Integer Programming (IP)

Problem

The CitruSun Corporation ships frozen orange juice concentrate from processing plants in Eustis and Clermont to distributors in Miami, Orlando, and Tallahassee. Each plant can produce 20 tons of concentrate each week.

The company has just received orders of 10 tons from Miami for the coming week, 15 tons for Orlando, and 10 tons for Tallahassee. The cost per ton for supplying each of the distributors from each of the processing plants is shown in the following table:

	Miami	**Orlando**	**Tallahassee**
Eustis	\$260	\$220	\$290
Clermont	\$220	\$240	\$320

The company wants to determine the **minimum** costly plan for filling their orders for the coming week.

Formulate the Model

In this problem we are trying to determine how many tons of OJ to ship to each distribution center and from which plants to minimize cost. This means that every cell on the above table represents a decision variable:

	Miami	**Orlando**	**Tallahassee**
Eustis	x_1	x_2	x_3
Clermont	x_4	x_5	x_6

Aligning this with the cost table above, we can formulate our objective function and constraints:

Minimize

$260x_1 + 220x_2 + 290x_3 + 220x_4 + 240x_5 + 320x_6$

Subject to

Each plant can produce 20 tons of concentrate each week

$x_1 + x_2 + x_3 \leq 20$

$x_4 + x_5 + x_6 \leq 20$

10 tons for Miami: $x_1 + x_4 = 10$

15 tons for Orlando: $x_2 + x_5 = 15$

10 tons for Tallahassee: $x_3 + x_6 = 10$

non-negativity: $x_1,x_2,x_3,x_4,x_5,x_6 \geq 0$

integer constraint: $x_1,x_2,x_3,x_4,x_5,x_6 \in integer$

Model in Python

This process doesn't differ much from our previous modeling, with the exception of specifying the `cat` of our `LpVariable` as `Integer`

In [2]:
```
from pulp import *
```

Define our variables specified above. Next add the constraints we defined above except the non-negativity and integer constraints. These are captured in the `lowBound` and `cat`, respectively.

In [5]:
```
intprob = LpProblem("OJ_Distribution", LpMinimize)

x1 = LpVariable('EustisToMiami', lowBound = 0, cat = 'Integer')
x2 = LpVariable('EustisToOrlando', lowBound = 0, cat = 'Integer')
x3 = LpVariable('EustisToTallahassee', lowBound = 0, cat = 'Integer')
x4 = LpVariable('ClermontToMiami', lowBound = 0, cat = 'Integer')
x5 = LpVariable('ClermontToOrlando', lowBound = 0, cat = 'Integer')
x6 = LpVariable('ClermontToTallahassee', lowBound = 0, cat = 'Integer')
```

In [8]:
```
# Objective function
intprob += 260*x1 + 220*x2 + 290*x3 + 220*x4 + 240*x5 + 320*x6, "Obj"

# Plant production constraint
intprob += x1 + x2 + x3 <= 20, "Eustis constraint"
intprob += x4 + x5 + x6 <= 20, "Clermont constraint"

# Order constraints
intprob += x1 + x4 == 10, "Miami order"
intprob += x2 + x5 == 15, "Orlando order"
intprob += x3 + x6 == 10, "Tallahassee order"
```

In [9]:
```
print(intprob)
```

```
OJ_Distribution:
MINIMIZE
220*ClermontToMiami + 240*ClermontToOrlando + 320*ClermontToTallahassee + 2
60*EustisToMiami + 220*EustisToOrlando + 290*EustisToTallahassee + 0
SUBJECT TO
Eustis_constraint: EustisToMiami + EustisToOrlando + EustisToTallahassee <=
20

Clermont_constraint: ClermontToMiami + ClermontToOrlando
 + ClermontToTallahassee <= 20

Miami_order: ClermontToMiami + EustisToMiami = 10

Orlando_order: ClermontToOrlando + EustisToOrlando = 15

Tallahassee_order: ClermontToTallahassee + EustisToTallahassee = 10

VARIABLES
0 <= ClermontToMiami Integer
0 <= ClermontToOrlando Integer
0 <= ClermontToTallahassee Integer
0 <= EustisToMiami Integer
0 <= EustisToOrlando Integer
0 <= EustisToTallahassee Integer
```

Finally, we can solve and print the optimal variables and value.

