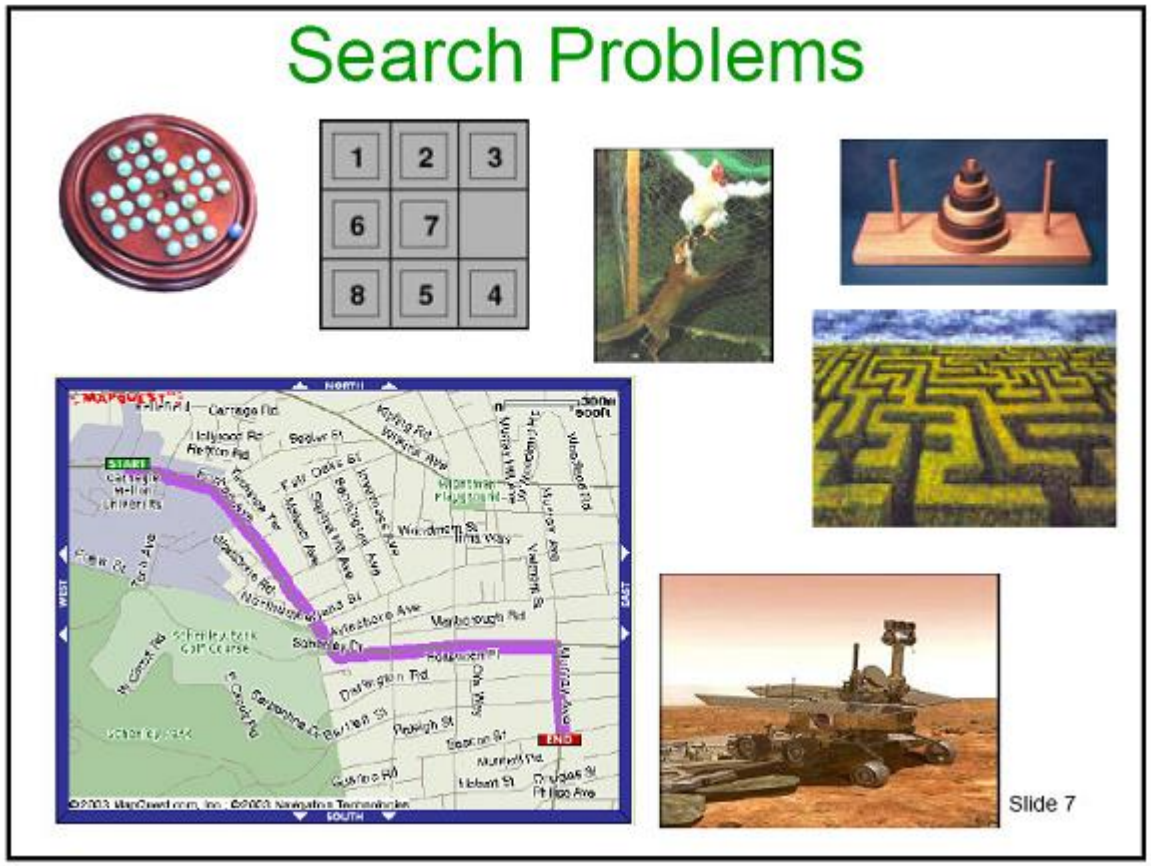
CHAPTER : THREE
Problem solving

Can solve many problems with lots of computing power that humans are not good



Problem

- It is a gap between what actually is and what is desired.
  - A problem exists when an individual becomes aware of the existence of an obstacle which makes it difficult to achieve a desired goal or objective.

- A number of problems are addressed in AI, both:
  - **Toy problems**: are problems that are useful to test and demonstrate methodologies.
    - Can be used by researchers to compare the performance of different algorithms
    - e.g. 8-puzzle, n-queens, vacuum cleaner world, towers of Hanoi, …
  - **Real-life problems:** are problems that have much greater commercial/economic impact if solved.
    - Such problems are more difficult and complex to solve, and there is no single agreed-upon description
    - E.g. Route finding, Traveling sales person, etc.

Solving a problem

Formalize the problem: Identify the collection of information that the agent will use to decide what to do.

- Define states
  - States describe distinguishable stages during the problem-solving process
  - Example- What are the various states in **route finding** problem?
    - The various places including the location of the agent

- Define the available operators/rules for getting from one state to the next
  - Operators cause an action that brings transitions from one state to another by applying on a current state

- Suggest a suitable representation for the **problem space/state space**
  - Graph, table, list, set, … or a combination of them

State space of the problem

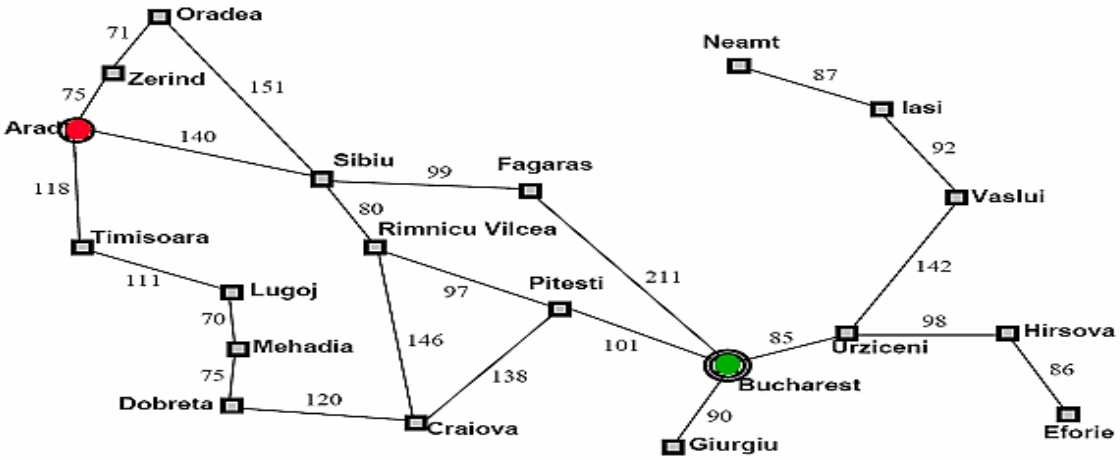
- The **state space** defines the set of all relevant states reachable from the initial state by (any) sequence of actions through iterative application of all permutations and combinations of operators
- **State space** (also called **search space/problem space)** of the problem includes the various states
  - Initial state: defines where the agent starts or begins its task
  - Goal state: describes the situation the agent attempts to achieve
  - Transition states: other states in between initial and goal states

Exercise– Find the state **space** for route finding problem where the agent wants to go from **polle** to **shewaber**.

- Think of the states reachable from the initial state until we reach to the goal state.

Example 1: Romania Travel.

Currently in Arad, need to get to Bucharest by tomorrow to catch a flight. What is the State Space?



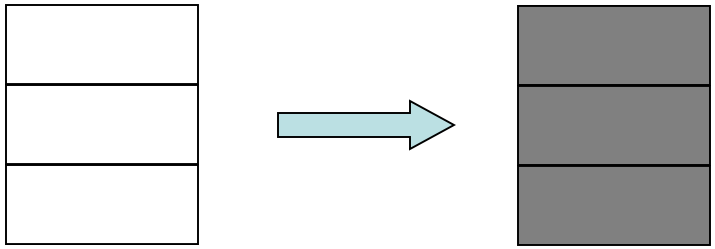
- **State space.**
  - **States**: the various cities you could be located in.
  - **Actions**: drive between neighboring cities.
  - **Initial state**: in Arad
  - **Desired condition (Goal):** be in a state where you are in Bucharest.

- Once the problem has been formulated as a state space search, various algorithms can be utilized to solve the problem

- Solution will be the route, the sequence of cities to travel through to get to Bucharest.

Example 2: Coloring problem

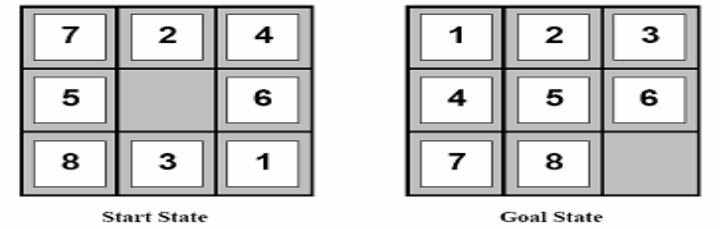
– There are 3 rectangles. Both are initially white. The problem is to change all rectangles with white color to black color. Color change is one rectangle at a time. Define state space for coloring problem?



Example 3: The 8 puzzle problem

- Arrange the tiles so that all the tiles are in the correct positions. You do this by moving tiles or space. You can move a tile/space up, down, left, or right, so long as the following conditions are met:
  A) there's no other tile blocking you in the direction of the movement; and

  B) you're not trying to move outside of the boundaries/edges.



▪**State space.**

• **States**: The different configurations of the tiles.

• **Actions**: Moving the blank up, down, left, right. Can every action be performed in every state?

• **Initial state**: as shown in the 1st picture.

