Chapter 3

Problem Solving

By Dr. Chhaya S Pawar

Search Algorithms

- Problem solving agent searches sequence of steps to reach solution
- Uninformed Search Algorithms: has no other information except the problem definition
- Informed Search Algorithms: Know where to look for the solution

Search Algorithms

function GENERAL-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

Example: Romania

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

Formulate goal:

be in Bucharest

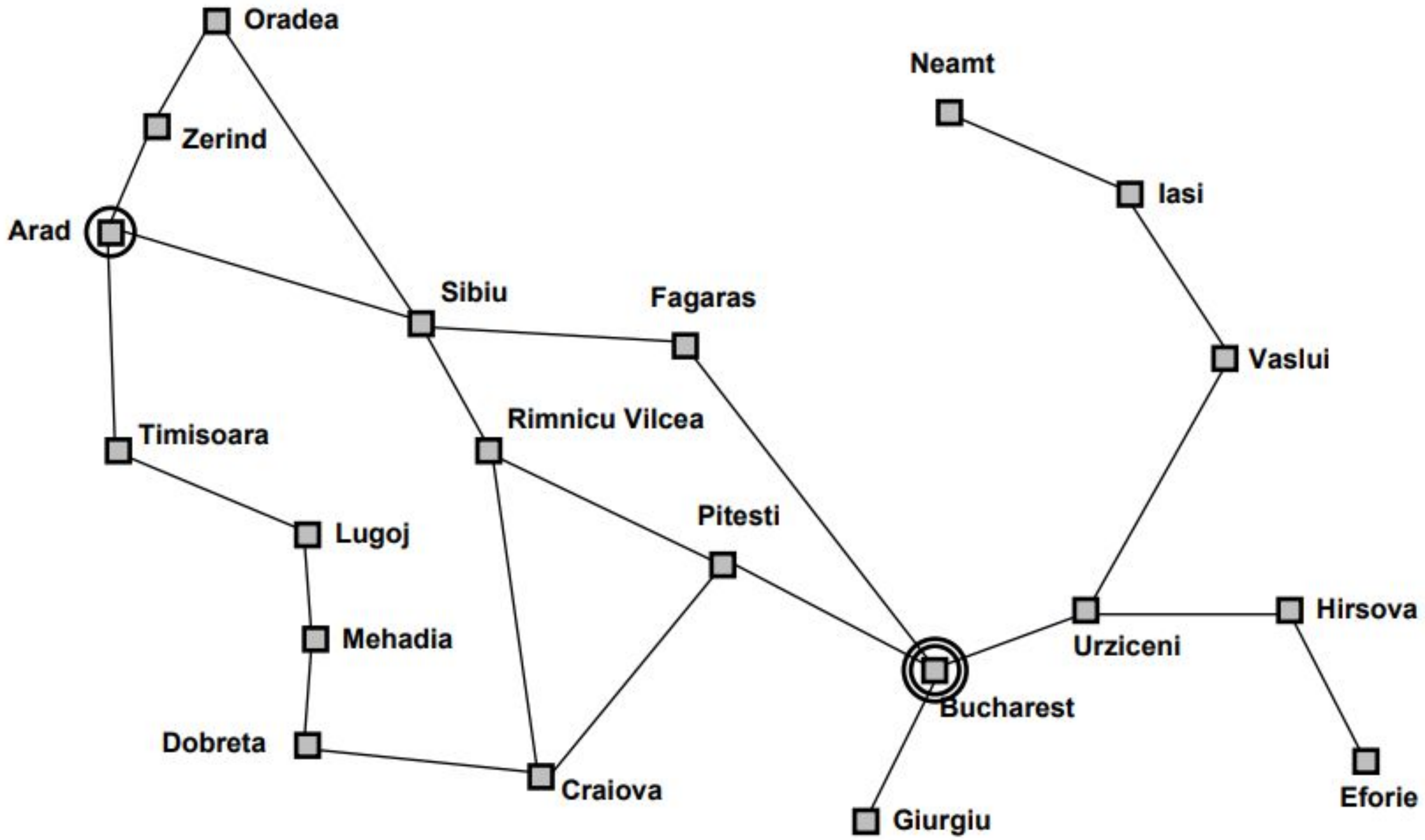
Formulate problem:

states: various cities
operators: 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:

initial state e.g., "at Arad"

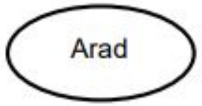
operators (or *successor function* $S(x)$)
e.g., Arad $\rightarrow$ Zerind Arad $\rightarrow$ Sibiu etc.

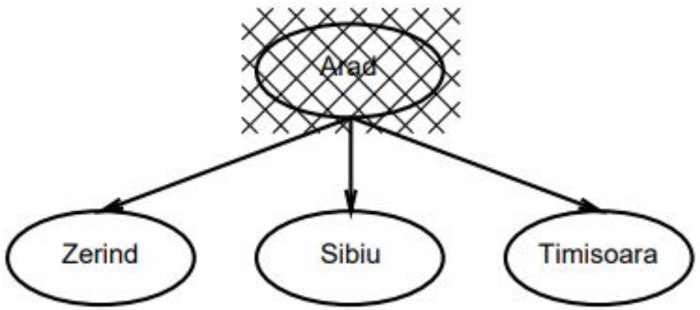
goal test, can be
explicit, e.g., $x =$ "at Bucharest"
implicit, e.g., $NoDirt(x)$

path cost (additive)
e.g., sum of distances, number of operators executed, etc.

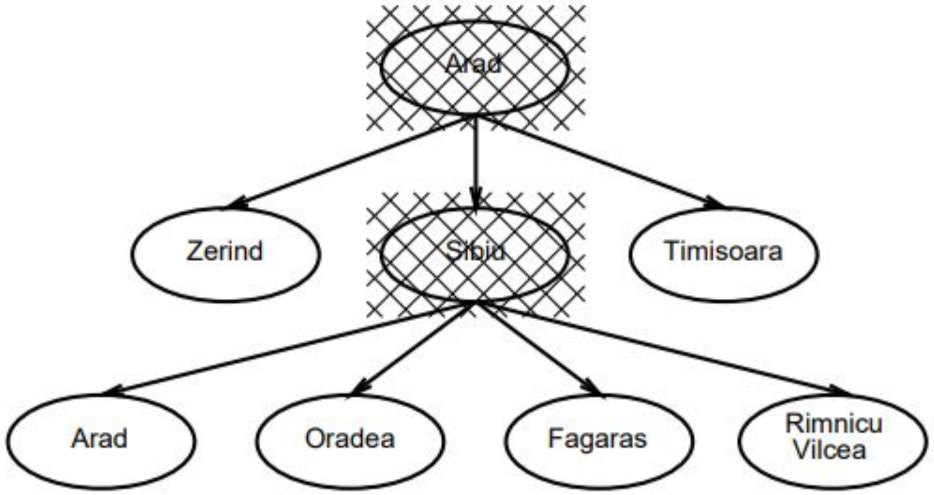
A *solution* is a sequence of operators
leading from the initial state to a goal state

