Transilvania **University of Braşov**
Faculty of Mathematics and Computer Science
Department of Mathematics and Computer Science

Song Recognition

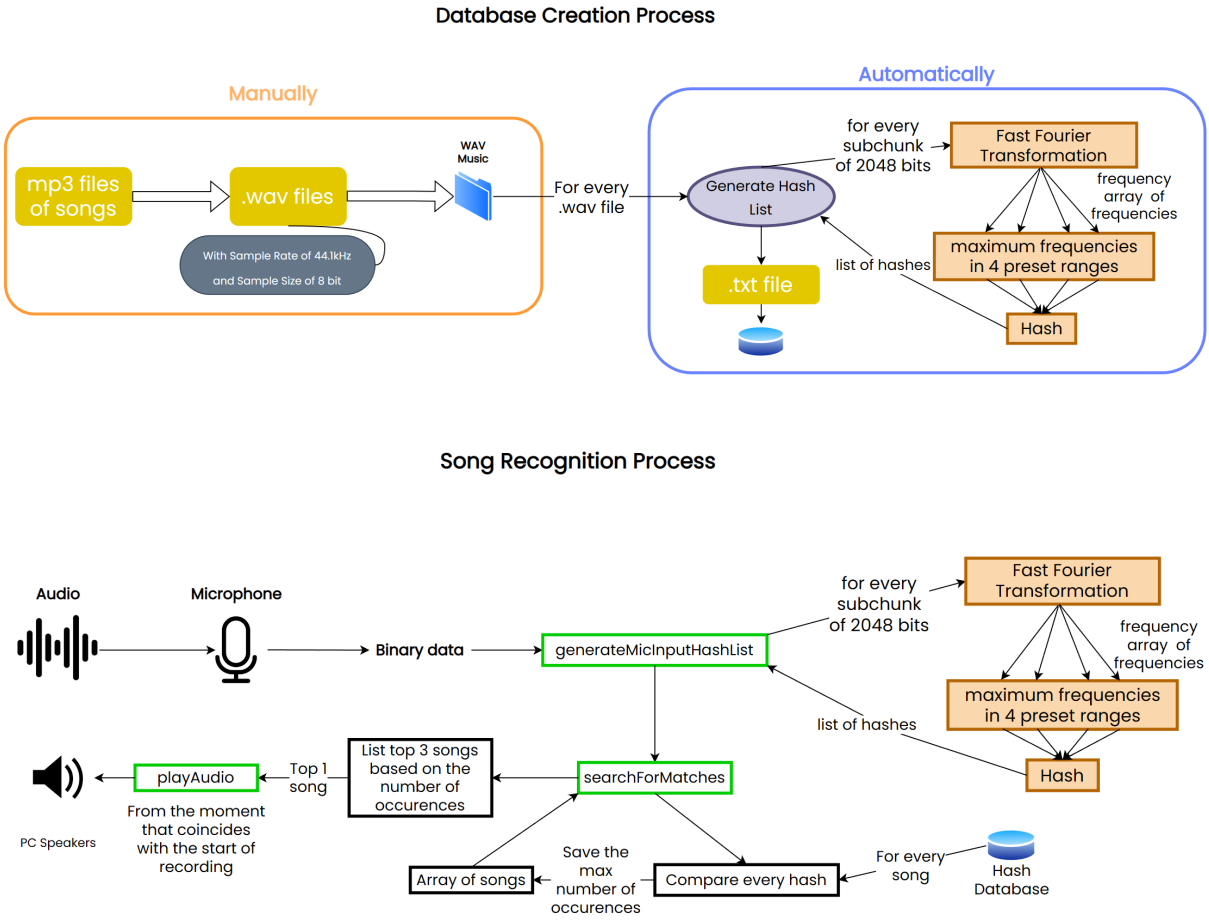
Authors: Rozmarin Andrei, Stănică Dragoş

Contents

1 Introduction

The aim of this application is to help the user find a matching song present in our database based on a given recording.

The app is structured in 2 main parts: the main code and the database generation. Both of these also use a header file containing utility functions.



