Московский авиационный институт (национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Дискретный анализ»

Студент: О. А. Мезенин
Преподаватель: Н. К. Макаров
Группа: М8О-306Б-21

Дата:
Оценка:
Подпись:

Москва, 2024

Курсовой проект

Задача: Реализуйте систему для поиска аудиозаписи по небольшому отрывку.

```
./prog index --input <input file> --output <index file>
```

Ключ	Значение
–input	входной файл с именами файлов для индексации
–output	выходной файл с индексом

```
./prog search --index <index file> --input <input file> --output <output file>
```

Ключ	Значение
–index	входной файл с индексом
–input	входной файл с запросами
–output	выходной файл с ответами на запросы

Все файлы будут даны в формате MP3 с частотой дискретизации 44100Гц.
Входные файлы содержат в себе имена файлов с аудио записями по одному файлу в строке.
Результатом ответа на каждый запрос является строка с названием файла, с которым произошло совпадение, либо строка “! NOT FOUND”, если найти совпадение не удалось.

1 Описание

Для получения частотных характеристик сигналов, развёрнутых во времени, используем оконное дискретное преобразование Фурье, применяя перед этим окно Ханна [3]. Для вычисления дискретного преобразования Фурье будем использовать метод быстрого преобразования Фурье (БПФ) [4].

Далее необходимо придумать и реализовать способ индексации данных аудиосигнала, а также способ быстрого поиска аудиозаписи среди них.
Чтобы сделать поиск быстрым, нужно представить базу данных в виде хеш-таблицы, и при поиске аудиозаписи находить совпадения по хешам. Такая идея используются в приложении Shazam [8].

Вычислять хеш мы будем по данным, которые получаются в результате быстрого преобразования Фурье для каждого интервала. Но такой интервал включает в себя очень много данных. Разобьём этот сигнал на несколько частотных интервалов и найдём частоты, соответствующие пикам, т.е. максимальной амплитуде сигнала. Возьмём интервалы частот, присущие важным музыкальным компонентам, например: 40-80 Гц, 80-120 Гц для низких звуков и 120-180 Гц, 180-300 Гц для средних и высоких звуков [10]. Хеш будет представлен как число, сконкатенированное из частот, соответствующих пикам для каждого частотного интервала. Например, пусть есть следующие частоты: 45, 101, 144, 207 — их хеш будет соответственно **207**144**10**45.
Вместе с хешами в файле с индексом будет храниться информация о времени и названии аудиозаписи. Для сохранения информации о времени мы используем номер интервала, из данных которого брали хеш. Для сохранения информации о названии аудиозаписи используем ID аудиозаписи, а саму таблицу соответствия названия аудиозаписи и её ID будем сохранять вместе с файлом индекса.

Осталось придумать способ поиска совпадений хешей. Проблема в том, что хеши разных аудиозаписей могут совпадать. Поэтому проверять хеши недостаточно — нужно также проверять отрезки времени. Пусть мы проходимся по всем отрезкам времени аудиозаписи от i до j (по сути, это номера интервалов, для каждого из которых посчитан хеш), для которой выполняем поиск. Пусть мы нашли такие же хеши в базе данных для интервала i и j — этим хешам будут соответствовать векторы v_i и v_j. Тогда этот отрезок аудиозаписи есть в базе данных, если в векторах v_i и v_j есть записи, у которых одно и то же название аудиозаписи и длина отрезка времени одинакова, т.е. $\exists k, l : v_i[k].musicId = v_j[l].musicId \ \ \& \ \ |i - j| = |v_i[k].timeBlock - v_j[l].timeBlock|$.
Каждое такое совпадение мы будем учитывать в статистике совпадений. Затем мы будем сортировать массив со статистикой и брать название аудиозаписи, для которой совпадений было больше всего. Если количество совпадений будет превышать минимальный порог совпадений (например, 5), то мы будем считать, что нашли аудиозапись.

2 Исходный код

В программе есть два класса: `Database` для работы с сохранением/загрузкой базы данных, а также её хранением; `Audiosearch` с единственной функцией для поиска аудиозаписи. Также есть набор различных функций в файле `tools.hpp` — туда входят вычисление хешей и декодирование mp3-файла.

