

# PROBLEM SOLVING AND SEARCH

## CHAPTER 3

## Outline

- ◇ Problem-solving agents
- ◇ Problem types
- ◇ Problem formulation
- ◇ Example problems
- ◇ Basic search algorithms

## Example: Romania

On holiday in Romania; currently in Arad.

Flight leaves tomorrow from Bucharest

Formulate goal:

be in Bucharest

Formulate problem:

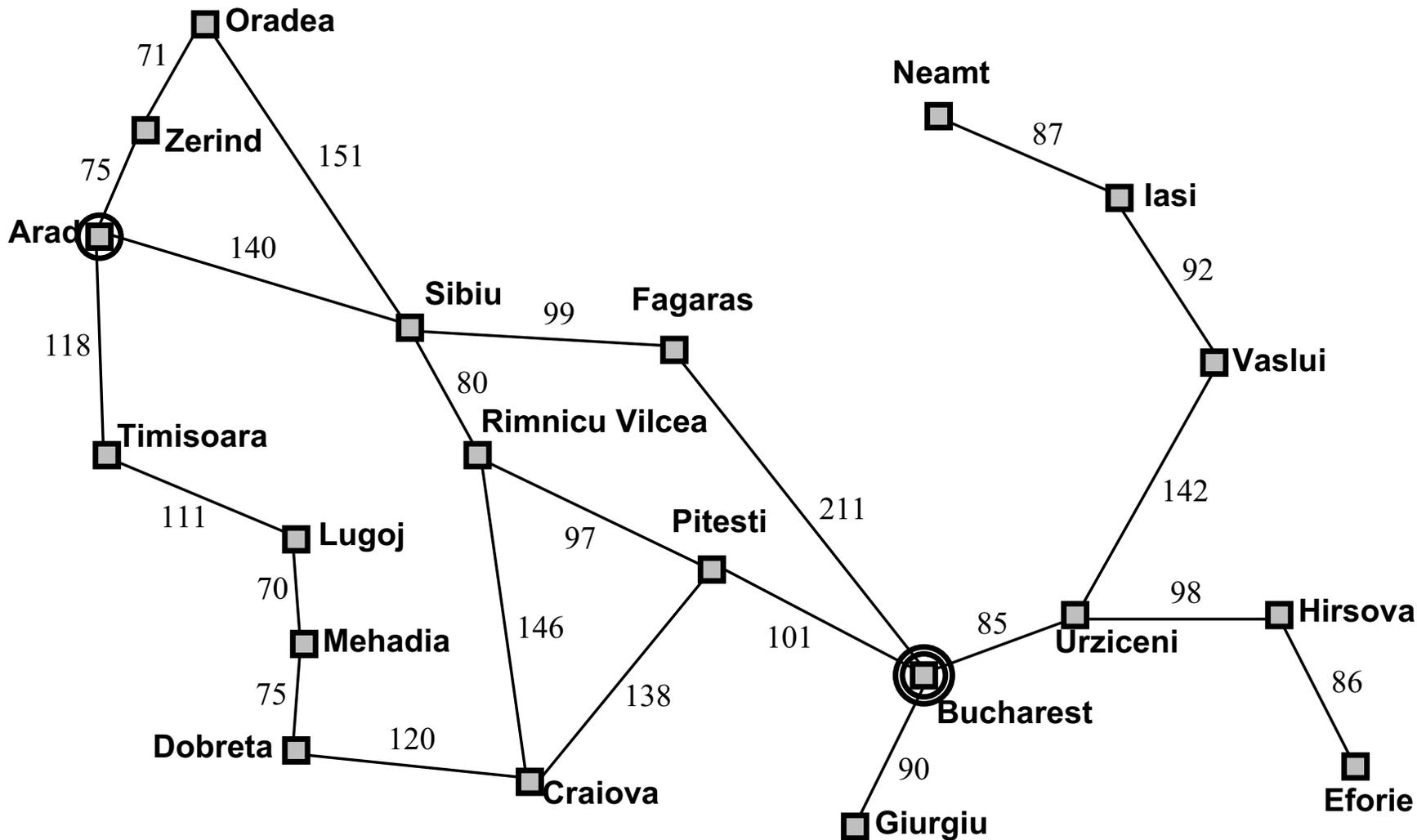
states: various cities

actions: drive between cities

Find solution:

sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

# Example: Romania



# Problem types

Deterministic, fully observable  $\implies$  single-state problem

Agent knows exactly which state it will be in; solution is a sequence

Non-observable  $\implies$  conformant problem

Agent may have no idea where it is; solution (if any) is a sequence

Nondeterministic and/or partially observable  $\implies$  contingency problem

percepts provide **new** information about current state

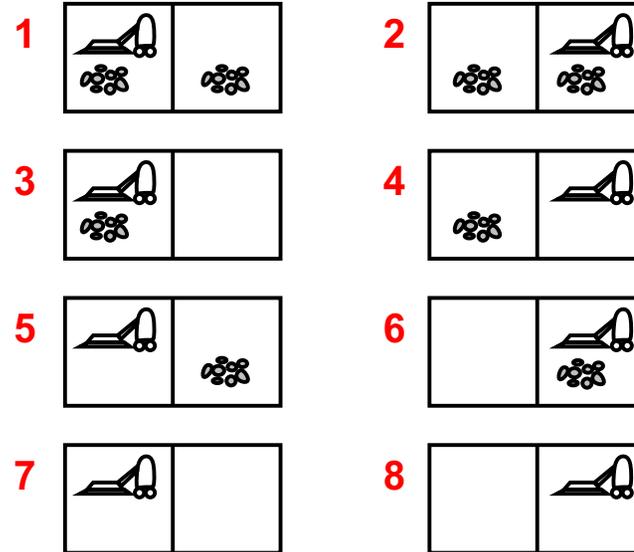
solution is a **contingent plan** or a **policy**

often **interleave** search, execution

Unknown state space  $\implies$  exploration problem (“online”)

# Example: vacuum world

Single-state, start in #5. [Solution??](#)



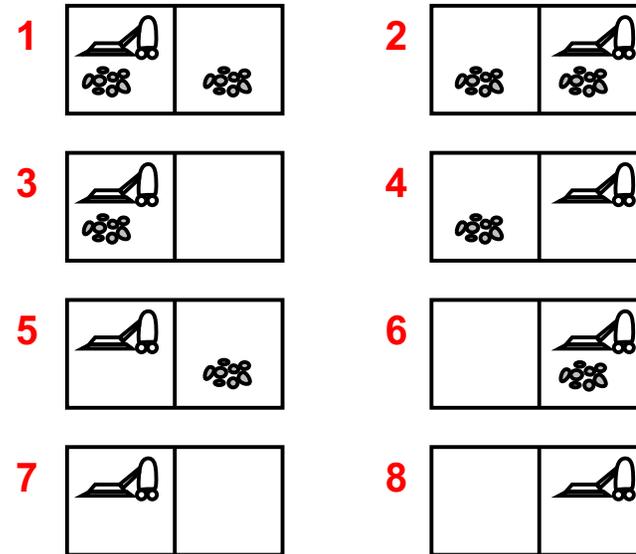
## Example: vacuum world

Single-state, start in #5. Solution??

