Artificial Intelligent Systems

(BMA-EL-IZB-LJ-RE 1. year 2024/2025)

Intelligent problem Solving

Simon Dobrišek

Lecture Topics

- General problem solving
- Graph representation of problems
- Examples of single state problems
- Solving problems by a tree search method
- Algorithms for searching trees
- Solving problems by decomposing them into sub-problems.

General Problem Solving

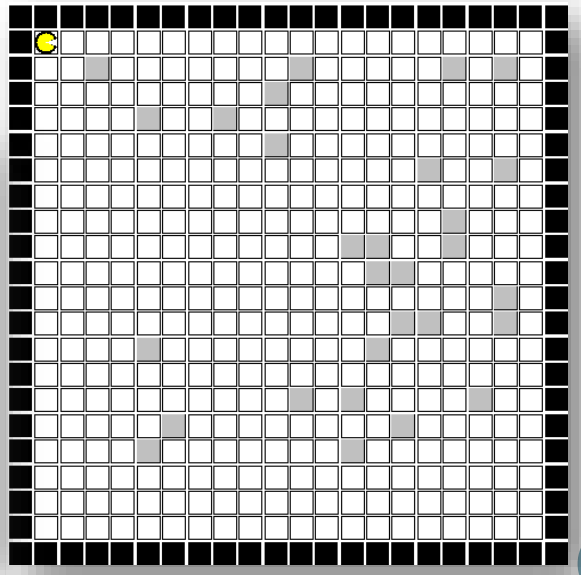
- Problem solving is a characteristic of many systems based on the concepts of artificial intelligence.
- In real life, not many problems are easily solved.
- Problem solving is the process of generating a solution from the observed data.
- Problem is characterized by a set of goals, a set of objects and their relations as well as a set of operations on objects that are poorly defined and may evolve during problem solving.

Types of Problems

- Deterministic and fully observable (single-state problems).
  - *An agent is fully aware of the current state in solving the problem, the solution is a sequence of operations.*
- Non-observable (conformant problems).
  - *An agent may have no idea what is the current state in solving the problem and may only reason about the consequences of the operations (has no sensors), the solution (if any) is a sequence of operations.*
- Nondeterministic and/or partially observable (contingency problem).
  - *Percepts provide new information about the current state, search and execution phases are interchanging, the solution is a tree or a contingent plan (a policy).*
- Nondeterministic and non/observable (exploration problem)
  - *Unknown problem state space (online search), discovering and learning about the environment without taking any actions.*

Solving Single-State Problems

- Simple problems are usually deterministic and fully observable single-state problems.
- Agent's world (environment), where problems need to be solved, can be represented by a discrete set of possible states.
- Possible agent's actions/operations that modify the world are represented by as set of discrete operators.
- Agent's world is static (unchanging) and deterministic.



Typical Examples of Problem Solving

- Assembling products in industrial production.
- Defining the optimum layout of the building blocks of integrated systems.
- Determining the optimal paths with visiting points in space.
- Rearranging (putting in order) an environment into a final desired state.
- Making a series of moves that leads to a victory in a game.
- Proving theorems in mathematics and logics.
- …

Problem Space

- A problem space is an abstract space.
- A problem space includes all the possible states of the problem that can be generated by any combination of operators on any combination of objects.
- A problem space may contain one or more of final states that represent a solution to the problem.
- Finding the solution is searching for the final state in a problem space that represent the solution.
- Search is carried out with different search strategies.

Planning Problem Solving

- Precise identification of the problem - defining the initial situations as well as the final situations that could represent acceptable solutions to the problem.
- Analysis of the problem – defining the most important characteristics of the problem that may have impact on the choice of the search strategy for finding a solution.
- Determination and presentation of the entire knowledge necessary to solve the problem (operations, objects, states, relations, etc).
- Choosing the best method of solving the problem according to the preliminary findings.

Formal Definition of the Problem

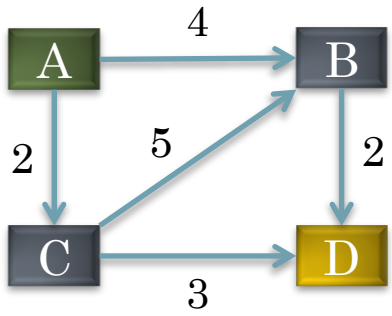
- A formal definition of a problem requires the following:
  - Defining a state space that contains all the possible configurations of the relevant objects that are part of the problem.
  - Specifying one or more initial states that describes possible situations from which the problem-solving process may start.
  - Specifying one or more final states that describes all the possible situations that are considered to be acceptable solutions to the problem.
  - Defining a set of rules that describe all the possible actions (operations) on the problem state space, and usually their costs as well.

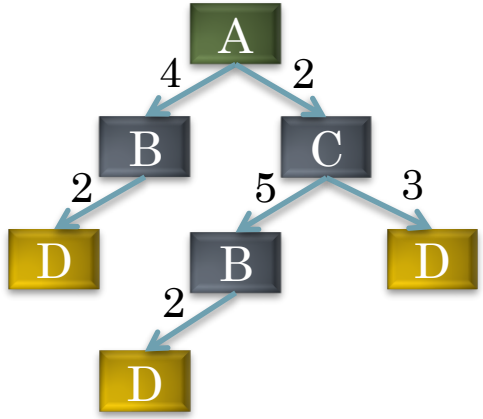
Problem Solving

- The problem is solved by using the rules and actions in combination with an appropriate strategy of traversing/moving through the problem space until a path from an initial state to the final state is found.
- This process is know as search.
- Search is a fundamental method to solve problems that are not solvable with more simple direct methods.
- Search provide a framework into which more simple direct methods for solving sub-problems can be embedded.
- A large number of AI problems are formulated as search problems.

Graph Representation of Problems

- A problem space is usually represented by a directed graph, where nodes represent problem states and directed arcs represent the operators that change the states.
- In order to simplify the search algorithm, it is often useful to represent the problem logically and programmatically as a tree.
- A tree usually decreases the complexity of a search, but at a cost, as nodes in the tree are duplicated.





Search Strategies

- Problem solving is characterized by a systematic search from the initial to the final state in the problem state space.
- Specific search strategies are developed for a given problem or general search strategies are used that incrementally reduce the difference/distance between the current problem state and the final problem state that is considered a solution.
- A search strategy is determined by the choice of the order in which graph or tree nodes are open/expanded.
- Search strategies are evaluated based on their completeness, computational complexity and optimality.

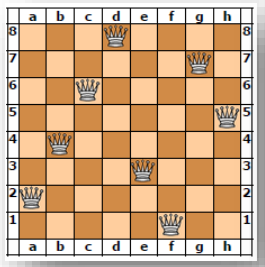
An Example Formulation of a Problem

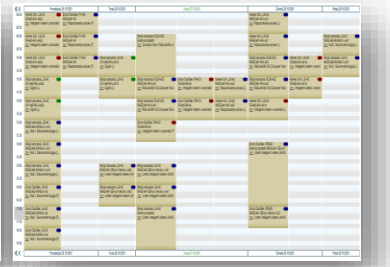
- A problem of finding an optimal road route can be formulated in four points by defining:
  - The initial state, e.g. *„in Ljubljana“*,
  - The function “successor” $N(x)$ that defines a set of successor/child states of the predecessor/parent state x,
    e.g. $N(\text{„in Ljubljana“}) = \{\text{„in Domžale“}, \text{„in Medvode“}, \text{„in Dobrova“}, \dots\}$,
  - The verification rule for the goal state, which can be defined explicitly, e.g. $(\text{x} == \text{„in Celje“})$ or implicitly, e.g. $HasCastle(x)$,
  - The costs for traversing from state to state, e.g. the cost for changing from x to y is denoted by $c(x, a, y)$, where $a \geq 0$ is the cost that could be the distance in kilometers or the number of crossroads on the route etc.
- The solution is a sequence of actions (state changes – a path) with the lowest cumulative cost that leads from the initial to the final state.

Examples of Single State Problems

- The problems for which the solution is a description of the path from the initial to the final state:
  - Assembling industrial products,
  - Finding an optimal route on maps,
  - Assembling a puzzle picture,
  - *The problem of the Tower of Hanoi* ,
  - *The Water Jugs problem* etc.

- The problems for which the solution is only a description of the final state:
  - Finding an optimal layout of elements (integrated circuits etc),
  - Defining the optimal time schedule for different parties,
  - A game of n-queens puzzle,
  - The Sudoku game etc.





The problem of the Tower of Hanoi

- The Tower of Hanoi is a mathematical game (puzzle) that consists of three rods, and a number of disks of different sizes which can slide onto any rod.
- The puzzle starts with the disks in a neat stack in ascending order of size on one rod making a conical shape.
- The objective of the puzzle is to move the entire stack to another rod, while obeying the following simple rules:
  - Only one disk can be moved at a time.
  - Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack.
  - No disk may be placed on top of a smaller disk.

Formulation of the Hanoi Tower problem

- The Hanoi Tower problem is solved by providing a problem state space, where each state represents one possible arrangement of the disks on the rods.
- The initial state of the problem represents the initial arrangement (the tower on the first rod).
- The final state of the problem represents the final arrangement of disks (the tower of the third rod).
- The problem state space is then represented as a directed graph, where nodes represent the current arrangement of disks on the rods (problem states) and arcs represent the operators, i.e., the possible movements of the disks in accordance with the rules.

Operators of the Hanoi Tower problem

- ***P12***: Move the disk from the first rod to the second rod!

  Condition: The second rod is empty or the diameter of the disk on the second rod is larger than of the disk on the first rod.

- ***P13***: Move the disk from the first rod to the third rod!

  Condition: The third rod is empty or the diameter of the disk on the third rod is larger than of the disk on the first rod.

- ***P21***: Move the disk from the second rod to the first rod!

  Condition: The first rod is empty or the diameter of the disk on the first rod is larger than of the disk on the second rod.

- ***P23***: …

- …

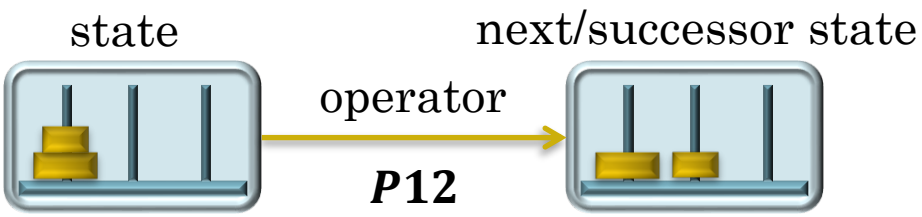
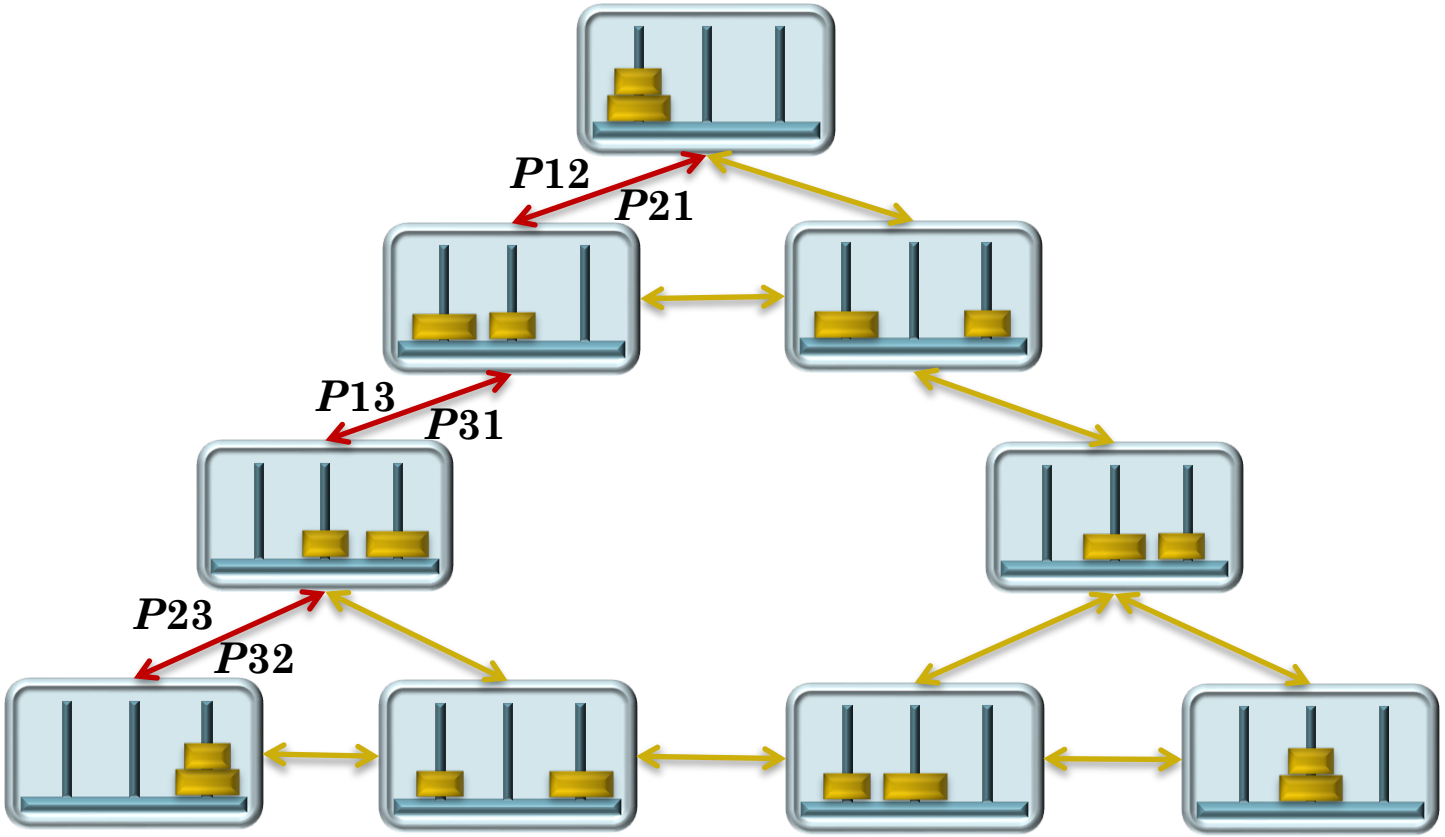


