Last chapter…

- Agents interact with environments through actuators and sensors
- The performance measure evaluates the environment sequence
- A perfectly rational agent maximizes expected performance
- PEAS descriptions define task environments

- Environments are categorized along several dimensions:

  observable? deterministic? episodic? static? discrete? single-agent?

- Several basic agent architectures exist:

  reflex, reflex with state, goal-based, utility-based

Solving Problem By Searching

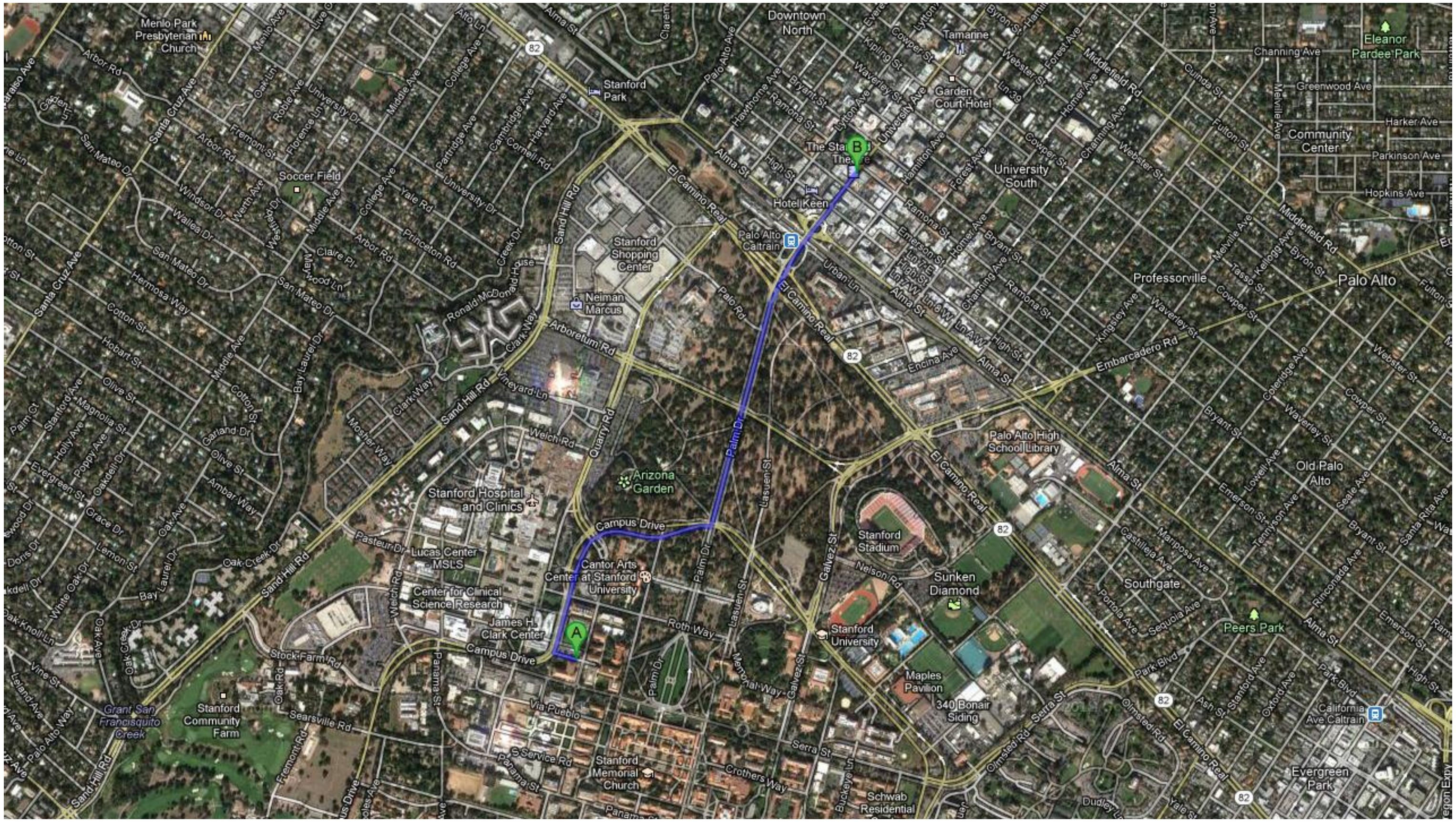
Chapter 3

Many AI Tasks can be Formulated as Search Problems

- Puzzles
- Games
- Navigation
- Assignment
- Layout
- Scheduling
- Routing

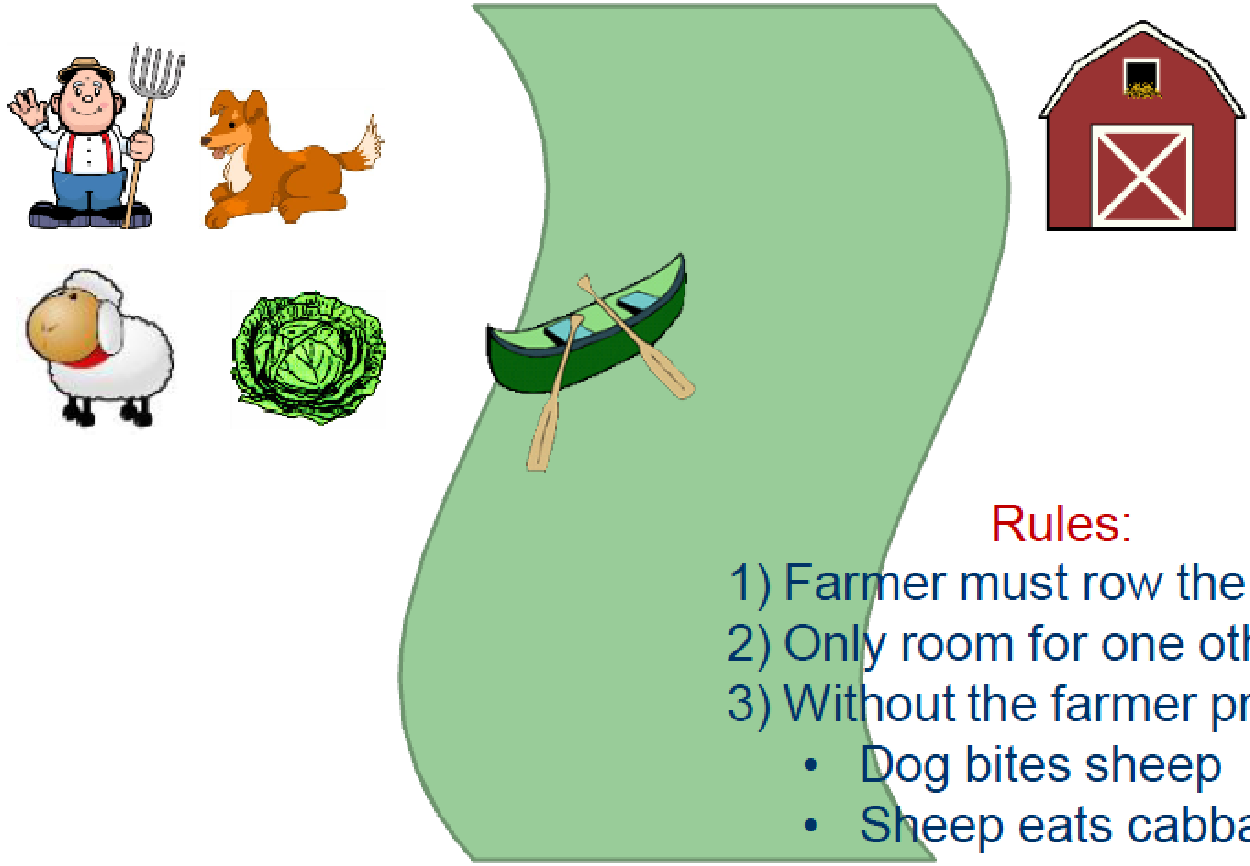
http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Search Example: Route Finding



http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

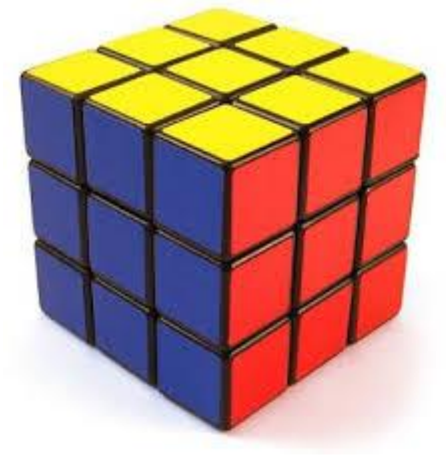
Search Example: River Crossing



Rules:

1) Farmer must row the boat
2) Only room for one other
3) Without the farmer present:
   - Dog bites sheep
   - Sheep eats cabbage

Search Example: Solving Puzzles



http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Search Example: Water Jugs

Given 4-liter and 3-liter pitchers, how do you get exactly 2 liters into the 4-liter pitcher?

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Outline

- Problem-solving agents/问题求解智能体
  - Goal-based agents
- Problem formulation
- Example problems
- Basic search algorithms

Problem-solving agents

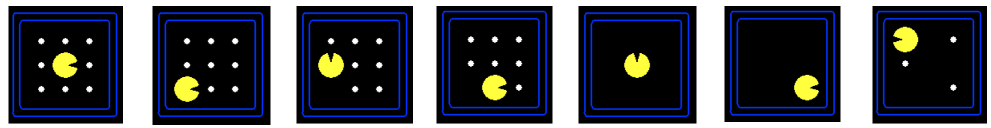

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
    static: seq, an action sequence, initially empty
            state, some description of the current world state
            goal, a goal, initially null
            problem, a problem formulation

