Лабораторная работа №1 по курсу дискретного анализа: строковые алгоритмы.

Выполнил: студент группы М8О-307Б-23
Дубровина Софья Андреевна

Условие:

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:

Сортировка подсчётом.

Тип ключа: почтовые индексы.

Тип значения: числа от 0 до $2^{64}-1$

Метод решения:

Для сортировки пар «ключ-значение» используется алгоритм сортировки подсчётом, который работает за линейное время при ограниченном диапазоне возможных ключей (в данном случае — ровно 1 000 000 вариантов индексов).

Общее описание алгоритма:

1. **Чтение данных.**
   Входные данные считываются из стандартного ввода. Каждая строка содержит ключ (почтовый индекс из 6 цифр), символ табуляции и значение (целое число uint64_t).

2. **Подсчёт частот.**
   Для каждого ключа подсчитывается количество его вхождений в массив count.

3. **Вычисление позиций.**
   На основе массива count формируется префиксная сумма, указывающая стартовую позицию для каждого ключа в отсортированном массиве.

4. **Сортировка.**
   Пары размещаются в новый массив sorted в порядке возрастания ключей. Используется проход с конца входного массива, что обеспечивает стабильность сортировки.

5. **Вывод.**
   Отсортированные пары выводятся в формате «ключ[TAB]значение». Почтовый индекс печатается с ведущими нулями.

Архитектура программы:

Программа состоит из:

- функции main — управляет чтением, сортировкой и выводом;
- вспомогательных функций:
  - ParseKey — переводит строковый индекс из 6 цифр в число;
  - PrintKey — печатает число обратно в виде шестизначной строки.

Основные структуры и данные:

- struct TRec — содержит два поля:
  - uint32_t Key (почтовый индекс в диапазоне 0..999999),

- uint64_t Value (значение).

- uint32_t count[1000000] — массив для подсчёта количества пар с каждым ключом.

- Динамические массивы для хранения исходных и отсортированных пар.

Сложность алгоритма

- **Временная сложность:** O(n + k), где
  - n — количество пар,
  - k = 1 000 000 — количество возможных ключей (почтовых индексов).
- **Пространственная сложность:** O(n + k), где
  - O(n) — память для исходного и отсортированного массивов,
  - O(k) — память для массива частот.

Описание программы

Программа реализована в одном файле main.cpp.

Основные типы данных

1. struct TRec — структура для хранения пары «ключ–значение»:
   - uint32_t Key — почтовый индекс (целое число от 0 до 999999);
   - uint64_t Value — значение (целое число от 0 до 2^64 − 1).

2. uint32_t* count — динамический массив счётчиков, размером KEY_RANGE = 1 000 000. Хранит количество вхождений каждого ключа.

3. TRec* records — динамический массив исходных записей, считываемых из ввода. Используется стратегия увеличения ёмкости в 2 раза при переполнении (динамическое расширение, как у вектора).

4. TRec* sorted — динамический массив для хранения отсортированных пар.

Основные функции

- ParseKey(const char* s)
    - Назначение: преобразует строковый почтовый индекс из 6 символов ("000123") в число (123).
    - Вход: строка из 6 цифр.
    - Выход: uint32_t — числовое значение индекса.
    - Временная сложность: O(6) → константа.
- PrintKey(uint32_t key)
    - Назначение: преобразует число key обратно в строку длиной 6 символов с ведущими нулями.
    - Вход: целое число от 0 до 999999.
    - Выход: печать строки в cout.
    - Временная сложность: O(6) → константа.

- main()
  Управляет всей программой:
  - Настройка быстрого ввода/вывода (sync_with_stdio(false), cin.tie(nullptr)).
  - Выделение памяти под массив count.
  - Чтение входных данных:
    - ключ преобразуется функцией ParseKey,
    - значение сохраняется в массив records,
    - параллельно увеличивается счётчик count[key].
  - Преобразование массива count в массив префиксных сумм.
  - Формирование массива sorted при помощи стабильной сортировки.
  - Вывод отсортированных данных (ключ печатается функцией PrintKey, затем выводится значение).
  - Очистка памяти (delete[]).

Логика работы программы

1. Входные данные читаются из стандартного ввода, сохраняются в динамический массив.
2. Алгоритм подсчёта фиксирует, сколько раз встречается каждый почтовый индекс.
3. На основе подсчитанных частот формируется карта позиций для ключей.

4. Элементы раскладываются по массиву sorted в правильном порядке (с обеспечением стабильности).

5. Результат выводится в стандартный вывод в том же формате, что и вход.

Тест производительности

Производительность программы измерялась при помощи команды:

time ./lab1 < input.txt > output.txt

Входные файлы input.txt были сгенерированы с помощью утилиты gen, запускаемой командой:

./gen <кол-во строк> > input.txt

Генератор создаёт случайные пары, где ключи — почтовые индексы (строки из 6 цифр, диапазон $0 \leq key \leq 999999$), а значения — случайные числа типа uint64_t в диапазоне $0 \leq value \leq 2^{64}-1$.

Методика тестирования:

- Измерялось время real (общее время выполнения, включая ввод-вывод и сортировку).

- Тесты проводились для разного количества пар (10 000, 50 000, 100 000).

- Для каждого набора данных выполнялось несколько запусков, брались средние значения.

Ожидаемые результаты:

- Сложность O(n + k) предполагает линейный рост времени с увеличением числа пар n.

- На малых объёмах данных существенную часть времени может занимать ввод-вывод.

- На больших объёмах время выполнения должно быть пропорционально n.

Результаты:

Количество строк	Время работы (сек)
10 000	0.042
50 000	0.043
100 000	0.084

Анализ:

- Время работы растёт линейно с увеличением числа строк, что подтверждает сложность O(n + k).

- При переходе от 10 000 к 50 000 строк рост времени минимален, что объясняется доминированием операций ввода-вывода над самим процессом сортировки.

- При дальнейшем увеличении объёма данных (до 100 000 строк) время увеличилось примерно в 2 раза, что соответствует линейной зависимости от n.

- Таким образом, сортировка подсчётом демонстрирует ожидаемое поведение и выигрывает по скорости у универсальных алгоритмов с O(n log n) при ограниченном диапазоне ключей.

Недочёты

1. **Ограниченная проверка входных данных.**
   Программа предполагает, что все строки входного файла корректны

(ключ — ровно 6 цифр, после которых идёт символ табуляции и целое число). Если формат нарушен, программа может работать некорректно.

2. **Дополнительные затраты памяти.**
   Для обеспечения стабильности сортировки используется отдельный массив sorted, что удваивает объём памяти для хранения пар. Однако это стандартное свойство реализации сортировки подсчётом.

3. **Слабая масштабируемость по ключам.**
   Алгоритм рассчитан на фиксированный диапазон ключей (0…999999). Если бы требовалось поддерживать больший диапазон индексов, массив count становился бы слишком большим.

Выводы

Алгоритм сортировки подсчётом показал высокую эффективность для задачи упорядочивания пар «почтовый индекс — значение»:

- **Временная сложность:** O(n + k), где n — количество пар, k = 1 000 000 — диапазон почтовых индексов.

- **Пространственная сложность:** O(n + k) за счёт массивов записей и массива счётчиков.

- **Скорость работы:** тестирование подтвердило линейный рост времени выполнения при увеличении числа элементов.

Область применения алгоритма:

- сортировка идентификаторов и кодов из фиксированного диапазона (почтовые индексы, коды товаров, ID-пользователей и т.п.);

- обработка больших массивов данных в реальном времени, где важно линейное время работы;

- подготовка данных для дальнейшей обработки в других структурах (хэш-таблицы, базы данных).

Сложность программирования относительно невысока, но потребовала:

- корректного управления динамической памятью;
- аккуратного вывода индексов с ведущими нулями;
- обеспечения стабильности сортировки.

Таким образом, в работе была реализована сортировка подсчётом для пар «почтовый индекс — число». Алгоритм продемонстрировал линейную временную сложность O(n + k), стабильность сортировки и укладывание в ограничения по времени и памяти. Тестирование подтвердило, что при увеличении объёма данных время работы растёт пропорционально числу элементов, что соответствует теоретическому анализу. Сортировка подсчётом оказалась удобной и эффективной для обработки данных с целочисленными ключами фиксированного диапазона и может применяться, например, для сортировки идентификаторов, кодов или индексов.