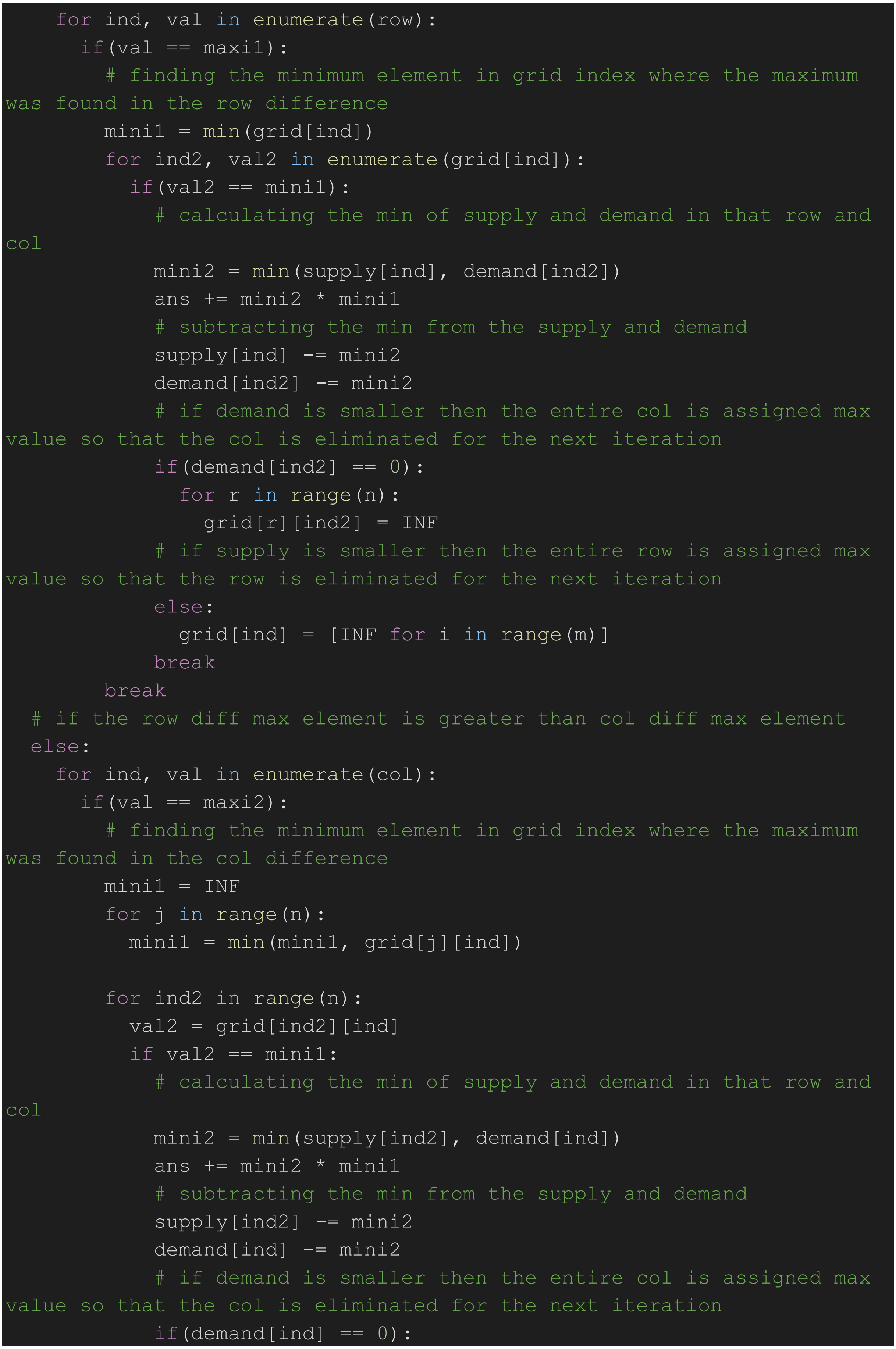
Optimization Technique Lab

Vogel’s Approximation Method

Code:

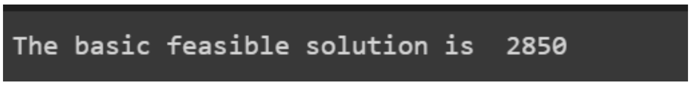
```
grid = [[3, 1, 7, 4], [2, 6, 5, 9], [8, 3, 3, 2]] # table
supply = [300, 400, 500] # supply
demand = [250, 350, 400, 200] # demand
INF = 10**3
n = len(grid)
m = len(grid[0])
ans = 0

# hepler function for finding the row difference and the column
difference
def findDiff(grid):
  rowDiff = []
  colDiff = []
  for i in range(len(grid)):
    arr = grid[i][:]
    arr.sort()
    rowDiff.append(arr[1]-arr[0])
  col = 0
  while col < len(grid[0]):
    arr = []
    for i in range(len(grid)):
      arr.append(grid[i][col])
    arr.sort()
    col += 1
    colDiff.append(arr[1]-arr[0])
  return rowDiff, colDiff


# loop runs until both the demand and the supply is exhausted
while max(supply) != 0 or max(demand) != 0:
  # finding the row and col difference
  row, col = findDiff(grid)
  # finding the maxiumum element in row difference array
  maxi1 = max(row)
  # finding the maxiumum element in col difference array
  maxi2 = max(col)

  # if the row diff max element is greater than or equal to col diff
max element
  if(maxi1 >= maxi2):
```



```
    for ind, val in enumerate(row):
      if(val == maxi1):
        # finding the minimum element in grid index where the maximum
was found in the row difference
        mini1 = min(grid[ind])
        for ind2, val2 in enumerate(grid[ind]):
          if(val2 == mini1):
            # calculating the min of supply and demand in that row and
col
            mini2 = min(supply[ind], demand[ind2])
            ans += mini2 * mini1
            # subtracting the min from the supply and demand
            supply[ind] -= mini2
            demand[ind2] -= mini2
            # if demand is smaller then the entire col is assigned max
value so that the col is eliminated for the next iteration
            if(demand[ind2] == 0):
              for r in range(n):
                grid[r][ind2] = INF
            # if supply is smaller then the entire row is assigned max
value so that the row is eliminated for the next iteration
            else:
              grid[ind] = [INF for i in range(m)]
            break
        break
  # if the row diff max element is greater than col diff max element
  else:
    for ind, val in enumerate(col):
      if(val == maxi2):
        # finding the minimum element in grid index where the maximum
was found in the col difference
        mini1 = INF
        for j in range(n):
          mini1 = min(mini1, grid[j][ind])

        for ind2 in range(n):
          val2 = grid[ind2][ind]
          if val2 == mini1:
            # calculating the min of supply and demand in that row and
col
            mini2 = min(supply[ind2], demand[ind])
            ans += mini2 * mini1
            # subtracting the min from the supply and demand
            supply[ind2] -= mini2
            demand[ind] -= mini2
            # if demand is smaller then the entire col is assigned max
value so that the col is eliminated for the next iteration
            if(demand[ind] == 0):
```
"

```
              for r in range(n):
                grid[r][ind] = INF
            # if supply is smaller then the entire row is assigned max
value so that the row is eliminated for the next iteration
            else:
              grid[ind2] = [INF for i in range(m)]
            break
        break

print("The basic feasible solution is ", ans)
```

Output :

