**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

по дисциплине «Алгоритмы и структуры данных»

на тему: «Сортировка»

Выполнил: студент гр. ИП-31

Казутин П. Н.

Принял: ст. преподаватель

Косинов Г. П.

Гомель 2021

**Цель работы:** изучить основные методы сортировки. Сравнить эффективность методов.

**Практическая часть:**

Сформировать случайным образом массив записей из 100000 штук. Скопировать его трижды. Провести сортировку по одному ключу, двум ключам и трем ключам. Оценить эффективность каждой сортировки по времени и количеству перестановок и сравнений. Провести четвертую сортировку, включив проверку на каждом шаге, что массив уже отсортирован. Оценить эффективность включения такой оценки. Организовать поиск по трем указанным ключам. Оценить эффективность поиска.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 9 | Data.data.str | У.УУ.ВВВ | шейкер | С барьером |

***AISD\_lab3.cpp***

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <chrono>

#include <random>

#include <vector>

#include "record.h"

#include "sorter.h"

#include "searcher.h"

#define DATA\_SIZE 10000

//#define PRINT

time\_t inputDate();

bool cmpFirst\_Up(record& a, record& b);

void print(std::vector<record>& data);

bool cmpFirst\_Down(record& a, record& b);

bool cmpSecond\_Up(record& a, record& b);

bool cmpSecond\_Down(record& a, record& b);

bool cmpThird\_Up(record& a, record& b);

bool cmpThird\_Down(record& a, record& b);

bool cmpFirst\_Exact(record& a, record& b);

bool cmpSecond\_Exact(record& a, record& b);

bool cmpThird\_Exact(record& a, record& b);

bool cmpAll\_Exact(record& a, record& b);

using namespace std::chrono;

int main()

{

srand(rand());

std::ios::sync\_with\_stdio(false);

std::cout << "Lab 3!\n";

std::vector<record> data;

data.resize(DATA\_SIZE);

for (int i = 0; i < DATA\_SIZE; i++)

{

data[i] = record();

}

#pragma region sorting

auto sc = score();

auto before = steady\_clock::now();

sorter::SortByFirst(data, cmpFirst\_Down, sc);

auto after = steady\_clock::now();

std::cout << "\nSorted by 1-st field:\n\n";

#ifdef PRINT

print(data);

#endif // !PRINT

std::cout << "Permutations: " << sc.permutations << "; compares: " << sc.compares << "; seconds: " << (after - before).count() \* 1E-9 << "\n";

before = steady\_clock::now();

sorter::SortByTwo(data, cmpSecond\_Down, sc);

after = steady\_clock::now();

std::cout << "\nSorted by 2-nd field:\n\n";

#ifdef PRINT

print(data);

#endif

std::cout << "Permutations: " << sc.permutations << "; compares: " << sc.compares << "; seconds: " << (after - before).count() \* 1E-9 << "\n";

sc.reset();

before = steady\_clock::now();

sorter::SortByFirst(data, cmpFirst\_Up, sc);

sorter::SortByTwo(data, cmpSecond\_Up, sc);

sorter::SortByThree(data, cmpThird\_Up, sc);

after = steady\_clock::now();

std::cout << "\nSorted by 3-rd field:\n\n";

#ifdef PRINT

print(data);

#endif // PRINT

std::cout << "Permutations: " << sc.permutations << "; compares: " << sc.compares << "; seconds: " << (after - before).count() \* 1E-9 << "\n";

//with check!

sc.reset();

for (int i = 0; i < DATA\_SIZE; i++)

{

data[i] = record();

}

before = steady\_clock::now();

sorter::SortByFirst(data, cmpFirst\_Up, sc, true);

sorter::SortByTwo(data, cmpSecond\_Up, sc, true);

sorter::SortByThree(data, cmpThird\_Up, sc, true);

after = steady\_clock::now();

std::cout << "\nSorted by 3-rd field with checking:\n\n";

#ifdef PRINT

print(data);

