UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

EDUARDO SEMKIW XAVIER

JONATHAN DA SILVA BATISTA

CRIAÇÃO DE UM BANCO DE DADOS NÃO RELACIONAL A PARTIR DE INFORMAÇÃO EXTRAÍDA DE TEXTOS

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2018

EDUARDO SEMKIW XAVIER

JONATHAN DA SILVA BATISTA

CRIAÇÃO DE UM BANCO DE DADOS NÃO RELACIONAL A PARTIR DE INFORMAÇÃO EXTRAÍDA DE TEXTOS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, do Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. André Koscianski

PONTA GROSSA

2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa

Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

CRIAÇÃO DE UM BANCO DE DADOS NÃO RELACIONAL A PARTIR DE INFORMAÇÕES EXTRAÍDA DE TEXTOS

por

JONATHAN DA SILVA BATISTA

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 29 de maio de 2018 como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado. A Folha de Aprovação assinada encontra-se arquivada na Secretaria Acadêmica.

_____
Prof. Dr. André Koscianski
Orientador

_____
Prof. Dr Erikson de Moraes
Membro titular

_____
Prof. Dr. Tarcizio Alexandre Bini
Membro titular

_____
Profª. Dra Helyane Bronoski Borges
Responsável pelo Trabalho de Conclusão de Curso

_____
Prof. Dr André Pinz Borges
Coordenador do curso

- A folha de Aprovação assinada encontra-se arquivada na Secretaria Acadêmica

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

CRIAÇÃO DE UM BANCO DE DADOS NÃO RELACIONAL A PARTIR DE INFORMAÇÕES EXTRAÍDA DE TEXTOS

por

EDUARDO SEMKIW XAVIER

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 29 de maio de 2018 como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado. A Folha de Aprovação assinada encontra-se arquivada na Secretaria Acadêmica.

_____
Prof. Dr. André Koscianski
Orientador

_____
Prof. Dr Erikson de Moraes
Membro titular

_____
Prof. Dr. Tarcizio Alexandre Bini
Membro titular

_____
Profª. Dra Helyane Bronoski Borges
Responsável pelo Trabalho de Conclusão de Curso

_____
Prof. Dr André Pinz Borges
Coordenador do curso

- A folha de Aprovação assinada encontra-se arquivada na Secretaria Acadêmica

Dedicamos este trabalho às nossas famílias, pela motivação e confiança nos dada.

AGRADECIMENTOS

Agradecemos primeiramente a Deus, por nos conceder a dedicação e coragem de seguir nos estudos e ter a oportunidade de fazer um trabalho de conclusão de curso. Também agradecemos nossas famílias por estarem ao nosso lado e que sempre nos conduziram no caminho do aprendizado.

Agradecemos ao nosso orientador Prof. Dr. André Koscianski, pela sabedoria com que nos guiou nesta trajetória.

Enfim, a todos os que por algum motivo contribuíram para a realização deste trabalho.

RESUMO

BATISTA, Jonathan da Silva; XAVIER, Eduardo Semkiw. **Criação de um banco de dados não relacional a partir de informação extraída de textos.** 2017. 38 f. Trabalho de Conclusão de Curso. Tecnologia em Análise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2017.

As informações e dados estão atualmente concentradas em enorme quantidade dentro de arquivos de texto. E o fato da maior parte das informações tratadas por humanos estarem em textos não estruturados, justifica a importância de se extrair dados. O objetivo deste trabalho é desenvolver uma aplicação capaz de analisar e extrair informações úteis a partir de arquivos PDF. A aplicação irá utilizar uma ferramenta externa para converter PDF e realizar a extração do conteúdo em arquivo de texto. Logo em seguida irá efetuar uma busca por padrões, como endereços e datas. Finalmente fará o armazenamento dos dados tratados em um banco de dados NoSQL. Visto que a extração de informação em arquivos PDF gera uma grande quantidade de dados, surge a necessidade de apoio automatizado ao usuário, devido dificuldade de se realizar isso de forma totalmente manual.

Palavras-chave: Extração de Informação. Mineração de Textos. Arquivos PDF.

ABSTRACT

BATISTA, Jonathan da Silva; XAVIER, Eduardo Semkiw. **Creation of a non-relational database from information extracted from texts**. 2017. 38 f. Undergraduate Final Work. Technology in Analysis and Development of Systems - Federal Technological University of Paraná. Ponta Grossa, 2017.

Information and data are currently concentrated in huge amounts within text files. And the fact that most information treated by humans is in unstructured texts justifies the importance of extracting data. The purpose of this paper is to develop an application capable of analyzing and extracting useful information from PDF files. The application will use an external tool to convert PDF and extract the content into text file. It will then search for patterns, such as addresses and dates. Finally, it will store the treated data in a NoSQL database. Since the extraction of information in PDF files generates a large amount of data, there is a need for automated support to the user, due to the difficulty of doing so in a totally manual way.

Keywords: Information Extraction. Text Mining. PDF files.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

DRM	Digital Rights Management
GNU	Gnu's Not Unix
GPL	General Public License
HTML	HyperText Markup Language
IBM	International Business Machines
JSON	Java Script Object Notation
kB	Kilobyte
KDT	Knowledge Discovered in Texts
LCS	Largest Common Subsequence
LGPL	Lesser General Public License
LZW	Lempel-Ziv-Welch
NoSQL	Not only Structured Query Language
PDF	Portable Document Format
PNG	Portable Network Graphics
SQL	Structured Query Language
UTFPR	Universidade Tecnológica Federal do Paraná

SUMÁRIO

1 INTRODUÇÃO

Em praticamente todos os setores de atividade humana, a informação é um elemento essencial. Somos cercados por sinais, ícones, textos, números, contratos. Dentro da computação, na maioria das vezes, dados se apresentam de maneira não estruturada, ou seja, são dados sem uma organização para serem classificados e recuperados.

Encontrar padrões buscando informações em uma base estruturada, de modo geral, é mais fácil do que em dados não estruturados. Para linguagens como SQL (*Structured Query Language*), que é baseada em álgebra linear, os dados estruturados são mais fáceis de serem tratados, pois permitem sua manipulação e consulta. Entretanto, seres humanos trabalham normalmente com informação não estruturada, que basicamente são documentos de texto.

Assim sendo, para os dados não estruturados ou textuais, é necessário processamento para coleta e classificação, para posterior arquivamento em uma base de dados. Esse procedimento precisa de apoio automatizado, pois é árduo, complicado e fatigante, aspectos que o deixam exposto a erros se feito manualmente. Uma das áreas que estuda esse problema é a mineração de textos que, essencialmente é um Processo de Descoberta de Conhecimento, que utiliza técnicas de análise e extração de dados a partir de textos, frases ou apenas palavras.

> Envolve a aplicação de algoritmos computacionais que processam textos e identificam informações úteis e implícitas. Esses textos normalmente não poderiam ser recuperados utilizando métodos tradicionais de consulta, pois a informação contida neles não pode ser obtida de forma direta por que estão em formato não estruturado (AMBROSIO; MORAIS, 2007).

Uma das aplicações interessantes de mineração de textos é a extração de dados que tenham valor estratégico e econômico. Um caso especial é o de anúncios classificados, carros, empregos e residências. A disponibilidade desses dados em um formato que o computador possa utilizar, abre várias possibilidades, como filtrar melhor pesquisas de informação na Internet, gerar comparações automáticas entre valores de produtos, e ainda estudos e análises, como por exemplo em simulação computacional.

O objetivo deste trabalho é ajudar a cobrir essa lacuna, entre massas de dados desestruturados e aplicações que possam utilizá-los. O trabalho consiste em

desenvolver uma aplicação que tenha a capacidade de extrair informações a partir de textos gravados em formato PDF, gerando como saída arquivos com uma estrutura organizada.

O problema em questão caracteriza-se em especial pela necessidade de se identificar e destacar a presença de endereços e preços em uma série de classificados de imóveis no formato PDF. Após localizar essas informações, a aplicação exibirá os resultados encontrados ao usuário, para que ele possa detectar a presença de endereços parecidos, como nome de rua pela metade, entre outras similaridades.

Após a análise do usuário a esses dados extraídos, toda informação será armazenada em arquivos JSON, para a importação em uma base de dados NoSQL. Essa base de endereços poderá ser usada em alguns projetos que estão sendo iniciados no PPGCC (Programa de Pós-Graduação em Ciência da Computação) da UTFPR, campus Ponta Grossa. A descrição de tais projetos está fora do escopo do presente trabalho.

Foi definido junto ao orientador que a interação com o usuário será via console. Consideramos a extração das informações dos arquivos PDF, a busca pelas expressões similares e o armazenamento das informações em uma base de dados de forma organizada, mais prioritário em relação ao desenvolvimento de uma interface gráfica.

Para expormos os assuntos tratados neste trabalho de forma mais didática, dividimos esses assuntos por capítulos. O capítulo 2 descreve uma visão geral sobre a mineração de textos, com um apanhado geral sobre algoritmos que fazem a comparação aproximada de *strings*, explanando algumas bibliotecas da linguagem C que fazem extração de textos de documentos no formato PDF e trazendo um panorama geral sobre o banco de dados NoSQL MongoDB, base de dados que será utilizada para armazenamento das informações extraídas dos arquivos PDF.

