AI Concepts and Techniques

Lesson 6: Explaining problems solving techniques in AI

Lesson	Date	Topics
1	18-Apr-22	Introduction to AI
2	21-Apr-22	Basics of Python
3	25-Apr-22	Linear Algebra + Python for AI (I)
4	28-Apr-22	Python for AI (II)
5	**4-May-22**	AI approaches
6	6-May-22	**AI concepts and techniques** **Agents** **AI & Problem Solving** **Search problems** **Uninformed search**
7	9-May-22	AI concepts and techniques Informed search Local search
8	12-May-22	AI concepts and techniques Constraint satisfaction problems Knowledge-based agents
9	17-May-22	Written Test
10	23-May-22	Future of AI, revision and review

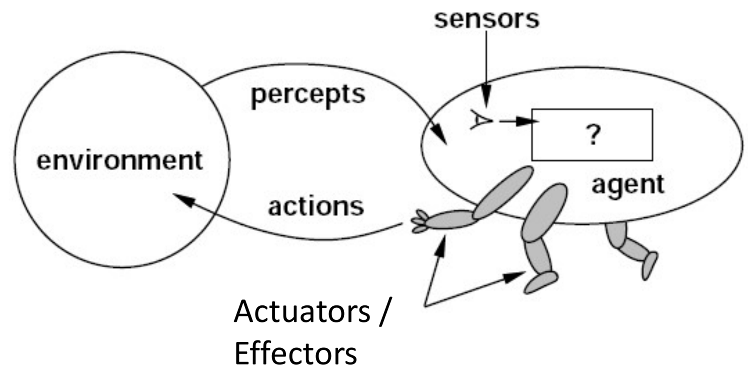
Search algorithms in AI

- Recall that AI is the study of **agents** that receive percepts from the environment and perform actions
- Generally, most of these agents are resolving some forms of search problems to achieve their tasks.

- A **search problem** consists of
  - Search space
  - Initial state
  - Goal state

- The search algorithm is the plan (sequence of actions) that transforms the initial state to the goal state
- This sequence of actions is the solution to the search problem

Agents



- **Perceives its environment through sensors**
- **Acts on that environment through actuators**

Example

Human agent:
- Sensors – eyes, ears etc.
- Actuators – hands, legs, voice etc.

Robotic agent:
- Sensors – cameras, IR etc.
- Actuators – motors etc.

Software agent:
- Sensors – keystrokes, file contents
- Actuators – displays, sent network packets etc.

Percept: Agent's perceptual input at any given time

Percept Sequence: log of all sensed inputs received by agent

Agent Function: (maps percept / percept history to actions)

$$f : \boldsymbol{P}^* \rightarrow A$$

Agent program: runs the physical architecture to produce the function.

Agent = architecture + program

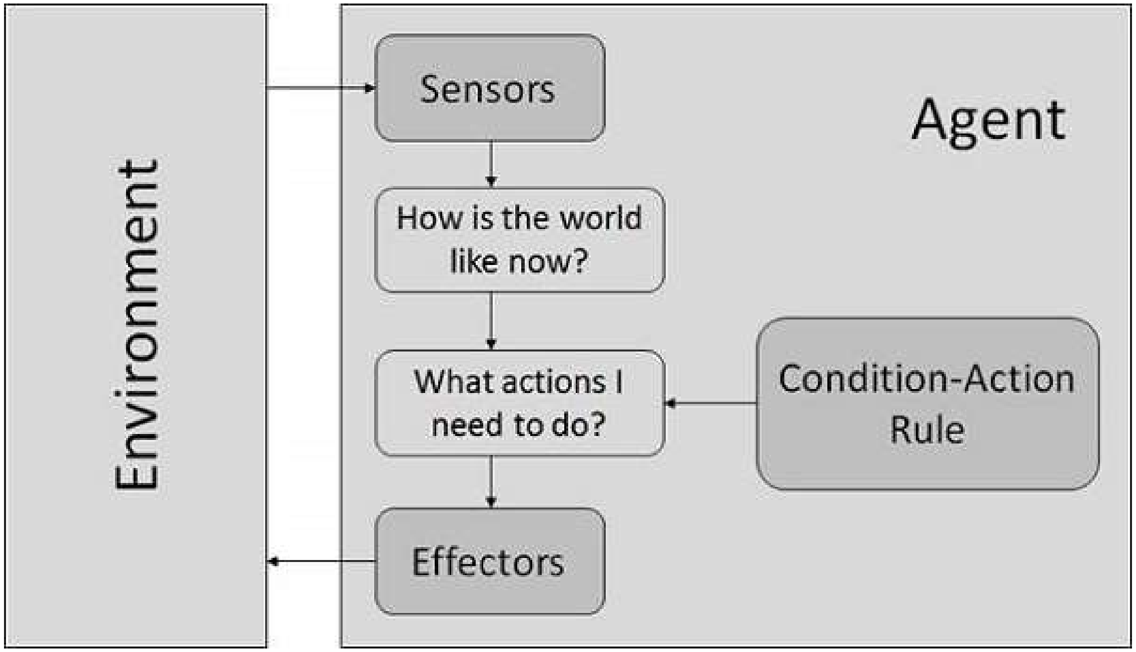
Some Types of Agents

- Simple Reflex Agents
- Model-Based Reflex Agents
- Goal-Based Agents
- Utility-Based Agents

Reflex Agents

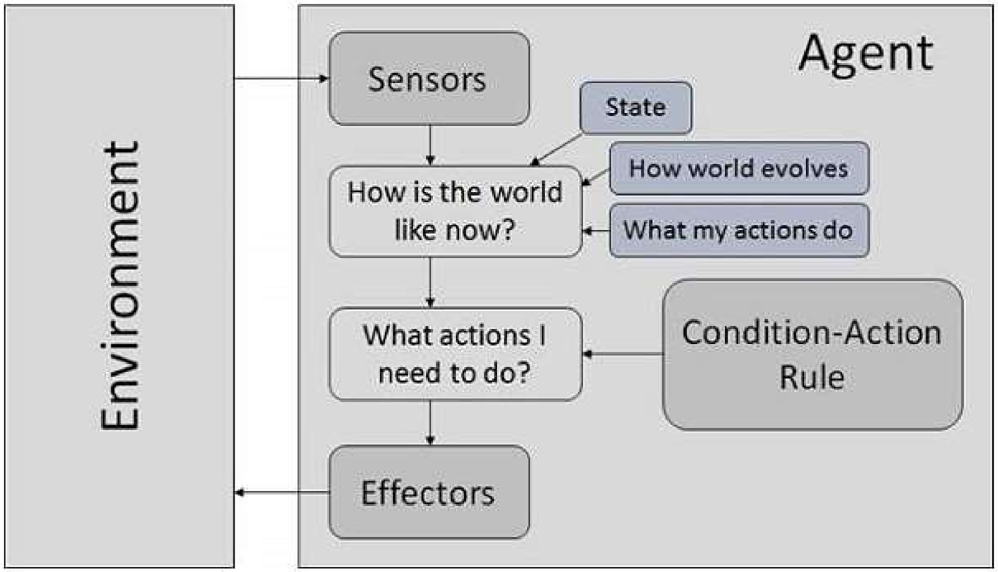
Simple Reflex Agent:

- **Chooses action based only on the current percept**
- No consideration for consequence of action



