Artificial Intelligence
Unit 2-1 Search Strategies

BE CSE VII semester

Engels. R

Engels. R
Engels. R
Engels. R
Engels. R
Engels. R
Engels. R
Engels. R
Engels. R
Engels. R

Unit 2 – Search Strategies

- **SEARCH STRATEGIES** (12)
  - Breadth-First Search
  - Uniform Cost Search
  - Depth-First Search
  - Depth-Limited Search
  - Iterative Deepening Search
  - Bidirectional Search
- **Heuristic Search Techniques**
  - A* Search
  - AO* Algorithm
- **Adversarial Search:**
  - Minimax Algorithm
  - Alpha beta Pruning

Engels. R

Problem solving by searching

- **Problem solving agent**
  - Atomic representations
  - no internal structure visible to the problem solving algorithms
- Planning agents
  - Goal based agents
  - use more advanced **factored** or **structured** representations
- Search algorithms
  - Uninformed: only problem definition. No other information. Usually inefficient
  - Informed: given some guidance about solution, usually better than uninformed

Task environment assumption (simplest yet useful) : Solution to a problem is always a fixed sequence of actions

Engels. R

Goal Formulation

- Goal
  - a set of environment/world states where goal is satisfied
- Goal formulation
  - First step in problem solving
  - Based on the current situation and the agent's performance measure
  - What are the goals?
  - Goals help organize behaviour
    - by limiting objectives that the agent is trying to achieve
    - and hence limits the actions it needs to consider

Engels. R

Problem formulation

- Question: What actions to perform to reach a goal state?
  - First, need to decide what actions and states need to be considered!
- Problem formulation
  - Process of deciding what actions and states to consider, given a goal
- Can agent with several immediate options of unknown value (relation to goal state) decide what to do ?
  - Yes, by first examining future actions that eventually lead to states of known value
- Assumptions: Observable, Discrete, Known and Deterministic

Engels. R

Problem formulation assumptions

Properties of Task Environments for Problem Solving Agent

- Observable
  - Agent always knows the current state
- Discrete
  - Any given state there are only finitely many actions to choose from
- Known
  - agent knows which states are reached by each action
- Deterministic
  - each action has exactly one outcome

*Under these assumptions, the solution to any problem is **a fixed sequence of actions***

*So can be solved by a **search approach***

Engels. R

Search algorithms

Search

The process of looking for a sequence of actions that reaches the goal

Takes a problem as input and returns a solution in the form of an action sequence

- Formulate a goal
- Formulate a problem to solve
- Call a search procedure to solve the problem(s)
- If a solution is found, it is used to guide actions in **execution** phase
- Agent removes first step from the sequence
  - Usually after performing the first action of the sequence
- The agent will formulate a new goal
  - Once the solution has been executed

Design of Search algorithms

1. Formulate
2. Search
3. Execute

Engels. R

Simple Problem Solving Agent

function **SIMPLE-PROBLEM-SOLVING-AGENT**(percept) returns an action

persistent **seq,** an action sequence, initially empty
state, some description of the current world state
goal, a goal, initially null
problem, a problem formulation

state **UPDATE-STATE**(state, percept)
if seq is empty then
goal <- **FORMULATE-GOAL**(state)
problem <— **FORMULATE-PROBLEM(** state, goal)
seq <- **SEARCH**(problem)
if seq = failure then return a null action
action <— FIRST(seq)
seq <— REST(seq)
return action

Formulates a goal and a problem
Searches for a sequence of actions that would solve the problem,
Executes the actions one at a time

When this is complete, agent formulates another goal and starts over

Open loop system as agent ignores percepts when selecting an action in solution sequence

Ignoring the percepts breaking the loop between agent and environment

Engels. R

Five components of problem definition

- **Initial state:** State agent starts in (***Initial state***)
- Description of the possible **Actions** available
  - given a particular state s,
  - **ACTIONS (s)** - returns the set of actions that can be executed in s
  - Each of these actions is APPLICABLE in s
- **Transition Model**
  - A description of what each action does specified by a function ***Result(s, a)*** that returns the state that results from doing action a in state s
  - **Successor**
    - Refers to any state reachable from a given state by a single action.

Engels. R

Five components of problem definition (2)

- **Goal state:**
  - Determines whether a given state is a goal state
  - Either explicit set of possible goal states or
  - specified by an abstract property (checkmate in Chess)
- **Path Cost**
  - Assigns a numeric cost to each path
  - Agent chooses a cost function that reflects its own performance measure
  - Simplifying assumptions
    - cost of a path can be described as the ***sum*** of the costs of the individual actions along the path
    - Nonnegative costs
- Usually the five components are gathered into a single data structure

Engels. R

State-space and other definitions

- **Solution**: action sequence from initial state to goal state
- **Solution quality**: measured by the path cost function
- **Optimal solution:** has the lowest path cost among all solutions
- The initial state, actions, goal state and transition model implicitly define the **state space** of the problem together
  - The set of all states reachable from the initial state by any sequence of actions
- The state space forms a directed network or **graph**
  - in which the nodes are states and the links between nodes are actions.
- A **path** in the state space
  - a sequence of states connected by a sequence of actions
  - Step cost of taking action *a* in state *s* to reach state *s'* = **e(s,a,s')**

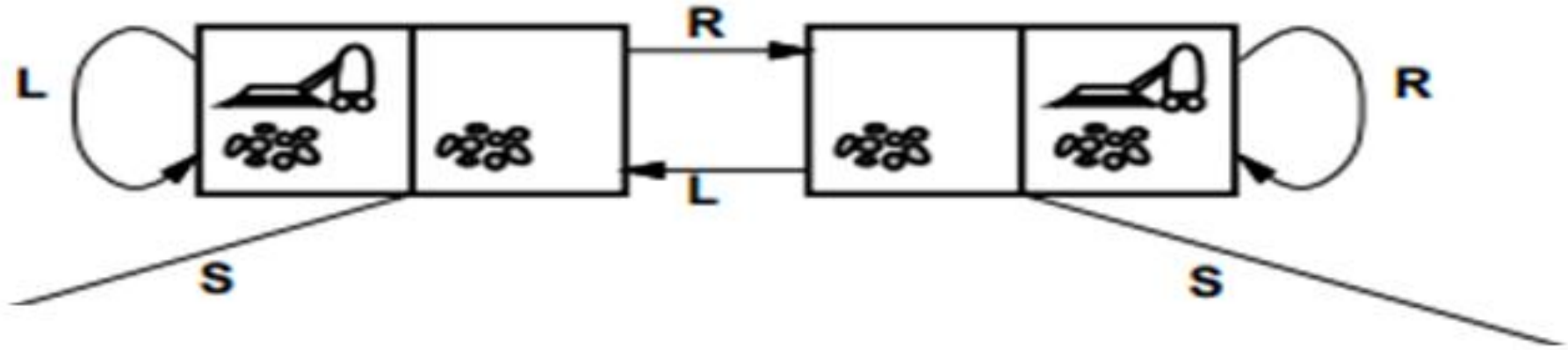
Engels. R

Problem types

- Toy problem
  - Intended to illustrate /exercise various problem solving methods
  - Can be given a concise, exact description
  - Hence performances of algorithms can be compared
- Real-world problem
  - one whose solutions people actually care about
  - Such problems tend not to have a single agreed-upon description
  - We can give the general flavour of their formulations.

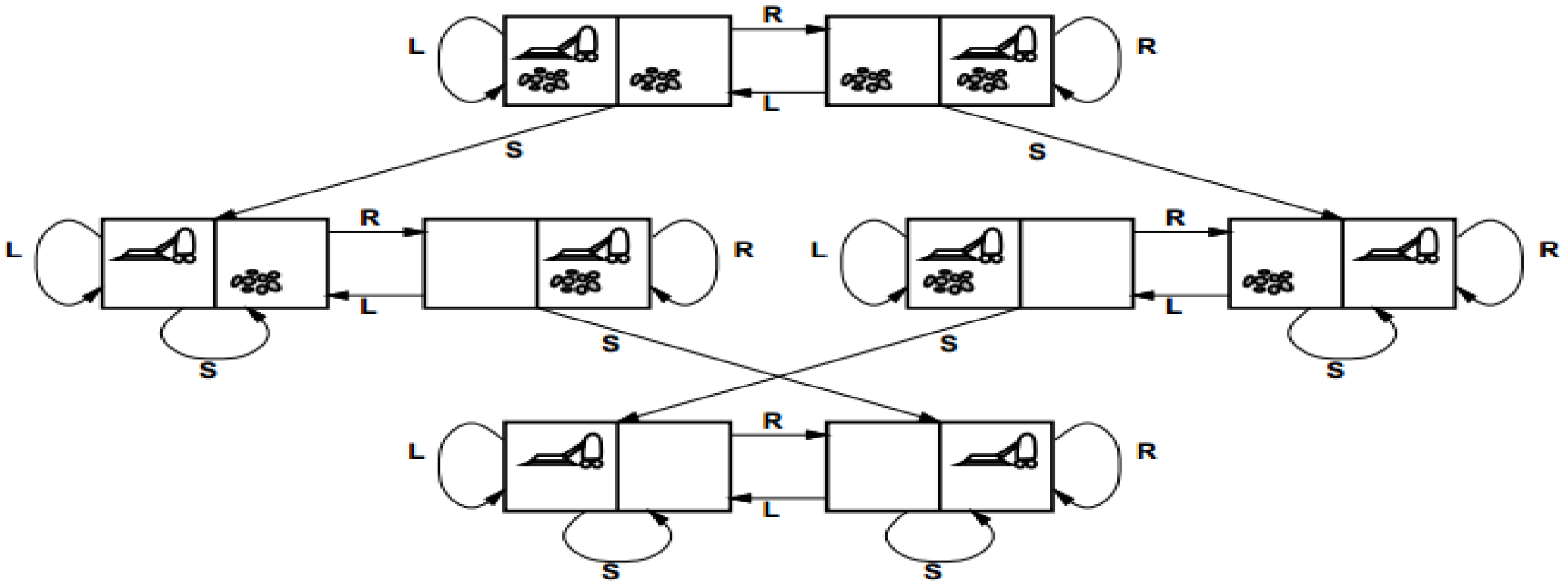
Engels. R

Abstraction



Engels. R
Engels. R
Engels. R
Engels. R
Engels. R
Engels. R

Abstraction



Engels. R

Toy Problem Example: Vacuum World

- Take 5 minutes to write down the following for the vacuum world toy problem

- **States**
- **Initial state:**
- **Actions:**
- **Transition model:**
- **Goal test:**
- **Path cost:**

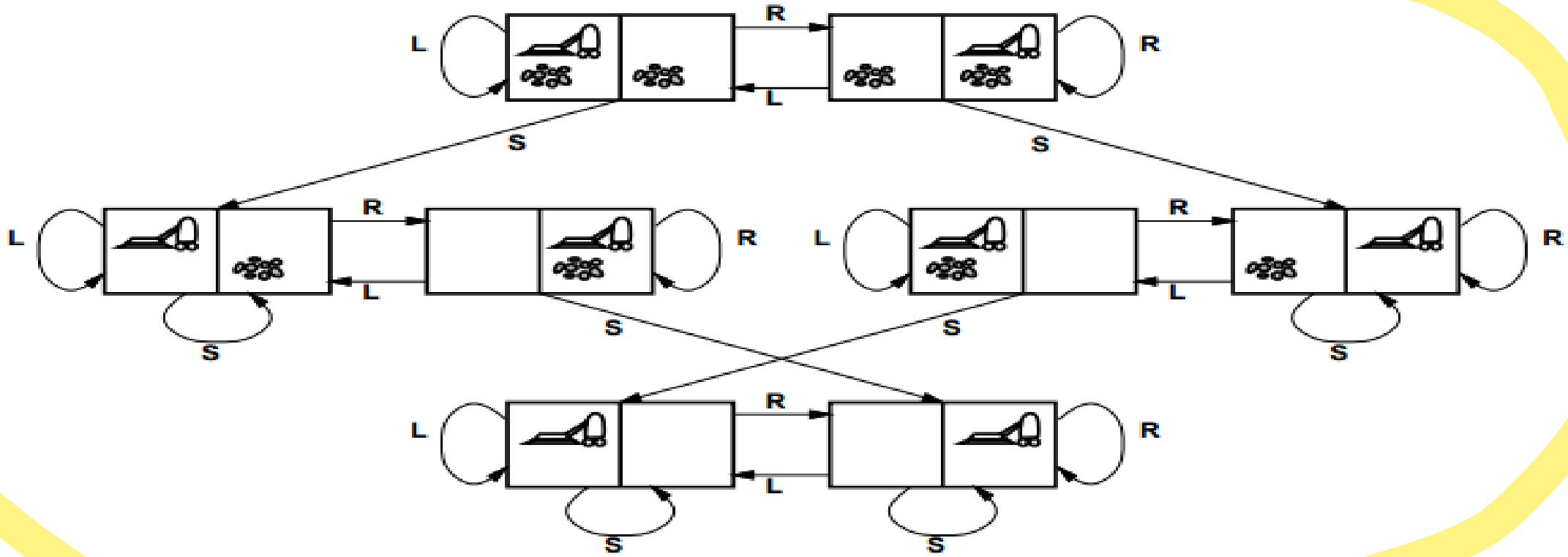
Engels. R

Toy Problem Example: Vacuum World

- **States**
  - State is determined by both the agent location and the dirt locations
  - The agent is in one of two locations, each of which might or might not contain dirt
  - There are $2 \times 2^2 = 8$ possible world states
    - A larger environment with n locations has $n2^n$ states.
- **Initial state:** Any state can be designated as the initial state
- **Actions:** Each state has just three actions: Left, Right, and Suck
  - Larger environments might also include Up and Down
- **Transition model**
  - The actions have their expected effects, except that
    - moving **Left** in the leftmost square,
    - moving **Right** in the rightmost square, and
    - Sucking in a clean square have no effect.
- **Goal test:** This checks whether all the squares are clean.
- **Path cost:** Each step costs 1, so the path cost is the number of steps in the path.

Engels. R

Abstraction



states??: integer dirt and robot locations (ignore dirt amounts etc.)
actions??: $Left$, $Right$, $Suck$, $NoOp$
goal test??: no dirt
path cost??: 1 per action (0 for $NoOp$)

Problem Types

Deterministic, fully observable $\Longrightarrow$ single-state problem

Agent knows exactly which state it will be in; solution is a sequence

Non-observable $\Longrightarrow$ conformant problem

Agent may have no idea where it is; solution (if any) is a sequence

Nondeterministic and/or partially observable $\Longrightarrow$ contingency problem

percepts provide **new** information about current state

solution is a contingent plan or a policy

often **interleave** search, execution

Unknown state space $\Longrightarrow$ exploration problem ("online")

Example: Vacuum world

Single-state, start in #5. Solution??

$[Right, Suck]$

Conformant, start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$

e.g., $Right$ goes to $\{2, 4, 6, 8\}$. Solution??

$[Right, Suck, Left, Suck]$

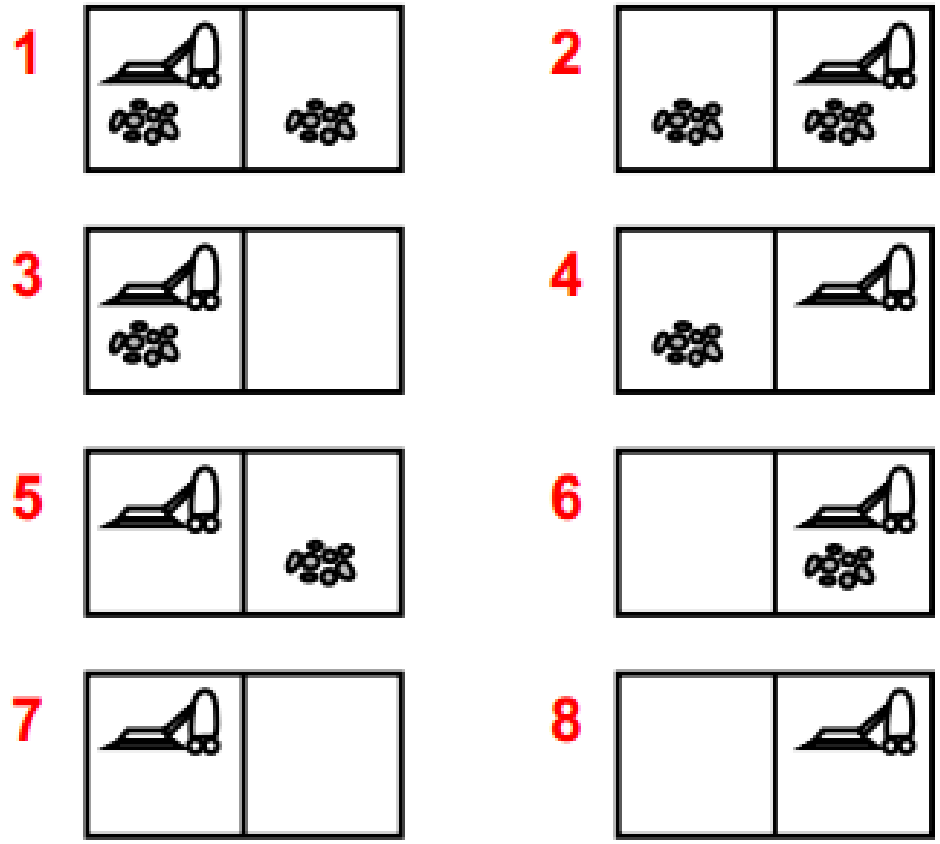
Contingency, start in #5

Murphy's Law: $Suck$ can dirty a clean carpet

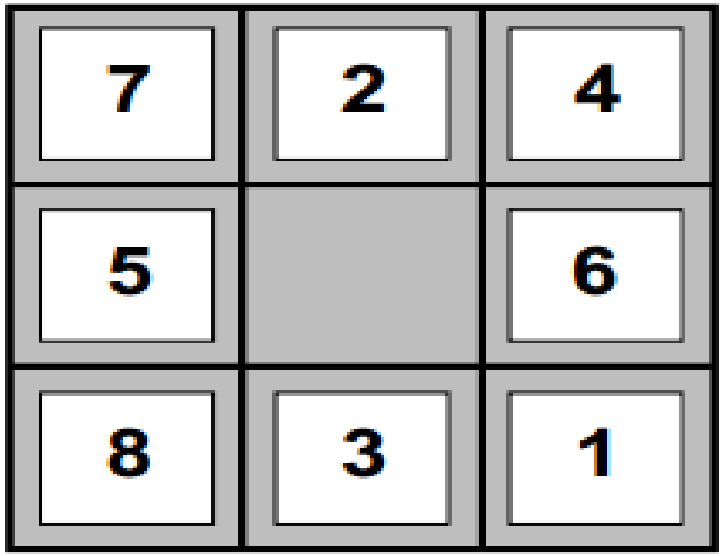
Local sensing: dirt, location only.

Solution??

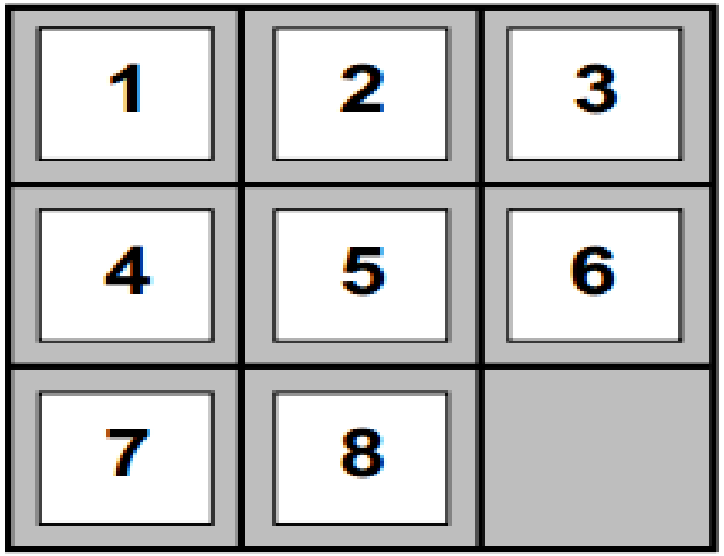
$[Right, \textbf{if } dirt \textbf{ then } Suck]$



Example: 8 puzzle



Start State



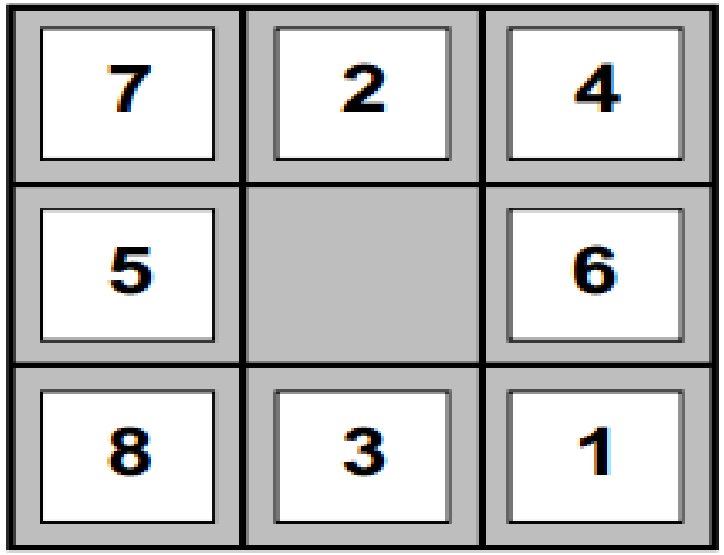
Goal State

For the 8 piece puzzle above define

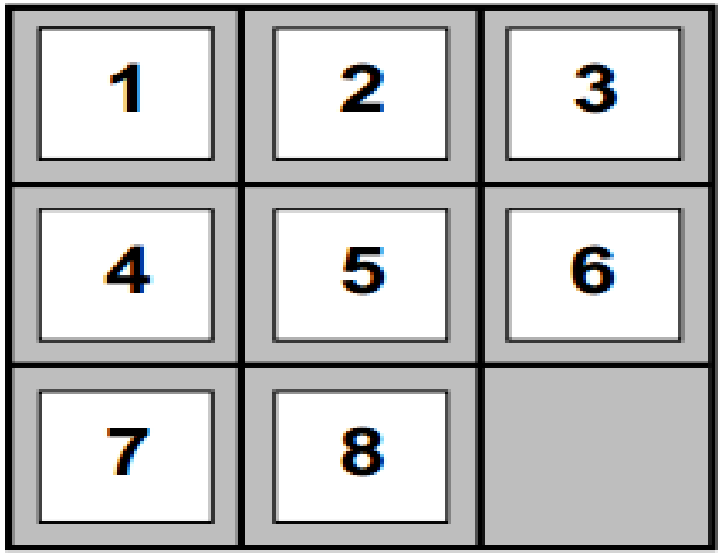
States, Actions, Goal Test and Path Cost

Engels. R

Example: 8 puzzle



Start State



Goal State

states??: integer locations of tiles (ignore intermediate positions)
actions??: move blank left, right, up, down (ignore unjamming etc.)
goal test??: $=$ goal state (given)
path cost??: 1 per move

[Note: optimal solution of n-Puzzle family is NP-hard]

8-Queens problem

- Goal
  - To place eight queens on a chessboard such that no queen attacks any other
- An incremental formulation
  - involves operators that augment the state description, starting with an empty state
  - Each action adds a queen to the state.
- A complete-state formulation
  - starts with all 8 queens on the board and moves them around

- For the 8 queens problem define **States, Actions, Transition model, Goal Test and Path Cost**

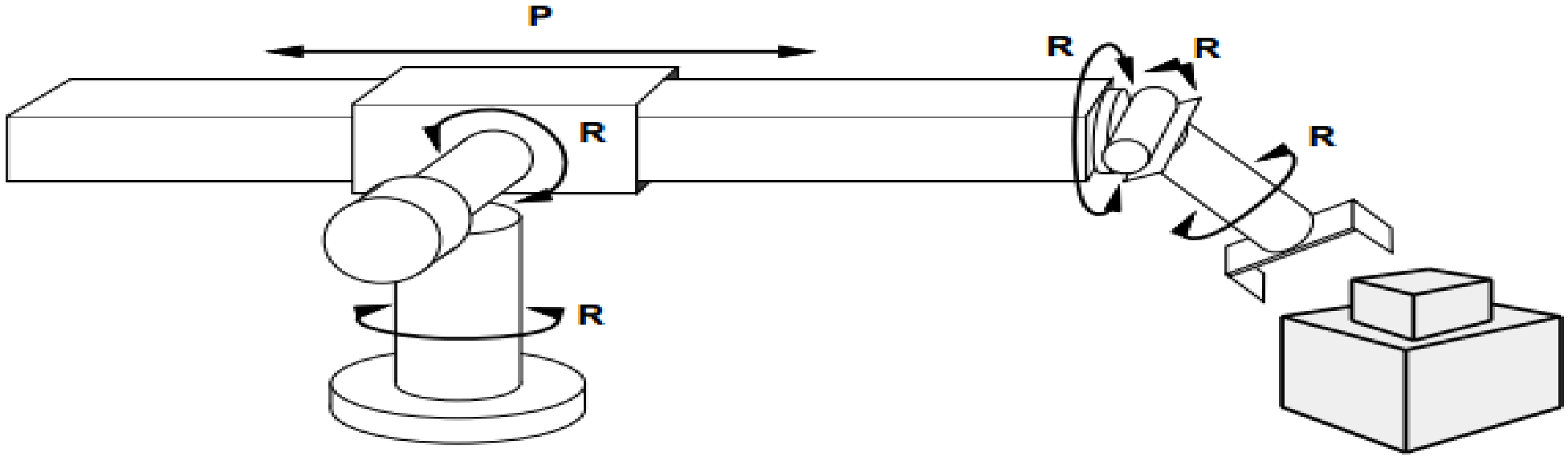
Engels. R

8-Queens problem: Incremental

- **States**: Any arrangement of 0 to 8 queens on the board
- **Initial state**: No queens on the board
- **Actions**: Add a queen to any empty square
- **Transition model**: Returns the board with a queen added to the specified square
- **Goal test**: 8 queens are on the board, none attacked
- A **better** formulation would prohibit placing a queen in any square that is already attacked:
  - **States**: All possible arrangements of n queens (0 <= n <= 8), one per column in the leftmost n columns, with no queen attacking another
  - **Actions:** Add a queen to any square in the left most empty column such that it is not attacked by any other queen

Engels. R

Real world problem : Robotic assembly



states??: real-valued coordinates of robot joint angles
parts of the object to be assembled

actions??: continuous motions of robot joints

goal test??: complete assembly **with no robot included!**

path cost??: time to execute

Assignments

- Home work problems are assignments – CO Based
- Due date: One week from announcement
- **CO1:**Apply the fundamental AI concepts to represent and solve real-world problems
- 1-1. Self study – **1.3.6 – 1.3.10** – 1 page summary
- 1-2. Self study **1.5** – Summary – ½ page summary
- 1-3. AIMA book Chapter 1 : **1.10 – 1.14**
- 1-4. AIMA book Chapter 2: **2.3, 2.4, 2.5, 2.6, 2.7, 2.9**
- 1-5. Next page

Engels. R

Assignments

- 1-5
- For the following problems suggest representation of environments with justification. The goal is to optimize both effectiveness and efficiency
  - Designing a Massively Multiplayer online Role Playing Game (MMORPG) where different entities are to be represented by AI models
  - Design a Natural Language Understanding module to recognize satire or parody elements in a Tamil story
  - Dataset with climate information with 20 different parameters and the task is to predict if there would be a hurricane in the next few days

Engels. R

SEARCH APPROACHES

Engels. R
Engels. R
Engels. R
Engels. R
Engels. R

Example problem

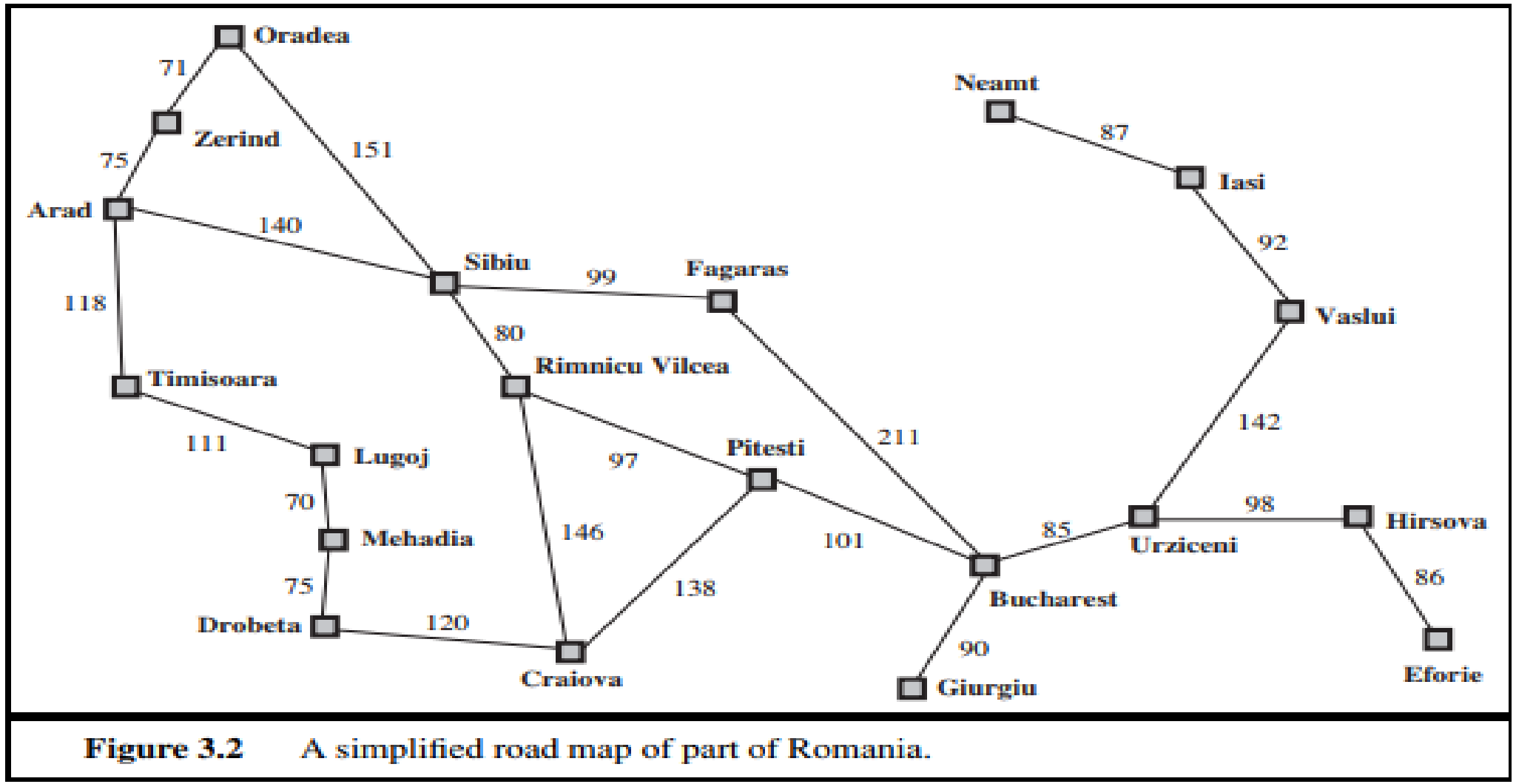


Figure 3.2 A simplified road map of part of Romania.

- Agent is Arad - Needs to fly out of Bucharest – Find the shortest path

Engels

Infrastructure for search algorithms

- Search algorithms require a data structure to keep track of the search tree that is being constructed
- For each node n of the tree, the structure contains 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.

Engels. R

Tree Search

- Solution is an action sequence
- Search algorithms work by considering various possible action sequences
- Possible action sequences starting at the initial state form a **search tree**
  - with the initial state at the root
- **Expanding** the current state (**Frontier** – leaf nodes available for expansion)
- **Generating** a new set of states
- **Exploring** the new states

Basic idea:

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

Engels. R

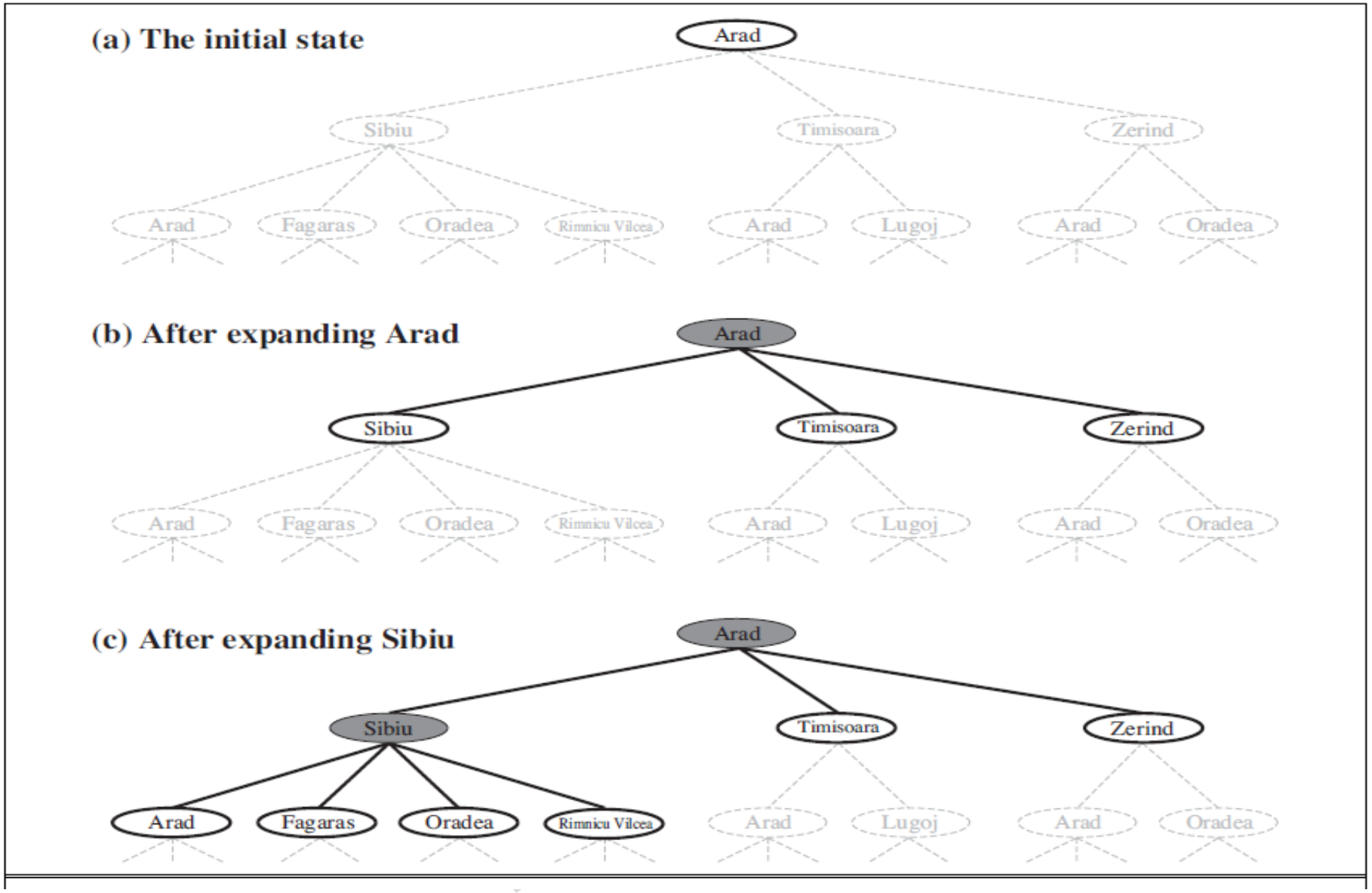
Tree Search

```
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
    end
```

- Search algorithms all share this basic structure
- Vary primarily on how they choose which state to expand next - called **search strategy**

