Solving problems by searching

Uninformed search – ch 3,1-3.4 -- blind

Informed search – ch 3.5-3.6 – heuristic

Beyond classic search – ch 4

Adversarial search – ch 5

Outline

- Problem-solving agents
  - Definition
  - Problem types
  - Problem formulation
  - Example problems
- Basic search algorithms
  - BFS
  - Uniform-cost search
  - DFS
  - Depth-limited Search
  - Iterative Deepening Search

Problem-solving agents

- Motivating problem
  - Bob (an undergraduate student) needs to turn in his degree plan by tomorrow.
  - What should Bob do?

Problem-solving agents

- Motivating problem
  - Bob (an undergraduate student) needs to turn in his degree plan by tomorrow.
  - What should Bob do?
    - Deciding when to graduate, focused area, constraints on a CS degree plan, … ⇒ Goal Formulation
    - Find out what courses will be offered in future semesters, load of courses, … ⇒ problem Formulation
    - Assign courses to the first semester, second semester, … ⇒Search
    - Backtrack, if necessary.

Problem-solving agents components

- Goal formulation
  - Decide what to achieve.
- Problem formulation
  - Decide possible [action, state] to consider.
- Search
  - Generate a sequence of actions that can achieve the goal.

Example: Holiday tourist

- On holiday; currently in SA; goal is DC.
- Formulate goal:
  - in[DC]
- Formulate problem:
  - states: in various cities   in[SA]
  - actions: drive between cities

    {go[Houston], in[Houston]}, {go[Atlanta], in[Atlanta]}, …
- Find solution (search):
  - sequence of cities, e.g., SA, Houston, Atlanta, DC.

Problem types

- Deterministic, fully observable → single-state problem
  - Agent knows exactly which state it will be in.
- Non-observable → sensorless problem
  - Agent may have no idea where it is.
- Nondeterministic and/or partially observable → contingency problem
  - percepts provide new information about current state
  - often interleave search and execution
- Unknown state space → exploration problem
  - The number of situations to be considered increases exponentially as the number of look-ahead steps increase.
  - This problem motivates the need for heuristic search methods.
  - Heuristic search uses heuristics to guide the search process to focus only on situations that deserve attentions.
  - Also called “inform search” or “use good guess”.

(Step 1)Single-state problem formulation

- A problem is defined by four items:
  - State in general and initial state e.g., in[SA]
  - successor function *S(x)* = set of action – state pairs

    e.g., *S(SA) =* {go[Houston], in[Houston]}, {go[Austin], in[Austin]},…
  - goal test, could be
    - explicit, e.g., g=in[DC].
    - implicit, e.g., a winning check state.
  - path cost

    e.g., sum of distances, number of actions executed, etc.
- A solution is a sequence of actions leading from the initial state to a goal state
- Solution quality is measured by the path cost function
- An optimal solution has the lowest path cost among all solutions

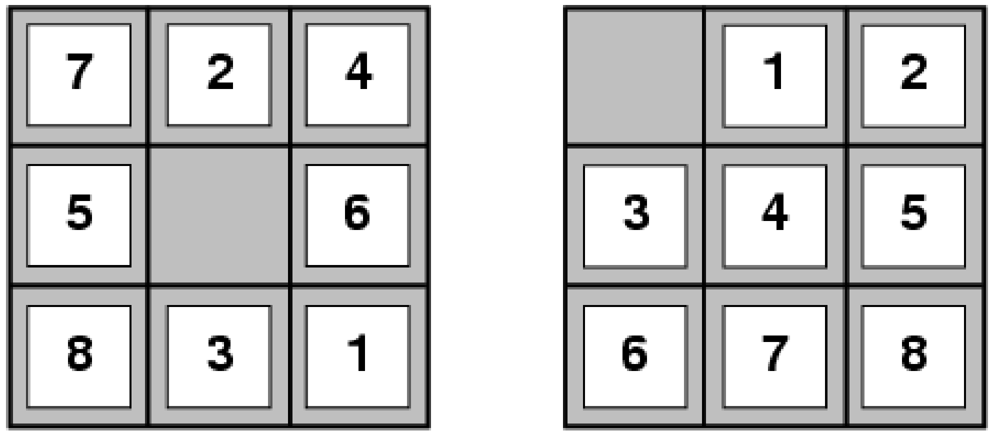
Example: Vacuum World

- states
- actions
- goal
- path cost

Example: Vacuum World

- states dirt and robot location
- actions *Left*, *Right*, *Suck*
- goal no dirt at all locations
- path cost 1 per action

