SATYAJIT DAS

16IT8020

import numpy as np

if \_\_name\_\_ == "\_\_main\_\_":

print("Enter the number of sources: ")

sources = int(input().strip())

print("Enter the number of destinations: ")

destins = int(input().strip())

cost = np.zeros((sources, destins))

print("Enter the cost matrix: ")

for i in range(sources):

for j in range(destins):

cost[i][j] = int(input().strip())

supply = []

demand = []

tot\_supply = 0

tot\_demand = 0

print("Enter the supply capacaties: ")

for i in range(sources):

supply.append(int(input().strip()))

tot\_supply = tot\_supply + supply[i]

print("Enter the demands: ")

for j in range(destins):

demand.append(int(input().strip()))

tot\_demand = tot\_demand + demand[j]

if tot\_demand != tot\_supply:

print("The model is not balanced!")

exit()

i = 0

j = 0

allocations = np.zeros((sources, destins))

tot\_cost = 0

while(i < sources and j < destins):

allocate = min(supply[i], demand[j])

allocations[i][j] = allocate

supply[i] = supply[i] - allocate

demand[j] = demand[j] - allocate

tot\_cost = tot\_cost + allocate\*cost[i][j]

if supply[i] == 0:

i = i + 1

elif demand[j] == 0:

j = j + 1

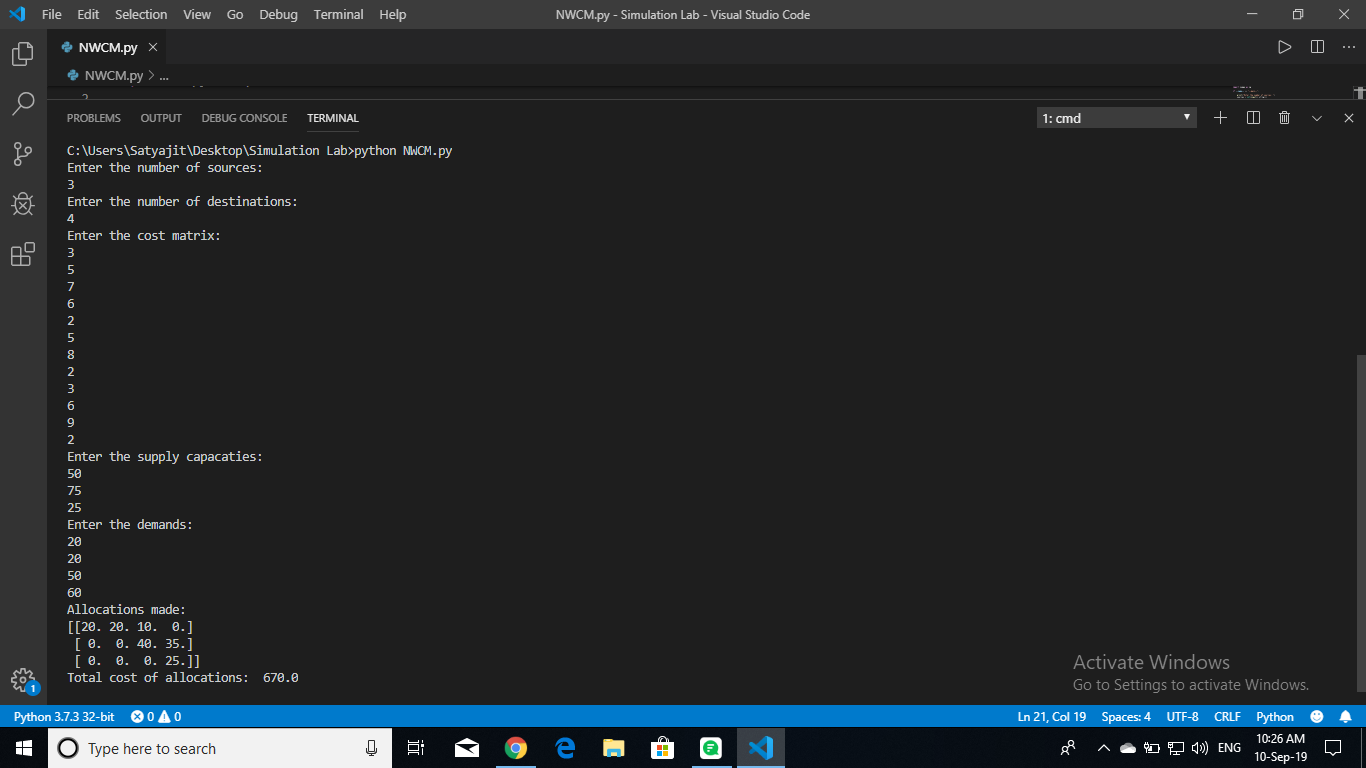
print("Allocations made: ")

print(allocations)

print("Total cost of allocations: ", tot\_cost)

