Problems Solving Agents Searching and Algorithms

HABIB ULLAH QAMAR

MSCS (SE) MBA-HRM

Agent and Problems

- An **agent** is anything that can be viewed as perceiving its **environment** through **sensors** and acting upon that environment through **actuators**.
- *An agent's choice of action at any given instant can depend on the entire percept sequence observed to date, but not on anything it hasn't perceived*
- We use the term **percept** to refer to the agent's perceptual inputs at any given instant. An agent's **percept sequence** is the complete history of everything the agent has ever perceived

Agent and Problems

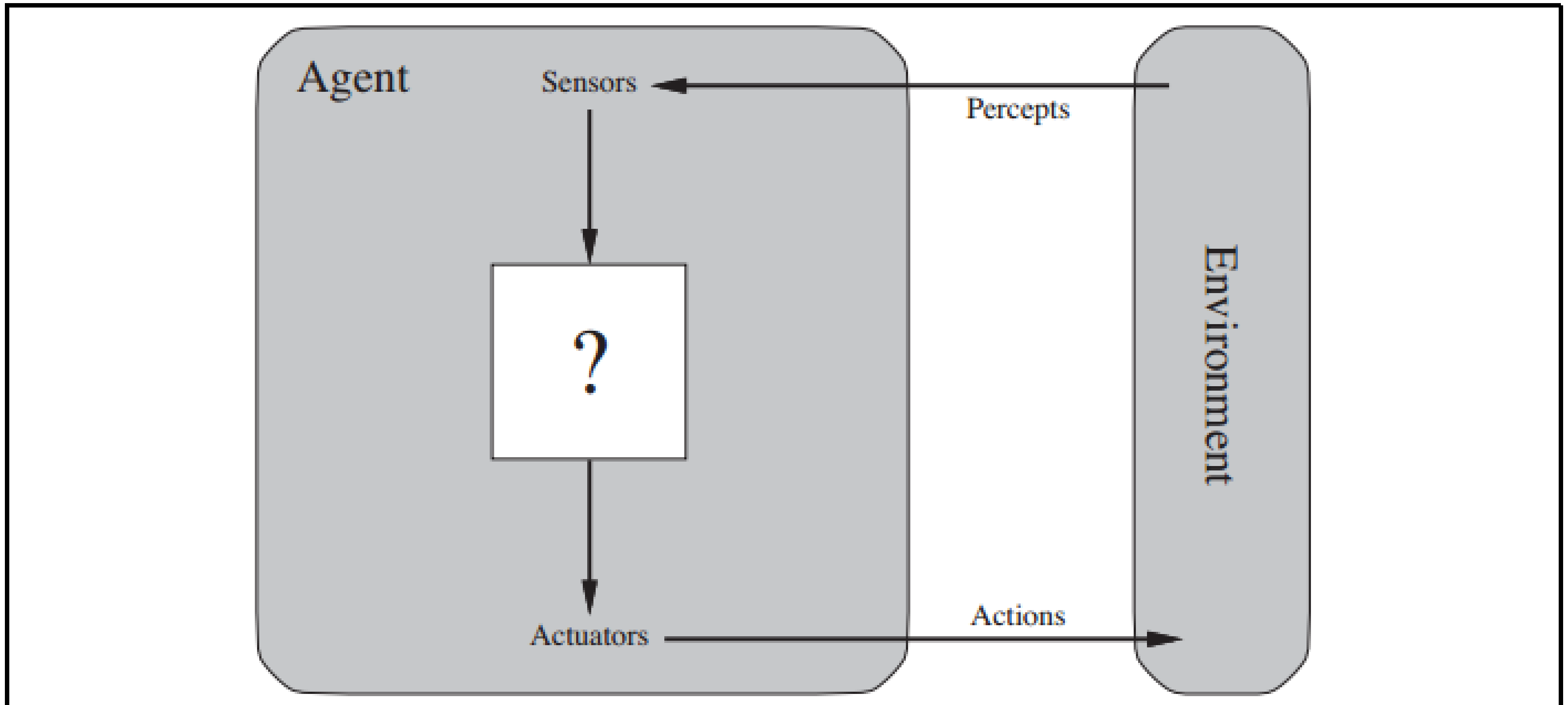


Figure 2.1 Agents interact with environments through sensors and actuators.

Agent and Problems

- An agent's behavior is described by the **agent function** that maps any given **percept sequence to an action.**
- *T*he agent function for an artificial agent will be implemented by an **agent program**. It is important to keep these two ideas distinct.
- The agent function is an **abstract mathematical description**;
- the agent program is a **concrete implementation**, running within some physical system.

