

平成31年度 知能情報工学実験演習2

# 離散構造モデリング

## 第3回演習資料

坂本比呂志 平田耕一 芳野拓也

情報工学部 知能情報工学科

# これまでのおさらい(1回目): ランレングス圧縮

run-length圧縮:

同一文字の連続(run)をその長さ(length)で置き換える圧縮法

文字コードの読み替えによって, runは  
次のように置き換え可能

AAAAAAAAAA = A9

AAAAAAAAABBBBBB = A9B5

AAAAAAAAABBBBBBCCCCC = A9B5C6

汎用的なrun-length圧縮: **長いrunがないデータでもうまく圧縮したい**

# これまでのおさらい(2回目): 文字列のソート

ポインタ配列を用いた文字列のソート

- ・ポインタ配列が文字列を指している
- ・ポインタが指している文字列を比較してポインタを入れ替える

0	1	2	3	4	5	6	7	8
0(4)	1	2	3	4	5	6	7	8
↓	↓	↓	↓	↓	↓	↓	↓	↓
B	A	B	C	A	B	A	B	C
B	B	B	A	¥0	B	B	B	A
C	¥0	A	B		B	C	¥0	A
¥0		B	C		¥0	¥0		¥0
		¥0	¥0					

## 5. BWTによる文字列の変換

# ソートを利用した圧縮(今回のテーマ)

BWTによって得られる文字列は同じ文字が連続しやすい

- ・例えば以下のように変換される

$S = \text{bababbababbababba}$



$\text{BWT}(S) = \text{bbbbbbbaabbaaaaaa}$

- ・このとき $\text{BWT}(S)$ は $\text{BWT}(S)' = \text{b8a1b3a1b1a6}$  と等価であり BWTは逆変換可能である

- ・ $|S| = |\text{BWT}(S)| = 20 \text{ bytes}$ ,  $|\text{BWT}(S)'| = 12 \text{ bytes}$  であるので  $S$  が圧縮されている

- ・BWTによるランレングス圧縮の改良が今回の演習テーマである

# 今回のテーマ:

## BWT (Burrows-Wheeler Transform)

**BWT**とは文字列**S**を以下の手続きで変換すること

- $S = \text{“BANANA”}$  の場合
- $S$  を左に巡回シフトした文字列  $S'$  をすべて求める
- $S'$  をソートして末尾の文字を連結したものが  $S$  の  $\text{BWT}(S)$

0:	B	A	N	A	N	A
1:	A	N	A	N	A	B
2:	N	A	N	A	B	A
3:	A	N	A	B	A	N
4:	N	A	B	A	N	A
5:	A	B	A	N	A	N

$S$ の巡回シフト行列

5:	A	B	A	N	A	N
3:	A	N	A	B	A	N
1:	A	N	A	N	A	B
0:	B	A	N	A	N	A
4:	N	A	B	A	N	A
2:	N	A	N	A	B	A

ソート後

$\text{BWT}(S) =$   
NNBAAA

# BWTの性質: 以下の性質によってBWTは圧縮に利用される

- BWT(S)はSをある規則で並べ替えたもの(Sの順列のひとつ)
- BWT(S)は同じ文字が**きわめて**連続しやすい      S: BANANA  
BWT(S): NNBAAA
- BWT(S)からSへ逆変換が可能

青い文字列だけから  
元の文字列を復元できる


0:	B	A	N	A	N	A
1:	A	N	A	N	A	B
2:	N	A	N	A	B	A
3:	A	N	A	B	A	N
4:	N	A	B	A	N	A
5:	A	B	A	N	A	N

5:	A	B	A	N	A	N
3:	A	N	A	B	A	N
1:	A	N	A	N	A	B
0:	B	A	N	A	N	A
4:	N	A	B	A	N	A
2:	N	A	N	A	B	A

# ポインタ配列のソートによる BWTの計算

巡回シフトした文字列のソートは  
以下のようにポインタ配列のソート  
によって計算できる  
(前回のプログラムを利用)

0:	B	A	N	A	N	A
1:	A	N	A	N	A	B
2:	N	A	N	A	B	A
3:	A	N	A	B	A	N
4:	N	A	B	A	N	A
5:	A	B	A	N	A	N



5:	A	B	A	N	A	N
3:	A	N	A	B	A	N
1:	A	N	A	N	A	B
0:	B	A	N	A	N	A
4:	N	A	B	A	N	A
2:	N	A	N	A	B	A

0	→	B	A	N	A	N	A
1	→	A	N	A	N	A	B
2	→	N	A	N	A	B	A
3	→	A	N	A	B	A	N
4	→	N	A	B	A	N	A
5	→	A	B	A	N	A	N

文字列の  
ソート



5	↗	B	A	N	A	N	A
3	↘	A	N	A	N	A	B
1	↖	N	A	N	A	B	A
0	↗	A	N	A	B	A	N
4	→	N	A	B	A	N	A
2	↘	A	B	A	N	A	N



# 手順

## 1. 文字列SからSSを作成

strcat(s\_1,s\_2)を用いる(s\_1の末尾にs\_2を連結する)

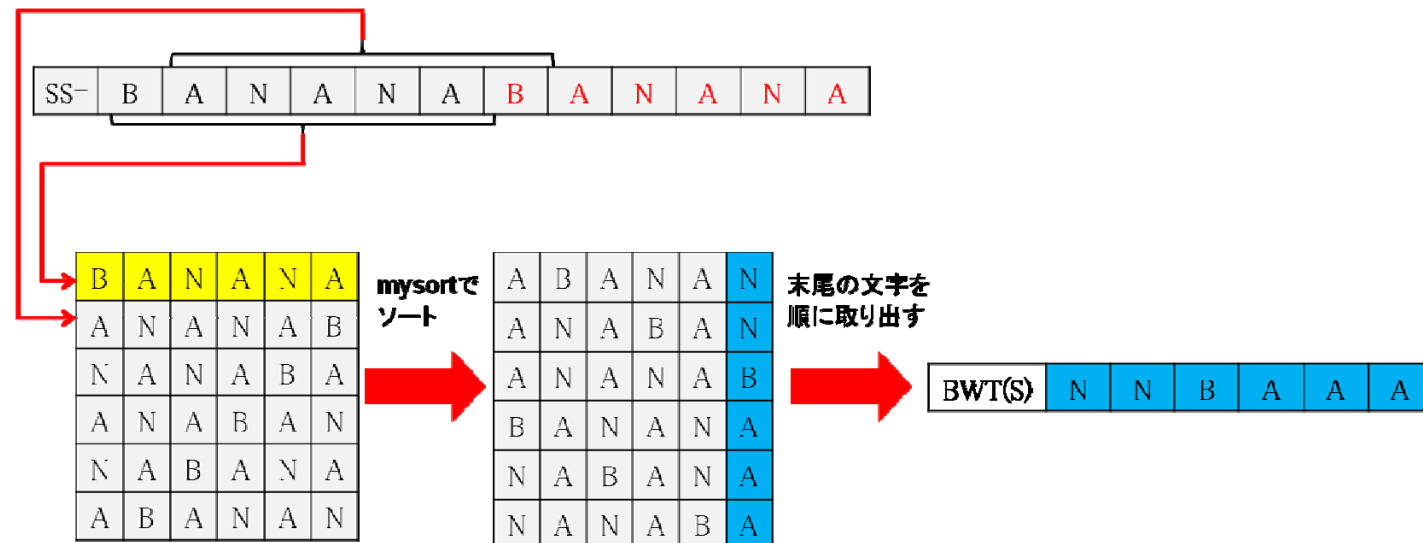
※注1: 自分自身をコピーするからといってstrcat(s,s)などとしてはならない.

※注2: まずsを作業用配列wにコピー(strcpy(w,s)を使う)してstrcat(w,s)とする.

## 2. 文字列SSを用いて巡回文字列を生成

## 3. 先週の演習で作成したソートプログラムを用いてソートする

## 4. ソートした後の文字列の末尾のみを出力 →BWT(S)



## 課題5

入力ファイル(Test5.txt)の文字列全体をSとする. BWT(S)を計算し, 結果をファイルに書きだすプログラムを実装せよ. 文字列ソートのプログラムは前回の課題プログラムを利用すればよい. また, BWTの復元プログラムを用いてBWTの変換が正しく元に戻ることを確認せよ.

デコーダは”make decode”とコマンド入力すると作成される.

※前回自分で作成した文字列のソートプログラムmysort.cをコピーしてbsort.cという名前で保存し, 今回の課題で利用すること(Makefileではbsort.cを読み込むことになっている)

# 課題5の提出方法

提出するもの

- Makefile
- func.h
- main.c
- check.c
- bsort.c
- Test5.txt (オリジナルデータ)
- Test5.bwt (Test5.txtをBWTした結果)

これらをフォルダ“kadai5\_学籍番号”にまとめて、ZIPで圧縮

(アドレス) [experiment@donald.ai.kyutech.ac.jp](mailto:experiment@donald.ai.kyutech.ac.jp)

(件名) kadai5\_学籍番号

# 原始的なBWTの問題点： 領域コスト

巡回シフト行列を作った場合に必要となるメモリは？

・ $|S| = 1\text{MB}$  の場合：

1MB = 1,000,000 文字

・**行列のサイズ = 1M × 1M Bytes = 1TB !!**

・よって実際にはこの行列は作れないのでどうするか(今回の課題)

0:	B	A	N	A	N	A
1:	A	N	A	N	A	B
2:	N	A	N	A	B	A
3:	A	N	A	B	A	N
4:	N	A	B	A	N	A
5:	A	B	A	N	A	N

Sの巡回シフト行列

5:	A	B	A	N	A	N
3:	A	N	A	B	A	N
1:	A	N	A	N	A	B
0:	B	A	N	A	N	A
4:	N	A	B	A	N	A
2:	N	A	N	A	B	A

ソート後

# 領域コストの解決法: 巡回文字列のポインタ表現

- ・S = BANANAをコピーして連結した文字列W=SSを考える

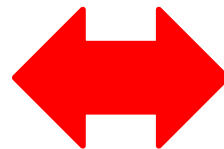
	0	1	2	3	4	5	6	7	8	9	10	11
W=	B	A	N	A	N	A	B	A	N	A	N	A

- ・Sの最初の巡回文字列はWの先頭から $|S| = 6$ 文字までの文字列  
→ これは (W, 0, 5) と表現可能

- ・次の巡回文字列はW[1]からW[6]なので (W, 1, 6)

0	→	B	A	N	A	N	A
1	→	A	N	A	N	A	B
2	→	N	A	N	A	B	A
3	→	A	N	A	B	A	N
4	→	N	A	B	A	N	A
5	→	A	B	A	N	A	N

同じ表現



0	→	0,5
1	→	1,6
2	→	2,7
3	→	3,8
4	→	4,9
5	→	5,10

+ W

## 課題6

領域コストの解決法を踏まえて、大きな入力ファイル(Test6.txt)の文字列全体をSとするBWT(S)を計算し、結果をファイルに書きだすプログラムを実装せよ。また、BWTの復元プログラムを用いてBWTの変換が正しく元に戻ることを確認せよ。

※課題6を提出すれば、課題5は提出しなくてもよいものとする。

ポイント:

- ・文字列SSを一本だけ保持し、ポインタで管理
- ・SSと開始位置で比較する文字列が与えられるので、前回の文字列のソートと同様にその大小比較をstrcmpまたはstrncmpで行う
- ・strcmp(s1,s2):s1とs2の文字列を比較する
- ・strncmp(s1,s2,n): s1とs1の文字列を先頭からn文字分比較する

# 課題6の提出方法

提出するもの

- Makefile
- func.h
- main.c
- check.c
- bsort.c
- Test6.txt (オリジナルデータ)
- Test6.bwt (Test6.txtをBWTした結果)

これらをフォルダ“kadai6\_学籍番号”にまとめて、ZIPで圧縮

(アドレス) [experiment@donald.ai.kyutech.ac.jp](mailto:experiment@donald.ai.kyutech.ac.jp)

(件名) kadai6\_学籍番号

# 発展的内容: BWTの復元(1)

復元前に与えられる情報は

(1)  $BWT(S) = NNBA\textcolor{red}{A}A$

(2) 巡回シフトにおけるSの順位(この場合は4)

A	B	A	N	A	<b>N</b>
A	N	A	B	A	<b>N</b>
A	N	A	N	A	<b>B</b>
B	A	N	A	N	<b><span style="color: red;">A</span></b>
N	A	B	A	N	<b>A</b>
N	A	N	A	B	<b>A</b>



巡回シフトでオリジナルのBANANA  
はこの位置(=4)

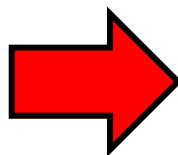
この部分の情報はわからない



## 補足: BWTの復元(2)

					N
					N
					B
					A
					A
					A

復元開始



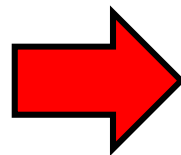
A					N
A					N
A					B
B					A
N					A
N					A

文字列をソートして  
BWTをつくったので  
1列目は最後の列を  
ソートしたもの

## 補足: BWTの復元(3)

A1					N1
A2					N2
A3					B1
B1					A1
N1					A2
N2					A3

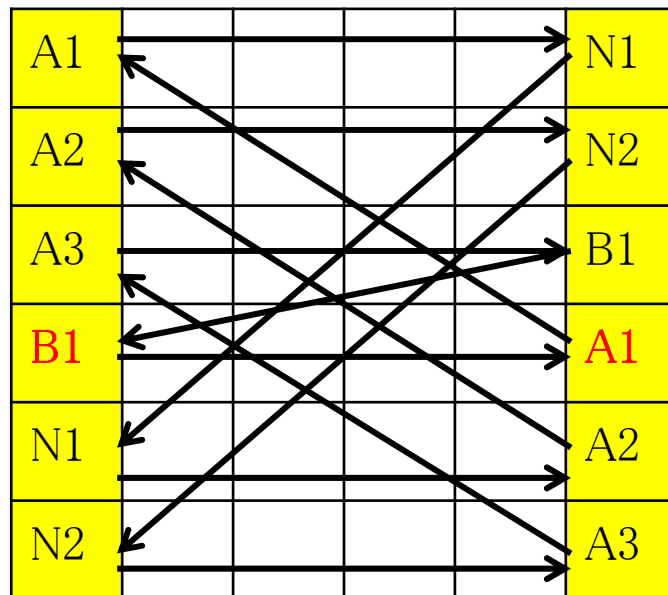
各列の同一文字に  
上から順位を付ける



A1					N1
A2					N2
A3					B1
B1					A1
N1					A2
N2					A3

先頭文字(この場合はB)からはじめて  
二つの列のリンクをたどっていく

## 補足: BWTの復元(4)



1列目に含まれる文字を  
リンクをたどりながら逆順で取り出す

BANANA

すべてのリンクをたどり終わると  
最初に帰ってくる