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

Выполнил студент группы М8О-208Б-22 МАИ *Иванов Андрей.*

Условие

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:

- H.3-2
- Поразрядная сортировка
- Тип ключа: число от 0 до $2^{64} - 1$
- Тип значения: строки переменной длины (до 2048 символов)

Метод решения

Для выполнения данного задания была реализована поразрядная сортровка. В данном случае был использован метод сортировки наименьшего значащего разряда (LSD - Least Significant Digit). Он работает путем сортировки чисел по их битовым представлениям, начиная с младших разрядов и постепенно перемещаясь к старшим. Основная идея алгоритма заключается в том, чтобы разделить числа на бакеты в соответствии с их разрядами и для каждого бакета применить устойчивую сортировку подсчетомю Элементы из бакетов распределяются в буфер в соответствии с их позицией в порядке сортировки и исходный массив переупорядочивается в соответствии с буфером.

Описание программы

Входные данные представляют собой пару "ключ-значение" , которые хранятся в структуре TPair. Тип данных для ключа size_t позволит хранить любой ключ вплоть до $2^{64} - 1$ символов. Значением ключа является строка перменной длины для её хранения был создан класс TString, содержащий информацию о размере строки и массив элементов типа char. В данном классе реализованы функции:

- TString() - конструктор
- TString(const char* str) - конструктор с заданием элемента
- TString(const TString other) - конструктор копирования

- TString operator=(const TString other) - оператор присваивания
- friend std::ostream operator«(std::ostream os, const TString str) - перегрузка оператора вывода
- TString() - деструктор класса

Для хранения пар был создан класс TVector, содержащий информацию о размере, ёмкости и данных объекта. В нем реализованы функции:

- TVector() - конструктор по умолчанию, создающий пустой вектор
- TVector(size_t newSize) - конструктор, создающий вектор заданного размера
- TVector(const TVector& other) - конструктор копирования
- ~TVector() - деструктор класса
- _t Size() const - возвращает текущий размер вектора.
- void PushBack(const T& value) - добавляет элемент в конец вектора
- void Assign(size_t& newSize, T value) - заменяет содержимое вектора новыми данными указанного размера.
- const T& operator[](size_t index) const - иозвращает константную ссылку на элемент вектора по указанному индексу
- T& operator[](size_t index) - возвращает ссылку на элемент вектора по указанному индексу
- T* begin() - возвращает указатель на первый элемент вектора
- T* end() - возвращает указатель на элемент, следующий за последним элементом вектора
- const T* begin() const - возвращает константный указатель на первый элемент вектора.
- const T* end() const - возвращает константный указатель на элемент, следующий за последним элементом вектора
- void Resize(size_t& newCapacity) - изменяет размер внутренней ёмкости вектора.

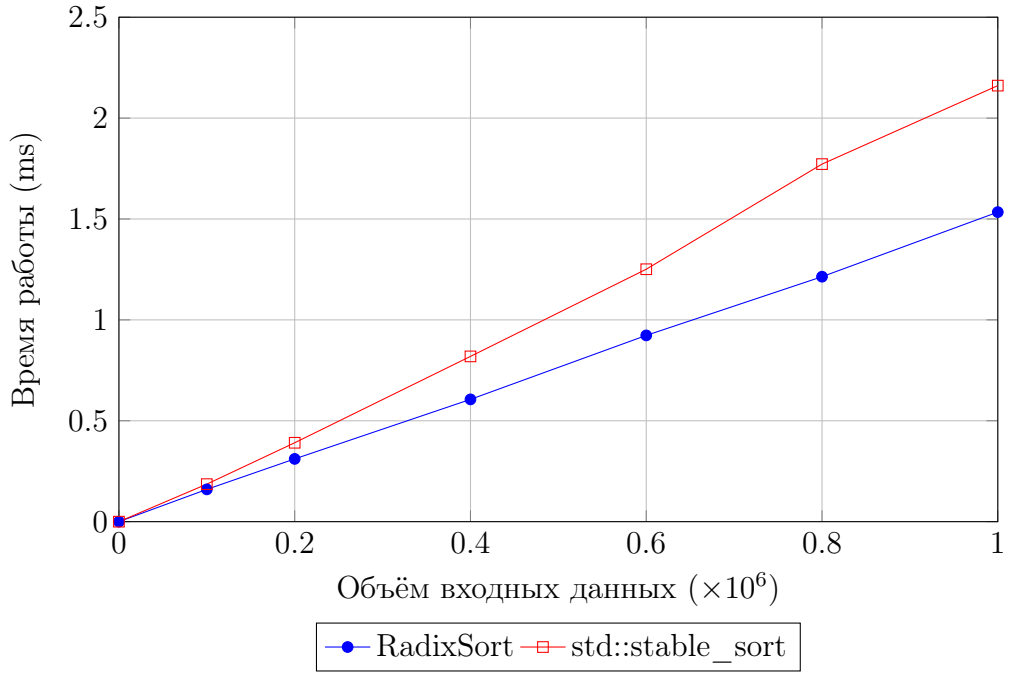
Для каждой пары в векторе применяется поразрядная сортировка по ключу.

Дневник отладки

В начале не был учтен завершающий нулевой символ при выборе максимального размера строки, из-за чего происходило переполнение буфера.
Далее производились попытки оптимизировать сортировку. Уменьшалось количество битов для вычисления разрядов, чтобы производить сортировку на меньшем количестве групп. Производился поиск максимального значения ключа, что позволяет определить диапазон значений ключей, которые могут присутствовать в массиве.
В конечном итоге вернулись к начальному варианту сортировки, но был создан отдельный вектор для хранения строк-значений ключей, а в пару к ключу записывался индекс этой строки типа size_t, что позволило уменьшить объем данных, который нужно сортировать и перемещать.

Тест производительности

Для измерения производительсти сравнивается время выполнения реализованной поразрядной сортировки и сортировки std::stable_sort на одинаковых входных данных. Подсчет времени производился с помощью библиотеки chrono, которая позволяет фиксировать время в начале и конце выполнения сортировки. Первыйтест состоит из 10^6 пар клбч-значения, второй из $2 * 10^6$ пар, третий из $4 * 10^6$, четвертый из $6 * 10^6$, пятый из $8 * 10^6$ и шестой из 10^7 пар.



На графике представлена зависимость времени выполнения сортировки от объёма

входных данных. Сложность поразрядной сортировки $O(n * k)$, где n - количество элементов массива, k - число разрядов ключа. Сложность stable_sort $O(n * log(n))$.

Выводы

В ходе данной лабораторной работы была изучена реализация классов STL, разобрана поразрядная сортировка за линейное время. Было выяснено, что данная сортировка работает наиболее эффективно именно с большим объёмом данных. Также были освоены отладка утечек памяти и оптимизация алгоритма сортировки. Полученные результаты подчеркивают важность выбора правильного алгоритма сортировки в зависимости от объема данных для достижения оптимальной производительности.