An Example of Seraching

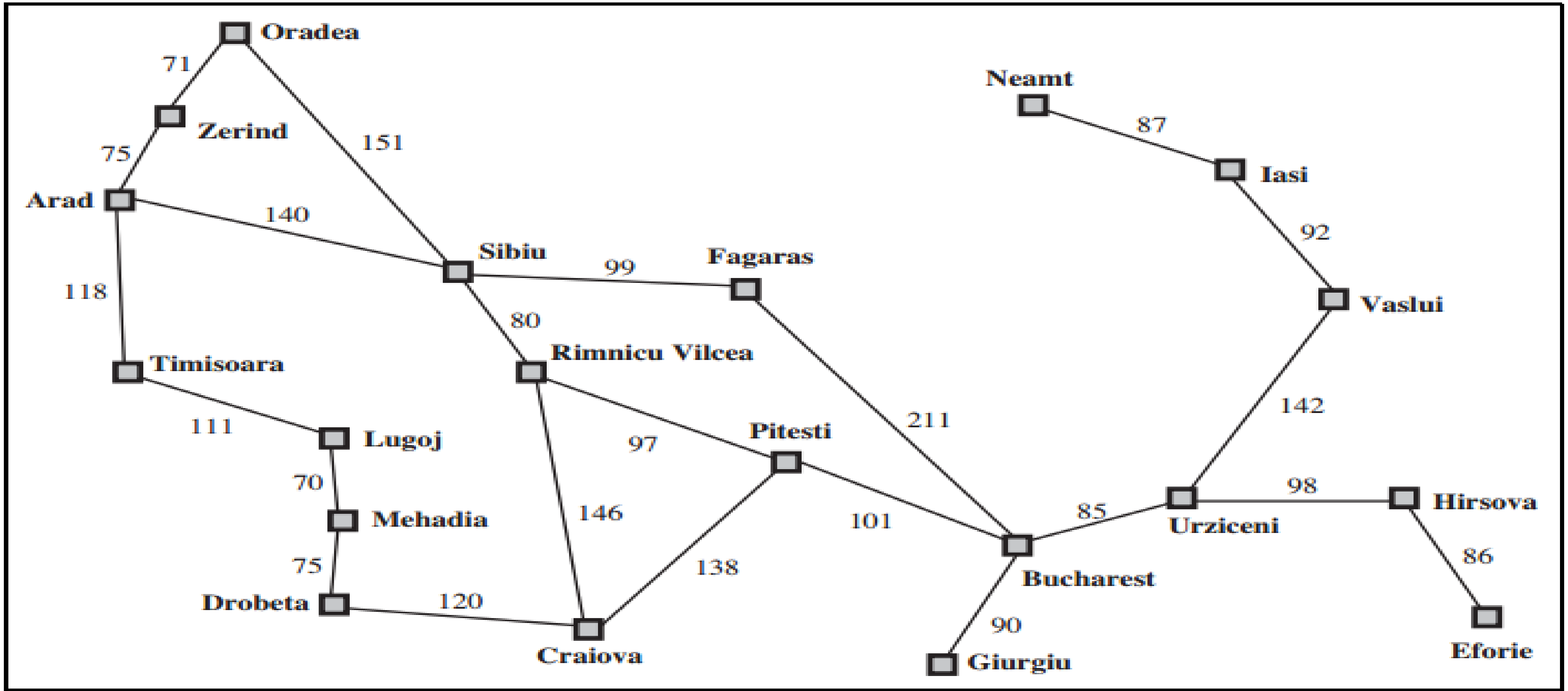


Figure 3.2 A simplified road map of part of Romania.

Problems solving agents

- Intelligent agents are supposed to maximize their performance measure by achieving a **goal** and aim at satisfying it.
- **Problem formulation** is the process of **deciding** what actions and states to consider, about a given a goal.
- Lets us consider a very common problem of rout finding in a city to which an agent do not know.

Problems solving agents

- Our agent has now adopted the **goal of driving to Bucharest** and is considering where to go from Arad.
- Three roads lead out of Arad, one toward Sibiu, one to Timisoara, and one to Zerind. **None of these achieves the goal,** so unless the agent is familiar with the geography of Romania, it will not know which road to follow.
- In other words, the agent will not know which of its possible **actions is best**, because it does not yet know enough about the state that results from taking each action.

Problems solving agents

- Once it has found a path on the map from Arad to Bucharest, it can achieve its goal by carrying out the driving actions to destination with best route.
- we assume that the environment is **observable**, so the agent always knows the current state.
- Known Environment and deterministic Environment

Problems solving agents and searching

- The process of looking for a sequence of actions that reaches the goal is called **search**.
- A search algorithm takes a problem as input and returns a **solution** in the form of an action sequence.
- Once a solution is found, the actions it recommends can be carried out. This is called the **execution** phase.
- An agent that carries out its plans with its eyes closed, so to speak, must be quite certain of what is going on. this an **open-loop** system, because ignoring the percepts.

Learn yourself

What is problem and what are characteristics of a good problem?

Introduction to Searching

- There exists some problems in real world that are too complex to solve. For example Traveling salesperson problem, Knapsack problem
- Root finding problems, DNA match
- A **VLSI layout** problem requires positioning millions of components and connections on a chip to minimize area, minimize circuit delays, minimize stray capacitances, and maximize manufacturing yield.

Solving by Searching

- A solution is **an action sequence**, so search algorithms work by considering **various possible action sequences**.
- A **search tree** with the initial state at the **root**; the branches are **actions** and the **nodes** correspond to **states** in the state space of the problem.
- The root node of the tree corresponds to the **initial state**

Solving by Searching

- Then we need to consider taking **various actions**. We do this by **expanding** the current state; that is, applying **each legal action to the current state**, there by **generating** a new set of states.
- In this case, we add three branches from the **parent** *In(Arad)* leading to three new **child nodes**: *In(Sibiu), In(Timisoara),* and *In(Zerind)*.
- Now we must choose which of these three possibilities to consider further.

Solving by Searching

- This is the **essence of search**—
- **Following up one option now** and putting the **others aside** for later, in case the **first choice does not lead to a solution**.
- Suppose we choose Sibiu first. We check to see whether it is a goal state (it is not) and then expand it to get *In(Arad)*, *In(Fagaras)*, *In(Oradea)*, and *In(RimnicuVilcea)*.
- We can then choose any of these four or go back and choose Timisoara or Zerind. Each of these six nodes is a **leaf node**, that is, **a node with no children in the tree**.

