Implementação e Análise Comparativa de Algoritmos de Distância de Edição de Árvores: Zhang-Shasha versus Levenshtein simplificado

Ana Fernanda Souza Cançado, Arthur de Sá Braz de Matos,
Gabriel Praes B Nunes, Guilherme Otávio de Oliveira,
Júlia Pinheiro Roque

06/2025

Resumo

Este trabalho apresenta a implementação e a análise comparativa de dois algoritmos para o cálculo da distância de edição entre árvores: o algoritmo clássico de Zhang-Shasha e uma versão simplificada baseada em programação dinâmica pós-ordem, inspirada na distância de Levenshtein. Ambos os algoritmos buscam determinar o custo mínimo de transformação de uma árvore de origem em uma árvore de destino por meio de operações básicas: inserção, deleção e substituição de nós. O algoritmo de Zhang-Shasha considera a estrutura hierárquica completa das árvores, utilizando uma abordagem eficiente baseada em pós-ordem e floresta, enquanto o algoritmo simplificado trata as árvores como listas lineares de nós em pós-ordem, ignorando a hierarquia. Este trabalho contempla uma análise teórica das complexidades, implementação em C++17 e avaliação experimental comparativa entre a precisão e o desempenho dos dois métodos.

Palavras-chave: distância de edição, árvores, programação dinâmica, Zhang-Shasha, complexidade computacional.

1 Introdução

A distância de edição entre árvores (Tree Edit Distance – TED) é uma métrica fundamental para quantificar a diferença entre estruturas hierárquicas. Ela representa o custo mínimo necessário para transformar uma árvore em outra por meio de operações básicas como inserção, deleção e substituição de nós. Esse conceito tem aplicações amplas em áreas como bioinformática, onde é utilizado para comparar estruturas de RNA, no processamento de linguagem natural, na comparação de árvores sintática, entre outros.

Dentre os algoritmos propostos para o cálculo da TED, o algoritmo de [1] é um dos mais reconhecidos pela sua eficiência e precisão em árvores ordenadas. Esse algoritmo utiliza uma abordagem baseada em programação dinâmica e percorre as árvores em pós-ordem, considerando as operações de edição em subárvores parciais chamadas de florestas. Ele introduz o conceito de nós-chave (keyroots) para reduzir o número de subproblemas, permitindo um cálculo mais eficiente da TED.

Por outro lado, é comum o uso de algoritmos simplificados, que ignoram parte da estrutura hierárquica da árvore. Neste trabalho, implementa-se uma versão baseada em

programação dinâmica clássica, semelhante ao algoritmo de [3], aplicado sobre os nós das árvores em ordem pós-fixada. Essa abordagem trata as árvores como listas lineares de rótulos e calcula a distância de edição entre elas sem considerar a relação estrutural entre pais e filhos, resultando em um algoritmo mais simples, com menor complexidade computacional, porém com perda significativa de precisão semântica.

Neste artigo, propomos a implementação e análise comparativa entre o algoritmo completo de Zhang-Shasha e essa versão simplificada baseada em listas pós-ordem. Avaliamos as diferenças em termos de desempenho, complexidade computacional e qualidade dos resultados, destacando os limites de aproximações lineares em contextos que exigem sensibilidade à estrutura.

2 Algoritmo de Zhang-Shasha

2.1 Descrição Teórica

O Algoritmo de Zhang-Shasha, proposto por Kaizhong Zhang e Dennis Shasha em 1989 [1], introduz uma abordagem inovadora que melhora substancialmente a eficiência computacional em relação aos métodos anteriores, particularmente o algoritmo de Tai [2].

A principal contribuição do algoritmo é a redução da complexidade de tempo para:

$$O(|T_1| \times |T_2| \times \min(\text{depth}(T_1), \text{leaves}(T_1)) \times \min(\text{depth}(T_2), \text{leaves}(T_2)))$$

onde $|T_i|$ representa o número de nós da árvore T_i, $\text{depth}(T_i)$ é a profundidade da árvore, e $\text{leaves}(T_i)$ é o número de folhas.

2.2 Conceitos Fundamentais

O algoritmo de Zhang-Shasha baseia-se em dois conceitos-chave que permitem uma decomposição eficiente do problema:

2.2.1 Leftmost Leaf (Folha Mais à Esquerda)

Para cada nó v em uma árvore ordenada, define-se $l(v)$ como a folha mais à esquerda na subárvore enraizada em v. Formalmente:

$$l(v) = \begin{cases} v & \text{se } v \text{ é folha} \\ l(\text{primeiro filho de } v) & \text{caso contrário} \end{cases}$$

2.2.2 Keyroot

Um nó v é denominado *keyroot* se e somente se não existe outro nó w tal que $l(v) = l(w)$ e w é ancestral próprio de v. Em outras palavras, um keyroot é um nó que possui uma folha mais à esquerda única em relação aos seus ancestrais.

Propriedade importante: O conjunto de keyrots de uma árvore tem cardinalidade igual ao número de folhas da árvore.

2.3 Operações de Edição

O algoritmo considera três operações básicas de edição:

- **Inserção:** Adicionar um nó à árvore (custo γ)
- **Remoção:** Remover um nó da árvore (custo γ)
- **Substituição:** Alterar o rótulo de um nó (custo $\delta(a, b)$ para substituir rótulo a por b)

2.4 Metodologia

O algoritmo de Zhang-Shasha opera através de um processo estruturado em quatro fases principais:

2.4.1 Fase 1: Pré-processamento

1. Calcular a numeração pós-ordem dos nós de ambas as árvores
2. Determinar $l(v)$ para cada nó v
3. Identificar o conjunto de keyrots para cada árvore
4. Construir mapeamentos entre índices e nós

2.4.2 Fase 2: Decomposição por Keyrots

Para cada par de keyrots (kr_1, kr_2), onde $kr_1 \in T_1$ e $kr_2 \in T_2$, o algoritmo resolve o subproblema de calcular a distância entre as subárvores enraizadas nesses keyrots.

2.4.3 Fase 3: Programação Dinâmica

Para cada subproblema, utiliza-se uma matriz de programação dinâmica $M[i][j]$ onde:

$$M[i][j] = \min \begin{cases} M[i-1][j] + \gamma & \text{(remoção)} \\ M[i][j-1] + \gamma & \text{(inserção)} \\ M[i-1][j-1] + \delta(T_1[i], T_2[j]) & \text{(substituição)} \end{cases}$$

2.4.4 Fase 4: Combinação de Resultados

Os resultados dos subproblemas são combinados para obter a distância final entre as árvores completas.

2.5 Complexidade

Complexidade de Tempo: $O(|T_1| \times |T_2| \times \min(\text{depth}(T_1), \text{leaves}(T_1)) \times \min(\text{depth}(T_2), \text{leaves}(T_2)))$
Complexidade de Espaço: $O(|T_1| \times |T_2|)$
Análise de Casos:

- **Melhor caso:** Árvores balanceadas com poucas folhas
- **Pior caso:** Árvores degeneradas (semelhantes a listas)

2.6 Implementação

A implementação prática do algoritmo utiliza a classe template `ZhangShashaCalculator<T>`. A classe mantém vetores com os nós das árvores em ordem pós-ordem, informações sobre folhas mais à esquerda e keyroots, além de duas matrizes bidimensionais (treedist e forestdist) para implementar a programação dinâmica. O sistema de custos é uniforme: inserção e deleção têm custo unitário (1), enquanto substituição tem custo zero para nós idênticos e unitário para diferentes. O algoritmo principal segue a estrutura teórica, executando pré-processamento, cálculo de folhas mais à esquerda, identificação de keyroots e programação dinâmica através de iteração sobre pares de keyroots.

3 Algoritmo adaptado de Levenshtein:

3.1 Descrição Teórica

O algoritmo de Tree Edit Distance (TED) representa uma extensão natural do algoritmo de Levenshtein para árvores, inspirando-se diretamente nos princípios de distância de edição para sequências [3]. Assim como o algoritmo de Levenshtein calcula a distância mínima entre strings através de operações de inserção, remoção e substituição, o TED aplica esses mesmos conceitos a árvores ordenadas, utilizando programação dinâmica para determinar o número mínimo de operações necessárias para transformar uma árvore em outra.

A complexidade temporal do algoritmo é $O(|T_1| \times |T_2|)$, onde $|T_1|$ e $|T_2|$ representam o número de nós das árvores de entrada. Esta abordagem direta oferece uma solução eficiente para problemas de comparação de estruturas arbóreas de tamanho moderado.

3.1.1 Representação Pós-Ordem

As árvores são representadas através de seus nós em ordem pós-ordem, criando uma sequência linear que preserva as relações estruturais da árvore original. Esta representação permite a aplicação direta do algoritmo de Levenshtein, adaptado para considerar as características específicas das estruturas arbóreas.

3.1.2 Mapeamento de Operações

Cada operação de edição na árvore corresponde diretamente às operações clássicas do algoritmo de Levenshtein, adaptadas para o contexto arbóreo:

- **Inserção:** Adicionar um elemento à sequência
- **Remoção:** Remover um elemento da sequência
- **Substituição:** Alterar um elemento da sequência

3.2 Operações de Edição

O algoritmo considera três operações básicas de edição com custos associados:

- **Inserção:** Adicionar um nó à árvore (custo c_i)
- **Remoção:** Remover um nó da árvore (custo c_d)

- **Substituição:** Alterar o rótulo de um nó (custo $c_s(a, b)$ para substituir rótulo a por b)

3.3 Metodologia

O algoritmo TED opera através de um processo estruturado em três fases principais:

3.3.1 Fase 1: Pré-processamento

1. Obter a representação pós-ordem de ambas as árvores
2. Determinar as dimensões da matriz de programação dinâmica
3. Inicializar as estruturas de dados necessárias

3.3.2 Fase 2: Inicialização da Matriz

A matriz de programação dinâmica $D[i][j]$ é inicializada com os casos base:

- $D[0][j] = \sum_{k=1}^{j} c_i(T_2[k])$ (inserir todos os nós de T_2)
- $D[i][0] = \sum_{k=1}^{i} c_d(T_1[k])$ (remover todos os nós de T_1)

3.3.3 Fase 3: Programação Dinâmica

Para cada posição (i, j) da matriz, calcula-se:

$$D[i][j] = \min \begin{cases} D[i-1][j] + c_d(T_1[i]) & \text{(remoção)} \\ D[i][j-1] + c_i(T_2[j]) & \text{(inserção)} \\ D[i-1][j-1] + c_s(T_1[i], T_2[j]) & \text{(substituição)} \end{cases}$$

3.4 Complexidade

Complexidade de Tempo: $O(|T_1| \times |T_2|)$
Complexidade de Espaço: $O(|T_1| \times |T_2|)$
Análise de Casos:

- **Melhor caso:** Árvores idênticas ou com alta similaridade estrutural
- **Pior caso:** Árvores completamente diferentes sem elementos em comum

3.5 Implementação

A implementação prática do algoritmo de Tree Edit Distance utiliza uma função template `TED<T>` que calcula a distância de edição entre duas árvores. A implementação adota programação dinâmica clássica com uma matriz bidimensional para resolver o problema de forma eficiente.

A função principal utiliza a representação pós-ordem dos nós e implementa a recorrência através de uma matriz $(n + 1) \times (m + 1)$, onde o resultado final é obtido da posição (n, m).

4 Análise de Escalabilidade

4.1 Metodologia dos Testes

Para avaliar o desempenho computacional dos algoritmos implementados, foram realizados testes de escalabilidade utilizando árvores de tamanhos crescentes, variando de 10 a 1600 nós.

4.2 Resultados Experimentais

Os resultados dos testes de escalabilidade são apresentados nas tabelas abaixo, mostrando o comportamento temporal de ambos os algoritmos:

4.2.1 Algoritmo TED (Tree Edit Distance)

Tamanho das Árvores	Distância Calculada	Tempo (s)
10	8	$8{,}00 \times 10^{-6}$
50	35	$6{,}31 \times 10^{-5}$
100	58	$2{,}84 \times 10^{-4}$
200	77	$1{,}17 \times 10^{-3}$
400	117	$5{,}35 \times 10^{-3}$
800	197	$2{,}21 \times 10^{-2}$
1600	357	$8{,}90 \times 10^{-2}$

Tabela 1: Desempenho temporal do algoritmo adaptado de Levenshtein

4.2.2 Algoritmo de Zhang-Shasha

Tamanho das Árvores	Distância Calculada	Tempo (s)
10	9	$3{,}38 \times 10^{-5}$
50	43	$6{,}95 \times 10^{-4}$
100	75	$4{,}70 \times 10^{-3}$
200	96	$2{,}93 \times 10^{-2}$
400	136	$1{,}29 \times 10^{-1}$
800	216	$5{,}93 \times 10^{-1}$
1600	376	2,47

Tabela 2: Desempenho temporal do algoritmo de Zhang-Shasha

4.3 Análise Comparativa

4.3.1 Crescimento Temporal

A análise dos resultados experimentais revela diferenças significativas no comportamento de escalabilidade dos dois algoritmos:

Algoritmo adaptado de Levenshtein: Demonstra melhor escalabilidade, com crescimento temporal mais controlado. O tempo aumenta de 8,00 $\times$ 10^{-6}s para 10 nós até 8,90 $\times$ 10^{-2}s para 1600 nós, um aumento de aproximadamente 11.125 vezes.

