Problem Solving by Search

Supplementary videos

Artificial Intelligence, nptel Lectures:

Prof Deepak Khemani (IITM) Video Lectures in YOUTUBE are good.

https://www.youtube.com/watch?v=52nI2IPyOu8&list=PLEdmaLOtiZFswiGRQAr9iVVucY3L3jvgA&index=6

Problem solving

- Knowledge Based
  - Memory Based (Case Based)
    - Past experience is stored
  - Rule Based
    - Expert has given some rules
- Search Based
  - Blind
    - BFS, DFS, …
  - Heuristic
    - A*, …

What we learn is …

II Problem-solving

3 Solving Problems by Searching ....

4 Informed Search and Exploration .

5 Constraint Satisfaction Problems ..

6 Adversarial Search ....

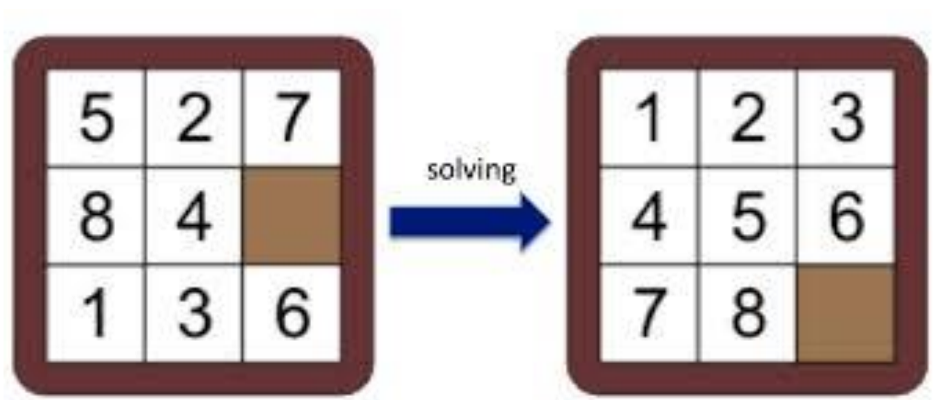
3 SOLVING PROBLEMS BY SEARCHING

4 Components

- The initial state
- Successor function
  - ***Successor-fn(x)***, where x is a particular state, returns a set of ***(action, successor)*** ordered pairs.
  - *Initial state* and *successor-fn* implicitly define a **state space**.
- Goal States
- Path Cost
  - Sum of ***step cost***s: $c(x, a, y)$

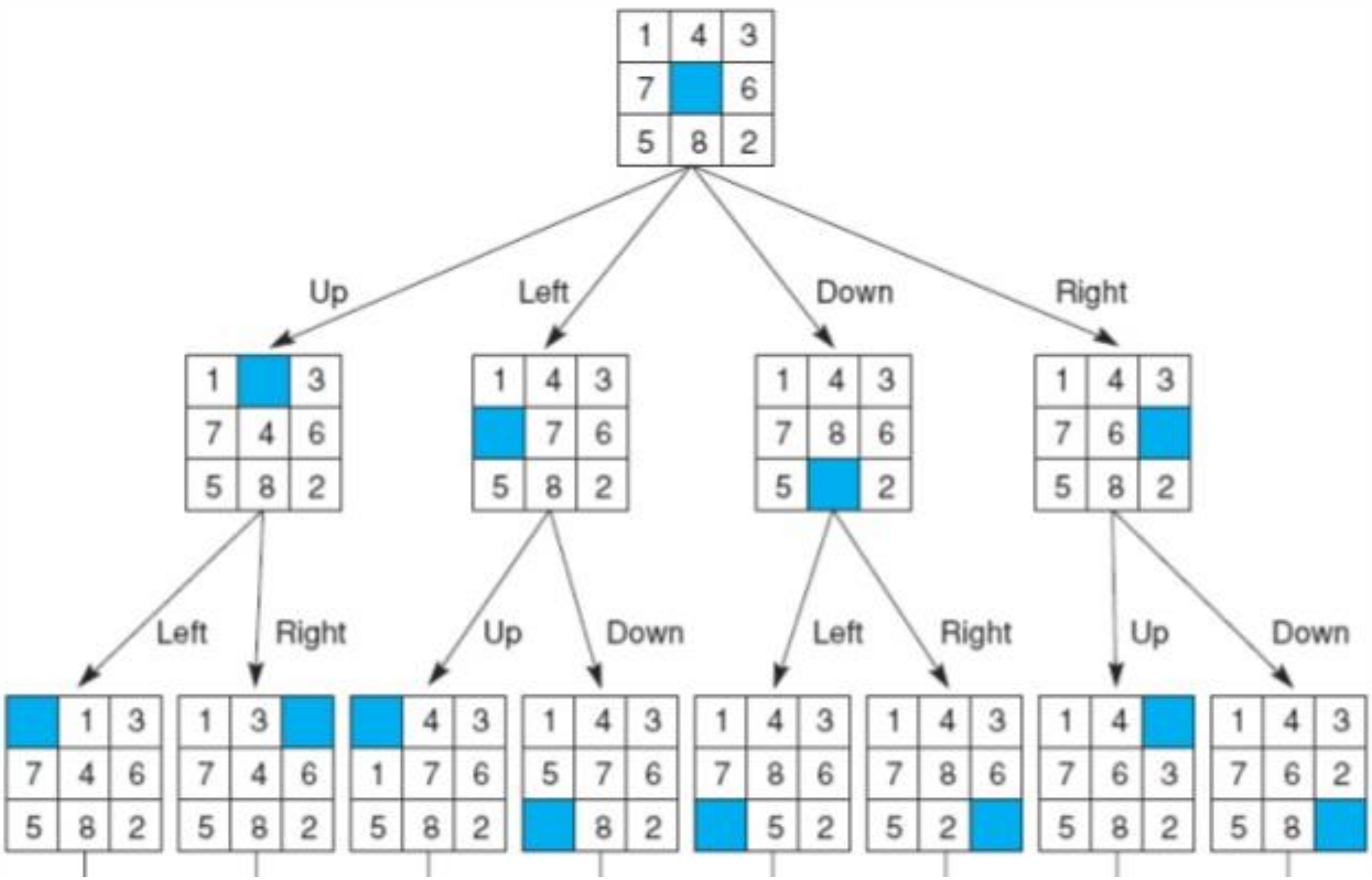
Sliding Block Puzzles:
8-Puzzle, 15-Puzzle

- Remember your childhood ☺



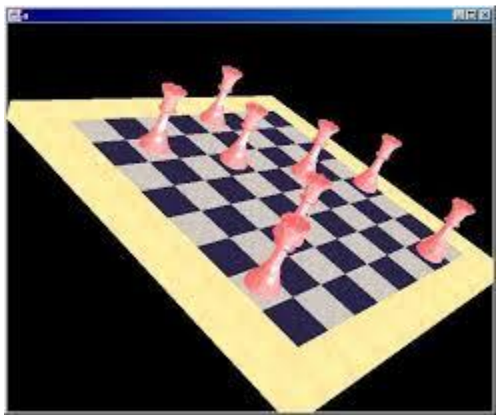
Start State **Goal State**

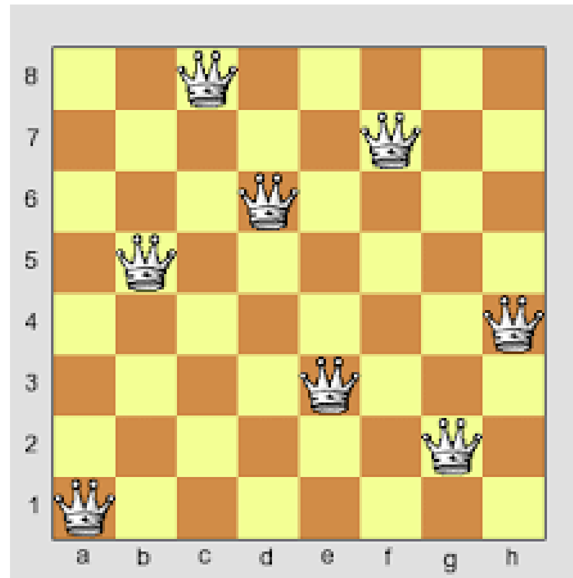
We want a path from start to goal – Planning Problem



We want a configuration. How you got this is not important!!

