CSC421 Intro to Artificial Intelligence

UNIT 02:
Solving Problems by Searching

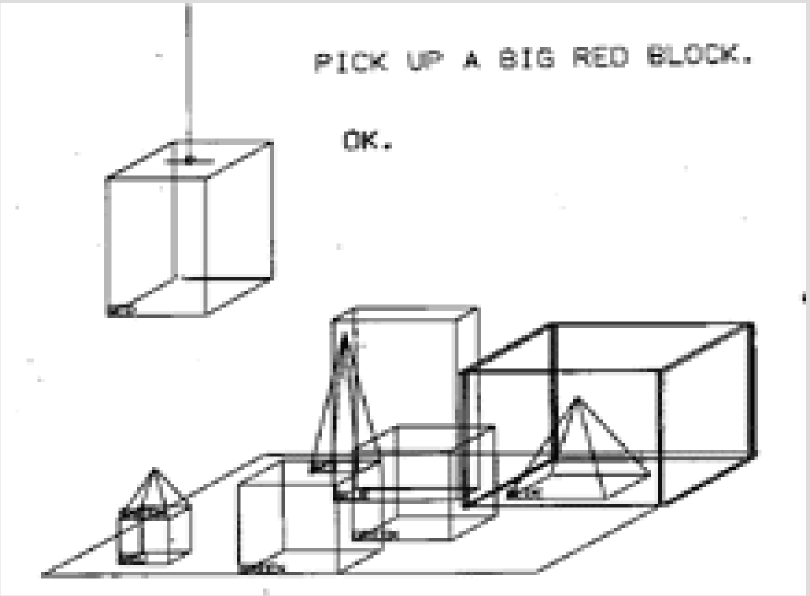
Search problems

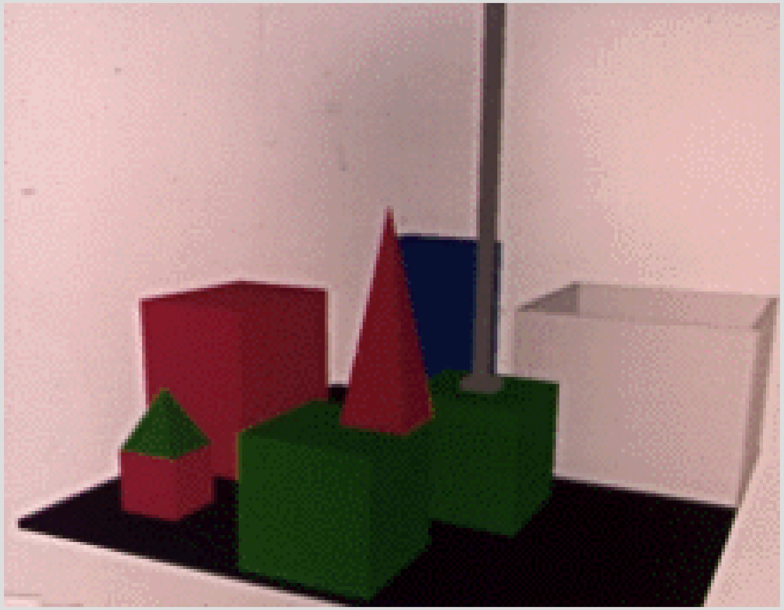
- Blind (no information about problem other than it's definition)
- Informed (heuristic) search (next lecture)
- Examples of search problems:
  - Board games
  - Scheduling
  - Theorem proving
  - Robot Navigation
  - Combat NPC in game
- The workhorse of old classic AI
- Works well for “toy words”

Sidenote I

- Search works really well in “Microworlds” or “toy worlds”
- SAINT (James Stagle – 1963) Closed-form calculus integration problems
- ANALOGY (Tom Evans-1968) Geometry analogy problems that appear in IQ test
- Blocks world (vision, constraint-propagation, learning, natural language understanding)

Sidenote II





Sidenote III

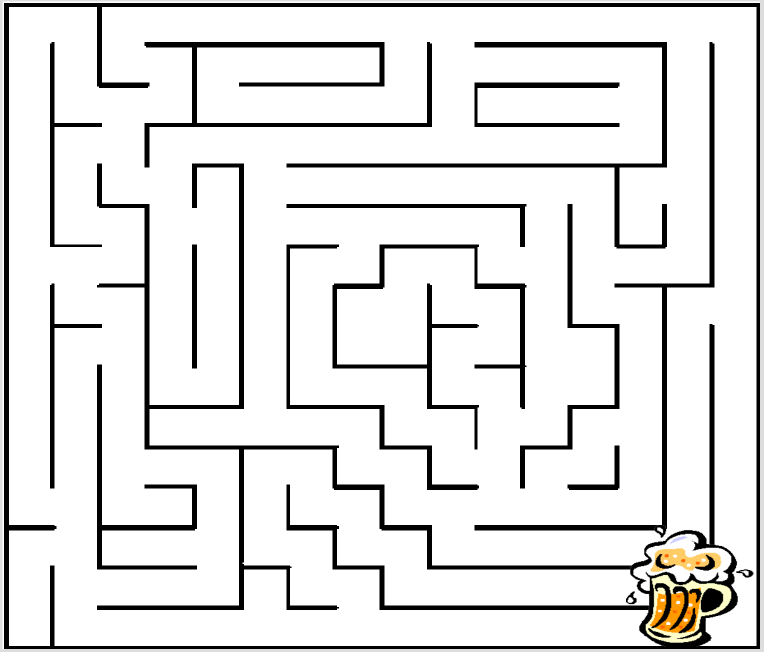
Person: PICK UP A BIG RED BLOCK.
Computer: OK. (does it)
Person: GRASP THE PYRAMID.
Computer: I DON'T UNDERSTAND WHICH PYRAMID YOU MEAN.
Person: FIND A BLOCK WHICH IS TALLER THAN THE ONE YOU ARE HOLDING AND PUT IT INTO THE BOX.
Computer: BY "IT", I ASSUME YOU MEAN THE BLOCK WHICH IS TALLER THAN THE ONE I AM HOLDING.
Computer: OK. (does it)
Person: WHAT DOES THE BOX CONTAIN?
Computer: THE BLUE PYRAMID AND THE BLUE BLOCK.
Person: WHAT IS THE PYRAMID SUPPORTED BY?
Computer: THE BOX.
Person: HOW MANY BLOCKS ARE NOT IN THE BOX?
Computer: FOUR OF THEM.

