České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Pavel Štíbal**

Studijní program: Otevřená informatika
Obor: Softwarové systémy

Název tématu: **Změna délky hudby**

Pokyny pro vypracování:

Prozkoumejte problematiku "inteligentní" změny délky skladeb (music retargeting), kde se z existující skladby (audionahrávky) vytvoří její varianta, kdy bez změny tempa a vnímané výšky tónu uvnitř nahrávky se změní délka celkové sklaby vynecháním případně zopakováním vhodně vybraných částí. Svoji implementaci opřete o článek zmíněný v položce Literatura. Vyberte vhodnou plaformu pro implementaci. Identifikujte parametry, které celý proces ovlivňují, a proveďte uživatelskou studii, ve které zjistíte optimální nastavení těchto parametrů vzhledem k subjektivnímu hodnocení výsledku participanty. Rozhsah práce konzultujte s vedoucím práce.

Seznam odborné literatury:

Simon Wenner, Jean-Charles Bazin, Alexander Sorkine-Hornung, Changil Kim, Markus Gross: Scalable Music: Automatic Music Retargeting and Synthesis. Computer Graphics Forum (Proceedings of Eurographics 2013), vol. 32, no. 2, pp 345--354.

Vedoucí: Ing. Adam Sporka, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 26. 3. 2015

České vysoké učení technické v Praze

Fakulta elektrotechniky

Katedra počítačů

Bakalářská práce

Změna délky hudby

Pavel Štíbal

Vedoucí práce: Ing. Adam Sporka, Ph.D.

Studijní program: Otevřená informatika, Bakalářský

Obor: Softwarové systémy

Květen 2015

Poděkování

Rád bych poděkoval všem, kteří mé v této práci podporovali. Děkuji PaedDr., Mgr., Haně Žáčkové za jazykovou korekci. Především děkuji Ing. Adamu Sporkovi, Ph.D., vedoucímu bakalářské práce, za odborné vedení, za pomoc, za rady při zpracování této práce. V poslední řadě bych chtěl poděkovat účastníkům závěrečného testování.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 15.5.2015 .....

Pavel Štíbal

Abstrakt

Cílem této práce je prozkoumat a naimplementovat problematiku změny délky skladeb, kde se nezmění rychlost tempa ani vnímaní výšky tónu. Dalším úkolem je identifikovat parametry, které proces ovlivňují a následně provést uživatelskou studii. Tato problematika je rozsáhlá a konkrétní řešení nejde zcela jednoduše popsat. Nicméně k vypracování jsem použil přidělený článek a některé algoritmy (upravený algoritmus pro prohledávání cesty a MFCC). Vytvořené řešení poskytuje prodloužení, nebo zkrácení skladby na požadovanou délku. Na základě provedené studie je popsána možnost optimálního nastavení.

Klíčová slova

Skladba; změna délky skladby; vytvoření varianty skladby; C/C++

Abstract

The target of this project is to look into the problematic of length retargeting of music when there is not change in the tempo and the height of tones. Another target is to identify the parameters which have influence on the process. After that the final task is to make the user study. This problematic is large and it is not easy to describe the definitive solution. However, I have used for the work an article that was given to me and also some algorithms, for example modified algorithm for searching a way and MFCC. The created solution provides prolongation or shortening music on size which are required. The suitable setting is found from the result of the user study.

Keywords

Music; length retargeting of music; creating a variant of music; C/C++

Obsah

Seznam obrázků

Seznam rovnic a tabulek

Seznam zkratek

BPM	Beats per minute (počet dob za minutu).
WAV	Waveform audio file format
RAW	raw data (nezpracovaný data)
RIFF	Resource interchange file format
ASCII	American Standard code for Information Interchange
PCM	Pulse-code modulation (Pulzně kódová modulace)
LPCM	Linear pulse-code modulation
MFCC	Mel Frequency Cepstral coefficients
CLI	Common Language Infrastructure
ECMA	European Computer Manufacturers Association
CLR	Common Language Runtime
GUI	Graphical User Interface

1 Úvod

Cíle této práce jsou podrobné prozkoumání a pochopení problematiky. Naimplementovat aplikaci, která umožní uživateli změnit délku nahrané hudby, Aplikace by měla umět načíst soubor typu WAV a vytvořit příslušnou variantu pomocí vstupního nastavení.

Zvolil jsem si příslušné téma, protože se zajímám o oblast hudby např.: vytváření hudby. Také mě velice zaujala problematika adaptivní hudby v počítačových hrách, kde se změna délky hudby může použit.

Tato práce se zabývá analýzou a implementací problematiky změny délky skladby, avšak v zadání je uvedeno změna délky hudby, což vychází z anglického názvu length retargeting of music, který má správně překládat jako změna délky skladby, kde rychlost tempa a vnímaní výšky tónu jsou stejné jako v originální skladbě. Problém je popsán ze dvou pohledů, protože pod názvem změny délky hudby si můžeme přestavit i změnu rychlosti tempa či beze změny rychlosti tempa. Proto zde bude lehce nastíněn i současný stav dané problematiky. Analýzou tohoto problému se budu snažit odhadnout potíže, které mohou nastat při programování. V této kapitole bude dále zjišťovat, jaký je nejlepší programovací jazyk pro tuto práci. V implementaci programu je se popsán program a celý algoritmy, které jsou použité v aplikaci. Dále zde upřesňuji vzniklé problémy a popisuji nové, které vznikly v průběhu programování a nebyly předpokládané z předchozí analýzy. Samozřejmě je zde vysvětleno jejich řešení. V testovaní je definovaná uživatelská studie, budou zde uvedeny parametry, které ovlivňují proces, dále i ty které nemůže měnit uživatel. Z nasbíraných dat je vytvořeno optimální nastavení.

2 Popis problematiky a specifikace cíle

Kapitola popisuje problém ze dvou pohledů, kterými jsou změna délky skladby i se změnou rychlosti tempa a beze změny rychlosti tempa. Protože tyto problematiky spolu souvisí a ve změně skladby bez změny rychlosti tempa se mohou použít některé techniky z druhé oblasti a naopak. Na závěr této kapitoly specifikuji cíle bakalářské práce a uvedu přehled očekávaných případů spuštění.

2.1 Se změnou rychlosti tempa

Tuto metodu je jednoduché vytvořit v programech, které pracují s editací zvuku nebo hudby. Jsou to např.: Audacity a Reaper.

Do daného programu je nahrána skladba. Jediný parametr, který se změní je vzorkovací frekvence (dále jen frekvence). Když je snížená frekvence, nahraná skladba je prodloužená a poslechově zpomalená, a jestliže je zvýšená frekvence, původní skladba je zkrácená a poslechově je zrychlená. Tato technika se nazývá: převzorkování.

Dále uvedené rovnice Rovnice **A.3** je viditelné to, co již bylo uvedené výše. Tedy pokud se zvýší frekvence, nebo se zvýší tempo originální skladby, daná skladba se zkrátí. Platí to i naopak, tím je myšleno, že pokud se sníží frekvence, nebo se sníží tempo originální skladby, daná skladba se prodlouží.

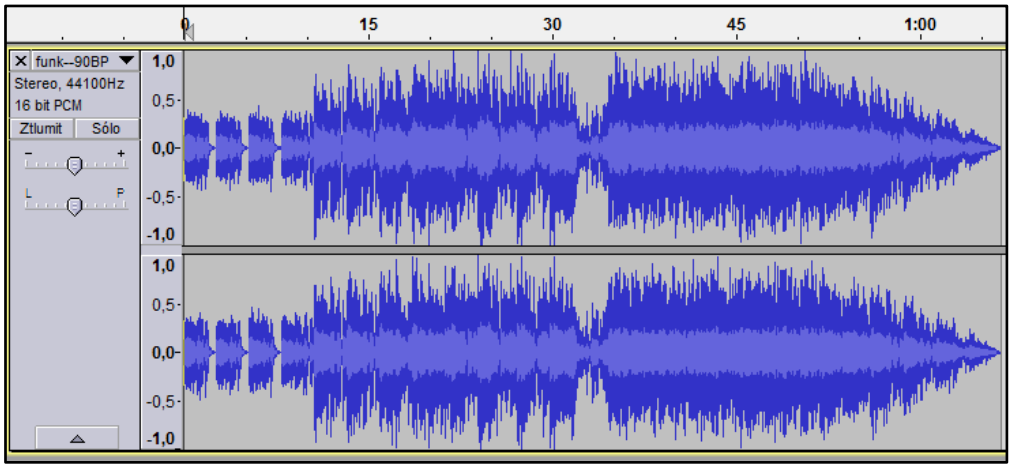
2.1.1 Převzorkování

Jedna se techniku, která má mnoho řešení a správné mohou být i všechna řešení. Můžeme se vydat dvěma cestama:

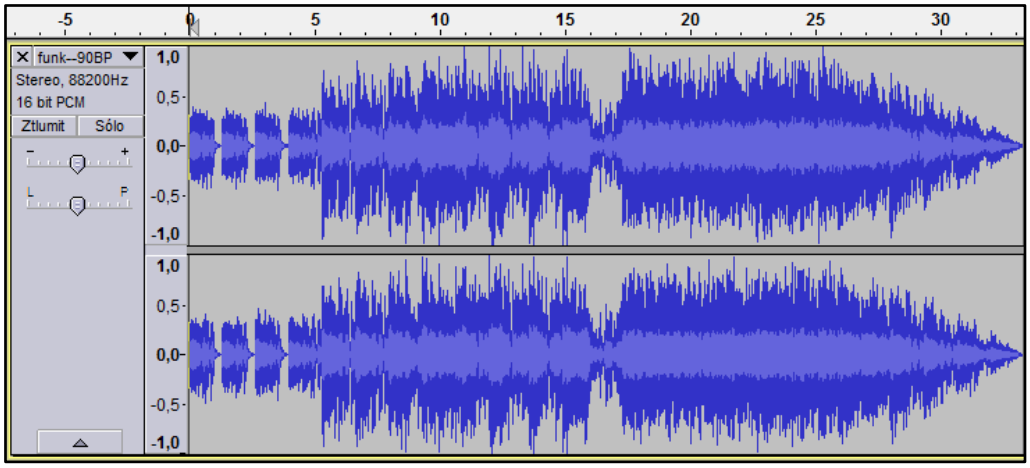
1. **Mění se výhradně frekvence,** tedy původní skladba se pouze změní v počtu vzorků za jednu sekundu.
2. **Změní se počet vzorků a frekvence zůstane stejná**. Pokud skladba má být zkrácená, vypočítá se kolik vzorků je hledáno na dobu z uvedené rovnice Rovnice **A.3** a za tuto frekvenci se musí dosadit frekvence jiná. Po výpočtu se zredukuje počet vzorku pomocí aritmetického průměru, mediánu nebo vynechání některých vzorků. Pokud naopak je požadováno prodloužení skladby, opět se vypočítá, kolik vzorků je hledáno na dob uvedené rovnice Rovnice **A.3** a za frekvenci se musí dosadit frekvence jiná. Po výpočtu se navýší počet vzorků pomocí zopakování některých vzorků nebo lineální interpolací. Mohou se použít i složitější techniky, než jsou zde uvedené.

Rozdíl mezi 1, a 2. metodou je v tom, že při použití 1. metody nikdy nepřijdeme o žádnou informaci a je jednoduché obnovit skladbu na původní frekvenci.

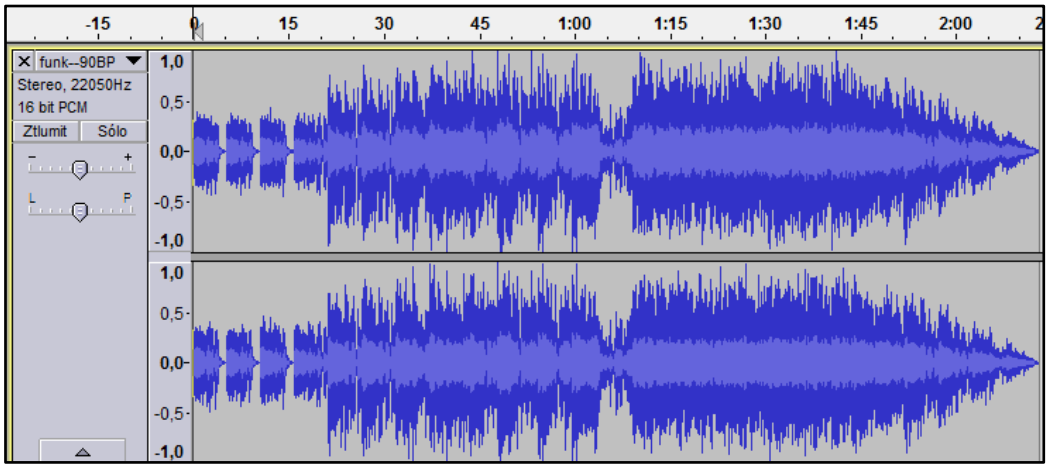
2.1.2 Ukázky audio signálů



Obrázek 2.1: Originální skladba s frekvencí 44,1kHz



Obrázek 2.2: Zkrácená skladba s frekvencí 88,2kHz



Obrázek 2.3: Prodložená skladba s frekvencí 22,05kHz

U všech výše uvedený obrázků je v horní části časová osa. Je zřetelné to, co jsem uvedl v kapitole 2.1 a 2.1.1, že je skladba zrychlená nebo zpomalená.

2.2 Bez změny rychlosti tempa

V této oblasti se nesetkáme s plnou automatizací změny délky skladby, protože tento problém je velice rozsáhlý. Nicméně opět může použít programy, které pracují se zvukem (viz kap. 2.1). Jelikož se nesetkáme v této problematice s automatizací, je člověk nucen si pozměnit originální skladbu manuálně. Přináší to velký problém, protože ne každý jedinec je zvukově nadaný tzn. nemá hudební sluch nebo hudební cit. Následkem toho výsledek může být velmi špatný a většinou se musí často opravovat, aby se dosáhlo kýženého výsledku.

V této problematice se nesmí změnit tempo, protože je nežádoucí, aby původní skladba byla zkreslená rychlostí (viz kap. 2.1). Když je požadovaná ztráta některých informací, může vést ke zkreslení dané skladby.

2.2.1 Zkrácení

Pokud je požadováno zkrácení původní skladby, provede se to tak, že se vhodně vynechají doby.

Princip této techniky je následující: nejprve vložíme do výsledku hledanou dobu. Pak nalezneme vhodnou dobu, která je podobná potřebné. Pokud takovou dobu nalezneme, přejdeme na dobu po ní a přidáme k výsledku, protože hledanou dobu již máme obsaženou ve výsledku a nepotřebujeme dva stejné zvuky za sebou, ba naopak přidáváme odlišné zvuky pro lepší zvukový vjem varianty původní skladby.