Algoritmo de Zhang-Shasha: Apresenta crescimento temporal aproximadamente quadrático, consistente com sua complexidade teórica em casos específicos. O tempo de execução aumenta de 3,38 $\times$ 10^{-5}s para 10 nós até 2,47s para 1600 nós, representando um aumento de aproximadamente 73.000 vezes.

4.3.2 Comportamento Assintótico

Os resultados experimentais confirmam as complexidades teóricas dos algoritmos:

Algoritmo adaptado de Levenshtein: O comportamento mais suave confirma simplicidade do algoritmo. A otimização através da abordagem direta de programação dinâmica resulta em ganhos substanciais de performance.

Zhang-Shasha: O comportamento em casos específicos pode apresentar crescimento maior devido à complexidade adicional dos cálculos de keyroots e decomposições, especialmente para certas estruturas de árvore.

4.4 Considerações sobre Complexidade Espacial

A complexidade espacial dos algoritmos segue padrões distintos:

Algoritmo adaptado de Levenshtein: Utiliza uma matriz de programação dinâmica de tamanho $n \times m$, onde n e m são os números de nós das duas árvores. Isso resulta em uma complexidade espacial de $O(n \times m)$, pois é necessário armazenar os custos parciais de edição para todos os pares possíveis de subárvores.

Zhang-Shasha: Emprega múltiplas estruturas de dados, como as matrizes `treedist` e `forestdist`, para calcular recursivamente as distâncias entre subárvores e florestas. Apesar de inicialmente também requerer espaço $O(n \times m)$, o algoritmo introduz otimizações que permitem reutilizar partes dessas matrizes em diferentes fases da computação. Com isso, é possível reduzir significativamente a memória ativa utilizada, sobretudo ao explorar o fato de que muitas subestruturas são reutilizadas e que não é necessário manter todos os valores simultaneamente em memória. Essas otimizações tornam a implementação mais eficiente em termos de espaço, especialmente em comparação com abordagens de programação dinâmica ingênuas.

5 Casos de Teste e Resultados Experimentais

Nesta seção, apresentamos os resultados dos testes realizados com dois algoritmos distintos para cálculo da distância de edição entre árvores.

5.1 Testes Funcionais Simples

Foram realizados testes com casos simples e esperados, visando validar a correção de ambos os algoritmos:

- **Árvores Idênticas**: Espera-se distância zero. Ambos os algoritmos retornaram 0.
- **Apenas Raiz**: Espera-se distância 1 (substituição da raiz). Ambos retornaram 1.

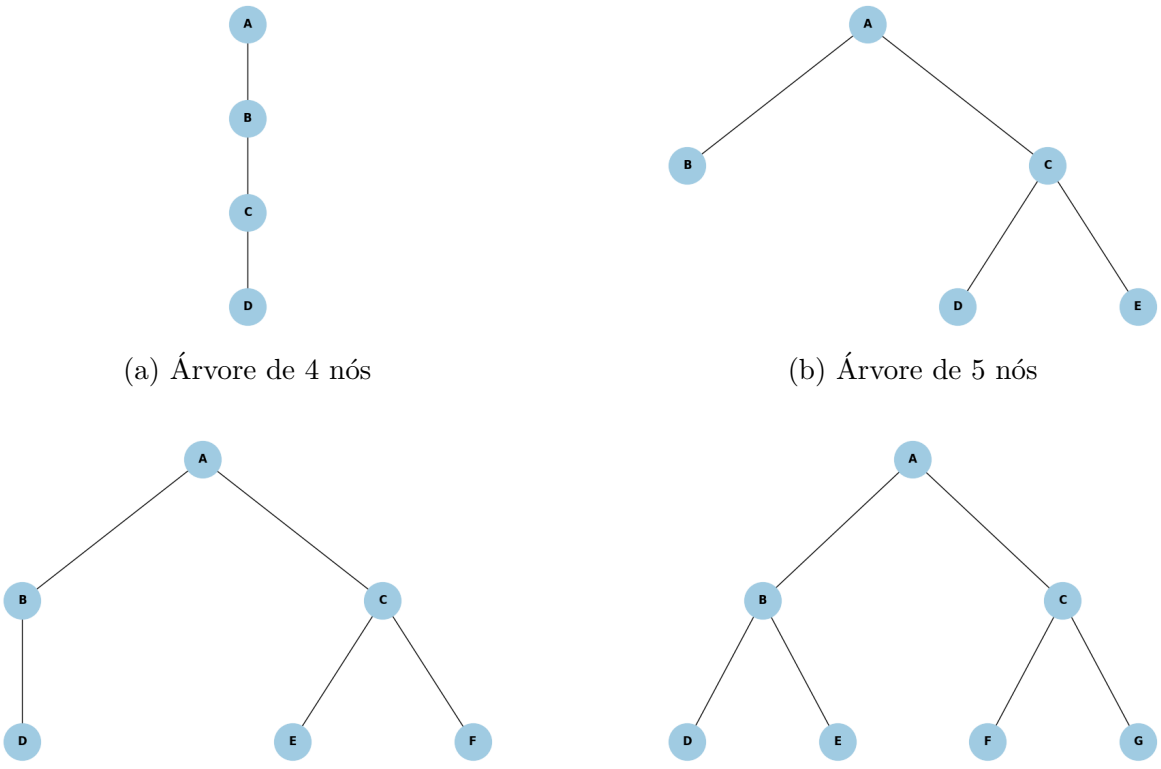
- **String Linear**: Estruturas similares com pequenas diferenças de ramos. Ambos os algoritmos retornaram 1.