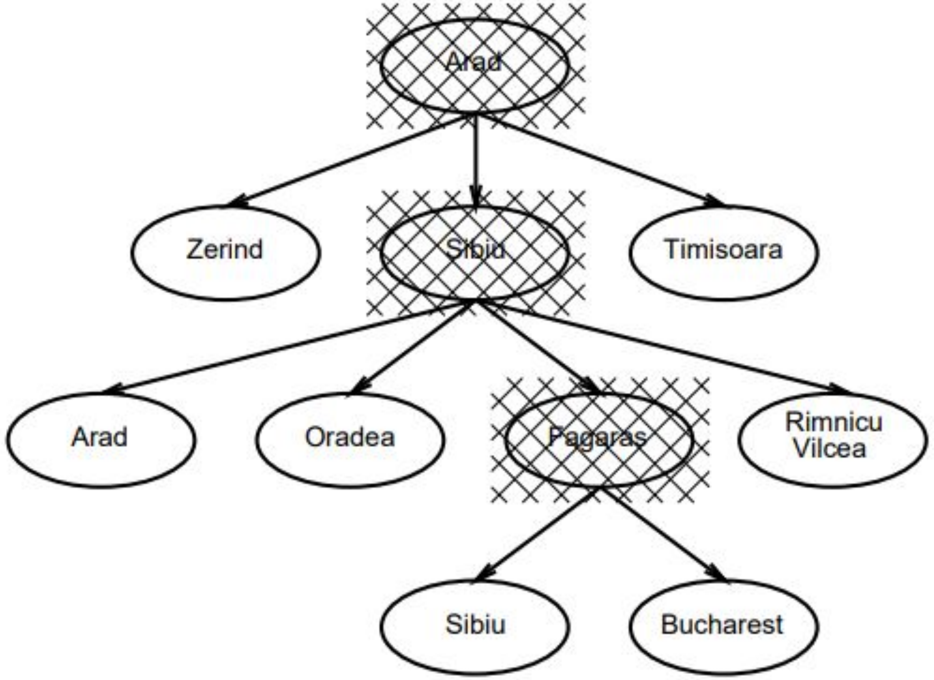
General search example





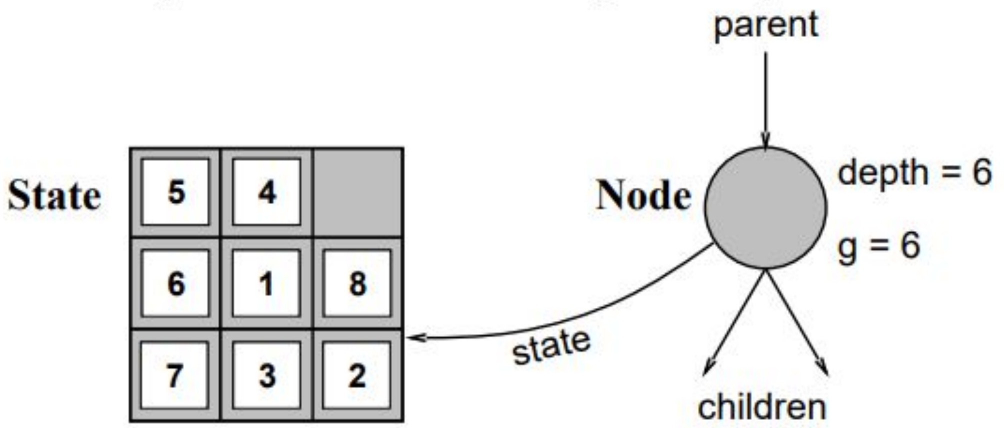
Arad
Zerind
Sibiu
Timisoara


Arad
Zerind
Sibiu
Timisoara
Arad
Oradea
Fagaras
Rimnicu Vilcea


Arad
Zerind
Sibiu
Timisoara
Arad
Oradea
Fagaras
Rimnicu Vilcea
Sibiu
Bucharest

Implementation contd: 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 *parent*, *children*, *depth*, *path cost* $g(x)$
States do not have parents, children, depth, or path cost!



The EXPAND function creates new nodes, filling in the various fields and using the OPERATORS (or SUCCESSORFN) of the problem to create the corresponding states.

Search strategies

A 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/expanded
- space complexity—maximum number of nodes in memory
- optimality—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
- d—depth of the least-cost solution
- m—maximum depth of the state space (may be ∞)

Uninformed search strategies

Uninformed strategies use only the information available in the problem definition

