Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет «Компьютерные науки и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу

«Дискретный анализ»

Тема работы

“Сортировки за линейное время”

Студент : Ю.И. Катаев
Группа : М8О-210Б-21
Преподаватель : Н.К. Макаров
Оценка : _____
Дата : _____
Подпись : _____

Москва, 2022

1. Постановка задачи

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 переменной длины (до 2048 символов).

2. Описание

Основная идея поразрядной сортировки - сортировать последовательность по каждому разряду, начиная с самого младшего (Least Significant Digit). Сортировка элементов одного разряда может происходить любым образом, но важно, чтобы она была устойчивой. Я выбрал сортировку подсчетом. Т.к. мой ключ был задан числом типа unsigned long long (8 байт), поэтому я запускал сортировку подсчетом для каждого байта. Такая реализация работает наиболее быстро.

3. Разбор программы

Для хранения входных данных создадим структуру данных Data, которая хранит ключ типа unsigned long long и значение типа char*.

```
struct Data {
   unsigned long long key;
   char* value;
};
```

Количество входных данных неизвестно, поэтому мне пришлось написать свою реализацию динамического массива. Был создан класс Vector. Если массив переполнится, то выполнится функция "AddMemory", которая выделит в два раза больше памяти под массив и копирует все старые значения.

```
void AddMemory() {
   Data* tmp = array;
   array = new Data[capacity * 2];
   for (int i = 0; i < capacity; ++i) {
      array[i] = tmp[i];
   }
   capacity *= 2;
   delete[] tmp;
}
```

Считывание происходит из файла "input.txt" с помощью метода "ReadFromFile". Т.к. значения - это строки переменной длины, то сначала я считываю строку в буфер размером 2048 байт, а далее выделяю память конкретно под эту строку с помощью "strlen". После этого, в новую память я копирую полученную строку. В динамический массив я сохраняю ключ и указатель на участок памяти с моей строкой.

Сортировка реализована в методе "RadixSort". Для каждого из 8 байтов числа я запускаю сортировку подсчетом.

В "CountingSort" я сначала нахожу текущий байт, по которому буду сравнивать ключи. Делаю это с помощью битового сдвига числа на byte * 8, где byte это порядковый номер байта, по которому будет вестись сравнение (от 0 до 7). У полученного числа я беру последний байт с помощью поразрядной конъюнкции с числом 255, тем самым получая число, которое лежит в интервале от 0 до 255. Далее в массив counter по этому индексу я делаю +1. Таким образом, в массиве counter i-ая ячейка отвечает за количество ключей, текущий байт которых равен i.

Далее я собираю префиксные суммы этого массива. После этого i-ая ячейка массива counter означает количество ключей цифра разряда которых не более i. Следующим шагом делаю копию исходного массива и выполняю сортировку. Она работает так: я иду с конца массива и смотрю текущий байт у ключа. Для него мне известно, сколько ключей не более его, значит я могу поставить ключ на место, индекс которого лежит в counter, при этом сделав -1.

4. Исходный код

```
#include <iostream>
#include <stdio.h>
#include <string.h>

struct Data {
    unsigned long long key;
    char* value;
};

class Vector {
private:
    Data* array_;
    int size_;
    int capacity_;
public:
    Vector() {
        capacity_ = 1;
        size_ = 0;
        array_ = new Data[capacity_];
    }
    void ReadFromFile(char* file_name) {
        FILE* input = fopen(file_name, "r");
        unsigned long long key;
        char buffer[2048];
        while (fscanf(input, "%llu\t%s", &key, buffer) != EOF) {
            char* str = new char[strlen(buffer) + 1];
            for (int i = 0; i < strlen(buffer); ++i) {
                str[i] = buffer[i];
            }
            str[strlen(buffer)] = '\0';
            if (size_ == capacity_) {
                AddMemory();
            }
            array_[size_] = {key, str};
            ++size_;
        }
        fclose(input);
    }
    void AddMemory() {
        Data* temp_array = array_;
        array_ = new Data[capacity_ * 2];
        for (int i = 0; i < capacity_; ++i) {
            array_[i] = temp_array[i];
        }
        capacity_ *= 2;
        delete[] temp_array;
```

```
    }
    void RadixSort() {
        for (int byte = 0; byte < 8; ++byte) {
            CountingSort(byte);
        }
    }
    void CountingSort(int byte) {
        int counter[256] = {0};
        for (int i = 0; i < size_; ++i) {
            counter[(array_[i].key >> byte * 8) & 255] += 1;
        }
        for (int i = 1; i < 256; ++i) {
            counter[i] += counter[i-1];
        }
        Data* copy_array = new Data[size_];
        for (int i = 0; i < size_; ++i) {
            copy_array[i] = array_[i];
        }
        for (int i = size_ - 1; i >= 0; --i) {
            int discharge_digit = (copy_array[i].key >> byte * 8) & 255;
             counter[discharge_digit] -= 1;
            array_[counter[discharge_digit]] = copy_array[i];

        }
        delete[] copy_array;
    }
    void Print() {
        for (int i = 0; i < size_; ++i) {
            printf("%llu\t%s\n", array_[i].key, array_[i].value);
        }
    }
    ~Vector() {
        for (int i = 0; i < size_; ++i) {
            delete[] array_[i].value;
        }
        delete[] array_;
    }
};

int main() {
    Vector data_array;
    char input_file[] = "input.txt";
    data_array.ReadFromFile(input_file);
    data_array.RadixSort();
    data_array.Print();
    return 0;
}
```

