Only problem 1 and 6 will be graded.

Problem 1 : Integer program

Solve the following program by using linprog function and branch and bound method:

$$Objective : max(3x + 4y)$$

$$\begin{aligned} x + 2y &\leq 7 \\ 3x - y &\geq 0 \\ x - y &\leq 2 \\ x, y &\in Z^+ \cup \{0\} \end{aligned}$$

```
Solution:
```

scipy.optimize.linprog: **minimize** a linear objective function subject to linear equality and inequality constraints.

$$Objective : min(-3x - 4y)$$

$$\begin{aligned} x + 2y &\leq 7 \\ -3x + y &\leq 0 \\ x - y &\leq 2 \\ x, y &\geq 0 \end{aligned}$$

In [1]:
```
import numpy as np
from scipy.optimize import linprog

c = [-3, -4]

A = [[1, 2],
     [-3, 1],
     [1, -1]]

b = [7, 0, 2]

x_bounds = (0, None)
y_bounds = (0, None)

result = linprog(c, A_ub=A, b_ub=b, bounds=[x_bounds, y_bounds], method='

print(f"Optimal value: {-result.fun}, x: {result.x[0]}, y: {result.x[1]}"
```

```
Optimal value: 17.666666666666668, x: 3.6666666666666665, y: 1.66666666666
66667
```

In [2]:
```
# Subproblem 1: x <= 3

x_bounds_sub1 = (0, 3)
```

```
result_sub1 = linprog(c, A_ub=A, b_ub=b, bounds=[x_bounds_sub1, y_bounds]

print(f"Subproblem 1 (x <= 3): Optimal value: {-result_sub1.fun}, x: {res
```

```
Subproblem 1 (x <= 3): Optimal value: 17.0, x: 3.0, y: 2.0
```

In [3]:

```
# Subproblem 2: x >= 4

x_bounds_sub2 = (4, None)

result_sub2 = linprog(c, A_ub=A, b_ub=b, bounds=[x_bounds_sub2, y_bounds]

if result_sub2.success:
    print(f"Subproblem 2 (x >= 4): Optimal value: {-result_sub2.fun}, x:
else:
    print("Subproblem 2 (x >= 4): No feasible solution.")
```

```
Subproblem 2 (x >= 4): No feasible solution.
```

Conclusion:

Since Subproblem 1 provides a feasible integer solution
and Subproblem 2 has no feasible solution,
the optimal integer solution to the original problem is:

$$\therefore \text{Optimal value} = 17, \quad x = 3, \quad y = 2$$

Problem 6 : Instraham

After several crises, Hamtaro is fed up with the manufacturing business and is now looking for new business opportunities. He finds out that opening social media platform could make a hefty sum of money. Moreover, since hamsters do not currently have a social media site, Hamtaro can monopolize the market easily. Therefore, he creates Instraham, the first social media website of hamsters, by hamsters, for hamsters.

After consulting with Koushi-kun, Hamtaro figures out that social network platforms often have the features shown in the table below. For each feature, the CPU load and storage load are shown with its associated business value score.

Feature name	CPU load (%)	storage load (%)	business value score
A	20	30	10
B	10	5	5
C	30	10	10
D	5	10	3
F	15	30	10
G	60	70	30
H	80	80	80

Feature name	CPU load (%)	storage load (%)	business value score
I	10	50	20
J	3	50	5

Feature A, and J is mandatory while the rest is optional. The objective is to maximize the business value score of the website while not overloading CPU and storage servers. His engineering friend, Taisho-kun, also suggests him that he could improve the website efficiency by performing the following operations:

- Feature compression. This method will reduce both CPU and storage load by half, but it also reduces the business value to 55% of the original value. Every feature could be compressed, but the number of compressed features in the website is limited to two.

- The usage of storage efficient algorithm. By using this method, the feature storage load is reduced by half but it also doubles the CPU load. However, only feature H, I, J can use this method. This method could not be used concurrently with feature compression.

