Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский Авиационный Институт»

Национальный Исследовательский Университет

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №1

по курсу «Дискретный анализ»

Студент:	Хренникова А. С.
Группа:	М8О-208Б-19
Преподаватель:	Капралов Н. С.
Подпись:	
Оценка:	
Дата:	

Москва, 2020

Лабораторная работа №1

Задача: Требуется разработать программу, осуществляющую ввод пар «ключ-значение», их упорядочивание по возрастанию ключа указанным алгоритмом сортировки за линейное время и вывод отсортированной последовательности.

Вариант сортировки: Поразрядная сортировка.

Вариант ключа: Числа от 0 до 2^{64} - 1.

Вариант значения: Строки фиксированной длины 64 символа, во входных данных могут встретиться строки меньшей длины, при этом строка дополняется до 64-х нулевыми символами, которые не выводятся на экран.

Москва, 2020

1 Описание

Для выполнения поставленной задачи необходимо реализовать поразрядную сортировку. На каждой непустой строке файла располагается пара «ключ - значение», поэтому создаем структуру TSortType, где хранится ключ и значение.

Экземпляры структуры TSortType перед сортировкой необходимо переместить в какой-нибудь контейнер, позволяющий обратиться к произвольному элементу за O(1). Для этого напишем шаблон класса, который будет представлять примерно тот же функционал, что и std::vector.

Рассмотрим поразрядную сортировку. Функция принимает вектор структур «ключ - значение» по константной ссылке, максимальное значение переданных ключей, а возвращает новый отсортированный вектор. В цикле для каждого разряда ключа осуществляется поразрядная сортировка. Внутри цикла создаются два вектора – результирующий вектор структуры и временный вектор цифр. Последний сначала инициализируется нулями. После этого в цикле рассматривается каждый элемент массива. Если значение разряда равно p, то счетчик для цифры p увеличивается на 1. Далее для каждой цифры с помощью суммирования определяется сколько разрядов элементов входного массива не превышают p. Наконец, в соответствии со значениями, хранящимися во временном векторе, заполняется результирующий массив, который и возвращается из функции.

2 Исходный код:

```
#include <iostream>
#include <cstdint>
#include <iomanip>
#include <algorithm>
#include <cmath>

template <typename T>
class TVector {
public:
    TVector() = default;

    TVector(size_t newSize)
        : data(new T[newSize]), size(newSize), capacity(newSize) {}

    TVector(size_t newSize, T defaultVal)
        : data(new T[newSize]), size(newSize), capacity(newSize) {
        for (size_t i = 0; i < size; ++i) {
            data[i] = defaultVal;
        }
    }

    TVector(const TVector& other)
        : data(new T[other.size]), size(other.size), capacity(other.size) {
        std::copy(other.begin(), other.end(), data);
    }

    TVector(TVector&& other)
        : data(other.data), size(other.size), capacity(other.capacity) {
        other.data = nullptr;
    }

    ~TVector() {
        delete[] data;
    }

    T& operator[] (size_t index) {
        return data[index];
    }

    const T& operator[] (size_t index) const {
        return data[index];
    }

    void PushBack(const T& elem) {
        if (size == capacity) {
            size_t newCap = capacity == 0 ? 1 : capacity * 2;
            T* temp = new T[newCap];
            std::copy(begin(), end(), temp);
            delete[] data;
            data = temp;
```

```
        capacity = newCap;
    }
    data[size] = elem;
    size++;
}

T* begin() {
    return data;
}

T* end() {
    return data + size;
}

const T* begin() const {
    return data;
}

const T* end() const {
    return data + size;
}

TVector& operator=(TVector& other) {
    if (&other == this) {
        return *this;
    }
    if (other.size <= capacity) {
        std::copy(other.begin(), other.end(), begin());
        size = other.size_;
    }
    else {
        TVector<T> tmp(other);
        std::swap(tmp.data, data);
        std::swap(tmp.size, size);
        std::swap(tmp.capacity, capacity);
    }
    return *this;
}

TVector& operator = (TVector&& other) {
    if (&other == this) {
        return *this;
    }
    delete[] data;
    data = other.data;
    other.data = nullptr;
    size = other.size;
    other.size = 0;
    capacity = other.capacity;
    other.capacity = 0;
    return *this;
}
```

```
    void ShrinkToFit() {
        if (size < capacity) {
            capacity = size;
            T* temp = new T[size];
            std::copy(begin(), end(), temp);
            delete[] data;
            data = temp;
        }
    }

    size_t Size() const {
        return size;
    }

private:
    T* data = nullptr;
    size_t size = 0;
    size_t capacity = 0;
};

struct TSortType {
    std::uint64_t key;
    char value[64];
};

std::ostream& operator << (std::ostream& is, TSortType& elem) {
    return is << elem.key << ' ' << elem.value;
}

TVector<TSortType> RadixSort(TVector<TSortType> v, std::uint64_t maxKey)
{
    std::uint64_t h = pow(10, 19);
    for (std::uint64_t exp = 1; maxKey / exp > 0 && exp <= h; exp *= 10) {

        int i, count[10] = { 0 };
        int k = v.Size();

        TVector<TSortType> res(k);

        for (i = 0; i < k; i++) {
            count[(v[i].key / exp) % 10]++;
            res[i] = v[i];
        }

        for (i = 1; i < 10; i++)
            count[i] += count[i - 1];

        for (i = k - 1; i >= 0; i--) {
            res[count[(v[i].key / exp) % 10] - 1] = v[i];
            count[(v[i].key / exp) % 10]--;
        }
```

```
        for (i = 0; i < k; i++) {
            v[i] = res[i];
        }

        if (exp == h)
            break;
    }
    return v;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    std::cout.tie(nullptr);
    TVector<TSortType> inputVector;
    TSortType temp;
    std::uint64_t maxKey = 0;
    while (std::cin >> temp.key >> temp.value) {
        inputVector.PushBack(temp);
        if (temp.key > maxKey) {
            maxKey = temp.key;
        }
    }
    inputVector.ShrinkToFit();
    TVector<TSortType> result = RadixSort(inputVector, maxKey);
    for (TSortType i : result) {
        std::cout << i << '\n';
    }
    return 0;
}
```

