北海道大学
Hokkaido University

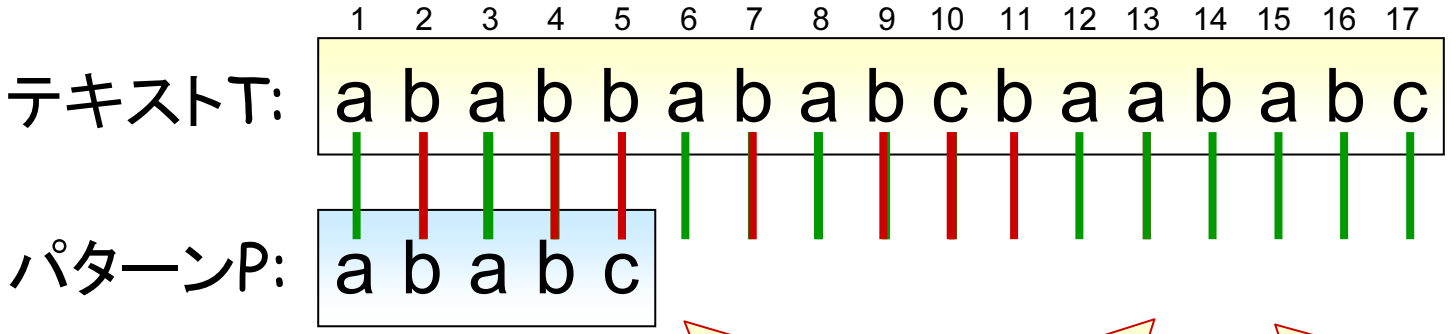
情報知識ネットワーク特論
「情報検索とパターン照合」

情報科学研究科 コンピュータサイエンス専攻
情報知識ネットワーク研究室
喜田拓也

第3回
Suffix型アルゴリズム

Boyer-Moore アルゴリズム
Galil アルゴリズム
Horspool アルゴリズム
Sunday アルゴリズム

Naïve アルゴリズム



パ
at po

パターン出現！
at position 13 of T

一文字づつずらして
マッチングしていく

テキスト上のポインタ
（比較する文字の現在
位置）が前後する！

```
Naïve-String-Matching (T, P)
1   n ← length[T].
2   m ← length[P].
3   for s ← 0 to n – m
4      do if P[1..m] = T[s+1…s+m]
5           then report an occurrence at s.
```

もっと大雑把
にいえば、
O(nm) 時間

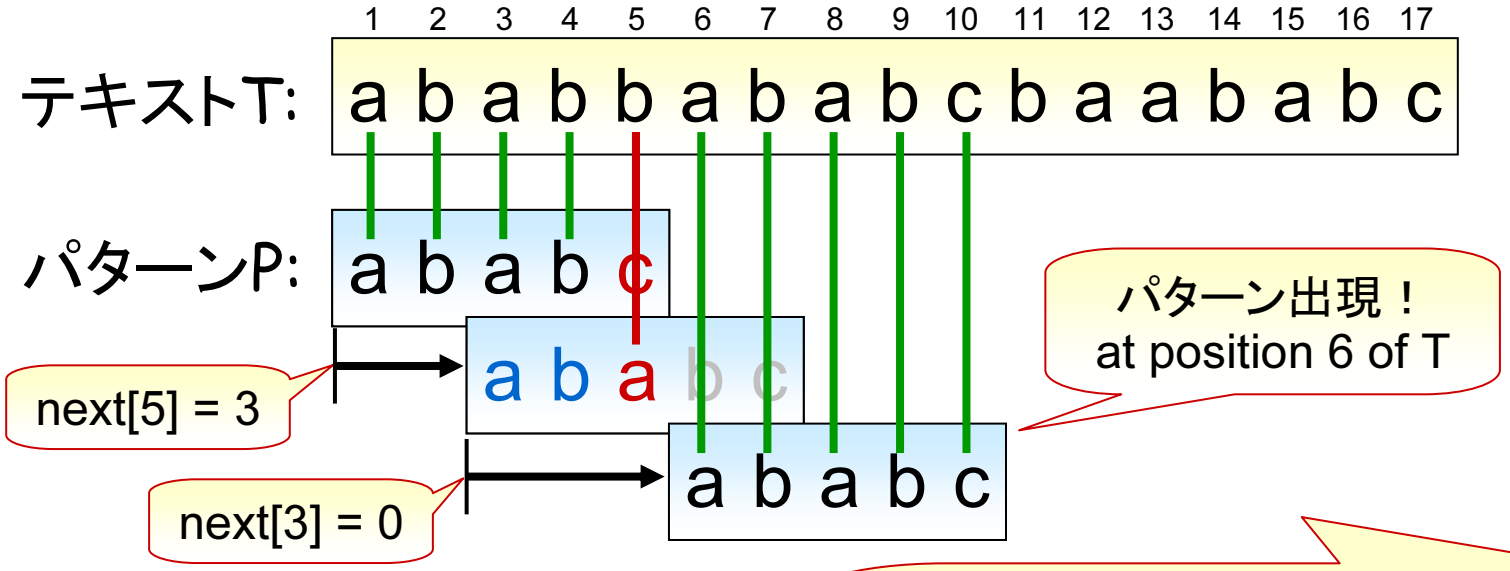
最悪の場合 $O((n-m+1)m)$ 時間かかる。

※演習：$T=a^8$, $P=a^4b$ の場合を文字比較の回数は何回か？

P=aaaab の意味

Knuth-Morris-Pratt アルゴリズム

D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt. Fast pattern matching in strings.
SIAM Journal on Computing, 6(1):323-350, 1977.




```
KMP-String-Matching (T, P)
1    n ← length[T].
2    m ← length[P].
3    q ← 1.
4    next ← ComputeNext(P).
5    for i ← 1 to n do
6       while q>0 かつ P[q]≠T[i] do q ← next[q];
7       if q=m then report an occurrence at i-m;
8       q ← q+1.
```

next関数によって次にPの何文字目とテキストを比較するかがわかる。(シフト量は q−next[q])
値が0のときは、テキストの次の文字と比較する。
テキストの各文字との比較はO(1)回ずつである。

最悪の場合でも O(n+m) 時間（nextはあらかじめ配列として計算）

効率的照合アルゴリズムの一般形

```
MatchingAlgorithm (P, T)
1    m ← length[P].
2    n ← length[T].
3    i ← 1.
4    while i ≦ n – m +1 do
5        i が出現位置であるか否かを決定する;
6        if i が出現位置 then report an occurrence at i;
7        パタンを右へシフトする量Δを求める;
8        i ← i + Δ.
```

KMP法、BM法をはじめとする多くの効率的パタン照合アルゴリズムがこの枠組みに入る※

※竹田正幸「全文テキスト処理のための高速パターン照合アルゴリズム」、情報学シンポジウム、1991年1月.

