КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Кафедра системного аналізу та теорії прийняття рішень

Кваліфікаційна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз»

за спеціальністю 124 «Системний аналіз»

на тему:

АЛГОРИТМИ ТА МЕТОДИ ЗНАХОДЖЕННЯ ТА РОЗПІЗНАННЯ АУДІОФАЙЛІВ

Студента 4 курсу
Пострелка Владислава Юрійовича
Науковий керівник:
асистент, кандидат тех. наук
Махно М.Ф.

Робота заслухана на засіданні кафедри системного аналізу та теорії прийняття рішень та рекомендована до захисту в ДЕК, протокол №1_0 від 0_7 червня 2022 р.

Завідувач кафедри системного аналізу та теорії прийняття рішень

професор, доктор фіз.-мат. наук Наконечний О.Г.

Київ – 2022

ЗМІСТ

ВСТУП

На сьогоднішній день музика є невід'ємною частиною нашого життя. Ми чуємо її де завгодно: на вулиці, у закладах, по телебаченню, навіть знаходячись вдома ми можемо почути її від сусідів, які голосно її слухають.

У кожної людини свої побажання щодо жанру музики, темпу, виконавця. Існують багато музичних сервісів, таких як: Spotify, Deezer, Youtube Music та інші, які накопичують в собі величезну бібліотеку пісень на любий смак. Технології, які містять в собі ці сервіси, спрощують знаходження пісень, які б сподобались людині і кожен день ці технології покращують або створюють нові.

Але я б хотів виокремити одну технологію, яка здається мені найцікавішою: розпізнання пісні. Принцип цього методу дуже простий. Наприклад, ви знаходитесь у барі і чуєте музику, яка вам сподобалась. Ви хочете дізнатись назву цієї мелодії. Для цього відкриваєте програму (Shazam, MusicBrainz або аналоги), натискаєте одну кнопку, чекаєте 10 секунд і застосунок видає вам результат. Хіба не магія? А це ми зараз і розберемо.

Почнемо з простої теорії – що таке звук? Це механічні коливання, що поширюються у вигляді хвиль у газі, рідині чи твердому тілі та сприймається слухом. В результаті ці коливання перетворюються у електричні імпульси та передаються по слуховим нервам до мозку. Пристрої для запису звуку дуже точно імітують цей процес, конвертуючи тиск звукової хвилі в електричний сигнал [17]. Оскільки такі сигнали в цифровому світі не дуже корисні, то їх треба перетворювати в дискретну форму.

Робиться це за допомогою вибірки значень, що становлять значення амплітуди сигналу. Під час такого перетворення, ми проводимо квантування аналогового сигналу, тобто аналого-цифровий перетворювач проводить дуже багато операцій з перетворення дрібних частин аналогового сигналу у цифровий. Цей процес називають дискретизацією або семплінгом. Завдяки теоремі Котельникова ми знаємо, яка частота дискретизації потрібна у тому,

щоб точно уявити безперервний сигнал, обмежений деякою частотою. Зокрема, для того, щоб захопити весь частотний спектр звуків, доступних людському слуху, ми повинні використовувати частоту дискретизації, яка вдвічі перевищує верхню межу частот, які чує людина. Вона може чути звуки в діапазоні приблизно від 20 Гц до 20000 Гц. В результаті звук найчастіше записують із частотою дискретизації 44100 Гц. Саме ця частота використовується у компакт-дисках. Вона ж найчастіше застосовується для кодування звуку групи стандартів MPEG-1 (VCD, SVCD, MP3).

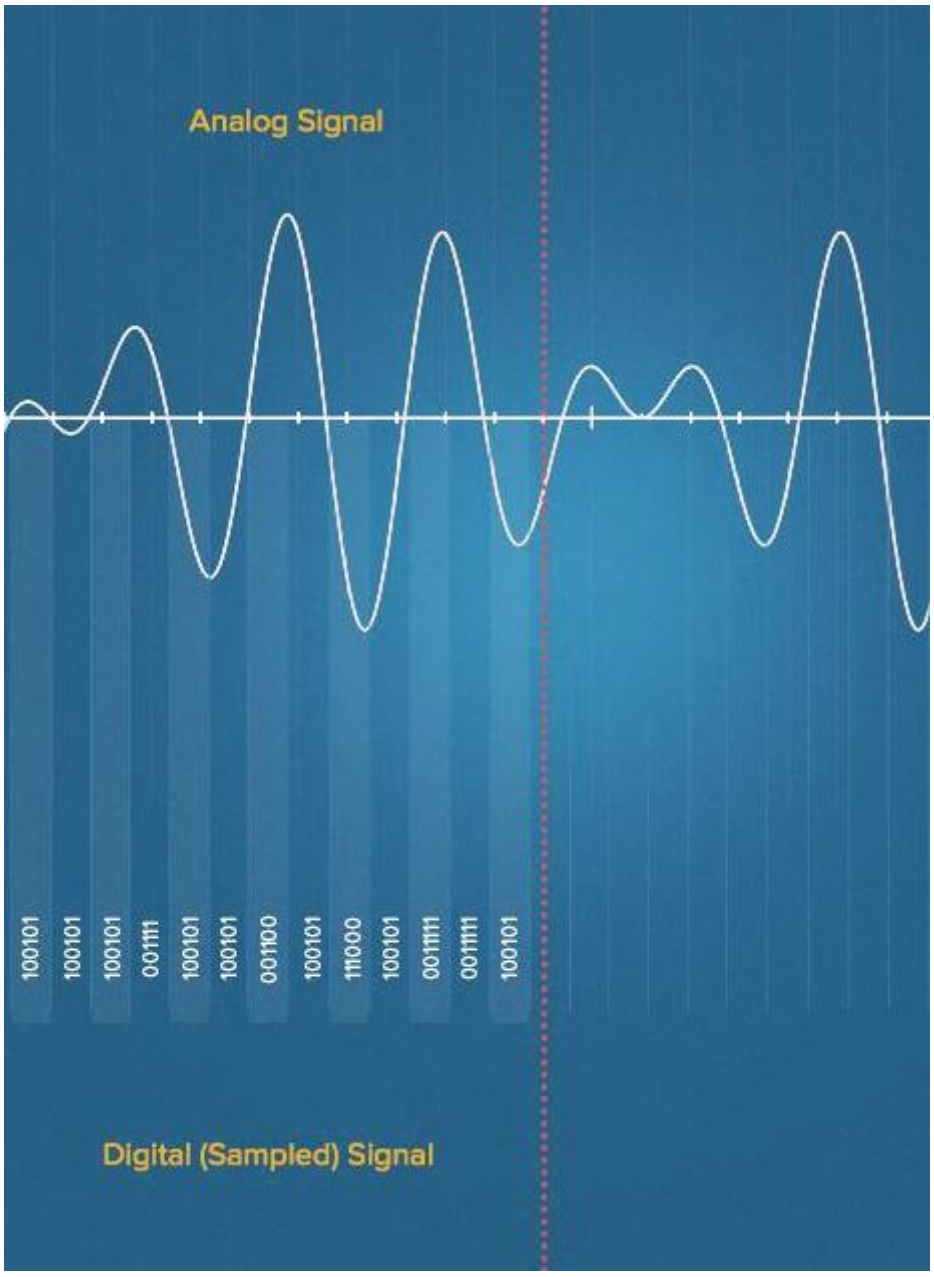


Рисунок 1.1 – Аналоговий (безперервний) та цифровий (дискретний) сигнали

Широкому використанню частоти дискретизації 44100 Гц ми зобов'язані, переважно, корпорації Sony. Свого часу звукові доріжки, закодовані таким чином , зручно було поєднувати з відео у стандартах PAL (25 кадрів за секунду) та NTSC (30 кадрів за секунду), працювати з ними, використовуючи існуюче обладнання. Дуже важливо і те, що ця частота є достатньою для якісної передачі звуку в діапазоні до 20000 Гц. Цифрове звукове обладнання, що використовує цю частоту дискретизації, цілком відповідало за якістю аналогового обладнання тих часів, коли відбувалося становлення стандартів цифрового звуку. У результаті, вибираючи частоту дискретизації звуку під час запису, ви, найімовірніше, зупиниться на 44100 Гц.