O capítulo 3 fala sobre a implementação desse trabalho, detalhando a problemática para a qual está sendo realizada essa pesquisa, o cenário onde se encontra tal problema e a motivação que originou este trabalho. Ainda é detalhado o desenvolvimento, apresentando por fim os resultados obtidos e os ganhos trazidos pela aplicação.

E por último no capítulo 4, é explanado a importância do desenvolvimento deste trabalho para nós, enquanto alunos, e para os usuários que utilizarão a

aplicação desenvolvida. Demonstrando o aumento da produtividade e redução de possíveis erros.

2 FERRAMENTAS E TECNOLOGIAS

2.1 VISÃO GERAL

Os sistemas de recuperação de informação têm como propósito proporcionar ao usuário uma busca pelo conhecimento de forma fácil e rápida. Para que tal objetivo se cumpra, essas aplicações aplicam uma gama de recursos computacionais para auxiliá-los nesta difícil tarefa, principalmente se a base de dados, onde será efetuada a busca, for puramente textual e desestruturada. Este capítulo apresenta ferramentas e tecnologias que contribuem com esse processo, que podem ser usadas nas mais diversas áreas de conhecimento.

2.2 MINERAÇÃO DE TEXTOS

Também conhecida como Descoberta de Conhecimento em Textos, a mineração de textos é uma técnica que utiliza a análise e extração de dados a partir de textos, como forma de contribuição na busca por padrões específicos em grandes volumes de documentos. Neste sentido, este mecanismo torna este processo muito mais ágil e eficiente. Será abordada nesse capítulo, a definição de mineração de textos, assim como alguns algoritmos de comparação de *strings*.

2.2.1 Visão Geral

Informações armazenadas em bancos de dados oferecem maior facilidade para serem tratadas por procedimentos computacionais, pois existem linguagens bem definidas como a SQL (*Structured Query Language*), que se originou através de um projeto da IBM, gerando a linguagem denominada inicialmente de SEQUEL (acrônimo para *Structured English Query Language*). “Mais tarde, por razões legais, a denominação foi alterada para SQL simplesmente” (HUTH, 2002). Permitindo assim consultas e manipulações de forma mais coerente e precisa.

A manipulação de dados não estruturados exige várias etapas: coleta, manipulação e armazenamento, assim, esses dados precisam de recursos computacionais adicionais. Pelo fato de muitas informações estarem armazenadas

em formato texto, “acredita-se que as técnicas de mineração de textos possuam um grande valor comercial” (BARION; LAGO, 2008).

> A Mineração de Textos, também conhecida como Descoberta de Conhecimento em Textos ou da sigla em inglês KDT (Knowledge Discovered in Texts), refere-se ao processo de extração de informação útil (conhecimento) em documentos de textos não-estruturados (BARION; LAGO,2008).

O procedimento de mineração de textos reúne muitas técnicas como as de recuperação e extração de informação, métodos de *data mining* e mineração de textos. Para Aranha e Passos (2006),a mineração de textos também pode ser entendida como “um conjunto de métodos usados para navegar, organizar, achar e descobrir informação em bases textuais”. Pode ser vista como uma extensão da área de *data mining*, focada na análise de textos. Segundo certas estimativas (BARION; LAGO, 2008) cerca de “apenas 20% das informações são usadas para manipulação de tomada de decisão dentro das empresas”.

> “A mineração de textos surgiu a partir da necessidade de se descobrir, de forma automática, informações potencialmente úteis com padrões e anomalias em textos” (ARANHA; PASSOS, 2006).

Com muitas informações armazenadas em forma de texto, as técnicas de mineração de textos são muito importantes para a recuperação do conhecimento implícito nestes documentos. Este processo difere de um mecanismo de busca, visto que, na busca, o usuário já tem conhecimento do que deseja encontrar, enquanto na mineração de textos se busca a descoberta de informações desconhecidas.

A técnica de mineração de textos pode ser aplicada em fóruns de discussão (BEHAR et al., 2010), sendo usada para desempenhar um diagnóstico qualitativo das colaborações textuais por alunos em fóruns de discussão. Isso permite identificar quais discentes redigiram mensagens que contemplam conceitos relativos ao tema da discussão, e quais não o fizeram. Desta forma, é possível ter subsídios para investigar quais alunos necessitam de maior auxílio, e motivá-los para discutir os conceitos importantes que fazem parte do tema em debate.

Outro exemplo é o uso da mineração de textos para identificar, a partir da descrição dos serviços prestados, notas emitidas incorretamente (MADEIRA, 2015). A partir dessas informações, pode-se respaldar um melhor planejamento de

fiscalizações, aplicando essa técnica sobre a grande quantidade de dados são gerados pelas notas fiscais eletrônicas (MADEIRA, 2015).

2.2.2 Algoritmos Para Comparação Aproximada De *Strings*

Existem inúmeras formas de executar essa comparação, mas independente de qual seja a mais rápida ou a mais eficiente, todas tem o objetivo de trazer possíveis soluções e não apenas soluções de resultado exato, fazendo que esses algoritmos se tornem propícios para tratar textos não estruturados. Nesta sessão serão apresentados alguns exemplos de algoritmos por busca aproximada, destacando a forma como cada um trata os textos.

2.2.2.1 *Largest common subsequence* (maior subsequência em comum)

O algoritmo *Largest Common Subsequence* realiza a busca por semelhanças entre *strings* procurando a maior *substring* contida entre as duas *strings*. Quanto maior a semelhança entre elas, maior subsequência em comum há entre elas. “A essência do problema de encontrar a maior subsequência comum entre os elementos de um conjunto de sequências de símbolos de qualquer natureza é determinar o grau de similaridade existente entre elas” (PAOLI, 2016).

Exemplificando, seu funcionamento seria da seguinte forma: as *strings* 'abacate' e 'abacaxi' possuem 'abaca' (5 caracteres) em comum. Já as *strings* 'uva' e 'pera' possuem em comum apenas o 'a' (1 caractere). Portanto 'abacaxi' é mais parecido com 'abacate' do que 'uva' é parecida com 'pera'.

2.2.2.2 Distância *levenshtein*

Criado pelo cientista russo Vladimir Levenshtein, que considerou esta distância já em 1965. “A distância *Levenshtein* ou distância básica de edição entre duas sequências de caracteres (*strings*) é dada pelo número mínimo de operações necessárias para transformar a *string* A na *string* B” (VASEL, 2007).

Essa distância é muito utilizada para aplicações que precisam determinar quão semelhantes são duas *strings*, como é, por exemplo, o caso com os verificadores ortográficos (DA SILVA, 2007).

A Distância *Levenshtein* leva em consideração o número mínimo de operações para que uma *string* seja igual a outra. Por exemplo, um endereço em determinado momento é escrito como: Rua Alberto Matos, 21. Em outro momento é mencionado como: Rua Alberto Matos, N 21. Neste caso, ao comparar as duas *strings*, obtemos o valor 2. Para alcançar uma igualdade entre as *strings* será necessário colocar a letra “N” e mais um espaço, logo duas alterações.

2.2.2.3 Monge elkan distance

> A Monge Elkan Distance é na verdade uma função de distância híbrida, pois trabalha tanto com *tokens*, dividindo as *strings* de entrada em palavras separadas, quanto com os caracteres individuais dessas palavras. A ordem dos *tokens* não é levada em consideração no cálculo da similaridade (DA SILVA, 2007).

Ela processa as *strings* de entrada as partindo em *tokens*, e faz a busca procurando similaridades, tentando descobrir qual *token* na segunda *string* melhor corresponde a cada *token* na primeira *string*. Ao final, é retornada uma média dos valores de similaridade obtidos para cada *token* da primeira *string*.

Esta função é eficiente para comparar textos longos, sendo sua principal aplicabilidade o casamento de registros (DA SILVA, 2007).

2.2.3 Algoritmo Bitap

O algoritmo Bitap calcula correspondência aproximada baseado em operações bit a bit, usando a distância Levenshtein. Ele determina se existe uma correspondência “aproximadamente igual” no texto dado, onde a igualdade aproximada é definida pela distância de *Levenshtein*(RABBANY, 2010).

Também conhecido como *shift-or*, *shift-and* ou Baeza-Yates-Gonnet, o algoritmo Bitap busca relações aproximadas em conjuntos de *strings*, dizendo se um determinado texto contém uma *substring* que é "aproximadamente igual" a um padrão

dado, ou seja, se a *substring* e o padrão estiverem dentro de uma determinada distância k um do outro, então o algoritmo considera-os iguais.

Levando em consideração que o algoritmo Bitap tem um limite de 31 caracteres em seu padrão de entrada (RABBANY, 2010), podemos inferir que, ele funciona melhor em padrões de entrada pequenos a um comprimento constante, e também prefere entradas com um pequeno conjunto de palavras. Esse limite é facilmente aumentado para 64 usando inteiros longos na implementação.

2.3 EXTRAÇÃO DE TEXTOS DE PDF

Esta sessão irá apresentar bibliotecas que tratam extração de textos em documentos PDF, bem como ferramentas e aplicações com a mesma finalidade. Vale ressaltar que será usado a biblioteca Xpdf para extração dos dados para o desenvolvimento do presente trabalho.

2.3.1 Biblioteca XPDF

É um visualizador de arquivos PDF com código aberto atuando sobre a licença GPL, que é a licença de uso para *software* livre e de código aberto, composto pelo Xpdf Reader que é seu editor próprio, bem como uma vasta biblioteca de funções para manipulação de arquivos PDF.

