Solving problems by searching
Uninformed search

CE417: Introduction to Artificial Intelligence
Sharif University of Technology
Spring 2016

Soleymani

"Artificial Intelligence: A Modern Approach", Chapter 3

Outline

- Problem-solving agents
  - Problem formulation and some examples of problems
- Uninformed search algorithms
  - Using only the problem definition

Problem-Solving Agents

- Problem Formulation: process of deciding what actions and states to consider
  - States of the world
  - Actions as transitions between states
- Goal Formulation: process of deciding what the next goal to be sought will be
- Agent must find out how to act now and in the future to reach a goal state
  - Search: process of looking for solution (a sequence of actions that reaches the goal starting from initial state)

Problem-Solving Agents

- A goal-based agent adopts a goal and aim at satisfying it

  (as a simple version of intelligent agent maximizing a performance measure)

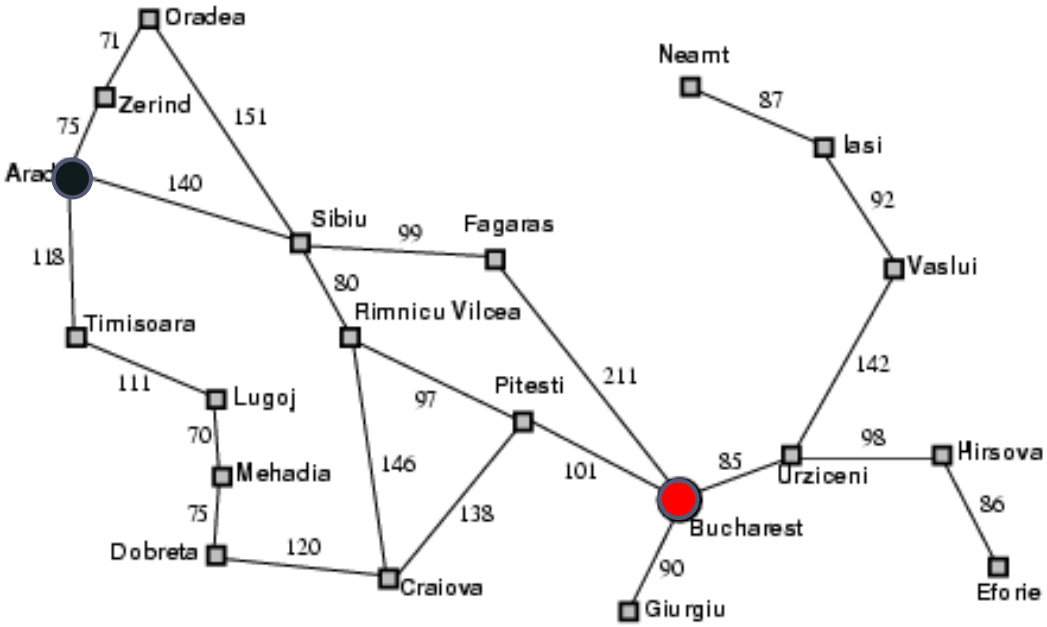
- "How does an intelligent system formulate its problem as a search problem"
  - Goal formulation: specifying a goal (or a set of goals) that agent must reach
  - Problem formulation: abstraction (removing detail)
    - Retaining validity and ensuring that the abstract actions are easy to perform

Example: Romania

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

- **Initial state**
  - currently in Arad
- **Formulate goal**
  - be in Bucharest
- **Formulate problem**
  - states: various cities
  - actions: drive between cities
- **Solution**
  - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Map of Romania



Example: Romania (Cont.)

- Assumptions about environment
  - Known
  - Observable
    - The initial state can be specified exactly.
  - Deterministic
    - Each applied action to a state results in a specified state.
  - Discrete

Given the above first three assumptions, by starting in an initial state and running a sequence of actions, it is absolute where the agent will be

- Perceptions after each action provide no new information
  - Can search with closed eyes (open-loop)

