

接尾辞配列

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>

□ 先週

- ◆ 接尾辞木の作り方

□ 今週

- ◆ 接尾辞配列とは
- ◆ 接尾辞配列の作成法
- ◆ 高さ配列の作成法
- ◆ 接尾辞配列を用いた検索
- ◆ 接尾辞配列から線形時間で接尾辞木を作る

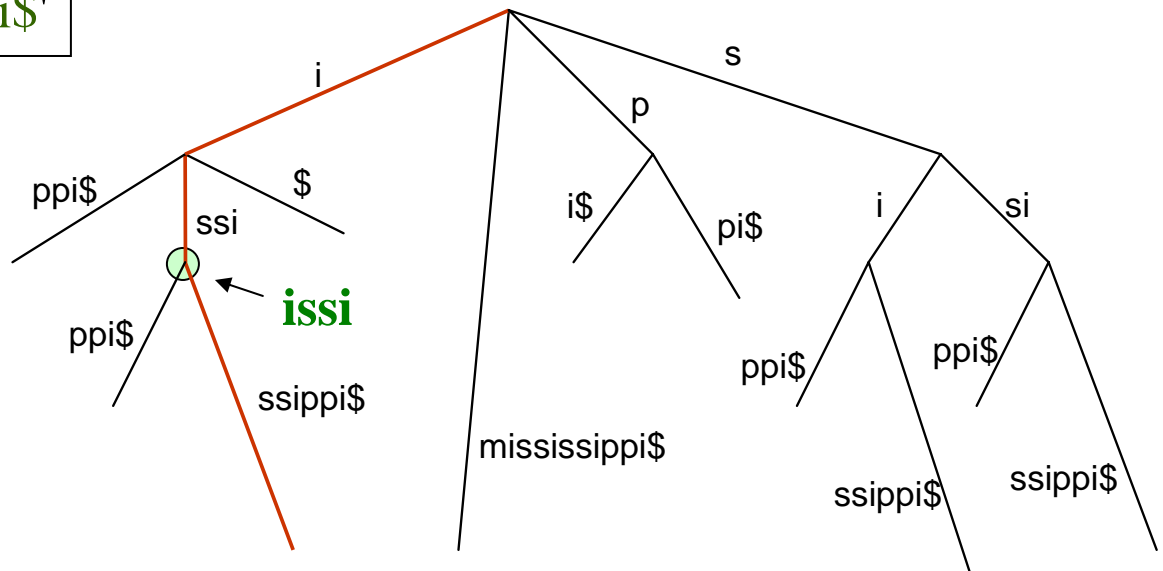
先週の復習：接尾辞木とは

- 文字列 S のすべての接尾辞を表した trie
 - 枝のラベル $\Leftrightarrow S$ の部分文字列
 - ルートから葉までのラベルを連結したもの $\Leftrightarrow S$ の接尾辞
 - 線形サイズ・線形時間で構成可能

Suffix tree of 'mississippi\$'

All the suffixes

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



□ 接尾辞配列

- ◆ すべての接尾辞のインデックスを辞書順にソートした配列
- ◆ 接尾辞木よりも遥かにコンパクト
- ◆ 接尾辞木上でのアルゴリズムの多くを実装できる
 - ▶ 子供がソートされた接尾辞木における葉を並べたものと同じ
 - ▶ ただし遅くなる場合もある
 - $+ \alpha$ のデータ構造を用いて接尾辞木に性能を近くすることができることも多い
 - ▶ メモリが少なくすむ分、速くなるアプリケーションもある

すべての
接尾辞

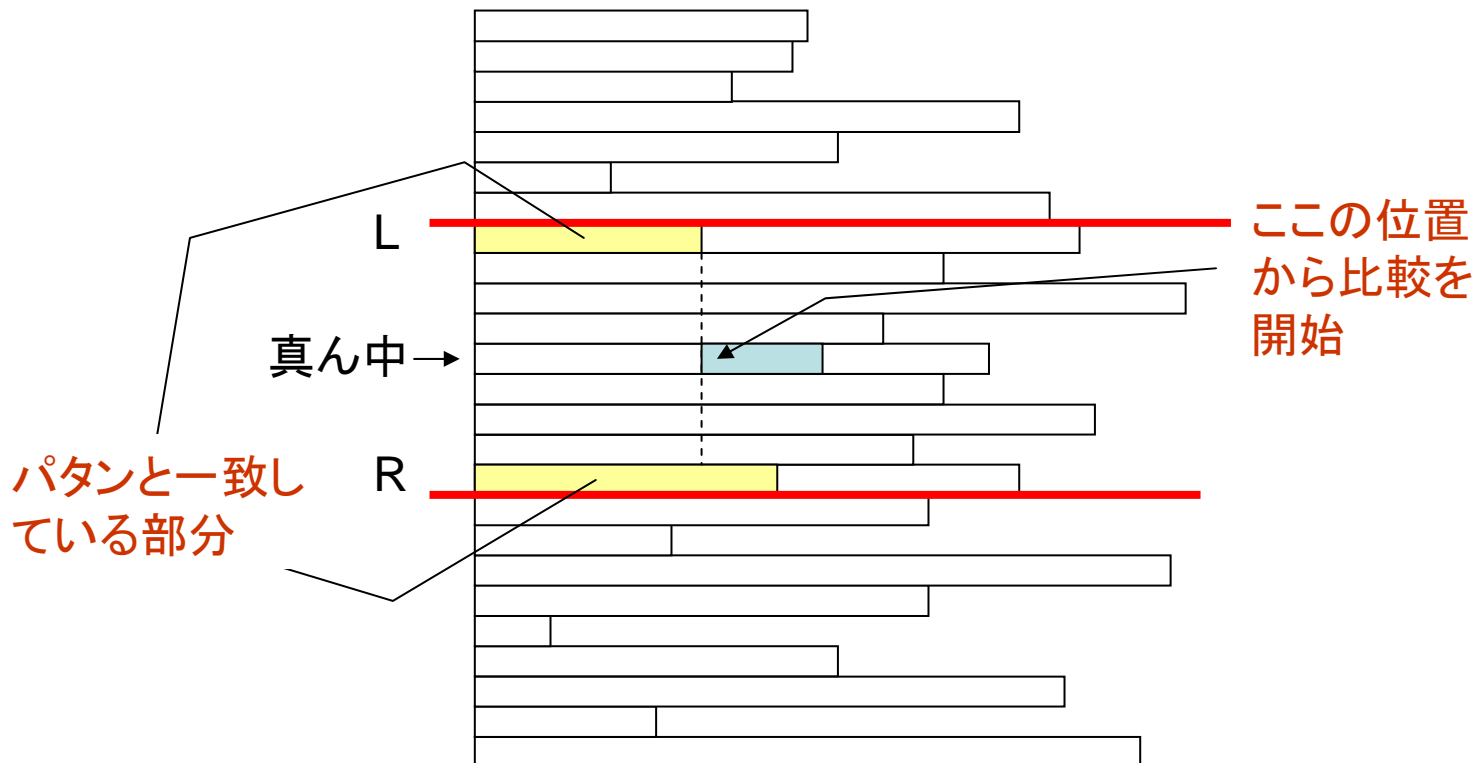
```
0: mississippi$
1: ississippi$
2: ssissippi$
3: sissippi$
4: issippi$
5: ssippi$
6: sippi$
7: ippi$
8: ppi$
9: pi$
10: i$
```