Esse *software* é capaz de decodificar o algoritmo LZW (Lempel-Ziv-Welch), que é um algoritmo de compressão “que se propõe a reduzir significantemente a redundância de codificação” (BEZERRA, 2009) e também ler arquivos PDF criptografados. Em sua versão oficial, respeita o DRM *(digital rights management*) dos arquivos que, em ambientes digitais, Busnello (2008) diz que é “uma das maneiras de se coibir a produção de cópias não autorizadas” prevenindo a conversão, cópias ou impressões de arquivos protegidos.

A GLYPH & COG, LLC foi formada em 2002 para apoiar e comercializar o projeto Xpdf, possuindo todos os direitos no código Xpdf e é dedicada a continuar o projeto em código aberto, além de oferecer outros *software*s comerciais e serviços de consultoria licenciados. Lançado pela primeira vez em 1995, foi escrito e ainda é desenvolvido por Derek Noonburg.

O Xpdf também é usado como base para inúmeros *softwares* editores de arquivos PDF que utilizam seus recursos. Podendo citar alguns como, BePDF no sistema operacional BeOS, !PDF no sistema Risc OS, PalmPDF no Palm OS e o Poppler que é uma biblioteca que renderiza arquivos PDF baseada no código Xpdf.

2.3.2 XPDF Reader

XpdfReader é um visualizador de PDF gratuito, mas não *open source*, que “é um método de desenvolvimento de software que aproveita todo o poder de distribuição, revisão e transparência do processo” (GONÇALVES, 2007). Estando na sua versão 4.00 lançada em 10 de agosto de 2017, que inclui alguns recursos extras não encontrados no visualizador Xpdf de código aberto. Já sendo compilado e pronto para uso, disponível para plataforma Linux e Windows, ambos em 32 e 64 bits.

2.3.3 XPDF Toolkit

O conjunto de ferramentas Xpdf de código aberto também inclui vários recursos de linha de comando que executam várias funções em arquivos PDF, sendo eles:

- pdftotext (converte PDF para arquivo de texto),
- pdftops(converte PDF para PostScript),
- pdftoppm (converte páginas PDF para arquivos de imagem netpbm),
- pdftopng (converte páginas PDF para arquivos de imagem PNG),
- pdftohtml (converte PDF para HTML),
- pdfinfo (extrai meta dados do PDF),
- pdfimages (extrai imagens brutas de arquivos PDF),
- pdffonts (lista as fontes usadas em arquivos PDF) e
- pdfdetach (extrai arquivos anexados de arquivos PDF).

Dentre os itens citados, as funções mais notáveis dentro do xpdf são a pdftotext, pdftopng, pdftohtml, pdffonts e pdfinfo. Sendo que a ferramenta pdftotext será utilizada para auxiliar no desenvolvimento da aplicação em questão neste trabalho. Sua sintaxe básica é: "pdftotext [options] [PDF-file [text-file]]". Sendo o termo pdftotext, a chamada do executável; o parâmetro [options] é o comando de

configuração e o último parâmetro [PDF-file [text-file]], que são respectivamente o caminho do arquivo PDF para conversão e o caminho e nome do arquivo texto convertido.

2.3.4 XPDF Writer

É uma biblioteca repleta de funções para criar, analisar e manipular arquivos PDF. Sendo seu download gratuito para usos em diversas finalidades. Foi desenvolvido pela Hummus PDF, de forma que não desperdice memória à medida que o arquivo cresce.

A biblioteca possui um conjunto de recursos de alta qualidade e usabilidade, alguns deles são: criar páginas, integração de imagens de diversos formatos, suporte ao tipo de texto *unicode*, análise de conteúdo de um arquivo PDF e modificação do mesmo.

Para suportar a condição de usar recursos mais sofisticados a biblioteca PDF Writer foi desenvolvida para ser muito extensível. Com todos os códigos fonte das funcionalidades e recursos disponíveis, é simples implementar mais recursos ou modificar os métodos existentes.

O projeto está licenciado no Apache 2.0 encontrando-se atualmente em sua versão 3.90. A biblioteca é escrita em C++ e também considerada *thread-safe*, ou seja, uma função é considerada *thread-safe* se puder ser invocada com segurança por vários *threads* ao mesmo tempo. Entretanto, se uma função não é *thread-safe*, então não podemos chamá-la a um *thread* enquanto ele está sendo executado em outro *thread* (KERRISK, 2010).

2.3.5 PoDoFo

PoDoFo é uma biblioteca que trabalha com o formato de arquivo PDF. O nome vem das duas primeiras letras de cada palavra da sigla PDF *(Portable Document Format).*

A biblioteca PoDoFo é desenvolvida em C++ gratuita e portátil que integra uma grande quantidade de classes para analisar arquivos PDF e modificar seus

conteúdos na memória. As mudanças podem ser gravadas de volta em disco com facilidade.

A classe que analisa os arquivos também pode ser usada para extrair informações de um arquivo PDF. Além de analisar o PoDoFo, inclui também classes muito simples para criar seus próprios arquivos PDF. Todas os recursos são muito bem documentados, por esse motivo, é simples desenvolver uma aplicação que usa classes do PoDoFo. Atualmente, pode-se citar como principais ferramentas:

- podofoencrypt, que criptografa qualquer arquivo PDF e permite definir permissões de PDF;
- podofoimgextract, que extrai todas as imagens de um determinado arquivo PDF;
- podofoimpose, sendo uma poderosa ferramenta de imposição de PDF, colocando páginas de um ou mais PDFs em páginas de um novo PDF, aplicando dimensionamento e posicionamento;
- podofopdfinfo, fornece algumas informações básicas sobre um PDF;
- podofotxt2pdf, converte um arquivo de texto em um PDF e
- podofotxtextract, que extrai todo o texto de um arquivo PDF.

Também existe o PoDoFo Browser, que é um programa *desktop* que possui todas funcionalidades citadas e o código da biblioteca PoDoFopode ser compilado com sucesso nas plataformas Unix, Mac OS X e Windows. Lançado sob a GNU *Lesser General Public License* (LGPL), está atualmente na sua versão 0.9.4.

2.3.6 LibHaru

A LibHaru é uma biblioteca de plataforma aberta, *open source*, para gerar arquivos PDF. Neste momento, o libHaru ainda não suporta a leitura e edição de arquivos PDF. Sendo assim, é uma biblioteca simples e de certo modo, fácil de implementação. Pode-se citar algumas das suas funcionalidades: a geração de arquivos PDF com linhas, textos e até imagens; esboços, anotações de texto e links; compressão de documentos com *deflate-decode,* que em suma é um algoritmo de compactação de dados que usa uma “combinação do algoritmo LZ77 e da codificação Huffman” (DEUTSCH, 1996); inclusão de imagens no formato jpeg e png; encriptação

de arquivos PDF. LibHaru está escrito em C, e suporta a maioria dos sistemas operacionais, como Windows e Linux.

A LibHaru é capaz de funcionar como biblioteca estática (.a, .lib) e biblioteca compartilhada (.so, .dll), estando na versão estável 2.3.0, desenvolvido pela HARU é distribuído gratuitamente sob a licença ZLIB / LIBPNG.

2.3.7 Poppler

Poppler é uma biblioteca de renderização de PDF baseada no visualizador de PDF Xpdf. Este pacote contém utilitários em linha de comando (baseados no Poppler) para obter informações de documentos PDF, convertê-los para outros formatos ou manipulá-los, muito semelhante às funções do Xpdf. Isso é útil para fornecer funcionalidades de manipulação de arquivos PDF como uma biblioteca compartilhada.

Poppler conta com vários *frontends* para interface gráfica, como cpp, glib, qt4 e qt5. Dentre suas principais ferramentas estão: pdfimages, para extração de imagens; pdfinfo, que retorna informações do arquivo e a pdftotext, para conversão para arquivo de texto. Conforme o *website* da companhia, atualmente a versão estável é Poppler-0.61.1 e estando licenciado sob a GPL.

2.4 BASE DE DADOS

“Os bancos de dados que não apresentam todas as características ACID são classificados como bancos não relacionais, como é o exemplo do modelo de banco de dados orientado a objetos e do NoSQL” (ANICETO; XAVIER, 2014). O paradigma de persistência de dados NoSQL é uma solução alternativa para os bancos de dados relacionais. Os bancos de dados NoSQL são muito flexíveis, possuem alta escalabilidade e disponibilidade, além de sua alta performance.

Hoje, existe uma grande quantidade de bancos de dados NoSQL no mercado, que trabalham de diferentes maneiras, como os bancos de dados orientados a documento, chave-valor, grafos e colunas. O MongoDB trabalha com documentos no formato JSON *(Java Script Object Notation),* para armazenar dados. Também fornece os recursos necessários para um ambiente de produção: balanceamento de carga, replicação, indexação, consulta e pode atuar como um sistema de arquivos.

Em nosso projeto optamos por trabalhar com um banco de dados orientado a documento. Em geral, os bancos de dados orientados a documento não possuem um esquema pré-definido, dessa forma, os documentos armazenados não precisam possuir estrutura padrão como em bancos de dados estruturados.

3 IMPLEMENTAÇÃO

A busca manual por expressões em qualquer base de dados e em qual seja a situação, por si só, traz um desgaste e um estresse desnecessário para quem o faz, além de contar o tempo e o trabalho que são “perdidos” com uma atividade como essa.