Searching is limited by

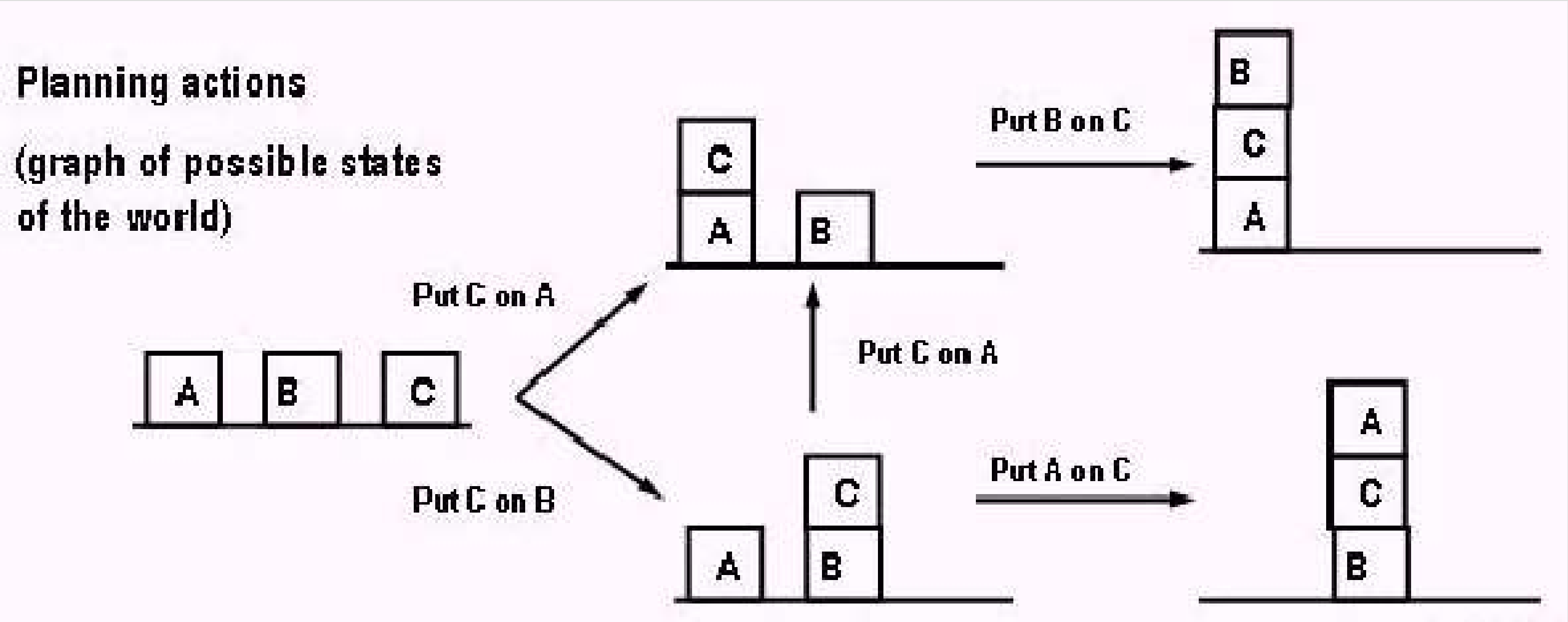
- Precise formulation of problem
- Typically
  - Static
  - Deterministic
  - Offline
- No learning
- Hard to deal with uncertainty
- Easier to implement and understand
- Therefore cover the corresponding material relatively quickly

Formulation: the main challenge

- Formulate-goal
  - Discover the scroll of infinite AI wisdom
- Formulate-problem
  - States: Grid locations
  - Actions: NW, N, NE, W, E, SW, S, SE
- Formulate-solution
  - Sequence of grid locations (states)