[*Right, Suck*]

Conformant, start in {1, 2, 3, 4, 5, 6, 7, 8}

e.g., *Right* goes to {2, 4, 6, 8}. Solution??



## Example: vacuum world

Single-state, start in #5. Solution??

[*Right, Suck*]

Conformant, start in {1, 2, 3, 4, 5, 6, 7, 8}

e.g., *Right* goes to {2, 4, 6, 8}. Solution??

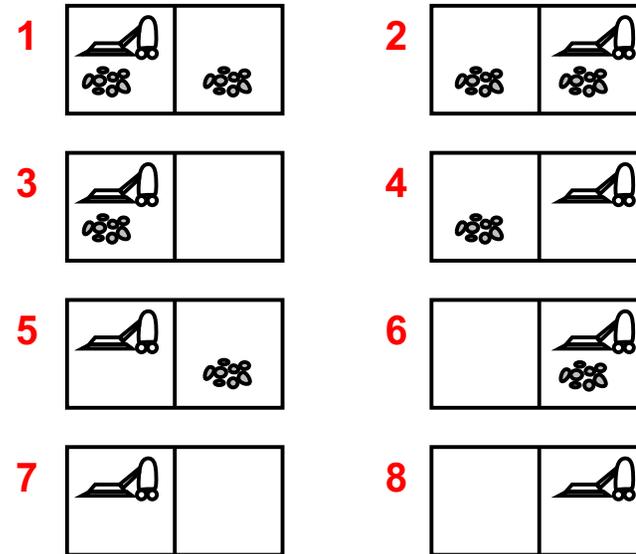
[*Right, Suck, Left, Suck*]

Contingency, start in #5

Murphy's Law: *Suck* can dirty a clean carpet

Local sensing: dirt, location only.

Solution??



## Example: vacuum world

Single-state, start in #5. Solution??

[*Right, Suck*]

Conformant, start in {1, 2, 3, 4, 5, 6, 7, 8}

e.g., *Right* goes to {2, 4, 6, 8}. Solution??

[*Right, Suck, Left, Suck*]

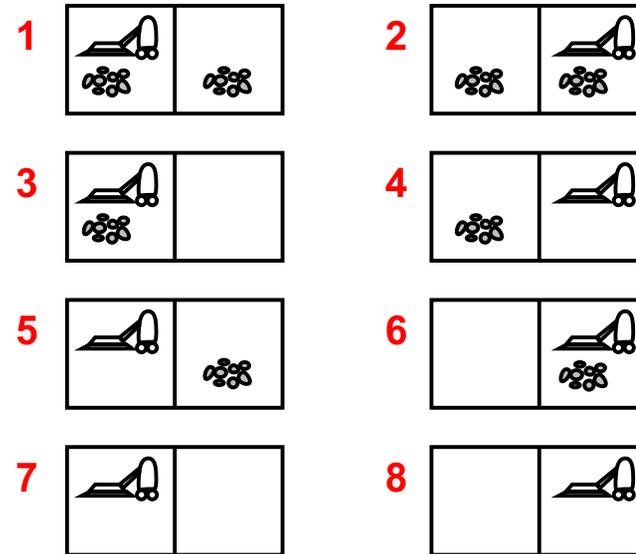
Contingency, start in #5

Murphy's Law: *Suck* can dirty a clean carpet

Local sensing: dirt, location only.

Solution??

[*Right, if dirt then Suck*]



## Single-state problem formulation

A **problem** is defined by four items:

**initial state** e.g., “at Arad”

**successor function**  $S(x)$  = set of action–state pairs

e.g.,  $S(\text{Arad}) = \{\langle \text{Arad} \rightarrow \text{Zerind}, \text{Zerind} \rangle, \dots\}$

**goal test**, can be

**explicit**, e.g.,  $x = \text{“at Bucharest”}$

**implicit**, e.g.,  $\text{NoDirt}(x)$

**path cost** (additive)

e.g., sum of distances, number of actions executed, etc.

$c(x, a, y)$  is the **step cost**, assumed to be  $\geq 0$

A **solution** is a sequence of actions  
leading from the initial state to a goal state

## Selecting a state space

Real world is absurdly complex

⇒ state space must be **abstracted** for problem solving

(Abstract) state = set of real states

(Abstract) action = complex combination of real actions

e.g., “Arad → Zerind” represents a complex set  
of possible routes, detours, rest stops, etc.

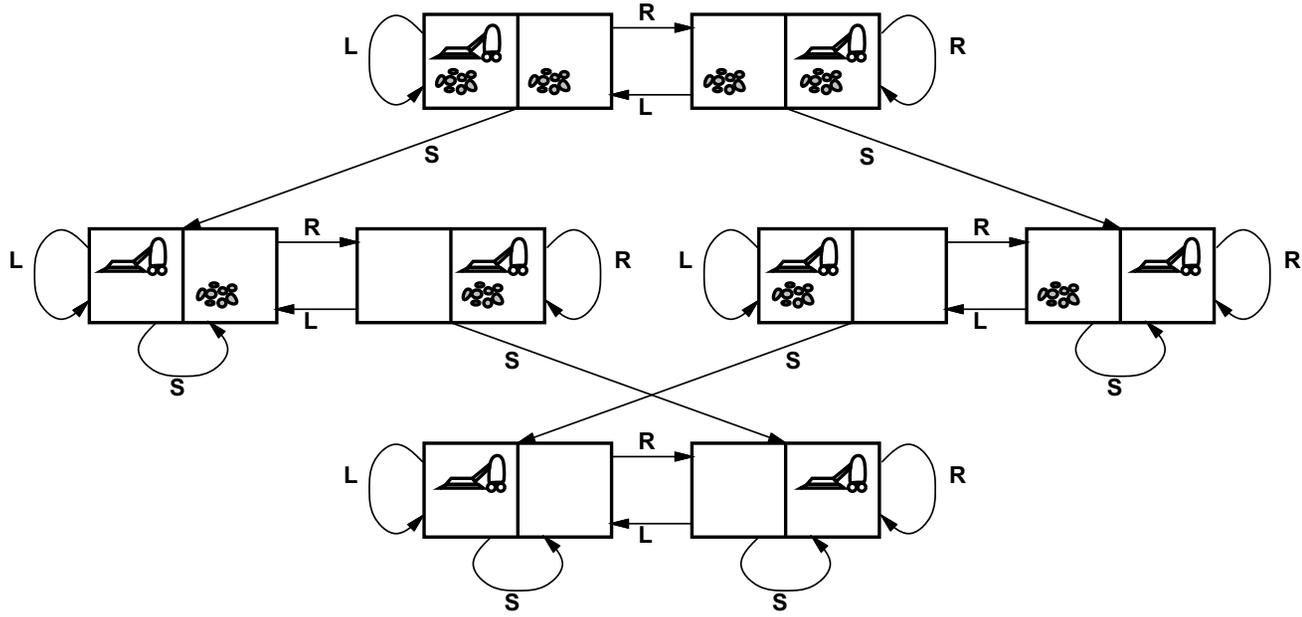
For guaranteed realizability, **any** real state “in Arad”  
must get to **some** real state “in Zerind”