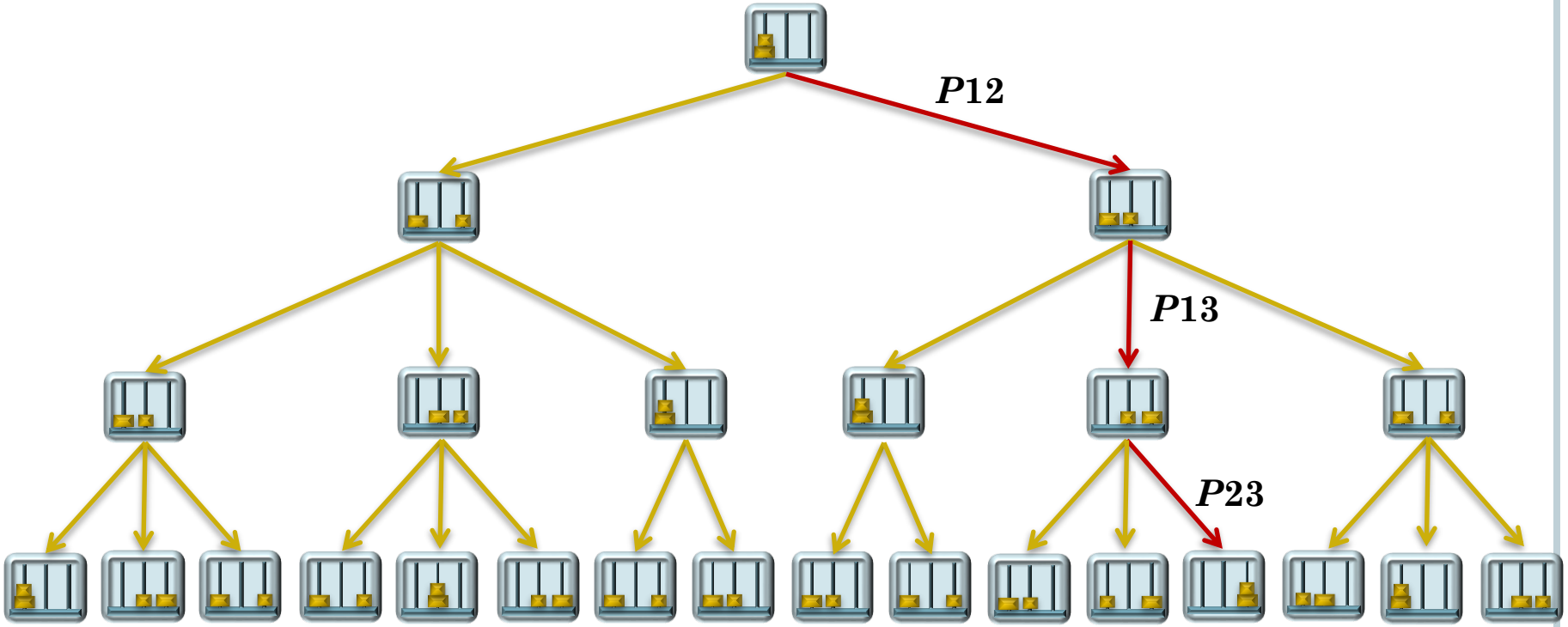
Illustration of the Hanoi Tower Problem

- The Hanoi Tower problem with two disk has nine states.
- The problem state space can be represented with a (bidirectional) directed graph.



The Tree Representation of the Problem

- The problem state space can be represented by a tree with a certain depth as well.



Notes on the Hanoi Tower Problem

- The path marked in red indicates a solution to the problem.
- The tree representation incorporates many more nodes than the graph representation.
- In the tree representation, the same problem state is represented by several duplicated nodes.
- The tree is much larger, however, it enables its representation in an implicit form.
- The implicit form allows a simultaneous creation of the nodes during searching for the optimal (shortest) path from the initial to the final node (state).

The Water Jugs problem

- Two water jugs are available, one with a volume of three and the other with a volume of four liters.
- Water can be poured into any of the two jugs from a tap, poured from one jug to another, or spilled away.
- The goal is to achieve exactly two liters of water in the four liter jug.

The State Space of the Water Jugs Problem

- Each of the problem states is defined by the amount of water in each of the two jugs.
- The problem state can be represented by two real variables, denoted as V_3 and V_4, indicating the amount of water in one of the two jugs.

$$0 \leq V_3 \leq 3\,, \quad 0 \leq V_4 \leq 4$$

- The initial state is indicated by the values

$$V_3 = 0\,, \quad V_4 = 0$$

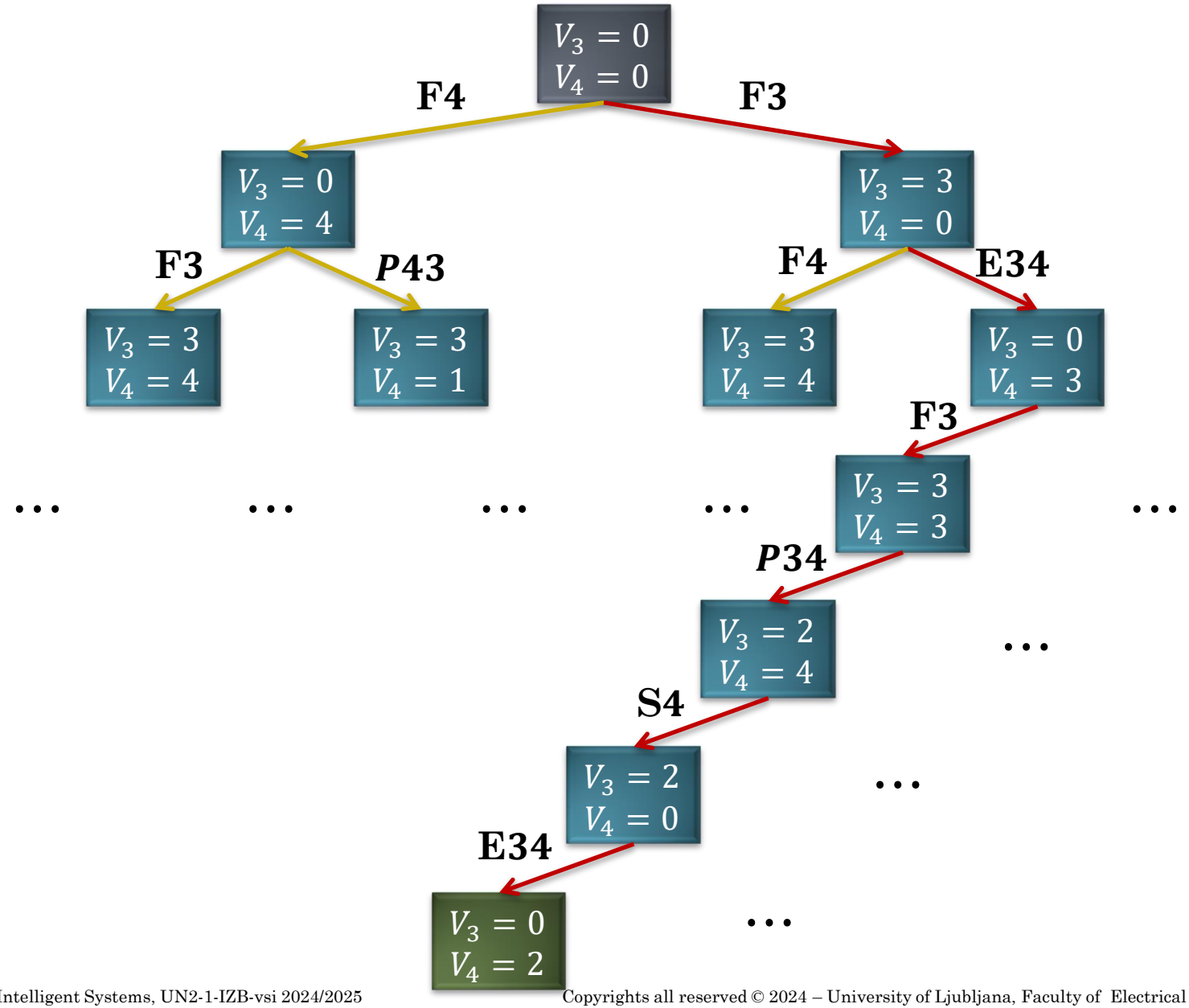
- The final state is indicated by the value of only the second variable (the value of V_3 is not relevant)

$$V_4 = 2$$

The Operators of the Water Jugs Problem

- **F4**: Fill the jug V_4!
  Condition: $V_4 < 4$ Consequence: $V_4 = 4$
- **S4**: Spill all the water from the jug V_4!
  Condition : $V_4 > 0$ Consequence : $V_4 = 0$
- ***P*43**: Pour the water from the jug V_4 into V_3 until it is full!
  Condition : $V_3 < 3 \wedge V_4 \geq 3 - V_3$ Consequence : $V_4 = V_4 - (3 - V_3), V_3 = 3$
- ***P*34**: Pour the water from the jug V_3 into V_4 until it is full!
  Condition : $V_4 < 4 \wedge V_3 \geq 4 - V_4$ Consequence : $V_3 = V_3 - (4 - V_4),\ V_4 = 4$
- **E34**: Empty the jug V_3 into V_4!
  Condition: $V_3 + V_4 < 4 \wedge V_3 > 0$ Consequence : $V_3 = 0,\ V_4 = V_3 + V_4$

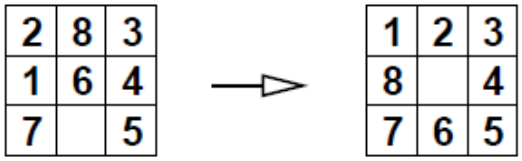
The Tree Representation of the Problem



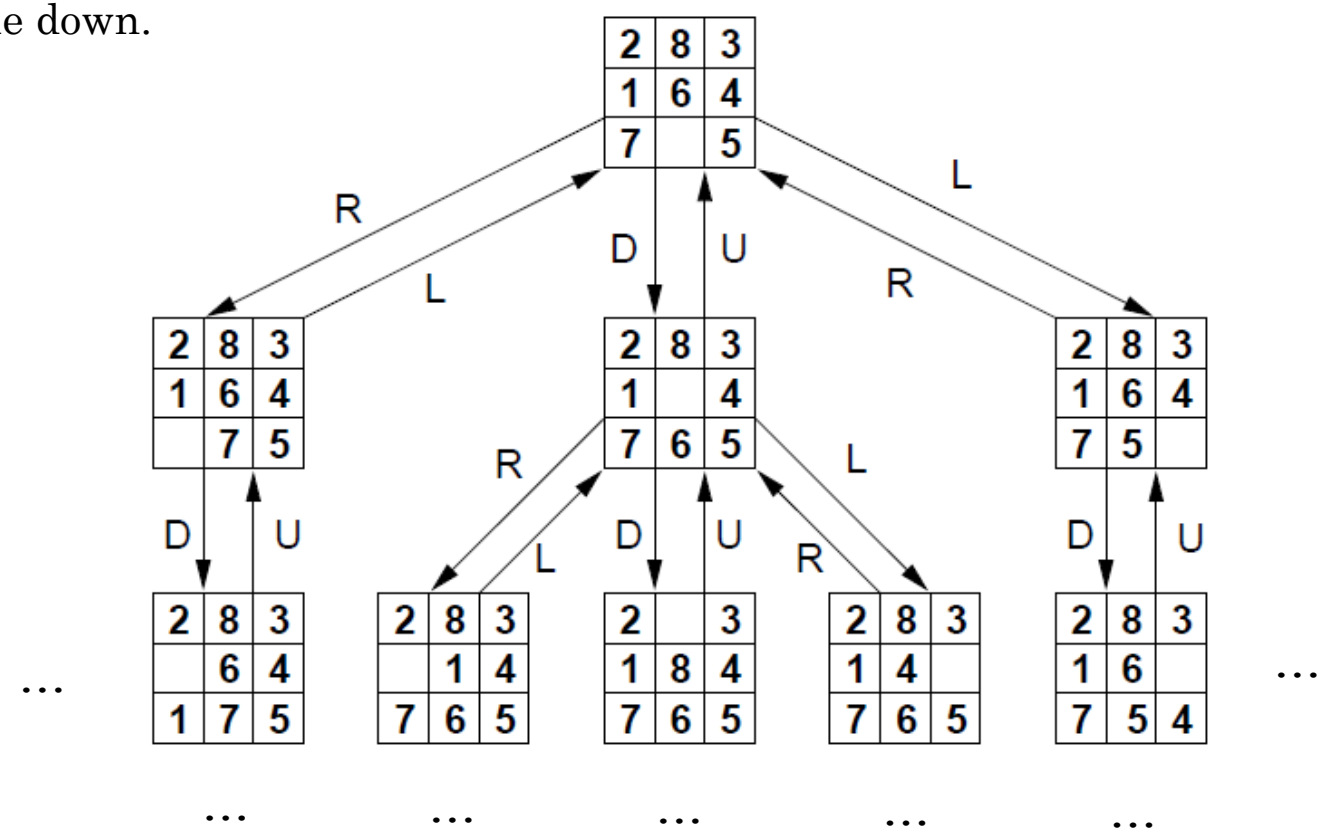
The problem of 8-sliding-tile puzzle

Operators:

- L: Move the tile to the left.
- R: Move the tile to the right
- U: Move the tile up.
- D: Move the tile down.