Breadth-first search

Uniform-cost search

Depth-first search

Depth-limited search

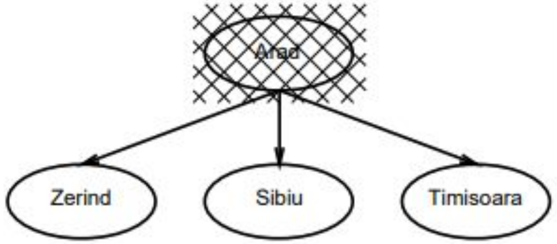
Iterative deepening search

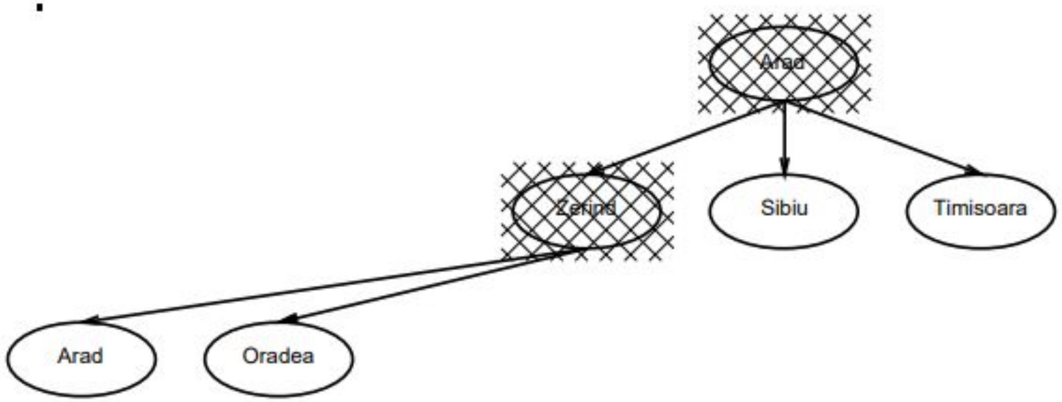
Breadth-first search

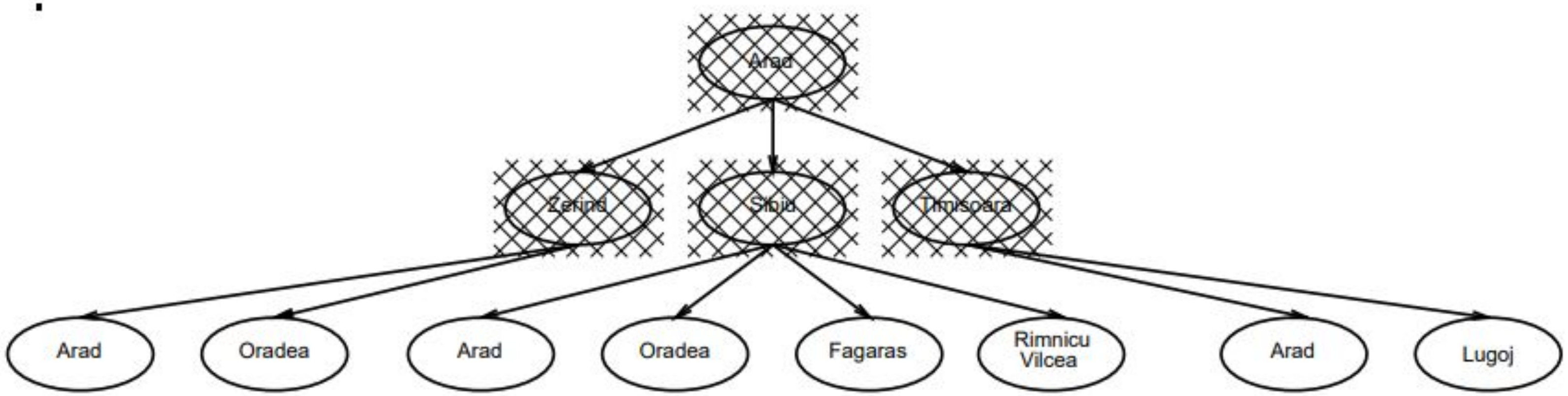
Expand shallowest unexpanded node

Implementation:

QueueingFn = put successors at end of queue


Arad
Zerind
Sibiu
Timisoara


Arad
Zerind
Sibiu
Timisoara
Arad
Oradea


Arad
Zerind
Sibiu
Timisoara
Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Arad
Lugoj

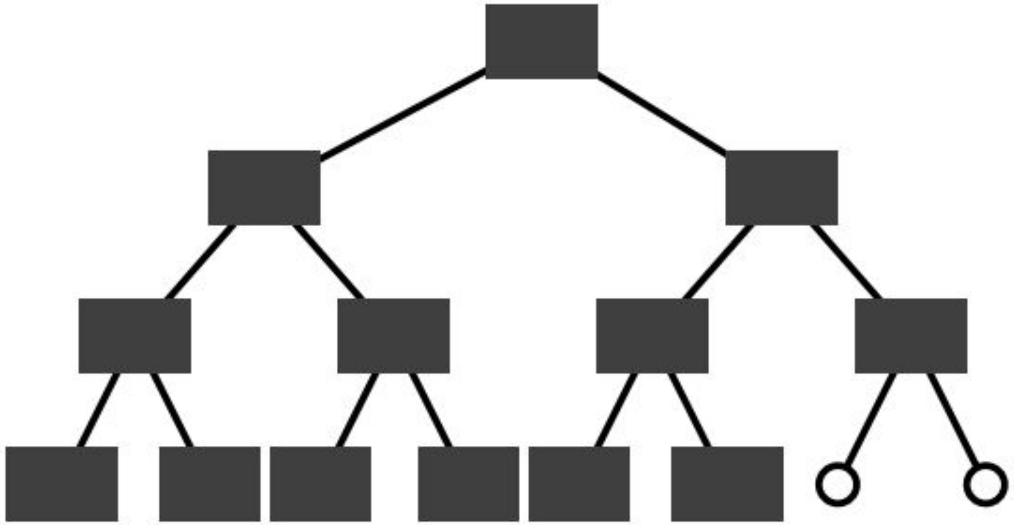
Properties of breadth-first search

Complete??

Time??

Space??

Optimal??



Breadth-First Search

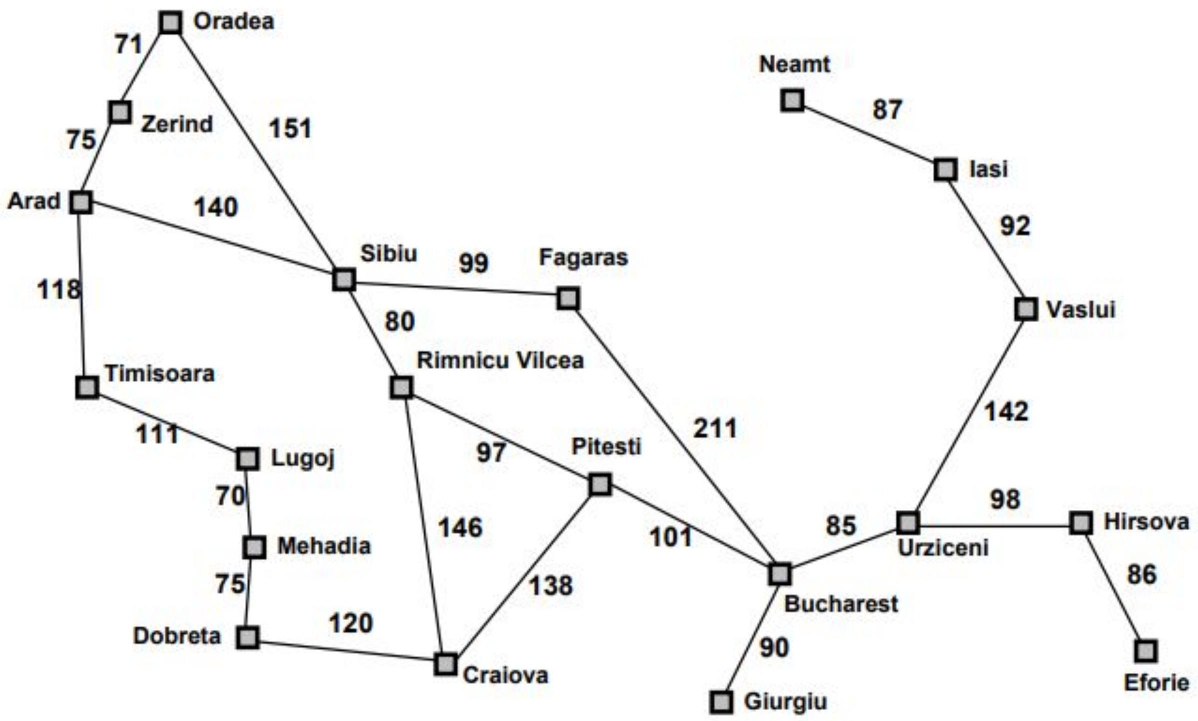
- Enqueue nodes on nodes in **FIFO** (first-in, first-out) order.

