

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ 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 № 6**

Выполнил:

Студент группы ВПР31

Андросов И.А.

Ростов-на-Дону  
 2023

# Введение

# Область исследования расписаний занимается задачами проектирования и управления в различных системах, где необходимо найти оптимальное значение критериев функционирования с учетом ограничений. В программировании для многопроцессорных систем важны расчет характеристик времени и количества операций, построение расписаний и выполнение параллельных программ. Модели параллельных программ и операционные характеристики процессов их выполнения используются для планирования параллельных вычислительных процессов. С увеличением числа распределяемых частей программ и используемых процессоров сложность построения оптимальных расписаний возрастает, поэтому важны простые и удобные в реализации приближенные расписания, близкие к оптимальным по времени выполнения параллельных программ.

# Постановка задачи

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

На обслуживание в ВС поступает набор из  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 , назначенного на процессор . При сделанных выше допущениях расписание можно представить разбиением множества заданий на  непересекающихся подмножеств , где .

Критерий, используемый для минимизации времени завершения обслуживания заданий, является минимальным критерием и определяется в следующем виде: , где  - время завершения работы процессора .

1. **Реализация программного модуля**

Требуемый программный модуль реализован на языке Python.

При запуске программы пользователь должен ввести все необходимые параметры, такие как количество задач, количество машин, диапазон генерации значений, количество особей в популяции, вероятность на кроссовер и мутацию.

После определения начальных параметров, создаётся список, который заполняется случайными значениями. Кроме того, в классе поколения создаётся разбиение, которое определит, на какую машину будет определена задача. Кроме того, создаётся начальное поколение. В каждой особи хранится список задач, расписание и фенотип.

После создания начального поколения запуститься основной алгоритм. В ходе основного алгоритма происходит скрещивание путём обмена значениями распределения. При скрещивании выбирается особь под номером i и скрещивается со случайной особью. Если произойдёт мутация, то у особи инвертируется случайный бит распределения задачи. После этого выбирается особь с лучшим фенотипом среди текущей особи и её потомков.

Данный процесс повторяется, пока лучший фенотип в поколении не повторится определённое количество раз.

## Листинг программы:

import random

import numpy as np

n = int(input('Число процессоров '))

m = int(input('Число задач '))

t1 = int(input('Минимальное время процесса '))

t2 = int(input('Максимальное премя процесса '))

osob = int(input('Число особей '))

finalCount = int(input('Количество повторений '))

verCross = int(input('Вероятность кросовера '))

verMut = float(input('Вероятность мутации '))

f = open('./result.txt', 'w')

def selection(population, el):

index = 0

while(True):

index = random.randint(0, len(population) - 1)

if index == el:

continue

else:

break

return population[index]

def crossover(parent1, parent2, crossover\_max, crossover\_min):

child = parent1[:crossover\_min] + parent2[crossover\_min:crossover\_max] + parent1[crossover\_max:]

f.write('\n Результат кросовера:\n' + ' '+ ' \t'.join([str(i) for i in child]))

return child

def mutation(individual, chanse):

ver = random.randint(0, 100)

f.write('\n\nвероятность мутации: ' + str(ver))

# f.write('\nMut ver: ' + str(ver))

if(ver <= chanse):

# Берем число

numIndex = random.randint(0, len(individual) - 1)

number = individual[numIndex]

# Переводим в бинарку

binary\_number = bin(number)[2:].zfill(8)

# Берем индексы битов

random\_index1 = 0

random\_index2 = 0

while(True):

random\_index1 = random.randint(0, 7)

random\_index2 = random.randint(0, 7)

if(random\_index1 == random\_index2):

continue

else:

break

# Меняем индексы местами

new\_str = ''

for i in range(len(binary\_number)):

if i == random\_index1:

new\_str = new\_str + binary\_number[random\_index2]

elif i == random\_index2:

new\_str = new\_str + binary\_number[random\_index1]

else:

new\_str = binary\_number[i]

# Переводим число обратно в десятичное

number = int(new\_str, 2)

# Возвращаем в ребенка

individual[numIndex] = number

# f.write( '\n' + '\t'.join([str(i[0]) + ' ' + str(i[1]) for i in individual]) + '\n')

return individual

def raspred\_mass(raspred\_value, individual, n, initMass):

zero\_mass = [0] \* n

for i in range(len(individual)):

zero\_mass[int(individual[i] / raspred\_value)] += initMass[int(individual[i] / raspred\_value)][i]

return zero\_mass

def fitness(individual, raspred\_value, n, initMass):

return max(raspred\_mass(raspred\_value, individual, n, initMass))

def genetic\_algorithm():

raspred\_value = int(256/n)

counter = 0

popul\_count = 0

lastMin = 0

fitness\_scores = []

f.write('\n\nМаксимальные веса:\n' + ' '.join([str(i) for i in fitness\_scores]))

while True:

popul\_count += 1

if(counter == finalCount - 1):

break

save\_population = population[:]

for el in range(len(save\_population)):

# Выбираем 2-х родителей

parent1 = save\_population[el][:]

parent2 = selection(save\_population, el)

f.write('\n Родитель 1:\n' + '\t'.join([str(i) for i in parent1]))

f.write('\n Родитель 2:\n' + '\t'.join([str(i) for i in parent2]))

# Пройдет ли кроссовер

ver\_cross = random.randint(0, 100)

f.write('\n\nВероятность кроссовера: ' + str(ver\_cross))

#Заранее объявляем детей

child1 = []

child2 = []