2 Usage

The user should start playing the song before running the app, as to ensure that the recording doesn't have any blank spots. After this, processing and matching can be controlled using the ESC key to jump from one section to the next. If he wishes to add a song to the database he should add a WAV file with the wave format used in the main program and then run the "Song Hashes Generator" app in order to generate its fingerprint. For optimal results, the recording should be at least 30 seconds long (see the pictures below).

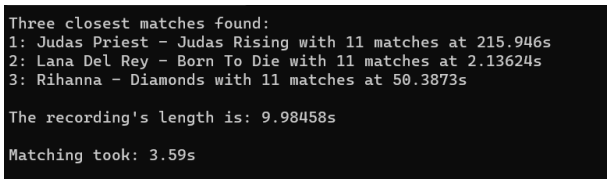


Figure 1: 10 second recording - the found match is incorrect

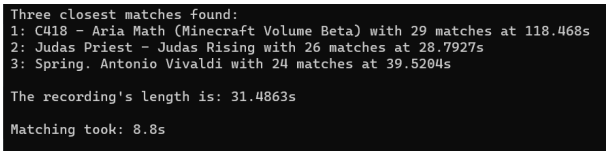


Figure 2: 30 second recording - the song was succesfully found

3 Main Code

This is the code responsible for recording the microphone input, processing and fingerprinting the audio file, searching for matches in the database and also playing the found part if neccessary.

3.1 Microphone Input

3.1.1 getMicInput

This is the function responsible for recording the microphone input and storing it in the form of bytes in a .bin file. Its type is void and it takes as parameter the format in which the user wishes to record, of type WAVEFORMATEX[1]. As a link between the microphone and app we use an object of type HWAVEIN in which we will store information regarding the recording source and format with the help of the waveInOpen function[2]. The actual data will be stored temporarily inside of a WAVEHDR[3] header before being flushed in the target file. This variable also requires a buffer of type char[] and a length which depends on the wave format. The header and HWAVEIN object are then linked with the help of the waveInPrepareHeader[4] and waveInAddBuffer[5] functions.

Once an empty file for storing the data is created, the recording starts after calling the waveInStart[6] function. While the user hasn't hit the ESC key, the function checks if the buffer has been filled and if yes, its contents are flushed in the file and the header is reset and relinked. Once the ESC key is hit, the waveInStop[7] function is called in order to stop and then the header is deactivated with the functions waveInUnprepareHeader[8] and waveInClose[9].

3.2 Generating the recording's fingerprint

3.2.1 generateMicInputHashList

Creates a file containing the recording's hash list. The function doesn't have any parameters and doesn't return anything.

After reading the data from the "recording.bin" file and casting it to the float type, the function separates the vector into subvectors, each containing a chunk of the recording. Following this, the Fast Fourier Transform is applied on every subvector, turning them into frequency subvectors. The maximum frequencies in the ranges of 30-40, 40-80, 80-120 and 120-180 Hz are picked and are then used in the getHash function in order to generate the chunk's hash. The resulting list represents the recording's hash list or "fingerprint" and is subsequently stored inside of a .txt file.

3.2.2 readFile

Returns a std::vector with char elements and receives as paramater the name of the file. After finding the end of the file and computing its size the function casts the bytes to char and dumps them into a std::vector, after which the vector is returned.

3.3 Searching for matches

3.3.1 Song structure

Contains information regarding the songs in the database.

It has 4 fields: the file path of the song as a std::string, the point in time where it was found as an int, the song name as a std::string and the number of hash matches with the recording as an int.

3.3.2 HashDistance

Returns the distance between two hashes. Its returned type is int and its parameters consist of 2 hashes of type long long int. The frequencies are extracted from the hashes and the sum of the absolute values of the differences of corresponding frequencies is returned.

3.3.3 searchForMatches

This function returns a std::vector containing the songs from the music database in descending order of the number of matches. It has no additional parameters.

After storing the recording hash list in a std::vector the function iterates through all the files in the music hash list database (the database is generated beforehand in the same way as the recording) and then searches the closest match to the recording based on the hashDistance function. After each iteration, a Song type object is generated with the specifics of the current song and said object is added to a std::vector.

