МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФБГОУ ВО «ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра «Программная инженерия»

**Отчёт**

По лабораторной работе №6

## «Сортировка во внешней памяти многопутевым сбалансированным слиянием»

**Выполнили:**

Студенты группы 92-ПГ

Погосян Ж.В.

**Проверили:**

Ужаринский А.Ю.

Орёл — 2021

**Код программы:**

#include <iostream>

#include <fstream>

#include <string>

#include <chrono>

#include <vector>

#include <algorithm>

struct Test {

int n;

int step;

double time;

};

using namespace std;

struct Data {

int key;

char c[200 - sizeof(int)];

};

const char fname[] = "FILE";

void CreateFile();

void DataInit(Data& z);

int Merge(int n);

void PushBack(int array[], int i, int length);

int main()

{

setlocale(0, "");

CreateFile();

vector<struct Test> v;

while (1) {

cout

<< "1 - Выполнить слияние\n"

<< "2 - Вывести таблицу опытов\n"

<< "0 - Выход\n";

int input;

cin >> input;

switch (input) {

default:

break;

case 1:

{

cout << "Выберите число путей слияния: ";

cin >> input;

chrono::high\_resolution\_clock::time\_point t1 = chrono::high\_resolution\_clock::now();

int result = Merge(input);

chrono::high\_resolution\_clock::time\_point t2 = chrono::high\_resolution\_clock::now();

if (result == -1)

cout << "Ошибка при слиянии!\n";

else {

chrono::duration<double> time\_span = chrono::duration\_cast<chrono::duration<double>>(t2 - t1);

cout << "Проходов: " << result << ", время слияния: " << time\_span.count() << " секунд\n";

v.push\_back({ input, result, time\_span.count() });

}

break;

}

case 2:

{

sort(v.begin(), v.end(), [](Test a, Test b) {return a.n < b.n; });

cout << "Путей\tКол-во проходов\tВремя\n";

for (auto x : v) {

cout << x.n << "\t" << x.step << "\t\t" << x.time << '\n';

}

break;

}

case 0:

return 0;

}

}

return 0;

}

void CreateFile() {

ofstream output(fname, ios::binary | ios::trunc);

Data z;

const int elements = 10000;

for (int i = 0; i < elements; i++) {

DataInit(z);

z.key = rand() % 10000;

output.write((char\*)&z, sizeof(Data));

}

}

void DataInit(Data& z) {

z.key = rand() % 10000;

}

int Merge(int n) {

Data buffer[5];

string\* fnames = new string[2 \* n];

for (int i = 0; i < 2 \* n; i++) {

fnames[i] = "f" + to\_string(i);

}

ifstream input(fname, ios::binary);

fstream\* outputs = new fstream[2 \* n];

for (int i = 0; i < 2 \* n; i++) {

outputs[i].open(fnames[i].c\_str(), ios::binary | ios::in | ios::out | ios\_base::ate | ios::trunc);

}

//распределение

int j = 0;

int l = 0;

int prevKey = -1;

int key = -1;

while (!input.eof()) {

Data tmp;

int pos = input.tellg();

input.read((char\*)&tmp, sizeof(Data));

if (input.eof())

break;

prevKey = key;

key = tmp.key;

if (key == 6970)

int lol = 1;

if (key >= prevKey) {

outputs[j].write((char\*)&tmp, sizeof(Data));

outputs[j].flush();

}

else {

input.seekg(pos);

key = -1;

j = (j + 1) % n;

l++;

}

}

l++;

//Инициализация индексной карты

int\* t = new int[n \* 2];

for (int i = 0; i < n \* 2; i++) {

t[i] = i;

}

//слияние из f[t[1]].. f[t[N]] в f[t[N+1]].. f[t[2N]] \*

int\* ta = new int[n];

for (int i = 0; i < n; i++) {

ta[i] = t[i];

}

int step = 0;

while (l > 1) {

int af = min(l, n);

/\* открыть для чтения f[t[1]]..f[t[af]]

открыть для записи f[t[N + 1]]..f[t[2N]]

инициализировать ta[1]..ta[af] индексами из ta[]\*/

for (int i = 0; i < af; i++) {

outputs[t[i]].close();

outputs[t[i]].clear();

outputs[t[i]].open(fnames[t[i]].c\_str(), ios::binary | ios::in);

outputs[t[i]].seekg(0);

}

for (int i = n; i < 2 \* n; i++) {

outputs[t[i]].close();

outputs[t[i]].clear();

outputs[t[i]].open(fnames[t[i]].c\_str(), ios::binary | ios::out | ios\_base::ate | ios::trunc);

outputs[t[i]].seekp(0);

}

for (int i = 0; i < n; i++) {

ta[i] = t[i];

}

l = 0;

j = n;

/\*ПРОХОД\*/

step++;

while (af != 0) {

l++;

if (l > 1) {

int abc = 0;

}

int ao = af;

//слияние первых активных отрезков в f[t[j]]

while (ao != 0) {

int minFile, minimum = 999999;

//Поиск m

Data buf;

for (int i = 0; i < ao; i++) {

int res;

int pos = outputs[ta[i]].tellg();

outputs[ta[i]].read((char\*)&buf, sizeof(Data));

res = buf.key;

if (res < minimum) {

minimum = res;

minFile = i;

}

outputs[ta[i]].seekg(pos);

}

int m = minFile;

int tmp;

//скопировать запись из f[ta[m]] в f[t[j]]

outputs[ta[m]].read((char\*)&buf, sizeof(Data));

tmp = buf.key;

outputs[t[j]].write((char\*)&buf, sizeof(Data));

outputs[t[j]].flush();

{

/\*if конец отрезка в f[ta[m]] then

модифицировать ao, ta[] \*/

int res;

int pos = outputs[ta[m]].tellg();

outputs[ta[m]].read((char\*)&res, sizeof(int));

bool fileEnd = outputs[ta[m]].eof();

outputs[ta[m]].seekg(pos);

if (fileEnd) {

// модифицировать af, ao, ta[]

af--;

ao--;

PushBack(ta, m, n);

}

else {

if (res < tmp) {

ao--;

PushBack(ta, m, ao + 1);

}

}

}

}

// переключить j на следующий выходной файл

j++;

if (j == 2 \* n)

j = n;

}

// переключить индексную карту

for (int i = 0; i < n; i++) {

int tmp = t[i];

t[i] = t[i + n];

t[i + n] = tmp;

}

}

//отсортированная последовательность в f[t[1]]

outputs[t[0]].clear();

outputs[t[0]].seekg(0);

int keyT = -1;

int prevT = -1;

while (1) {

Data test;

outputs[t[0]].read((char\*)&test, sizeof(Data));

if (outputs[t[0]].eof()) {

break;

}

prevT = keyT;

keyT = test.key;

if (prevT > keyT) {

cout << "ERROR\n";

return -1;

}

}

return step;

}

void PushBack(int array[], int i, int length) {

int tmp = array[i];

for (int j = i; j < length - 1; j++) {

array[j] = array[j + 1];

}

array[length - 1] = tmp;

}

**Результаты**

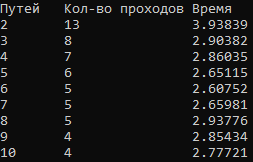


Рисунок 1 - Результаты тестирования



Рисунок 2 - Зависимость количества проходов и времени работы алгоритма от количества путей