r インデックスにおける接尾辞配列を模倣するデータ構造

Christina Boucher* Dominik Köppl† Herman Perera*
Massimiliano Rossi*

令和５年２月１日

概要 任意長さ $n \geq 1$ が持つテキスト T に対して，T の Burrows–Wheeler transform (BWT) の上で，パターン照合を行うことができる．ただし，パターンの出現は BWT の範囲で表現されている．r を BWT の連超圧縮のサイズとすると，BWT の検索を $\mathcal{O}(r)$ ワードで行うことができる．パターンの出現（すなわち，T の中の開始位置）が必要な場合，接尾辞配列 SA で BWT の位置を T の位置に写像できる．圧縮できるテキストにおいて $r \ll n$ であるため，n ワードの領域を取る SA は全体のインデクスの領域のボトルネックが生じる．そのために，Gagie et al. [4] は $\mathcal{O}(r)$ 領域で表現できる ϕ 配列で BWT の出力した範囲からパターンの開始位置を計算したが，その方法はかなり複雑で，実際的に遅い．その計算をより速く行うために本論では ϕ^{-1} グラフを提案する．ϕ^{-1} グラフは SA のサンプリングを使い，SA のアクセスを加速する目的がある．

序論 任意パターン文字列 P に対して，入力テキスト $T[1..n]$ のテキスト索引は以下の 2 つ問い合わせ（クエリー）を答えられる．count(P) は T の中に現れる P の出現の個数を出力し，locate(P) は T の中に現れる P の出現の開始位置を出力する．locate(P) の出力は集合 $\mathcal{S} := \{i \in [1..n] : T[i..i+|P|-1] = P\}$ である．count(P) は集合 $\mathcal{S}$ のサイズ $|\mathcal{S}|$ で計算できるが，より多くの索引は count(P) のほうが簡単に計算できる．BWT に基づいた FM-index [2] はその索引の一つである．BWT 以外，FM-index は完結データ構造のみでクエリー count(P) を答えられる．その度に，BWT の範囲 $[i..j] \subset [1..n]$ を出力する．接尾辞配列 SA で，locate(P) $= \{\mathrm{SA}[k] : k \in [i..j]\}$ を表現できる．そのため，BWT や SA 以外の比較的小さいデータ構造で $\mathcal{O}(n)$ 領域の FM-index を構築できる．

T = GATTACAT$GATACAT$GATTAGATA#

i	SA	BWT	rotations matrix
1	**27**	A	#GATTACAT$GATACAT$GATTAGATA
2	**9**	T	$GATACAT$GATTAGATA#GATTACAT
3	17	T	$GATTAGATA#GATTACAT$GATACAT
4	26	T	A#GATTACAT$GATACAT$GATTAGAT
5	5	T	ACAT$GATACAT$GATTAGATA#GATT
6	13	T	ACAT$GATTAGATA#GATTACAT$GAT
7	**22**	T	AGATA#GATTACAT$GATACAT$GATT
8	**7**	C	AT$GATACAT$GATTAGATA#GATTAC
9	**15**	C	AT$GATTAGATA#GATTACAT$GATAC
10	**24**	G	ATA#GATTACAT$GATACAT$GATTAG
11	11	G	ATACAT$GATTAGATA#GATTACAT$G
12	2	G	ATTACAT$GATACAT$GATTAGATA#G
13	**19**	G	ATTAGATA#GATTACAT$GATACAT$G
14	**6**	A	CAT$GATACAT$GATTAGATA#GATTA
15	14	A	CAT$GATTAGATA#GATTACAT$GATA
16	**23**	A	GATA#GATTACAT$GATACAT$GATTA
17	**10**	$	GATACAT$GATTAGATA#GATTACAT$
18	**1**	#	GATTACAT$GATACAT$GATTAGATA#
19	**18**	$	GATTAGATA#GATTACAT$GATACAT$
20	**8**	A	T$GATACAT$GATTAGATA#GATTACA
21	16	A	T$GATTAGATA#GATTACAT$GATACA
22	**25**	A	TA#GATTACAT$GATACAT$GATTAGA
23	**4**	T	TACAT$GATACAT$GATTAGATA#GAT
24	**12**	A	TACAT$GATTAGATA#GATTACAT$GA
25	**21**	T	TAGATA#GATTACAT$GATACAT$GAT
26	**3**	A	TTACAT$GATACAT$GATTAGATA#GA
27	**20**	A	TTAGATA#GATTACAT$GATACAT$GA

