МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

ТЕМА: ОЧЕРЕДИ С ПРИОРИТЕТОМ. ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА

Студент гр. 0304 _____ Докучаев Р.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2021

Цель работы.

Изучить очереди с приоритетом и параллельную обработку.

Задание.

На вход программе подается число процессоров n и последовательность чисел $t_0, \ldots, t_{m-1}$, где t_i— время, необходимое на обработку i-й задачи. Требуется для каждой задачи определить, какой процессор и в какое время начнёт её обрабатывать, предполагая, что каждая задача поступает на обработку первому освободившемуся процессору. Примечание: в работе запрещено использовать библиотечные реализации алгоритмов и структур.

Формат входа

Первая строка входа содержит числа n и m. Вторая содержит числа $t_0, \ldots, t_{m-1}$, где t_i — время, необходимое на обработку i-й задачи. Считаем, что и процессоры, и задачи нумеруются с нуля.

Формат выхода

Выход должен содержать ровно m строк: i-я (считая с нуля) строка должна содержать номер процессора, который получит i-ю задачу на обработку, и время, когда это произойдёт.

Основные теоретические положения.

Были использованы стандартные возможности языка Python: функции, классы, циклы. Также была использована утилита pytest для тестирования работы программы. Кроме того, была реализована очередь с приоритетом (тип очереди, при котором каждый элемент имеет свой приоритет и в соответствии с этим приоритетом элемент обслуживается), а также параллельная обработка (метод работы, при котором несколько задач выполняются одновременно, а не по очереди).

Выполнение работы.

1. Для обработки входных данных и работой с ними был создан класс Min_Heap, который содержит в себе следующие методы:
    - метод __init__, который осуществляет запись данных n (количество процессоров), m (количество задач) и time (время для выполнения каждой задачи; также были созданы поля answer – пустая строка, в которую будет записан ответ, и proc – список процессоров, где хранятся номер процессора и время его работы
    - методы get_left, get_right и get_parent, которые созданы для работы с мин-кучей; каждый из этих методов соответственно возвращает номер левого потомка элемента с индексом index, номер правого потомка элемента с индексом index и номером родителя элемента с индексом index
    - метод swap, которые меняет местами элементы с заданными в параметрах метода индексами
    - метод down, который опускает элемент под индексом index вниз в соотвествии с его приоритетом (временем, затраченным процессором на выполнение задачи; если время выполнения одинаковое, то приоритет выше у того элемента, индекс которого меньше)
    - метод proc_time, в котором в цикле перебираются все задачи и при помощи метода down происходит обработка результата; метод возвращает результат через поле answer
2. В основном модуле программы будет произведено считывание параметров с консоли, будет создан экземпляр класса, будет вызван метод proc_time и будет происходить вывод результат на экран пользователя.
3. Был создан тестовый модель test.py, который на примерах проверяет корректность выполнения сортировки.

Исходный код программы представлен в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	2 5 1 2 3 4 5	0 0 1 0 0 1 1 2 0 4	OK
2.	4 20 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	0 0 1 0 2 0 3 0 0 1 1 1 2 1 3 1 0 2 1 2 2 2 3 2 0 3 1 3 2 3 3 3 0 4	OK

		1 4 2 4 3 4	

Выводы.

Были изучены очереди с приоритетом и параллельная обработка. Была реализована программа, выполняющая при помощи очереди с приоритетом указанную задачу.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Min_Heap:
    def __init__(self, n, m, *time):
        self.answer = ""
        self.n = n
        self.proc = [[i, 0] for i in range(n)]
        self.m = m
        self.time = [j for j in range(m)]
        for k in range(m):
            self.time[k] = time[k]

    def get_left(self, index):
        if (2*index+1) < self.n:
            return 2*index+1
        else: return -1

    def get_right(self, index):
        if (2*index+2) < self.n:
            return 2*index+2
        else: return -1

    def get_parent(self, index):
        if (index-1)//2 < 0:
            return 0
        return (index-1)//2

    def swap(self, i, j):
        temp = self.proc[i]
        self.proc[i] = self.proc[j]
        self.proc[j] = temp

    def down(self, index):
        max = index

        if self.get_left(index) > 0:
            if self.proc[self.get_left(index)][1] <
               self.proc[max][1]:
                max = self.get_left(index)
            if self.proc[self.get_left(index)][1] ==
               self.proc[max][1]:
                if self.proc[self.get_left(index)][0] <
                   self.proc[max][0]:
                    max = self.get_left(index)

        if self.get_right(index) > 0:
            if self.proc[self.get_right(index)][1] <
               self.proc[max][1]:
                max = self.get_right(index)
            if self.proc[self.get_right(index)][1] ==
               self.proc[max][1]:
                if self.proc[self.get_right(index)][0] <
                   self.proc[max][0]:
                    max = self.get_right(index)
```

```
            if index != max:
                self.swap(index, max)
                self.down(max)

        def proc_time(self):
            for i in range(self.m):
                self.answer += str(self.proc[0][0]) + " "
                             + str(self.proc[0][1]) + "\n"
                self.proc[0][1] += self.time[i]
                self.down(0)
            return self.answer

    if __name__ == "__main__":
        n, m = map(int, input().split())
        time = list(map(int, input().split()))
        heap = Min_Heap(n, m, *time)
        result = heap.proc_time()
        print(result, end="")
```