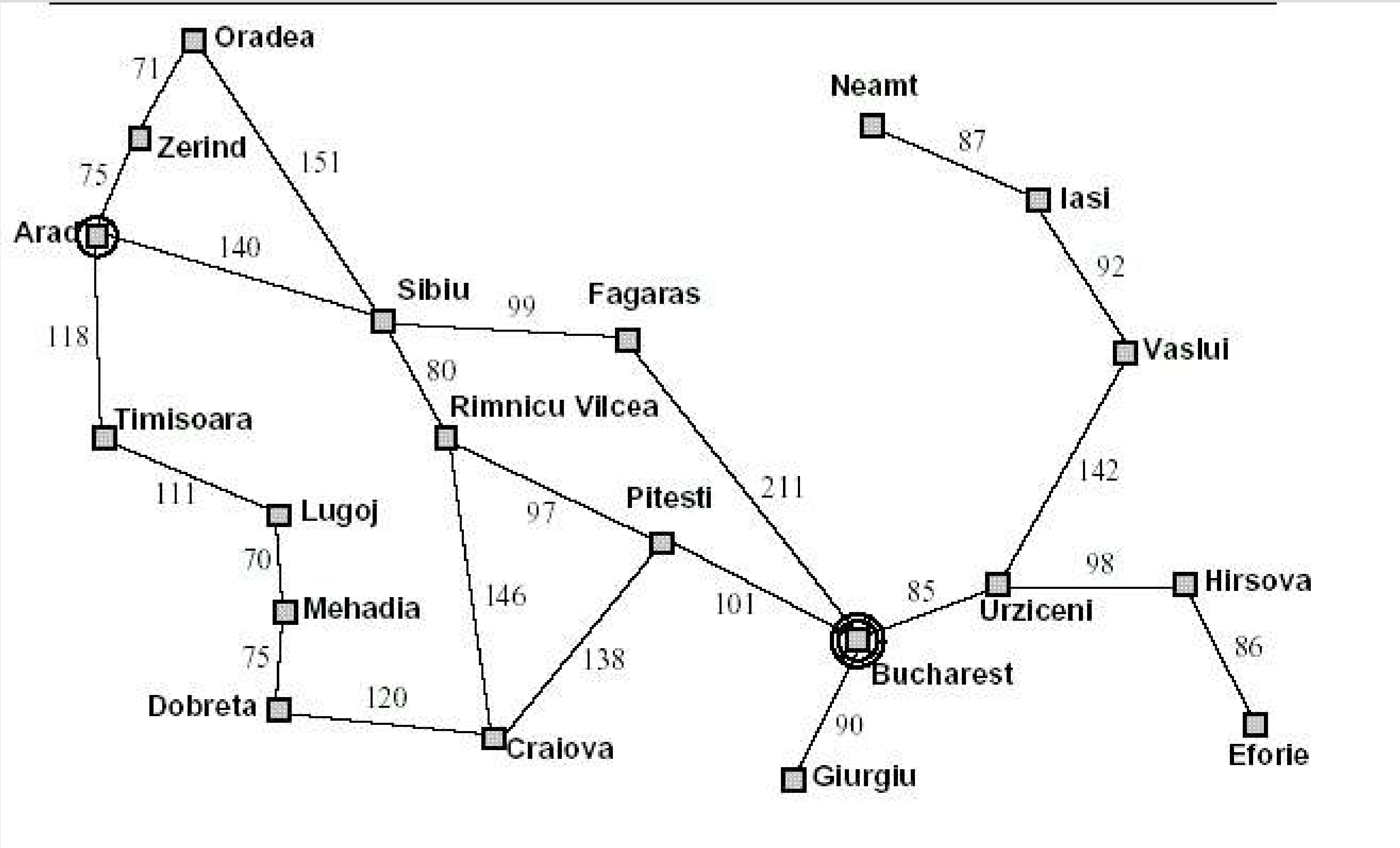


Graphs are your friend



- A **path** through such a graph (from a start node to a goal node) is a "plan of action" to achieve some desired goal state from some known starting state.
- Turn AI problems into graphs and you are almost done

Map example


Oradea
Zerind
Arad
Timisoara
Lugoj
Mehadia
Dobreta
Sibiu
Rimnicu Vilcea
Fagaras
Pitesti
Craiova
Bucharest
Giurgiu
Urziceni
Hirsova
Eforie
Vaslui
Iasi
Neamt
71
75
151
140
118
111
70
75
120
99
80
97
146
138
101
211
85
90
98
86
142
92
87

Single state-space formulation

A problem is defined by four items:

Initial state e.g., "at Arad"

Actions and successor function *S:* = set of **action-state** tuples

– e.g., *S(Arad)* = {*(goZerind, Zerind), (goTimisoara, Timisoara), (goSilbiu, Silbiu)*}

Goal test, can be

– explicit, e.g., x = "at Bucharest"
– implicit, e.g., *Checkmate(x)*

Path cost (additive)

– e.g., sum of distances, or 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