→
ソート

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

接尾辞配列を用いた検索

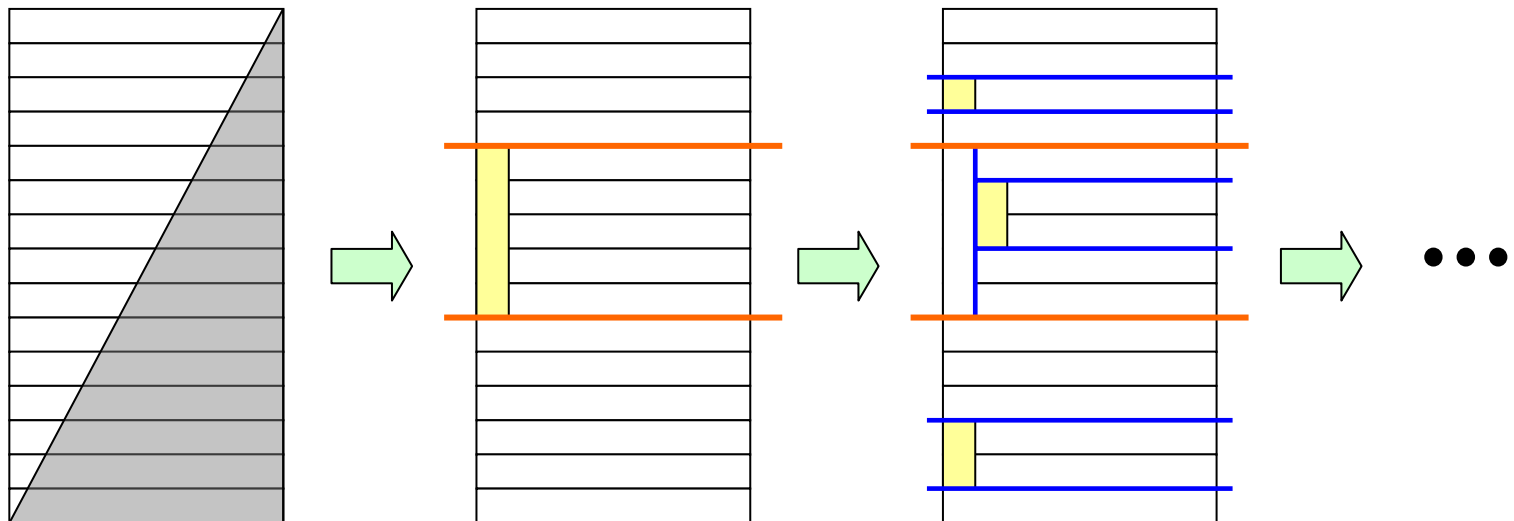
- 2分探索 - 若干賢く検索することは可能
 - ◆ $O(m \log n)$ (n :テキストサイズ m :パターンサイズ)
 - ◆ 平均的には $O(m + \log n)$
- 追加のデータ構造を用いれば、最悪でも $O(m + \log n)$ や $O(m)$ が可能
 - ◆ パターンを後ろから探索する方法もあつたりして奥は深い



- Naively from suffix trees
 - ◆ 常に $O(n)$ だが、メモリが多く必要・アルゴリズムが複雑なため現実には遅い
- 3-way (ternary, multi-key) quick sort
 - ◆ Bentley & Sedgwick '97
 - ◆ ランダムな配列に対しては高速 ($O(n \log n)$)
 - ◆ 繰り返しの多い塩基配列などでは遅く、 $O(n^2)$ になるケースも多い
- Doubling algorithm
 - ◆ Manber & Myers '92, Larsson and Sadakane '98
 - ◆ 常に $O(n \log n)$
- BWT (Burrows Wheeler Transform)-like
 - ◆ Itoh-Tanaka '99, Seward '00, Manzini-Ferragina '02
 - ◆ 省メモリ。 $O(n^2 \log n)$ だが、実際の計算では速い、という報告あり。
- Divide and conquer
 - ◆ Kärkkäinen & Sanders '03, Ko & Aluru '03, Kim et al. '03.
 - ▶ 常に $O(n)$
 - ◆ Burkhardt & Kärkkäinen '03
 - ▶ $O(n \log n)$ 、作業メモリが $o(n)$

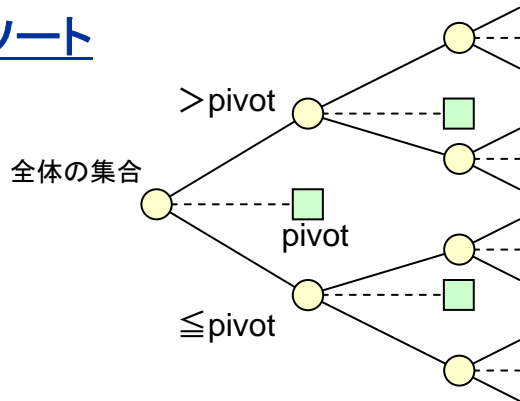
□ 3-way (ternary) quick sort

- ◆ ランダムなら $O(n \log n)$ だが、通常の文字列のソートの場合よりも $O(n^2)$ になるケースが多いことに注意
 - ▶ DNA などでは、「NNNNN」とか「ATATATATA」などの繰り返しがあったりするだけでかなり危険
 - ▶ 一般的な文章でも「わたし」の次には「は」が来ることが多い
 - いずれもうまく等分割とならない原因となる

