数理解析研究所講究録
第1649巻 2009年 173-180

置換のランク付けに対する$O(n\log\log n)$ビット領域の線形時間アルゴリズム

須藤 郁弥† 篠原 歩‡
†東北大学工学部 電気情報・物理工学科 ‡東北大学大学院 情報科学研究科

n個の要素を持つ集合S上の置換πを一意な整数$R(\pi,S)\in\{0,1,...,n!-1\}$に対応させる関数を置換のランク付け関数と呼ぶ．この関数を用いて，与えられた置換に対して対応する整数値を返す操作を置換のランク付けと呼ぶ．本稿では群全体における置換の辞書順をランクとして用いる．既存の結果として，置換の辞書順ランク付けおよび逆問題に対しては$O(n\log n)$ビット領域を用いた線形時間アルゴリズムが知られている．本稿では，既存の手法に対し領域計算量に関して優位な$O(n\log\log n)$ビット領域を用いた線形時間アルゴリズムを提案する．

1 はじめに

n個の要素を持つ集合Sに対し，SからSへの全単射をn次の**置換**と呼ぶ．本稿では簡単のために集合$X_n=\{0,1,...,n-1\}$上の置換を考える．

置換によってインデックス付けされた配列は多くのアプリケーションで用いられる．例としては，15パズルやルービックキューブなどの組み合わせパズルにおける状態空間の探索が挙げられる．このような配列の実現には，置換πを一意な整数$R(\pi,S)\in\{0,1,...,n!-1\}$に移す全単射のランク付け関数や，その逆関数が用いられることが多い．

ランク付け関数の実現には，初めに置換の順序を定義する必要がある．多くの場合は置換のランク付けには辞書順を用い，集合S上の置換πのランクは，S上の置換のうちπよりも辞書順が前のものの個数とする．

ランク付け関数の素朴な実装の計算時間は$O(n^2)$であることが知られている[1, 2]．また，[4]では，剰余演算やマージソートを利用した$O(n\log n)$時間の実装が紹介されている．さらに，Dietzのデータ構造[3]を用いることで，計算時間は$O(n\log n/\log\log n)$に削減できる．

置換のランク付け問題に対する最初の線形時間アルゴリズムは，辞書順でない順序をランク付けに用いて，Myrvoldらによって示された[5]．Myrvoldらは，最初にランク付けの方法を定義する従来のアプローチとは異なり，ランクから置換を生成する線形時間アルゴリズムにより置換のランクを定義し，その逆操作に対する線形時間アルゴリズムを構築することで，置換のランク付けに対する線形時間アルゴリズム実現した．Myrvoldらのアルゴリズムでは，配列上の要素の$O(n)$回の入れ替えを用いてランク付けを行う．

辞書順を用いた置換のランク付けに対する線形時間アルゴリズムは，Maresらによって示された[6]．Maresらのアルゴリズムは，$n(\lceil\log n\rceil+1)=O(n\log n)$ビットのビットベクタを用いて処理を行い，$O(n\log n)$ビットの整数に対する四則演算およびビット演算が定数時間で出来ると仮定できる場合に線形時間で実行できる．

本稿では，辞書順を用いた時間のランク付けに対し，既存の手法より領域計算量を改善した線形時間アルゴリズムを提案する．提案手法では，Maresらのアルゴリズムと同様にビットベクタを用いて処理を行うが，線形時間で実行するには，$O(n)$ビットの整数に対する四則演算およびビット演算が定数時間で実行できれば十分であり，領域計算量も$O(n\log\log n)$ビット領域に改善している．

2 置換のランク付け

$S\subseteq X_n=\{0,...,n-1\}$上の置換の集合を$P_S$と表す．また，置換$\pi$の$i$番目の要素を$\pi[i]$と表し，$\pi[i,...,j]\equiv(\pi[i],...,\pi[j])$とする．集合$S$上の置換$\pi$，$\pi'$に対し，$\pi$が$\pi'$よりも辞書順で前であることを$\pi<\pi'$と表わし，$S$の要素数$m$を用いて次のように定義する．

$$\pi<\pi'\Leftrightarrow_{def}\pi[0]<\pi'[0]\vee(\pi[0]=\pi'[0]\wedge(\pi[1,...,m-1])<(\pi'[1,...,m-1]))$$

集合S上の置換πの**辞書順ランク**を$R(\pi,S)$と表し，次のように定義する．

$$R(\pi,S)\Leftrightarrow_{def}|\{\pi'\in P_S\mid\pi'<\pi\}|$$

すなわち$R(\pi,S)$は，S上の置換の集合P_Sのうちπより辞書順で前に来るものの個数を表す．例えば，$n=3$としたとき，$X_3=\{0,1,2\}$上の置換の辞書順

入力 :集合X_n上の置換π
出力 :置換πの辞書順ランク

$rank \leftarrow 0$
$S \leftarrow X_n$
for$(i = 0; i < n; i++)\{$
  $rank \leftarrow rank + r(\pi[i], S) \cdot (n-i-1)!$
  $S \leftarrow S \backslash \{\pi[i]\}$
$\}$
return $rank$

図 1: 置換のランク付けアルゴリズム

入力 :集合X_n上の置換πのランク
出力 :置換π

$S \leftarrow X_n$;
for$(i = 0; i < n; i++)\{$
  $\pi[i] \leftarrow r^{-1}(\lfloor rank/(n-i-1)! \rfloor, S)$
  $rank \leftarrow rank \bmod (n-i-1)!$
  $S \leftarrow S \backslash \{\pi[i]\}$
$\}$
return π

図 2: 置換のランク付けの逆操作

ランクは以下の通りである.

$$R((0,1,2), X_3) = 0 \quad R((0,2,1), X_3) = 1$$
$$R((1,0,2), X_3) = 2 \quad R((1,2,0), X_3) = 3$$
$$R((2,0,1), X_3) = 4 \quad R((2,1,0), X_3) = 5$$

また, 集合 $S \subseteq X_n$ に対する, 要素 $x \in X_n$ の**ランク** $r(x,S)$ を次式で定義する.

$$r(x,S) \Leftrightarrow_{def} |\{y \in S \mid y < x\}|$$

$x \in S$のとき, $r(x,S)$ は x が集合 S の要素のうち昇順で何番目の要素であるかを示す. このとき, $0 \le i < |S|$ に対し, Sの要素のうち昇順で i 番目の要素を $r^{-1}(i,S)$ と表す. すなわち, $r^{-1}(i,S)$ は次式を満たす.

$$r^{-1}(i,S) \in S \text{ かつ } r(r^{-1}(i,S),S) = i$$

$S \subseteq X_n$ 上の置換 π_S を考える. S の要素数を m とすると, 集合 $S' = S \backslash \{\pi_S[0]\}$ 上の置換は $(m-1)!$ 種類存在するので, $r(x,S)$ を用いると, π_S の辞書順ランクに対して次が成り立つ.

$$R(\pi_S, S) = r(\pi_S[0], S) \cdot (m-1)! + R(\pi_S[1,...,m-1], S') \quad (1)$$

式 (1) は, m 次の置換のランク付けから $(m-1)$ 次の置換のランク付けへの還元である. これより, 図1,2に示される π の辞書順ランク付けおよびその逆操作のアルゴリズムが導かれる.

図1,2のアルゴリズムの計算時間は, S を扱うデータ構造によって変化する. S に関する操作 $r(\pi[i],S)$, $r^{-1}(i,S)$, $S \backslash \{\pi[i]\}$ の計算時間が最大 $t(n)$ であるとすれば, 両アルゴリズムの計算時間は $O(n \cdot t(n))$ となる.

置換のランク付けを線形時間で行うには, $t(n) = O(1)$, すなわち $r(\pi[i],S)$, $r^{-1}(i,S)$, $S \backslash \{\pi[i]\}$ をそれぞれ定数時間で実行する必要がある. 提案手法では, ビットベクタを用いて3種類の操作に相当する操作を定数時間で実現している.

3 提案手法

本章では, $O(n)$ ビットの整数に対する四則演算およびビット演算が定数時間で出来ることを仮定した上で, 提案する置換の辞書順ランク付けおよびその逆操作に対する $O(n \log\log n)$ ビット領域の線形時間アルゴリズムについて述べる.

3.1 置換のランク付け

置換のランク付けを線形時間で行うには, 集合 $S \subseteq X_n$ の要素 x に対し, $r(x,S)$ および $S \backslash \{x\}$ の計算が定数時間で出来れば良い. その実現のため, はじめに提案手法で用いる $r(x,S)$ の2つの性質について述べる.

補題 1. 集合 $S \subseteq X_n$ の任意の要素 x に対し, 次式が成り立つ.

$$r(x,S) = x - r(x, X_n \backslash S)$$

証明. $r(x,S)+r(x,X_n\backslash S)=x$ となることを示す.

$$\begin{aligned} & r(x,S)+r(x,X_n\backslash S) \\ &= |\{y\in S \mid y<x\}|+|\{y\in X_n\backslash S \mid y<x\}| \\ &= |\{y\in X_n \mid y<x\}| \\ &= x \end{aligned}$$

□

補題 2. 空でない集合 $S\subseteq X_n$ の要素 s に対し，集合 $S'=S\backslash\{s\}$ を考える．このとき，$x\in X_n$，$Y\subseteq X_n$ とすると次式が成り立つ.

$$r(x,Y\backslash S')=\begin{cases} r(x,Y\backslash S)+1 & (x>s \text{ かつ } s\in Y) \\ r(x,Y\backslash S) & (\text{その他}) \end{cases}$$

証明. $s\notin Y$ のとき $Y\backslash S=Y\backslash S'$ だから

$$r(x,Y\backslash S')=r(x,s\backslash S)$$

$s\in Y$ のとき

$$Y\backslash S'=Y\backslash(S\backslash\{s\})=(Y\backslash S)\vee\{s\}$$

$(Y\backslash S)\wedge\{s\}=\phi$ だから

$$\begin{aligned} & r(x,Y\backslash S') \\ &= r(x,(Y\backslash S)\vee\{s\}) \\ &= |\{z\in(Y\backslash S)\vee\{s\} \mid z<x\}| \\ &= |\{z\in(Y\backslash S) \mid z<x\}|+|\{z\in\{s\} \mid z<x\}| \\ &= r(x,Y\backslash S)+r(x,\{s\}) \end{aligned}$$

したがって,

$$r(x,Y\backslash S')=\begin{cases} r(x,Y\backslash S)+1 & (x>s \text{ かつ } s\in Y) \\ r(x,Y\backslash S) & (\text{その他}) \end{cases}$$

□

以上の性質を用いて，置換の辞書順ランク付けを行う．まず，集合 $S\subseteq X_n$ および要素 $x\in X_n$ に対し，$a[x]=r(x,X_n\backslash S)$ となる配列 a を考えると，補題 1 より配列 a の値を用いて $r(x,S)$ を計算できる．また，補題 2 において $Y=X_n$ とすると置換の要素 $\pi[i]$ を読み込んだとき，次の反復における $a[x]=r(x,X_n\backslash S')$ を現在の $a[x]$ を用いて計算できる．処理の開始時点では $S=X_n$ であるから，任意の $x\in Xn$ に対して $a[x]=r(x,X_n\backslash S)=r(x,\phi)=0$ が成り立つ．以上から，図 3 のアルゴリズムが導かれる．

入力 :集合X_n上の置換π
出力 :置換πの辞書順ランク

```
rank ← 0
for(i = 0; i < n; i++) a[i] ← 0
for(i = 0; i < n; i++){
    rank ← rank + (π[i] − a[π[i]]) · (n − i − 1)!
    for(j = π[i] + 1; j < n; j++)
        a[j] ← a[j] + 1
}
return rank
```

図 3: 補題 1,2 を用いた置換のランク付け

ここで，$O(n)$ ビットの整数に対する四則演算およびビット演算を定数時間でできると仮定すると，ビットベクタを用いることで，図 3 のアルゴリズムは線形時間で実行できる．以下ではその手法について述べる．

提案手法では，集合 X_n の要素に関する情報を要素 $\lceil\log n\rceil$ 個ずつに分けて管理する．$0\le j\le\lfloor\frac{n-1}{\lceil\log n\rceil}\rfloor$ に対し，集合 $Y_{(n,j)}$ を，$Y_{(n,j)}\equiv\{x\in X_n \mid j\lceil\log n\rceil\le x<(j+1)\lceil\log n\rceil\}$ と定義する．さらに，集合 $Y_{(n,j)}$ に対しビットベクタ $\mathbf{v}_j$ を考え，$0\le k<\lceil\log n\rceil$ に対し，下位から k 番目のフィールド $\mathbf{v}_j[k]$ には $r(j\lceil\log n\rceil+k,Y_{(n,j)}\backslash S)$ の値を格納する．定義より各フィールドの値は常に $\lceil\log n\rceil$ 未満であるので，ビットベクタ $\mathbf{v}_j$ は $\lceil\log\lceil\log n\rceil\rceil$ ビットのフィールド $\lceil\log n\rceil$ 個からなる．また，$\lceil\log n\rceil$ ビットのフィールド $\lfloor\frac{n}{\lceil\log n\rceil}\rfloor$ 個からなるビットベクタ $\mathbf{s}$ を考え，$0\le j<\lfloor\frac{n-1}{\lceil\log n\rceil}\rfloor$ に対し，下位から j 番目のフィールド $\mathbf{s}[j]$ には $|Y_{(n,j)}\backslash S|$ の値を格納する．このとき，次の補題が成り立つ.

補題 3. 集合 $S\subseteq X_n$ の要素 x に対し，$r(x,X_n\backslash S)$ は，ビットベクタ $\mathbf{v}_j$，$\mathbf{s}$ を用いて次式により求めることができる.

$$r(x,X_n\backslash S)=\mathbf{v}_t[x-t\lceil\log n\rceil]+\sum_{j=0}^{t-1}\mathbf{s}[j],$$

$$\text{ここに, } t=\left\lfloor\frac{x}{\lceil\log n\rceil}\right\rfloor$$

証明. $t\lceil\log n\rceil\le x<(t+1)\lceil\log n\rceil$ より，$x\in Y_{(n,t)}$

である．よって，$r(x,S)$ および $Y_{(n,j)}$ の定義から，

$$\begin{aligned} r(x,X_n\backslash S) &= \sum_{j=0}^{t} r(x,Y_{(n,j)}\backslash S) \\ &= r(x,Y_{(n,t)}\backslash S) + \sum_{j=0}^{t-1} r(x,Y_{(n,j)}\backslash S) \end{aligned}$$

ここで，

$$\mathbf{v}_t[x - t\lceil \log n \rceil] = r(x,Y_{(n,t)}\backslash S)$$

また，$t' < t$, $x' \in Y_{(n,t')}$ とすると，$x' < (t'+1)\lceil \log n \rceil \leq t\lceil \log n \rceil \leq x$ だから，

$$\sum_{j=0}^{t-1} \mathbf{s}[j] = \sum_{j=0}^{t-1} |Y_{(n,j)}\backslash S| = \sum_{j=0}^{t-1} r(x,Y_{(n,j)}\backslash S)$$

以上から，$r(x,X_n\backslash S)$ の式が得られる． □

また，フィールドサイズ b のビットベクタ $\mathbf{z}$ に対し，$\sum_i \mathbf{z}[i] < 2^b - 1$ であるとき，次式が成り立つ [6].

$$\begin{aligned} \sum_i \mathbf{z}[i] &= \sum_i \mathbf{z}[i]2^{bi} \mod (2^b - 1) \\ &= \mathbf{z} \mod (2^b - 1) \end{aligned} \quad (2)$$

補題3より，ビットベクタ $\mathbf{v}_j$, $\mathbf{s}$ を用いて $r(x,X_n\backslash S)$ を計算でき，ビットベクタ $\mathbf{s}$ に対し，$\sum_i \mathbf{s}[i] \leq n-1 < 2^{\lceil \log n \rceil} - 1$ が成り立つので，式 (2) を適用することで，計算は定数時間で実行できる．

$S = X_n$ のとき，ビットベクタ $\mathbf{v}_j$, $\mathbf{s}$ の全てのフィールドの値は 0 である．また，補題 2 において，$Y = Y_{(n,j)}$ とすると，ランク付け操作において置換の要素 $\pi[i]$ を読み込んだとき，次の反復における $\mathbf{v}_j[k]$ を現在の $\mathbf{v}_j[k]$ を用いて計算でき，$S' = S\backslash\{\pi[i]\}$ とすると，$\pi[i] \in Y_{(n,j)}$ となる j に限り $\mathbf{s}[j]$ の値が 1 増加する．さらに，各ビットベクタの更新は，値の増加するフィールドの値を 1 とし，それ以外のフィールドの値を 0 としたビットベクタを足し合わせることで定数時間で可能となる．

例えば，$n = 8$ のとき $\lceil \log n \rceil = 3$ であるから，$Y_{(n,0)} = \{0,1,2\}$, $Y_{(n,1)} = \{3,4,5\}$, $Y_{(n,2)} = \{6,7\}$ となる．このとき，$S = \{1,3,4,5,7\}$ とすると，ビットベクタ $\mathbf{s}$, $\mathbf{v}_j$ は図 4 の通りになる．いま，置換の要素 $\pi[i] = 3$ を読み込んだとする．$\pi[i] \in Y_{(n,1)}$ だから，$r(3,S)$ は補題 1，補題 3 より次のように求められる．

$$\begin{aligned} r(3,S) &= 3 - r(3,X_n\backslash S) \\ &= 3 - (\mathbf{v}_1[0] + \mathbf{s}[0]) \\ &= 1 \end{aligned}$$

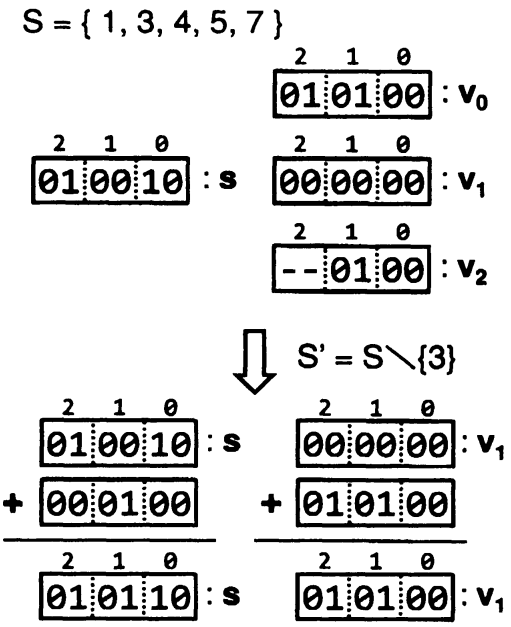

図 4: ランク付け操作における $n = 8$, $S = \{1,3,4,5,7\}$, $S' = S\backslash\{3\}$ のときのビットベクタ

また，ビットベクタ $\mathbf{s}$, $\mathbf{v}_1$ の更新は，補題 2 よりフィールドの値の変化を求め，図 4 に示すビットベクタを足し合わせれば良い．

以上から，次の定理が成り立つ．

定理 1. X_n 上の置換 π が与えられたとき，その辞書順ランクは $O(n \log\log n)$ ビット領域を用いて線形時間で計算できる．

アルゴリズムは図 5 の通りである．ここで，図 5 において，b はビットベクタ $\mathbf{s}$ のフィールドサイズ，b' はビットベクタ $\mathbf{v}_j$ のフィールドサイズ，**inc** は $\mathbf{v}_j$ の更新に用いるビットベクタである．領域計算量に関しては 3.3 節に述べる．

3.2 ランク付けの逆操作

ランク付けの逆操作，すなわち与えられたランクに対応する置換の算出を線形時間で行うには，図 2 において，集合 $S \subseteq X_n$ の要素 x および自然数 $i < |S|$ に対し，$r^{-1}(i,S)$ および $S\backslash\{x\}$ の計算が定数時間で出来れば良い．線形時間アルゴリズムでは，ランク付けと同様に $O(n)$ ビットの整数に対するビット演算および四則演算が定数時間で出来ることを仮定し，ビットベクタを用いて計算を行う．

入力 :集合X_n上の置換π
出力 :置換πの辞書順ランク

$rank \leftarrow 0$
$\mathbf{s} \leftarrow 0$
for$(j=0; j \leq \lfloor \frac{n-1}{\lceil \log n \rceil} \rfloor; j++)$ $\mathbf{v}_j \leftarrow 0$
$b \leftarrow \lceil \log n \rceil$
$b' \leftarrow \lceil \log \lceil \log n \rceil \rceil$
$\mathbf{inc} \leftarrow \sum_{i=0}^{\lceil \log n \rceil - 1} 2^{b'i}$
for$(i=0; i<n; i++)\{$
  $t \leftarrow \lfloor \pi[i] / \lceil \log n \rceil \rfloor$
  $r \leftarrow \mathbf{v}_t[\pi[i] - t\lceil \log n \rceil] + (\mathbf{s} \& 2^{bt}) \bmod (2^b - 1)$
  $rank \leftarrow rank + r \cdot (n-i-1)!$
  $\mathbf{v}_t \leftarrow \mathbf{v}_t + \mathbf{inc} \& \sim (2^{b'(\pi[i] - t\lceil \log n \rceil + 1)} - 1)$
  $\mathbf{s} \leftarrow \mathbf{s} + 2^{bt}$
$\}$
return *rank*

図 5: 置換のランク付けに対する提案手法

ランク付けの逆操作に対する提案手法では，ランク付けと同様に集合 X_n の要素に関する情報を要素 $\lceil \log n \rceil$ 個ずつに分けて管理する．集合 $Y_{(n,j)}$ に対しビットベクタ $\mathbf{v}_j$ を考え，$0 \leq k < \lceil \log n \rceil$ に対し，下位から k 番目のフィールド $\mathbf{v}_j[k]$ には $r^{-1}(k, Y_{(n,j)} \wedge S) - j\lceil \log n \rceil - k$ の値を格納する．定義より各フィールドの値は常に $\lceil \log n \rceil$ 未満であるので，ビットベクタ $\mathbf{v}_j$ は $\lceil \log \lceil \log n \rceil \rceil$ ビットのフィールド $\lceil \log n \rceil$ 個からなる．また，$\lceil \log n \rceil + 1$ ビットのフィールド $\lfloor \frac{n}{\lceil \log n \rceil} \rfloor$ 個からなるビットベクタ $\mathbf{s}$ を考え，$0 \leq j \leq \lfloor \frac{n-1}{\lceil \log n \rceil} \rfloor$ に対し，下位から j 番目のフィールド $\mathbf{s}[j]$ には $\sum_{k=0}^{j-1} |Y_{(n,k)} \wedge S|$ の値を格納する．このとき，次の補題が成り立つ．

補題 4. 集合 $S \subseteq X_n$，自然数 $k < |S|$ に対し，$r^{-1}(k, S)$ は，ビットベクタ $\mathbf{v}_j$，$\mathbf{s}$ を用いて次式により求めることができる．

$$r^{-1}(k, S) = \mathbf{v}_t[k - \mathbf{s}[t]] + t\lceil \log n \rceil + k - \mathbf{s}[t], \quad (3)$$

$$\text{ここに，} t = \max(j; j \in \mathbb{N}, \mathbf{s}[j] \leq k)$$

証明．式 (3) の右辺は，

$$\begin{aligned} & \mathbf{v}_t[k - \mathbf{s}[t]] + t\lceil \log n \rceil + k - \mathbf{s}[t] \\ = \; & r^{-1}(k - \mathbf{s}[t], Y_{(n,t)} \wedge S) \end{aligned}$$

と変形できる．$r^{-1}(k - \mathbf{s}[t], Y_{(n,t)} \wedge S) = x$ とおくと，t の条件より，

$$\mathbf{s}[t] \leq k < \mathbf{s}[t+1]$$

$$0 \leq k - \mathbf{s}[t] < \mathbf{s}[t+1] - \mathbf{s}[t] = |Y_{(n,t)} \wedge S|$$

よって，$x \in Y_{(n,t)} \wedge S$ が成り立ち，このとき

$$t\lceil \log n \rceil \leq x < (t+1)\lceil \log n \rceil$$

であるから，

$$r\left(x, \bigcup_{i<t} (Y_{(n,i)} \wedge S)\right) = \mathbf{s}[t]$$

$$r\left(x, \bigcup_{i>t} (Y_{(n,i)} \wedge S)\right) = 0$$

したがって，

$$\begin{aligned} r(x, S) &= r\left(x, \bigcup_{i} (Y_{(n,i)} \wedge S)\right) \\ &= r\left(x, \bigcup_{i<t} (Y_{(n,i)} \wedge S)\right) + r(x, Y_{(n,t)} \wedge S) \\ &= k \end{aligned}$$

$x \in S$ であるから，

$$r^{-1}(k, S) = x$$

以上から，$r^{-1}(k, S)$ の式が得られる． □

補題 5. 空でない集合 $S \subseteq X_n$ の要素 s に対し，集合 $S' = S \backslash \{s\}$ を考える．このとき，$Y \subseteq X_n$，$k \in \mathbb{N}$，$k < |Y \wedge S'|$ とすると次式が成り立つ．

$$r^{-1}(k, Y \wedge S') = \begin{cases} r^{-1}(k+1, Y \wedge S) \\ (s \in Y \text{ かつ } s \geq r^{-1}(k, Y \wedge S')) \\ r^{-1}(k, Y \wedge S) \quad (\text{その他}) \end{cases}$$

証明．$r^{-1}(k, Y \wedge S') = x$ とすると，定義より $r(x, Y \wedge S') = k$ が成り立つ．$s \notin Y$ のとき，$Y \wedge S' = Y \wedge S$ より，

$$r^{-1}(k, Y \wedge S') = r^{-1}(k, Y \wedge S)$$

$s \in Y$ のとき，$s \in Y \wedge S$ より，

$$Y \wedge S = (Y \wedge S') \vee \{s\}$$

であるから，

$$r(x, Y \wedge S) = r(x, Y \wedge S') + r(x, \{s\}) = k + r(x, \{s\})$$

よって，

$$\begin{aligned} r^{-1}(k, Y \wedge S') &= x \\ &= r^{-1}(r(x, Y \wedge S), Y \wedge S) \\ &= r^{-1}(k + r(x, \{s\}), Y \wedge S) \end{aligned}$$

したがって，

$$r^{-1}(k, Y \wedge S') = \begin{cases} r^{-1}(k+1, Y \wedge S) \\ (s \in Y \text{ かつ } s \geq r^{-1}(k, Y \wedge S')) \\ r^{-1}(k, Y \wedge S) \quad (\text{その他}) \end{cases}$$

□

補題 4 において，t は式 (2) を用いると，[6] 中で紹介されている手法により，定数時間で求めることができる．ビットベクタ $\mathbf{s}$ の全てのフィールドの値は常に $n(< 2^{\lceil \log n \rceil})$ 未満であるから，最上位ビットは常に 0 である．そのため，全てのフィールドの値を $2^{\lceil \log n \rceil} + k$ としたビットベクタ $\mathbf{c}$ を考え，$\mathbf{c} - \mathbf{s}$ を計算すると，$s[j] \leq k$ を満たすフィールドについてのみ最上位ビットは 1 のままである．$s[j]$ の値は下位のフィールドから単調増加であるから，最上位ビットのみを抽出し，式 (2) を用いて最上位ビットが 1 であるフィールドの個数を求めると t の値が得られる．したがって，補題 4 を用いると，ビットベクタ $\mathbf{v}_j$ および $\mathbf{s}$ を用いることで，$r^{-1}(i, S)$ は定数時間で計算できる．

$S = X_n$ のとき，ビットベクタ $\mathbf{v}_j$ の全てのフィールドの値は 0 であり，ビットベクタ $\mathbf{s}$ について $\mathbf{s}[j] = \sum_{k=0}^{j-1} \lceil \log n \rceil = j \lceil \log n \rceil$ である．また，補題 5 を用いると，ランク付けの逆操作において置換の要素 $\pi[i]$ を読み込んだとき，次の反復における $\mathbf{v}_j[k]$ を現在の $\mathbf{v}_j[k]$ を用いて計算できる．さらに，ビットベクタ $\mathbf{s}$ に対して，$S' = S \backslash \{\pi[i]\}$ とすると，$\pi[i] \in Y_{(n,j)}$ となる j に限り $Y_{(n,j)} \wedge S'$ の要素数が 1 だけ減少するので，$k > j$ を満たす k に対して，$\mathbf{s}[k]$ の値は 1 減少する．各ビットベクタの更新は，ランク付け操作の場合と同様に定数時間で可能である．

例として，ランク付けの場合と同様に $n = 8$ のときを考え，$S = \{1, 2, 3, 5, 7\}$ とすると，ビットベクタ $\mathbf{s}$, $\mathbf{v}_j$ は図 6 の通りになる．ただし，図では $\mathbf{s}$ の各フィールドの最上位ビットを省略している．$r^{-1}(2, S)$ を求めることを考えると，補題 4 において $t = 1$ より次のように求められる．

$$\begin{aligned} r^{-1}(2, S) &= \mathbf{v}_1[2 - \mathbf{s}[1]] + \lceil \log n \rceil + 2 - \mathbf{s}[1] \\ &= 3 \end{aligned}$$

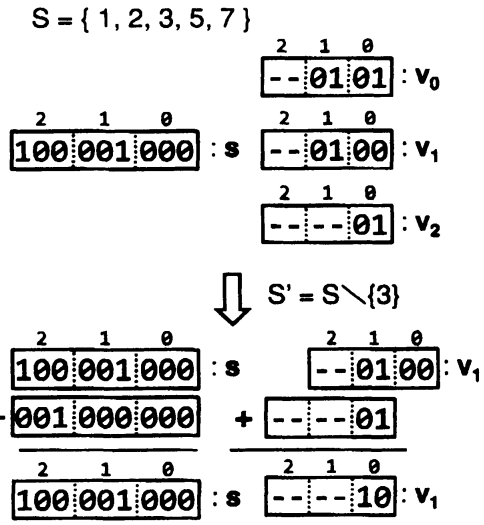

図 6: ランク付けの逆操作における $n = 8$, $S = \{1, 2, 3, 5, 7\}$, $S' = S \backslash \{3\}$ のときのビットベクタ

また，ビットベクタ $\mathbf{s}$, $\mathbf{v}_1$ の更新は，補題 5 より図 6 のように行えば良い．

以上から，次の定理が成り立つ．

定理 2. X_n 上の置換 π の辞書順ランクが与えられたとき，置換 π は $O(n \log \log n)$ ビット領域を用いて線形時間で計算できる．

アルゴリズムは図 7 の通りである．ここで，図 7 において，b, b' はそれぞれビットベクタ $\mathbf{s}$, $\mathbf{v}_j$ のフィールドサイズ，**inc**, **dec** はそれぞれ $\mathbf{v}_j$, $\mathbf{s}$ に対する操作に用いるビットベクタである．領域計算量に関しては 3.3 節に述べる．

3.3 領域計算量と実装

本節では 32 ビット環境における提案手法の実装および，定理 1，2 に示したアルゴリズムの作業領域が $O(n \log \log n)$ ビットであることを示す．

ランクの表現に用いる $[0, n! - 1]$ の範囲の全ての整数を表現できるのは，32 ビット環境では n が 12 以下の場合である．ここで，$n = 12$ の場合を考えると，提案手法におけるビットベクタ $\mathbf{v}_j$, $\mathbf{s}$ に関する性質を用いることで 32 ビット整数 1 個に全てのビットベクタを収めることができる．

まず，ランク付け操作について考える．ビットベクタ $\mathbf{v}_j$ に対し，補題 2 より常に $\mathbf{v}_j[0]$ =

入力 :集合X_n上の置換πのランク$rank$
出力 :置換π

$S \leftarrow X_n$;
for$(j=0; j \leq \lfloor \frac{n-1}{\lceil \log n \rceil} \rfloor; j++)$ $\mathbf{s}[j] \leftarrow j\lceil \log n \rceil$
for$(j=0; j \leq \lfloor \frac{n-1}{\lceil \log n \rceil} \rfloor; j++)$ $\mathbf{v}_j \leftarrow 0$
$b \leftarrow \lceil \log n \rceil + 1$
$b' \leftarrow \lceil \log \lceil \log n \rceil \rceil$
$\mathbf{inc} \leftarrow \sum_{i=0}^{\lceil \log n \rceil - 1} 2^{b'i}$
$\mathbf{dec} \leftarrow \sum_{i=0}^{\lfloor \frac{n-1}{\lceil \log n \rceil} \rfloor} 2^{bi}$
for$(i=0; i<n; i++)\{$
  $k \leftarrow \lfloor rank/(n-1-i)! \rfloor$
  $rank \leftarrow rank \bmod (n-1-i)!$
  $\mathbf{c} \leftarrow \quad \mathbf{dec} \cdot (\lceil \log n \rceil + k)$
  $t \leftarrow (((\mathbf{c}-\mathbf{s}) >> (b-1)) \& \mathbf{dec}) \bmod (2^b-1)$
  $\pi[i] \leftarrow \mathbf{v}_t[k-\mathbf{s}[t]] + t\lceil \log n \rceil + k - \mathbf{s}[t]$
  $\mathbf{s} \leftarrow \mathbf{s} - \mathbf{dec} \& \tilde{}(2^{b(t+1)}-1)$
  $\mathbf{v}_t \leftarrow ((\mathbf{v}_t + \mathbf{inc} >> b') \& \tilde{}(2^{b't}-1)) | (\mathbf{v}_t \& (2^{b't}-1))$

$\}$
return π

図 7: 置換のランク付けの逆操作に対する提案手法

$r(j\lceil \log n \rceil, Y_{(n,j)} \backslash S) = 0$であることが示される．また，ビットベクタ$\mathbf{s}$のフィールドサイズは式 (2) を用いるために$\lceil \log n \rceil$ビット必要であるが，$\mathbf{s}$の要素を表現するには，$0 \leq \mathbf{s}[j] \leq \lceil \log n \rceil$であるから，$\lceil \log (\lceil \log n \rceil + 1) \rceil$ビットあれば十分である．そこで，ビットベクタ$\mathbf{v}_j$の間に$\mathbf{s}$の要素を配置することで，式 (2) の利用に必要なフィールドサイズはビットマスクにより確保できるようになる．このとき，ビットベクタ全体でビット数は

$$\left(\left\lfloor \frac{n-1}{\lceil \log n \rceil} \right\rfloor + 1 \right) \lceil \log \lceil \log n \rceil \rceil (\lceil \log n \rceil - 1) + \left\lfloor \frac{n-1}{\lceil \log n \rceil} \right\rfloor \lceil \log (\lceil \log n \rceil + 1) \rceil \qquad (4)$$

ビットとなる．

次に，ランク付けの逆操作について考える．ビットベクタ$\mathbf{v}_j$に対し，$\mathbf{v}_j[\lceil \log n \rceil - 1]$が参照されるのは，$Y_{(n,j)} \wedge S = Y_{(n,j)}$のときだけであり，このとき$\mathbf{v}_j[\lceil \log n \rceil - 1] = 0$である．また，ビットベクタ$\mathbf{s}$において，最上位ビットは常に 0 であるから，ランク付け操作と同様にビットベクタを配置することでビットを削減できる．さらに，$\mathbf{s}[0]$は常に 0 であるため，この分のビットは削減することができる．このとき，ビットベクタ全体でビット数は，

$$\left(\left\lfloor \frac{n-1}{\lceil \log n \rceil} \right\rfloor + 1 \right) \lceil \log \lceil \log n \rceil \rceil (\lceil \log n \rceil - 1) + \left\lfloor \frac{n-1}{\lceil \log n \rceil} \right\rfloor \lceil \log n \rceil \qquad (5)$$

ビットとなる．

以上の構造を用いたランク付けおよびその逆操作のC++による実装を付録に示す．

ここで，式 (4)，(5) より，定理 1，2 に示したアルゴリズムの作業領域に関して次の補題が成り立つ．

補題 6. 図 5 のアルゴリズムによるランク付け操作および，図 7 のアルゴリズムによるランク付けの逆操作はいずれも$O(n \log \log n)$ビット領域で計算できる．

表 1: ランク付け操作に必要なビット数

n	提案手法 (ランク付け)	提案手法 (逆操作)	Mares らの手法
4	4	4	12
5	10	11	20
6	10	11	24
7	16	18	28
8	16	18	32
9	24	26	45
10	24	26	50
11	24	26	55
12	24	26	60

既存の手法である Mares らの手法 (ビット数は$n(\lceil \log n \rceil + 1)$ビット)[6] および提案手法におけるビットベクタの表現に必要なビット数は表 1 の通りである．提案手法では，既存の手法に対し大幅にビットを削減できており，32 ビット環境においては，ランクを表現できる範囲のnに対しては 32 ビット整数 1 個ですべてのビットベクタを表現できることが示された．なお，ビットベクタ$\mathbf{v}_j$および$\mathbf{s}$はそれぞれ分割して格納しても良い．この場合各ビットベクタのビット数は$O(n)$ビットであるから，提案手法の実行には$O(n)$ビットの整数に対する四則演算およびビット演算が定数時間で出来れば良い．

4 実験

Intel Core 2 Duo プロセッサ T7500 を搭載した動作周波数 2.2 Ghz の PC 上で，提案手法および，既存の線形時間アルゴリズムである Myrvold らのアルゴリズム [5]，Mares らのアルゴリズム [6] の実装を行い，ランク付け操作およびその逆操作の実行時間を測定した．測定は，それぞれ 10! 通りの置換・ランクに対する計算の総実行時間に対して行った．結果は表 2，3 の通りであった．また，Mares らのアルゴリズムに関しては，32 ビット環境で測定したため，n が 8 より大きい場合に関しては測定していない．

表 2: ランク付け操作の実行時間 (単位は [ms])

n	提案手法	Mares らの手法	Myrvold らの手法
4	18	16	45
5	26	26	56
6	33	39	64
7	37	46	78
8	41	58	80
9	51	-	101
10	53	-	110
11	58	-	115
12	62	-	134

提案手法はランク付け操作に関しては既存の手法に比べ高速であったが，逆操作ではいずれの手法よりも遅かった．提案手法では，ランク付け操作に比べて逆操作に必要な計算が多い．そのため，逆操作に対してはアルゴリズムの改良が必要とされる．

表 3: ランク付けの逆操作の実行時間 (単位は [ms])

n	提案手法	Mares らの手法	Myrvold らの手法
4	65	34	43
5	82	41	54
6	103	48	64
7	122	56	77
8	139	63	88
9	156	-	102
10	169	-	113
11	182	-	120
12	191	-	131

5 まとめ

本稿では，置換のランク付け操作およびその逆操作に対し，$O(n \log\log n)$ ビット領域の線形時間アルゴリズムを提案した．また，32 ビット環境に関して，提案手法はランクすなわち $[0, n! - 1]$ の範囲の整数が表現できる範囲の n に対しては 32 ビット整数 1 個を作業領域として実行可能であることを示した．さらに，ランク付け操作は既存の手法よりも高速に動作することを示した．今後の課題としては，ランク付けの逆操作の高速化が挙げられる．

参考文献

[1] J. Liebehenschel : Ranking and Unranking of Lexicographically Ordered Words: An Average-Case Analysis. Journal of Automata, Languages and Combinatiorics 2(4), 227-268 (1997)

[2] E. M. Reingold, J. Nievergelt, and N. Deo: Combinatorial Algorithms: Theory and Practice. Prentice Hall College Div., Englewood Cliffs (1977)

[3] P. F. Dietz : Optimal algorithms for list indexing ans subset rank, Workshop on Algorithms and Data Structures (WADS), Lecture Notes in Computer Science, 382, 39-46 (1989)

[4] Knuth, D.: The Art of Computer Programming, Sorting and Searching, vol. 3. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA (1998)

[5] W. Myrvold and F. Ruskey : Ranking and Unranking Permutations in Linear Time, Information Processing Letters 79(6), 281-284 (2001)

[6] M. Mares and M. Straka : Linear-Time Ranking of Permutations, ESA2007, LNCS 4698, pp. 187-193 (2007)