Student-id : 201801015
Language Used: Python(Jupyter Notebook)
Code: If YOU COPY-PASTE PYTHON CODE THEN INDENTATION WILL BE RUINED AND CODE WONT RUN BECAUSE THE INDENTATION IS NOT COPIED. SO PLEASE USE THE IPYNB FILE ATTACHED AT THE END.

```
from collections import defaultdict
import numpy
import sys

# INPUTS
r, c = input("Enter no. of Rows and Columns seperated by a space:").split(" ")
r, c = int(r), int(c)
col_i=[]
row_i=[]
for i in range(c):
    col_i.append('D'+str
                 (int(i)+1))
for i in range(r):
    row_i.append('S'+str(int(i)+1))

# Cost values
mat = numpy.zeros((r,c))
trial = [dict() for x in range(r)]
trial_l=[]
another_c=[]
current_net_cost={}
temp=[]
for i in range(r):
    temp=input("Enter " + str(c) + " values for Row " +str(i+1) +" seperated by a space:").split(" ")
    k=0
    for j in range(c):
        mat[i][j] = temp[j]
        trial[i][col_i[j]]=int(temp[j])
        trial_l.append(int(temp[j]))
    current_net_cost[row_i[i]]=trial[i]
    another_c.append(trial_l)
    trial_l=[]
    temp.clear()

# Ai values
current_supply={}
another_s=[]
temp=input("Enter " + str(r) + " Ai values seperated by a space:").split(" ")
for i in range(r):
    current_supply[row_i[i]]=int(temp[i])
    another_s.append(int(temp[i]))
temp.clear()
```

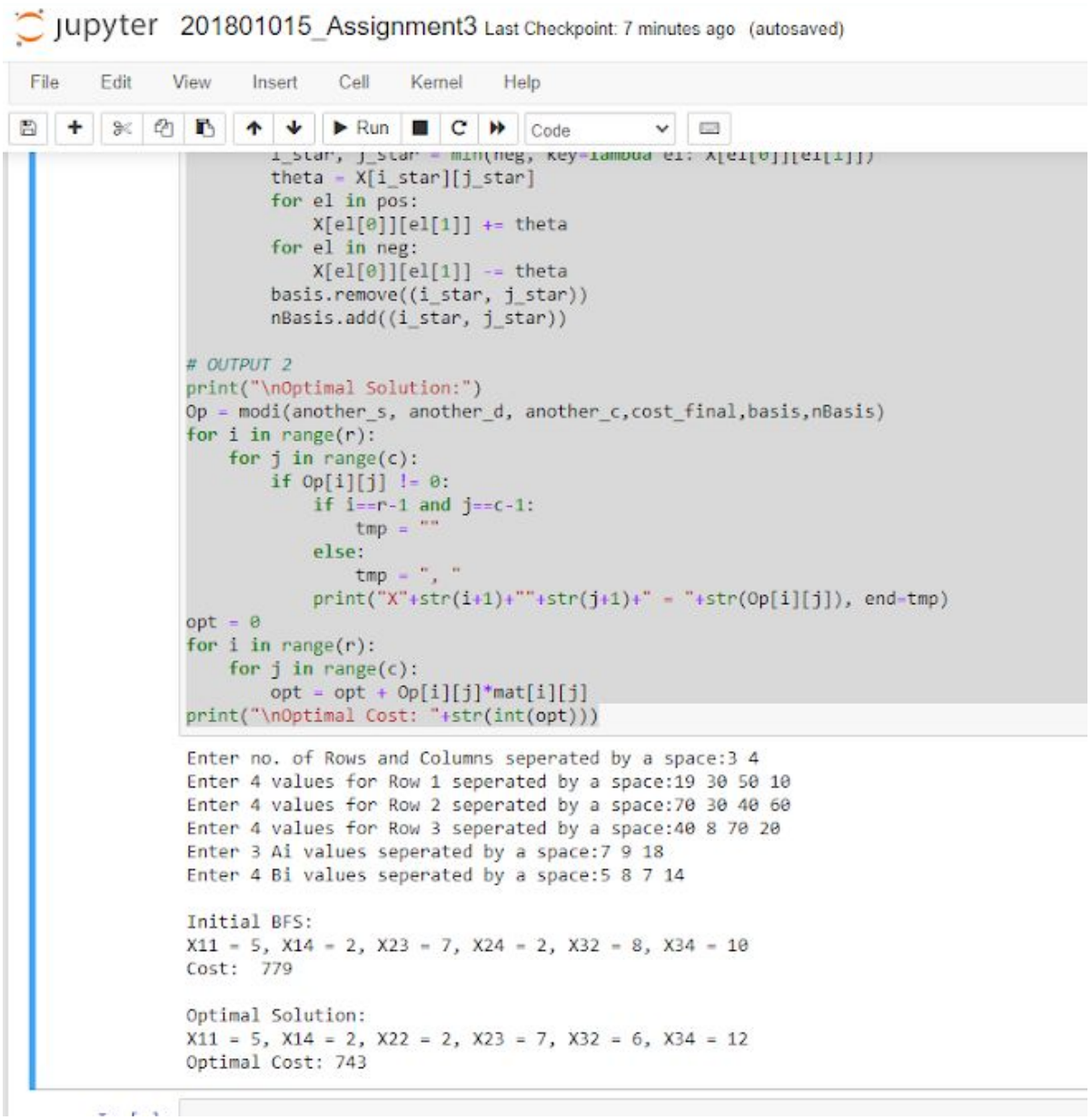
```
# Bi values
current_demand={}
another_d=[]
temp=input("Enter " + str(c) + " Bi values seperated by a space:").split(" ")
i=0
for i in range(c):
    current_demand[col_i[i]]=int(temp[i])
    another_d.append(int(temp[i]))
temp.clear()
col = sorted(current_demand.keys())

cDict = dict((k, defaultdict(int)) for k in current_net_cost)
g = {}
for x in current_supply:
    g[x] = sorted(current_net_cost[x].keys(), key=lambda g: current_net_cost[x][g])
for x in current_demand:
    g[x] = sorted(current_net_cost.keys(), key=lambda g: current_net_cost[g][x])

# VAN METHOD TO FIND INITIAL BASIC FEASIBLE SOLUTION
while g:
    d = {}
    for x in current_demand:
        d[x] = (current_net_cost[g[x][1]][x] - current_net_cost[g[x][0]][x]) if len(g[x]) > 1 else
current_net_cost[g[x][0]][x]
    s = {}
    for x in current_supply:
        s[x] = (current_net_cost[x][g[x][1]] - current_net_cost[x][g[x][0]]) if len(g[x]) > 1 else
current_net_cost[x][g[x][0]]
    f = max(d, key=lambda n: d[n])
    t = max(s, key=lambda n: s[n])