Example: The 8-puzzle

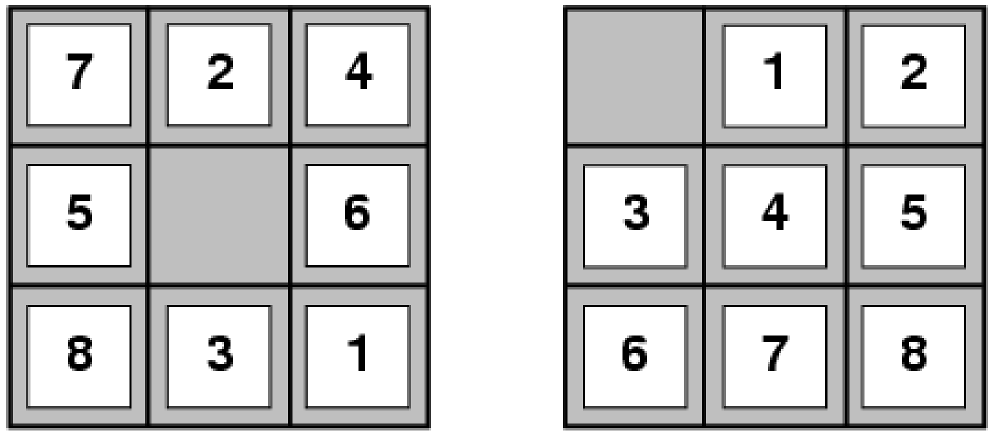


Start State Goal State

- states
- actions
- goal
- path cost

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

Example: The 8-puzzle



Start State Goal State

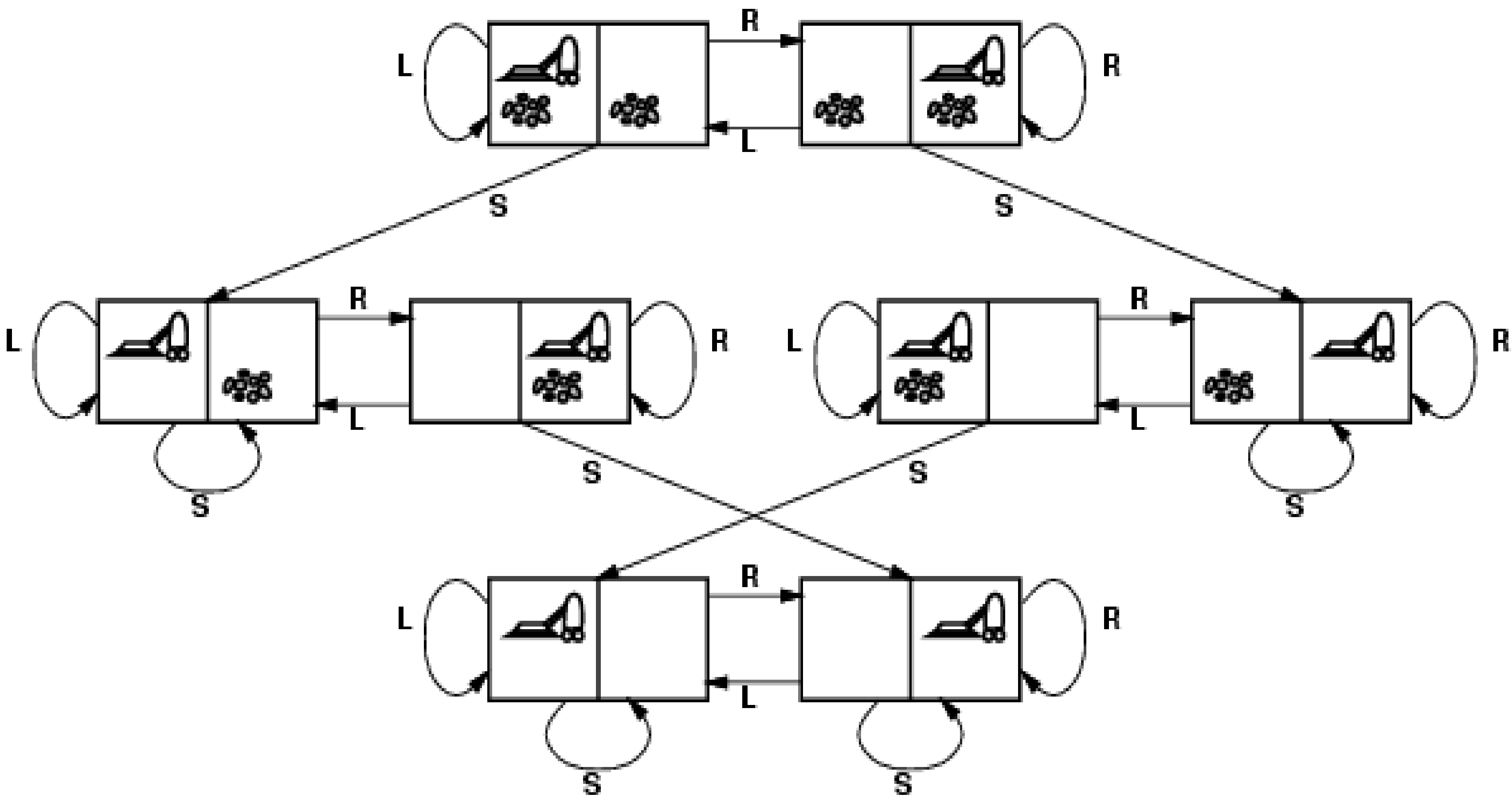
- states locations of tiles
- actions move blank left, right, up, down
- goal given
- path cost 1 per move

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

(2) Selecting a state space

- State space
  - The set of all states reachable from the initial state by performing actions.
- Real world is complex
  - State space must be abstracted for problem solving (removing details from representation).
  - e.g., “SA → Houston" represents a complex set of possible routes, detours, rest stops, etc.
- For guaranteed reachability, any real initial state (in SA) must get to some real goal state (in DC).
- Each abstract action should be easier than the original problem.

