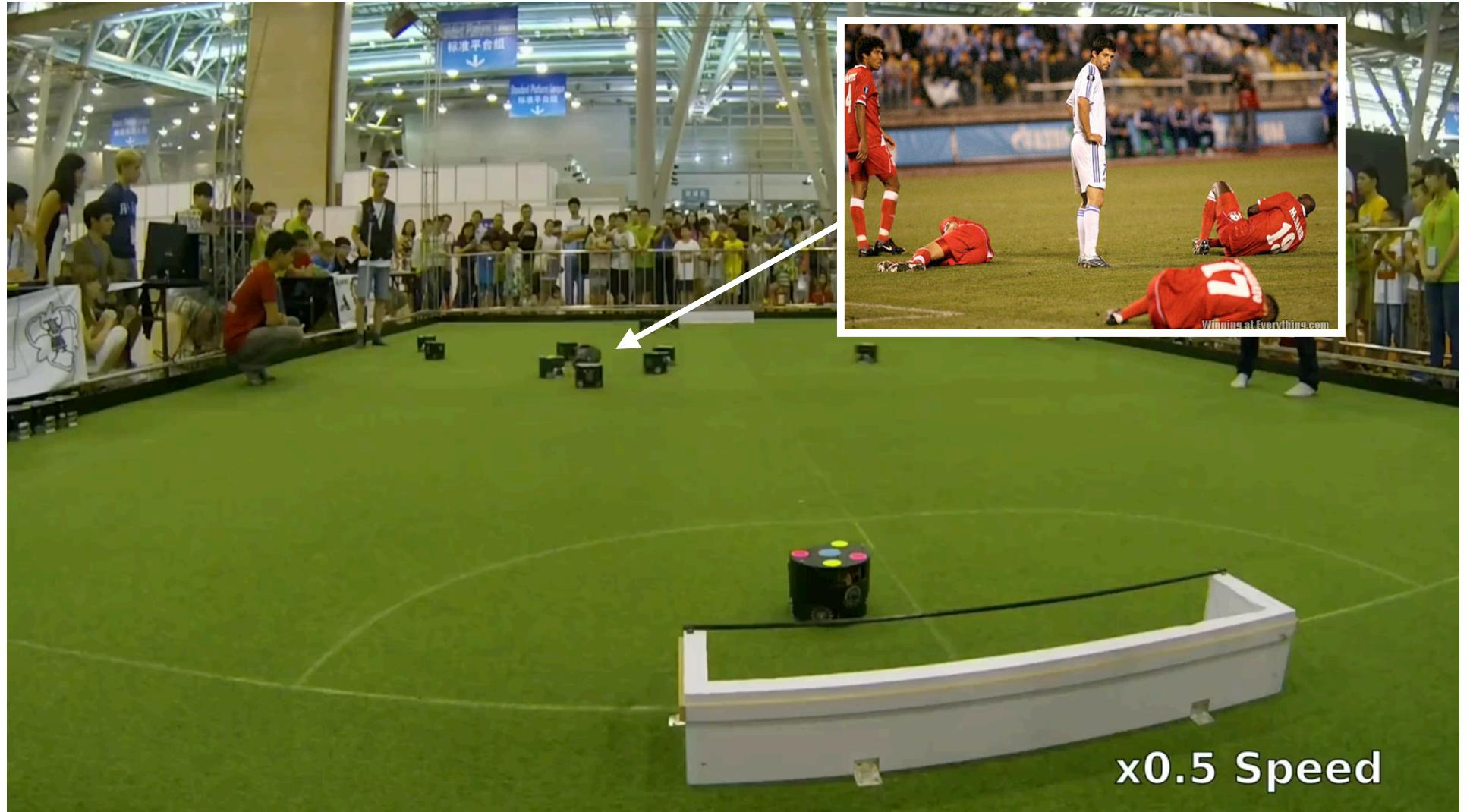


CMDragons 2016 Pass-ahead Goal



x0.5 Speed

CMDragons 2016 slow-motion multi-pass goal



x0.5 Speed

CMDragons 2016 slow-motion multi-pass goal

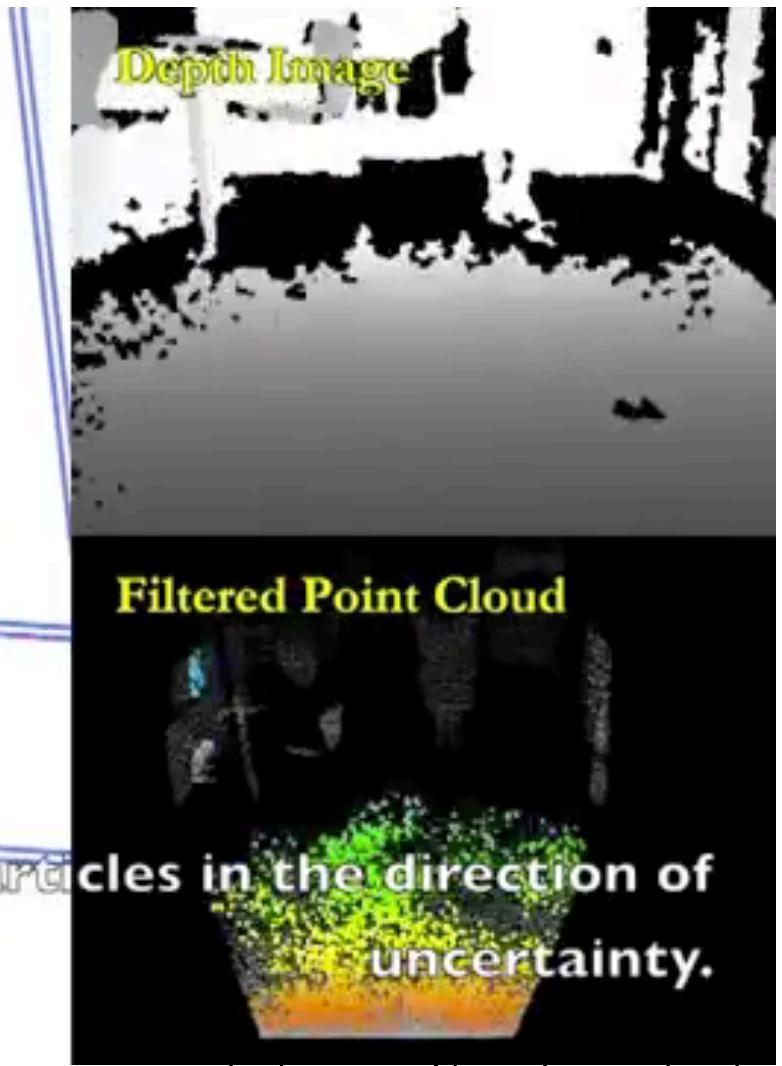


<http://www.cs.cmu.edu/~coral/projects/cobot/>



Currently, three CoBots can navigate the building.

<http://www.cs.cmu.edu/~coral/projects/cobot/>



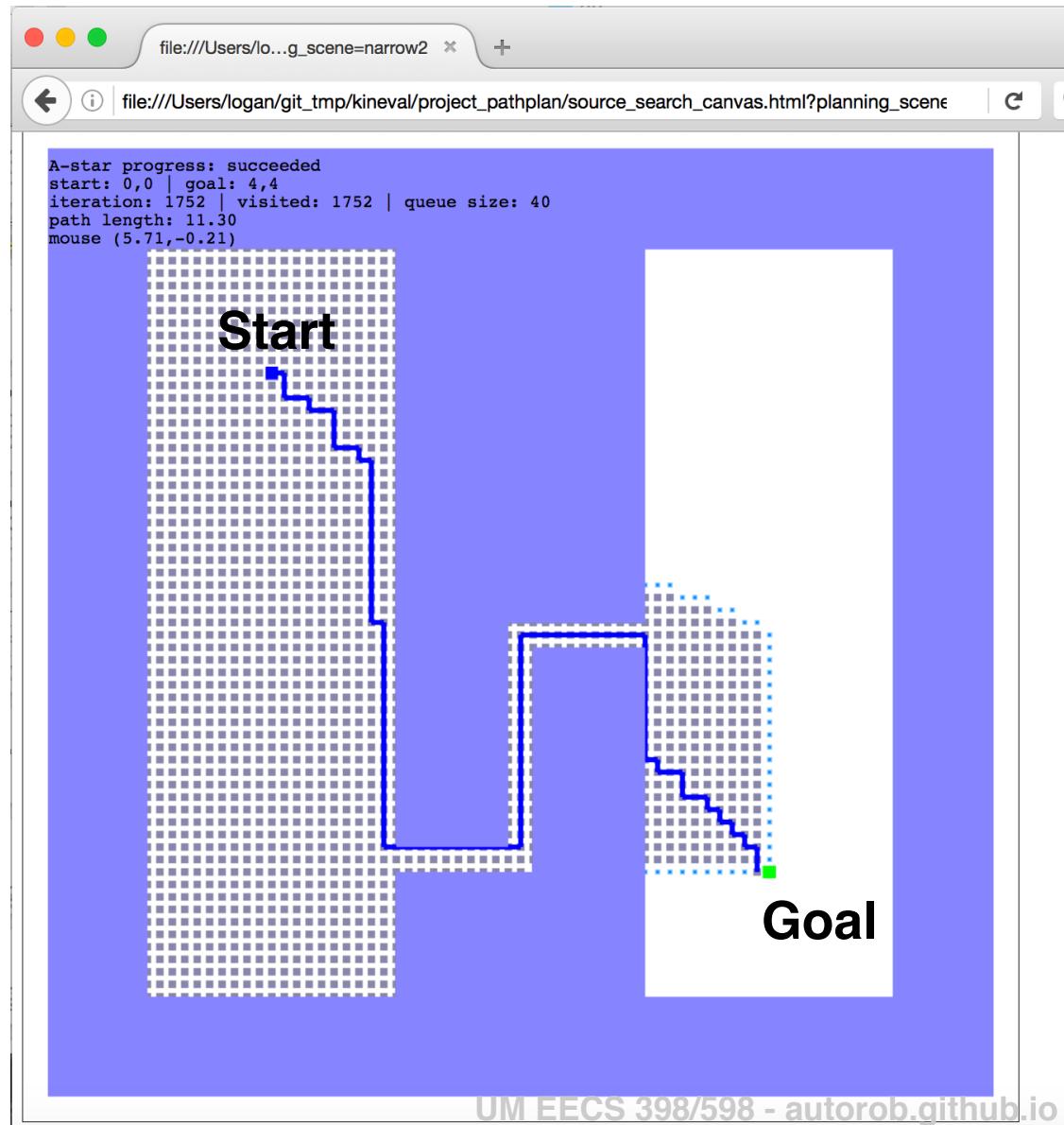
<http://www.cs.cmu.edu/~coral/projects/cobot/>



CoBot greets me in 2013

Project 1: 2D Path Planning

- A-star algorithm for search in a given 2D world
- Implement in JavaScript/HTML5
- Heap data structure for priority queue
- Grads: DFS, BFS, Greedy
- Submit through your git repository