# Если прошел выполняем

if(ver\_cross <= verCross):

crossover\_point1 = 0

crossover\_point2 = 0

else:

child1 = parent1

child2 = parent2

child1 = mutation(child1, verMut)

f.write('\nМутация результат:\n' + '\t'.join([str(i) for i in child1]))

child\_fitness1 = fitness(child1, raspred\_value=raspred\_value, n=n, initMass=initMass)

child\_fitness2 = fitness(child2, raspred\_value=raspred\_value, n=n, initMass=initMass)

pic\_child = []

if(child\_fitness1 >= child\_fitness2):

pic\_child = child2

else:

pic\_child = child1

population.append(pic\_child)

fitness\_scores = [ fitness(individual, raspred\_value=raspred\_value, n=n, initMass=initMass) for individual in population]

f.write('\n\nПопуляция:' + str(popul\_count) +'\n')

f.write('\nКонечные веса:\n' + ' '.join([str(i) for i in fitness\_scores]))

temp = np.array(fitness\_scores)

min\_indexes = np.argsort(temp, axis=None)[:osob]

new\_temp = [population[i] for i in min\_indexes]

f.write('\n\nРезультат до чистки:\n' + '\n'.join([" ".join([str(el) for el in i]) for i in population]))

population = new\_temp

f.write('\n\nРезультат популяции:\n' + '\n'.join([" ".join([str(el) for el in i]) for i in population]))

if not lastMin == 0 and lastMin == min(fitness\_scores):

counter += 1

else:

counter = 0

lastMin = min(fitness\_scores)

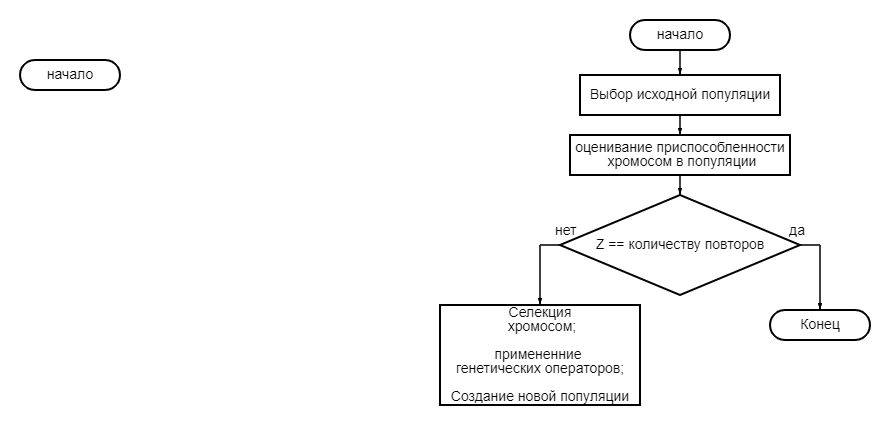
best\_individual = min(fitness\_scores)

return best\_individual

print(genetic\_algorithm())

f.close()

**5. Блок-схема**



**6.Результат работы программы**  
Конечные веса:

70 71 59 73 87 52 59 59 74 55

Результат до чистки:

226 113 117 22 212 9 44 58 27 148 202 166

213 129 184 72 232 128 252 215 151 35 139 221

104 13 89 187 166 73 167 85 3 38 15 35

65 178 82 125 52 151 47 16 188 23 72 1

108 77 28 102 41 65 53 142 237 81 125 66

226 2 117 22 212 9 252 215 151 35 139 166

104 13 89 187 166 73 167 85 3 38 15 35

104 13 89 187 166 73 167 85 3 38 15 1

104 13 89 187 166 151 167 85 3 38 15 35

65 178 82 125 52 65 53 142 188 23 72 1

Результат популяции:

226 2 117 22 212 9 252 215 151 35 139 166

65 178 82 125 52 65 53 142 188 23 72 1

104 13 89 187 166 73 167 85 3 38 15 35

104 13 89 187 166 73 167 85 3 38 15 35

104 13 89 187 166 73 167 85 3 38 15 1

Родитель 1:

226 2 117 22 212 9 252 215 151 35 139 166

Родитель 2:

104 13 89 187 166 73 167 85 3 38 15 35

Вероятность кроссовера: 16

Результат кросовера:

226 2 89 187 166 73 167 85 3 38 139 166

Результат кросовера:

104 13 117 22 212 9 252 215 151 35 15 35

вероятность мутации: 67

# 7. Вывод

# Исследования в области расписаний занимаются проектированием и управлением в различных системах с целью нахождения оптимальных значений критериев функционирования при ограничениях. В программировании многопроцессорных систем важны расчеты времени и количества операций, построение расписаний и выполнение параллельных программ. Для планирования параллельных вычислительных процессов используются модели параллельных программ и операционные характеристики процессов их выполнения. Сложность построения оптимальных расписаний возрастает с увеличением числа распределяемых частей программ и используемых процессоров, поэтому важны простые и удобные в реализации приближенные расписания, близкие к оптимальным по времени выполнения параллельных программ. Распределенные системы являются эффективным инструментарием для решения сложных задач, а целью их использования является оптимизация использования ресурсов и упрощение работы пользователя. Был разработан программный модуль, который может решать задачу генетическим алгоритмом для любого количества задач, машин, количества индивидов в поколении и критерии останова.

# 8. Литература

1. Коффман Э.Г. Теория расписания и вычислительные машины. – M.: Наука, 1987.