    t, f = (f, g[f][0]) if d[f] > s[t] else (g[t][0], t)
    v = min(current_supply[f], current_demand[t])
    cDict[f][t] += v
    current_demand[t] -= v
    if current_demand[t] == 0:
        for k, n in current_supply.items():
            if n != 0:
                g[k].remove(t)
        del g[t]
        del current_demand[t]
    current_supply[f] -= v
    if current_supply[f] == 0:
        for k, n in current_demand.items():
            if n != 0:
                g[k].remove(f)
        del g[f]
        del current_supply[f]
basis = set()
nBasis=set()
checkB=[]
```

```
cost_final=[]
cost = 0
for g in sorted(current_net_cost):
    trial=[]
    temp_m=[]
    for n in col:
        trial=[]
        y = cDict[g][n]
        temp_m.append(int(y))
        if y != 0:
            trial.append(int(g[-1])-1)
            trial.append(int(n[-1])-1)
            basis.add((int(g[-1])-1, int(n[-1])-1))
            checkB.append(trial)
        else:
            nBasis.add((int(g[-1])-1, int(n[-1])-1))
        cost += y * current_net_cost[g][n]
    cost_final.append(temp_m)

# OUTPUT 1
print("\nInitial BFS:")
for i in range(r):
    for j in range(c):
        if cost_final[i][j] != 0:
            if i==r-1 and j==c-1:
                tmp = ""
            else:
                tmp = ", "
            print("X"+str(i+1)+""+str(j+1)+" = "+str(cost_final[i][j]), end = tmp)
print ("\nCost: ",cost)

# MODI METHOD TO FIND THE OPTIMAL SOLUTION
def cycle(m, n, basis):
    basis = basis.copy()
    row, col = [0]*m, [0]*n
    for i, j in basis:
        row[i] += 1
        col[j] += 1
    while True:
        ex = True
        for k in range(m):
            if row[k] == 1:
                ex = False
                for i, j in basis:
                    if i == k:
                        col[j] -= 1
                        row[i] = 0
                        basis.remove((i, j))
                        break
        for k in range(n):
```

```
            if col[k] == 1:
                ex = False
                for i, j in basis:
                    if j == k:
                        row[i] -= 1
                        col[j] = 0
                        basis.remove((i, j))
                        break
        if ex:
            return basis
        if len(basis) < 4:
            return None

def poten(C, basis):
    inf = float('inf')
    u = [inf]*len(C)
    v = [inf]*len(C[0])
    u[0] = 0
    for _ in range(len(basis)):
        for i, j in basis:
            if v[j] == inf and u[i] != inf:
                v[j] = C[i][j] - u[i]
                break
            elif u[i] == inf and v[j] != inf:
                u[i] = C[i][j] - v[j]
                break
    return u, v

def split(ii, jj, cyc):
    neg, pos = set(), set()
    pos.add((ii, jj))
    for _ in range(len(cyc) >> 1):
        for i, j in cyc:
            if i == ii and j != jj:
                neg_i, neg_j = i, j
                break
        neg.add((neg_i, neg_j))
        for i, j in cyc:
            if j == neg_j and i != neg_i:
                ii, jj = i, j
                break
        pos.add((ii, jj))
    return neg, pos

def modi(a, b, C, X, basis, nBasis):
    m, n = len(a), len(b)
    dif = sum(a) - sum(b)
    if dif < 0:
        a.append(abs(dif))
        m += 1
```

```
        C.append([0 for _ in range(n)])
    elif dif > 0:
        b.append(dif)
        n += 1
        for row in C:
            row.append(0)
    if len(nBasis) == 0:
        return X

    while True:
        u, v = poten(C, basis)
        ii, jj = min(nBasis, key=lambda x: C[x[0]][x[1]]-u[x[0]]-v[x[1]])
        min_d = C[ii][jj]-u[ii]-v[jj]
        if min_d >= 0:
            return X
        basis.add((ii, jj))
        nBasis.remove((ii, jj))
        cyc = cycle(m, n, basis)
        neg, pos = split(ii, jj, cyc)
        i_star, j_star = min(neg, key=lambda el: X[el[0]][el[1]])
        theta = X[i_star][j_star]
        for el in pos:
            X[el[0]][el[1]] += theta
        for el in neg:
            X[el[0]][el[1]] -= theta
        basis.remove((i_star, j_star))
        nBasis.add((i_star, j_star))

# OUTPUT 2
print("\nOptimal Solution:")
Op = modi(another_s, another_d, another_c,cost_final,basis,nBasis)
for i in range(r):
    for j in range(c):
        if Op[i][j] != 0:
            if i==r-1 and j==c-1:
                tmp = ""
            else:
                tmp = ", "
            print("X"+str(i+1)+""+str(j+1)+" = "+str(Op[i][j]), end=tmp)
opt = 0
for i in range(r):
    for j in range(c):
        opt = opt + Op[i][j]*mat[i][j]
print("\nOptimal Cost: "+str(int(opt)))
```

Screenshot of output:



IDE: Google Collab / Jupyter Notebook
GOOGLE COLLAB LINK: COLLAB
AND DIRECTLY RUN IT IN COLLAB
JUPYTER NOTEBOOK CODE FILE: JUPYTER
AND OPEN IT IN JUPYTER NOTEBOOK