From this information, which features should Hamtaro develop? ~~Use Amdahl's law to find the best speedup~~. Formulate the problem as an integer program and solve for an optimal solution.

Note : This problem is based on the blog (https://engineering.fb.com/2021/07/29/data-infrastructure/linear-programming/)

```
Solution:
```

Decision variables:

- x_i: Binary variable (0 or 1) for whether feature i is *selected* for the platform.
- c_i: Binary variable (0 or 1) for whether feature i is *compressed*.
- s_i: Binary variable (0 or 1) for whether feature i is uses the *storage-efficient* algorithm.

Objective:
maximize the business value score

$$max(\sum_i (x_i v_i - 0.45 c_i v_i))$$

; v_i is business value score of feature i

Constraints:

- Total CPU Load

$$\sum Normal - Feature\ compression + Storage\ efficient\ algorithm \leq 100$$

$$\sum_i (x_i \cdot CPU_i) - (c_i \cdot x_i \cdot 0.5CPU_i) + (s_i \cdot x_i \cdot 2CPU_i) \leq 100$$

$$\therefore \sum_i (x_i \cdot CPU_i)(1 - 0.5c_i + 2s_i) \leq 100$$

- Individual CPU Load

$$Actual\ Load \leq Ideal\ Load$$

$$(x_i \cdot CPU_i) - (c_i \cdot x_i \cdot 0.5CPU_i) + (s_i \cdot x_i \cdot 2CPU_i) \leq CPU_i$$

; CPU_i is CPU Load of feature i

- Total Storage Load

$$\sum_i (x_i \cdot Storage_i) - (c_i \cdot x_i \cdot 0.5Storage_i) - (s_i \cdot x_i \cdot 0.5Storage_i) \leq 100$$

$$\therefore \sum_i (x_i \cdot Storage_i)(1 - 0.5c_i - 0.5s_i) \leq 100$$

- Individual Storage Load

$$Actual\ Load \leq Ideal\ Load$$

$$(x_i \cdot Storage_i) - (c_i \cdot x_i \cdot 0.5Storage_i) - (s_i \cdot x_i \cdot 0.5Storage_i) \leq Storage_i$$

; $Storage_i$ is storage load of feature i

- Mandatory features

$$x_A = 1, \quad x_J = 1$$

- If use Feature compression must be selected first

$$c_i \leq x_i$$

- If use Storage efficient algorithm must be selected first

$$s_i \leq x_i$$

- Feature compression limit

$$\sum_i c_i \leq 2$$

- Storage efficient algorithm can use only by H, I, J

$$s_A = 0,\ s_B = 0,\ s_C = 0,\ s_D = 0,\ s_F = 0,\ s_G = 0$$

- Storage efficient algorithm could not be used concurrently with feature compression