アルゴリズムの高速化のために大事なこと:
- 5行目をいかに最小の手間で決定できるか
- 7行目においてシフト量をどれだけ大きくできるか

Boyer–Moore アルゴリズム

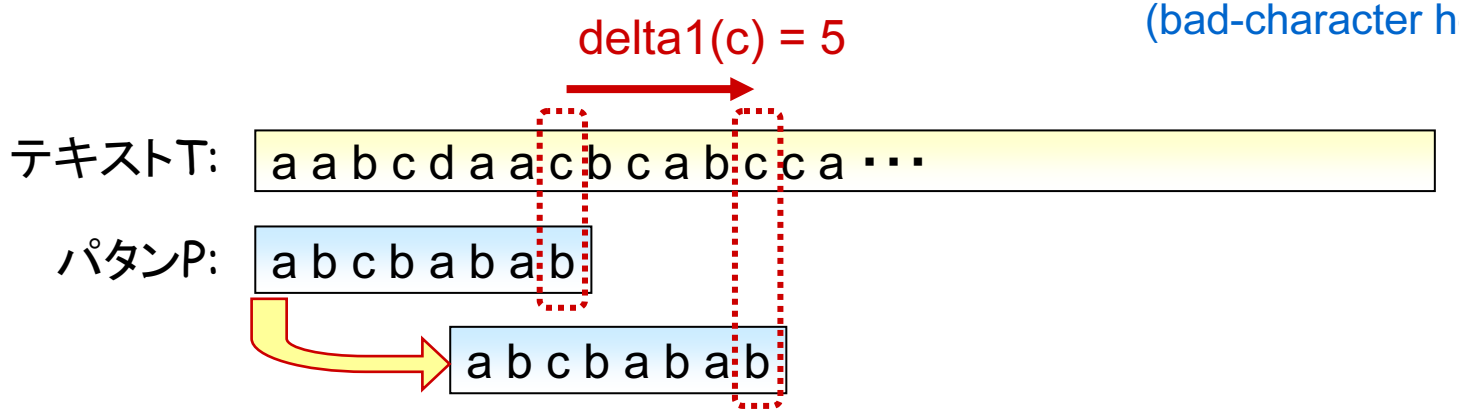
R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762-772, 1977.

特徴:

- パタンの右から左へ文字を比較していく
- 二つの関数(delta1とdelta2)の値を比較し、より大きいほうを使ってパタンをシフトする
- 最悪 O(mn) 時間だが、平均的には O(n/m) 時間となる (sub linear!!)

delta1(char) := パタン内のcharの最右の出現位置に合わせるようにシフトした際のポインタ(文字比較位置)のとび幅 (出現しない場合はパタン長)

(bad-character heuristic)

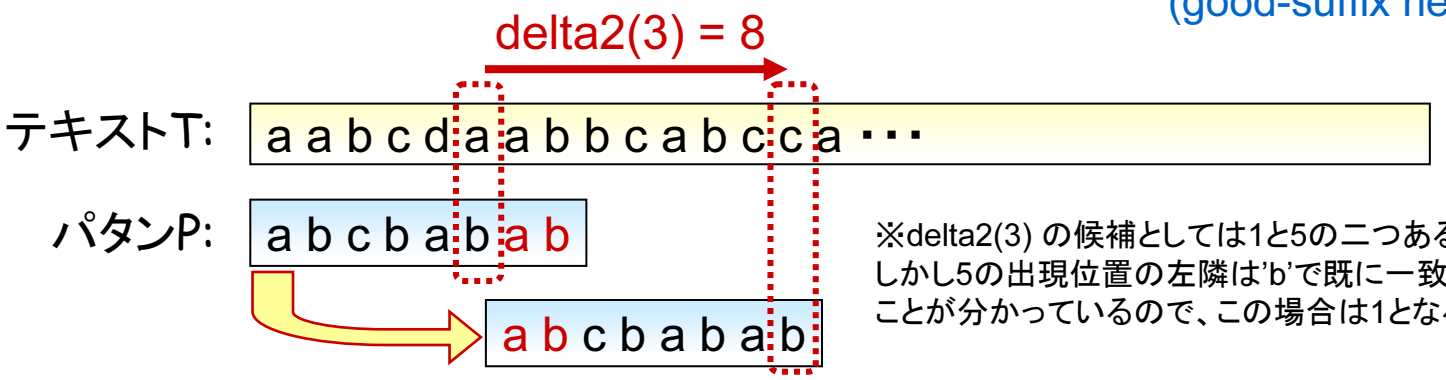


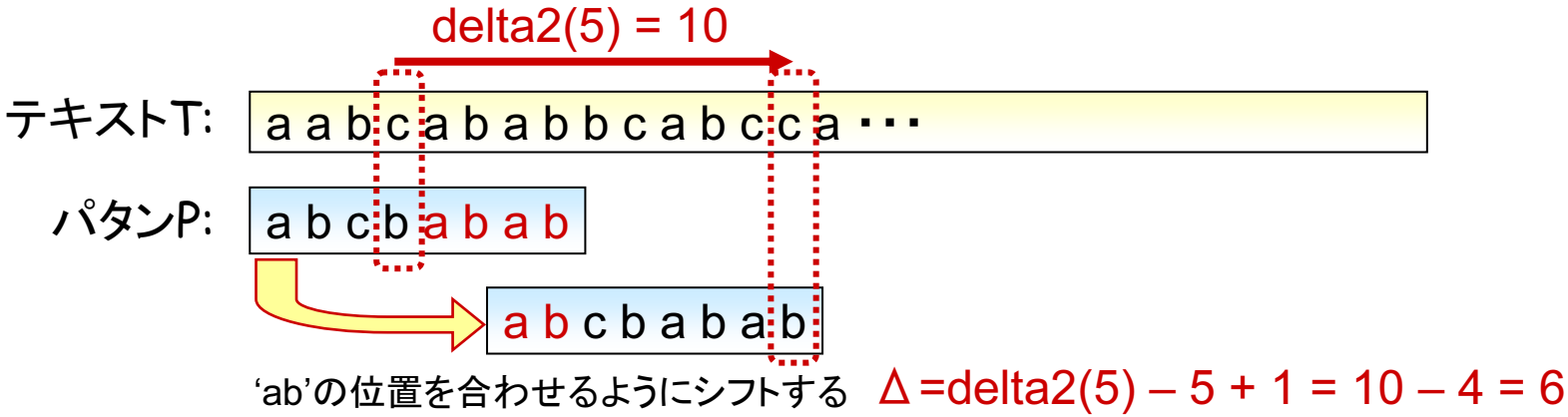
'c'の位置を合わせるようにシフトする Δ=delta1(char) – j + 1 = 5 – 0 = 5

delta2(j)

delta2(j) := パタンPの長さ j-1 のsuffixのP中の他の出現位置（あるいは最長一致するPrefix）に合わせた際のポインタのとび幅（出現しない場合はパタン長）

(good-suffix heuristic)





BM法の問題点

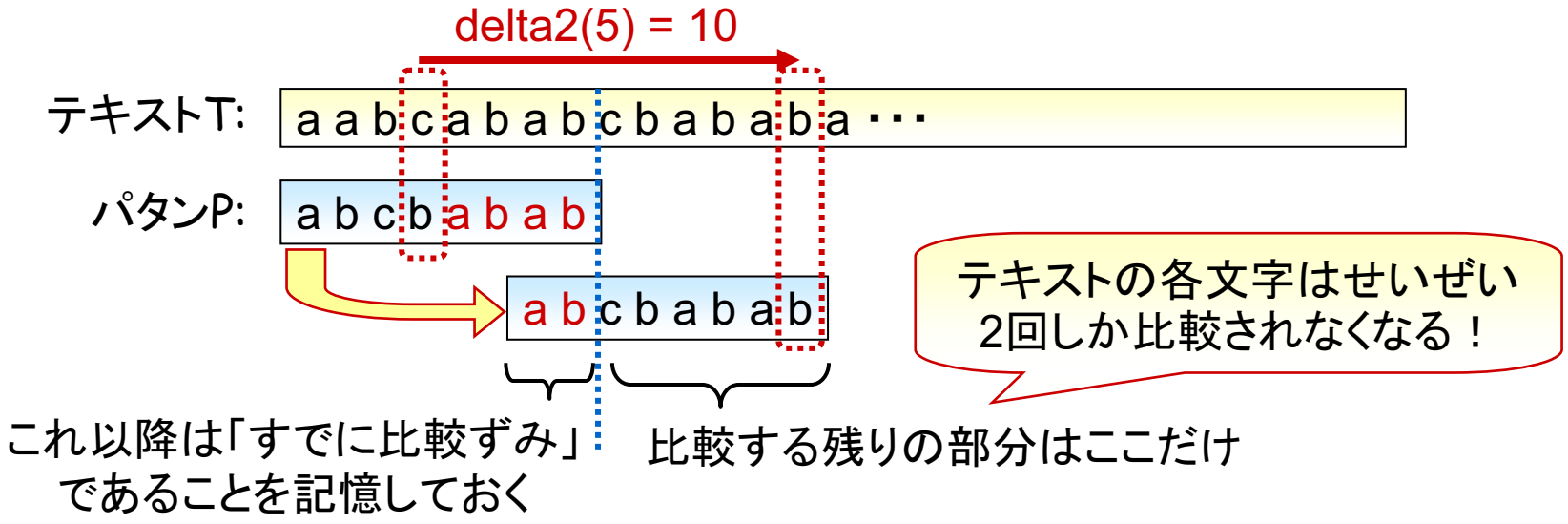
- delta関数を計算するのが難しい
  - 単純なやり方だと $O(m^2)$ 時間かかる
  - $O(m)$ 時間の手法はひと手間かかる → KMPの裏返し
- delta1 と delta2 の値をいちいち比較するので、手間がかかりすぎる
  - delta1 だけを用いる方法が一般的
    （ただしそのままでは、パタンがうまくシフトできないことがあるので、工夫が必要）
- 最悪の場合には、$O(mn)$ 時間かかってしまう
  - $T = a^n$, $P = ba^m$ の場合を考えよ
- アルファベットのサイズが小さいときには効率が悪い
  - テキストもパタンも$\Sigma=\{0,1\}$の場合は、ほとんどシフトできない！

つまり、バイナリテキスト

Galil アルゴリズム

Z. Galil. On improving the worst case running time of the Boyer-Moore string searching algorithm.
Communications of the ACM, 22(9):505-508, 1979.

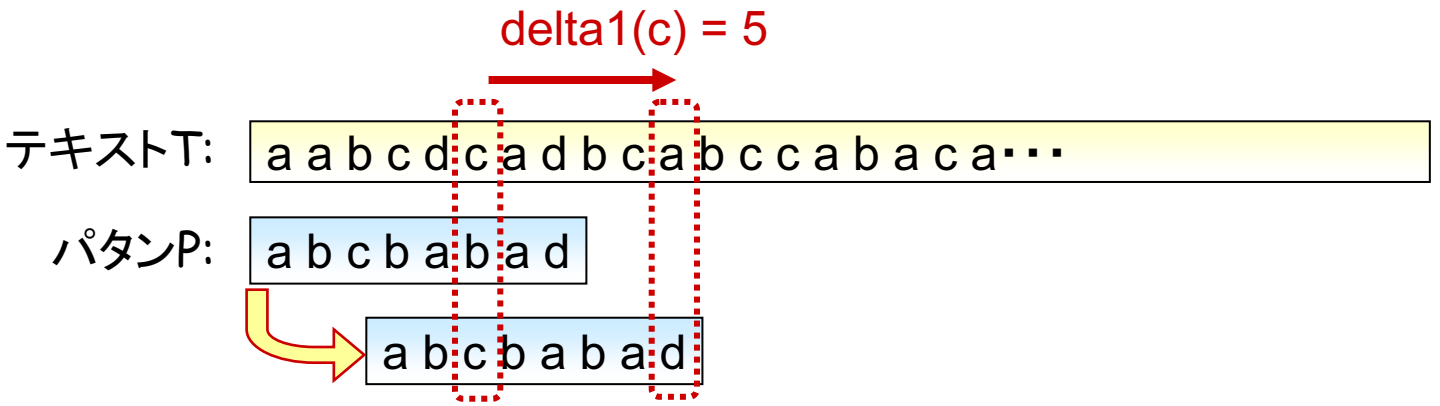
- 元のBM法では、一致した文字列の情報を「忘れてしまう」ので、O(mn)時間かかる
- Prefixが何文字一致しているかを記憶しておけばよい
- 理論的には、テキスト走査をO(n)時間で行える
- 実際には処理が煩雑になり、遅くなる

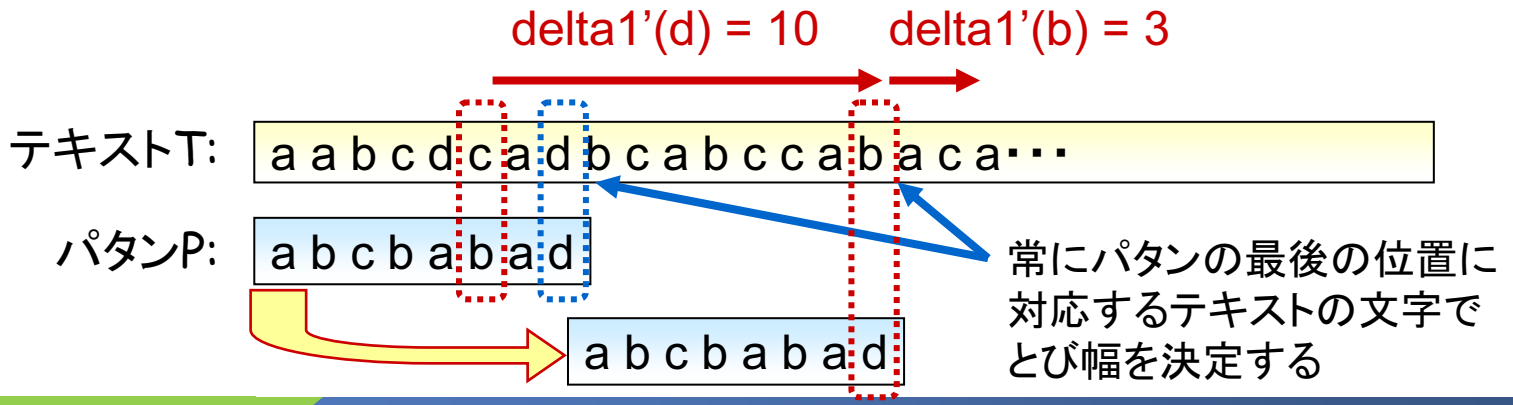


Horspool アルゴリズム

R. N. Horspool. Practical fast searching in strings. *Software Practice and Experience*, 10(6):501-506, 1980.

- Σが十分に大きい場合は、delta1関数（bad-character heuristic）が大抵の場合一番よいシフト量を与える
  → 少しの変更を加えることで、よりとび幅を増やすことができる！





