CS-541: Artificial Intelligence
Lecture 4a

Abdul Rafae Khan
Department of Computer Science
Stevens Institute of Technology
akhan4@stevens.edu

February 14, 2022

Neural Network Recap

$f : \mathbb{R} \mapsto \mathbb{R}$

input: $x^{(0)} \in \mathbb{R}$

$z^{(1)} = w^{(0)} x^{(0)}$

$x^{(1)} = \max\{0, z^{(1)}\}$

$z^{(2)} = w^{(1)} x^{(1)}$

$x^{(2)} = \max\{0, z^{(2)}\}$

$z^{(3)} = w^{(2)} x^{(2)}$

output: $f(x^{(0)}) = z^{(3)} \in \mathbb{R}$

loss: $L = \frac{1}{2}(f(x^{(0)}) - y)^2$

Neural Network Recap

$$f : \mathbb{R} \mapsto \mathbb{R}$$

input: $x^{(0)} \in \mathbb{R}$

$$z^{(1)} = w^{(0)} x^{(0)}$$

$$x^{(1)} = \max\{0, z^{(1)}\}$$

$$z^{(2)} = w^{(1)} x^{(1)}$$

$$x^{(2)} = \max\{0, z^{(2)}\}$$

$$z^{(3)} = w^{(2)} x^{(2)}$$

output: $f(x^{(0)}) = z^{(3)} \in \mathbb{R}$

loss: $L = \frac{1}{2}(f(x^{(0)}) - y)^2$

$$\frac{\partial L}{\partial z^{(3)}} =?$$

$$\frac{\partial L}{\partial z^{(2)}} =?$$

$$\frac{\partial L}{\partial w^{(2)}} =?$$

$$\frac{\partial L}{\partial z^{(1)}} =?$$

$$\frac{\partial L}{\partial w^{(1)}} =?$$

$$\frac{\partial L}{\partial w^{(0)}} =?$$

Neural Network Recap

$f : \mathbb{R} \mapsto \mathbb{R}$

input: $x^{(0)} \in \mathbb{R}$

$z^{(1)} = w^{(0)} x^{(0)}$

$x^{(1)} = \max\{0, z^{(1)}\}$

$z^{(2)} = w^{(1)} x^{(1)}$

$x^{(2)} = \max\{0, z^{(2)}\}$

$z^{(3)} = w^{(2)} x^{(2)}$

output: $f(x^{(0)}) = z^{(3)} \in \mathbb{R}$

loss: $L = \frac{1}{2}(f(x^{(0)}) - y)^2$

$$\frac{\partial L}{\partial z^{(3)}} = z^{(3)} - y$$

$$\frac{\partial L}{\partial z^{(2)}} = \frac{\partial z^{(3)}}{\partial z^{(2)}} \boxed{\frac{\partial L}{\partial z^{(3)}}} \quad \frac{\partial L}{\partial w^{(2)}} = \frac{\partial z^{(3)}}{\partial w^{(2)}} \boxed{\frac{\partial L}{\partial z^{(3)}}}$$

$$\frac{\partial L}{\partial z^{(1)}} = \frac{\partial z^{(2)}}{\partial z^{(1)}} \boxed{\frac{\partial L}{\partial z^{(2)}}} \quad \frac{\partial L}{\partial w^{(1)}} = \frac{\partial z^{(2)}}{\partial w^{(1)}} \boxed{\frac{\partial L}{\partial z^{(2)}}}$$

$$\frac{\partial L}{\partial w^{(0)}} = \frac{\partial z^{(1)}}{\partial w^{(0)}} \boxed{\frac{\partial L}{\partial z^{(1)}}}$$

Overview

1. **Planning Agent**
2. **Formulating Search Problem**
3. **Search Strategy**
4. **State Space vs Search Tree**
5. **Search Algorithms**
   5.1 Backtracking Search
   5.2 DFS
   5.3 BFS
   5.4 IDS
6. **Search - Dynamic Programming**
7. **Uniform Cost Search**

