

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ 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 Д. С. Кононов

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 суммарнаязагрузка процессора в столбце):

Выполнив алгоритм, получим расписание:

Результат: =20

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

fclass Krit:

def \_\_init\_\_(self, n, m):

self.mass = []

for i in range(m):

self.mass.append(randint(10, 22))

self.n = n

def \_\_sum(self, mass):

res = 0

for i in mass:

res += i

return res

def \_\_union(self, mass):

tmp = [self.\_\_sum(i) for i in mass]

return tmp.index(min(tmp))

def \_\_union2(self, mass):

tmp = [self.\_\_sum(i) for i in mass]

return max(tmp)

def \_\_krit(self, mass):

p = [[0] for i in range(self.n)]

for i in self.mass:

p[self.\_\_union(p)].append(i)

# print(p)

for i in range(len(p)):

print('p{} = {}; {}'.format(i + 1, self.\_\_sum(p[i]), p[i]))

print(p)

print('Tmax = {}'.format(self.\_\_union2(p)))

print('\n', '-' \* 20, '\n', sep='')

def krit\_withot\_sort(self):

print('T = {}'.format(self.mass))

self.\_\_krit(self.mass)

def krit\_ascending(self):

mass = Sort.t\_sort(self.mass.copy())

print(self.\_\_print\_matrix(mass))

print('T = {}'.format(mass))

self.\_\_krit(mass)

def krit\_descending(self):

mass = list(reversed(Sort.t\_sort(self.mass.copy())))

print(self.\_\_print\_matrix(mass))

print('T = {}'.format(mass))

self.\_\_krit(mass)

def \_\_print\_matrix(self, mass):

string = ''

for i in mass:

string += '{} '.format(i) \* self.n + '\n'

return string

**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:



