



INSTITUTO FEDERAL DE BRASÍLIA

CAMPUS BRASÍLIA

CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET

Davi Pereira Speck Alves

**BUSCA DE PALAVRAS E EXPRESSÕES EM LIVROS E ARTIGOS COM
RANQUEAMENTO POR APARIÇÃO E RELEVÂNCIA**

Brasília

2022

Davi Pereira Speck Alves

**BUSCA DE PALAVRAS E EXPRESSÕES EM LIVROS E ARTIGOS COM
RANQUEAMENTO POR APARIÇÃO E RELEVÂNCIA**

Trabalho de conclusão de curso de graduação apresentado à Coordenação do Curso Superior de Tecnologia em Sistemas para Internet do Instituto Federal de Brasília – *Campus* Brasília, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Prof. Marx Gomes van der Linden

Brasília

2022

Ficha Catalográfica

Dados Internacionais de Catalogação na Publicação

Inserir aqui a ficha catalográfica gerada pela Coordenação de Biblioteca.

Davi Pereira Speck Alves

**BUSCA DE PALAVRAS E EXPRESSÕES EM LIVROS E ARTIGOS COM
RANQUEAMENTO POR APARIÇÃO E RELEVÂNCIA**

Trabalho de conclusão de curso de graduação apresentado à Coordenação do Curso Superior de Tecnologia em Sistemas para Internet do Instituto Federal de Brasília – *Campus Brasília*, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Aprovado em: _____ de _____ de _____.

BANCA EXAMINADORA

Marx Gomes van der Linden - IFB
(orientador)

Prof. Dr. Fábio Henrique Monteiro Oliveira - IFB

Daniel Sundfeld - IFB

Dedicatória

Dedico este trabalho àqueles que acreditaram desde o início na minha formação e caminhada.

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus, o grande protetor dos meus sonhos e conquistas, por mais um desfecho de um ciclo concretizado.

Também, em especial, aos meus pais, Nayara e Sérgio, e minha família que me deram o suporte para chegar aqui. E, inclusive, à minha namorada, Déborah, que sempre me motivou a superar os impasses que tentaram me derrubar durante a caminhada, hoje sinto-me pronto para as próximas conquistas que minha luta proverá.

Agradeço ao meu orientador Prof. Marx Gomes van der Linden, pela sabedoria com que me guiou nesta trajetória e a paciência com todos os impasses que existiram no meio do caminho.

À Coordenação do curso e ao Registro acadêmico, pela colaboração e disponibilidade no decorrer do curso.

Enfim, gostaria de deixar registrado também, o meu reconhecimento e gratidão àqueles que acreditaram em mim desde o início, pois acredito que sem o apoio deles seria muito difícil alcançar mais essa conquista na minha vida.

RESUMO

ALVES, Davi. **Busca de palavras e expressões em livros e artigos com ranqueamento por aparição e relevância**. 2022. Trabalho de Conclusão de Curso (Graduação) – Tecnólogo em Sistemas para Internet. Instituto Federal de Brasília – *Campus Brasília*. Brasília/DF, 2022.

Resumir e sumarizar são dois termos que estão se tornando cada vez mais comuns em todas as áreas de atuação, no qual resumir é de fato compactar um texto da melhor forma possível, enquanto sumarizar é elencar tópicos importantes advindos do mesmo. Assim, tornou-se ainda mais importante entender as técnicas que são capazes de eficientizar ambos. Diante tal realidade, este trabalho tem o intuito de apresentar o *software* RO (*Read Optimizer*) de forma teórica e prática, demonstrando o quanto é possível poupar tempo no processo de leitura ao resumir ou sumarizar, isso, claro, sem desvalorizar a qualidade do processo. Dessa forma, o documento em questão é pautado na especificação das etapas de desenvolvimento, indo desde o surgimento da ideia e *benchmarking* até a experiência do usuário na utilização do programa em uma primeira versão. É importante ressaltar também, que o *Read Optimizer* não produz o resumo de um texto propriamente dito, ele age mais como uma ferramenta de apoio e direcionamento. Com isso, verificou-se que apesar de existirem outras aplicações sustentadas nos mesmos preceitos, há, no RO, particularidades e vantagens quando se trata da valorização da margem de interpretação por parte do usuário. Por fim, considerando tal constatação, pode-se concluir que para aqueles que procuram por algo mais direto, que realiza uma síntese de forma imediata e totalmente computadorizada, o que já existe no mercado pode satisfazer, mas, entendendo a existência daqueles que necessitam de uma maior precisão no entendimento, há o sistema aqui desenvolvido.

Palavras-chave: Resumir. Sumarizar. Eficientizar. *Read Optimizer*. Síntese.

ABSTRACT

ALVES, Davi. **Search for words and phrases in books and articles ranked by appearance and relevance**. 2022. Course Completion Work (Undergraduate) - Technologist in Systems for Internet. Federal Institute of Brasília - Brasília Campus. Brasília / DF, 2022.

Summing up and summarizing are two terms which are becoming increasingly common in all areas of activity, in which summing up is in fact compressing a text in the best possible way, while summarizing is listing important topics arising from it. So, it has become even more important to understand the techniques which are able to make both efficient. In view of this reality, this paper intends to present the RO (Read Optimizer) software in a theoretical and practical way, demonstrating how much time can be saved in the reading process when summing up or summarizing, and, of course, without devaluing the quality of the process. In this way, the document in focus is based on the specification of the development stages, from the emergence of the idea and benchmarking to the user experience in using the program in a first version. It is also important to emphasize which Read Optimizer does not produce the resume of a text itself, but rather acts as a support and guidance tool. Thus, it was verified that although there are other applications supported by the same precepts, there are at RO, particularities and advantages when it comes to valuing the margin of interpretation by the user. Finally, considering this evidence, one can conclude that for those who are looking for something more direct, which performs a synthesis in an immediate and totally computerized way, what already exists on the market can satisfy, but understanding the existence of those who need greater precision in comprehension, there is the system developed here.

Keywords: Summing up. Summarizing. Efficient. Read Optimizer. Synthesis.

LISTA DE FIGURAS

Figura 1 – Como as pessoas gastam o tempo delas?	13
Figura 2 – Logo do <i>software</i> TurbineText	17
Figura 3 – Logo do sistema responsável pelo Summarizer	18
Figura 4 – Logo do <i>software</i> Resoomer	19
Figura 5 – Logo da linguagem de programação C	20
Figura 6 – Logo Python	21
Figura 7 – Logo HTML5	21
Figura 8 – Logo CSS3	22
Figura 9 – Logo JavaScript e Node.js	22
Figura 10 – Estrutura do sistema RO	24
Figura 11 – Buscando “AABAAA” em “CAABAABAAAA”	26
Figura 12 – <i>Software</i> RO (<i>Read Optimizer</i>)	28
Figura 13 – Imagem saída (ranking.txt)	31
Figura 14 – Imagem saída (aparicoes.txt)	31
Figura 15 – Página inicial	32
Figura 16 – Ranking de aparições	33
Figura 17 – Aparições na íntegra	33

