Advanced Artificial Intelligence Concepts and Techniques

Press Space for next page →

Dr. Khalil Chebil

Table of Contents

Chapter 1: Introduction to Artificial Intelligence

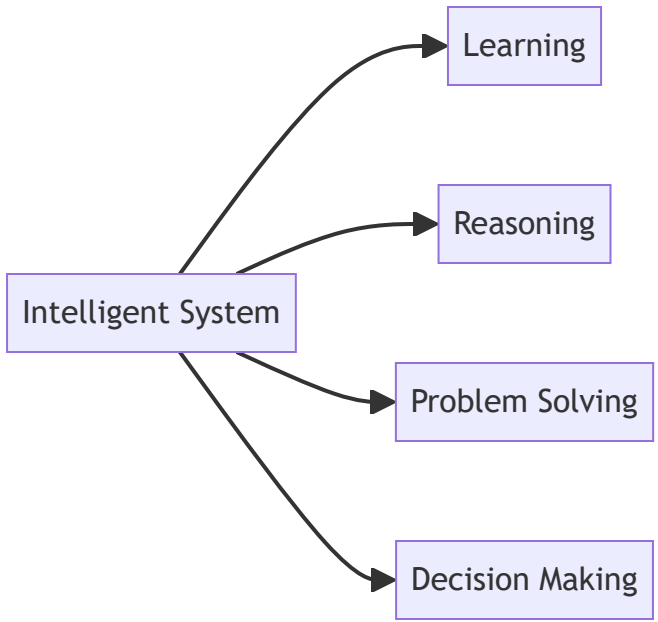
What is intelligence?

Intelligence involves **sensing**, **reasoning**, and **acting**.

- The ability to use reason to solve problems
- The ability to learn from experience
- The ability to acquire knowledge
- The ability to respond quickly and successfully to a new situation

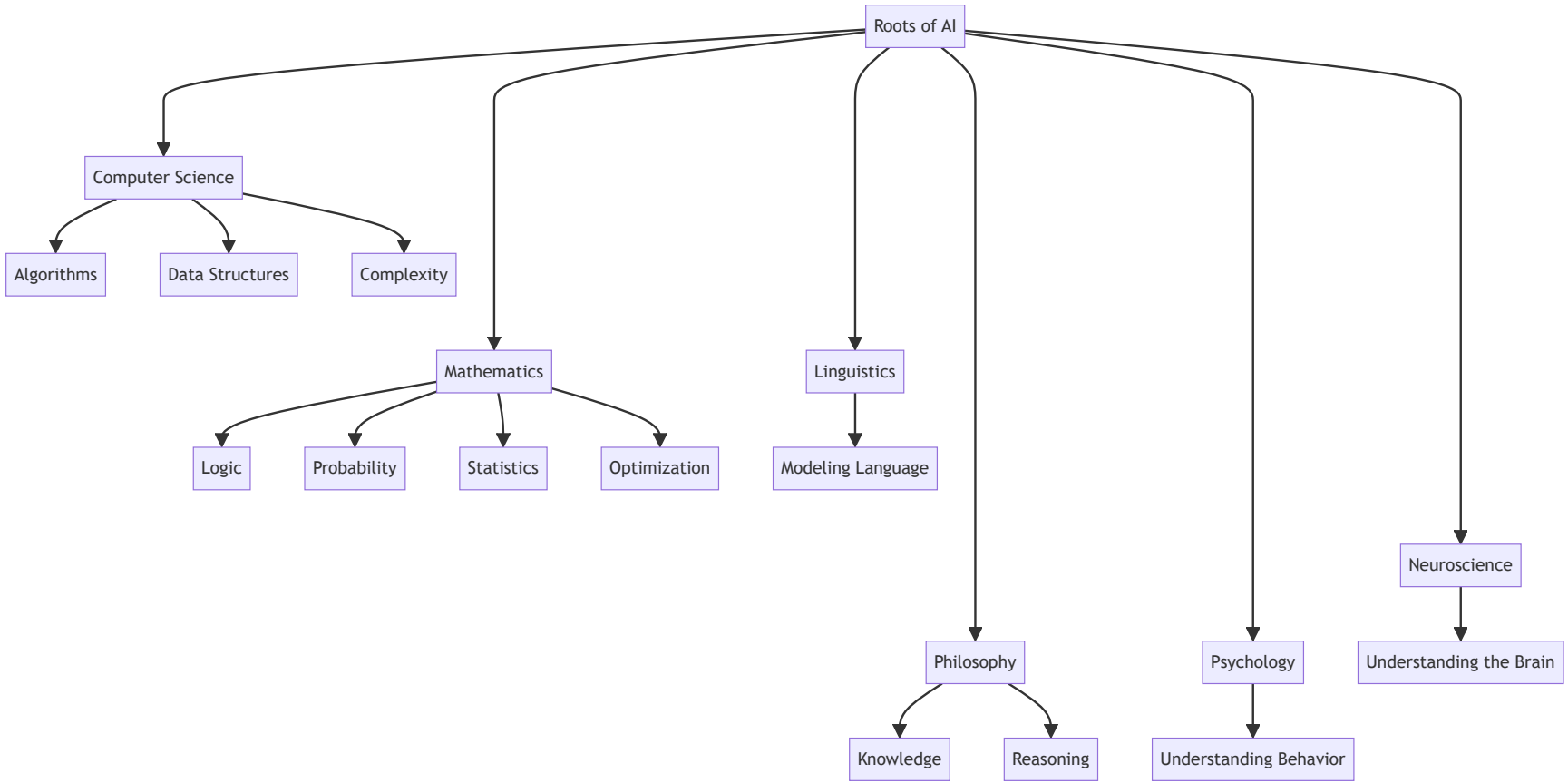
What is Artificial Intelligence?

A branch in computer science that is concerned with **the automation of intelligent behaviors**.

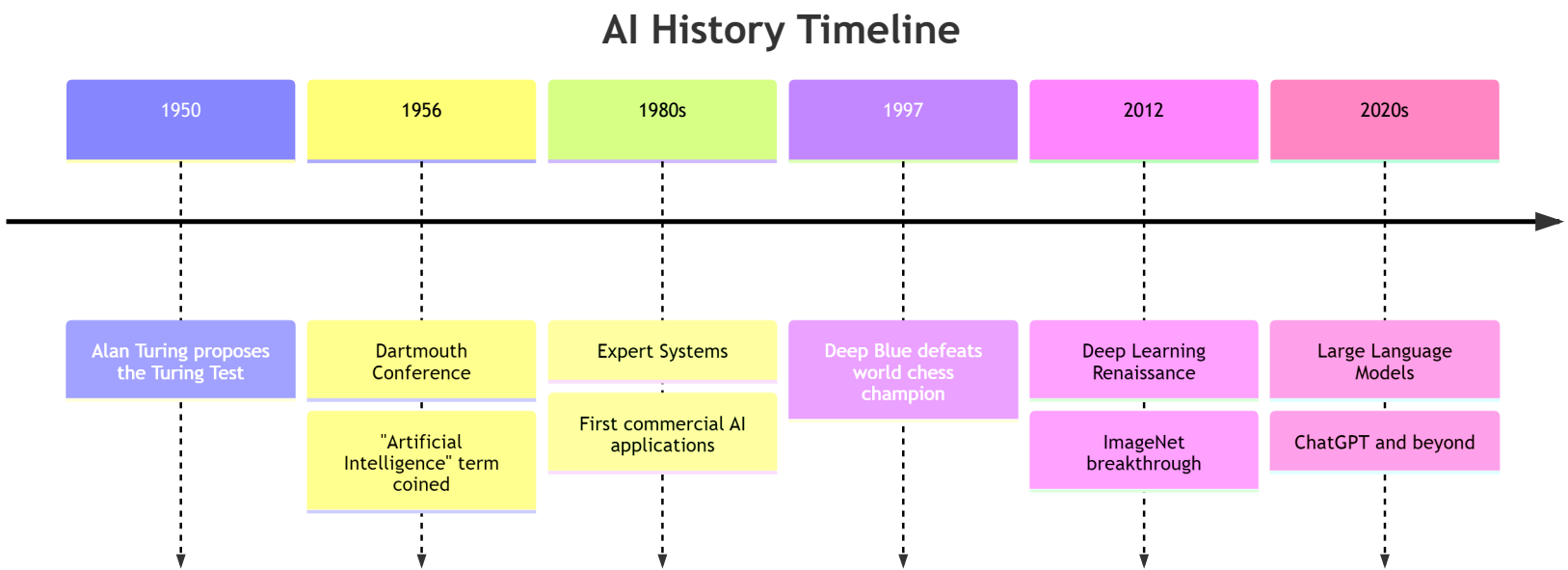


Such as: Speech recognition, Visual perception, Language translation...

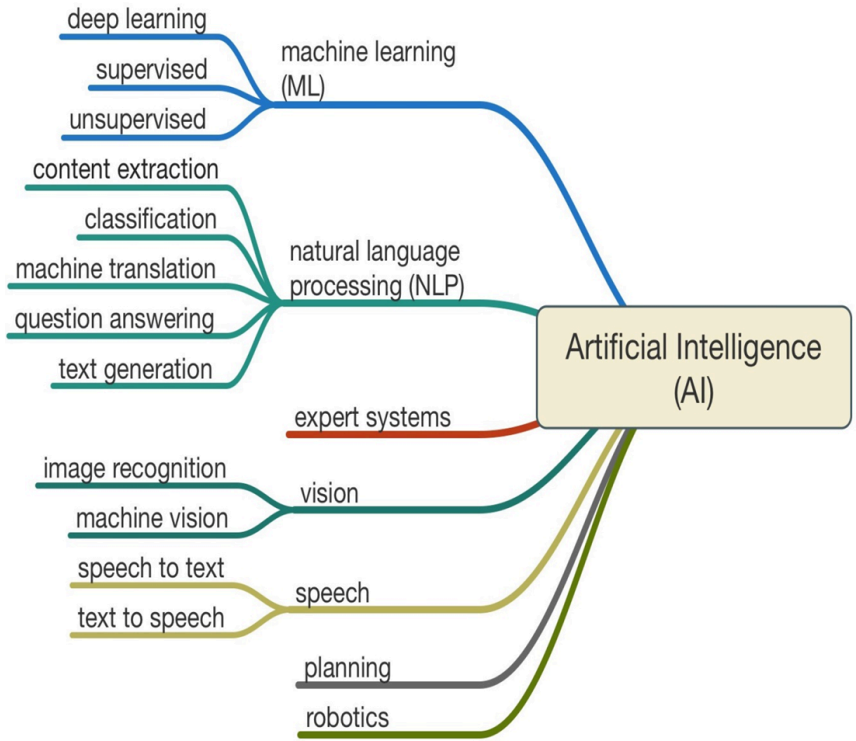
Roots of Artificial Intelligence

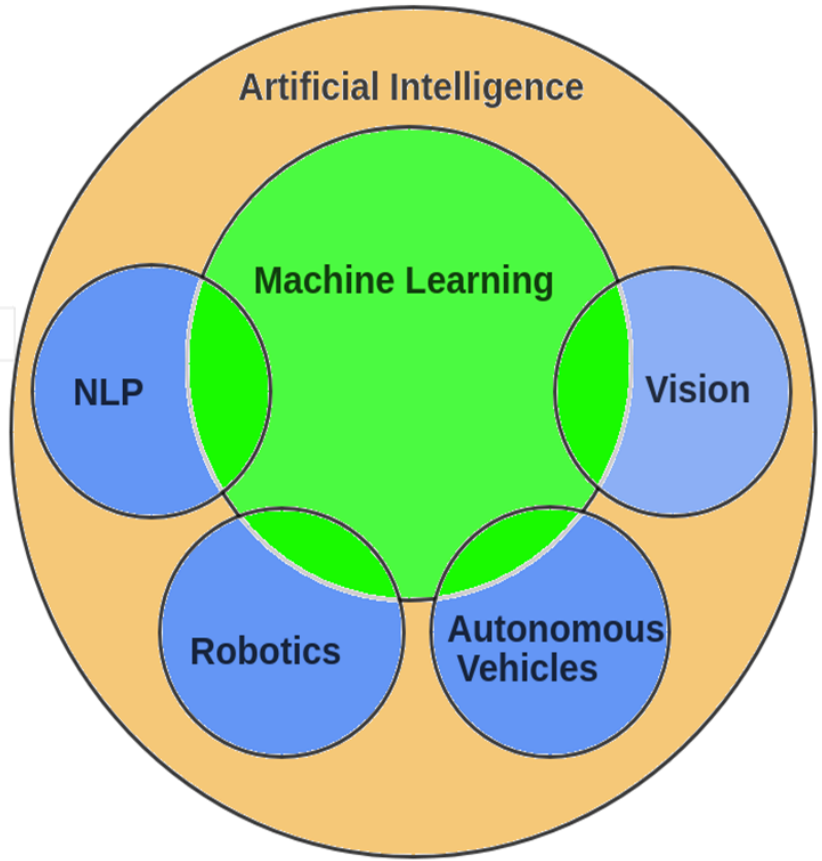


Timeline of AI History



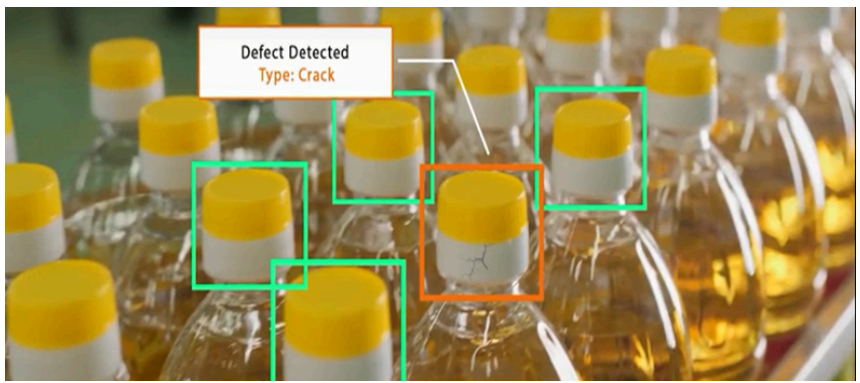
AI Subfields

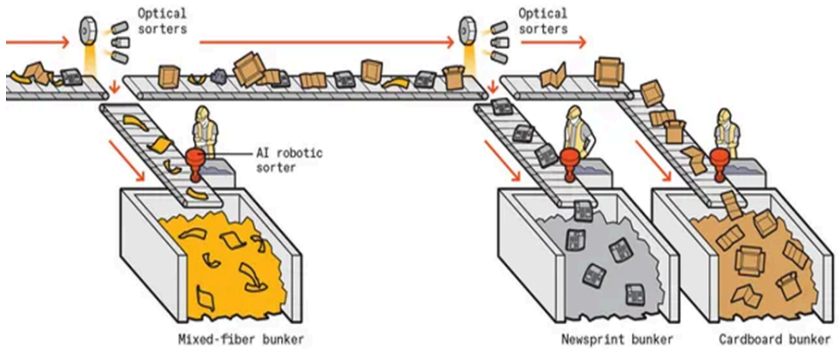




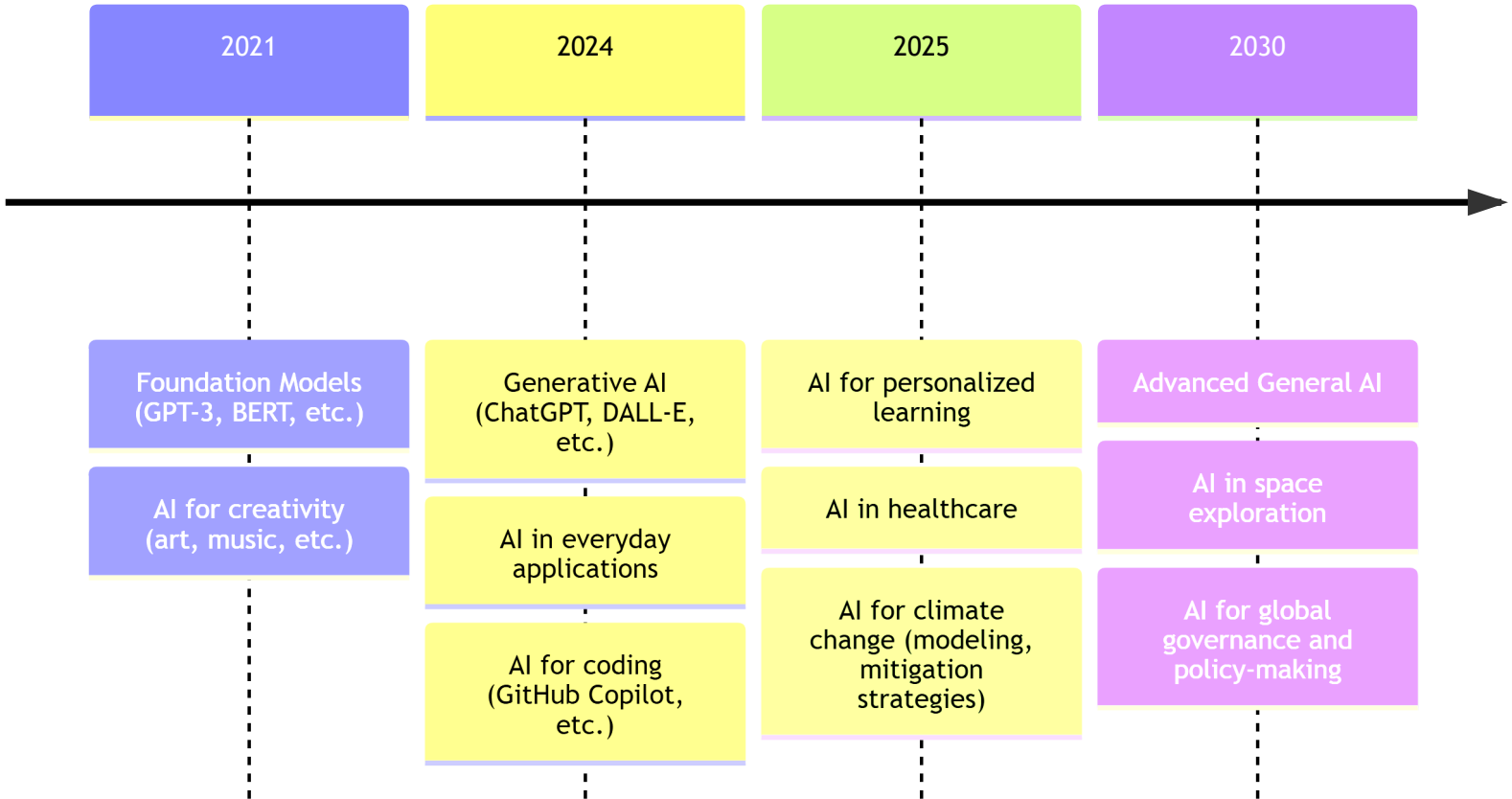
Applications of AI in Industry

- **Anomaly Detection:** in processes and equipment
- **Optimize processes:** Improve yield
- **Make smarter decisions** and minimize risk
- **Predict future scenarios** with neural networks



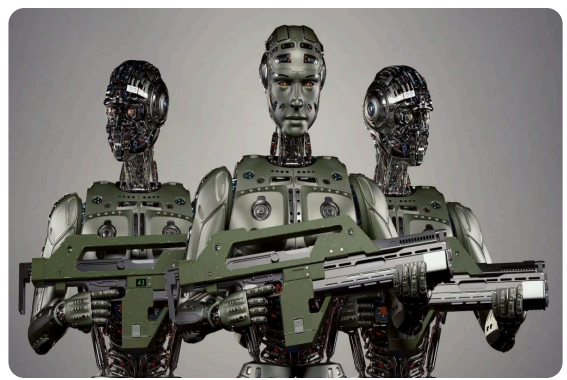


AI Current and Future Trends



Is artificial intelligence dangerous?

- AI can be dangerous if misused or poorly designed
- Risks include:
  - Job displacement
  - Privacy concerns
  - Bias and discrimination
  - Autonomous weapons
- Importance of ethical AI development and regulation



How to achieve AI?

Two Main Lines of Research:

1. Phenomenal Approach:

- **Knowledge Representation**: Encoding information about the world in a structured format for AI to process
- **Expert Systems and Planning**: AI use domain-specific knowledge to make decisions and plan actions
- **Searching**: AI systems explore possible solutions to find the most optimal path. Ex. Gaming
  - **Uninformed Search**: No additional information about states beyond the problem definition (e.g., BFS, DFS)
  - **Informed Search**: Uses problem-specific knowledge to find solutions more efficiently (e.g., A*, Greedy Best-First Search)

How to achieve AI?

Two Main Lines of Research:

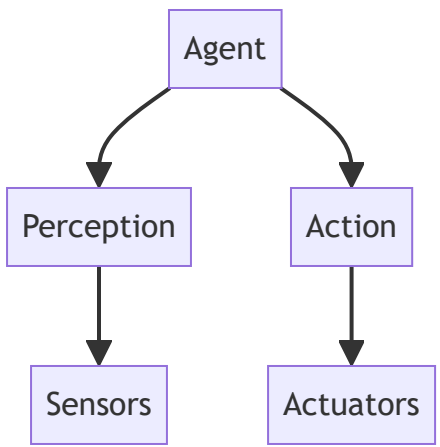
2. Biological Approach:

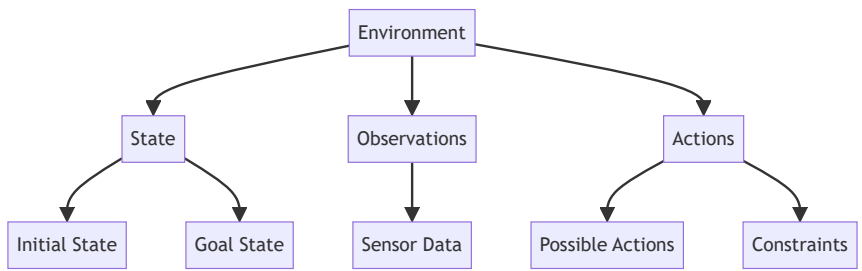
- **From Natural to Artificial Systems**: AI systems that mimic natural biological process
  - **Artificial Neural Networks**: Modeled after the human brain. – ANN
  - **Evolutionary Algorithms**: Inspired by natural selection (human evolution)
- **Learning**: AI systems learn by finding patterns in data and improve over time. Key Types:
  - **Supervised Learning**: Learning from labeled datasets.
  - **Unsupervised Learning**: Discovering hidden patterns in unlabeled data.
  - **Reinforcement Learning**: Trial-and-error learning to maximize rewards.
- **Agent**: AI systems interact with the environment by perception, communication, and action
  - **Natural Language Processing (NLP)**: Understanding and generating human language.
  - **Computer Vision**: Recognizing images, objects, and actions.
  - **Robotics**: Performing tasks based on sensor inputs

Chapter 2: Important Concepts in AI

The concept of Agents

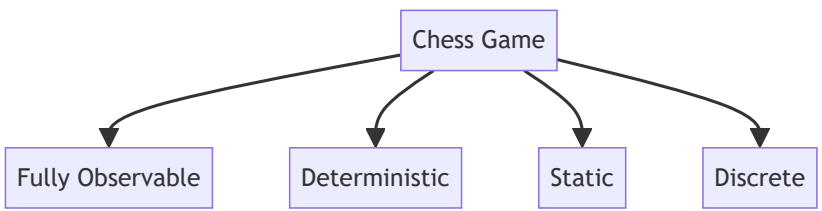
- An **agent** is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
- An agent can be a robot, a chatbot, or any other entity that can perceive and act. **The decisions of an agent are made in the context of its environment.**
- An **environment** is the surrounding or conditions in which an agent operates.

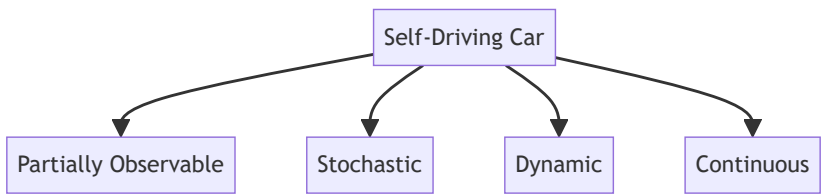




Types of environments

- **Fully observable**: The agent has access to all relevant information about the environment at all times.
- **Partially observable**: The agent has limited access to information.
- **Deterministic**: The outcome of an action is predictable and certain.
- **Stochastic**: The outcome of an action is uncertain and may involve randomness.
- **Static**: The environment does not change while the agent is deliberating.
- **Dynamic**: The environment can change while the agent is deliberating.
- **Discrete**: The number of possible states and actions is finite.
- **Continuous**: The number of possible states and actions is infinite.





Types of Agents

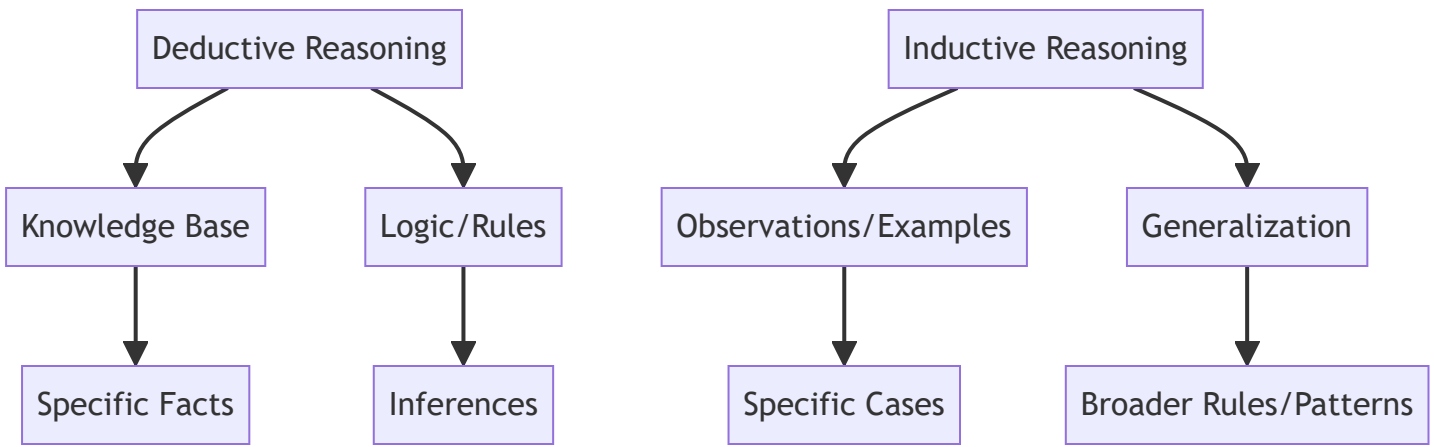
- **Simple Reflex Agents**: Act based on the current percept, ignoring the rest of the percept history.
- **Model-Based Reflex Agents**: Maintain an internal state to keep track of the world.
- **Goal-Based Agents**: Act to achieve specific goals, considering future actions.
- **Utility-Based Agents**: Act to maximize a utility function, which measures the desirability of different states.

Agent examples

Agent	Sensor/Input	Actuator/Output	Objective/Evaluation	State/Environment
Cleaning Robot	Camera, Joint sensor	Limbs, Joints	Cleanliness	Object positions
Chess Agent	Board input	Move output	Position score	Chess board
Self-driving Car	Camera, Sound sensor	Car controls	Safety, Speed, Goals	Traffic conditions
Chatbot	Keyboard	Screen	Chat quality	Dialog history

Inductive vs Deductive Reasoning

- **Deductive Reasoning**: Starts with a knowledge base of facts and use logic or other systematic methods in order to make inferences. There are several approaches to deductive reasoning, including search and logic-based methods.
- **Inductive Reasoning**: Starts with specific observations or examples and generalizes them to form broader rules or patterns. Inductive reasoning is often used in machine learning, where algorithms learn from data to make predictions or decisions.



Deductive reasoning in Artificial Intelligence

There are certain types of problems that repeatedly reappear in deductive forms of artificial intelligence. These are important representatives of “typical” problems, and their solutions can often be generalized to other similar problems. Therefore, studying these problems can provide insights into solving more general problems in the deductive setting.

1. Constraint Satisfaction Problems (CSPs):
- A set of variables, each with a domain of possible values.
- A set of constraints that specify allowable combinations of values for subsets of variables.

Example: Sudoku

- Variables: Each cell in the Sudoku grid.
- Domain: Numbers 1-9 for each cell.
- Constraints: No repeated numbers in any row, column, or 3x3 subgrid.

Sudoku as a CSP (Continued)

Given the following Sudoku puzzle, fill in the missing numbers to satisfy the constraints.

5	3	_	_	7	_	_	_	_
6	_	_	1	9	5	_	_	_
_	9	8	_	_	_	_	6	_
8	_	_	_	6	_	_	_	3
4	_	_	8	_	3	_	_	1
7	_	_	_	2	_	_	_	6
_	6	_	_	_	_	2	8	_
_	_	_	4	1	9	_	_	5
_	_	_	_	8	_	7	9	_

Sudoku as a CSP (Continued)

How to represent the Sudoku problem as a CSP:

- **Variables**: Each cell in the 9x9 grid (e.g., $(X_{1,1}, X_{1,2}, \ldots, X_{9,9})$).
- **Domains**: The possible values for each variable (1-9).
- **Constraints**:
  - Row constraints: All numbers in a row must be different.

$$\forall i \in \{1, \ldots, 9\}, \forall j, k \in \{1, \ldots, 9\}, j \neq k : X_{i,j} \neq X_{i,k}$$

  - Column constraints: All numbers in a column must be different.

$$\forall j \in \{1, \ldots, 9\}, \forall i, k \in \{1, \ldots, 9\}, i \neq k : X_{i,j} \neq X_{k,j}$$

  - Subgrid constraints: All numbers in each 3x3 subgrid must be different.

$$\forall m, n \in \{0, 1, 2\}, \forall i, j, k, l \in \{1, 2, 3\}, (i, j) \neq (k, l) : X_{3m+i,3n+j} \neq X_{3m+k,3n+l}$$

Sudoku as a CSP (Continued):

Row constraints:

All numbers in a row must be different.

$$\forall i \in \{1, \ldots, 9\}$$

$$\forall j, k \in \{1, \ldots, 9\}, j \neq k$$

$$X_{i,j} \neq X_{i,k}$$

i	j	k	Constraint
1	1	2	$X_{1,1} \neq X_{1,2}$
1	1	3	$X_{1,1} \neq X_{1,3}$
1	1	4	$X_{1,1} \neq X_{1,4}$
...	...	...	...
1	2	1	$X_{1,2} \neq X_{2,1}$
1	2	3	$X_{1,2} \neq X_{1,3}$
...	...	...	...
2	1	2	$X_{2,1} \neq X_{2,2}$
...	...	...	...

Sudoku as a CSP (Continued):

Column constraints:

All numbers in a column must be different.

$$\forall j \in \{1, \ldots, 9\}$$

$$\forall i, k \in \{1, \ldots, 9\}, i \neq k$$

$$X_{i,j} \neq X_{k,j}$$

i	j	k	Constraint
1	1	2	$X_{1,1} \neq X_{2,1}$
1	1	3	$X_{1,1} \neq X_{3,1}$
1	1	4	$X_{1,1} \neq X_{4,1}$
...	...	...	...
2	1	1	$X_{2,1} \neq X_{1,1}$
2	1	3	$X_{2,1} \neq X_{3,1}$
...	...	...	...
1	2	2	$X_{1,2} \neq X_{2,2}$
...	...	...	...

Sudoku as a CSP (Continued):

Subgrid constraints:All numbers in each 3x3 subgrid must be different.

$$\forall m, n \in \{0, 1, 2\}, \forall i, j, k, l \in \{1, 2, 3\}, (i, j) \neq (k, l)$$

$$X_{3m+i,3n+j} \neq X_{3m+k,3n+l}$$

$m, n \in \{0, 1, 2\}$	$(1, 1)$	$(3m + 1,\ 3n + 1)$
$m, n \in \{0, 1, 2\}$	$(1, 2)$	$(3m + 1,\ 3n + 2)$
$m, n \in \{0, 1, 2\}$	$(1, 3)$	$(3m + 1,\ 3n + 3)$
$m, n \in \{0, 1, 2\}$	$(2, 1)$	$(3m + 2,\ 3n + 1)$
$m, n \in \{0, 1, 2\}$	$(2, 2)$	$(3m + 2,\ 3n + 2)$
$m, n \in \{0, 1, 2\}$	$(2, 3)$	$(3m + 2,\ 3n + 3)$
$m, n \in \{0, 1, 2\}$	$(3, 1)$	$(3m + 3,\ 3n + 1)$
$m, n \in \{0, 1, 2\}$	$(3, 2)$	$(3m + 3,\ 3n + 2)$
$m, n \in \{0, 1, 2\}$	$(3, 3)$	$(3m + 3,\ 3n + 3)$

$X_{1,1} \neq X_{1,2},\ X_{1,1} \neq X_{1,3},\ X_{1,1} \neq X_{2,1},\ X_{1,1} \neq X_{2,2},\ X_{1,1} \neq X_{2,3},\ X_{1,1} \neq X_{3,1},\ X_{1,1} \neq X_{3,2},\ X_{1,1} \neq X_{3,3},$
$X_{1,2} \neq X_{1,3},\ X_{1,2} \neq X_{2,1},\ X_{1,2} \neq X_{2,2},\ X_{1,2} \neq X_{2,3},\ X_{1,2} \neq X_{3,1},\ X_{1,2} \neq X_{3,2},\ X_{1,2} \neq X_{3,3},$
$X_{1,3} \neq X_{2,1},\ X_{1,3} \neq X_{2,2},\ X_{1,3} \neq X_{2,3},\ X_{1,3} \neq X_{3,1},\ X_{1,3} \neq X_{3,2},\ X_{1,3} \neq X_{3,3},$
$X_{2,1} \neq X_{2,2},\ X_{2,1} \neq X_{2,3},\ X_{2,1} \neq X_{3,1},\ X_{2,1} \neq X_{3,2},\ X_{2,1} \neq X_{3,3},$
$X_{2,2} \neq X_{2,3},\ X_{2,2} \neq X_{3,1},\ X_{2,2} \neq X_{3,2},\ X_{2,2} \neq X_{3,3},$
$X_{2,3} \neq X_{3,1},\ X_{2,3} \neq X_{3,2},\ X_{2,3} \neq X_{3,3},$
$X_{3,1} \neq X_{3,2},\ X_{3,1} \neq X_{3,3},$
$X_{3,2} \neq X_{3,3}.$

$$m = 0, n = 0$$

Solving Sudoku as CSP using python-constraint library:

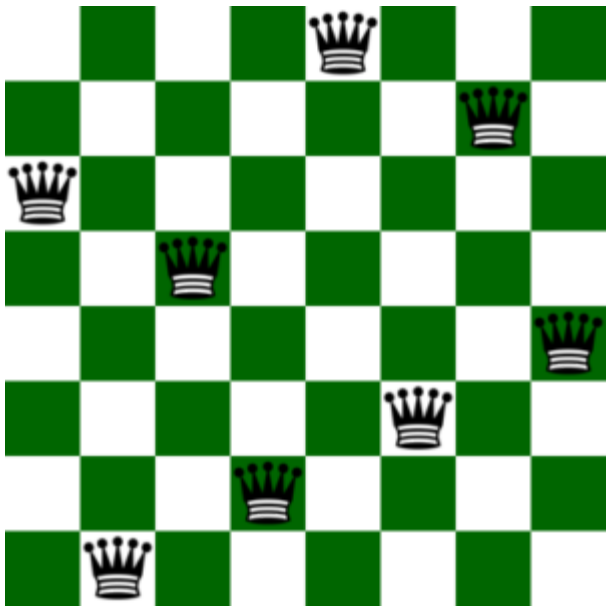
```
        return None
    #Example Sudoku puzzle (0 represents empty cells)
puzzle = [
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9]
]

#Solve the puzzle
solution = solve_sudoku(puzzle)
if solution:
    print("Sudoku solved successfully:")
    for row in range(9):
        print([solution[(row, col)] for col in range(9)])
else:
    print("No solution exists.")
```

Homework

- There are several CSP representations of the Sudoku problem. One can, for example, represent the problem using binary constraints instead of n-ary constraints. In this case, one would have to define a binary constraint between every pair of variables that share a row, column, or subgrid. This would lead to a larger number of constraints but would allow for more efficient constraint propagation techniques to be used.
- **Variables**: Each cell in the 9x9x9 grid (e.g., $(X_{1,1,1}, X_{1,1,2}, \ldots, X_{9,9,9})$).
- **Domains**: The possible values for each variable (0-1).
  for example $X_{1,1,5} = 1$ means that the cell in row 1 and column 1 contains the number 5.
- **Constraints**: ??

The 8-Queens Problem as a CSP:
- Place 8 queens on a chessboard such that no two queens threaten each other.



Modeling the 8-Queens problem as a CSP:

- Variables: $Q_1, Q_2, \ldots, Q_8$ (each representing the column position of a queen in each row).
- Domains: $\{1, 2, \ldots, 8\}$ for each variable.
- Constraints:
  - Row constraints: Each queen must be in a different row (inherent in the variable definition).
  - Column constraints: No two queens can be in the same column.

$$\forall i, j \in \{1, \ldots, 8\}, i \neq j : Q_i \neq Q_j$$

  - Diagonal constraints: No two queens can be on the same diagonal.

$$|Q_i - Q_j| \neq |i - j|, \forall i, j \in \{1, \ldots, 8\}, i \neq j$$

The 8-Queens Problem as a CSP - Continued:

- **Variables**: $Q_1, Q_2, \ldots, Q_8$ (each representing the column position of a queen in each row).
- **Domains**: $\{1, 2, \ldots, 8\}$ for each variable. Diagonal constraints: No two queens can be on the same diagonal.

$$|Q_i - Q_j| \neq |i - j|, \forall i, j \in \{1, \ldots, 8\}, i \neq j$$

- The absolute difference between the column positions of two queens must not equal the absolute difference between their row positions. This ensures that they are not on the same diagonal. for example, if queen Q_1 is in row 1 and column 3 ($Q_1 = 3$) and queen Q_2 is in row 2 and column 5 ($Q_2 = 5$), then:

$$|Q_1 - Q_2| = |3 - 5| = 2$$

$$|1 - 2| = 1$$

Since 2 is not equal to 1, the queens are not on the same diagonal.

Solving the 8-Queens problem as CSP using python-constraint library:

```
from constraint import Problem, AllDifferentConstraint

def solve_8_queens():
    problem = Problem()
    #Define variables and their domains
    problem.addVariables(range(1, 9), range(1, 9))
    #Column constraints
    problem.addConstraint(AllDifferentConstraint(), range(1, 9))
    #Diagonal constraints
    for i in range(1, 9):
        for j in range(i + 1, 9):
            problem.addConstraint(lambda i=i, j=j: abs(i - j) != abs(problem.getSolution()[i] - problem.getSolution()[j]
    #Get solutions
    solutions = problem.getSolutions()
    return solutions
#Solve the problem
solutions = solve_8_queens()
print(f"Number of solutions found: {len(solutions)}")
for solution in solutions:
    print(solution)
```

The travelling Salesman Problem (TSP) as a CSP:

- Given a set of cities denoted by $1, \ldots, n$ and the distances between each pair of cities denoted by $d(i, j)$, the goal is to find tour of the cities of cost at most C.
- There are $(n(n-1))$ variables $X_{i,j}$, where $X_{i,j} = 1$ if the tour goes directly from city i to city j, and $X_{i,j} = 0$ otherwise.
- The constraints are:

Each city must be entered and exited exactly once:

$$\forall i \in \{1, \ldots, n\} : \sum_{j=1, j \neq i}^{n} X_{i,j} = 1$$

$$\forall j \in \{1, \ldots, n\} : \sum_{i=1, i \neq j}^{n} X_{i,j} = 1$$

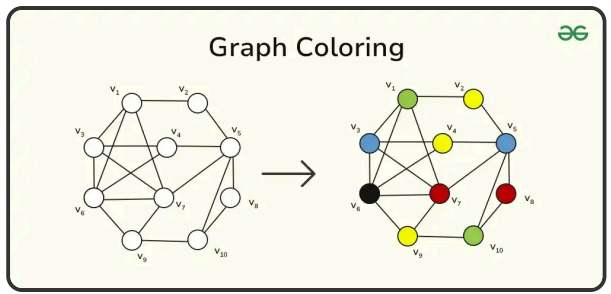
The total cost of the tour must not exceed C:

$$\sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} d(i, j) * X_{i,j} \leq C$$

Homework

Using $constraint$ python library, implement a solution for the TSP problem as a CSP.

The Graph Coloring Problem as a CSP: Given an undirected graph $G = (V, E)$ and a set of colors $1, \ldots, k$, the goal is to assign a color to each vertex such that no two adjacent vertices share the same color.



- This problem is closely related to the map coloring problem,wherein we wish to color the regions (e.g., provinces or states) in a map, so that each pair of adjacent regions has different colors.
- The map coloring problem has practical applications in cartography, because it helps visually distinguish the different regions in the map.

The Graph Coloring Problem as a CSP - Continued: Modeling the graph coloring problem as a CSP:

- **Variables**: $n \times k$ binary variables $X_{i,c}$, where $X_{i,c} = 1$ if vertex i is assigned color c, and $X_{i,c} = 0$ otherwise.
- **Domains**: $\{0, 1\}$ for each variable.
- **Constraints**:
  - Color each node exactly once:

$$\sum_{c=1}^{k} X_{i,c} = 1, \forall i \in \{1, \ldots, n\}$$

  - Adjacent nodes must have different colors:

$$X_{i,c} + X_{j,c} \leq 1, \forall (i,j) \in E, \forall c \in \{1, \ldots, k\}$$

2. Solving NP-Hard Problems:

- Many problems in artificial intelligence suffer from the combinatorial explosion in the number of possible solutions as the problem size increases. For example, in the case of the eight queens problem, one has to choose eight positions out of 64 positions. The number of possible solutions is given by

$$C(64, 8) = \frac{64!}{8!(64-8)!} = 4,426,165,368$$

- Generalizing the eight queens problem to the n-queens problem leads to an NP-hard setting.
- The Traveling Salesman Problem (TSP) is another classic NP-hard problem where the goal is to find the shortest possible route that visits a set of cities and returns to the origin city. An instance of TSP with n cities has $(n-1)!$ possible routes, which grows factorially with n.
- Many other problems in AI, such as scheduling, vehicle routing, and resource allocation, are also NP-hard.

3. Expert systems:

- Expert systems are AI programs that mimic the decision-making abilities of a human expert. They use a knowledge base of human expertise and an inference engine to solve specific problems within a certain domain.
- Components of expert systems:
  - **Knowledge Base**: Contains domain-specific knowledge, including facts and rules.
  - **Inference Engine**: Applies logical rules to the knowledge base to deduce new information or make decisions.
  - **User Interface**: Allows users to interact with the expert system, input data, and receive advice or solutions.
- Applications of expert systems:
  - Medical diagnosis (e.g., MYCIN)
  - Financial forecasting
  - Troubleshooting and repair (e.g., XCON)

3. Expert systems (continued):
For example, consider a patient, John, who comes to a doctor, while presenting the following facts about their situation:

- John is running a temperature
- John is coughing
- John has colored phlegm

Now imagine a case where the expert system contains the following subset of rules:

- IF coughing AND temperature THEN infection
- IF colored phlegm AND infection THEN bacterial infection
- IF bacterial infection THEN administer antibiotic

A doctor can then enter John's symptoms in the expert system and then use a chain of inferences in order to conclude that an antibiotic needs to be administered to John.

In practice, one would have to enter a very large number of rules and cases in order to make the system work well.

Classical Methods for Deductive Reasoning

There are several classical methods for deductive reasoning in artificial intelligence, including:

- **Search Algorithms**: Techniques like depth-first search, breadth-first search, and A* search are used to explore possible states and find solutions to problems. Indeed, many problems in AI can be represented as graphs, where nodes represent states and edges represent actions or transitions between states. Search algorithms are used to traverse these graphs to find a path from an initial state to a goal state.
- **Logic-Based Methods**: These involve using formal logic to represent knowledge and make inferences. Examples include propositional logic, first-order logic, and resolution.
- **Constraint Satisfaction Problems (CSPs)**: As discussed earlier, CSPs involve finding values for variables that satisfy a set of constraints. Techniques like backtracking, constraint propagation, and local search are commonly used to solve CSPs.

Strengths and Limitations of Deductive Reasoning

⚠️ The greatest strengths of deductive reasoning are also its greatest limitations

✅ Deductive reasoning requires a way to code up expert knowledge in a knowledge base. Coding up such knowledge requires a human to understand and interpret this knowledge. This results in highly interpretable systems, which are obviously desirable. However, this interpretability is also an Achilles heel in the goal towards human-like behavior, because many human decisions rely on a high level of understanding that is not easily interpretable.

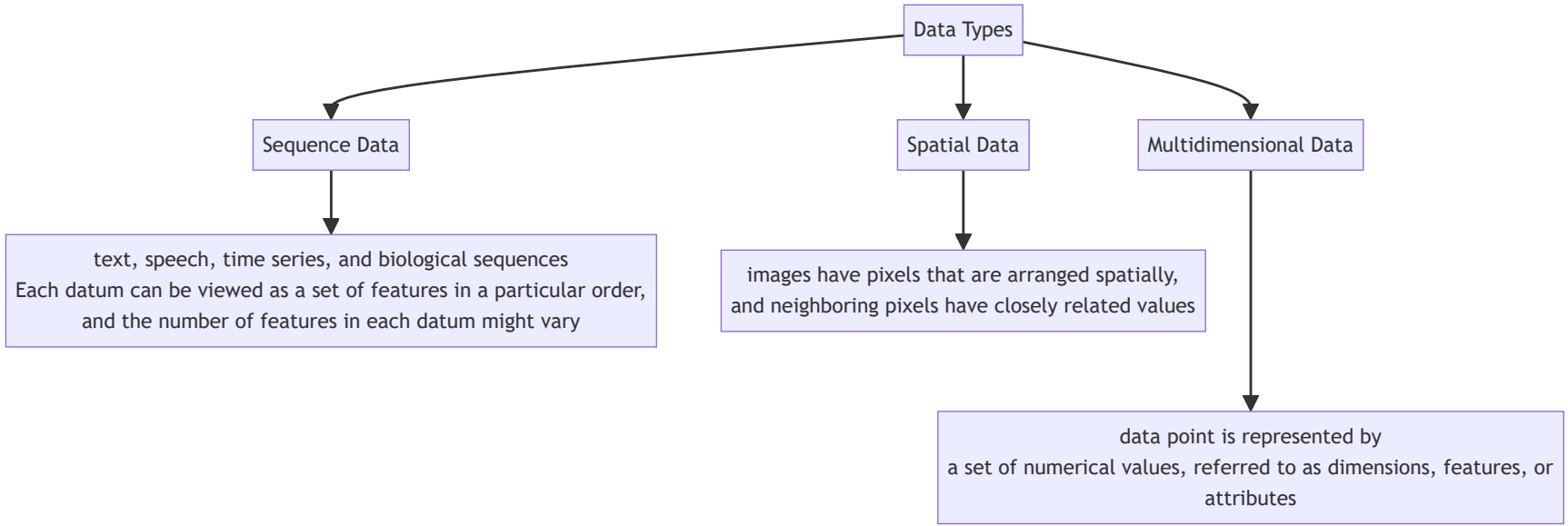
Inductive Learning in Artificial Intelligence

- While deductive reasoning systems try to encode domain knowledge within a knowledge base to make hypotheses. Inductive learning systems try to use data in order to create their own data-dependent hypotheses.

> ✅ **In inductive learning a mathematical model is used to define a hypothesis, the resulting model is used for prediction on examples that have not been seen before**

- The general idea of using examples in order to learn models for prediction is also referred to as **machine learning**.

Types of Data in Inductive Learning



✅ In an image, two adjacent pixels are highly likely to have the same value, and most of the information about the image is often embedded in a small number of pixels that lie in the regions of high variability. Machine learning models need to account for these relationships.

Types of Learning in Inductive Learning

Supervised Learning

The model is trained on a labeled dataset, where each input has a corresponding output label. The goal is to learn a mapping from inputs to outputs. Examples include classification and regression tasks.

Unsupervised Learning

The model is trained on an unlabeled dataset, and the goal is to discover patterns or structures within the data. Examples include clustering and dimensionality reduction tasks.

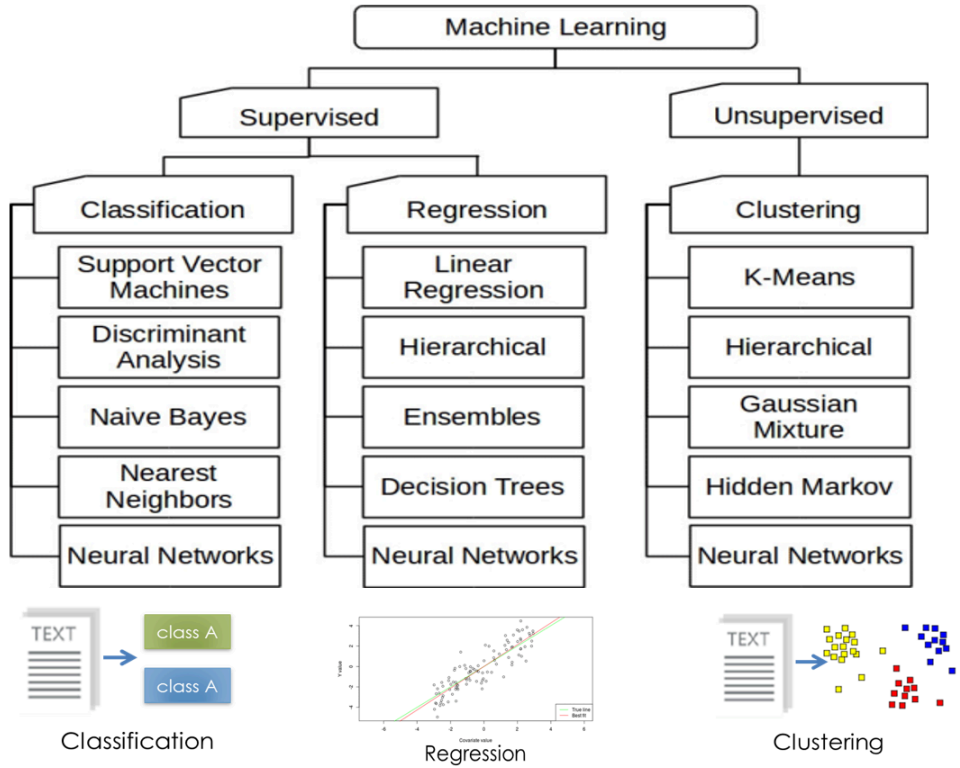
Semi-Supervised Learning

The model is trained on a combination of labeled and unlabeled data. This approach is useful when obtaining a fully labeled dataset is expensive or time-consuming.

Reinforcement Learning

The model learns by interacting with an environment and receiving feedback in the form of rewards or penalties. The goal is to learn a policy that maximizes cumulative rewards over time.

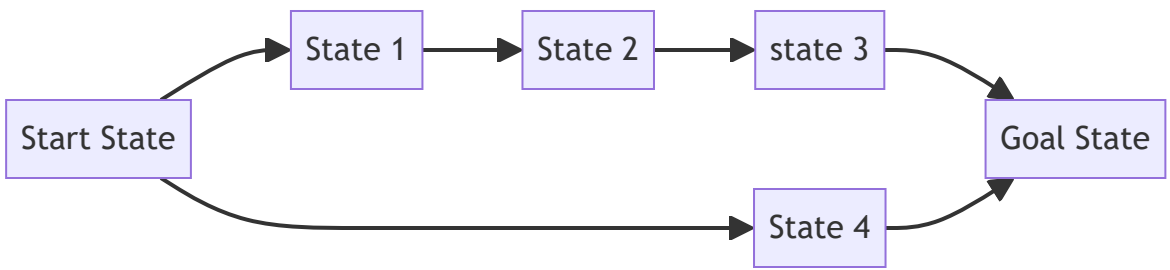
Common Machine Learning Algorithms



Chapter 3: Searching State spaces

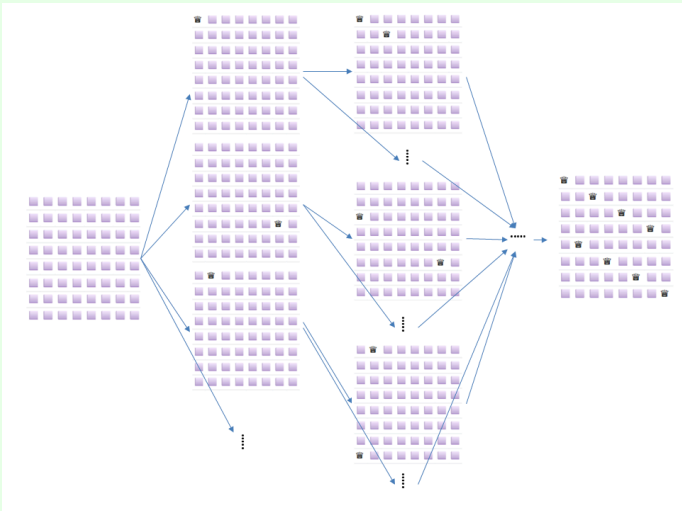
Introduction to State Space as a Graph

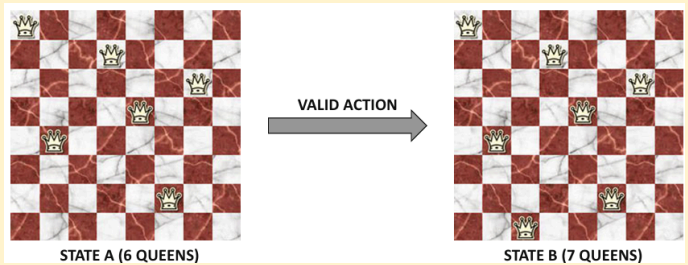
- The agent needs to search through the space in order to reach a particular goal or to maximize its reward. In some cases, the path chosen by the agent through the search space has an impact on the earned reward.
- In most real-world settings, **the state space is very large**, which makes the search process very challenging.
- The modeling of the state space as a graph enables the development of algorithms that leverage the graph structure of the space for search.
- A state space can be represented as a graph, where:
  - **Nodes** = possible states
  - **Edges** = transitions/actions



How to represent the 8-Queen problem as a state space graph?

- The 8-Queens problem is a classic combinatorial problem where the goal is to place 8 queens on an 8x8 chessboard such that no two queens threaten each other. This means that no two queens can be in the same row, column, or diagonal.
- The state space for search can be treated as a directed graph, in which each state can be treated as a node of the graph.





Dead End State

Search Problem 8-Queens

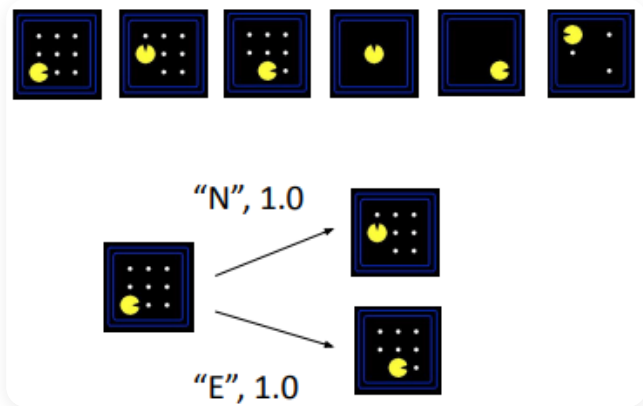
The State Space
Each state represents a configuration of the chessboard with a certain number of queens placed on it.

Successor Function
Generates all possible valid configurations that can be reached by placing an additional queen on the board.
For example, if there are already 3 queens placed, the successor function will generate all configurations with 4 queens by placing the new queen in any valid position on the next row.

Start State: An empty chessboard with no queens placed.
Goal Test: A configuration where all 8 queens are placed on the board without threatening each other.

Search problem pacman

The State Space
Each state represents Pacman's current position in the maze,and the remaining food pellets.



Successor Function
Moving Pacman in one of the four cardinal directions (N, E, S, W).

Start State: Pacman starts at a specific position in the maze with all food pellets present.
Goal Test: The goal is to reach a state where all food pellets have been collected.

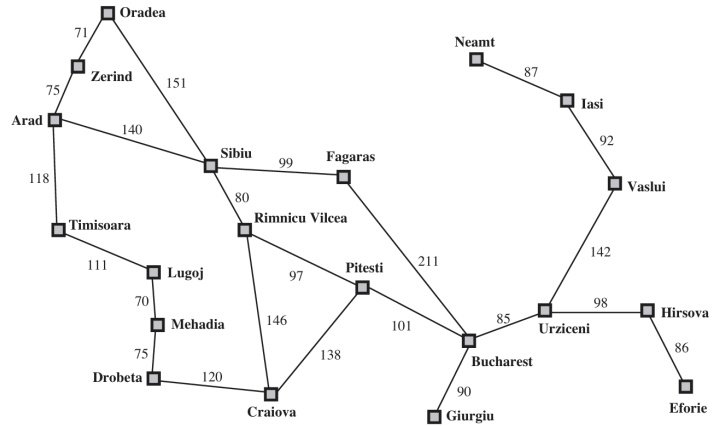
Search problem Travelling in Romania

The State Space

Each state represents a specific location in Romania and the cities that have been visited.

Successor Function

Moving from one city to another connected by a road with cost = distance.



Start State: The initial city where the traveler starts.
Goal Test: Reaching the destination city.

Uninformed Search

Uninformed search algorithms, also known as blind search algorithms, are a category of search strategies that operate without any domain-specific knowledge about the problem being solved. They explore the search space systematically, relying solely on the structure of the state space and the goal test to find a solution.

Generic Search Algorithm

The generic search algorithm forms the basis for many search strategies (BFS, DFS, etc.). It maintains a frontier of states to explore and a set of explored states to avoid revisiting.

```
Algorithm GenericSearch(Initial State: s, Goal Condition: G)
 begin
  LIST= { s };
  repeat
    Select current node i from LIST based on pre-defined strategy;
    Delete node i from LIST;
    Add node i to the hash table VISIT;
    for all nodes j ∈ A(i) directly connected to i via transition do
    begin
      if (j is not in VISIT) add j to LIST;
      pred(j)=i;
    end
  until LIST is empty or current node i satisfies G;
  if current node satisfies G return success
    else return failure;
  { The predecessor array can be used to trace back path from i to s }
 end
```

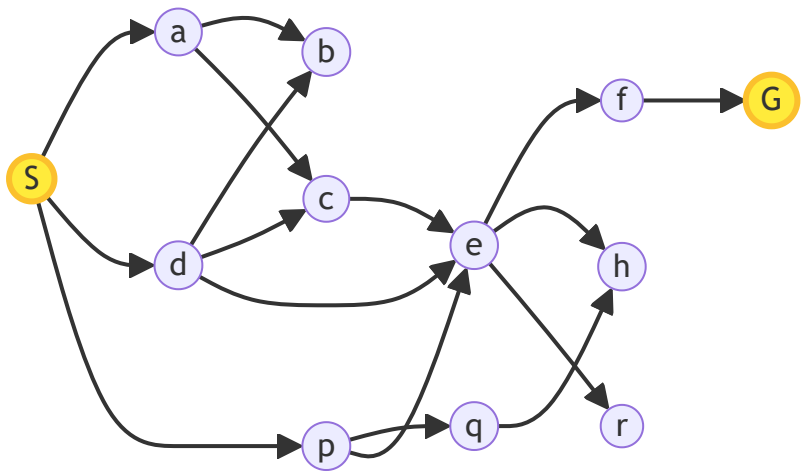
List: Used to store the frontier of states to be explored.