Engels. R

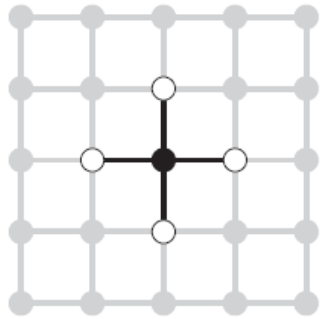
Tree search example



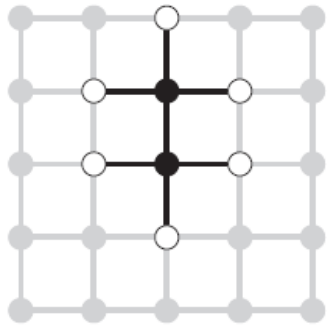
- Partial search trees for finding a route from Arad to Bucharest
- Expanded Nodes are shaded
- Generated but not yet expanded nodes are outlined in bold
- Not yet generated nodes in faint dashed lines

Graph Search

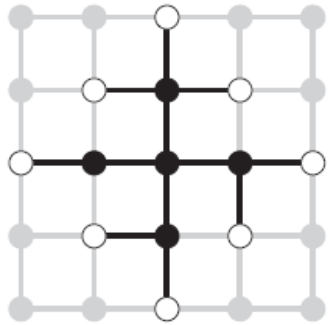
function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the 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



(a)



(b)



(c)

- ***Bold italic*** *in* GRAPH-SEARCH are additions needed to handle repeated states
- Redundant paths are unavoidable
  - *algorithms that forget their history are doomed to repeat it*
  - avoid exploring redundant paths is to remember where one has been

Engels, R

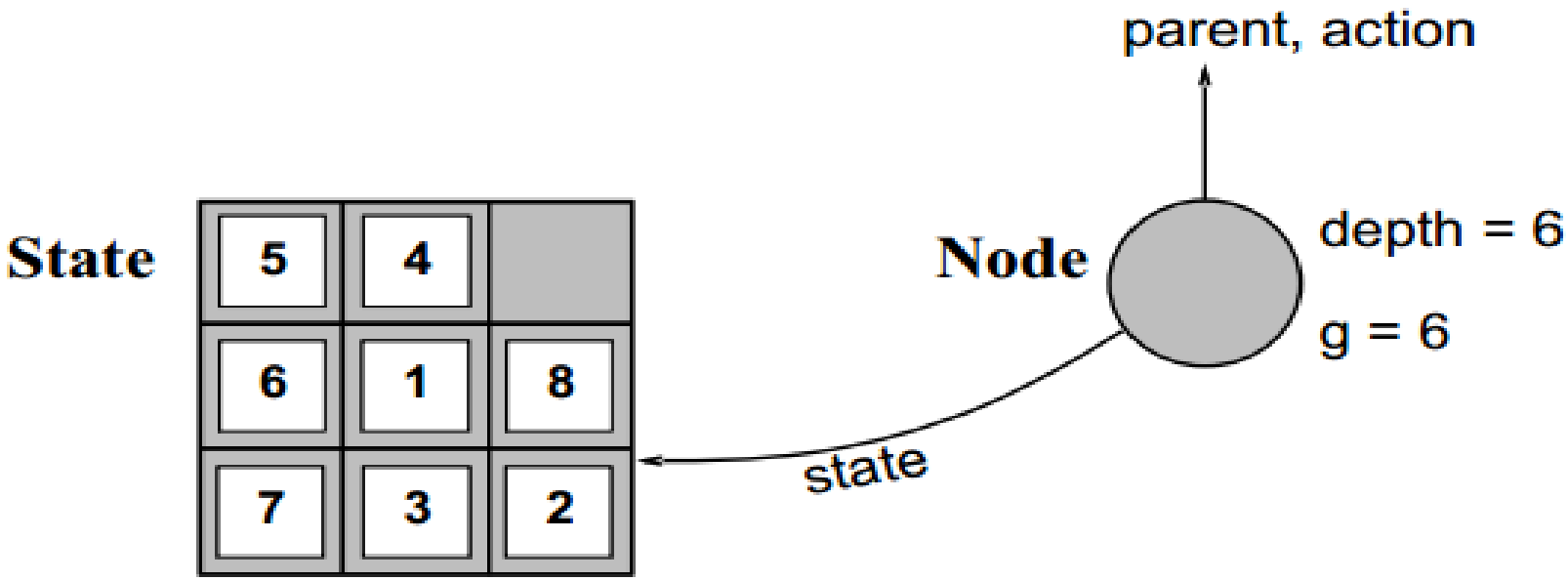
Implementation: States vs. Nodes

A state is a (representation of) a physical configuration

A node is a data structure constituting part of a search tree

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 SUCCESSORFN of the problem to create the corresponding states.

Tree Search: Implementation

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    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 node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Search Types

- **Uninformed (blind) search**
  - Strategies have no additional information about states beyond the problem definition
  - Can generate successors
  - Can distinguish a goal state from a non-goal state
  - **Strategies distinguished by the order** in which nodes are expanded
- Breadth-First search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search
- **Informed search / heuristic search**
  - Strategies know whether one non-goal state is "more promising" than another

Engels. R

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

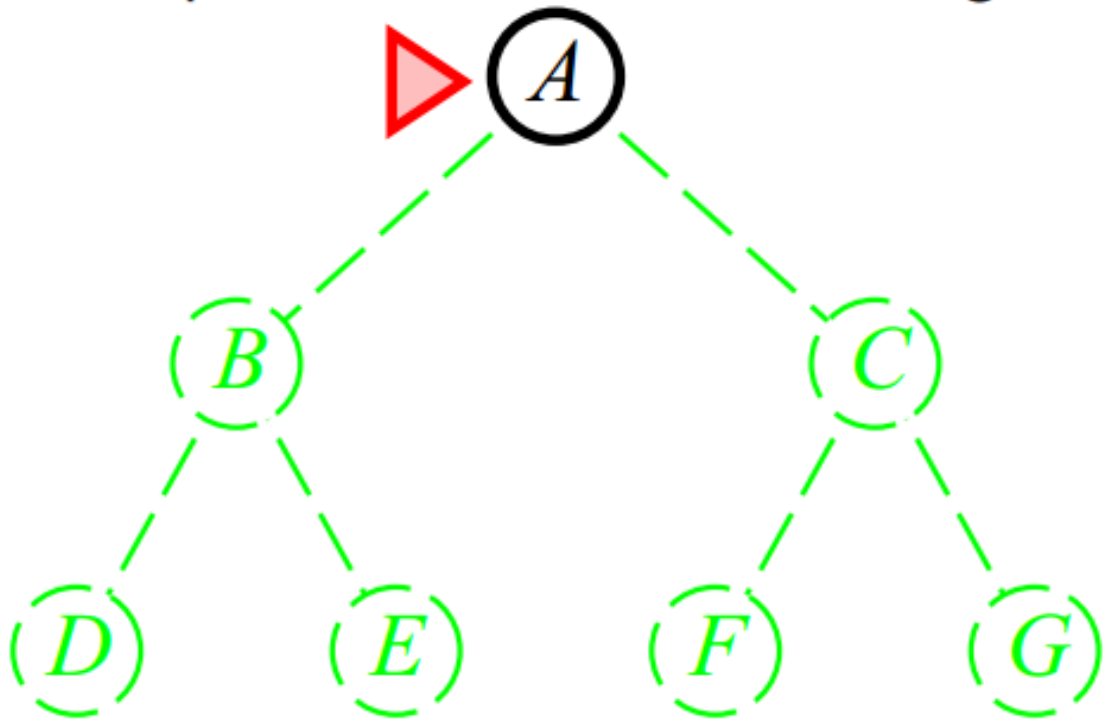
- b—maximum branching factor of the search tree
- d—depth of the least-cost solution
- m—maximum depth of the state space (may be ∞)

Breadth-First Search (BFS)

Expand shallowest unexpanded node

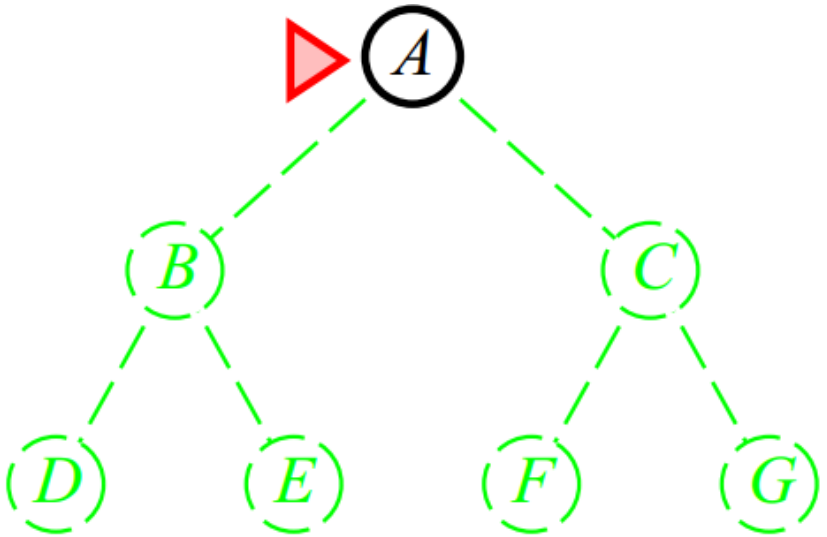
Implementation:

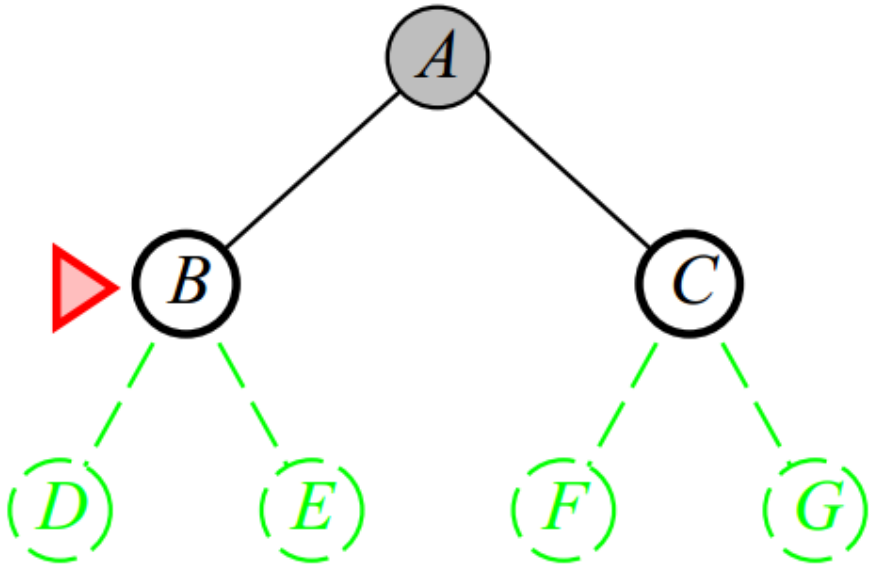
fringe is a FIFO queue, i.e., new successors go at end

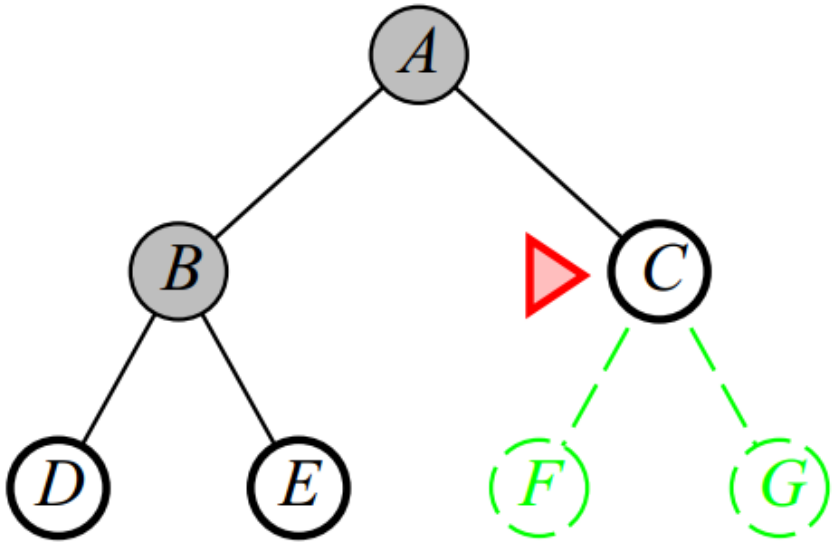


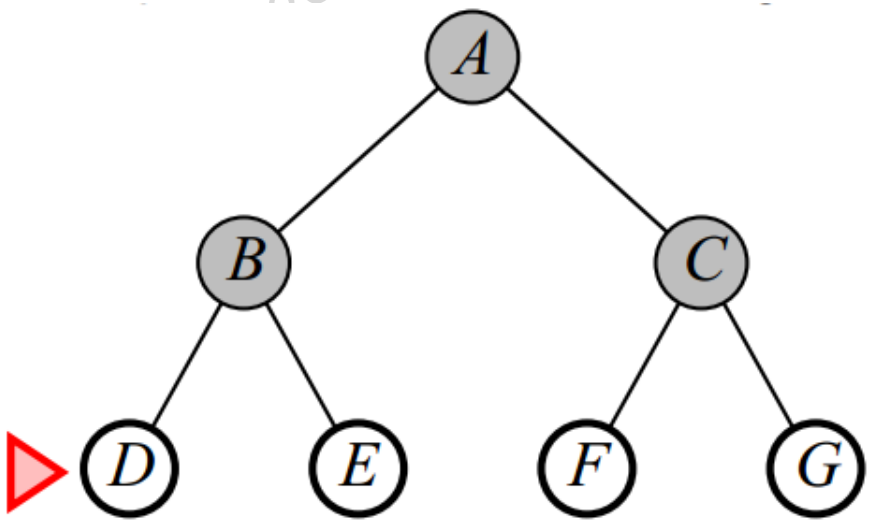
Engels. R
R
F
R

BFS – Contd.









Engels. R

Breadth-First Search Pseudo code

```
Inaction 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 q <—  POP(frontier)   /* chooses the shallowest node in  frontier */
    add node.STATE  to explored /* node should not be explored again*/
    for each  action in  problem.ACTIONS(node.STATE) do /* possible children */
            child <— CHILD-Node(problem, node, action)  /*get the child*/
            if child.STATE is not in explored or  frontier  then
                    if  problem.GOAL-TEST(child.STATE) then return  SOLUTION( child)
                    frontier.INSERT(child, frontier) /* not explored, add to frontier */
```

Engels. R

Breadth-First Search

Complete?? Yes (if b is finite)

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

Space?? $O(b^{d+1})$ (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 100MB/sec
so 24hrs $=$ 8640GB.

Assuming that 1 million nodes can be created a second and 1000 bytes storage per node

Engels. R

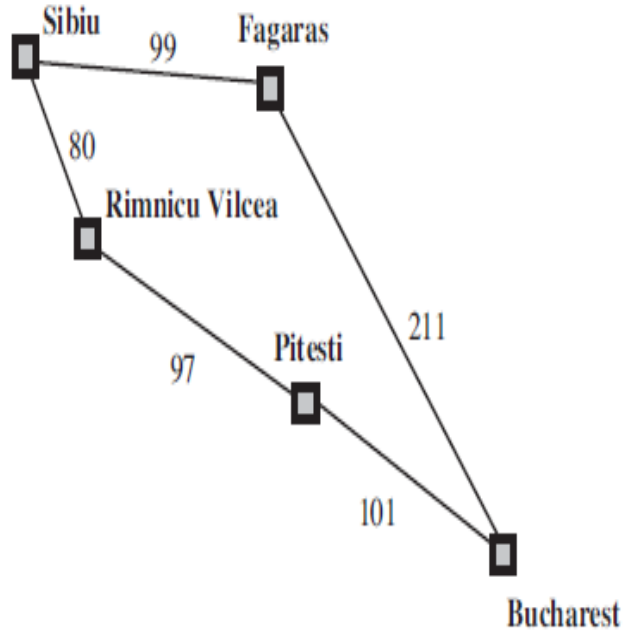
Uniform-cost search

- What if the costs are different?
- Breadth-First becomes Uniform-cost search, if the lowest step cost is chosen for selecting a node for expansion
- Differs from BFS
  - Goal test is applied when node is picked for expansion rather than creation
  - Better paths are looked for even after reaching goal state (at the lowest level – depth)
- Not based on b/d. Based on Cost!
- Assume each step costs at least ε

Engels. R

Uniform-cost search

- ***Uniform-cost search*** *expands nodes in order of their optimal path cost*
- Does not care about the ***number*** **of steps** a path has
- Only about the **total cost**
  - Can get stuck in an infinite loop if there is a path with an infinite sequence of zero-cost actions
- Book example: Successors of Sibiu are Rimnicu Vilcea and Fagaras, with costs 80 and 99, respectively
- Least-cost node, Rimnicu Vilcea, is expanded next, adding Pitesti with cost 80 + 97=177
- The least-cost node is now Fagaras, so it is expanded, adding Bucharest with cost 99+211=310
- Now a goal node has been generated, ***but uniform-cost search keeps going***,
- Choosing Pitesti for expansion and adding a second path to Bucharest with cost 80+97+101= 278
- **Final chosen path:**



Engels. R

Uniform-cost Search

Expand least-cost unexpanded node

Implementation:

$fringe$ = queue ordered by path cost, lowest first

Equivalent to breadth-first if step costs all equal

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

Time?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$
where C^* is the cost of the optimal solution

