Лабораторная работа №1 по курсу дискретного анализа: сортировка за линейное время

Выполнил студент группы М8О-203БВ-24 *Пятницкий Артём*.

Условие

1. Требуется разработать программу, осуществляющую ввод пар «ключ-значение», их упорядочивание по возрастанию ключа указанным алгоритмом сортировки за линейное время и вывод отсортированной последовательности. Вариант задания определяется типом ключа (и соответствующим ему методом сортировки) и типом значения: Поразрядная сортировка.

2. **Вариант Т. 7-2:**

   • Тип ключа: автомобильные номера в формате A 999 BC (используются буквы латинского алфавита).

   • Тип значения: строки переменной длины (до 2048 символов).

Метод решения

Для решения данной задач, было необходимо для начала придумать способ представления данных. В качестве данных была создана структура «Car», которая хранит в себе ключ –закодированный в 32 битовом числе номер автомобиля и значение – строка переменной длинны типа «std::string». Для реализации поразрядной сортировки («Radix Sort»), я решил использовать не стандартные математические операции, а битовые операции над числом.

Способ представления автомобильного номера в число был реализован следующим способом: данные, которые нам предоставлены могут быть всего 2 типов – пустая строка или пара ключ-значение. Ключ имеет фиксированную длину – 8 символов (3 – буквы, 3 – цифры, 2 – пробела). Для такого набора данных нам подойдёт 32 битное число. 26 букв, 9 цифр, пробелы учитывать необязательно – их можно добавить на этапе форматирования вывода. Чтобы понять число бит, которое нам необходимо, надо рассчитать какое число бит требуется на 1 букву и на 1 цифру. $2^5 \geq 26 \rightarrow 5$ бит на 1 букву; $2^4 \geq 9 \rightarrow 4$ бита на 1 цифру. Итого нам необходимо $4 * 3 + 3 * 5 = 27$ бит. Для этого нам подойдёт самое стандартное 4 байтовое целочисленное неотрицательное число из библиотеки «cstdint» uint32_t (https://learn.microsoft.com/ru-ru/cpp/c-runtime-library/standard-types?view=msvc-170), которое является беззнаковым целочисленным неотрицательным типом данных. Чтобы иметь возможность, как кодировать, так и декодировать автомобильный номер, необходимо реализовать биективное отображение символов номера в числа. Для этого была составлена следующая таблица соответствия:

Буквы латинского алфавита	Число	Символы цифр от 0 до 9	Число
«A»	0	«0»	0
…	…	…	…
«Z»	25	«9»	9

Для такого представления была функция кодирования и декодирования:

Каждый символ должен занимать своё место в 32 битном числе:

00000 **00000** ***0000 0000 0000*** **00000 00000**
Буква(с 22 по 26 бит); цифра (с 18 по 21 бит); цифра (с 14 по 17 бит); цифра(с 10 по 13 бит); буква(с 5 по 9 бит); буква (с 0 бита по 4 бит)

Имея число, результат кодирования получается в результате побитового смещения вправо значения каждого символа на соответствующее число бит:

```
result += (number[0] - 'A') << 22;
result += (number[2] - '0') << 18;
result += (number[3] - '0') << 14;
result += (number[4] - '0') << 10;
result += (number[6] - 'A') << 5;
result += (number[7] - 'A');
```

Для операции декодирования, производилась обратная операция побитового сдвига влево, с умножением на соответствующую маску:

```
result[0] = ((number >> 22) & 31) + 'A';
result[2] = ((number >> 18) & 15) + '0';
result[3] = ((number >> 14) & 15) + '0';
result[4] = ((number >> 10) & 15) + '0';
result[6] = ((number >> 5) & 31) + 'A';
result[7] = (number & 31) + 'A';
```

31 – 00000000000000000000000000011111 – маска для буквы латинского алфавита

15 – 00000000000000000000000000001111 – маска для символа цифры

Реализация алгоритма сортировки предполагает за собой многократный вызов сортировки подсчётом, суть которой заключается в нахождении префиксной суммы и последующим формировании результата.

