DD1418
3a: Basic text processing

Johan Boye, KTH

Tokens

A *token* is a meaningful minimal unit of text.

Usually, *spaces* and *punctuation* delimit tokens. But:

- `http://www.kth.se`
- `jboye@kth.se`
- `+46 (8) 12345678`
- `123.456.78.23`
- `e.g.`
- `J.P. Morgan & co`

The exact definition of a token is application-dependent.

- Sometimes remove punctuation (e.g. search engines)
- Sometimes keep punctuation (most other applications)

Simple tokenization using Unix tools

Many software packages perform tokenization.

Simple tokenization can be done using the Unix tools `cat` and `tr`.

Normalization

Sometimes we want to put each word in a 'normal' form

- Abbreviations `U.S. US` → `U.S.`
- Case folding `Window, window` → `window`
- Diacritica `a,å,à,â` → `a`
- Umlaut `götze` → `goetze`

Need for normalization is highly dependent on the application.

Case folding using Unix tools

```
cat corpus.txt | tr 'A-ZÅÄÖ' 'a-zåäö'
```

Lemmatization and Stemming

Two more advanced normalization techniques:

Lemmatization Find the lemma (basic) form of a given word. Requires morphological analysis.

- `The boys' cars are different colors` → `The boy car be different color`

Stemming Heuristically chop off suffixes, e.g.
endings
ending
end
Simpler to implement, and quicker! But sometimes gives undesired results (e.g. for *stockings*).

Counting and searching

Counting and tokens and words can be done using the Unix tools `wc`, `sort` and `uniq`.

Searching can be done using regular expressions and `grep`.

DD1418
3b: String similarity and alignment

Johan Boye, KTH

String similarity

How similar are two strings?

This is useful to know in many contexts, e.g.:

- Spell checking
- Version control
- Plagiarism checking
- Evaluation of machine translation, question answering, ...
- Bioinfomatics (comparing DNA strings)

Spell checking

Finding misspelled words, and suggesting corrections.

Suppose we encounter *recieve**.

Which correction should be suggested?

receive

retrieve

review

...

Evaluation of question answering

Q: *Who is reponsible for cultural matters in the Swedish government?*

Anticipated answer: *Kultur- och idrottsminister Amanda Lind*.

QA system answers: *Kulturministern Amanda Lind.*

If answers are not identical, it can be useful to calculate how similar the given answer is to the anticipated answer.

Levenshtein distance (Minimal edit distance)

An *edit* is a *substitution*, an *insertion* or a *deletion*.

E.g. *recieve** $\Rightarrow$ *retrieve*

`recieve`		`recieve`	
$\Downarrow$ subst `c` by `t`	2	$\Downarrow$ insert `t`	1
`retieve`		`retcieve`	
$\Downarrow$ insert `r`	1	$\Downarrow$ subst `c` by `r`	2
`retrieve`		`retrieve`	

Alignment (länkning):

r	e	c		i	e	v	e
|	|			|	|	|	|
r	e	t	r	i	e	v	e

r	e		c	i	e	v	e
|	|			|	|	|	|
r	e	t	r	i	e	v	e

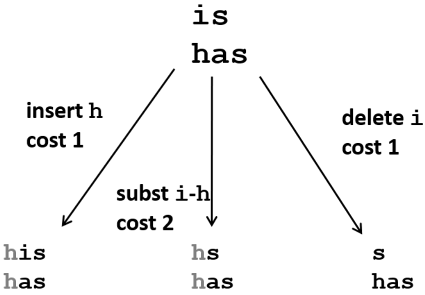
Levenshtein distance (Minimal edit distance)

`recieve`		`recieve`	
$\Downarrow$ subst `c` by `t`	2	$\Downarrow$ insert `t`	1
`retieve`		`retcieve`	
$\Downarrow$ insert `r`	1	$\Downarrow$ subst `c` by `r`	2
`retrieve`		`retrieve`	

If substitution costs 2, and insertion and deletion cost 1 each, the total cost (or distance) is 3.

Computing Levenshtein distance

A search problem: Find a path from one string to the other.
Count the cost of the necessary operations.

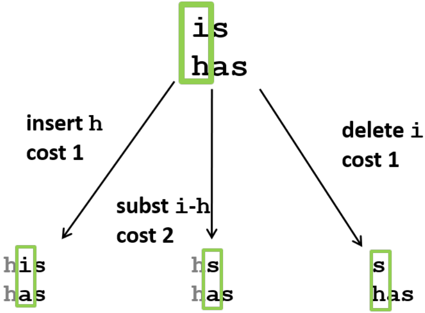


etc. Expand the tree until both strings are empty.

Total cost = the min of the costs in all branches.

Computing Levenshtein distance

A search problem: Find a path from one string to the other.
Count the cost of the necessary operations.

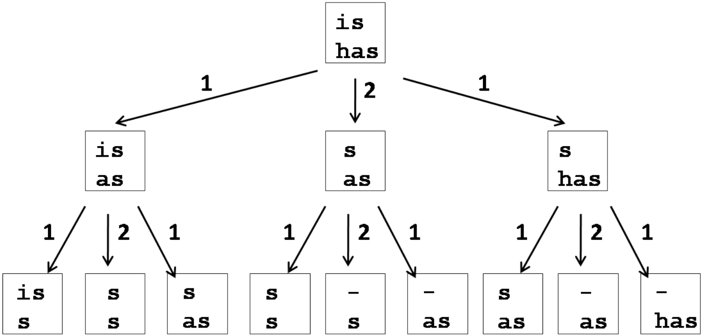


etc. Expand the tree until both strings are empty.

Total cost = the min of the costs in all branches.

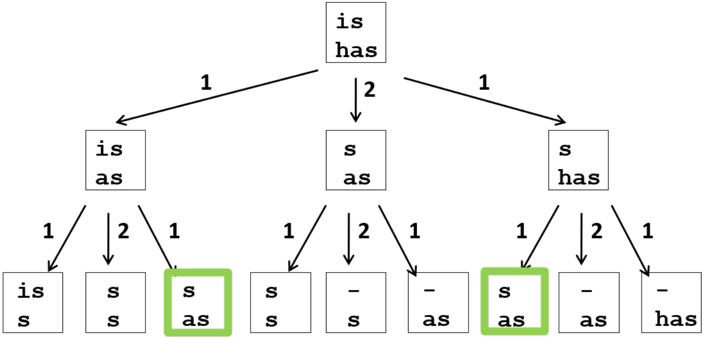
Computing Levenshtein distance

In fact, it makes sense to just advance string pointers (rather than talking about insertions, substitutions, and deletions).



Computing Levenshtein distance

In fact, it makes sense to just advance string pointers (rather than talking about insertions, substitutions, and deletions).



Computing Levenshtein distance

A naive exploration of the search tree results in an algorithm with exponential complexity.

Instead we will use a *dynamic programming* approach.

For two strings s and t, we define

$D(i, k)$ = *the Levenshtein distance between the first* i *characters of* s*, and the first* k *characters of* t.

The distance between s and t is thus

$$D(length(s), length(t))$$

Computing Levenshtein distance

$D(i, k)$ = the Levenshtein distance between the first i characters of s, and the first k characters of t.

Recursive definition:

$$
\begin{array}{ll}
D(i,0) = i & 0 \leq i \leq length(s) \\
D(0,k) = k & 0 \leq k \leq length(t) \\
D(i,k) = & \min \left\{ \begin{array}{l} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1)+\left\{ \begin{array}{ll} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{array} \right. \end{array} \right.
\end{array}
$$

Computing Levenshtein distance

	#	e	l	l	e	r
#						
f						
l						
e						
r						
a						

Computing Levenshtein distance

	#	e	l	l	e	r
#						
f						
l						
e						
r						
a						

$$D(i,0) = i \qquad 0 \le i \le length(s)$$
$$D(0,k) = k \qquad 0 \le k \le length(t)$$
$$D(i,k) = \min \begin{cases} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \begin{cases} 2 & \text{if } s[i] \ne t[k] \\ 0 & \text{if } s[i] = t[k] \end{cases} \end{cases}$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0					
f						
l						
e						
r						
a						

$$D(i,0) = i \qquad 0 \leq i \leq length(s)$$
$$D(0,k) = k \qquad 0 \leq k \leq length(t)$$
$$D(i,k) = \min \begin{cases} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \begin{cases} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{cases} \end{cases}$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0					
f	1					
l						
e						
r						
a						

$$
\begin{array}{ll}
D(i,0) = i & 0 \leq i \leq \mathit{length}(s) \\
D(0,k) = k & 0 \leq k \leq \mathit{length}(t) \\
D(i,k) = & \min \begin{cases} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \begin{cases} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{cases} \end{cases}
\end{array}
$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0					
f	1					
l	2					
e						
r						
a						

$$
\begin{array}{ll}
D(i,0) = i & 0 \leq i \leq \mathit{length}(s) \\
D(0,k) = k & 0 \leq k \leq \mathit{length}(t) \\
D(i,k) = & \min \left\{ \begin{array}{l} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \left\{ \begin{array}{ll} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{array} \right. \end{array} \right.
\end{array}
$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0					
f	1					
l	2					
e	3					
r						
a						

$$D(i,0) = i \qquad 0 \le i \le \mathit{length}(s)$$

$$D(0,k) = k \qquad 0 \le k \le \mathit{length}(t)$$

$$D(i,k) = \min \begin{cases} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \begin{cases} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{cases} \end{cases}$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0					
f	1					
l	2					
e	3					
r	4					
a						

$$D(i,0) = i \qquad 0 \leq i \leq length(s)$$
$$D(0,k) = k \qquad 0 \leq k \leq length(t)$$
$$D(i,k) = \min \begin{cases} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \begin{cases} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{cases} \end{cases}$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0					
f	1					
l	2					
e	3					
r	4					
a	5					

$$D(i,0) = i \qquad 0 \le i \le length(s)$$
$$D(0,k) = k \qquad 0 \le k \le length(t)$$
$$D(i,k) = \min \begin{cases} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \begin{cases} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{cases} \end{cases}$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1					
l	2					
e	3					
r	4					
a	5					

$$
\begin{array}{ll}
D(i,0) = i & 0 \leq i \leq length(s) \\
D(0,k) = k & 0 \leq k \leq length(t) \\
D(i,k) = & \min \left\{ \begin{array}{l} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \left\{ \begin{array}{ll} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{array} \right. \end{array} \right.
\end{array}
$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1					
l	2					
e	3					
r	4					
a	5					

$$
\begin{array}{ll}
D(i,0) = i & 0 \leq i \leq length(s) \\
D(0,k) = k & 0 \leq k \leq length(t) \\
D(i,k) = & \min \left\{ \begin{array}{l} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \left\{ \begin{array}{ll} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{array} \right. \end{array} \right.
\end{array}
$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2				
l	2					
e	3					
r	4					
a	5					

$$D(i,0) = i \qquad 0 \leq i \leq length(s)$$
$$D(0,k) = k \qquad 0 \leq k \leq length(t)$$
$$D(i,k) = \min \begin{cases} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \begin{cases} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{cases} \end{cases}$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2 2				
l	2					
e	3					
r	4					
a	5					

$$D(i,0) = i \qquad 0 \leq i \leq \mathit{length}(s)$$
$$D(0,k) = k \qquad 0 \leq k \leq \mathit{length}(t)$$
$$D(i,k) = \min \begin{cases} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \begin{cases} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{cases} \end{cases}$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1					
l	2					
e	3					
r	4					
a	5					

2 2
2

$$D(i,0) = i \qquad 0 \leq i \leq length(s)$$
$$D(0,k) = k \qquad 0 \leq k \leq length(t)$$
$$D(i,k) = \min \begin{cases} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \begin{cases} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{cases} \end{cases}$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2				
l	2					
e	3					
r	4					
a	5					

$$D(i,0) = i \qquad 0 \leq i \leq \mathit{length}(s)$$
$$D(0,k) = k \qquad 0 \leq k \leq \mathit{length}(t)$$
$$D(i,k) = \min \begin{cases} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \begin{cases} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{cases} \end{cases}$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2	3	4	5	6
l	2	3				
e	3					
r	4					
a	5					

$$
\begin{array}{ll}
D(i,0) = i & 0 \leq i \leq \mathit{length}(s) \\
D(0,k) = k & 0 \leq k \leq \mathit{length}(t) \\
D(i,k) = & \min \left\{ \begin{array}{l} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \left\{ \begin{array}{ll} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{array} \right. \end{array} \right.
\end{array}
$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2	3	4	5	6
l	2	3				
e	3	2 4 4				
r	4					
a	5					

$$D(i,0) = i \qquad 0 \leq i \leq length(s)$$

$$D(0,k) = k \qquad 0 \leq k \leq length(t)$$

$$D(i,k) = \min \begin{cases} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \begin{cases} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{cases} \end{cases}$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2	3	4	5	6
l	2	3				
e	3	2				
r	4					
a	5					

$$
\begin{array}{ll}
D(i,0) = i & 0 \le i \le \mathit{length}(s) \\
D(0,k) = k & 0 \le k \le \mathit{length}(t) \\
D(i,k) = & \min \left\{ \begin{array}{l} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \left\{ \begin{array}{ll} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{array} \right. \end{array} \right.
\end{array}
$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2	3	4	5	6
l	2	3				
e	3	2				
r	4	5 3 5				
a	5					

$$D(i,0) = i \qquad 0 \leq i \leq \mathit{length}(s)$$
$$D(0,k) = k \qquad 0 \leq k \leq \mathit{length}(t)$$
$$D(i,k) = \min \begin{cases} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \begin{cases} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{cases} \end{cases}$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2	3	4	5	6
l	2	3				
e	3	2				
r	4	3				
a	5					

$$D(i,0) = i \qquad 0 \le i \le length(s)$$
$$D(0,k) = k \qquad 0 \le k \le length(t)$$
$$D(i,k) = \min \begin{cases} D(i-1,k)+1 \\ D(i,k-1)+1 \\ D(i-1,k-1) + \begin{cases} 2 & \text{if } s[i] \neq t[k] \\ 0 & \text{if } s[i] = t[k] \end{cases} \end{cases}$$

Computing Levenshtein distance

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2	3	4	5	6
l	2	3	2	3	4	5
e	3	2	3	4	3	4
r	4	3	4	5	4	3
a	5	4	5	6	5	4

Computing Levenshtein distance

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2	3	4	5	6
l	2	3	2	3	4	5
e	3	2	3	4	3	4
r	4	3	4	5	4	3
a	5	4	5	6	5	4

Alignment (Länkning)

Pair up the symbols in the two strings, e.g.

```
eller
 |  ||
fl era
```

Compute *backpointers* based on the table. From cell (i, k), compare values of cells $(i-1, k-1)$, $(i-1, k)$ and $(i, k-1)$.

Start in the lower right corner.
Repeat until $i = 0$ and $k = 0$:
If $(i-1, k-1)$ is smallest, align $s[i]$ and $t[k]$. Go to $(i-1, k-1)$.
If $(i-1, k)$ is smallest, align $s[i]$ with ' '. Go to $(i-1, k)$.
If $(i, k-1)$ is smallest, align $t[k]$ with ' '. Go to $(i, k-1)$.

Computing backpointers and alignment

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2	3	4	5	6
l	2	3	2	3	4	5
e	3	2	3	4	3	4
r	4	3	4	5	4	3
a	5	4	5	6	5	4

Computing backpointers and alignment

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2	3	4	5	6
l	2	3	2	3	4	5
e	3	2	3	4	3	4
r	4	3	4	5	4	3
a	5	4	5	6	5	4

a

Computing backpointers and alignment

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2	3	4	5	6
l	2	3	2	3	4	5
e	3	2	3	4	3	4
r	4	3	4	5	4	3
a	5	4	5	6	5	4

```
r   a
|
r
```

Computing backpointers and alignment

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2	3	4	5	6
l	2	3	2	3	4	5
e	3	2	3	4	3	4
r	4	3	4	5	4	3
a	5	4	5	6	5	4

```
e  r  a
|  |
e  r
```

Computing backpointers and alignment

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2	3	4	5	6
l	2	3	2	3	4	5
e	3	2	3	4	3	4
r	4	3	4	5	4	3
a	5	4	5	6	5	4

```
  e r a
  | |
l e r
```

Computing backpointers and alignment

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2	3	4	5	6
l	2	3	2	3	4	5
e	3	2	3	4	3	4
r	4	3	4	5	4	3
a	5	4	5	6	5	4

l		e	r	a
|		|	|	
l	l	e	r	

Computing backpointers and alignment

	#	e	l	l	e	r
#	0	1	2	3	4	5
f	1	2	3	4	5	6
l	2	3	2	3	4	5
e	3	2	3	4	3	4
r	4	3	4	5	4	3
a	5	4	5	6	5	4

f	l		e	r	a
	|		|	|	
e	l	l	e	r	