2007-10: SOCCER WITH iROBOT CREATE



2007 - AR Tags for overhead localization

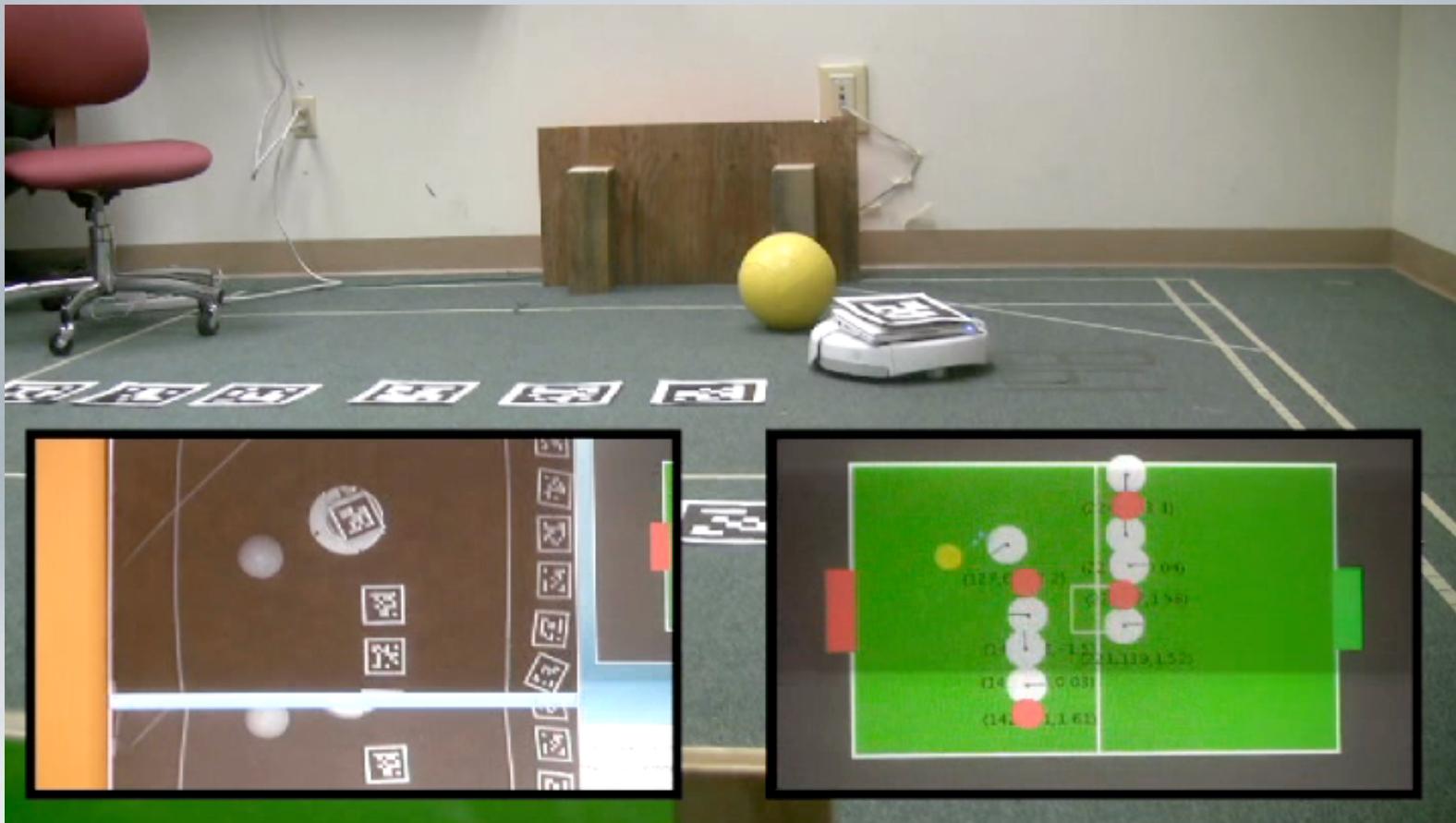


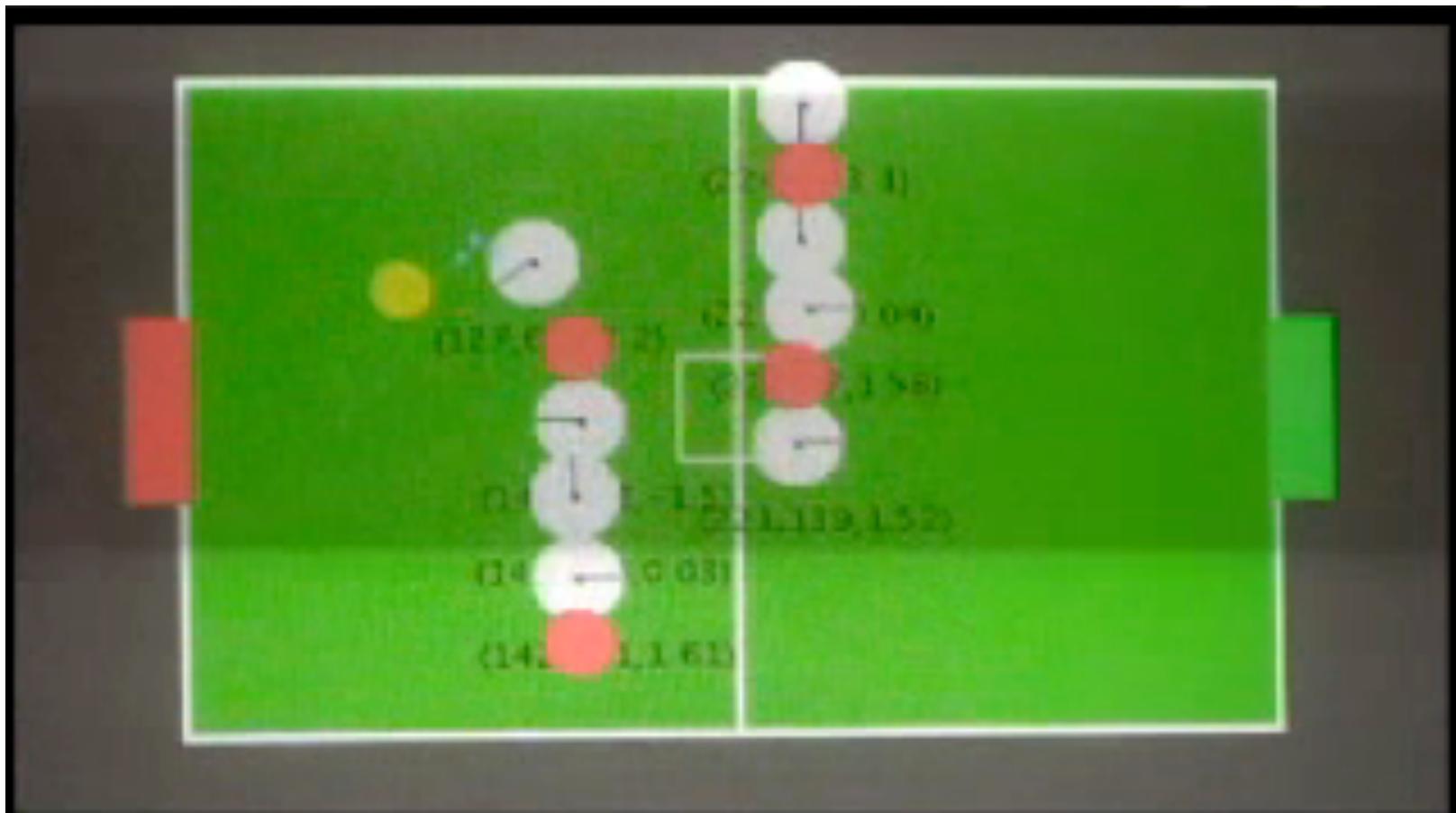
2007 - Mini ITX



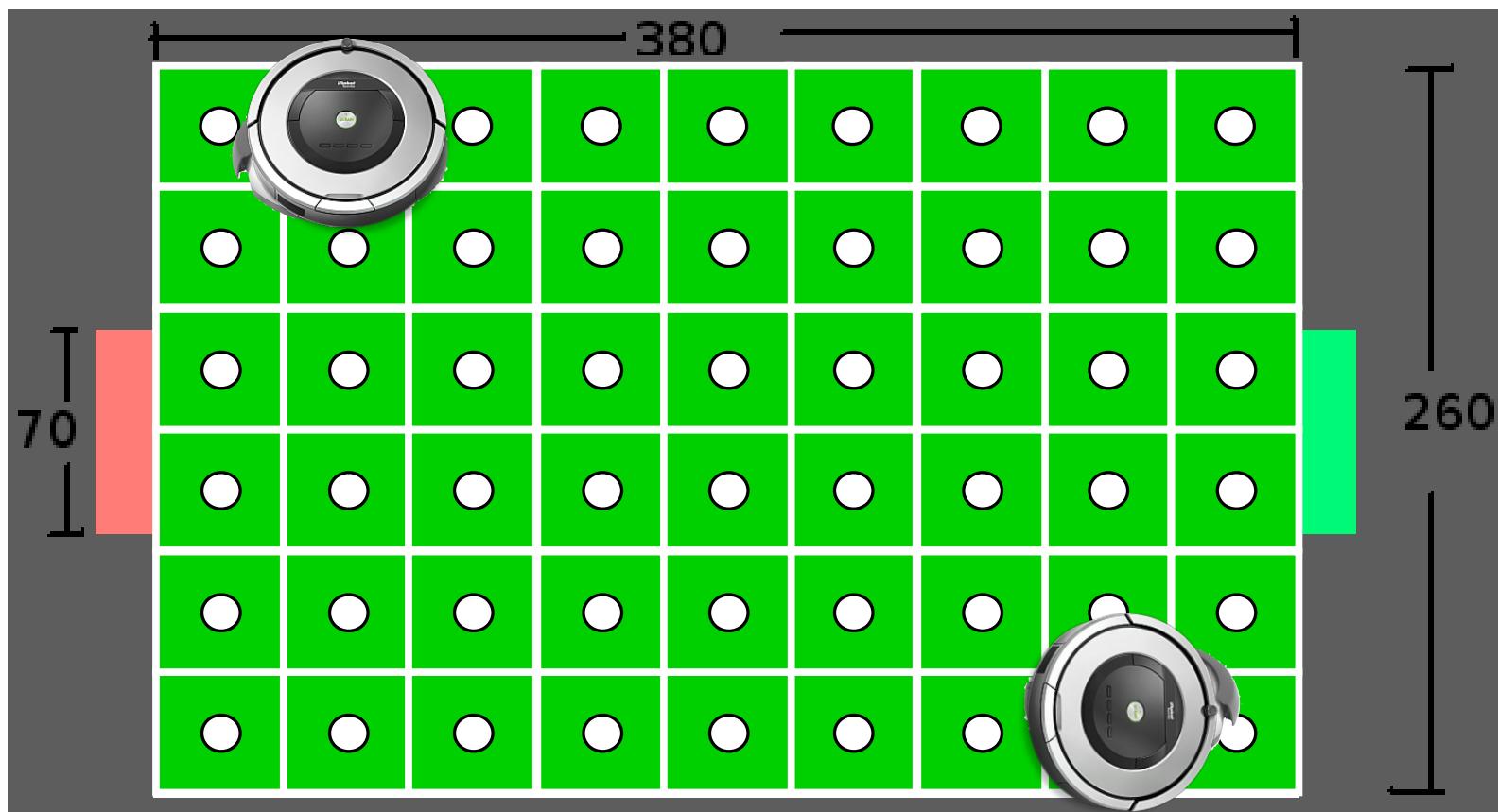
2009 - Asus EEE

<http://www.youtube.com/watch?v=88zR6IC7S0g>

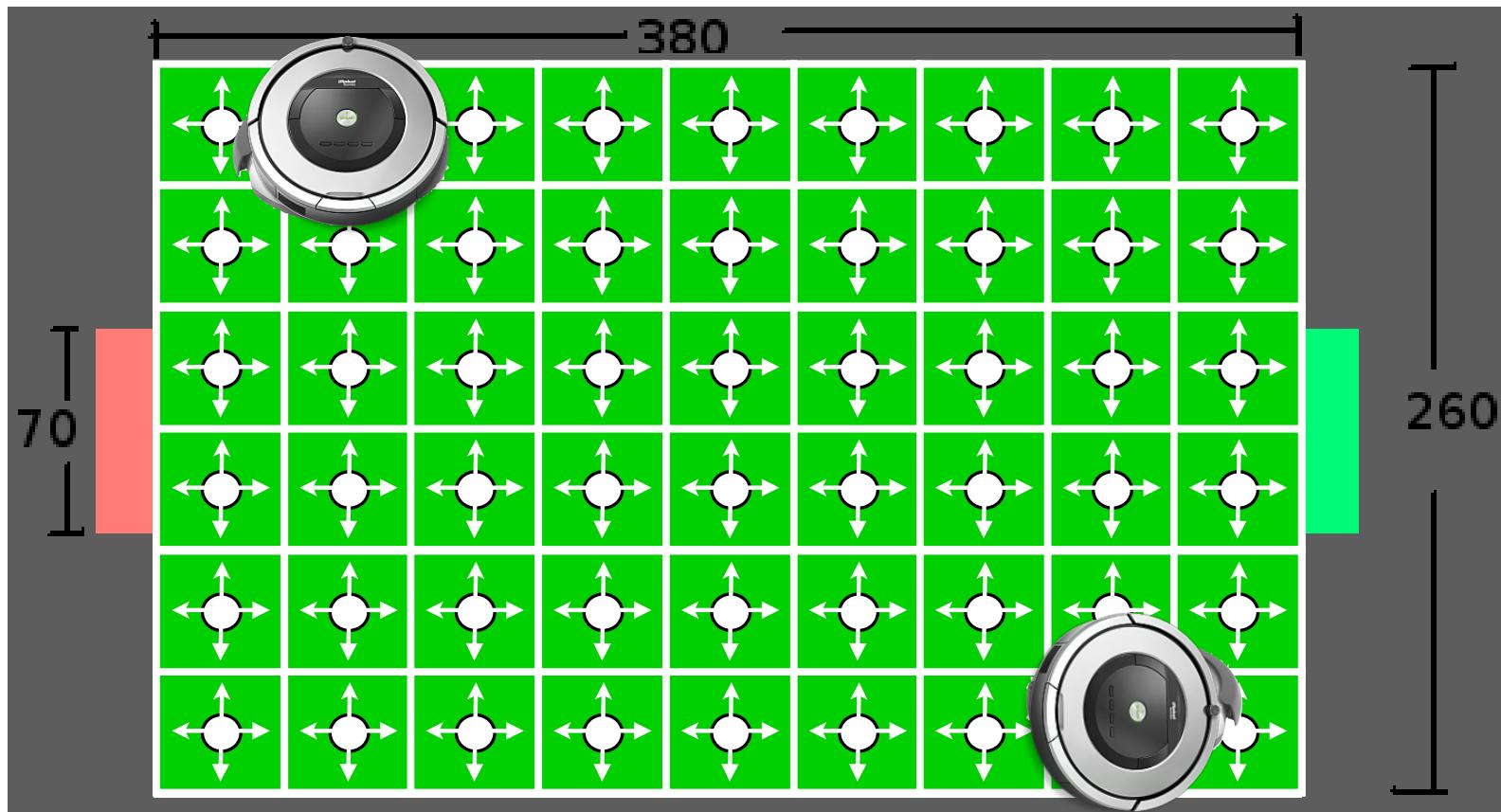




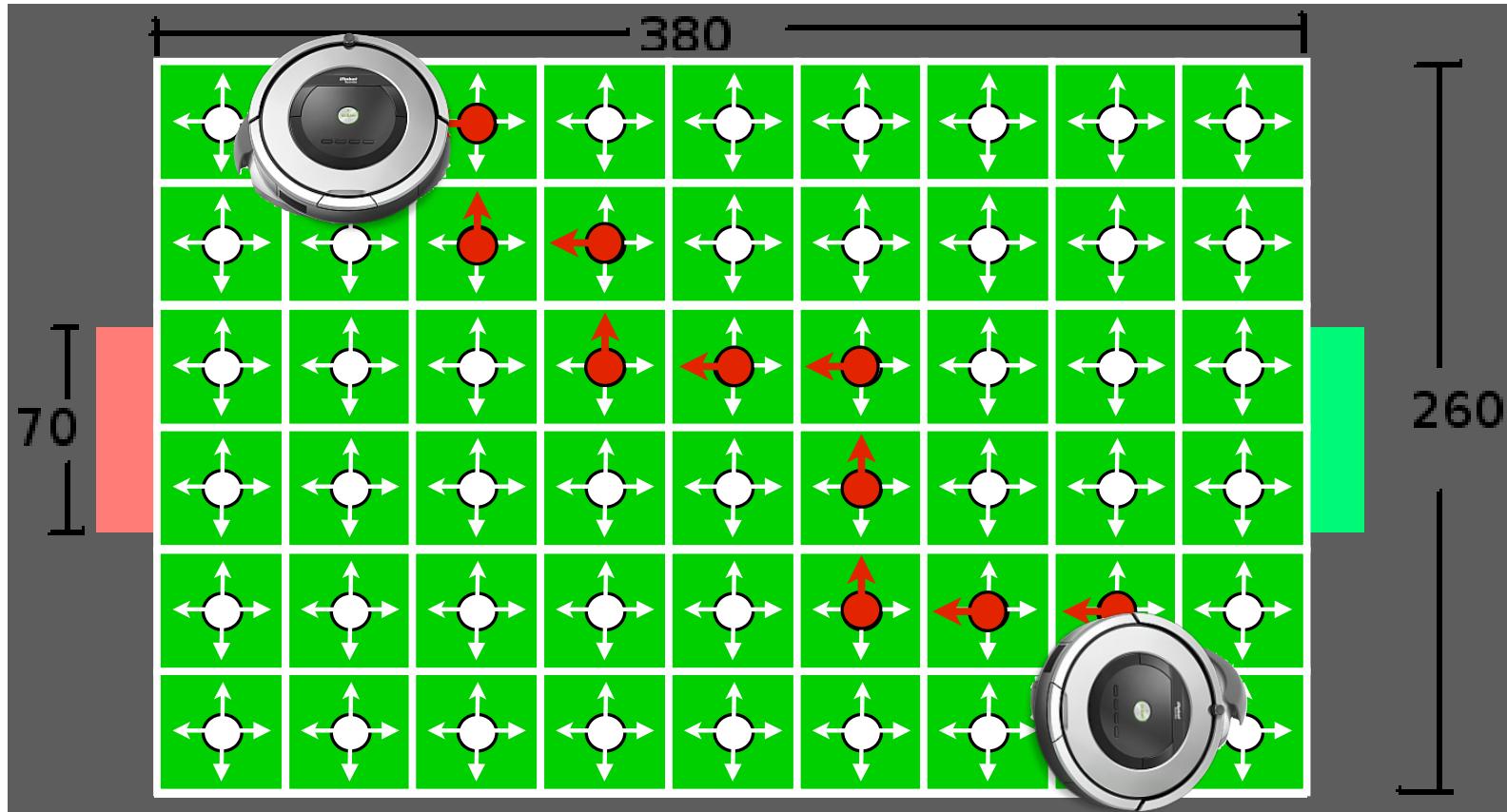
Consider all possible poses as uniformly distributed graph vertices



Consider all possible poses as uniformly distributed graph vertices
Edges connect adjacent (traversable) poses, weighted by distance



Consider all possible poses as uniformly distributed graph vertices
Edges connect adjacent (traversable) poses, weighted by distance
How to find a valid path in this graph?

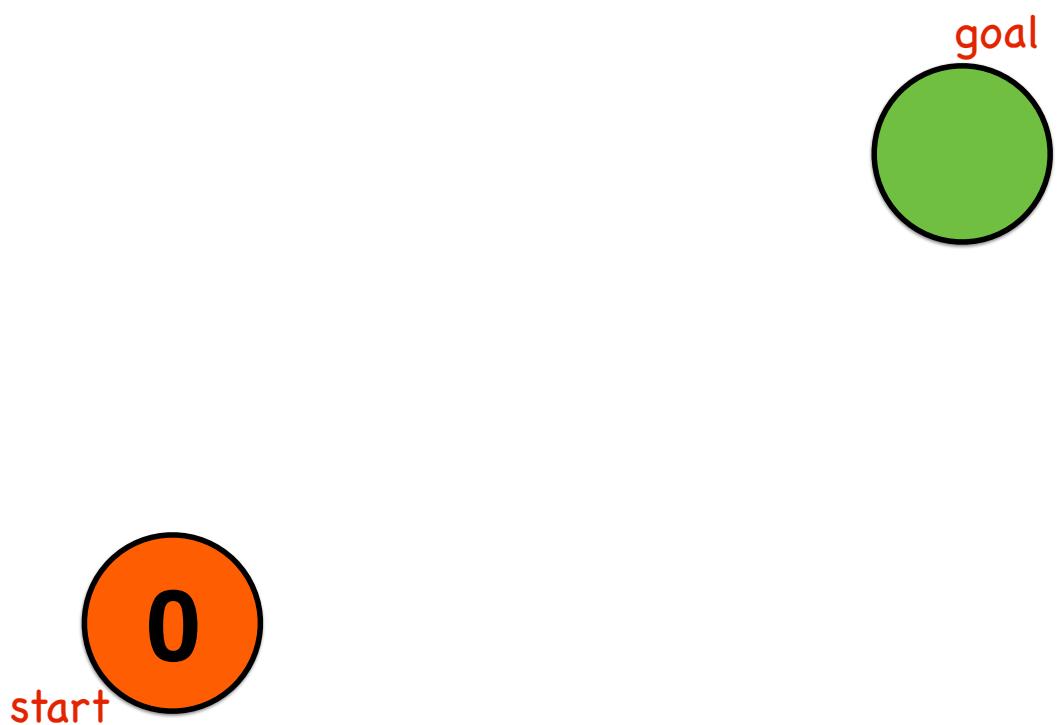


Approaches to motion planning

- Bug algorithms: Bug[0-2], Tangent Bug
- **Graph Search (fixed graph)**
 - **Depth-first, Breadth-first, Dijkstra, A-star, Greedy best-first**
- Sampling-based Search (build graph):
 - Probabilistic Road Maps, Rapidly-exploring Random Trees
- Optimization (local search):
 - Gradient descent, potential fields, Wavefront

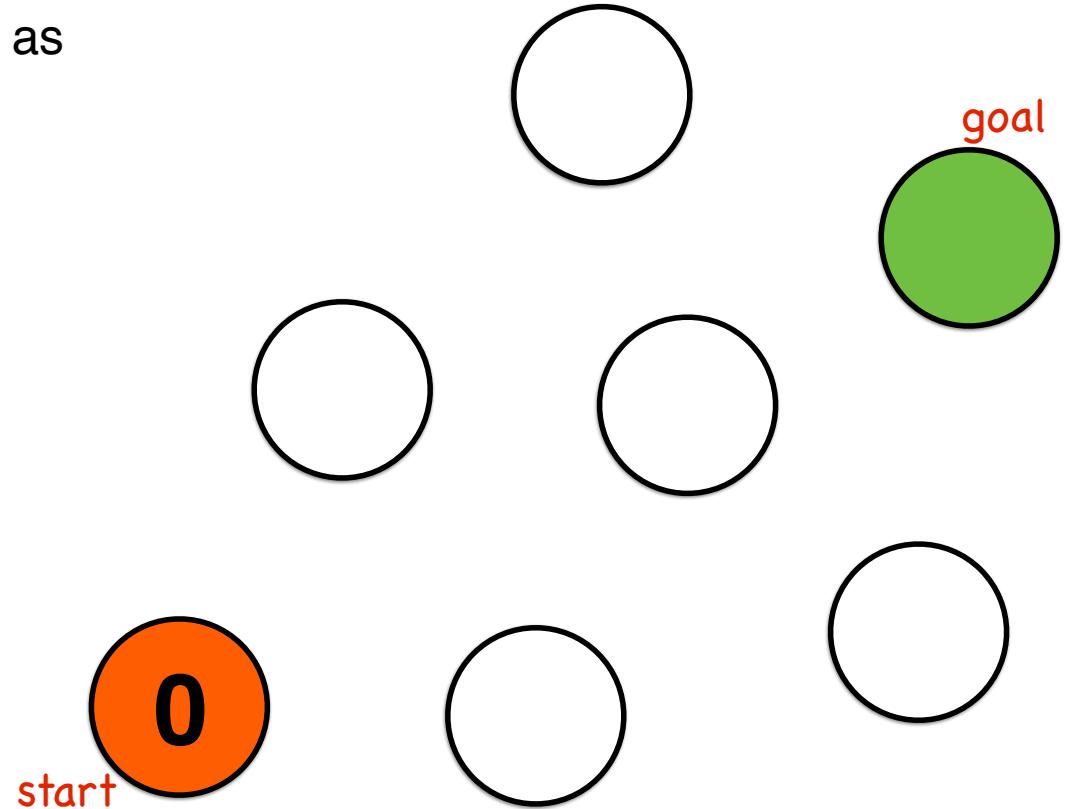
Consider a simple search graph

Consider each possible robot pose as node in a graph



Consider a simple search graph

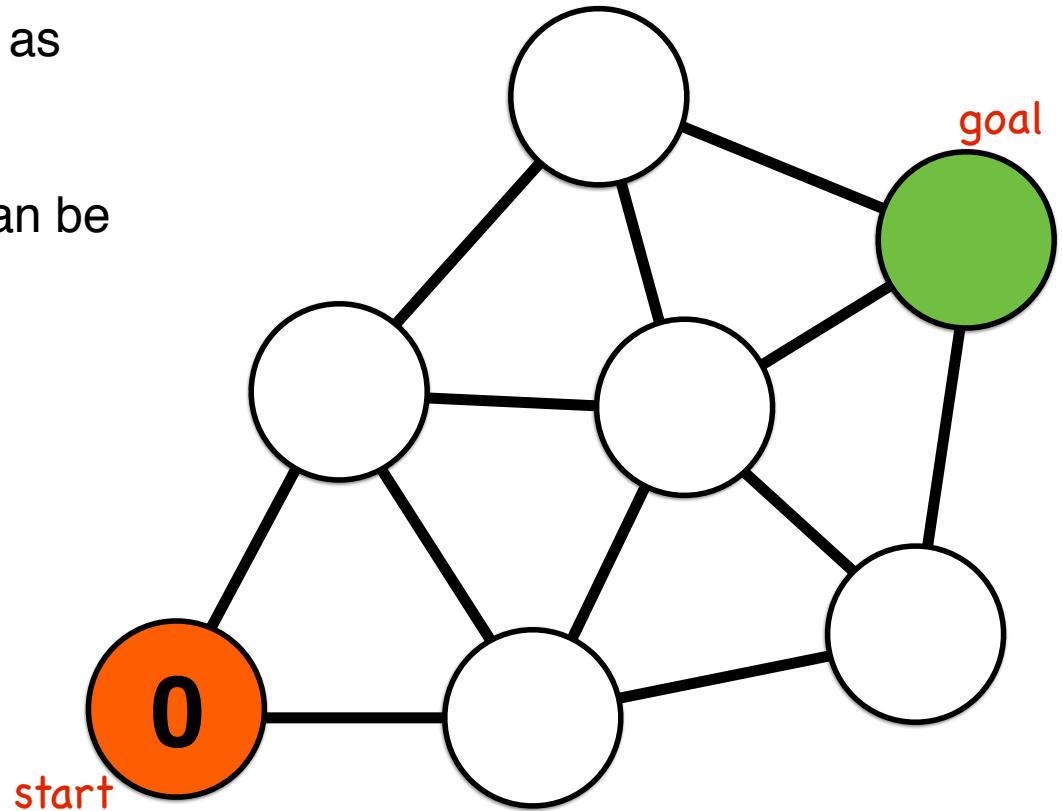
Consider each possible robot pose as node in a graph



Consider a simple search graph

Consider each possible robot pose as node in a graph

Graph edges connect poses that can be reliably moved between

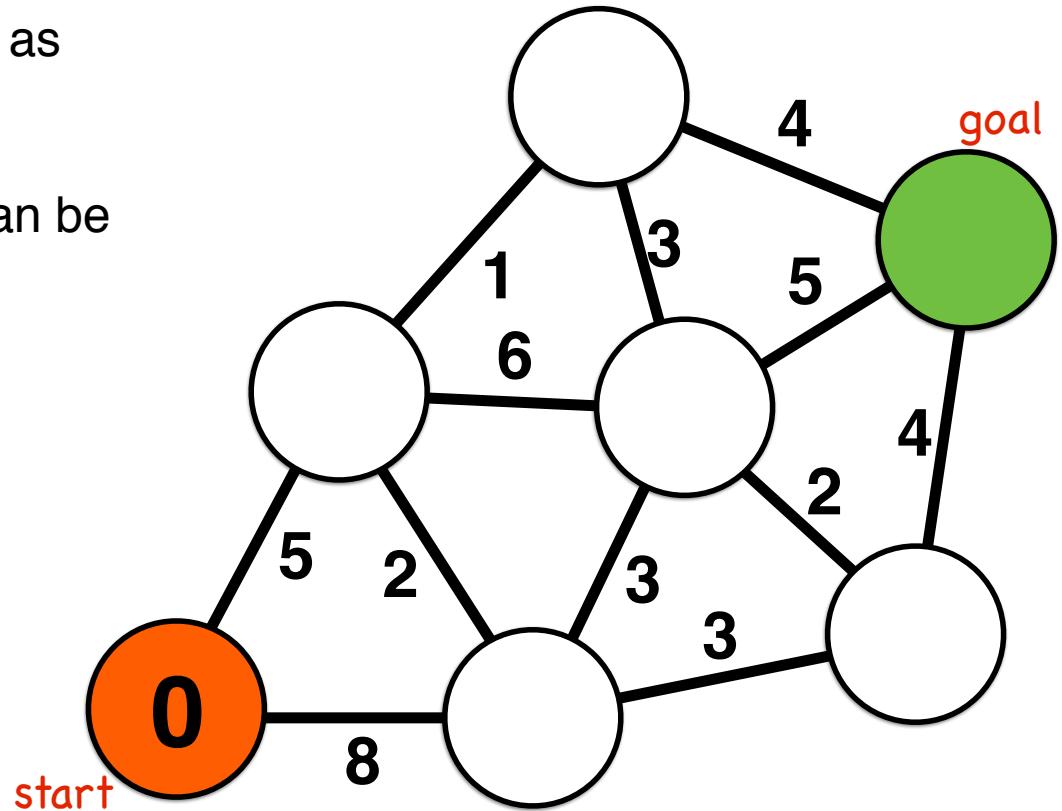


Consider a simple search graph

Consider each possible robot pose as node in a graph

Graph edges connect poses that can be reliably moved between

Edges have a cost for traversal



Consider a simple search graph

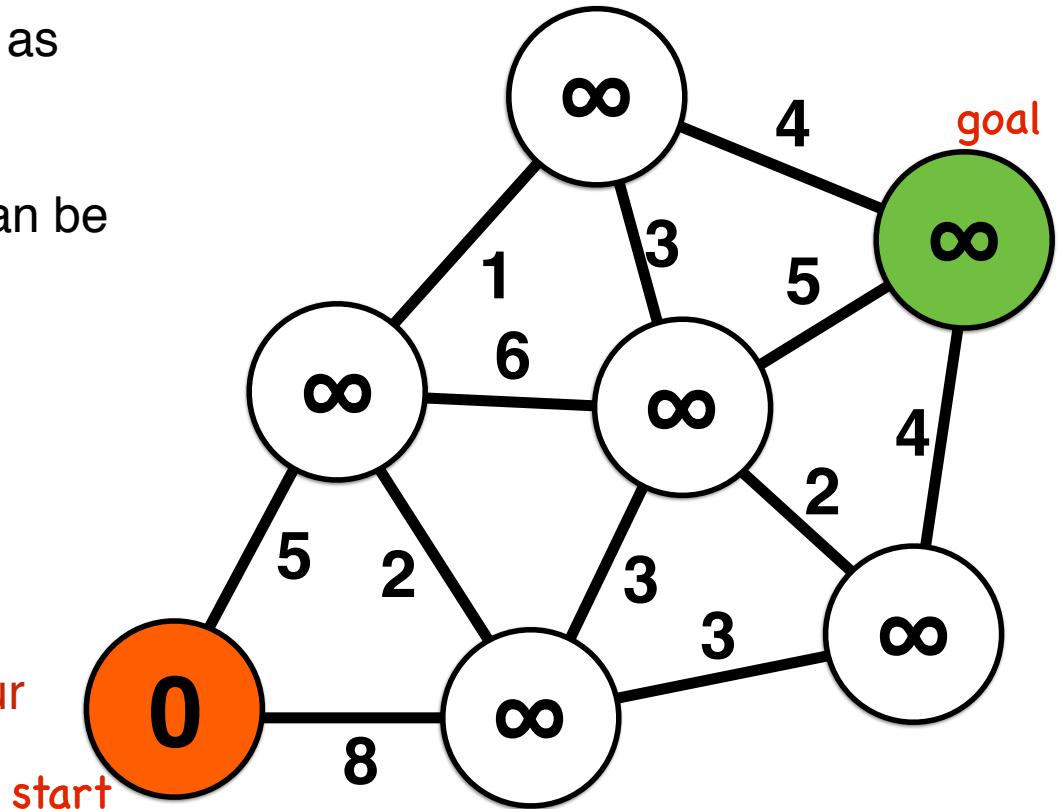
Consider each possible robot pose as node in a graph

Graph edges connect poses that can be reliably moved between

Edges have a cost for traversal

Each node maintains the distance traveled as a scalar cost

We will use a single template for our graph search algorithms



Search algorithm template

```
all nodes  $\leftarrow \{\text{dist}_{\text{start}} \leftarrow \text{infinity}, \text{parent}_{\text{start}} \leftarrow \text{none}, \text{visited}_{\text{start}} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{\text{dist}_{\text{start}} \leftarrow 0, \text{parent}_{\text{start}} \leftarrow \text{none}, \text{visited}_{\text{start}} \leftarrow \text{true}\}$ 
visit_list  $\leftarrow \text{start\_node}$ 
```

```
while visit_list != empty && current_node != goal
```

```
    cur_node  $\leftarrow \text{highestPriority}(\text{visit\_list})$ 
```

```
    visitedcur_node  $\leftarrow \text{true}$ 
```

```
    for each nbr in not_visited(adjacent(cur_node))
```

```
        add(nbr to visit_list)
```

```
        if distnbr > distcur_node + distStraightLine(nbr,cur_node)
```

```
            parentnbr  $\leftarrow \text{current\_node}$ 
```

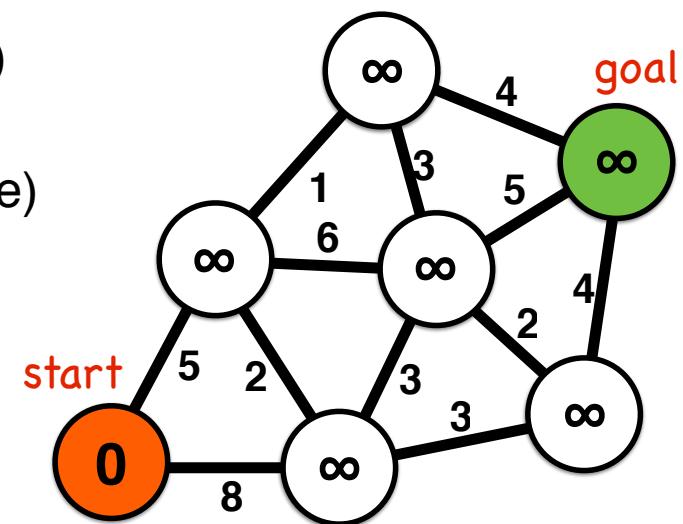
```
            distnbr  $\leftarrow \text{dist}_{\text{cur\_node}} + \text{distStraightLine}(\text{nbr}, \text{cur\_node})$ 
```

```
        end if
```

```
    end for loop
```

```
    end while loop
```

```
output  $\leftarrow \text{parent, distance}$ 
```



Search algorithm template

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited}_{start} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited}_{start} \leftarrow \text{true}\}$ 
visit_list  $\leftarrow \text{start\_node}$ 
```

Initialization

- each node has a distance and a parent
 - distance: distance along route from start
 - parent: routing from node to start
- visit a chosen start node first
- all other nodes are unvisited and have high distance

```
parentnbr  $\leftarrow \text{current\_node}$ 
```

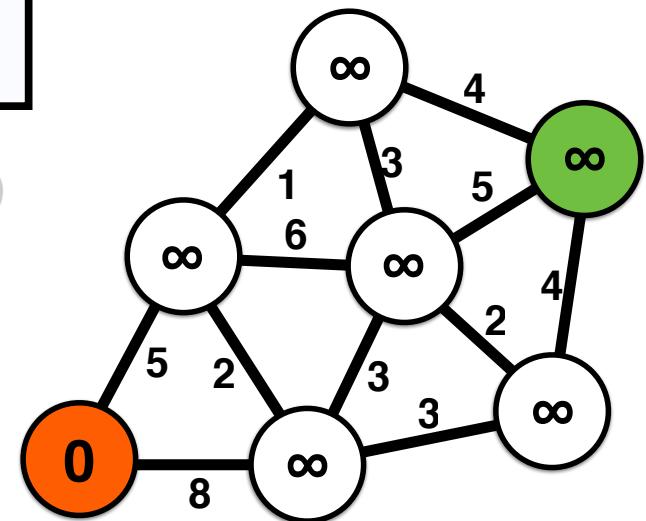
```
distnbr  $\leftarrow dist_{cur\_node} + distStraightLine(nbr, cur\_node)$ 
```

```
end if
```

```
end for loop
```

```
end while loop
```

```
output  $\leftarrow \text{parent, distance}$ 
```



Search algorithm template

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
visit_list  $\leftarrow \text{start\_node}$ 
```

```
while visit_list != empty && current_node != goal
    cur_node  $\leftarrow \text{highestPriority(visit\_list)}$ 
    visitedcur_node  $\leftarrow \text{true}$ 
```

Main Loop

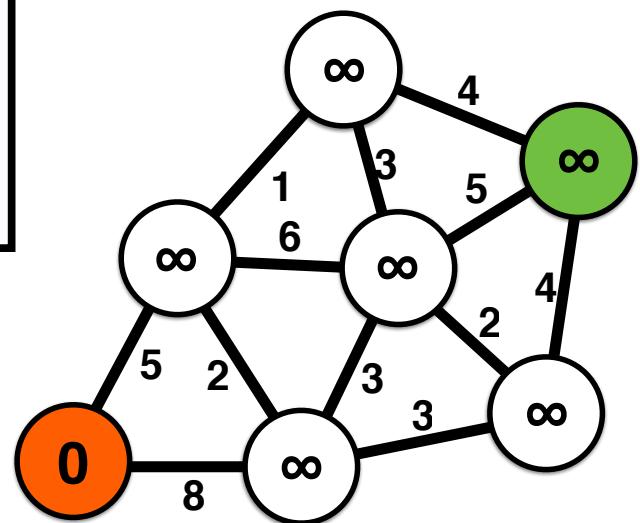
- visits every node to compute its distance and parent
- at each iteration:
 - select the node to visit based on its priority
 - remove current node from visit_list

end if

end for loop

end while loop

output $\leftarrow \text{parent, distance}$



Search algorithm template

```
all nodes  $\leftarrow \{\text{dist}_{\text{start}} \leftarrow \text{infinity}, \text{parent}_{\text{start}} \leftarrow \text{none}, \text{visited}_{\text{start}} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{\text{dist}_{\text{start}} \leftarrow 0, \text{parent}_{\text{start}} \leftarrow \text{none}, \text{visited}_{\text{start}} \leftarrow \text{true}\}$ 
visit_list  $\leftarrow \text{start\_node}$ 
```

```
while visit_list != empty && current_node != goal
```

```
    cur_node  $\leftarrow \text{highestPriority}(\text{visit\_list})$ 
```

```
    visitedcur_node  $\leftarrow \text{true}$ 
```

```
    for each nbr in not_visited(adjacent(cur_node))
```

```
        add(nbr to visit_list)
```

```
        if distnbr > distcur_node + distStraightLine(nbr,cur_node)
```

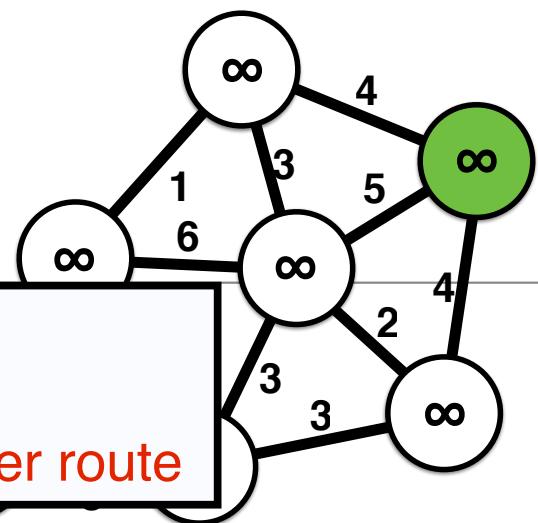
```
            parentnbr  $\leftarrow \text{current\_node}$ 
```

```
            distnbr  $\leftarrow \text{dist}_{\text{cur\_node}} + \text{distStraightLine}(\text{nbr}, \text{cur\_node})$ 
```

```
        end if
```

For each iteration on a single node

- add all unvisited neighbors of the node to the visit list
- add the node as a parent to a neighbor, if it creates a shorter route

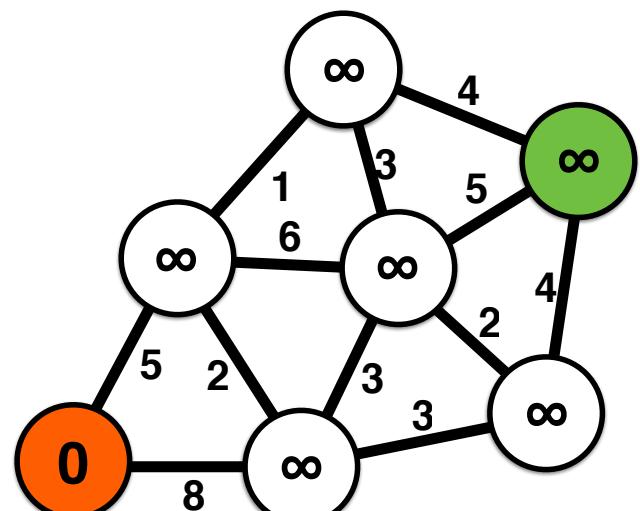


Search algorithm template

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
visit_list  $\leftarrow \text{start\_node}$ 

while visit_list != empty && current_node != goal
    cur_node  $\leftarrow \text{highestPriority}(\text{visit\_list})$ 
    visitedcur_node  $\leftarrow \text{true}$ 
    for each nbr in not_visited(adjacent(cur_node))
        add(nbr to visit_list)
        if distnbr > distcur_node + distance(nbr,cur_node)
            parentnbr  $\leftarrow \text{current\_node}$ 
            distnbr  $\leftarrow dist_{cur\_node} + distance(nbr,cur\_node)$ 
        end if
    end for loop
end while loop
output  $\leftarrow \text{parent, distance}$ 
```

Output the resulting routes and distances

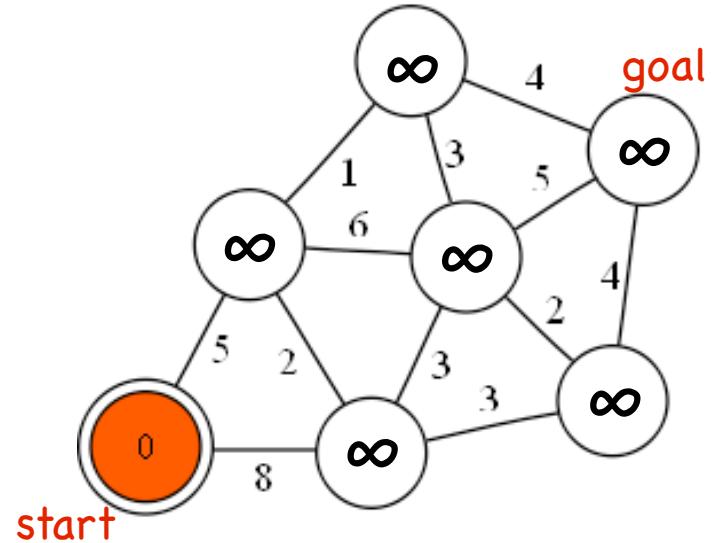


Depth-first search

Search algorithm template

```
all nodes  $\leftarrow \{\text{dist}_{\text{start}} \leftarrow \text{infinity}, \text{parent}_{\text{start}} \leftarrow \text{none}, \text{visited}_{\text{start}} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{\text{dist}_{\text{start}} \leftarrow 0, \text{parent}_{\text{start}} \leftarrow \text{none}, \text{visited}_{\text{start}} \leftarrow \text{true}\}$ 
visit_list  $\leftarrow \text{start\_node}$ 

while visit_list != empty && current_node != goal
    cur_node  $\leftarrow \text{highestPriority}(\text{visit\_list})$ 
    visitedcur_node  $\leftarrow \text{true}$ 
    for each nbr in not_visited(adjacent(cur_node))
        add(nbr to visit_list)
        if distnbr > distcur_node + distance(nbr,cur_node)
            parentnbr  $\leftarrow \text{current\_node}$ 
            distnbr  $\leftarrow \text{dist}_{\text{cur\_node}} + \text{distance}(\text{nbr}, \text{cur\_node})$ 
        end if
    end for loop
end while loop
output  $\leftarrow \text{parent, distance}$ 
```



Depth-first search

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited}_{start} \leftarrow \text{false}\}$   
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited}_{start} \leftarrow \text{true}\}$   
visit_stack  $\leftarrow$  start_node
```

```
while visit_stack != empty && current_node != goal
```

```
    cur_node  $\leftarrow$  pop(visit_stack)  $\leftarrow$  
```

```
    visitedcur_node  $\leftarrow$  true
```

```
    for each nbr in not_visited(adjacent(cur_node))
```

```
        push(nbr to visit_stack)
```

```
        if distnbr > distcur_node + distance(nbr,cur_node)
```

```
            parentnbr  $\leftarrow$  current_node
```

```
            distnbr  $\leftarrow$  distcur_node + distance(nbr,cur_node)
```

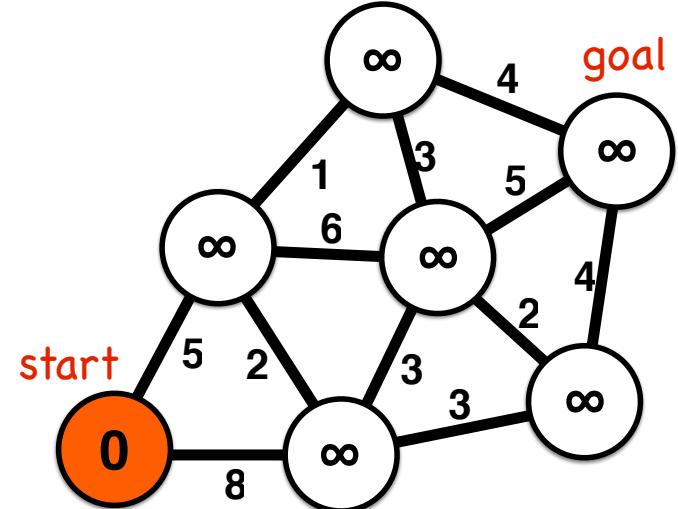
```
        end if
```

```
    end for loop
```

```
end while loop
```

```
output  $\leftarrow$  parent, distance
```

Priority:
Most recent



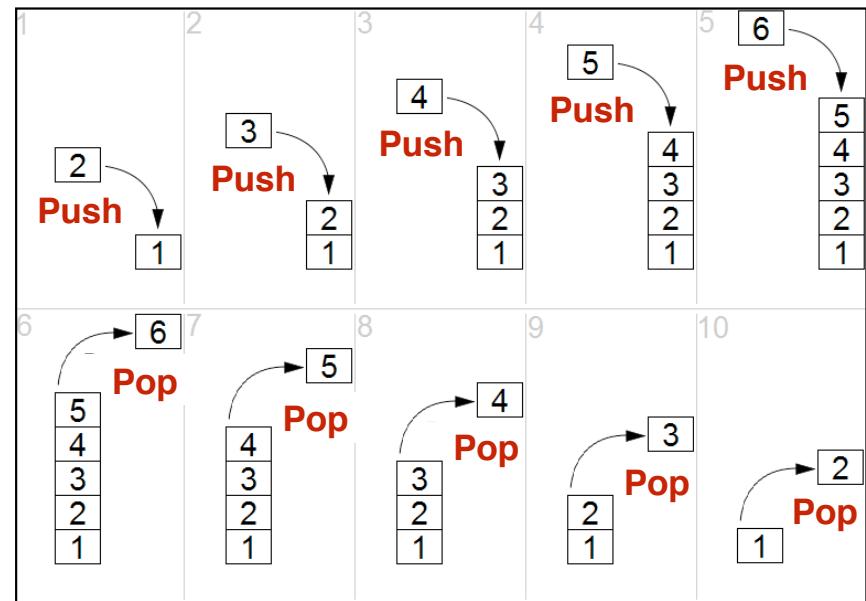
Stack data structure

A stack is a “last in, first out” (or LIFO) structure, with two operations:

push: to add an element to the top of the stack

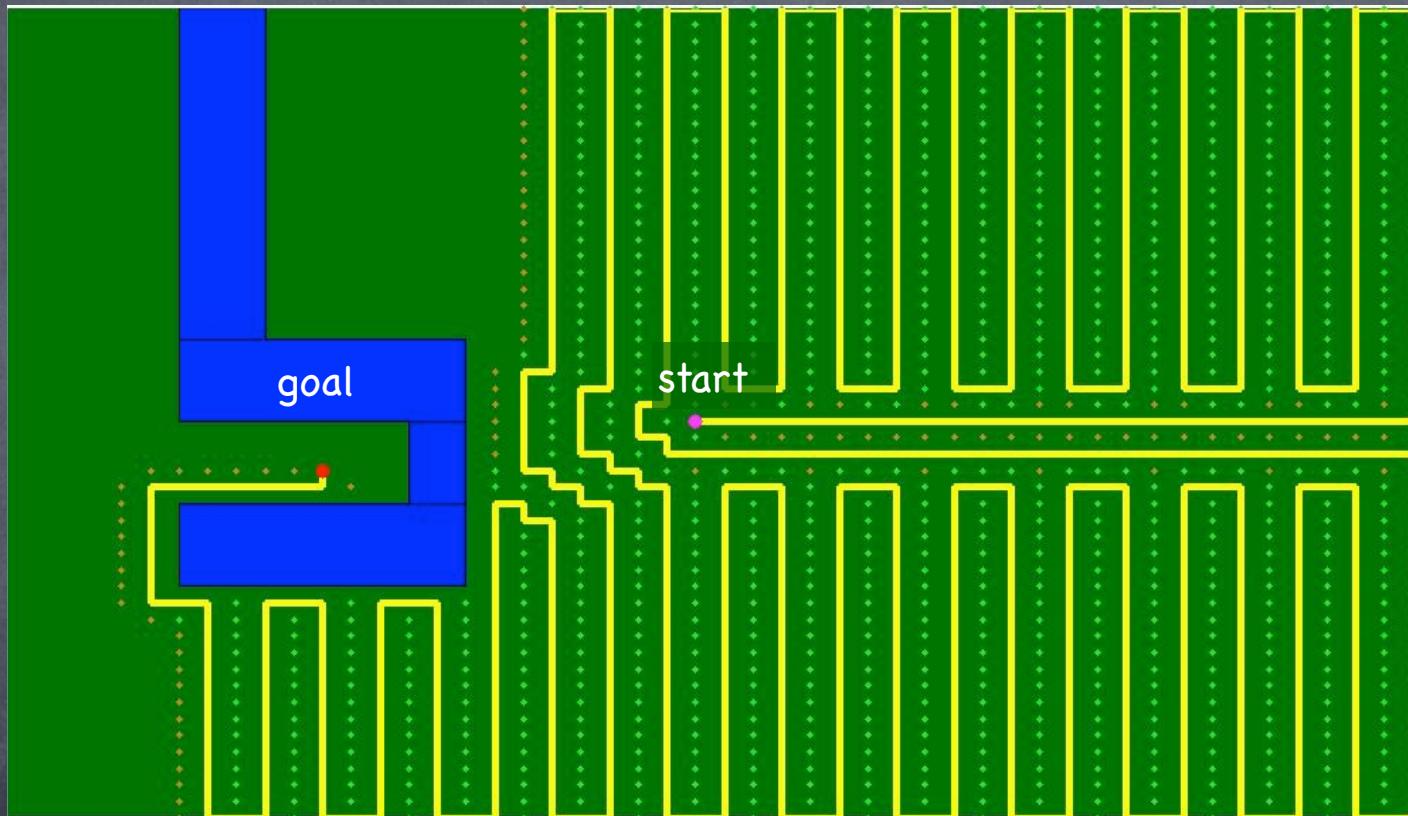
pop: to remove an element from the top of the stack