- **Complete** (if b is finite)
- **Optimal** if all operators have the same cost. Otherwise, not optimal but finds solution with shortest path length.

- **Exponential time and space complexity**, $O(b^d)$, where d is the depth of the solution and b is the branching factor (i.e., number of children) at each node

- Will take a **long time to find solutions** with a large number of steps because must look at all shorter length possibilities first

Romania with step costs in km



Straight–line distance to Bucharest

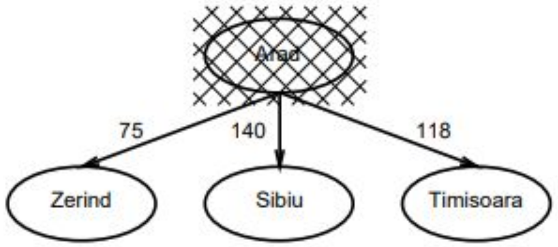
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

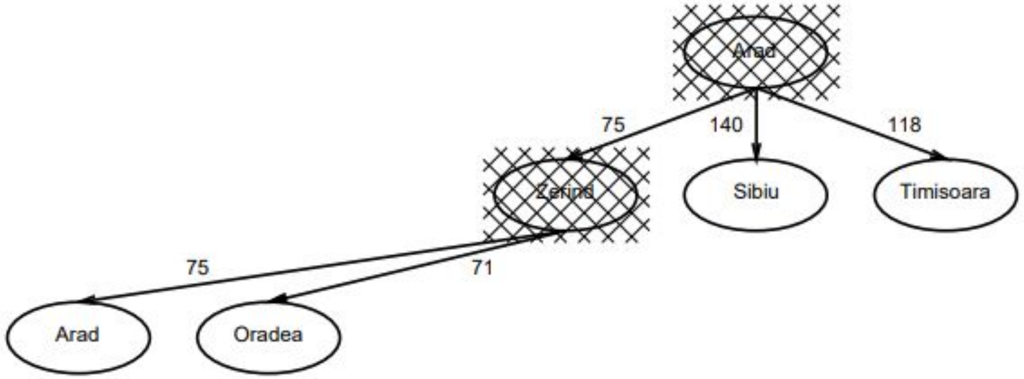
Uniform-cost search

Expand least-cost unexpanded node

Implementation:

$\textsc{QueueingFn}$ = insert in order of increasing path cost


Arad
75
140
118
Zerind
Sibiu
Timisoara



Arad
75
140
118
Zerind
Sibiu
Timisoara
75
71
Arad
Oradea

UCS vs. BFS

- When all step costs are the same, uniform-cost search is similar to breadth-first search except
  - BFS stops as soon as it generates a goal, whereas
  - uniform-cost search examines all the nodes at the goal's depth to see if one has a lower cost;
  - thus uniform-cost search does strictly more work by expanding nodes at depth d unnecessarily.

- Complexity analysis depends on path costs rather than depths

- Time
  - Complexity cannot be determined easily by d or d
  - Let C* be the cost of the optimal solution
  - Assume that every action costs at least ε
  - $O(b^{1+ \text{ceil}(C^*/ \varepsilon)})$

- Space
  - $O(b^{1+\text{ceil}(C^*/ \varepsilon)})$

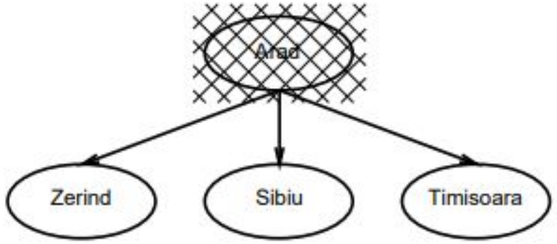
- Note: Uniform cost explores large trees of small steps
- So, $b^{1+\text{ceil}(C^*/\varepsilon)}$ can be much greater than b^{d+1}
- If all step costs are equal then $b^{1+\text{ceil}(C^*/\varepsilon)}$ is equal to b^{d+1}

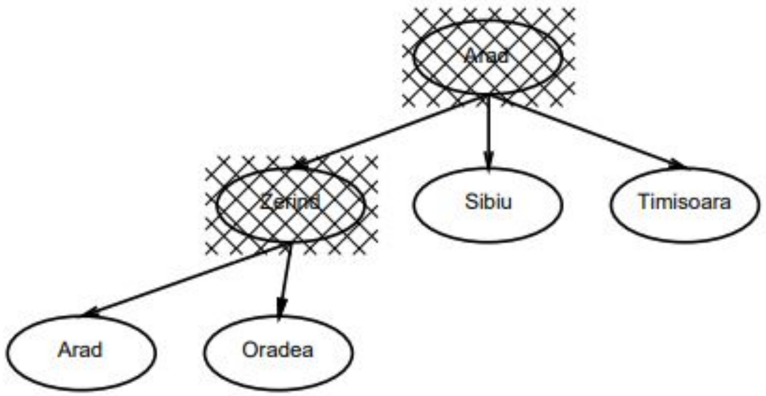
Depth-first search

Expand deepest unexpanded node

Implementation:

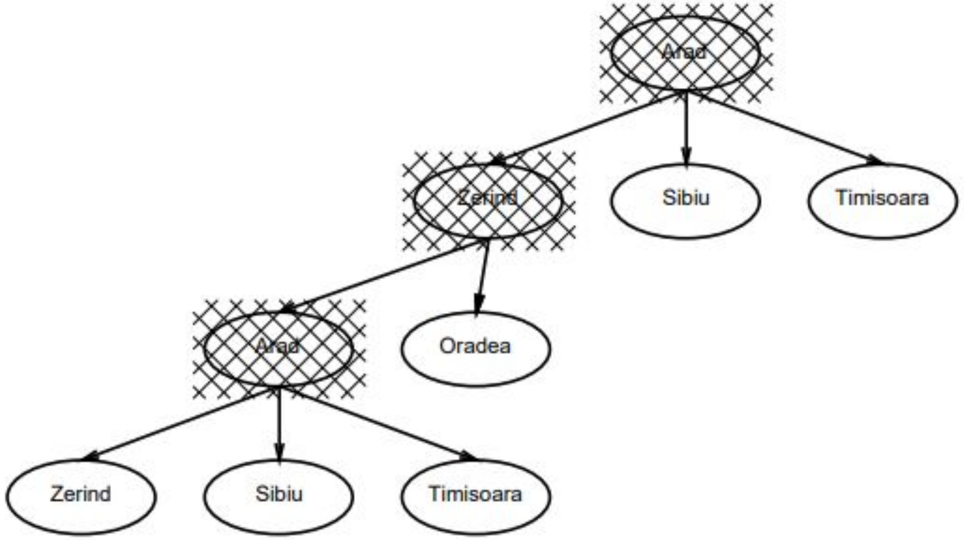
$\textsc{QueueingFn}$ = insert successors at front of queue


Arad
Zerind
Sibiu
Timisoara


Arad
Zerind
Sibiu
Timisoara
Arad
Oradea

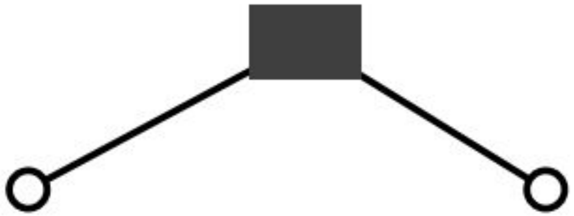
.

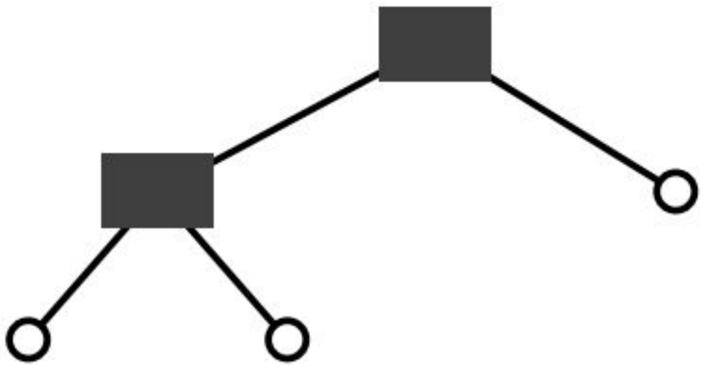
.

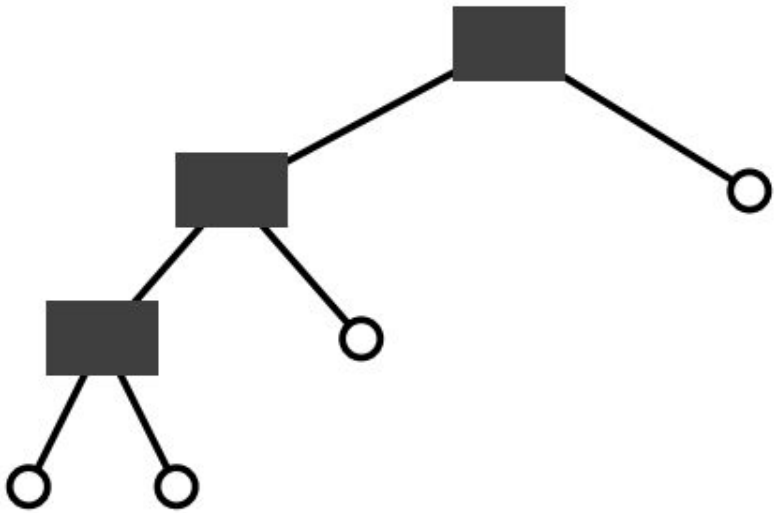


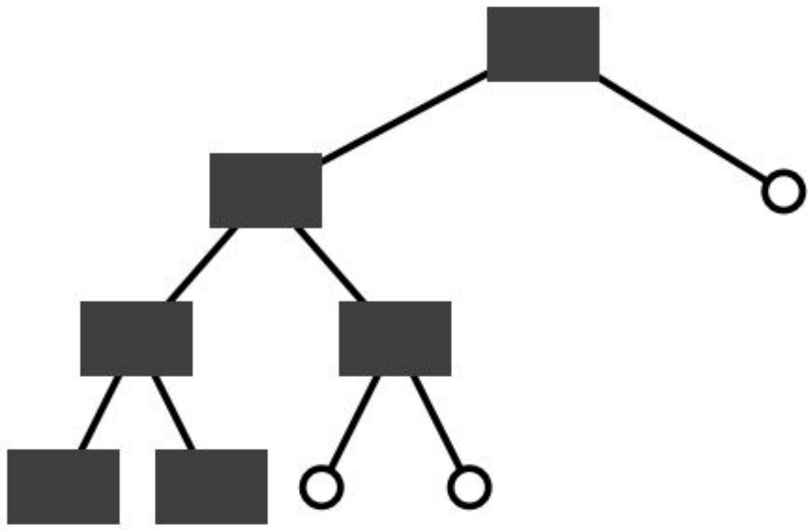
I.e., depth-first search can perform infinite cyclic excursions
Need a finite, non-cyclic search space (or repeated-state checking)

