Name: Jamia Begum

NIU:1676891

Exercise-1: Food manufacturing planning problem1

CPLEX Model:

```
/*********************************************
 * OPL 22.1.0.0 Model
 * Author: jamia
 * Creation Date: 29 Nov 2022 at 02:00:47
 *********************************************/
  // raw oil elemement declaration
{string} VegRaw = ...;
{string} NonVegRaw = ...;
{string} Raw = VegRaw union NonVegRaw;


// final month of planning horizon
int NbMonths   = ...;
// range representing the planning horizon
range Months = 1..NbMonths;


// Matrix respresenting the buying cost
// for each month (rows) and each raw material (column)
float CostRaw[Raw][Months] = ...;

// other constant attributes
int ProfitProd = ...;
int CostStore = ...;
int MaxVeg = ...;
int MaxOil = ...;
int InitialStock = ...;
int FinalStock = ...;
int MaxStore = ...;
int MinUse = ... ;

// hardness index of raw oils
float HardRaw[Raw] = ...;
float MinHard = ...;
float MaxHard = ...;

// Decision Variables
dvar float+ Produce[Months]; // production amount
dvar float+ Use[Months][Raw]; // used raw oils
dvar float+ Buy[Months][Raw]; // row oil procurement
dvar float Store[0..NbMonths][Raw] in 0..1000; // storing policy


 // Generic formulation of the objective function terms
```

```
dexpr float Profit = sum(j in Months) Produce[j] * ProfitProd; // formulate
here profit expression
dexpr float Cost = sum(j in Months) sum(k in Raw) Buy[j][k]*CostRaw[k][j]+
                   sum(j in Months)sum(k in Raw) Store[j][k]*CostStore; //
formulate here supply ad store cost expression

maximize Profit - Cost;
subject to {

      // initial and final stock constraints
        forall(j in Raw)
          ct1:
          Store[0][j]==500;
         forall(j in Raw)
           ct2:
        Store[NbMonths][j]==500;
      // vegetable production capacity constraint
      forall(j in Months)
             ct3:
             sum(k in VegRaw) Use[j][k]<= MaxVeg;

      // non vegetable production capacity constraint
      forall(j in Months)
             ct4:
             sum(k in NonVegRaw) Use[j][k]<= MaxOil;

      // quality estipulation constraint
             forall(j in Months)
               ct5:
               sum(k in Raw) Use[j][k]*HardRaw[k]>=  MinHard*Produce[j];
              forall(j in Months)
                ct6:
                sum(k in Raw) Use[j][k]*HardRaw[k] <=  MaxHard*Produce[j];


      // Material balance Constraint (all what is used is mixed in the
product)
        forall(j in Months)
          ct7:
          sum(k in Raw) Use[j][k] == Produce[j];

      // Material balance Constraint  (relationship Stock-Supply-Use)
           forall(j in Months,k in Raw)
          ct8:
           Store[j-1][k]+ Buy[j][k] == Use[j][k]+Store[j][k];
         }

// Expected result : 107843
```

CPLEX Data:

```
VegRaw = {Veg1, Veg2};
NonVegRaw = {Oil1, Oil2, Oil3};
NbMonths = 6;
CostRaw = [[110 130 110 120 100 90]
           [120 130 140  110 120 100]
           [130 110 130 120  150 140]
           [110 90 100 120 110 80]
           [115 115 95 125 105 135]
            ];
ProfitProd = 150;
CostStore = 5;
MaxVeg = 200;
MaxOil = 250;
InitialStock = 500;
FinalStock = 500;
MaxStore = 1000;
MinUse = 20 ;
HardRaw = [8.8 6.1 2 4.2 5];
MinHard = 3;
MaxHard = 6;
```