    state ← UPDATE-STATE(state, percept)
    if seq is empty then do
        goal ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, goal)
        seq ← SEARCH(problem)
    action ← FIRST(seq)
    seq ← REST(seq)
    return action
```

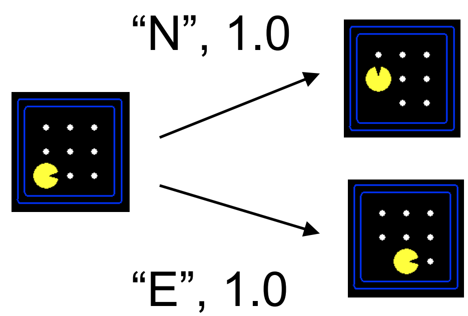
Note: this is offline problem solving; solution executed "eyes closed."
Online problem solving involves acting without complete knowledge.

Search Problems

- A search problem consists of:
  - A state space

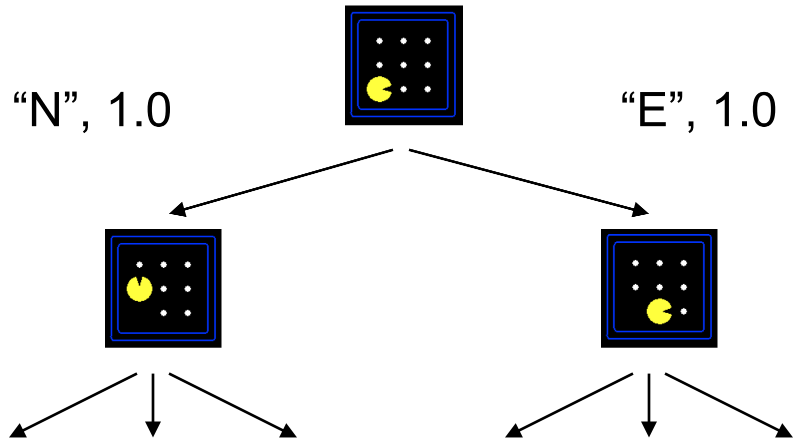


  - A successor function



  - A start state and a goal test
- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

Search Trees

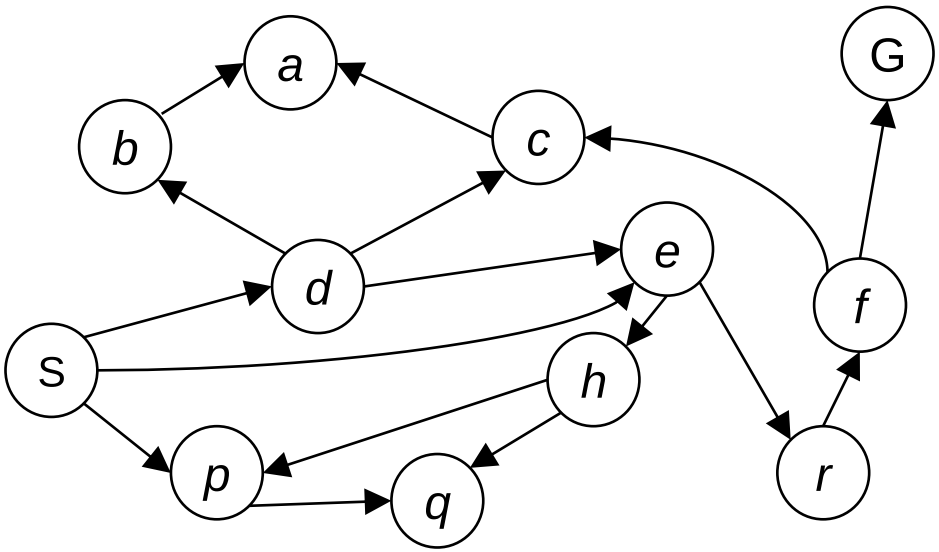


- A search tree:
  - This is a "what if" tree of plans and outcomes
  - Start state at the root node
  - Children correspond to successors
  - Nodes labeled with states, correspond to PLANS to those states
  - For most problems, can never actually build the whole tree
    - So, have to find ways of using only the important parts of the tree!

State Space Graphs

- There's some big graph in which
  - Each state is a node
  - Each successor is an outgoing arc

- Important: For most problems we could never actually build this graph



Laughably tiny search graph for a tiny search problem

Example: Romania

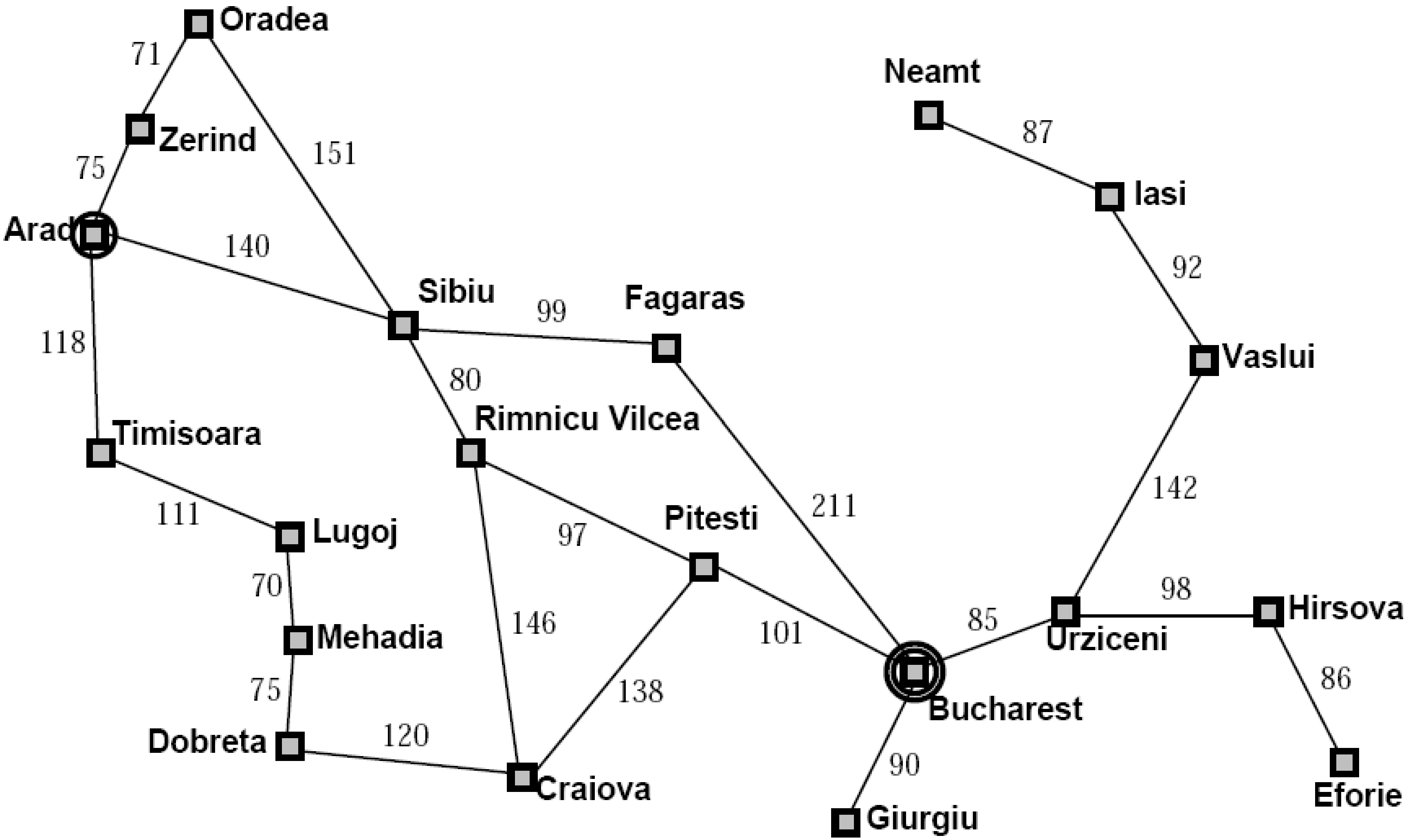
- On holiday in Romania; currently in Arad.
- Flight leaves tomorrow from Bucharest

- Formulate goal:
  - be in Bucharest

- Formulate problem:
  - states（状态）: various cities
  - actions（行为）: drive between cities

- Find solution:
  - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Example: Romania



Single-state problem formulation

A problem is defined by four items:

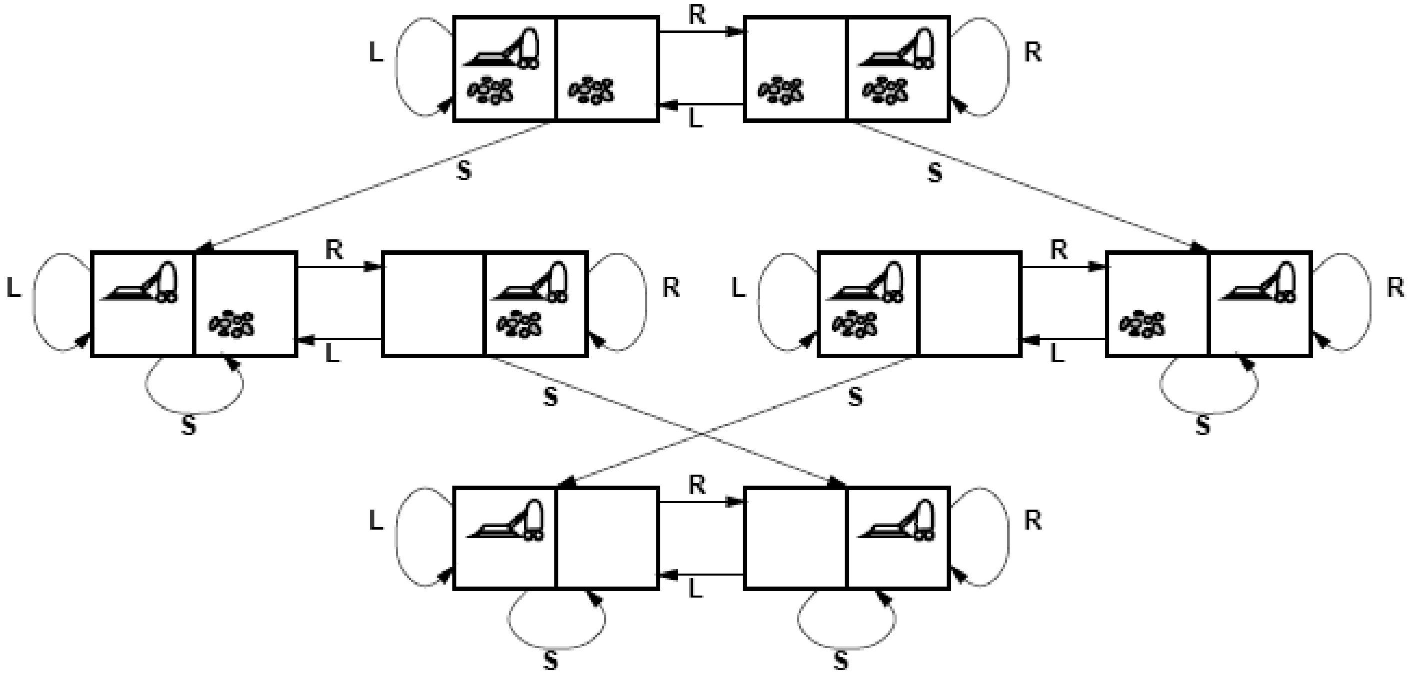
1. initial state（初始状态） e.g., “at Arad”
2. actions（行动） or successor function（后继函数） $S(x)$ = set of action–state pairs
   - e.g., $S(Arad) = \{<Arad \rightarrow Zerind, Zerind>, \ldots\}$
3. goal test（目标测试）, can be
   - explicit（明确的）, e.g., x = "at Bucharest"
   - implicit（隐含的）, e.g., $Checkmate(x)$
4. path cost（路径损耗） (additive)
   - e.g., sum of distances, number of actions executed, etc.
   - $c(x,a,y)$ is the step cost, assumed to be ≥ 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（抽象化） 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??

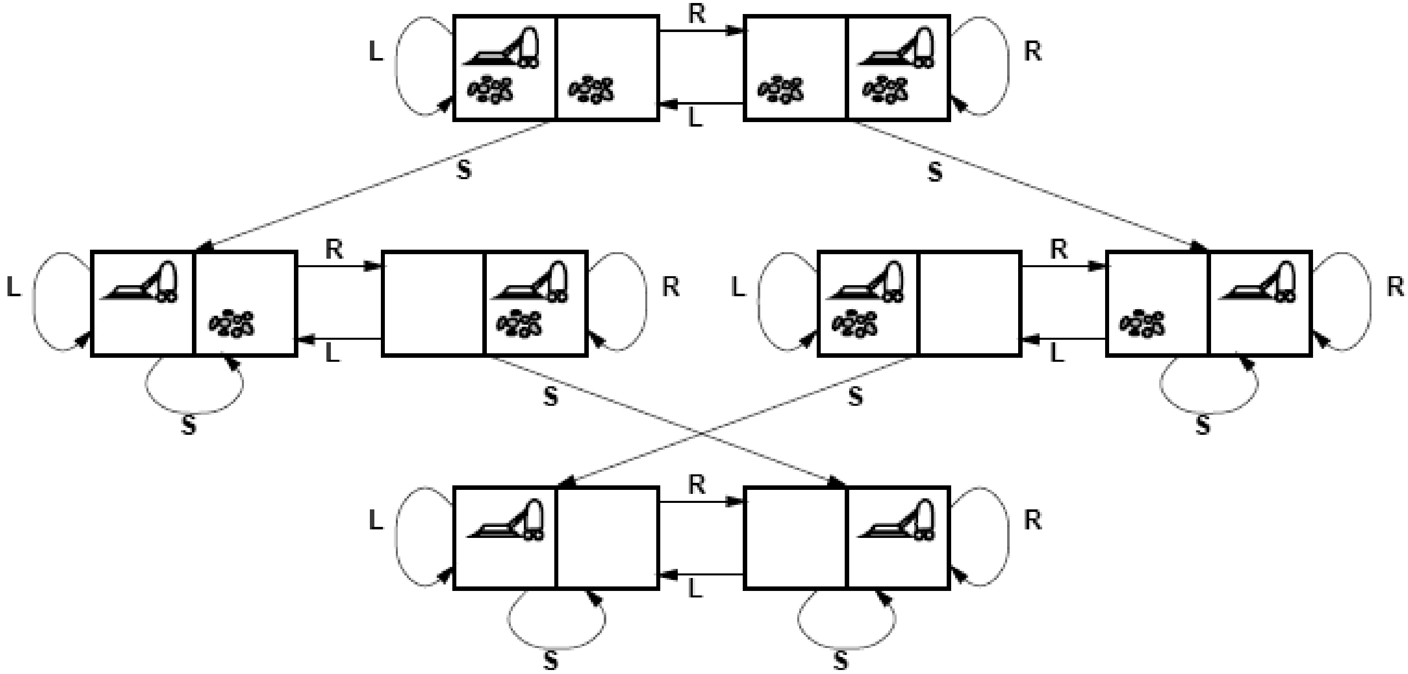
actions??

goal test??

path cost??

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Example: vacuum world state space graph



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

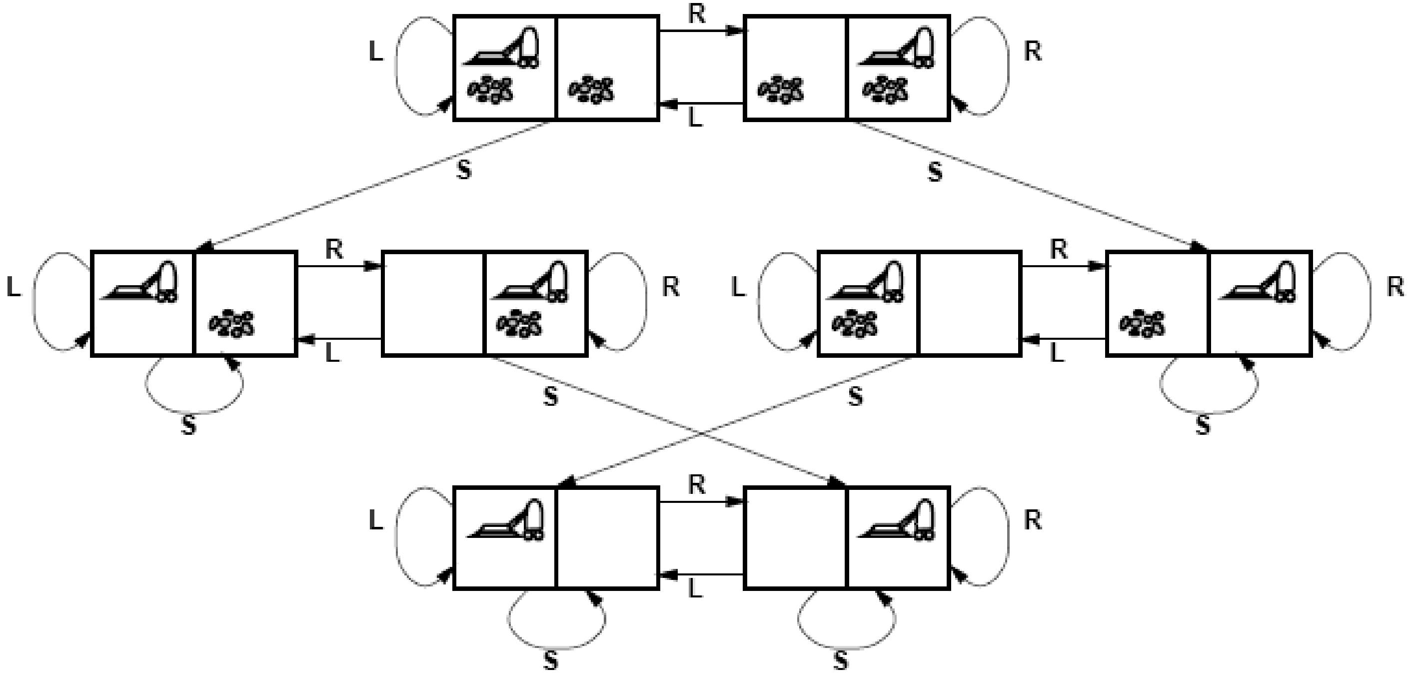
actions??

goal test??

path cost??

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Example: vacuum world state space graph



