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 _____ Клепнёв Д.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2021

Цель работы.

Изучить такую структуру данных, как очередь с приоритетом. А также рассмотреть ее реализацию на примере двоичной мин-кучи/макс-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.

Выполнение работы.

Был реализован класс *Heap*, который представляет двоичную мин-кучу. Содержит метод *performing*(), который решает задачу. После пользовательского ввода создается список, в котором каждый элемент соответственно список с двумя значениями, первое – номер процессора, второе – время, за которое

процессор перейдет к очередной задаче. Созданный список передается в конструктор экземпляра класса *Heap*. Метод *sift_down()* выполняет просеивание вниз для соблюдения условия мин-кучи. Как происходит просеивание: ищет ‘ребенка’ с наименьшим временем процессора у вершины ‘родителя’, если время обоих ‘детей’ равны, берет процессор с наименьшим номером и сравнивает его со временем процессора вершины ‘родителя’. Метод *performing*() прибавляет к текущему времени процессора время на обработку поступившей задачи, а затем вызывает *sift_down(0)*, будет получаться, что в корне находится процессор с наименьшим временем, то есть тот процессор, который освободился от задачи быстрее всех.

Тестирование.

№ теста	Входные данные	Выходные данные
1	2 1 1	0 0
2	1 4 1 2 3 4	0 0 0 1 0 3 0 6
3	2 0	-
4	3 7 1 2 3 4 5 1 2	0 0 1 0 2 0 0 1 1 2 2 3 2 4
5	2 5	0 0

	2 1 1 3 1	1 0 1 1 0 2 1 2

Выводы.

Изучил способы реализации очереди с приоритетом через реализацию двоичной мин-кучи.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
CLASS HEAP:
    DEF __INIT__(SELF, HEAP):
        SELF.HEAP = HEAP
        SELF.SIZE = LEN(HEAP)
        SELF.RESULT = []

    DEF GET_LEFT_CHILD(SELF, INDEX):
        RETURN INDEX * 2 + 1

    DEF GET_RIGHT_CHILD(SELF, INDEX):
        RETURN INDEX * 2 + 2

    DEF SIFT_DOWN(SELF, INDEX):
        CURRENT = INDEX
        LEFT = SELF.GET_LEFT_CHILD(INDEX)
        IF LEFT < SELF.SIZE:
            IF SELF.HEAP[LEFT][1] < SELF.HEAP[CURRENT][1]:
                CURRENT = LEFT
            ELIF   SELF.HEAP[LEFT][1]   ==   SELF.HEAP[CURRENT][1]   AND
SELF.HEAP[LEFT][0] < SELF.HEAP[CURRENT][0]:
                CURRENT = LEFT
        RIGHT = SELF.GET_RIGHT_CHILD(INDEX)
        IF RIGHT < SELF.SIZE:
            IF SELF.HEAP[RIGHT][1] < SELF.HEAP[CURRENT][1]:
                CURRENT = RIGHT
            ELIF   SELF.HEAP[RIGHT][1]   ==   SELF.HEAP[CURRENT][1]   AND
SELF.HEAP[RIGHT][0] < SELF.HEAP[CURRENT][0]:
                CURRENT = RIGHT
        IF CURRENT != INDEX:
            SELF.HEAP[CURRENT],  SELF.HEAP[INDEX]  =  SELF.HEAP[INDEX],
SELF.HEAP[CURRENT]
            SELF.SIFT_DOWN(CURRENT)

    DEF PERFORMING(SELF, ILIST):
        FOR I IN ILIST:
            SELF.RESULT.APPEND([SELF.HEAP[0][0], SELF.HEAP[0][1]])
            SELF.HEAP[0][1] += I
            SELF.SIFT_DOWN(0)


IF __NAME__ == "__MAIN__":
    N, M = MAP(INT, INPUT().SPLIT())
    HEAP = HEAP([[I, 0] FOR I IN RANGE(N)])
    HEAP.PERFORMING(LIST(MAP(INT, INPUT().SPLIT())))
    PRINT(HEAP.RESULT)
    FOR CURRENT IN HEAP.RESULT:
        PRINT(CURRENT[0], CURRENT[1])
```