

# Processamento de Texto: o básico

Prof. Dr. Thiago de Paulo Faleiros

# Roteiro da apresentação

- 1 Expressão Regulares
- 2 Palavras e Corpora
- 3 Minimum Edit Distance

# Expressão Regular são úteis em todos os lugares

- Toda tarefa de processamento de texto
  - Não é uma solução geral em PLN (por esse motivo utilizamos grande sistemas de PLN que serão vistos ao longo desta disciplina).
  - Muito útil como parte dos sistemas de PLN (ex., pré-processamento ou formatação de texto).
- Necessária para análise de dados do tipo texto
- Ferramenta amplamente utilizada na indústria e academia

# Expressão Regular

- Uma linguagem formal para especificar cadeias de caracteres
- Como podemos pesquisar menções deste animal fofo no texto?
- woodchuck
- woodchucks
- Woodchuck
- Woodchucks
- Groundhog
- groundhogs



## Expressão Regular: Disjunções

Letras dentro de cochetes [ ]

Pattern	Matches
[wW]oodchuck	Woodchuck, woodchuck
[123456789]	Qualquer dígito

Intervalos usando traço [A – Z]

Pattern	Matches	
[A – Z]	Letras maiúsculas	Thiago de Paulo
[a – z]	Letras minúsculas	estou impaciente
[0 – 9]	Um simples dígito	Capítulo 1: Expressão Regulares

## Expressão Regulares: Negação em Disjunção

Circunflexo ('^' – *carat*) como primeiro caracter em [ ] é a **negação** da lista

- Nota: ^ significa negação só quando ele é o primeiro em [ ]
- Caracteres especiais (., \*, +, ?) perde seu significado dentro da lista [ ]

Pattern	Matches	Exemplos
[^A – Z]	Não Letras maiúsculasa	Thiago de Paulo
[^Ss]	Nenhum 'S' ou 's'	Estou impaciente
[^.]	Não é um ponto	Capítulo 1: Expressão Regulares
[e^]	Nenhum 'e' ou '^'	tudo ^ bem

## Expressão Regulares: Atalhos convenientes

Pattern	Expansion	Matches	Exemplos
<code>\d</code>	<code>[0-9]</code>	Qualquer dígito	Fahreheit <b>4</b> 51
<code>\D</code>	<code>[^0-9]</code>	Qualquer não dígito	100 <b>k</b> m
<code>\w</code>	<code>[a-zA-Z0-9_]</code>	Qualquer alfanumérico ou <code>_</code>	<b>T</b> hiago
<code>\W</code>	<code>[^\w]</code>	Nenhum alfanumérico ou <code>_</code>	Olhe <b>!</b>
<code>\s</code>	<code>[\r\t\n\f]</code>	Espaço em branco (epaço,tab)	Olhe <u>l</u> para cima!
<code>\S</code>	<code>[^\s]</code>	Não espaço em branco	<b>T</b> hiago Faleiros

## Expressão Regulares: Mais Disjunções

Marmota, em inglês pode ser *Groundhog* ou *Woodchuck*  
Utilize o símbolo '|' para disjunção

Pattern	Matches
groundhog woodchuck	woodchuck
yours mine	yours
a b c	= [abc]
[gG]groundhog [Ww]oodchuck	Woodchuck





## Caracteres Curinga: ., ?, \*, +

Pattern	Matches	Exemplo
beg.n	Qualquer caracter	begin begun beg3n beg_n
woodchucks?	s é opcional	woodchuck woodchucks
to*	0 ou mais do caracter anterior	t to too tooo
to+	1 ou mais do caracter anterior	to too tooo

## Expressão Regular: Âncoras ^ \$

Pattern	Matches
<code>^[A-Z]</code>	<b>P</b> alo Alto
<code>^[^A-Za-z]</code>	'Hellow'
<code>\.\$</code>	The end.
<code>.\$</code>	The end?      The end!

