Corretor Ortográfico

Leonardo Araújo

UFSJ

Correção Ortográfica

A correção ortográfica é uma parte integral da escrita moderna, variando de **mensagens de texto** e **e-mails** à criação de documentos e **buscas na web**. Apesar de sua onipresença, os corretores ortográficos modernos não são perfeitos, como evidenciado por cenários de "autocorreção que deu errado".

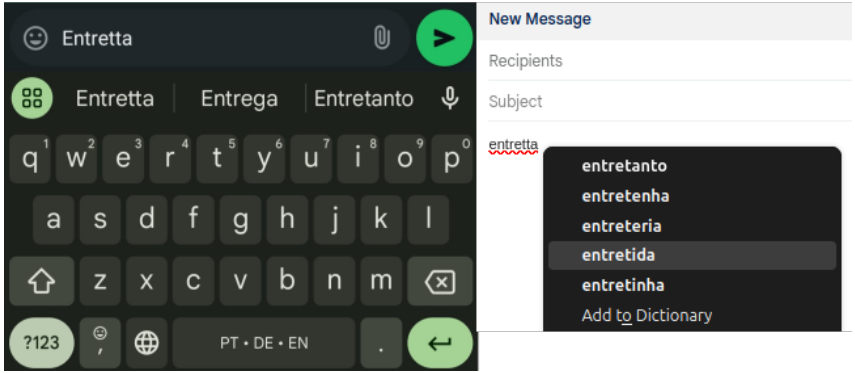


Figura 1: Spell checker.

Aplicações da Verificação Ortográfica

- Redação de Texto
- Sistemas Automatizados e de Informação
  - Sistemas de Entrada de Dados
  - Busca e Recuperação de Informação
  - Reconhecimento Óptico de Caracteres (OCR)
  - Chatbots
  - Sistemas de Tradução

Aplicações da Verificação Ortográfica

- Redação de Texto
- Sistemas Automatizados e de Informação
  - Sistemas de Entrada de Dados
  - Busca e Recuperação de Informação
  - Reconhecimento Óptico de Caracteres (OCR)
  - Chatbots
  - Sistemas de Tradução

Tarefa de Correção Ortográfica Automática

1. detecção de um erro;
2. geração de candidatos a correção;
3. classificação das correções candidatas;
4. realizar a correção automática.

Tarefa de Correção Ortográfica Automática

1. detecção de um erro;
2. geração de candidatos a correção;
3. classificação das correções candidatas;
4. realizar a correção automática.

Tarefa de Correção Ortográfica Automática

1. detecção de um erro;
2. geração de candidatos a correção;
3. classificação das correções candidatas;
4. realizar a correção automática.

Tarefa de Correção Ortográfica Automática

1. detecção de um erro;
2. geração de candidatos a correção;
3. classificação das correções candidatas;
4. realizar a correção automática.

Perspectivas na Correção Ortográfica

1. Correção Ortográfica de Não-Palavras
   - Detecta e corrige erros onde a **palavra não existe** no dicionário. Exemplo:
     - Entrada: `speling`, `asunto`
     - Correção: `spelling`, `assunto`

2. Correção Ortográfica de Palavras Reais
   - Detecta e corrige erros onde a **palavra existe** mas está contextualmente errada. Exemplo:
     - Entrada: `I no what to do.`, `Este método é mas confiável.`,
     - Correção: `I know what to do.`, `Este método é mais confiável.`

Perspectivas na Correção Ortográfica

1. Correção Ortográfica de Não-Palavras
   - Detecta e corrige erros onde a **palavra não existe** no dicionário. Exemplo:
     - Entrada: `speling`, `asunto`
     - Correção: `spelling`, `assunto`

2. Correção Ortográfica de Palavras Reais
   - Detecta e corrige erros onde a **palavra existe** mas está contextualmente errada. Exemplo:
     - Entrada: `I no what to do.`, `Este método é mas confiável.`,
     - Correção: `I know what to do.`, `Este método é mais confiável.`

Fontes de Erros na Ortografia

1. Erros Tipográficos:
   - Podem variar com dispositivos de entrada (teclado físico ou virtual, ou sistema de OCR) e condições ambientais.
   - Inserção: `escrevenmdo` → `escrevendo`
   - Deleção: `escrevndo` → `escrevendo`
   - Substituição: `escrevemdo` → `escrevendo`
   - Transposição: `esrcevendo` → `escrevendo`
   - Marcação diacrítica: a → á
2. Erros de Homófonos:
   - Homófonos: `their` / `there`, a / `há`
   - Quase-homófonos: `accept` / `except`, `sessão` / `cessão`
3. Erros Gramaticais:
   - `among` / `between`
4. Erros de Fronteira de Palavras:
   - `maybe` / `may be`, `talvez` / `tal vez`

Fontes de Erros na Ortografia

1. Erros Tipográficos:
   - Podem variar com dispositivos de entrada (teclado físico ou virtual, ou sistema de OCR) e condições ambientais.
   - Inserção: `escrevenmdo` → `escrevendo`
   - Deleção: `escrevndo` → `escrevendo`
   - Substituição: `escrevemdo` → `escrevendo`
   - Transposição: `esrcevendo` → `escrevendo`
   - Marcação diacrítica: `a` → `á`
2. Erros de Homófonos:
   - Homófonos: `their` / `there`, `a` / `há`
   - Quase-homófonos: `accept` / `except`, `sessão` / `cessão`
3. Erros Gramaticais:
   - `among` / `between`
4. Erros de Fronteira de Palavras:
   - `maybe` / `may be`, `talvez` / `tal vez`

Fontes de Erros na Ortografia

1. Erros Tipográficos:
   - Podem variar com dispositivos de entrada (teclado físico ou virtual, ou sistema de OCR) e condições ambientais.
   - Inserção: `escrevenmdo` → `escrevendo`
   - Deleção: `escrevndo` → `escrevendo`
   - Substituição: `escrevemdo` → `escrevendo`
   - Transposição: `esrcevendo` → `escrevendo`
   - Marcação diacrítica: `a` → `á`
2. Erros de Homófonos:
   - Homófonos: `their` / `there`, `a` / `há`
   - Quase-homófonos: `accept` / `except`, `sessão` / `cessão`
3. Erros Gramaticais:
   - `among` / `between`
4. Erros de Fronteira de Palavras:
   - `maybe` / `may be`, `talvez` / `tal vez`

Fontes de Erros na Ortografia

1. Erros Tipográficos:
   - Podem variar com dispositivos de entrada (teclado físico ou virtual, ou sistema de OCR) e condições ambientais.
   - Inserção: `escrevenmdo` → `escrevendo`
   - Deleção: `escrevndo` → `escrevendo`
   - Substituição: `escrevemdo` → `escrevendo`
   - Transposição: `esrcevendo` → `escrevendo`
   - Marcação diacrítica: `a` → `á`
2. Erros de Homófonos:
   - Homófonos: `their` / `there`, `a` / `há`
   - Quase-homófonos: `accept` / `except`, `sessão` / `cessão`
3. Erros Gramaticais:
   - `among` / `between`
4. Erros de Fronteira de Palavras:
   - `maybe` / `may be`, `talvez` / `tal vez`

Algoritmos e Ferramentas Notáveis

- **Soundex** (1918): Algoritmo fonético que mapeia nomes com sons semelhantes.

```
Stephen → S315, Perez  → P620, Juice  → J200, Robert → R163
Steven  → S315, Powers → P620, Juicy  → J200, Rupert → R163
Stefan  → S315, Price  → P620, Juiced → J230, Rubin  → R150
```

function SOUNDEX(*name*) **returns** *soundex form*

1. Keep the first letter of *name*
2. Drop all occurrences of non-initial a, e, h, i, o, u, w, y.
3. Replace the remaining letters with the following numbers:

   b, f, p, v $\rightarrow$ 1
   c, g, j, k, q, s, x, z $\rightarrow$ 2
   d, t $\rightarrow$ 3
   l $\rightarrow$ 4
   m, n $\rightarrow$ 5
   r $\rightarrow$ 6

4. Replace any sequences of identical numbers, only if they derive from two or more letters that were *adjacent* in the original name, with a single number (e.g., 666 $\rightarrow$ 6).
5. Convert to the form `Letter Digit Digit Digit` by dropping digits past the third (if necessary) or padding with trailing zeros (if necessary).