(Abstract) solution =

set of real paths that are solutions in the real world

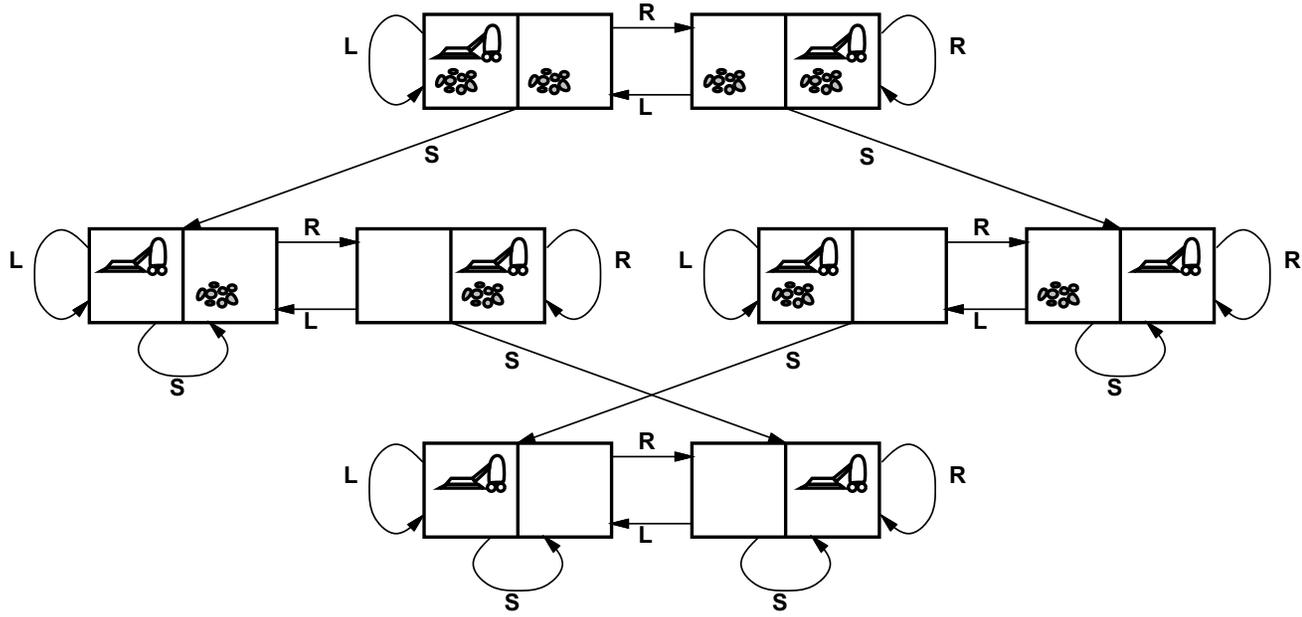
Each abstract action should be “easier” than the original problem!

# Example: vacuum world state space graph



- states??
- actions??
- goal test??
- path cost??

# Example: vacuum world state space graph



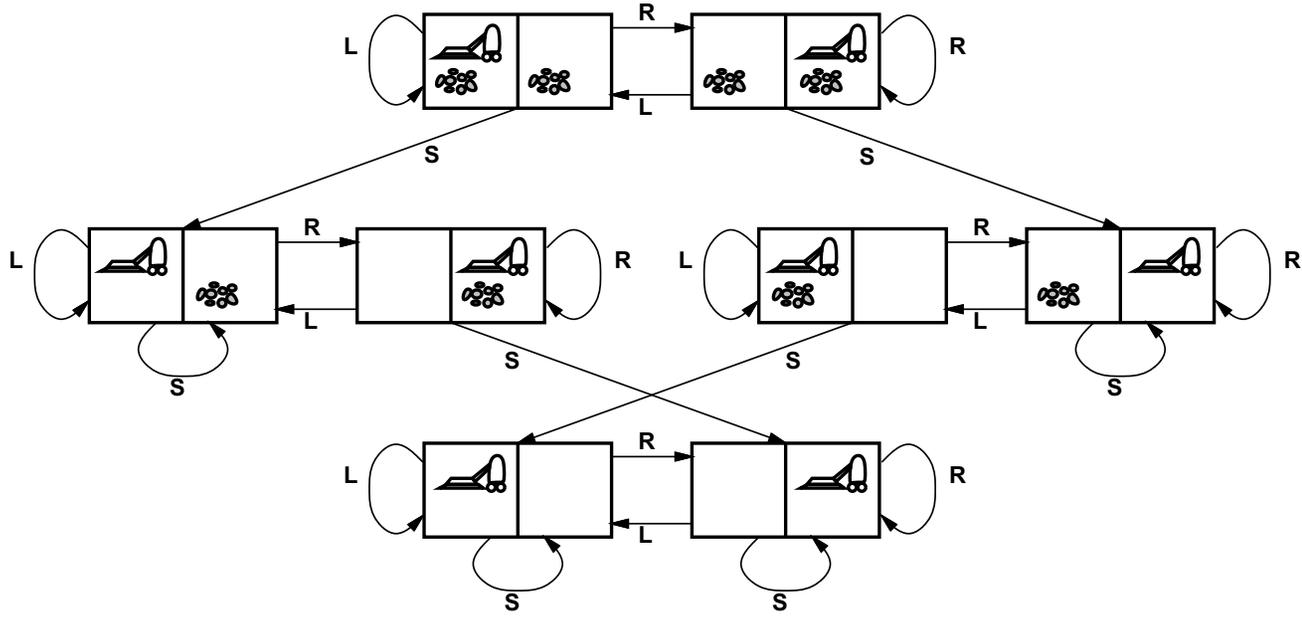
states??: integer dirt and robot locations (ignore dirt amounts etc.)

actions??

goal test??

path cost??

