

Chapter 1: Intelligent Agents



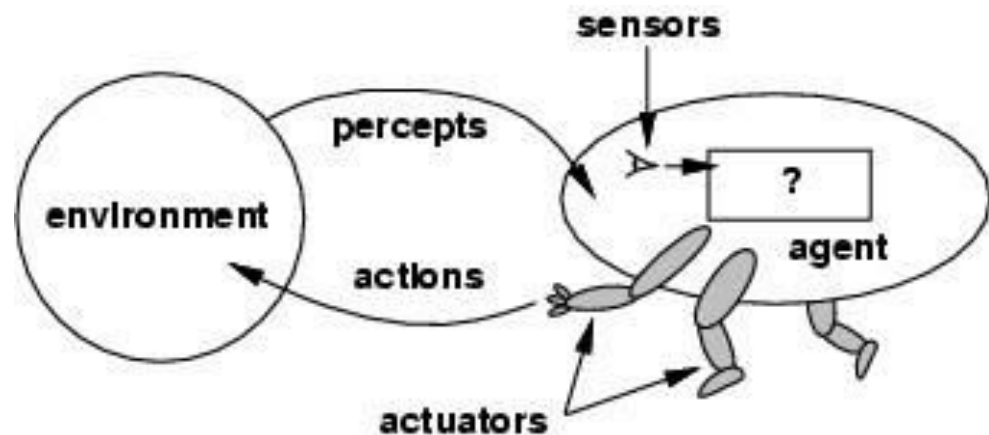


Outline

- Agents and Environments
- Agent Terminology
- ✓ ■ Rationality
- The structure of Intelligent Agent
- PEAS Representation

Agents

- An AI system is composed of an agent and its environment. The agent acts in their environment.
- ✓ ■ An **agent** is anything that can be viewed as **perceiving** its **environment** through **sensors** and **acting** upon that environment through **actuators**



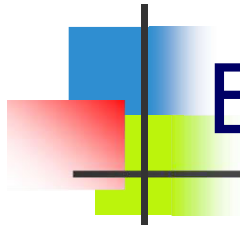


Agents (Cont.)

✓ **Sensor:** Sensor is a device which detects the change in the environment and sends the information to other electronic devices. An agent observes its environment through sensors.

Actuators: Actuators are the component of machines that converts energy into motion. The actuators are only responsible for moving and controlling a system. An actuator can be an electric motor, gears, rails, etc.

Effectors: Effectors are the devices which affect the environment. Effectors can be legs, wheels, arms, fingers, wings, fins, and display screen.



Examples of Agents

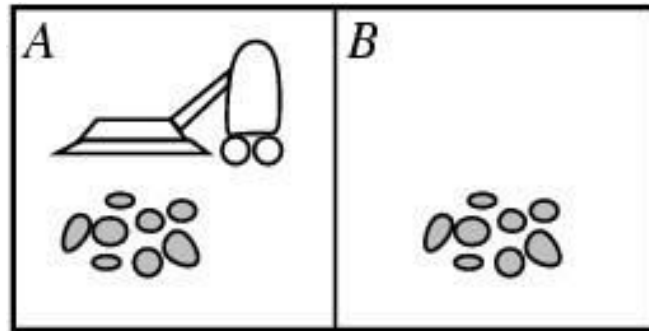
- Human agent: A human agent has eyes, ears, and other organs which work for sensors and hand, legs, vocal tract work for actuators.
- Robotic agent: A robotic agent can have cameras, infrared range finder, NLP for sensors and various motors for actuators.
- Software agent: Software agent can have keystrokes, file contents as sensory input and act on those inputs and display output on the screen.



Terminologies

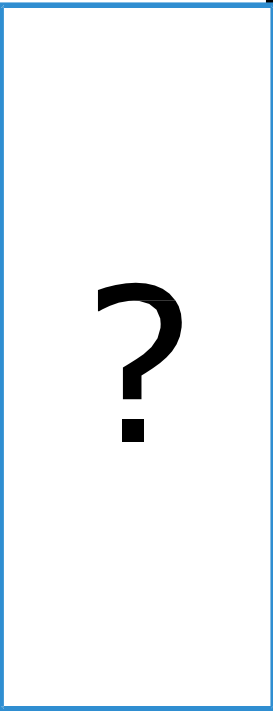
- ❑ **Performance Measure of Agent** – It is the criteria, which determines how successful an agent is.
- ❑ **Behaviour of Agent** – It is the action that agent performs after any given sequence of percepts.
- ❑ **Percept** – It is agent's perceptual inputs at a given instance.
- ❑ **Percept Sequence** – It is the history of all that an agent has perceived till date.
- ❑ **Agent Function** – It is a map from the precept sequence to an action.

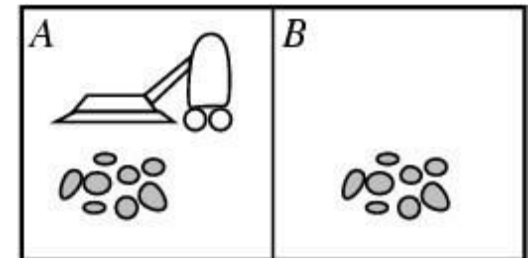
Vacuum-Cleaner World



- **Percepts:** location and contents, e.g., [A, dirty]
- **Actions:** Left, Right, Suck, NoOp

A Simple Agent Function

Percept sequence	Action
[A, Clean]	
[A, Dirty]	
[B, Clean]	
[B, Dirty]	
[A, Clean], [A, Clean]	
[A, Clean], [A, Dirty]	
...	
[A, Clean], [A, Clean], [A, Clean]	
[A, Clean], [A, Clean], [A, Dirty]	
...	





Rationality

An agent should "**do the right thing**", based on what it can perceive and the actions it can perform.

An ideal rational agent is the one, which is capable of doing expected actions to maximize its performance measure.

* Rationality of an agent depends on the following four factors –

The **performance measures**, which determine the degree of success.

Agent's **Percept Sequence** till now.

The agent's **prior knowledge about the environment**.

The **actions** that the agent can carry out.



Vacuum-Cleaner Example

- A simple agent that cleans a square if it is dirty and moves to the other square if not
- Is it rational?
- Assumption:
 - performance measure: 1 point for each clean square at each time step
 - environment is known a priori
 - actions = {left, right, suck, no-op}
 - agent is able to perceive the location and dirt in that location
- Given different assumption, it might not be rational anymore



Structure of Intelligent Agent

Agent's structure can be viewed as –

— Agent = Architecture + Agent Program

Architecture = the machinery that an agent executes on.

Agent Program = an implementation of an agent function.



✓ **PEAS** is a type of model on which an AI agent works upon. It is made up of four words:

P: Performance measure

E: Environment

A: Actuators

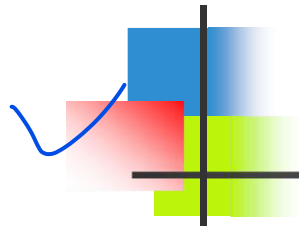
S: Sensors

Here **performance** measure is the objective for the success of an agent's behaviour.



Taxi Driver Example

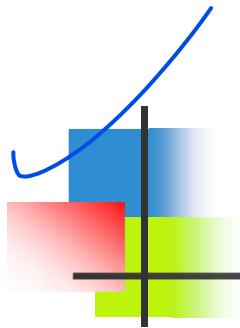
Performance Measure	Environment	Actuators	Sensors
safe, fast, legal, comfortable trip, maximize profits	roads, other traffic, pedestrians, customers	steering, accelerator, brake, signal, horn, display	camera, sonar, speedometer, GPS, odometer, engine sensors, keyboard, accelerator



Medical Diagnosis System

Performance Measure	Environment	Actuators	Sensors
healthy patient, minimize costs, lawsuits	patient, hospital, staff	display questions, tests, diagnosis, treatments, referrals	keyboard entry of symptoms, findings, patient's answers





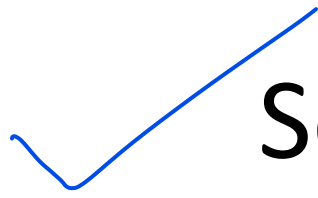
Mushroom-Picking Robot

Performance Measure	Environment	Actuators	Sensors
Percentage of good mushrooms in correct bins	Conveyor belt with mushrooms, bins	Jointed arm and hand	camera, joint angle sensors



Search Techniques

Lecture-02



Searching in AI

Rational agents use these search strategies or algorithms to solve a specific problem and provide the best result.

Search Terminology:

Problem Space – It is the environment in which the search takes place. (A set of states and set of operators to change those states)

Problem Instance – It is Initial state + Goal state.

Problem Space Graph – It represents problem state. States are shown by nodes and operators are shown by edges.

Depth of a problem – Length of a shortest path or shortest sequence of operators from Initial State to goal state.

Searching in AI (Cont..)

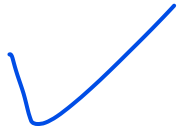
Space Complexity – The maximum number of nodes that are stored in memory.

Time Complexity – The maximum number of nodes that are created.

Admissibility – A property of an algorithm to always find an optimal solution.

Branching Factor – The average number of child nodes in the problem space graph.

Depth – Length of the shortest path from initial state to goal state.



Properties of Search Algorithms:

Following are the four essential properties of search algorithms to compare the efficiency of these algorithms:

Completeness: A search algorithm is said to be complete if it **guarantees to return a solution** if at least **any solution exists** for any random input.

Optimality: If a solution found for an algorithm is **guaranteed to be the best solution (lowest path cost)** among all other solutions, then such a solution for is said to be an optimal solution.

Time Complexity: Time complexity is a **measure of time for an algorithm** to complete its task.

Space Complexity: It is the maximum **storage space required** at any point during the search, as the complexity of the problem.

Types of search algorithms

Based on the search problems we can classify the search algorithms into **uninformed (Blind search) search** and **informed search (Heuristic search) algorithms**.

Uninformed Search:

The uninformed search does not contain any domain knowledge. It operates in a **brute-force** way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes.

For example, **BFS, DFS, Uniform Cost Search and others**.

Informed Search

Informed search algorithms **use domain knowledge**. In an informed search, problem information is available which can guide the search.

For example, **Greedy Search, A* Search**.



Breadth First Search

Breadth first search

Let *fringe* be a list containing the initial state

Loop

if *fringe* is empty return failure

Node \leftarrow remove-first (*fringe*)

if Node is a goal

then return the path from initial state to Node

else generate all successors of Node, and

(merge the newly generated nodes into *fringe*)

add generated nodes to the back of *fringe*

End Loop



Concept

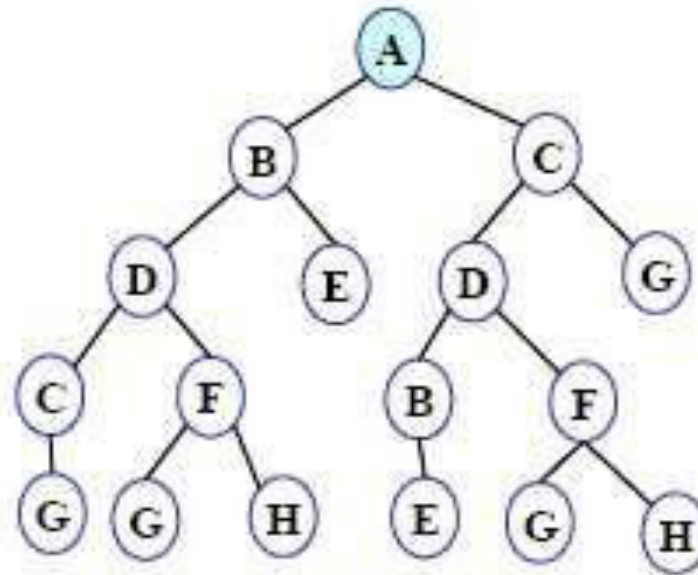
Step 1: Traverse the root node

Step 2: Traverse all neighbours of root node.

Step 3: Traverse all neighbours of neighbours of the root node.

Step 4: This process will continue until we are getting the goal node.

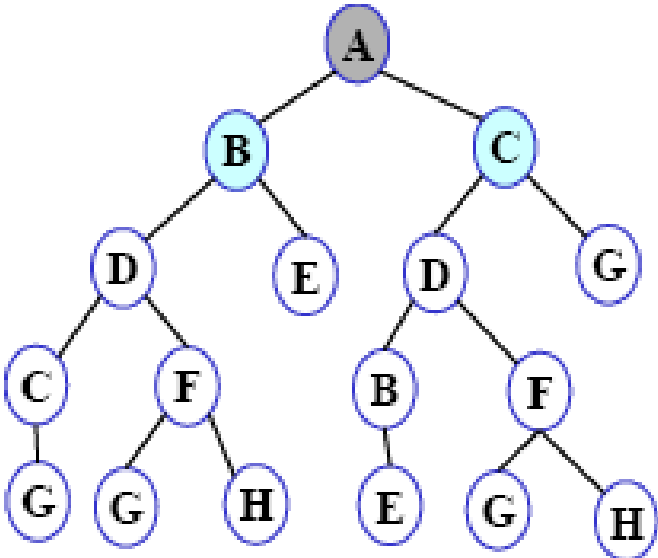
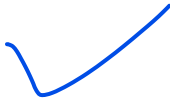
Step 1: Initially fringe contains only one node corresponding to the source state A.



FRINGE: A

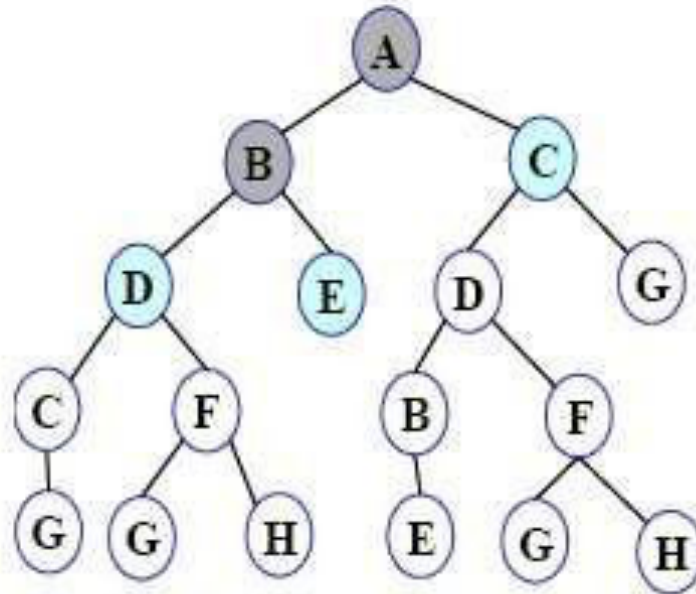


Step 2: A is removed from fringe. The node is expanded, and its children B and C are generated. They are placed at the back of fringe.



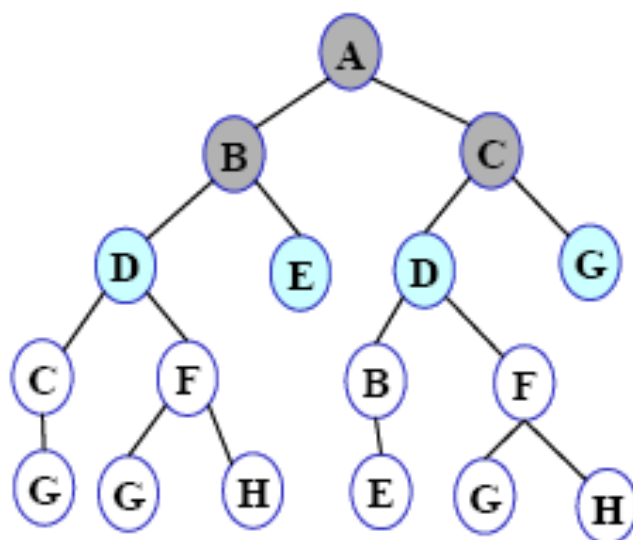
FRINGE: B C

Step 3: Node B is removed from fringe and is expanded. Its children D, E are generated and put at the back of fringe.



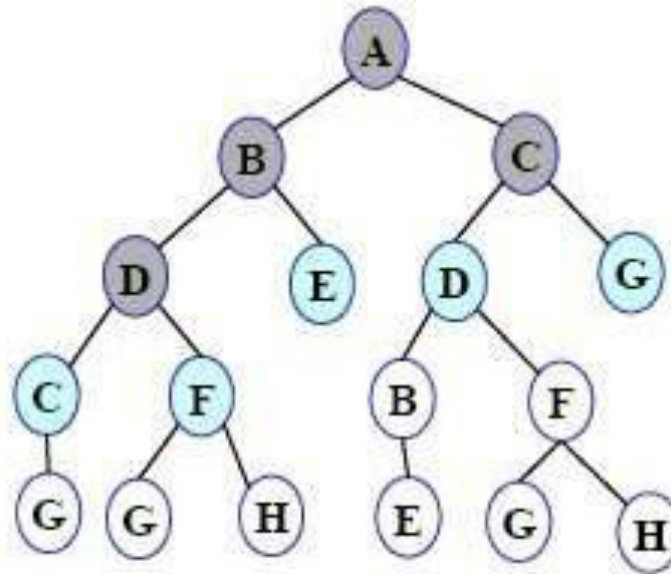
FRINGE: C D E

Step 4: Node C is removed from fringe and is expanded. Its children D and G are added to the back of fringe.



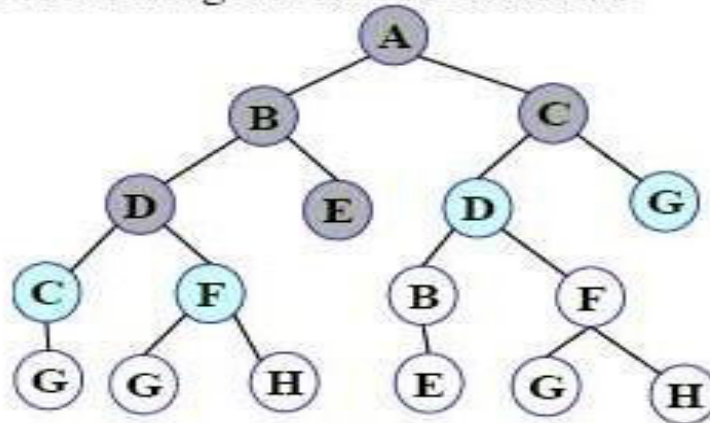
FRINGE: D E D G

Step 5: Node D is removed from fringe. Its children C and F are generated and added to the back of fringe.



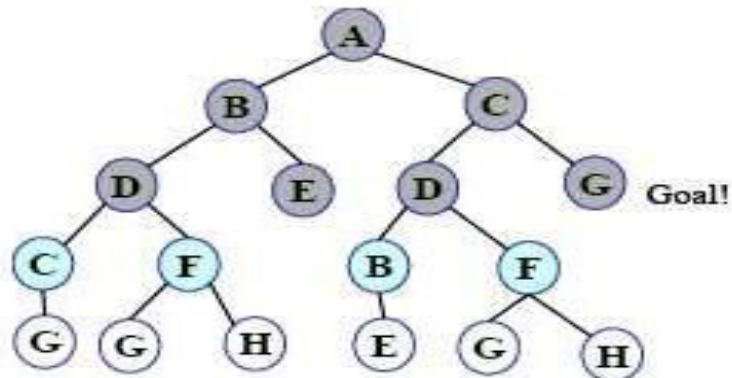
FRINGE: E D G C F

Step 6: Node E is removed from fringe. It has no children.



FRINGE: D G C F

Step 7: D is expanded, B and F are put in OPEN.



FRINGE: G C F B F

Step 8: G is selected for expansion. It is found to be a goal node. So the algorithm returns the path A C G by following the parent pointers of the node corresponding to G. The algorithm terminates.



Properties of BFS

- Complete.
- The algorithm is **optimal** (i.e., admissible) if all operators have the same cost. Otherwise, breadth first search finds a solution with the shortest path length.
- The algorithm has exponential time and space complexity. The time and space complexity of the algorithm is $O(bd)$ where d is the depth of the solution and b is the branching factor (i.e., number of children) at each node.



Advantages

- In this procedure at any way it will find the goal.
- It does not follow a single unfruitful path for a long time.
- It finds the minimal solution in case of multiple paths.

Disadvantages

- BFS consumes large memory space.
- Its time complexity is more.
- It has long pathways, when all paths to a destination are on approximately the same search depth.

Depth First Search

Depth First Search

Let *fringe* be a list containing the initial state

Loop

if *fringe* is empty return failure

Node \leftarrow remove-first (*fringe*)

if Node is a goal

then return the path from initial state to Node

else generate all successors of Node, and

merge the newly generated nodes into *fringe*

add generated nodes to the front of *fringe*

End Loop

Concept

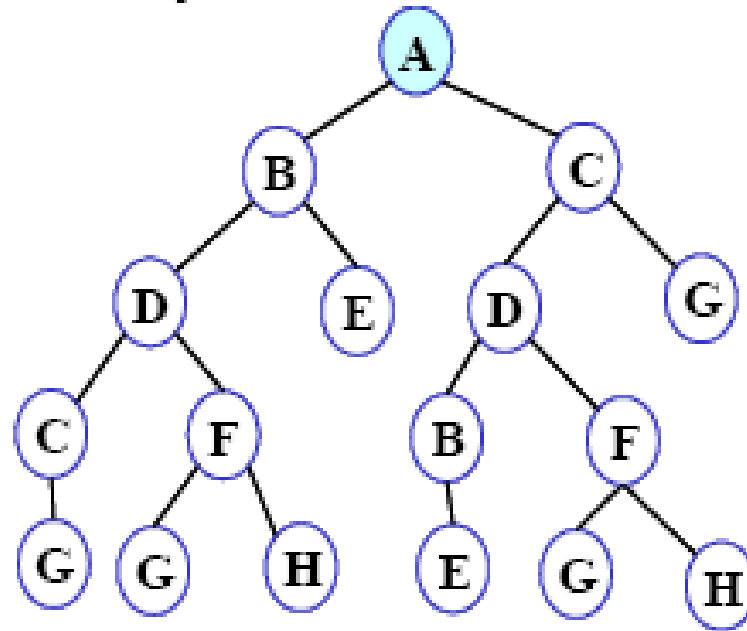
Step 1: Traverse the root node.

Step 2: Traverse any neighbour of the root node.

Step 3: Traverse any neighbour of neighbour of the root node.

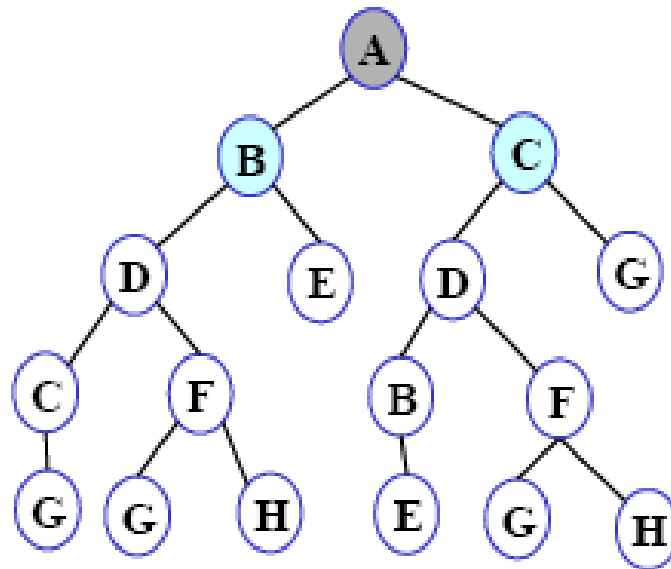
Step 4: This process will continue until we are getting the goal node.

Step 1: Initially fringe contains only the node for A.



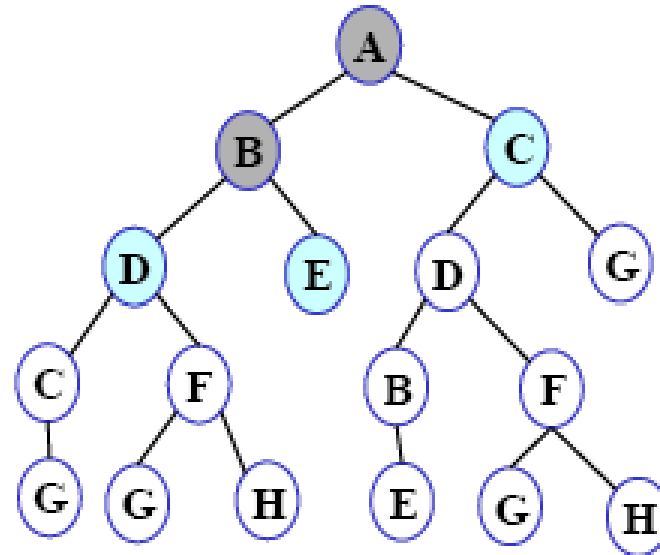
FRINGE: A

Step 2: A is removed from fringe. A is expanded and its children B and C are put in front of fringe.



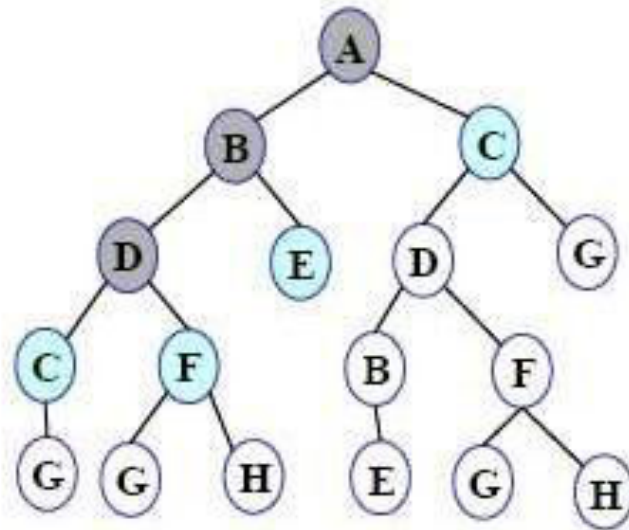
FRINGE: B C

Step 3: Node B is removed from fringe, and its children D and E are pushed in front of fringe.



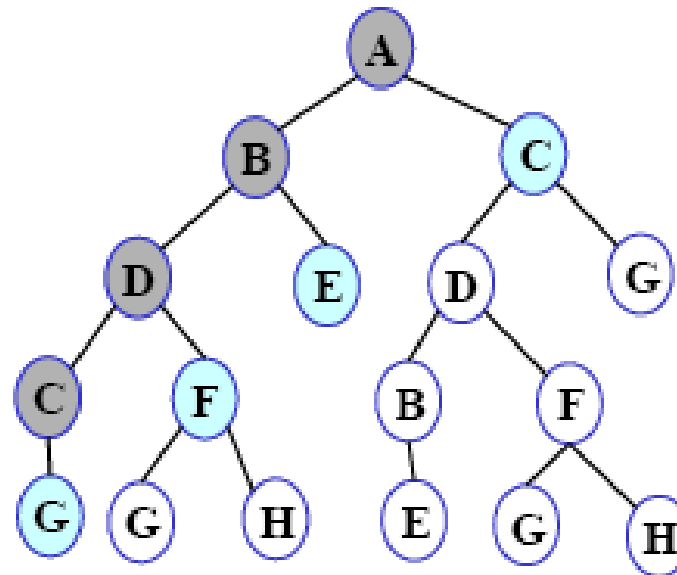
FRINGE: D E C

Step 4: Node D is removed from fringe. C and F are pushed in front of fringe



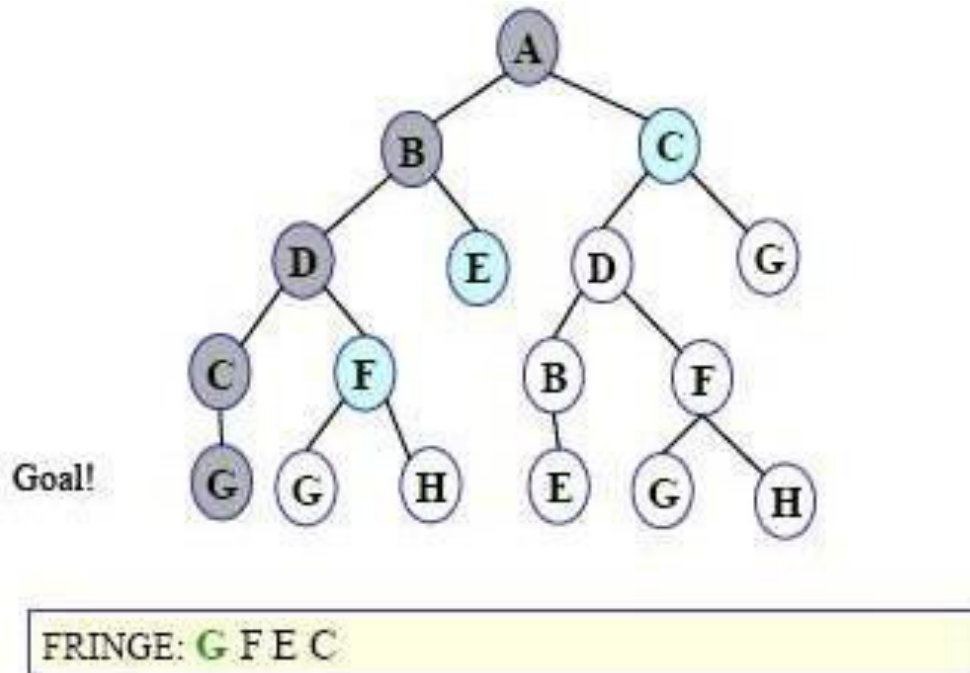
FRINGE: C F E C

Step 5: Node C is removed from fringe. Its child G is pushed in front of fringe.



FRINGE: G F E C

Step 6: Node G is expanded and found to be a goal node. The solution path A-B-D-C-G is returned and the algorithm terminates.





Properties of DFS

- The algorithm takes **exponential time**. If N is the maximum depth of a node in the search space, in the worst case the algorithm **will take time $O(b^d)$** . However the space taken is linear in the depth of the search tree, $O(bN)$.
- Note that the time taken by the algorithm is **related to the maximum depth of the search tree**. **If the search tree has infinite depth, the algorithm may not terminate**. This can happen **if the search space is infinite**. It can also happen if the search space contains **cycles**. **The latter case can be handled by checking for cycles in the algorithm**. **Thus Depth First Search is not complete**.



Advantages:

- DFS consumes very **less memory space.**
- It will reach at the goal node in a less time period than BFS if it traverses in a **right path.**
- It may find a solution without examining much of search because we may get the desired solution.

Disadvantages:

- It is possible that many states keep **reoccurring.**
- There is no guarantee of finding the goal node.
- Sometimes **the states may also enter into infinite loops.**

Difference between BFS and DFS

BFS

- It uses the data structure **queue**.
- BFS is **complete** because it finds the solution if one exists.
- BFS takes **more space** i.e. equivalent to $O(b^d)$ where b is the maximum branch exist in a search
- tree and d is the maximum depth exist in a search tree.
- In case of several goals, it finds the best one.

DFS

- It uses the data structure **stack**.
- It is **not complete** because it may take infinite loop to reach at the goal node.
- The space complexity is **$O(d)$** .
- In case of several goals, it will terminate the solution in any order.

Uninformed Search

Chapter 3.1 – 3.4

read this

Uninformed Search Strategies

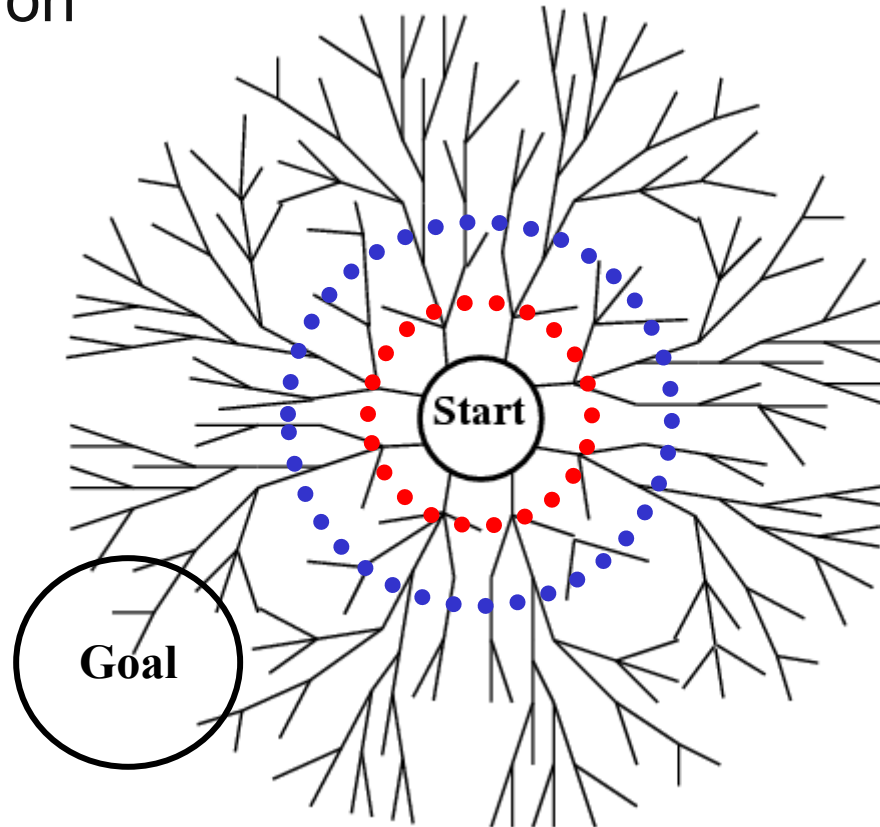
Uninformed Search: strategies that order nodes *without* using any domain specific information, i.e., don't use any information stored in a state

- **BFS:** breadth-first search
 - Queue (*FIFO*) used for the Frontier
 - remove from front, add to **back**
- **DFS:** depth-first search
 - Stack (*LIFO*) used for the Frontier
 - remove from front, add to **front**

Breadth-First Search (BFS)

Expand the shallowest node first:

1. Examine states **one** step away from the initial states
2. Examine states **two** steps away from the initial states
3. and so on

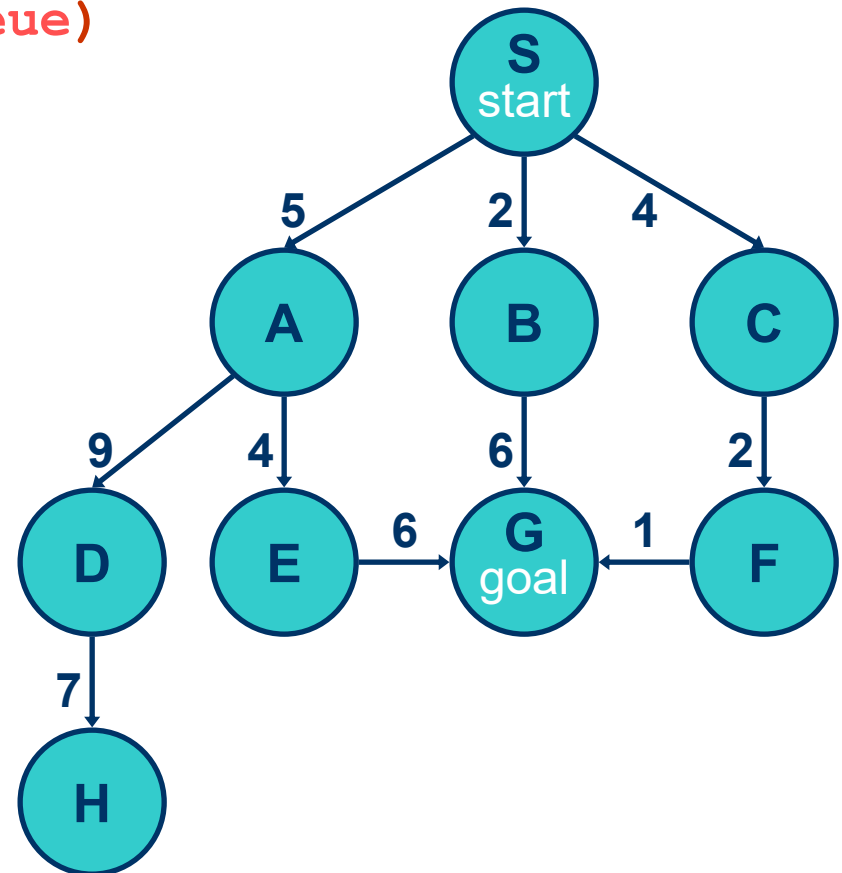


Breadth-First Search (BFS)

generalSearch(problem, queue)

of nodes tested: 0, expanded: 0

expnd. node	Frontier list
	{S}

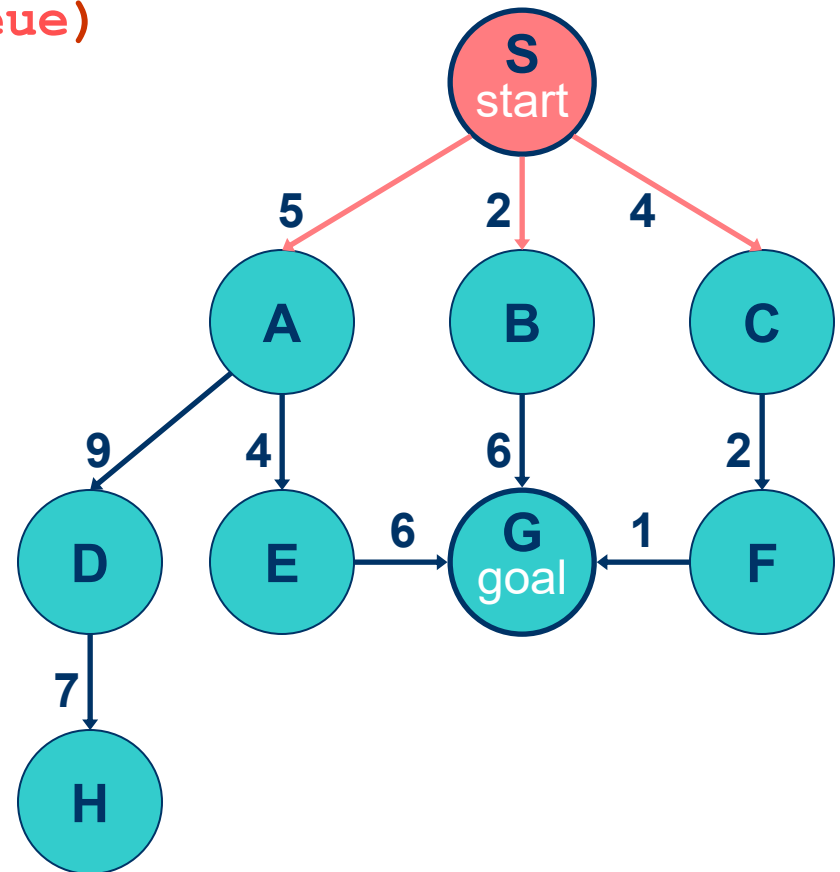


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

of nodes tested: 1, expanded: 1

expnd. node	Frontier list
	{S}
S not goal	{A,B,C}

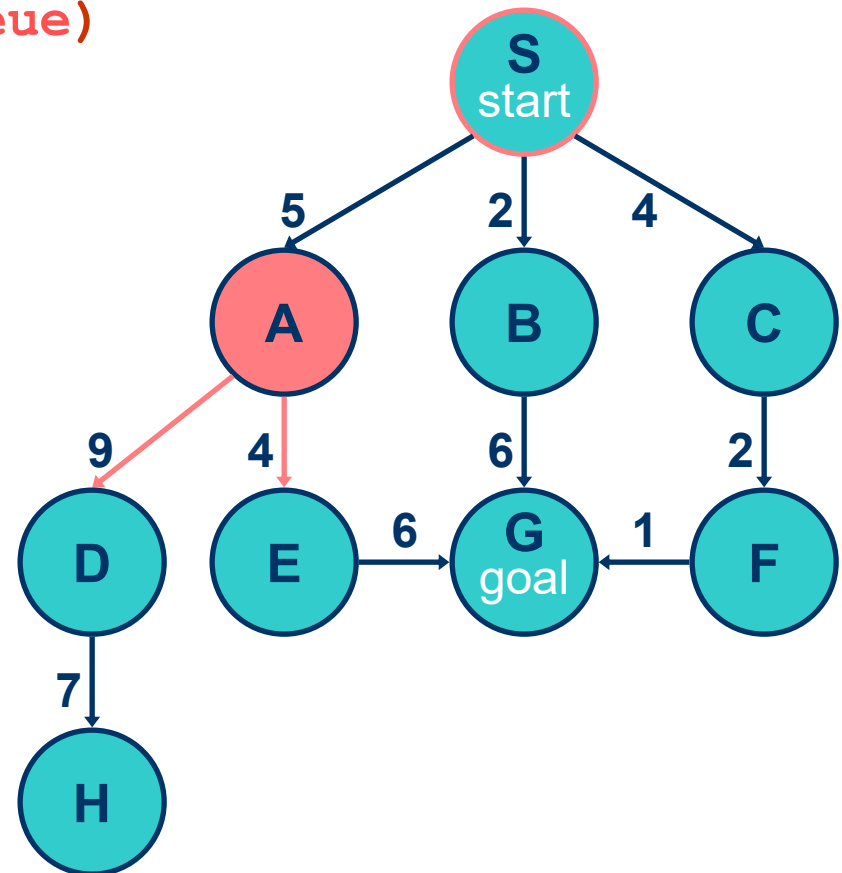


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

of nodes tested: 2, expanded: 2

expnd. node	Frontier list
	{S}
S	{A,B,C}
A not goal	{B,C,D,E}

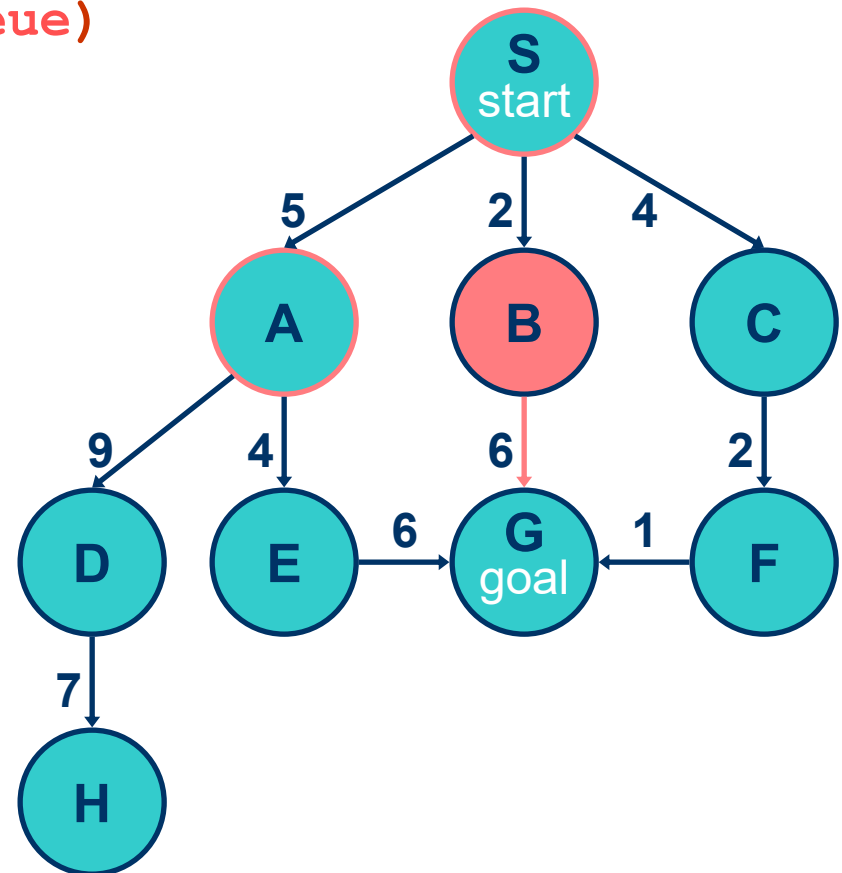


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

of nodes tested: 3, expanded: 3

expnd. node	Frontier list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B not goal	{C,D,E,G}

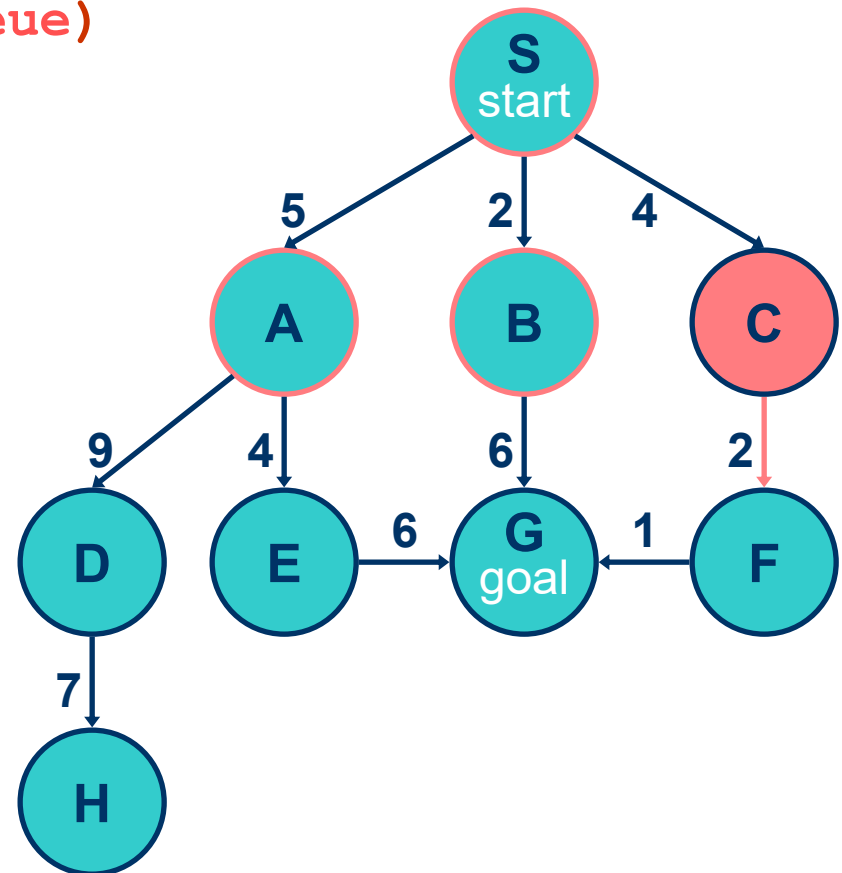


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

of nodes tested: 4, expanded: 4

expnd. node	Frontier list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C not goal	{D,E,G,F}

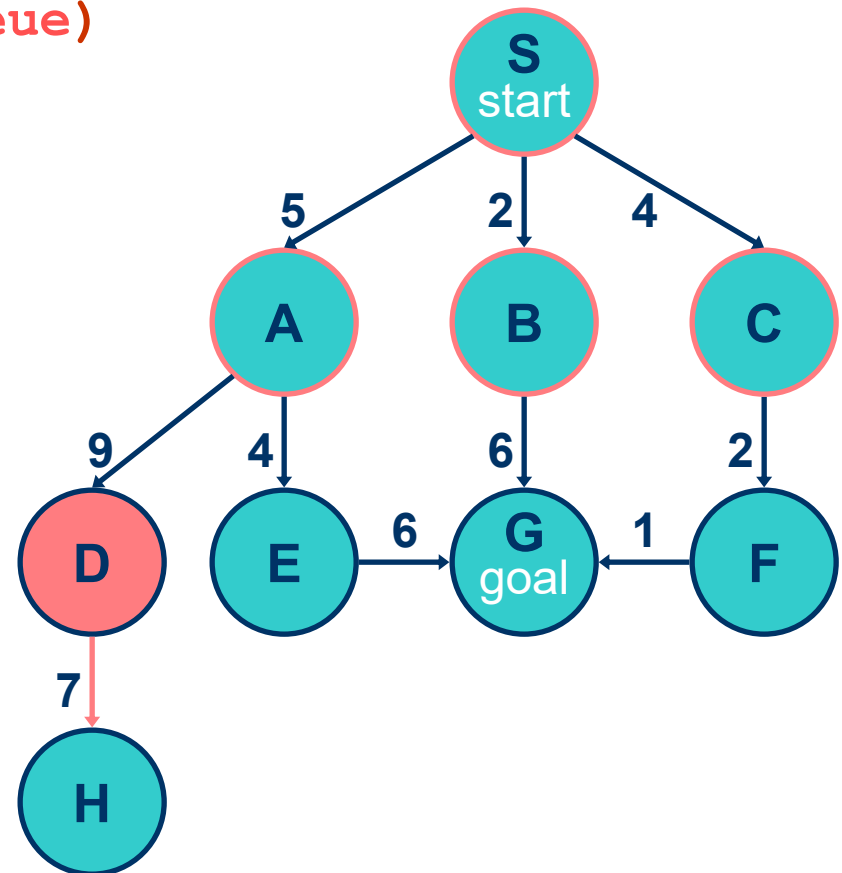


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

of nodes tested: 5, expanded: 5

expnd. node	Frontier list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D not goal	{E,G,F,H}