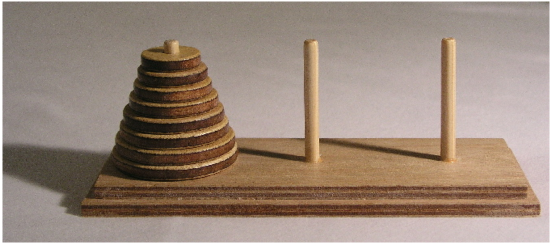
Planning Agent

Construct plans to achieve its goal and execute them
Analyze a situation and develop a strategy for achieving the goal.
Finding a sequence of actions for a desired outcome.
Search agents typically assume

- static world
- observable environment
- discrete states
- deterministic transition model

Revisiting Tower of Hanoi

According to a legend, there is a temple in India with a spacious room.
In the center are 64 golden discs stacked on 3 posts
Young priests are assigned the duty to move disks from one post to another
Larger disk cannot come above a smaller disc
The legend says, the world will end when the priest re-create the stack on another post



Does a solution exist? What is the cost?

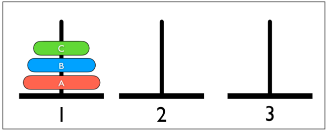
Formulating as Search Problem

state representation: How will a state be represented in code?
start state: What is the initial state?
goal state: What is the final state?
action: What are the possible movements from one state to another?
cost: What is the cost for an action?

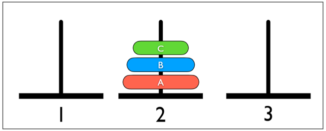
Formulating as Search Problem

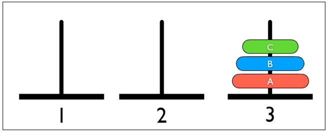
state representation: 3 lists

start state:



goal state: can be multiple goal states





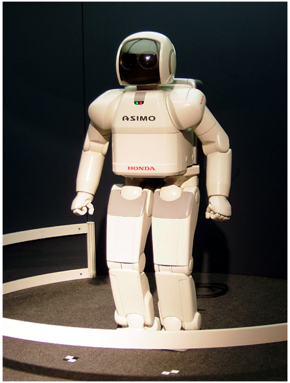
action: movement of a disc

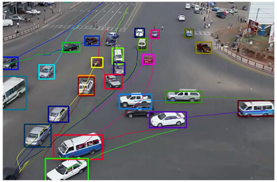
cost: 1 for each action

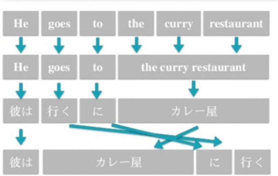
Formulating a Search Problem

start state: s_{start}
goal state: s_{end}
action(s): all possible actions that can be perform at state s
cost(s, a): cost associated with action a at state s
successor(s, a): returns the new state after taking action a at state s

Search Applications







Wolf, goat and cabbage problem

Cross all three across the river
Wolf can eat the goat if left alone
Goat can eat the cabbage if left alone

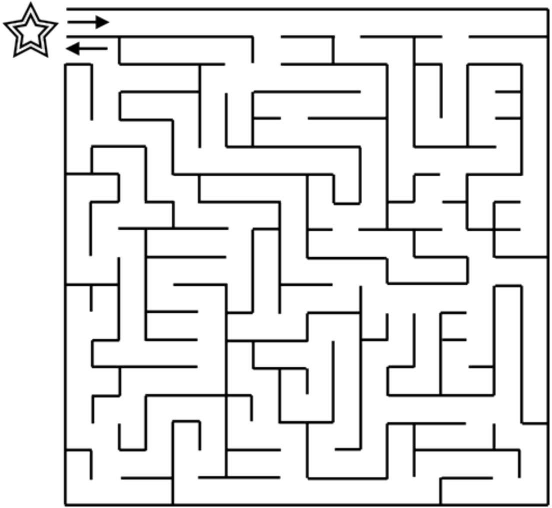
Wolf, goat and cabbage problem

state representation: ?
start state: ?
goal state: ?
actions: ?
cost: ?

Wolf, goat and cabbage problem

state representation: 2 lists
start state: All three on the left side of the river
goal state: All three on the right side of the river
actions: Move across the river
cost: 1

Maze



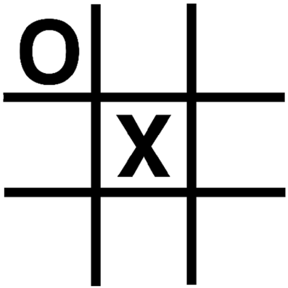
Maze

state representation: ?
start state: ?
goal state: ?
actions: ?
cost: ?

Maze

state representation: $m \times n$ matrix
start state: root node
goal state: some leaf node
actions: left, right, up, down
cost: 1 for each edge traversal

Tic-Tac-Toe



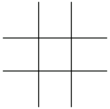
Tic-Tac-Toe

state representation: ?
start state: ?
goal state: ?
actions: ?
cost: ?

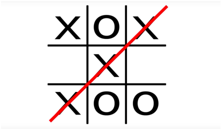
Tic-Tac-Toe

state representation: 3×3 matrix of 0s & 1s

start state: empty grid



goal state: three Os or Xs in a line

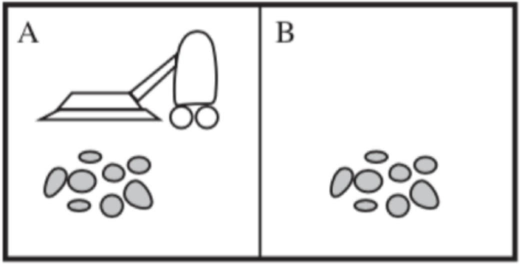


action: Add an X or O

cost: 0 or 1

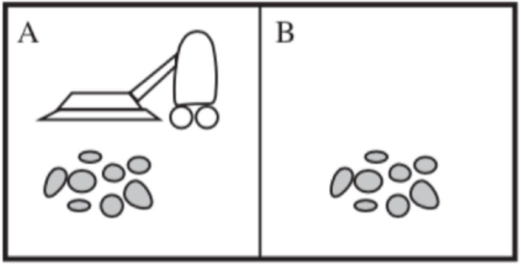
Toy Vacuum Problem

Vacuum cleaner world with only two rooms!



Vacuum Cleaner Problem

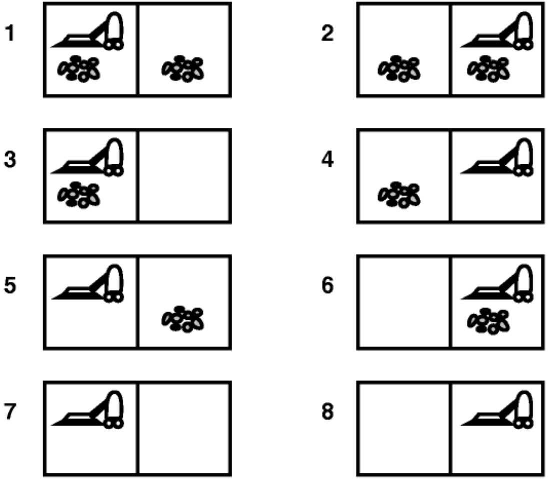
Vacuum cleaner world with only two rooms!



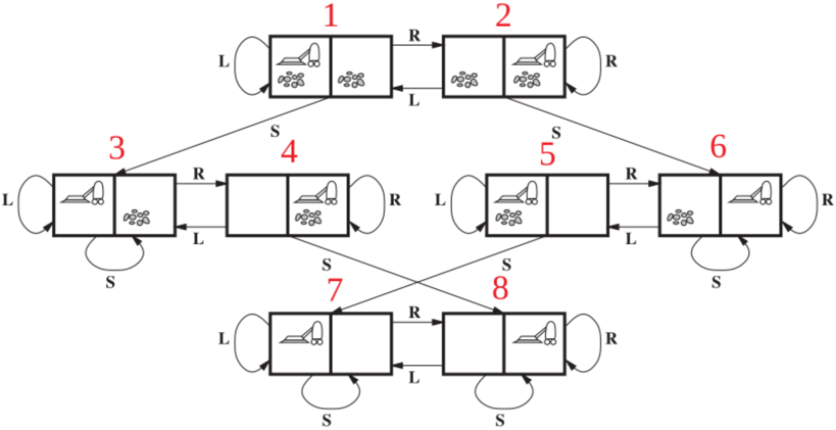
state representation: 3-tuple
start state: position of vacuum, position of dirt
goal state: No dirt
actions: Left, Right, Suck
cost: 1 for each step

Vacuum Cleaner Problem

8 possible states!

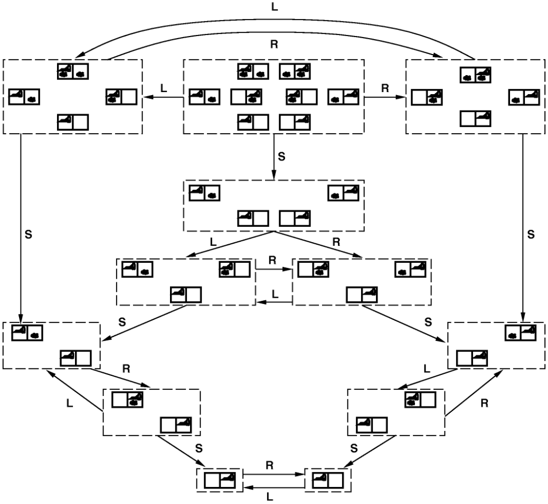


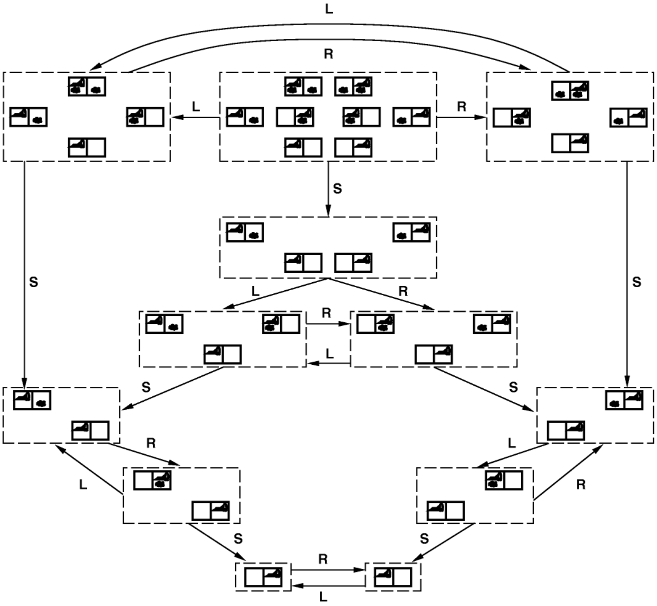
Vacuum Cleaner Problem



Vacuum Cleaner Problem

If no information of the environment
Multi-state problem
e.g. the vacuum has no senors




L
R
L
R
S
S
S
L
R
R
L
S
S
R
L
L
R
S
S
R
L

Search Problems

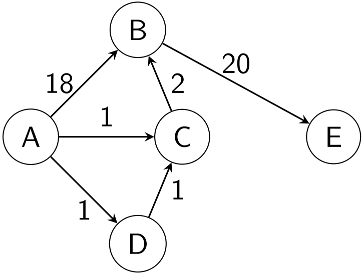
How to solve such problems?

