接尾辞配列（続き）
&
接尾辞木・配列の応用

渋谷

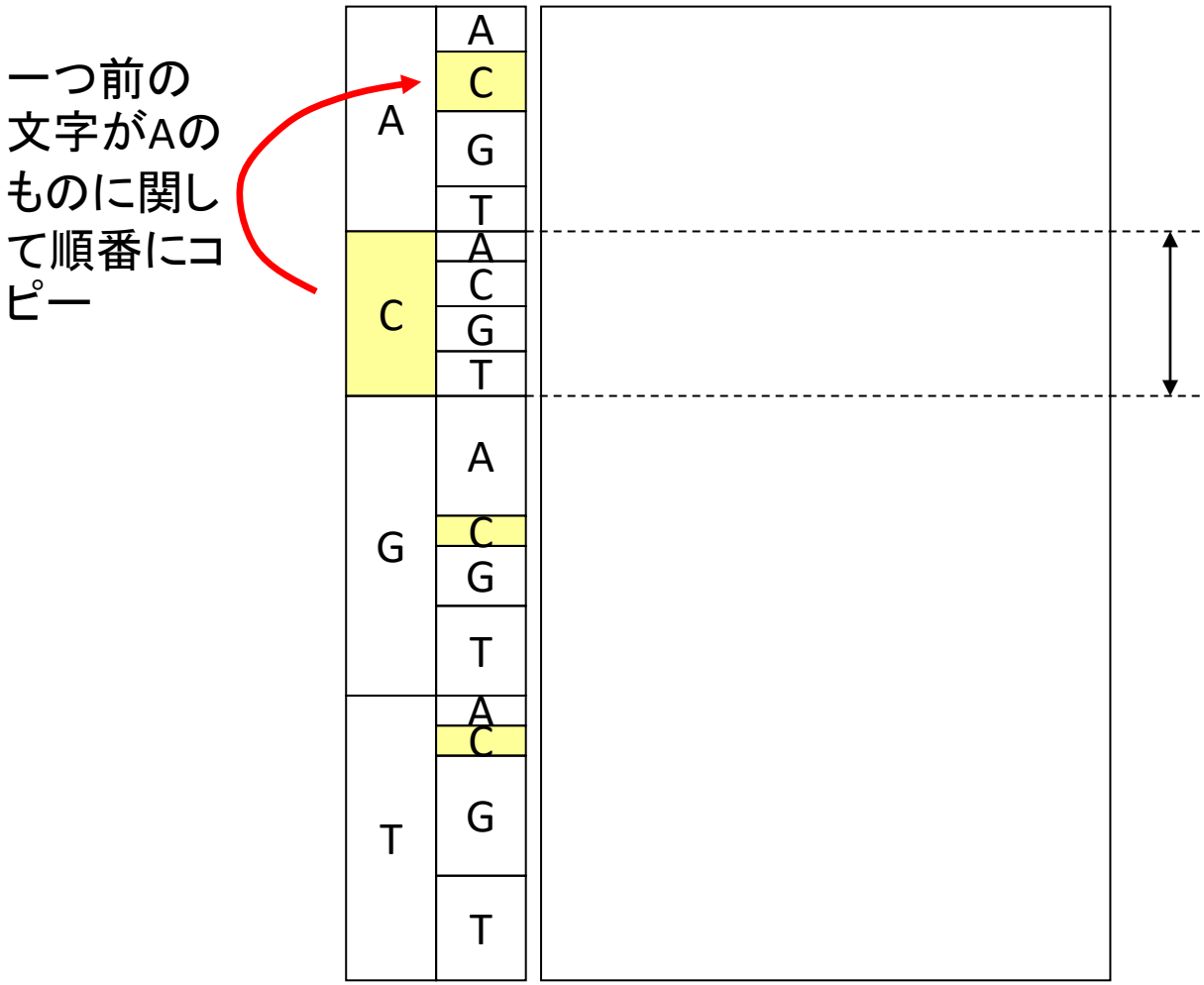
東京大学医科学研究所ヒトゲノム解析センター
（兼）情報理工学系研究科コンピュータ科学専攻
http://www.hgc.jp/~tshibuya

本日の話題

- 接尾辞配列（続き）

Seward "Copy" Algorithm '00（概略）

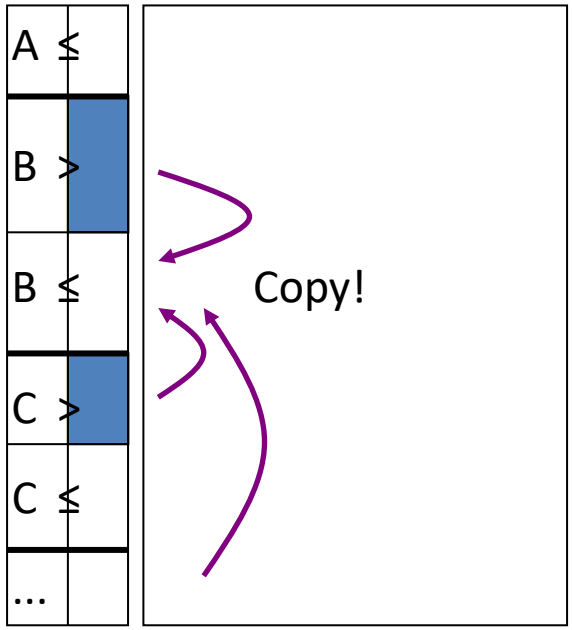
- SA中に似た並びがあることを利用！
  - 平均LCP長が短い場合は結構速い
  - 必要メモリ量が小さい



1. まずは2文字だけでソート
2. 一番少ない文字を探す（Cだとする）
3. まずはCで始まるsuffixをすべてソートする。ただし、CCで始まるsuffixのソートはCA,CG,CTの結果を用いれば計算できる
4. ＊Cで始まるものに計算結果をコピーする
5. 次に少ないものを選んで、繰り返す（但し、もう計算できているものは計算しない）

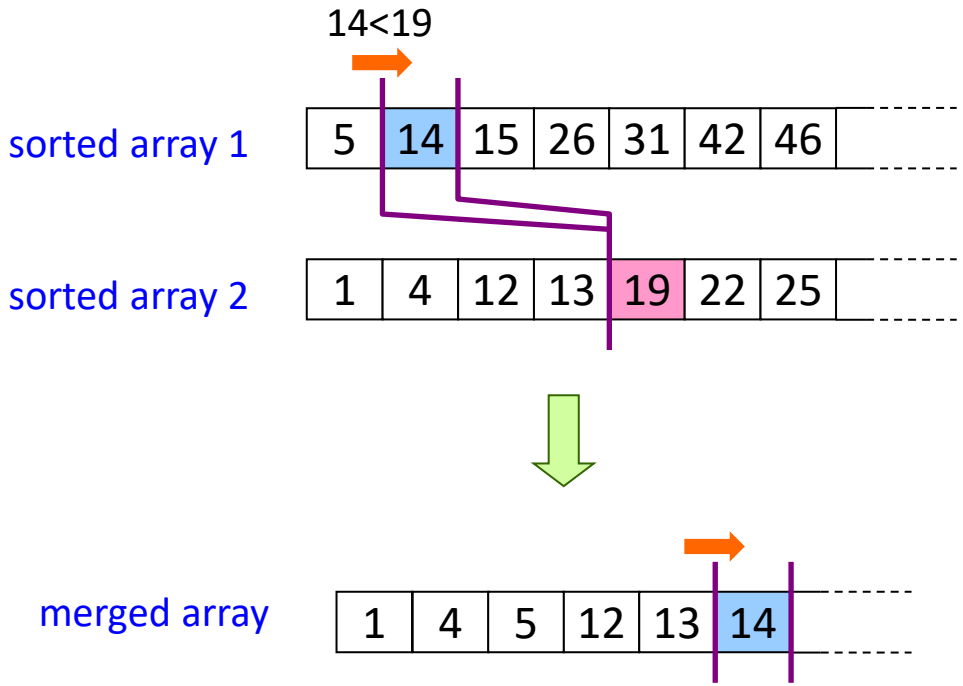
Itoh-Tanaka Algorithm '99

- 同様のCopy系アルゴリズム
  - 後の$O(n)$のKo-Aluru AlgorithmやInduced Sortingのアイディアの種となったアルゴリズム
- アルゴリズム
  - suffix を2つにわける
    - type 1：1文字目>2文字目　（BA....など）
    - type 2：1文字目≤2文字目　（AA... やAB...など）
  - Type 1を普通にソート（Ternary quick sort）
  - その結果を用いて全体をソート



