UNIVERSIDADE FEDERAL DO AMAZONAS - UFAM

INSTITUTO DE COMPUTAÇÃO - ICOMP

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA - PPGI

DANIEL DA COSTA XAVIER

UM MÉTODO EM DOIS NÍVEIS PARA COMPLEMENTAÇÃO AUTOMÁTICA DE SENTENÇAS

Manaus, AM

2019

DANIEL DA COSTA XAVIER

UM MÉTODO EM DOIS NÍVEIS PARA COMPLEMENTAÇÃO AUTOMÁTICA DE SENTENÇAS

Projeto Final apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Amazonas - UFAM como requisito parcial para a obtenção do grau de Mestre em Informática.

Orientador: Prof. D.Sc. Edleno Silva de Moura

Manaus, AM

2019

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Xavier, Daniel da Costa
X3u Um método em dois níveis para complementação automática de sentenças / Daniel da Costa Xavier. 2019
107 f.: il. color; 31 cm.

Orientador: Edleno Silva de Moura
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. complementação automática. 2. auto complementação. 3. complementar consulta. 4. preenchimento automático. 5. busca. I. Moura, Edleno Silva de II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Um método em dois níveis para Complementação Automática de Sentenças"

DANIEL DA COSTA XAVIER

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Edleno Silva de Moura - PRESIDENTE

Prof. Altigran Soares da Silva - MEMBRO INTERNO

Prof. Thierson Couto Rosa - MEMBRO EXTERNO

Manaus, 27 de Fevereiro de 2019

Resumo

Complementação automática de sentenças tolerante à erros de digitação tornou-se um recurso padrão em muitas aplicações que utilizam comandos textuais, especialmente para mecanismos de pesquisa, aumentando significativamente a qualidade da experiência de utilização dessas aplicações. Ao observar os métodos de complementação automática presentes na literatura, os principais fatores que indicam a viabilidade desses métodos para determinadas aplicações são o tempo de consulta e a quantidade de memória utilizada para indexação dos dados.

Este trabalho apresenta um novo método de complementação automática de sentenças que realiza a busca em dois níveis, possibilitando uma economia significativa de espaço de memória enquanto mantém o tempo de processamento de consultas aceitável em relação aos principais métodos presentes na literatura. Experimentos realizados em bases de dados de diferentes tamanhos indicam que o método apresentado reduz significativamente a quantidade de memória necessária para realizar a complementação automática quando comparado a trabalhos publicados na literatura. Tal resultado é obtido mantendo-se um tempo de processamento aceitável e, em alguns cenários, até melhor que os obtidos pelos melhores trabalhos encontrados na literatura.

Palavras-chave: auto complementação, complementação automática, preenchimento automático, complementar sentença, complementar consulta, processamento de texto, busca.

Abstract

Query autocompletion has become a standard feature in many applications that use textual commands, especially for search engines, significantly increasing the quality of the experience of using these applications. By observing the automatic complementation methods present in the literature, the main factors that indicate the viability of these methods for certain applications are the query process time and the amount of memory used to index the data.

This work presents a new method for query autocompletion that performs the search in two levels, allowing a significant saving of memory space, maintaining a performance in query processing time acceptable in relation to the main methods present in the literature. Experiments performed in databases of different sizes indicate that the presented method significantly reduces the amount of memory required to perform automatic complementation when compared to published works. This result is obtained by maintaining an acceptable processing time and, in some scenarios, can be even better than those obtained by the main works found in the literature.

Keywords: autocompletion, query autocompletion, sentence autocompletion, text processing, search.

"Transporte um punhado de terra todos os dias e fará uma montanha". Dedico este trabalho à todos os colegas, amigos e família que contribuíram, mesmo que tenha sido da mínima forma possível, para que este trabalho se tornasse realidade.

Agradecimentos

- Primeiramente à Deus por todo tipo de oportunidade que me permitiu desfrutar durante todo caminho de minha existência.

- Aos meus pais, Jairo e Dôra, que me proporcionaram o privilégio de poder chegar onde cheguei, sem que eu tivesse a grande preocupação de sanar minhas necessidades básicas de ser humano, através de muito esforço proporcionando o maior conforto que poderiam oferecer.

- Ao meu orientador, professor Edleno de Moura pela oportunidade de trabalhar com um dos maiores nomes da ciência mundial em sua área. Sou muito grato pelos conhecimentos e conselhos compartilhados, pra mim é uma honra.

- Aos meus parceiros Ruan, Victor e Anderson, que ajudaram muito na fase de concepção, contribuindo diretamente para que este trabalho fosse possível.

- Agradeço também à todos os professores do ICOMP, que contribuíram, desde minha graduação, para que eu pudesse chegar a este ponto. Agradeço especialmente também aos que marcaram não apenas meu caminho acadêmico, mas minha vida, professores Raimundo Barreto, Altigran da Silva, César Melo, Edson, Moisés Carvalho, Eduardo Feitosa, Eduardo Souto, Jõao Marcos, Arilo Dias e Bruno Gadelha. Obrigado por toda contribuição.

- Agradeço também à todos os funcionários da parte administrativa do ICOMP, que se esforçam para que não seja preciso nos preocupar com burocracias, resolvendo todo e qualquer tipo de problema que temos durante nossa jornada.

- Sou muito grato também à Lizandra Santos, que contribuiu de forma espetacular durante todo o período, aguentando minhas crises de estresse e ansiedade, por ser minha confidente pra tudo e saber me acalmar e apoiar em todos os momentos.

- À CAPES pelo auxílio financeiro para o desenvolvimento deste trabalho.

- À toda equipe do Teewa, com quem eu passei mais tempo do que minha própria família, Caio, Erick, Raphael, Ruan, Rúben, Marcos, Samantha, Gercidara, Henry, Ivo, Giulia, Rodrigo, Taigo, Daniel Zordan, Henrique.

- Aos amigos do PPGI/UFAM, com quem compartilhei momentos bons e ruins durante a caminhada: Rayol, Jordan, Bruno, Cassiano, Hendrio, Klinsman, Laiza, Raphael, Guilherme, Ludmila, Kevin, Victor, Anibrata, Denys, Roland, Fernando, Ralph, Tiago, Jackson, Maiara, Márcio Palheta, Luisa, Ivanilse e Josiane. Obrigado por tudo.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Complementação automática de sentenças tornou-se uma funcionalidade padrão e indispensável para mecanismos de busca ou sistemas que utilizam comandos textuais específicos. Várias aplicações populares utilizam complementação automática, como a busca do Google[1], Bing[2], Netflix[3], Facebook[4], Youtube[5], barra de endereços de navegadores web, agenda de contatos, sistemas de *e-commerce*, *etc.* Esses sistemas utilizam complementação automática para reduzir o número de caracteres digitados necessários para realização de consultas ou comandos, são utilizados também para diminuir a possibilidade de erros de consultas ou comandos cometidos pelos usuários (Arasu et al., 2006) (Zhou et al., 2016).

Os últimos trabalhos da literatura sobre complementação automática de sentenças indicam, como uma das funcionalidades padrão, o tratamento dos erros de digitação (Li et al., 2011) (Xiao et al., 2013) (Deng et al., 2016) (Zhou et al., 2016). Sendo assim, em um sistema de busca, ao digitar um prefixo de consulta contendo erros, o usuário é capaz de visualizar a consulta que tinha intenção de realizar nos resultados de sugestões. Cerca 20% das consultas realizadas possuem erros de digitação (Cucerzan and Brill, 2004) e a combinação entre complementação automática e correção de erros rende uma economia entre 40% e 60% no esforço de digitação do usuário (Ji et al.,

[1] https://www.google.com
[2] https://www.bing.com
[3] https://www.netflix.com
[4] https://www.facebook.com
[5] https://www.youtube.com

2009) (Zhou et al., 2016).

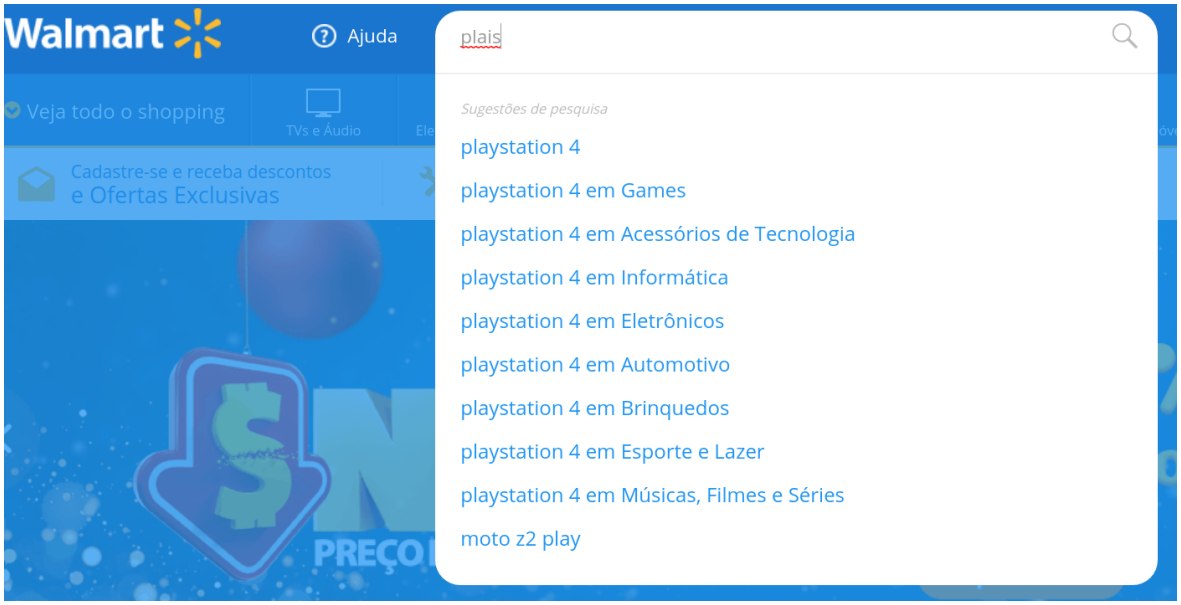

Figura 1.1: Utilização de complementação automática com tolerância a erro em uma plataforma de *e-commerce*.

A figura 1.1 ilustra a aplicação de complementação automática de sentenças em uma máquina de busca de uma plataforma de *e-commerce*[6]. No exemplo exibido, ao desejar encontrar o produto "*playstation 4*", o usuário digita a consulta com um erro inserido ("*plais*") em relação ao prefixo correto ("*plays*"), o sistema mostra as complementações corretas disponíveis mesmo com o erro inserido. A importância do sistema de complementação automática tolerante a erros nesse caso causou uma experiência satisfatória para o usuário, o que não ocorreria caso o erro digitado não fosse tratado, podendo resultar em uma eventual desistência da compra na plataforma caso o usuário não percebesse que havia cometido esse erro.

O problema de complementação automática de sentenças possui como um dos principais requisitos a eficiência de processamento de consultas, visto que as consultas são realizadas enquanto o usuário digita e, para que o usuário tenha uma experiência fluida, cada resultado das consultas deve ser exibido ao usuário antes que o próximo caractere seja digitado, evento que ocorre em uma média entre 100 e 200 milisegundos (ms) (Alsultan and Warwick, 2013), ou seja, o ideal é que as consultas sejam realizadas em, no

[6] https://www.walmart.com.br

máximo, 100 ms para evitar atrasos perceptíveis (Ji et al., 2009).

O crescimento do número de usuários utilizando sistemas *online* e do fluxo de dados e informações necessárias a serem armazenadas em sistemas atuais aumentam a demanda para formas de persistência de dados mais eficientes (Das et al., 2010) (Marx, 2013). Sendo assim, os sistemas de complementação automática de sentenças também precisam otimizar a quantidade de memória necessária para sua utilização.

Alguns dos principais métodos para o problema de complementação automática de sentenças (Chaudhuri and Kaushik, 2009)(Li et al., 2011) utilizam indexação das cadeias de caracteres dos dados em uma estrutura *trie*[7] e o conceito de nós ativos. Os nós ativos são os nós indexados na *trie* em que o prefixo formado pelos caracteres presentes no caminhamento entre o nó raiz e esses nós possuem distância de edição menor ou igual ao limite de distância de edição definido. O número de nós ativos é um fator importante para o desempenho do método, pois tanto a performance de processamento de consultas quanto o consumo de memória são proporcionais ao número de nós ativos e, no caso desses métodos, quanto maior o limite de distância de edição definido, mais nós ativos serão utilizados.

Outra abordagem (Xiao et al., 2013) propõe uma solução de tratamento de erros através do conceito de geração de vizinhança, que basicamente realiza a indexação na *trie* gerando caminhos de nós extras como possíveis erros a serem tratados. Apesar de conseguir uma boa eficiência em tempo de consulta, o tamanho do índice criado gera um custo de memória inviável para grandes bases de dados.

Outros dois métodos de complementação automática de consultas mais recentes não utilizam o conceito de nós ativos como solução principal. O primeiro, proposto por Zhou et al. (2016), apresenta um conceito baseado no pré-processamento de vetores de edição mapeados em estados de um autômato, melhorando a eficiência no processamento de consultas, além de apresentar estabilidade mesmo para distâncias de edição mais elevadas. O outro método, proposto por Deng et al. (2016), apresenta uma

[7] A estrutura de dados *trie*, árvore de prefixos, ou árvore digital (De La Briandais, 1959) (Fredkin, 1960) é uma árvore que armazena um conjunto de palavras e são capazes de recuperar qualquer termo indexado em um tempo proporcional ao seu comprimento, independentemente do tamanho e quantidade total das palavras armazenadas.

solução baseada em uma estrutura de comparação de cadeias de caracteres capaz de reduzir cálculos redundantes, que processa as sugestões com base em correspondência de caracteres.

Neste trabalho é apresentado e desenvolvido uma nova solução para realizar a tarefa de complementação automática reduzindo o custo de espaço de memória necessário para processamento de consultas. Para tanto, é proposto e avaliado um método para complementação automática que realiza uma busca em dois níveis, que combina busca em uma estrutura de índices com busca sequencial. Esse método foi avaliado em termos de custo de utilização de memória e de tempo para o processamento de consultas em relação aos métodos presentes na literatura, que representam o estado da arte para o problema de complementação automática de sentenças.

No primeiro nível, apenas uma parte inicial com n caracteres da *string* que representa o item é indexado em uma estrutura de dados *trie*, que pode realizar processamento textual com tratamento de erros, baseado no algoritmo ICAN (Ji and Li, 2011). Para toda consulta realizada em que o prefixo tiver comprimento menor que n, é possível obter os resultados definitivos do processamento de complementação apenas com o processo do primeiro nível.

Quando o prefixo de consulta possui comprimento maior que n, o primeiro nível funciona como filtro, pois o conjunto de resultados pode possuir itens irrelevantes para a consulta. Sendo assim, é necessário a utilização do segundo nível do método, que processa sequencialmente cada item do conjunto de resultados do primeiro nível, verificando a similaridade dos itens com o prefixo de consulta. A filtragem de itens realizada pelo primeiro nível resulta em uma quantidade muito pequena em relação à quantidade de itens na base de dados total, diminuindo os custos de processamento no segundo nível.

Por meio dos experimentos é possível afirmar que o método de dois níveis aqui proposto consegue realizar complementação automática em um tempo aceitável e consumindo uma quantidade de memória significativamente menor do que a realizada por métodos que estão hoje no estado da arte para a realização dessa tarefa.

A economia de memória que método alcança em relação aos demais métodos varia de acordo com o contexto, principalmente em relação à base de dados. Porém, segundo os experimentos realizados, essa economia pode variar de cerca de 15% para bases menores, até 95% para bases de dados maiores.

Quanto ao tempo de processamento, as médias de tempo obtidas pelo método de dois níveis é equivalente aos demais métodos, com aumentos de tempo mais significativos apenas quando é necessária a utilização do segundo nível, possuindo em alguns casos uma média de tempo melhor do que o ICAN.

O texto está organizado da seguinte forma: o capítulo 2 apresenta conceitos essenciais que servem como base para o entendimento do trabalho, onde aborda trabalhos significativos na literatura relacionados com a proposta do método apresentado. No capítulo 3 é descrita a implementação do método de auto complementação em dois níveis. No capítulo 4 são dispostos os resultados dos experimentos e, por fim, no capítulo 5 as conclusões do trabalho e as propostas de trabalhos futuros são apresentados.

Capítulo 2

Conceitos e trabalhos relacionados

Neste capítulo serão apresentados conceitos sobre sistemas de complementação automática de sentenças, busca em dois níveis e cálculo de similaridade entre cadeias de caracteres. Inicialmente, introduzimos o problema principal de complementação automática e um resumo sobre os principais métodos citados na literatura. Logo após, serão apresentadas algumas técnicas de cálculo de similaridade de termos.

2.1 Complementação automática de sentenças

Complementação automática, preenchimento automático ou auto complementação de sentenças é uma funcionalidade muito presente em aplicações que utilizam busca ou comandos textuais, sendo útil em vários ambientes. Tais sistemas apresentam, conforme o usuário digita partes de sua consulta, uma lista de sugestões apropriadas para conclusão do que já foi digitado. Os principais objetivos são ajudar a orientar a digitação do usuário e reduzir o esforço de digitação. Conforme os métodos e tecnologias acerca da complementação automática foram evoluindo, a possibilidade de informar e sugerir alternativas para erros de digitação do usuário também se tornou um objetivo importante (Chaudhuri et al., 2006) (Li et al., 2011) (Xiao et al., 2013).

Como o processo de comparação de textos realizado em um mecanismo de complementação automática ocorre por meios de prefixos de palavras, que são intenções do que o usuário deseja digitar, o cálculo de similaridade entre cadeias de caracteres

precisa ser realizado por meio de técnicas de comparação baseada em caracteres.

Em essência, a complementação automática de sentenças consiste em sugerir conclusões válidas de uma palavra ou frase de pesquisa inserida parcialmente com a intenção de orientar e minimizar a digitação do usuário. O conceito de complementação automática vem sendo estudado durante vários anos, onde a ideia principal é usar técnicas de recuperação de informação, modelos de linguagem e aprendizado de máquina para sugerir possíveis conclusões (Bast and Weber, 2006) (Nandi and Jagadish, 2007) (Chaudhuri and Kaushik, 2009).

Uma funcionalidade importante para solução do problema de complementação automática é permitir erros nos prefixos de consultas realizadas pelos usuários, como troca de caracteres, caracteres não digitados e caractere inserido de forma errada. Soluções que dispõem desse recurso são caracterizados como complementação automática tolerante a erros.

A indicação da complementação automática exata contribui para guiar o usuário durante a digitação, poupando significativas quantidades de cliques e, em certa medida, reduz a probabilidade de erros serem cometidos. Entretanto, a complementação automática tolerante a erros pode ser muito interessante em diversas situações. Por exemplo, se o usuário digita um caractere errado, a complementação automática exata já não indicaria o resultado desejado, o que é uma ocorrência bastante comum em digitações rápidas e nas relações entre fonética e o modo de escrita ortográfica. A complementação tolerante a erros também é útil para manter a integridade dos dados, uma vez que ser capaz de navegar rapidamente nos resultados aproximados pode reduzir as entradas duplicadas.

O mapeamento sistemático de trabalhos relacionados aos estudos sobre complementação automática de sentenças realizado, indica que a principal estrutura de dados utilizada para indexação dos termos a serem processados é uma versão de árvore[1] denominado como *trie*.

[1] Árvores são estruturas de dados formadas por elementos chamados de nós ou nodos dispostos em forma hierárquica. O primeiro nó, denominado como raiz, é ligado aos seus nós filhos, que por sua vez são ligados aos seus nós e assim sucessivamente até chegar ao nó folha, ou nó terminal, que não possui nós filhos. (Szwarcfiter and Markenzon, 1994)

2.1.1 Trie

A estrutura de dados *trie*, árvore de prefixos, ou árvore digital (De La Briandais, 1959) (Fredkin, 1960) é uma árvore que armazena um conjunto de palavras e é capaz de recuperar qualquer termo indexado em um tempo proporcional ao seu comprimento, independentemente do tamanho e quantidade total das palavras armazenadas (Baeza-Yates and Ribeiro-Neto, 2013).

A *trie* é composta por um conjunto de registros, aqui denominados como nós, que representam os caracteres dos termos onde o conjunto de todos os caracteres possíveis a serem utilizados é definido como alfabeto de caracteres. Cada nó possui basicamente a referência ao caractere representado e as referências aos nós filhos, cujo limite máximo é o tamanho do alfabeto de caracteres (Fredkin, 1960) (Liang, 1983). Cada nó que representa o fim de uma cadeia de caractere indexada precisa ser marcado com um indicador informando ser o fim de um item, distinguindo esse nó dos demais.

Inserir um item em uma *trie* é uma atividade simples. Cada caractere da cadeia de caractere de entrada é inserido como um nó individual na *trie*. O caractere de entrada atua como um índice nos filhos das listas de referências. Se a cadeia de caractere de entrada existe na *trie*, simplesmente marca-se seu último nó de entrada como fim de um item. Caso a cadeia de caractere de entrada ainda não exista na *trie*, é necessário construir novos nós, marcando o último nó como fim da cadeia. O comprimento dos itens de entrada determinam a profundidade da *trie*.

Realizar uma busca é semelhante à operação de inserção, no entanto, compara-se apenas os caracteres e caminha-se para para o nível abaixo. A busca pode terminar ao encontrar um marcador de fim de item, sendo assim, o prefixo de consulta existirá na *trie*. Caso contrário, a busca termina sem examinar todos os caracteres do prefixo de consulta, já que ela não está presente na *trie*.

Os custos de inserção e pesquisa são O(M) onde M é o tamanho da cadeia de caracteres a ser buscada, ou inserida, na *trie*. Nota-se que o custo é baixo e independe do número de elementos inseridos, o que torna a *trie* uma estrutura extremamente competitiva para a tarefa de complementação automática de sentenças.

Como os sistemas complementação automática de sentenças atuais precisam utilizar técnicas para tratar erros, é necessário utilizar mecanismos junto à *trie* para que os erros sejam identificados e sugestões corretas sejam disponibilizadas durante as consultas. Nos tópicos a seguir serão abordados trabalhos que levam em consideração o tratamento de erros como parte da solução de complementação automática de sentenças.

Estendendo a complementação automática para tolerar erros

Em Chaudhuri and Kaushik (2009) é apresentada uma abordagem de complementação automática de sentenças que leva em consideração os erros de digitação das entradas de texto. Até então, a forma mais comum de complementação automática para o cenário onde há uma lista de sentenças sendo pesquisadas se dava por comparação exata, ou seja, sem levar em consideração os erros de digitação inseridos na entrada.

O foco desse trabalho está no cenário em que, dado um conjunto T de cadeias de caracteres indexadas em uma estrutura e um prefixo de consulta dado pelo usuário, sejam sugeridas as cadeias de caracteres de T que são correspondentes ao prefixo de consulta. Nesse caso, há uma cadeia de caracteres parcialmente digitada e, em resposta, a complementação automática produz uma lista de complementos. O prefixo de consulta pode ser modificado através de alguma edição do usuário, em geral, o usuário pode modificar a cadeia de caractere inserindo ou excluindo um caractere em qualquer posição da mesma.

Chaudhuri and Kaushik (2009) propõem dois algoritmos de complementação automática tolerantes a erros. O primeiro é baseado nos algoritmos de distância de edição baseados em n-grama[2], até então a abordagem mais eficiente para complementação aproximada de sentenças (Arasu et al., 2006) (Chaudhuri et al., 2006) (Xiao et al., 2008). O segundo é um algoritmo baseado em *trie*, que mostra-se ser mais adequado à complementação automática de sentenças. O método apresentado em Chaudhuri and Kaushik (2009) utiliza distância de edição clássica como medida de similaridade entre cadeias de caracteres, juntamente com pré-computação de cadeias de caracteres curtas.

[2]Um n-grama de uma cadeia de caractere S é um prefixo contíguo de S de comprimento q. O conjunto q-grama é a lista de todos os q-gramas de S.

No algoritmo baseado em *trie*, o conjunto dos itens é organizado em uma estrutura em nós e arestas. A Figura 2.1 mostra um exemplo da *trie* gerada pelo método de Chaudhuri and Kaushik (2009). Por meio de uma *trie* é possível calcular potenciais erros de edição em qualquer posição da sentença. Ao processar um dado prefixo de consulta, o algoritmo calcula a distância de edição para todos os prefixos de comprimento máximo igual ao tamanho da mesma, somado ao limite de distância de edição. Os nós que satisfazem esse limite são denominados como nós correspondentes válidos da *trie*. O estudo de Chaudhuri and Kaushik (2009) mostra que para qualquer extensão k do prefixo de consulta, há no máximo $2k + 1$ prefixos que estão dentro da distância k, e que esses prefixos correspondem a um caminho contíguo na *trie*.

A Figura 2.1 ilustra como o algoritmo opera em um conjunto composto por três itens - *Johnny*, *Josef* e *Bond*. A cadeia de entrada que está sendo inserida é *Jonn*. Os nós sombreados denotam os nós válidos e as distâncias de edição são mostradas ao lado deles.

ICAN

Li et al. (2011) tratam o problema de complementação automática de sentenças utilizando o conceito de nós ativos para processamento de similaridades de consultas permitindo erros. Seja p um prefixo de consulta e led um limite de distância de edição. Chama-se um nó n da *trie* de nó ativo de p em relação a led, se a distância de edição (ed) entre a cadeia de caractere formada da raiz da *trie* até n e p apresentar distância de edição menor ou igual que led, ou seja $ed(n, p) \leq led$. Os termos referenciados nos nós folhas descendentes dos nós ativos são denominados como termos semelhantes de p. Por exemplo, considere a *trie* na Figura 2.2, supondo que o limite de distância de edição $led = 2$ e um usuário digita um prefixo $p =$ "nlis". Como ilustrado pela Figura 2.2(e), cada um dos prefixos "li", "lin", "liu" e "luis" tem uma distância de edição para p dentro de led. Assim, os nós 13, 15, 17 e 22 são nós ativos de p. As palavras semelhantes para o prefixo são "li", "lin", "liu" e "luis".

