Project 2

Name: Yun Du

UIN :330006637

**The main idea of Program:**

The main idea behind my code is to create an algorithm that checks a set of constraints to see if a certain condition can be satisfied, and then modifies the input data to find a solution or determine that none exists. To find a feasible solution, it first solves "lead to" conditions and then checks "False must exist" conditions.

**The pseudo code:**

function algoriam\_design(n, P, Q, k\_list, T\_list, M\_list):

x = [0] \* n # Initialize all variable to 0 (False)

# Solve "lead to" condition first

set\_cnt = {T\_list[i][0] for i in range(P) if k\_list[i] == 0}

while True:

made\_changes = False

for i in range(P):

if k\_list[i] == -1:

continue

elif k\_list[i] == 0:

set\_cnt.add(T\_list[i][0])

k\_list[i] -= 1

made\_changes = True

else:

cnt = k\_list[i]

j = 0

while j < cnt:

if T\_list[i][j] in set\_cnt:

T\_list[i].pop(j)

k\_list[i] -= 1

cnt -= 1

made\_changes = True

else:

j += 1

if not made\_changes:

break

# Solve "False must exist" condition

check = all(not set(M\_list[i]).issubset(set\_cnt) for i in range(Q))

if check:

x = [1 if i in set\_cnt else 0 for i in range(n)]

else:

x = []

return x

**The proof of correctness:**

Let's start with the case where there is a viable solution. The algorithm begins by iteratively adding the necessary variables to a set set cnt to solve the "lead to" constraints. After solving all "lead to" constraints, the algorithm checks the "False must exist" constraints by ensuring that none of the variables in the corresponding sets are present in set cnt. If this test is successful, the algorithm returns a viable solution by setting variables in set cnt to 1 and the rest to 0.

Consider the case where there is no feasible solution. The constraints "lead to" and "False must exist" cannot be satisfied in this case. The algorithm begins by iteratively adding variables to set cnt to solve the "lead to" constraints. If it is unable to satisfy all "lead to" constraints, it will be unable to satisfy the "False must exist" constraints, and will thus correctly return an empty list. If it meets all of the "lead to" constraints, the algorithm checks the "False must exist" constraints by ensuring that none of the variables from the corresponding sets are present in set cnt. If such a variable exists, the algorithm cannot satisfy the "False must exist" constraints and thus returns an empty list.

As a result, the algorithm is correct and will return a feasible solution if one exists or an empty list if none exist.

**The Analysis of the time complexity:**

The time complexity of the code is O(P \* k\_max^2 + Q \* k\_max), where P is the number of "lead to" constraints, Q is the number of "False must exist" constraints, and k\_max is the maximum size of any list in T\_list.

The main time-consuming part of the algorithm is the loop that solves the "lead to" constraints. The loop iterates over each constraint P times, and for each constraint, it may need to iterate over the list T\_list[i] up to k\_max times, checking whether each element is in the set\_cnt set. Therefore, the time complexity of this loop is O(P \* k\_max^2).

The "False must exist" constraints are checked by iterating over the M\_list and checking whether the corresponding set is a subset of set\_cnt. This loop has a time complexity of O(Q \* k\_max), since each set may have up to k\_max elements.

Therefore, the overall time complexity of the code is O(P \* k\_max^2 + Q \* k\_max).