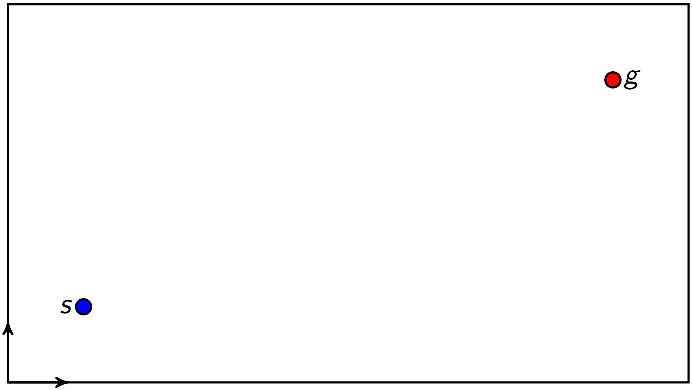
Motion Planning

Spring 2020

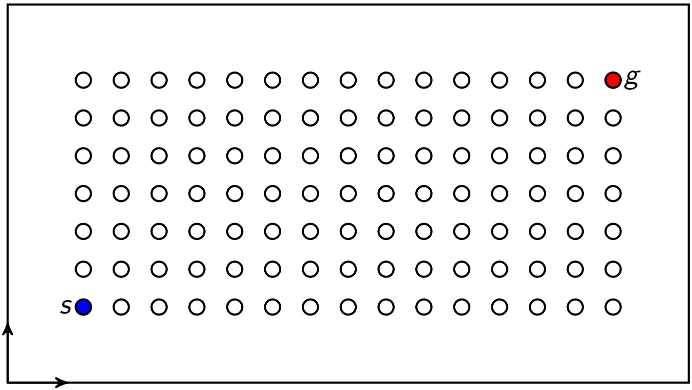
Hasan Poonawala

Graph Search



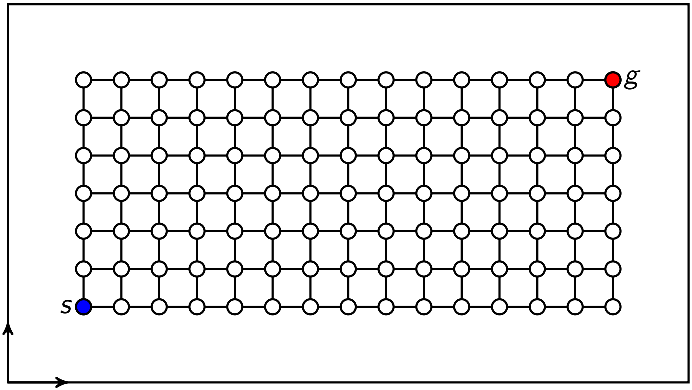
s = start, g = goal

Graph Search



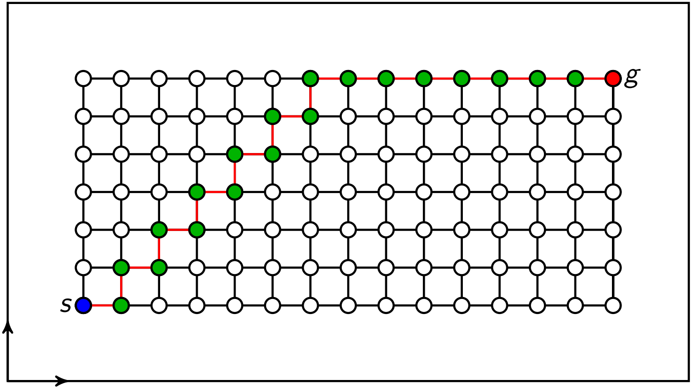
s = start, g = goal
Define nodes

Graph Search



s = start, g = goal
Define nodes
Define edges

Graph Search



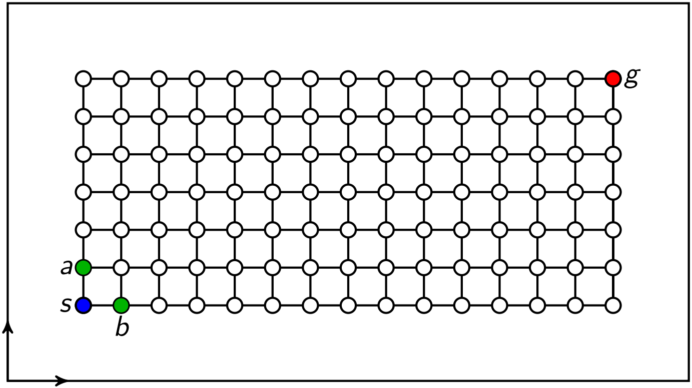
s = start, g = goal

Define nodes

Define edges

Find path(s)