Solution:

```
// solution (optimal) with objective 107842.592592593

Produce = [450
         450 450 450 450 450];
Buy = [[0 0 0 0 0]
             [0 0 0 750 0]
             [0 0 0 0 0]
             [0 0 0 0 0]
             [0 0 0 0 0]
             [659.26 540.74 0 750 0]];
Store = [[500 500 500 500 500]
             [477.78 322.22 500 250 500]
             [477.78 122.22 500 750 500]
             [318.52 81.481 500 500 500]
             [159.26 40.741 500 250 500]
             [0 0 500 0 500]
             [500 500 500 500 500]];
Use = [[22.222 177.78 0 250 0]
             [0 200 0 250 0]
             [159.26 40.741 0 250 0]
             [159.26 40.741 0 250 0]
             [159.26 40.741 0 250 0]
             [159.26 40.741 0 250 0]];
```

Exercise-2: Food manufacturing planning problem2

CPLEX MODEL:

Extra constraints:

```
//The food may never be made up of more than three oils in any month
       forall(j in Months)
```

```
               ct9:
               sum(k in Raw) (Use[j][k] >= MinUse) <=3;

    //oil use constraints (either use 0 unit or more than 20)
            forall(j in Months,k in Raw)
          ct10:
           (Use[j][k]==0) || (Use[j][k]) >= MinUse;

 //If either of VEG 1 or VEG 2 are used in a month then OIL 3 must also be
used
        forall(j in Months)
          ct11:
(Use [j]["Veg1"]>=MinUse) || (Use[j]["Veg2"] >= MinUse) =>
Use[j]["Oil3"] >= MinUse ;
```

Solution:

```
// solution (optimal) with objective 100278.703703704

Produce = [450
         450 450 405 405 450];
Buy = [[0 0 0 0 0]
             [0 0 0 190 0]
             [0 0 0 0 580]
             [0 0 0 0 0]
             [0 0 0 0 0]
             [480.37 629.63 0 730 0]];
Store = [[500 500 500 500 500]
             [414.81 385.19 500 500 250]
             [329.63 270.37 500 690 0]
             [329.63 70.37 500 460 560]
             [174.63 70.37 500 230 540]
             [19.63 70.37 500 0 520]
             [500 500 500 500 500]];
Use = [[85.185 114.81 0 0 250]
             [85.185 114.81 0 0 250]
             [0 200 0 230 20]
             [155 0 0 230 20]
             [155 0 0 230 20]
             [0 200 0 230 20]];
```

