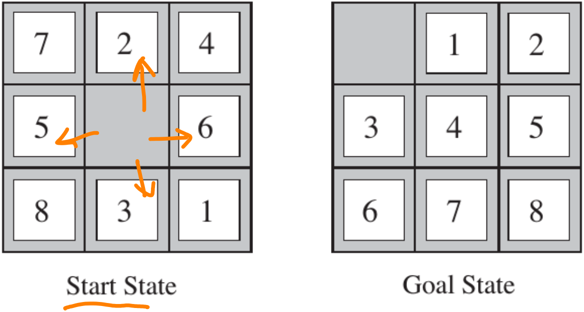
Eight puzzle problem



Searching for solutions

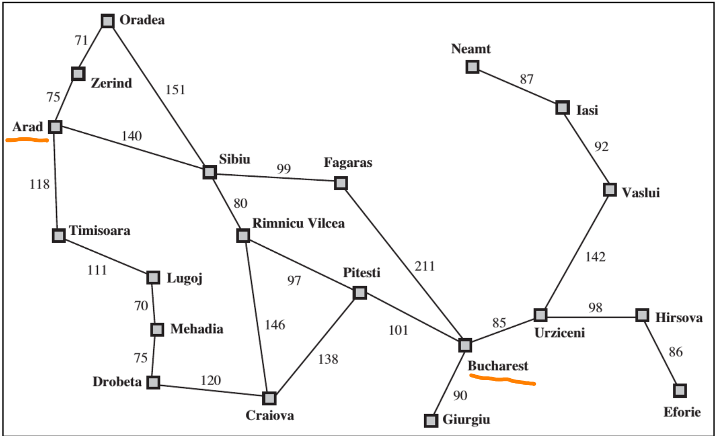


Figure 3.2 A simplified road map of part of Romania.

- Initial state $= In(Arad)$
- Goal state $= \{\ In(Bucharest)\ \}$

Searching for solutions

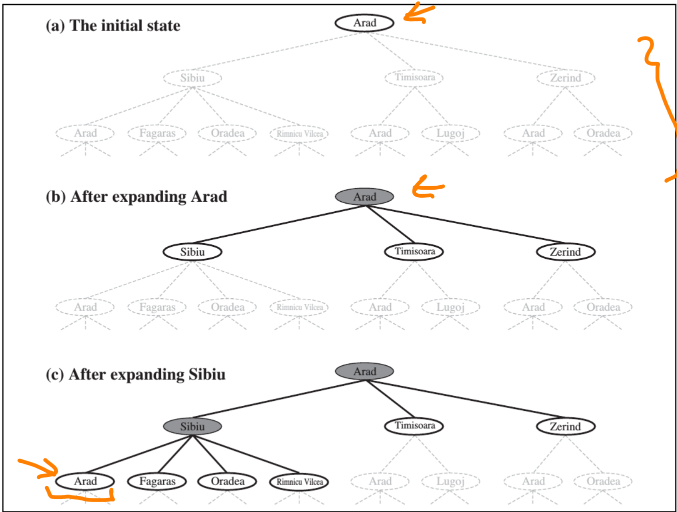


Figure 3.6 Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines.

Searching for solutions

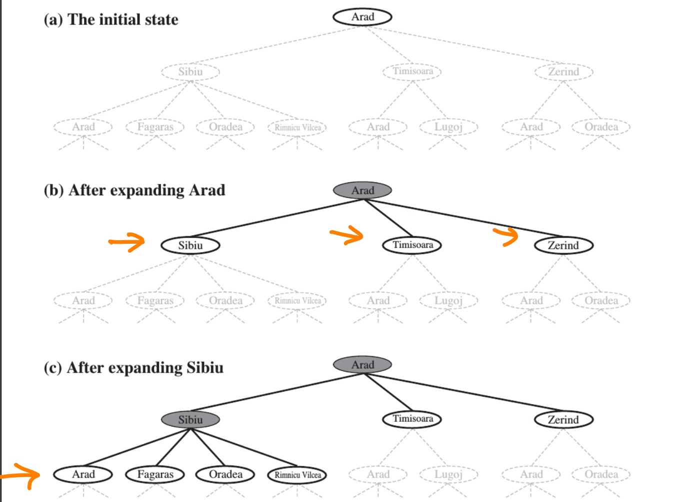


Figure 3.6 Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines.

- The frontier (after expanding Sibiu) includes Arad.

General Search Algorithm (Informal Description)

