Comparação Experimental dos Algoritmos de Zhang-Shasha e Klein para a Distância de Edição de Árvores

Arthur L. Setragni [**Pontifícia Universidade Católica de Minas Gerais**]
Guilherme H. Nascimento [**Pontifícia Universidade Católica de Minas Gerais**]
Gustavo G. Macedo [**Pontifícia Universidade Católica de Minas Gerais**]
Igor S. Paschoalino [**Pontifícia Universidade Católica de Minas Gerais**]
Pedro A. Souza [**Pontifícia Universidade Católica de Minas Gerais**]
Rafael V. Clark [**Pontifícia Universidade Católica de Minas Gerais**]
Pontifícia Universidade Católica de Minas Gerais (PUCMG)
Caixa Postal 1.686 – 30535-901 – Minas Gerais – MG – Brazil.

Abstract. This report presents the implementation and experimental analysis of two algorithms for computing the Tree Edit Distance (TED): the classic algorithm by Zhang and Shasha (1989) and a more recent, asymptotically faster algorithm by Philip Klein (1998). The study focuses on comparing their computational costs, both in time and space, across different types of rooted ordered trees. The experiments were designed to validate the theoretical claims, testing scenarios that favor each algorithm's strengths. The results show that while Klein's algorithm provides better worst-case guarantees, Zhang and Shasha's method can be more efficient for specific tree structures, highlighting the trade-offs between the two approaches.

Keywords: Tree Edit Distance, Zhang-Shasha Algorithm, Klein's Algorithm, Algorithm Analysis, Experimental Comparison.

Resumo. Este relatório apresenta a implementação e a análise experimental de dois algoritmos para o cálculo da Distância de Edição de Árvores (TED): o algoritmo clássico de Zhang e Shasha (1989) e um algoritmo mais recente e assintoticamente mais rápido de Philip Klein (1998). O estudo foca na comparação de seus custos computacionais, tanto de tempo quanto de espaço, em diferentes tipos de árvores ordenadas e enraizadas. Os experimentos foram projetados para validar as alegações teóricas, testando cenários que favorecem os pontos fortes de cada algoritmo. Os resultados mostram que, enquanto o algoritmo de Klein oferece melhores garantias de pior caso, o método de Zhang e Shasha pode ser mais eficiente para estruturas de árvores específicas, evidenciando os trade-offs entre as duas abordagens.

Keywords: Distância de Edição de Árvores, Algoritmo de Zhang-Shasha, Algoritmo de Klein, Análise de Algoritmos, Comparação Experimental.

1 Introdução

A distância de edição de árvores (TED, do inglês *Tree Edit Distance*) é uma métrica que define o custo mínimo para transformar uma árvore T_1 em outra árvore T_2 através de um conjunto de operações elementares: inserção, exclusão e substituição de nós. Esta noção intuitiva de distância encontrou aplicações em diversas áreas, como a comparação de estruturas secundárias de RNA em biologia, análise de documentos XML e análise de hierarquias em visão computacional.

O cálculo eficiente da TED é um problema desafiador. O algoritmo proposto por Zhang e Shasha em 1989 tornou-se um padrão na área [1]. Anos depois, Philip N. Klein (1998) propôs um novo algoritmo com uma complexidade de pior caso assintoticamente melhor, especialmente para o caso de árvores não-enraizadas [2].

Este trabalho tem como objetivo investigar se as vantagens teóricas do algoritmo de Klein se traduzem em melhorias de desempenho empíricas (tempo e espaço) em cenários práticos. Para isso, implementamos ambos os algoritmos e conduzimos uma análise experimental comparativa, focando no caso de árvores ordenadas e enraizadas.

2 Metodologia

Nesta seção, descrevemos a estrutura de dados utilizada, as principais características de cada algoritmo implementado e o planejamento dos experimentos realizados.

2.1 Estrutura de Dados e Implementação

Foram criadas funções adaptadoras que geram as representações específicas necessárias para cada algoritmo:

- **Para o algoritmo de Klein:** Foi implementada uma função que percorre a árvore interna e gera a sua **string de Euler** correspondente. O algoritmo de Klein opera diretamente sobre essa representação [2].
- **Para o algoritmo de Zhang e Shasha:** O método opera de forma natural sobre a estrutura de árvore, apli-

cando travessias (pós-ordem) para identificar seus componentes chave, sem a necessidade de uma conversão explícita de formato.

Ambos os algoritmos foram implementados em C++, conforme solicitado no enunciado do trabalho.

2.2 Algoritmo de Zhang e Shasha (1989)

O algoritmo de Zhang e Shasha (Z&S) é um método de programação dinâmica que calcula a TED entre duas árvores ordenadas e enraizadas. Sua estratégia se baseia na decomposição das árvores em ”caminhos da esquerda” (*leftmost paths*) e na identificação de ”keyroots”, que são nós especiais que definem os subproblemas da DP [1].

A complexidade do algoritmo é $O(|A| \cdot |B| \cdot \text{LR\_colldepth}(A) \cdot \text{LR\_colldepth}(B))$, onde $\text{LR\_colldepth}(T)$ é a profundidade colapsada de uma árvore T. Em seu pior caso, essa complexidade pode atingir $O(n^4)$, onde n é o tamanho total das árvores [1].

2.3 Algoritmo de Klein (1998)

O algoritmo de Klein também utiliza programação dinâmica, mas introduz duas inovações cruciais para melhorar a complexidade do pior caso [2]:

1. **Representação em String de Euler:** A árvore é representada como uma string de Euler, onde cada aresta é percorrida duas vezes (ida e volta), criando um par de ”dardos” análogos a parênteses.
2. **Decomposição em Caminhos Pesados:** Em vez da decomposição em caminhos da esquerda, Klein utiliza a **decomposição em caminhos pesados** (*heavy-path decomposition*) em uma das árvores. Esta técnica garante que a profundidade colapsada da decomposição seja no máximo logarítmica em relação ao tamanho da árvore [3].

Essas otimizações reduzem a complexidade de pior caso para $O(|A|^2|B|\log|B|)$, ou $O(n^3 \log n)$ [2].

2.4 Planejamento Experimental

Para avaliar os custos computacionais de tempo e espaço de ambos os algoritmos, foram definidos três cenários de teste:

- **Cenário 1: Árvores ”Longas e Finas”.** Árvores com grande profundidade e poucas ramificações, projetadas para explorar o pior caso do Z&S.
- **Cenário 2: Árvores ”Largas e Rasas”.** Árvores com alto fator de ramificação e baixa profundidade, para investigar cenários onde a menor sobrecarga de Z&S pode ser vantajosa.
- **Cenário 3: Caso Médio.** Árvores geradas aleatoriamente para observar o comportamento médio.

O tempo de execução foi medido cronometrando a função principal de cada algoritmo.

3 Implementação

Nesta seção, descrevemos como cada algoritmo foi implementado.

3.1 Algoritmo de Zhang e Shasha (1989)

O algoritmo implementado busca encontrar a distância mínima de edição entre duas árvores rotuladas e ordenadas com caminhamento pós ordem, através do uso de programação dinâmica. O ponto chave de tal abordagem é o vetor de ”keyroots”, chamado `LR_keyroots`, que será elaborado em breve. Utilizando-o em conjunto com uma tabela `TD` de programação dinâmica, é possível calcular a distância entre subarvores para então encontrar a distância total. Foram utilizados os seguintes valores e estruturas de dados:

- **Estrutura de Nó (`Node`)**: Representa um nó na árvore. Contém um índice de ordenamento (`index`), um rótulo (`nome`), um ponteiro para o pai do nó (`pai`), um ponteiro para o descendente mais à esquerda (`mais_esq`) e um vetor de filhos do nó (`children`).
- **Estrutura da árvore (`Tree`)**: Representa a árvore como um todo. Possui um nó raiz (`root`), um vetor de ”keyroots” (`LR_keyroots`), um vetor de descendentes mais à esquerda de cada nó (`l`) e um vetor com os rótulos de todos os nós (`nomes`).
- **Vetor de rótulos (`nomes[]`)**: Um vetor que guarda o rótulo de cada nó em sua respectiva posição de ordenamento.
- **Vetor de esquerdos (`l[]`)**: Vetor que armazena o `index` do descendente mais a esquerda de cada nó em sua respectiva posição de ordenamento. Se o nó n é folha, $\texttt{l}[n->index] = n$.
- **Vetor de ”keyroots” (`LR_keyroots[]`)**: Pode ser definido para uma árvore T como: $\texttt{LR\_keyroots}(T) = \{k \mid \text{não existe } k > k \text{ tal que } l(k) = l(k)\}$. Isto é, se k está presente em `LR_keyroots`, então ou k é a raiz da árvore ou $l[k->index] \neq l[k->pai->index]$ [1]. No que isso resulta, é que cada ”keyroot” corresponde a raiz de uma subarvore que será processada a iterativamente.
- **Tabela de distância entre árvores (`TD[][]`)**: Um vetor 2D que é progressivamente preenchido com a distância final obtida a cada iteração de `treedist`. `TD[i][j]` armazena o custo mínimo para transformar uma árvore na outra.
- **Tabela de distância entre florestas (`forestdist[][]`)**: Um vetor 2D temporário utilizado para encontrar a distância entre duas florestas. `forestdist[i][j]` armazena o custo mínimo para transformar uma floresta na outra.

O algoritmo consiste em um método principal `TED(Tree *t1,Tree *t2)`, que recebe as duas árvores para cálculo da distância, e um método auxiliar `treedist(vector<int> l1, vector<int> l2, int i, int j, Tree *t1,Tree *t2)` que retorna a distância mínima entre duas florestas presentes nas árvores.

No método `TED`,primeiro todos os vetores das árvores são devidamente preenchidos e seus nós indexados, depois, para

cada ”keyroot” de `t1`, são percorridas todas as ”keyroots” de `t2`, em cada iteração é feita uma chamada para `treedist`, passando as ”keyroots” atuais de `t1` e `t2` como os parâmetros `i` e `j` e os vetores `l` de cada árvore como `l1` e `l2`, respectivamente. Quando a repetição é finalizada, o método retorna o valor de `TD[i][j]`.

O método `treedist` começa inicializando `forestdist`, e automaticamente preenchendo os custos para inserção e deleção. Depois, uma iteração dupla é iniciada, para cada instância de `l1[i - 1]` até `i` é feita uma repetição de `l2[j - 1]` até `j`. Em cada iteração, o algoritmo seleciona o valor mínimo nas redondezas da tabela quando somado ao devido custo de operação, e o salva na posição atual de `forestdist`. Quando as repetições são concluídas, o método retorna `forestdist[i][j]`, como custo mínimo de edição.

3.2 Algoritmo de Klein (1998)

A implementação do algoritmo de distância de edição de árvores utilizada neste estudo calcula o custo mínimo para transformar uma árvore ordenada e enraizada em outra. A abordagem é baseada em uma decomposição recursiva, onde o problema de comparar duas árvores é reduzido ao problema de comparar suas florestas de filhos, que por sua vez é resolvido com programação dinâmica. Foram utilizados os seguintes valores e estruturas de dados:

- **Estrutura de Nó (`No`)**: Representa um nó na árvore. Contém um rótulo (`valor`), um ponteiro para o primeiro filho (`filho`) e um ponteiro para seu próximo irmão (`proximoIrmao`). Esta é uma representação *primeiro-filho, próximo-irmão*.
- **Strings de Entrada (`str1, str2`)**: As duas árvores são fornecidas como entrada no formato de strings parentizadas, como `"A(B()C())"`. Uma função de parsing (`transformaEmArvore`) converte essas strings para a estrutura de nós em memória.
- **Raízes das Árvores (`raiz1, raiz2`)**: Ponteiros para os nós raiz das duas árvores a serem comparadas, após o parsing das strings.
- **Tabela de Programação Dinâmica (`tabelaProgDinamica[][]`)**: Uma matriz 2D utilizada na função `distanciaFlorestas` para calcular o custo de edição entre duas florestas (listas de árvores irmãs). `tabelaProgDinamica[i][j]` armazena o custo para transformar as ‘i‘ primeiras árvores da primeira floresta nas ‘j‘ primeiras da segunda.

O algoritmo é partido em três funções principais que se inter-relacionam: uma função de entrada, uma função recursiva para árvores e uma função de programação dinâmica para florestas.

A função de entrada, `distanciaStringEuler`, orquestra todo o processo: ela recebe as duas árvores como strings, converte-as para a estrutura de dados interna, invoca o cálculo principal e gerencia a memória, retornando ao final o valor da distância de edição. O cálculo central é delegado à função `distanciaDeEdicaoRecursiva`, que recebe os nós de duas (sub)árvores e determina o custo mínimo para transformá-las, considerando as operações de substituição, deleção e inserção. Para comparar os conjuntos de filhos de cada nó, esta função, por sua vez, utiliza a `distanciaFlorestas`, uma rotina especializada que, por meio de programação dinâmica, calcula e retorna o custo ótimo para converter uma sequência de árvores (floresta) em outra.

4 Resultados e Discussão

Nesta seção, apresentamos os resultados obtidos ao comparar nossa implementação, baseada no algoritmo de Philip Klein (1998), com uma implementação do algoritmo clássico de Zhang e Shasha (1989), aqui denominado ”Sasha”. O algoritmo de Klein foi escolhido como base para nosso desenvolvimento devido à sua superioridade teórica em cenários de pior caso, com a expectativa de que nossa implementação demonstrasse uma performance competitiva.

Contudo, a análise dos dados revela duas observações críticas que desafiaram nossas expectativas:

Disparidade de Desempenho: A implementação do algoritmo de Sasha apresentou tempos de execução na escala de milissegundos (ms), enquanto nossa implementação baseada em Klein operou na escala de segundos. Isso indica uma performance drasticamente superior do algoritmo de Sasha em todos os testes realizados. Diferença nos Resultados: Foram notadas pequenas diferenças na ”Distância” final calculada pelos algoritmos para as árvores dos tipos ”Larga” e ”Média”, o que pode sugerir variações entre as implementações ou nos objetivos dos algoritmos testados.

4.1 Cenário 1: Árvores ”Longas e Finas”

Neste cenário, que teoricamente se aproximaria do pior caso para algoritmos mais simples como o de Zhang e Shasha, os resultados experimentais contradizem o que esperávamos da nossa implementação. O algoritmo ”Sasha” demonstrou uma performance vastamente superior em árvores com alta profundidade e baixa ramificação.

- No primeiro teste, para a árvore representada por `A(B(C(D(E(F(G(H(I(J(K(L(M(N(O(P()))))))))))))))))`, o algoritmo ”Sasha” concluiu em **0.3046 milissegundos**, enquanto nossa implementação de Klein levou **4.513 segundos**.
- De forma similar, para a árvore `Z(Y(X(W(V(U(T(S(R(Q(P(O(N(M(L(K(J(I()))))))))))))))))))`, a diferença foi ainda maior, com ”Sasha” registrando **0.6192 milissegundos** contra **8.343 segundos** do nosso algoritmo.

A superioridade teórica do algoritmo de Klein não se traduziu em performance na nossa implementação para estas instâncias.

4.2 Cenário 2: Árvores ”Largas e Rasas”

Para árvores com baixa profundidade e alta ramificação, a vantagem do algoritmo de Sasha sobre nossa implementação permaneceu esmagadora.

- Para a árvore com representação de Euler `A(B()C()D()E()F()G()H()I()J()K()L()M()N()O()P()Q())`, ”Sasha” levou **3.3553 ms**, em comparação com os **107 ms** (0.107 segundos) do nosso algoritmo.
- Em uma segunda árvore larga e mais complexa, `A(B(E()F()G()H()I())C(J()K()L()M()N())D(O()P()Q()R()S()))`, nosso algoritmo precisou de **63 ms**, enquanto ”Sasha” demorou apenas **2.7989 ms**.

A magnitude da diferença de tempo sugere que, para estas instâncias, a sobrecarga computacional da nossa implementação do algoritmo de Klein é significativamente maior do que a do algoritmo de Sasha.

4.3 Cenário 3: Caso Médio

No cenário com uma árvore de estrutura intermediária, o padrão de desempenho se manteve. O algoritmo de Sasha finalizou em 1.1245 ms, enquanto nossa implementação de Klein necessitou de 80 ms (0.080 segundos), sendo aproximadamente 71 vezes mais lenta.

5 Conclusão

Esta comparação experimental demonstrou que, apesar de ser fundamentada em um algoritmo teoricamente mais avançado (Philip Klein, 1998), nossa implementação não atingiu a performance esperada, sendo superada de forma consistente e significativa pela implementação do algoritmo de ”Sasha” (Zhang e Shasha, 1989).

Embora a teoria aponte a complexidade de Klein como uma vantagem, os resultados sugerem que a sobrecarga computacional (overhead) da nossa implementação ou os fatores constantes envolvidos na complexidade do algoritmo superaram os benefícios teóricos para os casos de teste utilizados. Isso evidencia que a superioridade assintótica nem sempre se traduz em melhor desempenho prático, especialmente quando a implementação pode não estar perfeitamente otimizada ou quando os dados não exploram o pior caso do algoritmo mais simples.

Com base exclusivamente neste conjunto de dados, concluímos que, embora nosso objetivo fosse desenvolver uma solução mais rápida, a implementação do algoritmo de ”Sasha” (Zhang e Shasha) se mostrou inequivocamente superior em termos de tempo de execução, representando a escolha mais prática e eficiente no contexto deste trabalho.

Declarações

Contribuição dos Autores

Todos os membros do grupo participaram de todas as fases do projeto. A distribuição de foco foi a seguinte: Pedro e Gustavo lideraram a implementação do algoritmo de Zhang e Shasha. Guilherme e Arthur focaram na implementação do algoritmo de Klein. Igor e Rafael desenvolveram o framework de testes e conduziram a análise de desempenho. A redação do relatório foi um esforço colaborativo.

Disponibilidade dos dados e materiais

Os conjuntos de softwares desenvolvidos durante o estudo estão disponíveis em: https://github.com/DE4THB0RN/Trabalho3-PAA.git.

References

[1] Kaizhong Zhang and Dennis Shasha. “Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems”. In: *SIAM J. Comput.* 18 (Dec. 1989), pp. 1245–1262. DOI: 10.1137/0218082.

[2] Philip N. Klein. “Computing the Edit-Distance between Unrooted Ordered Trees”. In: *Proceedings of the 6th Annual European Symposium on Algorithms*. ESA ’98. Berlin, Heidelberg: Springer-Verlag, 1998, pp. 91–102. ISBN: 3540648488.

[3] Daniel D. Sleator and Robert Endre Tarjan. “A data structure for dynamic trees”. In: *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*. STOC ’81. Milwaukee, Wisconsin, USA: Association for Computing Machinery, 1981, pp. 114–122. ISBN: 9781450373920. DOI: 10.1145/800076.802464. URL: https://doi.org/10.1145/800076.802464.

[4] Shin-Yee Lu. “A Tree-to-Tree Distance and Its Application to Cluster Analysis”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1.2 (1979), pp. 219–224. DOI: 10.1109/TPAMI.1979.6786615.

[5] Kuo-Chung Tai. “The Tree-to-Tree Correction Problem”. In: *J. ACM* 26.3 (July 1979), pp. 422–433. ISSN: 0004-5411. DOI: 10.1145/322139.322143. URL: https://doi.org/10.1145/322139.322143.