Министерство науки и высшего образования Российской Федерации

Государственное образовательное учреждение высшего профессионального образования

«Алтайский государственный технический университет им. И. И. Ползунова»

Факультет информационных технологий

Кафедра Прикладная математика

Отчет защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель         А. В. Сорокин

(подпись) (и.о.фамилия)

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_ 2022 г.

дата

Отчет

по дисциплине

ОСНОВЫ МОДЕЛИРОВАНИЯ

лабораторная работа №2

Правила Петрова для решения задачи упорядочивания nxm(реализация правил Петрова и метод перебора)

название работы

ЛР 09.03.04.23.002 О

обозначение документа

Студент группы гр. ПИ-02                                                     Чередов Р.А.

*(подпись*)

Преподаватель доцент, к.т.н.               А. В. Сорокин

должность, ученое звание и.о., фамилия

БАРНАУЛ 2022

# **Аннотация**

В данной лабораторной работе были рассмотрено использование правил Петрова для решения задачи упорядочения NxM. Так же построена программа для решения задачи упорядочения NxM методом перебора.

Оглавление

[**Аннотация** 2](#_Toc152004916)

[**Задание** 4](#_Toc152004917)

[**Решение** 5](#_Toc152004918)

# **Задание**

1. Используя теоретический материал из Темы 1 Теории к упражнениям разобраться с правилами Петорова для поиска субоптимального решения задачи упорядочения nxm, где n – число деталей(Партий деталей), m – число станков.
2. Ознакомиться с формулами построения параметров Pi1,Pi2 и подмножеств D1,D0,D2/
3. Получить последовательности запуска деталей в обработку с использованием 4-х правил Петрова.
4. Получить последовательность запуска деталей в обработку с использованием генератора случайных чисел.

# **Решение**

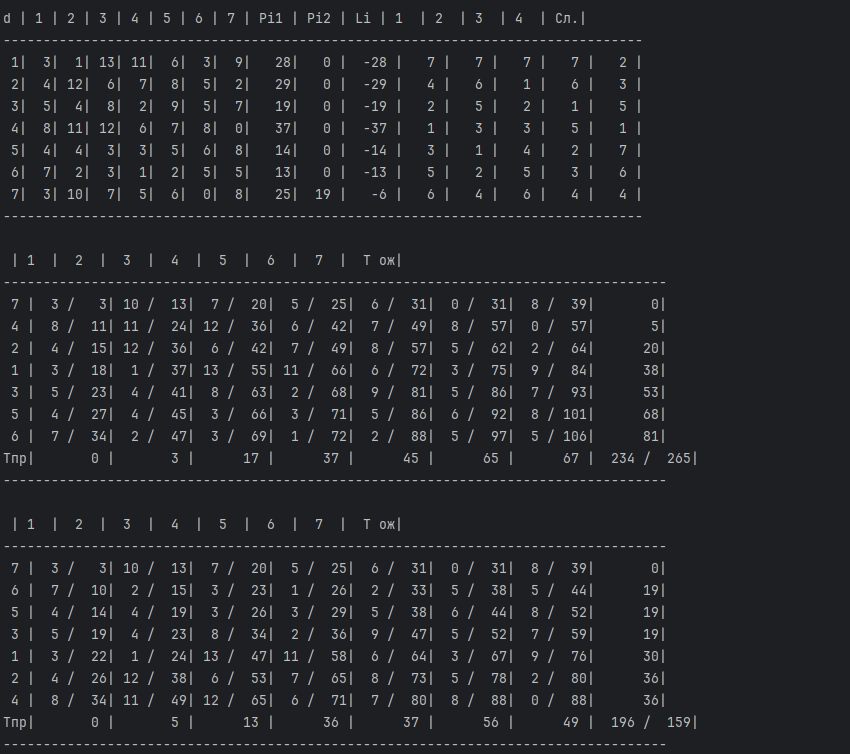
Для реализации правил Петрова, а так метода перебора была построена программа на языке Python.

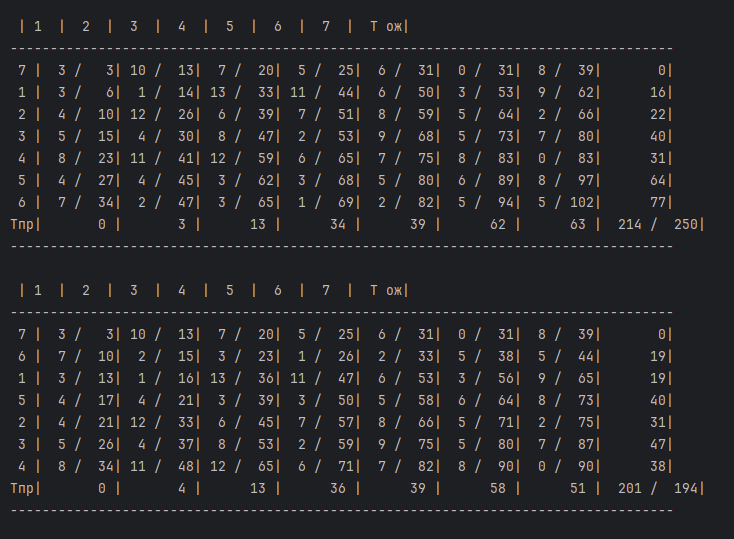
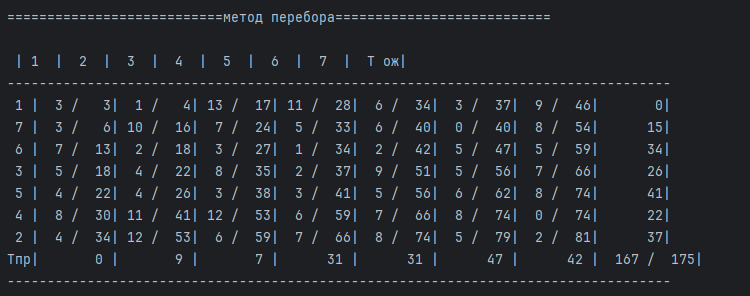
import numpy as np  
import math  
N = 7  
M = {  
 1:[3, 1, 13, 11, 6, 3, 9],  
 2:[4, 12, 6, 7, 8, 5, 2],  
 3:[5, 4, 8, 2, 9, 5, 7],  
 4:[8, 11, 12, 6, 7, 8, 0],  
 5:[4, 4, 3, 3, 5, 6, 8],  
 6:[7, 2, 3, 1, 2, 5, 5],  
 7:[3, 10, 7, 5, 6, 0, 8]  
}  
  
PI1 = [0] \* N  
PI2 = [0] \* N  
LI = [0] \* N  
  
# Массив последовательностей  
Posl = [  
 [0] \* N,  
 [0] \* N,  
 [0] \* N,  
 [0] \* N,  
 [0] \* N  
]  
  
# массив время окончания обработки i-ой детали на j-ом станке  
masT = [  
 [0] \* N,  
 [0] \* N,  
 [0] \* N,  
 [0] \* N,  
 [0] \* N,  
 [0] \* N,  
 [0] \* N  
]  
Tpr = [0] \* N # Время простоя станка  
Toj = [0] \* N # Время ожидания детали перед обработкой  
# Вычисление Pi1 и Pi2  
def calcPI():  
 if (N % 2 == 0):  
 ot = N/2  
 to = N/2  
 else:  
 ot = (N+1)/2 - 1  
 to = (N+1)/2  
  
 for i in range (0, N):  
 for j in range (0, round(to)):  
 PI1[i] += M[i+1][j]  
  
 for j in range (round(ot), N):  
 PI2[i] += M[i+1][j]  
# Вычисление лямбда  
def calcLI():  
 for i in range (0, N):  
 LI[i] = PI2[i] - PI1[i]  
# Правила Петрова  
def Rule1():  
 posled = []  
 D01 = []  
 D2 = []  
  
 for i in range (0, N):  
 if LI[i] >= 0:  
 D01.append([i, PI1[i], LI[i]])  
 else:  
 D2.append([i, PI2[i], LI[i]])  
 D01.sort(key=lambda x: (x[1], -x[2]))  
 D2.sort(key=lambda x: (x[1], -x[2]), reverse=True)  
  
 for i in D01:  
 posled.append(i[0] + 1)  
  
 for i in D2:  
 posled.append(i[0] + 1)  
  
 return posled  
def Rule2():  
 posled = []  
 copyLI = []  
  
 for i in range(0, N):  
 copyLI.append([i, LI[i], PI1[i]])  
 copyLI.sort(key=lambda x: (x[1], x[2]), reverse=True)  
 for i in copyLI:  
 posled.append(i[0] + 1)  
 return posled  
def Rule3():  
 posled = []  
 D0 = []  
 D1 = []  
 D2 = []  
 for i in range(0, N):  
 if LI[i] == 0:  
 D0.append([i, PI1[i]])  
 else:  
 if LI[i] > 0:  
 D1.append([i, PI1[i]])  
 else:  
 D2.append([i, PI2[i]])  
 D0.sort(key=lambda x: x[1])  
 D1.sort(key=lambda x: x[1])  
 D2.sort(key=lambda x: x[1], reverse=True)  
 for i in D1:  
 posled.append(i[0] + 1)  
  
 for i in D0:  
 posled.append(i[0] + 1)  
  
 for i in D2:  
 posled.append(i[0] + 1)  
  
 return posled  
def Rule4():  
 posled = []  
 pair = [0,0]  
 masPair = []  
 D0 = []  
 D1 = []  
 D2 = []  
 for i in range(0, N):  
 if LI[i] == 0:  
 D0.append([i, PI1[i], PI2[i]])  
 else:  
 if LI[i] > 0:  
 D1.append([i, PI1[i], PI2[i]])  
 else:  
 D2.append([i, PI1[i], PI2[i]])  
 while (len(D1) != 0):  
 if (len(D1) == 1):  
 break  
 pair[0] = max(D1, key=lambda x: x[2])  
 D1.remove(pair[0])  
 pair[1] = min(D1, key=lambda x: x[1])  
 D1.remove(pair[1])  
 masPair.append([pair[:], pair[0][2] - pair[1][1],  
 max(pair[0][2] - pair[0][1], pair[1][2] - pair[1][1]),  
 min(pair[0][2] - pair[0][1], pair[1][2] - pair[1][1])])  
  
 while (len(D0) != 0):  
 if (len(D1) == 1 and len(D0) == 1):  
 pair[0] = D1[0]  
 D1.remove(pair[0])  
 pair[1] = min(D0, key=lambda x: x[1])  
 D0.remove(pair[1])  
  
 masPair.append([pair[:], pair[0][2] - pair[1][1],  
 max(pair[0][2] - pair[0][1], pair[1][2] - pair[1][1]),  
 min(pair[0][2] - pair[0][1], pair[1][2] - pair[1][1])])  
 break  
 if (len(D0) == 1):  
 break  
  
 if (len(D1) == 1):  
 pair[0] = D1[0]  
 D1.remove(pair[0])  
 else:  
 pair[0] = max(D0, key=lambda x: x[2])  
 D0.remove(pair[0])  
  
 pair[1] = min(D0, key=lambda x: x[1])  
 D0.remove(pair[1])  
  
 masPair.append([pair[:], pair[0][2] - pair[1][1],  
 max(pair[0][2] - pair[0][1], pair[1][2] - pair[1][1]),  
 min(pair[0][2] - pair[0][1], pair[1][2] - pair[1][1])])  
 while (len(D2) != 0):  
 if ((len(D1) == 1 or len(D0) == 1) and len(D2) == 1):  
 if (len(D1) == 1):  
 pair[0] = D1[0]  
 D1.remove(pair[0])  
 else:  
 pair[0] = D0[0]  
 D0.remove(pair[0])  
 pair[1] = min(D2, key=lambda x: x[1])  
 D2.remove(pair[1])  
 masPair.append([pair[:], pair[0][2] - pair[1][1],  
 max(pair[0][2] - pair[0][1], pair[1][2] - pair[1][1]),  
 min(pair[0][2] - pair[0][1], pair[1][2] - pair[1][1])])  
 break  
 if (len(D2) == 1):  
 break  
 if (len(D1) == 1 or len(D0) == 1):  
 if (len(D1) == 1):  
 pair[0] = D1[0]  
 D1.remove(pair[0])  
 else:  
 pair[0] = D0[0]  
 D0.remove(pair[0])  
 else:  
 pair[0] = max(D2, key=lambda x: x[2])  
 D2.remove(pair[0])  
 pair[1] = min(D2, key=lambda x: x[1])  
 D2.remove(pair[1])  
  
 masPair.append([pair[:], pair[0][2] - pair[1][1],  
 max(pair[0][2] - pair[0][1], pair[1][2] - pair[1][1]),  
 min(pair[0][2] - pair[0][1], pair[1][2] - pair[1][1])])  
  
 if (N % 2 != 0):  
 if (len(D0) != 0):  
 dx = D0[0]  
 else:  
 if (len(D1) != 0):  
 dx = D1[0]  
 else:  
 dx = D2[0]  
 ldx = LI[dx[0]]  
 pos = -1  
 for i in range(0, len(masPair) - 1):  
 if (masPair[i][2] > ldx and masPair[i + 1][3] < ldx):  
 pos = i+1  
 break  
 for i in range(0, len(masPair)):  
 if i == pos:  
 posled.append(dx[0] + 1)  
 for j in masPair[i][0]:  
 posled.append(j[0] + 1)  
  
 if pos == -1:  
 posled.append(dx[0] + 1)  
 return posled  
# Вычисление массива времен  
def calcT(posled):  
 T = [  
 [0] \* N,  
 [0] \* N,  
 [0] \* N,  
 [0] \* N,  
 [0] \* N,  
 [0] \* N,  
 [0] \* N  
 ]  
 T[0][0] = M[posled[0]][0]  
 for i in range(1, N):  
 T[0][i] = T[0][i-1] + M[posled[0]][i]  
 T[i][0] = T[i-1][0] + M[posled[i]][0]  
 for i in range(1, N):  
 for j in range(1, N):  
 T[i][j] = max(T[i - 1][j], T[i][j - 1]) + M[posled[i]][j]  
 return T  
# Вычисление времен простоя станка  
def calcTpr(T):  
 for i in range(0, N):  
 sumSt = 0  
 for j in range(0,N):  
 sumSt += M[j+1][i]  
 Tpr[i] = T[N-1][i] - sumSt  
# Вычисление времен ожидания детали  
def calcToj(T, posled):  
 for i in range(0, N):  
 Toj[i] = T[i][N-1] - sum(M[posled[i]])  
# Печать исходной таблицы  
def printTable():  
 print ("\nd |", end=' ')  
 for i in range (1, N+1):  
 print(i, '|', end=' ')  
 print("Pi1 | Pi2 | Li |", end=' ')  
 for i in range(1, 5):  
 print(i, ' |', end=' ')  
 print("Сл.|")  
 print("--------------------------------------------------------------------------------")  
  
 for i in range(0, N):  
 print('%2d|' % (i + 1), end='')  
 for j in range(0, N):  
 print('%3d|' % M[i + 1][j], end='')  
 print('%5d| %3d | %4d |' % (PI1[i], PI2[i], LI[i]), end='')  
 for j in range(0, 4):  
 print('%4d |' % Posl[j][i], end='')  
 print('%4d |' % Posl[4][i])  
 print("--------------------------------------------------------------------------------")  
# Печать таблиц по последовательности  
def printTable2(posled, T):  
 print ("\n |", end=' ')  
 for i in range (1, N+1):  
 print(i, ' | ', end=' ')  
 print ( "T ож|")  
 print("-----------------------------------------------------------------------------------")  
  