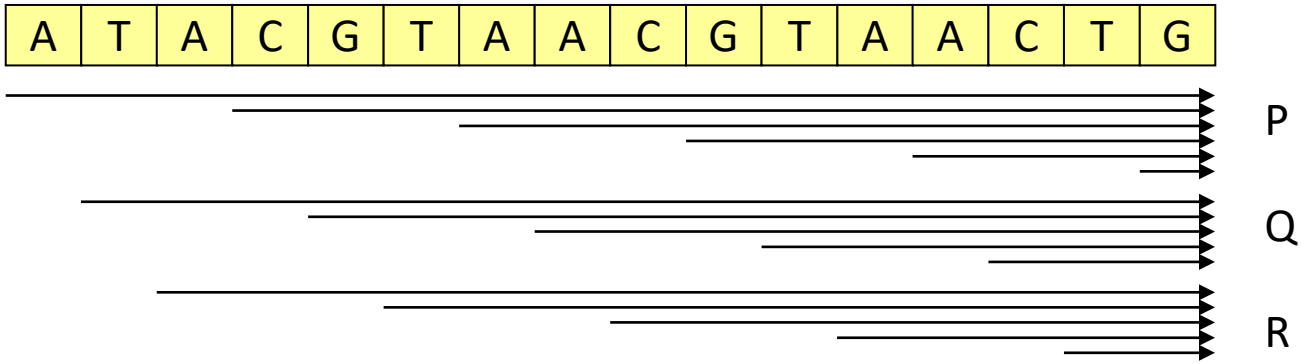
復習：Merge Sort

- 2つのソート済配列の併合をするには？
  - 2つの配列を端から順番に見て小さい方を出力
  - **数字の比較は$O(1)$でできる**ので全体で線形時間
    - ここが重要
    - 接尾辞は数字ではないので、接尾辞配列のマージは極めて難しい！
  - 数列のソートの場合には、マージを再帰的に適用することで（最悪でも）$O(n \log n)$で全体をソートできる



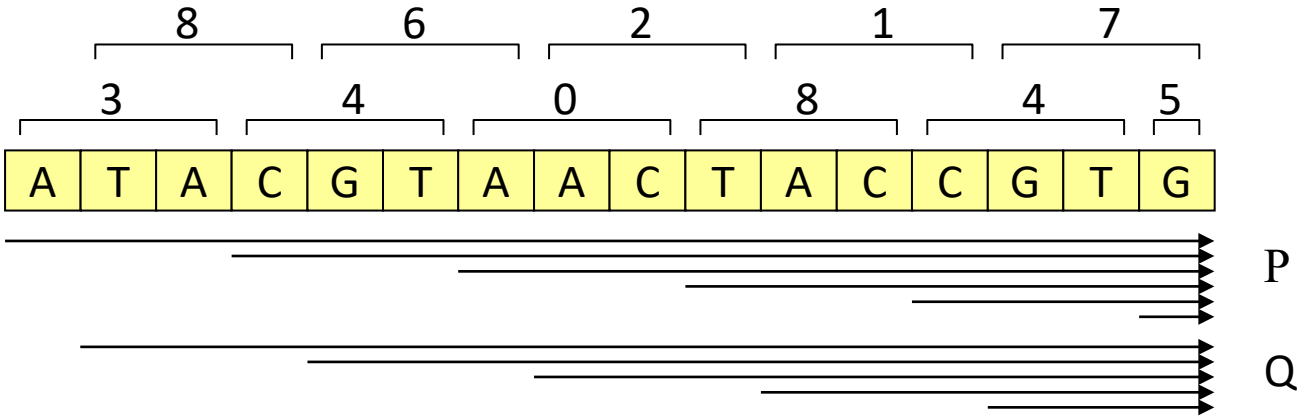
Kärkkäinen & Sanders '03 (1)

- Farachの接尾辞木のアルゴリズムを洗練させたアルゴリズム
- 接尾辞を3種類に分ける
  - それぞれ$3i$, $3i+1$, $3i+2$ 番目から始まる接尾辞集合をそれぞれP, Q, Rとする
  - 集合P+Qに含まれる接尾辞のみからなる接尾辞配列を作成する
    - $f(2n/3)$ 時間で再帰的に計算することが可能
      - $f(n)$: 全体の計算時間
  - その接尾辞配列を用いて接尾辞集合Rに含まれる接尾辞のみからなる接尾辞配列を作成する
    - $O(n)$ 時間
  - 2つの配列をマージする（逆接尾辞配列をうまく用いる）
    - $O(n)$ 時間



Kärkkäinen & Sanders '03 (2)

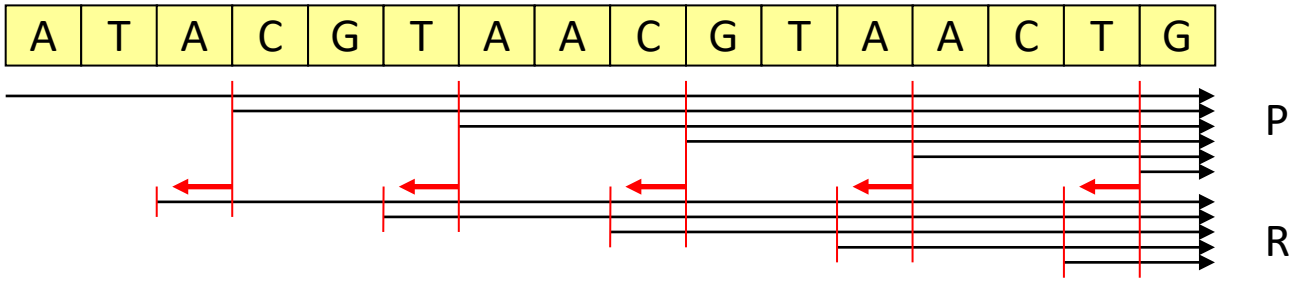
- 接尾辞集合P＋Qに対する接尾辞配列の作り方
  - P+Qに含まれる各接尾辞の先頭3文字の組をまずソート
    - 基数ソートで線形時間
  - ソート順の番号に3文字の組を変更したP,Qに対応する文字列それぞれ作る
    - 2つの文字列を連結し（間に終端記号をはさむ）、それに対して接尾辞配列を計算する（このアルゴリズム全体を再帰的に適用）
    - 結果を用いてP＋Qに対する接尾辞配列を作成（自明）



340845$86217に対する接尾辞配列を作成する

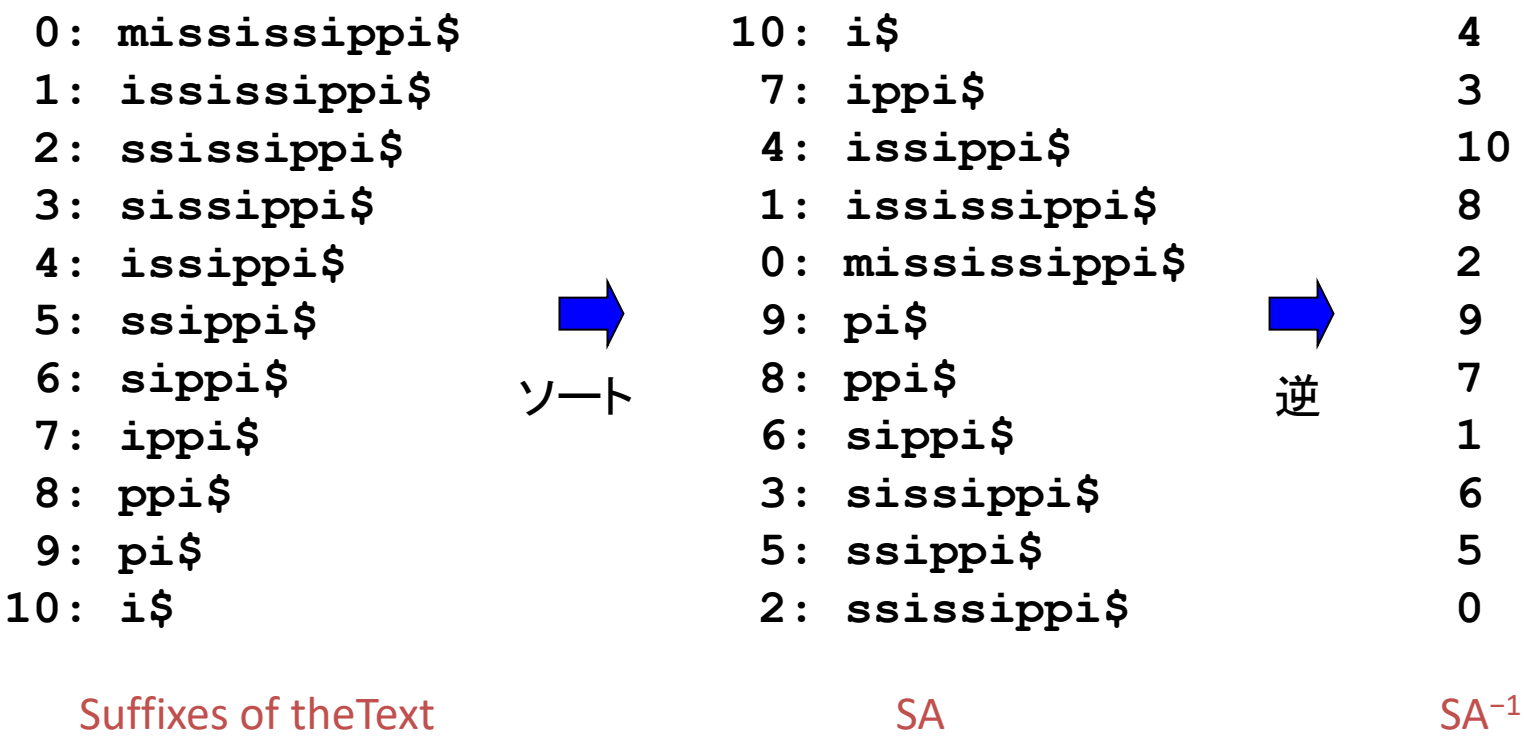
Kärkkäinen & Sanders '03 (3)

