## МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЕ РФ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

## “ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Лабораторная работа №1

По дисциплине: «Эвристические методы и алгоритмы» На тему «Теория расписаний»

Выполнил :

Студент группы ВМО31

Оганесьянц К.П.

Проверил:

Кобак В.Г.

Ростов-на-Дону 2022 г.

# 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

Широкое распространение получил простой и эффективный алгоритм списочного расписания: метод критического пути (Critical Method Path или CMP).

Решим задачу №1 алгоритмом *Критического пути.* Для начала упорядочим заданную матрицу по убыванию:

|  |  |  |  |
| --- | --- | --- | --- |
| 22 | 22 | 22 | 22 |
| 14 | 14 | 14 | 14 |
| 9 | 9 | 9 | 9 |
| 7 | 7 | 7 | 7 |
| 7  5  5  5 | 7  5  5  5 | 7  5  5  5 | 7  5  5  5 |

Множество заданий примет вид: T = [22 14 9 7 7 5 5 5]. На первый процессор назначается задание с весом 22. Потом, на второй процессор назначается – 14, на третий – 9 и т.д.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Шаг 1* |  |  |  |  |  |  | *Шаг 2* |  |  |  |  |
|  | 22 |  |  |  |  |  |  | 22 | 14 |  |  |
| Нагр: | 22 | 0 | 0 | 0 |  |  |  | 22 | 14 | 0 | 0 |

Значение нагрузки необходимо пересчитывать после каждого добавления задания на один из процессоров. Следующие задания направляются на наименее загруженный из всех процессоров.

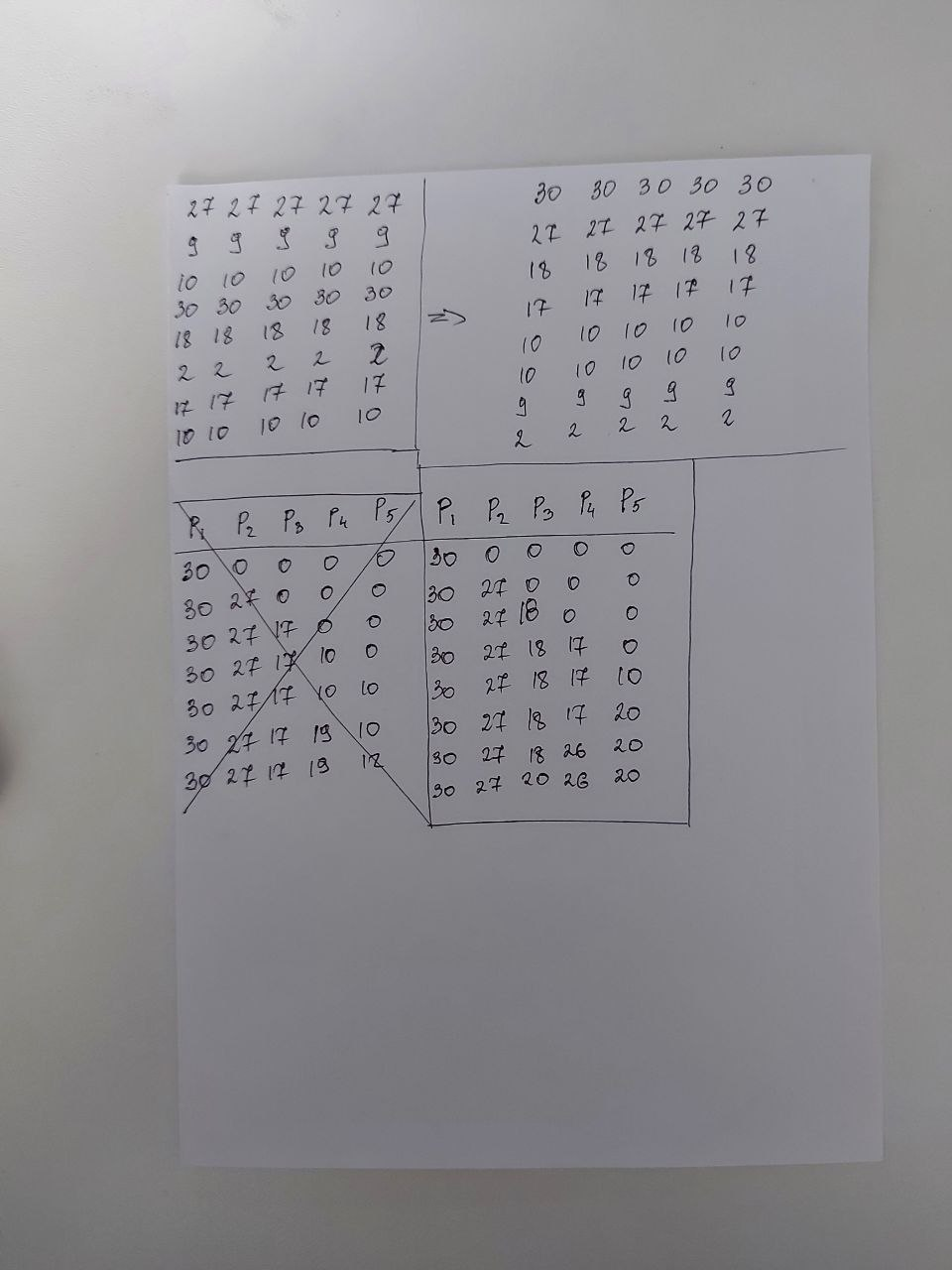
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Шаг 3* |  |  |  |  | *Шаг 4* |  |  |  |  |
|  | 22 | 14 | 9 |  |  | 22 | 14 | 9 | 7 |
| Нагр: | 22 | 14 | 9 | 0 |  | 22 | 14 | 9 | 7 |
| *Шаг 5* |  |  |  |  | *Шаг 6* |  |  |  |  |
|  | 22 | 14 | 9 | 7 |  | 22 | 14 | 9 | 7 |
|  |  |  |  | 7 |  |  |  | 5 | 7 |
| Нагр: | 22 | 14 | 9 | 14 |  | 22 | 14 | 14 | 14 |

