```
from lpsolve55 import * # Import lpsolve
import matplotlib.pyplot as plt # Import plot
import numpy as np # import numpy

# Data Import----------------------------#
CPU = ["Intel Core i9-7940X @ 3.10GHz",
        "Intel Core i9-7960X @ 2.80GHz",
        "Intel Core i9-7920X @ 2.90GHz",
        "Intel Core i9-7900X @ 3.30GHz",
        "AMD Ryzen Threadripper 1950X",
        "AMD Ryzen Threadripper 1920X",
        "Intel Core i7-7820X @ 3.60GHz",
        "Intel Xeon E5-2673 v3 @ 2.40GHz",
        "Intel Xeon E5-2680 v2 @ 2.80GHz",
        "Intel Core i7-7800X @ 3.50GHz",
        "AMD Ryzen 7 1700",
        "Intel Core i7-3970X @ 3.50GHz",
        "Intel Xeon E3-1270 v6 @ 3.80GHz",
        "Intel Xeon E3-1270 v3 @ 3.50GHz",
        "Intel Core i5-6402P @ 2.80GHz"]

GPU = ["GeForce GTX 1080 Ti",
        "GeForce GTX 1080",
        "GeForce GTX 980 Ti",
        "GeForce GTX 1070",
        "GeForce GTX 980",
        "GeForce GTX 780 Ti",
        "GeForce GTX 1060",
        "GeForce GTX 780",
        "GeForce GTX 960",
        "Radeon RX560",
        "GeForce GTX 570",
        "GeForce GTX 480",
        "GeForce GT 1030",
        "GeForce GT 640",
        "Radeon HD 5670"]

HardDrive = ["NVMe INTEL SSDPE2MW01",
             "NVMe INTEL SSDPE2MW01",
             "Intel DC P3700 400GB NVMe",
             "Intel SSD 750 800GB NVMe",
             "INTEL SSDPEDMW800G4",
             "Intel DC P3500 400GB NVMe",
             "PLEXTOR PX-512M8PeY",
             "INTEL SSDPEKKW010T7x1",
             "SAMSUNG XP941 M.2 256GB",
             "Intel 600p Series 256GB"]

RAM = ["Corsair CMD16GX4M2B3200C14 8GB",
        "Corsair CMK16GX4M2B3333C16 8GB",
        "Corsair CMK16GX4M2B3733C17 8GB",
        "Corsair CMK16GX4M2A2400C16 8GB",
        "Corsair CMSX16GX4M2A2400C16 8GB",
        "Mushkin 99[2/7/4]200F 8GB",
        "Mushkin 99[2/7/4]197F 8GB",
        "Crucial Technology CT8G4DFD8213.C16FAR1 8GB",
        "Kingston 9965589-013.A00G 8GB",
        "Samsung M393A1G43DB0-CPB 8GB"]

values_CPU = [100.0, 95.1, 80.2, 76.1, 75.4, 57.0, 52.1, 42.6,
39.3, 31.8, 28.1, 23.7, 19.1, 14.3, 8.2]

values_GPU = [100, 89.272, 84.053, 81.481, 71.14, 65.876, 64.791,
59.26, 43.119, 35.165, 32.838, 32.369, 16.995, 9.531, 7.977]

```

```
values_HardDrive = [100, 92.60720001, 76.22807085, 75.28534803, 73.72717826,
69.56877077, 60.92598279, 30.8351352, 29.05718521, 28.10888866]

values_RAM = [100.0, 92.7, 90.7, 74.3, 73.4, 64.4, 63.2, 62.8, 49.9, 48.2]

prices_cpu = [1399.00, 1699.00, 1129.89, 962.89, 979.99, 782.87, 574.99,
700.00, 559.00, 363.38, 299.99, 369.90, 364.98, 369.99, 189.99]

price_gpu = [739.99, 499.99, 565.50, 389.99, 417.04, 369.99, 259.99, 249.99,
139.99, 109.99, 99.99, 71.99, 69.99, 59.99, 47.84]

price_HardDrive = [999.99, 899.89, 706.87, 679.99, 594.25, 499.99,
409.98, 349, 208.99,109.99]

prices_RAM = [219.99, 169.99, 199.99, 166.99, 119.99, 86.99, 88.32, 82.99, 94.95, 90.00]
#----------------------------------------#

def nums(num, count):
    '''
    Generate a list of given number for given count of times

    Parameters
    --------
    num: desired value
    count: times that the number repeats
    '''
    result = []
    for i in range(count):
        result.append(num)
    return result


def maximizePerformance(cost, scale, ifPrint):
    '''
    Find the best performance using LP solve

    Parameters
    ----------
    cost: the cost restriction
    scale: if the client wants components at different weight, use scale to weight