Vacuum world state space graph



(Step 3) Searching for Solutions

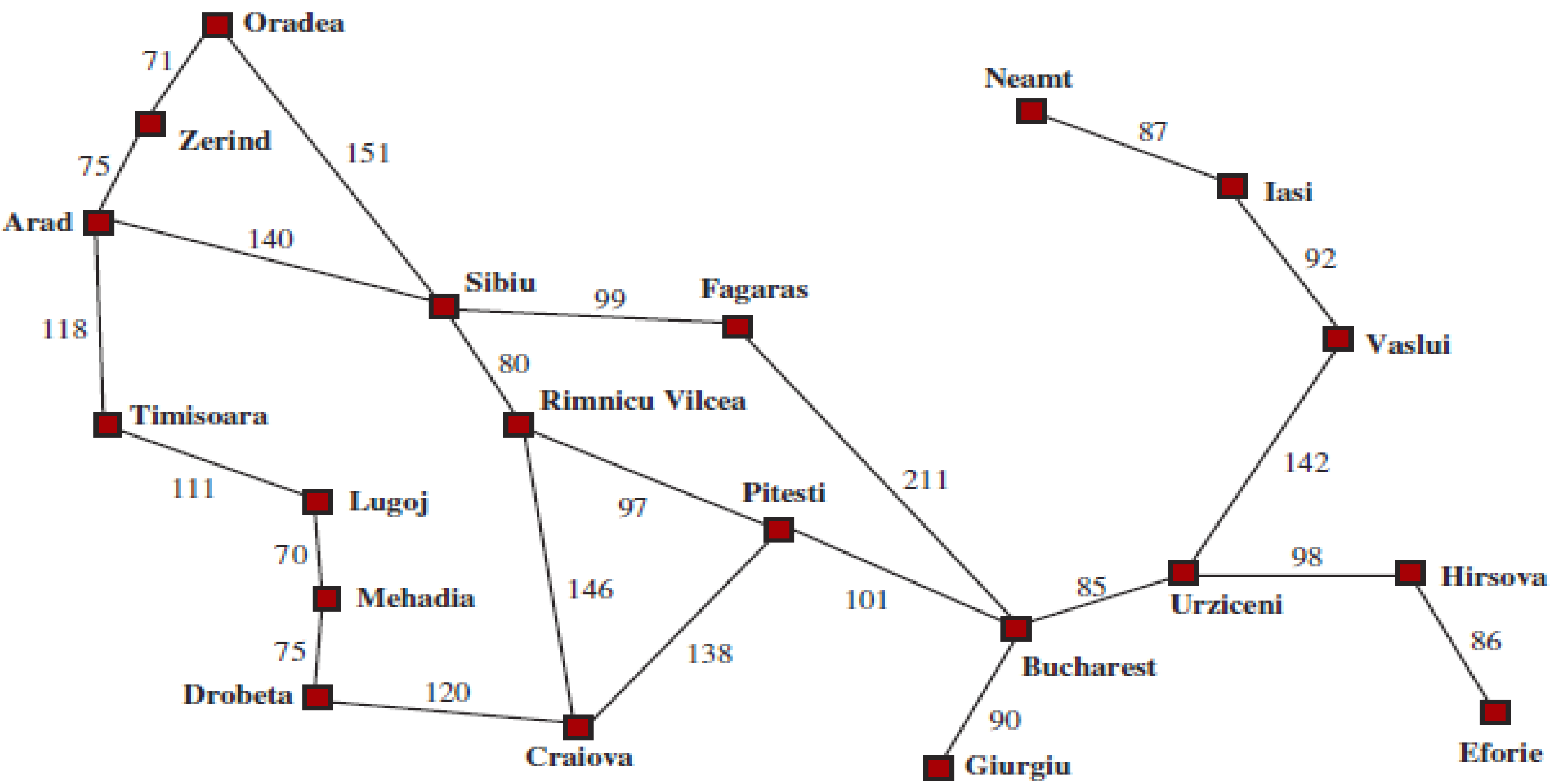
- Search tree
  - Basic idea: simulated exploration of state space by generating successors of already-explored states.