In [ ]:
```
intprob.solve()
print(LpStatus[intprob.status])
```

In [11]:
```
for variable in intprob.variables():
  print("{} = {}".format(variable.name, variable.varValue))

print("Optimial Function Value = {}".format(value(intprob.objective)))
```

```
ClermontToMiami = 10.0
ClermontToOrlando = 5.0
ClermontToTallahassee = 0.0
EustisToMiami = 0.0
EustisToOrlando = 10.0
EustisToTallahassee = 10.0
Optimial Function Value = 8500.0
```

IP Interpretation

Our optimal minimum cost is **$8500**

Which can be achieved by planning the orders as follows:

	Miami	Orlando	Tallahassee
Eustis	0	10	10
Clermont	10	5	0

Binary Integer Programming (BIP)

Problem

A young couple, Eve and Steven, want to divide their main household tasks (shopping, cooking, dishwashing, and laundering) between them so that each has two tasks, but the total time they spend on household duties is kept to a minimum.

Their efficiencies on these tasks differ, where the following table gives the time each would need to perform the task:

	Shopping	Cooking	Dishwashing	Laundry
Eve	4.5 hours	7.5 hours	3.5 hours	3.0 hours
Steven	5.0 hours	7.2 hours	4.5 hours	3.2 hours

Formulate the Model

Again, we can identify our decision variables according to the table:

	Shopping	Cooking	Dishwashing	Laundry
Eve	x_1	x_2	x_3	x_4
Steven	x_5	x_6	x_7	x_8

Using this and the information in the prompt, we will build our objective function and constraints.

Minimize

$4.5x_1 + 7.5x_2 + 3.5x_3 + 3x_4 + 5x_5 + 7.2x_6 + 4.5x_7 + 3.2x_8$

s.t.

Each has two tasks

$x_1 + x_2 + x_3 + x_4 = 2$

$x_5 + x_6 + x_7 + x_8 = 2$

Each task assigned to one person

$x_1 + x_5 == 1$

$x_2 + x_6 == 1$

$x_3 + x_7 == 1$

$x_4 + x_8 == 1$

non-negativity

$x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8 \geq 0$

integer constraint

$x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8 \in integer$

BIP in Python

For this model, the major change is ensuring that we set our upper-bound to 1 for each decision variable to indicate that the tasks are assigned to a single person. Otherwise, we can model this very similarly to the simple integer programming problem above.

In [19]:
```
bipprob = LpProblem("EvenStevens", LpMinimize)

x1 = LpVariable('EveShopping', lowBound = 0, upBound = 1, cat = 'Integer')
x2 = LpVariable('EveCooking', lowBound = 0, upBound = 1, cat = 'Integer')
x3 = LpVariable('EveDishwashing', lowBound = 0, upBound = 1, cat = 'Integer'
x4 = LpVariable('EveLaundry', lowBound = 0, upBound = 1, cat = 'Integer')
x5 = LpVariable('StevenShopping', lowBound = 0, upBound = 1, cat = 'Integer'
x6 = LpVariable('StevenCooking', lowBound = 0, upBound = 1, cat = 'Integer')
x7 = LpVariable('StevenDishwashing', lowBound = 0, upBound = 1, cat = 'Integ
x8 = LpVariable('StevenLaundry', lowBound = 0, upBound = 1, cat = 'Integer')
```