# Example: vacuum world state space graph



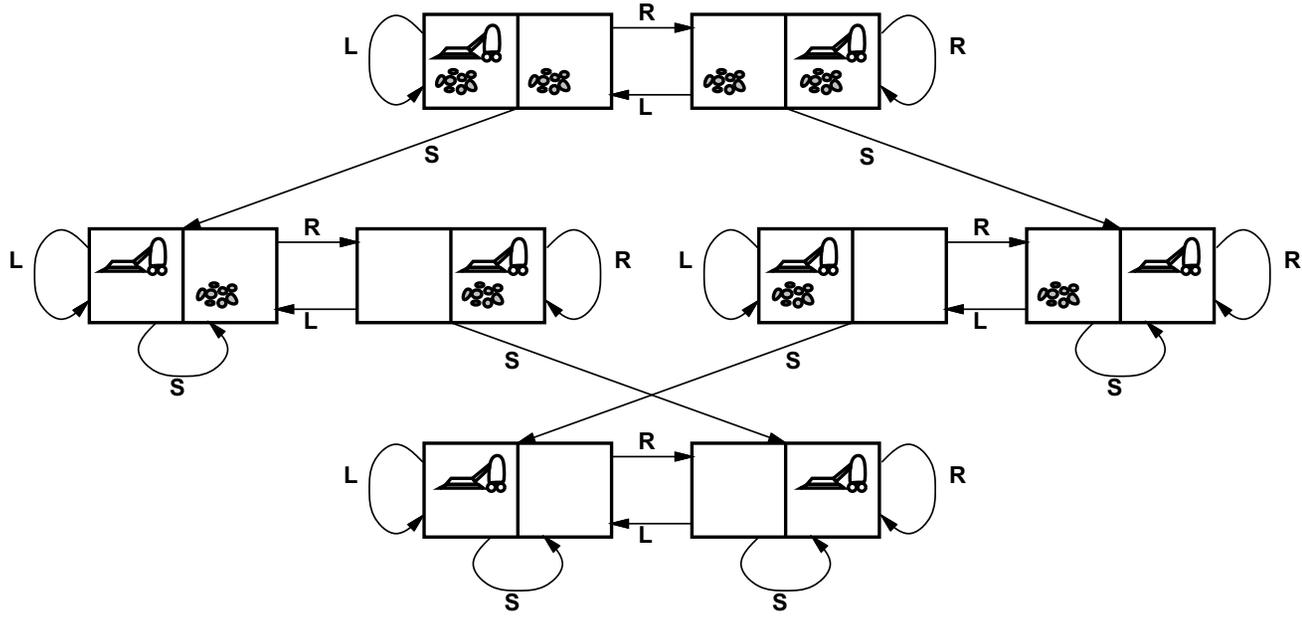
states??: integer dirt and robot locations (ignore dirt amounts etc.)

actions??: *Left, Right, Suck, NoOp*

goal test??

path cost??

# Example: vacuum world state space graph



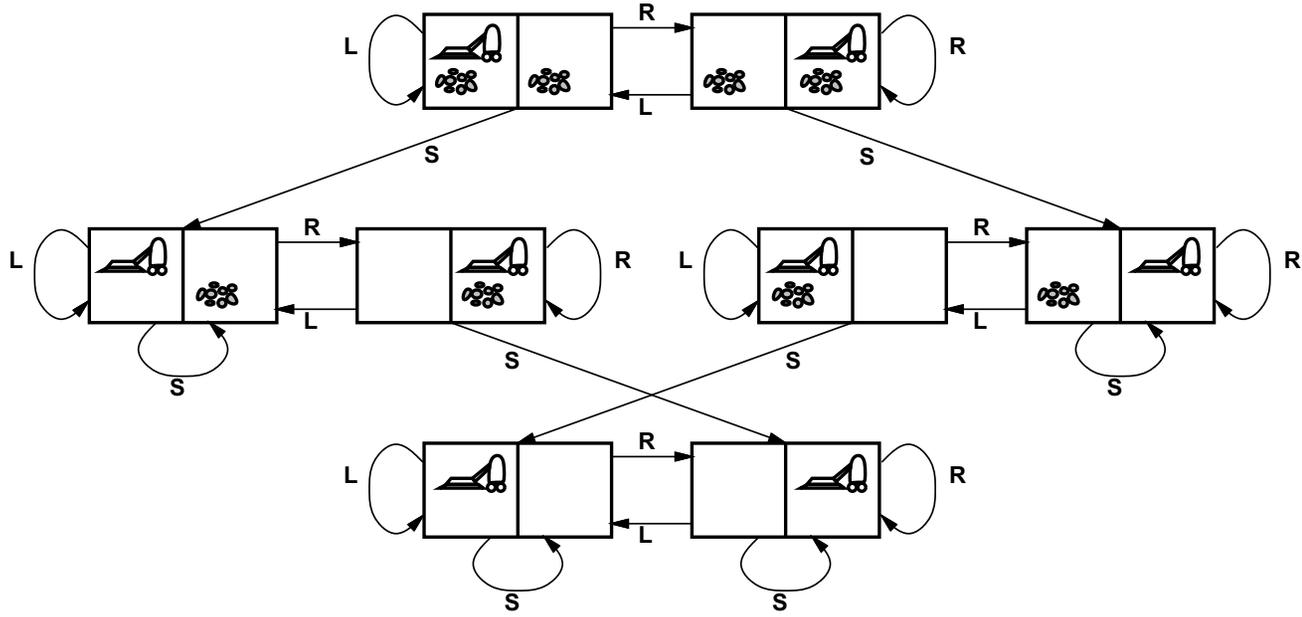
states??: integer dirt and robot locations (ignore dirt amounts etc.)

actions??: *Left, Right, Suck, NoOp*

goal test??: no dirt

path cost??

# Example: vacuum world state space graph



states??: integer dirt and robot locations (ignore dirt amounts etc.)

actions??: *Left, Right, Suck, NoOp*

goal test??: no dirt

path cost??: 1 per action (0 for *NoOp*)

## Example: The 8-puzzle

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

states??

actions??

goal test??

path cost??