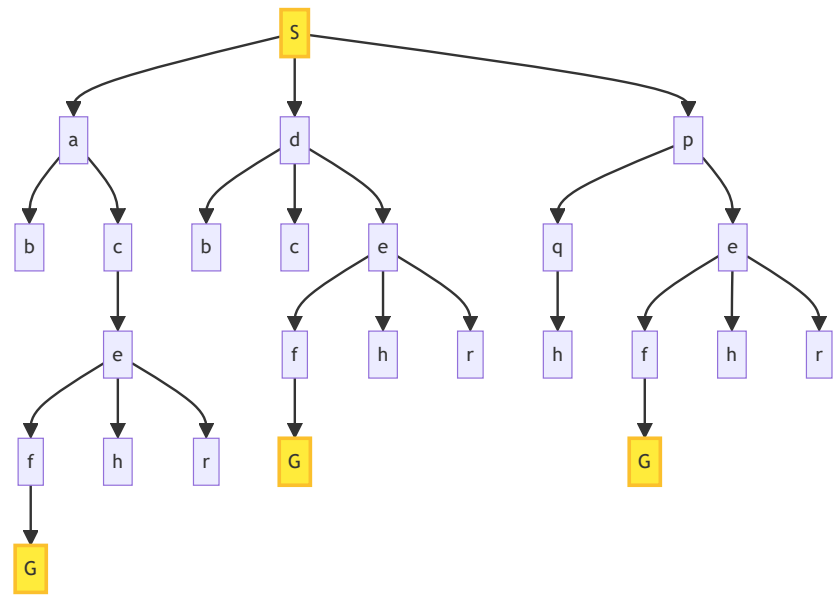
VISIT: A hash table to keep track of explored states to avoid cycles.

pred(j): An array to store the predecessor of each state for path reconstruction.

Seach Space VS Search Tree

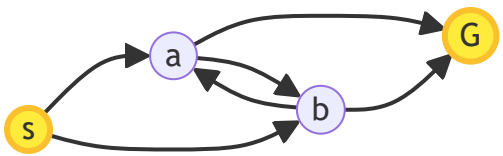
The state space graph represents all possible states and transitions in the problem domain, while the search tree represents the specific paths taken during the search process. The search tree is a subset of the state space graph, focusing on the nodes and edges explored by the search algorithm.

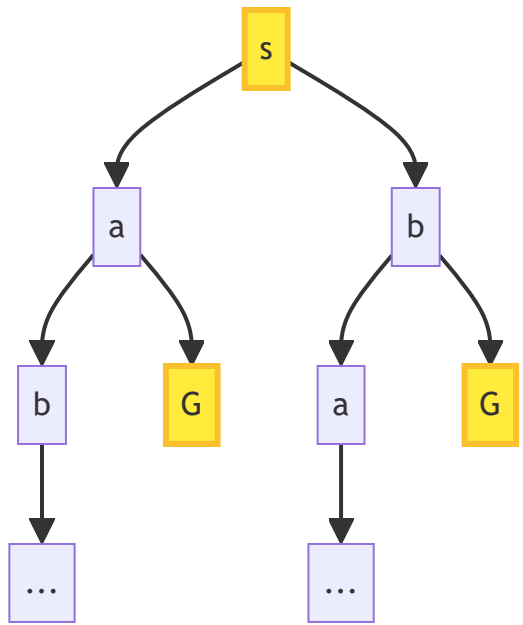




Seach Space VS Search Tree

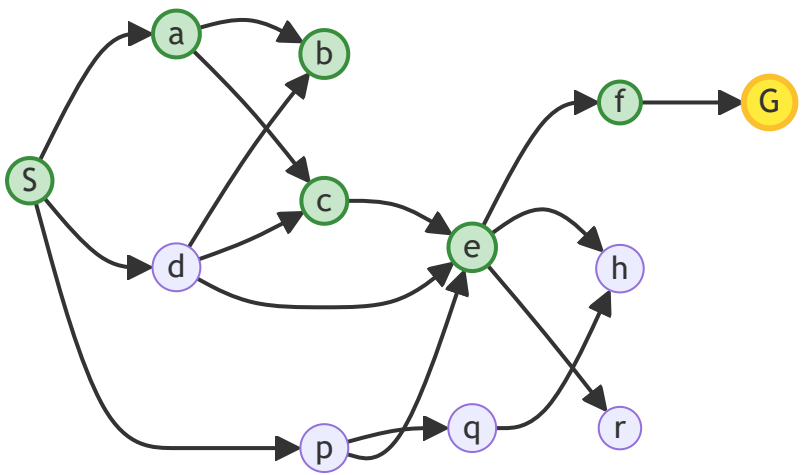
Note: The size of the search tree can be infinitely larger than the size of the state space graph due to the presence of cycles and multiple paths leading to the same state.

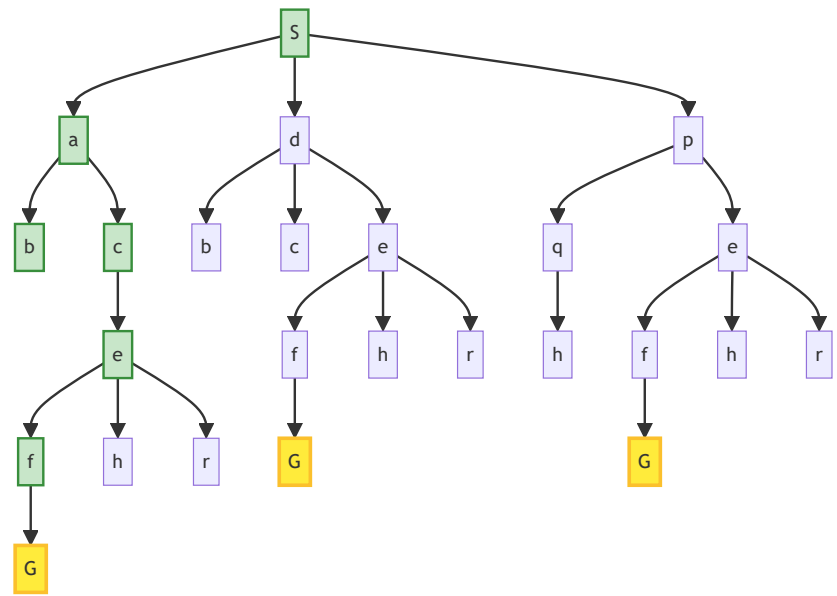




Depth-First Search (DFS)

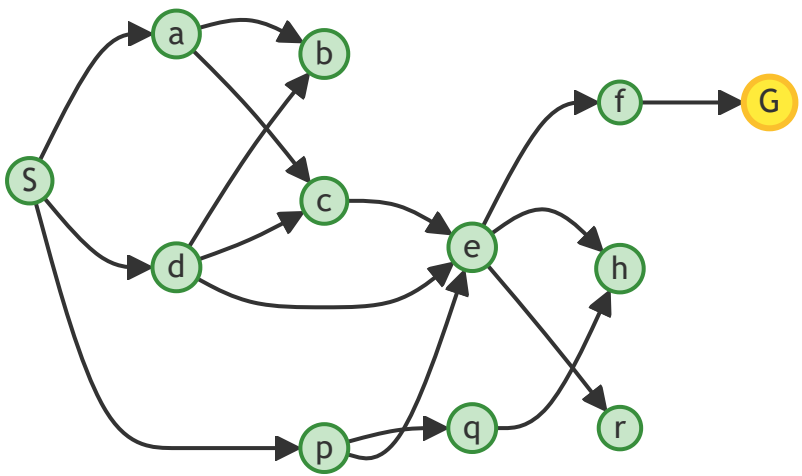
The insertion and deletion operations in the LIST are performed at the end of the list, making it a Last In First Out (LIFO) structure. This means that the most recently added node is expanded first, leading to a deep exploration of the search space before backtracking.

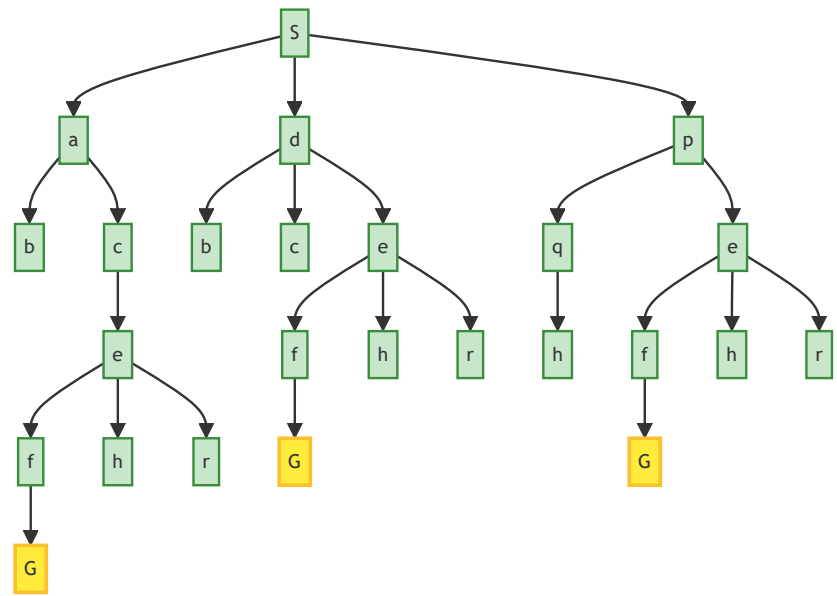




Breadth-First Search (BFS)

The insertion operation is performed at the end of the list, while the deletion operation is performed at the front of the list, making it a First In First Out (FIFO) structure. This means that nodes are expanded in the order they were added, leading to a level-by-level exploration of the search space.





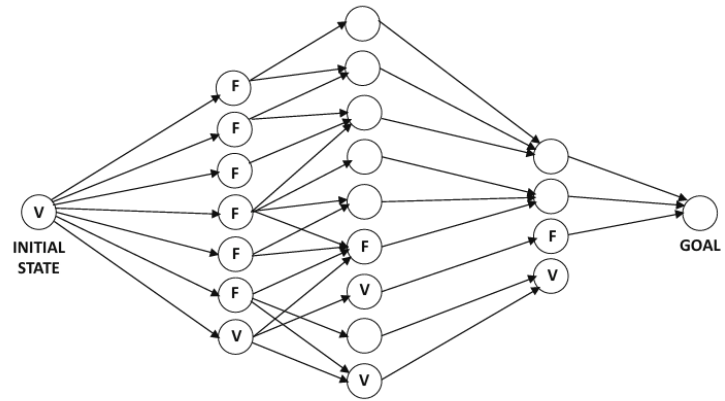
Depth-First Search vs Breadth-First Search

Depth-First Search (DFS)

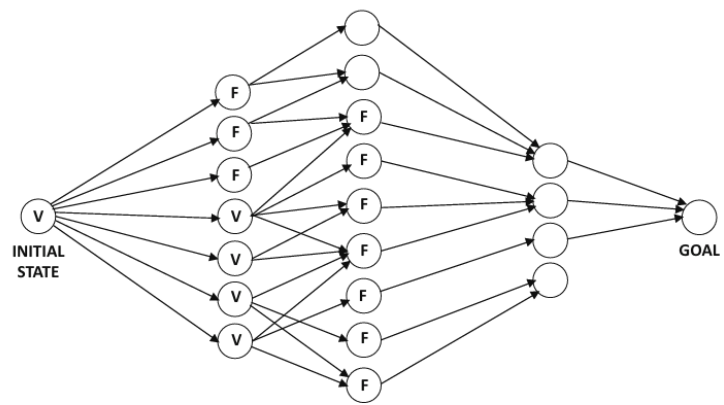
- Explores deep into branches before backtracking.
- Uses less memory (stores current path).
- May get stuck in infinite branches.
- Not guaranteed to find shortest path.

Breadth-First Search (BFS)

- Explores nodes level by level.
- Uses more memory (stores all nodes at current level).
- Guaranteed to find shortest path.
- Can be slower for deep trees.



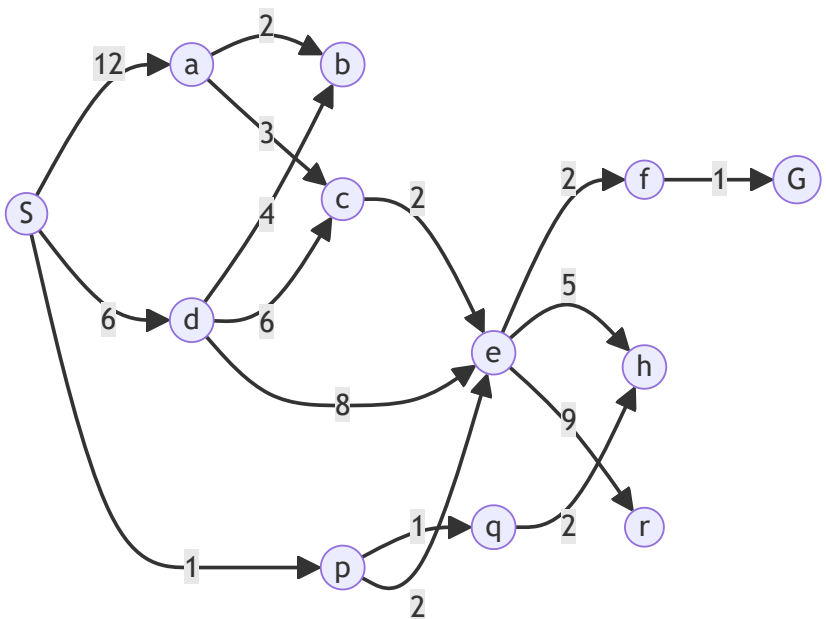
(a) Depth-first search

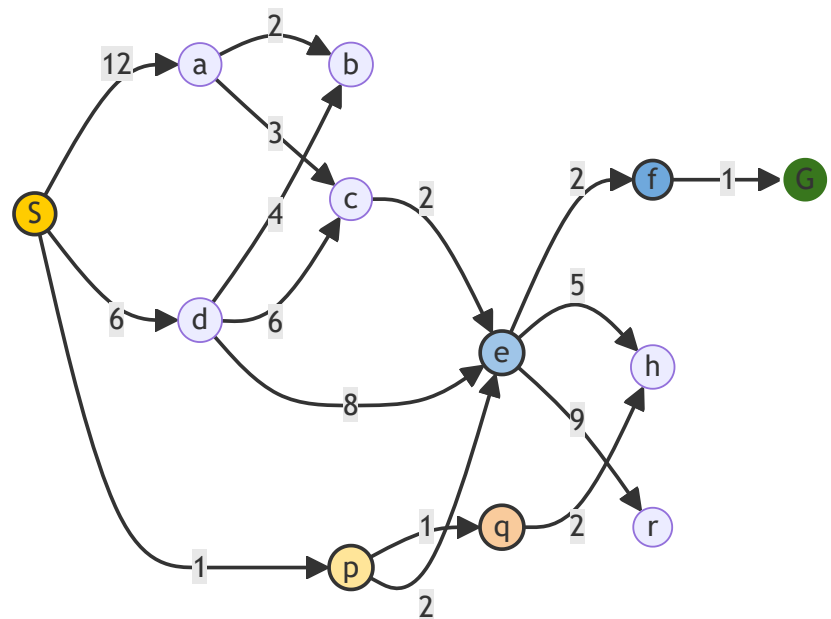


(b) Breadth-first search

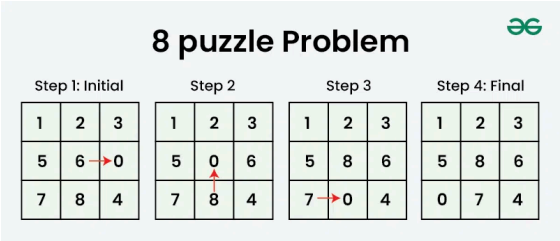
Uniform Cost Search (UCS)

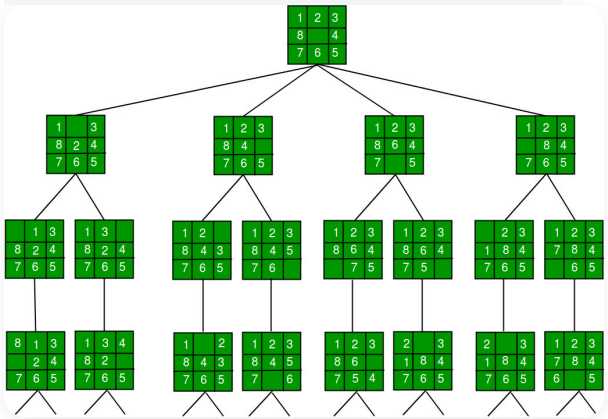
Uniform Cost Search (UCS) is an uninformed search algorithm that expands the least costly node first, ensuring that the path to the goal is optimal in terms of cost. It uses a priority queue to manage the frontier, where nodes are prioritized based on their cumulative cost from the start state.





Case Study: Eight-Puzzle Problem

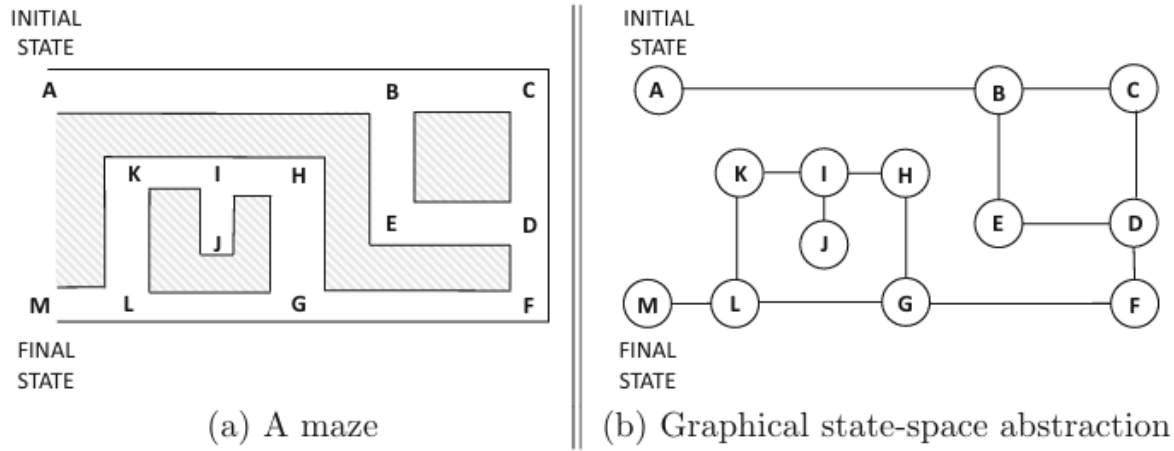




The Eight-Puzzle is a sliding puzzle that consists of a 3x3 grid with eight numbered tiles and one empty space. The objective is to arrange the tiles in a specific order by sliding them into the empty space.

State Space: Each state represents a configuration of the tiles on the board.
Successor Function: Generates all possible configurations by sliding a tile into the empty space.
Start State: An initial configuration of the tiles.
Goal Test: A configuration where the tiles are arranged in the desired order.

Case Study: Online Maze Search



(a) A maze (b) Graphical state-space abstraction

State Space: Each state represents the agent's current position in the maze.
Successor Function: Generates possible moves (up, down, left, right) based on the agent's current position and the maze layout.
Start State: The initial position of the agent in the maze.
Goal Test: The agent reaches the goal location.

Informed Search

Informed search algorithms, also known as heuristic search algorithms, are a category of search strategies that utilize domain-specific knowledge to guide the search process. They employ heuristics, which are problem-specific rules or strategies, to estimate the cost of reaching the goal from a given state. This allows informed search algorithms to explore the search space more efficiently and effectively than uninformed search algorithms.