Program 9

Aim: Solve a transportation problem of three variables.

Code:

```
clc
clear
prices = [1, 2, 3;
          8, 5, 4;
          3, 1, 6]
demand = [100, 30, 70]
supply = [110, 40, 50]
prices = evstr(x_matrix('setprices', prices));
demand = evstr(x_matrix('setdemand', demand));
supply = evstr(x_matrix('submit offer', supply));
LEFT = 1
RIGHT = 2
UP = 3
DOWN = 4
// The function of calculating the cost of the transferred plan
function res=cost(prices, plan)
    cntCols = length(prices(1,:))
    cntRows = length(prices(:,1))

    res = 0
    for i=1:cntRows
        for j=1:cntCols
            res = res + prices(i,j) * plan(i,j)
        end
    end
endfunction
// a function that looks for available angles in a given direction
// and returns them in descending order of proximity to the edge
function [corners, success]=getAvailableCorner(basis, direction, initialPoint, i, j)
    success = 0
    corners = []
    currentCorner = 1

    cntCols = length(basis(1,:))
    cntRows = length(basis(:,1))

    colModificator = 0;
    rowModificator = 0;

    if direction == LEFT then
        colModificator = -1
    end
```

```
    if direction == RIGHT then
        colModificator = 1
    end
    if direction == UP then
        rowModificator = -1
    end
    if direction == DOWN then
        rowModificator = 1
    end
    i = i + rowModificator
    j = j + colModificator

    while i ~= 0 && j ~= 0 && i <= cntRows && j <= cntCols
        if basis(i,j) ~= 0 || [i, j] == initialPoint then
            corners(currentCorner,:) = [i,j]
            currentCorner = currentCorner + 1
            success = 1
        end

        i = i + rowModificator
        j = j + colModificator
    end

    if success == 1 then
        cornersReverse = []
        for iter = 1:length(corners(:,1))
            cornersReverse(iter,:) = corners(length(corners(:,1)) - iter + 1, :)
        end

        corners = cornersReverse
    end


endfunction
// recursive looping function
function [nodes, success]=buildCycle(basis, initialPoint, currentPoint, direction)
    success = 0
    nodes = []

    possibleDirections = []
    if initialPoint == currentPoint then
        possibleDirections = [LEFT, RIGHT, UP, DOWN]
    else if direction == LEFT || direction == RIGHT then
        possibleDirections = [UP, DOWN]
    else if direction == UP || direction == DOWN then
        possibleDirections = [LEFT, RIGHT]
    end; end; end

    for directionIdx = 1:length(possibleDirections)
```

```
      [corners, suc] = getAvailableCorner(basis, possibleDirections(directionIdx), initialPoint,
currentPoint(1), currentPoint(2))
      if suc == 1 then
         possibleToCloseCycle = 0
         successWithCorners = 0
         for cornIdx = 1:length(corners(:,1))
            if (corners(cornIdx,:) == initialPoint) then
               possibleToCloseCycle = 1
               continue
            end

            [subNodes, suc] = buildCycle(basis, initialPoint, corners(cornIdx,:),
possibleDirections(directionIdx))
            if suc == 1 then
               successWithCorners = 1
               nodeIdx = 1

               nodes(nodeIdx, :) = currentPoint

               for subNodeIdx = 1:length(subNodes(:,1))
                  nodeIdx = nodeIdx + 1
                  nodes(nodeIdx, :) = subNodes(subNodeIdx,:)
               end

               break
            end
         end

         if successWithCorners == 1 then
            success = 1
            break
         else if possibleToCloseCycle == 1 then
            nodes(1, :) = currentPoint
            nodes(2, :) = initialPoint

            success = 1
            break
         end; end
      end
   end
endfunction
cntCols = length(prices(1,:))
cntRows = length(prices(:,1))
plan = [] // reference plan
plan(cntRows, cntCols) = 0 // fill with zeros
// Calculation of the original reference plan using the northwest corner method
tempDemand = demand
tempSupply = supply
for j=1:cntCols // iterate over columns (customers)
   for i=1:cntRows // iterate over rows (suppliers)
```

```
        currentSupply = min(tempDemand(j), tempSupply(i))