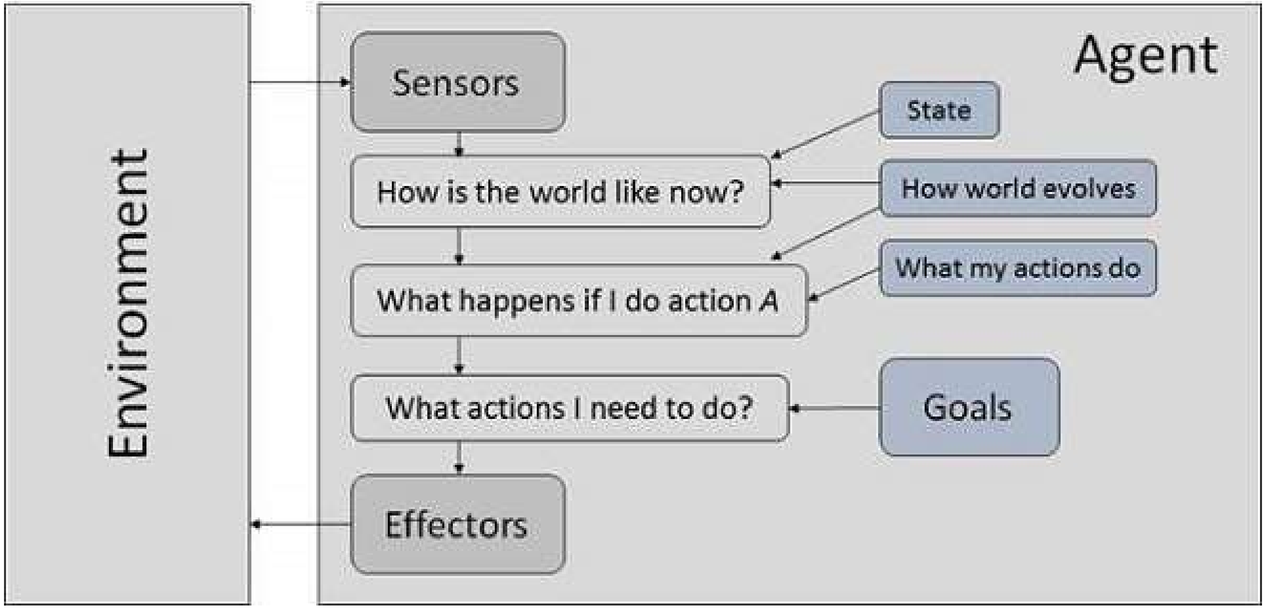
Model-based reflex agent

- Use model of world to choose action
- Maintain an internal state



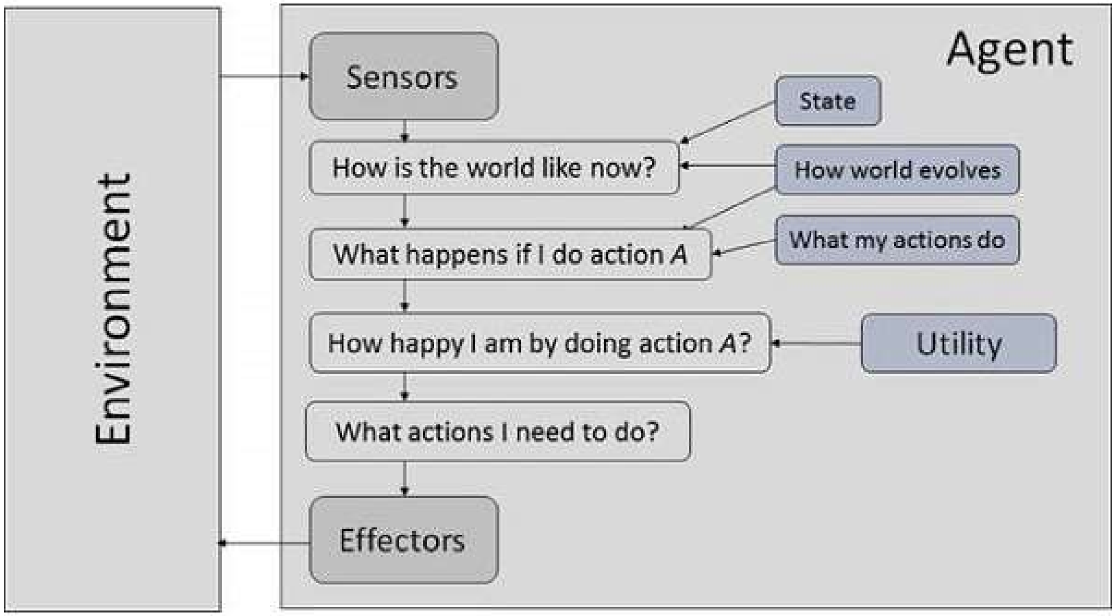
Goal-based agent

- Chooses actions in order to achieve goals.
- Goal is a description of a desirable situation
- Proactive rather than reactive
- Decisions are usually flexible (will change with progress)



Utility-based agent

- Chooses actions based on a preference (utility) for each state
- Goal are inadequate when there are conflicting goals,



PEAS

When designing an agent we must first specify the task environment using PEAS:

Performance Measure	Environment
Actuators	Sensors

PEAS

Example: Quality Control Robot

- P - % of parts in correct bin
- E – Conveyor belt, parts, bins
- A – Jointed arm, hand
- S - Camera

Environment Types

Real	Artificial
Fully Observable	Partially Observable
Discrete	Continuous

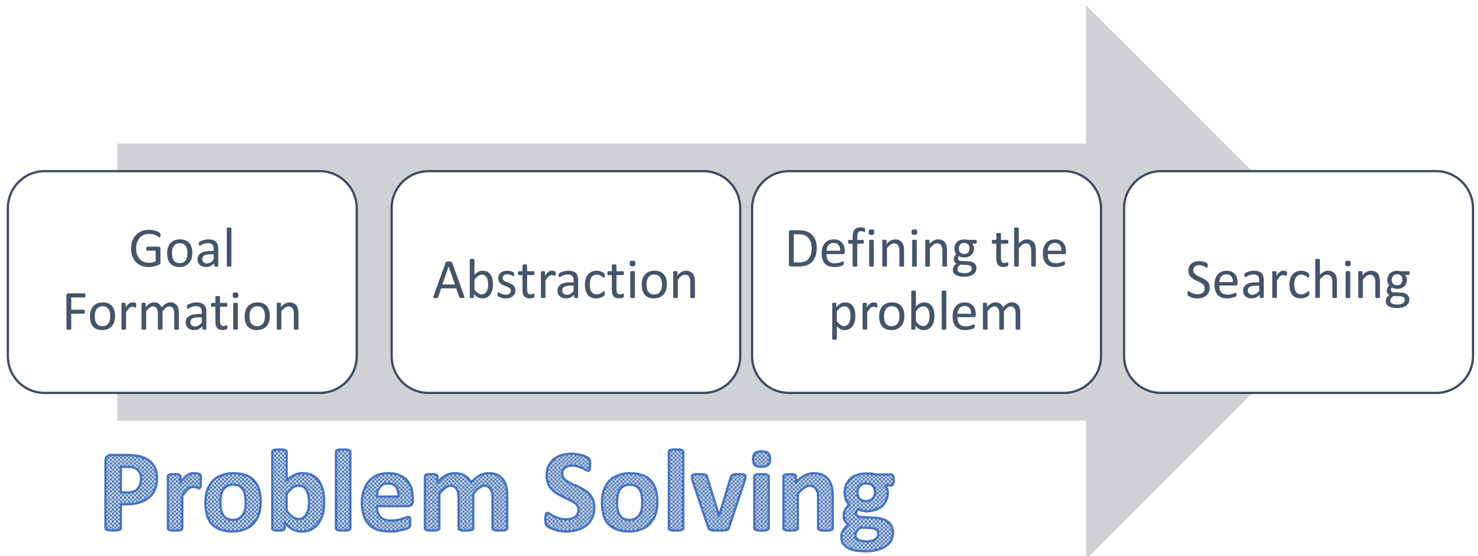
Environment Types

Deterministic	Stochastic
Episodic	Sequential
Static	Dynamic
Single agent	Multi agent

Problem solving in AI

AI & Problem Solving

A **problem solving agent** is a **goal-based agent** that develops **solutions** by finding sequences of actions that lead to desirable states.



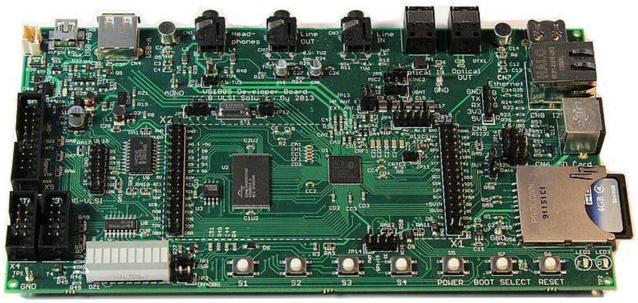
AI & Problem Solving – Goal Formation

Goal Formation is the first step in problem solving and is based on the current situation and the agent's performance measure.

Example: Designing a circuit board

- Components
- Wires
- Connection routes

Performance measure - minimum area possible.



AI & Problem Solving - Abstraction

Abstraction involves removing as much detail as possible in order to create a workable description without invalidating the problem.

Example: determining cleaning route for vacuum robot.

AI & Problem Solving – Defining the problem

- When representation a problem, we want the representation to be rich enough to express the knowledge needed to solve the problem
- As close to the problem as possible: compact, natural and maintainable
- Amenable to efficient computation
  - Able to express features of the problems that can be exploited for computation gain
  - Able to trade off accuracy and computation time and/ or space

AI & Problem Solving – Defining the Problem

Defining the Problem:

- **Initial State**
- **Possible Actions**
- **Transition Model**
- **Goal Test**
- **Path Cost**

Example: Finding the best route from Dhoby Ghaut to Bayfront.

AI & Problem Solving – Defining the Problem

Defining the Problem:

- **Initial State** — *In(Dhoby Ghaut)*
- **Possible Actions** — *Go(Somerset), Go(Little India), Go(City Hall), Go(Bras Basah), Go(Clarke Quay)*
- **Transition Model** — Result *In(Dhoby Ghaut), Go(Bras Basah) = In(Bras Basah)*
- **Goal Test** — Is current state the goal state?
- **Path Cost** — time taken, distance, fare cost etc.

Example: Finding the best route from Dhoby Ghaut to Bayfront.