After all songs have been processed, the Song vector is sorted in descending order based on the number of matches with the recording and is then returned.

3.4 Playing the match

3.4.1 playAudio

This function plays a song from a specific starting point and for a specified duration. Its type is void and it has 3 parameters: a std::string file path to the song, the starting point and the duration

After allocating memory for the blocks the function creates a handle for the file of type HANDLE[10] and attempts to link to the output device with the waveOutOpen[11]. If successful, the time frame is established and the function start calling the audioWrite function while there is still data to be processed.

After the data is finished playing the function unprepares the handles with waveOutUnprepareHeader[12], waveOutClose[13] and CloseHandle[14] and frees memory with HeapFree and DeleteCriticalSection[15].

3.4.2 allocateBlocks

This function returns a pointer to WAVEHDR variable, which will be used to store the data about to be played.

Its type is WAVEHDR* and has 2 parameters, those being the size of a memory block and number of block to be allocated to the buffer. After allocating the memory to the buffer using HeapAlloc[16], each block of the WAVEHDR dynamic array receives a pointer to the corresponding section in the buffer and then the dynamic array is returned.

3.4.3 freeBlocks

This function frees the memory of a WAVEHDR dynamic array using HeapFree[17]. Its type is void and has 1 parameter, the memory block that has to be freed.

3.4.4 writeAudio

This function iterates through a memory block and if the block is in the specified time frame, it plays the data through the default audio device. The function has the return type void and has 6 parameters: the output device handle, the data to be processed, the size of the data, the current time in the song, the time frame which needs to be played and the size in seconds of a block.

While there is still data to be processed, the output device is unprepared and prepared with the functions waveOutUnprepareHeader[12] and waveOutPrepareHeader[18], after which the data is copied into a buffer and sent to the audio device using waveOutWrite[19]. The calls caused by opening and closing the device are ignored with EnterCriticalSection[20] and LeaveCriticalSection[21]. When a data block is freed up the function moves on to the next memory block.

4 Utility Functions

These are functions which are frequently used throughout the app and that have been organized in a header.

4.1 setWaveFormat

This function sets the format of a WAVEFORMATEX type object to the desired values. Its type is void and has 4 parameters: the target object, the numbers of channels, the number of samples per second and the number of bits per sample.

4.2 fft

This is a function which performs a Fourier Transform on a given vector of amplitudes. Its return type is void and receives 2 parameters: a std::vector of std::complex elements representing the amplitude array to be processed and a boolean that decides whether or not to invert the angle of the complex numbers. The algorithm used is the Cooley-Tukey algorithm[22] for computing the Fast Fourier Transform and the implementation is based on the following code[23].

4.3 getHash

This function returns a hash based on 4 values. Its return type is long long int and it has 4 parameters representing the main frequencies of song section.

4.4 getIndex

The function returns a position in the RANGE global array referring to the interval of a frequency. Its return type is int and it has 1 parameter representing the frequency to be searched.

5 Database generation

This is kept separate from the rest of the code since its meant to be used only once to generate the fingerprint of each song in the music folder. The method is the exact same as the one used to generate the fingerprint of the recording with the only difference being that the first 44 bytes have to be skipped, since they are the header of the .wav file, containing various information about it.

6 Development

The project was initially inspired by Shazam with the original draft loosely following the old algorithm used by Shazam in order to find the song [24], [25].

The first truly big problem encountered during development was with comparing the hashes. Since the hashes of the songs and of the recording are almost never equal due to background noise or quality compression, we needed a way to see how much 2 hashes were alike. The first working iteration used the Jaro-Wrinkler algorithm, which gives a percentage based on how much 2 strings resemble one another. The precision was extremely high, with the app being able to accurately pinpoint songs from 10 second recordings or shorter, however, the speed suffered a lot due to how long the comparison took. It would take the app a good couple of minutes to process less than 10 songs. After this we switched to a direct comparison of hashes by extracting the main frequencies from each hash and computing the absolute value of the differences of the frequencies. This resulted in a massive speed increase, with approximately 30 songs taking just a couple seconds to be processed. In exchange for this, the accuracy of the matches was lowered by quite a lot, requiring the recording size to rise to around 40 seconds for a reliable match. We decided that overall, the longer recordings were well worth the speed improvements and decided to roll with the second version.