Justo uma problemática como essa foi a principal motivação para desenvolver esse projeto. O fato de desperdiçar tempo com um trabalho tão trivial, nos impulsionou a buscar uma solução que traga ganho a esse contexto.

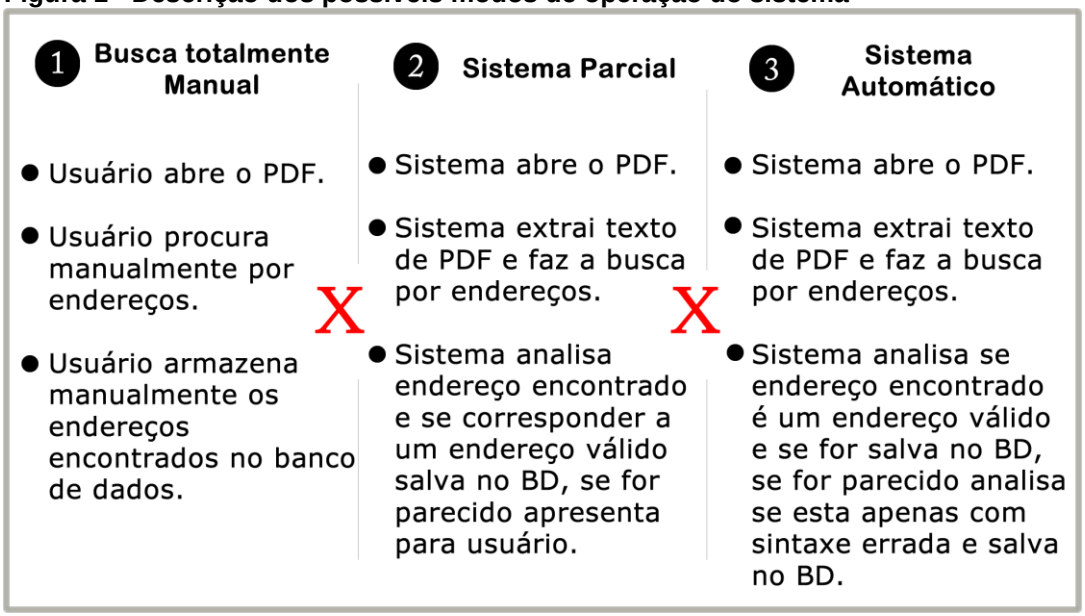
3.1 CONTEXTO DO PROBLEMA

O cenário onde se encontra o problema é: alguns projetos que estão sendo iniciados no PPGCC (Programa de Pós-Graduação em Ciência da Computação) da UTFPR, campus Ponta Grossa, precisam extrair algumas informações de uma série de arquivos no formato PDF. Esses PDF’s são colunas de classificados cedidos por um jornal local, de onde serão extraídas informações como endereços e preços.

O objetivo deste trabalho é desenvolver uma aplicação que atue como ferramenta auxiliar neste árduo processo de busca, com o propósito de fazer com que o usuário encontre a informação que está precisando rapidamente, de modo que não precise analisar os arquivos PDF por inteiro.

A figura 1 ilustra três situações: o contexto atual, o quadro que estamos propondo e uma perspectiva do que seria a condição perfeita para determinar a solução ideal do problema.

Figura 1 - Descrição dos possíveis modos de operação do sistema



Fonte: Autoria própria

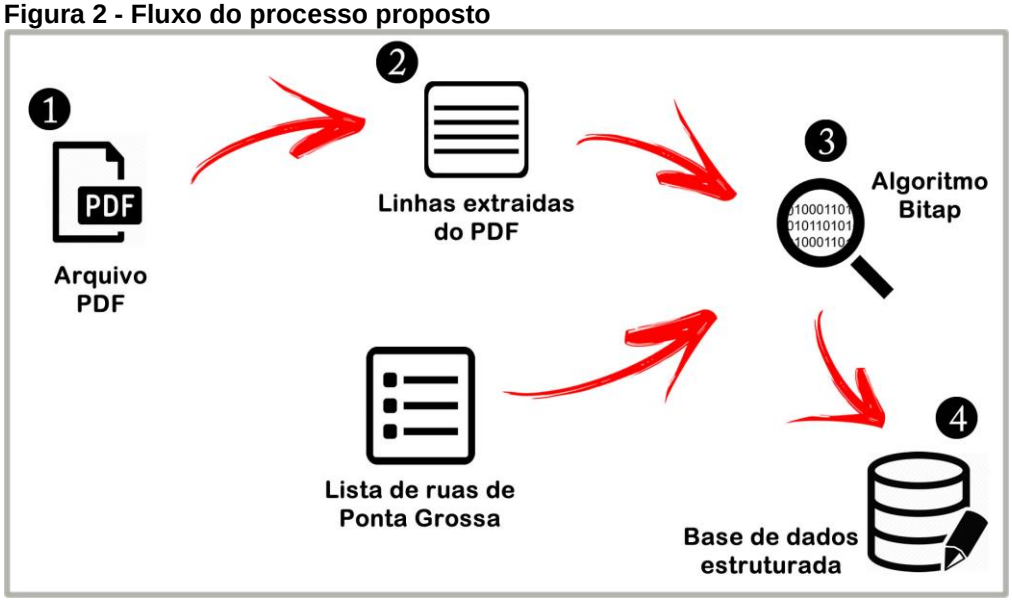
Na situação 1, o contexto é caracterizado por um processo realizado manualmente, sem nenhum tipo de automatização. Nessas circunstâncias, o usuário teria um trabalho desgastante e prolongado, pois precisa buscar de arquivo por arquivo a expressão desejada, copiando e colando palavra após palavra.

A situação 3 ilustra a circunstância totalmente automatizada, onde um *software* seria responsável por abrir uma serie de arquivos PDF, encontrar informações como nome de rua, preço e data, e então salvar em uma base de dados. Tudo isso sem a interferência do usuário e sem o risco de salvar uma palavra errada, deixar passar uma palavra escrita pela metade ou com alguma abreviação por exemplo.

Porem uma aplicação desse porte demandaria muito tempo, visto a sua complexidade, algo que apenas para um TCC não teríamos tempo hábil para o desenvolvimento. Talvez isso seja possível para um desenvolvimento posterior, por outros alunos, partindo do resultado que estamos propondo.

O quadro que estamos propondo, é representado pela etapa 2 da figura 1, e detalhado pela figura 2:

Figura 2 - Fluxo do processo proposto



Fonte: Autoria própria

A propositura explanada pela figura 2 trata-se de um *software* que faz a parte trabalhosa da tarefa, porém não atua absolutamente sozinho. No passo 1, ele carrega o arquivo PDF e através da ferramenta pdftotext realiza a extração do texto do arquivo PDF. No passo 2, a aplicação efetua a busca de endereços no texto extraído, identificando preço e data ligado ao endereço encontrado. No passo 3, por meio do algoritmo Bitap, ele faz a comparação aproximada com os nomes de ruas de Ponta Grossa, que estão gravados em uma lista de endereços.

Diferente de outros algoritmos como o de força bruta, Rabin Karp, entre outros que buscam por *strings* iguais, este algoritmo faz a comparação de duas *strings*, verificando se são parecidas. Isso diminui a chance de que uma palavra com escrita errada passe despercebida na busca.

Ao localizar uma expressão idêntica ou parecida com um dos endereços da lista, o resultado será apresentado na tela para que o usuário possa analisar se a expressão se trata de um endereço ou não, para que seja armazenada no banco de dados.

3.2 RESULTADOS ALCANÇADOS NA PRIMEIRA ETAPA

Depois da escolha do tema, começamos o trabalho de pesquisa, com o intuito de conhecer as ferramentas e tecnologias que teríamos à disposição e que melhor atenderiam no desenvolvimento desse trabalho. Como desde o início já havíamos definido, em comum acordo com o professor orientador, que a linguagem de programação usada no desenvolvimento da aplicação seria o C++. Apesar de não termos a pretensão, num primeiro momento, de automatizar todo o processo, sabemos que a aplicação que será desenvolvida a partir deste trabalho, agilizará muito o processo de busca em questão.

Através da implementação do algoritmo Bitap, faremos a comparação aproximada dos resultados obtidos na busca com uma lista de endereços previamente cadastrada. Por se tratar de um algoritmo de busca aproximada, que possibilita encontrar nomes de ruas com acentuação incorreta ou escrita com sintaxe errada, por exemplo, os resultados encontrados quando não tiverem sintaxe igual aos endereços da lista, serão exibidos em uma tela que possibilite a tomada de decisão por parte do usuário se a expressão encontrada corresponde a um endereço ou não. Se corresponder a um endereço o usuário ainda poderá fazer a alteração deste endereço para a sintaxe correta antes de autorizar que o mesmo seja armazenado na base de dados.

3.3 DESENVOLVIMENTO

Embasados em todo estudo feito durante o referencial teórico, em toda pesquisa realizada durante a fase de levantamento de requisitos e em todas tecnologias e ferramentas alçadas durante o processo de desenvolvimento, este capítulo descreve a fase de desenvolvimento do projeto.