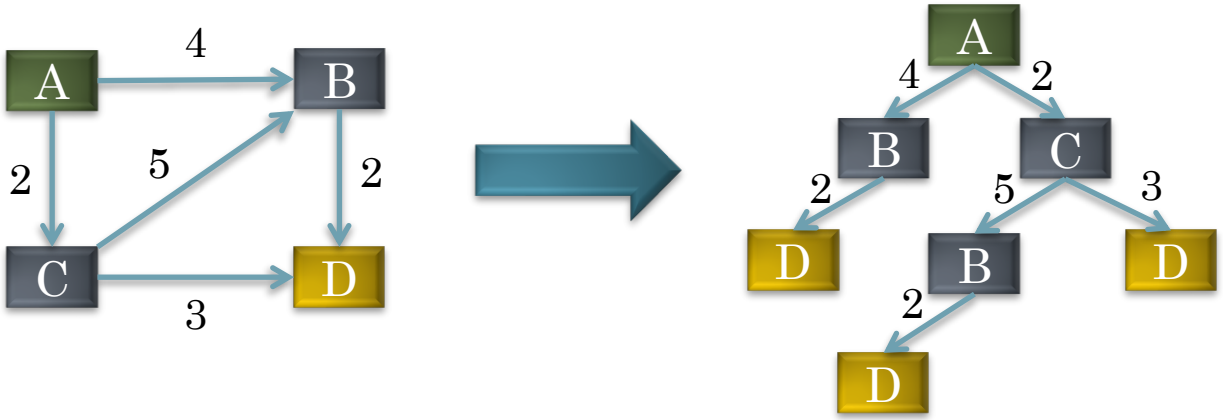


The initial state The final state



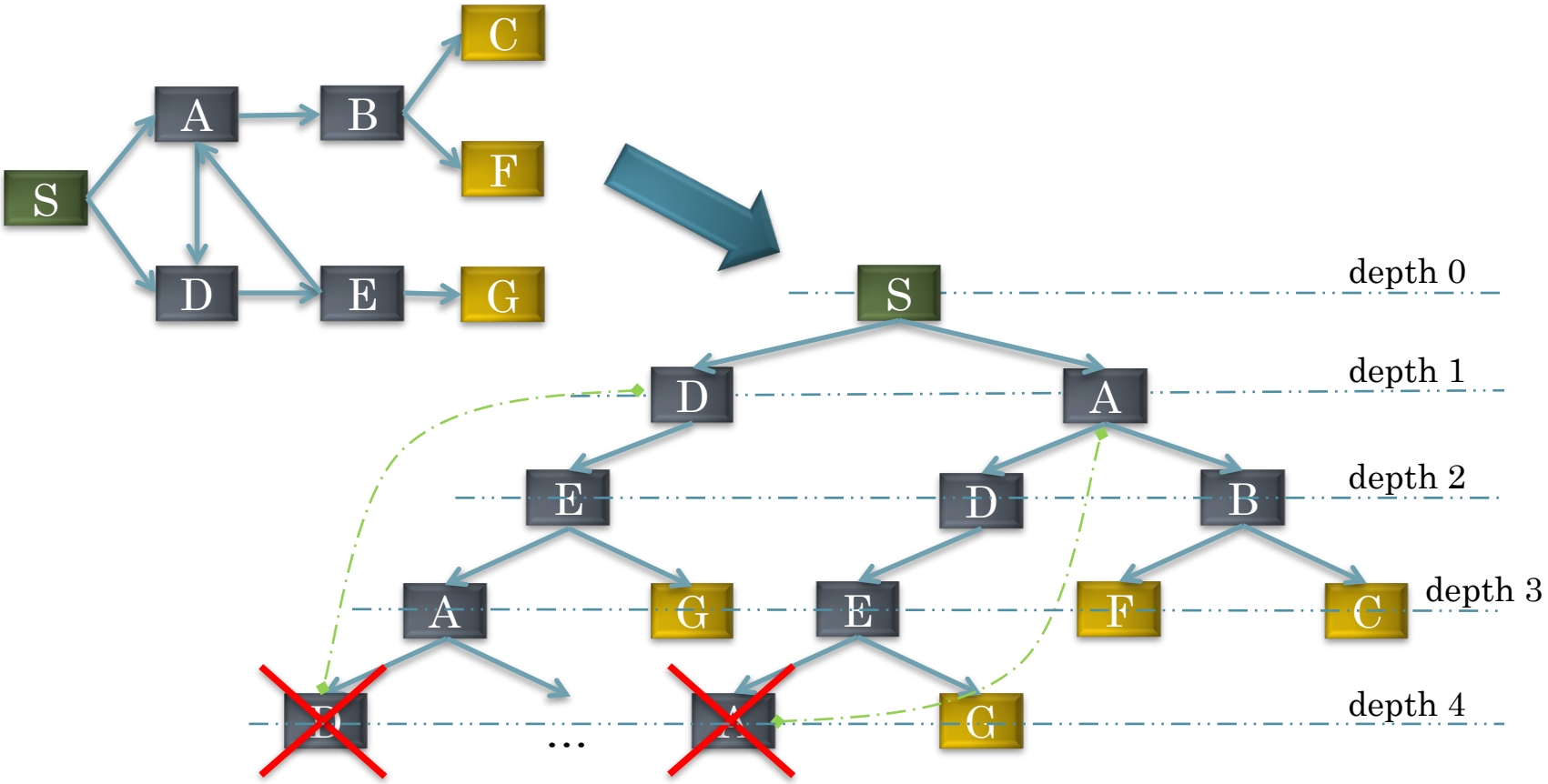
Converting directed graphs to trees

- A set of all possible paths in a directed graph can be represented as a tree.
  - A tree is a directed graph, where all the tree nodes except the root have exactly one predecessor node, called its parent.
  - The tree root is the node that has no predecessor node.
  - The tree leaves are the nodes that have no successor nodes.
  - The branching factor of a tree is the average number of node successors.
- A directed graph is converted to a tree by duplicating nodes and breaking cyclic paths, if they exist.



Converting directed graphs to trees

- First, the initial (root) node is selected and then all the paths to the final nodes (leaves) or nodes already visited are traced through the graph.



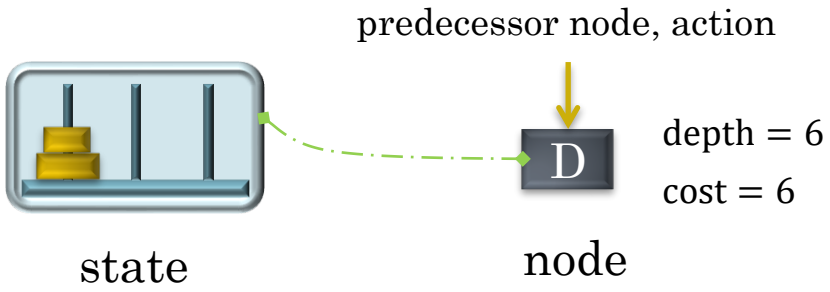
Searching for Problem Solutions

- A problem is solved by searching and expanding its tree nodes.

```
function tree_search(problem, strategy) returns a solution or failure
   initialize the tree frontier using the initial problem state
   loop do
      if the tree frontier is empty (it contains no node to be expanded)
         then return failure
      choose a node from the frontier in accordance with the strategy
      if the chosen node contains the goal state
         then return the corresponding solution
      else expand the chosen node, remove it from the frontier and
      add all its successor nodes
   end
```

A RELATION BETWEEN PROBLEM STATES AND NODES

- A state is a representation of a (physical) configuration of the problem.
- A node is a data structure that includes the identity of the predecessor node and the identity of the successor nodes, as well as the state, path cost, action and depth.
- The states themselves has no predecessors, successors, costs and depths.
- Expanding a node creates new nodes, fills in the various data fields and uses the successor function $N(x)$ of the problem to create the corresponding successor states.



Basic Tree Search Strategies

- Uninformed search strategies use only the information available in the problem definition.
- Search can be exhausted (complete and optimal) or heuristic (not necessarily complete and optimal).
- There are a number of proposed search strategies, such as:
  - Breadth-first search
  - Depth-first search
  - Uniform-cost search
  - Depth-limited search
  - Iterative deepening search
  - Informed search with the use of a heuristic function
  - …

Evaluation of Search Strategies

- Strategies are evaluated using the following criteria:
  - Completeness: Does it always find a solution, if it exists?
  - Time complexity: The number of nodes generated/expanded.
  - Space complexity: The number of nodes holds in memory.
  - Optimality: Does it always find a least-cost solution?
- Time and space complexity are measure 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 infinity)

Computational Complexity of Algorithms

- Computational complexity of a search strategy is defined by the time and space complexity of its algorithm.
- A theoretical measure of the time or space complexity of a given algorithm, which processes n data items, is denoted by $O(f(n))$.
- The complexity of $O(n)$ indicates that the complexity is proportional to the number of the data items.
- The time t, necessary for the execution of the algorithm, is in this case $t \propto n$, i.e., $t = kn$, where k is a non-negative constant.
- Similarly, the complexity of the algorithm can be $O(n^2)$, $O(2^n)$, $O(\log_2(n))$, $O(n \log_2(n))$ etc.

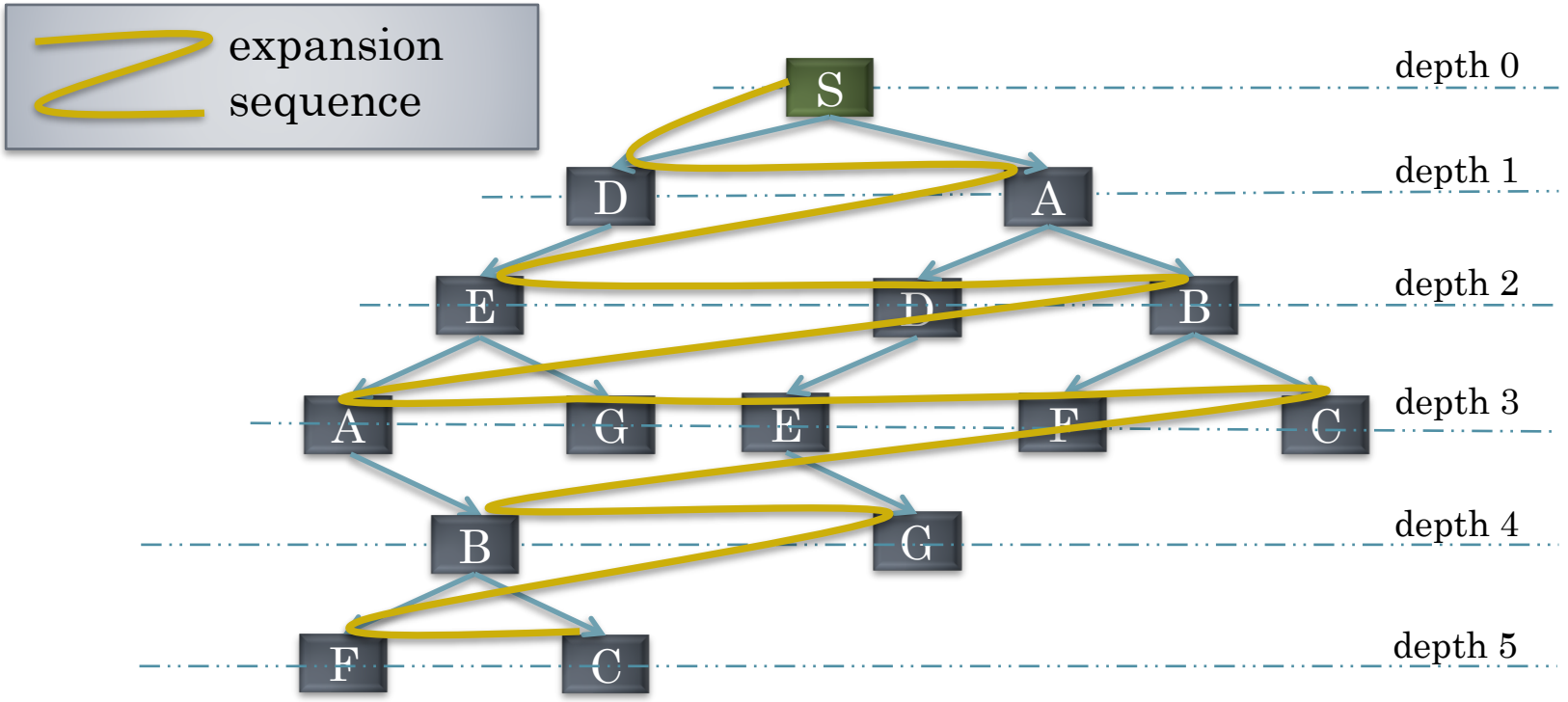
An Example Measurement of the Computational Complexity

Commands	No. operations
`min = a[0][0]`	1
`for(i=0 ; i<n ; i++)`	$1 + 2n$
`  for(j=0 ; j<n ; j++)`	$n * (1 + 2n)$
`  if(min > a[i][j])`	$n * n$
`  min = a[i][j]`	$< n * n$
`print min`	1

- The total number of operations is thus somewhere between $3n^2 + 3n + 3$ and $4n^2 + 3n + 3$.
- Constants are usually not taken into consideration, and it can thus be concluded that the complexity of the above algorithm is $O(n^2)$.