表 1: SA, BWT と入力テキスト T の辞書式順序でソートされた循環文字列．各 BWT の連に対して SA の値は太文字で表せる．この例で，連の個数は $r = 13$ である．

反復の多い圧縮可能なデータを入力として扱う場合，その長さに対して線形的な領域を取ることは大きくなるに従って不可能になる．しかし，r 連が持つ連長圧縮した BWT でも FM-index のように count(P) を答えることができる [5]．接尾辞配列の代わりに locate(P) をどのように計算するかということについて近年以下のように提案された．Gagie et al. [4] は r インデクスと呼ばれた連超圧縮した BWT に基づいて $\mathcal{O}(r)$ 領域を持つテキスト索引を提案した．r インデクスの方針は，SA の変わりに，$\mathcal{O}(r)$ 領域で表現できる ϕ 配列の使い方を工夫したものである．$\mathcal{O}(r)$ 領域の ϕ 配列で理論的に対数的な時間で SA にアクセスできるが，実際的に複数のランダムアクセスが必要なので，遅い演算であると知られている．本論では，SA のアクセスを高速的

*University of Florida
†東京医科歯科大学

表 2: 例の $\mathcal{S}$， $\mathcal{E}$，コスト c_x とリミット ℓ_x.

x	1	2	3	4	5	6	7	8	9	10	11	12	13
$\mathcal{E}[x]$	27	22	15	19	23	10	1	18	25	4	12	21	20
$\mathcal{S}[x]$	27	9	7	24	6	10	1	18	8	4	12	21	3
c_x	5	3	1	2	0	0	0	4	0	0	0	0	2
ℓ_x	1	1	3	1	2	2	3	1	2	6	3	2	

に計算できるという問題を取り込む ϕ^{-1} グラフを提案する [1]．ϕ^{-1} グラフは SA のサンプリングを使い，SA のアクセスを加速する目的がある．表 1 で連続の例で前述したデータ構造が示された．

既存研究　$\mathcal{O}(r)$ 領域で SA アクセスするために， Gagie et al. [3] は以下の補題を使っている．

補題 1. $\mathrm{BWT}[i] = \mathrm{BWT}[i+1] \implies \mathrm{SA}[i+1] - \mathrm{SA}[i] = \mathrm{SA}[j+1] - \mathrm{SA}[j]$，ただし $\mathrm{SA}[j] = \mathrm{SA}[i] - 1$.

補題の使い方を説明するために，以下の状況を満たす最大の $k \geq 0$ を仮定する．任意 $i' : \mathrm{SA}[i'] \in [\mathrm{SA}[i] - k + 1..SA[i]]$ に対して，$\mathrm{BWT}[i'] = \mathrm{BWT}[i'+1]$．それなら，$\mathrm{SA}[j] = \mathrm{SA}[i] - k$ とすると，補題 1によって，$\mathrm{SA}[i+1] - \mathrm{SA}[i] = \mathrm{SA}[j+1] - \mathrm{SA}[j]$．そのため，BWT の連の境目の SA 値を保存することで，SA にアクセスできる．

$\mathcal{S}[x]$ と $\mathcal{E}[x]$ は，それぞれに，x 番目の連の開始位置と終了位置の SA 値を示す，ただし $x \in [1..r]$ である．$\mathcal{E}$ の上で，SA に高速にアクセスするとめに，以下のクエリーは便利になる．$\mathcal{E}.\mathrm{pred}(p) = \max\{q \in \mathcal{E} : q \leq p\}$，$\mathcal{E}.\mathrm{succ}(p) = \min\{q \in \mathcal{E} : q > p\}$．表 2 に例を示す．ここまで，既存研究は 2 つのクエリーと $\mathcal{S}$ から $\mathcal{E}$ までの $\mathcal{O}(r)$ 領域が持つ ϕ 関数で locate を答えられる．