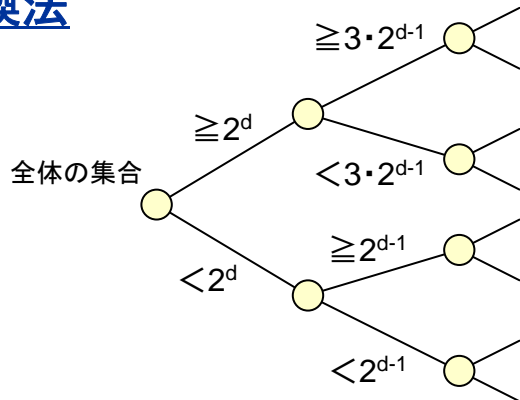


ソート法の比較

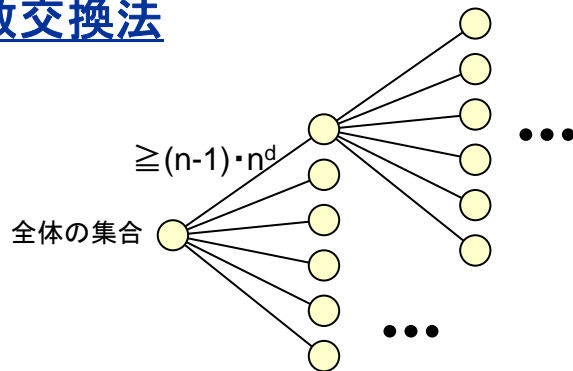
クイックソート



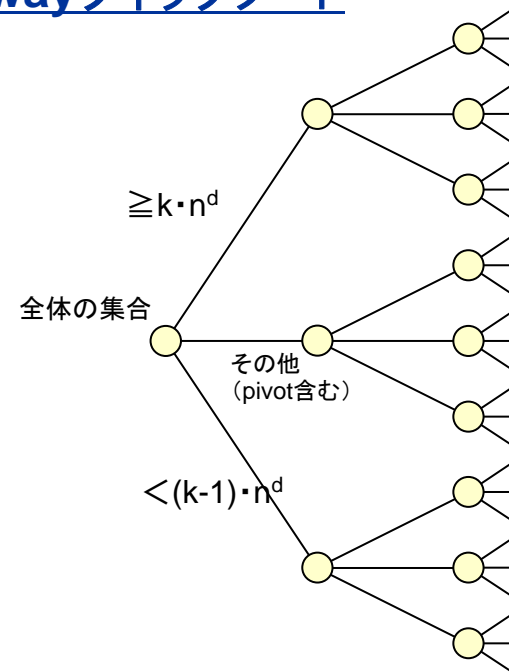
基数交換法



n進基数交換法



3-wayクイックソート



- 分割の方法が異なるだけ
 - ◆ 途中で別の方法に切り替えることが可能
 - ◆ ランダムな対象に対してはどれも概ね $O(n \log n)$

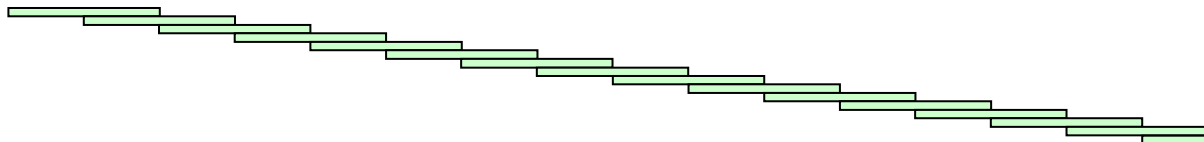
□ Doubling algorithm

◆ $\log n$ 回の基数ソートで計算

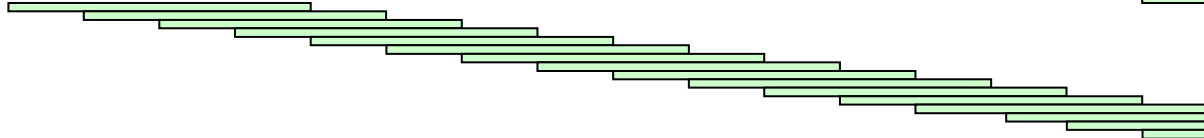
▶ それぞれの段階は線形時間で可能！

A	T	A	C	G	T	A	A	C	G	T	A	A	C	T	G
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

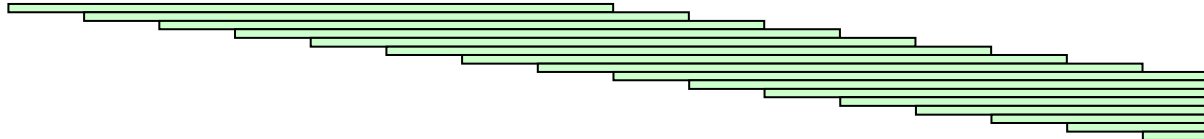
1文字をソート



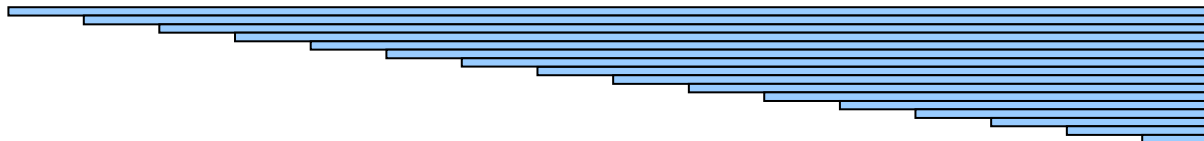
2文字をソート



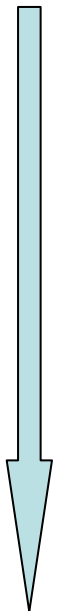
4文字をソート



8文字をソート



16文字をソート



■ 基数ソートの適用のしかた

最初の文字列 ($i=0$)

3 0 4 1 5 9 2 3 5 2 3 1 4 5 2 3 1 2

となりあう2文字でソートした時のソート順での番号に置き換える

04 ▶ 0	23 ▶ 5	45 ▶ a
12 ▶ 1	30 ▶ 6	52 ▶ b
14 ▶ 2	31 ▶ 7	59 ▶ c
15 ▶ 3	35 ▶ 8	92 ▶ d
2\$ ▶ 4	41 ▶ 9	

文字はinteger

このソートは2桁
の基数ソートで線
形時間で可能

6 0 9 3 c d 5 8 b 5 7 2 a b 5 7 1 4

次はとなりあう文字ではなく2つ隣の文字とあわせて
基数ソートを行う (i 番目では 2^i だけ隣とあわせる)

全部計算するか、アルファベットサイズと文字列長が
等しくなったら終了

Seward "Copy" Algorithm '00 (概略)

■ SA中に似た並びがあることを利用！

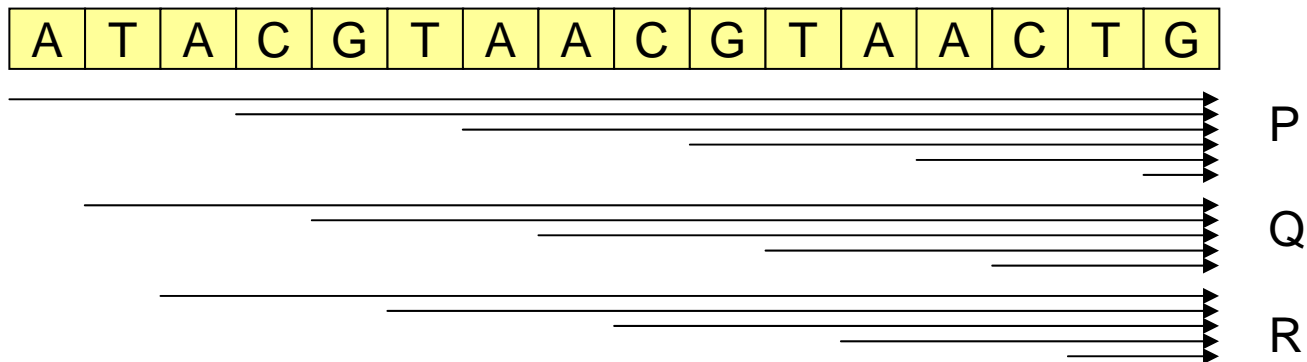
一つ前の文字がAのものに関して順番にコピー

A	A	
	C	
	G	
T		
C	A	
	C	
	G	
G	T	
	A	
	C	
T	G	
	T	
	A	

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のソートはそれ以外を用いれば計算できる
4. *Cで始まるものはこれを利用して計算
5. 次に少ないものを選んで、繰り返し(但し、もう計算できているものは計算しない)

