30. 英文綴り検査法†

川　合　慧††

1. はじめに

文書作成に際しては，全体構成を考えたり各節の表現を推敲したりする本質的な作業のほかに，ページごとのレイアウトや1行の中の単語配置，それに各単語の綴りチェックなどの，表示に必要な諸作業を行わなければならない．この表示のための作業は，文章の内容に立入る性質のものではないが，公表される文書としての読易さ・正確さに関係する．近来，いわゆるワード・プロセッサの普及とともに，これら諸作業の自動化についての研究が進み，多くの実用的なシステムが開発されてきている．本稿ではこれらのうちで，比較的研究が進んでいる英単語の綴りの検査と修正に話題を絞り，その概略と評価について述べる．

計算機を使って英単語の綴りを検査するという考えは，計算機が'ふつうの文字'を扱うようになってから当然のようにして現れた[18]．しかし初期においては，文書ファイルのサイズや処理速度の低さから，ごく小規模の文章しか対象となっていなかった．その中にあって綴り検査が高い有用性を発揮したのは，各種システムにおけるデータ入力の分野であった[2),7),8)]．この分野では，人名・地名やプログラム言語の原始テキスト中の名前など，ごく限られた単語集合のみを扱えばよい．しかも，OCR 入力における誤り（1文字単位の誤読あるいは読取り不能）に見られるように，誤りの種類を限ることができる場合が多いので，自動修正が成功する率も高い．このような理由で，自動修正による高信頼度の入力システムを構成する研究が，初期の活動として盛んに行われた．しかしこれらは，一般的な英文を処理対象としたものではなかったので，それぞれの応用システムの一部とみなされるにとどまっていた．

綴りの検査と修正がもっと一般的に取り扱われるようになったのは，エディタ等による文書処理が盛んになってきた1970年代からである．その処理対象も，単なる文字列の集合から，実際の使用環境における統計的な性質を基礎とした現実的な英語へと進展していった．それにつれて，検査・修正用の辞書の構成法，誤りの種類の分析，文字列間の各種の距離などの，多彩な研究活動が行われるようになった．

2. 綴り誤り

2.1 綴り誤りとその分類

英語の文章は，英字などの目に見える文字（図形文字と呼ばれることもある）と，空白・改行・改ページなどの書式制御文字との連なりであるが，連続した図形文字からなる単語と，それらを区切る区切り文字とから成るという構造を持っている．これを**単語構造** (word structure) とよぶ[18]．綴り誤りを，文章中の任意の文字に対する何らかの変化と定義したとすると，その中には単語構造が変化したものも含まれることになる．たとえば，

red ape ⟶ redtape　　(空白→t)
oneself ⟶ one elf　　(s→空白)

といった誤りも発生し得る．この種の誤りを**単語構造誤り** (word structure error) と呼ぶ．

単語構造の変化はないものと仮定した場合の綴り誤りは，もとの正しい綴りが別の正しい（すなわち辞書に載っている）綴りに変化する**単語誤り** (word error) と，辞書にはない綴りに変化する**文字誤り** (charactor error) とに分類される[6]．たとえば，綴り store 中の r が他の文字に変化した場合，

k，l，n，p，v　であれば単語誤り，
それ以外（r を除く）であれば文字誤り

となる*．

以上3種の誤りのうち，検査・修正の対象として扱われるのは，ほとんどの場合文字誤りのみである．辞

† Spell checking and correction of English texts by Satoru KAWAI (Department of Information Science, University of Tokyo).

†† 東京大学理学部

* Webster's Third New International Dictionary による．

書に載っている綴りであっても'誤り'と判定するためには，英文の構文・意味解釈にまで立入った処理が必要となるからである．単語構造誤りの場合，結果の綴りが辞書になければ文字誤りとして検出はできる．しかし，誤りの範囲の確定や完全な修正をするためには，やはり広域的な処理を行わねばならない．本稿でも，文字誤りの検出と修正に話を限ることにする．

2.2 綴り誤りの種類と検査・修正

文字誤りの範囲内では，つぎのような'誤り操作'が代表的なものとして知られている[4]．

（a） 1文字置換 letter ⟶ litter
（b） 1文字削除 ⟶ leter
（c） 1文字挿入 ⟶ lettear
（d） 隣接文字の互換 ⟶ lettre

現実の綴り誤りの約8割が，これらのどれかひとつを1回だけ施したものとして得られるという報告[4]がある．また，すべての綴り誤りに対して，これらの操作の逆操作と辞書とを併用した修正アルゴリズムによって，もとの正しい綴りが約95%の割合で復元できるともいう[4]．それで，多くの検査・修正アルゴリズムがこの4種の誤り操作を基本としている．また，これらを使った理論的研究[10),13)]も多くなされている．

現在知られている検査・修正法は

（i） 辞書の見出し語の表*の探索によるもの
（ii） 多字組の頻度統計を使用するもの

の2種に分類される．以下の章では，これらの方法の概略とその実現法について述べる．

3. 表探索による検査と修正

3.1 表探索と探索キー

綴り検査のための辞書においては，おもに以下の3種の探索キーが用いられている．

（1） **全綴キー** (full spelling key)

探索キーとして，入力綴りそのものを用いる．最も単純で確実な方法ではあるが，活用形や派生語をすべて登録してしまうことから，辞書のサイズと探索効率が問題となることが多い．修正を意識する場合には，よくまちがわれる綴りや，誤り操作によって得られる誤った綴りも，正しい綴りと一緒に登録しておく方法も試されている[5),12)]．

（2） **語幹キー** (word root key)

入力綴りから**接辞**(affix)，すなわち**接頭辞**(prefix)や**接尾辞** (suffix) を取り除いたものを探索キーとする．これは，活用形や派生語を語幹で代表させることによって，辞書を小さくすることを目的とした方法である[5),14),15)]．たとえば，接頭辞 dis と接尾辞 able, ment とを処理することによって，

agree, agreeable, agreement, disagree,
disagreeable, disagreement

の6語を agree で探索することができる．辞書が小さくなれば探索の効率も高まるし，辞書全体を主記憶に置いておける可能性も高くなる．

語幹キー方式の問題点は，接辞の連接規則に多くの例外が含まれていることである．各接辞が付く単語の品詞は限られているし，連接に際して綴りが不規則に変化することも多い．たとえば

begin + er ⟶ beginner
queut + ing ⟶ queuing
wife + s ⟶ wives

といったぐあいである．

そこでこの方式では，各キーに接辞規則の情報を付属させておくことが必要である*．この情報としては，いくつかに類型化された規則のうち，どれがそのキーに適用されるかを示すものが使われることが多い．規則の類型の例をふたつだけ示す[14)]．α と β は英字1文字を表わす．

$$\cdots\alpha+\mathrm{ve}\longrightarrow\begin{cases}\cdots\alpha\mathrm{ive} & (\alpha\neq\mathrm{e})\\ \cdots\mathrm{ive} & (\alpha=\mathrm{e})\end{cases}$$

$$\cdots\alpha\beta+\mathrm{s}\longrightarrow\begin{cases}\cdots\alpha\beta\mathrm{es} & (\beta=\mathrm{s,\ x,\ z,\ h})\\ \cdots\alpha\mathrm{ies} & (\beta=\mathrm{y\ and}\\ & \alpha\neq\mathrm{a,\ e,\ i,\ o,\ u})\\ \cdots\alpha\beta\mathrm{s} & (\mathrm{otherwise})\end{cases}$$

入力綴り implyes の探索過程を示す．

implye+s として implye で探索（失敗）
imply+es として imply で探索（成功）
imply に s または es を連接 (implies)

結局，もとの入力綴りが復元できないので，探索は失敗に終る．この例で示されるように，語幹キーの方式では探索手順が複雑になるので，辞書を小さくする必要度が高い場合に使われることが多い．

（3） **短縮キー** (abbreviation key)

綴りから一定の規則で文字を必要な数だけ抜いてゆき，残った短い文字列をキーとする．ここで，登録するべき単語集合の要素について，対応する短縮キーが

* 本稿では簡単のために，これを辞書と呼ぶことにする．より正確には，単語表 (word list) あるいは見出し表 (key list) と呼ぶべきであろう．

* 実際の英語の辞書におけるやり方に近い．

表-1 文字不必要（文献1)による）

a	b	c	d	e	f	g	h	i	j	k	l	m
5	1	5	0	7	1	2	5	6	0	1	5	1

n	o	p	q	r	s	t	u	v	w	x	y	z
3	4	3	0	4	5	3	4	1	1	0	2	1

表-2 位置不必要度（文献1)による）

位置	1	2	3	4	5	6	7	8	9以上
(先頭から)	0	2	4	5	5	6	6	6	7
(末尾から)	1	3	4	5	5	6	6	7	7

すべて異なるような最も短い長さが，キーの長さとして選ばれる．この方法では，キーの数は少なくならないが，キー自体の短縮による辞書サイズの減小によって，探索効率を高めることができる．綴り検査のための規則では，綴りの中の各文字の'不必要度'を計算し，その値の大きい文字から抜いてゆく．たとえば，英文中における出現頻度の高い文字は，持っている情報量が小さいので，不必要度が大きいと考えることもできる．よく知られている頻度順(e, t, a, o,…)を使うと，たとえば

positioner ⟶ psiinr

corrector ⟶ corrcr

となる．**表-1,2**に，これを基本とし，よくまちがわれる文字や綴りの中央付近の文字の不必要度を高めた例を示す[1]．これは経験的に決められたものである．実際の短縮処理の例を示す．

p	o	s	i	t	i	o	n	e	r	もとの綴り
3	4	5	6	3	6	4	3	7	4	文字不必要度
0	2	4	5	5	5	5	4	3	1	位置不必要度
3	6	9	11	8	11	9	7	10	5	総合不必要度
			∨		∨	∨		∨		抜く文字

結果は postnrとなる．同様にして corrector はcortorに縮められる．

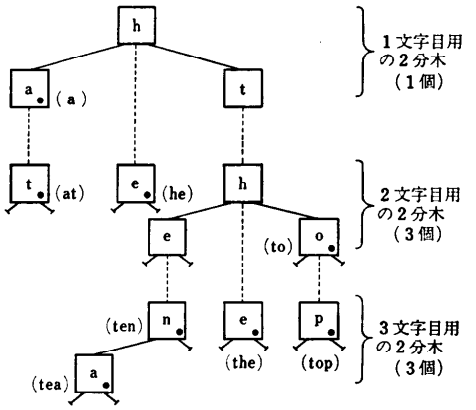
短縮キーの方法では，入力綴りから抜かれる文字がまちがっていたり，もともとその文字が脱落したりしていても，同じキーとなってしまう．たとえば

posetioner, postioner

などは，どちらも postnr に縮められ，誤りであることは発見できない．この事情を改善するには，もとの綴りを各キーに付加しておく形要がある（後述）．

3.2 探索表の構成

前節で示した種々のキーを，探索に適した形式に構成したものが，探索表である．ここでは，綴り検査に関係の深い構成法をいくつか示す．



●はキー（カッコ内に示した）の存在を示すマーク

図-1 文字探索木

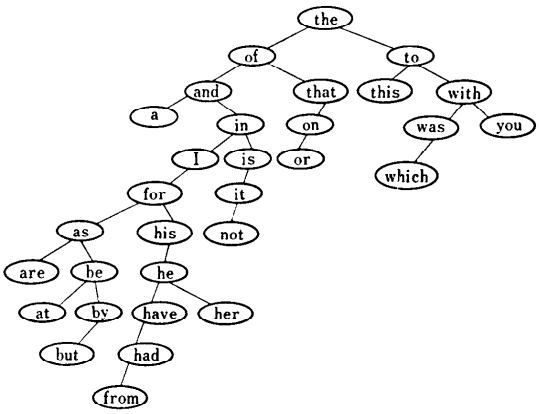


図-2 頻度順探索木（文献9)による）

(1) 文字探索木

綴りの先頭から1文字ずつ2分探索を行うための木である[15),19)]．各ノードには，そこまでの綴りのキーがあるかないかを示す印が含まれる（**図-1**）．この木は大きな記憶量を必要とするが，接辞の解析やある種の修正処理には適している．ほかに，各文字の探索を1回で済ませるための26（あるいはそれ以上）分木(trie) が使われることもある[9),15)]．

(2) 頻度順の探索木

空(empty)の2分木に，あらかじめわかっている探索頻度順にキーを挿入すると，高頻度のキーほど木の根に近く配置され，平均的な探索効率を上げることができる．高頻度の英単語31個にこの方法を適用したものを**図-2**に示す[9)]．この木における平均成功比較回数は4.04である．頻度分布を基礎とした最適な探索木は動的計画法の手法によって作られる．図-2と同じ

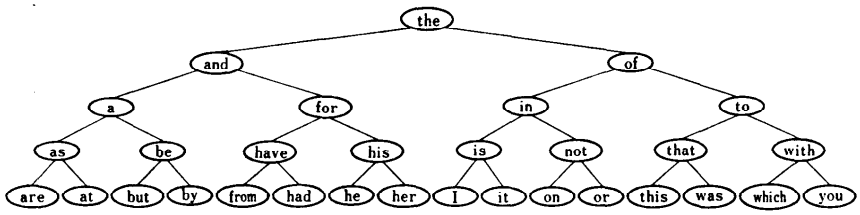


図-3 中央値分割探索木（文献17)による）

キー集合に対しては，平均成功比較回数が3.44である最適木が得られる[9]．

ふつうの2分探索木では，ノードのキー自体が，照合不成功の場合にどちらの部分木を選ぶかを決める分割値となっている．これに対して，キーとは別の分割値，特に左右部分木中のキーの中央値をノードに書いておくことによって，木全体のバランスを良くし，平均探索効率を高める提案がなされている[17]．**図-3**はこの例であり，平均成功探索回数は3.13である．

（3） 頻度別の探索表

図-2,3でデータとした31単語の，全英文中での使用頻度は約35%である．また，134語で50%，2000語で95%という報告がある[15]．この統計的事実から，辞書中のキーを頻度別にいくつかに分類し，それぞれ別々の探索表とすることが考えられる．特に，頻度が極めて小さい多数のキーを収めた大きな表を2次記憶に置くという構成が有効である．

（4） 全ハッシュ法

ふつうのハッシュ法では，最終的な照合のためにキー自体も保持する．これに対して，キーやポインタが占有するすべての記憶領域を，ビット構成の大きなハッシュ表として使用する方法が，全ハッシュ法である[19]．表の各エントリは，そのハッシュ値を持つキーの登録・非登録を示す1ビットのみである．この方法ではハッシュ値の範囲が何10倍にも広げられるので，異なるキーが同じハッシュ値を持つ衝突現象が起きる確率は非常に小さくなる．しかし衝突が起きた場合には，登録されていないキーが'見つかってしまう'ことになる．そこで，複数個の独立なハッシュ関数を同時に使用して，この確率をさらに小さくする試みもなされており，2000分の1という低衝突確率も実現されている．

3.3 表探索による修正

修正法としては，以下に示す3種が知られている．

（1） 逆操作による方法

入力綴りに誤り操作の逆操作を施して得られる綴り（複数個）を，辞書の中で探索する．ひとつだけ残ればそれを結果とする．複数の場合は，文脈情報の利用や対話処理によって，ひとつに絞る努力をする．それでもだめな場合は決定不能とする．**2.2**で述べた操作の逆操作によって得られる綴りの数は，入力綴りの長さをnとした場合，およそ次のようになる*．

置換	$25n$	削除	$26(n+1)$
挿入	n	互換	$n-1$

これらの操作は，綴りの長さを高々1しか変えないので，辞書をキーの長さごとに分割しておけば，この場合の探索の効率を高くすることができる[14),15]．また，綴りのうしろの方の変更は前の方に影響しないので，文字探索木[19]も有効である．

（2） 距離による方法

探索キーとの'距離'が最小となるキーについて，(1)と同じ処理をする．文字列間の距離としては，置換の必要最小回数による Hamming 距離（同長文字列），置換・削除・挿入の回数による Levenshtein 距離[13]，互換までを含めた編集列距離[10]，最長共有部分列(LCS)による距離などが提案されている．いずれにしても，探索キーが与えられた場合に，その相手とするキーの数を少なくする方法が，効率のよい修正のためには必須である．この方向の研究はあまり進んでおらず，全数検査を仮定した実験的・理論的考察に限られているようである．

（3） 辞書の付加情報による方法

入力綴りと'充分に近い'キーが決定できるときは，そのキーに付加された情報によって，修正が可能な場合がある．語幹キーの項で例としたimplyesも，implyからimpliesが導かれた点で，綴りの比較による修正ができる．誤り綴りも登録しておく全綴キーの方法[6),12]も，原理的には同じ修正法となる．

短縮キーの方法でも，キーにもとの綴りを付加しておけば，修正処理がある程度可能である[1]たとえば．

---

* 実際には文字の重複によりこれより少なくなる．

表-3 隣接2字組の頻度表（文献 3）より転載）

LETTER OF THE SECOND POSITION

LETTER OF THE FIRST POSITION	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	SPACE
A	1	32	39	15		10	18		16		10	77	18	172	2	21	1	101	67	124	12	24	7		27	1	49
B	8				58				6	2		21	1		11			6	5		25				19		2
C	44		12		55	1		46	15		8	16			59	1		7	1	38	16		1				7
D	45	18	4	10	39	12	2	3	57	1		7	9	5	37	7	1	10	32	39	8	4	9		6		222
E	131	11	64	107	39	23	20	15	40	1	2	46	43	120	46	32	14	154	145	80	7	16	41	17	17		446
F	21	2	9	1	25	14	1	6	21	1		10	3	2	38	3		4	8	42	11	1	4		1		99
G	11	2	1	1	32	3	1	16	10			4	1	3	23	1		21	7	13	8		2		1		50
H	84	1	2	1	251	2		5	72			3	1	2	46	1		8	3	22	2		7		1		68
I	18	7	55	16	37	27	10				8	39	32	169	63	3		21	106	88		14	1	1		4	2
J					2										4						4						
K					28				8					3	3				2	1			3		3		17
L	34	7	8	28	72	5	1		57	1	3	55	4	1	28	2	2	2	12	19	8	2	5		47		49
M	56	9	1	2	48			1	26				5	3	28	16			6	6	13		2		3		40
N	54	7	31	118	64	8	75	9	37	3	3	10	7	9	65	7		5	51	110	12	4	15	1	14		135
O	9	18	18	16	3	94	3	3	13		5	17	44	145	23	29		113	37	53	96	13	36		4	2	83
P	21	1			40			7	8			29			28	26		42	3	14	7		1		2		13
Q																					20						
R	57	4	14	16	148	6	6	3	77	1	11	12	15	12	54	8		18	39	63	6	6	10		17		95
S	75	13	21	6	84	13	6	30	42		2	6	14	19	71	24	2	6	41	121	30	2	27		4		274
T	56	14	6	9	94	5	1	315	128			12	14	8	111	8		30	32	53	22	4	16		21		196
U	18	5	17	11	11	1	12	2	5			28	9	33	2	17		49	42	45				1	1	1	5
V	15				53				19						6												
W	32		3	4	30	1		48	37			4	1	10	17	2		1	3	6	1	1	2				7
X	3		5		1				4						1	4				1	1						3
Y	11	11	10	4	12	3	5	5	18			6	4	3	28	7		5	17	21	1	3	14				132
Z					5				2			1														1	
SPACE	247	85	94	91	55	84	25	95	118	14	4	45	79	43	142	82	1	62	138	363	28	8	127	1	15	1	

4文字短縮とすると

souvenir（正）と souviner（誤）

はどちらも sovr に縮められる．そこで，ハッシュ法での場合と同じように，キー（sovr）を見つけたあと，そこに書かれている正しい綴りと比較すれば，少々誤っている綴りも修正することができる．誤りの程度が大きい場合には短縮形まで変化してしまうので，この方法では修正処理はできない．また，短縮キーのほかにもとの綴りを記録しておく必要があるので，辞書を小さくできるという特長は失なわれる．ただし，探索効率は変らない．

4. 多字組頻度による検査と修正

英単語の集合では，文字単位の隣接頻度にかなりの偏りがある．たとえば，hに続く文字は圧倒的にeが多く，a，i，o，tであることもあるが，gやjであることは滅多にない．本章では，この頻度統計を基礎とした検査・修正法の概略を示す．

表-4 高頻度の2字組と3字組（文献11)による）

（□は空白を表す）

2字組

e□	□t	th	he	□a	s□	in	er
t□	d□	re	an	□i	n□	□o	on
es	at	en	□s	y□	ti	nd	nt

……

3字組

□th	the	he□	□of	of□	□in	and
□an	nd□	□to	to□	ion	ed□	ing
ent	is□	in□	tio	□co	re□	□a□

……

4.1 多字組と頻度表

多字組[3),7),11),16)](n-gram) とは，複数個の英文字を組にしたもので，2字組 (digram)，3字組 (trigram) などがよく使われる．また，単語の前後を示す空白や省略を示す引用符を，英文字に含ませることもある．n字組の種類は，文字集合の大きさをSとして，S^nがあるが，このそれぞれについて，英単語中での頻度を求めたものを，**多字組頻度表**という．これには，

i）隣接多字組の頻度そのものを記録した**完全頻度表**[3)]

ii）各頻度を，あるしきい値より大きいか小さいかを示す1ビットで表わす**2値頻度表**，

iii）多字組中の各文字の単語中の位置まで指定した**位置別頻度**[16)]

などの種類がある．完全頻度表の例[3)]を**表-3**に，高頻度の2字組と3字組の例[11)]を**表-4**に，それぞれ示す．位置別のn字組頻度表は，長さlの単語まで考えるものとして，

$$_lC_n \cdot S^n$$

ものエントリを持つので，2値頻度が使われることが多い．2値頻度の有効性は，完全頻度表の0要素が，2字組で約30%，3字組で約75%であることから裏づけられる[7),16)]．

4.2 多字組による検査と修正

代表的な2例を述べる．

（1） **完全頻度による方法[3)]**

2字組を例にとって説明する．例は文献3)による．入力綴りの各文字について，直前の文字との2字組頻度と，直後の文字との2字組頻度とを求め，積を計算する．この値が小さい文字が'怪しい'と判定される．例を示す．

入力： g g e a t

頻度：25 1 32 131 124 196

積： 25 32 4192 16244 24304

頻度は表-3参照のこと．この結果，先頭のg，次に2番目のgが，この順に怪しいと判定される．修正は，辞書を併用して行われることが多い．上例では，まず先頭の1文字をaからzまで変えた26種の綴りを，頻度積の大きい順に辞書中で探索する．実際の順は

ageat, ngeat, …, cgeat, bgeat

となるが，これらの探索はすべて失敗する．次に，2文字目について同様の処理をすると

gheat, great, geeat, …

という順の great で探索が成功する．

挿入や削除を想定した修正はこれよりかなり複雑となるが，原理的には同様の処理となる．2字組を使ったこの方法による修正率は約74%であるという報告がある[3)]．これがかなり低い理由は，もとの正しい綴りが，かならずしも最大の頻度積を与えないことによる．

（2） **位置別2値頻度による方法[16)]**

この方法では，入力綴りの中のすべての位置別多字組について，その2値頻度を求める．たとえば，綴りqeatに含まれる位置別3字組は，文字位置を<と>で囲んで表わすとすると

qea<1, 2, 3>, qet<1, 2, 4>,

qat<1, 3, 4>, eat<2, 3, 4>

の4個である．このすべての頻度が'1'，すなわち'きわめて稀ではない'と判定されれば，入力綴りは正しいものと判定される．**表-5**に，長さ6の単語に1文字置換を1回施したものをデータとした，誤り検出能力の比較を[16)]を示す．この方法が割合よい結果を与えている理由は，多字組を位置別に分割したことにより，それぞれの頻度表（S^n 要素）がかなり粗になることである．たとえば，表-5の実験で使用した位置別3字組の頻度表における'1'の密度は，300単語で1.5%，800単語で3.5%，約3000単語でも約8%にすぎない[16)]．

表-5 綴り誤りの検出率（文献16)による）
単語の長さは6．誤りは置換ひとつで人工的に発生．
単位はパーセント

検出法	単語集合の大きさ 300	800	1300	2755
辞書法	99.8	99.8	99,8	99.7
隣接4字組	99.5	99.2	98.0	96.9
位置別3字組	99.8	99.7	99.7	98.6
隣接3字組	96.1	88.0	85.4	78.0
位置別2字組	95.0	84.6	78.9	69.6
隣接2字組	63.4	48.4	44.2	32.5

表-6 綴り誤りの修正率（文献16)による）
（条件は表-5のそれと同じ）

修正法	単語集合の大きさ 300	800	1300	2755
辞書法	95.7	92.2	89.8	84.1
隣接4字組	88.0	73.9	65.4	47.8
位置別3字組	95.5	88.1	81.4	62.4
隣接3字組	36.4	16.1	8.0	2.8
位置別2字組	28.6	6.7	4.4	0.5
隣接2字組	0.5	0.3	0.0	0.0

修正処理の概略はつぎのとおりである．

i）頻度'0'の多字組の位置から，誤り文字の位置を決める．たとえば上例において

qea<1, 2, 3>, qet<1, 2, 4>, qat<1, 3, 4>

の3個が頻度0を与えたとすると，1番目の文字が誤りであると判定される．

ii）誤り文字だけを別の文字に変えて，すべての多字組頻度が'1'となるようにする．上の例では

B_1='αea<1, 2, 3>の頻度が'1'であるαの集合'

B_2='αet<1, 2, 4>　〃　〃'

B_3='αat<1, 3, 4>　〃　〃'

という3個の文字集合の共通部分として，求める文字が決定される．

実際には，誤り文字の位置が決定できないこともある．その場合には，2カ所の誤り，挿入や削除の誤りといった仮定を順々に置いて調べなおす．この場合には，修正の能力はかなり低くなる．**表-6**に，表-5の実験における誤り修正能力を示す[16)]．

5. おわりに

綴りの処理を行うシステムは，すでに数多く実用化されている．完全なプログラム・リストを載せた文献もある[14]．前章までに示したような，割合不完全と思われる処理方法でも実用的であるのは，それらが完全自動修正を目的として使われることがないからである．ほとんどの場合，もとの綴りの情報は得られず，完全な修正というものは理論的にも存在しない．それで，多くのシステムが，検査や修正の結果を表示する対話形の構成となっている[11]．

同様の理由で，この分野に関連する文献を参照する場合には，統計や実験のデータの根拠に注意することが重要である．たとえば，多字組頻度の値にも文献によってかなりの変動が見られる．また，修正率の結果も，母集団である綴りの集合の作り方によって，かなり変ることが多い．辞書中の単語に一様乱数的な誤りを入れたものと，現場のタイピングによる生のデータとでは，結果が異なるのが当然である．綴り処理の研究発展のためには，新しいアルゴリズムや理論もさることながら，方式評価の共通データの構成も，重要な1項目となると思われる．

参考文献

1) Blair, C.R.: A Program for Correcting Spelling Errors, Inf. Control, Vol. 3, No. 1, pp. 60-67 (1960).
2) Carlson, G.: Techniques for Replacing Characters that are Garbled on Input, in Proc. SJCC 1966, AFIPS Press, Arlington (1966).
3) Cornew, R.W.: A Statistical Method of Spelling Correction, Inf. Control, Vol. 12, No. 2, pp. 79-93 (1968).
4) Damerau, F.J.: A Technique for Computer Detection and Correction of Spelling Errors, Comm. ACM, Vol. 7, No. 3, pp. 171-176 (1964).
5) Galli, E. J. and Yamada, H.M.: An automatic dictionary and verification of machine-readable text, IBM Syst. J., Vol. 6, No. 3, pp. 192-207 (1967).
6) Galli, E. J. and Yamada, H.M.: Experimental Studies in Computer-Assisted Correction of Unorthographic Text, IEEE Trans. Eng. Writing and Speech, Vol. EWS-11, No. 2, pp. 75-84 (1968).
7) Harmon, L.D.: Automatic Recognition of Print and Script, Proc. IEEE, Vol. 60, No. 10, pp. 1165-1176 (1972).
8) James, E.B. and Partridge, D.P.: Tolerance to inaccuracy in computer programs, Comput, J., Vol. 19, No. 3, pp. 207-212 (1976).
9) Knuth, D.E.: The Art of Computer Programing, Vol. 3: Sorting and Searching, Addison-Wesley, Reading, Mass. (1973).
10) Lowrance, R. and Wagner, R.A.; An Extension of the String-to-String Correction Problem, J. ACM, Vol. 22, No. 2, pp. 175-183 (1975).
11) McMahon, L.E., Cherry, L.L. and Morris, R.: Statistical Text Processing, Bell Syst. Tech. J., Vol. 57, No. 6, part 2, pp. 2137-2154 (1978).
12) Mor, M. and Fraenkel, A.S.; A Hash Code Method for Detecting and Correcting Spelling Errors, Comm. ACM, Vol. 25, No. 12, pp. 935-938 (1982).
13) Okuda, T., Tanaka, E. and Kasai, T.: A Method for the Correction of Garbled Words Based on the Levenshtein Metric, IEEE Trans. Comput. Vol. C-25, No. 2, pp. 172-177 (1976).
14) Peterson, J.L.: Design of a spelling program: An experiment in program design, Lecture Notes in Comput. Sci. 96, Springer-Verlag, New York (1980).
15) Peterson, J.L.: Computer Programs for Detecting and Correcting Spelling Errors, Comm. ACM, Vol. 23, No. 12, pp. 676-687 (1980).
16) Riseman, E.M. and Hanson, A.R.: A Contextual Postprocessing System for Error Correction Using Binary n-Grams, IEEE Trans. Comput., Vol. C-23, No. 5, pp. 480-493 (1974).
17) Sheil, B.A.: Median Split Trees: A Fast Lookup Technique for Frequently Occuring Keys, Comm. ACM, Vol. 21, No. 11, pp. 947-958 (1978).
18) Thorelli, L.E.: Automatic correction of errors in text, BIT, Vol. 2, No. 1, pp. 45-52 (1962).
19) Turba, T.N.: Checking for Spelling and Typographical Errors in Computer-Based Text, Sigplan Notices, Vol. 16, No. 6, pp. 51-60 (1981).

(昭和58年1月27日受付)