LISTA DE SIGLAS

RO	Read Optimizer
.txt	Extensão de arquivo para arquivos de texto
BCPL	<i>Basic Combined Programming Language</i>
Algol 68	<i>ALGOritmic Language 1968</i>
C#	C Sharp
KMP	Knuth-Morris-Pratt
dfa[]	deterministic finite-state automaton
HTML5	<i>Hypertext Markup Language</i> (versão 5)
CSS3	<i>Cascading Style Sheet</i> (versão 3)
JS	JavaScript
RI	Recuperação da Informação
RAM	<i>Random Access Memory</i>
UTF-8	<i>UCS Transformation Format 8</i>
.pdf	Extensão de arquivo Portable Document Format

SUMÁRIO

1	INTRODUÇÃO	13
1.1	TEMA.....	15
1.2	PROBLEMA.....	15
1.2.1	Objetivo geral.....	16
1.2.2	Objetivos específicos.....	16
1.3	ESTRUTURA DO TCC.....	16
1.3.1	Classificação da Pesquisa.....	16
2	REVISÃO DE SISTEMAS SIMILARES	17
2.1	TURBINETEXT.....	17
2.2	SUMMARIZER.....	18
2.3	RESOOMER	18
3	TECNOLOGIAS DO SISTEMA	20
3.1	Linguagem de programação C.....	20
3.2	Python.....	21
3.3	HTML5.....	21
3.4	CSS3	22
3.5	JavaScript e Node.....	22
4	READ OPTIMIZER	24
4.1	APLICAÇÃO EXTERNA	24
4.1.1	Algoritmos implementados	25
4.1.1.1	Algoritmo KMP.....	25
4.1.1.2	Selection Sort	26
4.1.1.3	Tratamento de texto	26
4.2	BACK-END	27
4.3	FRONT-END	27
4.4	VANTAGENS DO READ OPTIMIZER.....	28
4.5	RI (Recuperação da Informação)	28
4.5.1	Margem de interpretação do usuário	29
4.6	CÓDIGO DO SOFTWARE IMPLEMENTADO	30
4.6.1	Como funciona o software?	30
4.7	EXECUÇÃO	30

4.7.1	Exemplo de execução.....	30
4.7.2	Exemplo de saída (ranking.txt)	31
4.7.3	Exemplo de saída (aparicoes.txt).....	31
4.8	PÁGINAS DO SISTEMA.....	32
5	CONCLUSÃO	34
6	Referências.....	36

1 INTRODUÇÃO

Como já dizia a poetisa e contista brasileira Cora Coralina (2001): “Estamos todos matriculados na escola da vida, onde o mestre é o tempo”. Mediante essa realidade, nasce a proposta do *software* RO (*Read Optimizer*), um programa capaz de realizar o ranqueamento da busca de palavras e expressões, possibilitando ao usuário enxergar o quão relevante pode ser uma aparição em uma página, por exemplo. Além disso, é necessário deixar claro que o *Read Optimizer* não atua somente como uma ferramenta para resumo, mas também sua execução gira em torno do processo de sumarização, que, de acordo com HUTCHINS (1987), divide-se em três tipos: indicativo, informativo e crítico. Para ele, sumários indicativos são mais “superficiais” contendo por obrigatoriedade, o essencial do texto, mas não necessariamente detalhando de forma muito específica. Por outro lado, o informativo é dado como partes do texto, contendo sempre todos os aspectos principais, e o crítico cabe aos tipos anteriores, uma vez que seria o ato de opinar inserido ao processo. Assim, o *Read Optimizer* atua unindo o essencial do indicativo à integridade das partes do informativo.

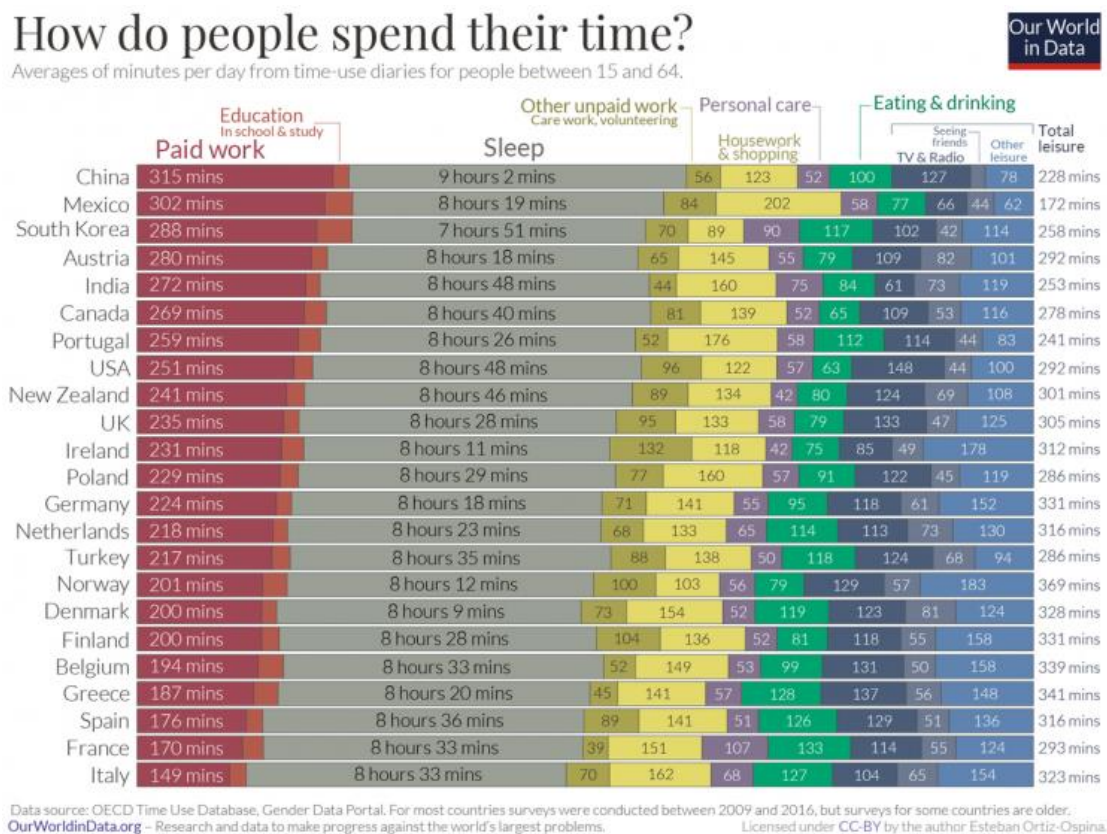


Figura 1: Como as pessoas gastam o tempo delas?

De acordo com os estudos da Our World in Data, as pessoas no âmbito global tendem a gastar mais de 50% do dia com responsabilidades e necessidades vitais (mais de 12 horas). Porém, é possível que o tempo restante ou até mesmo esse gasto, seja otimizado e, a melhor forma de alcançar tal, é integrando a tecnologia ao dia a dia. Por exemplo, se uma pessoa quer ler um livro para apenas obter conhecimento sobre um assunto específico e não o seu contexto como um todo, vale à pena ler o livro inteiro para obtê-lo? Será que não existe uma forma de tornar o alcance desse objetivo mais direcionado?