В качестве максимально возможного числа было принято число 256, так как в таблице ASCCI (https://www.ascii-code.com) нет ни 1 символа, который превышал бы это значение. Число «R» является числом, взятых бит за 1 раз, число «shift» - значение, которое показывает текущий разряд, по которому производится сортировка. Маска числа – значение состоящие из 8 единиц (в нашем случае) побитого смещённое на $R * shift$ бит. Получив префиксную сумму, можно поставить элемент на его конечную позицию путём вычисления индекса «pos». В нашем случае R = 8, нам будет необходимо 4 раза вызвать сортировку подсчётом, чтобы получить конечный результат.

Описание программы

Разделение по файлам, описание основных типов данных и функций.

Структура данных для представления:

```
struct Car {
    Car() = default;
    Car(uint32_t key, const std::string& id) : key(key), id(id) {}
    uint32_t key = 0;
    std::string id;
};
```

Функции кодирования и декодирования:

```
uint32_t encodeNumber(const std::string& number) {
    uint32_t result = 0;
    result += (number[0] - 'A') << 22;
    result += (number[2] - '0') << 18;
    result += (number[3] - '0') << 14;
    result += (number[4] - '0') << 10;
    result += (number[6] - 'A') << 5;
    result += (number[7] - 'A');
    return result;
}


std::string decodeNumber(uint32_t number) {
    std::string result(8, ' ');
    result[0] = ((number >> 22) & 31) + 'A';
    result[2] = ((number >> 18) & 15) + '0';
    result[3] = ((number >> 14) & 15) + '0';
    result[4] = ((number >> 10) & 15) + '0';
    result[6] = ((number >> 5) & 31) + 'A';
    result[7] = (number & 31) + 'A';
    return std::string(result);
}
```

Функции сортировки

```
void counting_sort(Vector<Car>& cars, Vector<Car>& res, int R, int shift) {
    const int k = 256;
    const uint32_t mask = 255u << (R * shift);

    Vector<int> counter(k, 0);

    for (size_t i = 0; i < cars.size(); ++i) {
        ++counter[(cars[i].key & mask) >> (R * shift)];
    }

    for (int i = 1; i < k; ++i) {
        counter[i] += counter[i - 1];
```

```
    }

    for (int i = cars.size() - 1; i >= 0; --i) {
        int pos = --counter[(cars[i].key & mask) >> (R * shift)];
        res[pos] = std::move(cars[i]);
    }
}

void radix_sort(Vector<Car>& cars) {
    Vector<Car> res(cars.size());

    for (int shift = 0; shift < 4; ++shift) {
        counting_sort(cars, res, 8, shift);
        cars.swap(res);
    }
}
```

Основная функция «main»

```
int main() {
    std::string inputLine;

    Vector<Car> cars;

    while(std::getline(std::cin, inputLine)) {
        if (inputLine.empty()) {
            continue;
        }
        size_t delimiterPos = inputLine.find('\t');
        if (delimiterPos == std::string::npos) continue;
        cars.pushBack(Car(encodeNumber(inputLine.substr(0, delimiterPos)),
                          inputLine.substr(delimiterPos + 1)));
    }
    cars.shrinkToFit();

    radix_sort(cars);

    for (const auto& car : cars) {
        std::cout << decodeNumber(car.key) << '\t' << car.id << '\n';
    }

    return 0;
}
```

Дневник отладки

В качестве первой попытки решить данную задачу, было придумано простое решение, путём обычного сравнения символов, которое было реализовано обычным проходом по строке, состоящей из 8 символов.

Такая реализация уложилась в ограничения времени и памяти (2с и 150 мб, было использовано 145 мб и максимальное время составляло 1.8с), но мной было принято решение оптимизировать решение. В вышеописанном алгоритме представлено описание представление строки номера автомобиля в 32 битное число. В процессе выполнения, были изучены простейшие битовые операции, более глубоко изучены операции битовых сдвигов.