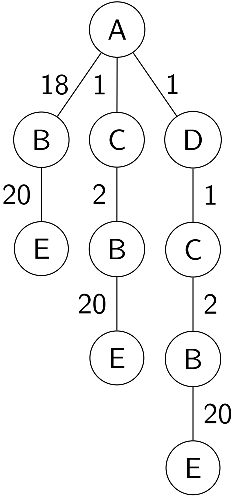
Search Problems

How to solve such problems?
Traverse all the states until you reach the goal state

Search Space vs Search Tree

start state: A
goal state: B



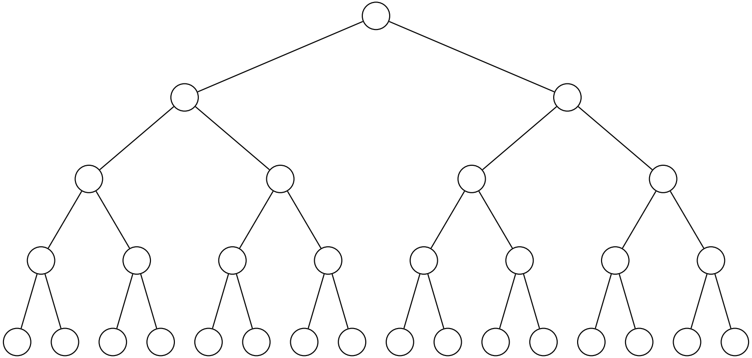


Search Problems

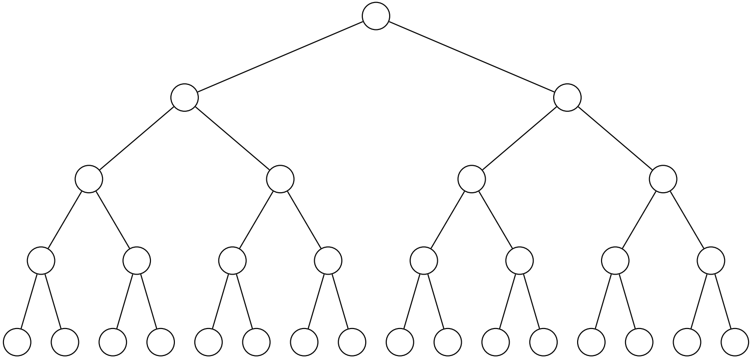
When developing a search strategy, consider
Complete: Will the solution be found?
Optimality: Will the optimal solution be found?
Time Complexity: How much time will it take?
Space Complexity: How much space will it require?
Informed/Uninformed: Does the search use additional domain specific information

Solving a Search Problem

One idea is a search tree

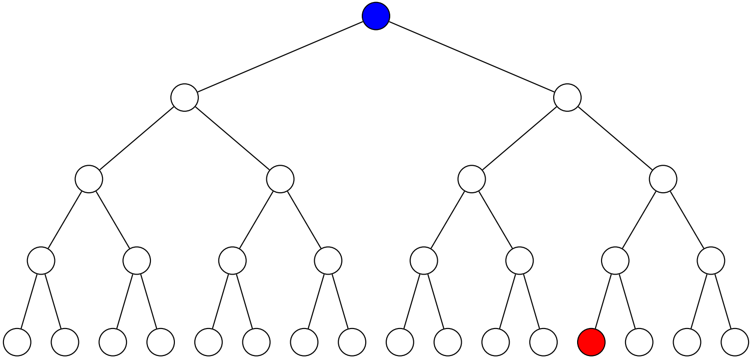


Tree Search



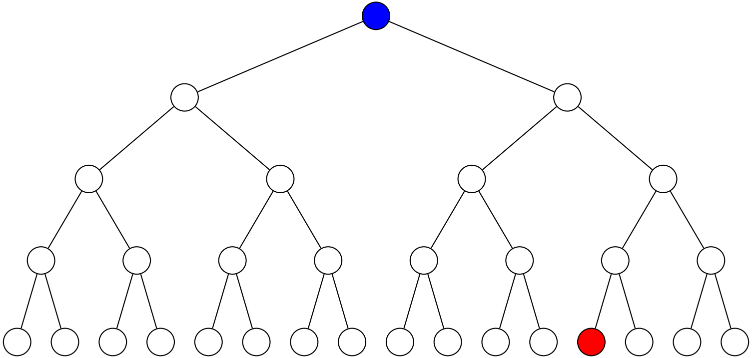
Each node represents a state
Each edge represents an action from one state to another

