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. 1381 _____ Возмитель В. Е

Преподаватель _____ Шевская Н. В.

Санкт-Петербург

2022

Цель работы.

Изучить структуру очереди с приоритетом и параллельную обработку.

Задание.

Параллельная обработка:

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.

Выполнение работы.

Был создан класс Heap, который реализует мин кучу, в которой хранится список процессоров и их значений. Создаются статические методы, для получения индексов родительской вершины и левой и правой дочерней вершины. Создается метод для вставки новой вершины, при этом увеличивается количество вершин и вызывается функция sift_up для упорядочивания родительских вершин. Создается метод extract_max, для получения задачи с наименьшим приоритетом, при этом вызывается функция sift_down для упорядочивания потомков, сравнивая их индексы.

Также была создана функция main, в которой происходит считывание количества процессоров, задач и время выполнения каждой задачи. Создаются экземпляр класса Heap, в конструктор класса передаем список из процессоров и их значений. Далее к минимальному по значению процессору прибавляем текущее время и вызываем метод sift_down, параллельно записывая нужную нам информацию в список res, и так до конца цикла.

Тестирование.

Для проверки работы программы был разработан код тестовой программы.

Всего 4 теста:

- *test_1*. Данный тест был взят с условия лабораторной с сайта. Вход: 2 процессора, 5 задач с временем: 1, 2, 3, 4, 5.

- *test_2*. Тест для проверки работы программы без ввода данных времени задач.

- test_3. Тест для проверки особого случая. Вход: 5 процессоров, 5 задач с временем: 1, 1, 1, 1, 1.

- test_4. Тест для проверки особого случая. Вход: 1 процессор, 4 задачи с временем: 1, 1, 1, 1.

Код файла с тестами находится в приложении А.

Выводы.

При выполнении лабораторной работы была изучена параллельная обработка и написана программа с использованием очереди с приоритетом.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: algos3.py

```
#python
class Heap:
    def __init__(self, heap):
        self.MAX_SIZE = 20
        self.heap = heap
        self.size = len(heap)
        self.res = []

    @staticmethod
    def get_parent(index):
        return (index - 1) // 2

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

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

    def insert(self, element):
        if self.size == self.MAX_SIZE:
            return -1
        self.heap[self.size] = element
        self.sift_up(self.size)
        self.size += 1

    def extract(self):
        max_element = self.heap[0]
                      self.heap[0],  self.heap[self.size  -  1]  =
self.heap[self.size - 1], None
        self.size -= 1
        self.sift_down(0)
        return max_element

    def sift_up(self, index):
        parent = self.get_parent(index)
        while index > 0 and self.heap[index] < self.heap[parent]:
            self.heap[parent], self.heap[index] = self.heap[index],
self.heap[parent]
            index = parent
            parent = self.get_parent(index)

    def sift_down(self, index):
        left = self.get_left_child(index)
        right = self.get_right_child(index)
        if left >= self.size and right >= self.size:
            return
```

```
            if right >= self.size:
                max_index = left if self.heap[left] < self.heap[index]
else index

            else:
                max_index = left if self.heap[left] < self.heap[right]
else right
                    max_index = max_index if self.heap[max_index] <
self.heap[index] else index

            if max_index != index:
                        self.heap[max_index], self.heap[index] =
self.heap[index], self.heap[max_index]
                self.sift_down(max_index)

        def __str__(self):
            return str(self.heap)


    def main(n, m, arr):
        n = n
        m = m
        times = arr
        procs = [[0, i] for i in range(n)]
        heap = Heap(procs)

        for time in times:
            cur_time = heap.heap[0][0]

            heap.res.append([heap.heap[0][1], cur_time])
            heap.heap[0][0] += time

            heap.sift_down(0)

        return heap.res


    n, m = map(int, input().split())
    arr = list(map(int, input().split()))
    res = main(n, m, arr)
    for i in range(len(res)):
        print(res[i][0], res[i][1])
```

Название файла: test3.py

```
    import unittest
    from algos3 import *


    class TestMethods(unittest.TestCase):
```

```
        def test_1(self):
            self.assertEqual(main(2, 5, [1, 2, 3, 4, 5]), [[0, 0], [1,
0], [0, 1], [1, 2], [0, 4]])

        def test_2(self):
            self.assertEqual(main(2, 0, []), [])

        def test_3(self):
            self.assertEqual(main(5, 5, [1, 1, 1, 1, 1]), [[0, 0], [1,
0], [2, 0], [3, 0], [4, 0]])

        def test_4(self):
            self.assertEqual(main(1, 4, [1, 1, 1, 1]), [[0, 0], [0, 1],
[0, 2], [0, 3]])


    if __name__ == "__main__":
        unittest.main()
```