Outline for today

- Agent approach to AI
- Properties of AI environments
- Types of agent programs

Review terminology

Definition of Artificial Intelligence?

Definition of agent?

Review terminology

Definition of Artificial Intelligence: Creating computational agents that act intelligently

Definition of agent: Entities embedded in environments that can perceive and act

Consider properties of environments and ways of implementing agents

Agents

An agent perceives its environment and acts upon its environment

- Robot
  - percepts: lidar readings, camera images
  - actions: move wheels, arms, grippers
- Software Agent
  - percepts: keystrokes, file contents
  - actions: displaying on screen, writing to file
- Search Agent
  - percepts: states of the problem
  - actions: operators that modify the state

Agent behavior

- Described by an agent function
  - f(percepts) -> actions
- Implemented by an agent program
  - Goal of AI is exploring ways to implement this function

Properties of environments

- fully observable vs partially observable
- single agent vs multi agent
- deterministic vs stochastic
- static vs dynamic
- discrete vs continuous

Fully vs Partially observable

- If the agent has complete access to the state of the task environment, then it is fully observable

Fully observable: Crossword puzzle

Partially observable: Poker

What about playing a video game like Tetris?

Single vs Multi agent

- If the agent need only consider its own actions to complete the task, then it is a single agent environment

Single agent: Crossword puzzle

Multi agent: Poker

What about driving a car?

Deterministic vs Stochastic

- If the next state of the environment is completely determined by the current state and the action taken by the agent, then the environment is deterministic

Deterministic: Crossword puzzle

Stochastic: Poker

What about playing soccer?

Static vs Dynamic

- If the environment can change while the agent is deciding what action to take, then the environment is dynamic

Static: Crossword puzzle

Dynamic: Autonomous driving cars

What about playing poker?

Discrete vs Continuous

- If there are a finite number of distinct states within an environment, then it is discrete

Discrete: Crossword puzzle

Continuous: Autonomous driving cars

What about playing Chess?

Water Pitcher Problem

From the point of view of an agent trying to solve this problem, the environment is:

A. Partially observable, stochastic, dynamic
B. Fully observable, stochastic, static
C. Fully observable, deterministic, dynamic
D. Fully observable, deterministic, static
E. Partially observable, deterministic, static

Water Pitcher Problem

- Fully observable
- Single agent
- Deterministic
- Static
- Discrete

Environments with these characteristics are the easiest for AI to solve, why?

Water Pitcher Problem

- Fully observable
- Single agent
- Deterministic
- Static
- Discrete

Environments with these characteristics are the easiest for AI to solve, why? Because we can enumerate states and search through them for optimal path from start to goal

Hardest problems for AI are

- Partially observable
  - Don’t have access to all key state information
- Multi agent
  - Other agents may be changing the environment
- Stochastic
  - Can’t predict exactly how the environment will change
- Dynamic
  - The environment is in flux while agent decides what to do
- Continuous
  - Agent can’t enumerate all possible states

Possible Agent Programs

- Simple reflex agents
  - Base action on current percept, ignoring the past
- Model-based reflex agents
  - Save internal state about the world
  - Base action on both current percept and saved state
- Goal-based agents
  - Choose actions that try to achieve a goal
- Utility-based agents
  - Estimate the value of being in particular states and try to maximize this value

Perhaps most promising: Learning agents

- Operate in an initially unknown environment
- Become more competent over time
- Eventually outperform the initial programming
- All types of agents (reflex, model-based, goal-based, and utility-based) can incorporate learning into their program

Practice Quiz

State space search assumes that

A. The agent has perfect knowledge about its current state
B. The agent's actions have predictable effects
C. The agent can recognize when it is in a goal state
D. A solution is a sequence of actions from the agent's current state to a goal state
E. All of the above

State space search is implemented as graph search where

A. The states are nodes and the actions are links
B. The actions are nodes and the states are links
C. Neither of the above

Solving problems by search

- Focus on goal-based agents
- Operating in fully observable, single agent, deterministic, static, and discrete environments
- Can use state space search to solve these types of problems
- Uninformed search, such as BFS, DFS
- Informed search, such as A*

Example problems where state space search is effective

- Puzzles and games
  - Water pitchers
  - Traffic jam
  - 8 puzzle
- Real-world problems
  - Route finding
  - Tour finding
  - VLSI layout

Play with Traffic Jam game

www.dr-mikes-math-games-for-kids.com/online-traffic-jam-game.html

- Discrete or Continuous?
- Deterministic or Stochastic?
- Static or Dynamic?

Traffic Jam game

- Discrete: every object of interest is in a well defined location
- Deterministic: we can list all possible actions and outcomes
- Static: nothing unexpected happens between actions

State space search

- Closely related to graph theory in CS
- Must be able to represent all relevant information about a problem domain in a state description
- Create a graph
  - Nodes represent states
  - Branches represent possible actions
  - Root of graph is a node containing start state
  - Search the graph systematically until a node containing a goal state is found
  - Sequence of actions from start node to goal node is the solution

Formulating problems for search

- A process of abstraction
- “The choice of a good abstraction involves removing as much detail as possible while retaining validity and ensuring that the abstract actions are easy to carry out. Were it not for the ability to construct useful abstractions, intelligent agents would be completely swamped by the world.” (Russell & Norvig)
- But who is making the abstraction, the programmer or the AI agent?