SATYAJIT DAS

16IT8020



SATYAJIT DAS

16IT8020

import numpy as np

def find\_min(allocations, cost, supply, demand):

A\_i = -1

A\_j = -1

min\_cost = 100000

allocate = 0

for i in range(len(supply)):

for j in range(len(demand)):

if demand[j] == 0 or supply[i] == 0 or allocations[i][j] != 0:

continue

if cost[i][j] < min\_cost:

min\_cost = cost[i][j]

A\_i = i

A\_j = j

allocate = min(supply[i], demand[j])

elif cost[i][j] == min\_cost and allocate < min(supply[i], demand[j]):

allocate = min(supply[i], demand[j])

A\_i = i

A\_j = j

return [A\_i, A\_j]

if \_\_name\_\_ == "\_\_main\_\_":

print("Enter the number of sources: ")

sources = int(input().strip())

print("Enter the number of destinations: ")

destins = int(input().strip())

cost = np.zeros((sources, destins))

print("Enter the cost matrix: ")

for i in range(sources):

for j in range(destins):

cost[i][j] = int(input().strip())

supply = []

demand = []

tot\_supply = 0

tot\_demand = 0

print("Enter the supply capacaties: ")

for i in range(sources):

supply.append(int(input().strip()))

tot\_supply = tot\_supply + supply[i]

print("Enter the demands: ")

for j in range(destins):

demand.append(int(input().strip()))

tot\_demand = tot\_demand + demand[j]

if tot\_demand != tot\_supply:

print("The model is not balanced!")

exit()

allocations = np.zeros((sources, destins))

tot\_cost = 0

while tot\_demand > 0 and tot\_supply > 0:

idx = find\_min(allocations, cost, supply, demand)

allocations[idx[0]][idx[1]] = min(supply[idx[0]], demand[idx[1]])

tot\_cost = tot\_cost + allocations[idx[0]][idx[1]]\*cost[idx[0]][idx[1]]

supply[idx[0]] = supply[idx[0]] - allocations[idx[0]][idx[1]]

demand[idx[1]] = demand[idx[1]] - allocations[idx[0]][idx[1]]

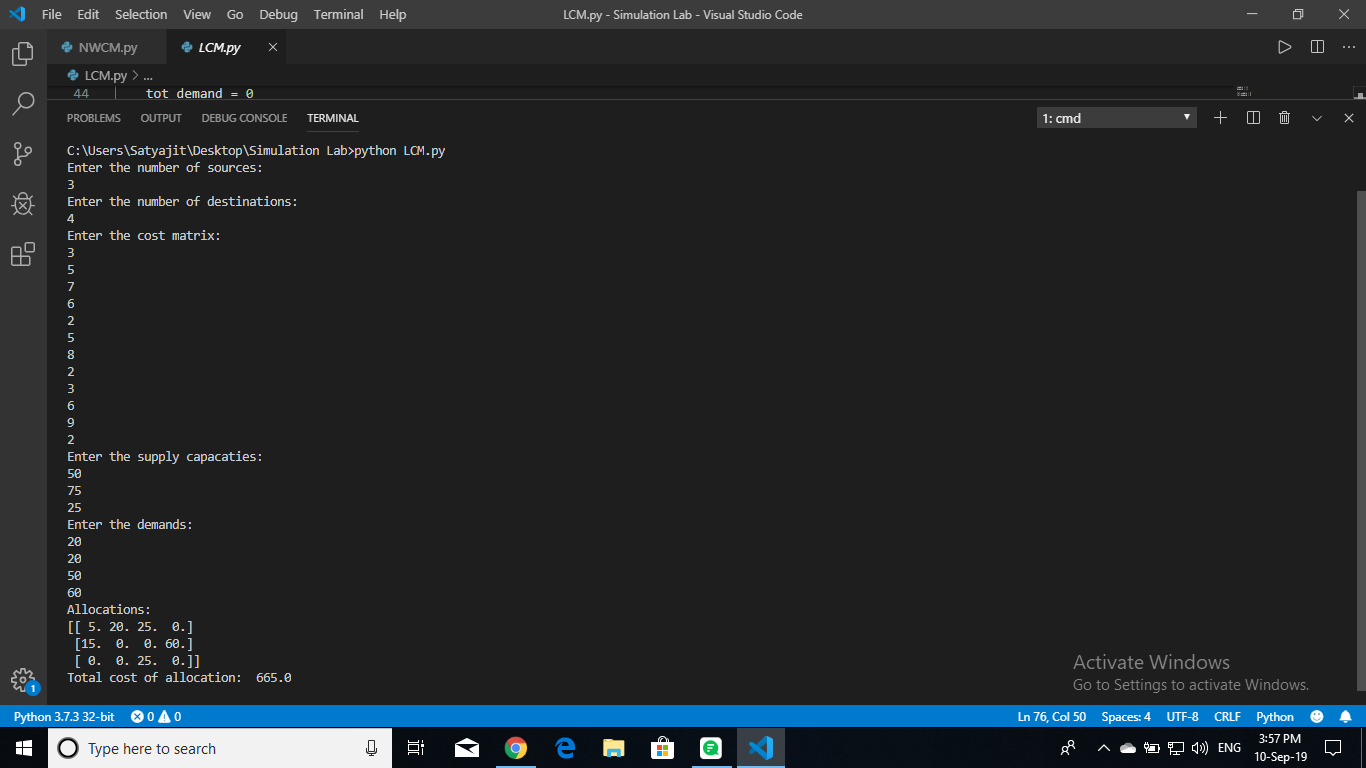
tot\_demand = tot\_demand - allocations[idx[0]][idx[1]]

