|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 1**

**Вариант 23**

|  |  |
| --- | --- |
| **Тема: Венгерский метод решения задачи о назначениях**  **Студент: Нгуен Н.Х.**  **Группа: ИУ7И-11М**  **Преподаватель: Власов П.А.** |  |

Москва.

2021 г.

# Теоретическая часть

## Содержательная постановка задачи о назначениях

В распоряжении работодателя имеется работ и исполнителей. Стоимость выполнения -й работы -м исполнителем составляет единиц. Требуется распределить все работы между исполнителями так, чтобы:

* каждый исполнитель исполнял ровно одну работу;
* суммарная стоимость выполнения всех работ была минимальной.

## Математическая постановка задачи о назначениях

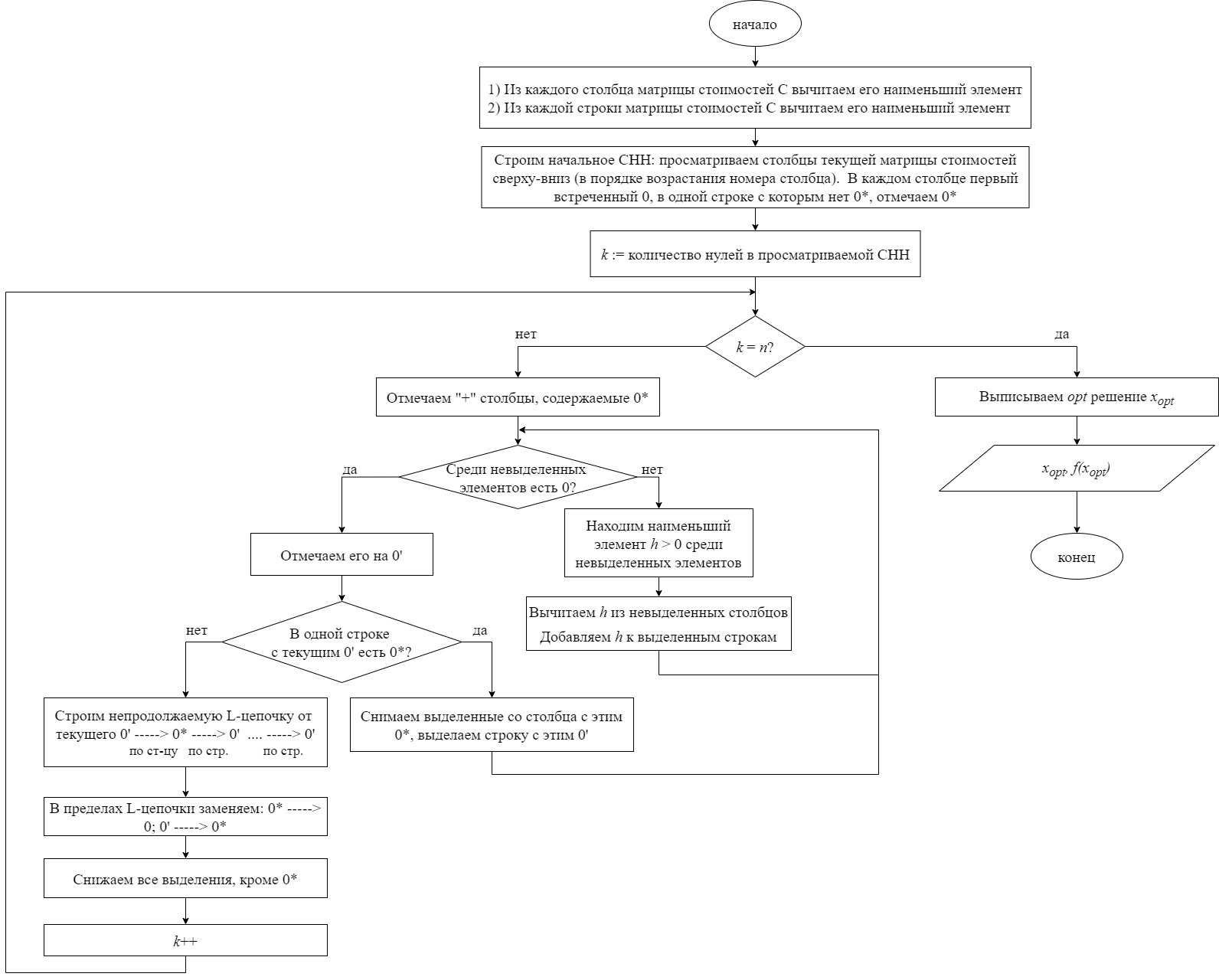
Введем управляемые переменные:

* – матрица назначения
* – матрица стоимостей

Тогда, суммарная стоимость выполнения всех работ:

Таким образом, введена математическая постановка задачи о назначениях:

## Венгерский метод решения задачи о назначениях



**Замечание**: Иногда встречается вариант задачи о назначениях в котором интерпретируются как прибыль от назначения -го работника на -ю работу. В этом случае задача о назначениях является задача максимизации:

Это задача о назначениях эквивалентна на задаче:

# Экспериментальная часть

## Исходные данные

Входные данные задаются в виде матрицы стоимостей C, для которой требуется решить задачу о назначениях в форме задач минимизации и максимизации:

## Текст программы

function lab1()

    clc();

    global MINIMIZATION; %#ok<GVMIS>

    MINIMIZATION = input('MINIMIZATION mode? y/n: [y]', 's');

    if isempty(MINIMIZATION) || MINIMIZATION == 'y'

        MINIMIZATION = true;

    else

        MINIMIZATION = false;

    end

    global DEBUG; %#ok<GVMIS>

    DEBUG = input('DEBUG mode? y/n: [y]', 's');

    if isempty(DEBUG) || DEBUG == 'y'

        DEBUG = true;

    else

        DEBUG = false;

    end

    if DEBUG

        fprintf('---- DEBUG MODE ----\n');

    else

        fprintf('---- TOTAL MODE ----\n');

    end

    % 1. Initilize matrix C

    CStruct = InitC();

    % 2. Normalize matrix C

    CStruct = NormalizeC(CStruct);

    % 3. Build initial СНН (Система независимых нулей) and get k

    [CStruct, k] = BuildInitSNN(CStruct);

    [~, n] = size(CStruct.C);

    iter = 1;

    while k < n

        if DEBUG

            fprintf('Итерация %d: k = %d\n', iter, k);

        end

        % 4. Mark columns that have 0\* with '+'

        CStruct = MarkPlusCols(CStruct);

        while true

            % 5. Find free 0 among unmarked columns

            [hasFreeZero, posZeroPrime] = FindFreeZero(CStruct);

            if hasFreeZero

                % 6.1. Mark free 0 to 0'

                CStruct = MarkZeroPrime(CStruct, posZeroPrime);

                % 7.1. Find 0\* in the row of the current 0'

                [hasZeroStar, posZeroStar] = FindZeroStarInRow( ...

                    CStruct, posZeroPrime);

                if hasZeroStar

                    % 8.1. Unmark column that has 0\*, Mark row that has 0'

                    CStruct = UpdateC(CStruct, posZeroPrime, posZeroStar);

                else

                    % 9.1. Build L-chain from the last 0'

                    L = BuildL(CStruct, posZeroPrime);

                    % 10.1. Flip 0'<->0\*

                    CStruct = Flip(CStruct, L);

                    % 11.1. Clean C

                    CStruct = CleanC(CStruct);

                    k = k + 1;

                    break;

                end

            else

                % 6.2. Find min positive value h among unmarked positions

                h = FindMinH(CStruct);

                % 7.2. Subtract h from unmarked cols, Add h to marked rows

                CStruct = ApplyH(CStruct, h);

            end

        end

        iter = iter + 1;

    end

    if DEBUG

        fprintf('---- DEBUG DONE ----\n');

    end

    % Because we mark CStar with 0/1, it turns into optimal matrix

    XOpt = CStruct.CStar;

    fOpt = CalculateF(CStruct.COrigin, XOpt);

    fprintf('X\_Opt: \n');

    disp(XOpt);

    fprintf('f(X\_Opt): \n');

    disp(fOpt);

end

%% 1. Initialize matrix C

function CStruct = InitC()