Implementing state space search

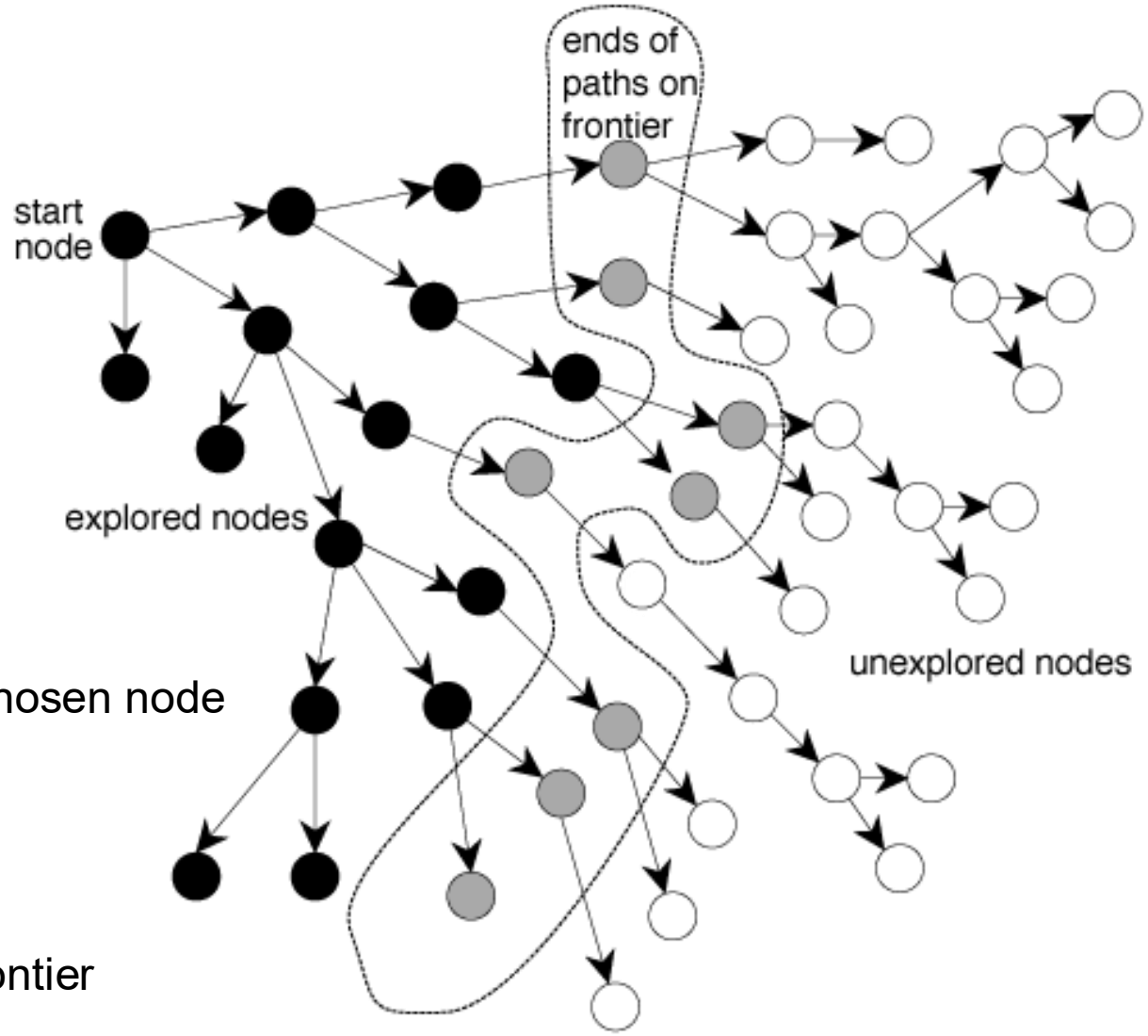
- Build a search tree on top of the state space
- Tree will consist of nodes representing each state with links representing actions
- Root will be a node containing the start state
- Frontier will be a data structure containing the successor nodes we need to explore
- Search strategies vary on how nodes are selected from the frontier

Nodes in graph may contain

- State description
- Action taken to get to this state
- Pointer to its parent node
- Step cost
- Total cost so far

Graph search pseudocode

```
GRAPH-SEARCH(problem) returns solution or failure
  initialize the frontier using the initial state of the problem
  # visited should be a data structure with efficient lookup
  initialize visited to contain initial state
  loop
    if the frontier is empty
      return failure
    remove a node from the frontier
    if the node contains a goal state
      return solution
   successor_states = apply operators to the state of the chosen node
   for each successor
        if successor not in visited
            add to visited
            create a node with successor_state and add to frontier
```



Search methods

- How new nodes are added to the frontier determines the type of search
- LIFO stack, creates depth-first search
- FIFO queue, creates breadth-first search
- Priority queue, where priority is based on some measure of closeness to goal (heuristic), creates a more directed search

Solving problems by searching

- Compare search methods based on completeness, optimality, time and space efficiency
- Discuss several uninformed search methods
  - BFS
  - Uniform Cost Search
  - DFS
  - Depth-limited Search
  - Iterative Deepening Search