As *tries* formadas na indexação dos termos utilizam um algoritmo incremental para

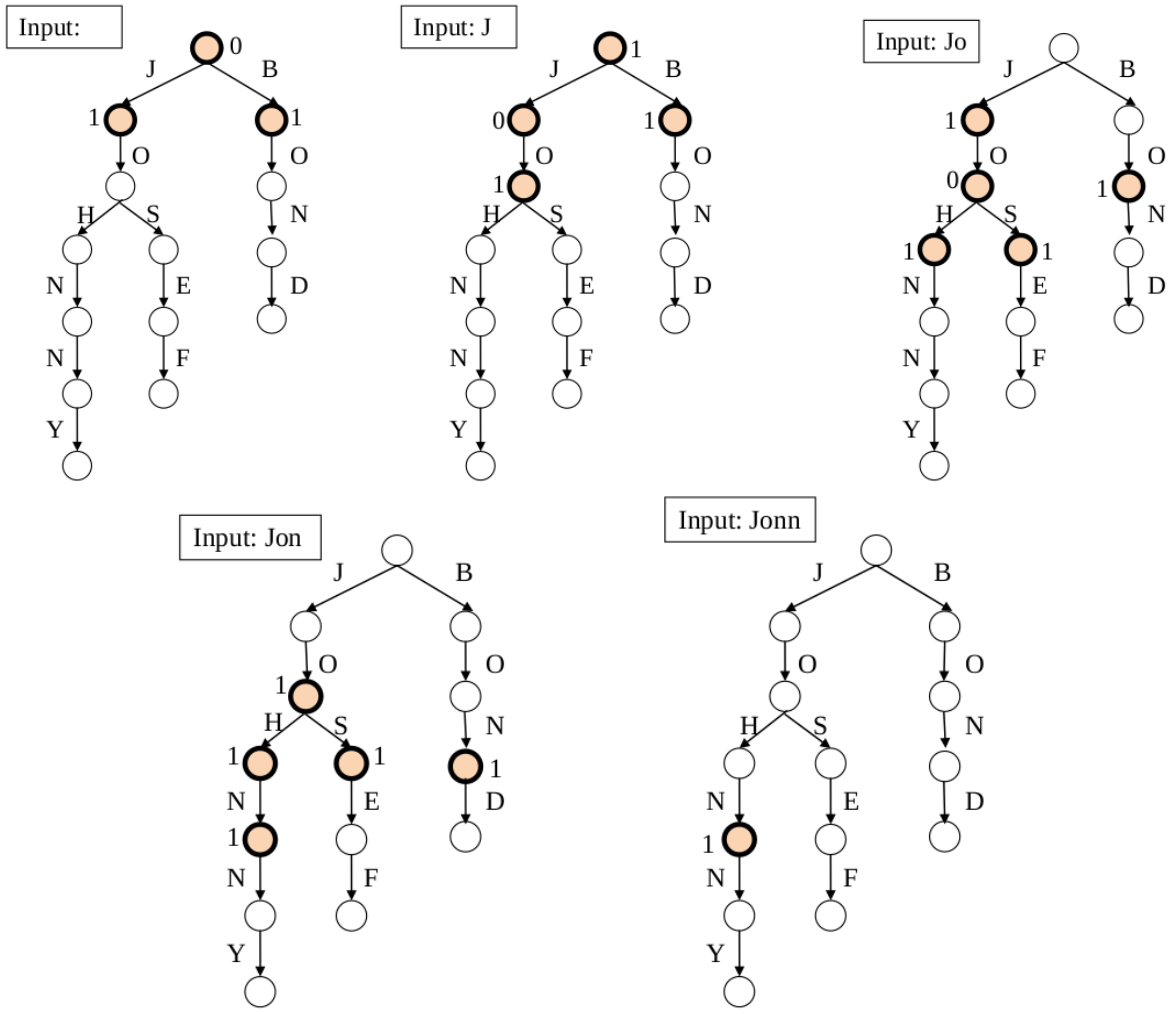

Figura 2.1: Exemplo de busca do prefixo de consulta "Jonn" em uma *trie* (com limite de distancia de edição 1). Passos na sequencia: Inicialização da consulta, consulta 'J', consulta 'Jo', consulta 'Jon', consulta 'Jonn'. Os nós sombreados denotam os nós válidos e as distâncias de edição são mostradas ao lado deles. Fonte: (Chaudhuri and Kaushik, 2009)

calcular os prefixos de cadeias de caracteres semelhantes de acordo com o prefixo dado. Dados sobre a distância de edição de cada prefixo para cada caractere são armazenados e atualizados dinamicamente de acordo com a atualização do texto da consulta. Ou seja, cada vez que o prefixo da consulta é atualizado, os nós formados pelos caracteres na *trie* podem ser ativados ou desativados, como pode ser observado na Figura 2.2. Essa técnica facilita a possibilidade de modificação da distância de edição suportada pelo cálculo de similaridade.

Buscando definir um subgrupo de nós ativos que poderiam ser usados para computar todos os termos similares para o prefixo da consulta de forma incremental e eficiente,

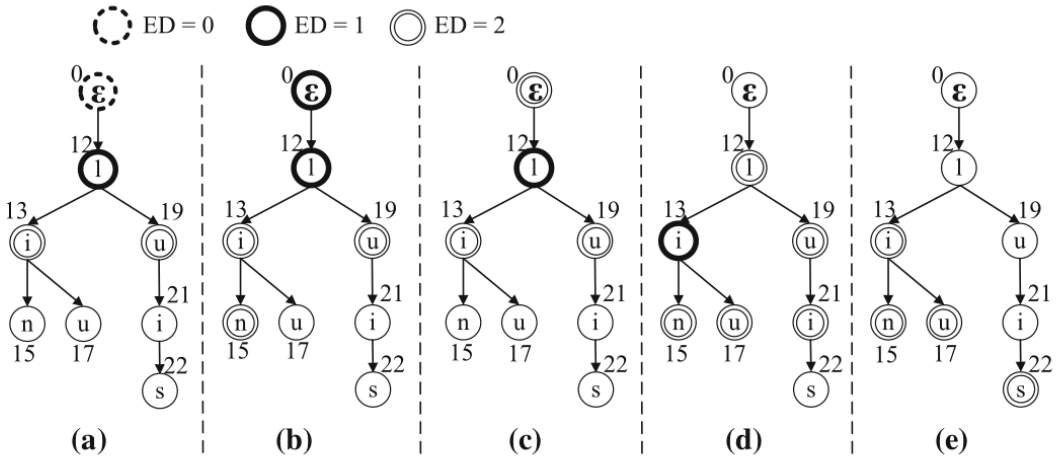

Figura 2.2: Exemplo de busca do prefixo de consulta "nlis" em uma *trie* parcial (com limite de distancia de edição igual a 2). (a) Inicialização, (b) consulta 'n', (c) consulta 'nl', (d) consulta 'nli', (e) consulta 'nlis'. Fonte: (Li et al., 2011)

o trabalho também apresenta uma variação do método, o ICPAN (*Incrementally Computing Pivotal Active Nodes*, ou Computação Incremental de Nós Ativos Principais), que utiliza apenas os nós ativos essenciais. Sendo assim, o algoritmo ICPAN, durante a digitação do prefixo da consulta, utiliza apenas o conjunto necessário de nós ativos para a realização dos cálculos.

IncNGTrie

Até a apresentação de Xiao et al. (2013), os métodos de complementação automática de sentenças com correção de erros utilizavam o paradigma de indexar as sequências de dados em uma trie e percorrê-lo incrementalmente para calcular a distância de edição entre os nós da trie e a consulta, apenas os nós que satisfazem as restrições de distância de edição são mantidos, sendo chamados de nós ativos. A eficiência desses métodos depende diretamente da quantidade de nós ativos que são gerados e mantidos durante a consulta, sendo possível que, para um número de limite de distância de edição igual a 3, por exemplo, os nós dos quatro primeiros níveis da trie sejam nós ativos.

Sendo assim Xiao et al. (2013) propõem melhorar drasticamente o desempenho da consulta através de pré-processamento dos dados e criando um índice de grande

proporção, mantendo apenas um pequeno conjunto de nós ativos durante a consulta.

Para isto, é proposto um algoritmo baseado em geração de vizinhança, o "IncNG-Trie", onde são indexadas também as possibilidades de prefixos para tratamento dos erros de digitação. Como pode ser visualizado na Figura 2.3, os termos "text" e "test" são indexados de tal forma que, além dos nós possuírem referência para os nós filhos que representam caracteres específicos dos termos, também possuem referência a um nó que representa um caractere genérico que permite a continuidade do caminhamento pela árvore caso haja na entrada um caractere desconhecido.

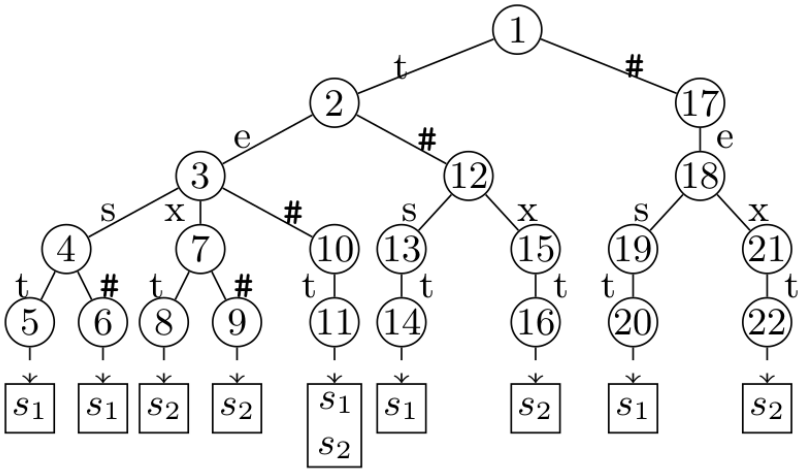

Figura 2.3: Exemplo da *trie* do IncNGTrie gerada para ($s1 = test$, $s2 = text$) Fonte: (Xiao et al., 2013)

A avaliação experimental realizada pelo estudo em conjuntos de dados reais em baixa escala demonstra que o IncNGTrie supera as soluções até então existentes em até duas ordens de grandeza, em termos de tempo de resposta da consulta. Entretanto, o pré-processamento dos dados e a criação do índice dos prefixos de grande proporção faz com que o método utilize uma quantidade de memória muito superior, não sendo viável para utilização em grandes bases ou com indexação de itens extensos.

META

No trabalho apresentado por Deng et al. (2016) é exposto que os métodos existentes possuíam três limitações. Primeiro, eles não podiam atender ao requisito de alto desempenho para grandes conjuntos de dados. O tempo de consulta para bases de dados

grandes torna-se longo. Em segundo lugar, eles envolviam cálculos redundantes para computar os nós ativos. Terceiro, neles era bastante difícil definir um limite de edição apropriado, porque um grande limite de edição retorna muitos resultados, enquanto um baixo limite de edição leva a poucos ou mesmo nenhum resultado, podendo não tratar corretamente os erros de entrada. Por exemplo, a consulta *"parefurnailia"* que corresponde ao termo *"paraphernalia"* tem uma distância de edição de 5, que é muito grande para palavras curtas e erros comuns. Uma alternativa proposta é retornar resultados top-k, ou seja, os k itens que sejam mais semelhantes à consulta. No entanto, os métodos existentes não podem suportar eficientemente as consultas de complementação automática top-k tolerantes a erros. Isso ocorre porque o conjunto de nós ativos depende do limite de distância de edição e, quando o limite é alterado, eles precisam calcular os nós ativos a partir do zero.

Sendo assim, Deng et al. (2016) propõem o "META", que é uma estrutura baseada em comparação de cadeias de caracteres, que calcula as respostas com base em caracteres correspondentes entre os dados da consulta e os dados indexados. Para processar as respostas para as consultas é utilizada uma técnica incremental que responde eficientemente consultas *top-k*. O META pode eficientemente executar consultas baseadas em limite de edição e consultas *top-k*.

O cálculo de distância de edição é realizado por meio de uma técnica de programação dinâmica, denominada como distância de edição deduzida, utilizando um mecanismo que pode calcular a distância de edição de termos ou de prefixo com poucas comparações, respeitando as seguintes normas:

Distância de edição deduzida (ED):

$$\text{Para todo q e s, } ED(q,s) = \min_{m \in M(q,s)} m_{(|q|,|s|)} \tag{2.1}$$

Onde q é o termo da consulta, s é o termo indexado a ser comparado, $M(q,s)$ é a matriz formada pelos caracteres de q e s, e $m_{(|q|,|s|)}$ é a distância de edição do prefixo de q para o termo s.

Distância de edição de prefixo deduzida (PED):

$$\text{Para todo q e s, } PED(q,s) = \min_{m \in M(q,s)} m_{|q|} \tag{2.2}$$

Onde q é o termo da consulta, s é o termo indexado a ser comparado, $M(q,s)$ é a matriz formada pelos caracteres de q e s, e $m_{|q|}$ é a distância de edição deduzida do prefixo de q.

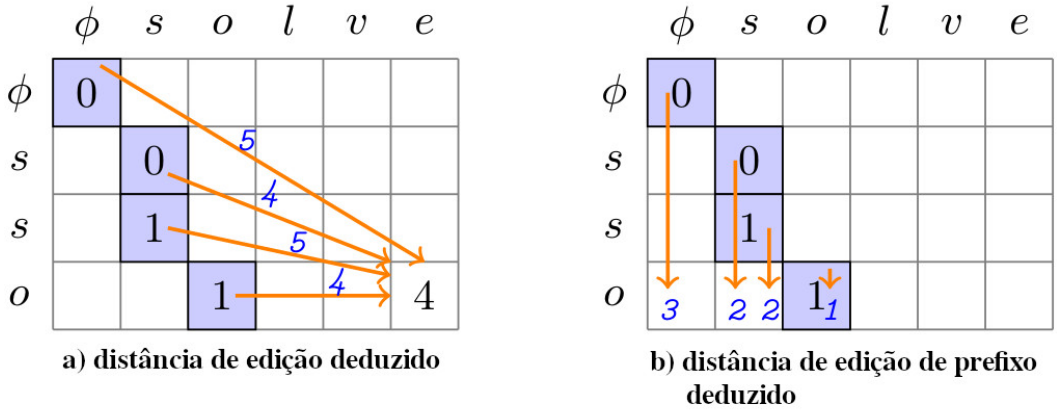

Figura 2.4: Matrizes de processamento de distância de edição para termos completos ou prefixos entre q = "sso" e s = "solve" Fonte: (Deng et al., 2016)

Um exemplo prático das matrizes geradas de q e s pode ser observado na Figura 2.4, onde o prefixo "sso" é comparado ao termo "solve". A matriz a ilustra o cálculo da distância de edição deduzida de "sso" para o termo completo "solve", já a matriz b ilustra o cálculo da distância de edição do prefixo deduzida de "sso" para o prefixo equivalente de "solve".

O trabalho apresenta também uma estrutura de árvore compacta para manter o relacionamento entre os nós ativos pai e filho, o que pode evitar os cálculos redundantes e garantir que cada nó da trie seja acessado no máximo uma vez. Foi proposto, além disso, um método eficiente para responder incrementalmente a uma consulta top-k, usando o número máximo de erros de edição entre a consulta e seus resultados.

BEVA

Zhou et al. (2016) apresenta um algoritmo de processamento de consultas eficiente para auto complementação tolerante a erros, alcançando o tamanho mínimo de nós ativos por meio da utilização do conjunto de nós ativos limite, e com a proposta do autômato EVA (autômato de vetores de edição) para uso incremental e manutenção dos nós ativos, além de possibilitar o retorno dos nós folha diretamente pelo autômato, evitando o uso do tradicional caminho ascendente-descendente (via nó pai e nó filho) nos resultados do algoritmo.

Os vetores de edição servem para manter as várias possibilidades de distâncias de edição entre uma dada consulta e uma *string*, com a atualização do vetor de edição de um nó em um espaço de tempo de *O(1)*. Assim, o autômato de vetores de edição mantêm eficientemente apenas o conjunto de prefixos ativos limite, que são usados para percorrer a *trie* durante o processo no de complementação automática.

Zhou et al. (2016) define duas condições para garantir que as respostas de um *framework* de auto complementação de consultas sejam corretas: completude (equação 2.3) e solidez (equação 2.4).

$$\forall d \in R_Q, \wp(d) \cap P_Q \neq \emptyset \tag{2.3}$$

$$\forall d \notin R_Q, \wp(d) \cap P_Q \neq \emptyset \tag{2.4}$$

Onde R_Q é o conjunto de palavras que satisfazem a consulta tolerante a erros, $\wp(d)$ o conjunto de todos os prefixos da coleção de palavras, P_Q todos os prefixos da consulta Q. Além das duas condições, Zhou et al. (2016) adiciona mais uma condição, para permitir a validade dos resultados para dado τ, como na equação 2.5:

$$\forall \rho \in P_Q, ed(\rho, Q) \leq \tau \tag{2.5}$$

Sendo assim, define que $\mathcal{V}_Q$ é o conjunto de prefixos (nós) ativos, sendo este o conjunto de nós ativos que são mantidos nas etapas de manutenção de nós (Chaudhuri

and Kaushik, 2009)(Ji et al., 2009) ou subconjuntos (Li et al., 2011) para uma distância de edição menor ou igual a um dado τ, como na equação 2.5.

Dado $\mathcal{V}_Q$, que contém dois prefixos p e p' onde $p' \preceq p$. Ao remover todos os prefixos repetidos, pode-se obter o conjunto de prefixos ativos limite, ou seja, para encontrar o conjunto resposta de prefixos ativos mínimo, Zhou et al. (2016) define como o *conjunto de prefixos ativos limite* B_Q, como ilustrado na Figura 2.5.

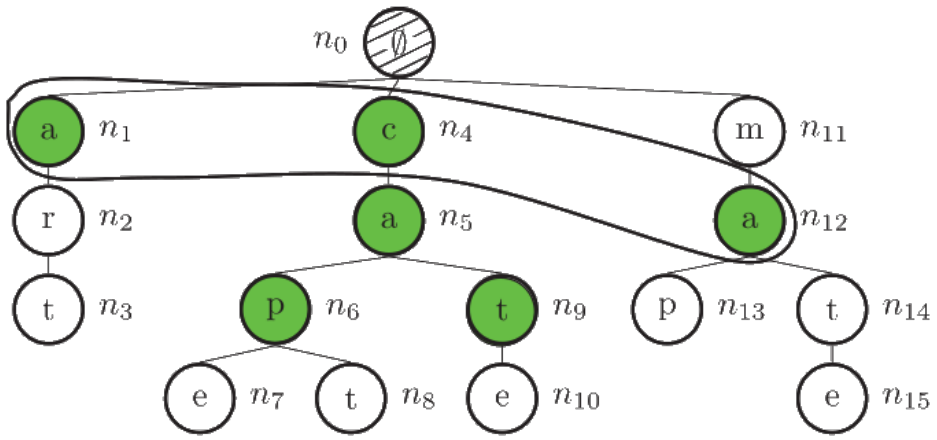

Figura 2.5: Considere o conjunto de prefixos ativos $\{n_1, n_4, n_5, n_6, n_9, n_{12}\}$, o conjunto de prefixos ativos limite é $\{n_1, n_4, n_{12}\}$. Fonte: (Zhou et al., 2016)

O ponto que devemos observar é que o tamanho do conjunto de nós (prefixos) mantidos pelo algoritmo impactam diretamente em sua eficiência de tempo e espaço, uma vez que um conjunto menor de nós reduz consequentemente o montante de tráfego (cliente-servidor) em cada etapa, seja na manutenção dos nós ou mesmo no retorno dos resultados.

O problema principal identificado nos algoritmos existentes é a manutenção de nós ativos, já que as soluções existentes permitem inerentemente relações ascendente-descendente entre os nós ativos, por motivo dessa redundância facilitar o acesso às informações de distância de edição dos nós descendentes.

A ideia para resolver essa dificuldade é manter para cada nó todas suas distâncias de edição das diagonais entre $-\tau$ e τ, dessa forma, garantindo que seus resultados sejam computados corretamente. Nenhum valor maior que τ é salvo, sendo assim, os valores dessas células são substituídos pelo símbolo especial # para gerar o vetor de edição.

O vetor de edição da coluna 0 sempre tem a forma $[\underbrace{\tau, \tau-1, \ldots, 1}_{\tau}, 0, \underbrace{1, 2, \ldots, \tau}_{\tau}]$, pois a *string* na coluna 0 está vazia. Essa característica é denominada como **vetor de edição inicial**. Da mesma forma, o vetor com todos os símbolos #, que é, $[\underbrace{\#, \#, \ldots, \#}_{2\tau+1}]$ é denominado como **vetor de edição final**. Como pode ser observado na Tabela 2.1.

Tabela 2.1: Vetores de edição, representados na comparação das *strings* "*abcd*" e "*xad*" (Zhou et al., 2016).

		0	1	3	4
		1	**x**	**a**	**d**
0		0	1		
1	**a**	1	1	1	
2	**b**		#	#	#
3	**c**			#	#
4	**d**				#
		V_0	V_1	V_2	V_3

A função de transição é definida como $= f(v_j, \mathcal{B})$, onde $\mathcal{B}$ é um *bitmap* binário de $2\tau + 1$ *bits* e $\mathcal{B} = \neg\delta(d[j+1], Q[j-\tau+i])$, para $\forall 1 \leq i \leq 2\tau+1$. Suponha que os vetores de edição da Tabela 2.1 para calcular a função de transição. $\neg\delta(x, \epsilon) = \mathbf{0}$, $\neg\delta(x, a) = \mathbf{0}$, $\neg\delta(x, b) = \mathbf{0}$, o *bitmap* $\mathcal{B} = \mathbf{000}$. Então $f([1, 0, 1], 000) = [1, 1, \#]$.

O vetor de edição pode então ser codificado como um estado em um autômato de vetor de edição (Figura 2.1.1). O cálculo do autômato de vetor de edição (EVA) é realizado nas seguintes etapas:

1. Adiciona o estado $\mathcal{S}_0$ associado com o vetor de edição inicial em uma fila vazia.

2. Enquanto a fila está vazia, o estado $\mathcal{S}_i$ da fila é removido. Computa-se $\mathcal{S}' = f(\mathcal{S}_i, \mathcal{B})$ para todo $2^{2\tau+1}$ possível valor de $\mathcal{B}$. Registra-se essas transições no autômato. Por fim, se $\mathcal{S}'$ é um novo estado, é adicionado na fila.

Para cada estado removido da fila, $2^{2\tau+1}$ transições são computadas. Estes novos conceitos de vetores de edição e autômato de vetores de edição permitem que os algoritmos mantenham a quantidade mínima de informações e melhorem substancialmente a eficiência no processamento de consultas. O estudo de Zhou et al. (2016) também compara o EVA favoravelmente com os autômatos determinísticos universais de *Le-*

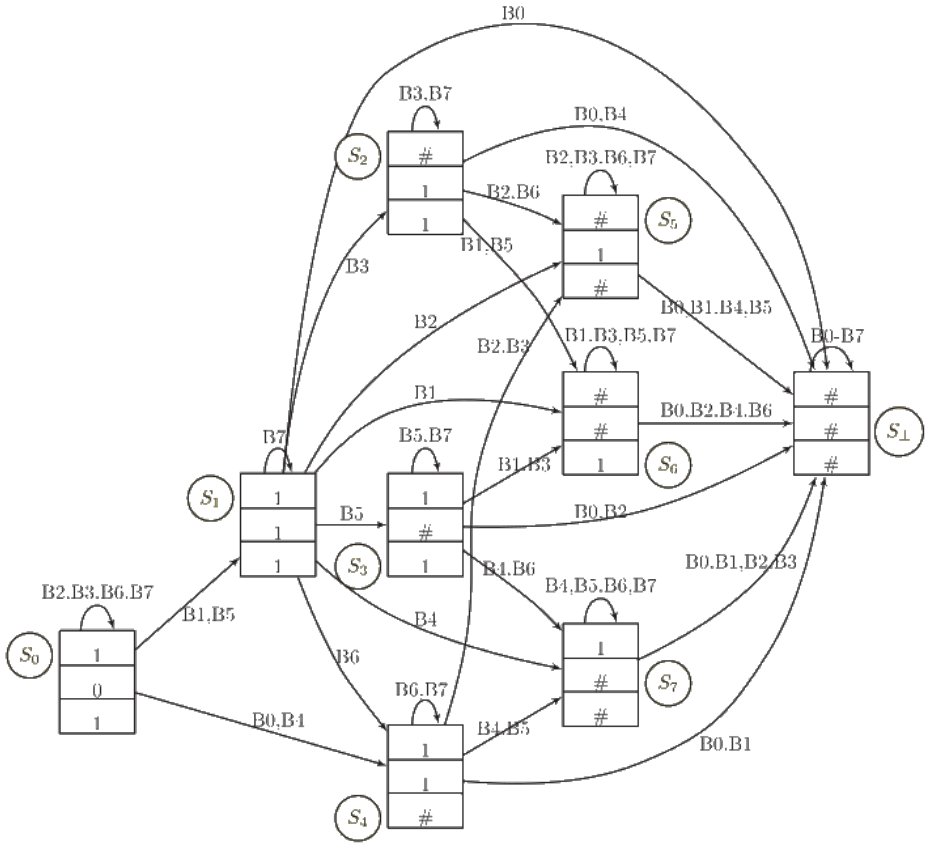

Figura 2.6: Representação de um autômato de vetor de edição

venshtein de última geração Mihov and Schulz (2004) em vários aspectos importantes, podendo também ser aplicados em outros contextos.

Complementação automática com sinônimos

Até este ponto, as soluções existentes para o problema de complementação automática de sentenças forneciam as sugestões baseadas no conjunto de caracteres de entrada iniciais da palavra desejada. No entanto, em muitas aplicações reais, uma entidade ou termo possui sinônimos ou abreviaturas. Sendo assim, Xu and Lu (2017) propõem um novo tipo de complementação automática utilizando sinônimos e abreviações. Foram apresentados três algoritmos baseados em *trie* para complementação automática eficiente de consultas *top-k* levando em consideração sinônimos e abreviações:

- Tries gêmeas (*Twin tries*): duas árvores *trie* são construídas, uma para representar as cadeias de caracteres e outra para representar as regras de sinônimos. Cada árvore é uma estrutura compactada, onde os filhos de cada nó são ordena-

dos pela maior pontuação entre seus respectivos descendentes. Para pesquisas de ambas as tentativas, um algoritmo *top-k* é executado para encontrar as regras de sinônimo.