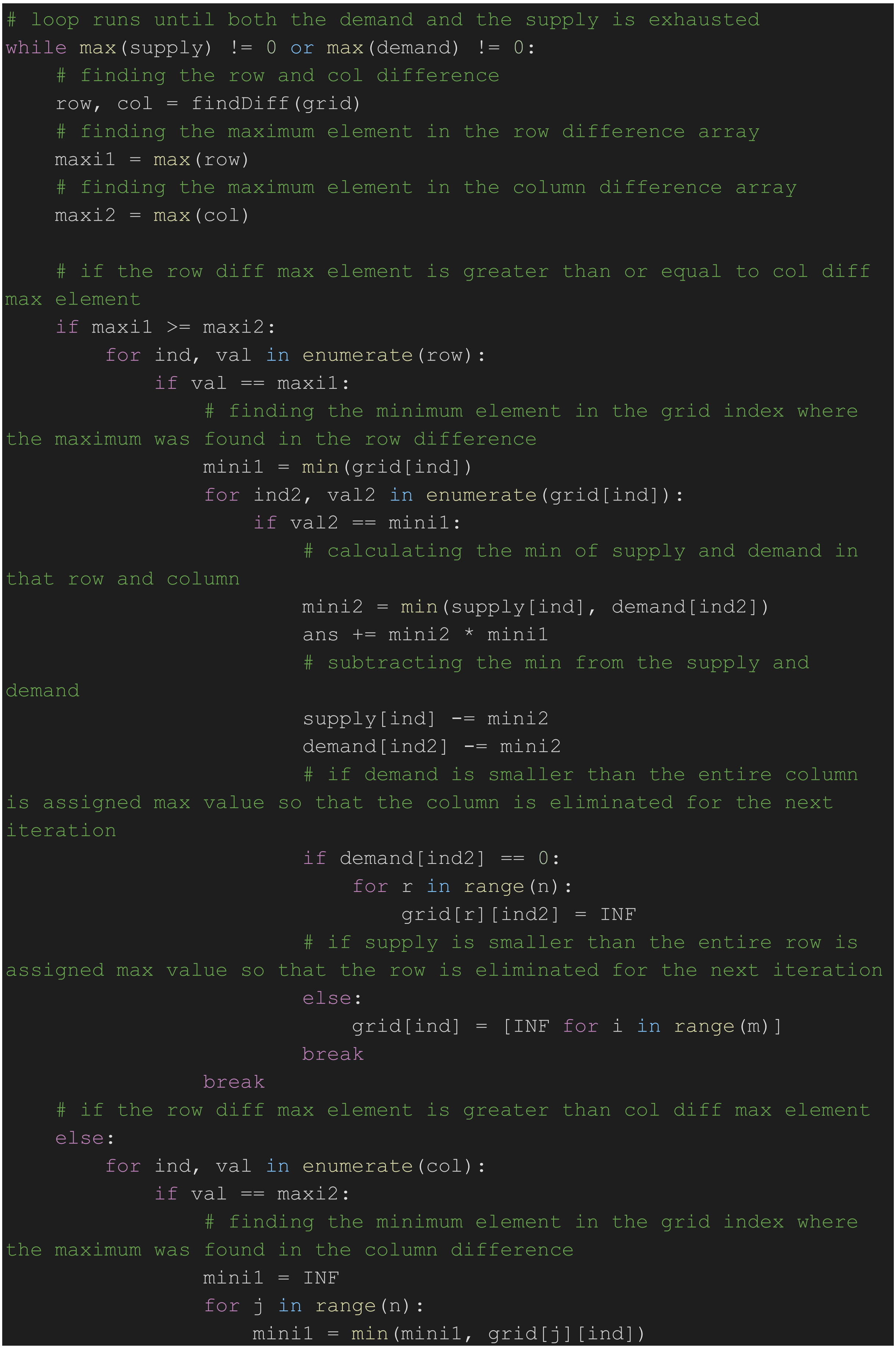


```
The basic feasible solution is  2850
```