## Example: The 8-puzzle

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

states??: integer locations of tiles (ignore intermediate positions)

actions??

goal test??

path cost??

## Example: The 8-puzzle

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

states??: integer locations of tiles (ignore intermediate positions)

actions??: move blank left, right, up, down (ignore unjamming etc.)

goal test??

path cost??

## Example: The 8-puzzle

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

states??: integer locations of tiles (ignore intermediate positions)

actions??: move blank left, right, up, down (ignore unjamming etc.)

goal test??: = goal state (given)

path cost??

## Example: The 8-puzzle

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

states??: integer locations of tiles (ignore intermediate positions)

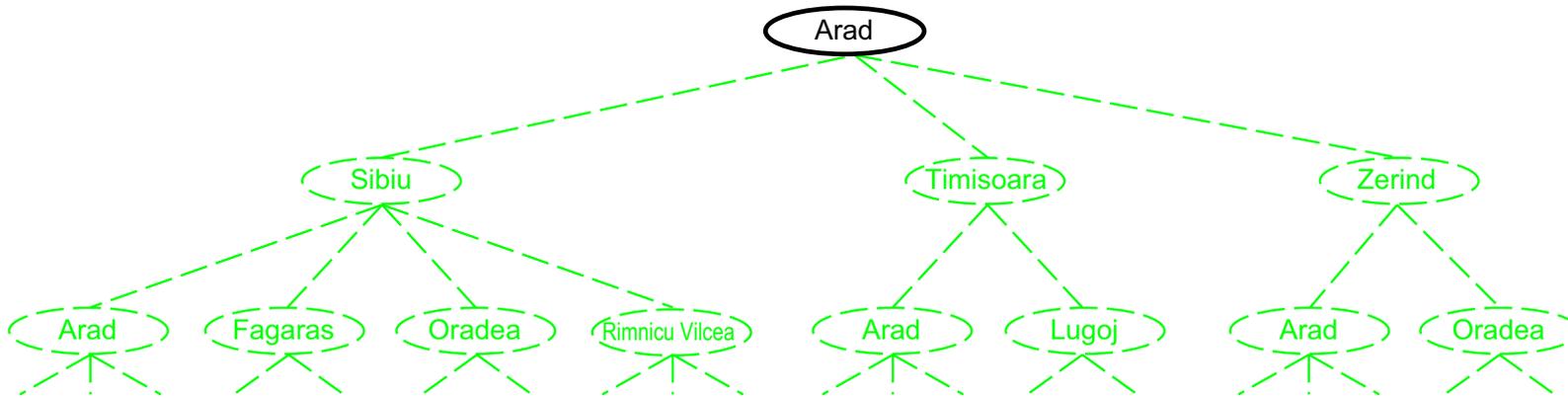
actions??: move blank left, right, up, down (ignore unjamming etc.)

goal test??: = goal state (given)

path cost??: 1 per move

[Note: optimal solution of  $n$ -Puzzle family is NP-hard]