Uninformed vs Informed methods

- Uninformed methods are only given the problem definition
  - start state
  - a way to check if at a goal state
  - a way to generate successors
  - Exs: DFS, BFS, Lowest cost, Depth-limited, IDS
- Informed methods are given additional information about the goal
  - estimated cost to reach goal
  - Exs: Greedy (aka Best First), A*
- All of these methods use a similar algorithm, what changes is the frontier data structure

Graph search pseudocode

```
GRAPH-SEARCH(problem) returns solution or failure
    make a node with initial state and add it to the frontier
        initialize visited to contain initial state  # visited is a data structure with efficient lookup
        loop
        if the frontier is empty
            return failure
        curr_node = frontier.remove()
        curr_state = curr_node.getState()
        if curr_state is goal
            return solution
        if curr_state not in visited
                add curr_state to visited
            successor_states = apply operators to curr_state
            for each successor
                        if successor not in visited
                    create a node with successor_state and add to frontier
```

Measuring search strategy performance

1. Completeness: Is the search guaranteed to find a solution if it exists?
2. Optimality: Is the search guaranteed to find the best solution in terms of cost?
3. Time complexity: How long does it take to find a solution?
4. Space complexity: How much memory is needed to perform the search?

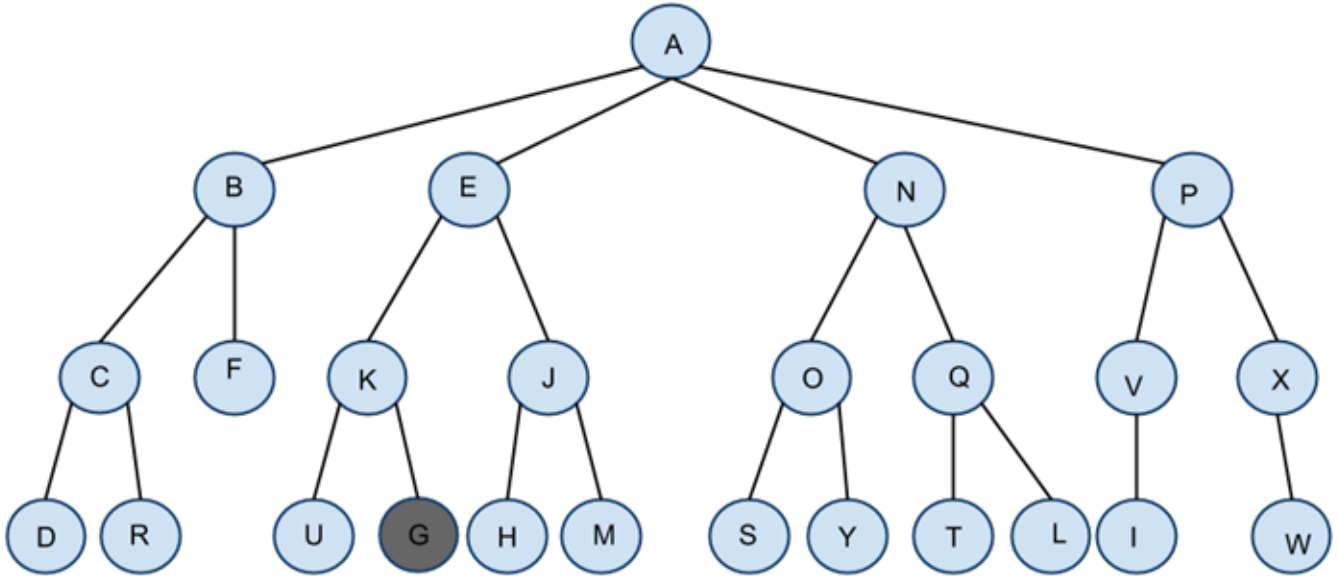
Assume:

- b=branching factor
- d=depth of the shortest path to the goal

Breadth-first Search

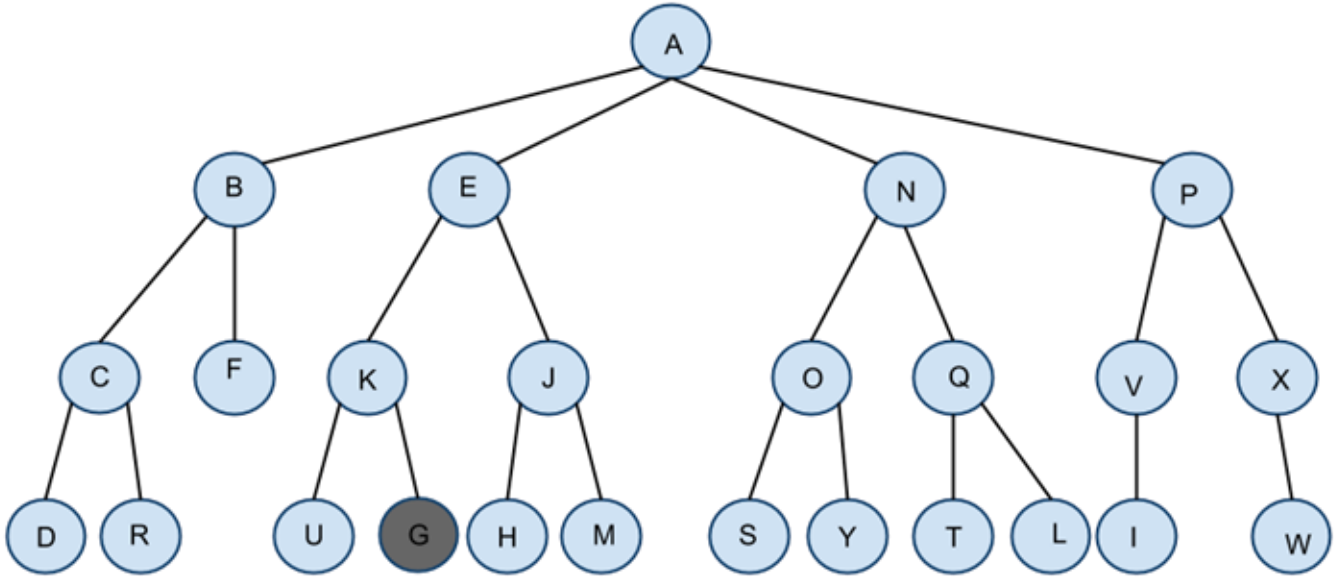
- Uses a queue to represent the frontier
- All nodes at a given level of the tree are expanded before any nodes at the next level
- Always expands the shallowest node in the frontier

