Artificial Intelligence
Problem Solving Methods, Heuristic Search Techniques and Analysis

05.01.2016

S. Kavitha

AP/CSE

SSN College of Engineering

Overview

- Problem introduction & components
- How to solve AI problems?
- Search techniques
  - Uninformed search techniques (Blind)
  - Informed search techniques (Heuristic)
- Performance analysis of search algorithms

Problem – Definition and sequence of steps in solving

- Collection of information that the agent will use to decide what to do
- Problem solving agent
- Define and analyze the problem
- Sequence of steps in problem solving
  - Goal formulation
  - Problem formulation
  - Search
  - Solution
  - Execution phase

Components of Problem description

- **Initial state**
- **Operator or successor function** - for any state x returns s(x), the set of states reachable from x with one action
- **State space** - all states reachable from initial by any sequence of actions

Components of Problem description

- **Path** - sequence through state space
- **Path cost** - function that assigns a cost to a path. Cost of a path is the sum of costs of individual actions along the path
- **Goal test** - test to determine if at goal state

Components of Problem description-Example-Eight puzzle

- States:
  - Description of the eight tiles and location of the blank tile
- Successor Function:
  - Generates the legal states from trying the four actions {*Left, Right, Up, Down*}
- Goal Test:
  - Checks whether the state matches the goal configuration
- Path Cost:
  - Each step costs 1

7	2	4
5		6
8	3	1

1	2	3
4	5	6
7	8	

Components of Problem description- Example-Eight puzzle

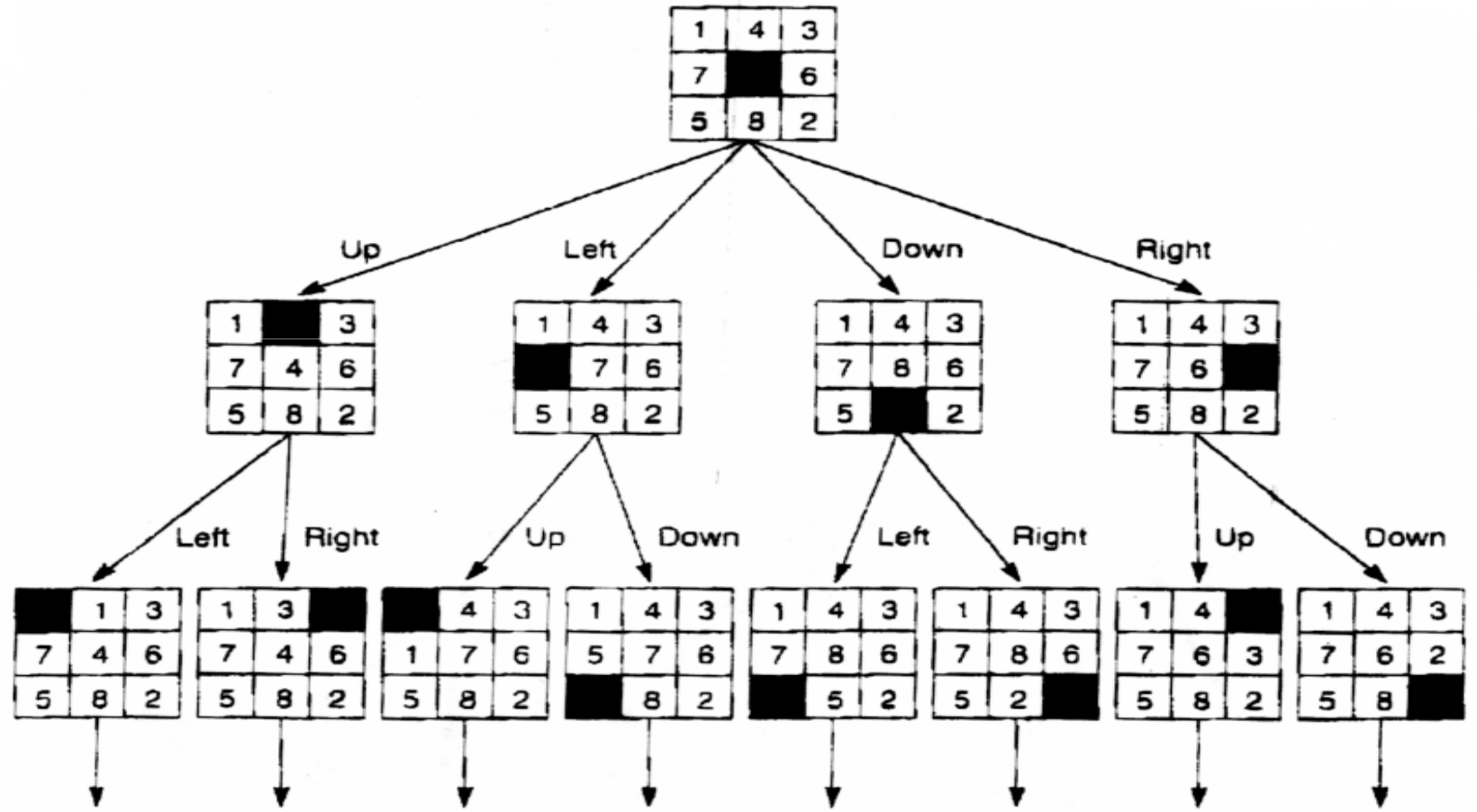


Figure 3.6 State space of the 8-puzzle generated by "move blank" operations.

Example Problems – contd..

- Toy problems
  - Vacuum world problem
  - 8-queens problem
  - Water-jug problem
  - Monkey-Banana problem
  - Missionaries and Cannibals problem
- Real world problems
  - Route finding – TSP, Robot navigation, Computer networks
  - Assembling – Electric motors, Robot arm assembly