# Tree search example

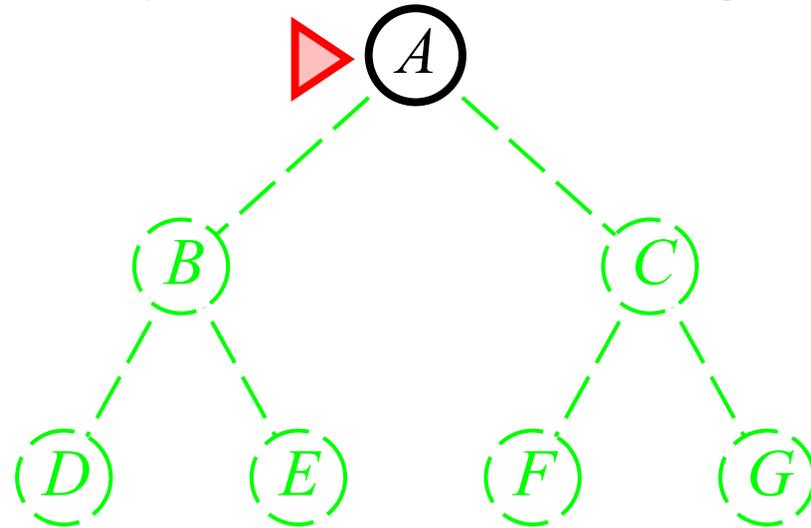


# Breadth-first search

Expand shallowest unexpanded node

## Implementation:

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

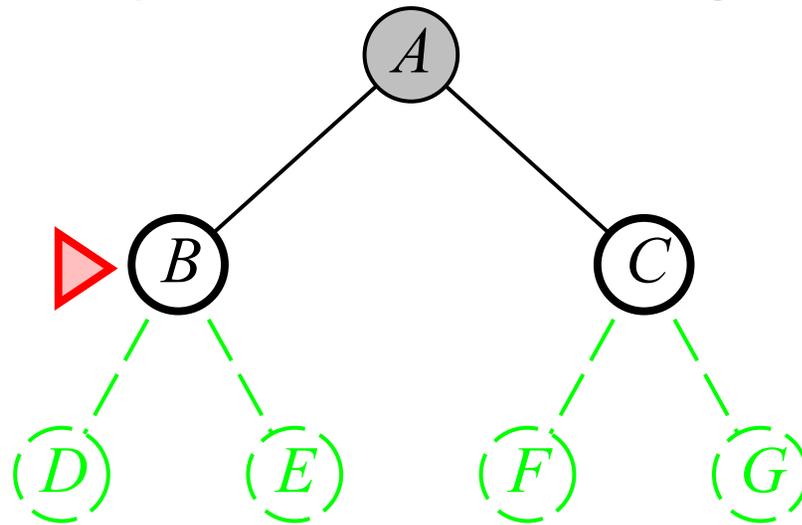


# Breadth-first search

Expand shallowest unexpanded node

## Implementation:

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

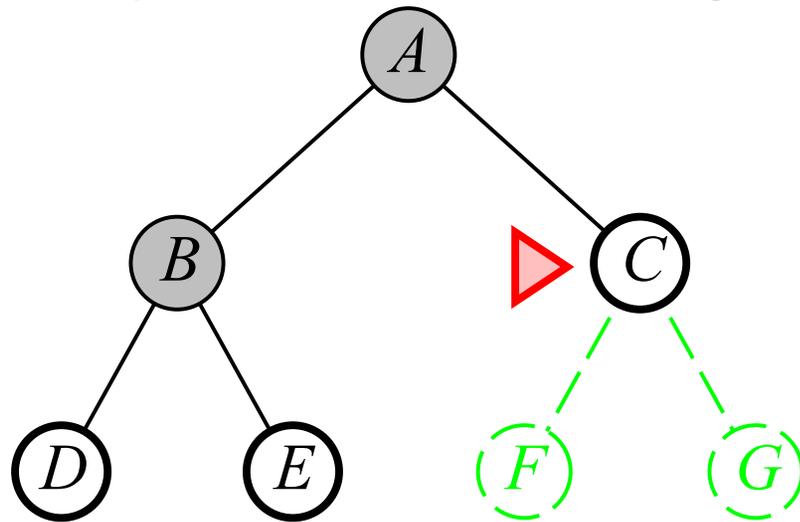


# Breadth-first search

Expand shallowest unexpanded node

## Implementation:

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

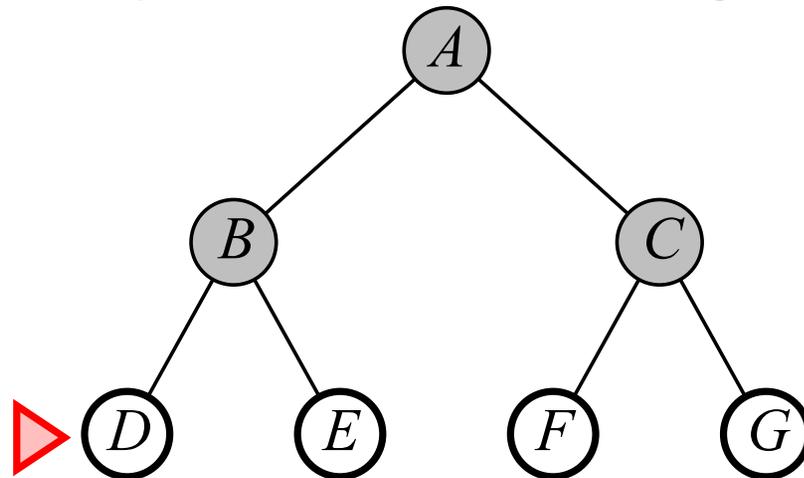


# Breadth-first search

Expand shallowest unexpanded node

## Implementation:

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



# Properties of breadth-first search

Complete??

## Properties of breadth-first search

Complete?? Yes (if  $b$  is finite)

Time??

## Properties of breadth-first search

Complete?? Yes (if  $b$  is finite)

Time??  $1 + b + b^2 + b^3 + \dots + b^d + b(b^d - 1) = O(b^{d+1})$ , i.e., exp. in  $d$

Space??

## Properties of breadth-first search

Complete?? Yes (if  $b$  is finite)

Time??  $1 + b + b^2 + b^3 + \dots + b^d + b(b^d - 1) = O(b^{d+1})$ , i.e., exp. in  $d$

Space??  $O(b^{d+1})$  (keeps every node in memory)

Optimal??

## Properties of breadth-first search

Complete?? Yes (if  $b$  is finite)

Time??  $1 + b + b^2 + b^3 + \dots + b^d + b(b^d - 1) = O(b^{d+1})$ , i.e., exp. in  $d$

Space??  $O(b^{d+1})$  (keeps every node in memory)

Optimal?? Yes (if cost = 1 per step); not optimal in general

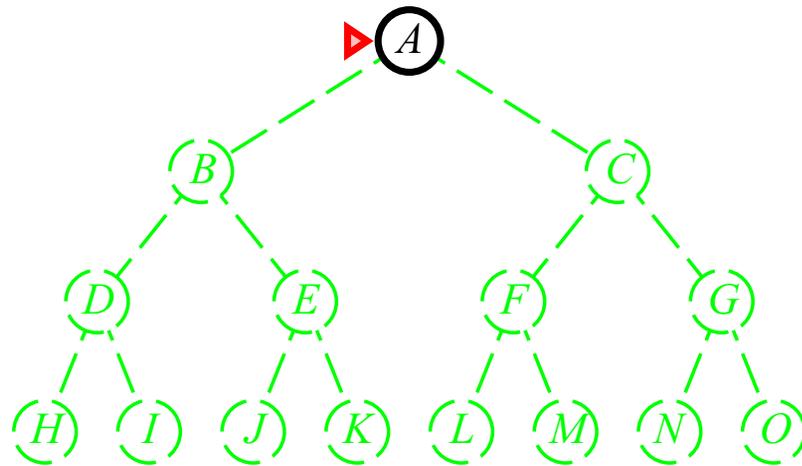
**Space** is the big problem; can easily generate nodes at 100MB/sec  
so 24hrs = 8640GB.