РОЗДІЛ 1

ОПИС МОДЕЛІ РОЗПІЗНАННЯ ЗВУКУ

1.1 Захоплення звуку

Записати семпльований звуковий сигнал – завдання досить просте. Сучасні звукові карти містять інтегровані аналого-цифрові перетворювачі. Тому достатньо вибрати мову програмування, знайти відповідну бібліотеку для роботи зі звуком, вказати частоту дискретизації, кількість каналів (зазвичай один або два, для монофонічного і стереофонічного звучання, відповідно), вибрати кількість бітів в одному семпле (наприклад, часто використовується 16 біт) [14]. Потім потрібно відкрити рядок даних зі звукової карти, так само як відкривається будь-який вхідний потік, і записати його вміст в байтовий масив.

1.2 Тимчасова та частотна області

У нашому масиві записано цифрове уявлення звукового сигналу у часовій області. Тобто, ми маємо відомості про те, як змінювалася амплітуда сигналу з часом.

У 19 столітті Жан Батіст Джозеф Фур'є зробив визначне відкриття. Полягає воно в тому, що будь-який сигнал у часовій області еквівалентний сумі деякої кількості (можливо, нескінченного) простих синусоїдальних сигналів, за умови, що кожна синусоїда має певну частоту, амплітуду та фазу. Набір синусоїд, які формують вихідний сигнал, називають рядом Фур'є [1].

Іншими словами, можна уявити практично будь-який сигнал, розгорнутий у часі, просто задавши набір частот, амплітуд та фаз, що відповідають кожній із синусоїд, які цей сигнал формують. Таке уявлення сигналів називають набором частотних інтервалів. У певному сенсі, відомості про частотні інтервали є чимось на кшталт «відбитків пальців» або сигнатур сигналів, розгорнутих у часі, даючи нам статичне уявлення динамічних даних.

На рис. 1.3 ми можемо побачити анімоване уявлення Ряду Фур'є для прямокутної хвилі частотою 1 Гц. Тут же показана апроксимація вихідного сигналу на основі набору синусоїд. На верхньому графіку сигнал показаний в амплітудно-часової області, на нижньому дано його подання в амплітудно-частотному вигляді.

Аналіз частотних характеристик сигналів значно полегшує вирішення безлічі завдань. Оперувати такими характеристиками у сфері обробки цифрових сигналів дуже зручно. Вони дозволяють вивчати спектр сигналу (його частотні характеристики), визначати, які частоти у цьому сигналі є, а які – ні. Після цього можна зробити фільтрацію, посилити або послабити деякі частоти, або просто розпізнати звук певної висоти серед набору частот.