In [20]:
```
# Objective function
bipprob += 4.5*x1 + 7.5*x2 + 3.5*x3 + 3*x4 + 5*x5 + 7.2*x6 + 4.5*x7 + 3.2*x8

# Plant production constraint
bipprob += x1 + x2 + x3 + x4 == 2, "Eve Constraint"
bipprob += x5 + x6 + x7 + x8 == 2, "Steven constraint"

# Task constraints
bipprob += x1 + x5 == 1, "Shopping constraint"
bipprob += x2 + x6 == 1, "Cooking constraint"
bipprob += x3 + x7 == 1, "Dishwashing constraint"
bipprob += x4 + x8 == 1, "Laundry constraint"
```

In [ ]:
```
print(bipprob)
```

In [ ]:
```
bipprob.solve()
```

In [23]:
```
for variable in bipprob.variables():
  print("{} = {}".format(variable.name, variable.varValue))

print("Optimial Function Value = {}".format(value(bipprob.objective)))
```

```
EveCooking = 0.0
EveDishwashing = 1.0
EveLaundry = 0.0
EveShopping = 1.0
StevenCooking = 1.0
StevenDishwashing = 0.0
StevenLaundry = 1.0
StevenShopping = 0.0
Optimial Function Value = 18.4
```

BIP Interpretation

The minimized optimal amount of work between the couple is **18.4 hours**.

To achieve this, Eve should do the dishwashing and shopping while Steven should do the cooking and the laundry.

Mixed-Integer Programming (MIP)

Problem

The XYZ Company has developed two new products to potentially add to their product line before the next holiday season. The cost of setting up production facilities for product 1 is \$50,000, and for product 2, it is $70,000. Once the initial costs are recovered, each unit of product 1 generates a profit of \$10, and each unit of product 2 generates a profit of \$15.

The company has two factories that can produce these products, but **only one factory can be used** to avoid doubling the setup costs. The choice of factory will be based on **maximizing profit.** If both new products are produced, they will be manufactured in the same factory for administrative reasons. In factory 1, product 1 can be produced at a rate of 50 per hour (or 1/50 hours per unit), and product 2 can be produced at a rate of 40 per hour (or 1/40 hours per unit). In factory 2, product 1 can be produced at a rate of 40 per hour (or 1/40 hours per unit), and product 2 can be produced at a rate of 25 per hour.

Before Christmas, there are 500 hours of production time available in factory 1 and 700 hours of production time available in factory 2. It is not known whether these products will be continued after the holiday season. Therefore, **the objective is to determine the optimal number of units of each product to produce** before Christmas to maximize the total profit.

Note that we express the production rates as hours-per-unit rather than units-per-hour.

	Product 1	**Product 2**	**Available Hours**
Factory 1	$\frac{1}{50}$	$\frac{1}{40}$	500
Factory 2	$\frac{1}{40}$	$\frac{1}{25}$	700
Startup Cost	50000	70000	
Profit	10	15	

MIP Modeling

For this objective function we have a new challenge - each product has a startup cost. These costs need to be subtracted, but they also need to be attached to their own y decision variables so they are only factored in if the product is chosen in our optimal solution.

Maximize

$10*x_1 + 15*x_2 – 50000*y_1 – 70000*y_2$

The need to choose a single factory of the two also presents a new challenge. This requires a neat track involving an additional variable to represent the binary decision, y_3, and another to represent a large number that we refer to as Big M

For each factory, we will add both variables to the right-hand side of their respective constraints. The trick is that in one we multiply Big M by y_3, and in the other we multiply it by $(1-y_3)$.

When y_3 is determined by the solution to be 1 or 0, plugging it in will make the RHS of one of these two constraints large enough to be unbounded for the scope of our solution. This indicates that the other constraint is the one affecting our solution, which makes it the optimal factory decision.

Subject to

Factory constraints

$\frac{1}{50}x_1 + \frac{1}{40}x_2 <= 500 + My_3$

$\frac{1}{40}x_1 + \frac{1}{25}x_2 <= 700 + M(1-y_3)$

Product constraints

$x_1 <= My_1$

$x_2 <= My_2$

Non-negativity constraint: $x_1,x_2,y_1,y_2,y_3 \geq 0$

Integer constraint: $y_1,y_2,y_3 \in integer$

MIP in Python

In addition to our y_3 variable for determining the factory, we also have additional y variables that indicate whether each product is "used" in the final solution. This allows us to implement the correct amount of fixed cost in the objective function.

The only additional change is defining a value for `bigM`. For the purpose and scale of this problem, a value of 1,000,000 is large that enough that it would nullify any constraint if it isn't mathematically removed.

In [7]:
```
from pulp import *
```

In [11]:
```
mipprob = LpProblem("MIP_Products", LpMaximize)

# Decision Variables
x1 = LpVariable('Product1_x', lowBound = 0)
x2 = LpVariable('Product2_x', lowBound = 0)

# Mixed Variables
y1 = LpVariable('Product1isUsed', lowBound = 0, upBound = 1, cat = "Integer"
y2 = LpVariable('Product2isUsed', lowBound = 0, upBound = 1, cat = "Integer"
y3 = LpVariable('FactoryBiVariable', lowBound = 0, upBound = 1, cat = "Integ

# Objective Function
mipprob += 10*x1 + 15*x2 - 50000*y1 - 70000*y2, "Obj"

# Big M
bigM = 1000000
```

In [12]:
```
# Constraints
mipprob += (1/50)*x1 + (1/40)*x2 <= 500 + bigM*y3
mipprob += (1/40)*x1 + (1/25)*x2 <= 700 + bigM*(1-y3)
mipprob += x1 <= bigM*y1
mipprob += x2 <= bigM*y2
```

In [13]:
```
print(mipprob)
```

```
MIP_Products:
MAXIMIZE
10*Product1_x + -50000*Product1isUsed + 15*Product2_x + -70000*Product2isUs
ed + 0
SUBJECT TO
_C1: - 1000000 FactoryBiVariable + 0.02 Product1_x + 0.025 Product2_x <= 50
0

_C2: 1000000 FactoryBiVariable + 0.025 Product1_x + 0.04 Product2_x <= 1000
700

_C3: Product1_x - 1000000 Product1isUsed <= 0

_C4: Product2_x - 1000000 Product2isUsed <= 0

_C5: Product1isUsed + Product2isUsed <= 2

VARIABLES
0 <= FactoryBiVariable <= 1 Integer
Product1_x Continuous
0 <= Product1isUsed <= 1 Integer
Product2_x Continuous
0 <= Product2isUsed <= 1 Integer
```

In [ ]:
```
mipprob.solve()
```

In [16]:
```
for variable in mipprob.variables():
  print("{} = {}".format(variable.name, variable.varValue))

print("Optimial Function Value = {}".format(value(mipprob.objective)))
```

```
FactoryBiVariable = 1.0
Product1_x = 28000.0
Product1isUsed = 1.0
Product2_x = 0.0
Product2isUsed = 0.0
Optimial Function Value = 230000.0
```

MIP Interpretation

These results give us two obvious pieces of information about our optimal solution:

1. We should produce 28000 units of Product 1 and none of Product 2
2. This results in an optimum profit of $230,000

We can confirm this relationship by multiplying those 28000 units by our profit-per-unit of \$10 and subtracting the \$50,000 startup cost, giving us \$230,000 in profit.

Less obvious is that our binary variable (y_3) is equal to 1. To interpret this, let's review our constraints using this variable:

$\frac{1}{50}x_1 + \frac{1}{40}x_2 <= 500 + My_3$

$\frac{1}{40}x_1 + \frac{1}{25}x_2 <= 700 + M(1-y_3)$

If we substitute 1 for y_3, this would give us an extremely large number on the right-hand side of the first equation representing Factory 1. This represents a virtually unbounded constraint as opposed to the second equation, where the 1 results in the Big M being multiplied by 0 and our original RHS of 700 hours remains a realistic boundary.

In addition to the product decision variables above, this tells us that Factory 2 should be used for all production.