        plan(i,j) = currentSupply
        tempDemand(j) = tempDemand(j) - currentSupply
        tempSupply(i) = tempSupply(i) - currentSupply

        if tempDemand(j) == 0 then
            break
        end
    end
end
disp("Initial plan:")
disp(plan)
printf("\nThe cost is %d Ñƒ.Ðµ.\n\n\n", cost(prices, plan))
// Plan optimization
optimal = 0
UNKNOWN_POTENCIAL = 9999999
iteration = 0
while optimal ~= 1
    iteration = iteration + 1
    potencialU = []
    potencialV = []

    for i = 1:cntRows
        potencialU(i) = UNKNOWN_POTENCIAL // type unknown yet potential
    end

    for i = 1:cntCols
        potencialV(i) = UNKNOWN_POTENCIAL
    end

    potencialU(1) = 0

    continuePotentialing  = 1

    // calculation of potentials by points in the route
    while continuePotentialing == 1
        continuePotentialing = 0
        // we continue to calculate the potentials if
        // for one of the values â€‹â€‹of the plan, both potentials are unknown

        for j=1:cntCols // iterate over columns (customers)
            for i=1:cntRows // iterate over rows (suppliers)
                if (plan(i,j) == 0) then
                    continue
                end

                if potencialU(i) == UNKNOWN_POTENCIAL && potencialV(j) ==
UNKNOWN_POTENCIAL then
                    continuePotentialing = 1
                    continue
```

```
            end

            if potencialU(i) == UNKNOWN_POTENCIAL then
                potencialU(i) = prices(i,j) - potencialV(j)
            end

            if potencialV(j) == UNKNOWN_POTENCIAL then
                potencialV(j) = prices(i,j) - potencialU(i)
            end
        end
    end
end

// Calculating estimates for non-basic variables
notBasis = [] // reference plan
notBasis(cntRows, cntCols) = 0 // fill with zeros

optimal = 1
maxI = 0;
maxJ = 0;
maxNB = 0;
for j=1:cntCols // iterate over columns (customers)
    for i=1:cntRows // iterate over rows (suppliers)
        if (plan(i,j) ~= 0) then
            continue
        end

        notBasis(i,j) = potencialU(i) + potencialV(j) - prices(i,j)
        if notBasis(i,j) > 0 then
            optimal = 0
            if maxNB < notBasis(i,j) then
                maxNB = notBasis(i,j)
                maxI = i
                maxJ = j
            end
        end
    end
end

if optimal == 1 then
    printf("Iteration %d. The current plan is optimal!", iteration)
    break
else
    printf("Iteration %d. Current plan is not optimal. Plan optimization", iteration)
end

[nodes, success] = buildCycle(plan, [maxI, maxJ], [maxI, maxJ], "")

if success == 0 then
    disp("Loop building error. Shutdown")
```

```
        break
    end

    // Among the even nodes of the cycle (those who will have a negative 0) looking for the minimum
value
    minNode = 99999999
    for node = 2:2:length(nodes(:,1))
        if minNode > plan(nodes(node, 1), nodes(node, 2)) then
            minNode = plan(nodes(node, 1), nodes(node, 2))
        end
    end

    for node = 2:length(nodes(:,1))
        nodeI = nodes(node, 1)
        nodeJ = nodes(node, 2)

        if modulo(node, 2) == 0 then
            plan(nodeI, nodeJ) = plan(nodeI, nodeJ) - minNode // for even subtract min. meaning
        else
            plan(nodeI, nodeJ) = plan(nodeI, nodeJ) + minNode // for odd ones add min. meaning
        end
    end

    disp("New plan:")
    disp(plan)
    printf("\nThe cost is %d Ñƒ.Ðµ.\n\n\n", cost(prices, plan))
end
tableStr = 2;
table = []
table(1,:) = [" " "From supplier" "To the consumer" "Quantity"];
for i = 1:cntRows
    for j = 1:cntCols
        if plan(i,j) ~= 0 then
            str = []
            str(1) = " "
            str(2:4) = string([i, j, plan(i,j)])
            table(tableStr,:) = str
            tableStr = tableStr + 1
        end
    end
end
disp(table)
f = createWindow();
f.figure_size = [400 400];
f.figure_name = "Final answer";
as = f.axes_size;
ut = uicontrol("style", "table",..
               "string", table,..
               "position", [0 -50 400 400],..
               "tag", "Final answer");
```

```
matrix(ut.string, size(table))
```

Output: