FM-index で高速文字列解析

72期 平田 誠治

はじめに

- どうしてレポートなのにスライド？

  → 図表を多用するためにスライドにしました。

レポートの流れ

- ▶ 1. 研究動機
- ▷ 2. 概要·序論
- ▷ 3. 研究方法
- ▷ 4. 研究結果
  - ▷ 4.1. Suffix Array
  - ▷ 4.2. BW 変換
  - ▷ 4.3. Bit Vector
  - ▷ 4.4. Wavelet Matrix
- ▷ 4. 研究結果
  - ▷ 4.5. FM-index
- ▷ 5. 結論·考察
- ▷ 6. 感想
- ▷ 7. 参考文献

研究動機

- 8/25 ～ 8/28　情オリ夏季セミナー

  高速文字列解析について勉強しました。

- アルゴリズムが面白かったので、

  レポートにもしようと思いました。

研究動機

・ 文字列アルゴリズムはDNAの解析にも応用されています。

研究動機

・ 文字列アルゴリズムはDNAの解析にも応用されています。

→ 新型コロナウイルスの研究にも役立つ！

レポートの流れ

- ▷ 1. 研究動機
- ▶ 2. 概要·序論
- ▷ 3. 研究方法
- ▷ 4. 研究結果
  - ▷ 4.1. Suffix Array
  - ▷ 4.2. BW 変換
  - ▷ 4.3. Bit Vector
  - ▷ 4.4. Wavelet Matrix
- ▷ 4. 研究結果
  - ▷ 4.5. FM-index
- ▷ 5. 結論·考察
- ▷ 6. 感想
- ▷ 7. 参考文献

問題設定

- 以下のような問題を高速に解きたいです。

- 文字列Ｓが与えられます。クエリに順に答えて下さい。
    - 文字列Ｔが与えられます。

      Ｓの中にＴは 何回出現しますか？

問題設定

- 例えば、 S = “mississippi” , T = “ssi” , “s” の場合

問題設定

- 例えば、 S = “mississippi” , T = “ssi” , “s” の場合

  → “ssi” は 2 回 登場しているので、答え は 2

  ”s” の答えは 4

問題設定

- 例えば、 S = “mississippi” , T = “ssi” , “s” の場合

  → “ssi” は 2 回 登場しているので、答え は 2

  ”s” の答えは 4

- 以降、|S| は S の長さを指します。

  また、”log” 単体では “$\log_2$”を指します。

愚直な解法

- 例えば、S のスタート地点を
  全探索する解法を考えます。

- すると、表のように比較回数が
  1 件あたり O(|S| |T|) となります。

	s	s	i	result
0	m	i	s	x
1	i	s	s	x
2	s	s	i	o
3	s	i	s	x
4	i	s	s	x
5	s	s	i	o
6	s	i	p	x
7	i	p	p	x
8	p	p	i	x
9	p	i	-	x
10	i	-	-	x

愚直な解法

- 今回は短い文字列でしたが、

  $|S| = 10^7, |T| = 10^4$ のようになったらどうでしょう？

愚直な解法

- 今回は短い文字列でしたが、

  $|S| = 10^7, |T| = 10^4$ のようになったらどうでしょう？

  → 1 件 あたり 10^{11} も時間計算量がかかってしまう！

愚直な解法

- 今回は短い文字列でしたが、

  $|S| = 10^7, |T| = 10^4$ のようになったらどうでしょう？

  → 1 件 あたり 10^{11} も時間計算量がかかってしまう！

- 普通のコンピュータが 1 秒 に処理できるのは、

  せいぜい 10^9 が限界です。

時間計算量目標

- 高速に処理を行うためには、|S| に依存しない時間計算量にする必要があります。

- 今回は、1 件あたり O (|T| log σ) を目指します。
  （σ は文字列に出現する文字の種類数）

空間計算量目標

- 高速に処理を行うためには、空間計算量（メモリ）も削減する必要があります。

- 空間計算量は $O(|S| \log |S|)$ を目指します。

レポートの流れ

研究方法

- 「高速文字列解析の世界」（著：岡野原 大輔）

  を読んで、実際にアルゴリズムを実装してみる。

- 情オリ夏季セミナーのメンバーで話し合いをしてみる。

レポートの流れ

- ▷ 1. 研究動機
- ▷ 2. 概要·序論
- ▷ 3. 研究方法
- ▶ 4. 研究結果
  - ▶ 4.1. Suffix Array
  - ▷ 4.2. BW 変換
  - ▷ 4.3. Bit Vector
  - ▷ 4.4. Wavelet Matrix
- ▷ 4. 研究結果
  - ▷ 4.5. FM-index
- ▷ 5. 結論·考察
- ▷ 6. 感想
- ▷ 7. 参考文献