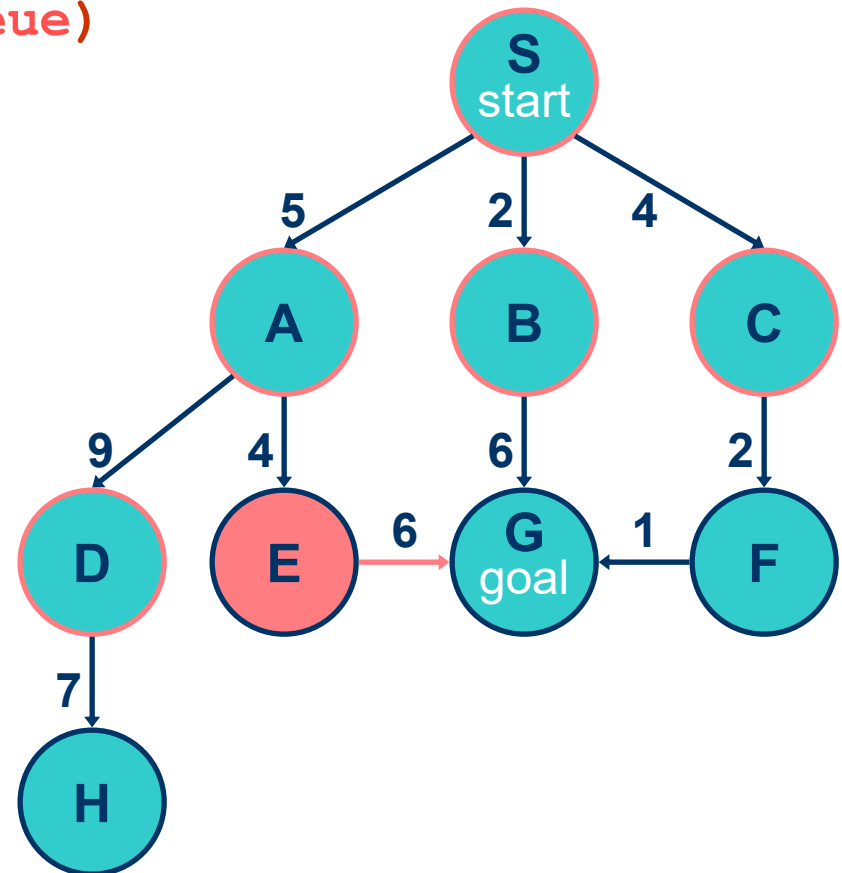


Breadth-First Search (BFS)

generalSearch(problem, queue)

of nodes tested: 6, expanded: 6

expnd. node	Frontier list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D	{E,G,F,H}
E not goal	{G,F,H,G}

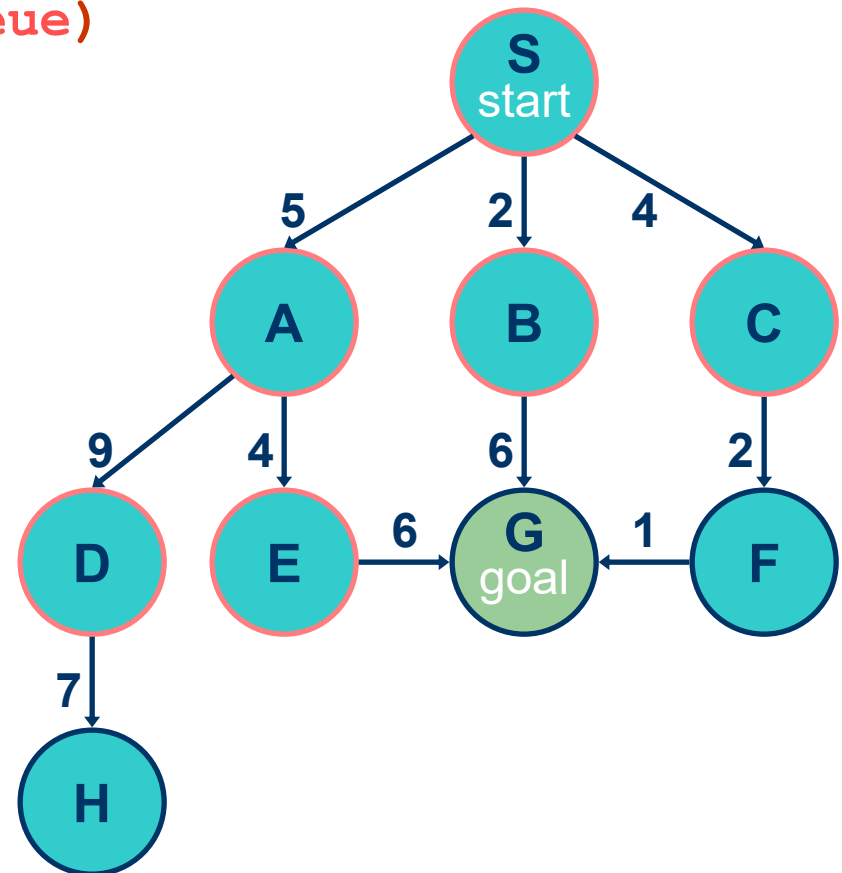


Breadth-First Search (BFS)

generalSearch(problem, queue)

of nodes tested: 7, expanded: 6

expnd. node	Frontier list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D	{E,G,F,H}
E	{G,F,H,G}
G goal	{F,H,G} no expand

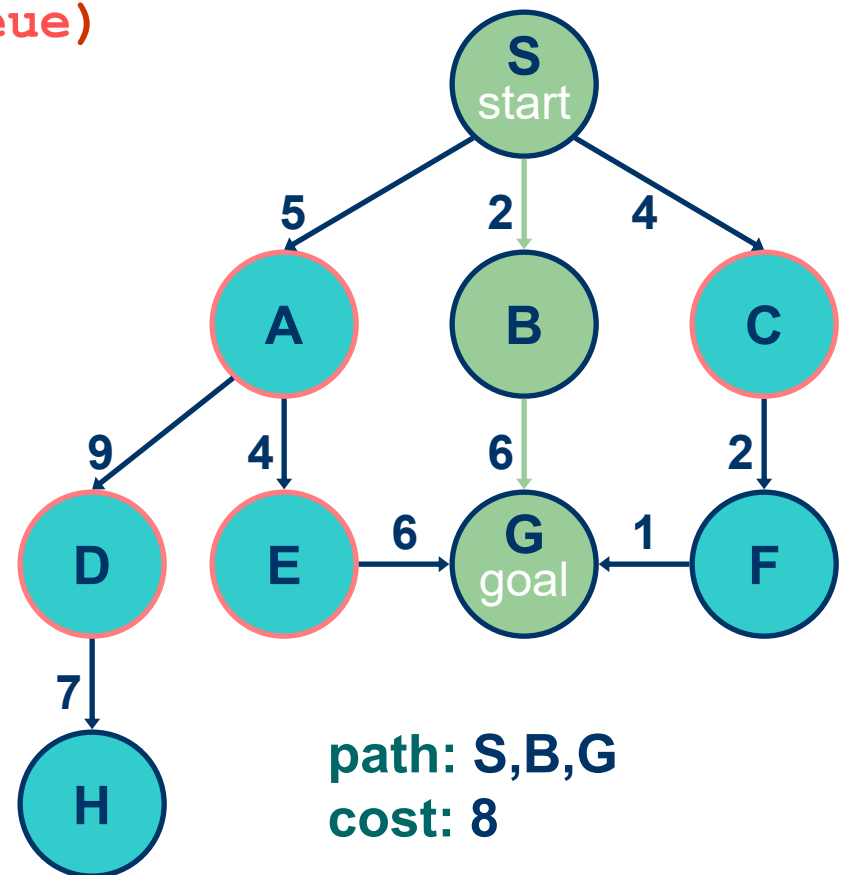


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

of nodes tested: 7, expanded: 6

expnd. node	Frontier list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D	{E,G,F,H}
E	{G,F,H,G}
G	{F,H,G}



Evaluating Search Strategies

- **Completeness**

If a solution exists, will it be found?

- a complete algorithm will find **a** solution (not all)

- **Optimality / Admissibility**

If a solution is found, is it guaranteed to be optimal?

- an admissible algorithm will find a **solution with minimum cost**

Evaluating Search Strategies

- **Time Complexity**

How long does it take to find a solution?

- usually measured for worst case
- measured by counting **number of nodes expanded**

- **Space Complexity**

How much space is used by the algorithm?

- measured in terms of the **maximum size of the *Frontier*** during the search

Breadth-First Search (BFS)

- **Complete**
- **Optimal / Admissible**
 - **Yes, *if*** all operators (i.e., arcs) have the same constant cost, or costs are positive, non-decreasing with depth
 - otherwise, not optimal but *does* guarantee finding solution of shortest *length* (i.e., fewest arcs)

Breadth-First Search (BFS)

- **Time and space complexity: $O(b^d)$ (i.e., exponential)**
 - d is the depth of the solution
 - b is the branching factor at each non-leaf node
- Very slow to find solutions with a large number of steps because must look at *all* shorter length possibilities first

Breadth-First Search (BFS)

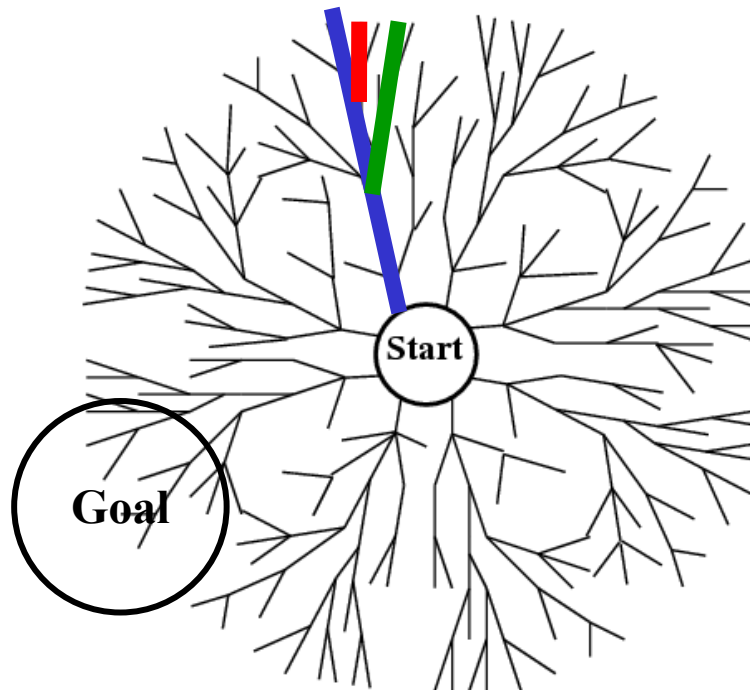
- **A complete search tree has a total # of nodes =**
 $1 + b + b^2 + \dots + b^d = (b^{(d+1)} - 1) / (b-1)$
 - d : the tree's depth
 - b : the branching factor at each non-leaf node
- **For example: $d = 12$, $b = 10$**
 $1 + 10 + 100 + \dots + 10^{12} = (10^{13} - 1)/9 = O(10^{12})$
 - If BFS expands 1,000 nodes/sec and each node uses 100 bytes of storage, then BFS will take 35 years to run in the worst case, and it will use 111 terabytes of memory!

Depth-First Search

Expand the **deepest** node first

1. Select a direction, go deep to the end —
2. Slightly change the end —
3. Slightly change the end some more... —

Use a **Stack** to order nodes on the **Frontier**

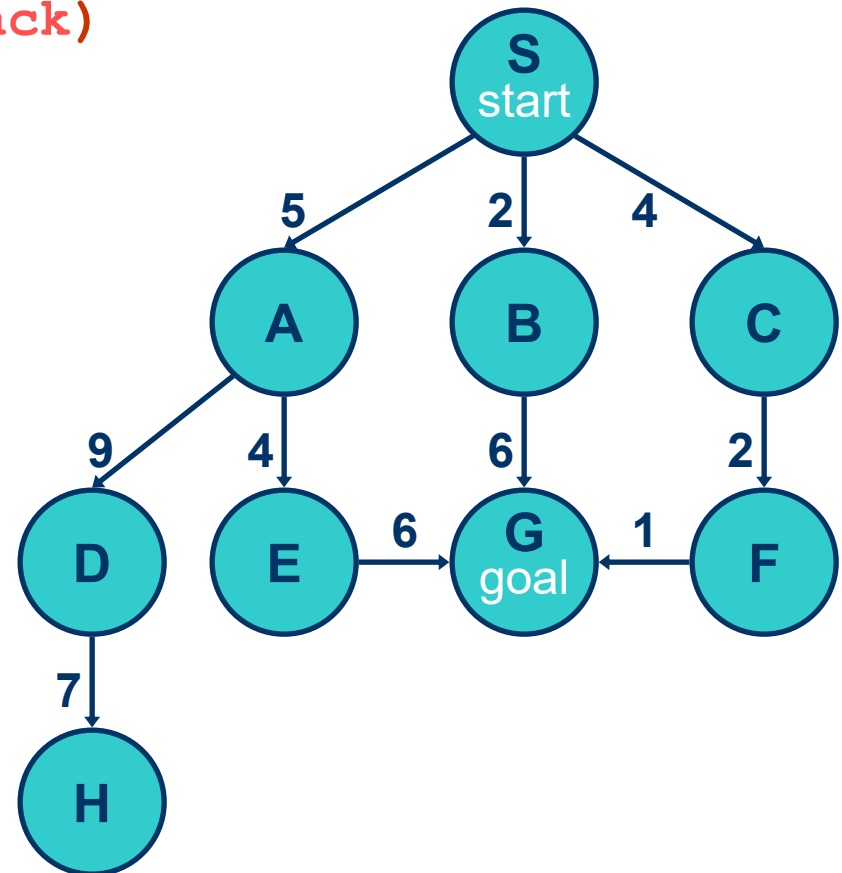


Depth-First Search (DFS)

`generalSearch(problem, stack)`

of nodes tested: 0, expanded: 0

expnd. node	Frontier
	{S}

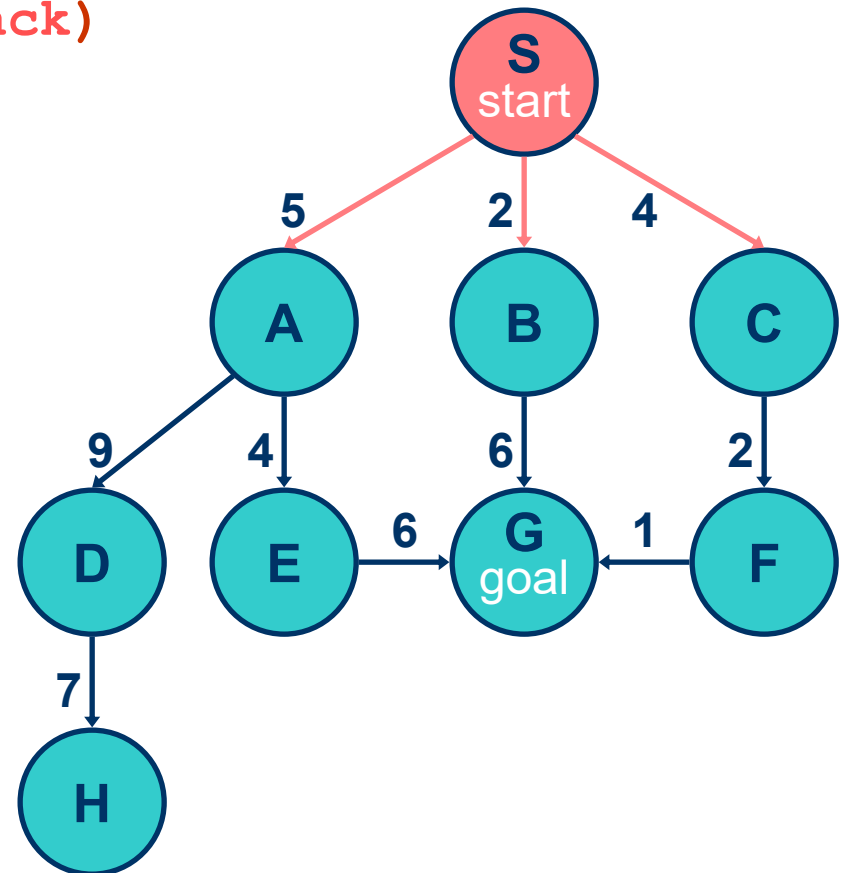


Depth-First Search (DFS)

`generalSearch(problem, stack)`

of nodes tested: 1, expanded: 1

expnd. node	Frontier
	{S}
S not goal	{A,B,C}

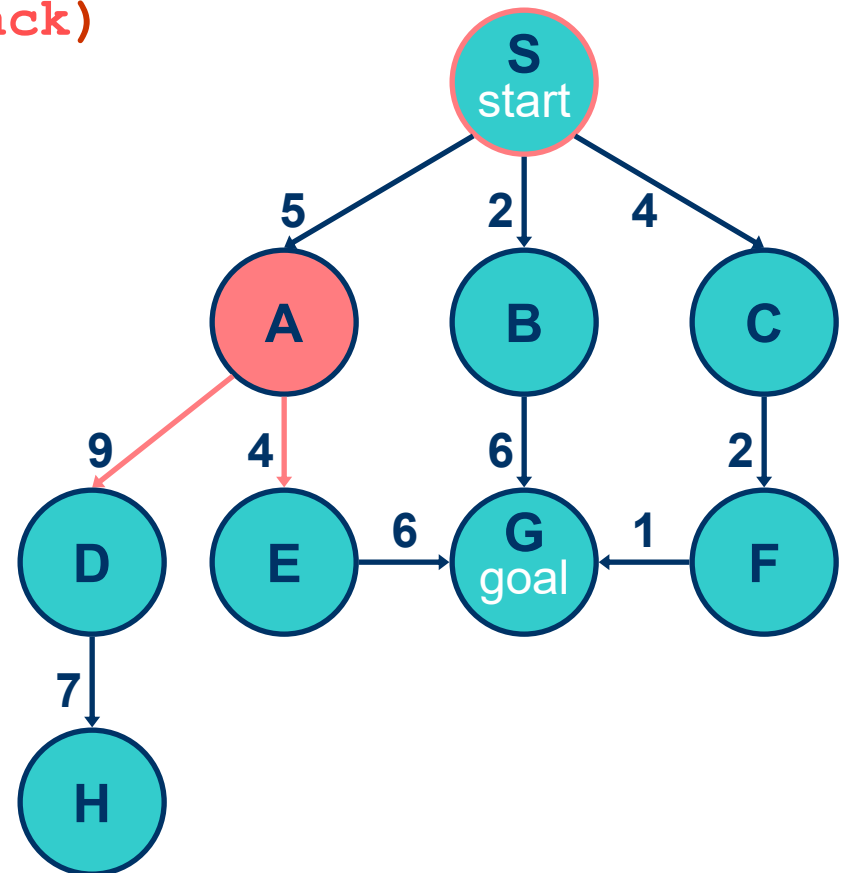


Depth-First Search (DFS)