AI & Problem Solving – Defining the Problem

- Water Jug Problem

  Have a jug of 3-litres and a jug of 4-litres. Need to fill 2 litres of water in 4-litre jug

Defining the Problem:

- Initial State (0, 0)
- Possible Actions Fill jug, empty jugs, any others?
- Transition Model (0,0),Fill 3 litre jug => (3, 0)
- Goal Test (0, 2)
- Path Cost no. of moves

Exercise – Classic AI Search Problems

Some classic AI problems include:

- **8 – Puzzle**
- **8 Queens**
- **Vacuum world**
- **Touring problem**

15 -20
minutes

Choose one of the following problems and complete the following tasks:

- Explain the goal of the problem (goal formation)
- Select an appropriate performance measure
- Define the task environment
- Define the problem
  - Initial state
  - Possible actions (from initial state)
  - Transition model
  - Select a suitable goal test
  - Decide an appropriate path cost

Search Problem

Search problem

- A search problem consists of
  - Search space
  - Initial state
  - Goal state

Search space

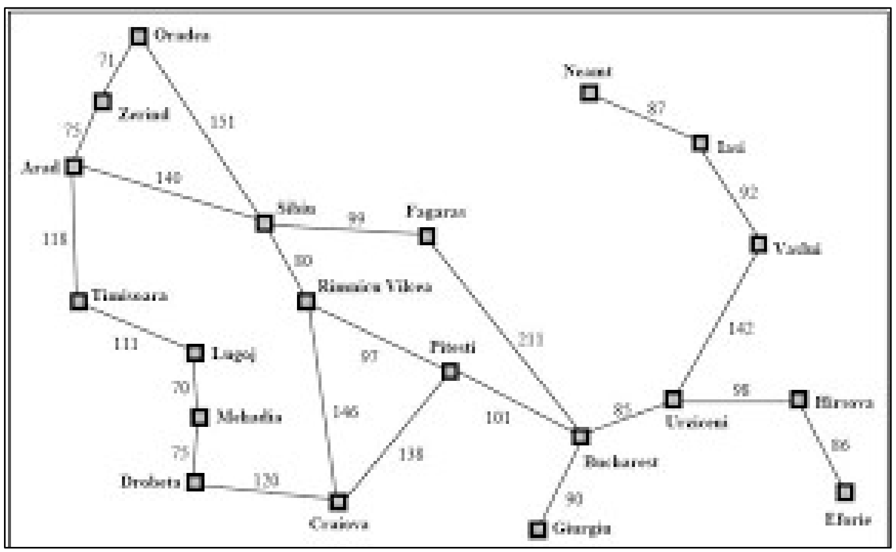
- The search space contains all the feasible solutions
- When trying to solve a problem, we are looking for the desired solution in the search space
- Also called the state space

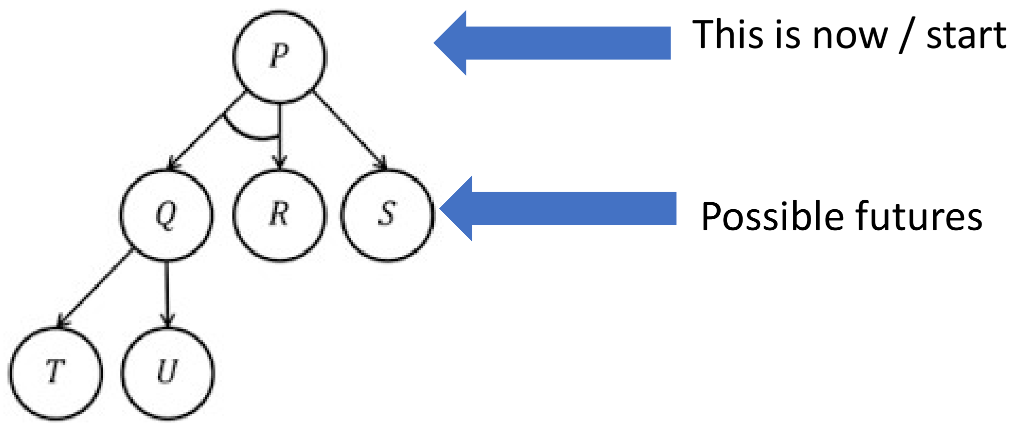
General Search Algorithm

- Start with initial state
- Repeatedly expand a state by generating its successors
- Stop when goal state is expanded, or all reachable states have been considered

Search Problems – Definitions

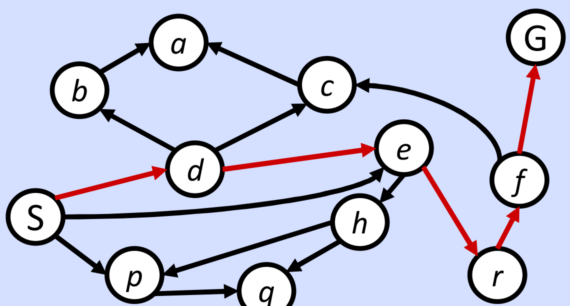
Search Structure: Graph Search vs Tree Search





State Space Graphs vs. Search Trees

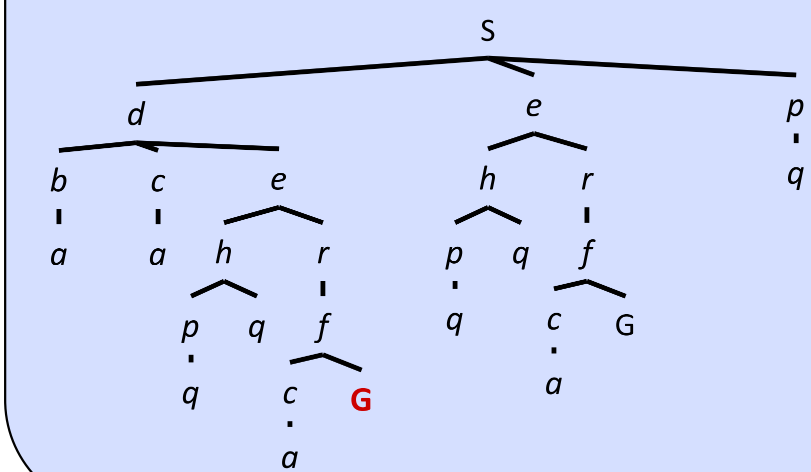
State Space Graph



Each NODE in in the search tree is an entire PATH in the state space graph.

Construct both on demand – and construct as little as possible.

Search Tree



State Space Graphs

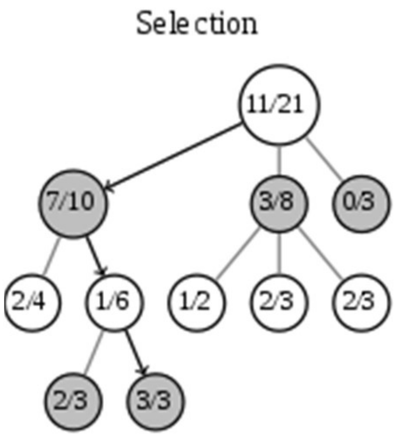
- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)
- In a search graph, each state occurs only once!
- Rarely build this full graph in memory

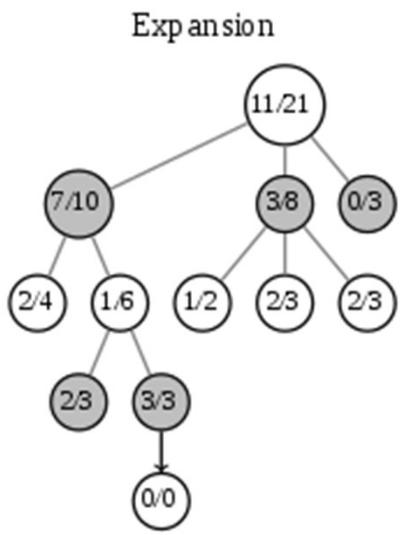
Search Trees

- A search tree:
  - A “what if” tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - Nodes show states, but correspond to PLANS that achieve those states

Search Problems - Definitions

Nodes





Parent
Child
Leaf
Frontier

Path cost
Depth

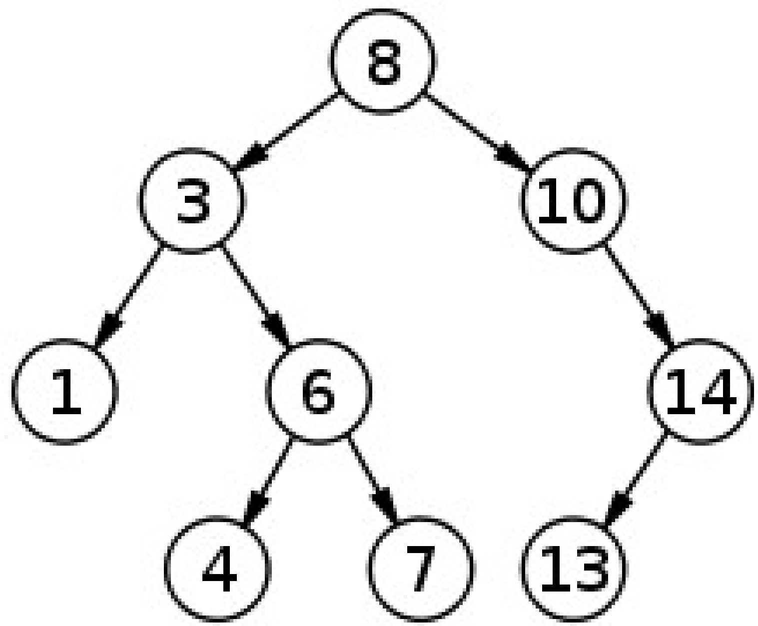
Search Problems - Queues

Queueing systems:

LIFO

FIFO

Priority



Search Problems – Data Structure

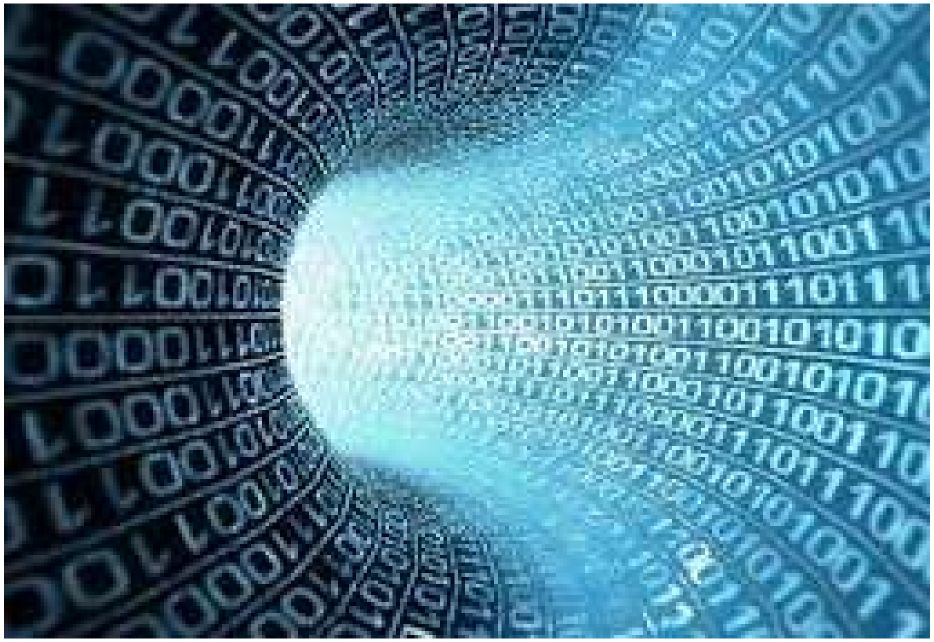
Search algorithms require a data structure to keep track of the search tree that is being constructed. For each node n of the tree, we have a structure that contains four components:

- **STATE:** the state in the state space to which the node corresponds;
- **PARENT**: the node in the search tree that generated this node;
- **ACTION**: the action that was applied to the parent to generate the node;
- **PATH-COST**: the cost, traditionally denoted by g(n), of the path from the initial state to the node, as indicated by the parent pointers.

Search Problems – Measuring Performance

It is important to measure the performance of a search method. Performance can be measured with respect to different considerations:

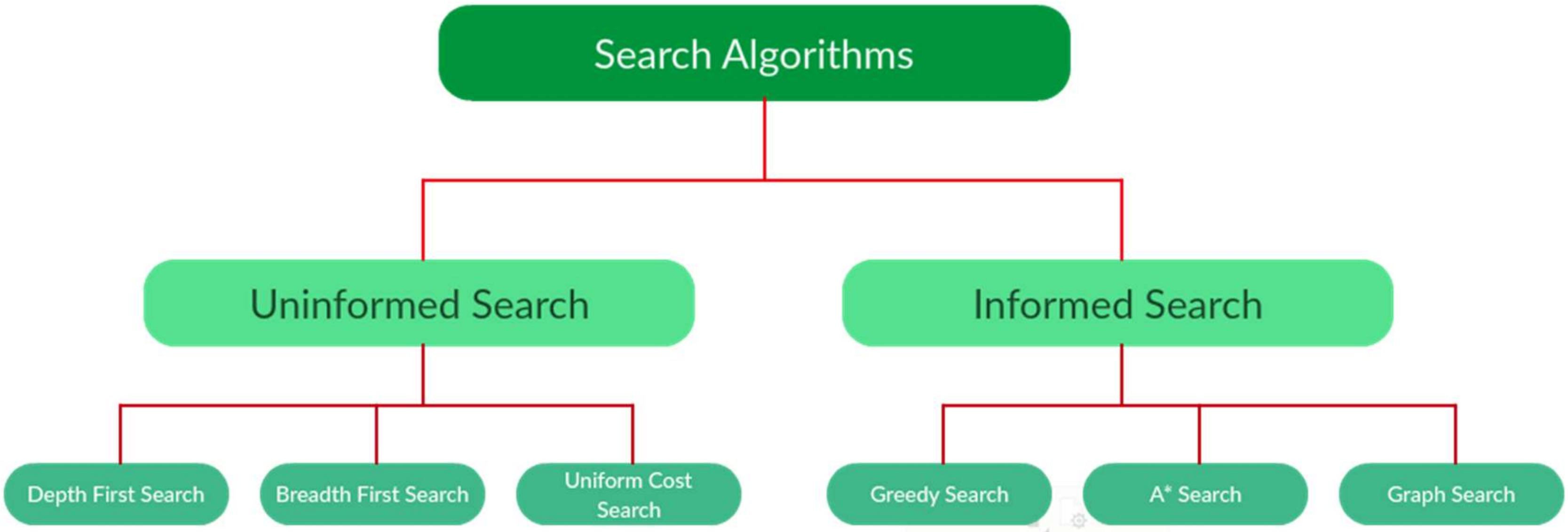
- **Completeness – Ability to always find a solution.**
- **Optimality – Will find the least-cost solution.**
- **Time Complexity – Time needed.**
- **Space Complexity – Storage space needed for generated nodes.**



Uninformed Search

Search strategies

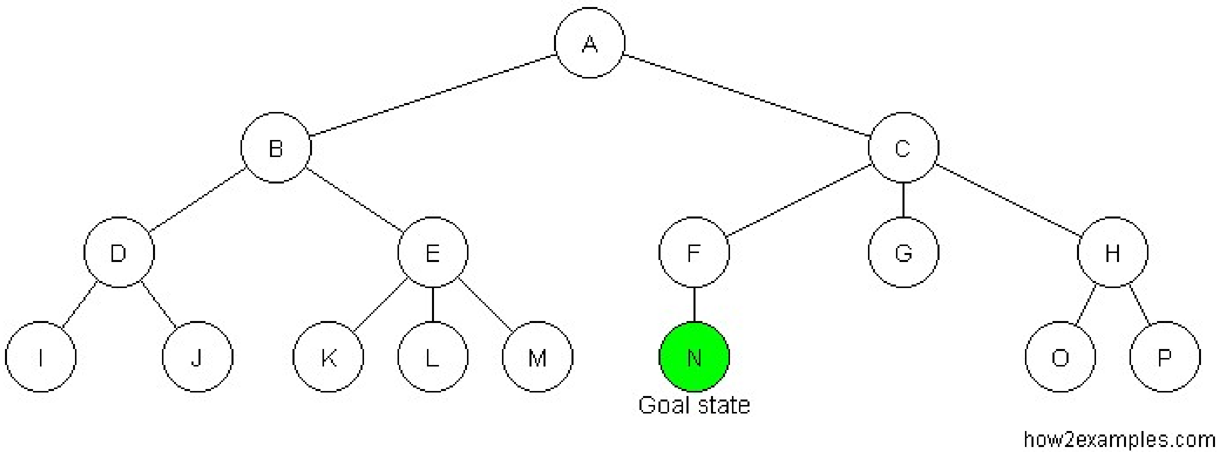
Search strategies can be informed or uninformed



Uninformed Search Strategies

There are several different strategies for searching:

- **Breadth first (FIFO)**
- **Depth First (LIFO)**
- **Depth Limited**
- **Iterative Deepening**
- **Bidirectional Search**
- **Uniform Cost (Priority)**



