Example: service robot

states??:

operators??:

goal test??:

path cost??:

Example: service robot

states??: real-valued coordinates of manipulator joint angles (retracted?)
real-valued position of robot (which room, floor?)
doors (where,open,closed) , people (where)
coffee machine (where, full)

operators??: continuous motions of robot (abstract navigation actions)

goal test??: coffee delivered to Henry

path cost??: time to execute

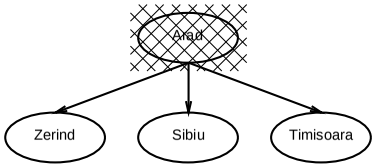
Search algorithms

Basic idea:
offline, simulated exploration of state space
by generating successors of already-explored states
(a.k.a. *expanding* states)

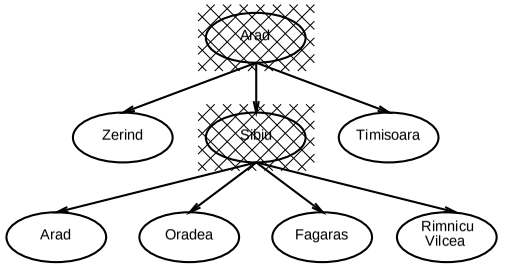
```
function General-Search( problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains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


General search example

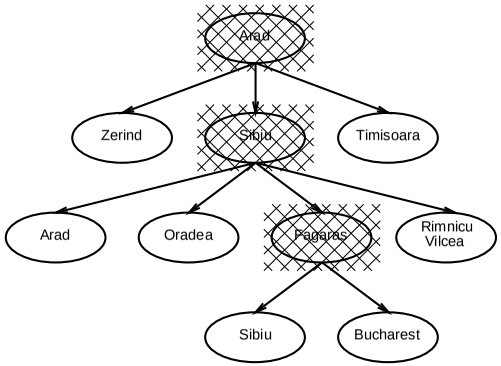
General search example



General search example



General search example



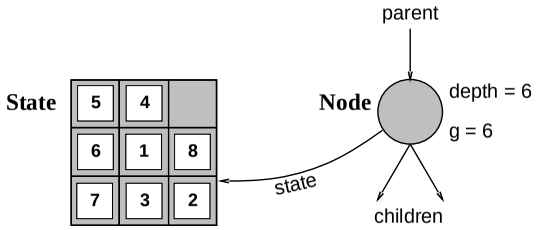
Implementation of search algorithms

function GENERAL-SEARCH(*problem*, QUEUING-FN) **returns** a solution, or failure

nodes ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[*problem*]))
loop do
  if *nodes* is empty **then return** failure
  node ← REMOVE-FRONT(*nodes*)
  if GOAL-TEST[*problem*] applied to STATE(*node*) succeeds **then return** *node*
  nodes ← QUEUING-FN(*nodes*, EXPAND(*node*, OPERATORS[*problem*]))
end

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

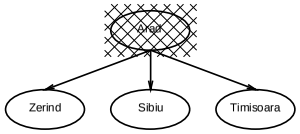
Implementation:
QUEUEINGFN = put successors at end of queue

Breadth-first search

.

.

.

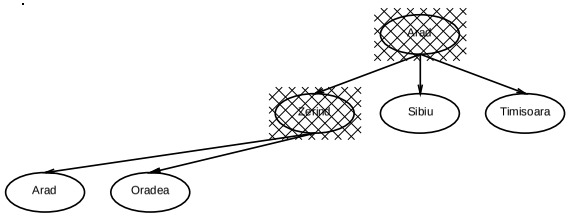


Breadth-first search

.

.

.

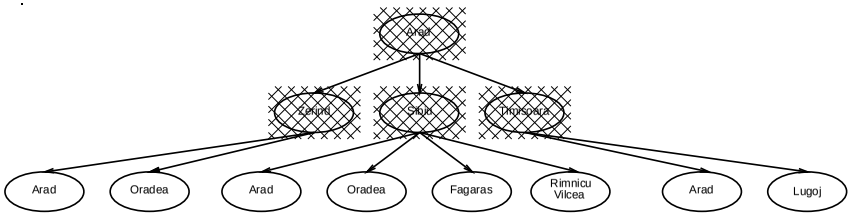


Breadth-first search

.

.

.



Properties of breadth-first search

Complete??

Time??

Space??

Optimal??

Properties of breadth-first search

Complete?? Yes (if b is finite)

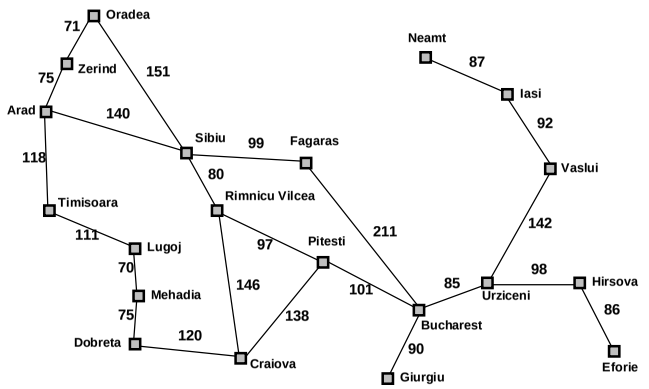
Time?? $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$, i.e., exponential in d

