

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 Крона (модификации)»

Выполнил:

ст. гр. ВПР-32 И.В. Гузиков

Проверил:

д.т.н., доцент В.Г. Кобак

Ростов-на-Дону

2020

**Цель работы:** Понять принцип работы алгоритма Крона и различных его модификаций. Программно реализовать данные алгоритмы.

# 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. 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.)

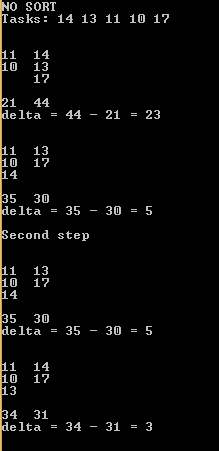
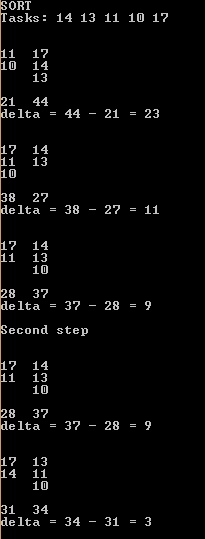
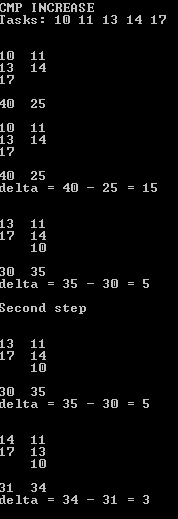
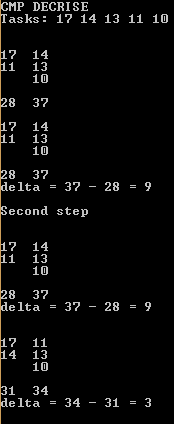


На обслуживание в ВС поступает набор из  независимых параллельных заданий (работ)  известно время решения  задания  на любом из устройств. При этом каждое задание может выполняться на любом из устройств (процессоре), в каждый момент времени отдельный процессор обслуживает не более одного задания и выполнение 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 

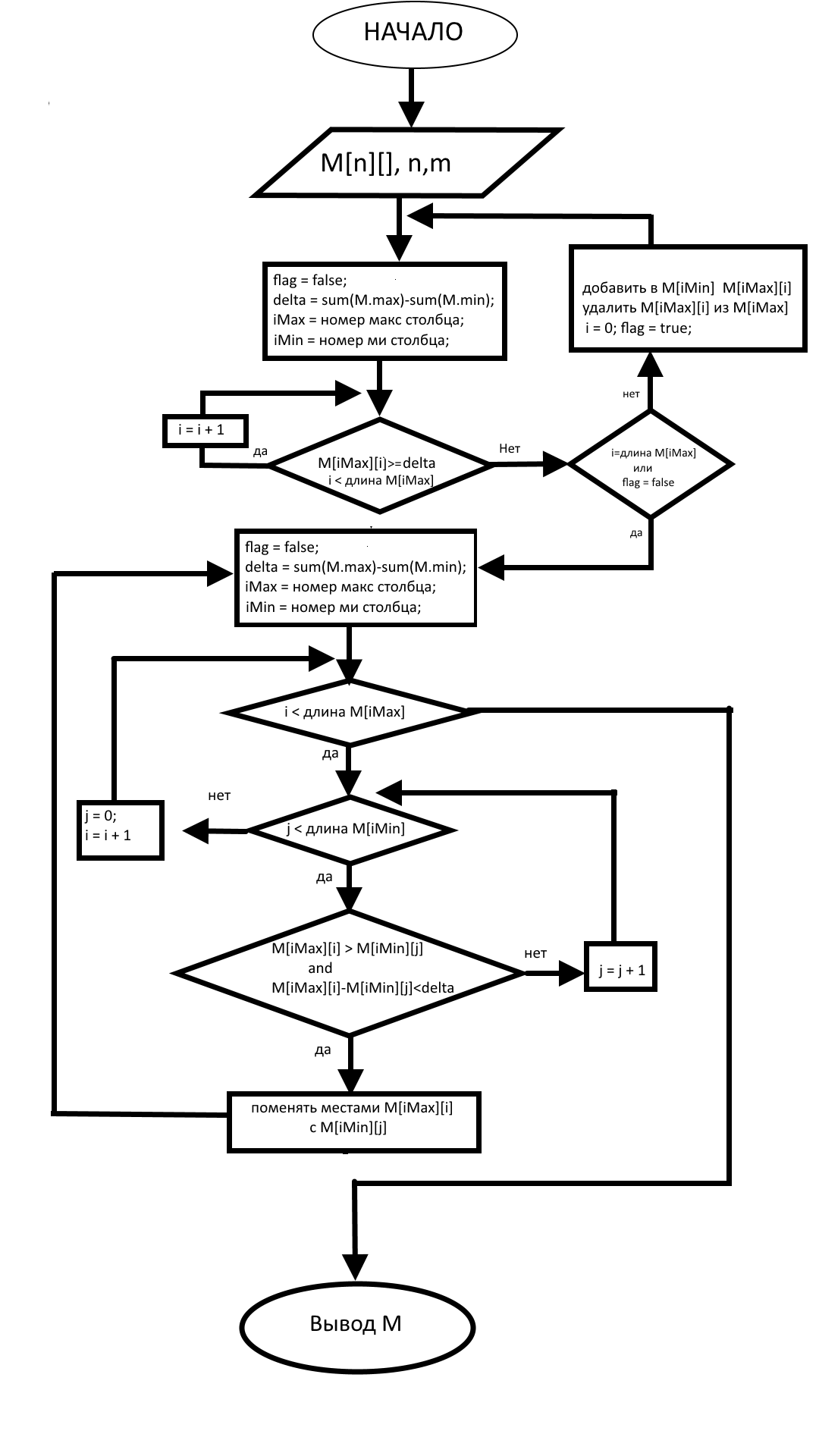
Критерий, используемый для минимизации времени завершения обслуживания заданий, является минимальным критерием и определяется в следующем виде: , где

