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

Выполнил студент группы 08-208 МАИ *Милько Павел.*

Условие

Требуется разработать программу, осуществляющую ввод пар "ключ-значение" и их сортировку за линейное время:

1. На каждой непустой строке входного файла располагается пара "ключ-значение", разделённые знаком табуляции. В выходных данных должны быть отсортированные строки исходной последовательности (за исключением пустых)

2. Вариант задания: 4-1

Ключи— Даты в формате DD.MM.YYYY, например:
1.1.1, 1.9.2009, 01.09.2009, 31.12.2009.

Значения— Строки фиксированной длины 64 символа, во входных данных могут встретиться строки меньшей длины, при этом строка дополняется до 64-х нулевыми символами, которые не выводятся на экран.

Метод решения

1. Данные на вход программе подаются через перенаправление вывода из файла, и, как следствие, весьма удобно считывать циклом while(особенно это важно при неизвестном количестве строк).
Когда будет считан символ EOF, цикл завершится.

2. Предусмотрена работа программы с неизвестным количеством входных данных.

3. По алгоритму сортировки необходим дополнительный массив ключей, размером равным количеству пар "ключ-значение", для переключения между двумя буферами (исходным и вспомогательным) используется дополнительный массив с ссылками на оба буфера.

4. Сортировку происходит по нескольким полям значений, для более лаконичного вида был использован цикл "foreach"
сортировка для каждого разряда числа производится так же в функции "main"

5. Собственно сам алгоритм сортировки принимает на вход ссылки на входной и выходной буфер, а так же переменную, описывающую текущий сортирующийся разряд. Так же содержит в себе цикл "while" с условием, что каждая строка "нашла своё место" в выходном буфере

Описание программы

- **main.cpp**
  Основной файл, содержит в себе собственно функцию “main” и функцию сортировки “RadixSort”

- **struct.cpp**
  По сути является дополнением к структуре, описанной в заголовочном файле. Содержит только функцию парсинга строки-ключа и запись полученных данных в структуру

- **struct.hpp**
  Содержит в себе все используемые константы, структуру ключа, прототип функции парсинга и целиком функцию перевыделения памяти (создание массива увеличенного размера и перезапись в него данных исходного)

Дневник отладки

При создании следующей таблицы была использована история локального гит-репозитория.

Время	Коммит	Описание
11 21:22:25	init	Начало работы, есть только шаблоны файлов и функций
11 22:05:16	parsing	Заготовки под функцию парсинга, поиск необходимых методов и тестирование функции + 3 таких же коммита
11 22:16:57	оптимизация указателей	Изменение структуры ключа, замена строки ссылкой на элемент отдельного массива строк
12 13:31:19	убрал утечки памяти	Не освобождалась вся память, в самой структуре ключа я выделил память для указателя на строку, потом это значение перезаписывалось с ввода, а выделенная ранее память не освобождалась
12 13:55:02	доделать сортировку	Была проблема с логикой алгоритма сортировки, цикл завершался досрочно, либо значения не отсортировывались в нужном порядке + 3 коммита мелкие исправления
15 21:33:11	cut last symbol	Заметил, что обрезается последний символ строки-значения + 18 коммитов на то, чтобы понять, что необходимо создать механизм динамического расширения выделенной памяти
19 14:36:07	WORKING!!!!	Рабочая версия программы, чекер отправил OK, далее проходила незначительная оптимизация кода и кодстайла
22 11:39:58	string->char*	Узнал что такое стандартные контейнеры STL в C++, пришлось переводить все вхождения на массив char-ов
22 21:47:39	Conditional jump or move	Надо было учесть, что строки обязаны заканчиваться символом \0, для корректной обработки строк