• **Desired condition (Goal):** be in a state where the tiles are all in the positions shown in the 2nd picture

▪Solution will be a sequence of moves of the blank that transform the initial state to a goal state.

Knowledge and types of problems

- There are **four types** of problems:
  - Single state problems (Type 1)
  - Multiple state problems (Type 2)
  - Exploration problems (Type 3)
  - Contingency Problems (Type 4)

- This classification is based on the level of knowledge that an agent can have concerning its action and the state of the world

- Thus, the first step in formulating a problem for an agent is to see what knowledge it has concerning
  - The effects of its action on the environment
  - Accessibility of the state of the world

- This knowledge depends on how it is connected to the environment via its percepts and actions

Single state problem

- **Fully observable:** The world is accessible to the agent
  - It can determine its exact state through its sensors
  - The agent's sensor knows which state it is in
- **Deterministic:** The agent knows exactly the effect of its actions
  - It can then calculate exactly which state it will be in after any sequence of actions
- Action sequence is completely planned

Multiple state problems

- **Partially observable:** The agent has limited access to the world state
  - It might not have sensors to get full access to the environment states or as an extreme, it can have no sensors at all (due to lack of percepts)
- **Deterministic:** The agent knows exactly what each of its actions do
  - It can then calculate which state it will be in after any sequence of actions
- If the agent has full knowledge of how its actions change the world, but does not know of the state of the world, it can still solve the task

Contingency Problems

- **Partially observable:** The agent has limited access to the world state
- **Non-deterministic:** The agent is ignorant of the effect of its actions
- Sometimes ignorance prevents the agent from finding a guaranteed solution sequence.
- Many problems in the real world are contingency problems (exact prediction is impossible)
  - For this reason many people keep their eyes open while walking around or driving.
- Suppose the agent is in **Murphy's law world**
  - The agent has to sense during the execution phase, since things might have changed while it was carrying out an action. This implies that
    - the agent has to compute a tree of actions, rather than a linear sequence of action.

Exploration problem

- The agent has no knowledge of the environment
  - **World Partially observable** : No knowledge of states (environment)
    - Unknown state space (no map, no sensor)
  - **Non-deterministic**: No knowledge of the effects of its actions
  - Problem faced by (intelligent) agents (like, **new born babies**)

- This is a kind of problem in the real world rather than in a model, which may involve significant danger for an ignorant agent. If the agent survives, it learns about the environment

- The agent must experiment, learn and build the model of the environment through its results, gradually, discovering
  - What sort of states exist and What its action do
  - Then it can use these to solve subsequent (future) problems

Well-defined problems and solutions

To define a problem, we need the following elements: states, operators, goal test function and cost function.

- **The Initial state:** is the state that the agent starts in or begins with.
  - **Example-** the initial state for each of the following:
    - Coloring problem
      - All rectangles white
    - Route finding problem
      - The point where the agent start its journey (SidistKilo?)
    - 8-puzzle problem ??

Operators

- The set of possible actions available to the agent, i.e.
  - –which state(s) will be reached by carrying out the action in a particular state
  - –**A Successor function S(x)**
    - Is a function that returns the set of states that are reachable from a single state by **any** single action/operator
    - Given state x, S(x) returns the set of states reachable from x by any single action

- Example:
  - –Coloring problem: Paint with black color

    paint (color w, color b)

  - –Route finding problem: Drive through cities/places

    drive (place x, place y)

  - – 8-puzzle ??

Goal test function

- The agent execute to determine if it has reached the goal state or not
  - Is a function which determines if the state is a goal state or not

- Example:
  - Route finding problem: Reach Addis Ababa airport on time

    → IsGoal(x, Addis_Ababa)

  - Coloring problem: All rectangles black

    → IsGoal(rectangle[], n)

  - 8-puzzle ??

Path cost function

- A function that assigns a cost to a path (sequence of actions).
  - Is often denoted by **g.** Usually, it is the sum of the costs of the individual actions along the path (from one state to another state)
  - Measure path cost of a sequence of actions in the state space. For example, we may prefer paths with fewer or less costly actions

- Example:
  - Route finding problem:
    - Path cost from initial to goal state
  - Coloring problem:
    - One for each transition from state to state till goal state is reached
  - 8-puzzle?

Steps in problem solving

- **Goal formulation**
  - is a step that specifies exactly what the agent is trying to achieve
  - This step narrows down the **scope** that the agent has to look at

- **Problem formulation**
  - is a step that puts down the **actions** and **states** that the agent has to consider **given a goal** (avoiding any redundant states), like:
    - the initial state
    - the allowable actions etc…

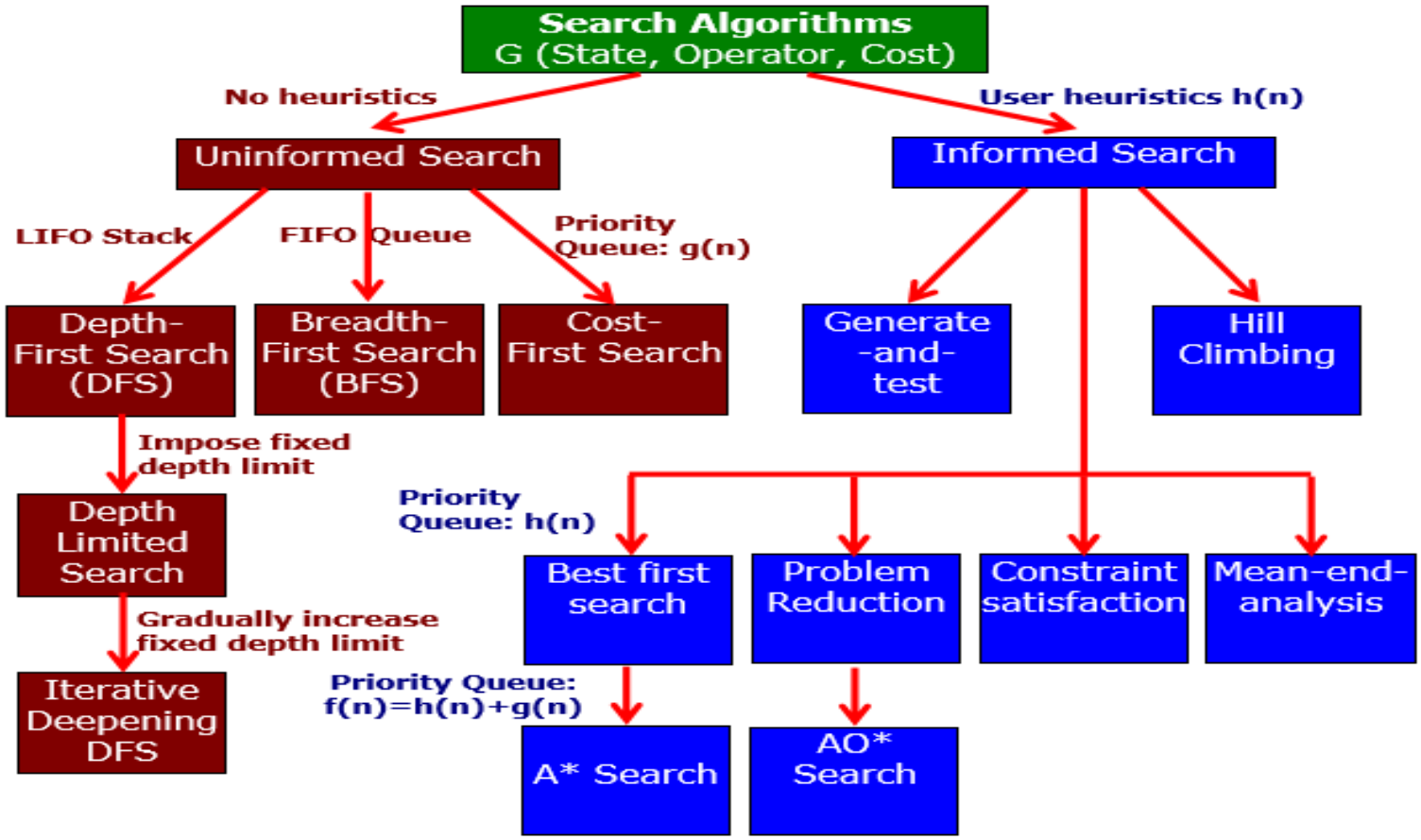
- **Search**
  - is the process of looking for the **various sequence of actions** that lead to a goal state, evaluating them and choosing the **optimal** sequence.