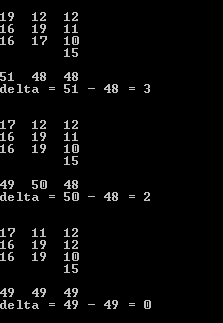
- время завершения работы процессора .

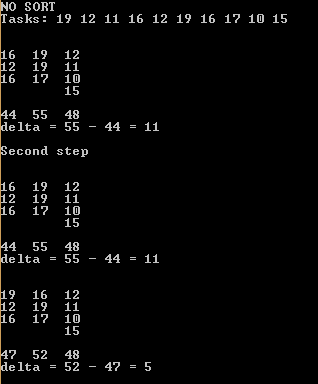
**Пример работы программы**

# БЛОК СХЕМА АЛГОРИТМА



**РУЧНОЙ ПРОСЧЕТ**



**19** 12 12

16 19 11

16 **17** 10

15

----------------------

51 48 48

17 **12** 12

16 19 **11**

16 19 10

15

----------------------

49 50 48

17 11 12

16 19 12

16 19 10

15

----------------------

49 49 49

16 19 12

12 19 11

16 17 10

15

----------------------

44 55 48

**16** **19** 12

12 19 11

16 17 10

15

----------------------

44 55 48

19 **16** 12

**12** 19 11

16 17 10

15

----------------------

47 52 48

**Выводы:**

В рамках данной лабораторной работы был изучен и программно реализован алгоритм Крона и три его модификации. Программное средство было протистированно для четырё случаев: без сортировки сортировка задач по возростанию, начальное распределения задач методами критического пути при сортировке начальных задач по возрастанию и убыванию. В результате тестирования алгоритма было выяснено, что наиболее эффективной модификацией является использование для начального распределения задач **метода критического пути при отсортированных задачах по возрастанию.**

# Литература

1. Коффман Э.Г. “Теория расписания и вычислительные машины” – M.: “Наука”, 1987
2. Романовский И.В. “Алгоритмы решения экстремальных задач” – М.: “Наука”, 1977
3. Пашкеев С.Д., Минязов Р.И., Могилевский В.Д. “Машинные методы оптимизации в технике связи” – М.: “Связь”, 1976.

**КОД ВЫЧИСЛЯЮЩЕЙ ПРОЦЕДУРЫ**

static void Kron(List<List<int>> table,int N, int M, bool sort=false)

{

List<int> sums = new List<int>();

if (sort == true) RowSort(ref table);

PrintTable(table);

while (true)

{

bool flag = false;

sums.Clear();

for (int i = 0; i < N; i++) { sums.Add(table[i].Sum()); }

int delta = sums.Max() - sums.Min();

Console.WriteLine($"\ndelta = {sums.Max()} - {sums.Min()} = {delta}");

int minCol = Index(sums, sums.Min()), maxCol = Index(sums, sums.Max());

for (int i = 0; i < table[maxCol].Count; i++)

{

if (table[maxCol][i] < delta)

{

table[minCol].Add(table[maxCol][i]);

table[maxCol].RemoveAt(i);

flag = true;

break;

}

}

if (sort == true) RowSort(ref table);

if (flag == false) break; else PrintTable(table);

}

Console.WriteLine("\nSecond step");

PrintTable(table);

while (true)

{

bool flag = false;

sums.Clear();

for (int i = 0; i < N; i++) { sums.Add(table[i].Sum()); }

int delta = sums.Max() - sums.Min();

Console.WriteLine($"\ndelta = {sums.Max()} - {sums.Min()} = {delta}");

int minCol = Index(sums, sums.Min()), maxCol = Index(sums, sums.Max());

for (int i = 0; i < table[maxCol].Count; i++)

{

for (int j = 0; j < table[minCol].Count; j++)

{

if (table[maxCol][i] > table[minCol][j] && (table[maxCol][i] - table[minCol][j])<delta )

{

flag = true;

int tmp = table[maxCol][i];

table[maxCol][i] = table[minCol][j];

table[minCol][j] = tmp;

goto exit;

}

}

}

exit:

if (sort == true) RowSort(ref table);

if (flag == false) break; else PrintTable(table);

}

}