Solving by Searching

- The set of all leaf nodes available for expansion at any given point is called the **frontier** or **Open List**.
- Search algorithms all share this basic structure; they vary primarily according to how they choose which state to expand next—the so-called **search strategy**.
- Redundant path, Loopy Path, Data Structures, Lifo Queue , FIFO Queue , Priority Queue
- **Measuring problem-solving performance**

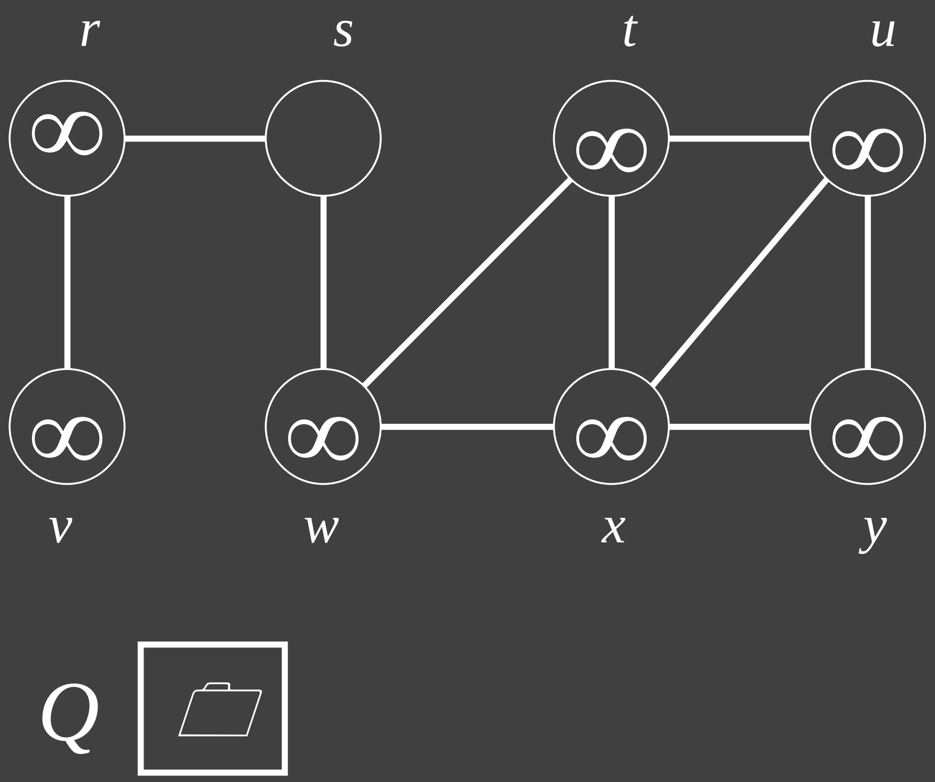
Uninformed Searching

- It is also called **blind search,** The term means that the strategies have **no additional information** about states beyond that provided in the problem definition.
- All they can do is generate successors and distinguish a goal state from a non-goal state. All search strategies are distinguished by the *order* in which nodes are expanded.
- BFS , DFS ,Depth limited Search , Iterative Deepening

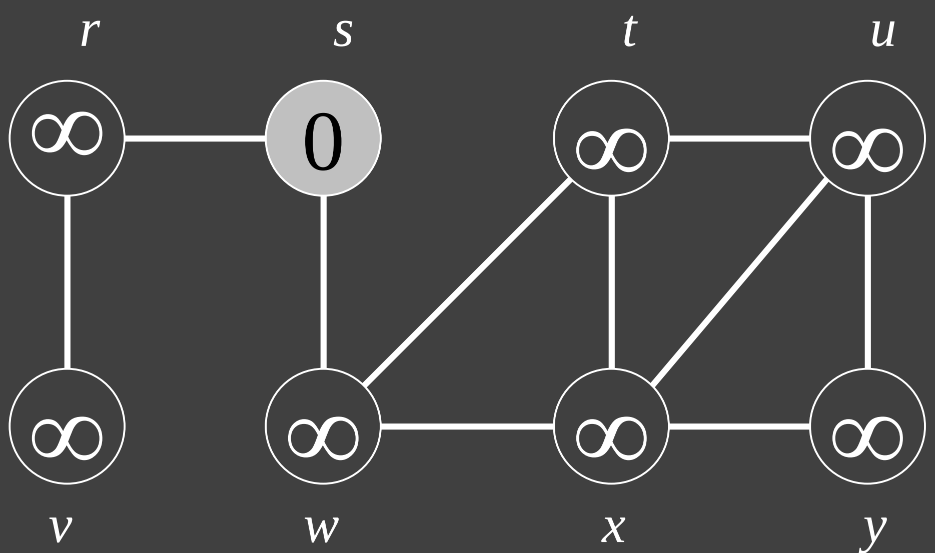
Breadth First Search

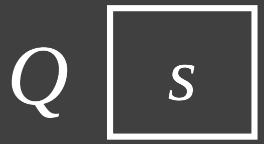
- **Breadth-first search** is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then *their* successors, and so on.
- In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.
- Breadth-first search is an instance of the general graph-search algorithm in which the *shallowest* unexpanded node is chosen for expansion. FIFO queue for the frontier.

