Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Дискретный анализ»

Студент: В. И. Пупкин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

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 999999.`

Вариант значения: `Числа от 0 до` 2^{64}`-1.`

1 Исходный код

Программа реализует сортировку подсчетом для пар «ключ-значение», где ключ — 32-битное беззнаковое целое число, а значение — 64-битное беззнаковое целое число.

Основная идея алгоритма: для каждого ключа подсчитывается количество элементов с таким ключом, затем вычисляются позиции элементов в отсортированном массиве путем накопления сумм счетчиков. Элементы размещаются в выходном массиве согласно вычисленным позициям.

Т.к код не помещается на одну страницу, приведена таблица с описанием функций и классов:

contest.hpp	
`template <typename T> class TVector`	Класс динамического массива с методами `PushBack`, `Reserve`, `Size`, `Empty`, операторами индексирования и итераторами
`TVector(size_t count, const T &value)`	Конструктор, создающий вектор из `count` элементов со значением `value`
`void Reserve(size_t newCapacity)`	Резервирование памяти для `newCapacity` элементов
`void PushBack(const T &value)`	Добавление элемента в конец вектора
`void CountingSort(TVector<TElement> &arr)`	Функция сортировки подсчетом. Находит минимум и максимум ключей, создает массив счетчиков, подсчитывает элементы, вычисляет позиции и размещает элементы в отсортированном порядке
`bool CompareById(const TElement &left, const TElement &right)`	Функция сравнения элементов по полю `id`
contest_main.cpp	
`int main()`	Считывает пары ключ-значение из стандартного ввода с помощью `scanf`, вызывает сортировку, выводит отсортированные данные с помощью `printf`

Структура для хранения элементов:

```
1 struct TElement {
2     uint32_t id;
3     uint64_t value;
4 };
```

Объявление класса TVector (без реализации методов):

```cpp
template <typename T>
class TVector {
  public:
    TVector() = default;
    TVector(size_t count, const T &value);
    TVector(const TVector &other);
    TVector(TVector &&other) noexcept;
    TVector &operator=(const TVector &other);
    TVector &operator=(TVector &&other) noexcept;
    ~TVector();

    void PushBack(const T &value);
    void PushBack(T &&value);
    void Reserve(size_t newCapacity);
    size_t Size() const;
    size_t Capacity() const;
    bool Empty() const;
    T &operator[](size_t index);
    const T &operator[](size_t index) const;
    T *Begin();
    const T *Begin() const;
    T *End();
    const T *End() const;
    void Swap(TVector &other) noexcept;

  private:
    std::allocator<T> allocator_;
    T *data_ = nullptr;
    size_t size_ = 0;
    size_t capacity_ = 0;
};
```

2 Консоль

```
$ make
g++ .obj/contest_main.o -o solution

$ ./solution <cases/5.txt
019383 3546839058697536693
205057 16563277856680048351
219581 8725518293060573749
526830 3189070866999033975
690718 11796781922971975819

$ cat cases/5.txt
690718 11796781922971975819
219581 8725518293060573749
205057 16563277856680048351
526830 3189070866999033975
019383 3546839058697536693
```

3 Тест производительности

Тест производительности сравнивает реализованную сортировку подсчетом с алгоритмом `std::stable_sort` из стандартной библиотеки C++.

1 Методика тестирования

Для тестирования использовались данные со следующими характеристиками:

- Количество элементов: от 5 до 1 000 000
- Диапазон ключей: от 000000 до 999999 (6 цифр)
- Каждый элемент содержит 32-битный ключ (`id`) и 64-битное значение (`value`)
- Ключи генерируются случайным образом из равномерного распределения

Замеры времени производились с помощью `std::chrono::system_clock` с точностью до микросекунд. Для каждого размера входных данных создавался отдельный тестовый файл с помощью генератора на Python.

2 Результаты измерений

N	Counting Sort (мкс)	STL Sort (мкс)	Ускорение
5	6 013	1	0.00x
50	8 812	4	0.00x
100	8 623	9	0.00x
1 000	9 554	88	0.01x
2 000	9 211	248	0.03x
3 000	9 307	288	0.03x
5 000	9 369	572	0.06x
10 000	9 268	1 025	0.11x
20 000	9 853	2 615	0.27x
25 000	9 894	3 545	0.36x
50 000	10 431	6 250	0.60x
100 000	12 149	15 859	1.31x
1 000 000	37 944	165 626	4.37x

Таблица 2: Результаты сравнения производительности алгоритмов сортировки

3 Анализ результатов

Из таблицы 2 и графика (рис. 1) видно, что производительность сортировки подсчетом существенно зависит от размера входных данных:

- **Малые размеры (n $<$ 10 000)**: Сортировка подсчетом проигрывает `std::stable_sort`. Это объясняется накладными расходами на создание вспомогательных массивов (массив счетчиков размером 1 000 000 элементов) и поиск минимального/-максимального элемента.
- **Средние размеры (10 000 $\leq$ n $<$ 100 000)**: Алгоритмы показывают сопоставимую производительность. При $n = 100\,000$ сортировка подсчетом начинает обгонять STL сортировку (ускорение 1.31x).
- **Большие размеры (n $\geq$ 100 000)**: Сортировка подсчетом значительно быстрее. На миллионе элементов ускорение составляет 4.37x.

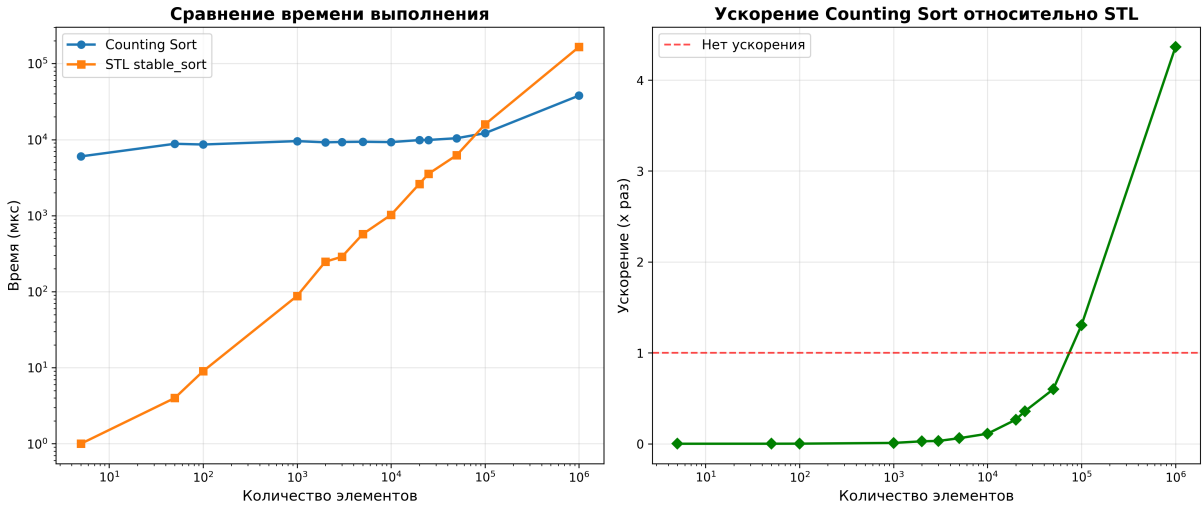

Рис. 1: Сравнение производительности алгоритмов сортировки

4 Выводы

Асимптотическая сложность алгоритмов:

- Сортировка подсчетом: $O(n + k)$, где $k = 1\,000\,000$ — диапазон ключей
- `std::stable_sort`: $O(n \log n)$

При малых n доминирует слагаемое k, что делает сортировку подсчетом неэффективной. Однако с ростом n линейная зависимость от количества элементов дает преимущество перед логарифмической сложностью STL алгоритма.

Точка равновесия находится примерно при $n \approx 100\,000$ элементов. При больших объемах данных сортировка подсчетом демонстрирует существенное преимущество, что подтверждает теоретические оценки сложности.

4 Выводы

Наверное самое важное это то, что я сходил на лекции (и немного почитал книжек) и понял, как работают более "продвинутые"сортировки. Я до конца не хотел их учить, потому что их не спрашивают на собесах - но это реально интересно, а никак времени на это времени. Спасибо ДискрАну за то, что я поизучал эти алгоритмы :) На cpp нас +- научили что-то несложное писать на ООП, поэтому про cpp супер нового не узнал. Вроде только не использовал, но знал про template-ы, тут заиспользовал их сразу в "боевых условиях на лабе

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 16.12.2013).

[3] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008