#endif // PRINT

std::cout << "Permutations: " << sc.permutations << "; compares: " << sc.compares << "; seconds: " << (after - before).count() \* 1E-9 << "\n";

#pragma endregion

#pragma region search

cout << "\tEnter first date: \n";

time\_t firstKey = inputDate();

record tmp;

tmp.field\_1 = firstKey;

int result = searcher::barierSearch(data, cmpFirst\_Exact, tmp);

if (result > 0)

cout << data[result].to\_string() << '\n';

else

cout << "not found!" << '\n';

cout << "\tEnter second date: \n";

time\_t secondKey = inputDate();

tmp.field\_2 = secondKey;

result = searcher::barierSearch(data, cmpSecond\_Exact, tmp);

if (result > 0)

cout << data[result].to\_string() << '\n';

else

cout << "not found!\n";

cout << "\tEnter thrid key: \n";

string thirdKey;

cin >> thirdKey;

thirdKey = thirdKey.substr(0, 5);

tmp.field\_3 = &thirdKey;

result = searcher::barierSearch(data, cmpThird\_Exact, tmp);

if (result > 0)

cout << data[result].to\_string() << '\n';

else

cout << "not found!" << '\n';

cout << "\t Exact search: \n";

result = searcher::barierSearch(data, cmpAll\_Exact, tmp);

if (result > 0)

cout << data[result].to\_string() << '\n';

else

cout << "not found!" << '\n';

#pragma endregion

}

time\_t inputDate() {

cout << "Enter day: ";

int day(0);

cin >> day;

cout << "Enter month: ";

int month(0);

cin >> month;

cout << "Enter year: ";

int year(0);

cin >> year;

tm\* dateStructure = new tm();

dateStructure->tm\_year = year;

dateStructure->tm\_mon = month - 1;

dateStructure->tm\_mday = day;

return mktime(dateStructure);

}

void print(std::vector<record>& data)

{

for (int i = 0; i < DATA\_SIZE; i++)

{

std::cout << data[i].to\_string() << "\r\n";

}

}

bool cmpFirst\_Up(record& a, record& b) {

return a.field\_1 >= b.field\_1;

}

bool cmpFirst\_Down(record& a, record& b) {

return a.field\_1 <= b.field\_1;

}

bool cmpSecond\_Up(record& a, record& b) {

return a.field\_2 > b.field\_2;

}

bool cmpSecond\_Down(record& a, record& b) {

return a.field\_2 < b.field\_2;

}

bool cmpThird\_Up(record& a, record& b) {

return a.field\_3->compare(\*b.field\_3) >= 0;

}

bool cmpThird\_Down(record& a, record& b) {

return a.field\_3->compare(\*b.field\_3) <= 0;

}

bool cmpFirst\_Exact(record& a, record& b) {

return a.field\_1 == b.field\_1;

}

bool cmpSecond\_Exact(record& a, record& b) {

return a.field\_2 == b.field\_2;

}

bool cmpThird\_Exact(record& a, record& b) {

return a.field\_3->compare(\*b.field\_3) == 0;

}

bool cmpAll\_Exact(record& a, record& b) {

return a.field\_3->compare(\*b.field\_3) == 0

&& a.field\_2 == b.field\_2

&& a.field\_1 == b.field\_1;

}

***record.h***

#pragma once

#pragma warning(disable : 4996)

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <ctime>

#include <string>

#include <iomanip>

#include <random>

#include <sstream>

using namespace std;

class record

{

public:

record() {

auto field\_1\_tm = new tm();

auto field\_2\_tm = new tm();

field\_1\_tm->tm\_year = field\_2\_tm->tm\_year = 121;

field\_1\_tm->tm\_mday = field\_2\_tm->tm\_mday = rand() % 10;

field\_1\_tm->tm\_mon = rand() % 6 + 1;

field\_2\_tm->tm\_mon = rand() % 4 + 3;

field\_1 = mktime(field\_1\_tm);

field\_2 = mktime(field\_2\_tm);

delete field\_1\_tm;

delete field\_2\_tm;

field\_3 = new string(" ");

for (int i(0); i < 5; i++) {

(\*field\_3)[i] = rand() % 5 + 97;