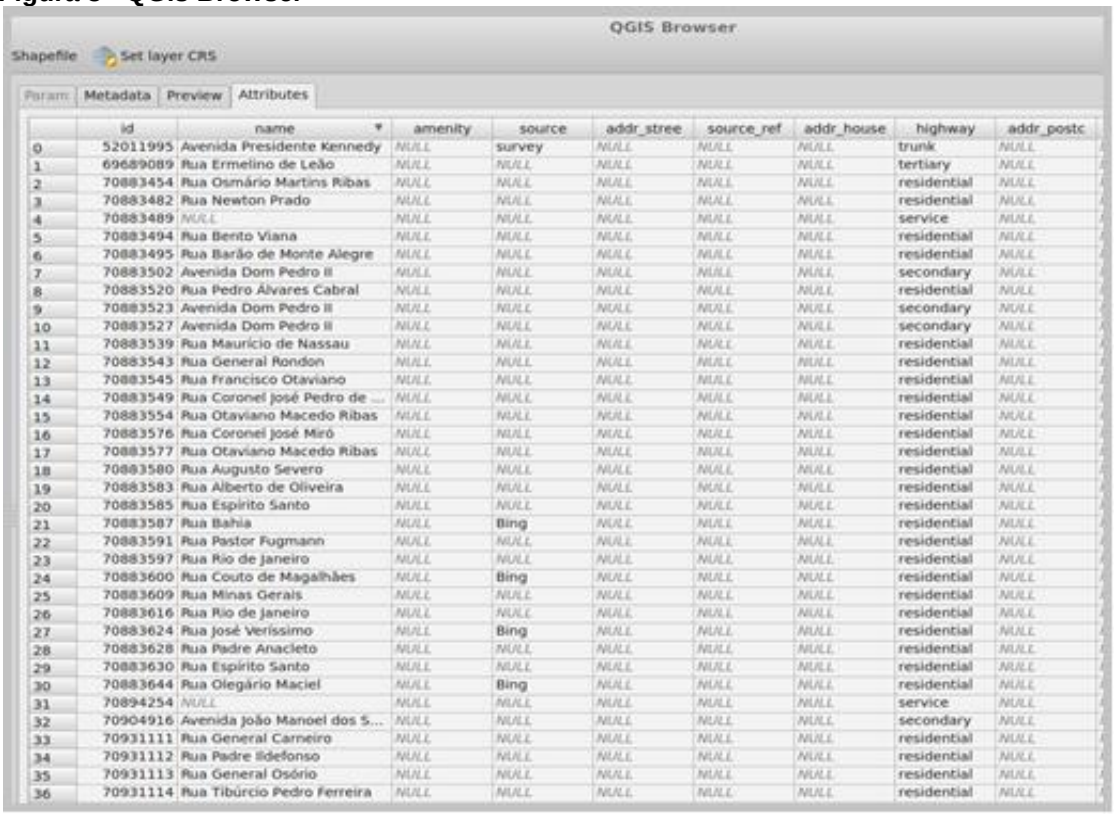
Para tal, foram utilizados scripts, recursos e bibliotecas da linguagem C++, a adição da biblioteca Poppler no projeto, o algoritmo Bitap e o paradigma de banco de dados NoSQL orientado a documentos, optando por documentos no formato JSON.

O desenvolvimento do projeto foi dividido em quatro etapas, que podem ser destacadas como: (1) conhecimento mais aprofundado das ferramentas e tecnologias e modelagem do sistema;(2) extração de texto dos arquivos PDF e busca por

endereços;(3) implementação do algoritmo Bitap e apresentação dos resultados ao usuário e por último (4) a organização das informações em arquivos JSON e exportação desses arquivos para uma base de dados NoSQL. Para identificarmos a aplicação que esta sendo desenvolvida por este projeto, batizamos seu nome de Prokurator.

A lista de ruas para uso no Prokurator foi obtida inicialmente da internet, do projeto Open Street Maps, com o auxílio da ferramenta Qgis que é ilustrado na figura 3.Estes endereços foram armazenados em arquivo de texto, que serve como base para a próxima etapa, que é a busca por endereços nos anúncios de imóveis.

Figura 3 - QGis Browser



	id	name	amenity	source	addr_stree	source_ref	addr_house	highway	addr_postc
0	52011995	Avenida Presidente Kennedy	NULL	survey	NULL	NULL	NULL	trunk	NULL
1	69689089	Rua Ermelino de Leão	NULL	NULL	NULL	NULL	NULL	tertiary	NULL
2	70883454	Rua Osmário Martins Ribas	NULL	NULL	NULL	NULL	NULL	residential	NULL
3	70883482	Rua Newton Prado	NULL	NULL	NULL	NULL	NULL	residential	NULL
4	70883489	NULL	NULL	NULL	NULL	NULL	NULL	service	NULL
5	70883494	Rua Bento Viana	NULL	NULL	NULL	NULL	NULL	residential	NULL
6	70883495	Rua Barão de Monte Alegre	NULL	NULL	NULL	NULL	NULL	residential	NULL
7	70883502	Avenida Dom Pedro II	NULL	NULL	NULL	NULL	NULL	secondary	NULL
8	70883520	Rua Pedro Álvares Cabral	NULL	NULL	NULL	NULL	NULL	residential	NULL
9	70883523	Avenida Dom Pedro II	NULL	NULL	NULL	NULL	NULL	secondary	NULL
10	70883527	Avenida Dom Pedro II	NULL	NULL	NULL	NULL	NULL	secondary	NULL
11	70883539	Rua Maurício de Nassau	NULL	NULL	NULL	NULL	NULL	residential	NULL
12	70883543	Rua General Rondon	NULL	NULL	NULL	NULL	NULL	residential	NULL
13	70883545	Rua Francisco Otaviano	NULL	NULL	NULL	NULL	NULL	residential	NULL
14	70883549	Rua Coronel José Pedro de ...	NULL	NULL	NULL	NULL	NULL	residential	NULL
15	70883554	Rua Otaviano Macedo Ribas	NULL	NULL	NULL	NULL	NULL	residential	NULL
16	70883576	Rua Coronel José Miró	NULL	NULL	NULL	NULL	NULL	residential	NULL
17	70883577	Rua Otaviano Macedo Ribas	NULL	NULL	NULL	NULL	NULL	residential	NULL
18	70883580	Rua Augusto Severo	NULL	NULL	NULL	NULL	NULL	residential	NULL
19	70883583	Rua Alberto de Oliveira	NULL	NULL	NULL	NULL	NULL	residential	NULL
20	70883585	Rua Espírito Santo	NULL	NULL	NULL	NULL	NULL	residential	NULL
21	70883587	Rua Bahia	NULL	Bing	NULL	NULL	NULL	residential	NULL
22	70883591	Rua Pastor Fugmann	NULL	NULL	NULL	NULL	NULL	residential	NULL
23	70883597	Rua Rio de Janeiro	NULL	NULL	NULL	NULL	NULL	residential	NULL
24	70883600	Rua Couto de Magalhães	NULL	Bing	NULL	NULL	NULL	residential	NULL
25	70883609	Rua Minas Gerais	NULL	NULL	NULL	NULL	NULL	residential	NULL
26	70883616	Rua Rio de Janeiro	NULL	NULL	NULL	NULL	NULL	residential	NULL
27	70883624	Rua José Veríssimo	NULL	Bing	NULL	NULL	NULL	residential	NULL
28	70883628	Rua Padre Anacleto	NULL	NULL	NULL	NULL	NULL	residential	NULL
29	70883630	Rua Espírito Santo	NULL	NULL	NULL	NULL	NULL	residential	NULL
30	70883644	Rua Olegário Maciel	NULL	Bing	NULL	NULL	NULL	residential	NULL
31	70894254	NULL	NULL	NULL	NULL	NULL	NULL	service	NULL
32	70904916	Avenida João Manoel dos S...	NULL	NULL	NULL	NULL	NULL	secondary	NULL
33	70931111	Rua General Carneiro	NULL	NULL	NULL	NULL	NULL	residential	NULL
34	70931112	Rua Padre Ildefonso	NULL	NULL	NULL	NULL	NULL	residential	NULL
35	70931113	Rua General Osório	NULL	NULL	NULL	NULL	NULL	residential	NULL
36	70931114	Rua Tibúrcio Pedro Ferreira	NULL	NULL	NULL	NULL	NULL	residential	NULL

Fonte: Autoria própria

Para dar maior velocidade de execução ao programa essa lista de endereços, que foi armazenada em um arquivo de texto, e carregada em um vetor cada vez que o Prokurator é iniciado.

Na segunda etapa iniciamos o desenvolvimento do sistema com a integração da aplicação com o Xpdf, cuja função é a conversão dos arquivos que contém os anúncios de imóveis no formato PDF para arquivos de texto como mostra a figura 4.

Figura 4 - Carregamento da biblioteca

```
char nomePDF[100];
printf("Digite o nome do arquivo de Anuncios onde será realizada a busca (sem colocar a extensão .pdf): ");
gets(nomePDF);
system("cls");
char inicioCaminho[200] = "C:\\Prokurator\\Sistema\\poppler-0.51\\bin\\pdftotext C:\\Prokurator\\EntradaPDF\\";
char fimCaminho[200] = ".pdf C:\\Prokurator\\SaidaTXT\\Imoveis3.txt";
strcpy(inicioCaminho,strcat(inicioCaminho,nomePDF));
system(strcat(inicioCaminho,fimCaminho));
```

Fonte: Autoria própria

Na terceira etapa continuamos com o desenvolvimento do sistema, iniciando sua segunda parte que engloba a implementação de algoritmo Bitap. Na figura 5 pode ser observada a chamada do algoritmo Bitap através da função buscaStringComBitap, após a abertura do arquivo de anúncios convertido para arquivo txt.