audiosearch.cpp	
std::string Audiosearch::search(const std::string &filename)	Функция поиска аудиофайла filename
uint32_t diff(uint32_t a, uint32_t b)	Вспомогательная функция для вычисления абсолютной разницы двух чисел
database.cpp	
void Database::loadFromNamesFile(const std::string &filename)	Функция для индексации файлов, пути к которым лежат в файле filename
void Database::loadFromIndexFile(const std::string &filename)	Функция для загрузки базы данных с файла с индексом filename
void Database::saveToFile(const std::string &filename)	Функция для сохранения базы данных в файл filename
std::vector<Entry> Database::findEntry(uint64_t hash)	Функция для нахождения записей (класса с полями timeBlock и musicId) по хешу hash
std::string Database::getMusic(uint32_t musicId)	Функция для возвращения названия аудиозаписи по musicId
tools.cpp	
std::vector<complex> hunn(const std::vector<short>& data, size_t start, size_t end)	Реализация окна Ханна
void fft(std::vector<complex> & data)	Реализация алгоритма Быстрого преобразования Фурье
uint64_t hash(const std::vector<short>& frequency)	Функция для вычисления хеша для частот frequency
size_t getFrequencyIndex(size_t frequency)	Функция для определения номера промежутка, в который входит частота frequency
std::vector<uint64_t> calculateHashes(const std::vector<short>& data)	Функция для применения окна Ханна и БПФ и вычисления всех хешей для данных data
std::vector<short> decoder(const std::string &filename)	Функция для чтения mp3-файла и возвращения значений амлитуд

3 Консоль

Для тестирования была взята музыка из игры «Космические рейнджеры». Будет два теста с разными файлами индекса.

Первый файл индекса будет содержать следующие аудиозаписи.

```
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5/tests$ cat input_for_index1.txt
./tests/Denis Korzhavin -The long way ballad.mp3
./tests/theo -Running Away.mp3
./tests/NiKiNiT -End of War.mp3
./tests/NiKiNiT -Fight 3.mp3
./tests/NiKiNiT -Fighter.mp3
```

Запустим команду для индексации и посмотрим на результат.

```
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5$ make
g++ -std=c++2a -pedantic -Wall -Wextra -Werror main.cpp audiosearch.cpp database.cpp
tools.cpp -o audiosearch.out
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5$ ./audiosearch.out index --input
./tests/input_for_index1.txt --output ./tests/index1.txt
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5$ head ./tests/index1.txt
5 49874
41 Denis Korzhavin -The long way ballad.mp3
23 theo -Running Away.mp3
24 NiKiNiT -End of War.mp3
21 NiKiNiT -Fight 3.mp3
21 NiKiNiT -Fighter.mp3
194123090042040 1 13007 4
273148082057040 1 13005 4
272132081041040 1 13004 4
209132081041040 1 13003 4
```

В индексе получили 5 аудиозаписей и 49874 хеша.

Для поиска подготовлены следующие аудиозаписи.

```
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5$ cat ./tests/input_for_search1.txt
./tests/Denis Korzhavin -The long way ballad (cut).mp3
./tests/Denis Korzhavin -The long way ballad (cut3 + bass).mp3
./tests/Denis Korzhavin -The long way ballad (recording full).mp3
./tests/Denis Korzhavin -The long way ballad (recording full) (cut).mp3
./tests/theo -Running Away (cut).mp3
./tests/Gregory Semenov -Fei (cut).mp3
```

Среди них соответственно:

- Denis Korzhavin - The long way ballad (cut).mp3 – отрывок оригинала;
- Denis Korzhavin - The long way ballad (cut3 + bass).mp3 – отрывок оригинала с наложенным эквалайзером;
- Denis Korzhavin - The long way ballad (recording full).mp3 – запись музыки с телефона;
- Denis Korzhavin - The long way ballad (recording full) (cut).mp3 – отрывок записи музыки с телефона;
- theo - Running Away (cut).mp3 – отрывок оригинала;
- Gregory Semenov - Fei (cut).mp3 – отрывок музыки, которой нет в базе.

Длина любого из отрывков не превышает 20 секунд.

Запустим команду для поиска и посмотрим на результат.

```
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5$ ./audiosearch.out search --input
./tests/input_for_search1.txt --index ./tests/index1.txt --output ./tests/output.txt
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5$ cat ./tests/output.txt
Denis Korzhavin -The long way ballad.mp3
Denis Korzhavin -The long way ballad.mp3
Denis Korzhavin -The long way ballad.mp3
! NOT FOUND
theo -Running Away.mp3
NiKiNiT -Fighter.mp3
```

Как видим, все аудиозаписи нашлись верно, кроме двух: совсем не нашёлся отрывок записи музыки с телефона и нашёлся (неверно) отрывок музыки, которой нет в базе.

