# 7.4 圧縮接尾辞配列

甲本 健太

# 目次

# 圧縮接尾辞配列の概要 (1/2)

**圧縮接尾辞配列** (**CSA**: compressed suffix array):
→ 接尾辞配列のサイズを小さくしたもの

## 接尾辞配列との比較

| データ構造 | サイズ | $SA[i]$ の計算 |
|---|---|---|
| 接尾辞配列（非圧縮） | $n \lg n$ bits | $O(1)$ 時間 |
| 圧縮接尾辞配列 | $O(n \lg \sigma)$ bits | $O(\lg^{\varepsilon} n)$ 時間 |

## 計算時間について

圧縮接尾辞配列の各要素は $\mathrm{polylog}(n)$ 時間で復元できる[1].
→ 検索速度はあまり落ちない

---

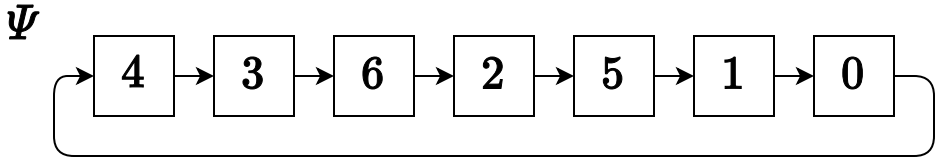
[1] $\mathrm{polylog}(n) := O(\lg^k n)$

# 圧縮接尾辞配列の概要 (2/2)

圧縮接尾辞配列 (CSA) は以下のように実現される．

1. 接尾辞配列の要素を間引く
   - 任意の自然数 $h$ に対して，$n/h$ 個の要素のみを格納
   - $h$ はデータ構造のパラメータ
2. 間引かれた要素は，関係を用いて表す
   - 残された要素同士の関係を関数 $\Psi$ で表す

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| SA | 7 | 6 | 4 | 2 | 1 | 5 | 3 |
| $\mathrm{SA}_0$ | | 6 | | | | | 3 |



# 関数 Ψ の性質 (1/5)

文字列 $T$ の接尾辞配列を $SA$ とする．このとき $\Psi$ 関数を次のように定義する．

$$\Psi[i] := \begin{cases} SA^{-1}[SA[i]+1] & \text{if} \quad SA[i] < n, \\ SA^{-1}[1] & \text{if} \quad SA[i] = n. \end{cases}$$

**気持ち**

- $SA[i]$: $\{T[j..] \mid 1 \leq j \leq n\}$ を辞書順に並べたとき $i$ 番目にあたる $j$ の値
- $SA^{-1}[j]$: $T[j..]$ は辞書順で何番目にあたるか

- $\Psi[i]$: 辞書順で $i$ 番目にあたる接尾辞 $T[j..]$ から**先頭 1 文字を削除した文字列** $T[j+1..]$ は辞書順で何番目にあたるか

# 関数 Ψ の性質 (2/5)

$T =$ “banana” としたときの例.

| $i$ | $SA[i]$ | $T[SA[i]..n+1]$ | $SA[i]+1$ | $\Psi[i]$ |
|---|---|---|---|---|
| 0 | 7 | $ | 1 | 4 |
| 1 | 6 | a$ | 7 | 0 |
| 2 | 4 | ana$ | 5 | 5 |
| 3 | 2 | anana$ | 3 | 6 |
| 4 | 1 | banana$ | 2 | 3 |
| 5 | 5 | na$ | 6 | 1 |
| 6 | 3 | nana$ | 4 | 2 |

| $j$ | $SA^{-1}[j]$ |
|---|---|
| 1 | 4 |
| 2 | 3 |
| 3 | 6 |
| 4 | 2 |
| 5 | 5 |
| 6 | 1 |
| 7 | 0 |

# 関数 Ψ の性質 (3/5)

関数 $\Psi$ は補題 7.2 の性質を満たす．

**補題 7.2**

$0 \leq i < j \leq n$ に対し，$T[SA[i]] = T[SA[j]]$ ならば $\Psi[i] < \Psi[j]$．

**証明**

定義より $\Psi[i] = SA^{-1}[SA[i]+1], \Psi[j] = SA^{-1}[SA[j]+1]$ である．つまり，$\Psi[i]$ と $\Psi[j]$ の大小関係は接尾辞 $T[SA[i]+1..n+1]$ と $T[SA[i]+1..n+1]$ の辞書順で定義される．
今，$T[SA[i]] = T[SA[j]]$ であるため，接尾辞 $T[SA[i]+1..n+1]$ と $T[SA[j]+1..n+1]$ の大小関係は $T[SA[i]..n+1]$ と $T[SA[j]..n+1]$ の大小関係と等しい．$i < j$ であり，接尾辞配列の定義から $T[SA[i]..n+1] < T[SA[j]..n+1]$ であるため，$\Psi[i] < \Psi[j]$ となる． □

