講義「アルゴリズムとデータ構造」

第x回　文字列照合アルゴリズム

大学院情報科学研究科 情報理工学専攻
情報知識ネットワーク研究室
喜田拓也

2018/5/23

パターン照合問題とは？

テキスト T 中に含まれるパターン P の出現を求める問題

パターン P: compress

テキスト T:

We introduce a general framework which is suitable to capture an essence of compressed pattern matching according to various dictionary based compressions. The goal is to find all occurrences of a pattern in a text without decompression, which is one of the most active topics in string matching. Our framework includes such compression methods as Lempel-Ziv family, (LZ77, LZSS, LZ78, LZW), byte-pair encoding, and the static dictionary based method. Technically, our pattern matching algorithm extremely extends that for LZW compressed text presented by Amir, Benson and Farach [Amir94]..

有名なアルゴリズム：

- KMP法 (Knuth&Morris&Pratt[1974])
- BM法 (Boyer&Moore[1977])
- Karp-Rabin法 (Karp&Rabin[1987])

パターン照合問題の種類

Existence problem:

テキスト T: プルルンプルンファミファミファ

パタン P: ファミファ

Yes!

All-occurrences problem:

テキスト T: プルルンプルンファミファミファ

パタン P: ファミファ

8　11

文書を単位として検索する場合は，Existence problemで充分

本講義では，基本的にAll-Occurrences problemを取り扱う

テキストアルゴリズムの基本用語

Σ : 文字（letters, characters, またはsymbols）の空でない集合．
これをアルファベットと呼ぶ．
例）$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots, \mathrm{z}\}$, $\Sigma = \{0,1\}$, $\Sigma = \{0\mathrm{x}00, 0\mathrm{x}01, \ldots, 0\mathrm{xFF}\}$

$x \in \Sigma^*$: アルファベット中の文字を0個以上並べたもので，文字列（string）と呼ぶ．語（word）とかテキスト（text）と呼ばれることもある．

$|x|$: 文字列 x の長さ．例）$|\mathrm{aba}| = 3$.

ε : 長さが 0 の文字列．空語（empty word）と呼ぶ．$|\varepsilon| = 0$.

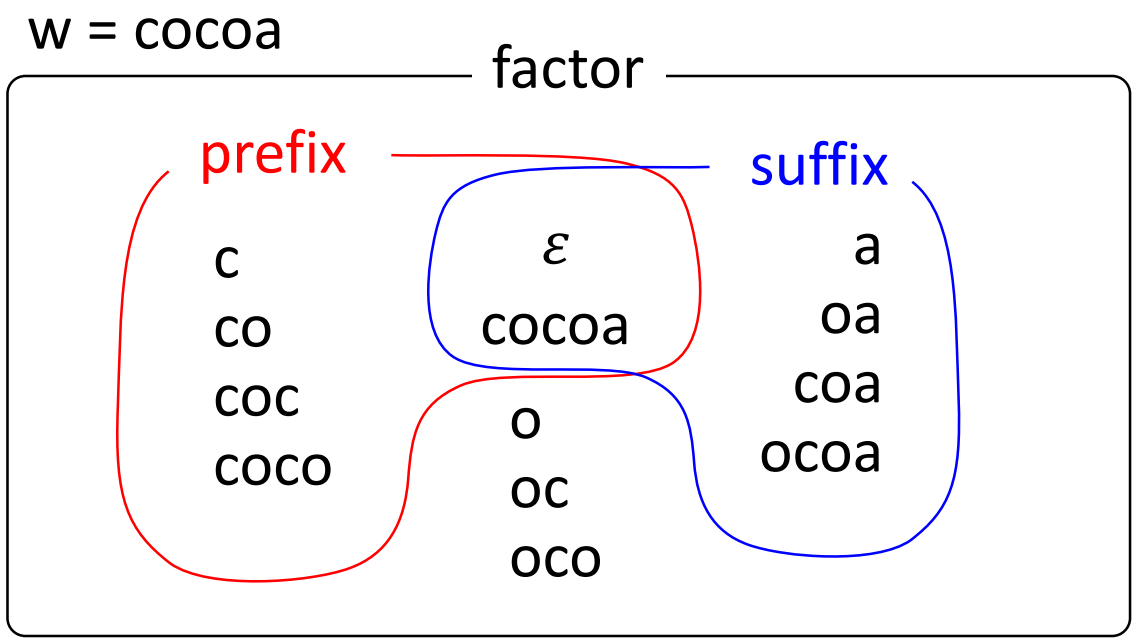
$x[i]$: 文字列 x の i 番目の文字．

$x[i:j]$: 文字列 x の i 番目から j 番目までの連続した文字のならび．ただし，$i > j$ の場合は，便宜上 $x[i:j] = \varepsilon$ とする．$x[i:j]$ を，文字列 x の部分文字列（substring, subword）またはファクター（factor）と呼ぶ．$x[i..j]$と書くこともある．

x^R : 文字列 $x = a_1 a_2 \ldots a_k \in \Sigma^*$ に対して，これを反転させた文字列．すなわち，$x^R = a_k a_{k-1} \ldots a_1$.

接頭辞(Prefix)と接尾辞(Suffix)

部分文字列(factor)のうち，$x[1:i]$ を特に x の接頭辞(prefix)という．
また，$x[i:|x|]$ を特に x の接尾辞(suffix)という．

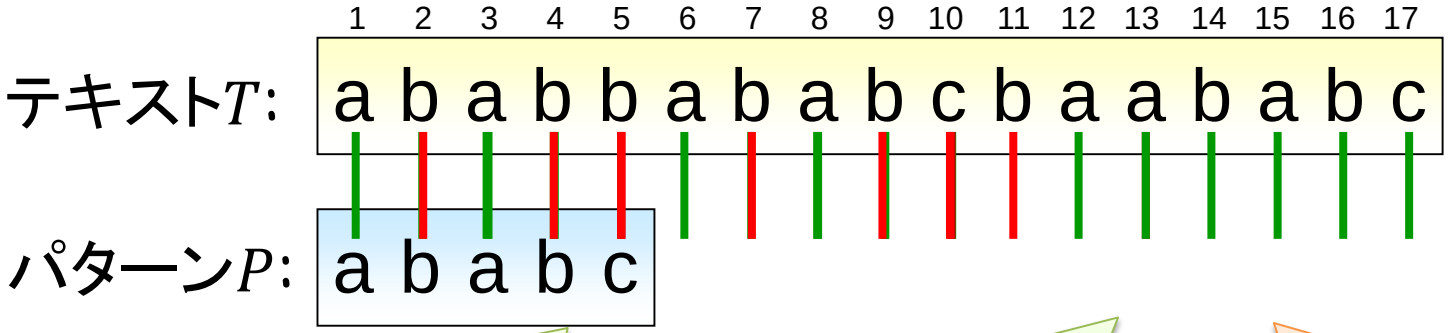


文字列 x と y について，y から 0 文字以上の文字を取り除くと，x に等しくなるとき，x を y の部分列(subsequence)という

例：$x = \mathrm{abba}$ は，$y = \mathrm{aaababaab}$ の部分列

部分文字列との違いに注意

Naïve アルゴリズム



パターン出現！
at position 6 of T

！
on 13 of T

一文字づつずらして
マッチングしていく

```
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.
```

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！

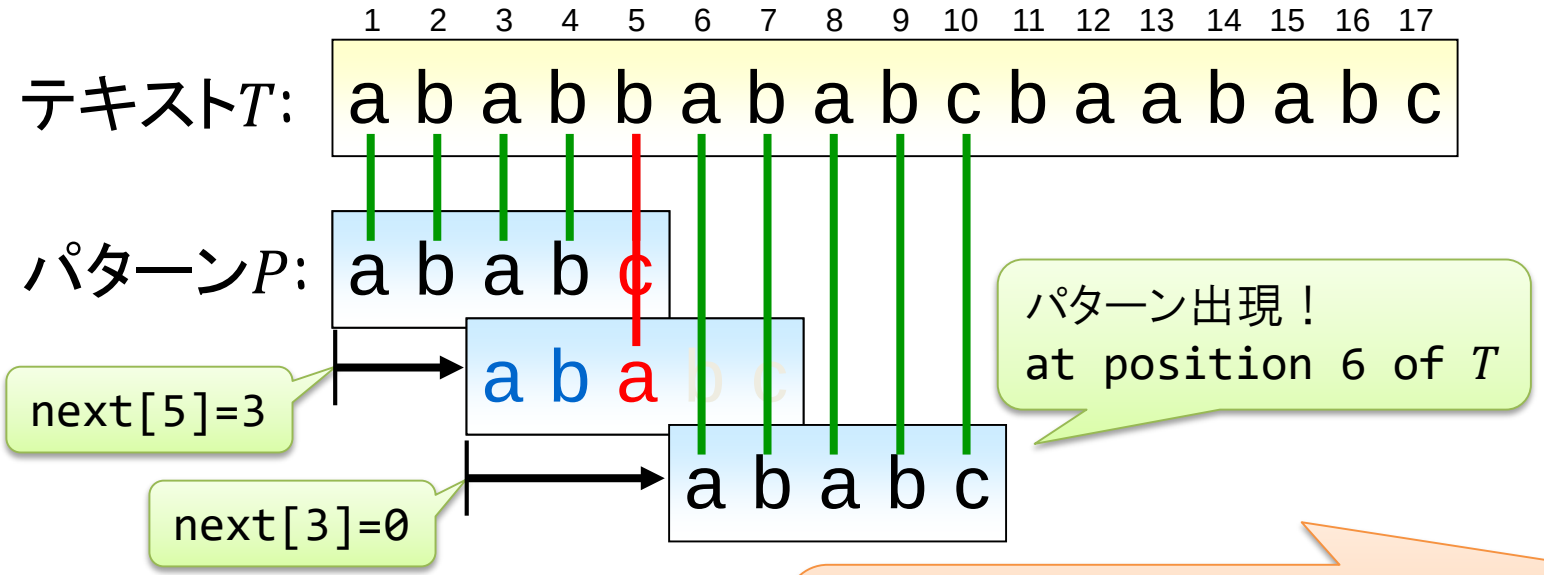
もっと大雑把
にいえば
$O(nm)$ 時間

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.