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
```

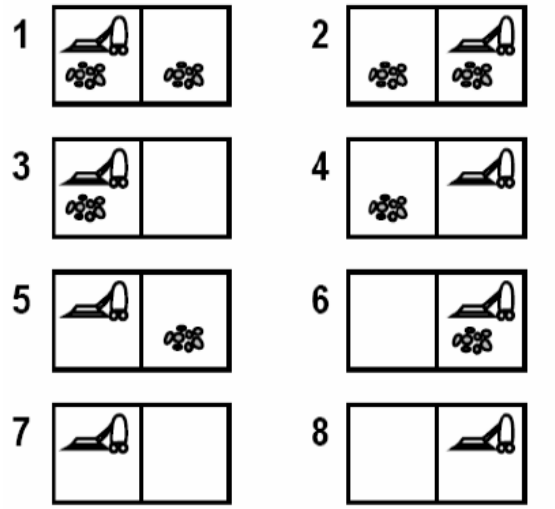
Formulate, Search, Execute

Problem types

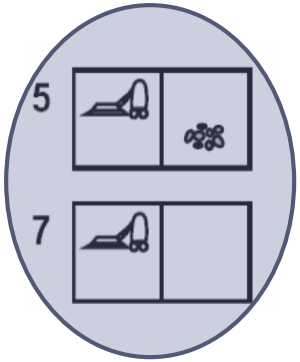
- Deterministic and fully observable (single-state problem)
  - Agent knows exactly its state even after a sequence of actions
  - Solution is a sequence
- Non-observable or sensor-less (conformant problem)
  - Agent's percepts provide no information at all
  - Solution is a sequence
- Nondeterministic and/or partially observable (contingency problem)
  - Percepts provide new information about current state
  - Solution can be a contingency plan (tree or strategy) and not a sequence
  - Often interleave search and execution
- Unknown state space (exploration problem)

Belief State

- In partially observable & nondeterministic environments, a state is not necessarily mapped to a world configuration
  - State shows the agent's conception of the world state
    - Agent's current belief (given the sequence of actions and percepts up to that point) about the possible physical states it might be in.



World states



A sample belief state

Example: vacuum world

- Single-state, start in {5}

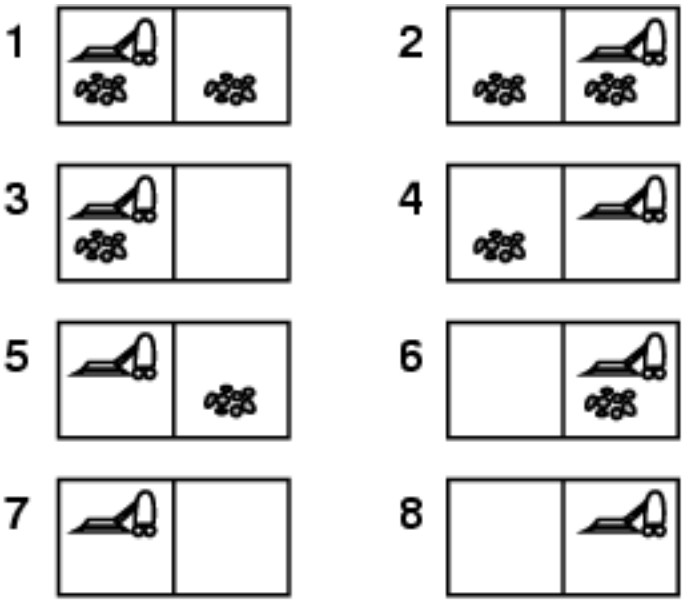
  Solution?

  [Right, Suck]

- Sensorless, 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
  - Nondeterministic: Suck may dirty a clean carpet
  - Partially observable: location, dirt only at the current location
  - Percept: [L, Clean], i.e., start in {5} or {7}

  Solution?

  [Right, **if** dirt **then** Suck] [Right, **while** dirt **do** Suck]

Single-state problem

- In this lecture, we focus on single-state problem
  - Search for this type of problems is simpler
  - And also provide strategies that can be base for search in more complex problems

Single-state problem formulation

A problem is defined by five items:

- Initial state e.g., $In(\ Arad)$
- Actions: $ACTIONS(s)$ shows set of actions that can be executed in s
  - e.g., $ACTIONS(In(Arad)) = \{Go(Sibiu), Go(Timisoara), Go(Zerind)\}$

Single-state problem formulation

A problem is defined by five items:

- Initial state e.g., $In(\ Arad)$
- Actions: $ACTIONS(s)$ shows set of actions that can be executed in s
- Transition model: $RESULTS(s, a)$ shows the state that results from doing action a in state s
  - e.g., $RESULTS(In(Arad), Go(Zerind)) = In(Zerind)$

Single-state problem formulation

A problem is defined by five items:

- Initial state e.g., $In(\ Arad)$
- Actions: $ACTIONS(s)$ shows set of actions that can be executed in s
- Transition model: $RESULTS(s, a)$ shows the state that results from doing action a in state s
- Goal test: $GOAL\_TEST(s)$ shows whether a given state is a goal state
  - explicit, e.g., *x* = "at Bucharest"
  - abstract e.g., *Checkmate(x)*

Single-state problem formulation

A problem is defined by five items:

- Initial state e.g., $In(Arad)$
- Actions: $ACTIONS(s)$ shows set of actions that can be executed in s
- Transition model: $RESULTS(s, a)$ shows the state that results from doing action a in state s
- Goal test: $GOAL\_TEST(s)$ shows whether a given state is a goal state
- Path cost (additive): assigns a numeric cost to each path that reflects agent's performance measure
  - e.g., sum of distances, number of actions executed, etc.
  - $c(x, a, y) \geq 0$ is the step cost

Single-state problem formulation

A problem is defined by five items:

- Initial state e.g., $In(Arad)$
- Actions: $ACTIONS(s)$ shows set of actions that can be executed in s
- Transition model: $RESULTS(s, a)$ shows the state that results from doing action a in state s
- Goal test: $GOAL\_TEST(s)$ shows whether a given state is a goal state
- Path cost (additive): assigns a numeric cost to each path that reflects agent's performance measure

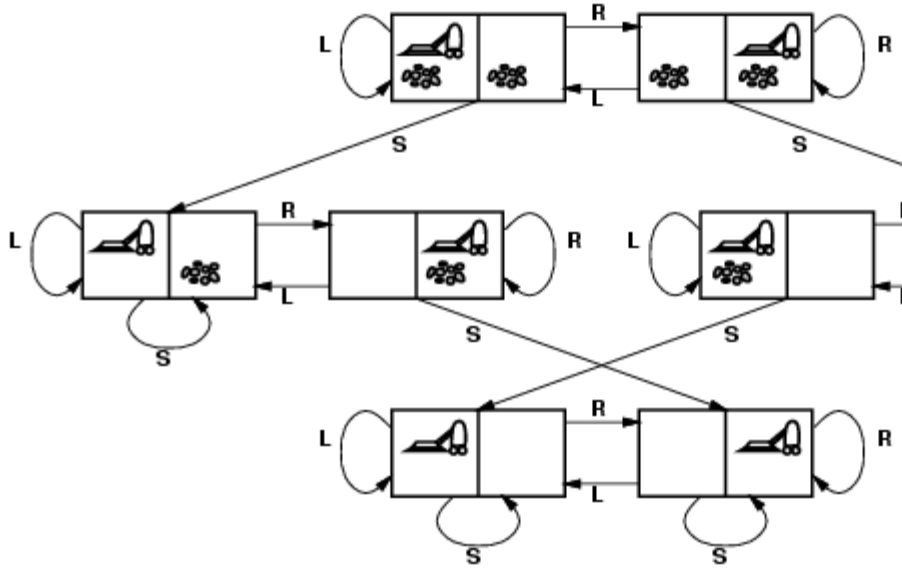
Solution: a sequence of actions leading from the initial state to a goal state

Optimal Solution has the lowest path cost among all solutions.

State Space

- State space: set of all reachable states from initial state
  - Initial state, actions, and transition model together define it
- It forms a directed graph
  - Nodes: states
  - Links: actions
- Constructing this graph on demand

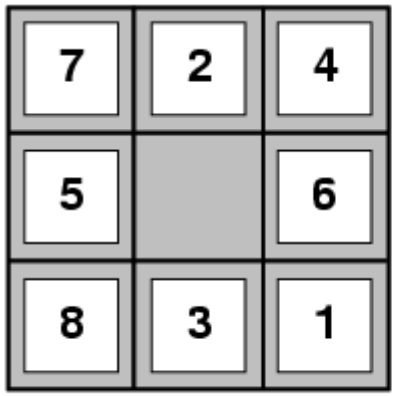
Vacuum world state space graph



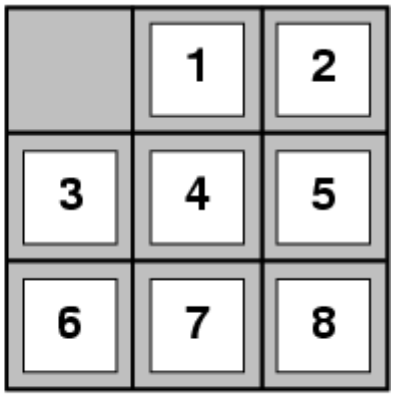
$2 \times 2^2 = 8$ States

- States? dirt locations & robot location
- Actions? Left, Right, Suck
- Goal test? no dirt at all locations
- Path cost? one per action

Example: 8-puzzle



Start State



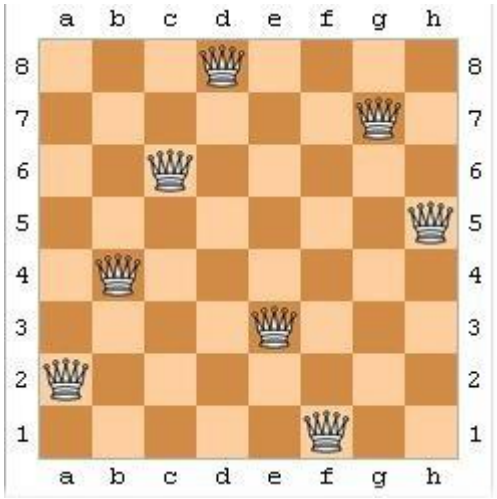
Goal State

$9!/2 = 181{,}440$ States

- States? locations of eight tiles and blank in 9 squares
- Actions? move blank left, right, up, down (within the board)
- Goal test? e.g., the above goal state
- Path cost? one per move

Note: optimal solution of *n*-Puzzle family is NP-complete

Example: 8-queens problem

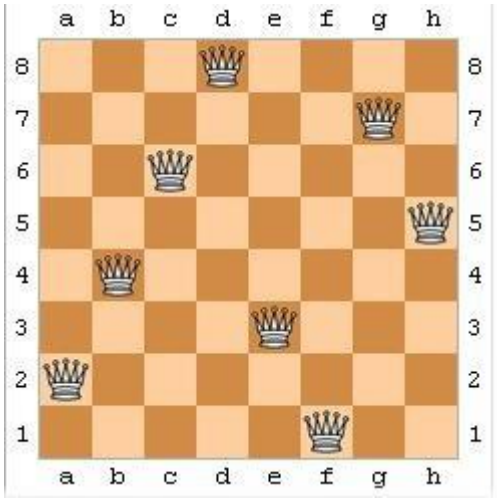


$64 \times 63 \times \cdots \times 57$
$\simeq 1.8 \times 10^{14}$ States

- Initial State? no queens on the board
- States? any arrangement of 0-8 queens on the board is a state
- Actions? add a queen to the state (any empty square)
- Goal test? 8 queens are on the board, none attacked
- Path cost? of no interest

  search cost vs. solution path cost

Example: 8-queens problem (other formulation)



2,057 States

- Initial state? no queens on the board
- States? any arrangement of *k* queens one per column in the leftmost *k* columns with no queen attacking another
- Actions? add a queen to any square in the leftmost empty column such that it is not attacked by any other queen
- Goal test? 8 queens are on the board
- Path cost? of no interest

Example: Knuth problem

- Knuth Conjecture: Starting with 4, a sequence of factorial, square root, and floor operations will reach any desired positive integer.
  - Example: $\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}} \right\rfloor = 5$

- States? Positive numbers
- Initial State? 4
- Actions? Factorial (for integers only), square root, floor
- Goal test? State is the objective positive number
- Path cost? Of no interest

Read-world problems

- Route finding
- Travelling salesman problem
- VLSI layout
- Robot navigation
- Automatic assembly sequencing

Tree search algorithm

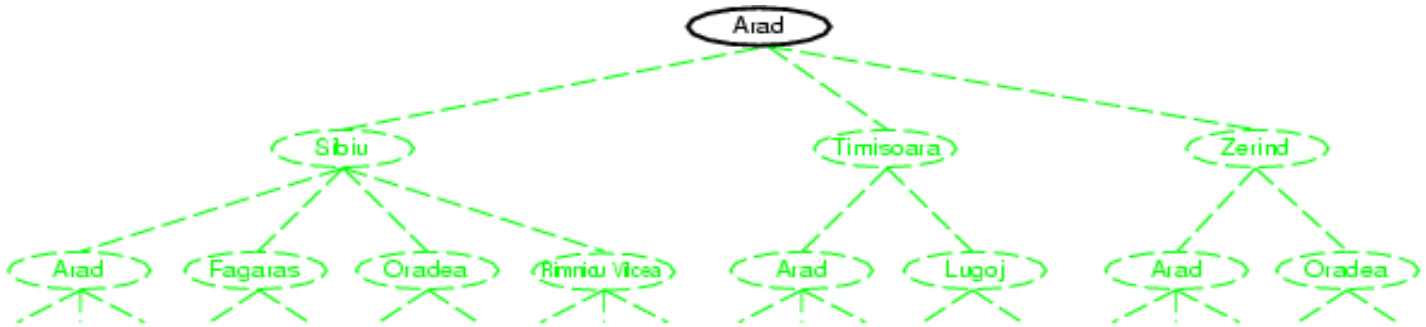
- Basic idea
  - offline, simulated exploration of state space by generating successors of already-explored states