- **Execute**
  - is the final step that the agent executes the chosen sequence of actions to get it to the solution/goal

Searching Algorithms

- **Uniformed**: also called blind, exhaustive, or brute force search. Uses no information about the problem to guide the search, *may not efficient*
  - *These strategies do not take into account any domain specific information about the particular search problem.*
  - Popular uninformed search techniques*:* *Breadth-First, Uniform-Cost, Depth-First, Depth-Limited, and Iterative Deepening search*
- **Informed**: also called heuristic or intelligent search, uses information about the problem to guide the search, usually guesses the distance to a goal state and *therefore efficient*, but the search may not be always possible

Searching Algorithms



Searching

- Examine different possible sequences of actions & states, and come up with **specific sequence** of operators/actions that will take you from the initial state to the goal state

  –*Given a state space with **initial state** and **goal state,** find* optimal ***sequence of actions** leading through a **sequence of states** to the **final goal state***

- **Searching in State Space**

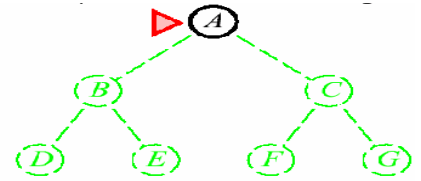
  – *Choose min-cost/max-profit node/state in the state space,*

  – *Test the state to know whether we reached to the goal state or not,*

  – *If not, expand the node further to identify its successor.*

Search Tree

- A search tree is a representation
  - in which *nodes denote paths* and *branches connect paths*.
  - The node with no parent is the **root node**.
  - The nodes with no children are called **leaf**



- *Two functions* needed for conducting search
  - ***Generator (or successors) function****:* Given a state and action, produces its successor states (in a state space)
  - ***Tester (or IsGoal) function***: Tells whether given state S is a goal state

    IsGoal(S) → True/False
  - IsGoal and Successors functions depend on problem domain.

Search algorithm

- ***Input***: a given problem (Initial State + Goal state + transit states and operators)
- ***Output***: returns optimal sequence of actions to reach the goal.
- **Algorithm Evaluation:**
  - *Completeness*: will the search always find a solution if a solution exists ?
  - *Optimality*: will the search always find the *least cost solution*? (when actions have costs)
  - *Time complexity*: how long does it take to find a solution ?
  - *Space complexity*: what is the maximum space used by the algorithm ?
    - *Fast programs can be written But they use up too much memory*
    - *Memory efficient programs can be written But they are slow*

Search strategy

- ***Uninformed (blind) search***
  - They do not need domain knowledge that guide them to the right direction towards the goal
  - Have no information about *the number of steps* or *the path cost* from the current state to the goal
  - It is important for problems for which *there is no additional information* to consider
  - They work fine with *small number of possible states*

Cont.

- ***Informed (heuristic) search***
  - Have problem-specific knowledge (knowledge that is true from experience)
  - Have knowledge about how far are the various state from the goal
  - *Can find solutions more efficiently than uninformed search*

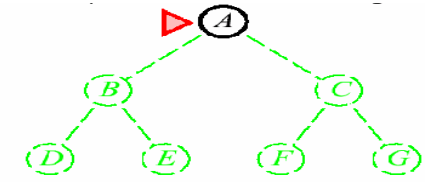
Cont.

- **Uninformed search**
  - Breadth first
  - Depth first
  - Uniform cost, …
  - Depth limited search
  - Iterative deepening
  - etc.

- **Informed search**
  - Greedy search
  - A*-search
  - Iterative improvement,
  - Constraint satisfaction
  - etc.

Uninformed search-*Breadth first search*

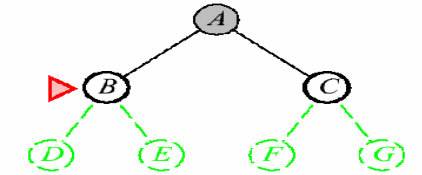
- *Expand all nodes on a given level* of the search tree before moving to the next level

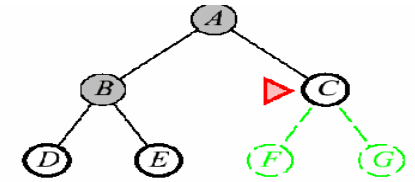


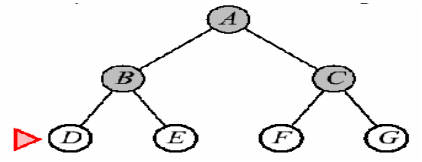
- **Implementation**: use **FIFO queue** data structure to store the list:

  – *New successors at the end of queue*

  – *Pop nodes from the front of the queue*





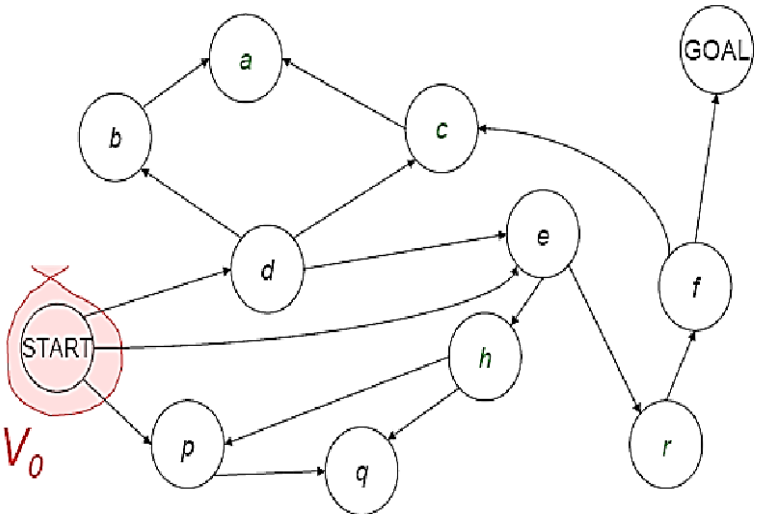


- **Properties**:
  - –Takes space: keeps every node in memory
  - –Optimal and complete: guarantees to find solution

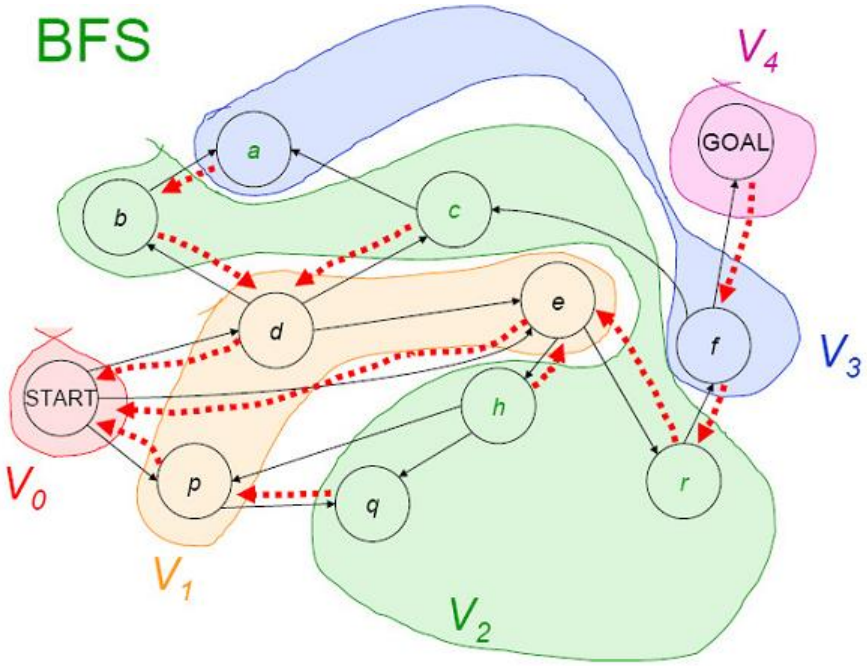
Example

Visiting a vertex – Going to a particular vertex

Exploration of Vertex - Visiting all adjacent vertexes



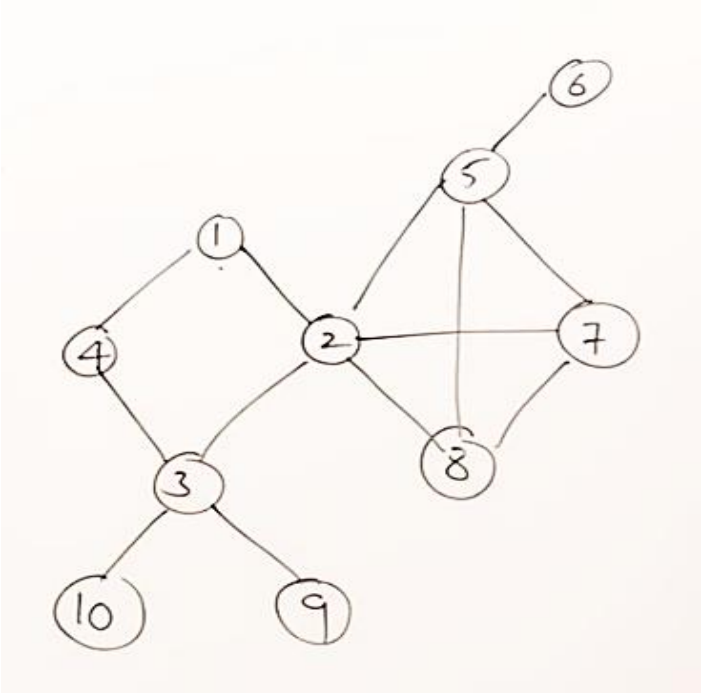
Cont.