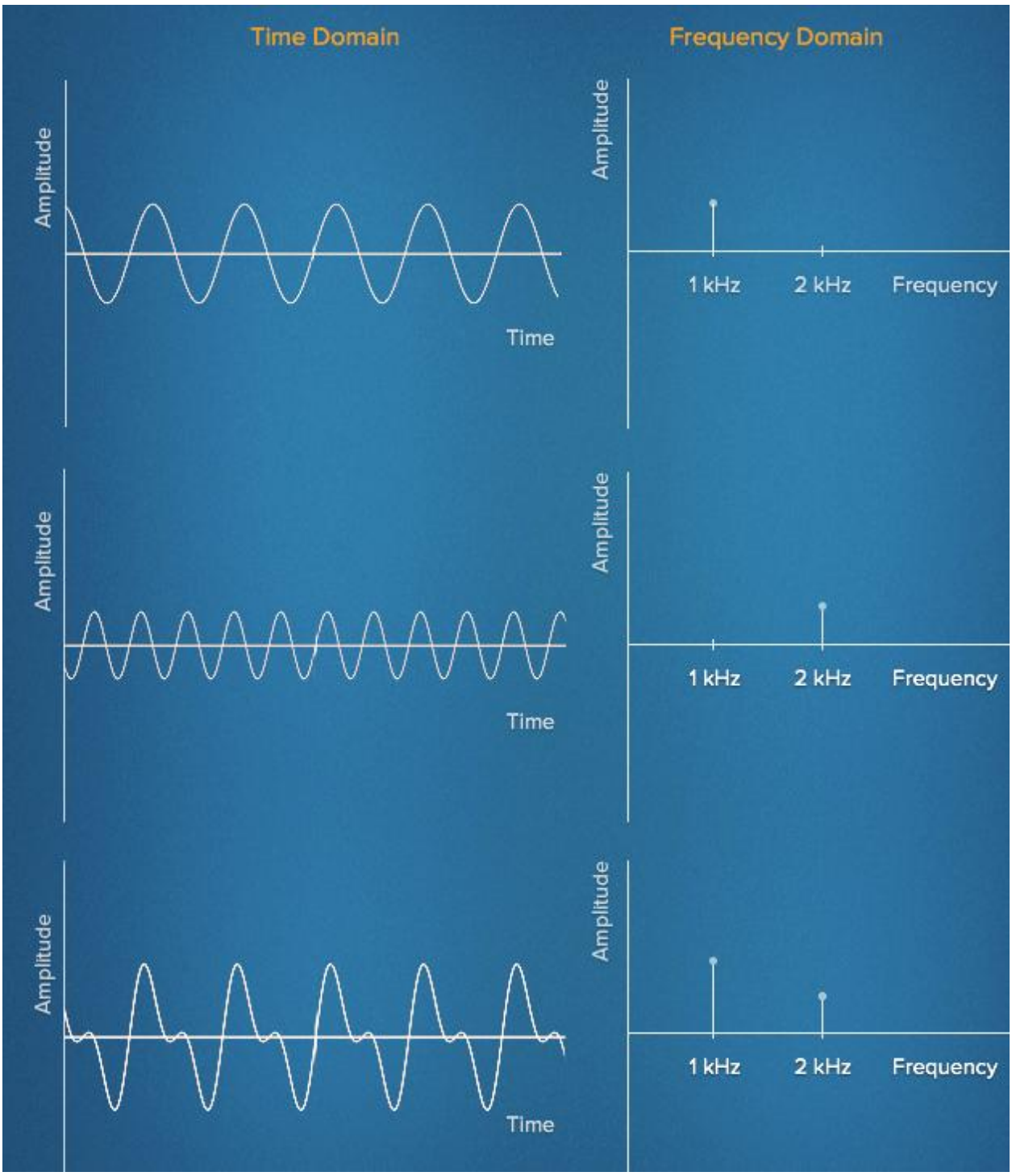


Рисунок 1.2 – Сигнали розгорнуті у часі та їх частотні характеристики

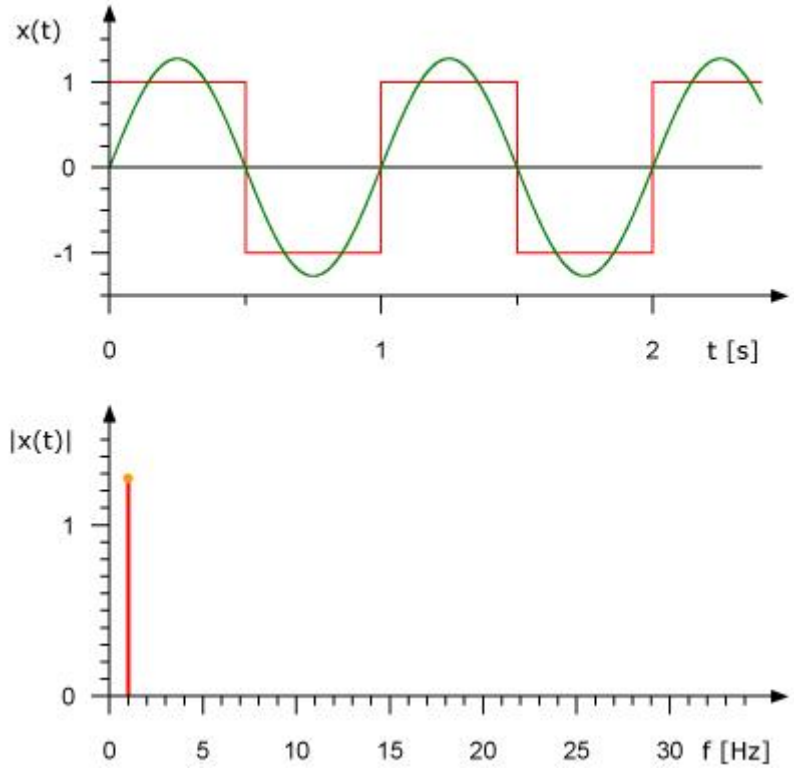


Рисунок 1.3 – Перетворення Фур’є у дії

1.3. Дискретне перетворення Фур’є

Отже, необхідно визначити спосіб отримання частотних показників сигналів, розгорнутих у часі. У цьому допоможе дискретне перетворення Фур'є (ДПФ, DFT, Discrete Fourier Transform). ДПФ - це математичний метод аналізу Фур'є для дискретних сигналів. З його допомогою можна перетворити кінцевий набір зразків сигналу, взятих з рівними проміжками часу, в список коефіцієнтів кінцевої комбінації комплексних синусоїд, упорядкованих за частотою, враховуючи, що ці синусоїди були дисретизовані з однією частотою [2].

Один із найпопулярніших чисельних алгоритмів для обчислення ДПФ називається швидке перетворення Фур'є (ШПФ, FFT, Fast Fourier Transformation). Насправді ШПФ представлений цілим набором алгоритмів**.** Серед них найчастіше використовуються варіанти алгоритму Кулі-Тьюкі (Cooley-Tukey). В основі цього алгоритму лежить принцип «поділяй і владарюй». У результаті обчислень використовується рекурсивне розкладання вихідного ДПФ на дрібні частини. Пряме обчислення ДПФ для деякого набору даних n вимагає O(n^2) операцій, а використання алгоритму Кулі-Тьюкі дозволяє вирішити ту ж задачу за O($n * \log(n)$) операцій.

Ось приклад функції обчислення ШПФ, написаної на C#. На її вхід подаються комплексні числа. Для того, щоб розібратися з взаємовідносинами між комплексними числами та тригонометричними функціями, корисно почитати про формулу Ейлера.

```
Complex[] Fft(Complex[] x)
{
    int N = x.Length;

    // fft парних елементів
    Complex[] even = new Complex[N / 2];
    for (int k = 0; k < N / 2; k++)
    {
        even[k] = x[2 * k];
    }

    Complex[] q = Fft(even);

    // fft непарних елементів
    Complex[] odd = even; // повторне використання масиву
    for (int k = 0; k < N / 2; k++)
    {
        odd[k] = x[2 * k + 1];
    }

    Complex[] r = Fft(odd);

    // комбінуємо
    Complex[] y = new Complex[N];
    for (int k = 0; k < N / 2; k++)
    {
        double kth = -2 * k * Math.PI / N;
        Complex wk = new Complex(Math.Cos(kth), Math.Sin(kth));
        y[k] = q[k] + wk * r[k];
        y[k + N / 2] = q[k] - wk * r[k];
    }

    return y;
}
```

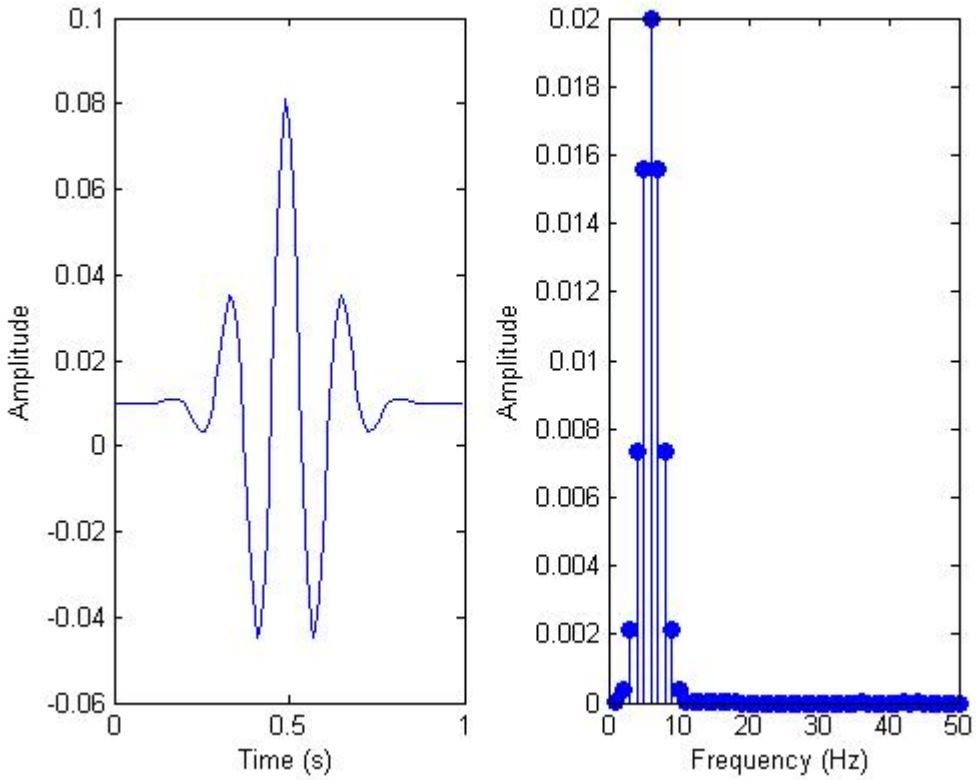


Рисунок 1.4 – Сигнал до і після ШПФ – аналізу

1.4. Сигнатури пісень

Один з неприємних побічних ефектів ШПФ полягає в тому, що, провівши аналіз, ми втрачаємо інформацію про час. (Хоча, теоретично, подібного можна уникнути, але на практиці для цього знадобиться величезна обчислювальна потужність.) Наприклад, для трихвилинної пісні ми можемо бачити звукові частоти та їх амплітуди, але де саме у творі ці частоти зустрічаються, не знаємо. А це – найважливіша характеристика, яка робить музичний твір тим, що він є. Нам потрібно якось дізнатися про точні значення часу, коли з'являється кожна з частот [3].

