Problems, Problem Spaces and Search

Dr. P. K. Nizar Banu

Associate Professor

Department of Computer Science

CHRIST (Deemed to be University)

Solving Problems with Search Algorithms

- **Input:** a problem *P*.
- **Preprocessing:**
  - Define *states* and a *state space*
  - Define *Operators*
  - Define a *start* state and *goal* set of states.
- **Processing:**
  - Activate a Search algorithm to find a *path* from start to one of the goal states.

Solving a Problem

* To build a system to do a particular task
  * Define the problem precisely
  * Analyse the problem
  * Isolate and represent the task knowledge
  * Choose the best problem-solving technique and apply

Formal Description of a Problem

- Define a **state space** that contains all possible configurations of the relevant objects
- Specify one or more states within that space that describes possible situations to start. These states are called the **initial state.**
- Specify one or more states that would be acceptable as solutions to the problem. These states are called **goal states**
- Specify a set of rules that describe the actions (operators) available.
  - ***Process of search is the fundamental to the problem solving process.***

Production Systems

* A set of rules, each consisting of a left side that determines the applicability of the rule and a right side that describes the operation to be performed
* One or more knowledge / databases that contain information of a particular task
* Control strategy
* A rule applier

Control Strategies

- Which rule to apply next during the process of searching for a solution to a problem???

- Causes Motion
- Systematic

Problem Characteristics

Analyze the problem

1. Is the problem decomposable into a set of independent smaller or easier sub-problems?
2. Can solution steps be ignored or at least undone if they prove unwise?
3. Is the problems universe predictable?
4. Is a good solution to the problem obvious without comparison to all other possible solutions

Contd…

5. Is the desired solution a state of the world or a path to a state?
6. Is a large amount of knowledge absolutely required to solve the problem or it is required only to constrain the search?
7. Can a computer simply return the solution or it requires interaction between the computer and a person?

Is the problem decomposable?

* The first step toward the design of a program to solve a problem must be the creation of a formal and manipulable description of the problem itself.
* This is called **operationalization.**

Can solutions steps be ignored or undone?

* Theorem proving, 8-puzzle, chess; illustrates the difference between

* Ignorable (Eg. Theorem Proving)
* Recoverable (Eg. 8-Puzzle)
* Irrecoverable (Eg. Chess)

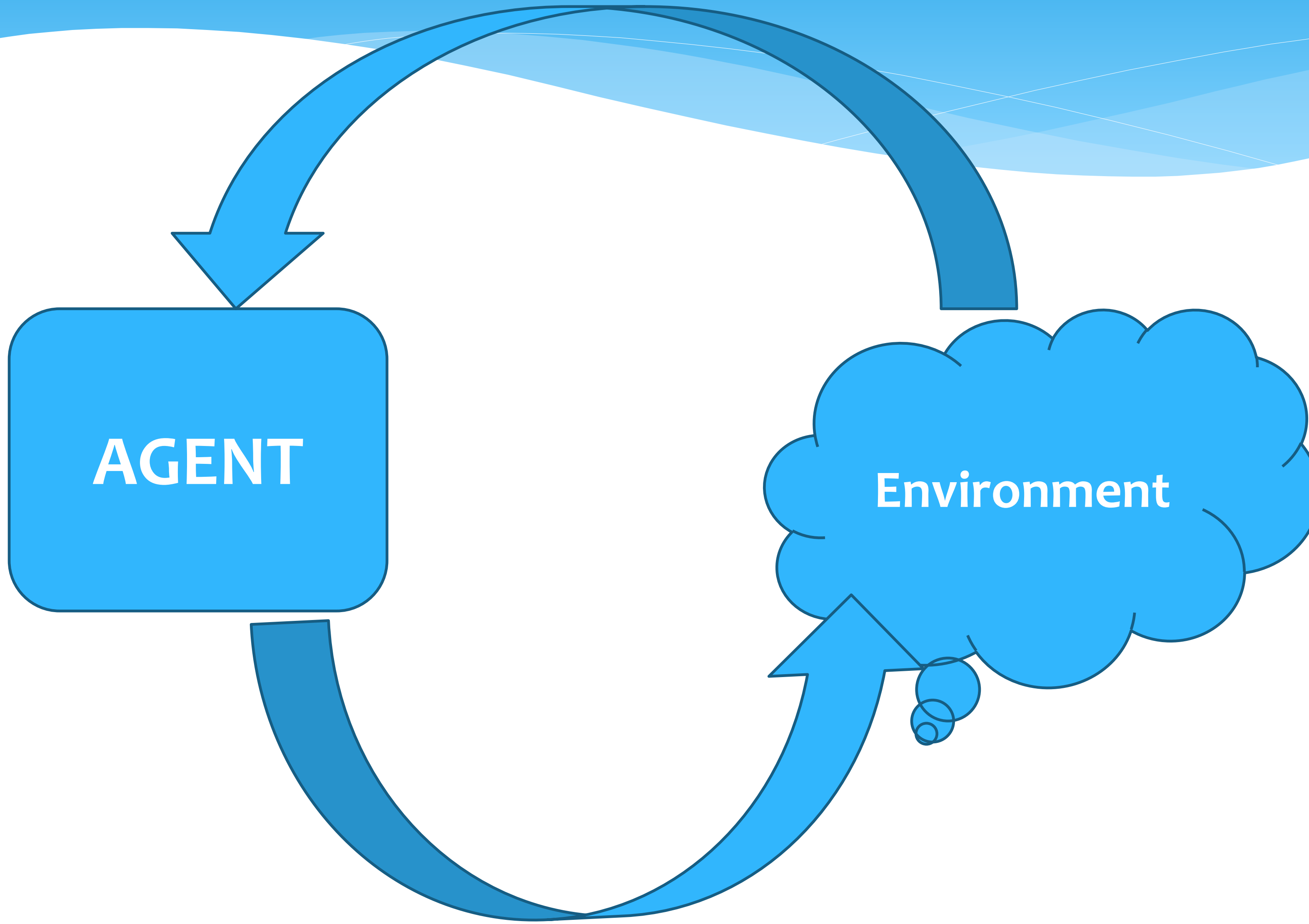
Contd…

* The **recoverability** of a problem determines the complexity of the control structure necessary for the problems solution
* **Ignorable** problems can be solved using a simple control structure that never backtracks
* **Irrecoverable** problems needs to be solved by a system that expends a great deal of effort

Is a good solution Absolute or Relative

- Travelling Sales man Problem
  - Shortest Route with minimum cost

Agent and environment



Agent

* Operates in an environment
* Perceives its environment through sensors
* Acts upon its environment through actuators / effectors
* Have goals
* Have sensors, actuators
* Implement mapping from percept sequence to actions
    * Operation which involves an actuator/effector is called **action**

Goal directed Agent

* A goal directed agent needs to achieve certain goals
* Many problems can be represented as a set of states and a set of rules of how one state is transformed to another
* The agent must choose a sequence of actions in order to achieve its goal.

Agent…

- Each state is an abstract representation of the agent's environment; denotes a configuration of the agent.
- **Initial State:** The initial state is a description of the starting configuration of the agent.
- An **action or an operator** takes an agent from one state to another state.
- By taking an action the agent moves from a current state to its successor state.
- A **plan** is a sequence of actions that the agent can take.

Contd…

* A **goal** is a description of a set of desirable states of the world.
* Path cost: path -> positive number
* Usually path cost=sum of step costs

Goal Based Agents

* Assumes the problem environment is:
  * **Static**
    * The plan remains the same
  * **Observable**
    * Agent knows the initial state
  * **Discrete**
    * Agent can enumerate the choices
  * **Deterministic**
    * Agent can plan a sequence of actions such that each will lead to an intermediate state

* The agent carries out its plans with its eyes closed
  * Certain of what's going on
  * Open loop system

Well Defined Problems and Solutions

* A problem
  * Initial state
  * Actions and Successor Function
  * Goal test
  * Path cost

Definitions

Problem formulation:

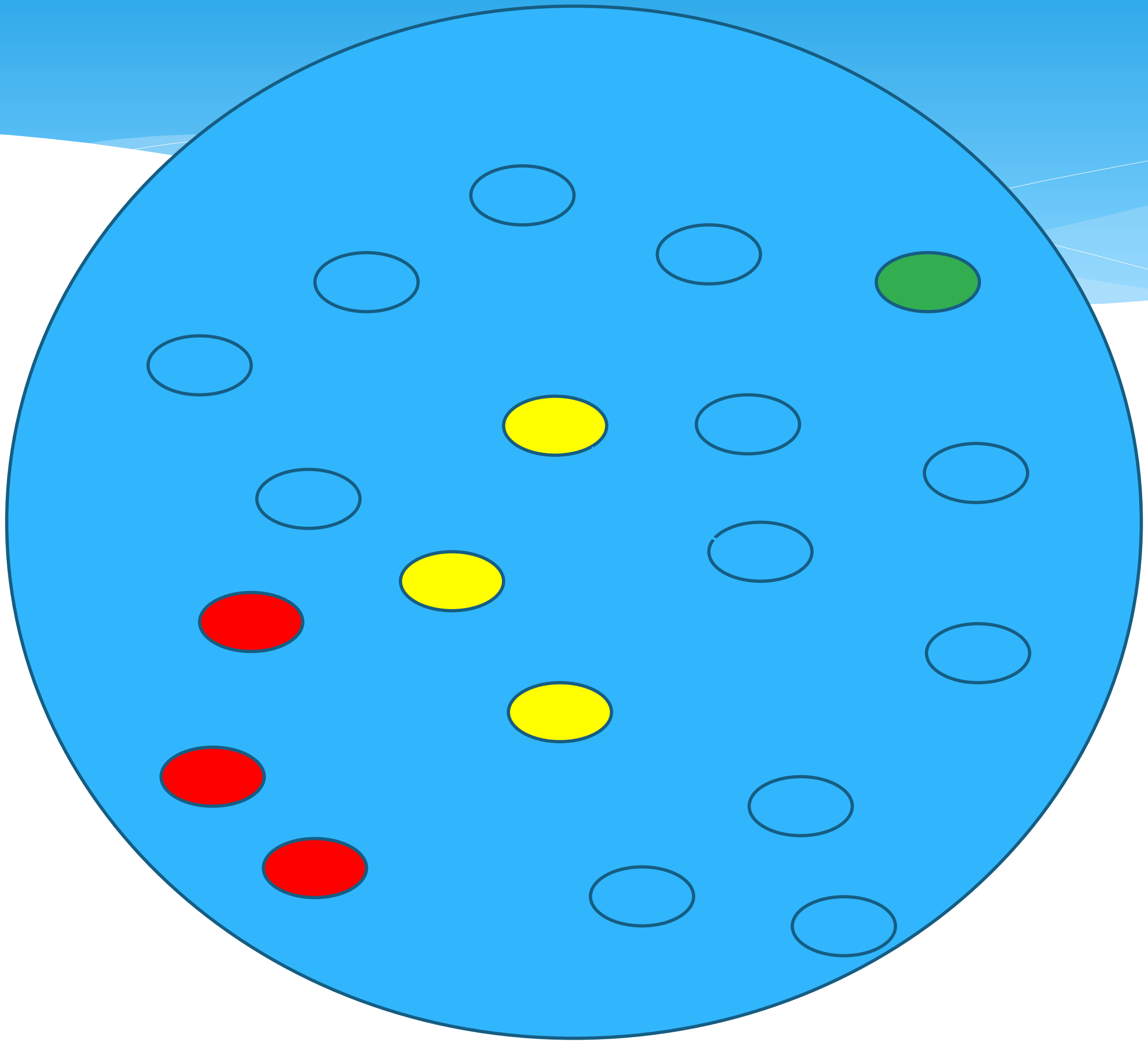
* Choosing a relevant set of states to consider and a feasible set of operators for moving from one state to another.

Search

* is the process of imagining sequences of operators applied to the initial state and checking with sequence reaches a goal state.

Search Problem

* **S**: the full set of states
* **So** : the initial state
* **A**: S->S set of operators
* **G**: the set of final states. G ϵS
* **Search problem:** finding a sequence of actions which transforms the agent from the initial state to a goal state.



