Московский Авиационный институт
(Национальный исследовательский университет)

Факультет №8
«Компьютерные науки и прикладная математика»

Кафедра 806
«Вычислительная математика и программирование»

Курсовая работа

по теме

«Сортировка и поиск»

Студент:	Ефименко К. И.
Группа:	М8О-109Б-22
Преподаватель:	Сысоев М. А.
Подпись:	
Оценка:	

Москва, 2023

Постановка задачи

Составить и отладить программу на языке C++ с использованием функций для сортировки таблицы заданным методом и двоичного поиска по ключу в таблице.

Метод сортировки

Сортировка методом Шелла

Структура таблицы

Тип ключа: строковый
Длина ключа в байтах: 6
Хранение данных и ключей: вместе
Число элементов таблицы: 12-14

Основная часть

Описание структуры

Структура содержит произвольный набор символов - запись. Доступ осуществляется по индексу. Есть массив ключей вещественного типа, индексы ключа в массиве и строки, ему соответствующей, совпадают.

Метод решения

После запуска программы открывается указанный файл на чтение. В таблицу вводятся строки и ключи. Осуществляется вывод несортированной таблицы. После запускается функция сортировки. Если таблица уже отсортирована, то ничего не меняется. Если таблица указана наоборот, осуществляется реверс строк и ключей. Если таблица никак не отсортирована, работает алгоритм сортировки. Дальше отсортированная таблица выводится, после чего можно вывести отдельные строки по ключу. Здесь используется бинарный поиск. Если указан несуществующий ключ, работа программы прекращается.

Функциональное назначение

Программа предназначена для демонстрации использования метода сортировки, бинарного поиска, структуры таблицы.

Сортировка Шелла

При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии (о выборе значения см. ниже). После этого процедура повторяется для некоторых меньших значений , а завершается сортировка Шелла упорядочиванием элементов при (то есть обычной сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше).

Невзирая на то, что сортировка Шелла во многих случаях медленнее, чем быстрая сортировка, она имеет ряд преимуществ:

- отсутствие потребности в памяти под стек;
- отсутствие деградации при неудачных наборах данных — быстрая сортировка легко деградирует до $O(n^2)$, что хуже, чем худшее гарантированное время для сортировки Шелла.

Текст программы

Line.hpp

```
#ifndef LINE_HPP_INCLUDED
#define LINE_HPP_INCLUDED
#include <iostream>
#include "Table.hpp"

template<typename T>
class Table;

template <typename T>
class Line {
private:
    T key;
    std::string value;

public:
    friend class Table<T>;

    Line() = default;
    Line(const T& key, const std::string& data);
    Line(const Line<T>& line);
    ~Line() = default;

    T& get_key();
    std::string& get_value();
```

```
    bool operator>(const Line<T>& line) const;
};

#include "../src/Line.cpp"

#endif
```

Line.cpp

```
#include "../include/Line.hpp"

template <typename T>
Line<T>::Line(const T& key, const std::string& value): key(ke
y), value(value) {}

template <typename T>
Line<T>::Line(const Line<T>& line) {
    this->key = line.key;
    this->value = line.value;
}

template<typename T>
T& Line<T>::get_key() {
    return key;
}

template<typename T>
std::string& Line<T>::get_value() {
    return value;
}

template <typename T>
bool Line<T>::operator>(const Line<T>& line) const {
    return key > line.key;
}
```

Table.hpp

```
#ifndef TABLE_HPP_INCLUDED
#define TABLE_HPP_INCLUDED

#include "../../KP7/include/Vector.hpp"
#include "Line.hpp"

template<typename T>
class Table {
private:
    Vector<Line<T>> lines;

public:
    Table() = default;
    ~Table() = default;

    void swap(size_t index1, size_t index2);
    void push_back(const T& key, const std::string& data);
    void sort();
    Line<T> search(const T& key);

    template<typename U>
    friend std::ostream& operator<<(std::ostream& out, Table<
U>& table);
};
```

```
#include "../src/Table.cpp"

#endif
```