□ 接尾辞を3種類に分ける

- ◆ $3i, 3i+1, 3i+2$ 番目から始まる接尾辞(P,Q,Rとする)を考える
- ◆ P,Qからなる接尾辞配列を作成する
 - ▶ $f(2n/3)$ 時間
- ◆ その接尾辞配列を用いてR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



□ P+Qの作り方

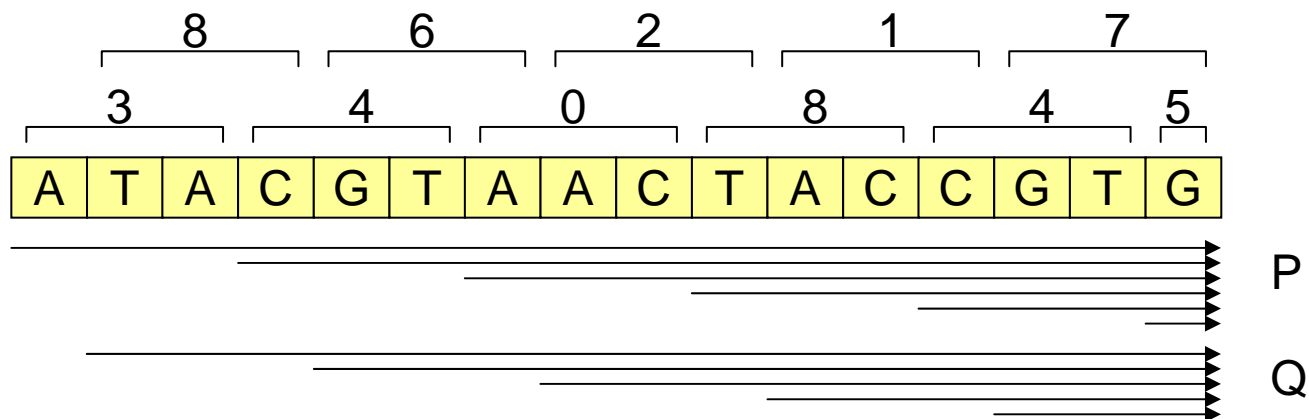
◆ P,Qの各接尾辞の先頭3文字の組をまずソート

▶ 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

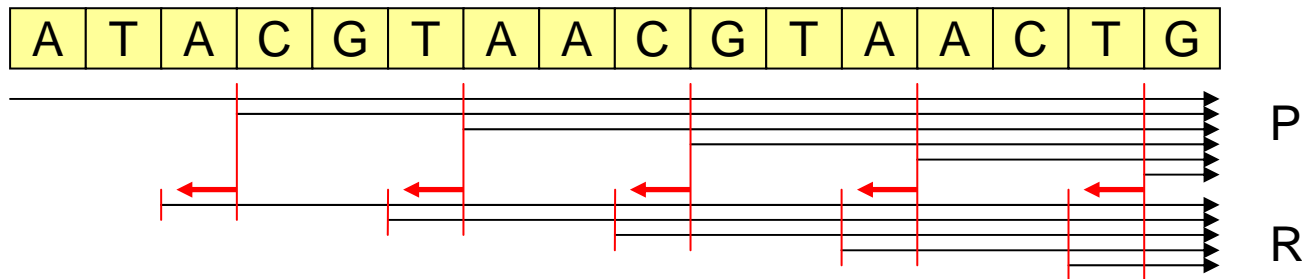
▶ 結果を用いてP+Qに対する接尾辞配列を作成(自明)



340845\$86217に対する接尾辞配列を作成する

□ Rの作り方

- ◆ Pから基数ソート一回で作ることができる！
- ◆ PはP+Qの一部



□ 逆接尾辞配列

- ◆ 様々な他のアルゴリズムでも使用されるデータ構造
- ◆ $SA^{-1}[i] = j \leftrightarrow SA[j] = i$

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

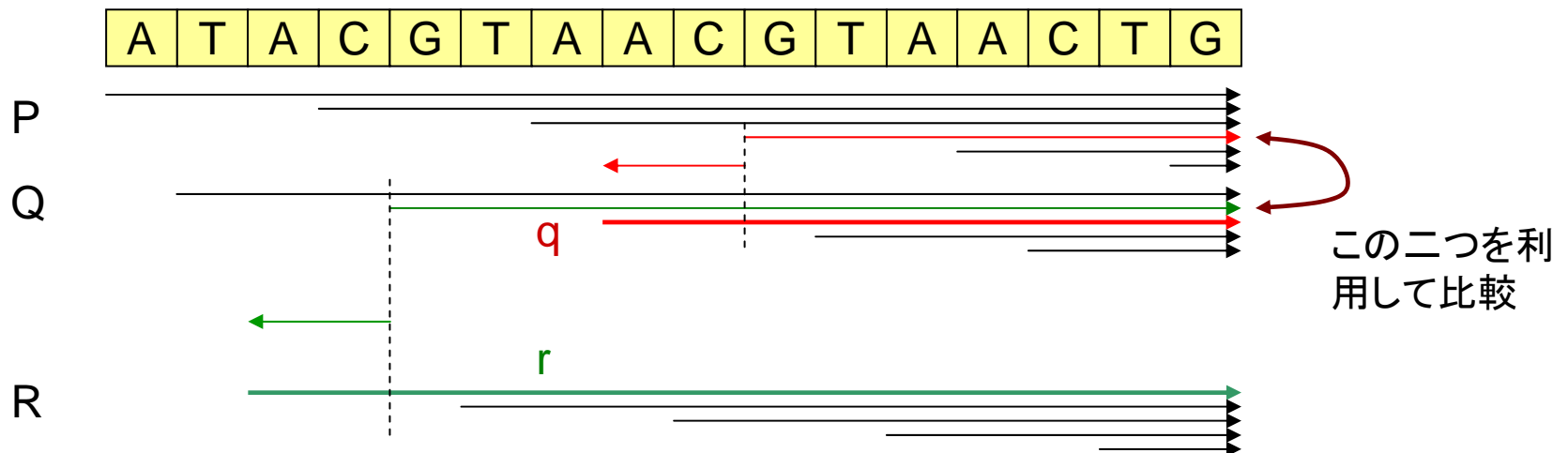
Suffixes of theText

SA

SA⁻¹

□ 2つのマージをするには？

- ◆ 接尾辞同士の比較が $O(1)$ であれば、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
 - ▶ $P+Q$ に入っている接尾辞同士が $O(1)$ で比較できることを利用
 - さらに1文字または2文字



