

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

**Лабораторная работа №7**

по дисциплине «Основы теории принятия решений»

**Выполнил:**

студент уч. гр. ВМО41

Волкова Э. Ю.

**Проверил:**

проф. Кобак В. Г.

Ростов-на-Дону

2025

**Введение**

Предметом области исследования расписаний является круг задач проектирования и организационного управления в различных системах, в которых требуется найти наилучшее (оптимальное) значение выбранных критериев их функционирования с учетом имеющихся ограничений.

В данном руководстве будут рассмотрены модификации алгоритма “Критического пути” для решения задачи получения расписания в однородной вычислительной системе.

В качестве основного критерия рассматривается минимаксный критерий, который минимизирует максимальное время окончания работы программ.

**Постановка задачи**

Имеется вычислительная система, состоящая из  несвязанных однородных устройств . На обработку поступает - множество независимых параллельных заданий (работ) , известно время решения **** каждого задания  на устройстве ****(****для всех устройств одинаково, если не равно бесконечности). При этом задание может выполняться на одном устройстве, время выполнения определяется значением ****, и не выполняться на другом совсем, тогда время выполнения задачи на этом устройстве определено как **= ∞**. В каждый момент времени выполняется не более одного задания на устройстве. Требуется найти такое распределение заданий по процессорам, при котором суммарное время выполнения заданий на каждом из процессоров было бы минимальным.

Минимаксный критерий определяется в следующем виде: , где - время завершения работы процессора .

**Модифицированные алгоритмы с учётом бесконечности**

Для решения задачи планирования с матрицами загрузки имеющими бесконечности рассматривается три различные модификации алгоритма.

Ш.1 Упорядочиваем строки в порядке убывания веса множества заданий; , при этом считаем количество бесконечностей в строке и учитываем при сортировке, первыми будут строки имеющие хотя бы один элемент - бесконечность.

Последующие шаги алгоритма согласно алгоритму распределения без изменений.

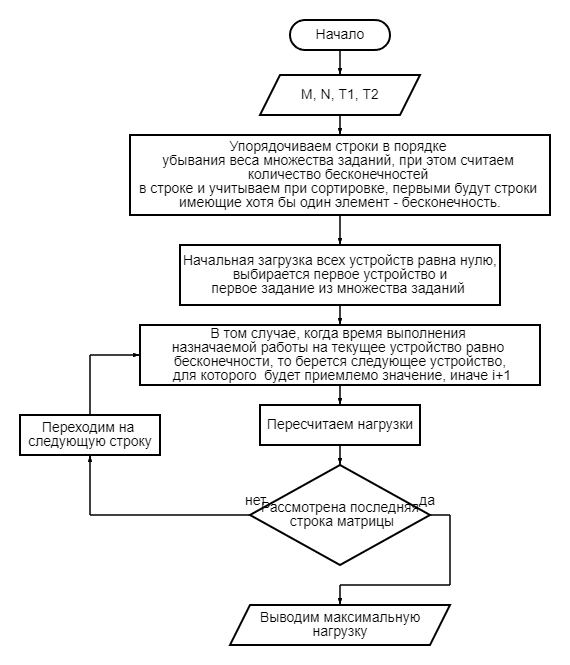


Рисунок 2 – Схема алгоритма распределения с учётом бесконечности

С учётом количества бесконечности в строке:

Ш.1 Упорядочиваем строки в порядке убывания веса множества заданий;  и в порядке убывания количества бесконечностей в строке, если строки имеют одинаковое число бесконечностей, то выполняем упорядочивание в порядке убывания элементов отличных от бесконечностей.

Последующие шаги алгоритма согласно алгоритму распределения без изменений.

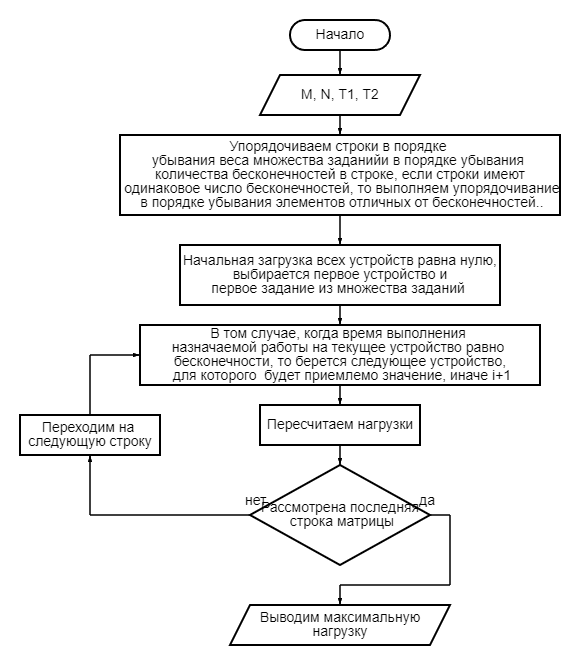


Рисунок 3 – Схема алгоритма распределения с учётом количества бесконечности в строке