Stack example for reversing
the order of six elements



matlab example: DFS

pathplan.m

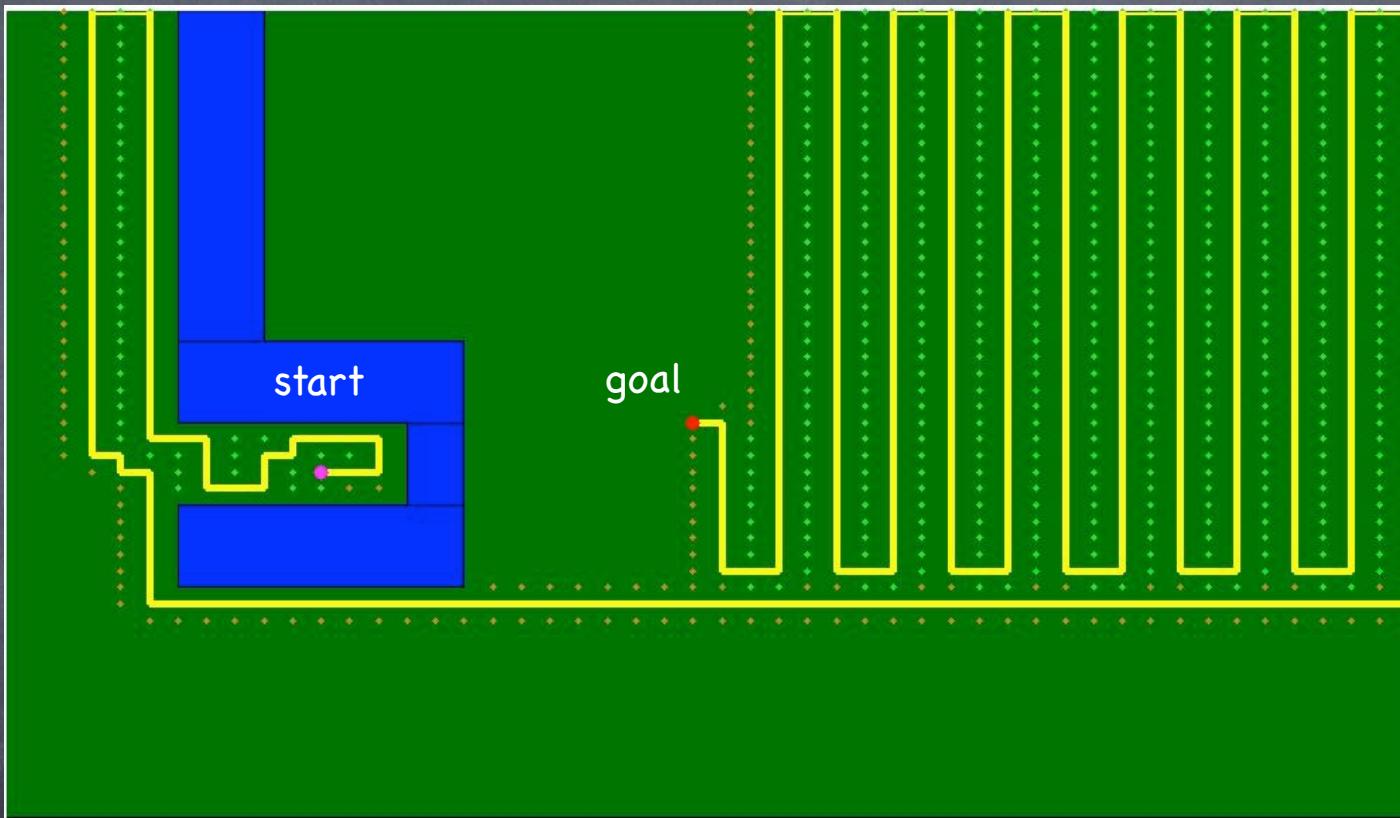


visited locations in light green
4-connected grid, right visited first

UM EECS 998/598 autorob.github.io

matlab example: DFS

pathplan.m



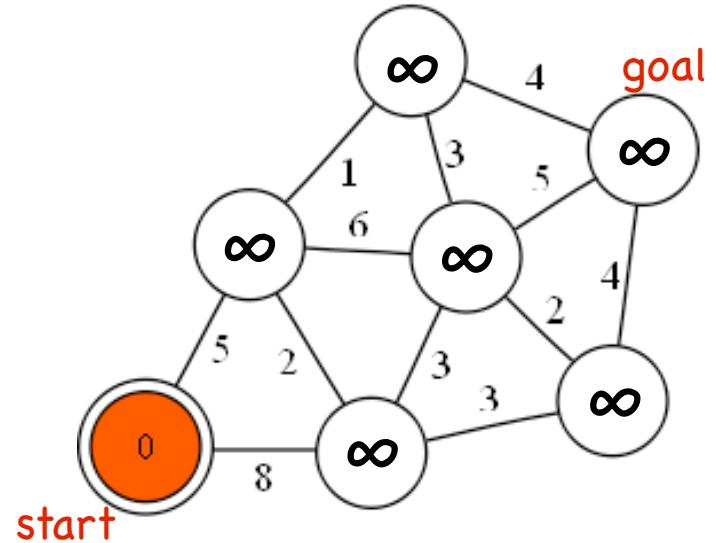
visited locations in light green
4-connected grid, right visited first
UM EECS 998/598 autorob.github.io

Breadth-first search

Search algorithm template

```
all nodes  $\leftarrow \{\text{dist}_{\text{start}} \leftarrow \text{infinity}, \text{parent}_{\text{start}} \leftarrow \text{none}, \text{visited}_{\text{start}} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{\text{dist}_{\text{start}} \leftarrow 0, \text{parent}_{\text{start}} \leftarrow \text{none}, \text{visited}_{\text{start}} \leftarrow \text{true}\}$ 
visit_list  $\leftarrow \text{start\_node}$ 

while visit_list != empty && current_node != goal
    cur_node  $\leftarrow \text{highestPriority}(\text{visit\_list})$ 
    visitedcur_node  $\leftarrow \text{true}$ 
    for each nbr in not_visited(adjacent(cur_node))
        add(nbr to visit_list)
        if distnbr > distcur_node + distance(nbr,cur_node)
            parentnbr  $\leftarrow \text{current\_node}$ 
            distnbr  $\leftarrow \text{dist}_{\text{cur\_node}} + \text{distance}(\text{nbr}, \text{cur\_node})$ 
        end if
    end for loop
end while loop
output  $\leftarrow \text{parent, distance}$ 
```



Breadth-first search

```
all nodes  $\leftarrow \{\text{dist}_{\text{start}} \leftarrow \text{infinity}, \text{parent}_{\text{start}} \leftarrow \text{none}, \text{visited}_{\text{start}} \leftarrow \text{false}\}$   
start_node  $\leftarrow \{\text{dist}_{\text{start}} \leftarrow 0, \text{parent}_{\text{start}} \leftarrow \text{none}, \text{visited}_{\text{start}} \leftarrow \text{true}\}$   
visit_queue  $\leftarrow \text{start\_node}$ 
```

```
while visit_queue != empty && current_node != goal
```

```
    cur_node  $\leftarrow \text{dequeue}(\text{visit\_queue})$  
```

```
    visitedcur_node  $\leftarrow \text{true}$ 
```

```
    for each nbr in not_visited(adjacent(cur_node))
```

```
        enqueue(nbr to visit_queue)
```

```
        if distnbr > distcur_node + distance(nbr,cur_node)
```

```
            parentnbr  $\leftarrow \text{current\_node}$ 
```

```
            distnbr  $\leftarrow \text{dist}_{\text{cur\_node}} + \text{distance}(\text{nbr}, \text{cur\_node})$ 
```

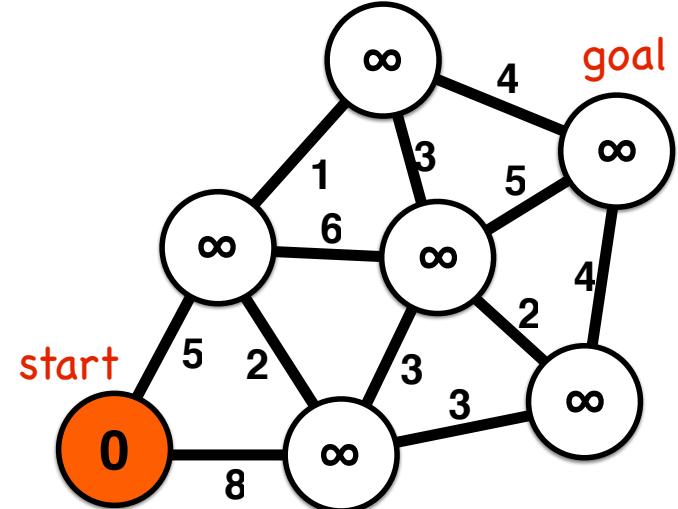
```
        end if
```

```
    end for loop
```

```
    end while loop
```

```
output  $\leftarrow \text{parent, distance}$ 
```

Priority:
Least recent

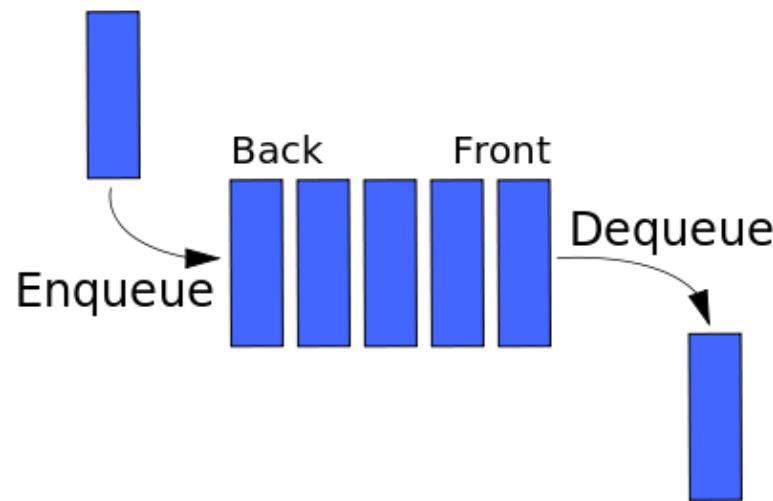


Queue data structure

A queue is a “first in, first out” (or FIFO) structure, with two operations

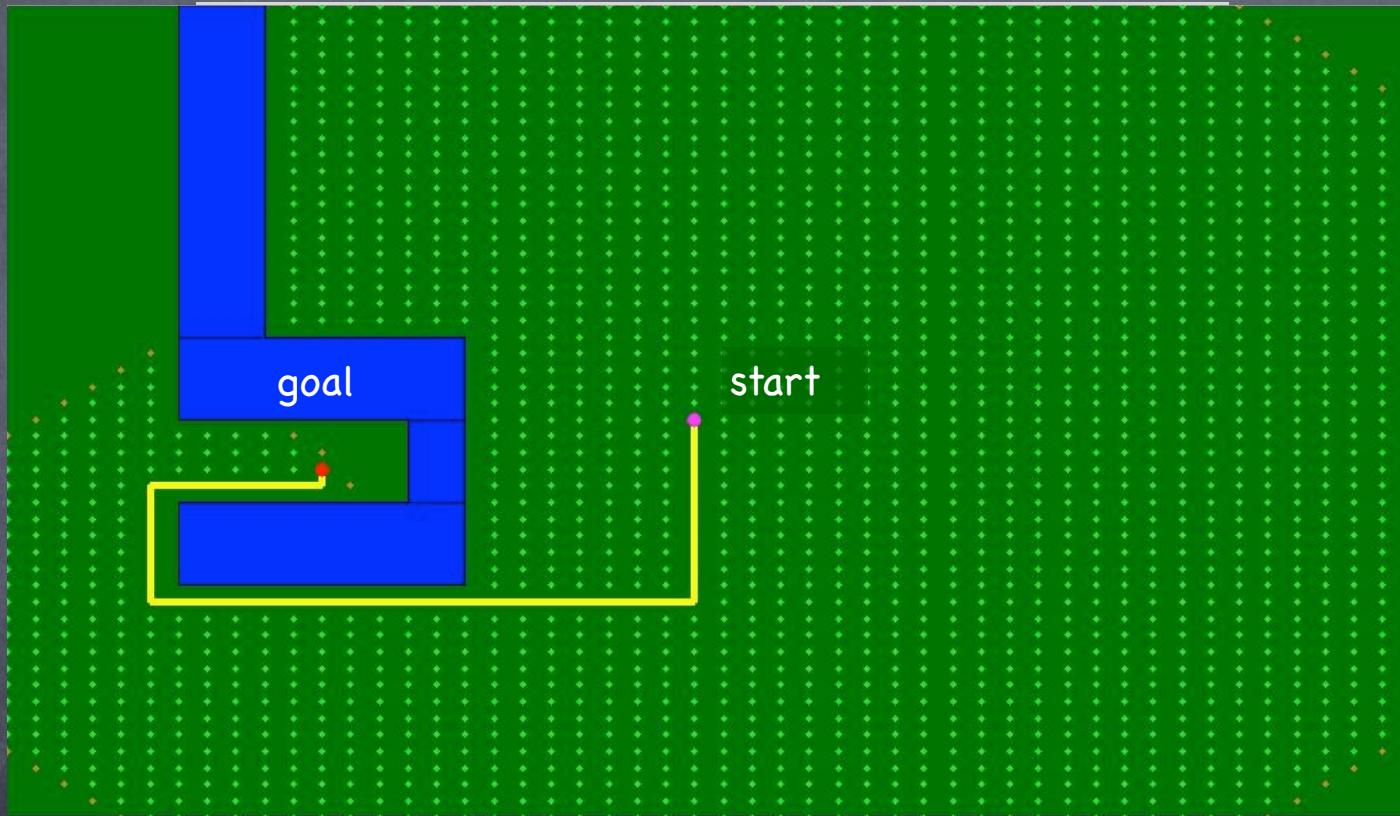
enqueue: to add an element to the back of the stack

dequeue: to remove an element from the front of the stack



matlab example: BFS

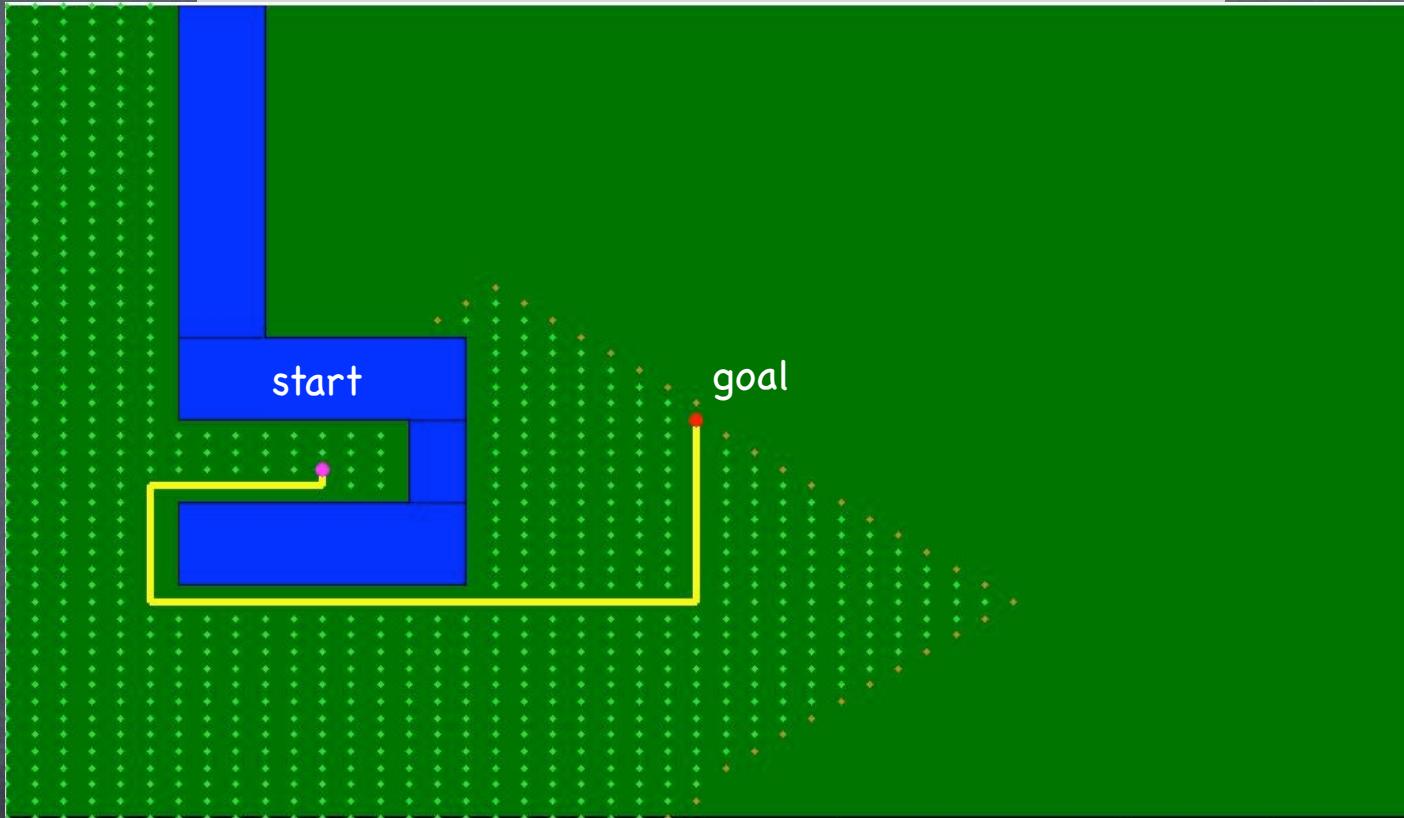
pathplan.m



visited locations in light green
4-connected grid, right visited first
UM EECS 998/598 autorob.github.io

matlab example: BFS

pathplan.m



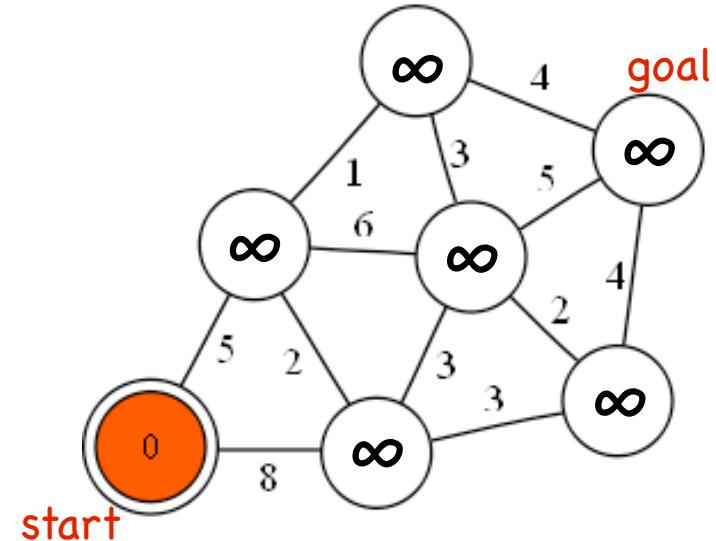
visited locations in light green
4-connected grid, right visited first
UM EECS 998/598 autorob.github.io

