Introduction to Integer Programming Problem

Integer Programming (IP) is a type Linear Programming Problem of optimization where some or all of the decision variables are required to be integers. It is used in cases where the decision variables represent discrete items, such as people, machines, or tasks. Integer programming is a subset of linear programming (LP) but with the additional constraint that variables must take on integer values.

Types of Integer Programming:

1. **Pure Integer Programming:** All decision variables are required to be integers.
2. **Mixed Integer Programming (MIP):** Some decision variables are integers while others can be continuous.
3. **Binary Integer Programming:** Decision variables can only take on values of 0 or 1.

Case Study - 1 (Pure Integer Problem)

A hotel operating 24 hours a day requires a certain minimum number of waiters during different periods of the day. The objective is to determine the number of waiters to schedule for each period such that the total number of waiters is minimized while meeting the minimum requirement for each period.

Given Data

- **Period 1:** 7–11am, Minimum waiters: 6
- **Period 2:** 11am–3pm, Minimum waiters: 12
- **Period 3:** 3–7pm, Minimum waiters: 8
- **Period 4:** 7–11pm, Minimum waiters: 16
- **Period 5:** 11pm–3am, Minimum waiters: 5
- **Period 6:** 3–7am, Minimum waiters: 3

A waiter reports at the beginning of the period and works for 8 consecutive hours.

Formulation as an Integer Programming Problem

Decision Variables:

Let x_i (for $i=1,2,\ldots,6$) represent the number of waiters starting their shift at the beginning of period i.

Objective Function:

Minimize the total number of waiters: min $z = x_1+x_2+x_3+x_4+x_5+x_6$

Constraints:

1. Waiters needed in period 1 (7–11am): $x1+x6 \geq 6$
2. Waiters needed in period 2 (11am–3pm): $x1+x2 \geq 12$
3. Waiters needed in period 3 (3–7pm): $x2+x3 \geq 8$
4. Waiters needed in period 4 (7–11pm): $x3+x4 \geq 16$
5. Waiters needed in period 5 (11pm–3am): $x4+x5 \geq 5$
6. Waiters needed in period 6 (3–7am): $x5+x6 \geq 3$

Integer Constraints: $xi \geq 0$ and integer for $i=1,2,\ldots,6$

Explanation

The above formulation ensures that the minimum number of waiters required for each period is met while minimizing the total number of waiters scheduled. The decision variables xi represent the number of waiters starting their shifts at different periods. The objective function aims to minimize the total number of waiters. The constraints ensure that for each period, the number of waiters present meets or exceeds the required minimum.

In [1]:
```python
import pulp

# Define the problem
problem = pulp.LpProblem("Minimize_Number_of_Waiters", pulp.LpMinimize)

# Define decision variables
x1 = pulp.LpVariable('x1', lowBound=0, cat='Integer')
x2 = pulp.LpVariable('x2', lowBound=0, cat='Integer')
x3 = pulp.LpVariable('x3', lowBound=0, cat='Integer')
x4 = pulp.LpVariable('x4', lowBound=0, cat='Integer')
x5 = pulp.LpVariable('x5', lowBound=0, cat='Integer')
x6 = pulp.LpVariable('x6', lowBound=0, cat='Integer')

# Objective function: minimize the total number of waiters
problem += x1 + x2 + x3 + x4 + x5 + x6, "Total_Waiters"

# Constraints
problem += x1 + x6 >= 6, "Period_1_Constraint"
problem += x1 + x2 >= 12, "Period_2_Constraint"
problem += x2 + x3 >= 8, "Period_3_Constraint"
problem += x3 + x4 >= 16, "Period_4_Constraint"
problem += x4 + x5 >= 5, "Period_5_Constraint"
problem += x5 + x6 >= 3, "Period_6_Constraint"

# Solve the problem
problem.solve()

# Print the results
print("Status:", pulp.LpStatus[problem.status])
print("Optimal number of waiters to start at each period:")
print("x1 (7-11am):", pulp.value(x1))
print("x2 (11am-3pm):", pulp.value(x2))
print("x3 (3-7pm):", pulp.value(x3))
print("x4 (7-11pm):", pulp.value(x4))
print("x5 (11pm-3am):", pulp.value(x5))
print("x6 (3-7am):", pulp.value(x6))
print("Total number of waiters required:", pulp.value(problem.objective))
```

```
Status: Optimal
Optimal number of waiters to start at each period:
x1 (7-11am): 3.0
x2 (11am-3pm): 9.0
x3 (3-7pm): 0.0
x4 (7-11pm): 16.0
x5 (11pm-3am): 0.0
x6 (3-7am): 3.0
Total number of waiters required: 31.0
```

