Index Page

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Lab No. | Title/Question | Submission Date | Signature | Remarks |
| 2.1 | Frame Implementation |  |  |  |
| 2.2 | Prolog Basics: |  |  |
| 2.3 | Ancestor |  |  |
| 2.4 | Family Tree |  |  |
| 2.5 | Semantic Net |  |  |
| 2.6 | Naïve Bayes Classifier |  |  |
| 2.7 | Disease Diagnosis |  |  |
| 2.8 | Perceptron Logic Gates |  |  |
| 2.9 | Back Propagation |  |  |
| 2.10 | N-Queen Problem |  |  |
| 2.11 | Water Jug Problem |  |  |
| 2.12 | Genetic Algorithm |  |  |
| 2.13 | Naïve Bayes (Repeat) |  |  |
| 2.14 | Natural Language Processing |  |  |

**Q2.1. Implementing Frame.** (Recommended language: C++)

"Ram is a person living in Nepal. He was born on 15th December of year 1990. He is 6

inch tall and has 75 kg weight. He has a job. He works at 'ABC company' as AI

Researcher and earns 1.5 lakhs per month. The company is situated at Kathmandu."

**THEORY**:

Frame-based knowledge representation is a method in Artificial Intelligence that organizes knowledge into structured units called frames. A frame is similar to a data structure or an object in programming, where each frame represents an entity, concept, or situation. It contains slots (attributes or properties) and associated values that describe the characteristics of that entity. Frames also support inheritance, meaning specific frames can derive properties from more general ones, creating a hierarchical representation of knowledge.

**Example:**  
For instance, a frame called *Car* might include slots such as *Color*, *Engine-Type*, and *Owner*. A more specific frame like *Sports Car* can inherit these slots from *Car* while adding new ones such as *Top-Speed*. If the *Car* frame specifies that the default *Color* is black, then unless otherwise stated, a *Sports Car* will also be assumed to be black.

The following code implements the idea of Frames. Each class or struct acts like a **frame**, and its **attributes (slots)** store values:

* **CCompany** → Frame with slots *Name*, *Location*.
* **CJob** → Frame with slots *Employer (Company frame)*, *Position*, *Salary*.
* **SDate** → Frame with slots *Year*, *Month*, *Day*.
* **CPerson** → Frame with slots *Name*, *Country*, *DoB (Date frame)*, *Height*, *Weight*, *Job (Job frame)*.

**Code:**#include <iostream>

#include <string>

#include <string\_view>

#include "src/commons.h" // footer():

enum class EMonth {

  January = 1, February,  March,  April,  May,  June,  July,

August,  September,  October,  November,  December

};

struct SDate {

  int Year;

  EMonth Month;

  int Day;

  friend std::ostream &operator<<(std::ostream &out, const SDate &date) {

    out << date.Year << " " << SDate::EnumToString(date.Month) << " "

        << date.Day;

    return out;

  }

public:

  static std::string\_view EnumToString(EMonth month) {

    switch (month) {

    case EMonth::January:      return "January";

    case EMonth::February:     return "February";

    case EMonth::March:       return "March";

    case EMonth::April:       return "April";

    case EMonth::May:       return "May";

    case EMonth::June:       return "June";

    case EMonth::July:       return "July";

    case EMonth::August:       return "August";

    case EMonth::September:    return "September";

    case EMonth::October:      return "October";

    case EMonth::November:     return "November";

    case EMonth::December:     return "December";

    }

    return "";

  }

};

class CCompany {

public:

  CCompany(const std::string &name, const std::string &location)

      : Name(name), Location(location) {}

public:

  std::string Name, Location;

};

class CJob {

public:

  CJob(const CCompany &employer, const std::string &pos, int salary)

      : Employer(employer), Position(pos), m\_salary(salary) {}

  friend std::ostream &operator<<(std::ostream &out, const CJob &job) {

    out << "Employer: " << job.Employer.Name << "\n"

        << "Work Address: " << job.Employer.Location << "\n"

        << "Position: " << job.Position << "\n"

        << "Salary: " << job.m\_salary;

    return out;

  }

public:

  CCompany Employer;

  std::string Position;

protected:

  int m\_salary;

};

class CPerson {

public:

  CPerson(const std::string &name, const std::string &country, const SDate &DoB,

          int height, int weight, const CJob &job)

      : Name(name), Country(country), m\_DoB(DoB), m\_height(height),

        m\_weight(weight), Job(job) {}

  void DisplayAllDetails() {

    std::cout << "Name: " << Name << "\n"

              << "DoB: " << m\_DoB << "\n"

              << "Height: " << m\_height << "\"\n"

              << "Weight: " << m\_weight << " kg\n"

              << "Country: " << Country << "\n"

              << "Job: {\n"

              << Job << "\n"

              << "}" << std::endl;

  }

public:

  std::string Name, Country;

  CJob Job;

protected:

  SDate m\_DoB;

  int m\_weight, m\_height;

};

int main() {

  CCompany company("ABC company", "Kathmandu");

  CJob job(company, "AI Researcher", 150'000);

  SDate DoB = {1990, EMonth::December, 15};

  CPerson ram("Ram", "Nepal", DoB, 6, 75, job);

  ram.DisplayAllDetails();

  footer();

}

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**Q2.2. Prolog Basics.**

a. About Language (What/ When/ Who/ Why)

b. Atoms, Variables, Facts and Rules in Prolog

**THEORY:**

Prolog is a **declarative programming language** developed in the early 1970s by **Alain Colmerauer and Robert Kowalski** for **artificial intelligence and symbolic reasoning**. Unlike procedural languages, Prolog focuses on **what is true rather than how to compute it**, making it ideal for knowledge representation and logical inference.

In Prolog, **atoms** are constant symbols representing objects or concepts,   
**variables** are placeholders that can take values,   
**facts** define known truths about the domain, and  
**rules** express relationships or logical implications between facts.

Together, these elements allow Prolog programs to model knowledge and automatically infer new information based on queries.

**Code:**

% [mary, pizza, john, alice, chocolate] are atoms aka constant symbols

likes(mary, pizza).

% These are the facts,

likes(john, mary).

likes(alice, chocolate).

% Rule :- <fact> {logical connectivities: and, or; not\+} <fact2>.

happy(X) :- likes(X, pizza); likes(X, chocolate).

% queries to automatically run as opposed to manually type in console

run\_queries :-

    % list of queries in prolog, TestQueries is var since it starts with upper case

    TestQueries = [happy(mary), happy(john), likes(john, Who), likes(alice, What)],

    % for loop, assigns member of TestQueries to Q and runs following commands

    % call(Q)->runs the Query, and choses true or the false statement (assigned to Result var)

    forall(member(Q, TestQueries),

           (

             (call(Q) -> Result = true ; Result = false),

    % format('') is printf, ~w is %s, ~n is \n, values are in list instead of open vars

             (format('~w -> ~w~n', [Q, Result]))

            )

    ).

% Initialization ensures run\_queries is executed automatically on load

:- initialization(run\_queries).

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

**Q2.3. Ancestor program using Prolog**

**THEORY:**

Prolog is a logic programming language often used in AI for knowledge representation and reasoning. This program demonstrates an "Ancestor" relationship using facts and rules. The system can infer if one person is an ancestor of another.

Key concepts used:

    - Facts: direct parent relationships

    - Rules: infer ancestor recursively

    - Queries: ask the system questions

**Code:**

% Facts: parent(Parent, Child)

parent(john, mary).

parent(mary, alice).

parent(john, mike).

parent(mike, bob).

% Rule: X is ancestor of Y if X is parent of Y

ancestor(X, Y) :- parent(X, Y).

% Rule: X is ancestor of Y if X is parent of Z and Z is ancestor of Y

ancestor(X, Y) :-

    parent(X, Z),

    ancestor(Z, Y).

run\_queries :-

    TestQueries = [ancestor(john, alice), ancestor(mary, bob), ancestor(john, bob)],

    forall(member(Q, TestQueries),

           (

             (call(Q) -> Result = true ; Result = false),

             (format('~w -> ~w~n', [Q, Result]))

            )

    ).

:- initialization(run\_queries).

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

**Q2.4.Family Relationship (family tree) using prolog**

**THEORY:**

A **Semantic Net (Semantic Network)** is a knowledge representation technique in Artificial Intelligence that organizes information in the form of a graph. In this structure, concepts or objects are represented as **nodes**, and the relationships between them are represented as **links (edges)**.

For example, if we have the concepts "Cat" and "Animal," a semantic net can represent their relationship through a link like *Cat → is\_a → Animal*. This allows hierarchical representation of knowledge, where properties can be inherited from general categories to specific instances (e.g., if "Animals can move," then "Cats can move" is automatically inferred).

Semantic nets are intuitive and easy to understand because they resemble how humans naturally associate ideas. They are particularly useful for representing structured knowledge, reasoning about relationships, and supporting inheritance. However, they can become complex when the knowledge base grows large and may struggle with exceptions or uncertain knowledge.

Here, a **family tree** is modeled using **facts and rules** in Prolog.

**Facts** define parent-child relationships (parent/2) and gender (male/1, female/1).   
**Rules** use these facts to

infer relationships: father/2 and mother/2   
determine parenthood by gender,   
finds grandparents via two parent links grandparent/2, and   
sibling/2 identifies siblings sharing the same parent.

The run\_queries/0 predicate tests a set of example queries, checking if statements like father(john, mary) or sibling(alice, bob) are true, and prints the results as true or false.

The program automatically runs run\_queries at startup due to the   
:- initialization(run\_queries). directive.

**Code:**

% Facts: parent(Parent, Child)

parent(john, mary).

parent(john, mike).

parent(mary, alice).

parent(mike, bob).

parent(alice, claire).

% Facts: gender

male(john). male(mike). male(bob).

female(mary). female(alice). female(claire).

% Rules

father(X, Y) :- parent(X, Y), male(X).

mother(X, Y) :- parent(X, Y), female(X).

grandparent(X, Y) :- parent(X, Z), parent(Z, Y).

sibling(X, Y) :- parent(Z, X), parent(Z, Y), X \= Y.

run\_queries:-

    TestQueries = [

        father(john, mary),

        mother(mary, alice),

        grandparent(john, alice),

        sibling(mary, mike),

        sibling(alice, bob)

    ],

    forall(member(Q, TestQueries),

           (

             (call(Q) -> Result = true ; Result = false),

             (format('~w -> ~w~n', [Q, Result]))

            )

    ).

:- initialization(run\_queries).

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

**Q2.5. Represent following facts in Semantic Net diagrammatically and also write a program in Prolog to represent the Semantic Net.**

⎯ Mat1 is a mat

⎯ Cat1 is a cat

⎯ Tom is a cat.

⎯ Bird1 is a bird.

⎯ Cat1 sat on Mat1.

⎯ Tom caught bird1.

⎯ Tom is owned by John.

⎯ Tom is ginger in color.

⎯ Cats like cream.

⎯ The cat sat on the mat.

⎯ A cat is a mammal.

⎯ A bird is an animal.

⎯ All mammals are animals.

⎯ Mammals have fur.

**THEORY:**

A **Semantic Net (Semantic Network)** is a knowledge representation technique in Artificial Intelligence that organizes information in the form of a graph. In this structure, concepts or objects are represented as **nodes**, and the relationships between them are represented as **links (edges)**.

For example, if we have the concepts "Cat" and "Animal," a semantic net can represent their relationship through a link like *Cat → is\_a → Animal*. This allows hierarchical representation of knowledge, where properties can be inherited from general categories to specific instances (e.g., if "Animals can move," then "Cats can move" is automatically inferred).

Semantic nets are intuitive and easy to understand because they resemble how humans naturally associate ideas. They are particularly useful for representing structured knowledge, reasoning about relationships, and supporting inheritance. However, they can become complex when the knowledge base grows large and may struggle with exceptions or uncertain knowledge. Overall, semantic networks provide a simple yet powerful way to model and reason about knowledge in AI.

**Code:**

% Objects and their types

is\_a(mat1, mat).

is\_a(cat1, cat).

is\_a(tom, cat).

is\_a(bird1, bird).

% Taxonomy / Hierarchy

is\_a(cat, mammal).

is\_a(bird, animal).

is\_a(mammal, animal).

% Relations

sat\_on(cat1, mat1).

sat\_on(cat, mat1). % general cat sat on mat

caught(tom, bird1).

owned\_by(tom, john).

color(tom, ginger).

likes(cat, cream).

has(mammal, fur).

run\_queries :-

    TestQueries = [

        is\_a(mat1, mat),        is\_a(cat1, cat),        is\_a(tom, cat),

        is\_a(bird1, bird),        sat\_on(cat1, mat1),   caught(tom, bird1),

        owned\_by(tom, john),      color(tom, ginger),   likes(cat, cream),

        sat\_on(cat, mat1),        is\_a(cat, mammal),    is\_a(bird, animal),

        is\_a(mammal, animal),     has(mammal, fur)

    ],

    forall(member(Q, TestQueries),

           (

             (call(Q) -> Result = true ; Result = false),

             (format('~w -> ~w~n', [Q, Result]))

           )

    ).

:- initialization(run\_queries).

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**Q2.6. Write a program to demonstrate the working of Naive Bayse classifier by taking a**

**suitable example problem.**

**THEORY:**

In Artificial Intelligence and Machine Learning, the Naive Bayes algorithm is a probabilistic classifier based on Bayes' theorem. It assumes that features (like words in an email) are independent given the class label (spam or not spam).

For spam detection:

- We calculate the probability of an email being spam

given the words it contains.

- Even though independence assumption is "naive"

(words in reality are not independent), it works well.

- The class with the highest probability is chosen.

Applications: Spam filters, document categorization, sentiment analysis

**Code:**

import os

from collections import defaultdict

from typing import List, Tuple, Dict

# Training dataset

training\_data: List[Tuple[str, str]] = [

    ("buy cheap now", "spam"),

    ("limited offer buy", "spam"),

    ("cheap deal discount", "spam"),

    ("meeting schedule today", "ham"),

    ("project deadline extended", "ham"),

    ("let us meet tomorrow", "ham"),

]

# Preprocess and count word frequencies per class

word\_counts: Dict[str, Dict[str, int]] = {"spam": defaultdict(int), "ham": defaultdict(int)}

class\_counts: Dict[str, int] = {"spam": 0, "ham": 0}

vocab: set = set()

for text, label in training\_data:

    class\_counts[label] += 1

    for word in text.split():

        word\_counts[label][word] += 1

        vocab.add(word)

def predict(text: str) -> str:

    """Predict if the given email text is spam or ham."""

    words: List[str] = text.split()

    results: Dict[str, float] = {}

    for label in ["spam", "ham"]:

        # Prior probability

        prob: float = class\_counts[label] / sum(class\_counts.values())

        # Likelihood with Laplace smoothing

        for word in words:

            prob \*= (word\_counts[label][word] + 1) / (sum(word\_counts[label].values()) + len(vocab))

        results[label] = prob

    return max(results, key=results.get)

def main() -> None:

    test\_emails: List[str] = ["cheap offer today", "project meeting tomorrow"]

    for email in test\_emails:

        print("Test email:", email)

        print("Prediction:", predict(email))

        print("-" \* 40)

    print("\nLab:", os.path.basename(\_\_file\_\_))

    print("Surab Parajuli\tSection: A\nRoll: 34\tSymbol: 80010139")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**Q2.7. WAP to develop a sample medical expert system capable of diagnosing disease based on the provided symptoms. (Recommended Language : Prolog or Python)**

**THEORY:**

In Artificial Intelligence, an Expert System is a program that mimics human decision-making by using a knowledge base of rules and an inference engine to draw conclusions.

For medical diagnosis:

- The knowledge base stores rules mapping symptoms to diseases.

- The inference engine checks which disease best matches

the observed symptoms.

- Such systems assist doctors in decision-making and can

provide preliminary diagnosis.

Applications: Medical diagnosis, troubleshooting, recommendation systems.

**Code:**

import os

from typing import List, Dict

# Knowledge base: Disease -> List of symptoms

knowledge\_base: Dict[str, List[str]] = {

    "Flu": ["fever", "cough", "sore throat", "fatigue"],

    "Cold": ["cough", "sneezing", "runny nose"],

    "Migraine": ["headache", "nausea", "sensitivity to light"],

    "Chickenpox": ["fever", "rash", "fatigue", "itching"],

}

def diagnose(symptoms: List[str]) -> str:

    """Diagnose disease based on symptoms using simple rule matching."""

    max\_match = 0

    probable\_disease = "Unknown"

    for disease, disease\_symptoms in knowledge\_base.items():

        match\_count = len(set(symptoms) & set(disease\_symptoms))

        if match\_count > max\_match:

            max\_match = match\_count

            probable\_disease = disease

    return probable\_disease

def main() -> None:

    test\_cases: List[List[str]] = [

        ["fever", "cough", "fatigue"],      # Should likely match Flu

        ["headache", "nausea", "fatigue"],  # Should likely match Migraine

    ]

    for i, symptoms in enumerate(test\_cases, 1):

        print(f"Test Case {i}: Symptoms -> {symptoms}")

        print("Diagnosis:", diagnose(symptoms))

        print("-" \* 40)

    print("\nLab:", os.path.basename(\_\_file\_\_))

    print("Surab Parajuli\tSection: A\nRoll: 34\tSymbol: 80010139")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**

A computer screen shot of a computer program

AI-generated content may be incorrect.

**Q2.8. Realization of AND, OR and NOT gates using Artificial Neurons (Recommended**

**Language : Python)**

**THEORY:**

In AI, an artificial neuron (or perceptron) models a simple biological neuron. It computes a weighted sum of inputs and passes it through an activation function (here, a step function).

Logic gates (AND, OR, NOT) can be realized using a single neuron by carefully choosing weights and bias.

- AND: fires only if all inputs are 1

- OR: fires if at least one input is 1

- NOT: inverts a single input

This illustrates how simple neural networks can perform logical reasoning.

**Code:**import os

from typing import List

def neuron(inputs: List[int], weights: List[int], bias: int) -> int:

    """Single artificial neuron using step activation function."""

    total = sum(i\*w for i, w in zip(inputs, weights)) + bias

    return 1 if total > 0 else 0

def AND\_gate(x1: int, x2: int) -> int:

    return neuron([x1, x2], weights=[1, 1], bias=-1)

def OR\_gate(x1: int, x2: int) -> int:

    return neuron([x1, x2], weights=[1, 1], bias=0)

def NOT\_gate(x: int) -> int:

    return neuron([x], weights=[-1], bias=0)

def main() -> None:

    print("AND Gate")

    for x1 in [0, 1]:

        for x2 in [0, 1]:

            print(f"{x1} AND {x2} = {AND\_gate(x1, x2)}")

    print("-" \* 40)

    print("OR Gate")

    for x1 in [0, 1]:

        for x2 in [0, 1]:

            print(f"{x1} OR {x2} = {OR\_gate(x1, x2)}")

    print("-" \* 40)

    print("NOT Gate")

    for x in [0, 1]:

        print(f"NOT {x} = {NOT\_gate(x)}")

    print("-" \* 40)

    print("\nLab:", os.path.basename(\_\_file\_\_))

    print("Surab Parajuli\tSection: A\nRoll: 34\tSymbol: 80010139")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**Q2.9. A Simple Example of Back Propagation Learning (Recommended Language : Python)**

**THEORY:**

Backpropagation is a supervised learning algorithm for training artificial neural networks. It minimizes the error by propagating the output error backwards and updating the weights using gradient descent.

Process:

- Forward pass: compute network output for given inputs.

- Compute error: difference between predicted and target output.

- Backward pass: propagate error to update weights.

Applications: Pattern recognition, function approximation, AI learning.

**Code:**

import os

import math

from typing import List, Tuple

def sigmoid(x: float) -> float:

    return 1 / (1 + math.exp(-x))

def sigmoid\_derivative(x: float) -> float:

    return x \* (1 - x)

class SimpleNN:

    def \_\_init\_\_(self) -> None:

        # Initialize weights and biases

        self.w\_input\_hidden = [[0.5, -0.4], [0.3, 0.1]]  # 2x2 weights

        self.b\_hidden = [0.0, 0.0]

        self.w\_hidden\_output = [0.2, -0.3]  # 1x2 weights

        self.b\_output = 0.0

        self.learning\_rate = 0.5

    def forward(self, x: List[float] | List[int]) -> float:

        self.h\_input = [

            x[0]\*self.w\_input\_hidden[0][0] + x[1]\*self.w\_input\_hidden[1][0] + self.b\_hidden[0],

            x[0]\*self.w\_input\_hidden[0][1] + x[1]\*self.w\_input\_hidden[1][1] + self.b\_hidden[1],

        ]

        self.h\_output = [sigmoid(h) for h in self.h\_input]

        self.o\_input = sum(h\*w for h, w in zip(self.h\_output, self.w\_hidden\_output)) + self.b\_output

        self.o\_output = sigmoid(self.o\_input)

        return self.o\_output

    def train(self, x: List[float] | List[int], y: float) -> None:

        # Forward pass

        output = self.forward(x)

        # Output layer error

        error = y - output

        delta\_output = error \* sigmoid\_derivative(output)

        # Hidden layer error

        delta\_hidden = [

            delta\_output \* self.w\_hidden\_output[i] \* sigmoid\_derivative(self.h\_output[i])

            for i in range(2)

        ]

        # Update weights hidden->output

        for i in range(2):

            self.w\_hidden\_output[i] += self.learning\_rate \* delta\_output \* self.h\_output[i]

        self.b\_output += self.learning\_rate \* delta\_output

        # Update weights input->hidden

        for i in range(2):

            self.w\_input\_hidden[0][i] += self.learning\_rate \* delta\_hidden[i] \* x[0]

            self.w\_input\_hidden[1][i] += self.learning\_rate \* delta\_hidden[i] \* x[1]

            self.b\_hidden[i] += self.learning\_rate \* delta\_hidden[i]

def main() -> None:

    nn = SimpleNN()

    training\_data = [

        ([0, 0], 0),

        ([0, 1], 1),

        ([1, 0], 1),

        ([1, 1], 0),

    ]  # Learning XOR for demo

    # Train for 1000 epochs

    for epoch in range(1000):

        for x, y in training\_data:

            nn.train(x, y)

    # Test examples

    test\_cases = [[0, 0], [1, 1]]

    for i, x in enumerate(test\_cases, 1):

        print(f"Test case {i}: Input {x} -> Output {nn.forward(x):.4f}")

    print("\nLab:", os.path.basename(\_\_file\_\_))

    print("Surab Parajuli\tSection: A\nRoll: 34\tSymbol: 80010139")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

**Q2.10. WAP to solve N Queen Problem.(Recommended Language: Python)**

(Problem: To find an arrangement of N queens on a chess board of size N×N, such that

no queen can attack any other queens on the board).

**THEORY:**

The N-Queens problem is a classic example in Artificial Intelligence and combinatorial search. The goal is to place N queens on an N×N chessboard such that no two queens threaten each other.

Backtracking is an AI search technique:

- Place a queen in a row and recursively attempt to place the next queen.

- If a conflict arises, backtrack and try a different column.

- Continue until a solution is found or all options are exhausted.

Applications: Constraint satisfaction problems, scheduling, puzzle solving.

**Code:**import os

from typing import List

def is\_safe(board: List[int], row: int, col: int) -> bool:

    for r in range(row):

        c = board[r]

        if c == col or abs(c - col) == abs(r - row):

            return False

    return True

def solve\_n\_queens\_util(board: List[int], row: int, N: int) -> bool:

    if row == N:

        return True  # All queens placed successfully

    for col in range(N):

        if is\_safe(board, row, col):

            board[row] = col

            if solve\_n\_queens\_util(board, row + 1, N):

                return True

            board[row] = -1  # Backtrack

    return False

def solve\_n\_queens(N: int) -> List[int]:

    """Return one solution as a list where index = row, value = column."""

    board = [-1] \* N

    if solve\_n\_queens\_util(board, 0, N):

        return board

    else:

        return []

def print\_board(board: List[int]) -> None:

    N = len(board)

    for row in range(N):

        line = ""

        for col in range(N):

            if board[row] == col:

                line += "Q "

            else:

                line += ". "

        print(line)

    print("-" \* 40)

def main() -> None:

    N: int = int(input("Enter the value of N: "))

    solution = solve\_n\_queens(N)

    if solution:

        print(f"One solution for N={N}:")

        print\_board(solution)

    else:

        print(f"No solution exists for N={N}")

    print("\nLab:", os.path.basename(\_\_file\_\_))

    print("Surab Parajuli\tSection: A\nRoll: 34\tSymbol: 80010139")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**Q2.11. WAP to solve Water Jug Problem. (Recommended Language: Python)**

(Problem statement: Given two jugs, a 4-gallon and 3-gallon having no measuring

markers on them. There is a pump that can be used to fill the jugs with water and the

water can be poured on the ground. How can you get exactly 2 gallons of water into 4-

gallon jug?)

**THEORY:**

The Water Jug problem is a classic AI search problem that can be solved using state-space search techniques like BFS or DFS.

Problem:

- Two jugs of different capacities (4-gallon and 3-gallon) with no

measuring markers.

- Goal: Get exactly 2 gallons in the 4-gallon jug.

Approach:

- Represent states as (amount\_in\_4\_gal, amount\_in\_3\_gal).

- Use BFS to explore all possible actions:

Fill a jug, empty a jug, pour one jug into another.

- Stop when the goal state is reached.

Applications: Constraint satisfaction, planning, AI problem-solving.

**Code:**import os

from collections import deque

from typing import Tuple, List, Set

def get\_successors(state: Tuple[int, int]) -> List[Tuple[int, int]]:

    jug4, jug3 = state

    successors = []

    # Fill actions

    successors.append((4, jug3))  # Fill 4-gal

    successors.append((jug4, 3))  # Fill 3-gal

    # Empty actions

    successors.append((0, jug3))  # Empty 4-gal

    successors.append((jug4, 0))  # Empty 3-gal

    # Pour 4->3

    pour = min(jug4, 3 - jug3)

    successors.append((jug4 - pour, jug3 + pour))

    # Pour 3->4

    pour = min(jug3, 4 - jug4)

    successors.append((jug4 + pour, jug3 - pour))

    return successors

def bfs\_solution():

    start = (0, 0)

    goal = 2

    queue = deque([(start, [start])])

    visited: Set[Tuple[int, int]] = set()

    while queue:

        state, path = queue.popleft()

        if state in visited:

            continue

        visited.add(state)

        if state[0] == goal:

            return path

        for succ in get\_successors(state):

            if succ not in visited:

                queue.append((succ, path + [succ]))

    return []

def main() -> None:

    path = bfs\_solution()

    if path:

        print("Steps to get 2 gallons in 4-gallon jug:")

        for step in path:

            print(f"4-gal: {step[0]}, 3-gal: {step[1]}")

    else:

        print("No solution found.")

    print("\nLab:", os.path.basename(\_\_file\_\_))

    print("Surab Parajuli\tSection: A\nRoll: 34\tSymbol: 80010139")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**

A computer screen shot of a program

AI-generated content may be incorrect.

**Q2.12. Write a program to demonstrate the steps in genetic algorithm taking suitable example problem.  
THEORY:**

Genetic Algorithms (GA) are search and optimization techniques inspired by natural evolution. They use mechanisms like selection, crossover, and mutation to evolve solutions over generations.

Steps:

1. Initialize a population of candidate solutions.

2. Evaluate fitness of each individual.

3. Select parents based on fitness.

4. Apply crossover to produce offspring.

5. Apply mutation to maintain diversity.

6. Repeat for multiple generations.

Applications: Optimization, scheduling, machine learning, AI problem-solving.

**Code:**import os

import random

from typing import List, Tuple

# Parameters

POP\_SIZE = 6

CHROM\_LENGTH = 5  # Binary string length to represent numbers 0-31

GENS = 5

MUTATION\_RATE = 0.1

def binary\_to\_int(chrom: str) -> int:

    return int(chrom, 2)

def fitness(chrom: str) -> int:

    x = binary\_to\_int(chrom)

    return x \* x

def create\_population() -> List[str]:

    return [''.join(random.choice('01') for \_ in range(CHROM\_LENGTH)) for \_ in range(POP\_SIZE)]

def select\_pair(pop: List[str]) -> Tuple[str, str]:

    total\_fitness = sum(fitness(c) for c in pop)

    probs = [fitness(c)/total\_fitness for c in pop]

    return random.choices(pop, weights=probs, k=2)

def crossover(p1: str, p2: str) -> Tuple[str, str]:

    point = random.randint(1, CHROM\_LENGTH-1)

    return p1[:point]+p2[point:], p2[:point]+p1[point:]

def mutate(chrom: str) -> str:

    chrom\_list = list(chrom)

    for i in range(len(chrom\_list)):

        if random.random() < MUTATION\_RATE:

            chrom\_list[i] = '1' if chrom\_list[i] == '0' else '0'

    return ''.join(chrom\_list)

def main() -> None:

    for i in range(2):

        print(f"{"-" \* 20} Run-{i+1}{"-" \* 20}")

        pop = create\_population()

        print("Initial Population:", pop)

        for gen in range(1, GENS+1):

            new\_pop = []

            while len(new\_pop) < POP\_SIZE:

                p1, p2 = select\_pair(pop)

                c1, c2 = crossover(p1, p2)

                c1, c2 = mutate(c1), mutate(c2)

                new\_pop.extend([c1, c2])

            pop = new\_pop[:POP\_SIZE]

            best = max(pop, key=fitness)

            print(f"Generation {gen}: Best={best} ({binary\_to\_int(best)}), Fitness={fitness(best)}")

        print("Final Population:", pop)

        best = max(pop, key=fitness)

        print("Best Solution:", best, "=", binary\_to\_int(best))

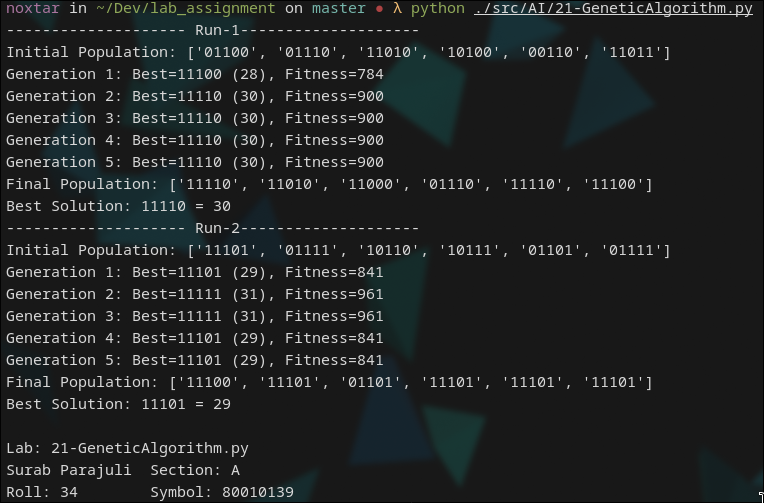
    print("\nLab:", os.path.basename(\_\_file\_\_))

    print("Surab Parajuli\tSection: A\nRoll: 34\tSymbol: 80010139")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**



**Q2.13. WAP to implement Naïve Bayse Algorithm to demonstrate binary classification. Use any dataset.**

**Repeat of Q2.6**

**Q2.14. WAP to demonstrate some NLP tasks using NLTK. (Recommended Language: Python)**

**[Sentence tokenization, Word tokenization, stop words filtering, word stemming, POS**

**tagging, etc.]  
THEORY:**

Natural Language Processing (NLP) is a branch of AI that deals with interaction between computers and human language. Common tasks:

- Sentence tokenization: Splitting text into sentences

- Word tokenization: Splitting sentences into words

- Stop words removal: Removing common words like 'the', 'is'

- Stemming: Reducing words to their root form

- POS tagging: Assigning parts of speech (noun, verb, etc.)

NLTK is a Python library that provides tools for these NLP tasks.

Applications: Text preprocessing, sentiment analysis, chatbots, information retrieval.

**Code:**import os

from typing import List

import nltk

from nltk.tokenize import sent\_tokenize, word\_tokenize

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

def main() -> None:

    nltk.download('wordnet')

    nltk.download('punkt\_tab')

    nltk.download('stopwords')

    nltk.download('averaged\_perceptron\_tagger\_eng')

    text = ("Natural Language Processing (NLP) is a field of Artificial Intelligence. "

            "It enables machines to understand and interpret human language.")

    print("Original Text:")

    print(text)

    print("-" \* 40)

    # Sentence Tokenization

    sentences: List[str] = sent\_tokenize(text)

    print("Sentence Tokenization:")

    print(sentences)

    print("-" \* 40)

    # Word Tokenization

    words: List[str] = word\_tokenize(text)

    print("Word Tokenization:")

    print(words)

    print("-" \* 40)

    # Stop words filtering

    stop\_words = set(stopwords.words('english'))

    filtered\_words: List[str] = [w for w in words if w.lower() not in stop\_words and w.isalpha()]

    print("After Stop Words Removal:")

    print(filtered\_words)

    print("-" \* 40)

    # Word Stemming

    stemmer = PorterStemmer()

    stemmed\_words: List[str] = [stemmer.stem(w) for w in filtered\_words]

    print("After Stemming:")

    print(stemmed\_words)

    print("-" \* 40)

    # POS Tagging

    pos\_tags = nltk.pos\_tag(filtered\_words)

    print("POS Tagging:")

    print(pos\_tags)

    print("-" \* 40)

    print("\nLab:", os.path.basename(\_\_file\_\_))

    print("Surab Parajuli\tSection: A\nRoll: 34\tSymbol: 80010139")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**

A computer screen shot of text

AI-generated content may be incorrect.