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

```
#include <iostream>
#include <vector>
#include <utility>
#include <string>
#include <array>

using Key = int;
using Value = std::string;
using Object = std::pair<Key, Value>;

const Key KEY_RANGE = 65536;

```

```cpp
void CountingSort(std::vector<Object>& objects) {
    if (objects.empty())
        return;

    std::array<Key, KEY_RANGE> counting_array{};

    for (const auto& obj : objects)
        ++counting_array[obj.first];

    for (int i = 1; i < KEY_RANGE; ++i)
        counting_array[i] += counting_array[i - 1];

    std::vector<Object> sorted_array(objects.size());

    for (int i = objects.size(); i > 0; --i) {
        const auto& obj = objects[i - 1];
        int position = --counting_array[obj.first];
        sorted_array[position] = obj;
    }

    objects.swap(sorted_array);
}

int main() {
    std::vector<Object> objects;

    Key key;
    Value value;
    while (std::cin >> key >> value)
        objects.emplace_back(key, value);

    CountingSort(objects);

    for (const auto& obj : objects)
        std::cout << obj.first << '\t' << obj.second << '\n';

    return 0;
}
```

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

В этом разделе сравнивается время работы собственной реализации сортировки подсчётом и стандартной стабильной сортировки C++ (`std::stable_sort`). Тестовые данные генерируются скриптом `generator.py`: ключи равномерно распределены в диапазоне $[0, 65535]$, значение – строка длиной от 1 до 64 знаков. Для каждого размера входа выполнено 5 независимых запусков, в таблице приведены медианные значения времени (в микросекундах) и отношение времени `STL/Counting`.

Команда запуска:

```
$ python3 generator.py >b.txt
$ ./benchmark.out <b.txt
```

Сводная таблица результатов:

n	Counting, `us`	STL, `us`	STL/Counting
100	39	3	0.08×
10 000	384	442	1.15×
100 000	3430	7092	2.07×
1 000 000	49307	111407	2.26×
10 000 000	528712	1907158	3.61×

На малом объёме данных ($n = 100$) сортировка подсчётом ожидаемо проигрывает: затраты на инициализацию массива счётчиков размера $k = 65536$ уже есть, а полезной работы ещё мало. Начиная с $n = 10^4$ ситуация меняется, и сортировка подсчётом становится быстрее. В диапазоне от 10^5 до 10^7 преимущество растёт от 2.07× до 3.61×, что согласуется с теорией: при фиксированном диапазоне ключей алгоритм работает за $O(n + k)$, тогда как `std::stable_sort` требует $O(n \log n)$ сравнений.

5 Выводы

В лабораторной работе была реализована стабильная сортировка подсчётом для пар «ключ-значение» и проведено сравнение со `std::stable_sort` на серии входов разного размера.

Практический результат оказался ожидаемым: на очень малом входе сортировка подсчётом проигрывает из-за постоянной стоимости обработки диапазона ключей ($k = 65536$), но начиная с 10^4 элементов стабильно выигрывает по времени. На больших объёмах данных преимущество становится заметным и в эксперименте достигает нескольких раз.

Стабильность в counting sort обеспечивается обратным проходом по исходному массиву; линейная сложность $O(n + k)$ даёт выигрыш только при ограниченном диапазоне ключей; сравнение производительности корректно делать по серии прогонов и медиане, поскольку абсолютные времена чувствительны к платформе, нагрузке и деталям реализации бенчмарка.

Итог: сортировка подсчётом — эффективный инструмент для задач с целочисленным ключом из небольшого фиксированного диапазона, если допустимы дополнительные затраты памяти под счётчики и временный массив результата.

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).