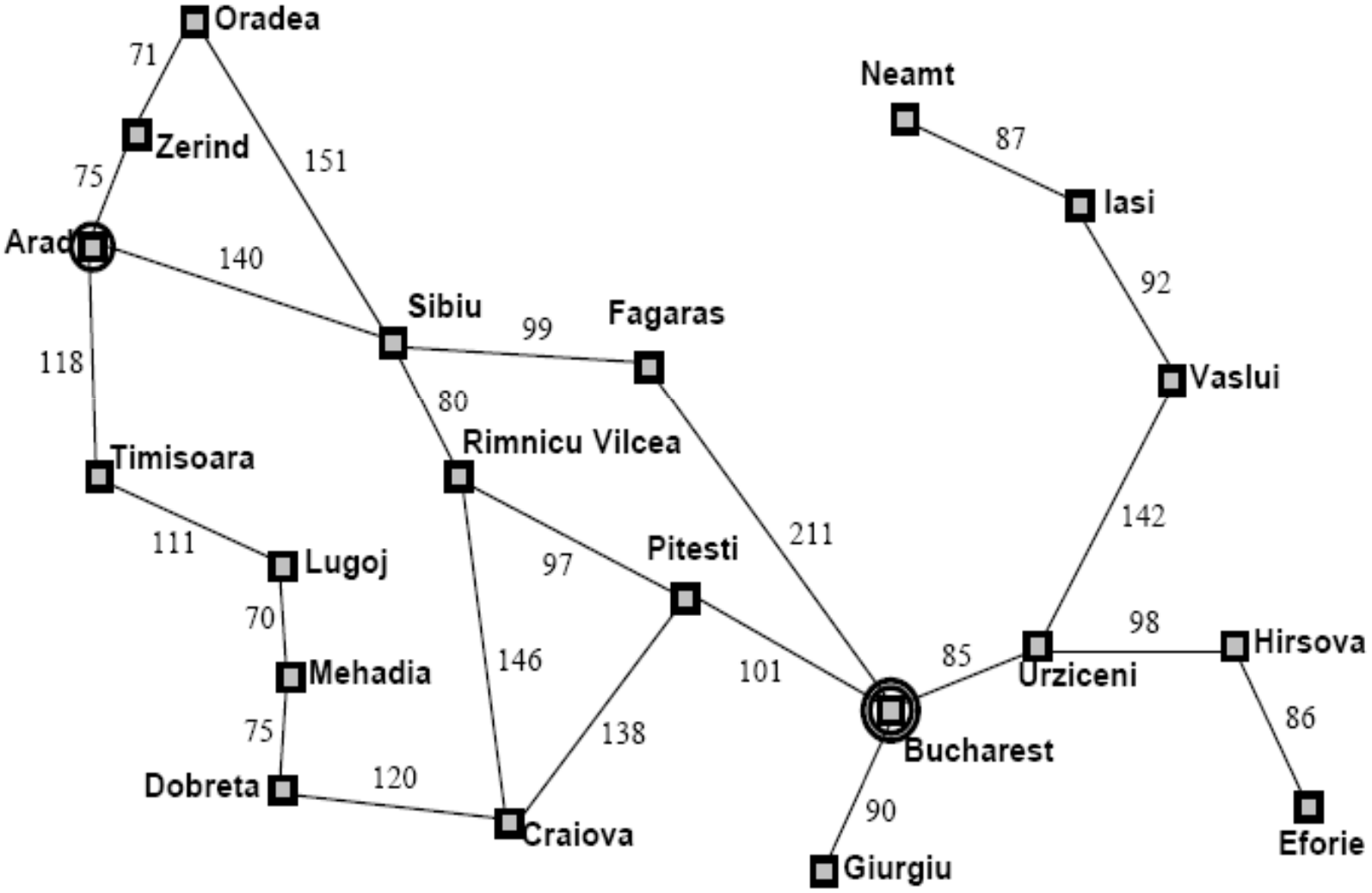
Problem Solving methods - Search

- Appropriate search technique is selected to solve the problem using sequence of actions
- Search process
- Tree based search – repeated states
- Graph based search – repeated states are avoided – cycle may exist

Tree vs Graph based Search

- Basic idea:
  - Exploration of state space by generating successors of already-explored states (a.k.a. expanding states).
  - Every state is evaluated: *is it a goal state*?
- In practice, the solution space can be a graph, not a tree - E.g., 8-puzzle
  - More general approach is graph search
  - Tree search can end up repeatedly visiting the same nodes
    - Unless it keeps track of all nodes visited
    - …but this could take vast amounts of memory

Tree search example- Traveling in Romania

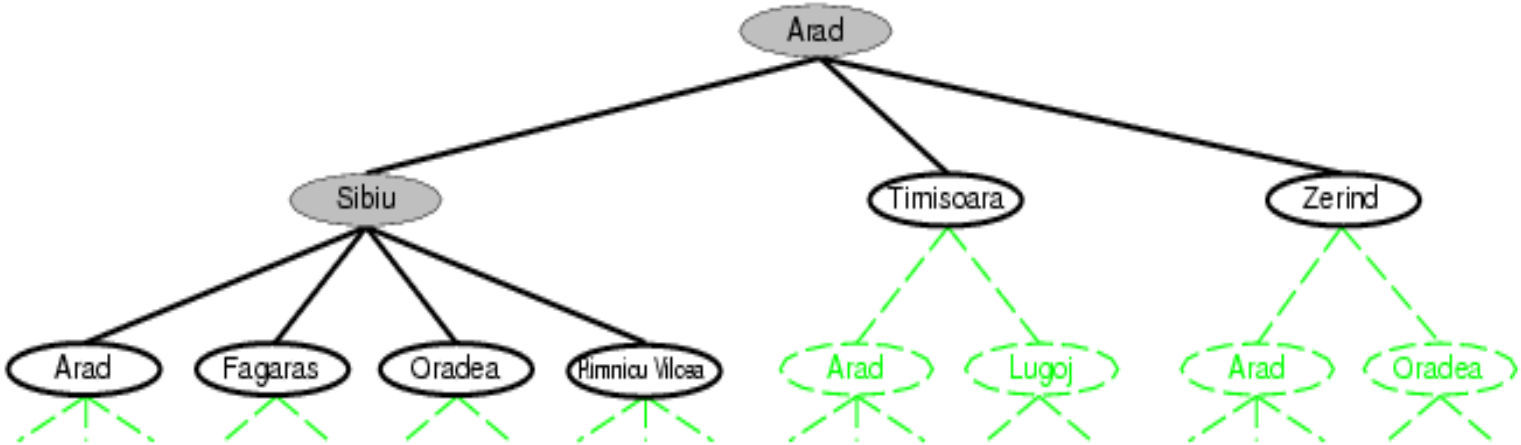


Tree search example- Traveling in Romania

Straight - line distance to B from:

- A - 366
- B - 0
- F - 178
- P - 98
- R - 193
- S - 253
- T - 329
- Z - 374

Tree search example



function TREE-SEARCH(*problem, strategy*) **returns** a solution, or failure

initialize the search tree using the initial state of *problem*

loop do

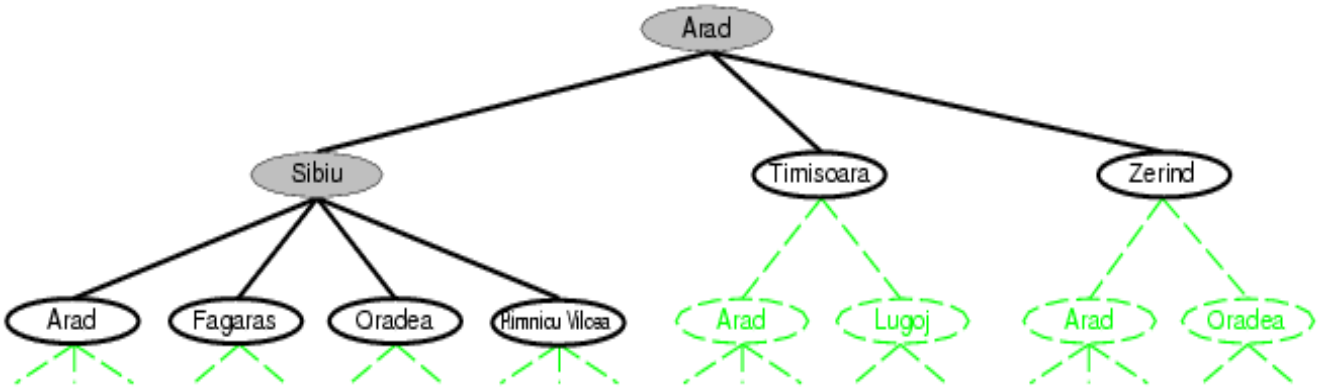
if there are no candidates for expansion **then return** failure

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

Tree search example



function TREE-SEARCH(*problem, strategy*) **returns** a solut

initialize the search tree using the initial state of *problem*

loop do

if there are no candidates for expansion **then return** failu

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

This “strategy” is what differentiates different search algorithms

Graph search - Check Duplicate Nodes

- Examine the set of nodes that have been created so far to see if the new node already exists.
- If it does not-simply add it to the graph just as for a tree.
- If it does already exist, then do the following:
  - Set the node that is being expanded to point to the already existing node corresponding to its successor rather than to the new one. The new one can simply be thrown away.
  - If you are keeping track of the best (shortest or otherwise least-cost) path to each node, then check to see if the new path is better or worse than the old one. If worse, do nothing. If better, record the new path as the correct path to use to get to the node and propagate the corresponding change in cost down through successor nodes as necessary.

Search strategies

- Uninformed or blind search strategies uses only the information available in the problem definition
- Informed or heuristic search strategies uses additional information

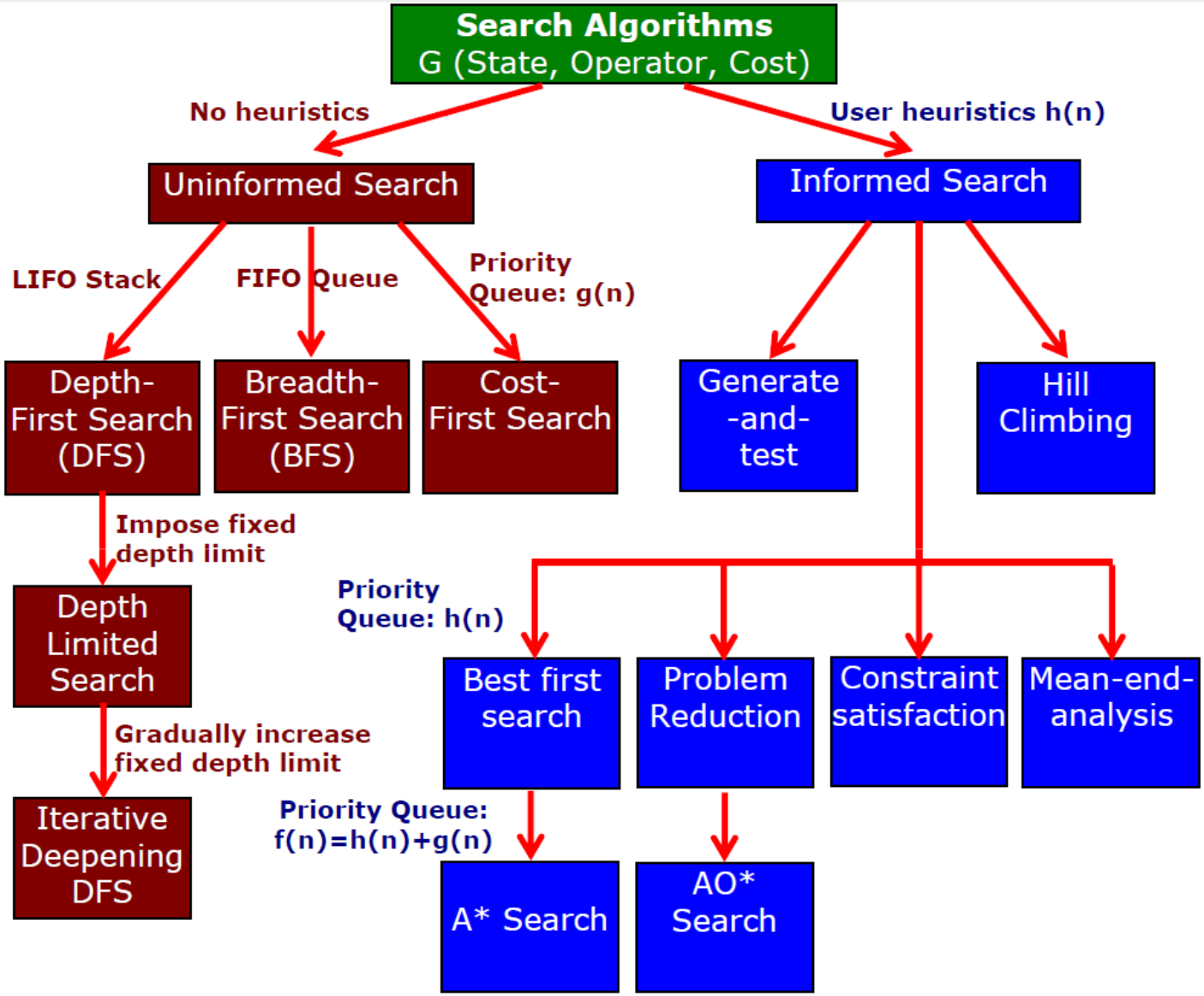


Fig. Different Search Algorithms

Search strategy - Algorithm's performance

- A search strategy is defined by picking the order of node expansion
- Strategy algorithms are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?