 for i in range(0, 7):  
 print('%2d |' % posled[i], end='')  
 for j in range(0, N):  
 print('%3d / %3d|' % (M[posled[i]][j], T[i][j]), end='')  
 print('%8d|' % Toj[i])  
 print("Tпр|", end=' ')  
 for i in range(1, N + 1):  
 print('%7d | ' % Tpr[i - 1], end='')  
 print('%4d / %4d|' % (sum(Tpr), sum(Toj)))  
 print("-----------------------------------------------------------------------------------")  
#Реализация перебора  
#Смена мест в массиве  
def swap (MAS, i, j):  
 t = MAS[i]  
 MAS[i] = MAS[j]  
 MAS[j] = t  
 return MAS  
#Следующая перестановка в лексиграфическом порядке  
def NextSet (a, n):  
 j = n - 2  
 while (j != -1 and a[j] >= a[j + 1]):  
 j -= 1  
 if (j == -1):  
 return 0  
 k = n - 1  
 while (a[j] >= a[k]):  
 k -= 1  
 a = swap(a, j, k)  
 l = j + 1  
 r = n - 1  
#Сортируем оставшуюся часть последовательности  
 while (l < r):  
 a = swap(a, l, r)  
 l += 1  
 r -= 1  
 return 1  
def allPosled():  
#Массив со всеми перестановки в лексиграфическом порядке  
 permutations = [0] \* math.factorial(N)  
 Nt = []  
 k = 0  
 for i in range(0, N):  
 Nt.append(i+1)  
 permutations[k] = Nt.copy()  
#Собираем массив перестановок  
 while (NextSet(Nt, N)):  
 k += 1  
 permutations[k] = Nt.copy()  
 return permutations  
permutations = allPosled()  
minT = 9999999  
minToj = 9999999  
for perm in permutations:  
 T = calcT(perm)  
 calcToj(T, perm)  
 if T[N - 1][N - 1] < minT:  
 minT = T[N - 1][N - 1]  
 minPerm = perm  
 minToj = sum(Toj)  
 if T[N - 1][N - 1] == minT:  
 if sum(Toj) < minToj:  
 minT = T[N - 1][N - 1]  
 minPerm = perm  
 minToj = sum(Toj)  
#Все вызовы  
calcPI()  
calcLI()  
Posl[0] = Rule1()  
Posl[1] = Rule2()  
Posl[2] = Rule3()  
Posl[3] = Rule4()  
Posl[4] = [2,3,5,1,7,6,4]  
  
printTable()  
  
for i in range(0,5):  
 T = calcT(Posl[i])  
 calcTpr(T)  
 calcToj(T, Posl[i])  
 printTable2(Posl[i], T)  
  
print('===========================метод перебора===========================')  
  
T = calcT(minPerm)  
calcTpr(T)  
calcToj(T, minPerm)  
printTable2(minPerm, T)

# **Вариант**



# **Вывод программы**



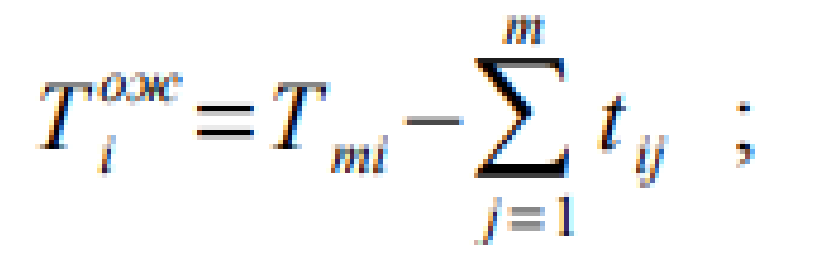
 

# **Заключение**

В данной лабораторной работе были рассмотрено использование правил Петрова для решения задачи упорядочения NxM. Так же построенна программа для решения задачи упорядочения NxM метода перебора. В процессе выполнения работы возникли сложности с пониманием и реализацией 4-го правила Петрова, но по ходу действий со всем разобрался, правило реализовал.

Лучший вариант – 4 правило.

# **Вопросы при защите**

1. Как рассчитывается время ожидания(пролеживания) i-й детали перед обработки. Ответ по формуле: 
2. Как влияет время ожидания(пролеживания) в экономическом плане.Ответ: Чем меньше время пролеживания- тем меньше экономических затрат. Для детали нужно складское помещение, за это платится аренда.