DFS on a depth-3 binary tree

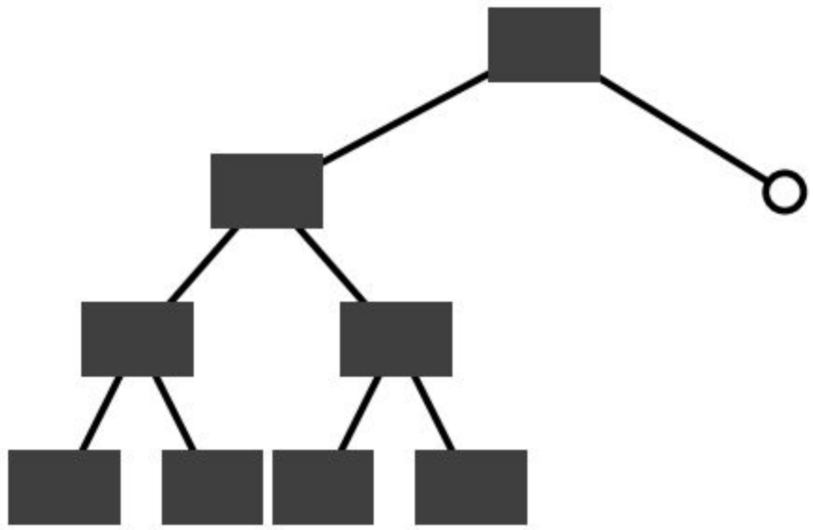


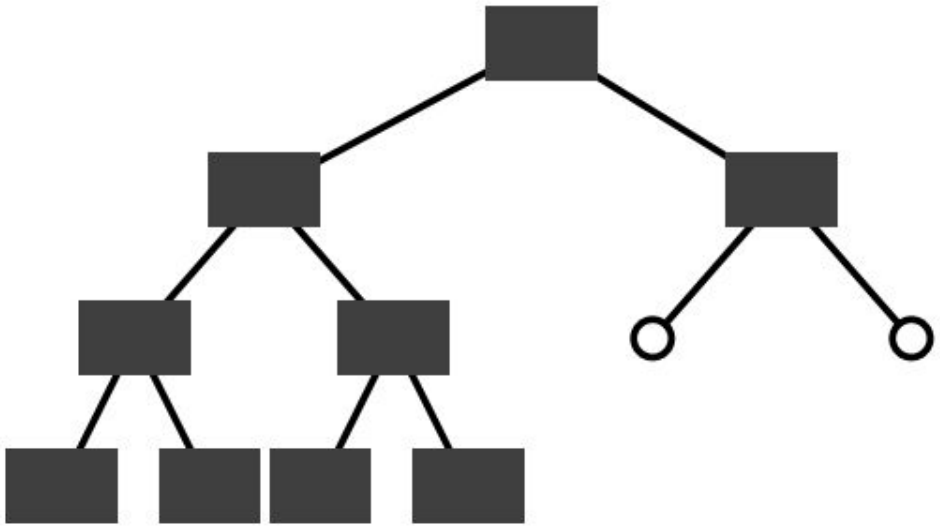


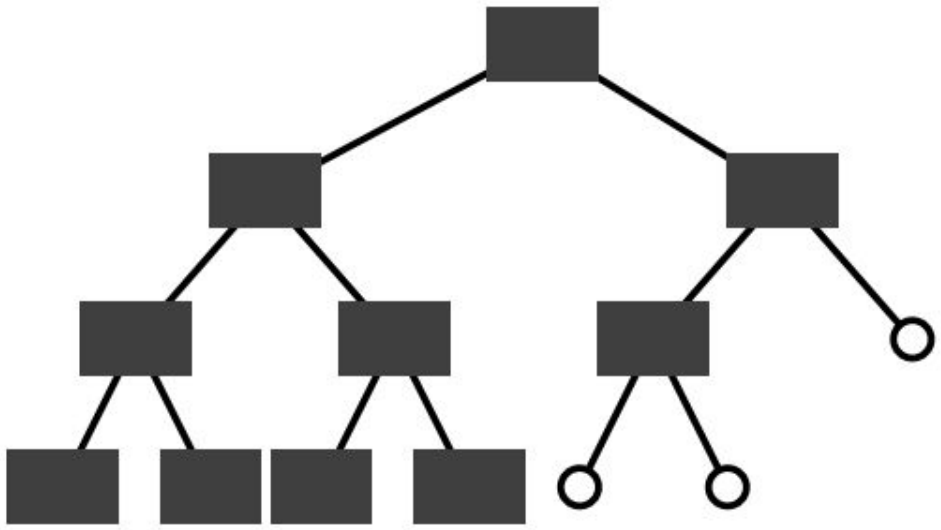


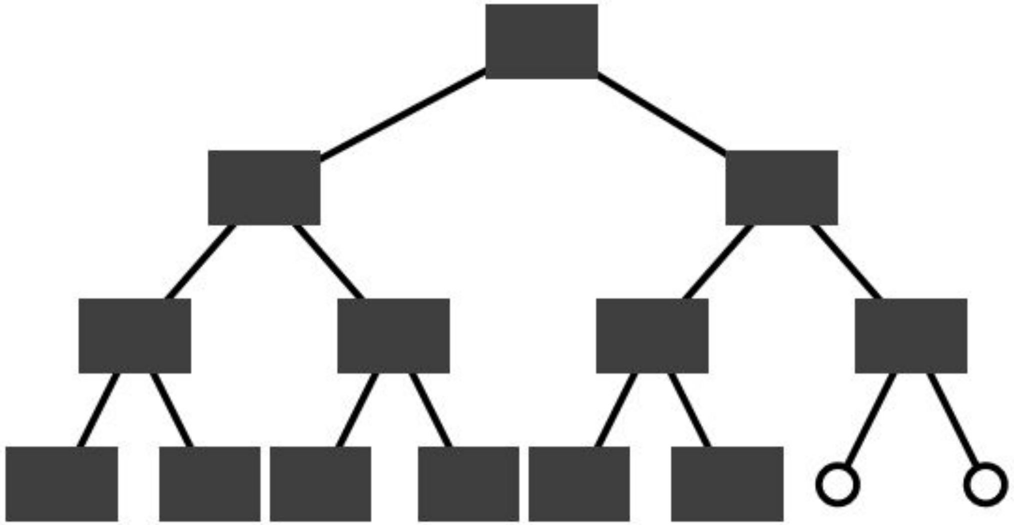


DFS on a depth-3 binary tree, contd.









Properties of depth-first search

Complete??

Time??

Space??

Optimal??