After this we also decided to add an auditory confirmation for the found match and a timer to see how long it took the app to run. The confirmation was made in the form of playing the closest song from the moment at which the match was found.

References

[1] *WAVEFORMATEX structure*. https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/ns-mmeapi-waveformatex. [Last accessed 31.01.2023].

[2] *waveInOpen function*. https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/nf-mmeapi-waveinopen. [Last accessed 31.01.2023].

[3] *WAVEHDR structure*. https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/ns-mmeapi-wavehdr. [Last accessed 31.01.2023].

[4] *waveInPrepareHeader function*. https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/nf-mmeapi-waveinprepareheader. [Last accessed 31.01.2023].

[5] *waveInAddBuffer function*. https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/nf-mmeapi-waveinaddbuffer. [Last accessed 31.01.2023].

[6] *waveInStart function*. https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/nf-mmeapi-waveinstart. [Last accessed 31.01.2023].

[7] *waveInStop function*. https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/nf-mmeapi-waveinstop. [Last accessed 31.01.2023].

[8] *waveInUnprepareHeader function*. https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/nf-mmeapi-waveinunprepareheader. [Last accessed 31.01.2023].

[9] *waveInClose function*. https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/nf-mmeapi-waveinclose. [Last accessed 31.01.2023].

[10] *handleapi.h header*. https://learn.microsoft.com/en-us/windows/win32/api/handleapi/. [Last accessed 31.01.2023].

[11] *waveOutOpen function*. https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/nf-mmeapi-waveoutopen. [Last accessed 31.01.2023].

[12] *waveOutUnprepareHeader function*. https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/nf-mmeapi-waveoutunprepareheader. [Last accessed 31.01.2023].

[13] *waveOutClose function*. https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/nf-mmeapi-waveoutclose. [Last accessed 31.01.2023].

[14] *CloseHandle function*. https://learn.microsoft.com/en-us/windows/win32/api/handleapi/nf-handleapi-closehandle. [Last accessed 31.01.2023].

[15] *DeleteCriticalSection function*. https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-deletecriticalsection. [Last accessed 31.01.2023].

[16] *HeapAlloc function*. https://learn.microsoft.com/en-us/windows/win32/api/heapapi/nf-heapapi-heapalloc. [Last accessed 31.01.2023].

[17] *HeapFree function*. https://learn.microsoft.com/en-us/windows/win32/api/heapapi/nf-heapapi-heapfree. [Last accessed 31.01.2023].

[18] *waveOutPrepareHeader function*. https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/nf-mmeapi-waveoutprepareheader. [Last accessed 31.01.2023].

[19] *waveOutWrite function*. https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/nf-mmeapi-waveoutwrite. [Last accessed 31.01.2023].

[20] *EnterCriticalSection function*. https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-entercriticalsection. [Last accessed 31.01.2023].

[21] *LeaveCriticalSection function*. https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-leavecriticalsection. [Last accessed 31.01.2023].

[22] James W. Cooley and John W. Tukey. *An Algorithm for the Machine Calculation of Complex Fourier Series*. https://www.eit.lth.se/fileadmin/eit/courses/eit085f/Cooley_Tukey_An_Algorithm_for_the_Machine_Calculation_of_Complex_Fourier_Series_Math_of_Comp_1965.pdf. American Mathematical Society, 1965.

[23] *Fast Fourier transform*. https://cp-algorithms.com/algebra/fft.html. [Last accessed 31.01.2023].

[24] Avery Li-Chun Wang. *An Industrial-Strength Audio Search Algorithm*. https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf. AShazam Entertainment, Ltd.

[25] *How Shazam Works*. https://laplacian.wordpress.com/2009/01/10/how-shazam-works/. [Last accessed 31.01.2023].