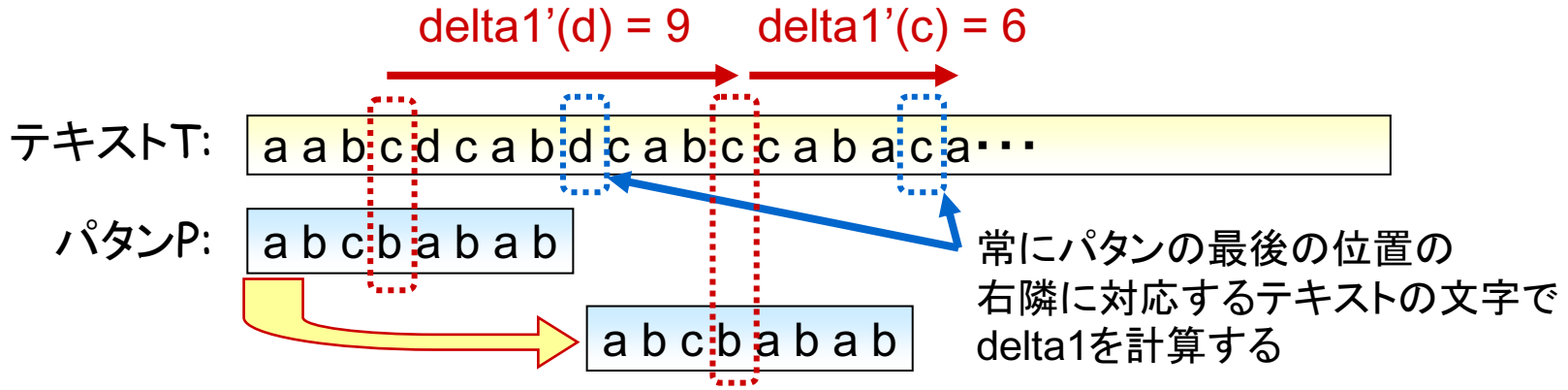
擬似コード

```
Horspool (P, T)
1    m ← length[P].
2    n ← length[T].
3    Preprocessing:
4        For each c∈Σ do delta1'[c] ← m.
5        For j←1 to m – 1 do delta1'[ P[j] ] ← m – j .
6    Searching:
7        i ← 0.
8        while i ≦ n – m do
9            j ← m;
10           while j > 0 かつ T[i+j] = P[j] do j ← j – 1;
11           if j = 0 then report an occurrence at i+1;
12           i ← i + delta1'[ T[i+m] ].
```

Sunday アルゴリズム

D. M. Sunday. A very fast substring search algorithm. *Communications of the ACM*, 33(8):132-142, 1990.

- 基本はBM型アルゴリズムと同じ
- 異なる点
  - パタンが一致するか否かを、パタン中の任意の文字順で比較する
    - 例えば、統計的に出現頻度の低い文字から順次比較を行う
  - delta1を引く際、パタンの最後の位置の右隣に対応するテキスト上の文字を使う（BMにおけるdelta2も計算し、長いほうを選択する）
- Horspoolよりもとび幅は長くなる傾向がある
  - ただし、メモリ消費量はHorspoolより大きい
  - また、とび幅を計算する手間がHorspoolよりかかる



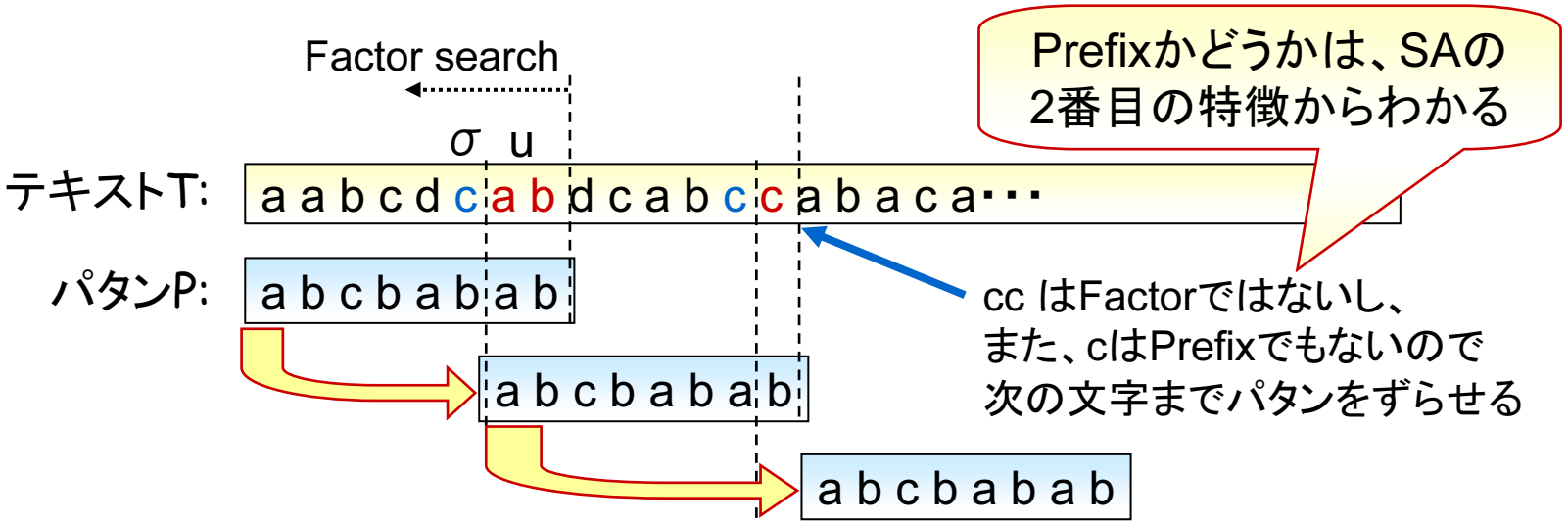
Factor型アルゴリズム

BDM アルゴリズム
BOM アルゴリズム
BNDM アルゴリズム

Backward Dawg Matching (BDM)アルゴリズム

M. Crochemore, A. Czumanj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter. Speeding up two string matching algorithms. *Algorithmica*, 12(4/5):247-267, 1994.

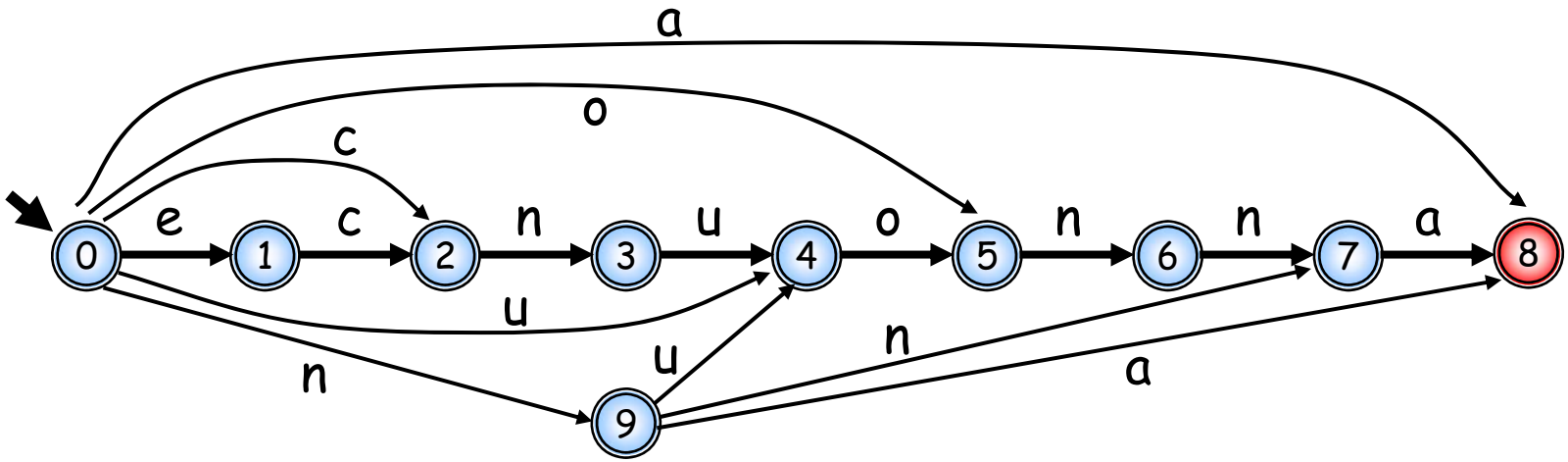
- 基本はBM型アルゴリズムと同じ
- 異なる点
  - パタンのSuffixと一致しているかではなく、Factorと一致しているかどうかでパタンの出現を判定する
  - Factorかどうかの判定はSuffix Automatonを使う(suffix treeでも可)
- Suffix automaton (SA) の特徴
  - 文字列 u がパタンPのFactorであるかどうかがO(|u|)時間で分かる
  - 文字列 u がパタンPのSuffixであるかどうかも判定できる
  - $P=p_0p_2\cdots p_m$ に対して、O(m)時間のオンラインアルゴリズムがある



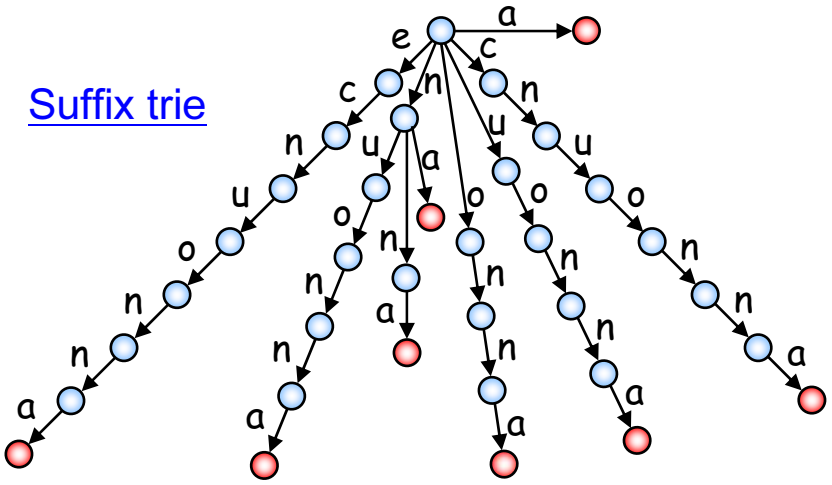
Suffix Automaton

A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. T. Chen and J. Seiferas. The smallest automation recognizing the subwords of a text. *Theoretical Computer Science* (40):31-55, 1985.

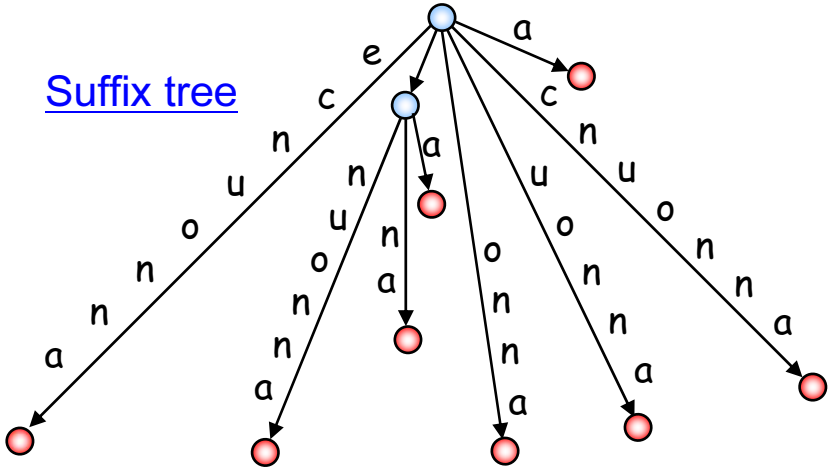
Suffix automaton　パタンP = announce の反転P^Rのfactorを受理する決定性オートマトン



Suffix trie



Suffix tree



On-line 構築アルゴリズム

```
SuffixAutomaton(P=p1p2…pm)
1    Create the one-node graph G=DAWG(e).
2    root ← sink ← the node of G. suf[root] ← θ .
3    for i ← 1 to m do
4        create a new node newsink;
5        make a solid edge (sink, newsink) labeled by a;
6        w ← suf[sink];
7        while w≠ θ かつ son(w,a)= θ do
8            make a non-solid a-edge (w, newsink);
9            w ← suf[w];
10       v ← son(w,a);
11       If w= θ then suf[newsink] ← root
12       else if (w,v) is a solid edge then suf[newsink] ← v
13       else
14           create a node newnode;
15           newnode has the same outgoing edges as v except that they are all non-solid;
16           change (w,v) into a solid edge (w, newnode);
17           suf[newsink] ← newnode;
18           suf[newnode] ← suf[v]; suf[v] ← newnode;
19           w ← suf[w];
20           while w≠ θ かつ (w,v) is a non-solid a-edge do
21               redirect this edge to newnode; w ← suf[w].
22       sink ← newsink.
```

