

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО**

**ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«Донской государственный технический университет»**

**(ДГТУ)**

Кафедра «Программное обеспечение вычислительной техники и

автоматизированных систем»

**ЛАБОРАТОРНАЯ РАБОТА № 2**

по дисциплине «Эвристические методы и алгоритмы»

тема: «Теория расписаний. Алгоритм построения расписания с произвольной загрузкой»

Выполнил:

ст. гр. ВПР 31 И.С. Недомерков

Проверил:

д.н., профессор В. Г. Кобак

Ростов-на-Дону

2020

**1. 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.

**2. 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.

**3. Ручной просчет**

Значения  задаются матрицей размером , где *i*-номер алгоритма, *j*-номер процессора. , где , . Исходная матрица времени выполнения работ:



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.

Решим задачу рассматриваемым алгоритмом*:* выполнив Ш.1 множество заданий примет вид:

Сложили элементы в строках матрицы.



Упорядочиваем строки в порядке убывания по суммам, матрица примет вид:



Согласно Ш.2 строим расписание (справа вверху над элементом указывается суммарная загрузка процессора в столбце):





Выполнив алгоритм, получим расписание:

 

Результат: =20

**4. Реализации алгоритма половинного деления множества заданий Python:**

from random import randint

def create\_matrix(N, M, down, up):

return [[randint(down, up) for j in range(N)] for i in range(M)]

def output(matrix):

for i in range(len(matrix)):

sum = 0

for j in range(len(matrix[i])):

sum += matrix[i][j]

print(matrix[i][j], end=' ')

print('sum = ', sum)

def rasp(matrix):

tmp = [0] \* N

i = 0

while i < M:

for j in range(len(result)):

tmp[j] = result[j] + matrix[i][j]

result[tmp.index(min(tmp))] = tmp[tmp.index(min(tmp))]

i += 1

print('T = ', result, 'Max = ', max(result))

def minimal(matrix):

result = [0] \* N

i = 0

while i < M:

result[matrix[i].index(min(matrix[i]))] += matrix[i][matrix[i].index(min(matrix[i]))]

i += 1

print('T = ', result, 'Max = ', max(result))

def Sort(matrix):

matrix = sorted(matrix, key=lambda matrix: sum((int(matrix[i]) for i in range(0, int(len(matrix))))))

output(matrix)

return matrix

def Sort2(matrix):

matrix = sorted(matrix, key=lambda matrix: sum((int(matrix[i]) for i in range(0, int(len(matrix))))))

matrix.reverse()

output(matrix)

return matrix

if \_\_name\_\_ == '\_\_main\_\_':

N, M, down, up = int(input('Введите число процессоров ')), int(input('Введите число заданий ')), int(input(

'Введите нижнюю границу ')), int(input('Введите верхнюю границу '))

matrix = create\_matrix(N, M, down, up)

result = [0] \* N

matrix2 = matrix.copy()

print('Исходный матрица: ')

output(matrix)

rasp(matrix)

print('Матрица повозрастанию: ')

Sort(matrix)

rasp(matrix)

print('Матрица поубыванию: ')

Sort2(matrix)

rasp(matrix)

print('Исходная матрица и алгоритм минимального элемента:')

output(matrix2)

minimal(matrix2)

**5. Результаты, выводы**

В ходе лабораторной работы была написана программа, реализующая Алгоритм построения расписания с произвольной, в которой количество процессоров и количество заданий вводятся с клавиатуры. Матрица генерируется рандомно, нижний и верхний предел для рандома задаются с клавиатуры пользователем.

Также было произведено сравнение работы алгоритма при сортированном списке заданий, при случайном и при обратном списке. Выяснили, что алгоритм позволяет получить точной результат, при использовании списка по убыванию. Остальные результаты являются приближенными.

Результат работы программы:







