Utilização da Distância de Levenshtein na Aproximação de Palavras em Integrações de Sistemas

Allan Fernando O. de Andrade, Adrieli Kethin Santos,
Frank Willian C. de Oliveira Marcelo F. Terenciani

[1]Instituto Federal do Paraná (IFPR) Campus Paranavaí
Paranavaí – PR – Brasil

{allansjp06, adrielikethin.dossantos}@gmail.com,

{frank.willian, marcelo.terenciani}@ifpr.edu.br

Abstract. *This article explores the mathematics behind the Levenshtein distance, used method for calculating similarity between words, and its application in system integration. We use this method to compare course name entries from external systems and avoid the reinsertion of duplicate records, proposing an approach that improves data integration consistency, efficiency, and storage savings, as well as standardizing the information.*

Resumo. *Este artigo explora a matemática por trás da Distância de Levenshtein, um método utilizado para calcular a similaridade entre palavras, e sua aplicação na integração de sistemas. Utilizamos esse método para comparar entradas de nomes de cursos em sistemas externos e evitamos a reinserção de registros duplicados, propondo uma abordagem que melhora a consistência, eficiência e economia de armazenamento na integração de dados, além de padronizar as informações.*

1. Introdução

A diversidade no uso da linguagem e as variações nas palavras inseridas por diferentes usuários frequentemente causam problemas na interpretação por outros usuários, tanto como na integração/migração de sistemas. Essas variações podem surgir por erros de digitação, sinônimos ou até mesmo diferenças regionais e culturais. Quando palavras são inseridas de maneiras inconsistentes, prejudicam a comunicação entre sistemas, que acabam duplicando registros e comprometendo a qualidade do dados.

À medida que os desenvolvedores buscam integrar sistemas, a falta de padronização das entradas pode ocasionar erros na tomada de decisão, problemas na análise de dados e desperdício de armazenamento. Sendo assim, torna-se essencial a implementação de métodos que lidem com essa diversidade linguística.

No Instituto Federal do Paraná (IFPR) Campus Paranavaí existe um sistema para a gestão das Atividades Complementares dos cursos do campus, denominado Cronos – Sistema de Gerenciamento das Atividades Complementares [IFPR - Campus Paranavaí 2024]. Anualmente, com a chegada de novas turmas, são cadastrados novos alunos, com isso novas inserções de cursos são realizadas. Esse processo garante que todos os estudantes do campus estejam devidamente registrados e possam utilizar o sistema para acompanhar suas atividades complementares. Cada aluno possui um usuário e senha para autenticação no Cronos.

Com a possível criação de novos sistemas no campus, uma alternativa considerada é a padronização do processo de autenticação, visando garantir uniformidade em todas as plataformas de acesso. Para isso, a Interface de Programação de Aplicações (API, acrônimo do inglês

Application Programming Interface) já existente do sistema Cronos pode ser utilizada para centralizar a autenticação, permitindo que alunos e demais usuários utilizem um único usuário e senha para acessar diversos serviços institucionais. Essa abordagem simplifica o processo de acesso, tornando-o mais intuitivo e prático.

No entanto, ao utilizar o mesmo processo de autenticação em diversos sistemas, existe o risco de conflito entre os dados presentes em cada um deles, o que pode gerar inconsistências e comprometer a integridade das informações. Essa padronização, embora traga praticidade e mais segurança em alguns aspectos, precisa ser implementada considerando medidas para garantir a confiabilidade dos serviços.

Entre 2022 e 2023 foi desenvolvido um Sistema de Gestão de Empréstimos de Armários [de Andrade and Terenciani 2024] para informatizar o empréstimo de armários disponíveis no campus para o estudantes, que até o momento era feito manualmente. A solução garante a idoneidade dos registros, otimiza o acesso à informação pelos servidores e promove a autonomia dos alunos. A primeira versão do sistema foi criada como parte de um projeto acadêmico e, após ajustes e melhorias com base nas devolutivas dos usuários, foi implantada na biblioteca do campus em outubro de 2023.

No Sistema de Gestão de Empréstimos de Armários foi necessária a integração com o sistema Cronos para possibilitar a autenticação e o auto-cadastro dos alunos, aproveitando a base de dados existente. Após a integração, como os cursos são inseridos pelos servidores no sistema de empréstimos, testes com diferentes alunos com a comparação direta entre os nomes dos cursos, revelaram a inserção redundante de novos cursos devido a variações na escrita do nome do curso, como “Eng de software” e “ENGENHARIA DE SOFTWARE”, o que resultou em uma duplicidade desnecessária na base de dados.

Uma possível solução para esse problema é a utilização de algoritmos de aproximação de palavras, que identificam e possibilitam a correção das diferenças nas informações cadastradas. A Distância de Levenshtein [Lcvenshtcin 1966], por exemplo, é uma técnica que mede a diferença entre duas *strings*/palavras, identificando a quantidade mínima de operações necessárias (inserções, deleções ou substituições) para transformar uma palavra em outra, sendo útil para detectar variações ortográficas.

Além desse, outros algoritmos podem ser aplicados para comparação de *strings*, como o algoritmo de Boyer-Moore, que otimiza a busca de padrões em grandes conjuntos de dados, o algoritmo Colussi Reverso, que trabalha de forma reversa para identificar padrões de maneira eficiente, e o algoritmo Raita, que realiza comparações a partir de caracteres específicos para otimizar a busca. Esses métodos oferecem alternativas para verificar e alinhar dados entre sistemas, garantindo maior consistência nas informações armazenadas [Charras and Lecroq 2004].

Nesse sentido, o objetivo deste trabalho é apresentar a utilização do algoritmo de Distância de Levenshtein na integração entre o sistema Cronos e o sistema de gestão de empréstimo de armários do IFPR Campus Paranavaí, visando reduzir inconsistências nos dados. Com o uso desse algoritmo de aproximação de palavras, busca-se melhorar a consistência e uniformidade dos dados.

Para isso foi realizada uma análise documental e experimental dos dados de autenticação e cadastro dos alunos nos dois sistemas. Foram geradas variações para os nomes de cursos ofertados no IFPR Campus Paranavaí, utilizando inteligência artificial para simular inconsistências comuns, como erros de digitação, abreviações e variações ortográficas. Em seguida, o algoritmo de Distância de Levenshtein foi aplicado para medir e identificar as dis-

crepâncias entre os registros originais e as variações geradas. Os resultados foram analisados quantitativamente para avaliar a eficácia do algoritmo em reconhecer e corrigir as variações, proporcionando uma análise de viabilidade para a utilização dessa abordagem na integração dos sistemas.

Este artigo está organizado em cinco seções principais. Na próxima seção (Seção 2), será apresentada a revisão teórica sobre a Distância de Levenshtein. Em seguida, a Seção 3 descreverá o processo de geração de variações textuais com o uso de inteligência artificial para a posterior aplicação do algoritmo. A quarta seção (Seção 4) discutirá os resultados obtidos e a eficácia da abordagem proposta. Por fim, a Seção 5 apresentará as conclusões e sugestões para estudos futuros.

2. Distância de Levenshtein

A Distância de Levenshtein [Lcvenshtcin 1966], desenvolvida pelo matemático russo Vladimir Levenshtein em 1965, é uma medida que quantifica a diferença entre duas cadeias de caracteres. Esse conceito tem sido amplamente utilizado em diferentes áreas, como reconhecimento de fala, correção ortográfica e bioinformática.

A Distância de Levenshtein entre duas palavras x e y é definida como o número mínimo de operações necessárias para converter x em y. São permitidas operações de inserção, remoção e substituição de caracteres em um determinada palavra [Lcvenshtcin 1966].

A fórmula para calcular a distância é dada pela equação:

$$D(i,j) = \min \begin{cases} D(i-1,j)+1 & \text{(remoção)} \\ D(i,j-1)+1 & \text{(inserção)} \\ D(i-1,j-1)+c(x_i,y_j) & \text{(substituição)} \end{cases}$$

onde $D(i,j)$ é a distância entre os primeiros i caracteres de x e os primeiros j caracteres de y.

Como visto em [Charras and Lecroq 2004], a função de substituição é definida da seguinte forma:

- A função de substituição de a por a é igual a 0:

$$Sub(a,a) = 0 \quad \text{para } a \in \Sigma;$$

- A função de substituição de a por b é igual a 1 quando a e b são diferentes:

$$Sub(a,b) = 1 \quad \text{para } a,b \in \Sigma \text{ e } a \neq b;$$

- A função de deleção de a e a função de inserção de a são iguais a 1:

$$Del(a) = Ins(a) = 1 \quad \text{para } a \in \Sigma.$$

Então, $T[m-1,n-1]$ representa a Distância de Levenshtein entre x e y.

Exemplo 1: Comparação entre palavras diferentes (”ovo”e ”oca”)

Para este exemplo, vamos calcular a distância entre as palavras **”ovo”** e **”oca”**.

Passo a passo para preencher a tabela

1. Inicializamos uma matriz de tamanho $(m+1) \times (n+1)$, onde m é o comprimento de ”ovo”e n é o comprimento de ”oca”. 2. A primeira linha e a primeira coluna são preenchidas com números crescentes, representando as operações de inserir ou deletar caracteres para alcançar uma string vazia. 3. Cada célula da matriz é preenchida com base na menor operação: - Substituição (se os caracteres forem diferentes). - Inserção. - Deleção.

		o	c	a
	0	1	2	3
o	1	0	1	2
v	2	1	1	2
o	3	1	2	2

- A primeira linha e a primeira coluna representam as operações para transformar uma string vazia na outra. - O valor na última célula $D(3,3)$ é **2**, indicando que são necessárias duas operações para transformar ”ovo”em ”oca”: 1. Substituir a letra ”v”por ”c”. 2. Substituir a última letra ”o”por ”a”.

Dessa forma, a Distância de Levenshtein entre ”ovo”e ”oca”é igual a 2.

Exemplo 2: Comparação entre palavras iguais (”ovo”e ”ovo”)

Neste exemplo, vamos calcular a distância entre as palavras **”ovo”** e **”ovo”**.

		o	v	o
	0	1	2	3
o	1	0	1	1
v	2	2	0	1
o	3	2	1	0

- A última célula $D(3,3)$ é **0**, indicando que não são necessárias operações para transformar ”ovo”em ”ovo”. - Isso ocorre porque as palavras já são iguais.

Para saber o próximo valor sempre olhamos a célula a esquerda, acima e na diagonal superior a esquerda, se precisarmos fazer deleção, substituição ou inserção somamos 1 ao menor numero nessas células vizinhas. Vejamos:

		o
	0	1
o	1	?

Em [1, 1] comparamos se o é igual a o, como não precisamos fazer nenhuma operação, pegamos o menor valor nas celulas vizinhas (esquerda, acima e diagonal superior a esquerda), na diagonal encontramos o 0 que é menor entre os 3 valores, sendo assim a célula [1, 1] = 0, se a comparação fosse entre o e b, teriamos que somar um pois é o menor valor das vizinhas mais uma operação.

Exemplo 3:

Neste exemplo, calcularemos a Distância de Levenshtein entre as palavras “info” (comprimento 4) e “informática” (comprimento 11).

Cada célula da matriz $dp[i][j]$ representa o número mínimo de operações para transformar o prefixo da palavra "info" até a posição i no prefixo de "informática" até a posição j. Vamos construir a matriz linha por linha:

A primeira linha e a primeira coluna representam as transformações de uma *string* vazia até um comprimento específico.

- **Primeira linha**: Transformar uma *string* vazia em um prefixo de "informática" (insere cada caractere de "informática").
- **Primeira coluna**: Transformar um prefixo de "info" em uma *string* vazia (remove cada caractere de "info").

	–	i	n	f	o	r	m	á	t	i	c	a
–	0	1	2	3	4	5	6	7	8	9	10	11
i	1	0	1	2	3	4	5	6	7	8	9	10
n	2	1	0	1	2	3	4	5	6	7	8	9
f	3	2	1	0	1	2	3	4	5	6	7	8
o	4	3	2	1	0	1	2	3	4	5	6	7

Etapas de Preenchimento

- **Primeira Linha** (Transformação de vazio para "informática"):
  Para passar de uma string vazia para um prefixo de "informática" com comprimento j, são necessárias j inserções, por isso os valores de 0 a 11.
- **Primeira Coluna** (Transformação de "info" para vazio):
  Similarmente, transformar "info" até um comprimento i em uma string vazia exige i remoções.
- **Outras Linhas e Colunas**:
  Para cada célula $dp[i][j]$:
  - Se o caractere de "info" e "informática" são iguais, usamos o valor da célula $dp[i-1][j-1]$ (nenhuma nova operação é necessária).
  - Se são diferentes, consideramos:
    - **Remover** um caractere de "info" (usando $dp[i-1][j]+1$),
    - **Inserir** um caractere em "info" (usando $dp[i][j-1]+1$),
    - **Substituir** o caractere de "info" (usando $dp[i-1][j-1]+1$),
  - O valor final da célula é o mínimo entre essas três operações.