- *Trie* expandida (*Expansion trie*): integra as regras de sinônimos e as cadeias de caracteres correspondentes na mesma árvore *trie*, utilizando regras para representação de sinônimos e de termos. A *trie* expandida fornece os resultados de consulta *top-k* em um tempo inferior que as *tries* gêmeas. Em contrapartida, a *trie* expandida costuma ocupar mais espaço de memória, pois é necessário que se expanda as cadeias de caracteres e suas regras aplicáveis.

- *Tries* hibridas (*Hybrid tries*): é a estrutura otimizada para encontrar o equilíbrio entre a *Trie* expandida e as *Tries* gêmeas. Partes das regras de sinônimos são cuidadosamente selecionadas para expandir as cadeias de caracteres, encontrando assim, um equilíbrio entre o tempo de pesquisa e o espaço utilizado.

A avaliação em larga escala das consultas realizada no estudo demonstra a eficácia das técnicas propostas, suportando a recuperação de consultas de sinônimos e abreviações de forma eficiente.

O foco do método é o processamento de consultas levando em consideração sinônimos e abreviações, não diretamente com o objetivo focado em eficiência de tempo de consulta ou espaço de memória utilizado. Sendo assim, este trabalho não utilizará esse método nos experimentos de comparação.

2.2 Busca em dois níveis

Sistemas de recuperação de informação têm o objetivo de contribuir para que seus usuários consigam encontrar informações relevantes de acordo com sua necessidade e interesse. Sendo assim, os sistemas de recuperação devem buscar a maior eficácia possível, maximizando a proporção entre a satisfação e o esforço do usuário (Baeza-Yates and Ribeiro-Neto, 2013). Tratando-se de busca em recuperação de informação,

várias abordagens são utilizadas para realização dos objetivos de acordo com contexto dos problemas. Este trabalho utiliza o conceito de busca em dois níveis.

A busca em dois níveis consiste na combinação de duas abordagens de processamento de consultas com objetivo de melhorar a eficiência, alcançando uma eficiência que não pode ser alcançada utilizando apenas uma das abordagens. Essa eficiência a ser melhorada pode ser em relação a tempo de consulta, qualidade dos resultados ou memória usada na indexação.

Em Manber et al. (1994), é apresentado o "GLIMPSE" (*GLobal IMPlicit Search*) ou "Busca Implícita Global", uma ferramenta que utiliza o conceito de busca em dois níveis para indexação e consultas de arquivos. O GLIMPSE oferece a possibilidade de indexação e consultas para sistemas de arquivos com baixa indexação, indexando apenas entre 2% e 4% do tamanho do texto.

A ideia da busca em dois níveis aplicada no GLIMPSE é um híbrido entre índices invertidos completos e busca sequencial sem indexação. Essa abordagem baseia-se na observação de que a busca sequencial é rápida o suficiente para textos de vários *megabytes* de tamanho, mostrando não ser necessário indexar todas as palavras do documento com suas localizações exatas. Na busca em dois níveis, o índice não fornece os locais exatos das palavras, mas apenas uma indicação da região onde a resposta possa ser encontrada. O uso de concordância para ambos os níveis, primeiro pesquisando o índice e depois pesquisando os arquivos reais, oferece grande flexibilidade, permitindo correspondências aproximadas e expressões regulares.

Apesar de economizar bastante espaço de indexação, a velocidade de busca mostrou-se ser inferior aos modelos que utilizam apenas indexadores baseados em lista invertida.

Outro trabalho que utilizou essa abordagem foi o sistema de busca proposto por Navarro et al. (2000), que utiliza indexação invertida de arquivos de texto compactados em blocos, endereçando em índice apenas os blocos. Os textos são divididos em blocos de tamanho fixo, onde cada bloco pode abranger vários documentos. O índice armazena apenas os blocos em que cada palavra aparece. Como o número de blocos é muito menor que o número de documentos, o espaço ocupado pelo índice é muito pequeno. Ao

realizar uma consulta, a estrutura de índices filtra os blocos onde constam os termos da consulta, então se dá a segunda parte do processamento, que é realizar pesquisa sequencial nos documentos dos blocos para definir a resposta. Ou seja, o índice é usado apenas como um dispositivo para filtrar alguns blocos da coleção que não podem conter uma correspondência com a consulta.

O sistema de Navarro et al. (2000) é capaz de reduzir a utilização de memória para menos de 40% do tamanho do texto original, obtendo também resultados de tempo de consulta satisfatórios, sendo até sete vezes mais rápido que o GLIMPSE.

Dado o cenário atual para o problema de complementação automática de sentenças, este trabalho visa propor uma nova técnica de solução para o problema através do processamento em dois níveis. Primeiramente, para processamento da primeira parte da sentença, os termos serão indexados em uma árvore *trie*. Em seguida, o processamento da consulta utilizará modelos clássicos de busca, como a busca sequencial.

Acredita-se que, ao indexar menos dados na *trie*, gera-se menos nós e, por consequência, menos nós ativos no processamento das consultas, economizando espaço de memória e melhorando a eficiência do processamento das consultas. Quando a indexação parcial da base de dados não é suficiente para determinar o resultado da complementação automática, é utilizado o processamento do segundo nível, que consiste na busca sequencial entre os conjunto de itens resultantes do processamento no primeiro nível que são relevantes para o prefixo de consulta.

2.3 Similaridade de cadeias de caracteres

Um sistema de recuperação de informação baseado em texto necessita processar as diferenças entre os termos. A forma trivial é comparar caractere por caractere, verificando a similaridade completa ou não entre dois termos. Essa abordagem mostra-se custosa computacionalmente, visto que, para comparar duas cadeias que possuem N caracteres, é necessário realizar até N^2 comparações de caracteres. Sendo assim, otimizar estes processamentos torna-se importante para que o custo computacional seja reduzido.

Há várias soluções para o cálculo de similaridade de cadeias de caracteres na li-

teratura, que abordam algoritmos, técnicas e métricas úteis para diversos contextos e aplicações (Gomaa and Fahmy, 2013) (Yu et al., 2016). Há soluções baseadas em termos, como a similaridade de Jaccard[3] e distância euclidiana[4]. Há também soluções baseadas em caracteres, como as métricas para cálculo de distância de edição baseadas em maior subsequencia comum[5], em n-grama e a métrica utilizada neste trabalho, definida como Damerau-Levenshtein ou "distância de Levenshtein", que define a distância entre duas cadeias de caracteres contando o número mínimo de operações necessárias para transformar uma cadeia de caractere na outra (distância de edição), onde uma operação é definida como uma inserção, exclusão ou substituição de um único caractere ou uma transposição de dois caracteres adjacentes (Levenshtein, 1966) (Hall and Dowling, 1980) (Peterson, 1980).

[3]A similaridade de Jaccard gera um coeficiente de similaridade dado pelo número de termos compartilhados em relação ao número de todos os termos exclusivos em ambas as sequências (Gomaa and Fahmy, 2013) (Yu et al., 2016).

[4]A distância euclidiana entre duas cadeias de caractere, denominada também como distância L2, é a raiz quadrada da soma das diferenças quadradas entre os elementos correspondentes de dois vetores formados pelas cadeias.

[5]A métrica de distância de edição baseada na maior subsequência comum considera que a similaridade entre duas cadeias de caracteres é baseada no comprimento da maior cadeia contígua de caracteres que existe em ambas as cadeias (Gomaa and Fahmy, 2013).

Capítulo 3

Método de complementação automática de sentenças em dois níveis

Este trabalho propõe um novo método para complementação automática de sentenças que combina busca sequencial com busca utilizando uma *trie* no intuito de realizar a tarefa de complementação de forma eficiente e, ao mesmo tempo, utilizando quantidade de memória reduzida quando comparado a outros métodos propostos na literatura.

3.1 Tecnologias utilizadas

A solução proposta consiste em indexar na *trie* apenas um prefixo de cada cadeia de caracteres presente no índice, criando uma estrutura que indica, para cada prefixo indexado, quais os elementos que o complementam na base de dados.

Na Figura 3.1 pode-se observar a *trie*, onde são indexadas partes iniciais dos itens (prefixos), cada nó que contém o caractere que representa o final desses prefixos possui referência à lista de itens que possuem esse prefixo em sua descrição.

O processamento de consultas no primeiro nível pode utilizar qualquer método proposto para busca em *tries*, em nossos experimentos optamos por utilizar um processamento baseado no método ICAN, proposto por Li et al. (2011), que utiliza pro-

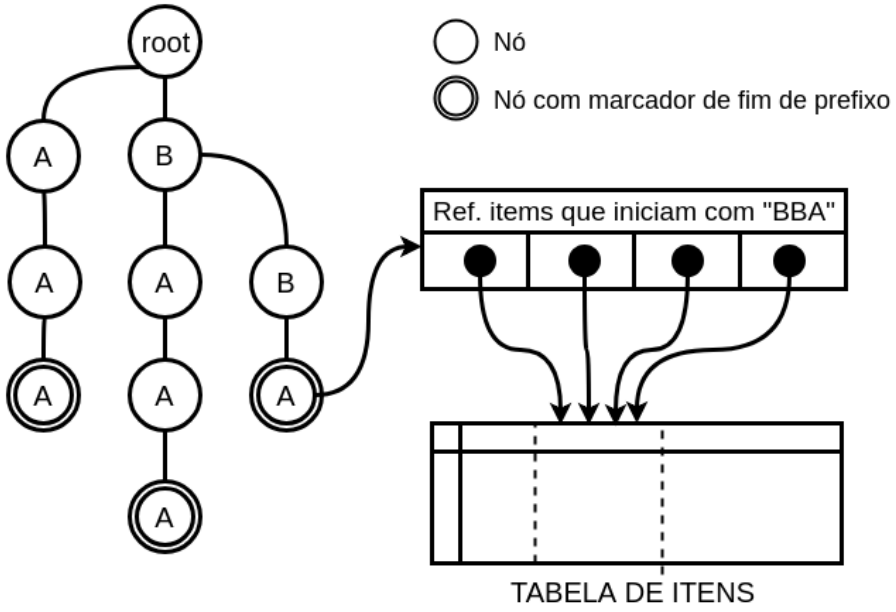

Figura 3.1: Arquitetura em dois níveis proposta para complementação automática de sentenças.

cessamento baseado em nós ativos, calculando a subárvore, de acordo com o limite de distância de edição dado. Este método é descrito na seção 2.1.1.

Para o segundo nível é realizada uma busca sequencial nos dados obtidos por meio da estrutura que armazena as referências dos itens, comparando o prefixo de busca com os resultados preliminares referenciados pelo primeiro nível. Os dados são ordenados de acordo com a similaridade dessa cadeia de caractere com o prefixo inserido pelo usuário.

O fluxo de consulta que o método utiliza pode ser observado na Figura 3.2. Dado o comprimento do índice no primeiro nível t e dada uma consulta q, para cada consulta um prefixo de comprimento t é gerado com os t primeiros caracteres. Esse novo prefixo é processado pela *trie*, gerando um conjunto de resultados. Caso o comprimento da consulta seja menor ou igual à t, os resultados gerados no primeiro nível são considerados os resultados finais.

Quando o comprimento da consulta for maior que t, é realizada uma busca sequencial no conjunto de resultados do primeiro nível, comparando a consulta completa com cada item, utilizando um algoritmo de cálculo de distância de edição. Os resultados finais são todos os itens que possuem distância de edição para a consulta menor ou

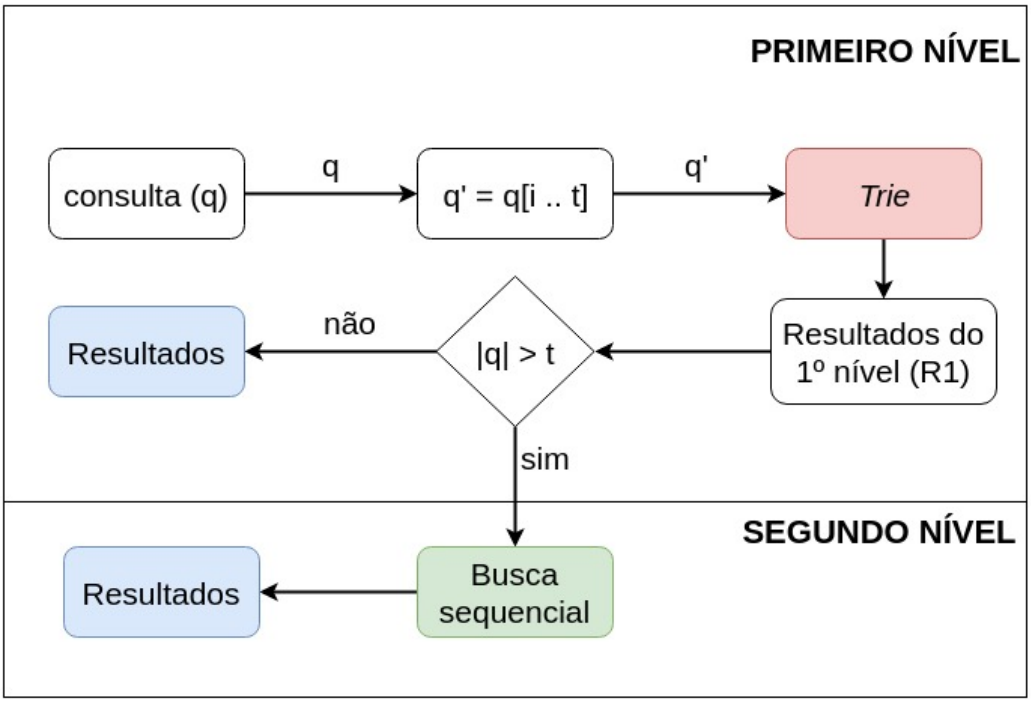

Figura 3.2: Fluxo de consulta realizado pelo método de dois nível. Onde q é a consulta e t é o tamanho do índice no primeiro nível

igual ao limite de distância de edição definido.

3.2 Implementação

O método de complementação automática de sentenças em dois níveis processa de forma eficiente a consulta de prefixos, realizando busca em uma *trie* apenas no primeiro nível, que é definido apenas com uma parte inicial da cadeia de caractere total a ser indexada. A possibilidade de indexar apenas parte da descrição dos itens causa um melhor aproveitamento de memória e, por possuir um conjunto de nós menor, causa também uma melhor eficiência em tempo de processamento da consulta na *trie*.

No exemplo usado nesta seção, a indexação do primeiro nível leva em consideração o prefixo com limite de tamanho de seis caracteres, itens menores ou iguais que seis caracteres são indexados completamente. Dado o conjunto de itens dispostos na Tabela 3.1, os prefixos correspondentes aos seis primeiros caracteres da descrição de cada um dos itens são inseridos na *trie*, onde ao fim de cada prefixo contém a referência para a estrutura que armazena as referências dos itens, como observado na Figura 3.3.

Tabela 3.1: Estrutura que armazena os itens

ITENS
United Kingdom
United States
United Arab Emirates
Saudi Arabia
Australia
Suriname
Uruguay
Argentina
Syria
Austria

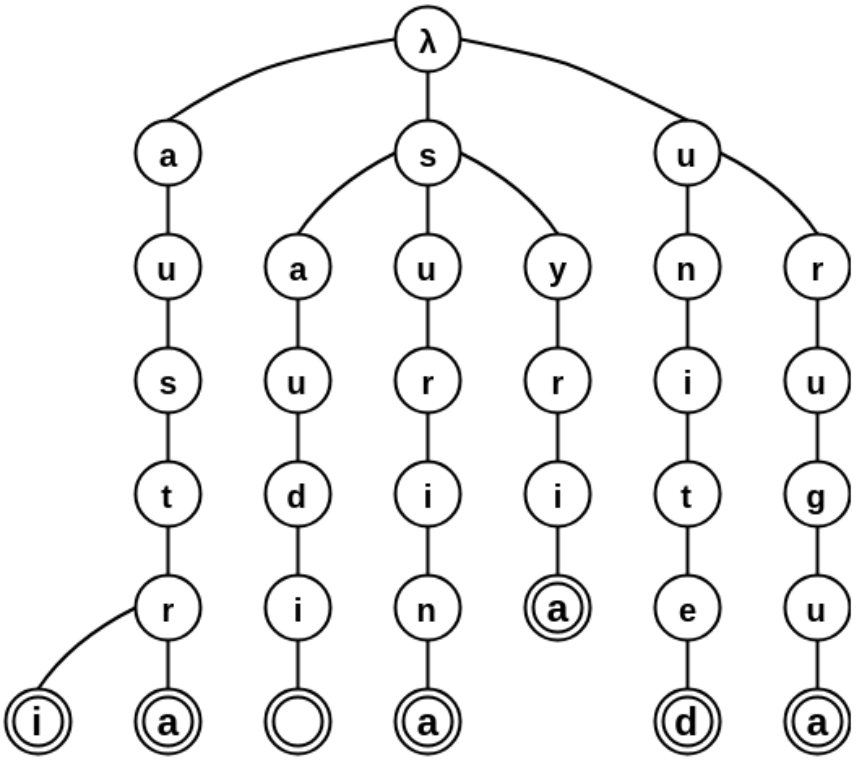

Figura 3.3: Disposição da indexação dos caracteres dos itens da Tabela 3.1, são indexados, no máximo, apenas os seis primeiros caracteres.

Os marcadores dos nós que indicam o fim de um item podem acessar as referências dos itens que possuem esse prefixo em sua descrição. Sendo assim, por exemplo, para o prefixo "united" indexado na *trie*, teremos uma lista de referências para os itens disposta na Tabela 3.2.

Dessa forma, ao digitar o prefixo "united", três itens seriam retornados. A partir do momento que usuário digita o sétimo caractere, o método utiliza apenas os resultados

Tabela 3.2: Estrutura de referências

ITENS
United Kingdom
United States
United Arab Emirates

dessa estrutura de referências para continuar a consulta de forma sequencial. Todos os itens que iniciam com o prefixo "united" em sua descrição estão disponíveis através dessa estrutura de referências.

O processo de busca sequencial realizado no segundo nível é baseado no cálculo de semelhança entre o prefixo de consulta e a descrição de cada item retornado no primeiro nível. O cálculo de similaridade é realizado através de uma técnica de comparação baseada em caracteres. A métrica de distância de edição entre cadeias de caracteres utilizada aqui, denominada como *Damerau–Levenshtein* (Levenshtein, 1966) (Hall and Dowling, 1980) (Peterson, 1980), define de forma eficiente a diferença entre o prefixo de consulta e a descrição dos itens.

O segundo nível desse método define os resultados para complementação da consulta respeitando o limite da distância de edição definido, levando em consideração o tamanho do prefixo digitado. Ao calcular a distância de edição entre o prefixo da consulta e os itens indexados, deve-se considerar apenas o prefixo das descrições de tamanho semelhante ao tamanho do prefixo de consulta, como pode ser observado no exemplo a seguir, o prefixo de consulta "uru" e o termo "uruguay" possuem distância de edição 4, entretanto, "uru" é um prefixo inicial de "uruguay" e para um sistema de sugestões de consultas é necessário mostrar as possibilidades de termos que o prefixo de consulta pode completar. Sendo assim, é necessário que o prefixo "uru" seja comparado com o prefixo inicial do termo "uruguay" que possui tamanho igual ao do prefixo de consulta, que no caso é "uru". Tendo o prefixo da consulta "uru" e o prefixo inicial do item "uru", a distância de edição calculada é igual a zero. Logo, para uma consulta com limite de distância de edição zero, "uruguay" é indicado como complemento de "uru".

Sendo assim, antes de comparar o prefixo de consulta com a descrição do item, utiliza-se o prefixo inicial da descrição do item, com o mesmo tamanho do prefixo de

consulta. Finalmente, calcula-se a distância de edição entre os prefixos. Ao realizar o cálculo, durante o processamento de cada caractere, é verificado se a distância de edição dos termos ultrapassa o limite definido. Caso o limite seja ultrapassado, o processo é finalizado e o algoritmo passa a comparar o prefixo de consulta com o próximo item da lista de resultados do primeiro nível. Ao término do cálculo da distância de edição entre os prefixos, o item é inserido no conjunto de resultados.

Algoritmo 1 Processo utilizado no segundo nível

```
 1: procedure SECONDLEVEL
 2:     query ← query prefix
 3:     limitED ← edit distance limit
 4:     resultsList ← list of results //starts empty
 5: for all results from first level:
 6:     if length(result) > length(query) then
 7:         prefixResult ← result.substring(0, length(query))
 8:     ed ← processEditDistance(query, prefixResult)  //Damerau-Levenshtein
 9:     if ed <= limitED then
10:         resultsList ← result
11: return resultList:
```

O processo realizado no segundo nível está ilustrado no algoritmo 1. Dado o prefixo de consulta (*linha 2*), o valor de limite de edição (*linha 3*) e a lista de resultados do primeiro nível (*linha 4*), para cada item da lista de resultados do primeiro nível (*linha 5*) é verificado se o tamanho da cadeia de caractere referente à descrição do resultado é maior que o tamanho do prefixo de consulta (*linha 6*), caso a verificação seja verdadeira, é gerado um prefixo da descrição do resultado do mesmo tamanho do prefixo da consulta (*linha 7*). Em seguida, é calculada a distância de edição entre o prefixo da consulta e o prefixo do resultado (*linha 8*) e, finalmente, é verificado se o valor da distância de edição calculado está de acordo com o limite de distância de edição definido para que o item seja inserido na lista de resultados (*linhas 9 e 10*).

O cálculo da distância de edição, na linha 8 do Algoritmo 1, é realizado através de um algoritmo de programação dinâmica, onde duas cadeias de caracteres são comparadas. O algoritmo utilizado calcula o mínimo de edições necessárias para que uma cadeia de caractere se torne igual à outra. O valor de distância de edição deve ser menor ou igual ao limite definido.

3.2.1 Tamanho do prefixo de indexação no primeiro nível

A eficiência da solução em dois níveis está relacionada com a quantidade de informações que são indexadas na estrutura de dados *trie*. Quanto maior a quantidade de informações, mais memória será utilizada na indexação e mais tempo será necessário para processar as consultas, pois haverá mais nós a serem processados no primeiro nível.

A peça central deste método é indexar na *trie* apenas um prefixo inicial referente à cadeia de caractere completa a ser indexada, sendo assim, é necessário definir qual tamanho do prefixo permite o melhor aproveitamento da relação entre espaço de memória utilizada, eficiência do processamento no primeiro nível e eficiência do processamento do segundo nível.

Capítulo 4

Experimentos

Neste capítulo serão apresentados os experimentos realizados com o objetivo de analisar o tamanho do prefixo de indexação na *trie* adequado para primeiro nível e, ainda, avaliar a performance em tempo de consulta e utilização de memória do método dois níveis através de comparações com os principais métodos dispostos na literatura.

Os principais objetivos são verificar a viabilidade em termos de tempo de computar complementos de sentenças utilizando uma estrutura em dois níveis proposta e, ainda, verificar qual o ganho que se pode obter em termos de redução de utilização de espaço para a realização da tarefa.

4.1 Configurações dos experimentos

Os experimentos utilizaram três bases de dados diferentes:

- **USADDR**[1], que é um conjunto com cerca de 10 milhões de endereços e lugares dos Estados Unidos, extraídos da coleção *SimpleGeo CC0*. Dessa base de dados foram extraídos os nomes dos lugares e inseridos em um arquivo de texto único, onde cada linha equivale a um item.

- **AOL**[2]: outra base de dados utilizada foi o histórico de *logs* de consultas da plataforma AOL, que possui cerca de 140 mil consultas únicas e datadas entre

[1] http://archive.org/download/2011-08-SimpleGeo-CC0-Public-Spaces/
[2] http://www.ccc.ipt.pt/~ricardo/experiments/AOL_DS.html

março e maio de 2006. Para os experimentos foi gerado um arquivo de texto onde cada linha representa um item da base de dados.

- **MEDLINE**[3]: é o principal banco de dados bibliográficos da Biblioteca Nacional de Medicina (NLM) dos Estados Unidos, que contém mais de 25 milhões de referências a artigos de revistas científicas em ciências da saúde, com ênfase em biomedicina. O MEDLINE é a versão *online* do MEDLARS (Sistema de Análise e Recuperação de Literatura Médica), originado em 1964. Extraiu-se o título de cada artigo citado, removendo duplicações. Cada título extraído equivale a um item.

A Tabela 4.1 apresenta informações sobre as bases de dados. Vale observar o tamanho médio dos itens, quanto maior o comprimento das cadeias de caracteres que formam esses itens, maior será a altura das árvores de indexação dos *baselines* e, para o método de dois níveis, maior será o comprimento das cadeias de caracteres a serem processadas no segundo nível.

Tabela 4.1: Informações das bases de dados

Dataset	Itens	Palavras	Média de palavras por item	Tam. médio itens	Tam. médio palavras
USADDR	10.251.121	32.407.449	3,161	19,757	6,249
AOL	143.591	690.698	4,810	30,080	6,253
MEDLINE	26.959.852	319.295.098	11,843	88,741	7,492

Os experimentos foram executados em uma máquina utilizando o processador Intel Xeon E5-4617 2.90 GHz e 64 GB de RAM, rodando Ubuntu 18.04.1 LTS. Todos os métodos implementados em C++.

4.2 Avaliando o tamanho do prefixo de indexação no primeiro nível

Foram realizados experimentos para analisar e definir o tamanho ideal do prefixo indexado no primeiro nível para o contexto de complementação automática de sentenças.