An informal general tree search algorithm

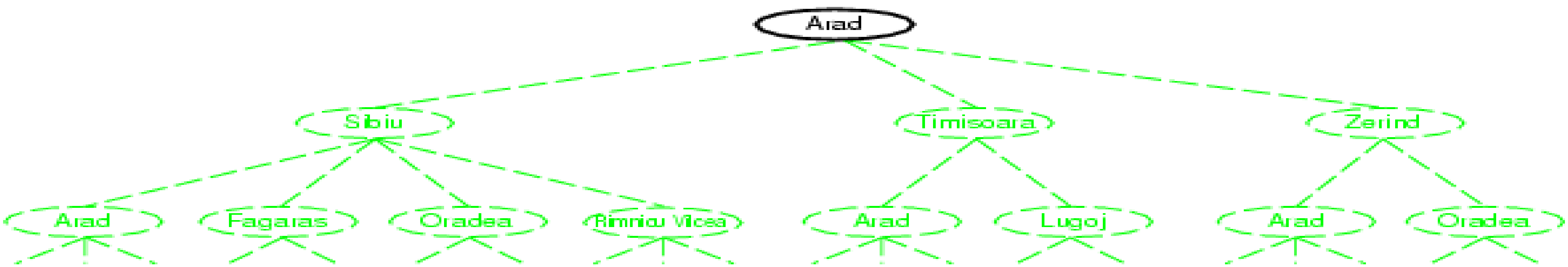
```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node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

Road Map of Part of Romania

Find a route from Arad to Bucharest

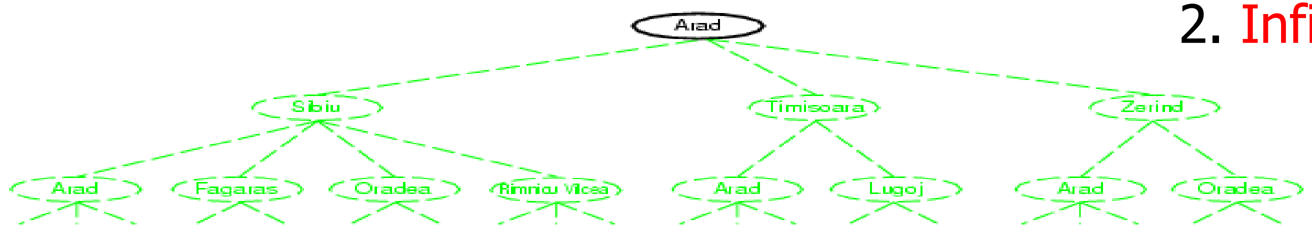


Tree search example

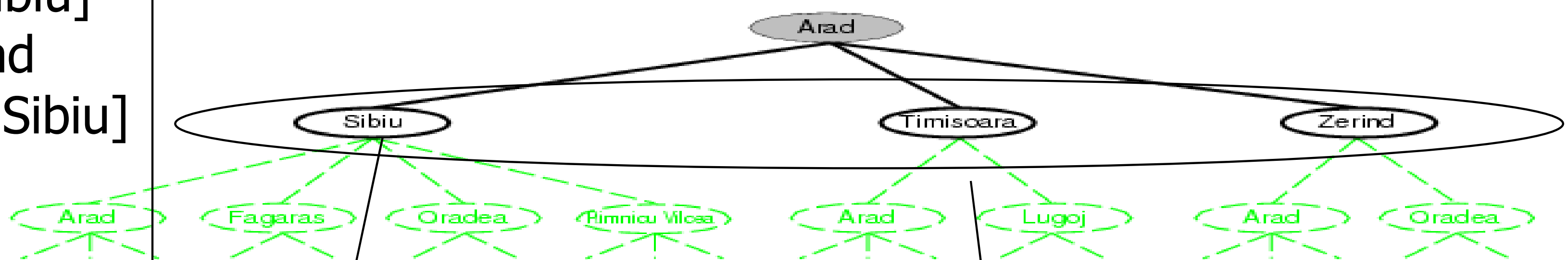


Tree search example

Characteristics
1. Repeated states
2. Infinite tree

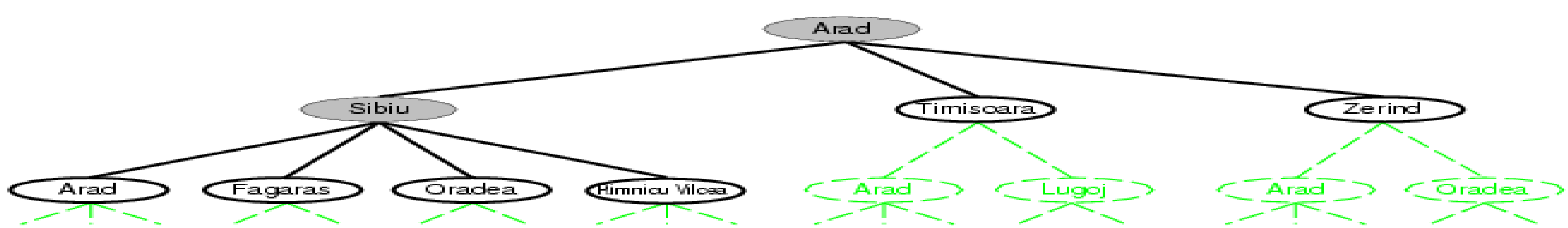


State: in[Sibiu]
Parent: Arad
Action: go[Sibiu]
Cost: 1
Depth: 1



Leaf node

Fringe: the collection of nodes that have been generated but not yet expanded.



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.
- Expand a node
  - *Creates new children nodes*.
  - Filling in the various fields.
  - Create the corresponding states.

Search strategies

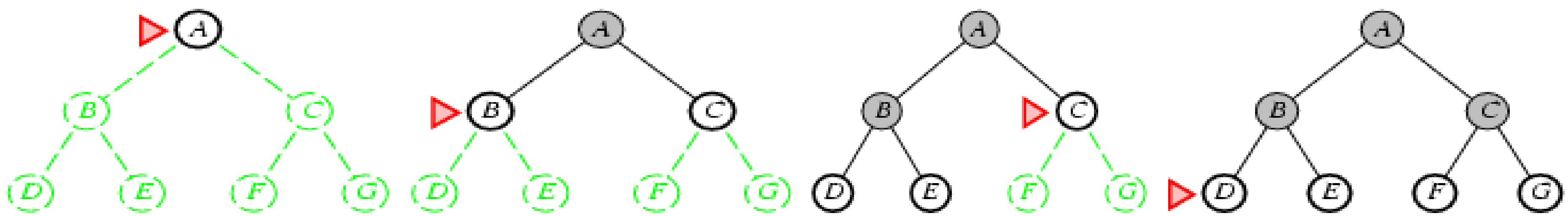
- A search strategy is defined by picking the order of node expansion.
  - Generating new states by applying operators to existing states until the goal state is reached.
- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated (in worst-average case). Denoted by Big O.
  - space complexity: how much memory is required? Denoted by Big O.
  - optimality: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - *b:* maximum number of successor of any node (**branching factor)**
  - *d:* depth of the least-cost solution
  - *m*: maximum depth of the state space (may be ∞)

Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition (also called blind search)