□ 計算時間

◆ $f(n) = f(2n/3) + O(n)$

◆ これは $O(n)$

□ 一般的に $f(n) = c_1 \cdot f(c_2 \cdot n) + O(n)$ ($0 < c_2 < 1$, $0 < c = c_1 c_2$) の時、

◆ $0 < c_1 c_2 < 1$ ならば $f(n) = O(n)$

◆ $c_1 c_2 = 1$ ならば $f(n) = O(n \log n)$

$$f(n) < c_1 \cdot f(c_2 \cdot n) + a \cdot n < c_1^2 \cdot f(c_2^2 n) + a(1+c)n < \dots < c_1^{\log n} \cdot f(1) + a(1+c+c^2+\dots) < a \cdot n / (1-c) + \text{const}$$

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

<http://www.cs.helsinki.fi/u/tpkarkka/publications/icalp03.pdf>

□ KSの省メモリ化

- ◆ $o(n)$ の追加空間計算量で $O(n \log n)$ 時間としたもの
- ◆ 意外と速いらしい?!

□ 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

(これが簡単だがもっと賢いカバーも存在する)



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

- 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
- 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、どの剰余同士の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 $f(n) = O(n \log n)$
 - ◆ KSは $k=3$ の時と同じ

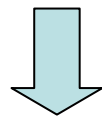
- となりの接尾辞との辞書順の大小で接尾辞を2つのタイプに分割し、少ない方に対して接尾辞配列を作成
 - ◆ 「うまく」ラディックスソートでランキングすることで小さい問題に変換
 - ◆ その結果から計算していない接尾辞も「うまく」ラディックスソートで計算
 - ◆ マージも容易
 - ▶ 同じ文字の>は<より必ず辞書順で小さい= $O(1)$ で比較可能

m	i	s	s	i	s	s	i	p	p	i	\$
---	---	---	---	---	---	---	---	---	---	---	----

< が少ない

>	<	>	>	<	>	>	<	>	>	>	*
---	---	---	---	---	---	---	---	---	---	---	---

←		←		←		←	
2	2	1	3				



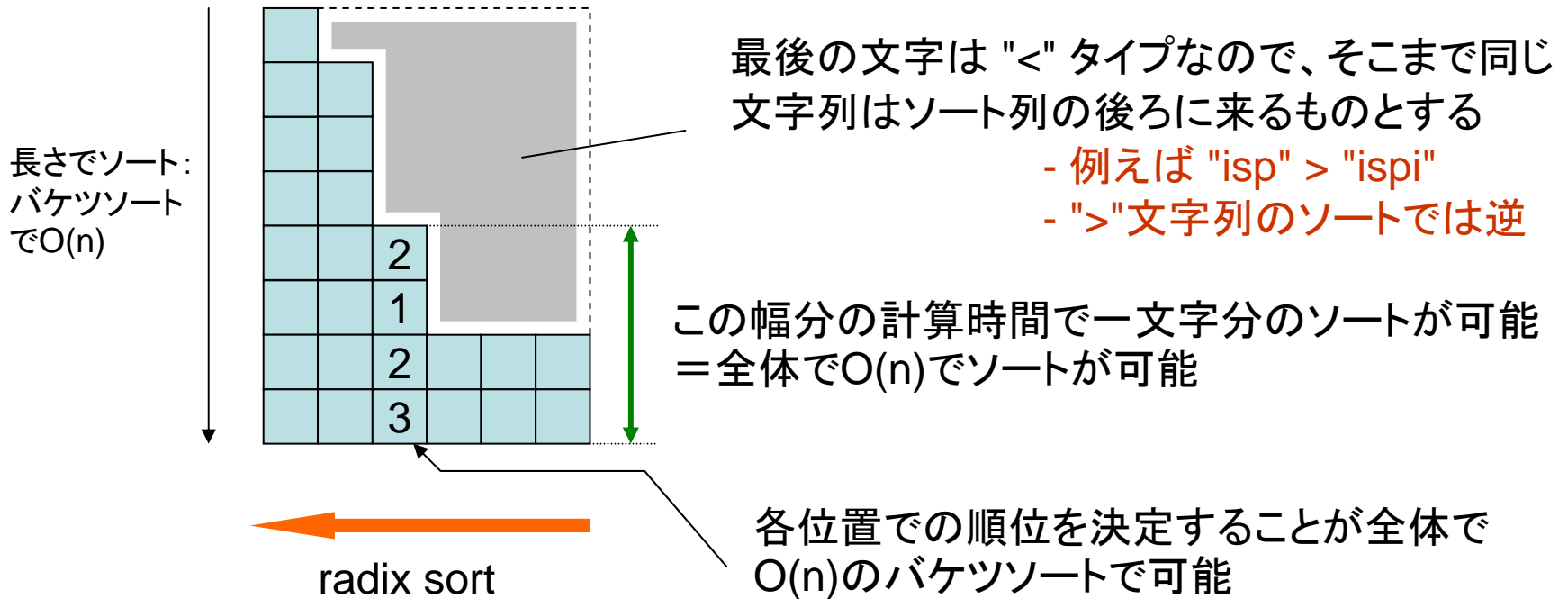
再帰的な呼び出し

辞書順でとなり同士を比較する:
後ろから計算すれば $O(n)$ で可能

1文字だと思ってラディックスソート:
ランクに書き換える

□ 「うまい」 radix sort

”<”文字列のソート



- MF (Sewardの改良)が高速、メモリも少ない
 - ◆ LS(MMの改良), BKがその次。
 - ◆ LSよりBKの方がメモリが少なくて済む
- KSは改良することで2倍以上速くできる
 - ◆ 改良KSとKAがほぼ同じ速度
 - ◆ 最悪の場合の計算量が気になる場合の選択
- オリジナルのKSを除き、接尾辞木から作るより速い
 - ◆ もちろんメモリは少なくて済む
- KS,KAといった線形時間アルゴリズムは必要メモリサイズが大きい