Graph Search



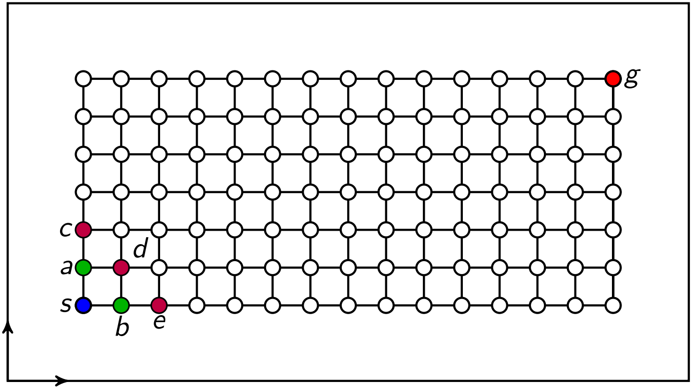
s = start, g = goal

Define nodes

Define edges

Let's focus on the nodes close to s

Graph Search



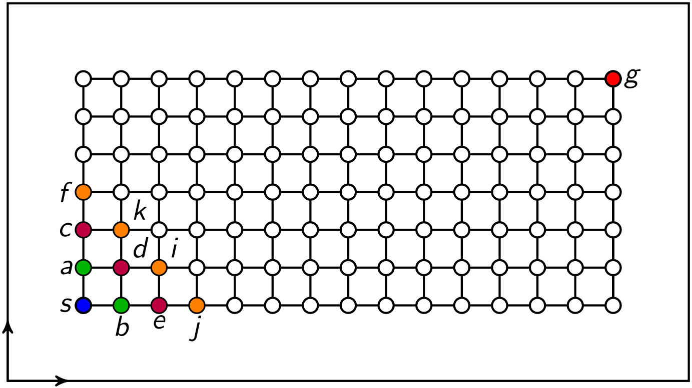
s = start, g = goal

Define nodes

Define edges

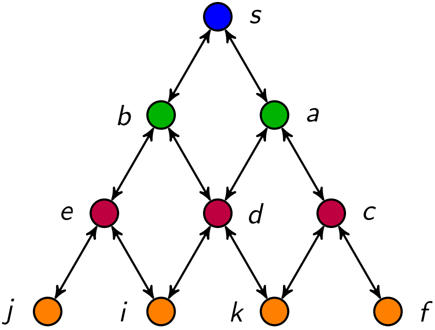
Let's focus on the nodes close to s

Graph Search



s = start, g = goal
Define nodes
Define edges
Let's focus on the nodes close to s

Graph As Seen From Start

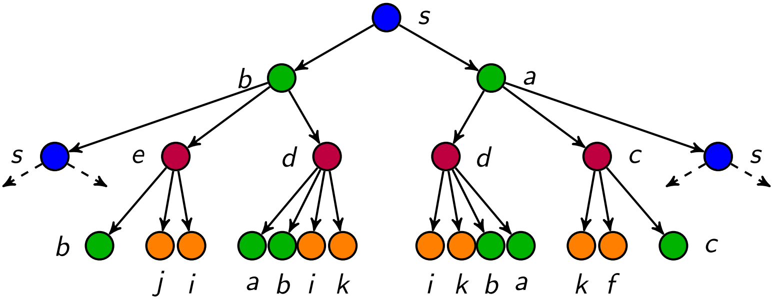


The undirected edges are equivalent to directed edges going forwards and back between two nodes.

We will use a search tree to traverse paths in this undirected graph.

Search Tree

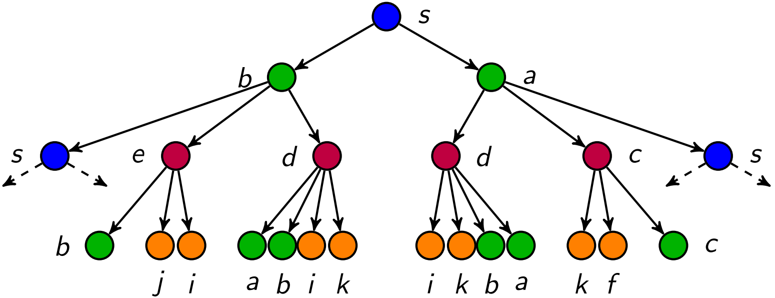
The root of the search tree is node s.
This tree depicts the possible paths starting from s.



Each 'layer' contains the nodes reached after taking a step from a node in a preceding layer.
Each layer has a depth, starting at 0 for s on top.
Node s appears at depth 0 and 2; b at 1 and 3, etc.

Search Tree

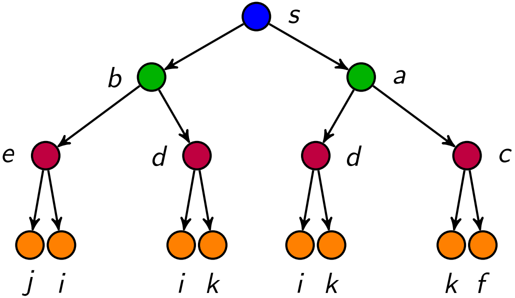
The root of the search tree is node s.
This tree depicts the possible paths starting from s.



The actual tree is much larger, with infinite depth.
For example, the tree recursively repeats itself whenever s appears at some depth.