tot\_supply = tot\_supply - allocations[idx[0]][idx[1]]

print("Allocations: ")

print(allocations)

print("Total cost of allocation: ", tot\_cost)



SATYAJIT DAS

16IT8020

SATYAJIT DAS

16IT8020

import numpy as np

def min\_row(mat, row, n, penality\_col):

    idx = -1

    val = 100000

    for i in range(n):

        if mat[row][i] < val and penality\_col[i] != -1:

            val = mat[row][i]

            idx = i

    return idx

def min\_2\_row(mat, row, n, min\_idx, penality\_col):

    idx = -1

    val = 100000

    for i in range(n):

        if mat[row][i] <= val and i != min\_idx and penality\_col[i] != -1:

            val = mat[row][i]

            idx = i

    return idx

def min\_col(mat, col, m, penality\_row):

    idx = -1

    val = 100000

    for i in range(m):

        if mat[i][col] < val and penality\_row[i] != -1:

            val = mat[i][col]

            idx = i

    return idx

def min\_2\_col(mat, col, m, min\_idx, penality\_row):

    idx = -1

    val = 100000

    for i in range(m):

        if mat[i][col] <= val and i != min\_idx and penality\_row[i] != -1:

            val = mat[i][col]

            idx = i

    return idx

def max\_p(penality):

    idx = 0

    val = penality[0]

    for i in range(1, len(penality)):

        if val < penality[i]:

            idx = i

            val = penality[i]

    return idx

def tot\_cost(alloc, cost, m, n):

    t\_cost = 0

    for i in range(m):

        for j in range(n):

            t\_cost += alloc[i][j] \* cost[i][j]

    return t\_cost

if \_\_name\_\_ == "\_\_main\_\_":

    print("Enter no. of sources and destinations: ")

    m, n = input().strip().split(' ')

    m = int(m)

    n = int(n)

    mat = np.zeros((m,n))

    alloc = np.zeros((m,n))

    print("Enter cost matrix: ")

    for i in range(m):

        for j in range(n):

            mat[i][j] = int(input())

    demand = []

    t\_sum = 0

    print("Enter the demand: ")

    for i in range(n):

        demand.append(int(input()))

        t\_sum += demand[i]

    supply = []

    print("Enter the supply: ")

    for i in range(m):

        supply.append(int(input()))

        t\_sum -= supply[i]

    if t\_sum != 0:

        print("Unbalanced problem")

        exit()

    penality\_row = [0 for \_ in range(m)]

    penality\_col = [0 for \_ in range(n)]

    loop = m + n - 1

    while(loop):

        # row

        for i in range(m):

            if penality\_row[i] == -1:

                continue

            min\_idx = min\_row(mat, i, n, penality\_col)

            min\_2\_idx = min\_2\_row(mat, i, n, min\_idx, penality\_col)

            if min\_2\_idx == -1:

                penality\_row[i] =  mat[i][min\_idx]

            else:

                penality\_row[i] = mat[i][min\_2\_idx] - mat[i][min\_idx]