При наличии одинаковых значений нагрузки на нескольких процессорах, приоритет получения следующего задания принадлежит тому процессору, который ближе к первому (по порядковому номеру).

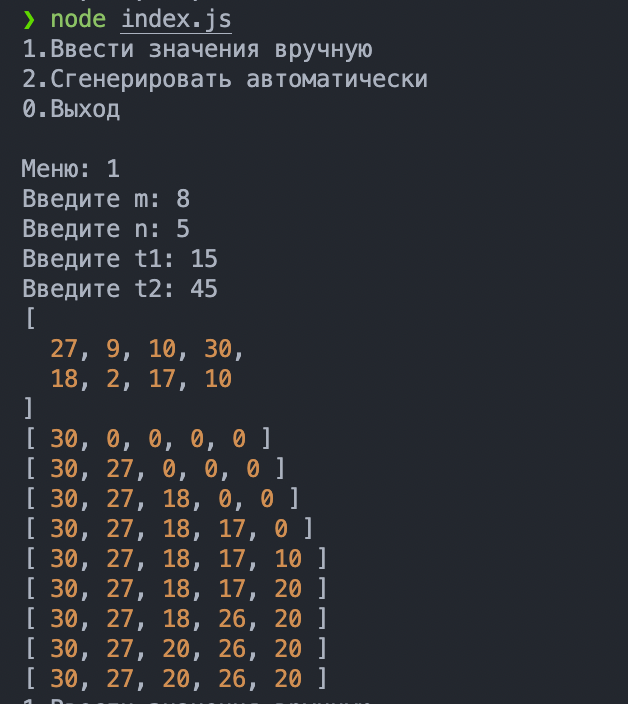
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Шаг 7* |  |  |  |  | *Шаг 8* |  |  |  |  |
|  | 22 | 14 | 9 | 7 |  | 22 | 14 | 9 | 7 |
|  |  | 5 | 5 | 7 |  |  | 5 | 5  5 | 7 |
| Нагр: | 22 | 19 | 14 | 14 |  | 22 | 19 | 19 | 14 |

Результат: max (22, 19, 19, 14)=22.

1. **Результаты тестирования программы**



Для тестирования программы были заданы параметры.



# Код программы

# Index.js(Блок тестирования)

import { calculate } from "./algorythm.js"

import readlineSync from 'readline-sync'

function randomIntFromInterval(min, max) { // min and max included

return Math.floor(Math.random() \* (max - min + 1) + min)

}

const testMenu = () => {

let choice = 123;

while(Number(choice) !== 0) {

console.log('1.Ввести значения вручную \n2.Сгенерировать автоматически \n0.Выход\n')

choice = readlineSync.question('Меню: ')

switch (Number(choice)) {

case 2:

console.log(calculate([22, 5, 7, 14, 5, 5, 9, 7], 4))

case 1:

const m = readlineSync.question('Введите m: ')

const n = readlineSync.question('Введите n: ')

const t1 = readlineSync.question('Введите t1: ')

const t2 = readlineSync.question('Введите t2: ')

const array = [];

for (let i = 0; i < m; i++) {

array.push(randomIntFromInterval(t1, t2))

}

console.log(array);

console.log(calculate(array, Number(n)))

}

}

}

testMenu();

**Algorythm.js**

export const calculate = (matrix, n) => {

let processArray = Array(n).fill(0);

matrix = matrix.sort(function(a, b) {

return b -a;

});

matrix.map(el => {

let k = 0;

let index = 0;

let minimum = processArray[0];

processArray.map(anEl => {

if (anEl < minimum){

minimum = anEl;

index = k

}

k += 1;

})

processArray[index] = processArray[index] + el;

console.log(processArray);

})

return processArray;

}

# Блок схема

****

1. **Вывод**

Алгоритм списочного расписания: метод критического пути (Critical Method Path или CMP) достаточно прост и эффективен в связи с чем и получил широкое распространение

# 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.