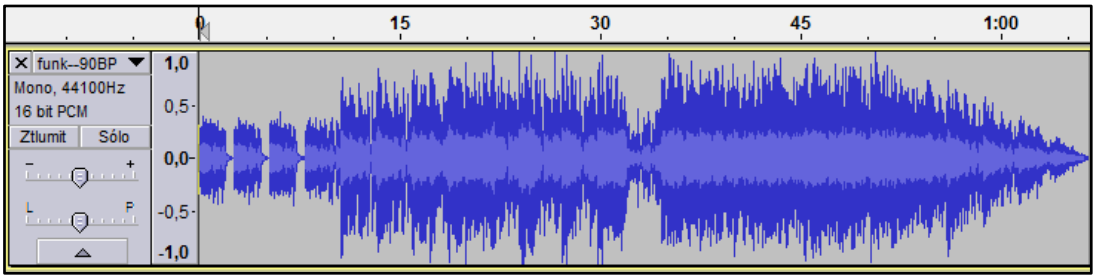
2.2.2 Prodloužení

Jestliže chceme prodloužit originální skladbu, vhodně zopakujeme doby či takty.

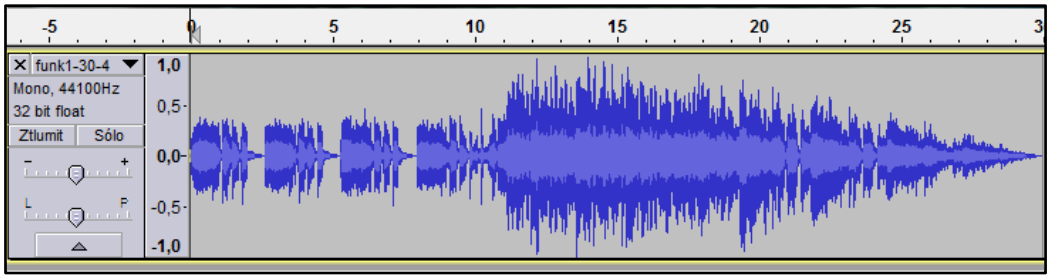
Princip techniky je následující: nejprve vložíme do výsledku hledanou dobu. Pak nalezneme vhodnou dobu, která je podobná hledané. Pokud takou dobu nalezneme, dvakrát přidáme do výsledku část mezi hledanou dobou a naleznou dobou, nebo danou část přidáme po taktech, tzn. pokud daná část má velikost osm dob, tak přidáme dvakrát první čtyři doby a pak dvakrát druhý čtyři doby.

Obě varianty prodloužení délky mají své klady i zápory, a nejde jednoznačně říci, která metoda je lepší. Ale oběma se dostane kýženého výsledku.

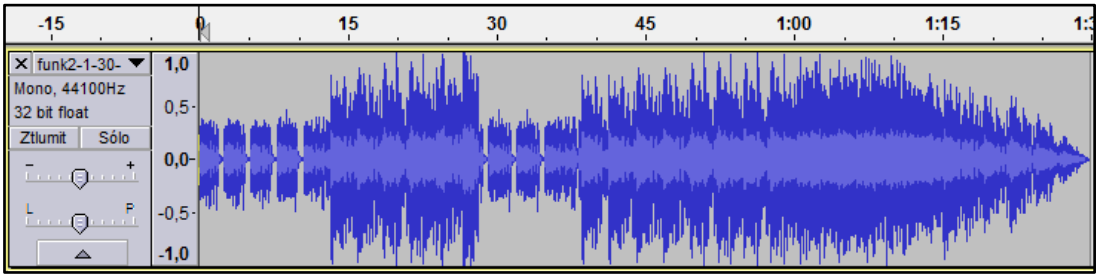
2.2.3 Ukázky audio signálů



Obrázek 2.4: Originální mono skladba



Obrázek 2.5: Zkrácená originální mono skladba



Obrázek 2.6: Prodloužená originální mono skladba

U všech výše uvedený obrázků je v horní části časová osa. Je patrný rozdíl mezi originální skladbou (viz Obrázek 2.4) a jakoukoliv variantou původní skladby (viz Obrázek 2.5 a Obrázek 2.6), kde byly některé časti vynechány, nebo naopak zopakovány.

2.3 Cíl práce

Implementace je opřena o přidělený odborný text. Cílem této práce je naimplementovat program, který dokáže automaticky vytvořit variantu nahrané skladby, která bude mít požadovanou délku. Nahranná skladba je typu WAV.

Výstupní varianta skladby může být typu RAV. Nicméně vhodnější by bylo, kdyby výstupní soubor byl také typu WAV. Protože soubor typu WAV se muže přehrát téměř v každém přehrávači a není potřeba speciální přehrávač, nebo program. Dále na výstupu jsou požadované dva obrázky. 1. obrázek bude reprezentovat rozdíl mezi původní variantou a variantou, která bude vytvořená. 2. obrázek bude znázorňovat matici podobnosti, která má v sobě hodnoty, jež určují vzdálenosti mezi jednotlivými dobami.

2.4 Případy užití

2.4.1 Práce se souborem

Nahrání hudby

1. Uživatel vybere položku „input“ v menu
2. Systém nabídne uživateli výběr souboru
3. Uživatel vybere soubor typu WAV
4. Načte se daný soubor
5. Systém zkontroluje, jestli je požadovaný soubor typu

   5a. Nenačte se cesta souboru

I. Systém upozorní uživatele
II. Uživatel vybere nový soubor

1b. Načte se cesta souboru

2. Systém ukončí akci

Vytvoření hudby

1. Uživatel vybere položku „output“ v menu
2. Systém nabídne uživateli okno pro uložení souboru
3. Uživatel vytvoří soubor typy WAV
4. Systém zkontroluje, jestli je požadovaný soubor typu

4a. Nenačte se cesta souboru

I. Systém upozorní uživatele
II. Uživatel vybere nový soubor

4b. Načte se cesta souboru

5. Systém ukončí akci

Exportování obrázku

1. Uživatel vybere položku „save image“ v menu
2. Systém nabídne uživateli okno pro uložení souboru
3. Uživatel vytvoří soubor typy BMP
4. Provede se kontrola, jestli je požadovaný soubor typu

4a. Systém nemůže uložit soubor

I. Systém upozorní uživatele
II. Uživatel vybere nový soubor

4b. Systém může uložit soubor

I. Systém obrázek uloží
II. Systém upozorní uživatele, že byl soubor uložen

5. Systém ukončí akci

2.4.2 Ostatní

Ostatní spuštění

1. Uživatel nastaví cílovou cestu k souboru, který chce nahrát
2. Uživatel nastaví počet BPM
3. Uživatel nastaví cílovou cestu k souboru, který chce vytvořit
4. Uživatel nastaví počet filtrů
5. Uživatel nastaví počet rámců na dobu
6. Uživatel nastaví konečnou délku

6a. Počet minutu

6b. Počet sekund

7. Uživatel vybere metodu
8. Uživatel vybere položku „start program“
9. Systém provede kontrolu

9a. Program nebude pokračovat

I. Systém upozorní uživatele

II. Uživatel opraví vstupy nebo ukončí program

9b. Program pokračuje

I. Systém upozorní uživatele

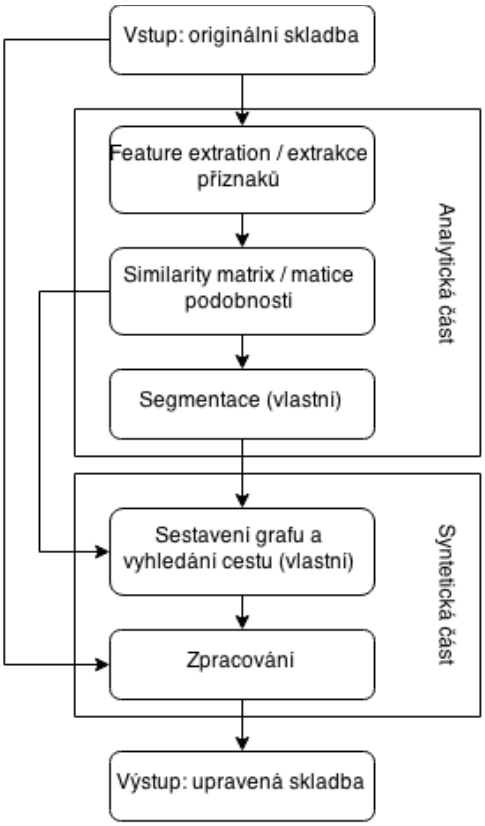
II. Systém vytvoří požadovaný soubor a k tomu dva obrázky

Přepínání obrázků

1. Uživatel zvolí položku „Show difference between original and modified song“ ne „show similarity matrix“
2. Systém zobrazí příslušný obrázek

3 Analýza problematiky

Jak bylo zmíněno výše, implementační část se hodně opírá o přidělenou literaturu a v ní je uvedený algoritmus, který bude v této kapitole podrobně zanalyzován. Dále zhodnotím, jaké programovací jazyky jsou vhodné pro tento algoritmus.



Obrázek 3.1: Diagram algoritmu/problematiky

Problematika se skládá ze dvou velkých částí, které obsahují menší úseky, a dále ze vstupu a výstupu (viz Obrázek 3.1). Šipky znázorňují to, s čím je pracováno v určitém segmentu, aby bylo dosaženo kýženého výsledku.

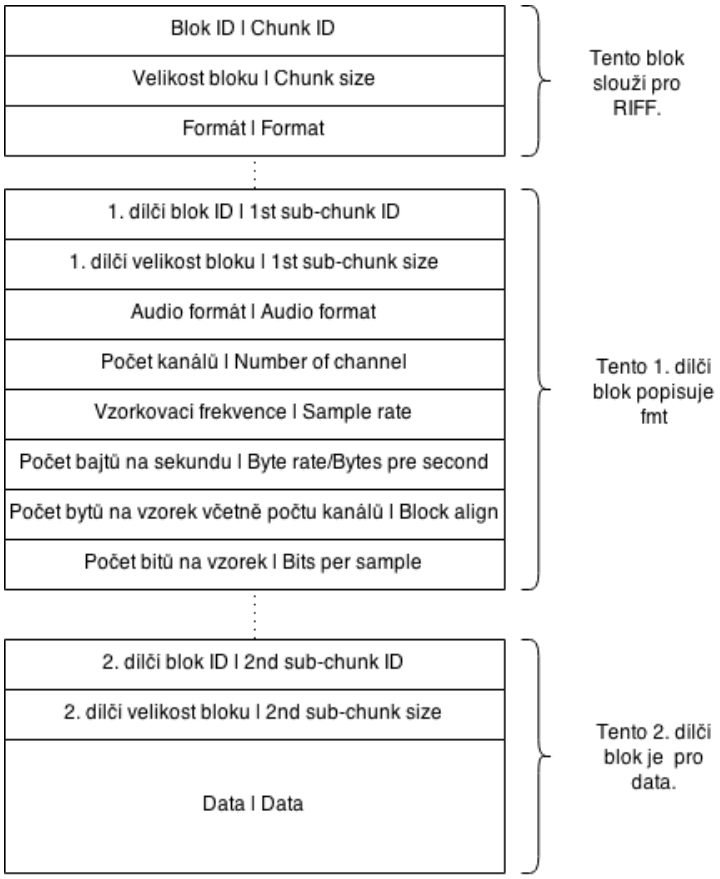
3.1 Vstup a výstup

Jak bylo uvedeno v kapitole 2.3, aby se skladba mohla nahrát, musí být soubor typu WAV. S tím přichází menší komplikace, která souvisí s volbou hlavičky, nebo struktury daného souboru. V některých programovacích jazycích je nutné si vytvořit vlastní WAV načítač, nebo použít nějakou open souce knihovnu. Nicméně ne všechny knihovny poslouží svému účelu, a proto je zde rozeberu podrobně.

Jak bylo uvedeno výše, ne každá knihovna může být komplexní, nebo má vlastní načítaní, proto bych v této kapitole rozebral strukturu typu RAW. Z toho důvodu, že je jednoduší na vytvoření.

Jelikož mým úkolem není rozpoznat jaké BPM má nahrána skladba a kolik dob obsahuje jeden takt, bylo by vhodné prokoumat, jaké problémy přinese toto usnadnění.

3.1.1 Struktura souboru typu WAV



Obrázek 3.2: Struktura souboru typu WAV

WAV struktura se používá k reprezentaci digitalizovaných zvuků. A hlavička je veliká 44 bajtů. V popisu WAV struktury budu vycházet z obrázku (viz Obrázek 3.2). Postupovat budu následujícím způsobem: popíšu nejprve celý blok, poté přejdu k jednotlivým částem.

1. blok popisuje RIFF, který obsahuje další tři části, které jsou:
   I. je blok ID, jenž obsahuje písmena „RIFF“ v ASCII kódování. Tím se označuje, že jde o soubor typu RIFF. Každý znak je dlouhý 1 bajt. Čili blok ID má rozsah 4 bajty.
   II. je velikost bloku, která reprezentuje velikost celkového souboru, do kterého není započtených prvních 8 bajtů, Tedy není zahrnuto do této velikosti blok ID a velikost bloku. Rozsah této části je 4 bajty. Rozsah se vypočítá dvěma způsoby:

$$Velikost\ bloku = 36 + 2.\,dílčí\ velikost\ bloku$$

Rovnice 3.1: 1. výpočet velikosti bloku

$$Velikost\ bloku = 4 + 8 + 1.\,dílčí\ velikost\ bloku + 8 + 2.\,dílčí\ velikost\ bloku$$

Rovnice 3.2: 2. výpočet velikosti bloku

Vypočet je přesnější, když se použije Rovnice 3.2.

III. přestavuje formát souboru, který obsahuje pro soubor typu WAV vždy písmena „WAVE“. Opět každý znak je dlouhý 1 bajt, tedy formát je velký 4 bajty. Formát WAV se skládá ze dvou dílčích bloků.