Саме тому ми будемо користуватися чимось на зразок ковзного вікна, або блоку даних, і трансформувати лише ту частину сигналу, яка в це «вікно» потрапляє. Розмір кожного блоку можна визначити за допомогою різних

підходів. Наприклад, якщо ми записуємо двоканальний звук із розміром зразка рівним 16 біт і з частотою дискретизації 44100 Гц, одна секунда такого звуку займе 176 Кб пам'яті (44100 зразків * 2 байти * 2 канали). Якщо ми встановимо розмір, що дорівнює 4 Кб, то кожну секунду нам потрібно буде проаналізувати 44 блоки даних. Це досить висока роздільна здатність для детального аналізу композиції.

```
string enteredData = Console.ReadLine();
byte[] byteArrayFromData = Encoding.UTF8.GetBytes(enteredData!);
byte[] audio = byteArrayFromData;
int totalSize = audio.Length;
int chunkSize = 8;
int sampledChunkSize = totalSize/chunkSize;
Complex[][] result = new Complex[sampledChunkSize][];

for(int j = 0; j < sampledChunkSize; j++) {
    var complexArray = new Complex[chunkSize];

    for(int i = 0; i < chunkSize; i++) {
        complexArray[i] = new Complex(audio[j*chunkSize+i], 0);
    }

    result[j] = FFT.fft(complexArray);
}
```

У внутрішньому циклі ми поміщаємо дані з тимчасової області (зразки звуку) в комплексні числа з уявною частиною, що дорівнює 0. У зовнішньому циклі проходимо по всіх блоках даних і для кожного з них запускаємо БПФ-аналіз.

Як тільки у нас будуть відомості про частотні характеристики сигналу, можна розпочинати формування цифрової сигнатури музичного твору. Це найважливіша частина всього процесу розпізнавання музики, який реалізує Shazam. Головна складність тут – вибрати з величезної кількості частот саме ті, які є найважливішими. Суто інтуїтивно ми звертаємо увагу на частоти з максимальними амплітудами (зазвичай їх називають піками).

Однак, в одній пісні діапазон "сильних" частот може змінюватись, скажімо, від ноти "до" контроктави (32,70 Гц), до ноти "до" п'ятої октави (4186,01 Гц). Це – величезний інтервал. Тому, замість того, щоб за відразу проаналізувати весь частотний діапазон, ми можемо вибрати кілька дрібніших інтервалів. Вибір можна зробити, ґрунтуючись на частотах, які

зазвичай притаманні важливим музичним компонентам, та проаналізувати їх окремо. Наприклад, можна скористатися інтервалами, які цей програміст використав для своєї реалізації алгоритму Shazam [18]. А саме, це 30 Гц – 40 Гц, 40 Гц – 80 Гц та 80 Гц – 120 Гц для низьких звуків (сюди потрапляє, наприклад, бас-гітара). Для середніх та більш високих звуків застосовуються частоти 120 Гц – 180 Гц та 180 Гц – 300 Гц (сюди входить вокал та більшість інших інструментів).

Тепер, коли ми визначилися з інтервалами, можна просто знайти частоти з найвищими рівнями. Ці відомості формують сигнатуру для конкретного аналізованого блоку даних, а вона, у свою чергу, є частиною сигнатури всієї пісні.

```
// Функція для визначення того, в якому діапазоні знаходиться частота
int GetIndex(int freq)
{
    int[] RANGE = new int[] {40, 80, 120, 180, 300};
    int i = 0;
    while (RANGE[i] < freq)
        i++;
    return i;
}

// Result - це комплексна матриця, отримана на попередньому
for (int t = 0; t < result.Length; t++)
{
    for (int freq = 40; freq < 300; freq++)
    {
        // Отримаємо силу сигналу:
        double mag = Math.Log(results[t][freq].abs() + 1);

        // Визначимо, в якому ми діапазоні:
        int index = GetIndex(freq);

        // Збережемо саме високе значення сили сигналу та відповідну частоту:
        if (mag > highscores[t][index])
        {
            points[t][index] = freq;
        }
    }

    // сформуємо хеш-тег
    long h = Hash(points[t][0], points[t][1], points[t][2], points[t][3]);
}
private static int FUZ_FACTOR = 2;

private static long Hash (long p1, long p2, long p3, long p4)
{
    return (p4 - (p4 % FUZ_FACTOR)) * 100000000 + (p3 - (p3 % FUZ_FACTOR))
           * 100000 + (p2 - (p2 % FUZ_FACTOR)) * 100
           + (p1 - (p1 % FUZ_FACTOR));
}
```

Зауважте, що ми повинні враховувати те, що запис виконано не в ідеальних умовах (тобто не в звукоізольованому приміщенні). Як результат, треба передбачити наявність у запису сторонніх шумів і можливе спотворення звуку, що записується, що залежить від характеристик приміщення. До цього питання варто підійти дуже серйозно, у реальних системах варто реалізувати налаштування аналізу можливих спотворень та сторонніх звуків (fuzz factor) залежно від умов, у яких проводиться запис [16].

Для спрощення пошуку музичних композицій їх сигнатури використовуються як ключі у хеш-таблиці. Ключам відповідають значення часу, коли набір частот, для яких знайдена сигнатура, з'явився у творі та ідентифікатор самого твору (назва пісні та ім'я виконавця, наприклад). Ось варіант того, як такі записи можуть виглядати в базі даних.

Хеш-тег	Час, в секундах	Пісня
30 51 99 121 195	53,52	Пісня А виконавця А
33 56 92 151 185	12,32	Пісня Б виконавця Б
39 26 89 141 251	15,34	Пісня В виконавця В
32 67 100 128 270	78,43	Пісня Г виконавця Г
30 51 99 121 195	10,89	Пісня Д виконавця Д
34 57 95 111 200	54,52	Пісня А виконавця А
34 41 93 161 202	11,89	Пісня Д виконавця Д

Якщо обробити в такий спосіб бібліотеку музичних записів, можна буде побудувати базу даних з повними сигнатурами кожного твору.

1.5 Пошук співпадінь

Для того, щоб з'ясувати, яка пісня грає зараз на радіо, треба записати звук за допомогою телефону і прогнати його через описаний процес обчислення сигнатур. Потім можна запустити пошук обчислених хеш-тегів у базі даних.

Але не все так просто. Річ у тім, що у багатьох фрагментів різних творів хеш-теги збігаються. Наприклад, може бути так, що якийсь фрагмент пісні А звучить так само, як якась ділянка пісні Д. І тут немає нічого дивного. Музиканти та композитори постійно «запозичують» один в одного вдалі музичні компоненти [4].

Щоразу, коли вдається виявити хеш-тег, що збігається, число можливих збігів зменшується, але дуже ймовірно, що тільки ці відомості не дозволять нам настільки звузити діапазон пошуку, щоб зупинитися на єдиній правильній пісні. Тому в алгоритмі розпізнавання музичних творів нам потрібно перевіряти ще дещо. А саме – йдеться про позначки часу.