BFS order of exploration?



A is the start state
G is the goal state
Assume neighbors added to frontier in R to L order

BFS order of exploration



A, P, N, E, B, X, V, Q, O, J, K, F, C, W, I, L, T, Y, S, M, H, G

Breadth-First Search Performance

- Complete?
- Optimal?
- Space efficient?
- Time efficient?

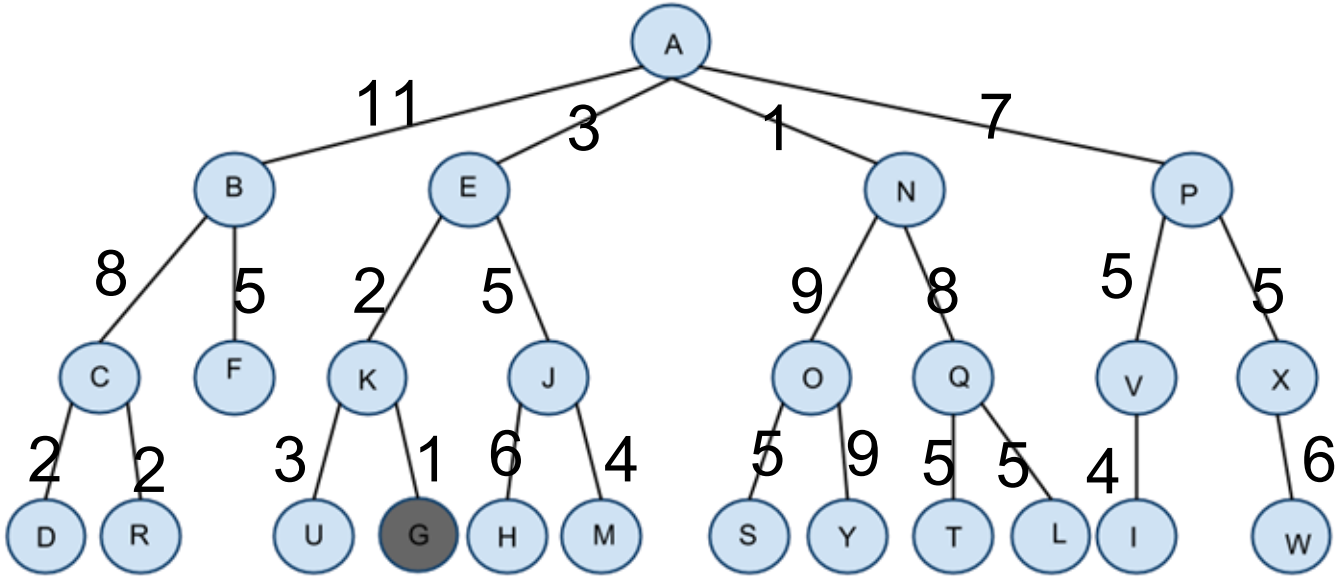
Breadth-First Search Performance

- Complete? Yes
  - Guaranteed to find shallowest goal node
- Optimal? Yes*
  - *If step costs are equal
- Space efficient? No
  - Nodes expanded $b + b^2 + b^3 + \ldots + b^d$
  - Frontier will contain $O(b^d)$ nodes, which can be huge
- Time efficient? No

Uniform Cost Search (better name would be Lowest Cost)