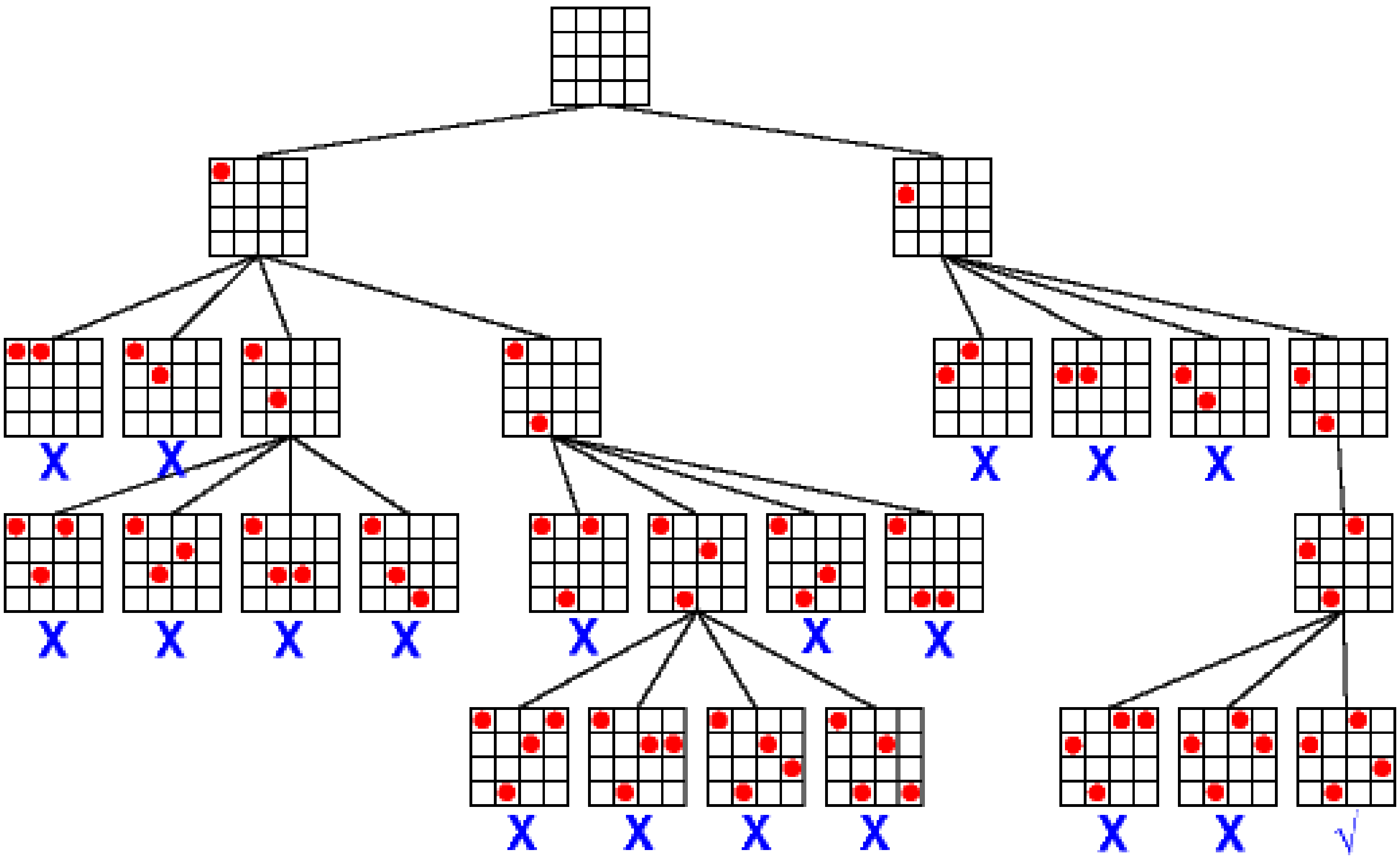
8-Queens Problem
(a Constraint Satisfaction Problem)



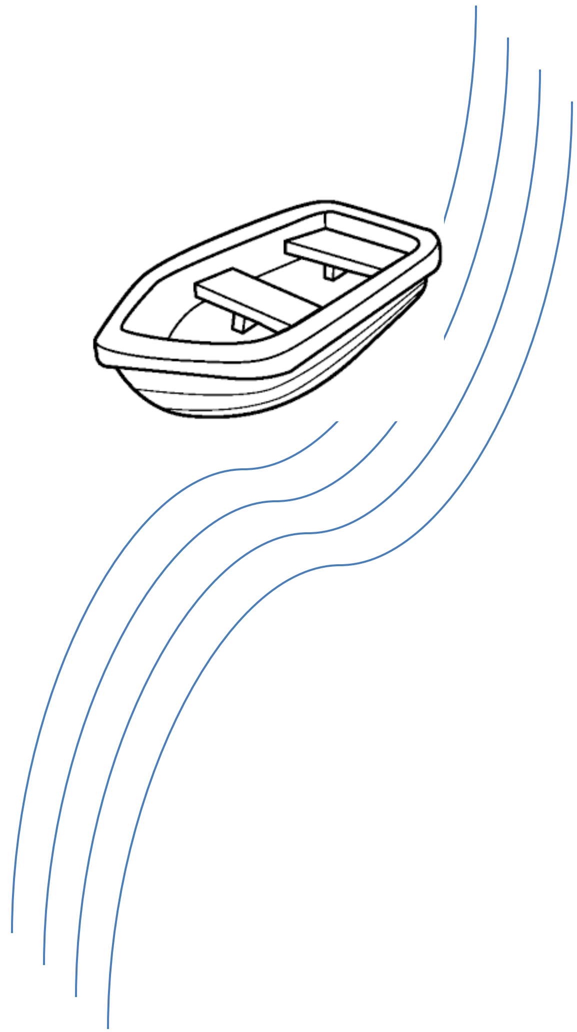


- We are not interested in the path, but only on the final configuration.
- So, goal is stated as something which obeys some properties. We do not give explicit goal state.