```
function TREE-SEARCH( problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
```

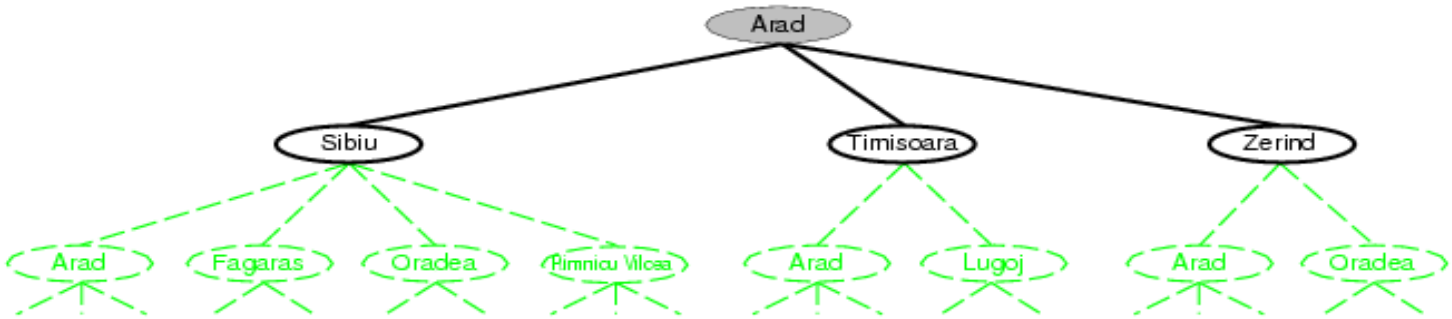
Frontier: all leaf nodes available for expansion at any given point

Different data structures (e.g, FIFO, LIFO) for **frontier** can cause different orders of node expansion and thus produce different search algorithms.

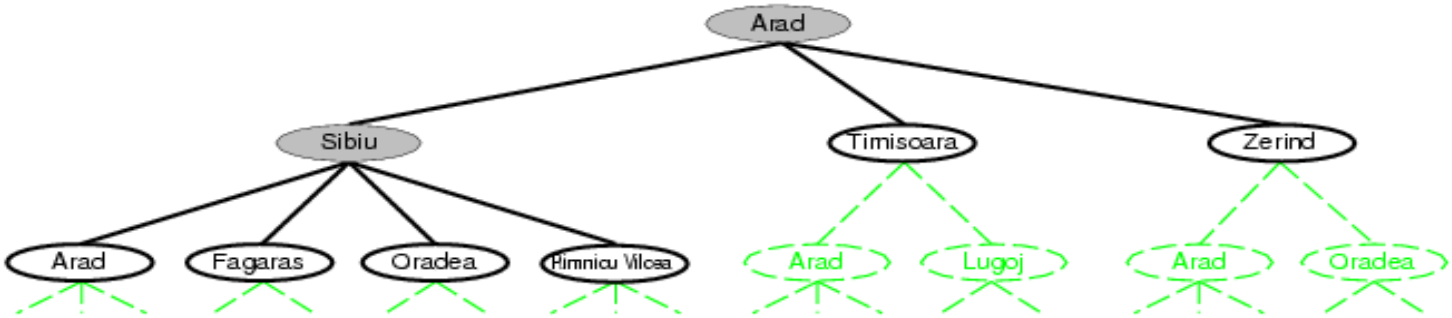
Tree search example



Tree search example



Tree search example



Graph Search

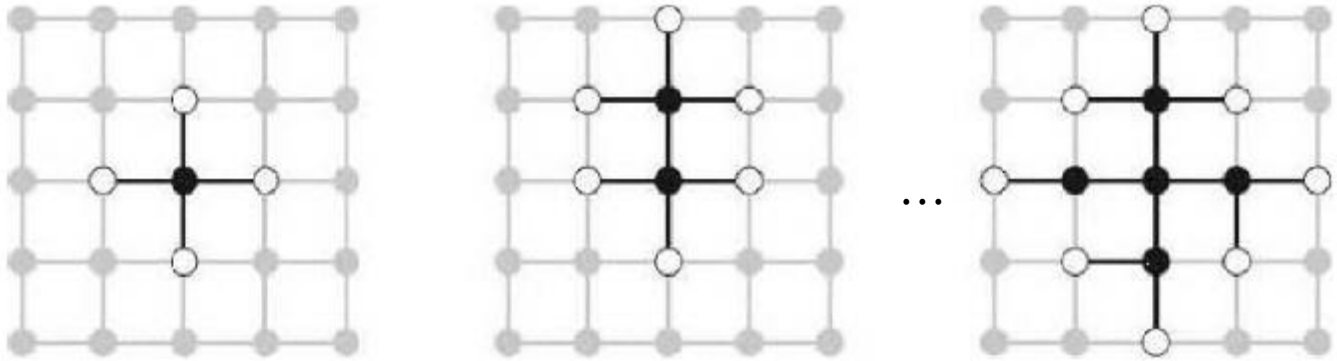
- Redundant paths in tree search: more than one way to get from one state to another
  - may be due to a bad problem definition or the essence of the problem
  - can cause a tractable problem to become intractable

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
    initialize the frontier using the initial state of *problem*
    loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
        only if not in the frontier or explored set

explored set: remembered every explored node

Graph Search

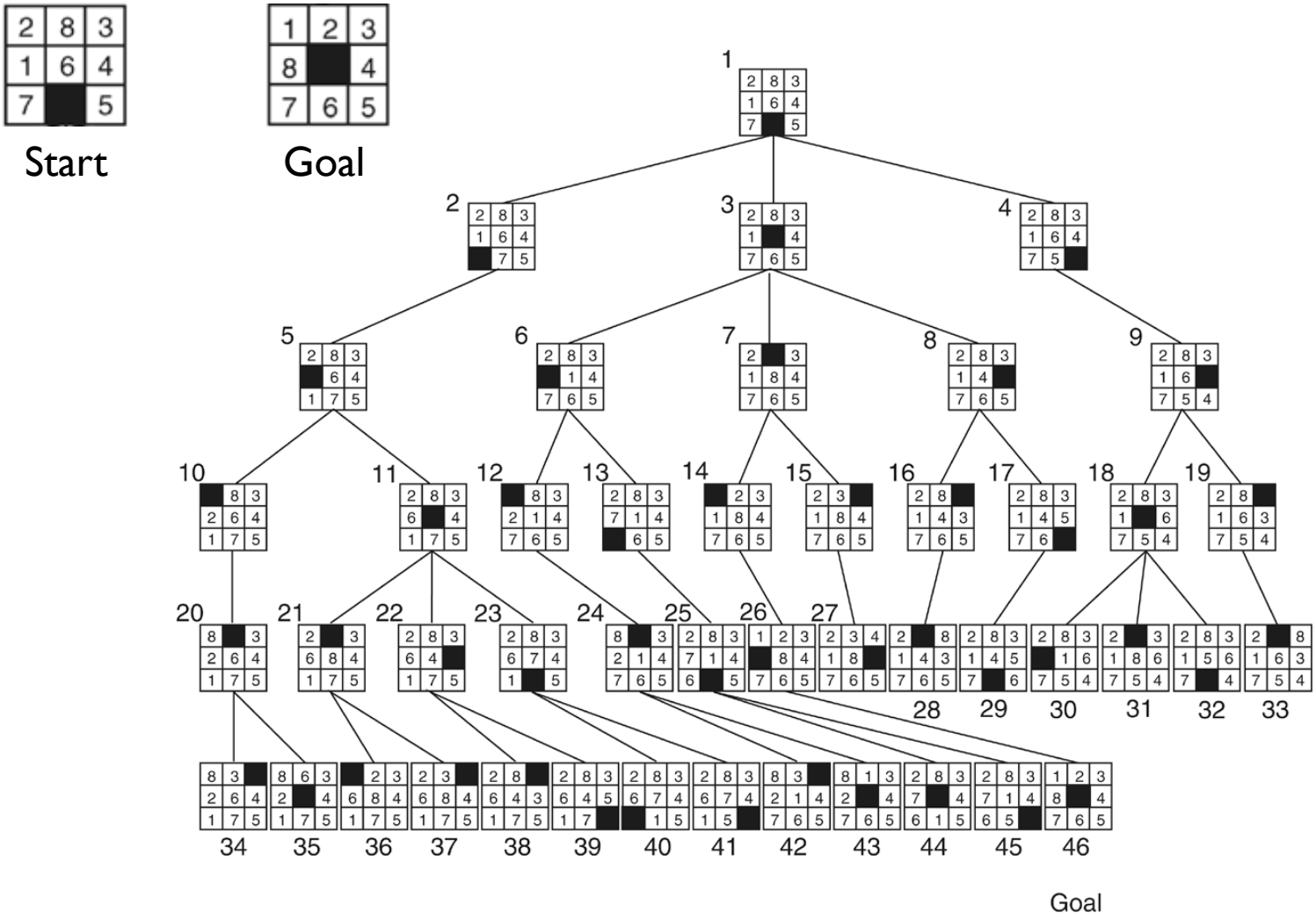
- Example: rectangular grid



explored

frontier

Search for 8-puzzle Problem

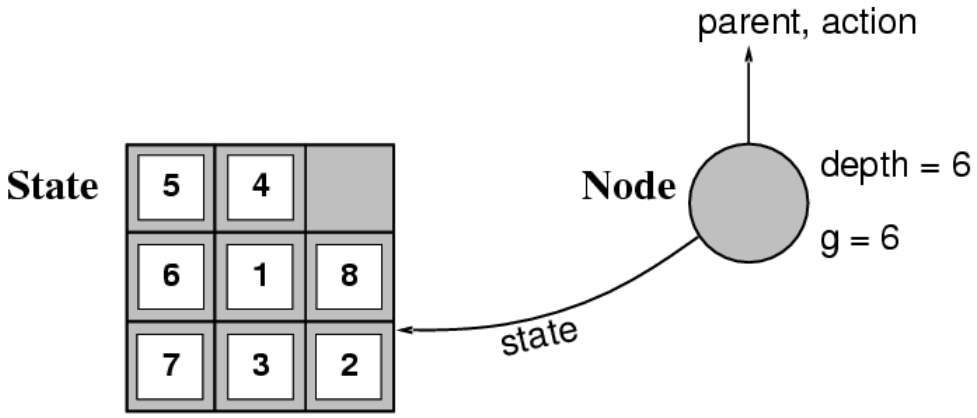


Taken from: http://iis.kaist.ac.kr/es/

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 $g(x)$, depth



Search strategies

- Search strategy: order of node expansion
- Strategies performance evaluation:
  - **Completeness**: Does it always find a solution when there is one?
  - **Time complexity**: How many nodes are generated to find solution?
  - **Space complexity**: Maximum number of nodes in memory during search
  - **Optimality**: Does it always find a solution with minimum path cost?

- Time and space complexity are expressed by
  - *b (branching factor):* maximum number of successors of any node
  - *d (depth):* depth of the shallowest goal node
  - *m*: maximum depth of any node in the search space (may be ∞)

