コンフリクト・フラグをもったハッシュ記憶法*

古　川　康　一**

Abstract

The mean reject time for the scatter storage technique with conflict flag is given. (The reject time means the time required to identify an item as a nonmember of the given table).

Comparisons with some other scatter storage methods are made. It is shown that the one-bit information reduces in the reject time less than 1/2 when the load factor exceeds 60%. Some applications of this method are mentioned.

1. はじめに

表検索において，そのファイルの大きさ（項目数）によらずに一定の回数（統計的に）で情報をとり出すことができる方法に，ハッシュ記憶法[1]がある．この方法は，表へ項目を登録する場面と検索する場面が分離できないような問題に対して，特に有効である．それは，ソーティングのような項目全体に対する前処理を要しないからである．また，項目の追加，削除も簡単に項目毎に行えるので，動的なファイルに対しても有効である．(静的なファイルに対しても，検索時間に関しては，他の方法よりも優れているが，順序を表わせないというような欠点もある．) 最もよく使われるのは，コンパイラやアセンブラなどの言語プロセッサでの記号処理に対してである．

ハッシュ記憶法は，前に述べたように，検索表の作り方の1つである．それを，項目のキーと番地付けの関係からみると，①キーとデータを，表を構成する記憶領域にすきまなく詰めていく方法（1つのデータ項目の長さは一定とする）と，②キー自身を番地として，記憶領域のその場所にデータを置く（キーは置かなくてよい）方法の中間であると考えられる．ハッシュ記憶法では，キーにある演算をほどこして，表の番地に変換し，そこにキーとデータをおく．つまりキーの持っている情報が番地を求めるのに使われる点では②と同じであるが，この変換が1対1でないために，①と同様キーを表中に持たなければならない．(キーの情報のうち，番地を求めるために使われた分は置く必要はないが，普通は簡単のためそのまま置くことが多い．) ①の長所は記憶領域に無駄がないことで，欠点は検索時間がファイルの大きさに依存することである．これにくらべて②は全く反対で，検索時間はファイルの大きさによらず一定の高速処理ができるが，通常，記憶領域は，非常に無駄を生じる．(キーが，キーとしての性格を強めれば強めるほど，その情報量つまりキーの長さは，ファイルの中でその項目を識別するのに要する情報量よりはるかに大きくなるからである．)

ところで，ハッシュ記憶法は，この両者の長所をある程度保存しており，全体として，システムの効率を著しく改善するのに役立つ．つまり，記憶領域をそれほど無駄にせず，ファイルの大きさによらない，ほぼ一定の処理速度を実現できる．

一方，変換が1対1でないということが，この方法の処理を複雑にしている．異ったキーが同じ番地に変換される現象をコンフリクトというが，このコンフリクトの処理方法によって検索速度が変ってくる．これまでにいくつかの方法が提案され，それらの効率が計算されているが，それらは全て表中に存在する項目に対しての平均探索回数 (average probe number) である．

この論文では，表中に存在しない項目に対する検索時間，すなわち，それが表に含まれていないと判定されるまでの時間（これを棄却時間と呼ぶ）を短くするようなアルゴリズムを与えた．

検索表の各セルには，空かどうかを示す1ビットのフラグの他に，コンフリクトを引き起こしたかどうかを示す1ビットのフラグをもたせる．このフラグによって，存在しない項目に対する検索時間を大幅に短縮することができることがわかった．それに加えて，こ

* Hash Addressing with Conflict Flag, by Koichi Furukawa (Information System Section, Software division, Electrotechnical Laboratory)

** 電子技術総合研究所ソフトウェア部情報システム研究室

のフラグは項目の削除に対しても重要な働きをすることがわかった．つまり，このフラグを用いて，簡単に項目を削除できることが示された．

2. では，コンフリクト・フラグを使ったときの項目の登録，検索および削除のアルゴリズムを与える．3. では，この方法での平均棄却時間を計算によって求め，コンフリクト・フラグをつけないときの平均棄却時間および直接連鎖法による平均棄却時間との比較を行う．4. では，この方法の二，三の応用例について述べる．

2. 項目の登録，検索および削除のアルゴリズム

ハッシュ表の大きさを N とし，その各番地を $1\sim N$ とする．各セルは，次のようなフィールドから成っているものとする．

UFLAG （1ビット・フィールド）
CFLAG （1ビット・フィールド）
KEY （キー・フィールド）

UFLAG はこのセルが使われている (used) かどうかを示す1ビットのフラグで，使われているときは1で，使われていないとき（すなわち空セル）は0である．CFLAG はコンフリクト・フラグで，CFLAG=1 は，他の項目がこのセルをアクセスした後に他の場所へ登録された（すなわち，コンフリクトがこのセルで生じた）ことを示す．登録されるキーを K で表わし，ハッシュ関数系を $h_0, h_1, \cdots, h_i, \cdots$ で表わす．またハッシュ関数の添え字変数を i で表わす．キー K にハッシュ関数 h_i をほどこした結果を $h_i[K]$ で表わす．これは，$h_0[K], h_1[K], \cdots, h_{i-1}[K]$ と異なる $1\sim N$ の中のある整数値をとる．各セルは，データ領域（またはデータ領域を指すポインタ）を持っているが，ここでは不要であるので取り上げない．以下のアルゴリズムでは，Knuth[2] による記述方法に従っている．ここで P および $P0$ はポインタ変数で，ハッシュ表のセルの1つを指している．P で指されたセルの各フィールドは，CFLAG(P) のように書き表わされる．α は表占有率の上限を与える．NUMBER は，表のアフレを管理する変数で，ハッシュ表の外にあるものとする．

2.1 登録 (Enter)

Algorithm E (Enter)

E1 [Check number] If NUMBER$\geq\alpha N$, the algorithm terminates. (overflow.)

E2 [Calculate h_0] Set $P\leftarrow h_0[K]$, $i\leftarrow 1$, $P0\leftarrow\Lambda$. ($P0$ is a pointer used to point the first empty cell.)

E3 [Check emptiness] If UFLAG(P)=0 and $P0=\Lambda$, set $P0\leftarrow P$, and go to E5.

E4 [Check multi-entry] If KEY(P)=K, the algorithm terminates (multi-entry).

E5 [Check conflict] If CFLAG(P)=1, set $P\leftarrow h_i[K]$, $i\leftarrow i+1$, and go to E3.

E6 [Check $P0$] (Now, P points to the end cell of the conflict chain.) If $P0=\Lambda$, set $P0\leftarrow P$, and go to E7; otherwise set KEY ($P0$) $\leftarrow K$, UFLAG ($P0$) $\leftarrow 1$, NUMBER$\leftarrow$NUMBER$+1$. The algorithm terminates successfully.

E7 [Find empty cell] Set CFLAG($P0$)$\leftarrow$1, $P0\leftarrow h_i[K]$, $i\leftarrow i+1$. If UFLAG($P0$)=0, go to E6 (Empty cell was found.); otherwise go to E7. (Here, the pointer $P0$ is used instead of P.)

このアルゴリズムは，2つの部分に分けられる．それは E1～E5 および E6～E7 である．前半の部分は，項目 K に対するコンフリクト・チェイン ($h_i[K]$, $i=0, 1, 2, \cdots$ によって得られるセルの系列で，CFLAG が初めて0になるセルまでの部分) をサーチして，空セルを見つけたり，重複登録をチェックしたりする．複数個の空セルがあるときは，最初に見つかった空セルに登録することにする．それは，その方が検索の効率が良いからである．E6 に来て初めて重複登録のないことがわかるので，この時点ですでに空セルが見つかっていれば，そこに登録し，見つかっていなければ，コンフリクト・チェインを伸ばして，新たな空セルを見出す．

もし表内で項目を他のセルに移すことが許されていれば（すなわち，外部で，その項目を，それを含むセルの番地によって参照していなければ），重複登録の場合，すでにその前に空セルが見つかっていれば，それを，空セルに移すことによって，局所的に garbage collection がなされる．

2.2 検索 (Retrieve)

検索は登録に比べて，ずっと容易である．また，コンフリクト・フラグにより，棄却時間を短縮できる．

Algorithm R (Retrieve)

R1 [Calculate h_0] Set $P\leftarrow h_0[K]$, $i\leftarrow 1$.

R2 [Found?] If KEY(P)=K and UFLAG(P)=1, the algorithm terminates successfully. (P

points the desired cell.)

R3 [Check conflict] If CFLAG(P)=0, the algorithm terminates with failure. (Not found.)

R4 [Calculate h_i] Set $P \leftarrow h_i[K]$, $i \leftarrow i+1$. Go to R2.

2.3 削除 (Delete)

項目の削除は，UFLAG を 0 にすることによって行われる．コンフリクト・フラグは変えないので，そのセルでコンフリクトを生じた項目の検索にはさしつかえない．ただし，その分だけ棄却時間が余分にかかり，コンフリクト・フラグの効果が減少する．

Algorithm D (Delete)

D1 [Calculate h_0] Set $P \leftarrow h_0[K]$, $i \leftarrow 1$.

D2 [Found?] If KEY(P)=K and UFLAG(P)=1, Set UFLAG(P)$\leftarrow$0. The algorithm terminates succesfully.

D3 [Check conflict] If CFLAG(P)=0, the algorithm terminates with failure. (not found.)

D4 [Calculate h_i] Set $P \leftarrow h_i[K]$, $i \leftarrow i+1$. Go to D2.

セルの削除によってコンフリクト・フラグは減少しないので，登録，削除を頻繁にくり返すと，コンフリクト・フラグが 1 のセルが増大していく．これを防ぐには，1 ビットのコンフリクト・フラグのかわりにコンフリクト・カウンタをおいて，登録時に 1 を加え，削除時に 1 を引けばよい．この場合注意しなければならないのは，重複登録や，未登録項目の削除などの誤動作の場合である．そのときは，すでに加え，あるいは差し引いたカウンタをもとの値に直しておかなければならない．(あるいは誤動作でないことを確めてから 1 を加えるか減ずるかを行なってもよい．)

こうすれば，削除するときに，その項目だけがコンフリクトをおこしていたセルのコンフリクト・カウンタは 1 から 0 になるので，コンフリクトを起していないセルに変る．

3. 棄却時間の計算

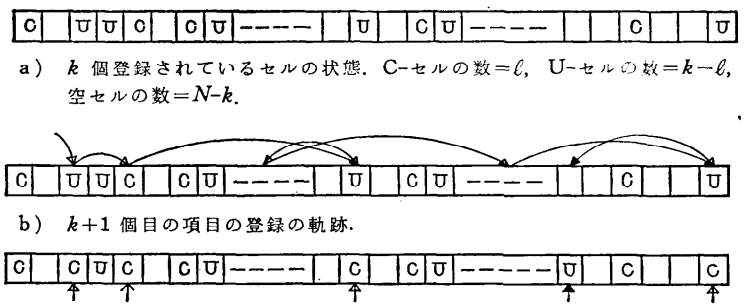
ハッシュ表での探索回数は，一定ではなく，確率的に変動する．以下の計算では，表の各セルに変換される確率がすべてのセルで等しいと仮定する．また，コンフリクトを起こした項目は，残りのセル（すでにアクセスしたセル以外のセル）に対して等確率でアクセスする（そのセル番号に変換される）ものと仮定する．実際には，コンフリクトを起こした項目同志が干渉し合う場合が多く（例えば，線形探索法と呼ばれている方法では，次の番地のセルをアクセスするが，この場合，コンフリクトは必ず表の部分的塊りを生じ，この塊りは，他の項目のコンフリクトの処理をおそくする），後者の仮定が完全に成り立つようなアルゴリズムは数少ない[3),4)]．

いま，ハッシュ表（表長を N とする）に k 個の要素を登録したときにコンフリクト・フラグが 1 であるセルが j 個ある確率を $P_k(j)$ とする．(途中で項目の削除はないものとする．) $P_k(j)$ に対して，次の漸化式が成り立つ．

$$P_{k+1}(j)=\sum_{l=0}^{j} P_k(l)\cdot\sum_{i=0}^{l}\binom{j-l+i}{i}\cdot(N-k)\cdot\prod_{h=0}^{i-1}(l-h)\cdot\prod_{g=0}^{j-l-1}(k-l-g)\cdot\prod_{m=0}^{i+j-l}(N-m)^{-1}.$$

(ここで $\prod_{g=0}^{-1}(k-l-g)\equiv 1$ および $P_k(k)\equiv 0$ と定義する.)　　(1)

(1) 式における $P_k(l)$ の係数は，k 個の項目を登録したときにコンフリクト・フラグが l 個たっていて，次の 1 個を入れたときに，それが j 個になる（新たに $j-l$ 個のフラグが 0 から 1 になる）確率を与える．この和の第 i 項 $\binom{j-l+i}{i}(N-k)\prod_{h=0}^{i-1}(l-h)\cdot$



a) k 個登録されているセルの状態．C-セルの数=ℓ，U-セルの数=$k-\ell$，空セルの数=$N-k$.

b) $k+1$ 個目の項目の登録の軌跡.

c) $k+1$ 個目の項目の登録後の状態．↑：C-セルにアクセスした（j 個）
↟：U-セルが C-セルに変わった　($j-\ell$個)，↑：空セルがU-セルに変わった（1 個），C-セルの数=$j(=\ell-(j-\ell))$，U-セルの数=$k+1-j$，空セルの数=$N-(k+1)$.

図 1　第 $k+1$ 項目の登録によるセルの状態の変化

$\prod_{g=0}^{j-l-1}(k-l-g)\prod_{m=0}^{i+j-l}(N-m)^{-1}$ は，このとき，すでにコンフリクト・フラグが1であるセル（これを C-セルと呼ぶ）を i 個アクセスした場合を表わしている．(C-セルは何度アクセスしても，コンフリクト・フラグの変化には関係がない．) つまり l 個の C-セル，$k-l$ 個の，コンフリクト・フラグが 0 でしかも占有されているセル（これを U-セルと呼ぶ），および $N-k$ 個の空セルがあったときに，C-セルに i 回，U-セルに $j-l$ 回アクセスした後に，初めて空セルにアクセスする確率である．この項の先頭にある2項係数は，C-セルと U-セルのアクセスの順序を数え上げるためである．このときのセルの状態の変化を第1図に示す．

(1)式は，次式のように書きかえることができる．

$$P_{k+1}(j)=\sum_{l=0}^{j}P_k(l)\cdot\frac{N-k}{N}\cdot\prod_{g=0}^{j-l-1}\frac{k-l-g}{N-g-1}\cdot\sum_{i=0}^{l}\binom{j-l+i}{i}\cdot\prod_{h=0}^{i-1}\frac{l-h}{N-j+l-h-1}. \tag{2}$$

ここで，

$$f_j(l)=\sum_{i=0}^{l}\binom{j-l+i}{i}\prod_{h=0}^{i-1}\frac{l-h}{N-j+l-h-1} \quad \text{for } l\leq j<N \tag{3}$$

とおくと，(2)式は，

$$P_{k+1}(j)=\sum_{l=0}^{j}P_k(l)\cdot\frac{N-k}{N}\cdot\prod_{g=0}^{j-l-1}\frac{k-l-g}{N-g-1}\cdot f_j(l) \tag{2}'$$

となる．この $P_{k+1}(j)$ をより簡単な式で表わしたいのであるが，まず $f_j(l)$ に関して，次の補助定理が成り立つ．(証明は，付録1参照.)

補助定理 1 $N>j\geq l\geq 1$ なる j，l に対して，$f_j(l)$ は，次式で与えられる．

$$f_j(l)=\prod_{h=0}^{l-1}\frac{N-h}{N-j+l-h-1}. \tag{4}$$

補助定理1と(2)′式より，次の補助定理2が得られる．

補助定理 2 $P_{k+1}(j)$ は，$j=1, 2, \cdots, k$ に対して

$$P_{k+1}(j)=\frac{N-k}{\prod_{i=0}^{j}(N-i)}\sum_{l=0}^{j}P_k(l)\prod_{g=0}^{j-l-1}(k-l-g)\cdot\prod_{h=0}^{l-1}(N-h) \tag{5}$$

で与えられる．また $j=0$ に対しては，

$$P_{k+1}(0)=\frac{N-k}{N}P_k(0) \tag{6}$$

で与えられる．

補助定理2の(6)式は，(1)式から直接得られる．この補助定理より，容易に次の定理を得る．(証明は，付録2を参照.)

定理 1 $P_{k+1}(j)$ は，$P_k(m)$ $(m=0, 1, 2, \cdots, j)$ と，$P_{k+1}(m)$ $(m=0, 1, \cdots, j-1)$ により，次式のように表わされる．

$$P_{k+1}(j)=\frac{N-k}{N-j}\left(\sum_{m=0}^{j}P_k(m)-\sum_{m=0}^{j-1}P_{k+1}(m)\right). \tag{7}$$

次に，k 個の要素が登録されているときの平均棄却時間 $E_c(k)$ を求める．j 個の要素が登録されていて，次の1個を登録するときの平均探索回数を $E_n(j)$ とすると，$E_c(k)$ は，次の定理により与えられる．

定理 2 k 個の要素が登録されているときの平均棄却時間 $E_c(k)$ は，

$$E_c(k)=\sum_{j=0}^{k-1}P_k(j)\cdot E_n(j) \tag{8}$$

で与えられる．

証明．N 個のセルのうち j 個が C-セルのときの平均棄却時間は，j 個の要素が登録されているときに，次の1個を登録するときの平均探索回数に等しい．また，j 個の C-セルができる確率は $P_k(j)$ で与えられるので(8)式が成り立つ． 証終.

ここで $E_n(j)$ は，

$$E_n(j)=\frac{N+1}{N-j+1} \tag{9}$$

で与えられる[1]．

定理1は，$P_{k+1}(j)$ が，容易に数値計算により求められることを示している．その結果と，定理2および(9)式より $E_c(k)$ を求めることができる．

次に，この方法を直接連鎖法 (direct chaining method[1]) と比較するために，直接連鎖法での平均棄却時間 $E_d(k)$ を求める．$E_d(k)$ について，次の定理が成り立つ．

定理 3 直接連鎖法による平均棄却時間 $E_d(k)$ は，次の近似式により与えられる．

$$E_d(k)\approx e^{-\alpha}+\alpha. \tag{10}$$

ただし $\alpha=k/N$.

証明．k 個の要素を登録したとき，長さ i の連鎖が n_i 個できるとすると，そのときの棄却時間 $t_d(k)$ は，

$$t_d(k)=\frac{n_0}{N}+\sum_{i=1}^{k}\frac{n_i}{N}\cdot i=\frac{n_0}{N}+\frac{k}{N} \tag{11}$$

となる．ここで n_0 は空セルの数である．上の式は，空セルにアクセスしたときと，長さ1の連鎖にアクセ

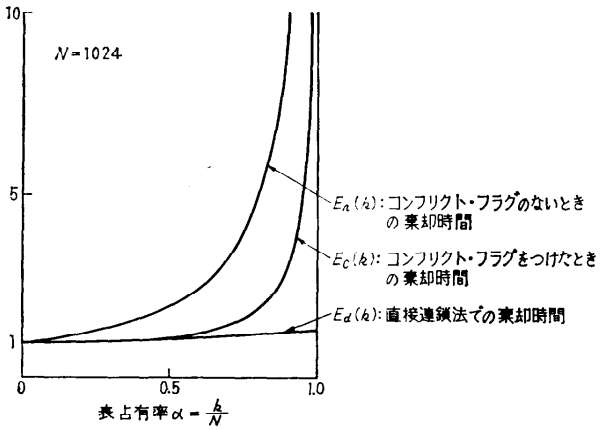


図 2　各方法の棄却時間の表占有率に対するグラフ

スしたときの探索回数を共に1としたが，実際には，長さ1の連鎖の場合の方が，キーの比較時間および，次のセルへのポインタの検査の分だけ余分にかかる．しかし，その差はハッシュ関数をほどこして初めの探索セルを求める演算に比べて無視できる（キーが長くて，比較に時間がかかる場合は別であるが）ので，ここでは，両者を同等に扱った．

N と k が十分に大きいとき，n_0 は $\lambda=Ne^{-\alpha}$ のポアソン分布

$$P(n_0;\lambda)=e^{-\lambda}\frac{\lambda^{n_0}}{n_0!} \tag{12}$$

に従うことが知られている[5]．n_0 の期待値は，λ で与えられるので，

$$E_d(k)=\frac{1}{N}\cdot Ne^{-\alpha}+\alpha=e^{-\alpha}+\alpha.$$

証終．

第2図に，$E_c(k)$, $E_n(k)$ および $E_d(k)$ のグラフを与えた．

この図からわかるとおり，表占有率 α が小さいところでは，$E_c(k)$ は $E_d(k)$ に近く，α が1に近いところでは $E_c(k)$, $E_n(k)$ はともに非常に大きくなる．

$\alpha=0.6$ ぐらいのところで $E_n(k)\simeq 2E_c(k)$ となり，棄却時間が半分以下になる．

4. この方法の応用例

この方法の直接の応用は，ある集合に，特定の要素が属しているかどうかを判定する問題である．この場合には，多くの項目がその集合に属していないことは十分にあり得るので，棄却時間の短縮によってシステム全体の効率は良くなる，例えば，アセンブラでの命令表は，普通，機械命令と，いくつかの擬似命令からできている．またマクロ・アセンブラでは，この他にマクロ命令があるが，それらはプログラマによっても定義できるので，これは命令表にはない．その場合，アセンブラがマクロ命令に出会うと，それは必ず棄却される．故にこの場合には，集合要素判定問題に帰着される．

この方法は，ハッシュ記憶法の最も困難な問題であるハッシュ表のアフレの処理に使うこともできる．アフレの処理としては，2通り考えることができる．1つは，すべての項目をより大きい表に再びハッシュして作り直す方法である．これは，恒久的なファイルの場合などには適している．他の方法は，アフレた分の項目をすべて他の場所に置くことである．これは，アセンブラやコンパイラなどの記号表のように一時的にしか使われない表の場合に適している．

第2の方法は，やはり集合要素判定問題を含んでいると考えることができる．何故ならば，はじめの表からアフレた項目に対しては，その表を構成する項目の集合に属していないからであり，それに対する棄却時間の短縮は，その項目の探索回数を減らすことになるからである．アフレた項目自身も同様のハッシュ表で管理することができる．これを一般化して，いくつものハッシュ表をリスト状につなげれば，システム全体としてのアフレを防ぐことができる．

いま，アフレ点を適当に定め（表占有率が α を越えたとき，アフレが生じたとする），そのときの棄却時間の平均を t_α とする．k 個のリストがすべて一杯のときの平均探索回数 E_k は，

$$\begin{aligned}E_k&=\frac{1}{k}E_1+\frac{1}{k}(t_\alpha+E_1)+\frac{1}{k}(2t_\alpha+E_1)+\cdots\\&\quad+\frac{1}{k}\{(k-1)t_\alpha+E_1\}\\&=\frac{k-1}{2}t_\alpha+E_1\end{aligned} \tag{13}$$

で与えられている．ここで E_1 は，$k=1$ の場合で，表占有率が α のときの平均探索回数である．

(13) 式からわかるように，E_k は棄却時間 t_α にほぼ比例している．故に棄却時間の短縮は，直接 E_k の短縮につながる．逆に言えば，t_α が小さくできることによってはじめてこの方法が有効に使い得るということができる．

謝 辞

本研究の機会を与えていただいた西野博二ソフトウェア部長，加藤雄士情報システム研究室長，ご討論いただいた情報システム研究室の諸氏および有沢誠氏に感謝する．

参考文献

1) Morris, R.: Scatter storage techniques. CACM 11, 1 (Jan. 1968), pp. 38～44.
2) Knuth. D. E.: The art of computer programming. Vol. 1/Fundamental Algorithm. Addison-Wesley, pp. 1～9.
3) Bell, J. R.: The quadratic quotient method: A hash code eliminating secondary clustering. CACM 13, 2 (Feb. 1970), pp. 107～109.
4) Bell, J. R. & Kaman C. H.: The linear quotient hash code. CACM 13, 11(Nov. 1970), pp. 675～677.
5) Feller, W.: An Introduction to Probability Theory and Its Applications, Vol. 1. John Wiley & Sons, New York, 1950.

付 録

1. 補助定理1の証明

これは，l と j に関する帰納法により証明される．まず（3）式を変形すると容易に次式が得られる．

$$f_j(l)=\sum_{i=0}^{l}\binom{l}{i}\cdot\prod_{h=0}^{i-1}\frac{j-l+i-h}{N-j+l-h-1}$$

for $l\leq j<N$. (A·1)

帰納法のステップは，図 A.1 のように進められる．$l=1$ のときは，(A.1) 式より，

$$f_j(1)=1+\binom{1}{1}\frac{j}{N-j}=\frac{N}{N-j},$$

for $1\leq j<N$.

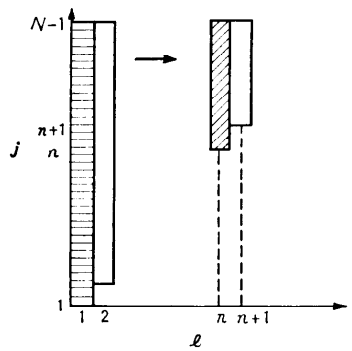


図 A.1 帰納法のステップ

故に，（4）式は $l=1$, $1\leq j<N$ に対して成り立つ．（図 A.1 の横線の領域．）

いま，$l=n$, $n\leq j<N$ に対して（4）式が成り立つとすると（斜線部分），$l=n+1, n+1\leq j<N$ に対して，

$$f_j(l)=f_j(n+1)$$

$$=\sum_{i=0}^{n+1}\binom{n+1}{i}\prod_{h=0}^{i-1}\frac{j-n-1+i-h}{N-j+n-h}$$

$$=\sum_{i=0}^{n+1}\left\{\binom{n}{i-1}+\binom{n}{i}\right\}\prod_{h=0}^{i-1}\frac{j-n-1+i-h}{N-j+n-h}$$

（ただし $\binom{n}{i}=0$ for $i<0$ or $n<i$）

$$=\frac{j-n}{N-j+n}f_j(n)+f_{j-1}(n)$$

$$=\prod_{h=0}^{n}\frac{N-h}{N-j+l-h-1}.$$

証終.

2. 定理1の証明

（5）式より，次式が成り立つ．

$$(N-m-1)P_{k+1}(m+1)-(k-m)P_{k+1}(m)$$
$$=(N-k)P_k(m+1). \quad (A.2)$$

∵）

$$\text{左辺}=(N-m-1)\cdot\frac{N-k}{\prod_{i=0}^{m+1}(N-i)}\cdot\sum_{l=0}^{m+1}P_k(l)\cdot\prod_{g=0}^{m-l}(k-l-g)\cdot\prod_{h=0}^{l-1}(N-h)$$

$$-(k-m)\frac{N-k}{\prod_{i=0}^{m}(N-i)}\cdot\sum_{l=0}^{m}P_k(l)\cdot\prod_{g=0}^{m-l-1}(k-l-g)\cdot\prod_{h=0}^{l-1}(N-h)$$

$$=\frac{N-k}{\prod_{i=0}^{m}(n-i)}\cdot\sum_{l=0}^{m+1}P_k(l)\cdot\prod_{g=0}^{m-l}(k-l-g)\cdot\prod_{h=0}^{l-1}(N-h)-\frac{N-k}{\prod_{i=0}^{m}(N-i)}\cdot\sum_{l=0}^{m}P_k(l)\cdot\prod_{g=0}^{m-l}(k-l-g)\cdot\prod_{h=0}^{l-1}(N-h)$$

$$=\frac{N-k}{\prod_{i=0}^{m}(N-i)}\cdot P_k(m+1)\cdot\prod_{g=0}^{-1}(k-m-1-g)\cdot\prod_{h=0}^{m}(N-h)$$

$$=(N-k)P_k(m+1)$$

$$=\text{右辺}.$$

$$\left(\prod_{g=0}^{-1}(k-m-1-g)=1 \quad (\text{定義より.})\right)$$

(A.2) 式を $m=0, 1, \cdots, j-1$ について加えると,

$$(N-j)P_{k+1}(j)+\sum_{m=1}^{j-1}(N-k)\cdot P_{k+1}(m)-kP_{k+1}(0)$$

$$=(N-k)\sum_{m=1}^{j}P_k(m) \qquad (A.3)$$

となる. (A.3) 式に (6) 式を代入して, $P_{k+1}(j)$ について解くと,

$$P_{k+1}(j)=\frac{N-k}{N-j}\cdot\left(\sum_{m=0}^{j}P_k(m)-\sum_{m=0}^{j-1}P_{k+1}(m)\right)$$

となり, 定理1が得られる.

(昭和47年1月10日 受付)
(昭和47年5月13日 再受付)