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

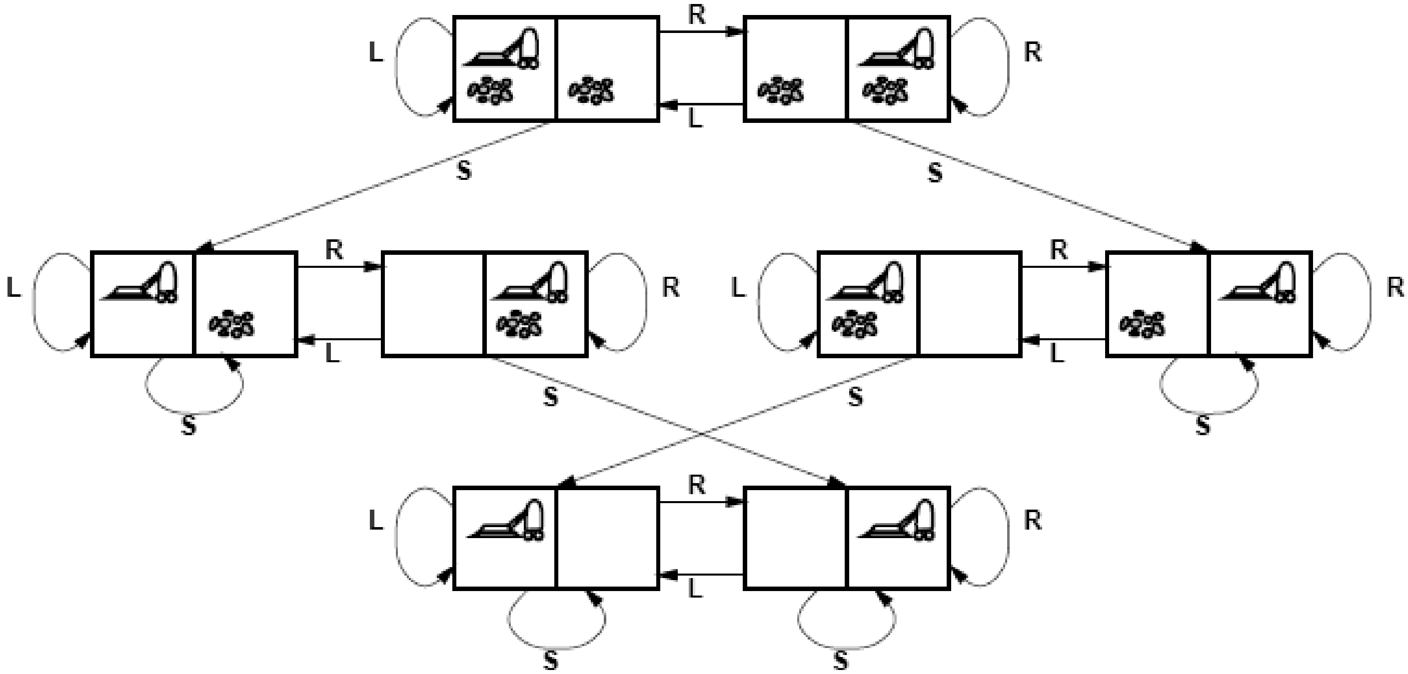
actions?? : *Left* , *Right* , *Suck* , *NoOp*

goal test??

path cost??

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Example: vacuum world state space graph



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

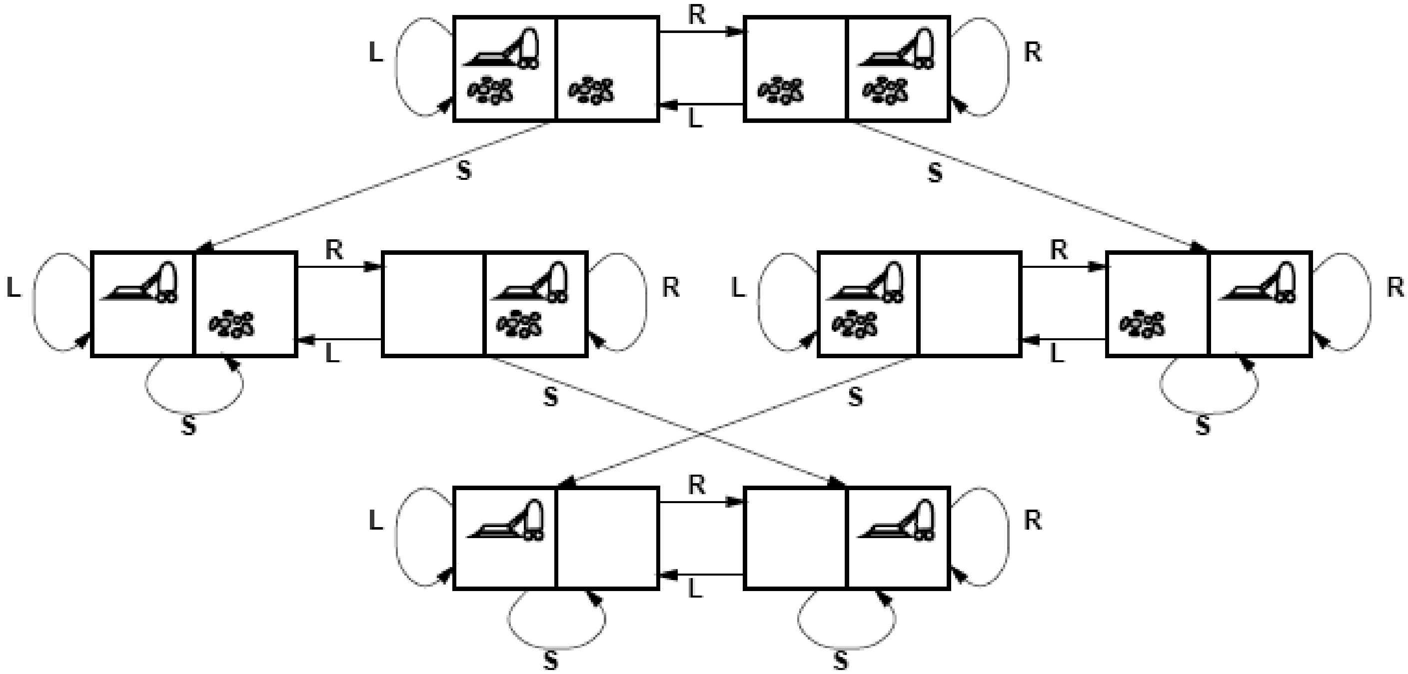
actions?? : *Left* , *Right* , *Suck* , *NoOp*

goal test?? : no dirt

path cost??

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Example: vacuum world state space graph



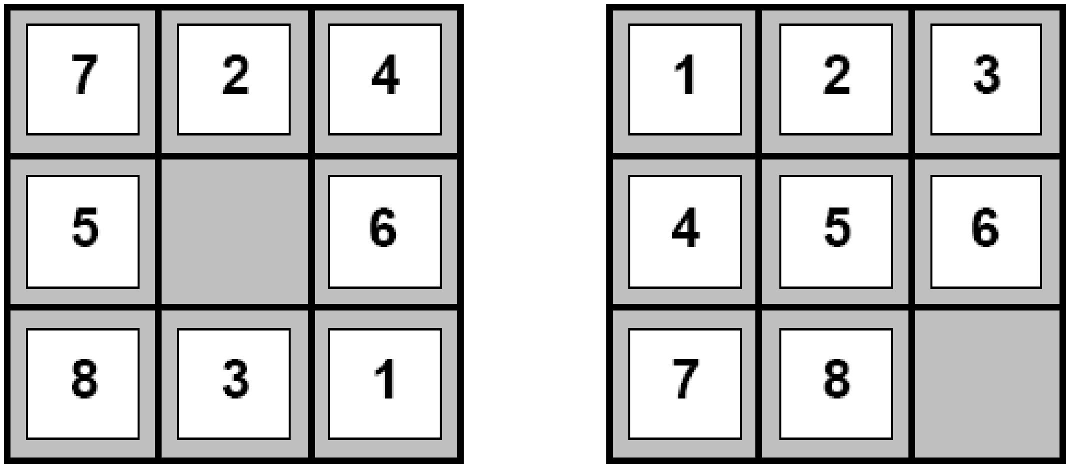
states?? : integer dirt and robot locations (ignore dirt amounts etc.)

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

goal test?? : no dirt

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

Example: The 8-puzzle



Start State   Goal State

states??

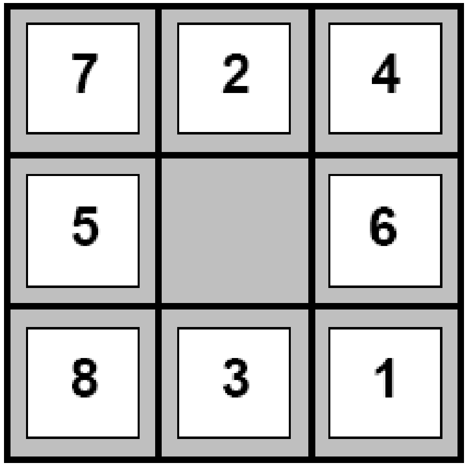
actions??

goal test??

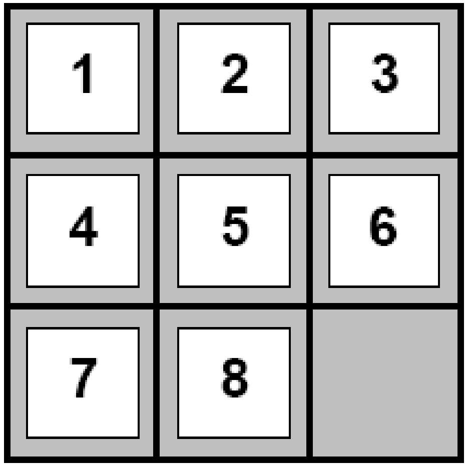
path cost??

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Example: The 8-puzzle



Start State



Goal State

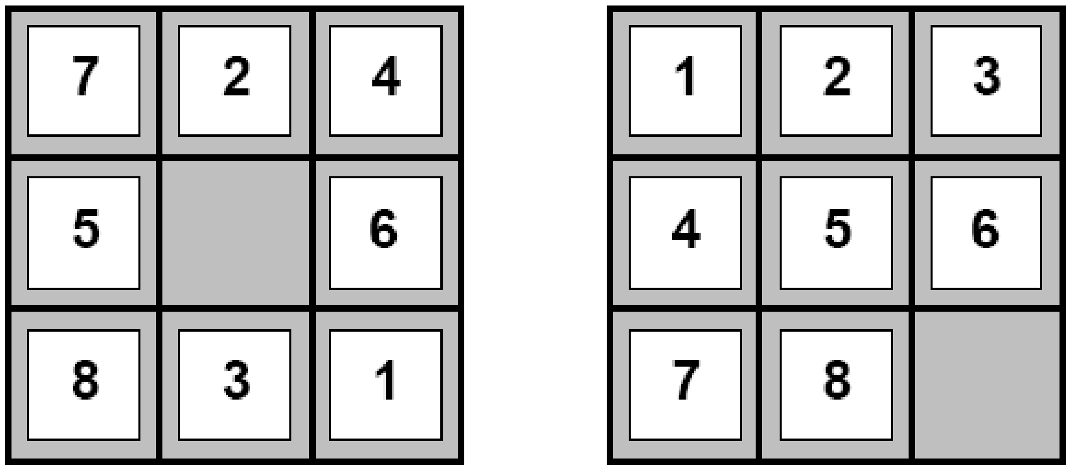
states?? : integer locations of tiles (ignore intermediate positions)

actions??

goal test??

path cost??

Example: The 8-puzzle



Start State

Goal State

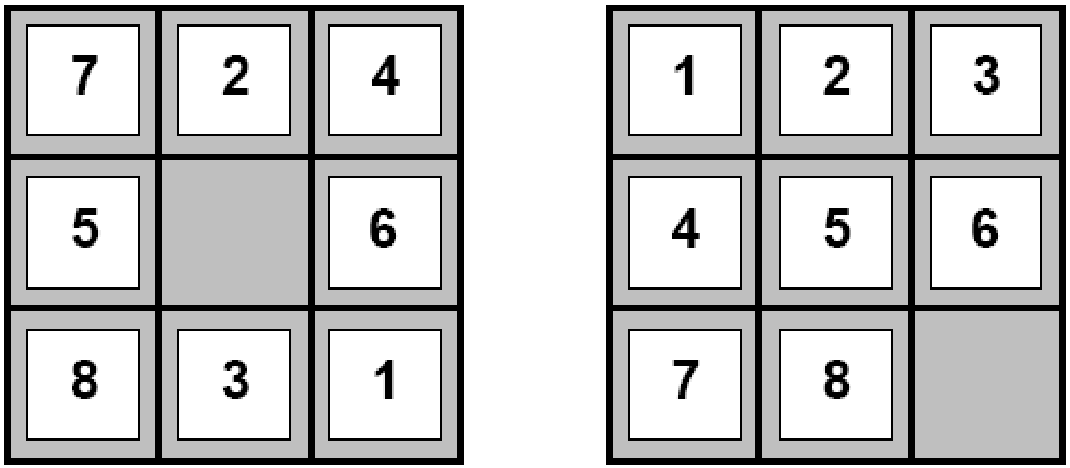
states?? : integer locations of tiles (ignore intermediate positions)

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

goal test??

path cost??

Example: The 8-puzzle



Start State Goal State

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

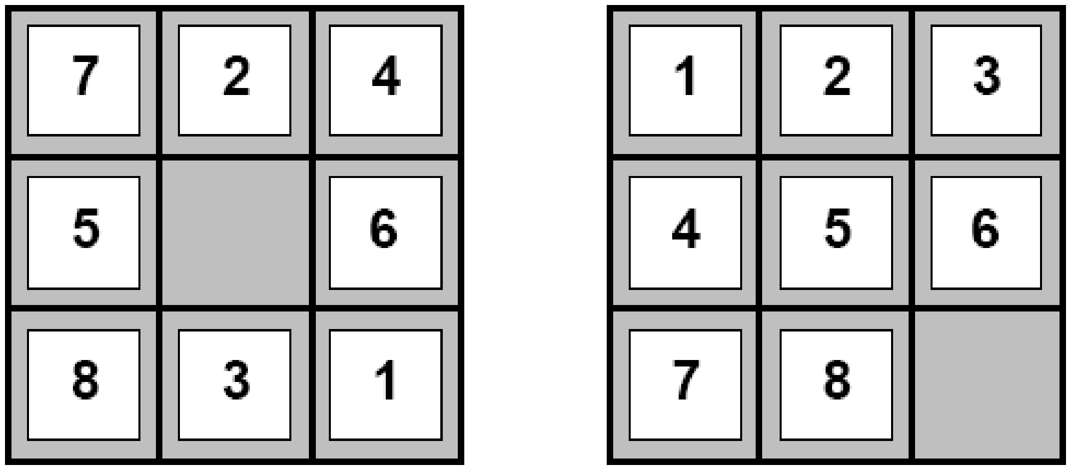
actions?? : move blank left, right, up, down (ignore unjamming etc.)

goal test?? : = goal state (given)

path cost??

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Example: The 8-puzzle



Start State Goal State

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

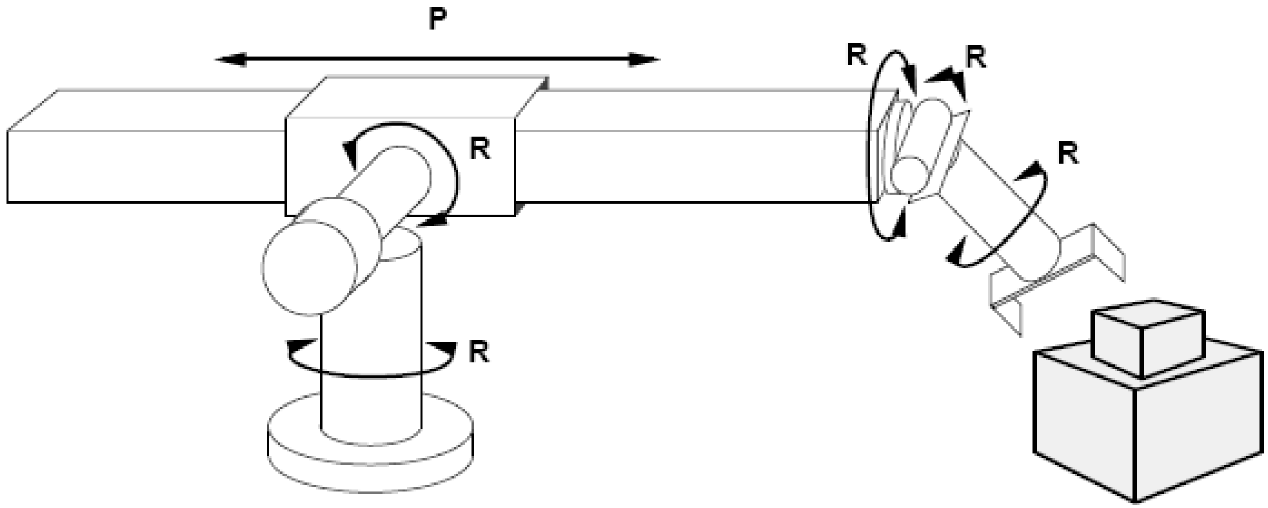
actions?? : move blank left, right, up, down (ignore unjamming etc.)

goal test?? : = goal state (given)

path cost?? 1 per move

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

Example: robotic assembly



states?? : real-valued coordinates（坐标） of robot joint angles
parts of the object to be assembled

actions?? : continuous motions of robot joints

goal test?? : complete assembly with no robot included!

path cost?? : time to execute

Outline

- Problem-solving agents/问题求解智能体
  - Goal-based agents
- Problem formulation
- Example problems
- Basic search algorithms

Tree search algorithms — 搜索树

Basic idea:

offline, simulated exploration of state space

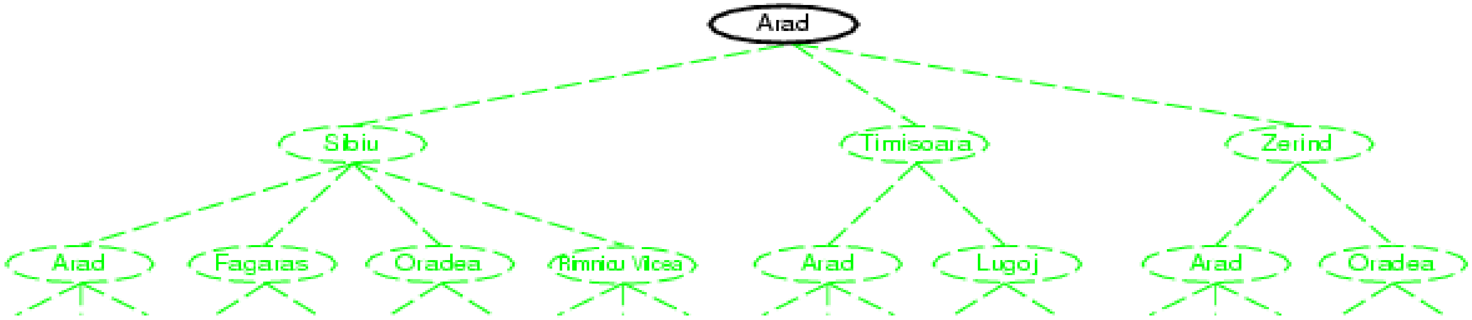
by generating successors of already-explored states

(a.k.a. expanding states)