- Time & space are described for tree search
  - For graph search, analysis depends on redundant paths

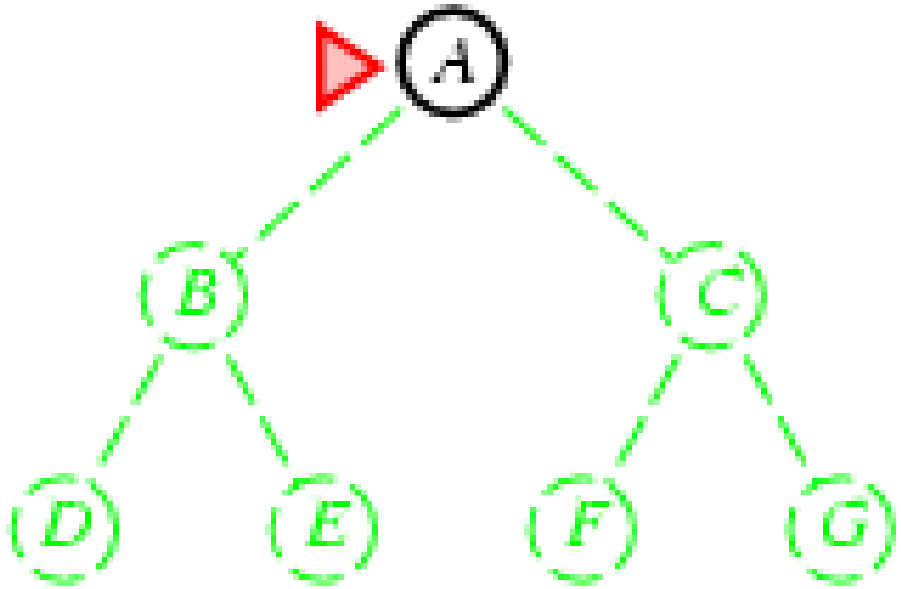
Uninformed Search Algorithms

Uninformed (blind) search strategies

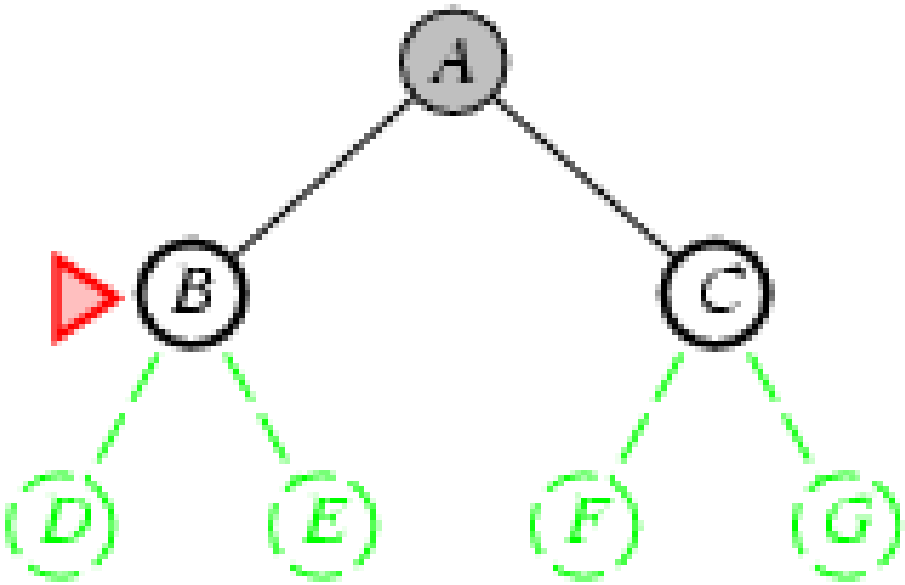
- No additional information beyond the problem definition
  - Breadth-First Search (BFS)
  - Uniform-Cost Search (UCS)
  - Depth-First Search (DFS)
  - Depth-Limited Search (DLS)
  - Iterative Deepening Search (IDS)

Breadth-first search

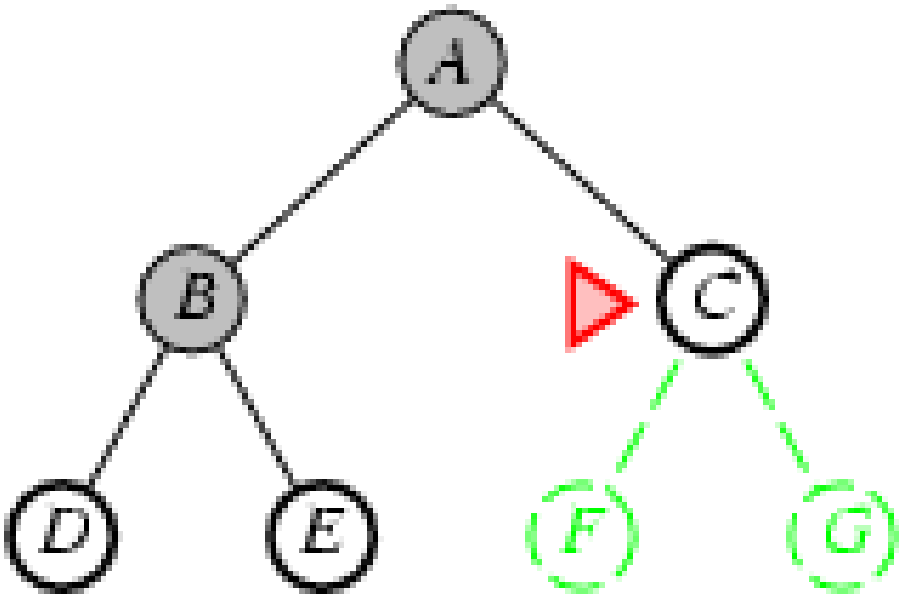
- Expand the shallowest unexpanded node
- Implementation: FIFO queue for the frontier



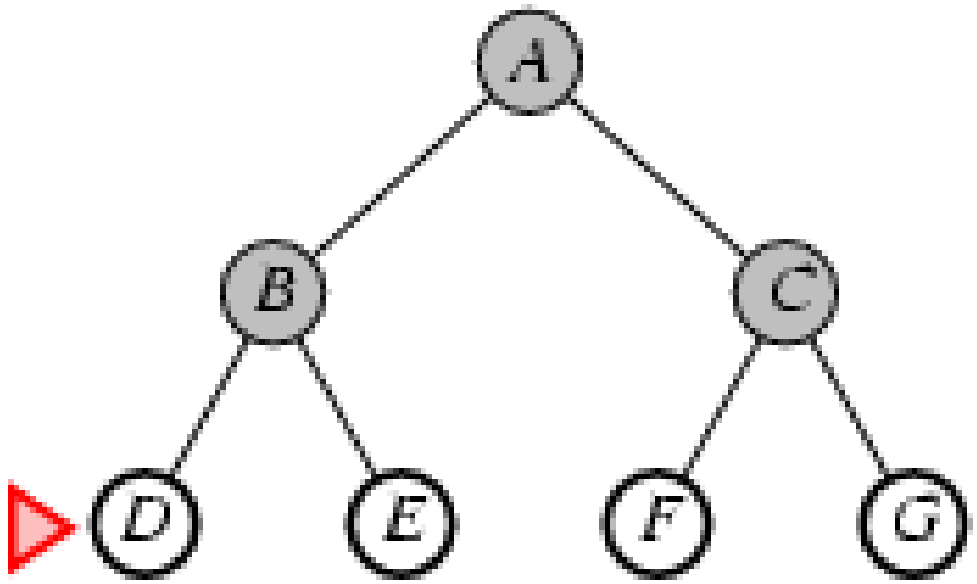
Breadth-first search



Breadth-first search



Breadth-first search



Properties of breadth-first search

- Complete?
  - Yes (for finite b and d)
- Time
  - $b + b^2 + b^3 + \cdots + b^d = O(b^d)$ total number of generated nodes
    - goal test has been applied to each node when it is generated
- Space
  - $O(b^{d-1})$ (explored) + $O(b^d)$ (frontier) $= O(b^d)$ (graph search)
    - Tree search does not save much space while may cause a great time excess
- Optimal?
  - Yes, if path cost is a non-decreasing function of *d*
    - e.g. all actions having the same cost

Properties of breadth-first search

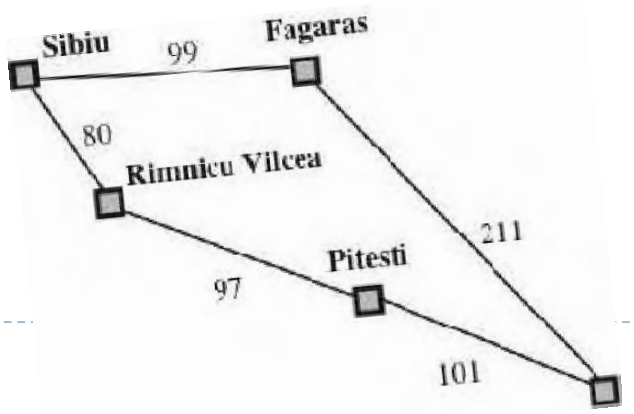
- Space complexity is a bigger problem than time complexity
- Time is also prohibitive
  - Exponential-complexity search problems cannot be solved by uninformed methods (only the smallest instances)

1 million node/sec, 1kb/node

d	Time	Memory
10	3 hours	10 terabytes
12	13 days	1 pentabyte
14	3.5 years	99 pentabytes
16	350 years	10 exabytes

Uniform-Cost Search (UCS)

- Expand node n (in the frontier) with the lowest path cost $g(n)$
  - Extension of BFS that is proper for any step cost function
- Implementation: Priority queue (ordered by path cost) for frontier