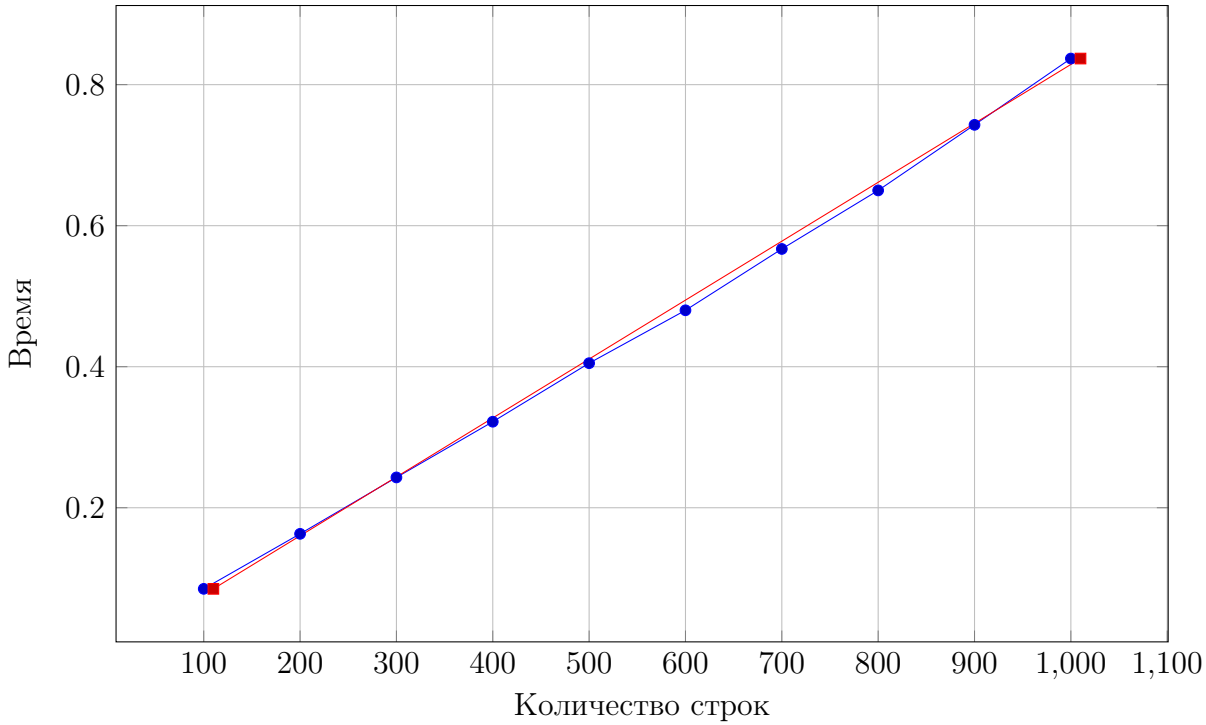
Размер файла	Имя файла и количество тысяч строк в нём
4,1M	100k
8,1M	200k
13M	300k
17M	400k
21M	500k
25M	600k
29M	700k
33M	800k
37M	900k

Так же при проверке работы программы учитывалось время её исполнения (утилита time), осуществлялся контроль различных ошибок и утечек памяти (утилита valgrind) и отдельно для тестов применялась утилита memusage.

