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: Волков А.А.
Преподаватель: Михайлова С.А.
Группа: М8О-201БВ-24

Дата: 27 февраля 2026 г.
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 переменной длины (до 2048 символов).`

1 Описание

Требуется реализовать сортировку подсчётом для последовательности пар «ключ-значение». Ключи лежат в диапазоне от 0 до 65535 (задано вариантом), однако входные данные могут содержать не все ключи из этого диапазона. В реализации используется адаптивный подход: сначала определяются минимальный и максимальный ключи во входных данных, что позволяет сократить используемую память под массив счётчиков.

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. Поиск минимального и максимального ключа во входном массиве.
2. Подсчёт частот ключей в массиве счётчиков размера $(\max\_key - \min\_key + 1)$.
3. Преобразование массива частот в массив префиксных сумм.
4. Размещение элементов во временный массив при проходе исходного массива справа налево для сохранения стабильности.

Асимптотика реализации:

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

2 Исходный код

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

Логика функции `couting_sort(std::vector<Object> data)`:

1. Если входной массив пуст, функция возвращает пустой результат.
2. Определяются `min_key` и `max_key` проходом по массиву.
3. Создаётся и инициализируется нулями вектор счётчиков размера `range_size = max_key - min_key + 1`.
4. В цикле по входным данным для каждого элемента увеличивается счётчик `count[data[i].first - min_key]`.

5. Далее строятся префиксные суммы.

6. Создаётся выходной вектор `result` размером `data.size()`.

7. Выполняется проход по `data` с конца к началу: для элемента вычисляется индекс `index = data[i].first - min_key`, итоговая позиция `pos = count[index] - 1`, после чего элемент помещается в `result[pos]`, а счётчик уменьшается.

8. Функция возвращает отсортированный вектор `result`.

Логика функции `main`:

1. Для ускорения ввода-вывода отключается синхронизация с C-потоками и `cin` отвязывается от `cout`: `std::ios_base::sync_with_stdio(false); std::cin.tie(NULL);`

2. Считываются пары `key value` из стандартного потока до конца файла.

3. Каждая считанная пара добавляется в вектор `data`.

4. После завершения ввода вызывается `couting_sort(data)`.

5. Отсортированные пары выводятся в формате `key\tvalue` (ключ, символ табуляции, значение). Использование '`\n`' вместо `std::endl` позволяет избежать лишнего сброса буфера вывода, что критично для производительности.

Полный текст программы:

Листинг 1: Реализация лабораторной работы

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <memory>
#include <string>
#include <utility>

std::vector<std::pair<int, std::string>> counting_sort(const std::vector<std::pair<int
    , std::string>> &data) {
    if (data.empty()) return {};

    int max_key = data[0].first;
    int min_key = data[0].first;

    for (const auto& p : data) {
        if (p.first > max_key) max_key = p.first;
        if (p.first < min_key) min_key = p.first;
    }
```

```
18
19      int range_size = max_key - min_key + 1;
20      std::vector<int> count(range_size, 0);
21      for (const auto& p : data) {
22          int index = p.first - min_key;
23          ++count[index];
24      }
25
26      for (size_t i = 1; i < count.size(); ++i) {
27          count[i] += count[i-1];
28      }
29
30      std::vector<std::pair<int, std::string>> result(data.size());
31      for (int i = data.size() - 1; i >= 0; --i) {
32          int index = data[i].first - min_key;
33          int pos = count[index] - 1;
34          result[pos] = data[i];
35          --count[index];
36      }
37      return result;
38  }
39
40
41  #ifndef BENCHMARK
42  int main() {
43      std::ios_base::sync_with_stdio(false);
44      std::cin.tie(NULL);
45      std::vector<std::pair<int, std::string>> data;
46      int key;
47      std::string value;
48
49      while(std::cin >> key >> value) {
50          data.emplace_back(key,value);
51      }
52
53      auto sorted_data = counting_sort(data);
54      for (const auto& p : sorted_data) {
55          std::cout << p.first << '\t' << p.second << '\n';
56      }
57      return 0;
58  }
59  #endif
```

3 Консоль

В данном примере приведена компиляция и запуск программы на операционной системе Arch Linux.