**気持ち**

文字列 $S, T$ について，$S < T$ かつ $S[1] = T[1]$ であるとき，$S[2..] < T[2..]$ である．

# 関数 Ψ の性質 (4/5)

補題 7.2 より，$\Psi$ 関数は高々 $\sigma$ 個の**狭義単調増加列**に分解できる．
→ 単調増加列の圧縮といえば… Elias-Fano 符号とそれを拡張したデータ構造 GV．

---

## 復習: Elias-Fano 符号

Elias-Fano 符号（イライアス・ファノ符号と読む）は，広義単調増加する非負整数の列に対する符号である．

**符号化方法**

$n$ 個の非負整数 $0 \leq x_1 \leq \cdots \leq x_n = u$ とする．$\{x_i\}$ の符号は以下のように行う．

1. $x_i$ の下位 $s$ ビットを $r_i$，下位 $s$ ビットを除いたものを $q_i$ とする．
2. $\{q_i\}$ は広義単調増加になっていることから，
   $q_i$ の代わりに直前の値との差分 $d_i := q_i - q_{i-1}$ を 1 進数符号で符号化する．
3. $\{r_i\}$ は $s$ ビットの 2 進数で符号化する．

（データ構造 GV は 第 7 回資料 を参照）

# 関数 Ψ の性質 (5/5)

補題 7.2 より，$\Psi$ 関数は高々 $\sigma$ 個の狭義単調増加列に分解できる．さらに，

$$\Psi'[i] = T[SA[i]] \cdot (n+1) + \Psi[i]$$

と定義すると，$\Psi'$ 関数は狭義単調増加関数になる．
（ただし，$T[SA[i]]$ は整数にエンコードしてあるものとする．）

関数 $\Psi'$ はデータ構造 GV を用いて，

- $n(2 + \lg \sigma) + O(n \lg \lg n / \lg n)$ ビットで表現できる．
- $\Psi'[i], \Psi[i], T[SA[i]]$ は定数時間で求まる．
  - $\Psi[i] = \Psi'[i] \bmod (n+1)$
  - $T[SA[i]] = \lfloor \Psi'[i]/(n+1) \rfloor$

  とすればよい．

# 圧縮接尾辞配列上で SA[i]を求める (1/2)

CSA は，以下の 2 つの情報からなる．

1. SA の要素で， $SA[i]$ が $h$ の倍数であるもののみからなる配列
2. 関数 $\Psi[i]$ を GV で圧縮したもの

このとき，1 に格納されていない SA の要素は次のように求める．

$\mathrm{CSA}(h=3)$

| $i$ | $\mathrm{SA}[i]$ | $\Psi[i]$ | $T[\mathrm{SA}[i]..n+1]$ |
|---|---|---|---|
| 0 | 7 | 4 | $ |
| 1 | 6 | 0 | a$ |
| 2 | 4 | 5 | ana$ |
| 3 | 2 | 6 | anana$ |
| 4 | 1 | 3 | banana$ |
| 5 | 5 | 1 | na$ |
| 6 | 3 | 2 | nana$ |

3 の倍数のみ格納

GV で圧縮

例） $SA[4]$ を求める

1. $SA[4]$ は格納されている？
   → No.
2. $\Psi[4]=3$，$SA[3]$ は格納されている？
   → No.
3. $\Psi[3]=6$，$SA[6]$ は格納されている？
   → Yes. $SA[6]=3$
4. よって，$SA[\Psi^2[4]]=3$
   $SA[\Psi[i]]=SA[i]+1 \Rightarrow SA[4]=1.$

# 圧縮接尾辞配列上で SA[i]を求める (2/2)

先のページの内容をちゃんと言うと，次のようになる．

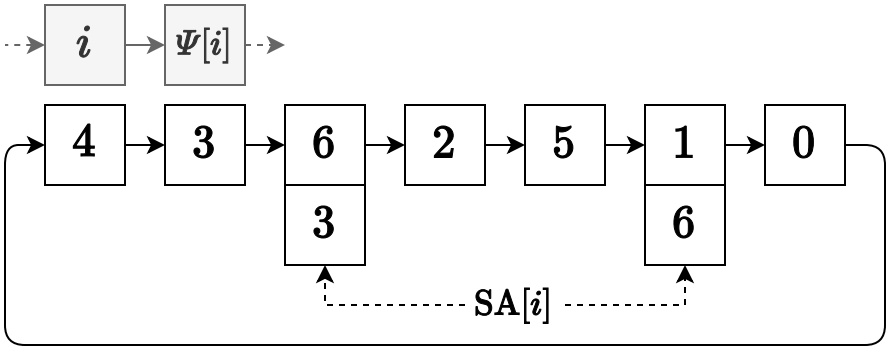
**補題 a**

配列 $SA_0$ を用いることで，すべての $i$ に対して高々 $2h-2$ 回の $\Psi$ の計算で $SA[i]$ を求めることができる．

**証明**

$j = SA[i]$ を求める場合，$SA[\Psi[i]] = j+1$ であることから，$SA[\Psi^k[i]]$ が $h$ の倍数である最小の $k$ を求めればよい．