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

Breadth-first search

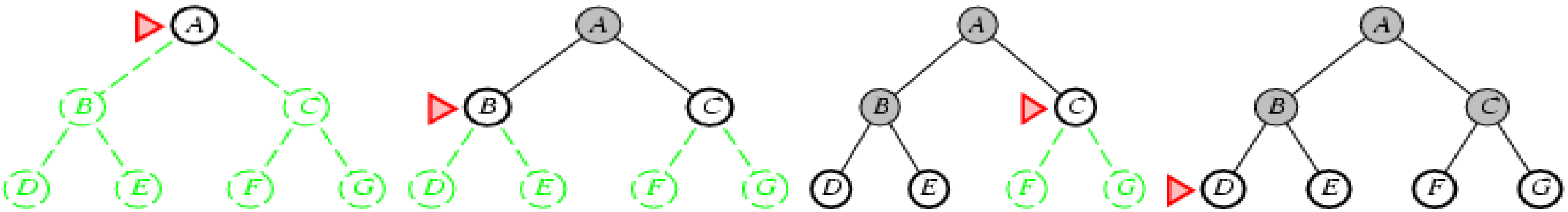
- Expand shallowest unexpanded node (Guided by depth)
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end



- Complete?
- Time?
- Space?
- Optimal?

Breadth-first search

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

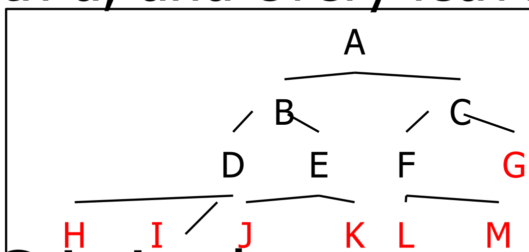


- Complete? Yes (if b is finite). Shallowest goal.
- Time? $1+b+b^2+b^3+\ldots+b^d+(b^{d+1}-b) \approx O(b^{d+1})$
- Space? $b^{d+1}-b+1 \approx O(b^{d+1})$
- Optimal? Yes (if all step costs are equal or non-decreasing). Shallowest goal.

BFS: Space Complexity

- Worse case (G is in the right-most side of depth d)
  - keep G at depth d, and every leave node at depth d+1)

A
BC
CDE
DEFG
EFGHI
FGHIJK
GHIJKLM

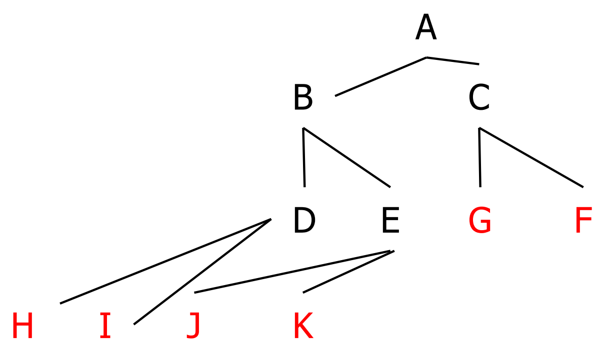


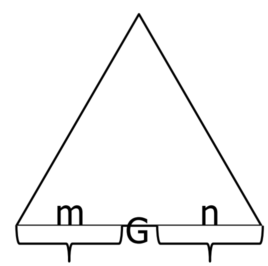
Total Space Complexity

1 +
$(b^{d+1}-b)$

- General case (G is in the somewhere of depth d)
  - G, keep all the nodes on right of G in depth d, at depth d+1 keep all the children of "the nodes on left of G in depth d"

A
BC
CDE
DEGF
EGFHI
GFHIJK





$1+n+m\times b$

G　nodes on right of G　children of the nodes on left of G

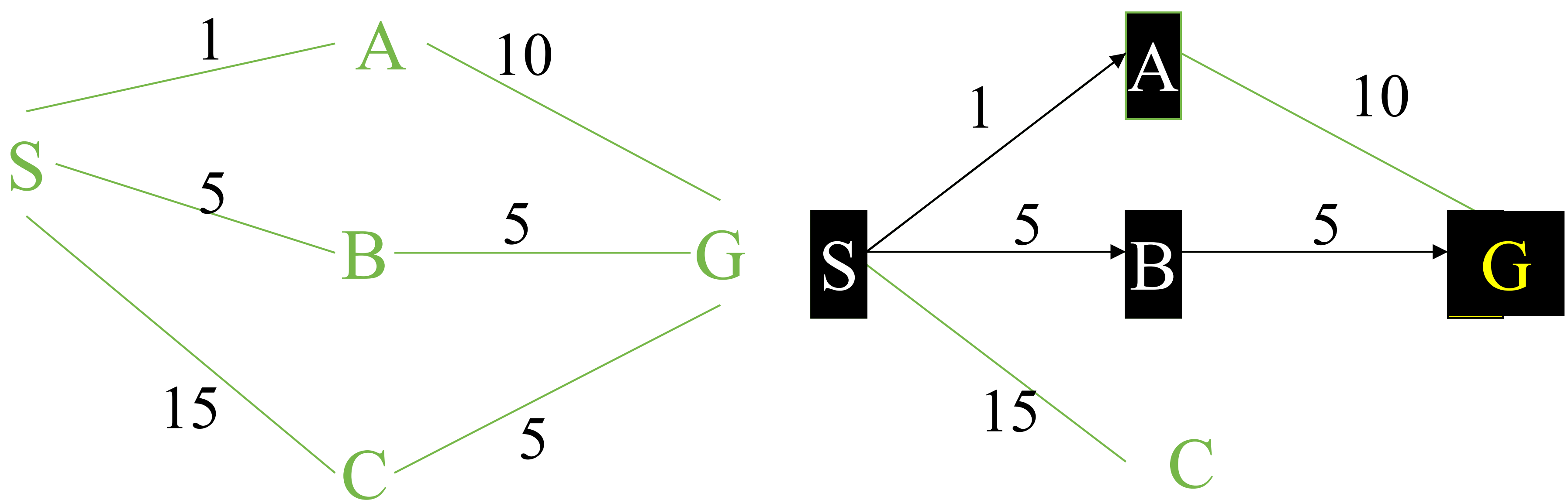
Breadth-first search

- Memory requirements are a bigger problem.
  - 1,100 nodes can be generated in 0.11 seconds, but each node need 1k bytes of storage. Total 1 megabyte.
- Time requirements are still a major factor.
  - Exponential-complexity search cannot be solved by uninformed methods for any but the smallest instances.
  - B=10, d=12, will take 35 years for any uninformed search ($O(10^{13})$).

Uniform-cost search

- Expand least-cost unexpanded node, where cost is the path cost, *g(n)*.
- Guided by path costs.
- Equivalent to BFS if g(n)=depth(n), i.e. step costs all equal.
- Implementation:
  - *fringe* = queue ordered by path cost (priority queue)

Uniform-cost search



We wish to find the shortest route from node S to node G; that is, node S is the initial state and node G is the goal state. In terms of path cost, we can clearly see that the route *SBG* is the cheapest route. However, if we let breadth-first search loose on the problem it will find the non-optimal path *SAG*, assuming that A is the first node to be expanded at level 1.

Uniform-cost search

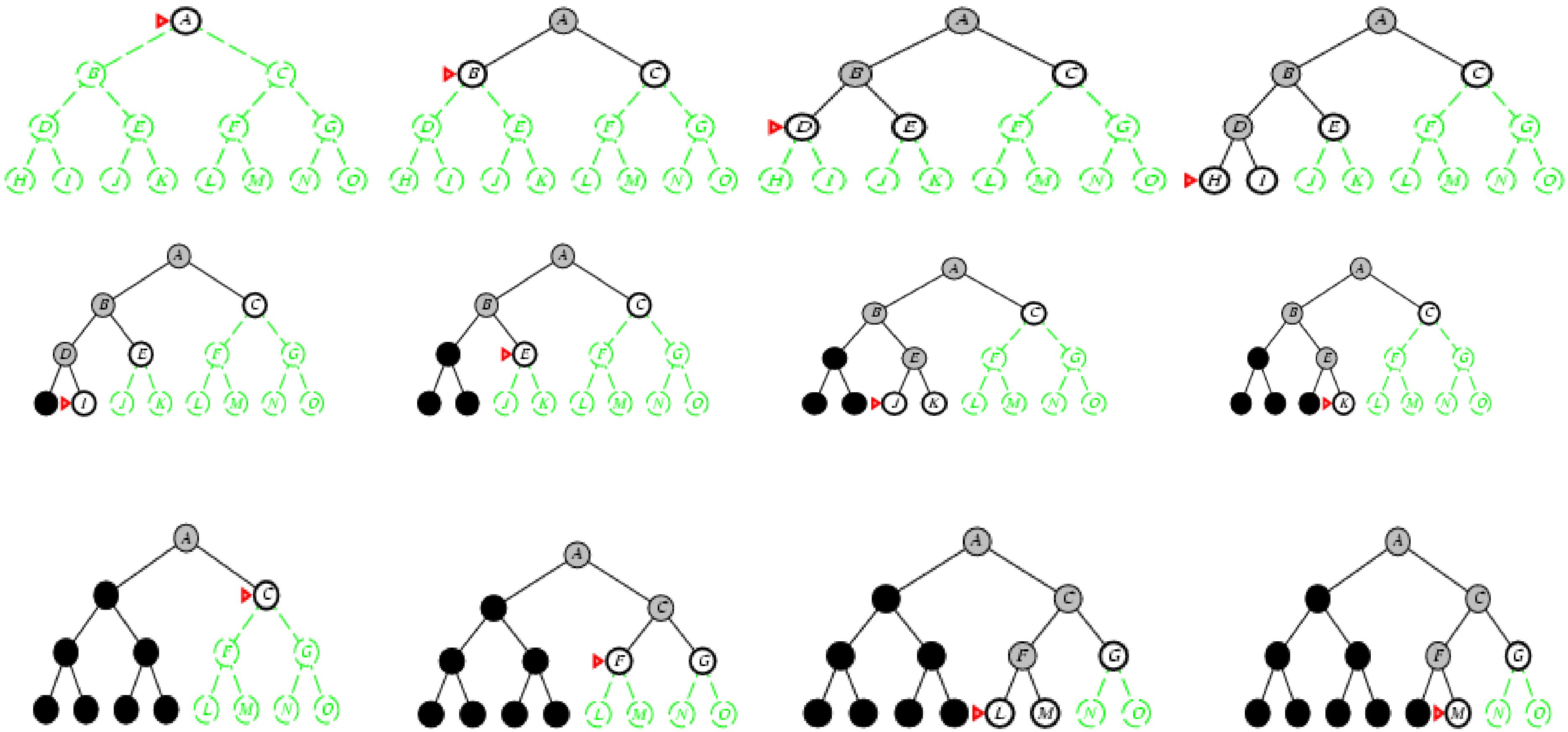
Complete?
Time?
Space?
Optimal?

Uniform-cost search

Complete? Yes, if step cost ≥ ε, a lower bound on the cost of each action.
Time? # of nodes with *g* ≤ cost of optimal solution, $O(b^{C^*/\varepsilon})$ where C^* is the cost of the optimal solution. When all step costs are equal, *then* $O(b^{C^*/\varepsilon}) = O(b^d)$, *meaning UCS=BFS.*
Space? # of nodes with *g* ≤ cost of optimal solution, $O(b^{C^*/\varepsilon})$
Optimal? Yes – nodes expanded in increasing order of $g(n)$.

Depth-first search

- Expand deepest unexpanded node in the current fringe
- Implementation:
  - *fringe* = LIFO stack, i.e., put successors at front