**Вычислительные эксперименты**

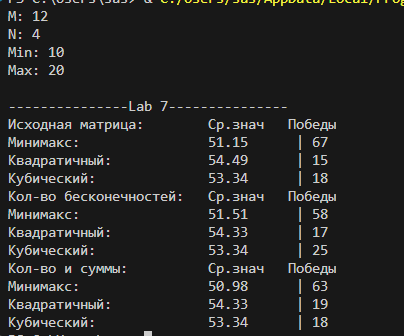


Рисунок 4 – Эксперимент №1

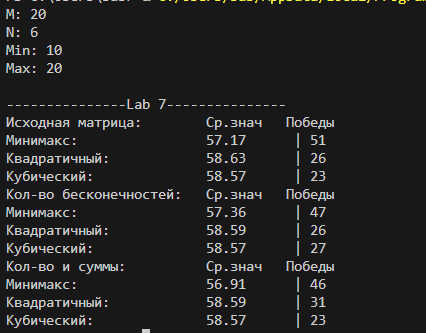


Рисунок 5 – Эксперимент №2

**Листинг программы**

import numpy as np

import numpy as np

def generate\_random\_matrix(M, N, min\_val=1, max\_val=10, inf\_prob=0.2):

    matrix = np.random.randint(min\_val, max\_val + 1, size=(M, N)).astype(float)

    mask = np.random.random(size=(M, N)) < inf\_prob

    for i in range(M):

        if np.all(mask[i]):

            j = np.random.randint(0, N)

            mask[i, j] = False

    matrix[mask] = np.inf

    return matrix

def sort\_matrix(matrix, sort\_type='Исходная'):

    if sort\_type == 'Исходная':

        return matrix.copy()

    rows = []

    for i, row in enumerate(matrix):

        inf\_count = np.sum(row == np.inf)

        row\_sum = np.sum(row[row != np.inf])

        rows.append((i, inf\_count, row\_sum))

    if sort\_type == 'Кол-во бесконечностей':

        rows.sort(key=lambda x: (-x[1], x[0]))

    elif sort\_type == 'Кол-во бесконечностей и сумма':

        rows.sort(key=lambda x: (-x[1], -x[2], x[0]))

    sorted\_matrix = np.array([matrix[row[0]] for row in rows])

    return sorted\_matrix

def PZ\_algorithm(matrix, criterion='Минимакс', sort\_type='Исходная'):

    M, N = len(matrix), len(matrix[0])

    tasks = []

    for i in range(M):

        row = matrix[i]

        if criterion == 'Минимакс':

            weight = max([x for x in row if x != np.inf])

        elif criterion == 'Квадратичный':

            weight = sum(x \*\* 2 for x in row if x != np.inf)

        elif criterion == 'Кубический':

            weight = sum(x \*\* 3 for x in row if x != np.inf)

        tasks.append((i, weight))

    tasks.sort(key=lambda x: -x[1])

    sums = np.zeros(N)

    for task in tasks:

        task\_idx = task[0]

        current\_pi = np.argmin(sums)

        if matrix[task\_idx][current\_pi] == np.inf:

            ch = sums.copy()

            while matrix[task\_idx][current\_pi] == np.inf:

                ch[current\_pi] = np.inf

                current\_pi = np.argmin(ch)

        sums[current\_pi] += matrix[task\_idx][current\_pi]

    return sums, np.max(sums)

M = int(input("M: "))

N = int(input("N: "))

min\_val = int(input("Min: "))

max\_val = int(input("Max: "))

num\_matrix = 100

all\_rand\_minimax, all\_rand\_square, all\_rand\_cube = [], [], []

all\_count\_minimax, all\_count\_square, all\_count\_cube = [], [], []

all\_sum\_minimax, all\_sum\_square, all\_sum\_cube = [], [], []

rand\_win\_minimax, rand\_win\_square, rand\_win\_cube = 0, 0, 0

count\_win\_minimax, count\_win\_square, count\_win\_cube = 0, 0, 0

sum\_win\_minimax, sum\_win\_square, sum\_win\_cube = 0, 0, 0

for i in range(num\_matrix):

    matrix = generate\_random\_matrix(M, N, min\_val, max\_val)

    for sort\_type in ['Исходная', 'Кол-во бесконечностей', 'Кол-во бесконечностей и сумма']:

        for crit in ['Минимакс', 'Квадратичный', 'Кубический']:

            matrix\_sort = sort\_matrix(matrix, sort\_type)

            sums, makespan = PZ\_algorithm(matrix\_sort, crit, sort\_type)

            match sort\_type:

                case 'Исходная':

                    match crit:

                        case 'Минимакс':

                            all\_rand\_minimax.append(makespan)

                        case 'Квадратичный':

                            all\_rand\_square.append(makespan)