    COrigin = [1 4 7 9 4;

               9 3 8 7 4;

               3 4 6 8 2;

               8 2 4 6 7;

               7 6 9 8 5];

    [rows, cols] = size(COrigin);

    assert(rows == cols, 'C is not squared');

    CStar = zeros(rows, cols);

    CPrime = zeros(rows, cols);

    CStruct = ConstructC(COrigin, COrigin, ...

                         CStar(:,1), CStar(1,:), ...

                         CStar, CPrime, [0 0], [0 0]);

    global DEBUG; %#ok<GVMIS>

    tmp = DEBUG;

    DEBUG = true; %#ok<NASGU>

    PrintC(CStruct, 'Матрица стоимостей C');

    DEBUG = tmp;

    global MINIMIZATION; %#ok<GVMIS>

    if ~MINIMIZATION

        CStruct.C = max(max(CStruct.C)) - CStruct.C;

        PrintC(CStruct, 'Матрица максимизации C');

    end

end

%% 2. Normalize matrix C - Subtract every col and row by their min value

function CStruct = NormalizeC(CStruct)

    C               = CStruct.C;

    [rows, cols]    = size(C);

    % 2.1. Subtract cols

    for c = 1:cols

        C(:,c) = C(:,c) - min(C(:,c));

    end

    CStruct.C = C;

    PrintC(CStruct, 'Вычитать столбцы');

    % 2.2. Subtract rows

    for r = 1:rows

        C(r,:) = C(r,:) - min(C(r,:));

    end

    CStruct.C = C;

    PrintC(CStruct, 'Вычитать строки');

end

%% 3. Build initial СНН and return k = |СНН|

function [CStruct, k] = BuildInitSNN(CStruct)

    C               = CStruct.C;

    CStar           = CStruct.CStar;

    [rows, cols]    = size(C);

    k               = 0;

    for c = 1:cols

        for r = 1:rows

            if C(r, c) == 0 && ~sum(CStar(r,:))

                CStar(r, c) = 1;

                k = k + 1;

                break;

            end

        end

    end

    CStruct.CStar = CStar;

    PrintC(CStruct, 'Построить нач. СНН');

end

%% 4. Mark columns that have 0\* with '+'

function CStruct = MarkPlusCols(CStruct)

    C           = CStruct.C;

    CStar       = CStruct.CStar;

    markedCols  = CStruct.markedCols;

    [~, cols]   = size(C);

    for c = 1:cols

        if sum(CStar(:,c))

            markedCols(c) = 1;

        end

    end

    CStruct.markedCols = markedCols;

    PrintC(CStruct, 'Отметить "+" столбцы, содержащие 0\*');

end

%% 5. Find free 0 among unmarked positions

function [hasFreeZero, posZeroPrime] = FindFreeZero(CStruct)

    C               = CStruct.C;

    CStar           = CStruct.CStar;

    CPrime          = CStruct.CPrime;

    markedRows      = CStruct.markedRows;

    markedCols      = CStruct.markedCols;

    [rows, cols]    = size(C);

    hasFreeZero     = false;

    posZeroPrime    = [0 0];

    for c = 1:cols

        for r = 1:rows

            if ~markedCols(c) && ~markedRows(r) && C(r, c) == 0 && ...