Dijkstra shortest path

Search algorithm template

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
visit_list  $\leftarrow \text{start\_node}$ 
```

```
while visit_list != empty && current_node != goal
    cur_node  $\leftarrow \text{highestPriority}(\text{visit\_list})$ 
    visitedcur_node  $\leftarrow \text{true}$ 
    for each nbr in not_visited(adjacent(cur_node))
        add(nbr to visit_list)
        if distnbr > distcur_node + distance(nbr,cur_node)
            parentnbr  $\leftarrow \text{current\_node}$ 
            distnbr  $\leftarrow \text{dist}_{cur\_node} + \text{distance}(\text{nbr}, \text{cur\_node})$ 
        end if
    end for loop
end while loop
output  $\leftarrow \text{parent, distance}$ 
```



Dijkstra shortest path algorithm

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
```

```
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
```

```
visit_queue  $\leftarrow \text{start\_node}$ 
```

```
while visit_queue != empty && current_node != goal
```

```
    cur_node  $\leftarrow \text{min\_distance(visit\_queue)}$ 
```

Priority:
Minimum distance

```
    visitedcur_node  $\leftarrow \text{true}$ 
```

```
    for each nbr in not_visited(adjacent(cur_node))
```

```
        enqueue(nbr to visit_queue)
```

```
        if distnbr > distcur_node + distance(nbr,cur_node)
```

```
            parentnbr  $\leftarrow \text{current\_node}$ 
```

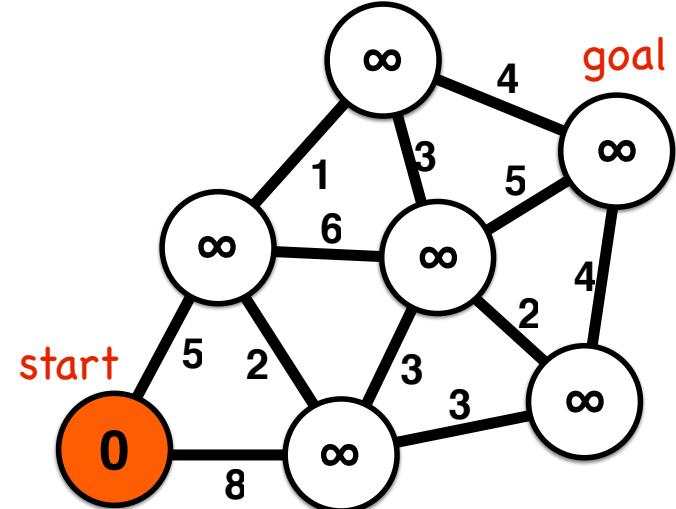
```
            distnbr  $\leftarrow \text{dist}_{cur\_node} + \text{distance}(nbr,cur\_node)$ 
```

```
        end if
```

```
    end for loop
```

```
    end while loop
```

```
output  $\leftarrow \text{parent, distance}$ 
```

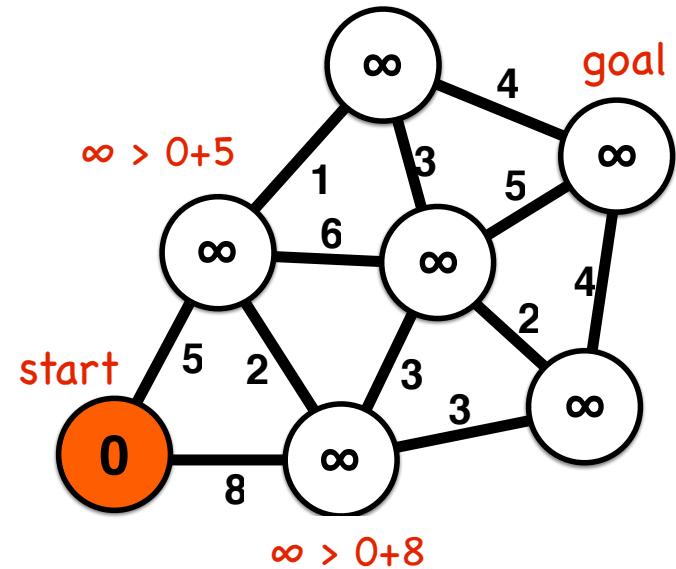


Dijkstra shortest path algorithm

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
visit_queue  $\leftarrow \text{start\_node}$ 

while visit_queue != empty && current_node != goal
    cur_node  $\leftarrow \text{min\_distance(visit\_queue)}$ 
    visitedcur_node  $\leftarrow \text{true}$ 
    for each nbr in not_visited(adjacent(cur_node))
        enqueue(nbr to visit_queue)
        if distnbr > distcur_node + distance(nbr,cur_node)
            parentnbr  $\leftarrow \text{current\_node}$ 
            distnbr  $\leftarrow dist_{cur\_node} + distance(nbr,cur\_node)$ 
        end if
    end for loop
end while loop
output  $\leftarrow \text{parent, distance}$ 
```

Dijkstra walkthrough

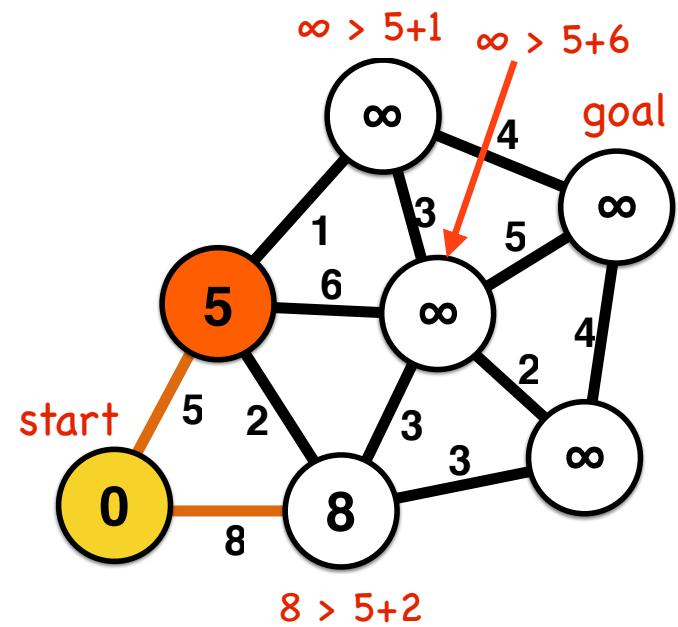


Dijkstra shortest path algorithm

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
visit_queue  $\leftarrow \text{start\_node}$ 

while visit_queue != empty && current_node != goal
    cur_node  $\leftarrow \text{min\_distance(visit\_queue)}$ 
    visitedcur_node  $\leftarrow \text{true}$ 
    for each nbr in not_visited(adjacent(cur_node))
        enqueue(nbr to visit_queue)
        if distnbr > distcur_node + distance(nbr,cur_node)
            parentnbr  $\leftarrow \text{current\_node}$ 
            distnbr  $\leftarrow dist_{cur\_node} + distance(nbr,cur\_node)$ 
        end if
    end for loop
end while loop
output  $\leftarrow \text{parent, distance}$ 
```

Dijkstra walkthrough

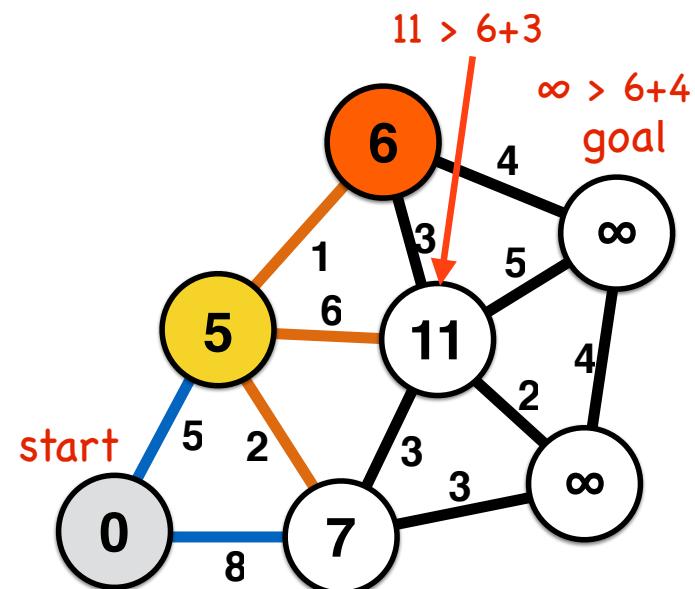


Dijkstra shortest path algorithm

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
visit_queue  $\leftarrow \text{start\_node}$ 

while visit_queue != empty && current_node != goal
    cur_node  $\leftarrow \text{min\_distance(visit\_queue)}$ 
    visitedcur_node  $\leftarrow \text{true}$ 
    for each nbr in not_visited(adjacent(cur_node))
        enqueue(nbr to visit_queue)
        if distnbr > distcur_node + distance(nbr,cur_node)
            parentnbr  $\leftarrow \text{current\_node}$ 
            distnbr  $\leftarrow dist_{cur\_node} + distance(nbr,cur\_node)$ 
        end if
    end for loop
end while loop
output  $\leftarrow \text{parent, distance}$ 
```

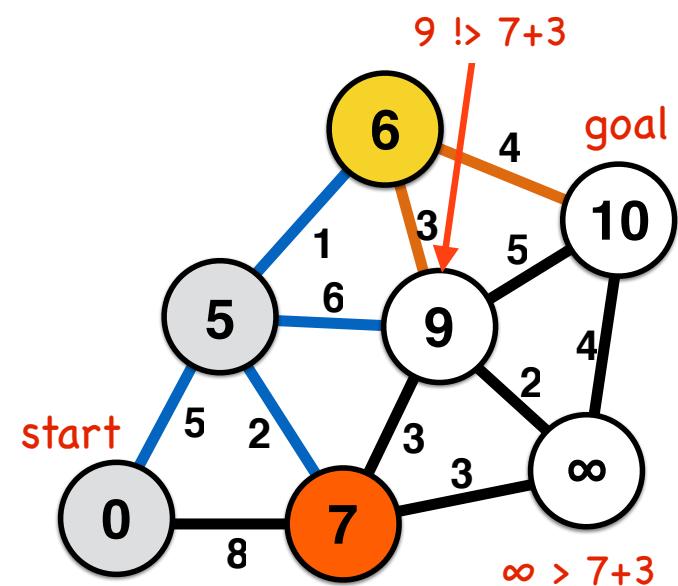
Dijkstra walkthrough



Dijkstra shortest path algorithm

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
visit_queue  $\leftarrow \text{start\_node}$ 

while visit_queue != empty && current_node != goal
    cur_node  $\leftarrow \text{min\_distance(visit\_queue)}$ 
    visitedcur_node  $\leftarrow \text{true}$ 
    for each nbr in not_visited(adjacent(cur_node))
        enqueue(nbr to visit_queue)
        if distnbr > distcur_node + distance(nbr,cur_node)
            parentnbr  $\leftarrow \text{current\_node}$ 
            distnbr  $\leftarrow dist_{cur\_node} + distance(nbr,cur\_node)$ 
        end if
    end for loop
end while loop
output  $\leftarrow \text{parent, distance}$ 
```



Dijkstra shortest path algorithm

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$   
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$   
visit_queue  $\leftarrow \text{start\_node}$ 
```

```
while visit_queue != empty && current_node != goal
```

```
    cur_node  $\leftarrow \text{min\_distance(visit\_queue)}$ 
```

```
    visitedcur_node  $\leftarrow \text{true}$ 
```

```
    for each nbr in not_visited(adjacent(cur_node))
```

```
        enqueue(nbr to visit_queue)
```

```
        if distnbr > distcur_node + distance(nbr,cur_node)
```

```
            parentnbr  $\leftarrow \text{current\_node}$ 
```

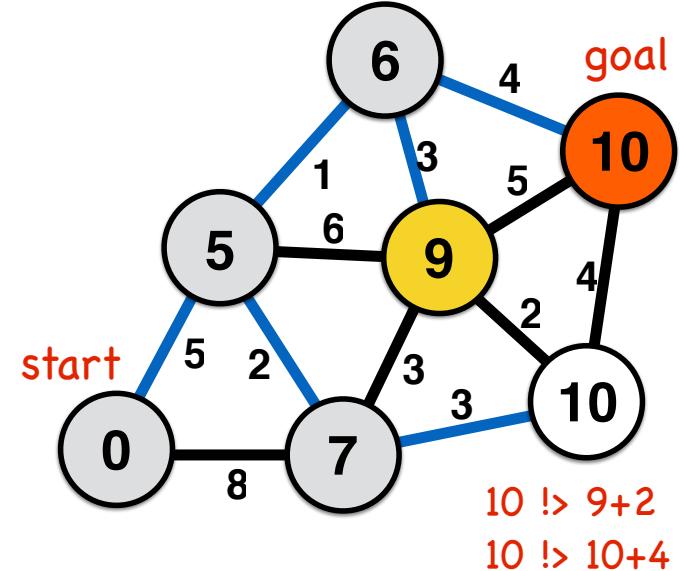
```
            distnbr  $\leftarrow dist_{cur\_node} + distance(nbr,cur\_node)$ 
```

```
        end if
```

```
    end for loop
```

```
    end while loop
```

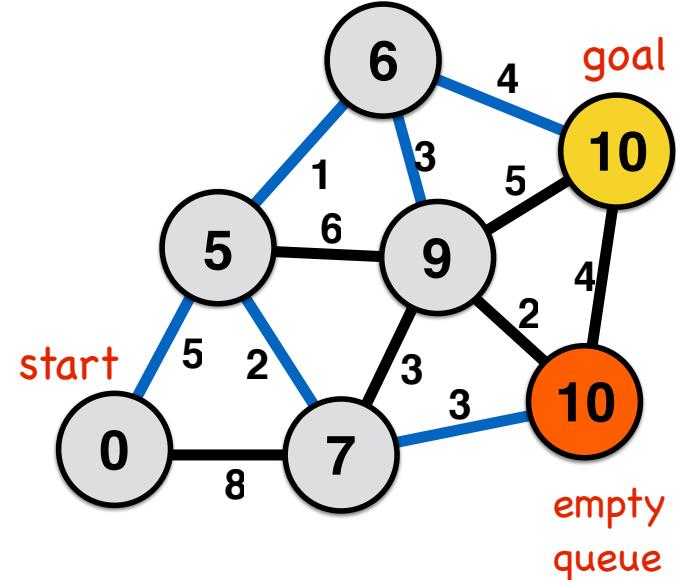
```
output  $\leftarrow \text{parent, distance}$ 
```



Dijkstra shortest path algorithm

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
visit_queue  $\leftarrow \text{start\_node}$ 

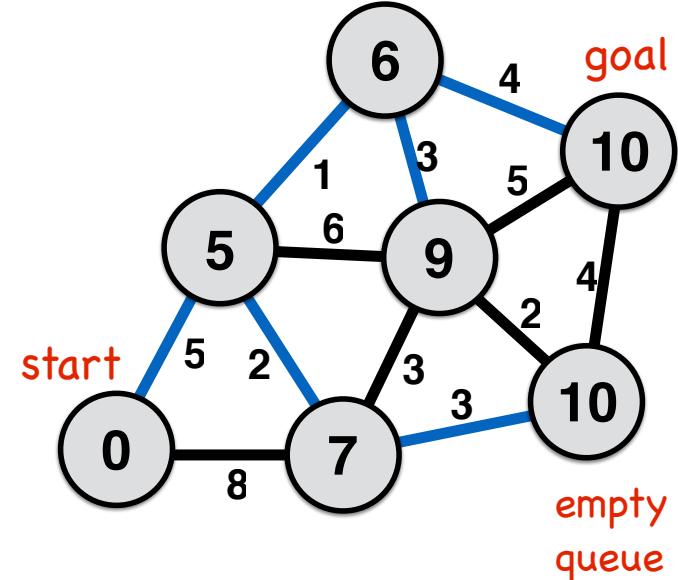
while visit_queue != empty && current_node != goal
    cur_node  $\leftarrow \text{min\_distance(visit\_queue)}$ 
    visitedcur_node  $\leftarrow \text{true}$ 
    for each nbr in not_visited(adjacent(cur_node))
        enqueue(nbr to visit_queue)
        if distnbr > distcur_node + distance(nbr,cur_node)
            parentnbr  $\leftarrow \text{current\_node}$ 
            distnbr  $\leftarrow dist_{cur\_node} + distance(nbr,cur\_node)$ 
        end if
    end for loop
end while loop
output  $\leftarrow \text{parent, distance}$ 
```



Dijkstra shortest path algorithm

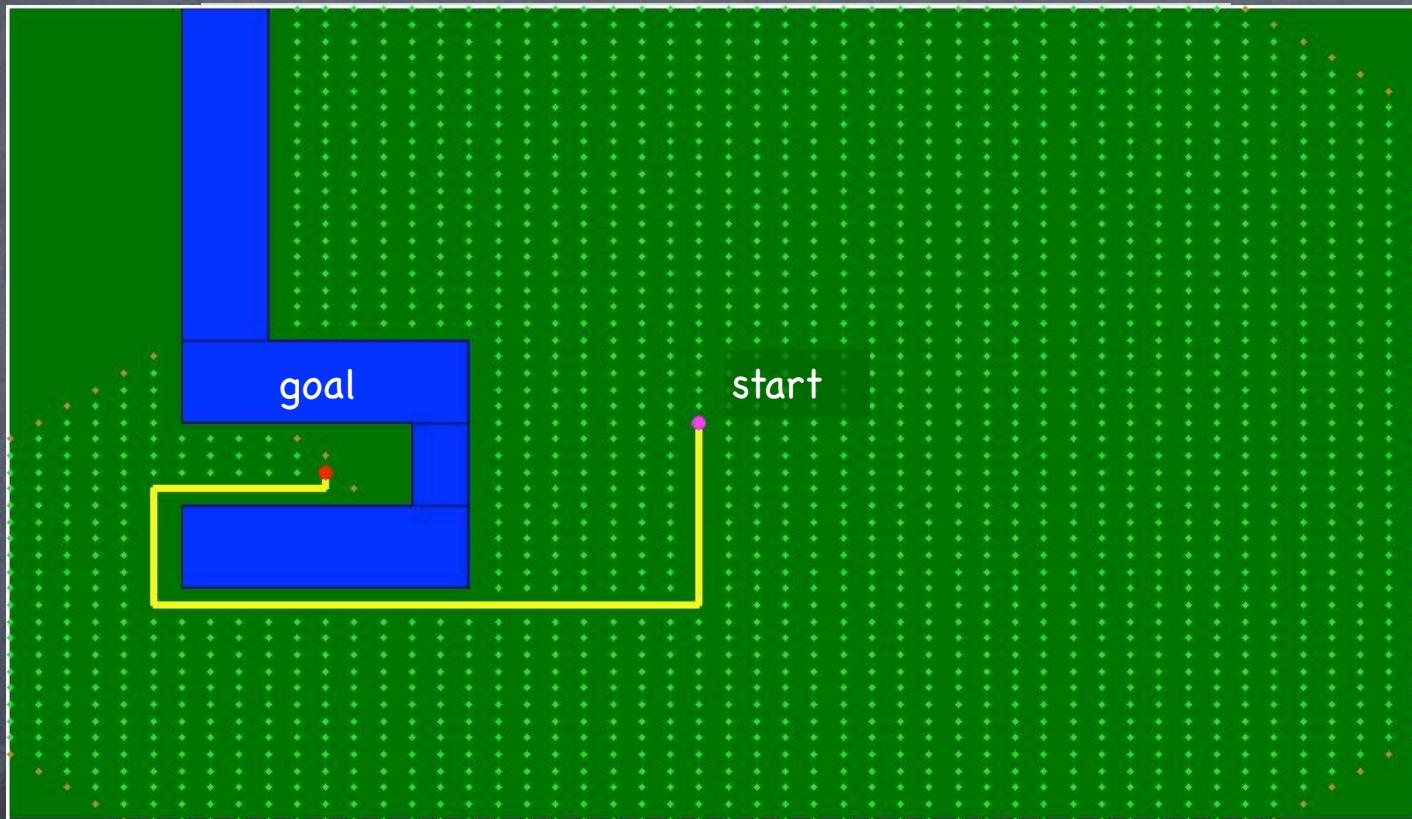
```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
visit_queue  $\leftarrow \text{start\_node}$ 

while visit_queue != empty && current_node != goal
    cur_node  $\leftarrow \text{min\_distance(visit\_queue)}$ 
    visitedcur_node  $\leftarrow \text{true}$ 
    for each nbr in not_visited(adjacent(cur_node))
        enqueue(nbr to visit_queue)
        if distnbr > distcur_node + distance(nbr,cur_node)
            parentnbr  $\leftarrow \text{current\_node}$ 
            distnbr  $\leftarrow dist_{cur\_node} + distance(nbr,cur\_node)$ 
        end if
    end for loop
end while loop
output  $\leftarrow \text{parent, distance}$ 
```