A Distância de Levenshtein entre "info" e "informática" é o valor na célula $dp[4][11]$, que é **7**. Isso significa que são necessárias 7 operações para transformar "info" em "informática".

2.1. `fast-levenshtein` - Algoritmo de Levenshtein em Javascript

A biblioteca `fast-levenshtein` [Nair 2015] é uma implementação do algoritmo de distância de Levenshtein, que permite calcular a diferença entre duas cadeias de caracteres. Essa biblioteca é amplamente utilizada em aplicações que requerem comparação de *strings*, como verificação de similaridade em entradas de texto, correção automática, busca aproximada, e deduplicação de dados.

A principal função desta biblioteca é calcular o número mínimo de operações (inserções, deleções ou substituições) necessárias para transformar uma *string* em outra. O resultado é uma métrica que quantifica a similaridade entre as strings: quanto menor a distância,

mais semelhantes são os textos. Isso se mostra especialmente útil em sistemas que lidam com dados não padronizados ou entradas de usuários, como em integrações de sistemas, onde pequenas variações nos nomes podem causar duplicações ou erros de registro. A biblioteca é fácil de usar e pode ser integrada em projetos JavaScript para aprimorar a manipulação e comparação de strings de maneira eficaz.

A utilização da biblioteca se deve ao trabalho em que está inserida, que faz um estudo comparativo entre Java e Javascript, portanto a utilização de bibliotecas em JavaScript complementa a análise ao demonstrar como as linguagens lidam com problemas similares de maneiras distintas. Isso permite destacar as diferenças e semelhanças na implementação de algoritmos e no uso de bibliotecas externas, fornecendo uma visão mais abrangente das capacidades e ferramentas disponíveis em cada linguagem.

3. Geração de variações textuais com o uso de inteligência artificial

O Instituto Federal do Paraná - campus Paranavaí possuí vários cursos, os quais estão presentes no Cronos, que são: Técnico em informática, Licenciatura em química, Técnico em meio ambiente, Tecnologia em análise e desenvolvimento de sistemas, Técnico em agroindústria, Técnico em eletromecânica, Bacharelado em engenharia elétrica, Técnico em mecatrônica, Técnico em alimentos e Engenharia de software.

Para a realização dos testes, foi cadastrado no sistema de empréstimos os citados cursos com os devidos nomes, da maneira que foram escritos logo acima, prática padrão no momento da implantação do sistema de armários, que é feito por um servidor que digita manualmente. Posteriormente solicitado ao ChatGPT [OpenAI 2023] que gerasse uma lista com 100 variações dos cursos, incluindo erros gramaticais, com o objetivo de simular uma API com variações gramaticais nos nomes dos cursos.

O comando: ”Gere 100 variações dos seguintes termos: Técnico em informática, Licenciatura em química, Técnico em meio ambiente, Tecnologia em análise e desenvolvimento de sistemas, Técnico em agroindústria, Técnico em eletromecânica, Bacharelado em engenharia elétrica, Técnico em mecatrônica, Técnico em alimentos e Engenharia de software. Inclua nas variações erros de gramatica.”. Gerou 100 registros que estão dispostos na Tabela 1.

Para este teste, vamos simular a entrada de dados da API, com erros gramaticais e variações nos nomes dos cursos, com o objetivo de mitigar a duplicação de registros similares, garantindo que só sejam criados novos registros quando exceder a tolerância definida para a Distância de Levenshtein.

4. Resultados e Discussões

Esta sessão apresenta a implementação da integração, a variação do código para a realização dos testes, tanto como seus resultados.

4.1. Implementação na Integração Entre Sistemas

A Distância de Levenshtein foi aplicada como parte da integração entre o sistema de gerenciamento de empréstimos e o sistema Cronos, utilizado para o registro de estudantes. Quando um estudante realiza o cadastro no sistema de empréstimos, os dados necessários, incluindo o nome do curso, são captados diretamente do Cronos via API.

