|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | | | | | |
| **Дальневосточный федеральный университет** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **Департамент программной инженерии и искусственного интеллекта** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **О Т Ч Е Т** | | | | | | | | | | | | |
| по лабораторной работе №3  дисциплина «Фундаментальные структуры данных и алгоритмы» | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Студент гр. Б9123-09.03.04прогин(2) | | | |
|  |  |  | | М.О. Привалов | |
|  | | | | | | |  |  | (подпись) | |  | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Проверил: | | |  |
|  | | | | | | |  |  | ассистент ДПИиИИ | | | |
|  | | | |  |  | |  |  |  |  | А.А. Шулятьев | |
|  | | | |  |  | |  |  | (подпись) |  | (И.О. Фамилия) | |
|  |  |  |  | | |  |  |  |  | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | |
| 2025 | | | | | | | | | | | | |

**1 Неформальная постановка задачи**

Разработать пакет подпрограмм сортировок (простой выбор и Шелла) и алгоритм, удовлетворяющий условиям:

Входные данные:

1. текстовый файл, каждая из строк которого содержит 2 ключа в табличном

виде (первый столбец - 1 ключ представлен полями паспорта (серия, номер), второй столбец - 2 ключ представлен полями ФИО (фамилия, имя, отчество) (типы полей ключей должны соответствовать типу данных в предметной области - все что является числом, должно храниться в виде числа, а не строки, например, время - это не строка, а структура из двух полей целочисленного типа). Все данные во входном файле корректные (проверять на корректность не нужно). Количество строк входного файла - 1000000.

2) целое число n - количество строк входного файла для обработки (10 ≤ n ≤ 1000000).

Выходные данные: 2 текстовых файла, строки которых содержат

отсортированные данные входного файла (первый файл – алгоритмом

сортировки простым выбором, второй файл – алгоритмом сортировки Шелла

(расстояние Шелла)), последняя строка – время, затраченное на сортировку.

1. Отсортировать данные входного файла в соответствии с заданным порядком сортировки ключей. Данные в выходном файле должны быть представлены в табличном виде, первый и второй столбец должны содержать значения ключей в соответствии с заданным порядком сортировки, третий столбец - номер строки входного файла.

2. Проверить на устойчивость (привести примеры, доказывающие, что сортировка неустойчивая).

3. Сравнить алгоритмы сортировки по времени (без учета чтения из файла и записи в файл).

4. Определить, на каких значениях ключей достигается наихудшее и наилучшее значение времени сортировки.

**2 Описание алгоритмов сортировки**

**Алгоритм сортировки «Простой выбор»**

Алгоритм сортировки простым выбором основан на принципе последовательного поиска минимального (или максимального) элемента в неотсортированной части массива и его перемещения в отсортированную часть.

Реализация алгоритма представлена главным циклом, который проходит по всем элементам массива (инкрементирует индекс (i) до достижения конца рассматриваемой части массива (m))

В цикле объявлена переменная, которая хранит индекс минимального элемента, каждую итерацию цикла она принимает значение текущего индекса (i).

(maxmin\_passport\_index = i)

Внутри главного цикла находится внутренний цикл, который проходит по элементам с индексом от i + 1 до m, используя индекс j

В условии проверяется, является ли текущий элемент  меньшим, чем элемент по индексу, чем текущий минимальный. Если это так, то минимальный элемент обновляется. (maxmin\_passport\_index = j)

После завершения внутреннего цикла, если был найден новый минимальный элемент, происходит обмен местами. Перемещаем найденный элемент на позицию i, а предыдущий элемент на место, откуда был взят новый элемент.

Цикл продолжается до тех пор, пока не будут отсортированы все элементы массива.

Алгоритм сортировки выбором имеет временную сложность O(n^2), что делает его неэффективным для больших массивов.

**Алгоритм сортировки «сортировка Шелла»**

Алгоритм сортировки Шелла основан на идее улучшения алгоритма сортировки вставками. Он использует технику "разделяй и властвуй", позволяя сортировать элементы, находящиеся на большом расстоянии друг от друга, что помогает ускорить процесс сортировки по сравнению с обычной сортировкой вставками.

В реализации выбрано как шаг (step) - расстояние Шелла.