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier
```

```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set
```

Infrastructure for Search Algorithms

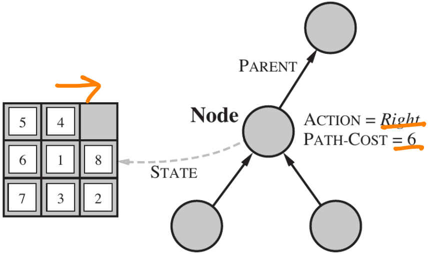
- Each node in the tree has four components:

  - n.STATE: the state in the state space to which the node corresponds;
  - n.PARENT: the node in the search tree that generated this node;
  - n.ACTION: the action that was applied to the parent to generate the node;
  - n.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.

- Each node in the tree has four components:

  - n.STATE: the state in the state space to which the node corresponds;
  - n.PARENT: the node in the search tree that generated this node;
  - n.ACTION: the action that was applied to the parent to generate the node;
  - n.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.

- Node:



Infrastructure for Search Algorithms

- We use a queue having following operations:
  - EMPTY?(*queue*) returns true only if there are no more elements in the queue.
  - POP(*queue*) removes the first element of the queue and returns it.
  - INSERT(*element*, *queue*) inserts an element and returns the resulting queue.
- Queue variants: LIFO queue, FIFO queue, Priority queue

Performance of an algorithm

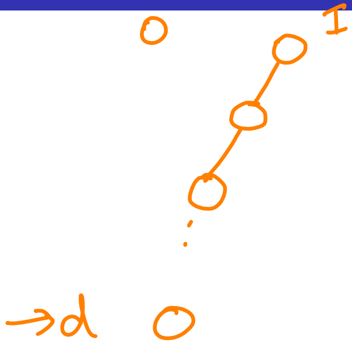
- Evaluating performance:
  - Completeness
  - Optimality

Performance of an algorithm

- Evaluating performance:
  - Completeness
  - Optimality
  - Time complexity
  - Space complexity

$|V|, |E|$

Performance of an algorithm



- Evaluating performance:
  - Completeness
  - Optimality
  - Time complexity
  - Space complexity
- Expressing complexity:
  - b, branching factor
  - d, depth of shallowest goal node
  - m, maximum length of any path in state space

Uninformed Search Strategies

- No information beyond what is provided by the problem definition.
- Breadth-first Search

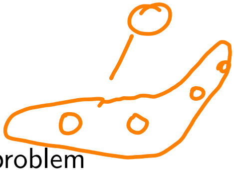
Uninformed Search Strategies

- No information beyond what is provided by the problem definition.
- Breadth-first Search
  - FIFO queue for the frontier nodes

Uninformed Search Strategies

- No information beyond what is provided by the problem definition.
- Breadth-first Search
  - FIFO queue for the frontier nodes
  - Goal test is performed when a node is generated, and not when it is selected from the queue for expansion.

Uninformed Search Strategies



- No information beyond what is provided by the problem definition.
- Breadth-first Search
  - FIFO queue for the frontier nodes
  - Goal test is performed when a node is generated, and not when it is selected from the queue for expansion.
  - BFS always has the shallowest path to the nodes in the frontier.

Breadth-first Search

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)   /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier ← INSERT(child, frontier)
```

Breadth-first Search

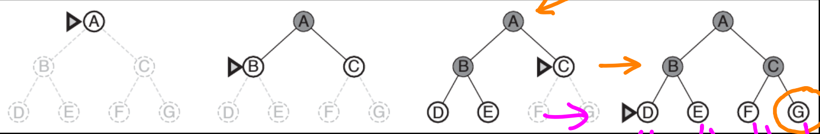


Figure 3.12 Breadth-first search on a simple binary tree. At each stage the node to be expanded next is indicated by a marker.

$$3^0 + 3^1 + 3^2 = 13 < 3^3$$

$$13 + 3^3 = 40$$

$$b^{d+1} > b^0 + b^1 + \dots b^d$$

Performance of Breadth-first Search

- Is it complete?

Performance of Breadth-first Search

- Is it complete?
- Is it optimal?

Performance of Breadth-first Search

- Is it complete?
- Is it optimal?
- Breadth-first search is optimal if the path cost is a nondecreasing function of the depth of the node [P. 82].

$path(n) \overset{d}{=} \underbrace{f(d)}$ $\quad\quad \underbrace{f(d+1) \geq f(d)}$

Performance of Breadth-first Search

- Is it complete?
- Is it optimal?
- Breadth-first search is optimal if the path cost is a nondecreasing function of the depth of the node [P. 82].
- Time complexity will be measured in terms of number of nodes generated:

  $$b + b^2 + b^3 + \ldots + b^d = O(b^d)$$

- Space complexity will be measured in terms of number of nodes expanded (*explored* set) and number of nodes generated (in the *frontier queue*)

Performance of Breadth-first Search

- Is it complete?
- Is it optimal?
- Breadth-first search is optimal if the path cost is a nondecreasing function of the depth of the node [P. 82].
- Time complexity will be measured in terms of number of nodes generated:

  $$b + b^2 + b^3 + \ldots + b^d = O(b^d)$$

- Space complexity will be measured in terms of number of nodes expanded (*explored* set) and number of nodes generated (in the *frontier queue*)
  - Size of *explored* set $= b^0 + b^1 + b^2 + \ldots + b^{d-1} = O(b^{d-1})$

Performance of Breadth-first Search

- Is it complete?
- Is it optimal?
- Breadth-first search is optimal if the path cost is a nondecreasing function of the depth of the node [P. 82].
- Time complexity will be measured in terms of number of nodes generated:

  $$b + b^2 + b^3 + \ldots + b^d = O(b^d)$$

- Space complexity will be measured in terms of number of nodes expanded (*explored* set) and number of nodes generated (in the *frontier queue*)
  - Size of *explored* set $= b^0 + b^1 + b^2 + \ldots + b^{d-1} = O(b^{d-1})$
  - Size of *frontier* $= b^d$

Performance of Breadth-first Search

- Is it complete?
- Is it optimal?
- Breadth-first search is optimal if the path cost is a nondecreasing function of the depth of the node [P. 82].
- Time complexity will be measured in terms of number of nodes generated:

  $$b + b^2 + b^3 + \ldots + b^d = O(b^d)$$

- Space complexity will be measured in terms of number of nodes expanded (*explored* set) and number of nodes generated (in the *frontier queue*)
  - Size of *explored* set $= b^0 + b^1 + b^2 + \ldots + b^{d-1} = O(b^{d-1})$
  - Size of *frontier* $= b^d$
  - Space complexity is $O(b^d)$.

Complexity exponential in d

- Exponential complexity $O(b^d)$ leads to the following growth in time and space requirements:

Complexity exponential in d

15

24

- Exponential complexity $O(b^d)$ leads to the following growth in time and space requirements:

50

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.