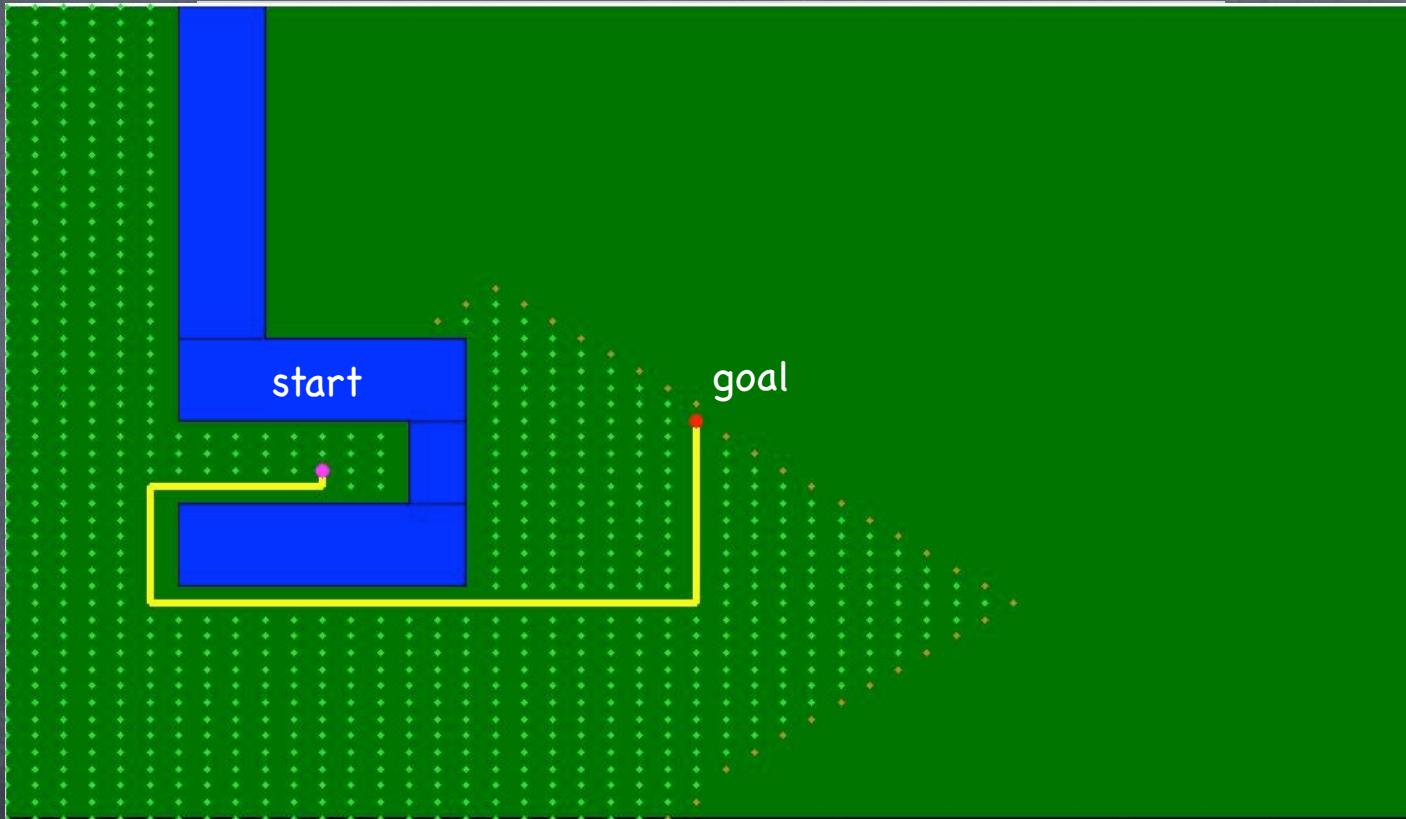
matlab example: Dijkstra

pathplan.m



matlab example: Dijkstra

pathplan.m



A-star shortest path

Dijkstra shortest path algorithm

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
visit_queue  $\leftarrow \text{start\_node}$ 
```

```
while visit_queue != empty && current_node != goal
```

```
    cur_node  $\leftarrow \text{min\_distance(visit\_queue)}$ 
```

```
    visitedcur_node  $\leftarrow \text{true}$ 
```

```
    for each nbr in not_visited(adjacent(cur_node))
```

```
        enqueue(nbr to visit_queue)
```

```
        if distnbr > distcur_node + distance(nbr,cur_node)
```

```
            parentnbr  $\leftarrow \text{current\_node}$ 
```

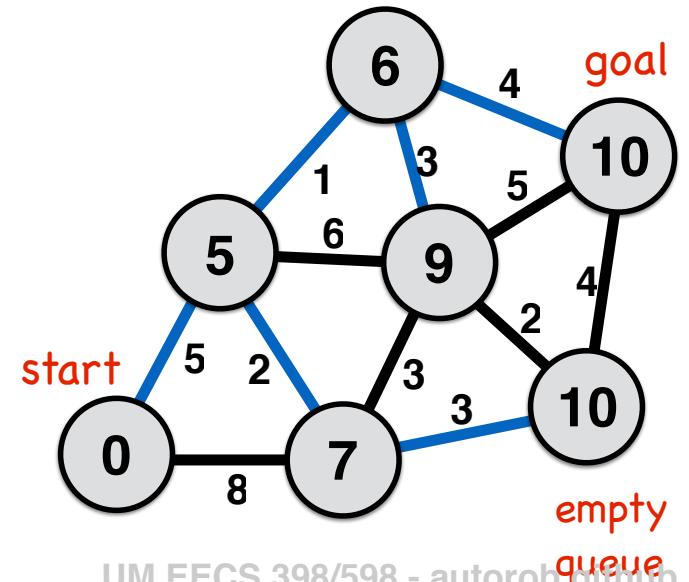
```
            distnbr  $\leftarrow dist_{cur\_node} + distance(nbr,cur\_node)$ 
```

```
        end if
```

```
    end for loop
```

```
    end while loop
```

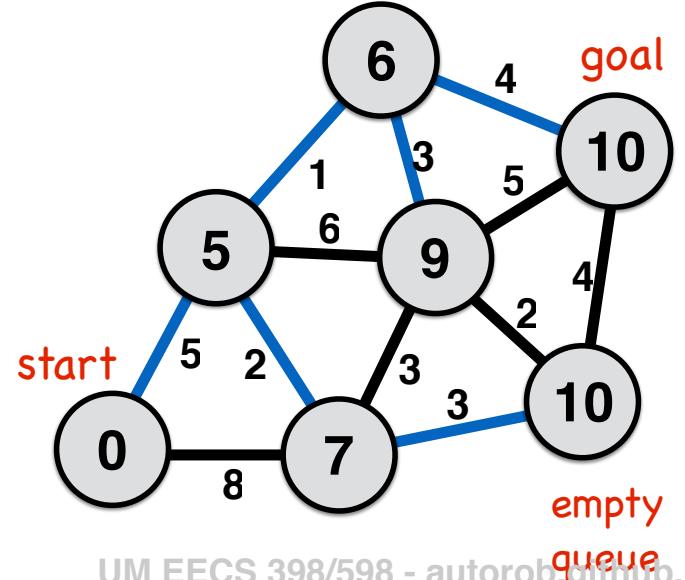
```
output  $\leftarrow \text{parent, distance}$ 
```



A-star shortest path algorithm

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
visit_queue  $\leftarrow \text{start\_node}$ 

while (visit_queue != empty) && current_node != goal
    cur_node  $\leftarrow \text{dequeue(visit\_queue, f\_score)}$ 
    visitedcur_node  $\leftarrow \text{true}$ 
    for each nbr in not_visited(adjacent(cur_node))
        enqueue(nbr to visit_queue)
        if distnbr > distcur_node + distance(nbr,cur_node)
            parentnbr  $\leftarrow \text{current\_node}$ 
            distnbr  $\leftarrow dist_{cur\_node} + distance(nbr,cur\_node)$ 
            f_score  $\leftarrow distance_{nbr} + line\_distance_{nbr,goal}$ 
        end if
    end for loop
end while loop
output  $\leftarrow \text{parent, distance}$ 
```



A-star shortest path algorithm

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$   
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$   
visit_queue  $\leftarrow \text{start\_node}$ 
```

```
while (visit_queue != empty) && current_node != goal
```

```
    cur_node  $\leftarrow \text{dequeue(visit\_queue, f\_score)}$ 
```

```
    visitedcur_node  $\leftarrow \text{true}$ 
```

```
    for each nbr in not_visited(adjacent(cur_node))
```

```
        enqueue(nbr to visit_queue)
```

```
        if distnbr > distcur_node + distance(nbr,cur_node)
```

```
            parentnbr  $\leftarrow \text{current\_node}$ 
```

```
            distnbr  $\leftarrow dist_{cur\_node} + distance(nbr,cur\_node)$ 
```

```
            f_score  $\leftarrow distance_{nbr} + line\_distance_{nbr,goal}$ 
```

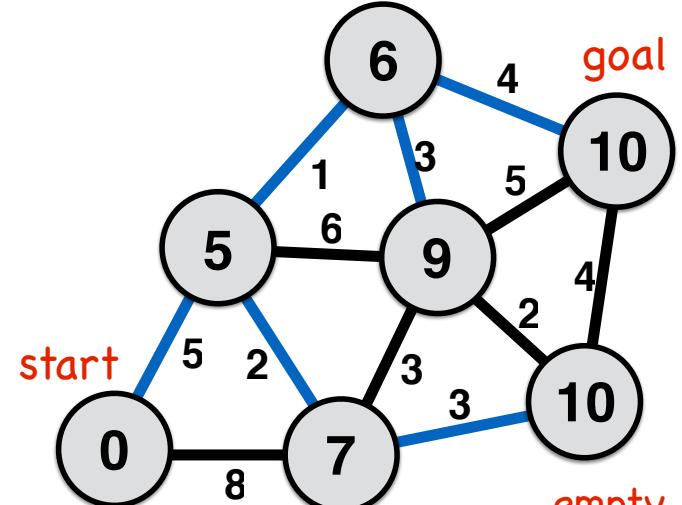
```
        end if
```

```
    end for loop
```

```
end while loop
```

```
output  $\leftarrow \text{parent, distance}$ 
```

priority queue wrt. f_score
(implement min binary heap)



g_score: distance along current path back to start

h_score: best possible distance to goal

A-star shortest path algorithm

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
visit_queue  $\leftarrow \text{start\_node}$ 
```

```
while (visit_queue != empty) && current_node != goal
```

```
    cur_node  $\leftarrow \text{dequeue(visit\_queue, f\_score)}$ 
```

priority queue wrt. f_score
(implement min binary heap)

```
    visitedcur_node  $\leftarrow \text{true}$ 
```

```
    for each nbr in not_visited(adjacent(cur_node))
```

```
        enqueue(nbr,to visit_queue)
```

```
        if  $dist_{nbr} > dist_{cur\_node} + dist_{cur\_node, nbr}$  then
```

```
            parentnbr  $\leftarrow \text{current\_node}$ 
```

```
            distnbr  $\leftarrow dist_{cur\_node} + \text{distance}(nbr, cur\_node)$ 
```

```
            fscore  $\leftarrow \text{distance}_{nbr} + \text{line\_distance}_{nbr,goal}$ 
```

```
        end if
```

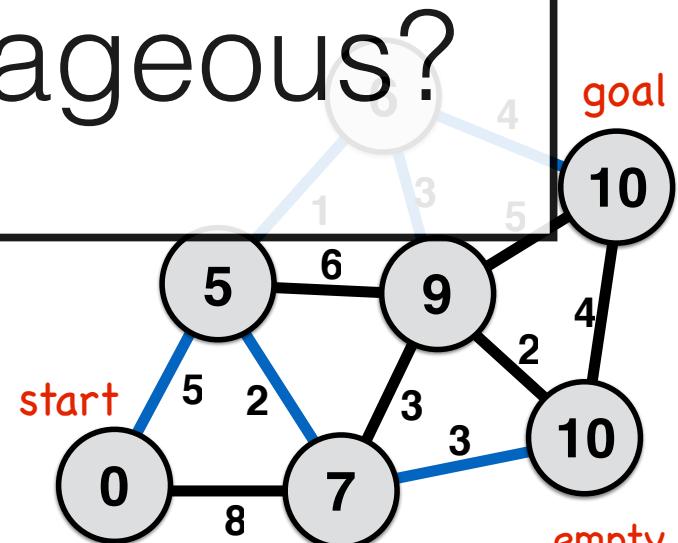
```
    end for loop
```

```
end while loop
```

```
output  $\leftarrow \text{parent, distance}$ 
```

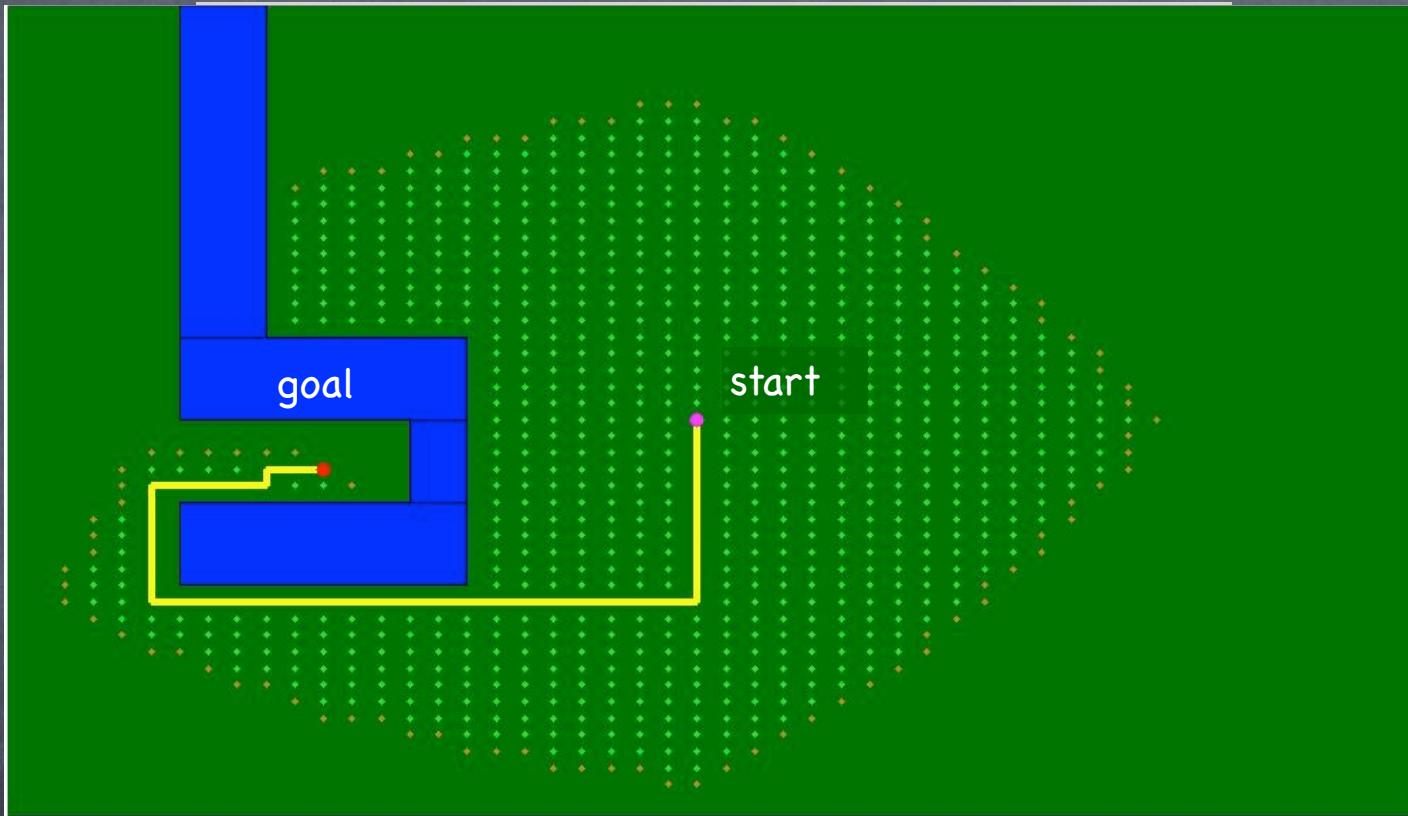
g_score: distance along current path back to start

h_score: best possible distance to goal



matlab example: A*

pathplan.m



Admissibility: Straight line heuristic

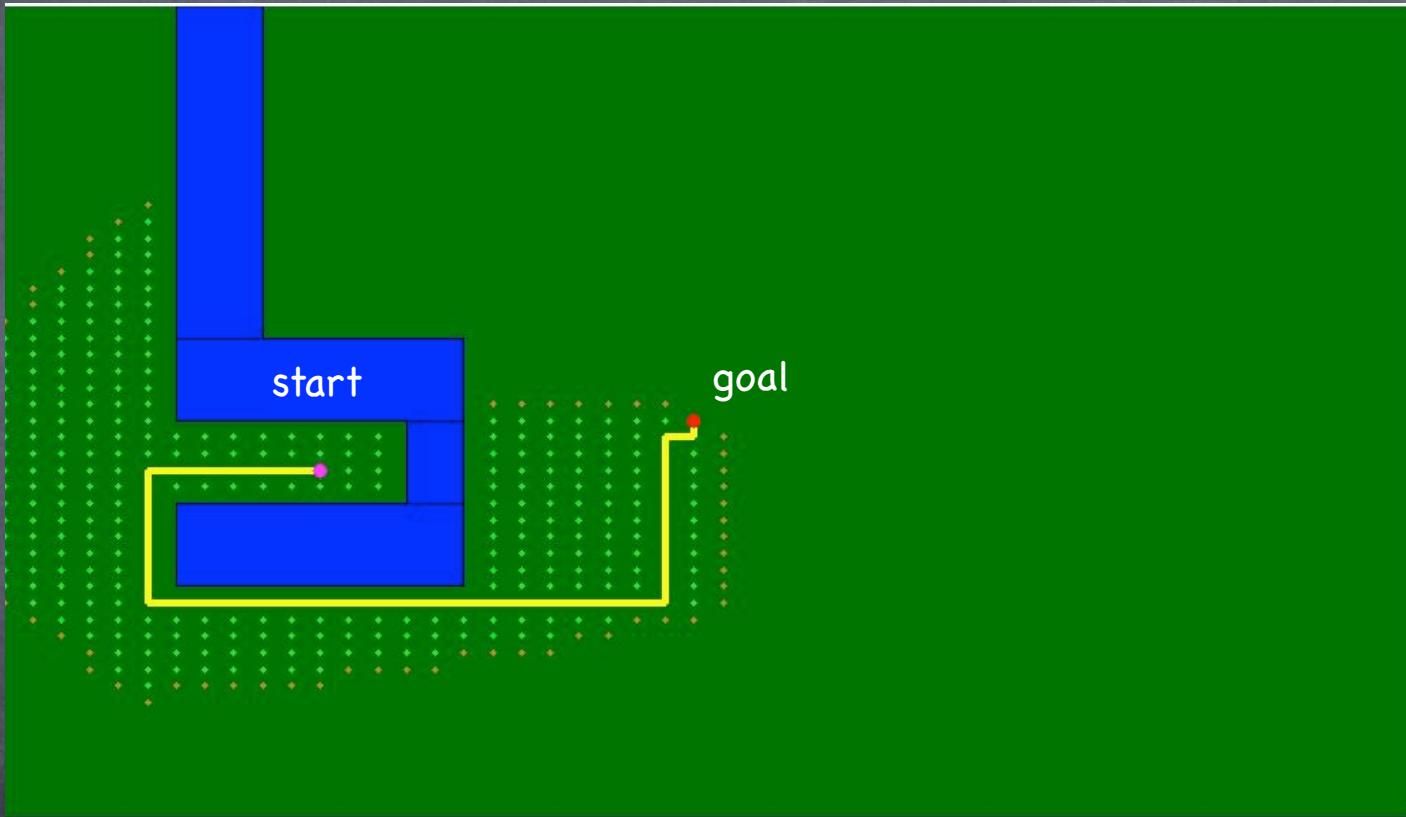
The straight line h_score is an admissible heuristic

Admissibility states that the lowest possible distance from current location to the goal will be less than the h_score

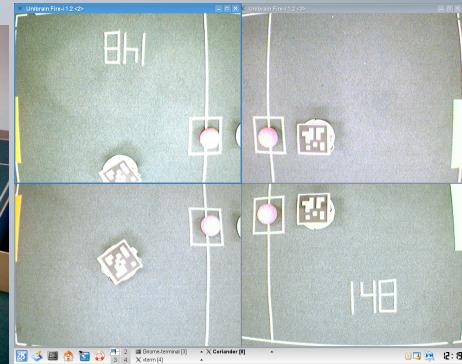
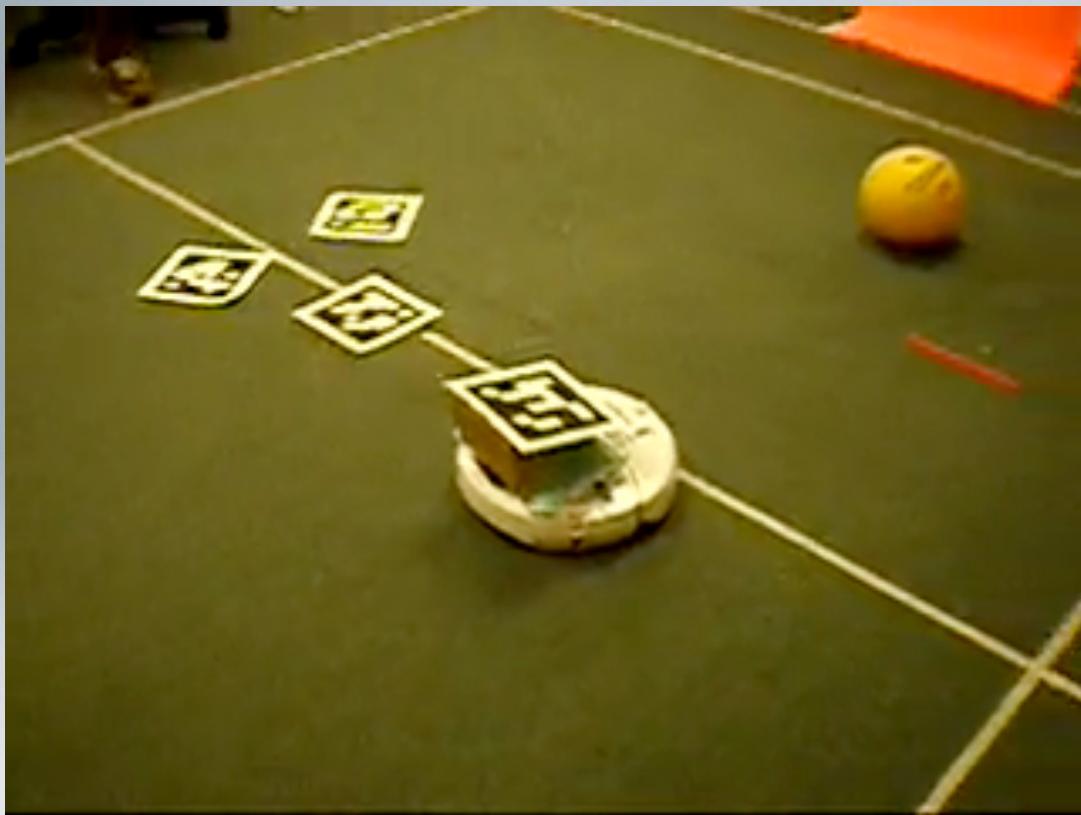
The diagram shows a 2D grid-based search space. A blue rectangular region represents an obstacle. A grey circle at the bottom-left is labeled 'g' (goal). A red square at the top-right is labeled 'start'. A yellow L-shaped path starts from 'g', goes right, then down, ending at a red dot labeled 'h'. A purple arrow points from 'g' to the red dot 'h'. A red arrow points from the red dot 'h' to the start point. The background is green with a dotted pattern.

matlab example: A*

<http://www.cs.brown.edu/courses/cs148/pub/pathplan.m>



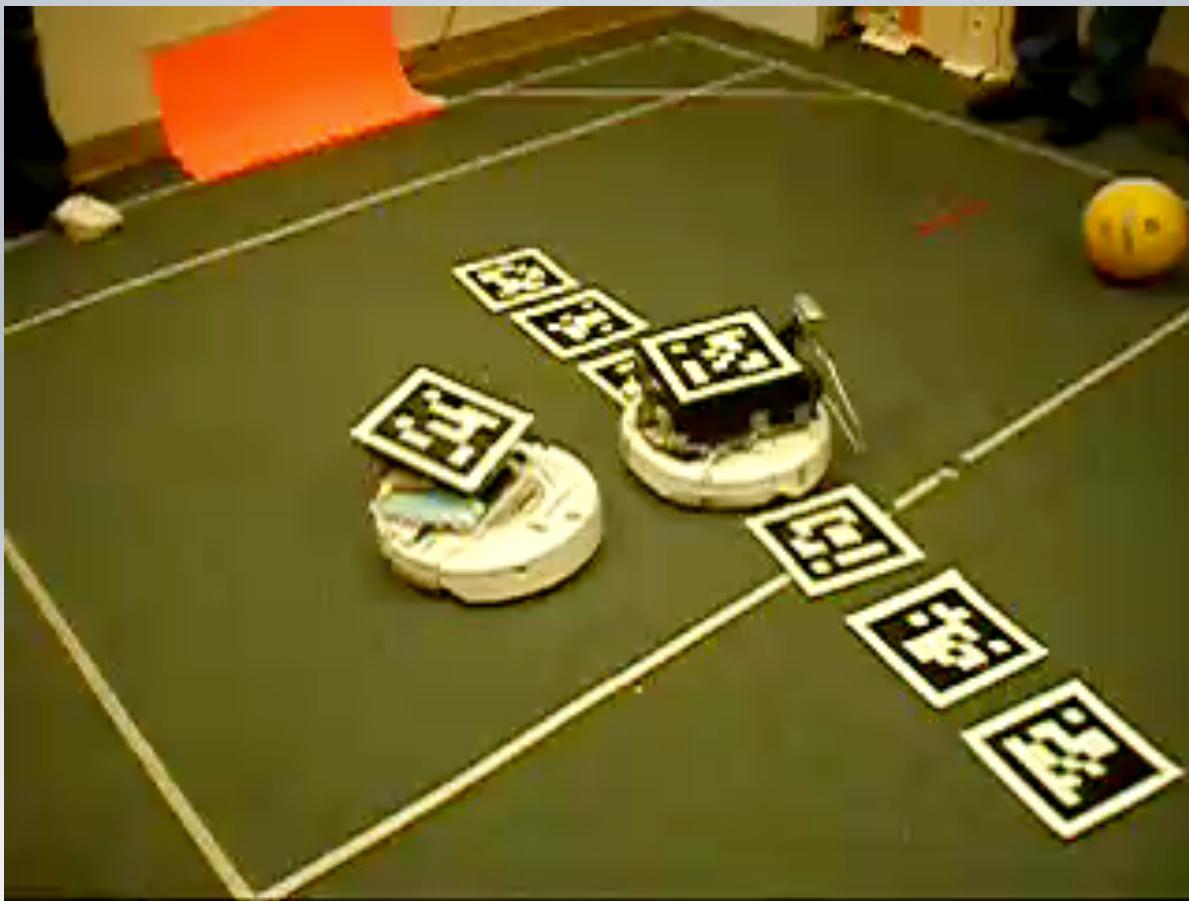
A* PATH PLANNING



Lisa Miller, <http://www.youtube.com/watch?v=2Z2RyeofsZg>

UM EECS 398/598 - autorob.github.io

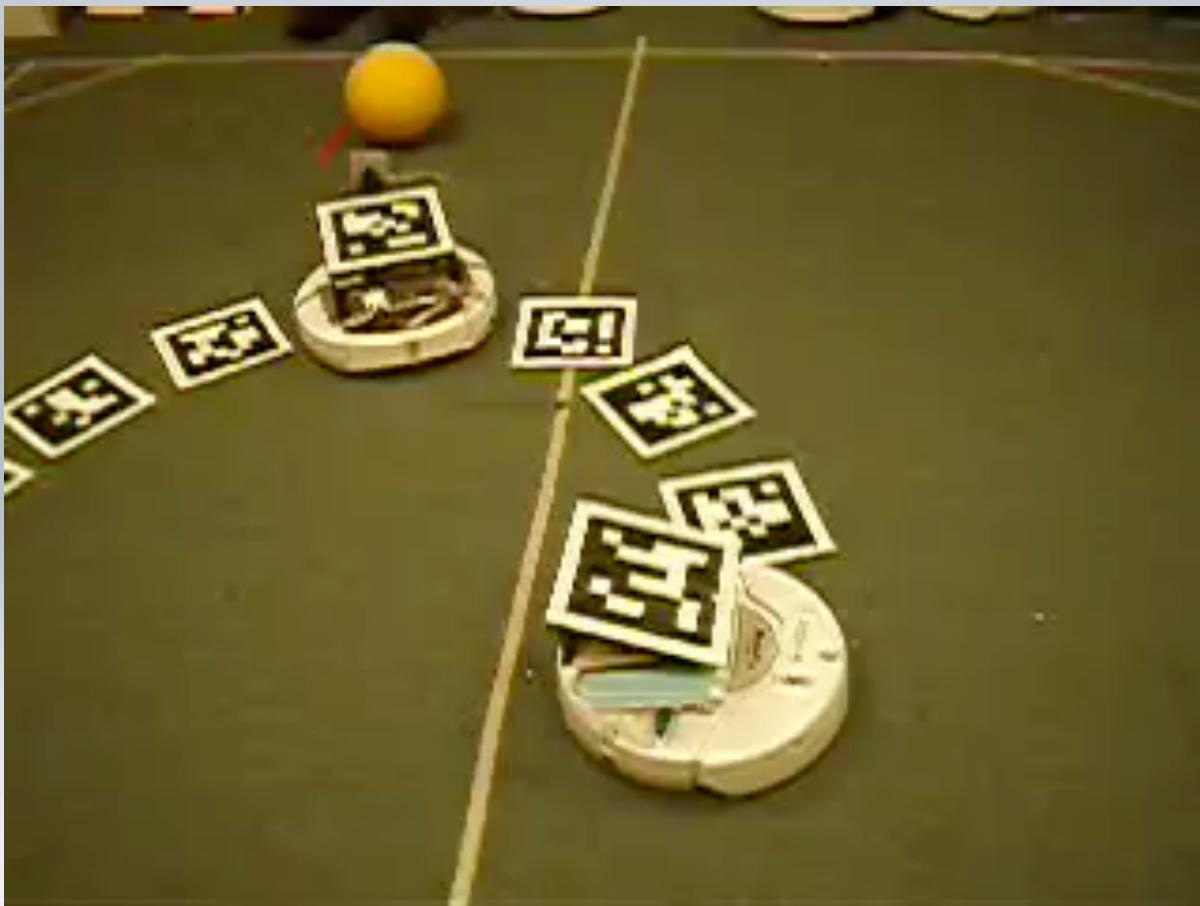
NAVIGATING AROUND A WALL



Lisa Miller

UM EECS 398/598 - autorob.github.io

AVOIDING DEAD-ENDS



Lisa Miller

<http://www.youtube.com/watch?v=k6Kj4VjTKc8>

UM EECS 398/598 - autorob.github.io

Greedy best-first search

A-star shortest path algorithm

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$   
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$   
visit_queue  $\leftarrow \text{start\_node}$ 
```

```
while (visit_queue != empty) && current_node != goal
```

```
    cur_node  $\leftarrow \text{dequeue(visit\_queue, f\_score)}$ 
```

```
    visitedcur_node  $\leftarrow \text{true}$ 
```

```
    for each nbr in not_visited(adjacent(cur_node))
```

```
        enqueue(nbr to visit_queue)
```

```
        if distnbr > distcur_node + distance(nbr,cur_node)
```

```
            parentnbr  $\leftarrow \text{current\_node}$ 
```

```
            distnbr  $\leftarrow dist_{cur\_node} + distance(nbr,cur\_node)$ 
```

```
            f_score  $\leftarrow distance_{nbr} + line\_distance_{nbr,goal}$ 
```

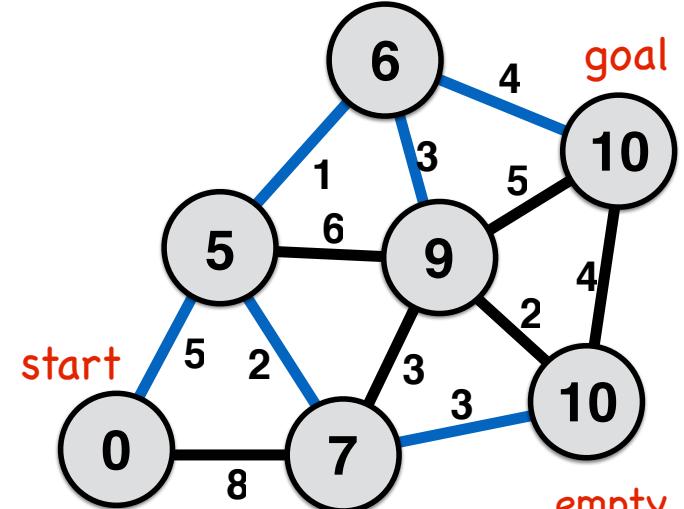
```
        end if
```

```
    end for loop
```

```
end while loop
```

```
output  $\leftarrow \text{parent, distance}$ 
```

priority queue wrt. f_score
(implement min binary heap)



Greedy best-first search

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
```

```
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
```

```
visit_queue  $\leftarrow \text{start\_node}$ 
```

```
while (visit_queue != empty)  $\&\&$  current_node != goal
```

```
    cur_node  $\leftarrow \text{dequeue(visit\_queue, f\_score)}$ 
```

```
    visitedcur_node  $\leftarrow \text{true}$ 
```

```
    for each nbr in not_visited(adjacent(cur_node))
```

```
        enqueue(nbr to visit_queue)
```

```
        if distnbr > distcur_node + distance(nbr,cur_node)
```

```
            parentnbr  $\leftarrow \text{current\_node}$ 
```

```
            distnbr  $\leftarrow \text{dist}_{cur\_node} + \text{distance}(nbr,cur\_node)$ 
```

```
            f_score  $\leftarrow \text{distance}_{nbr} + \text{line\_distance}_{nbr,goal}$ 
```

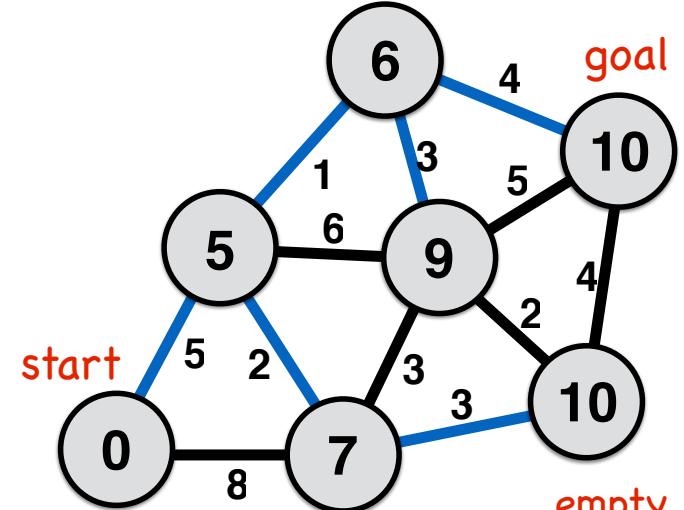
```
        end if
```

```
    end for loop
```

```
    end while loop
```

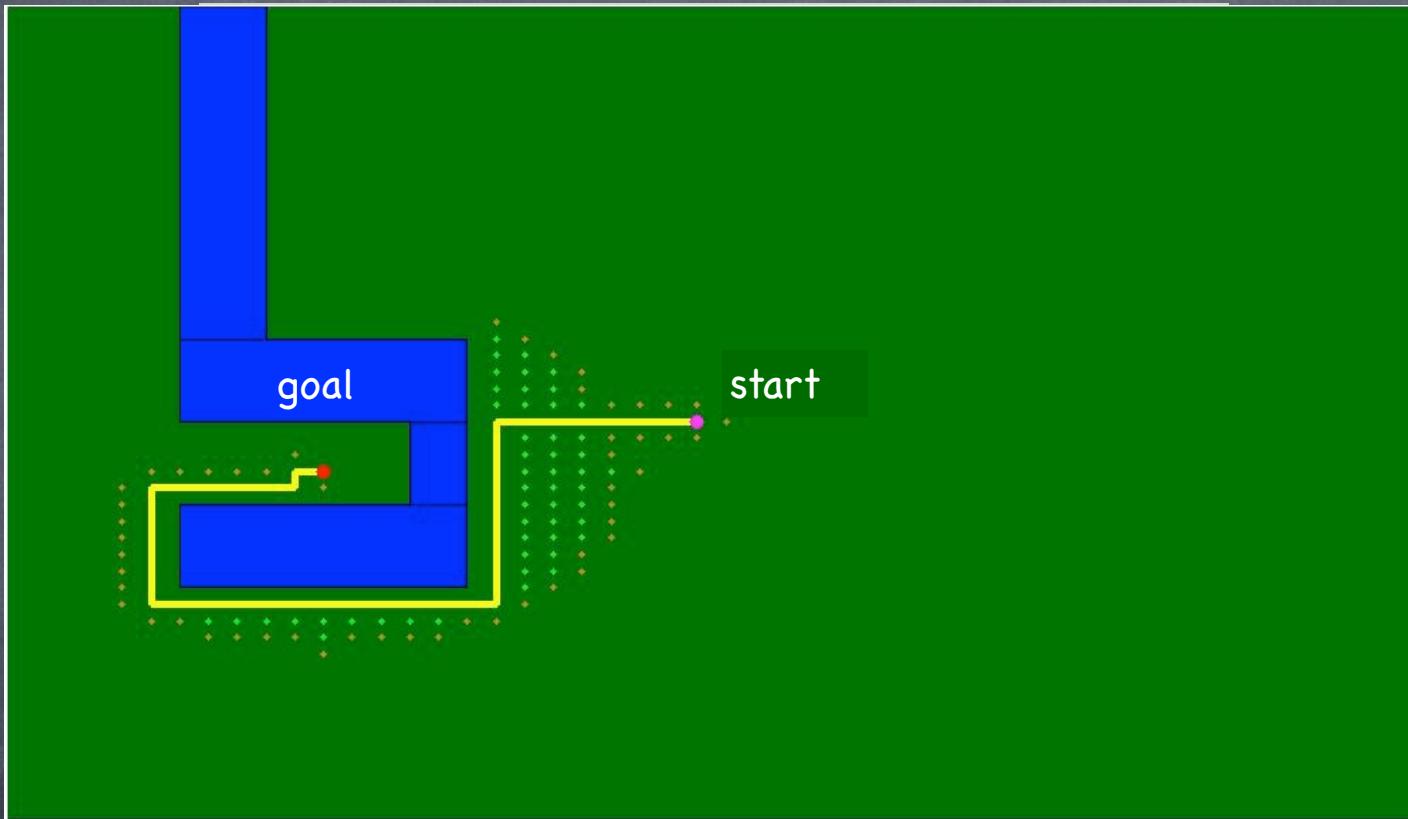
```
output  $\leftarrow \text{parent, distance}$ 
```

priority queue wrt. f_score
(implement min binary heap)



