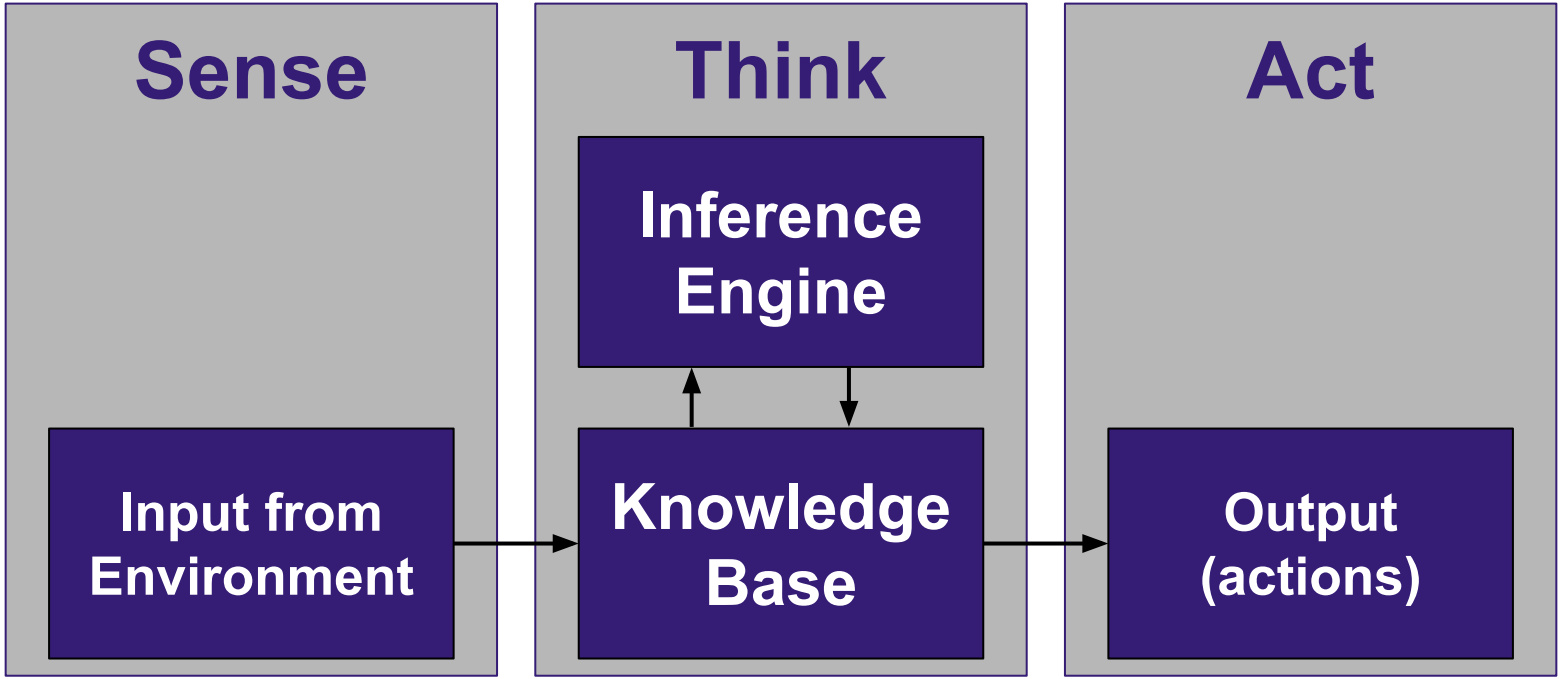
Intelligent Agents, Problem Formulation and Search

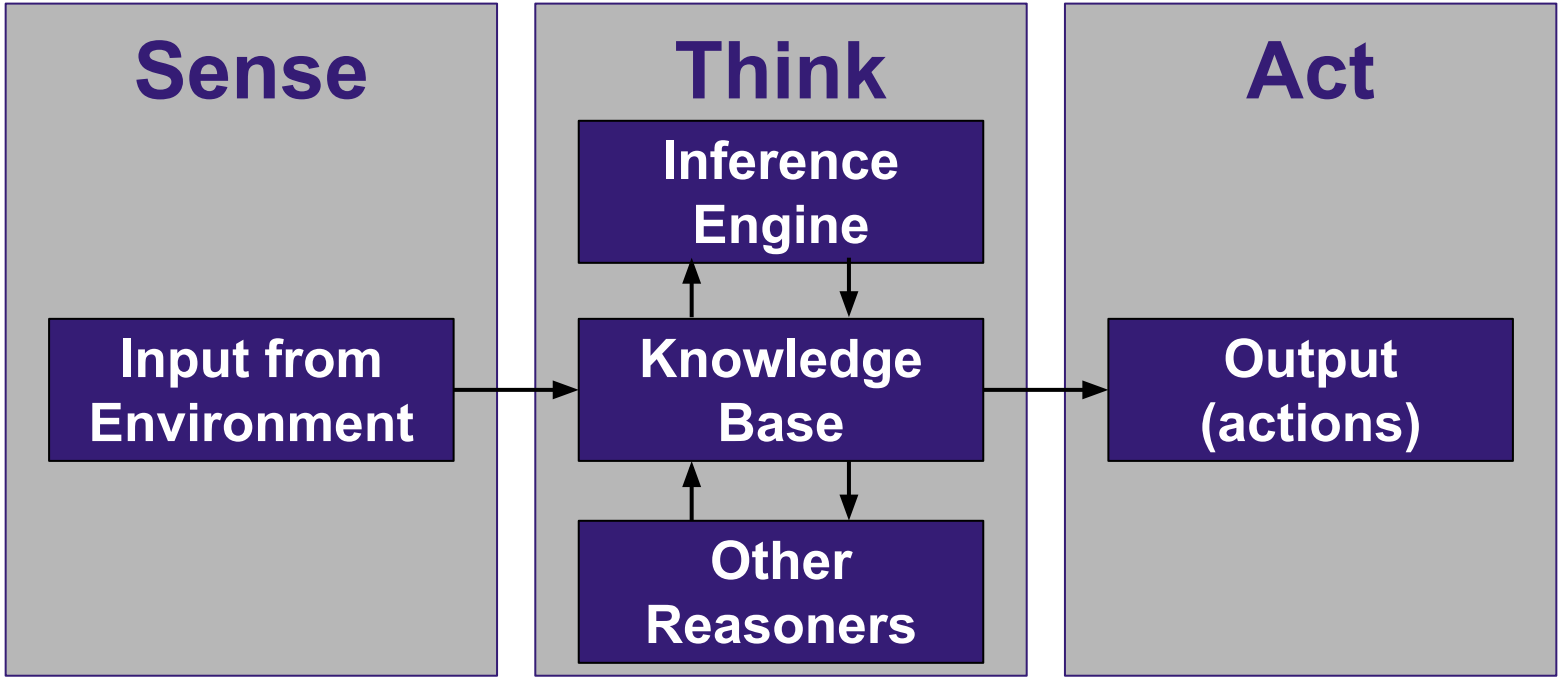
willie
(adapted from slides from Sara Owsley Sood)