$$s_i + c_i \leq 1$$

```
In [4]: import pulp

        prob = pulp.LpProblem("Instraham", pulp.LpMaximize)

        # Data
        features = ['A', 'B', 'C', 'D', 'F', 'G', 'H', 'I', 'J']
        cpu = {'A': 20, 'B': 10, 'C': 30, 'D': 5, 'F': 15, 'G': 60, 'H': 80, 'I':
        storage = {'A': 30, 'B': 5, 'C': 10, 'D': 10, 'F': 30, 'G': 70, 'H': 80,
        business_value = {'A': 10, 'B': 5, 'C': 10, 'D': 3, 'F': 10, 'G': 30, 'H'

        # Decision variables
        x = pulp.LpVariable.dicts("select", features, cat='Binary')  # Feature se
        c = pulp.LpVariable.dicts("compress", features, cat='Binary')  # Compress
        s = pulp.LpVariable.dicts("storage_efficient", features, cat='Binary')  #

        # Objective function: Maximize business value
        prob += pulp.lpSum([business_value[f] * (x[f] - 0.45 * c[f]) for f in fea
```

```
In [5]: # Constraints

        # CPU Load
        # individual
        for f in features:
            cpu_load = (cpu[f] * x[f]) - (0.5 * cpu[f] * c[f]) + (cpu[f] * s[f])
            prob += cpu_load <= cpu[f]
        # total
        prob += pulp.lpSum([cpu[f] * x[f] - 0.5 * cpu[f] * c[f] + cpu[f] * s[f] f

        # Storage Load
        # individual
        for f in features:
            storage_load = storage[f] * x[f] - 0.5 * storage[f] * c[f] - 0.5 * st
            prob += storage_load <= storage[f]
        # total
        prob += pulp.lpSum([storage[f] * x[f] - 0.5 * storage[f] * c[f] - 0.5 * s

        # Mandatory features: A and J
        prob += x['A'] == 1
        prob += x['J'] == 1

        # Feature Compression
        prob += pulp.lpSum(c[f] for f in features) <= 2  # Limit
        for f in features:
            prob += c[f] <= x[f] # must be selected features

        # Storage-efficient algorithm
        # Only H, I, J can use storage-efficient
        for f in ['A', 'B', 'C', 'D', 'F', 'G']:
            prob += s[f] == 0
        for f in features:
            prob += s[f] <= x[f] # must be selected features
            prob += s[f] + c[f] <= 1  # no concurrent

        # Solve the problem
        status = prob.solve()

        # Print results
        print("\nOptimal Feature Selection:")
        print("-" * 50)
print("Selected Features:")
for f in features:
    if x[f].value() > 0.5:  # Using 0.5 to handle floating-point imprecis
        optimizations = []
        if c[f].value() > 0.5:
            optimizations.append("compressed")
        if s[f].value() > 0.5:
            optimizations.append("storage-efficient")
        opt_str = f" ({', '.join(optimizations)})" if optimizations else
        print(f"- Feature {f}{opt_str}")

print("\nTotal Business Value:", pulp.value(prob.objective))
print("CPU Load:", sum(cpu[f] * x[f].value() - 0.5 * cpu[f] * c[f].value(
print("Storage Load:", sum(storage[f] * x[f].value() - 0.5 * storage[f] *
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - /Users/pupipatsingkhorn/miniconda3/envs/datascience/lib/pyt
hon3.9/site-packages/pulp/solverdir/cbc/osx/64/cbc /var/folders/m6/fz_qjnl
51s70hy69d_st2z240000gn/T/9007824fe965472fab77249139f4e57f-pulp.mps -max -
timeMode elapsed -branch -printingOptions all -solution /var/folders/m6/fz
_qjnl51s70hy69d_st2z240000gn/T/9007824fe965472fab77249139f4e57f-pulp.sol
(default strategy 1)
At line 2 NAME          MODEL
At line 3 ROWS
At line 61 COLUMNS
At line 313 RHS
At line 370 BOUNDS
At line 398 ENDATA
Problem MODEL has 56 rows, 27 columns and 179 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Continuous objective value is 79.7733 - 0.00 seconds