    ifPrint: True for printing objective function and variables after solving. False otherwise

    Returns
    ----------
    obj_value: the value after the maximization
    var: the coefficients of the x's
    '''
    variables = 50

    GIVEN_COST = cost

    lp = lpsolve('make_lp', 0, variables)

    for i in range(variables):
        ret = lpsolve('set_binary', lp, i, True) # Set all variables to be binary

    # Value Maximization
    value = []

    value += [v * scale[0] for v in values_CPU]
    value += [v * scale[1] for v in values_GPU]
    value += [v * scale[2] for v in values_HardDrive]
    value += [v * scale[3] for v in values_RAM]

    lpsolve('set_obj_fn', lp, value)
```

```
    lpsolve('set_maxim', lp)

    # Price Restriction
    price = []

    price += prices_cpu
    price += prices_cpu
    price += price_HardDrive
    price += prices_RAM

    lpsolve('add_constraint', lp, price, LE, GIVEN_COST)

    # constraint: one and only one CPU
    cpu_constraint = nums(1, 15)
    cpu_constraint += nums(0, 35)
    lpsolve('add_constraint', lp, cpu_constraint, EQ, 1)

    # constraint: one and only one GPU
    gpu_constraint = nums(0, 15)
    gpu_constraint += nums(1, 15)
    gpu_constraint += nums(0, 20)
    lpsolve('add_constraint', lp, gpu_constraint, EQ, 1)

    # constraint: one and only one HardDrive
    hd_constraint = nums(0, 30)
    hd_constraint += nums(1, 10)
    hd_constraint += nums(0, 10)
    lpsolve('add_constraint', lp, hd_constraint, EQ, 1)

    # constraint: one and only one RAM
    ram_constraint = nums(0, 40)
    ram_constraint += nums(1, 10)
    lpsolve('add_constraint', lp, ram_constraint, EQ, 1)

    lpsolve("solve", lp)
    obj_value = lpsolve('get_objective', lp)
    var = lpsolve('get_variables', lp)[0]
    if ifPrint:
        print(obj_value)
        print(var)
    return obj_value, var

def minimizeCost(scores):
    variables = 50

    lp = lpsolve('make_lp', 0, variables)

    # Set all variables to be binary
    for i in range(variables):
        ret = lpsolve('set_binary', lp, i, True)

    # Price Minimization
    price = []

    price += prices_cpu
    price += prices_cpu
    price += price_HardDrive
    price += prices_RAM

    lpsolve('set_obj_fn', lp, price)
    lpsolve('set_minim', lp)

    # Performance Resitrction
    cur = 0
    value = nums(0, 50)
```

```
196         for i in range(len(values_CPU)):
197             value[i] = values_CPU[i]
198         lpsolve('add_constraint', lp, value, GE, scores[0])
199         value = nums(0, 50) # Reset
200         cur += len(values_CPU)
201 
202         for i in range(len(values_GPU)):
203             value[i + cur] = values_GPU[i]
204         lpsolve('add_constraint', lp, value, GE, scores[1])
205         value = nums(0, 50) # Reset
206         cur += len(values_GPU)
207 
208         for i in range(len(values_HardDrive)):
209             value[i + cur] = values_HardDrive[i]
210         lpsolve('add_constraint', lp, value, GE, scores[2])
211         value = nums(0, 50) # Reset
212         cur += len(values_HardDrive)
213 
214         for i in range(len(values_RAM)):
215             value[i + cur] = values_RAM[i]
216         lpsolve('add_constraint', lp, value, GE, scores[3])
217         value = nums(0, 50) # Reset
218 
219         # constraint: one and only one CPU
220         cpu_constriant = nums(1, 15)
221         cpu_constriant += nums(0, 35)
222         ret = lpsolve('add_constraint', lp, cpu_constriant, EQ, 1)
223 
224         # constraint: one and only one GPU
225         gpu_constraint = nums(0, 15)