`generalSearch(problem, stack)`

of nodes tested: 2, expanded: 2

expnd. node	Frontier
	{S}
S	{A,B,C}
A not goal	{D,E,B,C}

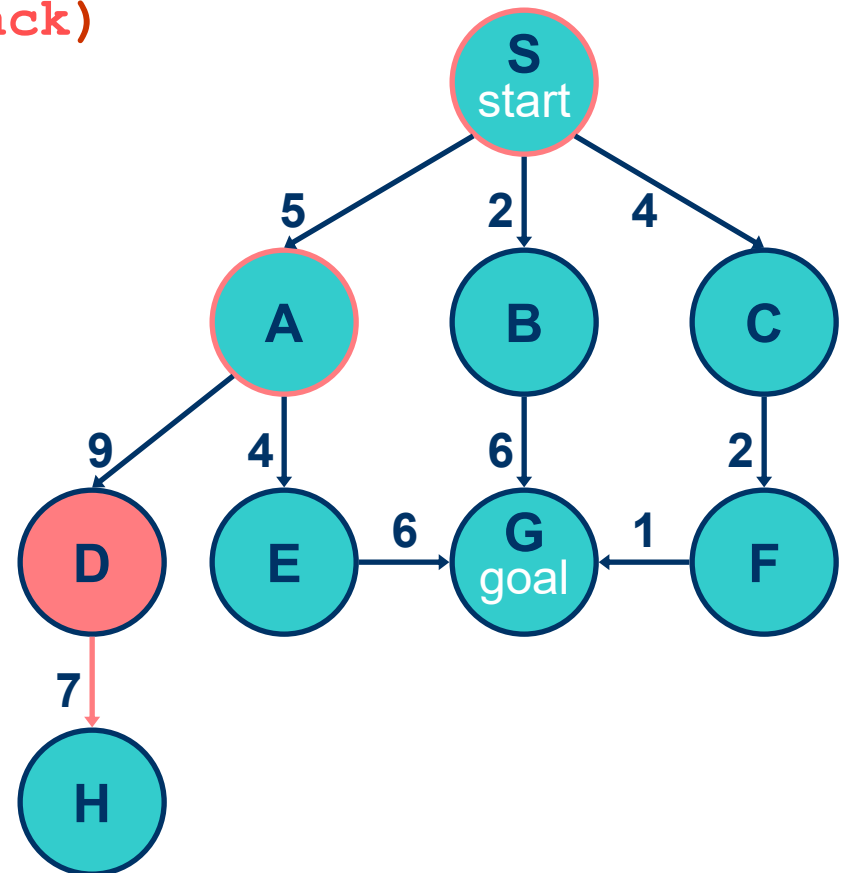


Depth-First Search (DFS)

`generalSearch(problem, stack)`

of nodes tested: 3, expanded: 3

expnd. node	Frontier
	{S}
S	{A,B,C}
A	{D,E,B,C}
D not goal	{ H ,E,B,C}

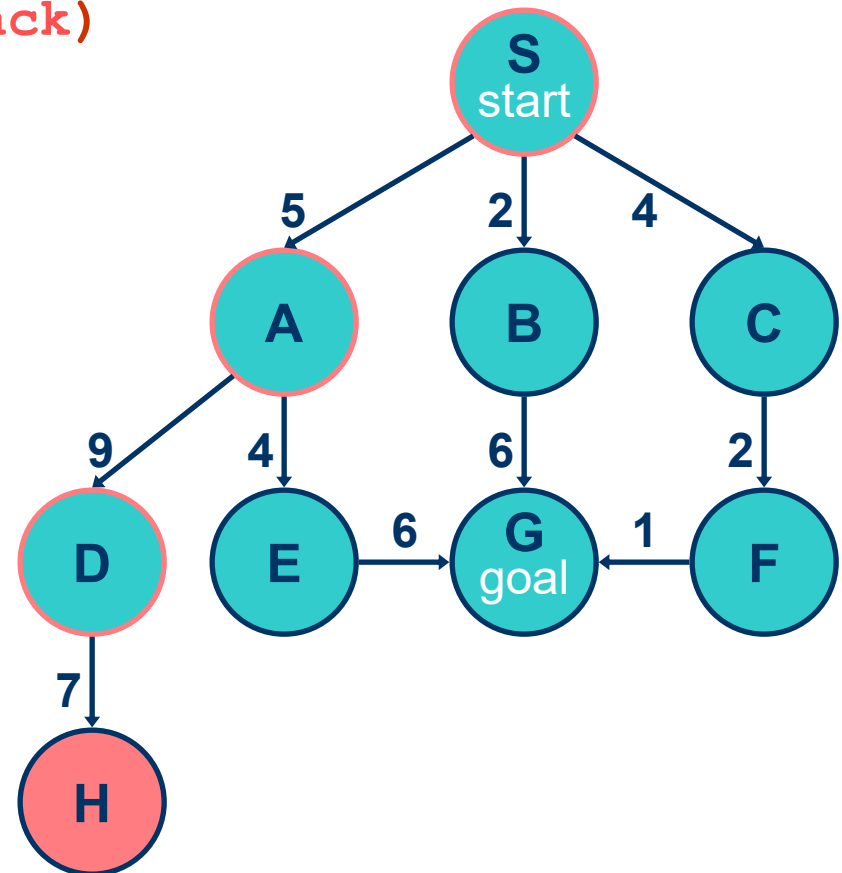


Depth-First Search (DFS)

`generalSearch(problem, stack)`

of nodes tested: 4, expanded: 4

expnd. node	Frontier
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H not goal	{E,B,C}

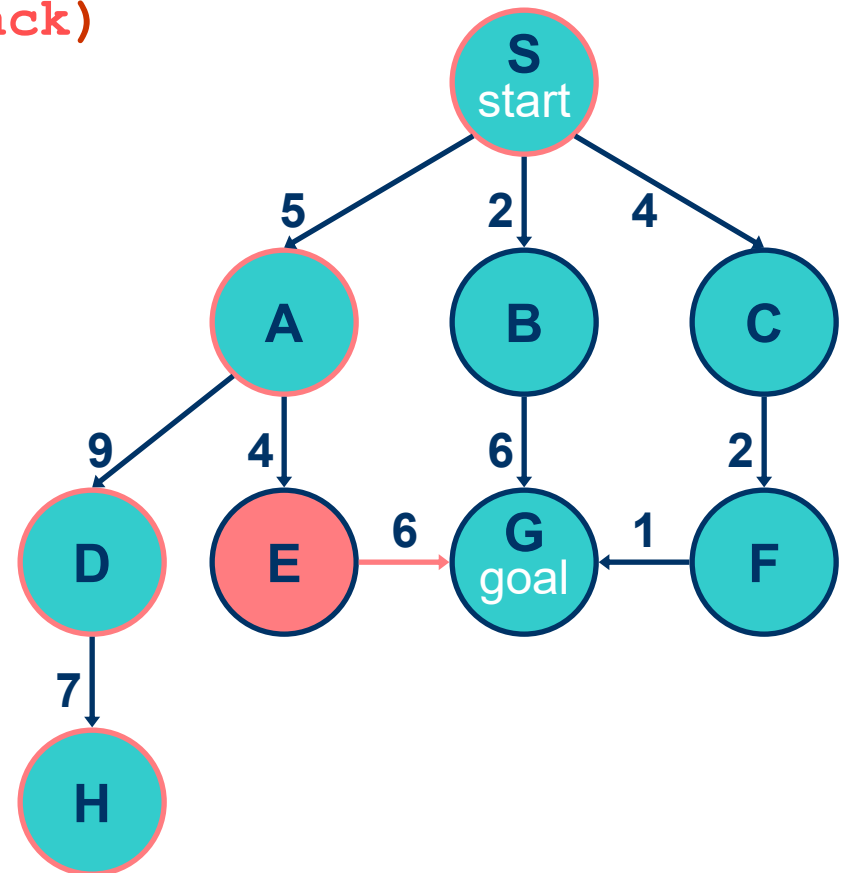


Depth-First Search (DFS)

`generalSearch(problem, stack)`

of nodes tested: 5, expanded: 5

expnd. node	Frontier
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H	{E,B,C}
E not goal	{G,B,C}

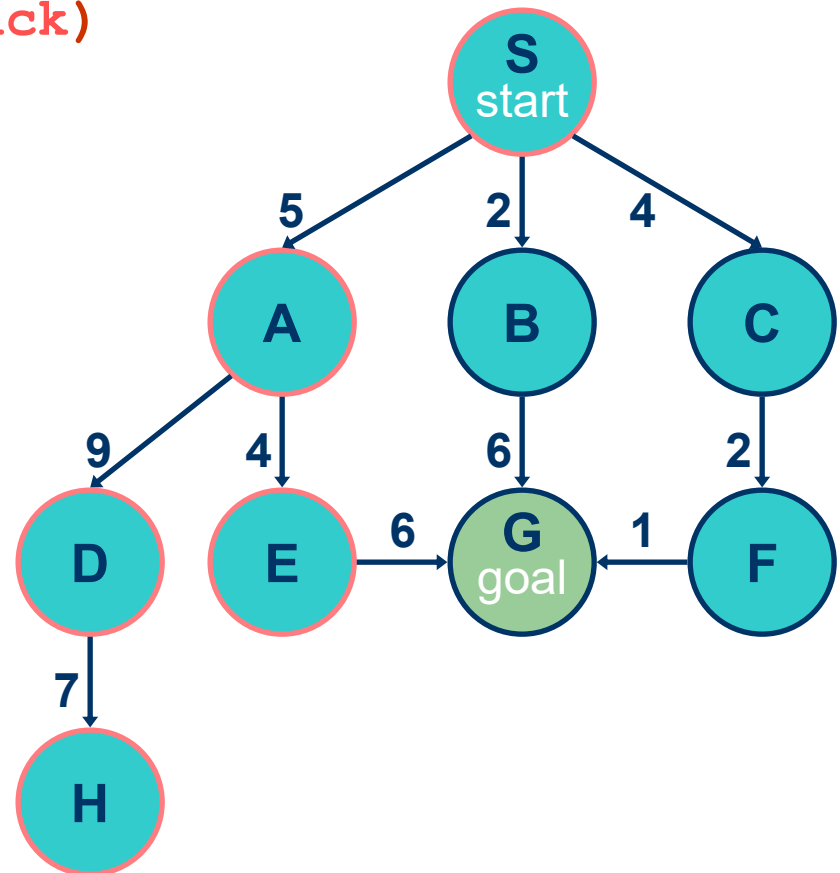


Depth-First Search (DFS)

`generalSearch(problem, stack)`

of nodes tested: 6, expanded: 5

expnd. node	Frontier
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H	{E,B,C}
E	{G,B,C}
G goal	{B,C} no expand

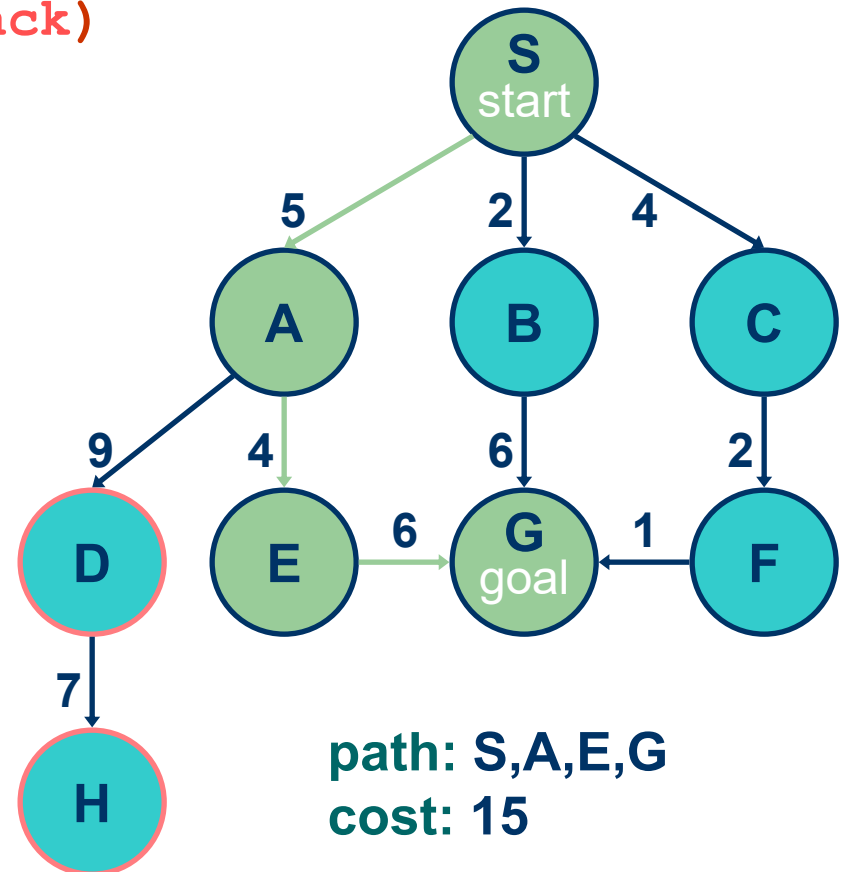


Depth-First Search (DFS)

`generalSearch(problem, stack)`

of nodes tested: 6, expanded: 5

expnd. node	Frontier
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H	{E,B,C}
E	{G,B,C}
G	{B,C}



Depth-First Search (DFS)

- May not terminate without a **depth bound**
i.e., cutting off search below a fixed depth, D
- **Not complete**
 - with or without cycle detection
 - and, with or without a depth cutoff
- **Not optimal / admissible**
- *Can find long solutions quickly if lucky*

Depth-First Search (DFS)

- **Time complexity: $O(b^d)$ exponential**
Space complexity: $O(bd)$ linear
 - d is the depth of the solution
 - b is the branching factor at each non-leaf node
- Performs “**chronological backtracking**”
 - i.e., when search hits a dead end, backs up *one* level at a time
 - problematic if the mistake occurs because of a bad action choice near the top of search tree

Uniform-Cost Search (UCS)

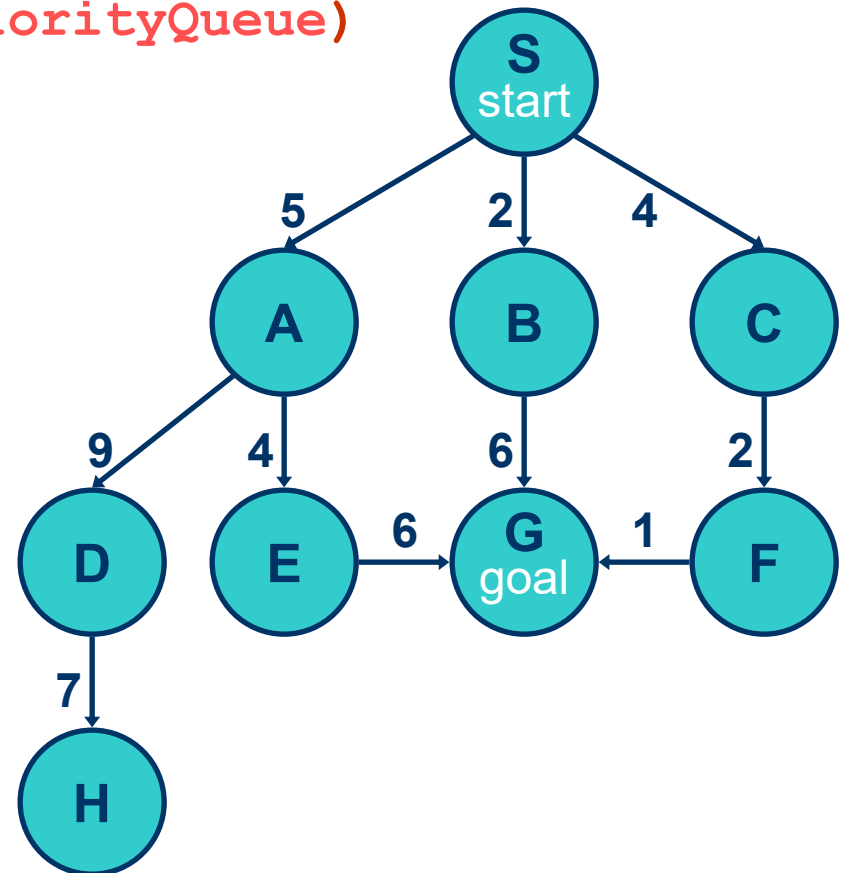
- Use a “**Priority Queue**” to order nodes on the *Frontier* list, sorted by path cost
- Let $g(n)$ = cost of path from start node s to current node n
- Sort nodes by increasing value of g