Breadth First Search

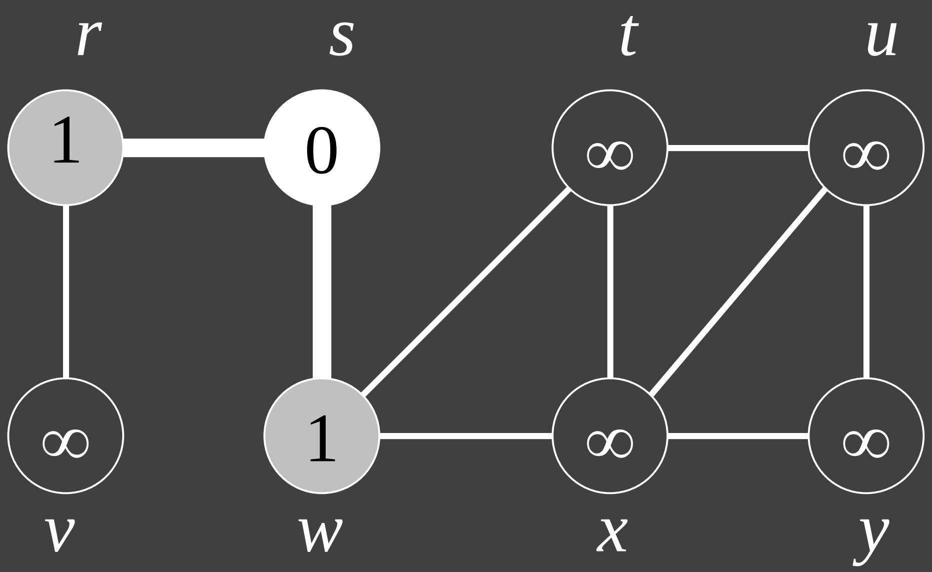


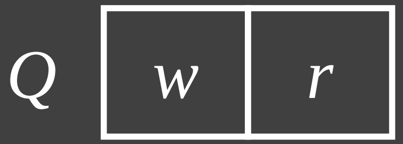
Breadth First Search



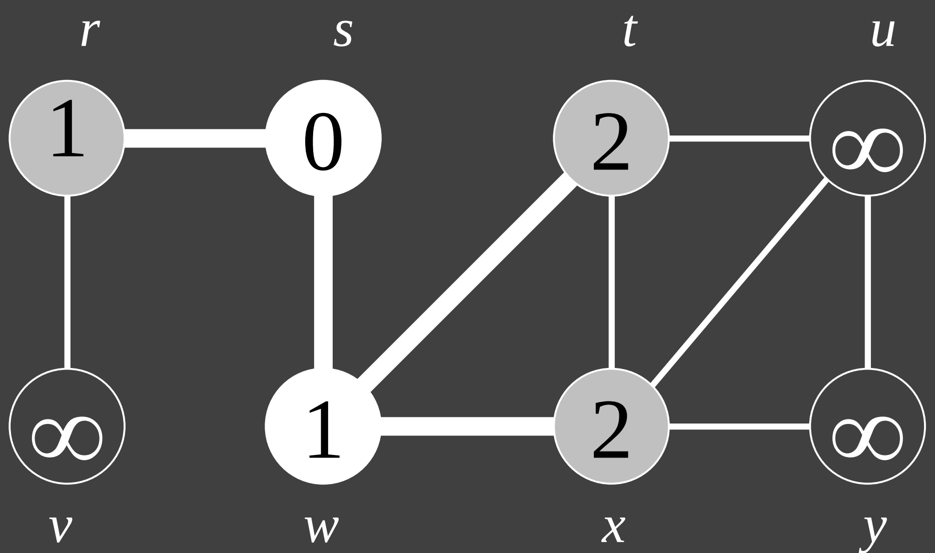


Breadth First Search



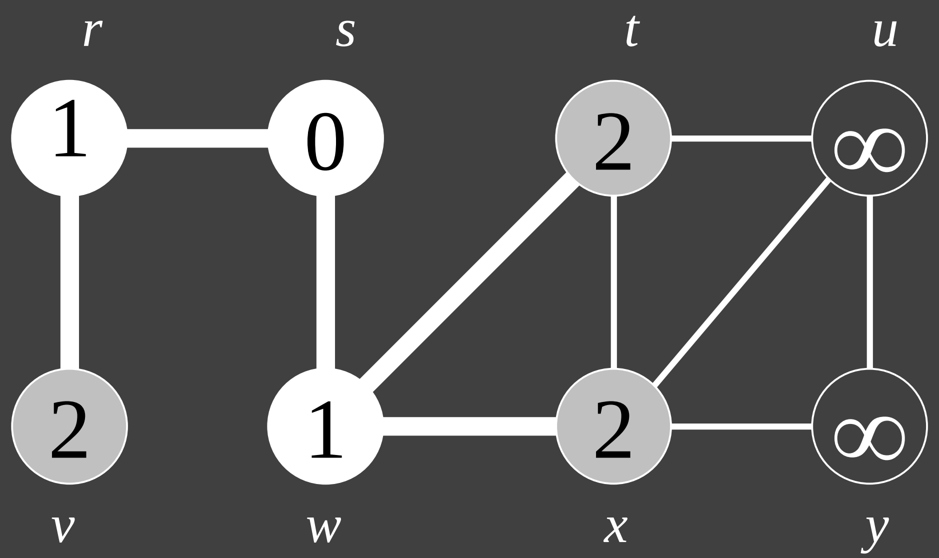


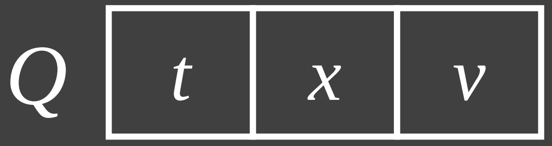
Breadth First Search



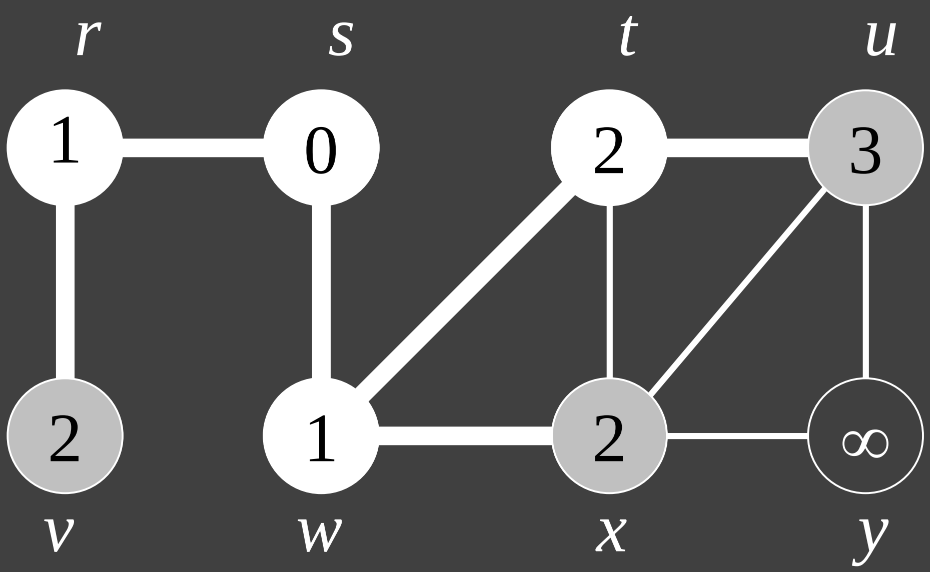
Q	r	t	x

Breadth First Search





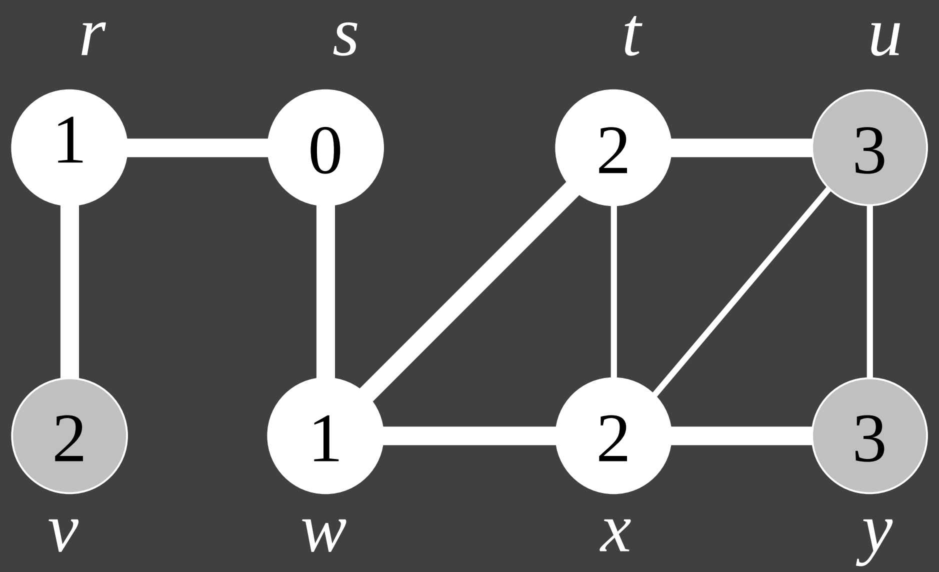
Breadth First Search



Q

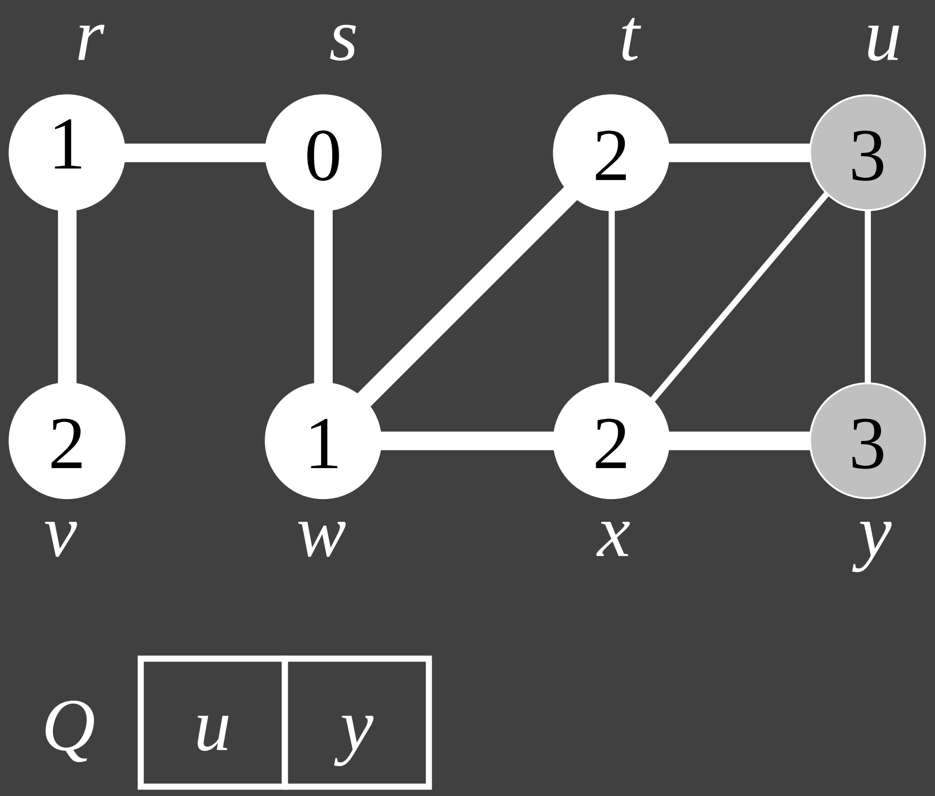
x	v	u

Breadth First Search

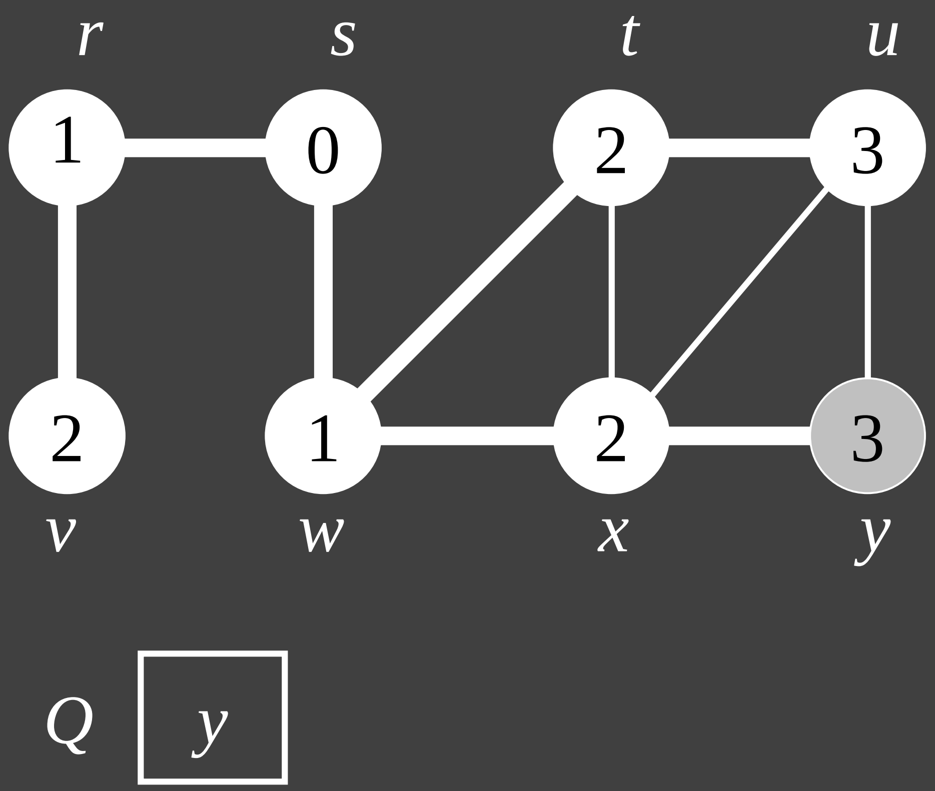


Q	v	u	y

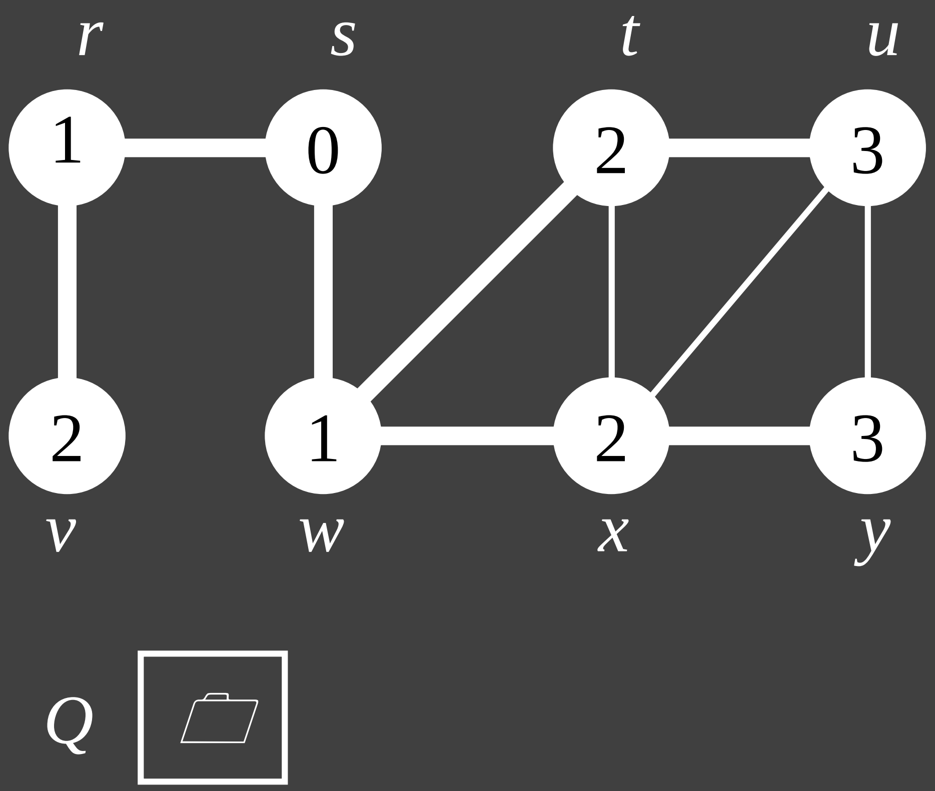