[3] https://www.nlm.nih.gov/databases/download/pubmed_medline.html

São analisadas as performances para prefixos de tamanho igual a quatro até o tamanho igual a dez, valores definidos levando em consideração o objetivo de economizar memória utilizando pouca indexação na *trie* e mantendo, ao mesmo tempo, performance satisfatória em tempo de processamento de consulta. Indexação menor que quatro caracteres faz com que a busca no primeiro nível faça um filtro de resultados pouco relevante, fazendo com que o processamento seja quase todo realizado no segundo nível e, ainda, quanto maior o tamanho da indexação, menor é a ocorrência de utilização do segundo nível e como esse experimento visa avaliar tanto o primeiro quanto o segundo nível, os tamanhos de indexação serão avaliados até o tamanho dez. Foram utilizados tamanhos de consultas iguais a três, cinco, sete, nove, onze e treze, apenas com tamanhos ímpares para simplificar a visualização dos resultados. Foram comparados o tempo de processamento de consultas no primeiro e no segundo nível, a quantidade de dados filtrados no primeiro nível a serem processados no segundo nível e o espaço de memória utilizado.

4.2.1 Tempo de processamento de consultas

Foram realizadas três avaliações para o experimento de tempo de processamento de consultas, a primeira compara a média de tempo realizado no processamento de consultas no primeiro e no segundo nível, a segunda avaliação compara a média de tempo de processamento geral variando o tamanho do prefixo de consulta e a terceira avaliação compara o tempo de processamento de consultas variando o limite da distância de edição. Foram utilizados limites de distância de edição 1, 2 e 3. Experimentos realizados em Oliveira (2018) indicam que o tamanho médio do comprimento de prefixos de consultas para o usuário escolher uma sugestão ou executar a consulta é de 6 a 8 cadeias de caracteres, sendo assim, processar prefixos de consultas para limites de edição acima de 3 geram muitos resultados irrelevantes.

Cada execução dos experimentos utilizou mil consultas, as quais foram geradas extraindo-se aleatoriamente mil itens da base de dados, seguindo Chaudhuri and Kaushik (2009). Foram gerados seis arquivos de consultas, o primeiro utilizando apenas

os primeiros três caracteres dos itens, o segundo utilizando os cinco primeiros, o terceiro utilizando os sete primeiros e o quarto, quinto e sexto arquivos utilizando os nove, onze e treze caracteres iniciais, respectivamente. Para cada consulta dos arquivos, um caractere foi modificado para simular um erro de digitação.

Tabela 4.2: Tempo médio (em milissegundos) de processamento total variando comprimento do prefixo de consulta para cada tamanho do prefixo de indexação na *trie*(de quatro a dez caracteres), para limite de distância de edição 1.

	Comprimento do prefixo de consulta					
	|q| = 3	|q| = 5	|q| = 7	|q| = 9	|q| = 11	|q| = 13
4 caracteres	0,512	10,387	10,684	11,082	11,863	12,713
5 caracteres	0,518	0,311	4,599	4,628	4,900	5,288
6 caracteres	0,530	0,307	2,969	3,077	3,174	3,383
7 caracteres	0,542	0,317	0,343	1,782	1,808	1,895
8 caracteres	0,539	0,313	0,309	1,532	1,586	1,622
9 caracteres	0,550	0,334	0,308	0,302	0,743	0,738
10 caracteres	0,555	0,364	0,334	0,302	0,659	0,666

Tabela 4.3: Tempo médio (em milissegundos) de processamento para cada nível do processo variando comprimento do prefixo de consulta para cada tamanho do prefixo de indexação na *trie*(de quatro a dez caracteres), para limite de distância de edição 1.

	Comprimento do prefixo de consulta											
	|q| = 3		|q| = 5		|q| = 7		|q| = 9		|q| = 11		|q| = 13	
nível:	1º	2º	1º	2º	1º	2º	1º	2º	1º	2º	1º	2º
4 caracteres	0,52	-	0,38	9,61	0,39	9,85	0,40	10,24	0,38	11,08	0,38	12,1
5 caracteres	0,52	-	0,29	-	0,35	4,08	0,36	4,18	0,36	4,56	0,37	4,89
6 caracteres	0,53	-	0,29	-	0,35	2,59	0,35	2,63	0,35	2,79	0,36	2,98
7 caracteres	0,54	-	0,30	-	0,29	-	0,34	1,49	0,33	1,49	0,35	1,61
8 caracteres	0,54	-	0,32	-	0,29	-	0,34	1,19	0,35	1,23	0,35	1,27
9 caracteres	0,55	-	0,34	-	0,32	-	0,29	-	0,32	0,39	0,31	0,40
10 caracteres	0,56	-	0,34	-	0,31	-	0,30	-	0,32	0,33	0,35	0,35

Pode-se observar na Tabela 4.2 e na ilustração da Figura 4.1 que a média de tempo de processamento das consultas aumenta consideravelmente entre as variações de tamanho de indexação da *trie* no primeiro nível, quanto menor for o tamanho de indexação no primeiro nível, maior é o tempo necessário para processar as consultas. Outra observação é sobre a diferença da média de tempo entre as versões do tamanho de indexação, que diminui quanto maior for o tamanho de indexação da *trie*, por exemplo, a diferença média de tempo de consulta entre uma *trie* com indexação de 4 caracteres e uma *trie* com indexação de 5 caracteres é de aproximadamente 7,5 milissegundos, já a diferença

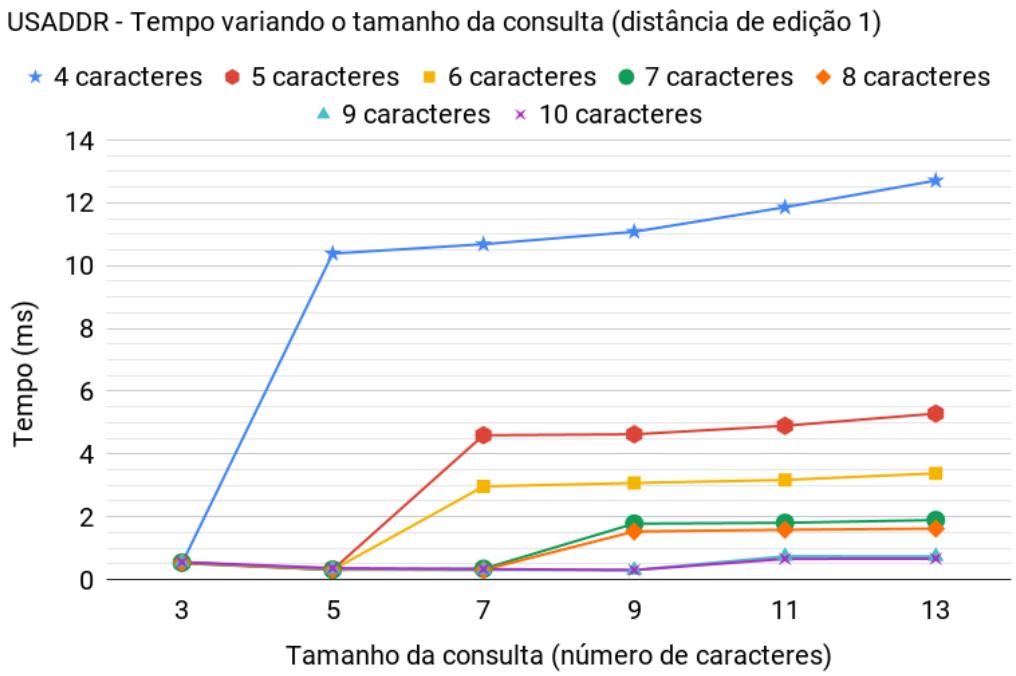

Figura 4.1: Comparação do tempo médio de processamento total de consultas variando o comprimento do prefixo de consulta para cada tamanho do prefixo de indexação na *trie* (de quatro a dez caracteres). Para limite de distância de edição igual a um.

entre a *trie* com indexação de 5 caracteres e a de 6 caracteres é de aproximadamente 1,6 milissegundos.

Para cada tamanho de prefixo de indexação do primeiro nível, o tempo de processamento varia consideravelmente quando ocorre a necessidade da utilização do processamento no segundo nível. A Tabela 4.3 e a Figura 4.2 mostram a razão entre os tempos utilizados no primeiro e no segundo nível para o processamento das consultas. Pode-se observar através desses dados que o tempo do processo no primeiro nível possui variação insignificante em proporção à variação observada no tempo de execução do processo no segundo nível.

Um maior número de nós na estrutura utilizada no primeiro nível resulta em um aumento no processamento de nós ativos na medida que o limite de distância de edição é incrementado. Comparando as versões do método de dois níveis no tamanho do prefixo de indexação no primeiro nível, as Tabelas 4.4 e 4.5 e a Figura 4.3 mostram a comparação dos tempos de processamento obtidos realizados com o limite de distância de edição igual a dois. A Tabelas 4.6 e 4.7 e a Figura 4.4 mostram a comparação dos

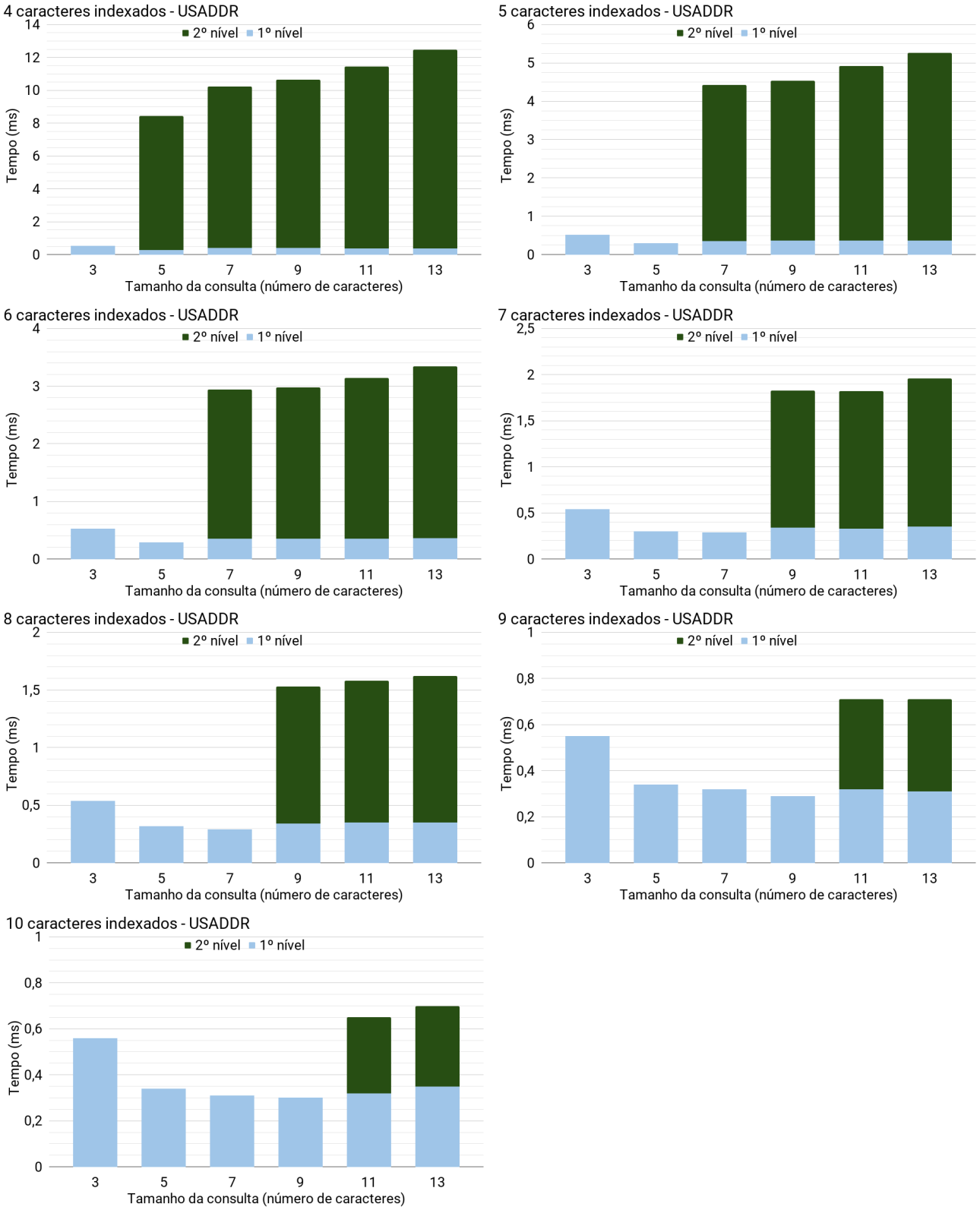

Figura 4.2: Tempo médio de processamento de consultas, variando o comprimento do prefixo de consulta para cada versão de indexação do prefixo no primeiro nível, de tamanho quatro a dez caracteres, para limite de distância de edição 1. São comparados o tempo de processamento do primeiro e segundo nível.

tempos de processamento obtidos realizados com o limite de distância de edição igual a três.

Tabela 4.4: Comparação do tempo médio (em milissegundos) de processamento total de consultas para o limite de distância de edição 2, para cada tamanho do prefixo de indexação na trie(de quatro a dez caracteres). Para consultas de tamanho igual a cinco, sete, nove, onze e treze

	Comprimento do prefixo de consulta				
	q = |5|	q = |7|	q = |9|	q = |11|	q = |13|
4 caracteres	96,651	102,065	119,089	131,133	157,372
5 caracteres	9,648	33,429	38,871	43,308	45,580
6 caracteres	10,461	17,523	23,261	23,617	24,361
7 caracteres	10,608	10,303	16,146	19,090	19,232
8 caracteres	10,937	10,719	15,305	15,103	15,125
9 caracteres	11,284	11,014	11,696	14,556	14,727
10 caracteres	11,491	11,101	11,639	14,010	14,108

Tabela 4.5: Tempo médio (em milissegundos) de processamento para cada nível do processo variando comprimento do prefixo de consulta para cada tamanho do prefixo de indexação na *trie*(de quatro a dez caracteres), para limite de distância de edição 2.

	Comprimento do prefixo de consulta									
	q = |5|		q = |7|		q = |9|		q = |11|		q = |13|	
nível:	1°	2°	1°	2°	1°	2°	1°	2°	1°	2°
4 caracteres	9,17	87,48	9,10	92,96	9,22	109,87	9,16	121,97	9,30	148,07
5 caracteres	9,64	-	10,22	23,21	9,92	28,95	10,10	33,20	10,20	35,38
6 caracteres	10,46	-	10,25	7,27	10,07	13,19	10,08	13,53	10,18	14,18
7 caracteres	10,61	-	10,30	-	10,61	5,53	10,44	8,65	10,52	8,71
8 caracteres	10,94	-	10,72	-	10,92	4,38	10,59	4,64	10,12	5,00
9 caracteres	11,28	-	11,01	-	11,70	-	12,02	2,53	11,88	2,84
10 caracteres	11,49	-	11,10	-	11,64	-	11,82	2,19	11,89	2,21

Assim como no primeiro nível, o tempo de processamento no segundo nível também aumenta. Isso acontece pois a quantidade de dados retornados no primeiro nível aumenta consideravelmente ao aumentar o limite de distância de edição. As Figuras 4.6 e 4.7 mostram o aumento do percentual de itens referentes à base de dados que o segundo nível deve processar. Comparando com os valores da Figura 4.5, percebe-se um aumento no número de itens processados no segundo nível maior ainda entre limites de edição 1 e 2. A *trie* com prefixos de tamanho 4 varia 1,58 pontos percentuais em relação à base de dados entre os resultados do primeiro nível entre os limites de edição 1 e 3, um aumento de mais de 3000%, já na *trie* com prefixos de tamanho 10 varia apenas

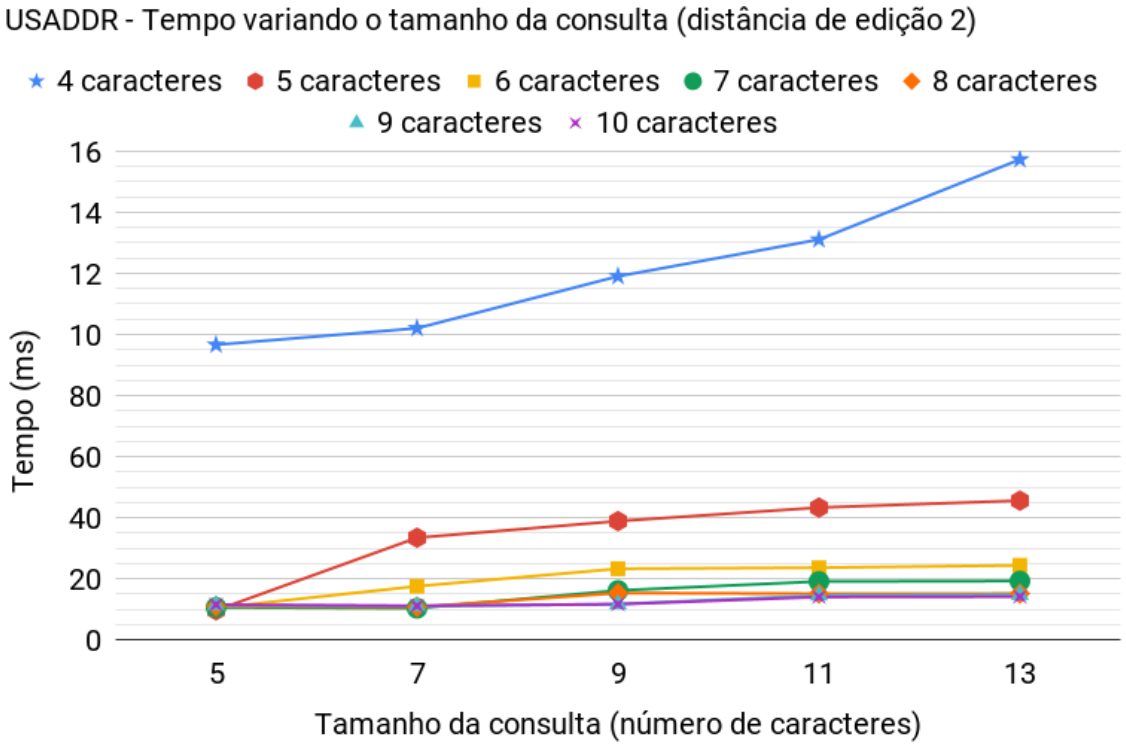

Figura 4.3: Comparação do tempo médio de processamento total de consultas para o limite de distância de edição 2, para cada tamanho do prefixo de indexação na trie(de quatro a dez caracteres). Para consultas de tamanho igual a cinco, sete, nove, onze e treze.

0,016 pontos percentuais, que é um aumento de 400%. Sendo assim, quanto maior o tamanho do prefixo de indexação na *trie*, menor será o impacto de processamento ao aumentar o limite de distância de edição.

Dessa forma, ao aumentar o limite de distância de edição, há um aumento na proporção do tempo de processamento do primeiro nível em relação ao segundo nível, como pode ser observado nas Tabelas 4.3, 4.5 e 4.7. Por exemplo, para limite de distância de edição 1, o tempo médio de processamento de prefixos com tamanho onze e indexação na *trie* de tamanho nove é de 0,32 ms e 0,39 ms, respectivamente para o primeiro e segundo nível. Já quando se processa levando em consideração distância de edição 2, o tempo médio do primeiro nível é de 11,57 ms e o do segundo nível é de 2,68 ms, tendo o primeiro nível sofrido um aumento bem maior. Esse aumento é maior ainda quando o limite de distância de edição é igual a 3, onde a média de tempo de processamento de prefixos com tamanho onze e indexação na *trie* de tamanho nove do

primeiro nível é de 195 ms, enquanto o segundo nível é executado em apenas 7,2 ms. Sendo assim, ao aumentar o limite de distância de edição, o aumento na quantidade de nós ativos a serem processados no primeiro nível faz com que o tempo gasto para processar as consultas no primeiro nível aumente significativamente. Em contrapartida, o aumento o tempo no segundo nível é consideravelmente menor, indicando menor impacto de performance do segundo nível do método referente ao aumento do limite de distância de edição.

Tabela 4.6: Comparação do tempo médio (em milissegundos) de processamento total de consultas para o limite de distância de edição 3, para cada tamanho do prefixo de indexação na trie(de quatro a dez caracteres). Para consultas de tamanho igual a cinco, sete, nove, onze e treze

	Comprimento do prefixo de consulta				
	q = |5|	q = |7|	q = |9|	q = |11|	q = |13|
4 caracteres	1323,811	1451,932	1626,48	1895,372	2229,295
5 caracteres	169,683	299,15	319,788	330,484	363,615
6 caracteres	190,613	212,765	216,529	218,249	219,855
7 caracteres	190,476	191,18	210,024	209,906	211,361
8 caracteres	186,357	193,446	204,424	204,815	206,234
9 caracteres	195,212	194,586	194,917	201,892	202,825
10 caracteres	196,047	194,132	195,231	200,318	200,801

Tabela 4.7: Tempo médio (em milissegundos) de processamento para cada nível do processo variando comprimento do prefixo de consulta para cada tamanho do prefixo de indexação na *trie*(de quatro a dez caracteres), para limite de distância de edição 3.

	Comprimento do prefixo de consulta									
	q = |5|		q = |7|		q = |9|		q = |11|		q = |13|	
nível:	1°	2°	1°	2°	1°	2°	1°	2°	1°	2°
4 caracteres	110,1	1213,7	106,0	1345,9	105,6	1520,8	106,8	1788,5	108,9	2120,4
5 caracteres	169,7	-	168,0	131,1	170,6	149,2	169,8	160,7	169,1	194,5
6 caracteres	190,6	-	190,0	22,8	192,5	24,0	192,9	25,3	191,9	27,9
7 caracteres	190,5	-	191,2	-	193,8	16,2	192,2	17,7	192,0	19,3
8 caracteres	186,4	-	193,4	-	194,0	10,4	193,3	11,5	192,9	13,3
9 caracteres	195,2	-	194,6	-	194,9	-	195,3	6,6	195,0	7,8
10 caracteres	196,0	-	194,1	-	195,2	-	194,8	5,6	195,9	4,9

Outra observação importante é que, na medida que o tamanho do prefixo de indexação no primeiro nível aumenta, o tempo de processamento do segundo nível diminui, isso ocorre pois o processamento de uma busca mais profunda na *trie* com um prefixo com mais caracteres gera resultados menores do que buscas com profundidades me-

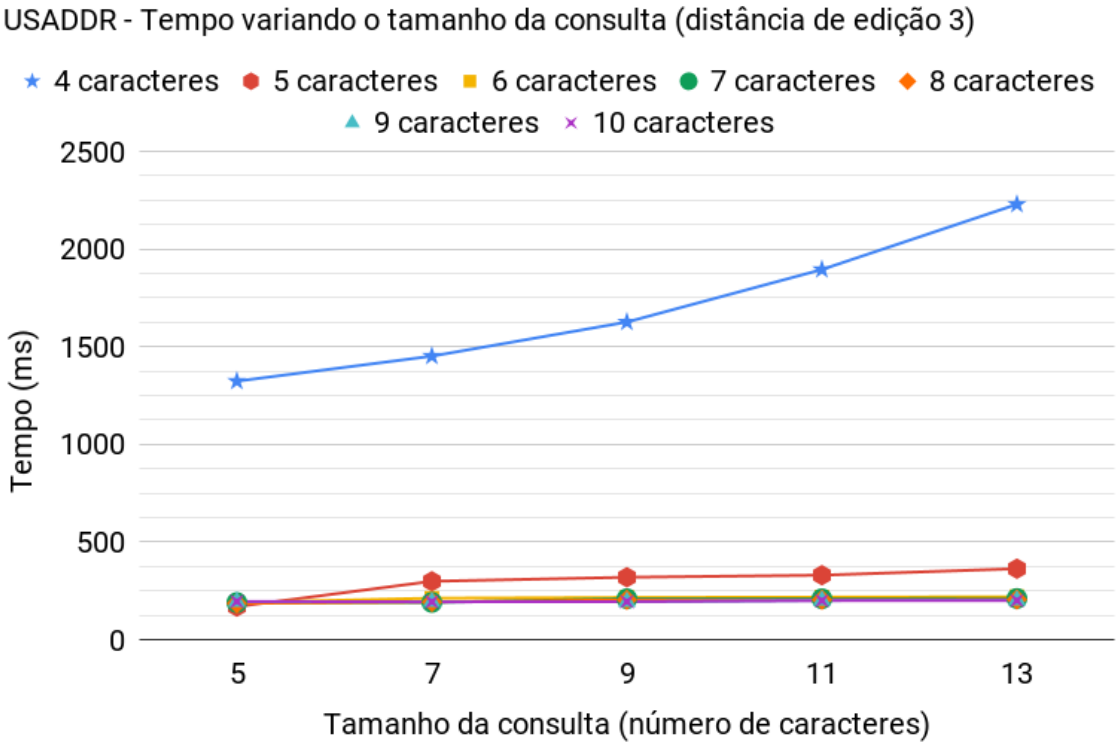

Figura 4.4: Comparação do tempo médio de processamento total de consultas para o limite de distância de edição 3, para cada tamanho do prefixo de indexação na trie(de quatro a dez caracteres). Para consultas de tamanho igual a cinco, sete, nove, onze e treze.

nores, a porcentagem da base de dados filtrada pode ser observada na Figura 4.5. O percentual médio da base de dados utilizado para realização da busca sequencial no segundo nível do prefixo indexado na *trie* de tamanho quatro (0,13%) é quase três vezes maior que o de tamanho cinco (0,05%), chegando a ser mais de trinta vezes maior que o percentual utilizado pela versão de prefixo indexado igual a dez (0,004%), ou seja, uma diferença de mais de quarenta mil itens para a base de dados USADDR.

Quanto maior o tamanho do prefixo indexado no primeiro nível, maior terá que ser o tamanho do prefixo de consulta para que o segundo nível seja utilizado. Sendo assim, quando o segundo nível for utilizado, quanto maior for o prefixo de indexação no primeiro nível, menos resultados serão filtrados, permitindo menos processamento realizado na busca sequencial, que possui um custo maior.

Pode-se notar também que, no processamento do segundo nível, a variância do tempo cresce significativamente, isso indica uma variação alta nos tempos obtidos para

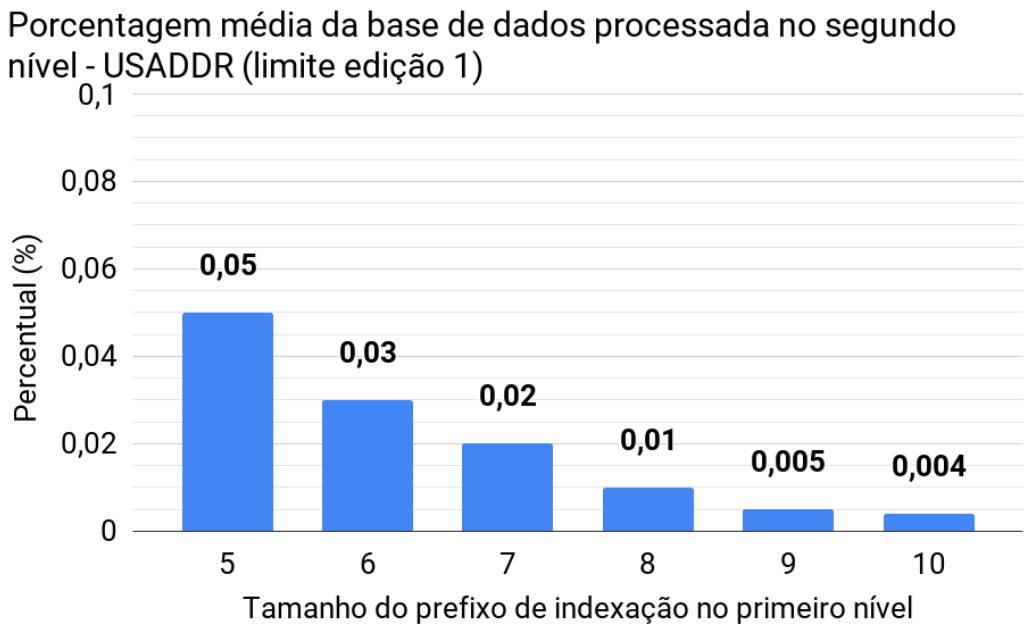

Figura 4.5: Porcentagem média de entradas da base de dados processadas no segundo nível ao variar o tamanho dos prefixos indexados na *trie* no primeiro nível, para limite de distância de edição 1.

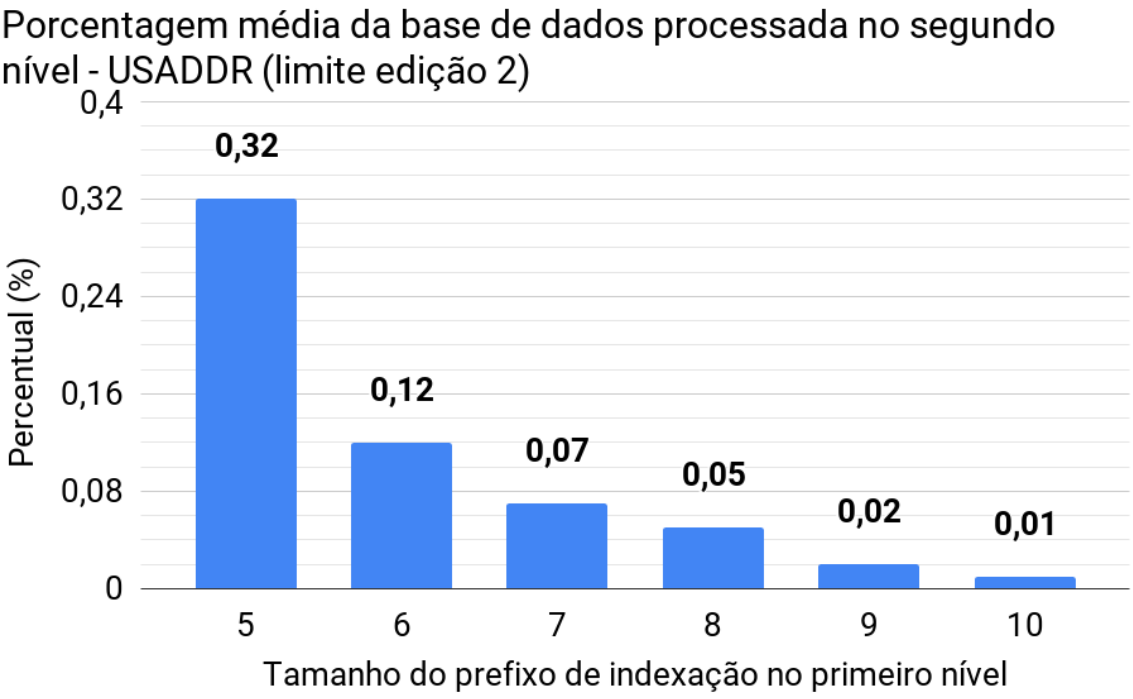

Figura 4.6: Porcentagem média de entradas da base de dados processadas no segundo nível ao variar o tamanho dos prefixos indexados na *trie* no primeiro nível, para limite de distância de edição 2.

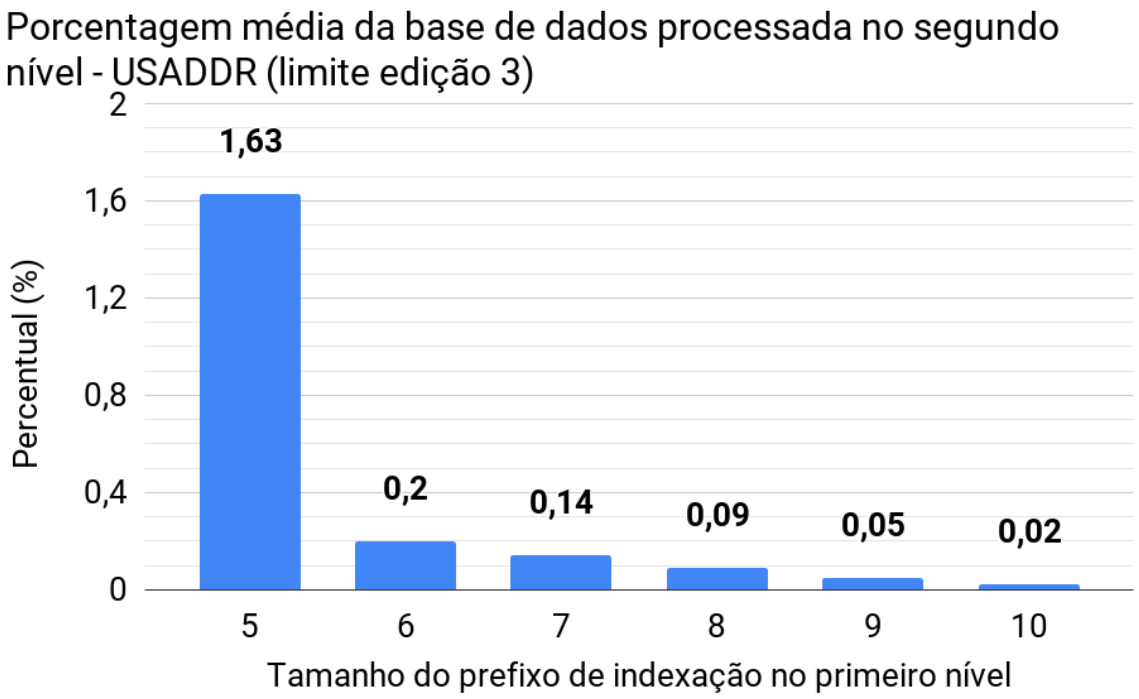

Figura 4.7: Porcentagem média de entradas da base de dados processadas no segundo nível ao variar o tamanho dos prefixos indexados na *trie* no primeiro nível, para limite de distância de edição 3.

cada prefixo. Tal fenômeno é causado pela variação muito grande na quantidade de resultados obtidos no primeiro nível. A Figura 4.8 apresenta o número de resultados no primeiro nível para cada uma das mil consultas de prefixos de tamanho onze realizadas em uma *trie* com indexação no primeiro nível de tamanho dez, para ilustrar a quantidade dos resultados analisados no segundo nível para cada consulta.

Para entender melhor a quantidade dos resultados analisados no segundo nível para cada consulta, é ilustrado na Figura 4.8 o gráfico de tendência do número de resultados no primeiro nível para cada uma das mil consultas de prefixos de tamanho onze realizadas em uma *trie* com indexação no primeiro nível de tamanho dez.

O gráfico da figura 4.8 mostra a quantidade de complementos filtrados no primeiro nível para um conjunto de mil consultas utilizadas nos experimentos. As consultas estão ordenadas pela quantidade de itens selecionados para serem analisados no segundo nível. Pode-se perceber que há poucas consultas onde há uma quantidade muito grande de itens, enquanto a maioria tem uma quantidade muito pequena. A quantidade de

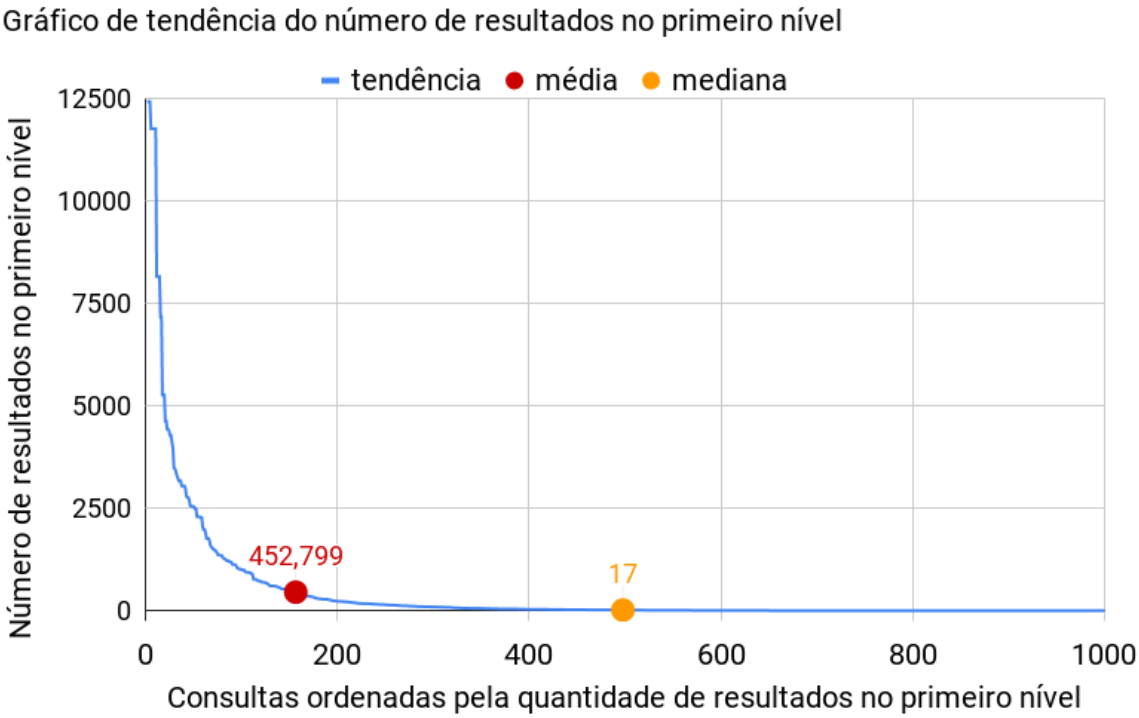

Figura 4.8: Gráfico de tendência para a quantidade de itens retornados no primeiro nível do método de dois níveis com indexação na *trie* de tamanho dez. Para mil consultas de tamanho onze utilizando a base de dados USADDR.

itens que vão para o segundo nível dentre as mil consultas mostradas varia de cerca de doze 12 mil itens até 1. O gráfico também serve para explicar a alta variância apresentada nos experimentos do método de dois níveis, que é decorrente de uma distribuição muito tendenciosa na distribuição dos valores. Uma medida bastante utilizada para saber-se o quão tendenciosa é uma distribuição é calcular a razão entre a média e a mediana dos valores. No gráfico, pode-se notar que há uma enorme discrepância entre a média, que é de 452,799 e a mediana, que é de 17, indicando que a distribuição é muito tendenciosa.

Entre as consultas realizadas, podemos analisar o tempo de processamento das que possuíram a maior quantidade de itens retornados no primeiro nível, que por consequência passaram a possuir os maiores tempos. A média das consultas de tamanho 11 para método com *trie* de tamanho 10 e limite de distância de edição 1 é de 0,659 ms, mas com uma distribuição de resultados do primeiro nível e tempo geral de processamento tendenciosa, podemos observar consultas com tempo de processamento que atingem 10

Tabela 4.8: Tempo (em milissegundos) de processamento para as cinco consultas de tamanho 11 com maiores tempos de processamento, para tamanho da *trie* no primeiro nível de 10 caracteres e limite de distância de edição 1.

	Tempo de consulta (ms)
consulta 258	10,849
consulta 515	10,555
consulta 507	10,482
consulta 665	10,463
consulta 905	10,446

ms, como pode ser observado na Tabela 4.8, que mostra as consultas com os cinco maiores tempos de processamento. Esses tempos são cerca de 1.400% maiores que a média das mil consultas, mostrando que a distribuição tendenciosa pode provocar, mesmo que em apenas alguns casos, um tempo de consulta bem alto. Esse comportamento pode ser, em alguns casos, categorizado como uma limitação do método de dois níveis.

4.2.2 Espaço de memória utilizado

Quanto ao espaço de memória utilizado pelo método de dois níveis, o número de nós da *trie* utilizada varia de acordo com tamanho dos prefixos indexados. Quanto maior o tamanho do prefixo indexado na estrutura, maior é o espaço de memória utilizado. Para ilustrar a variação da utilização de memória dado o tamanho da base de dados, foram geradas mais três versões da base de dados USADDR, extraindo-se itens aleatoriamente. Foram geradas porções de dados com 25%, 50% e 75% dos itens totais da base, a Figura 4.9 e a Tabela 4.9 mostram a variação de utilização de espaço entre os diferentes tamanhos de indexação no primeiro nível. Na Tabela 4.9 compara-se também o percentual de espaço utilizado em relação à indexação todos os caracteres dos itens.

Na Tabela 4.9 pode-se observar que o percentual de espaço utilizado em relação à indexação total não varia significativamente ao aumentar a quantidade de itens a serem indexados, entretanto nota-se que essa variação diminui na medida que o tamanho da *trie* aumenta. Por exemplo, primeiro nível com 4 caracteres varia positivamente 3,19 pontos percentuais entre 25% e 100% da base de dados, enquanto o primeiro nível com

Tabela 4.9: Quantidade de memória (em GB) utilizada para indexação do prefixo no primeiro nível e a porcentagem em relação à quantidade de memória utilizada ao indexar o item completo, variando o tamanho de 4 a 10. Para 25%, 50%, 75% e 100% da base dedados USADDR.

	Porcentagem da base de dados			
Tamanho do prefixo indexado no primeiro nível	25%	50%	75%	100%
Indexação completa	**2,828**	**5,165**	**7,176**	**9,366**
4 caracteres	0,503 (17,78%)	0,990 (19,16%)	1,394 (19,42%)	1,936 (20,67%)
5 caracteres	0,526 (18,59%)	1,021 (19,76%)	1,429 (19,91%)	1,977 (21,10%)
6 caracteres	0,566 (20,01%)	1,082 (20,94%)	1,506 (20,98%)	2,068 (22,07%)
7 caracteres	0,626 (22,12%)	1,180 (22,84%)	1,630 (22,71%)	2,219 (23,69%)
8 caracteres	0,708 (25,03%)	1,315 (25,45%)	1,806 (25,16%)	2,442 (26,07%)
9 caracteres	0,816 (28,85%)	1,497 (28,98%)	2,042 (28,45%)	2,738 (29,23%)
10 caracteres	0,945 (33,41%)	1,717 (33,12%)	2,338 (32,58%)	3,112 (33,22%)

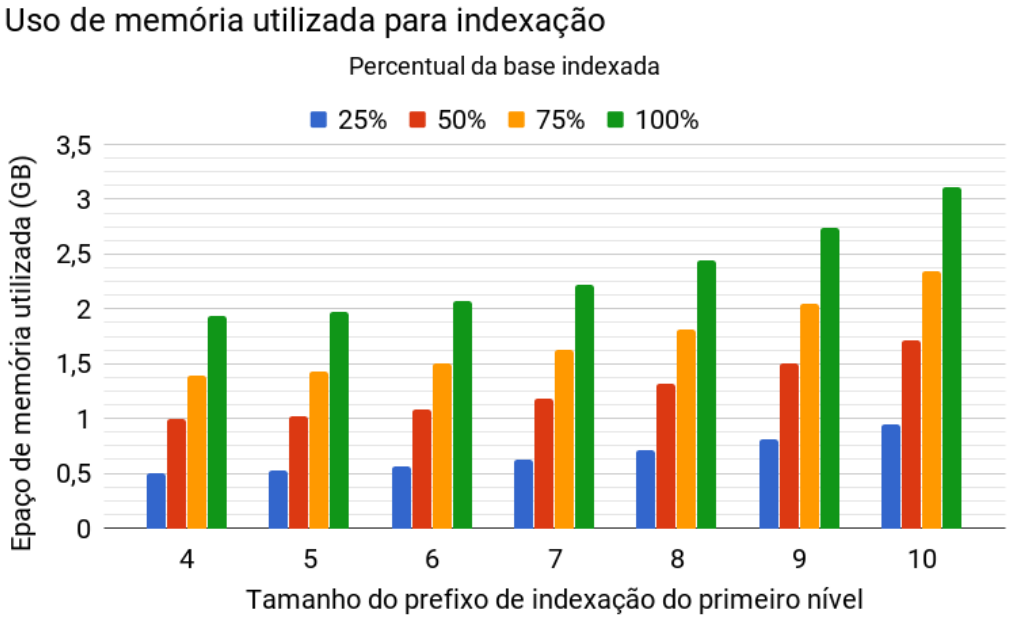

Figura 4.9: Ilustração comparativa da quantidade de memória em GB utilizada para indexação do prefixo no primeiro nível variando o tamanho de 4 a 10. Para 25%, 50%, 75% e 100% da base de dados USADDR.

7 caracteres possui uma variação positiva de 1,57 pontos percentuais e o primeiro nível com 10 caracteres varia negativamente 0,19 pontos percentuais. Ou seja, na medida que a quantidade de itens na base de dados aumenta, menor é a variação de uso de memória para *tries* maiores em relação à indexação completa.

Outra informação presente na Tabela 4.9 é a diferença entre o espaço ocupado pelas *tries* que indexam apenas prefixos de tamanho limitado no primeiro nível e o

espaço ocupado pela indexação completa da base de dados, quanto maior o tamanho dos prefixos no primeiro nível, maior será a diferença. É possível notar que a *trie* de 10 caracteres possui uma diferença de cerca de 4,5 pontos percentuais em relação à *trie* de 9 caracteres, chegando a usar cerca de 33% de espaço em relação ao espaço utilizado pela indexação completa da base de dados. Já a diferença média entre as *tries* de 5 e 4 caracteres é de cerca entre 0,5 pontos percentuais, com a indexação de 4 caracteres usando entre 17,78% e 20,67% do espaço ocupado pela indexação completa.

A variação entre as *tries* de 4 e 10 caracteres é de cerca de cerca de 26 pontos percentuais em 25% da base e de 13 pontos percentuais para a base completa, mostrando também que essa variação diminui na medida que a base de dados aumenta. Em valores de memória, a diferença é de 0,442 GB para 25% da base de dados, um aumento de cerca de 87%, enquanto para 100% da base a diferença é de 1,176 GB, uma diferença maior em valor absoluto, mas foi um aumento menor se levar em consideração a proporção desse espaço entre 4 e 10 caracteres, por volta de 60%.

Essa variação na diferença de espaço de memória, na medida que a base de dados aumenta, ocorre pois quanto mais itens são inseridos na *trie*, mais caminhos de nós são reaproveitados. Ou seja, quanto maior a quantidade de informação indexada, menos nós serão criados ao inserir um item novo.

Através dos dados obtidos nos experimentos, pode-se notar que a eficiência do processamento das consultas no primeiro nível é consideravelmente superior em relação ao processamento no segundo nível. Quanto a isso, o fator mais crítico para definir o custo de processamento no segundo nível é o tamanho do prefixo de indexação do primeiro nível, pois quanto maior for a *trie*, menos dados serão processados no segundo nível.

O tamanho do prefixo indexado no primeiro nível deve ser definido de acordo com a aplicação dessa solução, algumas informações de contexto podem ser levadas em consideração, como tamanho da base de dados e média de tamanho dos prefixos de consulta antes do usuário selecionar uma sugestão.

Caso a aplicação seja para um sistema que tenha um interesse maior em economizar

espaço de memória ao invés de otimizar o tempo de processamento, os experimentos mostram que a indexação de prefixos menores utiliza menos memória, entretanto é importante lembrar que para limites de distância de edição acima de dois, o tempo de consulta subiu muito acima do limite de 200 ms nos experimentos realizados, que é o tempo tolerado para uma boa experiência para o usuário. Tal parâmetro limite para o tempo deve sempre ser levado em conta na escolha do tamanho da partição entre os dois níveis.

Para sistemas que buscam, além de uma menor utilização de memória, uma boa eficiência em tempo de processamento de consultas, quanto maior o for tamanho do prefixo, melhor será o tempo de processamento, mas para um bom equilíbrio de eficiência de uso de memória e tempo de consulta, vale observar que quanto maior for o tamanho do prefixo indexado no primeiro nível, menor é a diferença de tempo de consulta entre essas versões de indexação. Os experimentos aqui realizados mostram que os prefixos indexados a partir de tamanho seis possuem tempos parecidos. Logo, para este contexto, é válido observar nos *logs* do sistema o tamanho médio dos prefixos de consulta antes do usuário selecionar uma sugestão, pois o ideal é que a definição do tamanho do prefixo indexado no primeiro nível faça com que o sistema retorne as sugestões relevantes antes que precise utilizar o processamento do segundo nível, onde o custo de tempo de processamento é maior. Sendo assim, para sistemas em que os usuários digitam em média oito caracteres para poder selecionar uma sugestão, o ideal é que o tamanho do prefixo indexado seja maior ou igual a oito.

A título de ilustração, o trabalho de Oliveira (2018) mostra que os tamanhos de prefixos de busca realizados pelos usuários antes de selecionar uma das sugestões tipicamente variam entre três e oito caracteres, para um conjunto de lojas de comércio eletrônico estudadas em seu trabalho, com a média de tamanho variando entre seis e oito caracteres dependendo do contexto do sistema (Oliveira, 2018).

Entre as versões do método de dois níveis utilizados nos experimentos, pode-se observar que ao aumentar a indexação no primeiro nível, ganha-se bastante eficiência entre indexações menores (4 ou 5 caracteres) mas essa proporção de ganho vai dimi-

nuindo, como na pequena diferença de média de tempo observada entre a indexação de 9 e 10 caracteres. A partir de um determinado valor, o ganho em tempo de processamento obtido ao aumentar o tamanho do prefixo de indexação no primeiro nível torna-se insignificante, enquanto a eficiência em compressão continua diminuindo de forma significativa. Como os valores de tempo entre a indexação no primeiro nível de tamanho 9 e 10 para a base de dados USADDR já possuem uma diferença baixa, para esse contexto não seria interessante aumentar mais ainda o tamanho da indexação, visto que não haveria ganhos significativos, aumentando apenas a quantidade de memória utilizada.

Sendo assim, para contexto desse experimento, nota-se que o ponto de equilíbrio entre performance em tempo de processamento e economia de espaço de memória encontra-se entre os tamanho de indexação do primeiro nível 8, 9 e 10. Vale lembrar que esses valores devem ser definidos de acordo com o contexto em que o método será utilizado. Visando simplificar os próximos experimentos e não poluir os dados com muita informação de versões do método de dois níveis, as versões do método usados nos experimentos utilizaram valores aqui definidos como tamanho da indexação do primeiro nível.

4.3 Comparação com os *baselines*

Para avaliar a hipótese de que é possível, utilizando um método de dois níveis, realizar complementação automática em tempo aceitável e consumindo uma quantidade significativamente menor de espaço do que outras soluções, o método de complementação automática de sentenças em dois níveis será avaliado e comparado com as principais soluções de complementação automática de sentenças existentes na literatura. O experimento avaliará a quantidade de memória utilizada para indexação das bases de dados e a média de tempo de processamento das consultas variando o tamanho da consulta e o limite de distância de edição.

As seguintes soluções foram comparadas nos experimentos:

- **ICAN** é um algoritmo para complementação automática tolerante a erros baseado em trie (Li et al., 2011).

- **ICPAN** é uma variação otimizada do ICAN, utilizando apenas os nós ativos essenciais para processamento (Li et al., 2011).

- **IncNGTrie** é um algoritmo para complementação automática tolerante a erros que utiliza intermediação de espaços para aumentar a eficiência (Xiao et al., 2013).

- **META** é uma estrutura que utiliza indexação em árvores compactas, sendo capaz de reduzir cálculos redundantes no processamento de complementação automática de sentenças. Nos experimentos será utilizado a versão com limite de distância de edição fixo (Deng et al., 2016).

- **BEVA** é um método de complementação automática baseado em automatos vetoriais. (Zhou et al., 2016)

- **2-level** é o método proposto neste trabalho, que consiste na combinação de processamento utilizando índices em trie e processamento sequencial. Serão usados três versões nos experimentos, variando o tamanho do prefixo e indexação no primeiro nível em 8, 9 e 10 caracteres, aqui denominados respectivamente como 2-level-8, 2-level-9 e 2-level-10.

Por motivos de limitação indicada no método IncNGTrie, que possui dificuldades de indexação quanto ao espaço para itens que possuam tamanho maior que 13 caracteres, o método será avaliado apenas nos experimentos utilizando a base de dados AOL. Durante os experimentos, não foi possível indexar as bases de dados USADDR e MEDLINE usando o IncNGTrie, excedendo o limite de 64 GB de memória que a máquina utilizada possui. Da mesma forma, o método BEVA demanda mais memória que o disponível para indexar a base de dados MEDLINE, não sendo possível realizar os testes.

4.3.1 Espaço de utilização de memória

Nesta seção serão apresentados os resultados referente à avaliação da quantidade de espaço que a indexação dos métodos utiliza. Em um sistema de complementação automática de sentenças, todos os dados necessários para processamento das consultas e correções de erros são indexados em memória, pois é necessário obter o máximo de eficiência em tempo de processamento. Sendo assim, é importante que as soluções não utilizem muito espaço de memória.

Para verificar a evolução em escalabilidade da indexação dos métodos, foram geradas porções das bases de dados com 25%, 50% e 75% através da seleção aleatória de itens. As informações das parciais geradas através da bases de dados USADDR estão dispostas na Tabela 4.10, assim como as parciais da base de dados AOL na Tabela 4.11 e da base de dados MEDLINE na 4.12.

Tabela 4.10: USADDR - informações das porções geradas da base de dados

Parcial	Itens	Palavras	Média de palavras por item	Tam. médio itens	Tam. médio palavras
25%	2.562.781	8.102.291	3,161	19,757	6,249
50%	5.125.561	16.205.708	3,161	19,760	6,249
75%	7.688.340	24.318.074	3,162	19,762	6,248
100%	10.251.121	32.407.449	3,161	19,757	6,249

Tabela 4.11: AOL - informações das porções geradas da base de dados

Parcial	Itens	Palavras	Média de palavras por item	Tam. médio itens	Tam. médio palavras
25%	35.898	172.524	4,805	30,015	6,246
50%	71.796	345.709	4,815	30,070	6,245
75%	107.694	520.746	4,835	30,305	6,267
100%	143.591	690.698	4,810	30,080	6,253

Tabela 4.12: MEDLINE - informações das porções geradas da base de dados

Parcial	Itens	Palavras	Média de palavras por item	Tam. médio itens	Tam. médio palavras
25%	6.739.963	79.829.677	11,844	88,747	7,492
50%	13.479.926	159.649.384	11,843	88,742	7,492
75%	20.219.889	239.261.660	11,832	88,649	7,491
100%	26.959.852	319.295.098	11,843	88,741	7,492

Os valores de utilização de memória dos métodos utilizando a base de dados AOl, dispostos na Tabela 4.13 e Figura 4.10, mostram um padrão de crescimento de uso de memória entre os métodos ICAN, ICPAN, META e as variações do 2-level, com uma média de crescimento que varia entre 45% e 52% para cada quarto da base de dados adicionada, tendo um aumento médio de 300% a 340% entre as porções da base de dados de 25% e 100%. O método BEVA obteve uma média de 29% de crescimento ao inserir cada porção da base de dados, variando 213% entre as porções de 25% e 100%. Já o IncNGTrie obteve os maiores valores, variando 67% em cada aumento de porção, totalizando um aumento de uso de memória em 456% entre as porções de 25% e 100%.

Tabela 4.13: AOL - valores de espaço de memória utilizado (em GB).

	Parcial da base de dados			
	25%	50%	75%	100%
IncNGTrie	0,215 GB	0,419 GB	0,619 GB	0,982 GB
META	0,078 GB	0,147 GB	0,202 GB	0,262 GB
ICAN	0,094 GB	0,177 GB	0,245 GB	0,318 GB
ICPAN	0,093 GB	0,176 GB	0,244 GB	0,317 GB
BEVA	0,334 GB	0,475 GB	0,588 GB	0,712 GB
2-level-8	**0,017 GB**	**0,029 GB**	**0,039 GB**	**0,051 GB**
2-level-9	0,019 GB	0,034 GB	0,044 GB	0,058 GB
2-level-10	0,022 GB	0,039 GB	0,051 GB	0,067 GB

Ao analisar a quantidade de memória utilizada, pode-se perceber de forma clara a diferença de variação do uso de memória entre os métodos comparados. Os métodos ICAN e ICPAN geraram um aumento médio de 0,07 GB de memória em cada transição de tamanhos parciais da base de dados AOL, tendo uma diferença média de 0,22 GB entre a porção de 25% e a porção de 100%. O META e o BEVA variam entre as porções em média 0,06 GB e 0,12 GB respectivamente, aumentando entre a menor e a maior porção cerca de 0,18 GB para o META e 0,37 GB para o BEVA. O IncNGTrie obteve um uso de memória consideravelmente superior em relação aos outros métodos, variando em uma média de 0,25 GB em cada transição de porção, tendo uma diferença total entre 25% e 100% de 0,76 GB, mais do que o dobro do segundo maior aumento, pertencente ao BEVA.O 2-level-8, por sua vez, obteve a menor variação de uso de memória, com uma média de crescimento de 0,01 GB pra cada 25% de dados indexados e com um

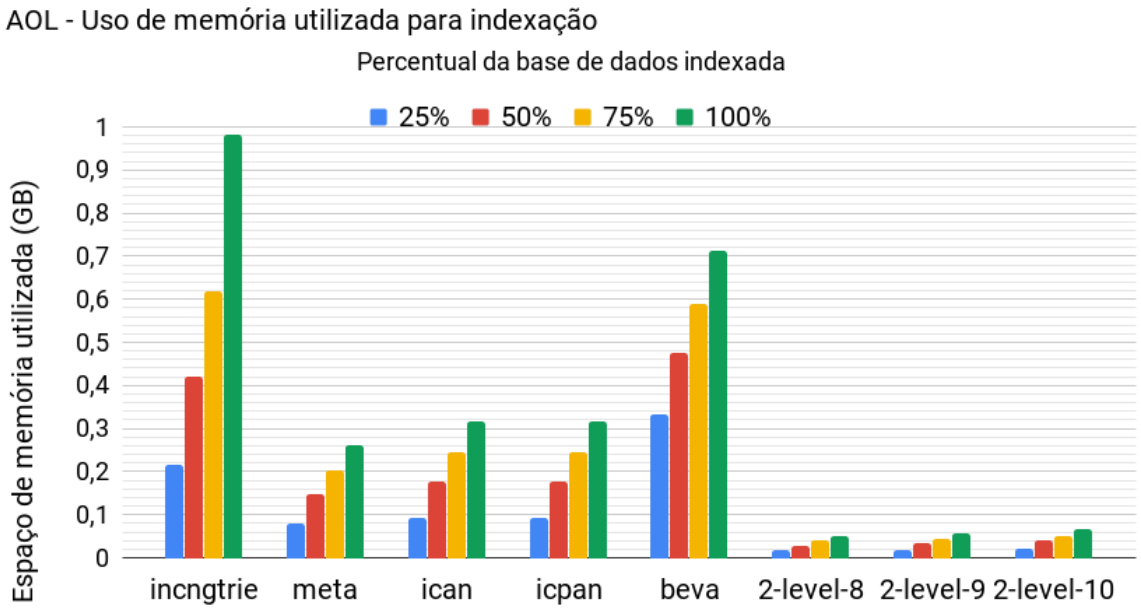

Figura 4.10: AOL - quantidade de memória em GB utilizada por cada método. Para 25%, 50%, 75% e 100% da base de dados.

aumento total em 0,03 GB entre a indexação de 25% e 100% da base de dados, uma casa decimal a menos do que a menor variação de memória entre os métodos não baseados no método de dois níveis, que foi o ICPAN. O 2-level-9 e 2-level-10 obtiveram uma diferença média entre as porções de cerca de 0,015 GB, com uma diferença entre 25% e 100% da base de dados de cerca de 0,04 GB.

Para a base de dados USADDR, os valores de utilização de memória dos métodos, dispostos na Tabela 4.14 e na Figura 4.11, mostram um padrão de crescimento de uso de memória entre os métodos mais uniforme, variando entre 45% e 52% para cada quarto da base de dados adicionada, tendo um aumento médio de 330% entre as porções da base de dados de 25% e 100%. Entretanto, os valores de espaço de memória utilizada mostram diferenças entre esses métodos.

Em bases de dados maiores, como é o caso da USADDR, os valores tendem a crescer e exibir com mais ênfase as diferenças de utilização de memória entre os métodos comparados. Os métodos ICAN e ICPAN geraram um aumento médio de 2,1 GB de memória em cada transição de de tamanhos parciais da base de dados USADDR, tendo uma diferença de 6,5 GB entre a porção de 25% e a porção de 100%. O META

e o BEVA variaram entre as porções em média 1,83 GB e 3,64 GB respectivamente, aumentando entre a menor e a maior porção cerca de 5,51 GB para o META e 10,94 GB para o BEVA. O 2-level-8 também obteve o menor aumento em quantidade de memória na base de dados USADDR, obtendo aumento médio de 0,57 GB para cada 25% de dados indexados, com um crescimento acumulado de 1,73 GB entre a indexação de 25% e 100% da base de dados, tendo seu valor de uso de memória em 100% da base de dados menor do que a quantidade de memória utilizada pelos outros métodos para indexar apenas 25% da base de dados. Apesar dos métodos 2-level-9 e 2-level-10 obterem quantidade de memória superior ao 2-level-8, a diferença não chega a ser significativa em relação aos demais métodos, tendo pouco mais de 2 GB de diferença entre as porções de 25% e 100% da USADDR.

Tabela 4.14: USADDR - valores de espaço de memória utilizado (em GB).

	Parcial da base de dados			
	25%	50%	75%	100%
META	2,337 GB	4,302 GB	6,061 GB	7,849 GB
ICAN	2,828 GB	5,165 GB	7,176 GB	9,366 GB
ICPAN	2,828 GB	5,165 GB	7,177 GB	9,369 GB
BEVA	5,399 GB	8,974 GB	12,569 GB	16,347 GB
2-level-8	**0,708 GB**	**1,315 GB**	**1,806 GB**	**2,442 GB**
2-level-9	0,816 GB	1,497 GB	2,042 GB	2,738 GB
2-level-10	0,945 GB	1,717 GB	2,338 GB	3,112 GB

O tamanho médio dos itens indexados mostra-se relevante para a quantidade de memória a ser utilizada na indexação das informações. Observa-se que a solução de geração de vizinhança para correção de erros do IncNGTrie, necessária para um processamento de consultas bem eficiente, gera uma necessidade de utilização de memória tão grande que inviabiliza o método para grandes bases de dados. Mesmo para a base AOL, bem menor, o método precisa utilizar mais de 900 MB de memória, acentuadamente superior aos demais métodos, sendo quase duas casas decimais maior que o conjunto de métodos 2-level.

Como pode ser observado nas Figuras 4.10 e 4.11, apesar dos métodos 2-level possuírem estrutura de indexação baseada no ICAN, não há a necessidade de indexar a descrição completa do item na *trie*, os métodos 2-level possibilitam uma melhor utiliza-

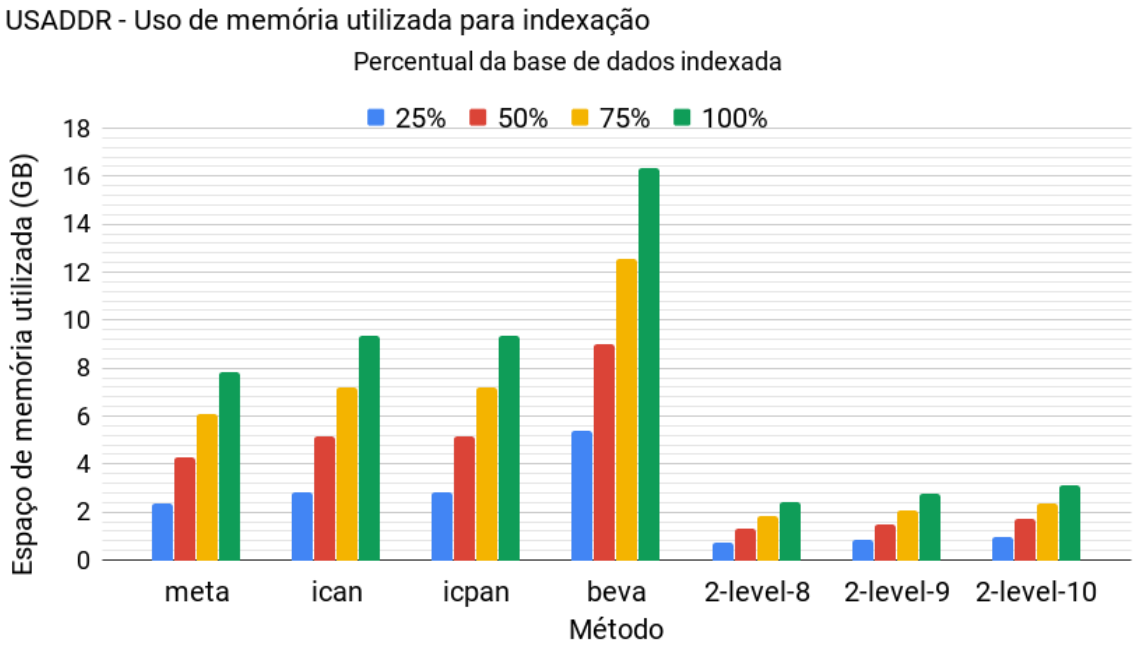

Figura 4.11: USADDR - quantidade de memória em GB utilizada por cada método. Para 25%, 50%, 75% e 100% da base de dados.

ção de espaço, permitindo uma economia de memória relevante em relação ao método utilizado no algoritmo do ICAN.

Ao utilizar uma base de dados de proporções maiores, tanto no número de itens a serem indexados quanto no comprimento das descrições desses itens, nota-se nos valores da Tabela 4.15 e da Figura 4.12 uma diferença maior entre os métodos 2-level e os outros métodos. Apenas o 2-level-8, 2-level-9 e 2-level-10 conseguiu indexar completamente a base de dados MEDLINE, os métodos META, ICAN e ICPAN foram capazes de indexar apenas a base de dados parcial equivalente a 25% da base MEDLINE, utilizando uma quantidade relativamente alta de memória em relação aos métdos 2-level. Não foi possível indexar nenhuma parcial da base de dados utilizando o BEVA. Para indexar a parcial de 25% da MEDLINE, o META utilizou 41,988 GB, o ICAN e ICPAN utilizaram 52,011 GB de memória, enquanto isso, o 2-level-8 utilizou 2,343 GB, o 2-level-9 indexou 2,411 GB e o 2-level-10 indexou 2,500 GB, cerca de vinte vezes menos memória que o META e vinte e cinco vezes menos que o ICAN. O 2-level-8, 2-level-9 e 2-level-10 também utilizaram, respectivamente, 9,035 GB, 9,202 GB e 9,423 GB de memória para 100% da base de dados MEDLINE.

Tabela 4.15: MEDLINE - valores de espaço de memória utilizado (em GB). Nota-se que apenas o 2-level conseguiu indexar a base de dados completa

	Porcentagem da base de dados			
	25%	50%	75%	100%
META	41,988 GB	≈ 70 GB	≈ 90 GB	≈ 100 GB
ICAN	52,011 GB	≈ 90 GB	≈ 130 GB	≈ 160 GB
ICPAN	52,011 GB	≈ 90 GB	≈ 130 GB	≈ 160 GB
2-level-8	**2,343 GB**	**4,585 GB**	**6,935 GB**	**9,035 GB**
2-level-9	2,411 GB	4,692 GB	7,072 GB	9,202 GB
2-level-10	2,500 GB	4,833 GB	7,254 GB	9,423 GB

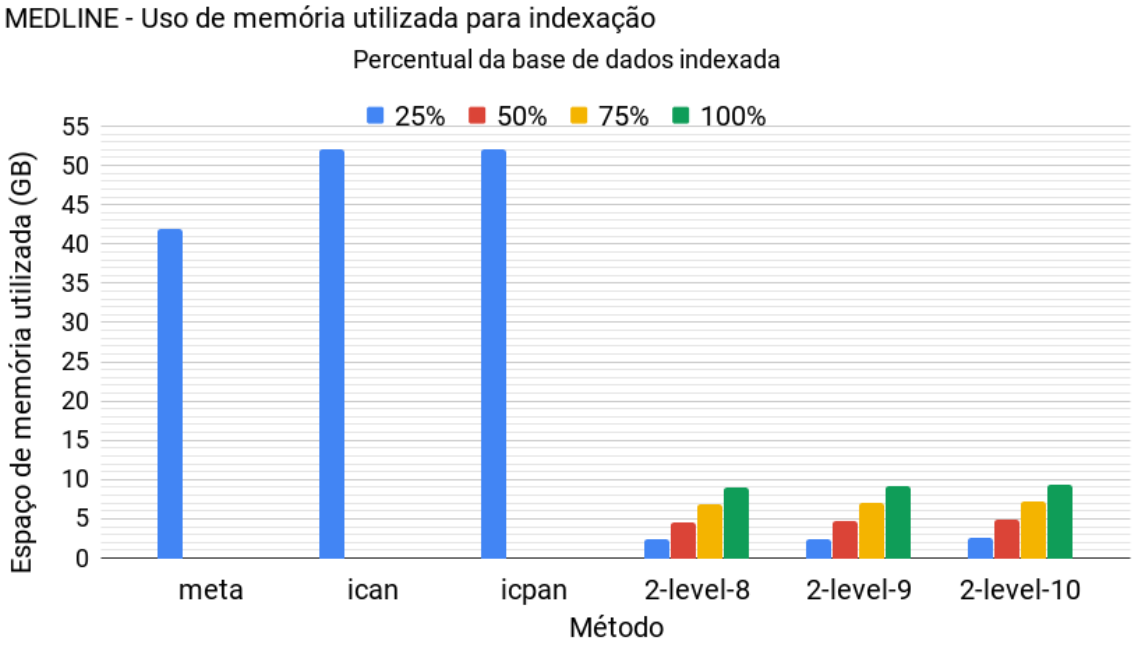

Figura 4.12: MEDLINE - quantidade de memória em GB utilizada por cada método. Para 25%, 50%, 75% e 100% da base de dados. Nota-se que os métodos META, ICAN e ICPAN mostram apenas um valor, isso ocorreu pela insuficiência de memória disponível na máquina dos experimentos, que é de 64 GB. O método BEVA não conseguiu indexar completamente nenhuma parcial da base de dados.

Os dados obtidos mostram que, para que as soluções eficientes para complementação automática de consultas até então existentes na literatura consigam indexar as bases de dados muito grandes, é necessário a utilização de uma grande quantidade de memória. Em contrapartida, os métodos 2-level conseguem indexar grandes bases de dados e processar consultas utilizando uma quantidade de memória muito menor.

Apesar de utilizar uma quantidade de memória muito inferior em relação ao IncNG-Trie, o BEVA utiliza uma quantidade de memória relativamente superior comparado

com os outros métodos. A utilização do BEVA para uma base de dados do tamanho da USADDR requer mais de 16 GB de memória apenas para utilização do método de complementação automática, o que pode ser considerado inviável em servidores ou máquinas mais simples, com menos quantidade de memória, sendo indicado para aplicações onde a quantidade de memória disponível não seja um fator importante.

A análise da utilização de memória dos métodos mostra um certo padrão de valores entre o META e os métodos ICAN e ICPAN. Como toda sentença não precisa ser indexada, os métodos 2-level estão bem abaixo desse padrão, obtendo uma utilização otimizada de memória, observa-se nas Tabelas 4.13, 4.14 e 4.15 que a razão da quantidade de memória utilizada pelo 2-level-8, 2-level-9 e 2-level-10 em relação aos outros métodos é significativamente menor. A menor razão registrada no experimento foi de 31,11% para a base de dados completa USADDR. Ao comparar com o método base do método de dois níveis, o ICAN, consegue-se poupar cerca de 85% de memória na base de dados AOL, 75% na base de dados USADDR e cerca de 95% em 25% da base de dados MEDLINE.

Portanto, os dados obtidos através dos experimentos confirmam a eficácia em economia de memória ao utilizar uma indexação de itens para processamento em dois níveis. Os experimentos a seguir mostrarão os dados referente à eficiência em tempo de processamento de consulta do do método de dois níveis em relação aos principais métodos da literatura, importante para avaliar a viabilidade da utilização de um método em dois níveis para complementação automática de sentenças.

4.3.2 Variando o tamanho da consulta

Para que o usuário tenha uma boa experiência ao usar um sistema de complementação automática de sentenças, deve-se mostrar as sugestões a uma velocidade maior do que o tempo de digitação do usuário, proporcionando assim uma sensação de instantaneidade. Nesta seção serão analisados os dados obtidos nos experimentos variando o tamanho das consultas.

Os primeiros caracteres são os mais importantes em um sistema de complementação

automática de sentenças, pois a tendência é que o usuário encontre, ou não encontre, o que ele procura nesses primeiros caracteres digitados.

As consultas utilizadas para os testes foram geradas através de seleção aleatória de mil itens da base de dados, seguindo Chaudhuri and Kaushik (2009). Para cada consulta, modificou-se um caractere aleatório para caracterizar uma consulta com erro. Para todas as bases de dados foram criados seis conjuntos de consultas, variando o tamanho da consulta em $|q| = 3$, $|q| = 5$, $|q| = 7$, $|q| = 9$, $|q| = 11$ e $|q| = 13$, serão utilizados apenas valores ímpares para que seja possível abranger um intervalo maior de tamanhos de consultas mantendo a disposição dos dados de resultados de forma mais simplificada. É importante verificar a diferença no processamento a partir do tamanho $|q| = 8$, onde o 2-level passa a utilizar o segundo nível no processamento das consultas. Os experimentos foram realizados com o limite de distância de edição igual a um, visto que o fator importante para esse experimento é a variação do tamanho das consultas. Os resultados em tempo de execução para as consultas realizadas podem ser observadas nas Tabelas 4.16, 4.17 e 4.18 e suas respectivas ilustrações gráficas nas Figuras 4.13 e 4.14 e 4.15.

A Tabela 4.16 e a Figura 4.13 mostram uma pequena diferença do padrão de eficiência de tempo entre os métodos para a base de dados AOL, somente o META possui um aumento de tempo maior na medida que o tamanho do prefixo de consulta aumenta. Como a base de dados AOL é pequena em relação às outras bases de dados aqui utilizadas, é importante notar que não há um aumento muito grande no tempo de consulta do 2-level, sinalizando que a quantidade de dados filtrados pelo primeiro nível e processados no segundo não demanda muito tempo de processamento.

O método IncNGTrie mostra-se bastante eficiente nos experimentos utilizando a base de dados AOL. A indexação das sentenças utilizando geração de vizinhança para gerar as possíveis correções diminui a quantidade de processamentos durante a consulta. Como resultado, possui uma eficiência acima do padrão dos outros métodos. Entretanto, a alta utilização de memória para indexação não permitiu que esse método fosse testado nas demais bases de dados.

Tabela 4.16: AOL - Tempos médios (em milissegundos) para as consultas variando o comprimento do prefixo de consulta para cada método de complementação automática de sentenças. Com o limite de distância de edição definido igual a um.

	Comprimento do prefixo de consulta					
	$\|q\| = 3$	$\|q\| = 5$	$\|q\| = 7$	$\|q\| = 9$	$\|q\| = 11$	$\|q\| = 13$
IncNGTrie	**0,027 ± 0,001**	**0,02 ± 0,001**	**0,014 ± 0,001**	**0,019 ± 0,001**	**0,031 ± 0,001**	**0,03 ± 0,001**
META	0,054 ± 0,01	0,072 ± 0,03	0,087 ± 0,03	0,104 ± 0,04	0,125 ± 0,04	0,138 ± 0,04
ICAN	0,084 ± 0,08	0,075 ± 0,07	0,075 ± 0,02	0,075 ± 0,02	0,077 ± 0,02	0,076 ± 0,03
ICPAN	0,063 ± 0,06	0,051 ± 0,02	0,051 ± 0,02	0,051 ± 0,02	0,05 ± 0,02	0,05 ± 0,02
BEVA	0,043 ± 0,008	0,045 ± 0,009	0,046 ± 0,01	0,046 ± 0,009	0,047 ± 0,01	0,048 ± 0,01
2-level-8	0,072 ± 0,08	0,062 ± 0,01	0,061 ± 0,01	0,071 ± 0,04	0,073 ± 0,05	0,076 ± 0,06
2-level-9	0,073 ± 0,90	0,063 ± 0,01	0,062 ± 0,01	0,061 ± 0,01	0,067 ± 0,01	0,073 ± 0,01
2-level-10	0,073 ± 0,04	0,064 ± 0,01	0,061 ± 0,03	0,062 ± 0,01	0,064 ± 0,02	0,068 ± 0,02

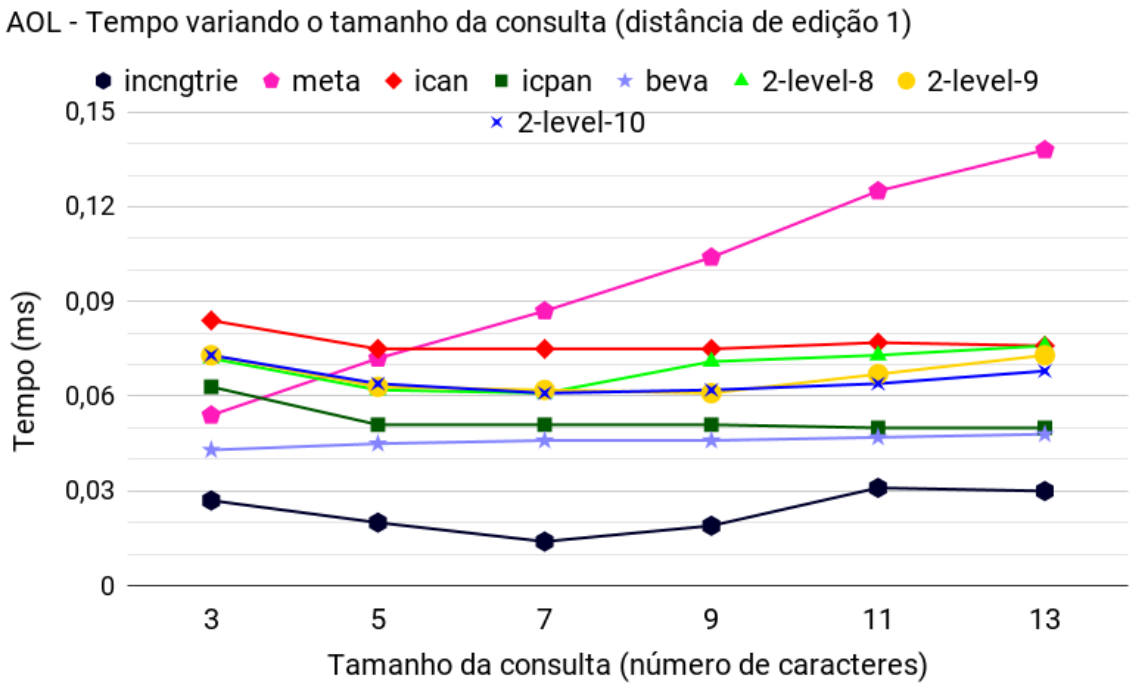

Figura 4.13: AOL - comparação do tempo médio de processamento total de consultas variando comprimento do prefixo de consulta para cada método de complementação automática de sentenças. Com o limite de distância de edição definido igual a um

No geral, todos os métodos realizaram os processamentos abaixo de 0,20 ms, muito abaixo do limite de tempo aceitável entre 100 e 200 ms, comportamento esperado se tratando de uma base de dados pequena.

Tabela 4.17: USADDR - Tempos médios (em milissegundos) para as consultas variando o comprimento do prefixo de consulta para cada método de complementação automática de sentenças. Com o limite de distância de edição definido igual a um.

	Comprimento do prefixo de consulta					
	|q| = 3	|q| = 5	|q| = 7	|q| = 9	|q| = 11	|q| = 13
META	0,492 ± 1,39	0,346 ± 0,08	0,387 ± 0,09	0,449 ± 0,08	0,507 ± 0,09	0,578 ± 0,09
ICAN	0,521 ± 1,63	0,298 ± 0,18	0,295 ± 0,16	0,28 ± 0,16	0,284 ± 0,14	0,277 ± 0,13
ICPAN	0,508 ± 0,98	0,191 ± 0,17	0,182 ± 0,15	0,181 ± 0,15	0,188 ± 0,15	0,181 ± 0,16
BEVA	**0,586 ± 0,35**	**0,181 ± 0,23**	**0,142 ± 0,21**	**0,125 ± 0,14**	**0,124 ± 0,12**	**0,122 ± 0,12**
2-level-8	0,485 ± 0,91	0,253 ± 0,29	0,243 ± 0,25	1,478 ± 2,31	1,591 ± 2,49	1,713 ± 2,98
2-level-9	0,493 ± 1,21	0,267 ± 0,10	0,242 ± 0,05	0,257 ± 0,07	0,754 ± 2,13	0,828 ± 2,78
2-level-10	0,496 ± 0,93	0,273 ± 0,04	0,248 ± 0,05	0,261 ± 0,03	0,669 ± 2,33	0,756 ± 3,02

Para a base de dados USADDR, a Tabela 4.17 e a Figura 4.14 mostram uma diferença de tempo maior entre os métodos. Uma maior quantidade de itens significa um resultado com um número maior de itens no primeiro nível dos métodos 2-level-8, 2-level-9 e 2-level-10, resultando em um salto entre o tempo de consulta em que é necessário utilizar o segundo nível, sendo assim, o tempo de processamento para consultas de comprimento maior que oito encontra-se fora do padrão de tempo dos outros métodos, com uma casa decimal a mais para essas consultas.

Apesar do tempo médio de consulta dos métodos ser maior que as médias realizadas na base de dados AOL, todos os métodos realizaram as consultas em menos de 0,6 ms, exceto os métodos 2-level quando necessita usar o segundo nível, mas mesmo assim não ultrapassa 2 ms de tempo de processamento. Dessa forma, para a base de dados USADDR, todos os métodos também processaram as consultas em um tempo médio muito inferior ao limite aceitável.

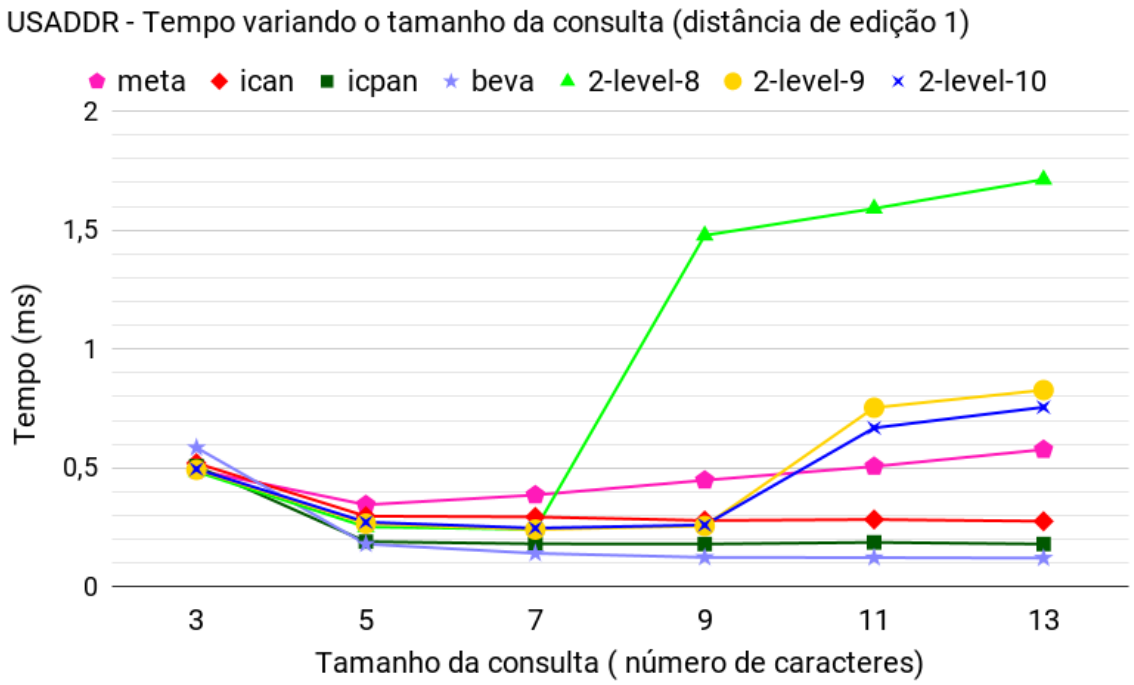

Figura 4.14: USADDR, comparação do tempo médio de processamento total de consultas variando comprimento do prefixo de consulta para cada método de complementação automática de sentenças. Com o limite de distância de edição definido igual a um

As informações sobre os experimentos realizados utilizando a base de dados MEDLINE estão dispostas na Tabela 4.18 e na Figura 4.15. Como os demais métodos além do 2-level-8, 2-level-9 e 2-level-10 não conseguiram indexar a base de dados completa utilizando a quantidade de memória disponível, os experimentos foram realizados utilizando a parcial de 25% da base. Não foi possível realizar a indexação do método BEVA mesmo em apenas 25% da base de dados, então não será incluído nestes experimentos.

Apesar do tempo médio ser superior em relação às outras bases de dados, o META, ICAN e ICPAN mantiveram o mesmo padrão de diferença de tempo. Entretanto, os resultados mostram um salto maior entre as consultas realizadas pelos métodos 2-level utilizando apenas o primeiro nível e quando passa a utilizar o segundo nível. Outra observação é sobre as consultas que utilizam o segundo nível, o crescimento de tempo de consulta na medida que o comprimento do prefixo de consulta aumenta está mais acentuado, isso ocorre pois o tamanho médio das descrições dos itens é maior, então é necessário mais esforço de processamento no segundo nível para calcular a distância

Tabela 4.18: MEDLINE - Tempos médios (em milissegundos) para as consultas variando o comprimento do prefixo de consulta para cada método de complementação automática de sentenças para 25% da base de dados. Com o limite de distância de edição definido igual a um.

	Comprimento do prefixo de consulta					
	|q| = 3	|q| = 5	|q| = 7	|q| = 9	|q| = 11	|q| = 13
META	0,697 ± 1,39	0,489 ± 0,13	0,491 ± 0,13	0,592 ± 0,14	0,676 ± 0,15	0,715 ± 0,16
ICAN	1,041 ± 1,98	0,523 ± 0,13	0,496 ± 0,12	0,468 ± 0,11	0,471 ± 0,11	0,473 ± 0,10
ICPAN	**0,991 ± 2,30**	**0,352 ± 0,09**	**0,276 ± 0,09**	**0,315 ± 0,10**	**0,336 ± 0,10**	**0,307 ± 0,10**
2-level-8	0,947 ± 0,12	0,388 ± 0,11	0,379 ± 0,06	5,124 ± 1,07	5,735 ± 4,17	6,559 ± 5,90
2-level-9	0,945 ± 1,01	0,384 ± 0,07	0,381 ± 0,07	0,382 ± 0,06	3,978 ± 4,87	4,281 ± 5,37
2-level-10	0,95 ± 1,86	0,392 ± 0,09	0,385 ± 0,07	0,385 ± 0,07	3,367 ± 4,38	3,691 ± 9,14

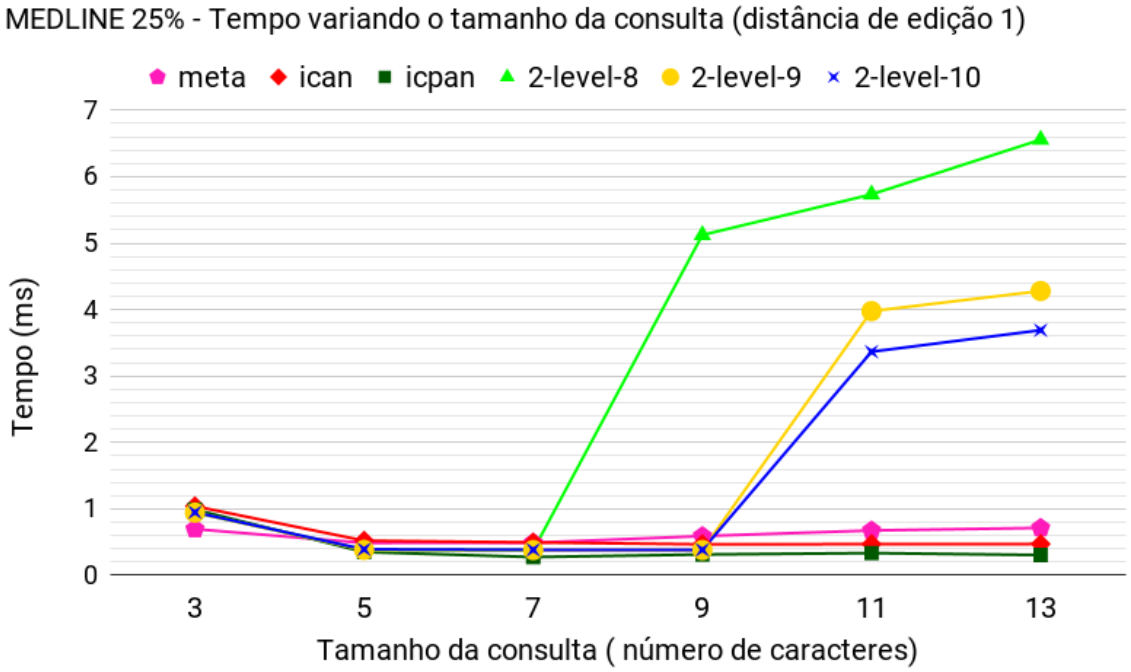

Figura 4.15: MEDLINE - comparação do tempo médio de processamento total de consultas variando comprimento do prefixo de consulta para cada método de complementação automática de sentenças. Com o limite de distância de edição definido igual a um

de edição.

Assim como ocorrido nos tempos médios dos experimentos das bases de dados AOL

e USADDR, todos os métodos realizaram as consultas em tempo médio muito inferior limite aceitável, que é entre 100 e 200 ms. A maior média de tempo obtida foi do 2-level-8, 6,559 ms, ao utilizar o processamento do segundo nível em consultas de comprimento igual a treze. Ao considerar apenas os demais métodos, a maior média de tempo registrada foi de 0,715 ms, alcançado pelo método META.

O META mostrou-se bastante eficiente para o cálculo dos primeiros caracteres, porém, o aumento do tempo ao crescer o tamanho da consulta mostra que a necessidade do processamento da estrutura para calcular os caracteres correspondentes demanda mais tempo para consultas maiores.

A utilização apenas da *trie* como forma de processamento das consultas, faz com que os métodos ICAN e ICPAN mantenham o mesmo padrão de tempo para consultas de qualquer tamanho, obtendo um padrão entre ambas, com uma eficiência melhor para o ICPAN, que utiliza apenas os nós ativos principais para economizar comparações e realizar os cálculos de distância de edição de forma mais otimizada em relação à utilização de todos os nós ativos, como ocorre no ICAN. Entre os métodos comparados na base de dados MEDLINE, o BEVA possui a melhor performance em média de tempo de processamento de consultas para o limite de distância de edição igual a um.

O BEVA também manteve a eficiência no processamento de consultas de forma constante. Entre os métodos comparados na base de dados USADDR, o BEVA possui a melhor performance em média de tempo de processamento de consultas para o limite de distância de edição igual a um.

Para consultas de comprimento do prefixo abaixo do tamanho dos prefixos de indexação na trie, ocorrendo em |q| = 3, |q| = 5 e q = |7|, os métodos baseados na abordagem de dois níveis mostram-se superiores ao ICAN, em quem são baseados no primeiro nível, essa superioridade se dá pela inferior quantidade de caracteres indexados na *trie*, pois o 2-level indexa apenas os oito primeiros caracteres. Com isso, há menos processos ao percorrer os filhos dos nós ativos.

Como é o único a utilizar duas formas de calcular as consultas, de acordo com o tamanho do prefixo da consulta, o 2-level-8 perde eficiência a partir do nono caractere,

o 2-level-9 a partir do décimo, e o 2-level-10 perde eficiência a partir do décimo primeiro caractere, mostrando perdas maiores de eficiência em escala, como no caso do USADDR e MEDLINE.

4.3.3 Variando o limite de distância de edição

A possibilidade de corrigir erros de digitação enviados pelo usuário contribui para a recuperação de informações relevantes, quanto maior for a distância de edição suportada pelo processamento da consulta, mais erros poderão ser tratados, porém o tempo de processamento também aumenta. Nesta seção serão analisados os resultados dos experimentos variando a distância de edição.

Ao levar em consideração distância de edição em uma consulta textual, deve-se observar que, além de corrigir erros quando os usuários inserirem caracteres errados na consulta, todas as possibilidades de inserção, remoção e troca de caracteres da consulta em relação aos dados indexados serão analisados. Então um efeito colateral de processar consultas considerando o limite de distância de edição, além de maior custo computacional, é a inserção de resultados que não possuem relação com a expectativa de resultados para o usuário. Sendo assim, limites de edição altos podem fazer com que um grande conjunto de resultados seja retornado, mesmo que, em sua maior parte, os resultados sejam irrelevantes.

Sendo assim, levando em consideração que usuários, na maioria dos casos, decidem qual sugestão utilizar quando o prefixo de consulta está com tamanho entre 6 e 8 (Oliveira, 2018), nesses casos o limite de edição acima de 2 já pode retornar muitos resultados irrelevantes. Sendo assim, os experimentos desta seção levarão em conta os limites de edição iguais a 2 e 3.

Nesse experimento, os métodos foram executados utilizando limite de distância de edição igual a dois e três, para cada limite de edição os métodos executaram consultas de tamanho cinco, sete, nove, onze e treze, visto que consultas abaixo de tamanho cinco retornam um conjunto de resultados com muitos itens irrelevantes para a consulta. As consultas utilizadas para os testes foram geradas através de seleção aleatória de cem

itens da base de dados, seguindo Chaudhuri and Kaushik (2009). Para cada consulta, um caractere aleatório foi modificado para caracterizar uma consulta com erro. Os resultados em tempo de execução para as consultas realizadas pode ser observado nas Tabelas 4.19, 4.20, 4.21, 4.22, 4.23 e 4.24, com os gráficos ilustrando os valores nas Figuras 4.16, 4.17 e 4.18.

Tabela 4.19: AOL - Tempo médio (em milissegundos), com intervalo de confiança de 95%, para as consultas variando o comprimento do prefixo de consulta, para limite de distância de edição igual a 2. Para a base de dados AOL

	Comprimento do prefixo de consulta				
	|q| = 5	|q| = 7	|q| = 9	|q| = 11	|q| = 13
IncNGTrie	**0,029 ± 0,01**	**0,023 ± 0,01**	**0,032 ± 0,01**	**0,026 ± 0,01**	**0,023 ± 0,01**
META	1,478 ± 0,55	2,118 ± 0,60	2,834 ± 0,64	3,103 ± 0,67	3,809 ± 0,73
ICAN	1,424 ± 0,18	1,335 ± 0,18	1,388 ± 0,18	1,349 ± 0,19	1,332 ± 0,23
ICPAN	0,593 ± 0,13	0,598 ± 0,14	0,695 ± 0,13	0,603 ± 0,13	0,587 ± 0,14
BEVA	0,986 ± 0,07	1,023 ± 0,09	1,038 ± 0,08	1,066 ± 0,09	1,061 ± 0,08
2-level-8	1,209 ± 0,13	1,175 ± 0,13	1,206 ± 0,18	1,248 ± 0,13	1,275 ± 0,16
2-level-9	1,233 ± 0,18	1,224 ± 0,12	1,108 ± 0,13	1,162 ± 0,13	1,152 ± 0,24
2-level-10	1,257 ± 0,15	1,237 ± 0,14	1,119 ± 0,23	1,135 ± 0,13	1,119 ± 0,13

As Tabelas 4.19 e 4.20 e a Figura 4.16 mostram que, para os limites de distância de edição 2 e 3, o IncNGTrie obteve uma média de tempo melhor que os outros métodos, obtendo um tempo máximo de 0,62 ms para uma consulta levando em consideração o limite de distância de edição igual a 2, valores de duas casas decimais menores que os demais métodos. Entretanto o IncNGTrie precisa indexar os termos e as suas possibilidades de erros, sendo assim, a quantidade de memória necessária para indexação cresce consideravelmente para cada aumento no limite de distância de edição.

O 2-level possui uma variação menor entre o tempo médio de processamento de consultas que utilizam apenas o primeiro nível e o tempo médio de quando o segundo nível é utilizado. Para limite de distância de edição 1, o crescimento médio de tempo

Tabela 4.20: AOL - Tempo médio (em milissegundos), com intervalo de confiança de 95%, para as consultas variando o comprimento do prefixo de consulta, para limite de distância de edição igual a 3. Para a base de dados AOL

	Comprimento do prefixo de consulta				
	$\|q\| = 5$	$\|q\| = 7$	$\|q\| = 9$	$\|q\| = 11$	$\|q\| = 13$
IncNGTrie	**0,053 ± 0,02**	**0,062 ± 0,02**	**0,048 ± 0,02**	**0,057 ± 0,02**	**0,061 ± 0,02**
META	25,599 ± 6,85	30,733 ± 8,72	31,524 ± 15,66	32,317 ± 14,28	32,826 ± 9,03
ICAN	10,006 ± 1,63	13,966 ± 1,20	13,004 ± 2,03	13,874 ± 1,98	14,287 ± 1,72
ICPAN	5,236 ± 0,46	2,652 ± 0,72	2,681 ± 0,44	2,673 ± 0,45	2,725 ± 0,44
BEVA	5,759 ± 0,25	6,219 ± 0,28	6,321 ± 0,21	6,251 ± 0,29	6,593 ± 0,98
2-level-8	11,333 ± 1,60	8,758 ± 1,20	9,756 ± 0,90	9,947 ± 1,10	9,075 ± 1,40
2-level-9	10,915 ± 2,43	10,499 ± 1,98	9,91 ± 1,00	9,836 ± 1,21	11,032 ± 2,74
2-level-10	10,158 ± 1,50	10,085 ± 1,14	10,829 ± 1,30	11,059 ± 1,35	11,644 ± 3,89

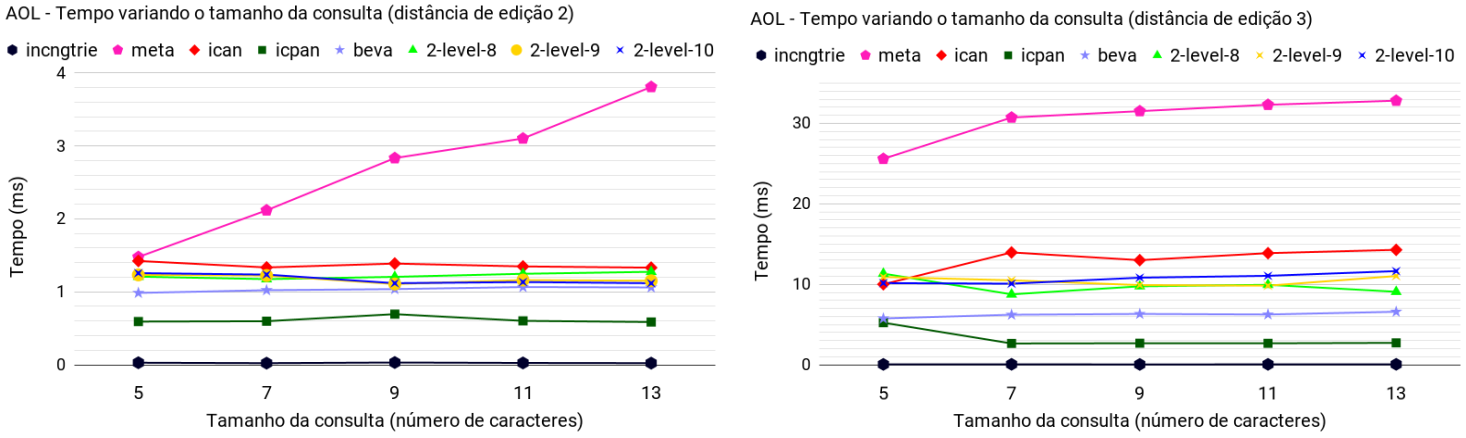

Figura 4.16: AOL - comparação do tempo médio de processamento total de consultas dos métodos variando o tamanho de consulta, para os limites de distância de edição 2 e 3.

entre $|q| = 7$ e $|q| = 9$ para a base AOL é de 16%, para limite de distância de edição 2 é de cerca de 2%. Isso acontece pois o processo utilizado no primeiro nível torna-se mais custoso quanto maior for o limite de edição, pois haverá mais nós ativos para serem processados, ao contrário do processo utilizado no segundo nível, que mantém o mesmo custo de processamento independente do limite de distância de edição utilizado.

Um outro dado importante para notar é tempo de processamento do META, que revelou-se muito superior aos demais métodos, crescendo bastante ao incrementar o limite de distância de edição, mesmo para uma base de dados relativamente pequena, chegando a uma média de tempo de processamento de quase 4 ms para distância de edição 2 e chegando em quase 35 ms para distância de edição 3, quase o dobro do tempo máximo utilizado pelo ICAN, que obteve a segunda maior média de tempo tanto para o limite de edição 2 quanto o limite de edição 3.

As médias de tempo para todos os métodos estiveram abaixo do limite de tempo crítico para processamento de complementação automática de consultas, que é entre 100 ms e 200 ms. Todos os métodos são capazes de oferecer uma solução satisfatória de complementação automática para a base de dados AOL.

Tabela 4.21: USADDR - Tempo médio (em milissegundos), com intervalo de confiança de 95%, para as consultas variando o comprimento do prefixo de consulta, para limite de distância de edição igual a 3. Para a base de dados USADDR

	Comprimento do prefixo de consulta				
	|q| = 5	|q| = 7	|q| = 9	|q| = 11	|q| = 13
META	72,049 ± 16,59	73,701 ± 15,51	77,515 ± 15,70	82,918 ± 18,41	85,528 ± 15,43
ICAN	11,342 ± 2,58	10,919 ± 3,40	11,347 ± 3,09	11,303 ± 3,15	11,729 ± 3,57
ICPAN	7,041 ± 0,70	6,501 ± 0,60	6,633 ± 0,74	9,796 ± 0,70	10,428 ± 0,69
BEVA	**7,955 ± 0,13**	**7,374 ± 0,26**	**7,273 ± 0,31**	**7,695 ± 0,29**	**8,356 ± 0,35**
2-level-8	9,168 ± 1,05	12,566 ± 0,74	14,176 ± 6,85	16,933 ± 7,48	17,452 ± 8,64
2-level-9	9,349 ± 0,51	10,587 ± 0,80	11,983 ± 0,80	14,593 ± 4,09	15,233 ± 4,61
2-level-10	9,992 ± 1,24	10,557 ± 0,98	12,293 ± 1,02	13,715 ± 5,86	14,261 ± 3,99

Os dados dos experimentos realizados utilizando a base de dados USADDR, dispostos nas Tabelas 4.21 e 4.22 e na Figura 4.17, mostram uma diferença no padrão de performance entre os métodos observado nos experimentos utilizando a base de dados AOL, além da média de tempo maior, que ocorre naturalmente devido à maior quantidade de informações a serem processadas.