Solution is:

Start => **e** => **r** => **f** => **Goal**

Exercise



- **1** is the starting point and **6** is the goal
- *Identify the possible path to reach the goal using BFS*

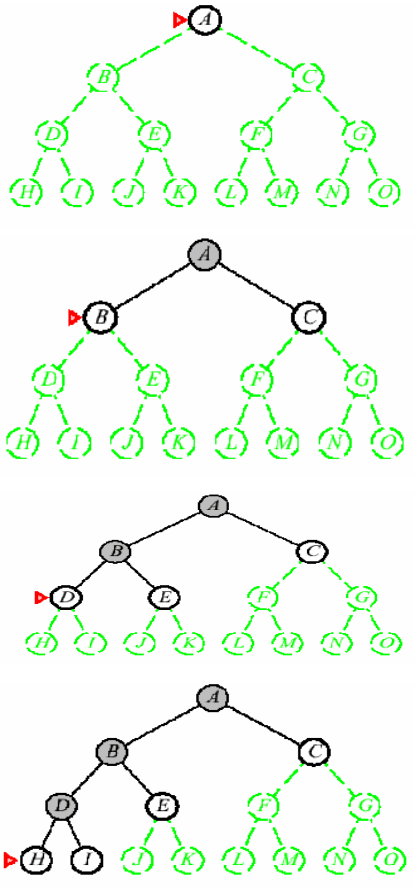
Cont.

- If ***branching factor*** (average number of child nodes for a given node) = **b** and ***depth*** = **d**, then number of nodes at level d **=b^d**
- The total no of nodes created in worst case is $b + b^2 + b^3 + \ldots\ldots + b^d$

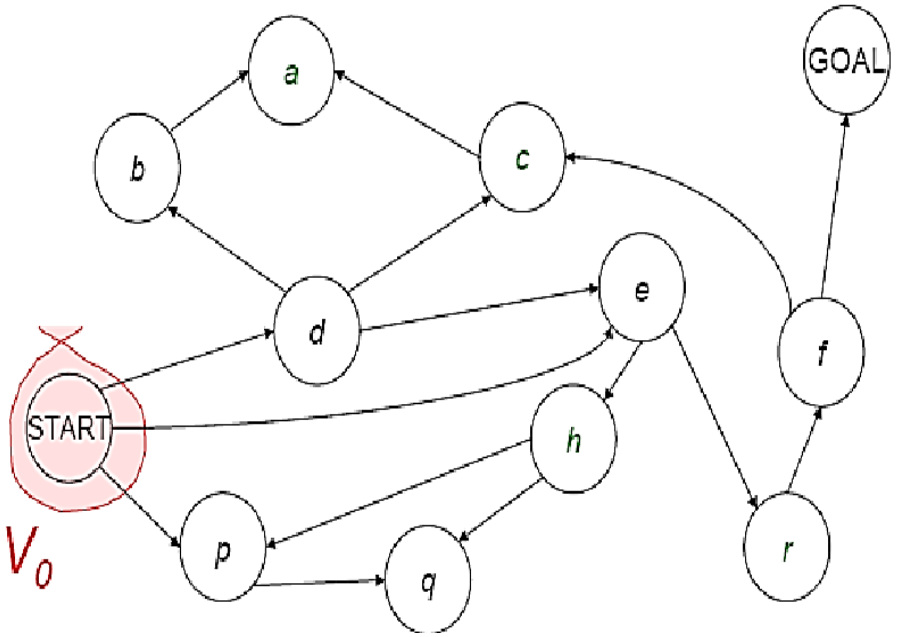
- *Disadvantage:* Since each level of nodes is saved for creating next one, it consumes a lot of memory space

Uninformed search-*Depth-first search*

- *Expand one of the node at the deepest level* of the tree.
  - –Only when the search hits a non-goal dead end does the search go back and expand nodes at shallower levels
- **Implementation**: treat the list as **LIFO Stack**
  - – *New successors at the top of stack*
  - – *Pop nodes from the top of the stack*
- **Properties**
  - – Incomplete and not optimal: fails in infinite-depth spaces, spaces with loops.
  - – Takes less space (Linear): Only needs to remember up to the depth expanded

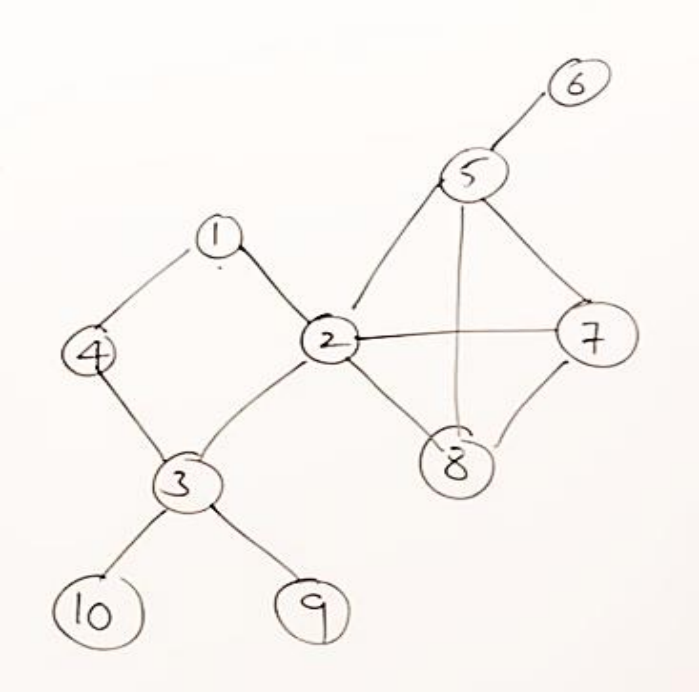


Example



Solution is:
Start => **e** => **r** => **f** => **Goal**

Exercise



- **1** is the starting point and **6** is the goal
- *Identify the possible path to reach the goal using DFS*

Cont.

- *Disadvantage:* This algorithm may not terminate and go on infinitely on one path
  - *The solution to this is to choose a cut-off depth*

Uniformed Searching strategy

BFS

BFS Stands for "**Breadth First Search**".

BFS starts traversal from the root node and then explore the search in the level by level manner i.e. as close as possible from the root node.

Breadth First Search can be done with the help of **queue** i.e. **FIFO** implementation.

This algorithm works in single stage. The visited vertices are removed from the queue and then displayed at once.

BFS is **slower** than DFS.

BFS requires **more** memory compare to DFS.

Applications of BFS

> To find Shortest path
> Single Source & All pairs shortest paths
> In Spanning tree
> In Connectivity

BFS is useful in finding shortest path.BFS can be used to find the shortest distance between some starting node and the remaining nodes of the graph.

DFS

DFS stands for "**Depth First Search**".

DFS starts the traversal from the root node and explore the search as far as possible from the root node i.e. depth wise.

Depth First Search can be done with the help of **Stack** i.e. **LIFO** implementations.

This algorithm works in two stages – in the first stage the visited vertices are pushed onto the stack and later on when there is no vertex further to visit those are popped-off.

DFS is more **faster** than BFS.

DFS require **less** memory compare to BFS.

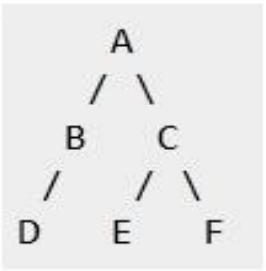
Applications of DFS

> Useful in Cycle detection
> In Connectivity testing
> Finding a path between V and W in the graph.
> useful in finding spanning trees & forest.

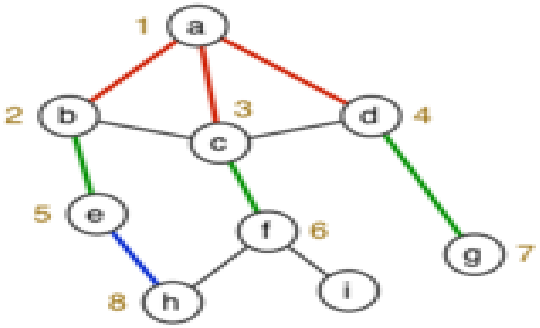
DFS in not so useful in finding shortest path. It is used to perform a traversal of a general graph and the idea of DFS is to make a path as long as possible, and then go back (**backtrack**) to add branches also as long as possible.