        # column

        for j in range(n):

            if penality\_col[j] == -1:

                continue

            min\_idx = min\_col(mat, j, m, penality\_row)

            min\_2\_idx = min\_2\_col(mat, j, m, min\_idx, penality\_row)

            if min\_2\_idx == -1:

                penality\_col[j] = mat[min\_idx][j]

            else:

                penality\_col[j] = mat[min\_2\_idx][j] - mat[min\_idx][j]

        r\_p = max\_p(penality\_row)

        c\_p = max\_p(penality\_col)

        if(penality\_row[r\_p] >= penality\_col[c\_p]):

            r\_min = min\_row(mat, r\_p, n, penality\_col)

            alloc[r\_p][r\_min] = min(supply[r\_p], demand[r\_min])

            supply[r\_p] -= alloc[r\_p][r\_min]

            demand[r\_min] -= alloc[r\_p][r\_min]

            if supply[r\_p] == 0:

                penality\_row[r\_p] = -1

            elif demand[r\_min] == 0:

                penality\_col[r\_min] = -1

        else:

            c\_min = min\_col(mat, c\_p, m, penality\_row)

            alloc[c\_min][c\_p] = min(supply[c\_min], demand[c\_p])

            supply[c\_min] -= alloc[c\_min][c\_p]

            demand[c\_p] -= alloc[c\_min][c\_p]

            if supply[c\_min] == 0:

                penality\_row[c\_min] = -1

            elif demand[c\_p] == 0:

                penality\_col[c\_p] = -1

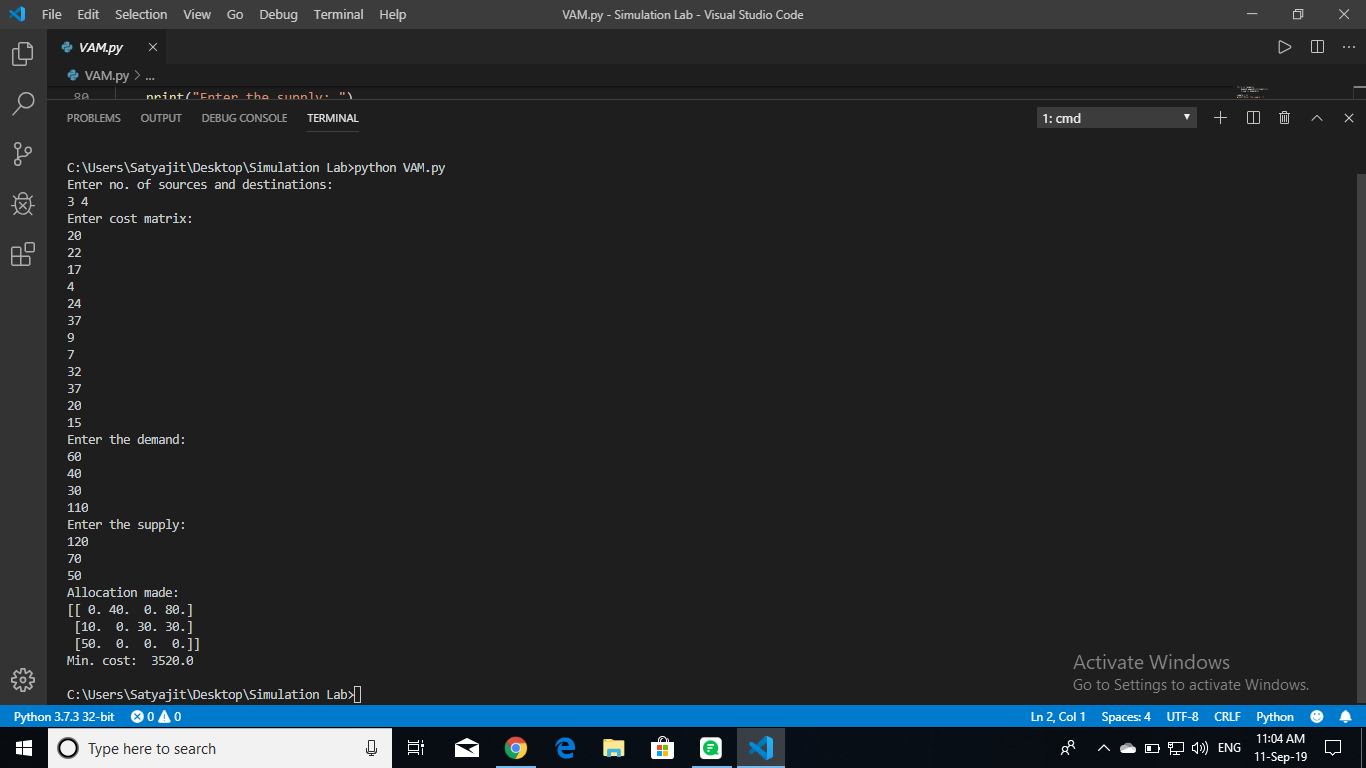
        loop -= 1

    print("Allocation made: ")

    print(alloc)

    print("Min. cost: ", tot\_cost(alloc, mat, m, n))

SATYAJIT DAS

16IT8020

SATYAJIT DAS

16IT8020

import numpy as np

from math import isnan

def get\_an\_occupied\_cell(allocs, i, j):

    try:

        if (allocs[i][j + 1] != 0):

            return i, j+1

    except IndexError:

        pass

    try:

        if (allocs[i+1][j] != 0):

            return i+1, j

    except IndexError:

        pass

    try:

        if (allocs[i-1][j] != 0):

            return i-1, j

    except IndexError:

        pass

    try:

        if (allocs[i][j-1] != 0):

            return i, j-1

    except IndexError:

        pass

    return -1, -1

def create\_cycle(allocs, neg\_i, neg\_j):

    cycle\_coordinates = []

    i, j = get\_an\_occupied\_cell(allocs, neg\_i, neg\_j)

    print("@", i, j)

    while i != neg\_i or j != neg\_j:

        if i == -1 or j == -1:

            raise Exception("Couldn't find another cell")

        cycle\_coordinates.append((i, j))

        i, j = get\_an\_occupied\_cell(allocs, i, j)

    print("%", cycle\_coordinates)

    return cycle\_coordinates

def get\_min\_in\_cycle(allocs, cycle\_coordinates):

    min = 0

    for i in cycle\_coordinates:

        if min > allocs[i[0]][i[1]]:

            min = allocs[i[0]][i[1]]

    return min

def modify\_cycle(allocs, cycle\_coordinates):

    t = 1

    min = get\_min\_in\_cycle(allocs, cycle\_coordinates)

    for x, y in cycle\_coordinates:

        allocs[x][y] = min + int(t \* allocs[x][y])

        if t == 1:

            t = -1

        else:

            t = 1

def get\_all\_allocated\_coordinates(allocs):

    coordinates = []

    for i in range(len(allocs)):

        for j in range(len(allocs[0])):

            if allocs[i][j] > 0:

                coordinates.append((i, j))

    return coordinates

def isAnyNan(l):

    for i in l:

        if isnan(i):

            return True

    return False

def get\_uv(costs, allocs):

    u = []

    v = []

    for \_ in costs:

        u.append(float('nan'))

    for \_ in costs[0]:

        v.append(float('nan'))

    u[0] = 0

    coordinates = get\_all\_allocated\_coordinates(allocs)

    stopLoopCount = 0

    while isAnyNan(u) or isAnyNan(v):

        stopLoopCount += 1

        for x, y in coordinates:

            if not isnan(u[x]) and isnan(v[y]):

                v[y] = costs[x][y] - u[x]

            elif isnan(u[x]) and not isnan(v[y]):

                u[x] = costs[x][y] - v[y]

        if stopLoopCount % 100 == 0:

            ans = input("Loop ran for {0} times. Do you want to continue? [y] ".format(stopLoopCount))

            if ans != 'y':

                break

    return u, v

def isValid(costs, allocs, u, v):

    min\_val = 1000000000000000

    min\_x = -1

    min\_y = -1

    #print(u, v)

    for i in range(len(costs)):

        for j in range(len(costs[0])):

            if allocs[i][j] == 0:

                x = costs[i][j] - u[i] - v[j]