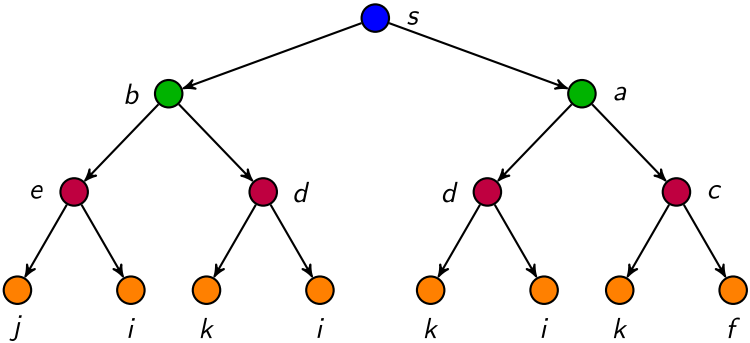
Search Tree

The root of the search tree is node s.
This tree depicts the possible paths starting from s.



If the edges in the graph were directed, we would never return to nodes located at smaller depths, simplifying the tree.

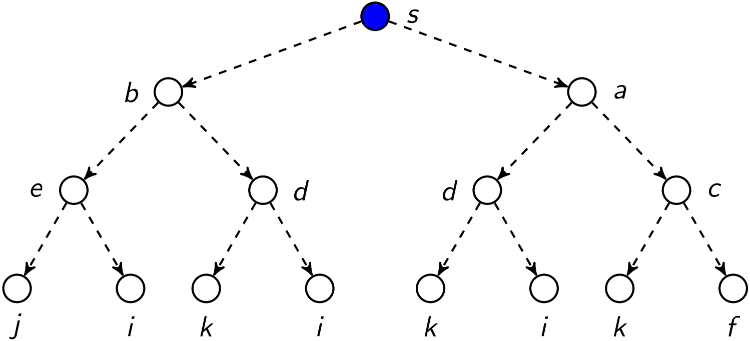
Search Tree



If a path exists from s to g, it is one of the possible paths in this search tree.

The key question is: how do we enumerate all the possible paths starting from s?

Search Algorithms



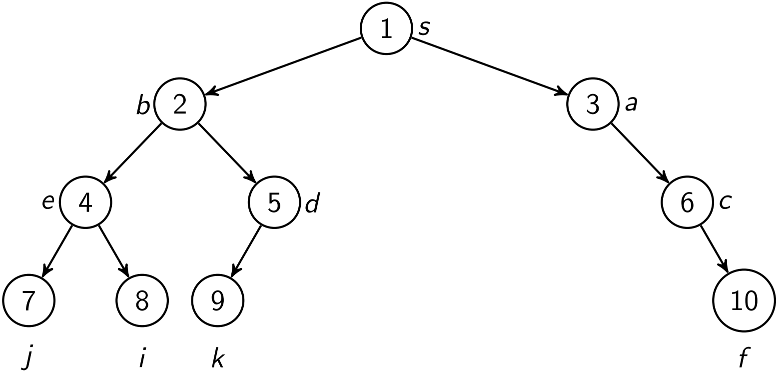
Search algorithms avoid representing all paths, or the entire tree, at one go.
Instead, the search tree is incrementally traversed.
Start with root node, and no other nodes/edges.

Search Algorithms

Search algorithms differ in the order in which paths are evaluated, which boils down to the order in which unvisited successor states are checked for being the goal.

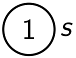
- Uninformed Search: Only know the successors at each node
  - Breadth First Search: Adds successors to end of list (Queue: first in first out)
  - Breadth First Search: Adds successors to start of list (Stack: first in last out)
- Informed Search: Use an estimate of potential value of a node to order unvisited nodes (neither FIFO nor FILO).

Breadth-First Search



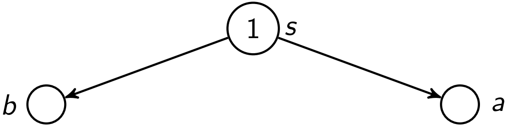
Order of visiting nodes under BFS. 'Row-by-row'

Breadth-First Search



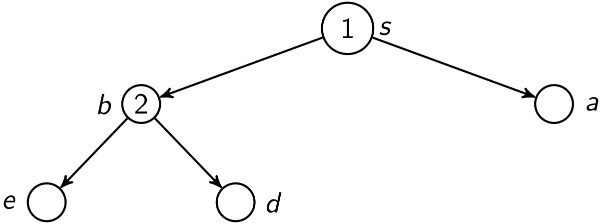
What the search tree looks like as it is built

Breadth-First Search



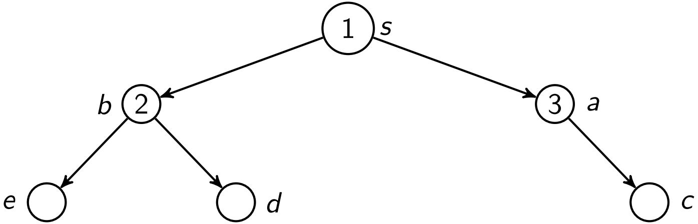
What the search tree looks like as it is built

Breadth-First Search



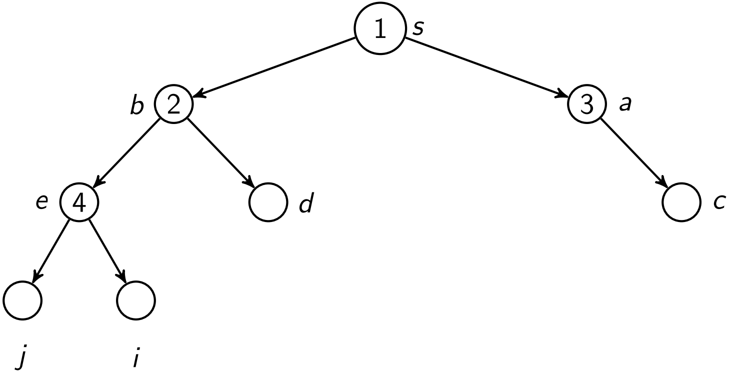
What the search tree looks like as it is built