Space?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$

Optimal?? Yes—nodes expanded in increasing order of $g(n)$

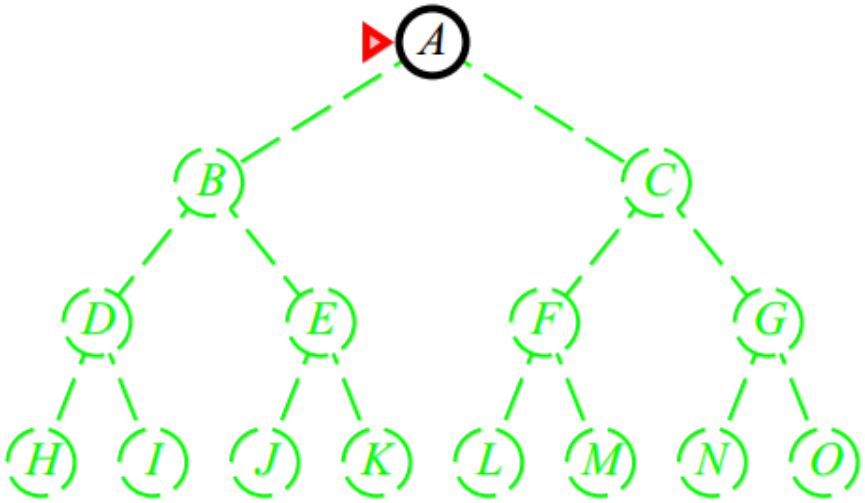
Engels. R

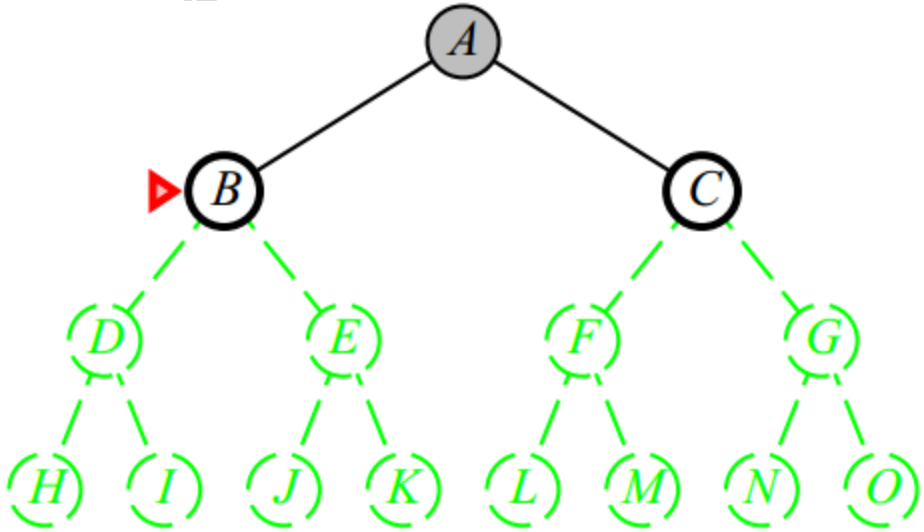
Depth-First Search (DFS)

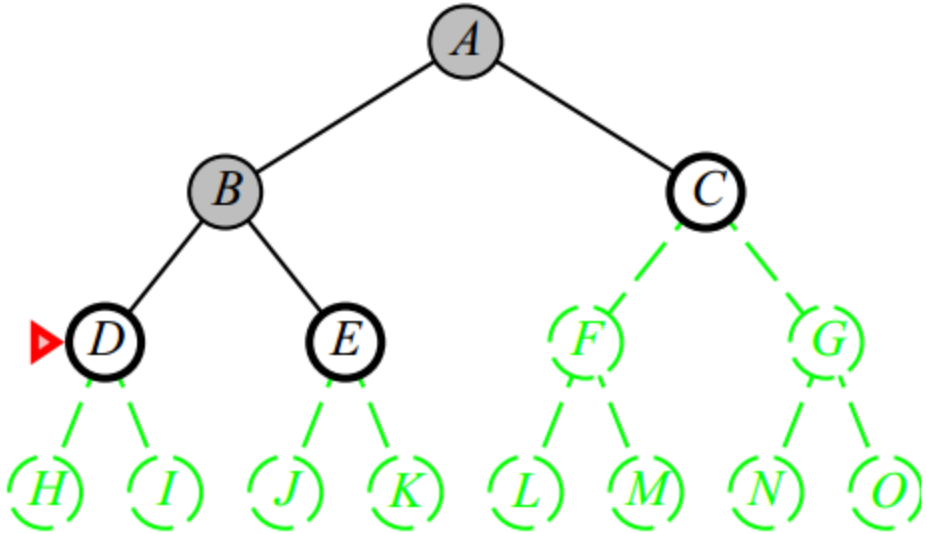
- Expands the **deepest** node in the current frontier of the search tree
- Proceeds immediately to the deepest level of the search tree, where the nodes have no successors.
- As nodes are expanded, they are dropped from the frontier, so then the search **backs up** to the **next deepest node** that still has **unexplored successors**
- fringe =LIFO queue ,i.e., put successors at front

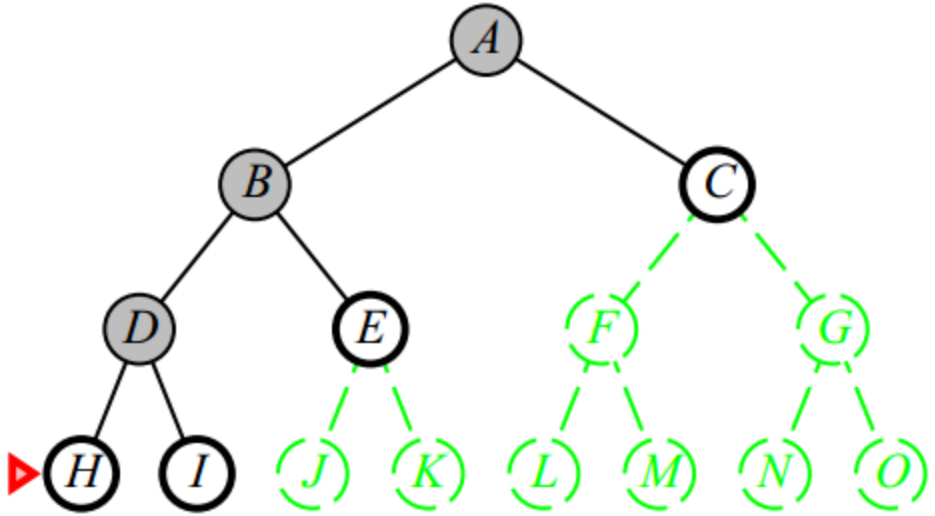
Engels. R

DFS-Contd.



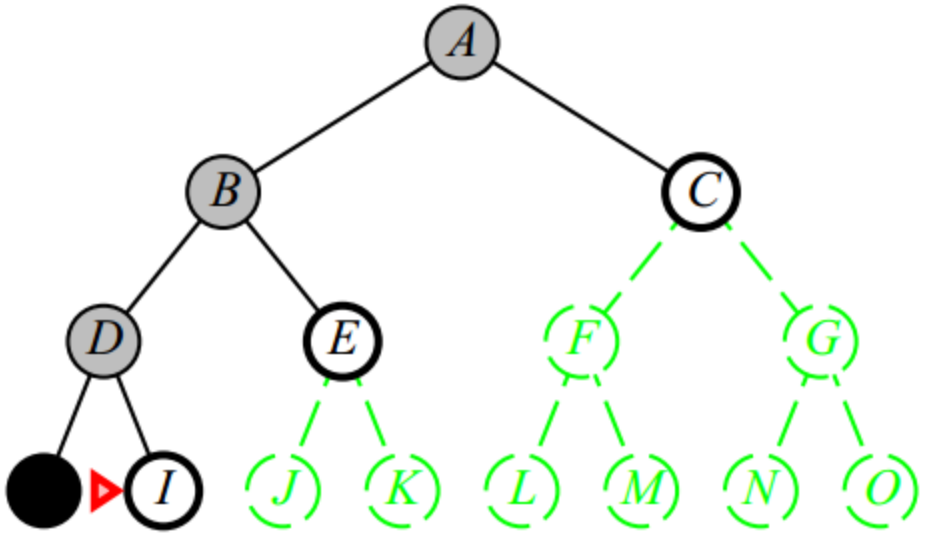


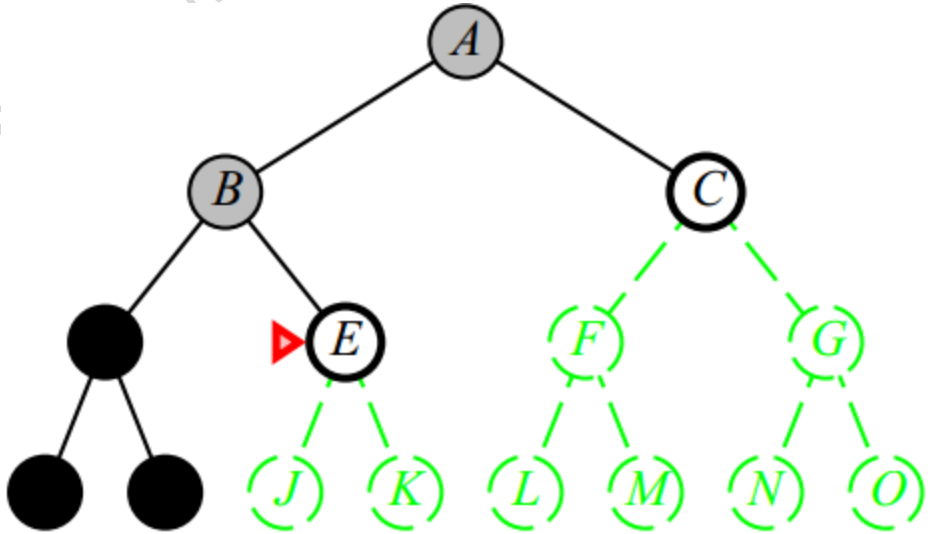


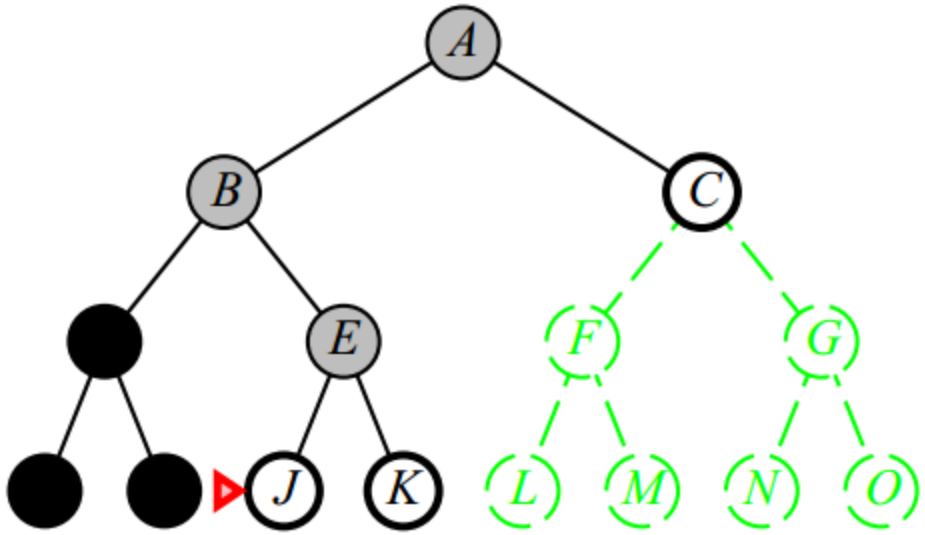


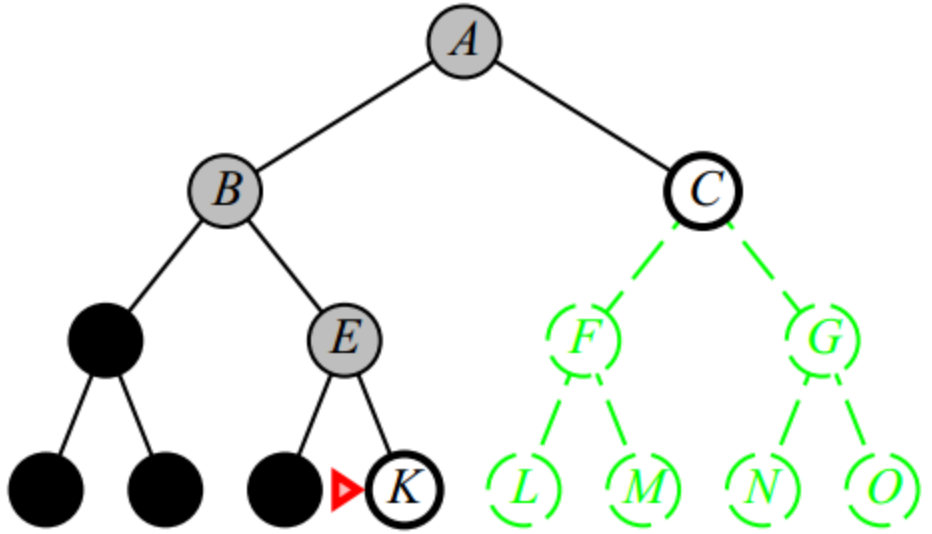
Engels. R

DFS-Contd.



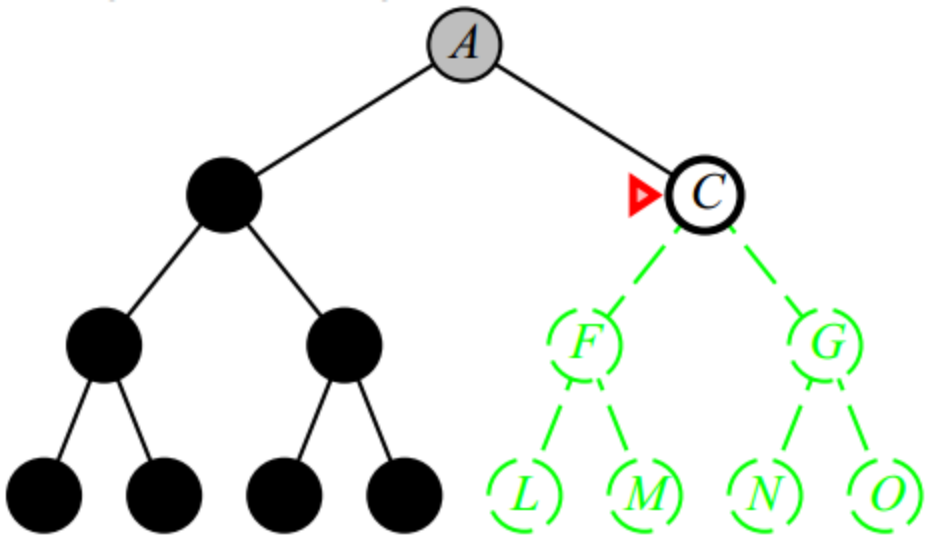


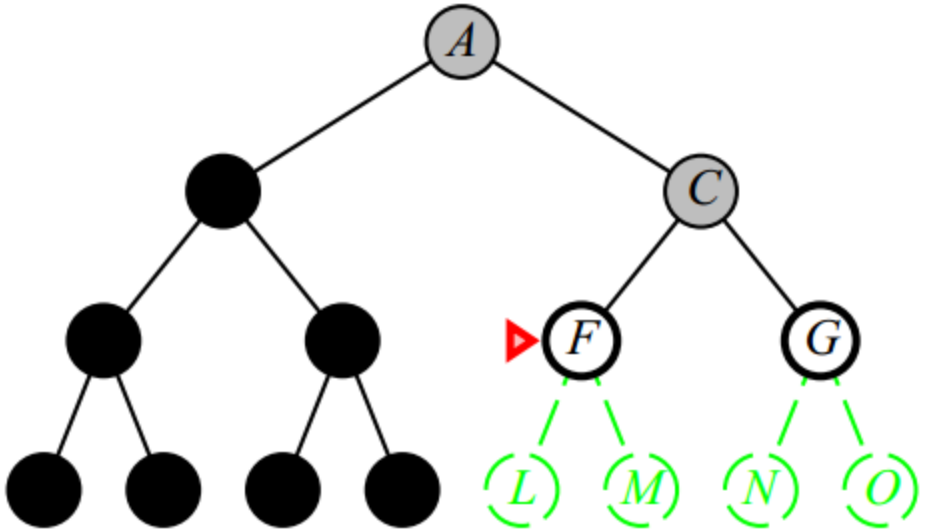


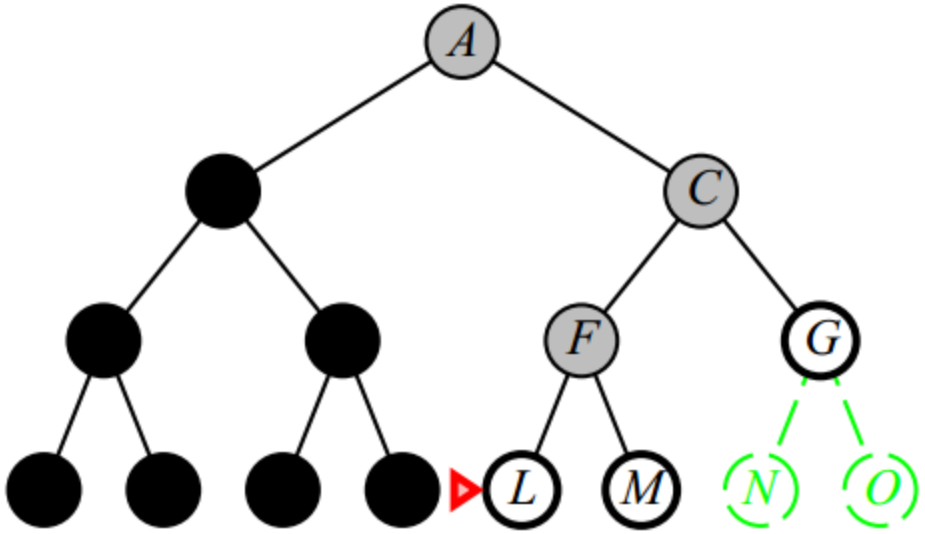


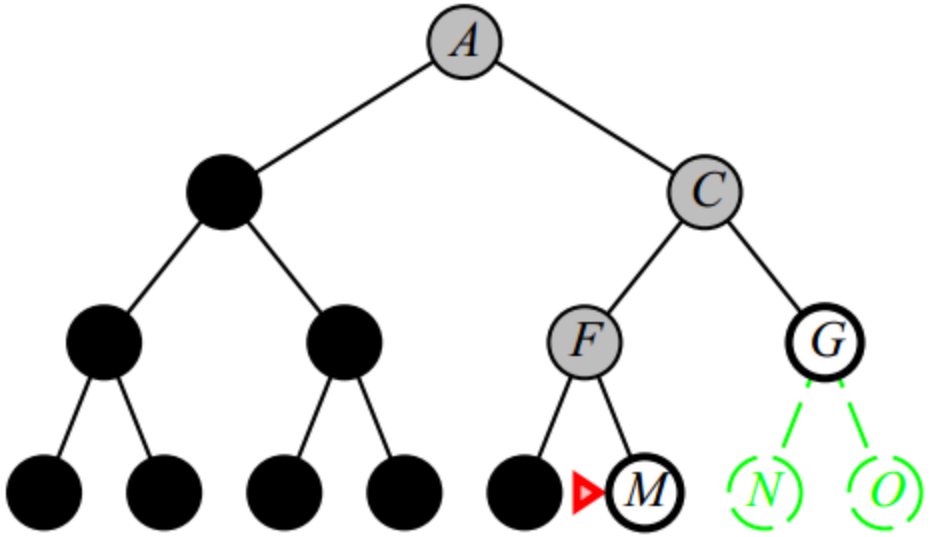
Engels. R

DFS-Contd.