2. 1. dílčí blok reprezentuje formát zvukových dat. Skládá se z osmy menších částí, které jsou:
   I. 1. dílčí blok ID obsahuje písmena „fmt“ a zároveň zahrnuje koncový znak, jenž je null. Opět každý znak je dlouhý 1 bajt. To vede k tomu, že tato část je také dlouhá 4 bajty.
   II. 1. dílčí velikost bloku obsahuje kladné číslo, které reprezentuje délku formátování. Typicky pro PCM je 16. Velikost této oblasti je 4 bajty.
   III. Audio formát se pro PCM rovná 1. Tato hodnota označuje, o jakou kompresi se jedná. Pokud je rovna 1, která je pro tento typ souboru typická, jedná se o PCM s lineárním kvantováním (LPCM) a neobsahuje žádnou ztrátovou kompresi dat. Tato oblast je velká 2 bajty.
   IV. Počet kanálů je reprezentován kladným číslem. Pokud je číslo rovno 1, tzn. skladba obsahuje jeden kanál, kterému se říká „mono“. Naopak pokud je číslo rovno 2, tzn. skladba obsahuje dva kanály a které jsou nazývány: „stereo“. Samozřejmě počet kanálů může být větší než 2. Tato část zabírá v hlavičce 2 bajty
   V. Frekvence: tento pojem je podrobně popsán v příloze 0. Velikost, jež je reprezentovaná v struktuře, je 4 bajty
   VI. Počet bajtů za sekundu je reprezentován nezáporným číslem, které zabírá v hlavičce 4 bajty. Vypočítá se tímto způsobem:

$$Počet\ bajtů\ za\ sekundu = \frac{frekvence \times počet\ kanálů \times počet\ bitů\ na\ vzorek}{8}$$

Rovnice 3.3: Výpočet počtu bajtů za jednu sekundu

   VII. Počet bajtů na vzorek včetně všech kanálů je vyznačen nezáporným číslem, které je velké 2 bajty. Vypočítá se:

$$Počet\ bajtů\ na\ vzorek = \frac{počet\ kanálů \times počet\ bitů\ na\ vzorek}{8}$$

Rovnice 3.4: Výpočet počtu na jeden vzorek včetně všech kanálů

   VIII. Počet bitů na vzorek je reprezentován kladným číslem, které zabírá v struktuře 2 bajty. Číslo nabývá hodnotu takovou, která je dělitelná 8 beze zbytku a je větší než 7. Např.: 8, 16, atd.

3. 2. dílčí blok popisuje data daného zvukového souboru a skládá se ze tří menších částí:
   I. 2. dílčí blok ID obsahuje písmena „data“. Každý znak je dlouhý 1 bajt. Tedy tato část také má rozsah 4 bajty.
   II. 2. dílčí velikost bloku obsahuje číslo, které reprezentuje délku dat v bajtech. Velikost této oblasti je 4 bajty. Vypočet je následující:

$$2.\,dílčí\ velikost\ bloku = \frac{Počet\ vzoků + počet\ kanálů + počet\ bitů\ na\ vzorek}{8}$$

Rovnice 3.5: Výpočet velikosti 2. dílčího bloku

III. Data označují aktuální zvukové vzorky, které jsou reprezentovány kladným nebo záporným desetinným číslem. Data jsou uložená pomocí PCM

3.1.2 Struktura souboru typu RAW

Audio formát RAW je formát pro ukládání nekomprimovaných skladeb v nezpracované formě. Velikost souboru je srovnatelná s výše uvedeným typem, čili s WAV.

Obrázek 3.3: Typická struktura souboru typu RAW

Struktura má vzhled, který je znázorněn na obrázku Obrázek 3**.3.** Tedy struktura obsahuje pouze data, která reprezentují vzorky aktuální skladby. Jako v předešle kapitole i v této kapitole data jsou znázorněna kladnou či zápornou desetinnou číselnou hodnotou. Data jsou uloženy pomocí PCM, nebo ASCII. Velikost daného souboru je určená pouze daty.

K přehrání toho typu souboru je zapotřebí specializovaný nástroj, který má tuto možnost. Většinou je to jakýkoliv software, které pracuje zvukem. Např.: audacity.

3.1.3 BPM a počet dob na takt

BPM musí být celá kladná číselná hodnota. Slouží k výpočtu vzorků na dobu (viz Rovnice A.3). Jestliže je požadovaná hodnota BPM menší než 60, skladba má pomalejší tempo než normální. Naopak když BPM je vetší než 60, skladba je přehrávána rychleji, než je určené normálním tempem.

Počet dob na takt je reprezentován kladným číslem, které současné hudbě nabývá hodnoty 4. Samozřejmě může nabýt jiné kladné hodnoty.

Nicméně ani jeden ze dvou zvýše uvedených pojmu by neměl ovlivňovat běh programu.

3.2 Analytická část

První velká část se nazývá analytická, protože v této části probíhá analýza nahrané skladby. Pod pojmem analýza je myšleno práce a určité operace se vzorky. Přejde se do této oblasti téměř okamžitě po nahrání skladby. Tento úsek se skládá ze tří menších částí (viz Obrázek 3.1) a ze dvou částí, které nejsou uvedené na obrázku, ale určitě stojí za připomínku. V této kapitole popíši jednotlivé části.

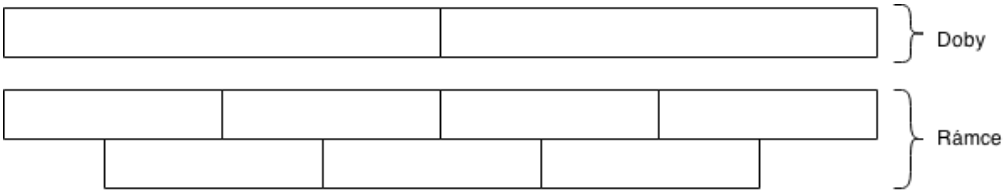
3.2.1 Rámcování

´ Pod pojmem rámcování se skrývá vytváření počtu rámců na dobu. Počet rámců na dobu se může lišit podle zadané hodnoty, která je uvedena jako kladná sudá číselná hodnota.

Tedy může nabývat hodnotu např.: 2, 4, 6, atd. Čím větší bude, zadaná hodnota, tím bude více rámců na dobu. Tzn., že analýza vzorků bude mnohem přesnější. Vytvořené rámce mohou být následující:

1. S překrýváním
2. Bez překrývání

Pro problematiku, kterou mám za úkol naimplementovat, je mnohem vhodnější varianta s překrýváním rámců.



Obrázek 3.4: Vytvořeni rámců

Princip vytváření rámců vychází z Obrázek 3.4. V obrázku jsou vyznačené dvě doby. Na jednu dobu odpovídají čtyři rámce. V první řádce v bloku pro rámce uvedeny jsou pouze rámce, které jsou označeny lichými číslicemi a 1. rámce začíná na nulové hodnotě doby. V druhé řádce jsou rámce, jenž jsou označeny sudými hodnotami a 1. sudý rámec začíná o polovinu rámce dále. Pro vytvoření jednoho rámce se používá 2. metoda převzorkování (viz kap. 2.1.1). Nejčastěji se používá vynechání některých vzorků. Tato metoda se používá, protože při vytváření meších částí dochází téměř vždy ke ztrátě některých dat. Pokud se nepoužije převzorkování, mohlo by dojít k problému, a to že rámce budou posunuté doprava, nebo doleva. Tzn., že pokud budou posunuté doleva, tak jakýkoliv třetí rámec pro každou dobu nebude končit na konci určité doby, ale bude ukončen o něco dříve. Naopak pokud budou posunuté doprava, tak jakýkoliv třetí rámec pro každou dobu nebude končit na konci dané doby, ale bude mít konec o něco dále za dobou. Jelikož skladby jsou delší než dvě doby, tak tento problém v kontextu celé skladby může přestavovat nemalý problém, a může dojít k velkému kreslení.

V této kapitole jsou potřeba dva výpočty. 1. slouží k výpočtu vzorků na jeden rámec, který vychází z rovnice (viz Rovnice A.3). 2. ukazuje, kde začíná 1. sudě označený rámec. Pokud bychom používali rámce bez překrývání, tak 2. rovnici bychom vůbec nepotřebovali. Výpočty jsou následující:

$$počet\ vzorků\ na\ rámec = frekvence \times \frac{munuta}{BPM} \times \frac{počet\ rámců\ na\ dobu}{2}$$

Rovnice 3.6: Vypočet počtu vzorků na jeden rámec

$$začátek\ prvního\ sudého\ rámce = \frac{počet\ vzorků\ na\ rámec}{2}$$

Rovnice 3.7: Výpočet začátku 1. sudého rámce

3.2.2 Okénkování

Okénkování slouží ke změně hodnoty daného prvku určitého rámce a provádí se nad každým rámcem většinou pomocí metody zvané. „Hamming window“. Tuto metodu navrhl Richard W. Hamming. Vychází z následující rovnice:

$$w(n) = \alpha - \beta \times cos\left(\frac{2\pi n}{N-1}\right)$$

Rovnice 3.8: Hamming Windows

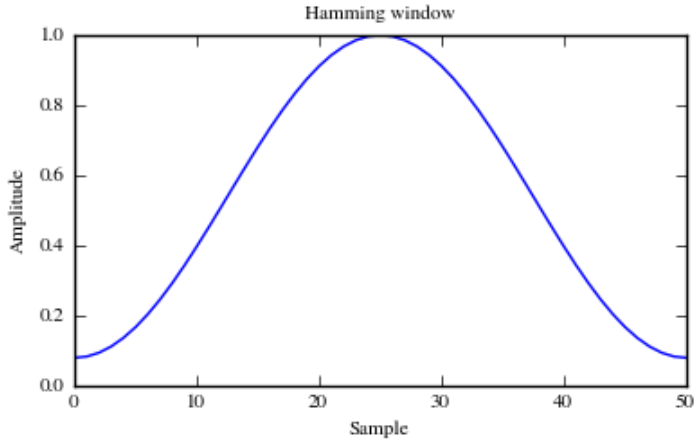
Je možné použít i jiné metody, které je určené pro okénkování např.:Hann window, nebo trojúhelníkové okénkování. Tyto metody jsou podobné Hamming window. Nicméně metoda, kterou jsem požil je lepší (viz kap. 3.3.2).

Nyní popíší jednotlivé znaky v rovnici: α je rovná hodnotě 0,54. β je vypočítá takto $\beta = 1 - \alpha = 0{,}46$. n reprezentuje celou kladnou hodnotu, která určuje pozici v rámci. n může být větší nebo rovno $N - 1$. N znamená velikost rámce, tedy v kontextu s naší problematikou je číselná hodnota, která je určená počtem na rámec. Ta se vypočítá z rovnice (viz Rovnice 3.6). Poté vypočítanou hodnotu $w(n)$ vynásobím vzorkem. Tedy po dosazení a upravení vypadá vzorec následovně:

$$w(n) = vzorek\ v\ rámci(n) \times \left[0{,}54 - 0{,}46 \times cos\left(\frac{2\pi \times n}{počet\ vzorků\ na\ rámec - 1}\right)\right]$$

Rovnice 3.9: Upravená a dosazená rovnice pro okénkování

Pokud vzorky v rámci jsou konstantní a rovny 1, tak Hamming window má vzhled jako je uveden na Obrázek 3.5). Z obrázku je patrné, že dochází ke změně výšky tónu nad rámcem.



Obrázek 3.5: Hamming window

3.2.3 Vytváření příznaků (features)

Provádí se pomocí algoritmu MFCC, který vytvoří příznaky k jednotlivým okénkům. Algoritmus MFCC je obvykle reprezentován vektorem o 13 příznaků pro jedno okénko. V této části přiblížím, co je algoritmus MFCC a kdy se nejčastěji používá.

Kdybych měl správně popsat tento algoritmus musel, bych zde znovu rozebírat rámcování (viz kap. 3.2.1) a okénkování (viz kap. 3.2.2). K algoritmu MFCC se vztahuje Mel měřítko, tedy je také tento algoritmus založen na lidském sluchu. Algoritmus MFCC se používá téměř u všech systémů, ve kterém je požadované automatickému identifikování složek zvukového signálu. S algoritmem MFCC se dá nejčastěji setkat u rozpoznávání řeči.

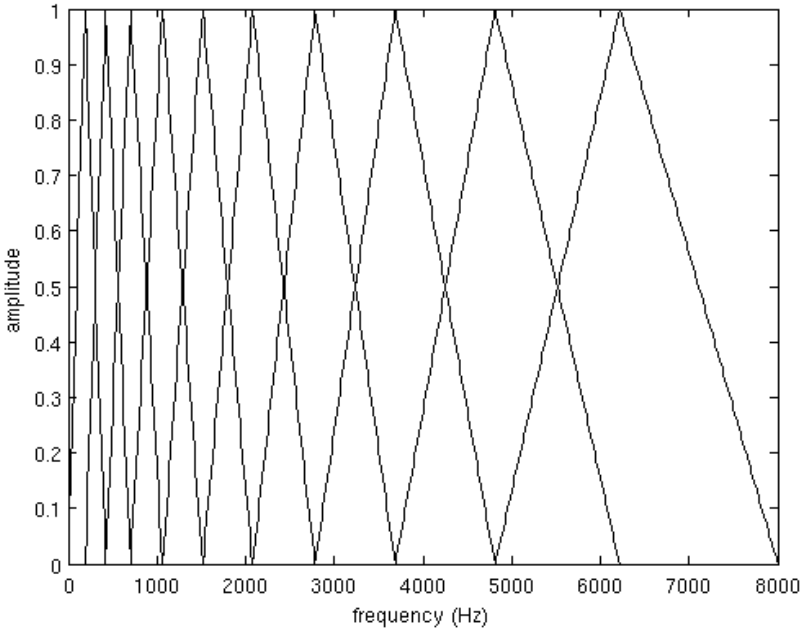
V algoritmu se používá Fourierova transformace pro okénko. Pomocí toho se zjistí odhad výkonového spektra, který je dán vztahem:

$$P_i(k) = \frac{1}{N} \times |w_i(k)|^2 \; 1 \leq k \leq K, 1 \leq i \leq N$$

Rovnice 3.10: Odhad výkonového spektra

kde *N* znamená velikost rámce, tedy v kontextu s naší problematikou jde o číselnou hodnotu, která je určená počtem na rámec. K je počet hledaných příznaků.

Dále je potřeba vypočítat Mel rozložení filtrační skupiny, která obsahuje množinu trojúhelníkových filtrů. Na tuto množinu je potřeba aplikovat odhad výkonného spektra. To vede k výpočtu energii filtračních skupin tak, že každá filtrační skupina je vynásobená odhadem výkonového spektra, poté jsou sečtené koeficienty (složky). Jaký může mít vzhled filtrační skupina je znázorněná na Obrázek 3.6.



Obrázek 3.6: Celá filtrační skupina

Postup je následující: provede se logaritmus každé energie filtrační skupiny, tato operace nezmění počet složek na energii, ale pouze zlogaritmuje koeficienty vektorů. Dále se použije se diskrétní kosinová transformace z logaritmů energií, čímž se získá z původního počtu hodnot pouhých 13 hodnot, které jsou uložené ve spodní části vektoru. Tyto hodnoty reprezentují počet hledaných příznaků.

3.2.4 Matice podobnosti

Matice podobnosti obsahuje skóre, které přestavuje podobnosti mezi počtem daných prvků. Každý prvek matice reprezentuje míru podobnosti mezi dvěma body. Tyto matice se používají v clusteringu (shlukování), v porovnávání sekvencí a v programování. Matice, která je potřebná do tohoto algoritmu, obsahuje vzdálenosti mezi příznaky.

Matice podobnosti bude obsahovat jednotlivé vzdálenosti mezi všemi příznaky. Pro vypočet vzdáleností použiji euklidovskou vzdálenost, která je pro tento účel nejvhodnější a je dána vztahem:

$$D_i(f_i, f_j) = \sqrt{f_i + f_j} = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{in} - x_{jn})^2}$$

$$i, j = \{1, 2, \ldots, počet\ příznaků\ na\ rámec (Většinou\ tato\ hodnota\ je\ 13)\}$$

Rovnice 3.11: Euklidovská vzdálenost

Nyní popíši slovně, jak se vypočítá vzdálenost, přičemž budu vyházet ze vzorce (viz Rovnice 3.11). Nejprve odečteme jednotlivé složky vektorů, které reprezentují příznaky, od sebe. Tato operace vytvoří nový vektor o *n* hodnotách, které následně jsou umocněny na druhou. Po umocnění sečtou všechny složky vytvořeného vektoru. Tato operace vypočítá jednu hodnotu, která musí být odmocněná. Pokud je číselná hodnota odmocněné, reprezentuje hledanou vzdálenost mezi danými příznaky. Vzdálenost může být nezáporná reálná hodnota.

Když jsou vypočítané všechny vzdálenosti, výsledná velikost matice je $N \times N$, kde N je celkový počet rámců. Na diagonále budou samé nuly, protože vzdálenost mezi stejným prvkem (příznakem) je nulová. Stejná čísla najdeme v i-tém řádků a v i-tém sloupci. Jinak řečeno, jsou stejná čísla v prvním řádku a v prvním sloupci matice, také i druhém řádků a v druhém sloupci matice, atd.

3.2.5 Segmentace

Segmentací je myšleno jakékoliv omezení, jako je hledaná cílová délka skladby, části, které se zahrnují do vytočení, počáteční a koncový uzel. Omezení dále může být takové, že se bude přecházet mezi rámci, nebo doby, či takty. Může představovat hodnotu, která reprezentuje možné přeskočení z doby na dobu,

3.3 Syntetická část

Druhá velká část se jmenuje syntetická. V této části nalezneme požadovanou variantr skladby a vytvoříme ji. Do této oblasti se přejde až když je provedená analýza na nahranou skladbu. Tento úsek se skládá ze dvou menších oblastí (viz Obrázek 3.1). V této kapitole popíši jednotlivé menší části.

3.3.1 Sestavení grafu a nalezení cesty

V této oblasti se pracuji se segmentací (viz kap. 3.2.5) a maticí podobnosti (viz kap. 3.2.4). Graf může být reprezentován i maticí, ve které je znázorněno hodnocení hran, které je

reprezentováno hodnotami v matici. Pomocí těchto hodnot se může nalézt vhodná cesta pro vytvoření požadované varianty skladby

Může se použít algoritmus pro vyhledání nejkratší cesty, nebo její modifikace. Samozřejmě se dá použít i jiný algoritmus pro vyhledávání vhodné cesty, kterou se dojde ke správnému výsledku. Jinými slovy: nalezení cesty se může provést mnoha způsoby a u všech je možnost se dobrat kýženého výsledku. Doporučuje se najít cestu z prvního uzlu do n-tého vrcholu a z n-tého uzlu do prvního vrcholu. Samozřejmě se může stát, že cesta nebude nalezena až do konce, tedy do požadovaného uzlu. Proto by se měly použít oba směry, které se porovnají a vytvoří se jejich nejlepší kombinace.

Všechny algoritmy vycházejí ze dvou základních principů, jenž jsou: prohledávání do šířky a prohledávání do hloubky.

Princip prohledávání do šířky je takový, že začneme procházet graf z libovolného uzlu. Tento vrchol se zpracuje a nalezne dosud nenalezené potomky, které uloží do fronty a tyto se potupně se zpracují stejným způsobem, jako předchozí uzel. Takto se postupuje, dokud fronta není prázdná.

Pseudokód:

```
funkce BFS (G,v)
    fronty se přidá (v)
    v je označen jako nalezený
    while fronta není prázdná
        v ← fronta.pop()
        for všechny hrany z v do w leží v G
                    if v není označen jako nalezený
                fronty se přidá (w)
                 w je označen jako nalezený
konec funkce
```

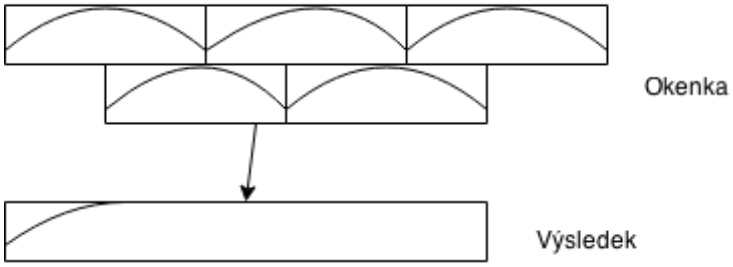
Princip prohledávání do hloubky je takový, že se zvolí libovaný uzel a označí se, jako návratová hodnota. Zpracuje se tak, že se přidá do zásobníku a pokračuje pomocí rekurze a potomka, který je jako poslední uzel (list) dané větve. Při návratu z rekurze se vrcholy označí jako uzavřené. Takto se pokračuje u všech větví.

Pseudokód:

```
funkce DFS (G, v)
     v je označen jako nalezený
     for všechny hrany z v do w leží v G
                if v není navštíven
          DFS(G, v)
konec funkce
```

3.3.2 Zpracování

Tato části pracuje s nalezenou cestou a originální skladbou, respekte okénky. Zde je zapotřebí sestavit variantu nahrané skladby bez změny výšky tonu. To se docílí tím způsobem, že se sestaví dohromady okénka, podobně jako se vytvořilo rámcování (viz kap. 3.2.1). Skládání je znázorněno na Obrázek 3.7.



Obrázek 3.7: Skládání pro jednu dobu

Princip skládání je takový, že se sečtou hodnoty na určité pozici lichých a sudých okének. Tedy pokud pod lichým okénkem není žádné sudé okénko, tak sudé okénko je reprezentováno nulami, platí to i naopak. Tudíž pokud nad sudým okénkem není část lichého okénka. Je tato část nahrazena nulami, jelikož se okénka překrývají a maximální hodnota je jedna. Tedy, když se sečtou hodnoty na pozicích, vyjde hodnota na dané pozici, která nezmění výšku tónu. Jinými slovy: nikdy se nemůže stát, že se překročí maximální hodnota.

3.4 Výběr programovacího jazyka

Nyní bude analyzovat, jaký programovací jazyk je nejvhodnější pro daný algoritmus. Jaké jsou tedy požadavky na programovací jazyk:

1. Jednoduchá práce s maticemi, jelikož se pracuje s maticovými operacemi
2. Rychlý výpočet

Z těchto požadavků mně vyjdou dva jazyky: matlab a C/C++.

Matlab, z toho důvodu, že je vhodnější pro maticové operace. Má mnoho potřebných funkcí již naimplementované, např.: Hamming. Nicméně matlab není příliš rychlý programovací jazyk.

C/C++ je jeden z nerychlejších programovacích jazyků, ale v tomto jazyku se využívají open source knihovny, ve kterých můžou být maticové operace. Mnoho funkcí se musí dopsat, či opravit open source knihovny tak, abychom je použít pro daný problém. Nicméně použitím C/C++ se docílí k lepšímu pochopení problematiky.

4 Realizace

Tato kapitola se zabývá samotnou implementací a strukturou aplikace. Například jaké komponenty byly využité. V této části se objeví zvolené technologie. Zde také budou popsané algoritmy, které jsou skutečně použity.

4.1 Použité algoritmy

Zde předvedu všechny důležité algoritmy a podrobně je popíši, pokud již nebyly popsány v jiné části. K některým uvedu i pseudokód.

4.1.1 MFCC

MFCC algoritmus slouží k vytvoření příznaků, který je podrobně popsán v kapitole 3.2.3. Byl představen v roce 1980 Davisem a Mermelsteinem a použili ho pro vzorkovací kmitočet 10kHz, pro šířku pásma řeči, která byla od 0 do 4600 Hz, a pro filtrační skupinu měla hodnotu 20. Dále bych uvedl jeho pseudokód.

Pseudokód:

```
funkce MFCC (okénko, n, velikost_okénka, frekvence)
    velikost = velikost_okénka / n
    for od 0 do n pro i
        lineární _start = 700 × 10 umocnit na (( i × velikost) / 2595) × (-1))
        linearní_konec = 700 × 10 umocnit na (((i + 1) × velikost) / 2595) × (-1))
        fft_frekvence = frekvence / 64

        N = (lineární_konec - lineární _start) / fft_frekvence
        for od 0,0 do N pro f
            melLog = 0
            melLog = melLog + fft pro vyše uvedené hodnoty
```

4.1.2 Nalezení cesty

V této části popíši dva algoritmy, první je pro nalezení cesty z počátečního bodu do koncového uzlu, Druhý je pro vytvoření jedné cesty, ze dvou nalezených dílčích cest.

Nalezení dílčích cest

Jelikož nepotřebuji najít nejkratší cestu od začátku do konce skladby, proto nepoužiji přímo Dijkstrův algoritmus, nebo A* algoritmus. Ale budu vycházet z nich a z principů prohledávání do šířky i hloubky (viz kap. 3.3.1). Vnitřek algoritmu se trochu liší. Podle toho, zda pracuji s prodloužením, nebo zkrácením. Nicméně nalezení vhodné hrany se neliší.

Nejprve popíši algoritmus, poté předvedu pseudokód. Na rozdíl od Dijkstrova algoritmu není potřeba se dostat do konečné hrany. Ale je požadovaný určitý počet uzlů. Algoritmus také pracuje s množinami vrcholů *V* a hran *E*. Na začátku jsou všechny vrcholy označené jako nenavštívené. Poté v cyklu je nalezená první hrana, která splňuje podmínky. Následně je přidaný vrchol, který je určen nalezenou hranou, do množiny a je označen jako navštívený.

K nalezení je možno použit i omezení. Co může být eventuálně použito, je popsáno v kapitole 3.2.5. Pokračuje se tak dlouho, dokud není nalezena požadovaná délka. Jelikož jde o skladbu, je žádoucí, aby existoval přechod, pouze na vyšší hodnotu uzlu, pokud se hledá od prvního uzlu. Tudíž kdyby se hledala cesta z koncového uzlu do prvního vrcholu, nesmí existovat přechod na vyšší hodnotu uzlu. Proto se může prohledávat jen od dané pozice vrcholu do koncového uzlu. S tím nastává problém, že do koncového uzlu se můžeme dostat dříve, než bude nalezen požadovaný počet. To vede k opakování algoritmu se změněnými parametry.

Pseudokód:

```
funkce najít_cestu (G, s, l, konec_V, omezení[n])
    for každý vrchol v v V
        nalezen[v] = 0 // nastavení nenalezeno
    p = s
    i = 0
    while i není větší než l
        nalezen[p] = nalezen[p] + 1
        cesta[i] = p
        i = i +1
        if p je rovna konec_V
            najit_cestu(G, s, l, konec_V, změněné_omezení)
        for některý sousední vrchol v z u, který je věší, než p
        a posouvá se o omezení[0]
            if G[p, v] < (omezeni[1] × G[p, p + 1]) a není označená jako nalezená
                Zde je nalezena hodnota a provede se dané operace pro danou metodu.
                Pokud se nenaleze žádná hodnota, přiřadí se uzel do množiny, který je
                za v tedy v + 1, což je zde označeno G[p, p + 1].
```

Tento algoritmus se provede ze dvou stran. Tedy vzniknou dvě cesty stejné dlouhé, ale s rozdílnou množinou vrcholů *V*.

Vytvoření výsledné cesty

Z nalezených dvou cest je potřeba vytvořit jednu cestu. Vytvoří se tak, že se porovnávají obě pole po dobách. Protože je striktně určeno, že na první pozici musí být první uzel z grafu a na poslední pozici musí být koncový vrchol grafu. Tedy na začátek výsledné cesty se přidá první množina uzlů V_1, která reprezentuje první dobu z cesty, která byla hledaná od prvního do posledního vrcholu (dále tuto cestu budu označovat *C1*). Na konec výsledné cesty se první přidá množina vrcholů V_2, která reprezentuje také první nalezenou dobu z cesty, která byla vytvořená od poledního do poledního uzlu (dále danou cestu budu označovat *C2*).

Po přidání obou dob se vypočítá výsledná vzdálenost pro první množinu vrcholů V_1. Dále se spočítá vzdálenost pro druhou množinu vrcholu V_1 z *C1* a také pro druhou množinu V_2 odzadu z *C2*. Vypočítaná vzdálenost z *C2* se porovná s výslednou vzdáleností. Pokud není podobná výsledné vzdálenosti, tak se do výsledné cesty přidá druhá množina uzlů z *C1* a změní se výsledná vzdálenost na vzdálenost *C1*. Jestliže je podobná výsledné vzdálenosti a také vzdálenost z *C1*. Zjistí se, která vzdálenost je blíže. Když je vhodnější vzdálenost z *C1*,

přidá se do výsledné cesty druhá množina vrcholů z *C1* a změní se výsledná vzdálenost na vzdálenost *C1*. Naopak pokud je blíže vzdálenost z *C2*, přidá se do výsledné cesty druhá množina od zadu z *C2* a označí se, že je v *C2*. Poslední možnost, která může nastat, je vzdálenost z *C2* podobná výsledné vzdálenosti a vzdálenost z *C1* nikoliv. Přidá se do výsledné cesty druhá množina od zadu z *C2* a označí se, že je v *C2*. Tak se pokračuje dále, dokud není vložená předposlední doba, nebo není označeno, že se nacházíme v *C2*.

Jestliže algoritmus označí, že se nachází v *C2*. Neprobíhají žádná porovnání a do výsledné cesty se přidá zbytek *C2* odzadu.

4.1.3 Skládaní hudby

Princip byl uveden v kapitole 3.3.2 a z něj se vychází. V této části se pracuje s okénky a s nalezenou cestou, která obsahuje index vrcholů, tedy indexy okének.

Algoritmus postupně prochází cestu. Vždy vybere index okénka, který je uložen na dané pozici cesty. První celé okénko se přidá do množiny (pole, či buffru) *Z*, jenž reprezentuje vytvořenou variantu skladby. Následné vybrané okénko se přidá do množiny *Z*, avšak nepřidá se za poslední vložené okénko. Přidá se do množiny *Z* tam, kde je polovina předchozího okénka. V části, ve které se okénka překrývají, se provede součet na daných pozicích okének. A druhou část okénka jednoduše vloží do množiny *Z*. Pokračuje se takhle dále, dokud se nepřidá poslední okénko

Například: pokud velikost okénka je dvě stě, tedy jeho polovina je na pozici devadesát devět, nebo sto podle indexace. První okénko je již uloženo do množiny a druhé se chystá uložit jeho data. Tedy druhé okénko chce vložit svou první hodnotu na pozici sto, kde již je uložené jiné číslo. Původní číslo na pozici sto se sečte s prvním prvkem přidávaného okénka. Stejným způsobem to probíhá pro hodnoty, které jsou na pozicích od 101 do 199 vloženého okénka a pro prvky, které jsou od druhého do stého přidávaného okénka. Zbylá data vkládaného okénka se pouze přidají do množiny. Dvě okénka vytvoří množinu o velikosti sto padesát prvků.

4.1.4 Další algoritmy

- Rámcování, s tím to problémem jsem se zabýval v kapitole 3.2.1
- Hamming window, který je použit pro okénkování. Jeho výpočet a popis je uveden v kapitole 3.2.2
- Výpočet vzdálenosti, který je proveden pomocí euklidovské vzdálenosti. Ta byla probrána včetně výpočtu v kapitole 3.2.4

4.2 Zvolené technologie

Programovací jazyk C/C++ byl pro práci vybrán pro svou volnou dostupnost, podporu open source knihoven a především proto, že se tak lépe porozumí této problematice. Kupodivu se tento jazyk stále může pyšnit velkou uživatelskou podporou v podobě různých knihoven, stejně jako velkým množstvím dokumentace, či rad od uživatelů.

Pro grafickou část jsem zvolil C++/CLI. C++/CLI je specifikace jazyka vytvořeného společností Microsoft. Slouží k zjednodušení Managed C++ syntaxi, která je nyní zastaralá. C++/CLI je standardizován ECMA. Toto rozšíření se používá i CLR, které slouží jako garbage collection. S C++/CLI se může každý uživatel setkat ve vývojovém prostředí Visual studio a téměř v každé nové aplikaci, která je psána pro Windows.

Tedy aplikace může obsahovat prvky za C, C++ i CLI..

4.3 Implementace

V této části přiblížím implementační stránku programu. Budu vycházet především z kapitoly 3. Implementace se skládá ze dvou částí. První je logická část, kde probíhají veškeré výpočty, i zde probíhá kontrola vstupů. Druhá je grafická, která slouží pro znázornění výsledů v podobě obrázků. Nejprve popíši použité knihovny a externí komponenty. V poslední řadě se zde objeví všechny komponenty.

4.3.1 Knihovny a externí komponenty

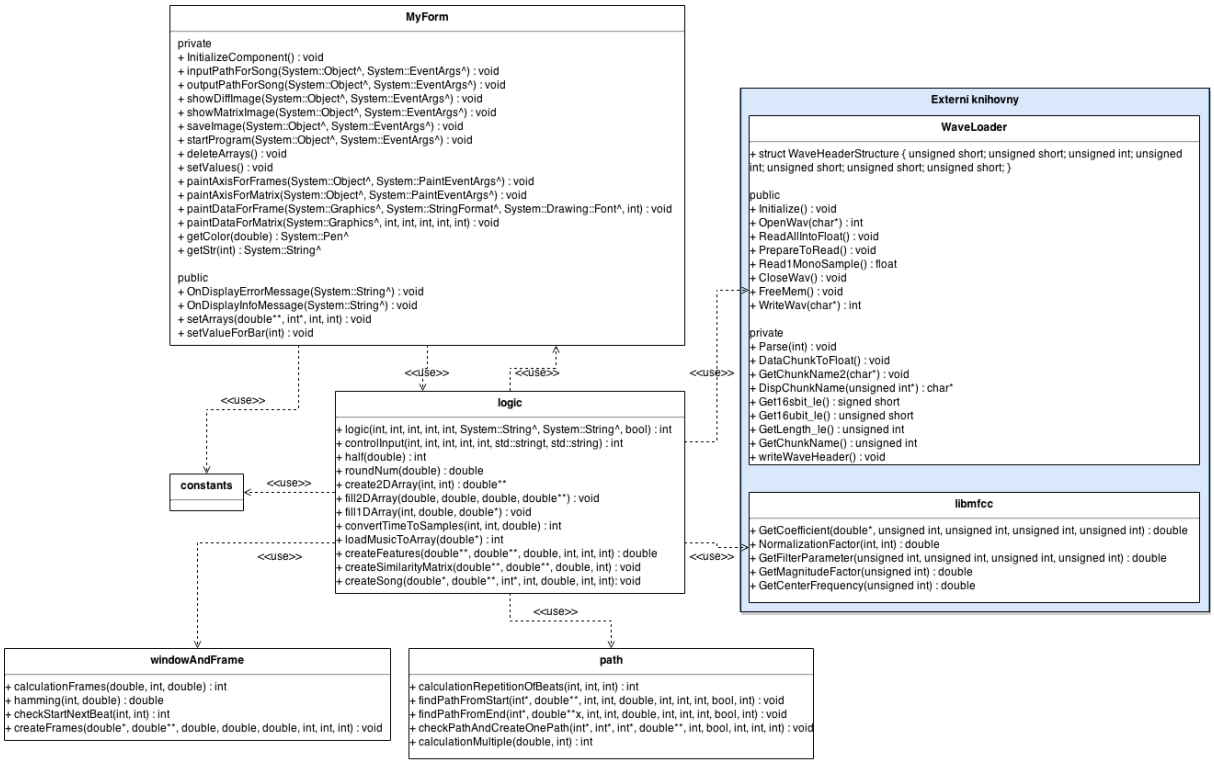
Jak bylo zmíněné v kapitole 4.2 jsou použité programovací jazyky C/C++ a C++/CLI. Tudíž aplikace využívá standardní knihovny těchto jazyků.

Po dohodě s vedoucím práce aplikace využívá dvě open source knihovny WaveLoader.h[1] a libmfcc.h[2]. WaveLoader.h slouží pro načtení a vytvoření Wav souboru. V libmfcc.h, či v libmfcc.cpp je možné nanést algoritmus MFCC, který je poměrně těžký vytvořit.

Hlavním účelem této práce je navržené celé aplikace, nikoliv vytvoření algoritmu MFCC či jak správně načítat a vytvářet soubory typu WAV. Tyto problematiky jsem měl pouze za úkol pochopit a správně zakomponovat do aplikace.

Zdrojový kód se skládá z osmi komponent, které jsou: *constants, libmfcc, logic, MyForm, path, WaveLoader, windowAndFrame* a *main.* Závislosti mezi jednotlivými komponenty jsou znázorněny na Obrázek 4.1. Komponenty znázorňují třídy, nebo knihovny, které složí soubor typu CPP. Komponenta *main* slouží pouze ke spuštění program a neobsahuje nic jiného než metodu main a inicializaci formuláře (GUI). Také díky tomu není vložená v obrázku.

[1] http://buzzwiki.robotplanet.dk/index.php/.wav_class_source
[2] https://code.google.com/p/libmfcc/



Obrázek 4.1: Závislosti mezi komponenty

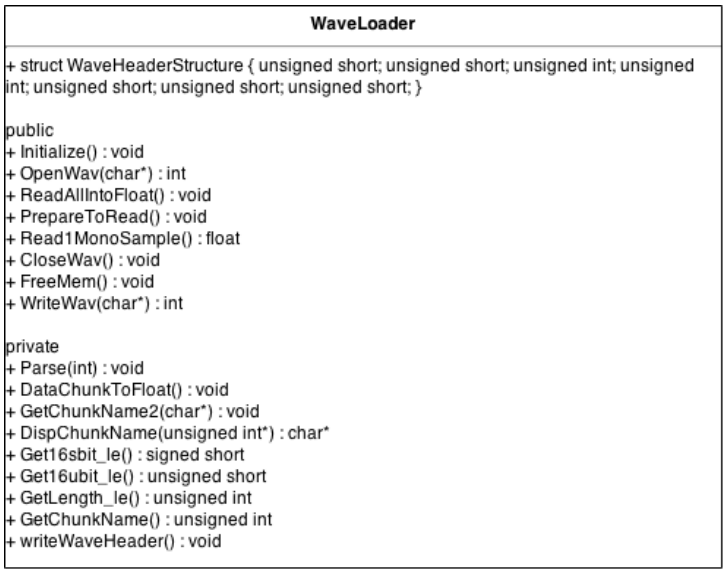
4.3.2 WaveLoader

Jak bylo uvedeno v kapitole 4.3.1 jedná se o externí třídu, která z slouží k načtení a vytvoření souboru typu WAV. Do této třídy jsem přidal metodu *writeWaveHeader*, protože bez této metody byl soubor ukládán jako RAW (viz kap. 3.1.2). Tato metoda slouží k vytvoření hlavičky souboru typu WAV (viz kap. 3.1.1). Je zavolaná z metody *WriteWav*, která slouží pro vytvoření souboru a vepsání do něj příslušných dat. Pro tento účel se používá bitový posun.

Třída využívá metodu *OpenWav* k otevření příslušného souboru, se kterým se bude pracovat. Dále pomocí funkce *PrepareToRead* se nastaví pozice na začátek datového bloku. To slouží k tomu, aby se lépe pracovalo se souborem a nemusel se vždy procházet od začátku. Bohužel třída obsahuje metodu *Read1MonoSample*, která načte skladbu jako mono, i když neobsahuje pouze jeden kanál. Sice to vypadá jako velká komplikace, ale opak je pravdou, protože tento aspekt nemá žádný vliv na výsledek aplikace. Vytvoření vzorku probíhá tak, že se načte vzorek z prvního kanálu, poté z dalšího. Takhle se to opakuje, dokud není načten vzorek z posledního kanálu. Výsledná hodnota je dělená počtem kanálů.

Další metody, které stojí za zmínku, jsou *CloseWav* a *FreeMen*. *CloseWav* slouží pro zavření obou souborů. Tedy zavření prvního souboru, který slouží pro čtení, a zavření druhého souboru, který slouží k vytvoření nového souboru typu WAV. *FreeMen* se využívá k uvolnění pamětí.

Všechny metody jsou uvedené na Obrázek 4.2.

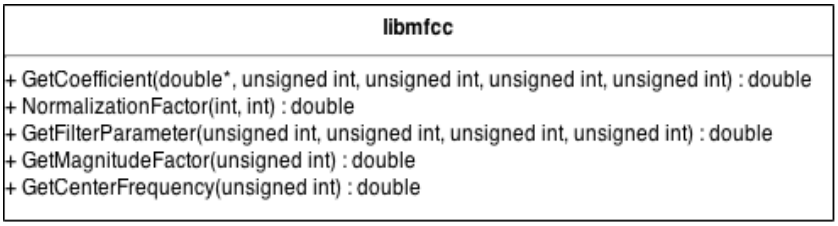


Obrázek 4.2: Znázornění metod WaveLoaderu

4.3.3 libmfcc

Jak bylo uvedeno v kapitole 4.3.1 jedná se o externí knihovnu, ve které je uveden algoritmus MFCC. Nachází se v ní čtyři metody. První funkce je *GetCoefficient* slouží k vrácení koeficientu, nebo příznaku. Další metody jsou volané pouze vnitřně. Druhá metoda je *NormalizationFaktor*, ve které se vypočítá normalizovaný činitel. Třetí funkcese jmenuje *GetFilterParameter*, zde se vypočítá parametr filtru pro zadanou frekvenci a flitračního pásma. Předposlední metoda je *GetMagnitudeFactor*, která slouží k výpočtu pásmově závislému modulového činitele pro danou filtrační skupinu. Poslední funkce se nazývá *GetCenterFrequency,* v této metodě se vypočítá střední frekvence pro zadané filtrační pásmo.

Všechny metody jsou uvedené na Obrázek 4.3.



Obrázek 4.3: Znázornění metod pro libmfcc

4.3.4 constants

Jedná se o knihovnu, které obsahuje pouze konstanty. Kontanty jsou typu double, int a string. Všechny konstanty jsou nadefinované marka #define. Stringy slouží k výpisům v grafické části.

4.3.5 windowAndFrame

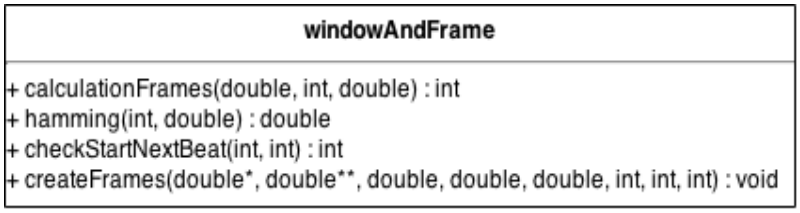
Jsou pojmenované tímto názvem dva soubory. První soubor je pro knihovnu a druhý soubor reprezentuje soubor, s kterým knihovna pracuje. Tato knihovna slouží především pro výpočty, které jsou spojené s rámci či okénky, a k vytvoření rámců i okének.

Metoda *calculationFrames* slouží k výpočtu počtu rámců. Tato metoda se používá pro zjištění počtu požadovaných rámců, který určují velikost varianty skladby. S funkcí *hamming* se pracuje při okénkování (viz kap. 3.2.2 a kap. 4.1.4). Uvedená funkce je zavolaná z metody *createFrames*.

CreateFrames slouží k vytvoření rámců (viz kap. 3.2.1) a okének nad skladbou. K vytvoření se používá načtená skladba. Jelikož se rámce překrývají, může nastat koplikace, že poslední rámec přesáhne délku skladby. Pokud rámec přesáhne konec skladby, doplní se do zbytku rámce nuly. Což v digitální hudbě znamená ticho. Zároveň při vytváření rámců se používá okénkování. Jelikož tyto problematiky spolu úzce souvisejí, tu díž mohou se naimplementovat dohromady. Poslední metoda, která je volaná z této funkce je *checkStartNextBeat.*

Metoda *checkStartNextBeat* se používá k zjištění, kde začíná další doba. Pokud začíná na stejné pozici, kterou uvádí pro vytvoření rámců na jednu dobu, tak se nic neděje. Pokud naopak hodnoty jsou rozdílné. Tato metoda posune začátek dalšího rámce na požadovanou pozici. Tedy to může být doleva i doprava.

Všechny metody jsou uvedené Obrázek 4.4.



Obrázek 4.4: Znázornění metod pro knihovnu windowAndFrame

4.3.6 path

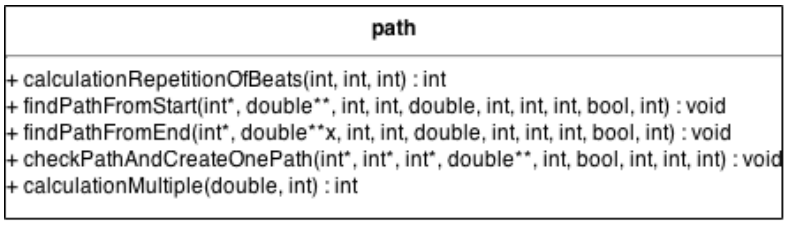
Jsou pojmenované tímto názvem dva soubory. První soubor je pro knihovnu a druhý soubor reprezentuje soubor, s kterým knihovna pracuje. Tato knihovna slouží především pro nalezení cest z obou směrů a pro vytvoření výsledné cesty.