Breadth First Search



Breadth First Search



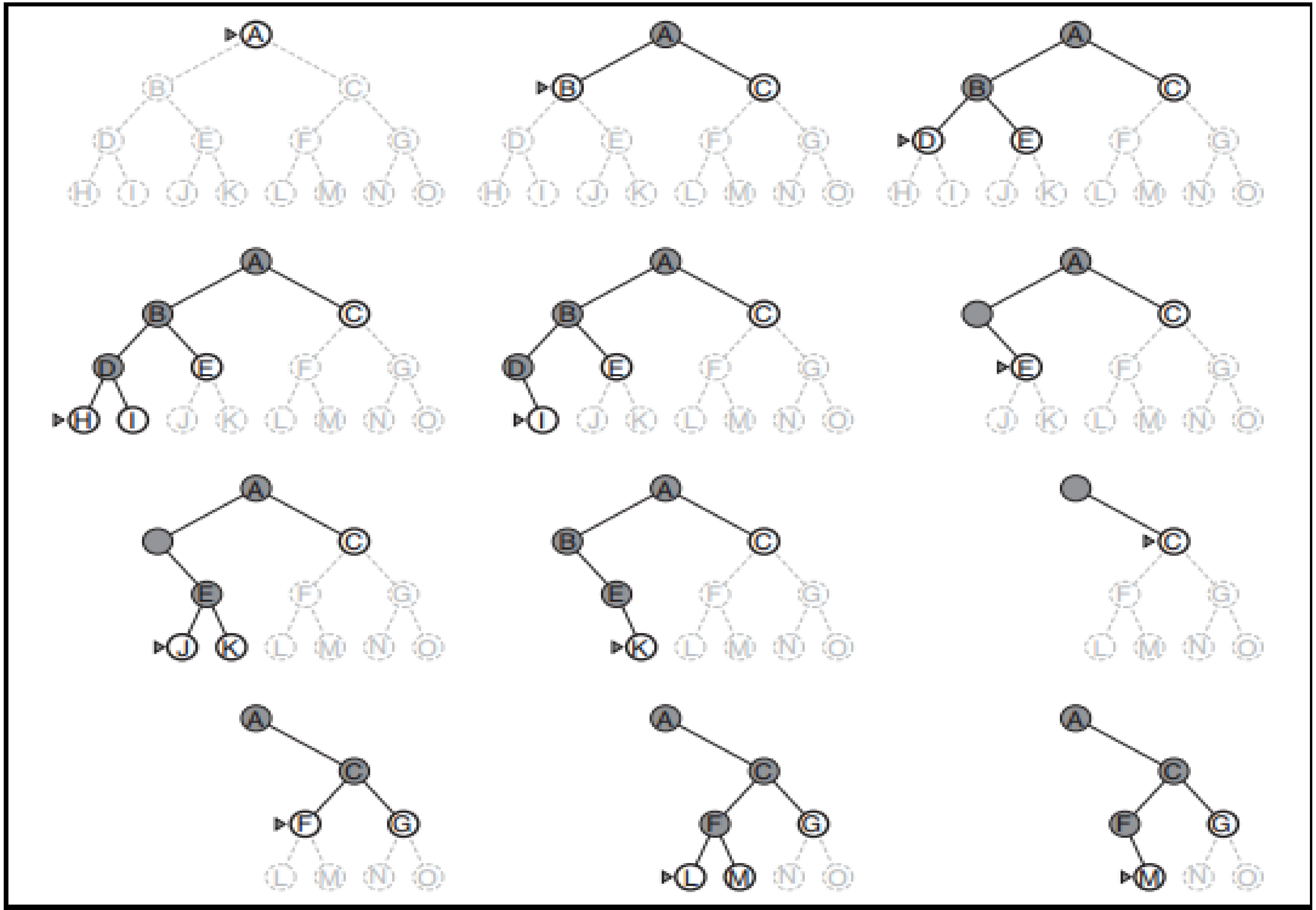
Breadth First Search



Depth First Search

- **Depth-first search** always expands the *deepest* node in the current frontier of the search tree. The progress of the search is illustrated in Figure 3.16. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes
are expanded, they are dropped from the frontier, so then the search “backs up” to the next deepest node that still has unexplored successors.
The depth-first search algorithm is an instance of the graph-search algorithm in Figure 3.7;uses a LIFO queue.

