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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: «Очереди с приоритетом. Параллельная обработка».

Студент гр. 1304 _____ Байков Е.С.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2022

Цель работы

Изучить очереди с приоритеты и с их помощи реализовать параллельную обработку.

Задание

На вход программе подается число процессоров n и последовательность чисел t_0 , . . . , 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 #1: в работе необходимо использовать очередь с приоритетом (т.е. min или max-кучу)

Примечание #2: в работе запрещено использовать библиотечные реализации алгоритмов и структур.

Формат входа

Первая строка входа содержит числа n и m. Вторая содержит числа t_0 , . . . , 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.

Выполнение работы

Был создан класс *MinHeap*, с помощью которого была реализована мин-куча. В полях класса хранится список списков, каждый элемент которого выглядит как *[t, n]* – где *t* – время освобождения процессора, после выполнения очередной задачи, а *n* – номер процессора. Объект класса

создается при передаче в него количества процессоров. Внутри класса также реализованы такие функции как:

- *parent(self, index)* – возвращает индекс родителя, в зависимости от переданного индекса ребенка
- *left_child(self, index)* – возвращает индекс левого ребенка в зависимости от индекса родителя
- *right_child(self, index)* – возвращает индекс правого ребенка в зависимости от индекса родителя.
- *sift_down(self, index)* – просеивает элементы вниз. С помощью операции сравнения выбирается первый свободный элемент (тот у которого меньше время), если таких элементов несколько то выбирается тот, у которого меньше номер процессора.

Также создано две функции *process_1* и *process_2*. Первая функция получает на вход две строки, из первой в программу передается *n* и *m*, во второй время выполнения процессов. И выполняется создание кучи и с помощью цикла каждый процессор получает время выполнения последнего процесса, а также выводится информация, какой процесс в какое время начал обрабатывать следующий процесс. В конце цикла происходит просеивание верхнего элемента вниз.

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	
2.	4 10 3 0 9 2 8 1 9 8 8 4	0 0 1 0 1 0 2 0 3 0 2 2 0 3 2 3 3 8 1 9	
3.	2 15 0 0 1 0 0 0 2 1 2 3 0 0 0 2 1	0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 0 2 1 2 0 4 0 4 0 4 0 4 1 5	

Выводы

В ходе выполнения лабораторной работы была реализована программа, которая с помощью очереди с приоритетом, созданной на основе двоичной мин-кучи, выполняет параллельную обработку процессов.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class MinHeap:
    def __init__(self, size) -> None:
        self.size = size
        self.view = [[0, i] for i in range(size)]

    def parent(self, index):
        return (index - 1) // 2
    def left_child(self, index):
        return 2 * index + 1
    def right_child(self, index):
        return 2 * index + 2

    def sift_down(self, index):
        left = self.left_child(index)
        right = self.right_child(index)
        if left >= self.size and right >= self.size:
            return
        ind = index
        if left < self.size and self.view[left] < self.view[ind]:
            ind = left
        if right < self.size and self.view[right] < self.view[ind]:
            ind = right

        if ind != index:
            self.view[ind],   self.view[index]   =   self.view[index],
self.view[ind]
            self.sift_down(ind)

def process_1():
    n, m = map(int, input().split())
    arr = list(map(int, input().split()))[:m]
    min_heap = MinHeap(n)
    for elem in arr:
        print(min_heap.view[0][1], min_heap.view[0][0])
        min_heap.view[0][0] += elem
        min_heap.sift_down(0)


def process_2(num_1 : int, num_2 : int, string : str):
    n, m = num_1, num_2
    arr = list(map(int, string.split()))[:m]
    answer = ''
    min_heap = MinHeap(n)

    for elem in arr:
        answer += f'{min_heap.view[0][1]} {min_heap.view[0][0]}\n'
        min_heap.view[0][0] += elem
        min_heap.sift_down(0)

    return answer
```

Название файла: tests.py

```
from main import process_2

def test():
    answer = process_2(num_1=2, num_2=5, string='1 2 3 4 5')
    assert answer == '0 0\n1 0\n0 1\n1 2\n0 4\n'
    answer = process_2(4, 10, '3 0 9 2 8 1 9 8 8 4')
    assert answer == '0 0\n1 0\n1 0\n2 0\n3 0\n2 2\n0 3\n2 3\n3 8\n1
9\n'
    answer = process_2(2, 15, '0 0 1 0 0 0 2 1 2 3 0 0 0 2 1')
    assert answer == '0 0\n0 0\n0 0\n1 0\n1 0\n1 0\n1 0\n0 1\n0 2\n1
2\n0 4\n0 4\n0 4\n0 4\n1 5\n'
```