Figura 5 - Chamada do algoritmo Bitap

```
string texto_str;
int i = 0;
ifstream arq;
arq.open("../SaidaTXT/Imoveis.txt");
while (std::getline(arq, texto_str)){
    string op, op2;
    for (int indexVetor=0; indexVetor < vetLogradouros.size(); indexVetor++){
        int rua = buscaStringComBitap(texto_str, vetLogradouros[indexVetor], 1);
        if(rua != -1){
            cout << "Nesta linha: " << texto_str << "\nEstá contido o endereco: " << vetLogradouros[i
            cin >> op;
            if(op == "s"){
                anuncio.endereco = vetLogradouros[indexVetor];
                i++;
                break;
            }
        }
    }

    if(i == 1){
        for(int y=0; y<vetPossiveisPrecos.size(); y++){
        int index = buscaStringComBitap(texto_str, vetPossiveisPrecos[0], 1);
        if(index != -1){
            cout << "Esta linha corresponde ao valor do imóvel? " << texto_str << "\n";
            cin >> op2;
            if(op2 == "s"){
                anuncio.preco = texto_str;
                i++;
                break;
```

Fonte: Autoria própria

Após a abertura do arquivo de anúncios, é feita a leitura de cada linha do arquivo txt, dentro de um laço de repetição *while*. A cada iteração do laço *while* existe dois laços de repetição *for*.

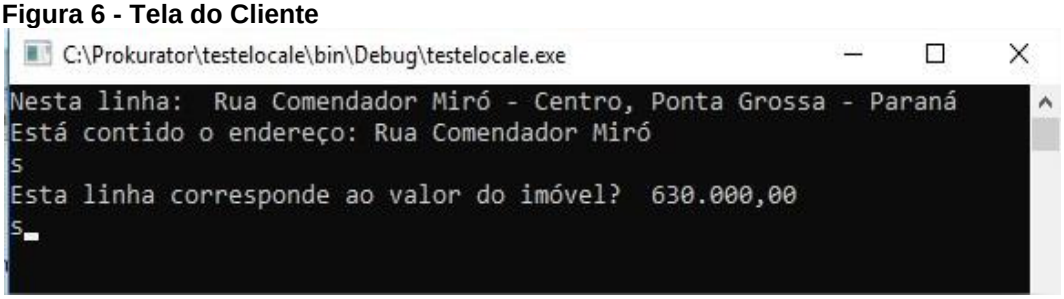
No primeiro laço *for*, a cada iteração é feita a chamada do algoritmo Bitap com o objetivo de comparar a atual linha do arquivo de anúncios com as posições da lista de endereços armazenada em um vetor, que foi carregado a partir do arquivo de texto no início da execução do Prokurator.

O segundo laço *for* é executado apenas após o Bitap localizar uma ocorrência de endereços similares, na linha atual no arquivo txt de anúncios com uma das posições do vetor de endereços. Após a execução do laço *for* é realizada a chamada do Bitap novamente, com o intuito de buscar pelo valor mais próximo ao endereço encontrado, tomando como base outro vetor já preenchido com possíveis formatos de valores, no início da execução do Prokurator, com o qual será comparado.

Ainda nesta etapa foi implementado a apresentação dos resultados ao usuário, para que este tome a decisão de salvar as informações, se realmente se tratar de um endereço e valor. Junto com o endereço e preço do imóvel, antes do armazenamento é anexado a data atual do sistema para se ter uma base da data em que foram analisados os anúncios.

Foi definido junto ao orientador que a interação com o usuário seria via console. Essa decisão deve-se ao fato de que a extração das informações dos arquivos PDF, a busca pelas expressões similares e o armazenamento das informações em uma base de dados de forma organizada, foi considerado prioridade em relação ao desenvolvimento da interface gráfica. A forma de interação com o usuário pode ser observada na figura 6.

Figura 6 - Tela do Cliente



Fonte: Autoria própria

A quarta etapa corresponde a parte final do desenvolvimento, contemplado o armazenamento dos resultados, de forma organizada, em arquivos JSON. Para isso adotamos o paradigma de banco de dados NoSQL orientado a documentos, optando por documentos no formato JSON, conforme a figura 7.

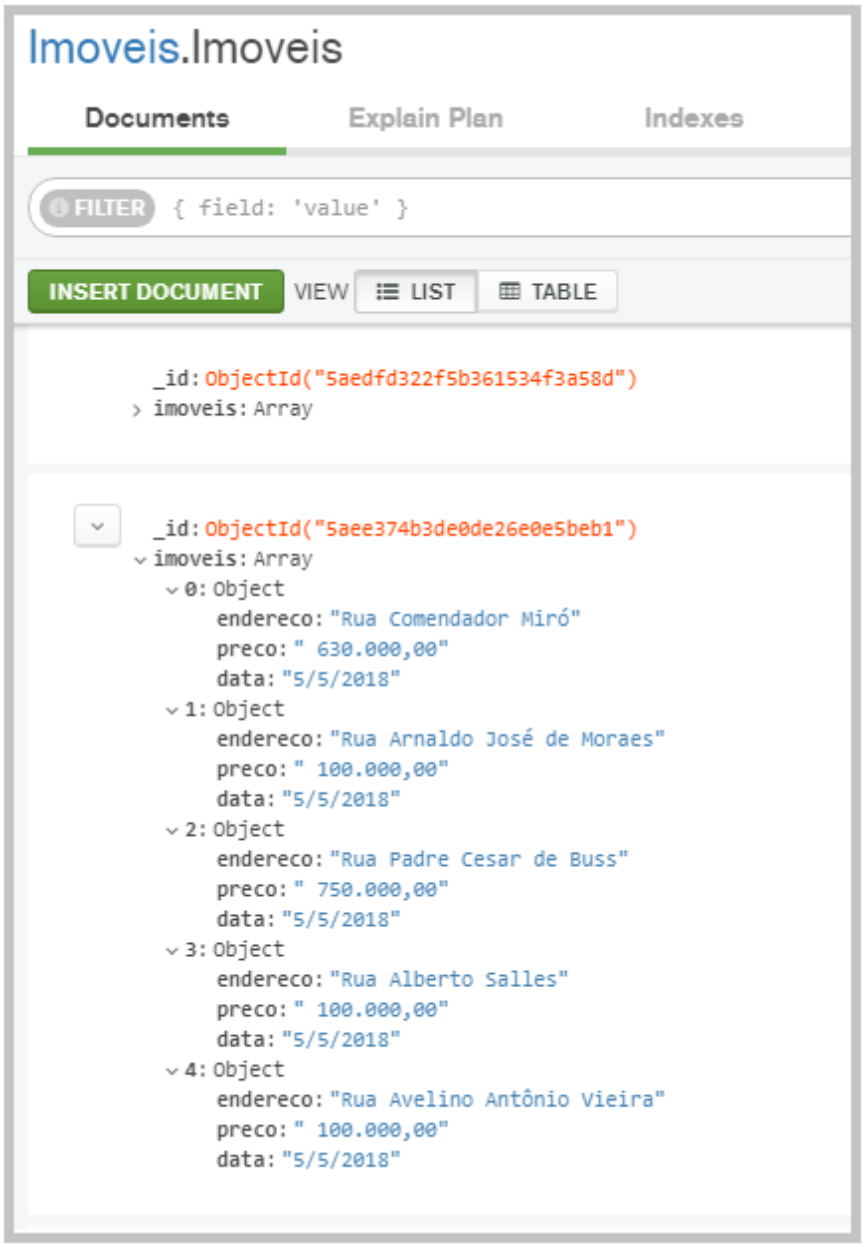
Figura 7- Dados no formato JSON


```
{
   "imoveis":[
      {
         "endereco":"Rua Comendador Miró",
         "preco":" 630.000,00",
         "data":"5/5/2018"
      },
      {
         "endereco":"Rua Arnaldo José de Moraes",
         "preco":" 100.000,00",
         "data":"5/5/2018"
      },
      {
         "endereco":"Rua Padre Cesar de Buss",
         "preco":" 750.000,00",
         "data":"5/5/2018"
      },
      {
         "endereco":"Rua Alberto Salles",
         "preco":" 100.000,00",
         "data":"5/5/2018"
      },
      {
         "endereco":"Rua Avelino Antônio Vieira",
         "preco":" 100.000,00",
         "data":"5/5/2018"
      }
   ]
}
```


Fonte: Autoria própria

A figura 8, mostra os resultados já armazenados no banco de dados NoSQL MongoDB.