O *Read Optimizer* tem como objetivo unir princípios da sumarização ao ato de resumir, sem perder a margem de interpretação humana durante a execução do programa. Tal fato será realizado por meio da busca, direcionamento e ranqueamento de palavras e expressões, alcançando, portanto, a otimização do processo de leitura, agindo como mais uma ferramenta para valorizar o tempo. Além disso, vale destacar que todo esse processo ocorre pautado sobre as ideias da “Recuperação da Informação”. Nas palavras do mestre em Ciência da Computação Fábio Augusto Guimarães Teixeira (2010) “é o nome para o processo ou método segundo o qual um provável usuário da informação é capaz de converter a sua necessidade por informação em uma lista real de citações para documentos armazenados contendo informação útil para ele”. Princípio esse, que caminha ao lado do objetivo principal da aplicação: valorização da margem de interpretação do usuário.

Dessa forma, por mais que a *internet* já apresente diversas opções para auxílio no processo de sintetização, como as ferramentas: TurbineText, Summarizer e Resoomer, que serão especificadas ainda nesse trabalho, essas ferem a integridade da obra literária ou didática original ao resumir automaticamente. Isso acontece com a eliminação de períodos ou parágrafos, sem dar a chance ao usuário de definir a verdadeira relevância desses no texto.

Por outro lado, sustentando-se na valorização da margem de interpretação do usuário, há a Teoria da Subjetividade. De acordo com González Rey (2017), a subjetividade é: “a forma complexa como a psique humana se dá no desenvolvimento das pessoas e de todos os processos humanos”, ou seja, é ela que possibilita a realidade de cada pessoa ser única no seu modo de pensar, falar e agir. A fim de exemplificar a utilização do termo no contexto do projeto, considere a seguinte situação: se pedirmos a um grupo de pessoas para que realize o resumo de uma obra, é fato a impossibilidade de que todos apareçam com as mesmas ideias, destacando os mesmos pontos. Justamente por isso, o *Read Optimizer* entra como a resolução de um problema que a máquina por si só é incapaz de resolver: a não

valorização da interpretação e do pensamento do usuário.

Por sua vez, no *Read Optimizer*, a execução do sistema apresenta uma forma organizada de dispor o conteúdo buscado, mantendo fielmente a integridade de tudo que há no documento. É fato que um processo de aprendizagem bem feito não é medido pelo tempo em que acontece, mas sim pela qualidade. Por isso, a agilidade contribui com algo extremamente positivo ao auxiliar os usuários em tarefas como: traçar o perfil de uma personagem em uma obra literária ou definir a relevância de um conteúdo bem específico nos capítulos de um livro didático.

1.1 TEMA

A **busca de palavras e expressões em livros e artigos com ranqueamento por aparição e relevância** pode representar mais do que se pode imaginar. Nesse viés, pensando em meios de utilizações mais generalistas, com o tema é possível constatar a importância e o significado das aparições encontradas na obra determinada pelo usuário. Além da funcionalidade apresentada pelo *software* RO, a interpretação de quem utiliza é extremamente valorizada, pois a integridade existente no resultado da execução possibilita a utilização da subjetividade do usuário, já que serão apresentadas as aparições. Porém, o significado de cada uma partirá da análise do mesmo.

1.2 PROBLEMA

O tempo tem se tornado um problema destaque no cotidiano de toda a população, assim como é visível no estudo da Our World in Data apresentado na introdução desse documento, então tudo que vier como uma solução eficiente e ágil, é promissor, afinal o dia nunca vai ter mais de 24 horas, certo? O *Read Optimizer* age em conjunto com um princípio de alcançar agilidade em objetivos que exigem análise e estudo de muitas palavras, como uma obra literária, um artigo ou um livro didático, por exemplo. Em suma, a ideia é possibilitar a sintetização de um conteúdo grande, metaforicamente falando, “digerir conhecimento” será algo mais fácil com o *software* aqui apresentado.

1.2.1 Objetivo geral

Otimizar o processo de “busca de palavras e expressões em livros e artigos com ranqueamento por aparição e relevância”, entendendo o quanto o tempo tem caráter definitivo no caminhar da vida de qualquer ser humano.

1.2.2 Objetivos específicos

- Realizar a busca de palavras e expressões;
- Ranquear as aparições por relevância quantitativa;
- Possibilitar ao usuário direcionamento para sintetização;
- Traçar possíveis conclusões por meio da quantificação de aparições por página;
- Facilitar a leitura e a aprendizagem.

1.3 ESTRUTURA DO TCC

Este trabalho está estruturado na seguinte ordem: apresentação de sistemas similares, contextualização do tema, tecnologias envolvidas no desenvolvimento dele e estruturação do *Read Optimizer*, além de exemplos práticos em torno da execução.

1.3.1 Classificação da Pesquisa

Apesar do *Read Optimizer* surgir em meio a um universo já conhecido, a pesquisa se concentra como algo exploratório, pois o intuito principal é tornar mais eficiente algo já existente, utilizando uma metodologia não presente em *softwares* similares. Existem diversas etapas envolvidas no desenvolvimento de um sistema, por isso, pesquisas bibliográficas, documentais e experimentais fazem parte do planejamento, além dos diversos estudos associados. Esses, alcançam, por fim, um perfil qualitativo à pesquisa, em busca de um programa realmente usual, ou seja, que cumpra com a proposta de eficientizar o tempo de quem o utiliza, sendo ainda, capaz de oferecer qualidade profissional aos usuários.

2 REVISÃO DE SISTEMAS SIMILARES

O posicionamento de sistemas similares ao que será desenvolvido no trabalho em questão vem como uma forma de demonstrar a diferença do mesmo em relação aos que já estão disponíveis no mercado, destacando sempre o problema a ser resolvido, a melhoria inerente ou ambos, surgidos em meio ao novo sistema.

2.1 TURBINETEXT

[TurbineText](#) é um *software* que busca reduzir o tamanho de um texto. A ideia é realizar um resumo do texto inserido pelo usuário, seja por digitação ou *upload* de um arquivo “.txt”. Para isso, o sistema realiza uma análise minuciosa, procurando manter a coesão e coerência. Além disso, é possível escolher dentre vários idiomas e até mesmo a porcentagem ou quantidade de linhas em que se quer reduzir o texto original. Fora o resumo, há também a identificação de sinônimos e busca de plágio a partir do texto inserido.



Figura 2: Logo do *software* TurbineText

Entretanto, um dos pontos não necessariamente negativos, já que não é o objetivo da ferramenta, é a eliminação da margem de interpretação por parte do usuário durante a utilização do [TurbineText](#), pois todo o processo de resumo é feito automaticamente, o que acarreta perda de integridade na saída do texto e faz com que o usuário não possa traçar um caminho do que realmente deseja e pretende com aquela sintetização.

2.2 SUMMARIZER

O [Summarizer](#) tem como objetivo a redução do tamanho de um texto. A diferença entre este e o apresentado anteriormente está na implementação do que existe por trás do sistema, o que torna difícil identificar o que seja “melhor” dentre eles, já que um pode atuar melhor a depender da situação, isto é, pode ser que para textos maiores, o [Summarizer](#) tenha uma execução mais adequada, assim como o [TurbineText](#) também.



Figura 3: Logo do sistema responsável pelo Summarizer

Além da funcionalidade de sumarizar dividida em duas formas: por palavras chaves ou parágrafos, existem outras, como checador gramatical ou gerador de citações. Tem-se ainda, o fato de não acometer a perda total da margem de interpretação por parte do usuário, que coloca o [Summarizer](#) como um sistema mais ideal para aquele que busca manter “originalidade” no texto. Contudo, mesmo que exista uma forma de preservar a margem de interpretação do usuário, dependendo da perspectiva, ainda sim, é montado um resumo de forma inteiramente computacional, que fere intensamente a permanência do conteúdo original após o fim do processo.

2.3 RESOOMER

Uma outra opção, inclusive mais conhecida que as anteriores e também mais direcionada somente à produção de resumos, é o [Resoomer](#). Um diferencial desse em relação aos outros, é a manipulação da saída depois da sintetização ter sido feita e o fato de ser um sistema disponibilizado gratuitamente para uso, o que claramente aumenta sua popularidade. Quanto a esse *software*, pode-se dizer que ele age como uma otimização em relação ao ato de resumir, pois a flexibilidade no manejo do texto existe inclusive depois do fim da execução do programa. Desse modo, apesar das possibilidades de alteração

existentes, ainda são mudanças computacionais envolvidas, ou seja, determinadas por algoritmos e uma inteligência artificial, pois não há escrita e interação do usuário no processo, há somente a definição de filtros e opções previamente programadas.



Figura 4: Logo do *software* Resoomer

3 TECNOLOGIAS DO SISTEMA

Neste tópico, será explicado quanto à arquitetura do sistema aqui desenvolvido, ou seja, quais tecnologias foram utilizadas, suas particularidades e consequentemente o porquê de suas implementações no *Read Optimizer*.

3.1 Linguagem de programação C



Figura 5: Logo da linguagem de programação C

A linguagem C foi criada pelo cientista da computação Dennis Richie, sendo ela, derivada de outras duas: a BCPL (*Basic Combined Programming Language*) e a Algol 68 (*ALGOritmic Language 1968*). Inicialmente, foi pensada com o propósito exclusivo de atuar no desenvolvimento de uma nova versão do sistema operacional Unix, mas, hoje atua nos mais variados projetos. A sintaxe (“regras” para a construção de um programa) da linguagem C não foge muito dos conceitos de outras linguagens, isto é, relacionam-se com tipos (definem as propriedades dos dados), funções (indicam as ações que serão executadas pelo programa) e declarações (trechos do programa que alocam na memória funções, variáveis e afins). Ademais, é válido destacar que a linguagem C é compilada estaticamente e seus códigos transformados diretamente em código nativo (linguagem de máquina).

Segundo o *site* Trybe, a linguagem C “ainda é uma das mais populares do mercado devido às diversas vantagens que apresenta. Uma das grandes vantagens dessa linguagem é a capacidade de gerar códigos rápidos, ou seja, com um tempo de execução baixo. Além disso, a programação em C é bastante simplificada, pois sua estrutura é simples e flexível. Tendo isso em mente, podemos dizer que as principais características da linguagem C são:

- portabilidade;
- geração de código eficiente;
- regularidade.

Por fim, a popularidade da linguagem C é e foi grande, já que ela ainda influenciou

diretamente a estrutura e sintaxe de outras, como C++, Objective C e C#.”

3.2 Python



Figura 6: Logo Python

Python é uma linguagem de alto nível – ou *High Level Language* - é interpretada e fracamente tipada, ou seja, não precisamos declarar o tipo de uma variável, por exemplo. Sabe-se também, que um dos motivos para tamanho destaque no século XXI está em sua dinamicidade que permite a utilização em diversas plataformas contemporâneas. Python é uma linguagem orientada a objetos, sendo assim, há uma aproximação do mundo real e o mundo virtual, por meio da criação e interação entre objetos, atributos, códigos, métodos, classes e outros.

Assim, de acordo com o *site* Devmedia: “Publicado em 1991, o Python traz características que possibilitam escrever o mesmo requisito em menos linhas de código que o necessário em outras linguagens de programação e hoje, além de adotado na construção de soluções *web*, também está sendo muito utilizado em aplicações que lidam com processamento de texto, *machine learning* e recomendação de conteúdo, áreas que não param de crescer”. Por isso, é importante destacar o quanto tem sido utilizada em conjunto com a biblioteca Pandas para a área de *Data Science* e, de certa forma, na produção do RO, Python foi utilizado com princípios de *Data Science* ao converter um arquivo de PDF para TXT.

3.3 HTML5



Figura 7: Logo HTML5

HTML (*Hypertext Markup Language*) é uma Linguagem de Marcação de Hipertexto em sua tradução, sendo ela a principal do desenvolvimento *web*. Dentre suas atribuições no contexto de desenvolvimento, a linguagem funciona como o desenho de como aparecerá o conteúdo estático da página, isto é, textos, imagens, vídeos, aqueles que não apresentam dinamicidade e interatividade por si só. Segundo o *site* Hostinger: “essa linguagem funciona de modo estático, o que significa que você não pode criar uma página na *internet* dinâmica ou interativa usando o HTML. Ela apenas modifica os elementos estáticos de uma página na *web*, como cabeçalho de conteúdo, rodapé, posição de imagens, etc”.

3.4 CSS3



Figura 8: Logo CSS3

CSS (*Cascading Style Sheet*) é usado para estilizar elementos escritos na *web* com a linguagem de marcação HTML, por exemplo. Sendo assim, o CSS em sua 3ª versão (CSS3) separa o conteúdo da sua representação visual no *site*, entregando dinamicidade e interatividade. Além disso, de acordo com o *site* LinhaDeCodigo, o CSS3 aprimorou ainda mais o que aconteceu com a chegada do CSS2: “algumas novidades interessantes foram incorporadas, o que melhorou e muito a qualidade dos *websites* desenvolvidos, mas assim como o que aconteceu com o CSS1, após um tempo tornou-se necessário que tivesse algo a mais para ser acrescentado...”.

3.5 JavaScript e NodeJS



Figura 9: Logo JavaScript e NodeJS

JavaScript é uma linguagem de programação que permite a implementação de itens que exigem um maior grau de complexidade às páginas *web*, de certa forma, entrega ainda mais interatividade e dinamicidade como o CSS, mas agora, tendo comunicação com o servidor. Além disso, junto ao JavaScript para a programação do back-end no ambiente *web*, foi integrado o Node que amplia a utilização da linguagem, podendo inclusive ser utilizada fora do *browser*. Uma das principais motivações para a utilização do NodeJS seria o fato de sua execução ser *single-thread*, que de acordo com o *site* Opus-Software, isso na prática significa: “em um servidor *web* utilizando linguagens tradicionais, para cada requisição recebida é criada uma nova *thread* para tratá-la. A cada requisição, serão demandados recursos computacionais (memória RAM, por exemplo) para a criação dessa nova *thread*. Uma vez que esses recursos são limitados, as *threads* não serão criadas infinitamente, e quando esse limite for atingido, as novas requisições terão que esperar a liberação desses recursos alocados para serem tratadas. No modelo Node.js, apenas uma *thread* (*Event Loop*) é responsável por tratar as requisições”.