Generate and test

- simplest form of all heuristic search methods

Algorithm

1. Generate a possible solution. For some problems, this means generating a particular point in the problem space. For others, it means generating a path from a start state.
2. Test to see if this is actually a solution by comparing the chosen point or the endpoint of the chosen path to the set of acceptable goal states.
3. If a solution has been found, quit. Otherwise, return to step 1.

Best-first-search

- Nodes are ordered and expanded using evaluation function
- Best evaluation function is expanded first
- Two types of evaluation function
  - Greedy
  - A* search

Best-first-search

function BEST-FIRST-SEARCH(*problem, EVAL-FN*) **returns** a solution sequence

inputs: *problem*, a problem

EVAL-FN, an evaluation function

QUEUEING-FN←a function that orders nodes by *EVAL-FN*

return TREE-SEARCH(*problem*, QUEUEING-FN)

Greedy Best First Search

Evaluation function

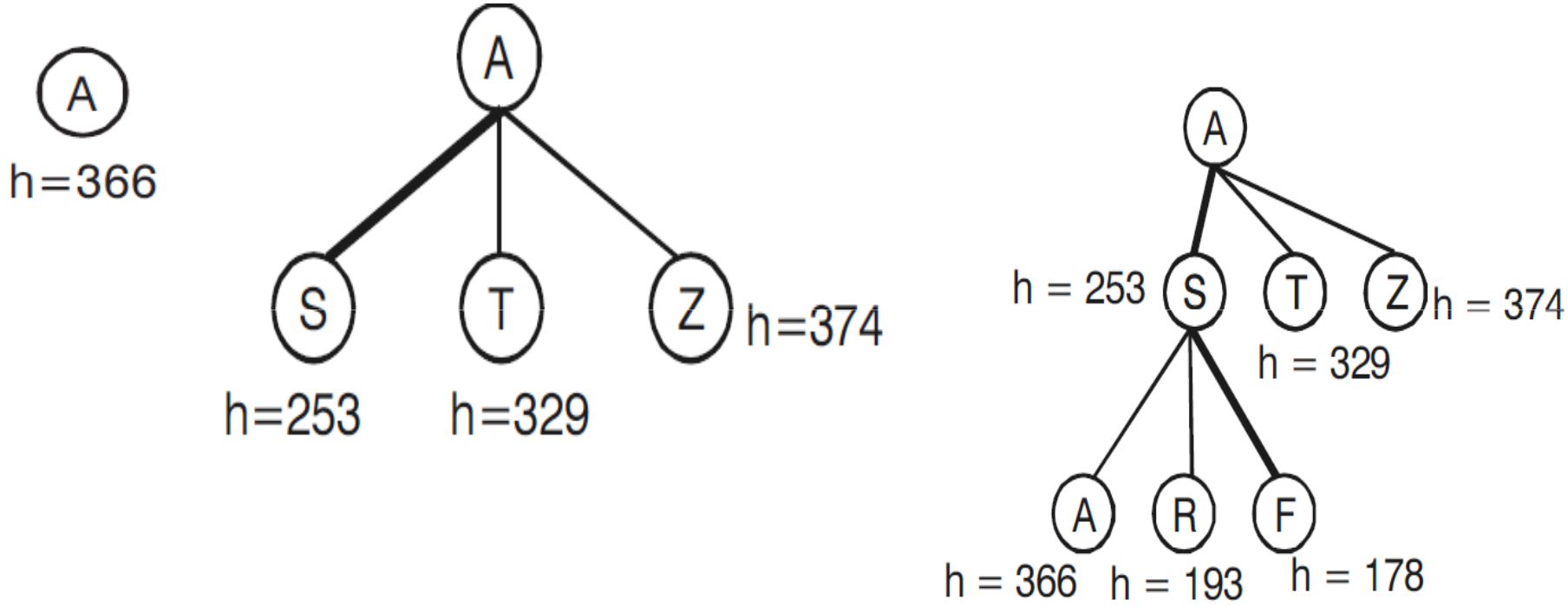
- *h(n) = estimated cost of the cheapest path from the state at node n to a goal state*

Algorithm

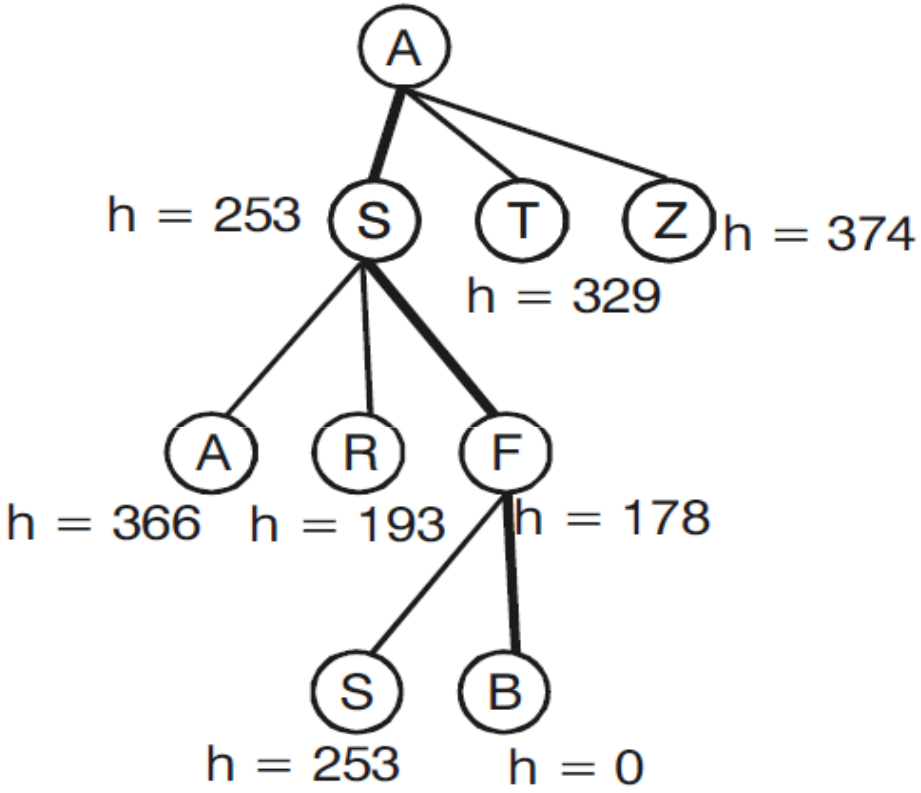
function GREEDY-BEST-FIRST SEARCH(*problem)*
returns a solution or failure

return BEST-FIRST-SEARCH (*problem, h)*

Greedy Best First Search-Example



Greedy Best First Search-Example



A - S - F - B = 450 (i.e) (140 + 99 + 211)

