13. 動的記憶領域割付け法†

佐々　政孝††

1. はじめに

コンピュータの利用が定形的処理にとどまらず，人工知能，自然言語処理などの応用に広がっていくにつれ，それを扱うためのプログラミング・システムも，実行時に動的に記憶領域の構造が変化するデータ構造を提供する必要が生じてくる．この解説では，その基礎となる**動的記憶領域**の割付け方法について述べる．

他の解説との重複を避けるため，ここでは主にヒープ（heap）方式と呼ぶ方式に焦点を当てる．また，ごみ集め法については他の解説に譲る．

全般的な参考文献として文献 1)〜3)，文献の孫引き用として文献 4) をあげる．

2. 動的記憶領域割付け法の諸方式

この章では，動的記憶領域割付けの諸方式について述べる．割付けの対象とするものは連続番地からなる一連の語であるが，それを**領域**（block, area）と呼ぶこととする．

動的記憶領域の割付け法は大きくスタック（LIFO）方式とヒープ方式に分類できる．

2.1 スタック（LIFO）方式

スタック（stack）とは，最後に割付けられたものが最初に不要となるような性質をもつものをいう．典型的な例は，再帰的呼出しを許すプログラミング言語の実行時環境（局所変数など）の割付けである．これについては，本誌 19「再帰呼出しの実現法」を参照して頂きたい．

2.2 ヒープ方式

動的記憶領域を用いた多くの応用では，切出された領域は，手続き呼出しや戻りに無関係に，必要とされる間は保持されなければならない．**ヒープ**（heap）方式はこのようなものを扱う．この方式では，多数の領域を1つの共通の記憶領域（全記憶空間とも呼ぶ）の中で管理する．プログラムの実行の任意の時点で領域が要求されると，記憶領域管理プログラムは指定した大きさの領域を**切出す**．領域の寸法については同一寸法の場合と多種寸法の場合とがある．記憶領域管理上，面倒なのは，主として多種寸法の場合であるので，その点を中心に3章で詳しく述べる．

明示的戻しと暗黙の戻し

切出された領域の使用が終ったとき，その領域は全記憶空間に返して構わない．これを**戻し**（**解放**，liberation）という．戻しのしかたを，暗黙の戻し（またはごみ集め）と明示的戻しとに分けることができる．本解説では，戻すべき領域がプログラム内で明示的に指示される明示的戻しを取扱う．ごみ集めについては本誌 15「シーケンシャル・ガーベジ・コレクション」を見られたい．

3. ヒープ方式による動的記憶領域割付け法

この章では，ヒープ方式による動的記憶領域割付け法について，さまざまな寸法の領域がある場合を中心に述べる．この章は内容の多くを文献 1), 2) によっている．

3.1 記憶領域の表現方式

使用中の領域を**占有**（reserved）**領域**，使用されていない領域を**空き**（free）**領域**と呼ぶ．それぞれの領域中には記憶管理のための欄が必要である．この欄には，占有／空きを示す目印や，寸法，他の領域へのリンク（つなぎ）などが必要に応じてしまわれる．

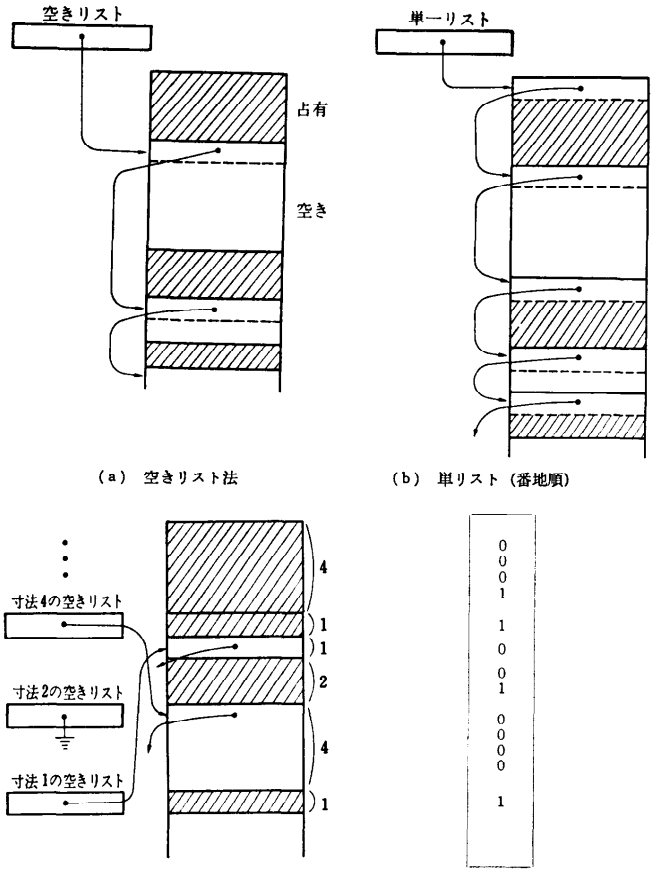
次に記憶領域全体に目を向けると，その中には占有領域と空き領域とが混在している．この状態を管理するために色々な方式が使われる．よく使われる方式を**図-1** に例を示しつつ述べる．

（1）**空きリスト**（または空き領域リスト，free list）**法**

これは空き領域をつないだリストを使う方法である．これには，すべての空き領域を一本のリストにつ

† Dynamic Storage Allocation by Masataka SASSA (Institute of Information Sciences and Electronics, University of Tsukuba).

†† 筑波大学電子情報工学系



(a) 空きリスト法　(b) 単リスト（番地順）

(c) Buddy（分身）法　(d) ビットテーブル法（(3)に対し Tail Lamp 法[10]を使った場合）

(注) 領域には図にある以外に，占有/空きを示す目印や寸法の欄などが必要に応じて置かれる．また領域の寸法は一般にはもっと大きい．

図-1 動的記憶領域の表現方式

ないだ**単一空きリスト法**と，領域の可能な寸法（この寸法に幅をもたせることもある）ごとに何本ものリストを使う**寸法別複数空きリスト法**とがある．

空きリストのつなぎ順にも色々な方式があり，空き領域の番地順，領域の戻しの順序／逆順序にあたるFIFO 順／LIFO 順，空き領域の寸法順，任意順などがありうる．

（2） **単一リスト法**

これは占有領域も空き領域もともにつないだ単一のリストを使う方法である．ふつうは，番地順の単一リストが使われる．

（3） **Buddy（分身）法**[5]

領域の寸法を2の冪乗である1, 2, 4, 8, 16, …に限ることによって，考案された面白い方式である（詳しくは**3.3**節で述べる）．

この方式の変形として，フィボナッチ Buddy 法と呼ばれるものも考えられている[6),7),3),2)]．これは領域の寸法として1, 2, 3, 5, 8, 13, … のフィボナッチ数列のものを用いるものである．更に一般化したものもあるが，それについては文献 8) を見られたい．

（4） **ビットテーブル法**

記憶領域の各語に対し1ビットのフラグを用意し，それを別の領域にまとめたものをビットテーブルと呼ぶ．この方式は上の（1）—（3）を補う形で使われることが多い[9),10)]．

このほかの分類法として，**分離記憶 (segregated storage) 法**をあげる人もいる[2)]．これにはいくつかのバリエーションがあるが，すでに述べた寸法別複数空きリスト法もこの分類に含めるほか，全記憶空間を小さいゾーンに分離し，各々のゾーンでそれぞれの割付け法を用いる方法などを含むとしている．

以下では，紙面も限られているので，上記の方式のうち，空きリスト法と Buddy 法に焦点をあて，領域の切り出し方と戻し方について述べる．

3.2 空きリスト法における領域の切出しと戻し

切出し

n語の領域が要求されたとする．この時は，空きリストをたどり，$m(\geqq n)$ 語の領域を見つけ，そこからn語を切出し，残り $(m-n)$ 語を空き領域として残せばよい．ちょうど $m=n$ なら，空き領域は残さない．

最初適合法と最良適合法

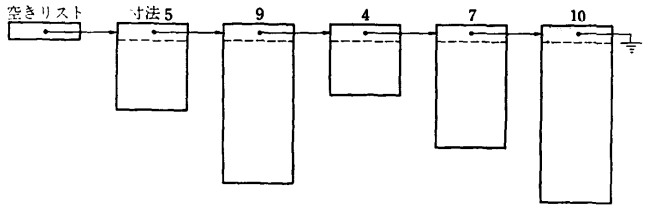
この $m(\geqq n)$ 語の領域の見つけ方であるが，これには次の2つの方法がある（**図-2**）．

（1） **最初適合法 (first-fit)**

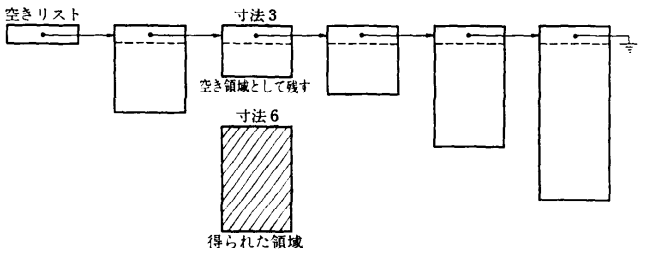
空きリストをたどり，nより大きい領域が見つかったら，直ちにそこから切出して返す方法である（**図-2**(b)）．

（2） **最良適合法 (best-fit)**

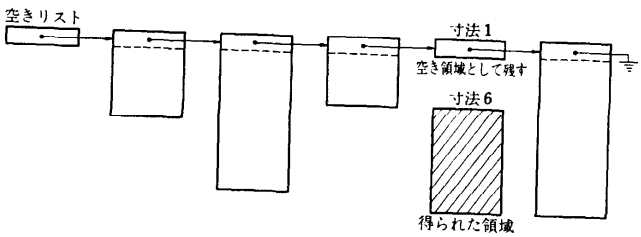
n以上の大きさの領域のうち最もnに近い大きさのものから切出して返す方法である（**図-2**(c)）．



(a)　空きリストの例



(b)　最初適合法（但し改良前）の適用例((a)で寸法6の領域が要求されたとする)



(c)　最良適合法（但し改良前）の適用例(同上)

図-2　空きリスト法の切出しにおける最初適合法と最良適合法

この2つの方式を比べると，一見，最良適合法が良さそうに思われるが，多くの研究の結果，今では最初適合法が好ましいとされている[2),1),4),11)].

その主な理由は，最良適合法では空きリストすべてを調べるために時間がかかりすぎることである．記憶効率については，Shore が詳しい解析を行っているが，それによると，要求される記憶領域の寸法の変動が大きいときは最初適合法がよい．しかし，2つの方式における時間記憶積で計算した記憶効率の差は，一般に 1～3% 以内にすぎないという[11)]．

切出し方の改良

空きリストをたどるときは，最後に切出しをした箇所からはじめるとよいことがある．これを**回転開始点法**とか**徘徊変数** (Roving Pointer) **法**と呼ぶ．これを最初適合法と組合わせたものを **Next-fit 法**[12)]と呼び，切出し時間が大幅に改善されるが，記憶効率はやや落ちる（比較は文献 12), 4), 2) を参照のこと）．

また，もう一つの改良として，小さい空き領域ができてしまうのを避けるため，n 語の領域が要求されたときには，寸法 $n+c$（c はおまけの限度）までは領域を分割しないでそのまま切出すのがよい．

戻し

ごく単純な戻し方は，指定された領域を空きリストにつなげるだけのものである．しかし，切出し時には領域が分割されることが多いのだから，このままでは全記憶空間が細切れになるばかりである．これを避けるために，戻し時には，空きリストにつなげるだけでなく，戻された領域の前後の領域を調べて，それが空き領域ならそれとの**統合**を行うようにするとよい．1つの方法を次に紹介する．

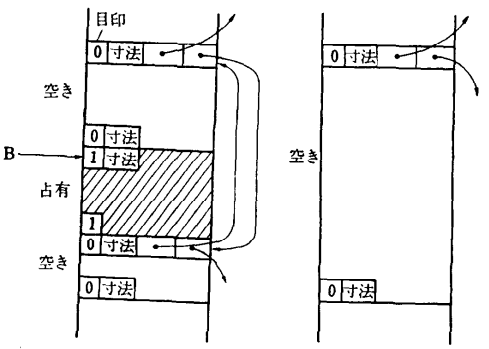
統合を伴う戻し（境界目印を用いる）**(図-3)**

占有領域，空き領域とも，最初と最後の語に占有か空きかを示す目印の欄を設けておく．こうすれば，戻すべき領域の直前と直後の番地の目印をみることで，それぞれ前後の領域が空き領域か否かがわかるので，可能ならば統合を行う．

また，空きリストは二重つなぎ(doubly-linked) リストとする．これは，前後の領域が統合されればそれも含めた空き領域を空きリストに戻す際，空きリストをたどる操作をなくすためである．

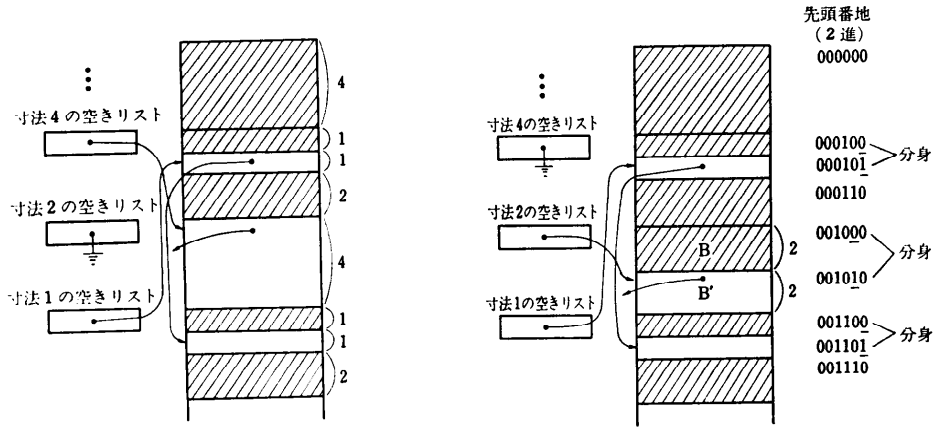
圧縮（詰め直し，compacting, compaction)

全記憶空間の細切れから回復するもう一つの方法として，**圧縮**と呼ばれる手法がある．これは占有領域を



(a)　戻す前　　(b)　Bの指す領域を戻した後

図-3　境界目印を用いる統合の例



(a)→(b) 切出しの例（寸法2の領域が要求され，領域Bが得られた）
(b)→(a) 戻しの例　（領域Bが戻され，B′ と統合された）
実際の領域はふつうもっと寸法が大きい．

図-4 Buddy 法における切出しと戻し，分身の関係．

全記憶空間の一方の側に集めて，空き領域を他方の側にひとまとめにすることである．この際は，占有領域で使われているポインタの値を更新する必要がある[2]．

圧縮には弱点もある．ポインタの更新ができるためには，記憶領域管理システムが，領域内のどの語がポインタであるかを認識できなければならないが，これはいつも可能とは限らない．更新操作もやや面倒である．

これらの点から，一般に圧縮は，領域が多種寸法のときや，領域が同一寸法でも仮想空間での局所性が問題となったり，リストの線型化（連続した番地に並べること）をしたいときなどに，ごみ集めと組にして行うのがふつうである[13),14)]．

3.3 Buddy 法における領域の切出しと戻し

Buddy(分身)法[5]は，領域の寸法を1, 2, 4, 8, 16, … と2の冪乗だけに限っている点が特徴である（図-4）．ある寸法の領域が要求されたとき，それがちょうど2の冪乗でない場合は，それより大きい2の冪乗の寸法の領域が要求されたとみなす．

全記憶空間の初期状態は一番大きな寸法の空き領域1個である．その後，寸法 2^k の領域が要求されたとき，ちょうどその大きさの空き領域がないときは，それより大きい空き領域を2分割する．このとき2分割された後のそれぞれの領域を **buddy**（分身）という．

一般に空き領域は寸法1, 2, 4, … ごとの空きリストにつなげておく．

切出し（2^k 語の領域が要求されたとする）

1. 寸法 2^k の空きリストから領域を取ってくる．それが空ならひとまわり大きい寸法 2^{k+1}，だめなら寸法 2^{k+2}, … の空きリストから領域を取ってくる．
2. その領域を空きリストからはずす．
3. その領域が寸法 2^k であればそれを返して終り．
4. そうでなければ，その領域を2分割し，使わない半分を当該寸法の空きリスト（それまでは空であったはず）に挿入し，残りの半分の領域をもって3へ戻る．

例を図-4（a）→（b）に示す．

なお，ここでは，各領域の占有／空きの情報や寸法情報を表わす方法については触れないが，1語につき1ビットのビットテーブルを使うだけで効率よい記憶管理をする Tail Lamp 法というものが知られている[10]．

分身のみつけ方

Buddy 法では自分の分身を簡単にみつけることができる．2^k 語の領域の（先頭）番地は 2^k の倍数である．2^k 語の領域の分身の番地は，自分の領域の番地を2進法で表わしたとき，(0, 1, 2, … と数えて）下から k ビット目（2^k を表わすビット）の0と1を反転したものである．たとえば図-4（b）のBとB′をみよ．

戻し（先頭番地を L，寸法を 2^k とする）

1. L が指す領域の分身の番地を求め，そこにある領域が空き領域でなかったり，空いていても寸法 2^k の領域でなかったならば，統合できないので3へ．
2. ［分身があるので統合する］分身を寸法 2^k の空きリストからはずす．$k \leftarrow k+1$ とし，自分と分身とを統合し，その番地をあらためて L として，1へ戻る．
3. L が指す領域を寸法 2^k の空きリストに戻す．

例を図-4（b）→（a）に示す．

4. 各方法の比較

この章では，テストケースを用いて記憶領域割付けの各方法を比較した Nielsen の結果を中心に述べる．

4.1 Nielsen による比較[4]

Nielsen は，シミュレーション・プログラムを対象とし，35 種の動的記憶領域割付け法について 18 のテストケースを用いて比較を行った．35 種というのは，記憶領域割付け法を大きく 6 種類に分け，それをさらに細かい戦略によって分けたものである．戦略としては，最初適合／最良適合，回転開始点か否か，領域内に制御情報をどのくらい置くか（寸法，後向きリンクなど），記憶の統合をいつ行うか（切出し時，戻し時，ごみ集め時）などがある．

解析は多岐にわたるので原論文を見て頂きたいが，Nielsen は次のような結論を引出している．領域の可能な寸法ごとに複数の空きリストを用いる方法が処理時間も記憶効率も良い．また，番地順単一リスト（占有領域も空き領域も 1 本のリストでつなぐもの）による単純な方法も良い．Buddy 法は処理時間は良いが記憶効率が悪い．フィボナッチ Buddy 法はこれを解決するかもしれない．空きリスト法は一般に他の方法より良くない．

以上が彼の結論であるが，これについては彼も述べているように，対象がシミュレーション・プログラムであるので，文字列処理やリスト処理など他の分野のものについては結果が異なってくるかもしれない．

4.2 Weinstock による比較[15]

もう一つの比較結果として，Weinstock のものがある．彼は 5 種類の典型的な記憶領域割付け法を 6 つのテストケースにより比較した．その結果，記憶効率の良いのは番地順空きリストを最良適合法で切出す方法であり，処理時間の速いのは Quick-fit 法と Buddy 法であったと述べている．ここで **Quick-fit 法**とは，よく使う寸法の領域のための寸法別空きリストを複数本と，残りのすべての寸法の空き領域をつなぐための空きリスト 1 本とを併用する方法である．

文献 2) には，この Weinstock の結果をはじめ，いくつかの比較結果が引用されている．

比較結果を参考にする際は，比較の対象とした記憶領域割付け法は何と何であるか，またテストケースはどのような性質のものであるか，に留意して，最適な方法を選ぶ必要があろう．

5. おわりに

動的記憶領域割付け法につき，主な方法を紹介した．アルゴリズムの詳細やこの解説で触れられなかった点については，参考文献を見て頂くようお願いする．

参考文献

1) Knuth, D. E.: *The Art of Computer Programming*, Vol. 1, *Fundamental Algorithms*, Addison-Wesley, 1973 の Section 2.5. (邦訳) 米田，筧(訳)：基本算法／情報構造，サイエンス社，1978 の 5 章.
2) Standish, T. A.: *Data Structure Techniques*, Addison-Wesley, 1980 の Chapter 6.
3) Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: *Data Structures and Algorithms*, Addison-Wesley, 1983 の Chapter 12.
4) Nielsen, N. R.: Dynamic Memory Allocation in Computer Simulation, Com. ACM, Vol. 20, No. 11, pp. 864-873 (Nov. 1977).
5) Knowlton, K. C.: A Fast Storage Allocator, Comm. ACM, Vol. 8, No. 10, pp. 623-625 (Oct. 1965).
6) Hirschberg, D. S.: A Class of Dynamic Memory Allocation Algorithms, Comm. ACM, Vol. 16, No. 10, pp. 615-618 (Oct. 1973).
7) Cranston, B. and Thomas, R.: A Simplified Recombination Scheme for the Fibonacci Buddy System, Comm. ACM, Vol. 18, No. 6, pp. 331-332 (June 1975).
8) Peterson, J. L. and Norman, T. A.: Buddy Systems, Comm. ACM, Vol. 20, No. 6, pp. 421-431 (June 1977).
9) Terashima, M. and Goto, E.: Genetic Order and Compactifying Garbage Collectors, Inf. Process. Lett., Vol. 7, No. 1, pp. 27-32 (Jan, 1978).
10) Isoda, S., Goto, E. and Kimura, I.: An Efficient Bit Table Technique for Dynamic Storage Allocation of 2^n-word Blocks, Comm, ACM, Vol. 14. No. 9, pp. 589-592 (Sep. 1971).
11) Shore, J. E.: On the External Storage Fragmentation Produced by First-Fit and Best-Fit Allocation Strategies, Comm. ACM, Vol. 18, No. 8, pp. 433-440 (Aug. 1975).
12) Bays, C.: A Comparison of Next-fit, First-fit, and Best-fit, Comm. ACM, Vol. 20, No. 3, pp. 191-192 (Mar. 1977).
13) 日比野靖：ガーベジコレクションとそのハードウェア，情報処理，Vol. 23, No. 8, pp. 730-741 (Aug. 1982).
14) Cohen, J.: Garbage Collection of Linked Data Structures, Comput. Surv., Vol. 13, No. 3, pp. 341-367 (Sep. 1981). (邦訳) 寺島元章(訳)：つなぎのあるデータ構造のくず集め，コンピュータ・サイエンス ('81), pp. 125-149, 共立出版 (Jan. 1983).
15) Weinstock, C. B.: Dynamic Storage Allocation Techniques, Ph. D. Thesis, Dept. Computer Science, Carnegie-Mellon Univ., Pittsburgh, Pa. (Apr. 1976).

(昭和 57 年 12 月 14 日受付)