Exercise-3:Refinery planning problem

CPLEX MODEL:

```
{string} Crude = ...;
{string} Naptha = ...;
{string} Resid = ...;
{string} Oil = ...;
{string} ReformProd = ...;
{string} CrackProd = ...;
```

```
{string} Petrol = ...;
{string} Fuel = ...;
{string} Lube = ...;

float DistillNaptha[Crude][Naptha] = ...;
float DistillOil[Crude][Oil] = ...;
float DistillResid[Crude][Resid] = ...;

float ResidProcess[Resid][Lube] = ...;
float ReformProcess[Naptha][ReformProd] = ...;
float CrackProcess[Oil][CrackProd] = ...;

float VaporOil[Oil] = ...;
float VaporResid[Resid] = ...;
float VaporCrkOil = ...;
float LimVaporJF = ...;

float LimCrude[Crude] = ...;
float LimDistill = ...;
float LimReform = ...;
float LimCrack = ...;
float LoLube[Lube] = ...;
float UpLube[Lube] = ...;

float OctaneNaptha[Naptha] = ...;
float OctaneReform[ReformProd] = ...;
float OctaneCG = ...;
float ReqOctane[Petrol] = ...;
float ReqRatioPetrol = ...;

float ReqOilFO[Oil] = ...;
float ReqCrkFO = ...;
float ReqResidFO[Resid] = ...;

float ProfitPetrol[Petrol] = ...;
float ProfitFuel[Fuel] = ...;
float ProfitLube[Lube] = ...;


/* What to produce and use dvars */

// Fuels (JF jet fuel/FO fuel oil)
dvar float+ Fpf[Fuel];
// Petrols (PMF premium motor fuel/RMF regular motor fuel)
dvar float+ Fpp[Petrol];
// LBO lube-oil
dvar float+ Fpl[l in Lube] in LoLube[l]..UpLube[l];
// Crude oil to use
dvar float+ Cr[c in Crude] in 0..LimCrude[c];
// Naphthas from distillation
dvar float+ Nap[Naptha];
// Naphthas for reforming
dvar float+ Napref[Naptha];
// Naphthas for blending
dvar float+ Napb[Naptha][Petrol];
// Reforming products for blending petrol
dvar float+ Refb[ReformProd][Petrol];
// Reform products
dvar float+ Ref[ReformProd];
```

```
// Oils from distillation
dvar float+ OilVar[Oil];
// Distilled oils for cracking
dvar float+ Oilcrk[Oil];
// Distilled oils for blending
dvar float+ Oilb[Oil][Fuel];
// Cracked products
dvar float+ Crk[CrackProd];
// Cracked gasoline for blending petrol
dvar float+ Crkg[Petrol];
// Cracked oild for blending
dvar float+ Crko[Fuel];
// Residuum from distillation
dvar float+ ResidVar[Resid];
// Residuum used for lube-oil
dvar float+ Residl[Resid];
// Residuum used for blending
dvar float+ Residbf[Resid][Fuel];

// Objective
dexpr float TotalProfitPetrol = sum(p in Petrol) ProfitPetrol[p]*Fpp[p];
dexpr float TotalProfitFuel = sum(f in Fuel) ProfitFuel[f]*Fpf[f];
dexpr float TotalProfitLube = sum(l in Lube) ProfitLube[l]*Fpl[l];

maximize TotalProfitPetrol + TotalProfitFuel + TotalProfitLube;

/* this is equivalent to

maximize sum(p in Petrol) ProfitPetrol[p]*Fpp[p] +
         sum(f in Fuel) ProfitFuel[f]*Fpf[f] +
         sum(l in Lube) ProfitLube[l]*Fpl[l];
*/


subject to {
  // Distillation capacity
  // Cr["CRA"] + Cr["CRB"] <= LimDistill;
  sum(c in Crude) Cr[c] <= LimDistill;

  // Reforming capacity
  // Napref["LN"] + Napref["MN"] + Napref["HN"] <= LimReform;
  sum(n in Naptha) Napref[n] <= LimReform;

  // Cracking capacity
  sum(o in Oil) Oilcrk[o] <= LimCrack;

  // Distillation products
  forall(n in Naptha)
    sum(c in Crude) Cr[c]*DistillNaptha[c][n] == Nap[n];
  forall(o in Oil)
    sum(c in Crude) Cr[c]*DistillOil[c][o] == OilVar[o];
  forall(r in Resid)
    sum(c in Crude) Cr[c]*DistillResid[c][r]== ResidVar[r];


  // Reformer products
forall(r in ReformProd)
  sum(n in Naptha) Napref[n]*ReformProcess[n][r]== Ref[r];
  // Cracking products
forall(c in CrackProd)
  sum(o in Oil) Oilcrk[o]*CrackProcess[o][c]== Crk[c];


  // Balance constraints on Napthas
forall(n in Naptha)
    Napref[n]+ sum(p in Petrol)Napb[n][p]==Nap[n];

  // Balance constraints on Oils
forall(o in Oil)
  Oilcrk[o] + sum(f in Fuel)Oilb[o][f]== OilVar[o];

  // Balance constraints on Residuums
forall(r in Resid)
   Residl[r] + sum(f in Fuel)Residbf[r][f]== ResidVar[r];


  // Balance constraint on Reformed products
forall(r in ReformProd)
   sum(p in Petrol)Refb[r][p] == Ref[r];
   // Balance constraint on crack products
   sum(p in Petrol)Crkg[p]==Crk["CG"];
   sum(f in Fuel)Crko[f]== Crk["CO"];

  // Balance constraints on Petrols
 forall(p in Petrol)
sum(n in Naptha)Napb[n][p]+  sum(r in ReformProd)Refb[r][p]+ Crkg[p] ==
Fpp[p];

  // Balance constraint on Fuels
   forall(f in Fuel)
sum(o in Oil)Oilb[o][f]+sum(r in Resid)Residbf[r][f]+Crko[f]== Fpf[f];

//Balance constraints for lube
forall(l in Lube)
  sum (r in Resid) Residl[r]*ResidProcess[r][l]== Fpl[l];

  // Fixed proportions required for Fuel Oil
  forall(o in Oil)
Oilb[o]["FO"]==ReqOilFO[o]*Fpf["FO"];
 Crko["FO"] == ReqCrkFO*Fpf["FO"];
 forall(r in Resid)
 Residbf[r]["FO"]==ReqResidFO[r]*Fpf["FO"];

  // Required ratio between petrols
  Fpp["PMF"]>=ReqRatioPetrol*Fpp["RMF"];

  // Qualities  Octane
forall(p in Petrol)
sum(n in Naptha) OctaneNaptha[n]*Napb[n][p]+
sum(r in ReformProd) OctaneReform[r]*Refb[r][p]+
OctaneCG*Crkg[p]>= ReqOctane[p]*Fpp[p];


  // Vapor Pressure constraint on Jet Fuel
sum(o in Oil) VaporOil[o]*Oilb[o]["JF"] +
sum(r in Resid) VaporResid[r]*Residbf[r]["JF"] +
VaporCrkOil*Crko["JF"] <= LimVaporJF*Fpf["JF"];


}
```