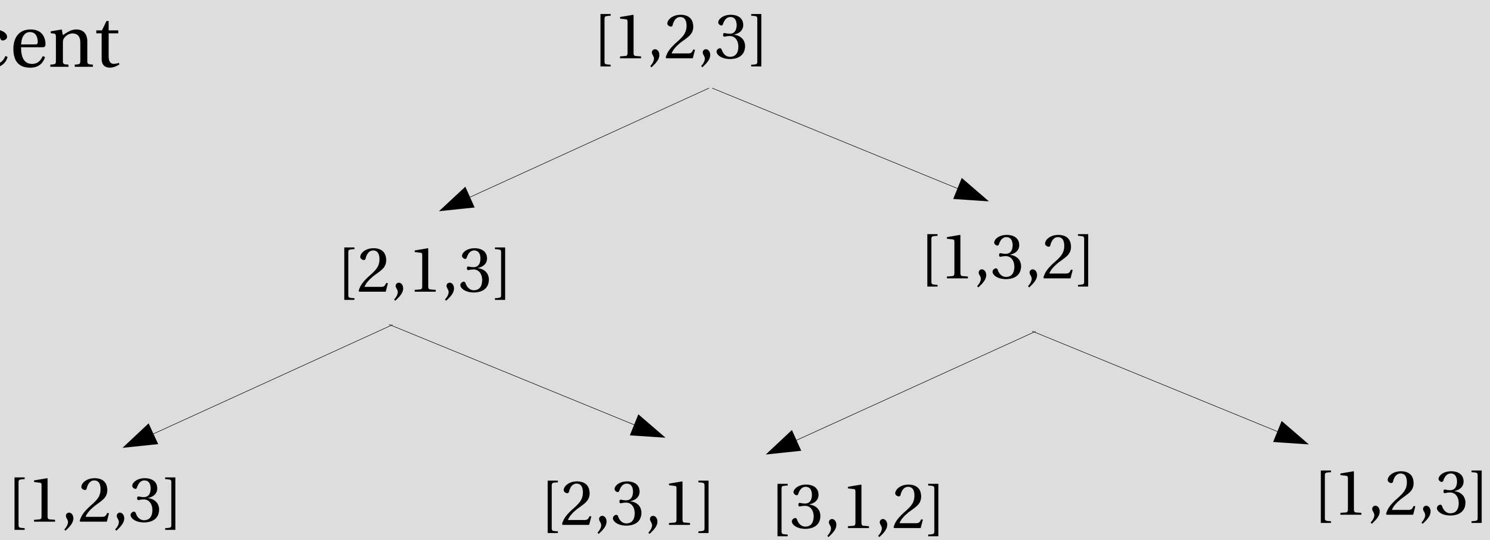
State Spaces

- Real world is absurdly complex - state space must be abstract

For for problem solving :
- (Abstract) state = set of real states
- (Abstract) action = complex combination of real actions e.g., “go from Arad to 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!

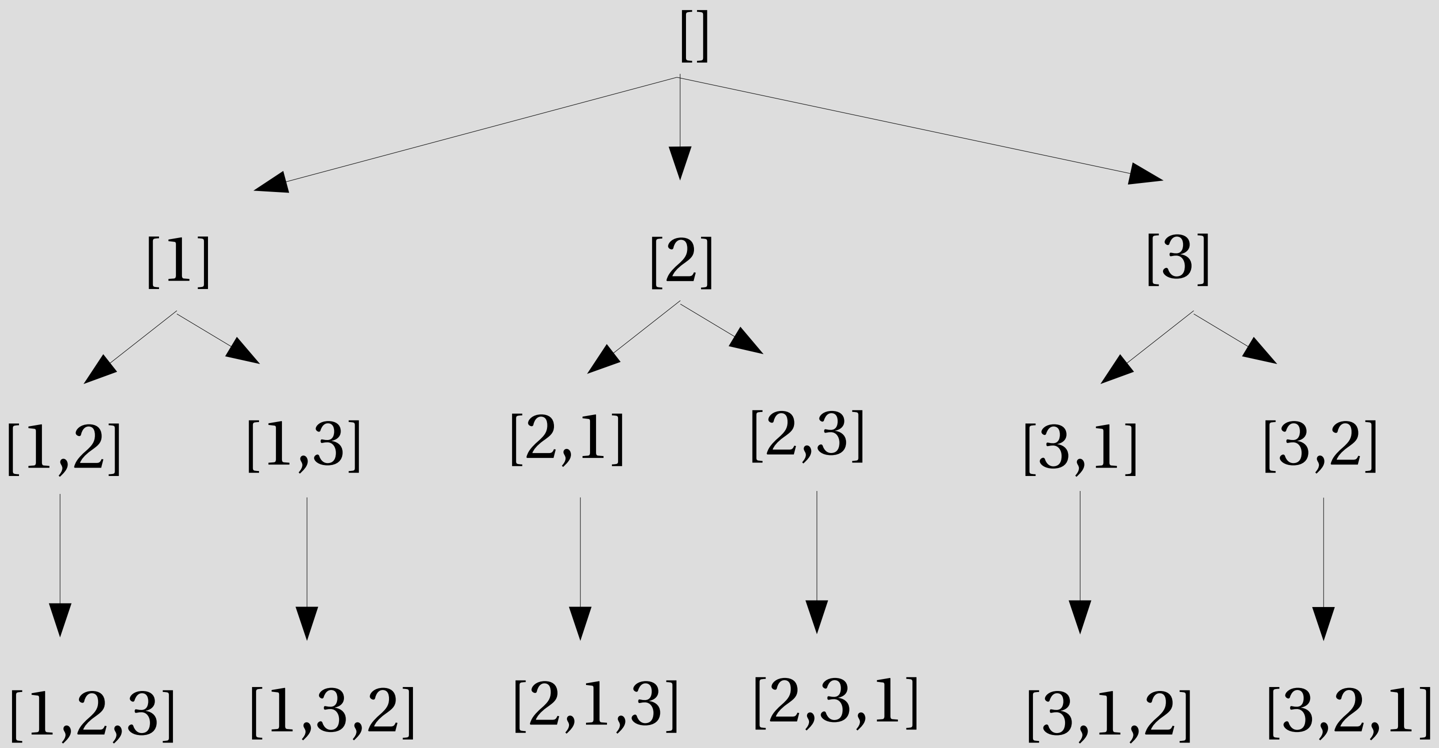
Appliance Factory Example

Three work cells 1,2,3 – arrange cells to form assembly line
Each work cell capable of performing different assembly operations with dependencies
State Space 1 – all permutations, operators reorder adjacent