Tree Search



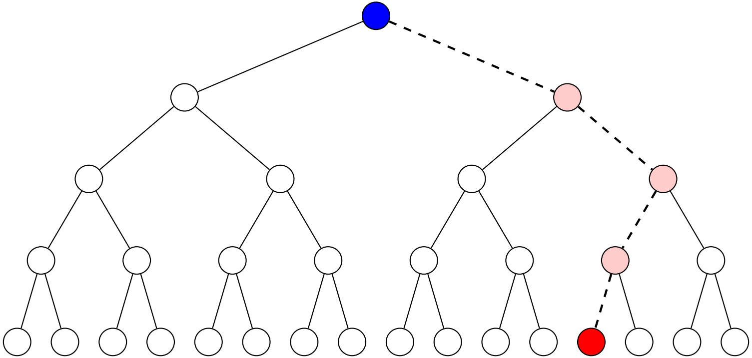
start state: blue
goal state: red

Tree Search



branching factor: number of children for each node (b)
depth: number of edges from root to leaf (D)

Tree Search



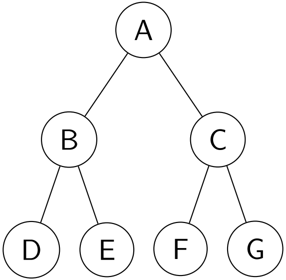
The solution is the sequence of edges (actions) from start state to goal state

Tree Search

Lets look at a small example

start state: A

goal state: F



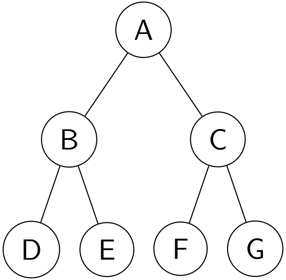
Tree Search

Look at the complete tree for the solution

Cost: Any

start state: A

goal state: F



Whiteboard

Tree Search

Algorithm	Cost	Time	Space
Backtracking Search	Any	$O(b^D)$	$O(D)$

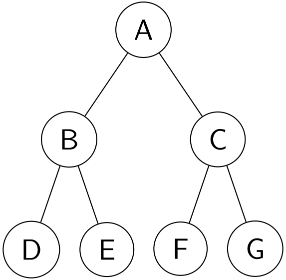
Depth First Search

Similar to backtracking search with early stop

Cost: 0

start state: A

goal state: F



Whiteboard

Tree Search

Algorithm	Cost	Time	Space
Backtracking Search	Any	$O(b^D)$	$O(D)$
DFS	0	$O(b^D)$	$O(D)$

b = branching factor
D = total depth of tree
d = depth of solution

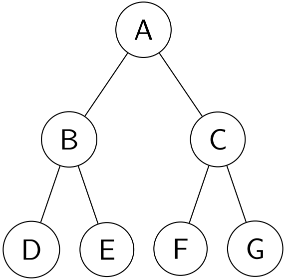
Breadth First Search

Search all the adjacent nodes first

Cost: ≥ 0

start state: A

goal state: F



Whiteboard

Tree Search

Algorithm	Cost	Time	Space
Backtracking Search	Any	$O(b^D)$	$O(D)$
DFS	0	$O(b^D)$	$O(D)$
BFS	≥ 0	$O(b^d)$	$O(b^d)$

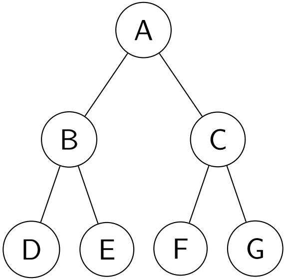
b = branching factor
D = total depth of tree
d = depth of solution