               ~CStar(r, c) && ~CPrime(r, c)

                hasFreeZero = true;

                posZeroPrime = [r c];

                return;

            end

        end

    end

end

%% 6.1. Mark 0 to 0'

function CStruct = MarkZeroPrime(CStruct, posPrime)

    CStruct.CPrime(posPrime(1), posPrime(2)) = 1;

    CStruct.curPosPrime = posPrime;

    PrintC(CStruct, 'Отметить 0''');

end

%% 7.1. Find 0\* in the row of the current 0'

function [hasZeroStar, posZeroStar] = FindZeroStarInRow(CStruct, posPrime)

    C           = CStruct.C;

    CStar       = CStruct.CStar;

    [~, cols]   = size(C);

    r           = posPrime(1);

    hasZeroStar = false;

    posZeroStar = [0 0];

    for c = 1:cols

        if CStar(r, c)

            hasZeroStar = true;

            posZeroStar = [r c];

            return;

        end

    end

end

%% 8.1. Unmark column that has 0\*, Mark row that has 0'

function CStruct = UpdateC(CStruct, posPrime, posStar)

    r = posPrime(1);

    c = posStar(2);

    CStruct.curPosStar = posStar;

    CStruct.markedCols(c) = 0;

    CStruct.markedRows(r) = 1;

    PrintC(CStruct, 'Снимать выделенные со столбца с 0\*, выделать строку с 0''')

end

%% 9.1. Build L-chain from the last 0'

function L = BuildL(CStruct, posPrime)

    CStar           = CStruct.CStar;

    CPrime          = CStruct.CPrime;

    L               = [];

    % this variable determines the direction of the search

    % true  - find 0\*

    % false - find 0'

    dir = true;

    curPos = posPrime;

    while IsEqual(size(curPos), [1 2])

        L(end + 1, :) = curPos; %#ok<AGROW>

        if dir

            curPos = [find(CStar(:,curPos(2)) == 1), curPos(2)];

        else

            curPos = [curPos(1), find(CPrime(curPos(1),:) == 1)];

        end

        dir = ~dir;

    end

    PrintL(L, 'L-цеп');

end

%% 10.1. Flip 0'<->0\*

function CStruct = Flip(CStruct, L)

    CStar   = CStruct.CStar;

    CPrime  = CStruct.CPrime;

    [rows, ~] = size(L);

    for i = 1:rows

        r = L(i, 1);

        c = L(i, 2);

        if CStar(r, c)

            CStar(r, c) = 0;

            CPrime(r, c) = 1;

        elseif CPrime(r, c)

            CStar(r, c) = 1;

            CPrime(r, c) = 0;

        end

    end

    CStruct.CStar = CStar;

    CStruct.CPrime = CPrime;

    CStruct.curPosPrime = [0 0];

    CStruct.curPosStar = [0 0];

    PrintC(CStruct, 'Flip 0\*<->0''');

end

%% 11.1. Remove every mark except 0\*

function CStruct = CleanC(CStruct)

    CPrime = zeros(size(CStruct.C));

    CStruct.CPrime = CPrime;

    CStruct.markedRows = CPrime(:,1);

    CStruct.markedCols = CPrime(1,:);

    CStruct.curPosPrime = [0 0];

    CStruct.curPosStar = [0 0];

    PrintC(CStruct, 'Снимать все выделения, кроме 0\*');

end

%% 6.2. Find min positive value h among unmarked positions

function h = FindMinH(CStruct)

    C               = CStruct.C;

    markedRows      = CStruct.markedRows;

    markedCols      = CStruct.markedCols;

    [rows, cols]    = size(C);

    h = max(max(C));

    for c = 1:cols

        for r = 1:rows

            if ~markedCols(c) && ~markedRows(r)

                h = min(h, C(r, c));

            end

        end

    end

    global DEBUG; %#ok<GVMIS>

    if DEBUG

        fprintf('h = %d\n\n', h);

    end

end

%% 7.2. Subtract h from unmarked cols, Add h to marked rows

function CStruct = ApplyH(CStruct, h)

    C               = CStruct.C;

    markedRows      = CStruct.markedRows;

    markedCols      = CStruct.markedCols;

    [rows, cols]    = size(C);

    for c = 1:cols

        if ~markedCols(c)

            C(:,c) = C(:,c) - h;

        end

    end

    for r = 1:rows

        if markedRows(r)

            C(r,:) = C(r,:) + h;

        end

    end

    CStruct.C = C;

    PrintC(CStruct, 'Apply h to C');

end

function f = CalculateF(C, XOpt)

    [rows, cols] = size(C);

    f = 0;

    for r = 1:rows

        for c = 1:cols

            f = f + C(r, c) \* XOpt(r, c);

        end

    end

end

%% ------------------------------ HELPER ------------------------------ %%

% Return a struct holding required parameters of C

% CStar: матрица звёзд

% СPrime: матрица штрих

function CStruct = ConstructC(COrigin, C, markedRows, markedCols, ...