Appliance Factory II

State Space 2 – all sequence, operators reorder

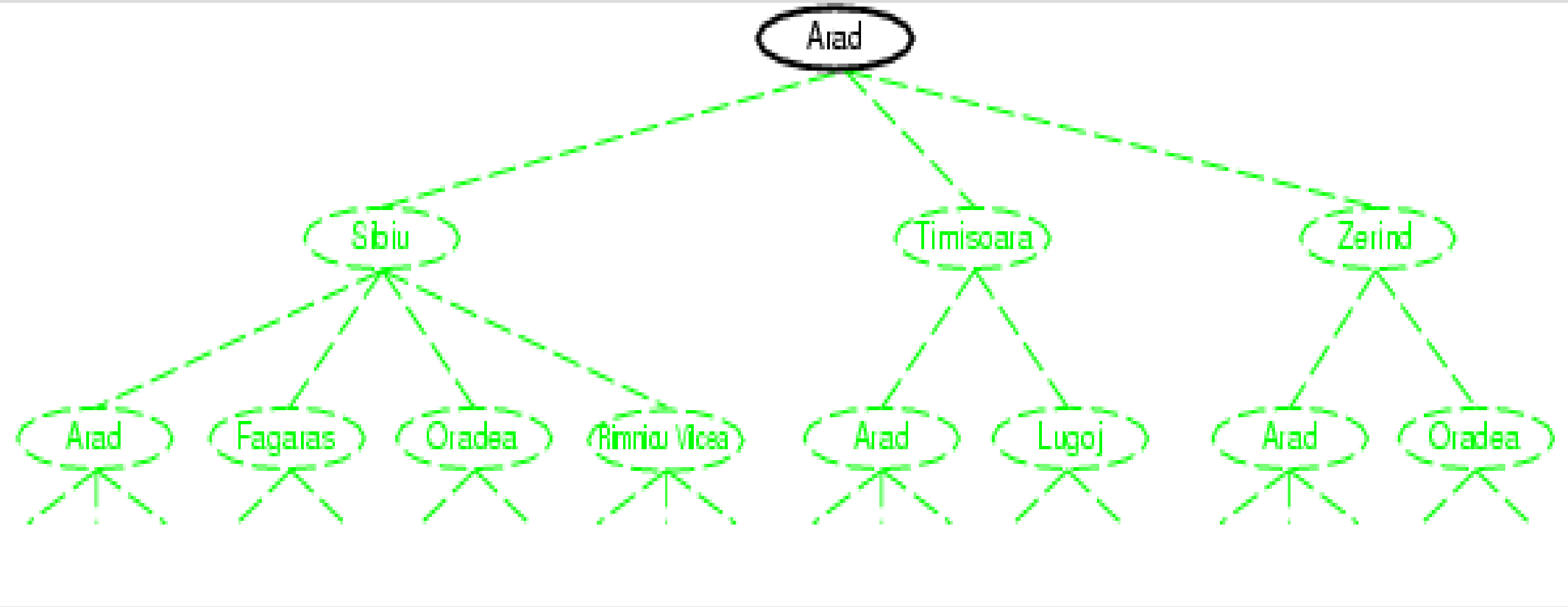


Tree search algorithms

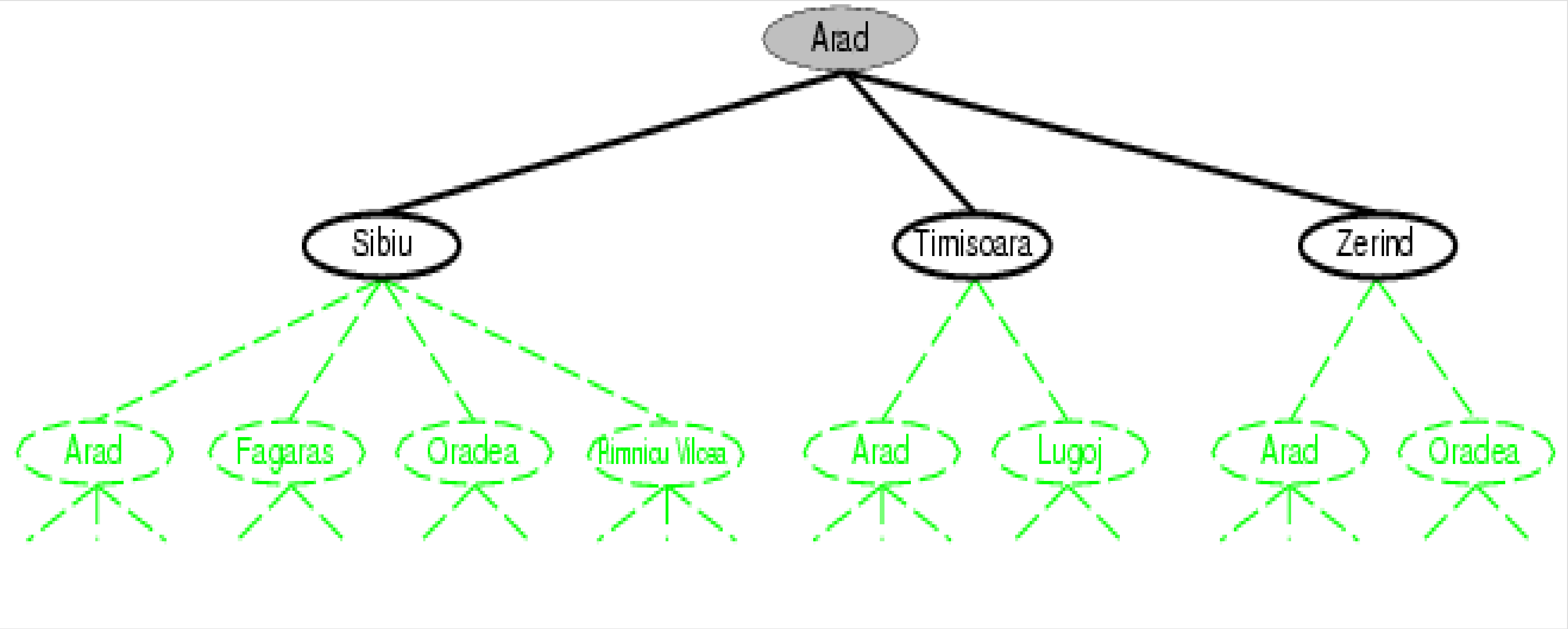
- Basic idea:
  - Exploration of state space by generating successors of already explored states

```
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
        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
```

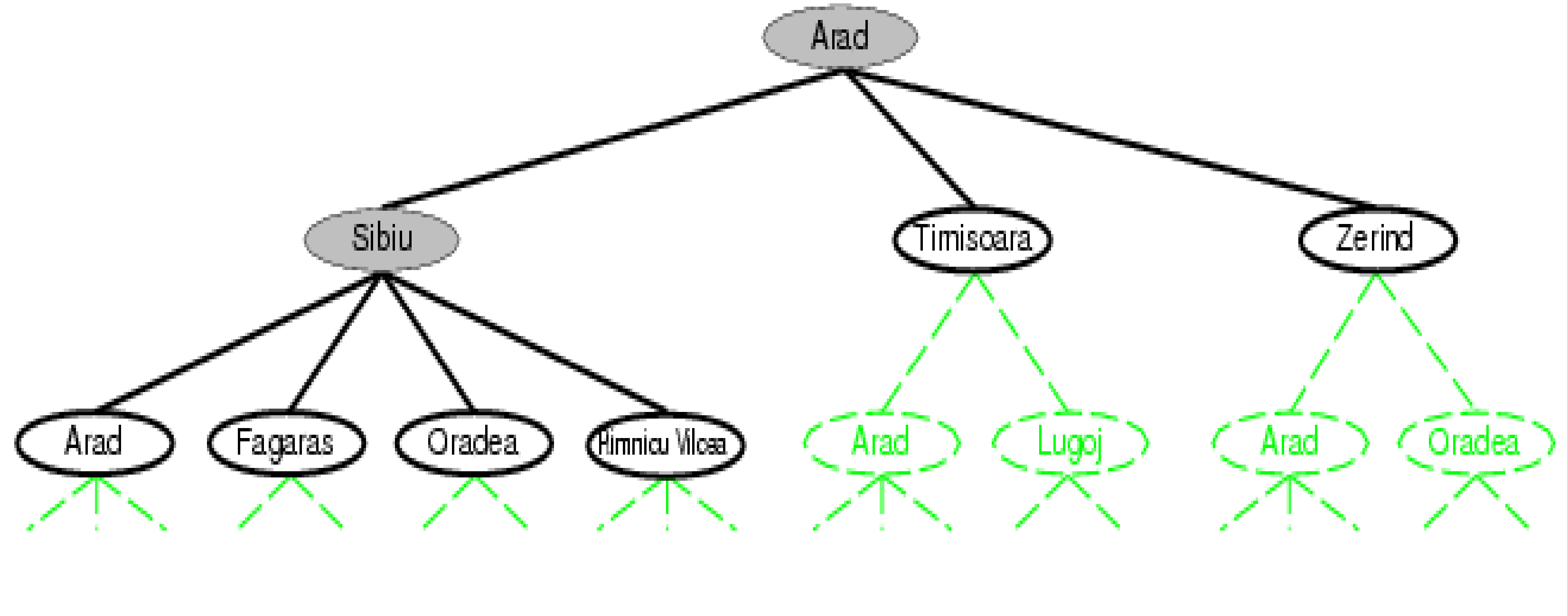
Tree search example



Tree Search example

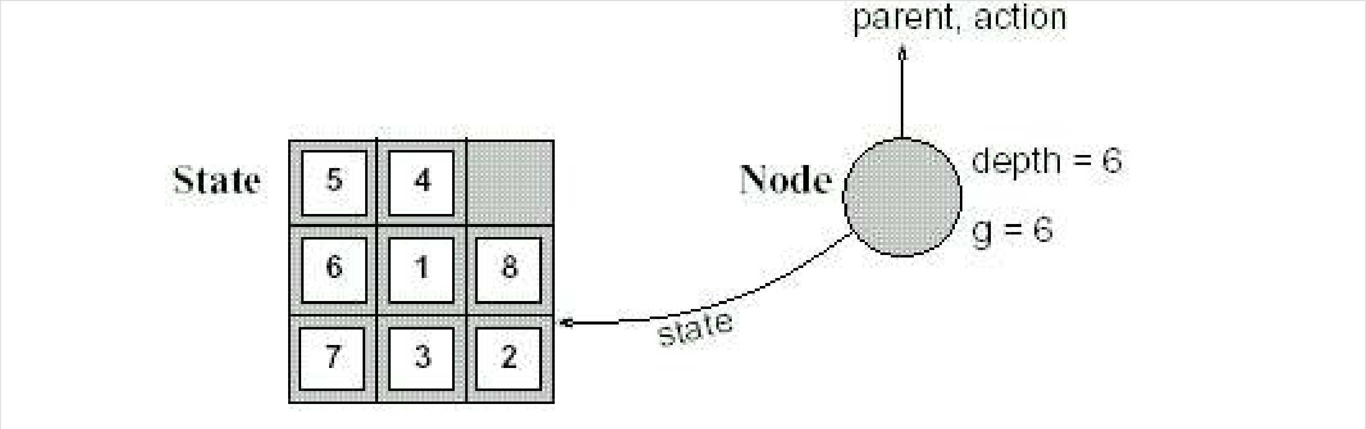


Tree search example



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 including (parent, children, depth, cost g(x))
- States do not have parents, children or costs



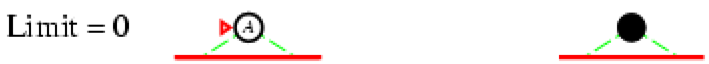
Search Strategies

- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?
- Time and space complexity:
  - ***b***: maximum branching factor of the search tree
  - ***d***: depth of the least-cost solution
  - ***m***: maximum depth of the state space (can be infinite)