Difference between BFS and DFS | BFS vs. DFS

Example :

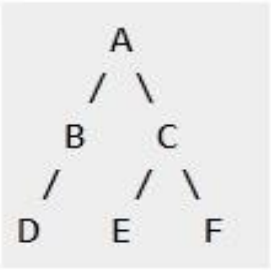


A, B, C, D, E, F

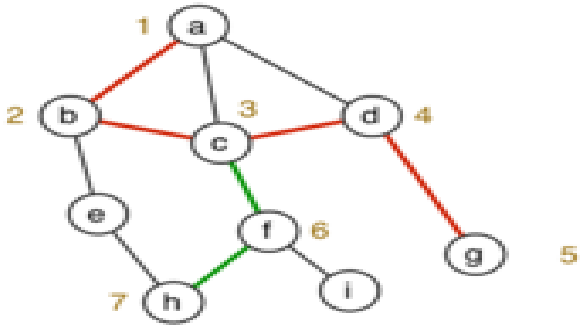


Breadth-first Search

Example :

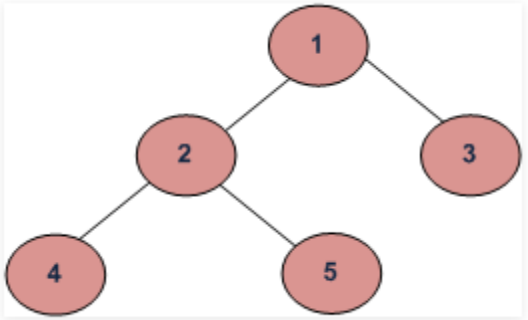


A, B, D, C, E, F



Depth-first Search

- **BFS vs DFS for Binary Tree**
- **What are BFS and DFS for Binary Tree?**
  A Tree is typically traversed in two ways:
- Breadth First Traversal (Or Level Order Traversal)
- Depth First Traversal
  - In order Traversal (Left-Root-Right)
  - Preorder Traversal (Root-Left-Right)
  - Post order Traversal (Left-Right-Root)
- BFS and DFSs of above Tree
- Breadth First Traversal : 1 2 3 4 5
- Depth First Traversals:
- In order Traversal : 4 2 5 1 3
- Preorder Traversal : 1 2 4 5 3
- Post order Traversal : 4 5 2 3 1



Discussion Point

- Compare breadth-first and depth-first search.
  - When would breadth-first be preferable?
  - When would depth-first be preferable?

- Breadth-First:
  - *When completeness is important.*
  - *When optimal solutions are important.*
- Depth-First:
  - *When solutions are dense and low-cost is important, especially space costs*.

Assignment - 1

- Read and discuss each of the following searching algorithms using an example clearly.
  - ***Uniform-Cost Search***
  - ***Iterative Deepening Search (IDS)***
  - ***Bidirectional Search***
  - ***Depth limited search***

Informed search

- In uninformed search, we don't try to evaluate which of the nodes on the frontier/limit are most promising.

- Search efficiency would improve greatly if there is a way to order the choices so that the most promising are explored first.

- This requires domain knowledge of the problem (i.e. heuristic) to undertake focused search.

Cont.

- A **Heuristic** is an operationally-effective piece of information on how to direct search in a problem space.
- Heuristics are only approximately correct. Their purpose is to minimize search on average.
  - It is an estimate of how close we are to a goal state
  - **h(n)** = guesses the cost of getting to the goal from node n
- There are different ways of guessing this cost in different domains. I.e., heuristics are domain specific.

Cont.

- Search algorithms are admissible with respect to a search method if it guarantees finding the **optimal solution first**, even when its value is only an estimate.
  - The algorithm always terminates in an optimal path from the initial state to goal node if one exists.

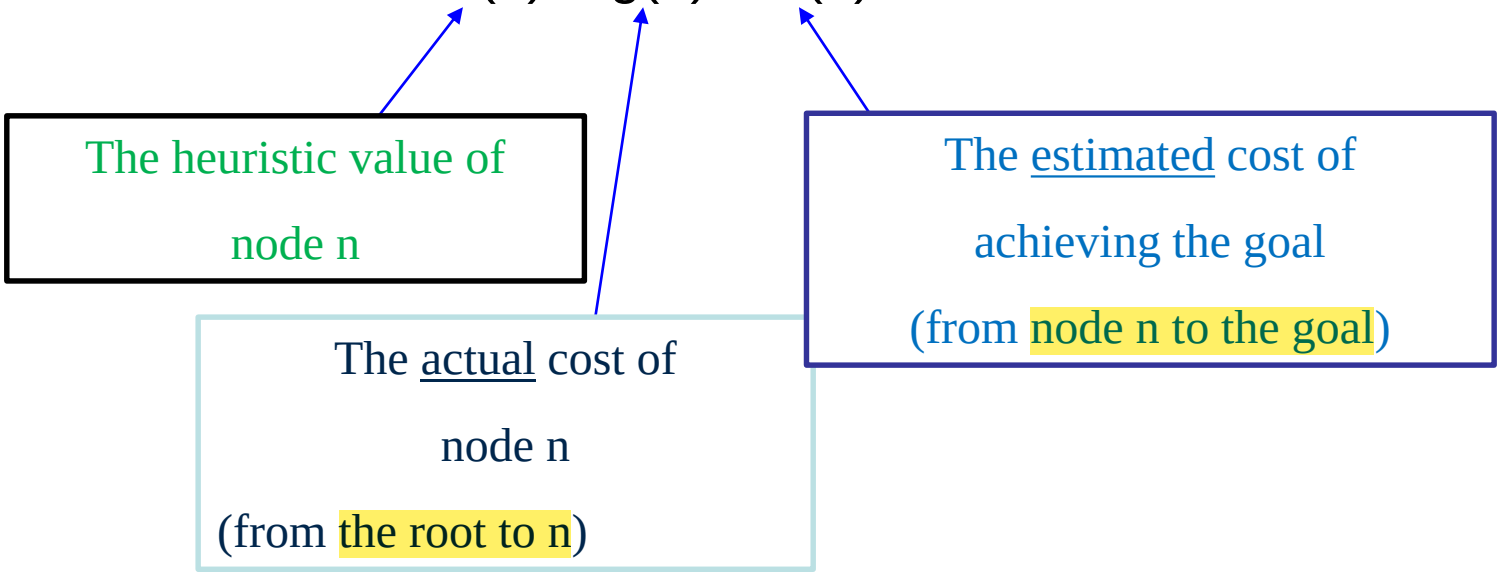
Informed search- Best-first search

- It is a generic name to the class of informed methods
- The two best first approaches to find the shortest path:
  - **Greedy Search**
  - **A*Search**
- When expanding a node *n* in the search tree, *greedy search uses the estimated cost to get from the current state to the goal state*, define as ***h(n).***
- And A*Search uses ***h(n)*** and the sum of the cost to reach that node from the start state, define as ***g(n).***

Cont.

- For each node in the search tree, an evaluation function *f(n)* can be defined as the sum of these functions.

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



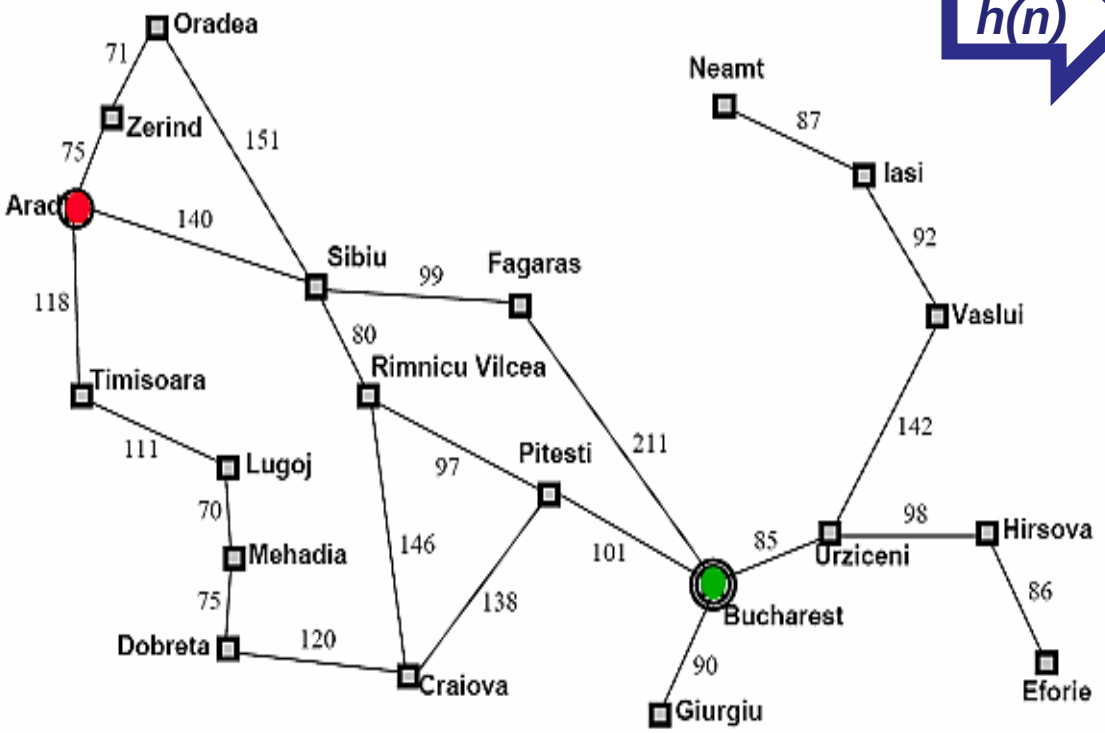
Informed search- *Greedy best-first search*

- Is a best first search that uses a heuristic function h(n) alone to guide the search
  - –Selects node to expand that is closest to a goal node (hence it's greedy)
  - –The algorithm doesn't take minimum path costs from initial to current nodes into account
  - –Always expand node with lowest h-value.