Greedy Best First Search-Performance Analysis

- Time and space complexity – $O(b^m)$
- Optimality – no
- Completeness - no

A* search

Evaluation function

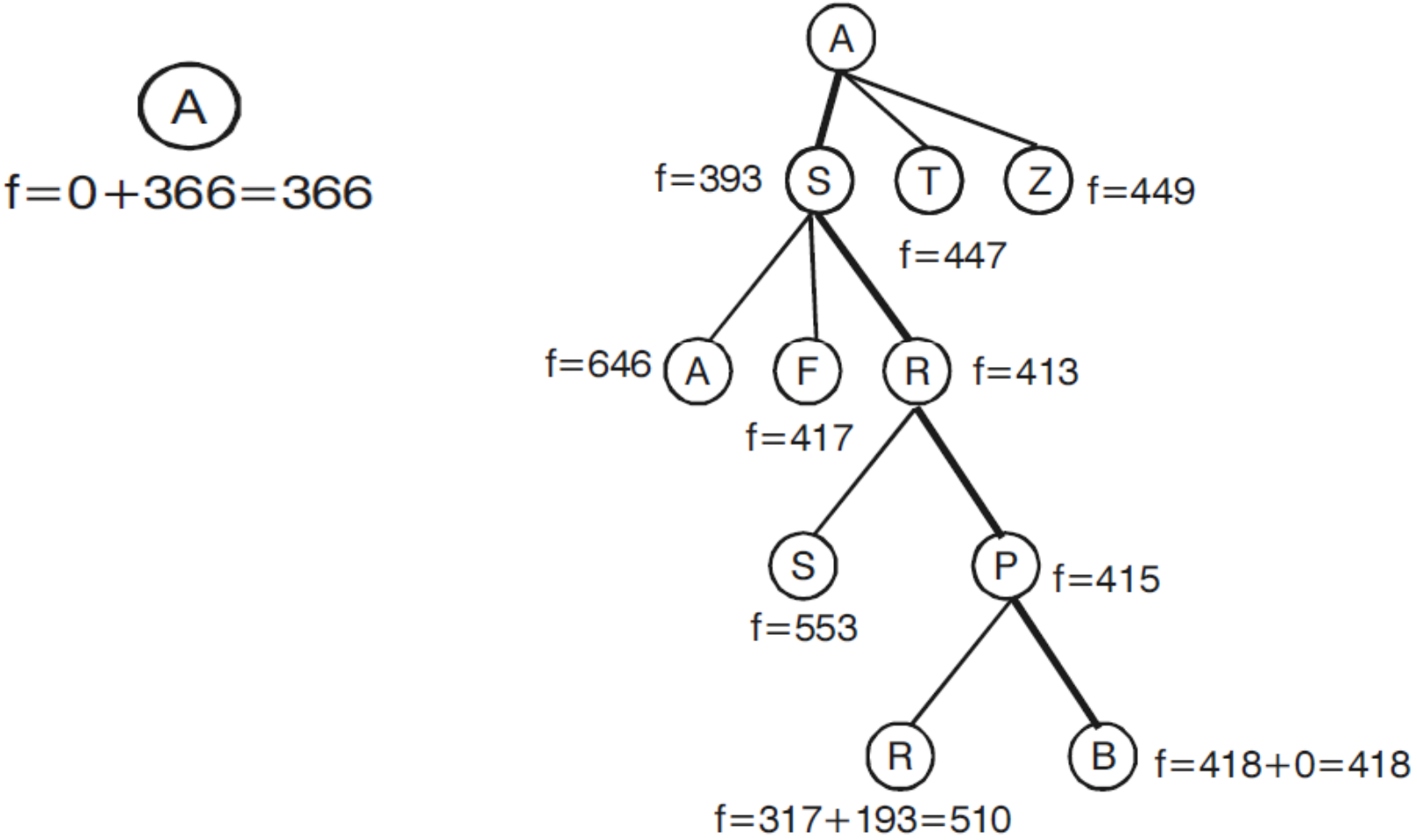
- *f(n) = h(n)+g(n)*
- *f(n) = cost of the cheapest solution through n*
- *g(n) = actual path cost from the start node to node n*

Algorithm

function A* SEARCH(*problem) returns a solution or failure*

return BEST-FIRST-SEARCH (*problem, g+h)*

A* search - Example



A* search -Performance Analysis

- Time complexity – depends on heuristic function and admissible heuristic value
- space complexity – $O(b^m)$
- Optimality – yes (locally finite graphs)
- Completeness – yes (locally finite graphs)

Constraint Satisfaction Problem (CSP) - search

- Search procedure that operates in a space of constraint sets.
- Initial state - contains the constraints that are originally given in the problem description.
- Goal State - any state that has been constrained "enough," where "enough" must be defined for each problem
- Example: Crypt arithmetic, graph coloring

CSP – Example - Crypt arithmetic

Statement : the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, each letter stand for a different digit.

Example:

- Given : FORTY+ TEN+ TEN=SIXTY
- 29786 + 850+ 850 = 31486
- F=2, O=9, R=7, T=8, Y=6, E=5, N=0

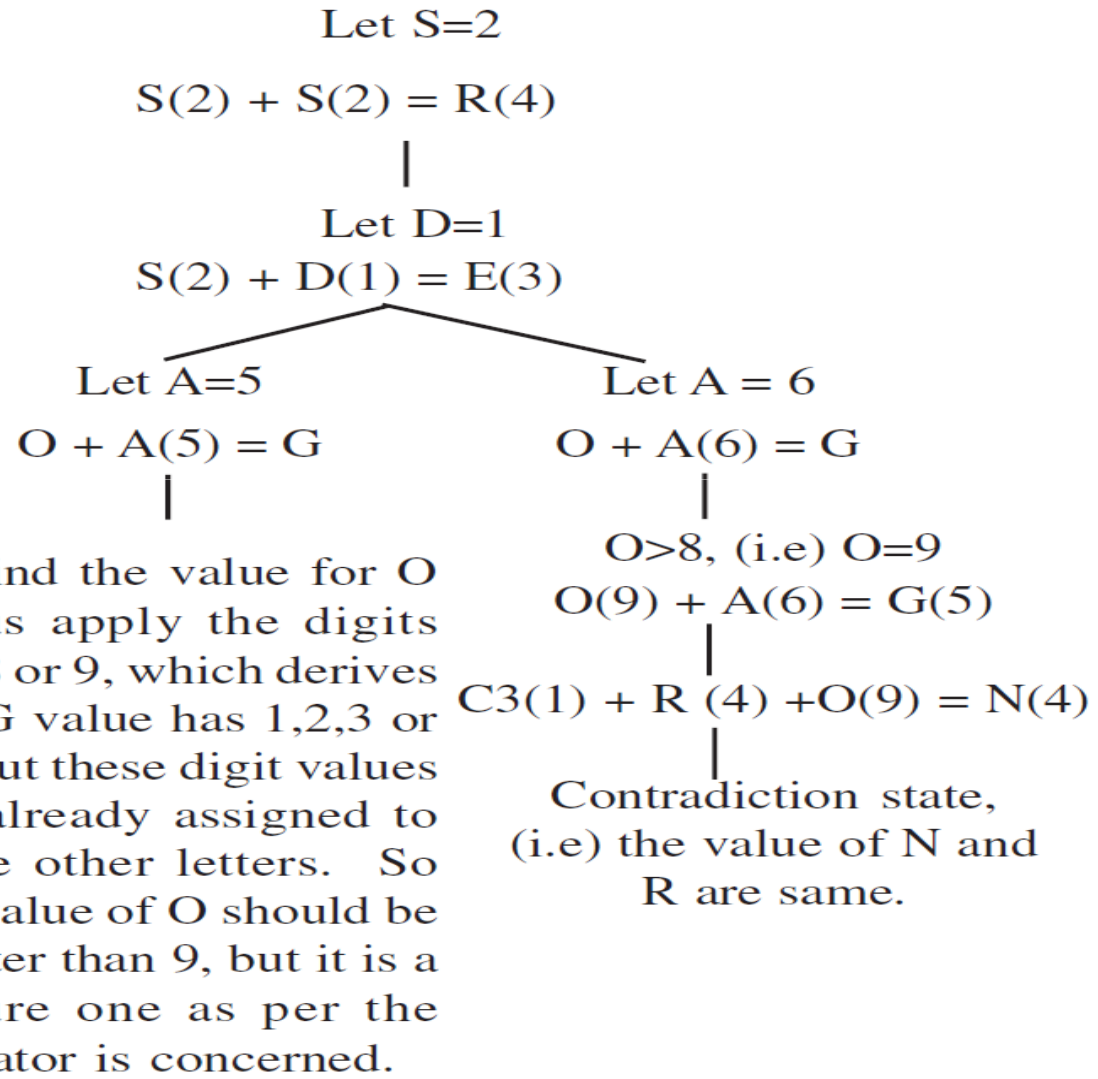
CSP – Procedure

- two-step process
  - Constraints are discovered and propagated as far as possible throughout the system. Then, if there is still not a solution, search begins.
  - A guess about something is made and added as a new constraint. Propagation can then occur with this new constraint, and so forth

CSP – Example - Crypt arithmetic

- Given : CROSS + ROADS = DANGER
- Initial state : C R O S A D N G E C1 C2 C3 C4 = ?
- Goal state : The digits to the letter should be assigned in such a manner that the sum is satisfied.

CSP – Example - Crypt arithmetic



CSP – Example - Crypt arithmetic

Let S=3

S(3) +S(3)=R(6)

Let D=1

S(3) + D(1)=E(4)

Let O=2

A + O(2)=G

Let A=5

A(5) + O(2)=G(7)

Let R=6

R(6) + O(2)=N(8)

C + R(6)=A(5)

$\therefore C=9$

CSP – Example - Crypt arithmetic

	C4(0)	C3(0)	C2(0)	C1(0)	
	C(9)	R(6)	O(2)	S(3)	S(3)
	R(6)	O(2)	A(5)	D(1)	S(3)
D(1)	A(5)	N(8)	G(7)	E(4)	R(6)

Means-Ends Analysis

- search process reduces the difference between the current state and the goal state until the required goal is achieved
- Allows both backward and forward searching
- Solve major parts of a problem first and then return to smaller problems when assembling the final solution – operator subgoaling
- Example :
  - GPS was the first AI program to exploit means-ends analysis.
  - STRIPS (A robot Planner)

Means-Ends Analysis - procedure

- Until the goal is reached or no more process are available:
  - Describe the current state, the goal state and the differences between the two.
  - Use the difference between the current state and goal state, possibly with the description of the current state or goal state, select a promising procedure.
  - Use the promising procedure and update current state.
- If goal is reached then **success** otherwise **failure**.

Example: Household robot domain

- **Problem:** Move desk with two things on it from one location S to another G. Find a sequence of actions robot performs to complete the given task.
- Operators are: PUSH, CARRY, WALK, PICKUP, PUTDOWN and PLACE given with preconditions and results.
- S B_____C G

Start PUSH Goal

Example: Household robot domain

- S(Start)→ walk(start_desk_loc) → pickup(obj1) → putdown(obj1) → pickup(obj2)→ putdown(obj2) →push(desk, goal_loc) → walk(start_desk_loc) → pickup(obj1) → carry(obj1,goal_loc) → place(obj1,desk) → walk(start_desk_loc) → pickup(obj2) → carry(obj2,goal_loc) → place(obj2,desk) →G(Goal).

Simple Hill Climbing

- Search technique that moves in the direction of increasing value (Peak)
- When the choice of the neighbor solution is done by taking the one locally maximizing the criterion, the meta heuristic takes the name hill climbing

Drawbacks of hill – climbing search

- ***Local maxima (or) Foot hills***
- ***Plateau (or) shoulder***
- ***Ridges***

Steepest – Ascent Hill Climbing

- considers all the moves from the current state and selects the best one as the next state.
- A variation from the previous algorithm is considering all possible moves of current state