Search Problem

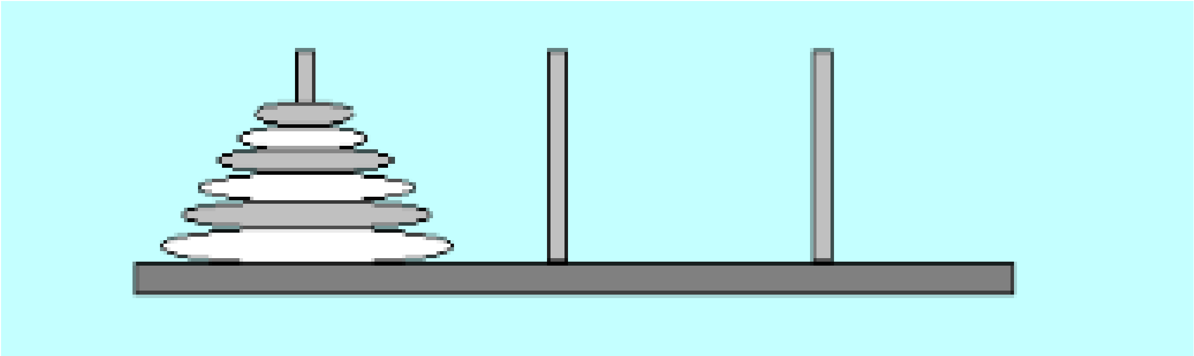
* A search problem consists of finding a sequence of actions which transforms the agent from the initial state to a goal state.
* Representing search problems
  * A search problem is represented using a **directed graph**
    * The states are represented as nodes
    * The allowed actions are represented as arcs

Searching Process

* Check the current state
* Execute allowable actions to move to the next state
* Check if the new state is a solution state
* If it is not, the new state becomes the current state and the process is repeated until a solution is found or the state space is exhausted.

The Tower of Hanoi

* The ancient puzzle of the Towers Of Hanoi consists of a number of wooden disks mounted on three poles, which are in turn attached to a baseboard. The disks each have different diameters and a hole in the middle large enough for the poles to pass through.
* The object of the puzzle is to move all the disks over to the right pole, one at a time, so that they end up in the original order on that pole.
* You can use the middle pole as a temporary resting place for disks, but at no time is a larger disk to be on top of a smaller one.



Tower of Hanoi

- We have three pegs and we have red, blue and green on peg a. The following operators are allowed. One may ***move the top most disks or any peg to the top most position of any other peg.*** These are the only allowable operations. And our objective is to reach this configuration where the red, blue and green or disks are in this same position on peg b and these are the allowable operators. We will see how we can find a solution to this problem

Tower of Hanoi

* The actions we will take here is
* move from a to c, that is
* move from a to b,
* move from a to c,
* move from b to a,
* move from c to b,
* move from a to b,
* then move from c to b and
* finally we have this desired goal configuration.

To play

https://www.mathsisfun.com/games/towerofhanoi.html

Example: Water Pouring

* Given a 4 gallon Jug and a 3 gallon Jug, how can we measure exactly 2 gallons into one jug?
  * There are no markings on the jug
  * You must fill each jug completely

Problem Representation

- State Representation and Initial State:
- Tuple(x,y);
  - X represents 4-gallon jug
  - Y represents 3 gallon jug
  - 0 <= x <=4 and 0<=y<=3
  - Initial State: (0,0)
  - Goal Sate_ (2,y) where 0<=y<=3

Operators

1	Fill 4-gallon Jug	(x,y) X < 4	(4,y)
2	Fill 3-gallon jug	(x,y) Y<3	(x,3)
3	Empty 4-gallon jug	(x,y) X > 0	(0,y)
4	Empty 3-gallon jug	(x,y) Y > 0	(x,0)
5	Pour water from 3-gal jug to fill 4-gal jug	(x,y) 0 < x+y >+ 4 and y > 0	(4, y-(4-x))
6	Pour water from 4-gal jug to fill 3-gal jug	(x,y) 0 < x+y >= 3 and x>0	(x – (3-y), 3)
7	Pour all water from 3-gal jug to 4-gal jug	(x,y) 0 < x+y <=4 and y >= 0	(x+y, 0)
8	Pour all water from 4-gal jug to 3-gal jug	(X,Y) 0 < x+y <=3 and x >= 0	(0, x+y)

A state space search

Gals in 4-gallon jug	Gals in 3-gallon jug	Rule Applied
0	0	
4	0	1. Fill 4
1	3	6. Pour 4 into 3 to fill
1	0	4. Empty 3
0	1	8. Pour all of 4 into 3
4	1	1. Fill 4
2	3	6. Pour into 3

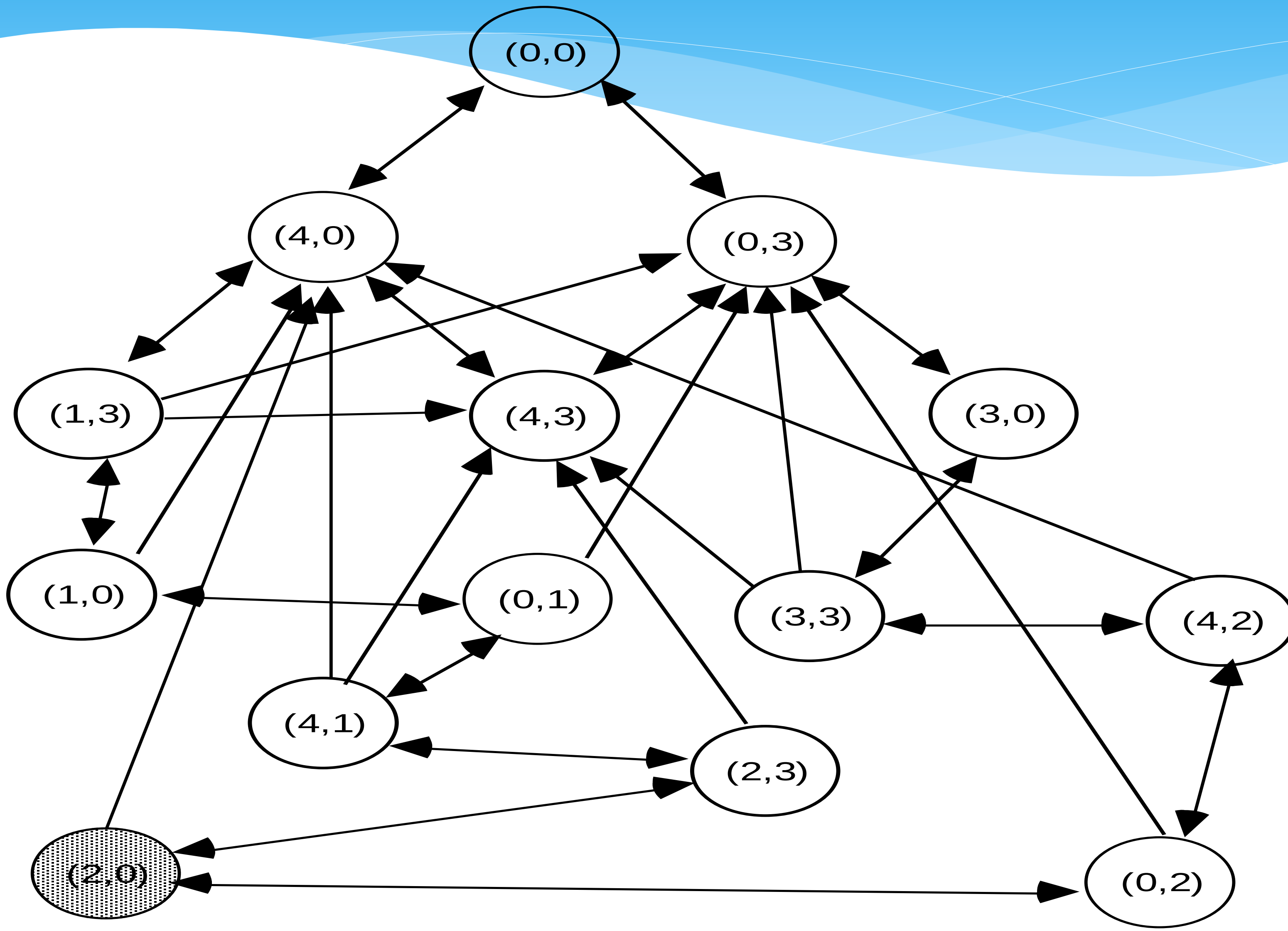
To Play

* https://www.mathsisfun.com/games/jugs-puzzle.html

Example: Water Pouring

* Initial state:
  * The Jugs are empty
  * Represented by the tuple (0 0)
* Goal state:
  * One of the Jug has two gallons of water in it
  * Represented by either (x 2) or (2 x)
* Path cost:
  * 1 per unit step

Example: Water Pouring



A state space search

* (x,y): order pair
* X: water in 4-gallons x=0,1,2,3,4
* Y:water in 3-gallons y=0,1,2,3
* Start state: (0,0)
* Goal state(2,n) where n=any value
* Ruels:1. Fill the 4 gallon-jug (4,-)
* 2. Fill the 3 gallon-jug (0,3)
* Empty the 4 gallon-jug (0,-)
* Empty the 3 gallon-jug (-,0)

Contd…

- Pour contents of one jug into another
  - (x y) -> (0 x+y) or (x+y-4, 4)
  - (x y) -> (x+y 0) or (3, x+y-3)

Example: Eight Puzzle

- **States:**
  - Description of the eight tiles and location of the blank tile
- **Successor Function:**
  - Generates the legal states from trying the four actions {*Left*, *Right*, *Up*, *Down*}
- **Goal Test:**
  - Checks whether the state matches the goal configuration
- **Path Cost:**
  - Each step costs 1