- Equivalent to breadth-first if all step costs are equal
  - Two differences
    - Goal test is applied when a node is selected for expansion
    - A test is added when a better path is found to a node currently on the frontier



$$80 + 97 + 101 < 99 + 211$$

Properties of uniform-cost search

- Complete?
  - Yes, if step cost $\geq \varepsilon > 0$ (to avoid infinite sequence of zero-cost actions)
- Time
  - Number of nodes with $g \leq$ cost of optimal solution, $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$ where C^* is the optimal solution cost
    - $O(b^{d+1})$ when all step costs are equal
- Space
  - Number of nodes with $g \leq$ cost of optimal solution, $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$
- Optimal?
  - Yes – nodes expanded in increasing order of $g(n)$

Difficulty: many long paths may exist with cost $\leq C^*$

Uniform-cost search (proof of optimality)

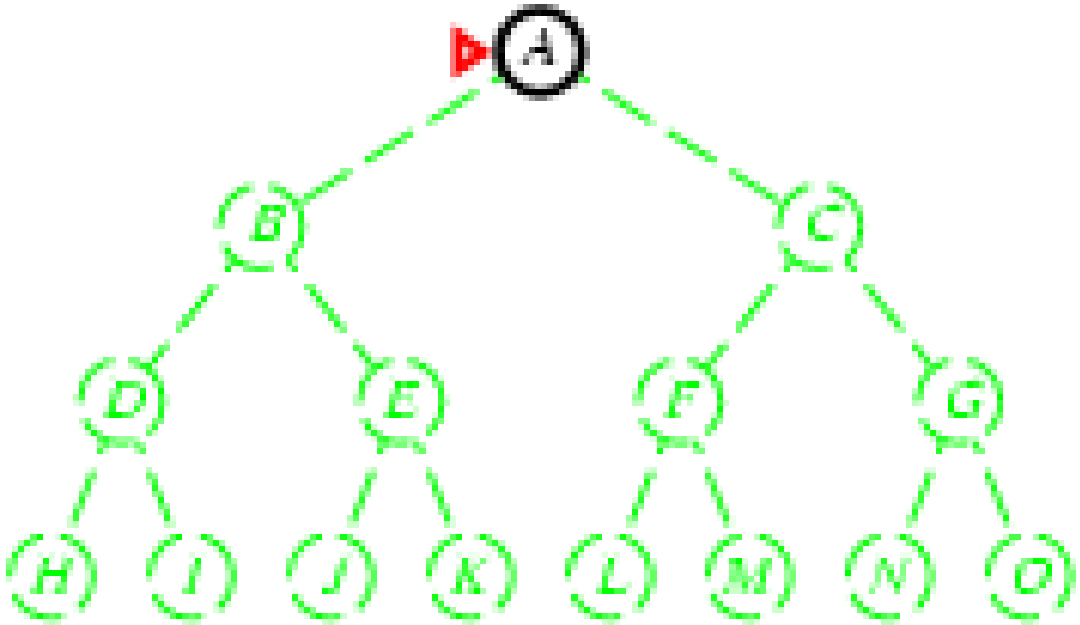
- **Lemma**: If UCS selects a node n for expansion, the optimal solution to that node has been found.

  Proof by contradiction: Another frontier node n' must exist on the optimal path from initial node to n (using graph separation property). Moreover, based on definition of path cost (due to non-negative step costs, paths never get shorter as nodes are added), we have $g(n') \leq g(n)$ and thus n' would have been selected first.

⇒ Nodes are expanded in order of their optimal path cost.

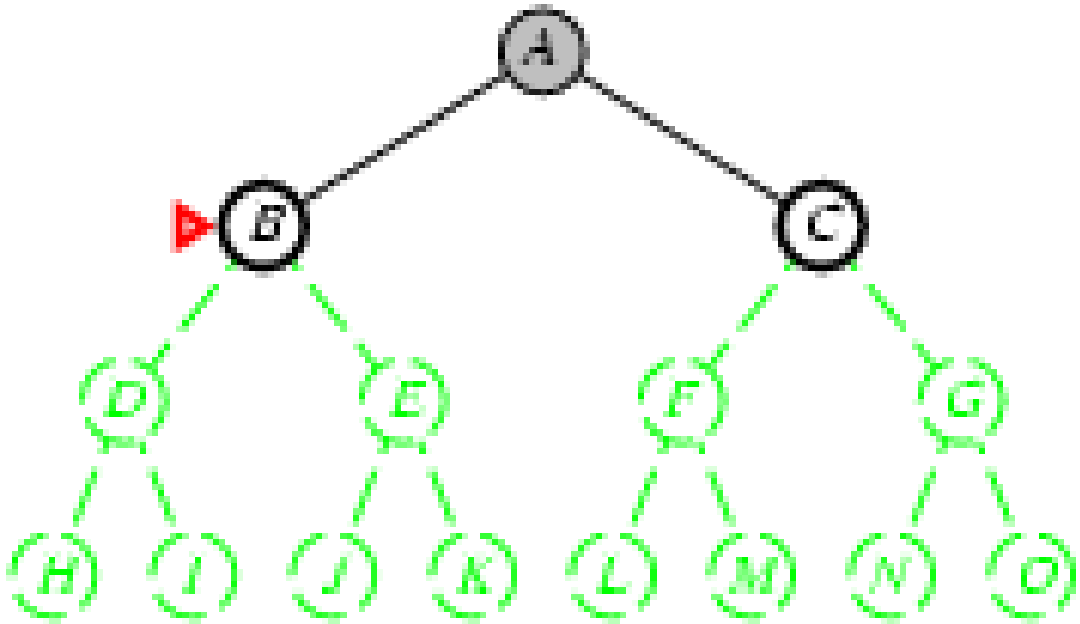
Depth First Search (DFS)

- Expand the deepest node in frontier
- Implementation: LIFO queue (i.e., put successors at front) for frontier



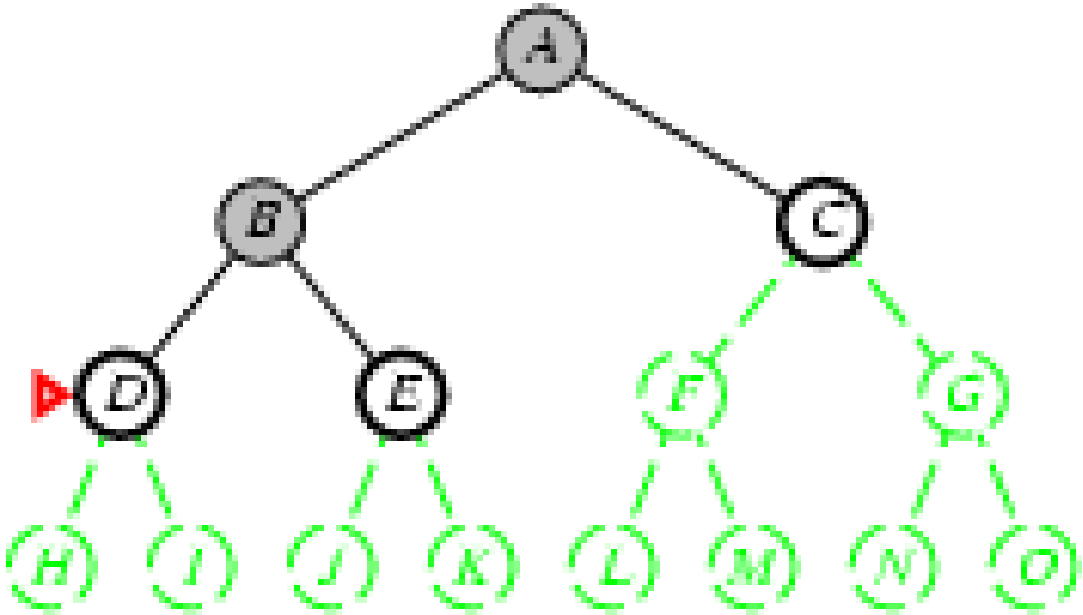
DFS

- Expand the deepest unexpanded node in frontier



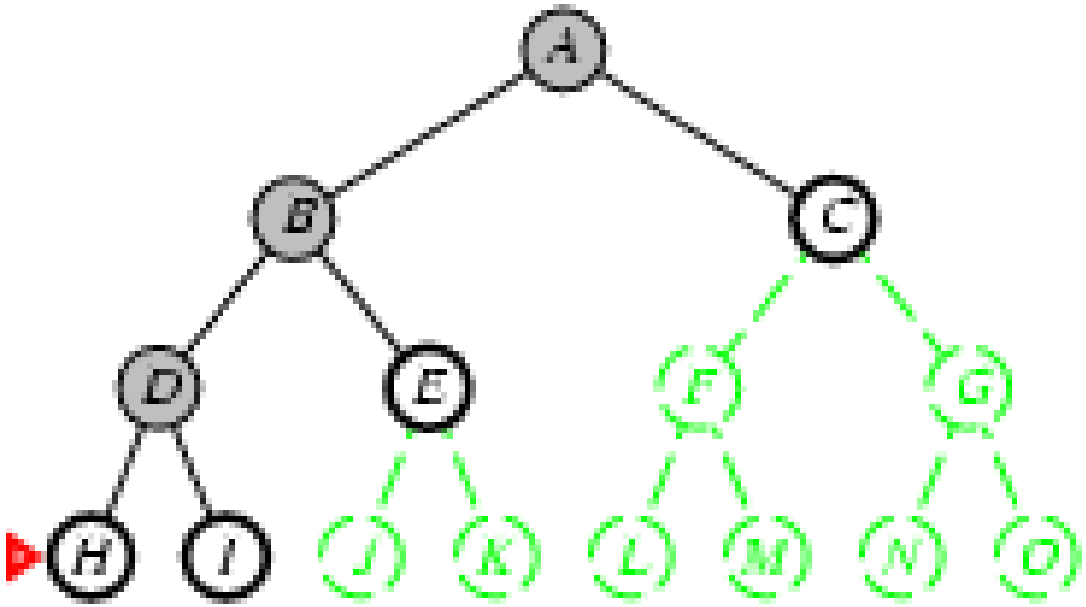
DFS

- Expand the deepest unexpanded node in frontier



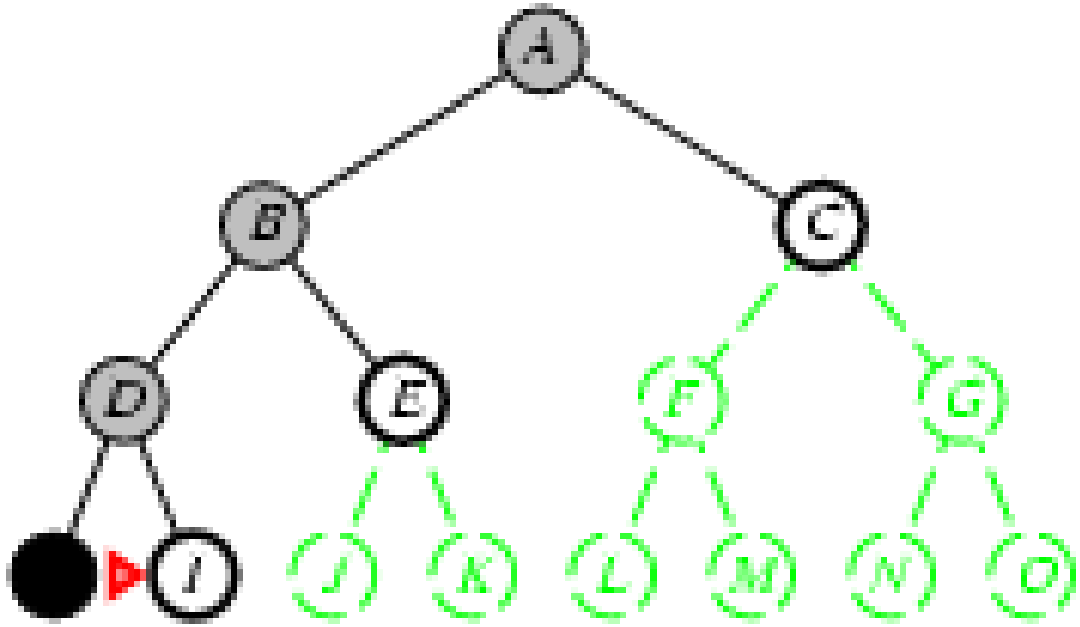
DFS

- Expand the deepest unexpanded node in frontier