                #print('#', x, i, j)

                if x < min\_val:

                    min\_val = x

                    min\_x, min\_y = i, j

    if (min\_val < 0):

        return min\_x, min\_y

    else:

        return -1, -1

if \_\_name\_\_ == '\_\_main\_\_':

    costs = []

    allocs = []

    rows\_count = int(input("Enter rows: "))

    print("Enter the cost matrix: ")

    for i in range(rows\_count):

        costs.append(list(map(int, input().strip().split(' '))))

    print("Enter the allocation matrix: ")

    for i in range(rows\_count):

        allocs.append(list(map(int, input().strip().split(' '))))

    u, v = get\_uv(costs, allocs)

    x, y = isValid(costs, allocs, u, v)

    while x != -1 and y != -1:

        cycle\_coordinates = create\_cycle(allocs, x, y)

        modify\_cycle(allocs, cycle\_coordinates)

        print("  ##  ")

        print(u, v)

        print(x, y)

        print(cycle\_coordinates)

        print(allocs)

        print("  ##  ")

        ans = input("Continue? [y]")

        if ans != 'y':

            break

        u, v = get\_uv(costs, allocs)

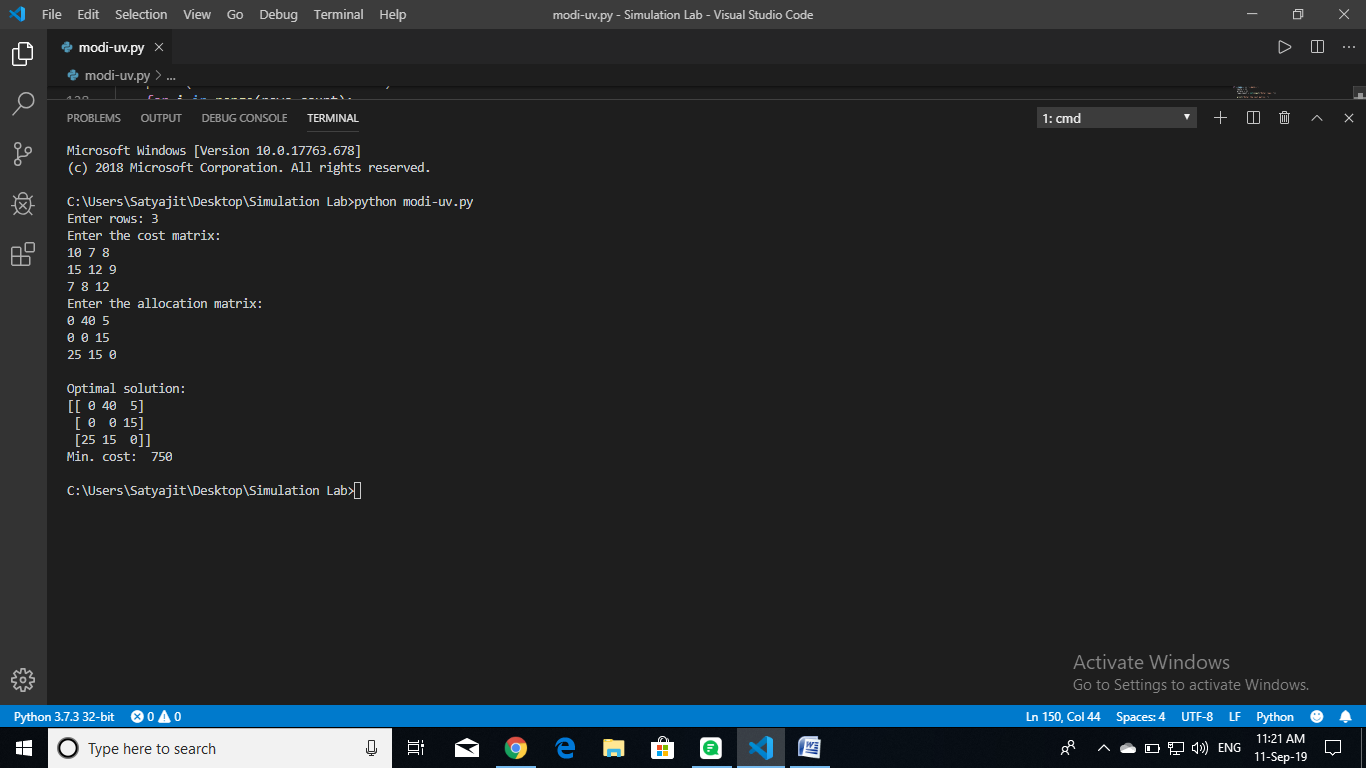
        x, y = isValid(costs, allocs, u, v)

    print("\nOptimal solution: ")

    print(np.array(allocs))

print("Min. cost: ", tot\_cost(allocs, costs, rows\_count, len(costs[0])))

SATYAJIT DAS

16IT8020