```
function Tree-Search (problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
     (根据不同策略选择扩展节点)
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
   end
```

Strategy employed determines the type of tree search performed

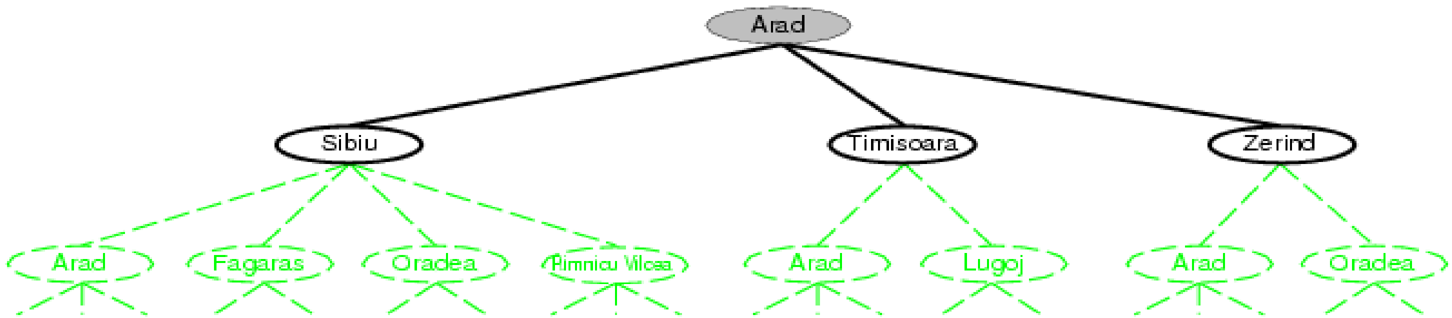
Tree search example



function TREE-SEARCH(*problem*, *strategy*) **returns** a solution, or failure
  initialize the search tree using the initial state of *problem*
  loop do
    if there are no candidates for expansion **then return** failure
    choose a leaf node for expansion according to *strategy*
    if the node contains a goal state **then return** the corresponding solution
    else expand the node and add the resulting nodes to the search tree

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

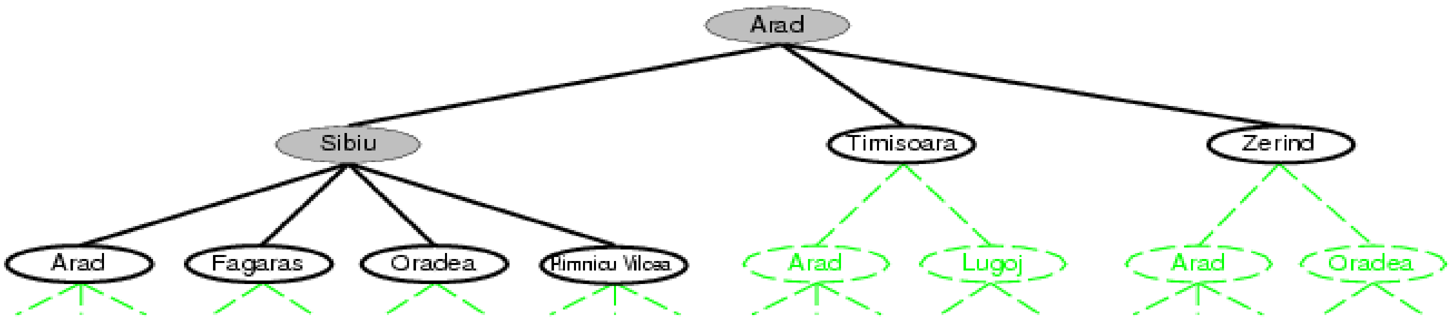
Tree search example



function TREE-SEARCH(*problem*, *strategy*) **returns** a solution, or failure
    initialize the search tree using the initial state of *problem*
    loop do
        if there are no candidates for expansion **then return** failure
        choose a leaf node for expansion according to *strategy*
        if the node contains a goal state **then return** the corresponding solution
        else expand the node and add the resulting nodes to the search tree

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Tree search example

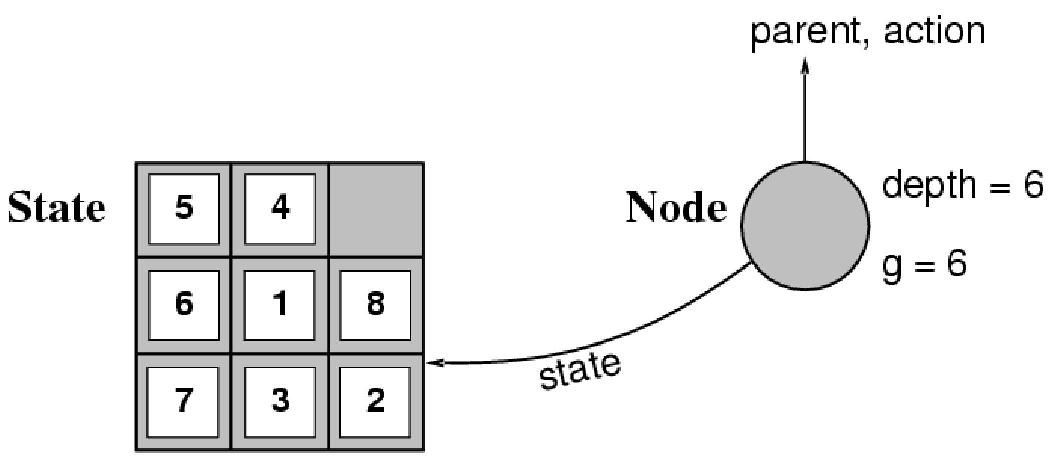


function TREE-SEARCH(*problem*, *strategy*) **returns** a solution, or failure
  initialize the search tree using the initial state of *problem*
  loop do
    if there are no candidates for expansion **then return** failure
    choose a leaf node for expansion according to *strategy*
    if the node contains a goal state **then return** the corresponding solution
    else expand the node and add the resulting nodes to the search tree

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Implementation: 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(n)*, depth



The `Expand` function creates new nodes, filling in the various fields and using the `SuccessorFn` 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
```