4 READ OPTIMIZER

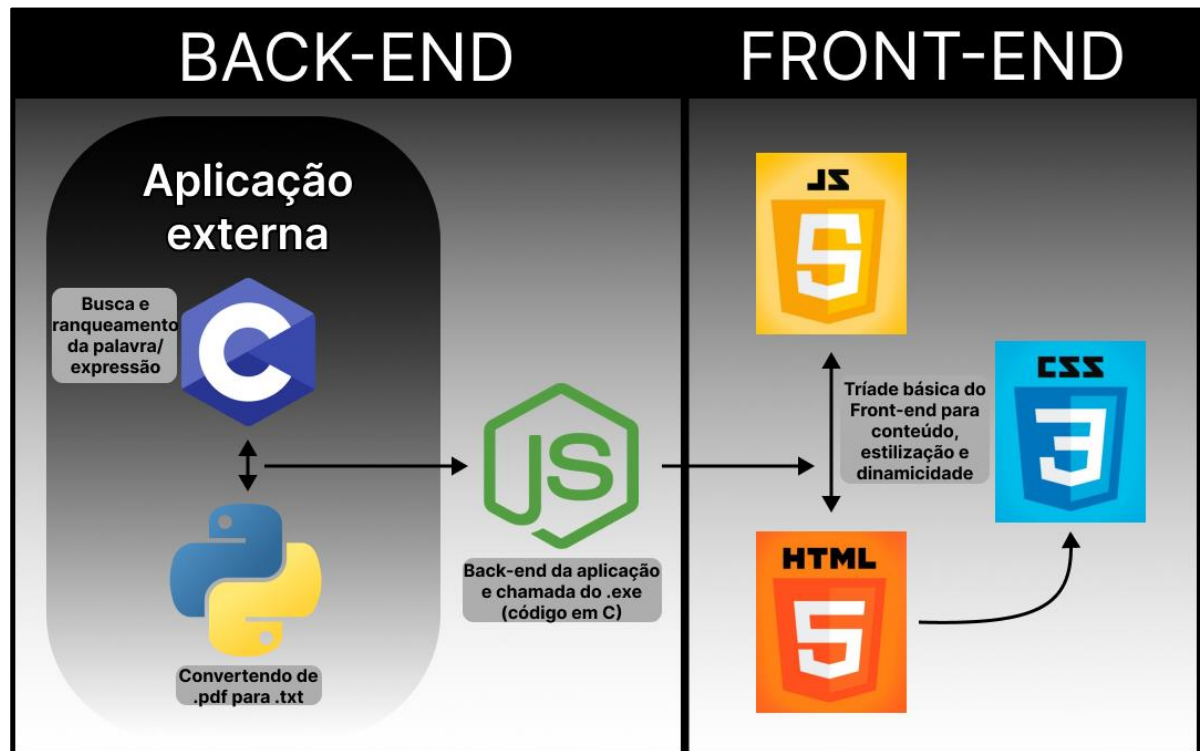


Figura 10: Estrutura do sistema RO

O algoritmo por trás da execução do sistema é uma aplicação externa desenvolvida em C, majoritariamente, com um pequeno detalhe em Python, que seria simplesmente a conversão de um documento PDF para TXT, identificando também as páginas do mesmo. Além disso, para a comunicação com o Front-end, foi utilizado NodeJS com Express.js, tornando possível a funcionalidade da busca ao chamar o executável da aplicação externa.

4.1 APLICAÇÃO EXTERNA

A aplicação externa, já como tratada anteriormente, foi desenvolvida em C, principalmente pela maleabilidade da linguagem. Com isso, é possível alcançar uma comunicação mais precisa com o *hardware* de qualquer máquina, determinando, por exemplo, exatamente o quanto deve ser alocado de memória para uma execução em específico.

C é uma linguagem de programação compilada de propósito geral, pautada sob 3 paradigmas: programação estruturada, imperativa e procedural. De forma sucinta, trata-se de um desenvolvimento sequencial em torno de decisões e iterações, que altera

constantemente o valor das variáveis envolvidas, a fim de alcançar um passo a passo bem definido diante o objetivo da aplicação.

Dentro do *Read Optimizer*, uma das principais motivações para utilizar a linguagem C, foi a possibilidade de gerar um executável por meio da compilação do programa. A partir desse, foi possível determinar a comunicação do back-end com a aplicação externa, simplesmente diante a execução do sistema por meio de um executável.

Por fim, é importante ressaltar que uma parte da aplicação externa foi concretizada a partir da biblioteca PyPDF, implementada por um *script* em Python. Basicamente, esse *script* é chamado logo após a identificação do texto no formato PDF que fará parte da execução do programa. Tecnicamente falando, antes da determinação da palavra que será buscada e ranqueada, foi necessário converter o texto para o formato .txt, sem perder a numeração de páginas. Portanto, possibilitando maior compatibilidade e, também, oportunidade de manipulação generalizada do conteúdo, tanto na aplicação externa, quanto no back-end da aplicação *web*.

4.1.1 Algoritmos implementados

Além de implementações próprias, existem algoritmos já conhecidos que foram tanto utilizados na íntegra, quanto readaptados. Assim como também existe uma função em específico que está pautada sob a estrutura de algoritmo, já que se trata de algo que pode ser implementado em qualquer outro sistema.

4.1.1.1 Algoritmo KMP

O algoritmo Knuth Morris e Pratt, aqui desenvolvido na linguagem C, é responsável pela busca de *substrings* em *strings*, que em termos usuais, é a procura de palavras ou expressões em um texto, justamente o ponto chave do *Read Optimizer*. Mas afinal, tecnicamente falando, como funciona o algoritmo?

O algoritmo KMP examina os caracteres de TXT um a um, da esquerda para a direita, sem nunca retroceder. Em cada iteração, o algoritmo sabe qual posição k de pat deve ser emparelhada com a próxima posição $i+1$ de um arquivo no formato .txt. Em outras palavras, no fim de cada iteração, o algoritmo sabe qual índice k deve fazer o papel de j na próxima iteração.

Para fazer isso, o algoritmo KMP usa uma tabela `dfa[][]` que armazena os índices mágicos `k` (o nome da tabela deriva da expressão *deterministic finite-state automaton*). As colunas da tabela são indexadas pelos índices `0..M-1` do padrão e as linhas são indexadas pelo alfabeto, que é o conjunto de todos os caracteres do texto e do padrão.

	C	A	A	B	A	A	B	A	A	A	A
uma tentativa:	A	A	B	A	A	A					
não precisa tentar:	A	A	B	A	A	A					
não precisa tentar:	A	A	B	A	A	A					
próxima tentativa:	A	A	B	A	A	A					

Figura 11: Buscando “AABAAA” em “CAABAABAAAA”

4.1.1.2 Selection Sort

Como algoritmo de ordenação foi escolhido o Selection Sort. Esse, no sistema, determina, inicialmente, a quantidade de aparições encontradas no texto inteiro (a partir do KMP) e a coloca como ponto de partida para ordenação. Ou seja, se uma execução obtiver 350 como a quantidade de aparições da palavra buscada em um texto inteiro, a ordenação acontecerá buscando página a página e, se a quantidade de aparições na mesma for equivalente ao número procurado ($350-1\dots$), diminui um em um, até chegar a um. Então, tem-se uma posição do *ranking* identificada.

E por que utilizar Selection Sort ao invés de Bubble Sort ou QuickSort? Primeiro porque ele faz sentido conceitualmente ao ser implementado junto ao KMP, pois as características dele complementam o fato do algoritmo KMP não retroceder sobre a leitura. E, tratando de forma mais específica quanto às características, o Selection Sort ao contrário do usual, no *Read Optimizer*, atua realizando uma ordenação decrescente. Além disso, trata-se de um algoritmo de ordenação local que não exige armazenamento temporário, algo que faz mais sentido ao entender o fato de ser utilizado para ordenar um vetor que contém a quantidade de aparições página a página.