# Nota sobre Expressão Regulares em Python

- Regex e Python usam “\” para caracteres especiais. Você precisa de um “\” extra!
  - “\\d+” para procurar 1 ou mais dígitos
  - “\n” em Python significa nova linha. Precisaria de “\\n” para dois caracteres.
- Utilize *raw string notation* para regex:
  - r“[tT]he”
  - r“\d+” para um ou mais dígitos
    - em vez de “\\d+”

## Processo iterativo para escrita de Regex

Encontre todas as instâncias da palavra “the” no texto.

- `the`
  - Faltou a letra maiúscula
- `[tT]he`
  - Pode encontrar palavras como *Theology*
- `\W[tT]\W`

## Falso positivo e falso negativo

O processo foi baseado na correção de dois erros:

- 1 Não encontrar coisas que deveriam encontrar (The) – Falso Negativos
- 2 Encontrar strings que não deveriam ser encontradas (there, then, other) – Falso Positivos

## Caracterizando o trabalho em PLN

Em PLN nós sempre tratamos com esses erros

Reduzir a taxa de erro para uma aplicação frequentemente envolve dois esforços antagônicos:

- Aumentar a cobertura (ou recall) – Minimizando os falsos negativos
- Aumentando a acurácia (ou precisão) – Minimizando os falsos positivos

# Expressões Regulares são importantes!

Amplamente usadas na academia e Indústria

- 1 Parte da maioria das tarefas de processamento de texto, mesmo para grandes modelos de redes neurais.
  - Incluindo formatação e pré-processamento de texto
- 2 Muito útil para análise de dados do tipo texto

# Substituição

Substituição em Python e UNIX:

- `s/regex1/pattern/`
  - `s/colour/color/`



## Capturando grupos

- Queremos colocar ângulos (< >) em volta dos números:
  - the 35 boxes → the <35> boxes
- Use parênteses ( ) para capturar um padrão em indexado sequencialmente pela ocorrência (1, 2, 3, ...)
- Use \1 para referir ao conteúdo do primeiro padrão
  - s/([0-9]+)/<\1>/

## Capturando grupos: registros múltiplos

/the (.\*)er they (.\*) , the \1er we \2/

- Matches
  - *the faster they ran, the faster we ran*
- Mas não
  - *the faster they ran, the faster we ate*

## Suponha que não queremos capturar

Parênteses tem função dupla: agrupa e captura termos

- Não capturando grupos: adicione ?: depois do parênteses:  
`/(?:some|a few) (people|cats) like some \1/`
- matches
  - some cats like some cats
- mas não
  - some cats like some some

## Assertivas de Antevisão

- `(?= pattern)` é verdade se o padrão casa, mas sem incluir na correspondência final
- `(?! pattern)` verdade se o padrão não casa.
  - Como casar, no começo de uma linha, qualquer palavra simples que não comece com “Volcano”:  
`/^(?!Volcano)[A-Za-z]+/`

## Quantas palavras na sentença?

“I do uh main- mainly business data processing”

- Fragmentos e pausas

“Seuss’s cat in the hat is different from other cats!”

- **Lemma**: mesmo stem, part-of-speech e radical
  - cat e cats = mesmo lema
- **Wordform**: a palavra flexionada completa
  - cat e cats = diferentes wordforms

## Quantas palavras na sentença?

they lay back on the San Francisco grass and looked at the stars  
and their

- Tipo: um elemento do vocabulário
- Token: uma instância do tipo no texto

Quantas palavras?

- 15 tokens (ou 14)
- 13 tipos (ou 12 ou 11?)

## Quantas palavras na sentença?

- **N** = número de tokens
- **V** = vocabulário = conjunto de tipos,  $|V|$  é o tamanho do vocabulário
- Heaps Law = Herdan's Law =  $|V| = kN^\beta$  onde  $0.67 < \beta < 0.75$ .

O tamanho do vocabulário cresce maior do que a raiz quadrada do número de tokens

	Tokens= $N$	Tipos= $ V $
Conversas telefônicas	2.4 milhões	20 mil
Shakespeare	884 mil	31 mil
COCA	440 milhões	2 milhões
Google N-grams	1 trilhão	13+ milhões

# Corpora

As palavras vem de algum lugar!