通常は $k \leq h-1$ であるが，文字列の末尾で $\Psi$ を求めると $SA$ の値が 1 になるため，さらに $h-1$ 回の $\Psi$ の計算が必要である． □



# 圧縮接尾辞配列の計算量

- どの要素が $SA_0$ に格納されているか → 長さ $n$ のビットベクトル $B$ で管理
  - $B[i] = 1$ ならば $SA[i]$ が格納されている
  - $SA[i]$ は $SA_0[rank_1(B, i)]$ に格納
- 計算量:

  $h = \Theta(\lg^{1+\varepsilon} n)$（$\varepsilon$ は任意の正定数）を仮定すると，
  - $\Psi'$ は GV を用いると $n(2 + \lg \sigma) + O(n \lg\lg n / \lg n)$ ビットで表現できる
  - $B$ は FID を用いると $\mathcal{B}(n/h, n) + O(n \lg\lg n / \lg n)$ ビットで格納可能
    - $h = \Omega(\lg n)$ より，$O(n \lg\lg n / \lg n)$ ビット
  - $SA_0$ は $\lg n \cdot n/h = o(n)$ ビットで格納可能
  - $SA[i]$ は $O(h) = O(\lg^{1+\varepsilon} n)$ 時間で復元可能

以上より，CSA の計算量に関して以下の定理が得られる．

**定理 7.1 (データ構造 CSA-GV)**

長さ $n$，アルファベットサイズ $\sigma$ の文字列の接尾辞配列は，任意の正定数 $h$ を用いて $n(2 + \lg \sigma) + O(n \lg n/h) + O(n \lg\lg n / \lg n)$ ビットで表現でき，接尾辞配列の 1 つの要素は $O(h)$ 時間で計算できる．

# 圧縮接尾辞配列上で SA[i]を求める（疑似コード）

CSA 上で $SA[i]$ を求める関数の疑似コードは以下の通り.

アルゴリズム 7.2 $csa\_lookup(i)$: $SA[i]$ を求める.

```
1:  k ← 0
2:  while B[i] = 0 do
3:      k ← k + 1
4:      i ← Ψ[i]
5:  end while
6:  j ← SA_0[rank_1(B, i)] − k
7:  if j ≤ 0 then
8:      j ← j + n + 1
9:  end if
10: return j
```

# CSA のさらなる省スペース化 / 高速化

- 配列 $SA_0$ をそのまま格納せず，それを再帰的に表現することもできる
  - i. $SA_0$ は長さ $n/h$ の別の文字列 $T'$ の接尾辞配列になっている
    - SA から元の文字列を復元: ARC044: D - suffix array
  - ii. $T'$ の圧縮接尾辞配列を再帰的に作成する
- これにより，$SA[i]$ の復元速度を $O(\lg n)$ 時間から $O(\lg^{\varepsilon} n/\varepsilon)$ 時間にすることができる．（$\varepsilon$ は任意の定数）
  - $h = O(\lg n)$ を仮定？
- データ構造のサイズは $O(n \lg \sigma/\varepsilon)$ ビットになる．

# CSA の実用上の実装 (1/2)

※ 1 枚前のスライドとは関係ないです

今までは，$SA[i]$ が $h$ の倍数のときに $SA_0$ に格納していた
→ $i$ が $h$ の倍数のときに $SA'_0$ に格納するよう，修正

- メリット:
  - ビットベクトル $B$ を保持する必要がなくなるため，省スペース
  - $B[i]$ を求める操作も不要になるため，その分高速になる
- デメリット:
  - $i$ が $h$ の倍数になるまでに $\Psi$ を計算する回数の上限値が抑えられない
  - 期待値としても，回数が $\Psi$ の計算回数が 2 倍になる

次ページに，この方針で実装した場合のアルゴリズムを示す．

# CSA の実用上の実装 (2/2)

- $SA'_0$ : $i$ が $h$ の倍数である $SA[i]$ のみを格納した配列

---

アルゴリズム 7.3 $csa\_lookup2(i)$: $SA[i]$ を求める.

```
1:  k ← 0
2:  while i mod h ≠ 0 do
3:      k ← k + 1
4:      i ← a[i]
5:  end while
6:  j ← SA'_0[i/h] − k
7:  if j ≤ 0 then
8:      j ← j + n + 1
9:  end if
10: return j
```

