

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ 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 №3**

по дисциплине «Эвристические методы и алгоритмы»

**Выполнила:**

студентка уч. гр. ВМО31

Басенко Белла Валерьевна

**Проверил:**

проф. Кобак В. Г.

Ростов-на-Дону

2024

**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.

**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 

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

- время завершения работы процессора .

**Алгоритмы решения распределительной задачи**

Метод Крона.

Принцип действия алгоритма Крона можно описать в два этапа. Первый этап заключается в случайном распределении множества заданий на множество приборов, второй этап уточняет полученное распределение.[1] Алгоритм первого этапа можно представить в виде последовательности шагов:

Ш1. Сгенерировать случайное число pi – номер прибора, равномерно распределённое на интервале [1,N]. Инициализировать j = 1.

Ш2. Назначить на прибор с номером pi j-ое задание. Увеличить j = j + 1.

Ш3. Проверить, больше ли значение j количества заданий M. Если условие выполняется, то прекратить распределение, а иначе перейти к шагу 1.

Второй этап – уточнение, включает следующие шаги:

Ш1. Вычислить время загрузки каждого прибора {Tj} ( j = 1,..,N ) путем суммирования времен выполнения заданий загруженных в каждый j-й прибор.

Ш2. Из полученного множества {Tk} выбрать номера приборов с максимальным Tmax и минимальным Tmin значениями из набора {Tk} соответственно.

Ш3. Для каждого прибора с минимальной и максимальной загрузкой проверить условие по принципу каждый с каждым

tkmax – tlmin < Δ,

где Δ = Tmax - Tmin, k,l = 1,.., M. При этом tkmax > tlmin.

Если условие выполняется, то перейти к следующему шагу, а иначе прекратить выполнение алгоритма.

Ш4. Переставить значения tkи tl местами и перейти к шагу 1.

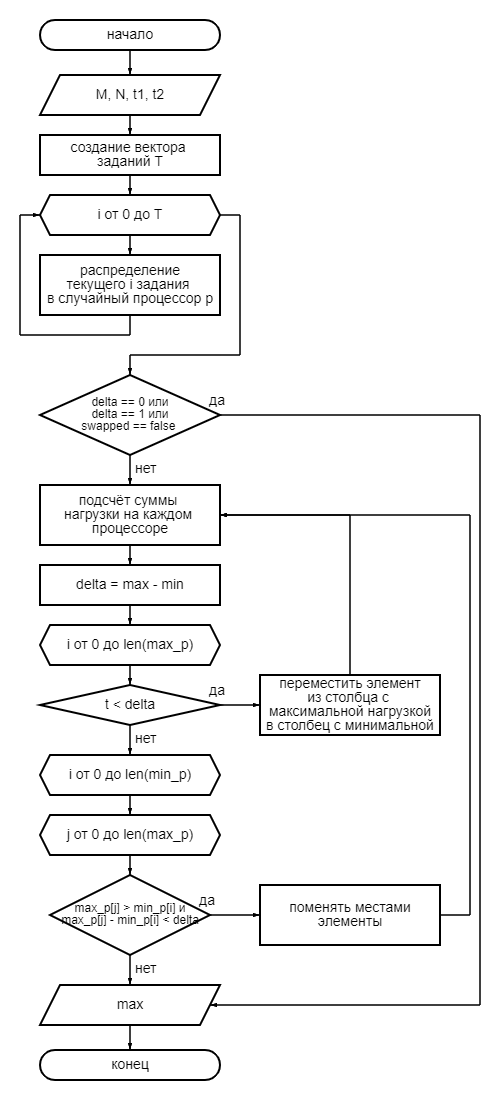


Рисунок 1 – Схема метода Крона

**Вычислительные эксперименты**

**A screenshot of a computer program

Description automatically generated**

**Листинг программы**

import random, numpy as np

def Krohn(matrix, sorted=0):

print("начальная матрица:", matrix)

if sorted == 1:

for i in matrix: i.sort(reverse=True)

print("отсортированная матрица:", matrix)

while True:

summ = [sum(row) for row in matrix]

print("матрица:", matrix)

print("суммы строк:", summ)

max\_index = summ.index(max(summ))

min\_index = summ.index(min(summ))

delta = max(summ) - min(summ)

swapped = False

for elem in matrix[max\_index]:

if elem < delta:

matrix[max\_index].remove(elem)

matrix[min\_index].append(elem)

swapped = True

break

if not swapped:

for i in range(len(matrix[min\_index])):

for j in range(len(matrix[max\_index])):

if matrix[max\_index][j] > matrix[min\_index][i] and matrix[max\_index][j] - matrix[min\_index][i] < delta:

matrix[max\_index][j], matrix[min\_index][i] = matrix[min\_index][i], matrix[max\_index][j]

swapped = True

break

if swapped: break

if not swapped or delta in (0, 1): break

print("полученная матрица:")

for row in matrix: print(row)

print(max(summ), summ)

N, M = int(input("N = ")), int(input("M = "))

print("введите границы диапазона:")

t1, t2 = int(input("t1 = ")), int(input("t2 = "))

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

print(T)

matrix = [[] for \_ in range(N)]

processors = []

print("\n1. Метод Крона\n2. Метод Крона с сортировкой по убыванию\n3. CMP + Метод Крона\n4. CMP + Метод Крона с сортировкой по убыванию")

choice = int(input("ВВОД: "))

match choice:

case 1:

for i in T: matrix[random.randint(0, N - 1)].append(i)

Krohn(matrix)

case 2:

for i in T: matrix[random.randint(0, N - 1)].append(i)

Krohn(matrix, sorted=1)

case 3:

for i in range(0, N):

matrix[i].append(T[i])

processors.append(T[i])

for i in range(N, len(T)):

matrix[processors.index(min(processors))].append(T[i])

Krohn(matrix)

case 4:

for i in range(0, N):

matrix[i].append(T[i])

processors.append(T[i])

for i in range(N, len(T)):

pi = processors.index(min(processors))

matrix[pi].append(T[i])

processors[pi] += T[i]

Krohn(matrix, sorted=1)

**Вывод**

Проанализировав данные, полученные входе экспериментов, можно сделать вывод, что метод Крона дает более оптимальный результат при использовании его модификации, включающей метод критического пути с сортировкой по убыванию веса заданий.

Литература

1. Коффман Э.Г. “Теория расписания и вычислительные машины” – M.: “Наука”, 1987
2. Кобак В.Г., Титов Д.В., Золотых О.А.“ Порядок совмещения алгоритма Крона и его модификации в однородных системах обработки информации ” Системный анализ, управление и обработка информации- тр. 2-го междунар. семинара студентов, аспирантов и ученых / ДГТУ. - Ростов-н/Д, 2011.
3. Кобак В.Г., Титов Д.В., Золотых О.А. Исследование алгоритма Крона и его модификации при различных исходных данных // Вестник Дон. гос. техн. ун-та. – 2012. – № 8.