Вторым тестом будет проверка на то, сможет ли программа найти замедленную аудиозапись. К предыдущему файлу индекса прибавится ещё одна аудиозапись, а искать мы будем отрывок её замедленной (75% скорость) версии.

```
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5$ cat ./tests/input_for_index2.txt
./tests/Denis Korzhavin -The long way ballad.mp3
./tests/theo -Running Away.mp3
./tests/NiKiNiT -End of War.mp3
./tests/NiKiNiT -Fight 3.mp3
./tests/NiKiNiT -Fighter.mp3
./tests/Gregory Semenov -Fei.mp3
```

```
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5$ ./audiosearch.out index --input
./tests/input_for_index2.txt --output ./tests/index2.txt
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5$ head -n 12 ./tests/index2.txt
6 62256
41 Denis Korzhavin -The long way ballad.mp3
23 theo -Running Away.mp3
24 NiKiNiT -End of War.mp3
21 NiKiNiT -Fight 3.mp3
21 NiKiNiT -Fighter.mp3
25 Gregory Semenov -Fei.mp3
184147091055040 1 22324 5
183164083041040 1 22314 5
184145110050040 1 22306 5
218145083073040 1 22294 5
182128091050040 2 22286 5 22296 5
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5$ cat ./tests/input_for_search2.txt
./tests/Gregory Semenov -Fei (75% speed) (cut).mp3
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5$ ./audiosearch.out search --input
./tests/input_for_search2.txt --index ./tests/index2.txt --output ./tests/output.txt
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5$ cat ./tests/output.txt
Gregory Semenov -Fei.mp3
```

Тест пройден — программа верно нашла аудиозапись.

4 Тест производительности

Замерим время для индексации файлов из второго теста и поиска отрывка аудиозаписи. Общий размер файлов для индексации составляет 28,9 МБ. Размер файла для поиска составляет 390,4 кБ.

```
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5$ time ./audiosearch.out index --input
./tests/input_for_index2.txt --output ./tests/index2.txt

real    6m1,583s
user    6m1,411s
sys     0m0,124s
aprold@SAI:~/Documents/GitHub/MAI-DA/CPfor5$ time ./audiosearch.out search
--input ./tests/input_for_search2.txt --index ./tests/index2.txt --output ./tests/outp

real    0m9,529s
user    0m9,162s
sys     0m0,009s
```

5 Выводы

Выполнив курсовой проект по курсу «Дискретный анализ», узнал про преобразование Фурье и его разновидность — оконное преобразование, в котором для сглаживания спектра при делении сигнала на интервалы применяется оконная функция, например, окно Ханна. Узнал про быстрое преобразование Фурье, которое вычисляет дискретное преобразование Фурье за время $O(n \log n)$. А также узнал об одном из применении преобразования Фурье в обработки сигнала: оно позволяет разложить сигнал на составляющие его частоты.
Узнал, как работает алгоритм поиска аудиозаписи в приложении Shazam. Разработал свою версию аудиопоиска. Программа хоть и прошла большинство представленных ей тестов, но она всё же нуждается в доработке. Во-первых, необходимо реализовать настройку анализа возможных искажений и посторонних звуков — скорей всего, отрывок записи с телефона не нашёлся именно из-за шумов и искажений. Во-вторых, стоит доработать условие определения, является ли найденная запись подходящей — потому что в процессе тестов нашлась запись, которой нет в базе данных. В-третьих, стоит поработать над оптимизацией и повысить скорость работы программы — одним из решений повышения скорости работы программы является добавление многопоточности.

Список литературы

[1] *Преобразование Фурье*
URL: https://ru.wikipedia.org/wiki/Преобразование_Фурье (дата обращения: 12.11.2023).

[2] *Дискретное преобразование Фурье*
URL: https://ru.wikipedia.org/wiki/Дискретное_преобразование_Фурье (дата обращения: 12.11.2023).

[3] *Оконное преобразование Фурье*
URL: https://ru.wikipedia.org/wiki/Оконное_преобразование_Фурье (дата обращения: 12.11.2023).

[4] *Быстрое преобразование Фурье за O (N log N)*
URL: https://e-maxx.ru/algo/fft_multiply (дата обращения: 12.11.2023).

[5] *Introduction to the Fast Fourier Transform Algorithm in C++*
URL: https://cpp.helpful.codes/algorithms/FFT-Algorithm/ (дата обращения: 12.11.2023).

[6] *Проектирование оконных функций, суммирующихся в единицу с заданным уровнем перекрытия*
URL: https://habr.com/ru/articles/430536/ (дата обращения: 12.11.2023).

[7] *Понимание аудиоданных, преобразования Фурье, БПФ, спектрограммы и распознавания речи*
URL: https://questu.ru/articles/276549/ (дата обращения: 12.11.2023).

[8] *Shazam: алгоритмы распознавания музыки, сигнатуры, обработка данных*
URL: https://habr.com/ru/companies/wunderfund/articles/275043/ (дата обращения: 05.01.2024).

[9] *An Industrial-Strength Audio Search Algorithm*
URL: https://www.ee.columbia.edu/ dpwe/papers/Wang03-shazam.pdf (дата обращения: 05.01.2024).

[10] *Creating Shazam in Java*
URL: https://www.royvanrijn.com/blog/2010/06/creating-shazam-in-java/ (дата обращения: 05.01.2024).