4.1.1.3 Tratamento de texto

Na aplicação externa foi desenvolvida, também, uma função responsável por tirar as acentuações de todas as vogais que existem no texto inserido, objetivando ampliar o grau

de compatibilidade do sistema. Para essa retirada, é feita a busca do início ao fim do texto por caracteres que se enquadram nessa necessidade, sendo realizada por meio da identificação dos números que representam esses caracteres acentuados dentro da tabela ASCII. Por fim, a partir do momento em que é identificado esse valor, o mesmo é substituído pelo número que representa a vogal não acentuada.

4.2 BACK-END

Para o back-end, foi escolhido o Node.js, justamente pela sua conceituação e estrutura, já que se trata de um *software* de código aberto executado por um mecanismo JavaScript que possibilita a execução do JavaScript fora do navegador *web*. Além disso, há também o fato do Node funcionar de acordo com o princípio *single-thread*, determinado de forma específica dentro da seção “Tecnologias Utilizadas”.

Na organização dos métodos de requisição HTTP (GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, PATCH), foi utilizado o *framework* Express.js, que em linhas gerais, possibilita o desenvolvimento de aplicações JavaScript com o Node.js.

Algo muito importante a se destacar, foi a estratégia de manipulação das variáveis comuns à aplicação externa. A primeira pergunta realizada foi: de que forma conseguir manipular variáveis tanto em C, quanto em Node.js? E aproveitando a conversão de PDF para TXT providenciada por um *script* em Python, a resposta veio de forma clara: por meio da manipulação de arquivos de texto.

Com essa estratégia, tornou-se possível utilizar a seguinte metodologia de execução: o executável gerado a partir da compilação do programa em C identifica o nome do arquivo PDF e a palavra a ser buscada, ambos inseridos pelo próprio usuário na interface *web*, em um arquivo de texto. Dessa forma, qualquer exigência que o executável tenha no sentido de variáveis, pode ser obtida por meio de um arquivo TXT manipulado pelo Node.js, a partir das inserções do usuário na interface *web*.

4.3 FRONT-END

Para o front-end existem duas opções: ou implementar a tríade HTML, CSS e JavaScript de forma pura ou utilizar um *framework* como ReactJS, AngularJS ou VueJS,

por exemplo. No caso do *Read Optimizer*, principalmente por ser um MVP (*Minimum viable product*), o objetivo foi manter a simplicidade utilizando as linguagens de forma pura, pois a concretização da interface *web* foi exatamente o que permitiu a análise do desempenho da aplicação externa.

Desse modo, não faz sentido ter uma visualização robusta sem antes confirmar toda a qualidade que a aplicação externa deve oferecer. Portanto, essa seria uma primeira versão, considerando que esse é o período de validação e melhoria no que se refere ao processo de execução da aplicação externa e sua comunicação com o front-end por meio do back-end.

4.4 VANTAGENS DO READ OPTIMIZER

Tendo como base os sistemas similares supracitados, o aspecto negativo de acometer perda de integridade no texto gerado pela execução do sistema, é geral. Logo, é justamente nisso que o *Read Optimizer* atua com sua proposta de resolução, ao apenas fracionar partes de um texto baseado na busca desejada. Assim, há preservação na originalidade do texto, o que possibilita amplitude na margem de interpretação do usuário, já que o sistema atuará oferecendo uma sumarização particionada em frases no resultado de sua execução.



Figura 12: *Software RO (Read Optimizer)*

4.5 RI (Recuperação da Informação)

O presente sistema foi construído em cima de preceitos formalizados de recuperação da informação, a qual é embasada numa área computacional que trabalha com o armazenamento de documentos e a recuperação automática de informações desses. Dentre as várias disponibilidades de informações, o objetivo do sistema de recuperação da informação é recuperar e localizar o que realmente é relevante para o usuário. Geralmente essa informação é conhecida como necessidade de informação do usuário. No caso do *Read Optimizer*, a margem de interpretação do usuário é preservada e, por isso, definir o que é

realmente relevante para si faz parte do propósito.

Assim, para obter o documento de interesse, é necessário que se traduza uma necessidade de informação em consulta. Essa, define-se como o conjunto de palavras-chave para recuperação de documentos, justamente a funcionalidade principal do RO, com a busca de palavras e expressões e o ranqueamento, procurando facilitar a percepção de relevância por meio da quantidade de aparições.

Outrossim, considerando variáveis de resposta para uma consulta, é possível determinar os trechos que contêm as palavras em um *ranking*, o que simplifica o processo e amplia o propósito do sistema para além do que se pensou em seu desenvolvimento. Por fim, entende-se em pleno século XXI o poder da informação desde o seu estado inicial, o dado bruto, e, com isso, constata-se a partir de uma busca organizada e direcionada, o quão é forte o potencial do *Read Optimizer* para com seus usuários.

4.5.1 Margem de interpretação do usuário

Foi tratado diversas vezes quanto ao processo de valorização da margem de interpretação do usuário na execução do *Read Optimizer*, assim como esse mesmo ponto foi posicionado dentre os objetivos principais do sistema. Mas o que de fato vem a ser essa valorização?

Em uma primeira análise, o *Read Optimizer* também não permite manipulação durante a execução, certo? Detalhe esse, justamente colocado como explicação do porquê os outros sistemas afetam negativamente essa margem de interpretação. Sendo assim, de que forma o *software* soluciona essa problemática, se não dando poder ao usuário no momento do processamento?

Basicamente, em linhas gerais, o *Read Optimizer* não resume um texto propriamente dito, nem sequer retira alguma palavra, na verdade. A execução gira em torno do processo de sumarização, é uma forma de direcionar o usuário a partir justamente da palavra que ele deseja. Ou seja, se o usuário deseja buscar uma personagem X, o sistema trará um *ranking* de aparições e, junto a esse, estará todas as aparições na íntegra, facilitando o encontro justamente do que se procura.

4.6 CÓDIGO DO SOFTWARE IMPLEMENTADO

O código implementado para a funcionalidade principal do *Read Optimizer* está localizado em um repositório público no Github:

https://github.com/DaviSpeck/TCC_IFB.

4.6.1 Como funciona o *software*?

1. O código lê o documento enviado pelo usuário em PDF e o transcreve para TXT, baseado na codificação UTF-8;
2. É identificada a palavra ou expressão inserida para ser buscada;
3. São buscadas página a página, as respectivas aparições de acordo com a palavra inserida no passo 2;
4. Depois de caminhado todo o documento atrás das aparições, é feito o ranqueamento de páginas, considerando a quantidade de aparições por página;
5. Após determinado um *ranking*, é disponibilizado a possibilidade de navegação pelas aparições de uma página X escolhida pelo usuário, contendo aproximadamente 75 caracteres antes e depois da aparição, considerando espaços em branco;

Depois desses passos, é possível que o usuário defina, por exemplo, o caminho trilhado por uma personagem dentro do enredo de uma obra literária e, por conseguinte, por meio dos períodos definidos a partir das aparições da palavra buscada, são visíveis também os locais onde esteve essa personagem, com quem interagiu e outras constatações.