Реализация представлена главным циклом, в котором изменяется шаг. Он представлен расстоянием Шелла, то есть step = m / 2, где m – размерность массива (число элементов). Каждую итерацию цикла будет выполняться операция деления на 2 (step = step / 2). Шаг (step) определяет расстояние между элементами, которые будут сравниваться и сортироваться.

Далее происходит сортировка вставками выделенных подмассивов.

Внутренний цикл проходит по всем элементам массива от расстояния шага (step) до конца массива (индекс – m - 1). Он отвечает за выбор текущего элемента, который будет вставлен в правильное место в отсортированной подчасти массива.

Внутри него (внутреннего цикла) происходит сохранение текущего элемента в переменную tmp.

Третий вложенный цикл выполняет функцию сравнения и определения позиции вставки элемента (вставка элемента tmp в отсортированную подчасть массива). Он сравнивает текущий элемент с элементами, находящимися на расстоянии (step), и сдвигает их вправо, если они больше текущего элемента.

По нахождению правильного места для tmp, мы вставляем его в массив.

Сортировка продолжается пока шаг больше нуля (step > 0)

Временная сложность алгоритма сортировки Шелла зависит от выбора последовательности шагов. В среднем она составляет O(n\*log(n)), но в худшем случае может быть O(n^2).

**3 Текст программы**

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

using namespace std;

struct Passport {

private:

char abc[26] {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'};

int get\_letter\_pos(char letter) const {

for (int i = 0; i < 26; i++) {

if (letter == abc[i]) {

return i;

}

}

return -1;

}

public:

int series;

int number;

string name;

string surname;

string patronymic;

int line\_number;

Passport(int series, int number, string name, string surname, string patronymic, int line\_number) : series{series}, number{number}, name{name}, surname{surname}, patronymic{patronymic}, line\_number{line\_number} {};

bool operator>(const Passport& other) const {

return (series > other.series) || (series == other.series && number > other.number);

}

bool operator<(const Passport& other) const {

return (series < other.series) || (series == other.series && number < other.number);

}

bool operator==(const Passport& other) const {

return (series == other.series) && (number == other.number);

}

bool operator<=(const Passport& other) {

int name\_1\_pos = get\_letter\_pos((char)tolower(name[0]));

int surname\_1\_pos = get\_letter\_pos((char)tolower(surname[0]));

int patronymic\_1\_pos = get\_letter\_pos((char)tolower(patronymic[0]));

int name\_2\_pos = get\_letter\_pos((char)tolower(other.name[0]));

int surname\_2\_pos = get\_letter\_pos((char)tolower(other.surname[0]));

int patronymic\_2\_pos = get\_letter\_pos((char)tolower(other.patronymic[0]));

if (name\_1\_pos < name\_2\_pos) return true;

if (name\_1\_pos > name\_2\_pos) return false;

if (surname\_1\_pos < surname\_2\_pos) return true;

if (surname\_1\_pos > surname\_2\_pos) return false;

return patronymic\_1\_pos <= patronymic\_2\_pos;

}

bool operator>=(const Passport& other) {

int name\_1\_pos = get\_letter\_pos((char)tolower(name[0]));

int surname\_1\_pos = get\_letter\_pos((char)tolower(surname[0]));

int patronymic\_1\_pos = get\_letter\_pos((char)tolower(patronymic[0]));

int name\_2\_pos = get\_letter\_pos((char)tolower(other.name[0]));

int surname\_2\_pos = get\_letter\_pos((char)tolower(other.surname[0]));

int patronymic\_2\_pos = get\_letter\_pos((char)tolower(other.patronymic[0]));

if (name\_1\_pos > name\_2\_pos) return true;

if (name\_1\_pos < name\_2\_pos) return false;

if (surname\_1\_pos > surname\_2\_pos) return true;

if (surname\_1\_pos < surname\_2\_pos) return false;

return patronymic\_1\_pos >= patronymic\_2\_pos;

}

};

const int N = 1000000;

void selection\_sort(Passport\*\* elements, int m) {

if (m >= 10) {

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

int maxmin\_passport\_index = i;

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

if ( (\*elements[j] < \*elements[maxmin\_passport\_index]) ||

(\*elements[j] == \*elements[maxmin\_passport\_index] && \*elements[j] >= \*elements[maxmin\_passport\_index]))

{

maxmin\_passport\_index = j;

}

}

Passport\* first\_elem = elements[i];

elements[i] = elements[maxmin\_passport\_index];

elements[maxmin\_passport\_index] = first\_elem;

}

} else {

cout << "Размерность массива меньше 10-ти" << endl;

}

};

void shell\_sort(Passport\*\* elements, int m) {

if (m >= 10) {

for (int step = m/2; step > 0; step /= 2) {

for (int i = step; i < m; i++) {

Passport\* tmp = elements[i];

int j;

for (j = i; j >= step && ( (\*elements[j - step] > \*tmp) ||

(\*elements[j - step] == \*tmp && \*elements[j - step] <= \*tmp) ); j -= step) {

elements[j] = elements[j - step];

}

elements[j] = tmp;

}

}

} else {

cout << "Размерность массива меньше 10-ти" << endl;

}

}

void split(string str, char del, string\* list) {

string temp = "";

int j = 0;

for (int i = 0; i < (int)str.size(); i++) {

if (str[i] != del && i != (int)str.size() - 1) {

temp += str[i];

} else {

list[j] = temp;

temp = "";

j++;

}

}

}

void get\_data(string name\_of\_file, Passport\*\* elements, int m = 10) {

string string\_item;

ifstream input\_name;

input\_name.open(name\_of\_file);

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

getline(input\_name, string\_item);

string key\_arr[5];

split(string\_item, ' ', key\_arr);

elements[i] = new Passport(stoi(key\_arr[0]), stoi(key\_arr[1]), key\_arr[2], key\_arr[3], key\_arr[4], i);

}

input\_name.close();

}

void set\_data(string name\_of\_file, Passport\*\* elements, double time, int m = 10) {

std::ofstream output;

output.open(name\_of\_file);

if (output.is\_open()) {

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

output << elements[i]->series << ' ' << elements[i]->number << ' ' << elements[i]->name << ' ' << elements[i]->surname << ' ' << elements[i]->patronymic << ' ' << elements[i]->line\_number << endl;

}

output << time << endl;

}

output.close();

}

int main() {

string path\_to\_input\_file = "inputs/input\_random.txt";

int n = 25000;

auto\*\* array = new Passport\*[1000000];

get\_data(path\_to\_input\_file, array, n);

clock\_t start = clock();

selection\_sort(array, n);

clock\_t end = clock();

double seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

set\_data("outputs/output\_selection.txt", array, seconds, n);

get\_data(path\_to\_input\_file, array, n);

start = clock();

shell\_sort(array, n);

end = clock();

seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

set\_data("outputs/output\_shell.txt", array, seconds, n);

return 0;

}

**4 Тесты**

**void selection\_sort(Passport\*\* elements, int m);**

Принимает на вход динамический массив **Passport\*\* elements** состоящий из указателей на структуры **Passport** и его размерность (число элементов) **int m**, сортирует данный массив **elements** сортировкой простым выбором. Массив передаётся по ссылке, следовательно, изменяется после вызова подпрограммы.

Входные данные: **Passport\*\*, integer.** Выходные данные: -

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Сортировка массива из элементов | **Passport\*\* elements** – динамический массив состоящий из указателей на структуру **Passport**  **int m >= 10 –** размерность массива | **Passport\*\* elements –** отсортированный динамический массив (изменённый по указателю) |
| Удаление всех элементов из пустого дерева | **Passport\*\* elements** – динамический массив состоящий из указателей на структуру **Passport**  **int m < 10 –** размерность массива | (на экран)  “Размерность массива меньше 10-ти” |

**void shell\_sort(Passport\*\* elements, int m);**

Принимает на вход динамический массив **Passport\*\* elements** состоящий из указателей на структуры **Passport** и его размерность (число элементов) **int m**, сортирует данный массив **elements** сортировкой Шелла. Массив передаётся по ссылке, следовательно, изменяется после вызова подпрограммы.