- **Implementation**:
  - –expand 1st the node closest to the goal state, i.e. with evaluation function f(n) = h(n)
    - –h(n) = 0 if node n is the goal state
    - –Otherwise if h(n) ≥ 0; continue with an estimated cost of the cheapest path from node n to a goal state

Cont.

- **E.g.**

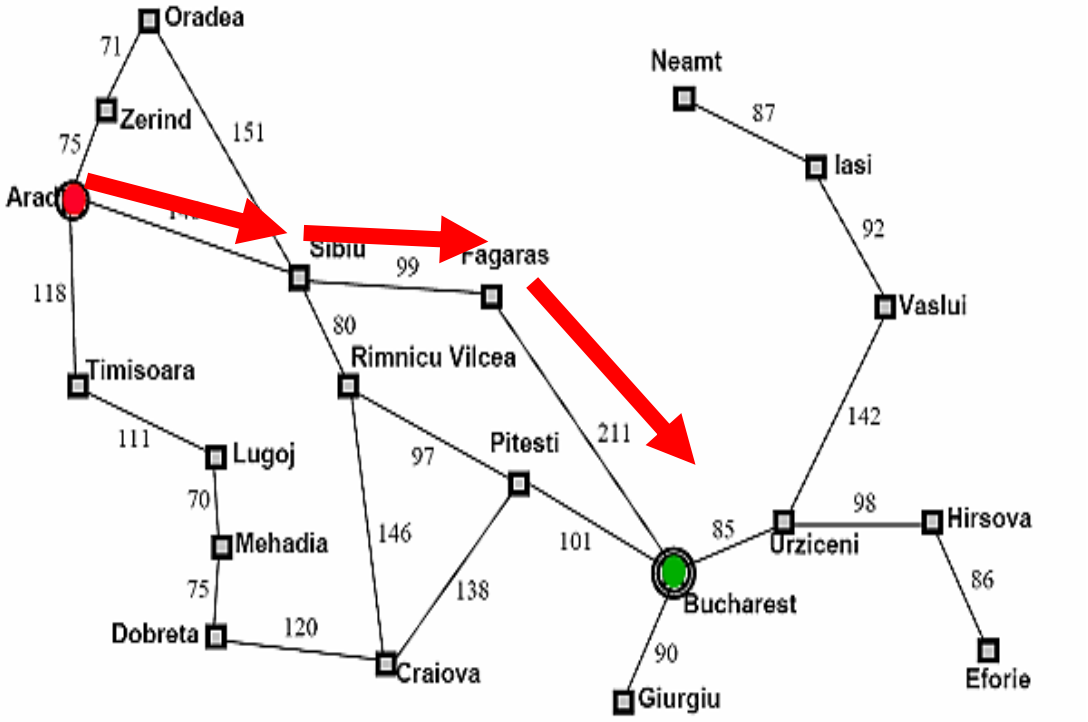


h(n)

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

Cont.

- **Solution**
  - We use h(n) to rank the nodes on the frontier.

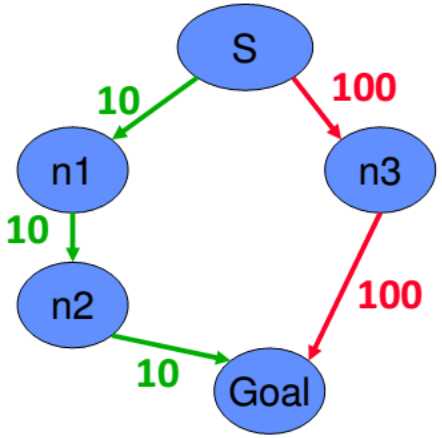


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Cont.

- However, this method **ignores the cost of getting to n**, so it can be lead astray exploring nodes that cost a lot to get to but seem to be close to the goal:

- **E.g.**



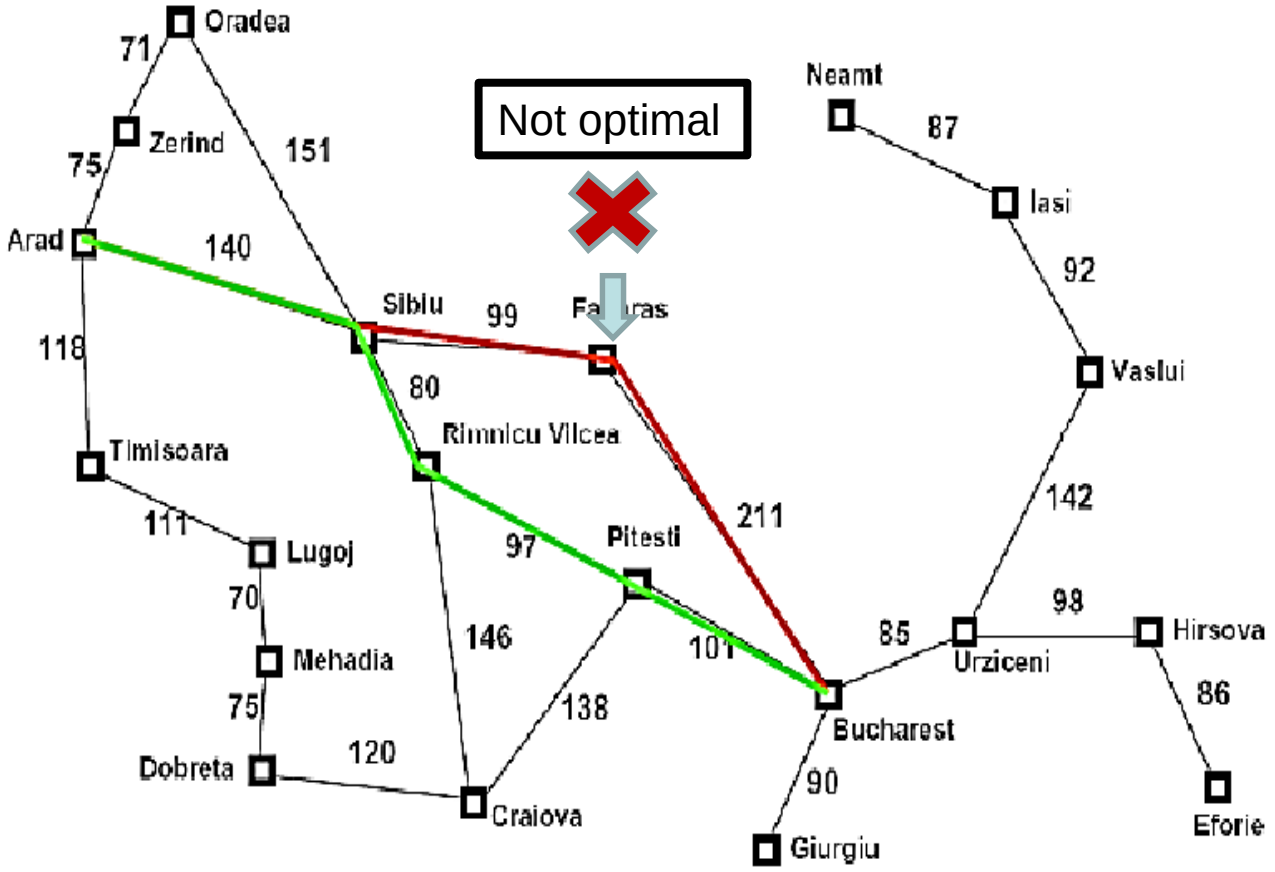
h(n1) = 70
h(n3) = 50

Solution
S => n3 => Goal

Not optimal

Cont.

- **E.g. 2**



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

Cont.

- **Properties**
  - **Incomplete and Not optimal**
    - May not reach goal state because it may start down an infinite path and never return to try other possibilities
  - **Exponential time complexity and memory requirement**
    - It retains all the nodes in memory
    - With good heuristic function (that estimate cost nearer to actual cost) the space and time complexity can be reduced substantially