Cgl0002I 6 variables fixed
Cgl0003I 2 fixed, 0 tightened bounds, 1 strengthened rows, 0 substitutions
Cgl0004I processed model has 10 rows, 16 columns (16 integer (16 of which
binary)) and 55 elements
Cutoff increment increased from 1e-05 to 0.04995
Cbc0038I Initial state - 1 integers unsatisfied sum - 0.3125
Cbc0038I Pass   1: suminf.    0.31250 (1) obj. -68.25 iterations 2
Cbc0038I Solution found of -13.25
Cbc0038I Rounding solution of -33.25 is better than previous of -13.25

Cbc0038I Before mini branch and bound, 15 integers at bound fixed and 0 co
ntinuous
Cbc0038I Mini branch and bound did not improve solution (0.02 seconds)
Cbc0038I Round again with cutoff of -36.8
Cbc0038I Pass   2: suminf.    0.31250 (1) obj. -68.25 iterations 0
Cbc0038I Pass   3: suminf.    0.29438 (1) obj. -36.8 iterations 1
Cbc0038I Pass   4: suminf.    0.29438 (1) obj. -36.8 iterations 0
Cbc0038I Pass   5: suminf.    0.29438 (1) obj. -36.8 iterations 0
Cbc0038I Pass   6: suminf.    0.27578 (2) obj. -36.8 iterations 3
Cbc0038I Pass   7: suminf.    0.27578 (2) obj. -36.8 iterations 1
Cbc0038I Pass   8: suminf.    0.25000 (1) obj. -68.25 iterations 1
Cbc0038I Pass   9: suminf.    0.35688 (1) obj. -36.8 iterations 1
Cbc0038I Pass  10: suminf.    0.10688 (1) obj. -36.8 iterations 2
Cbc0038I Pass  11: suminf.    0.10688 (1) obj. -36.8 iterations 2
Cbc0038I Pass  12: suminf.    0.12500 (1) obj. -38.25 iterations 1
Cbc0038I Pass  13: suminf.    0.42771 (2) obj. -36.8 iterations 3
Cbc0038I Pass  14: suminf.    0.42771 (2) obj. -36.8 iterations 1
Cbc0038I Pass  15: suminf.    0.43750 (1) obj. -57.75 iterations 1
Cbc0038I Pass  16: suminf.    0.30063 (1) obj. -36.8 iterations 1
Cbc0038I Pass  17: suminf.    0.58047 (2) obj. -36.8 iterations 2
Cbc0038I Pass  18: suminf.    0.58047 (2) obj. -36.8 iterations 1
Cbc0038I Pass  19: suminf.    0.41404 (4) obj. -36.8 iterations 5
Cbc0038I Pass  20: suminf.    0.09438 (1) obj. -36.8 iterations 4
Cbc0038I Pass  21: suminf.    0.12500 (1) obj. -39.25 iterations 2
Cbc0038I Pass  22: suminf.    0.20000 (1) obj. -57.2 iterations 8
Cbc0038I Solution found of -56.75
Cbc0038I Rounding solution of -61.75 is better than previous of -56.75

Cbc0038I Before mini branch and bound, 6 integers at bound fixed and 0 con
tinuous
```

```
Cbc0038I Full problem 10 rows 16 columns, reduced to 2 rows 4 columns
Cbc0038I Mini branch and bound did not improve solution (0.02 seconds)
Cbc0038I Round again with cutoff of -63.1
Cbc0038I Reduced cost fixing fixed 9 variables on major pass 3
Cbc0038I Pass  23: suminf.    0.31250 (1) obj. -68.25 iterations 0
Cbc0038I Pass  24: suminf.    0.37687 (1) obj. -63.1 iterations 1
Cbc0038I Pass  25: suminf.    0.37687 (1) obj. -63.1 iterations 0
Cbc0038I Pass  26: suminf.    0.25000 (1) obj. -68.25 iterations 2
Cbc0038I Pass  27: suminf.    0.31438 (1) obj. -63.1 iterations 1
Cbc0038I Pass  28: suminf.    0.25000 (1) obj. -68.25 iterations 1
Cbc0038I Pass  29: suminf.    0.25000 (1) obj. -68.25 iterations 0
Cbc0038I Pass  30: suminf.    0.25000 (1) obj. -68.25 iterations 1
Cbc0038I Pass  31: suminf.    0.25000 (1) obj. -68.25 iterations 1
Cbc0038I Pass  32: suminf.    0.83244 (2) obj. -63.1 iterations 3
Cbc0038I Pass  33: suminf.    0.25000 (1) obj. -68.25 iterations 3
Cbc0038I Pass  34: suminf.    0.83244 (2) obj. -63.1 iterations 3
Cbc0038I Pass  35: suminf.    0.83244 (2) obj. -63.1 iterations 1