Figura 8 – Relação criada no MongoDB



Fonte: Autoria própria

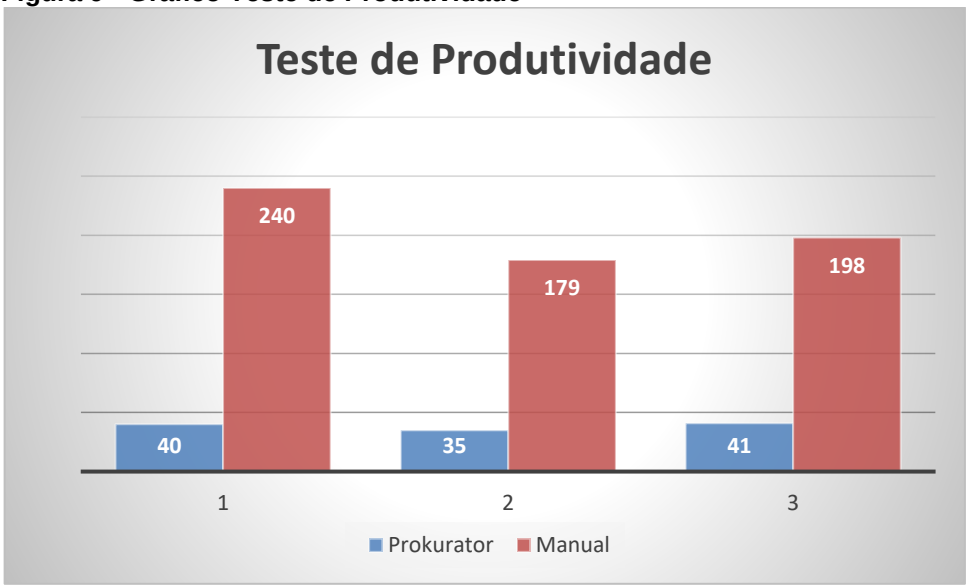
4 RESULTADOS ESPERIMENTAIS

Com o intuito de comprovar a produtividade da aplicação desenvolvida neste trabalho, foram realizados três testes experimentais de busca por endereços e preços em um arquivo PDF.

Tal arquivo, apenas para fins de teste, foi montado com anúncios extraídos da internet. Contendo três páginas, este arquivo possui 10 anúncios de imóveis à venda com endereço e preço.

Os resultados apresentados na figura 9estão em segundos, e são correspondentes a três testes manuais por busca e armazenamento dos resultados encontrados, e três testes utilizando o Prokurator.

Figura 9 - Gráfico Teste de Produtividade



Fonte: Autoria própria

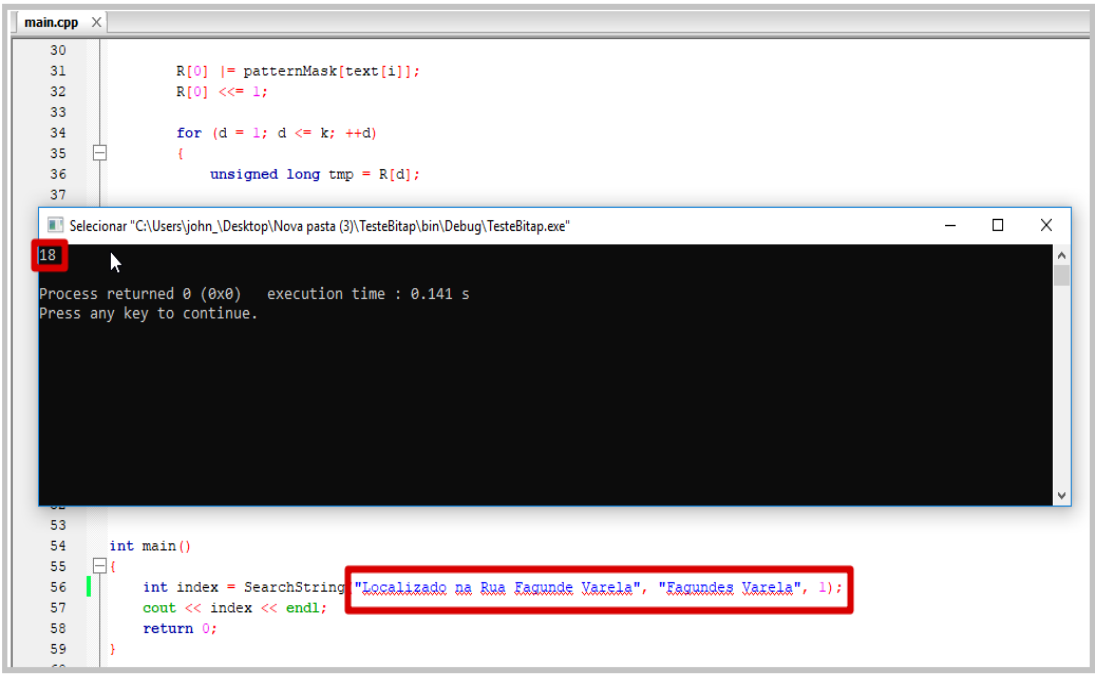
O teste demonstra uma produtividade 5,31 vezes maior utilizando o Prokurator em relação a busca manual. A média do Prokurator ficou em 38,66 segundos enquanto a busca manual teve uma média de 205,66 segundos.

Além da superioridade produtiva, o Prokurator armazena os resultados em arquivos no formato JSON para serem importados para o MongoDB, adicionando a data da busca ao objeto JSON. Enquanto, neste teste, os resultados da busca manual foram salvos em um simples arquivo de texto.

Também foram realizados alguns testes de efetividade do algoritmo Bitap na busca por *strings* parecidas. Apesar de sua eficácia, comprovada nos testes

realizados, podemos observar que existe um limite para sua assertividade, como podemos observar nas figuras 10, 11 e 12.

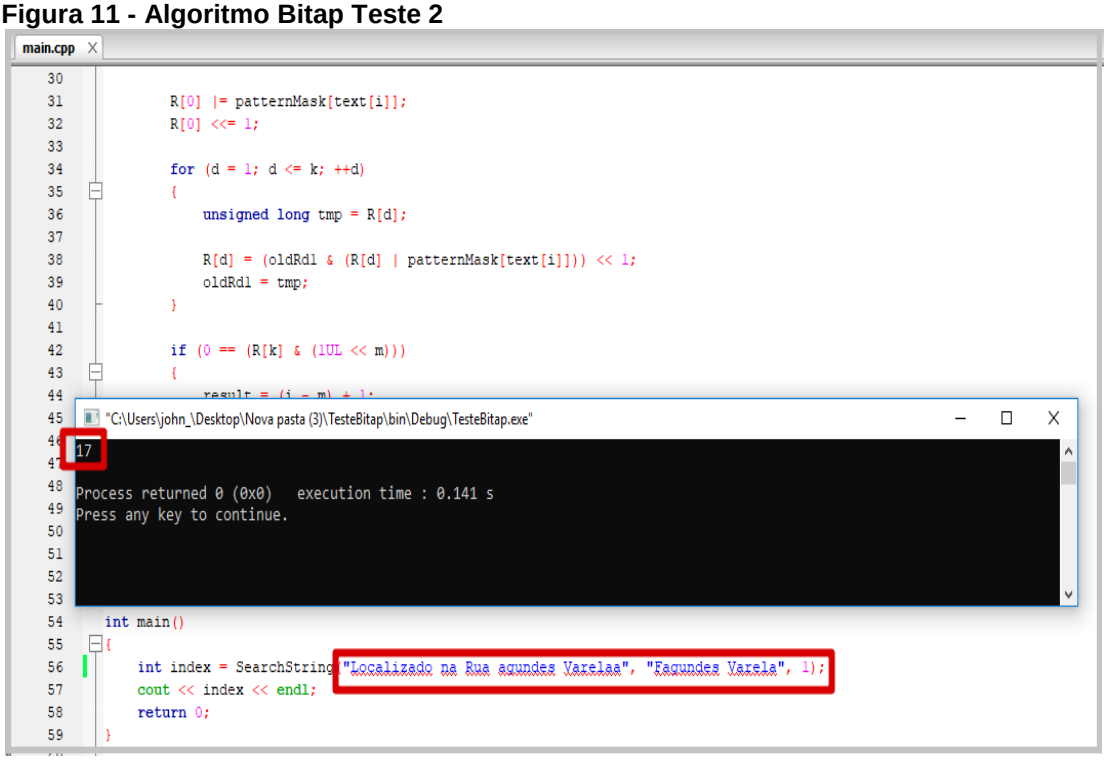
Figura 10 - Algoritmo Bitap Teste 1



Fonte: Autoria própria

A figura 10 ilustra o primeiro teste de efetividade utilizando o algoritmo Bitap. Neste teste a busca é feita em uma *string* com um caracter a menos em relação ao padrão a ser encontrado. O padrão desejado é “Fagundes Varela” e a *string* onde é feita a busca contém “Fagunde Varela”. O algoritmo Bitap retorna a posição onde encontrou a ocorrência parecida das *strings*.