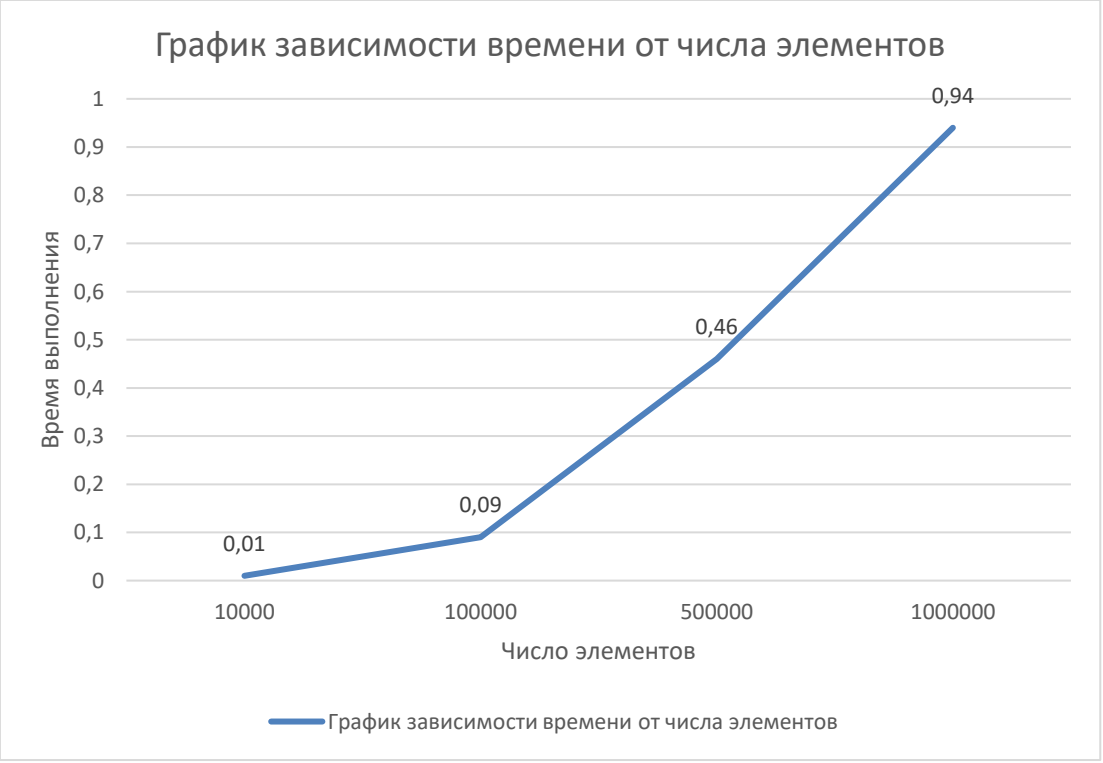
Некоторые ошибки, которые были получены в результате тестирования программы, заключались в основном в невнимательности, и были быстро исправлены