Algorithm

function HILL-CLIMBING(*problem) returns a state that is a local maximum*

inputs:

problem, a problem

local variables: *current, a node*

next, a node

current←MAKE-NODE(INITIAL-STATE[problem])

loop do

next ← a highest-valued successor of current

if VALUE[next]≤VALUE[current] then return STATE[*current]*

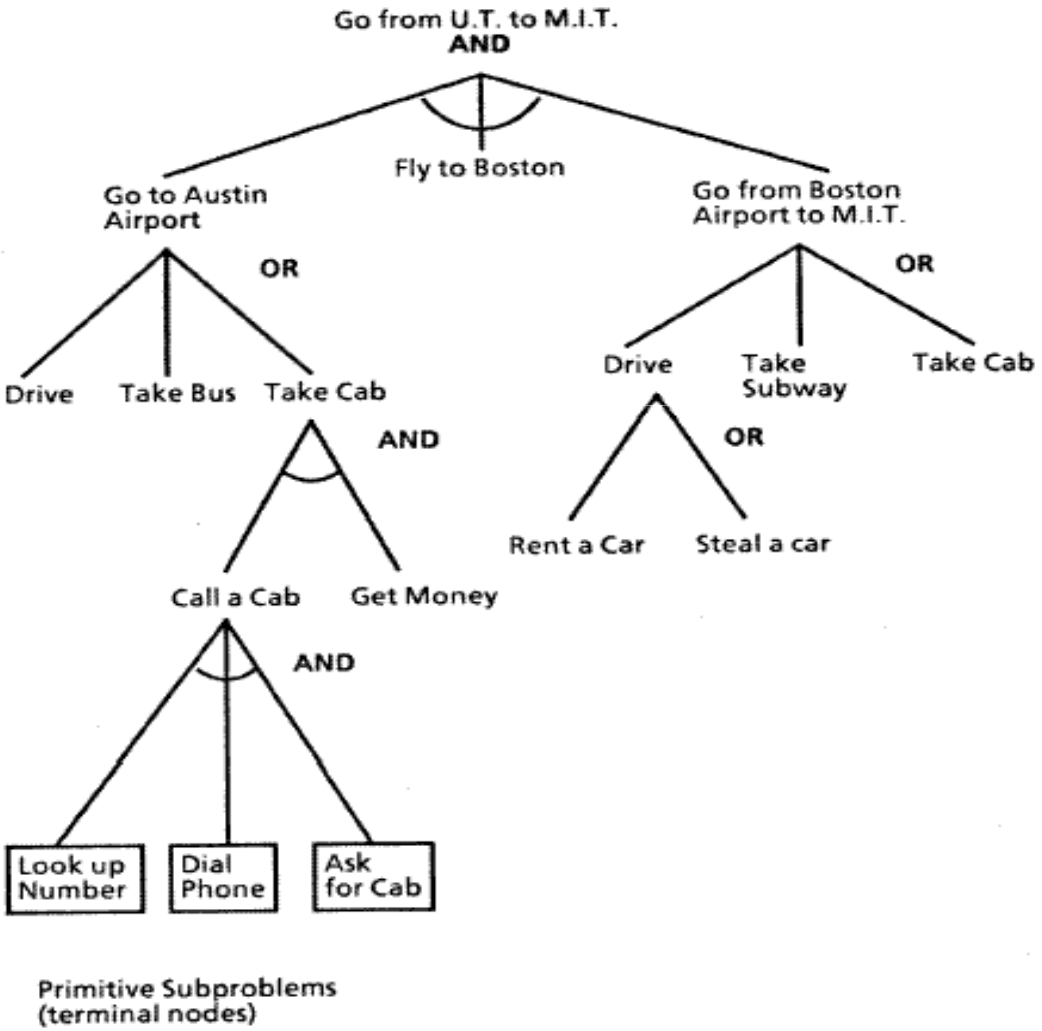
current←next

Problem Reduction- AND-OR graph

AND-OR graph is useful for certain problems where

- The solution involves decomposing the problem into smaller problems.
- We then solve these smaller problems

AND-OR graph - Example

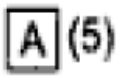


Algorithm – AO*

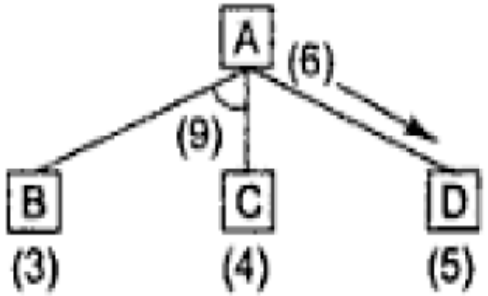
1. Initialise the graph to start node
2. Traverse the graph following the current path accumulating nodes that have not yet been expanded or solved
3. Pick any of these nodes and expand it and if it has no successors call this value *FUTILITY* otherwise calculate only *f'* for each of the successors.
4. If *f'* is 0 then mark the node as *SOLVED*
5. Change the value of *f'* for the newly created node to reflect its successors by back propagation.
6. Wherever possible use the most promising routes and if a node is marked as *SOLVED* then mark the parent node as *SOLVED*.
7. If starting node is *SOLVED* or value greater than *FUTILITY*, stop, else repeat from 2.