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Search strategies

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

- Strategies are evaluated along the following dimensions:
  - completeness（完备性）: does it always find a solution if one exists?
  - time complexity（时间复杂度）: number of nodes generated
  - space complexity（空间复杂度） : maximum number of nodes in memory
  - optimality（最优性）: does it always find a least-cost solution?

- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree — 分支因子
  - *d:* depth of the least-cost solution — 最浅的目标节点的深度
  - *m*: maximum depth of the state space (may be ∞) — 最大深度

Uninformed（无信息的） search strategies

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

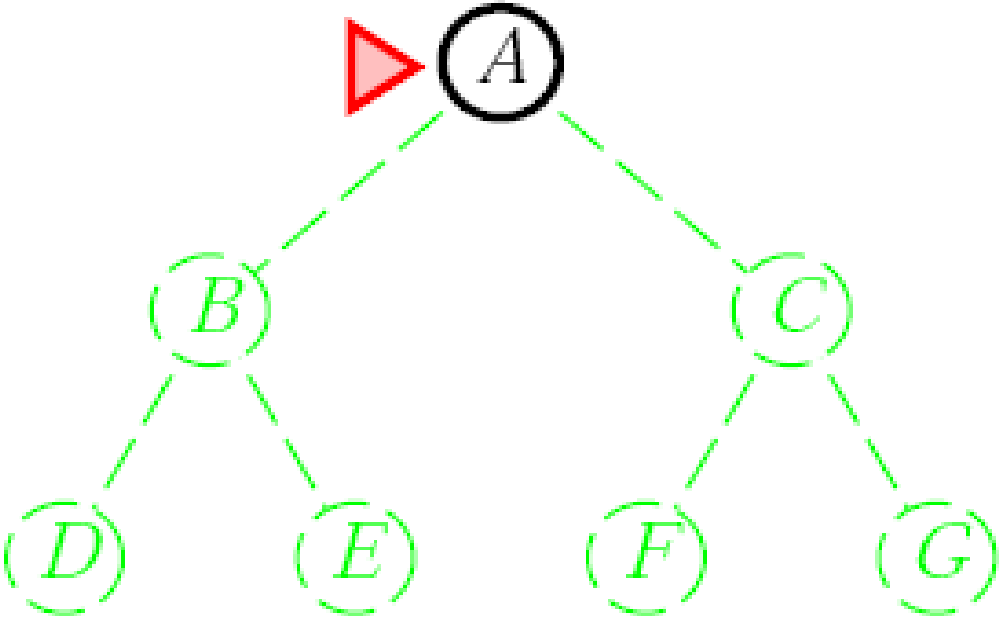
- Breadth-first search（广度优先搜索）
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

Breadth-first search

- Expand shallowest unexpanded node

- Implementation:

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



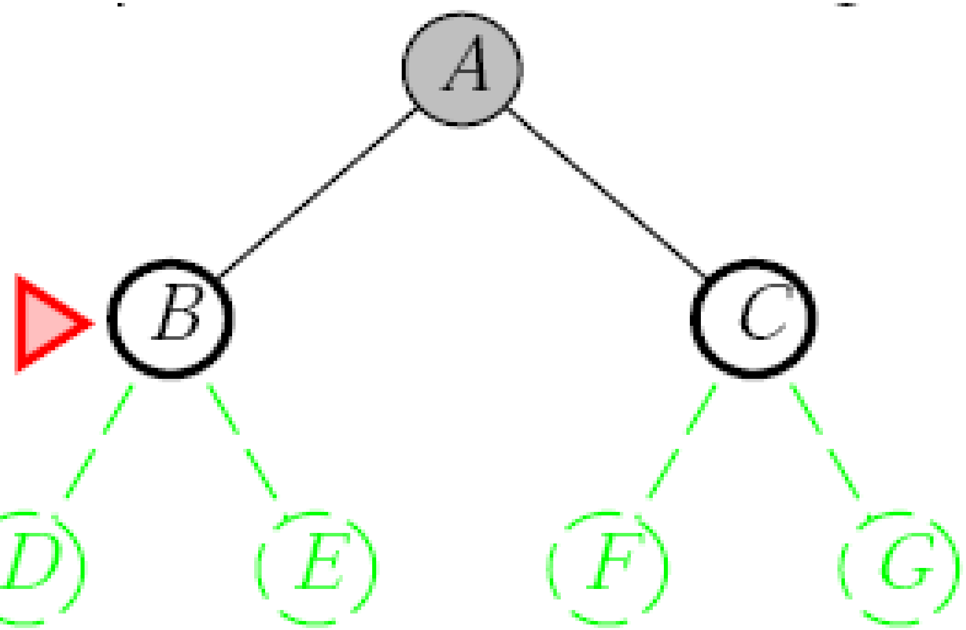
http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Breadth-first search

- Expand shallowest unexpanded node

- Implementation:

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



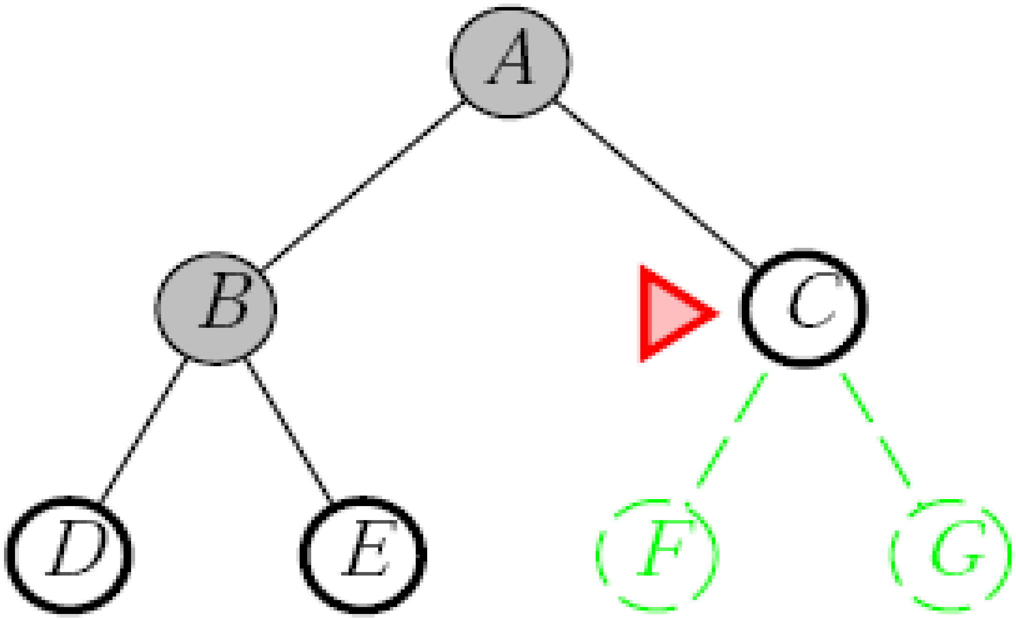
http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Breadth-first search

- Expand shallowest unexpanded node

- Implementation:

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



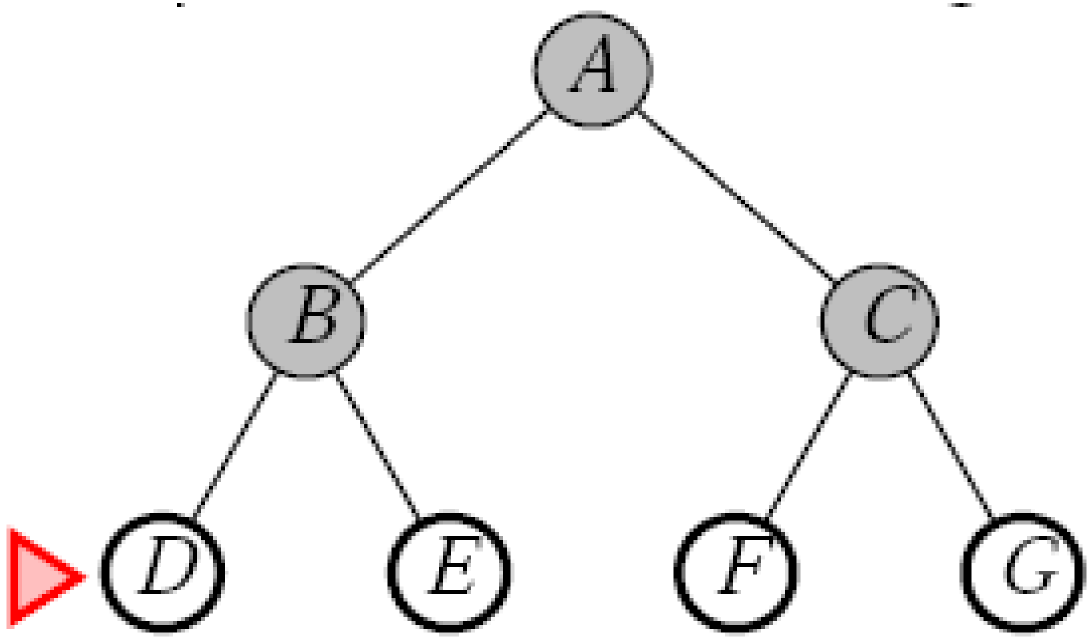
http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Breadth-first search

- Expand shallowest unexpanded node

- Implementation:

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



http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Properties of breadth-first search

- Complete??

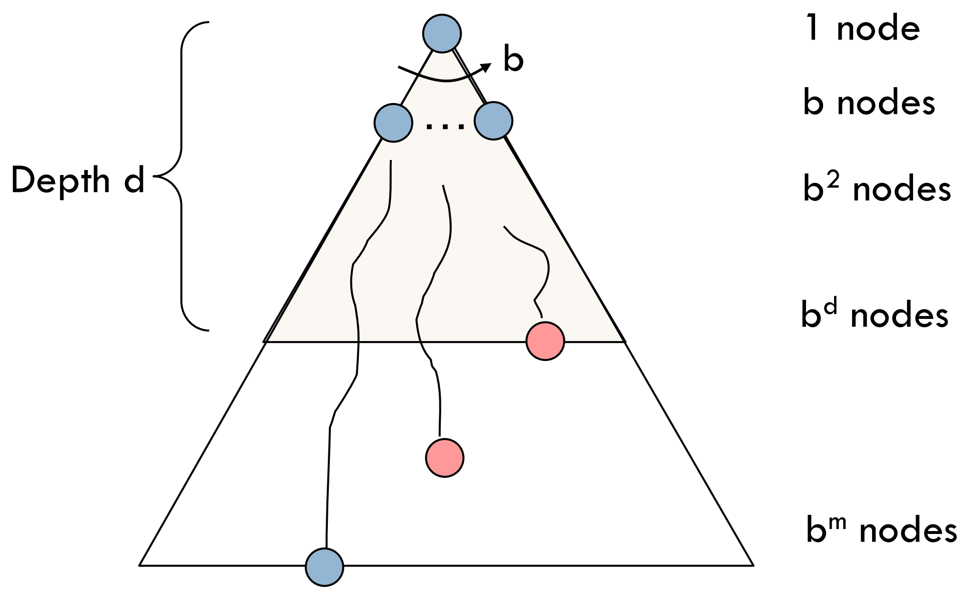
Properties of breadth-first search

- Complete?? Yes (if *d* is finite)
- Time??

b: Branching factor
d: Solution depth
m: Maximum depth

Properties of breadth-first search

- Complete?? Yes (if d is finite)
- Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d
- Space??

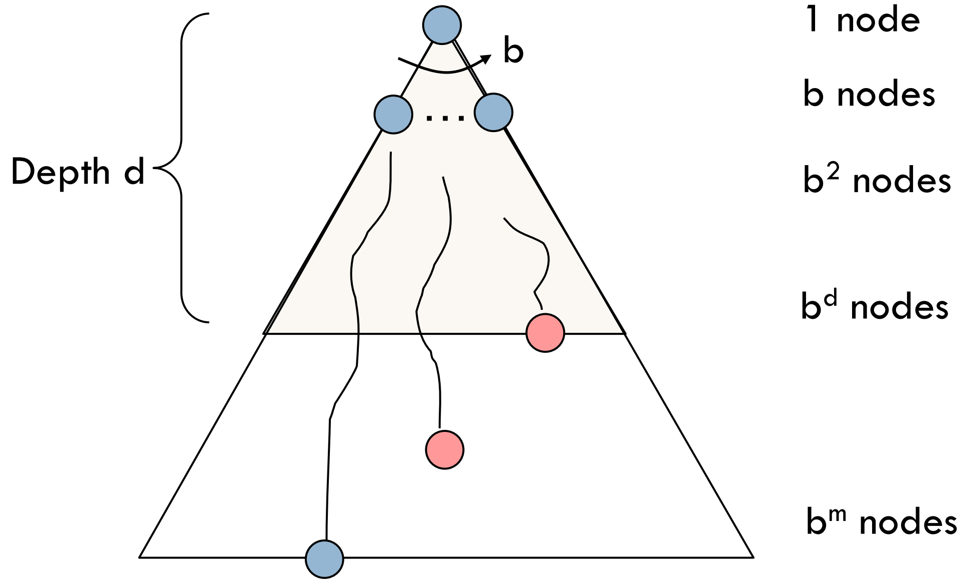


b: Branching factor
d: Solution depth
m: Maximum depth

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Properties of breadth-first search

- Complete?? Yes (if d is finite)
- Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d
- Space?? $O(b^{d+1})$ (keeps every node in memory)
- Optimal??



b: Branching factor
d: Solution depth
m: Maximum depth

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Properties of breadth-first search

- Complete?? Yes (if d is finite)
- Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, 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.

b: Branching factor
d: Solution depth
m: Maximum depth

Uninformed search strategies

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

- Breadth-first search
- Uniform-cost search（代价一致搜索）
- Depth-first search
- Depth-limited search
- Iterative deepening search

Uniform-cost search

- Expand least-cost unexpanded node

- Implementation:
  - *fringe* = queue ordered by path cost

- Equivalent to breadth-first if step costs all equal

Complete? Yes, if step cost ≥ ε

Time? # of nodes with $g \leq$ cost of optimal solution, $O(b^{ceiling(C^*/\varepsilon)})$ where C^* is the cost of the optimal solution

Space? # of nodes with $g \leq$ cost of optimal solution, $O(b^{ceiling(C^*/\varepsilon)})$

Optimal? Yes – nodes expanded in increasing order of $g(n)$

Uninformed search strategies

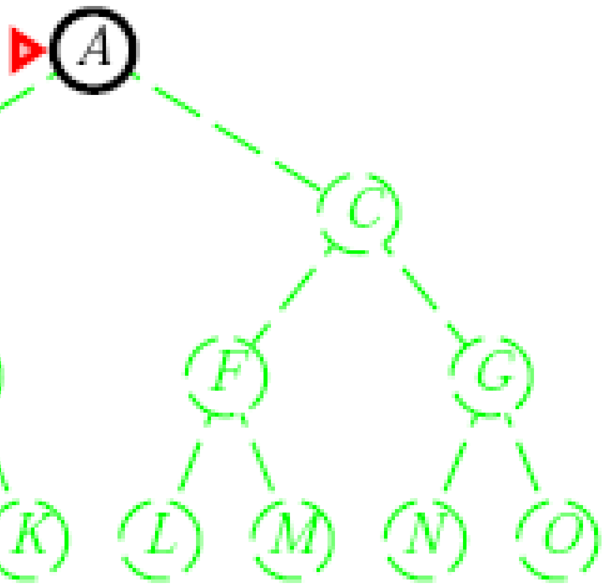
Uninformed search strategies use only the information available in the problem definition

- Breadth-first search
- Uniform-cost search
- Depth-first search（深度优先搜索）
- Depth-limited search
- Iterative deepening search