Breadth-first Search

- The strategy is based on the principle – *„Expand the shallowest unexpanded node!“*
- The use of a FIFO queue of open nodes, i.e., newly expanded successor nodes go to the end of the queue.

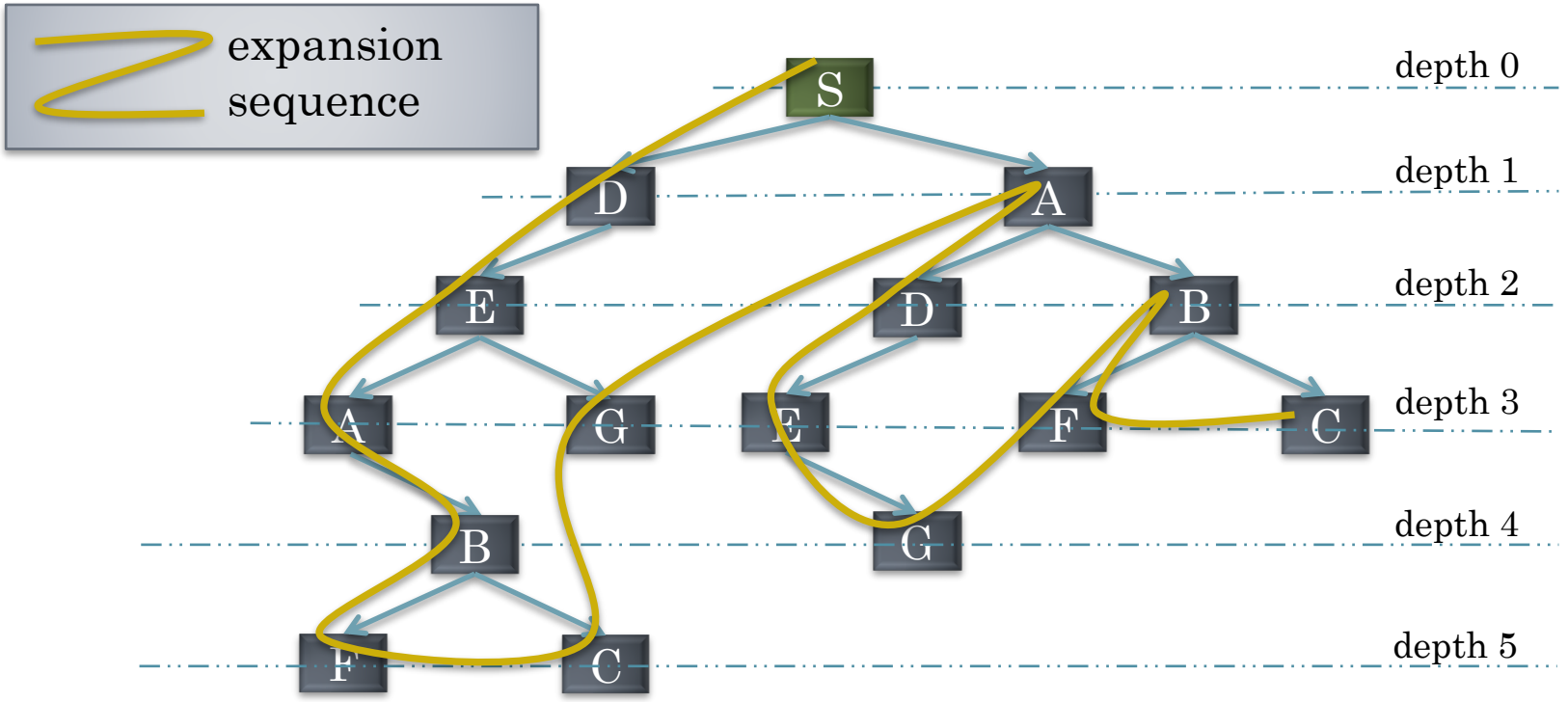


Characteristics of the Breadth-first Search

- Let b denotes the maximum branching factor of the tree, d the depth of the final nodes with the lowest cost path and m the maximum depth of the state space (may be ∞)
- The strategy is complete, if b is finite.
- The time complexity of the algorithm is $O(b^{d+1})$.
- The space complexity of the algorithm is $O(b^{d+1})$ as well.
- The strategy is optimal if all the operation costs equal to 1.
- The biggest drawback of this strategy is the space complexity of the algorithm.

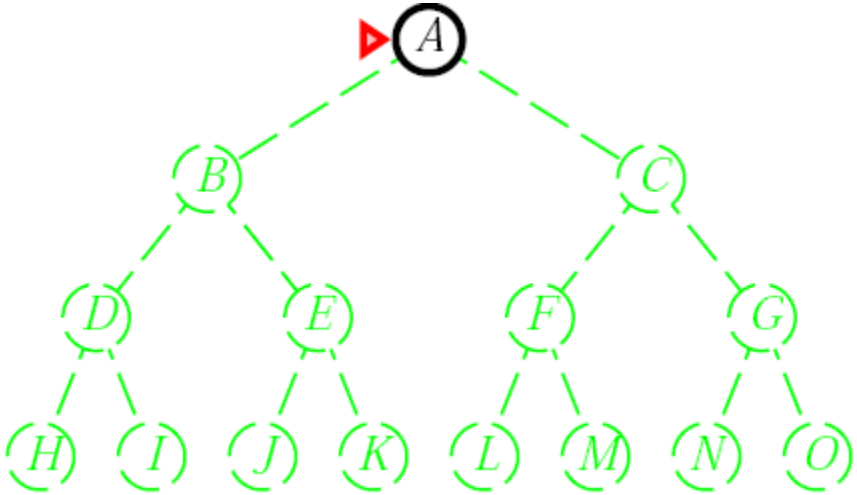
Depth-first Search

- The strategy is based on the principle – *„Expand the deepest unexpanded node!“*
- The use of a LIFO queue of open nodes, i.e., newly expanded successor nodes go to the beginning of the queue.



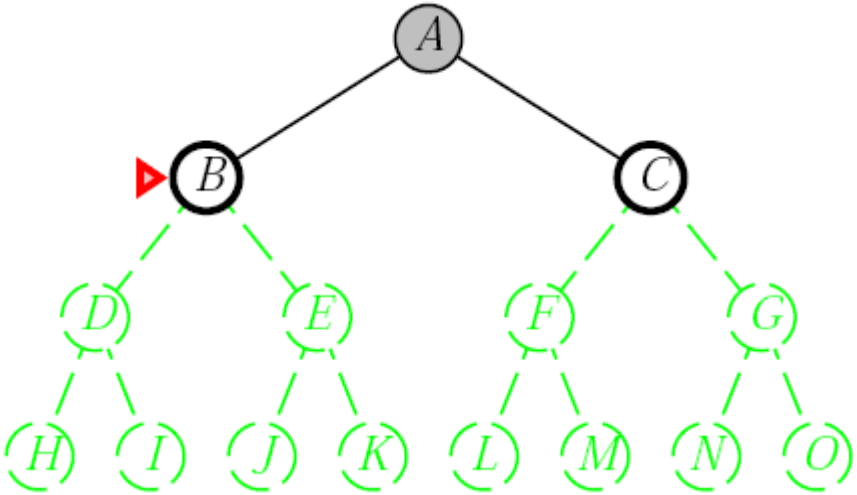
Depth-first Search Illustration

- Illustration of the depth-first search algorithm (strategy), where the investigated, expanded and closed nodes are highlighted.



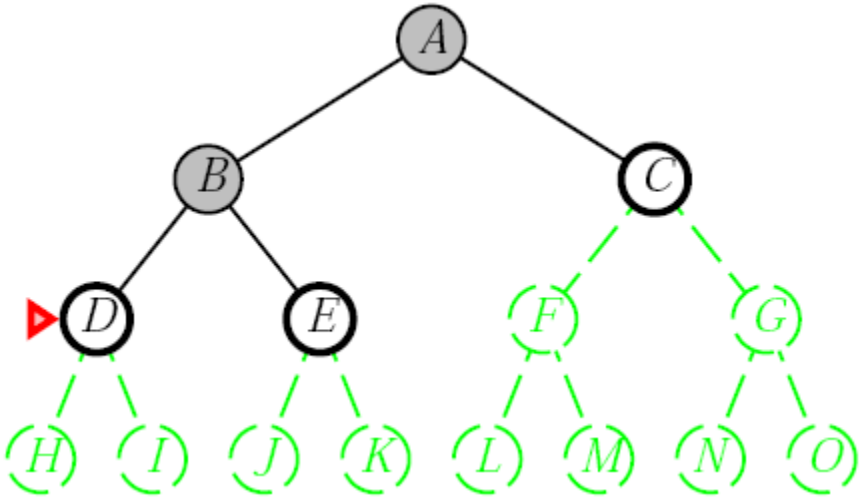
Depth-first Search Illustration

- Illustration of the depth-first search algorithm (strategy), where the investigated, expanded and closed nodes are highlighted.



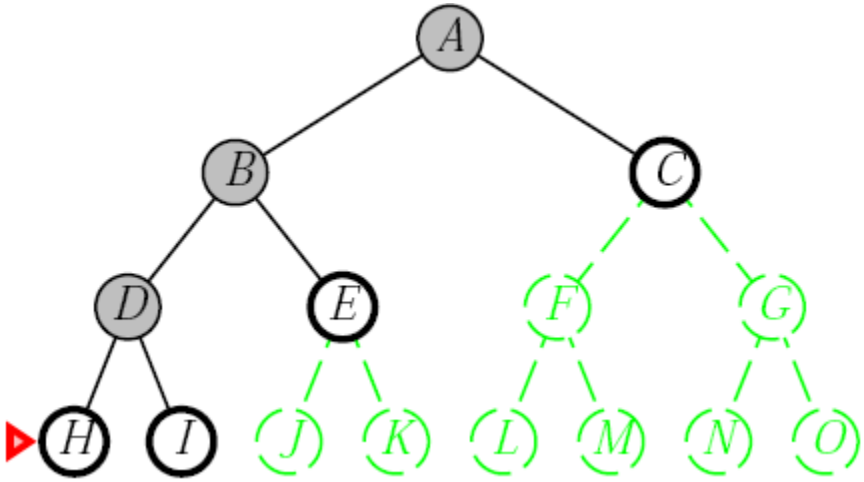
Depth-first Search Illustration

- Illustration of the depth-first search algorithm (strategy), where the investigated, expanded and closed nodes are highlighted.



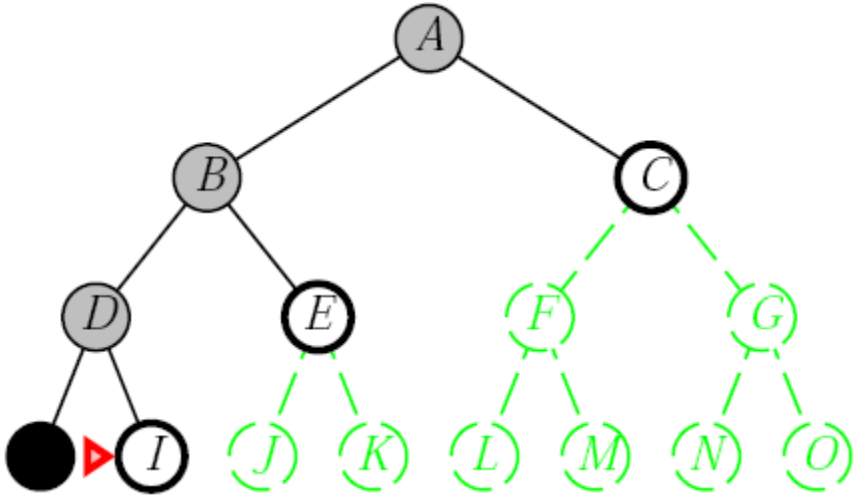
Depth-first Search Illustration

- Illustration of the depth-first search algorithm (strategy), where the investigated, expanded and closed nodes are highlighted.



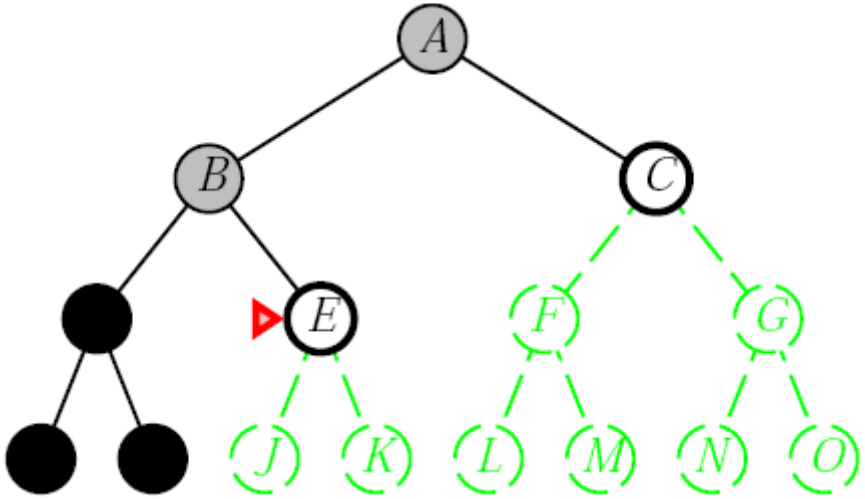
Depth-first Search Illustration

- Illustration of the depth-first search algorithm (strategy), where the investigated, expanded and closed nodes are highlighted.



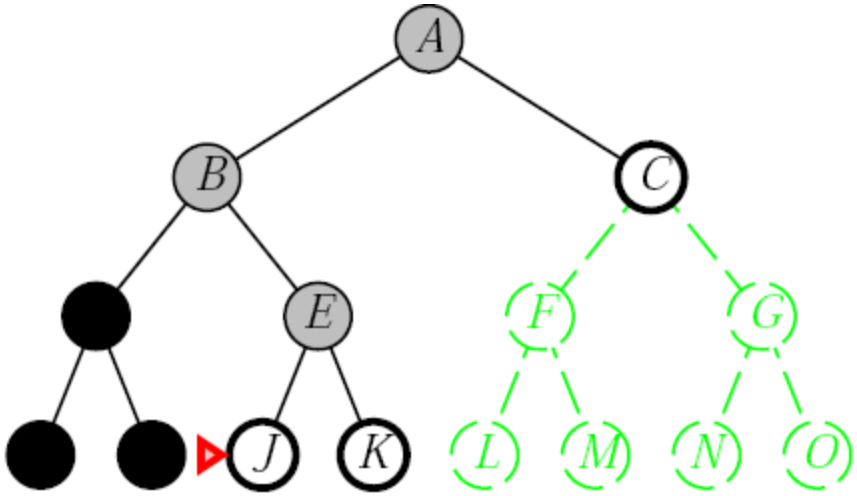
Depth-first Search Illustration

- Illustration of the depth-first search algorithm (strategy), where the investigated, expanded and closed nodes are highlighted.



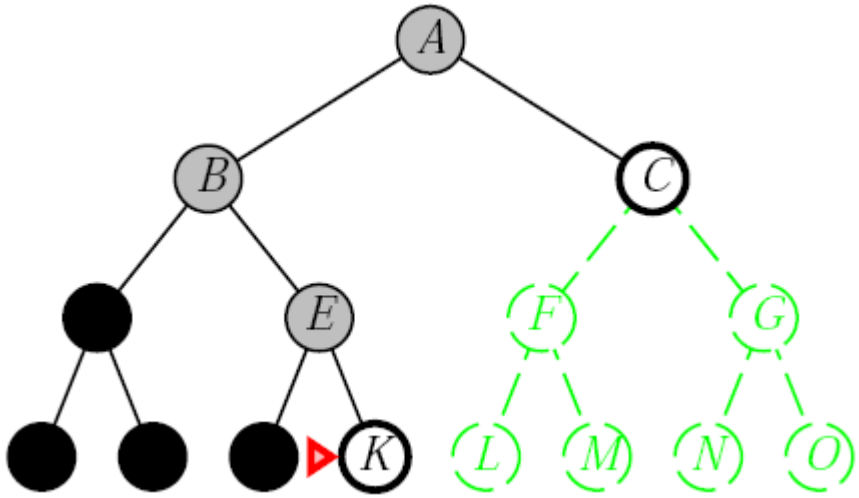
Depth-first Search Illustration

- Illustration of the depth-first search algorithm (strategy), where the investigated, expanded and closed nodes are highlighted.



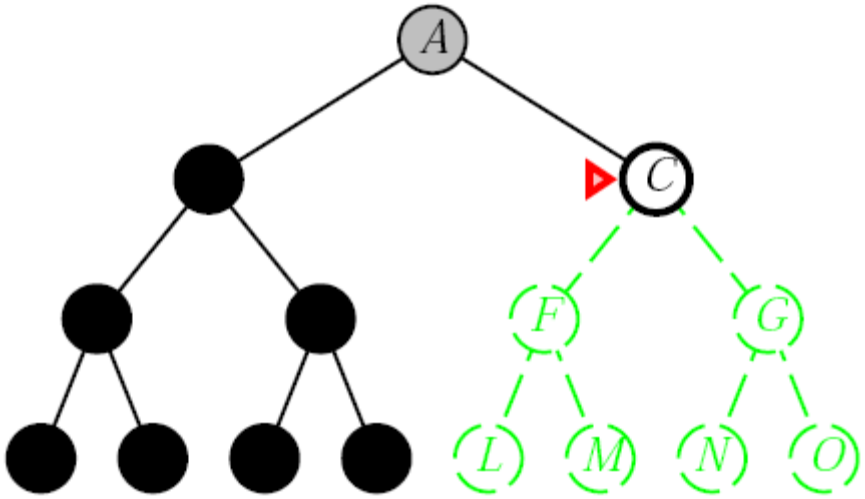
Depth-first Search Illustration

- Illustration of the depth-first search algorithm (strategy), where the investigated, expanded and closed nodes are highlighted.



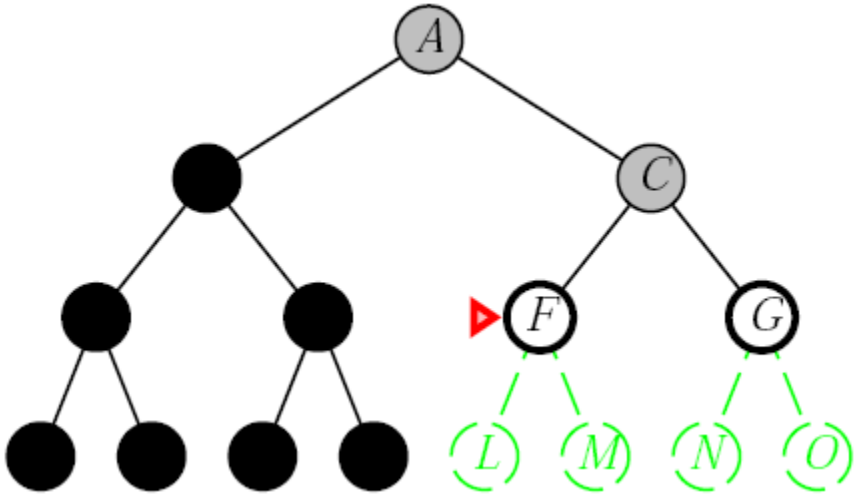
Depth-first Search Illustration

- Illustration of the depth-first search algorithm (strategy), where the investigated, expanded and closed nodes are highlighted.



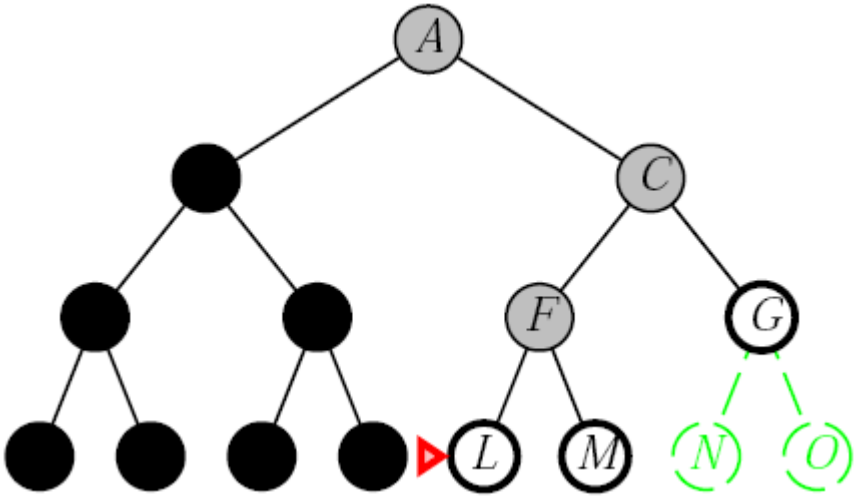
Depth-first Search Illustration

- Illustration of the depth-first search algorithm (strategy), where the investigated, expanded and closed nodes are highlighted.



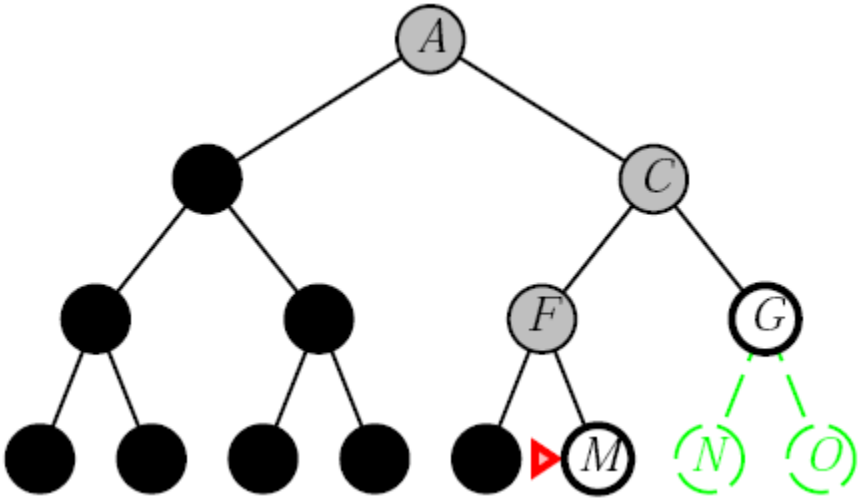
Depth-first Search Illustration

- Illustration of the depth-first search algorithm (strategy), where the investigated, expanded and closed nodes are highlighted.



Depth-first Search Illustration

- Illustration of the depth-first search algorithm (strategy), where the investigated, expanded and closed nodes are highlighted.



Characteristics of the Depth-first Search

- Let b denotes the maximum branching factor of the tree, d the depth of the final nodes with the lowest cost path and m the maximum depth of the state space (may be ∞)
- The strategy is complete, if m is finite.
- The time complexity of the algorithm is $O(b^m)$.
- The space complexity of the algorithm is $O(bm)$.
- The strategy is not optimal even if all the operation costs equal to 1.
- The biggest drawback of this strategy is the time complexity of the algorithm, if $b \gg d$.

Uniform-cost search

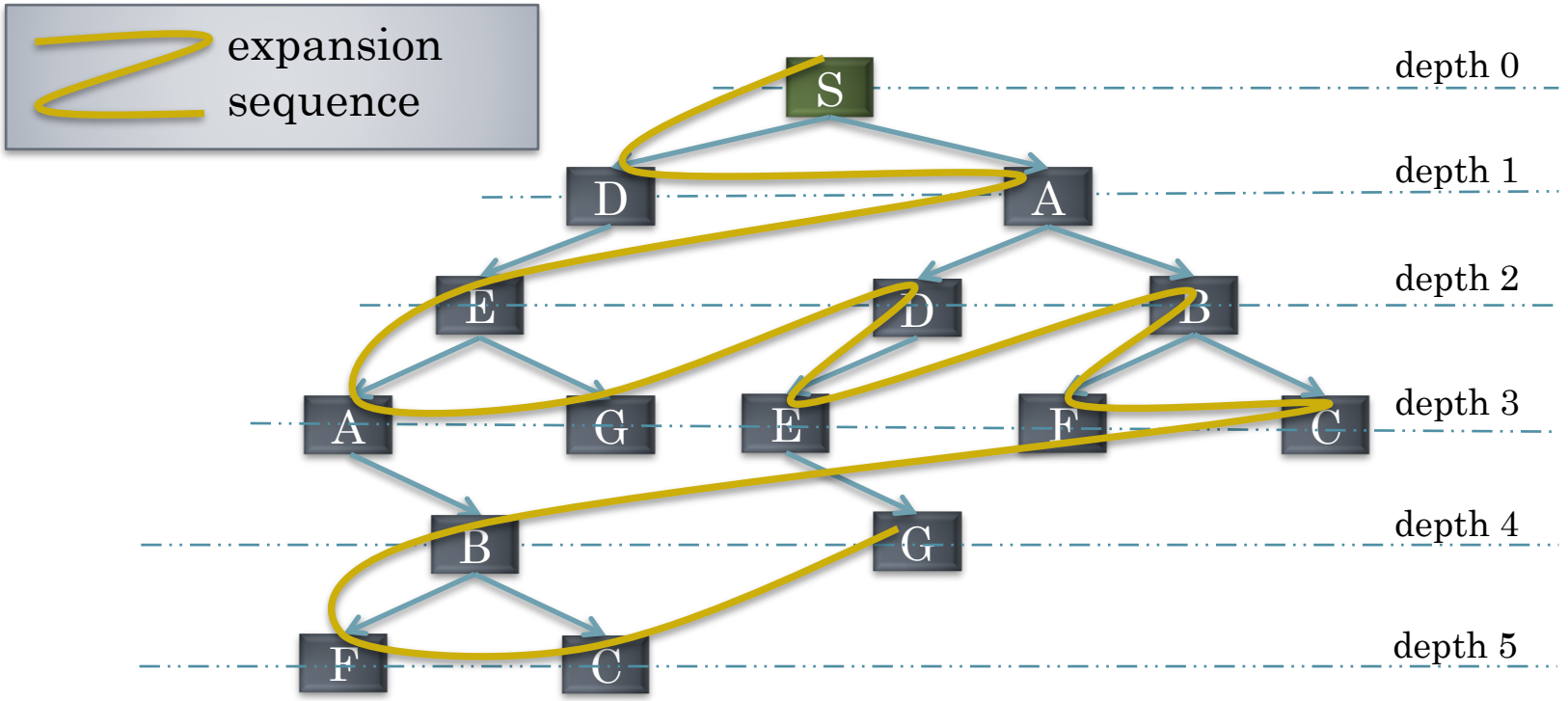
- The strategy is based on the principle – *„Expand the node with the lowest cost of the already made path!“*
- The queue of open nodes is sorted with regards to their path costs.
- In the case of constant operation costs, this strategy is equivalent to the breath-first search strategy.
- The strategy is complete and optimal.
- Time and space complexity of this strategy is proportional to the number of intermediate nodes with the path costs that are lower than the lowest cost of the path to the final node.

Depth-limited search

- Based on the depth-first search strategy by limiting the maximum depth of search.
- If the maximum investigated depth, denoted as l, is greater than the depth of the final node with the lowest path cost, i.e. $l \geq d$, then this strategy is complete.
- The time complexity of the algorithm is $O(b^l)$.
- The space complexity of the algorithm is $O(bl)$.
- The strategy is not optimal even if the costs of operations equal 1.
- The biggest disadvantage of this strategy is the problem of choosing a suitable depth l, as d is usually not known in advance.

Iterative deepening search

- Based on the iterative changing of the maximum depth l in the depth-limited search strategy.
- An illustration of an iterative deepening search with l = 1.



Characteristics of Iterative deepening search

- The strategy is complete.
- The time complexity of this search algorithm is $O(b^d)$.
- The space complexity of this search algorithm is $O(bd)$.
- The strategy is optimal if the costs of operation equal 1.
- The strategy can be adapted to the uniform-cost search.

Informed search strategies

- Based on the principle – *„Expand the node with the best prospect to be part of the path to the final node with the lowest cost!“*
- The strategy is called informed search or best-first search, and their main variants are greedy search and A* search.
- The greedy search strategy is based on expanding the nodes that appear to be closest to the final state.
- This strategy is based on the use of a heuristic function $h(n)$ that provides for the given state n the estimation of the cost of the path to the closest final node.
- This strategy is not complete.
- The time and space complexities are $O(b^m)$, and can be greatly improved with the good choice of $h(n)$.
- The strategy is not optimal even if the costs of operation equal 1.

A* SEARCH

- Based on the principle – *„Expand the node with the lowest partial cost of the already made path and the best prospect to reach the final node with the lowest cost!“*
- The strategy is based on using the combination of the cost $g(\mathrm{n})$ of the already made path to the given state n and the heuristic function $h(\mathrm{n})$ that provides for a given state n an estimate of the path cost to the closest final state.
- The queue of open nodes is then sorted based on the overall estimation of the cost $f(\mathrm{n}) = g(\mathrm{n}) + h(\mathrm{n})$ for the given state n.
- For the optimality of the algorithm, the heuristic function $h(\mathrm{n})$ has to satisfy a number of conditions.

The properties of the algorithm A*

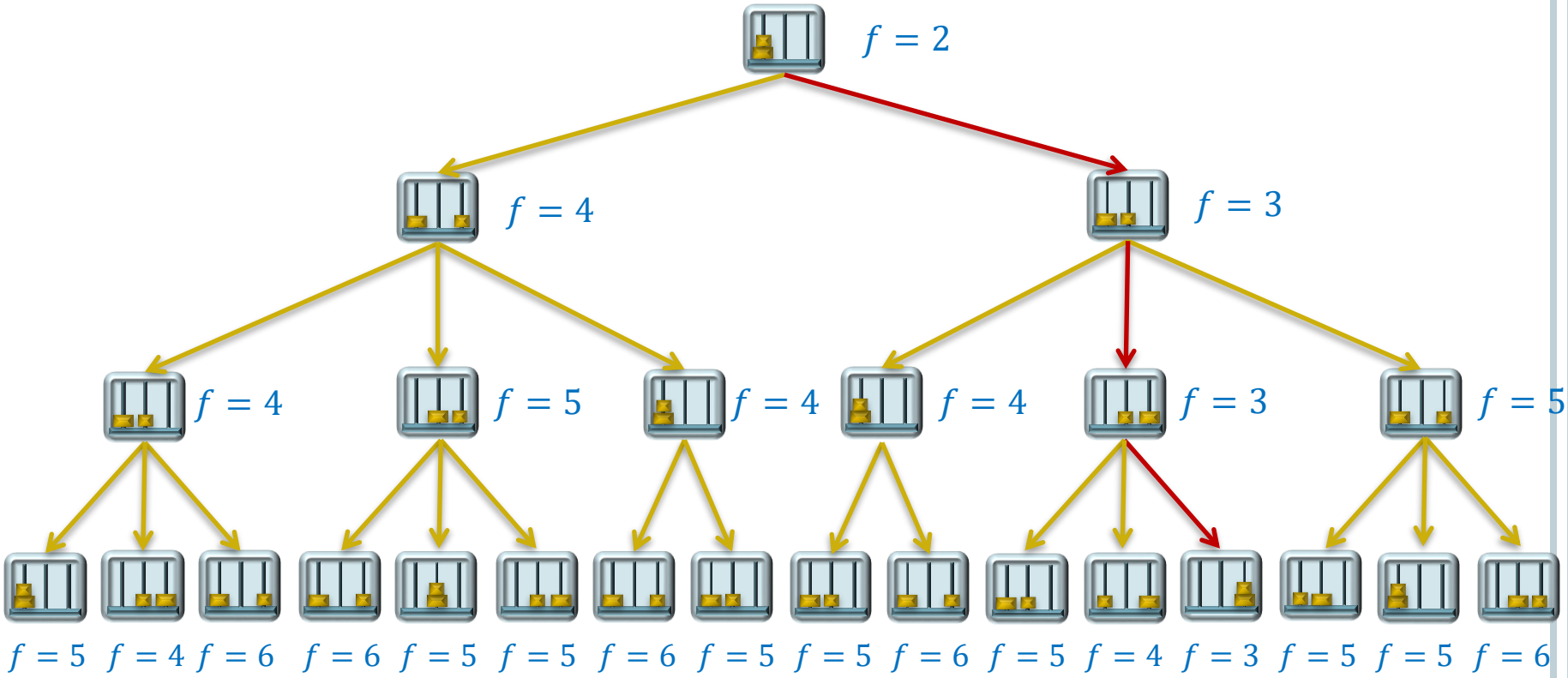
- The heuristic function $h(n)$ must return an estimation of the cost for the path from the given to the final state that is less than or equal to the actual cost of this path, i.e. $h(n) \leq h^*(n)$, where $h^*(n)$ is the actual cost.
- The heuristic function $h(n)$ must be non-negative, i.e. $h(n) \geq 0$, and its value for the final state must be zero, i.e. $h(G) = 0$, where G denotes the final state.
- The strategy is complete.
- The time and space complexities are $O(b^m)$, and can be greatly improved with the good choice of $h(n)$.
- The strategy is optimal if the heuristic function $h(n)$ satisfies the above conditions.

Examples of heuristic functions

- When searching for the optimal road route on the map, the heuristic function is normally the direct/air distance to the target location.
- In the Tower of Hanoi problem, the heuristic function is inversely proportional to the number and diameters of the discs on the third rod.
- In the n-sliding-tiles puzzle problem, the heuristic function is proportional to the number of tiles in the wrong location.
- In the assembling industrial product problem, the heuristic function is proportional to the number of missing components of the product.
- …

The Hanoi Tower example

- The values of the heuristic function f give the estimate of the total cost for the path from the initial to the final state in the tree.



Questions

- What usually characterizes a problem?
- What kinds of problems are considered in the field of AI?
- What are typical examples of problem solving?
- How a problem is formally defined?
- How problems are solved by search?
- What are examples of single-state problems?
- How a directed graph is converted into a tree?
- What search strategies are used to search problem trees?
- How search strategies are evaluated?

Solving Problems by Decomposing them into Sub-problems.

- In addition to using heuristic functions, problems can be solved using the knowledge about the structure of the problem.
- More difficult problems are intelligently solved by decomposing/dividing them into sub-problems.
- Sub-problems are always easier to solve than the entire problem.
- The solution of the entire problem is then achieved by combining the solutions of easier sub-problems into a problem solving plan.

Problem decomposition 1/2

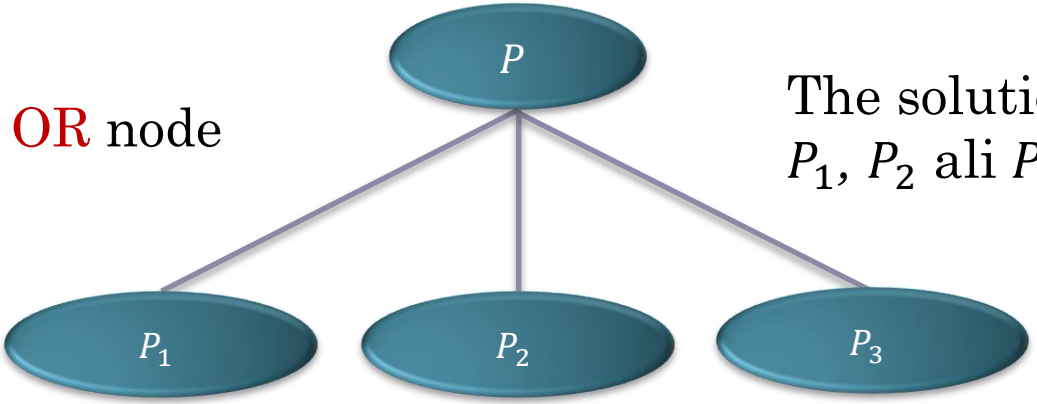
- If a given problem P_i cannot be solved with one operation it is divided/decomposed into sub-problems.
- Decomposition is continued until the sub-problems that can be solved by a single operation (so-called basic problems) are encountered.
- The decomposition of the problem into sub-problems is represented by an AND/OR tree.
- The leaves of the AND/OR trees are basic sub-problems that are trivial to solve.

Problem decomposition 2/2

- The decomposed problem is solved by solving its basic problems in accordance with the final problem solving plan.
- This solves the sub-problems in the intermediate nodes of the AND/OR tree all the way up to the root of the tree, which represents the solution of the entire problem.
- If an AND/OR tree contains only AND nodes then there is only one possible solution to the problem; otherwise, there are several.
- The solution of the problem is no longer a path, but each subtree of the AND/OR tree containing only the AND nodes.
- Sub-problems should normally be independent of one another (the order of solving them should not matter).