Suffix Array とは？

- 日本語名で「接尾辞配列」と言い、

  文字列の接尾辞を辞書順に並べた配列の事。

Suffix Array とは？

- 日本語名で「接尾辞配列」と言い、

  文字列の接尾辞を辞書順に並べた配列の事。

- 実際には、各接尾辞の開始位置のみ保存します。

Suffix Array の例

- 簡単のため文字列の末尾には
  辞書順最小の“$” がついています。

	位置	Suffix
0	11	$
1	10	i$
2	7	ippi$
3	4	issippi$
4	1	ississippi$
5	0	mississippi$
6	9	pi$
7	8	ppi$
8	6	sippi$
9	3	sissippi$
10	5	ssippi$
11	2	ssissippi$

Suffix Array の構築

- SA-IS と呼ばれる Suffix Array を
  O(|S|) で構築するアルゴリズムが
  ありますが、割愛。
- 空間計算量は O(|S| log |S|) となります。

Suffix Array で出来ること

- 文字列 T が含まれる範囲を
  二分探索で特定できます。

Suffix Array で出来ること

- 文字列 T が含まれる範囲を
  二分探索で特定できます。

- 時間計算量は O(|T| log |S|)
  結構速い！

Suffix Array で出来ること

- “ssi” → 10 ～ 11 番目

	位置	Suffix
0	11	$
1	10	i$
2	7	ippi$
3	4	issippi$
4	1	ississippi$
5	0	mississippi$
6	9	pi$
7	8	ppi$
8	6	sippi$
9	3	sissippi$
10	5	ssippi$
11	2	ssissippi$

Suffix Array で出来ること

- “ssi” → 10 ～ 11 番目
- “s” → 8 ～ 11 番目

	位置	Suffix
0	11	$
1	10	i$
2	7	ippi$
3	4	issippi$
4	1	ississippi$
5	0	mississippi$
6	9	pi$
7	8	ppi$
8	6	sippi$
9	3	sissippi$
10	5	ssippi$
11	2	ssissippi$

Suffix Array で出来ること

- “ssi” → 10 ～ 11 番目
- “s” → 8 ～ 11 番目
- “tsukukoma” → なし

	位置	Suffix
0	11	$
1	10	i$
2	7	ippi$
3	4	issippi$
4	1	ississippi$
5	0	mississippi$
6	9	pi$
7	8	ppi$
8	6	sippi$
9	3	sissippi$
10	5	ssippi$
11	2	ssissippi$

Suffix Array で出来ること

- 文字列 T が含まれる範囲を

  二分探索で特定できます。

  しかし、まだ |S| に依存している！

- 時間計算量は O(|T| log |S|)

  結構速い！

レポートの流れ

- ▷ 1. 研究動機
- ▷ 2. 概要·序論
- ▷ 3. 研究方法
- ▶ 4. 研究結果
  - ▷ 4.1. Suffix Array
  - ▶ 4.2. BW 変換
  - ▷ 4.3. Bit Vector
  - ▷ 4.4. Wavelet Matrix
- ▷ 4. 研究結果
  - ▷ 4.5. FM-index
- ▷ 5. 結論·考察
- ▷ 6. 感想
- ▷ 7. 参考文献

BW変換とは？

- 簡単に言うと
  「各文字を、1つ後の接尾辞をキーとした辞書順に
  並び替えたもの」

BW変換とは？

- 簡単に言うと
  「各文字を、1つ後の接尾辞をキーとした辞書順に
  並び替えたもの」

- 定義的には
  「i 文字目は、Suffix Array の i 番目の1つ前の文字」

BW変換の例

- 0 文字目 … ‘$’(11) の前は ‘i’ (10)

	位置	Suffix
0	11	$
1	10	i$
2	7	ippi$
3	4	issippi$
4	1	ississippi$
5	0	mississippi$
6	9	pi$
7	8	ppi$
8	6	sippi$
9	3	sissippi$
10	5	ssippi$
11	2	ssissippi$

BW変換の例

- 0 文字目 … ‘$’(11) の前は ‘i’ (10)
- 1 文字目 … ‘i’ (10) の前は‘p’ (9)

	位置	Suffix
0	11	$
1	10	i$
2	7	ippi$
3	4	issippi$
4	1	ississippi$
5	0	mississippi$
6	9	pi$
7	8	ppi$
8	6	sippi$
9	3	sissippi$
10	5	ssippi$
11	2	ssissippi$

BW変換の例

- 0 文字目 … ‘$’(11) の前は ‘i’ (10)
- 1 文字目 … ‘i’ (10) の前は‘p’ (9)
- 2文字目 … ‘i’ (7)　の前は ‘s’ (6)

	位置	Suffix
0	11	$
1	10	i$
2	7	ippi$
3	4	issippi$
4	1	ississippi$
5	0	mississippi$
6	9	pi$
7	8	ppi$
8	6	sippi$
9	3	sissippi$
10	5	ssippi$
11	2	ssissippi$

BW変換の例

- 0 文字目 … ‘$’(11) の前は ‘i’ (10)
- 1 文字目 … ‘i’ (10) の前は‘p’ (9)
- 2文字目 … ‘i’ (7)　の前は ‘s’ (6)

  …

- “ipssm$pissii” と変換された。

	位置	Suffix
0	11	$
1	10	i$
2	7	ippi$
3	4	issippi$
4	1	ississippi$
5	0	mississippi$
6	9	pi$
7	8	ppi$
8	6	sippi$
9	3	sissippi$
10	5	ssippi$
11	2	ssissippi$

