|  |  |
| --- | --- |
| 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\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**по Лабораторной работе №1**

**по курсу**

**«Методы вычислений»**

**Тема**

**«Венгерский метод решения задачи о назначениях»**

**Вариант 10**

Студент \_\_\_ИУ7-11М\_\_\_\_  **\_\_\_**Карпухин А.С.\_\_\_

(Группа) (И.О.Фамилия)

Преподаватель  **\_\_\_\_**Власов П.А.**\_\_\_\_\_\_**

(И.О.Фамилия)

*2020 г.*

1. **Постановка задачи о назначениях**
   1. **Содержательная постановка задачи**

В распоряжении работодателя имеется n работ и n исполнителей. Стоимость выполнения i-ой работы j-ым исполнителем составляет cij ≥ 0 единиц. Требуется распределить работы между исполнителями так, чтобы:

1. Каждый исполнитель выполнил ровно одну работу;
2. Общая стоимость выполнения всех работ была бы минимальной - в случае задачи минимизации, или максимальной - в случае задачи максимизации.
   1. **Математическая постановка задачи**

Введем матрицу C, элемент cij ≥ 0 которой – стоимость выполнения i-ой работы j-ым исполнителем. Введем управляемые переменные:

Тогда общая стоимость выполнения работ будет иметь вид:

Математическая постановка задачи о назначениях в случае задачи минимизации примет вид:

В случае задачи максимизации постановка задачи примет вид:

Данная задача может быть сведена к эквивалентной ей задаче минимизации вида:

* 1. **Исходные данные варианта**

Матрица стоимостей:

1. **Венгерский метод решения задачи о назначениях**

Псевдокод венгерского метода приведен ниже.

|  |
| --- |
| **Алгоритм 1.** Венгерский метод решения задачи о назначениях |
| **Вход:** С – матрица стоимостей размером n на n.  **Выход:** Xopt – матрица оптимального решения,  fopt = f(Xopt) – оптимальное решение задачи о назначениях.  **Начало**   1. В каждом столбце матрицы С найти минимальный элемент и вычесть его из всех элементов данного столбца; 2. В каждой строке матрицы С найти минимальный элемент и вычесть его из всех элементов данной строки; 3. **Для** каждого столбца в матрице С   3.1. **Для** каждого элемента в текущем столбце  3.1.1. **Если** текущий элемент = 0 **то** пометить его 0\* и выйти из внутреннего цикла;   1. Полученный набор 0\* - система независимых нулей (СНН); 2. k = число 0\* в полученной СНН; 3. **Пока** k !**=** n   6.1. Отметить столбцы с 0\* знаком +;  6.2. **Бесконечный цикл**  6.2.1. **Если** среди невыделенных элементов матрицы С есть 0 **то**  6.2.1.1. Отметить такой 0 знаком `;  6.2.1.2. **Если** в одной строке с 0` есть 0\* **то** снять выделение со столбца с 0\* и отметить строку с 0` знаком +;  6.2.1.3. **Иначе**  6.2.1.3.1. Построить от текущего 0` непродолжаемую L-цепочку (от 0` к 0\* по столбцу, от 0\* к 0` по строке);  6.2.1.3.2. В пределах L-цепочки заменить 0\* на 0, 0` на 0\*;  6.2.1.3.3. Снять в матрице С все выделения кроме 0\*;  6.2.1.3.4. Выйти из бесконечного цикла;  6.2.2. **Иначе**  6.2.2.1. Найти наименьший положительный элемент h среди невыделенных элементов матрицы С;  6.2.2.2. Вычесть h из невыделенных столбцов;  6.2.2.3. Прибавить h к выделенным строкам;  7. Составить нулевую матрицу оптимального решения Xopt размером n на n;  8. Поставить 1 на те позиции в матрице Xopt, на которых стоят 0\* в матрице C;  9. Вычислить оптимальное решение как  **Конец.** |

1. **Текст программы**

Код разработанной программной реализации венгерского метода приведен в листингах 1 и 2.

