Курсовой проект по курсу дискретного анализа: Аудио-поиск

Выполнил студент группы М8О-307Б-20 МАИ *Чекменев Вячеслав.*

Условие

- Реализуйте систему для поиска аудиозаписи по небольшому отрывку. ./prog index –input <input file> –output <index file> — индексация файлов
- ./prog search –index <index file> –input <input file> –output <output file> — поиск файлов. 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.

Метод решения

1. Из полученной зависимости амплитуд от частот получить частоты.
2. Однако, в одной песне диапазон «сильных» частот может варьироваться. Поэтому, вместо того, чтобы за сразу проанализировать весь частотный диапазон, мы можем выбрать несколько более мелких интервалов. Выбор можно сделать, основываясь на частотах, которые обычно присущи важным музыкальным компонентам, и проанализировать их по отдельности. Найти среди них самые "сильные"частоты (с максимальным значением амплитуды) в каждом из интервалов между точками 0, 40, 80, 120, 180, 300, 500, 1000. Это действие проделать для каждого "окна"в 4096 отсчетов.
3. Тогда мы получим таблицу в строках которой будет храниться нужные нам частоты. Эти сведения и формируют сигнатуру для конкретного анализируемого блока данных, а она, в свою очередь, является частью сигнатуры всей песни.
4. Для упрощения поиска музыкальных композиций их сигнатуры используются как ключи в хэш-таблице. Когда набор частот, для которых найдена сигнатура, появился в произведении, и идентификатор самого произведения (название песни и имя исполнителя, например). Возьмем хэш-функцию:

$$h(f1, f2, ..., f7) = \sum_{k=0}^{7} (f_k - (f_k mod 2)) * 10^k$$

5. Тогда набор хэшей для каждого трека и будет являться сигнатурой наших аудиозаписей

6. При флаге "index"будем просто вычислять сигнатуры и записывать их в вектор, а потом и в файл

7. Всякий раз, когда удаётся обнаружить совпадающий хэш-тег, число возможных совпадений уменьшается, но вероятно, что только лишь эти сведения не позволят нам настолько сузить диапазон поиска, чтобы остановиться на единственной правильном треке. Поэтому в алгоритме распознавания музыкальных произведений нам нужно проверять ещё и длину временного интервала. А именно, сначала положим в 2d вектор сигнатуры исходных файлов, потом вычислим сигнатуру для сэмпла и найдем два значения в таблице сигнатур: первый элемент сигнатуры сэмпла, последний элемент сигнатуры сэмпла. А также сравним временной интервал между найденными совпадениями в сигнатурах исходных треков и длину самого сэмпла.

8. Если матч произошел сразу с больше, чем одним треком, то выведем первый, так как тут ничего поделать нельзя.

Дневник отладки

1. 04.01.23 Изучена теория

2. 05.01.23 Написаны тесты

3. 10.01.23 Написан код

4. 10.01.23 Протестирован код

Тестирование

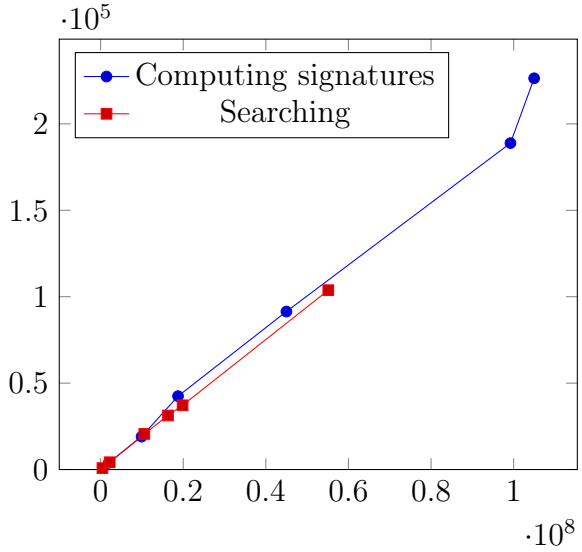
Для тестирования было взято две базы данных. Первая содержит опенинги из разных аниме (30 штук) длиной 1 минута 30 секунд и их части длиной 10 секунд, которые нам и надо найти. Вторая содержит случайно сгенерированные треки треки длиной примерно 45 секунд, а также их части длиной 15 секунд.

- Индексация треков:

Кол-во элементов в массиве pcm	Время (в миллисекундах)
1984500	3914
9922500	19039
18742500	42447
44982000	91415
99225000	188870
104958000	226389

- Поиск:

Кол-во элементов в массиве pcm	Время (в миллисекундах)
441000	819
2205000	4214
10584000	20609
16317000	31284
19845000	37183
55125000	103777



Сложность алгоритмов можно оценить как O(n*log(n))

- Точность поиска

Первый тест

```
01-Acknowledged_Product.mp3
Matches: Acknowledged_Product.mp3,

02-Bony_Communication.mp3
Matches: Bony_Communication.mp3,

03-Chirpy_Situation.mp3
Matches: Convivial_Measurement.mp3, Chirpy_Situation.mp3

04-Compassionate_Exam.mp3
Matches: Compassionate_Exam.mp3,

26-Primal_Arrival.mp3
```

```
! NOT FOUND

05-Conspicuous_Bedroom.mp3
Matches: Conspicuous_Bedroom.mp3,

06-Convivial_Measurement.mp3
Matches: Convivial_Measurement.mp3,

07-Cozy_Communication.mp3
Matches: Cozy_Communication.mp3,

28-Rightful_Indication.mp3
! NOT FOUND

08-Curvaceous_Elevator.mp3
Matches: Curvaceous_Elevator.mp3,

09-Destined_Goal.mp3
Matches: Destined_Goal.mp3,

10-Engaging_Agency.mp3
Matches: Engaging_Agency.mp3,
```

Не найдено или найдено неправильно - 3 из 12 треков => точность = 75%

Второй тест

```
01-attack_on_titan_op3.mp3
Matches: attack_on_titan_op3.mp3,

02-attack_on_titan_op4.mp3
! NOT FOUND

03-attack_on_titan_op4_2.mp3
Matches: attack_on_titan_op4_2.mp3,

04-death_note_op1.mp3
Matches: death_note_op1.mp3,

05-guilty_crown_op1.mp3
Matches: guilty_crown_op1.mp3,

06-guilty_crown_op2.mp3
Matches: guilty_crown_op2.mp3,
```

```
11-hunterxhunter_op1.mp3
Matches: hunterxhunter_op1.mp3,

13-naruto_storm_op4.mp3
Matches: psychopass_op4.mp3,

14-naruto_storm_op7.mp3
Matches: naruto_storm_op7.mp3,

15-naruto_ua_op2.mp3
Matches: naruto_ua_op2.mp3,

17-psychopass_ed1.mp3
! NOT FOUND

18-psychopass_op2.mp3
Matches: psychopass_op2.mp3,

19-psychopass_op4.mp3
Matches: psychopass_op4.mp3,

20-sao_op1.mp3
Matches: sao_op1.mp3,
```

Не найдено или найдено неправильно - 3 из 14 треков $=>$ точность $= 79\%$

Недочеты

Возникают ошибки при сэмплах взятых из центра трека и сильно зашумленных. В таком случае сигнатуры сэмпла и трека часто отличаются.

Выводы

Проделав данный курсовой проект я узнал много нового про аудиопоиск и в целом про обработку аналоговых сигналов. Глубже разобрался как работает преобразование фурье, смог достать из него не только магнитуды, но и самые сильные частоты. Также разобрался с возможными диапазонами частот для разных типов музыки. Еще мне пришлось придумать хэш функцию, которая будет создавать наименьшее количество коллизий - это у меня получилось и количество различных хэшей составляло $4/5$ от всех возможных.

Также столкнулся с проблемой при матчинге треков. Искать всю сигнатуру в исходной таблице сигнатур было бы неэффективно. Поэтому я воспользовался трюком: брал только перую и последнюю частоту и искал их в таблице, а также сравнивал временные интервалы в сэмпле и исходном треке.