

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 № 1**

по дисциплине «Эвристические методы и алгоритмы»

тема: «Алгоритмы списочных расписаний. Алгоритм критического пути»

Выполнил:

ст. гр. ВПР 31 Д. С. Кононов

Проверил:

д.н., профессор В. Г. Кобак

Ростов-на-Дону

2020

**1. 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.

Программирование для многопроцессорных машинных систем связано с распараллеливанием и синхронизацией вычислений и организацией выполнения параллельных вычислительных процессов. Это выдвигает целый ряд сложных задач, среди которых весьма важными являются, расчет характеристик времени и количества операций, требующихся для выполнения параллельных программ, и построения расписаний (планов), выполнения параллельных программ на многопроцессорных и многомашинных вычислительных системах.

Модели параллельных программ и операционные характеристики процессов их выполнения служат основой для планирования параллельных вычислительных процессов, т.е. для построения расписаний указанных процессов. Расписания параллельных вычислительных процессов определяют порядок выполнения программы на вычислительной системе, включая распределение частей программы по процессам. С увеличением числа распределяемых частей программ и количества используемых процессоров сложность построения оптимальных расписаний обычно резко возрастает. Поэтому важное значение имеют простые в построении и удобные в реализации приближенные расписания параллельных вычислительных процессов, близкие к оптимальным с точки зрения времени выполнения параллельных программ.

# 2. Постановка задачи

Имеется вычислительная система (ВС), состоящая из *N* несвязанных идентичных устройств (приборов, процессоров и т.п.)

На обслуживание в ВС поступает набор из *M* независимых параллельных заданий (работ) известно время решения задания на любом из устройств. При этом каждое задание может выполняться на любом из устройств (процессоре), в каждый момент времени отдельный процессор обслуживает не более одного задания и выполнение задания не прерывается для передачи на другой процессор. Требуется найти такое распределение заданий по процессорам, при котором суммарное время выполнения заданий на каждом из процессоров было бы минимальным. Под расписанием следует понимать отображение , такое что, если , то говорят что задание , в расписании *R* назначенного на процессор . При сделанных выше допущениях, расписание можно представить разбиением множества заданий *T* на *N* непересекающихся подмножеств

Критерий, используемый для минимизации времени завершения обслуживания заданий, является минимальным критерием и определяется в следующем виде: , где

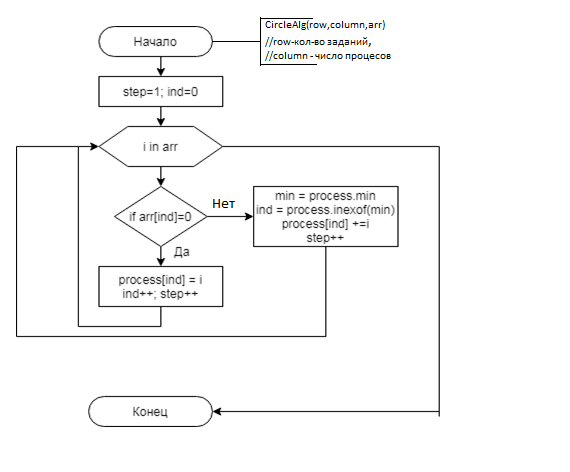
- время завершения работы процессора .

**3. Теоретическая часть.**

Простой и эффективный алгоритм списочного расписания: метод критического пути.

Имеющуюся однородную матрицу упорядочивают по убыванию. Сначала процессы получают задания, идущие по порядку. Значение нагрузки необходимо пересчитывать после каждого добавления задания на один из процессоров. Следующие задания направляются на наименее загруженный из всех процессоров. При наличии одинаковых значений нагрузки на нескольких процессорах, приоритет получения следующего задания принадлежит тому процессору, который ближе к первому (по порядковому номеру).

Блок схема алгоритма:



**4. Реализации алгоритма критического пути множества заданий Python**

from random import randint

class Sort:

@staticmethod

def \_\_find(mass\_t, res):

if len(mass\_t) == 2:

if mass\_t[0] > mass\_t[1]:

res.append(mass\_t[1])

return

else:

res.append(mass\_t[0])

return

new\_mass\_t = []

for i in range(len(mass\_t) // 2):

if mass\_t[i\*2] > mass\_t[i\*2+1]:

new\_mass\_t.append(mass\_t[i\*2+1])

else:

new\_mass\_t.append(mass\_t[i\*2])

if len(mass\_t) % 2 == 1:

new\_mass\_t.append(mass\_t[-1])

Sort.\_\_find(new\_mass\_t, res)

@staticmethod

def t\_sort(mass\_t):

res = []

for i in range(len(mass\_t)):

Sort.\_\_find(mass\_t, res)

mass\_t[mass\_t.index(res[-1])] = 9999

return res

class Krit:

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

self.mass = []

for i in range(m):

self.mass.append(randint(10, 100))

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 \_\_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('\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

def \_\_str\_\_(self):

string = ''

for i in self.mass:

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

return string

a = Krit(8, 14)

print(a)

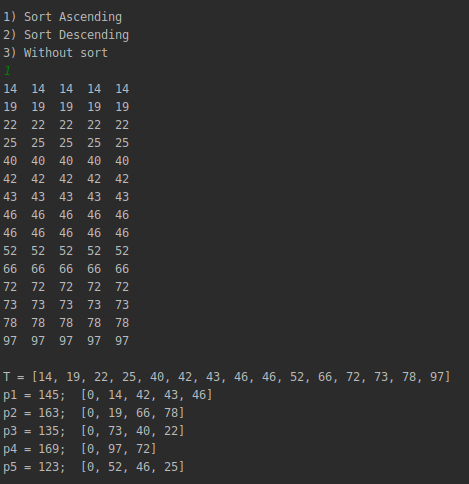
a.krit\_withot\_sort()

a.krit\_ascending()

a.krit\_descending()

**5. Результаты, выводы**

В ходе лабораторной работы была написана программа, реализующая алгоритм критического пути множества заданий, в которой количество процессоров и количество заданий вводятся с клавиатуры. Матрица генерируется рандомно, сортировка проводится с помощью алгоритма турнирной сортировки.

Результат работы программы: