解説

大規模データ処理のための簡潔データ構造

定兼 邦彦　九州大学大学院システム情報科学研究院

データ列に対して検索効率などを効率化するため，索引を付加することがある．演算を効率化するために，データに対して特定の情報を付加したものを，ここではデータ構造と呼ぶこととする．本稿ではこのようなデータ構造のうち，もとのデータの長さ n に対して $o(n)$ 程度の付加情報のみを与える，簡潔データ構造と呼ばれる分野について解説する．特に，最も基本的かつ応用範囲の広いビットベクトルに関する簡潔データ構造に焦点を当てる．ビットベクトル B に対して，先頭から i 番目までのビット中の 1 の数を与える $rank_1(B, i)$ と，i 番目の 1 の位置を与える $select_1(B, i)$ という演算は，基本的かつ重要な演算である．これらの演算が定数時間で可能な簡潔データ構造について，具体的なデータ構造とアルゴリズムを紹介し，次に付加するデータサイズの下界についての結果を示し，最後に今後の展望について述べる．

データ処理と簡潔データ構造

今日さまざまなデータが蓄積され，それを活用することが重要となっている．たとえば，商店での販売データ，Web ページ，ゲノム情報などのデータは大量であるため，それらの検索や統計情報の計算を効率よく行うことが必要である．この際に問題になるのは，処理速度とデータを格納する領域である．記憶領域を削減するにはデータの圧縮が必要になるが，通常はデータの処理速度は遅くなる．また，検索を行う場合に索引と呼ばれる情報を追加すると処理速度は向上するが，さらにデータ量が増加する．そのため，データ量を増やさず処理速度を早める手法が求められており，その 1 つが簡潔データ構造である．

■簡潔データ構造

本稿で扱うデータ構造は，索引付けデータ構造，または単に索引と呼ばれる．これは，データの検索(問合せ)を高速化するためにデータに付加する情報のことである．簡潔データ構造とは，索引付けデータ構造のうちで，そのサイズがデータのサイズと比較して十分小さいものである．正確には，データのサイズが L ビットのとき，索引のサイズが $o(L)$ ビットのものを簡潔データ構造と呼ぶ．また，サイズが小さいだけでなく，問合せも高速に行える必要がある．多くの簡潔データ構造は，基本的な問合せを定数時間で実現している．

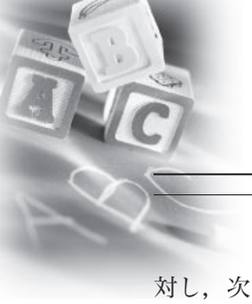
■計算モデル

計算モデルとしては，語長 w の word-RAM を用いる．このモデルでは，CPU は $[0, 2^w-1]$ の範囲の 2 つの整数 (w ビット整数) の論理演算 (AND, OR など) や四則演算が定数時間でできるとする．また，CPU はメモリの指定された番地の w ビット整数を定数時間で読み書きできるとする．なお，ここでは簡単のためにデータに対して番地を取り扱う演算が定数時間になるよう，データのサイズが L ビットの時 $w = \Theta(\log L)$ ビットと仮定する☆1．

ビットベクトルの簡潔データ構造

$B \in \{0, 1\}^n$ を長さ n のビットベクトルとする．各 $i \in \{1, 2, ..., n\}$ に対し，$B[i]$ は B の位置 i の値とし，$i, j \in \{1, 2, ..., n\}$ $(i \leq j)$ に対し，$B[i..j]$ を $B[i], B[i+1], ..., B[j]$ からなるビットベクトルとする．ビットベクトルに

☆1 ある定数 $c > 0$ に対して $w = c \log n$ となる．



対し，次の演算を定義する．

- $rank_0(B, i)$: $B[1..i]$ の中の 0 の数を返す
- $rank_1(B, i)$: $B[1..i]$ の中の 1 の数を返す
- $select_0(B, i)$: B の先頭から i 番目の 0 の位置を返す
- $select_1(B, i)$: B の先頭から i 番目の 1 の位置を返す

本稿では，任意に与えられた B に対して，$rank_0(B, i)$, $rank_1(B, i)$, $select_0(B, i)$, $select_1(B, i)$ を効率よく計算するためのデータ構造を構築する問題を考える．この，$rank$, $select$ とも何の補助情報なしで計算しようとすると，最悪では，B 中のビットを $\log n$ ビットずつすべて読まないといけないので，$O(n/\log n)$ 時間かかってしまう．一方，あらかじめすべての値を計算し，テーブルサーチを行うことを考えると，テーブルのサイズはいずれも高々 n なので，ポインタの長さは $O(\log n)$ となるため定数時間で行える．ただし，この場合，$rank$, $select$ とも値の最大値が n(表現は $\log n$ ビット) なので，$O(n \log n)$ ビットのテーブルが必要になってしまう．ビットベクトルに対する簡潔データ構造とは，付加する情報を $O(n)$ に抑えつつ，$rank$, $select$ とも高速(定数時間)で計算できるような付加情報である☆2．

■ビットベクトルの応用

ビットベクトルで効率的に $rank$ と $select$ を計算するデータ構造は，その他の複雑なデータ構造の基本となるため非常に重要である．たとえば，これらのデータ構造は木[5]，グラフ[2]，順列や関数，文字列の索引，区間和，多項式等を表現するデータ構造で用いられている．たとえば，木は括弧列で表現されるため（**図 -1** 参照），この括弧列の上での木の巡回はビットベクトルの $rank$ や $select$ 等で実現される．

ビットベクトルに対する $rank$ と $select$ は，その他の簡潔データ構造でも用いられる重要な問合せである．たとえば，あるデータを圧縮する場合，通常はそれをいくつかのブロックに分割し，各ブロックを独立に圧縮し，連結する．このようにして得られたビットベクトルを C とする．各ブロックは異なるビット数に圧縮されるため，ブロックへのポインタを格納する必要があるが，それを別のビットベクトル $B[1..|C|]$ で表現する．このベクトルは $B[i]$=1 のときに C の i 番目のビットがあるブロックの開始位置であることを表す．すると B での $select_1$ 問合せにより C 中のブロックを復元する場合にその開始位置を求めることができる．

☆2 $o(n)$ は $n \to \infty$ のときに n で割ると 0 になる関数である．

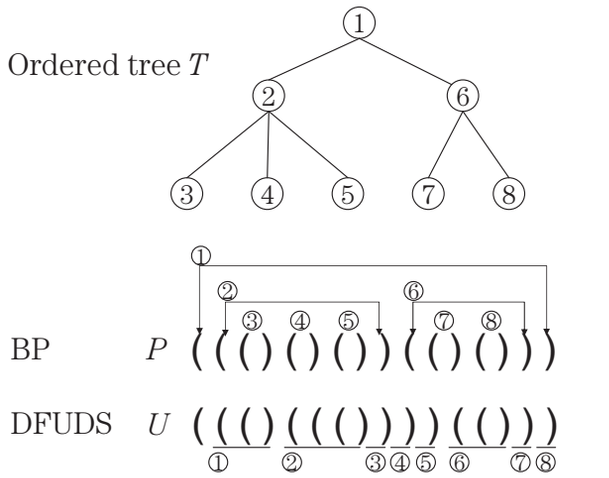


図 -1　順序木の BP 法と DFUDS 法による括弧列表現

ビットベクトルの索引サイズの上界

Jacobson[2] はビットベクトルに対して $rank$ を定数時間，$select$ を $O(\log n)$ 時間で求める索引付けデータ構造を提案した．索引に必要な領域は $O(n \log\log n/\log n)$ ビットである．その後，このデータ構造は改良され，$O(n \log\log n/\log n)$ ビットの索引で $select$ も定数時間で計算できるようになった．

本章では，$rank$ に対する Munro[4] の索引と，$select$ に対する Golynski[1] の索引を述べる．なお，簡単のため，全ビット長 n などは 2 のべき乗など分割に対してきりのよい値になっていると仮定し，細かい端数の議論は省略する．

■*rank* を計算する索引付けデータ構造

ここでは Munro[4] によるビットベクトルに対する簡潔データ構造を紹介する．これは 2 つの配列 R_l と R_s をあらかじめ計算する．長さ n のビットベクトル B に対して，$\ell = \log^2 n$ ごとに区切った部分を大ブロックと呼ぶ．一方，$s = 1/2 \log n$ に区切った部分を小ブロックという．ここで，境界はすべて一致し，各大ブロックには小ブロックが $2 \log n$ 個含まれるとする．このとき，$R_l[x]$ は最初から x ブロック目までの 1 の総数を入れる．一方，$R_s[y]$ は y 番目の小ブロックが属する大ブロックに対して，その大ブロック内での始めから y までの小ブロックまでの 1 の総数を入れる（**図 -2**）．$R_l[x]$ の値の最大値は n なので，R_l のサイズは $O(n/\log^2 n \log n) = O(n/\log n)$，一方，$R_s[y]$ の値の最大値は $\log^2 n$ なので，R_s のサイズは $O(2n/\log n \log(\log^2 n)) = O(n \log\log n/\log n)$ となる．

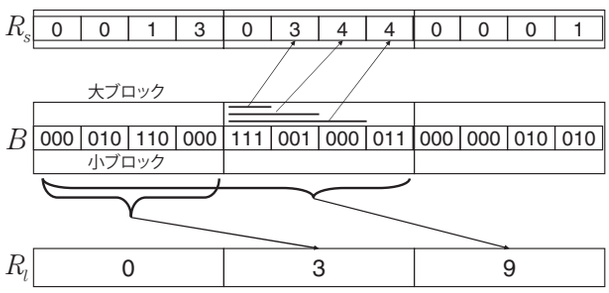


図 -2　ビットベクトルの索引の例．$\ell = 12, s = 3$ とする．

小ブロック中の位置

	1	2	3
000	0	0	0
001	0	0	1
010	0	1	1
011	0	1	2
100	1	1	1
101	1	1	2
110	1	2	2
111	1	2	3

$\frac{1}{2}\log n$ ビットの B の全パタン

$\sqrt{n}$ 個

図 -3　小ブロックでの *rank* を計算する表の例．
$s = \frac{1}{2}\log n = 3$ とする．

このようなビットベクトル B と配列 R_l, R_s に対して，B 中の先頭から i ビット目までの 1 の数 $rank_1(B, i)$ を定数時間で計算する方法を与える．これは，i ビット目の直前の小ブロックまでの 1 の数と，i ビット目が含まれる小ブロックの 1 の数を分けて計算する．

(1) まず i ビットがどの大ブロック，小ブロックに含まれるかを計算し，R_l, R_s の値を利用する．実際，i ビット目が含まれるブロックの直前は大ブロックに関しては $x = i/\log^2 n$，小ブロックに関しては $y = 2i/\log n$ となる (割算の結果は小数以下を切捨てにする)．すると，$R_l[x] + R_s[y]$ がここまでの 1 の総数になる．この計算は高々 $\log n$ ビットなので，word-RAM の仮定により定数時間で終了する．

(2) i ビット目の含まれる小ブロックをそのまま読み込む．これは高々 $1/2 \log n$ ビットなので word-RAM で定数時間で読み込める．一方，$1/2 \log n$ ビットのあらゆるビット列に対して，0 ビット目まで，1 ビット目まで，..., $1/2 \log n$ ビット目までのそれぞれに対して 1 の数の和をあらかじめ計算しておいた 2 次元の数表を用意しておく (**図 -3** 参照)．i ビット目のビットは読み込んだ小ブロックの中では $i-(2i/\log n)\cdot 1/2 \log n$ ビット目にあたる．そのため，小ブロックの値とこの i ビットの位置の計算値から，数表をアクセスすることで 1 の総数を得る．小ブロックの長さが $1/2 \log n$，ビットの位置を示す値も $\log(1/2 \log n)$ なので，キーの長さは $\log n$ 未満となる．したがって，仮定より定数時間で検索でき，表は $o(n)$ ビットで格納できる．

(3) (1)+(2) より $rank_1(B, i)$ を定数時間で計算できる．

一方，$rank_0(B, i) = i - rank_1(B, i)$ より $rank_0(B, i)$ も定数時間で計算できる．以上より次の定理を得る．

定理 1.　長さ n のビットベクトルが与えられたとき，$O(n)$ 時間の前処理により，$O(n \log\log n/\log n)$ ビットの補助データ構造を用いて $rank_1$ と $rank_0$ は定数時間で計算できる．

■*select* を計算する索引付けデータ構造

この節では Golynski[1] による *select* を定数時間で計算できる簡潔データ構造の概略を示す．$\ell = \log^2 n$ とし，B の中の 1 の位置のうち，$(i\ell)$ 番目 $(i = 1, 2, ..., \frac{n}{\ell})$ のものだけを格納する配列を作成する．この配列に格納されている連続する 2 つの位置の間の領域を上ブロックと呼ぶことにする．もし，ある上ブロックの長さが $\log^4 n$ 以上のとき，そのブロックは疎であるという．すべての疎な上ブロックに対し，その中のすべての 1 の位置をそのまま無圧縮で格納する．疎な上ブロックの個数は $\frac{n}{\log^4 n}$ 個以下であるため，1 の位置を格納するために必要なスペースは最大でも $\frac{n}{\log^4 n} \cdot \log^2 n \cdot \log n = \frac{n}{\log n}$ ビットである．もし，ある上ブロックが疎でなければ，それを中ブロックに分割する．各中ブロックには 1 が $\log n \log\log n$ 個存在する．ある中ブロックの長さが $(\log n \log\log n)^2$ より大きい場合は疎であるといい，その場合は各 1 の位置をそれぞれ $\log(\log^4 n)$ ビットを用いて格納する．疎ではない中ブロックは下ブロックに分割し，同様に 1 の位置を格納する．

このスキームのアイディアは以下の通りである．あるブロックが疎ならばその中の 1 の位置を格納するためのスペースは無視でき，$select_1$ の答は容易に求まる．もしブロックが疎でなければ，指定されたランクを持つ 1 の位置は表引きを用いて計算できる．どちらの場合でも，定数回のメモリアクセスで計算できる．

$select_0$ を計算する場合は，$select_1$ と同様の表を作成すればいい．以上より，次の定理を得る．

定理 2. 長さ n のビットベクトルが与えられたとき，$O(n)$ 時間の前処理の後，$O(n \log\log n/\log n)$ ビットの索引を用いて $select_1$ と $select_0$ は定数時間で計算できる．

ビットベクトルの索引サイズの下界

索引データ構造のサイズと，*rank*/*select* の計算時間の間にはトレードオフがある．つまり，データ構造のサイズが小さいと問合せに時間がかかるようになる．すべての答を格納しておく自明な索引では問合せ時間は定数だがサイズは $O(n \log n)$ ビットになる．一方，索引をまったく用いない場合にはビットベクトル全体を読み込む必要があり，問合せは遅くなる．索引サイズの下界に関する以下の結果がある．

定理 3. [3)]

1. 語長 w の word-RAM において，長さ n のベクトル B の上で *rank* を t 時間で計算するサイズ r ビットの索引データ構造について，以下の不等式が成立する．
   $2(2r + \log(w+1))tw \geq n \log(w+1)$.
2. 語長 w の word-RAM において，長さ n のベクトル B の上で *select* を t 時間で計算するサイズ r ビットの索引データ構造について，以下の不等式が成立する．
   $3(r+2)\,(tw+1) \geq n$.

特に，$t = O(1)$，$w = O(\log n)$ の場合，定理 3 は *rank* を定数時間で計算する索引のサイズの下界 $r = \Omega(n \log\log n/\log n)$ と *select* を定数時間で計算する索引のサイズの下界 $r = \Omega(n/\log n)$ を与える．

また，Golynski により，これらよりも強い下界も示されている．

定理 4. [1)] ベクトル B の上で *rank* または *select* を計算するアルゴリズムが r ビットの索引の任意の個所を読むことができ，任意の量の計算ができるとした場合でも，B のビットを参照できる個所が $O(\log n)$ 個所に制限されていると $r = \Omega(\frac{n \log\log n}{\log n})$ が成立する．

つまり定理 1 と定理 2 の上界は漸近的に最適である．なお，定理 4 は非常に強力な結果であり，問合せの計算量に制限はなく，B を読み込むビットの位置は連続でなくても成り立つ．

Golynski[1)] は以下の定理も示した．

定理 5. [1)] ベクトル B においてちょうど m 個所だけ 1 があるとする．もし B での *rank* か *select* を計算するアルゴリズムで B の高々 t 個所のビットを読み，サイズ r ビットの索引を任意の量だけ読み，任意の量の計算ができるものが存在するなら，$r = \Omega(\frac{m}{t} \cdot \log t)$ である．

まとめと今後の課題

本稿では，ビットベクトルの簡潔データ構造を解説した．このデータ構造はその他の簡潔データ構造でも用いられる基本的かつ重要なものであり，漸近的にはサイズ・問合せ時間ともに最適なものが提案されている．今後の課題としては，多値ベクトルに対する索引，B が変化しても問合せが高速に計算できるような動的データ構造の開発，データ構造のサイズの下界の証明☆3，実際的な実装などがある．これらについては初歩的なものしかなく，改良の余地がある．ビットベクトルに対する索引データ構造のサイズは漸近的には最適であるが，実際のデータに対する索引のサイズは大きすぎ，無視できない．よって，実際的な応用のためにはオーダー表記に隠れた定数ファクタの小さいデータ構造の開発が重要である．

参考文献

1) Golynski, A. : Optimal Lower Bounds for Rank and Select Indexes, In Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006), Volume 4051 of Lecture Notes in Computer Science, pp.370-381, Springer-Verlag (2006).
2) Jacobson, G. : Space-efficient Static Trees and Graphs, In Proceedings of the 30th IEEE Symposium on Foundations of Computer Science (FOCS 1989), pp.549-554 (1989).
3) Miltersen, P. B. : Lower Bounds on the Size of Selection and Rank Indexes, In Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005), pp.11-12 (2005).
4) Munro, J. I. : Tables, In Proceedings of the 16th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1996), Volume 1180 of Lecture Notes in Computer Science, pp.37-42, Springer-Verlag (1996).
5) Munro, J. I. and Raman, V. : Succinct Representation of Balanced Parentheses and Static Trees, SIAM Journal on Computing, 31(3), pp.762-776 (2001).

(平成 19 年 6 月 11 日受付)

☆3 定理 3 から 5 での下界はビットベクトルが n ビットを用いてそのまま格納されている場合のみに成立するもので，B が圧縮されている場合には成り立たない．

定兼　邦彦（正会員）
sada@csce.kyushu-u.ac.jp

2000 年東京大学大学院理学系研究科情報科学専攻博士課程修了．同年より東北大学大学院情報科学研究科助手．2003 年より九州大学大学院システム情報科学研究院助教授．情報検索のアルゴリズムとデータ構造，データ圧縮の研究に従事．