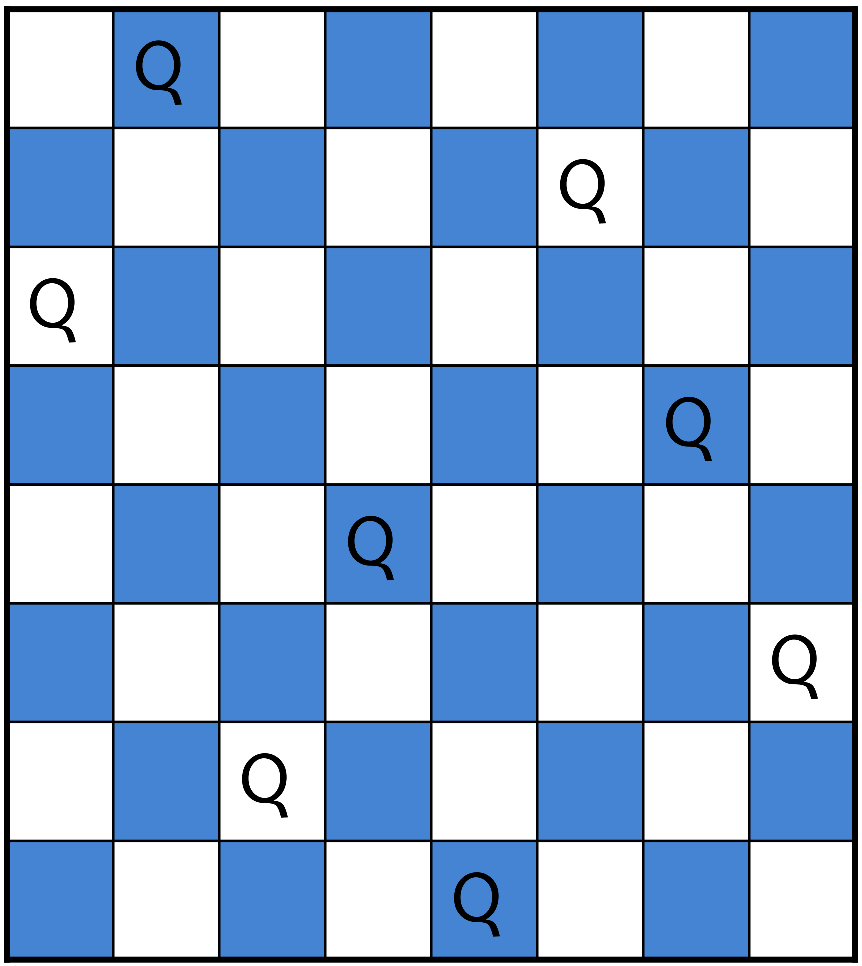
2	8	3
1	6	4
7	5	

1	2	3
8		4
7	6	5

To Play

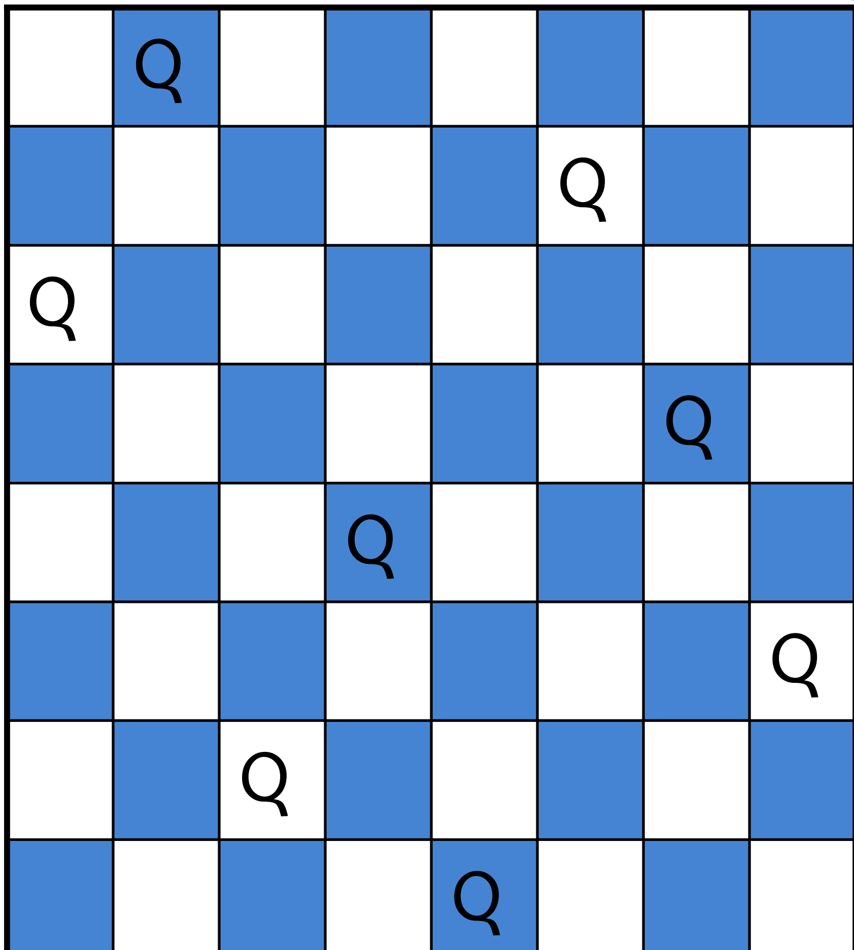
* https://sliding.toys/mystic-square/8-puzzle/

Example: Eight Queens



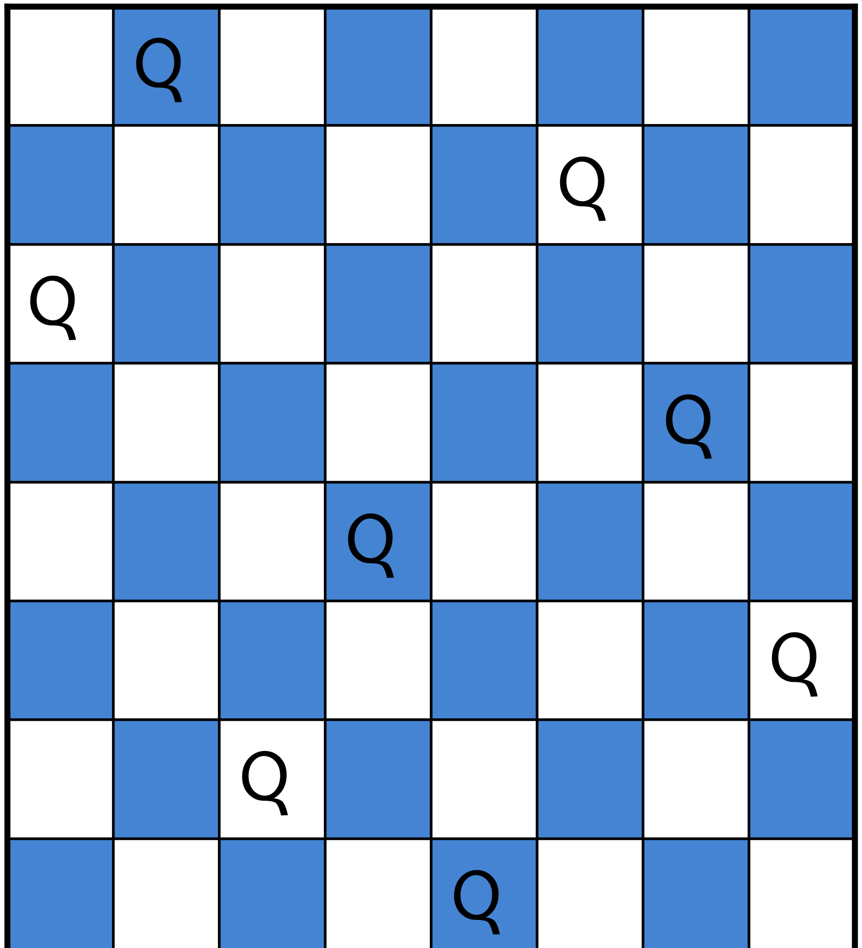
- Place eight queens on a chess board such that no queen can attack another queen
- No path cost because only the final state counts!
- Incremental formulations
- Complete state formulations

Example: Eight Queens



- **States:**
  - Any arrangement of 0 to 8 queens on the board
- **Initial state:**
  - No queens on the board
- **Successor function:**
  - Add a queen to an empty square
- **Goal Test:**
  - 8 queens on the board and none are attacked

Example: Eight Queens

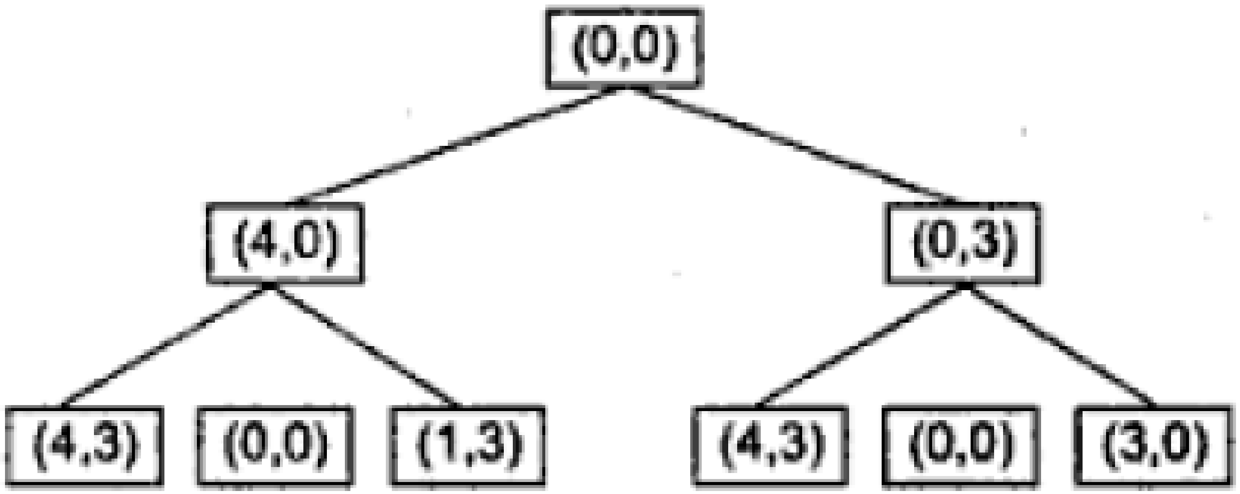


* **States:**
  * Arrangements of n queens, one per column in the leftmost n columns, with no queen attacking another are states
* **Successor function:**
  * Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen.

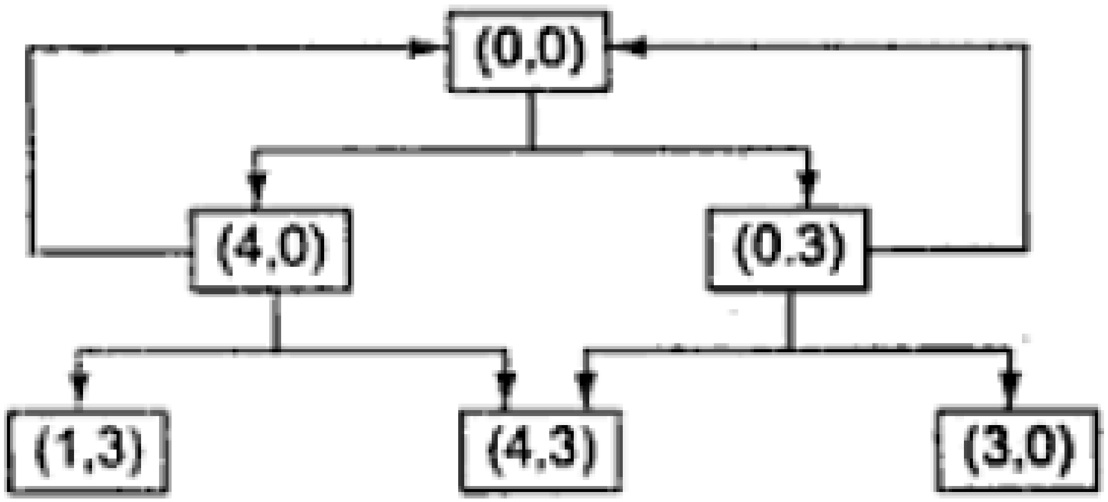
To Play

- https://www.brainbashers.com/queens.asp

Search Tree for the Water-jug problem



Search Graph for the Water Jug Problem



Missionaries and Cannibals Problem

* Three missionaries and three cannibals find themselves on one side of a river.
* They have agreed that they would all like to get to the other side.
* But the missionaries are not sure what else the cannibals have agreed to.
* So the missionaries want to manage the trip across the river in such a way that the number of missionaries on either side of the river is never less than the number of cannibals who are on the same side.
* The only boat available holds only two people at a time.
* How everyone can get across the river without the missionaries risking being eaten?

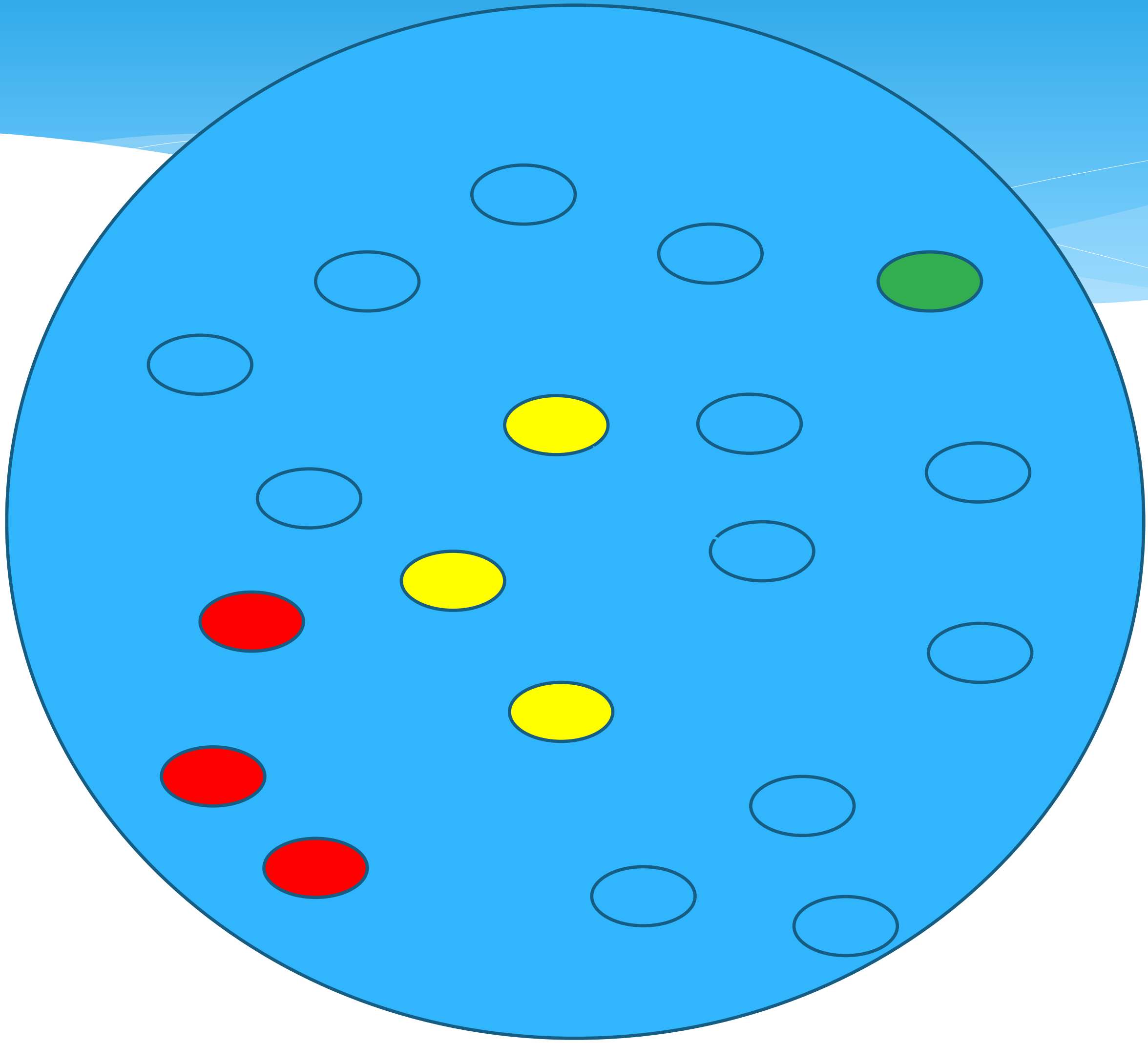
To Play

* **1. Missionaries and Cannibals Problem**
* **2. Monkey and Banana Problem**

Water Jug Problem

* You have three jugs measuring 12 gallons, 8 gallons and 3 gallons and a water pump. You need to measure out exactly one gallon.