Explanation of the Code

1. **Problem Definition:**
    - We define the problem as a minimization problem using pulp.LpProblem.
2. **Decision Variables:**
    - We define the decision variables x1 to x6, each representing the number of waiters starting their shift at the beginning of each period. These variables are non-negative integers.
3. **Objective Function:**
    - The objective function is to minimize the total number of waiters, represented by the sum of all decision variables.
4. **Constraints:**
    - We add the constraints for each period to ensure the minimum number of waiters required in each period is met.
5. **Solving the Problem:**
    - We solve the problem using problem.solve().
6. **Printing the Results:**
    - Finally, we print the status of the solution and the optimal values for each decision variable, along with the total number of waiters required.

Case Study - 2 (Mixed Integer Problem)

A company manufactures three products: P1, P2, and P3. Each product has an associated profit and requires a specific amount of time on three different machines. The objective is to determine the number of units of each product to produce in order to maximize profit, subject to the available machine hours.

Data

- **Profit per unit:**
    - **P1: Rs. 200**
    - **P2: Rs. 400**
    - **P3: Rs. 300**
- **Time required on each machine (hours per unit):**
    - **Machine 1: Available for 600 hours**
    - **Machine 2: Available for 400 hours**
    - **Machine 3: Available for 800 hours**

Product	Machine 1 (hours/unit)	Machine 2 (hours/unit)	Machine 3 (hours/unit)
P1	30	20	10
P2	40	10	30
P3	20	20	20

- **Constraints:**
    - **Fractional units of products P1 and P2 are not allowed.**

Formulation as an Integer Programming Problem

Decision Variables:

- **x1 : Number of units of P1 produced**
- **x2 : Number of units of P2 produced**
- **x3 : Number of units of P3 produced (can be fractional)**

Objective Function: Maximize the profit: max Z =200x1 +400x2+ 300x3

Constraints:

1. **Machine 1 : 30x1+40x2+20x3≤600**
2. **Machine 2 : 20x1+10x2+20x3≤400**
3. **Machine 3 : 10x1+30x2+20x3≤800**
4. **Integer constraints : x1,x2≥0 and integer**
5. **Non-negativity constraints : x3≥0**

Explanation

This IPP ensures that the total time used on each machine does not exceed its available hours while maximizing the profit. Products P1 and P2 must be produced in integer quantities, while P3 can be fractional.

In [2]:
```
import pulp

# Define the problem
problem = pulp.LpProblem("Maximize_Profit", pulp.LpMaximize)

# Define decision variables
x1 = pulp.LpVariable('x1', lowBound=0, cat='Integer')
x2 = pulp.LpVariable('x2', lowBound=0, cat='Integer')
x3 = pulp.LpVariable('x3', lowBound=0, cat='Continuous')

# Objective function: maximize the total profit
problem += 200*x1 + 400*x2 + 300*x3, "Total_Profit"

# Constraints
problem += 30*x1 + 40*x2 + 20*x3 <= 600, "Machine_1_Constraint"
problem += 20*x1 + 10*x2 + 20*x3 <= 400, "Machine_2_Constraint"
problem += 10*x1 + 30*x2 + 20*x3 <= 800, "Machine_3_Constraint"

# Solve the problem
problem.solve()

# Print the results
print("Status:", pulp.LpStatus[problem.status])
print("Optimal number of products to produce:")
print("P1:", pulp.value(x1))
print("P2:", pulp.value(x2))
print("P3:", pulp.value(x3))
print("Total profit:", pulp.value(problem.objective))
```

```
Status: Optimal
Optimal number of products to produce:
P1: 0.0
P2: 7.0
P3: 16.0
Total profit: 7600.0
```

Explanation of the Code

1. **Problem Definition:**
   - **We define the problem as a maximization problem using pulp.LpProblem.**
2. **Decision Variables:**
   - **We define the decision variables x1, x2, and x3, where x1 and x2 are integers, and x3 is continuous.**
3. **Objective Function:**
   - **The objective function is to maximize the total profit, represented by 200*x1 + 400*x2 + 300*x3.**
4. **Constraints:**
   - **We add the constraints for each machine to ensure that the total time used does not exceed the available hours.**
5. **Solving the Problem:**
   - **We solve the problem using problem.solve().**
6. **Printing the Results:**
   - **Finally, we print the status of the solution, the optimal number of products to produce for each type, and the total profit.**