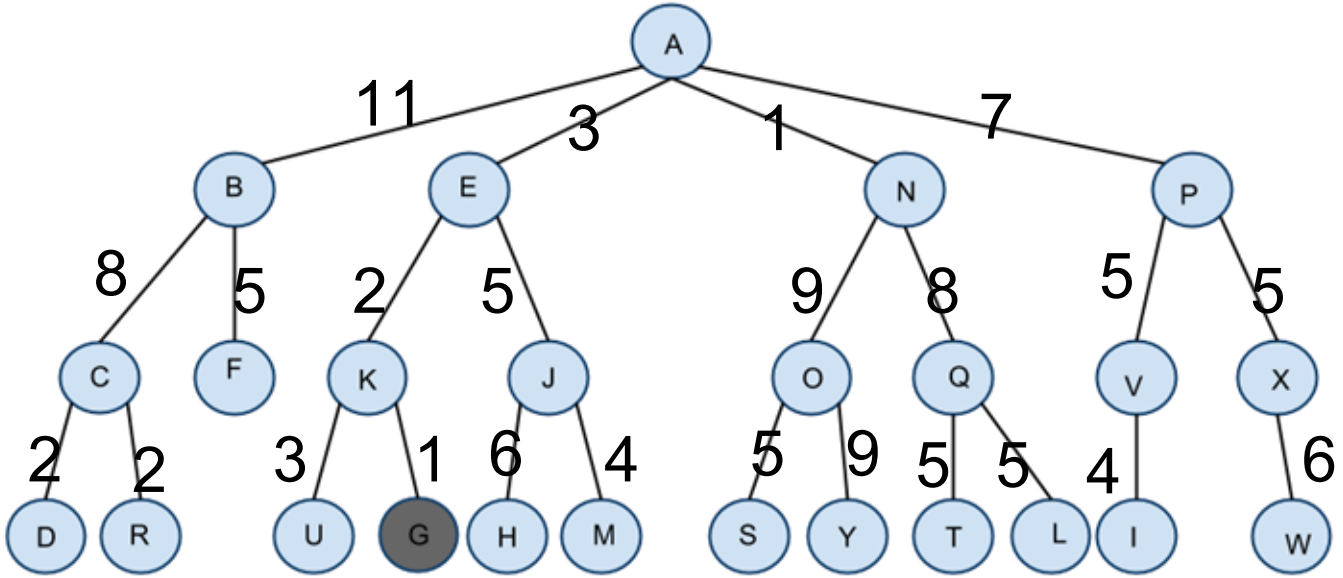
- Similar to BFS, but rather than expanding the shallowest node, expands the node with the lowest path cost
- Uses a priority queue to store the frontier
- More complicated to implement
  - Can’t just ignore a repeated state
  - When a repeated state is found, check to see if the new path cost is less, and if so replace the old node with this new node

Uniform Cost order of exploration?



A is the start state
G is the goal state

Uniform Cost order of exploration



A, N, E, K, G

Uniform Cost Search Performance

- Complete?
- Optimal?
- Space efficient?
- Time efficient?

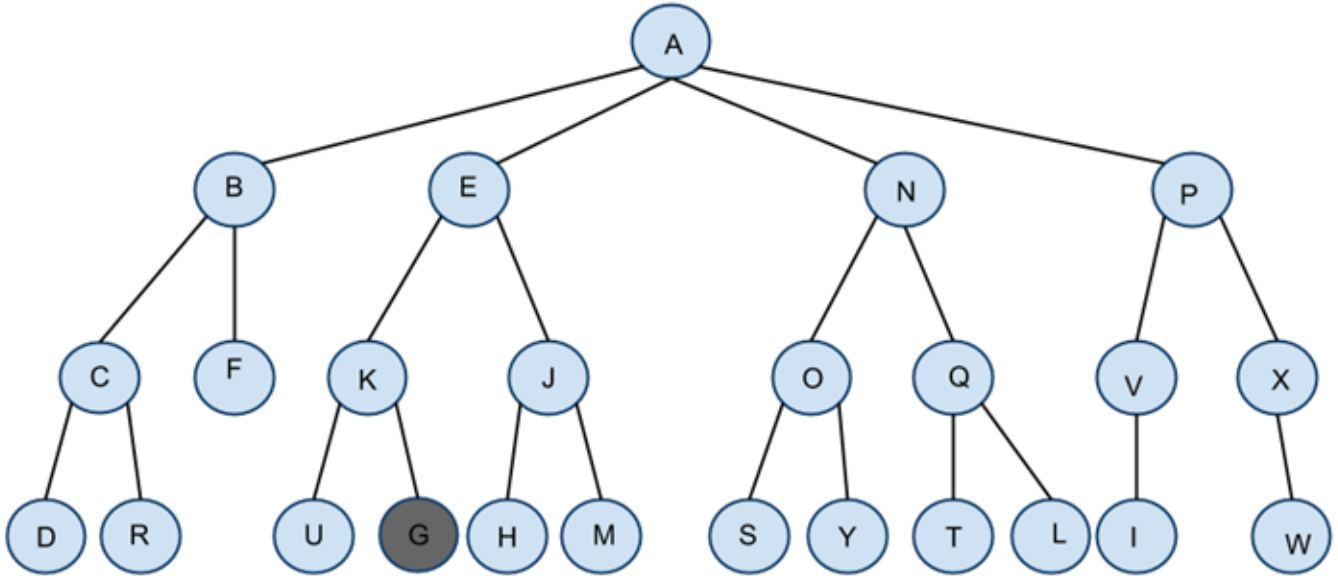
Uniform Cost Search Performance

- Complete? Yes
  - As long as step costs are positive
- Optimal? Yes
  - Expands nodes in order of their path cost (not depth)
  - So when goal is found, guaranteed to be lowest cost
- Space efficient? No, but better than BFS
- Time efficient? No, but better than BFS