Той фрагмент пісні, що записали в ресторані, може бути з будь-якого її місця, тому ми просто не в змозі безпосередньо порівнювати відносний час усередині записаного фрагмента з тим, що є в базі даних [15].

Однак, якщо знайдено кілька збігів, можна проаналізувати відносний таймінг збігів, і, таким чином, підвищити достовірність пошуку.

Наприклад, якщо поглянути в наведену вище таблицю, можна виявити, що хеш-тег ‘30 51 99 121 195’ відноситься і до пісні А, і до пісні Д. Якщо секундою ми будемо перевіряти хеш-тег ‘34 57 95 111 200’, то про збіг з піснею А, до того ж, у подібному випадку ми знатимемо про те, що збігаються і хеш-теги та їх розподіл у часі.

```
// Клас, котрий описує конкретний момент в творі
private class DataPoint {

    private readonly int _time;
    private readonly int _songId;

    public DataPoint (int songId, int time) {
        this._songId = songId;
        this._time = time;
    }

    public int GetTime() {
        return _time;
    }
    public int GetSongId() {
        return _songId;
    }
}
```

Нехай i_1 *та* i_2 – це позначки часу у записаній пісні, j_1 *та* j_2 – позначки часу у пісні з бази даних. Ми можемо говорити, що є два збіги, з урахуванням збігу різниці в часі, якщо виконується така умова:

$$RecordedHash(\boldsymbol{i_1}) = SongInDBHash(\boldsymbol{j_1})\ AND\ RecordedHash(\boldsymbol{i_2}) = SongInDBHash(\boldsymbol{j_2})\ AND\ abs(\boldsymbol{i_1} - i_2) = abs\ (\boldsymbol{j_1} - j_2)$$

Це дає можливість не дбати про те, на яку частину пісні доводиться запис: на початок, середину, або на самий кінець.

І, нарешті, малоймовірно, що кожен оброблений фрагмент записаної в «диких» умовах пісні співпаде з аналогічним фрагментом бази даних, побудованої на основі студійних записів. Запис, на основі якого ми хочемо знайти назву твору, буде включати багато шуму, що призведе до деяких розбіжностей при порівнянні. Тому замість того, щоб намагатися виключити зі списку збігів все, крім єдиної вірної композиції, в кінці процедури зіставлення з базою даних ми відсортуємо записи, в яких знайшлися збіги. Сортуватимемо у спадному порядку. Чим більше збігів – тим ймовірніше те, що ми знайшли необхідну композицію. Відповідно, вона опиниться на вершині списку.

РОЗДІЛ 2

АЛГОРИТМИ РОЗПІЗНАННЯ ЗВУКУ

2.1 Алгоритм пошуку

Принцип роботи пошукового алгоритму: після отримання хеш-таблиць оригіналів з бази даних та наспіваного фрагмента дані в кожній хеш-таблиці покриваються об'єктами класу «Target», які мають такі властивості:

1) мають розмір 400 Гц на 5 секунд;

2) мають властивість «target_property», що є асоціативним масивом, або хеш-таблицею, де зберігаються хеші, сформовані як $df : dt$, де df і dt – різниці частоти та часу між усіма точками об'єкта типу «Target» і точкою, що має найменшу частоту і часом відповідно (тобто точку, що має найменшу відстань до точки відліку в системі координат);

3) мають навантаження оператора «==». Для навантаження оператора «==» необхідно вибрати два коефіцієнти: допустиме відхилення dt для того, щоб вважати два хеші однаковими –k_1; необхідна кількість хешів, що збігаються, для того, щоб вважати два об'єкти типу «Target» однаковими - k_2.

Ступінь схожості записів визначається ставленням кількості знайдених рівних об'єктів до їхнього загального числа в краплинному записі.

Однак результати досліджень даного алгоритму, були все одно недостатньо хороші, навіть з урахуванням «покращення» за рахунок обрізки частотного діапазону до області людського голосу, тому необхідно було шукати інші шляхи поліпшення. Було помічено, що записи, де тембри голосів акапельного і оригінального виконавців сильно розрізнялися, мало розпізнавались [5].

Очевидно, ця особливість є одним із слабких місць цього алгоритму, тому оптимальніше було б побудувати алгоритм, який не залежить від частотного діапазону співака. При цьому очевидно, що якщо виконавець не

фальшивить, він виконуватиме ті ж ноти, що і в оригінальному записі, але в іншій октаві. Крім того, виконавець може співати швидше, або повільніше за оригінал, проте відношення тимчасових інтервалів має бути однаковим задля збереження правильного ритмічного малюнка композиції [6].

Так, було висунуто гіпотезу у тому, якщо транскрибувати оригінал і акапельну запис у музичну нотацію і проводити порівняння з них, то точність розпізнавання можливо буде підвищена.

2.2 Алгоритм розпізнання нот

Завдання розпізнавання нот, що стоять окремо, зводиться до двох підзавдань: безпосереднього розпізнавання ноти і розпізнавання нот у тимчасовому інтервалі, тобто відділенні однієї ноти від іншої.

Алгоритм розпізнавання окремої ноти, досить простий. У першому етапі виділяється спектр звуку з допомогою перетворення Фур'є (рисунок 2.1, а). Наступним кроком виділяються точки, підозрілі на локальні екстремуми (у разі цікавлять лише максимуми функції). Для цього реалізована функція, в якій для кожної точки, що досліджується, обчислюється різниця з двома її сусідами.

Якщо обидва ці значення позитивні, то список записується більше двох значень. Після цього вибирається перший екстремум, для якого значення списку більше певного, емпірично підібраного значення толерантності. Далі береться отримане значення частоти й у ньому визначається нота.

Щоб не зберігати весь список нот, а також щоб надалі мати можливість порівнювати одну і ту ж мелодію наспівану в різних діапазонах, тобто. не залежати від октави у якій виконана нота, вона розраховується щодо 0-ї октави, тобто субконтроктави. Тому в пам'яті зберігається тільки невеликий асоціативний масив на 7 нот: note_dict = {'E': 20.61, 'F': 21.82, 'G': 24.5, 'A': 27.5, 'B': 30.87, 'C': 32.7, 'D': 36.95}. Отримана на попередньому етапі частота

ділиться послідовно кожне з цих чисел без залишку. Щоб визначити, яка саме нота потрапила на вхід алгоритму, залишається лише визначити яке із значень після розподілу є ступенем двійки. Для цього достатньо обчислити результат побітової операції «ТА» для отриманого значення і значення, на одиницю меншого, ніж отримане. Результат застосування оператора & дорівнюватиме 0 тільки для ступеня двійки та нуля [7].