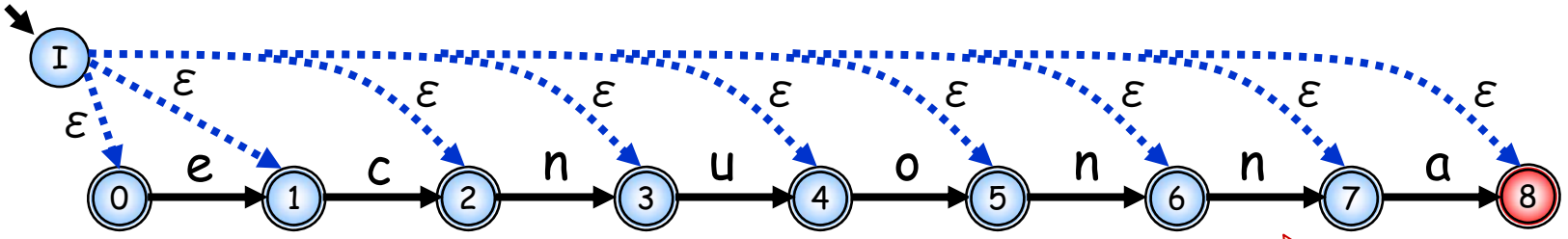
とっても複雑なので、
構築するのは
結構たいへん！

BNDM アルゴリズム

G. Navarro and M. Raffinot. Fast and flexible string matching by combining bit-parallelism and suffix automata. ACM Journal of Experimental Algorithmics (JEA), 5(4), 2000.

- 基本はBDMと同じ
- 異なる点
  - パタンのfactorかどうかを調べるため、
    非決定性（nondeterministic）のSuffix automataを用いる
  - そのNFAの状態遷移をBit-parallel手法でシミュレートする

パタンP = announce の反転P^Rのsuffixを受理する非決定性オートマトン



このNFAを
シミュレートする

初期状態：$R^0 = 1^m$

状態遷移：R = (R << 1) & M[T[i]]

Shift-Andと同じ
Mask table

擬似コード

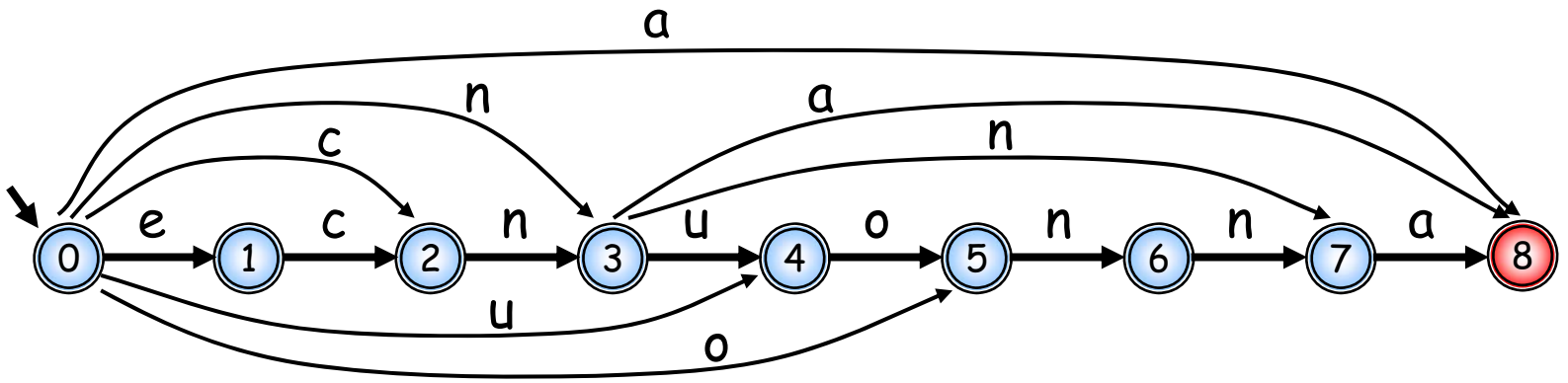
```
BNDM (P, T)
1    m ← length[P].
2    n ← length[T].
3    Preprocessing:
4        for c ∈ Σ do M[c] ← 0^m.
5        for j ← 1 to m do M[ P[j] ] ← M[ P[j] ] | 0^(m–j)10^(j–1).
6    Searching:
7        s ← 0.
8        while s ≦ n – m do
9            j ← m, last ← m, R ← 1^m;
10           while R ≠ 0^m do
11               R ← (R << 1) & M[ T[s+j] ];
12               j ← j – 1;
13               If R & 10^(m-1) ≠ 0^m then
14                   If j > 0 then last ← j;
15                   else report an occurrence at s+1;
16               R ← R << 1;
17           s ← s + last.
```

Backward Oracle Matching (BOM)アルゴリズム

C. Allauzen, M. Crochemore, and M. Raffinot. Efficient experimental string matching by weak factor recognition. *In Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching*, LNCS2089:51-72, 2001.

- BDMとアイデアは同じ
- 異なる点
  - 複雑なSuffix automatonではなく、Factor oracleを使う
  - BDMにおいて必要なことは、文字列uがFactorであることではなく、σuがFactorではないこと。
- Factor oracleの性質
  - パタンPのFactor以外の文字列も受理してしまう可能性がある
    - 例：下の図で、cnnはP^RのFactorではない
  - O(m)時間で構築できるうえに、実装が容易で少メモリ
    - 状態数m+1個、遷移関数の実現サイズ2m−1

P=announceの場合の、P^Rに対するFactor oracle



Factor oracleの構築アルゴリズム

Oracle-on-line ($P=p_1p_2\ldots p_m$)
1 Create Oracle(ε) with
2 One single initial state 0, $S(0) \leftarrow \theta$.
3 **for** $i \in 1\ldots m$ **do**
4 Oracle($P=p_1p_2\ldots p_j$)
5 $\leftarrow$ **Oracle_add_letter** (Oracle($P=p_1p_2\ldots p_{j-1}$), p_j).

Oracle_add_letter (Oracle($P=p_1p_2\ldots p_m$), σ)
1 Create a new state m+1.
2 $\delta(m, \sigma) \leftarrow m+1$.
3 $k \leftarrow S(m)$
4 **while** $k \neq \theta$ かつ $\delta(k, \sigma) = \theta$ **do**
5 $\delta(k, \sigma) \leftarrow m+1$;
6 $k \leftarrow S(k)$.
7 **If** $k = \theta$ **then** $s \leftarrow 0$;
8 **else** $s \leftarrow \delta(k, \sigma)$.
9 $S(m+1) \leftarrow s$.
10 return Oracle($P=p_1p_2\ldots p_m \sigma$).

ちょっと、ひといき・・・

- ここまでのまとめ
  - Suffix型アルゴリズム
    - パターンを右から左へ照合する！
    - テキストの文字を飛び飛びで照合できる！
  - Factor型アルゴリズム
    - パターンのすべてのFactorと比較しつつ、テキストの文字を飛び飛びで照合できる！

いわとびペンギン（旭山動物園にて '05.3.13）

～トリビア～

二つの整数xとyの内容を余計な変数を用いずに交換できるか？

Suffix･Factor型の複数パターンへの拡張

Set Horspool アルゴリズム
Wu-Manberアルゴリズム

Suffix・Factor型 複数パターン照合

Commentz-Walterアルゴリズム

B. Commentz-Walter. A string matching algorithm fast on the average. In Proceedings of the 6th International Colloquium on Automata, Languages and Programming, LNCS71:118-132, 1979.