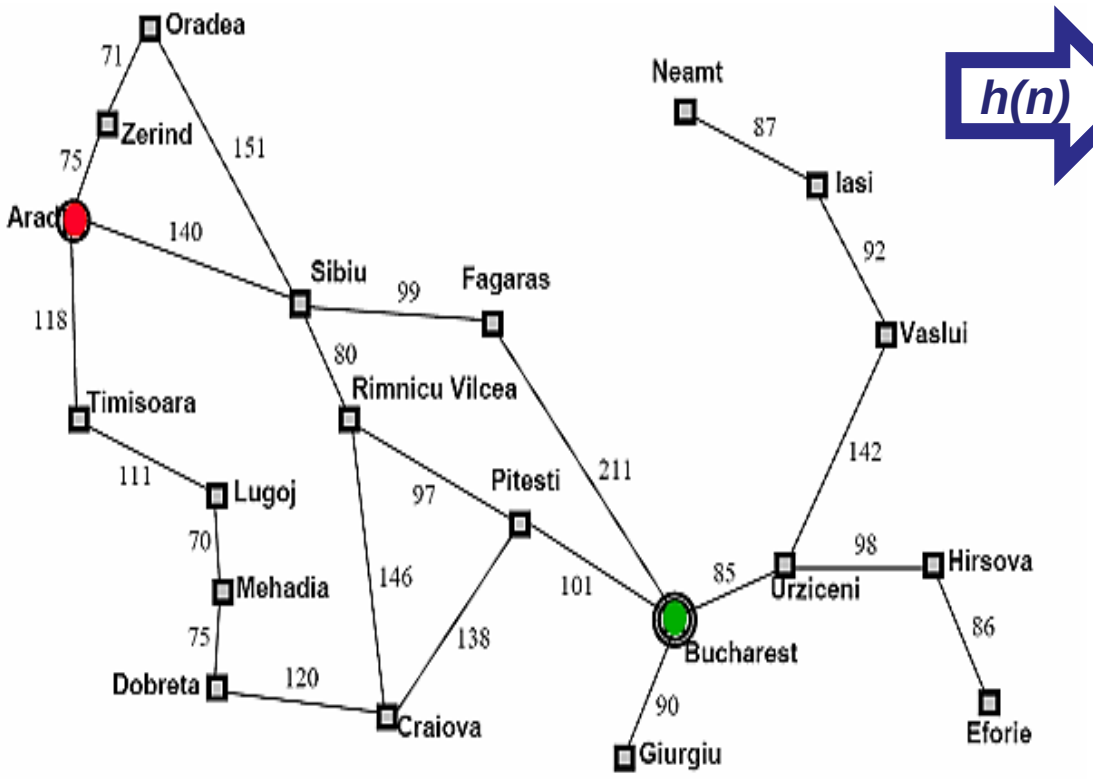
- The **problem** with greedy search is that *it didn't take costs so far into account.*
  - We need a search algorithm (like A* search) that takes into consideration this cost so that it avoids expanding paths that are already expensive

Informed search- A*-search Algorithm

- It considers both *estimated cost of getting from n to the goal node* **h(n)**, and *cost of getting from initial node to node n*, **g(n)**
- Apply three functions over every nodes
  - g(n): Cost of path found so far from initial state to n
  - h(n): Estimated cost of shortest path from n to z
    - ***E.g. in Route finding*** *h(n) is straight line distance*
  - f(n): Estimated total cost of shortest path from a to z via n
  - Evaluation function f(n) = h(n) + g(n)

- **Implementation**: Expand the node for which the evaluation function f(n) is lowest
  - Rank nodes by f(n) that goes from the start node to goal node via given node

Cont.

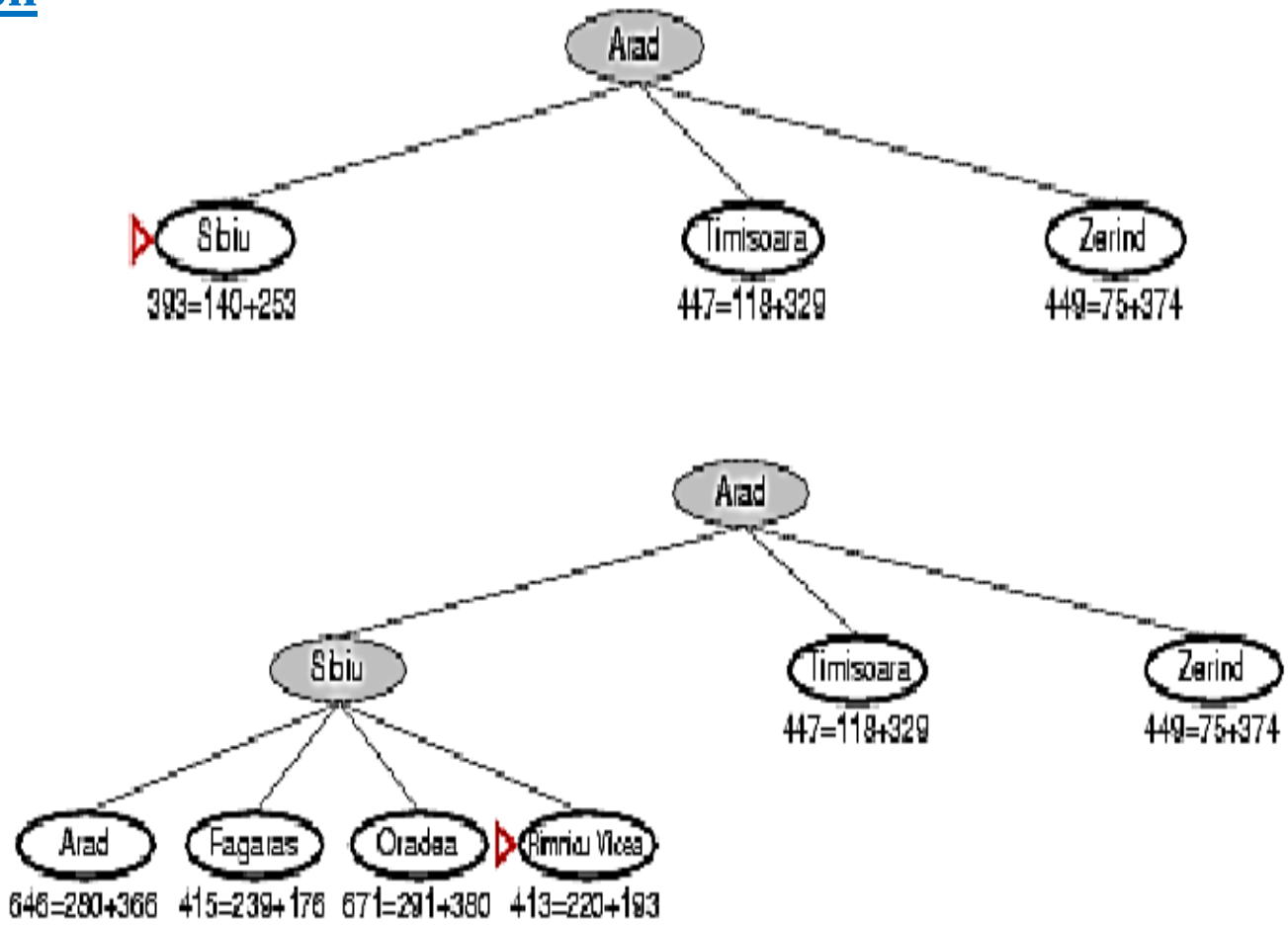


h(n)

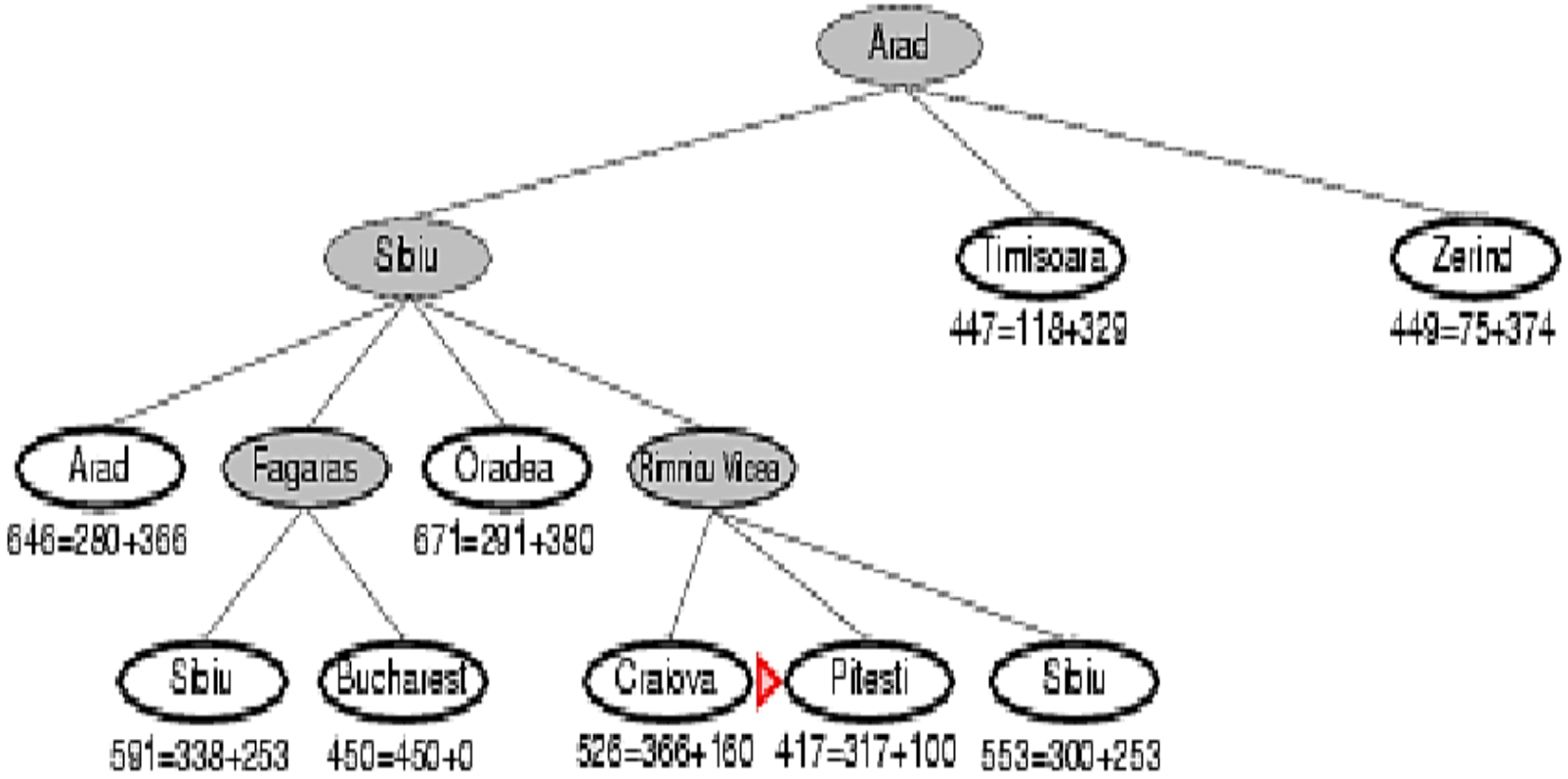
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Cont.