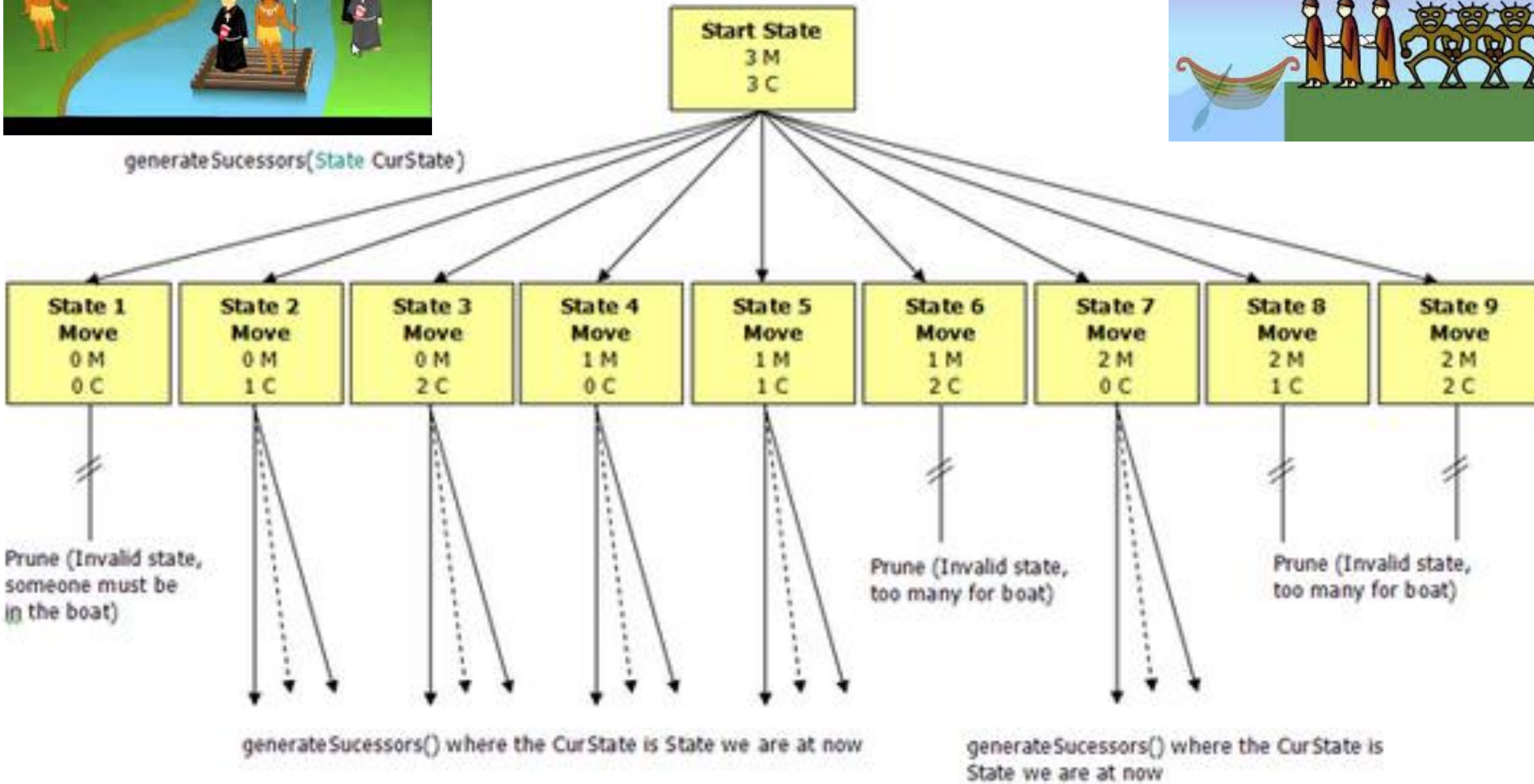
4-Queens State Space



Man, lion, goat, vegetables problem

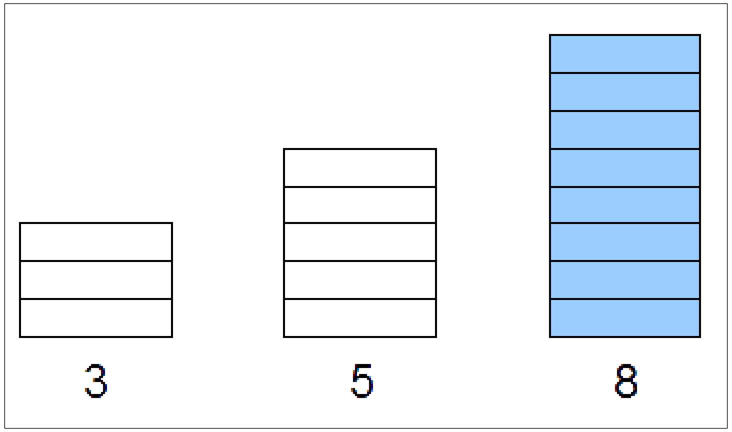


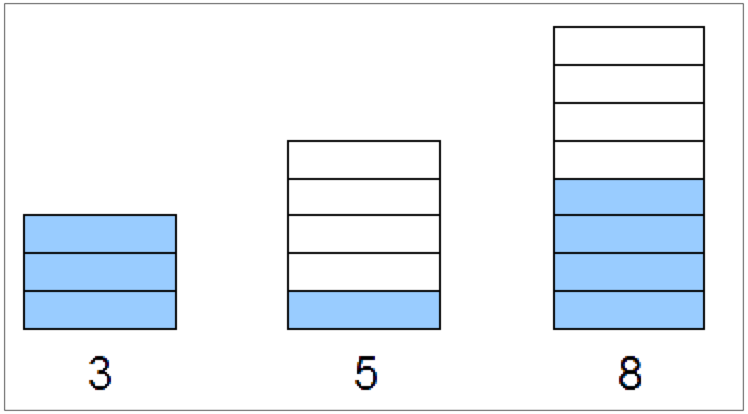
Missionaries and Cannibals Problem

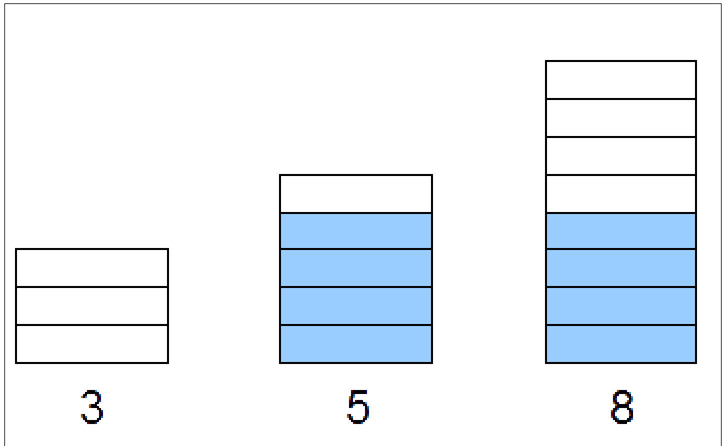


Water jugs problem

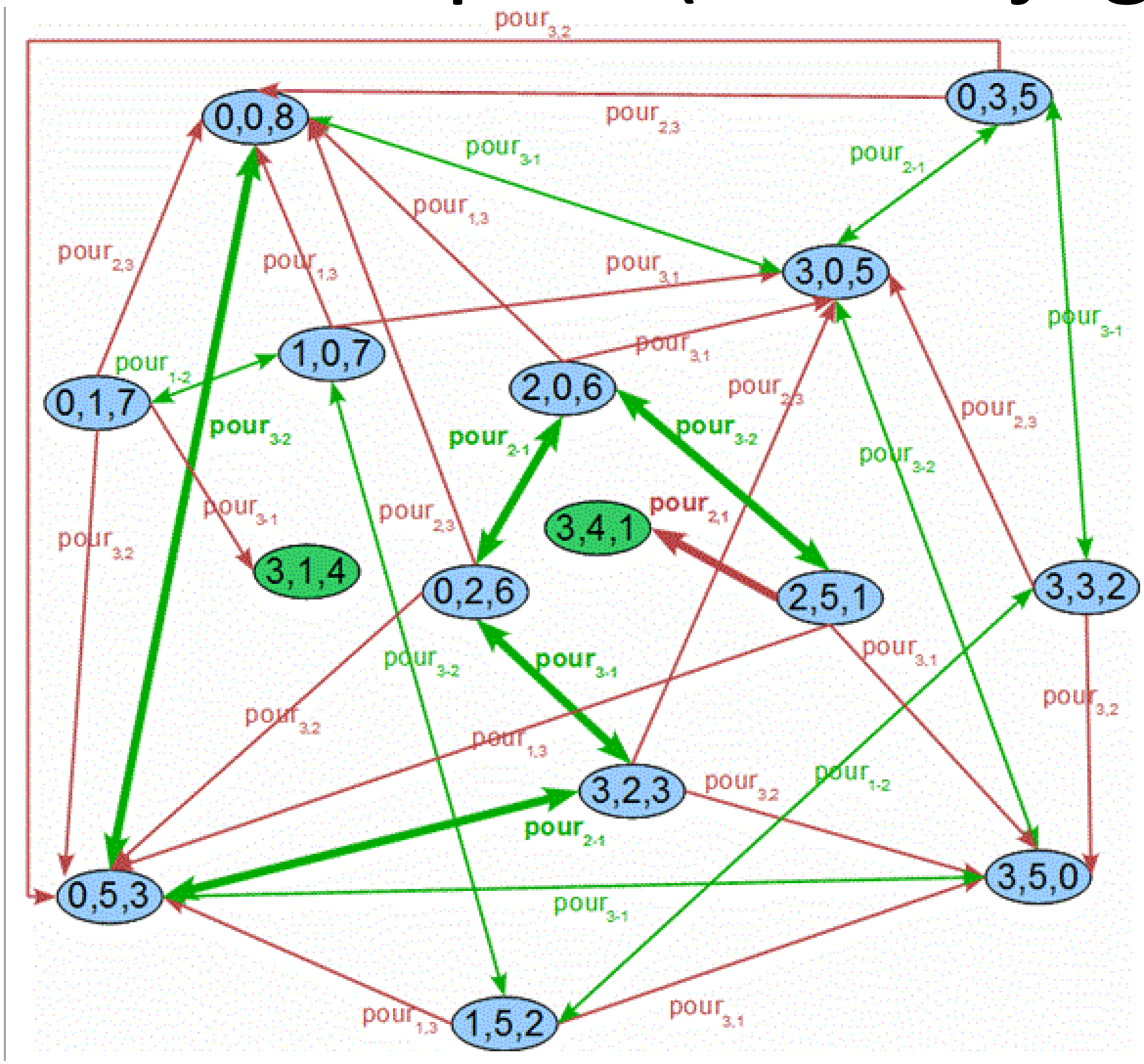
- Three jugs of capacity 3, 5 and 8 liters; 8 liters jug is with full of water; we want to take away 4 liters.







State space(Water jugs problem)

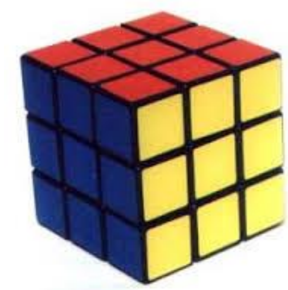


- Red edges are uni-directional.
- Green edges are bi-directional.
- State is represented as three numbers (self explanatory).

