מטלה 6 שאלה 4

https://github.com/MayaHayat/EconAlgo_Ex6Q4/tree/main

```
[127] def get_player_room(node1, node2):
        id1, id2 = int(node1.split('_')[-1]), int(node2.split('_')[-1])
        if 'player' in node1:
          return id1, id2
        elif 'player' in node2:
          return id2, id1
        else:
          print("Invalid node format:", node1, node2)
          exit(1)
```

This function extracts the room_id and player_id from the nodes as the “nx.max_weight_matching” doesn’t return the nodes in the same order each time.

```
def find_rent_prices(valuations: list[list[float]], price):
    """
    Finds the optimal room rents given player-room valuations and a total rental price.

    Args:
    - valuations (list[list[float]]): A 2D list representing the valuations of players for rooms.
    - price (float): Total rental price.

    Returns:
    - G (nx.Graph): The bipartite graph representing the player-room matching.

    Examples:
    >>> G = find_rent_prices([[20,30,40],[40,30,20],[30,30,30]], 90)
    Max matching {('room_2', 'player_0'), ('room_0', 'player_1'), ('player_2', 'room_1')}
    {0: 2, 1: 0, 2: 1}
    Status: optimal
    Value: 0.0
    rents: room_0=31.667, room_1=26.667, room_2=31.667

    >>> G = find_rent_prices([[350,250,350],[600,400,400],[400,200,250]], 1000)
    Max matching {('room_1', 'player_2'), ('player_0', 'room_2'), ('room_0', 'player_1')}
    {2: 1, 0: 2, 1: 0}
    Status: optimal
    Value: 0.0
    rents: room_0=439.085, room_1=239.085, room_2=321.831

    >>> G = find_rent_prices([[10,140],[0,150]], 100)
    Max matching {('room_1', 'player_1'), ('player_0', 'room_0')}
    {1: 1, 0: 0}
    Status: optimal
    Value: 0.0
    rents: room_0=-20.000, room_1=120.000

    >>> G = find_rent_prices( [[20,130],[0,150]], 120)
    Max matching {('room_1', 'player_1'), ('player_0', 'room_0')}
    {1: 1, 0: 0}
    Status: optimal
    Value: 0.0
    rents: room_0=-5.000, room_1=125.000

    """
```

```python
G = nx.Graph()  # Create an undirected graph

num_players = len(valuations)
num_rooms = len(valuations[0])

# ------------------------------ Part 1: match rooms - players ------------------------------

# Add players and rooms as nodes
for i in range(num_players):
    player_name = f"player_{i}"
    #player_name = i
    G.add_node(player_name, bipartite=0)

for j in range(num_rooms):
    room_name = f"room_{j}"
    #room_name = j
    G.add_node(room_name, bipartite=1)

# Add edges with weights and named nodes
for i in range(num_players):
    for j in range(num_rooms):
        weight = valuations[i][j]
        if weight > 0:
            G.add_edge(f"player_{i}", f"room_{j}", weight=weight)
```

```python
max_weight_matching = nx.max_weight_matching(G, maxcardinality=True)
print("Max matching", max_weight_matching)

# Create a dict to store player-room assignments
matches = {}
for node1, node2 in max_weight_matching:
  player_id, room_id = get_player_room(node1, node2)
  matches[player_id] = room_id
print(matches)

# ------------------------------ Part 2 : set pricing ------------------------------
variables = [cvxpy.Variable() for _ in range(num_rooms)]
fixed_constraints = [sum(variables) == price]
for player_id in range(num_players):
  matched_room_id = matches[player_id]
  for room_id in range(num_rooms):
    if matched_room_id == room_id:
      continue
    constraint = (valuations[player_id][matched_room_id] - variables[matched_room_id]) \
                >= (valuations[player_id][room_id] - variables[room_id])

    matched_room_id = matches[player_id]
    fixed_constraints.append(constraint)

prob = cvxpy.Problem(cvxpy.Minimize(0), constraints=fixed_constraints)
prob.solve()
print("Status:", prob.status)
print("Value:", prob.value)

print('rents: ' + ', '.join(['room_{}={:.3f}'.format(room_id, variable.value) for room_id, variable in enumerate(variables)]))
```

This function creates a bipartite graph, where the players are the nodes on one side and the rooms are on the other side.
The function adds edges between the nodes with the given valuations from the matrix as the weights. We run the max_weight_matching function which returns a set which we use to create a dictionary where the key is the player's index in the valuations matrix and the value is the room's index.
The next step is to set all the constraints and use the cvxpy library to solve the problem and find what price should be set for each room.