Depth-First Search

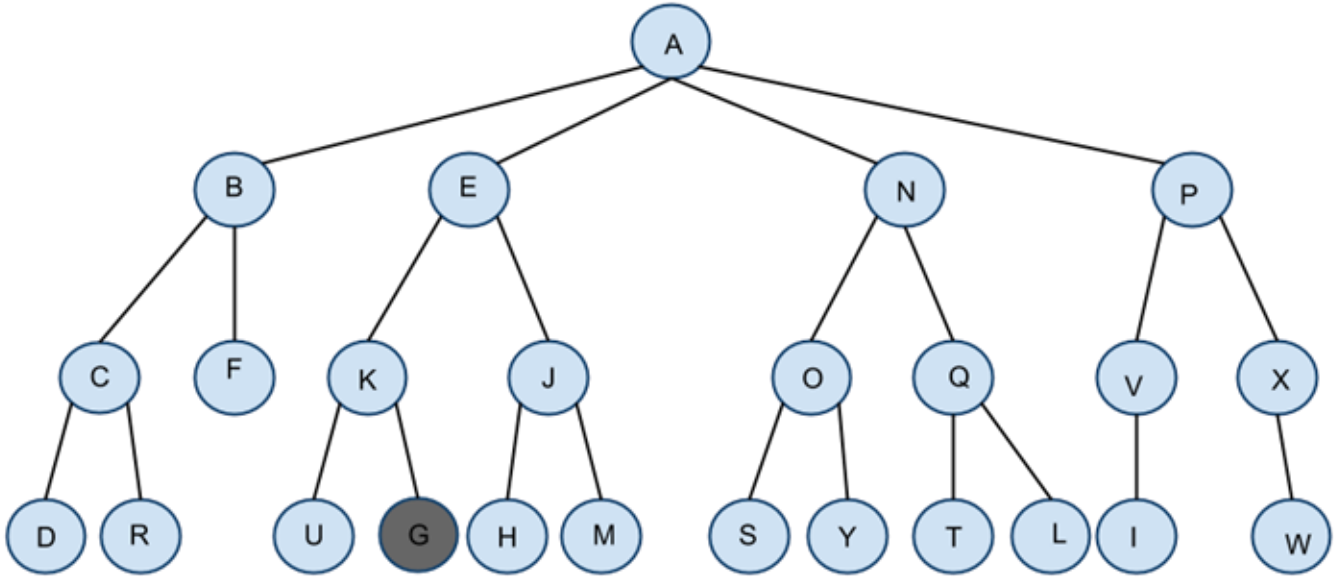
- Uses a stack to represent the frontier
- Always expands the deepest node in the frontier
- Can get caught in loops if repeated states are not checked

DFS order of exploration?



A is the start state
G is the goal state
Assume neighbors added to frontier in R to L order

DFS order of exploration



A, B, C, D, R, F, E, K, U, G

Depth-First Search Performance

- Complete?
- Optimal?
- Space efficient?
- Time efficient?

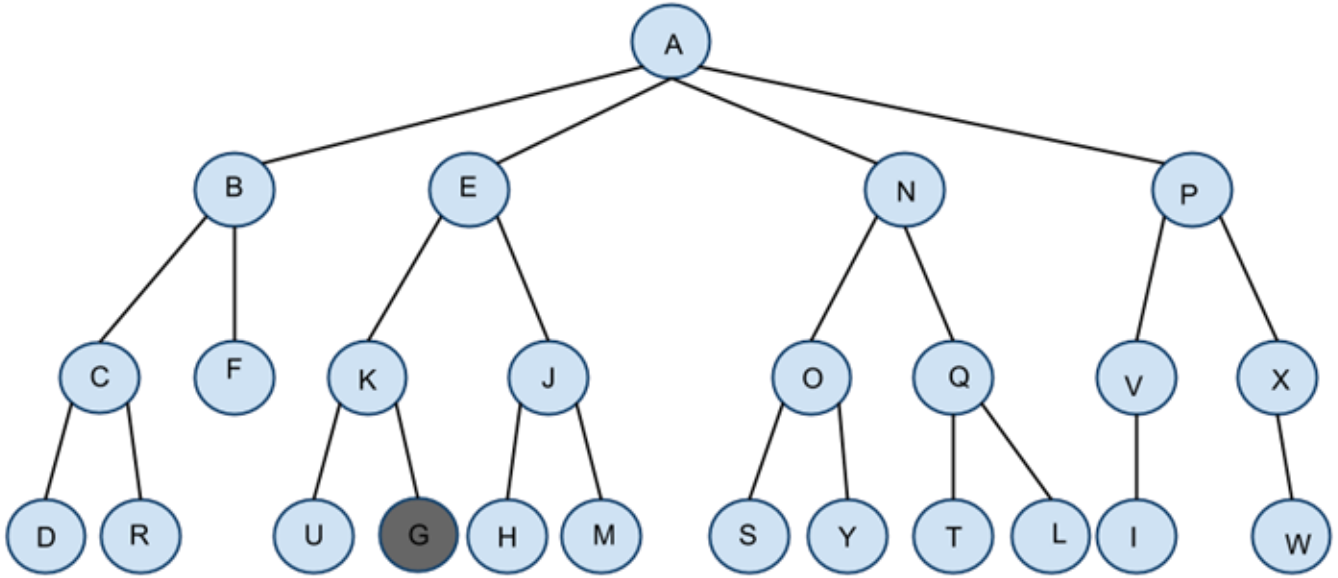
Depth-First Search Performance

- Complete? No
  - Can go forever down an infinite path that doesn't lead to a goal
- Optimal? No
  - May find a suboptimal path, when shorter paths exist
- Space efficient? Yes
  - Only needs to store a single path from the root to a leaf node
- Time efficient? Depends
  - Sometimes finds solutions very quickly