Листинг 1. Код класса, реализующего венгерский метод

classdef HungarianMethod

properties

Matrix (:,:) double

end

methods

function obj = HungarianMethod(matrix)

obj.Matrix = matrix;

end

function [srow, scol] = BuildZeroSystem(obj, isMax)

if isMax

matrix = max(obj.Matrix, [], 'all') - obj.Matrix;

else

matrix = obj.Matrix;

end

fprintf("Исходная матрица:\n");

MatrixOperations.PrintMatrix(...

matrix, [], [], [], [], [], [], [], []);

matrix = MatrixOperations.SubtractMinInColumn(matrix);

fprintf("\nВычитание минимумов в столбцах:\n");

MatrixOperations.PrintMatrix(...

matrix, [], [], [], [], [], [], [], []);

matrix = MatrixOperations.SubtractMinInRow(matrix);

fprintf("\nВычитание минимумов в строках:\n");

MatrixOperations.PrintMatrix(...

matrix, [], [], [], [], [], [], [], []);

[srow, scol] = MatrixOperations.BuildZeroSystem(matrix);

fprintf("\nСистема независымых нулей №1:\n");

MatrixOperations.PrintMatrix(...

matrix, srow, scol, [], [], [], [], [], []);

i = 2;

while length(srow) ~= size(matrix, 1)

[matrix, srow, scol] = ...

MatrixOperations.UpgradeZeroSystem(matrix, srow, scol);

fprintf("\nСистема независымых нулей №%d:\n", i);

MatrixOperations.PrintMatrix(...

matrix, srow, scol, [], [], [], [], [1,2,3], [3,2,1]);

i = i + 1;

end

end

function [X, f] = CalcOptSolution(obj, row, col)

arguments

obj

row, col (1,:) double

end

X = zeros(size(obj.Matrix, 1));

for i = 1:length(row)

X(row(i), col(i)) = 1;

end

f = sum(obj.Matrix .\* X, 'all');

end

function Result = EvalDebug(obj)

Result = 0;

end

end

end

Листинг 2. Код класса, реализующего вспомогательные функции работы с матрицами.

|  |
| --- |
| classdef MatrixOperations  methods(Static)  % Вычесть минимумы в каждой строке матрицы  function Result = SubtractMinInRow(matrix)  arguments  matrix (:,:) double  end    tr = transpose(matrix);    Result = transpose(tr - min(tr));  end  % Вычесть минимумы в каждом столбце матрицы  function Result = SubtractMinInColumn(matrix)  arguments  matrix (:,:) double  end    Result = matrix - min(matrix);  end    % Построить систему независимых нулей  function [sRow, sCol] = BuildZeroSystem(matrix)  arguments  matrix (:,:) double  end    srow = []; scol = [];  [row, col] = find(~matrix);    for i = 1:length(row)  if (~any(scol == col(i))) && (~any(srow == row(i)))  srow = [srow, row(i)];  scol = [scol, col(i)];  end  end    sRow = srow; sCol = scol;  end    % Улучшить систему независимых нулей  function [Matrix, sRow, sCol] = UpgradeZeroSystem(matrix, row, col)  arguments  matrix (:,:) double  row, col (1,:) double  end    % Столбцы и строки с 0\*  mrow = []; mcol = col;  % Индексы элементов, помеченных '  srow = []; scol = [];    fprintf("\nОтмечаем столбцы, содержащие 0\*:\n");  MatrixOperations.PrintMatrix(...  matrix, row, col, srow, scol, mrow, mcol, [], []);    while true  % Находим среди неотмеченных элементов 0  [found, x, y] = MatrixOperations.FindNonMarkedZero(...  matrix, mrow, mcol);  % Если не найден, вычитаем из невыделенных столбцов и  % прибавляем к выделенным строкам минимум среди  % невыделенных элементов  while ~found  matrix = MatrixOperations.SubtractNonMarkedMin(...  matrix, mrow, mcol);    fprintf("\nНет нулей, вычитаем минимум:\n");  MatrixOperations.PrintMatrix(...  matrix, row, col, srow, scol, mrow, mcol, [], []);    [found, x, y] = MatrixOperations.FindNonMarkedZero(...  matrix, mrow, mcol);  end  % Добавляем новый 0' в список помеченных '  srow = [srow, x]; scol = [scol, y];    fprintf("\nОтмечаем невыделенный 0 символом ':\n");  MatrixOperations.PrintMatrix(...  matrix, row, col, srow, scol, mrow, mcol, [], []);    % Если в одной строке с 0' есть 0\*  if any(row == x)  ind = mcol == col(row == x);  % Снять выделение со столбца с 0\*  mcol(ind) = [];  % Пометить строку с 0'  mrow = [mrow, x];    fprintf("\nВ строке с 0' есть 0\*, снимаем " + ...  "выделение со столбца, выделяем строку:\n");  MatrixOperations.PrintMatrix(...  matrix, row, col, srow, scol, mrow, mcol, [], []);  else  [sRow, sCol] = MatrixOperations.BuildLChain(...  matrix, row, col, srow, scol);  Matrix = matrix;  break;  end  end  end    % Найти непомеченный 0 среди невыделенных элементов матрицы  function [found, x, y] = FindNonMarkedZero(matrix, row, col)  arguments  matrix (:,:) double  row, col (1,:) double  end    for i = setdiff(1:size(matrix, 1), row)  for j = setdiff(1:size(matrix, 2), col)  if matrix(i, j) == 0  x = i; y = j; found = true;  return;  end  end  end    [found, x, y] = deal(false, -1, -1);  end    % Вычесть из невыделенных столбцов и прибавить к выделенным  % строкам минимум среди невыделенных элементов  function Result = SubtractNonMarkedMin(matrix, row, col)  arguments  matrix (:,:) double  row, col (1,:) double  end    nrow = setdiff(1:size(matrix, 1), row);  ncol = setdiff(1:size(matrix, 2), col);  min = max(matrix, [], 'all');  for i = nrow  for j = ncol  if matrix(i, j) < min  min = matrix(i, j);  end  end  end    for i = ncol  for j = 1:size(matrix, 1)  matrix(j, i) = matrix(j, i) - min;  end  end    for i = row  for j = 1:size(matrix, 2)  matrix(i, j) = matrix(i, j) + min;  end  end    Result = matrix;  end    % Построить непродолжаемую L-цепочку и обновить систему нулей  function [sRow, sCol] = BuildLChain(matrix, row, col, srow, scol)  arguments  matrix (:,:) double  row, col (1,:) double  srow, scol (1,:) double  end    % Начиная с последнего отмеченного 0'  [x, y] = deal(srow(length(srow)), scol(length(scol)));  lrow = x; lcol = y;    while true  x = row(col == y);  if isempty(x)  break;  end  lrow = [lrow, x]; lcol = [lcol, y];  y = scol(srow == x);  lrow = [lrow, x]; lcol = [lcol, y];  end  fprintf("\nВ строке с 0' нет 0\*, строим L-цепочку:\n");  MatrixOperations.PrintMatrix(...  matrix, row, col, srow, scol, [], [], lrow, lcol);    for i = 1:length(lrow)  % Если 0\*, то снимаем \*  if mod(i, 2) == 0  rowind = row == lrow(i);  colind = col == lcol(i);  row(colind & rowind) = [];  col(colind & rowind) = [];  % Если 0', то помечаем \*  else  row = [row, lrow(i)];  col = [col, lcol(i)];  end  end  [sRow, sCol] = deal(row, col);  end    function PrintMatrix(...  matrix, row, col, srow, scol, mrow, mcol, lrow, lcol)  arguments  matrix (:,:) double  row, col (1,:) double  srow, scol (1,:) double  mrow, mcol (1,:) double  lrow, lcol (1,:) double  end    for i = 1:size(matrix, 1)  if any(mcol == i)  fprintf(' + ');  else  fprintf(' ');  end  end  fprintf('\n');    for i = 1:size(matrix, 1)  for j = 1:size(matrix, 2)  if any(lrow == i & lcol == j) && ...  any(row == i & col == j)  fprintf('%3d\*L ', matrix(i, j));  elseif any(lrow == i & lcol == j) && ...  any(srow == i & scol == j)  fprintf('%3d''L ', matrix(i, j));  elseif any(row == i & col == j)  fprintf('%4d\* ', matrix(i, j));  elseif any(srow == i & scol == j)  fprintf('%4d'' ', matrix(i, j));  else  fprintf('%5d ', matrix(i, j));  end  end  if any(mrow == i)  fprintf(' +\n');  else  fprintf('\n');  end  end  end  end  end |

1. **Результаты расчётов**

Результаты расчетов оптимального решения для задач минимизации и максимизации согласно исходным данным варианта приведены в листинге 3.

Листинг 3. Результаты расчетов согласно исходным данным варианта.

Задача минимизации:

Матрица назначений:

X =

0 0 0 1 0

0 0 1 0 0

0 0 0 0 1

0 1 0 0 0

1 0 0 0 0

Оптимальное решение:

f = 18

Задача максимизации:

Матрица назначений:

X =

0 0 0 1 0

0 0 0 0 1

0 0 1 0 0

1 0 0 0 0

0 1 0 0 0

Оптимальное решение:

f = 37