Depth-first search

- Expand deepest unexpanded node

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

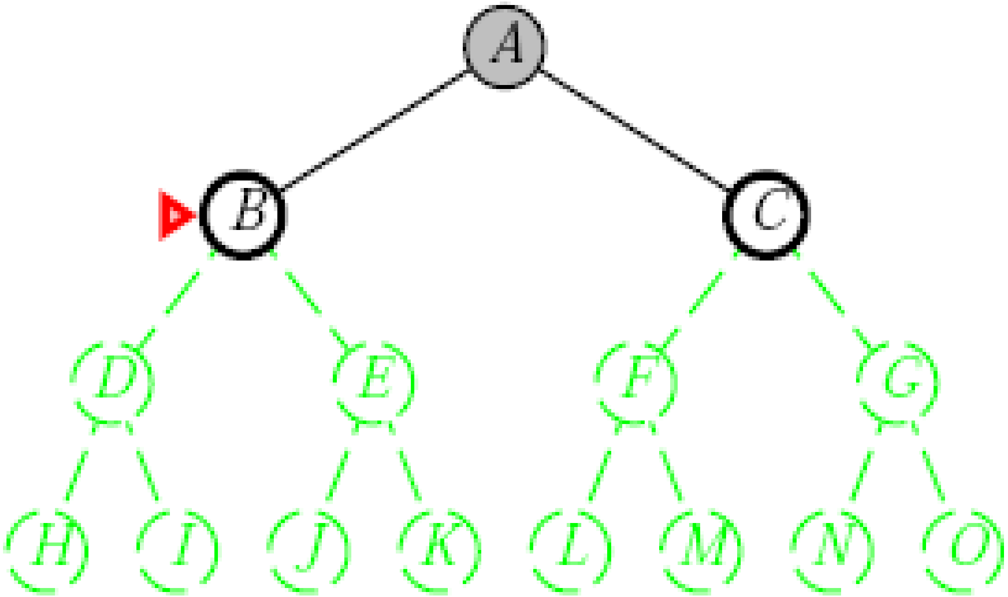


http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Depth-first search

- Expand deepest unexpanded node

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

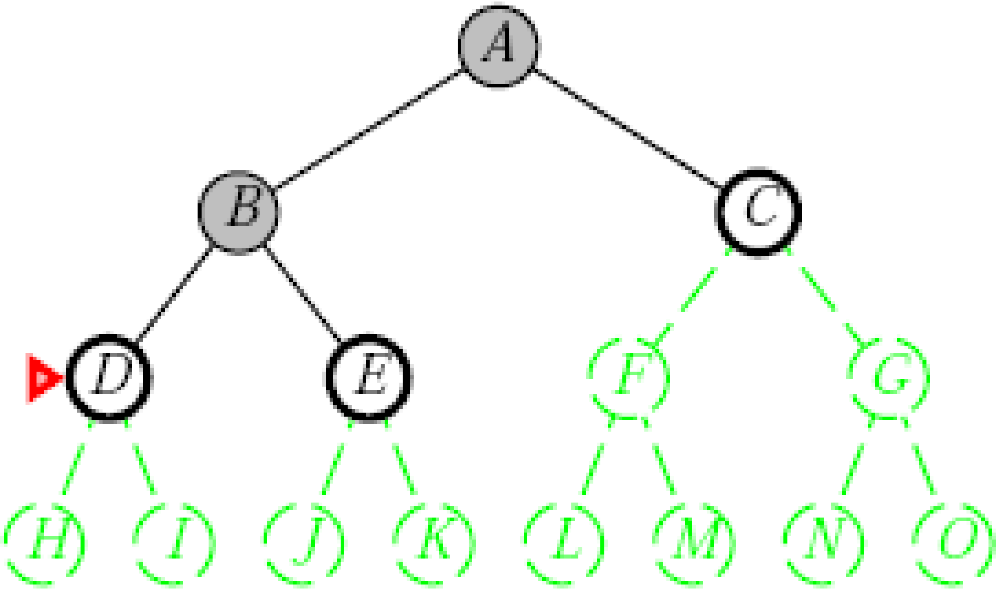


http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Depth-first search

- Expand deepest unexpanded node

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

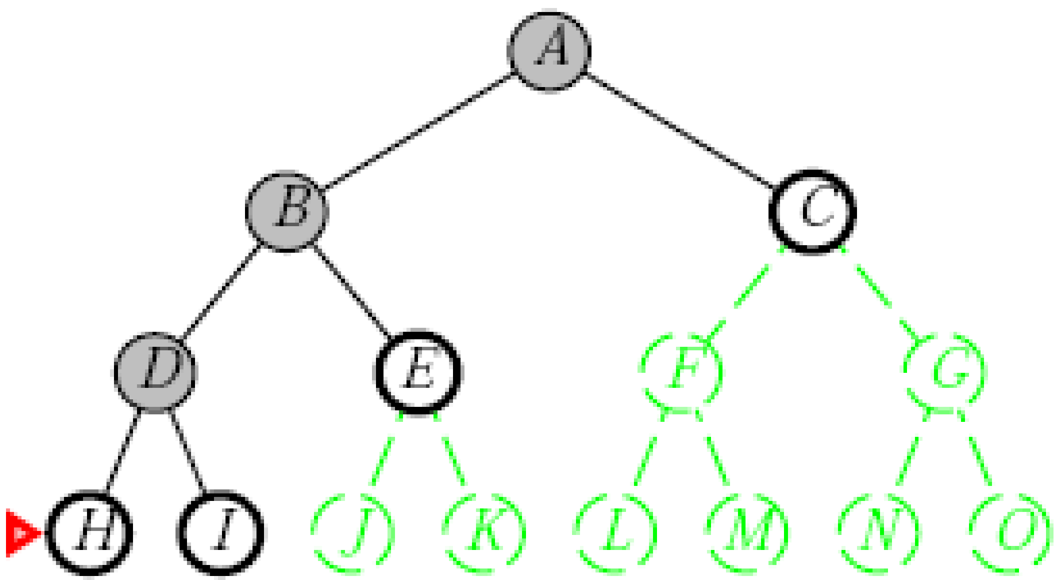


http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Depth-first search

- Expand deepest unexpanded node

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

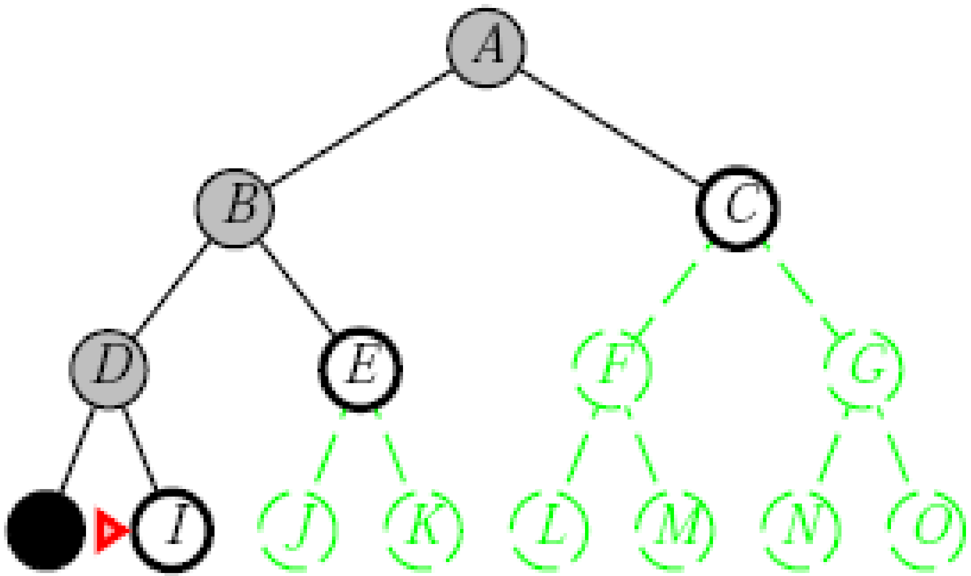


http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Depth-first search

- Expand deepest unexpanded node

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

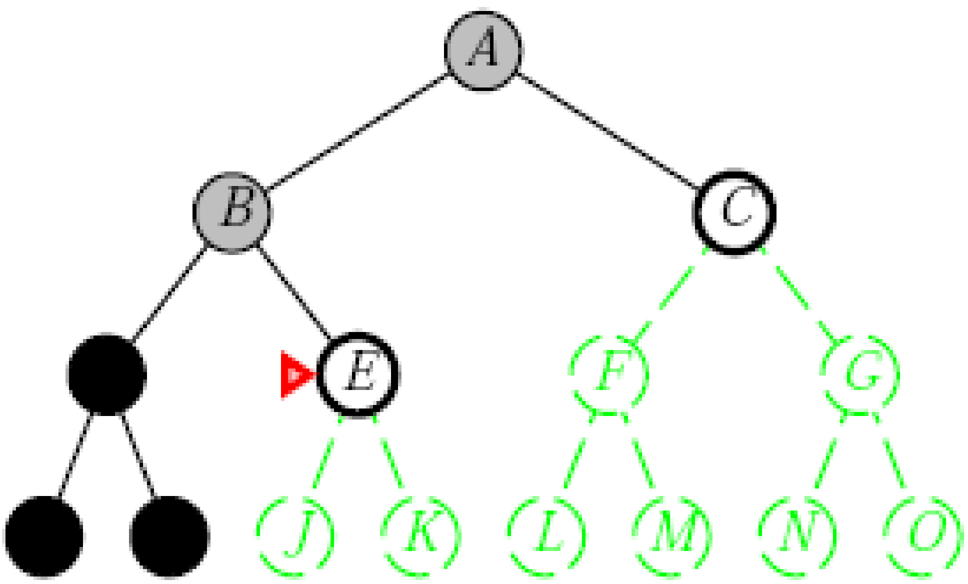


http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Depth-first search

- Expand deepest unexpanded node

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

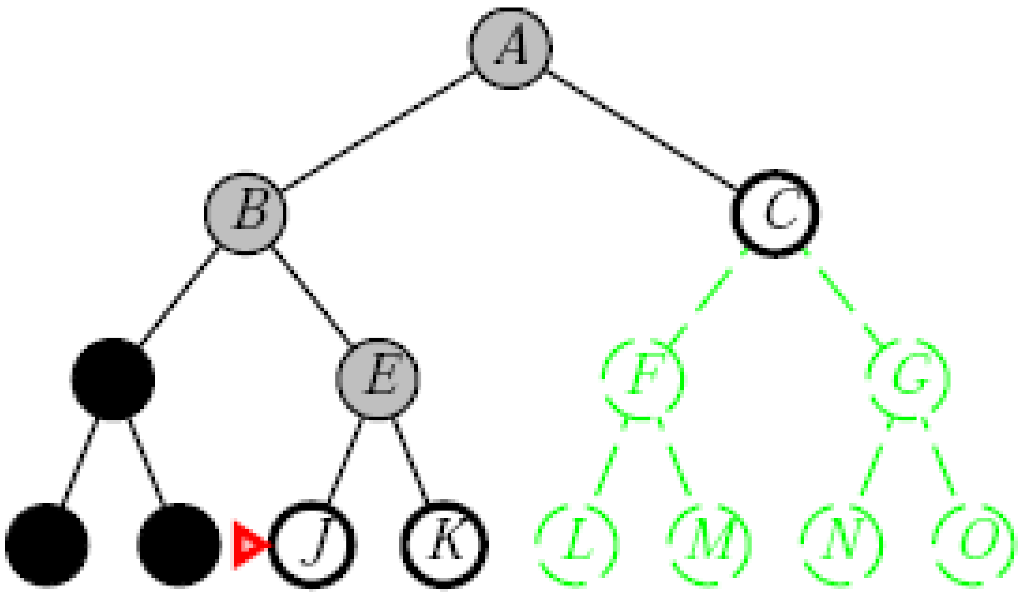


http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Depth-first search

- Expand deepest unexpanded node

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

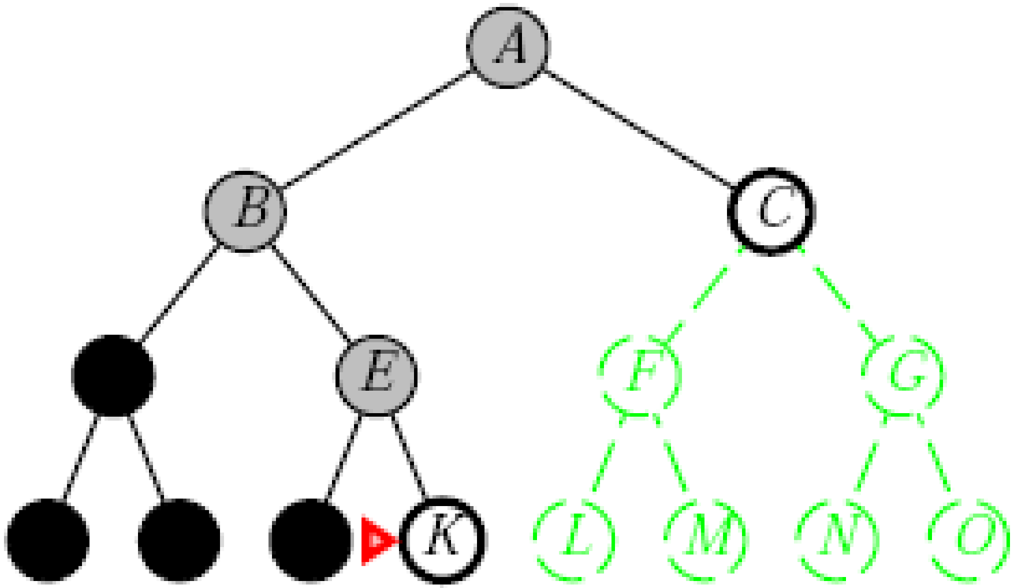


http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Depth-first search

- Expand deepest unexpanded node

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

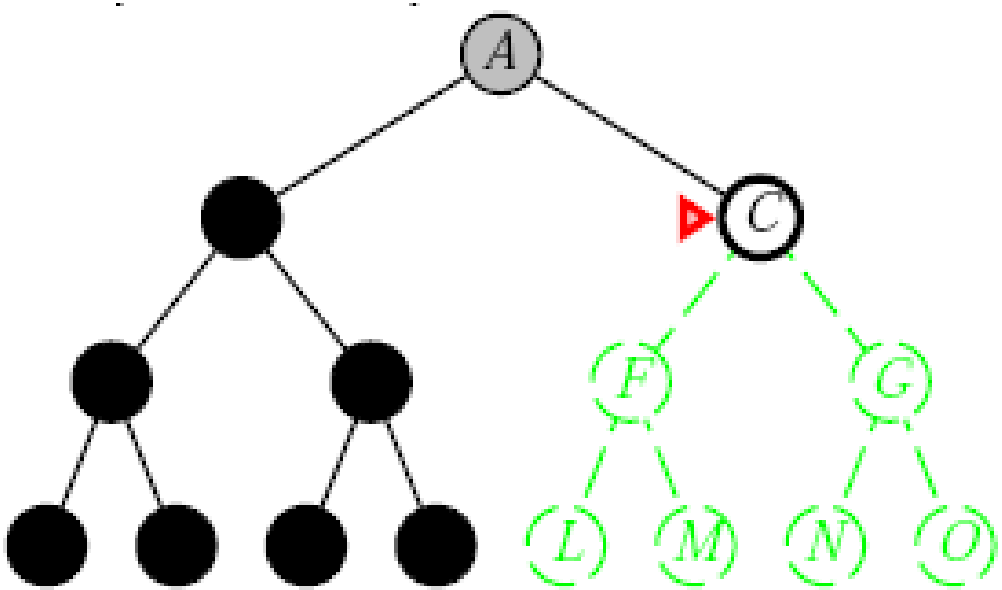


http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Depth-first search

- Expand deepest unexpanded node

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

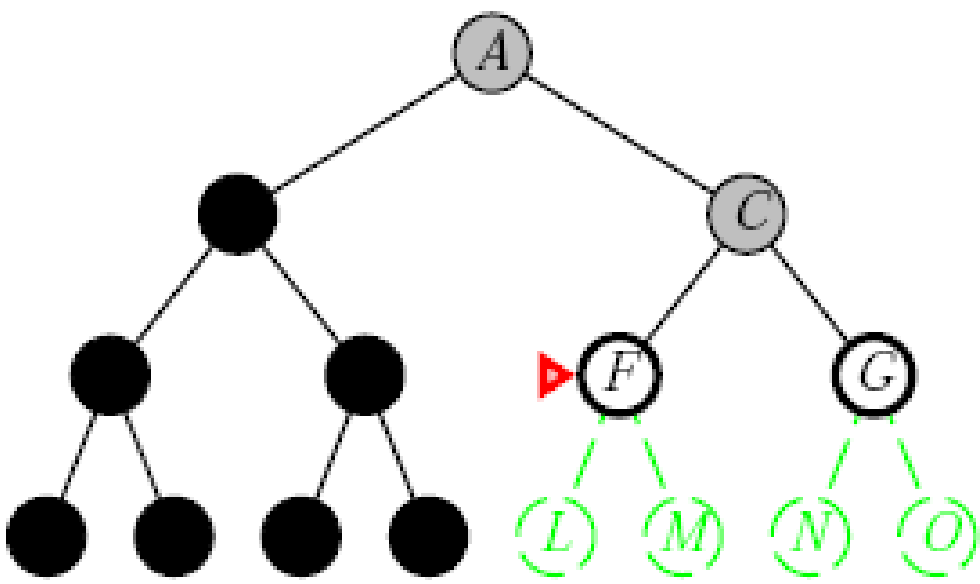


http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Depth-first search

- Expand deepest unexpanded node

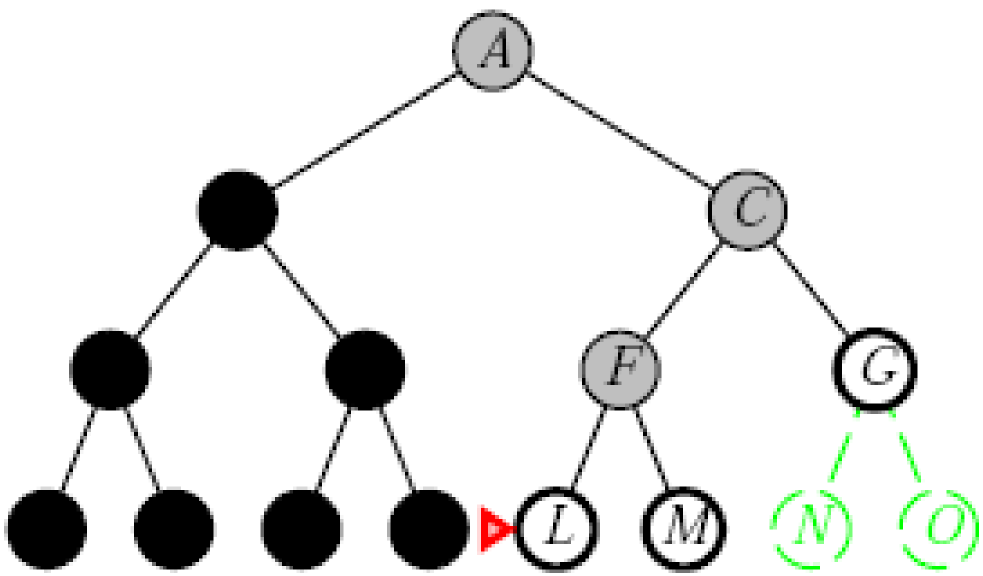
- Implementation:
- *fringe* = LIFO queue, i.e., put successors at front



http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Depth-first search

- Expand deepest unexpanded node
- Implementation:
- *fringe* = LIFO queue, i.e., put successors at front

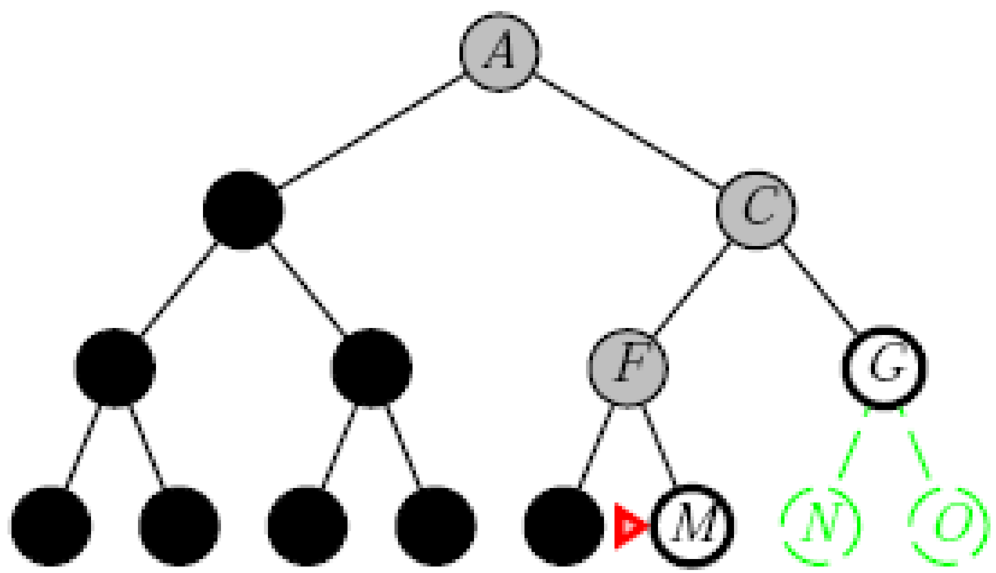


http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Depth-first search

- Expand deepest unexpanded node

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



http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Properties of depth-first search

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

  Modify to avoid repeated states along path

  $\Rightarrow$ complete in finite spaces

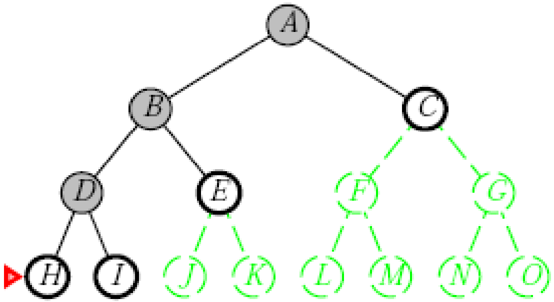
- Time?? $O(b^m)$: terrible if m is much larger than d

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

b: Branching factor
d: Solution depth
m: Maximum depth

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

- Optimal?? No



http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Uninformed search strategies

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

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search（深度有限搜索）
- Iterative deepening search

Depth-limited search

= depth-first search with depth limit *l*,

i.e., nodes at depth *l* have no successors

- Solves infinte-depth path problem
- if $l<d$, possibly incomplete
- If $l>d$, not optimal

- Recursive（递归） implementation

Uninformed search strategies

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

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search（迭代深入深度优先搜索）

Iterative deepening search

由Depth-limited search演化而成

每轮增加深度限制

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution

  inputs: *problem*, a problem

  for *depth* 0 **to** ∞ **do**

    result DEPTH-LIMITED-SEARCH(*problem*, *depth*)