Depth-Limited Search

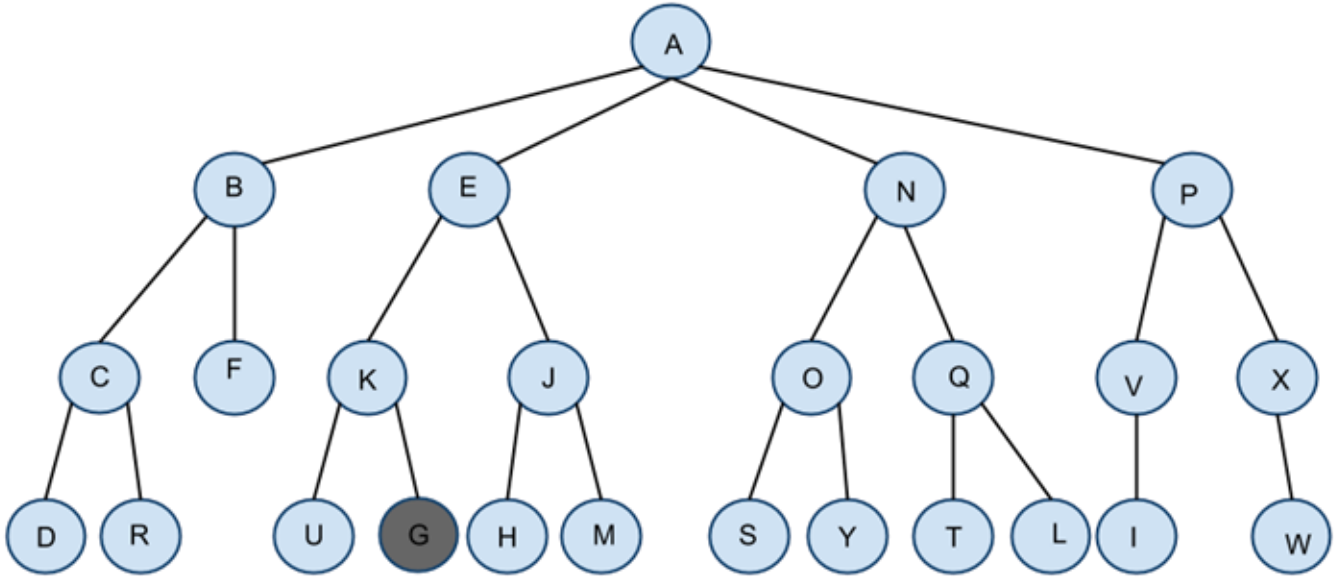
- Improvement on DFS, can handle infinite state spaces
- Provide a predetermined depth limit, when limit is reached, backtrack
- New problem: How best to determine an appropriate limit?

Depth-limited order of exploration?



A is the start state
G is the goal state
Assume neighbors added to frontier in R to L order
and depth limit is 2

Depth-limited order of exploration



With depth limit of 2:
A, B, C, F, E, K, J, N, O, Q, P, V, X, fails

Depth-Limited Search Performance

- Complete?
- Optimal?
- Space efficient?
- Time efficient?