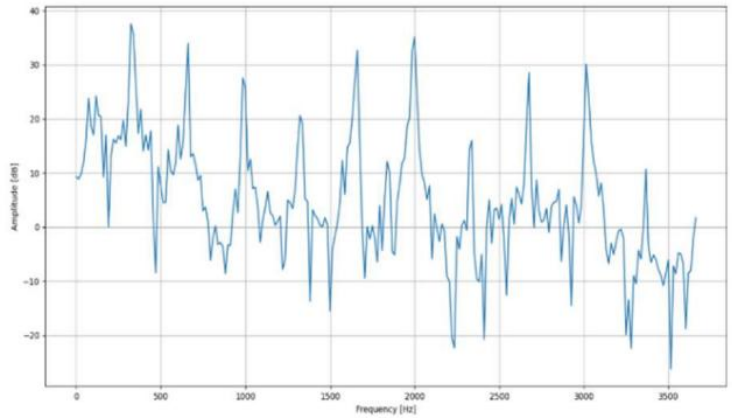
Так само, найімовірніше, для різних інструментів через особливості їх конструкцій та звучання необхідно підбирати віконну функцію для виділення спектра. Наприклад, для гітари такою функцією виявилася функція Хана:

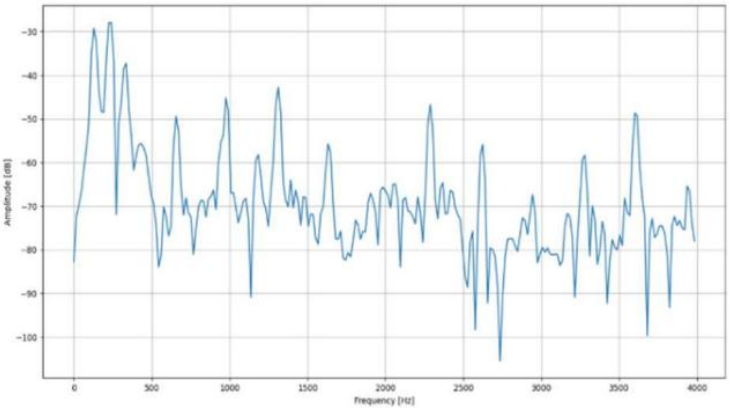
$$w[n] = w_0\left(\frac{L}{N}\left(n - \frac{N}{2}\right)\right) = \frac{1}{2}[1 - \cos\frac{2\pi n}{N}] = \sin^2\frac{\pi n}{N}, 0 \le n \le N$$

На малюнку 2 (рис. 2.1, б) видно, який ефект привносить використання функції Хана порівняно зі спектром, отриманим без використання віконної функції.

Цей механізм можна також застосовувати для розпізнавання декількох різних нот, що йдуть підряд.

Однак у цьому випадку вже просто виділити спектр сигналу на всьому аудіофайлі не вдасться. Щоб зрозуміти, як вирішити цю проблему необхідно було розглянути амплітудно-часову залежність, яку нескладно отримати, оскільки спочатку аудіофайл зчитується на вході в алгоритм функцією, яка зберігає дані саме в такому форматі [8]. Неважко зрозуміти, що для відокремлення однієї ноти від іншої необхідно просто виділити сигнал серед шуму. Для простоти реалізації було прийнято, що сигнал можна назвати корисним, якщо він гучніший за середній по аудіофайлу протягом чверті секунди. Однак ці наближення потребують корекції та ретельнішого підбору при аналізі «живих даних».





а *б*

Рисунок 2.1 Спектр ноти E4 без застосування віконної функції (*а*) та з використанням функції Хана для виділення (*б*)

2.3 Алгоритм розпізнання акордів

При написанні музики використовуються не тільки одиночні ноти, а й акорди, що складаються одночасно з кількох нот, що сприймаються людиною як єдиний звук [9].

Звук, який можна порівняти з певним акордом, є сумою гармонік. Насправді кожна наступна гармоніка звучить тихіше попередньої і гірше помітна. Частота звуку тієї чи іншої ноти визначається її основною гармонікою.

Більшість сучасних алгоритмів транскрипції акордів складаються з двох основних етапів.

На першому етапі аудіозапис перетворюється на послідовність ознак векторів. На наступному етапі, використовуючи шаблони, кожен вектор

ознак ставиться у відповідність кожному акорду із заданого простору акордів. Існує безліч методик зіставлення із зразків на основі шаблонів, прихованих моделей Маркова або складніших Байєсовських мереж. Ідея цього підходу у тому, щоб заздалегідь підготувати набір шаблонів, відповідних набору акордів у ряді. Інтуїтивно кожен шаблон у наборі – це прототип вектору кольору, який відповідає одному акорду. На вхід алгоритму подається звукова доріжка, у якій, як і алгоритмі «Shazam», виділяються екстремальні точки [10]. Однак для розпізнавання не беруться всі точки, що потрапили до карти сузір'їв, як у попередньому алгоритмі. До розпізнавання беруться ті частоти, які коливаються навколо одного значення деякий проміжок часу. Кожній такій частоті ставиться відповідно до шаблону нота.

Алгоритми шаблонного розпізнавання засновані на зіставленні конкретного акорду із заданого акордного простору з вхідним вектором особливостей за допомогою мінімізації відстані між векторами. Таке завдання можна вирішувати за допомогою алгоритмів кластеризації.

При розпізнаванні нот і акордів дуже важлива точність значень частот, що одержуються, так як один акорд може складатися з нот з невеликою різницею в частотах. Тому для реалізації нового алгоритму було прийнято рішення написати виділення спектра, ґрунтуючись на швидкому перетворенні Фур'є, з можливістю самостійно задавати кількість компонентів розкладання та отримувати при цьому можливість досліджувати вплив застосування різних віконних функцій [11].

Результуюча частота звуку розраховується як середня частота нот. На рисунках 3, 4 можна побачити результуючу функцію суми періодичних функцій, що становлять основні гармоніки нот акорду С.