Figura 2: Algoritmo Soundex.

- **Shannon (1948)**: A Mathematical Theory of Communication.

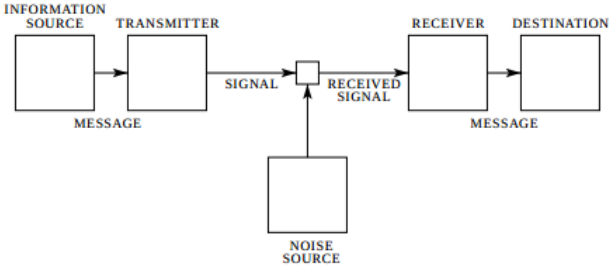


Fig. 1—Schematic diagram of a general communication system.

Figura 3: Canal Ruidoso.

- **Shannon (1950)**: Introduction of n-gram models in text analysis.

Figura 4: A predição de palavras é cruel?

- **Blair (1960)**: Algoritmo inicial para correção de erros ortográficos.
  - Blair introduziu o conceito de teclas similares para agrupar palavras com base na probabilidade de serem confundidas umas com as outras.
  - Abreviação de r letras de uma palavra de n letras
    - A teoria da informação assume que a informação transmitida é inversamente proporcional à sua probabilidade a priori de ocorrência.
    - 1ª proposta: eliminar $n - r$ letras na ordem de sua frequência esperada
    - 2ª proposta: eliminar com base na frequência de sua ocorrência como erros (melhor abordagem)
    - também deve ser dado peso à posição da letra na palavra

EXAMPLE

A	B	S	O	R	B	E	N	T		A	B	S	O	R	B	A	N	T
5	1	5	4	4	1	7	3	3.	Letter score	5	1	5	4	4	1	5	3	3
0	2	4	5	5	5	4	3	1	Position score	0	2	4	5	5	5	4	3	1
5	3	9	9	9	6	11	6	4	Sum of scores	5	3	9	9	9	6	9	6	4
		*	*	*		*	*		Delete			*	*	*		*	*	
					A	B	B	T	Abbreviation	A	B	B	T					

Figura 5: Exemplo do algoritmo de Blair.

- **Blair (1960)**: Algoritmo inicial para correção de erros ortográficos.
  - Blair introduziu o conceito de teclas similares para agrupar palavras com base na probabilidade de serem confundidas umas com as outras.
  - Abreviação de r letras de uma palavra de n letras
    - A teoria da informação assume que a informação transmitida é inversamente proporcional à sua probabilidade a priori de ocorrência.
    - 1ª proposta: eliminar $n - r$ letras na ordem de sua frequência esperada
    - 2ª proposta: eliminar com base na frequência de sua ocorrência como erros (melhor abordagem)
    - também deve ser dado peso à posição da letra na palavra

EXAMPLE

A	B	S	O	R	B	E	N	T		A	B	S	O	R	B	A	N	T
5	1	5	4	4	1	7	3	3.	Letter score	5	1	5	4	4	1	5	3	3
0	2	4	5	5	5	4	3	1	Position score	0	2	4	5	5	5	4	3	1
5	3	9	9	9	6	11	6	4	Sum of scores	5	3	9	9	9	6	9	6	4
		*	*	*		*	*		Delete			*	*	*		*	*	
					A	B	B	T	Abbreviation	A	B	B	T					

Figura 5: Exemplo do algoritmo de Blair.

- **Blair (1960)**: Algoritmo inicial para correção de erros ortográficos.
  - Blair introduziu o conceito de teclas similares para agrupar palavras com base na probabilidade de serem confundidas umas com as outras.
  - Abreviação de r letras de uma palavra de n letras
    - A teoria da informação assume que a informação transmitida é inversamente proporcional à sua probabilidade a priori de ocorrência.
    - 1ª proposta: eliminar $n - r$ letras na ordem de sua frequência esperada
    - 2ª proposta: eliminar com base na frequência de sua ocorrência como erros (melhor abordagem)
    - também deve ser dado peso à posição da letra na palavra

EXAMPLE

A	B	S	O	R	B	E	N	T		A	B	S	O	R	B	A	N	T
5	1	5	4	4	1	7	3	3.	Letter score	5	1	5	4	4	1	5	3	3
0	2	4	5	5	5	4	3	1	Position score	0	2	4	5	5	5	4	3	1
5	3	9	9	9	6	11	6	4	Sum of scores	5	3	9	9	9	6	9	6	4
		*	*	*		*	*		Delete			*	*	*		*	*	
					A	B	B	T	Abbreviation	A	B	B	T					

Figura 5: Exemplo do algoritmo de Blair.

- **Blair (1960)**: Algoritmo inicial para correção de erros ortográficos.
  - Blair introduziu o conceito de teclas similares para agrupar palavras com base na probabilidade de serem confundidas umas com as outras.
  - Abreviação de r letras de uma palavra de n letras
    - A teoria da informação assume que a informação transmitida é inversamente proporcional à sua probabilidade a priori de ocorrência.
    - 1ª proposta: eliminar $n - r$ letras na ordem de sua frequência esperada
    - 2ª proposta: eliminar com base na frequência de sua ocorrência como erros (melhor abordagem)
    - também deve ser dado peso à posição da letra na palavra

EXAMPLE

A	B	S	O	R	B	E	N	T		A	B	S	O	R	B	A	N	T	
5	1	5	4	4	1	7	3	3.	Letter score	5	1	5	4	4	1	5	3	3	
0	2	4	5	5	5	4	3	1	Position score	0	2	4	5	5	5	4	3	1	
5	3	9	9	9	6	11	6	4	Sum of scores	5	3	9	9	9	6	9	6	4	
		*	*	*		*	*		Delete			*	*	*		*	*		
					A	B	B	T	Abbreviation	A	B	B	T						

Figura 5: Exemplo do algoritmo de Blair.

- **Blair (1960)**: Algoritmo inicial para correção de erros ortográficos.
  - Blair introduziu o conceito de teclas similares para agrupar palavras com base na probabilidade de serem confundidas umas com as outras.
  - Abreviação de r letras de uma palavra de n letras
    - A teoria da informação assume que a informação transmitida é inversamente proporcional à sua probabilidade a priori de ocorrência.
    - 1ª proposta: eliminar $n - r$ letras na ordem de sua frequência esperada
    - 2ª proposta: eliminar com base na frequência de sua ocorrência como erros (melhor abordagem)
    - também deve ser dado peso à posição da letra na palavra

EXAMPLE

A	B	S	O	R	B	E	N	T		A	B	S	O	R	B	A	N	T	
5	1	5	4	4	1	7	3	3.	Letter score	5	1	5	4	4	1	5	3	3	
0	2	4	5	5	5	4	3	1	Position score	0	2	4	5	5	5	4	3	1	
5	3	9	9	9	6	11	6	4	Sum of scores	5	3	9	9	9	6	9	6	4	
		*	*	*		*	*		Delete			*	*	*		*	*		
					A	B	B	T	Abbreviation	A	B	B	T						

Figura 5: Exemplo do algoritmo de Blair.

TABLE I

THE LOGARITHM OF THE DESIRABILITY OF DELETING A LETTER AS A FUNCTION OF ITS NAME

Letter	Score	Letter	Score
A	5	N	3
B	1	O	4
C	5	P	3
D	0	Q	0
E	7	R	4
F	1	S	5
G	2	T	3
H	5	U	4
I	6	V	1
J	0	W	1
K	1	X	0
L	5	Y	2
M	1	Z	1

Figura 6: Pontuação da letra.

TABLE II

THE LOGARITHM OF THE DESIRABILITY OF DELETING A LETTER AS A FUNCTION OF ITS POSITION

Position	Score	Position	Score
1	0	9	5
2	1	10	5
3	2	11	6
4	3	12	6
5	4	13	6
6	4	14	6
7	5	15	6
8	5	16 up	7

Figura 7: Pontuação da posição.

- Distância **Damerau-Levenshtein** (1964, 1966): Uma métrica de *string* para medir a distância de edição entre duas sequências.

		S	u	n	d	a	y
	0	1	2	3	4	5	6
S	1	0	1	2	3	4	5
a	2	1	1	2	3	3	4
t	3	2	2	2	3	4	4
u	4	3	2	3	3	4	5
r	5	4	3	3	4	4	5
d	6	5	4	4	3	4	5
a	7	6	5	5	4	3	4
y	8	7	6	6	5	4	3

delete a

delete t

replace r with n

Figura 8: Quantas operações são necessárias para transformar *Saturday* em *Sunday*?

Levenshtein Distance Calculator

https://phiresky.github.io/levenshtein-demo/

Distância Levenshtein entre duas *strings* a, b (de comprimento $|a|$ e $|b|$ respectivamente) é dada por

$$\mathrm{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \mathrm{lev}\left(\mathrm{tail}(a), \mathrm{tail}(b)\right) & \text{if } \mathrm{head}(a) = \mathrm{head}(b), \\ 1 + \min \begin{cases} \mathrm{lev}\left(\mathrm{tail}(a), b\right) & \text{deletion} \\ \mathrm{lev}\left(a, \mathrm{tail}(b)\right) & \text{insertion} \\ \mathrm{lev}\left(\mathrm{tail}(a), \mathrm{tail}(b)\right) & \text{replacement} \end{cases} & \text{otherwise} \end{cases}$$

Distância de Damerau-Levenshtein: também permite a transposição de símbolos adjacentes.

As operações são caras e dependem da língua: por exemplo, a partir da versão 16.0, o Unicode define um total de 98682 caracteres chineses.

Distância Levenshtein entre duas *strings* a, b (de comprimento $|a|$ e $|b|$ respectivamente) é dada por

$$\mathrm{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \mathrm{lev}\left(\mathrm{tail}(a), \mathrm{tail}(b)\right) & \text{if } \mathrm{head}(a) = \mathrm{head}(b), \\ 1 + \min \begin{cases} \mathrm{lev}\left(\mathrm{tail}(a), b\right) & \text{deletion} \\ \mathrm{lev}\left(a, \mathrm{tail}(b)\right) & \text{insertion} \\ \mathrm{lev}\left(\mathrm{tail}(a), \mathrm{tail}(b)\right) & \text{replacement} \end{cases} & \text{otherwise} \end{cases}$$

Distância de Damerau-Levenshtein: também permite a transposição de símbolos adjacentes.

As operações são caras e dependem da língua: por exemplo, a partir da versão 16.0, o Unicode define um total de 98682 caracteres chineses.

Distância Levenshtein entre duas *strings* a, b (de comprimento $|a|$ e $|b|$ respectivamente) é dada por

$$\mathrm{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \mathrm{lev}\left(\mathrm{tail}(a), \mathrm{tail}(b)\right) & \text{if } \mathrm{head}(a) = \mathrm{head}(b), \\ 1 + \min \begin{cases} \mathrm{lev}\left(\mathrm{tail}(a), b\right) & \text{deletion} \\ \mathrm{lev}\left(a, \mathrm{tail}(b)\right) & \text{insertion} \\ \mathrm{lev}\left(\mathrm{tail}(a), \mathrm{tail}(b)\right) & \text{replacement} \end{cases} & \text{otherwise} \end{cases}$$

Distância de Damerau-Levenshtein: também permite a transposição de símbolos adjacentes.

As operações são caras e dependem da língua: por exemplo, a partir da versão 16.0, o Unicode define um total de 98682 caracteres chineses.

- **Árvores BK** (1973): Busca eficiente por correspondências próximas usando a distância de Levenshtein.
  - Um elemento arbitrário a é selecionado como nó raiz, então, conforme novos elementos são adicionados, mova para o nó filho onde a diferença absoluta entre a distância do novo elemento para o pai e a distância do nó filho para o pai é mínima. Para inserir um novo nó, encontre um nó folha ou descubra que nenhuma distância do nó filho corresponde de perto o suficiente.
  - A k-ésima subárvore é construída recursivamente de todos os elementos b tais que $d(a, b) = k$.
  - Ideia de busca: restringir a exploração da árvore a nós que possam apenas melhorar o melhor candidato encontrado até agora (usar a desigualdade triangular).

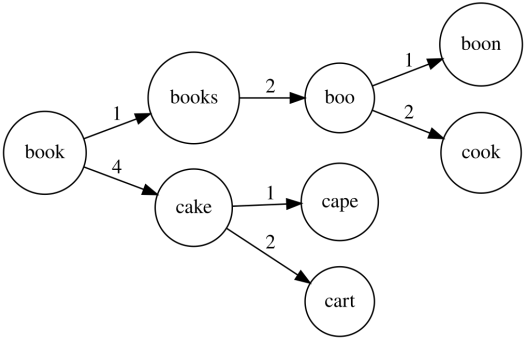
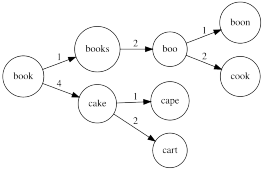
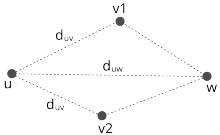


Figura 9: Árvore Burkhard-Keller.

Search for $w =$ 'cool'





1 $d_u = d(w, u) = d(\text{'cool'}, \text{'book'}) = 2$, set $d_{\text{best}} = 2$;
2 $v =$ 'books', $|d_{uv} - d_u| = |1 - 2| = 1 < d_{\text{best}}$, then select v;
3 $v =$ 'cake', $|d_{uv} - d_u| = |4 - 2| = 2 \not< d_{\text{best}}$, do not select v;
4 $d_u = d(w, u) = d(\text{'cool'}, \text{'books'}) = 3$, $d_u \not< d_{\text{best}}$;
5 $d_u = d(w, u) = d(\text{'cool'}, \text{'boo'}) = 2$, $d_u \not< d_{\text{best}}$;
6 $v =$ 'boon', $|d_{uv} - d_u| = |2 - 1| = 1 < d_{\text{best}}$, then select v;
7 $v =$ 'cook', $|d_{uv} - d_u| = |2 - 2| = 0 < d_{\text{best}}$, then select v;
8 $d_u = d(w, u) = d(\text{'cool'}, \text{'cook'}) = 1$, $d_u < d_{\text{best}}$, set $d_{\text{best}} = 1$;
9 $d_u = d(w, u) = d(\text{'cool'}, \text{'boon'}) = 2$, $d_u \not< d_{\text{best}}$;
10 'cook' is returned as the answer with $d_{\text{best}} = 1$.

- **SPELL** (Unix, 1975)
  - Detecção de erros apenas.
  - Remoção de prefixos e sufixos (reduz a lista para menos de 1/3);
    - `buzzed` → `buzz`, `mapping` → `map`, `possibly` → `possible`, `antisocial` → `social`, `metaphysics` → `physics`.
  - Hashing (descartando 60% dos bits restantes);
    Exemplos de funções de hashing:
    1. Shift-and-Add: $h = (h << 1) + \text{char}\%m$
    2. Hashing Multiplicativo: $h = (a \cdot h + \text{char})\%m$ (com a tipicamente 31 ou 33)
    3. Hashing baseado em XOR: $h = h \oplus (\text{char} << k)$
  - As palavras eram representadas por palavras de máquina de 16 bits;
  - Filtro de Bloom;
  - Falsos positivos.

- **SPELL** (Unix, 1975)
  - Detecção de erros apenas.
  - Remoção de prefixos e sufixos (reduz a lista para menos de 1/3);
    - `buzzed` → `buzz`, `mapping` → `map`, `possibly` → `possible`, `antisocial` → `social`, `metaphysics` → `physics`.
  - Hashing (descartando 60% dos bits restantes);
    Exemplos de funções de hashing:
    1. Shift-and-Add: $h = (h << 1) + \text{char}\%m$
    2. Hashing Multiplicativo: $h = (a \cdot h + \text{char})\%m$ (com a tipicamente 31 ou 33)
    3. Hashing baseado em XOR: $h = h \oplus (\text{char} << k)$
  - As palavras eram representadas por palavras de máquina de 16 bits;
  - Filtro de Bloom;
  - Falsos positivos.

- **SPELL** (Unix, 1975)
  - Detecção de erros apenas.
  - Remoção de prefixos e sufixos (reduz a lista para menos de 1/3);
    - `buzzed` → `buzz`, `mapping` → `map`, `possibly` → `possible`, `antisocial` → `social`, `metaphysics` → `physics`.
  - Hashing (descartando 60% dos bits restantes);
    Exemplos de funções de hashing:
    1. Shift-and-Add: $h = (h << 1) + \text{char}\%m$
    2. Hashing Multiplicativo: $h = (a \cdot h + \text{char})\%m$ (com a tipicamente 31 ou 33)
    3. Hashing baseado em XOR: $h = h \oplus (\text{char} << k)$
  - As palavras eram representadas por palavras de máquina de 16 bits;
  - Filtro de Bloom;
  - Falsos positivos.

- **SPELL** (Unix, 1975)
  - Detecção de erros apenas.
  - Remoção de prefixos e sufixos (reduz a lista para menos de 1/3);
    - `buzzed` → `buzz`, `mapping` → `map`, `possibly` → `possible`, `antisocial` → `social`, `metaphysics` → `physics`.
  - Hashing (descartando 60% dos bits restantes);
    Exemplos de funções de hashing:
    1. Shift-and-Add: $h = (h << 1) + \text{char}\%m$
    2. Hashing Multiplicativo: $h = (a \cdot h + \text{char})\%m$ (com a tipicamente 31 ou 33)
    3. Hashing baseado em XOR: $h = h \oplus (\text{char} << k)$
  - As palavras eram representadas por palavras de máquina de 16 bits;
  - Filtro de Bloom;
  - Falsos positivos.

- **SPELL** (Unix, 1975)
  - Detecção de erros apenas.
  - Remoção de prefixos e sufixos (reduz a lista para menos de 1/3);
    - `buzzed` → `buzz`, `mapping` → `map`, `possibly` → `possible`, `antisocial` → `social`, `metaphysics` → `physics`.
  - Hashing (descartando 60% dos bits restantes);
    Exemplos de funções de hashing:
    1. Shift-and-Add: $h = (h << 1) + \text{char}\%m$
    2. Hashing Multiplicativo: $h = (a \cdot h + \text{char})\%m$ (com a tipicamente 31 ou 33)
    3. Hashing baseado em XOR: $h = h \oplus (\text{char} << k)$
  - As palavras eram representadas por palavras de máquina de 16 bits;
  - Filtro de Bloom;
  - Falsos positivos.

- **SPELL** (Unix, 1975)
  - Detecção de erros apenas.
  - Remoção de prefixos e sufixos (reduz a lista para menos de 1/3);
    - `buzzed` → `buzz`, `mapping` → `map`, `possibly` → `possible`, `antisocial` → `social`, `metaphysics` → `physics`.
  - Hashing (descartando 60% dos bits restantes);
    Exemplos de funções de hashing:
    1. Shift-and-Add: $h = (h << 1) + \text{char}\%m$
    2. Hashing Multiplicativo: $h = (a \cdot h + \text{char})\%m$ (com a tipicamente 31 ou 33)
    3. Hashing baseado em XOR: $h = h \oplus (\text{char} << k)$
  - As palavras eram representadas por palavras de máquina de 16 bits;
  - Filtro de Bloom;
  - Falsos positivos.

- **Similaridade de Jaro** (1989)

A similaridade de Jaro sim_j de duas *strings* dadas s_1 e s_2 é

$$sim_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

- $|s_i|$ é o comprimento da *string* s_i;
- m é o número de ''caracteres correspondentes'' (veja abaixo);
- t é o número de ''transposições'' (veja abaixo).

A pontuação de similaridade de Jaro é 0 se as *strings* não corresponderem de forma alguma, e 1 se forem uma correspondência exata. Na primeira etapa, cada caractere de s_1 é comparado com todos os seus caracteres correspondentes em s_2. Dois caracteres de s_1 e s_2, respectivamente, são considerados **correspondentes** apenas se forem iguais e não estiverem a mais do que $\left\lfloor \frac{\max(|s_1|,|s_2|)}{2} \right\rfloor - 1$ caracteres de distância. **Transposição** é o número de caracteres correspondentes que não estão na ordem correta dividido por dois.

- **Similaridade de Jaro** (1989)

A similaridade de Jaro sim_j de duas *strings* dadas s_1 e s_2 é

$$sim_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

- $|s_i|$ é o comprimento da *string* s_i;
- m é o número de ''caracteres correspondentes'' (veja abaixo);
- t é o número de ''transposições'' (veja abaixo).

A pontuação de similaridade de Jaro é 0 se as *strings* não corresponderem de forma alguma, e 1 se forem uma correspondência exata. Na primeira etapa, cada caractere de s_1 é comparado com todos os seus caracteres correspondentes em s_2. Dois caracteres de s_1 e s_2, respectivamente, são considerados **correspondentes** apenas se forem iguais e não estiverem a mais do que $\left\lfloor \frac{\max(|s_1|,|s_2|)}{2} \right\rfloor - 1$ caracteres de distância. **Transposição** é o número de caracteres correspondentes que não estão na ordem correta dividido por dois.

- **Similaridade de Jaro-Winkler** (1990)
  - Introduz a modificação de Winkler.
  - Comprimento do prefixo ℓ: se duas *strings* compartilham um prefixo comum, é provável que sejam mais semelhantes.
  - Fator de escala p: aumenta a pontuação de similaridade de Jaro com base no comprimento do prefixo comum (geralmente definido como 0,1 e não deve exceder 0,25).

$$sim_w = sim_j + \ell p(1 - sim_j)$$

$1 - sim_j$: Este componente ajusta a contribuição do termo de similaridade do prefixo em relação à pontuação de similaridade de Jaro base (sim_j). Se sim_j já for alto, o impacto do ajuste do prefixo diminui, mas quando sim_j é mais baixo, a similaridade do prefixo pode aumentar significativamente a pontuação final de similaridade.

- **Similaridade de Jaro-Winkler** (1990)
  - Introduz a modificação de Winkler.
  - Comprimento do prefixo ℓ: se duas *strings* compartilham um prefixo comum, é provável que sejam mais semelhantes.
  - Fator de escala p: aumenta a pontuação de similaridade de Jaro com base no comprimento do prefixo comum (geralmente definido como 0,1 e não deve exceder 0,25).

$$sim_w = sim_j + \ell p(1 - sim_j)$$

$1 - sim_j$: Este componente ajusta a contribuição do termo de similaridade do prefixo em relação à pontuação de similaridade de Jaro base (sim_j). Se sim_j já for alto, o impacto do ajuste do prefixo diminui, mas quando sim_j é mais baixo, a similaridade do prefixo pode aumentar significativamente a pontuação final de similaridade.

- **Similaridade de Jaro-Winkler** (1990)
  - Introduz a modificação de Winkler.
  - Comprimento do prefixo ℓ: se duas *strings* compartilham um prefixo comum, é provável que sejam mais semelhantes.
  - Fator de escala p: aumenta a pontuação de similaridade de Jaro com base no comprimento do prefixo comum (geralmente definido como 0,1 e não deve exceder 0,25).

$$sim_w = sim_j + \ell p(1 - sim_j)$$

$1 - sim_j$: Este componente ajusta a contribuição do termo de similaridade do prefixo em relação à pontuação de similaridade de Jaro base (sim_j). Se sim_j já for alto, o impacto do ajuste do prefixo diminui, mas quando sim_j é mais baixo, a similaridade do prefixo pode aumentar significativamente a pontuação final de similaridade.

- **Similaridade de Jaro-Winkler** (1990)
  - Introduz a modificação de Winkler.
  - Comprimento do prefixo ℓ: se duas *strings* compartilham um prefixo comum, é provável que sejam mais semelhantes.
  - Fator de escala p: aumenta a pontuação de similaridade de Jaro com base no comprimento do prefixo comum (geralmente definido como 0,1 e não deve exceder 0,25).

$$sim_w = sim_j + \ell p(1 - sim_j)$$

$1 - sim_j$: Este componente ajusta a contribuição do termo de similaridade do prefixo em relação à pontuação de similaridade de Jaro base (sim_j). Se sim_j já for alto, o impacto do ajuste do prefixo diminui, mas quando sim_j é mais baixo, a similaridade do prefixo pode aumentar significativamente a pontuação final de similaridade.

- **Metaphone** (1990), Double Metaphone (2000), Metaphone 3 (2009): Extrai informações fonéticas para melhor correspondência.
  - Conjunto de regras que melhora o algoritmo Soundex.
  - `Smith → SM0, [SM0, XMT]`, `Schmidt → SXMTT, [XMT, SMT]`,
  - `Taylor → TLR, [TLR]`, `Taylor → EFNS, [AFNS]`,
  - `Roberts → RBRTS, [RPRTS]`
  - `spelling → SPLNK, [SPLNK]`, `speling → SPLNK, [SPLNK]`, `speeling → SPLNK, [SPLNK]`, `sprlling → SPRLNK, [SPRLNK]`

- **Modelo de Canal Ruidoso (Kernighan et al., 1990 e Mays et al., 1991)**: Modelos combinados de prior e verossimilhança.

No modelo de canal ruidoso, imaginamos que a forma superficial que vemos é na verdade uma forma "distorcida" de uma palavra original que passou por um canal ruidoso. O decodificador passa cada hipótese por um modelo deste canal e escolhe a palavra que melhor corresponde à palavra ruidosa superficial. (Jurafsky e Martin, 2024)

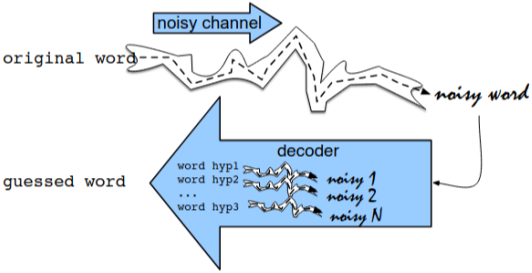


Figura 10: Modelo de Canal Ruidoso

Este modelo de canal ruidoso é uma forma de **inferência bayesiana**.

Dentre todas as palavras possíveis no vocabulário V, queremos encontrar a palavra $\hat{w}$ tal que $P(w|x)$ seja a mais alta para uma dada *string* observada x.

$$\hat{w} = \arg\max_{w \in V} P(w|x)$$

Usando Bayes: $P(x, w) = P(w|x)P(x) = P(x|w)P(w)$,

$$\hat{w} = \arg\max_{w \in V} \frac{P(x|w)P(w)}{P(x)} = \arg\max_{w \in V} \underbrace{P(x \mid w)}_{\substack{\text{channel model} \\ \text{or likelihood}}} \underbrace{P(w)}_{\text{prior}}$$

$$\hat{w} = \arg\max_{w \in V} (\log P(x \mid w) + \log P(w))$$

```
function NOISY CHANNEL SPELLING(word x, dict D, lm, editprob) returns correction

  if x ∉ D
    candidates, edits ← All strings at edit distance 1 from x that are ∈ D, and their edit
    for each c, e in candidates, edits
      channel ← editprob(e)
      prior ← lm(c)
      score[c] = log channel + log prior
    return argmax_c score[c]
```

Figura 11: Modelo de canal ruidoso para correção ortográfica de palavras desconhecidas (Jurafsky e Martin, 2024).

Example

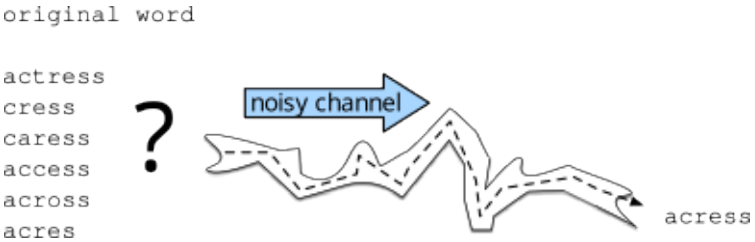


Figura 12: Exemplo: erro de ortografia `acress`.

		Transformation			
Error	Correction	Correct Letter	Error Letter	Position (Letter #)	Type
acress	actress	t	—	2	deletion
acress	cress	—	a	0	insertion
acress	caress	ca	ac	0	transposition
acress	access	c	r	2	substitution
acress	across	o	e	3	substitution
acress	acres	—	s	5	insertion
acress	acres	—	s	4	insertion

Figura 13: Correções candidatas para o erro de ortografia "acress" e as transformações que poderiam ter produzido o erro (Kernighan et al. (1990)). "—" representa o caractere nulo. (Jurafsky e Martin, 2024)

w	count(w)	p(w)
actress	9,321	.0000231
cress	220	.000000544
caress	686	.00000170
access	37,038	.0000916
across	120,844	.000299
acres	12,874	.0000318

Figura 14: Modelo de linguagem a partir das 404.253.213 palavras no Corpus of Contemporary English (Jurafsky e Martin, 2024).

Modelo de Erro

- Um modelo perfeito precisaria de todos os tipos de fatores: quem era o digitador, se o digitador era canhoto ou destro, e assim por diante.
- Podemos obter uma estimativa bastante razoável de $P(x|w)$ apenas observando o **contexto local**: a identidade da letra correta, o erro de ortografia e as letras ao redor.
- Matrizes de Confusão:
  - `del[x, y]: count(xy typed as x)`
  - `ins[x, y]: count(x typed as xy)`
  - `sub[x, y]: count(x typed as y)`
  - `trans[x, y]: count(xy typed as yx)`

Modelo de Erro

- Um modelo perfeito precisaria de todos os tipos de fatores: quem era o digitador, se o digitador era canhoto ou destro, e assim por diante.
- Podemos obter uma estimativa bastante razoável de $P(x|w)$ apenas observando o **contexto local**: a identidade da letra correta, o erro de ortografia e as letras ao redor.
- Matrizes de Confusão:
  - `del[x, y]: count(xy typed as x)`
  - `ins[x, y]: count(x typed as xy)`
  - `sub[x, y]: count(x typed as y)`
  - `trans[x, y]: count(xy typed as yx)`

Modelo de Erro

- Um modelo perfeito precisaria de todos os tipos de fatores: quem era o digitador, se o digitador era canhoto ou destro, e assim por diante.
- Podemos obter uma estimativa bastante razoável de $P(x|w)$ apenas observando o **contexto local**: a identidade da letra correta, o erro de ortografia e as letras ao redor.
- Matrizes de Confusão:
  - `del[x, y]: count(xy typed as x)`
  - `ins[x, y]: count(x typed as xy)`
  - `sub[x, y]: count(x typed as y)`
  - `trans[x, y]: count(xy typed as yx)`

sub[X, Y] = Substitution of X (incorrect) for Y (correct)

X \ Y (correct)	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0

Figura 15: Matriz de confusão para erros de ortografia (Kernighan et al., 1990).

Estimando o modelo de canal

$$P(x|w) = \begin{cases} \frac{\text{del}[x_{i-1}, w_i]}{\text{count}[x_{i-1} w_i]}, & \text{se apagamento} \\ \frac{\text{ins}[x_{i-1}, w_i]}{\text{count}[w_{i-1}]}, & \text{se inserção} \\ \frac{\text{sub}[x_i, w_i]}{\text{count}[w_i]}, & \text{se substituição} \\ \frac{\text{trans}[w_i, w_{i+1}]}{\text{count}[w_i w_{i+1}]}, & \text{se transposição} \end{cases}$$

Candidate Correction	Correct Letter	Error Letter	x|w	P(x|w)
actress	t	-	c|ct	.000117
cress	-	a	a|#	.00000144
caress	ca	ac	ac|ca	.00000164
access	c	r	r|c	.000000209
across	o	e	e|o	.0000093
acres	-	s	es|e	.0000321
acres	-	s	ss|s	.0000342

Figura 16: Modelo de canal para *acress*; as probabilidades são tomas das matrizes de confusão de `del[]`, `ins[]`, `sub[]`, e `trans[]`, como mostradas em Kernighan et al. (1990).

Final probabilities for each of the potential corrections

Candidate Correction	Correct Letter	Error Letter	x|w	P(x|w)	P(w)	10^9*P(x|w)P(w)
actress	t	-	c|ct	.000117	.0000231	2.7
cress	-	a	a|#	.00000144	.000000544	0.00078
caress	ca	ac	ac|ca	.00000164	.00000170	0.0028
access	c	r	r|c	.000000209	.0000916	0.019
across	o	e	e|o	.0000093	.000299	2.8
acres	-	s	es|e	.0000321	.0000318	1.0
acres	-	s	ss|s	.0000342	.0000318	1.0

Figura 17: Cálculo da classificação para cada correção candidata, usando o modelo de linguagem mostrado anteriormente e o modelo de erro. A pontuação final é multiplicada por 10^9 para facilitar a leitura (Jurafsky and Martin, 2024).

Infelizmente, o algoritmo estava errado aqui; a intenção do escritor se torna clara a partir do contexto: *. . . was called a "stellar and versatile* ***acress*** *whose combination of sass and glamour has defined her . . . "*. As palavras ao redor deixam claro que "*actress*" e não "*across*" era a palavra pretendida. (Jurafsky e Martin, 2024)

Usando o *Corpus of Contemporary American English* para calcular as probabilidades de **bigramas** para as palavras *actress* e *across* em seu contexto usando suavização de adição única, obtemos as seguintes probabilidades:

$$P(\text{actress}|\text{versatile}) = .000021$$
$$P(\text{across}|\text{versatile}) = .000021$$
$$P(\text{whose}|\text{actress}) = .0010$$
$$P(\text{whose}|\text{across}) = .000006$$

Multiplicando esses valores, obtemos a estimativa do modelo de linguagem para os dois candidatos em contexto:

$$P(\text{versatile actress whose}) = .000021 \times .0010 = 210 \times 10^{-10}$$
$$P(\text{versatile across whose}) = .000021 \times .000006 = 1 \times 10^{-10}$$

Usando o *Corpus of Contemporary American English* para calcular as probabilidades de **bigramas** para as palavras *actress* e *across* em seu contexto usando suavização de adição única, obtemos as seguintes probabilidades:

$$P(\text{actress}|\text{versatile}) = .000021$$
$$P(\text{across}|\text{versatile}) = .000021$$
$$P(\text{whose}|\text{actress}) = .0010$$
$$P(\text{whose}|\text{across}) = .000006$$

Multiplicando esses valores, obtemos a estimativa do modelo de linguagem para os dois candidatos em contexto:

$$P(\text{versatile actress whose}) = .000021 \times .0010 = 210 \times 10^{-10}$$
$$P(\text{versatile across whose}) = .000021 \times .000006 = 1 \times 10^{-10}$$

Jurafsky, D., & Martin, J. H. (2024). *Speech and Language Processing*.

Kernighan, M. D. et al. (1990). *A spelling correction program based on a noisy channel model*.

Mays, E. et al. (1991). *Context based spelling correction*.

- Modelo de Canal Ruidoso
  - Correct (Unix, 1990): Recebe entradas de palavras rejeitadas pelo SPELL e fornece candidatos. Operações: Inserção, Deleção, Substituição, Reversão. Usa probabilidades de erro.

- **Modelo de canal Brill-Moore** (2000): Edições de *string* para *string*.
  - Seja Σ um alfabeto, o modelo permite todas as operações de edição da forma $\alpha \rightarrow \beta$, onde $\alpha, \beta \in \Sigma^*$.
  - $P(\alpha \rightarrow \beta)$ é a probabilidade de que, quando o usuário pretende digitar α, ele digite β em vez disso.
  - $P(\alpha \rightarrow \beta | PNS)$ é a probabilidade condicionada pela posição na *string*
    - P(e | a) não varia muito com a posição.
    - P(ent | ant) é altamente dependente da posição.
    - As pessoas raramente digitam *antler* como *entler*, mas frequentemente digitam *reluctant* como *reluctent*.

- **Modelo de canal Brill-Moore** (2000): Edições de *string* para *string*.
  - Seja Σ um alfabeto, o modelo permite todas as operações de edição da forma $\alpha \rightarrow \beta$, onde $\alpha, \beta \in \Sigma^*$.
  - $P(\alpha \rightarrow \beta)$ é a probabilidade de que, quando o usuário pretende digitar α, ele digite β em vez disso.
  - $P(\alpha \rightarrow \beta | PNS)$ é a probabilidade condicionada pela posição na *string*
    - P(e | a) não varia muito com a posição.
    - P(ent | ant) é altamente dependente da posição.
    - As pessoas raramente digitam *antler* como *entler*, mas frequentemente digitam *reluctant* como *reluctent*.

- **Modelo de canal Brill-Moore** (2000): Edições de *string* para *string*.
  - Seja Σ um alfabeto, o modelo permite todas as operações de edição da forma $\alpha \rightarrow \beta$, onde $\alpha, \beta \in \Sigma^*$.
  - $P(\alpha \rightarrow \beta)$ é a probabilidade de que, quando o usuário pretende digitar α, ele digite β em vez disso.
  - $P(\alpha \rightarrow \beta | PNS)$ é a probabilidade condicionada pela posição na *string*
    - P(e | a) não varia muito com a posição.
    - P(ent | ant) é altamente dependente da posição.
    - As pessoas raramente digitam *antler* como *entler*, mas frequentemente digitam *reluctant* como *reluctent*.

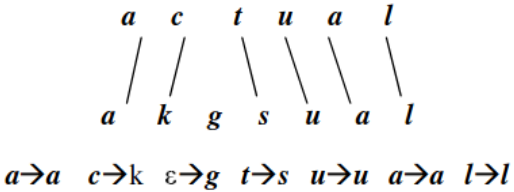


Figura 18: Alinhamento de *String* (Brill e Moore, 2000).

Brill, E. e Moore, R. C. (2000). *An Improved Error Model for Noisy Channel Spelling Correction*.

- **Aspell** (2000): Combina correção ortográfica e fonética.
  - Hashing para Verificação Ortográfica: Busca eficiente de candidatos usando tabelas hash.
  - Algoritmo Metaphone: Trata correções fonéticas ao combinar palavras que soam semelhantes.
  - Estratégia de Quase Erro do Ispell:
    - Foca na distância de edição 1 para reduzir o espaço de busca.
    - Filtragem precoce do dicionário: Elimina candidatos inválidos durante a geração.

- **Aspell** (2000): Combina correção ortográfica e fonética.
  - Hashing para Verificação Ortográfica: Busca eficiente de candidatos usando tabelas hash.
  - Algoritmo Metaphone: Trata correções fonéticas ao combinar palavras que soam semelhantes.
  - Estratégia de Quase Erro do Ispell:
    - Foca na distância de edição 1 para reduzir o espaço de busca.
    - Filtragem precoce do dicionário: Elimina candidatos inválidos durante a geração.

- **Aspell** (2000): Combina correção ortográfica e fonética.
  - Hashing para Verificação Ortográfica: Busca eficiente de candidatos usando tabelas hash.
  - Algoritmo Metaphone: Trata correções fonéticas ao combinar palavras que soam semelhantes.
  - Estratégia de Quase Erro do Ispell:
    - Foca na distância de edição 1 para reduzir o espaço de busca.
    - Filtragem precoce do dicionário: Elimina candidatos inválidos durante a geração.

Exemplo - Tratamento de Homófonos no Aspell

- Palavra com erro de ortografia: `ther`
- Candidatos: `there`, `their`, `they're`

1. Metaphone
   - O algoritmo Metaphone transforma palavras em códigos fonéticos com base na pronúncia.
   - Códigos fonéticos para as palavras candidatas:
     - `there` → `0R`
     - `their` → `0R`
     - `they're` → `0R`
   - Homófonos compartilham o mesmo código (`0R`).

Exemplo - Tratamento de Homófonos no Aspell

- Palavra com erro de ortografia: `ther`
- Candidatos: `there`, `their`, `they're`

1. Metaphone
   - O algoritmo Metaphone transforma palavras em códigos fonéticos com base na pronúncia.
   - Códigos fonéticos para as palavras candidatas:
     - `there` → `0R`
     - `their` → `0R`
     - `they're` → `0R`
   - Homófonos compartilham o mesmo código (`0R`).

2. Fluxo de Trabalho:
   - Entrada: Palavra com erro de ortografia `ther`.
   - Etapa 1: Gerar candidatos usando **distância de edição 2 ou menos**:
     - Candidatos: `there`, `their`, `thee`, `thor`, `her`, `the`, `they're`.
   - Etapa 2: Calcular códigos Metaphone para todos os candidatos:
     - Candidatos foneticamente semelhantes a `ther` (`OR`) têm classificação mais alta: `there`, `their`, `thor`, `they're`.
   - Etapa 3: Classificar e sugerir com base em:
     - Frequência da palavra, Distância de edição, Similaridade fonética, Probabilidade de erro.
3. Limitações:
   - Metaphone combina palavras pelo som, mas carece de **compreensão contextual**.
   - Exemplo:
     - Entrada: *"Eu fui à ther house."*
     - Sugestões: `thee`, `their`, `there`, `therm`, `the`, `her`, `Thar`, `Thea`, `Thor`, `Thur`.
     - O Aspell não pode inferir a palavra correta (`their`) sem considerar o contexto da frase.

2. Fluxo de Trabalho:
   - Entrada: Palavra com erro de ortografia `ther`.
   - Etapa 1: Gerar candidatos usando **distância de edição 2 ou menos**:
     - Candidatos: `there`, `their`, `thee`, `thor`, `her`, `the`, `they're`.
   - Etapa 2: Calcular códigos Metaphone para todos os candidatos:
     - Candidatos foneticamente semelhantes a `ther` (`OR`) têm classificação mais alta: `there`, `their`, `thor`, `they're`.
   - Etapa 3: Classificar e sugerir com base em:
     - Frequência da palavra, Distância de edição, Similaridade fonética, Probabilidade de erro.
3. Limitações:
   - Metaphone combina palavras pelo som, mas carece de **compreensão contextual**.
   - Exemplo:
     - Entrada: *"Eu fui à ther house."*
     - Sugestões: `thee`, `their`, `there`, `therm`, `the`, `her`, `Thar`, `Thea`, `Thor`, `Thur`.
     - O Aspell não pode inferir a palavra correta (`their`) sem considerar o contexto da frase.

GNU Aspell

- **Hunspell** (2002): Analisador morfológico com regras de afixos e correspondência fonética.

- Análise Morfológica:
  - Suporta línguas complexas com rica morfologia (por exemplo, húngaro, turco, finlandês).
  - Trata raízes de palavras, prefixos e sufixos usando regras de afixos.
- Sistema de Dicionário:
  - Dois componentes:
    1. Arquivo de Dicionário: Contém formas raiz das palavras.
    2. Arquivo de Afixos: Define regras para combinar raízes com prefixos/sufixos.
- Distância de Levenshtein:
  - Usa *distância de edição* para gerar e classificar correções candidatas.
- Correspondência Fonética:
  - Usa um algoritmo de transcrição fonética baseado em tabela, emprestado do Aspell. É útil para línguas com ortografias que não são baseadas na pronúncia.
- Similaridade n-gram:
  - Melhora as sugestões.

- **Hunspell** (2002): Analisador morfológico com regras de afixos e correspondência fonética.

- Análise Morfológica:
  - Suporta línguas complexas com rica morfologia (por exemplo, húngaro, turco, finlandês).
  - Trata raízes de palavras, prefixos e sufixos usando regras de afixos.
- Sistema de Dicionário:
  - Dois componentes:
    1. Arquivo de Dicionário: Contém formas raiz das palavras.
    2. Arquivo de Afixos: Define regras para combinar raízes com prefixos/sufixos.
- Distância de Levenshtein:
  - Usa *distância de edição* para gerar e classificar correções candidatas.
- Correspondência Fonética:
  - Usa um algoritmo de transcrição fonética baseado em tabela, emprestado do Aspell. É útil para línguas com ortografias que não são baseadas na pronúncia.
- Similaridade n-gram:
  - Melhora as sugestões.

- **Hunspell** (2002): Analisador morfológico com regras de afixos e correspondência fonética.

- Análise Morfológica:
  - Suporta línguas complexas com rica morfologia (por exemplo, húngaro, turco, finlandês).
  - Trata raízes de palavras, prefixos e sufixos usando regras de afixos.
- Sistema de Dicionário:
  - Dois componentes:
    1. Arquivo de Dicionário: Contém formas raiz das palavras.
    2. Arquivo de Afixos: Define regras para combinar raízes com prefixos/sufixos.
- Distância de Levenshtein:
  - Usa *distância de edição* para gerar e classificar correções candidatas.
- Correspondência Fonética:
  - Usa um algoritmo de transcrição fonética baseado em tabela, emprestado do Aspell. É útil para línguas com ortografias que não são baseadas na pronúncia.
- Similaridade n-gram:
  - Melhora as sugestões.

- **Hunspell** (2002): Analisador morfológico com regras de afixos e correspondência fonética.

- Análise Morfológica:
  - Suporta línguas complexas com rica morfologia (por exemplo, húngaro, turco, finlandês).
  - Trata raízes de palavras, prefixos e sufixos usando regras de afixos.
- Sistema de Dicionário:
  - Dois componentes:
    1. Arquivo de Dicionário: Contém formas raiz das palavras.
    2. Arquivo de Afixos: Define regras para combinar raízes com prefixos/sufixos.
- Distância de Levenshtein:
  - Usa *distância de edição* para gerar e classificar correções candidatas.
- Correspondência Fonética:
  - Usa um algoritmo de transcrição fonética baseado em tabela, emprestado do Aspell. É útil para línguas com ortografias que não são baseadas na pronúncia.
- Similaridade n-gram:
  - Melhora as sugestões.

- **Hunspell** (2002): Analisador morfológico com regras de afixos e correspondência fonética.

- Análise Morfológica:
  - Suporta línguas complexas com rica morfologia (por exemplo, húngaro, turco, finlandês).
  - Trata raízes de palavras, prefixos e sufixos usando regras de afixos.
- Sistema de Dicionário:
  - Dois componentes:
    1. Arquivo de Dicionário: Contém formas raiz das palavras.
    2. Arquivo de Afixos: Define regras para combinar raízes com prefixos/sufixos.
- Distância de Levenshtein:
  - Usa *distância de edição* para gerar e classificar correções candidatas.
- Correspondência Fonética:
  - Usa um algoritmo de transcrição fonética baseado em tabela, emprestado do Aspell. É útil para línguas com ortografias que não são baseadas na pronúncia.
- Similaridade n-gram:
  - Melhora as sugestões.

- Suporte Multilíngue:
  - Disponível para 98 idiomas com dicionários extensivos.

Aplicações:

- Integrado em ferramentas como LibreOffice, Firefox e Chrome para verificação ortográfica multilíngue.
- Suporta dicionários personalizados para campos especializados (por exemplo, médico, jurídico).

Hunspell no GitHub

- Suporte Multilíngue:
  - Disponível para 98 idiomas com dicionários extensivos.

Aplicações:

- Integrado em ferramentas como LibreOffice, Firefox e Chrome para verificação ortográfica multilíngue.
- Suporta dicionários personalizados para campos especializados (por exemplo, médico, jurídico).

Hunspell no GitHub

- Suporte Multilíngue:
  - Disponível para 98 idiomas com dicionários extensivos.

Aplicações:

- Integrado em ferramentas como LibreOffice, Firefox e Chrome para verificação ortográfica multilíngue.
- Suporta dicionários personalizados para campos especializados (por exemplo, médico, jurídico).

Hunspell no GitHub

- Suporte Multilíngue:
  - Disponível para 98 idiomas com dicionários extensivos.

Aplicações:

- Integrado em ferramentas como LibreOffice, Firefox e Chrome para verificação ortográfica multilíngue.
- Suporta dicionários personalizados para campos especializados (por exemplo, médico, jurídico).

Hunspell no GitHub

- **Algoritmo de Norvig** (2007): Usa a distância de Damerau-Levenshtein para gerar candidatos.

Principais Recursos:

- Distância de Edição:
  - Gera todas as palavras possíveis dentro de uma determinada distância de edição (por exemplo, 1 ou 2) a partir da palavra com erro de ortografia.
  - Trata inserção, deleção, substituição e transposição.
- Busca no Dicionário:
  - Filtra candidatos validando-os contra um dicionário de palavras.
- Classificação:
  - Classifica candidatos válidos com base em:
    - Frequência da Palavra: Palavras mais frequentes têm prioridade.
    - Probabilidade de Erros: Com base no Modelo de Canal Ruidoso (opcional).

How to Write a Spelling Corrector: https://norvig.com/spell-correct.html

- **Algoritmo de Norvig** (2007): Usa a distância de Damerau-Levenshtein para gerar candidatos.

Principais Recursos:

- Distância de Edição:
  - Gera todas as palavras possíveis dentro de uma determinada distância de edição (por exemplo, 1 ou 2) a partir da palavra com erro de ortografia.
  - Trata inserção, deleção, substituição e transposição.
- Busca no Dicionário:
  - Filtra candidatos validando-os contra um dicionário de palavras.
- Classificação:
  - Classifica candidatos válidos com base em:
    - Frequência da Palavra: Palavras mais frequentes têm prioridade.
    - Probabilidade de Erros: Com base no Modelo de Canal Ruidoso (opcional).

How to Write a Spelling Corrector: https://norvig.com/spell-correct.html

- **Algoritmo de Norvig** (2007): Usa a distância de Damerau-Levenshtein para gerar candidatos.

Principais Recursos:

- Distância de Edição:
  - Gera todas as palavras possíveis dentro de uma determinada distância de edição (por exemplo, 1 ou 2) a partir da palavra com erro de ortografia.
  - Trata inserção, deleção, substituição e transposição.
- Busca no Dicionário:
  - Filtra candidatos validando-os contra um dicionário de palavras.
- Classificação:
  - Classifica candidatos válidos com base em:
    - Frequência da Palavra: Palavras mais frequentes têm prioridade.
    - Probabilidade de Erros: Com base no Modelo de Canal Ruidoso (opcional).

How to Write a Spelling Corrector: https://norvig.com/spell-correct.html

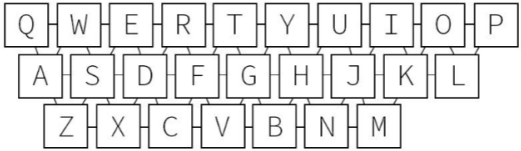
- **Algoritmo de Norvig** (2007): Usa a distância de Damerau-Levenshtein para gerar candidatos.

Principais Recursos:

- Distância de Edição:
  - Gera todas as palavras possíveis dentro de uma determinada distância de edição (por exemplo, 1 ou 2) a partir da palavra com erro de ortografia.
  - Trata inserção, deleção, substituição e transposição.
- Busca no Dicionário:
  - Filtra candidatos validando-os contra um dicionário de palavras.
- Classificação:
  - Classifica candidatos válidos com base em:
    - Frequência da Palavra: Palavras mais frequentes têm prioridade.
    - Probabilidade de Erros: Com base no Modelo de Canal Ruidoso (opcional).

How to Write a Spelling Corrector: https://norvig.com/spell-correct.html

- **Distância QWERTY de Levenshtein Ponderada**: leva em consideração a distância no teclado



	a	s	d	...	l	p
a	0	1	2	...	8	9
s	1	0	1	...	7	8
d	2	1	0	...	6	7
...	...	...	...	0	...	...
l	8	7	6	...	0	1
p	9	8	7	...	1	0

Figura 19: Teclado QWERTY e matriz de distância no teclado.

- A distância entre as teclas está no intervalo [0,9]. Elas são multiplicadas por 2/9.

- Deleção: ponderada pela média das distâncias para os caracteres adjacentes na *string*.
- Inserção: inalterada, peso 1.
- Substituição: ponderada de acordo com a distância entre o caractere que é removido e o caractere que é inserido.
- Transposição: inalterada, peso 1.

Samuelsson, 2017

- **Modelos Baseados em Redes Neurais**: Aproveitam o aprendizado profundo para detecção e correção avançadas de erros.
  - Utilizam técnicas de aprendizado profundo para melhorar a verificação ortográfica:
    - Redes Neurais Recorrentes (RNNs)
    - Embeddings de Palavras
    - Transformers
  - Consciência Contextual
  - Aprendizado a partir de Dados
  - Tratamento de Erros de Digitação

Exemplos:

- Smart Compose do Google
- Grammarly
- Microsoft Editor
- LanguageTool

Verificadores Ortográficos Específicos de Domínio

1. Médico
   - MedSpell: um aplicativo de correção ortográfica e autocorreção médica
   - OpenMedSpel (código aberto)
2. Programação
   - CodeSpell: projetado principalmente para verificar palavras com erro de ortografia em código-fonte
3. Aprendizado
   - Kidspell: Um verificador ortográfico orientado para crianças, baseado em regras e fonético
4. Acessibilidade
   - Real Check: Um verificador ortográfico para Dislexia
5. Dicionários Personalizados
   - Hunspell e Aspell: Adicionam vocabulários especializados

Referências

- Shannon, C. E. (1950). *Prediction and Entropy of Printed English*.
- Blair, C. R. (1960). *A program for correcting spelling errors*.
- Damerau, F. J. (1964). *A technique for computer detection and correction of spelling errors*.
- Levenshtein, V. I. (1966). *Binary codes capable of correcting deletions, insertions, and reversals*.
- Burkhard W. and Keller R. (1973). *Some approaches to best-match file searching*.
- Jaro, M. A. (1989). *Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida*.
- Winkler, W. E. (1990). *String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage*.
- Kernighan, M. D. et al. (1990). *A spelling correction program based on a noisy channel model*.
- Mays, E. et al. (1991). *Context based spelling correction*.
- Atkinson, K. (2000). *GNU Aspell*.
- Németh, L. (2002). *Hunspell*.
- Samuelsson, A. (2017). *Weighting Edit Distance to Improve Spelling Correction in Music Entity Search*.
- Brill, E. and Moore, R. C. (2000). *An Improved Error Model for Noisy Channel Spelling Correction*.
- Jurafsky, D., and Martin, J. H. (2024). *Speech and Language Processing*.