- BMアルゴリズムの直接的な拡張

Set Horspoolアルゴリズム

- Commentz-WalterのアルゴリズムをHorspoolのアイデアに基づいて簡略化したもの

Uratani-Takedaアルゴリズム

- ACアルゴリズムのアイデアをBM型に転用したもの。CWより高速

Set Backward Oracle Matching (SBOM)アルゴリズム

C. Allauzen and M. Raffinot. Factor oracle of a set of words.
Techinical report 99-11, Institut Gaspard-Monge, Universite de Marne-la-Vallee, 1999.

- Factor oracleを複数文字列に拡張したものを利用。

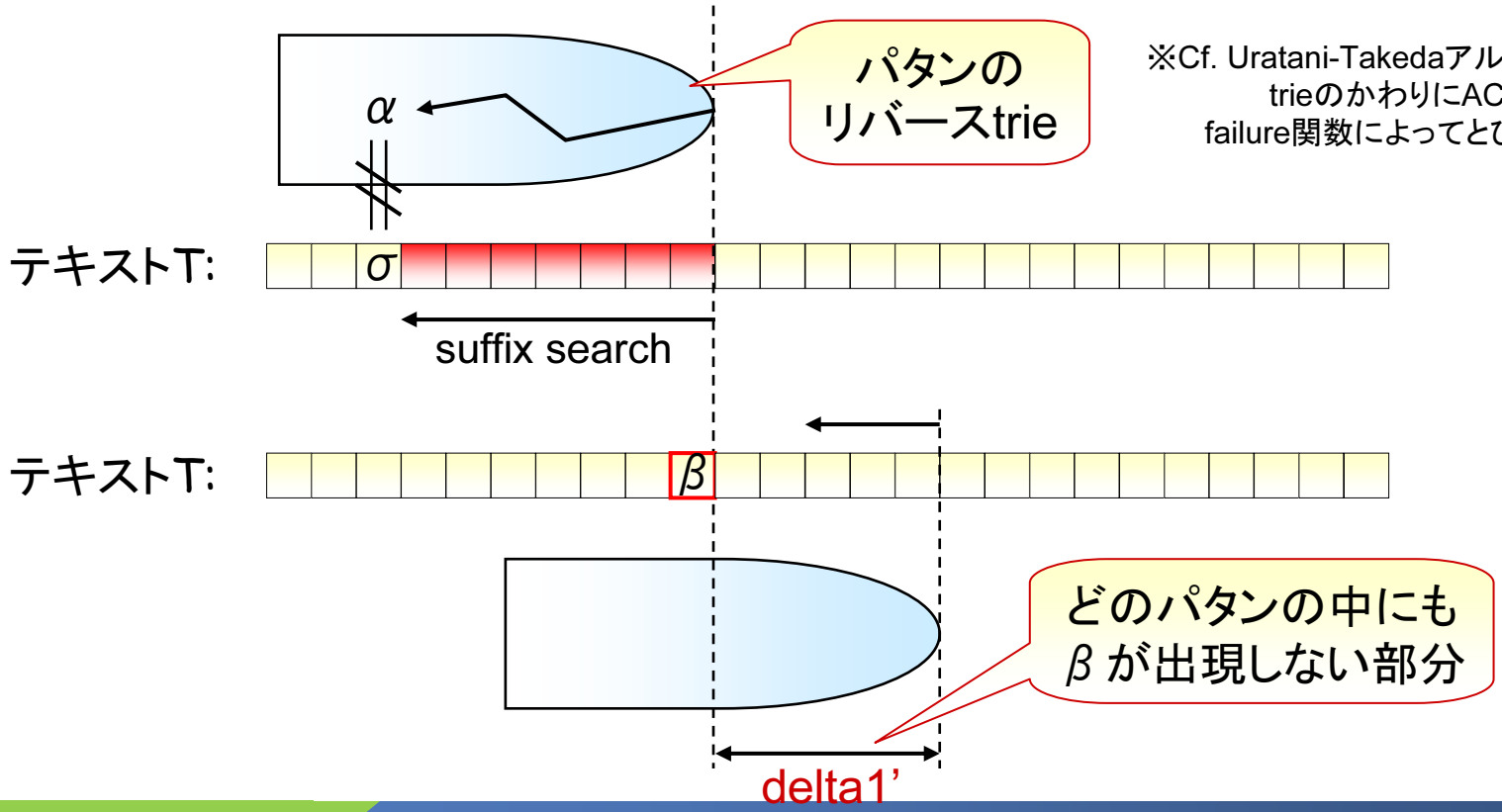
Wu-Manberアルゴリズム

S. Wu and U. Manber. A fast algorithm for multi-pattern searching.
Report TR-94-17, Department of Computer Science, University of Arizona, Tucson, AZ, 1994.

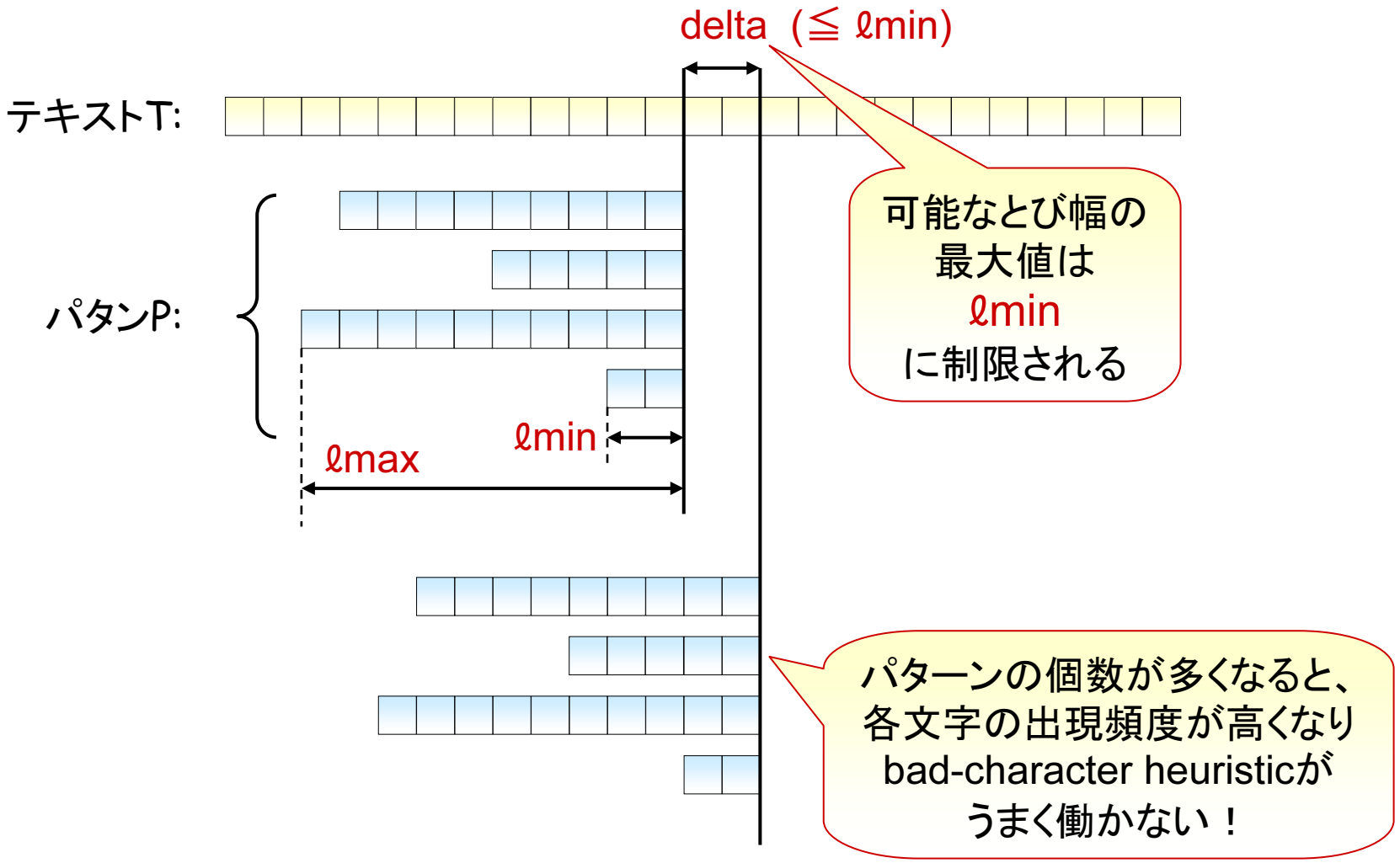
- 実用的に高速なアルゴリズム。Agrepにも用いられている

Set Horspool アルゴリズム

- パタン集合の各要素を反転(reverse)した文字列のtrieを作る
- あとはHorspoolと同じ
  - Suffix search をしながらtrieをたぐる
  - どのパタンのSuffixでもないことが判ったら、delta1’でパタンをシフトする



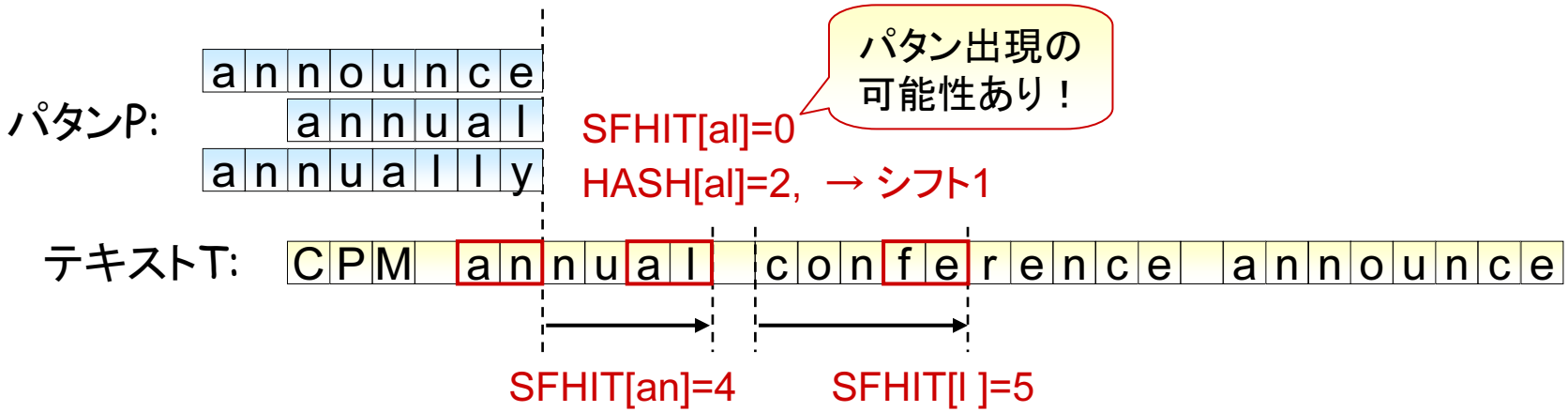
パフォーマンスが悪くなる理由



Wu-Manberアルゴリズム

S. Wu and U. Manber. A fast algorithm for multi-pattern searching.
Report TR-94-17, Department of Computer Science, University of Arizona, Tucson, AZ, 1994.

- テキストの照合位置からB文字分(T[i−B+1…i])を用いて、パタンが出現する可能性を調べる
  - SHIFT[T[i−B+1…i]]： T[i−B+1…i]が、あるパタンの接尾辞であるとき0。そうでなければ、可能な最大シフト長を返す。
  - HASH[T[i−B+1…i]]： SHIFTが0（すなわちT[i−B+1…i]があるパタンの接尾辞）だった場合、出現の可能性があるパタンのリストを返す。



SHIFT[BI] =

文字列	ll	no ou	an	un nc	ua al	ly	nn nu	ce	*
シフト量	1	3	4	1	0	0	2	0	5

HASH[BI] =

文字列	ce ly	ua al	*
パタン番号	3, 1	2	ϕ

擬似コード

※agrep ver4.02 の実装(mgrep.c)では、SFHIT・HASH・Bはそれぞれ、4096、8192、3となっている（ようだ）

```
Construct_SHIFT (P={p^1,p^2,…,p^r})
1    initialize SHIFT table by ℓmin–B+1.
2    For each Bl=p^i[j–B+1…j] do
3        If SHIFT[h1(Bl)] > m_i – j do SHIFT[h1(Bl)] = m_i – j.
```

```
Wu-Manber (P={p^1,p^2,…,p^r}, T=T[1…n])
1    Preprocessing:
2        Computation of B.
3        Construction of the hash tables SHIFT and HASH.
4    Searching:
5        pos ← ℓmin.
6        while pos≦n do
7            i ← h1( T[pos–B+1…pos] );
8            If SHIFT[i] = 0 then
9                list ← HASH[ h2( T[pos–B+1…pos] ) ];
10               Verify all the patterns in list one by one against the text;
11               pos ← pos + 1;
12           else pos ← pos + SHIFT[i].
```

第3回 まとめ

- Suffix型アルゴリズム
  - パターンを右から左へ照合する
  - 最悪 O(mn) 時間だが、平均的には O(n/m) 時間
  - Boyer-Moore、Galil、Horspool、Sunday らのアルゴリズム
- Factor型アルゴリズム
  - パターンのFactorかどうかを判定して、テキストを飛び飛びに照合する
  - BDM、BNDM、BOMアルゴリズム
- Suffix型・Factor型アルゴリズムの複数パターンへの拡張
  - パターンの個数が多くなると、各文字の出現頻度が高くなり bad-character heuristicがうまく働かない
  - Set Horspool、Wu-Manberアルゴリズム
- 次回のテーマ
  - 近似文字列照合アルゴリズム： 誤りを許したパターン照合