AO* Example – Route finding

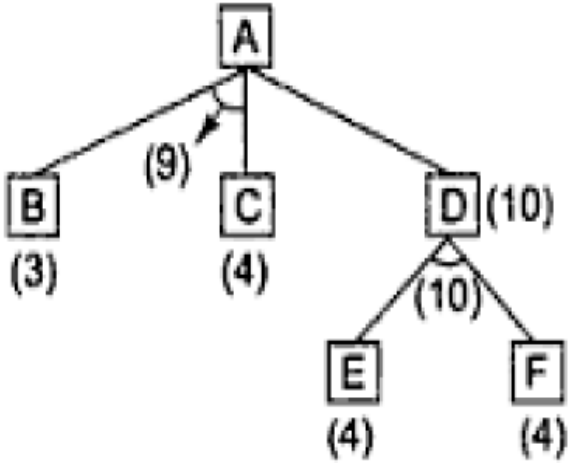
Before step 1



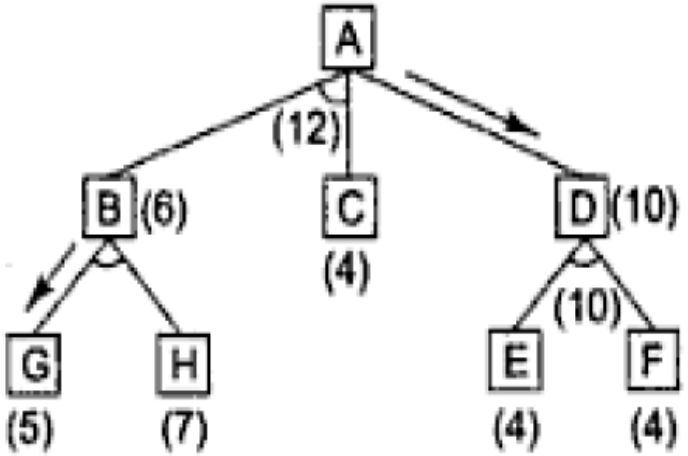
Before step 2



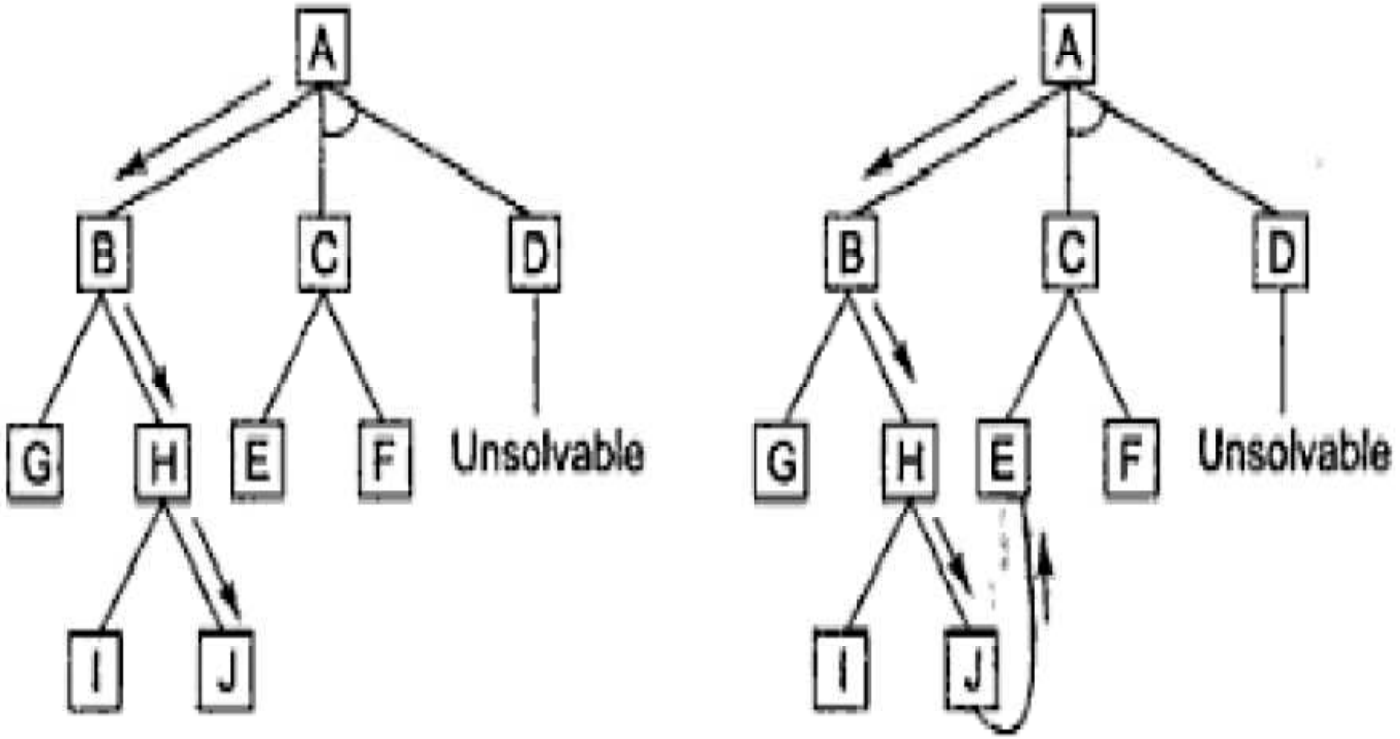
Before step 3



Before step 4



AO* Example – Route finding



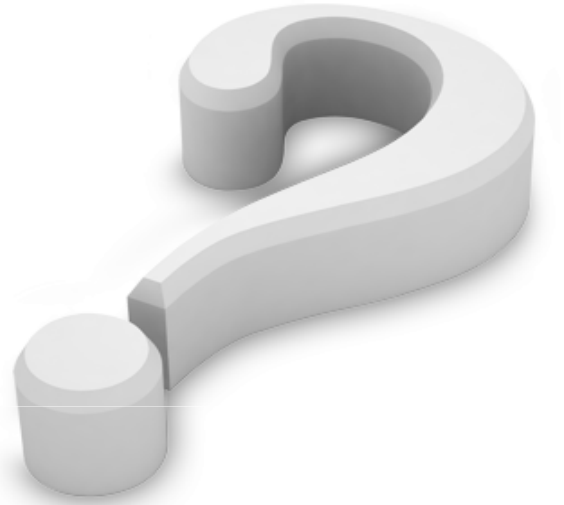
To solve….

SEND + MORE = MONEY

To solve….

SEND + MORE = MONEY

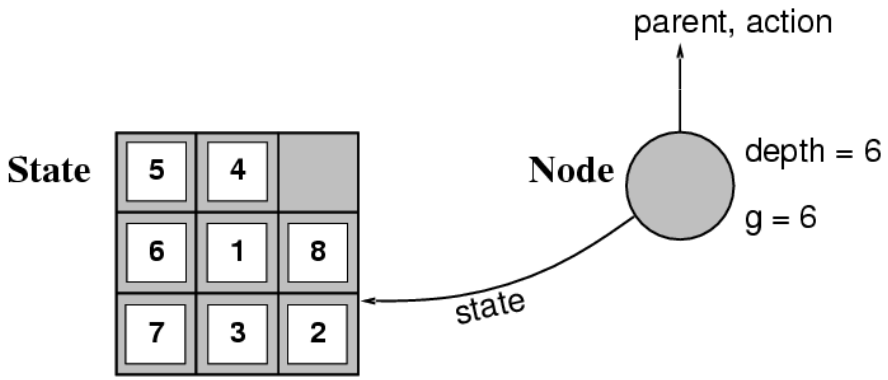
Solution: 9567 + 1085 = 10652



Thank You

States versus Nodes

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree contains info such as: state, parent node, action, path cost *g(x)*, depth



- The Expand function creates new nodes, filling in the various fields and using the SuccessorFn of the problem to create the corresponding states.

State Spaces versus Search Trees

- State Space
  - Set of valid states for a problem
  - Linked by operators
  - e.g., 20 valid states (cities) in the Romanian travel problem
- Search Tree
  - Root node = initial state
  - Child nodes = states that can be visited from parent
  - Note that the depth of the tree can be infinite
    - E.g., via repeated states
  - Partial search tree
    - Portion of tree that has been expanded so far
  - Fringe
    - Leaves of partial search tree, candidates for expansion

Search trees = data structure to search state-space