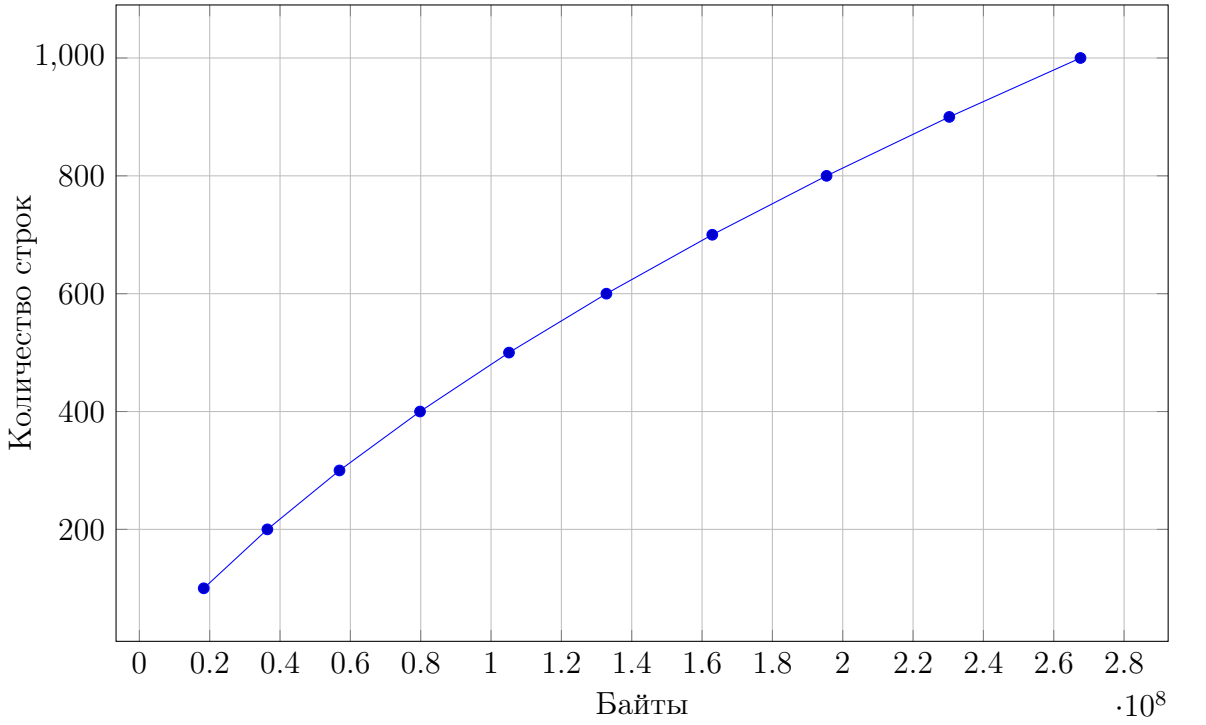
Тест производительности

Тесты создавались с помощью небольшой программы на языке Python:

```
def tests():
    from random import randint as ri
    from os import urandom as ur
    res=[]
    for count in range(1,10):
        f=open(str(count)+'00k','w')
        for i in range(100000):
            res.append(str(ri(0,30))+'.'+\
            str(ri(0,12))+'.'+\
            str(ri(0,2020))+'\t'+\
            ur(ri(0,32)).hex())
        for i in res:
            f.write(i+'\n')
        f.close()
```



Итого, по графику результатов времени выполнения программы, её сложность близка к линейной.



Небольшой изгиб графика обусловлен тем, что при заполнении выделенной памяти выделяется дополнительная память для 100 000 значений, а не для одного, что

более логично, но менее эффективно. При числах, на порядок больших чем размер изначально выделяемой памяти, зависимость размера выделяемой памяти приближается к линейной.

Недочёты

В ходе проверки работы на чекере выяснилось, что весьма важной характеристикой работы программы является время вывода данных в консоль (например для 1 000 000 входных строк только вывод конечного результата работы программы занимал 7.1 секунды, при времени сортировки около 0.9 секунд). Было найдено несколько способов оптимизации вывода, применимых для C++.

Наиболее результативным оказалось использование printf в качестве вывода и опции std::ios_base::sync_with_stdio(false), которая отключает синхронизацию потоков Си и C++. Т. е. не происходит копирование данных из буфера в буфер, что даёт некоторый выигрыш в скорости вывода данных на стандартный вывод.

Выводы

Данную программу можно применять для сортировки важных исторических событий, для их последующей обработки и добавления, например, в учебник.

Также существует вариант создания базы данных военкомата, в которой в роли строки-значения будет выступать адрес проживания призывника, а в качестве ключа будет использоваться дата рождения. Такая база, отсортированная по дате рождения обеспечивает более лёгкий анализ данных о призывниках текущего года.

Написание программы было сложным из за отсутствия навыков работы с языком C++, большая часть времени была потрачена на поиск оптимального, кратчайшего и эффективного способа выполнения функций.

Проблемы возникали при работе с указателями на динамические массивы, часто встречались логические ошибки при выделении, освобождении и перезаписи ячеек памяти.

В целом написание даной лабораторной было полезным, так как послужило поводом к изучению механизмов динамического выделения памяти, и методов парсинга строк. Понимание этих алгоритмов важно для написания эффективного и лаконичного кода.