Breadth-First Search



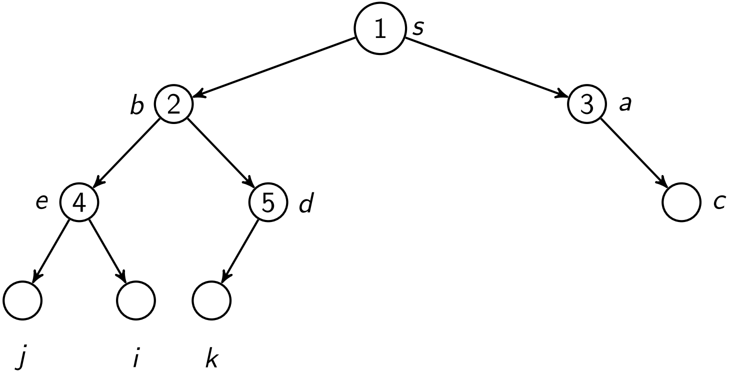
What the search tree looks like as it is built

Breadth-First Search



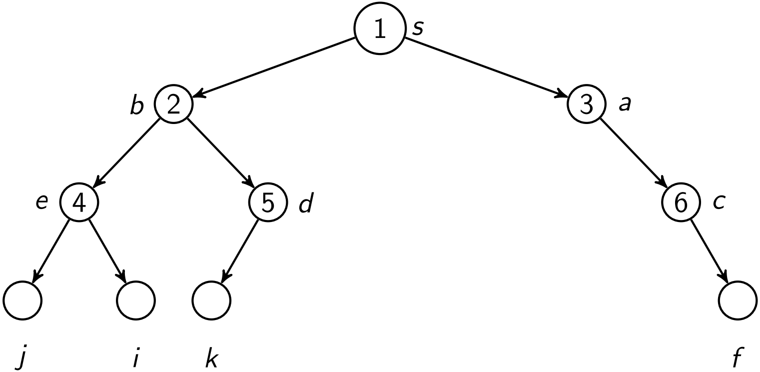
What the search tree looks like as it is built

Breadth-First Search



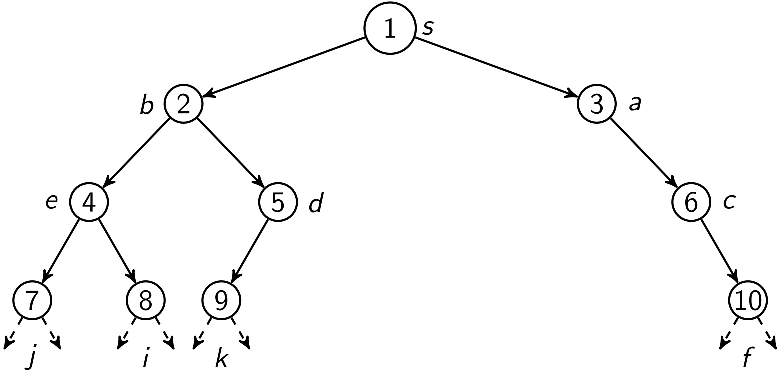
What the search tree looks like as it is built

Breadth-First Search



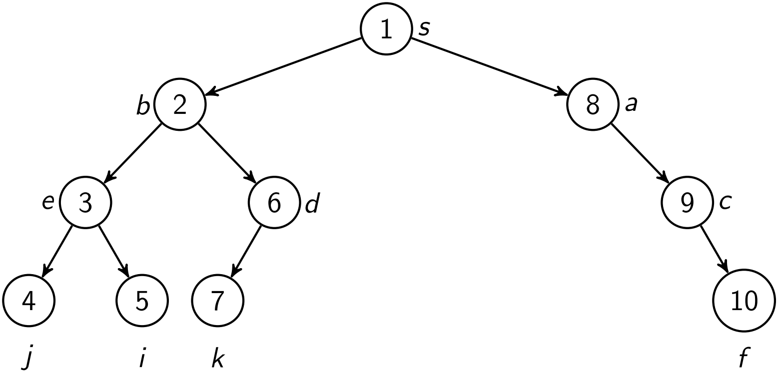
What the search tree looks like as it is built

Breadth-First Search



What the search tree looks like as it is built

Depth-First Search

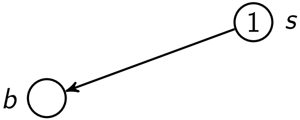


Order of visiting nodes under DFS. 'Column-by-column'

Depth-First Search

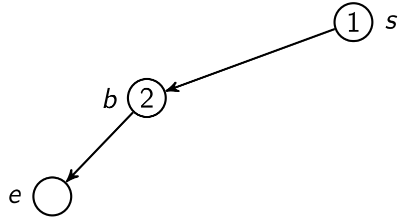
What the search tree looks like

Depth-First Search



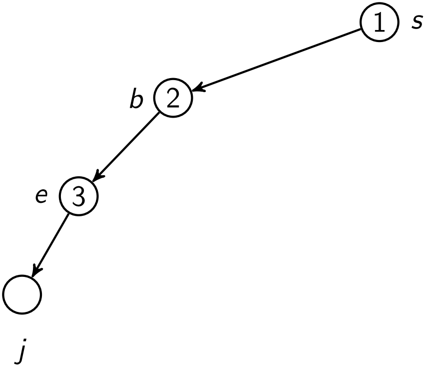
What the search tree looks like

Depth-First Search



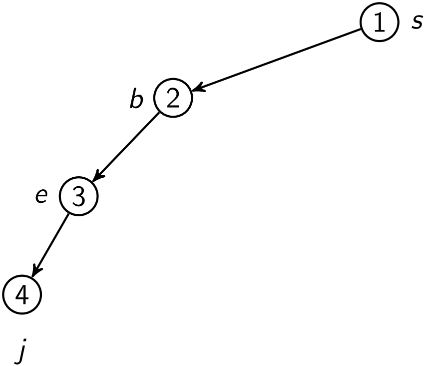
What the search tree looks like

Depth-First Search



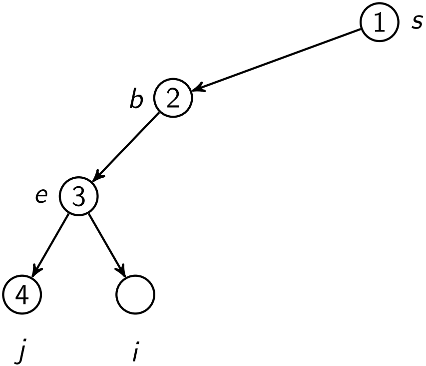
What the search tree looks like

Depth-First Search



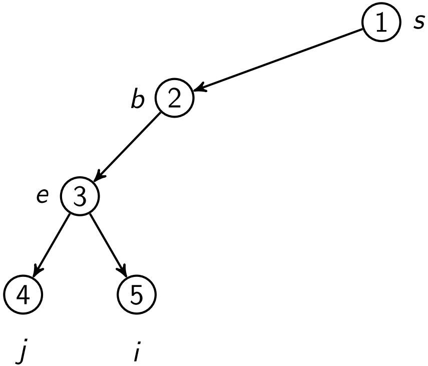
What the search tree looks like

Depth-First Search



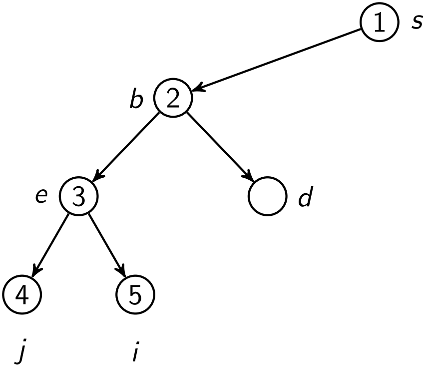
What the search tree looks like

Depth-First Search



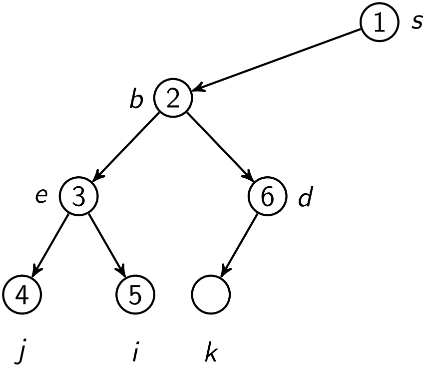
What the search tree looks like

Depth-First Search



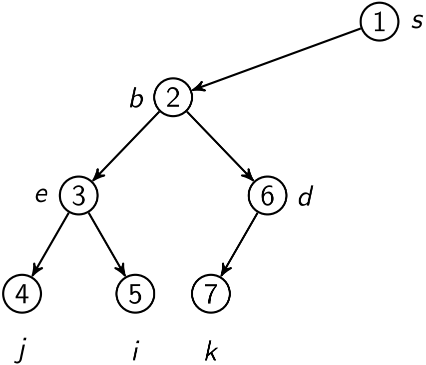
What the search tree looks like

Depth-First Search



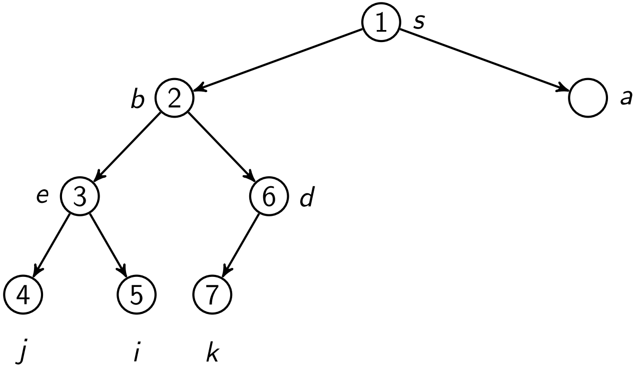
What the search tree looks like

Depth-First Search



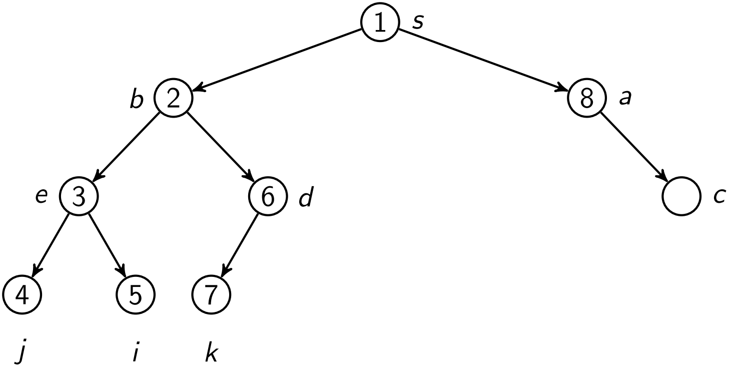
What the search tree looks like

Depth-First Search



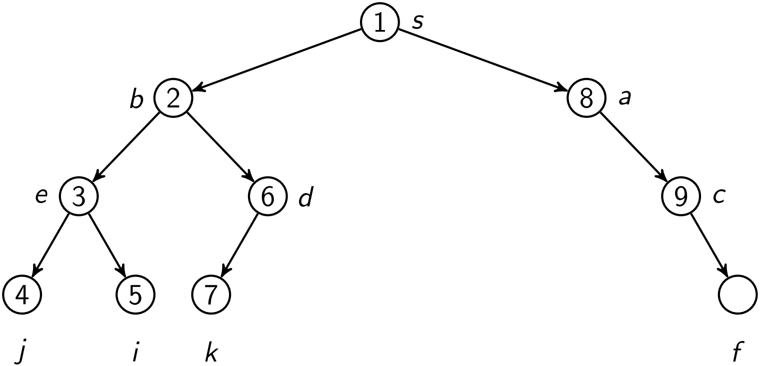
What the search tree looks like

Depth-First Search



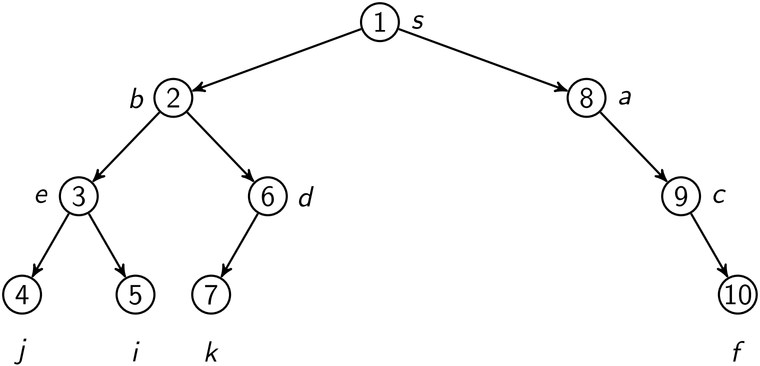