Breadth first

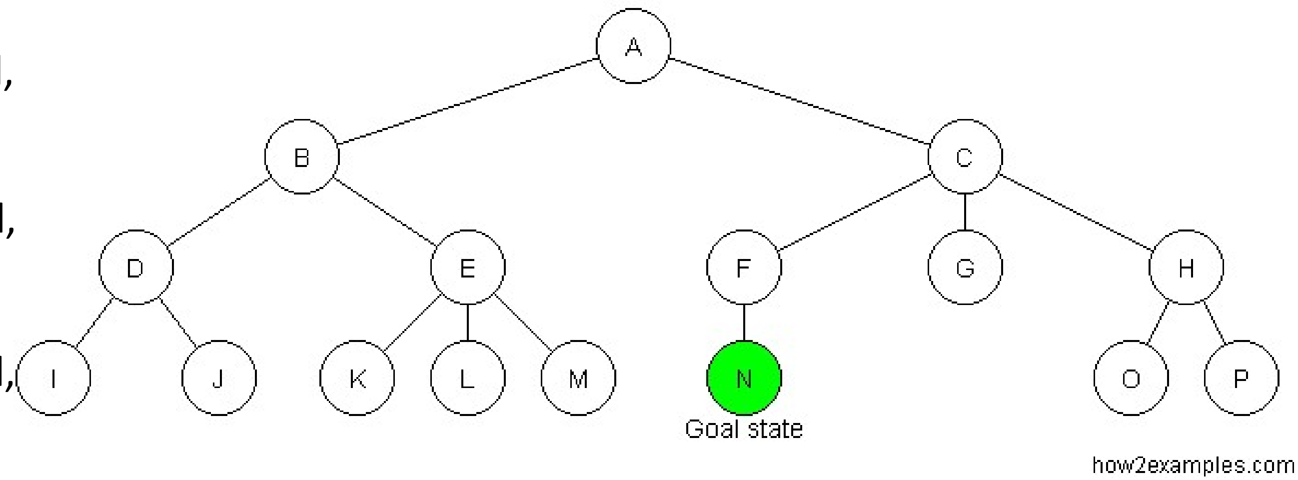
- Check initial state if goal state node is reached
- Set the initial node in the “frontier” queue (FIFO)
- Initialize “explored” set as empty
- Loop
  - If frontier queue empty → unable to find solution
  - Get first node in frontier queue
  - Set node in explored set
  - Iterate through child nodes
    - If childe node is not explored or in frontier
      - If goal state reached, return child node
      - Else add child node to frontier

Uninformed Search Strategies - Breadth first

- Expand shallowest unexpanded node
- FIFO queue. As nodes are expanded, the new successors go at the end.
- When A is visited and expanded,
  → Frontier queue = [B,C]
- When B is visited and expanded,
  → Frontier queue = [C, D, E]
- When C is visited and expanded,
  → Queue = [D, E, F, G, H]

D → Queue = [E, F, G, H, I, J]
E → Queue = [F, G, H, I, J, K, L, M]
F → N generated and pass goal test

Algorithm returns when N is generated and it is the goal



Breadth first

Uninformed Search Strategies - Depth first

- Expand deepest unexpanded node
- LIFO (stack). As nodes are being visited, the new successors go to the front of the queue/top of the stack.
- If A is not visited, add it to the visited list and for each of its unvisited neighbours/successors, add them to the front of the queue/top of the stack,

  → stack = [B, C]
- When B is visited

  → stack = [D, E, C]
- When D is visited and expanded,

  → stack = [I, J, E, C]

I → stack = [J, E, C]

J → stack = [E, C]

E → stack = [K, L, M, C]

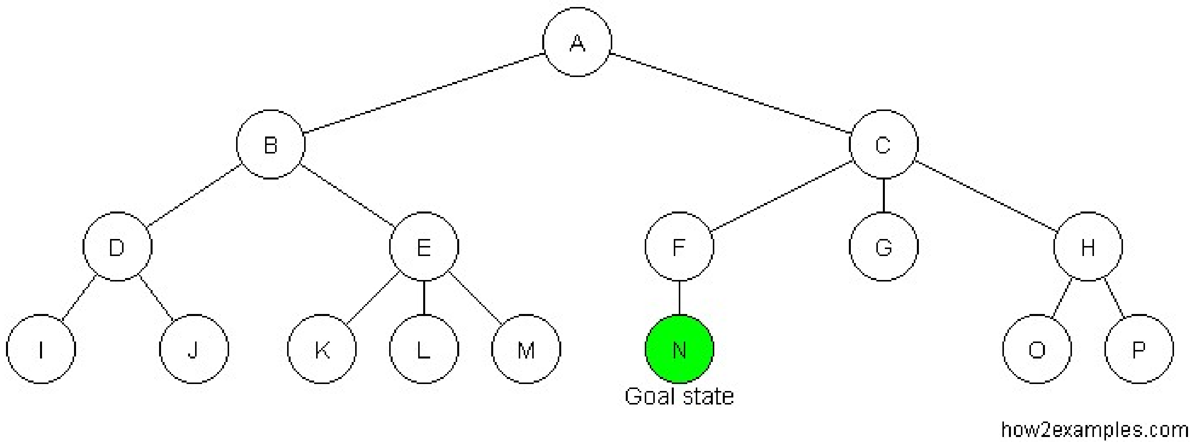
K → stack = [L, M, C]

L → stack = [M, C]

M → stack = [C]

C → stack = [F, G, H]

F → stack = [N, G, H]



Algorithm returns when N is visited and it is the goal

Uninformed Search Strategies

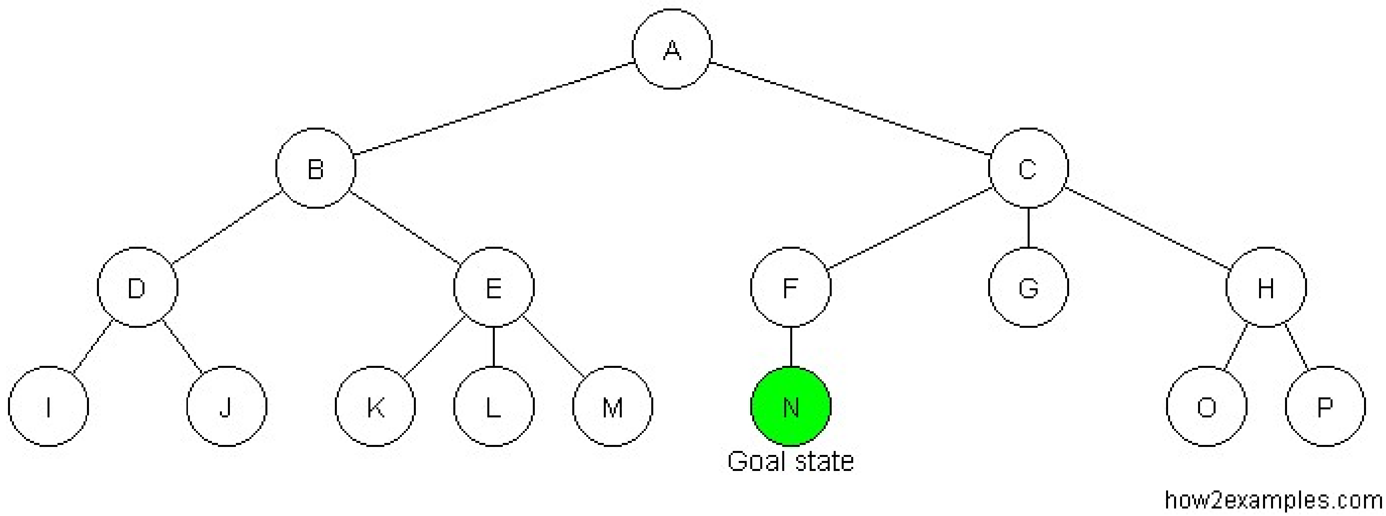
There are several different strategies for searching:

- **Depth limiting**

  e.g. depth level = 2

  Sequence of visits

  1. A
  2. B
  3. D
  4. E
  5. C
  6. F
  7. G
  8. H

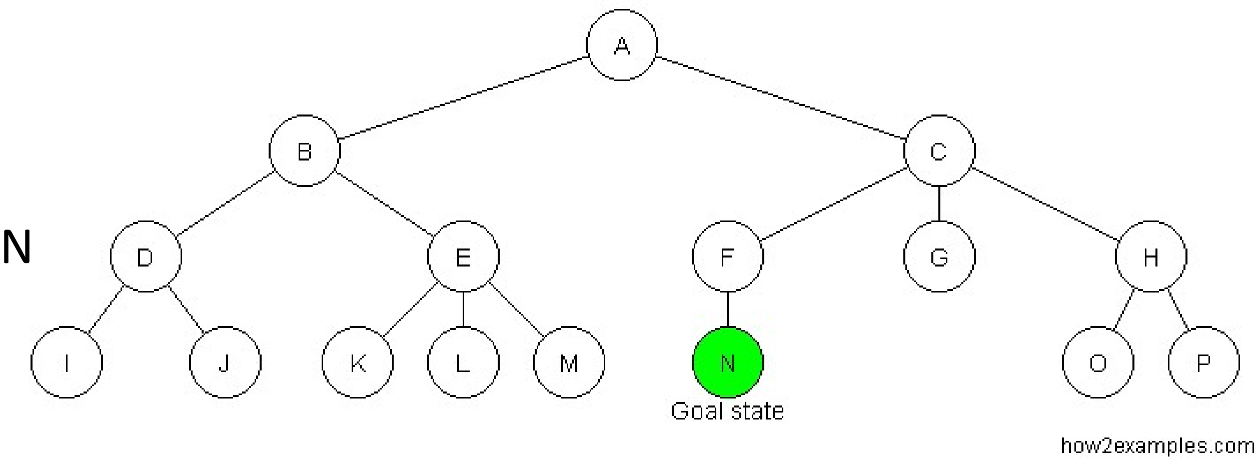


Uninformed Search Strategies

- **There are several different strategies for searching:**
- **Iterative Deepening**