matlab example: gr.best-first

pathplan.m

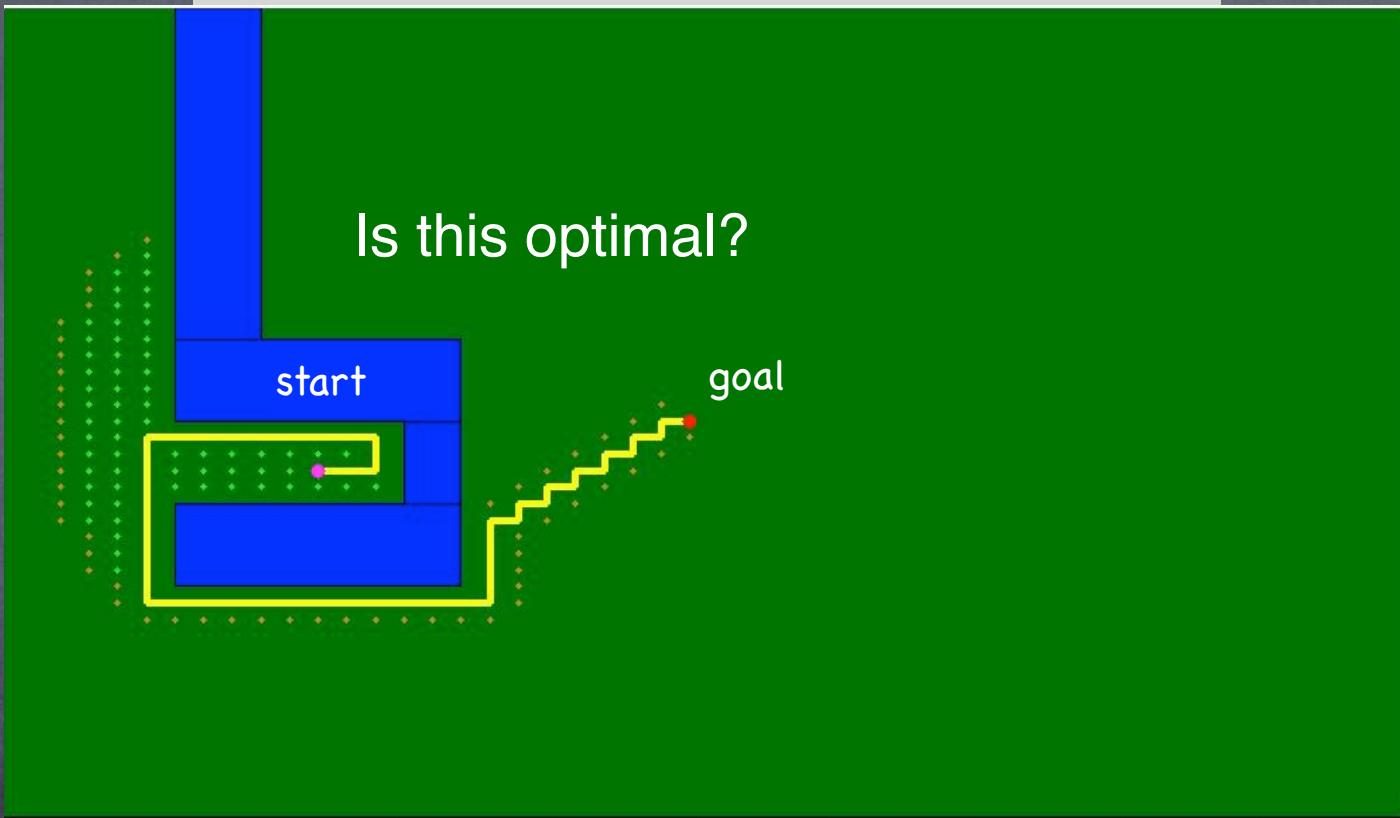


visited locations in light green
4-connected grid

UM EECS 398/598 - autorob.github.io

matlab example: gr.best-first

pathplan.m



visited locations in light green
4-connected grid

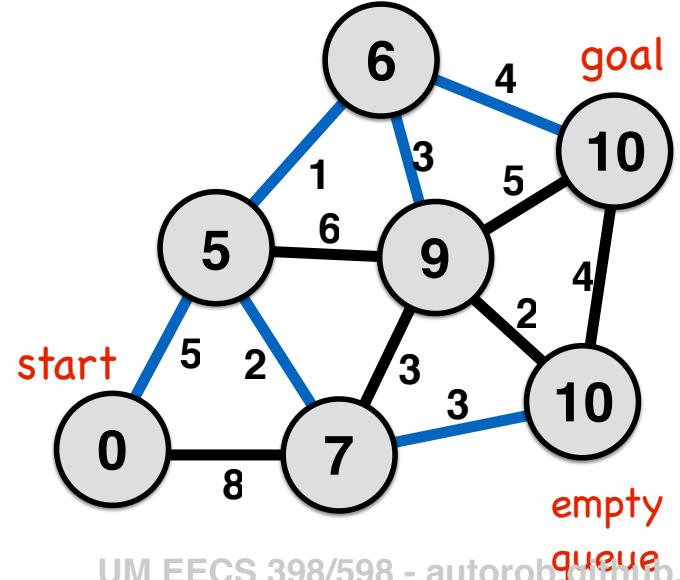
UM EECS 398/598 - autorob.github.io

Heaps and Priority Queues

A-star shortest path algorithm

```
all nodes  $\leftarrow \{dist_{start} \leftarrow \text{infinity}, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{false}\}$ 
start_node  $\leftarrow \{dist_{start} \leftarrow 0, parent_{start} \leftarrow \text{none}, visited_{start} \leftarrow \text{true}\}$ 
visit_queue  $\leftarrow \text{start\_node}$ 

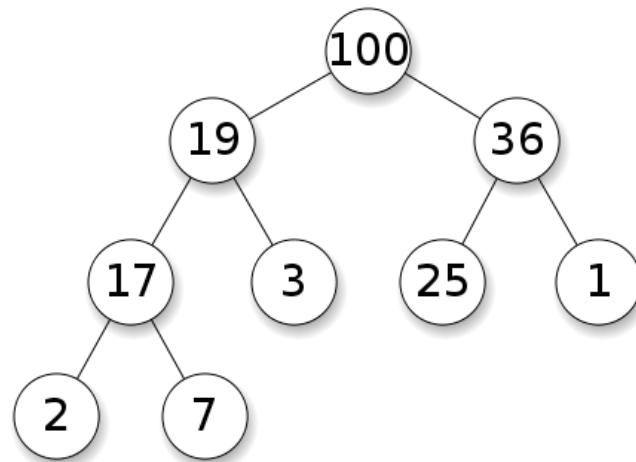
while (visit_queue != empty) && current_node != goal
    dequeue: cur_node  $\leftarrow f\_\text{score}(\text{visit\_queue})$  min binary heap for priority queue
    visitedcur_node  $\leftarrow \text{true}$ 
    for each nbr in not_visited(adjacent(cur_node))
        enqueue: nbr to visit_queue
        if distnbr > distcur_node + distance(nbr,cur_node)
            parentnbr  $\leftarrow \text{current\_node}$ 
            distnbr  $\leftarrow dist_{cur\_node} + distance(nbr,cur\_node)$ 
            f_score  $\leftarrow distance_{nbr} + line\_distance_{nbr,goal}$ 
        end if
    end for loop
end while loop
output  $\leftarrow parent, distance$ 
```



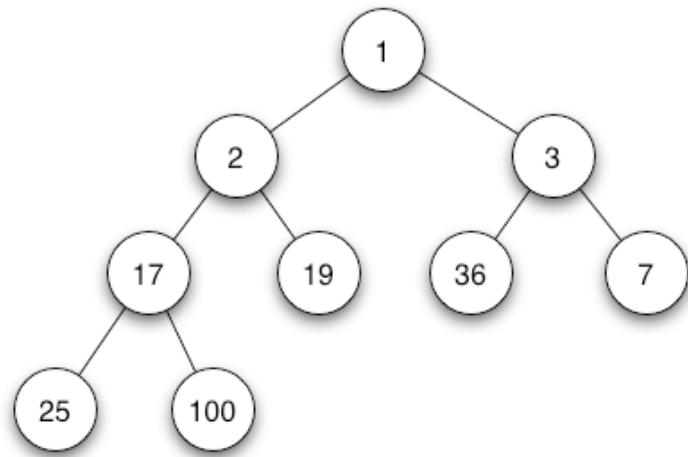
Binary Heaps

A heap is a tree-based data structure satisfying the heap property:
every element is greater (or less) than its children

Binary heaps allow nodes to have up to 2 children

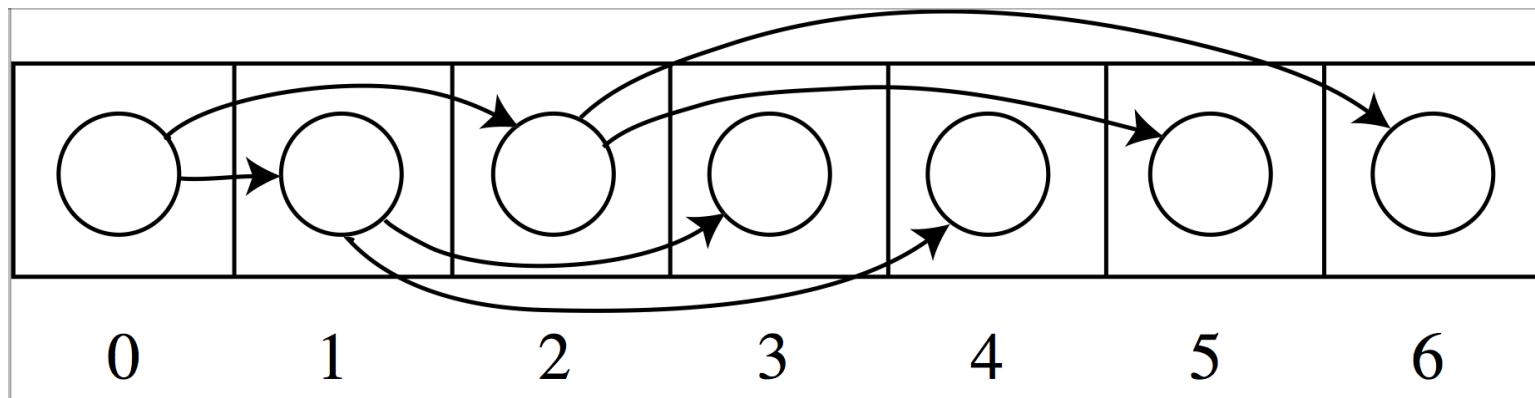


max heap



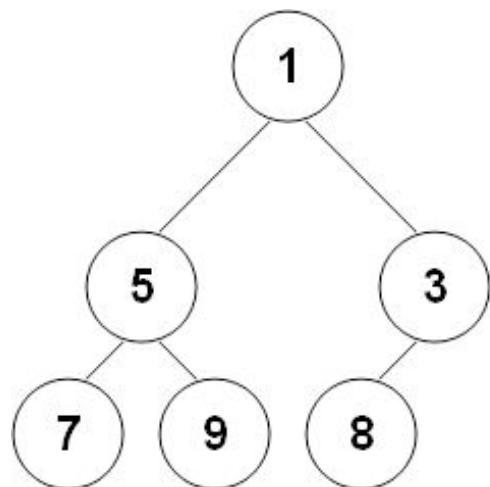
min heap

Heaps as arrays



- Heap element at array location i has
 - children at array locations $2i+1$ and $2i+2$
 - parent at $\text{floor}((i-1)/2)$

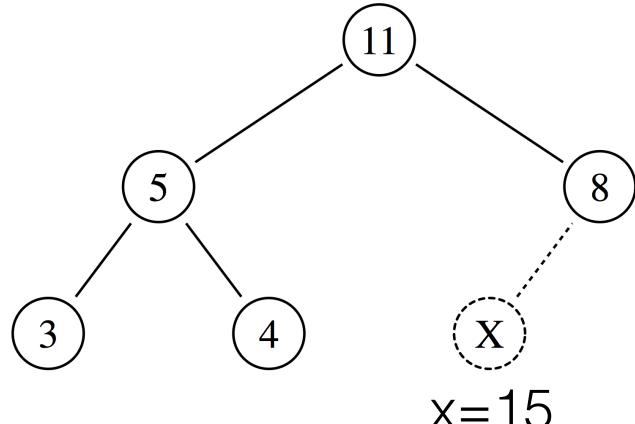
Heap array example



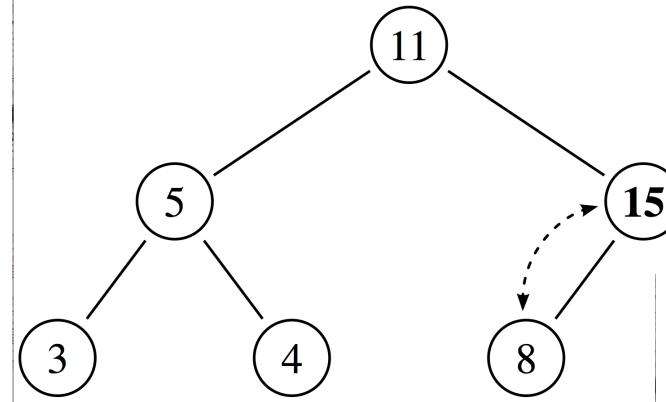
| Node | 1 | 5 | 3 | 7 | 9 | 8 |
|-------|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 |

Heap operations: Insert

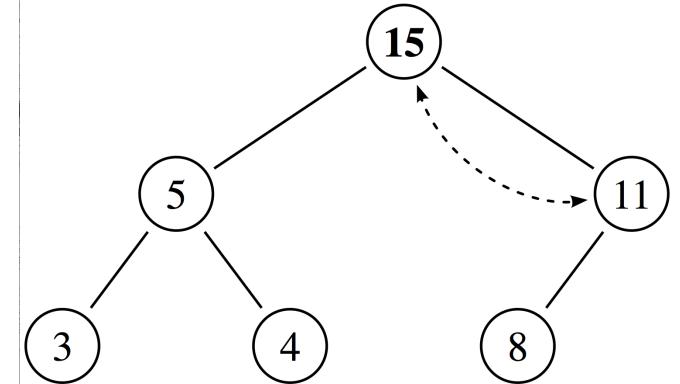
1) add new element to end of tree



2) swap with parent



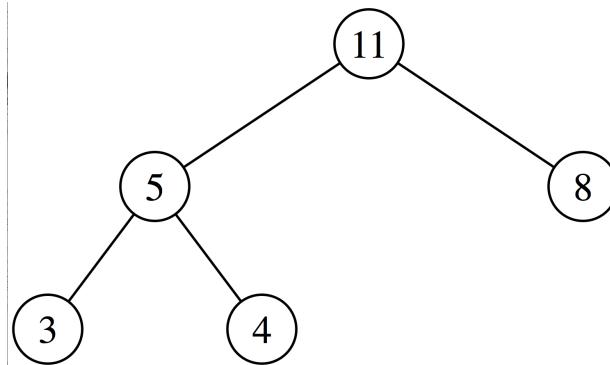
3) until heaped, do (2)



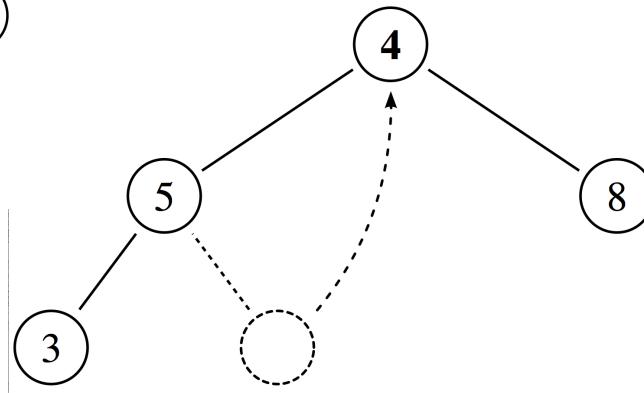
For priority queue, previously non-queued locations will be inserted with fscore priority

Heap operations: Extract

1) extract root element

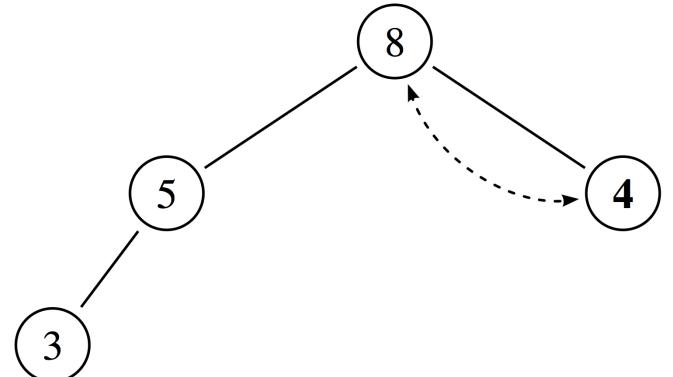


2) put last element at root



3) swap with higher priority child

4) until heaped, do (3)

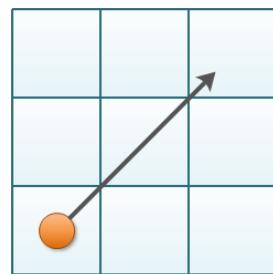


For priority queue, the root of the heap
will be the next node to visit

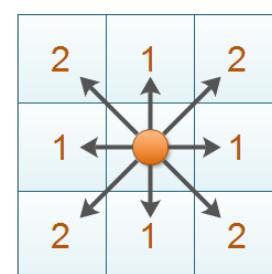
Questions to consider

- How many operations are needed for heap insertion and extraction?
- How much better is a min heap than an array wrt. # of operations?
- How should we measure distance on a uniform grid?
- Is a choice of distance measure both metric and admissible?

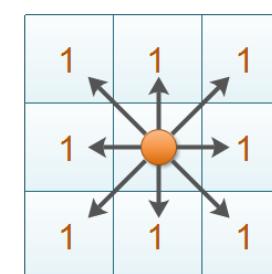
Euclidean Distance



Manhattan Distance



Chebyshev Distance



$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

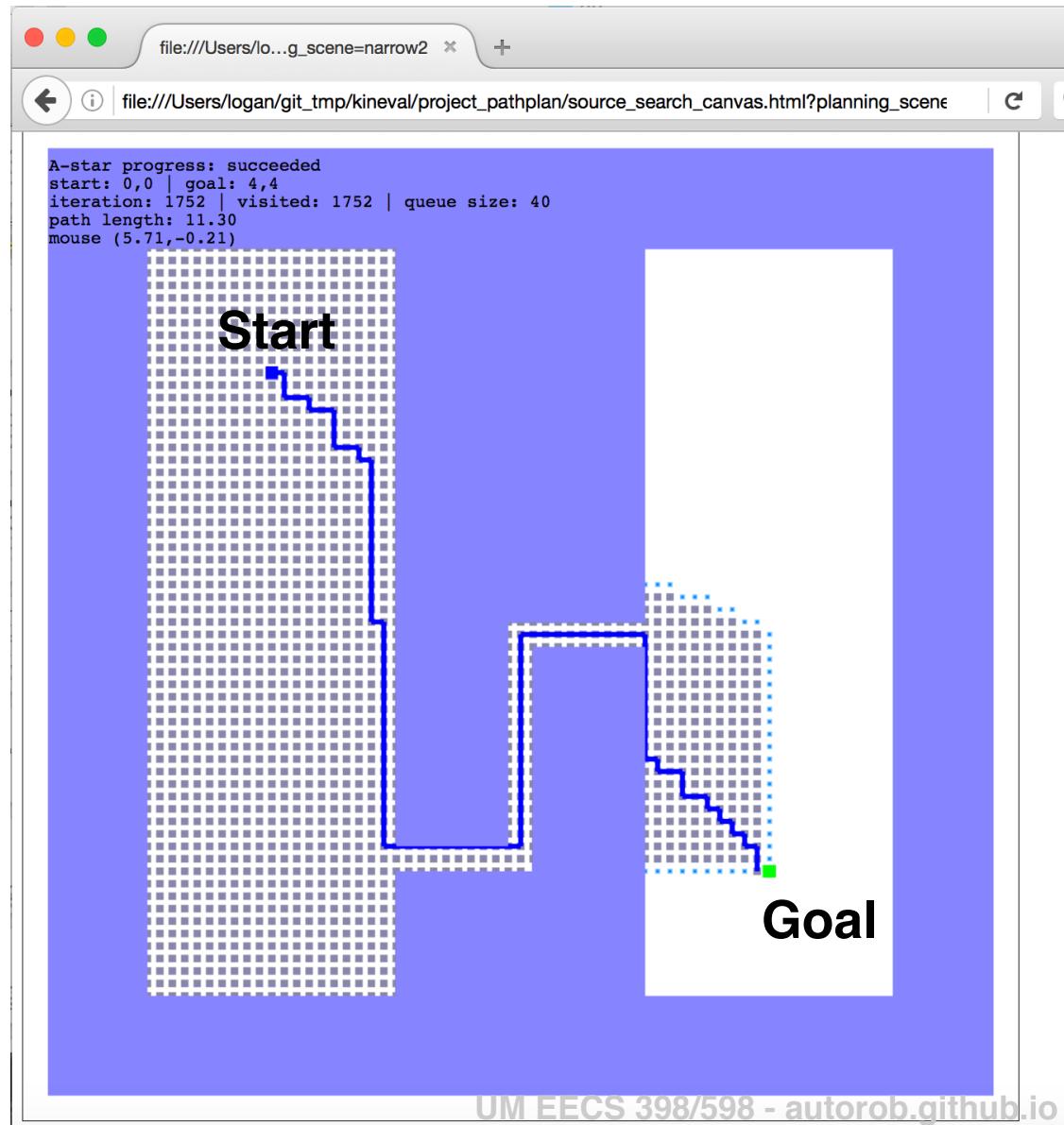
$$|x_1 - x_2| + |y_1 - y_2|$$

$$\max(|x_1 - x_2|, |y_1 - y_2|)$$

<https://lyfat.wordpress.com/2012/05/22/euclidean-vs-chebyshev-vs-manhattan-distance/>

Project 1: 2D Path Planning

- A-star algorithm for search in a given 2D world
- Implement in JavaScript/HTML5
- Heap data structure for priority queue
- Grads: DFS, BFS, Greedy
- Submit through your git repository



```
<html>
<title>How do we implement this planner?</title>

<body>
<h1>Next lecture:</h1>
<p>JavaScript/HTML5 and git Tutorial</p>

<a href="http://autorob.github.io">
EECS 398 Introduction to Autonomous Robotics <br>
ME/EECS 567 Robot Kinematics and Dynamics
</a>

</body>
</html>
```