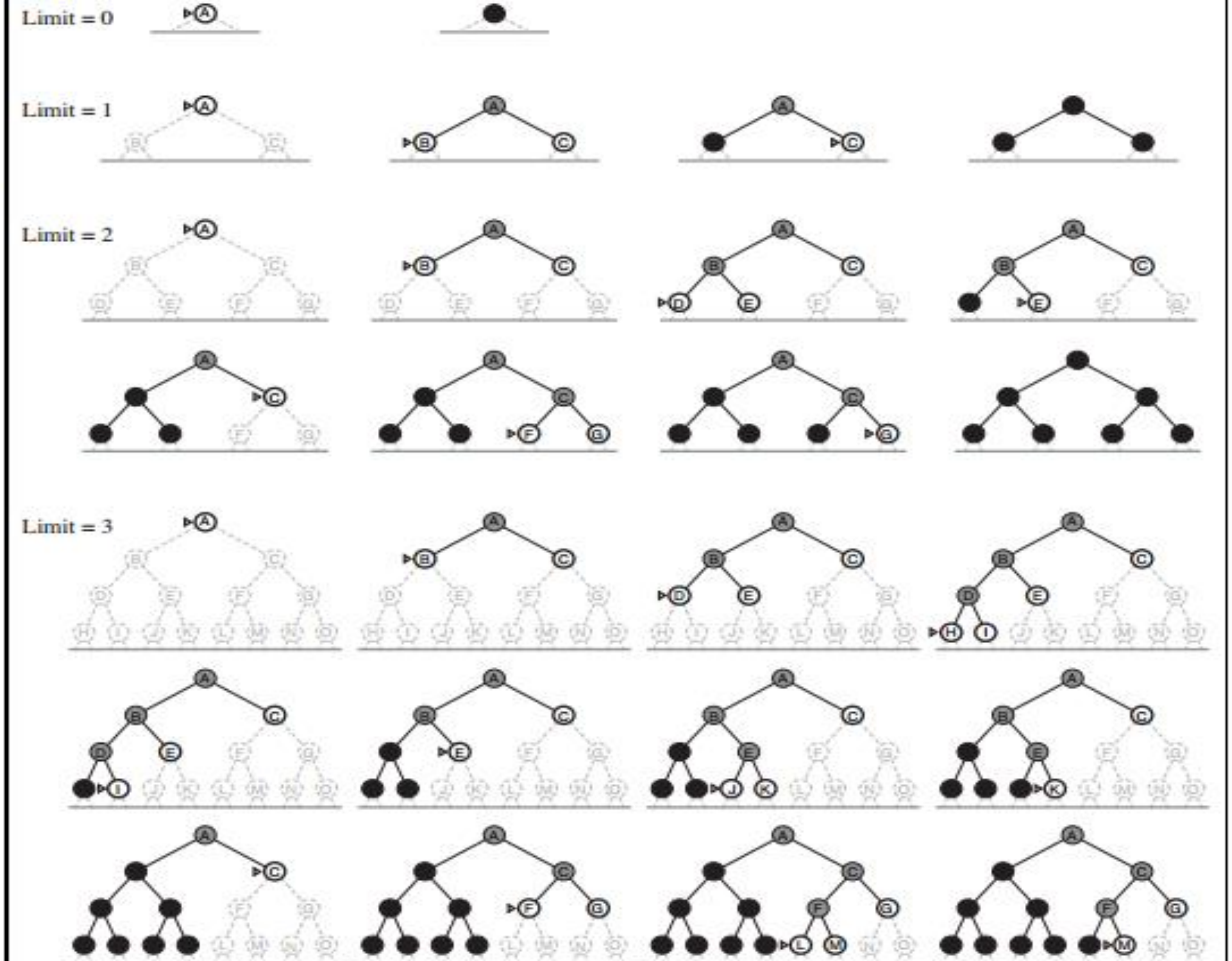
DFS



Iterative Deepening Depth First Search

- **Iterative deepening search** (or iterative deepening depth-first search) is a general strategy, often used in combination with depth-first tree search, that finds the best depth limit.
- It does this by gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found.
- This will occur when the depth limit reaches d, the depth of the shallowest goal node. The algorithm is shown in Figure 3.18. Iterative deepening combines the benefits of depth-first and breadth-first search.

Iterative Deepening Depth First Search

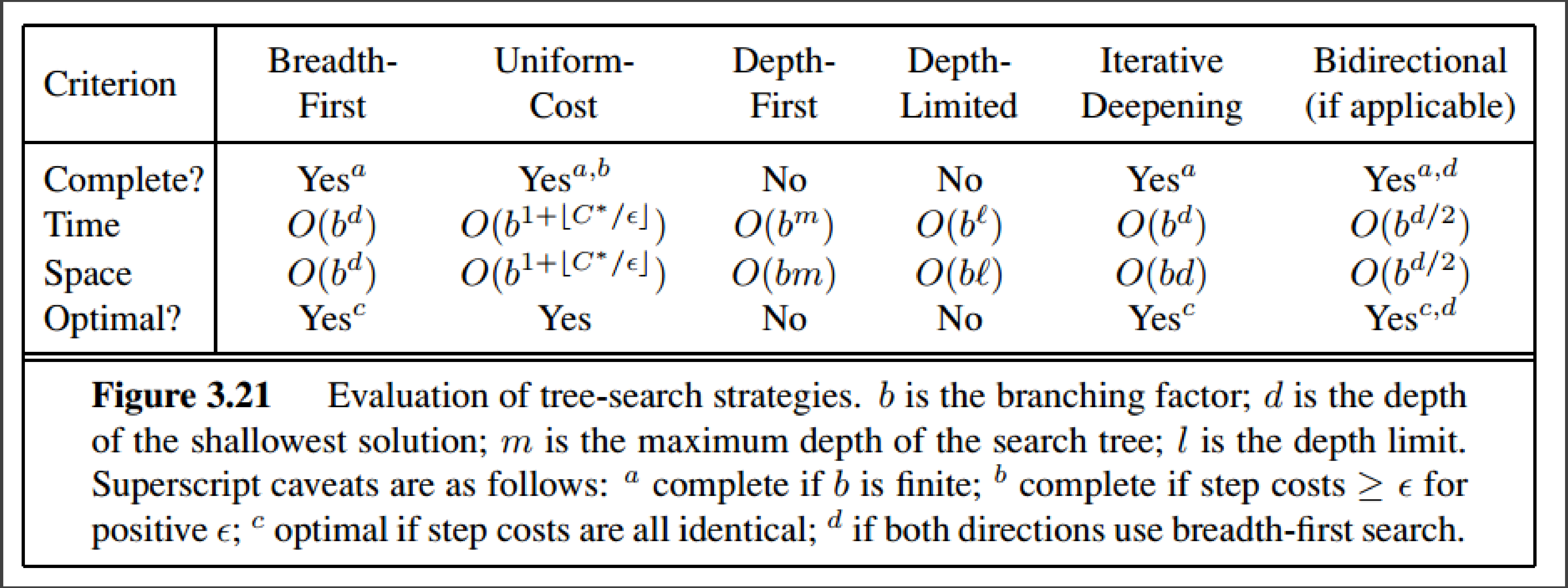


Reference Material

Artificial Intelligence: A Modern Approach, by Russell and Norvig, Prentice Hall. 2ndEdition. ISBN-10: 0137903952

Chapter 2 and 3

Comparison uninformed algorithms



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

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

Thanks……

HABIB ULLAH QAMAR

MSCS SE (MBA HRM)