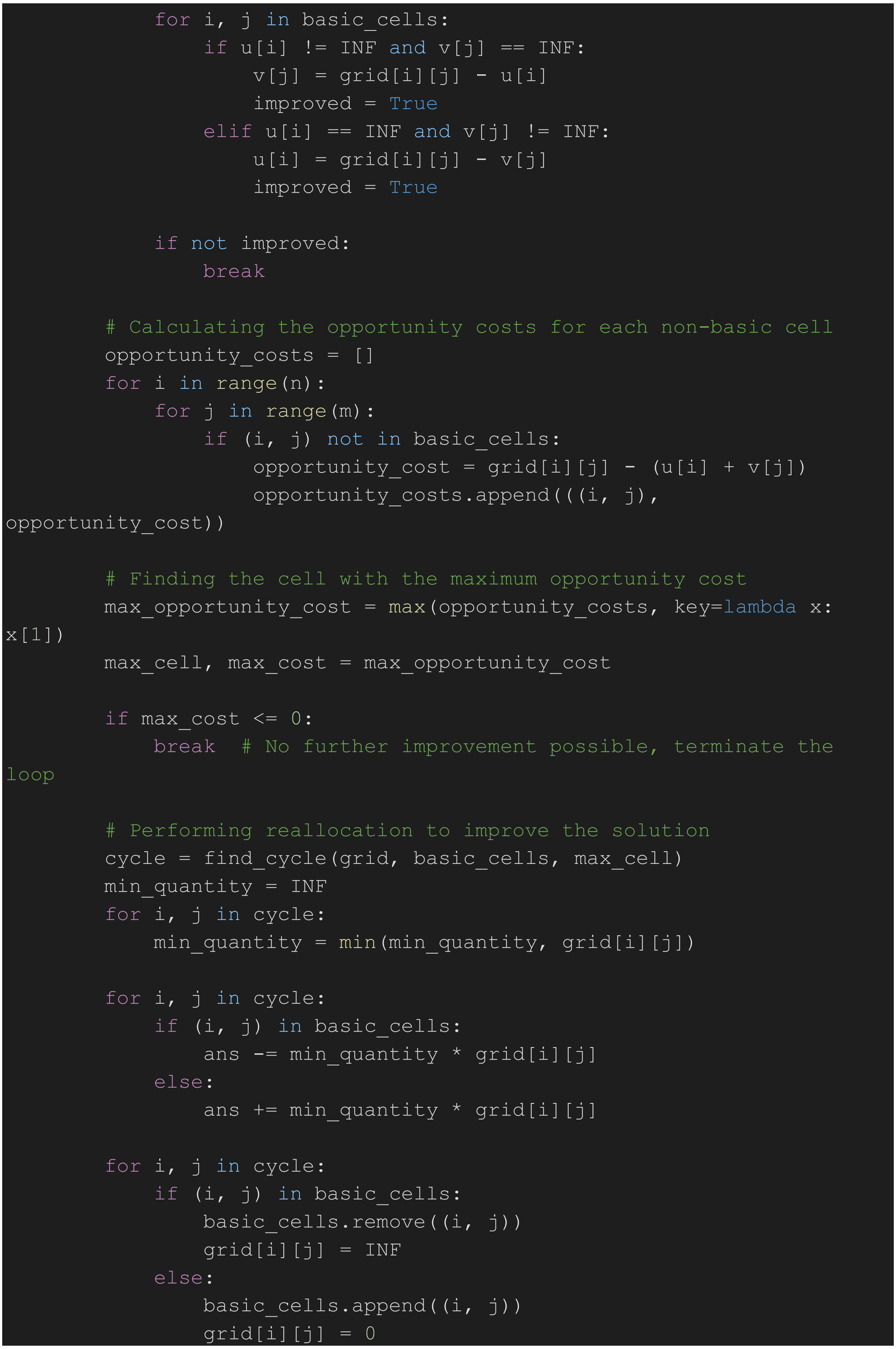
Modi Method :

Code:

```
grid = [[3, 1, 7, 4], [2, 6, 5, 9], [8, 3, 3, 2]]  # table
supply = [300, 400, 500]  # supply
demand = [250, 350, 400, 200]  # demand
INF = 10 ** 3
n = len(grid)
m = len(grid[0])
ans = 0


# helper function for finding the row difference and the column
difference
def findDiff(grid):
    rowDiff = []
    colDiff = []
    for i in range(len(grid)):
        arr = grid[i][:]
        arr.sort()
        rowDiff.append(arr[1] - arr[0])
    col = 0
    while col < len(grid[0]):
        arr = []
        for i in range(len(grid)):
            arr.append(grid[i][col])
        arr.sort()
        col += 1
        colDiff.append(arr[1] - arr[0])
    return rowDiff, colDiff
```



