МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

|  |
| --- |
| КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ |

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент |  |  |  | Н. А. Волкова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| КОНТРОЛЬНАЯ РАБОТА №2 |
| **Разработка алгоритмов методов решения оптимизационных задач** |
| по дисциплине: Прикладные модели оптимизации |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | Z1431 |  |  |  | М.Д.Быстров |
|  | номер группы |  | подпись, дата |  | инициалы, фамилия |
| Студенческий билет № | 2021/3572 | |  |  |  |

Санкт-Петербург 2025

# Оглавление

[Оглавление 2](#_Toc187792880)

[Задание 3](#_Toc187792881)

[Решение 4](#_Toc187792882)

[Исходный код 7](#_Toc187792883)

# Задание

**Цель**: сформулировать, формализовать и решить с помощью венгерского метода задачу о назначении размерностью 5 × 5 как задачу линейного программирования.

Показатели эффективности назначения i-го кандидата на j-ю работу.

| № | Таблица показателей эффективности |
| --- | --- |
| 4 | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 14 | 1 | 1 | 7 | 2 | | 11 | 12 | 4 | 3 | 1 | | 10 | 3 | 10 | 8 | 12 | | 15 | 10 | 9 | 1 | 5 | | 15 | 2 | 12 | 1 | 10 | |

# Решение

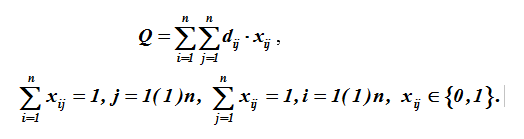
На одном из этапов производства возникла необходимость распределения между несколькими работниками определенных задач. Количество работников 5, количество задач – 5. Известна эффективность каждого работника для каждой из задач.

| № | Таблица показателей эффективности |
| --- | --- |
| 4 | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 14 | 1 | 1 | 7 | 2 | | 11 | 12 | 4 | 3 | 1 | | 10 | 3 | 10 | 8 | 12 | | 15 | 10 | 9 | 1 | 5 | | 15 | 2 | 12 | 1 | 10 | |

Чем меньше показатель эффективности сотрудника для задачи, тем быстрее сотрудник может выполнить задание.

Необходимо распределить задачи между сотрудниками таким образом, чтобы минимизировать общие затраты рабочего времени.

Математическая модель.



Решение.

Таблица показателей эффективности:

[[14. 1. 1. 7. 2.]

[11. 12. 4. 3. 1.]

[10. 3. 10. 8. 12.]

[15. 10. 9. 1. 5.]

[15. 2. 12. 1. 10.]]

**Шаг 1**

Произведена редукция:

[[ 6. 0. 0. 6. 1.]

[ 3. 11. 3. 2. 0.]

[ 0. 0. 7. 5. 9.]

[ 7. 9. 8. 0. 4.]

[ 7. 1. 11. 0. 9.]]

Метод проб и ошибок

Решение не найдено, модификация матрицы

Удалены строки {0, 1, 2}, столбцы {3}

Минимальный элемент 1.0

Модифицированная матрица

[[ 6. 0. 0. 5. 1.]

[ 3. 11. 3. 1. 0.]

[ 0. 0. 7. 4. 9.]

[ 6. 8. 7. 0. 3.]

[ 6. 0. 10. 0. 8.]]

**Шаг 2**

Произведена редукция:

[[ 6. 0. 0. 5. 1.]

[ 3. 11. 3. 1. 0.]

[ 0. 0. 7. 4. 9.]

[ 6. 8. 7. 0. 3.]

[ 6. 0. 10. 0. 8.]]

Метод проб и ошибок

Найдено решение, выбор клеток [[0, 2], [1, 4], [2, 0], [3, 3], [4, 1]]

**Ответ:**

Кандидат 1 должен выполнять работу 3

Кандидат 2 должен выполнять работу 5

Кандидат 3 должен выполнять работу 1

Кандидат 4 должен выполнять работу 4

Кандидат 5 должен выполнять работу 2

Итоговая минимальная сумма затрат Cmin = 15.0

# Исходный код

Для решения написана программа на языке Python.

Файл “cr2.py”

import numpy as np;

# редукция таблицы

def reduction(a):

    b = a.copy();

    # редукция по строкам

    for r in b:

        minv = min(r);

        for i in range (len(r)):

            r[i] = r[i] - minv;

    # редукция по столбцам

    for i in range(b.shape[1]):

        minv = float("inf");

        for j in range(b.shape[0]):

            minv = min(b[j][i], minv);

        for j in range(b.shape[0]):

            b[j][i] = b[j][i] - minv;

    return b;

# метод проб и ошибок - выбор суммы из нулей перебором

def try\_and\_error(a, p = [], e = set()):

    # рассчитываем текущую строку

    row = len(p);

    for col in range(a.shape[1]):

        if (row == a.shape[1]): breakpoint();

        # если в строке есть ноль, который не вычеркнут

        if (a[row][col] == 0

            and (row,col) not in e):

            # копия вычеркнутых клеток

            excluded = e.copy();