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

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

Optimal?? No

Depth-limited search

= depth-first search with depth limit l

Implementation:

Nodes at depth l have no successors

Iterative deepening search

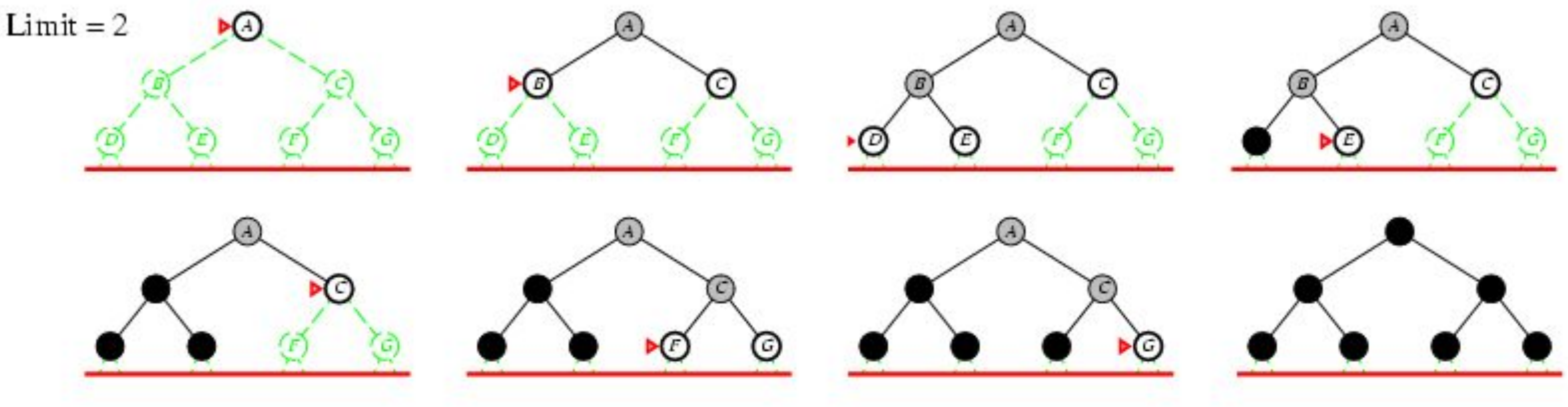
```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution sequence
    inputs: problem, a problem

    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
    end
```