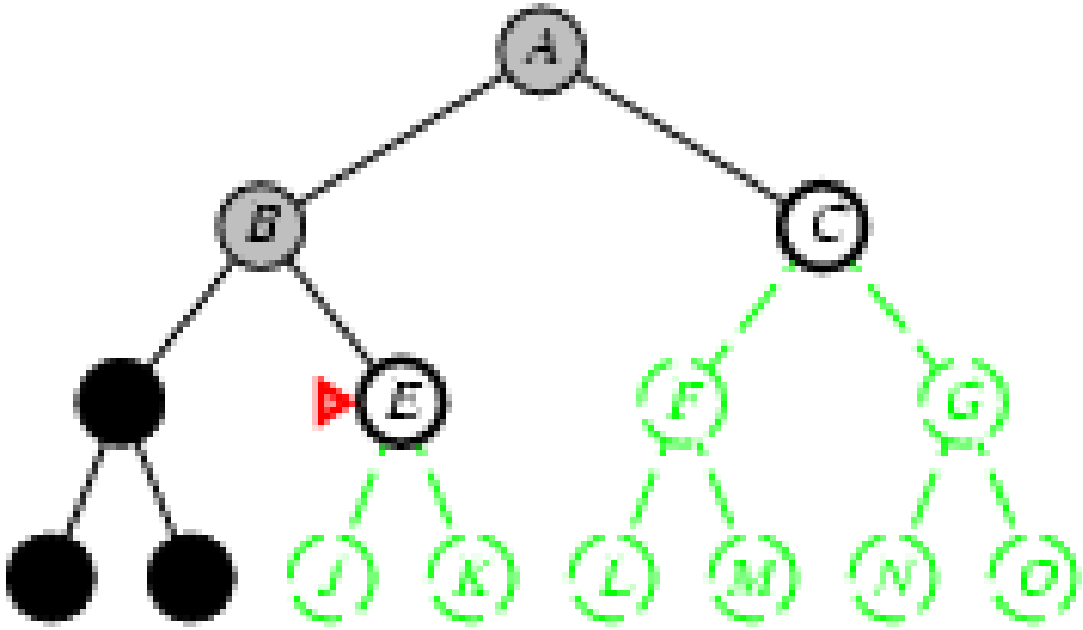
DFS

- Expand the deepest unexpanded node in frontier



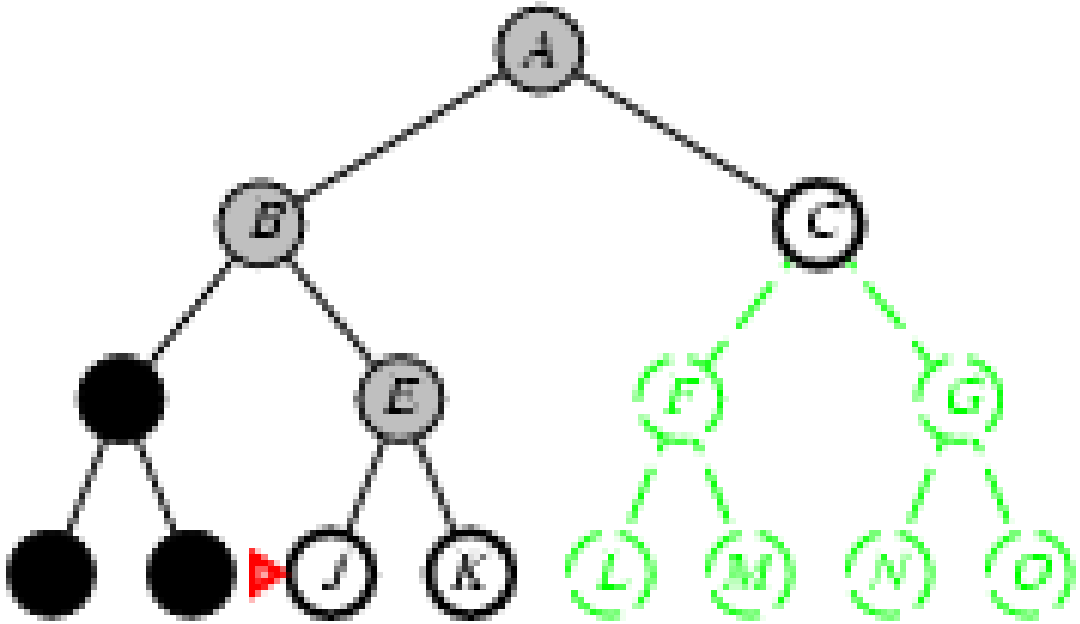
DFS

- Expand the deepest unexpanded node in frontier



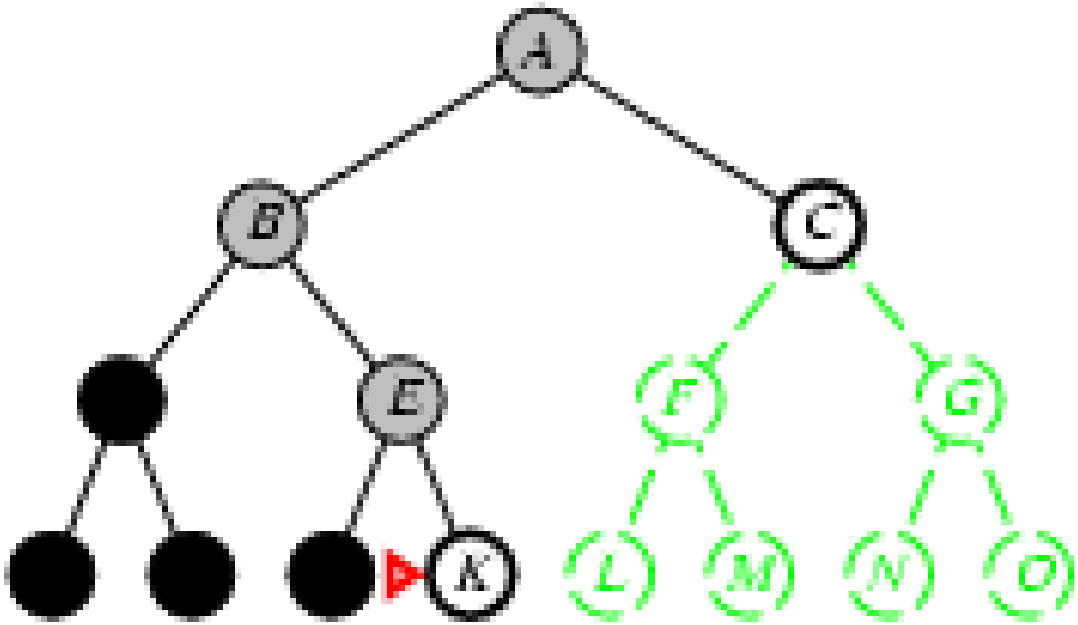
DFS

- Expand the deepest unexpanded node in frontier



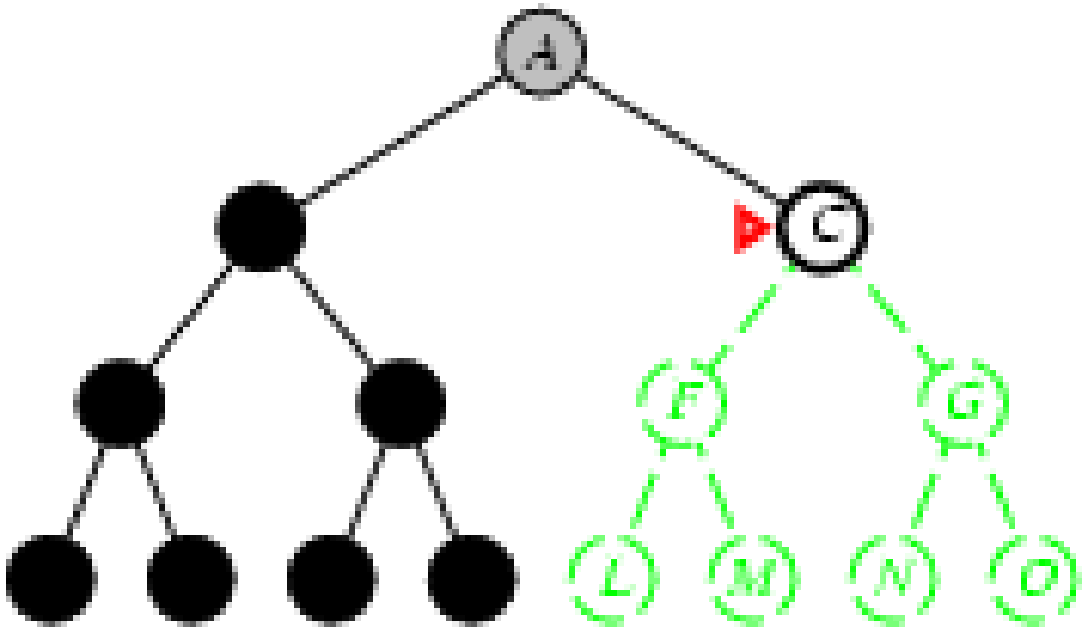
DFS

- Expand the deepest unexpanded node in frontier



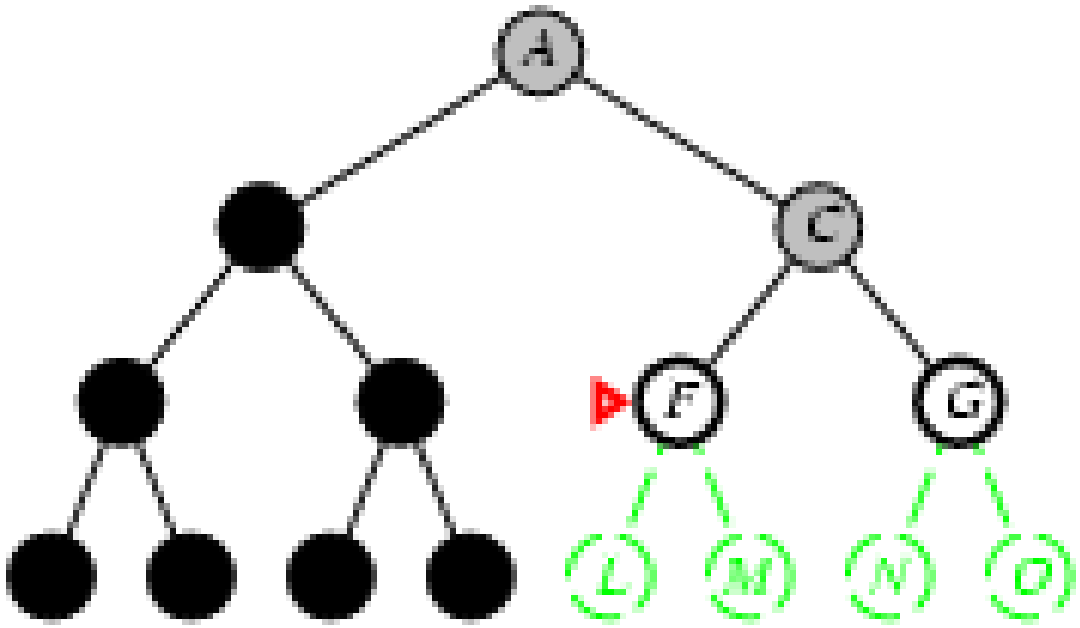
DFS

- Expand the deepest unexpanded node in frontier



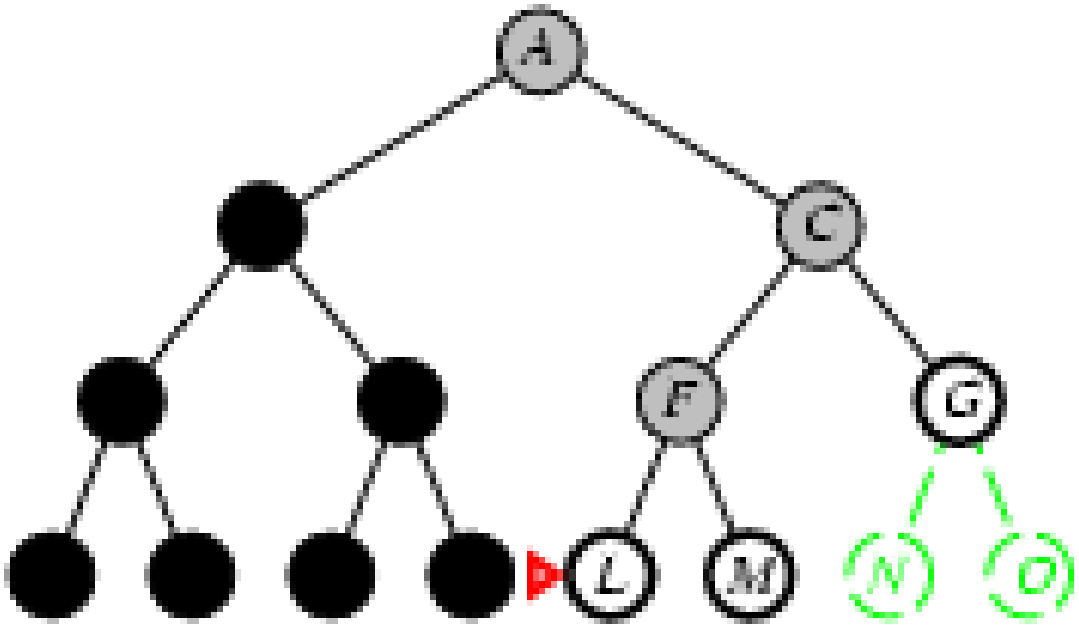
DFS

- Expand the deepest unexpanded node in frontier



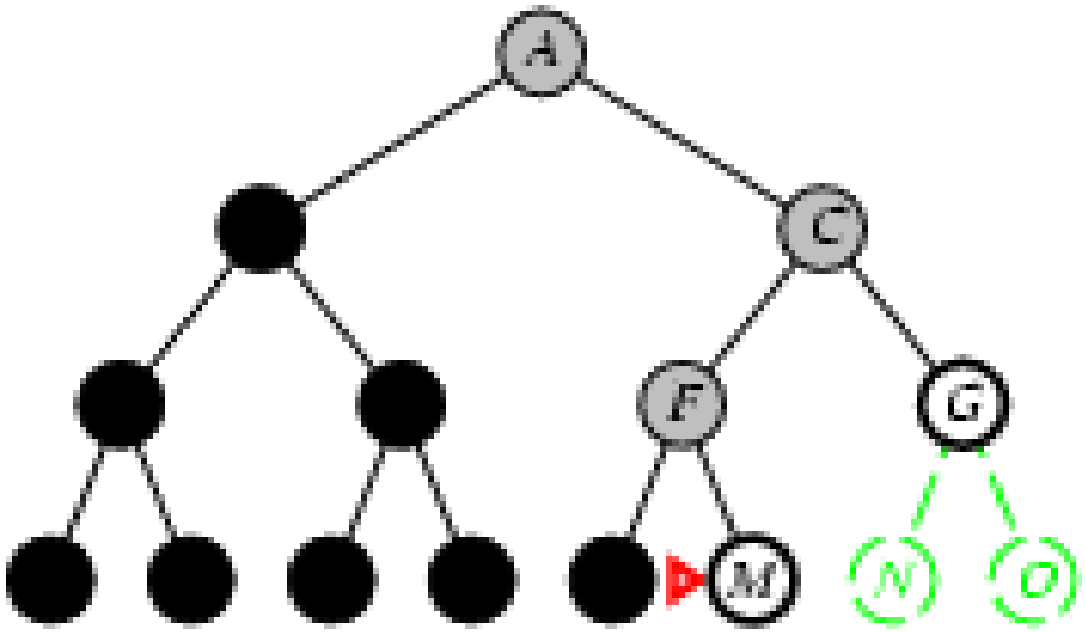
DFS

- Expand the deepest unexpanded node in frontier



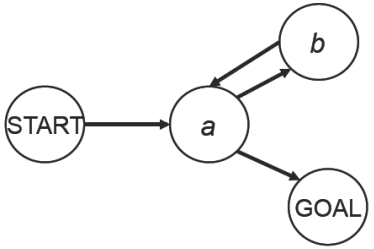
DFS

- Expand the deepest unexpanded node in frontier



Properties of DFS

- Complete?
  - Tree-search version: not complete (repeated states & redundant paths)
  - Graph-search version: fails in infinite state spaces (with infinite non-goal path) but complete in finite ones



- Time
  - $O(b^m)$: terrible if m is much larger than d
    - In tree-version, m can be much larger than the size of the state space
- Space
  - $O(bm)$, i.e., linear space complexity for tree search
    - So depth first tree search as the base of many AI areas
  - Recursive version called backtracking search can be implemented in $O(m)$ space
- Optimal?
  - No

DFS: tree-search version

Depth Limited Search

- Depth-first search with depth limit l (nodes at depth l have no successors)
  - Solves the infinite-path problem
  - In some problems (e.g., route finding), using knowledge of problem to specify l
- Complete?
  - If $l > d$, it is complete
- Time
  - $O(b^l)$
- Space
  - $O(bl)$
- Optimal?
  - No

Iterative Deepening Search (IDS)

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or failure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
```

