2K20-MC-116 SAKSHAM HOODA

```
!pip install pulp
```

```
Collecting pulp
  Downloading PuLP-2.7.0-py3-none-any.whl (14.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.3/14.3 MB 73.7 MB/s eta 0:00:00
Installing collected packages: pulp
Successfully installed pulp-2.7.0
```

> A company uses four special tank trucks to deliver four different gasoline products to customers. Each tank has five compartments with different capacities: 500, 750, 1200, 1500, and 1750 gallons. The daily demands for the four products are estimated at 10, 15, 12, and 8 thousand gallons. Any quantities that cannot be delivered by the company's four trucks must be subcontracted at the additional costs of 5, 12, 8, and 10 cents per gallon for products 1, 2, 3, and 4, respectively. Develop the optimal daily loading schedule for the four trucks that will minimize the additional cost of subcontracting.

```
import pulp


model = pulp.LpProblem("A3_Q3", pulp.LpMinimize)


# Decision Variables
S = pulp.LpVariable.dicts("Subcontracted", [1,2,3,4,5], 0)
X = pulp.LpVariable.dicts("Gallons_filled", [(i,j,k) for i in range(1,5) for j in range(1,6) for k in range(1,5)],
Y = pulp.LpVariable.dicts("Binary_Y", [(i,j,k) for i in range(1,5) for j in range(1,6) for k in range(1,5)], 0, 1,
B = pulp.LpVariable.dicts("Binary_B", [(i,k) for i in range(1,3) for k in range(1,5)], 0, 1, pulp.LpBinary)



#objective 5*S[1] + 12*S[2] + 8*S[3] + 10*S[4]
model += 5*S[1] + 12*S[2] + 8*S[3] + 10*S[4], "Total_Subcontracting_Cost"


# Demand Constraints
model += pulp.lpSum([X[(1, j, k)] for j in range(1, 6) for k in range(1, 5)]) + S[1] == 10000
model += pulp.lpSum([X[(2,j,k)] for j in range(1,6) for k in range(1,5)]) + S[2] == 15000
model += pulp.lpSum([X[(3,j,k)] for j in range(1,6) for k in range(1,5)]) + S[3] == 12000
model += pulp.lpSum([X[(4,j,k)] for j in range(1,6) for k in range(1,5)]) + S[4] == 8000

# Capacity and linking constraints
capacities = [500, 750, 1200, 1500, 1750]
for i in range(1,5):
    for k in range(1,5):
        for j, cap in enumerate(capacities, start=1):
            model += X[(i,j,k)] - cap*Y[(i,j,k)] <= 0

for j in range(1,6):
    for k in range(1,5):
        model += pulp.lpSum([Y[(i,j,k)] for i in range(1,5)]) <= 1

# Part-II Constraints
for k in range(1,5):
    model += B[(1,k)] + B[(2,k)] <= 1

for i in range(1,3):
    for k in range(1,5):
        model += pulp.lpSum([Y[(i,j,k)] for j in range(1,6)]) - 5*B[(i,k)] <= 0
        model += B[(i,k)] - pulp.lpSum([Y[(i,j,k)] for j in range(1,6)]) <= 0



model.solve()
```

```