Source: http://www.tankonyvtar.hu/hu/tartalom/tamop425/0038_informatika_MestInt-EN/ch03s03.html

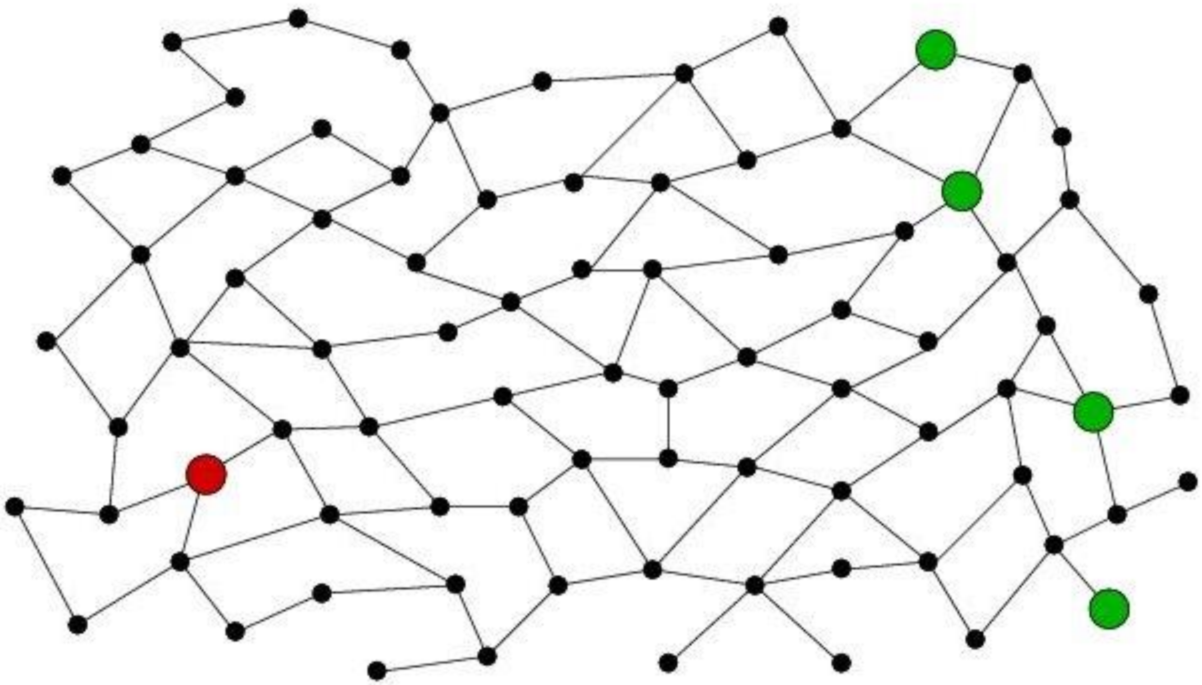
Rubik’s Cube

- How to represent a state
- How to represent a move



- We should be able to apply the general domain independent search strategy.

State space is a graph



- Often an implicit graph (*Start state*, *successor-fn* defines this graph)
- Red node is the start state, green nodes are goal nodes.

How actually it is done

There are many ways to represent nodes, but we will assume that a node is a data structure with five components:

- STATE: the state in the state space to which the node corresponds;
- PARENT-NODE: the node in the search tree that generated this node;
- ACTION: the action that was applied to the parent to generate the node;
- PATH-COST: the cost, traditionally denoted by $g(n)$, of the path from the initial state to the node, as indicated by the parent pointers; and
- DEPTH: the number of steps along the path from the initial state.

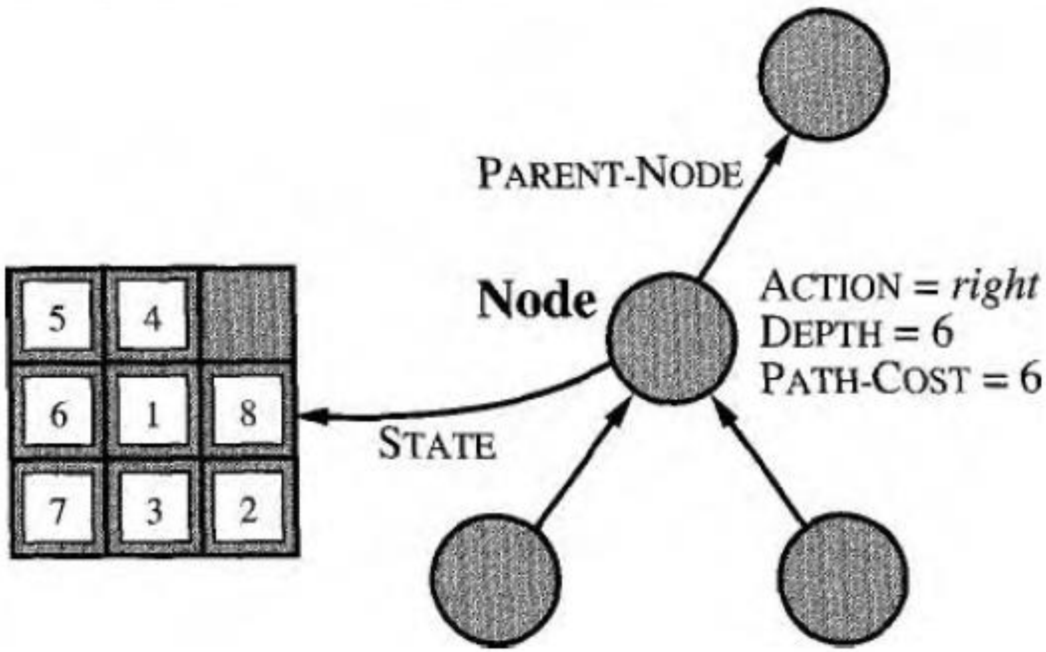


Figure 3.8 Nodes are the data structures from which the search tree is constructed. Each has a parent, a state, and various bookkeeping fields. Arrows point from child to parent.

Expand(current state or node)

- This will apply the *successor-fn*( ) with the current state and creates child nodes in the search tree.