4.7 EXECUÇÃO

Entrada: texto.pdf

Processamento: inserção da palavra a ser buscada

Saída: ranking.txt e aparicoes.txt

4.7.1 Exemplo de execução

Entrada: texto.pdf (Dom Casmurro – Machado de Assis)

Processamento: Capitu (palavra buscada)

Saída: ranking.txt e aparicoes.txt

4.7.2 Exemplo de saída (ranking.txt)

Aqui estão, em ordem decrescente, as páginas em que mais tiveram aparições da palavra buscada (Capitu). Isso permitirá ao usuário identificar se realmente essas páginas representam maior relevância por ter maior quantidade de aparições por interpretação própria sem desconsiderar a subjetividade envolvida no processo.

```
1º Lugar - Página 43 - 9 aparições
2º Lugar - Página 12 - 8 aparições
3º Lugar - Página 30 - 8 aparições
4º Lugar - Página 38 - 8 aparições
5º Lugar - Página 65 - 8 aparições
```

Figura 13: Imagem saída (ranking.txt)

4.7.3 Exemplo de saída (aparicoes.txt)

Como um auxílio à saída anterior, há aqui os trechos associados a cada aparição em sua respectiva página, auxiliando ainda mais o processo de definição para a relevância, funcionando como mais uma ferramenta para agilizar o processo de leitura e sintetização por parte do usuário.

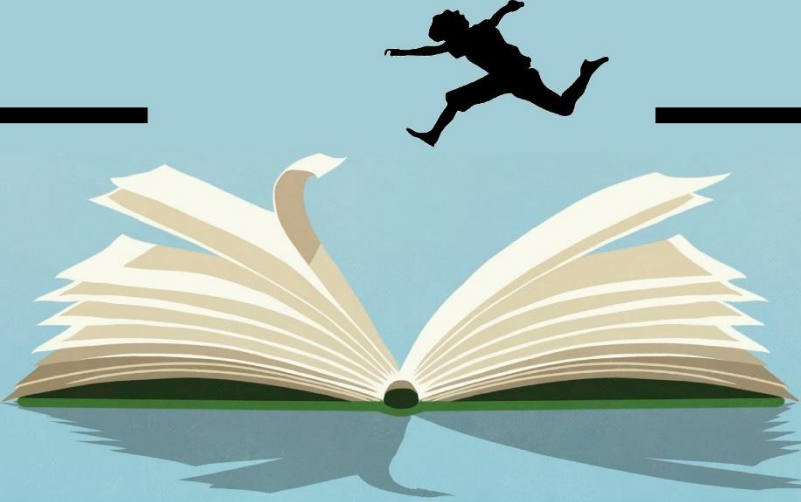
```
Aparicao da pagina 3 - . Basta a idade; Bentinho mal tem quinze anos. Capitu fez quatorze a semar
Aparicao da pagina 10 - a coisa de brincadeira. Arranjavamos um altar, Capitu e eu. Ela servia de
Aparicao da pagina 11 - ra da mesma opiniao. Com que entao eu amava Capitu, e Capitu a mim? Re
Aparicao da pagina 11 - a opiniao. Com que entao eu amava Capitu, e Capitu a mim? Realmente, e
Aparicao da pagina 11 - eria das nossas conversações era a de sempre. Capitu chamava-me as vezes
```

Figura 14: Imagem saída (aparicoes.txt)

4.8 PÁGINAS DO SISTEMA

RO

- Artigos científicos
- Obras literárias
- Material didático



Read Optimizer

Dê um salto na sua leitura!!!

O Read Optimizer é um software capaz de auxiliar seus usuários em uma leitura, entregando uma maior capacidade de sintetização junto à uma interpretação mais clara. Com isso, tem-se a função de busca por palavra, que particiona as aparições ranqueando por página e, também, coloca a possibilidade de saber os 30 caracteres que aparecem antes e depois de uma aparição específica.

Como usar?

1. Escolha o arquivo
2. Escreva a palavra
3. Busque

Fácil né? Tá esperando o quê para experimentar?

Escolher arquivo

Nenhum arquivo escolhido

Palavra a ser buscada

Buscar

Desenvolvido por Davi Speck - 2021

Figura 15: Página inicial

Ranking de Páginas

Ranking	Página	Aparições
1º Lugar	Página 43	9 aparições
2º Lugar	Página 12	8 aparições
3º Lugar	Página 30	8 aparições
4º Lugar	Página 38	8 aparições
5º Lugar	Página 65	8 aparições
6º Lugar	Página 14	7 aparições

Figura 16: Ranking de aparições

Aparições

Nº	Aparições
1	ao me atrevia. Nao me parece bonito que o nosso Bentinho ande metido nos cantos com a filha do Tartaruga, e esta e a dificuldade, porq ue se eles pegvv
2	E ummodo defalar. Em segredinhos, sempre juntos. Bentinho quase naosai de la. A pe quena e uma desmiolada; o pai faz que nao ve; tomara ele que as coisvv
3	ca vi nada que faça desconfiar. Basta a idade; Bentinho mal tem quinze ano s . Capitu fez quatorze a semana passada; sao doiscriançolas. Nao se esqu
4	eia de o fazer padre, tem-se ganho o principal. Bentinho ha de satisfazer os desejos de sua mae. E depois a igreja brasileira tem altos destinos. Nao ww

Figura 17: Aparições na íntegra

5 CONCLUSÃO

A fim de atingir uma compreensão da entrega de agilidade e eficiência no processo de “busca de palavras e expressões em livros e artigos com ranqueamento por aparição e relevância”, definiu-se a necessidade de estudo dos diversos processos de aprendizagem e sintetização no exercício de leitura.

Em primeiro lugar, é importante ressaltar que os estudos em programação da aplicação foram iniciados com a linguagem C e toda a exposição a risco que a linguagem entrega, no fato de permitir uma configuração mais dinâmica para a alocação de memória, faz-se por "obrigatoriedade" estudar e entender, de fato, o que é digitado em cada linha de código.

Além disso, com o desenvolvimento do projeto em questão, possibilitou-se o entendimento de diversas ferramentas que antes, nem sequer, motivação e objetivo existia para que se fossem estudadas. Como exemplo, pode-se tratar do seguinte questionamento durante o desenvolvimento: “Como compilar e executar um código em C no sistema *web* prototipado?”. A resposta inicial era aquilo que mais acontece em meio à tecnologia: “Sabe-se que é possível, como quase tudo, mas como? E o que se enxerga para o futuro do *Read Optimizer*?”. Contudo, apesar de saber que existem diversas perguntas que surgem ao responder essas, o que mais se faz presente na resposta é o fato de saber o futuro brilhante que existe por trás da ferramenta, claro que com ajustes, contexto e melhorias, os quais são alguns dos diversos passos que existem no caminho até chegar nesse futuro.

Por sua vez, uma das ideias já pretendida é a possibilidade de armazenar resultados já realizados em um banco de dados, possibilitando que a execução seja imediata para um próximo usuário que buscar pela mesma palavra no mesmo livro e, claro, deve existir um exemplo real e validado da utilização da ferramenta. Assim, já que foi tão tratado dos exemplos de utilização em livros literários, por que não solicitar essa validação a um escritor que possa, de fato, utilizar bastante do *Read Optimizer*? Enfim, as ideias para o futuro do *software* só crescem conforme a criatividade e o conhecimento em tecnologia, o que significa que prospecções nunca faltarão.

Tratando de implementações futuras, é amplamente considerada a atribuição de uma inteligência artificial ao sistema, que em pensamentos iniciais, faria a identificação de uma personagem por meio de um pronome, por exemplo. Outrossim, é claro que a identificação do trecho em que a aparição possui relevância também é um fator que deve ser levado em

consideração pela inteligência artificial, entendendo que caracteres após e depois não é o suficiente para determinar o contexto da aparição.

Portanto, a partir da análise minuciosa discurrida durante o trabalho, conclui-se o quão é importante eficientizar qualquer que seja o processo do dia a dia e, por isso, o *Read Optimizer* mostra-se como uma solução completamente cabível e realista diante o proposto.

6 REFERÊNCIAS

Departamento de Ciência da Computação, Instituto de Matemática e Estatística.

AlgoritmoKMP para busca de substring. USP, 2018. Disponível em:

<https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/kmp.html>. Acesso em: 05/08/2021

BRUNET, João. **Ordenação por Comparação: Selection Sort.** Github, 2019.

Disponível em: <https://joaoarthurbm.github.io/eda/posts/selection-sort/>. Acesso em: 10/08/2021

NOLETO, Cairo. **Linguagem c: o que é e quais os principais fundamentos!** Trybe,

2020. Disponível em: <https://blog.betrybe.com/linguagem-de-programacao/linguagem-c/>. Acesso em: 20/01/2022

“devmedia”. **Guia Completo de Python.** Devmedia, 2019. Disponível em:

<https://www.devmedia.com.br/guia/python/37024#mais-sobre>. Acesso em: 20/09/2021.

HERTEL, Rafael. **O Que é HTML5 e Quais Vantagens ele Traz para seu Site.**

Hostinger, 2021. Disponível em: <https://www.hostinger.com.br/tutoriais/diferenca-entre-html-e-html5>. Acesso em: 20/01/2022

GONÇALVES, Ariane. **O que é CSS? Guia Básico para Iniciantes.** Hostinger,

2021. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-css-guia-basico-de-css>. Acesso em: 20/01/2022

MDN Web Docs. **O que é JavaScript?** Moz://a, 2018. Disponível em:

https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/What_is_JavaScript. Acesso em: 20/01/2022

lenon. **Node.js – O que é, como funciona e quais as vantagens.** Opus Software,

2018. Disponível em: <https://www.opus-software.com.br/node-js/>. Acesso em: 20/01/2022

MARTINS, Camilla; PARDO, Thiago; ESPINA, Alice; RINO, Lucia.

INTRODUÇÃO À SUMARIZAÇÃO AUTOMÁTICA. USP, 2018. Disponível em:

<https://sites.icmc.usp.br/taspardo/RTDC00201-CMartinsEtAl.pdf>. Acesso em: 12/02/2022

HUTCHINS, John (1987). *Summarization: Some problems and Methods.* Jones Meaning:

The frontier of informatics. Cambridge. London, pp. 151-173

SOUZA, Elias; TORRES, José. **A Teoria da Subjetividade e seus conceitos centrais.**

UFU, 2019. Disponível em: <https://seer.ufu.br/index.php/Obutchenie/article/view/50574>. Acesso em: 14/02/2022

“Psicanálise Clínica”. **O que é subjetividade? Conceito e exemplos.** Psicanálise

Clínica, 2019. Disponível em: <https://www.psicanaliseclinica.com/o-que-e-subjetividade/>. Acesso em: 20/02/2022

BASTOS, Henrique. **Diferenças entre uma linguagem compilada e uma linguagem interpretada.** Oficina da Net, 2010. Disponível em:

https://www.oficinadanet.com.br/artigo/programacao/diferencas_entre_linguagem_compilada_e_linguagem_interpretada. Acesso em: 15/03/2022

SHILDT, Herbert (1995). **C Completo e Total – Terceira Edição**. São Paulo: PearsonMakron Books, 2004.

ROVEDA, Ugo. **O que é Python, para que serve e por que aprender?** Kenzie, 2013. Disponível em: <https://kenzie.com.br/blog/o-que-e-python/>. Acesso em: 25/03/2022

GUIMARÃES, Leandro. **Qual a diferença entre dado e informação? Entenda agora!** Know Solutions, 2020. Disponível em: <https://www.knowsolution.com.br/diferenca-dado-e-informacao/>. Acesso em: 25/06/2022

“Oracle”. **O que é Ciência de Dados?** Oracle Brasil, 2020. Disponível em: <https://www.oracle.com/br/data-science/what-is-data-science/>. Acesso em: 27/06/2022

“Wikipedia”. **Recuperação de Informação**. Wikipedia, 2021. Disponível em: https://pt.m.wikipedia.org/wiki/Recuperação_de_informação. Acesso em: 28/06/2022

LOPES, Ilza (2002). **Uso das linguagens controlada e natural em base de dados: revisão da literatura**. v.31, n. 1, p. 41-52. Brasília: [s.n.]

Samuel. **Sistema Recuperação da Informação – Área medica**. Blogspot, 2010. Disponível em: <https://sistemaderecuperacaodainformacao.blogspot.com/2010/09/sistema-recuperacao-da-informacao-area.html>. Acesso em: 01/07/2022

“turbinetext”. **Gerador de Texto Automático**. TurbineText, 2015. Disponível em: <https://www.turbinetext.com>. Acesso em: 18/08/2022

“quillbot”. **Summarizer**. QuillBot, 2022. Disponível em: https://quillbot.com/summarize?utm_medium=paid_search&utm_source=google&utm_campaign=sitelinks&campaign_type=extension. Acesso em: 18/08/2022

“resoomer”. **Resoomer**. Resoomer, 2022. Disponível em: <https://resoomer.com/en/>. Acesso em: 19/08/2022

“idealista.pt”. **Como é que as pessoas passam o tempo?** Idealista, 2021. Disponível em: <https://www.idealista.pt/news/financas/economia/2021/02/05/46179-como-e-que-as-pessoas-gastam-o-seu-tempo-tempo>. Acesso em: 21/08/2022

“Wikipedia”. **C (linguagem de programação)**. Wikipedia, 2013. Disponível em: [https://pt.wikipedia.org/wiki/C_\(linguagem_de_programação\)](https://pt.wikipedia.org/wiki/C_(linguagem_de_programação)). Acesso em: 22/08/2022

“Wikipedia”. **Programação estruturada**. Wikipedia, 2020. Disponível em: https://pt.wikipedia.org/wiki/Programação_estruturada. Acesso em: 22/08/2022

“Wikipedia”. **Programação imperativa**. Wikipedia, 2015. Disponível em: [https://pt.wikipedia.org/wiki/C_\(linguagem_de_programação\)](https://pt.wikipedia.org/wiki/C_(linguagem_de_programação)). Acesso em: 22/08/2022

“Wikipedia”. **Programação procedural**. Wikipedia, 2011. Disponível em: https://pt.wikipedia.org/wiki/Programação_procedural. Acesso em: 22/08/2022

“treinaweb”. **O que é Express.js**. Ana Paula de Andrade, 2021. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-o-express-js>. Acesso em: 24/08/2022.

“geeksforgeeks”. **Selection Sort Algorithm**. Geeks for Geeks, 2021. Disponível em: <https://www.geeksforgeeks.org/selection-sort/>. Acesso em: 25/08/2022.

“treinaweb”. **Conheça os principais algoritmos de ordenação**. Daniel Viana,

2017. <https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao>. Acesso em: 25/08/2022