




# Distância de Edição entre Árvores: Uma Análise Comparativa entre os Algoritmos de Selkow e Zhang-Shasha


Gabriel de Cortez Mourão   [ Pontifícia Universidade Católica de Minas Gerais | [gabriel.mourao.1449838@sga.pucminas.br](mailto:gabriel.mourao.1449838@sga.pucminas.br) ]


Luís Augusto Lima de Oliveira   [ Pontifícia Universidade Católica de Minas Gerais | [luís.oliveira.695168@sga.pucminas.br](mailto:luís.oliveira.695168@sga.pucminas.br) ]

Mateus Fernandes Barbosa   [ Pontifícia Universidade Católica de Minas Gerais | [mateus.barbosa@sga.pucminas.br](mailto:mateus.barbosa@sga.pucminas.br) ]

Rafael Fleury Barcellos Ceolin de Oliveira  [ Pontifícia Universidade Católica de Minas Gerais | [rafael.barcellos@sga.pucminas.br](mailto:rafael.barcellos@sga.pucminas.br) ]

Victor Ferraz de Moraes   [ Pontifícia Universidade Católica de Minas Gerais | [vmoraes@sga.pucminas.br](mailto:vmoraes@sga.pucminas.br) ]

Vitória Símil Araújo  [ Pontifícia Universidade Católica de Minas Gerais | [vitoria.araujo.1321449@sgapucminas.br](mailto:vitoria.araujo.1321449@sgapucminas.br) ]

 Instituto de Ciências Exatas e Informática, Pontifícia Universidade Católica de Minas Gerais, R. Dom José Gaspar 500, Coração Eucarístico, Belo Horizonte, MG, 30535-901, Brazil.

**Abstract.** A distância de edição entre árvores (Tree Edit Distance - TED) é uma medida fundamental para avaliar a similaridade entre estruturas hierárquicas, com aplicações relevantes em bioinformática, visão computacional e linguística computacional. Este trabalho compara dois algoritmos clássicos para o cálculo da TED: o algoritmo de Zhang-Shasha (1989), conhecido por seu uso extensivo de programação dinâmica, e o algoritmo de Selkow (1977). Foram realizadas implementações em C++ e experimentos empíricos com árvores geradas aleatoriamente de diferentes tamanhos e estruturas. Os resultados mostram os custos computacionais em tempo e espaço de cada abordagem, destacando os trade-offs entre simplicidade de implementação, uso de memória e desempenho em diferentes cenários. A análise contribui para a escolha informada de algoritmos de TED conforme as restrições e exigências das aplicações reais.

**Keywords:** Distância de edição de árvores, Algoritmo de Zhang-Shasha, Algoritmo de Selkow, Análise de algoritmos, Programação dinâmica

## 1 Introdução

O campo de estudos em Projeto e Análise de Algoritmos tem como foco principal o estudo e desenvolvimento de algoritmos que resolvem problemas computacionais de forma eficiente. Isso significa criar soluções que funcionem corretamente e utilizem bem os recursos do computador, como tempo de execução e memória. Esse conhecimento é essencial para qualquer área da computação, pois permite escolher ou construir algoritmos adequados para diferentes tipos de aplicações, desde sistemas embarcados até grandes bancos de dados.

Um dos tópicos de estudo da área é a comparação entre estruturas de dados, fornecendo um aprofundamento mais detalhado de quando devemos utilizá-las em face ao contexto e problema abordado. Dentre essas estruturas, as árvores são um tipo de grafo em que os dados são organizados de forma hierárquica, com um "nó raiz" e vários "filhos" conectados. A distância de edição de árvores (TED - Tree Edit Distance) é uma maneira de medir o quão diferentes duas árvores são, ou seja, qual o menor número de mudanças (como inserir, remover ou renomear um nó) necessárias para transformar uma árvore na outra. Essa ideia é muito útil em aplicações reais como comparação de estruturas de RNA, análise de documentos XML e detecção de plágio em códigos-fonte.

Neste trabalho, são comparados dois algoritmos diferentes que resolvem o problema da TED. O primeiro é o algoritmo de Zhang-Shasha [1989], que usa programação dinâmica para calcular a distância de edição entre árvores de forma eficiente. O segundo é o algoritmo de Selkow [1977], um algoritmo que calcula a distância

de edição recursivamente em cada nó da subárvore.

## 2 Definições

Considere uma árvore  $T$ , em que cada nó possui um rótulo. Um nó se difere de outro quando possui rótulos distintos. As possíveis edições em uma árvore são:

- Edição: A simples renomeação de um rótulo de um nó para o outro, sem alteração na estrutura da árvore
- Remoção: Implica em fazer todos filhos de um nó  $i$  os filhos da pai de  $i$ , e a remoção subsequente desse nó
- Adição: Adicionar um nó  $i_1$  como filho de  $i$  implica que todos os filhos de  $i$  serão filhos de  $i_1$  e  $i_1$  será filho de  $i$

Tendo isso em mente, uma sequência de edições  $S$  de uma árvore  $T_1$  para uma árvore resultante  $T_2$  define um conjunto de operações  $a \rightarrow b$ , onde  $a \in T_1$  e  $b \in T_2$ , para gerar todos os elementos de  $T_2$  a partir de  $T_1$ . Remoções podem ser representadas como  $a \rightarrow \lambda$  e adições como  $\lambda \rightarrow b$ .

Será definido  $\gamma()$  uma função de custo arbitrário para as operações, onde:

- $\gamma(a \rightarrow b) = 0$  onde  $a, b \neq \lambda$  e  $a \neq b$
- $\gamma(a \rightarrow b) = 0$  onde  $a, b \neq \lambda$  e  $a = b$
- $\gamma(a \rightarrow \lambda)$
- $\gamma(\lambda \rightarrow b)$

Considerando a estrutura da árvore, também é possível definir os seguintes aspectos:

- Se existem operações  $i_1 \rightarrow j_1$  e  $i_2 \rightarrow j_2$ , se  $i_1$  estiver a esquerda em ordenação pós-ordem de  $i_2$ , o mesmo se aplica para  $j_1$  e  $j_2$  para preservar relações de irmandade
- Se existem operações  $i_1 \rightarrow j_1$  e  $i_2 \rightarrow j_2$ , se  $i_1$  for ancestral de  $i_2$ , o mesmo se aplica para  $j_1$  e  $j_2$ , preservando ancestralidade

Essas propriedades definem o problema de edição e são usadas como base na construção dos algoritmos.

### 3 Modelagem das Árvores

#### 3.1 Estrutura de dados em `arvore.cpp`

O arquivo é composto por duas classes principais `No` e `Arvore`.

**Classe `No`:**

```
class No {
public:
    string rotulo;
    vector<unique_ptr<No>> filhos;
};
```

Cada nó contém:

- Um **rótulo** do tipo `string`, que identifica semanticamente o nó;
- Um **vetor de filhos** implementado com `unique_ptr`, que garante a posse exclusiva dos nós filhos;
- Um método para adicionar filhos dinamicamente.

Essa representação configura uma árvore **enária rotulada com gerenciamento automático de memória**, permitindo uma quantidade arbitrária de filhos por nó.

**Classe `Arvore`:**

```
class Arvore {
private:
    unique_ptr<No> raiz;
public:
    Arvore();
    No* obterRaiz();
};
```

A classe `Arvore` encapsula a raiz da árvore e oferece métodos auxiliares:

- `obterNoRaiz()`: retorna um ponteiro constante para o nó raiz da árvore.
- `obterFilhos(const No* no)`: Retorna um vetor com os filhos de um nó específico.

### 4 Algoritmo de Zhang-Shasha

O algoritmo de Zhang-Shasha é um método eficiente para calcular a distância de edição entre árvores. A distância de edição entre árvores mede o número mínimo de operações

necessárias para transformar uma árvore em outra. Esse algoritmo foi proposto por Kaizhong Zhang e Dennis Shasha [1989] e é amplamente utilizado em problemas relacionados à comparação de estruturas hierárquicas.

#### 4.1 Funcionamento do Algoritmo

O algoritmo utiliza uma abordagem baseada em programação dinâmica para calcular a distância de edição entre duas árvores. Ele é uma extensão do conceito de distância de edição entre strings, mas adaptado para lidar com a estrutura hierárquica das árvores, seu passo a passo inclui:

1. Pré-processamento das Árvores:
  - Realiza-se a travessia pós-ordem para calcular os índices, assim como os valores de  $L(i)$  e identificar os *keyroots* de ambas as árvores.
  - $keyroots(T)$  é dado como todos elementos de uma árvore  $T$  de maior ancestralidade entre todos os demais elementos que possuem mesmo  $l(i)$
2. Construção das Matrizes:
  - Inicializam-se as matrizes de distância (`tree_dist`) e (`forest_dist`).
3. Cálculo Recursivo:
  - Para cada par de *keyroots*, calcula-se a distância de edição entre as subárvores correspondentes utilizando a abordagem de programação dinâmica.
4. Resultado Final:
  - O valor final da distância de edição é armazenado na última célula da matriz (`tree_dist`), representando o custo mínimo para transformar uma árvore na outra.

#### 4.2 Funcionamento do algoritmo

##### 4.2.1 Árvore

A representação da árvore é feita por meio de duas classes principais: `Node` (para representar os nós) e `Tree` (para representar a estrutura hierárquica). As funções implementadas nesta parte incluem:

- `post_order`: Realiza a travessia pós-ordem da árvore, atribuindo índices aos nós e calculando o valor ( $L_i$ ) (índice do primeiro nó na subárvore)
- `find_Keyroots`: Identifica os *keyroots* (raízes-chave) da árvore, que são usados no cálculo eficiente da distância de edição.

##### 4.2.2 Cálculo de Distância de Edição

O cálculo da distância de edição entre duas árvores é realizado pela classe `Tree_Editing`. Esta classe utiliza programação dinâmica para computar o custo mínimo de transformar uma árvore em outra. As principais funções implementadas incluem:

- `post_order`: `initialize_matrix`: Inicializa as matrizes de distâncias (`tree_dist`) e (`forest_dist`) com os custos básicos de inserção e remoção.

- `interval_calc`: Calcula o tamanho do intervalo relevante para a matriz de floresta, com base nos valores  $L(i)$  e  $L(j)$ , sendo  $L$  definido como o nó mais à esquerda possível.
- `comput_tree_dist`: Calcula a distância de edição entre duas subárvores específicas usando a matriz de floresta (`forest_dist`)).
- `tree_dist_calc`: Calcula a distância de edição total entre duas árvores, iterando sobre os keyroots de ambas as árvores e chamando `comput_tree_dist`.

## 5 Algoritmo de Selkow

O algoritmo de Selkow é uma abordagem "força bruta" capaz de calcular a distância de edição entre árvores não ordenadas. Como mencionado nas seções anteriores, o custo de edição mede o número mínimo de operações necessárias para transformar uma árvore não ordenada em outra. Esse algoritmo foi proposto em 1979 por Stanley M. Selkow e, pelo elevado custo em termos de tempo e espaço, deixou de ser considerado uma opção viável, sendo substituído pelo algoritmo de Zhang-Shasha.

### 5.1 Custos de Edição

A implementação do modelo de custos do algoritmo de distância de edição de árvores foi realizada por meio da classe `CalculadorDeCustos`, presente no arquivo `custo.cpp`. Esta classe é responsável por quantificar os custos associados às três operações básicas da edição: inserção, remoção e substituição (rotulação) de nós.

O custo básico de cada operação é fornecido ao construtor da classe e armazenado como atributos internos. Para a inserção e remoção de subárvores inteiras, foram implementados métodos recursivos `calcularCustoInsercaoRecursivo` e `calcularCustoDelecaoRecursivo`, que acumulam os custos de cada nó visitado. Para evitar recomputações, esses métodos utilizam *memoization* por meio de *hashmaps*, que armazenam os resultados intermediários para cada ponteiro de nó já processado.

A operação de substituição de rótulo entre dois nós utiliza um modelo de similaridade textual. Para isso, foi incorporado o algoritmo de distância de Levenshtein [1966], que calcula a quantidade mínima de edições necessárias para transformar uma string em outra. O custo final é armazenado em *cache*, utilizando pares de strings como chave.

A classe oferece ainda métodos públicos `custoInsercaoSubarvore`, `custoDelecaoSubarvore` e `custoRotulacao`, que encapsulam as lógicas anteriores e são utilizados diretamente durante o cálculo da distância de edição. Por fim, o método `limparCache` permite reinicializar os caches entre diferentes execuções do algoritmo, garantindo consistência nos resultados experimentais.

### 5.2 Funcionamento do Algoritmo

A abordagem deste algoritmo combina programação dinâmica com uma estratégia de fragmentação do problema,

similar à divisão e conquista. Contudo, ele não garante a divisão das instâncias pela metade, pois a cada diagonal calculada da matriz de custos, uma chamada recursiva gera uma nova matriz para uma subárvore específica. Tal como o algoritmo de Zhang-Shasha, este também se baseia no conceito de distância de edição de strings, mas com adaptações para lidar com a estrutura de árvores.

Dadas duas árvores  $T1$  e  $T2$ , a sequência de passos da execução do algoritmo se dá:

#### 1. Pré-processamento dos custos:

- Cria-se um vetor contendo o custo de inserção de cada uma das sub-árvores derivadas da árvore alvo ( $T2$ ).
- Cria-se um vetor contendo o custo de remoção de cada uma das sub-árvores derivadas da árvore de origem ( $T1$ ).
- Calcula-se o custo de transformar/editar a *label* da raiz de  $T1$  na *label* da raiz de  $T2$ .

#### 2. Construção da Matriz principal:

- Após a criação de uma matriz de tamanho  $(|T1| + 1) \times (|T2| + 1)$ , tal que  $|T|$  representa o total de nós numa árvore, inicializa-se a posição  $[0][0]$  com o valor do custo de edição das raízes de  $T1$  e  $T2$ , previamente calculados.
- Em seguida, inicializa-se a primeira linha da matriz com os valores do vetor dos custos de inserção das sub-árvores de  $T2$  e inicializa-se a primeira coluna da matriz com os valores do vetor dos custos de remoção das sub-árvores de  $T1$ .

#### 3. Preenchimento da Matriz:

- Para as demais células da matriz inicializa-se um duplo `for` em função de  $i$  e de  $j$ , com ambos começando em 1, em seguida, cada célula  $[i][j]$  subsequente recebe o menor valor entre a soma da posição  $[i - 1][j]$  com a inserção da sub-árvore  $T2[j]$ , a soma da posição  $[i][j - 1]$  com o custo de remoção da sub-árvore  $T1[i]$  e a soma da posição  $[i - 1][j - 1]$  com o retorno da chamada recursiva do algoritmo de Selkow passando como parâmetro as sub-árvores com raiz  $T1[i]$  e  $T2[j]$ .

#### 4. Resultado Final:

- O resultado final é encontrado na posição  $[m][n]$  da matriz, tal que  $m$  corresponde ao número de filhos da raiz de  $T1$  e  $n$  corresponde ao número de filhos da raiz de  $T2$ , e corresponde ao número mínimo de operações de edição necessárias para transformar a árvore  $T1$  na árvore  $T2$ .

## 5.3 Análise de Complexidade

Para efetuar o cálculo de complexidade, optou-se pela análise que consiste na comparação de árvores completas, de modo que a raiz tenha a maior quantidade de filhos possível, ou seja, há um nó raiz com  $N - 1$  filhos, tal que  $N$  é igual a  $|T|$ . Nesse cenário, observou-se que o custo em termos de espaço se dá em função do número de filhos de cada uma das raízes; isso ocorreu porque, nessas circunstâncias, há apenas uma matriz

com tamanho igual à  $|T1| \times |T2|$ . Considerando que ambas as árvores têm a mesma quantidade de nós, observou-se uma matriz quadrática que precisa ser preenchida e a cada nova célula é feita uma chamada recursiva que computa o custo de edição de dois nós. Dessa forma, sabendo que o custo de edição de dois nós é 1, e que cada nó tem o tamanho de um *double*, ou seja, 8 bytes, estimou-se que o custo de espaço é equivalente à  $((N - 1)^2 + N^2) * 8$ . Concluindo  $O(N^2)$ . Sob essa mesma perspectiva, obteve-se por meio de uma redução polinomial com os dados obtidos, concluiu-se que a ordem de complexidade em termos de tempo é  $O(N^2)$ .

## 6 Testes e análise

Nesta seção, apresentamos os resultados experimentais obtidos com as implementações dos dois algoritmos, avaliando desempenho em tempo de execução (ms), custo da distância de edição (TED) e uso de memória (MB) para diferentes tamanhos de entrada.

**Table 1.** Selkow em piores casos

T1	T2	T(ms)	TED	Espaço (MB)
10	10	0.3	0	0.0014
100	100	27.9	0	0.158
1000	1000	3942.7	0	15.98
10k	10k	501.2k	0	1599.8

**Table 2.** Zhang-Shasha em melhor e pior caso

T1	T2	T(ms)	Dist	Mem (MB)
10k	10k	5545.3	0	0.76
10k	10k	277.4k	0	0.76

Comparando os dois algoritmos:

- Selkow tem uma implementação mais simples, mas não escalável, e alto custo com árvores completas.
- Zhang-Shasha tem custo inicial de programação dinâmica maior, mas oferece consistência e melhor desempenho em escalas maiores.

## 7 Divisão de Tarefas

O trabalho foi separado em grupos de forma a não sobrecarregar nenhum dos membros da equipe. A divisão de tarefas, portanto, foi realizada da seguinte maneira:

Integrante	Algoritmo Atribuído	Tarefas Realizadas
Gabriel	Zhang-Shasha	Implementação Cálculo de Edição
Luis	Zhang-Shasha	Implementação Cálculo de Edição
Mateus	Selkow	Distância de Edição de Árvore e Análise de Resultados
Rafael	Selkow	Implementação da estrutura de árvore
Victor	Selkow	Implementação de cálculo dos custos de edição
Vitória	Zhang-Shasha	Árvore e Cálculos Li e Keyroots

**Table 3.** Divisão de Tarefas

## 8 Conclusão

A partir dos testes realizados, foi possível identificar diferenças importantes entre os algoritmos de Zhang and Shasha [1989] e Selkow [1977] para calcular a distância de edição entre árvores. O algoritmo de Zhang-Shasha mostrou um desempenho consistente mesmo em situações mais exigentes, como nos piores casos. Além disso, sua performance variou pouco entre os testes com entradas aleatórias e os casos extremos, o que indica que é uma solução confiável para diferentes tipos de problemas. Em contrapartida, os experimentos com o algoritmo de Selkow revelaram limitações significativas. Quando submetido a entradas grandes, seu tempo de execução e uso de memória aumentaram de forma abrupta. Por exemplo, com árvores de 10.000 nós, o tempo de processamento ultrapassou 500 segundos e o consumo de memória passou de 1.5 GB. Isso mostra que ele não é indicado para casos com estruturas grandes ou muito complexas. Com isso, é possível concluir que o algoritmo proposto por Zhang-Shasha é mais eficiente e robusto, sendo mais apropriado para aplicações que exigem desempenho constante e previsível, como comparação de estruturas biológicas ou análise de documentos em formato de árvore. Apesar da simplicidade de implementação do algoritmo de Selkow, ele deve ser usado com cautela. Para trabalhos futuros, uma alternativa interessante seria aplicar paralelismo ou desenvolver melhorias específicas para reduzir os custos computacionais desse algoritmo.

## References

- Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710. Original in Russian: Doklady Akademii Nauk SSSR, 163(4):845–848, 1965.
- Selkow, S. M. (1977). The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186. DOI: 10.1016/0020-0190(77)90064-3.
- Zhang, K. and Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262. DOI: 10.1137/0218082.