Входные данные: **Passport\*\*, integer.** Выходные данные: -

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Сортировка массива из элементов | **Passport\*\* elements** – динамический массив состоящий из указателей на структуру **Passport**  **int m >= 10 –** размерность массива | **Passport\*\* elements –** отсортированный динамический массив (изменённый по указателю) |
| Удаление всех элементов из пустого дерева | **Passport\*\* elements** – динамический массив состоящий из указателей на структуру **Passport**  **int m < 10 –** размерность массива | (на экран)  “Размерность массива меньше 10-ти” |

**5 Исследование худших и лучших случаев, проверка на устойчивость**

**Сравнение по времени выполнения**

Случай 1: Сортировка малого числа элементов массива (n = 10000)

Сортировка простым выбором – 0.304 сек

Сортировка Шелла – 0.021 сек

Случай 2: Сортировка среднего по величине массива (n = 100000)

Сортировка простым выбором – 63.415 сек

Сортировка Шелла – 1.068 сек

Случай 3: Сортировка массива большой величины (n = 1000000)

Сортировка простым выбором занимает слишком много времени из-за временной сложности алгоритма (O(n^2)).

Сортировка Шелла – 87 сек

Следующие случаи будем рассматривать на оптимальной размерности массива (n = 100000):

Случай 1: Сортировка массива, значения в котором уже отсортированы.

Сортировка простым выбором – 64.178 сек

Сортировка Шелла – 7.546 сек

Случай 2: Сортировка массива, значения в котором отсортированы в обратном заданному порядке.

Сортировка простым выбором – 170.752 сек

Сортировка Шелла – 7.506 сек

Случай 3: Сортировка массива, значения в котором расположены в случайном порядке

Сортировка простым выбором – 63.415 сек

Сортировка Шелла – 1.068 сек

Анализируя три случая, поймём, что худший случай для сортировки простым выбором, когда на вход подаётся массив отсортированный в обратном заданному порядке (~170 сек), лучший – в случайном порядке. Для Шелла худшим случаем будет уже отсортированный массив (или отсортированный в обратном порядке), а лучшим – в случайном порядке.

**Устойчивость**

Исследуем сортировку простым выбором. Возьмём уже отсортированный простым выбором набор данных. (рис. 1)

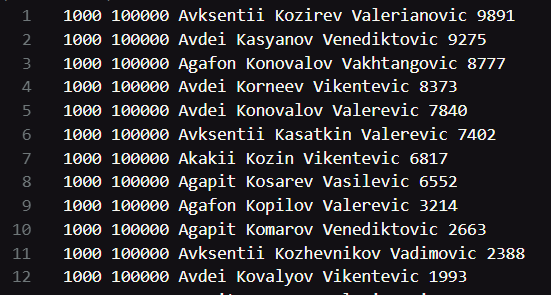


рис. 1 Набор отсортированных данных простым выбором

Данные представлены серией, номером паспорта (по возрастанию), именем, фамилией и отчеством (по убыванию: первые буквы имени, фамилии, отчества)).

Рассматривая представленные данные, можно заметить, что элементы с 1 по 12 строку имеют одинаковые ключи. Однако, если взглянуть на номера строк исходного документа, можно заметить, что они идут в обратном порядке, т. е. самый последний из одинаковых ключей в исходном документе является первым после сортировки, следовательно сортировка не устойчивая.

Также исследуем сортировку Шелла. Возьмём уже отсортированный сортировкой Шелла набор данных. (рис. 2)

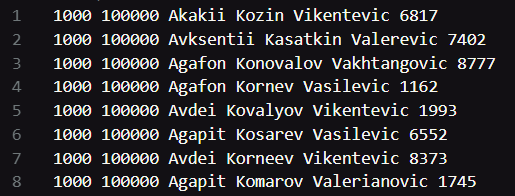


рис. 2 Набор отсортированных данных сортировкой Шелла

Данные представлены серией, номером паспорта (по возрастанию), именем, фамилией и отчеством (по убыванию: первые буквы имени, фамилии, отчества)).

Рассматривая представленные данные, можно заметить, что элементы с 1 по 12 строку тоже имеют одинаковые ключи. Но элементы взяты почти в произвольном порядке и их исходный порядок был утерян. Вывод: сортировка Шелла не является устойчивой.

**6 Вывод**

Сравнение алгоритмов сортировки (простым выбором, сортировка Шелла) показало то, что сортировка Шелла является более эффективным решением для сортировки данных, особенно на больших массивах данных.

В то же время сортировка простым выбором, несмотря на свою простоту, крайне неэффективна и даже на небольшом массиве входных значений не показала ощутимого преимущества.

Однако ни один из этих алгоритмов не является устойчивым, что важно учитывать при работе с данными, где требуется сохранить исходный порядок элементов с одинаковыми ключами.