Sequence of nodes visited

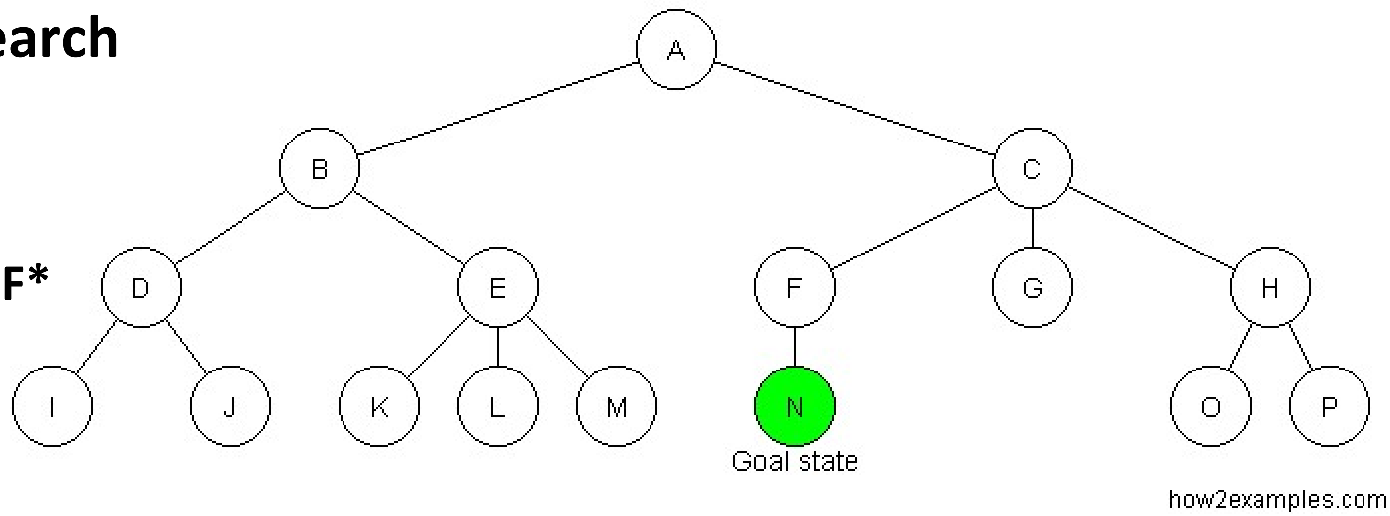
1. Depth 0: A
2. Depth 1: ABC
3. Depth 2: ABDE CFGH
4. Depth 3: ABDIJ EKLM CFN



Uninformed Search Strategies

There are several different strategies for searching:

- **Bidirectional Search**
- Paths founds
  1. **AB, AC, NF**
  2. **ABD, ABE, ACF***



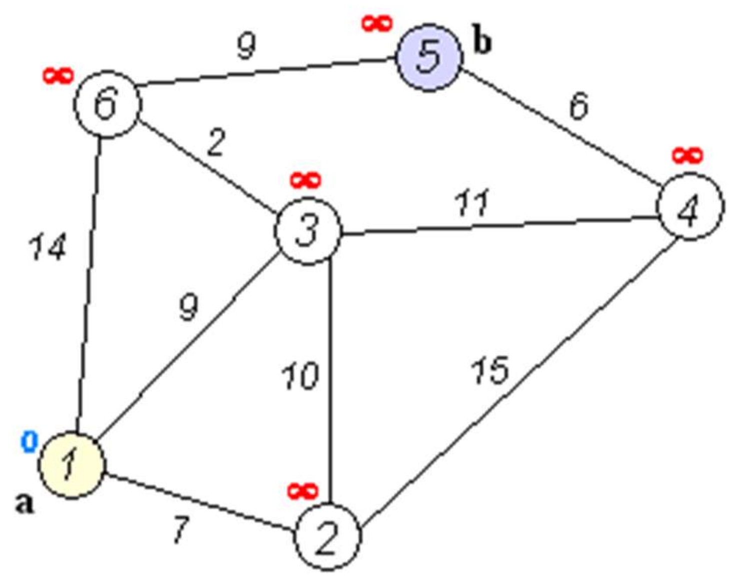
Dijkstra's algorithm

- Find the shortest path between nodes in a graph, which may represent for example, a road network.

Dijkstra's algorithm to find the shortest (distance) path between *a* and *b*.

It picks the unvisited vertex with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark visited (set to red) when done with neighbors.

https://www.youtube.com/watch?v=pVfj6mxhdMw

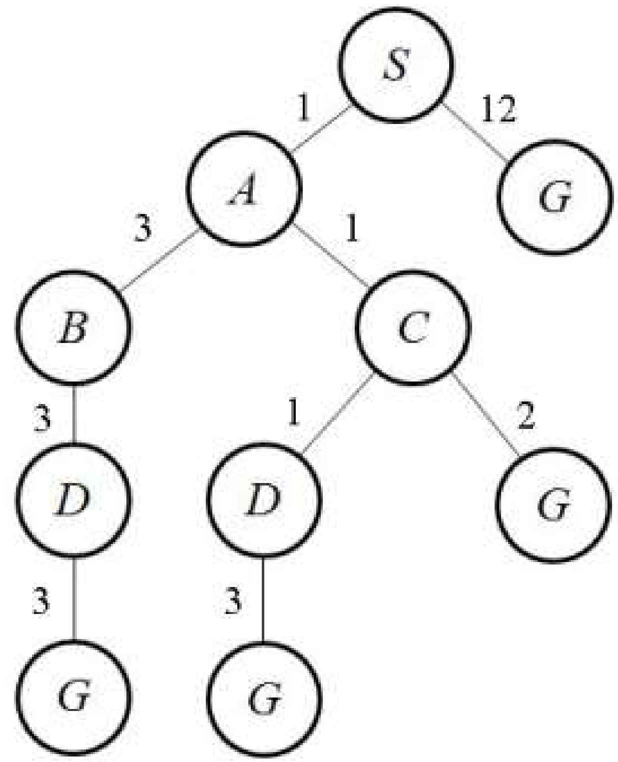


Uninformed Search Strategies

Uniform Cost (Priority)

- **Incorporates step costs**
- **Expand the least-cost unexpanded note**
- **Similar to Dijkstra algorithm, but with goal test**
- **Equivalent to breadth-first search if steps costs are equal**

S is the initial state
G is the end state

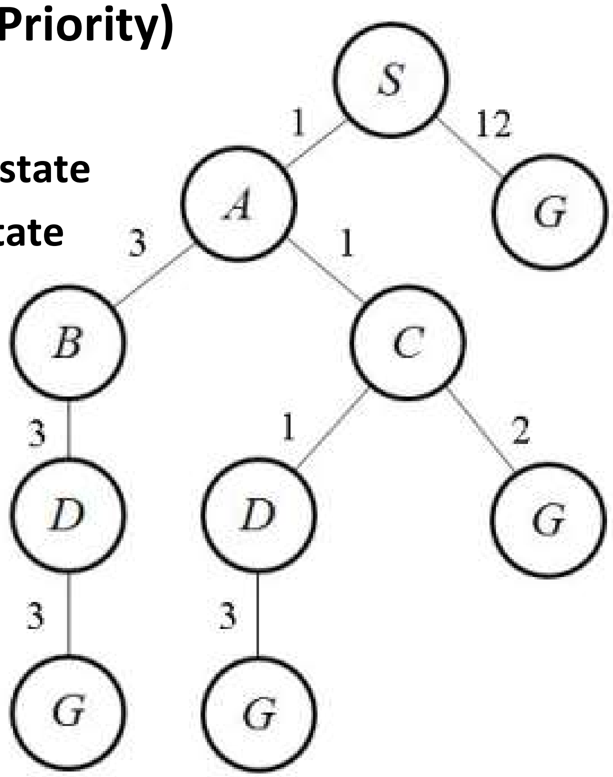


Uninformed Search Strategies

Uniform Cost (Priority)

S is the initial state

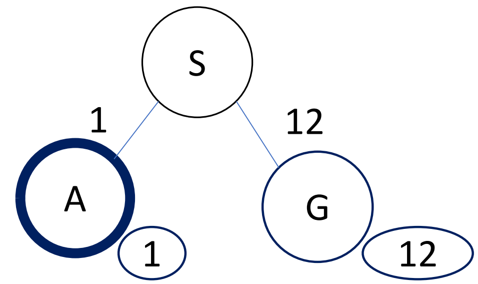
G is the end state



Visited Nodes

1. **S,**

2. S**A,**

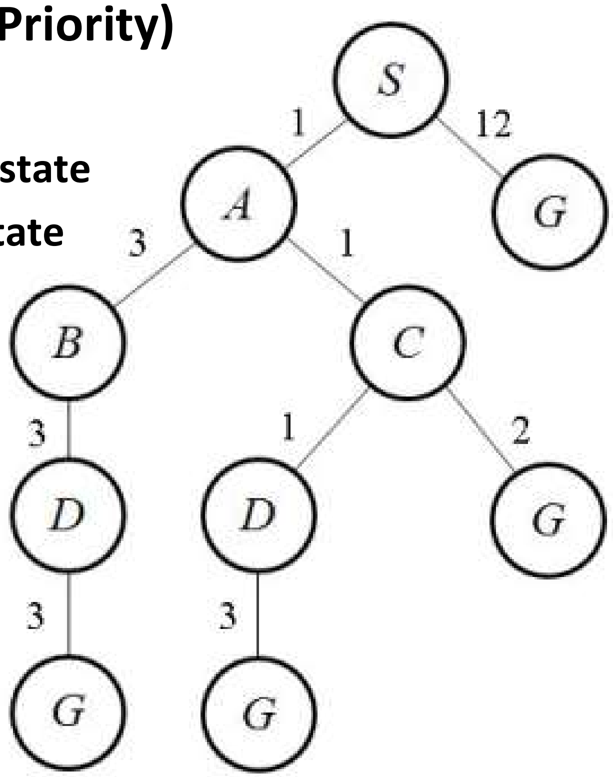


Uninformed Search Strategies

Uniform Cost (Priority)

S is the initial state

G is the end state

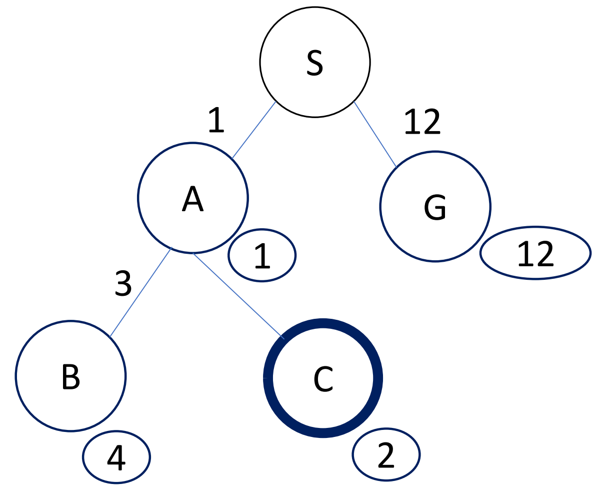


Visited nodes

1. **S**

2. S**A**

3. SA**C**

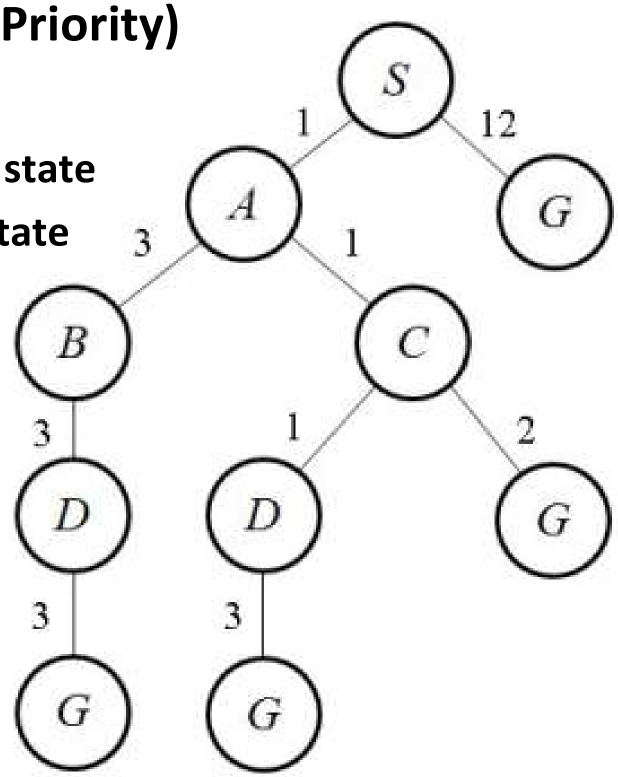


Uninformed Search Strategies

Uniform Cost (Priority)

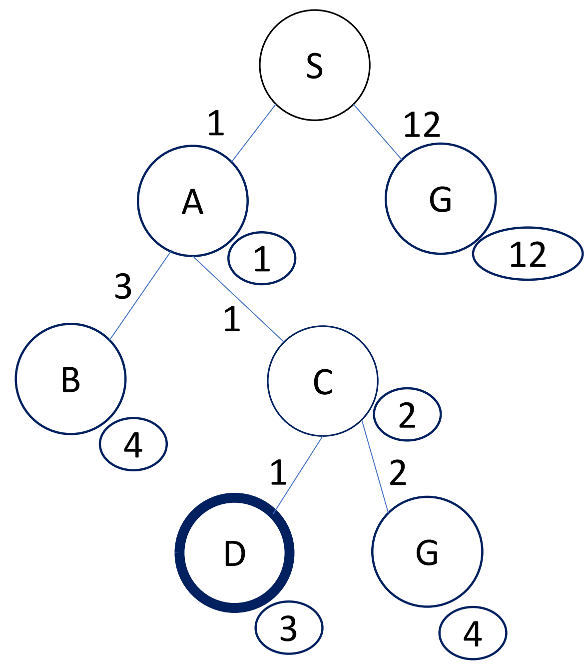
S is the initial state

G is the end state



Visited nodes

1. **S**
2. S**A**
3. SA**C**
4. SAC**D**

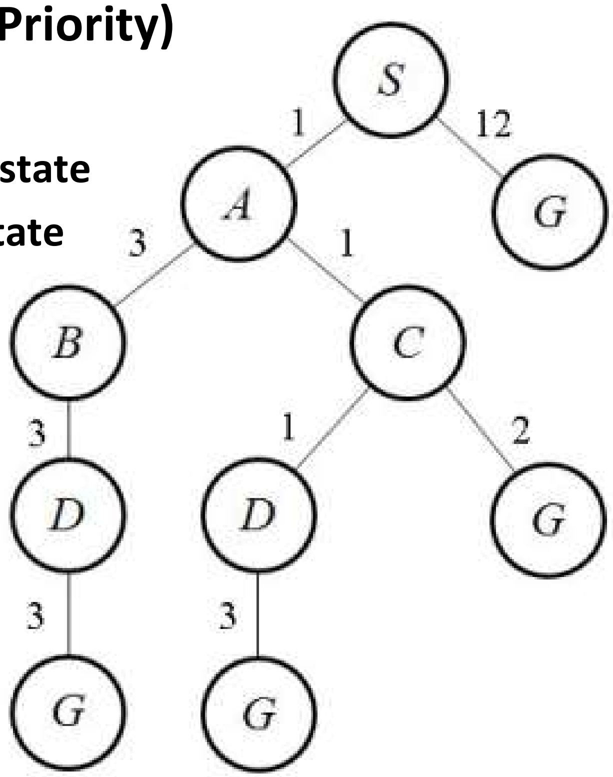


Uninformed Search Strategies

Uniform Cost (Priority)

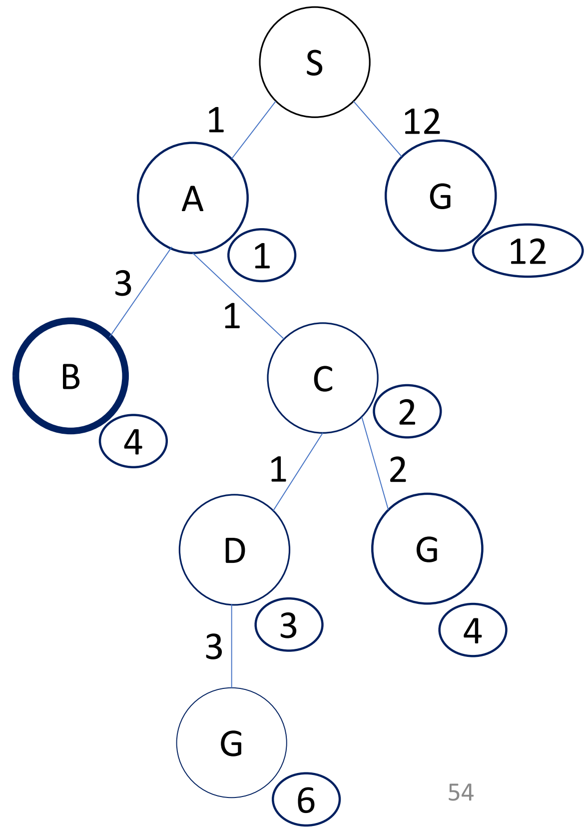
S is the initial state

G is the end state



Visited nodes

1. **S**
2. S**A**
3. SA**C**
4. SAC**D**
5. SA**B**

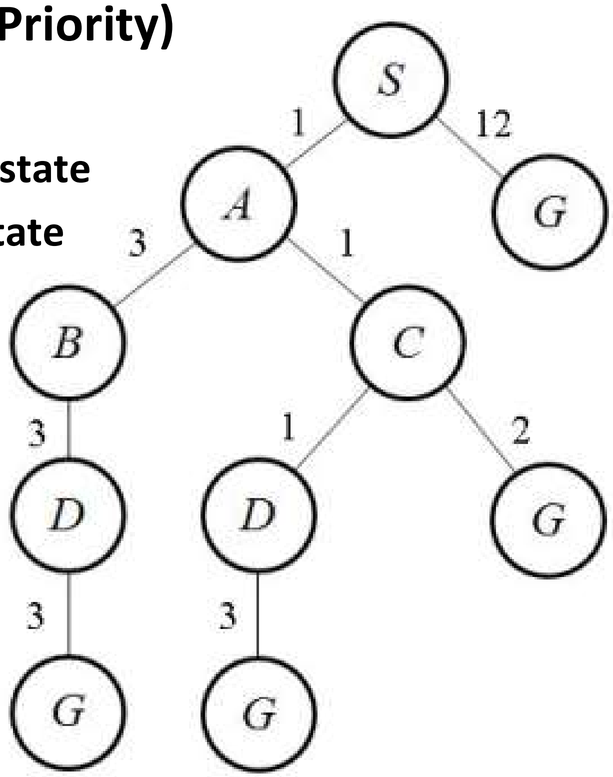


Uninformed Search Strategies

Uniform Cost (Priority)