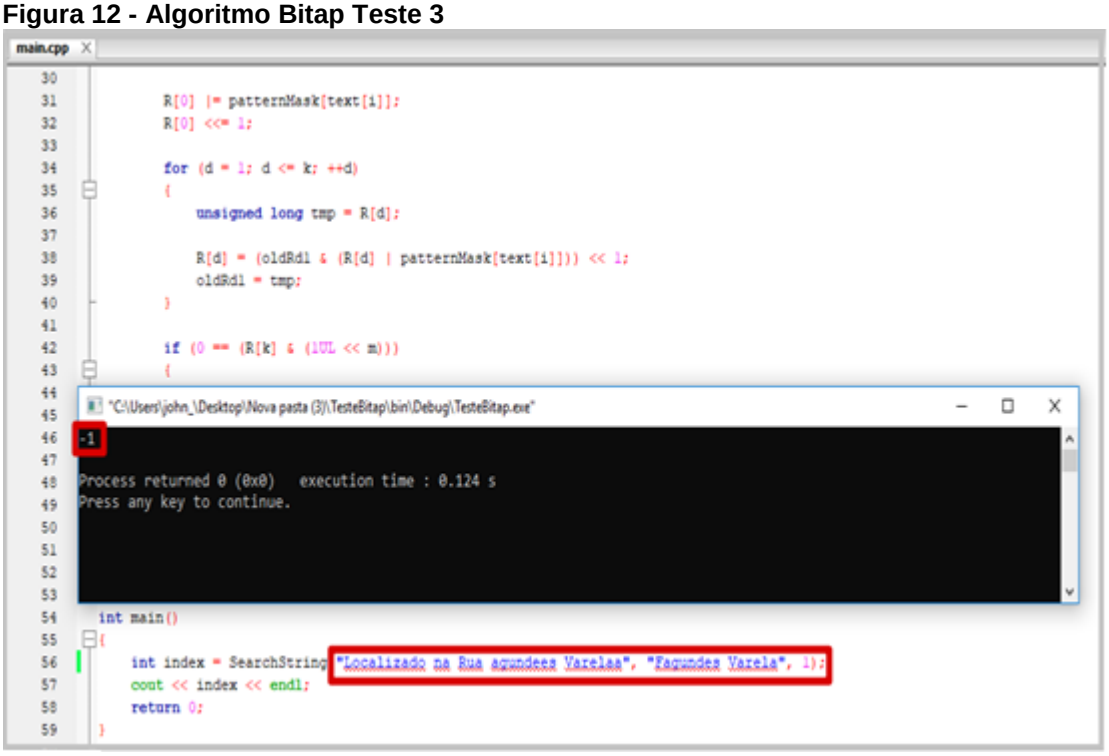
Figura 11 - Algoritmo Bitap Teste 2



Fonte: Autoria própria

Na figura 11 é possível observar o segundo teste realizado com o algoritmo Bitap. Neste teste existem dois erros na *string* onde é feita a busca, onde consta “agundes Varelaa”, o padrão desejado é “Fagundes Varela”. Da mesma forma o algoritmo Bitap retorna a posição da similaridade encontrada.

Figura 12 - Algoritmo Bitap Teste 3



Fonte: Autoria própria

A figura 12 demonstra o último teste realizado com o algoritmo Bitap. Desta vez buscando o padrão “Fagundes Varela” em uma *string* contendo “adundees Varella”. Neste teste a *string* correspondente, na linha pesquisada, contém três erros, o que não permitiu ao Bitap encontrar a similaridade entre as duas.

Ao concluir os testes pode-se dizer, que mesmo que o Bitap não tenha encontrado o endereço no terceiro teste, os resultados são bastante aceitáveis para o objetivo neste trabalho.

5 CONCLUSÃO

O desenvolvimento do presente trabalho nos permitiu, através de todo conhecimento adquirido durante o curso, juntamente com as descobertas provenientes das pesquisas contidas neste texto, ajudar a otimizar uma tarefa de um cenário real. Mais do que adquirir conhecimento, as pesquisas realizadas durante todo o projeto, nos oportunizou a sair de um desenvolvimento voltado apenas a sala de aula, para uma solução que traga maior produtividade para o meio a qual será inserida.

Conforme citado na introdução, o objetivo central deste trabalho, que é possibilitar que informações contidas em textos desestruturados possam ser utilizadas em aplicações que necessitam de tais informações, foi alcançado. O processo de se identificar e destacar a presença de endereços e preços em classificados de imóveis no formato PDF, após o desenvolvimento do Prokurator, teve sua produtividade aumentada em no mínimo 5 vezes, conforme teste de produtividade realizados. A gravação da informação, que antes teria que ser feita em arquivos de texto simples, com o uso do Prokurator, agora é feita automaticamente em arquivos JSON, que serão exportados para uma base de dados NoSQL.

A eficácia da busca dos endereços nos arquivos PDF, comparando com a lista de ruas contidas na aplicação, também é muito boa, conforme testes de eficácia realizado. Levando em consideração que o algoritmo Bitap encontra *strings* parecidas em um texto, pode ocorrer a indicação de duas ruas com nomes parecidos como sendo a mesma. O que não é problema, pois é o usuário quem define se a similaridade está correta ou não.

Com o uso do Prokurator, a base de dados não corre o risco de ter nomes de ruas com sintaxe errada em seus registros. Ao encontrar similaridades e após confirmação do usuário, o nome da rua que será registrado na base de dados será o que está contido na lista interna contida na aplicação.

É importante frisar que o Prokurator é uma ferramenta auxiliar de um processo cansativo e demorado. Ele não é uma aplicação totalmente automática, porém de grande assessoria. Além de proporcionar uma produtividade muito maior, do que se a atividade fosse realizada manualmente, reduz consideravelmente as possibilidades de erros e inconsistências.

Para finalizar, deixamos expresso nosso desejo de que a ferramenta desenvolvida seja aprimorada em futuros trabalhos. Como sugestão, ainda deixamos a possibilidade de trabalhar com mineração de dados após a fase de busca e armazenamento de dados.

REFERÊNCIAS

AMBROSIO, Ana Paula L. MORAIS, Edison Andrade Martins. **Mineração de Textos**. Relatório Técnico. Instituto de Informática - Universidade Federal de Goiás. Dezembro, 2007.

ANICETO, Rodrigo Cardoso; XAVIER, Renê Freire. **Um Estudo Sobre a Utilização do Banco de Dados NoSQL Cassandra em Dados Biológicos**. 2014. Monografia (Graduação) - Universidade de Brasília. Brasília, 2014.

ARANHA, Christian. PASSOS, Emmanuel. **A Tecnologia de Mineração de Textos**. RESI - Revista Eletrônica de Sistemas de Informação. Nº 2. 2006.

AZEVEDO, Breno Fabrício Terra. BEHAR, Patrícia Alejandra. REATEGUI, Eliseo Berni. **Aplicação da Mineração de Textos em Fóruns de Discussão**. Centro Interdisciplinar de Novas Tecnologias na Educação - Universidade Federal do Rio Grande do Sul. Vol. 8, Nº 3. dezembro, 2010.

BARION, Eliana Cristina Nogueira. LAGO, Décio. **Mineração de Textos**. Revista de Ciências Exatas e Tecnologia, Valinhos (SP), Vol. 3, Nº 3, p. 123-140. Dezembro, 2008.

BEZERRA, Vilker Silva. **Compressão digital**: um estudo comparativo entre codecs padrão MPGE-4 AVC/H.264. 2009. 67 p. Monografia – Faculdade Faria Brito Departamento de Ciência da Computação. Fortaleza – CE, 2009.

BUSNELLO, Felipe Octaviano Delgado. **Software Livre e os Direitos do Autor**. Artigo extraído do Trabalho de Conclusão de Curso - Faculdade de Direito - Pontifícia Universidade Católica do Rio Grande do Sul. 2008.

DA SILVA, Maria Estela Vieira. **XSimilarity**: Uma ferramenta para consultas por similaridade embutidas na linguagem XQuery. Trabalho de Conclusão de Curso - Graduação em Ciências da Computação - Universidade Federal do Rio Grande do Sul. Porto Alegre, 2007.

DEUTSCH, Laurence Peter. **DEFLATE Compressed Data Format Specification version 1.3**. Normas técnicas. Maio 1996. Disponível em: <https://www.ietf.org/rfc/rfc1951.txt>. Acesso em: 01 mai. 2018.

GLYPH & COG, LLC. **Company Information**. Disponível em: <https://www.glyphandcog.com/company.html>. Acesso em: abril 2018.

GONÇALVES, Rodrigo Moura. **Aplicação de uma ferramenta web open source de CRM**. 2007. 68 f. Trabalho de Conclusão de Curso – Sistemas de Informação, Universidade Federal de Santa Catarina. Florianópolis, 2007.

HUTH, Guilherme. **Um modelo para o gerenciamento de bancos de dados SQL através de Stored Procedures**. 2002. 94 f. Dissertação (Mestrado) – Ciência da Computação, Universidade Federal de Santa Catarina. Florianopolis, 2002.

KERRISK, Michael. **The Linux programming interface**: a Linux and UNIX system programming handbook. 1. ed. San Francisco, 2010.

MADEIRA, Renato de Oliveira Caldas**. Aplicação de técnicas de mineração de texto na detecção de discrepâncias em documentos fiscais**. 2015. 66 f. Dissertação (Mestrado) – Fundação Getúlio Vargas, Escola de Matemática Aplicada. Rio de Janeiro, 2015.

PAOLI, Antônio Roberto. **Um estudo avançado do problema da Maior Subsequência Comum**. 2016. 70 p. Monografia – Universidade Federal da Bahia – UFB a Departamento de Ciência da Computação Programa de Graduação. Salvador, 2016.

RABBANY, Reihaneh. **Information Extraction from Clinical Records**: Project Report for Information Extraction Meets Databases Course. Computing Science Department–Universityof Alberta. Fall, 2010.

VASEL, Rafael Luís. **Um Sistema de Extração e Publicação de Informações Georreferenciadas em um Domínio Turístico**. 2007. Trabalho de Conclusão de Curso Graduação em Ciências da Computação - Universidade Federal de Santa Catarina. Florianópolis, 2007.