General Search

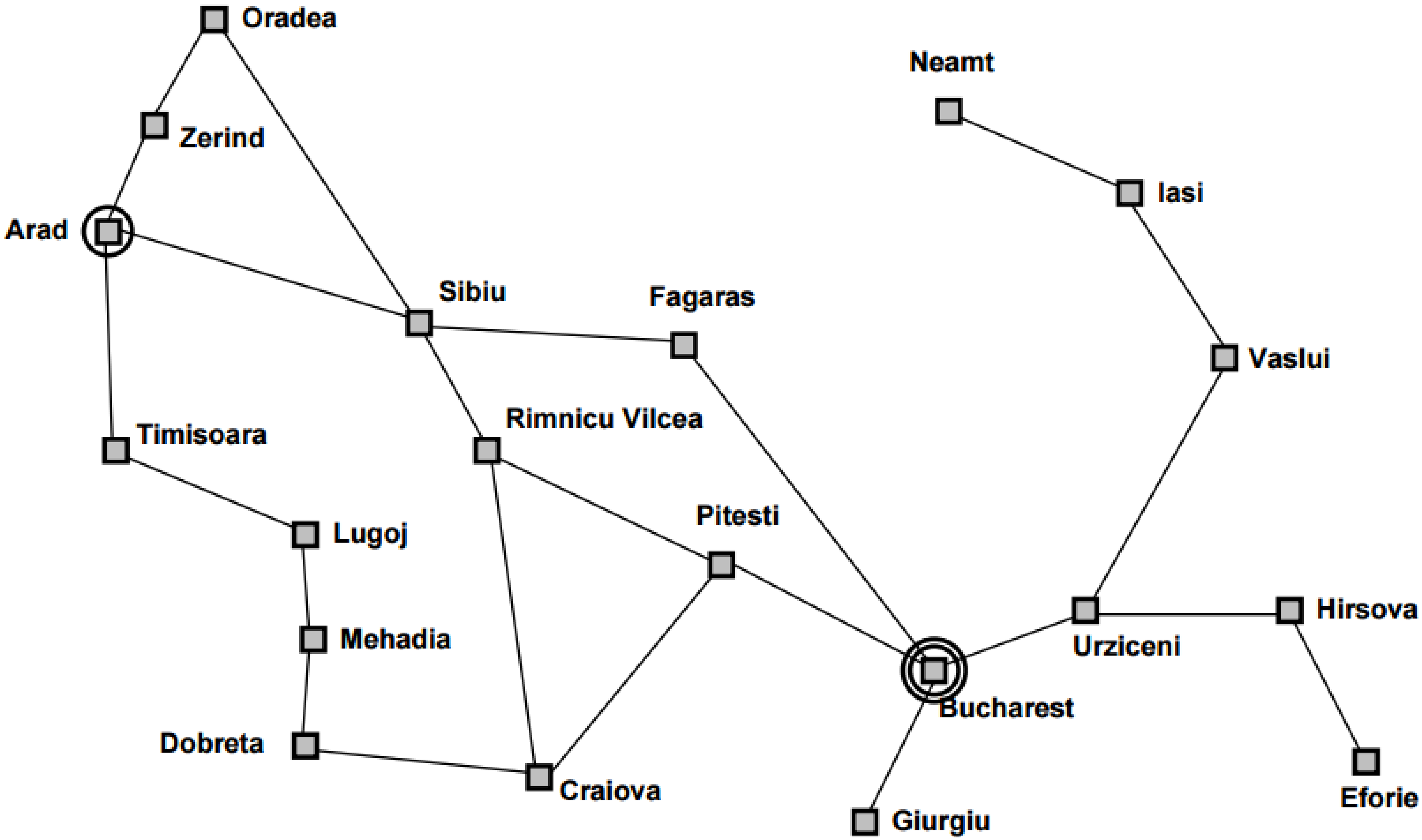
```
function GENERAL-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the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

- Strategy : Blind or Heuristic;

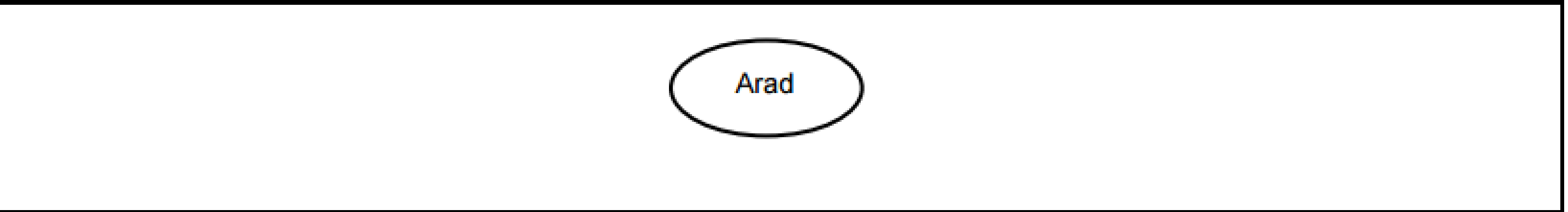
  BFS, DFS, Iterative Deepening, etc;

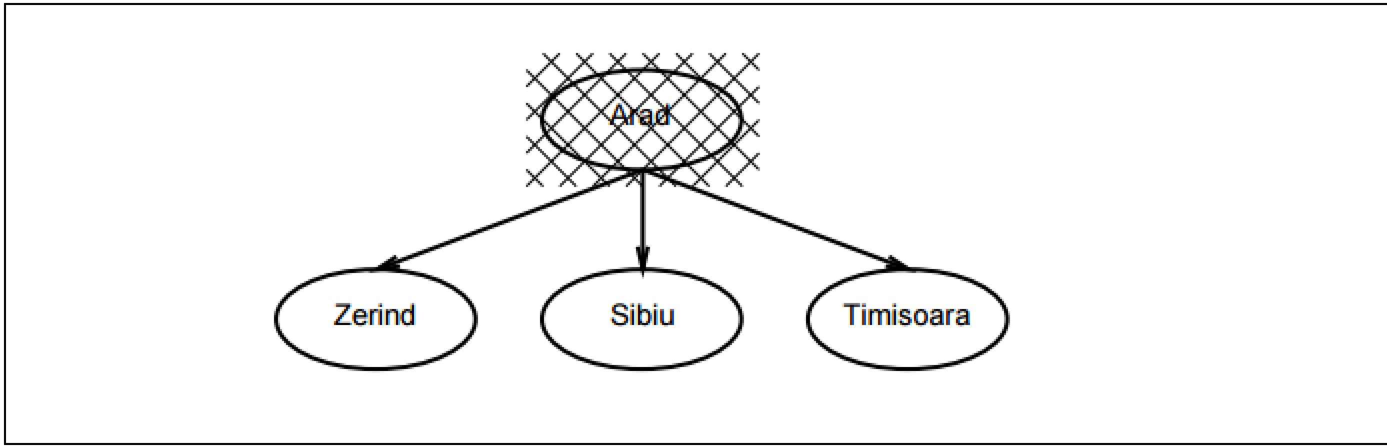
  Etc…

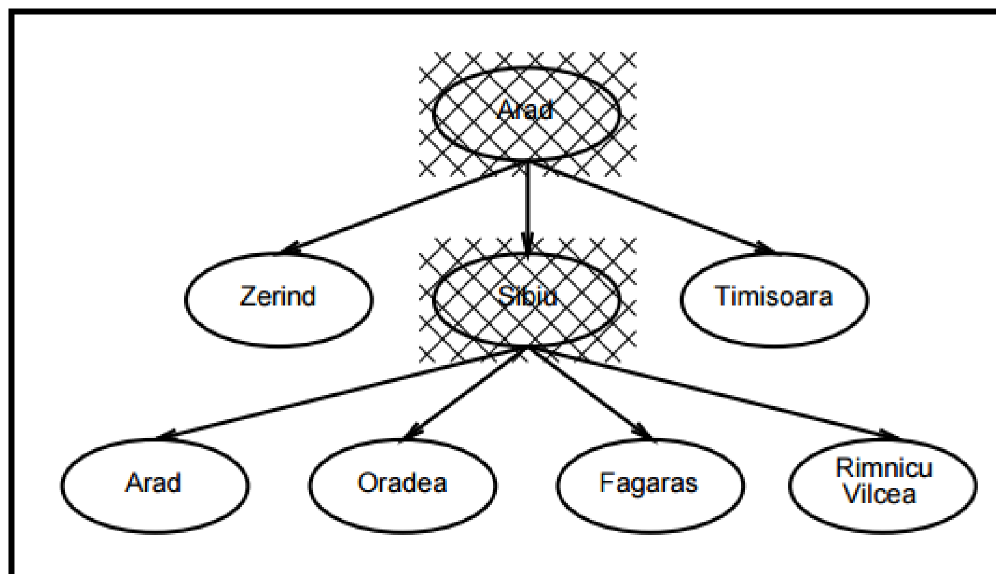
Example: Romania



General search example







State Space Vs Search Tree

- State space and search tree are different
- For the cities route finding there are only 20 states in the state space.
- But, there are infinite number of paths between any two cities. How?
- So, search tree has an infinite number of nodes.
- Good search algorithm avoids following repeated paths.

General-Search