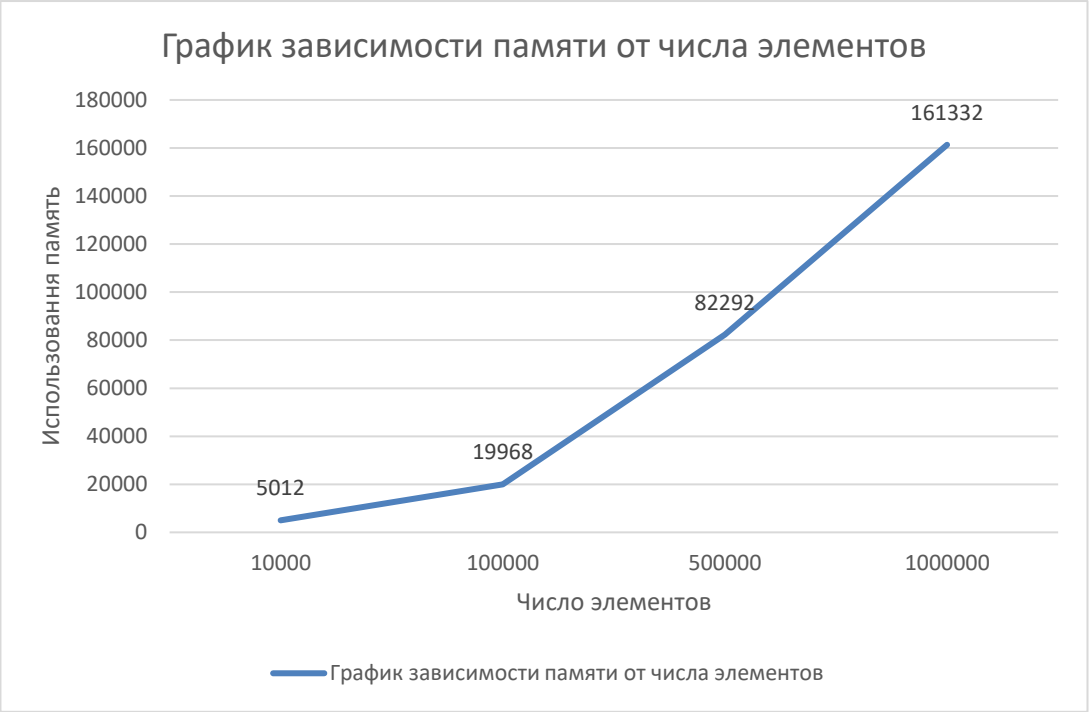
Тест производительности

Для проверки скорости работы программы, проведём замер, постепенно увеличивая количество данных и проведём замеры скорости выполнения и потребления памяти:

Замер проводился на WSL (Windows Subsystem for Linux)

Число элементов (N)	Время (с)	Память (КБ)
10000	0.01	5012
100000	0.09	19968
500000	0.46	82292
1000000	0.94	161332





Также ниже приведены тесты с проверяющей решение платформы:

№	Вердикт	Ресурсы
1	ok	2ms / 660.00Kb
2	ok	2ms / 668.00Kb
3	ok	2ms / 536.00Kb
4	ok	2ms / 660.00Kb
5	ok	2ms / 532.00Kb
6	ok	2ms / 660.00Kb
7	ok	2ms / 536.00Kb
8	ok	2ms / 540.00Kb
9	ok	2ms / 668.00Kb
10	ok	3ms / 668.00Kb
11	ok	10ms / 1008.00Kb
12	ok	89ms / 8.89Mb
13	ok	0.943s / 89.70Mb

Можем заметить, что тесты и графики демонстрируют линейную сложность, как памяти, так и времени:

Сложность по времени – $O(\frac{b}{R} * (n + 2^R)$, где в нашем случае d = 32; R = 8, откуда следует:

$$O(4 * (n + 256))$$

Сложность по памяти: $O(n + k)$

Недочёты

На мой взгляд недочётом данного алгоритма является способ кодирования данных. В случае изменения данных, придётся придумывать новый способ кодирования и декодирования данных.

Выводы

В результате выполнения данной лабораторной работы, был реализован алгоритм поразрядной сортировки, который использует битовые операции для ускорения выполнения программы. В качестве типовых задач можно рассмотреть реальные данные номеров машин, которые используются в базах данных ГБДД и требуют сортировки. Конечная сложность программы, как и предполагало условие – линейная. Анализ производительности показал, что по мере возрастания числа данных, время выполнения линейно увеличивается. Возникшие проблемы при решении, были незначительными и заключались в небольших невнимательных ошибках. Основной сложностью написания алгоритма стал способ сопоставления данных номера машины и числа. Конечный результат программы демонстрирует корректную работу, проходя все тесты платформы, а сложность алгоритма удовлетворяет ожидаемой.