Texto é produzido por:

- um escrito específico
- em um determinado momento
- de um determinado gênero
- em uma linguagem específica
- para uma determinada função



## Corpora varia conforme dimensões

- linguagens – 7097 linguagens no mundo
- Variedade – Variações de linguagens
  - iont (I don't)
- Múltiplas linguagens:  
*Por primeira vez veo a @username actually being hateful! it was beautiful :)*
- Gênero: notícias, ficção, artigos científicos, Wikipedia
- Autor demográfico: idade, gênero, etnia, ..

# Tabela do Corpus

- Motivação:
  - Qual o motivo da coleta do corpus?
  - Por quem?
  - Quem financiou?
- Situação: Em que situação o texto foi escrito?
- Processo de coleta: Se é uma subamostra? Como foi amostrada? Existe pré-processamento?
  - processo de anotação, variedade linguística, demográfica, etc...

# Normalização do texto

Toda tarefa de PLN requiere normalização de texto:

- 1 Tokenização (segmentação) de palavras
- 2 Normalização do formato das palavras
- 3 Segmentação das sentenças

## Segmentação baseada em espaço

A forma mais simples de tokenização

- Para linguagens que utilizam espaço em branco entre palavras
  - Árabe, Grego, latim, etc.
- Segmenta tokens entre instâncias de espaços em branco
- comando “tr” em UNIX

```
$ tr -sc 'A-Za-z' '\n' < shakes.txt | sort | uniq -c
```

## Problemas com a tokenização

Não pode remover cegamente as pontuações:

- m.p.h, Ph.D, AT&T, cap'n
- preços (\$45.55)
- datas (01/02/06)
- URLs (<http://unb.br>)
- hashtags ([#nlproc](#))
- e-mails ([thiagodepaulo@unb.br](mailto:thiagodepaulo@unb.br))

Clítico: uma palavra que não depende de outra foneticamente ou gramaticamente

- “are” (we're), “je” (j'ai), “le” (l'honneur), me (viu-me)

Expressões com múltiplas palavras são palavras únicas?

- New York, Rock'n roll

# Tokenização com o NLTK

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)      # set flag to allow verbose regexps
...     ([A-Z]\.)+          # abbreviations, e.g. U.S.A.
...     | \w+(-\w+)*        # words with optional internal hyphens
...     | \$?\d+(\.\d+)?%?   # currency and percentages, e.g. $12.40, 82%
...     | \.\.\.            # ellipsis
...     | [][.,;"'()?:_-'] # these are separate tokens; includes ], [
...     '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

# Tokenização de linguagens sem espaço

Várias linguagens (como o Chinês, Japonês, Tailandês) não usam espaço para separar as palavras.

Como decidir onde ficam os limites dos tokens?

## Tokenização de palavras em Chinês

- Palavras em chinês são compostas por caracteres chamados “hanzi”.
- Cada um representa uma unidade semântica chamada morfema.
- Cada palavra tem em média 2.4 morfemas.
- Mas decidir o que vale como uma palavra é complexo.



# Como fazer tokenização de palavras em Chinês?

姚明进入总决赛 “Yao Ming reaches the finals”

3 words?

姚明 进入 总决赛

YaoMing reaches finals

5 words?

姚 明 进入 总 决赛

Yao Ming reaches overall finals

7 characters? (don't use words at all):

姚 明 进 入 总 决 赛

Yao Ming enter enter overall decision game

# Tokenização

Em chinês é comum apenas tratar cada caracter como um token.

- O processo de segmentação é simples

Em outras linguagens (Como o japonês), é necessário um processo complexo de segmentação.

- O algoritmo padrão utiliza modelo de redes neurais sequenciais treinado por um técnicas de aprendizagem de máquina supervisionada.

## Outras opções de Tokenização

Em vez de

- Segmentação por espaços em branco
- Segmentação por caracteres simples

Usar os próprios dados para dizer como tokenizar.

Tokenização em subpalavras

- Tokens podem ser partes da palavra ou toda a palavra.