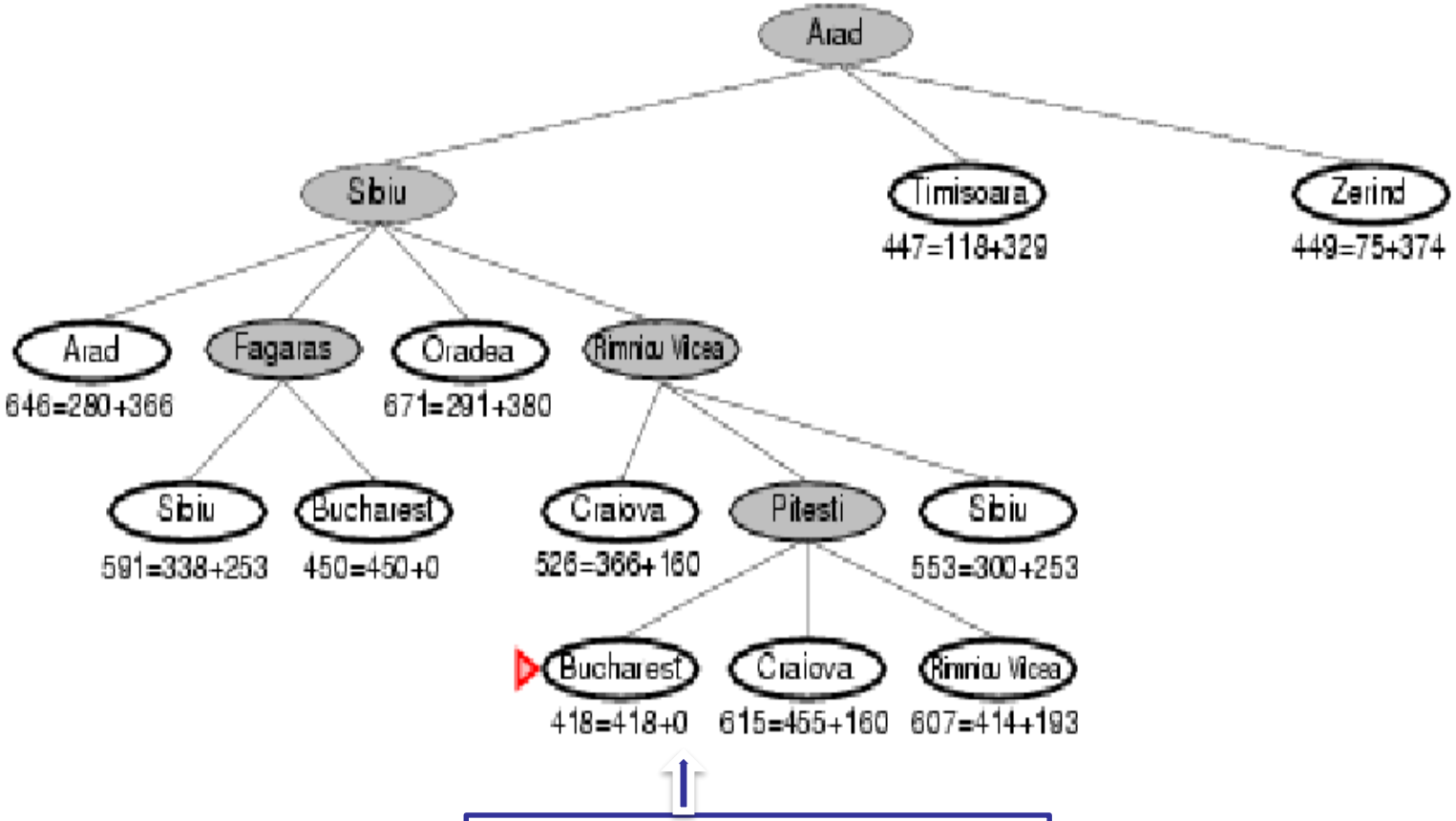
- **Solution**



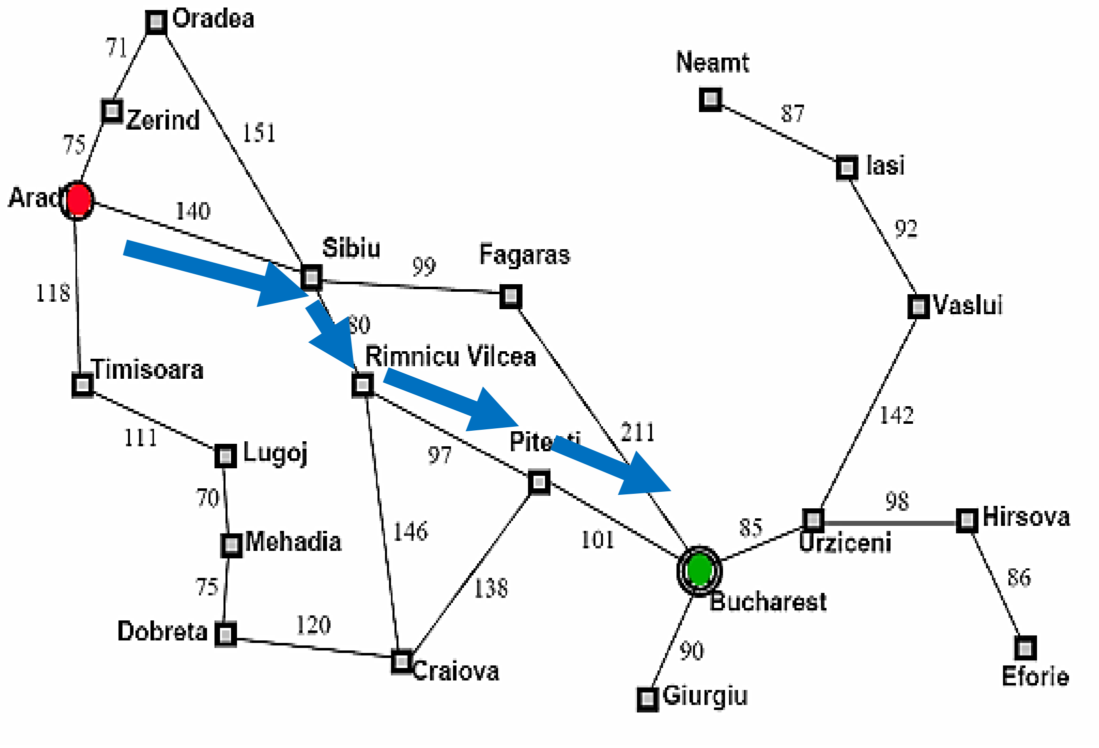
Cont.


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vicea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

Cont.


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193
h(n) = 0 it is the goal state

Cont.



Oradea
71
Neamt
Zerind
87
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
211
111
Lugoj
97
Pitesti
70
98
Hirsova
85
Urziceni
Mehadia
146
101
86
138
Bucharest
75
120
Dobreta
90
Craiova
Eforie
Giurgiu

Cont.

- **Properties**
  - **A* is complete**
    - It guarantees to reach to goal state
  - **A* is optimality**
    - If the given heuristic function is admissible, then A* search is optimal.
  - **Takes memory:**
    - It keeps all generated nodes in memory
    - Both time and space complexity is exponential

- Time and space complexity of heuristic algorithms depends on the quality of the heuristic function.

Assignment - 2

- Read and discuss each of the following searching algorithms using an example clearly.
  - ***AO* search***
  - ***Iterative improvement,***
  - ***Constraint satisfaction***
  - ***Problem reduction***
  - ***Hill climbing***
  - ***Generate and test***

Group Assignment (Due: 30 days)

Consider one of the following problem:

- Missionary-and-cannibal problem
- Towers of Hanoi
- Tic-Tac-Toe
- Travelling salesperson problem
- 8-queens problem
- 5 sticks
- Water Jug Problem(3:4; 4:5)
- 8-puzzle problem
- Monkey and Bananas problem
- Crypt arithmetic

1. Identify the set of states and operators; based on which construct the state space of the problem
2. write goal test function
3. Determine path cost
4. Implement the solution