Metoda *findPathFromStart* hledá potřebnou cestu od začátku do konce matice, nebo grafu (viz kap. 3.3.1 a kap. 4.1.2). Jak bylo zmíněno několikrát v této práci je potřeba nalézt cestu požadované délky. Funkce *findPathFromEnd* hledá cestu oproti předchozí metodě od konce do začátku grafu.

Metoda *checkPathAndCreateOnePath* nejprve zkontroluje, jestli nejsou obě cesty stejné. Pokud ano. Vloží se do výsledné cesty první nalezená cesta, tedy cesta, která byla nalezena od začátku do konce. Pokud cesty jsou rozdílné postupuje se podle algoritmu **nalezení cesty vytvoření výsledné cesty** (viz kap. 4.1.2).

Funkce *calculationMultiple* měla sloužit k výpočtu násobku. K tomu i slouží, ale za účelem zjištění kolikrát se vejde počet rámců do požadovaného počtu rámců. Výsledek označuje, kolikrát může navštívit rámec v metodách *findPathFromStart* a *findPathFromEnd.*

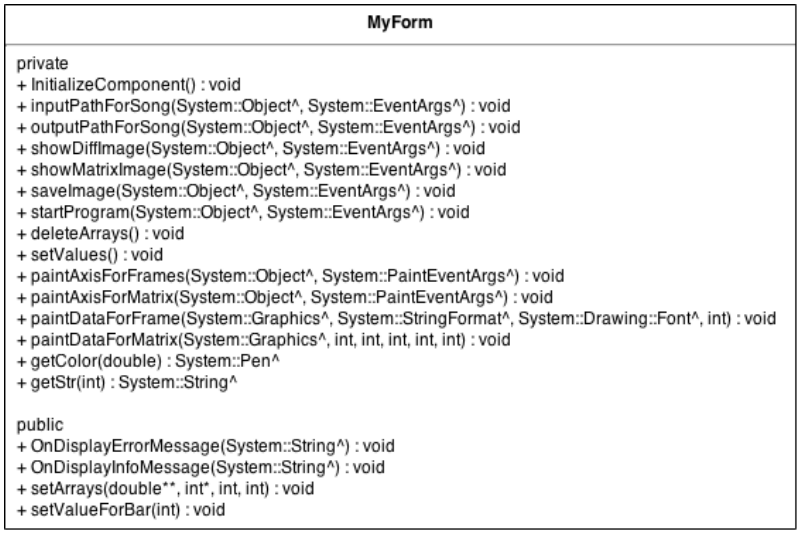
Všechny metody jsou uvedené na Obrázek 4.5.



Obrázek 4.5: Znázornění metod pro knihovnu path

4.3.7 Myform

Tato komponenta reprezentuje třídu, která slouží jako vlastní namespace. Tedy může být volaná téměř z jakéhokoliv místa programu pomocí namspacové syntaxe. Obsahuje metody, které jsou veřejné (public) a soukromé (private). Všechny metody jsou uvedené na Obrázek 4.6.



Obrázek 4.6: Znázornění metod pro třídu MyForm

Veřejné funkce slouží k nastavení některých proměnných, nebo polích. K těmto účelům se používají dvě metody *setValueForBar* a *setArrays*. Funkce *setValueForBar* slouží k nastavení hodnoty progres baru. Progres bar slouží k identifikování průběhu, nebo také v jaké části se nachází program. Tento panel zde musí být, protože někdy může operace trvat dlouho. Funkce *setArrays* pouze zkopíruje prvky polí do privátních polí.

Další veřejné metody pracují s dialogovým oknem, které jsou typu informativního hlášení a chybového hlášení. Metody se nazývají *OnDisplayErrorMessage* a *OnDisplayInfoMessage*.

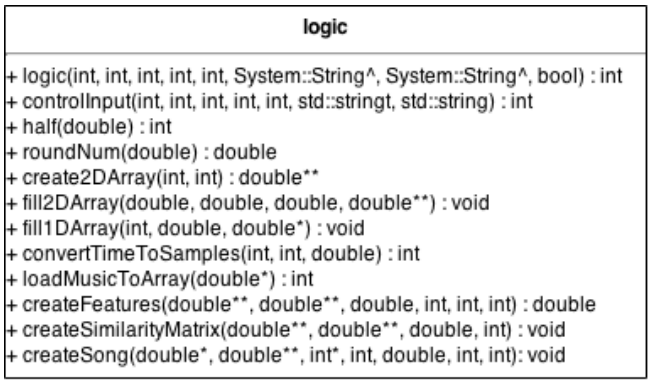
Soukromé funkce především slouží ke grafické stránce aplikace. Do této části paří naslouchače, metody, který pracují s grafikou, a pomocné funkce.

Naslouchače pro tlačítka, které jsou *inputPathForSong*, *outputPathForSong*, *showDiffImage*, *showMatrixImage*, *saveImage* a *startProgram*. Naslouchače *inputPathForSong*, *outputPathForSong* a *saveImage* fungují na podobné bázi. Pracují s otevíracím dialogovým oknem, nebo s ukládacím dialogovým oknem. Přes který se nalezne cesta k určitému souboru, který je požadovaný k otevření, či k uložení. Naslouchače *showDiffImage* a *showMatrixImage* slouží pouze k přepínání vytvořených obrázků. Poslední naslouchač *startProgram* spustí program a zavolá funkci *logic*, která převezme práci nad aplikaci (viz 4.3.8)

Metody, které pracují s grafikou, jsou *paintAxisForFrames*, *paintAxisForMatrix*, *paintDataForFrame* a *paintDataForMatrix*. Funkce *paintAxisForFrames* a *paintDataForFrame* vytvoří obrázek pro rámce, kde je viditelný rozdíl mezi originální skladnou a vytvořenou variantou skladby. Vytvořený obrázek obsahuje osy včetně pojmenování a vyznačených hodnot. Metody *paintAxisForMatrix* a *paintDataForMatrix* slouží k vytvoření obrázku pro znázornění vzdáleností v matici pomocí spektra. Ve vytvořeném obrázku je spektrum barev, které znázorňují, jak jsou daleko od sebe rámce, spektrum matice, osy včetně pojmenování a vyznačených hodnot.

Pomocné funkce jsou *getColor*, *getStr*, *deleteArrays* a *setValues*. Slouží k ulehčení práce v určitém okamžiku. Metoda *getColor* vrátí barvu a využívá se u spektra. Funkce *getStr* vrátí řetězec, nebo string. Používá se pro výpis v grafickém rozhraní. Metoda *deleteArrays* slouží k vymazání polí, které se používají globálně. Poslední funkce *setValues* nastaví proměnné na určitou hodnotu.

4.3.8 logic



Obrázek 4.7: Znázornění metod pro knihovnu logic

Jsou pojmenované tímto názvem dva soubory. První soubor je pro knihovnu a druhý soubor reprezentuje soubor, s kterým knihovna pracuje. Tato knihovna slouží k veškeré komunikaci mezi již uvedenými komponenty a pro různé výpočty. Všechny metody jsou uvedené na Obrázek 4.7. V tomto souboru se vypočítá např.: kolik je vzorků na jednu dobu, vypočítá se kolik rámců je na celou skladbu, atd. Do této knihovny se vstoupí po zavolání metody *logic*. Dále pomocí funkce *controlInput* se zkontroluje, zda vstup byl správně zadán. Metodou *loadMusicToArray* se načte skladba do předem vytvořeného pole, které je jednodimenzionální. Pro tuto metodu se využívá práce s třídou **WaveLoader** (viz kap. 4.3.2).

Funkce pro výpočty a zaokrouhlování jsou *half*, *roundNum* a *covertTimeToSamples*. Metoda *half* slouží k výpočtu poloviny dané hodnoty a zároveň vypočítanou hodnotu zaokrouhlí. Funkce *roundNum* zaokrouhlí požadovanou hodnotu, níže pokud je rozdíl mezi typem double a typem int dané hodnoty menší než 0.09. Pokud není hodnota je zaokrouhlena nahoru. Metoda *covertTimeToSamples* převede konečný čas na celkový počet vzorků.

Funkce *create2DArry* vytvoří dvoudimenzionální pole. S uvedenou funkcí může spolupracovat *fill2DArray*, která naplní dvoudimenzionální pole určitou hodnotou. Jinými slovy: nastaví všechny prvky pole na danou hodnotu. Na podobném principu pracuje i metoda *fill1DArray*, která nastaví všechny pozice jednodimenzionálnímu poli na určitou hodnotu.

Funkce *createFeatrues* slouží vytvoření vektoru příznaků pro daný rámec. Kde se využívají vytvořená okénka a knihovna **libmfcc** (viz kap. 4.3.3).

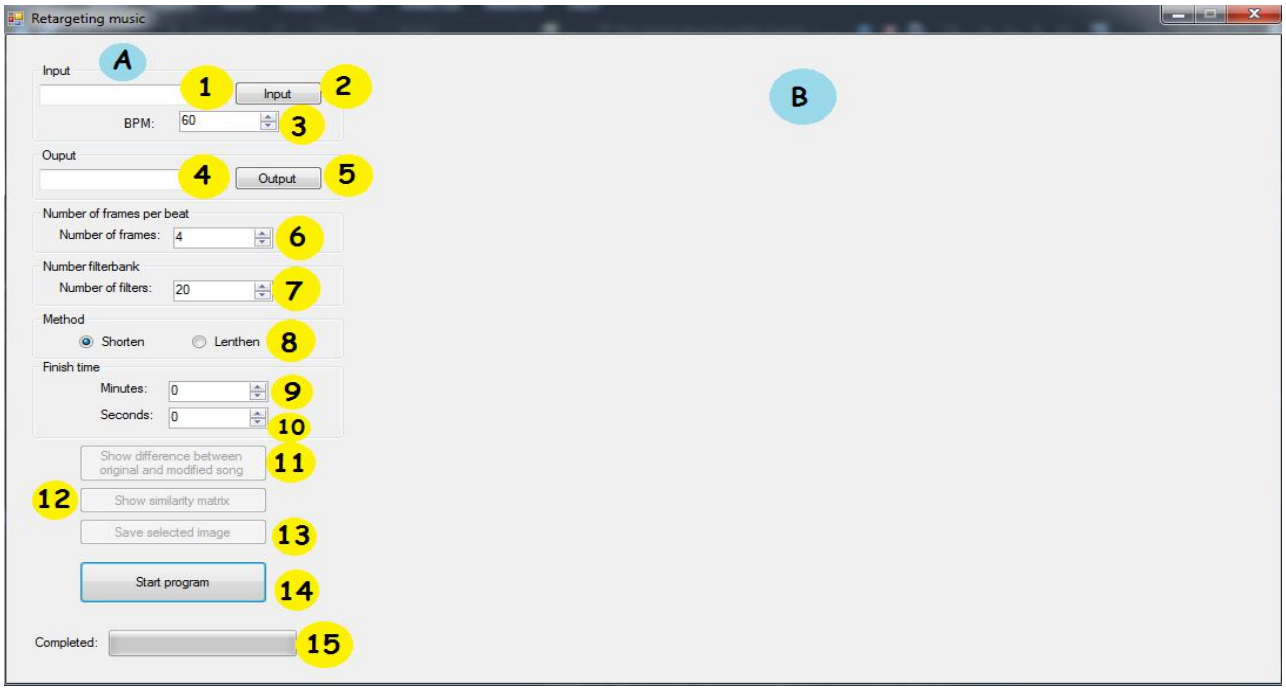
Metoda *createSimilarityMatrix* vytvoří matici podobnosti. K tomu se využívají vytvořené příznaky.

Funkce *createSong* slouží k vytvoření varianty skladby. Zde se využívají vytvořená okénka a nalezená výsledná cesta. K tomuto účelu slouží algoritmus pro skládání hudby (viz kap. 4.1.3)

4.4 GUI

Pro grafickou část jsem zvolil C++/CLI (viz 4.2). GUI umožňuje uživateli jednoduše nastavit veškeré parametry, které jsou pro tuto aplikaci důležité. GUI je grafická stránka aplikace, která je celá obsažená ve třídě **MyForm** (viz kap. 4.3.7). Dále slouží k znázornění výsledků pomocí grafické podoby a k výpisům hlášek nejen chybových.

4.4.1 Popis prvku GUI a ukázka GUI



Obrázek 4.8: GUI s označenými prvky

Nyní popíši všechny prvky, které jsou uvedené na Obrázek 4.8. GUI se nemůže zvětšit. Může se zavřít přez křížek a minimalizovat přes minimalizační tlačítko. GUI je se skládá ze dvou bloků *A* a *B*.

Blok *B* slouží pouze k zobrazení výsledných obrázků, které jsou obrázky pro znázornění rozdílu mezi originální a upravenou skladbou a pro znázornění vzdáleností pomocí spektra.

Blok *A* v tomto grafickém rozhraní reprezentuje menu, které skládá z patnácti prvků. Prvky přiblížím blíže:

1. Textové pole pro vstup. Do tohoto pole může uživatel zadat cestu k souboru, který chce otevřít
2. Tlačítko input slouží k otevření otevíracímu dialogovému oknu. Přes toto okno je jednoduší najít daný soubor. Cesta pak bude vložena do textového pole pro vstup
3. Reprezentuje počet BPM pro danou skladbu. Která musí být zadaná
4. Textové pole pro výstup. Do tohoto pole může uživatel zadat cestu k souboru, který chce vytvořit.
5. Tlačítko output slouží k otevření ukládajícího dialogovému oknu. V tomto okně se lépe vytvoří daný soubor. Cesta bude následně přidána do textového pole pro výstup
6. Počet rámců na jednu dobu. Tzn. nakolik částí bude rozdělena doba
7. Počet filtrů se v této aplikaci je uvedeno proto, že algoritmu MFCC se používá většinou pro rozpoznání řeči.
8. Slouží k výběru metody. Jinými slovy jakou operaci je nutné použít pro danou skladbu.

9. Nastavení počet minut
10. Natavení počet sekund
11. Pokud je vytvoření skladba toto tlačítko se zpřístupní. Po zmáčknutí se objeví v bloku *B* obrázek, který reprezentuje rozdíl mezi původní a vytvořenou variantou skladby
12. Pokud je vytvořená skladba, toto tlačítko se zpřístupní. Po zmáčknutí se objeví v bloku *B* obrázek, který reprezentuje vzdálenosti mezi rámci.
13. Pokud je vytvoření skladba toto tlačítko se zpřístupní. Po zmáčknutí se objeví ukládací dialogové okno. Zde se může vybrat kam se uloží obrázek, který je v bloku *B*.
14. Po zmáčknutí tlačítka se rozběhne program.
15. Panel průběhu ukazuje, v jaké části průběhu se nachází aplikace. Pokud je celý šedý, jako na obrázku, nachází se na začátku procesu. Pokud je celý zelený, znázorňuje, že proces je hotov.