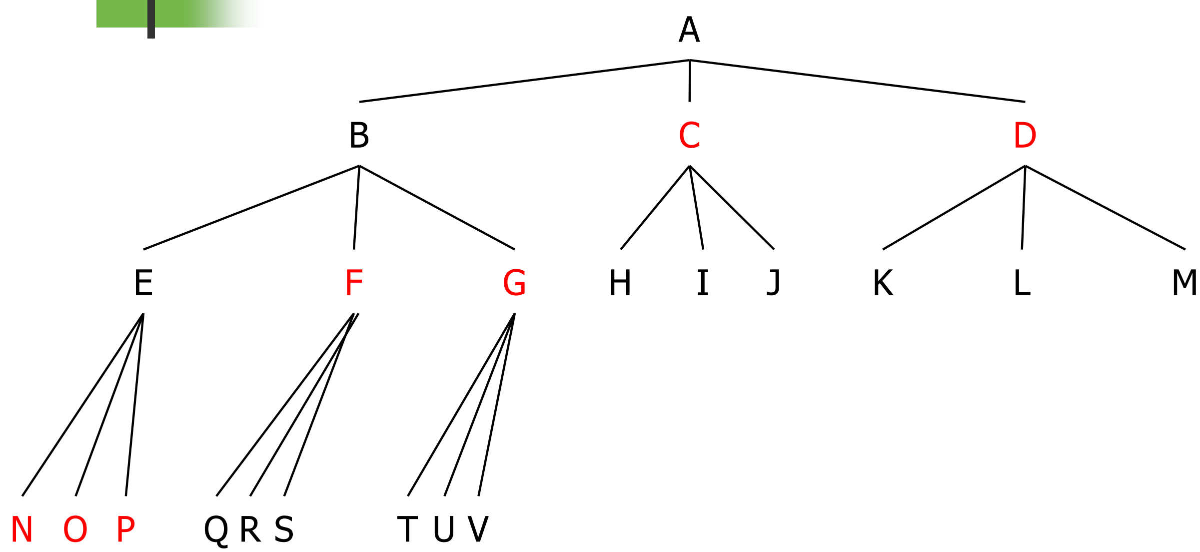
Properties of depth-first search

- Complete?
- Time?
- Space?
- Optimal?

Properties of depth-first search

- Complete? No: fails in infinite-depth spaces, spaces with loops
  - complete in finite tree
- 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. (if the left subtree were of unbounded depth but contained no answer, DFS would not terminated)

DFS: Space Complexity



b-1 +

b-1 +

.

.

. +

b

(m-1) times

			P							
			O	O			S			
		E	N	N	N		R	R		
		F	F	F	F	F	Q	Q	Q	
	B	G	G	G	G	G	G	G	G	**G**
	C	C	C	C	C	C	C	C	C	C
A	D	D	D	D	D	D	D	D	D	D

Total Space Complexity

$$1 + (m-1) \times (b-1) + b$$

$$= O(bm)$$

BFS vs. DFS

- Time complexity
  - If goal nodes are much shallower than the depth of the tree, **?** is better.
  - If the goal nodes exist in the left portion of the tree, **?** is better.
  - What if the goal nodes exist in the rightmost and deepest portion of the tree? **?** is better.
- Space complexity (assuming goal nodes found are at depth d of the tree)
  - **BFS requires b^{d+1} memory**
  - **DFS requires bd memory**
- Completeness/Optimality
  - **BFS is better.**

Depth-limited search

=depth-first search with depth limit *l* (nodes at *l* have no successors)

- Complete?
- Time?
- Space?
- Optimal?

Depth-limited search

=depth-first search with depth limit L (nodes at L have no successors)

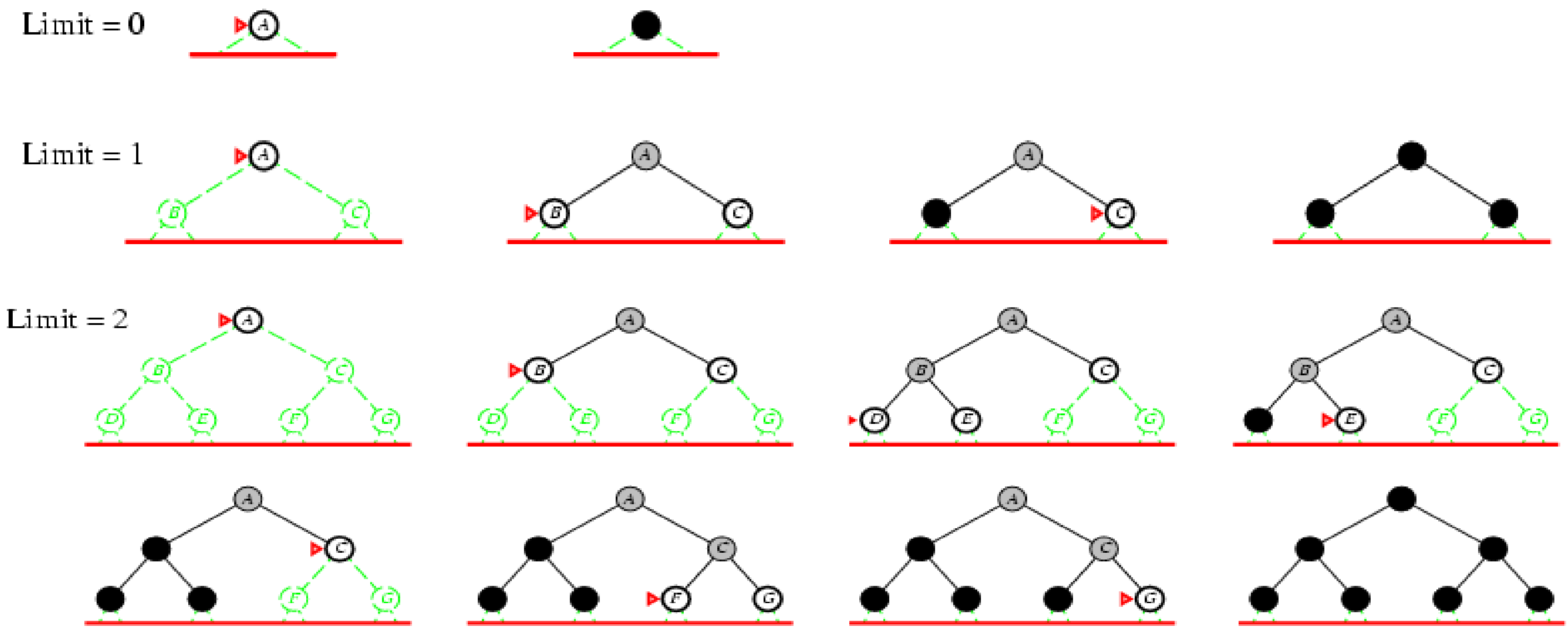
Fail if L<d.

- Complete? No (if L<d)
- Time? $1+b+b^2+b^3+\ldots+b^L = O(b^L)$
- Space? $O(bL)$
- Optimal? No