```
PATH Search(STATE s, STATE g)  /* Path finding problem */
{
    OpenList = {s}; ClosedList = { };
    CurrentNode = Select(OpenList);
    Do{
        If (CurrentNode == goal) return (buildpath(CurrentNode, Closed));
        Else {
            ChildList = Expand(CurrentNode);
            RemoveDuplicates(ChildList, OpenList, ClosedList); /*Should be done carefully. Depends on the search
                                                                 strategy*/
            Add(ChildList, OpenList);
            CurrentNode = Select(OpenList)
        }
    }While(OpenList is not empty);
    return FAILURE;
}
```

How Select is done determines BFS, DFS, …

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

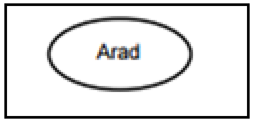
Uniform-cost search

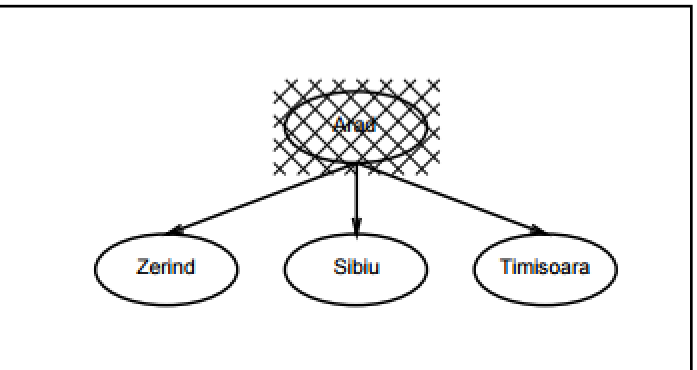
Depth-first search

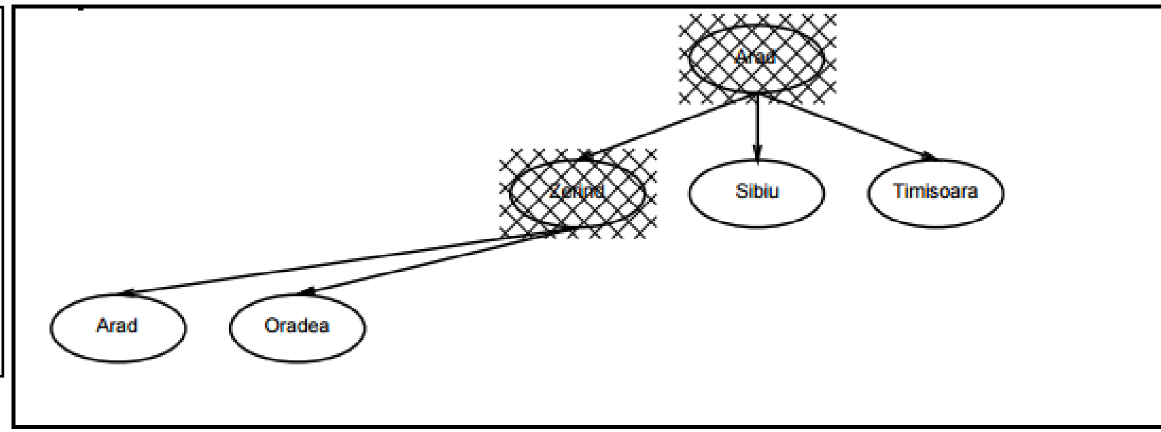
Depth-limited search

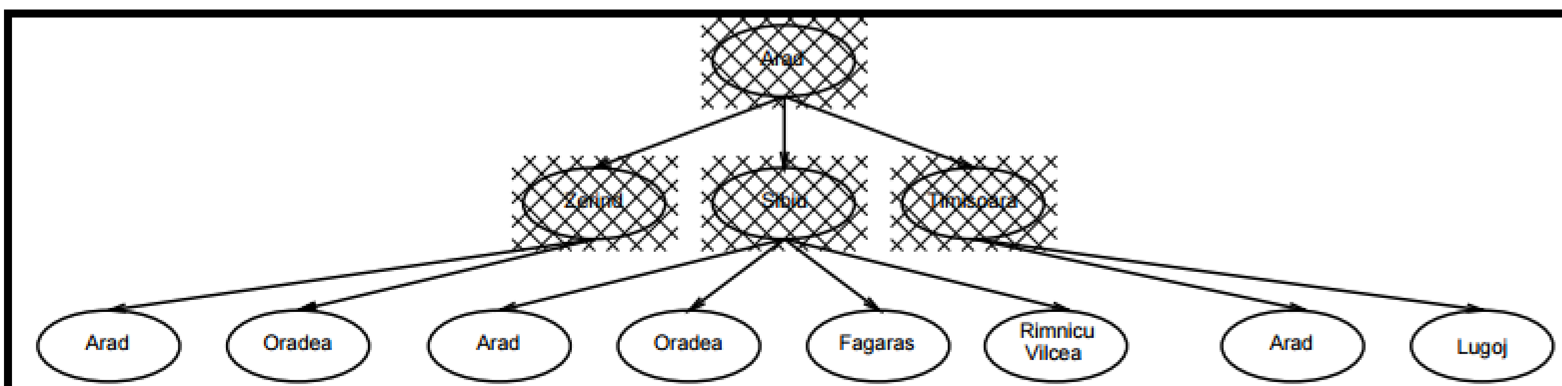
Iterative deepening search

BFS :
OpenList is maintained in a Queue









Properties of BFS

- Completeness: Yes (of course if b is finite).

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

Space is the big problem; can easily generate nodes at 1MB/sec
so 24hrs $=$ 86GB.

BFS: Memory & time req. exponential

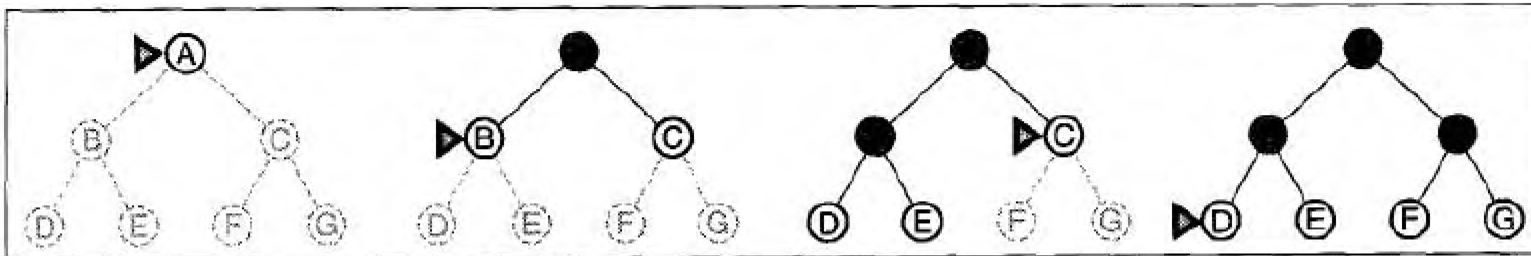


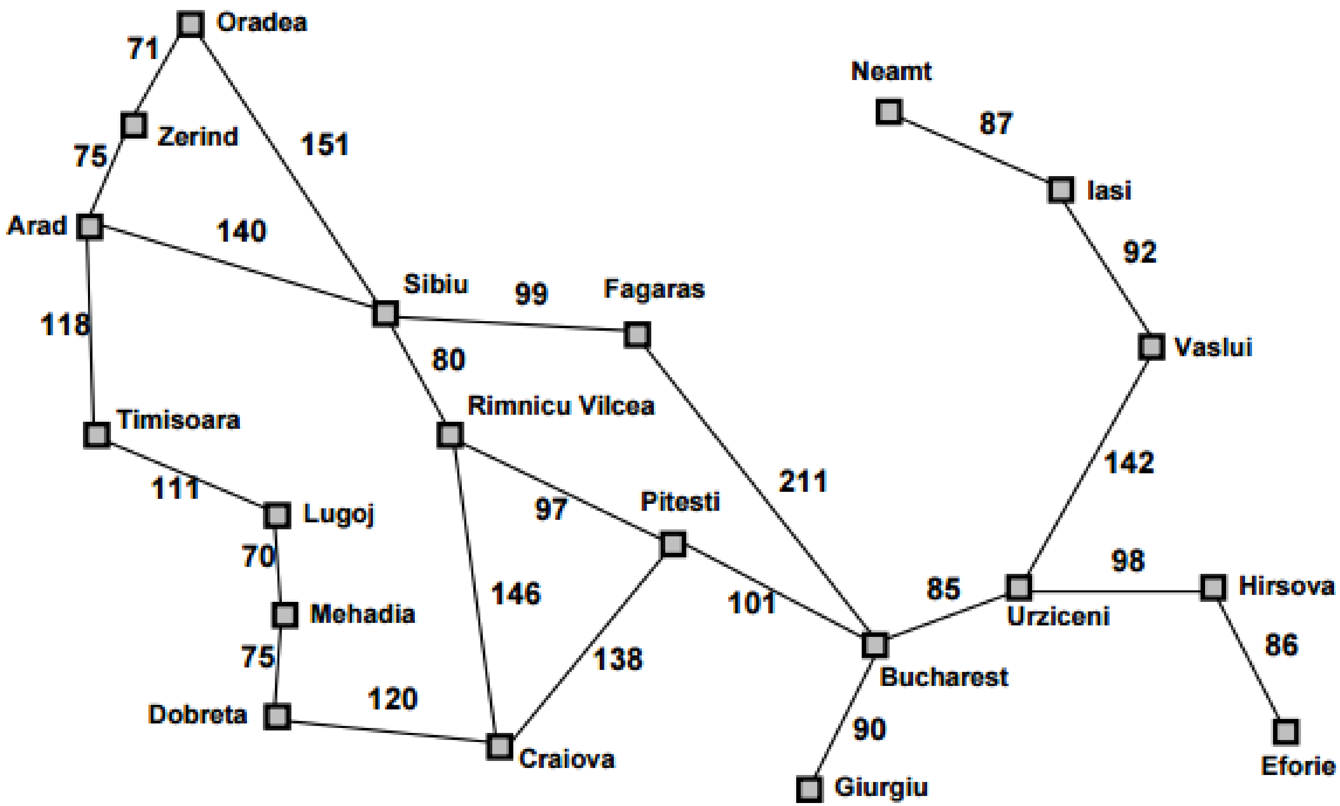
Figure 3.10 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

Depth	Nodes	Time	Memory
2	1100	.11 seconds	1 megabyte
4	111,100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabytes
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

Figure 3.11 Time and memory requirements for breadth-firstsearch. The numbers shown assume branching factor $b = 10$; 10,000 nodes/second; 1000 bytes/node.

- Exponential time and space complexity makes most of uninformed search methods suitable only for small problem.

Romania with step costs in km



Straight–line distance to Bucharest

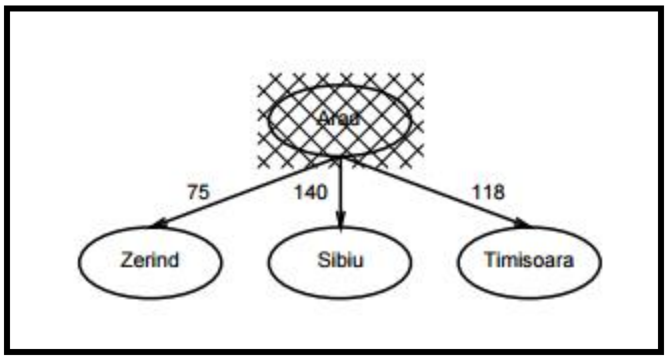