3 Консоль:

```
lina_tucha@LAPTOP-44CRFC1U:~/labs/da/lab1$ cat test1.txt
68543186543 sdfghjretdyugihojkhgfdesasdrfghjkl
546215421 sadfghhgrewrvtbbyfbhvcgxz
54154 wsdfrthyjyrtsdrsd
11 edtfyghj
0 sdfghj
54654 edffgjhikjkjhygfds
187046545450 sdfgchjgdfsgsfxcsaedtfgh
54 sdrzxtyfugkhi
4574 et
654 edfhj
EOF
lina_tucha@LAPTOP-44CRFC1U:~/labs/da/lab1$ ./main < test1.txt
0 sdfghj
11 edtfyghj
54 sdrzxtyfugkhi
654 edfhj
4574 et
54154 wsdfrthyjyrtsdrsd
54654 edffgjhikjkjhygfds
546215421 sadfghhgrewrvtbbyfbhvcgxz
68543186543 sdfghjretdyugihojkhgfdesasdrfghjkl
187046545450 sdfgchjgdfsgsfxcsaedtfgh
lina_tucha@LAPTOP-44CRFC1U:~/labs/da/lab1$ cat test2.txt
65421 asdfgh
63521 esrdfgthyjuhgfd
541 wesdfjm
45612045120 esdrghjgfdsadfhjkl;jhgfdsaxzcvbn
56231 esdrgtfyukjhgfdcx
1 esrdgtrhyui
541 erdtfgyh
46534654 rrhjyjsfsfg
4564 tdgthyujkmtnhgdfxfdsdrfxgcvsdfxcgv
EOF
lina_tucha@LAPTOP-44CRFC1U:~/labs/da/lab1$ ./main < test2.txt
1 esrdgtrhyui
541 wesdfjm
541 erdtfgyh
4564 tdgthyujkmtnhgdfxfdsdrfxgcvsdfxcgv
56231 esdrgtfyukjhgfdcx
63521 esrdfgthyjuhgfd
65421 asdfgh
46534654 rrhjyjsfsfg
45612045120 esdrghjgfdsadfhjkl;jhgfdsaxzcvbn
lina_tucha@LAPTOP-44CRFC1U:~/labs/da/lab1$ cat test3.txt
8645 wsdfxghjkhjgfxds
45 asdfghjkljhgfdsadfghjkhmngfdsa
1 wesrdgyujkhgfdsaZxc
11 aesrdtgyui
11 wasdfhjkgfd
```

```
0 dfghjk
22 jhgfdsazdfgh
3 esrtdyguij
5555555 wserdtghjikol
865421 sdxfcgvbhjnk
645 wesdfgchvbmn
6543 wsdfghj
EOF
lina_tucha@LAPTOP-44CRFC1U:~/labs/da/lab1$ ./main < test3.txt
0 dfghjk
1 wesrdgyujkhgfdsaZxc
3 esrtdyguij
11 aesrdtgyui
11 wasdfhjkgfd
22 jhgfdsazdfgh
45 asdfghjkljhgfdsadfghjkhmngfdsa
645 wesdfgchvbmn
6543 wsdfghj
8645 wsdfxghjkhjgfxds
865421 sdxfcgvbhjnk
5555555 wserdtghjikol
```