Tabela 4.22: USADDR - Tempo médio (em milissegundos), com intervalo de confiança de 95%, para as consultas variando o comprimento do prefixo de consulta, para limite de distância de edição igual a 3. Para a base de dados USADDR

	Comprimento do prefixo de consulta				
	$\|q\| = 5$	$\|q\| = 7$	$\|q\| = 9$	$\|q\| = 11$	$\|q\| = 13$
META	7049,317 ± 1141,43	7356,571 ± 1641,42	7380,944 ± 1722,31	7474,746 ± 1642,44	7589,579 ± 1722,84
ICAN	198,402 ± 29,58	212,815 ± 22,40	220,087 ± 31,65	207,665 ± 24,36	217,774 ± 28,14
ICPAN	69,121 ± 18,96	71,019 ± 21,67	70,362 ± 17,55	73,321 ± 20,14	73,473 ± 15,54
BEVA	**70,395 ± 4,25**	**71,035 ± 5,16**	**73,183 ± 5,86**	**69,499 ± 5,98**	**71,065 ± 6,08**
2-level-8	182,175 ± 8,07	189,943 ± 18,23	202,904 ± 19,37	198,254 ± 24,32	199,739 ± 28,32
2-level-9	183,022 ± 9,23	189,152 ± 26,44	196,776 ± 35,82	199,188 ± 30,66	200,081 ± 28,35
2-level-10	183,913 ± 5,24	190,401 ± 24,59	194,871 ± 28,57	198,399 ± 31,40	199,403 ± 29,36

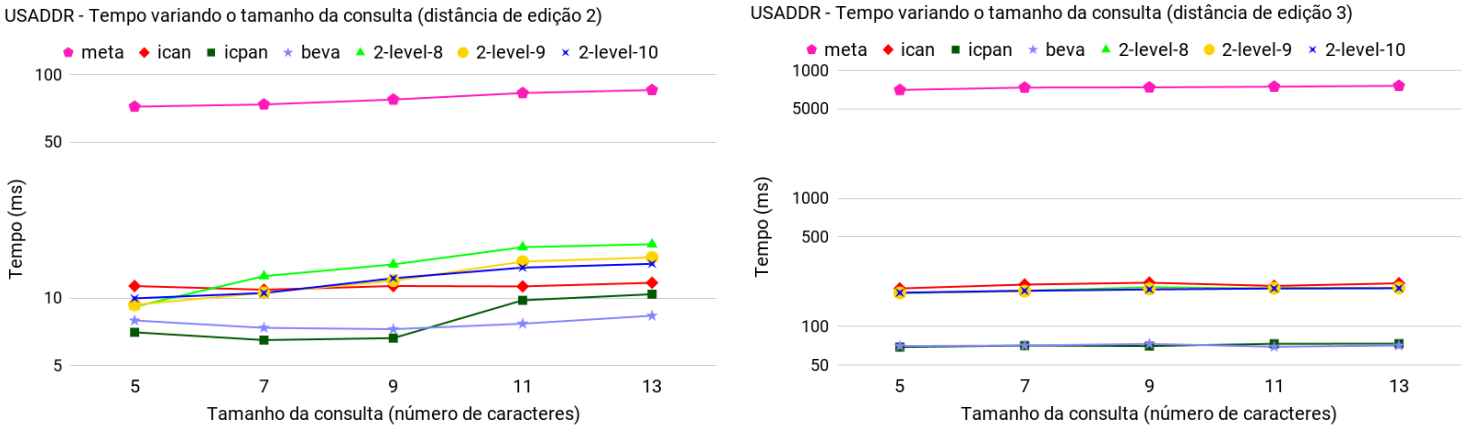

Figura 4.17: USADDR - comparação em escala logarítmica do tempo médio de processamento total de consultas dos métodos variando o tamanho de consulta, para os limites de distância de edição 2 e 3.

O META obteve uma performance inviável para o tamanho da base de dados nas consultas levando em consideração limite de edição 3, tendo uma média de tempo superior a 7.000 ms, o que inviabiliza sua utilização para complementação automática de sentenças nesse contexto. Para a limite de edição 2, as médias de tempo ficam abaixo de 100 ms, ainda sendo viável sua utilização para esse contexto, mesmo sendo uma média de tempo muito acima das médias de tempo obtidas pelos demais métodos.

A principal diferença percebida em relação aos resultados da base AOL foi quanto à performance entre o ICAN e os métodos 2-level. Apesar do 2-level obter resultados melhores em relação ao ICAN nos experimentos utilizando a base AOL, os resultados para a base USADDR e limite de distância de edição 2 mostram uma performance melhor do ICAN em relação ao 2-level. Isso ocorre pois a quantidade de dados a serem processados faz com que, mesmo para limite de edição 2, o segundo nível do 2-level ainda seja bem mais custoso do que o primeiro nível, enquanto o ICAN, que utiliza apenas a trie, continua com tempo relativamente mais estável na medida que o prefixo de consulta cresce. Já para o limite de edição 3, com um custo para processamento de nós ativos maior, a diferença entre a média de tempo entre ICAN e 2-level seguiu o mesmo padrão que nos experimentos da base AOL, com o 2-level obtendo uma média de tempo menor.

Desconsiderando o META, a maior média de tempo registrada para o limite de edição 2 foi de 17,45 ms, obtido pelo 2-level-8, muito abaixo do limite crítico para complementação automática. Já para o limite de edição 3, apenas o ICPAN e o BEVA obtiveram média de tempo abaixo do limite aceitável para uma boa experiência dos usuários, com uma média de 73 ms e 71 ms respectivamente. O 2-level teve uma média de tempo de cerca de 195 ms, no extremo do limite crítico, que é entre 100 ms e 200 ms, o ICAN obteve uma média um pouco acima dos 200 ms. Os valores de intervalo de segurança são maiores para estes resultados, sendo assim, para o ICAN, 2-level-8, 2-level-9 e 2-level-10, a média de tempo pode variar bastante para cima ou para baixo de 200 ms.

A execução dos experimentos utilizando a parcial de 25% da base de dados MEDLINE, cujos dados estão dispostos nas Tabelas 4.23 e 4.24 e na Figura 4.18, não teve participação do método BEVA, por conta do alto consumo de memória não suportado pela máquina utilizada. Apesar da base de dados MEDLINE possuir quase três vezes mais itens que a USADDR, estes experimentos foram executados utilizando 25% da base de dados MEDLINE, motivo pela qual as médias de tempo serem inferiores às médias obtidas ao utilizar a base de dados USADDR.

Tabela 4.23: MEDLINE 25% - Tempo médio (em milissegundos), com intervalo de confiança de 95%, para as consultas variando o comprimento do prefixo de consulta, para limite de distância de edição igual a 2. Para 25% da base de dados MEDLINE.

	Comprimento do prefixo de consulta				
	|q| = 5	|q| = 7	|q| = 9	|q| = 11	|q| = 13
META	70,148 ± 24,36	74,524 ± 15,23	76,338 ± 19,08	77,298 ± 16,68	78,987 ± 19,02
ICAN	8,389 ± 1,60	8,442 ± 1,58	8,304 ± 1,61	8,224 ± 1,48	9,135 ± 1,58
ICPAN	**4,901 ± 1,07**	**4,914 ± 0,89**	**4,722 ± 0,93**	**4,505 ± 1,22**	**4,575 ± 1,28**
2-level-8	6,514 ± 1,27	6,623 ± 6,87	10,375 ± 4,15	12,148 ± 6,16	11,549 ± 6,09
2-level-9	6,661 ± 1,08	6,649 ± 5,32	10,508 ± 1,41	11,515 ± 5,13	11,413 ± 9,92
2-level-10	8,044 ± 2,43	7,129 ± 4,69	9,856 ± 2,31	10,825 ± 4,95	10,907 ± 5,96

Tabela 4.24: MEDLINE 25% - Tempo médio (em milissegundos), com intervalo de confiança de 95%, para as consultas variando o comprimento do prefixo de consulta, para limite de distância de edição igual a 3. Para 25% da base de dados MEDLINE.

	Comprimento do prefixo de consulta				
	|q| = 5	|q| = 7	|q| = 9	|q| = 11	|q| = 13
META	2677,057 ± 730,55	2655,779 ± 743,01	2767,132 ± 731,90	2763,759 ± 761,22	2717,245 ± 762,04
ICAN	109,257 ± 31,70	100,923 ± 20,70	99,785 ± 23,94	103,119 ± 17,80	103,264 ± 25,86
ICPAN	**32,852 ± 12,37**	**32,801 ± 11,85**	**35,851 ± 11,39**	**35,063 ± 12,24**	**37,049 ± 9,68**
2-level-8	88,369 ± 10,36	89,822 ± 17,88	95,166 ± 26,03	95,261 ± 27,06	95,963 ± 17,72
2-level-9	89,136 ± 17,91	89,951 ± 20,63	93,791 ± 24,01	96,543 ± 31,55	94,732 ± 22,53
2-level-10	87,775 ± 19,99	90,333 ± 21,84	93,206 ± 28,94	96,775 ± 24,99	94,361 ± 25,36

Assim como nos experimentos utilizando a base de dados USADDR, o META obteve uma média de tempo muito superior aos demais métodos, com quase 80 ms para limite de edição 2 e mais de 2.000 ms para limite de edição 3. Entre os demais métodos, a maior média obtida para limite de edição 2 é de cerca de 12 ms pelo 2-level-8 e para limite de edição 3 é de cerca de 103 ms pelo ICAN.

Para a base de dados MEDLINE, apenas o 2-level é capaz de indexar e processar

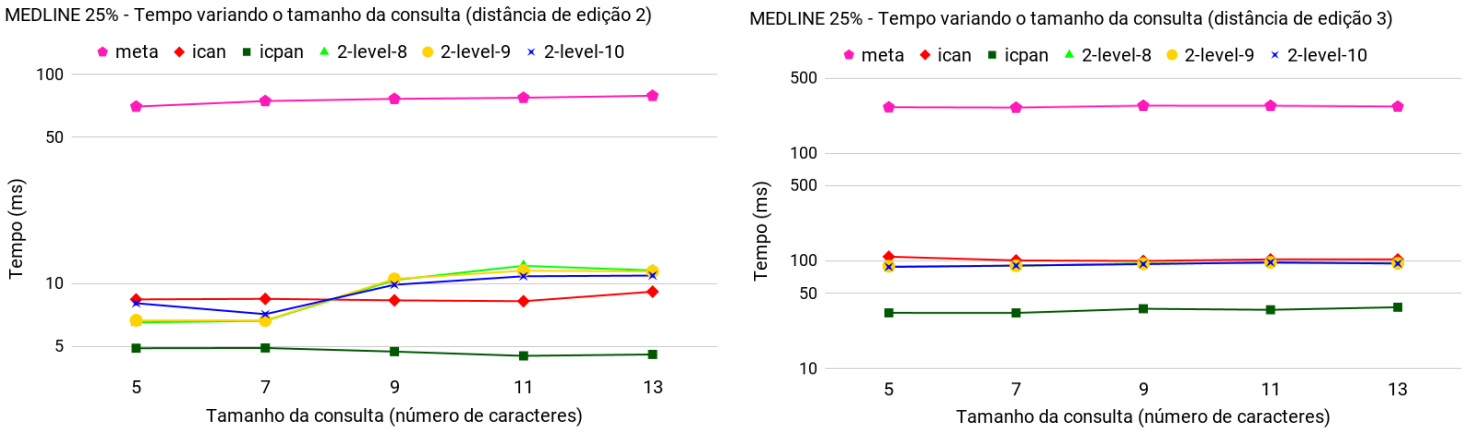

Figura 4.18: MEDLINE - comparação em escala logarítmica do tempo médio de processamento total de consultas em 25% da base de dados MEDLINE variando o tamanho de consulta, para os limites de distância de edição 2 e 3.

consultas levando em consideração a limitação de memória de 64 GB, mas como o experimento leva em consideração apenas 25% da base de dados, é possível verificar o comportamento dos demais métodos. E para 25% da base de dados, é possível verificar a viabilidade de todos os métodos para limite de edição 2 e, excluindo o META, verificamos a viabilidade para limite de edição 3, com a maior média obtida (103 ms) estando no limite inicial do intervalo crítico.

Tabela 4.25: MEDLINE - Tempo médio (em milissegundos), com intervalo de confiança de 95%, para as consultas variando o comprimento do prefixo de consulta, para limite de distância de edição igual a 1. Para a base de dados MEDLINE

	Comprimento do prefixo de consulta					
	$\|q\| = 3$	$\|q\| = 5$	$\|q\| = 7$	$\|q\| = 9$	$\|q\| = 11$	$\|q\| = 13$
2-level-8	3,046 ± 5,14	0,763 ± 1,09	0,582 ± 1,13	19,682 ± 1,04	22,093 ± 5,25	24,777 ± 5,62
2-level-9	3,012 ± 4,98	0,773 ± 1,11	0,643 ± 1,49	0,604 ± 1,21	19,597 ± 1,04	22,149 ± 1,10
2-level-10	3,005 ± 4,91	0,763 ± 1,03	0,644 ± 1,14	0,637 ± 1,27	17,897 ± 4,86	20,247 ± 1,02

Como nenhum dos outros métodos foi capaz de indexar 100% da base de dados MEDLINE com a quantidade de memória disponível, as Tabelas 4.25, 4.26 e 4.27 mostram os valores médios de tempo de processamento variando o comprimento da consulta para o 2-level-8, 2-level-9 e 2-level-10, com a média de tempo para limites de distância de edição 1, 2 e 3, seguindo o mesmo padrão dos experimentos anteriores.

Tabela 4.26: MEDLINE - Tempo médio (em milissegundos), com intervalo de confiança de 95%, para as consultas variando o comprimento do prefixo de consulta, para limite de distância de edição igual a 2. Para a base de dados MEDLINE

	Comprimento do prefixo de consulta				
	|q| = 5	|q| = 7	|q| = 9	|q| = 11	|q| = 13
2-level-8	39,046 ± 7,23	16,614 ± 7,30	18,251 ± 14,60	27,55 ± 21,00	31,272 ± 23,67
2-level-9	12,953 ± 2,46	12,961 ± 2,53	12,977 ± 2,99	26,091 ± 23,06	27,179 ± 22,42
2-level-10	13,310 ± 3,00	12,365 ± 2,45	12,874 ± 3,02	23,119 ± 19,93	24,180 ± 21,39

Tabela 4.27: MEDLINE - Tempo médio (em milissegundos), com intervalo de confiança de 95%, para as consultas variando o comprimento do prefixo de consulta, para limite de distância de edição igual a 3. Para a base de dados MEDLINE

	Comprimento do prefixo de consulta				
	|q| = 5	|q| = 7	|q| = 9	|q| = 11	|q| = 13
2-level-8	194,144 ± 47,389	197,169 ± 37,01	201,369 ± 51,39	212,801 ± 58,58	218,804 ± 53,26
2-level-9	186,924 ± 39,57	192,251 ± 49,76	188,266 ± 42,00	200,896 ± 52,76	209,219 ± 41,75
2-level-10	184,854 ± 33,03	186,25 ± 45,82	177,381 ± 42,44	200,013 ± 43,64	206,619 ± 45,51

Percebe-se um crescimento maior ainda entre as médias de tempo ao utilizar apenas o primeiro nível e ao utilizar o segundo nível, saltando de cerca de 20 ms entre o tamanho de consulta sete e nove para o 2-level-8, e entre nove e onze para o 2-level-9 e o 2-level-10. Apesar do tempo ser superior se comparado com as outras bases de dados, as médias de tempo máximas registrada para os limites de distância de edição 1 e 2 ainda são bem inferiores ao limite de tempo crítico, entre 100 e 200 ms, sendo totalmente viável para utilização em uma base de dados dessa proporção. Para limite de distância de edição 3, a média de tempo obtida ficou acima de 200ms, podendo ficar abaixo levando em consideração o intervalo de confiança.

Ao analisar os dados dos experimentos, o resultado que mais chama atenção é o aumento acentuado que ocorre no META, o tempo médio cresce em padrão exponencial com a variação do limite de distância de edição.

Outro padrão que pode ser observado nos experimentos para todas as bases de dados é que, quanto maior o limite de edição, menor é a diferença entre as médias

de tempo ao aumentar o comprimento. Na medida que aumenta o limite de distância de edição e, por consequência, aumenta os números de nós a serem processados, a relevância do número de caracteres do prefixo de consulta torna-se menor para o custo de processamento.

No geral, para o tempo de processamento das consultas levando em consideração ao tamanho das bases de dados, os limites de edição e o comprimento das consultas, os métodos obtiveram uma performance satisfatória para limites de edição 1 e 2, passando a poder ter demoras perceptíveis aos usuários com os custos maiores para 2-level-8, 2-level-9, 2-level-10 e ICAN para limite de edição 3 na base de dados USADDR. Além disso, método META mostrou-se inviável para ser utilizado com limites de distância de edição maiores que 2.

Analisando o método proposto por este trabalho, todas as versões do 2-level comparadas com os *baselines* mostraram performance satisfatória para realização de complementação automática de sentenças, com médias de tempo comparáveis às médias de tempo dos principais métodos encontrados na literatura.

O fator mais crítico observado nos métodos experimentados é acerca da utilização de memória. Com o limite de memória disponível de 64 GB, o IncNGTrie pode ser testado em apenas uma base de dados, além de nenhum dos *baselines* experimentados ter sido capaz de indexar completamente a base de dados MEDLINE. Como abordado anteriormente, como sistemas de complementação automática precisam ficar indexados em memória para que o processamento seja rápido, o espaço que esses sistemas utilizam tende a ser limitado à quantidade de memória disponível. Sendo assim, a quantidade de memória que um método utiliza pode definir sua viabilidade para determinados contextos, como o tamanho da base dedados a ser indexada.

Os dados obtidos nestes experimentos mostram que, com o 2-level, que utiliza o conceito de busca em dois níveis, é possível realizar uma economia significativa de memória em relação ao método em que foi baseado, o ICAN, chegando a economizar cerca de 85%, 75% e 95% de memória para as bases AOL, USADDR e MEDLINE, respectivamente. Obtendo também uma alta economia de memória se comparado

com os demais métodos. Além disso, o método de dois níveis consegue não apenas manter uma performance de processamento de consultas satisfatória para o problema de complementação automática, mas obter um tempo competitivo em relação ao ICAN, dependendo do contexto. Portanto, é possível validar a hipótese deste trabalho, que é apresentar um método de complementação automática de dois níveis capaz de consumir uma quantidade de memória significativamente menor em comparação aos principais métodos presentes na literatura, mantendo um tempo de processamento de consultas aceitável.

Capítulo 5

Conclusões

Este trabalho apresentou um método para complementação automática de sentenças através de uma abordagem de processamento em dois níveis. O objetivo era realizar complementação automática consumindo uma quantidade de memória significativamente menor que os principais métodos dispostos na literatura, mantendo uma performance de tempo de processamento aceitável. Para isso, foi apresentado um método em dois níveis que combina busca em uma estrutura de índices com busca sequencial.

No primeiro nível, os dados são parcialmente indexados em uma estrutura de dados *trie*, utilizando um mecanismo de pesquisa textual tolerante a erros baseado no algoritmo ICAN, que é um método proposto por Ji and Li (2011). Neste trabalho, apenas um número n de caracteres de cada item é indexado no primeiro nível. Sendo assim, caso a *string* de uma consulta tenha um comprimento maior que n caracteres, os itens retornados pela estrutura representam um conjunto expandido com candidatos em potencial para serem mostrados na resposta, necessitando que seja realizado um outro procedimento para, dentre esses, definir as respostas de sugestões corretas.

O segundo nível realiza, quando necessário, o processamento dos resultados retornados pelo primeiro nível. É verificado, através de um algoritmo de busca sequencial, se cada resultado do primeiro nível possui uma semelhança com a consulta suficiente para ser considerado uma sugestão correta. O número de itens analisados neste nível do método tende a ser muito menor que o número de itens processados no primeiro nível.

Foram analisados diferentes tamanhos de prefixos indexados na *trie* para processamento no primeiro nível, podendo ser observado que o ganho de performance em tempo de processamento para prefixos de indexação no primeiro nível vai diminuindo na medida que este prefixo cresce. Sendo assim, foi possível perceber que, para cada contexto, é possível encontrar um ponto de equilíbrio entre performance de tempo de processamento e economia de memória entre os tamanhos de prefixos indexados na *trie*.

Os experimentos realizados no capítulo 4 mostram que, para limite de distância de edição 1, apesar de perder eficiência em tempo de processamento quando necessário à utilização do segundo nível, o método oferece uma performance muito abaixo do limite crítico de 100 ms, apontado na literatura em Alsultan and Warwick (2013), para a base de dados USADDR. Além disso, ao utilizar uma abordagem de busca em dois níveis, foi possível observar uma economia de cerca de 70% de utilização de memória em relação à indexação completa dos itens na *trie*.

Ao comparar com os *baselines*, foi possível notar que o método de dois níveis utiliza uma quantidade de memória consideravelmente inferior à utilizada nos demais métodos comparados. Comparando com o ICAN, o método de dois níveis pode economizar até 15% de memória para a base de dados AOL, até 75% para a base de dados USADDR e, ainda, consegue economizar até 95% de memória para a base de dados MEDLINE. Esses valores de economia de memória podem ser ainda maiores se comparados com outros métodos, como o caso do BEVA, onde o método de dois níveis consegue uma economia de até 92% na base de dados AOL, 85% na base de dados USADDR e podendo chegar a mais de 95% de economia na base de dados MEDLINE.

Para que o método de dois níveis possa economizar memória na indexação dos itens, espera-se uma diminuição de performance no tempo de processamento de consultas. Entretanto, a perda de performance só pode ser percebida para consultas com limite de edição 1 com consultas de comprimento superior ao tamanho do prefixo de indexação no primeiro nível. Nos demais casos, a performance média de tempo de processamento de consultas do método de dois níveis mostra-se equivalente, e em alguns casos superior,

ao ICAN, que possui a estrutura utilizada no primeiro nível do método de dois níveis.

De modo geral, os experimentos mostraram que a performance média em tempo de processamento do método de dois níveis é equivalente às dos principais métodos de complementação automática de sentenças existentes na literatura. Essa característica agregada à boa economia de memória proporcionada, torna válida a hipótese de que é possível reduzir a utilização de uma quantidade significativa de memória, mantendo um tempo de processamento de consultas aceitável através de um método de complementação automática baseado na abordagem de busca em dois níveis.

Como trabalhos futuros, sugere-se estudar outras versões do método de dois níveis, substituindo a estrutura utilizada no primeiro nível, podendo analisar o comportamento do método de dois níveis com a estrutura utilizada pelos demais métodos referenciados neste trabalho. Planeja-se também avançar em melhorias para o segundo nível do método, avaliando outras soluções em algoritmos mais eficientes para busca de proximidade e busca exata de textos. Há também a possibilidade de estudar formas mais compactas de representar uma *trie*, com intuito de melhorar ainda mais a relação entre a eficiência de texto e otimização de memória.

Referências Bibliográficas

Alsultan, A. and Warwick, K. (2013). Keystroke dynamics authentication: a survey of free-text methods. *International Journal of Computer Science Issues (IJCSI)*, 10(4):1.

Arasu, A., Ganti, V., and Kaushik, R. (2006). Efficient exact set-similarity joins. In *Proceedings of the 32nd international conference on Very large data bases*, pages 918–929. VLDB Endowment.

Baeza-Yates, R. and Ribeiro-Neto, B. (2013). *Recuperacao de Informacao-: Conceitos e Tecnologia das Maquinas de Busca*. Bookman Editora.

Bast, H. and Weber, I. (2006). Type less, find more: fast autocompletion search with a succinct index. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 364–371. ACM.

Chaudhuri, S., Ganti, V., and Kaushik, R. (2006). A primitive operator for similarity joins in data cleaning. In *Data Engineering, 2006. ICDE'06. proceedings of the 22nd International Conference on*, pages 5–5. IEEE.

Chaudhuri, S. and Kaushik, R. (2009). Extending autocompletion to tolerate errors. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 707–718. ACM.

Cucerzan, S. and Brill, E. (2004). Spelling correction as an iterative process that exploits the collective knowledge of web users. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*.

Das, S., Agrawal, D., and El Abbadi, A. (2010). G-store: a scalable data store for transactional multi key access in the cloud. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 163–174. ACM.

De La Briandais, R. (1959). File searching using variable length keys. In *Papers presented at the the March 3-5, 1959, western joint computer conference*, pages 295–298. ACM.

Deng, D., Li, G., Wen, H., Jagadish, H., and Feng, J. (2016). Meta: an efficient matching-based method for error-tolerant autocompletion. *Proceedings of the VLDB Endowment*, 9(10):828–839.

Fredkin, E. (1960). Trie memory. *Communications of the ACM*, 3(9):490–499.

Gomaa, W. H. and Fahmy, A. A. (2013). A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18.

Hall, P. A. and Dowling, G. R. (1980). Approximate string matching. *ACM computing surveys (CSUR)*, 12(4):381–402.

Ji, S. and Li, C. (2011). Location-based instant search. In *International Conference on Scientific and Statistical Database Management*, pages 17–36. Springer.

Ji, S., Li, G., Li, C., and Feng, J. (2009). Efficient interactive fuzzy keyword search. In *Proceedings of the 18th international conference on World wide web*, pages 371–380. ACM.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710.

Li, G., Ji, S., Li, C., and Feng, J. (2011). Efficient fuzzy full-text type-ahead search. *The VLDB Journal—The International Journal on Very Large Data Bases*, 20(4):617–640.

Liang, F. M. (1983). Word hy-phen-a-tion by com-put-er. Technical report, Calif. Univ. Stanford. Comput. Sci. Dept.

Manber, U., Wu, S., et al. (1994). Glimpse: A tool to search through entire file systems. In *Usenix Winter*, pages 23–32.

Marx, V. (2013). Biology: The big challenges of big data.

Mihov, S. and Schulz, K. U. (2004). Fast approximate search in large dictionaries. *Computational Linguistics*, 30(4):451–477.

Nandi, A. and Jagadish, H. (2007). Effective phrase prediction. In *Proceedings of the 33rd international conference on Very large data bases*, pages 219–230. VLDB Endowment.

Navarro, G., De Moura, E. S., Neubert, M., Ziviani, N., and Baeza-Yates, R. (2000). Adding compression to block addressing inverted indexes. *Information retrieval*, 3(1):49–77.

Oliveira, V. (2018). Estratégias de ordenação para completar consultas em e-commerce. Master's thesis.

Peterson, J. L. (1980). Computer programs for detecting and correcting spelling errors. *Communications of the ACM*, 23(12):676–687.

Szwarcfiter, J. L. and Markenzon, L. (1994). *Estruturas de Dados e seus Algoritmos*, volume 2. Livros Técnicos e Científicos.

Xiao, C., Qin, J., Wang, W., Ishikawa, Y., Tsuda, K., and Sadakane, K. (2013). Efficient error-tolerant query autocompletion. *Proceedings of the VLDB Endowment*, 6(6):373–384.

Xiao, C., Wang, W., and Lin, X. (2008). Ed-join: an efficient algorithm for similarity joins with edit distance constraints. *Proceedings of the VLDB Endowment*, 1(1):933–944.

Xu, P. and Lu, J. (2017). Top-k string auto-completion with synonyms. In *International Conference on Database Systems for Advanced Applications*, pages 202–218. Springer.

Yu, M., Li, G., Deng, D., and Feng, J. (2016). String similarity search and join: a survey. *Frontiers of Computer Science*, 10(3):399–417.

Zhou, X., Qin, J., Xiao, C., Wang, W., Lin, X., and Ishikawa, Y. (2016). Beva: An efficient query processing algorithm for error-tolerant autocompletion. *ACM Transactions on Database Systems (TODS)*, 41(1):5.