注) MF (BSなども)は実際には長い配列(長さ数百~)をたくさん繰り返したような配列(平均LCPが長い)では非常に遅い

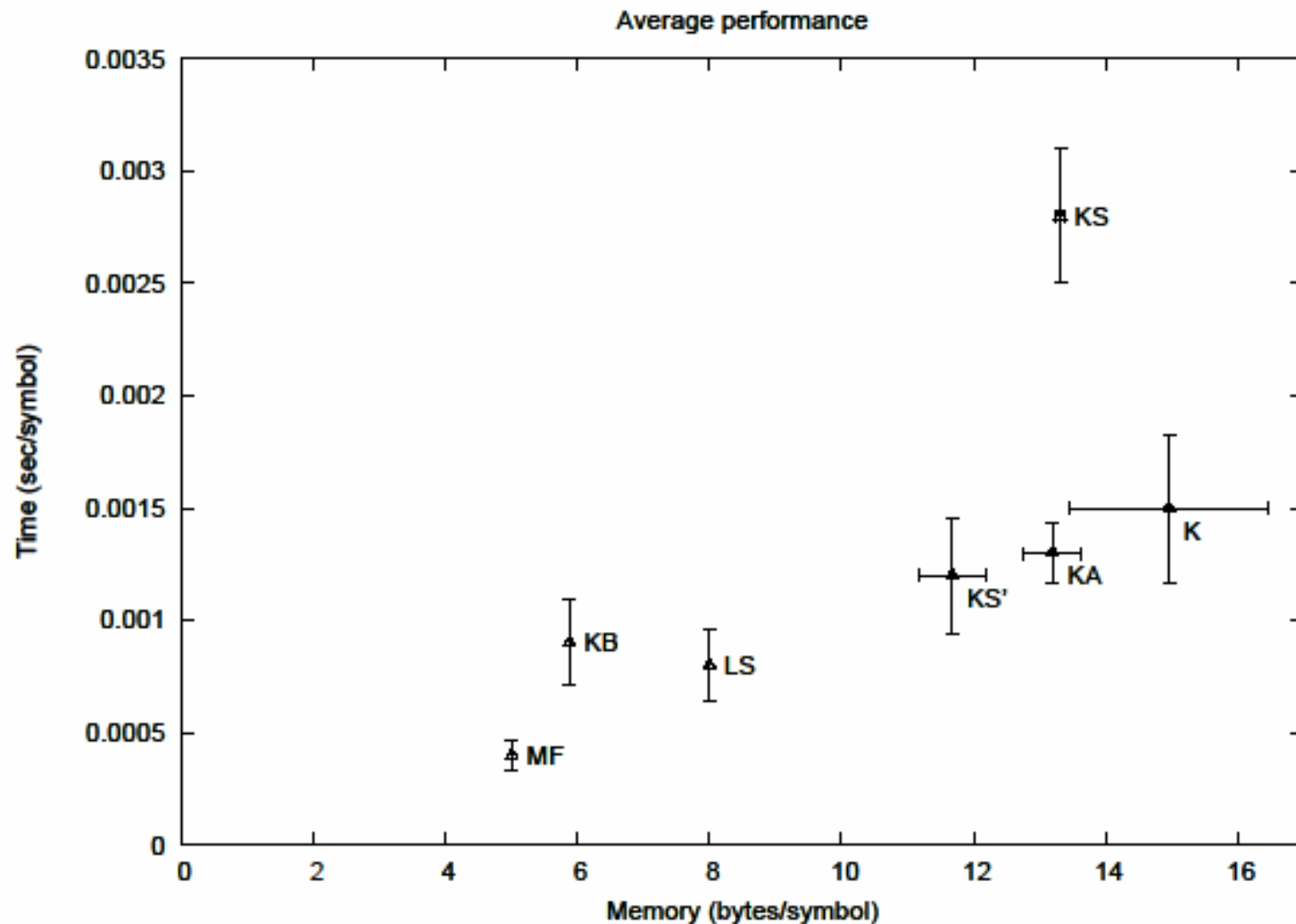


Figure 4: Resource requirements of the seven algorithms averaged over the real-world test corpus. Error bars are one standard deviation.

高さ配列(Height Array)とは

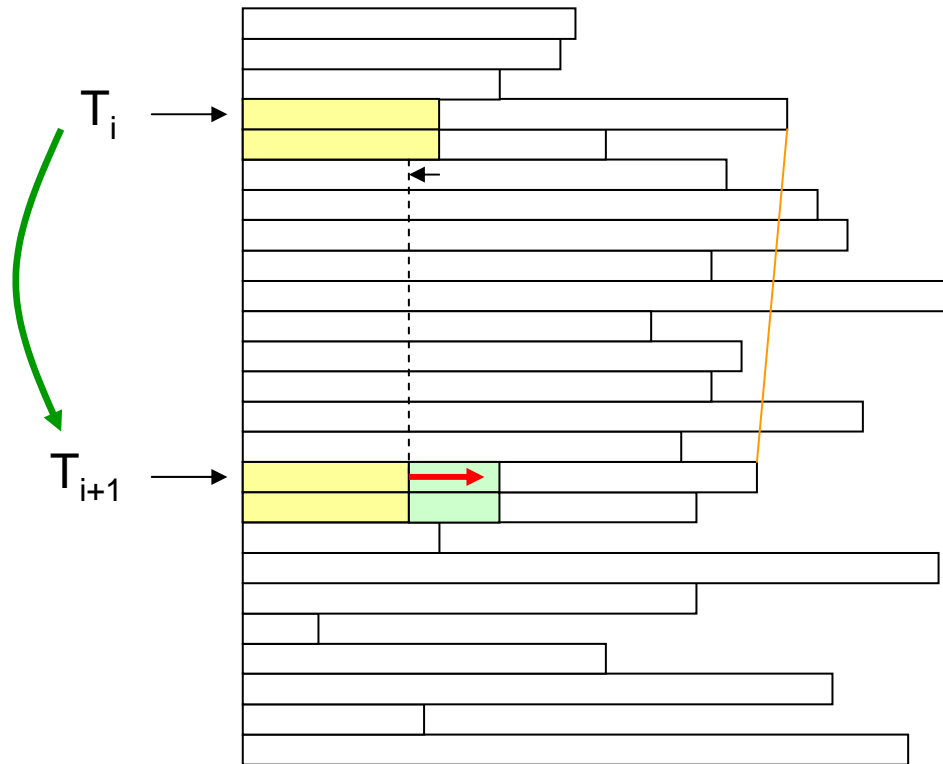
- Suffix Arrayにおいて、隣同士の接尾辞のLCP (longest common prefix) lengthの配列
 - ◆ ississippiとissippiのLCPは4
- 接尾辞配列から線形時間で計算可能 [Kasai et al. 2001]
- 様々な応用
 - ◆ RMQと組み合わせれば任意の接尾辞間でLCPが求められる
 - ◆ 検索は $O(m+\log n)$ にできる
 - ◆ 接尾辞木を線形時間で作成できる