}

}

string to\_string()

{

std::stringstream ss;

char\* buffer = new char[255]{ 0 };

strftime(buffer, 255, "%d/%m/%G", localtime(&this->field\_1));

auto tmp = string(buffer);

ss << tmp << "; ";

strftime(buffer, 255, "%d/%m/%G", localtime(&this->field\_2));

tmp = string(buffer);

ss << tmp << "; " << \*this->field\_3;

return ss.str();

}

time\_t field\_1 = 0;

time\_t field\_2 = 0;

string\* field\_3 = nullptr;

};

class score {

public:

unsigned long permutations = 0;

unsigned long compares = 0;

void reset() {

permutations = compares = 0;

}

};

***searcher.h***

#pragma once

#include <functional>

#include <vector>

#include "record.h"

using namespace std;

class searcher

{

public:

static int barierSearch(vector<record>& data, function<bool(record&, record&)> comparer, record& target);

};

***searcher.cpp***

#include "searcher.h"

int searcher::barierSearch(vector<record>& data, function<bool(record&, record&)> comparer, record& target)

{

record rec = data.back();

(\*data.rbegin()) = target;

int i = 0;

while (!comparer(data[i], target)) {

i++;

}

if (comparer(rec, target)) {

(\*data.rbegin()) = rec;

return i;

}

else {

(\*data.rbegin()) = rec;

return -1;

}

***sorter.h***

#pragma once

#include "record.h"

#include <functional>

class sorter

{

public:

static void SortByFirst(std::vector<record>& array, std::function<bool(record&, record&)> cmp, score& sc, bool check = false);

static void SortByTwo(std::vector<record>& array, std::function<bool(record&, record&)> cmp, score& sc, bool check = false);

static void SortByThree(std::vector<record>& array, std::function<bool(record&, record&)> cmp, score& sc, bool check = false);

static bool isSorted(std::vector<record>& array, std::function<bool(record&, record&)> cmp, score& sc);

};

***sorter.cpp***

#include "sorter.h"

void sorter::SortByFirst(std::vector<record>& array, std::function<bool(record&, record&)> cmp, score& sc, bool check) {

//шейкерная сортировка

int control = array.size() - 1;

int left = 0;

int right = array.size() - 1;

do {

for (int i = left; i < right; i++) {

sc.compares++;

if (cmp(array[i], array[i + 1])) {

std::swap(array[i], array[i + 1]);

sc.permutations++;

control = i;

}

}

right = control;

for (int i = right; i > left; i--) {

sc.compares++;

if (!cmp(array[i], array[i - 1])) {

sc.permutations++;

std::swap(array[i], array[i - 1]);

control = i;

}

}

left = control;

if (check && sorter::isSorted(array, cmp, sc))

break;

} while (left < right);

}

void sorter::SortByTwo(std::vector<record>& array, std::function<bool(record&, record&)> cmp, score& sc, bool check) {

//SortByFirst(array);

auto element = array[0].field\_1;

int left = 0, right = 1;

bool sortFlag = true;

while (sortFlag && right < array.size())

{

while (array.size() > right && element == array[right].field\_1)

{

sc.compares++;

right++;

}

if (right < array.size())

element = array[right].field\_1;

else

{

sortFlag = false;

continue;

}

right--;

int control = right;

do {

for (int i = left; i < right; i++) {

sc.compares++;

if (cmp(array[i], array[i + 1])) {

sc.permutations++;

std::swap(array[i], array[i + 1]);

control = i;

}

}

right = control;

for (int i = right; i > left; i--) {

sc.compares++;

if (!cmp(array[i], array[i - 1])) {

sc.permutations++;

std::swap(array[i], array[i - 1]);

control = i;

}

}

left = control;

if (check && sorter::isSorted(array, cmp, sc))

break;

} while (left < right);

right++;

left = right;