Space?? $O(b^d)$ (keeps every node in memory)

Optimal?? Yes (if cost = 1 per step); not optimal in general

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

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:

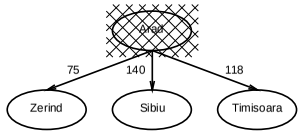
QUEUEINGFN = insert in order of increasing path cost

Uniform-cost search

.

.

.

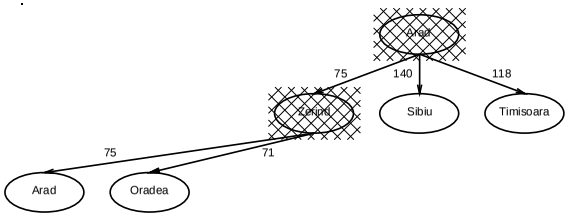


Uniform-cost search

.

.

.

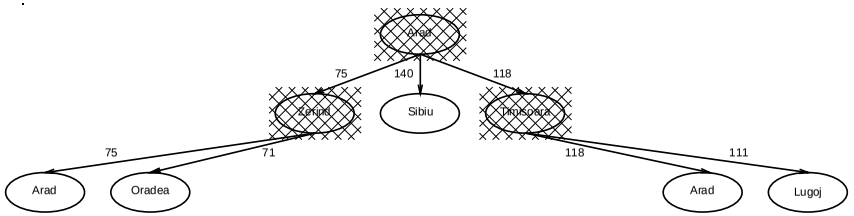


Uniform-cost search

.

.

.



Properties of uniform-cost search

Complete?? Yes, if step cost $\geq \epsilon$

Time?? # of nodes with $g \leq$ cost of optimal solution

Space?? # of nodes with $g \leq$ cost of optimal solution

Optimal?? Yes

Depth-first search

Expand deepest unexpanded node

Implementation:

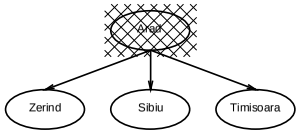
QUEUEINGFN = insert successors at front of queue

Depth-first search

.

.

.

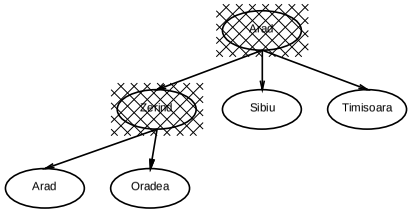


Depth-first search

.

.

.

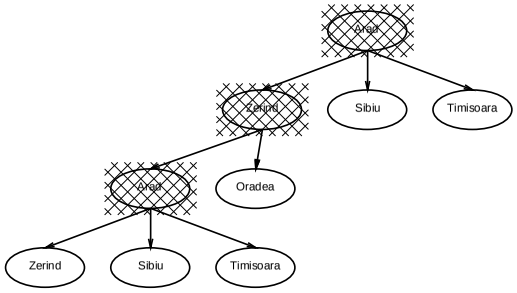


Depth-first search

.

.

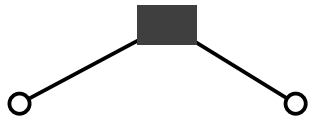
.



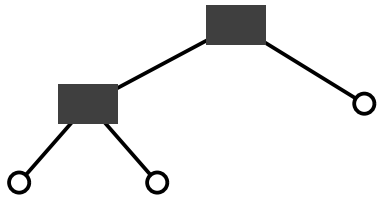
I.e., depth-first search can perform infinite cyclic excursions
Need a finite, non-cyclic search space (or repeated-state checking)

DFS on a depth-3 binary tree

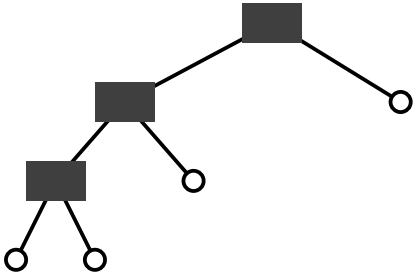
DFS on a depth-3 binary tree



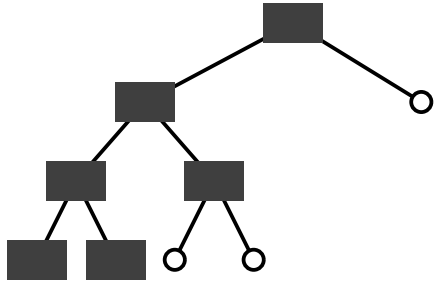
DFS on a depth-3 binary tree



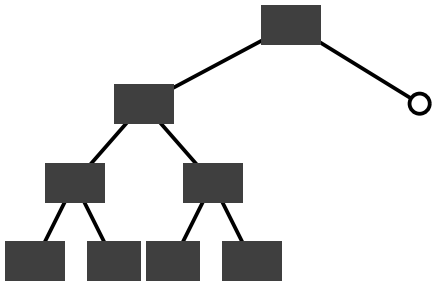
DFS on a depth-3 binary tree



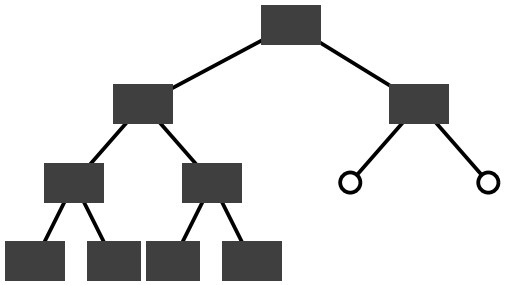
DFS on a depth-3 binary tree



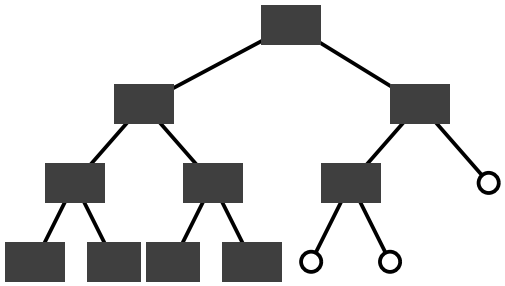
DFS on a depth-3 binary tree, contd.



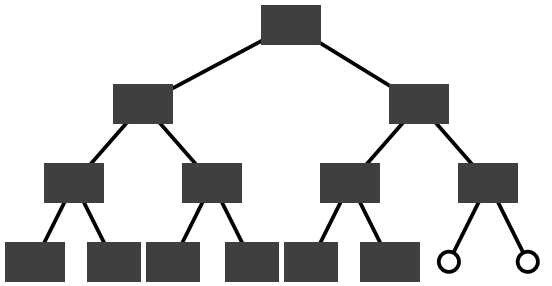
DFS on a depth-3 binary tree



DFS on a depth-3 binary tree



DFS on a depth-3 binary tree



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

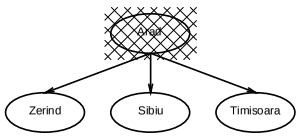
```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution sequence
    inputs: problem, a problem