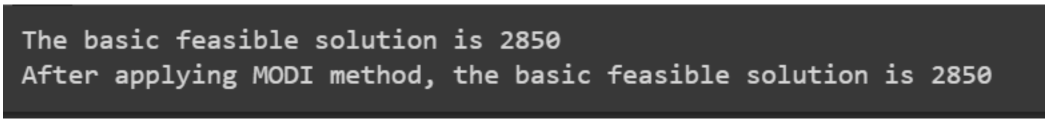
```
# loop runs until both the demand and the supply is exhausted
while max(supply) != 0 or max(demand) != 0:
    # finding the row and col difference
    row, col = findDiff(grid)
    # finding the maximum element in the row difference array
    maxi1 = max(row)
    # finding the maximum element in the column difference array
    maxi2 = max(col)

    # if the row diff max element is greater than or equal to col diff max element
    if maxi1 >= maxi2:
        for ind, val in enumerate(row):
            if val == maxi1:
                # finding the minimum element in the grid index where the maximum was found in the row difference
                mini1 = min(grid[ind])
                for ind2, val2 in enumerate(grid[ind]):
                    if val2 == mini1:
                        # calculating the min of supply and demand in that row and column
                        mini2 = min(supply[ind], demand[ind2])
                        ans += mini2 * mini1
                        # subtracting the min from the supply and demand
                        supply[ind] -= mini2
                        demand[ind2] -= mini2
                        # if demand is smaller than the entire column is assigned max value so that the column is eliminated for the next iteration
                        if demand[ind2] == 0:
                            for r in range(n):
                                grid[r][ind2] = INF
                        # if supply is smaller than the entire row is assigned max value so that the row is eliminated for the next iteration
                        else:
                            grid[ind] = [INF for i in range(m)]
                        break
                break
    # if the row diff max element is greater than col diff max element
    else:
        for ind, val in enumerate(col):
            if val == maxi2:
                # finding the minimum element in the grid index where the maximum was found in the column difference
                mini1 = INF
                for j in range(n):
                    mini1 = min(mini1, grid[j][ind])
                for ind2 in range(n):
                    val2 = grid[ind2][ind]
                    if val2 == mini1:
                        # calculating the min of supply and demand in
that row and column
                        mini2 = min(supply[ind2], demand[ind])
                        ans += mini2 * mini1
                        # subtracting the min from the supply and
demand
                        supply[ind2] -= mini2
                        demand[ind] -= mini2
                        if demand[ind] == 0:
                            for r in range(n):
                                grid[r][ind] = INF
                        # if supply is smaller than the entire row is
assigned max value so that the row is eliminated for the next iteration
                        else:
                            grid[ind2] = [INF for i in range(m)]
                        break
                break

print("The basic feasible solution is", ans)

# MODI method to improve the initial solution
def modi_method(grid, supply, demand, ans):
    n = len(grid)
    m = len(grid[0])
    basic_cells = []  # List to store the indices of basic cells in the
solution

    # Finding the basic cells in the initial solution
    for i in range(n):
        for j in range(m):
            if grid[i][j] != INF:
                basic_cells.append((i, j))

    while True:
        # Constructing the dual matrix for opportunity cost
calculations
        u = [INF for _ in range(n)]
        v = [INF for _ in range(m)]
        u[0] = 0  # Arbitrarily set the first supply constraint to 0

        # Applying the stepping-stone method to calculate dual
variables
        while True:
            improved = False
```



```
            for i, j in basic_cells:
                if u[i] != INF and v[j] == INF:
                    v[j] = grid[i][j] - u[i]
                    improved = True
                elif u[i] == INF and v[j] != INF:
                    u[i] = grid[i][j] - v[j]
                    improved = True

            if not improved:
                break

        # Calculating the opportunity costs for each non-basic cell
        opportunity_costs = []
        for i in range(n):
            for j in range(m):
                if (i, j) not in basic_cells:
                    opportunity_cost = grid[i][j] - (u[i] + v[j])
                    opportunity_costs.append(((i, j),
opportunity_cost))

        # Finding the cell with the maximum opportunity cost
        max_opportunity_cost = max(opportunity_costs, key=lambda x:
x[1])
        max_cell, max_cost = max_opportunity_cost

        if max_cost <= 0:
            break  # No further improvement possible, terminate the
loop

        # Performing reallocation to improve the solution
        cycle = find_cycle(grid, basic_cells, max_cell)
        min_quantity = INF
        for i, j in cycle:
            min_quantity = min(min_quantity, grid[i][j])

        for i, j in cycle:
            if (i, j) in basic_cells:
                ans -= min_quantity * grid[i][j]
            else:
                ans += min_quantity * grid[i][j]

        for i, j in cycle:
            if (i, j) in basic_cells:
                basic_cells.remove((i, j))
                grid[i][j] = INF
            else:
                basic_cells.append((i, j))
                grid[i][j] = 0
```

```python
    return ans


# Helper function to find a cycle in the solution
def find_cycle(grid, basic_cells, start_cell):
    cycle = [start_cell]
    while True:
        curr_cell = cycle[-1]
        for i, j in basic_cells:
            if (i, j) != curr_cell and (i == curr_cell[0] or j == curr_cell[1]):
                cycle.append((i, j))
                if cycle.count((i, j)) > 1:
                    return cycle[cycle.index((i, j)):]
                break

    return cycle


initial_solution = ans
# Improve the solution using the MODI method
improved_solution = modi_method(grid, supply, demand, initial_solution)
print("After applying MODI method, the basic feasible solution is", improved_solution)
```

Output :



```
The basic feasible solution is 2850
After applying MODI method, the basic feasible solution is 2850
```