BW変換の性質

- BW変換をすると同じ文字が並びやすくなります。

（自然言語は、ある文字の後に来る文字に傾向がある）

BW変換の性質

- BW変換をすると同じ文字が並びやすくなります。

（自然言語は、ある文字の後に来る文字に傾向がある）

→ 文字列の圧縮がしやすくなる！

BW変換の性質

・例えば、下の文章を変換すると...

```
Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the
industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and
scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap
into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the
release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing
software like Aldus PageMaker including versions of Lorem Ipsum.
```

BW変換の性質

・例えば、下の文章を変換すると...

```
Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the
industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and
scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap
into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the
release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing
software like Aldus PageMaker including versions of Lorem Ipsum.
```

→ 圧倒的に同じ文字が並んでいる！！

```
.eeemmmm..f.gfsketn,egsn,eshdyogtyamtdregpmdeeoddestytmsnepee,dtsrgessyyennofhtrfadcyngt,ysyssgsmkdy560
191         $  e   PstmmMgi r    thedwlhwherp mu  ine nen sereenennen u
lnnhhkkpvhhrshrhpcgllrplslvgbhr rrrrrhmescckvtvgippd sLess ttloooo onnnnnnn anattc  ttttttswstnrlcs s
ntnthdtaarr s rl wwfvooiaansuA eebl baaacpnlteeeeuuuue aiuu immeewieaiuiaaaaiiiiiiiiiaoiiuo
kooieieietts    soor ictbtpmLLLLnnao yysyym   IIII oeeectaao  oooo uppateuttuatnai'a0ee0s aisaee i
rel  ae uupppp uxIxoeiInsn  ii      nnttnn  kecsseeenf    plpssssdd  tsdddbiie rt    oeellemmllrrtttt
```

BW変換の性質

・例えば、下の文章を変換すると...

```
Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the
industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and
scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap
into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the
release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing
software like Aldus PageMaker including versions of Lorem Ipsum.
```

でも、復元できないと意味ないよね？

→ 圧倒的に同じ文字が並んでいる！！

```
.eeemmmm..f.gfsketn,egsn,eshdyogtyamtdregpmdeeoddestytmsnepee,dtsrgessyyennofhtrfadcyngt,ysyssgsmkdy560
191         $  e   PstmmMgi r    thedwlhwherp mu  ine nen sereenennen u
lnnhhkkpvhhrshrhpcgllrplslvgbhr rrrrrhmescckvtvgippd sLess ttloooo onnnnnnn anattc  ttttttswstnrlcs s
ntnthdtaarr s rl wwfvooiaansuA eebl baaacpnlteeeeuuuue aiuu immeewieaiuiaaaaiiiiiiiiiaoiiuo
kooieieietts    soor ictbtpmLLLLnnao yysyym   IIII oeeectaao  oooo uppateuttuatnai'a0ee0s aisaee i
rel  ae uupppp uxIxoeiInsn  ii      nnttnn  kecsseeenf    plpssssdd  tsdddbiie rt    oeellemmllrrtttt
```

BW変換の性質

- 実は、BW変換後と Suffix Array は

  同じ文字種間で相対順序が保たれています。

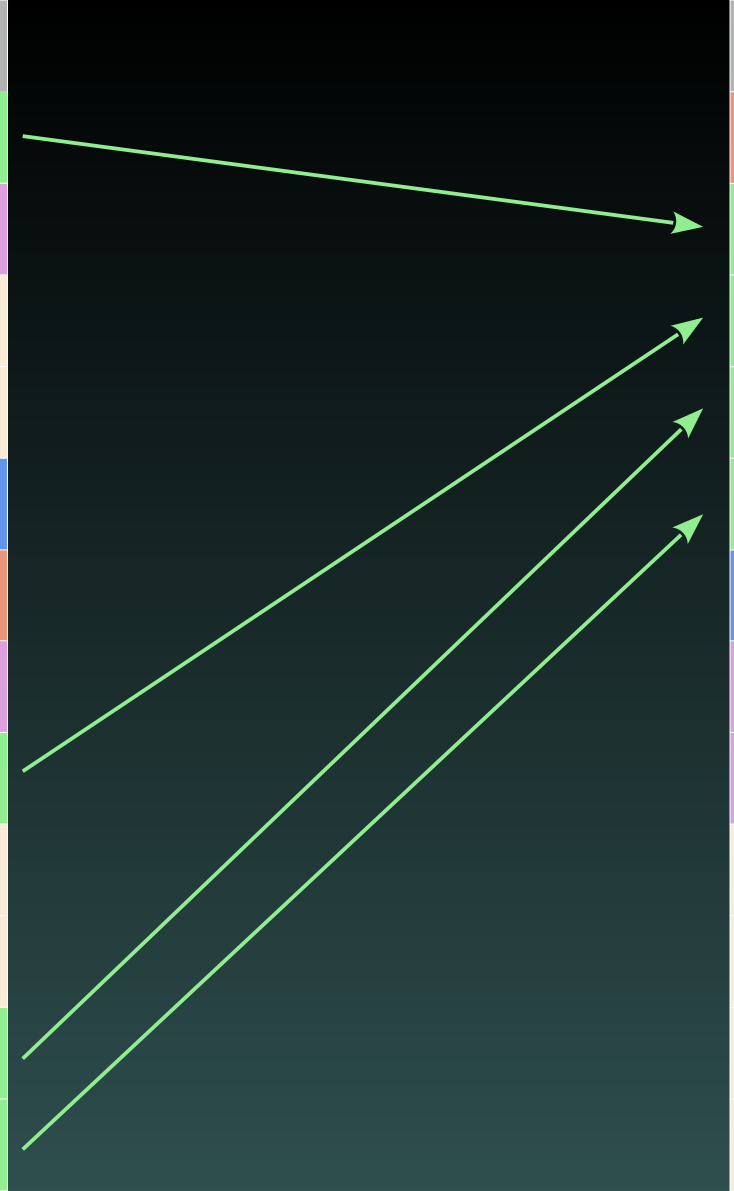
BW変換の性質

- 実は、BW変換後と Suffix Array は

  同じ文字種間で相対順序が保たれています。

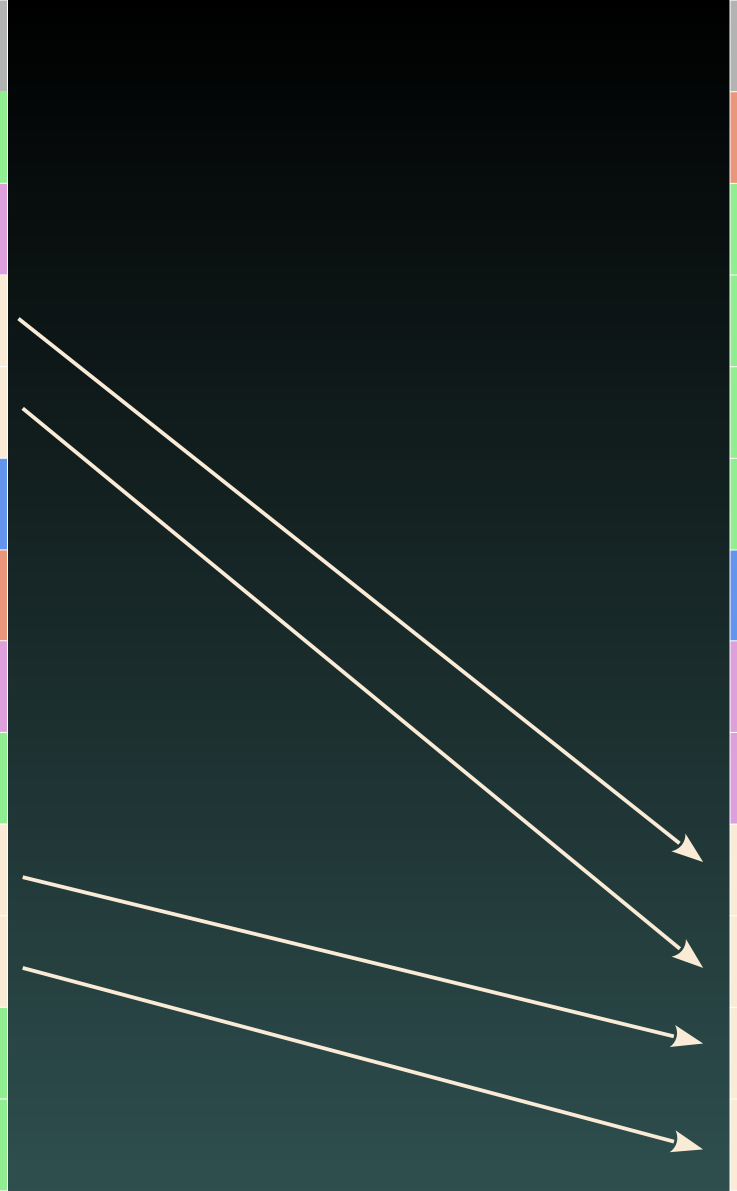
  → 「LF mapping」と呼ぶ

	位置	Suffix
0	10	i$
1	9	pi$
2	6	sippi$
9	3	sissippi$
5	0	mississippi$
0	11	$
7	8	ppi$
2	7	ippi$
10	5	ssippi$
11	2	ssissippi$
3	4	issippi$
4	1	ississippi$



	位置	Suffix
0	11	$
1	10	i$
2	7	ippi$
3	4	issippi$
4	1	ississippi$
5	0	mississippi$
6	9	pi$
7	8	ppi$
8	6	sippi$
9	3	sissippi$
10	5	ssippi$
11	2	ssissippi$

	位置	Suffix
0	10	i$
1	9	pi$
2	6	sippi$
9	3	sissippi$
5	0	mississippi$
0	11	$
7	8	ppi$
2	7	ippi$
10	5	ssippi$
11	2	ssissippi$
3	4	issippi$
4	1	ississippi$



	位置	Suffix
0	11	$
1	10	i$
2	7	ippi$
3	4	issippi$
4	1	ississippi$
5	0	mississippi$
6	9	pi$
7	8	ppi$
8	6	sippi$
9	3	sissippi$
10	5	ssippi$
11	2	ssissippi$

BW変換の性質

- この LF mapping を利用する事で、

  変換後の文字列のみから復元する事ができます。

  （詳細は割愛）

レポートの流れ

- ▷ 1. 研究動機
- ▷ 2. 概要·序論
- ▷ 3. 研究方法
- ▶ 4. 研究結果
  - ▷ 4.1. Suffix Array
  - ▷ 4.2. BW 変換
  - ▶ 4.3. Bit Vector
  - ▷ 4.4. Wavelet Matrix
- ▷ 4. 研究結果
  - ▷ 4.5. FM-index
- ▷ 5. 結論·考察
- ▷ 6. 感想
- ▷ 7. 参考文献

Bit Vector とは？

- bool 型 (true/false)の配列 B に対して、
  以下のような操作を定数時間で処理可能：

Bit Vector とは？

- bool 型 (true/false)の配列 B に対して、
  以下のような操作を定数時間で処理可能：

  - rank(b,i): $j < i$ で $B_j = b$ となる j の個数

  （他にもありますが割愛）

Bit Vector とは？

- bool 型 (true/false)の配列 B に対して、
  以下のような機能を提供：

？？？「B_i = rank(1, i) を前計算するだけじゃダメなんでしょうか？」

  - rank(b,i): j < i で B_j = b となる j の個数を返す

（他にも色々ありますが割愛）

Bit Vector とは？

- bool 型 である事を活かしてメモリを減らそう！

Bit Vector とは？

- bool 型 である事を活かしてメモリを減らそう！

→ 64bit 毎にrankの値を記録 ,
i 番目の値を (i mod 64) bit目に保存

Bit Vector の例

- 簡単のため、4 bit 毎の場合の例

- 以降、popcount(i)= i で立っているbitの個数

	0	1	2	3	4	5	6	7	8
val	0	1	1	0	1	1	0	1	-
rank	-	-	-	-	2	-	-	-	5
bit	-	-	-	-	0110 = 6	-	-	-	1011 = 11

Bit Vector の例

- rank(1,5) と聞かれたら？

	0	1	2	3	4	5	6	7	8
val	0	1	1	0	1	1	0	1	-
rank	-	-	-	-	2	-	-	-	5
bit	-	-	-	-	0110 = 6	-	-	-	1011 = 11

Bit Vector の例

- rank(1,5) と聞かれたら？

→ rank(1,4) + popcount(1011 & 0001) = 3

	0	1	2	3	4	5	6	7	8
val	0	1	1	0	1	1	0	1	-
rank	-	-	-	-	2	-	-	-	5
bit	-	-	-	-	0110 = 6	-	-	-	1011 = 11

Bit Vector の例

- rank(1,7) と聞かれたら？

	0	1	2	3	4	5	6	7	8
val	0	1	1	0	1	1	0	1	-
rank	-	-	-	-	2	-	-	-	5
bit	-	-	-	-	0110 = 6	-	-	-	1011 = 11

Bit Vector の例

- rank(1,7) と聞かれたら？

→ rank(1,4) + popcount(1011 & 0111) = 4

	0	1	2	3	4	5	6	7	8
val	0	1	1	0	1	1	0	1	-
rank	-	-	-	-	2	-	-	-	5
bit	-	-	-	-	0110 = 6	-	-	-	1011 = 11

Bit Vectorの用途

- この後の Wavelet Matrix で使います。

レポートの流れ

- ▷ 1. 研究動機
- ▷ 2. 概要·序論
- ▷ 3. 研究方法
- ▶ 4. 研究結果
  - ▷ 4.1. Suffix Array
  - ▷ 4.2. BW 変換
  - ▷ 4.3. Bit Vector
  - ▶ 4.4. Wavelet Matrix
- ▷ 4. 研究結果
  - ▷ 4.5. FM-index
- ▷ 5. 結論·考察
- ▷ 6. 感想
- ▷ 7. 参考文献

Wavelet Matrixとは？

- 数列 A に対して、以下のような操作を
  O(log σ) で処理可能：

Wavelet Matrixとは？

- 数列 A に対して、以下のような操作を

  O(log σ) で処理可能：

  - rank(x,i) … j < i で $A_j = x$ となる j の個数

Wavelet Matrixとは？

- 数列 A に対して、以下のような操作を

  O(log σ) で処理可能：

  - rank(x,i) … j < i で $A_j = x$ となる j の個数

- 他にもquantileなどありますが、割愛

Wavelet Matrixの仕組み

- 前処理として、先程のBit Vectorを用いた
  空間計算量O(N log σ)の索引を作成します。

Wavelet Matrixの仕組み

- 前処理として、先程のBit Vectorを用いた
  空間計算量O(N log σ)の索引を作成します。

- rankでは、その索引を用いて範囲を絞ります。
  詳細は複雑なので割愛...

Wavelet Matrixの用途

- 次のFM-indexで使います。

- もちろん、FM-index以外にも色々使えるデータ構造です！

レポートの流れ

- ▷ 1. 研究動機
- ▷ 2. 概要·序論
- ▷ 3. 研究方法
- ▷ 4. 研究結果
  - ▷ 4.1. Suffix Array
  - ▷ 4.2. BW 変換
  - ▷ 4.3. Bit Vector
  - ▷ 4.4. Wavelet Matrix
- ▶ 4. 研究結果
  - ▶ 4.5. FM-index
- ▷ 5. 結論·考察
- ▷ 6. 感想
- ▷ 7. 参考文献

FM-indexとは？

・今まで紹介してきたデータ構造・アルゴリズムは

全てこのデータ構造のためのものです（衝撃）

FM-indexとは？

- 今まで紹介してきたデータ構造・アルゴリズムは全てこのデータ構造のためのものです（衝撃）

- 圧縮全文索引と言って、文字列クエリに対する処理を $O(|T| \log \sigma)$ で行えます。

FM-indexとは？

- 今まで紹介してきたデータ構造・アルゴリズムは

  全てこのデータ構造のためのものです（衝撃）

  今度こそ |S| に依存していない！！（歓喜）

- 圧縮全文索引と言って、文字列クエリに対する

  処理を O(|T| log σ) で行えます。

FM-indexの仕組み

- 主なアイデアはBW変換のLF mappingです。

- 文字列を後ろから見ていき、

  範囲がどう移動するかを見ていきます。

FM-indexの仕組み

- BW変換後の文字列は、

  Suffix Arrayの1つ前の文字です。

FM-indexの仕組み

- BW変換後の文字列は、
  Suffix Arrayの1つ前の文字です。

- また、Suffix Arrayは1文字目が辞書順で、
  BW変換後の文字列と相対順序は不変です。

FM-indexの仕組み

- つまり、i文字目の時点でj番目を指していた場合

  i - 1文字目でjは

  ($S_k < S_j$ となるkの個数) + rank(S_j, j)

  に移動します（重要）

FM-indexの仕組み

- これを答えとなる区間 [sp,ep) に当てはめると

  sp,epが遷移して行き、

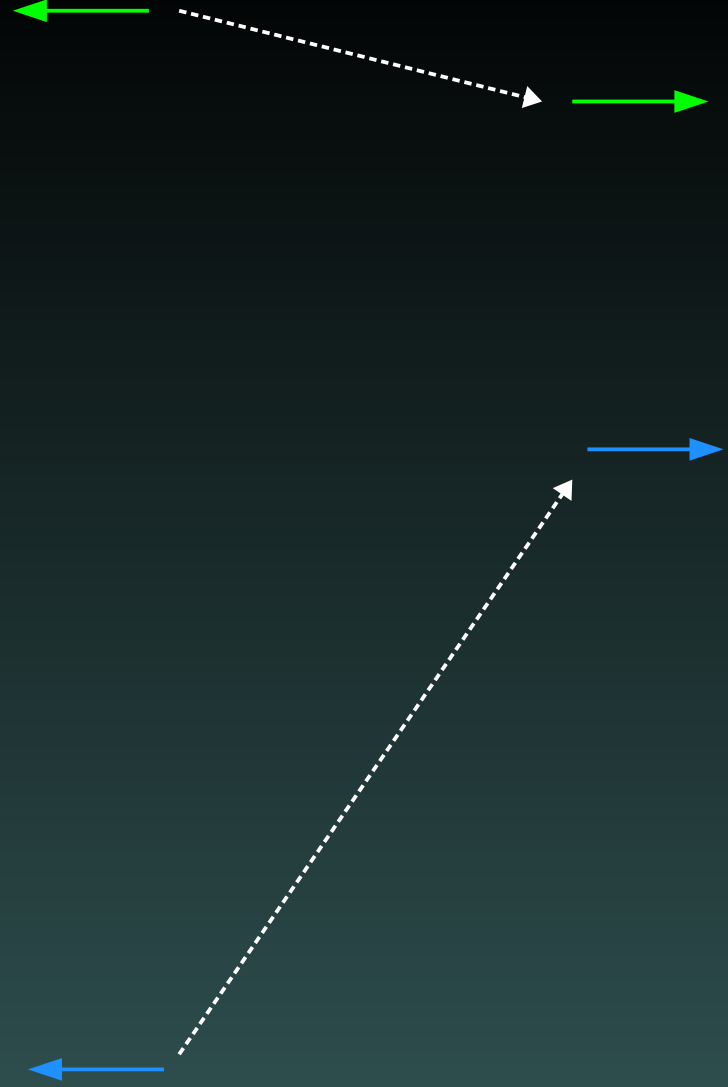
  最終的にSuffix Arrayの該当区間を指します。

FM-indexの例

- 毎度おなじみ”mississippi$” で実験してみます。
- クエリは “issi” です。

	位置	Suffix
0	10	i$
1	9	pi$
2	6	sippi$
9	3	sissippi$
5	0	mississippi$
0	11	$
7	8	ppi$
2	7	ippi$
10	5	ssippi$
11	2	ssissippi$
3	4	issippi$
4	1	ississippi$

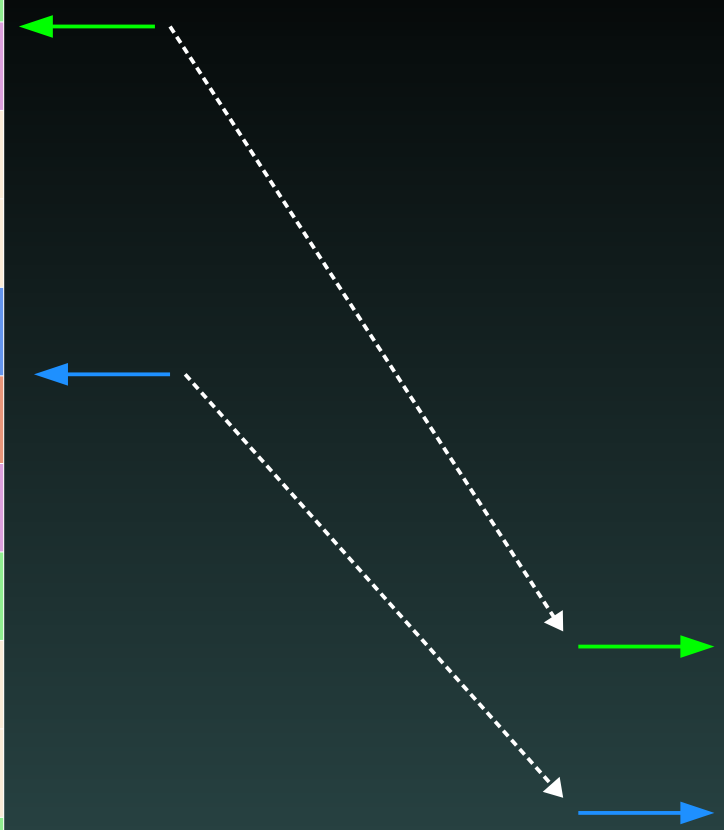
issi



	位置	Suffix
0	11	$
1	10	i$
2	7	ippi$
3	4	issippi$
4	1	ississippi$
5	0	mississippi$
6	9	pi$
7	8	ppi$
8	6	sippi$
9	3	sissippi$
10	5	ssippi$
11	2	ssissippi$

issi

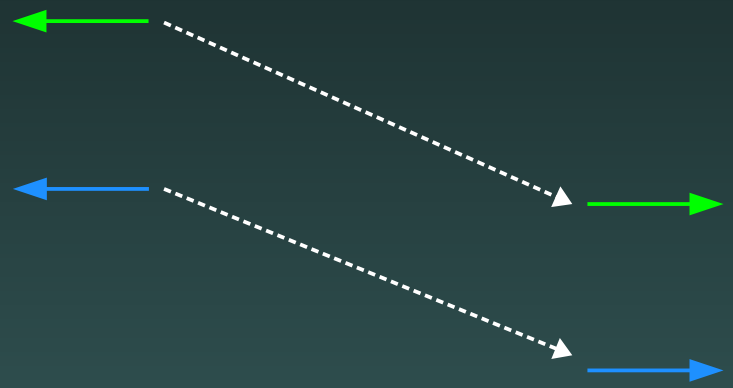
	位置	Suffix
0	10	i$
1	9	pi$
2	6	sippi$
9	3	sissippi$
5	0	mississippi$
0	11	$
7	8	ppi$
2	7	ippi$
10	5	ssippi$
11	2	ssissippi$
3	4	issippi$
4	1	ississippi$



	位置	Suffix
0	11	$
1	10	i$
2	7	ippi$
3	4	issippi$
4	1	ississippi$
5	0	mississippi$
6	9	pi$
7	8	ppi$
8	6	sippi$
9	3	sissippi$
10	5	ssippi$
11	2	ssissippi$

issi

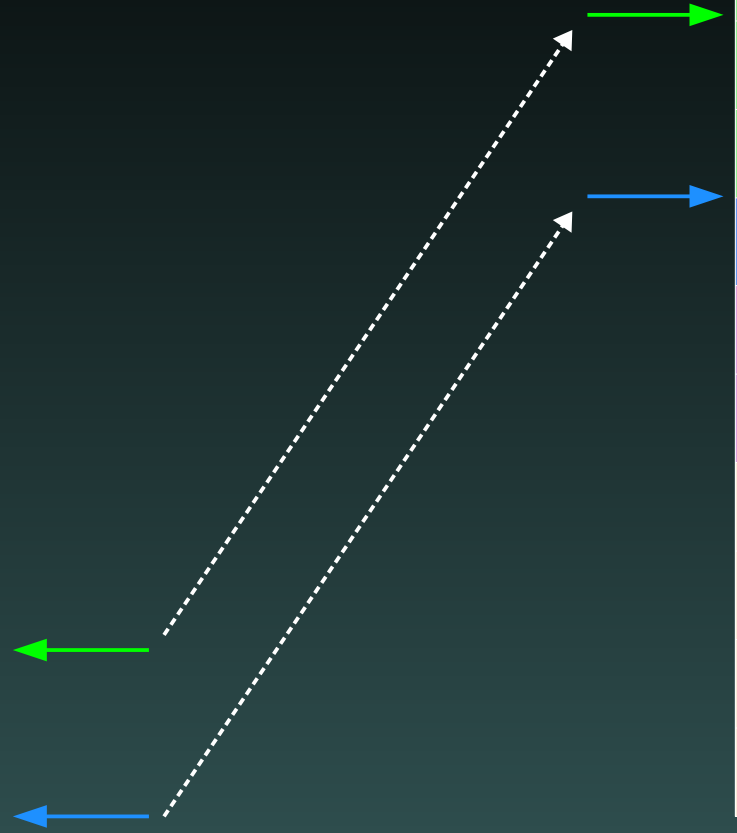
	位置	Suffix
0	10	i$
1	9	pi$
2	6	sippi$
9	3	sissippi$
5	0	mississippi$
0	11	$
7	8	ppi$
2	7	ippi$
10	5	ssippi$
11	2	ssissippi$
3	4	issippi$
4	1	ississippi$