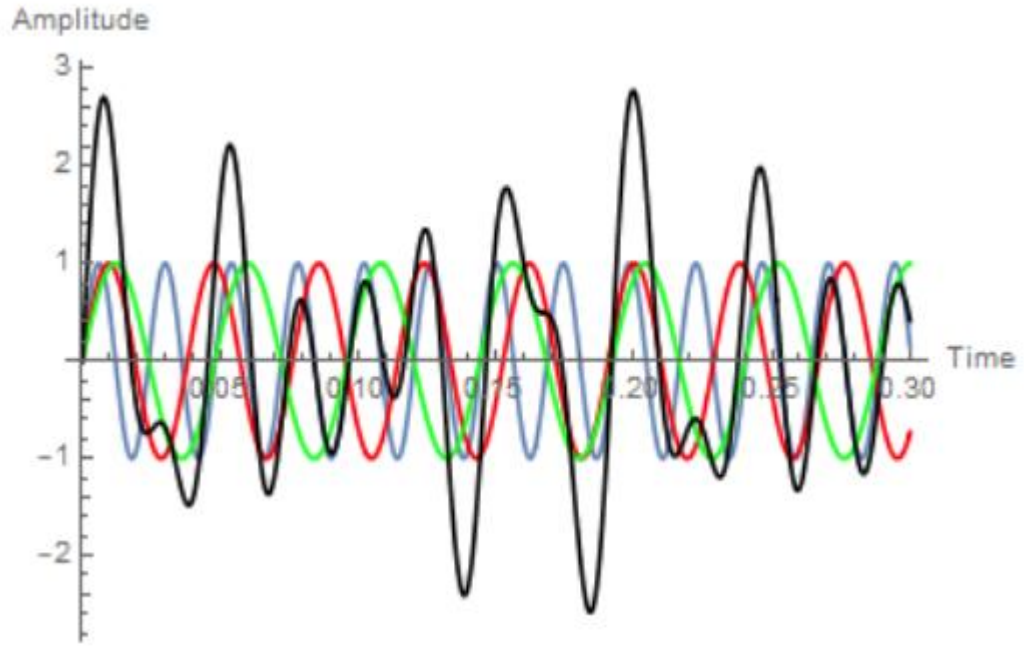


Рисунок 2.2 Графік періодичних функцій основних гармонік нот акорду С та їх суми

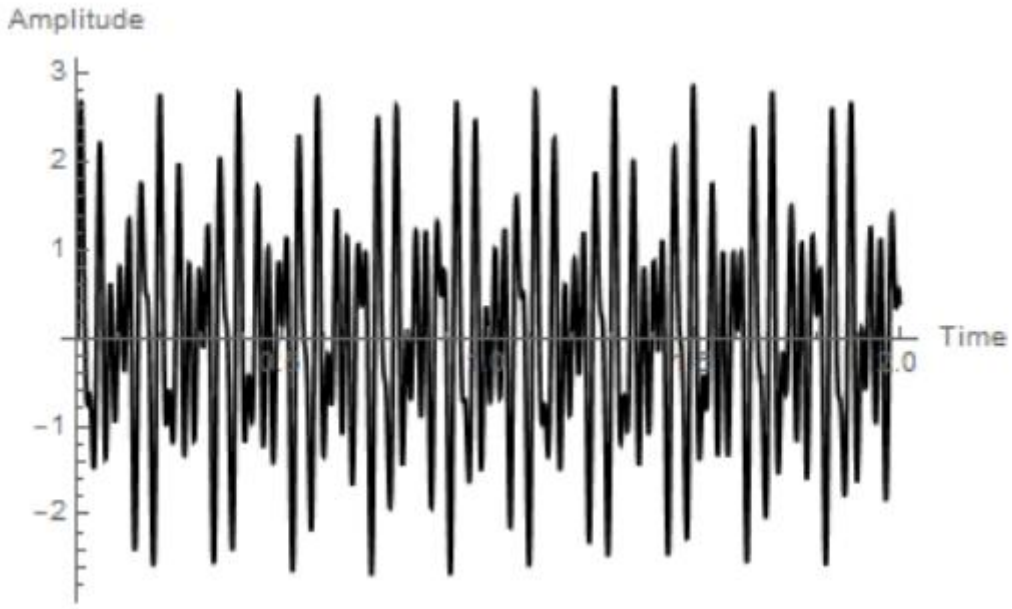


Рисунок 2.3 Графік суми основних гармонік нот акорду С

Із цього випливає, що при реалізації алгоритму не вдасться виділяти три ноти окремо і за ними розпізнавати акорд. Вирішенням цієї проблеми є створення бази ознак акордів або розпізнавання акорду з його результуючої частоти.

Насправді виявилося, що ці методи застосовні. Дійсно, іноді в акорді вдається виділити результуючу частоту, проте даний метод нестабільний і схильний до впливів шумів, а також на його результативність сильно впливає обрана кількість компонентів розкладання Фур'є.

Іншим ефективним методом є створення шаблонів характеристик акордів, що цікавлять. Способів виділення векторів особливостей є дуже багато.

Якщо для розпізнавання окремої ноти досить виділити одну першу основну гармоніку, тобто глобальний екстремум, то для розпізнавання акордів необхідно було виділити 6–10 локальних екстремумів спектра [12].

Однак ці точки також не можуть бути гарантовано такими, як і у векторі властивостей. Інструмент, на якому виконаний акорд, може посилювати ті чи інші частоти в залежності від властивостей його резонаторів, а також свій внесок може робити записувальне обладнання. Крім того, сам виконавець може зіграти акорд недостатньо чисто. Тому з наявних еталонних векторів виділяються найбільш схожі компоненти і щодо них обчислюється відстань до досліджуваного вектора.

Необхідно відзначити, що даний алгоритм є стійким до зміщення діапазону виконання, оскільки спирається не на конкретні приватні значення частот, а на Евклідову метрику [13].

РОЗДІЛ 3. РОЗРОБКА

3.1 Підготовка даних

За вхідні данні візьмемо зразок аудіо тривалістю 10 секунд записаний на мікрофон у програмі. Це буде живий приклад, оскільки запис буде зроблений з шумами навколишнього середовища (ехо, вітер, шум ноутбуку). Як базу даних пісень будемо використовувати папку з музикою, кількість яких перевищує 500 штук. Для виведення звуку з мікрофону та запис його у .wav файл, використаємо бібліотеку мови програмування C# під назвою “NAudio”.

Створимо пустий .wav, в який збережемо запис з мікрофону. Задаємо формат файлу, в який запишемо десяти та п’яти секундний запис з мікрофону. Також визначимо подію, яка буде відслідковувати чи записаний зміст нашої промови до файлу. Також додамо опцію «стоп», що мати можливість зупинити запис. Мовою C# це буде виглядати так:

```
waveSource.WaveFormat = new
NAudio.Wave.WaveFormat(44100, 1);
waveSource.DataAvailable += WaveSource_DataAvailable);
waveFile = new
WaveFileWriter(@"C:\Users\Vlad\Desktop\test\test.wav",
waveSource.WaveFormat);
waveSource.StartRecording();
Console.WriteLine("Press enter to stop");
Console.ReadLine();
waveSource.StopRecording();
waveFile.Dispose();
```

3.2 Підбір алгоритму

В роботі було досліджено 4 алгоритми пошуку аудіофайлу. Для досліду результатів програми, ми використаємо бібліотеку “Soundfingerprinting”, яка використовує суміш алгоритмів ШПФ та мінімального хешування «відбитку файлу». Було вибрано саме такий алгоритм для того, щоб дослідити чому він не такий популярний, як, наприклад, чистий алгоритм швидкого перетворення Фур’є.

На минулому кроці ми записали вхідні файли у тестовий файл. Зараз отримаємо хеш, тобто «відбиток» зразку для того, щоб порівняти його з кожним аудіофайлом у нашій папці. Напишемо це мовою C#:

```
string path = waveFile.Filename;
var fingerprints :AVHashes = await FingerprintCommandBuilder.Instance // IFingerprintCommandBuil
    .BuildFingerprintCommand() // ISourceFrom
    .From(path) // IWithFingerprintConfiguration
    .WithFingerprintConfig( amendFunctor: config :AVFingerprintConfiguration =>
    {
        // встановимо крок між послідовними "відбитками" до 1_024 зразків
        config.Audio.Stride = new IncrementalStaticStride( incrementBy: 1_024);
        return config;
    }) // IUsingFingerprintServices
    .Hash(); // Task<AVHashes>

Console.WriteLine($"Згенеровано {fingerprints.Count} хешів");
if (fingerprints.Audio != null)
    foreach (var hashes :HashedFingerprint in fingerprints.Audio)
    {
        foreach (var hash :int in hashes.HashBins)
        {
            Console.WriteLine($"Початок хешу: {hashes.StartsAt}, значення: {hash}");
        }
    }
```

3.3 Реалізація та отримання результатів

Отримаємо усі файли формату .wav з нашої папки. Для цього задамо змінну, яка зберігає в собі шлях до нашого музичного каталогу. Потім запишемо в іншу змінну шляхи до всіх .wav файлів.

```
var pathToDb = @"C:\Users\Vlad\Desktop\test";
var listOfFiles = Directory.GetFiles(pathToDb).Where(x
=> x.EndsWith(".wav"));
```