Knowledge

Knowledge-based agent



Knowledge-based agent

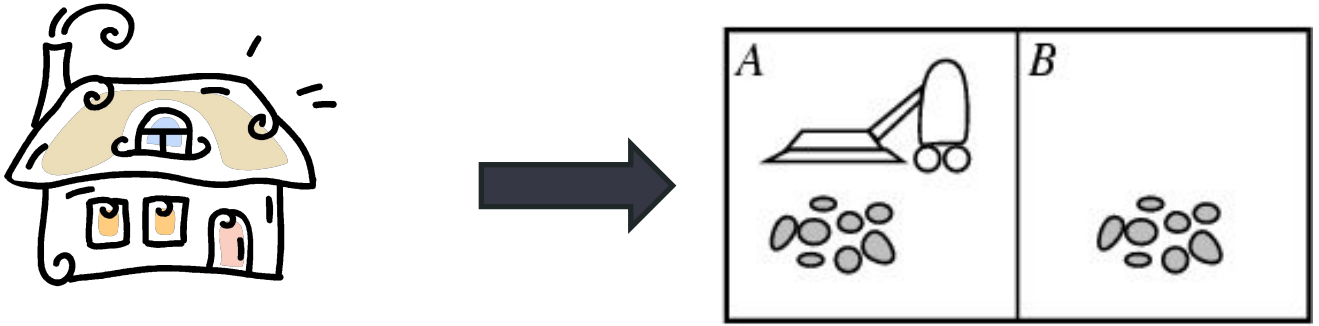


How do we make a vacuum "smart?"

How do we represent this task?

We want the vacuum to "clean the house." What does this mean? What's involved for the vacuum cleaner? How can we formalize this problem a bit?

Vacuum Cleaner World



Percepts: location and contents, e.g., [A,Dirty]

Actions: *Left*, *Right*, *Suck*, *NoOp*

What does the agent DO?

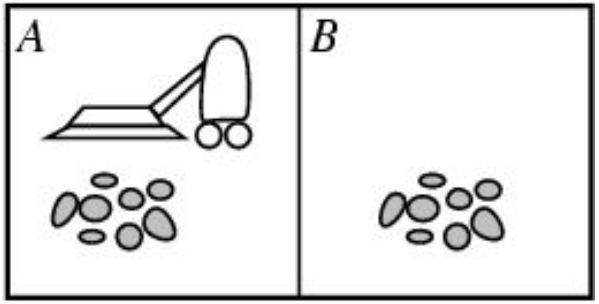
Our goal as AI programmers is develop agents that behave rationally. This means we must specify what the agent does given:

- Its goals
- Its precepts
- Its possible actions
- Its prior knowledge

This plan of action is the agent's **policy or 'agent function'**

Programming a Rational Agent → Policy Design and Implementation

Defining a Policy



How might we define a policy for the vacuum agent above if the goal is to clean both rooms?

Approaches

Random

Agent responds randomly or pseudorandomly to its observations

Logical inference

Agent's response is based on what is logically entailed from its observations

Search

Agent searches through a space of possibilities to find the "best" action to take

Utility

Agent's response is based on the value of the effects of each of its actions

Learning

Agent's response is based on what has worked best in the past

Approaches

Random

Agent responds randomly or pseudorandomly to its observations

Logical inference

Agent's response is based on what is logically entailed from its observations

Search

Agent searches through a space of possibilities to find the "best" action to take

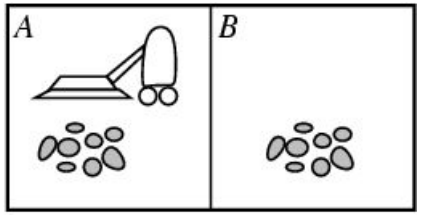
Utility

Agent's response is based on the value of the effects of each of its actions

Learning

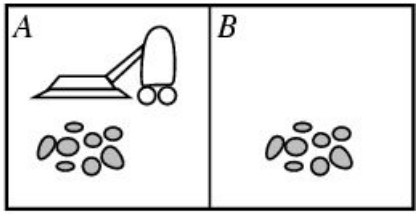
Agent's response is based on what has worked best in the past

Vacuum world knowledge-based agent using search



1. Formulate problem and goal
2. Search for a sequence of actions that will lead to the goal (the policy)
3. Execute the actions one at a time

Vacuum world knowledge-based agent using search



1. **Formulate problem and goal**
2. Search for a sequence of actions that will lead to the goal (the policy)
3. Execute the actions one at a time

Well-defined problem:

(State space)