Table.cpp

```
#include "../include/Table.hpp"

template <typename T>
void Table<T>::push_back(const T& key, const std::string& dat
a) {
    lines.push_back(Line<T>(key, data));
}

template <typename T>
std::ostream& operator<<(std::ostream& out, Table<T>& table)
{
    for (size_t i = 0; i < table.lines.get_size(); ++i) {
        out << table.lines[i].get_key() << '\t' << table.line
s[i].get_value() << std::endl;
    }

    return out;
}

template <typename T>
void Table<T>::swap(size_t index1, size_t index2) {
    Line<T> temp;
    temp = lines[index2];
    lines[index2] = lines[index1];
    lines[index1] = temp;
}

template<typename T>
void Table<T>::sort() {
    const double factor = 1.247;
    double step = lines.get_size() - 1;

    while (step >= 1) {
        for (size_t i = 0; i + step < lines.get_size(); ++i)
{
            if (lines[i].get_key() > lines[i + step].get_key(
)) {
                swap(i, i + step);
            }
        }
        step /= factor;
    }
}

template<typename T>
Line<T> Table<T>::search(const T& key) {
    size_t left = 0;
    size_t right = lines.get_size();

    while (left <= right) {
        size_t m = left + (right - left) / 2;

        if (lines[m].get_key() == key) {
            return lines[m];
        }

        if (lines[m].get_key() < key) {
            left = m + 1;
```

```
        } else {
            right = m - 1;
        }
    }

    return Line<T>(key, "\0");
}
```

Benchmark.cpp

```
#include "../include/Table.hpp"
#include <algorithm>
#include <chrono>
#include <iostream>
#include <string>
#include <vector>

void benchmark() {
    std::cout << "Comparing my \"Sort\" and \"sort\" from STL
\n\n";

    Table<std::string> test1;
    std::vector<std::string> test2;

    for (size_t i = 0; i != 100000; ++i){
        test1.push_back("lajdf;laj" + std::to_string(i), "sad
;fnsa;df.");
        test2.push_back("asd/fn" + std::to_string(i));
    }

    std::chrono::steady_clock::time_point start_time = std::c
hrono::steady_clock::now();
    test1.sort();
    std::chrono::steady_clock::time_point end_time = std::chr
ono::steady_clock::now();
    std::cout << "Time of work my Sort: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>
(end_time - start_time).count() << " milliseconds\n";

    start_time = std::chrono::steady_clock::now();
    std::sort(test2.begin(), test2.end());
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work original sort: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>
(end_time - start_time).count() << " milliseconds\n\n";
}
```

Run.cpp

```
#include <iostream>
#include "../include/Table.hpp"
#include <string>
#include "benchmark.cpp"

int main() {
    benchmark();

    Table<std::string> table;

    table.push_back("I", "Clash Royale");
    table.push_back("B", "Brawl Stars");
    table.push_back("K", "Drive Ahead");
```

```
    table.push_back("C", "Hay Day");
    table.push_back("Y", "Clash of Clans");
    table.push_back("E", "Beach Boom");
    table.push_back("K", "Minecraft");
    table.push_back("O", "Warface");
    table.push_back("Q", "Rust");
    table.push_back("W", "CS:GO");
    table.push_back("H", "Blockade 3D");
    table.push_back("A", "Free Fire");
    table.push_back("N", "Standoff 2");
    table.push_back("V", "Roblox");
    table.push_back("U", "Flappy bird");
    table.push_back("G", "Sub Way Surfes");
    table.push_back("P", "Fruit ninja");
    table.push_back("F", "PUBG");
    table.push_back("Z", "APEX Legends");
    table.push_back("A", "SUPER STRIKE MEGA BIG SIMULATOR 300
0");

    table.sort();

    std::cout << table << std::endl;

    std::string str = "A";
    Line<std::string> search_res = table.search(str);

    if (search_res.get_value() != "\0") {
        std::cout << "Object with key '" << search_res.get_ke
y()
                  << "' found! Data inside is '" << search_
res.get_value()
                  << "'" << std::endl;

    } else {
        std::cout << "Object with key '" << search_res.get_ke
y()
                  << "' not found!" << std::endl;
    }

    return 0;
}
```

Тесты производительности

Comparing my "Sort" and "sort" from STL

Time of work my Sort: 502 milliseconds
Time of work original sort: 166 milliseconds

Моя сортировка работает медленнее, чем std::sort, я предполагаю, что это из-за сложности алгоритма. Сортировка Шелла имеет сложность n * (log n)^2, а стандартная сложность n * log n.

Заключение

В задании номер 9 КП я познакомился с различными методами сортировок, а именно с сортировкой Шелла. Реализовать ее было непросто, но довольно интересно и полезно. Так же еще я создал свою таблицу с данными, которые хранятся там по ключу и отсортировал ее по ключу. Довольно неплохо задание с прикладным смыслом.

Список литературы

1. https://habr.com/ru/companies/edison/articles/508646/
2. Методические указания к выполнению курсовых работ. Зайцев В. Е.
3. http://all-ht.ru/inf/prog/c/func/fgets.html
4. https://www.cyberforum.ru/cpp-beginners/thread401868.html