Лабораторная работа № 1 по курсу дискретного анализа: сортировка за линейное время

Выполнил студент группы 08-207 МАИ *Дружинин Данила.*

Условие

1. Требуется разработать программу, осуществляющую ввод пар «ключ-значение», их упорядочивание по возрастанию ключа указанным алгоритмом сортировки за линейное время и вывод отсортированной последовательности.

2. Задача AA. $9 - 3$

   Вариант задания определяется типом ключа (и соответствующим ему методом сортировки) и типом значения: Карманная сортировка.

   Тип ключа: вещественные числа в промежутке $[-100, 100]$.

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

Метод решения

Я реализовал алгоритм карманной сортировки (Bucket Sort), обеспечивающий линейное время работы при равномерном распределении входных данных.

Суть данного алгоритма заключается в разбиении исходного диапазона значений ключа на фиксированное количество интервалов. Каждый элемент входной последовательности распределяется в соответствующий карман в зависимости от значения ключа. После распределения элементы внутри каждого кармана упорядочиваются, а затем карманы последовательно объединяются, формируя итоговую отсортированную последовательность.

В данной задаче диапазон ключей ограничен интервалом $[-100, 100]$. Этот интервал разбивается на заранее заданное количество карманов одинаковой ширины. Индекс кармана вычисляется по формуле:

$$i = \left\lfloor \frac{x - a}{b - a} \cdot m \right\rfloor,$$

где x — значение ключа, a и b — границы диапазона, m — количество карманов.

Для хранения данных реализован собственный динамический массив (класс TVector), работающий по принципу автоматического увеличения ёмкости при переполнении. Увеличение размера массива происходит по схеме удвоения текущей ёмкости.

Сортировка элементов внутри каждого кармана выполняется методом сортировки вставками (Insertion Sort), что эффективно при малом количестве элементов в каждом кармане.

Итоговая сложность алгоритма составляет $O(n+k)$, где n — количество элементов, k — количество карманов. При равномерном распределении входных данных алгоритм работает за линейное время.

Ввод данных осуществляется из стандартного потока ввода, вывод — в стандартный поток вывода.

Описание программы

Программа реализована на языке C++ и состоит из одного исходного файла.

Структура программы

1. **Структура `TKeyValuePair`.**

   Используется для хранения пары «ключ–значение». Поле `Key` имеет тип `double`, поле `Value` — тип `unsigned long long`.

2. **Класс `TVector`.**

   Реализует динамический массив для хранения элементов типа `TKeyValuePair`. Класс содержит:

   - поле `Size` — текущее количество элементов,
   - поле `Capacity` — текущую ёмкость массива,
   - указатель `Data` — динамически выделенный массив элементов.

   При переполнении массива выполняется увеличение ёмкости в два раза (метод `Resize`). Добавление элемента осуществляется методом `PushBack`.

3. **Функция `InsertionSort`.**

   Выполняет сортировку элементов внутри одного кармана методом сортировки вставками. Применяется к каждому непустому карману после распределения элементов.

4. **Функция `BucketSort`.**

   Реализует алгоритм карманной сортировки:

   (a) создаётся массив карманов,

   (b) элементы распределяются по карманам в зависимости от значения ключа,

   (c) элементы внутри каждого кармана сортируются,

   (d) карманы последовательно объединяются в итоговую последовательность.

5. **Функция `main`.**

   Выполняет чтение входных данных из стандартного потока ввода, вызов функции сортировки и вывод отсортированных данных в стандартный поток вывода.

Логика работы программы

1. Чтение входных данных до достижения конца входного потока.
2. Вычисление ширины кармана на основе заданного диапазона ключей.
3. Распределение элементов по карманам.
4. Локальная сортировка элементов внутри каждого кармана.
5. Формирование итоговой упорядоченной последовательности.
6. Вывод результата в требуемом формате.

Дневник отладки

В ходе выполнения лабораторной работы в код было внесено много изменений и исправлений. В основном действия касались трёх аспектов: корректности алгоритма, устойчивости вычислений и строгого соблюдения формата вывода.

1. Проблемы формата ввода-вывода.

Изначально программа использовала перенаправление потоков через `freopen`. Локально это работало корректно, однако в системе тестирования возникали ошибки WA.

После удаления `freopen` программа была переведена на `std::cin / std::cout`. Далее возникли расхождения в формате вывода (лишние пробелы, различие между пробелом и табуляцией, неточное количество знаков после запятой). Для устранения этих проблем потоковый вывод был заменён на форматированный вывод через `printf`, что позволило строго задать формат:

`%.*f`

с фиксированной точностью.