What the search tree looks like

Depth-First Search



What the search tree looks like

Depth-First Search



What the search tree looks like

Informed Search

- Breadth-First Search will find the goal, but it is slow when there are many successor states
- Depth-First Search might get stuck in loops, due to its implementation using recursion.
- While DFS and BFS creates an order between groups of successor nodes, they don't have a clear way to order the nodes within a group of successors
- Some estimate of how likely a node will lead to the goal would be a useful way to break ties
- This idea leads to informed search, where the estimate for each node is provided by a *heuristic* function

Two IS Algorithms

There are two quantities we can assign to a node:

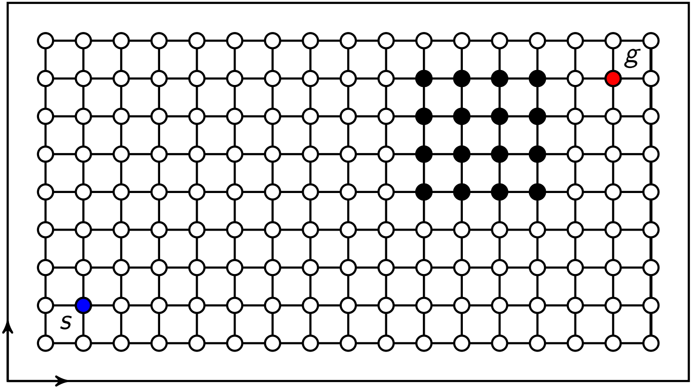
- The lowest cost $c(n)$ of the path – given the search tree uncovered so far – from root s to node n.
- The estimated least cost over all paths from n to goal g. We don't know this least-cost path, but we let estimate $h(n)$ be a guess of its true value $h^\star(n)$ (unknown to us).

These quantities lead to two algorithms:

1. Best-First Search: order using $c(n)$
2. A*: order using $c(n) + h(n)$. Guaranteed performance when

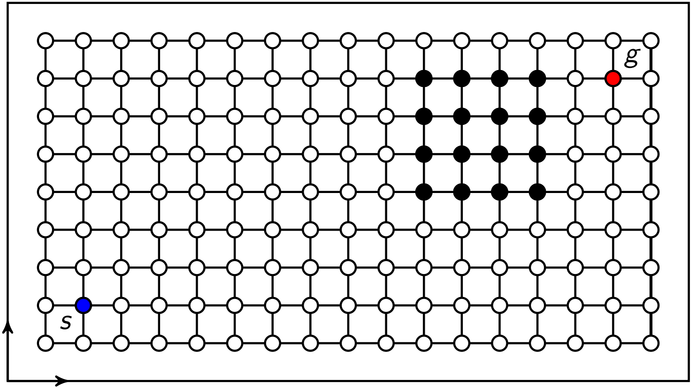
$$h(n) \leq h^\star(n).$$

Motion Planning Discrete Space



Black nodes are 'obstacle' nodes

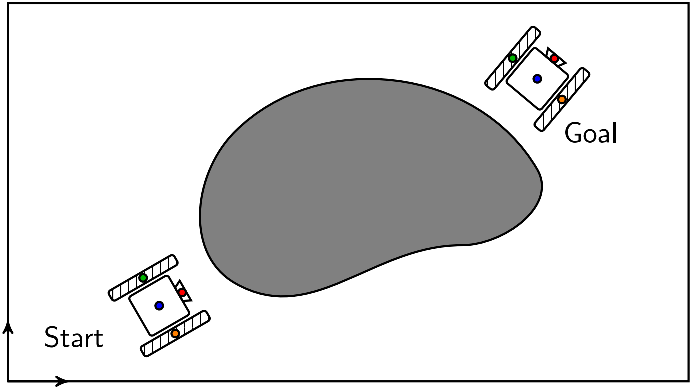
Motion Planning Discrete Space



Black nodes are ‘obstacle’ nodes

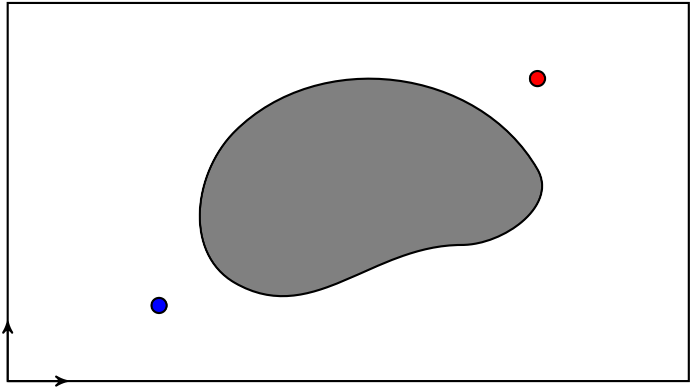
While we ‘see’ the graph all at once, our algorithms deal with this graph using a search tree

Motion Planning



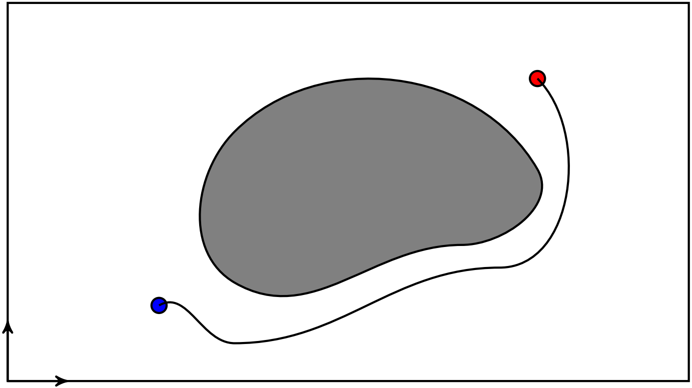
Motion Planning Problem

Motion Planning



Over-simplify the problem

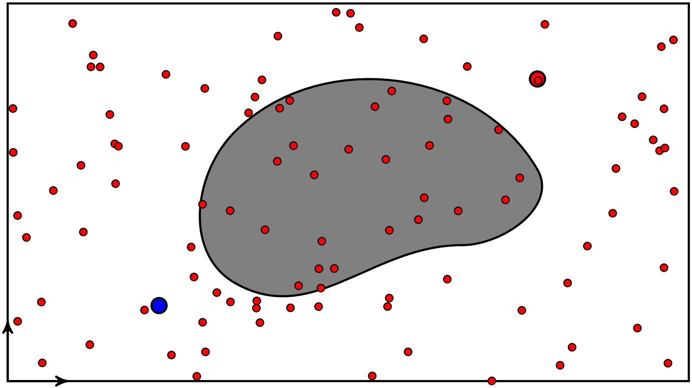
Motion Planning



A valid continuous path

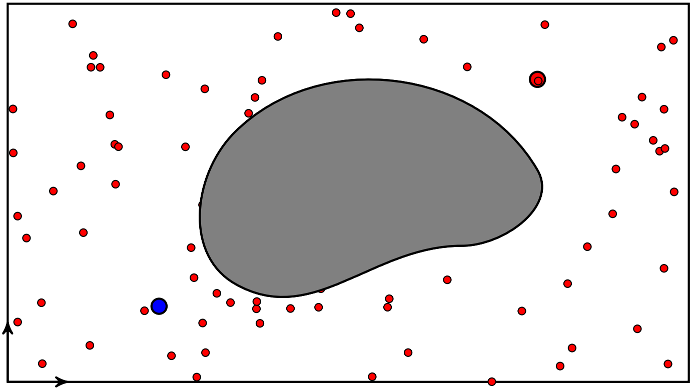
How would we obtain such a path using graph search?