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

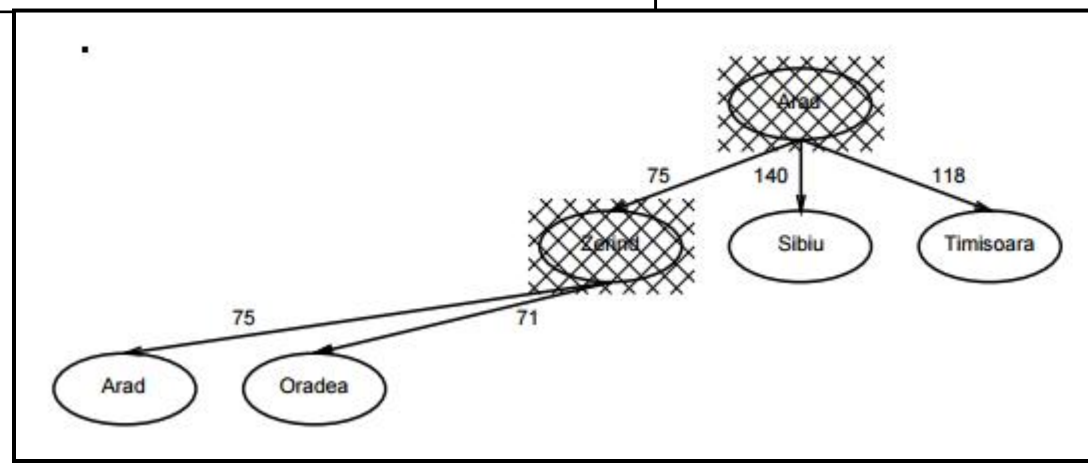
Uniform-cost search

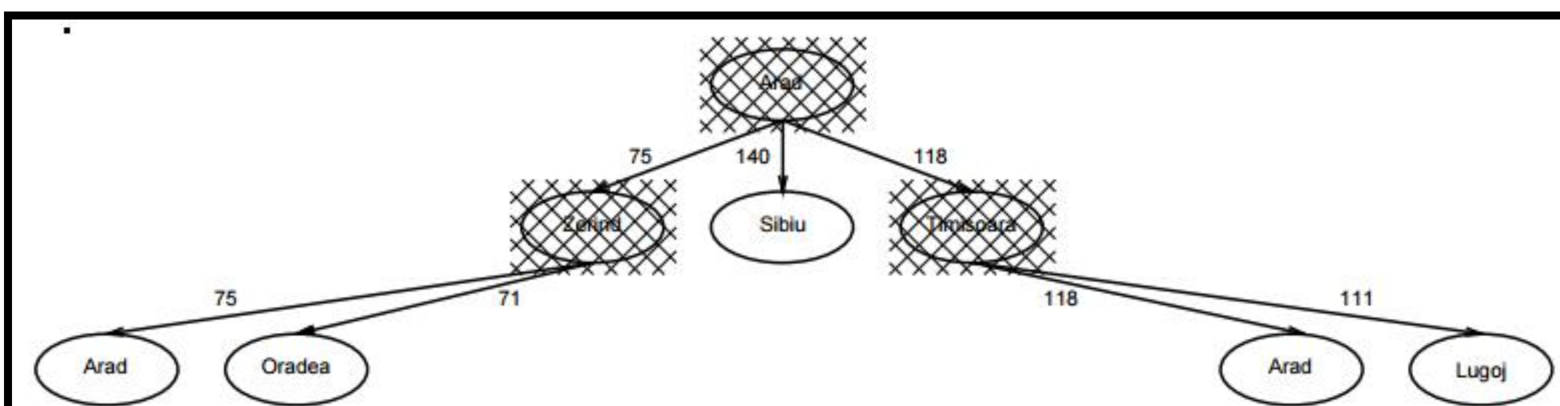
Expand least-cost unexpanded node

Implementation:

QUEUEINGFN = insert in order of increasing path cost







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

- If C^* is the cost of the optimal path, each edge has at-least ϵ cost, then the time complexity is $O\left(b^{1+\lfloor C^*/\epsilon \rfloor}\right)$, which can be much greater than b^d.

Dijkstra’s Algorithm Vs Uniform Cost Search

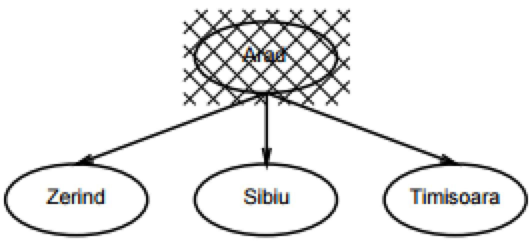
- Logically it is same.
- Dijkstra’s is for single source, all destinations.
- Uniform Cost Search is to a specific goal node.
- Dijkstra’s keep all nodes in OPEN (directly unreachable from the visited nodes are given ∞ cost).

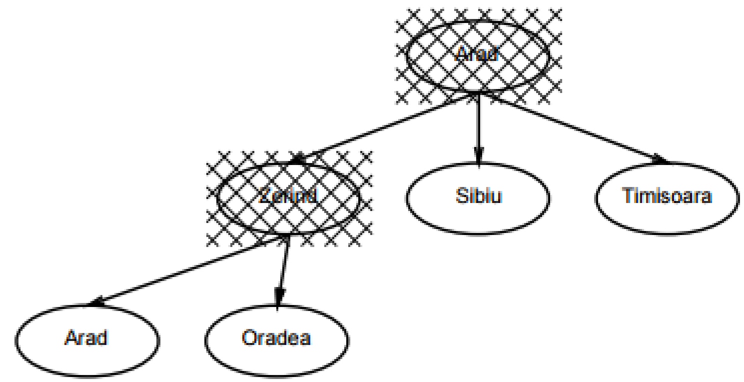
Depth-first search

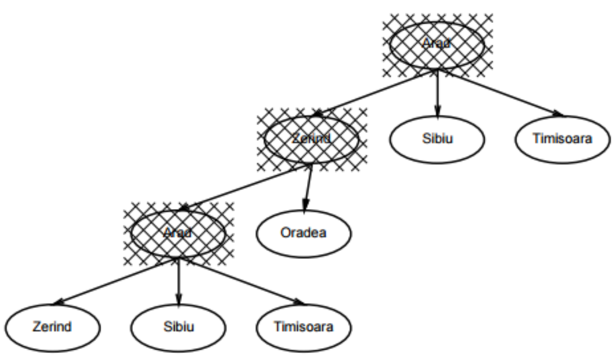
Expand deepest unexpanded node

Implementation:

QUEUEINGFN = insert successors at front of queue







Even with BFS this problem is there !

I.e., depth-first search can perform infinite cyclic excursions

Need a finite, non-cyclic search space (or repeated-state checking)

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

For Configuration problems.
For planning problems, it is same as BFS.

Depth-limited search

= depth-first search with depth limit l

Implementation:

Nodes at depth l have no successors

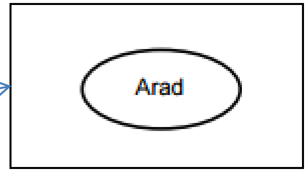
Iterative deepening search

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

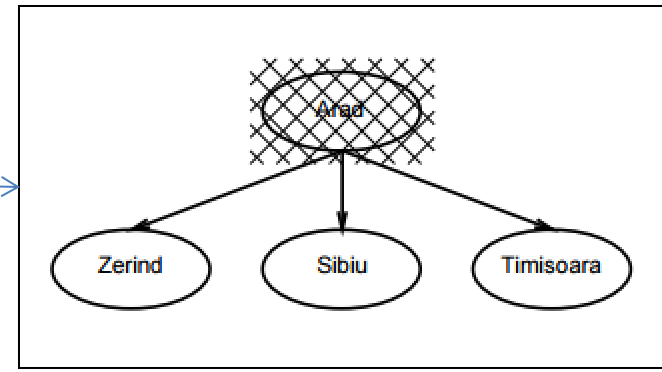
  for $depth \leftarrow 0$ **to** ∞ **do**
    $result \leftarrow$ DEPTH-LIMITED-SEARCH(*problem*, *depth*)
    if $result \neq$ cutoff **then return** *result*
  end

Iterative Deepening Search

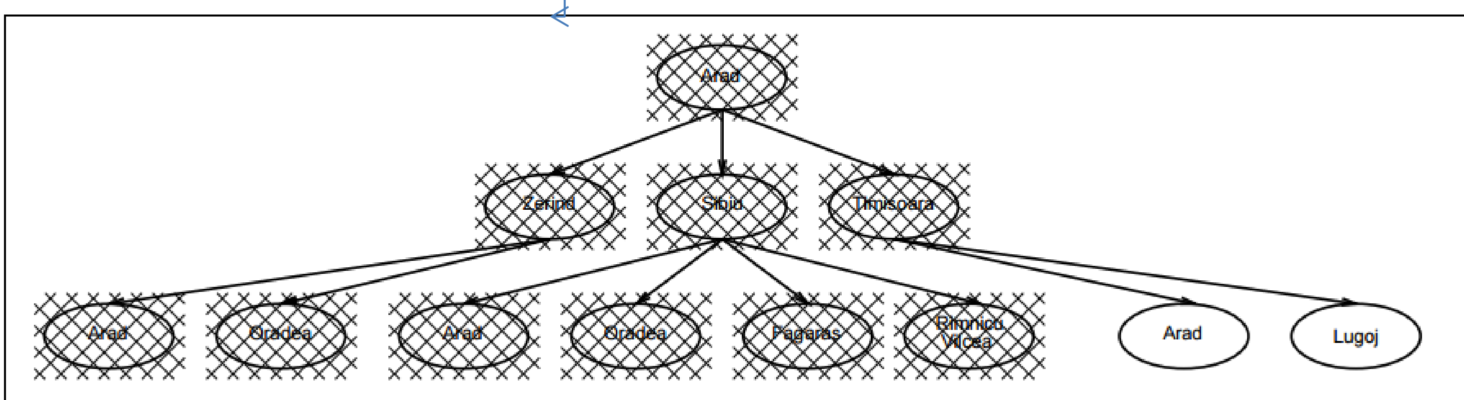
- $l = 0$



- $l = 1$



- $l = 2$



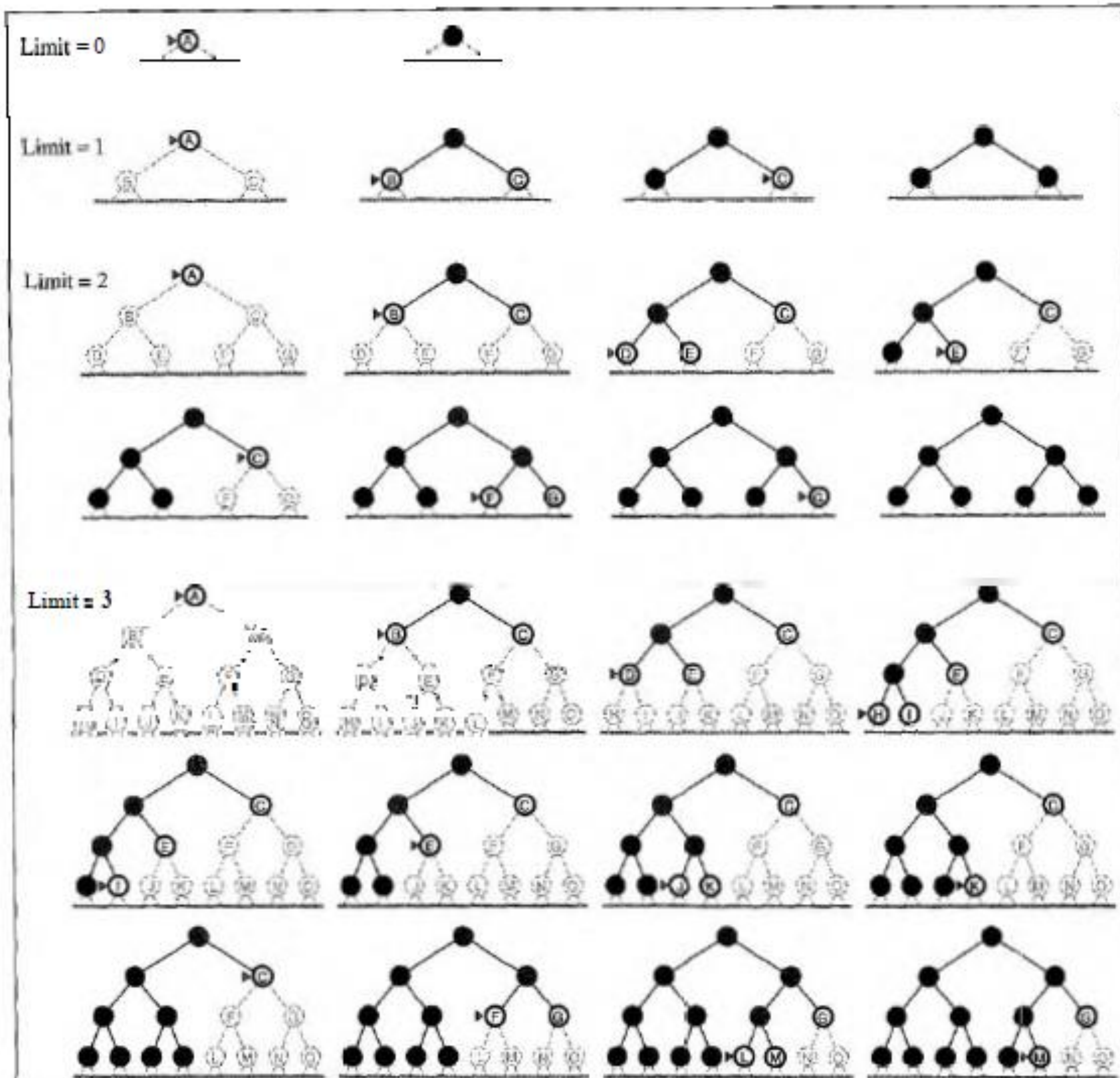


Figure 3.15 Four iterations of iterative deepening search on a binary tree.

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?? $O(b^d)$

Optimal?? Yes, if step cost = 1

Can be modified to explore uniform-cost tree

b—maximum branching factor of the search tree
d—depth of the least-cost solution
m—maximum depth of the state space (may be ∞)

$$N(\text{IDS}) = (d)b + (d-1)b^2 + \ldots + (1)b^d ,$$

which gives a time complexity of $O(b^d)$. We can compare this to the nodes generated by a breadth-first search:

$$N(\text{BFS}) = b + b^2 + \ldots + b^d + (b^{d+1} - b) .$$

Notice that breadth-first search generates some nodes at depth $d+1$, whereas iterative deepening does not. The result is that iterative deepening is actually *faster* than breadth-first search, despite the repeated generation of states. For example, if $b = 10$ and $d = 5$, the numbers are

$$N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$

$$N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100$$

In general, iterative deepening is the preferred uninformed search method when there is a large search space and the depth of the solution is not known.

Bidirectional Search

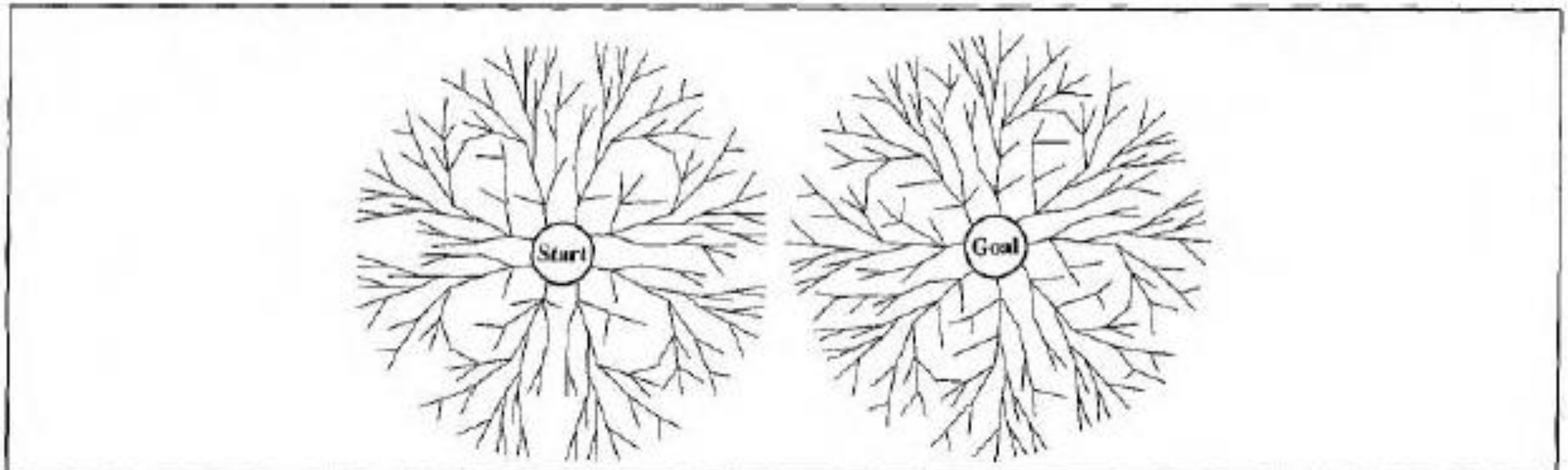


Figure 3.16 A schematic view of a bidirectional search that is about to succeed, when a branch from the start node meets a branch from the goal node.

The motivation is that $b^{d/2} + b^{d/2}$ is much less than b^d,

- **Need to maintain two OPEN and two CLOSED lists.**
- **Final solution path should be from START to GOAL.**

Duplicate Elimination

- This needs to be done for correctness and to avoid infinite recursion.
- In BFS, DFS, DFID
  - As soon as a node is expanded, before adding the resulting nodes to the OPEN list one has to verify whether it is already there either in the CLOSED or in the OPEN. If it is there, then do not add.
- For Uniform cost search
  - Least cost path needs to be retained. So which one to remove, either the new or the old has to be decided based on the cost.

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