Report 3

Team information.

- Team leader: Oleynik Maxim
- Team member 1: Lobov Gleb 5/5
- Team member 2: Fakhrutdinov Bulat 5/5
- Team member 3: Kachmazov Alexander 5/5
- Team member 4: Oleynik Maxim 5/5

All members have made maximum participation

Link to the product.

- The product is available: https://github.com/Dart-NEW/IntroToOptimizationHW3

Programming language.

- Programming language: Python

Transportation Model

Source	Destination				Supply
	B_1	B_2	B_3	B_4	
A_1	7	8	1	2	160
A_2	4	5	9	8	140
A_3	9	2	3	6	170
Demand	120	50	190	110	470

Source	Destination				Supply
	B_1	B_2	B_3	B_4	
A_1	31	21	56	12	13
A_2	65	54	21	21	12
A_3	21	19	56	52	22
Demand	12	5	19	11	47

Source	Destination				Supply
	B_1	B_2	B_3	B_4	
A_1	22	33	55	22	555
A_2	43	88	11	44	444
A_3	33	77	66	22	555
Demand	222	333	444	555	1554

Input

The input contains:

- A vector of coefficients of supply - S.
- A matrix of coefficients of costs - C.
- A vector of coefficients of demand - D.

Output/Results

The output contains:

- The string "The method is not applicable!"

  or

- The string "The problem is not balanced!"

  or

- Print (demonstrate) input parameter table (a table constructed using matrix C, vectors S and D).

Code

```
def not_applicable():
    print("The method is not applicable!")
    exit()


def find_difference(cost, suppy, demand, axis):
    res = []
    if axis == 0:
        for i in range(len(suppy)):
            temp = []
            for j in range(len(demand)):
                if suppy[i] != 0 and demand[j] != 0:
                    temp.append(cost[i][j])
            temp = sorted(temp)[:2]
            if len(temp) == 2:
                res.append(temp[1] - temp[0])
            else:
                res.append(-1)
    else:
        for i in range(len(demand)):
            temp = []
            for j in range(len(suppy)):
                if suppy[j] != 0 and demand[i] != 0:
                    temp.append(cost[j][i])
            temp = sorted(temp)[:2]
            if len(temp) == 2:
                res.append(temp[1] - temp[0])
            else:
                res.append(-1)
    return res


def north_west(supply, cost, demand):
    i, j = 0, 0
    res = []
    res_sum = 0
    while i != 2 or j != 3:
        if supply[i] == 0:
            i += 1
        if demand[j] == 0:
            j += 1
        temp = demand[j]
        demand[j] -= min(supply[i], temp)
        res.append({'position': (i, j), 'cost': cost[i][j], 'allocation': min(
           supply[i], temp)})
        res_sum += cost[i][j] * min(supply[i], temp)
        supply[i] -= min(supply[i], temp)
    return res, res_sum


def vogel_approximation(supply, cost, demand):
    res = []
    res_sum = 0
    while True:
        row_diff = find_difference(cost, supply, demand, 0)
        col_diff = find_difference(cost, supply, demand, 1)
```

```
        if (row_diff.count(-1) + col_diff.count(-1)) == 7:
            break
        max_in_row = max(row_diff)
        max_in_col = max(col_diff)
        if max_in_row > max_in_col:
            min_el = 10 ** 10
            ind_row = row_diff.index(max_in_row)
            ind_col = 0
            for i in range(len(cost[ind_row])):
                if demand[i] > 0 and cost[ind_row][i] < min_el:
                    min_el = cost[ind_row][i]
                    ind_col = i
        else:
            ind_col = col_diff.index(max_in_col)
            ind_row = -1
            min_el = 10 ** 10
            for i in range(len(cost)):
                if supply[i] > 0 and cost[i][ind_col] < min_el:
                    min_el = cost[i][ind_col]
                    ind_row = i
        temp = demand[ind_col]
        demand[ind_col] -= min(supply[ind_row], temp)
        res.append(
            {'position': (ind_row, ind_col), 'cost': cost[ind_row][ind_col], 'allocation': min(supply[ind_row], temp)})
        res_sum += cost[ind_row][ind_col] * min(supply[ind_row], temp)
        supply[ind_row] -= min(supply[ind_row], temp)
    for i in range(len(supply)):
        if supply[i] > 0:
            for j in range(len(demand)):
                if demand[j] > 0:
                    temp = demand[j]
                    demand[j] -= min(supply[i], temp)
                    res.append({'position': (i, j), 'cost': cost[i][j], 'allocation': min(supply[i], temp)})
                    res_sum += cost[i][j] * min(supply[i], temp)
                    supply[i] -= min(supply[i], temp)
    return res, res_sum

def find_min_index(matrix):
    min_value = float('inf')
    min_index = (-1, -1)
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            if matrix[i][j] < min_value:
                min_value = matrix[i][j]
                min_index = (i, j)

    return min_index


def russel_approximation(supply, cost, demand):
    res = []
    res_sum = 0
    while any(supply) and any(demand):
        max_cols_matrix = []
        max_row_matrix = []
        dif_matrix = [[0 for _ in range(len(cost[0]))] for _ in range(len(cost))]
        for i in cost:
            max_row_matrix.append(max(i))
        for i in [*zip(*cost)]:
            max_cols_matrix.append(max(i))
        for i in range(len(cost)):
            for j in range(len(cost[0])):
                if cost[i][j] != -1:
                    dif_matrix[i][j] = cost[i][j] - max_cols_matrix[j] - max_row_matrix[i]
        min_index = find_min_index(dif_matrix)
        i, j = min_index
        allocation = min(supply[i], demand[j])
        res.append({"position": min_index, "cost": cost[i][j], "allocation": allocation})
        res_sum += allocation * cost[i][j]
```

```python
        supply[i] -= allocation
        demand[j] -= allocation
        if supply[i] == 0:
                for k in range(len(cost[0])):
                    cost[i][k] = -1
        if demand[j] == 0:
                for k in range(len(cost)):
                    cost[k][j] = -1
    return res, res_sum



print("Enter coefficients of supply for each of 3 sources:")
supply_vector = [int(input()) for _ in range(3)]

print("Enter 3 by 4 matrix of coefficients of costs:")
cost_matrix = []
for i in range(3):
    row = list(map(int, input().split()))
    cost_matrix.append(row) if len(row) == 4 else not_applicable()

print("Enter coefficients of demand for each of 4 destinations:")
demand_vector = [int(input()) for _ in range(4)]

if sum(supply_vector) != sum(demand_vector):
    not_applicable()

print("North-West", north_west(supply_vector.copy(), cost_matrix.copy(), demand_vector.copy()), '\n')
print("Vogel's approximation", vogel_approximation(supply_vector.copy(), cost_matrix.copy(), demand_vector.copy()), '\n')
print("Russel's approximation", russel_approximation(supply_vector.copy(), cost_matrix.copy(), demand_vector.copy()))
```

Listing 1: Transportation Model Python