}

}

void sorter::SortByThree(std::vector<record>& array, std::function<bool(record&, record&)> cmp, score& sc, bool check) {

//SortByTwo(array);

auto element = array[0];

int left = 0, right = 1;

bool sortFlag = true;

while (right < array.size())

{

while (right < array.size()

&& element.field\_2 == array[right].field\_2

&& element.field\_1 == array[right].field\_1)

{

sc.compares += 2;

right++;

}

if (right < array.size())

element = array[right];

else

{

sortFlag = false;

continue;

}

right--;

int control = right;

do {

for (int i = left; i < right; i++) {

sc.compares++;

if (cmp(array[i], array[i + 1])) {

sc.permutations++;

std::swap(array[i], array[i + 1]);

control = i;

}

}

right = control;

for (int i = right; i > left; i--) {

sc.compares++;

if (!cmp(array[i], array[i - 1])) {

sc.permutations++;

std::swap(array[i], array[i - 1]);

control = i;

}

}

left = control;

if (check && sorter::isSorted(array, cmp, sc))

break;

} while (left < right);

right++;

left = right;

}

}

bool sorter::isSorted(std::vector<record>& array, std::function<bool(record&, record&)> cmp, score& sc)

{

bool isSorted = true;

for (int i(0); i < array.size() - 1 && isSorted; i++) {

sc.compares++;

if (!cmp(array[i], array[i + 1])) {

isSorted = false;

}

}

return isSorted;

}

Результат выполнения:

Lab 3!

Sorted by 1-st field:

09/07/2021; 09/05/2021; cadae

06/07/2021; 06/06/2021; dbecd

05/07/2021; 05/06/2021; edeca

30/06/2021; 30/06/2021; ecaed

05/06/2021; 05/06/2021; ccabb

02/05/2021; 02/06/2021; ebcdb

02/04/2021; 02/07/2021; bbeaa

09/02/2021; 09/07/2021; cecea

08/02/2021; 08/06/2021; abeeb

01/02/2021; 01/06/2021; cecde

Permutations: 25; compares: 38; seconds: 2.1e-06

Sorted by 2-nd field:

09/07/2021; 09/05/2021; cadae

06/07/2021; 06/06/2021; dbecd

05/07/2021; 05/06/2021; edeca

30/06/2021; 30/06/2021; ecaed

05/06/2021; 05/06/2021; ccabb

02/05/2021; 02/06/2021; ebcdb

02/04/2021; 02/07/2021; bbeaa

09/02/2021; 09/07/2021; cecea

08/02/2021; 08/06/2021; abeeb

01/02/2021; 01/06/2021; cecde

Permutations: 25; compares: 47; seconds: 2.4e-06

Sorted by 3-rd field:

01/02/2021; 01/06/2021; cecde

08/02/2021; 08/06/2021; abeeb

09/02/2021; 09/07/2021; cecea

02/04/2021; 02/07/2021; bbeaa

02/05/2021; 02/06/2021; ebcdb

05/06/2021; 05/06/2021; ccabb

30/06/2021; 30/06/2021; ecaed

05/07/2021; 05/06/2021; edeca

06/07/2021; 06/06/2021; dbecd

09/07/2021; 09/05/2021; cadae

Permutations: 45; compares: 72; seconds: 2.3e-06

Sorted by 3-rd field with checking:

28/02/2021; 30/06/2021; ceead

05/03/2021; 05/07/2021; eaacc

31/03/2021; 30/04/2021; baabc

01/04/2021; 01/06/2021; dedca

05/05/2021; 05/04/2021; eedea

07/05/2021; 07/04/2021; ebece

09/05/2021; 09/04/2021; aecda

09/05/2021; 09/06/2021; caaee

05/06/2021; 05/04/2021; abbcd

30/06/2021; 30/06/2021; bacca

Permutations: 16; compares: 79; seconds: 3.3e-06

**Вывод**: в ходе выполнения лабораторной работы изучил основные методы сортировки. Сравнил эффективность методов.