Neste processo de integração, foi fundamental implementar um mecanismo que identificasse e tratasse cursos que, apesar de semanticamente semelhantes, possuem variações mínimas no nome. Para isso, a Distância de Levenshtein foi empregada na comparação dos nomes dos

Tabela 1. Variações de Cursos

Entradas		
Técnico em informátic	Técnico em informática 1	Técnico em Informática
Técnico em Informatica 2	Técnico em Inforamática	Técnco em Informática
Técnico em Informatica 3	Técnioco em informática	Técnico em Informática 4
Técinico em informática	Licenciatura em químico	Licenciatura em quimica 1
Licenciatura em Química	Licenciatura em Química 2	Licenciatura em Qímica
Licenciatura em Quimica 3	Licenciatura em Quimica 4	Licenciatura em Qumica
Licenciatura em Química 5	Licenciatura em Químic	Licenciatura em Química 6
Técnico em Meio Ambinete	Técnico em meio ambiente 1	Técnico em Meio Ambiente
Técnioco em Meio Ambiente	Técnico em Meioambiente	Técncio em Meio Ambiente
Tecnico em Meio Ambiente 2	Técnico em Miao Ambiente	Técinico em Meio Ambiente
Tecnologia em análise e desenolvimento de sistemas	Tecnologia em Análise e Desenvolvimento de Sistmas	Tecnologia em Análise e Desenvolvimento de Sistmas 2
Tecnologia em Analize e Desenvolvimento de Sistema	Tecnoligia em Análise e Desenvolvimento de Sistemas	Tecnologia em Análise e Desenvovimento de Sistemas
Tecnologia em Análise e Deesnvolvimento de Sistemas	Tecnologia em análise e desenvolvimento de sisitemas	Tecnolgia em Análise e Desenvolvimento de Sistemas
Tecnolgía em Análise e Desenvolvimento de Sistemas	Técnico em Agroindústría	Técnico em Agroindústria 1
Técinico em Agroindústria	Técnico em Agroindústra	Técnico em Agroindústrias
Técnco em Agroindústria	Técnico em Agroindustia	Técnico em Agroindústría 2
Técnico em Agroindústria 3	Técnico em Agroindústria 4	Técnico em Eletromecâica
Técnico em Eletromecânica 1	Técnico em Eletromecânica	Técniico em Eletromecânica
Técnico em Eletrocemânica	Técnico em Eletromecânica 2	Técnico em Eletromecânica 3
Técnico em Eletromecânica 4	Bacharelado em Engenharia Elétrica	Bacharelado em Engenharia Elétrico
Bacharelado em Engenharia Elétrica 1	Bacharelado em Engenharia Elétrica 2	Bacharelado em Engenharia Elétrica 3
Bacharelado em Engenharia Elétrica 4	Bacharelado em Engenehria Elétrica	Bacharelado em Engenharia Elétricca
Bacharelado em Engenharia Elétrica	Bacharelado em Engenharia Elétrica 6	Técnico em Mecatrônica
Técnico em Mectrônica	Técnico em Mecatrônica 1	Técinico em Mecatrônica
Técinico em Mectrônica	Técnico em Meatrônica	Técnico em Mecatrônica 2
Técnico em Mecatronica	Técnico em Mecatrônica 3	Técnico em Alimentos
Técnico em Alimmentos	Técinico em Alimentos	Técnico em Alimemntos
Técnico em Alimontes	Técnico em Alimnetos	Técnico em Alimontos
Técnico em Alimentação	Técnico em Alimentos 2	Engenharia de Software
Engenharia de Sofware	Engenharia de Softeware	Engenharia de Software 1
Engenharia de Software 2	Engenhari de Software	Engenehria de Software
Engenharia de Software Avançado	Engenharia de Sofware 3	Engenharia de Sofware 4
Engenharia de Software Básico		

cursos captados do Cronos com os registros já existentes no banco de dados do sistema de empréstimos. O objetivo dessa implementação é evitar a duplicação de registros, mantendo a integridade dos dados e reduzindo a necessidade de correções manuais.

Por exemplo, se o curso “Eng de Software” já estiver registrado no banco de dados e o Cronos enviar o curso “ENGENHARIA DE SOFTWARE”, o sistema poderá identificar que os dois cursos são similares o suficiente para serem tratados como o mesmo registro, sem a necessidade de criar um novo curso.

A utilização de uma tolerância de 60% se deu devido ao caso citado, onde a distancia entre as palavras é de 0.68181 de similaridade (tolerância que, pode ser definida conforme regra

de negócio) baseada na Distância de Levenshtein, garante que apenas cursos significativamente diferentes sejam registrados como novos. A saída do método ”levenshteinMath”para este caso é:

```
Executing (default): SELECT "id_curso" AS "id", "nome", "ativo" FROM "
    tb_curso" AS "Curso" WHERE "Curso"."nome" = 'engenharia de software';
Executing (default): SELECT "id_curso" AS "id", "nome", "ativo" FROM "
    tb_curso" AS "Curso";
Distancia calculada por Levenshtein entre os nomes: 7
A similaridade das palavras e: 0.6818181818181819
O curso "Eng de Software" possui uma similaridade de: 0.6818181818181819
    que esta dentro da tolerancia de: 0.6
```

O trecho de código a seguir demonstra como é feito o cadastro do aluno no sistema de empréstimos utilizando a API do Cronos.

```
async function criaEstudanteCronos(ra, senha, telefone) {
  try {
    const user = await Estudante.findOne({ where: { ra } });
    if (user) {
      return { sucesso: false, cadastrado: true, mensagem: MensagemUtil.
    ESTUDANTE_JA_CADASTRADO };
    } else {
      const response = await cronosRequisicao(ra, senha);
      if (response.status === 401) {
        return { sucesso: false, mensagem: MensagemUtil.
    LOGIN_SENHA_INCORRETA };
      } else if (!response.ok) {
        return { sucesso: false, mensagem: MensagemUtil.CRONOS_AIR_OUT };
      } else {
        const data = await response.json();
        let primeiroNome = data.nome.split(' ')[0];
        let nomeRestante = data.nome.split(' ').slice(1).join(' ');
        let encriptedPassword = converteSenhaParaSha256Hex(senha);
        try {
          const curso = await buscaCursoOuCriaCurso(data.alunoTurma[0].
    curso.toLowerCase());
          const estudante = await criaEstudante(ra, primeiroNome,
    nomeRestante, data.email, encriptedPassword, curso.id, telefone);
          return { sucesso: true, mensagem: MensagemUtil.CRONOS_SUCESSO };
        } catch (innerError) {
          return { sucesso: false, mensagem: innerError.message };
        }
      }
    }
  } catch (error) {
    return { sucesso: false, mensagem: error.message };
  }
};
```

Aqui mostra como é feito a criação do curso, verificando antes se não existe no banco, posteriormente compara similares, e não encontrando similar, insere como um novo:

```
const buscaCursoOuCriaCurso = async (curso) => {
  try {
    const cursoExistente = await Curso.findOne({ where: { nome: curso } });
    if (cursoExistente) {
      return cursoExistente;
    } else {
```

```
      const cursoAproximado = await levenshteinMath(curso);
      if (cursoAproximado) {
        return cursoAproximado;
      }
    }
    const novoCurso = await Curso.create({ nome: curso, ativo: true });
    return novoCurso;
  } catch (error) {
    throw new Error(`Erro ao buscar ou criar curso: ${error.message}`);
  }
};
```

Abaixo está o código utilizado para verificar a similaridade entre nomes de cursos.

```
const { Curso } = require("../models");
const levenshtein = require("fast-levenshtein");

const levenshteinMath = async (curso) => {
    try {
        const tolerancia = parseFloat(process.env.TOLERANCIA_A_CURSOS) || 0.6;
        let cursos = await Curso.findAll();
        let cursoEncontrado = null;
        let maiorSimilaridade = 0;

        cursos.forEach(cursoBanco => {
            let nomeBanco = cursoBanco.nome.toLowerCase();
            console.log(`Nome do curso no banco: ${nomeBanco} | Registro na API Cronos: ${curso}`);
            let distancia = levenshtein.get(curso, nomeBanco);
            console.log(`Distancia calculada por Levenshtein entre os nomes : ${distancia}`);
            let maxTamanho = Math.max(curso.length, nomeBanco.length);
            let similaridade = 1 - (distancia / maxTamanho);
            console.log(`A similaridade das palavras e: ${similaridade}`);

            if (similaridade > maiorSimilaridade) {
                maiorSimilaridade = similaridade;
                cursoEncontrado = cursoBanco;
            }
        });

        if (maiorSimilaridade >= tolerancia) {
            console.log(`O curso "${cursoEncontrado.nome}" possui uma similaridade de: ${maiorSimilaridade} que esta dentro da tolerancia de: ${tolerancia}`);
            return cursoEncontrado;
        } else {
            return null;
        }
    } catch (error) {
        throw new Error(`Erro na comparacao de cursos: ${error.message}`);
    }
};

module.exports = levenshteinMath;
```