4 Тест производительности:

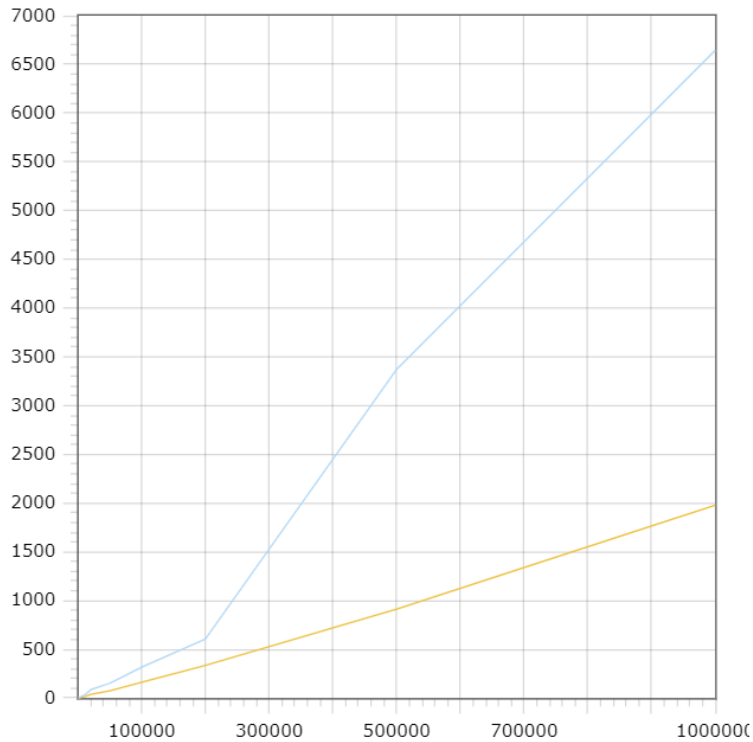
Программа для генерации тестов:

```
import random

string = "abcdefghijklmnopqrstuvwxyzabcdef" * 2
l = len(string)
f = open('test.txt', 'w')
f.write('')
f.close()

for i in range(2000000):
    x = random.randint(0, 2 ** 64 - 1)
    sint = random.randint(1, l)
    sint = int(l / sint)
    s = ''
    for j in range(0, l, sint):
        s += string[j]
    f = open('test.txt', 'a')
    f.write(str(x) + ' ' + s + '\n')
    f.close()
```

Для измерения производительности подсчитываем время работы написанной поразрядной сортировки и stable_sort на одинаковых входных данных. Замер времени работы производится с помощью библиотеки chrono, позволяющей фиксировать временные моменты: начала, конца сортировки.



На графике представлена зависимость времени сортировки от размера входных данных. По горизонтали отмечен размер входных данных, по вертикали время сортировки в ms.

Москва, 2020

Сложность сортировки подсчетом для чисел размера unsigned long long: O(d*n), где d = 20 – число разрядов. Сложность stable_sort: O(n*log(n)). До тех пор, пока d меньше, чем log(n), поразрядная сортировка будет работать медленнее.

Москва, 2020

5 Выводы:

Во время выполнения лабораторной работы я научилась работать с шаблонными классами на языке C++ на примере реализации своей структуры данных vector, разобралась в работе алгоритма сортировки подсчетом за линейное время.

Москва, 2020