6.

def tsp\_nearest\_neighbor(dist\_matrix, start=0):

    """

    Solves the TSP problem using the Nearest Neighbor heuristic.

    Parameters:

    - dist\_matrix: A 2D list representing the distance matrix between cities.

    - start: The starting city index. Default is 0.

    Returns:

    - path: A list of city indices representing the TSP path.

    - total\_cost: The total distance of the TSP path.

    """

    n = len(dist\_matrix)

    path = [start]

    total\_cost = 0

    unvisited = set(range(n))

    unvisited.remove(start)

    current\_city = start

    while unvisited:

        # Find the nearest unvisited city to the current city

        next\_city = min(unvisited, key=lambda city: dist\_matrix[current\_city][city])

        total\_cost += dist\_matrix[current\_city][next\_city]

        path.append(next\_city)

        unvisited.remove(next\_city)

        current\_city = next\_city

    # Close the loop by returning to the starting city

    total\_cost += dist\_matrix[current\_city][start]

    path.append(start)

    return path, total\_cost

# Example usage

dist\_matrix = [[0, 29, 20, 21], [29, 0, 15, 17], [20, 15, 0, 28], [21, 17, 28, 0]]

start\_city = 0

path, total\_cost = tsp\_nearest\_neighbor(dist\_matrix, start=start\_city)

print("Approximate TSP Path:", path)

print("Total Distance:", total\_cost)

7.

class ForwardChaining:

    def \_\_init\_\_(self):

        self.facts = set()

        self.rules = []

    def add(self, fact=None, condition=None, conclusion=None):

        if fact: self.facts.add(fact)

        elif condition and conclusion: self.rules.append((condition, conclusion))

    def infer(self):

        while True:

            new\_facts = False

            for condition, conclusion in self.rules:

                if condition.issubset(self.facts) and conclusion not in self.facts:

                    print(f"Applying rule: if {condition} then {conclusion}")

                    self.facts.add(conclusion)

                    new\_facts = True

            if not new\_facts: break

    def get\_facts(self):

        return self.facts

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

    fc = ForwardChaining()

    fc.add(fact="fever")

    fc.add(fact="cough")

    fc.add(condition={"fever", "cough"}, conclusion="possible flu")

    fc.add(condition={"fever"}, conclusion="possible infection")

    fc.add(condition={"cough"}, conclusion="possible bronchitis")

    fc.infer()

    print("\nFinal Diagnoses:")

    for fact in fc.get\_facts():

        print(fact)

8.

from sympy import symbols, Not

from itertools import combinations

def resolve(clause1, clause2):

    """ Resolve two clauses and return the resulting resolvent. """

    resolvent = set()

    for literal1 in clause1:

        for literal2 in clause2:

            if literal1 == Not(literal2) or literal2 == Not(literal1):

                resolvent.update({l for l in (clause1 + clause2) if l not in (literal1, literal2)})

    return list(resolvent)

def resolution(clauses):

    """ Apply resolution to a set of clauses until no new clauses can be generated. """

    new\_clauses = [tuple(sorted(clause, key=lambda x: str(x))) for clause in clauses]

    new\_clauses\_set = set(new\_clauses)

    while True:

        n = len(new\_clauses\_set)

        print("Current Clauses:")

        for clause in new\_clauses\_set:

            print(clause)

        print()

        pairs = list(combinations(new\_clauses\_set, 2))

        for clause1, clause2 in pairs:

            print(f"Resolving: {clause1} and {clause2}")

            resolvent = resolve(list(clause1), list(clause2))

            print(f"Resolvent: {resolvent}")

            print()

            if not resolvent:

                # Empty clause found, contradiction reached

                return True

            resolvent = tuple(sorted(resolvent, key=lambda x: str(x)))

            if resolvent not in new\_clauses\_set:

                new\_clauses\_set.add(resolvent)

        if n == len(new\_clauses\_set):

            # No new clauses can be generated, exit loop

            return False

# Example usage:

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

    # Example clauses in CNF (Conjunctive Normal Form)

    P, Q = symbols('P Q')

    clause1 = [P, Not(Q)]

    clause2 = [Not(P), Q]

    clause3 = [Not(P), Not(Q)]

    # List of clauses

    clauses = [clause1, clause2, clause3]

    result = resolution(clauses)

    if result:

        print("The set of clauses is unsatisfiable (contradiction found).")

    else:

        print("The set of clauses is satisfiable.")