МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЕ РФ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**“ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”**

Кафедра «Программное обеспечение вычислительной техники и

автоматизированных систем»

ОТЧЕТ

По дисциплине: «Эвристические методы и алгоритмы»

На тему «Теория расписаний»

Выполнил студент группы ВПР32

Ли Н. С.

Проверил:

Проф. Кобак В. Г.

Ростов-на-Дону

2022

# Введение

Задачи проектирования и управления в системах, для которых необходимо распределение работы между параллельно работающими разнородными вычислительными устройствами занимают значимое место в теории построения расписаний. Практическая актуальность таких задач определяется существенными возможностями экономии машинного времени и вытекающими функциональными и эксплуатационными преимуществами.

Теоретическая сложность нахождения наилучшего распределения связана с необходимостью решения экстремальных задач комбинаторного типа, требующих больших вычислительных ресурсов, так что эффект от нахождения близкого к оптимальному, с точки зрения времени выполнения, распределения может быть сведен на нет затратами на его получение.

В настоящем руководстве приводятся методы получения расписаний, приводящие к небольшим затратам на вычисление за счет отказа от получения оптимального решения, но в тоже время позволяющие найти приемлемое решение, близкое к оптимальному.

# Постановка задачи

Имеется  независимых работ , которые необходимо распределить на  параллельно работающих разнородных устройств  по критерию , где - время завершения работы процессора . Каждое устройство  выполняет только одну работу в определенный момент времени и выполнение задания не прерывается для передачи на другой процессор. Известно (вес) время выполнения  задания  на любом из устройств . Требуется найти такое распределение заданий по процессорам, при котором суммарное время выполнения заданий на каждом из процессоров было бы минимальным.

Получение оптимального распределения в такой постановке приводит к громоздким вычислениям, требующим значительного времени машинного счета, поэтому цель – продемонстрировать алгоритмы, с помощью которого можно находить с малыми затратами достаточно приемлемое решение.

## Алгоритм построения расписания с произвольной загрузкой

Описанный ниже метод более эффективен по скорости поиска приемлемого по точности решения.

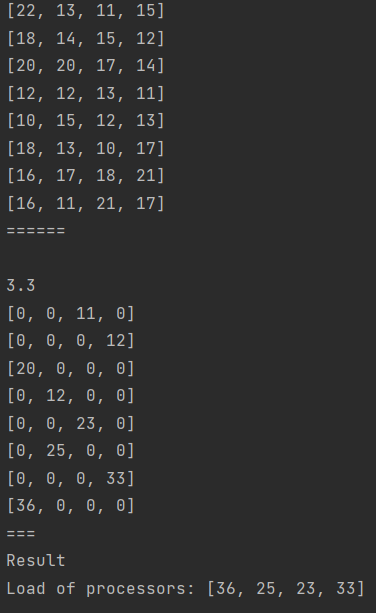
Дана прямоугольная матрица .

Ш.1 Упорядочим строки матрицы *T* по убыванию сумм всех их элементов.

Ш.2 В преобразованной матрице *T’* первой строке и найдем в ней минимальный элемент. Примем этот элемент за элемент распределения и прибавим его к соответствующему элементу следующей строки.

Ш.3 Следующая строка теперь учитывает предыдущее решение. Выберем из нее минимальный элемент, прибавим его к соответствующему элементу третьей строки и т.д.

Результат просчёта представлен на рис. 1:



## Рис. 1 – Результат работы Алгоритма построения расписания с произвольной загрузкой

1. **Код программы**

import random

import copy

N = 4

M = 8

t1, t2 = 10, 22

m = [[random.randint(t1, t2) for i in range(N)] for j in range(M)]

for arr in m:

print(arr)

print("======")

def min\_alg(matrix, proc\_count):

processors = [0 for i in range(proc\_count)]

sch = ["" for i in range(proc\_count)]

for arr in matrix:

min\_i = arr.index(min(arr))

processors[min\_i] += arr[min\_i]

sch[min\_i] += f" {arr[min\_i]}"

for arr in matrix:

print(f"{arr} {sum(arr)}")

print(sch)

print(processors)

def schedule\_alg(matrix, proc\_count):

c\_matrix = copy.deepcopy(matrix)

processors = [0 for i in range(proc\_count)]

sch = ["" for i in range(proc\_count)]

for i in range(len(c\_matrix)):

min\_i = c\_matrix[i].index(min(c\_matrix[i]))

processors[min\_i] = c\_matrix[i][min\_i]

for j in range(len(c\_matrix[i])):

if j != min\_i:

c\_matrix[i][j] = 0

for j in range(i+1, len(c\_matrix)):

c\_matrix[j][min\_i] += c\_matrix[i][min\_i]

print(c\_matrix[i])

# sch[min\_i] += f"{c\_matrix[i][min\_i]} "

print("===\nResult")

print(f"Load of processors: {processors}")

# print(f"Schedule: {sch}")

return c\_matrix

def to\_down(matrix):

c\_matrix = copy.deepcopy(matrix)

for j in matrix:

for i in range(len(c\_matrix)-1):

if sum(c\_matrix[i]) < sum(c\_matrix[i+1]):

c\_matrix[i], c\_matrix[i+1] = c\_matrix[i+1], c\_matrix[i]

for arr in c\_matrix:

print(f"{arr} {sum(arr)}")

print("===")

return c\_matrix

def to\_up(matrix):

c\_matrix = matrix

for j in matrix:

for i in range(len(c\_matrix)-1):

if sum(c\_matrix[i]) > sum(c\_matrix[i+1]):

c\_matrix[i], c\_matrix[i+1] = c\_matrix[i+1], c\_matrix[i]

for arr in c\_matrix:

print(f"{arr} {sum(arr)}")

print("===")

return c\_matrix

print("\n3.3")

schedule\_alg(copy.deepcopy(m), N)

print("\nОтсортировано по убыванию")

schedule\_alg(to\_down(copy.deepcopy(m)), N)

print("\nПо возрастанию")

schedule\_alg(to\_up(copy.deepcopy(m)), N)

print("\nmin")

min\_alg(copy.deepcopy(m), N)

1. **Блок-схема программы**

На рис. 2 представлена блок-схема программного средства

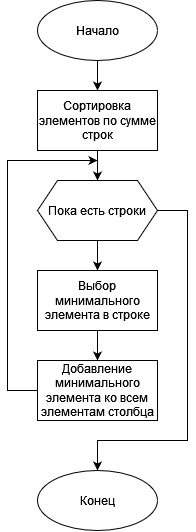


Рис. 2 Блок-схема

1. Вывод

В ходе выполнения работы было реализовано программное средство, реализующее алгоритм метода построения расписания с произвольной нагрузкой, выполнена задача в соответствии с вариантом, по методическим указаниям.

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