# 自己索引化 (1/4)

7.4.1 項で扱った CSA と通常の SA の比較は以下の通り.

| | **接尾辞配列** | **圧縮接尾辞配列** |
|---|---|---|
| 配列のサイズ | $n \lg n$ ビット | $O(n \lg \sigma)$ ビット |
| $SA[i]$ を求める | $O(1)$ 時間 | $O(\lg^{\varepsilon} n)$ 時間 |
| パタン $P$ の出現頻度クエリ | $O(\|P\| \lg n)$ 時間 | $O((\|P\| + \lg^{\varepsilon} n) \lg n)$ 時間 |

**自己索引とは**

- CSA を用いると配列のサイズは元のテキストと同じオーダーになった
- 一方，検索のためには元のテキスト $T$ を保持する必要がある
  → CSA を自己索引化すれば，元のテキストを保持しなくてよい

**自己索引**:
データを検索するための索引構造で，検索時に元データを必要としないデータ構造

# 自己索引化 (2/4)

**復習: SA でのパタン $P$ の検索**

- SA の 2 分探索で求める.

  i. 接尾辞配列の中央の要素 $j = SA[n/2]$ を求める
  ii. $T[j..j + |P| - 1]$ と $P$ を比較する
  ⋮

  ステップ 2 で元のテキスト $T$ を参照する必要があった.

**CSA からの部分文字列の復元**

- 実は，$T$ の部分文字列 $T'(:= T[j..j + |P| - 1])$ は $\Psi'$ 関数から復元できる

  i. $T'$ の 1 文字目 $T[j] = T[SA[i]] = \lfloor \Psi'[i]/(n + 1) \rfloor$
  ii. $T'$ の 2 文字目 $T[j + 1]$:
    - $\Psi[i]$ が $T[j + 1..n + 1]$ を表すことを利用する.
    - 疑似コードを次ページに示す.

# 自己索引化 (3/4)

アルゴリズム 7.4 $csa\_substring(i, \ell)$: 部分文字列 $T[SA[i]..SA[i]+\ell+1]$ を求める.

```
1: for k ← 1, ℓ do
2:     S[k] ← ⌊Ψ'[i]/(n+1)⌋          ▷ T[SA[i]]
3:     i ← Ψ'[i] mod (n+1)            ▷ T[Ψ[i]]
4: end for
5: return S
```

| $i$ | $\mathrm{SA}[i]$ | $\Psi[i]$ | $\Psi'[i]$ |
|---|---|---|---|
| 0 | 7 | 4 | 4 |
| 1 | 6 | 0 | 7 |
| 2 | 4 | 5 | 12 |
| 3 | 2 | 6 | 13 |
| 4 | 1 | 3 | 17 |
| 5 | 5 | 1 | 22 |
| 6 | 3 | 2 | 23 |

| $T[\mathrm{SA}[i]..n+1]$ |
|---|
| $ |
| a$ |
| ana$ |
| anana$ |
| banana$ |
| na$ |
| nana$ |

| | |
|---|---|
| a | 1 |
| b | 2 |
| n | 3 |

# 自己索引化 (4/4)

## $T$ の部分文字列 $T[s..t]$ の復元

- $csa\_substring(i, \ell)$ を用いて $T$ の部分文字列 $T[s..t]$ を復元したい
- 接尾辞 $T[s..n+1]$ の辞書順が必要
  → 接尾辞配列の逆関数 $i = SA^{-1}[s]$ を計算する必要がある
  - $SA^{-1}$ も CSA と同様に値を間引いて格納する
  - $SA_0^{-1}[1..n/h]$ を $SA_0^{-1}[i] = SA^{-1}[ih]$ とする
    → 高々 $2h-2$ 回 $\Psi$ 関数を計算すれば $SA^{-1}[s]$ が求まる
  - 配列 $SA_0^{-1}$ は $n \lg n/h$ ビット

## 自己索引化のまとめ

**定理 7.2**

文字列 $T$ 中のパタン $P[1..m]$ の存在または頻度問い合わせは，$T$ の $\Psi$ 関数のみを用いて $O(m \lg n)$ 時間で行える．また，$O(n \lg n/h)$ ビットの補助データ構造を追加することで（$h$ は任意の正定数），$T$ の接尾辞配列の 1 つの要素は $O(h)$ 時間で計算でき，$T$ の任意の位置の長さ $\ell$ の部分文字列は $O(\ell + h)$ 時間で復元できる．