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

→ 高さ配列

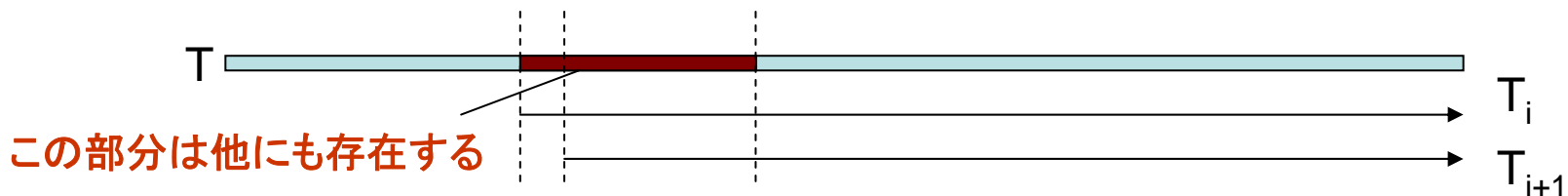
□ 線形時間アルゴリズムが存在

◆ 逆接尾辞配列を用いる



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

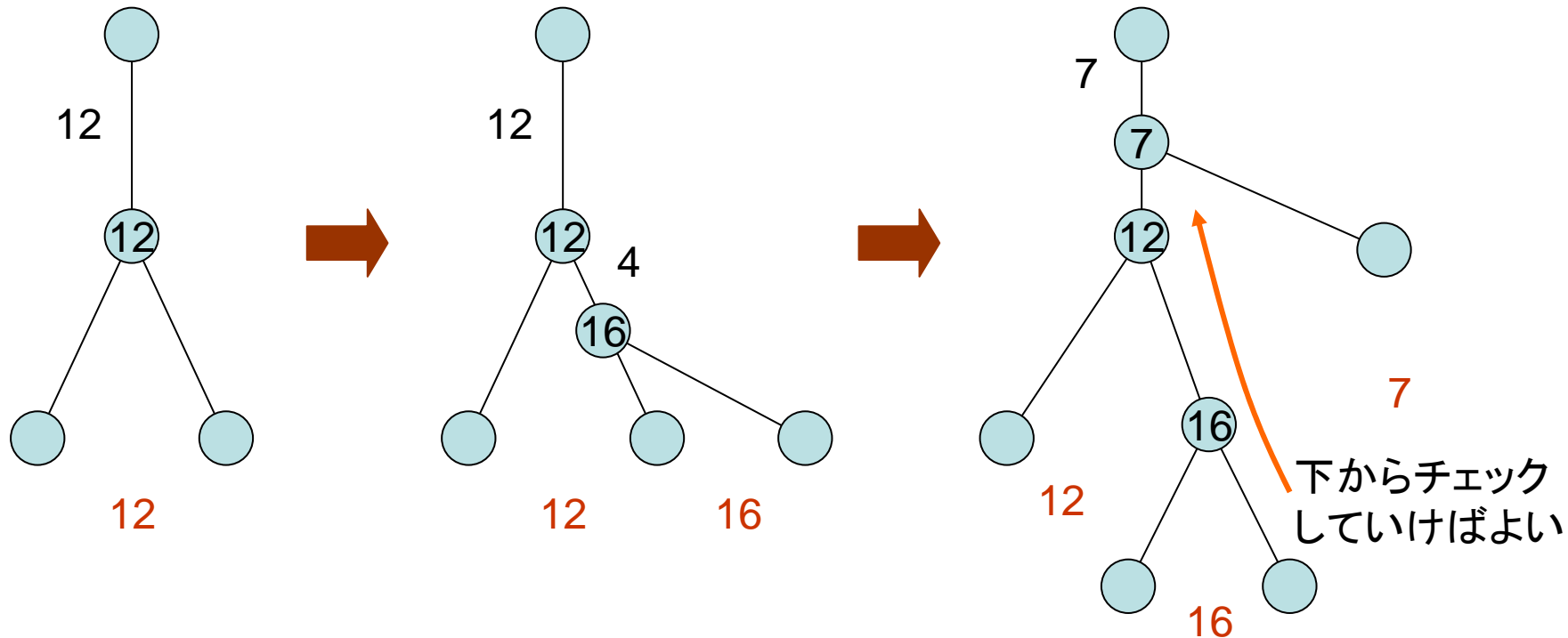
↓
どう頑張っても2n以上チェックすることはない!