Initial state

Goal test

Actions/Successor function

Path cost

Vacuum world

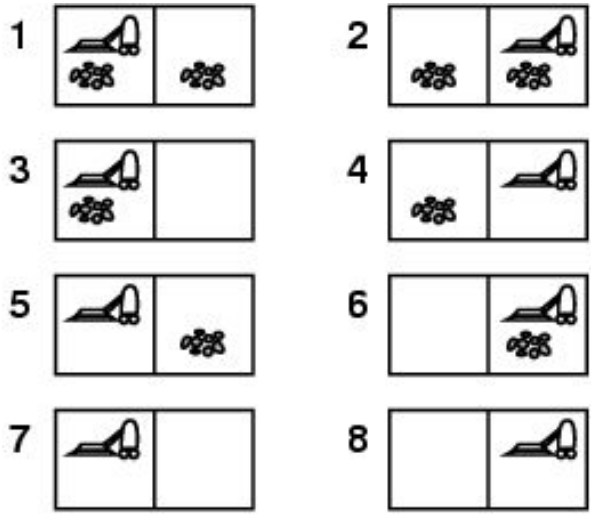
States: Shown to the right

Actions: R, L, S, NoOp

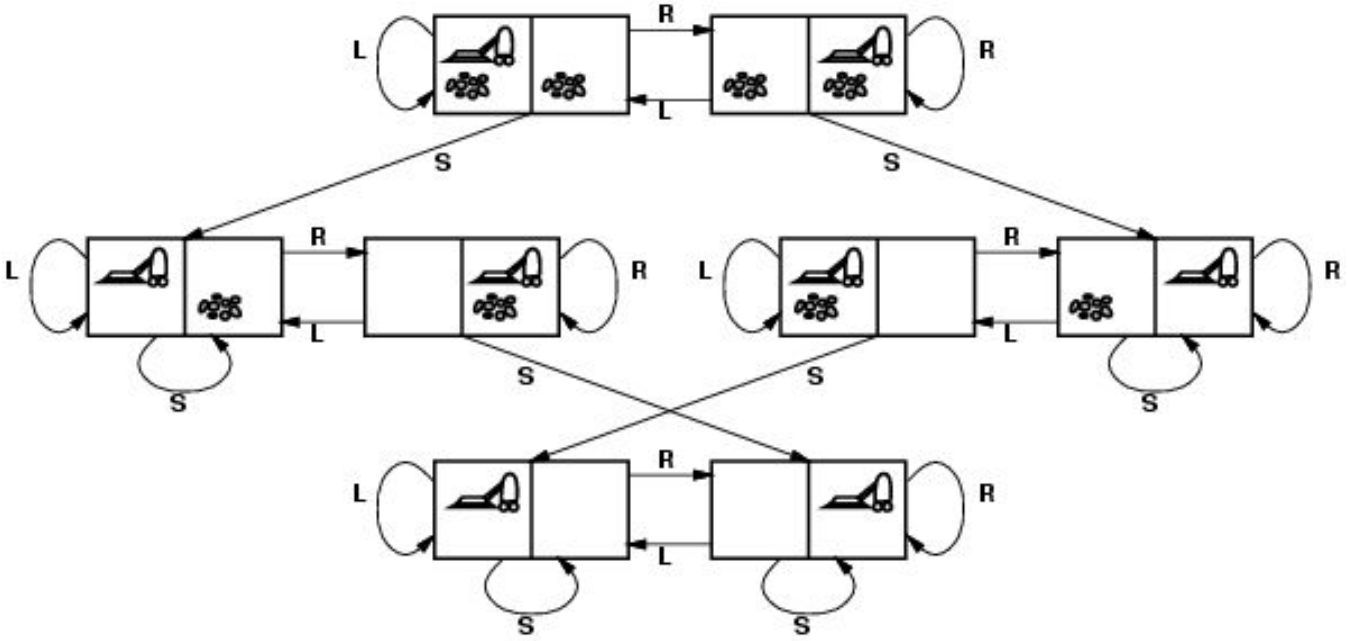
Successor function given by *state graph* (next slide)

Goal test: In state 7 or 8?

Path cost: +1 for each move and each suck



Vacuum world state space graph

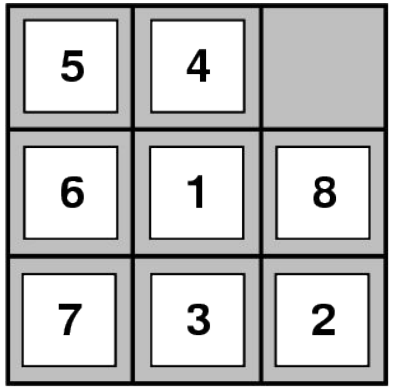


Some example problems

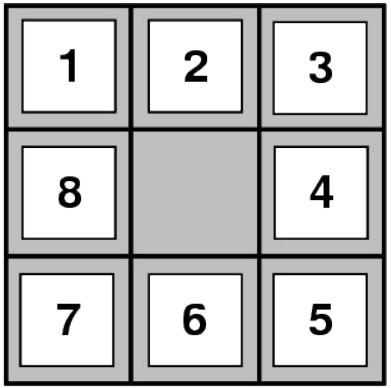
Toy problems and micro-worlds:

- 8-Puzzle
- Missionaries and Cannibals
- Water Jug Problem
- Roomers
- Nqueens

Another problem: 8-Puzzle



Start State



Goal State

8-Puzzle

state:

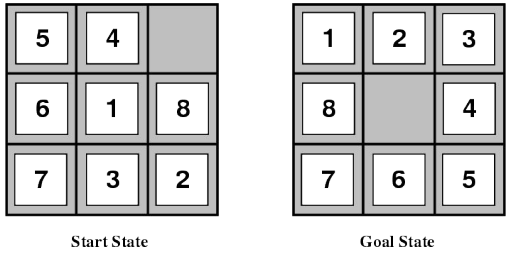
- all 3 x 3 configurations of the tiles on the board

actions:

- Move Blank Square Left, Right, Up or Down.
- This is a more efficient encoding than moving each of the 8 distinct tiles

path cost:

- +1 for each action



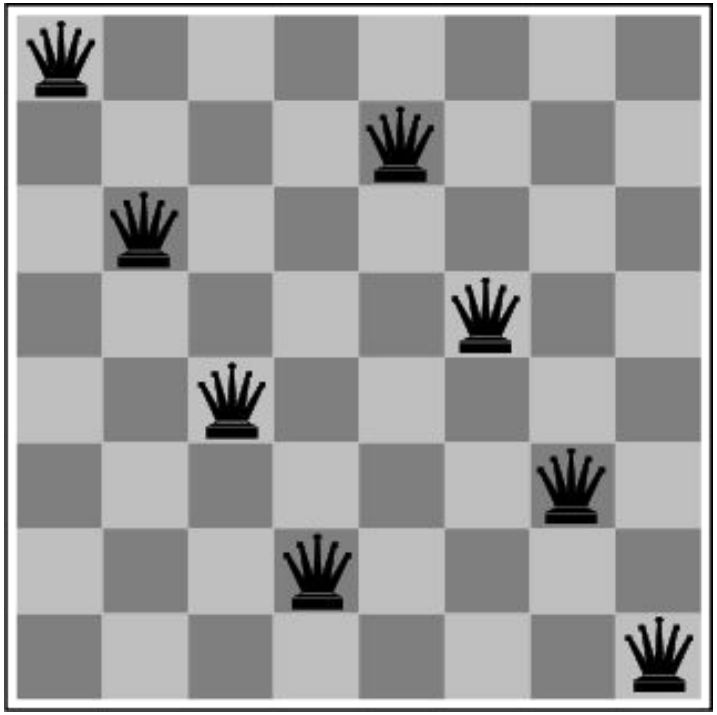
The 8-Queens Problem

State transition: ?

Initial State: ?

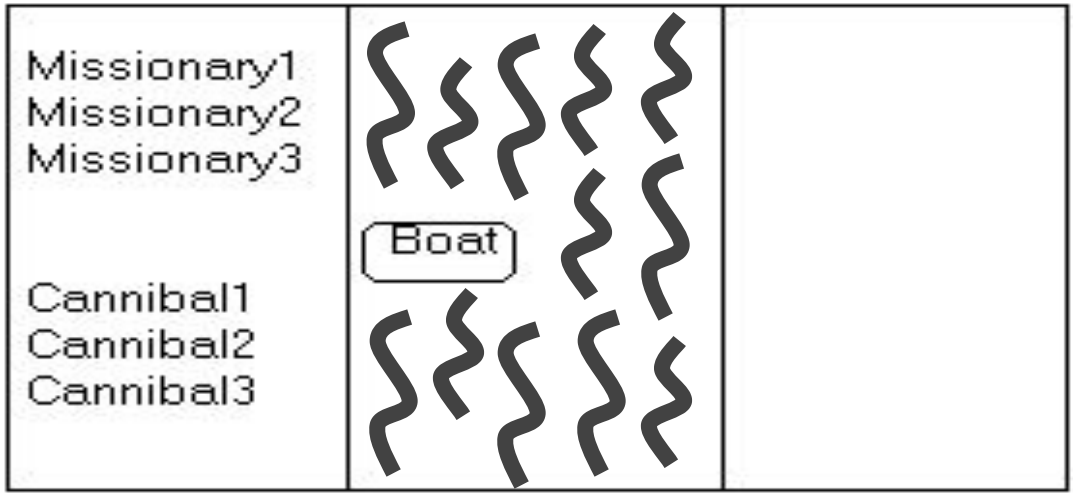
Actions: ?