- 部分接尾辞集合Rに対する接尾辞配列の作り方
  - Pに対する接尾辞配列から基数ソート一回で作ることができる！（線形時間）
  - PはP+Qの一部なので、先に作成したP+Qに対する接尾辞配列の中からPに対応するindexのみ取り出せばPに対する接尾辞配列が自明に作成可能（線形時間）



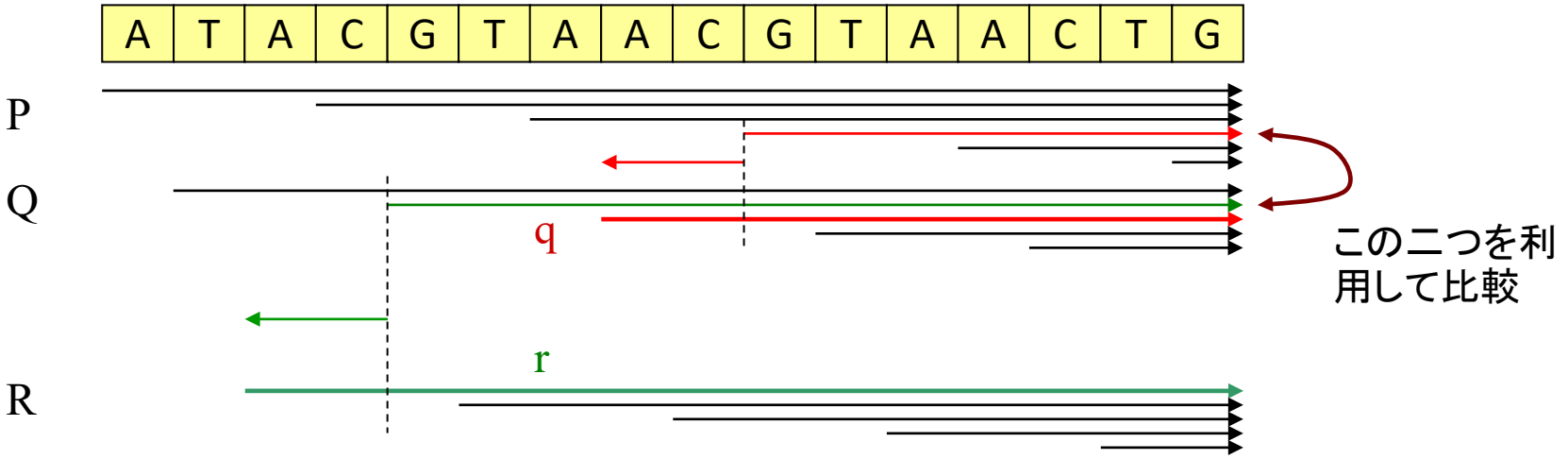
Kärkkäinen & Sanders '03 (4)

- 逆接尾辞配列
  - $\mathrm{SA}^{-1}[i] = j \leftrightarrow \mathrm{SA}[j] = i$
  - 様々な他のアルゴリズムでも使用されるデータ構造
  - 線形時間で作成可能



Kärkkäinen & Sanders '03 (5)

- 2つのマージをするには？
  - 接尾辞同士の比較が $O(1)$ でできれば、線形時間のマージができる（マージソートと同じ）
  - 逆接尾辞配列を用いる
    - 逆接尾辞配列を用いれば、P+Qに入っている接尾辞同士が$O(1)$で比較できる
      - それに加えてその直前1文字または2文字を比較すれば、P，R間あるいはP，Q間の比較ができる



Kärkkäinen & Sanders '03 (6)

- 計算時間
  - $f(n) = f(2n/3) + O(n)$
  - これは $O(n)$
- 一般的に $f(n)=c_1 \cdot f(c_2 \cdot n)+O(n)$ $(0<c_2<1,\ 0<c=c_1c_2)$の時、
  - $0<c<1$ ならば $f(n) = O(n)$
  - $c=1$ ならば $f(n) = O(n \log n)$
  - $c>1$ ならば $f(n) = O(n^{-\log_{c_2} c_1})$

$$f(n) < c_1 \cdot f(c_2 \cdot n) + a \cdot n < {c_1}^2 \cdot f({c_2}^2 n) + a(1+c)n < \cdots < \underbrace{{c_1}^{\log n} \cdot f(\text{const})}_{O(n)} + \underbrace{a\left(1 + c + c^2 + \cdots + c^{\log n}\right)n}_{0<c<1 \text{ ならば } f(n) = O(n)}$$

原論文: Kärkkäinen, J., and Sanders, P. (2003) "Simple Linear Work Suffix Array Construction", *Proc. ICALP, LNCS 2719*, pp. 943-955.

Burkhardt & Kärkkäinen '03 (1)

- KSの（理論的）省メモリ化
  - $o(n)$ の追加空間計算量で $O(n \log n)$ 時間としたもの
- Difference Coverというものを使う
  - $[0..n-1]$ に対し $O(n^{1/2})$ の大きさのカバーを $O(n^{1/2})$ 時間で求めることができる　（より厳密には[Colbourn&Ling, '00]）

[0..99] のdifference coverの例

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90

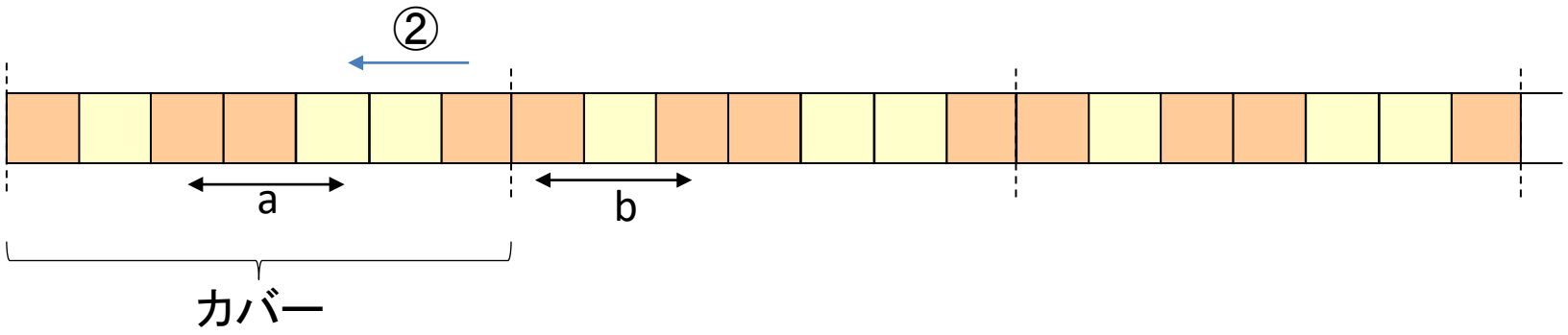
（これが簡単だがもう少しだけ賢いカバーも存在する）

$0 \sim 99$ を $(d_i - d_j) \bmod 100$ で表現できる

cf. 素数ものさし
577円（消費税込で623円）
@京大生協

Burkhardt & Kärkkäinen '03 (2)

- より大きな範囲でK&Sと同様の計算を行う



①  に対する接尾辞配列を作成する

②  に対する接尾辞配列をradix sortで作成する

③ それらをマージする

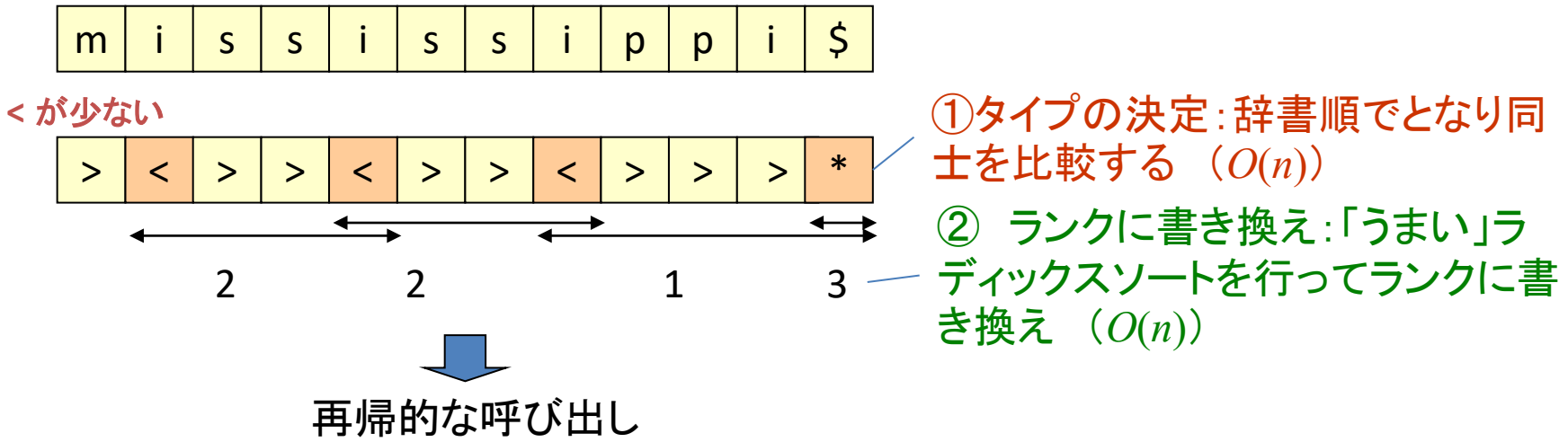
たとえば、aの比較は、先頭5文字の比較＋bの比較で行えば
O(カバーのサイズ)で計算可能

Burkhardt & Kärkkäinen '03 (3)

- Difference cover に入っている剰余に対応する接尾辞だけからなる接尾辞配列を作成
- 入っていないものに関してはそれぞれradix sortで作成可能
  - radix sortの回数はcoverの大きさに比例
- ウィンドウサイズがkならば、
  - この時カバーのサイズは $c \cdot k^{1/2}$
  - k回の比較演算で、どの違う2つの剰余に対応する接尾辞の辞書順比較が可能
  - すなわち $f(n) = f(c \cdot n/k^{1/2}) + O(kn)$
  - $k = \log n$ とすると $f(n) = O(n \log n)$
  - K&Sは $k=3$ としたアルゴリズムに相当
    - $\{0,1\}$は$\{0,1,2\}$のカバー

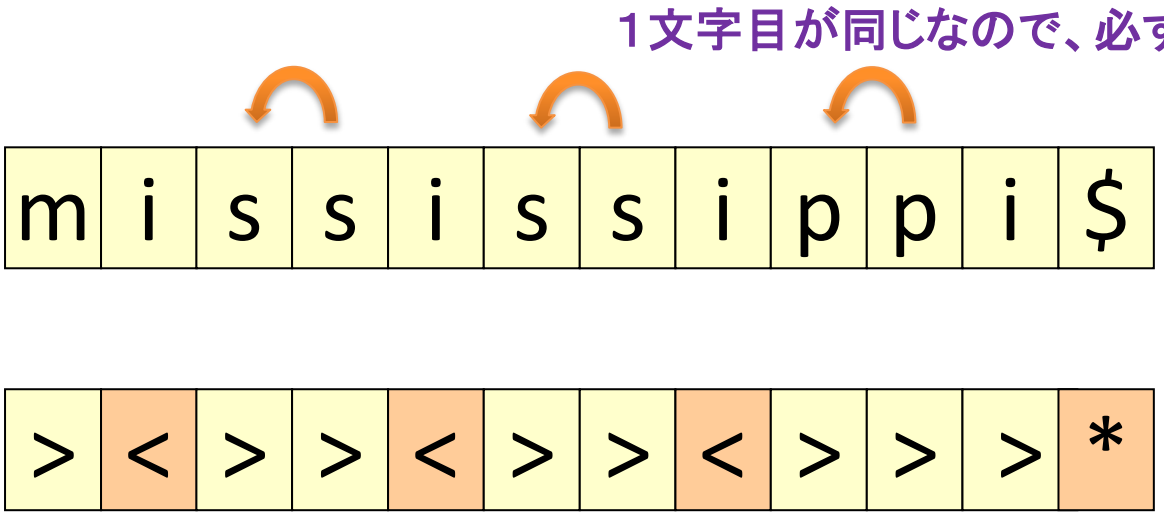
Ko and Aluru '03 (1)

- K&Sと同様のDivide and Merge 系アルゴリズム
  - となりの接尾辞との辞書順の大小で接尾辞を「＞」と「＜」の2つのタイプに分割し、それらのうち数の少ない方に対して接尾辞配列を作成
    - Itoh−Tanaka algorithmのアイディアにも類似
    - 「うまく」ラディックスソートでランキングすることで小さい問題に変換
  - その結果を利用して、計算していない接尾辞も同様の「うまい」ラディックスソートで計算
  - マージも容易
    - 同じ文字で始まる「>」タイプの接尾辞は「<」タイプの接尾辞より必ず辞書順で小さい　＝　そこから先は$O(1)$で比較可能



Ko and Aluru '03 (2)

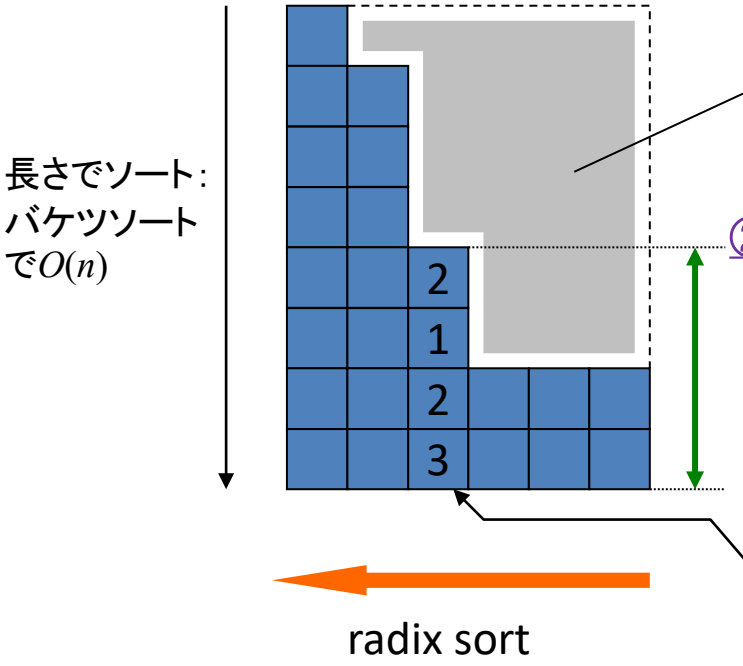
- 「<」タイプと「>」タイプの線形時間決定法
  - 後ろから見ていくだけ
    - 一つ後ろの接尾辞と先頭１文字だけチェックし、先頭文字が異なる場合
      - その大小で「＞」タイプか「＜」タイプかがO(1)でわかる
    - 先頭文字が同じ場合
      - 一つ後ろの接尾辞と同じタイプなので、やはりそれ以上チェックする必要はない！



Ko and Aluru '03 (3)

「うまい」radix sort　（このルーチンを2箇所で使う）

”<”文字列のソート



最後の文字は "<" タイプなので、そこまで同じ文字列は
ソート列の後ろに来ることになる
- 例えば "isp" > "ispi"
- ">"文字列のソートでは逆