Engels. R

DFS - Properties

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

- A variant of depth-first search is called **backtracking search** uses still less memory.
- In backtracking, only one successor is generated at a time rather than all successors;
- Each partially expanded node remembers which successor to generate next
  - only O(m) memory is needed rather than O(bm)

Depth-limited search

- DFS failure in infinite spaces can be addressed by limiting depth
- Depth limit l (hopefully l ≤ d, where d is the minimum depth at which the shallowest goal is available)
- Depth-limited search will also be non-optimal if l >> d
- Time complexity $O(b^l)$, Space Complexity O(bl)
- DFS is special case of DLS, with l = infinity
- 20 cities are in Romania map example
  - So |hops| to reach any node is ≤ n (so l could be set as 19)
- However in this specific example, any city can be reached from any other city in at most 9 steps
- So l = 9 is the best choice (9 is the **diameter** of state space)

Engels. R

Iterative deepening depth-first search

- Combination of depth-first tree search, with finding the best depth limit
  - by gradually increasing the limit
  - first 0, then 1, then 2, and so on
  - until a goal is found
- Goal is at depth limit **d**, the depth of the shallowest goal node
- Iterative deepening combines the benefits of depth-first and breadth-first search
  - Modest memory requirements (O(bd)) as DFS
  - Like BFS, Complete when the branching factor is finite and
  - optimal when path cost is non-decreasing function of depth of node

Engels. R

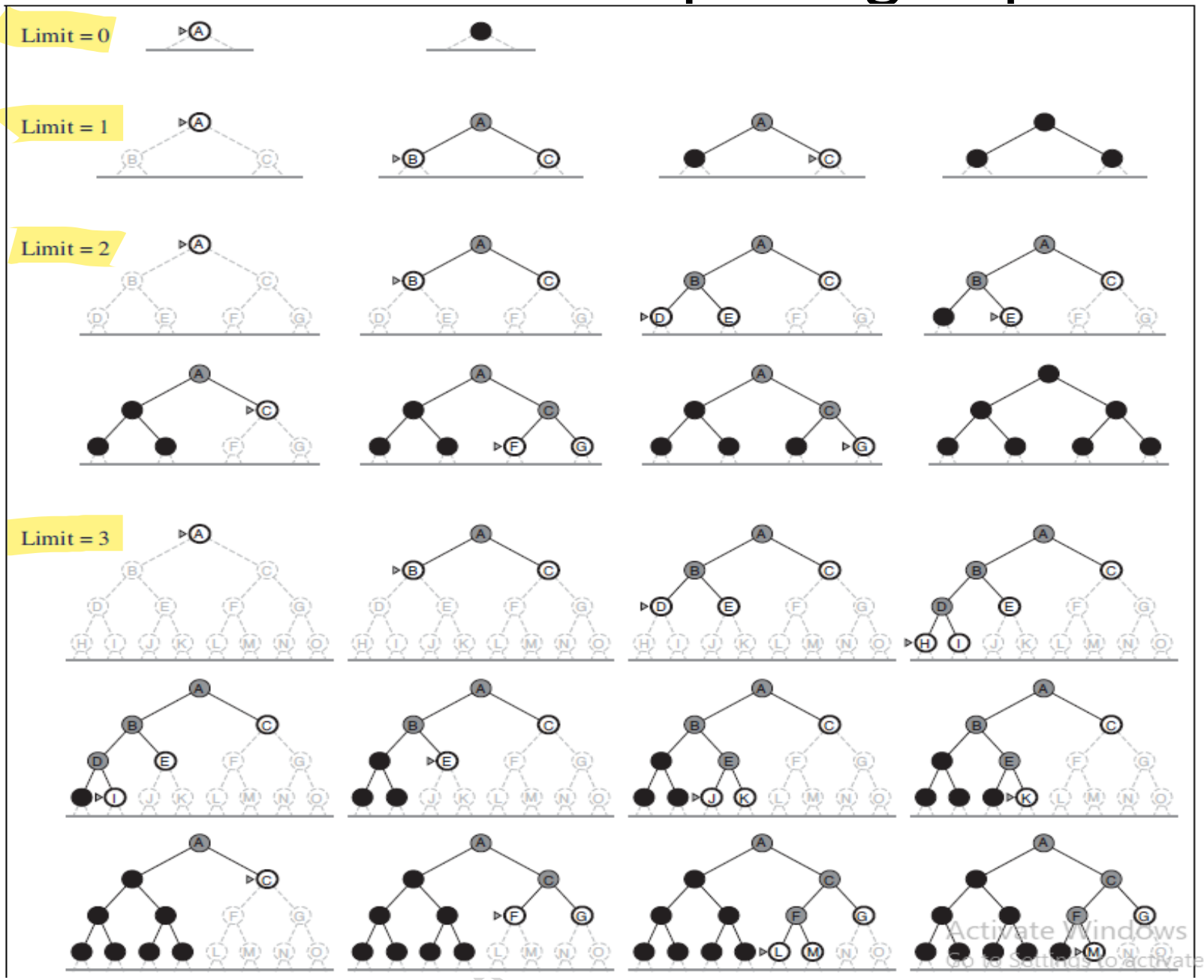
Iterative deepening depth-first search

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    for depth = 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
```

- Iterative deepening search algorithm
  - repeatedly applies depth limited search with increasing limits
  - terminates
    - when a solution is found or
    - if the depth limited search returns failure, meaning that no solution exists

Engels. R

Iterative deepening depth-first search



- Is it wasteful? because states are generated multiple times
  - No, this is not too costly.
- In a search tree with the same / nearly the same branching factor at each level, most of the nodes are in the bottom level
  - Upper levels are generated multiple times
  - Nodes on bottom level (depth d) are generated once

Total number of nodes generated in the worst case is

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

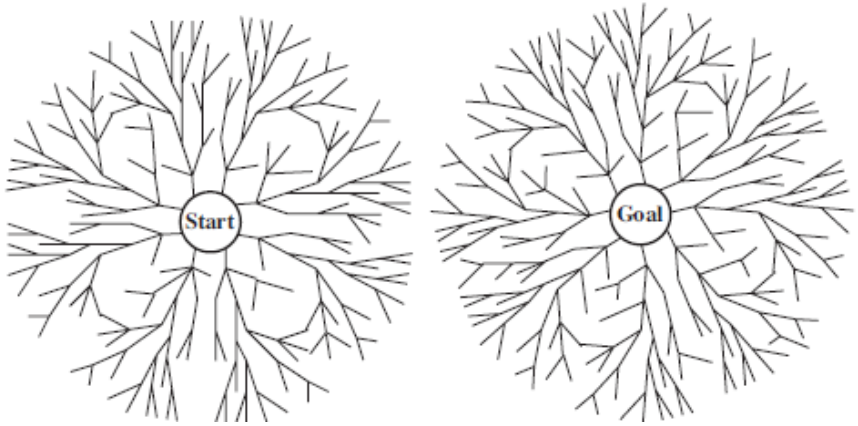
Time complexity = $O(b^d)$ (same as BFS)

Iterative deepening is the preferred uninformed search method when the search space is large and the depth of the solution is not known

Engels R
Activate Windows
Go to Settings to activate

Bidirectional Search

- Bidirectional search
  - runs two simultaneous searches
    - one forward from the initial state and
    - the other backward from the goal
  - hoping that the two searches meet in the middle
- Implemented by replacing goal test with a check
  - Do frontiers of the two searches intersect?
    - if they do, a solution has been found
- How do we search backwards?
  - Not very easy
  - requires a method for computing predecessor states
    - Possible For 8 piece and travel to reach destination
  - For several goal states (such as vacuum world)
    - A single state can be added as a successor for all goal states
  - For No queen attacks other queen
    - Searching from goal is impossible



A bidirectional search which is about to succeed

Engels. R

Comparing uninformed search strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

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?

Engels. R

Informed Search Strategy

- Uses problem-specific knowledge beyond the definition of the problem itself
- Can find solutions more efficiently than an uninformed strategy
- General approach of Informed Search Strategy is called **best-first search**
- Informed Search = Heuristic Search
- Most best-first algorithms include a **heuristic function**

- A **heuristic function** (aka **heuristic)** is a **function**
  - ***Ranks alternatives*** in search algorithms ***at each branching step***
    - Based on available information to decide which branch to follow
    - For example, it may approximate the exact solution
  - A function to calculate an ***approximate*** cost to a problem to rank alternatives
- Admissibility of a heuristic function = the characteristic that the heuristic function ***never overestimates the true cost***

Best First Search

- Recall that Uniform Cost Search expanded node n with the *lowest path cost* g(n)
  - UCS did this by storing the frontier as a priority queue ordered by g
- **Best First Search**
  - A node is selected for expansion based on an **evaluation function, f(n)**
- The evaluation function is construed as a cost estimate
  - the node with the ***lowest evaluation*** is expanded first

- Best-first includes as **a component of f, a heuristic function h(n):**
  - h(n) = estimated cost of cheapest path from the state at node n to a goal state
- h(n) takes a node as input
  - unlike g(n), it ***depends only*** on the state at that node
- For now, consider h as arbitrary, nonnegative, problem-specific function
  - Only one constraint: **if n is a goal node, then h(n)=0**

Engels. R

Best-First Search – Contd.

Idea: use an evaluation function for each node
– estimate of "desirability"

$\Rightarrow$ Expand most desirable unexpanded node

Implementation:
fringe is a queue sorted in decreasing order of desirability

Special cases:
greedy search
A* search

Greedy Search / Greedy Best First Search

- Valuation function h(n)(heuristic) = estimate of cost from **n** to the closest goal
  - So in GBFS, f(n) = g(n)
- Greedy search expands the node that appears to be closest to goal