5 Testování

Tato kapitola se zabývá testováním naimplementovaného programu, který je i není předmětem testování. Testování s uchazeči mělo nalézt optimální nastavení aplikace, pro vytváření variant skladeb. Během vytváření variant skladby jsem se všiml, že několik aspektů ovlivňuje chod programu.

5.1 Testování optimálního nastavení

V této časti, jsou předmětem testování všechny skladby, z kterých zanalyzuji data a navrhnu optimální nastavení.

5.1.1 Cíl testování

Test má za úkol nalézt optimální nastavení vstupů. Optimální nastavení se může lišit pro zkrácení a prodloužení. Pro testování je potřeba vytvořit testovací scénář, který bude sloužit k identifikování optimálního nastavení pro danou metodu. Dále k testu bude použito devět neznámých originálních skladeb. Skladby jsou odlišného žánru. K originálním skladbám jsou vytvořené varianty.

Test bude prováděn s participanty, kteří budou poslouchat neznámé skladby a budou je hodnotit, podle toho jak jim připadají přirozené.

5.1.2 Cílová skupina

Mojí cílovou skupinou byli studenti středních a vysokých škol ve věku 18 – 25 let, kteří se jakýmkoliv způsobem zajímají o hudbu, tedy i pouze poslechem. Chtěl jsem docílit k nalezení dvaceti studentů, ze kterých by bylo deset mužů a deset žen. Tento náročný požadavek se nakonec povedlo splnit. Tato práce se zabývá změnou délky hudby, proto jsem vybral účastníky, kteří jsou v každodenním kontaktu s hudbou. Pro zjednodušení hledání vhodného participanta byl vytvořen screener (viz příloha B)

5.1.3 Skladby

Použil jsem šest originální žárově rozdílných skladeb. Pokud možno neznámé pro všechny praticipanty. Použil jsem skladby s licencí CC, která znamená, že skladby je volně šiřitelné, a také skladby od mého vedoucího práce.

Vytvoření varianty skladby z originálů. Nejprve popíši parametry, které jsou pevně dané cesty k souborům, BPM, což je požadovaná délka a metoda. Vstupní cesta obsahuje cestu k souboru, který je požadován k načtení, Výstupní cesta reprezentuje cestu k souboru, který je požadován k vytvoření a měl by obsahovat podobné jméno jako vstupní soubor. BPM je pevně učené se vstupní skladbou, proto nesmí být zadaná jiná hodnota, než která má být. Požadovaná délka bude u každé metody jiná. U zkrácení bude cílový čas zvolen jako polovina délky originální skladby. U prodloužení bude požadovaný čas pevně zvolen o polovinu větší, než má originální skladba. Příklad: originální skladby má délku 1:30 minut, tedy polovina této délky je 0:45 minut, což reprezentuje požadovanou délku ke zkrácení skladby. Okamžitě je vidět, kolik bude požadovaná délka pro prodloužení skladby a cílový čas je 2:15 minut.

Toto testování se dělá především pro zjištění optimálního nastavení počtu rámců na dobu a počtu filtrů. Počet rámců na dobu bude pro každý filtr obsahovat 4, 6, 8 a 10 rámců na dobu. Počet filtrů, které budou vytvořený pro všechny rámce, jsou 15, 20 a 25. Vycházím z předpokladů, že filtrové pásmo, které se nejčastěji používá pro algoritmus MFCC (viz kap. 4.1.1) je 20 někdy i 40. Proto chci otestovat počet filtrů okolo hodnoty 20.

5.1.4 Průběh testování

Každému účastníkovi bylo vysvětleno, co se bude dít a uvedeno, že test není časově nikterak omezen. Před testem obdržel participant pre-test dotazník, který je určen pro upřesnění informací oblasti hudby a především v problematice se změnou délky hudby (viz příloha C). Dotazník byl vyplněn s mojí pomocí.

Poté účastník dostal seznam patnácti skladeb (viz příloha E). Tento seznam obsahuje název skladeb, které jsou různě namíchané. Dále tento seznam obsahuje další kolonku pro napsání čísla od jedné do pěti. Tyto hodnoty reprezentují, jak moc přirozeně zní skladba. Jednička je přirozené a pětka velmi nepřirozené. Také obsahuje testovací pokyny (testovací scénář). Participant postupuje podle testovacího scénáře. Tedy projde seznamy, upozorní na vzniklé chyby. Po opravení všech chyb, které mohly nastat, si přehraje skladbu. Následně jí ohodnotí příslušnou hodnotou. Je předpokládáno, že participant provede nad každou uvedenou skladbou vlastní ohodnocení aniž by byl ovlivněn předchozími skladbami. Účastník není limitován pořadím přehrávání skladeb, tedy každý si může zvolit vlastní pořadí.

Po doplnění poslední hodnoty jsem si vzal seznam skladeb. Poté jsem s účastníkem prošel post-test dotazník (viz příloha D), kde měl za úkol vybrat pouze jednu nelepší a nejhorší skladbu. A snažil jsem se, aby participant odešel ve stejném, nebo v lepším psychickém stavu než ve kterém přišel.

5.1.5 Výsledek testování

Zanalyzoval jsem výsledky ohodnocených seznamů. Provedl jsem to tak, že jsem vypočítal aritmetický průměr ohodnocených hodnot k příslušným skladbě.

Počet filtrů \ poče rámců	**4**	**6**	**8**	**10**
15	2,25	2,1	2,33	1,5
20	2,7	2	1,93	1,58
25	2,22	1,92	2,17	*1,48*

Tabulka 5.1: Výsledné hodnoty pro zmenšení skladby

Počet filtrů \ poče rámců	**4**	**6**	**8**	**10**
15	2,67	2,14	1,92	1,97
20	2,38	3.2	2,21	2,28
25	1,75	2,42	1,73	*1,57*

Tabulka 5.2: Výsledné hodnoty pro prodloužení skladby

Z tabulek Tabulka 5.**1** a Tabulka 5.**2** je patrné, že optimální nastavení pro metodu zkrácení délky hudby je nejvhodnější využít deset rámce a dvacet pět filtrů, totéž platí i pro prodloužení délky skladby.

5.2 Test výkonu

Při generování variant skladeb jsem si všiml, že tři aspektů, které ovlivňuj rychlost chodů aplikace. Jsou to:

1. Délka skladby
2. Počet rámců na jednu dobu
3. Počet filtrů

Druhý a třetí aspekt ovlivňuje program tak, že čím větší číslo je zadáno tím déle tato aplikace vykonává svojí práci.

5.2.1 Navrhnutí možných řešení

Rychlost vykonání aplikace by se dala zrychlit upravením algoritmu pro vytvoření matice podobností. A to způsobem, že se nebude každý prvek počítat znovu, ale bude vypočítána vzdálenost pro prvek v řádku a následně vypočítána vzdálenost bude přidána do sloupce na stejnou pozici jako v řádku. Jinými slovy: vypočítala by se pouze horní trojúhelníková část matice. Tím to způsobem by se dala vytvořit matice za poloviční čas.

Počet filtrů je spojený s algoritmem MFCC, který bohužel nelze zrychlit. Jediným způsobem jak lze zrychlit tento algoritmus je zadávání menších počtů filtrů, nebo rámců na jednu dobu, případně i kratší skladby.

6 Závěr

6.1 Splnění cílů

Tématem této bakalářské práce je vytvoření varianty skladby bez změny tempa skladby a výšky tónu. V souvislosti s ní se používá několik termínů, jejichž smysl není vždy úplně zřejmý. Tedy tyto pojmy pro nezasvěceného člověka mohou splývat. V příloze 0 jsou tyto pojmy vysvětlené. Vytváření variant skladeb není vždy jedno a to samé. Jsou dva způsoby, které jsou podrobně rozebrány v kapitolách 2.1 a 2.2.

Podstatnou částí práce je pochopení problematiky a navržení aplikace, ke které se využívá algoritmus, jenž je podrobně analyzován v kapitole 3. V této části nejsou pouze popsány bloky algoritmů, ale nacházejí, se zde i rozbory jak daný problém vyřešit, nebo jakým způsobem, postupovat aby se dostalo kýženého výsledku. V jedné podkapitole je též definovaná hlavička souboru typu WAV.

Implementace výše uvedeného algoritmu v programovacím jazyce C/C++ je uvedená v kapitole 4. V té jsou podrobně popsané části programu, které metody k čemu slouží, a i externí knihovny. Dále jsou zde popsány skutečně použité algoritmy. V poslední části této kapitoly je uvedeno, jak vypadá GUI, kde jsou popsané komponenty.

Testování optimálního nastavení vstupních dat je popsané v kapitole 5. Pomocí tohoto testování byly nalezeny tři varianty vhodných optimálních nastavení. Dále během generování variant skladeb byly odhalené některé nedostatky aplikace. K daným nedostatkům byly navrženy možná řešení daných nedostatků.

6.2 Shrnutí požadavků na program

Aplikace měla za úkol vytvořit varianty k zadané skladbě. Varianta skladby měla být zkrácená nebo prodloužená bez změny tempa skladby a výšky tónu. Nebyla nikterak omezená časovým trváním vytvoření varianty. Aplikace by měla mít grafické rozhraní, kde budou zobrazeny obrázky. Aplikace by měla běžet aspoň v operačním systému Windows

Naimplementovaná aplikace splňuje výše uvedené požadavky. Uživatel muže vytvořit varianty zadané sklady. Aplikace umožňuje:

- ukládaní vytvořených obrázků
- nastavit počet rámců na dobu
- nastavit počet filtrů pro danou skladbu
- nastavit metodu, kterou chce uživatel vykonat (zkrácení, nebo prodloužení)
- nastavit požadovanou délku k příslušné metodě, která je reprezentovaná cílovým časem
- načtení požadované skladby pomocí cesty k souboru, ke které je nastaven příslušný BPM
- vytvoření požadované skladby pomocí cesty, kde má být uložená.

6.3 Možnosti dalšího rozšíření

První věc, která by se měla provést při vytvoření rozšíření, je zrychlení algoritmu na vytvoření matice podobnosti (viz kap. 5.2.1). Mezi nápady, které slouží pro rozšíření dané aplikace, jsou např.:

- Přidaní počet dob na takt, po vhodné implementaci by se dosáhlo lepšího provedení prodloužení varianty nahrané skladby
- Vytvořit analýzu vstupních vzorků, kde by se zjistilo přesné BPM a počet dob na takt, což by vedlo k menšímu počtu zadávání vstupních parametrů
- Možnost vytvořit variantu i se změnou tempa
- Nečítání a vytváření všech typu skladeb. Typem je myšleno WAV, RAW, atd.
- Možné jsou i další varianty

Literatura

1. Aldebara, K. (2005). The MFCC. 1-14.
2. Černý, J. (2010). Z·kladnÌ grafovÈ algoritmy. 99-118.
3. Microsoft. (2015). *Dokumentace*. Získáno 10. 5 2015, z https://msdn.microsoft.com/cs-cz/library/
4. Microsoft. (1994). Multimedia. V M. Corporation, *New Multimedia Data Types and Data Techniques.* 12-21.
5. *Musical Time Theory & A Manifesto*. (2015). Získáno 10. 3 2015, z http://www.zeuxilogy.home.ro/media/manifesto.pdf
6. Practicalcryptography. (2019). *Mel Frequency Cepstral Coefficient (MFCC) tutorial*. Získáno 9. 5 2015, z http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/#deltas-and-delta-deltas
7. Rajamani, K., Lai, Y.-S., & Furrow, C. W. (200). An efficient algorithm for sample rate conversion from CD to DAT. *IEEE Signal Processing Letters* , 288-290.
8. S. Wenner, J. B.-H. (2013). Scalable Music: Automatic Music Retargeting and Synthesis. *Computer Graphics Forum, vol. 32, no. 2* , 345-354.
9. sengpielaudio. (2015). *BPM*. Získáno 2. 2 2015, z http://www.sengpielaudio.com/calculator-bpmtempotime.htm
10. Soundfile. (2015). *WAVE PCM soundfile format*. Získáno 10. 4 2015, z http://soundfile.sapp.org/doc/WaveFormat/
11. Tiwari, V. (2010). MFCC and its applications in speaker recognition. *International Journal on Emerging Technologies* , 19-22.
12. Todor Ganchev, N. F. (2005). Comparative Evaluation of Various MFCC Implementations on the Speakker Verification Task. 191–194.
13. Wenger, S., & Magnor, M. (2012). A Genetic Algorithm for Audio Retargeting. *Proc. {ACM} Multimedia 2012* , 705--708.
14. Wikipedia. (2015). *Mel scale*. Získáno 8. 5 2015, z http://en.wikipedia.org/wiki/Mel_scale
15. wikipedia. (2015). *Mel-frequency cepstrum*. Získáno 8. 5 2015, z http://en.wikipedia.org/wiki/Mel-frequency_cepstrum
16. Wikipedia. (2015). *Sample rate conversion*. Získáno 15. 3 2015, z http://en.wikipedia.org/wiki/Sample_rate_conversion
17. Wikipedia. (2015). *Window function*. Získáno 15. 4 2015, z http://en.wikipedia.org/wiki/Window_function

A. Rejstřík pojmů

ASCII formátování/kódování: ASCII je tabulka, která definuje znaky anglické abecedy a jiné znaky používané v informatice. Z historického pohledu jde o nejúspěšnější znakovou sadu, z které vyhází většina současných standardů pro kódování textu. Tabulka obsahuje tisknutelné znaky: písmena, číslice a jiné znaky. Např.: závorky, maticové znaky, atd. ASCII Kód je v současné době osmibitový, tedy obsahuje 256 platných znaků. Nicméně je stéle velmi malý, aby obsahoval veškeré národní abecedy.

Bitový posun: Bitový posun používá bitový operátor, s kterým se setkáme v různých programovacích jazycích. Procuje se s bitovými vzory, nebo binárními čísly. Umožňuje manipulovat s hodnotami přímo na úrovni bitů. Používá se především, že je na mnoha počítačích rychlejší než operace sčítání a odčítání. Když se pomocí bitových operací provádí násobení, nebo dělení je doba na vykonání operace s pomocí bitových operací výrazně kratší.

Doba: Doba představuje pevně daný časový interval např.: každé klepnutí metronomu představuje jednu dobu. A je základní jednotka rytmu. Určuje se podle not, např.: čtvrťová nota trvá jednu dobu.