②それから「かしこい」ラディックスソート

この幅がwならば$O(w)$で一文字分のソートが可能
＝全体で $O(N)$ でソートが可能

①まず「かしこく」ランキング

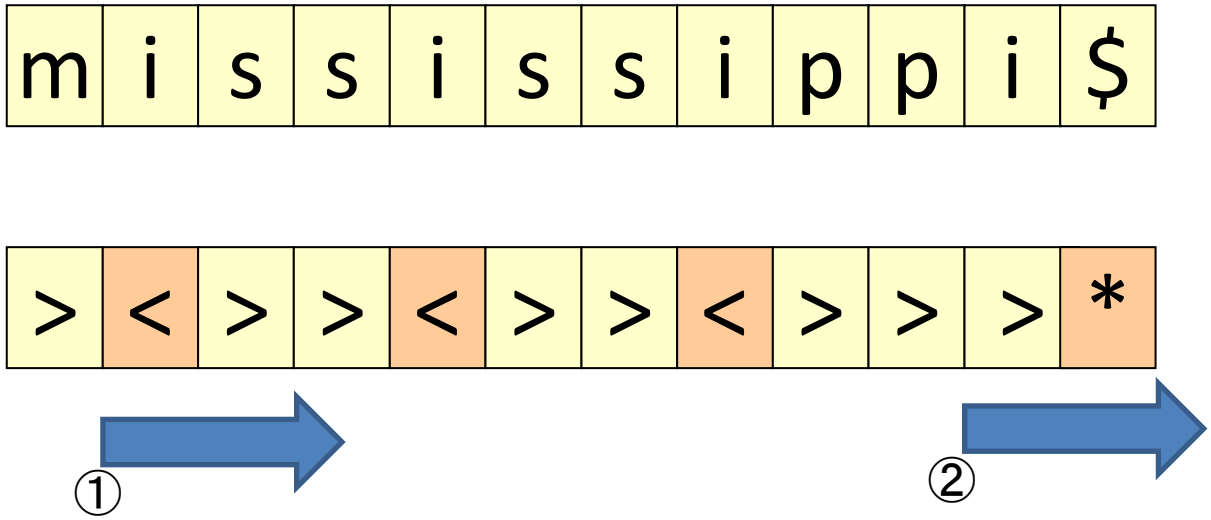
テーブル中の全文字の順位（ランク）を決定する
ことが全体で $O(n+N)$ のバケツソートで可能
Step 1: この表に含まれる全文字をソート
Step 2: そのランク結果を（縦の）各列に分配
さらに分配時に、その列でのランクに変換！

N（表のサイズ（表中の文字列の長さの総和））　≤　$1.5 \times n$（元の文字列の長さ）

ここのインプリはかなり速度を左右することに注意

Ko and Aluru '03 (4)

- マージ時の$O(1)$比較
  - ＜と＞の比較は1文字目のみでOK！

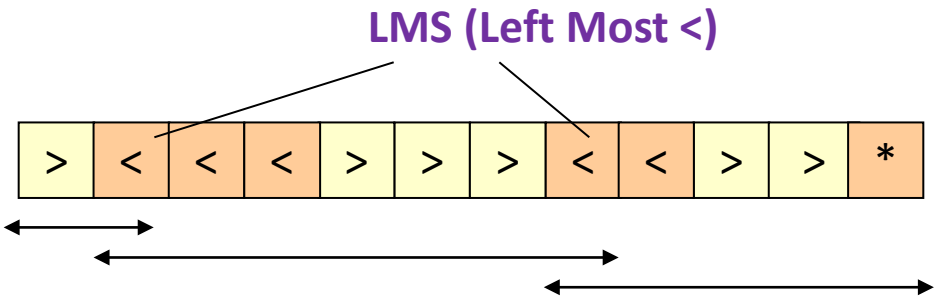


①と②は、どちらも‘i’で始まる接尾辞だが、異なるタイプなので、どちらの接尾辞が辞書順で速いかは2文字目以降を見ずとも決定可能！

Induced Sorting (1)

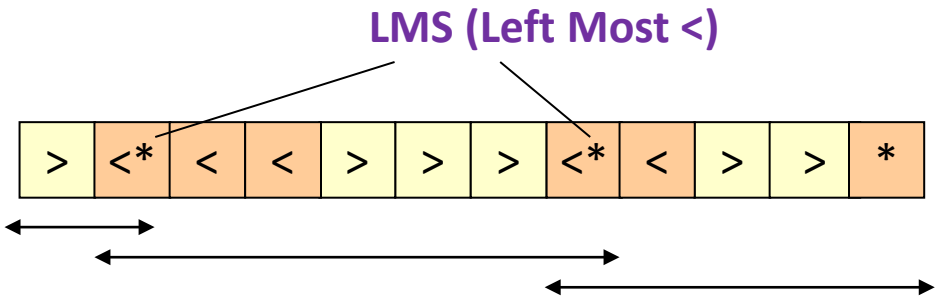
[Nong-Zhang-Chan '09]

- Ko-Aluru
  - Kärkkäinen-Sandersと比べて高速
    - KSと比べて部分問題サイズが小さいことによる
- Induced Sorting [Nong-Zhang-Chan ’09]
  - 接尾辞配列アルゴリズムの決定版
    - Ko-Aluru のアイディアの改良
  - 文字列の分割方法をさらに洗練させ、再帰的に解く「小さい部分問題」をより小さくしたことにより、再帰計算の時間が大幅に減少した
  - （論文によれば）KAより約1.5倍高速。メモリ量も2/3程度。

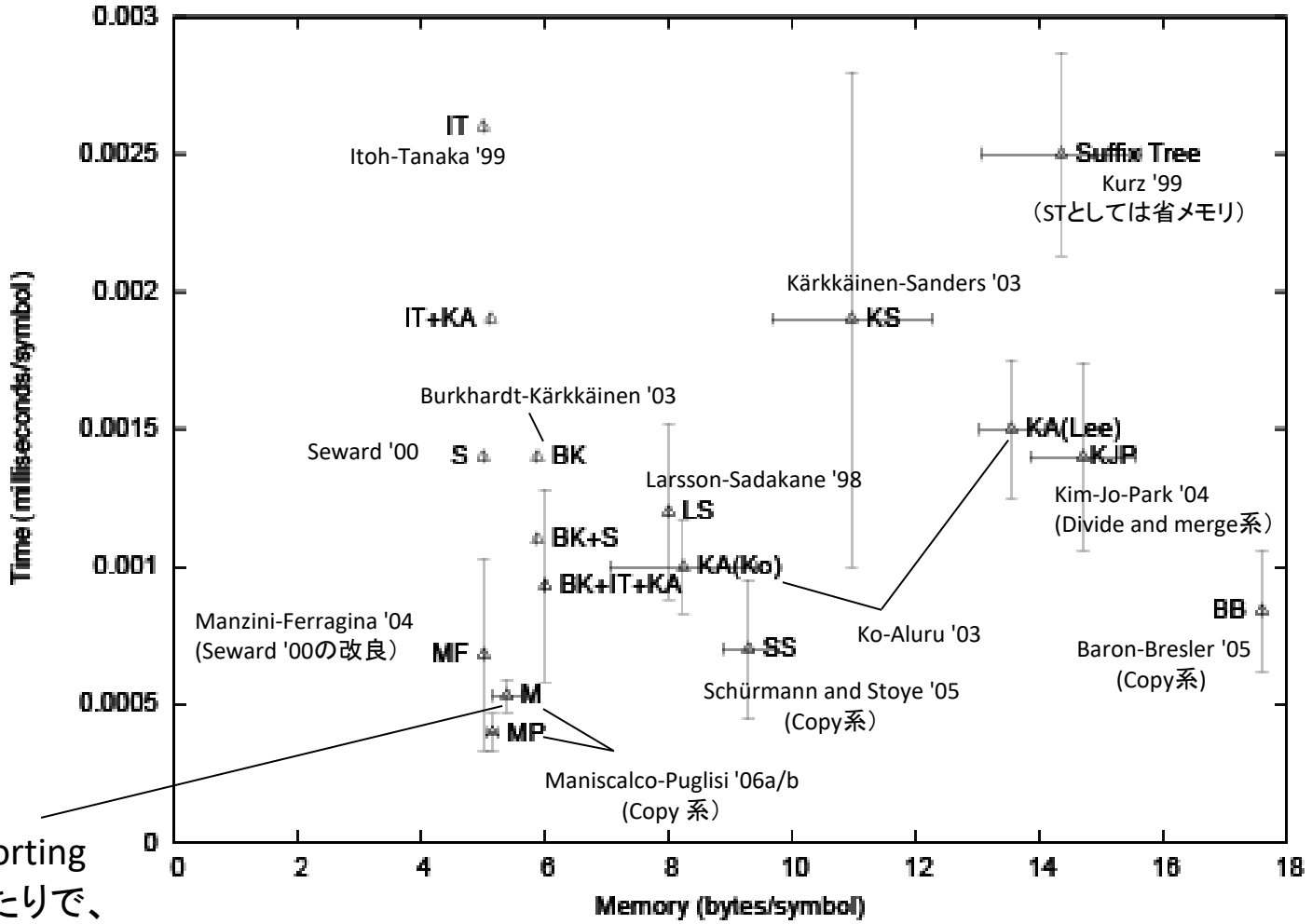


Induced Sorting (2)

- Ko-Aluruと同様 全接尾辞を「<」と「 >」の2つのタイプに分ける （線形時間）
- さらに連続する「<」の中で一番左のもの「<*」（LMS）を考える。
- Induced sorting ではこの 「<*」タイプのみをまずソートする（再帰）
  - LMSの数は全体の半分(正確には+0.5)以下
    - 「＜」と「＞」が交互に来る場合が一番多くなるが、その時の数
    - 実際にはもっと小さくなる
  - この部分問題サイズが他のアルゴリズムに比べてかなり小さいことが、全体のアルゴリズム性能の高い理由となっている
- 「>」タイプのソートは「<*」から「うまい」radix sortで可能
- 全「<」タイプのソートは「>」タイプから「うまい」radix sortで可能
  - ここが2段階となっている、というのがInduced sortingの新しいアイディア！
- それらのマージもKo-Aluruと同様に各比較は一文字目のみで可能（すなわち $O(1)$で可能！）



様々なアルゴリズムの計算機実験による実性能比較



Induced sorting はこのあたりで、最速、最小メモリ付近、でかつ理論的にも$O(n)$

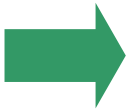
Fig. 8 from Simon J. Puglisi, W. F. Smyth & Andrew Turpin, "A taxonomy of suffix array construction algorithms", *ACM Computing Surveys,* 39-2 (2007).

接尾辞配列の最重要付加データ： 高さ配列(Height Array)

- Suffix Arrayにおいて、隣同士の接尾辞のLCP (longest common prefix) lengthの配列
  - ississippiとissippiのLCPは4
- 接尾辞配列から線形時間で計算可能 [Kasai et al. 2001]
- 様々な応用
  - RMQと組み合わせれば任意の接尾辞間でLCPが求められる
  - 検索は $O(m+\log n)$ にできる
  - 接尾辞木を線形時間で作成できる

Suffix Array

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

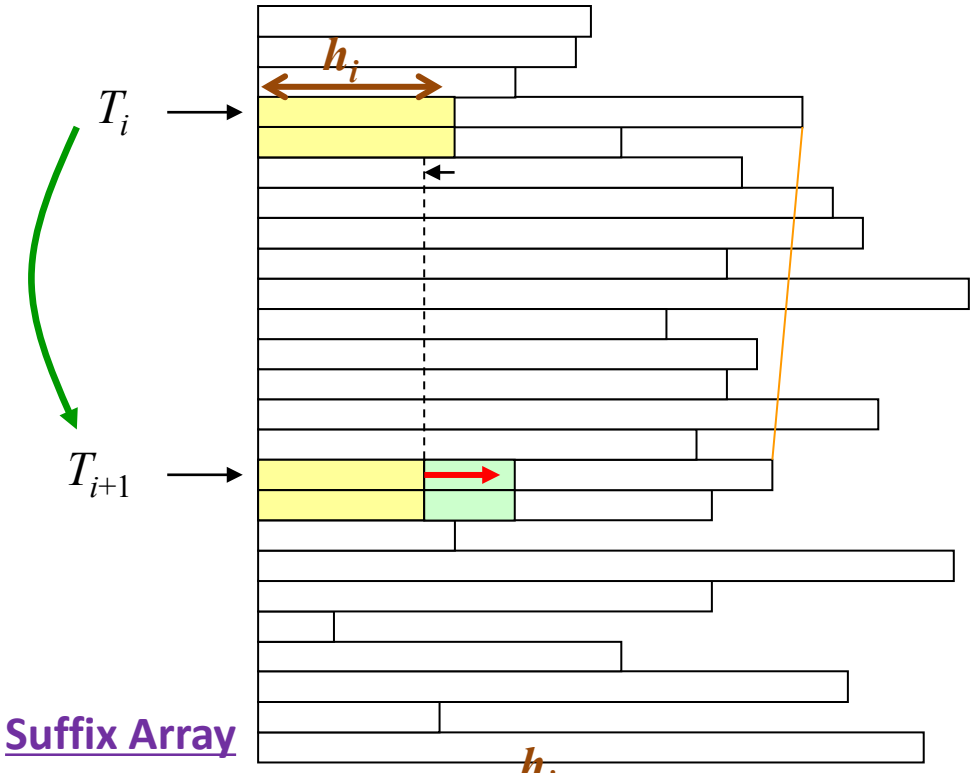


```
1
1
4
0
0
1
0
2
1
3
```

高さ配列

高さ配列の作成法

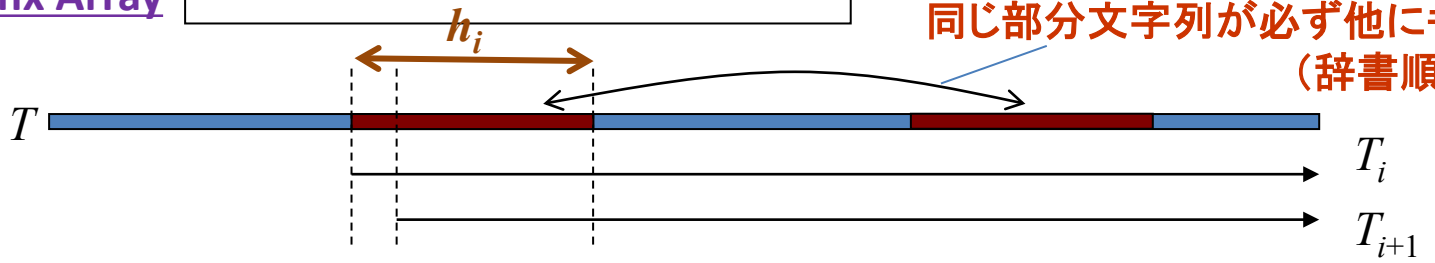
- 線形時間アルゴリズムが存在（しかも、アルファベットサイズに関係ない！）
  - 元の文字列の位置の順番で高さ配列を作成するだけ！
    - 逆接尾辞配列を用いる
  - ナイーブに作成すると$O(n^2)$必要