Uniform-Cost Search (UCS)

`generalSearch(problem, priorityQueue)`

of nodes tested: 0, expanded: 0

expnd. node	Frontier list
	{S}

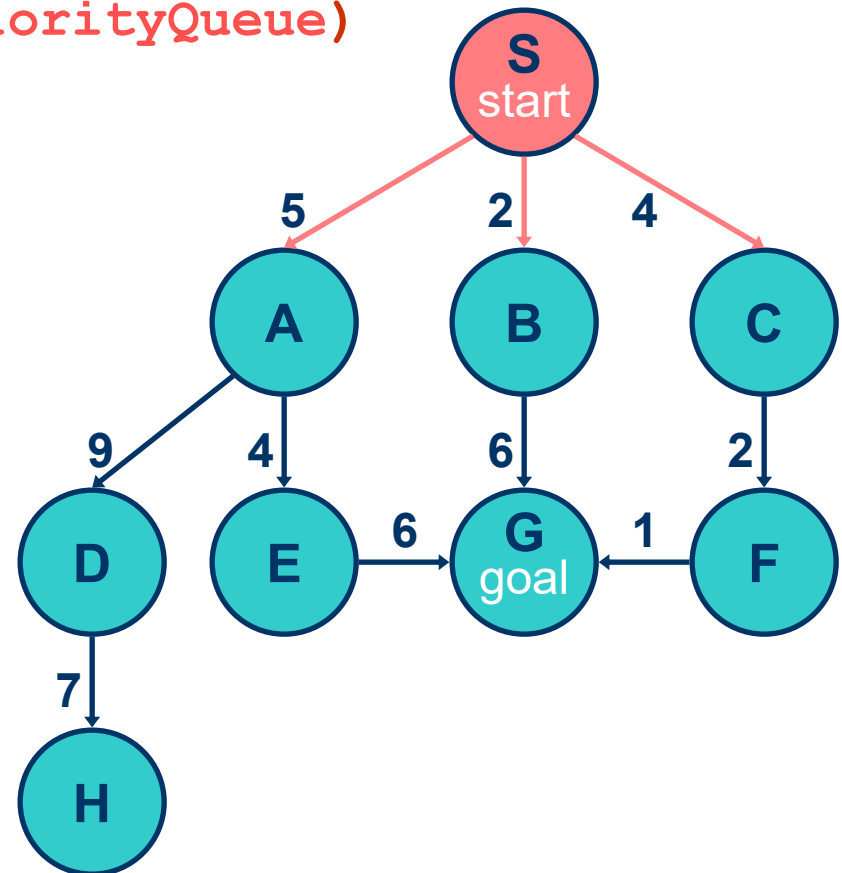


Uniform-Cost Search (UCS)

`generalSearch(problem, priorityQueue)`

of nodes tested: 1, expanded: 1

expnd. node	Frontier list
	{S:0}
S not goal	{B:2,C:4,A:5}

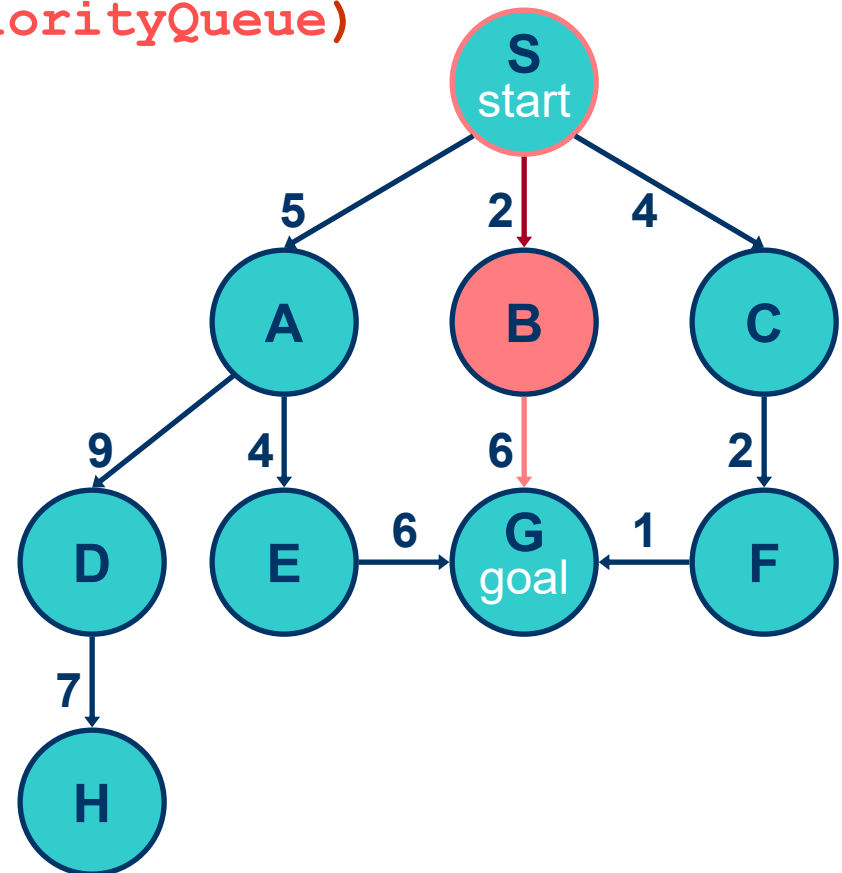


Uniform-Cost Search (UCS)

`generalSearch(problem, priorityQueue)`

of nodes tested: 2, expanded: 2

expnd. node	Frontier list
	{S}
S	{B:2,C:4,A:5}
B not goal	{C:4,A:5,G:2+6}

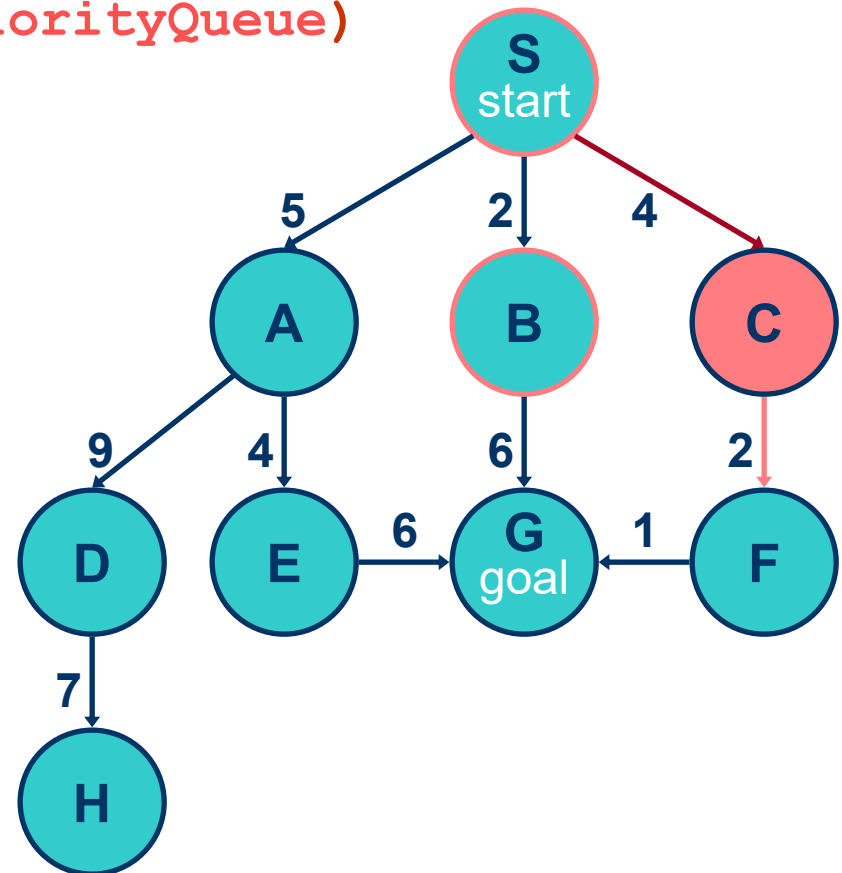


Uniform-Cost Search (UCS)

`generalSearch(problem, priorityQueue)`

of nodes tested: 3, expanded: 3

expnd. node	Frontier list
	{S}
S	{B:2,C:4,A:5}
B	{C:4,A:5,G:8}
C not goal	{A:5,F:4+2,G:8}

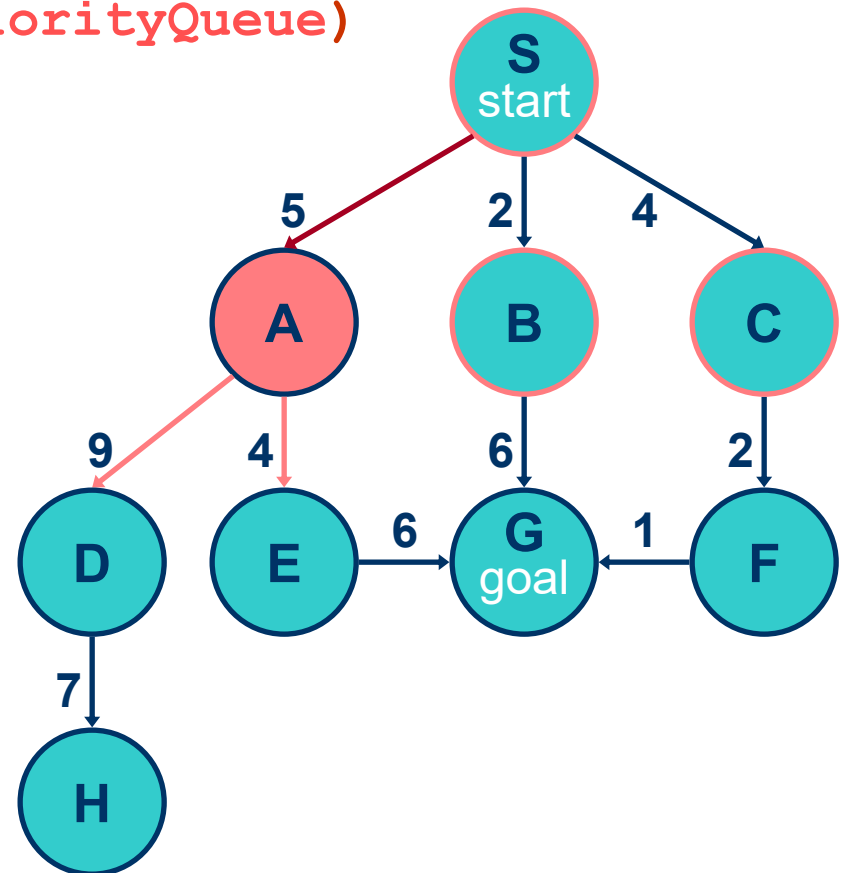


Uniform-Cost Search (UCS)

`generalSearch(problem, priorityQueue)`

of nodes tested: 4, expanded: 4

expnd. node	Frontier list
	{S}
S	{B:2,C:4,A:5}
B	{C:4,A:5,G:8}
C	{A:5,F:6,G:8}
A not goal	{F:6,G:8,E:5+4, D:5+9}

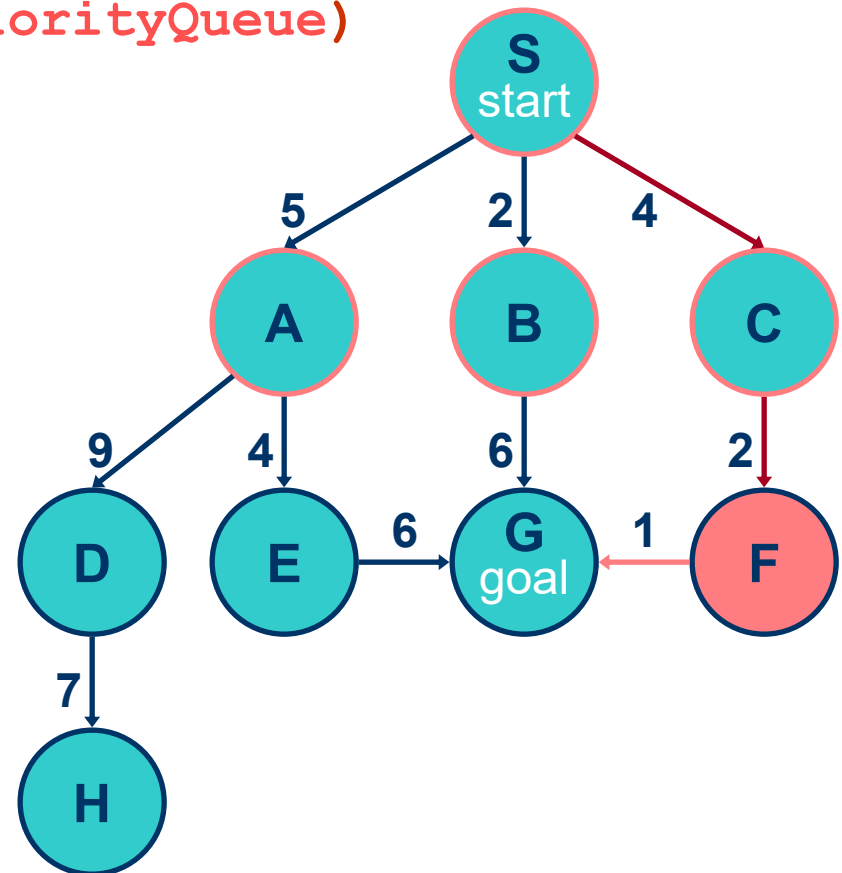


Uniform-Cost Search (UCS)

`generalSearch(problem, priorityQueue)`

of nodes tested: 5, expanded: 5

expnd. node	Frontier list
	{S}
S	{B:2,C:4,A:5}
B	{C:4,A:5,G:8}
C	{A:5,F:6,G:8}
A	{F:6,G:8,E:9,D:14}
F not goal	{G:4+2+1, G:8, E:9, D:14}

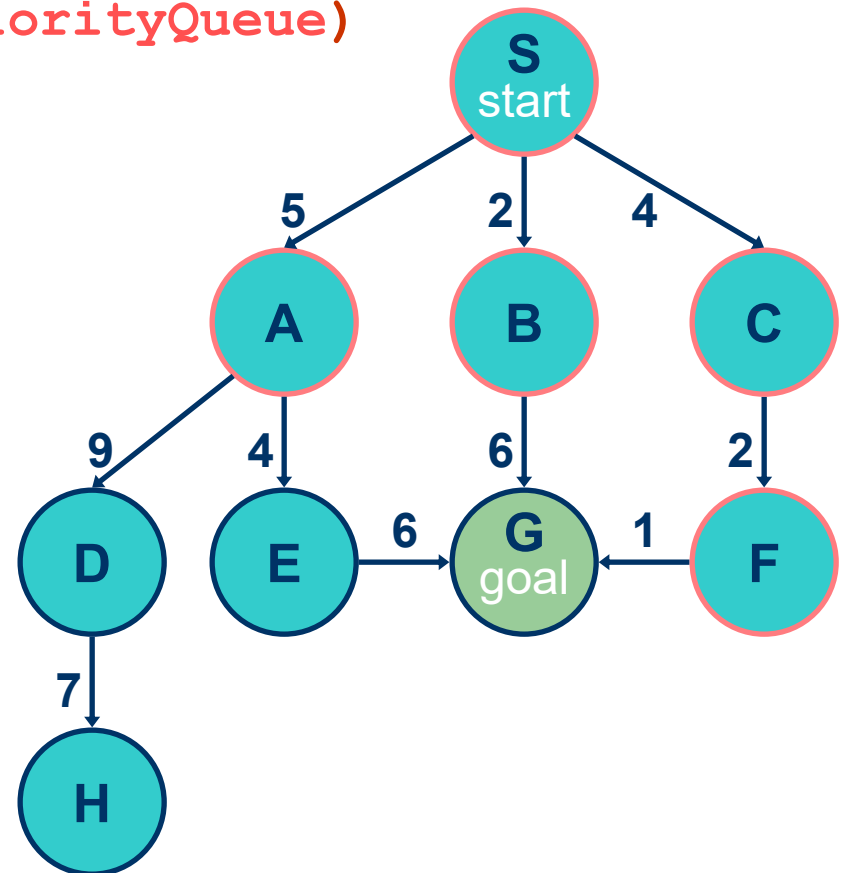


Uniform-Cost Search (UCS)

