**UNIVERSIDADE FEDERAL DE JUIZ DE FORA**

**IGOR PIRES**

**LUIS AUGUSTO TOSCANO GUIMARÃES**

**HASHMUSIC**

**JUIZ DE FORA - MG**

**2014**

**SUMÁRIO**

[**1. ATIVIDADES REALIZADAS PELOS MEMBROS** 1](#_Toc391587105)

[**2. ALGORITMOS IMPLEMENTADOS** 2](#_Toc391587106)

[**3. FUNÇÕES DE HASH** 5](#_Toc391587107)

[**3.1. FUNÇÕES HASH DE DIRETÓRIO** 5](#_Toc391587108)

[**3.2. FUNÇÕES HASH DE NOME** 6](#_Toc391587109)

[**4. GRÁFICOS SOBRE ÍNDICES DE COLISÃO** 6](#_Toc391587110)

[**5. ANÁLISE SOBRE OS RESULTADOS OBTIDOS** 10](#_Toc391587111)

# **1. ATIVIDADES REALIZADAS PELOS MEMBROS**

O trabalho tem como objetivo criar um software que simule operações básicas (Armazenamento e busca) de carregamento de dados em dispositivos multimídia portáteis, como o *IPod Classic*, que possui baixo poder de processamento e grande capacidade de armazenamento e, ainda assim, consegue ser mais eficiente que outros dispositivos que possuem uma menor capacidade de armazenamento, para tal, será utilizado técnicas de hashing e de ordenação de registros, o que garante tempos melhores nestas operações. Para a realização do trabalho foi formado o grupo entre Igor Pires dos Santos e Luis Augusto T. Guimarães e a divisão de tarefas foi realizada em forma de papéis:

• Papel 1: Definição de 3 funções de hash para alocação em pastas, testes com as funções e geração dos gráficos

• Papel 2: Definição de 3 funções de hash para alteração do nome dos arquivos, testes com as funções e geração dos gráficos

• Papel 3: Códigos para inserção da música no sistema, leitura das tags ID3 dos arquivos, criação de um arquivo "music.dat", renomeação da música e alocação na pasta correta.

• Papel 4: Códigos para geração dos relatórios ordenados solicitados pelo usuário.

Para uma melhor distribuição de tarefas entre os integrantes do grupo, os papéis foram atribuídos da seguinte forma: A decisão sobre quais funções de hash seriam utilizadas tanto para alocação dos diretórios quanto para a alteração dos nomes dos arquivos ficaram a cargo dos dois integrantes, já a implementação das funções, o integrante Luis Augusto ficou responsável pela implementação das funções de diretório e o integrante Igor ficou responsável pela implementação das funções de alteração do nome dos arquivos; A implementação dos algoritmos de inserção de músicas no sistema, que envolvem a operação de ler tags ID3 dos arquivos MP3, a criação de um arquivo "music.dat" para armazenar informações das músicas importadas para o repositório, a alocação na pasta correta e a renomeação das músicas ficou a cargo do integrante Luis Augusto; A implementação de algoritmos para a geração de relatórios, a ordenação de registros e os gráficos sobre colisões ficou a cargo do integrante Igor.

# **2. ALGORITMOS IMPLEMENTADOS**

Os algoritmos em si serão colocados em apêndice, pois, são extensos e podem dificultar a leitura, este tópico apresentará como a interface do software funciona e a ideia básica do algoritmo por trás dela.

A linguagem de programação utilizada para a implementação dos algoritmos foi a linguagem Java. A escolha foi determinada pela fácil portabilidade do código entre os sistemas mais comuns do mercado como o Windows/Linux e também devido a melhor afinidade dos integrantes do grupo com a linguagem. Para a interface gráfica foi utilizada uma API própria da linguagem Java chamada Swing.

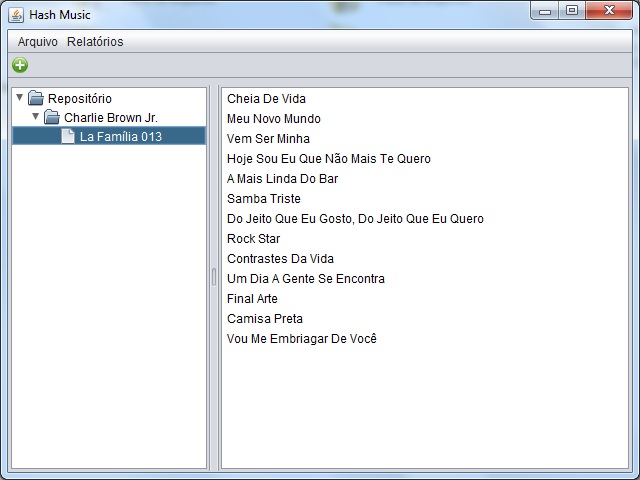


Figura: Tela inicial do software (HashMusic)

A interface básica do software apresenta a seguinte configuração: Um menu superior aonde há os menus "Arquivos" e "Relatórios", sendo que o menu "Arquivos" possui um simples submenu com a função de sair; Um botão que possibilita a importação de novas músicas para o repositório; uma árvore na lateral esquerda que lista a hierarquia de Artistas->Álbuns das músicas adicionadas no repositório; Uma lista de músicas que é preenchida ao selecionar uma hierarquia Artista->Álbum.

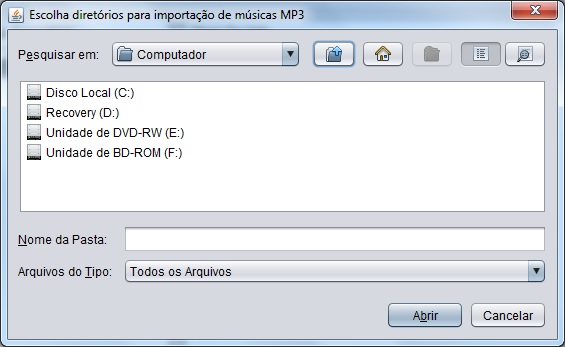


Figura: Janela para a escolha de diretórios para importação

Os algoritmos mais importantes do software são o de importação das músicas e o de ordenação. O de importação funciona da seguinte forma: Ao clicar no botão de importação é aberta uma janela de diálogo que possibilita selecionar um ou mais diretórios. Ao usuário selecionar os diretórios que deseja é aberta uma janela de progresso indeterminado. Neste momento o programa cria um nova thread que executa os processos de importação, evitando que a interface fique congelada enquanto a importação ocorre. O processo de importação se baseia nos seguintes passos: Para cada diretório que o usuário selecionou, um algoritmo encontra todos os arquivos, filtrando pela extensão ".mp3", da raiz do diretório e faz uma chamada recursiva para todos os seus subdiretórios, desta forma, dada uma pasta, eu irei encontrar todos os arquivos Mp3 contidos na raiz desta pasta e em seus subdiretórios. Ao término desta operação, cada Mp3 que foi encontrado é processado da seguinte forma: Primeiro o algoritmo verifica se o arquivo contém tags ID3 e se estão devidamente preenchidas (Artista, Álbum e Título), o próximo teste é verificar se a música já não existe no repositório, só é adicionado novas músicas ao repositório se elas não existirem. Passando pelos testes, um novo objeto MusicInfo (Classe que o sistema usa para guardar informações das músicas) é criado e preenchido com as informações da tag ID3. O próximo passo é gerar o hash para a alocação do diretório correto, para isto, existe uma função chamada getHashDir que recebe um MusicInfo como parâmetro e retorna a String do hash, definimos que a chave para o hash de diretório seria a combinação: Nome do artista + '#' + Nome do álbum, gerado o hash, existe uma verificação se o diretório existe ou não, caso não exista é criado um novo diretório, ou seja, a criação é sob demanda. Após este processo, é gerado o hash para a alteração do nome do arquivo, da mesma forma, há um método chamado getHashName que faz o processo de geração do hash, neste caso, definimos que a chave seria: Nome do artista + '#' + Nome do álbum + '#' + Título da música. Para facilitar a troca de funções para os testes e também manter o código modularizado, criamos interfaces como "IHashDirectory" que deve ser implementada por funções de hash para diretório e "IHashName" que deve ser implementada por funções de hash para alteração do nome dos arquivos. Desta forma, para alterar a função que gera o hash do diretório, basta trocar a instância da classe que implementa a interface "IHashDirectory", o mesmo é válido no caso das funções de alteração do nome do arquivo. Com o hash do diretório e do nome do arquivo, o algoritmo concatena os dois em formato de caminho de sistema de arquivos e copia a o MP3 para o lugar e com o nome corretos. Terminado esses processos para cada arquivo MP3 encontrado dos diretórios selecionados pelo usuário para a importação, é gerado um arquivo "music.xml" e um arquivo "importLog.xml". O arquivo "music.xml" guarda a lista de informações de todas as músicas do repositório, já o "importLog.xml" guarda informações de possíveis erros que tenham ocorrido durante o processo de importação, contendo qual arquivo MP3 ocorreu o problema e a mensagem de erro. A escolha pelo formato XML foi feita para simplificar a leitura e escrita destes arquivos.

Quanto aos algoritmos de ordenação era necessário que fossem estáveis por haver a necessidade de manter a posição original entre objetos de mesmo valor, portanto escolhemos o Merge Sort.