Cbc0038I Pass  36: suminf.    0.83244 (2) obj. -63.1 iterations 1
Cbc0038I Pass  37: suminf.    0.43938 (1) obj. -63.1 iterations 4
Cbc0038I Pass  38: suminf.    0.37500 (1) obj. -68.25 iterations 2
Cbc0038I Pass  39: suminf.    0.43937 (1) obj. -63.1 iterations 1
Cbc0038I Pass  40: suminf.    0.83244 (2) obj. -63.1 iterations 4
Cbc0038I Pass  41: suminf.    0.96101 (2) obj. -63.1 iterations 3
Cbc0038I Pass  42: suminf.    0.49813 (1) obj. -63.1 iterations 5
Cbc0038I Pass  43: suminf.    0.43750 (1) obj. -68.25 iterations 1
Cbc0038I Pass  44: suminf.    0.31250 (1) obj. -68.25 iterations 1
Cbc0038I Pass  45: suminf.    0.37687 (1) obj. -63.1 iterations 1
Cbc0038I Pass  46: suminf.    0.31250 (1) obj. -68.25 iterations 1
Cbc0038I Pass  47: suminf.    0.43750 (1) obj. -68.25 iterations 2
Cbc0038I Pass  48: suminf.    0.43750 (1) obj. -68.25 iterations 1
Cbc0038I Pass  49: suminf.    0.43750 (1) obj. -68.25 iterations 0
Cbc0038I Pass  50: suminf.    0.89494 (2) obj. -63.1 iterations 4
Cbc0038I Pass  51: suminf.    0.31250 (1) obj. -68.25 iterations 3
Cbc0038I Pass  52: suminf.    0.31250 (1) obj. -68.25 iterations 2
Cbc0038I No solution found this major pass
Cbc0038I Before mini branch and bound, 12 integers at bound fixed and 0 co
ntinuous
Cbc0038I Mini branch and bound did not improve solution (0.03 seconds)
Cbc0038I After 0.03 seconds - Feasibility pump exiting with objective of -
61.75 - took 0.00 seconds
Cbc0012I Integer solution of -61.75 found by feasibility pump after 0 iter
ations and 0 nodes (0.03 seconds)
Cbc0038I Full problem 10 rows 16 columns, reduced to 2 rows 2 columns
Cbc0001I Search completed - best objective -61.75, took 0 iterations and 0
nodes (0.03 seconds)
Cbc0035I Maximum depth 0, 9 variables fixed on reduced cost
Cuts at root node changed objective from -68.25 to -27.75
Probing was tried 0 times and created 0 cuts of which 0 were active after
adding rounds of cuts (0.000 seconds)
Gomory was tried 0 times and created 0 cuts of which 0 were active after a
dding rounds of cuts (0.000 seconds)
Knapsack was tried 0 times and created 0 cuts of which 0 were active after
adding rounds of cuts (0.000 seconds)
Clique was tried 0 times and created 0 cuts of which 0 were active after a
dding rounds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 0 times and created 0 cuts of which 0 were
active after adding rounds of cuts (0.000 seconds)
FlowCover was tried 0 times and created 0 cuts of which 0 were active afte
r adding rounds of cuts (0.000 seconds)
TwoMirCuts was tried 0 times and created 0 cuts of which 0 were active aft
```

```
er adding rounds of cuts (0.000 seconds)
ZeroHalf was tried 0 times and created 0 cuts of which 0 were active after
adding rounds of cuts (0.000 seconds)

Result - Optimal solution found

Objective value:                61.75000000
Enumerated nodes:               0
Total iterations:               0
Time (CPU seconds):             0.01
Time (Wallclock seconds):       0.03

Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.01   (Wallclock seconds):       0.03


Optimal Feature Selection:
--------------------------------------------------
Selected Features:
- Feature A
- Feature B
- Feature H (compressed)
- Feature J (compressed)

Total Business Value: 61.75
CPU Load: 71.5
Storage Load: 100.0
```

Conclusion:

Optimal Feature Selection:

Selected Features:

- Feature A
- Feature B
- Feature H (compressed)
- Feature J (compressed)

Optimal Business Value Score: 61.75
CPU Load: 71.5
Storage Load: 100.0