```
$ g++ main.cpp -o lab1.out -std=c++20 -O2
$ cat input.txt
0    xGfxrxGGxrxMMMMfrrrG
65535    xGfxrxGGxrxMMMMfrrr
0    xGfxrxGGxrxMMMMfrr
65535    xGfxrxGGxrxMMMMfr
$ ./lab1.out <input.txt
0    xGfxrxGGxrxMMMMfrrrG
0    xGfxrxGGxrxMMMMfrr
65535    xGfxrxGGxrxMMMMfrrr
65535    xGfxrxGGxrxMMMMfr
```

4 Тест производительности

Для оценки эффективности реализованного алгоритма было проведено сравнение с универсальными алгоритмами сортировки из стандартной библиотеки C++: `std::sort` (нестабильная, на основе Introsort) и `std::stable_sort` (стабильная, на основе MergeSort).

Эксперимент проводился на данных с ключами в диапазоне $[0, 65535]$. Приблизительные результаты замеров времени (в секундах) приведены в таблице 1.

N элементов	**Counting Sort**	**std::sort**	**std::stable_sort**	**Ускорение***
1 000	0.00022 с	0.00008 с	0.00010 с	0.47x
10 000	0.00039 с	0.00090 с	0.00113 с	2.91x
100 000	0.00326 с	0.00983 с	0.01627 с	4.99x
1 000 000	0.04653 с	0.10953 с	0.18987 с	4.08x
3 000 000	0.13439 с	0.24740 с	0.47246 с	3.51x

Таблица 1: Сравнение времени работы (*относительно std::stable_sort)

Анализ результатов

1. **Влияние диапазона ключей.** При малом количестве элементов ($N = 1000$) сортировка подсчётом демонстрирует худшее время. Это обусловлено тем, что сложность алгоритма составляет $O(N+K)$, где $K = 65536$. Накладные расходы на инициализацию и обход массива счётчиков превосходят время работы $O(N \log N)$ на малых N.

2. **Преимущество линейного времени.** Начиная с $N = 10\,000$, сортировка подсчётом начинает опережать стандартные алгоритмы. На больших объёмах данных ($N = 3 \cdot 10^6$) преимущество над стабильной сортировкой `std::stable_sort` достигает 3.5 раз, а над быстрой `std::sort` — более 2 раз.

3. **Стабильность.** Так как по заданию необходимо сохранять относительный порядок элементов с равными ключами, наиболее корректным является сравнение с `std::stable_sort`. В этом случае реализованный алгоритм оказывается эффективнее на всех наборах данных, где N сопоставимо или больше K.

5 Выводы

В результате проделанной работы можно выделить несколько ключевых моментов.

Об алгоритме. Реализация сортировки подсчётом подтвердила её применимость для задач с небольшим диапазоном ключей: при $k = 65536$ время обработки растёт линейно с увеличением объёма данных. Использование предварительного поиска минимума и максимума позволяет гибко управлять памятью, выделяя ровно столько счётчиков, сколько действительно необходимо для текущего набора ключей.

О стабильности. На практике убедился, что стабильность сортировки достигается только при проходе исходного массива справа налево на этапе расстановки элементов. Любое отклонение от этого порядка приводит к потере исходной последовательности записей с равными ключами.

О производительности. Наибольший практический интерес представляет влияние ввода-вывода на итоговое время работы. Первая посылка программы (1.097 с) упёрлась в тайм-лимит исключительно из-за использования `std::endl`, который принудительно сбрасывает буфер после каждой строки. После замены на '`\n`' и оптимизации потоков время упало до 463 мс — это более чем двукратное ускорение без изменения самой сортировки.

Сравнение с универсальными алгоритмами. На основе проведённого бенчмарка можно сделать вывод, что для специфических условий (ограниченный диапазон ключей) сортировка подсчётом значительно превосходит универсальные алгоритмы сортировки, основанные на сравнениях (как `std::sort` или `std::stable_sort`). Однако она требует выделения существенного объёма дополнительной памяти $O(N + K)$, что при очень больших диапазонах ключей может быть неприемлемым.

Итог. Работа показала, что понимание внутренней структуры данных и ограничений предметной области позволяет выбрать специализированный алгоритм, который на порядок эффективнее стандартных решений.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — М.: Вильямс, 2007.