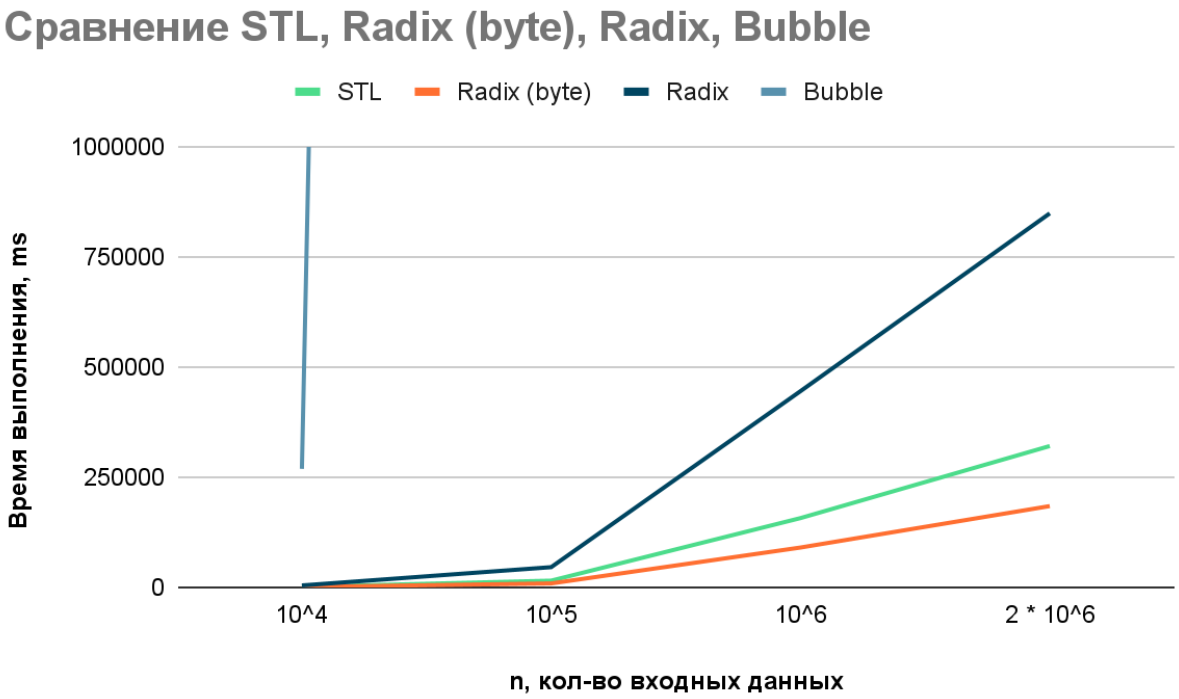
5. Консоль

```
thinkpad@ThinkPad:/mnt/d/ВУЗ/third/DA/lab1$ cat test.txt
4       x1t5iy2k99eibzh
3       ctk6g3dyav666idaz8
10      m
3       uiisisvudnbxfu6mhrgx
78      3tlmyuzqbi6szn2n4pet
thinkpad@ThinkPad:/mnt/d/ВУЗ/third/DA/lab1$ ./a.out
3       ctk6g3dyav666idaz8
3       uiisisvudnbxfu6mhrgx
4       x1t5iy2k99eibzh
10      m
78      3tlmyuzqbi6szn2n4pet
thinkpad@ThinkPad:/mnt/d/ВУЗ/third/DA/lab1$ cat test2.txt
1060    a48h92ofzbi2qpfsfx
6316    p2gc8q
6402    5msailn2h
6868    n9vkmqxfx8h
5155    6vdvt
thinkpad@ThinkPad:/mnt/d/ВУЗ/third/DA/lab1$ ./a.out
1060    a48h92ofzbi2qpfsfx
5155    6vdvt
6316    p2gc8q
6402    5msailn2h
6868    n9vkmqxfx8h
```

6. Тест производительности

Тест производительности представляет из себя следующее: сравним 4 вида сортировки на различных входных данных. Замеры времени будут проводиться с помощью библиотеки chrono. Тестируемые сортировки: 1) Radix Sort (по байтам); 2) STL Sort (встроенная); 3) Radix Sort (по десятичным разрядам); 4) Bubble Sort (наивная сортировка). По полученным данным построим графики. Сделаем оценку программы по памяти и по времени. Сравним результаты графиков и нашей оценки.

N, Кол-во элементов	Radix (optim), ms	STL, ms	Radix (byte), ms	Bubble, ms
1000	83	78	489	3416
1 * 10^4	862	1180	4421	268837
1 * 10^5	8956	15282	45852	26656441
1 * 10^6	90380	157143	445018	Очень долго
2 * 10^6	184377	320575	848254	Очень долго



Временная оценка Radix sort:
Всего байтов в ключе 8. Для каждого байта я делаю 1 проход по массиву за n для заполнения массива counter, собираю префиксные суммы countera за 256 действий, делаю копию исходного массива за n и еще один проход по массиву для сортировки. Таким образом, сложность составляет O(8 * (N + 256 + N + N)) = O(8 * (3N + 256)) = O(24N) = O(N).
Данную временную оценку подтверждает график.

Оценка по памяти Radix sort:
Чтобы хранить входной массив тратиться N памяти. Массив counter занимает в памяти 256 * 4 байта (int). Копия исходного массива занимает N памяти. Таким образом, сложность по составляет O(N + 256 * 4 + N) = O(2N + 1024) = O(2N) = O(N)

7. Выводы

Выполнив первую лабораторную работу по курсу Дискретный анализ, я узнал о сортировках за линейное время и смог реализовать Radix Sort. Вспомнил, выделять память для строки нужно на количество символов + 1, т.к. нужно сохранять терминальный символ “\0”. Научился искать куски кода, которые используют излишнюю память, и которые замедляют выполнение программы. Научился измерять скорость выполнения программы с помощью библиотеки chrono.