Esses testes confirmam a consistência e correção funcional de ambas as abordagens em casos base.

5.2 Testes com Estruturas Pequenas e Assimétricas

Foram aplicados testes adicionais em árvores pequenas com topologias distintas para avaliar a sensibilidade à forma estrutural:

- **Árvores com 4 e 5 vértices**: Uma árvore em forma de ramo profundo e outra assimétrica. Ambos algoritmos retornaram distância 3.
- **Árvores com 6 e 7 vértices**: Uma árvore balanceada em largura e outra binária completa. Ambos algoritmos retornaram distância 3.



(a) Árvore de 4 nós

(b) Árvore de 5 nós

(c) Árvore de 6 nós

(d) Árvore de 7 nós

Figura 1: Conjuntos de árvores utilizadas nos testes: 4 - 5 nós e 6 - 7 nós.

Esses testes reforçam que os dois algoritmos avaliam diferenças estruturais de forma coerente.

6 Divisão de Responsabilidades

- **Ana Fernanda:** Implementação do algoritmo 1 e escrita do artigo.

- **Arthur Matos:** Implementação do algoritmo 2 e execução dos testes.
- **Gabriel Praes:** Implementação do algoritmo 2 e escrita do artigo.
- **Guilherme Otávio:** Implementação do algoritmo 1 e escrita do artigo.
- **Júlia Pinheiro:** Implementação do algoritmo 1 e execução dos testes.

7 Conclusões

Este trabalho realizou uma análise comparativa entre dois algoritmos para cálculo da distância de edição entre árvores: o algoritmo clássico de Zhang-Shasha e uma versão simplificada baseada em programação dinâmica pós-ordem, inspirada no algoritmo de Levenshtein.

Os testes funcionais demonstraram que ambos os algoritmos são corretos, retornando os valores esperados para casos triviais e pequenas variações estruturais. No entanto, os testes de escalabilidade revelaram diferenças significativas de desempenho.

O algoritmo baseado em Levenshtein, por sua simplicidade estrutural e abordagem linearizada, demonstrou eficiência superior em tempo de execução, principalmente à medida que o tamanho das árvores aumentava. Por outro lado, o algoritmo de Zhang-Shasha, embora mais lento, oferece uma análise mais sensível à estrutura hierárquica das árvores, sendo mais apropriado para aplicações que exigem precisão semântica elevada, como comparação de árvores sintáticas ou estruturas biológicas complexas.

Então, o algoritmo baseado em Levenshtein é altamente vantajoso em cenários que demandam alto desempenho e escalabilidade, mesmo com uma perda controlada de sensibilidade estrutural.

O algoritmo Zhang-Shasha é mais indicado para situações em que a preservação da estrutura hierárquica é essencial, ainda que com maior custo computacional.

Ambas as abordagens possuem mérito dependendo do contexto de aplicação, e os resultados obtidos reforçam a importância de balancear precisão e desempenho na escolha do algoritmo. Futuras extensões podem explorar variantes híbridas ou otimizações específicas voltadas a estruturas de árvores particulares, ampliando ainda mais as possibilidades de aplicação desses métodos.

Referências

[1] Zhang, K., & Shasha, D. (1989). *Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems*. SIAM Journal on Computing, 18(6), 1245-1262.

[2] Tai, K. C. (1979). *The Tree-to-Tree Correction Problem*. Journal of the ACM, 26(3), 422-433.

[3] Levenshtein, V.I. (1966) Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Soviet Physics Doklady, 10, 707-710.