Iterative Deepening Search

Search all the adjacent nodes first

start state: A

goal state: F



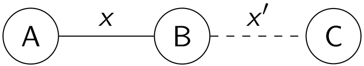
Whiteboard

Tree Search

Algorithm	Cost	Time	Space
Backtracking Search	Any	$O(b^D)$	$O(D)$
DFS	0	$O(b^D)$	$O(D)$
BFS	≥ 0	$O(b^d)$	$O(b^d)$
IDS	≥ 0	$O(b^d)$	$O(d)$

b = branching factor
D = total depth of tree
d = depth of solution

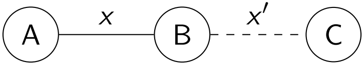
Dynamic Programming



$cost(A, a) = x$ (cost from state A for action a)
$total\_cost(B) = x'$ (total cost from state B to goal state)

$total\_cost(A) = ?$ (total cost from state A to goal state)

Dynamic Programming



$cost(A, a) = x$ (cost from state A for action a)
$total\_cost(B) = x'$ (total cost from state B to goal state)

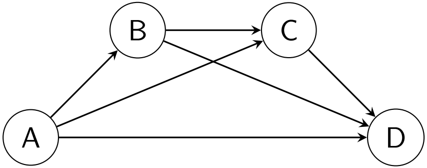
$$total\_cost(A) = \begin{cases} 0 & \text{if } is\_goal(A) \\ \min_{a \in actions(A)} \{cost(A, a) + total\_cost(successor(A, a))\} & \text{otherwise} \end{cases}$$

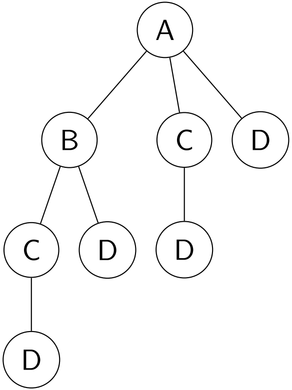
Dynamic Programming

Future cost only depends upon current state

start state: A

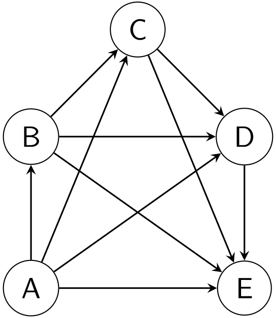
goal state: D



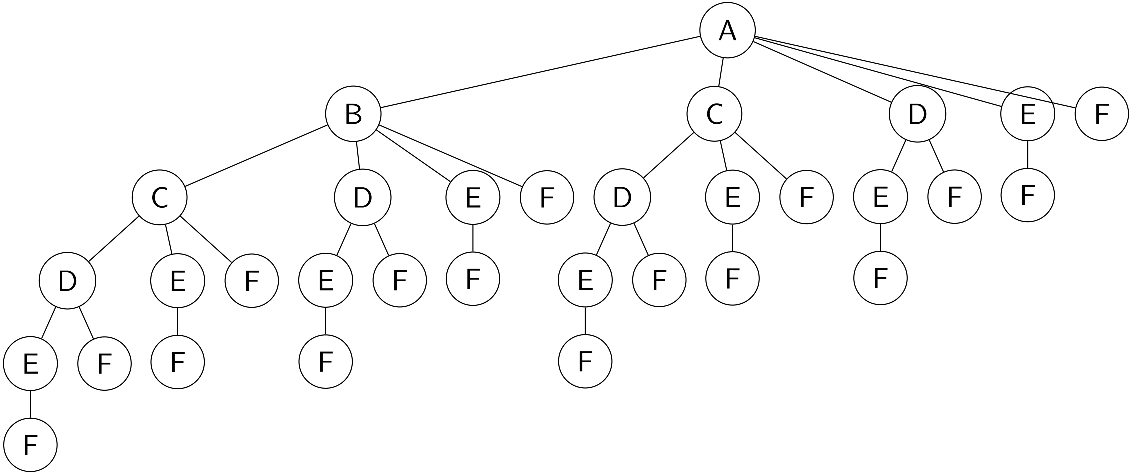


Dynamic Programming

start state: A
goal state: F



Dynamic Programming

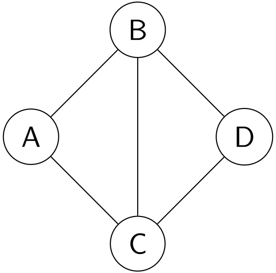


Dynamic Programming

Tree grows expotentially as number of nodes increase
With memoization, we can keep track of costs of visited nodes
Therefore we do not have to explore them again
However, it only works with acyclic graphs