Garbage collector (GC): GC je forma automatické zprávy paměti.

Graf: Jsou dva druhy orientovaný a neorientovaný graf.

- **Neorientovaný graf** je dvojce G = <V, E>, kde V je neprázdná množina vrcholů, nebo uzlů, a E je množina dvouprvkových množin vrcholů, tzv. (neorientovaných) hran. Hrany neorientovaného grafu nemají orientaci, tudíž výrazy (x, y) a (y, x) označují stejnou hranu
- **Orientovaný graf** je také dvojce G = <V, E>, kde V je neprázdná množina vrcholů a E je podmnožina uspořádaných dvojic vrcholů, tzv. (orientovaných) hran

Komprese: Je ztrátová a neztrátová. Může se použít při převodu analogového signálu na digitální, nebo pro úsporu místa v počítači. Většinou se používá ztrátová. Nicméně tak, aby při poslechu hudby nedošlo ke zkreslení.

Mel měřítko/stupnice: Měřítko Mel je založené na lidském sluchovém systému. Jinými slovy: to co člověk slyší. Používají se dva vzorce:

$$m = M(f) = 1125 \times ln\left(1 + \frac{f}{700}\right)$$

Rovnice A.1: Vzorec pro převod z frekvence na měřítko Mel

$$f = M^{-1}(m) = 700\left(e^{\frac{m}{1125}} - 1\right)$$

Rovnice A.2: Vzorec pro převod z Mel na frekvenci

Okénko: Okénko přestavuje číselnou hodnotu, která je vypočítána ze vzorce (viz Rovnice 3.8)

Open source: je počítačový software s otevřeným zdrojovým kódem. Otevřenost u tohoto pojmu znamená: technickou a legální dostupnost kódu. Obsahuje licenci software, která umožňuje při dodržení podmínek, uživatelům zdrojový kód využít.

Participant: Participant je účastník testu, který je proveden za účelem otestování nějakého řešení za pomocí osoby, jenž nebyli u vývoje daného řešení.

PCM: PCM je modulační metoda převodu analogového zvukového signálu na signál digitální. Princip spočívá v tom, že se pravidelně odečítají hodnoty signálů pomocí A/D převodníku. Hodnota je znázorněná binárně. Může se převést digitální signál na analogový pomocí D/A převodníku. Určujícím parametrem je vzorkovací frekvence, která určuje jemnost rozlišení jednotlivých hodnot.

Počet kanál: Počtem kanálů v hudbě je míněno o typ písničky jde. Např.: skladba typu mono obsahuje pouze jeden kanál, ale skladba typu stereo obsahuje dva kanály.

Rámec: Rámec je část doby, která je určená počtem rámců na dobu. Rámec může být veliký jako doba, ale nikdy nemůže být větší. Rámce se mohou překrývat.

Rytmus: Rytmus je časová složka v hudbě, ve které se střídají různé délky přízvučných a nepřízvučných dob. Pokrývá celou délku hudby, čili může v písničce být zopakován nějaký takt a pořád se nezmění rytmus.

Skladba/soubor typu WAV: Soubor obsahuje hlavičku, která je typická pro tento soubor a dále obsahuje vzorky, které reprezentují tóny. Soubor tohoto typu typicky ukládá vzorky nekomprimované.

Skladba/soubor typy RAW: Soubor může obsahovat hlavičku, ale typicky obsahuje pouze vzorky. Soubor tohoto typu ukládá vzorky pouze nekomprimované.

Sceener: Sceener může být např.: dotazník. Pomocí sceeneru se naleznou vhodný participaci pro testování.

Takt: Takt je krátký časový úsek hudební skladby, ve kterém se střídají přízvučné a nepřízvučné, stejně dlouhé doby. Například označení 3/4 znamená, že jeden takt má tři doby. Takty se dělí podle délky doby. Například jednodobé, dvoudobé, třídobé, čtyřdobé, atd. V moderní hudbě se nejčastěji setkáme se čtyřdobými takty.

Tempo: Tempo charakterizuje rychlost skladby v závislosti na čase. Tempo se označuje číselně, dané číslo je nazýváno BPM. Určuje délku taktu, to jest počet dob v taktu. Například 60 BPM odpovídá jedné době za sekundu.

Vhodné doby: Vhodné doby jsou ty, které jsou stejné nebo velmi podobné. Čili pokud je přehrajeme za sebou, uslyšíme dvakrát stejný tón.

Výška tónu: je základní charakteristikou tónu, je úměrná frekvenci tónu. Podle výšky tón je možné rozlišovat vysoké a hluboké tóny. Výšku tónu určuje přesně počet kmitů za sekundu.

Vzorek: Vzorek je hodnota nebo soubor hodnot v bodě času. Termín vzorek se převážně používá v diskrétních signálech a v digitální technice. Tedy pokud je analogový signál nahrán do počítačového souboru typy raw, wav, atd.

Vzorkovací frekvence: Frekvence definuje počet vzorků za jednotku času, obvykle je za jednu sekundu. Frekvence pro audio je typicky 44100Hz. Výpočet počtu vzorků na doby se vypočítá ze vztahu:

$$počet\ vroků\ na\ dobu = frekvence\ \times \frac{munuta}{tempo\ (BPM)} = frekvence\ \times \frac{60{,}000\ ms}{tempo\ (BPM)}$$

Rovnice A.3: Rovnice pro výpočet vzorků na doby

B. Screener

1. Pohlaví

- ☐ Muž
- ☐ Žena

2. Věk

- ☐ Méně jak 18 let
- ☐ 18 – 25 let
- ☐ Více než 25 let

3. Zaměstnání

- ☐ Plný úvazek
- ☐ Částečná úvazek
- ☐ Nezaměstnaný
- ☐ Student

4. Jak často posloucháte hudbu?

- ☐ Méně než jednou za týden
- ☐ Jednou až třikrát za týden
- ☐ Čtyřikrát až šestkrát za týden
- ☐ Každý den

5. Jaké máte zkušenosti s hudbou?

- ☐ Pouze poslouchám skladby
- ☐ Jsem muzikant
- ☐ Občas remixuji skladby
- ☐ A jiné: ……

C. Pre-test dotazníky

1. Kdy a kde posloucháte hudbu?

2. Jaký žánr máte nejraději?

3. Co si přestavíte podpojmy: změna délky skladby a music retargeting?

4. Jaký znáte programy na úpravu hudby?

D. Post-test dotazník

1. Jak se Vám líbilo testování?

2. Která písnička znale nejlépe?

3. Která písnička zněla nejhůře?

4. Které skladby jsou podle Vás originály?

E. Ukázkový testovací arch

Seznam úkolů

1. Pročtěte seznam skladeb a zkontrolujte se seznamem písniček ve složce
2. Upozorněte na chyby např.: chybí skladba, nebo určitou skladbu znám.
3. Přehrajte si skladbu
4. Ohodnoťte skladbu od 1 do 5. (1 znamená, že je plynula a poslechově normální, naopak znamená, že v písničce jsou velké nesrovnalosti)
5. Opakujte body 3 a 4, dokud nebudete mít všechny skladby ohodnocené

Seznam skladeb

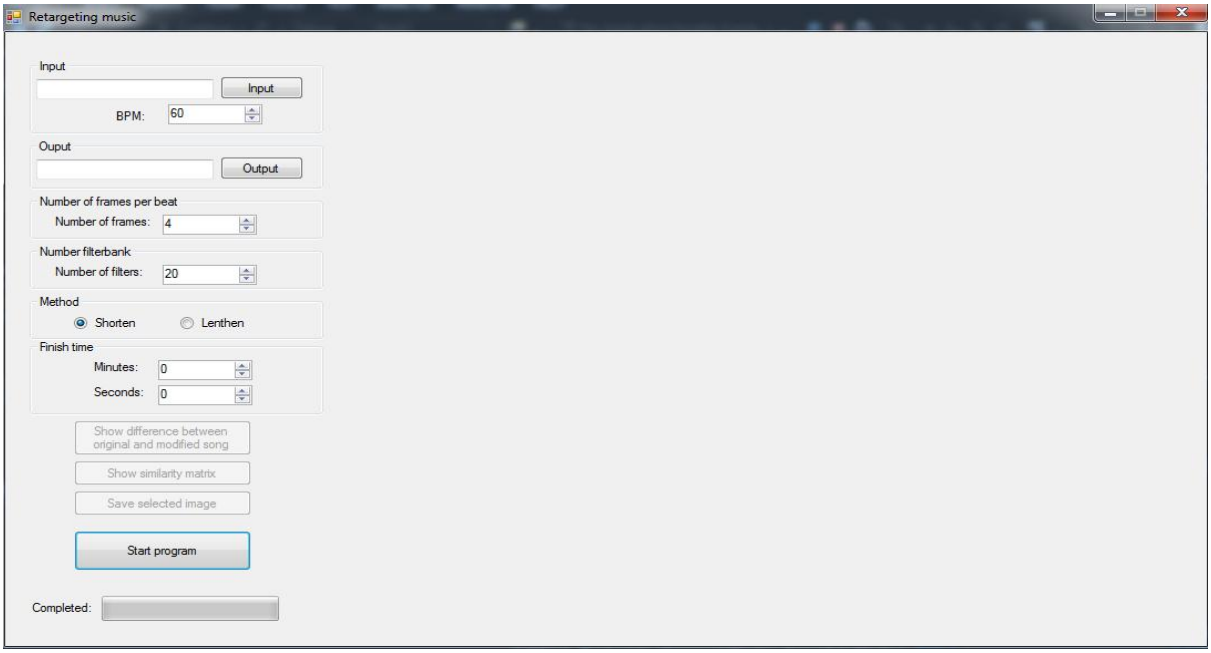
Název	Hodnocení
Critical_Dad-4-15-z53	
Critical_Dad-8-20-z53	
Critical_Dad-10-15-p2_39	
Critical_Dad-10-25-z53	
Derek_Clegg_-_07_-_Flip_Out-125BPM	
Derek_Clegg-4-25-z1_19	
Derek_Clegg-6-20-p3_58	
Derek_Clegg-6-20-z1_19	
Derek_Clegg-6-25-z1_19	
Derek_Clegg-8-20-p3_58	
Friction-8-15-p2_45	
genre-4-20-z1_47	
genre-6-25-z1_27	
JoshWoodward-6-15-p4_6	
JoshWoodward-10-25-z1_22	

F. Uživatelská příručka

Do programu se může nahrát skladba, které splňuje tyto požadavky: Je typu WAV, první doba začíná na nulté pozici. Pokud není zadávaná skladby typu WAV, aplikace vyhodí hlášku, že daný soubor nemá správný typ, nebo nepůjde otevřít. Pokud nebude nastavená první doba na první pozici, velké pravděpodobnosti dojde ke zkreslení nahrané skladby.

Výsledkem je vygenerovaná varianta skladby podle vstupních parametrů, kde ovšem nedojde ke změně tempa, ani výšky tónů.

Popis programu



Obrázek F.1: Ukázka aplikace

Aplikace vypadá jako na obrázku Obrázek F.**1**. Je rozdělena do dvou blok *A* a *B*. Oba bloky mají rozdílnou funkčnost.

Blok *A* slouží v této aplikaci jako menu, kde uživatel nastaví vstupní data. Nyní popíši nastavení vstupní parametrů a případy omezení:

První dílčí blok se nazývá *input* a obsahuje tři komponenty, které jsou: textové pole, počítací pole a tlačítko. Když uživatel klikne na tlačítko, objeví se otevírací dialogové okno, kde vybere skladby typu WAV. Po zavření otevíracího okna, a pokud je správně vybraný soubor. V textovém poli se objeví absolutní adresa k dané skladbě, a také v textovém poli se může měnit adresa. K příslušné skladbě se musí nastavit její BPM, které v aplikaci může reprezentována hodnotu od 50 do 200.

Druhý dílčí blok se pojmenován *output*, který obsahuje dvě komponenty textové pole a tlačítko. Když uživatel klikne na tlačítko, objeví se ukládací dialogové okno, ve kterém nasta-

ví adresu, kam chce uložit variantu skladby a pod jakým názvem. Nejde vytvořit jiný soubor než typu WAV. Jinými slovy: musí obsahovat koncovku „.wav.“

Třetí dílčí blok je nazývá *Number of frames per beat* (v českém jazyce: počet rámců na dobu). Čím větší hodnotu uživatel zvolí, tím výsledek bude přesnější, avšak proces (průběh) programu bude trvat déle. Zde jsem naimplementoval omezení, které je: zadané číslo musí být od 2 do 64 a musí být dělitelné dvojkou bezezbytku. Tedy můžou se nastavit pouze sudé hodnoty.

Čtvrtý dílčí blok je pojmenován *Number filterbank*, která obsahuje opět jedno počítadlo. Čím větší hodnotu uživatel zvolí, tím výsledek bude přesnější, avšak proces (průběh) aplikace bude trvat déle. Dále zde je naimplementováno omezení, které je následující: zadané číslo musí být od 1 do 128.

Pátý dílčí blok je pod názvem *Method,* v tomto bloku jsou dva radiobuttony. Tzn. že uživatel musí vždy zvolit jednu metodu ze dvou metod, samozřejmě podle toho, co chce provést.

Šestý dílčí blok můžeme ho nález pod pojmem *Finish time*. Uživatel nastaví požadovaný koncový čas, který reprezentuje délku skladby Maximální rozsah, který je možné zadat: 10:59 minut, a minimální rozsah, který může být zadán: 0:1 minut. Další omezení je pomocí zvolené metody. Např.: skladba je dlouhá 1:00 a uživatel zvolil metodu zmenšení, tak zadaný koncový čas musí být menší než jedna minuta.

Tlačítkem *start program* uživatel spustí aplikaci, která zkontroluje zadané vstupní parametry, jestli jsou správné. Pokud ano, spustí se program. Pokud naopak nejsou zadány správně uživatel je upozorněn.

Zablokovaná tlačítka se zpřístupní po dokončení procesu. Konec procesu aplikace se pozná na ukazateli průběhu, pokud je celý zelený, tzn. že program dokončil požadavek od uživatele. První dvě zablokovaná tlačítka slouží k přepínání mezi vytvořenýma obrázky. Posledním tlačítkem *save selected image* uživatel může uložit obrázek, který je znázorněn v bloku B. Blok B zobrazuje vždy jeden obrázek ze dvou vytvořených obrázků.

G. Obsah přiloženého CD

- Elektronická verze bakalářka práce ve formátu PDF, včetně zadaní
- Elektronická verze bakalářka práce v původním formátu
- Testovací balíčky pro participanty
- Vygenerované obrázky, které obsahují: spektrum matice podobnosti a rozdíl mezi originální skladbou a vytvořenou variantou skladby.
- Projekt, který obsahuje EXe i ko´dy
- README.txt