	位置	Suffix
0	11	$
1	10	i$
2	7	ippi$
3	4	issippi$
4	1	ississippi$
5	0	mississippi$
6	9	pi$
7	8	ppi$
8	6	sippi$
9	3	sissippi$
10	5	ssippi$
11	2	ssissippi$

issi

	位置	Suffix
0	10	i$
1	9	pi$
2	6	sippi$
9	3	sissippi$
5	0	mississippi$
0	11	$
7	8	ppi$
2	7	ippi$
10	5	ssippi$
11	2	ssissippi$
3	4	issippi$
4	1	ississippi$



	位置	Suffix
0	11	$
1	10	i$
2	7	ippi$
3	4	issippi$
4	1	ississippi$
5	0	mississippi$
6	9	pi$
7	8	ppi$
8	6	sippi$
9	3	sissippi$
10	5	ssippi$
11	2	ssissippi$

FM-indexの時間計算量

・このように、|T|ステップで範囲を特定できました。

FM-indexの時間計算量

- このように、|T|ステップで範囲を特定できました。
- rank で Wavelet Matrixを使うので、

  log σ がついて O(|T| log σ) です。

FM-indexの空間計算量

- Suffix Arrayがネックとなり

  $O(|S| \log |S|)$ となります。

- 余談：圧縮Suffix Arrayなるものもあるらしい...

レポートの流れ

- ▷ 1. 研究動機
- ▷ 2. 概要·序論
- ▷ 3. 研究方法
- ▷ 4. 研究結果
  - ▷ 4.1. Suffix Array
  - ▷ 4.2. BW 変換
  - ▷ 4.3. Bit Vector
  - ▷ 4.4. Wavelet Matrix
  - ▷ 4.5. FM-index
- ▶ 5. 結論·考察
- ▷ 6. 感想
- ▷ 7. 参考文献

結論

- 目的のクエリに対し、FM-indexを用いる事で
  - 構築 時間計算量 $O(|S|)$
  - 1件あたり 時間計算量 $O(|T| \log \sigma)$
  - 空間計算量 $O(|S| \log |S|)$

  で処理できました。

結論

・せっかくなので、実装もしてみました（300行）。

結論

- せっかくなので、実装もしてみました（300行）。
- $|S| \leqq 10^6$, クエリ件数 10000件 ,
  $|T| \leqq 10^4$ を合計僅か 0.5秒で処理できています。
- AOJの問題の提出コード

考察

- AOJの悪意あるケースですら、FM-indexとSuffix Arrayは50msしか変わりませんでした。

- |S| が余程長くなければ、実用的には定数倍でSuffix Arrayの全文検索の方が高速なのでは？

レポートの流れ

- ▷ 1. 研究動機
- ▷ 2. 概要·序論
- ▷ 3. 研究方法
- ▷ 4. 研究結果
  - ▷ 4.1. Suffix Array
  - ▷ 4.2. BW 変換
  - ▷ 4.3. Bit Vector
  - ▷ 4.4. Wavelet Matrix
- ▷ 4. 研究結果
  - ▷ 4.5. FM-index
- ▷ 5. 結論·考察
- ▶ 6. 感想
- ▷ 7. 参考文献

感想

- 実装は結構大変でしたが、面白いアルゴリズムを知る事ができて楽しかったです。
- 今回は S が不変でしたが、Sに操作クエリが加わる場合のアルゴリズムも勉強してみたいです。

レポートの流れ

- ▷ 1. 研究動機
- ▷ 2. 概要·序論
- ▷ 3. 研究方法
- ▷ 4. 研究結果
  - ▷ 4.1. Suffix Array
  - ▷ 4.2. BW 変換
  - ▷ 4.3. Bit Vector
  - ▷ 4.4. Wavelet Matrix
- ▷ 4. 研究結果
  - ▷ 4.5. FM-index
- ▷ 5. 結論·考察
- ▷ 6. 感想
- ▶ 7. 参考文献

参考文献

- 高速文字列解析の世界 (著：岡野原 大輔)

おしまい