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 _____ Мажуга Д. Р.

Преподаватель _____ Берленко Т. А.

Санкт-Петербург
2021

Цель работы.

Изучить очереди с приоритетом и параллельную обработку.

Задание.

Параллельная обработка. Python

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: 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.

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

Основные теоретические положения.

Очередь с приоритетом — это тип очереди, в котором у каждого элемента имеется приоритет, в соответсвии с которым он обслуживается.
Параллельная обработка — метод работы, при котором несколько задач обрабатываются одновременно.

Выполнение работы.

Был создан класс *MinHeap* в конструкторе которого, мы передаём количество процессов, количество заданий, которые нам следует выполнить, и время которое необходимо на обработку *i*-той задачи

Методы которые были выполнены:

1. Метод *sift_down(index)* — это метод который выполняет просеевание вниз в соответсвии с его приоритетом. На вход мы передаём индекс элемента

который в последствии будет просеян, в качестве приоритета выступает время затраченное процессором на обработку всех задач, в случае когда у нескольких процессов совпадает время, высший приоритет будет иметь тот процессор у которого меньше индекс.

2. Метод *procces_time()* - метод который возвращает строку в которой содержиться номер процессора,который получит i-ю задачу на обработку, и время, когда это произойдет. В данном методе мы в цикле обходим все задачи и на каждой итерации добавляем в заранее созданную строку result, номер процессора и время, когда процессор получил задачу. Затем ко времени у соответсвующего процессора добавляется время, требуемое для выполнение новой задачи, после чего выполняем просеивание внизу с помощью метода sift_down().

3. В связи с тем что мин-кучу удобно хранить в виде массива у которого *i*-тый элемент, это элемент в корне, а потомками, т.е. левыми и правыми детьми являются *2i* — индекс левого ребёнка и *2i + 1* — индекс правого ребёнка соотвественно. Были реализованны методы *parent(index)*, *left_child(index)* и *right_child(index)* для работы с мин-кучей.

    1. Метод *parent(index)* - возвращает индекс родителя предварительно проверяя не является ли полученный нами индекс родителя меньше нуля, если нет, то возвращает индекс родителя.

    2. Метод *left_child(index)* — возвращает индекс левого ребёнка и для того чтобы не выйти за пределы массива предварительно проверяет не является ли полученный нами индекс больше количества процессоров.

    3. Метод *right_child(index)* - возвращает индекс правого ребёнка и для того чтобы не выйти за пределы массива предварительно проверяет не является ли полученный нами индекс больше количества процессоров.

Тестирование.

Были написанны тесты, необходимые для проверки корректности результатов программы.Результаты тестирования представлены в табл. 1.

№	Входные данные	Выходные данные	Комментарий
1	2 5 1 2 3 4 5	0 0 1 0 0 1 1 2 0 4	OK
2	1 5 1 2 3 4 5	0 0 0 1 0 3 0 6 0 10	OK
3	12 5 1 2 3 4 5	0 0 1 0 2 0 3 0 4 0	OK
4	10 10 1 2 3 4 5 6 7 8 9 10	0 0 1 0 2 0 3 0 4 0 5 0 6 0 7 0 8 0 9 0	OK

Таблица 1 — Результаты тестирования

Вывод.

Была разработана программа, принимающая на вход количество процессоров, количество задач и время, требуемое для выполнени каждой, которая выводит номер процессора и время начала обработки задачи под номером, соответсвующим номеру строки вывода

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
from MinHeap import MinHeap

if __name__ == '__main__':
    num_process, num_task =
map(int, input().split())
    times = list(map(int,
input().split()))
    heap =
MinHeap(num_process,
num_task, *times)
    result = heap.process_time()
    print(result)
```

Название файла: MinHeap.py

```
class MinHeap:
    def __init__(self,
num_process, num_task,
*times):
        self.number_process =
num_process
        self.task = num_task
        self.times = [i for i in
range(num_task)]

        self.processors = [[i, 0] for
i in range(num_process)]
        for i in range(num_task):
            self.times[i] = times[i]

    def sift_down(self, index):
        max_index = index
        left_child =
self.left_child(index)
        if left_child > 0:
```

```
            if
self.processors[left_child][1] <
self.processors[max_index][1]:
                max_index =
left_child
            if
self.processors[left_child][1] ==
self.processors[max_index][1]:
                if
self.processors[left_child][0] <
self.processors[max_index][0]:
                    max_index =
left_child

        right_child =
self.right_child(index)
        if right_child > 0:
            if
self.processors[right_child][1] <
self.processors[max_index][1]:
                max_index =
right_child
            if
self.processors[right_child][1]
== self.processors[max_index]
[1]:
                if
self.processors[right_child][0] <
self.processors[max_index][0]:
                    max_index =
right_child

        if index != max_index:
            self.processors[index],
self.processors[max_index] =
self.processors[max_index],
self.processors[index]

self.sift_down(max_index)
```

```
    def process_time(self):
        answer = ''
        for i in range(self.task):
            answer +=
str(self.processors[0][0]) + ' ' +
str(self.processors[0][1]) + '\n'
            self.processors[0][1] +=
self.times[i]
            self.sift_down(0)
        return answer

    @staticmethod
    def parent(index):
        if (index - 1)//2 < 0:
            return 0
        return (index - 1) // 2

    def left_child(self, index):
        if (2 * index + 1) <
self.number_process:
            return 2 * index + 1
        return -1

    def right_child(self, index):
        if (2 * index + 2) <
self.number_process:
            return 2 * index + 2
        return — 1
```

Название файла: tests.py

```
from MinHeap import MinHeap



class TestCase:
    @staticmethod
    def frst_test():
        num_process = 2
        num_task = 5
        time_process = [1, 2, 3, 4,
5]
```

```
        guess_value = '0 0\n1 0\n0
1\n1 2\n0 4\n'

        answer =
MinHeap(num_process,
num_task,
*time_process).process_time()
        assert answer ==
guess_value
        print('test first: OK')

    @staticmethod
    def scnd_test():
        num_process = 1
        num_task = 5
        time_process = [1, 2, 3, 4,
5]
        guess_value = '0 0\n0 1\n0
3\n0 6\n0 10\n'

        answer =
MinHeap(num_process,
num_task,
*time_process).process_time()
        assert answer ==
guess_value
        print('test second: OK')

    @staticmethod
    def thrd_test():
        num_process = 12
        num_task = 5
        time_process = [1, 2, 3, 4,
5]
        guess_value = '0 0\n1 0\n2
0\n3 0\n4 0\n'

        answer =
MinHeap(num_process,
num_task,
```

```
*time_process).process_time()
        assert answer ==
guess_value
        print('test third: OK')

    @staticmethod
    def four_test():
        num_process = 10
        num_task = 10
        time_process = [1, 2, 3, 4,
5, 6, 7, 8, 9, 10]
        guess_value = '0 0\n1 0\n2
0\n3 0\n4 0\n5 0\n6 0\n7 0\n8
0\n9 0\n'

        answer =
MinHeap(num_process,
num_task,
*time_process).process_time()
        assert answer ==
guess_value
        print('test four: OK')


if __name__ == '__main__':
    case = TestCase
    case.frst_test()
    case.scnd_test()
    case.thrd_test()
    case.four_test()
```