`generalSearch(problem, priorityQueue)`

of nodes tested: 6, expanded: 5

expnd. node	Frontier list
	{S}
S	{B:2,C:4,A:5}
B	{C:4,A:5,G:8}
C	{A:5,F:6,G:8}
A	{F:6,G:8,E:9,D:14}
F	{G:7,G:8,E:9,D:14}
G goal	{G:8,E:9,D:14} no expand

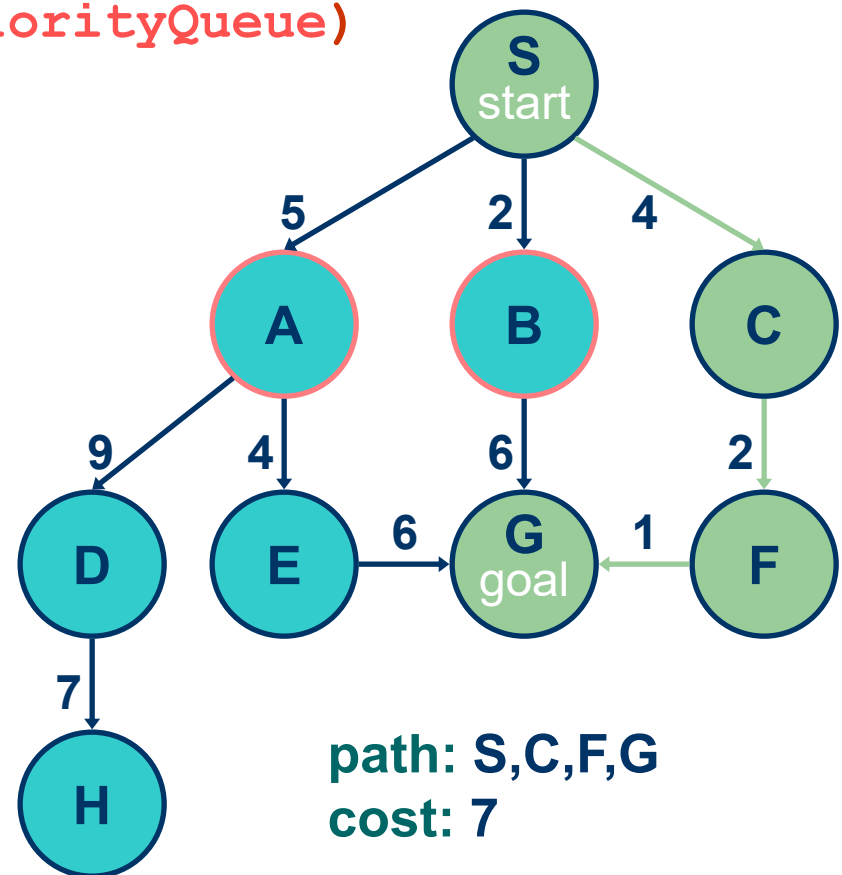


Uniform-Cost Search (UCS)

`generalSearch(problem, priorityQueue)`

of nodes tested: 6, expanded: 5

expnd. node	Frontier list
	{S}
S	{B:2,C:4,A:5}
B	{C:4,A:5,G:8}
C	{A:5,F:6,G:8}
A	{F:6,G:8,E:9,D:14}
F	{G:7,G:8,E:9,D:14}
G	{G:8,E:9,D:14}



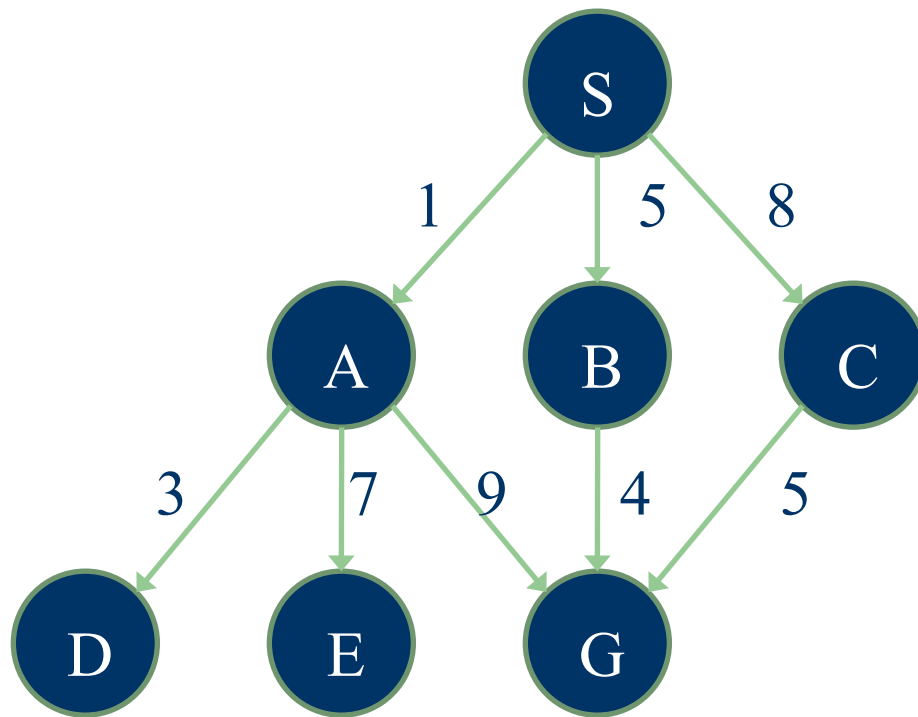
Uniform-Cost Search (UCS)

- Called *Dijkstra's Algorithm* in the algorithms literature
- Similar to *Branch and Bound Algorithm* in Operations Research literature
- Complete
- Optimal / Admissible
 - requires that the goal test is done when a node is **removed** from the *Frontier* rather than when the node is generated by its parent node

Uniform-Cost Search (UCS)

- **Time and space complexity: $O(b^d)$ (i.e., exponential)**
 - d is the depth of the solution
 - b is the branching factor at each non-leaf node
- **More precisely, time and space complexity is $O(b^{C^*/\epsilon})$ where C^* is the best goal path cost**

Example



How are nodes expanded by

- Depth First Search
- Breadth First Search
- Uniform Cost Search

Are the solutions the same?

Knowledge Representation

Lecture 3

Outline



1

- Knowledge Representation
- Properties of Representation Systems
- Approaches to Knowledge Representation
- Knowledge Representation Types

Knowledge representation



A subarea of Artificial Intelligence concerned with understanding, designing, and implementing ways of representing information in computers so that programs (agents) can use this information

- to derive information that is implied by it,
- to converse with people in natural languages,
- to decide what to do next
- to plan future activities,
- to solve problems in areas that normally require human expertise.



- **What to Represent?**

Let us first consider what kinds of knowledge might need to be represented in AI systems:

- **Objects**

- -- Facts about objects in our world domain. *e.g.* Guitars have strings, trumpets are brass instruments.

- **Events**

- -- Actions that occur in our world. *e.g.* Steve played the guitar in Frank Zappa's Band.

- **Performance**

- -- A behaviour like *playing the guitar* involves knowledge about how to do things.

- **Meta-knowledge**

- -- knowledge about what we know.

Properties of Knowledge Representation

- Representational adequacy
 - ability to represent the required knowledge
- Inferential adequacy
 - ability to manipulate knowledge
- Inferential efficiency
 - ability to respond with limited resources (time, storage)
- Acquisitional efficiency
 - ability to acquire new knowledge

Approaches to Knowledge representation

1. Simple Relational Knowledge

The simplest way of storing facts is to use a relational method where each fact about a set of objects is set out systematically in columns.

Musician	Style	Instrument	Age
Miles Davis	Jazz	Trumpet	deceased
John Zorn	Avant Garde	Saxophone	35
Frank Zappa	Rock	Guitar	deceased
John McLaughlin	Jazz	Guitar	47

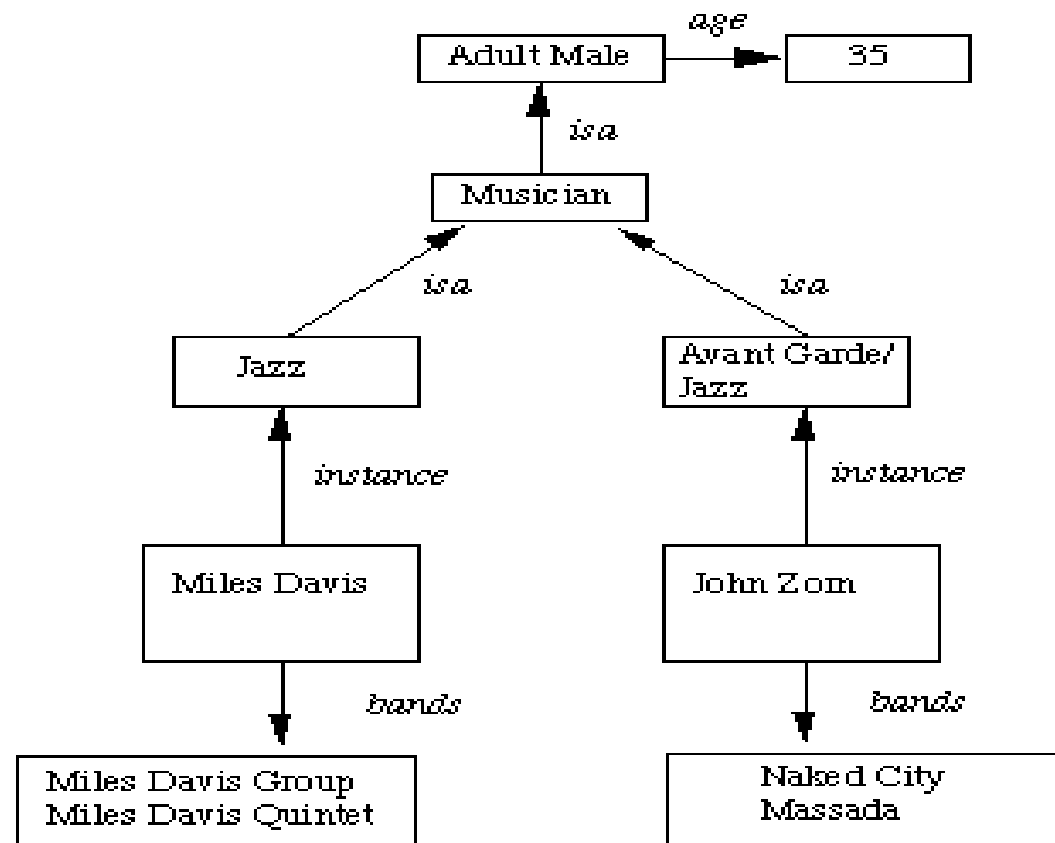
2. Inheritable Knowledge

Relational knowledge is made up of objects consisting of attributes and associated values.

Elements inherit values from being members of a class.

Boxed nodes -- objects and values of Attributes.

Arrows -- point from object to its value.



3. Inferential Knowledge

Represent knowledge as formal logic.

For example, *All dogs have tails, can be represented as*

$$\forall(x) : \text{dog}(x) \rightarrow \text{has-a-tail}(x)$$

4. Procedural Knowledge

- Knowledge encoded in some procedures -
 - small programs that know how to do specific things, how to proceed
 - In this approach, one important rule is used which is **If-Then rule**

Types of Knowledge representation

The Knowledge Representation is generally of four types -

- Logical Representations

Logical representation means drawing a conclusion based on various conditions.

- Semantic Networks

In Semantic networks, we can represent our knowledge in the form of graphical networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects.

- Production Rules

Production rules system consist of (**condition**, **action**) pairs which mean, "If condition then action".

- Frames

A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world.

Knowledge Representation

Lecture -3

new his

What is knowledge representation?

- Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world. **But how machines do all these things comes under knowledge representation and reasoning.** Hence we can describe Knowledge representation as following:
- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.
- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.
- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

What to Represent:

- Following are the kind of knowledge which needs to be represented in AI systems:
- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behavior which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

Types of knowledge

1. Declarative Knowledge:

- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.
- It is also called **descriptive knowledge** and expressed in declarative sentences.
- It is simpler than procedural language.

Types of knowledge

2. Procedural Knowledge

- It is also known as **imperative knowledge**.
- Procedural knowledge is a type of knowledge which is responsible for knowing **how to do something**.
- It can be directly applied to any task.
- It includes **rules**, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied.

3. Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta-knowledge.

Types of knowledge

4. Heuristic knowledge:

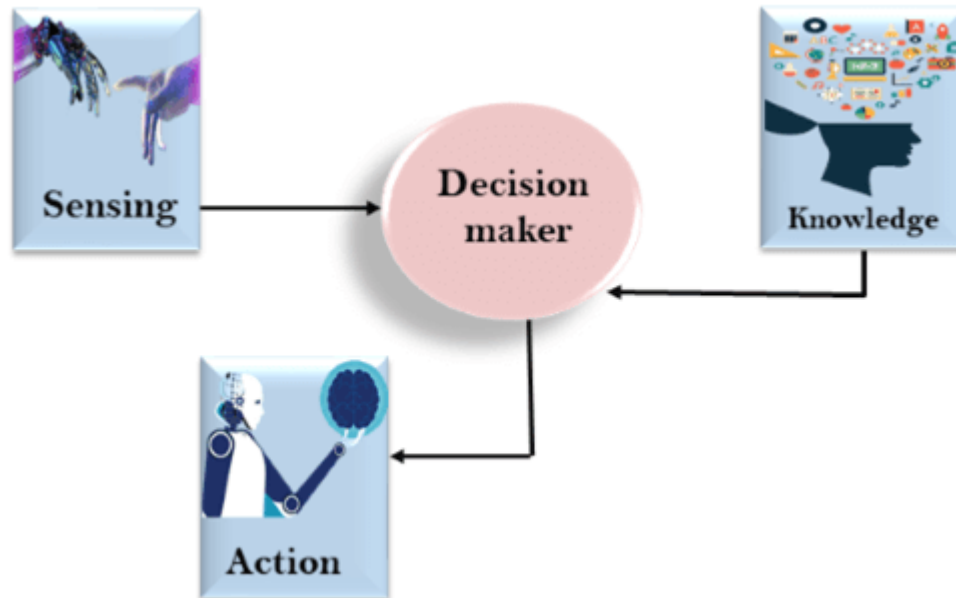
- Heuristic knowledge is representing knowledge of some experts in a field or subject.
- Heuristic knowledge is **rules of thumb** based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

5. Structural knowledge:

- Structural knowledge is **basic knowledge to problem-solving**.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

The relation between knowledge and intelligence:

- Knowledge plays an important role in demonstrating intelligent behavior in AI agents. An agent is only able to accurately act on some input when he has some **knowledge** or experience about that input.
- Let's suppose if you met some person who is speaking in a language which you don't know, then how you will be able to act on that. The same thing applies to the intelligent behavior of the agents.



Approaches to knowledge representation

- There are mainly four approaches to knowledge representation, which are given below:

1. **Simple relational knowledge:**

- It is the simplest way of storing facts which uses the **relational** method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in **database systems** where the relationship between different entities is represented.
- This approach has little opportunity for **inference**.

Approaches to knowledge representation

- **Example:** The following is the simple relational knowledge representation.

Player	Weight	Age
Player1	65	23
Player2	58	18
Player3	75	24

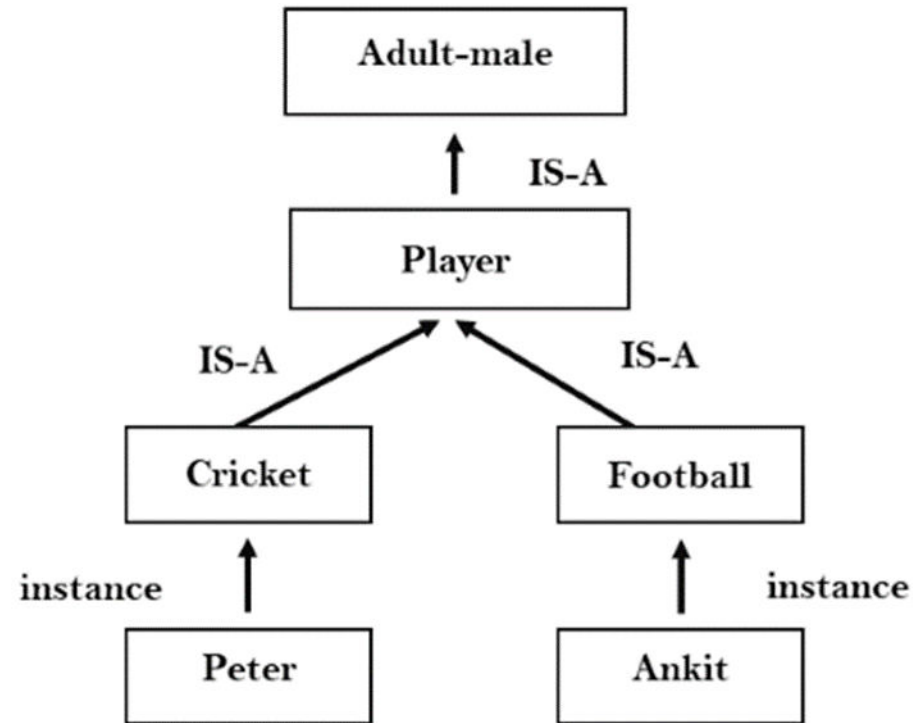
Approaches to knowledge representation

2. Inheritable knowledge:

- In the inheritable knowledge approach, all data must be stored into a **hierarchy** of classes.
- All classes should be arranged in a generalized form or a hierarchal manner.
- In this approach, we apply **inheritance** property.
- **Elements inherit values from other members of a class.**
- This approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation.
- Every individual frame can represent the collection of attributes and its value.
- In this approach, objects and values are represented in Boxed nodes.
- We use Arrows which point from objects to their values.