1
```

```
print(f"Status: {pulp.LpStatus[model.status]}")
print(f"Total Subcontracting Cost: ${pulp.value(model.objective)}")
```

```
Status: Optimal
Total Subcontracting Cost: $148000.00000000003
```

```
# Print the optimal values for the decision variables
print("Optimal values for Subcontracted Gallons (S):")
for i in range(1,6):
    print(f"S{i}: {S[i].value()}")
```

```
Optimal values for Subcontracted Gallons (S):
S1: 10000.0
S2: 1.7347672e-12
S3: 12000.0
S4: 200.0
S5: None
```

```
print("\nOptimal values for Gallons Filled (X):")
for i in range(1,5):
    for j in range(1,6):
        for k in range(1,5):
            print(f"X({i},{j},{k}): {X[(i,j,k)].value()}")
```

```
X(2,1,3): 500.0
X(2,1,4): 500.0
X(2,2,1): 0.0
X(2,2,2): 750.0
X(2,2,3): 0.0
X(2,2,4): 750.0
X(2,3,1): 0.0
X(2,3,2): 0.0
X(2,3,3): 0.0
X(2,3,4): 0.0
X(2,4,1): 1500.0
X(2,4,2): 1500.0
X(2,4,3): 0.0
X(2,4,4): 1500.0
```

```
X(4,1,1): 0.0
X(4,1,2): 0.0
X(4,1,3): 0.0
X(4,1,4): 0.0
X(4,2,1): 750.0
X(4,2,2): 0.0
X(4,2,3): 750.0
X(4,2,4): 0.0
X(4,3,1): 1200.0
X(4,3,2): 1200.0
X(4,3,3): 1200.0
X(4,3,4): 1200.0
X(4,4,1): 0.0
X(4,4,2): 0.0
X(4,4,3): 1500.0
X(4,4,4): 0.0
X(4,5,1): 0.0
X(4,5,2): 0.0
X(4,5,3): 0.0
X(4,5,4): 0.0
```

```
print("\nOptimal values for Binary Decision Y:")
for i in range(1,5):
    for j in range(1,6):
        for k in range(1,5):
            print(f"Y({i},{j},{k}): {Y[(i,j,k)].value()}")
```

```
Optimal values for Binary Decision Y:
Y(1,1,1): 0.0
Y(1,1,2): 0.0
Y(1,1,3): 0.0
Y(1,1,4): 0.0
Y(1,2,1): 0.0
Y(1,2,2): 0.0
Y(1,2,3): 0.0
Y(1,2,4): 0.0
Y(1,3,1): 0.0
Y(1,3,2): 0.0
Y(1,3,3): 0.0
Y(1,3,4): 0.0
Y(1,4,1): 0.0
Y(1,4,2): 0.0
Y(1,4,3): 0.0
Y(1,4,4): 0.0
Y(1,5,1): 0.0
Y(1,5,2): 0.0
Y(1,5,3): 0.0
Y(1,5,4): 0.0
Y(2,1,1): 1.0
Y(2,1,2): 1.0
Y(2,1,3): 1.0
Y(2,1,4): 1.0
Y(2,2,1): 0.0
Y(2,2,2): 1.0
Y(2,2,3): 0.0
Y(2,2,4): 1.0
Y(2,3,1): 0.0
Y(2,3,2): 0.0
Y(2,3,3): 0.0
Y(2,3,4): 0.0
Y(2,4,1): 1.0
Y(2,4,2): 1.0
Y(2,4,3): 0.0
Y(2,4,4): 1.0
Y(2,5,1): 1.0
Y(2,5,2): 1.0
Y(2,5,3): 1.0
Y(2,5,4): 1.0
Y(3,1,1): 0.0
Y(3,1,2): 0.0
```

```
Y(3,1,3): 0.0
Y(3,1,4): 0.0
Y(3,2,1): 0.0
Y(3,2,2): 0.0
Y(3,2,3): 0.0
Y(3,2,4): 0.0
Y(3,3,1): 0.0
Y(3,3,2): 0.0
Y(3,3,3): 0.0
Y(3,3,4): 0.0
Y(3,4,1): 0.0
Y(3,4,2): 0.0
Y(3,4,3): 0.0
Y(3,4,4): 0.0
```

```
print("\nOptimal values for Binary Decision B:")
for i in range(1,3):
    for k in range(1,5):
        print(f"B({i},{k}): {B[(i,k)].value()}")
```

```
Optimal values for Binary Decision B:
B(1,1): 0.0
B(1,2): 0.0
B(1,3): 0.0
B(1,4): 0.0
B(2,1): 1.0
B(2,2): 1.0
B(2,3): 1.0
B(2,4): 1.0
```