226         gpu_constraint += nums(1, 15)
227         gpu_constraint += nums(0, 20)
228         ret = lpsolve('add_constraint', lp, gpu_constraint, EQ, 1)
229 
230         # constraint: one and only one HardDrive
231         hd_constraint = nums(0, 30)
232         hd_constraint += nums(1, 10)
233         hd_constraint += nums(0, 10)
234         ret = lpsolve('add_constraint', lp, hd_constraint, EQ, 1)
235 
236         # constraint: one and only one RAM
237         ram_constraint = nums(0, 40)
238         ram_constraint += nums(1, 10)
239         ret = lpsolve('add_constraint', lp, ram_constraint, EQ, 1)
240 
241         lpsolve("solve", lp)
242         result = lpsolve('get_objective', lp)
243         print(result)
244         print(lpsolve('get_variables', lp)[0])
245         return result
246 
247     # Plot a "Performance vs Cost" graph
248     costs = np.arange(600, 6200, 200).tolist()
249     performance = []
250     for cost in costs:
251         result = maximizePerformance(cost, [1, 1, 1, 1], False)
252         performance.append(result[0])
253         components = result[1]
254         for i in range(15):
255             if components[i] != 0:
256                 cpu = i
257                 break
258         for i in range(15, 30):
259             if components[i] != 0:
260                 gpu = i - 15
```

```
            break
    for i in range(30, 40):
        if components[i] != 0:
            hd = i - 30
            break
    for i in range(40, 50):
        if components[i] != 0:
            ram = i - 40
            break
    print("----------------------------")
    print("=> When the cost is " + str(cost) + ", the best performance score is " + str(result[0
    ]))
    print("=> At this cost level, the components are: ")
    print("CPU: " + CPU[cpu])
    print("Graphic: " + GPU[gpu])
    print("Hard Drive: " + HardDrive[hd])
    print("RAM: " + RAM[ram])
    print("----------------------------")
# Plot the graph
plt.plot(costs, performance, 'b--')
plt.xlabel("Given Cost")
plt.ylabel("Best Performance")
plt.title("Using Linear Programming To Find Best Performance")
plt.show()

# Variations

# 1. Maximize the performance with given cost and given weight for differnt components
result = maximizePerformance(1000, [5, 9, 3, 5], True)
components = result[1]
for i in range(15):
    if components[i] != 0:
        cpu = i
        break
for i in range(15, 30):
    if components[i] != 0:
        gpu = i - 15
        break
for i in range(30, 40):
    if components[i] != 0:
        hd = i - 30
        break
for i in range(40, 50):
    if components[i] != 0:
        ram = i - 40
        break
performance = values_CPU[cpu] + values_GPU[gpu] + values_HardDrive[hd] + values_RAM[ram]

print("----------------------------")
print("=> When the cost is 1000, and we want CPU at weight 5, \
GPU at weight 9, Hard Drive at weight 3, and RAM at weight 5,\
the performance is " + str(performance))
print("=> Under this circumstance, the components are: ")
print("CPU: " + CPU[cpu])
print("Graphic: " + GPU[gpu])
print("Hard Drive: " + HardDrive[hd])
print("RAM: " + RAM[ram])
print("----------------------------")

# 2. Minimize the cost with required scores for different components
cost = minimizeCost([70, 80, 30, 30])
print("----------------------------")
print("=> When we want CPU at least 70, GPU at least 80, Hard Drive \
at least 30, and RAM at least 30")
print("=> The minimum cost found using LP is: $" + str(cost))
```

```
325   print("---------------------------")
```