Essa abordagem não só evita a reinserção de dados duplicados, mas também contribui

para a padronização dos registros, o que facilita consultas e relatórios futuros, além de otimizar a integração de sistemas, minimizando erros causados por pequenas variações nos nomes dos cursos. Abordagem que, se aplicada em diferentes cenários pode representar uma diminuição de suporte por erros de inserção e integração.

Análise dos Testes

Realizamos adaptações ao código para que itere sobre a lista de variações:

```
const cursos = [...]   // lista de variacoes

const buscaCursoOuCriaCurso = async () => {
  const resultados = [];

  for (const { curso } of cursos) {
      try {
          const cursoExistente = await Curso.findOne({ where: { nome: curso } });
          if (cursoExistente) {
              resultados.push(cursoExistente);
          } else {
              const cursoAproximado = await levenshteinMath(curso);
              if (cursoAproximado) {
                  resultados.push(cursoAproximado);
              } else {
                  const novoCurso = await Curso.create({ nome: curso, ativo: true });
                  resultados.push(novoCurso);
              }
          }
      } catch (error) {
          console.error(`Erro ao buscar ou criar curso "${curso}": ${error.message}`);
      }
  }

  return resultados;
};
```

Execução dos Testes

Análise da Porcentagem de Acerto

A porcentagem de acerto pode ser calculada dividindo o número de acertos pela quantidade total de registros inseridos vezes 100. Por tanto:

$$\text{Porcentagem de Acerto} = \left(\frac{100}{100}\right) \times 100 = 100\%$$

Essa análise demonstra a eficácia da técnica utilizada na comparação e padronização dos registros. O cálculo da porcentagem de acerto reflete a precisão da abordagem ao evitar duplicações e garantir que cada entrada na amostra foi corretamente associada a um registro existente no banco de dados. A Tabela 2 demonstra os registros que foram inseridos, a saída esperada e a saída.

A ausência de novos registros indica que o sistema conseguiu identificar com sucesso as correspondências, mesmo diante de possíveis variações nos dados de entrada, então, com a

Tabela 2. Tabela acertos (tolerância a 60%)

Variação de Entrada	Saída Esperada	Saída Produzida
Técinico em informática	Técnico em informatica	Técnico em informatica
Engenehria de Software	Engenharia de Software	Engenharia de Software
Técnico em Meatrônica	Técnico em mecatrônica	Técnico em mecatrônica
Licenciatura em Qumica	Licenciatura em Química	Licenciatura em química
Técnico em Alimontes	Técnico em alimentos	Técnico em alimentos
Técnico em Agroindústrias	Técnico em agroindústria	Técnico em agroindústria
Tecnologia em Análise e Desnvolvimento de Sistemas	Tecnologia em análise e desenvolvimento de sistemas	Tecnologia em análise e desenvolvimento de sistemas
Técnico em Meioambiente	Técnico em meio ambiente	Técnico em meio ambiente
Técnico em Eletrocemânica	Técnico em eletromecânica	Técnico em eletromecânica
Bacharelado em Engenharia Elétrico	Bacharelado em engenharia elétrica	Bacharelado em engenharia elétrica
Técnico em Inforamática	Técnico em Informática	Técnico em Informática
...	...	...

implementação, mesmo que os servidores cadastrem cursos com diferenças na escrita, quando um aluno se auto-cadastrar no sistema, o mesmo será capaz de identificar a similaridade e não realizar a inserção duplicada. O que reduz a correção manual futura, onde o servidor precisaria alterar aluno por aluno para o registro do curso correto, e somente depois, apagar a duplicata.

Isso evidencia que o método empregado é robusto e pode ser aplicado em cenários que demandam alta confiabilidade na integração e manipulação de dados, reduzindo significativamente a probabilidade de erros e inconsistências.

Com base nesses resultados, pode-se concluir que a técnica não apenas melhora a qualidade do banco de dados, mas também contribui para a eficiência operacional ao minimizar a necessidade de revisões manuais ou correções posteriores. Além disso, a taxa de acerto de 100% demonstra o potencial da abordagem para ser escalada e aplicada a bases de dados maiores, mantendo os mesmos padrões de desempenho e precisão.

Economia de Armazenamento em Integração de Sistemas

A utilização desse método também pode ser aplicada na integração de múltiplos sistemas que fornecem dados redundantes. Vamos considerar o cenário de um sistema de login que integra três APIs diferentes (incluindo o Cronos) para a captação de dados de 30.000 alunos, como é o caso do Instituto Federal do Paraná (IFPR). Nesse cenário, se 10% dos registros capturados de uma API apresentarem variações mínimas nos nomes (devido a erros de digitação ou diferentes convenções de nomenclatura), a Distância de Levenshtein pode ser usada para evitar a duplicação desses 3.000 registros.

Considerando que cada registro de aluno ocupa aproximadamente 1 KB no banco de dados, evitar a reinserção de 3.000 registros duplicados resultaria em uma economia de 3 MB de armazenamento. Esse valor pode parecer pequeno isoladamente, mas em sistemas de larga escala, onde múltiplos tipos de dados e entidades são envolvidos, a economia total de armazenamento pode ser significativamente maior. Além disso, esse processo reduz a carga de manutenção dos dados, minimizando o esforço manual de identificação e remoção de duplicatas em futuros ciclos de auditoria.

5. Conclusão

A aplicação da distância de Levenshtein no contexto de integração do Cronos com o Sistema de Gerenciamento de Empréstimos de Armários se mostrou eficaz para garantir que dados semanticamente semelhantes, mas com pequenas variações, sejam tratados de maneira apropriada. Com uma tolerância de 60%, foi possível evitar a duplicação de registros, enquanto novas inserções são realizadas apenas quando estritamente necessário. Isso contribui para uma maior integridade e padronização dos dados, além de reduzir o esforço manual para a correção de entradas duplicadas.

A abordagem pode ser facilmente aplicada em outros contextos. Um exemplo similar seria no cadastro de sabores de pizza em sistemas de *delivery*. Muitas vezes, pequenos erros de digitação ou variações nos nomes podem gerar múltiplos registros de um mesmo sabor, como "Calabresa" e "Calabreza". Usar a Distância de Levenshtein permitiria ao sistema identificar essas pequenas variações e sugerir ao usuário a utilização do sabor já existente, economizando espaço no banco de dados e evitando redundância.

Outro exemplo, seria no cadastro de endereços em sistemas de gerenciamento imobiliário ou logístico. Muitas vezes, pequenas variações na digitação ou formatação podem gerar múltiplos registros de um mesmo endereço, como "Av. Paulista, 1000" e "Avenida Paulista 1000".

Essa aplicação não apenas mantém os dados mais organizados, como também melhora a experiência do usuário ao evitar confusões e inconsistências. Além disso, assegura que relatórios e análises de dados sejam mais precisos e facilita a integração com outros sistemas e a automação de processos logísticos.

Dessa forma, a utilização da Distância de Levenshtein se apresenta como uma solução prática e eficiente para a normalização e integridade de dados em diversos domínios. Sua flexibilidade e facilidade de integração em diferentes sistemas garantem uma base de dados mais confiável e funcional, ao mesmo tempo em que otimizam processos e reduzem custos operacionais associados à manipulação de dados redundantes ou inconsistentes. A implementação dessa técnica reforça a importância de estratégias inteligentes para a gestão de dados em sistemas modernos.

Referências

Charras, C. and Lecroq, T. (2004). *Handbook of Exact String Matching Algorithms*. Accessed: 2024-11-02.

de Andrade, A. F. O. and Terenciani, M. F. (2024). Sistema de gerenciamento de armários. Accessed: 2024-11-02.

IFPR - Campus Paranavaí (2024). Cronos - sistema de gerenciamento das atividades complementares. Accessed: 2024-11-02.

Lcvenshtcin, V. (1966). Binary coors capable or 'correcting deletions, insertions, and reversals. In *Soviet Physics-Doklady*, volume 10, número 8.

Nair, R. (2015). fast-levenshtein: Levenshtein algorithm in javascript. `https://www.npmjs.com/package/fast-levenshtein`. Accessed: 2024-11-02.

OpenAI (2023). Chatgpt. A conversational AI model by OpenAI.