ϕ^{-1} グラフ　基本的な方針は，$\mathcal{E}.\mathrm{pred}$ の呼び出す回数を減らすために，計算のステップをグラフで表現し，ノードに訪れる度に $\mathcal{E}.\mathrm{pred}$ の呼び出しを省くことができる．厳密に言えば，$\mathcal{E}$ ノードとして扱い，$\mathcal{E}[y] = \mathcal{E}.\mathrm{pred}(\mathcal{S}[x+1])$ の場合は，$\mathcal{E}[x]$ から $\mathcal{E}[y]$ までのアークを書くことで，ϕ^{-1} グラフが成立する．各ノードは外に向かうアークを最大一つ持つので，ノードとアークを同一視できる．$\mathcal{E}[x]$ のアークを以下の 2 つの指数でラベル付けする．$c_x := \mathcal{S}[x+1] - \mathcal{E}.\mathrm{pred}(\mathcal{S}[x+1])$ は x

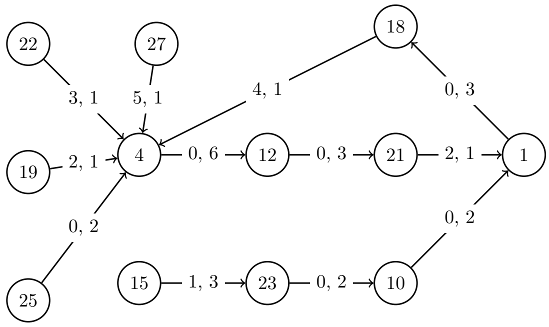

図 1: 連続の例の ϕ^{-1} グラフ．ノードは BWT の連の最後の SA 値を表現する．各アークはコストとリミットでラベルされている．

番目の連のコストと呼ばれ，$\ell_x := \mathcal{E}.\mathrm{succ}(\mathcal{E}[x]) - \mathcal{E}[x]$ は x 番目の連のリミットと呼ばれる．　連続の例として，完成したグラフを図 1 で表している．

このようにして，SA のアクセスを以下のアルゴリズムで行うことができる．まず，ノード $p \leftarrow \mathcal{E}.\mathrm{pred}(\mathrm{SA}[i])$ を訪れる，ただしアルゴリズムは積み重ねるコスト t を管理する．最初に，t は $\mathrm{SA}[i] - p$ を設定する．任意の訪れたノード p に対して，p のアークは (p, v) だとすると，(p, v) のリミットは t を超える場合は，アルゴリズムをストップする．そうではない場合は，(p, v) のコストを p に追加し，v を訪れ，$p + v$ という次の SA 値を出力し，繰り返す．ストップの場合は，新しい pred を選択すべきである．そのために，最後の出力は $p + v$ とすることで，$p \leftarrow \mathcal{E}.\mathrm{pred}(p + v)$ を設定し，アルゴリズムを続けることができる．

参考文献

[1] C. Boucher, D. Köppl, H. Perera, and M. Rossi. Accessing the suffix array via ϕ^{-1}-forest. In *Proc. SPIRE*, volume 13617 of *LNCS*, pages 86–98, 2022.

[2] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proc. FOCS*, pages 390–398, 2000.

[3] T. Gagie, G. Navarro, and N. Prezza. Optimal-time text indexing in BWT-runs bounded space. In *Proc. SODA*, pages 1459–1477, 2018.

[4] T. Gagie, G. Navarro, and N. Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM*, 67(1):2:1–2:54, 2020.

[5] V. Mäkinen and G. Navarro. Succinct suffix arrays based on run-length encoding. *Nord. J. Comput.*, 12 (1):40–66, 2005.