# Depth-first search

Expand deepest unexpanded node

## Implementation:

*fringe* = LIFO queue, i.e., put successors at front

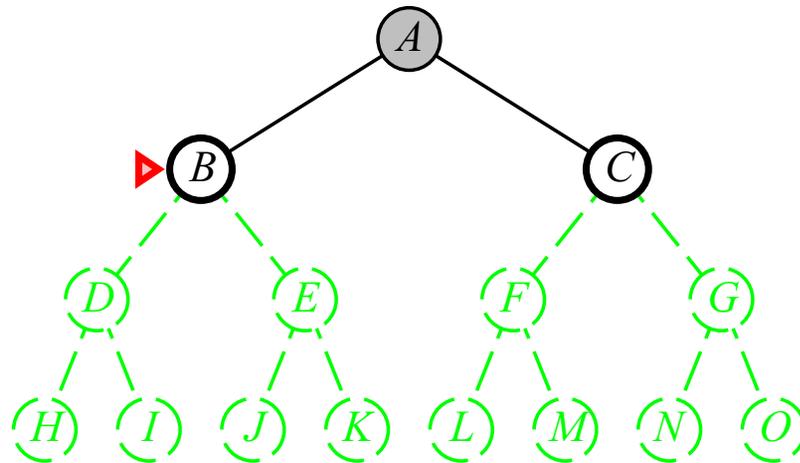


# Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front

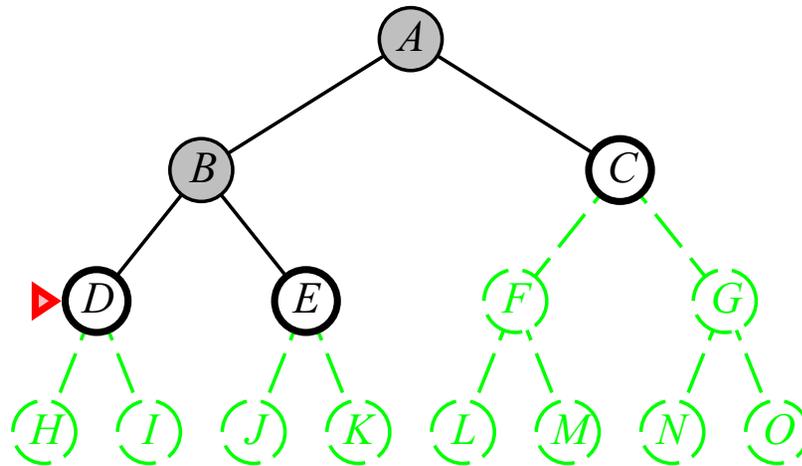


# Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front

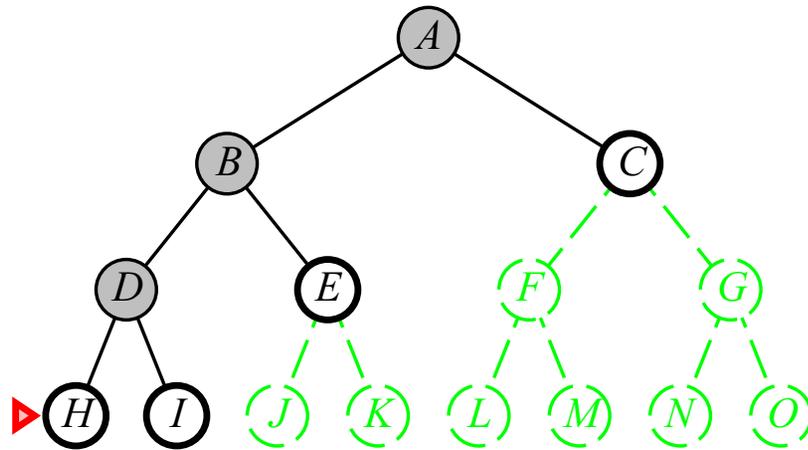


# Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front

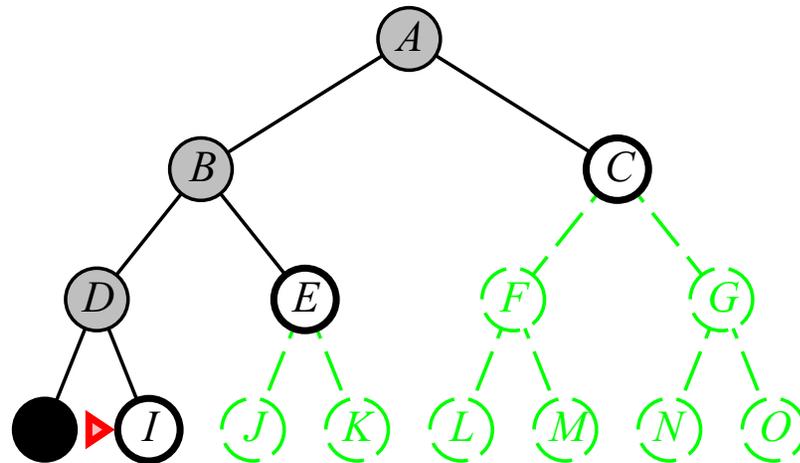


# Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front

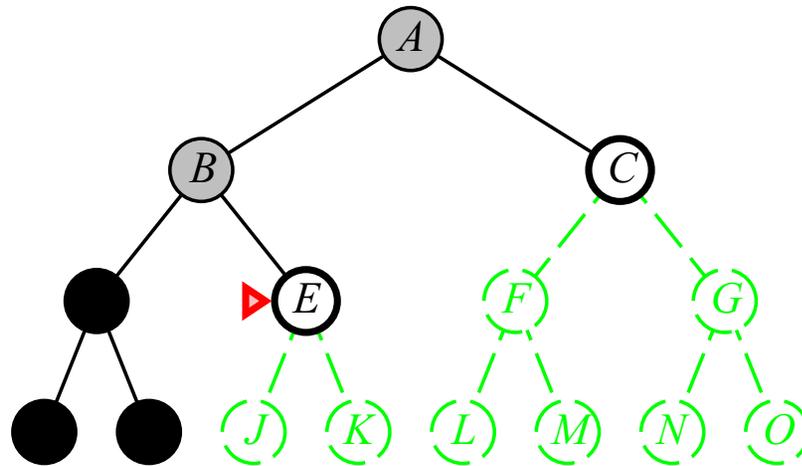


# Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front

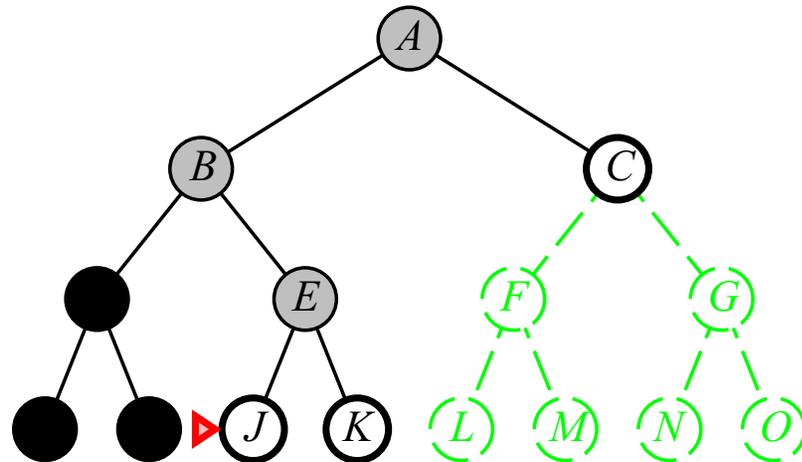


# Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front

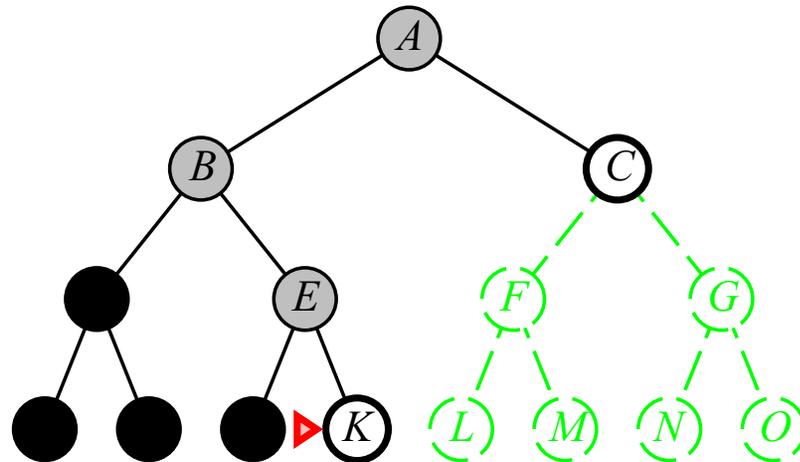


# Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front

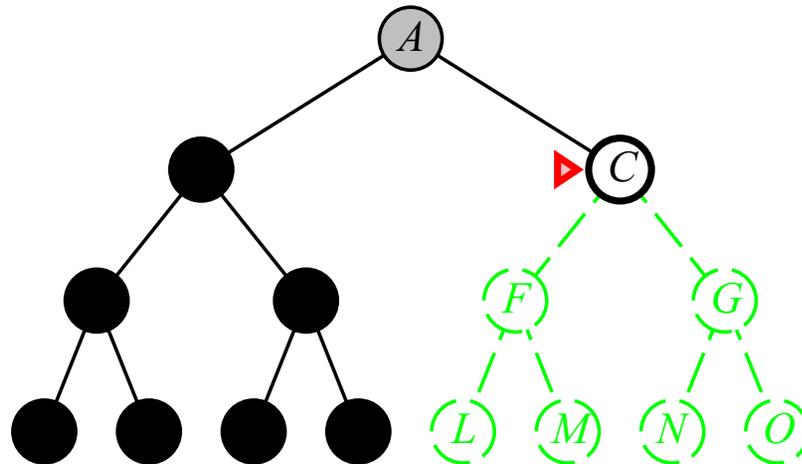


# Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front

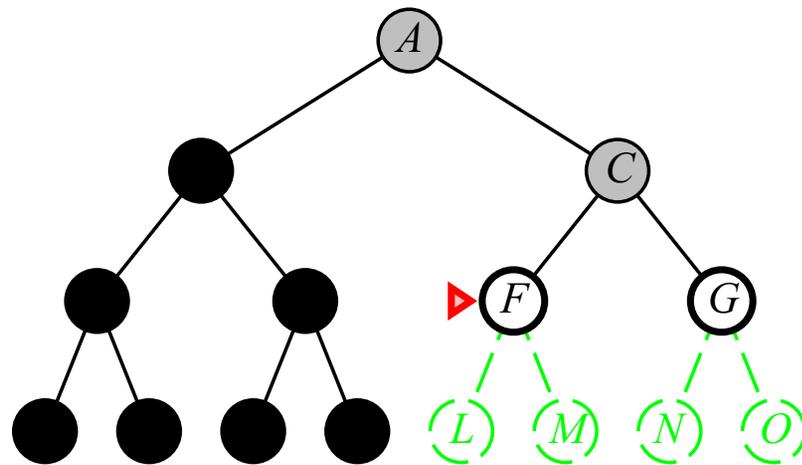


# Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front

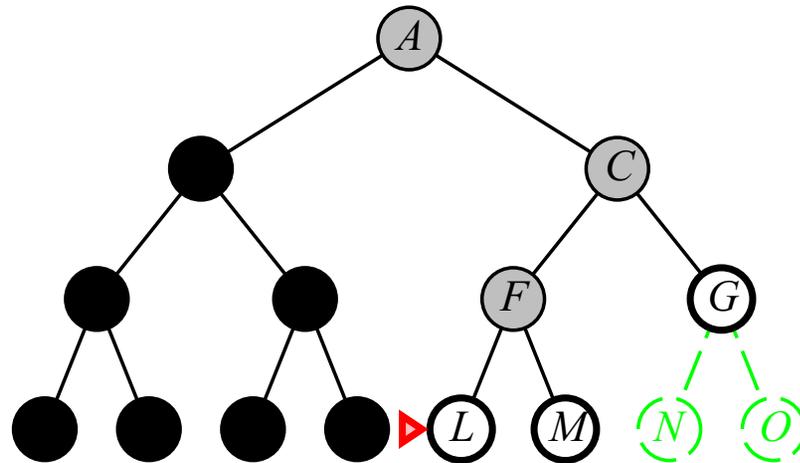


# Depth-first search

Expand deepest unexpanded node

## Implementation:

*fringe* = LIFO queue, i.e., put successors at front