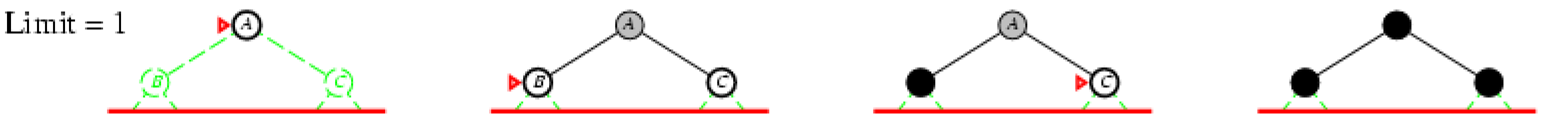
    if *result* ≠ cutoff **then return** *result*

  end

Iterative deepening search $l=0$

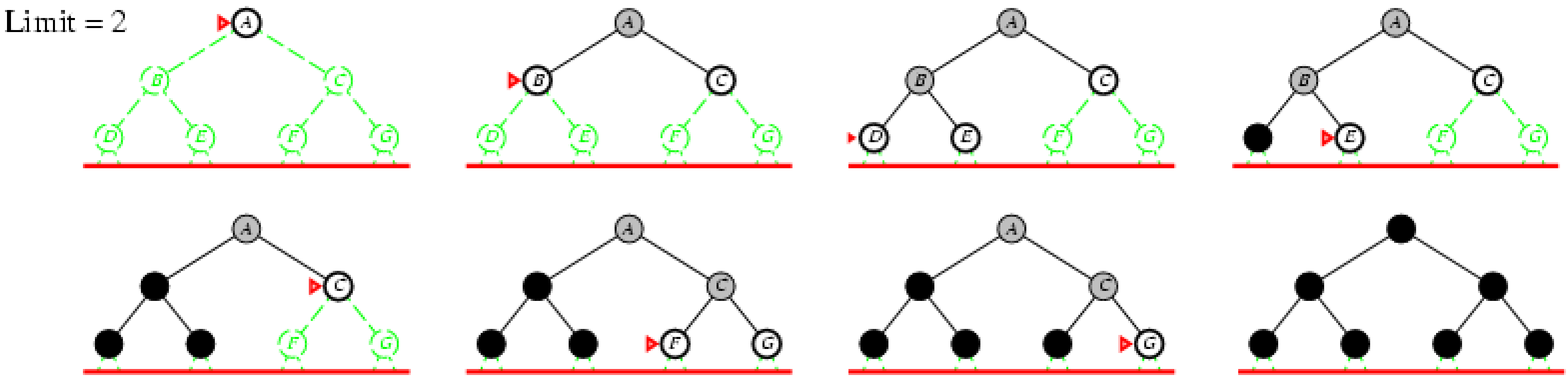
Limit = 0

Iterative deepening search *l* =1



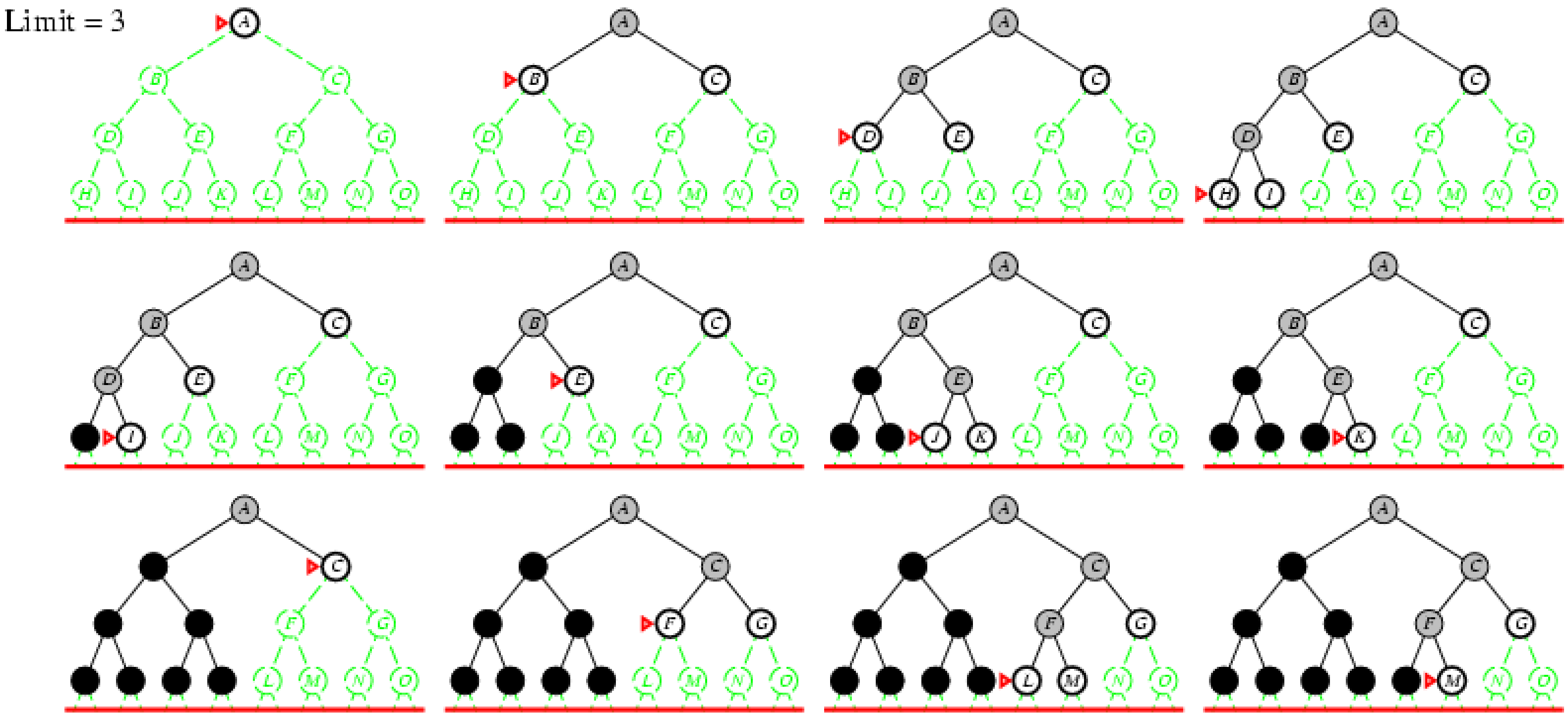
http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Iterative deepening search *l* =2



http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Iterative deepening search *l*=3



http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Properties of iterative deepening search

- Complete??

Properties of iterative deepening search

- Complete?? Yes
- Time??

Properties of iterative deepening search

- Complete?? Yes
- Time?? $(d + 1)b^0 + db^1 + (d - 1)b^2 + \ldots + b^d = O(b^d)$
- Space??

Properties of iterative deepening search

- Complete?? Yes
- Time?? $(d + 1)b^0 + db^1 + (d - 1)b^2 + \ldots + b^d = O(b^d)$
- Space?? $O(bd)$
- Optimal??

Properties of iterative deepening search

- Complete?? Yes
- Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space?? $O(bd)$
- Optimal?? Yes, if step cost = 1
- Can be modified to explore uniform-cost tree

Numerical comparison for $b = 10$ and $d = 5$, solution at far right leaf:

- $N(\text{IDS}) = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}456$
- $N(\text{BFS}) = 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 + 999{,}990 = 1{,}111{,}100$

- IDS does better because other nodes at depth d are not expanded

Summary of algorithms

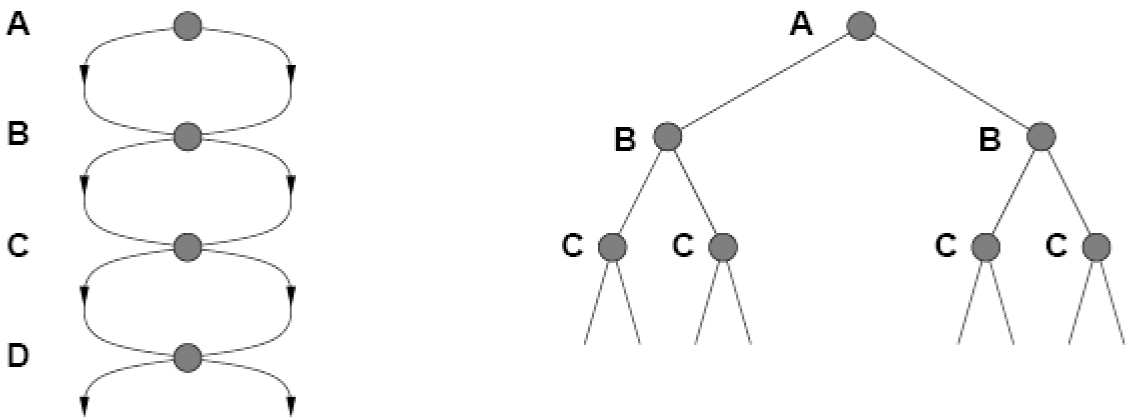
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

b: Branching factor
d: Solution depth
m: Maximum depth

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Repeated states

- Failure to detect repeated states can turn a linear problem into an exponential one!



Graph search

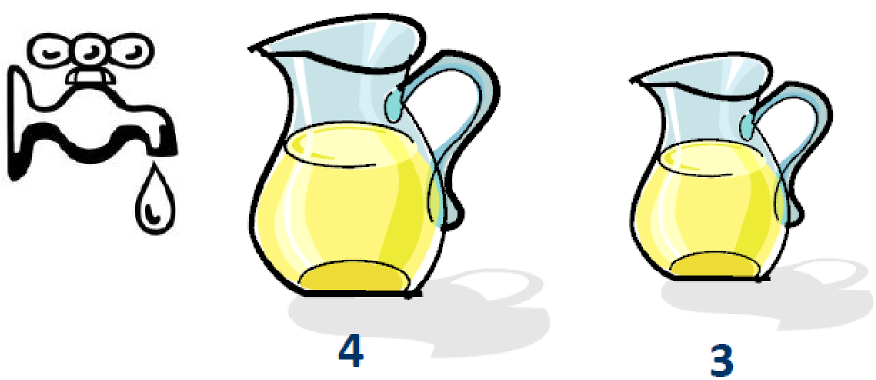

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]),fringe)
    loop do
   if fringe is empty then return failure
   node ← REMOVE-FRONT(fringe)
   if GOAL-TEST(problem , STATE[node]) then return node
   if STATE[node] is not in closed then
        add STATE[node] to closed
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Exercise

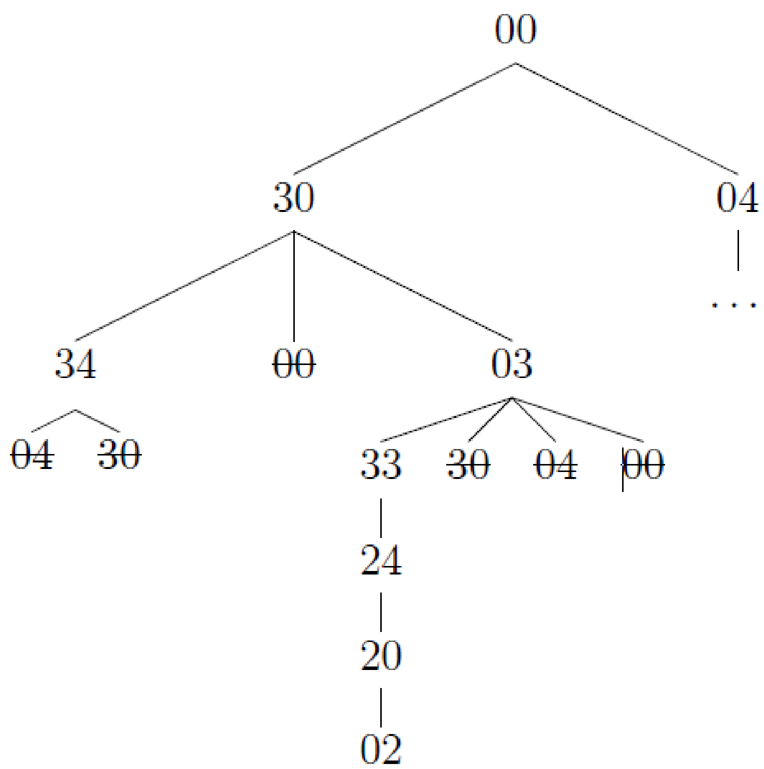
Water jugs problem

Given a 3 liter jug and a 4 liter jug, where jugs can be filled with water, emptied, or water can be poured from one jug into another until either the source jug is empty or the destination jug is full. Consider a problem where the jugs are initially empty and the goal is to achieve 2 gallons in the 4 gallon jug.



Exercise

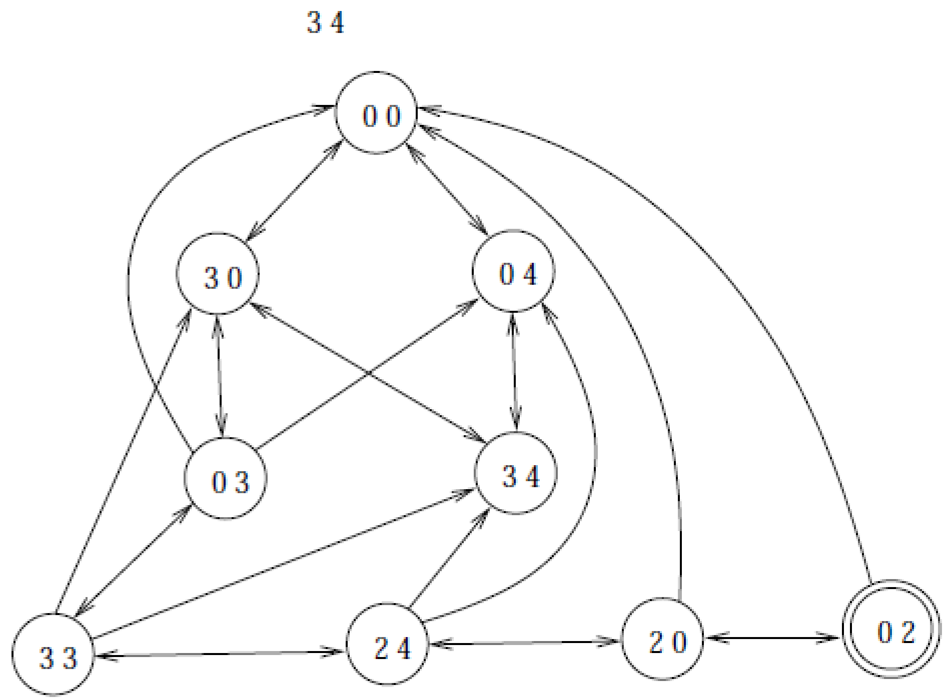
Solution: Systematically expand a search tree for the problem to find the solution as follows



http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Exercise

- In general, the *search graph* for this problem is given as follows



http://staff.ustc.edu.cn/~linlixu/ai2023spring/ai2023spring.html

Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored

- Variety of uninformed search strategies

- Iterative deepening search uses only linear space

  and not much more time than other uninformed algorithms

- Graph search can be exponentially more efficient than tree search

作业

- 3.7(a,b,d) （第二版）=3.6(a,b,d) （第三版）
- 3.9