O Merge Sort funciona dividindo o objeto List pela metade até que este tenha tamanho um, de maneira recursiva, e quando estes são unidos (processo de Merge) sempre teremos listas ordenadas de suas duas metades então basta uni-las em outro objeto List<> percorrendo ambas as listas, comparando-as e adicionando sempre o menor objeto. Porém neste caso a comparação era entre objetos String e para que possamos fazer a comparação entre estes objetos é necessário usar o método String.compareToIgnoreCase(String) que compara os dois objetos e retorna um inteiro, caso seja menor do que zero a primeira String é maior do que a segunda e vice-versa.

O mesmo funcionou para os objetos MusicInfo, já que suas propriedades álbum e artista eram Strings, só houve diferença na ordenação de artista e álbum, pois somente quando os artistas eram iguais se comparava o álbum.

# **3. FUNÇÕES DE HASH**

Foram desenvolvidas seis funções de hash sendo três destas para diretórios e outras três para no nome do arquivo mp3.

## **3.1. FUNÇÕES HASH DE DIRETÓRIO**

O primeiro hash de diretório foi o hash de divisão que soma todos os caracteres da String em um inteiro e pega o módulo deste inteiro com um número primo e em seguida pega o módulo desta divisão com o tamanho da tabela, retornando o este segundo módulo em String.

O segundo hash utilizado foi o hash de extração, que divide o String em duas metades, concatena a segunda metade com a primeira, ou seja, inverte suas metades, em seguida pega os cinco caracteres do meio dessa nova String, soma estes e retorna o módulo da soma com o tamanho da tabela em String.

Por último foi utilizado o hash java, que pega cada caractere da chave e multiplica-os por uma potência decrescente de 31 e soma os resultados, supondo uma chave de 3 caracteres a soma se daria da seguinte forma:

* O 1º caractere é multiplicado por 31^2
* O 2º caractere é multiplicado por 31^1
* O 3º caractere e multiplicado por 31^0

Ao fim da soma é retornado o módulo desta pelo tamanho da tabela em String.

## **3.2. FUNÇÕES HASH DE NOME**

O primeiro hash de nome foi o hash da multiplicação que primeiro realiza a soma de todos os caracteres da String e os multiplica pelo resultado da divisão entre o tamanho da String e um número primo retornando o valor da multiplicação em String.

O segundo hash utilizado foi o hash de meio-quadrado que soma todos os caracteres e eleva a soma ao quadrado e caso o resultado seja maior que o valor máximo os caracteres máximos do meio são selecionados e retornados em String caso contrário retorna-se apenas o resultado da potência.

E por último foi utilizado o hash pires que se assemelha ao hash do meio-quadrado, pois novamente realiza a soma e eleva o seu resultado ao quadrado, porém o resultado é colocado na base vinte para que o resultado tenha um tamanho ainda menor, caso o resultado ainda seja maior que o valor máximo para hashings de nome, novamente, os caracteres máximos do meio são selecionados e retornados em String caso contrário retorna-se apenas o resultado da potência na base vinte.

# **4. GRÁFICOS SOBRE ÍNDICES DE COLISÃO**

Primeiramente foram analisados os hashs de diretório, começando pelo hash da divisão. No qual percebemos que este hash possui um número crescente de colisões, chegando a ter 402 colisões em 500 músicas, ou seja, possui um valor médio de colisões muito alto, entretanto suas colisões são bem espalhadas.

Para o hashing de extração o mesmo vale, possuindo 410 colisões com 500 músicas, ou seja, um valor médio de colisões superior ao da divisão e estas colisões ainda são menos distribuídas que a do hashing de divisão.

Em seguida foi analisado o hash java, que foi considerado o pior, pois possui o nível mais elevado de colisões e estes ainda menos distribuidos, chegando a distribuição média de 17,73.

Em seguida foram analisados os hashs de nome, começando pelo hash de multiplicação. Este hash surpreendeu por ser extremamente simples e não ter apresentado nenhuma colisão.

Já o hash de meio-quadrado apresentou o valor máximo de 26 colisões sendo estas bem distribuidas tendo 2 colisões com o mesmo nome (por isso 1,04 na distribuição média de colisões). E sua quantidade média de colisões totais foi menor do que 10%.

Já o hash pires obteve exatamente as mesmas colisões que o hash meio-quadrado, por estes funcionarem da mesma forma, tendo como diferença apenas a base em que os resultados são exibidos.

**5. ANÁLISE SOBRE OS RESULTADOS OBTIDOS**

Ao final do trabalho obtivemos um programa capaz de incluir bibliotecas de música, ordená-las rapidamente e dispensamos a necessidade de armazenar o caminho para cada uma delas.

Quanto à interface o programa possui poucos botões, tornando-se uma interface fácil de ser utilizada, apresentando as músicas separadas por artista e álbum e relatórios de colisão.

Quanto as funções de hashing, estas tiveram um bom funcionamento, visto que a intenção dos hashs de diretório era agrupar em pastas as músicas que tinha artista e álbum semelhantes, por isso o grande número de colisões, e o hash de nome selecionado não apresentou nenhuma colisão.