* Initial State: All three jugs are empty
* Goal test: Some jug contains exactly one gallon.



Basic Search Algorithm

Let L be a list containing the initial state (L = the fringe)
Loop
    If L is empty return failure
    Node ← select(L)
    If Node is a goal
    Then return Node (the path from initial state to Node)
    Else apply all applicable operators to Node
    And merge the newly generated states into L
End Loop

Basic Search Algorithm: Search Issues

* **Search tree may be unbounded**
  * Because of loops
  * Because state space is infinite
* **Return a path or a node?**
* **How are merge and select done?**
  * Is the graph weighted or unweighted?
  * How much is known about the quality of intermediate states?
  * Is the aim to find a minimal cost path or any path as soon as possible?

Search Tree

* List all possible paths
* Eliminate cycles from paths
* Result: A Search Tree

Search Tree Terminology

- Root Node
- Leaf Node
- Ancestor / Descendant
- Branching factor
- Complete path / Partial Path
- Expanding open nodes results in closed nodes

Uninformed search strategies

Uninformed:

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

While searching you have no clue whether one non-goal state is better than any other. Your search is blind.

Contd…

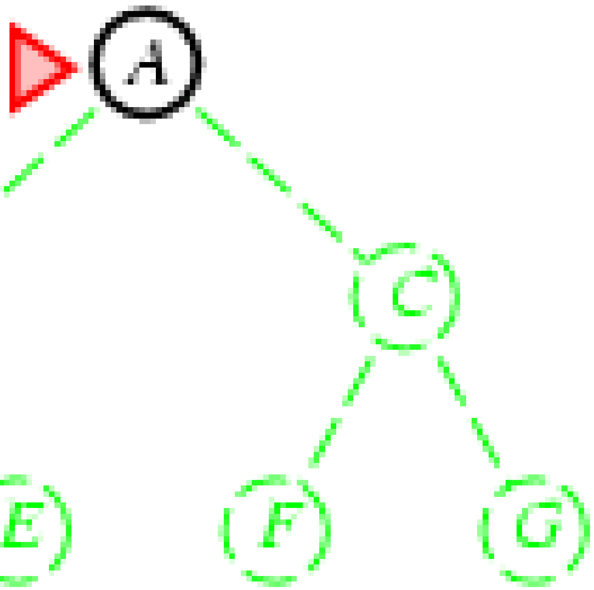
Various blind strategies:

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Iterative deepening search

Breadth-first search

* Expand shallowest unexpanded node
* Implementation:
  * *fringe* is a first-in-first-out (FIFO) queue, i.e., new successors go at end of the queue.

Is A a goal state?

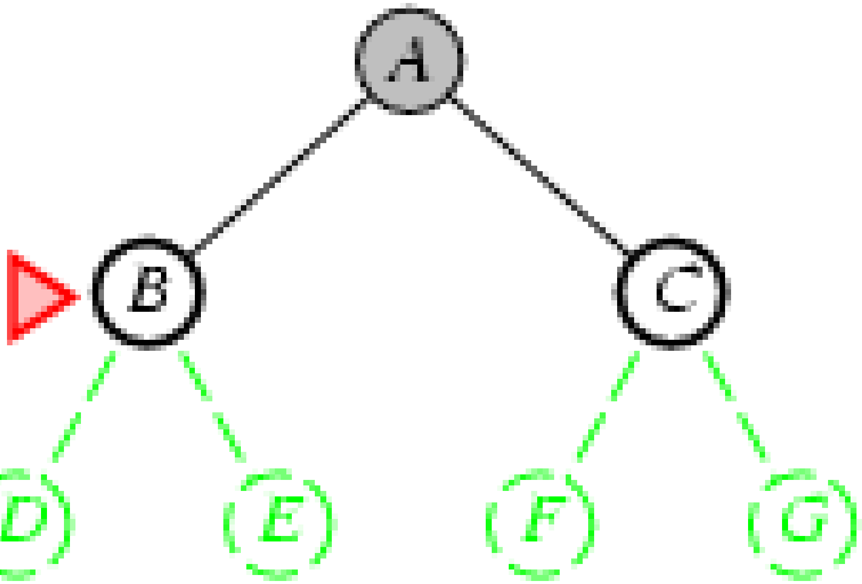


Contd..

* Expand shallowest unexpanded node
* Implementation:
  * *fringe* is a FIFO queue, i.e., new successors go at end

Expand:
fringe = [B,C]

Is B a goal state?

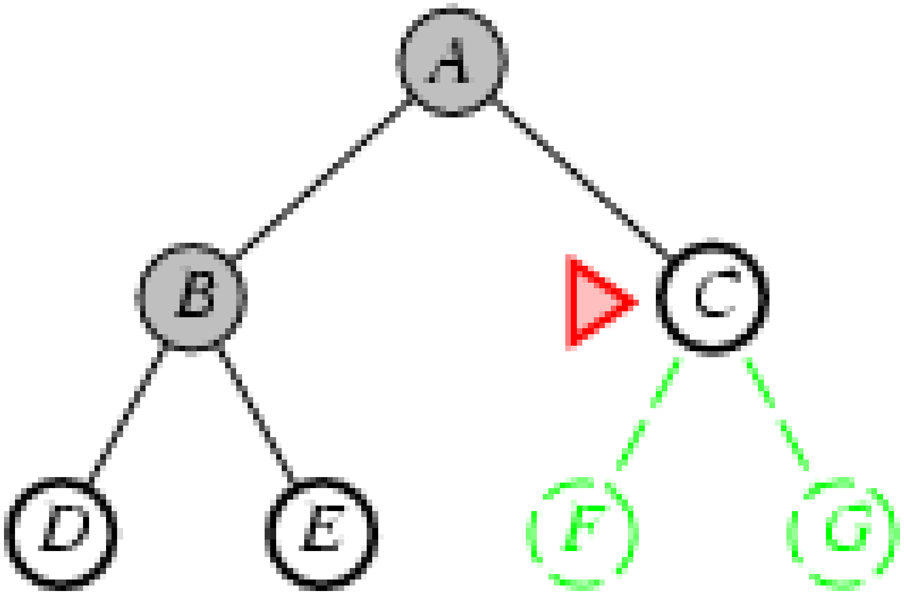


Contd…

* Expand shallowest unexpanded node
*
* Implementation:
  * *fringe* is a FIFO queue, i.e., new successors go at end

Expand:
fringe=[C,D,E]

Is C a goal state?

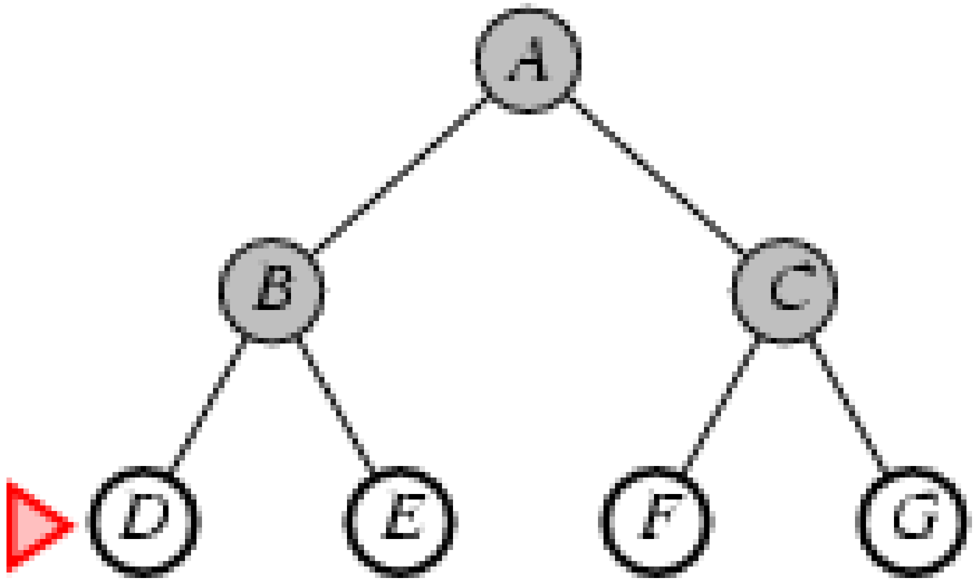


Contd..

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end

Expand:
fringe=[D,E,F,G]

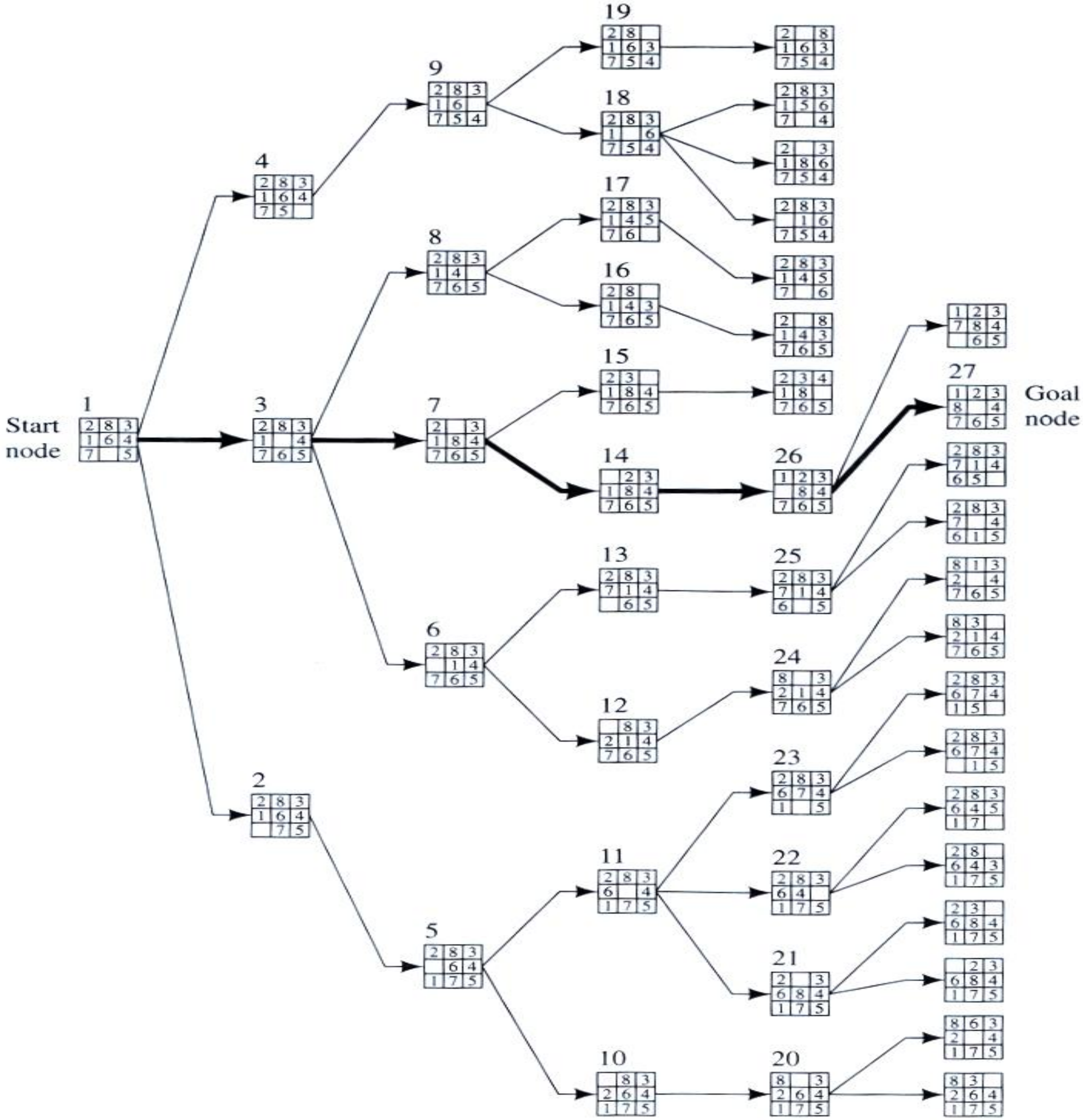
Is D a goal state?



Comparing Uninformed Search Strategies

- **Completeness**
  - Will a solution always be found if one exists?
- **Time**
  - How long does it take to find the solution?
  - Often represented as the number of nodes searched
- **Space**
  - How much memory is needed to perform the search?
  - Often represented as the maximum number of nodes stored at once
- **Optimal**
  - Will the optimal (least cost) solution be found?

Example BFS



Start node
Goal node
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

Breadth First Search

Let fringe be a list containing the initial state

Loop

    If fringe is empty return failure

    Node ← remove-first (fringe)

    If Node is a goal

    Then return the path from initial state to Node

    Else generate all successors of Node.

    And merge the newly generated states into fringe

End Loop

Properties of breadth-first search

Complete? Yes it always reaches goal (if *b* is finite)

Time? $1+b+b^2+b^3+\ldots +b^d + (b^{d+1}-b)) = O(b^{d+1})$

(this is the number of nodes we generate)

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

either in fringe or on a path to fringe).

Optimal? Yes (if we guarantee that deeper solutions are less optimal, e.g. step-cost=1).

Space is the bigger problem (more than time)

Uniform-Cost Search

* Same idea as the algorithm for breadth-first search,
  * Expand the ***least-cost*** unexpanded node
  * FIFO queue is ordered by cost
  * Equivalent to regular breadth-first search if all step costs are equal

Contd…

Breadth-first is only optimal if step costs is increasing with depth (e.g. constant).

Uniform-cost Search:

Expand node with smallest path cost g(n).

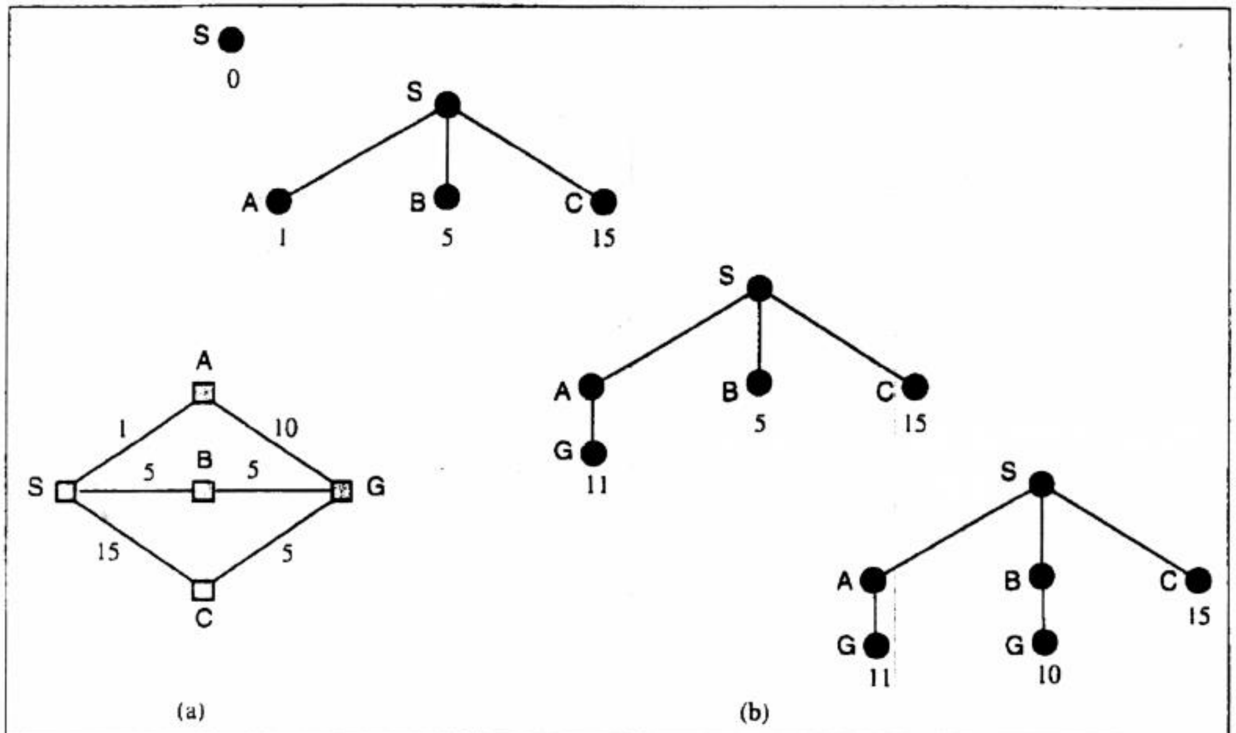


Figure 3.13 A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

Uniform-cost search

Implementation: *fringe* = queue ordered by path cost
Equivalent to breadth-first if all step costs all equal.

Complete? Yes, if step cost ≥ ε
(otherwise it can get stuck in infinite loops)

Time? # of nodes with *path cost* ≤ cost of optimal solution.

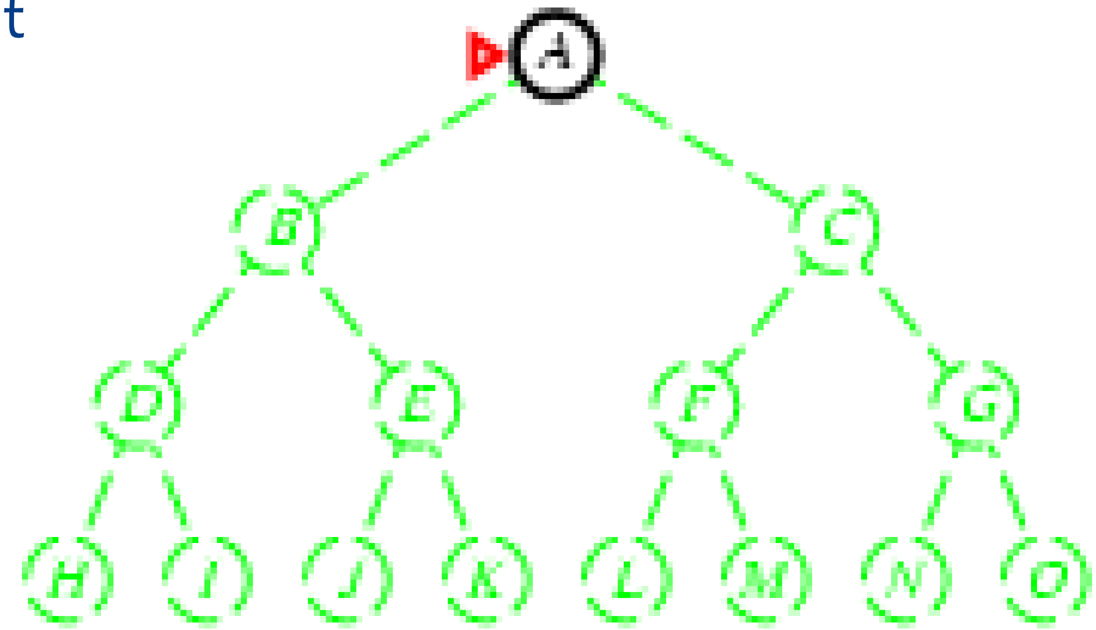
Space? # of nodes on paths with path cost ≤ cost of optimal solution.

Optimal? Yes, for any step cost.

Depth-first search

* Expand *deepest* unexpanded node
* Implementation:
  * *fringe* = Last In First Out (LIFO) queue, i.e., put successors at front

Is A a goal state?

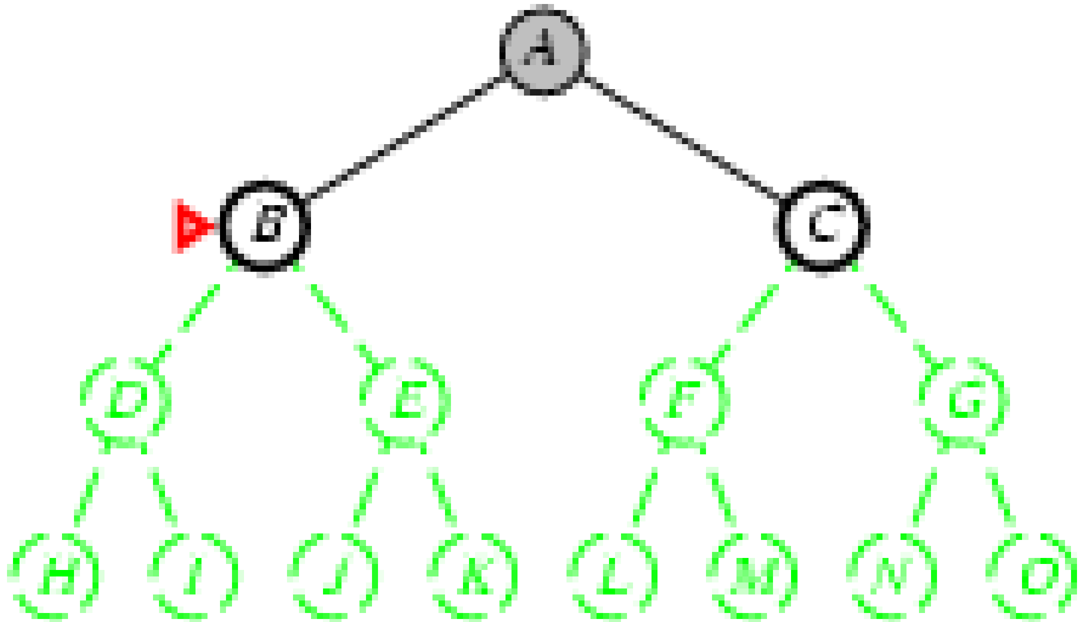


Depth-first search

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

queue=[B,C]

Is B a goal state?

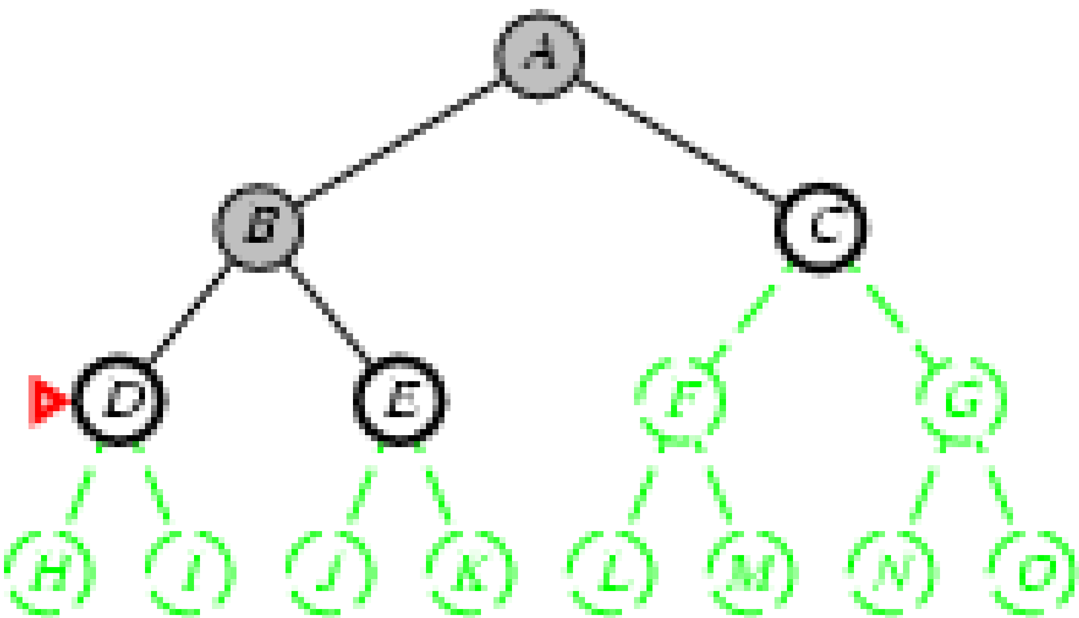


Depth-first search

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

queue=[D,E,C]

Is D = goal state?

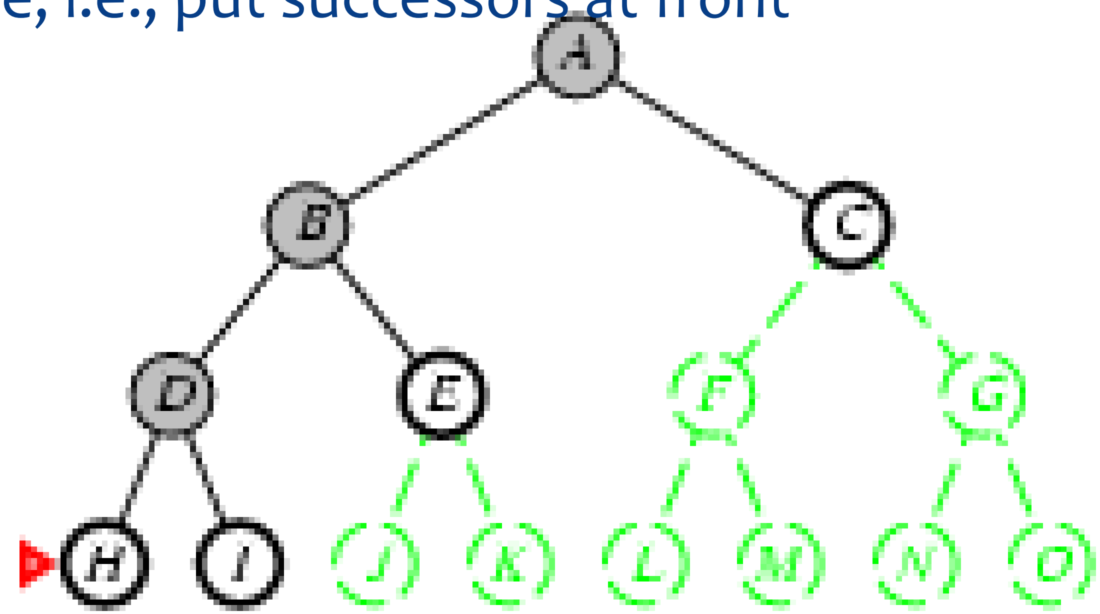


Contd…

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

queue=[H,I,E,C]

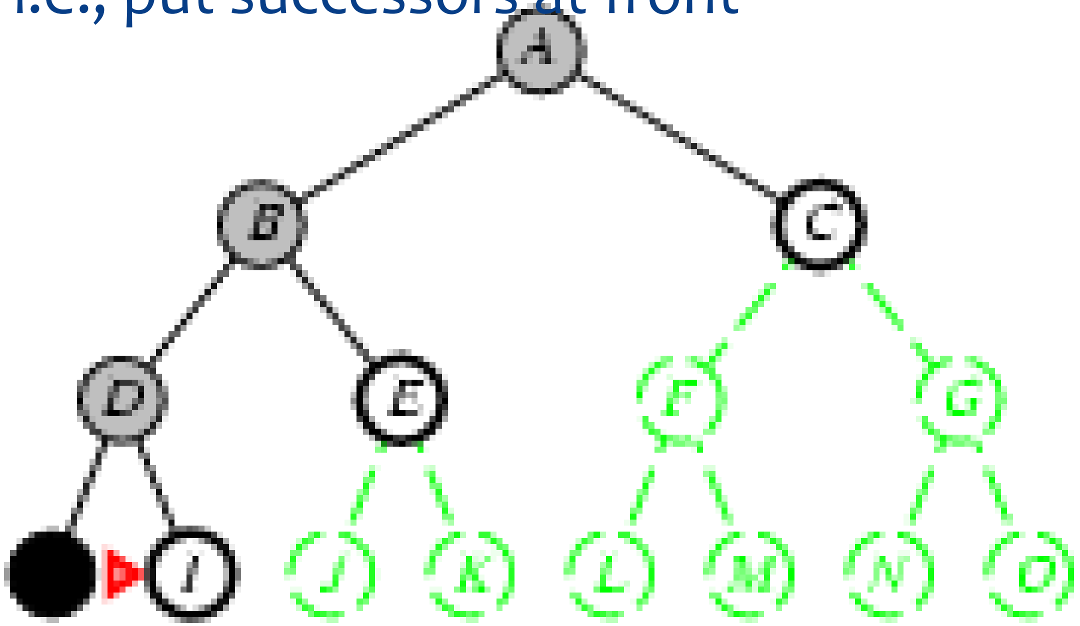
Is H = goal state?



Depth-first search

Expand deepest unexpanded node

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



queue=[I,E,C]

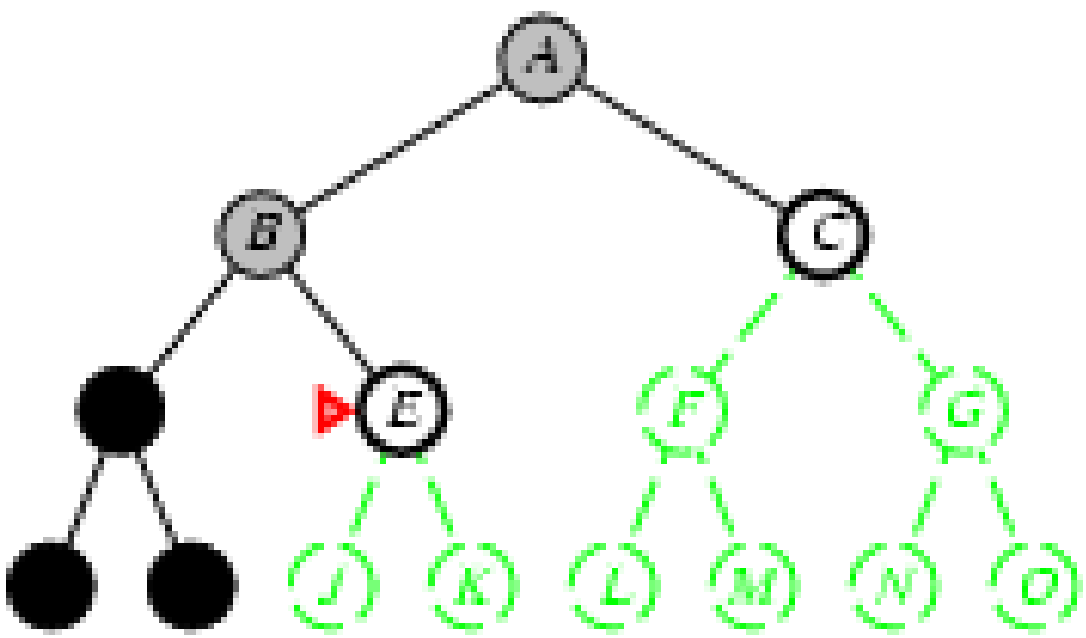
Is I = goal state?

Depth-first search

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

queue=[E,C]

Is E = goal state?

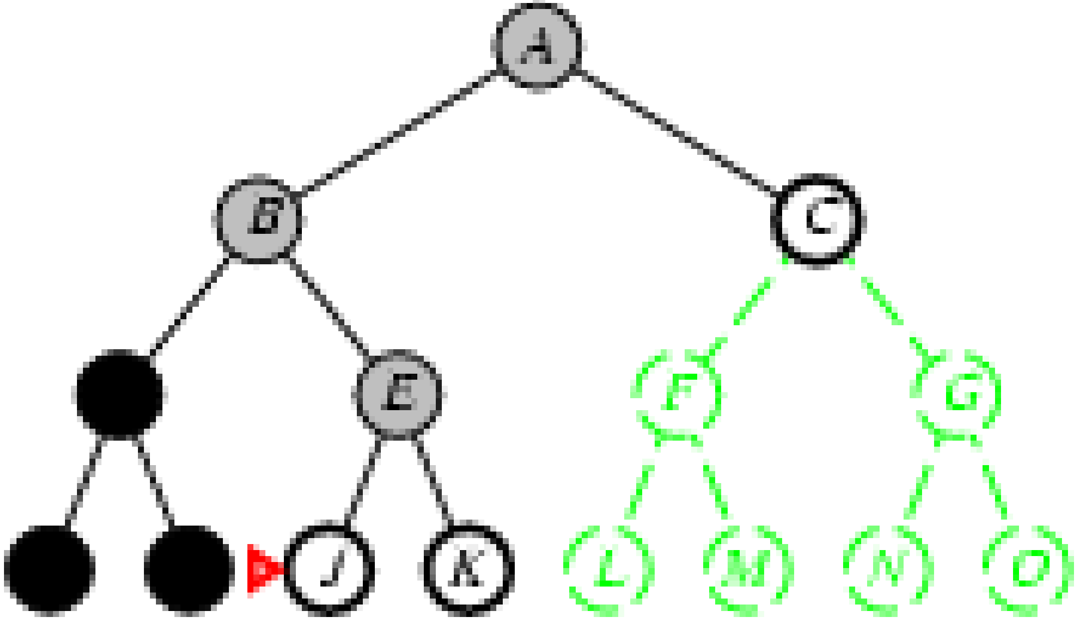


Depth-first search

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

queue=[J,K,C]

Is J = goal state?

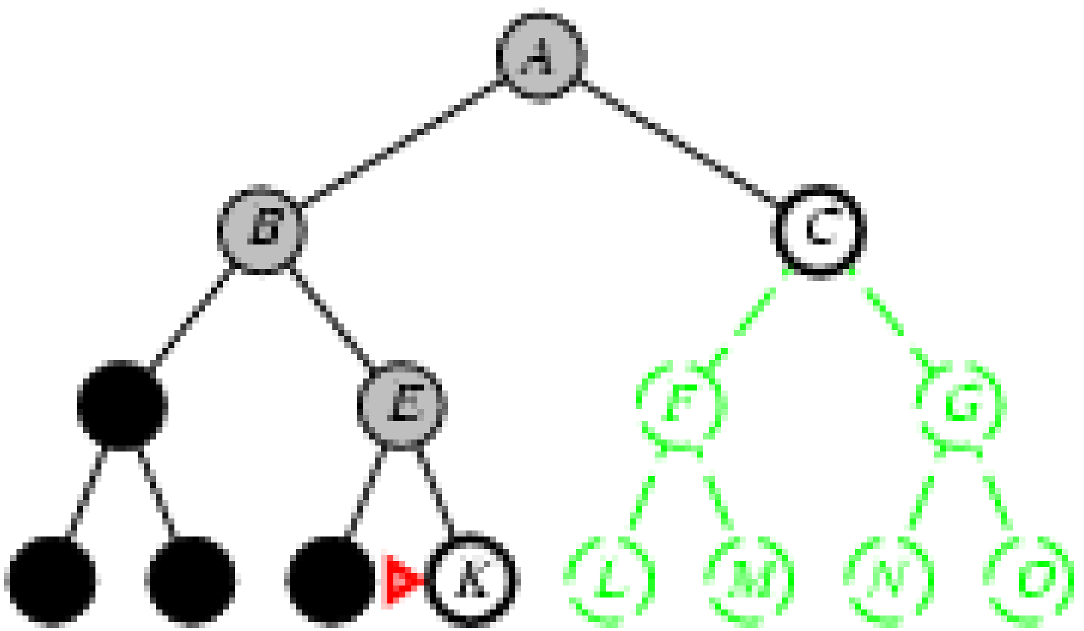


Depth-first search

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

queue=[K,C]

Is K = goal state?

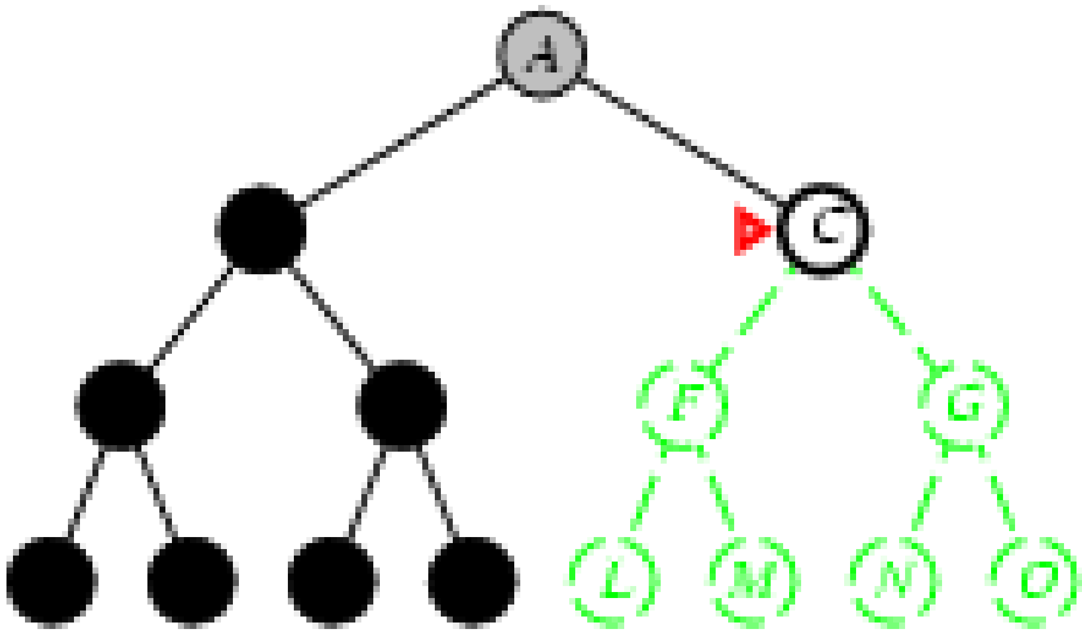


Depth-first search

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

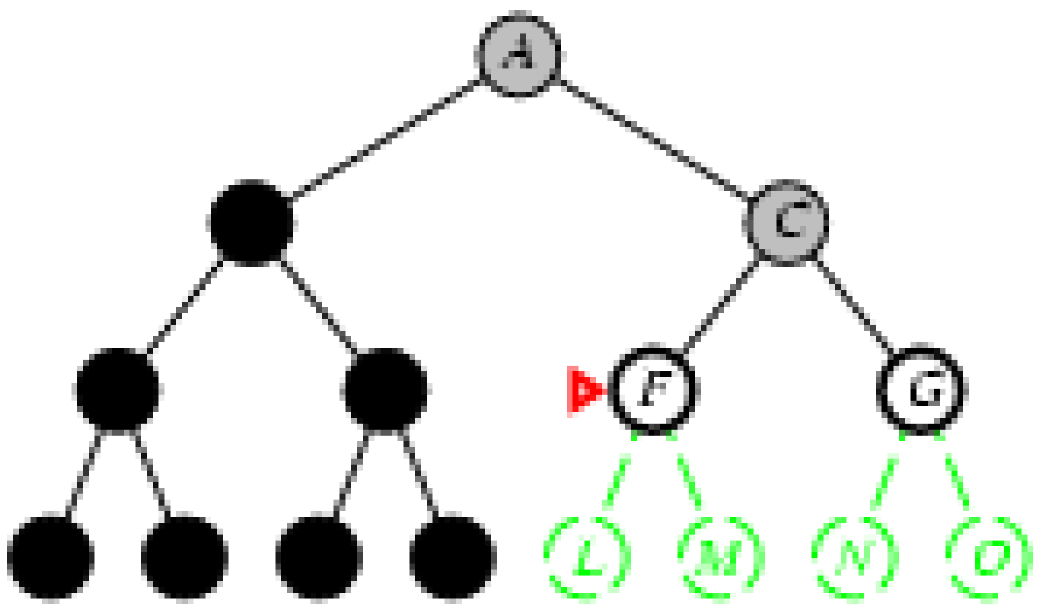
queue=[C]

Is C = goal state?



Depth-first search

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



queue=[F,G]

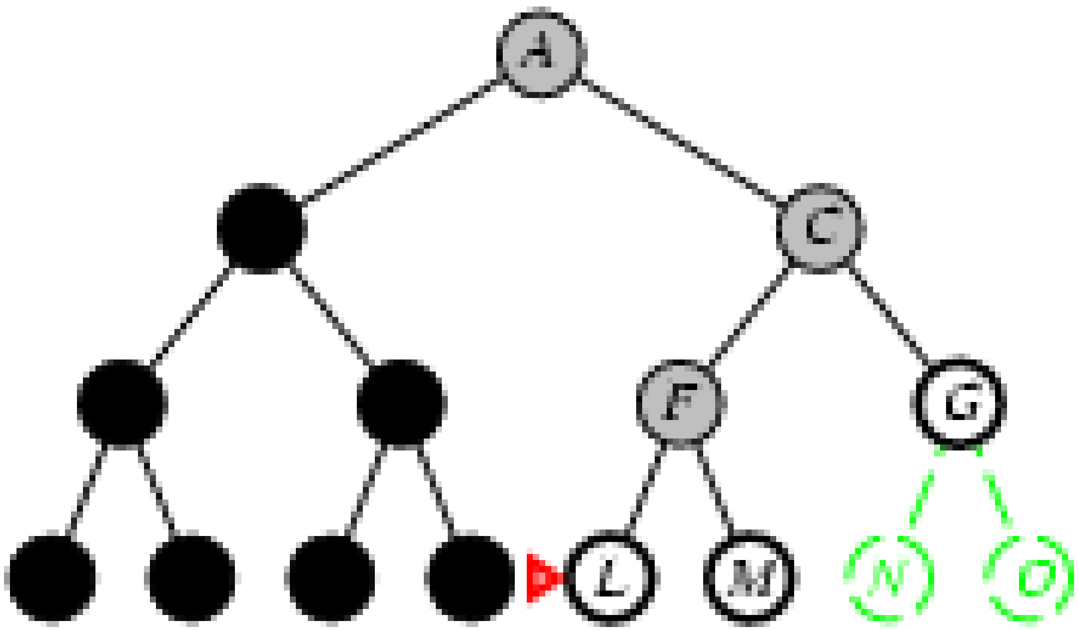
Is F = goal state?

Depth-first search

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

queue=[L,M,G]

Is L = goal state?

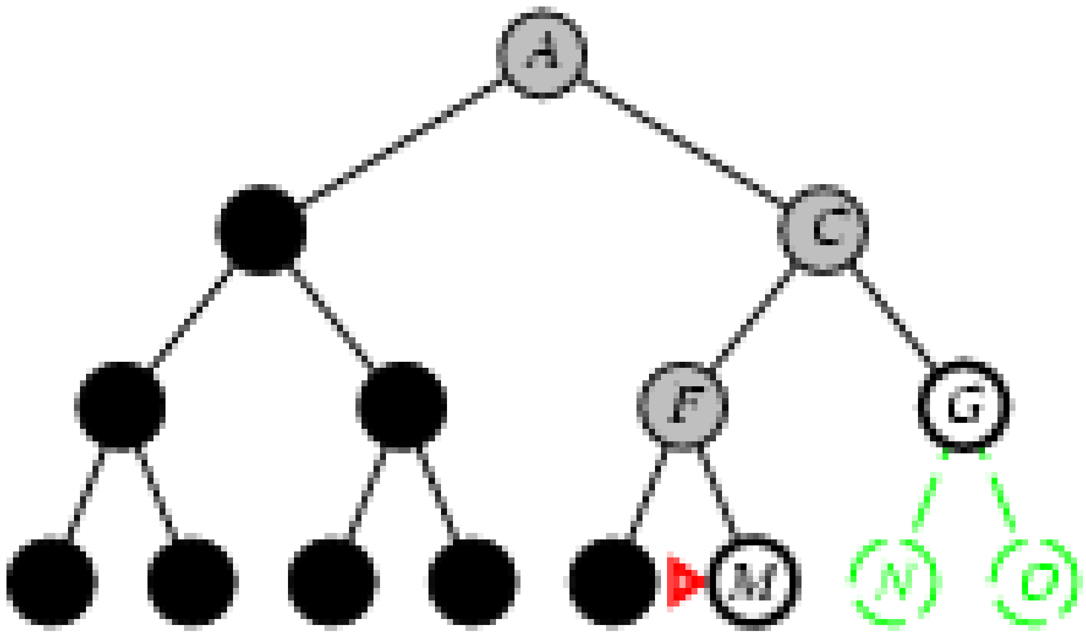


Depth-first search

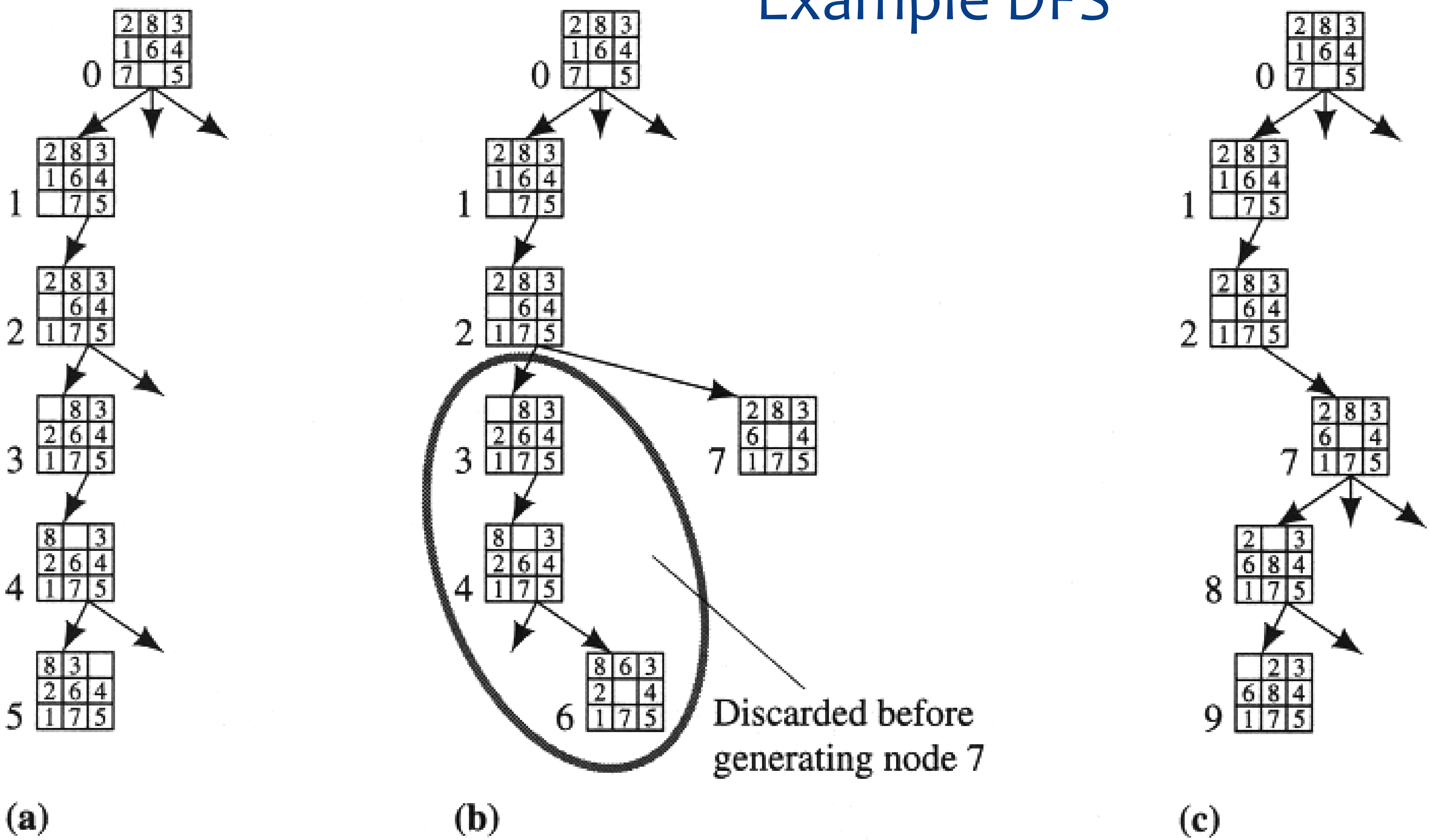
- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[M,G]

Is M = goal state?



Example DFS



Generation of the First Few Nodes in a Depth–First Search

Properties of depth-first search

* **Complete?** No: fails in infinite-depth spaces

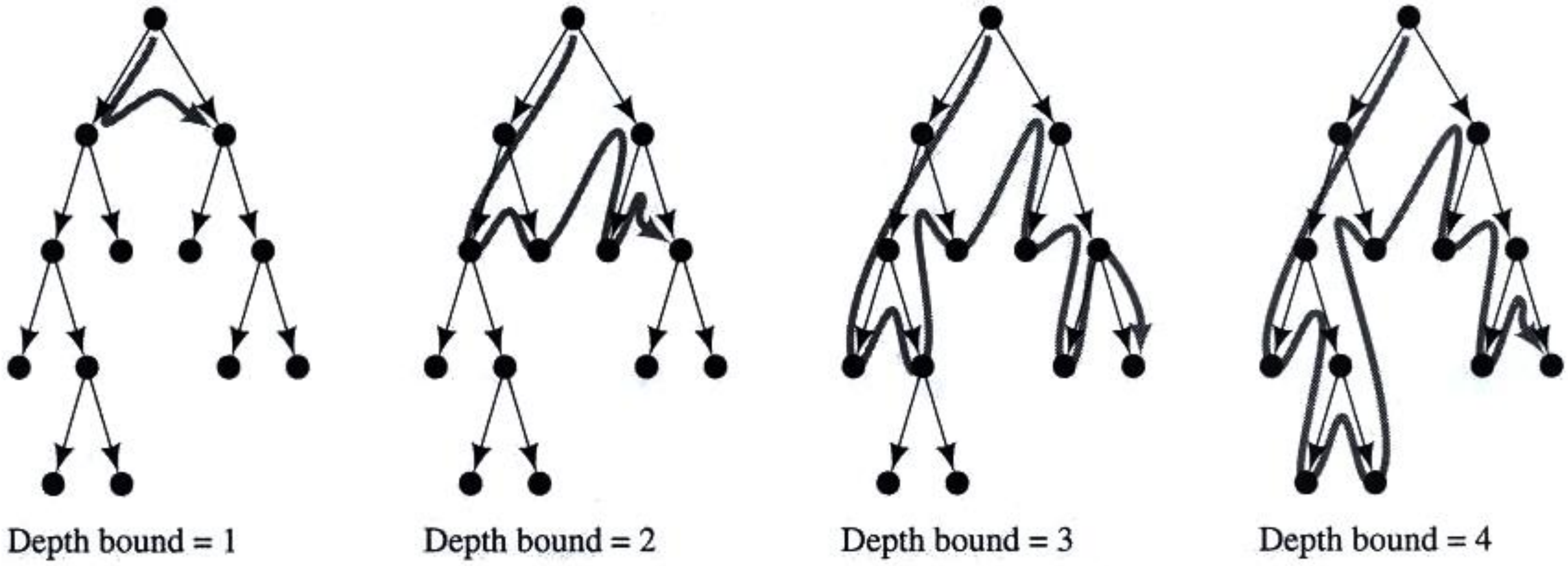
  Can modify to avoid repeated states along path
* **Time?** $O(b^m)$ with m=maximum depth, terrible if m is much larger than d but if solutions are dense, may be much faster than breadth-first
* **Space?** $O(bm)$, i.e., linear space! (we only need to remember a single path + expanded unexplored nodes)
* **Optimal?** No (It may find a non-optimal goal first)

Iterative deepening search

- To avoid the infinite depth problem of DFS, we can decide to only search until depth L, i.e. we don't expand beyond depth L. **Depth-Limited Search**

- What if solution is deeper than L? → Increase L iteratively.
  → **Iterative Deepening Search**

Example IDS

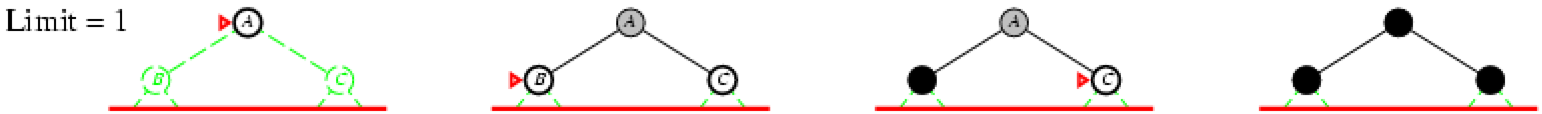


Stages in Iterative-Deepening Search

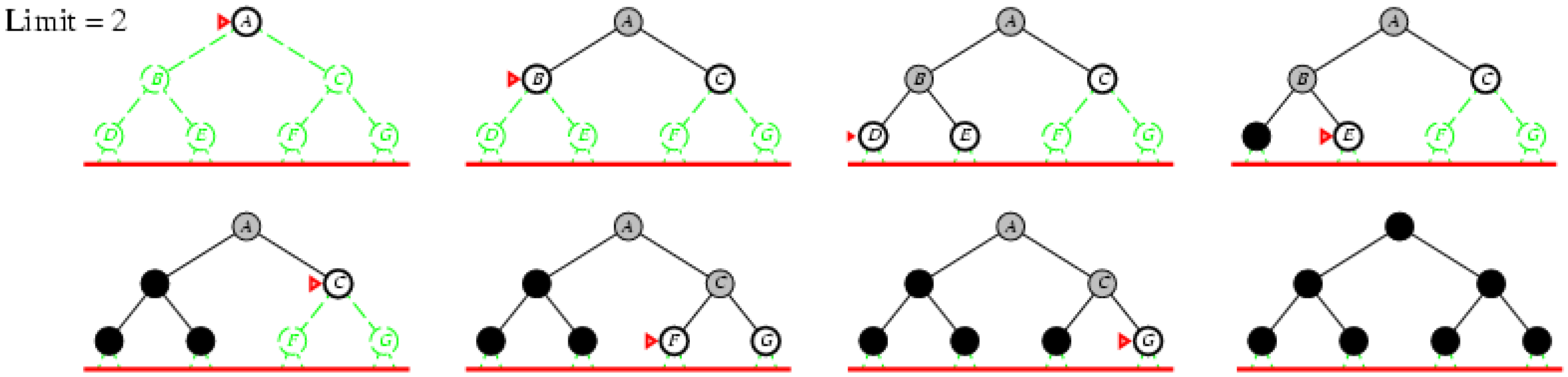
Iterative deepening search *L*=0

Limit = 0

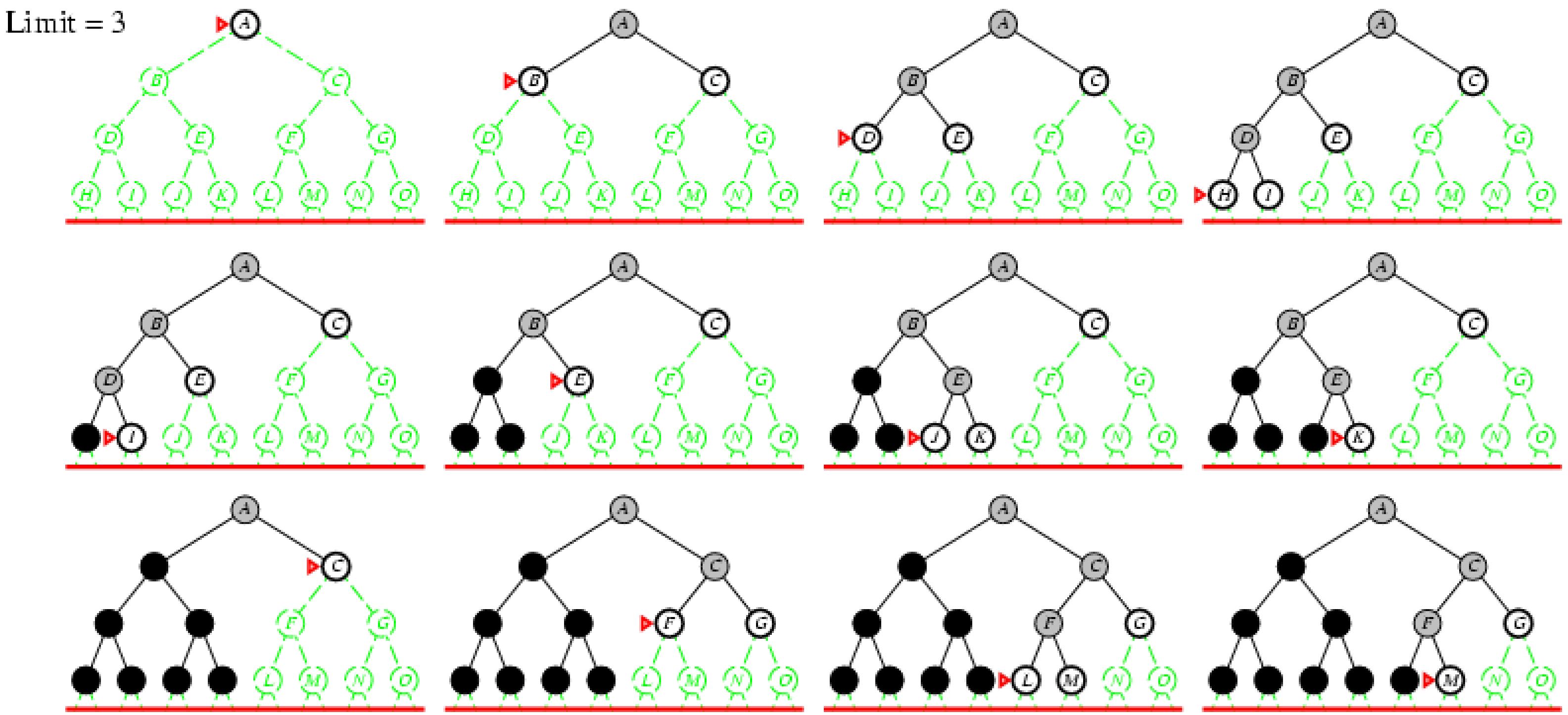
Iterative deepening search *L*=1



Iterative deepening search *l*L=2



Iterative deepening search *l*L=3



Iterative deepening search

- Number of nodes generated in a depth-limited search to depth *d* with branching factor *b*:

  $N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$

- Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:

  $N_{IDS} = (d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d$

- For $b = 10$, $d = 5$,
  - $N_{DLS} = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}111$
  - $N_{IDS} = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}450$
  - N_{BFS} = ... = 1,111,100

$$O(b^d) \neq O(b^{d+1})$$

Properties of iterative deepening search

- **Complete?** Yes
- **Time?** $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- **Space?** $O(bd)$
- **Optimal?** Yes, if step cost = 1 or increasing function of depth.

Evaluating algorithms!!

What makes one search scheme better than another?

- Completeness: Find solution?
- Time complexity: How long?
- Space complexity: Memory?
- Optimality: Find shortest path?

Comparing Uninformed Search Strategies

- Time and space complexity are measured in
  - **b** – maximum branching factor of the search tree
  - **m** – maximum depth of the state space
  - **d** – depth of the least cost solution

Depth vs. Breadth-first

Let $|T(s)| \leq b$ (branching factor), goal at depth d

* How to implement priority queue?
* Completeness?
* Time complexity?
* Space complexity?
* Optimality?

Breadth First Search

* Completeness?
  * Yes
* Time complexity?
  * $O(b^d)$
* Space complexity?
* $O(b^d)$
* Optimality?
  * yes

Depth First Search

* Completeness?
  * Yes, assuming state space finite
* Time complexity?
  * O($|\mathcal{S}|$), can do well if lots of goals
* Space complexity?
  * O(n), n deepest point of search
* Optimality?
  * No

Depth-limited Search

DFS, only expand nodes depth $\leq l$.

- Completeness?
  - No, if $l \leq d$.
- Time complexity?
  - $O(b^l)$
- Space complexity?
  - $O(l)$
- Optimality?
  - No

Iterative Deepening Search

Depth limited, increasing l.

- Completeness?
  - Yes.
- Time complexity?
  - $O(b^d)$, even with repeated work!
- Space complexity?
  - $O(d)$
- Optimality?
  - Yes

Searching For Solutions

```
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
    end
```