Тепер для отримання кожного «відбитка» треку, пройдемось по кожному з файлів через цикл і добавимо кожен відбиток у сховище, яке буде зберігати назву треку та його хеш. Сховище буде працювати за рахунок оперативної пам’яті.

```
foreach (var file in listOfFiles)
{
    var fingerprint = await
FingerprintCommandBuilder.Instance
        .BuildFingerprintCommand()
        .From(file)
        .WithFingerprintConfig(config =>
        {
            config.Audio.Stride = new
IncrementalStaticStride(1_024);
            return config;
        })
        .Hash();
    var track = new TrackInfo($"{id}", file.Name,
file.Artist);
    modelService.Insert(track, fingerprint);
    ++id;
}
```

Отже, на вхід програми я вибрав 10 – секундний шматок пісні Bon Jovi – It’s My Life, результати програми з кроком у 1024 зразки:

```
Програма виконана за 00:00:01.2323900. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.2345019. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.2422166. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.2170156. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.2111396. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.2823979. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.2754201. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.2257088. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.2154326. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.2201342. Результат: It's My Life - Bon Jovi
```

Маємо, що програма приблизно за 1 секунду знаходила трек серед 500 файлів.

Також був даний на вхід запис тривалістю 5 секунд того самого треку, результат:

```
Програма виконана за 00:00:01.2460140. Результат: не знайдено
Програма виконана за 00:00:01.2817121. Результат: не знайдено
Програма виконана за 00:00:01.2171272. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.2145804. Результат: не знайдено
Програма виконана за 00:00:01.2359781. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.2638207. Результат: не знайдено
Програма виконана за 00:00:01.2090699. Результат: не знайдено
Програма виконана за 00:00:01.2375178. Результат: не знайдено
Програма виконана за 00:00:01.2260104. Результат: не знайдено
Програма виконана за 00:00:01.2379359. Результат: It's My Life - Bon Jovi
```

З 10 спроб, трек було знайдено 3 рази.

Задля експерименту, зменшимо крок між послідовними відбитками з 1024 до 512.

Результат за 10 секунд:

```
Програма виконана за 00:00:01.9260591. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.0900948. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.0976534. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.9001906. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.9283923. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.9794376. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.0673659. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.0097575. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.9674222. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.0162797. Результат: It's My Life - Bon Jovi
```

Результат за 5 секунд:

```
Програма виконана за 00:00:02.0958609. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.0179137. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.9303758. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.9288920. Результат: не знайдено
Програма виконана за 00:00:01.9538479. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.0759594. Результат: не знайдено
Програма виконана за 00:00:01.9508137. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:01.9928219. Результат: не знайдено
Програма виконана за 00:00:02.0484311. Результат: не знайдено
Програма виконана за 00:00:02.0688147. Результат: It's My Life - Bon Jovi
```

Ще раз зменшимо крок до 256.

Результат за 10 секунд:

```
Програма виконана за 00:00:02.4558497. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4714051. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4537026. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4830959. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4113325. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4615225. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4370919. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4162984. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4203084. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4729795. Результат: It's My Life - Bon Jovi
```

Результат за 5 секунд:

```
Програма виконана за 00:00:02.4877725. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4551938. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4756924. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4154754. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4984645. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4528444. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4992350. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4342815. Результат: It's My Life - Bon Jovi
Програма виконана за 00:00:02.4424737. Результат: не знайдено
Програма виконана за 00:00:02.4710496. Результат: It's My Life - Bon Jovi
```

3.4 Висновки щодо результатів

Результати з усіма кроками на 10 секунд виявили трек. Але з записами на 5 секунд ситуація цікавіша. З кроком у 1024, ми маємо 3 знайдених треки з 10, з кроком у 512 – 6, з кроком 256 – 9. З цього випливає, що при зменшенні кроку у 2 рази між послідовними «відбитками» аудіозапису, ми збільшуємо точність співпадінь. Але з іншого боку, програма витрачає більше часу для того, щоб зробити більше «відбитків», по яким потім знаходити схожий трек.

ВИСНОВКИ

В роботі викладено підхід розпізнання аудіофайлів на основі різних алгоритмів, які допомагають створити мелодійний контур з використанням музики. Також була розроблена програма, яка дозволяє розпізнавати пісні та можливістю використання в майбутньому у різних сервісах. Було здійснено опис моделі розпізнання медіафайлу, де поетапно викладено кожен крок для отримання бажаного результату.

За наявності певних вхідних даних, що були зібрані в процесі дослідження, проведено експерименти, що демонструють процес дослідження та частково засвідчують ефективність викладеного підходу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Фролов, А. В. Синтез и распознавание речи. Современные решения. [Електронний ресурс] http://www.frolov-lib.ru/books/hi/ch01.html
2. Мещеряков, Р. В. Структура систем синтеза и распознавания речи. // Известия Томского политехнического университета. Т.315, №5. – 2009. – С. 121-126.
3. ISO/IEC FDIS 15938-4:2001(E) Information Technology - Multimedia Content Description Interface – Part 4: Audio, 2001
4. Julius Tutorial [Електронний ресурс] URL: https://julius.osdn.jp/en_index.php
5. Neuros Audio web site [Електронний ресурс] URL: http://www.neurosaudio.com/
6. Розуміння алгоритму ШПФ. [Електронний ресурс] URL: https://habr.com/ru/company/otus/blog/449996/
7. A large set of audio features for sound description – G. Peeters. [Електронний ресурс] URL: http://recherche.ircam.fr/anasyn/peeters/ARTICLES/Peeters_2003_cuidadoaudiofeatures.pdf
8. An Industrial-Strength Audio Search Algorithm. [Електронний ресурс] URL: https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf
9. Jaap Haitsma, Antonius Kalker, “A Highly Robust Audio Fingerprinting System”, International Symposium on Music Information Retrieval (ISMIR) 2002, pp. 107-115.
10. Audible Magic web site [Електронний ресурс] URL: http://www.audiblemagic.com/
11. Audio Fingerprinting- Understanding the Concept, Process & Application [Електронний ресурс] URL: https://www.pathpartnertech.com/audio-fingerprinting-understanding-the-concept-process

12. How the Web Audio API is used for audio fingerprinting [Електронний ресурс] URL: https://fingerprintjs.com/blog/audio-fingerprinting/
13. Audio Fingerprinting: Concepts and Applications [Електронний ресурс] URL: https://www.researchgate.net/publication/225574479_Audio_Fingerprinting_Concepts_And_Applications
14. Stratonovich, R.L. Conditional Markov Processes / Stratonovich, R.L. // Theory of Probability and its Applications – 1960. – No. 5 (2). – pp. 156–178.
15. Audio Fingerprinting with Python and Numpy [Електронний ресурс] – Режим доступу до ресурсу: https://willdrevo.com/fingerprinting-and-audiorecognition-with-python/
16. Kim Hyoung-Gook,Moreau Nicolas,Thomas Sikora, MPEG-7 Audio and Beyond: Audio Content Indexing and Retrieval, JohnWiley & Sons, 2005.
17. Звукові хвилі [Електронний ресурс] – Режим доступу до ресурсу: https://sites.google.com/site/kolivannaihvilijk/home/zvukovi-hvili.
18. K. Kondo, “Method of changing tempo and pitch of audio by digital signal processing.” Google Patents, 1999.