Goal: Place eight queens on a chessboard such that no queen attacks any other!



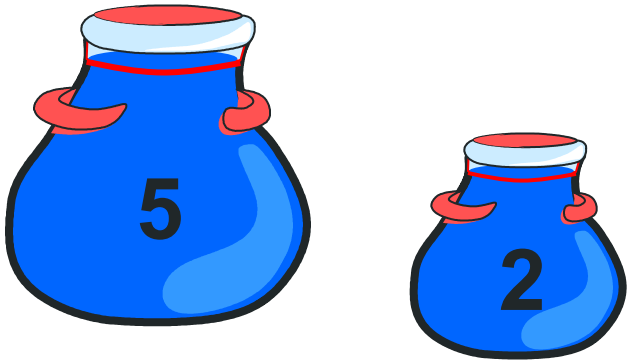
Missionaries and Cannibals

Three missionaries and three cannibals wish to cross the river. They have a small boat that will carry up to two people. Everyone can navigate the boat. If at any time the Cannibals outnumber the Missionaries on either bank of the river, they will eat the Missionaries. Find the smallest number of crossings that will allow everyone to cross



Water Jug Problem

Given a full 5-gallon jug and a full 2-gallon jug, fill the 2-gallon jug with exactly one gallon of water.



Roomers

Baker, Cooper, Fletcher, Miller, and Smith live on different floors of an apartment house that contains only five floors. Baker does not live on the top floor. Cooper does not live on the bottom floor. Fletcher does not live on either the top or the bottom floor. Miller lives on a higher floor than does Cooper. Smith does not live on a floor adjacent to Fletcher's. Fletcher does not live on a floor adjacent to Cooper's. Where does everyone live?

[Taken verbatim from Abelson and Sussman, Structure and Interpretation of Computer Programs, second edition, M.I.T. Press, 1996. The authors attribute the puzzle to Dinesman, Superior Mathematical Puzzles, Simon and Schuster, 1968.]

Some real-world problems

Route finding

- directions, maps
- computer networks
- airline travel

VLSI layout

Touring (traveling salesman)

Agent planning

Search-based agent

1. Formulate problem and goal
2. **Search for a sequence of actions that will lead to the goal (the policy)**
3. Execute the actions one at a time

Search algorithms

We've defined the problem

Now we want to find the solution!

Use search techniques

- offline, simulated exploration of state space by generating successors of already-explored states (a.k.a. **expanding** states)
- Start at the initial state and search for a goal state

What are candidate search techniques?

- BFS
- DFS

Finding the path: Tree search algorithms

Basic idea:

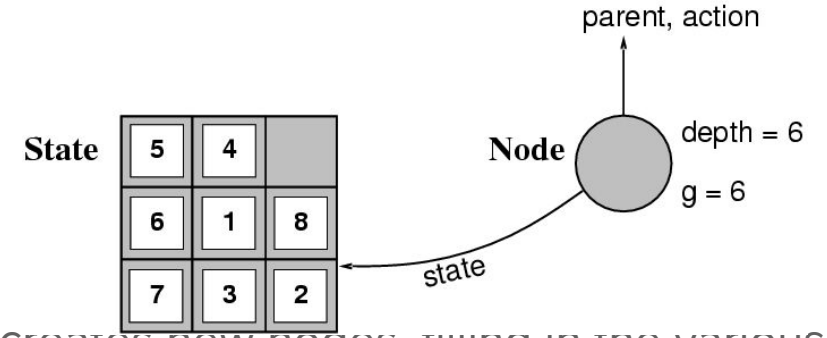
- offline, simulated exploration of state space by generating successors of already-explored states (a.k.a. **expanding** states)

```
def TreeSearch(problem, strategy):
  initialize search tree using information in the problem
  while true:
    if there are no candidates for expansion, return failure
    choose a leaf node for expansion according to strategy
    if node contains goal state, return solution
    else expand node and add resulting nodes to search tree
```

Careful! states vs. nodes

A state is a (representation of) a physical configuration

A node is a data structure constituting part of a search tree includes state, parent node, action, path cost *g(x)*, depth



The Expand function creates new nodes, filling in the various fields and using the Successor function of the problem to create the corresponding states.

Implementation: general tree search

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        fringe ← INSERTALL(EXPAND(node, problem), fringe)

function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Tree Search Algorithm

1. Add the initial state (root) to the <fringe>
2. **Choose a node (curr) to examine from the** <fringe>
   (if there is nothing in <fringe> - FAILURE)
3. Is curr a goal state?
   If so, SOLUTION
   If not, continue
4. Expand curr by applying all possible actions (add the new resulting states to the <fringe>)
5. Go to step 2

Search strategies

A search strategy is defined by picking the order of node expansion

How to evaluate a strategy?

Uninformed search strategies

Uninformed search strategies use only the information available in the problem definition

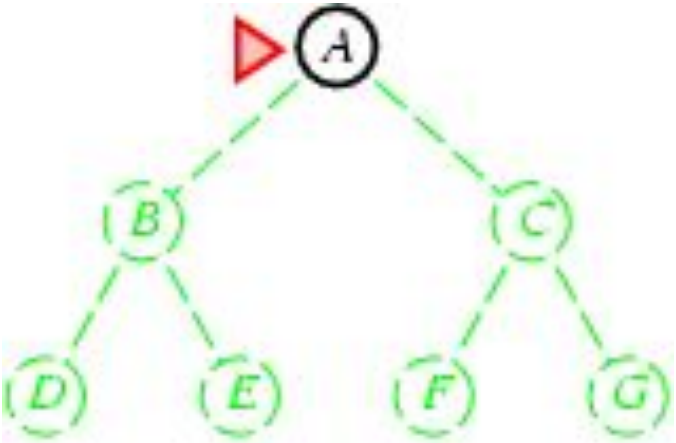
- Depth First Search
- Breadth First Search

Breadth-first search

Expand shallowest unexpanded node

Implementation:

- *fringe* is a FIFO queue, i.e., new successors go at end

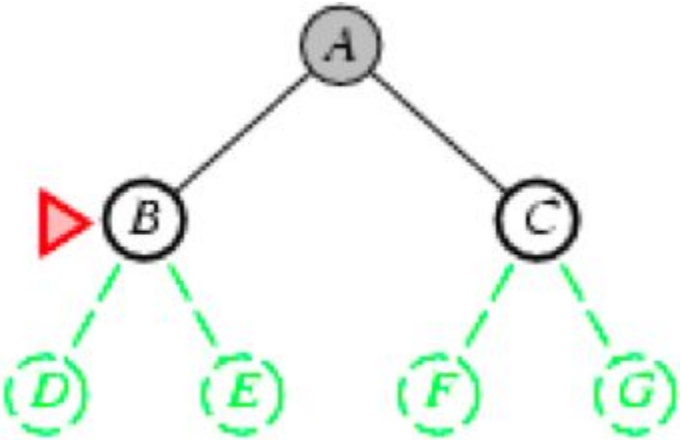


Breadth-first search

Expand shallowest unexpanded node

Implementation:

- *fringe* is a FIFO queue, i.e., new successors go at end

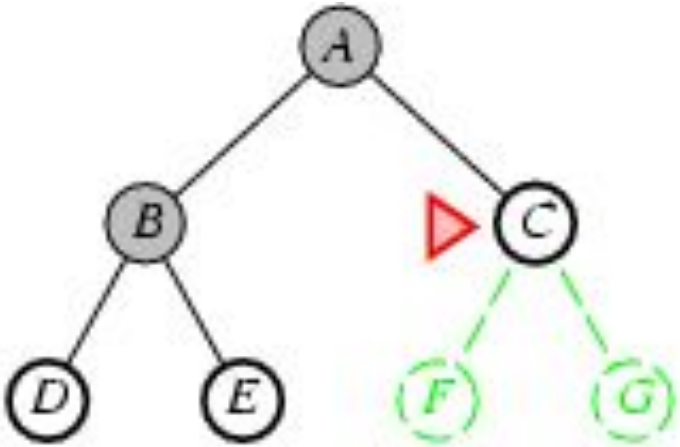


Breadth-first search

Expand shallowest unexpanded node

Implementation:

- *fringe* is a FIFO queue, i.e., new successors go at end

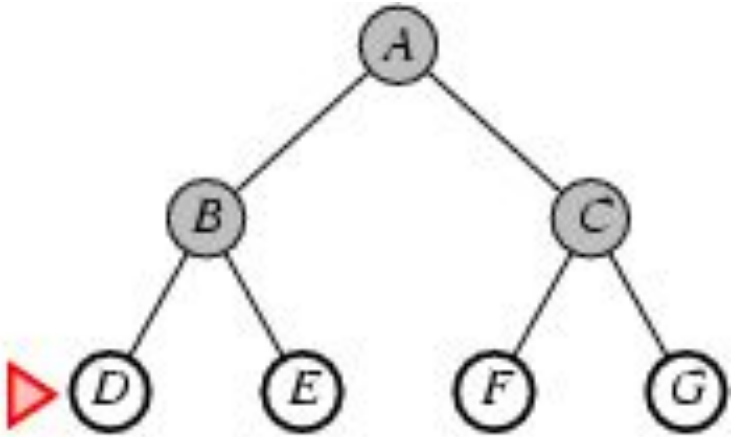


Breadth-first search

Expand shallowest unexpanded node

Implementation:

- *fringe* is a FIFO queue, i.e., new successors go at end

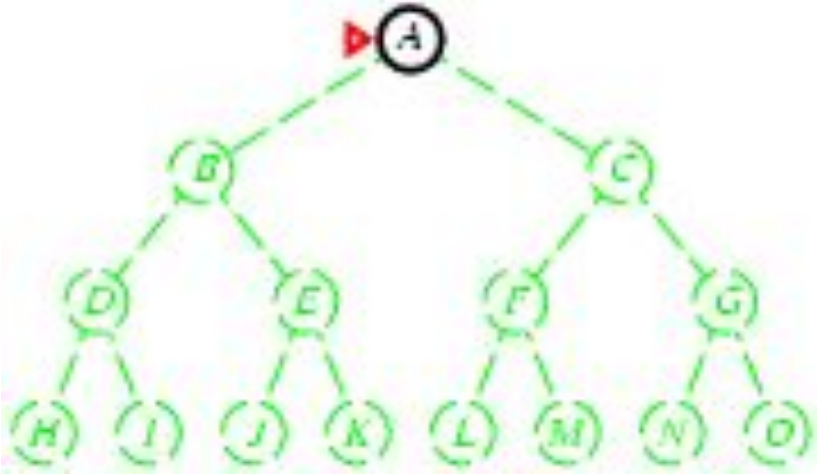


Depth-first search

Expand deepest unexpanded node

Implementation:

- *fringe* is a LIFO stack, i.e., put successors at front

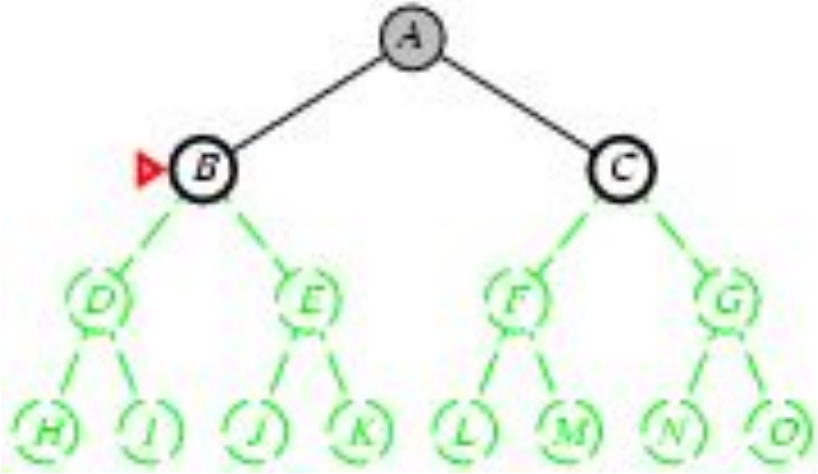


Depth-first search

Expand deepest unexpanded node

Implementation:

- *fringe* is a LIFO stack, i.e., put successors at front

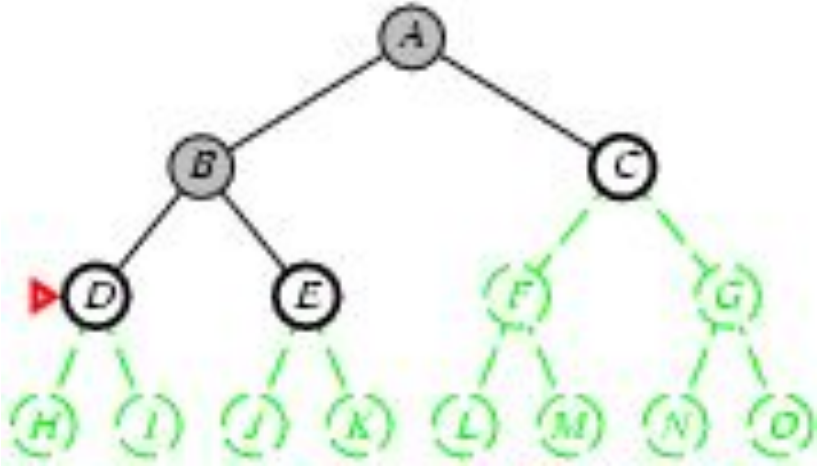


Depth-first search

Expand deepest unexpanded node

Implementation:

- *fringe* is a LIFO stack, i.e., put successors at front

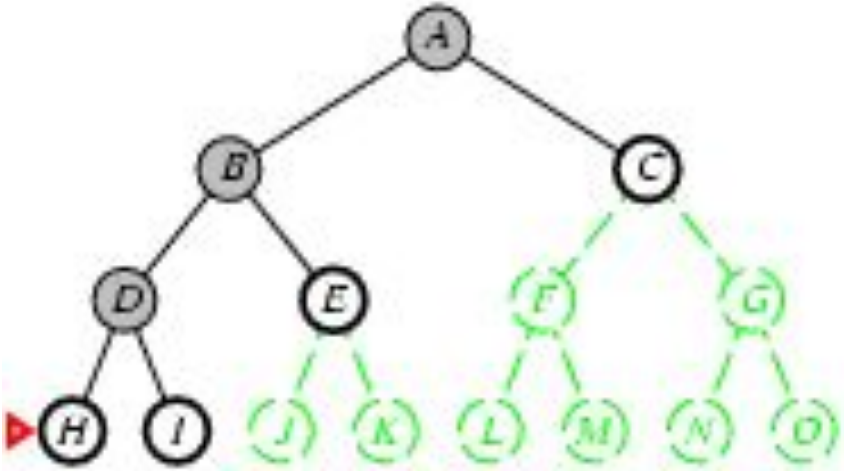


Depth-first search

Expand deepest unexpanded node

Implementation:

- *fringe* is a LIFO stack, i.e., put successors at front

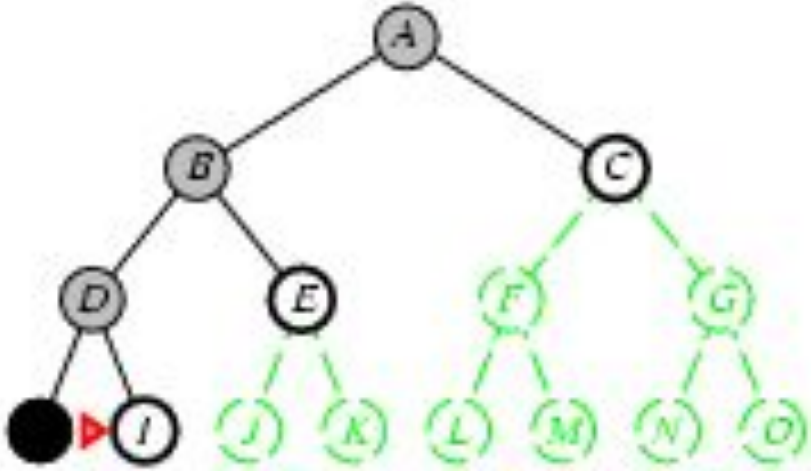


Depth-first search

Expand deepest unexpanded node

Implementation:

- *fringe* is a LIFO stack, i.e., put successors at front

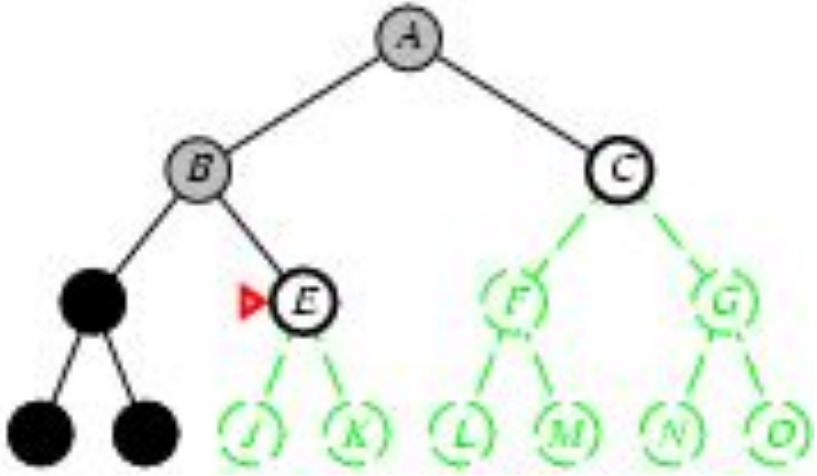


Depth-first search

Expand deepest unexpanded node

Implementation:

- *fringe* is a LIFO stack, i.e., put successors at front

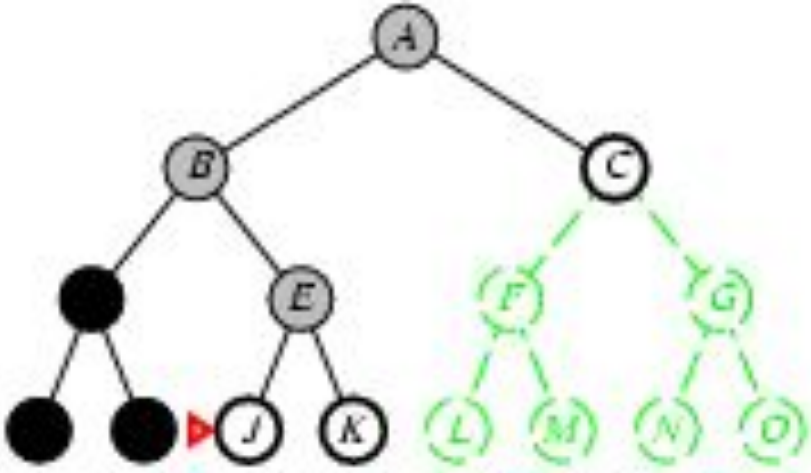


Depth-first search

Expand deepest unexpanded node

Implementation:

- *fringe* is a LIFO stack, i.e., put successors at front

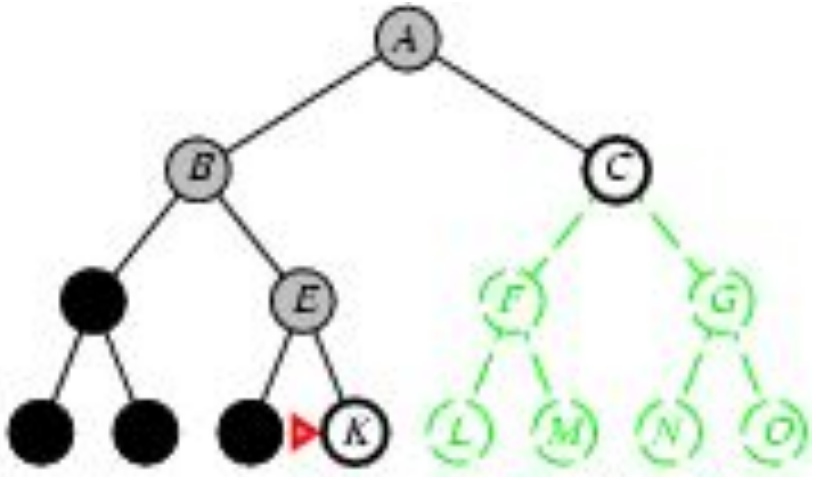


Depth-first search

Expand deepest unexpanded node

Implementation:

- *fringe* is a LIFO stack, i.e., put successors at front

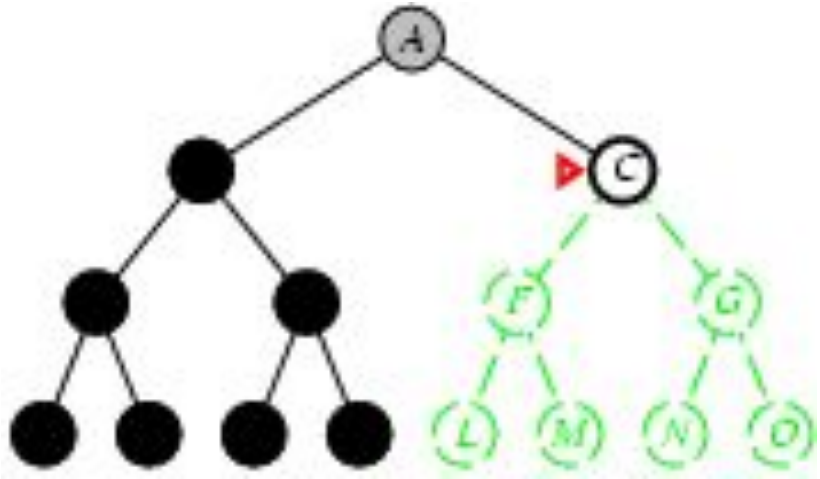


Depth-first search

Expand deepest unexpanded node

Implementation:

- *fringe* is a LIFO stack, i.e., put successors at front

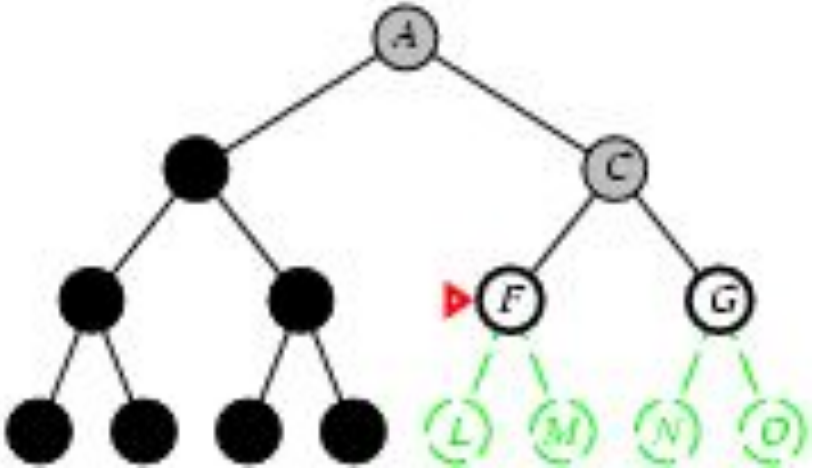


Depth-first search

Expand deepest unexpanded node

Implementation:

- *fringe* is a LIFO stack, i.e., put successors at front

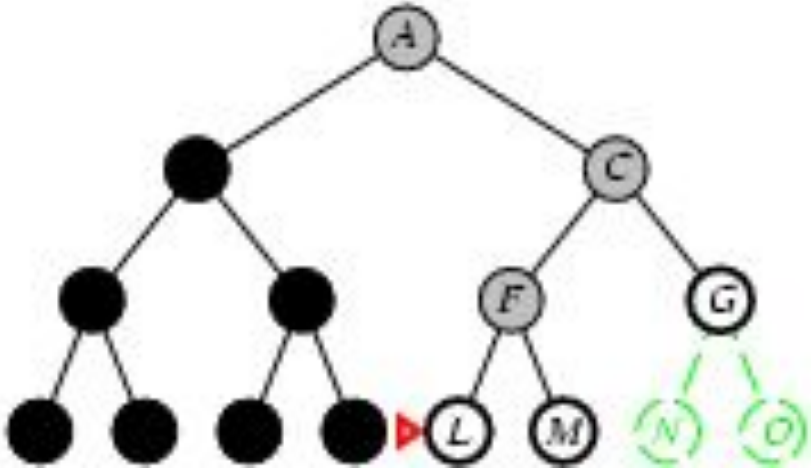


Depth-first search

Expand deepest unexpanded node

Implementation:

- *fringe* is a LIFO stack, i.e., put successors at front

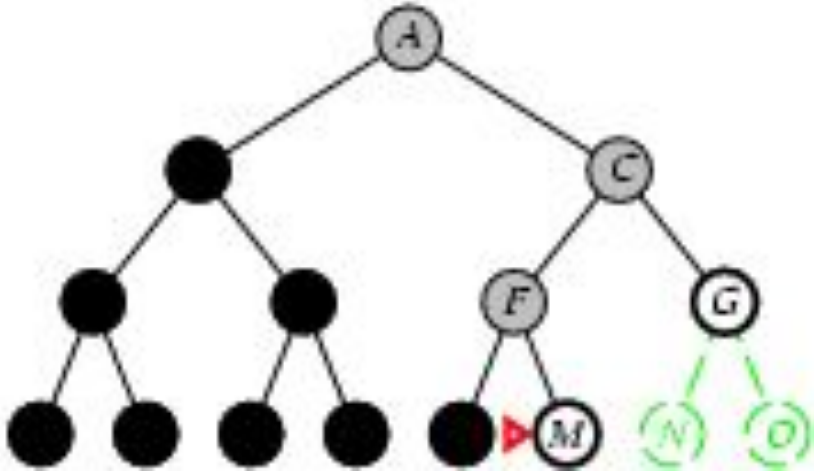


Depth-first search

Expand deepest unexpanded node

Implementation:

- *fringe* is a LIFO stack, i.e., put successors at front



Search algorithm properties

Time (using Big-O)

- Approximate the number of nodes generated (not necessarily examined)

Space (using Big-O)

- The max # of nodes stored in memory at any time

Complete

- If a solution exists, will we find it?

Optimal

- If we return a solution, will it be the best/optimal (really just shallowest) solution

Activity

Analyze DFS and BFS according to the criteria time, space, completeness and optimality

(for time and space, analyze in terms of *b*, *d*, and *m* (max depth); for complete and optimal - simply YES or NO)

- Which strategy would you use and why?

