CONDITION SATISFIABILITY PROBLEM

Main Idea :-

If $x_1, x_2 \cdots x_n$ are 'n' Boolean Variables which mean x_i is either true or false

$\vee$ => OR

$\wedge$ => AND

$\bar{\ }$ => NOT

There are two different types of conditions

① "lead to"

$$(x_{i_1} \wedge x_{i_2} \wedge \cdots \wedge x_{i_k}) \Rightarrow x_j$$

It means that "if $x_{i_1}, x_{i_2} \cdots x_{i_k}$" are all true, then x_j is also true."

Degenerate type has $k=0$, of form

$$\Rightarrow x_j$$

In this case x_j is true

② "False must exist"

$$(\bar{x}_{i_1} \vee \bar{x}_{i_2} \vee \cdots \vee \bar{x}_{i_k})$$

among above 'k' atleast one has to be true. The above is true if and only if atleast one of them is False

Given variables in the dataset

"numInstances" => total number of instances

"n_list" => total number of boolean vars in i^{th} instance

"P_list" => total number of lead-to conditions in i^{th} instance

"Q_list" => total number of false must exist conditions in i^{th} instance

"k_list" => Number of boolean vars to the left of a lead-to condition

"m_list" => Number of boolean vars in the false must exist condition.

"T_list" => Has lead-to condition variables, last variable is the variable on the right side of lead-to

"M_list" => Has the false must exist condition variables

To solve this problem, I have first initialised all the variables to False.

Then I have iterated through the lead-to conditions in the increasing order of number of elements to the left.

If there are no elements to the left, I will directly modify the right element to True.

If there are elements to the left, then I will check if all of them are true, In that case I will change the right element to true.

I will run the above lead to code till there is no update to the variables.

As I have already initialised them with False and only modified the necessary variable to True on the basis of lead to conditions, I will not modify any True to False as it will certainly fail the lead to conditions.

So, I will just check if the unmodified X variable values satisfy all the False must exist, if it fails any False must exist then I will return [] indicating that there is no satisfying solution.

Pseudo code :-

```
FIND_VARS_COND_SATISFY (dataset):
    data_size = dataset['numInstances']
    res = []
    for iter_data in range(data_size):
        n = dataset['n_list'][iter_data]
        P = dataset['P_list'][iter_data]
        Q = dataset['Q_list'][iter_data]
        K = dataset['K_list'][iter_data]
        M = dataset['M_list'][iter_data]
        T = dataset['T_list'][iter_data]
        M = dataset['M_list'][iter_data]

        X = [False] * n
        l_f = 0

        K_sort_id = np.argsort(np.array(K))
```

```
while(1):
    x_before = list(x)
    for i in x_sort_id:
        T_i = T[i]
        n_i = K[i]
        if n_i == 0:
            if not x[T_i[0]]:
                x[T_i[0]] = True
            continue
        left = [None]*(n_i)
        for j in range(n_i):
            left[j] = x[T_i[j]]
        left = all(left)
        if left:
            x[T_i[n_i]] = True
    if x_before == x:
        break
for i in range(q):
    M_i = M[i]
    n_i = m[i]
    left = [None]*n_i
```

```
for j in range(n_i):
    left[j] = x[M_i[j]]

left_false = all(left)

if left_false:
    l_f = 1

new_lst = [1 if i else 0 for i in x]

res.append(new_lst if l_f == 0 else [])

return res
```

Proof of Correctness

The above algorithm initialises all the variables to false initially. It only modifies the necessary variables to true while traversing through the lead to conditions in the increasing order of the number of variables on the left side of the equation. The lead to conditions are reevaluated till they no longer change. The algorithm does not perform any modification with respect to false must exist, because the change in any true value will fail the lead to condition that was the reason behind setting this particular variable to True from false. The algorithm just checks, if the modifications after the lead to iterations satisfy all the false must exist conditions and if they satisfy these modifications are the final modifications else it will return an empty [] list. As the variables are iterated till they no longer need modification are not modified with respect to false must exist, the algorithm considers all

conditions and is correct.

Time Complexity analysis

The following analysis is made for a single data instance

n -> total number of variables
p -> total number of lead to conditions

```
# p is total number of lead to conditions
# k has the number of left vars for all the p eq

# lets store indices in k in asc order

arr = np.array(k)
k_sort_id = np.argsort(arr)

while(1):

    x_before = list(x)

    for i in k_sort_id:

        # first set all that is 0 in the left to true

        T_i = T[i]

        #left vars
        n_i = k[i]

        if n_i == 0:
            if not x[T_i[0]]:
                x[T_i[0]] = True
            continue

        #now check for all the conditions


        #laod all the left values
        left = [None] * (n_i)
        for j in range(n_i):
            left[j] = x[T_i[j]]

        #check if everything in left is true
        left = all(left)

        #if left is true then change the right to true
        if left:
            x[T_i[n_i]] = True

    if x_before == x:
        break


# for every false must exist condition
for i in range(q):

    #list of vars in the condition
    M_i = M[i]

    #total vars
    n_i = m[i]


    #load all elements
    left = [None] * n_i
    for j in range(n_i):

        left[j] = x[M_i[j]]


    #check if there exists atleast 1 false
    left_false = all(left)
    # if there is no false, then there is no solution and final output is null
    if left_false:
        l_f = 1

new_lst = [1 if i else 0 for i in x]

res.append(new_lst if l_f == 0 else [])
```

Sorting -> $O(p \log p)$

max = n (till n vars change to true)

p iterations

$O(np)$

max = $n-1 \approx n$ -> $O(n)$

$O(n)$

$O(n^2 p)$

q iterations

max = n -> $O(n)$

$O(nq)$

Assuming p & q dont have very big difference.

Dominant term is $O(pn^2)$

∴ the time complexity $= O(pn^2)$