Лабораторная работа 3: ассоциативные контейнеры

Цель работы:

* Реализовать приложение для построения частотного словаря документа. Сравнить производительность различных вариантов реализации.
* Реализовать приложение для нечеткого поиска в словаре методом n-грамм.

Структура проекта:

task-03-associative-containers

|-- .clang-format [опционально]

|-- .gitignore

|-- .gitlab-ci.yml

|-- Makefile

|-- freq\_umap.cpp

|-- freq\_map.cpp

`-- ngspell.cpp

**Задание 1: частотный словарь**

Приложение в качестве аргументов командной строки принимает на вход путь к файлу, для которого нужно построить частотный словарь. Файл содержит по одному слову в строке.

Приложение выводит ТОП-10 наиболее частотных слов по убыванию частоты, время заполнения частотного словаря и время сортировки в микросекундах. Слова с одинаковой частотой упорядочиваются лексикографически.

Пример сессии:

$ ./freq\_map ./file.txt

150 the

50 a

3 an

<часть вывода пропущена>

build: 42

sort: 3

Напишите две программы, которые будут отличаться типом ассоциативного контейнера для подсчета слов. Сравните время построения словаря и топа частотных слов. Объясните результаты.

Для получения достоверных результатов замера времени:

1. Собирайте приложения с опцией -O3.
2. Исключите из замеров операции ввода-вывода.

Для замеров можете воспользоваться образцом кода:

#include <chrono>

const auto start = std::chrono::steady\_clock::now();

// ...Исследуемый участок кода...

const auto finish = std::chrono::steady\_clock::now();

const auto duration = std::chrono::duration\_cast<std::chrono::microseconds>((finish - start));

std::cout << duration.count() << '\n';

**Задание 2: нечеткий поиск**

Запуск приложения:

ngspell <dict\_file\_path>

dict\_file\_path — путь к файлу со словарем. Файл содержит по одному слову на строке. Пример файла для отладки: dict10000.txt. Не нужно коммитить словарь в репозиторий.

После запуска приложение строит в памяти обратный индекс для поиска слов. Ключ индекса — 3-грамма, значение — список слов, содержащих эту 3-грамму. Слова короче трех символов игнорируются.

Пусть содержимое словаря:

акробат

дилетант

крокодил

проходил

Тогда разбиение слов на 3-граммы:

акробат: [акр, кро, роб, бат]

дилетант: [дил, иле, лет, ета, тан, ант]

крокодил: [кро, рок, око, код, оди, дил]

проходил: [кро, рох, охо, ход, оди, дил]

По этому разбиению будет построен обратный индекс (приведен фрагмент):

акр: [акробат]

кро: [акробат, крокодил]

дил: [дилетант, крокодил, проходил]

рок: [крокодил]

око: [крокодил]

код: [крокодил]

оди: [крокодил, проходил]

…

После построения индекса приложение считывает из стандартного потока ввода слова, по одному на строку и выполняет их поиск в индексе. Приложение завершается при вводе :q.

Алгоритм поиска:

1. Разбить поисковый запрос на 3-граммы.
2. Для каждой 3-граммы найти в обратном индексе соответствующие слова.
3. Вычислить, сколько раз таким образом удалось найти каждое слово. Этот счетчик будет весом.
4. Оставить только кандидатов с максимальным весом.

В результате нужно вывести количество кандидатов и найденные слова в алфавитном порядке.

Пример. Пусть пользователь ввел слово кракодил. Тогда его разбиение на 3-граммы:

кракодил: [кра, рак, ако, код, оди, дил]

При поиске слов по построенным 3-граммам в словаре получим следующий список кандидатов со счетчиками попаданий:

крокодил 3

проходил 2

дилетант 1

Максимальное количество попаданий — в слово «крокодил». Отсекаем все слова со значениями счетчиков меньше.

Пример сессии в приложении:

$ ./ngspell mydict.txt

кракодил

1

крокодил