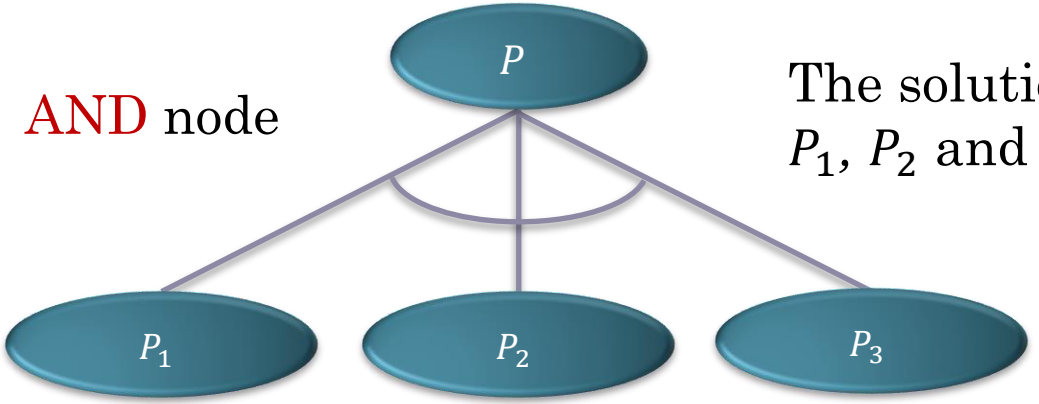
Nodes of the AND/OR Trees

OR node



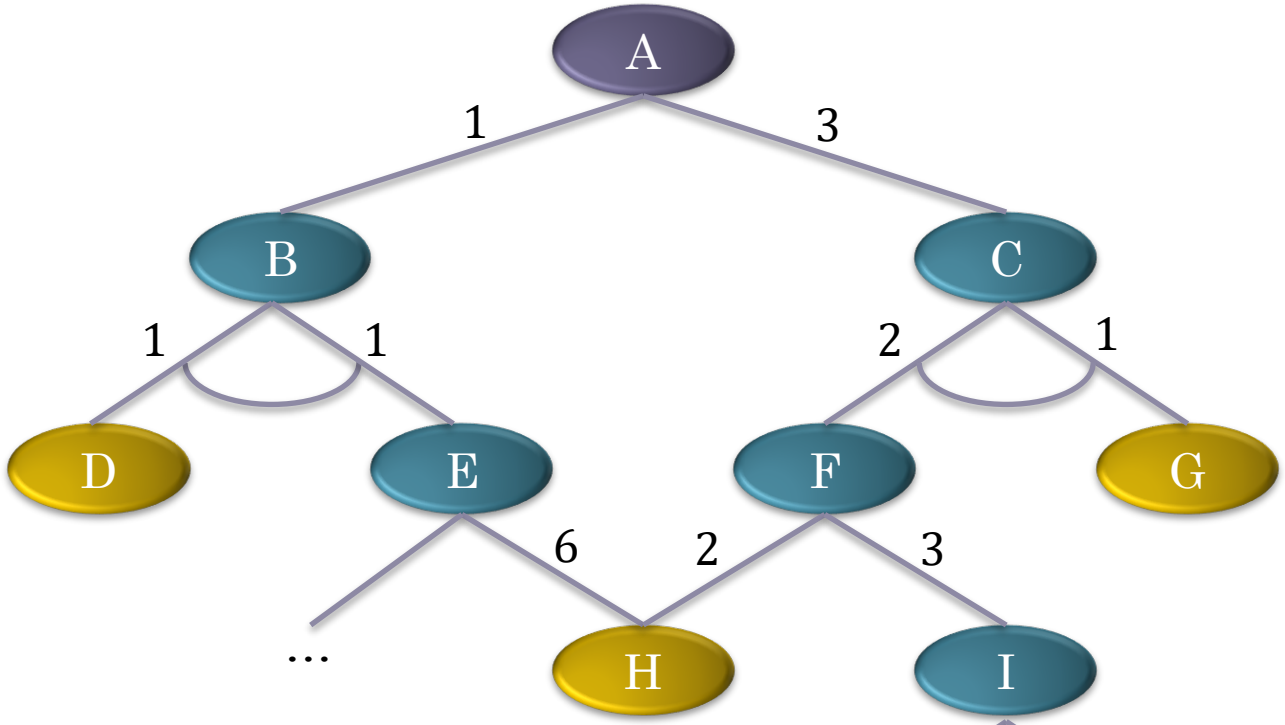
The solution to any sub-problem P_1, P_2 ali P_3 solves the problem P

AND node



The solution to all the sub-problems P_1, P_2 and P_3 solves the problem P

Assigning costs to relations between sub-problems

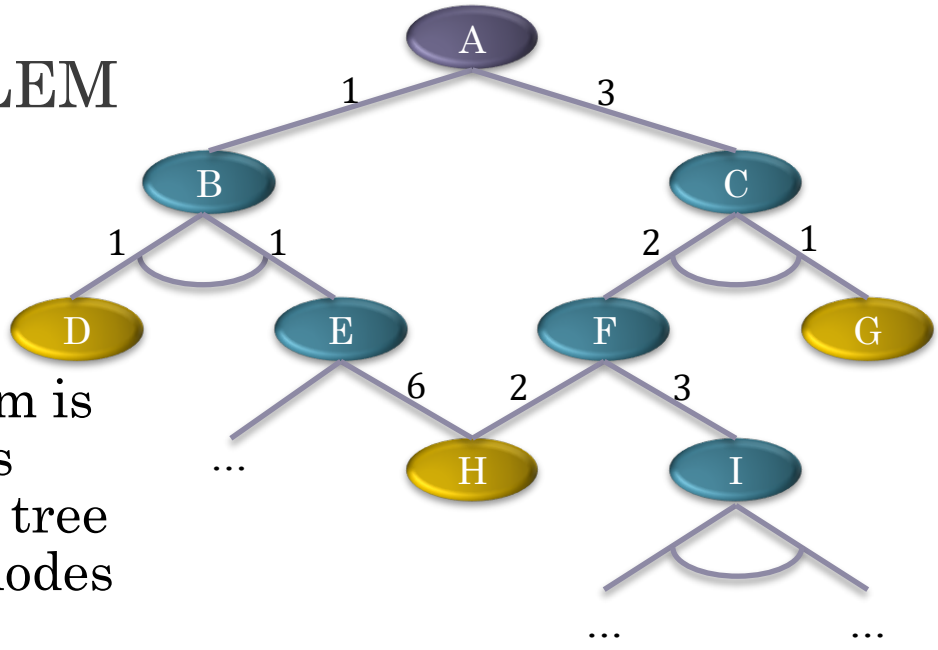


A — An entire problem

B — An intermediate sub-problem

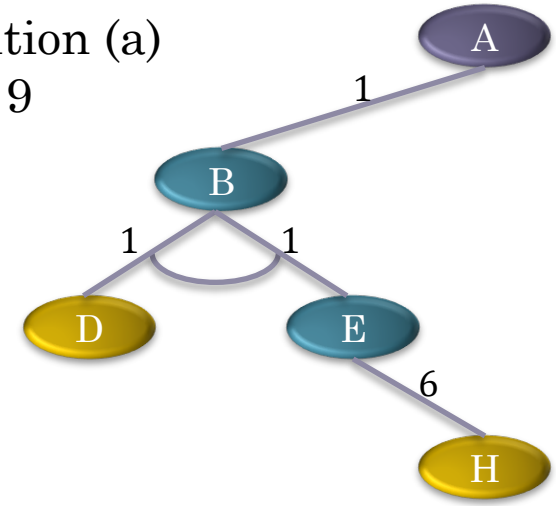
H — A trivial/basic sub-problem

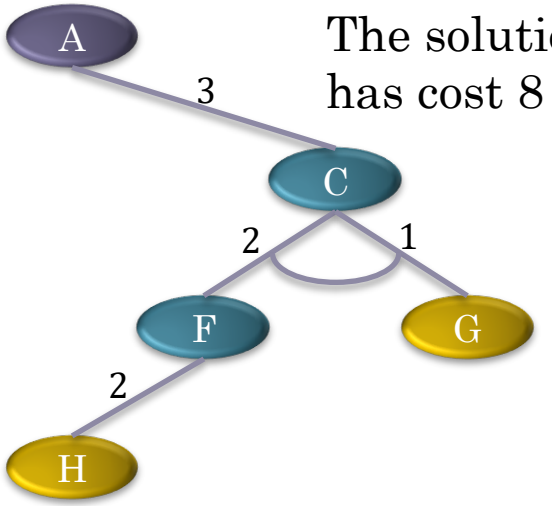
Solution is a Problem solving plan



- The final solution to a problem is a problem solving plan that is each sub-tree of the AND/OR tree that contains only the AND nodes and has leaves with only the trivial sub-problems.

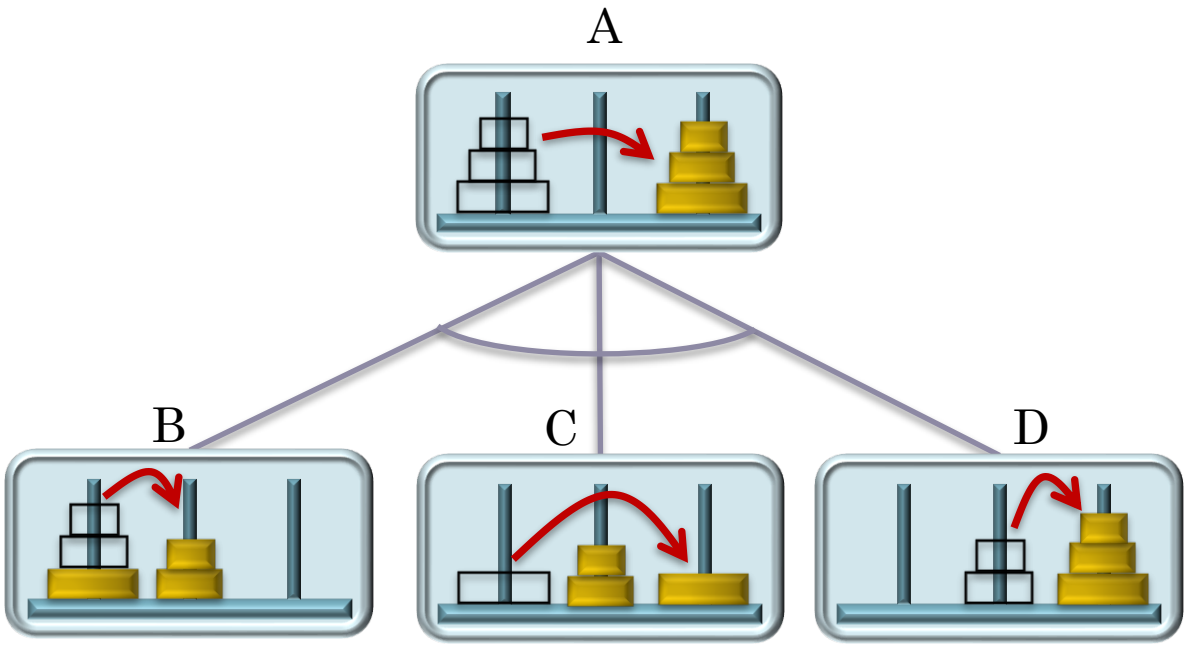
The solution (a) has cost 9





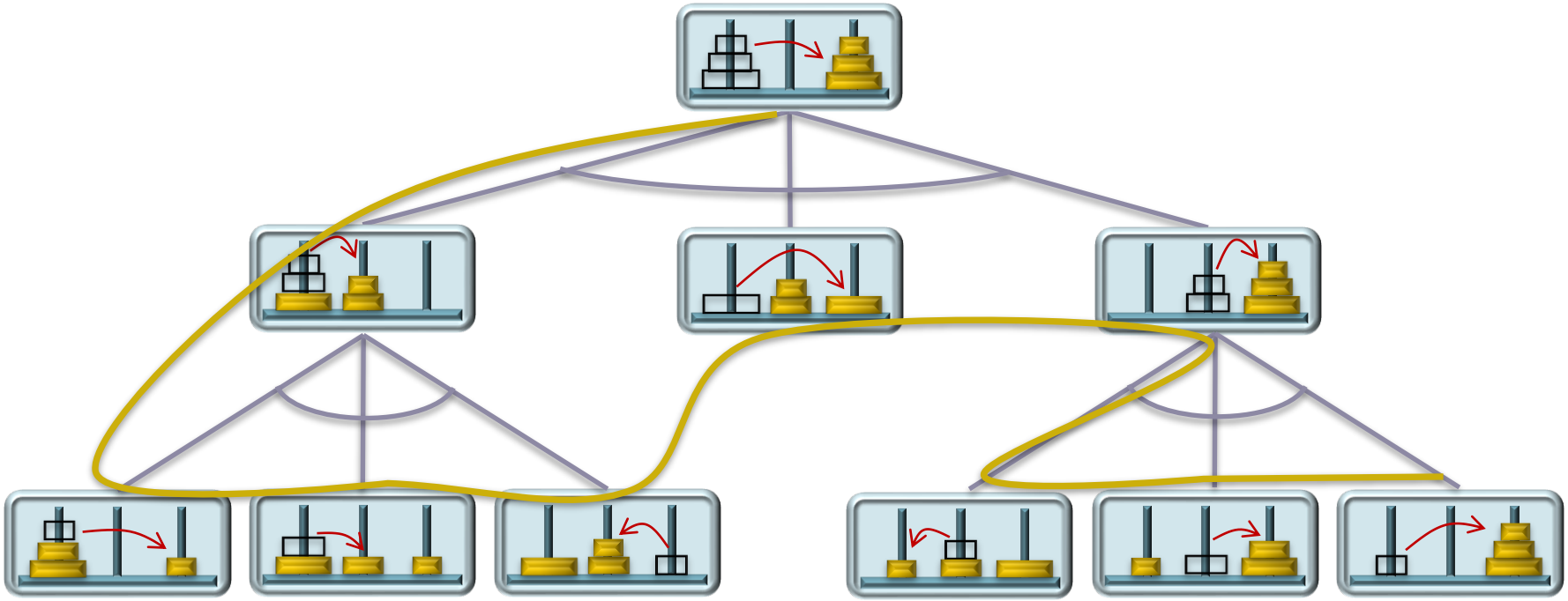
The solution (a) has cost 8

The Hanoi Tower Example 1/4



- The problem A is decomposed into three sub-problems B, C and D.
- The sub-problem is trivial, but the sub-problems A and D, has to be further decomposed into simpler sub-problems.
- The sub-problems B, C and D are not independent and should be solved in order from left to right.

The Hanoi Tower Example 2/4



- The expanded tree contains only AND nodes and therefore also represents one possible problem solving plan.
- Problem solving plan is carried out similarly to the depth-first strategy of searching trees.

