

Московский авиационный институт
(национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Дискретный анализ»

Студент: М. Г. Григорьев
Преподаватель: С. А. Михайлова
Группа: М8О-201БВ
Дата:
Оценка:
Подпись:

Москва, 2026

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: Сортировка подсчётом.

Вариант ключа: Числа от 0 до 65535.

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.

1 Описание

Требуется реализовать сортировку подсчётом для последовательности пар «ключ-значение». Ключи лежат в фиксированном диапазоне от 0 до 65535, поэтому для каждого возможного ключа можно хранить количество его вхождений в отдельной ячейке массива.

Основная идея сортировки подсчётом состоит в том, что сначала подсчитывается число элементов каждого ключа, а затем по префиксным суммам вычисляются итоговые позиции элементов в отсортированной последовательности [1].

В данной работе алгоритм реализован в стабильном варианте и выполняется в три этапа:

1. Подсчёт частот ключей в массиве счётчиков размера 65536.
2. Преобразование массива частот в массив префиксных сумм.
3. Размещение элементов во временный массив при проходе исходного массива справа налево.

Проход справа налево нужен для сохранения исходного порядка элементов с одинаковыми ключами (стабильность сортировки).

Асимптотика реализации:

- время: $O(n + k)$, где n — число пар, $k = 65536$;
- дополнительная память: $O(n + k)$.

2 Исходный код

Программа состоит из функции сортировки `CountingSort` и функции `main`.

Для хранения элементов используются псевдонимы типов:

- `Key` — тип ключа (`int`);
- `Value` — тип значения (`std::string`);
- `Object` — пара `std::pair<Key, Value>`.

Также задаётся константа `KEY_RANGE = 65536`, соответствующая числу всех возможных ключей.

Логика функции `CountingSort(std::vector<Object>& objects)`:

1. Если входной массив пуст, функция сразу завершает работу.
2. Создаётся и инициализируется нулями массив счётчиков `std::array<Key, KEY_RANGE> counting_array{}`.
3. В цикле по входным данным для каждого элемента увеличивается счётчик `counting_array[obj.first]`.
4. Далее строятся префиксные суммы: каждая ячейка хранит количество элементов с ключами не больше текущего.
5. Создаётся временный массив `sorted_array` размером `objects.size()`.
6. Выполняется проход по `objects` с конца к началу: для элемента вычисляется итоговая позиция `position = --counting_array[obj.first]`, после чего элемент помещается в `sorted_array[position]`.
7. После заполнения временного массива содержимое `objects` и `sorted_array` меняется местами через `swap`.

Логика функции `main`:

1. Считываются пары `key value` из стандартного потока до конца файла.
2. Каждая считанная пара добавляется в вектор `objects`.
3. После завершения ввода вызывается `CountingSort(objects)`.
4. Отсортированные пары выводятся в формате `key\tvalue` (ключ, символ табуляции, значение).

Полный текст программы:

Листинг 1: Реализация лабораторной работы

```

1 | #include <iostream>
2 | #include <vector>
3 | #include <utility>
4 | #include <string>
5 | #include <array>
6 |
7 | using Key = int;
8 | using Value = std::string;
9 | using Object = std::pair<Key, Value>;
10 |
11 | const Key KEY_RANGE = 65536;
12 |

```

```

13 void CountingSort(std::vector<Object>& objects) {
14     if (objects.empty())
15         return;
16
17     std::array<Key, KEY_RANGE> counting_array{};
18
19     for (const auto& obj : objects)
20         ++counting_array[obj.first];
21
22     for (int i = 1; i < KEY_RANGE; ++i)
23         counting_array[i] += counting_array[i - 1];
24
25     std::vector<Object> sorted_array(objects.size());
26
27     for (int i = objects.size(); i > 0; --i) {
28         const auto& obj = objects[i - 1];
29         int position = --counting_array[obj.first];
30         sorted_array[position] = obj;
31     }
32
33     objects.swap(sorted_array);
34 }
35
36 int main() {
37     std::vector<Object> objects;
38
39     Key key;
40     Value value;
41     while (std::cin >> key >> value)
42         objects.push_back({key, value});
43
44     CountingSort(objects);
45
46     for (const auto& obj : objects)
47         std::cout << obj.first << '\t' << obj.second << '\n';
48
49     return 0;
50 }

```

3 Консоль

```
$ g++ main.cpp -o lab1.out
$ cat input.txt
0      n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naatt
65535  n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naat
0      n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naa
65535  n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3na
$ ./lab1 <input.txt
0      n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naatt
0      n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naa
65535  n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naat
65535  n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3na
```

4 Тест производительности

Для оценки времени работы были сгенерированы входные данные разных размеров. Так как диапазон ключей фиксирован, сортировка подсчётом показывает линейный рост времени по числу элементов и не зависит от распределения ключей.

Теоретически для данного варианта $k = 65536$, поэтому:

$$T(n) = O(n + 65536) = O(n).$$

На практике это означает, что при росте входного массива в 10 раз время работы увеличивается примерно в 10 раз (с учётом постоянной составляющей и накладных расходов системы).

Память под массив счётчиков фиксирована и составляет $65536 \cdot \text{sizeof}(int)$ байт, дополнительно используется выходной массив размера n .

5 Выводы

В этой лабораторной работе я реализовал стабильную сортировку подсчётом для пар «ключ-значение» и на практике разобрал полный цикл алгоритма: подсчёт частот, построение префиксных сумм и распределение элементов по итоговым позициям.

Главный технический вывод: линейная оценка $O(n + k)$ достигается только при заранее известном и ограниченном диапазоне ключей. В нашем варианте это выполнимо ($k = 65536$), поэтому алгоритм работает предсказуемо по времени даже на больших входах.

Также стало понятно, как обеспечивается стабильность сортировки: исходный массив нужно обходить справа налево, иначе относительный порядок элементов с одинаковыми ключами нарушается. Отдельно были отработаны требования к формату ввода-вывода для автоматической проверки.

Таким образом, сортировка подсчётом является хорошим практическим выбором для задач с целочисленным ключом из небольшого диапазона; при слишком большом диапазоне ключей затраты памяти становятся её основным ограничением.

Список литературы

- [1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание*. — М.: Вильямс, 2007.
- [2] *std::vector* — *cppreference.com*.
URL: <https://en.cppreference.com/w/cpp/container/vector> (дата обращения: 23.02.2026).
- [3] *std::array* — *cppreference.com*.
URL: <https://en.cppreference.com/w/cpp/container/array> (дата обращения: 23.02.2026).