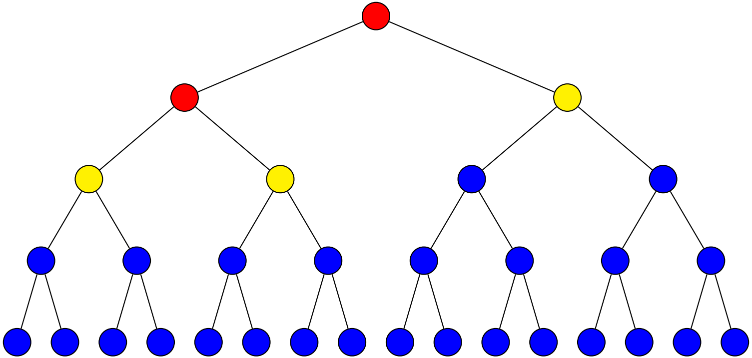
Graph with Cycles

What if state graph has cycles?
start state: A
goal state: D



Whiteboard

Tree Search



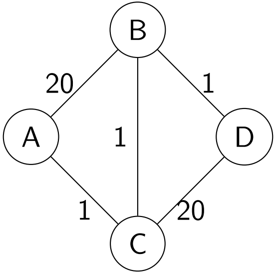
Explored: red
Frontier: yellow
Unexplored: blue

Uniform Cost Search

Keep a list of visited states

start state: A

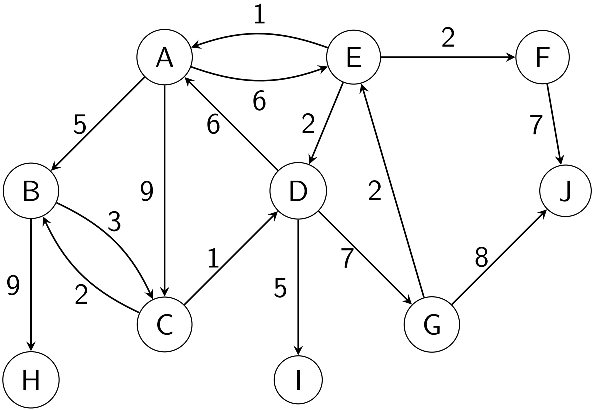
goal state: D



Whiteboard

Uniform Cost Search

start state: A
goal state: H, I, J



Whiteboard

Uniform Cost Search

```
Add (start,0) to frontier Q
While Q is not empty:
    u,p = remove state with smallest value from Q
    if u == goal:
        return
    Add u to explored E
    for a in actions(u):
        v = successor(u,a)
        if v not in E
            Add (v,p+cost(u,a)) to Q
```

Tree Search

Algorithm	Cost	Time	Space	Complete	Optimal
Backtracking Search	Any	$O(b^D)$	$O(D)$	Yes	Yes
DFS	0	$O(b^D)$	$O(D)$	No	Yes
BFS	≥ 0	$O(b^d)$	$O(b^d)$	Yes	If costs are equal
IDS	≥ 0	$O(b^d)$	$O(d)$	Yes	If costs are equal
UCS	≥ 0	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	Yes	Yes

$b =$ branching factor
$D =$ total depth of tree
$d =$ depth of solution
$C^* =$ cost of optimal path

Recap

Search Strategies Formulation Backtracking Search DFS & BFS Dynamic Programming Uniform Cost Search

References

Stuart Russell and Xiaodong Song (2021)
CS 188 — Introduction to Artificial Intelligence
University of California, Berkeley

Chelsea Finn and Nima Anari (2021)
CS221 — Artificial Intelligence: Principles and Techniques
Stanford University

The End