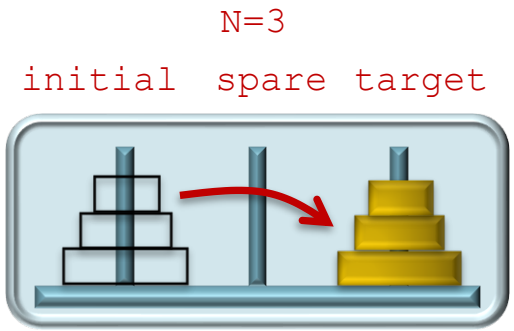
The Hanoi Tower Example 3/4

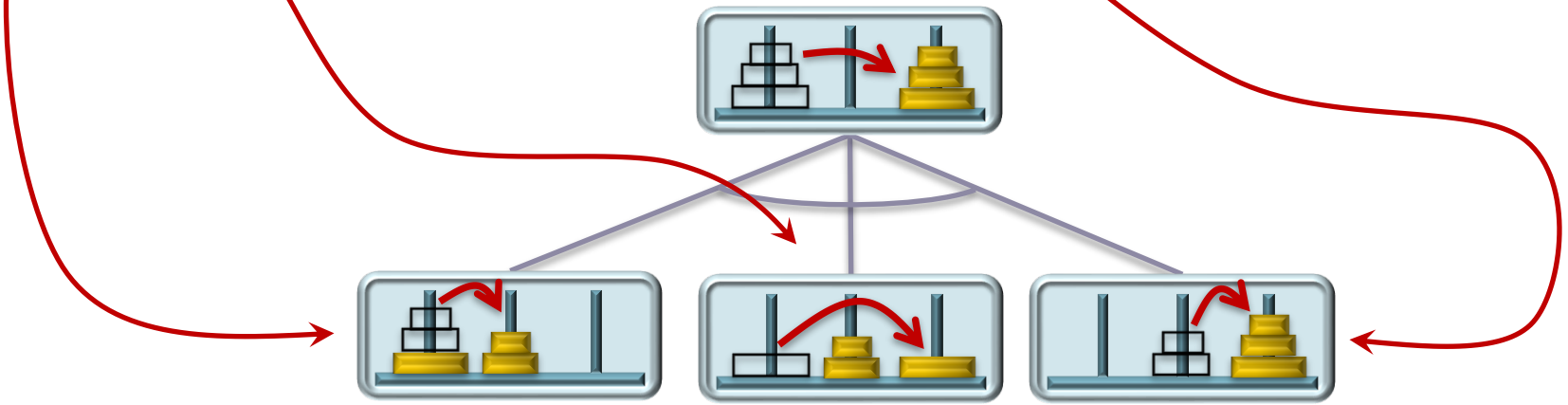
- In the given example, the decomposition of a given (intermediate) problem to its three more simple sub-problems is always carried out in the same way that can be described as follows:
  - Move the (n-1)-disk tower from the given initial rod to the given spare rod.
  - Move the last n-th largest disk from the given initial rod to the given target rod (trivial / basic sub-problem).
  - Move the (n-1)-disk tower from the given spare rod to the given target rod (which now has the largest disc).
- If on the given initial rod is only one disk then just move it to the target rod (trivial / basic sub-problem).

The Hanoi Tower Example 4/4

- The described approach can be implemented by using the recursive function **`solve`**, that decompose a given problem to the three mentioned sub-problems.

```
function solve(N, initial, spare, target)
   if N = 0
      return
   else
      solve(N-1,initial, target, spare)
      move_disk(initial, target)
      solve(N-1,spare, initial,target)
   endif
end
```





Searching for the optimal problem solving plan 1/2

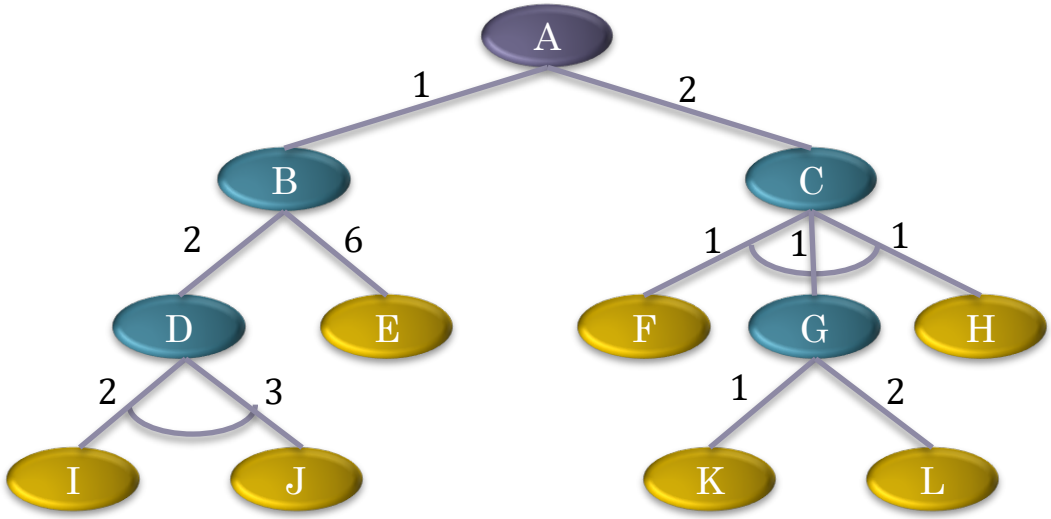
- In the case when the given problem decomposition is represented by a tree that contains also OR nodes, then there are several solutions to the problem (problem solving plans).
- The optimal problem solving plan (with the lowest cost) can be found by searching the AND/OR tree using similar search strategies that are used for searching problem state space trees.
- Search begins at the root of the tree, which represents the entire problem, and continues to the leaves representing the trivial problems.
- During the search, all the successors of the expanded AND nodes and at least one successor of the expanded OR nodes are further expanded.

Searching for the optimal problem solving plan 2/2

- A given node is considered to be solved, if:
  - The node is the OR node and at least one of its successors is solved;
  - The node is the AND node and all its successors are solved;
  - The node is the leaf node and the corresponding trivial problem is solved.
- In the OR nodes, it is important, which successor is selected to be expanded.
- When all the expanded nodes are solved, then the entire problem is solved and one of the possible problem solving plans is obtained.
- In the search, costs of solving nodes should be included, on the basis of which we can estimate which problem solving plan is better or worse.

An example search for an optimal problem solving plan 1/2

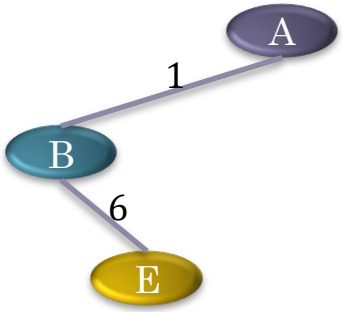
- Let us assume that the decomposition of a problem is represented by the below given AND/OR tree.



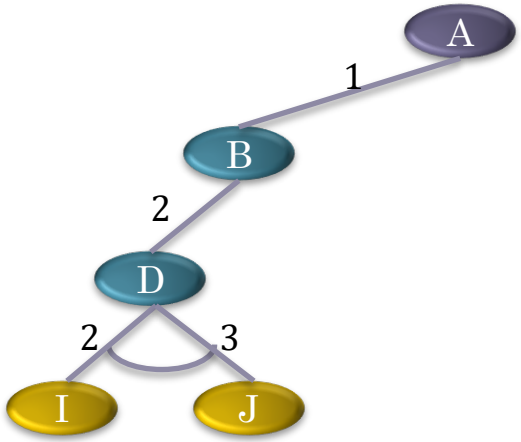
- It may be noted that there are four problem solving plans to solve the entire problem A.

An example search for an optimal problem solving plan 2/2

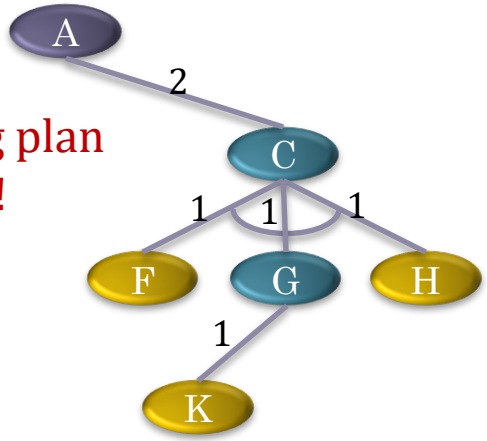
1)



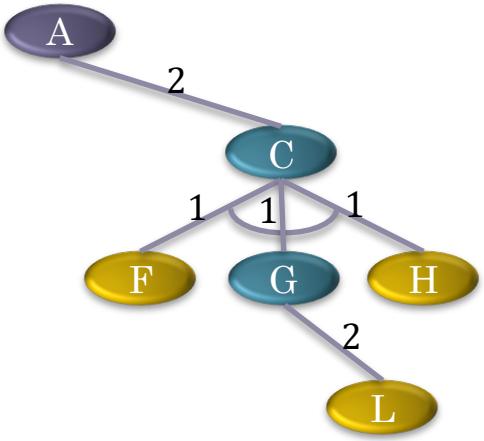
2)



3)



4)



The problem solving plan with the lowest cost!

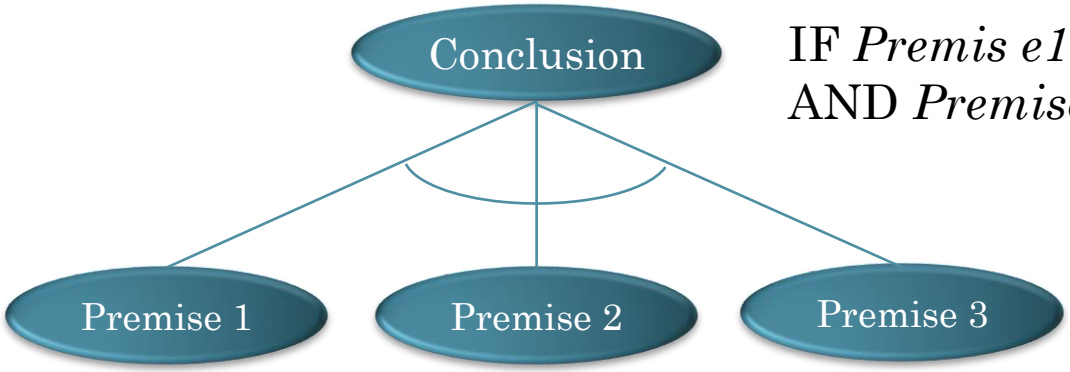
Search Strategy for AND/OR trees

- The strategy is based on the principle - *„Expand the node with the lowest total estimated cost of solving the problem“.*
- The AO* search algorithm is usually used for searching such trees (the AO* algorithm is a generalization of the A* algorithm - from only OR trees to AND/OR trees).
- Let $\tilde{h}\,(P)$ denote the heuristic estimation of the actual cost $h(P)$ for solving the problem as seen from the given node P.
- For the OR nodes, $\tilde{h}\,(P) = \min_i \{c(P, P_i) + \tilde{h}(P_i)\}$, where $c(P, P_i)$ denote the cost for expanding the successor node P_i.
- For the AND nodes, $\tilde{h}\,(P) = \sum_i c(P, P_i) + \tilde{h}(P_i)$.
- For the leaf nodes $\tilde{h}\,(P)$ equals the cost for solving the predecessor node's problem.

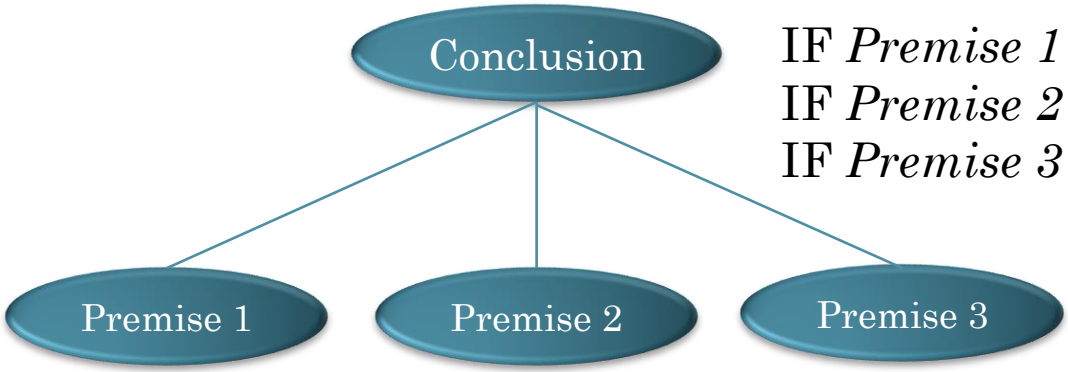
Wider uses of AND/OR trees

- The trees that represent problem-state spaces may be seen as the trees with only the OR nodes.
- When a tree contains only the OR nodes then the solution is a path from the root node to one of the leaf nodes.
- The AND/OR trees are thus more general and are used for:
  - Solving problems by decomposing them into sub-problems;
  - Representing knowledge that is based on production (IF … THEN) rules;
  - Reasoning based on the given knowledge and from the given input facts (forward and backward rules chaining).
  - …

Presentation of the production rules with AND/OR trees



IF *Premis e1* AND *Premise 2*
AND *Premise 3* THEN *Conclusion*



IF *Premise 1* THEN *Conclusion*
IF *Premise 2* THEN *Conclusion*
IF *Premise 3* THEN *Conclusion*

Questions

- In what ways the decomposition of a problem to sub-problems is usually presented?
- What do represent the nodes in a AND/OR tree?
- What constitutes a solution to the problem that is decomposed to sub-problems?
- How many solutions are contained in the tree with only the AND nodes?
- Describe how the Hanoi Tower problem can be solved by decomposing it into sub-problems.
- How production rules can be represented by an AND/OR tree?