Motion Planning



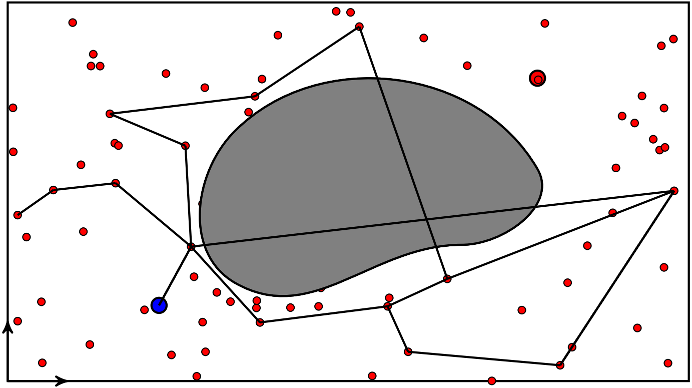
Randomly pick configurations to be nodes

Motion Planning



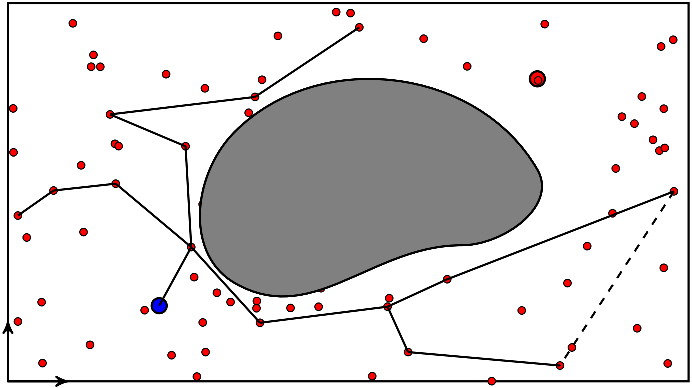
Randomly pick configurations to be nodes
Discard nodes in obstacles

Motion Planning



Randomly pick configurations to be nodes
Discard nodes in obstacles
Build graph by adding edges

Motion Planning



Randomly pick configurations to be nodes
Discard nodes in obstacles
Build graph by adding edges that can be physically realized

Motion Planning Continuous Space

Sampling-based motion planning (MP) algorithms define nodes/edges for continuous space and then develop a graph (PRM/RRG) or a tree (RRT).

- PRM: Probabilistic Road Map
- RRT: Rapidly-Expanding Random Tree
- RRG: Rapidly-Expanding Random Graph

Motion Planning Continuous Space

Sampling-based motion planning (MP) algorithms define nodes/edges for continuous space and then develop a graph (PRM/RRG) or a tree (RRT).

- PRM: Probabilistic Road Map
- RRT: Rapidly-Expanding Random Tree
- RRG: Rapidly-Expanding Random Graph

This conversion of MP into a graph enables use of graph search algorithms

Motion Planning Continuous Space

Sampling-based motion planning (MP) algorithms define nodes/edges for continuous space and then develop a graph (PRM/RRG) or a tree (RRT).

- PRM: Probabilistic Road Map
- RRT: Rapidly-Expanding Random Tree
- RRG: Rapidly-Expanding Random Graph

This conversion of MP into a graph enables use of graph search algorithms

The large variety in sampling-based motion planning algorithms are variations of the two following steps.

1. Randomly sample configurations to create 'nodes'
2. Use motion models/constraints to 'connect' samples

Every algorithm has

- Sampling mechanism + collision check for creating nodes
- Select existing nodes to try and connect new samples to
- Local planner to check if we can connect new sample with selected existing nodes (dynamics, obstacles along path, local planner, etc.)

Motion Planning Continuous Space

Evolution:

- Early methods generated new sample by randomly choosing a control and 'taking a step'. Poor exploration.

Motion Planning Continuous Space

Evolution:

- Early methods generated new sample by randomly choosing a control and 'taking a step'. Poor exploration.
- PRM said choose a configuration, use planner to figure out control between them. Often connections failed.

Motion Planning Continuous Space

Evolution:

- Early methods generated new sample by randomly choosing a control and 'taking a step'. Poor exploration.
- PRM said choose a configuration, use planner to figure out control between them. Often connections failed.
- RRT said create new sample, then create sample corresponding to step from existing node towards new sample. Rapid exploration unlocked.

Motion Planning Continuous Space

Evolution:

- Early methods generated new sample by randomly choosing a control and ‘taking a step’. Poor exploration.
- PRM said choose a configuration, use planner to figure out control between them. Often connections failed.
- RRT said create new sample, then create sample corresponding to step from existing node towards new sample. Rapid exploration unlocked.
- RRT*: rewire connections so that all paths in **tree** are optimal

Motion Planning Continuous Space

Evolution:

- Early methods generated new sample by randomly choosing a control and 'taking a step'. Poor exploration.
- PRM said choose a configuration, use planner to figure out control between them. Often connections failed.
- RRT said create new sample, then create sample corresponding to step from existing node towards new sample. Rapid exploration unlocked.
- RRT*: rewire connections so that all paths in **tree** are optimal
- RRG: Connect to multiple neighbors, use shortest-path algos later.