LING/C SC/PSYC 438/538

Lecture 19

Sandiway Fong

Administrivia

- **Next Monday**
  - guest lecture from Dr. Jerry Ball of the Air Force Research Labs to be continued
  - formalism described on Monday
  - implementation of the Double R grammar
  - complete with demo

Today's Topics

- Chapter 3 of JM
  - 3.10 Spelling and Correction of Spelling Errors
  - *Also discussed in section 5.9*
  - 3.11 Minimum Edit Distance

- Reading
  - Chapter 4: N-grams
  - Pre-requisite:
    - An understand of simple probability concepts

But first

- From last time…
- Revisit Finite State Transducers (FST)

e-insertion FST

- Context-sensitive rule:

$$\epsilon \rightarrow e \,/\, \left\{ \begin{array}{c} x \\ s \\ z \end{array} \right\} \hat{}\ \underline{\hspace{2em}}\ s\# \qquad (3.4)$$

right context

left context

^ morpheme boundary
# word boundary

- Corresponding FST:

other = any feasible pair not mentioned explicitly in this transducer

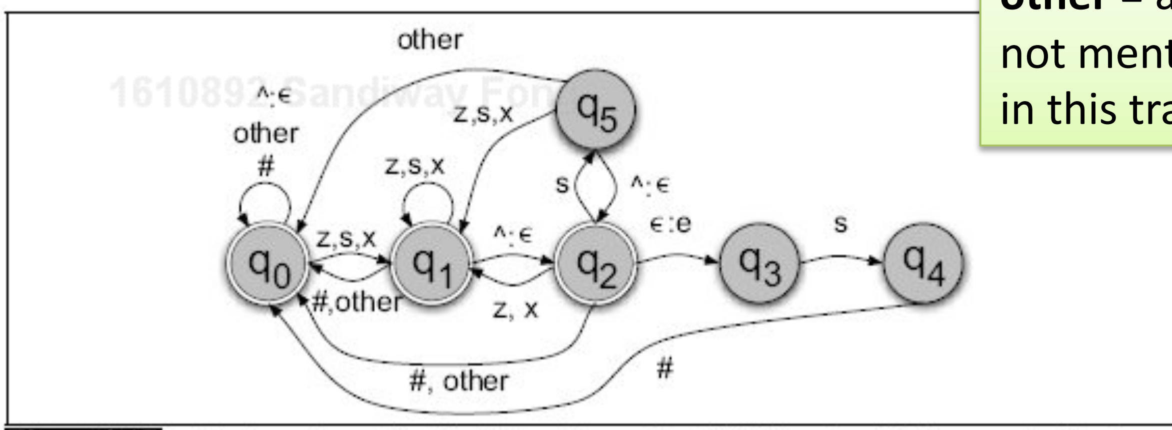


Figure 3.17 The transducer for the E-insertion rule of (3.4), extended from a similar transducer in Antworth (1990). We additionally need to delete the # symbol from the surface string; we can do this either by interpreting the symbol # as the pair #:ϵ or by postprocessing the output to remove word boundaries.

e-insertion FST

- FST:

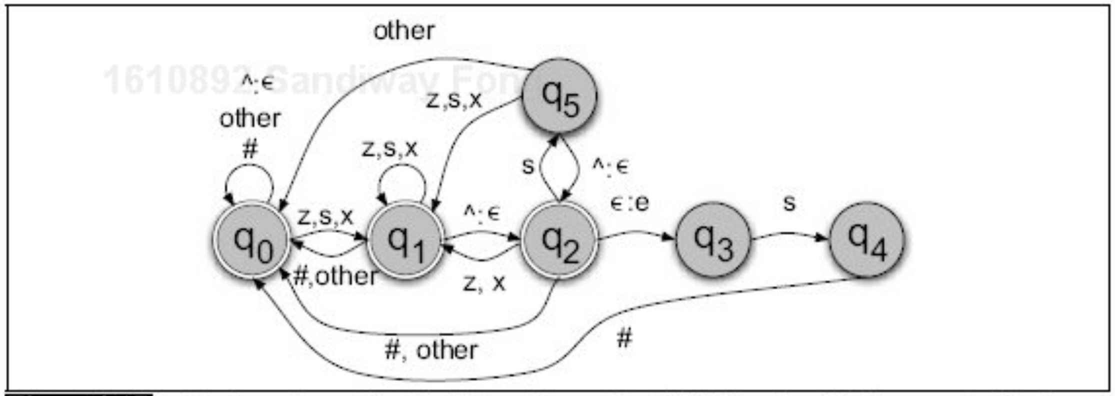


Figure 3.17 The transducer for the E-insertion rule of (3.4), extended from a similar trans-

Input

- State transition table:

State \ Input	s:s	x:x	z:z	^:ϵ	ϵ:e	#	other
q_0:	1	1	1	0	-	0	0
q_1:	1	1	1	2	-	0	0
q_2:	5	1	1	0	3	0	0
q_3	4	-	-	-	-	-	-
q_4	-	-	-	-	-	0	-
q_5	1	1	1	2	-	-	0

Figure 3.18 The state-transition table for the E-insertion rule of Fig. 3.17, extended from a similar transducer in Antworth (1990).

Output

f	o	x	e	s	#

e-insertion FST

- FST:

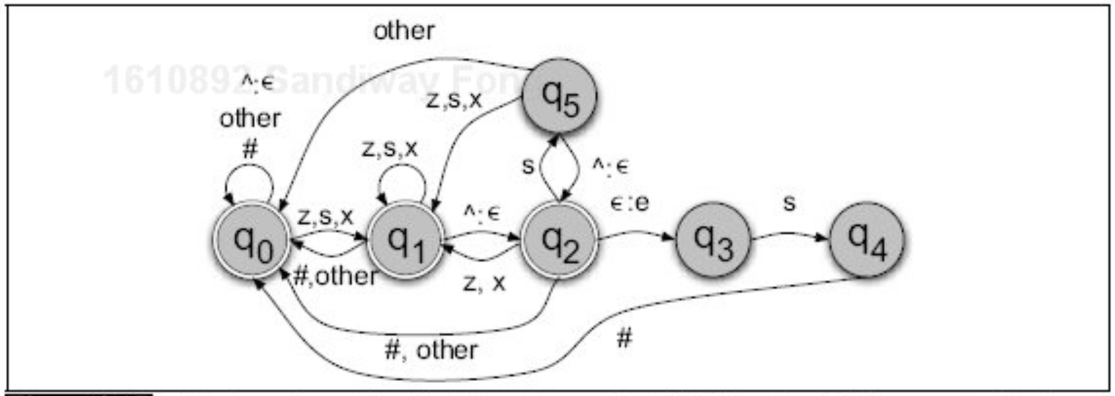


Figure 3.17 The transducer for the E-insertion rule of (3.4), extended from a similar trans-

Input

z	i	p	^	s	#

- State transition table:

State \ Input	s:s	x:x	z:z	^:ϵ	ϵ:e	#	other
q_0:	1	1	1	0	-	0	0
q_1:	1	1	1	2	-	0	0
q_2:	5	1	1	0	3	0	0
q_3	4	-	-	-	-	-	-
q_4	-	-	-	-	-	0	-
q_5	1	1	1	2	-	-	0

Figure 3.18 The state-transition table for the E-insertion rule of Fig. 3.17, extended from a similar transducer in Antworth (1990).

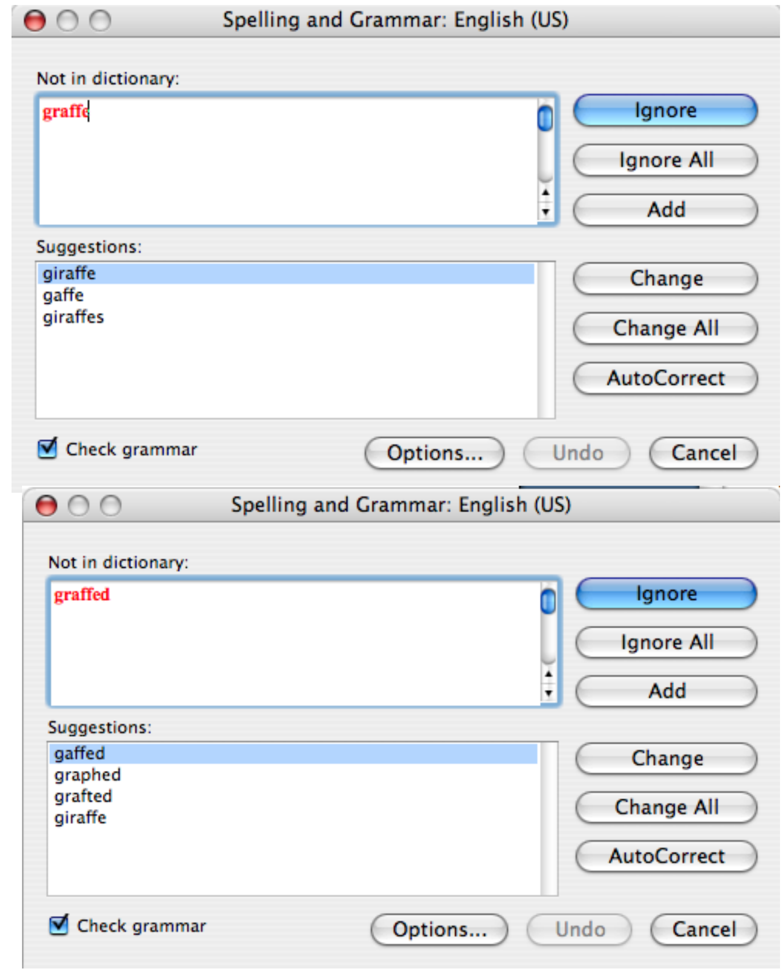
Output

z	i	p	s	#	

Spelling Errors

Spelling Errors

- Textbook cites (Kukich, 1992):
  - Non-word detection (easiest)
    - graffe (giraffe)
  - Isolated-word (context-free) error correction
    - graffe (giraffe,…)
    - graffed (gaffed,…)
    - by definition cannot correct when error word is a valid word
  - Context-dependent error detection and correction (hardest)
    - your an idiot ⇒ you're an idiot
    - (Microsoft Word corrects this by default)



Spelling Errors

- OCR
  - visual similarity
    - h⇔b, e⇔c , jump⇔jurnps
- Typing
  - keyboard distance
    - small⇔smsll, spell⇔spel;
- Graffiti (many HCI studies)
  - stroke similarity
    - Common error characters are: V, T, 4, L, E, Q, K, N, Y, 9, P, G, X
    - Two stroke characters: B, D, P (error: two characters)
- Cognitive Errors
  - bad spellers
    - separate⇔seperate

correct

- textbook section 5.9

- Kernighan et al. (`correct`)
  - take typo *t* (not a word)
    - mutate *t* minimally by deleting, inserting, substituting or transposing (swapping) a letter
    - look up "mutated *t*" in a dictionary
    - candidates are "mutated *t*" that are real words
  - **example** (5.2)
    - t = acress
    - C = {actress,cress,
      caress, access,
      across, acres,
      acres}

		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	–	2	5	insertion
acress	acres	–	2	4	insertion

correct

- formula
  - $\hat{c}$ = argmax $_{c \in C}$ P(t|c) P(c)   (Bayesian Inference)
  - C = {actress, cress, caress, access, across, acres, acres}
- Prior: P(c)
  - estimated using frequency information over a large corpus (N words)
  - P(c) = freq(c)/N
  - P(c) = freq(c)+0.5/(N+0.5V)
    - avoid zero counts (non-occurrences)
    - (add fractional part 0.5)
    - add one (0.5) smoothing
    - V is vocabulary size of corpus

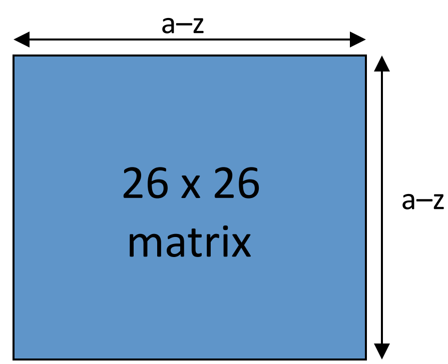
c	freq(c)	p(c)
actress	1343	.0000315
cress	0	.000000014
caress	4	.0000001
access	2280	.000058
across	8436	.00019
acres	2879	.000065

correct

- ***Likelihood***: P(t|c) *probability of typo t given candidate word c*

Very hard to collect this data

  - *using some corpus of errors*
  - compute following 4 ***confusion matrices***
  - del[x,y] = freq(correct xy mistyped as x)
  - ins[x,y] = freq(correct x mistyped as xy)
  - sub[x,y] = freq(correct x mistyped as y)
  - trans[x,y] = freq(correct xy mistyped as yx)
  - P(t|c) = del[x,y]/f(xy) if c related to t by deletion of y
  - P(t|c) = ins[x,y]/f(x) if c related to t by insertion of y *etc…*



c	freq(c)	p(c)	p(t|c)	p(t|c)p(c)	%
actress	1343	.0000315	.000117	3.69×10^{-9}	**37%**
cress	0	.000000014	.00000144	2.02×10^{-14}	**0%**
caress	4	.0000001	.00000164	1.64×10^{-13}	**0%**
access	2280	.000058	.000000209	1.21×10^{-11}	**0%**
across	8436	.00019	.0000093	1.77×10^{-9}	**18%**
acres	2879	.000065	.0000321	2.09×10^{-9}	**21%**
acres	2879	.000065	.0000342	2.22×10^{-9}	**23%**

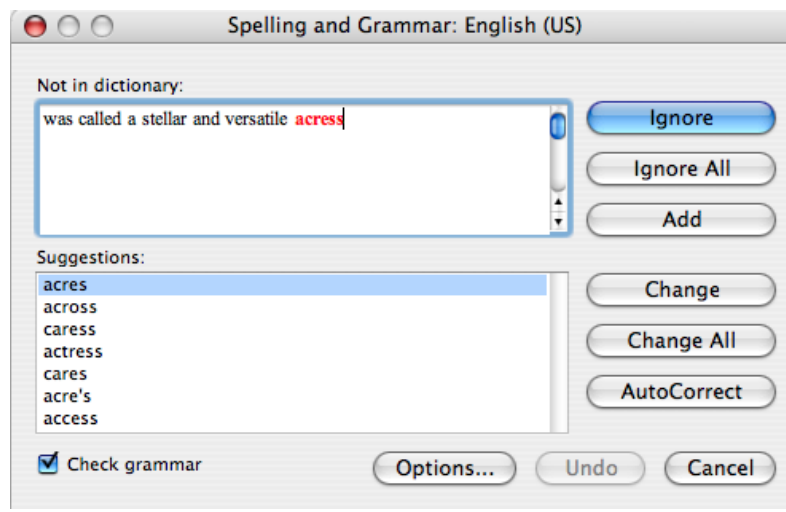
correct

- example
  - t = acress
  - $\hat{c}$ = acres (44%)

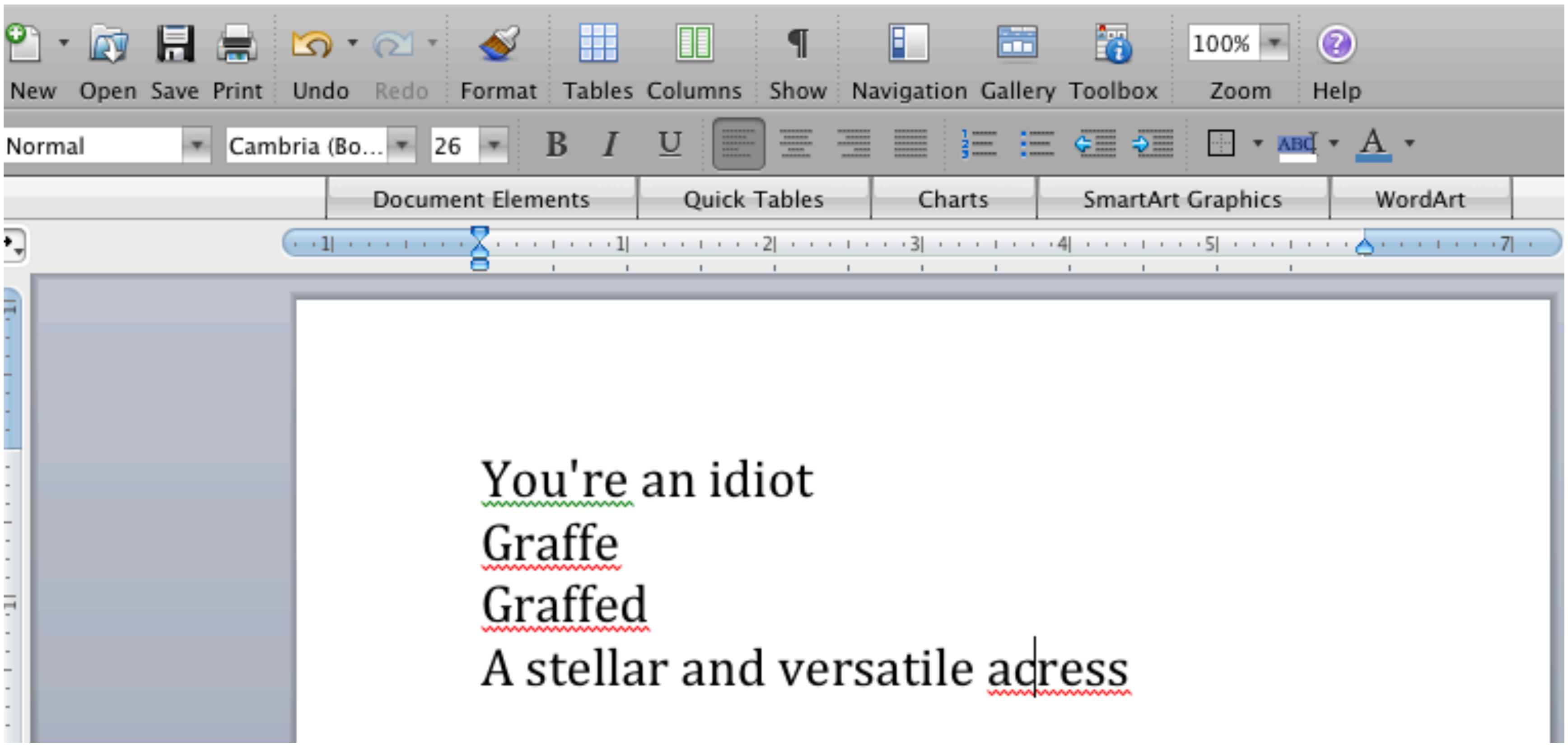
c	freq(c)	p(c)	p(t|c)	p(t|c)p(c)	%
actress	1343	.0000315	.000117	3.69×10^{-9}	**37%**
cress	0	.000000014	.00000144	2.02×10^{-14}	**0%**
caress	4	.0000001	.00000164	1.64×10^{-13}	**0%**
access	2280	.000058	.000000209	1.21×10^{-11}	**0%**
across	8436	.00019	.0000093	1.77×10^{-9}	**18%**
acres	2879	.000065	.0000321	2.09×10^{-9}	**21%**
acres	2879	.000065	.0000342	2.22×10^{-9}	**23%**

- despite all the math
- wrong result for
  - was called a stellar and versatile acress

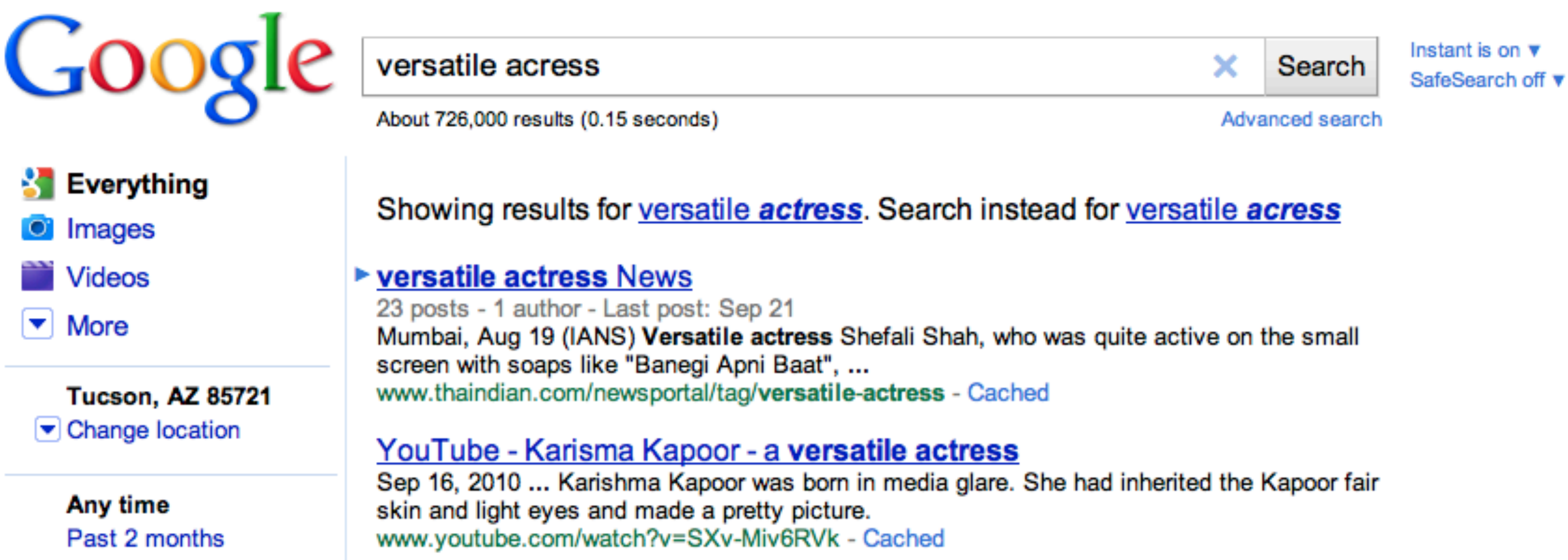
- **what does Microsoft Word use?**
  - *was called a stellar and versatile acress*



Microsoft Word



Google



versatile acress
Search
Instant is on ▼
SafeSearch off ▼
About 726,000 results (0.15 seconds)
Advanced search
Everything
Images
Videos
More
Tucson, AZ 85721
Change location
Any time
Past 2 months
Showing results for versatile actress. Search instead for versatile acress
versatile actress News
23 posts - 1 author - Last post: Sep 21
Mumbai, Aug 19 (IANS) Versatile actress Shefali Shah, who was quite active on the small screen with soaps like "Banegi Apni Baat", ...
www.thaindian.com/newsportal/tag/versatile-actress - Cached
YouTube - Karisma Kapoor - a versatile actress
Sep 16, 2010 ... Karishma Kapoor was born in media glare. She had inherited the Kapoor fair skin and light eyes and made a pretty picture.
www.youtube.com/watch?v=SXv-Miv6RVk - Cached

Minimum Edit Distance

- textbook section 3.11

- general string comparison
- edit operations are insertion, deletion and substitution
- not just limited to distance defined by a single operation away (`correct`)
- we can ask how different is string *a* from *b* by the ***minimum edit distance***

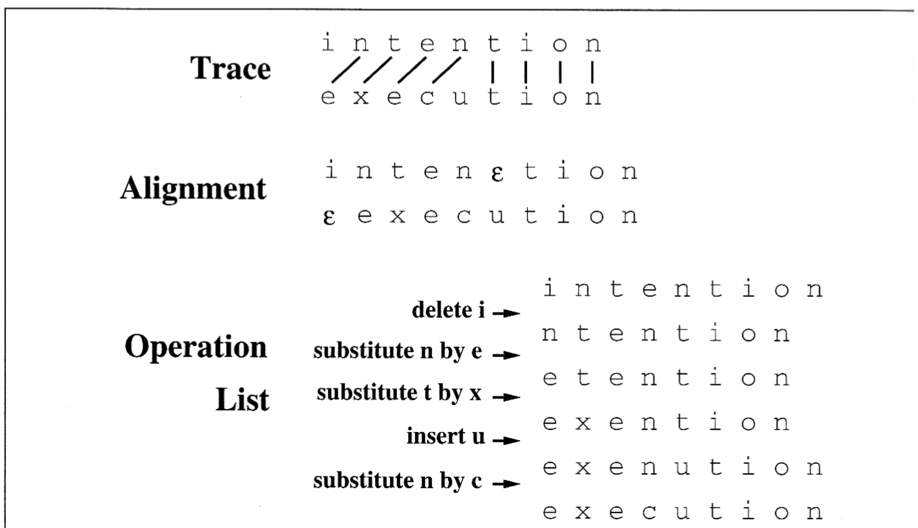


Figure 5.4 Three methods for representing differences between sequences (after Kruskal (1983))

Minimum Edit Distance

- **applications**
  - could be used for multi-typo correction
  - used in Machine Translation Evaluation (MTEval)
  - **example**
    - **Source**: 生産工程改善について
    - **Translations**:
    - (Standard) For improvement of the production process
    - (MT-A) About a production process betterment
    - (MT-B) About the production process improvement
    - **method**
      - compute edit distance between MT-A and Standard and MT-B and Standard in terms of word insertion/substitution etc.

Minimum Edit Distance

- **cost models**
  - **Levenshtein**
    - insertion, deletion and substitution all have unit cost
  - **Levenshtein** (alternate)
    - insertion, deletion have unit cost
    - substitution is twice as expensive
    - *substitution = one insert followed by one delete*
  - **Typewriter**
    - insertion, deletion and substitution all have unit cost
    - modified by key proximity

Minimum Edit Distance

- ***Dynamic Programming***
  - divide-and-conquer
    - to solve a problem we divide it into sub-problems
  - sub-problems may be repeated
    - don’t want to re-solve a sub-problem the 2nd time around
  - idea: put solutions to sub-problems in a table
    - and just look up the solution 2nd time around, thereby saving time
    - ***memoization***

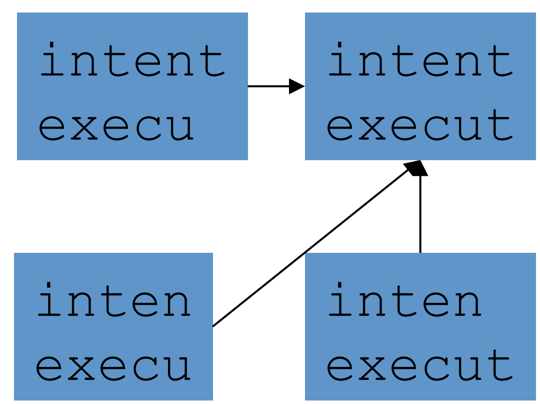
Fibonacci Series

- definition
  - F(0)=0
  - F(1)=1
  - F(n) = F(n-1) + F(n-2) for n≥2
- sequence
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …
- **computation**
  - F(5)
  - F(4)+F(3)
  - F(3)+F(2)+F(3)
  - F(2)+F(1)+F(2)+F(2)+F(1)

Minimum Edit Distance

- can be defined incrementally
- **example**:
  - `intention`
  - `execution`
- **suppose we have the following minimum costs**
  - insertion, deletion have unit cost, substitution is twice as expensive
  - `intent` ⇒ `execu` (9)
  - `inten` ⇒ `execut` (9)
  - `inten` ⇒ `execu` (8)
- **Minimum cost for intent ⇒ execut (8) is given by computing the possibilities**
  - `intent` ⇒ `execu` (9) ⇒ `execut` *(insert t)* (10)
  - `intent` *(delete t)* ⇒ `inten` ⇒ `execut` (9) (10)
  - `inten` ⇒ `execu` (8): substitute t for t (*zero cost*) (8)

one edit operation away



Minimum Edit Distance

- Generally

$$distance[i,j] = \min \begin{cases} distance[i-1,j] + \text{ins-cost}(target_{i-1}) \\ distance[i-1,j-1] + \text{sub-cost}(source_{j-1}, target_{i-1}) \\ distance[i,j-1] + \text{del-cost}(source_{j-1})) \end{cases}$$

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**

Figure 3.27 When entering a value in each cell, we mark which of the three neighboring cells we came from with up to three arrows. After the table is full we compute an **alignment** (minimum edit path) by using a **backtrace**, starting at the **8** in the upper-right corner and following the arrows. The sequence of dark grey cells represents one possible minimum cost alignment between the two strings.

Minimum Edit Distance

- Programming Practice: *could be easily implemented in Perl*

```
function MIN-EDIT-DISTANCE(target, source) returns min-distance

  n ← LENGTH(target)
  m ← LENGTH(source)
  Create a distance matrix distance[n+1,m+1]
  Initialize the zeroth row and column to be the distance from the empty string
     distance[0,0] = 0
     for each column i from 1 to n do
         distance[i,0] ← distance[i-1,0] + ins-cost(target[i])
     for each row j from 1 to m do
         distance[0,j] ← distance[0,j-1] + del-cost(source[j])
  for each column i from 1 to n do
     for each row j from 1 to m do
        distance[i,j] ← MIN( distance[i−1,j] + ins-cost(target_{i−1}),
                             distance[i−1,j−1] + sub-cost(source_{j−1}, ]target_{i−1}),
                             distance[i,j−1] + del-cost(source_{j−1}))
  return distance[n,m]
```


Figure 3.25 The minimum edit distance algorithm, an example of the class of dynamic programming algorithms. The various costs can either be fixed (e.g., $\forall x$, ins-cost$(x) = 1$) or can be specific to the letter (to model the fact that some letters are more likely to be inserted than others). We assume that there is no cost for substituting a letter for itself (i.e., sub-cost$(x,x) = 0$).

Minimum Edit Distance

- Generally

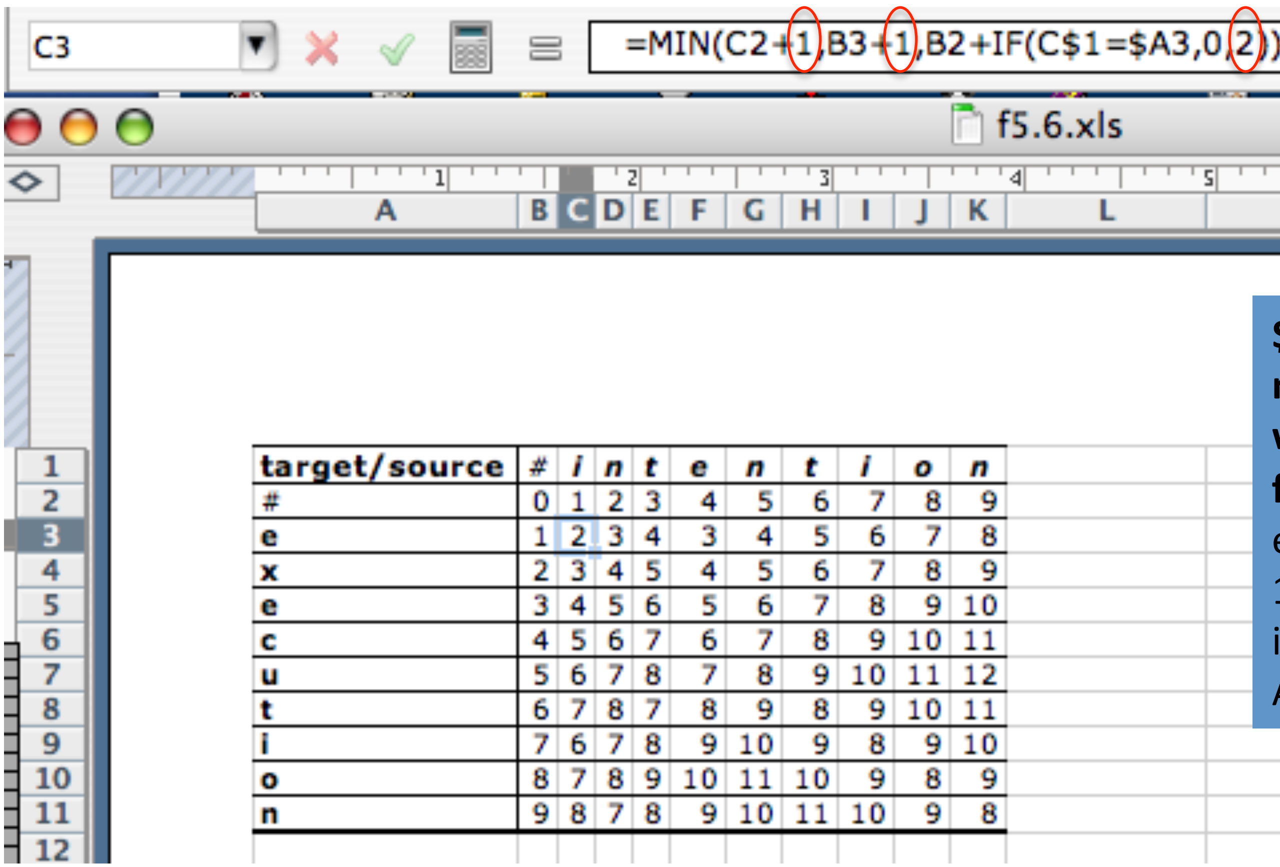
$$distance[i,j] = \min \begin{cases} distance[i-1,j] + \text{ins-cost}(target_{i-1}) \\ distance[i-1,j-1] + \text{sub-cost}(source_{j-1}, target_{i-1}) \\ distance[i,j-1] + \text{del-cost}(source_{j-1})) \end{cases}$$

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**

Figure 3.26 Computation of minimum edit distance between *intention* and *execution* with the algorithm of Fig. 3.25, using Levenshtein distance with cost of 1 for insertions or deletions, 2 for substitutions. In italics are the initial values representing the distance from the empty string.

Minimum Edit Distance Computation

- Or in Microsoft Excel, file: eds.xls (on website)



target/source	#	i	n	t	e	n	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
e	1	2	3	4	3	4	5	6	7	8
x	2	3	4	5	4	5	6	7	8	9
e	3	4	5	6	5	6	7	8	9	10
c	4	5	6	7	6	7	8	9	10	11
u	5	6	7	8	7	8	9	10	11	12
t	6	7	8	7	8	9	8	9	10	11
i	7	6	7	8	9	10	9	8	9	10
o	8	7	8	9	10	11	10	9	8	9
n	9	8	7	8	9	10	11	10	9	8

$ in a cell reference means don't change when copied from cell to cell
e.g. in C$1
1 stays the same
in $A3
A stays the same