            # вычеркиваем невычеркнутые элементы в колонке

            for r in range(a.shape[0]):

                if (a[r][col] == 0

                    and (r, col) not in excluded):

                    excluded.add((r,col));

            # вычеркиваем невычеркнутые элементы в строке

            for c in range(a.shape[1]):

                if (a[row][c] == 0

                    and (row, c) not in excluded):

                    excluded.add((row,c));

            path = p.copy();

            path.append([row, col]);

            if (len(path) == a.shape[0]):

                return path;

            # рекурсивный вызов на следующую строку

            ret = try\_and\_error(a, path, excluded);

            # рекурсия вернула путь - выбор возможен

            if (len(ret)):

                return ret;

    # перебором не добились результата -

    # выбор из нулей невозможен

    # возвращаем пустой путь

    return [];

# модификация матрицы

def modify(a):

    b = a.copy();

    excluded\_rows = set();

    excluded\_cols = set();

    # пока остаются нули

    while (1):

        zeros\_exist = 0;

        # проверка остались ли неисключенные нули

        for i in range(b.shape[0]):

            for j in range(b.shape[1]):

                if (i not in excluded\_rows

                and j not in excluded\_cols

                and b[i][j] == 0):

                    zeros\_exist = 1;

                    break;

        # если нулей не осталось - перестаем вычеркивать

        if (zeros\_exist == 0):

            break;

        row\_zeros\_num = {};

        col\_zeros\_num = {};

        # подсчет кол-ва нулей в неисключенных строках и столбцах

        for i in range(b.shape[0]):

            for j in range(b.shape[1]):

                if (i not in excluded\_rows

                and j not in excluded\_cols

                and b[i][j] == 0):

                    row\_zeros\_num[i] = row\_zeros\_num[i] + 1 if i in row\_zeros\_num else 1;

                    col\_zeros\_num[j] = col\_zeros\_num[j] + 1 if j in col\_zeros\_num else 1;

        zeros\_row = {};

        zeros\_col = {};

        # группируем строки и столбцы по пол-ву нулей

        for row, zeros in row\_zeros\_num.items():

            zeros\_row[zeros] = zeros\_row[zeros] + [row] if zeros in zeros\_row else [row];

        for col, zeros in col\_zeros\_num.items():

            zeros\_col[zeros] = zeros\_col[zeros] + [col] if zeros in zeros\_col else [col];

        # узнаем максимальное кол-во нулей в строках и столбцах

        max\_zeros\_row = max(zeros\_row.keys());

        max\_zeros\_col = max(zeros\_col.keys());

        # в приоритете удаление строки

        if (max\_zeros\_row >= max\_zeros\_col and b.shape[0] > 0):

            # удаляем строку

            delete\_row = zeros\_row[max\_zeros\_row][0];

            excluded\_rows.add(delete\_row);

        else:

            # удаляем колонку

            delete\_col = zeros\_col[max\_zeros\_col][0];

            excluded\_cols.add(delete\_col);

    if (b.shape[0] == 0):

        raise Exception("В ходе модификации удалена вся таблица");

    # поиск минимального элемента в сокращенной таблице

    remain\_min = float("inf")

    for i in range(b.shape[0]):

        for j in range(b.shape[1]):

            if (i not in excluded\_rows

            and j not in excluded\_cols):

                remain\_min = min(remain\_min, b[i][j]);

    print(f"Удалены строки {excluded\_rows}, столбцы {excluded\_cols}");

    print(f"Минимальный элемент {remain\_min}");

    for i in range(b.shape[0]):

        for j in range (b.shape[1]):

            # из сокращенной матрицы вычитаем её минимум

            if (i not in excluded\_rows

                and j not in excluded\_cols):

                b[i][j] = b[i][j] - remain\_min;

            # к элементам на пересечении этот минимум добавляем

            if (i in excluded\_rows

                and j in excluded\_cols):

                b[i][j] = b[i][j] - remain\_min;

    return b;

a = np.array(

    [[14, 1, 1, 7, 2],

     [11, 12, 4, 3, 1],

     [10, 3, 10, 8, 12],

     [15, 10, 9, 1, 5],

     [15, 2, 12, 1, 10]],

    float);

# тест

# a = np.array(

#     [[1, 4, 6, 3],

#      [9, 7, 10, 9],

#      [4, 5, 11, 7],

#      [8, 7, 8, 5]],

#     float);

print("Таблица показателей эффективности:");

print(a);

print();

source = a.copy();

stage = 1;

while 1:

    a = reduction(a);

    print();

    print(f"Шаг {stage}");

    print();

    print("Произведена редукция:");

    print(a);

    print("Метод проб и ошибок");

    path = try\_and\_error(a);

    if (len(path) > 0):

        print(f"Найдено решение, выбор клеток {path}");

        sum = 0;

        for i,j in path:

            print(f"Кандидат {i+1} должен выполнять работу {j+1}");

            sum = sum + source[i][j];

        print(f"Итоговая сумма Cmin = {sum}");

        exit();

    else:

        print("Решение не найдено, модификация матрицы");

        a = modify(a);

        print("Модифицированная матрица");

        print(a);

    stage = stage + 1;