**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. 1303 |  | Чубан Д.В. |
| --- | --- | --- |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2022

* 1. **Цель работы.**

Изучение структуры данных «мин-куча», реализация данной структуры данных для решения задачи распараллеливания обработки.

* 1. **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.

**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.**

* 1. 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.
  2. **Выполнение работы.**

Для реализации данной задачи был реализован класс *Processor.* Полями процессора являются *id* (уникальный номер процессора) и поле *time,* в котором хранится общее время работы процессора. Для этого класса был перегружен оператор сравнения «<».

Кроме того, была реализована функция *heapify(arr, i)*, преобразующая список в мин-кучу. Индекс родительского узла передается в функцию, вычисляются правый и левый ребенок. Затем каждый из детей сравнивается с родительским узлом. В случае, если родительский узел не меньше детей, то минимальный из них ставится на место родительского узла, а затем функция рекурсивно вызывается вновь, но в качестве индекса родительского узла передается индекс ребенка.

Функция вызывается в цикле для элементов списка, начиная с середины списка, до -1 элемента с шагом один. Такой проход позволяет восстановить правильный порядок элементов мин-кучи.

**Тестирование программы.**

Тестирование функции сортировки производится с помощью unit-тестов, описанных в файле *test.py.* Данные тесты покрывают следующие случаи:

* На вход подается 2 процессора и 5 различных задач
* На вход подается несколько задач для одного процессора
* На вход подаются 2 процессора и список задач, таких что время выполнения одной больше суммы времени выполнения остальных задач
* На вход подается несколько процессоров, кол-во задач кратно их числу, каждая задача выполняется за 1 единицу времени.

**Вывод.**

В ходе выполнения данной лабораторной работы была освоена структура данных «мин-куча». Реализована программа, выполняющая распределение задач для параллельного выполнения процессорами. Написанная программа покрыта юнит-тестами.

1. **Приложение А  
   Исходный код программы**

Название файла: main.py

class Processor:

def \_\_init\_\_(self, time, id):

self.time = time

self.id = id

def \_\_lt\_\_(self, other):

if self.time == other.time:

return self.id < other.id

return self.time < other.time

def heapify(arr, i):

minn = i

l = 2 \* i + 1

r = 2 \* i + 2

if l < len(arr) and arr[l] < arr[minn]:

minn = l

if r < len(arr) and arr[r] < arr[minn]:

minn = r

if minn != i:

arr[i], arr[minn] = arr[minn], arr[i]

heapify(arr, minn)

n, m = map(int, input().split())

time\_arr = list(map(int, input().split()))

proc\_arr = [Processor(0, i) for i in range(n)]

start = len(proc\_arr) // 2 - 1

for elem in time\_arr:

for i in range(start, -1, -1):

heapify(proc\_arr, i)

print(proc\_arr[0].id, proc\_arr[0].time)

proc\_arr[0].time += elem

1. **Приложение Б  
   Исходный код UNIT-ТЕСТОВ**

Название файла: test.py

from main import \*

import pytest

def test\_normal\_proc():

time\_arr = [1, 2, 3, 4, 5]

proc\_arr = [Processor(0, i) for i in range(2)]

res = []

for elem in time\_arr:

for i in range(start, -1, -1):

heapify(proc\_arr, i)

res.append((proc\_arr[0].id, proc\_arr[0].time))

proc\_arr[0].time += elem

assert res == [(0, 0), (1, 0), (0, 1), (1, 2), (0, 4)]

def test\_one\_proc():

time\_arr = [1, 2, 3]

proc\_arr = [Processor(0, i) for i in range(1)]

res = []

for elem in time\_arr:

for i in range(start, -1, -1):

heapify(proc\_arr, i)

res.append((proc\_arr[0].id, proc\_arr[0].time))

proc\_arr[0].time += elem

assert res == [(0, 0), (0, 1), (0, 3)]

def test\_two\_proc():

time\_arr = [15, 2, 3, 4, 5]

proc\_arr = [Processor(0, i) for i in range(2)]

res = []

for elem in time\_arr:

for i in range(start, -1, -1):

heapify(proc\_arr, i)

res.append((proc\_arr[0].id, proc\_arr[0].time))

proc\_arr[0].time += elem

assert res == [(0, 0), (1, 0), (1, 2), (1, 5), (1, 9)]

def test\_little\_time():

time\_arr = [1, 1, 1, 1, 1, 1, 1, 1]

proc\_arr = [Processor(0, i) for i in range(4)]

res = []

for elem in time\_arr:

for i in range(start, -1, -1):

heapify(proc\_arr, i)

res.append((proc\_arr[0].id, proc\_arr[0].time))

proc\_arr[0].time += elem

assert res == [(0, 0), (1, 0), (2, 0), (3, 0), (0, 1), (1, 1), (2, 1), (3, 1)]