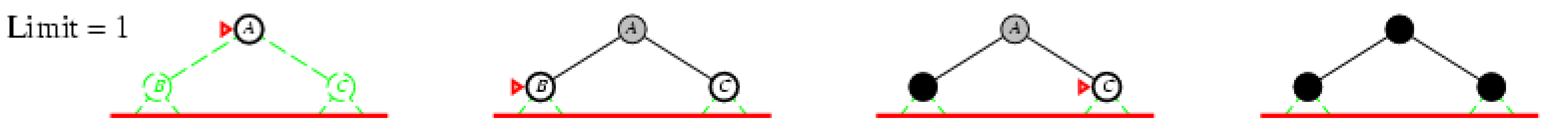
Uninformed Search Strategies

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative-deepening search
- Fringe (collection of nodes that have been generated but not yet expanded)
  - BFS – FIFO
  - UCS – PRIORITY QUEUE
  - DFS – LIFO
  - DLS – DFS with cutoff
  - IDS - Multiple times DLS

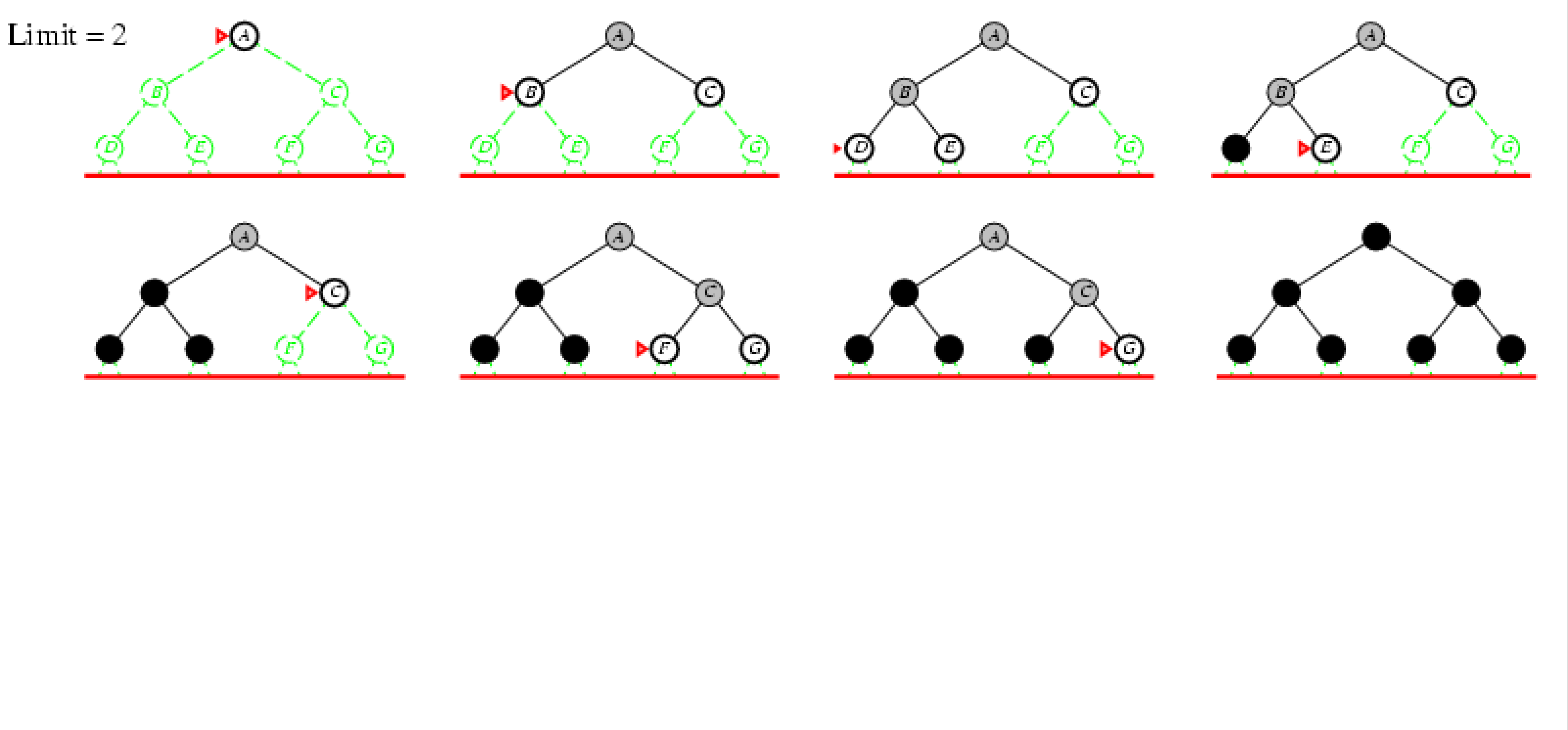