Inheritable knowledge

- Example:



Inferential knowledge

- Inferential knowledge approach represents knowledge in the form of **formal logics**.
- This approach can be used to derive more facts.
- It guaranteed correctness.
- **Example:** Let's suppose there are two statements:
 - Marcus is a man
 - All men are mortalThen it can represent as;

man(Marcus)

$\forall x = \text{man}(x) \text{ -----} \rightarrow \text{mortal}(x)$

Procedural knowledge

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.
- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

Requirements for knowledge Representation system

- A good knowledge representation system must possess the following properties.

1. Representational Accuracy:

KR system should have the ability to represent all kind of required knowledge.

2. Inferential Adequacy:

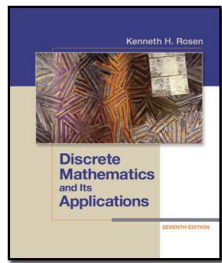
KR system should have ability to manipulate the representational structures to produce new knowledge corresponding to existing structure.

3. Inferential Efficiency:

The ability to direct the inferential knowledge mechanism into the most productive directions by storing appropriate guides.

4. Acquisitional efficiency:

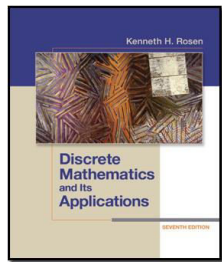
The ability to acquire the new knowledge easily using automatic methods.



Propositional Logic

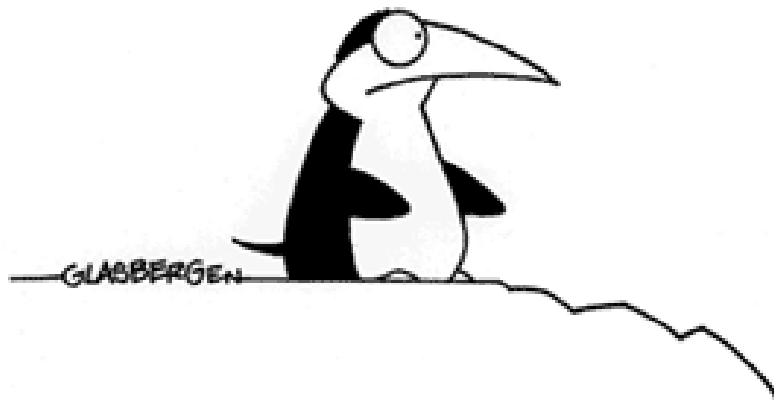
TOPICS

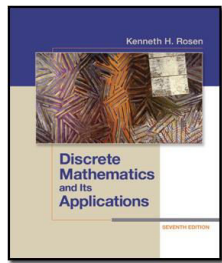
- Propositional Logic
- Logical Operations
- Equivalences



Logic?

**Penguins are black and white
Some old TV shows are black and white
Therefore
Some penguins are old TV shows**





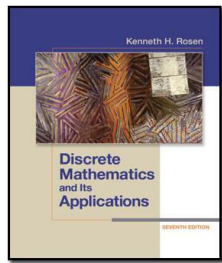
What is logic?

Logic is a truth-preserving system of inference

Truth-preserving:
If the initial
statements are
true, the inferred
statements will
be true

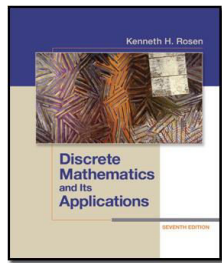
System: a set of
mechanistic
transformations, based
on syntax alone

Inference: the process of
deriving (inferring) new
statements from old
statements



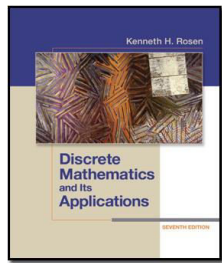
Propositional Logic

- A *proposition* is a statement that is either true or false
- Examples:
 - This class is CS122 (true)
 - Today is Sunday (false)
 - It is currently raining in Singapore (???)
- Every proposition is true or false, but its *truth value* (true or false) may be unknown



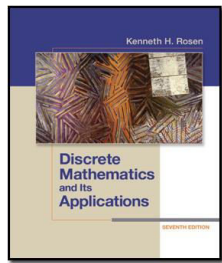
Propositional Logic (II)

- A propositional statement is one of:
 - A simple proposition (atomic proposition)
 - denoted by a capital letter, e.g. 'A'.
 - A negation of a propositional statement
 - e.g. $\neg A$: “not A”
 - Two propositional statements joined by a *connective*
 - e.g. $A \wedge B$: “A and B” (complex proposition)
 - e.g. $A \vee B$: “A or B”
 - If a connective joins complex statements, parenthesis are added
 - e.g. $A \wedge (B \vee C)$



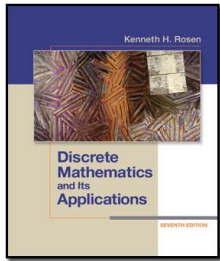
Logical negation

- Negation of proposition A is $\neg A$
- A: It is snowing.
- $\neg A$: It is not snowing
- A: Newton knew Einstein.
- $\neg A$: Newton did not know Einstein.
- A: I am not registered for CS195.
- $\neg A$: I am registered for CS195.



Negation Truth Table

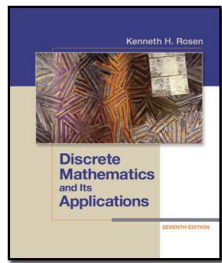
A	$\neg A$
0	1
1	0



Logical and (*conjunction*)

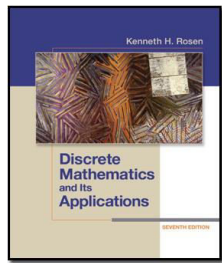
- Conjunction of A and B is $A \wedge B$
 - A: CS160 teaches logic.
 - B: CS160 teaches Java.
 - $A \wedge B$: CS160 teaches logic and Java.

- Combining conjunction and negation
 - A: I like fish.
 - B: I like sushi.
 - I like fish but not sushi: $A \wedge \neg B$



Truth Table for Conjunction

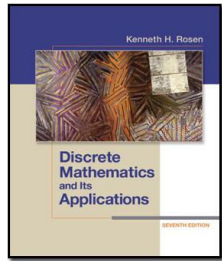
A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1



Logical or (*disjunction*)

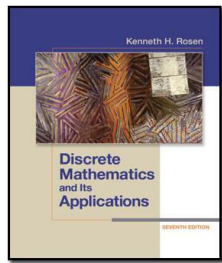
- Disjunction of A and B is $A \vee B$
 - A: Today is Friday.
 - B: It is snowing.
 - $A \vee B$: Today is Friday or it is snowing.

- This statement is true if any of the following hold:
 - Today is Friday
 - It is snowing
 - Both
- Otherwise it is false



Truth Table for Disjunction

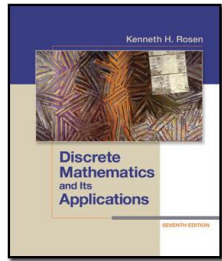
A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1



Exclusive Or

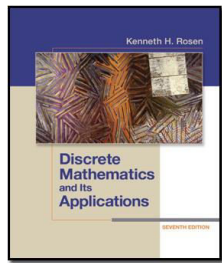
- The “or” connective \vee is inclusive: it is true if either *or both* arguments are true
- There is also an exclusive or (either or): \oplus

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



Confusion over Inclusive OR and Exclusive OR

- Restaurants typically let you pick one (either soup or salad, not both) when they say “The entrée comes with a soup or salad”.
 - Use exclusive OR to write as a logic proposition
- Give two interpretations of the sentence using inclusive OR and exclusive OR:
 - Students who have taken calculus or intro to programming can take this class

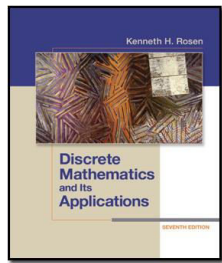


Conditional & Biconditional Implication

- The conditional implication connective is \rightarrow
- The biconditional implication connective is \leftrightarrow
- These, too, are defined by truth tables

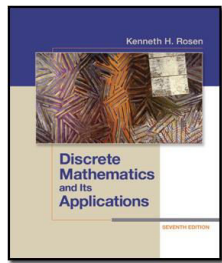
<i>A</i>	<i>B</i>	$A \rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

<i>A</i>	<i>B</i>	$A \leftrightarrow B$
0	0	1
0	1	0
1	0	0
1	1	1



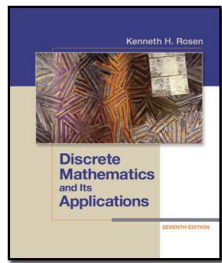
Conditional implication

- A: A programming homework is due.
- B: It is Tuesday.
- $A \rightarrow B$:
 - If a programming homework is due, then it must be Tuesday.
- Is this the same?
 - If it is Tuesday, then a programming homework is due.



Bi--conditional

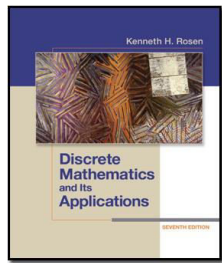
- A: You can take the flight.
- B: You have a valid ticket.
- $A \leftrightarrow B$
 - You can take the flight if and only if you have a valid ticket (and vice versa).



Compound Truth Tables

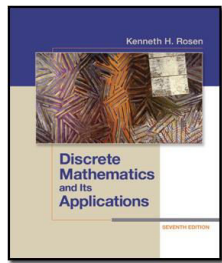
- Truth tables can also be used to determine the truth values of compound statements, such as $(A \vee B) \wedge (\neg A)$ (fill this as an exercise)

A	B	$\neg A$	$A \vee B$	$(A \vee B) \wedge (\neg A)$
0	0	1	0	0
0	1	1	1	1
1	0	0	1	0
1	1	0	1	0



Tautology and Contradiction

- A *tautology* is a compound proposition that is always true.
- A *contradiction* is a compound proposition that is always false.
- A *contingency* is neither a tautology nor a contradiction.
- A compound proposition is *satisfiable* if there is at least one assignment of truth values to the variables that makes the statement true.



Examples

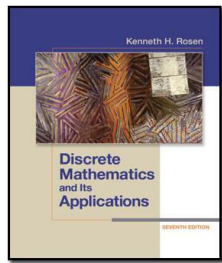
A	$\neg A$	$A \vee \neg A$	$A \wedge \neg A$
0	1	1	0
1	0	1	0

Result is always
true, no matter
what A is

Therefore, it is a
tautology

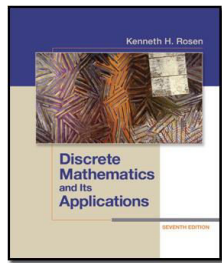
Result is always
false, no matter
what A is

Therefore, it is a
contradiction



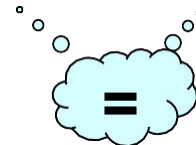
Logical Equivalence

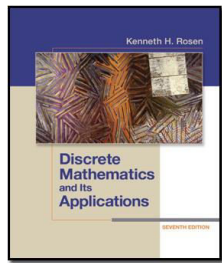
- Two compound propositions, p and q , are logically equivalent if $p \leftrightarrow q$ is a tautology.
- Notation: $p \equiv q$
- De Morgan's Laws:
 - $\neg (p \wedge q) \equiv \neg p \vee \neg q$
 - $\neg (p \vee q) \equiv \neg p \wedge \neg q$
- How so? Let's build a truth table!



Prove $\neg(p \wedge q) \equiv \neg p \vee \neg q$

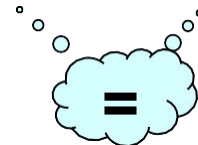
p	q	$\neg p$	$\neg q$	$(p \wedge q)$	$\neg(p \wedge q)$	$\neg p \vee \neg q$
0	0	1	1	0	1	1
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	0

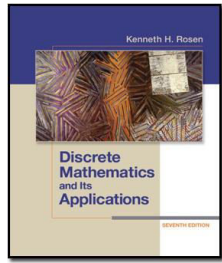




Show $\neg(p \vee q) \equiv \neg p \wedge \neg q$

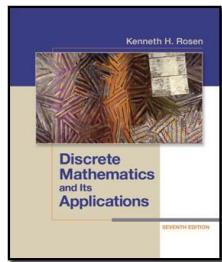
p	q	$\neg p$	$\neg q$	$(p \vee q)$	$\neg(p \vee q)$	$\neg p \wedge \neg q$
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0





Other Equivalences

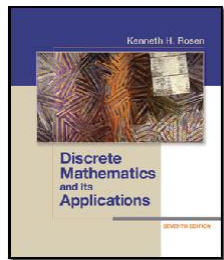
- Show $p \rightarrow q \equiv \neg p \vee q$
- Show Distributive Law:
 - $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$



Show $p \rightarrow q \equiv \neg p \vee q$

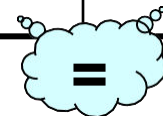
p	q	$\neg p$	$p \rightarrow q$	$\neg p \vee q$
0	0	1	1	1
0	1	1	1	1
1	0	0	0	0
1	1	0	1	1

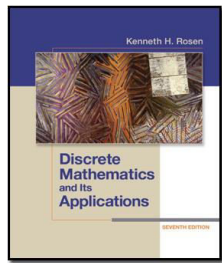




Show $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

p	q	r	$q \wedge r$	$p \vee q$	$p \vee r$	$p \vee (q \wedge r)$	$(p \vee q) \wedge (p \vee r)$
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	1	0	0	1	0	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

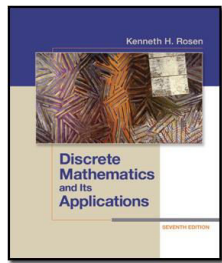




More Equivalences

Equivalence	Name
$p \wedge T \equiv p$ $p \vee F \equiv p$	Identity
$p \wedge q \equiv q \wedge p$ $p \vee q \equiv q \vee p$	Commutative
$p \vee (p \wedge q) \equiv p$ $p \wedge (p \vee q) \equiv p$	Absorption

See Rosen for more.



Equivalences with Conditionals and Biconditionals, Precedence

■ Conditionals

- $p \rightarrow q \equiv \neg p \vee q$

- $p \rightarrow q \equiv \neg q \rightarrow \neg p$

- $\neg(p \rightarrow q) \equiv p \wedge \neg q$

■ Biconditionals

- $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

- $p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$

- $\neg(p \leftrightarrow q) \equiv p \leftrightarrow \neg q$

■ Precedence: (Rosen chapter 1, table 8)

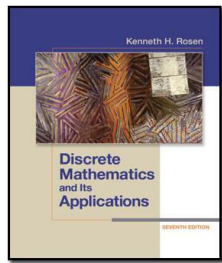
- \neg highest

- \wedge higher than \vee

- \wedge and \vee higher than \rightarrow and \leftrightarrow

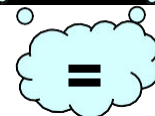
- equal precedence: left to right

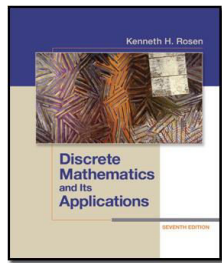
- $()$ used to define priority, and create clarity



Prove Biconditional Equivalence

p	q	$\neg q$	$p \leftrightarrow q$	$\neg(p \leftrightarrow q)$	$p \leftrightarrow \neg q$
0	0	1	1	0	0
0	1	0	0	1	1
1	0	1	0	1	1
1	1	0	1	0	0





Contrapositive

- The *contrapositive* of an implication $p \rightarrow q$ is:

$$\neg q \rightarrow \neg p$$

The contrapositive is equivalent to the original implication.

Prove it!

so now we have:

$$p \rightarrow q \equiv \neg p \vee q \equiv \neg q \rightarrow \neg p$$