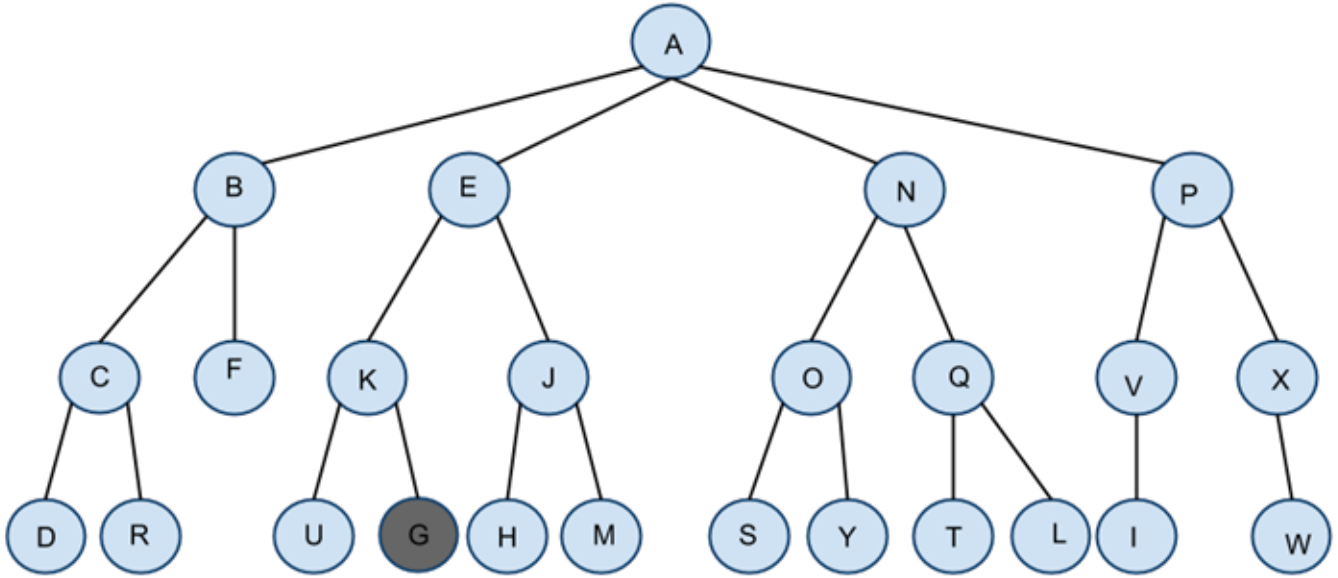
Depth-Limited Search Performance

- Complete? No
  - Only complete if shallowest goal is within chosen limit
- Optimal? No
  - May find a suboptimal path when shorter paths exist
- Space efficient? Yes
- Time efficient? Similar to DFS

Iterative Deepening DFS

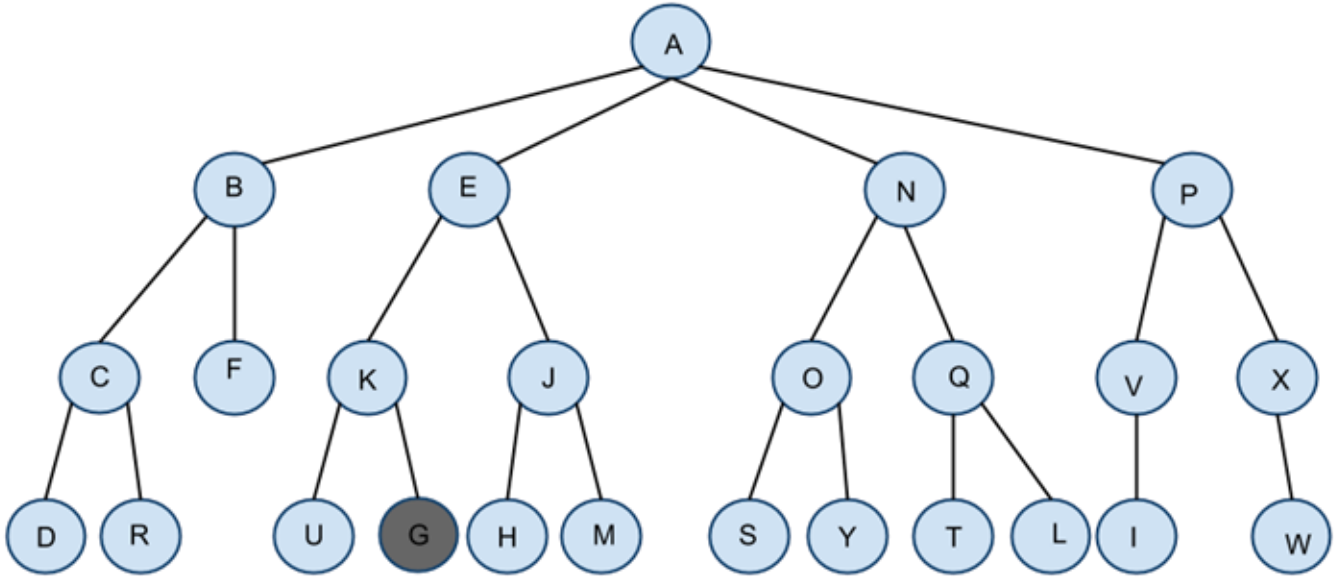
- Improvement on Depth-Limited Search, because no predetermined limit is needed
- Perform repeated Depth-Limited searches, where the limit starts at 0 and is incremented each time

Iterative deepening order of exploration?



A is the start state
G is the goal state
Assume neighbors added to frontier in R to L order

Iterative deepening order of exploration



limit 0: A
limit 1: A, B, E, N, P
limit 2: A, B, C, F, E, K, J, N, O, Q, P, V, X
limit 3: A, B, C, D, R, F, E, K, U, G

Iterative Deepening Performance

- Complete?
- Optimal?
- Space efficient?
- Time efficient?

Iterative Deepening Performance

- Complete? Yes
  - Each repetition goes one level deeper into the tree
  - Can handle infinite state spaces
- Optimal? Yes*
  - *Will find the shallowest goal node, so optimal when step costs are equal
- Space efficient? Yes, similar to DFS
- Time efficient? Surprisingly, no worse than BFS
  - Most of the nodes in a tree are at the bottom not the top, so repeatedly re-searching the top is not that expensive

Iterative Deepening Performance

- Combines the benefits of both BFS & DFS
- Completeness and optimality of BFS
- Space efficiency of DFS
- This is the preferred method of search when the search space is large and the depth of the solution is unknown

Exercise

- Consider a state space where the start state is number 1 and each state k has two successors 2*k and (2*k)+1
- Draw a portion of the state space for states 1 to 15
- Suppose the goal is 6
- List order in which states will be explored when we visit neighbors in R to L order:
  - BFS
  - DFS
  - Depth-limited with limit 2
  - Iterative deepening