**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: **Очереди с приоритетом. Параллельная обработка**.

|  |  |  |
| --- | --- | --- |
| Студент гр. 0381 |  | Дзаппала Д. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2021

## Цель работы.

Изучить и реализовать структуру данных «Очередь с приоритетом», основанную на двоичной мин куче.

## 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: 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, которыйполучит i-ю задачу на обработку, и время, когда это произойдёт.

## Выполнение работы.

Была реализована структура proc\_t, представляющая процессор, и которая хранит номер процессора, и время обработки задачи.

Для выполнения задания, был написан класс SolveProblem, который хранит вектор процессоров и вектор времен. Метод problemSolving решает саму задачу и возвращает результативный вектор (нужен для тестов). В методе запускается цикл по вектору времени, на каждом шаге цикла печатается процессор, находящийся в корне бин мин кучи. Печатается именно он, так как на каждом шаге идет просейвание кучи вниз, в результате чего в корне получаем процессор, у которого минимальное время выполнения задачи (или, в случае одинаковых времен, берется процессор с меньшим номером), и следовательно, оно освободиться раньше, и возьмет на себя выполнение следующей задачи. На каждом шаге время работы корневого процессора увеличивается.

## Выводы.

Была изучена работа данной структуры данных. А также решана задача.

# Приложение А Исходный код программы

Название файла: src.cpp

#include<iostream>

#include<vector>

#include<cassert>

struct proc\_t{

long count = 0;

long time = 0;

friend std::ostream& operator<<(std::ostream& output, proc\_t& el){

output << el.count << " " << el.time;

return output;

}

friend bool operator==(const proc\_t& a, const proc\_t& b){

return a.count == b.count && a.time == b.time;

}

};

class SolveProblem{

public:

SolveProblem(const std::vector<proc\_t>& \_procs,

const std::vector<long>& \_times) : procs(\_procs), times(\_times){

}

SolveProblem(long procC, long timesC) : procs(procC), times(timesC){

for (long i = 0; i < procC; ++i){

procs[i].count = i;

}

long t;

for (long i = 0; i < timesC; ++i){

std::cin >> t;

times[i] = t;

}

}

std::vector<proc\_t>& problemSolving(){

for(long& time : times){

res.push\_back(procs[0]);

std::cout << procs[0] << std::endl;

procs[0].time += time;

siftDown(0);

}

return res;

}

void siftDown(long ind){

long newInd = ind;

long leftC = 2 \* ind + 1;

if (leftC < procs.size() && (procs[leftC].time < procs[newInd].time ||

(procs[leftC].time == procs[newInd].time &&

procs[leftC].count < procs[newInd].count))) newInd = leftC;

long rightC = 2 \* ind + 2;

if (rightC < procs.size() && (procs[rightC].time < procs[newInd].time ||

(procs[rightC].time == procs[newInd].time &&

procs[rightC].count < procs[newInd].count))) newInd = rightC;

if (ind != newInd){

std::swap(procs[newInd], procs[ind]);

siftDown(newInd);

}

}

private:

std::vector<proc\_t> procs;

std::vector<long> times;

//test result

std::vector<proc\_t> res;

};

void test(){

int i = 0;

std::cout << "Test #" << ++i << ":" << std::endl;

SolveProblem pr1({{0, 0}, {1, 0}, {2, 0}},

{1, 2, 3, 4, 5});

std::vector<proc\_t> ans1{{0, 0}, {1, 0}, {2, 0}, {0, 1}, {1, 2}};

assert(pr1.problemSolving() == ans1 && "Failed!");

std::cout << "Passed!" << std::endl << std::endl;

std::cout << "Test #" << ++i << ":" << std::endl;

SolveProblem pr2({{0, 0}, {1, 0}, {2, 0}},

{2, 4, 3, 8, 6, 9, 5, 7});

std::vector<proc\_t> ans2{{0, 0}, {1, 0}, {2, 0}, {0, 2}, {2, 3}, {1, 4}, {2, 9}, {0, 10}};

assert(pr2.problemSolving() == ans2 && "Failed!");

std::cout << "Passed!" << std::endl << std::endl;

std::cout << "Test #" << ++i << ":" << std::endl;

SolveProblem pr3({{0, 0}, {1, 0}},

{0, 0, 1, 0, 0, 0, 2, 1, 2, 3, 0, 0, 0, 2, 1});

std::vector<proc\_t> ans3{{0, 0}, {0, 0}, {0, 0}, {1, 0}, {1, 0},

{1, 0}, {1, 0}, {0, 1}, {0, 2}, {1, 2},

{0, 4}, {0, 4}, {0, 4}, {0, 4}, {1, 5}};

assert(pr3.problemSolving() == ans3 && "Failed!");

std::cout << "Passed!" << std::endl << std::endl;

}

int main(){

// long procCount, timesCount;

// std::cin >> procCount >> timesCount;

// SolveProblem pr(procCount, timesCount);

// std::vector<proc\_t> res = pr.problemSolving();

test();

return 0;}