Aim :- To solve 1-1 Assignment Problem Optimally.

Algorithm :-

Backtracking Solution

Explores each Solution and If obtained Solution has Better assignment then update the final solution.

Either We can assign a user an Interval from its favourable intervals or left it unassigned and repeat the same for Next User.

This can be achieved either by Iteratively or recursively.

Recursive solution is easier to implement and backtrack.

Description of Code:-

We have one array UserAssigned to keep track of interval assigned to that user and is set as –1 if not assigned any interval.

We have another array IntervalAssigned to keep track of user assigned to that interval and is set as –1 if not assigned any user.

We have vector of vector of pair of integers (User and assigned Interval) to store the various obtained.

Whenever we arrive at base case of recurssion , we have a solution and we pop previous solution having less assignment than the current solution.

After the algorithm terminates we obtained various solution.

We sort the solution vectot on the basis of assignments (Size of Solution).

We print top 4 solutions.

Code in cpp:-

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;    const int N=1e2+5;  vector<vector<int>> FeasibleInterVals(N);  vector<int> UserAssigned(N,-1);  vector<int> InterValAssigned(N,-1);  vector<vector<pair<int,int>>> Solution;    void assignInterVals(int userId, int noUser)  { //termination condition arrived and store the solution    if(userId==noUser+1)  {  vector<pair<int,int>> PossibleSolution;  for(int user=1;user<=noUser;user++)  {  if(UserAssigned[user]>0)  PossibleSolution.push\_back(make\_pair(user,UserAssigned[user]));  }  Solution.push\_back(PossibleSolution);  //deleting smaller solutions    for(auto oneSolution =Solution.begin();oneSolution!=Solution.end();oneSolution++)  {  if(oneSolution->size()<PossibleSolution.size())  {  Solution.erase(oneSolution);  oneSolution--;  }  }  return;  }  //excluding any assignment  assignInterVals(userId+1,noUser);    for(int interval : FeasibleInterVals[userId])  { //already assigend hence continue  if(InterValAssigned[interval]!=-1) continue;    UserAssigned[userId]=interval;  InterValAssigned[interval]=userId;      assignInterVals(userId+1,noUser);  //backtracking  UserAssigned[userId]=-1;  InterValAssigned[interval]=-1;    }      }  bool operator<(vector<pair<int,int> >& p, vector<pair<int,int> > & q)  {  return (p.size()<q.size());  }  void PrintSolution()  { //removing duplicate Solutions  auto iterAtor=unique(Solution.begin(),Solution.end());  Solution.resize(iterAtor-Solution.begin());  //Printing the Optimal Solutions    for(int ithSolution=0;ithSolution <min(4,(int)Solution.size());ithSolution++)  {  vector<pair<int,int>> oneSolution =Solution[ithSolution];  sort(oneSolution.begin(),oneSolution.begin());    for(int user=0;user<oneSolution.size();user++)  {  cout<<oneSolution[user].first<<"th user is assigned "<<oneSolution[user].second<<"th interval"<<endl;  }  cout<<endl;  }  }  int main()  {      time\_t start, end;  time(&start);  int noUser;  //enter the number of users  cin >> noUser;    int noInterval;  //enter the number of intervals  cin >> noInterval;    for(int user=1;user<=noUser;user++)  {    //enter the number of favaroble intervals of ith user  int noFeasibleIntervals; cin >> noFeasibleIntervals;  while(noFeasibleIntervals--)  {  //enter the interval for ith user  int ithFavourableInterval;  cin >> ithFavourableInterval;  FeasibleInterVals[user].push\_back(ithFavourableInterval);  }  }    assignInterVals(1,noUser);    PrintSolution();    time(&end);    double time\_taken = double(end - start);  cout << "Time taken by program is : " << fixed << time\_taken << setprecision(5);  cout << " sec " << endl;    return 0;  } |

Tracing the Algorithm:-

Consider the simple case

A-I1,I2

B-I1,I2

First A wil be left unassigned

->B is left unassigned and obtained solution would be empty.

->B will be assigned I1 and solution would be (B-I1).

-> B will be assigned I2 and solution would be (B-I2).

Now A will be Assigned I1

->B will left unassigned and obtained solution is (A-I1).

->B will assigned I2 and solution obtained is (A-I1,B-I2).

Now A will be assigned to I2

->B will be left unassigned and obtained solution is (A-I2).

-> B will be assigned to I1 and the obtained solution is (A-I2,B-I1).

Last step is to print the top 4 solution based on solution size.

Result :-

(A-I1,B-I2) (A-I2,B-I1) (A-I1) (A-I2)

Efficiency of Algorithm:-

Consider the worst case

Nouser=Nointervals=n.

Every interval is favourable for every User

Time Complexity:-

O(n!)

Space Complexity

O(n!)

Pros of Algorithm:-

1) Explores every possible assignement.

2) Best results for nouser<=4 and noInterval<=4.

Cons of Algorithm:-

1) Time complexity is very poor and for larger inputs very slow solutions.

2)Solution won’t work nouser>9 as n! Will cross the range of array and run time error will generated.