一つ前の接尾辞に対応する高さ配列の値より1小さい位置から調べる

↓

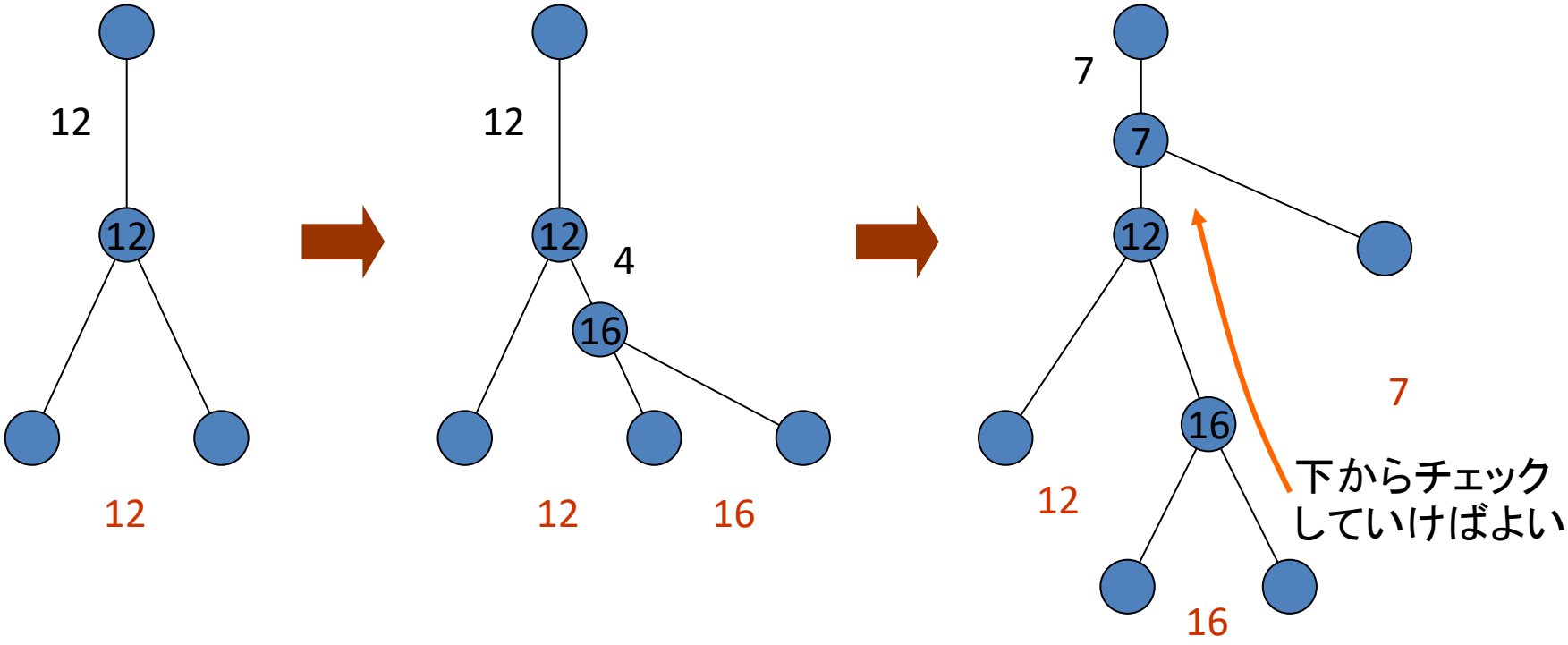
どう頑張っても$2n$以上チェックすることはない！



同じ部分文字列が必ず他にも存在する（辞書順にも注意）

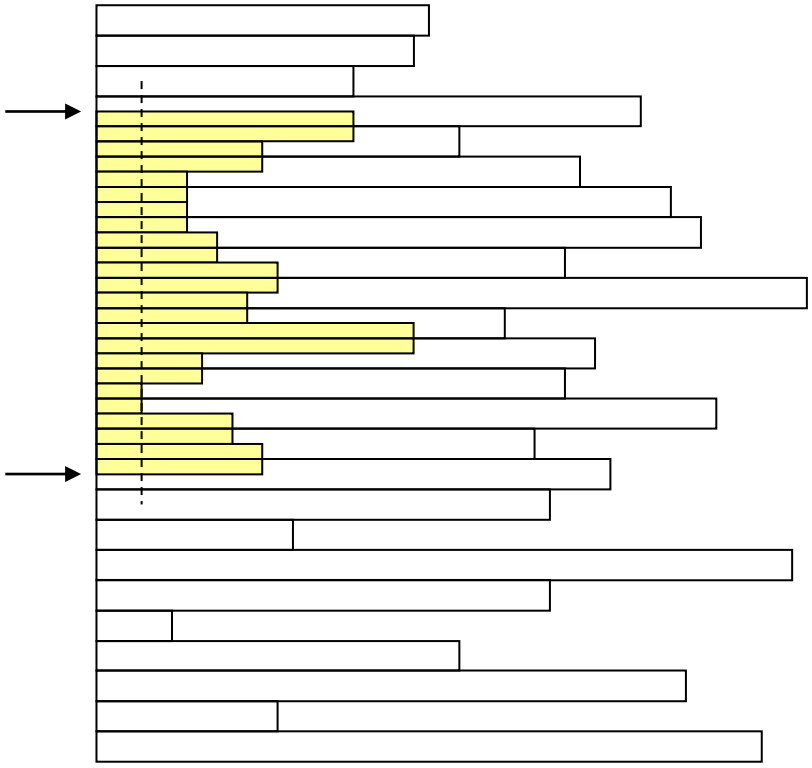
接尾辞配列→接尾辞木

- Height Array さえあれば、$O(n)$ で接尾辞木を構成可能
  - 左から順に作っていけばよいだけ
    - RMQの時のCartesian Treeの作成方法と似たアルゴリズム



LCPの計算

- LCP = Height Array + RMQ



Suffix Array

Height Array の該当する範囲内で最も最小値を求めるだけでよい

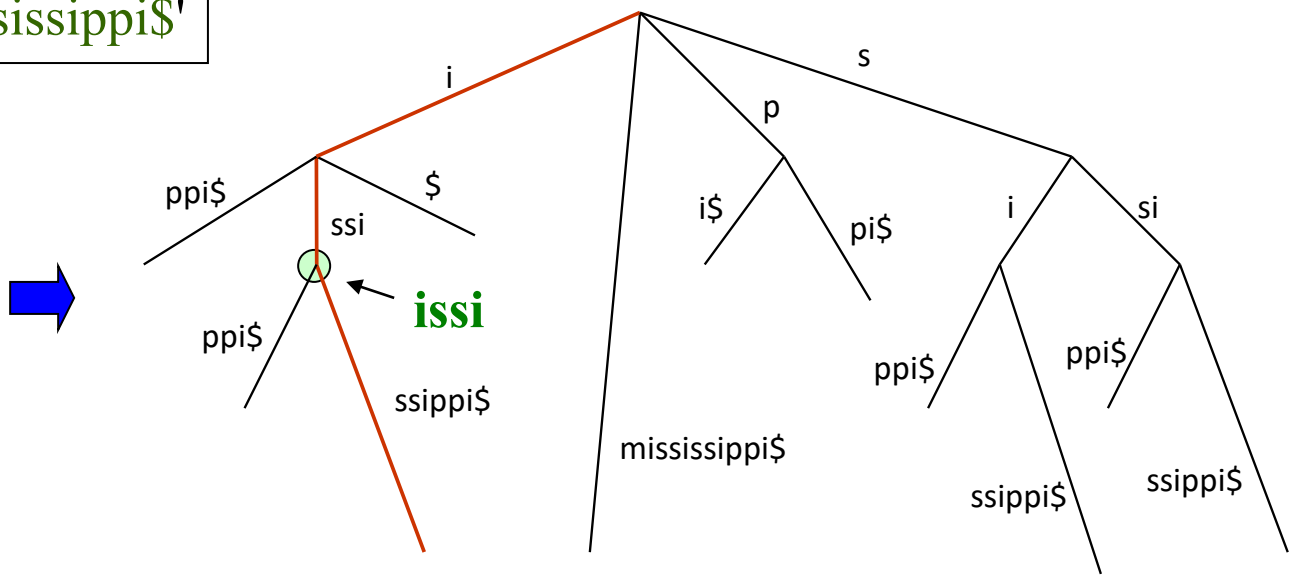
応用1： 接尾辞木の最も基本的な応用：Exact matching

- 作成して上から辿るだけ（接尾辞木）
- 二分探索、逆方向探索等で探索（接尾辞配列）

Suffix tree of 'mississippi$'

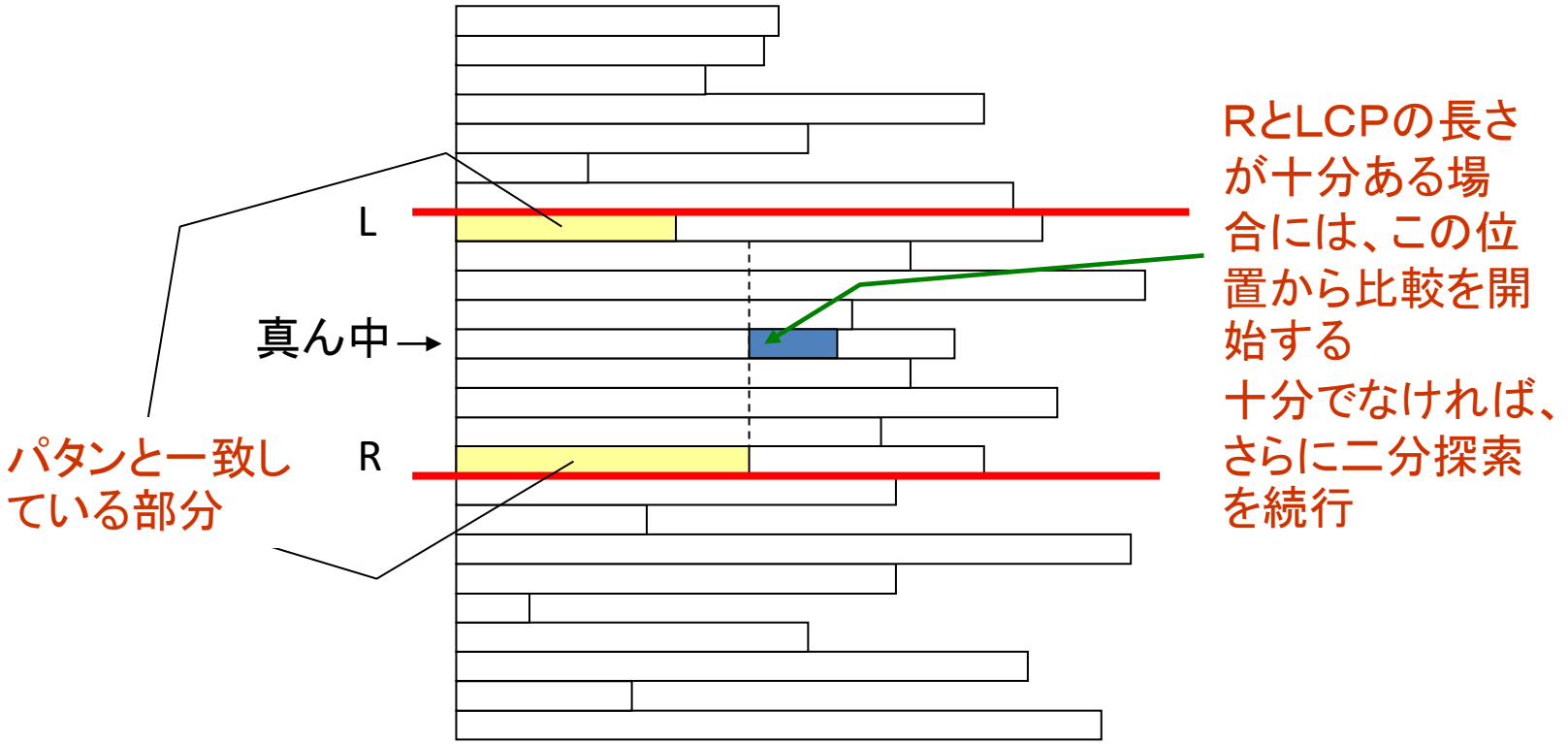
All the suffixes

mississippi$
ississippi$
ssissippi$
sissippi$
issippi$
ssippi$
sippi$
ippi$
ppi$
pi$
i$



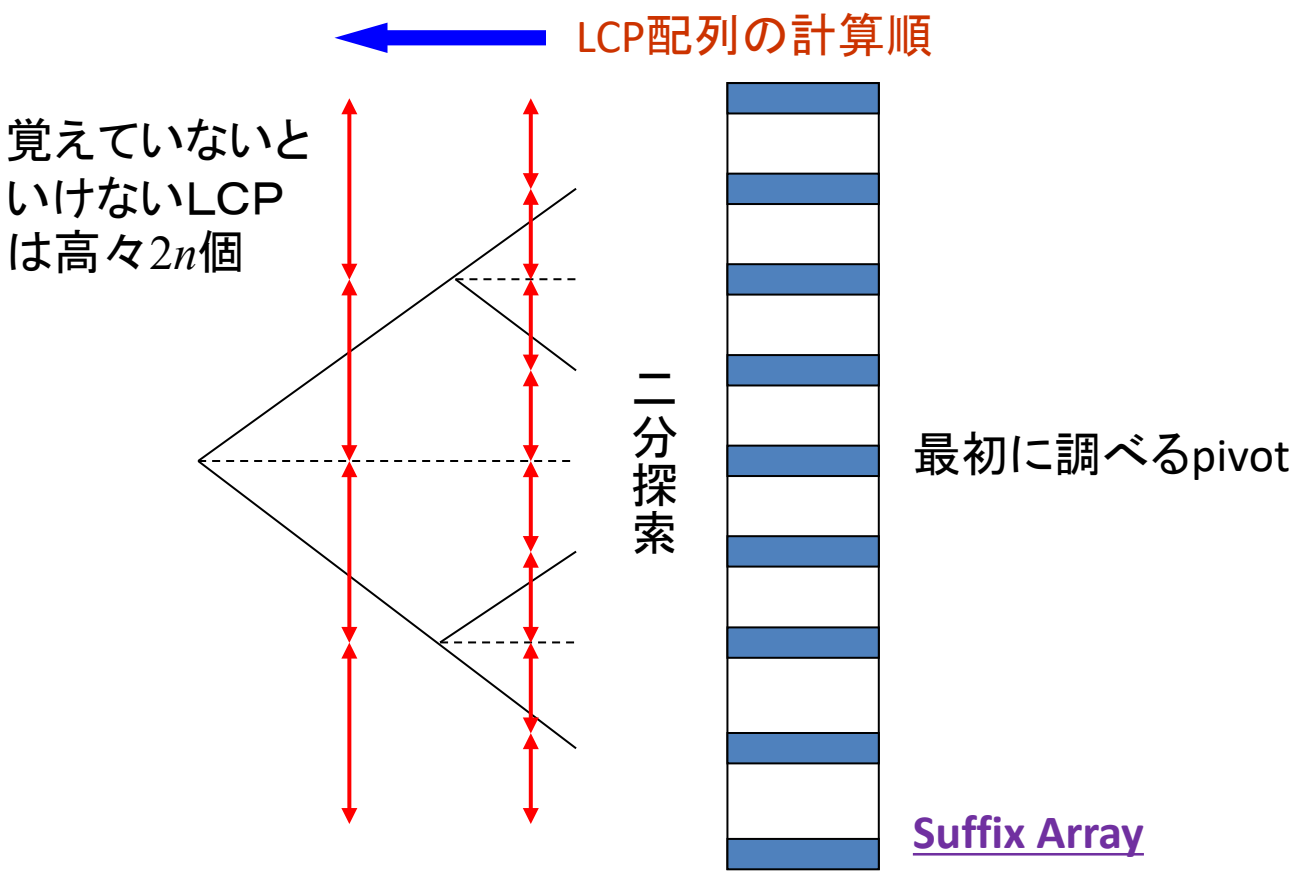
高さ配列を用いた検索

- RMQと組み合わせれば、LCPの長さが$O(1)$でわかるので、$O(m+\log n)$ で検索可能
  - 但し効率はよいとは言えない（実際には、最初に紹介した、「賢い検索」でも十分）



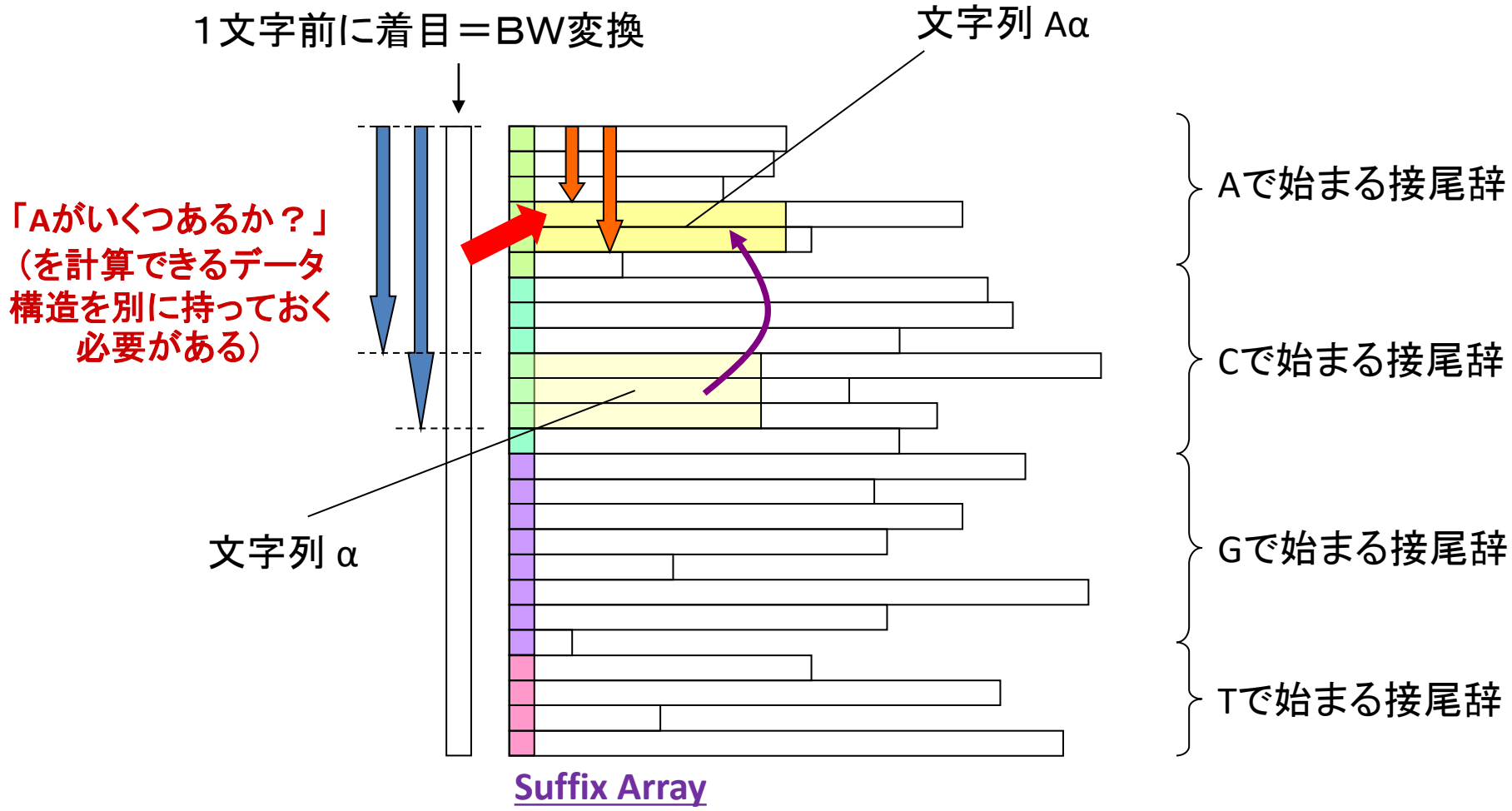
LCP Arrayによる検索 [Manber and Myers '92]

- $O(m + \log n)$ を直接可能にするデータ
  - 二分探索の枝分けは固定なので、必要なLCP長は$O(n)$個
    - 作成時間は$O(n)$ （高さ配列から計算）
    - MMのオリジナルの論文では $O(n \log n)$
      - 当時は高さ配列を $O(n)$ で作成する方法が知られていなかった



逆方向検索 （→ FM-Indexにつながっていく）

- なんと検索は逆方向にもできる！
  - Weiner Algorithmの逆suffix linkを辿ることに相当する
    - *cf.* BW変換（→圧縮アルゴリズムの紹介の中で紹介予定）



まとめ

- 接尾辞配列（続き）
- 次回
  - 接尾辞木の応用、拡張
  - 不完全一致検索、パタン発見など