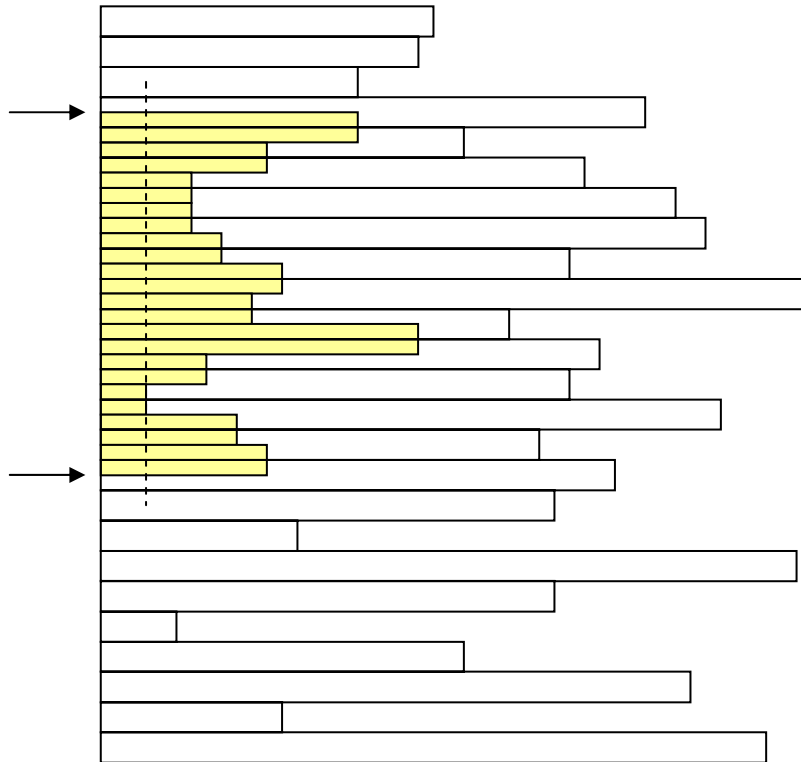
この部分は他にも存在する

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の作成方法とほぼ同じ



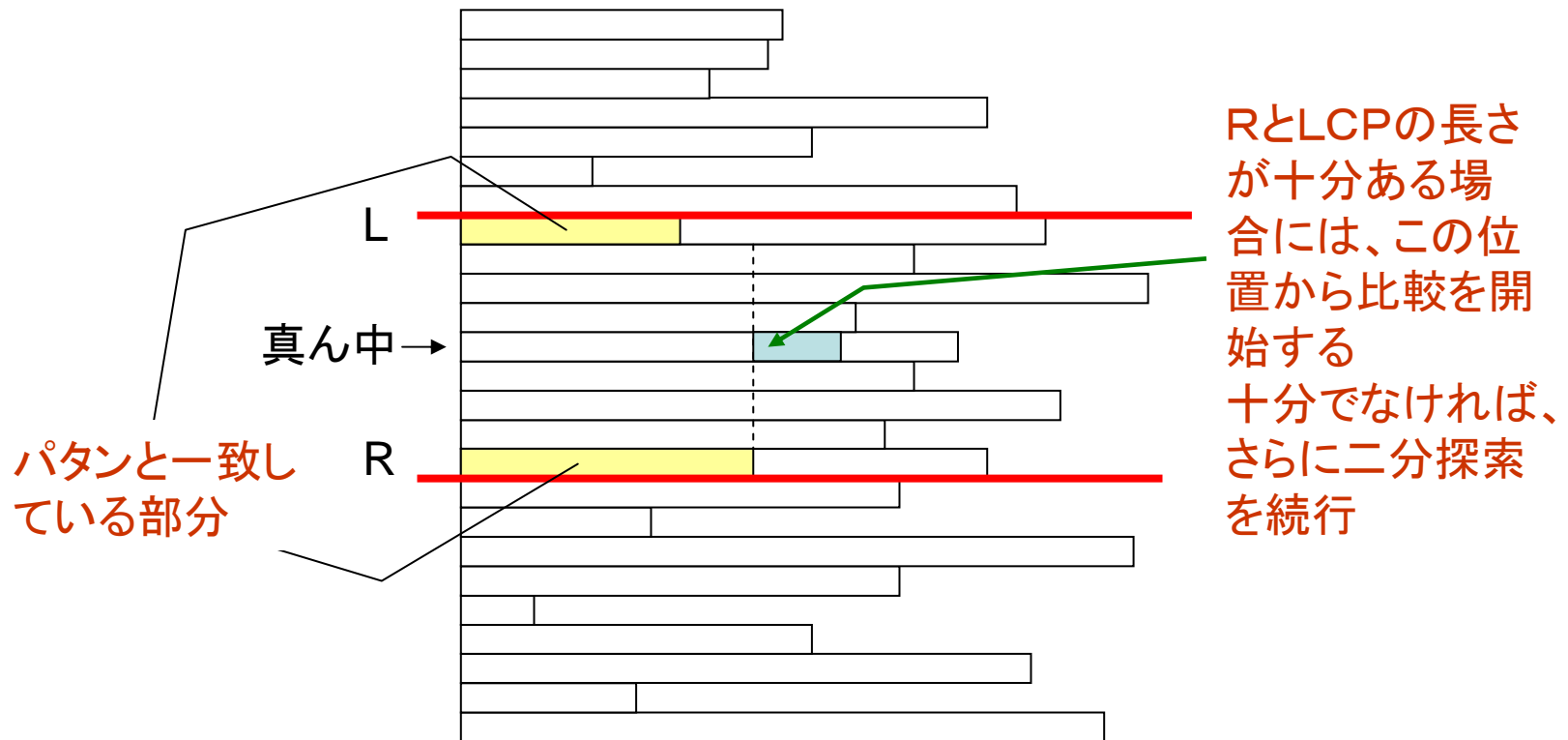
□ LCP = Height Array + RMQ



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

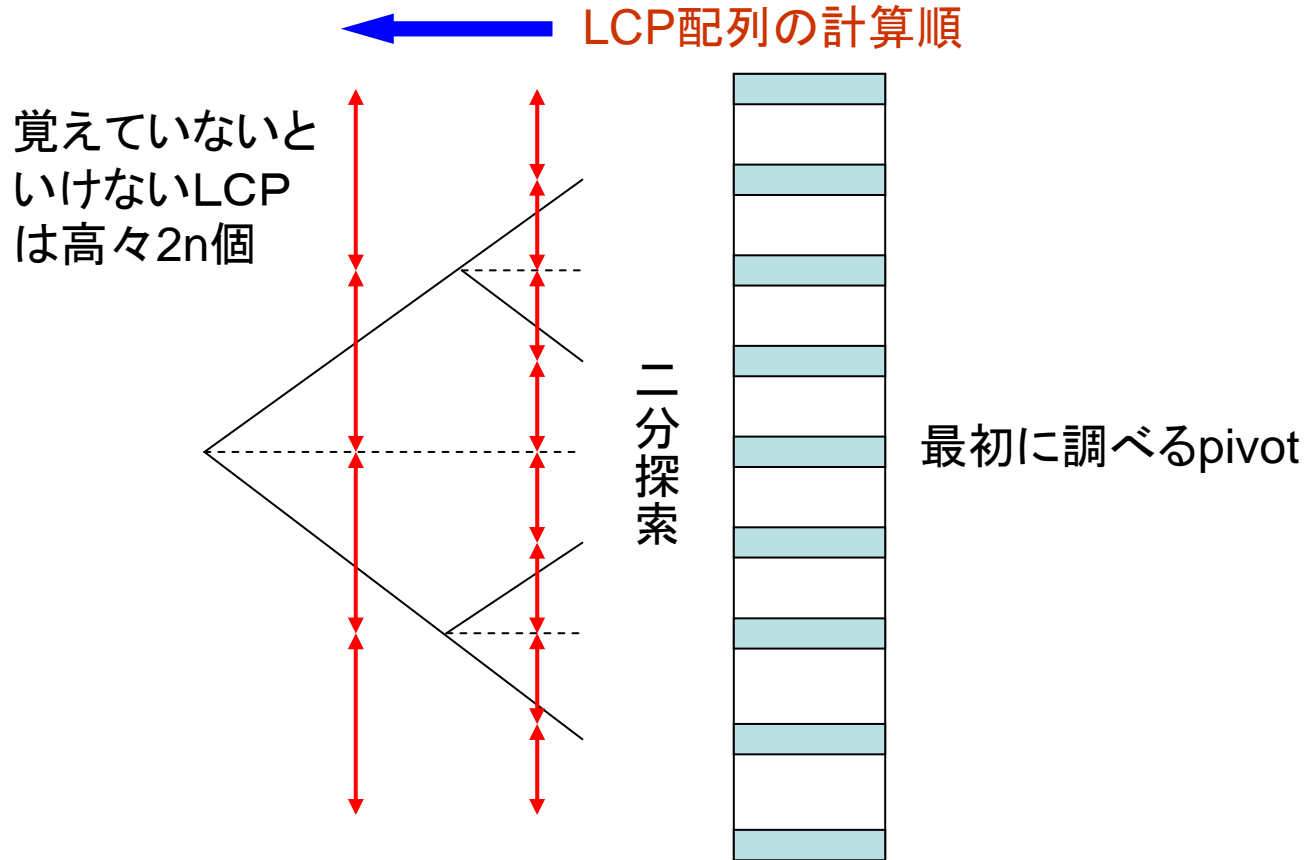
高さ配列を用いた検索

- 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



□ $O(m + \log n)$ 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 $O(n \log n)$



□ 接尾辞配列とその構成アルゴリズムのいろいろ

- ◆ Bentley & Sedgewick '97
- ◆ Manber & Myers '92
- ◆ Seward '00
- ◆ Kärkkäinen & Sanders '03, Ko & Aluru '03, Kim et al. '03.
- ◆ Burkhardt & Kärkkäinen '03

□ 高さ配列

- ◆ その作成法
- ◆ 接尾辞木の作成法
- ◆ それを用いた検索法

□ 次回

- ◆ 接尾辞木・接尾辞配列の応用のいろいろ