In the second part we had to try to set all prices to be larger than 0, if it is possible of course:

```
def find_rent_with_nonnegative_prices(valuations: list[list[float]], price):
    """
    Examples:
    >>> G = find_rent_with_nonnegative_prices([[20,30,40],[40,30,20],[30,30,30]], 90)
    Max matching {('room_2', 'player_0'), ('room_0', 'player_1'), ('player_2', 'room_1')}
    {0: 2, 1: 0, 2: 1}
    Status: optimal
    Min price: 29.999999999415706
    rents: room_0=30.000, room_1=30.000, room_2=30.000

    >>> G = find_rent_with_nonnegative_prices([[10,140],[0,150]], 100)
    Max matching {('room_1', 'player_1'), ('player_0', 'room_0')}
    {1: 1, 0: 0}
    Status: optimal
    Min price: -15.000000005359155
    rents: room_0=-15.000, room_1=115.000

    >>> G = find_rent_with_nonnegative_prices([[20,130],[0,150]], 120)
    Max matching {('room_1', 'player_1'), ('player_0', 'room_0')}
    {1: 1, 0: 0}
    Status: optimal
    Min price: 4.999999994141455
    rents: room_0=5.000, room_1=115.000

    >>> G = find_rent_with_nonnegative_prices([[36,34,31,0],[31,36,33,0],[34,30,36,0],[32,33,35,0]], 100)
    Max matching {('room_1', 'player_1'), ('room_2', 'player_2'), ('player_3', 'room_3'), ('room_0', 'player_0')}
    {1: 1, 2: 2, 3: 3, 0: 0}
    Status: optimal
    Min price: -0.24999999998378383
    rents: room_0=32.750, room_1=32.750, room_2=34.750, room_3=-0.250
```

In the third example we can see that it did make a difference, the player who got the worse room has to pay in this case 5$ and the second player pays only 115$. In the first algorithm we got the the first player receives 5$ and the other player pays 125$ in total which he wouldn't probably want to do.

```
max_weight_matching = nx.max_weight_matching(G, maxcardinality=True) # we must add the maxcardinality so it matches all players to room
print("Max matching", max_weight_matching)

# Create a dict to store player-room assignments
matches = {}
for node1, node2 in max_weight_matching:
  player_id, room_id = get_player_room(node1, node2)
  matches[player_id] = room_id
print(matches)

# ------------------------------ Part 2 : set pricing ------------------------------
variables = [cvxpy.Variable() for _ in range(num_rooms)]
min_price = cvxpy.Variable() # set a min price variable so we can check if >  0

fixed_constraints = [sum(variables) == price] +\
                    [variables[i] >= min_price for i in range(len(variables))]

for player_id in range(num_players):
  matched_room_id = matches[player_id]
  for room_id in range(num_rooms):
    if matched_room_id == room_id:
      continue
    constraint = (valuations[player_id][matched_room_id] - variables[matched_room_id]) \
                 >= (valuations[player_id][room_id] - variables[room_id])

    matched_room_id = matches[player_id]
    fixed_constraints.append(constraint)

prob = cvxpy.Problem(cvxpy.Maximize(min_price), constraints=fixed_constraints)
prob.solve()
print("Status:", prob.status)
```

In order to achieve this, the following changes were made:
We first had to create another variable named "min_price" and we added to the constraints that all prices must be larger than or equal to min_price. After having added all those constraints we changed the problem so it maximizes the min_price variable.