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. 1304 _____ Поршнев Р.А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2022

Цель работы.

Изучить очередь с приоритетом через реализацию min-кучи.

Задание.

На вход программе подается число процессоров n и последовательность чисел t0 , . . . , tm−1 , где ti — 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 t0 , . . . , tm−1 , где ti — 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.

Ограничения

$1 \le n \le 10^5$; $1 \le m \le 10^5$; $0 \le t_i \le 10^9$.

Выполнение работы.

В main происходит считывание число процессоров n и количество задач m. Затем происходит считывание время выполнения каждой задачи. Следующим шагом каждому процессору присваивается время получения задачи равное 0. Данные, указанные выше, хранятся в списке списков data.

```
data = [[0, i] for i in range(n)]
```

Затем создаётся экземпляр класса MinHeap, в конструктор которого передаётся информация о каждом процессоре data. Далее вызывается метод

solve у экземпляра heap, в который передаётся список задач. Следующим шагом в переменную ans возвращается список, в котором хранится информация о каждом процессоре, а именно его номер и время получения задачи. Далее циклом выводится ответ на исходную задачу.

Реализован класс MinHeap, который имеет следующие методы:

1) get_left_child(index) – возвращает индекс левого ребёнка для index;
2) get_right_child(index) – возвращает индекс правого ребёнка для index;
3) sift_down(self, index) – просеивание вниз начиная с элемента data[index], которое нужно для сохранения свойства min-кучи;
4) solve(self, tasks) – решает задачи параллельной обработки для списка tasks. Для каждой задачи из списка tasks выполняются следующие действия:
    1) Фиксируется время и номер процессора, который лежит в корне min – кучи, и записывается в список ответов ans;
    2) К корневому элементу прибавляется время получения текущей задачи и происходит просеивание вниз для сохранения свойств min-кучи.

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	Ответ правильный, тест из примера
2.	5 8 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	Ответ правильный, случай при времени выполнении каждой задачи равному 0

		0 0 0 0 0 0 0 0	
3.	2 2 100 1	0 0 1 0	Ответ правильный, случай наличия у корня только левого ребёнка
4.	1 1 10	0 0	Ответ правильный, случай наличия у дерева только корня
5.	2 5 2 2 2 2 2	0 0 1 0 0 2 1 2 0 4	Ответ правильный, проверяется случай при одинаковом времени выполнения каждой задачи
6.	5 10 4 9 4 4 8 5 7 3 3 6	0 0 1 0 2 0 3 0 4 0 0 4 2 4 3 4 3 7 4 8	Ответ правильный, проверяется случай при большом количестве процессоров и задач

Вывод.

В ходе выполнения лабораторной работы была изучена min-куча для реализации очереди с приоритетом.

Разработана программа, выполняющая параллельную обработку процессов с помощью очереди с приоритетом, которая реализована на основе min-кучи.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class MinHeap:
    def __init__(self, data):
        self.ans = []
        self.data = data
        self.size = len(data)

    @staticmethod
    def get_left_child(index):
        return 2 * index + 1

    @staticmethod
    def get_right_child(index):
        return 2 * index + 2

    def sift_down(self, index):
        min_index = index
        left_child_index = self.get_left_child(index)
        if left_child_index < self.size and
self.data[left_child_index] < self.data[min_index]:
            min_index = left_child_index
        right_child_index = self.get_right_child(index)
        if right_child_index < self.size and
self.data[right_child_index] < self.data[min_index]:
            min_index = right_child_index
        if index != min_index:
            self.data[min_index], self.data[index] =
self.data[index], self.data[min_index]
            self.sift_down(min_index)

    def solve(self, tasks):
        for task in tasks:
            proc_info = []
            time = self.data[0][0]
            processor = self.data[0][1]
            proc_info.append(processor)
            proc_info.append(time)
            self.ans.append(proc_info)
            self.data[0][0] += task
            self.sift_down(0)

    def get_ans(self):
        return self.ans


if __name__ == "__main__":
    n, m = map(int, input().split())
    tasks = list(map(int, input().split()))
    data = [[0, i] for i in range(n)]
    heap = MinHeap(data)
    heap.solve(tasks)
    ans = heap.get_ans()
    for i in range(len(ans)):
```

```
        print(ans[i][0], ans[i][1])
```

Название файла: tests.py

```
from main import MinHeap


def test_1():
    n = 2
    tasks = [100, 1]
    data = [[0, i] for i in range(n)]
    heap = MinHeap(data)
    heap.solve(tasks)
    assert heap.get_ans() == [[0, 0], [1, 0]]


def test_2():
    n = 5
    tasks = [0, 0, 0, 0, 0, 0, 0, 0]
    data = [[0, i] for i in range(n)]
    heap = MinHeap(data)
    heap.solve(tasks)
    assert heap.get_ans() == [[0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0]]


def test_3():
    n = 1
    tasks = [10]
    data = [[0, i] for i in range(n)]
    heap = MinHeap(data)
    heap.solve(tasks)
    assert heap.get_ans() == [[0, 0]]


def test_4():
    n = 2
    tasks = [2, 2, 2, 2, 2]
    data = [[0, i] for i in range(n)]
    heap = MinHeap(data)
    heap.solve(tasks)
    assert heap.get_ans() == [[0, 0], [1, 0], [0, 2], [1, 2], [0,
4]]


def test_5():
    n = 5
    tasks = [4, 9, 4, 4, 8, 5, 7, 3, 3, 6]
    data = [[0, i] for i in range(n)]
    heap = MinHeap(data)
    heap.solve(tasks)
assert heap.get_ans() == [[0, 0], [1, 0], [2, 0], [3, 0], [4, 0],
[0, 4], [2, 4], [3, 4], [3, 7], [4, 8]]
```