  - Uses **Straight Line Distance** Heuristic, $\mathbf{h_{SLD}}$
- Constraints
  - 1. Values of h_{SLD} cannot be computed from the problem description itself
  - 2. Domain expertise is needed to correlate h(n) values with actual road distances

Engels. R

Greedy BF search – Example

Actual distances between cities are given in roadmap

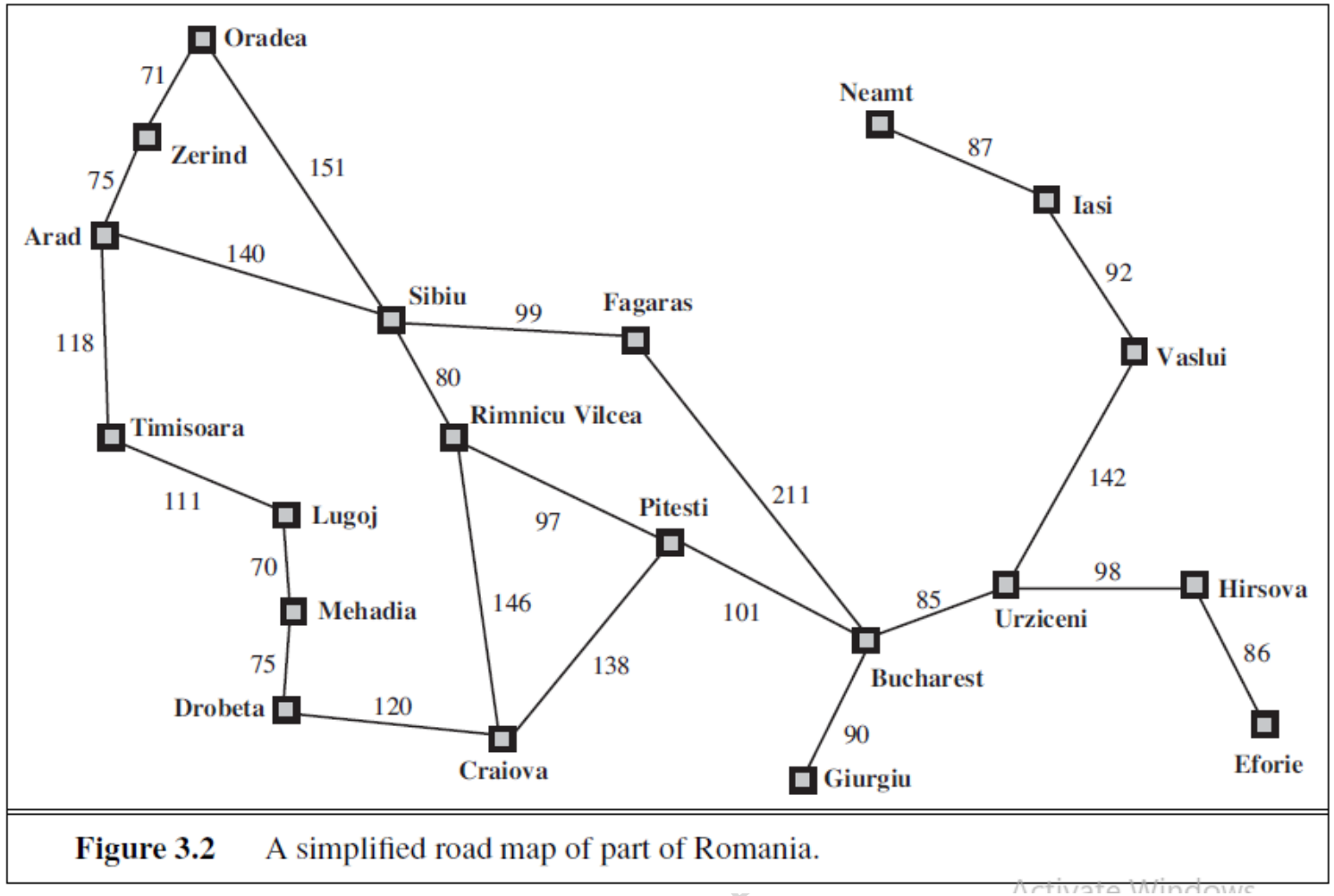
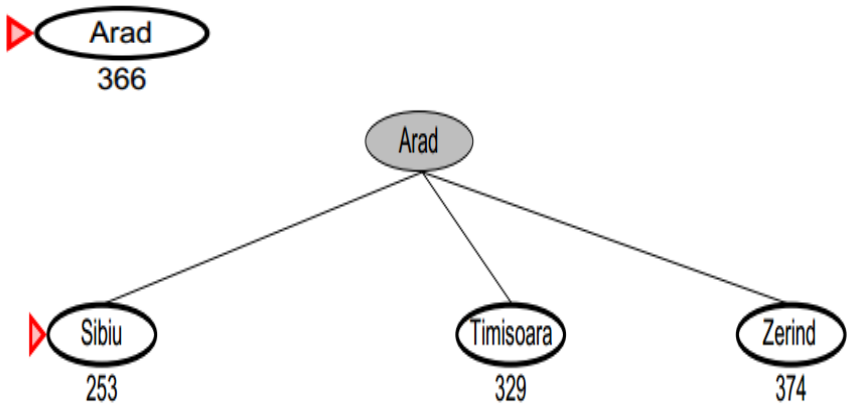


Figure 3.2 A simplified road map of part of Romania.

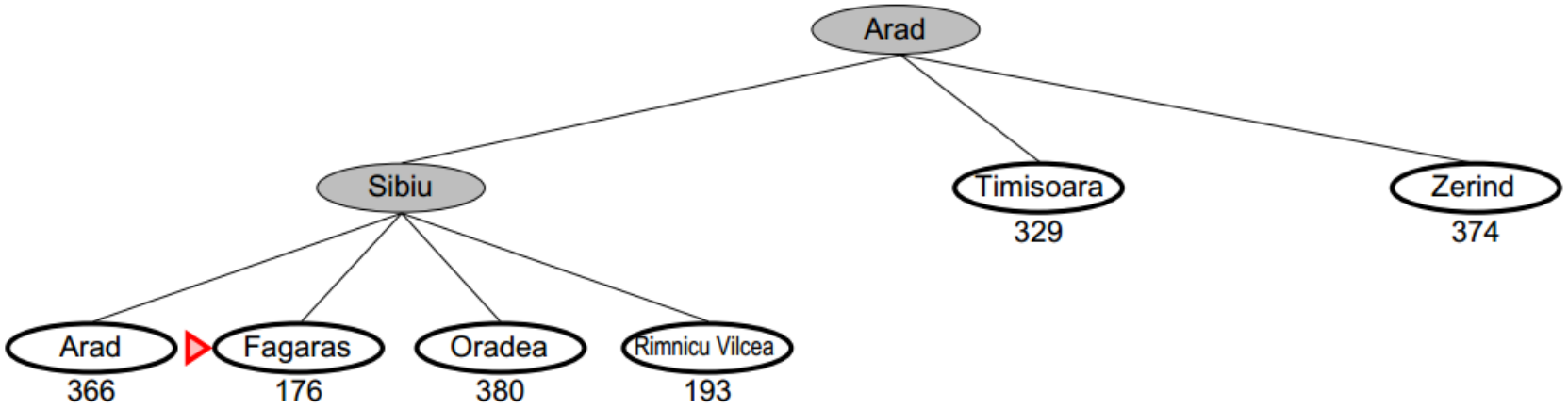
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

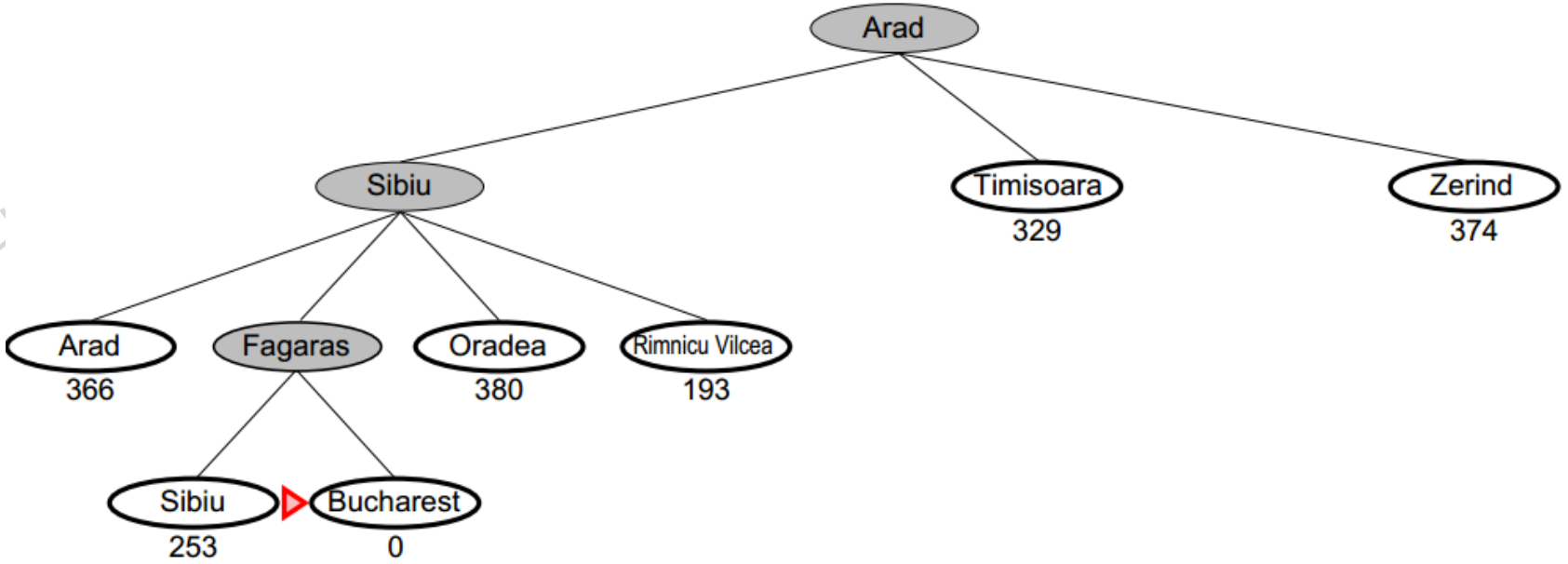
Values of h_{SLD}—straight-line distances to Bucharest.

Straight line distances between cities are in table



Greedy search – contd.





Engels. R

Properties of Greedy Search

Complete?? No–can get stuck in loops, e.g.,

Iasi → Neamt → Iasi → Neamt →

Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

Optimal?? No

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?

A* algorithm (A-star algorithm)

- Most widely known form of best-first search
- Evaluates nodes by combining
  - g(n), path cost from the start node to node n, and
  - h(n), the estimated cost of cheapest path from n to the goal

f(n) = g(n) + h(n)

f(n) = estimated cost of the cheapest solution through n

- Idea = Avoid expanding nodes which are already expensive
- Identical to Uniform Cost Search
  - except that A∗ uses **g + h** instead of just g
- A* strategy is more than just reasonable
- Provided that the heuristic function h(n) satisfies certain conditions, **A∗ search is both complete and optimal**

Engels. R

Admissible Heuristic

- First condition required for optimality is that h(n) be an **admissible heuristic**
  - ***never overestimates*** the cost to reach the goal
- Remember h(n) = 0 when n is the goal state
  - i.e. h(n) ≥ 0, such that at any goal state node G, h(G) = 0, otherwise h(n) > 0
  - and f(n)=g(n) + h(n), [g(n) = actual cost to reach n along the current path]
  - Then at G, f(n) = g(n) + 0 = g(n) [actual distance]
- The above have an immediate consequence that ***f(n) never overestimates the true cost of a solution along the current path through n***
  - h(n) ≤ h*(n) where h*(n) is the true cost of n to goal
- Example: $h_{SLD}(n)$ ***never overestimates*** the actual road distance
  - So A* using h_{SLD} is optimal

Engels. R

Admissible Heuristic

- The tree-search version of A* is optimal if h(n) is admissible
- The graph-search version of A* is optimal if h(n) is consistent
- **Consistency (Monotonicity)**
  - A heuristic h(n) is **consistent**
  - if, for every node n and every successor n' of n generated by any action a,
  - **the estimated cost** of reaching the goal from n **is no greater than** (a) the step cost of getting to n' + (b) the estimated cost of reaching goal from n'
- Consistency/Monotonicity is a form of the general **triangle inequality**,
  - Each side of a triangle cannot be longer than sum of other two sides
  - Here, the triangle is formed by n, n' and the goal G_n closest to n
- For an admissible heuristic, the inequality makes perfect sense:
  - if there were a route from n to Gn via n' that was cheaper than h(n), ***that would violate*** the property that h(n) is a lower bound on the cost to reach G_n.

Engels. R

A* - Roadmap example

Actual distances between cities are given in roadmap

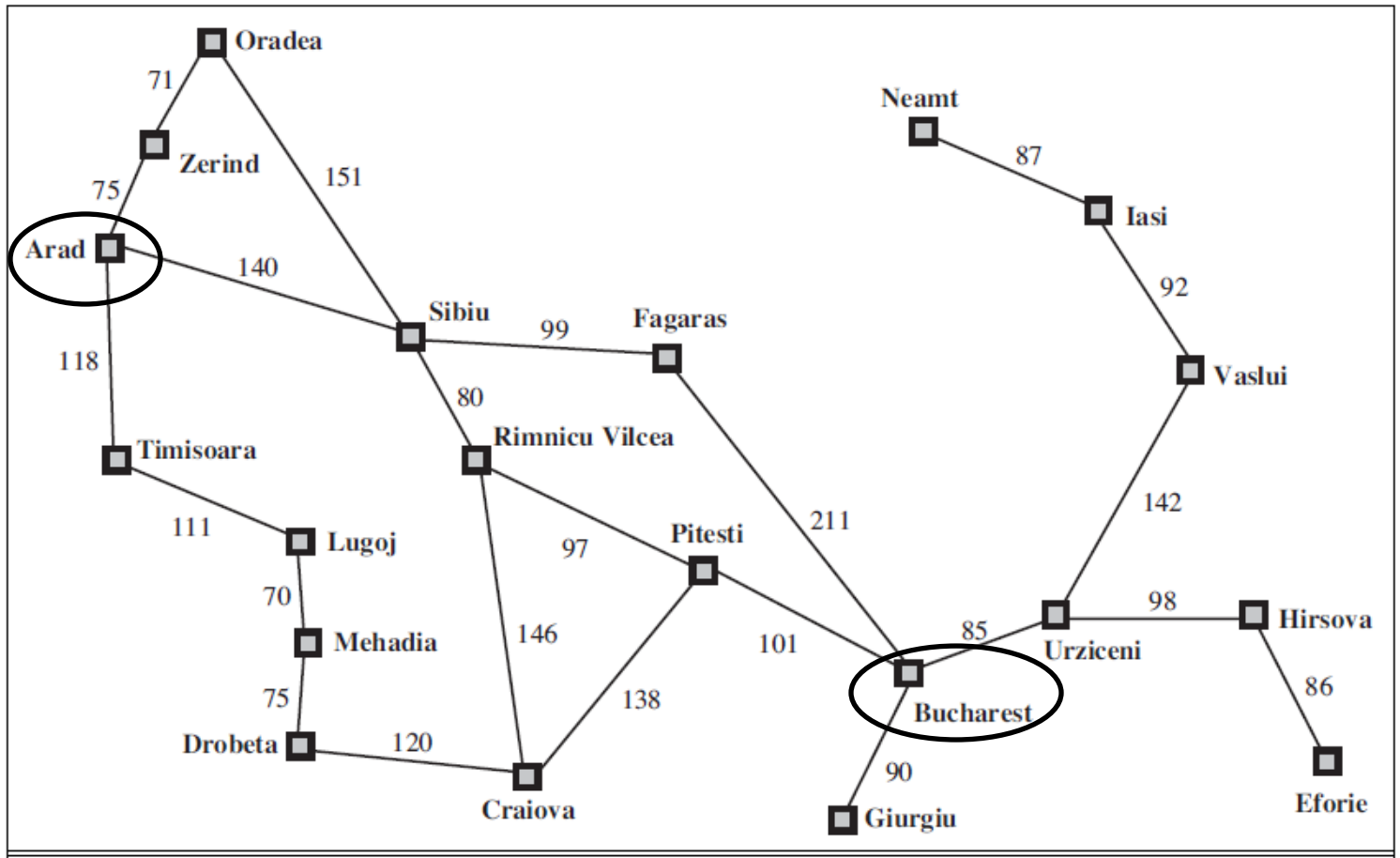


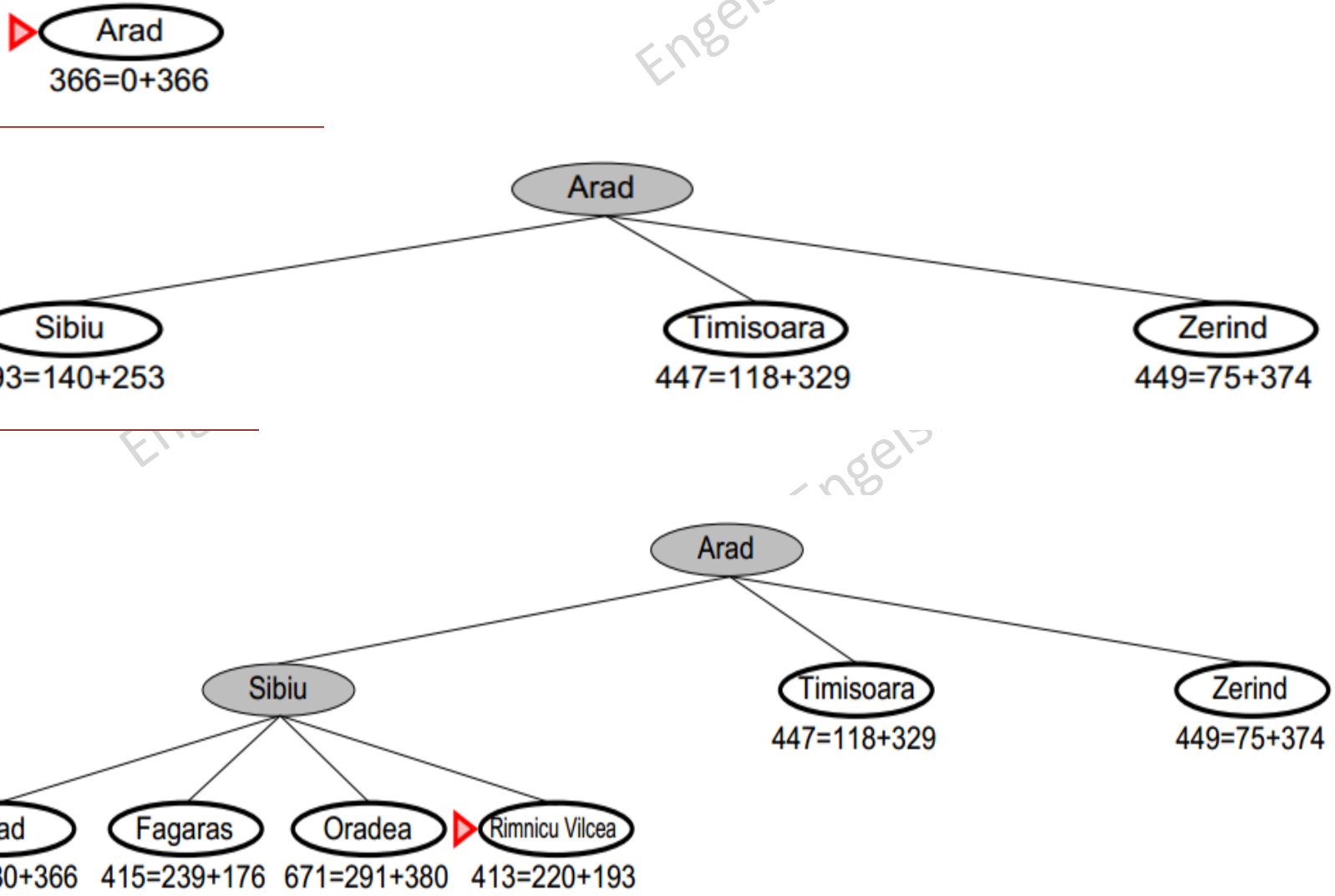
Figure 3.2 A simplified road map of part of Romania.

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Values of h_{SLD}—straight-line distances to Bucharest.

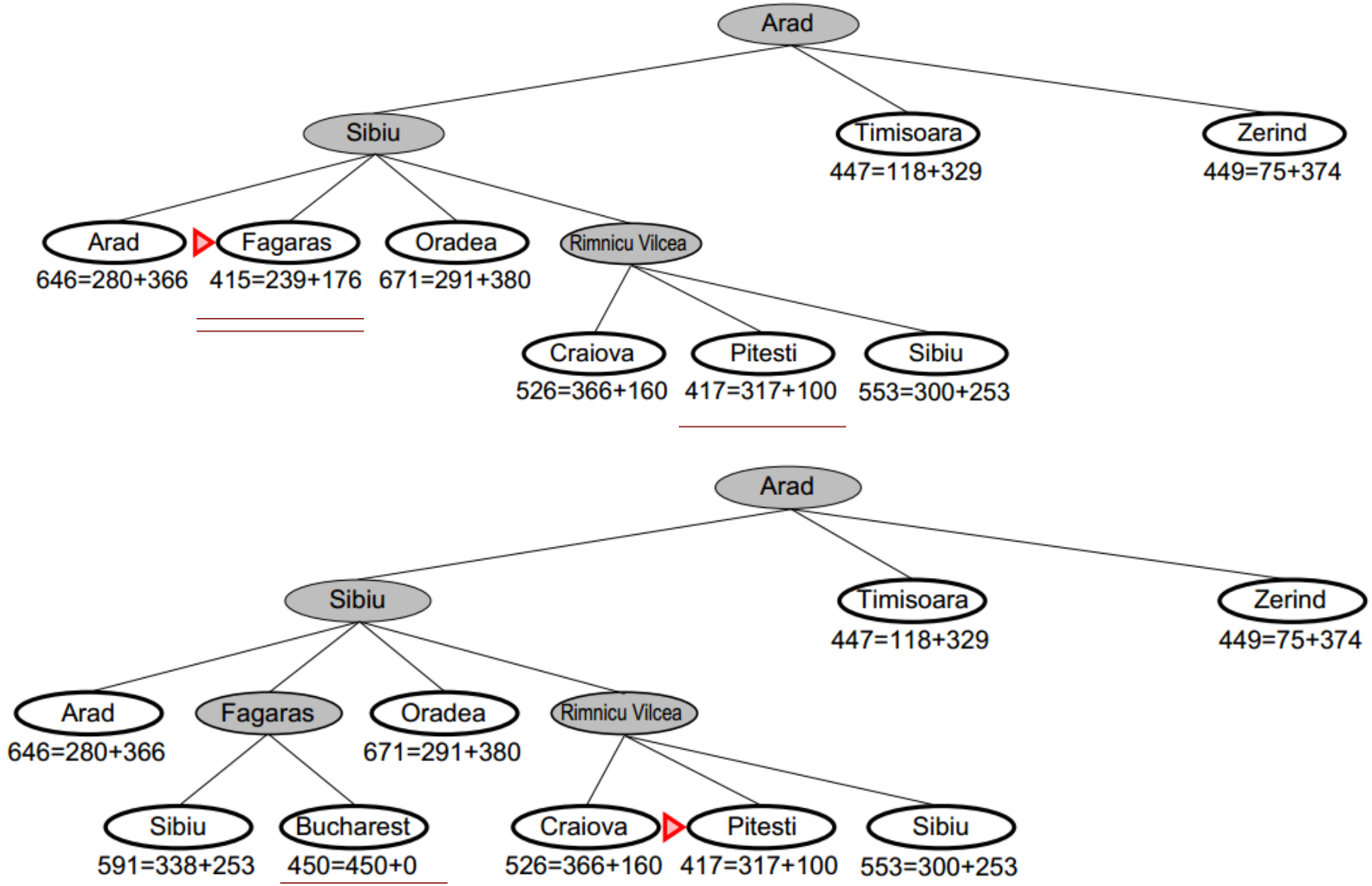
Straight line distances between cities are given in table

A* Example



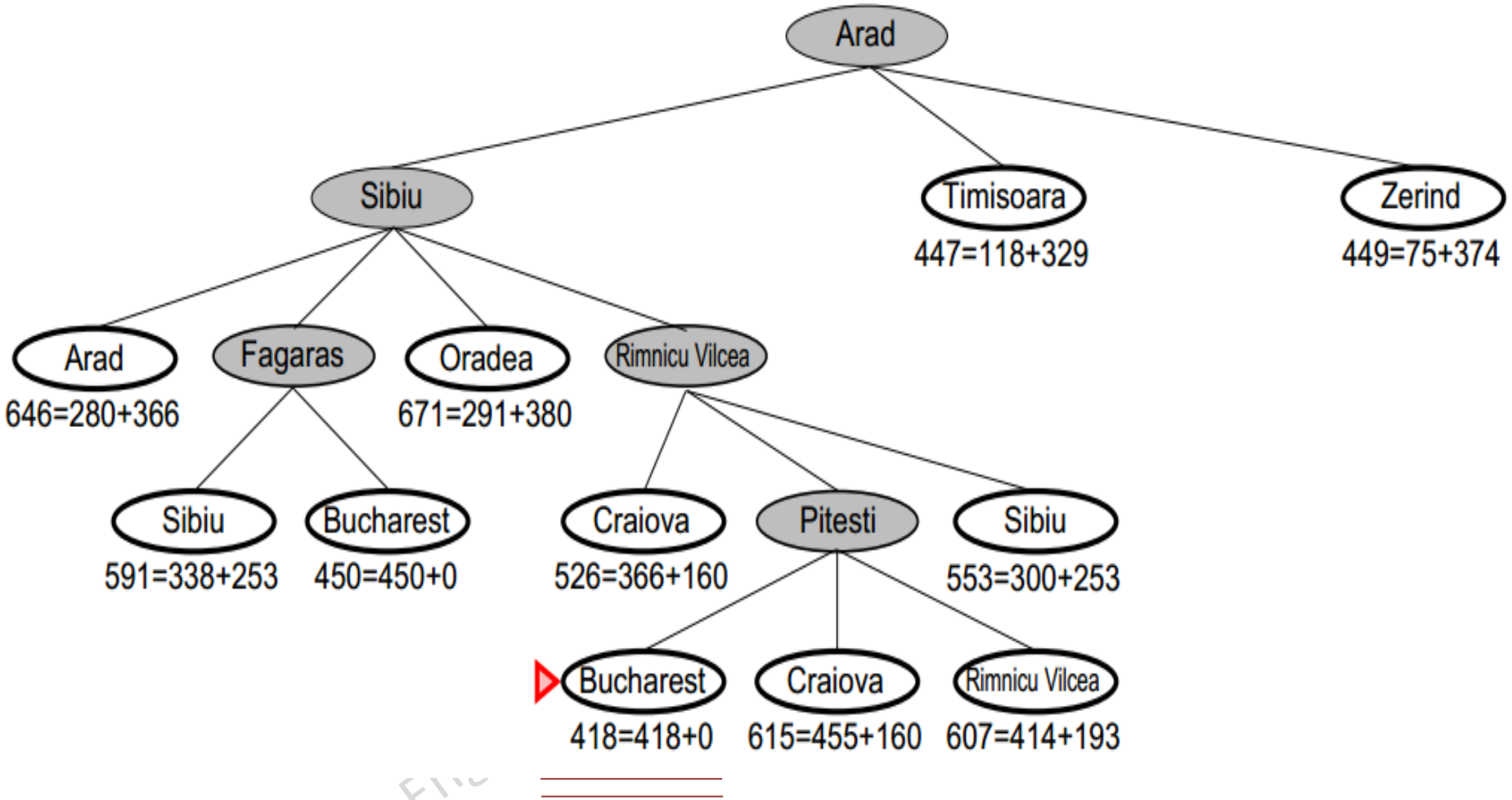
Engels. R

A* Example – Contd.



Engels. R

A* Example – Contd.



Engels. R

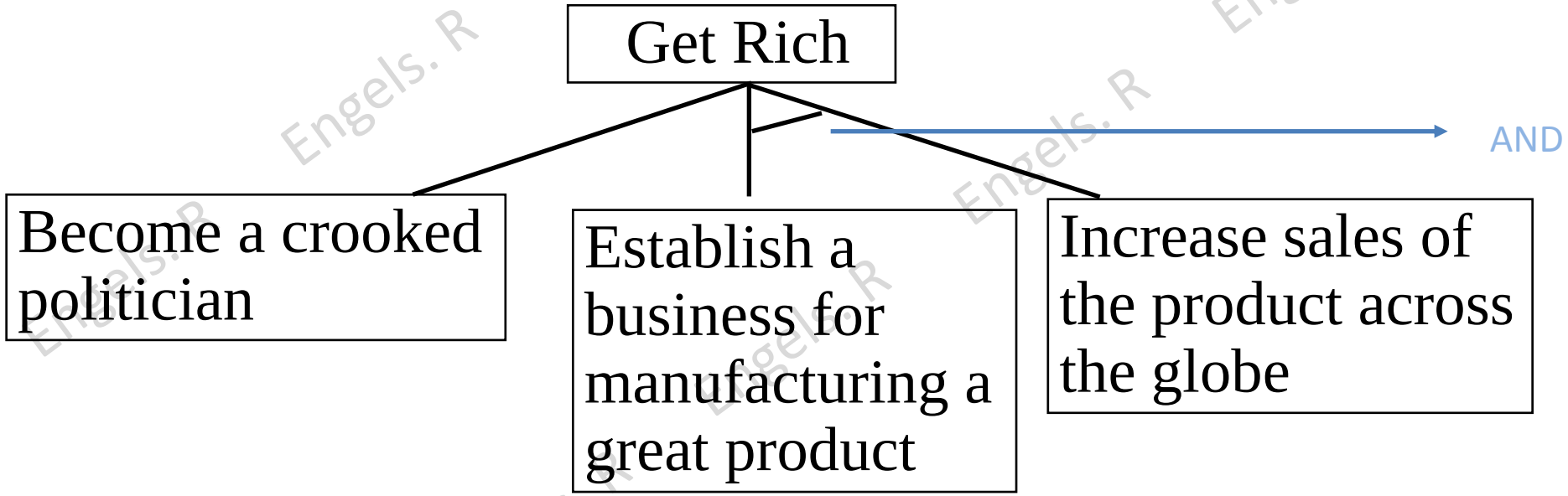
A*-summary

- Dijkstra's algorithm is a special case of A*, when we set h(v) = 0 for all v
- Path finding algorithms such as A* help in planning route (proactive) rather than lazy discovery of the problem (reactive)
- Slower than UCS or BF-Greedy, with more computations
- One of the most popular search algorithms in AI
  - Quality of results depends on quality of Heuristic used
- Where used?
  - AI Games
  - Robotics / Machine Learning

Engels. R

AND/OR graphs

- Some problems are best represented as achieving subgoals, some of which achieved simultaneously and independently (AND)
- Up to now, we only dealt with OR options



Engels. R

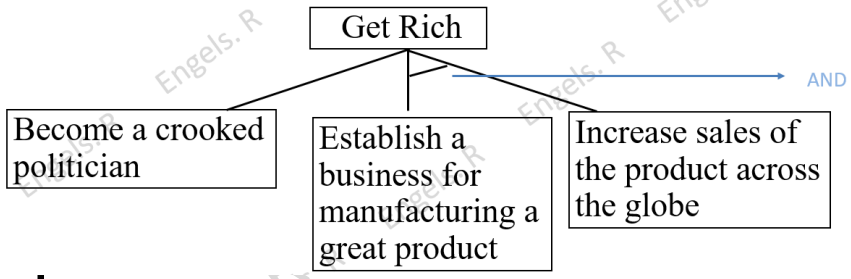
Searching AND/OR graphs

- A solution for an AND–OR search problem is a subtree that

(1) has a goal node at every leaf,

(2) specifies one action at each of its OR nodes, and

(3) includes every outcome branch at each of its AND nodes



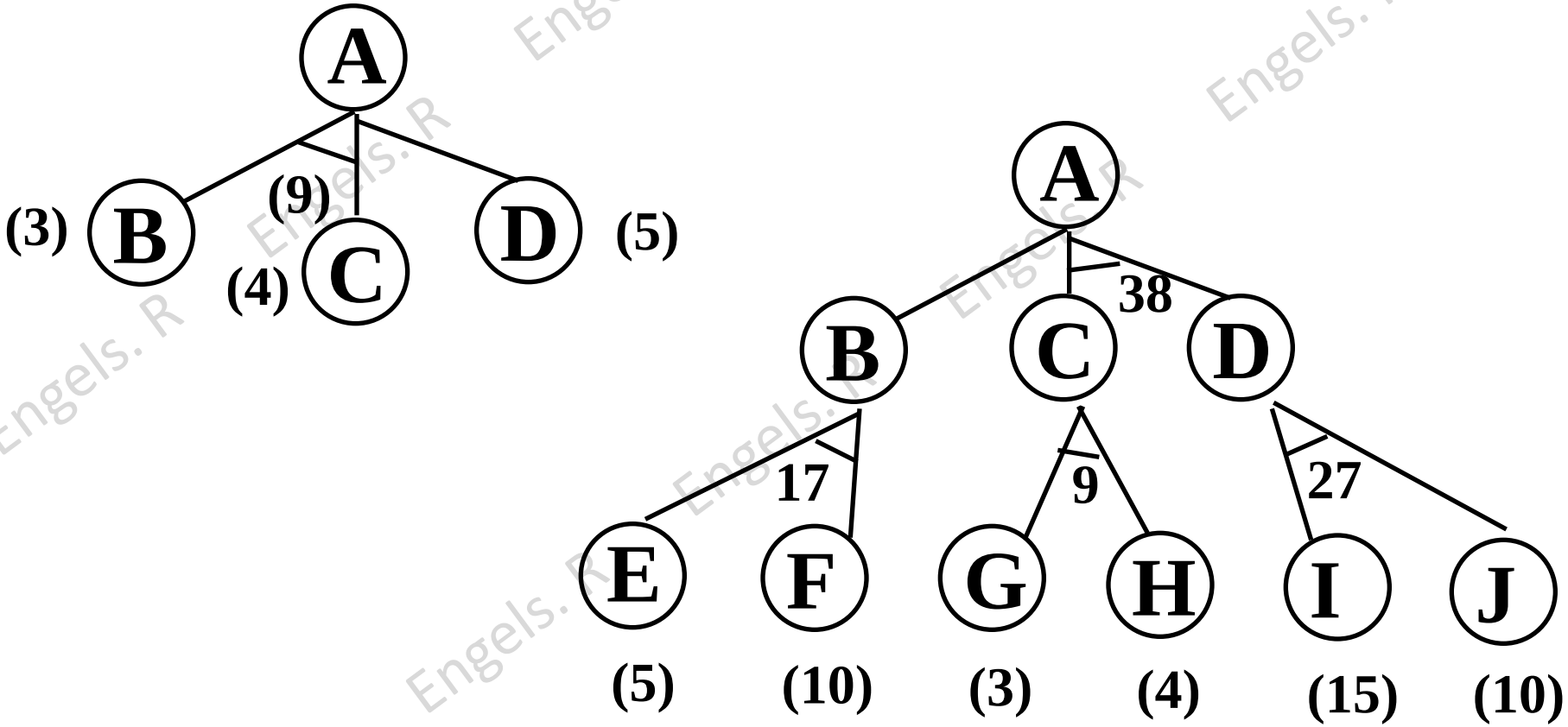
- Cost function: sum of costs in AND node
- $f(n) = f(n_1) + f(n_2) + ... + f(n_k)$

- How can we extend A* to search AND/OR trees?
- The AO* algorithm

Engels. R

AND/OR search

- We must examine several nodes simultaneously when choosing the next move



Engels. R

AO* algorithm

1. Let *G* be a graph with only starting node *INIT*
2. Repeat the followings until INIT is labeled SOLVED or h(INIT) > FUTILITY
   a) *Select an unexpanded node from the most promising path from INIT (call it NODE)*
   b) Generate successors of NODE. If there are none, set h(NODE) = FUTILITY (i.e., NODE is unsolvable); otherwise for each SUCCESSOR that is not an ancestor of NODE do the following:
      i. Add SUCCESSSOR to G.
      ii. If SUCCESSOR is a terminal node, label it SOLVED and set h(SUCCESSOR) = 0.
      iii. If SUCCESSOR is not a terminal node, compute its h

Engels. R

AO* algorithm (Cont.)

c) Propagate the newly discovered information up the graph by doing the following: let S be set of SOLVED nodes or nodes whose h values have been changed and need to have values propagated back to their parents. Initialize S to Node. Until S is empty repeat the followings:

i. Remove a node from S and call it CURRENT.

ii. Compute the cost of each of the arcs emerging from CURRENT. Assign minimum cost of its successors as its h.

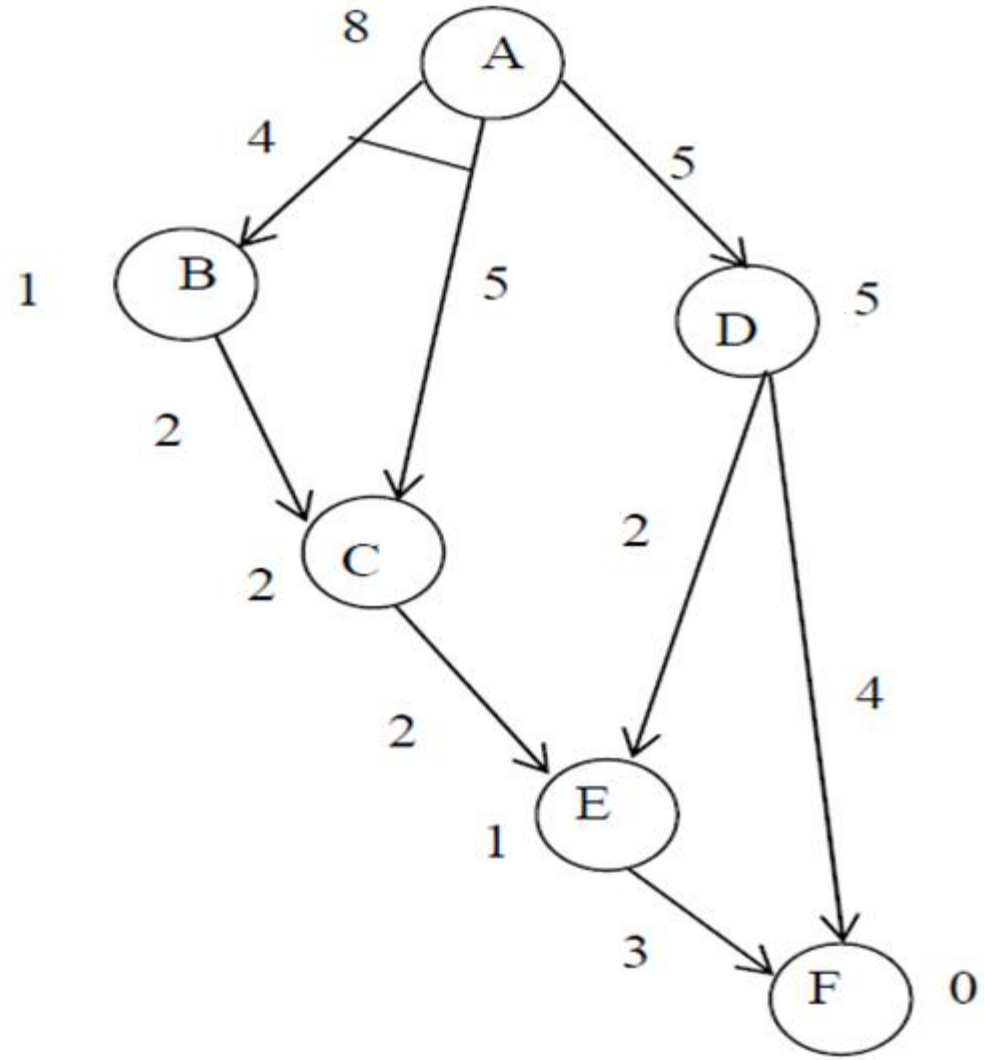
iii. Mark the best path out of CURRENT by marking the arc that had the minimum cost in step ii

iv. Mark CURRENT as SOLVED if all of the nodes connected to it through new labeled arc have been labeled SOLVED

v. If CURRENT has been labeled SOLVED or its cost was just changed, propagate its new cost back up through the graph. So add all of the ancestors of CURRENT to S.

Engels. R

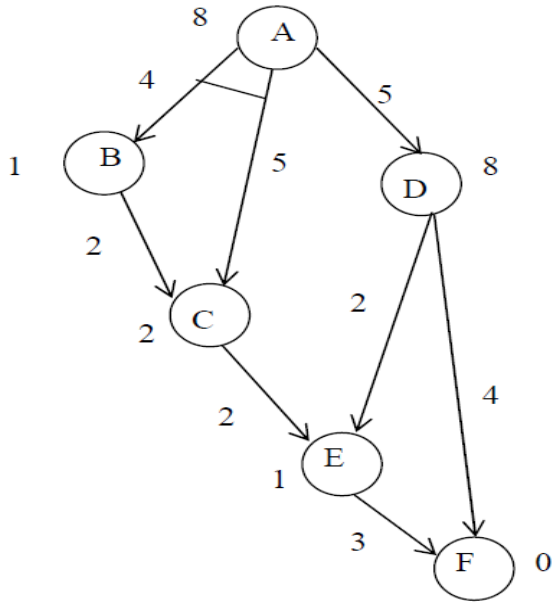
An Example



Engels. R

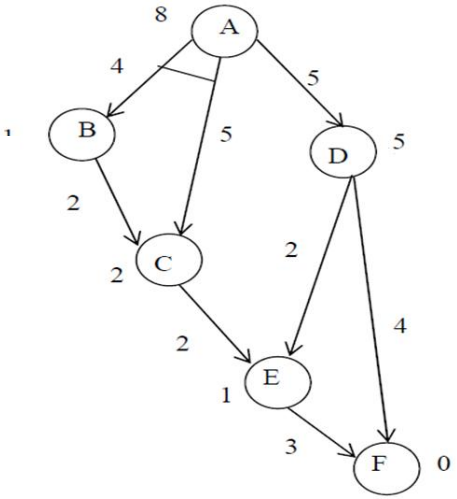
An Example

(8) Ⓐ



Engels. R

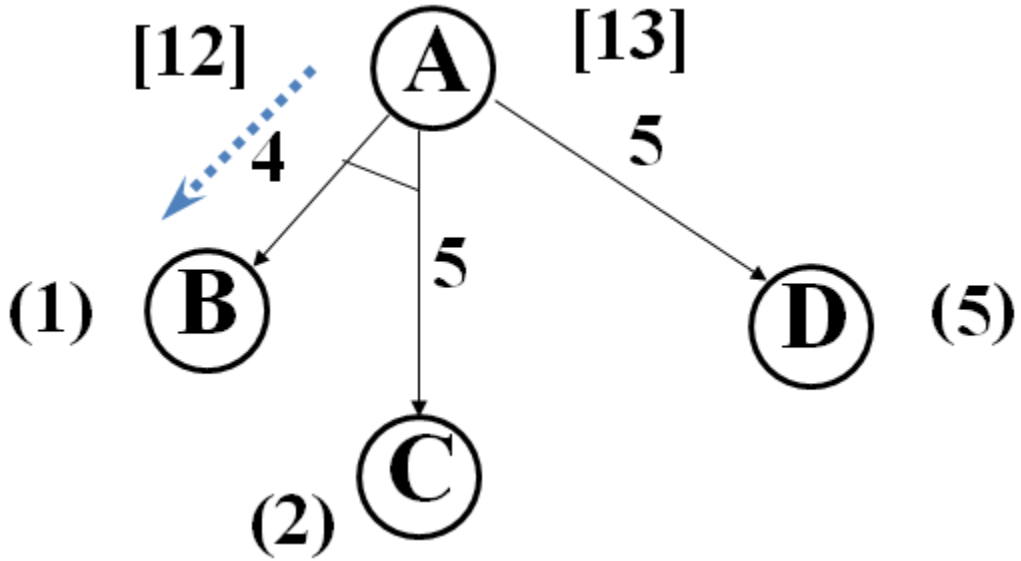
An Example



Only heuristic costs are visible. So taking the path towards B.

By moving to B the actual cost is realized (8 + 4 =12)

Then since there is an AND all paths beneath C and B should be completed.



Engels. R

An Example

Then since there is an AND all paths beneath C and B should be completed.

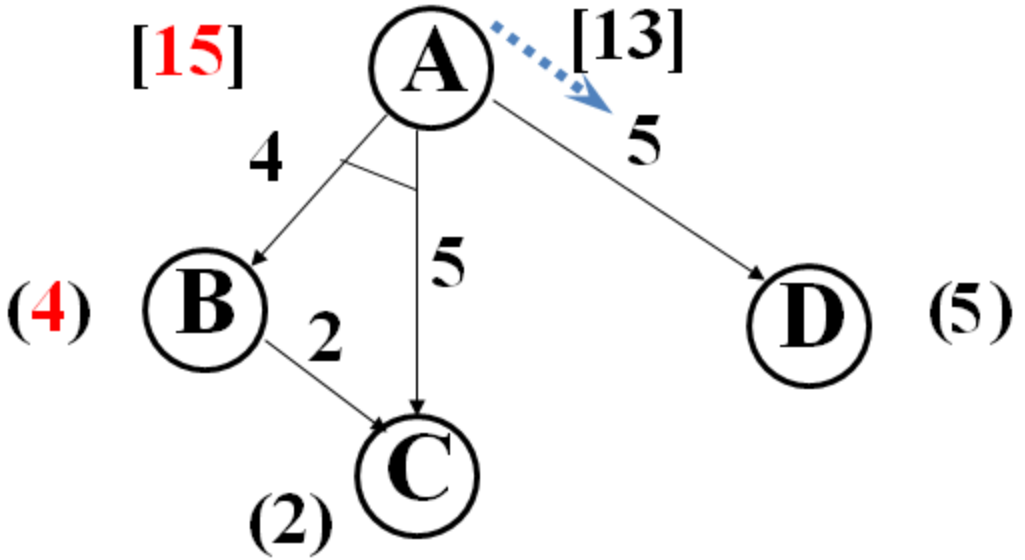
All heuristic values are updated with actuals, if explored

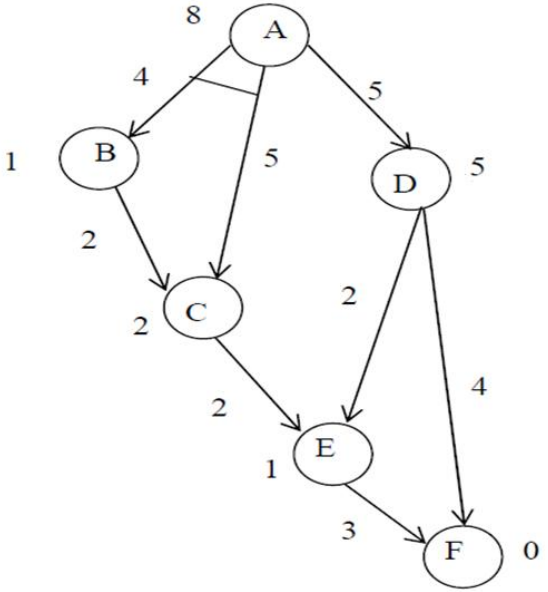
So from A to C, value is now 8+4+2 =14 through B

The heuristic from C to E adds 1 (So 14+1 = 15 for A-B-C-E)

Directly from A to C, cost is 5. So total (since AND) = 14+5 = 19

So explore towards D with heuristic cost 8





Engels. R

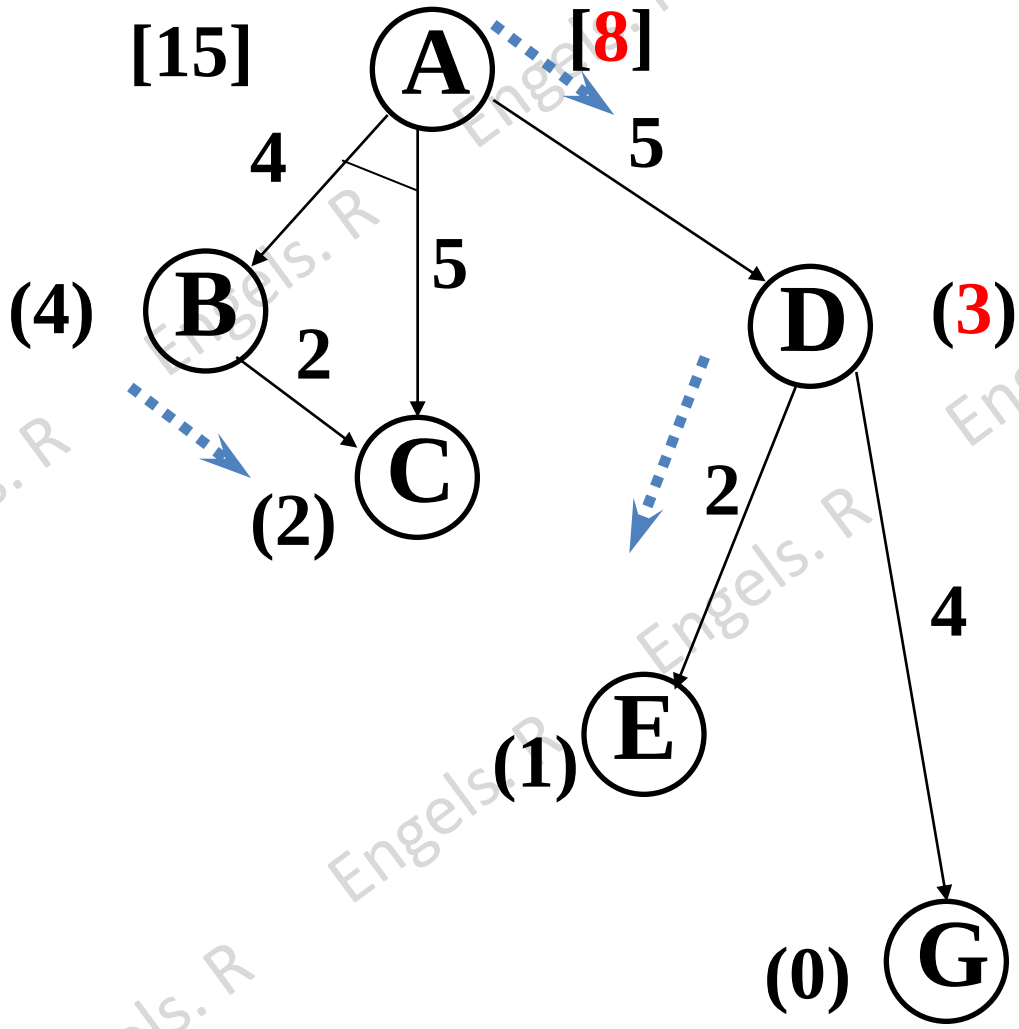
An Example

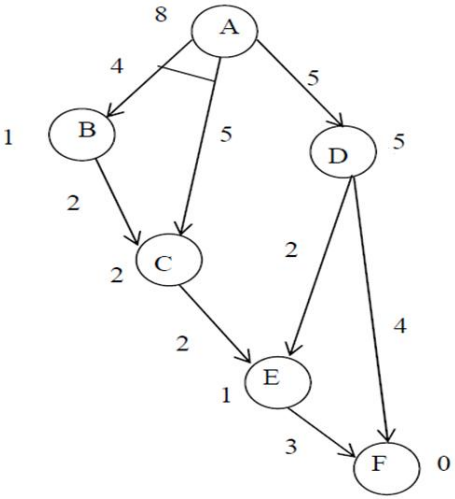
Exploring D with heuristic cost 8, actual cost becomes

8+5 =13

From D, either E or F are possible (F is also Goal State G), with heuristic 0

So that path is taken (8 + 5 + 4) = 17



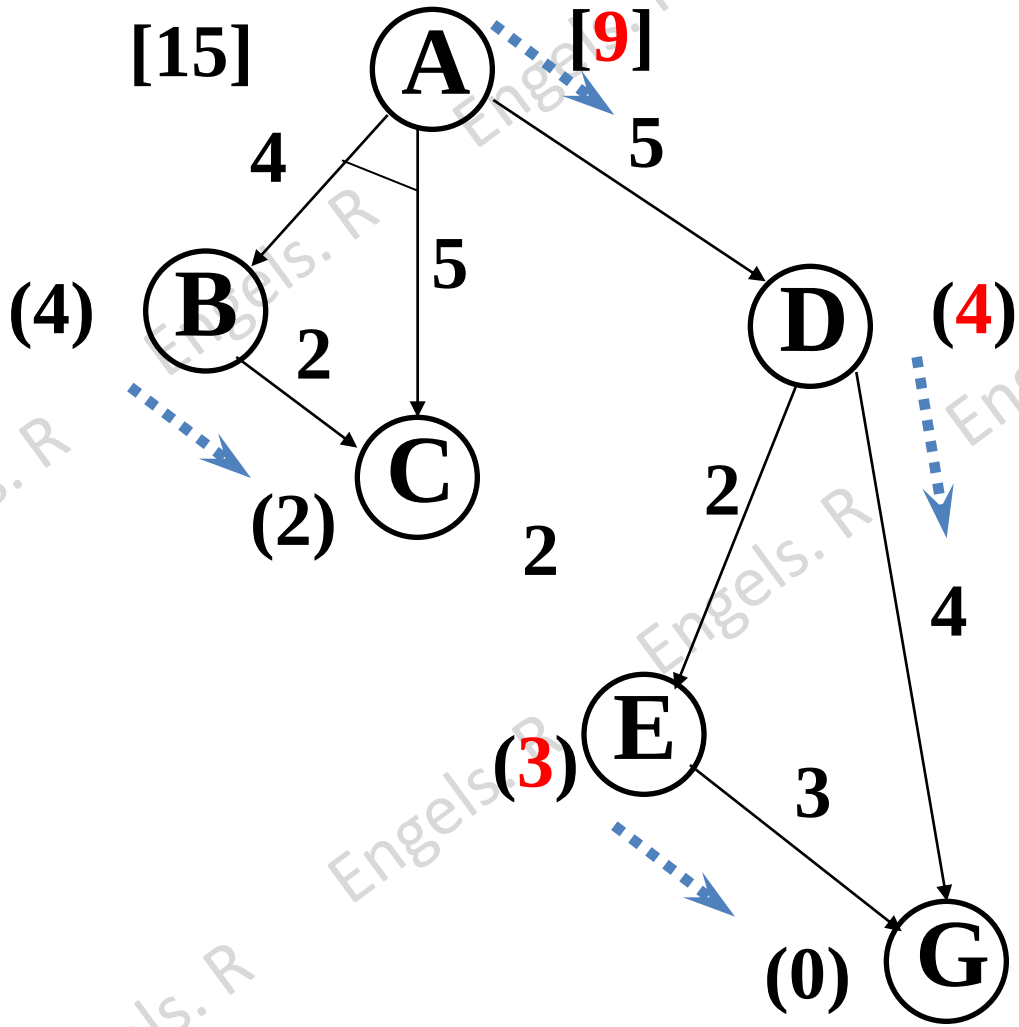


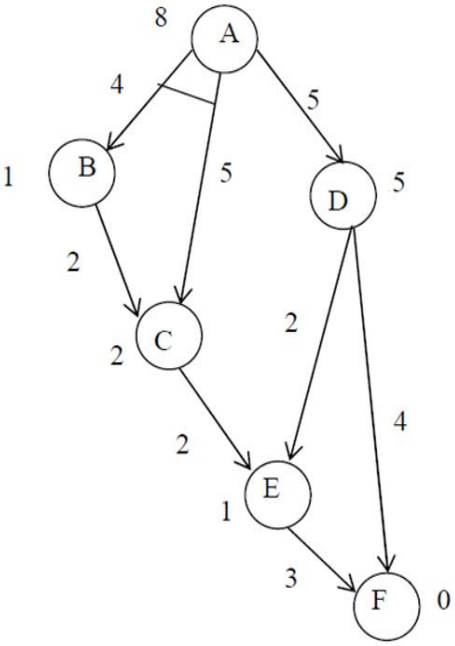
Engels. R

Exploring D further through E with heuristic 1, 8+5+2 =15

Finally from E to G, path is taken (8 + 5 + 2 +3) =18

An Example



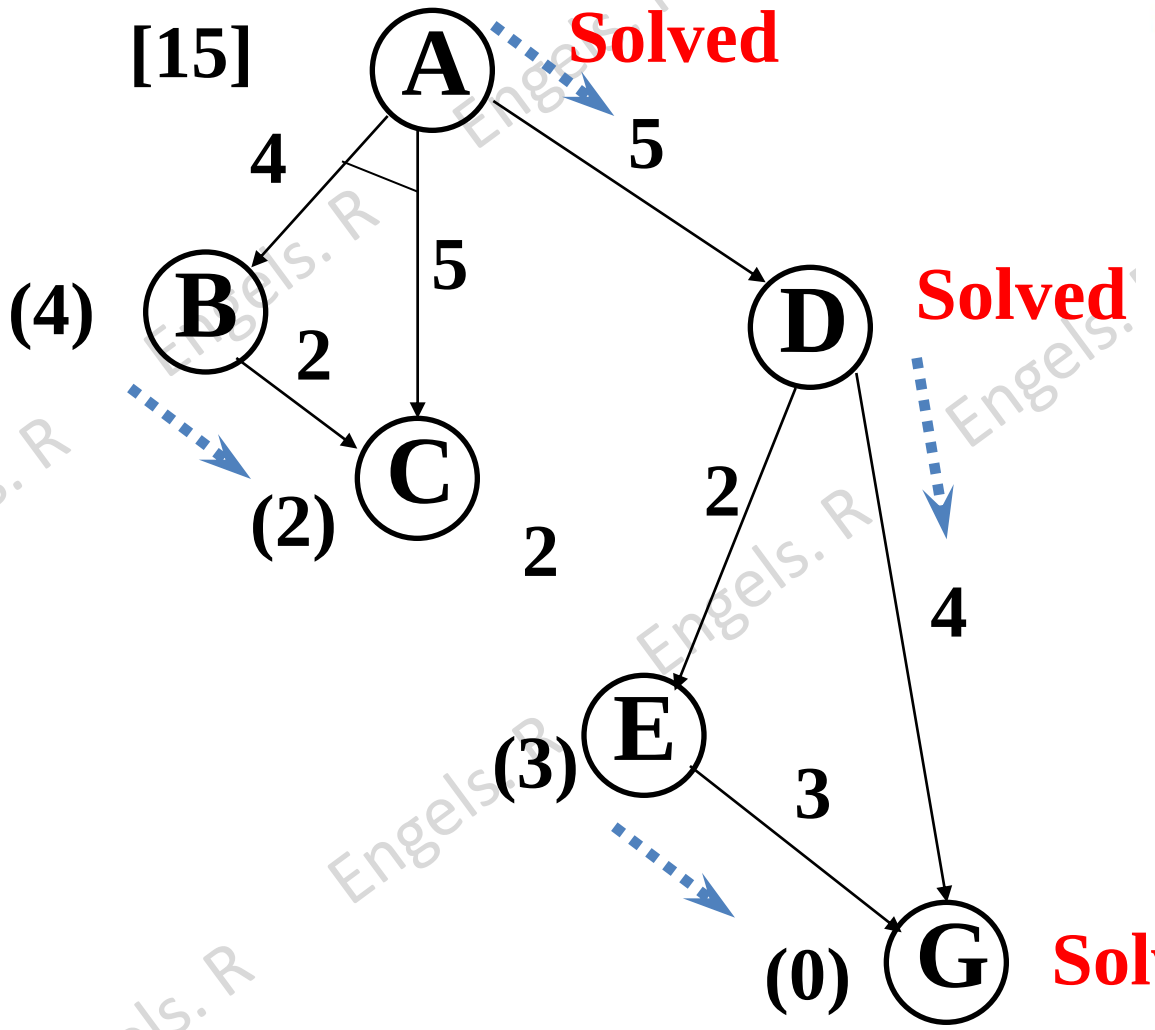


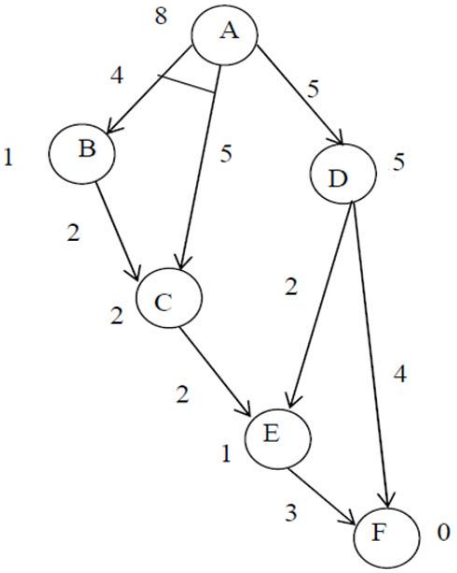
Engels. R

Final cost is 8 + 5 + 4 = 17

Exploring C is not necessary as by moving from C to E the cost is not going to be better (A-C is costlier than A-D)

An Example





Engels. R

Summary

- Agent
- Agent function
- Agent program
- Rational agent
- Task environment
  - **P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors (PEAS)
- Types of agents
  - TDA, Simplex reflex, model-based, goal-based, utility based, learning agent

Engels. R

References

- AIMA
  - Artificial Intelligence - A Modern Approach 3rd Edition - RUSSELL & NORVIG

Engels. R