# Tokenização em subpalavras

Três algoritmos comuns:

- Byte-Pair Encoding (BPE) (*Sennrich et al., 2016*)
- Unigram language modeling tokenization (*Kudo, 2018*)
- WordPiece (*Schuster and Nakajima, 2012*)

Todos tem duas partes:

- Aprender o que são tokens a partir do corpus cru e induzir um vocabulário (um conjunto de tokens)
- Um segmentador de tokens que pega uma sentença de teste e tokeniza ela de acordo com o vocabulário criado.

# Byte Pair Encoding (BPE)

Seja o vocabulário o conjunto de todos os caracteres =  $\{A, B, C, D, \dots, a, b, c, d, \dots\}$ .

Repita:

- Escolha dois símbolos que mais ocorrem frequentemente adjacentes (digamos 'A' e 'B').
- Adicione um novo símbolo da união, formando 'AB', no vocabulário.
- Substitua cada adjacência dos símbolos 'A' e 'B' no corpus pelo novo símbolo 'AB'.

Faça esse processo até  $k$  uniões de símbolos serem realizadas.

# Algoritmo BPE

**function** BYTE-PAIR ENCODING(strings  $C$ , number of merges  $k$ ) **returns** vocab  $V$

$V \leftarrow$  all unique characters in  $C$                       # initial set of tokens is characters

**for**  $i = 1$  **to**  $k$  **do**                                      # merge tokens til  $k$  times

$t_L, t_R \leftarrow$  Most frequent pair of adjacent tokens in  $C$

$t_{NEW} \leftarrow t_L + t_R$                               # make new token by concatenating

$V \leftarrow V + t_{NEW}$                               # update the vocabulary

    Replace each occurrence of  $t_L, t_R$  in  $C$  with  $t_{NEW}$       # and update the corpus

**return**  $V$

# Algoritmo BPE

- A maioria das subpalavras estão separadas por espaço
- Então é comumente adicionado um símbolo ' \_ ' de fim-de-palavra antes do espaço no corpus de treinamento.

# Algoritmo BPE

Corpus original (e muito fascinante :P )

low low low low low lowest lowest newer newer newer newer  
newwer wider wider wider new new

Adiciona o token de fim-de-palavra, resultando no seguinte vocabulário:

- Vocabulário:

\_, d, e, i, l, n, o, r, s, t, w



# Algoritmo BPE

## corpus

5 l o w \_  
2 l o w e s t \_  
6 n e w e r \_  
3 w i d e r \_  
2 n e w \_

## vocabulário

\_, d, e, i, l, n, o, r, s, t, w

Unindo e r para er

## corpus

5 l o w \_  
2 l o w e s t \_  
6 n e w e r \_  
3 w i d e r \_  
2 n e w \_

## vocabulário

\_, d, e, i, l, n, o, r, s, t, w, er

# Algoritmo BPE

## corpus

5 l o w \_  
2 l o w e s t \_  
6 n e w e r \_  
3 w i d e r \_  
2 n e w \_

## vocabulário

\_, d, e, i, l, n, o, r, s, t, w, er

Unindo **er \_** para **er\_**

## corpus

5 l o w \_  
2 l o w e s t \_  
6 n e w e r\_  
3 w i d e r\_  
2 n e w \_

## vocabulário

\_, d, e, i, l, n, o, r, s, t, w, er, er\_

# Algoritmo BPE

## corpus

5 l o w \_  
2 l o w e s t \_  
6 n e w e r \_  
3 w i d e r \_  
2 n e w \_

## vocabulário

\_, d, e, i, l, n, o, r, s, t, w, er, er\_

Unindo **n e** para **ne**

## corpus

5 l o w \_  
2 l o w e s t \_  
6 n e w e r \_  
3 w i d e r \_  
2 n e w \_

## vocabulário

\_, d, e, i, l, n, o, r, s, t, w, er, er\_, ne

# Algoritmo BPE

- Unindo **ne w** para **new**
- Unindo **i o** para **lo**
- Unindo **lo w** para **low**
- Unindo **new er\_** para **newer\_**
- Unindo **low \_** para **low\_**

## corpus

5 low\_  
2 low e s t \_  
6 newer\_  
3 w i d er\_  
2 new \_

## vocabulário

\_, d, e, i, l, n, o, r, s, t, w, er, er\_, ne, new,  
lo, low, newer\_, low\_

## Propriedades dos tokens do BPE

- Normalmente incluem palavras frequentes
- Incluem também subpalavras frequentes
  - Que são morfemas, como -est ou -er
- Um morfema é a menor unidade portadora de significado de uma língua
  - 'unlikeliest' tem 3 morfemas 'un-', 'likely' e '-est'.

# Normalização de palavras

Colocando palavras/tokens em um formato padrão

- U.S.A. ou USA
- uhhuh ou hu-huh
- Fed ou fed
- am, is, be, are

# Capitalizar palavras

- Aplicações de Recuperação de Informação: reduz tudo para minúsculo
  - Os usuários tendem a usar letras minúsculas
  - Casos possíveis de palavras maiúsculas no meio de sentenças
    - General Motors
    - Fed vs fed
    - SAIL vs sail
- Para análise de sentimentos, Mineração de texto, extração de informação
  - Palavras maiúsculas podem ser importantes

# Lematização

- Representa todas as palavras com os seus lemas e radicais
  - am, are, is → be
  - car, cars, car's, cars' → car
  - quero, quieres → querer
  - correram → correr
  - maiores → grande
- “Ele está lendo histórias de detetive.”
  - → “Ele estar ler história de detetive.”



# Lematização é feita por análise morfológica

## Morfemas:

- Menores unidades de significação que compõem as palavras
- **Stems:** As unidades centrais portadoras de significado
- **Afixos:** Partes que aderem ao stems, geralmente com funções gramaticais

## Analizador Morfológico (Morphological Parsers):

- 'cats' tem dois morfemas 'cat' e 's'
- 'amarão' tem o morfema 'amar' e características morfológicas terceira pessoa no plural e futuro do presente.

# Stemming

Reduzindo os termos para os stems, removendo sufixo e flexões, deixando apenas as raízes que transmitem o sentido básico das palavras

- “Este não era o mapa que encontramos no baú de Billy Bones, mas uma cópia precisa, completa em todas as coisas - nomes e alturas e sondagens - com a única exceção das cruces vermelhas e das notas escritas.”
- → “Este não era o mapa que encontr no baú de Billy Bones, mas uma cóp precis, complet em tod as cois - nom e altur e sond - com a únic exceç das cruz vermelh e das not escrit.”

# Porter Stemmer

- O algoritmo de Porter divide o processo de stemming em várias etapas.
- Cada etapa contém regras que identificam padrões específicos nos sufixos das palavras.
- A aplicação dessas regras é sequencial, e, em cada etapa, certas condições precisam ser satisfeitas para que uma substituição ocorra.

Algumas regras:

- ATIONAL  $\rightarrow$  ATE (ex.: relational  $\rightarrow$  relate)
- ING  $\rightarrow$   $\epsilon$  (ex.: motoring  $\rightarrow$  motor, se o stem contém vogal)
- SSES  $\rightarrow$  SS (ex.: grasses  $\rightarrow$  grass)

# Stemming em Português

- Porter Stemmer adaptado para o português
  - Snowball Stemmer para português é uma extensão do algoritmo de Porter.
  - Mais simples e rápido!
- RSLP (Removedor de Sufixos da Língua Portuguesa)
  - O RSLP segue um processo baseado em várias fases, onde diferentes tipos de sufixos são removidos
  - Considerado o algoritmo mais adequado para stemming em português devido à sua complexidade e tratamento detalhado dos sufixos da língua.

## Segmentação de Sentenças

- símbolos como '!' e '?' são muito ambíguas, mas o ponto final '.' é muito mais ambiguo.
  - Pode ser os limites de uma sentença
  - Abreviação (Dr., Prof.)
  - Números (.02%, 3.14)
- Algoritmos comuns: Tokenizar primeiro depois utilize regras ou algoritmos de ML para classificar pontos como (a) parte da palavra ou (b) limites da sentença.
  - Utilizar dicionários com abreviações pode ajudar
- Segmentação de sentenças podem frequentemente ser feito por regras baseadas nas tokenizações.

# O quanto similar são duas strings?

- Corretor automático
- Biologia Computacional
- Tradução de Máquina
- Extração de Informação
- Reconhecimento de fala

# Edit Distance

- O número mínimo de edições entre duas strings
- O número de operação de edições
  - Inserção
  - Deleção
  - Substituição
- Necessário para transformar uma string em outra

# Minimum Edit Distance

- Alinhando duas Strings

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N



# Minimum Edit Distance

- Alinhando duas Strings

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N
d	s	s		i	s				

- Se cada operação tem custo 1
  - Distância será 5
- Se substituição tem custo 2 (Distância de Levenshtein)
  - Distância será 8

## Outros usos da Edit Distance em PLN

- Avaliação de Tradução de Máquina e reconhecimento de fala
  - *Spokesman confirms senior government adviser was shot*
  - *Spokesman said the senior adviser was shot dead*
- Extração de Entidades Nomeadas e Conferência de Entidades
  - IBM Inc. announced today
  - IBM profits
  - Stanford President John Hennessy announced yesterday
  - for Stamford University President John Hennessy

# Como encontrar a Mínima Distância de Edição?

- Procurando um caminho (sequência de edições) da string inicial até a final
  - **Estado inicial:** A palavra que estamos transformando
  - **Operador:** inserção, deleção, substituição
  - **Estabo objetivo:** A palavra que estamos querendo obter
  - **Custo do caminho:** O que queremos minimizar: o número de edições

## Edições Mínimas como busca

- O espaço de todas as sequências de edições é enorme!
  - Não podemos nos dar ao luxo de navegar ingenuamente
  - Vários caminhos distintos acabam no mesmo estado
    - Não precisamos registrar todos
    - Apenas o caminho mais curto para cada estado

## Definindo a Distância mínima de Edição

- Para duas Strings
  - $X$  de tamanho  $n$
  - $Y$  de tamanho  $m$
- Nos definimos  $D(i, j)$ 
  - A distância entre  $X[1 \dots i]$  e  $Y[1 \dots j]$ 
    - *i.e.*, os primeiros  $i$  caracteres de  $X$  e os primeiros  $j$  de  $Y$ .
  - A distância de edição entre  $X$  e  $Y$  é então  $D(n, m)$ .

# Programação Dinâmica para Minimum Edit Distance

- **Programação Dinâmica:** Computação de uma tabela que armazena os valores de  $D(n, m)$
- Resolvendo problemas pela combinação de soluções de subproblemas
- Bottom-up
  - Computamos  $D(i, j)$  para pequenos valores de  $i$  e  $j$
  - Some grandes valores de  $D(i, j)$  baseado nos valores menores computados anteriormente
  - *i.e.*, compute  $D(i, j)$  para todo  $i(0 < i < n)$  e  $j(0 < j < m)$ .

## Definindo Distância de Edições (Levenshtein)

- Inicialização

$$D(i, 0) = i$$

$$D(0, j) = j$$

- Relação de Recorrência

Para cada  $i = 1 \dots m$

Para cada  $j = 1 \dots n$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 \\ D(i, j-1) + 1 \\ D(i-1, j-1) + 2 \text{ se } X(i) \neq X(j) \\ D(i-1, j-1) + 0 \text{ se } X(i) = Y(j) \end{cases}$$

- Finalização  $D(n, m)$  é a distância

# Tabela de Distância de Edição


N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N



# Tabela de Distância de Edição

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$



# Tabela de Distância de Edição

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

# Tabela de Distância de Edição

N	9	8	9	10	11	12	11	10	9	8
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

# Computando o alinhamento

- Distância de Edição não é suficiente
  - Frequentemente é necessário alinhar cada caracter de cada string
- Isso é feito fazendo o “backtrace”
- Toda vez que entramos em uma célula, lembre-se de onde veio
- Quando alcançado o fim,
  - Volte o caminho do canto da parte superior direita para definir o alinhamento

# Tabela de Distância de Edição

<b>n</b>	9	↓ 8	↙↘ 9	↙↘ 10	↙↘ 11	↙↘ 12	↓ 11	↓ 10	↓ 9	↙ <b>8</b>	
<b>o</b>	8	↓ 7	↙↘ 8	↙↘ 9	↙↘ 10	↙↘ 11	↓ 10	↓ 9	↙ <b>8</b>	← 9	
<b>i</b>	7	↓ 6	↙↘ 7	↙↘ 8	↙↘ 9	↙↘ 10	↓ 9	↙ <b>8</b>	← 9	← 10	
<b>t</b>	6	↓ 5	↙↘ 6	↙↘ 7	↙↘ 8	↙↘ 9	↙ <b>8</b>	← 9	← 10	↘ 11	
<b>n</b>	5	↓ 4	↙↘ 5	↙↘ 6	↙↘ 7	↙↘ <b>8</b>	↙↘ 9	↙↘ 10	↙↘ 11	↘ 10	
<b>e</b>	4	↙ 3	← 4	↙↘ <b>5</b>	← <b>6</b>	← 7	↘ 8	↙↘ 9	↙↘ 10	↓ 9	
<b>t</b>	3	↙↘ 4	↙↘ <b>5</b>	↙↘ 6	↙↘ 7	↙↘ 8	↙ 7	← 8	↙↘ 9	↓ 8	
<b>n</b>	2	↙↘ <b>3</b>	↙↘ 4	↙↘ 5	↙↘ 6	↙↘ 7	↙↘ 8	↓ 7	↙↘ 8	↙ 7	
<b>i</b>	<b>1</b>	↙↘ 2	↙↘ 3	↙↘ 4	↙↘ 5	↙↘ 6	↙↘ 7	↙ 6	← 7	← 8	
#	<b>0</b>	1	2	3	4	5	6	7	8	9	
	#	e	x	e	c	u	t	i	o	n	

## Adicionando o Backtrace

- Condição Base:

$$D(i, 0) = i$$

$$D(0, j) = j$$

- Relação de Recorrência

Para cada  $i = 1 \dots m$

Para cada  $j = 1 \dots n$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 \\ D(i, j-1) + 1 \\ D(i-1, j-1) + 2 \text{ se } X(i) \neq X(j) \\ D(i-1, j-1) + 0 \text{ se } X(i) = Y(j) \end{cases}$$

$$ptr(i, j) = \min \begin{cases} \text{ESQUERDA} & \text{inserção} \\ \text{DIREITA} & \text{deleção} \\ \text{DIAGONAL} & \text{substituição} \end{cases}$$

## Resultado do Backtrace

- As duas Strings e seu alinhamento:

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N

# Desempenho

- Tempo:  $O(nm)$
- Espaço:  $O(nm)$
- Backtrace  $O(n + m)$



# Distância de Edição ponderada

- Qual o motivo para adicionar pesos na computação da Edit Distance?
  - Corretor gramatikal: Algumas letras são mais prováveis de serem erradas do que outras
  - Biologia: Certos tipos de deleção ou inserção são mais prováveis do que outras

# Matriz de Confusão para corretor gramatical

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

## Adicionando o Backtrace

- Inicialização:

$$D(0, 0) = 0$$

$$D(i, 0) = D(i - 1, 0) + del[X(i)] \quad 1 < i \leq n$$

$$D(0, j) = D(0, j - 1) + ins[Y(j)] \quad 1 < j \leq m$$

- Relação de Recorrência

$$D(i, j) = \min \begin{cases} D(i - 1, j) + 1 & +del[X(i)] \\ D(i, j - 1) + 1 & +ins[X(i)] \\ D(i - 1, j - 1) + 2 & +sub[X(i), Y(j)] \end{cases}$$

- Finalização  $D(n, m)$  é a distância