記 号 処 理 52－2
（１９８９．９．２９）

圧縮と複写　２つのくず集め方式の接点

寺島 元章
電気通信大学 情報工学科

圧縮と複写という利害得失の異なる２方式を組み合せたくず集めについて提案する。可変容量セルに対する効率的なくず集めとして定評のある圧縮法と複写法の具体例についてその処理時間を解析し、両者の処理時間が現役セルの占有率に対し対照的であることを示す。その結果から処理時間が最適となるような、複写・圧縮選択方式くず集めの具現法について述べる。

Joining of Two Garbage Collectors Based on Compactifying and Copying Schemes

Motoaki Terashima

Department of Computer Science
and Information Mathematics
University of Electro-Communications

1-5-1 Chofugaoka, Chofu-Shi, Tokyo 182 Japan

This paper presents a garbage collector made up of compactifying and copying parts which have merits and demerits contrary to each other. The analysis of processing time for three typical garbage collectors based on either the compactifying or the copying shows that the processing time of copying is in striking contrast to another. An implementation of the garbage collector based on the change (selection) of two parts to make the processing time optimal is also described.

1．はじめに

可変容量セルのくず集めは複写方式と圧縮方式とに大別される。前者はセルの格納領域(heap)を２つの半領域(semispace) に分割し、現役（使用中）のセルを交互に複写することで退役（使用済）のセルをふくむ格納領域を一括回収するという方式である。後者は現役のセルをそれらの配置順序を保存したままで同じ格納領域の一方に圧縮する方式である。表１はこの両者の特徴を対比させて示したものである。

圧縮方式は格納領域量で複写方式より優位にあるが、処理時間では複写方式に劣る。後者の要因は格納領域全体の(１回以上の)走査とポインタ補正による負担増である。

本報告ではまず、圧縮方式に基づく２つの既成のくず集めの間の解析について述べる。次に複写方式の処理時間と対比し、現役（使用中の）セル比率に基づいた両者の利害得失を端的に述べる。この解析結果から得られた、複写・圧縮切換え方式による効率的なくず集め法について提案する。

2．圧縮方式

圧縮方式には、ポインタ補正と再配置（圧縮）とを別個の手順として行う方法と、それらを混ぜて１つの手順として行う方法とがある。前者の例としては 補正表を用いる圧縮法[1,2,3]が、後者の例としては Morrisの圧縮法[1,4]がある。

表1　圧縮方式 vs 複写方式

	処理時間	格納領域
圧縮方式	セルの総容量（格納納領域量）に比例	領域は１つで済む
複写方式	現役セルの総容量に比例	区別可能な２つの同容量の半領域が必要

2.1 補正表を用いる圧縮法

2.1.1 ポインタ補正

現役セル中の各フィールドの特定ビット（これをマークビットと呼ぶ）をオン(on)にするマーキングの手続きが済むと、格納領域は現役セルが連続した１つの塊（A_i, i=0..n、以下単に現役塊と呼ぶ）とそれらの塊の間に存在する退役セルの塊（I_j, j=1..n、以下単に退役塊と呼ぶ）とに分割される。そこで、各現役セルをアドレスの小さいほう(low側)に圧縮することを考えると、A_iは $-|I_1|-|I_2|- \ldots -|I_i| = e_i$だけ番地が変化する。絶対値記号は番地に換算した容量を表す。「番地」とはセルの各フィールドを単位とするアドレスのことである。以下、「番地」はこの意味で用いる。そこで A_i中のセルを指していたポインタは圧縮後も同じセルを指すように補正値(e_i)が加えられる。この作業のことをポインタ補正と呼ぶ。

2.1.2 補正表

補正表は、格納領域を細分化したときの各先頭セルの補正値を登録したものである。その細区画の単位を 2^m番地(m>0)とすると、表の作成は格納領域を走査しながら、$2^m \cdot i+low$番地の補正値をその i番目の成分(E[i])にすることで達成される。改良型圧縮法[3]の補正表はスタック用領域(の未使用域)に作られ 記憶領域の効率的運用に寄与している。

2.1.3 補正値の算出

補正表と格納領域のマークビットの情報から p番地($2^m \cdot j+low \leq p < 2^m \cdot (j+1)+low$) の補正値は以下に述べる手順で算出される。

(1) 補正値が0である塊（A_0）の検出

A_0が low番地から始まる場合、A_0を指すポインタの補正値は0であり ポインタ補正は不要である。そこで、(2.1)式に示すneg番地を補正表作成時に求めておき 補正対象のポインタ(p)と最初に比較する。p<neg ならば補正値は0である。以下補正値が0である不動塊(A_0)のことを単に A_0と呼ぶ。

$$neg = \begin{cases} low+|A_0| & A_0がlow番地から始まる場合 \\ low & A_0がlow番地から始まらずに、その前に退役塊がある場合 \end{cases} \quad (2.1)$$

これは、A_0 を指すポインタの補正を定数時間で行なうことから、処理時間の短縮に寄与する。さらに、A_0 に対応する(一部の)補正表の作成を不要にし記憶領域の節約にも役立つ。

(2) 二分走査

p番地の補正値の算出は 2^m番地単位に分割されたセル格納領域の1区画をその先頭番地($2^m \cdot j+low$, $j=\llcorner (p-low)/2^m \lrcorner$)から p番地まで走査しながらマークビットがオフのフィールドを数えることである。二分走査は、この区画をさらに二等分したその一方の半区画の走査だけで補正値を求める技法である。すなわち、$(p-low) \bmod 2^m < 2^m/2$ ならば low側の半区画を、そうでなければ、high側の半区画を走査する。二分走査により、各フィールドの探索回数の期待値は線形探索の $2^m/2$ から $2^m/4$ になり、算出時間の短縮が図られる。なお、補正値の算出は(2.2)式で与えられる。

$$\begin{aligned} &low側 : E[j] - \| \{ k \mid 2^m \cdot j+low \leqq k < p \text{ かつ } k番地のマークビットがオフ \} \| \\ &high側 : E[j+1]+\| \{ k \mid p < k < 2^m \cdot (j+1)+low \text{ かつ } k番地のマークビットがオフ \} \| \end{aligned} \quad (2.2)$$

ただし、$\|S\|$ は集合Sの要素数を表す。補正表は、$E[0]=0$, $E[i] \leqq 0$ $(i \geqq 1)$である。走査すべき半区画は p-lowを二進表現したときの下位から m番目のビット状態で決まる。このビットは、アセンブリ言語を用いて 2^mの除算と $\bmod 2^m$の剰余算とをシフト演算にしたときの剰余側の最上位ビットになるので、その検知は容易かつ高速である。

2.1.4 現役塊の連鎖化

現役塊は格納領域中に散在するが、退役塊の先頭フィールドを利用してこれらを連鎖にすると、各現役塊は論理的に結合された状態になる。これを利用すると、ポインタ補正や圧縮の対象であるすべての現役セルの検出は格納領域量ではなく、現役セルの総容量に比例する時間で行うことができる。

2.1.5 時間計算量

表2は改良型圧縮法の時間計算量を示したものである。その導出過程は 文献[3]に詳細な記載があるので、その過程で用いた仮定のみを示す。

H-1:

現役セル中の非ポインタ比率(f)は定数である。

この仮定は現役セルの個数(あるいは容量)が十分大きいとき統計的に成り立つ。

H-2:

A_0 と他の現役塊との相互参照ポインタの総和は A_0 内の相互参照ポインタ数に比べて十分小さい。

この仮定はセルの共有や書換えの操作が少ない場合や 現役塊中に占めるA_0の比が十分大きい場合に成り立つ。

この両者の仮定は、A_0 を指すポインタ比率をA_0の容量比に置き換えることを可能にする。

2.1.6 処理時間

表2で特徴的なことは、ポインタ補正の時間計算量が M_Aでなく、M_A から M_{A0}の寄与分を減じた

表2　改良型圧縮法の時間計算量

マーキング	$\underline{O}(M_A)$
表作成	$\underline{O}(M)$
ポインタ補正	$\underline{O}(M_A-d \cdot M_{A0})$
圧縮	$\underline{O}(M_A)$

記号の意味

M ： セルの総容量（格納領域量）　d ： 定数

M_A ： 現役セルの総容量　M_{A0} ： A_0の容量

ものに比例することである。それはポインタ補正の処理時間(T_P)が (2.3)式で表されることを意味する。

$T_P = (a \cdot \underline{x} + b \cdot \underline{y}) \cdot M$

a, b : 定数

$\underline{x}$: 現役セルの占有率(M_A/M, $0 \leqq \underline{x} < 1$)

$\underline{y}$: A_0のの占有率(M_{A0}/M) (2.3)

T_Pが単なる $\underline{x}$の一次式でないことは表３の結果から明白である。すなわち、ポインタ補正の正規化値が $\underline{x}$の増加に伴って減少することである。表３は高機能ワークステーション Sun3/50上の完全遅延評価系[3]に 具現された改良型圧縮法の処理時間（実測値）とその正規化値である。２つの定数(a, b/a=d)は 多種のプログラムを異なる格納領域量で評価し、ポインタ補正に要する時間の「統計値」として算出することができる。ただし、比較のために A_0を指すポインタの(早期)検知がないように手直をしたくず集め（圧縮法）での実測値が必要である。こうして得られた改良型圧縮法の処理時間を表す式が (2.4)式である。

P-1 : $(2.17 + 20\underline{x} - 7.70\underline{y}) \cdot M$

P-2 : $(2.23 + 20\underline{x} - 7.23\underline{y}) \cdot M$ (2.4)

ただし、(2.4)式は$\underline{y}$の小さいところで正規化値からずれる。その理由は同一プログラムにおいても d値に違いがあるからである。特に $\underline{y}$が小さいときそれが顕著である。これは 2.1.5項で述べた２つの仮定(H1とH2)が普遍的でないことを示している。しかし $\underline{y}$が大きくなるにつれて多くのプログラムで d値はある値(0.71)の近傍となる。このことは $\underline{y}$が十分大きいとき２つの仮定が成立する可能性のあることを示している。

2.1.7 問題点

補正表を用いる圧縮法の問題点はその表に必要な領域量である。A_0 が存在しないか、その容量比が十分小さいとき、表の容量は(セルの)格納領域の $1/2^m$となる。m=3とすると、その比は 1/8であり、格納領域量が大きくなるとこの量は無視できなくなる。m を大きな値に選ぶことはポインタ補正の処理時間の増加を生む。

2.2 Morrisの圧縮法

Morrisは、セルの格納領域を行きと帰りの２回の走査でポインタ補正と圧縮とが同時に行えるアルゴリズムを考案しその正当性を証明した[1,4]。これが、Morrisの圧縮法と呼ばれているものである。この方法はアルゴリズムが簡潔であるという利点もあるが、具現上の問題点がいくつかある。すなわち、

(1) １個のポインタについて、マークビットのほかに追加ビットが１つ必要である。Morris自身が指摘しているように、処理速度を重視すると、このビットはセル内に確保してポインタと一緒に読み取らなければならない。

(2) 現役セルへのポインタに対して、ビット操作と「ポインタ交換」が必要である。また、共有セルの存在はその応分の「ポインタ交換」を必要とする。これは、処理時間が格納領域量以外にも比例項をもつことを意味する。

(3) データの内部表現がタグ付となる場合、ビット操作と「ポインタ交換」に関してアルゴリズムの手直しが必要である。

(4) マーキングの手続きで、マークビットをオンにしたセルの容量の累計を行う必要がある。

2.2.1 処理時間

Morrisの圧縮法を具現し、その処理時間を示したのが表３である。Morrisの圧縮法の処理時間を定量的に評価するために、その正規化値に古典的くず集めの処理時間を表す (2.5)式をあてはめるとマーキングの処理も含む(2.6)式が得られる。

$T_O = (c + a \cdot \underline{x}) \cdot M$ a, c : 定数 (2.5)

$T_M = (5.78 + 16.7\underline{x}) \cdot M$ (2.6)

(2.4)式と (2.6)式の Mに係る項を比較すると、

Morrisの圧縮法の特徴はその定数部分が大きいことである。これは格納領域を２回走査する作業の寄与分である。

2.2.2 処理の高速化

格納領域の２回目の走査は現役塊だけで十分である。これは (2.6)式の定数部分を半分にする効果をもつ。この技法は 2.1.4項で述べたものであり 文献[5]にもその記述がある。(現役塊の)連鎖を作る時間を無視すると、処理時間(推定値)は (2.7)式で与えられる。 この式を用いて算出した正規化値が表３の「新版」の記載値である。

$$T_M = (2.89 + 19.6\underline{x})\cdot M \qquad (2.7)$$

2.2.3 問題点

(2.4)式と(2.7)式を比較すると、前項の改良による Morris法 (以下 新Morris法と呼ぶ) は $\underline{x}$が小さいとき改良型圧縮法に比べて劣ることはない。処理時間比で約 1.2である。しかし、$\underline{x}$ が大きくなると、その比率は次第に増加し、$\underline{x}$〜0.4で 1.5を超える (図３参照) 。

3. 複写法

3.1 マーキングと複写

A-1は C言語で記述されたマーキングアルゴリズムである。そこで使用されている型と関数は次の意味をもつ。

cell_field	: セルの各成分
unmarkedp	: セルがマークされていないか
cell_size	: セルの容量 (成分の個数)
mark_field	: セルの該当フィールドのマークビットをオンにする(マークする)
cellp	: セルを指すポインタか
copiedp	: 複写されたセルか
allocate	: 領域の確保
push	: スタックへプッシュ (マクロ)
pop	: スタックからポップ (マクロ)

ただし、ポインタはセルの先頭フィールドのみを指し、遅くともそこではセルの容量が読み取れるものとする。

A-1. Fast marking algorithm

```
mark(p) cell_field *p;
{int i;
   push(-1);
M: if unmarked(p)
     for (i=0; i<cell_size(p); i++)
     `{mark_field(p+i);
       if cellp(p[i]) push(p[i]);};
   pop(p); if(p>=0) goto M;
};
```

A-2. Copy algorithm

```
cell_field *copy(p) cell_field *p;
{cell_field *q, *r; int i;
 if copiedp(p) return(p);
 if copiedp(*p) return(*p);
   else {q=(r=allocate(cell_size(p)));
         push(-1); goto C2;};
C1: if copiedp(*p) q[0]=*p;
      else
     {q=(q[0]=allocate(cell_size(p)));
C2    for (i=0; i<cell_size(p); i++)
        if cellp(q[i]=p[i]) push(q+i);
      p[0]=q;};
    pop(q); if (q>=0) {p=*q; goto C1;};
 return(r);
};
```

A-1は 再帰を除去した可変容量セルに対するアルゴリズムである。こうした再帰の除去は高速化のための公知の技法である。ポインタ値は非負数であり、最初にプッシュされる負数(-1)は空スタック(stack empty)に対する番兵である。

複写アルゴリズムは、マーキングアルゴリズムに似せて作ることができる。それは、両者ともセル (の各成分) を再帰的にたどることで目的が達成されるからである。前者の目的はセルの各フィールドをマークすることであり、後者の目的はセルの各成分を別の領域に複写し、その行き先番地(forwarding address)を元のセルに残すことである。A-2は、A-1に類似させた可変容量セル用の複写アルゴリズムである。A-1とA-2の構文要素が変換される対象計算機(Sun3/50)の命令語(MC68020)の実行時間を基に両者の (理論的な) 処理時間比が算定できる。その処理時間比(複写／圧縮)は１個のセルに対して(3.1)式で与えられる。

$$r = (74n_P + 64n_T + 90)/(62n_P + 52n_T + 24) \qquad (3.1)$$

n_P : セルを指すポインタの個数

n_T : それ以外の成分の個数

実際のマーキングや複写では、その対象である現役セルの個数（容量）は十分に大きいとしてよい。そこで、n_Pを $(1-f)\cdot M_A$に、n_Tを $f\cdot M_A$に置き換え、M_A》1 とすると、処理時間比は (3.2)式となる。f はポインタ比率である。

$$R = 1 + 12/(62-10f) \qquad (3.2)$$

$0 \leqq f \leqq 1$の条件から処理時間比(R)は $1.19 \leqq R \leqq 1.23$となる。これから、複写方式のくず集めの処理時間はマーキングの処理時間の約 1.2倍と推定できる。ただし、この比率は対象計算機を Sun3/50に規定した場合であり、対象計算機が異なればこの比率も変化する。ただし、Sun3/50 の命令語(体系)が特異でないことはそのクロック周期を他の計算機の対応する命令語実行時間と比較すれば明らかであろう。

3.2 処理時間

前節の結論と表３のマーキングの正規化値から、複写に必要な時間(T_c)は (3.3)式で与えられる。

$$T_c = 7.88\cdot\underline{x}\cdot M \qquad (7.88 = 1.2\times 6.57)$$

6.57はマーキングの正規化値の平均 (3.3)

圧縮法と複写法の両者の処理時間を比較する場合、セルの格納に用いる領域量（厳密に言うならば、回収により使用できた領域量）を同じにするのが原則である。Mを格納領域量とすると 圧縮法は Mがすべて使用し尽くされたときに引用される。複写法では M/2の半領域が交互に使用されるため、くず集めは M/2が使用し尽くされたときに引用される。そこで、前者が Mから $\underline{x}\cdot M$の現役セルを圧縮する作業は、後者が２回呼ばれ、もとの半領域に $\underline{x}\cdot M$の現役セルを複写する作業に相当する。ただし、使用できた領域量は (3.4)式に示すような違いがある。

総容量が $\underline{x}_1\cdot M$（前回）と$\underline{x}\cdot M$（今回）の現役セルを複写するのに必要な時間は (3.3)式の $\underline{x}$を単に$\underline{x}_1+\underline{x}$で置き換えることで得られる。しかし、次

圧縮法 : $(1-\underline{x}_0)\cdot M$

複写法 : $(1-\underline{x}_0-\underline{x}_1)\cdot M$

$\underline{x}_0$: 前回圧縮後、または前々回複写後の現役セル容量比

$\underline{x}_1$: 前回複写後の現役セル容量比 (3.4)

式で示されるように圧縮法と複写法では回収により使用できた領域量が異なる。同じ領域量で両者の処理時間を比較するためには (3.4)式に基づいた補正が必要である。その補正結果は次式で示される。

$$T_c = 7.88\cdot(\underline{x}_1+\underline{x})\cdot M\cdot(1-\underline{x}_0)/(1-\underline{x}_0-\underline{x}_1) \qquad (3.5)$$

(3.5)式は 複写法によるくず集めの処理時間を表す。式中の$\underline{x}_0$と$\underline{x}_1$は実行プログラムと Mとに依存して決る値である。しかし、くず集めの回数が多く、しかもその何回目であるかを特定しなければ、それらの関係を推定することができる。くず集めごとに現役セルが増える場合は、明らかに $0<\underline{x}_0<\underline{x}_1<\underline{x}<1/2$である。逆に、現役セルが減る場合は $\underline{x}_0>\underline{x}_1>\underline{x}$であるが、くず集めで常に現役セルが減少していくことは一般的なプログラムの実行ではまずありえない。現役セルの減少があれば、それに見合うかたちで増加があったと考えるべきである。こうした場合、くず集めの回数が多ければ $\underline{x}_0\sim\underline{x}_1\sim\underline{x}$となる。以上のことから、$T_c$に関する(3.6)式が得られる。

$$7.88\underline{x}\cdot M \leqq T_c \leqq 7.88\cdot(1+1/(1-2x))\cdot\underline{x}\cdot M \qquad (3.6)$$

なお、上限は $\underline{x}_0\leqq\underline{x}_1\leqq\underline{x}$の関係式、下限は $0\leqq\underline{x}_0\leqq\underline{x}_1$ の関係式から得られる。(3.6)式は複写法の処理時間(正規化値)を表す。図１にT_cの取り得る範囲が図示されている。それによれば $\underline{x}\geqq 0.27$で上限値が改良型圧縮法を超え、さらに $\underline{x}\geqq 0.31$で新Morris法を超える。くず集めが頻繁になれば、$\underline{x}_0\lesssim\underline{x}_1\lesssim\underline{x}$でかつ$\underline{x}$が大きくなる。こうした場合、$T_c$はその上限近くの値を取るので、複写法は改良

型圧縮法や新Morris法より処理時間が増大する。これが複写法の最大の問題点である。

4．複写・圧縮混合法

4.1 原理

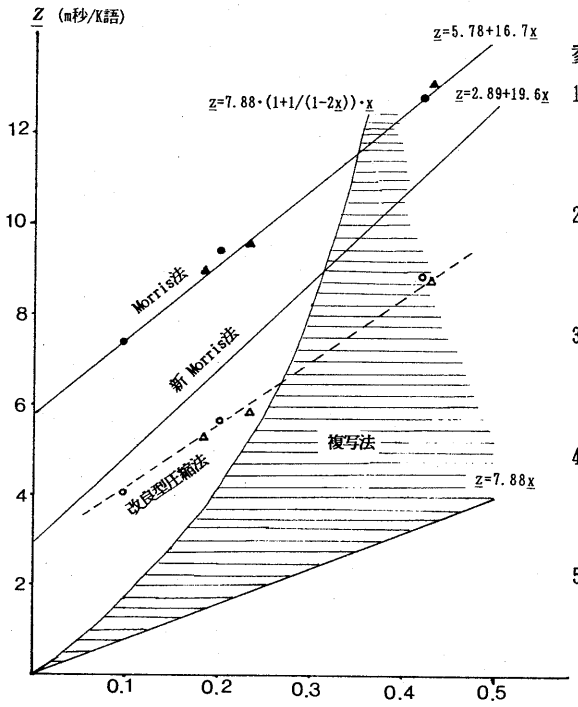
現役セルの占有率(x)が小さいとき 処理時間では複写法が、逆に xが大きいときは圧縮法が有利になる。そこで、くず集め法として両者を用意しておき xの大小で両者のいずれか(有利な方)を実行すれば、処理時間として最適になる。しかし、x の値はマーキング、あるいは複写それ自体が実行されない限り確定できない。マーキングを行なうことは圧縮法の処理を意味する。複写は複写法を意味する。当然、最初のくず集めでそのどちらを行なうべきかも明白な解答があるわけではない。このため、次のような手順を採用している。

(1) 最初は複写法を行なう。

(2) 前回のくず集めで算出した xが 0.3以下であれば今回は複写法を、そうでなければ(x>0.3)、圧縮法を行なう。

図1　処理時間

(注)　図中の点は実測値(正規化値)である。



4.2 具現法

4.2.1 記憶領域

セルの格納領域は半領域に２分割されるが、これは便宜的なもので、圧縮法によるときは全領域が使用対象となる。セルの領域配置上の局所性が保存される保証はない。補正表は "demand page" で実現される。その容量は(セルの)格納領域量の 1/10 以下と考えられる。

4.2.2 内部表現

データの内部表現では、マークビットが必要である。Morris法を採用しないため、他に追加ビットは不要である。

4.4.3 手続き

複写と圧縮の２つの手続き自体の共用部分はない。複写手続きは 3.1節で示した A-2のアルゴリズムに基づいている。圧縮は改良型圧縮法の技法を使用している。

参考文献

1. Cohen, J.: Garbage collection of linked data structures, ACM Computing Surveys, 13 (3), pp. 341-367 (1981).
2. Terashima, M. and Goto, E.: Genetic order and compactifying garbage collectors, Inf. Process. Lett., 7 (1), pp. 27-32 (1978).
3. 寺島 佐藤 : 可変容量セルの効率的なくず集めについて, 情報処理学会論文誌 30 (9), pp. 945-955 (1989).
4. Morris, F.L.: A time- and space-efficient garbage compaction algorithm, Comm. ACM, 21 (8), pp. 662-665 (1978).
5. 高橋 : 複合タグ方式のLisp処理系におけるガベージ・コレクタの実現とその問題点,情報処理学会論文誌 30 (3), pp. 339-346 (1989).

表３　処理時間

P-1. 7 Queen（8王妃問題の7王妃版）

格納領域量	印付け	表作成	補正	圧縮	Morris法	新版*	計
64 K語 x=0.098 y=0.0095 GC 18回	0.48 <6.22>	1.70 (2.16)	0.80 <10.4>	0.22 <2.85>	---- ----	---- ----	3.20 (4.07)
	0.42 <5.45>	---- ---- ---	---- ---- ----	---- ---- ----	5.36 (6.82) ----	---- ---- (4.14)	5.78 (7.35) (4.80)
32 K語 x=0.202 y=0.070 GC 28回	1.18 <6.38>	2.20 (2.18)	1.54 <8.33>	0.50 <2.70>	---- ----	---- ----	5.22 (5.69)
	1.28 <6.92>	---- ---- ----	---- ---- ----	---- ---- ----	7.08 (7.71) ----	---- ---- (5.46)	8.36 (9.41) (6.85)
16 K語 x=0.420 y=0.225 GC 83回	3.74 <6.54>	2.94 (2.16)	3.88 <6.79>	1.46 <2.45>	---- ----	---- ----	12.0 (8.84)
	3.74 <6.54>	---- ---- ----	---- ---- ----	---- ---- ----	13.66 (10.1) ----	---- ---- (8.25)	17.40 (12.8) (11.12)

P-2. リストの長さに基づくフィボナッチ数（fb(17)）計算

格納領域量	印付け	表作成	補正	圧縮	Morris法	新版*	計
64 K語 x=0.186 y=0.095 GC 12回	1.02 <6.98>	1.70 (2.16)	1.04 <7.10>	0.40 <2.73>	---- ----	---- ----	4.16 (5.29)
	0.98 <6.69>	---- ---- ----	---- ---- ----	---- ---- ----	6.08 (7.73) ----	---- ---- (5.27)	7.06 (8.98) (6.53)
48 K語 x=0.237 y=0.147 GC 17回	1.16 <5.86>	1.88 (2.25)	1.10 <5.56>	0.72 <3.64>	---- ----	---- ----	4.86 (5.82)
	1.30 <6.57>	---- ---- ----	---- ---- ----	---- ---- ----	6.70 (8.02) ----	---- ---- (5.92)	8.00 (9.57) (7.53)
32 K語 x=0.429 y=0.314 GC 36回	3.76 <7.44>	2.70 (2.29)	2.50 <4.95>	1.38 <2.73>	---- ----	---- ----	10.34 (8.77)
	3.70 <7.32>	---- ---- ----	---- ---- ----	---- ---- ----	11.72 (9.94) ----	---- ---- (8.36)	15.42 (13.1) (11.3)

注（1）数値は処理時間（CPU時間、単位は秒）を表す。ただし、
()内の数値は、処理時間÷(GCの回数×格納領域量)で、
<>内の数値は、処理時間÷(GCの回数×現役セル容量)でそれぞれ正規化された値（単位は m秒/K語）である。
なお、新版*(Morris)の正規化値は推定値である。
（2）xは現役セルの占有率、yはA$_0$の占有率を表す（$0 \leq y \leq x < 1$）。