CPLEX Data:

```
Crude = {CRA, CRB};
Naptha = {LN, MN, HN};
Resid = {R};
Oil = {LO, HO};
ReformProd = {RG};
CrackProd = {CG, CO};
Petrol = {PMF, RMF};
Fuel = {JF, FO};
Lube = {LBO};

DistillNaptha = [[0.10 0.20 0.20]
                 [0.15 0.25 0.18]];
DistillOil = [[0.12 0.20]
              [0.08 0.19]];
DistillResid = [[0.13]
                [0.12]];

ReformProcess = [[0.60]
                 [0.52]
                 [0.45]];

CrackProcess = [[0.28 0.68]
                [0.20 0.75]];
ResidProcess = [[0.50]];

OctaneNaptha = [90 80 70];
OctaneReform = [115];
OctaneCG = 105;
ReqOctane = [94 84];
ReqRatioPetrol = 0.40;

ReqOilFO = [.55 .17];
ReqCrkFO = 0.22;
ReqResidFO = [0.06];


VaporOil = [1.0 0.6];
VaporResid = [0.05];
VaporCrkOil = 1.5;
LimVaporJF = 1.0;

LimCrude = [20000 30000];
LimDistill = 45000;
LimReform = 10000;
LimCrack = 8000;
LoLube = [500];
UpLube = [1000];
ProfitPetrol = [7 6];
ProfitFuel   = [4 3.5];
ProfitLube   = [1.5];
```

Solution:

```
// solution (optimal) with objective 211365.134768933

Fpp = [6817.8
         17044];
Fpf = [15156 0];
Fpl = [500];
Cr = [15000 30000];
Napref = [0 0 5406.9];
Oilcrk = [4200 3800];
Nap = [6000 10500 8400];
OilVar = [4200 8700];
ResidVar = [5550];
Ref = [2433.1];
Crk = [1936 5706];
Napb = [[5726.9 273.07]
             [0 10500]
             [0 2993.1]];
Oilb = [[0 0]
             [4900 0]];
Residl = [1000];
Residbf = [[4550 0]];
Refb = [[1090.8 1342.2]];
Crkg = [0 1936];
Crko = [5706 0];
```