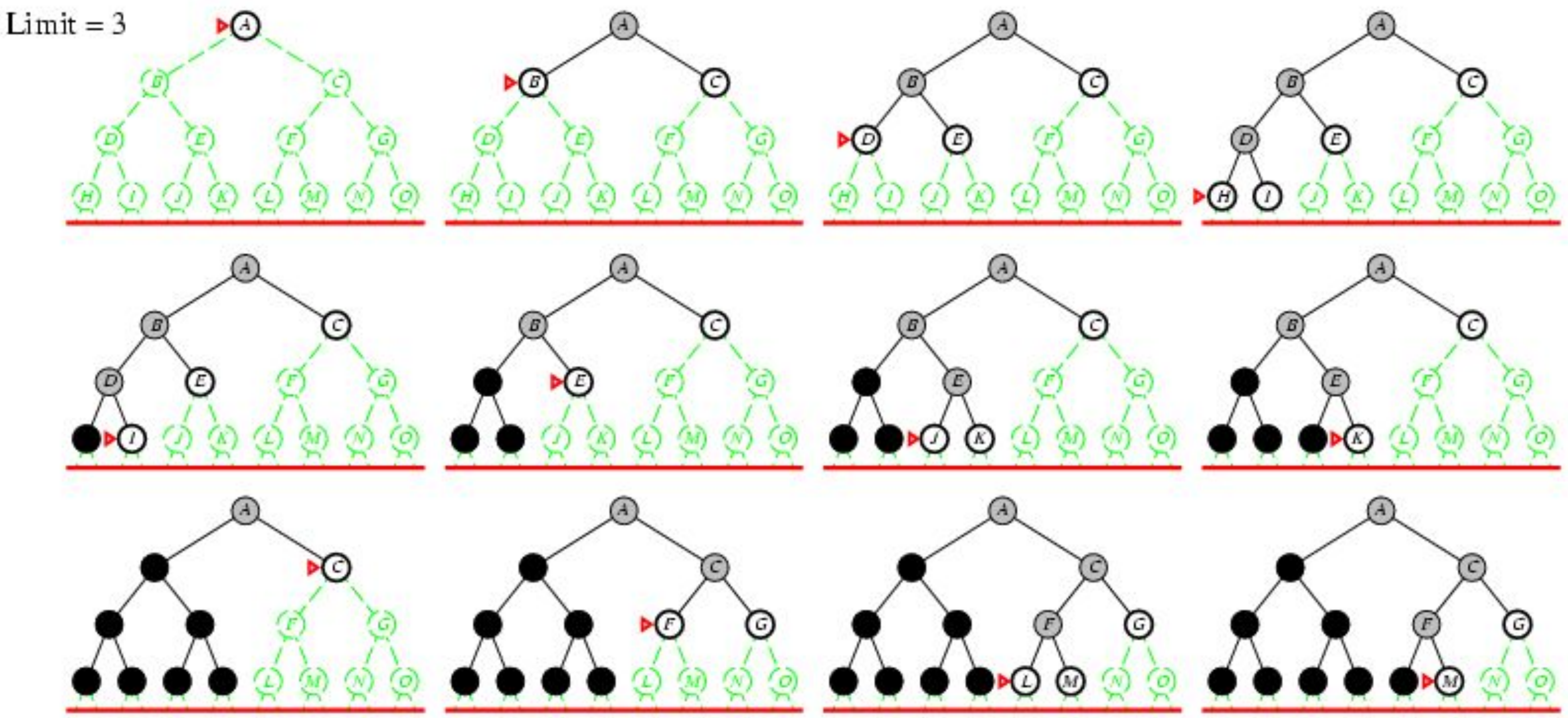
Iterative deepening search L=0

Iterative deepening search L=1

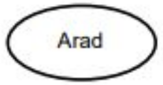
Iterative deepening search *L*=2



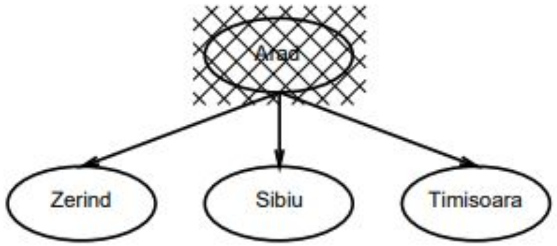
Iterative Deepening Search L=3



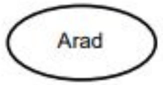
Iterative deepening search $l = 0$

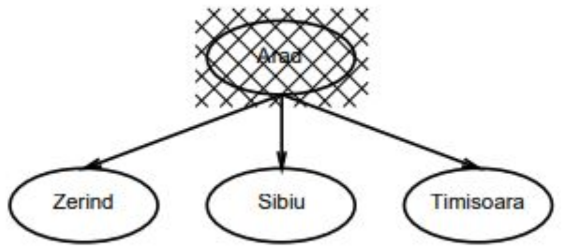


Iterative deepening search $l = 1$

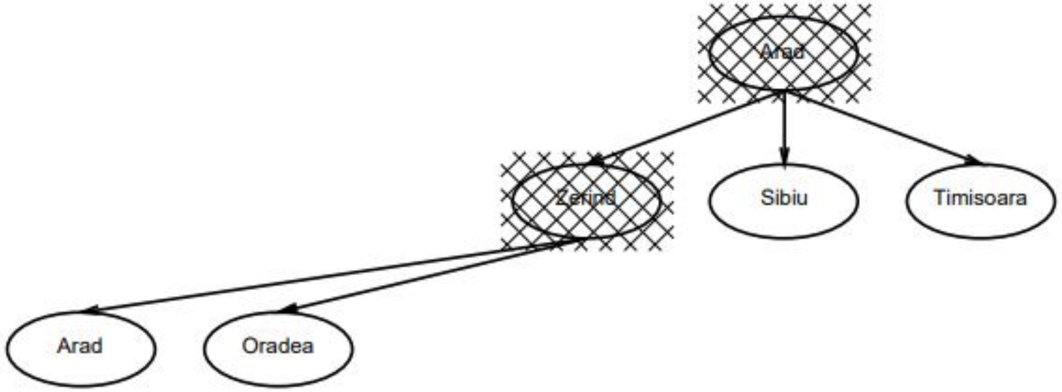

Arad
Zerind
Sibiu
Timisoara

Iterative deepening search $l = 2$

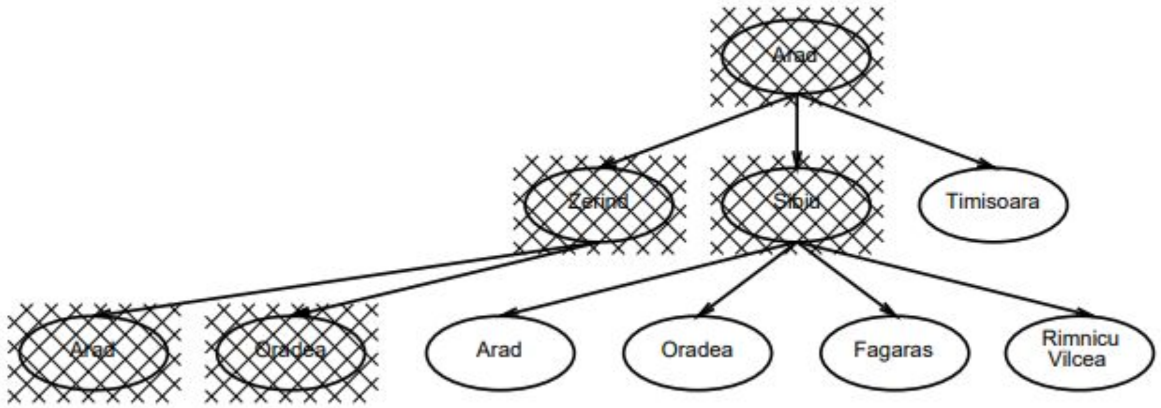




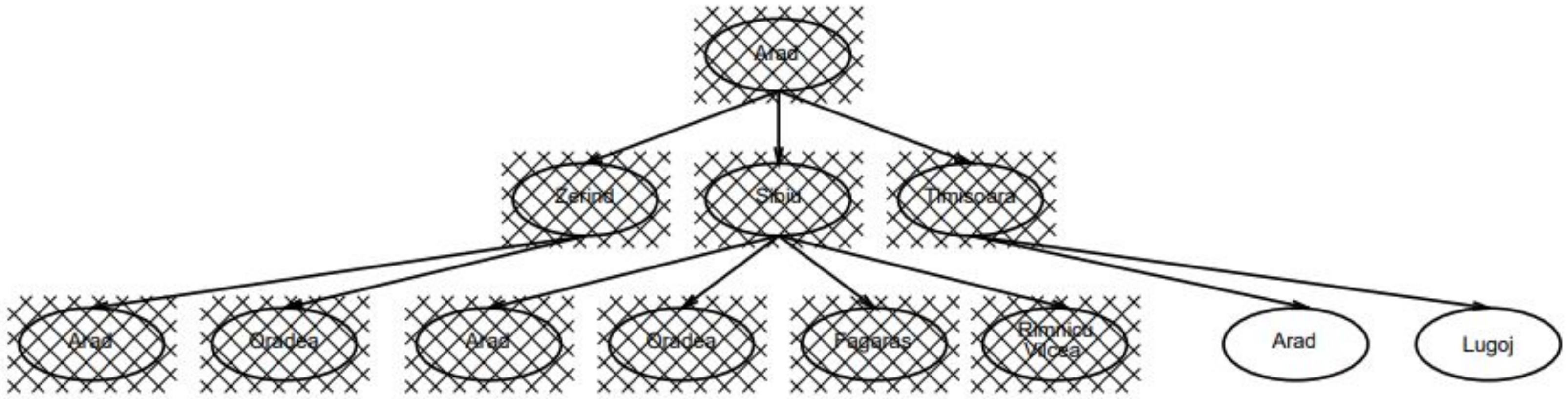
Arad
Zerind
Sibiu
Timisoara



Arad
Zerind
Sibiu
Timisoara
Arad
Oradea



Arad
Zerind
Sibiu
Timisoara
Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea



Arad
Zerind
Sibiu
Timisoara
Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Arad
Lugoj

Properties of iterative deepening search

Complete??

Time??

Space??

Optimal??

Properties of Iterative Deepening Search

- Complete : Yes
- Optimal: Yes, if step cost are equal
- Time Complexity: $O(b^d)$

  = d(b)+(d-1)b2+(d-2)b3+….+(1)bd
- Space Complexity: O(bd)

Iterative Deepening Depth-First Search

- **Exponential time complexity,** $O(b^d)$, like BFS
- **Linear space complexity**, $O(bd)$, like DFS

- Has advantage of BFS (i.e., completeness) and also advantages of DFS (i.e., limited space and finds longer paths more quickly)
- Generally preferred for **large state spaces** where **solution depth is unknown**

Comparing Search Strategies

Uninformed search strategies

b = branching factor

d = solution depth

l = depth limit

a complete if *b* is finite

b complete if step *cost* >= *ε* for positive ε

- m = max tree depth
- C* = cost of optimal
- ε = min action cost
- c optimal is step cost are identical
- d if both directions use BFS

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