Brainstorm improvements to DFS and BFS

BFS

Time: $O(b^{d+1})$

Space: $O(b^{d+1})$

Complete = YES

Optimal = YES

Time and Memory requirements for BFS

Depth	Nodes	Time	Memory
2	1100	.11 sec	1 MB
4	111,100	11 sec	106 MB
6	10^7	19 min	10 GB
8	10^9	31 hours	1 terabyte
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

BFS with b=10, 10,000 nodes/sec; 10 bytes/node

DFS

Time: $O(b^m)$

Space: O(bm)

Complete = NO (YES, if space is finite and no circular paths)

Optimal = NO

Problems with BFS and DFS

BFS

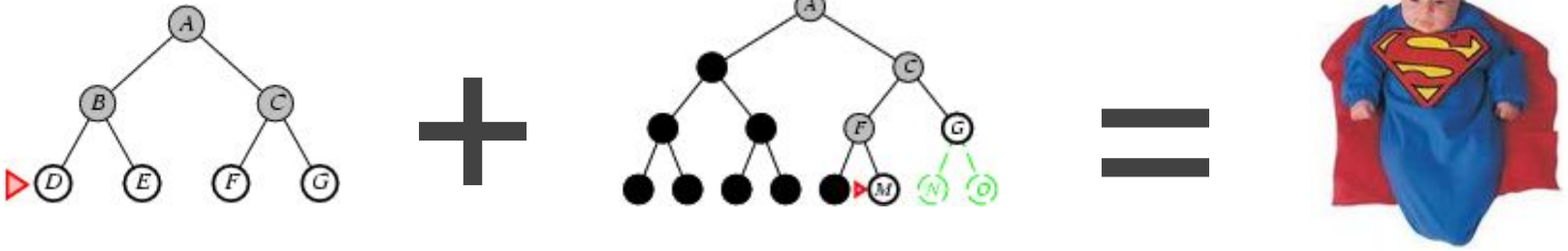
- memory! ☹

DFS

- Not optimal
- And not even necessarily complete!

Ideas?

Can we combined the optimality and completeness of BFS with the memory of DFS?



Depth limited DFS

DFS, but with a depth limit **L** specified

- nodes at depth **L** are treated as if they have no successors
- we only search down to depth **L**

Time?

- $O(b^L)$

Space?

- O(bL)

Complete?

- No, if solution is longer than L

Optimal

- No, for same reasons DFS isn’t

Ideas?

Iterative deepening search

For depth 0, 1, ..., ∞
    run depth limited DFS
    if solution found, return result

Blends the benefits of BFS and DFS

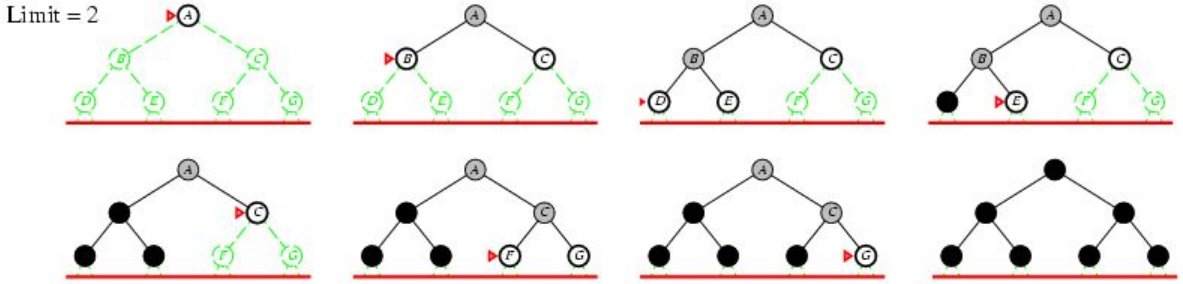
- searches in a similar order to BFS
- but has the memory requirements of DFS

Will find the solution when **L** is the depth of the shallowest goal

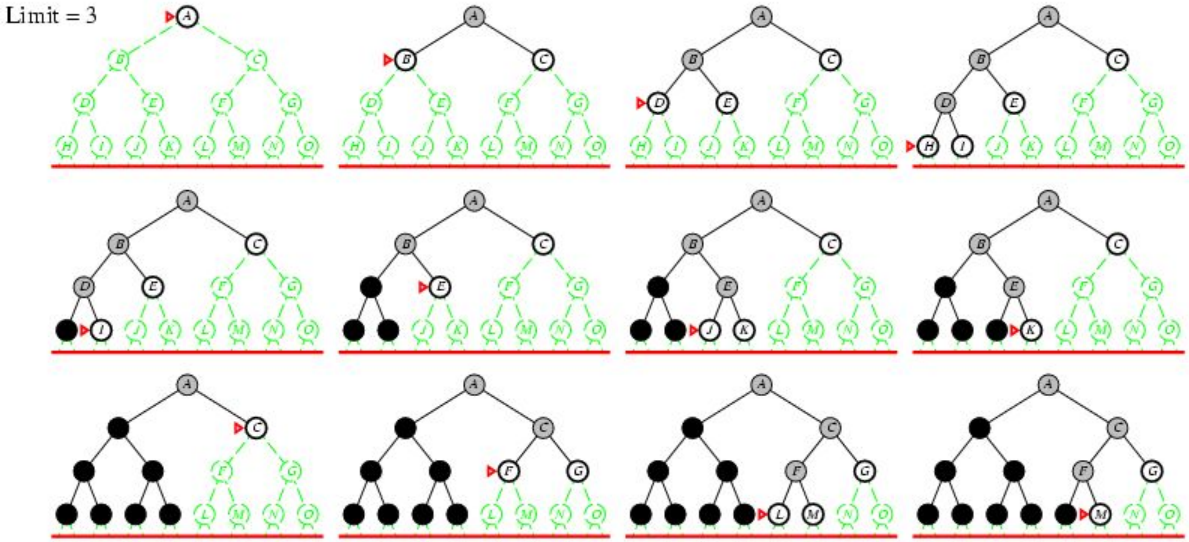
Iterative deepening search L = 0

Iterative deepening search L = 1

Iterative deepening search L = 2



Iterative deepening search L = 3



Time?

L = 0: 1
L = 1: $1 + b$
L = 2: $1 + b + b^2$
L = 3: $1 + b + b^2 + b^3$
...
L = d: $1 + b + b^2 + b^3 + \ldots + b^d$

Overall:

- $d(1) + (d-1)b + (d-2)b^2 + (d-3)b^3 + \ldots + b^d$
- $O(b^d)$
- the cost of the repeat of the lower levels is subsumed by the cost at the highest level

Properties of iterative deepening search

Space?

O(bd)

Complete?

Yes

Optimal?

Yes

Next week...