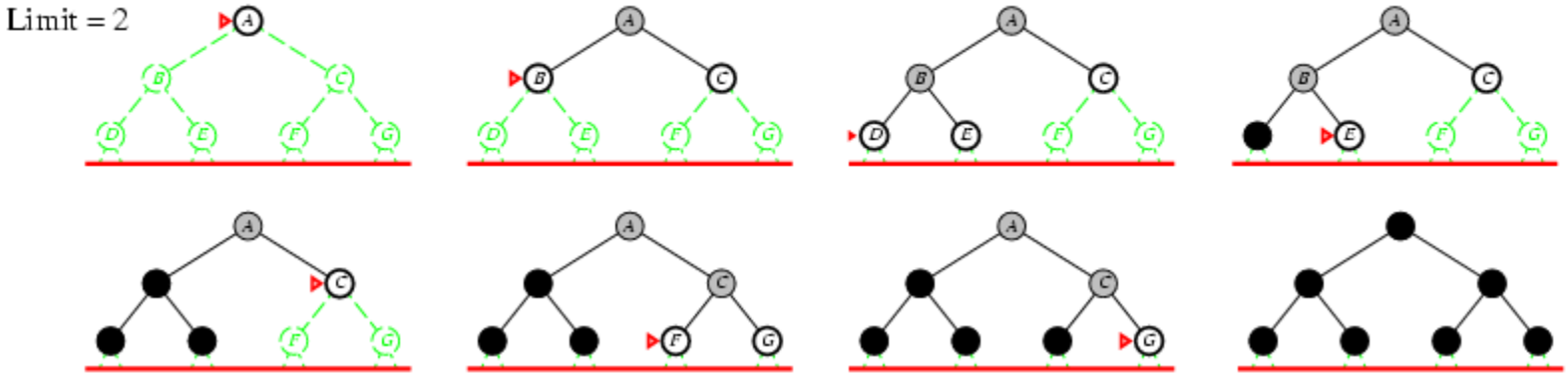
- Combines benefits of DFS & BFS
  - DFS: low memory requirement
  - BFS: completeness & also optimality for special path cost functions
- Not such wasteful (most of the nodes are in the bottom level)

