Курсовая работа по курсу дискретного анализа: Аудиопоиск

Выполнил студент группы 08-310 МАИ *Железнов Илья*.

Условие

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.

Метод решения

Как и в варианте на "3"мы сначала используем наш декодер, только немного его меняем, а именно усредняем значение каналов для много канальных аудиофайлов. Такая оптимизация, помогает нам обрабатывать стерео- или моноканальный звук, создавая один моноканал. И действительно, такая оптимизация очень сильно улучшает наш результат, что логично. В ФФТ изменений нет, так как мы все также преобразуем сигнал в частотный спектр. Для каждого окна у нас теперь есть набор значений, где смысл i-го в том, что в окрестности i Гц имееют интенсивность равную модулю комплексного числа. Следовательно для каждого аудио мы можем сделать отпечаток со значениями после ФФТ.

Следующий этап — нам нужно реализовать помехоустойчивость, так не всегда нам будет приходить фрагмент (сэмпл) музыки, который будет без помех, следовательно

надо как-то это учесть. Можно заметить, что при искажениях, частостная характеристика не искажается, а значит мы можем разбить спектр на частоты. Можно разбить спектр на 7 октав и в каждом диапазоне сохраним частоту максимальной интенсивностью. Для каждого окна отсчетов получим отпечаток, в котором вместо большого значения будет 7 чисел.

Тепер поговорим о поиске. Алгоритмы, которые точно сравнивают значнеия нам не подходят, даже алгоритмы нахождения подпоследовательностей. Значит надо использовать алгоритм, который покажет нам разницу между двумя занчениями. Используем алгоритм Ливенштейна, который говорит о том, за сколько атомарных операций, мы получим из 1 слова 2. Но просто использовать это расстояние нам не подойдет, поэтому будем считать некий коэффициент: 1 - (Lev(a, s) / (2*n)), где n - это длина s. Ну и следовательно, получая такой коэффициент, мы можем понять, чем ближе наше значение к 1, тем больше совпадение у них. Реализовывать это будем при помощи алгоритма Вагнера-Фишера, это алгоритм динамического программирования и работает он за $O(n^2)$.

Но также у нас есть проблема в том, какие куски нам сравнивать, ведь отрывок нам могут дать из любой части песни. И если сравнивать "в лоб то такой алгоритм будет работать за $O(n * m^2)$, где n - длина семпла, а m - самого аудио. Значит нужно делать по-другому. Решение этому идти по отсчетам кратным длинне отрывка и высчитывать коэффициент, учитывая предидущий. Если их сумма больше определенного значения, значит начинаем сравнивать промежуток между ними. Будем двигаться так, чтобы искать там, где получается наибольший коэффициент, двигаяс по правилам бинарного поиска. Следовательно сложность падает до $O(n * m * \log(m))$.

Таким образом, программа в целом считывает аудиосигнал, декодирует его, применяет преобразование Фурье для анализа спектра, и выводит максимальные амплитуды для каждого блока аудиоданных, что может быть полезно для задач анализа звуковых частот или работы с аудио в реальном времени.

Описание программы

Пограмма состоит из основного файла main.cpp

1. *main.cpp* (В нем находится реализация кодирование аудио и преобразование сигнала. А так же все алгоритмы и функции, для реализации поиска.

Дневник отладки

Проблема с которой я столкнулся, это неверный вывод данных. Изначально я неправильно прочитал условие вывода и поэтому долго провозился с проблемой. Ну и также

долго подбирал коэффициенты для сравнения, чтобы получить лучшую точность.

Тест производительности

Для начала посмотрим, как вообще работает наш поиск с определенными помехами:

Помехи	**Audio 1 coeff**	**Audio 2 coeff**	**Audio 3 coeff**
Нет помех	0.979036	0.985484	0.741593
Диктофон	0.614183	0.649013	0.679231
Ускорение	0.699499	0.619239	0.700213
Замедление	0.744533	0.644513	0.715123
Тональность +	0.901722	0.799722	0.822722
Тональность -	0.745717	0.647727	0.717227
Эквалайзер	0.843655	0.747727	0.794025

Как мы видим, довольно неплохой результат, при условии неких помех. Лучше всего, как мы видим работает с повышением тональности и эквалайзером.

Сначала протестируем время работы индексации. Возьмем несколько аудиодорожек одной длины (1 минута) и будем индексировать их по несколько за раз:

COUNT OF AUDIO count	**TIME sec**
5	19.327
10	41.924
15	62.121

Результаты явно нам говорят что время при константной длине растет линейно.

Теперь протестируем поиск. Будем делать это так, возьмем одинаковое количество кусков и такой же количество в нашей библиотеке. При одинаковых k - количество треков и l - запросов. Мы получаем сложность O(n*m). А так сложность при условии, что поиск работает за O(n * m), то получается O(m * n * k * l).

NUMBER (k, l) count	**TIME sec**
5	14.922
10	55.122
15	102.671

Недочёты

В интернете я видел несколько разных реализаций, но показлось, что это более менее понятный и простой алгоритм.

Выводы

В ходе данной лабораторной работы был реализован алгоритм поиска аудиофайлов на основе спектрального анализа с использованием быстрого преобразования Фурье (FFT). Основное внимание уделялось оптимизации временной и пространственной сложности на этапах индексирования и поиска. Реализация индексации аудиофайлов с временной сложностью O(n + m * log(n))продемонстрировала высокую производительность при обработке длинных треков, что подчеркнуло значимость эффективного использования алгоритмов спектрального анализа. Сравнение спектров осуществлялось с применением динамического программирования для вычисления расстояния Левенштейна, что улучшило точность поиска.

Полученные результаты продемонстрировали важность выбора эффективных алгоритмов при работе с большими объёмами аудиоданных. Использование FFT позволило значительно сократить время обработки сигналов по сравнению с наивными подходами. Реализация поиска с временной сложностью O(m * n * k * l), где k — количество треков в базе, l - количество запросов, n — длина трека, а m — длина образца, показала практическую применимость методов обработки сигналов. Это укрепило понимание алгоритмической оптимизации и подтвердило важность анализа временной и пространственной сложности в разработке высокопроизводительных приложений.