Iterative deepening search

- =DFS + depth limited search
- Gradually increase the bound on depth limited search
  - DLS with limit = 0
  - DLS with limit = 1
  - DLS with limit = 2
  - …
  - Until a solution is found
- Implementation:
  - *fringe* = LIFO stack

Iterative deepening search



b^0

b^0+b^1

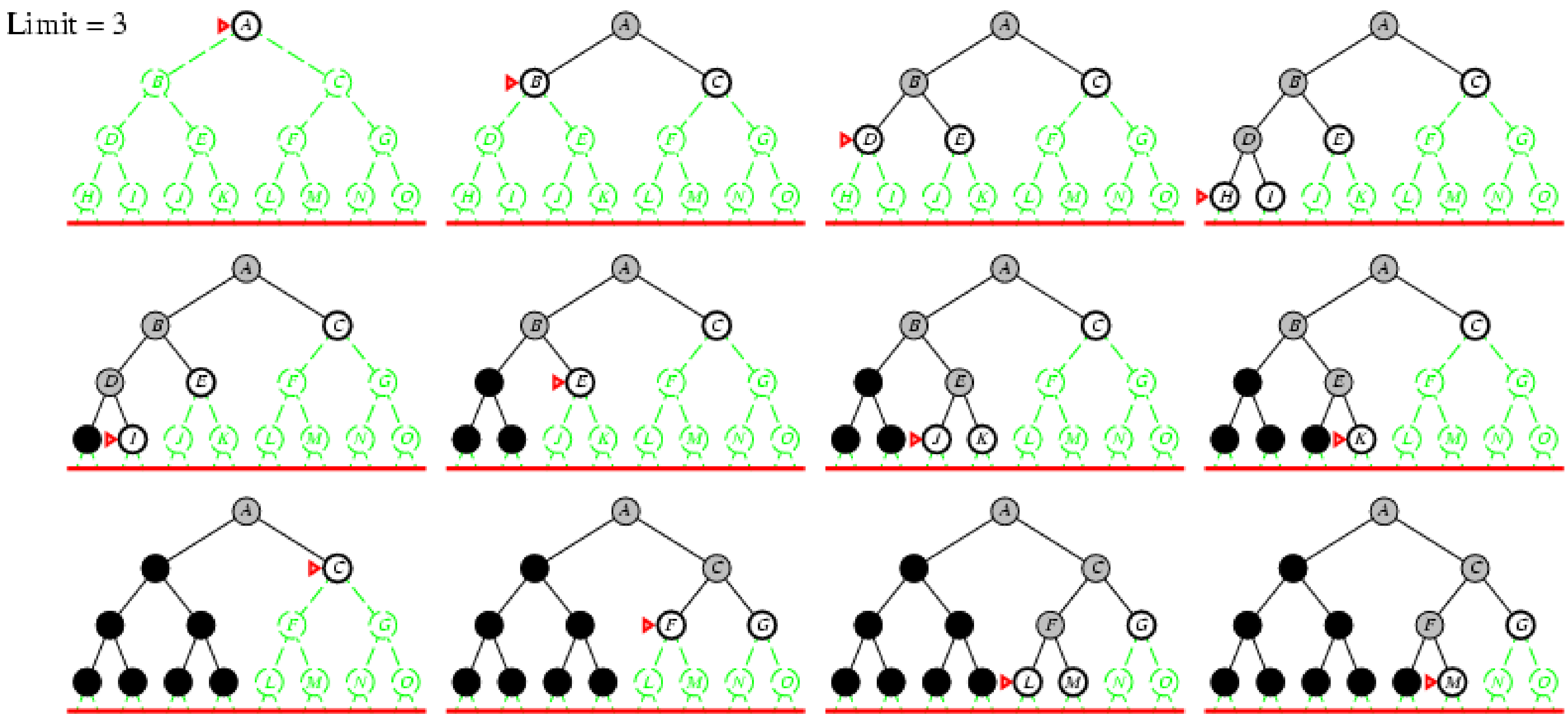
$b^0+b^1+b^2$

.
.
.

Limit = d

$b^0+b^1+b^2+\ldots+b^d$

Iterative deepening search



Properties of iterative deepening search

- Complete?
- Time?
- Space?
- Optimal?

Properties of iterative deepening search

- Complete? Yes (shallowest goal by DFS)
- Time? $(d+1)b^0 + db^1 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if all step costs are equal or non-decreasing. Find shallowest goal because it explores a complete layer of new nodes at each iteration before going on to the next layer.
- IDS is preferred when there is a large search space and m is unknown.

  $b=10 \quad d=5$

  $O(b^d)$ $N(IDS)=6\times1+5\times10+4\times10^2+3\times10^3+2\times10^4+1\times10^5=123{,}456$

  $O(b^{d+1})$ $N(BFS)=1+10+10^2+10^3+10^4+10^5+10^6-10=1{,}111{,}101$

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

If all step costs are equal or non-decreasing

Summary (why need search with partial info)

- What assumption are made by a problem-solving agent?
  - The ENV should be static, deterministic, fully-observable (Single state)
- What if these assumption are violated?
  - Belief state replaced by Env state
  - Informed search algorithms (Search with partial info)