IDS: Example l =0

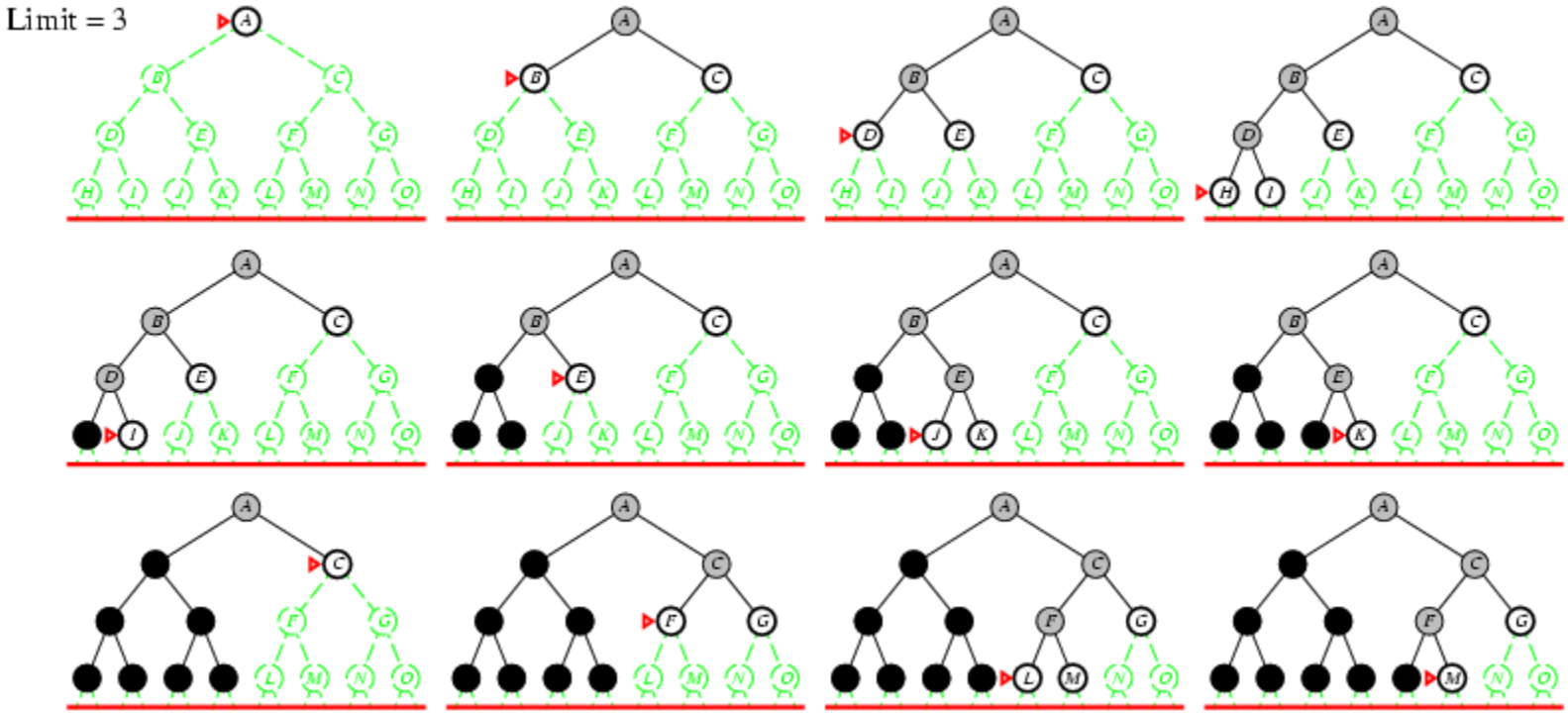
Limit = 0

IDS: Example $l = 1$

IDS: Example l =2



IDS: Example l =3



Properties of iterative deepening search

- Complete?
  - Yes (for finite b and d)
- Time
  - $d \times b^1 + (d-1) \times b^2 + \cdots + 2 \times b^{d-1} + 1 \times bd = O(b^d)$
- Space
  - $O(bd)$
- Optimal?
  - Yes, if path cost is a non-decreasing function of the node depth
- IDS is the **preferred method** when search space is large and the depth of solution is unknown

Iterative deepening search

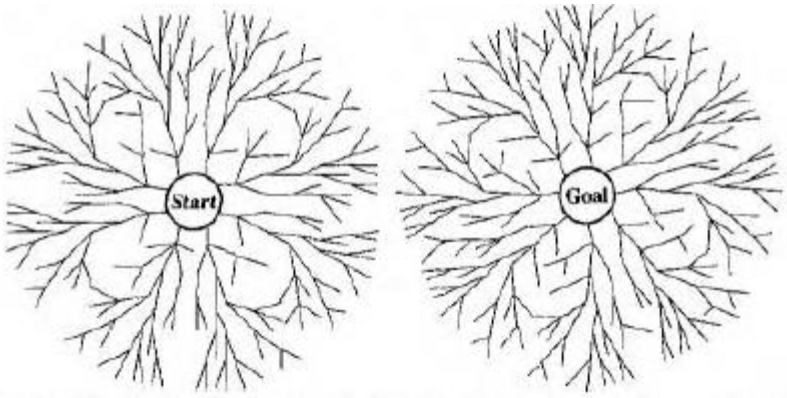
- Number of nodes generated to depth *d*:

$$N_{IDS} = d \times b^1 + (d-1) \times b^2 + \ \ldots + 2 \times b^{d-1} + 1 \times bd$$
$$= O(b^d)$$

- For $b = 10, d = 5$, we compute number of generated nodes:
  - N_{BFS} = 10 + 100 + 1,000 + 10,000 + 100,000 = 111,110
  - N_{IDS} = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450
  - Overhead of IDS = (123,450 - 111,110)/111,110 = 11%

Bidirectional search

- Simultaneous forward and backward search (hoping that they meet in the middle)
  - Idea: $b^{d/2} + b^{d/2}$ is much less than b^d
  - "Do the frontiers of two searches intersect?" instead of goal test
    - First solution may not be optimal
- Implementation
  - Hash table for frontiers in one of these two searches
    - Space requirement: most significant weakness
  - Computing predecessors?
    - May be difficult
  - List of goals? a new dummy goal
    - Abstract goal (checkmate)?!



Summary of algorithms (tree search)

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

a	Complete if *b* is finite
b	Complete if step cost ≥ ε>0
c	Optimal if step costs are equal
d	If both directions use BFS

- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms