```
#!/usr/bin/env python
from __future__ import division
from gurobipy import *
from collections import defaultdict
import numpy as np
import argparse


"""
=============================================================================
========
  LP Approach
=============================================================================
========
"""

def solve(P, M, N, C, items, constraints, filename):
    """
    LP Solver
    Return: a list of strings, corresponding to item names.
    """
    Problem = False
    resale, cost, weight = [], [], []
    name = []

    print "Constructing Data for {}".format(filename)
    class_bin = defaultdict(list)
    class_num = set()
    for i in range(N):
        item = items[i]
        class_bin[item[1]].append(i)
        class_num.add(item[1])
        name.append(item[0])
        weight.append(item[2])
        cost.append(item[3])
        resale.append(item[4])
    name = np.array(name)
    w = np.array(weight)
    k = np.array(cost)
    r = np.array(resale)
    index = range(N)
    weight_dict = dict(zip(index, w))
    cost_dict = dict(zip(index, k))
    resale_dict = dict(zip(index, r))
    class_num = list(class_num)


    # try:
```

```
    #     print "try to read from files"
    #     model = read('problem_log/'+filename+".mps")
    #     model.read('problem_log/'+filename+".mst")
    # except:
    # print "Build up models"
    # print "failed to read from file, generating model"
    model = Model("Resale_Solver")
    x = model.addVars(index, vtype=GRB.BINARY, name='list')
    rep = model.addVars(class_num, vtype=GRB.INTEGER, name='indicator')
    model.ModelSense = GRB.MINIMIZE
    model.setObjective(x.prod(resale_dict) + M - x.prod(cost_dict))
    model.addConstr(x.prod(cost_dict) <= M, name='cost')
    model.addConstr(x.prod(weight_dict) <= P, name='weight')

    for clas in class_num:
        for item in class_bin[clas]:
            model.addConstr(rep[clas] >= x[item], name='rep({},
{})'.format(clas, item))

    for i in range(C):
        constr, expr = constraints[i], 0
        empty_class = True
        for clas in constr:
            if clas in class_num:
                empty_class = False
                expr += rep[clas]
        if not empty_class:
            model.addConstr(expr <= 1, name="constr_{}".format(i))

    print "finished building model, solving..."
    # Read in the parameter
    model.read("tune_param.prm")
    model.update()
    model.optimize()
    print "Done Solving, checking solutions"
    shop_list = np.array([name[i] for i in index if x[i].X > 0.9])
    # Status checking
    status = model.Status
    if status == GRB.Status.INF_OR_UNBD or \
       status == GRB.Status.INFEASIBLE  or \
       status == GRB.Status.UNBOUNDED:
        print('The model cannot be solved because it is infeasible or
unbounded')
        Problem = True
        # sys.exit(1)

    if status != GRB.Status.OPTIMAL:
        print('Optimization was stopped with status ' + str(status))
        Problem = True
```

```
        # sys.exit(1)
    write_output('instance_output2/' + filename + '.out', shop_list)
    model.write('problem_log/'+filename+".mps")
    model.write('problem_log/'+filename+".sol")
    model.write('problem_log/'+filename+".mst")

    return shop_list, Problem



def tune(model):
    print "Tuning Parameter..."
    model.tune()
    for i in range(model.tuneResultCount):
        model.getTuneResult(i)
        model.write('tune'+str(i)+'.prm')

"""
================================================================================
========
  No need to change any code below this line.
================================================================================
========
"""

def read_input(filename):
    """
    P: float
    M: float
    N: integer
    C: integer
    items: list of tuples
    constraints: list of sets
    """
    with open(filename) as f:
        P = float(f.readline())
        M = float(f.readline())
        N = int(f.readline())
        C = int(f.readline())
        items = []
        constraints = []
        for i in range(N):
            name, cls, weight, cost, val = f.readline().split(";")
            items.append((name, int(cls), float(weight), float(cost),
float(val)))
        for i in range(C):
            constraint = set(eval(f.readline()))
            constraints.append(constraint)
    return P, M, N, C, items, constraints
```

```
def write_output(filename, items_chosen):
    with open(filename, "w") as f:
        for i in items_chosen:
            f.write("{0}\n".format(i))

if __name__ == "__main__":
    #
    # parser = argparse.ArgumentParser(description="PickItems solver.")
    # parser.add_argument("input_file", type=str, help="____.in")
    # parser.add_argument("output_file", type=str, help="____.out")
    # args = parser.parse_args()

    print "Loading Input Files"
    problems = []
    for fi in range(21):

        input_file, output_file =
'project_instances_extracredit/problem{}.in'.format(fi+1),
'EC/problem{}.out'.format(fi+1)
        P, M, N, C, items, constraints = read_input(input_file)

        print "Start Solving..."
        items_chosen, problem = solve(P, M, N, C, items, constraints,
"problem{}".format(fi+1))
        problems.append(problem)
        print "Finished Solving, Write to file."
        write_output(output_file, items_chosen)
```