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

Iterative deepening search $l = 0$

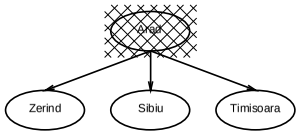
Iterative deepening search $l = 1$

Iterative deepening search $l = 1$

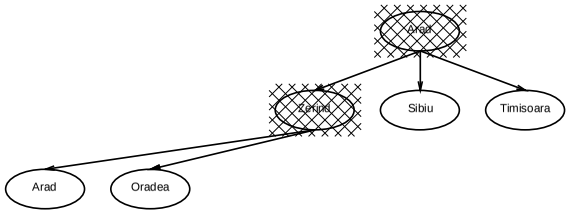


Iterative deepening search $l = 2$

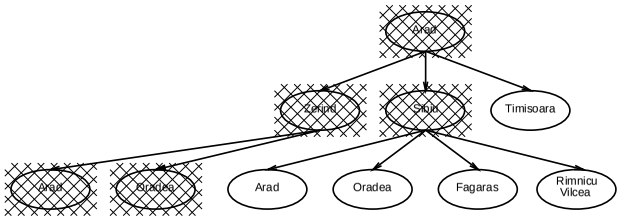
Iterative deepening search $l = 2$



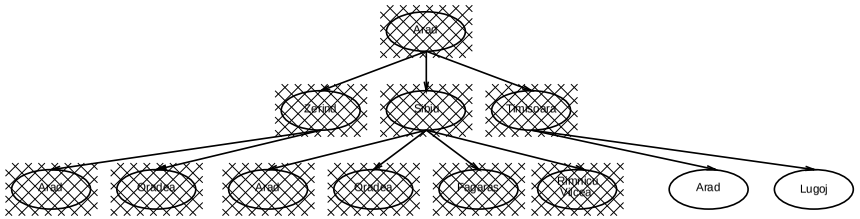
Iterative deepening search $l = 2$



Iterative deepening search $l = 2$



Iterative deepening search $l = 2$



Properties of iterative deepening search

Complete??

Time??

Space??

Optimal??

Properties of iterative deepening search

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal?? Yes, if step cost $= 1$

Properties of iterative deepening search

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal?? Yes, if step cost $= 1$

Breadth-first:	1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111
Iterative deepening:	6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456

For $b = 10$, iterative deepening search expands only 11% more nodes than breadth-first search!

Summary

Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored

Variety of uninformed search strategies

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