                              CStar, CPrime, curPosPrime, curPosStar)

    CStruct = struct('COrigin', COrigin, ...

                     'C', C, ...

                     'markedRows', markedRows, ...

                     'markedCols', markedCols, ...

                     'CStar', CStar, ...

                     'CPrime', CPrime, ...

                     'curPosPrime', curPosPrime, ...

                     'curPosStar', curPosStar);

end

% Print C to console

% CStruct: структура матрицы С

% msg: сообщение

function PrintC(CStruct, msg)

    C               = CStruct.C;

    CStar           = CStruct.CStar;

    CPrime          = CStruct.CPrime;

    markedRows      = CStruct.markedRows;

    markedCols      = CStruct.markedCols;

    curPosPrime     = CStruct.curPosPrime;

    curPosStar      = CStruct.curPosStar;

    [rows, cols]    = size(C);

    global DEBUG; %#ok<GVMIS>

    if DEBUG

        fprintf('%s:\n', msg);

        for r = 1:rows

            for c = 1:cols

                if IsEqual([r c], curPosPrime)

                    fprintf('<strong>%5g</strong>', C(r, c));

                    fprintf('<strong>''</strong>');

                elseif IsEqual([r c], curPosStar)

                    fprintf('<strong>%5g</strong>', C(r, c));

                    fprintf('<strong>\*</strong>');

                else

                    fprintf('%5g', C(r, c));

                    if CStar(r, c)

                        fprintf('\*');

                    elseif CPrime(r, c)

                        fprintf('''');

                    else

                        fprintf(' ');

                    end

                end

            end

            if markedRows(r)

                fprintf('    +\n');

            else

                fprintf('     \n');

            end

        end

        for c = 1:cols

            if markedCols(c)

                fprintf('%5c ', '+');

            else

                fprintf('%5c ', ' ');

            end

        end

        fprintf('\n\n');

    end

end

% Print L to console

function PrintL(L, msg)

    global DEBUG; %#ok<GVMIS>

    if DEBUG

        fprintf('%s:\n', msg);

        fprintf('0''(%d,%d)', L(1, 1), L(1, 2));

        if length(L) <= 2

            fprintf('\n\n');

            return;

        end

        for l = 2:length(L)

            if ~mod(l, 2)

                fprintf('--> 0\*(%d,%d)', L(l, 1), L(l, 2));

            else

                fprintf('--> 0''(%d,%d)', L(l, 1), L(l, 2));

            end

        end

        fprintf('\n\n');

    end

end

% Compare index of 2 position A and B

function isEqual = IsEqual(posA, posB)

    if posA(1) == posB(1) && posA(2) == posB(2)

        isEqual = true;

    else

        isEqual = false;

    end

end

## Результаты

* Задача минимизации

---- TOTAL MODE ----  
Матрица стоимостей C:  
 1 4 7 9 4   
 9 3 8 7 4   
 3 4 6 8 2   
 8 2 4 6 7   
 7 6 9 8 5   
   
  
X\_Opt:   
 1 0 0 0 0  
 0 1 0 0 0  
 0 0 0 0 1  
 0 0 1 0 0  
 0 0 0 1 0  
  
f(X\_Opt):   
 18

* Задача максимации

---- TOTAL MODE ----  
Матрица стоимостей C:  
 1 4 7 9 4   
 9 3 8 7 4   
 3 4 6 8 2   
 8 2 4 6 7   
 7 6 9 8 5   
   
  
X\_Opt:   
 0 0 0 1 0  
 1 0 0 0 0  
 0 1 0 0 0  
 0 0 0 0 1  
 0 0 1 0 0  
  
f(X\_Opt):   
 38