2. Ошибка в вычислении индекса кармана.

В ранней версии индекс кармана вычислялся через нормализацию:

$$\text{index} = \left(\frac{x-a}{b-a}\right) \cdot (m-1).$$

На практике при $x = b$ из-за особенностей представления числа типа `double` выражение могло давать значение, немного превышающее 1, что приводило к выходу индекса за границы массива. Это проявлялось на граничных тестах.

Для повышения численной устойчивости формула была переписана в эквивалентной, но более контролируемой форме:

$$\text{bucketWidth} = \frac{b-a}{m}, \quad \text{index} = \left\lfloor \frac{x-a}{\text{bucketWidth}} \right\rfloor.$$

Дополнительно была введена явная проверка границ индекса. Это гарантировало корректное распределение элементов даже при погрешностях округления.

3. Выбор количества карманов.

В начальной версии использовалось относительно небольшое количество карманов (1000). Это приводило к высокой плотности элементов внутри отдельных карманов. Поскольку внутри кармана применяется сортировка вставками, в худшем случае это ухудшало асимптотику до квадратичной внутри отдельных сегментов.

Количество карманов было увеличено для более равномерного распределения элементов по диапазону. Это позволило приблизить практическую сложность к ожидаемой $O(n+m)$ и уменьшить нагрузку на локальную сортировку.

4. Структурные упрощения.

Первоначально использовались:

- пространство имён,
- шаблонный класс вектора,

Эти элементы не влияли на корректность алгоритма, но усложняли анализ кода и потенциально затрудняли проверку стилевых требований. В финальной версии код был упрощён до минимально необходимой структуры с сохранением всей логики алгоритма.

5. Технические ошибки.

В процессе отладки были устранены:

- пропущенная фигурная скобка в функции `main`,
- неточности в вычислении границ диапазона.

После внесения всех исправлений программа:

- корректно распределяет элементы по карманам по формуле $i = \left\lfloor \frac{x-a}{b-a} \cdot m \right\rfloor$,
- устойчива к граничным значениям диапазона,
- соблюдает требуемый формат вывода,
- демонстрирует линейную сложность при равномерном распределении входных данных.

Тест производительности

Для оценки практической сложности алгоритма были проведены замеры времени работы программы при различных объёмах входных данных от 1 до 15 миллионов элементов. Данные генерировались равномерно в диапазоне $[-100, 100]$. При проведении измерений вывод отсортированной последовательности отключался, поскольку операции ввода-вывода существенно искажают результаты замера времени.

Результаты измерений показали устойчивый рост времени выполнения при увеличении количества элементов. Для наглядности зависимость времени работы от объёма входных данных представлена на графике (рис. 1).

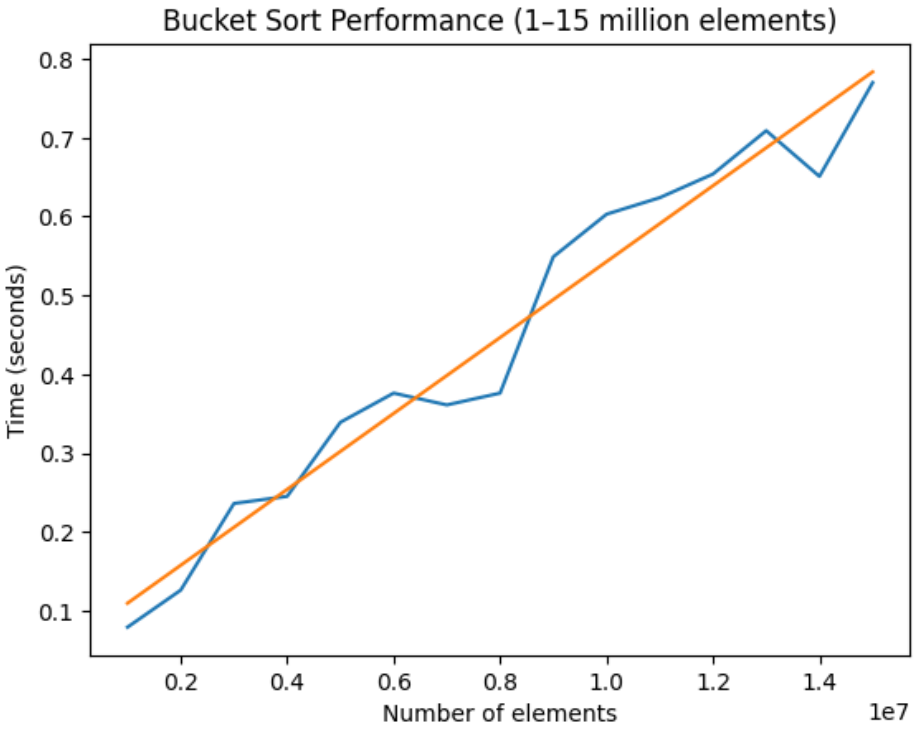

Рис. 1: Зависимость времени работы карманной сортировки от количества элементов (1–15 млн)

Как видно из графика, рост времени выполнения имеет практически линейный характер. Небольшие отклонения от идеальной прямой объясняются особенностями работы динамической памяти, влиянием кэширования процессора и неравномерностью распределения элементов по карманам. Тем не менее общий тренд подтверждает, что при фиксированном числе карманов алгоритм демонстрирует поведение, близкое к линейной сложности.

Экспериментально подтверждается соответствие практического времени работы теоретической оценке $O(n+m)$. Поскольку число карманов m фиксировано, доминирующей становится линейная часть $O(n)$..

Недочёты

Несмотря на корректную работу программы, в реализации присутствуют некоторые технические упрощения и компромиссы.

Во-первых, для обработки граничных случаев используется явная проверка выхода индекса кармана за пределы допустимого диапазона:

$$\text{if (index} >= \text{BUCKETS\_COUNT) index} = \text{BUCKETS\_COUNT - 1;}$$

Данная проверка добавлена для защиты от погрешностей представления вещественных чисел типа `double`. Формально при точных вычислениях выход за границы происходить не должен, однако на практике возможны округления. Это решение является защитным механизмом и упрощает обработку граничных случаев.

Во-вторых, число карманов задаётся фиксированной константой. Оптимальным с точки зрения теории было бы выбирать число карманов в зависимости от объёма входных данных, однако в рамках лабораторной работы выбрано фиксированное значение, обеспечивающее приемлемую производительность при тестируемых размерах входа. Адаптивный выбор числа карманов не реализован для сохранения простоты алгоритма.

Внутри каждого кармана используется сортировка вставками. При крайне неравномерном распределении данных это может привести к ухудшению производительности вплоть до квадратичного поведения внутри отдельного кармана. В рамках задания предполагается равномерное распределение ключей, поэтому более сложные методы локальной сортировки не применялись.

С точки зрения производительности следует отметить, что реализация активно использует динамическое выделение памяти (создание массива карманов и расширение внутренних массивов). При очень больших объёмах данных это может приводить к дополнительным накладным расходам времени из-за операций выделения и освобождения памяти. Более оптимизированная версия могла бы заранее резервировать память или использовать пул памяти, однако такие усложнения не требовались условиями работы.

Также алгоритм требует дополнительную память размером $O(n + m)$: помимо исходного массива создаётся массив карманов, каждый из которых содержит собственный динамический массив. При больших значениях n это увеличивает общее потребление памяти. Тем не менее в рамках тестируемых объёмов входных данных память использовалась в допустимых пределах.

Перечисленные ограничения являются осознанными компромиссами, не влияющими на корректность работы программы и не нарушающими требований лабораторной работы.

Выводы

В рамках лабораторной работы была реализована карманная сортировка для упорядочивания пар «ключ–значение» по вещественному ключу в фиксированном диапазоне. Алгоритм эффективен при обработке больших массивов данных с равномерным распре-

делением ключей, так как в среднем демонстрирует линейную временную сложность $O(n)$.

Область применения данного алгоритма связана с задачами, где заранее известен диапазон значений ключей и допускается использование дополнительной памяти. Типовыми примерами являются предварительная обработка числовых данных, сортировка результатов вычислений, обработка статистических выборок и других структур, где требуется быстрая сортировка большого объёма вещественных значений.

С точки зрения сложности программирования задача оказалась не тривиальной. Основные трудности были связаны с корректным вычислением индекса кармана, обработкой граничных значений диапазона, а также обеспечением строгого соответствия формату ввода и вывода. Дополнительные сложности возникли при реализации собственного динамического массива без использования стандартных контейнеров.

В ходе работы удалось проанализировать особенности численных вычислений с плавающей точкой, влияние распределения данных на производительность алгоритма и соотношение между временем работы и объёмом используемой памяти.

Таким образом, поставленная задача была решена, а полученная реализация демонстрирует ожидаемую линейную зависимость времени работы от объёма входных данных при выполнении предположений о равномерности распределения ключей.