                        case 'Кубический':

                            all\_rand\_cube.append(makespan)

                case 'Кол-во бесконечностей':

                    match crit:

                        case 'Минимакс':

                            all\_count\_minimax.append(makespan)

                        case 'Квадратичный':

                            all\_count\_square.append(makespan)

                        case 'Кубический':

                            all\_count\_cube.append(makespan)

                case 'Кол-во бесконечностей и сумма':

                    match crit:

                        case 'Минимакс':

                            all\_sum\_minimax.append(makespan)

                        case 'Квадратичный':

                            all\_sum\_square.append(makespan)

                        case 'Кубический':

                            all\_sum\_cube.append(makespan)

for i in [all\_rand\_minimax, all\_rand\_square, all\_rand\_cube]:

    aver = sum(i)/len(i)

    i.append(aver)

for i in [all\_count\_minimax, all\_count\_square, all\_count\_cube]:

    aver = sum(i)/len(i)

    i.append(aver)

for i in [all\_sum\_minimax, all\_sum\_square, all\_sum\_cube]:

    aver = sum(i)/len(i)

    i.append(aver)

for i in range(num\_matrix):

    min\_res\_val\_rand = min(all\_rand\_minimax[i], all\_rand\_square[i], all\_rand\_cube[i])

    if min\_res\_val\_rand == all\_rand\_minimax[i]:

        rand\_win\_minimax += 1

    elif min\_res\_val\_rand == all\_rand\_square[i]:

        rand\_win\_square += 1

    elif min\_res\_val\_rand == all\_rand\_cube[i]:

        rand\_win\_cube += 1

    min\_res\_val\_count = min(all\_count\_minimax[i], all\_count\_square[i], all\_count\_cube[i])

    if min\_res\_val\_count == all\_count\_minimax[i]:

        count\_win\_minimax += 1

    elif min\_res\_val\_count == all\_count\_square[i]:

        count\_win\_square += 1

    elif min\_res\_val\_count == all\_count\_cube[i]:

        count\_win\_cube += 1

    min\_res\_val\_sum = min(all\_sum\_minimax[i], all\_sum\_square[i], all\_sum\_cube[i])

    if min\_res\_val\_sum == all\_sum\_minimax[i]:

        sum\_win\_minimax += 1

    elif min\_res\_val\_sum == all\_sum\_square[i]:

        sum\_win\_square += 1

    elif min\_res\_val\_sum == all\_sum\_cube[i]:

        sum\_win\_cube += 1

print("\n---------------Lab 7---------------")

print(f"{'Исходная матрица:':<25}{'Ср.знач':<10}{'Победы':<10}")

print(f"{'Минимакс:':<25}{round(all\_rand\_minimax[-1], 2):<10} | {rand\_win\_minimax:<5}")

print(f"{'Квадратичный:':<25}{round(all\_rand\_square[-1], 2):<10} | {rand\_win\_square:<5}")

print(f"{'Кубический:':<25}{round(all\_rand\_cube[-1], 2):<10} | {rand\_win\_cube:<5}")

print(f"{'Кол-во бесконечностей:':<25}{'Ср.знач':<10}{'Победы':<10}")

print(f"{'Минимакс:':<25}{round(all\_count\_minimax[-1], 2):<10} | {count\_win\_minimax:<5}")

print(f"{'Квадратичный:':<25}{round(all\_count\_square[-1], 2):<10} | {count\_win\_square:<5}")

print(f"{'Кубический:':<25}{round(all\_count\_cube[-1], 2):<10} | {count\_win\_cube:<5}")

print(f"{'Кол-во и суммы:':<25}{'Ср.знач':<10}{'Победы':<10}")

print(f"{'Минимакс:':<25}{round(all\_sum\_minimax[-1], 2):<10} | {sum\_win\_minimax:<5}")

print(f"{'Квадратичный:':<25}{round(all\_sum\_square[-1], 2):<10} | {sum\_win\_square:<5}")

print(f"{'Кубический:':<25}{round(all\_sum\_cube[-1], 2):<10} | {sum\_win\_cube:<5}")

**Вывод**

Проведя анализ результатов экспериментов, можно сделать следующий вывод: модифицированный алгоритм построения расписания с произвольной загрузкой демонстрирует более высокую эффективность по сравнению с классической версией алгоритма. Это подтверждается как меньшим временем поиска оптимального решения, так и более равномерным распределением заданий между устройствами, что приводит к улучшению общей производительности системы.

Литература

1. Поспелов Д.А. “Введение в теорию вычислительных машин” – M.: “Советское радио”, 1972

2. Пашкеев С.Д. “Основы мультипрограммирования для специализированных вычислительных машин” – M.: “Советское радио”, 1964

3. Плотников В.Н., Зверев В.Ю. “Техническая кибернетика №3” M., 1974

4. Бондаренко А.Т., Сапатый П.С. “Техническая кибернетика №4” –Киев, 1975