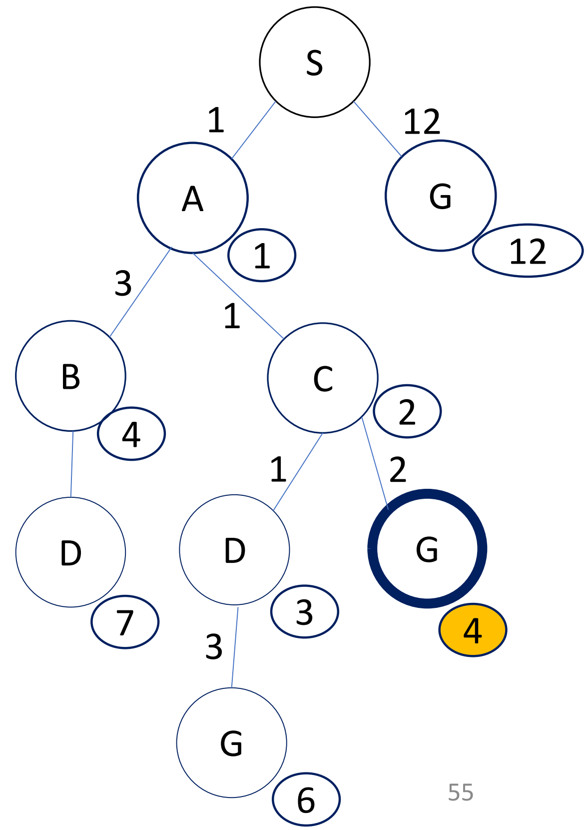
S is the initial state

G is the end state



Visited nodes

1. S,
2. SA,
3. SAC
4. SACD
5. SAB



Summary

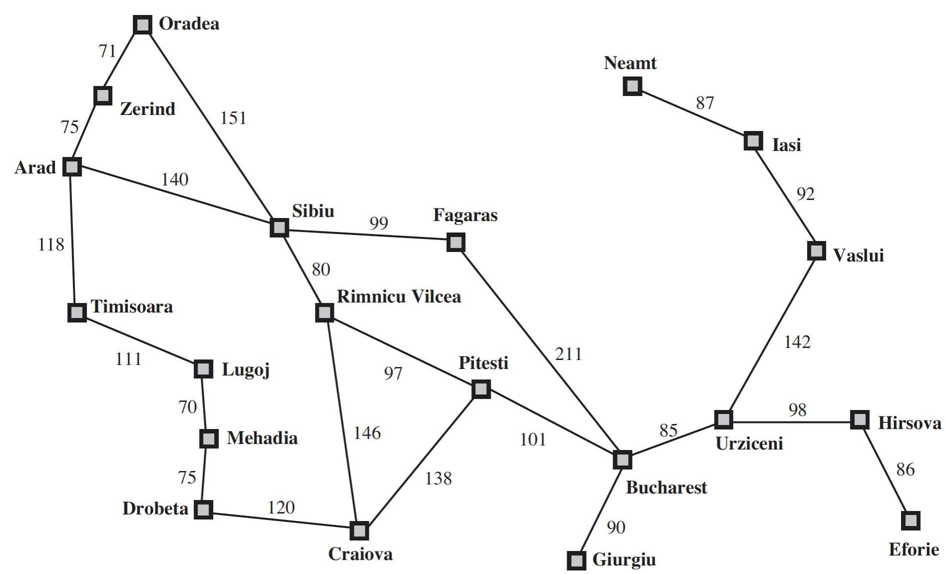
Topics	
Agents AI & problem solving Search problems Uninformed search	
Informed search Local search	Session 7
Constraint satisfaction problems Knowledge-based agents	Session 8

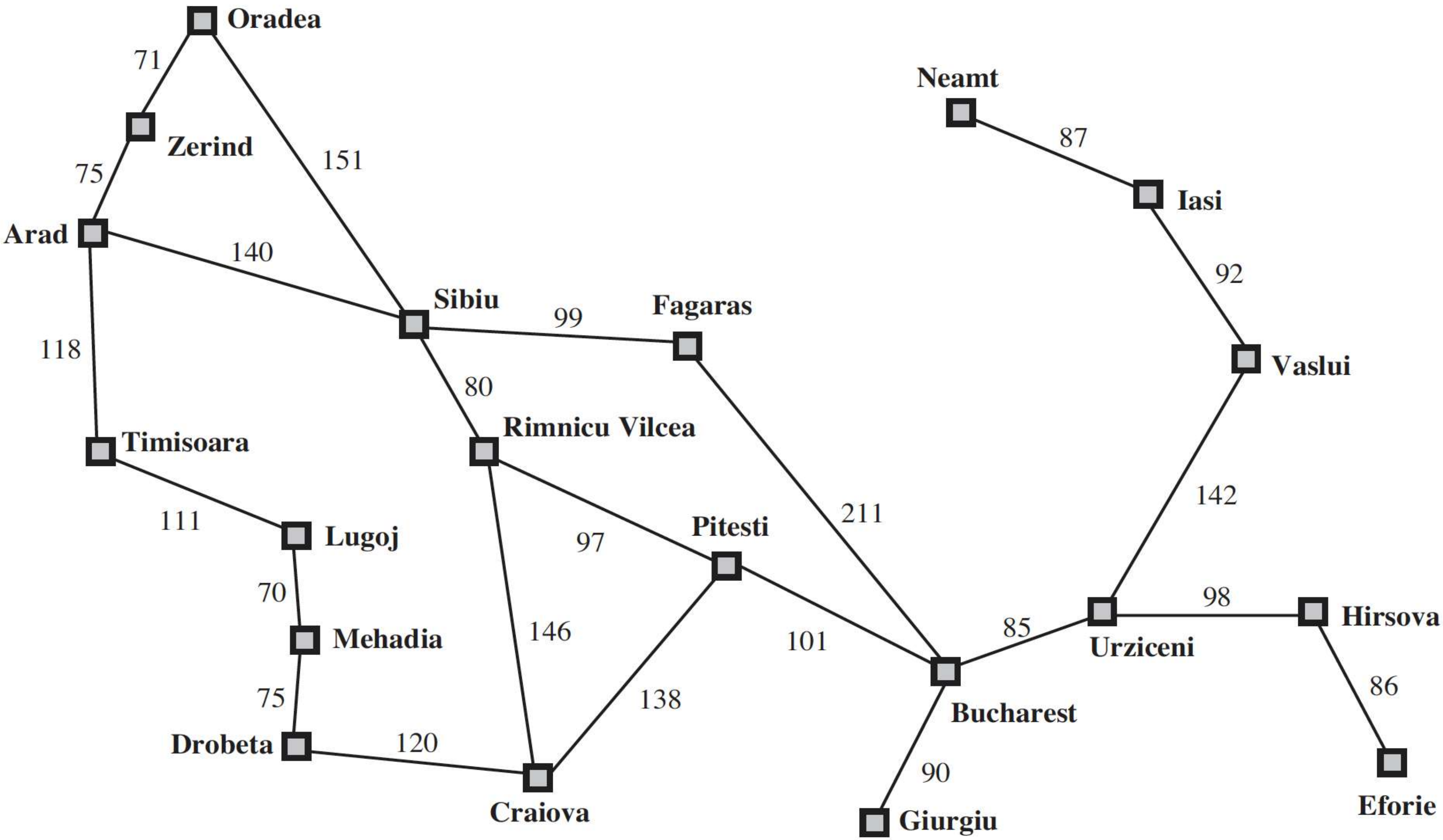
Exercise

Goal: To get to Bucharest from Arad

Based on the map given, describe/show how the nodes are expanded using Breadth-first and Uniform-cost search strategy.

Discuss if the strategies are naturally complete and/or naturally optimal.





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