Iterative Deepening



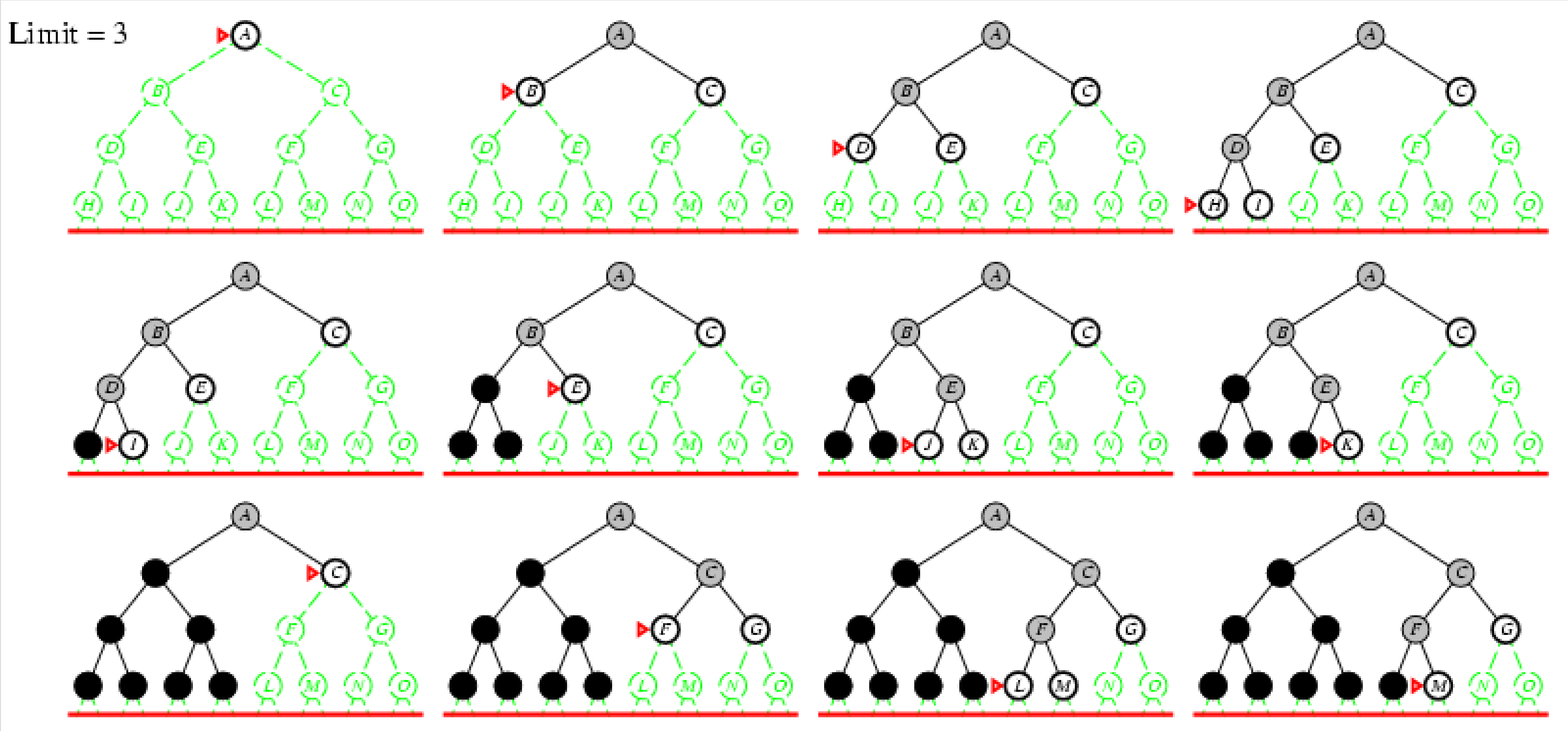
Iterative Deepening



Iterative Deepening



Iterative Deepening

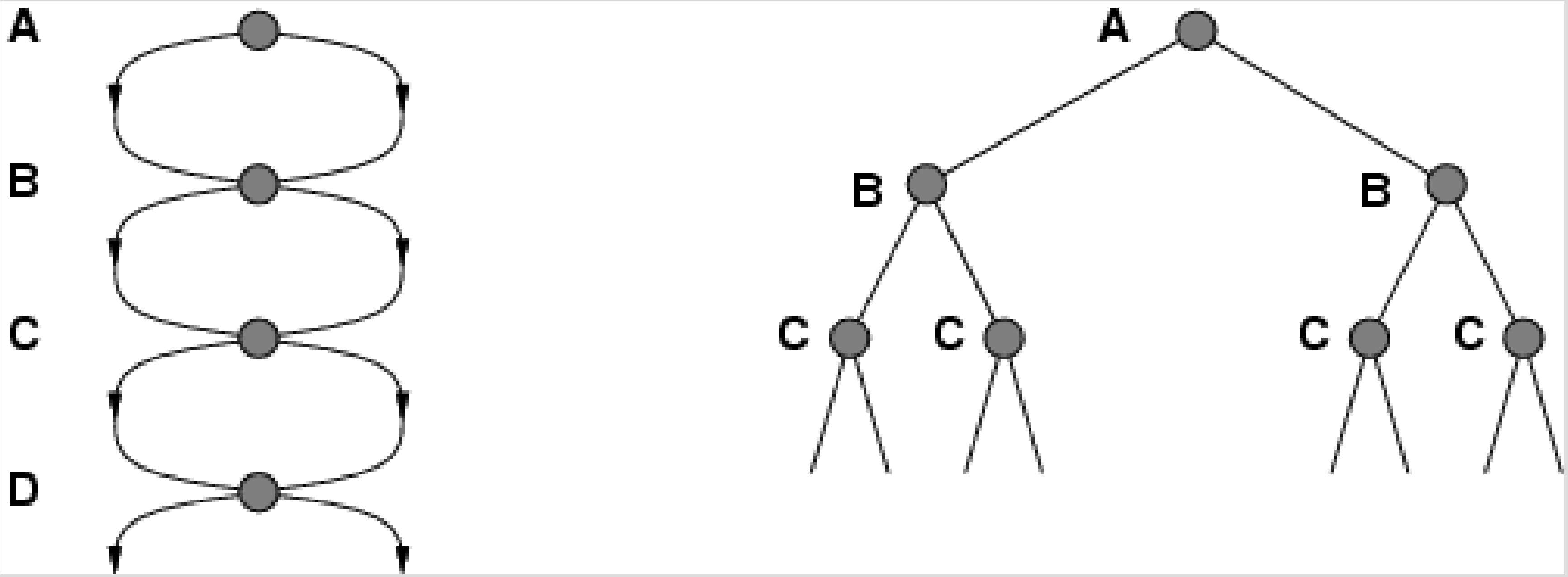


Summary of algorithms

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

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 SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
```