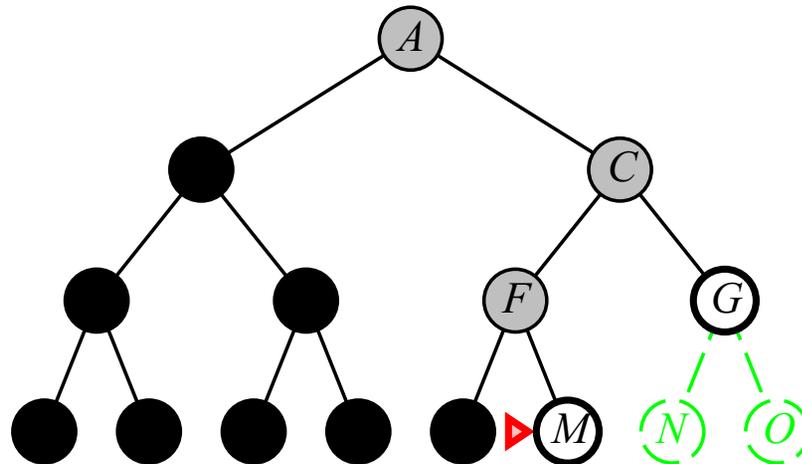


# Depth-first search

Expand deepest unexpanded node

## Implementation:

*fringe* = LIFO queue, i.e., put successors at front



# Properties of depth-first search

Complete??

## Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops

Modify to avoid repeated states along path

⇒ complete in finite spaces

Time??

## Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops

Modify to avoid repeated states along path

⇒ complete in finite spaces

Time??  $O(b^m)$ : terrible if  $m$  is much larger than  $d$

but if solutions are dense, may be much faster than breadth-first

Space??

## Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops

Modify to avoid repeated states along path

⇒ complete in finite spaces

Time??  $O(b^m)$ : terrible if  $m$  is much larger than  $d$

but if solutions are dense, may be much faster than breadth-first

Space??  $O(bm)$ , i.e., linear space!

Optimal??

## Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops

Modify to avoid repeated states along path

⇒ complete in finite spaces

Time??  $O(b^m)$ : terrible if  $m$  is much larger than  $d$

but if solutions are dense, may be much faster than breadth-first

Space??  $O(bm)$ , i.e., linear space!

Optimal?? No