ANALYSIS OF ALGOGITHM

PROJECT : 2

Name : ASHUTOSH CHAUHAN

UIN: 232009024

Q.) The "Condition Satisfiability Problem"

MAIN IDEA

⇒ Let $x_1, x_2, \ldots x_n$ be n Boolean variables where each variable x_i can take one of only two possible values: "True" or "False". We need to make sure that following conditions are met. In lead-to condition if it has form $\Rightarrow x_j$, it should be "True" or if all LHS are 'True' than RHS should also be 'true'. In False-must-exist cond" to pass, one variable should be "False" in each condition.

⇒ The main idea is to initialize a solution for every instance as an array with all variables as 'FALSE'. Than we will greedily change the min possible variables to 'TRUE' such that all lead-to conditions follows. After this we will check false-must-exist conditions. If our solution satisfy all false-must exist condition then it is a possible solution else no solution will exist for that particular instance.

⇒ We will sort the lead-to conditions basis the no. of variables and will loop through each. We will check each condition, if size is 1 i.e it only has RHS then we will change the corresponding variable value in our solution to TRUE, if size > 1 and all LHS variable values are TRUE then we will assign RHS variable value in our solution to TRUE

We will run & check the above conditions for all conditions in P. If throughout the loop there is any assignment to our solution, we will repeat complete loop again, else we will stop.
After this we will loop through every False-must exist condition and check if it have any variable as FALSE. If all the conditions pass then we return the solution for this instance whereas if single false-must exist condition fails then we could conclude that given instance has no possible solution.

~~# Contradiction:~~

Proof of Correctness

~~Case case~~ To prove the correctness of this algorithm, we need to show two things :-

- If there exists a satisfying solution, the algorithm will find it.
- If there is no satisfying solution, the algo will correctly return "no satisfying solution exists".

Suppose there exists a satisfying solution that satisfy all conditions. We will show that the greedy algorithm will find it.
We are initializing all the variables as "False" during start.

We will then make min possible variables in our solution to 'TRUE' such that all lead-to conditions are satisfied by following ways. First making variables corresponding to degenerate conditions as 'true' and then checking iteratively if there exist a condition where all LHS variables became 'true', If yes then assigning its RHS variable also a 'true'. This will only change minimum number of mandatory assignments required to pass lead-to conditions. Since we made minimum assignments it means our solution has maximum number of "False" available in it.

Since we checking in each False-must exist condition for ~~a~~ atleast 1 false so our solution should satisfy all. If any single condition fails we could say that solution does not exist. We can prove this using contradiction. If any false-must exist condition fails it means it have all variables as 'true'. For it to pass some variable must be changed to "false". But as proved above our solution already have max no. of possible ~~solutions~~ "false", so it is not possible.

Pseudo code

⇒ function condition_satisfy(n, P, Q, k, m, T, M):

```
instance_soln = [0]*n      # initializing soln array
ptracker = [0]*P           # initializing a array to track lead-to conditions.
k_sorted = np.argsort(k)   # getting indexes of lead-to condition sorted by size.
                           └O(PlogP)
lead_condition = 1
while lead_condition == 1:                                   - O(1)
    lead_condition = 0                                       — O(Pn)
    for i in k_sorted:                                       - O(1)
        lead_flag = False
        if ptracker[i] = 0:
            if k[i] == 0:
                instance_soln[T[i][0]] = 1
                ptracker[i] = 1
            else:                                            O(n)
                for idx, j in enumerate(T[i][:-1]):
                    if instance_solution[j] == 0:
                        lead_flag = True
                        break.
            if lead_flag == False:
                instance_soln[T[i][-1]] = 1
                ptracker[i] = 1
                lead_cond = 1.
```

Iteratively assigning variable values to TRUE such that all lead-to conditions pass.

false_must_exist_flag = False,
for q in range(Q): ..... O(Qn)
    create array of variable for q^th case ... O(n).
    if all TRUE in above array:
        false_must_exist_flag = ~~TRUE~~ True
        break.
return instance_soln if false_must_exist_flag == False else []

⇒ function main :-
read instances using pickle
share n, P, Q, K, m, T, M values of every instances in loop.
satisfying_soln = Condition_Satisfy(n, P, Q, K, m, T, M)
Append output in solution list.

Time Complexity Analysis :-
From above pseudo code and corresponding time analysis, we can observe that O(P*n + Q*n) time is required for reading input. O(Qn) time is required to validate false-must-exist conditions. For lead-to condition it will take O(n*(Pn)) if P > n and O(P*(Pn)) if n > P.

∴ Overall Time Complexity = O(min(P, n) * (P*n) + Q*n)