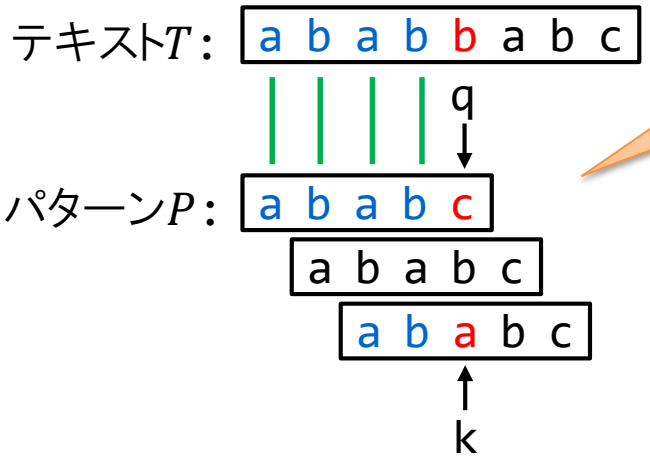

```
KMP-String-Matching(T, P)
1    n ← length[T];
2    m ← length[P];
3    q ← 1;
4    next ← ComputeNext(P);
5    for i ← 1 to n do
6       while q>0 かつ P[q]≠T[i] do q ← next[q];
7       if q=m then report an occurrence at i-m;
8       q ← q+1;
```

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

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）

next関数の計算

- next[q]=k の条件
  $P[1:k-1]$ が $P[1:q-1]$ の接尾辞かつ$P[k] \neq P[q]$を満たす最長のもの



パターンをずらしながら比較し、
next[q]を計算する

q	1	2	3	4	5	6
P[q]	a	b	a	b	c	
next(q)	0	1	0	1	3	1

$O(m)$ 時間・領域

```
ComputeNext(P)
1   m ← length[P]; next[1] ← 0; k ← 0;
2   for q ← 1 to m do
3       while k>0 かつ P[q]≠P[k] do k ← next[k];
4       k ← k+1; q ← q+1;
5       if P[q]=P[k] then
6           next[q] ← next[k]
7       else
8           next[q] ← k;
```

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は

- 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：

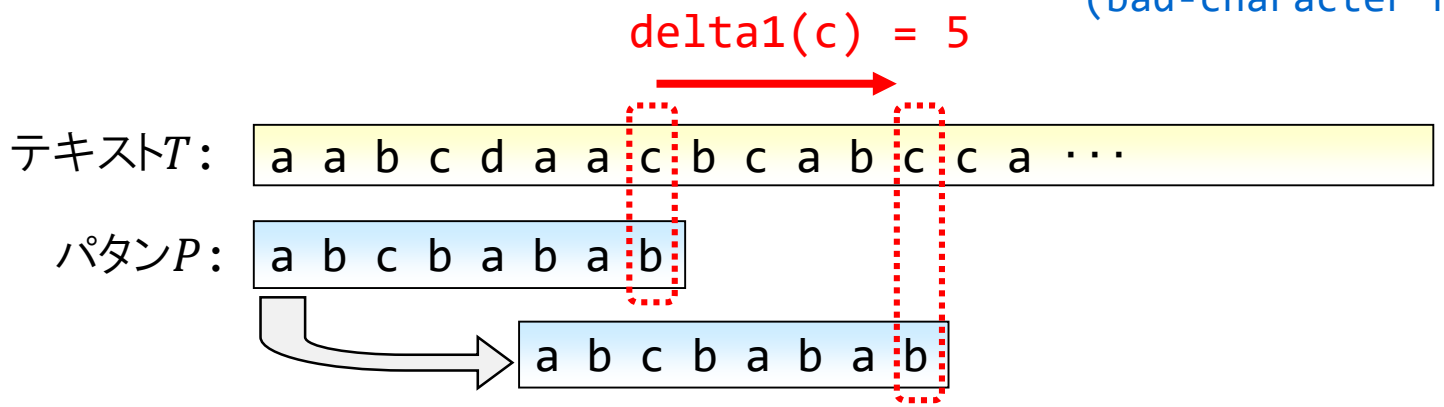
パタンの右から左へ文字を比較していく

シフト量を決めるために二つの関数（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)

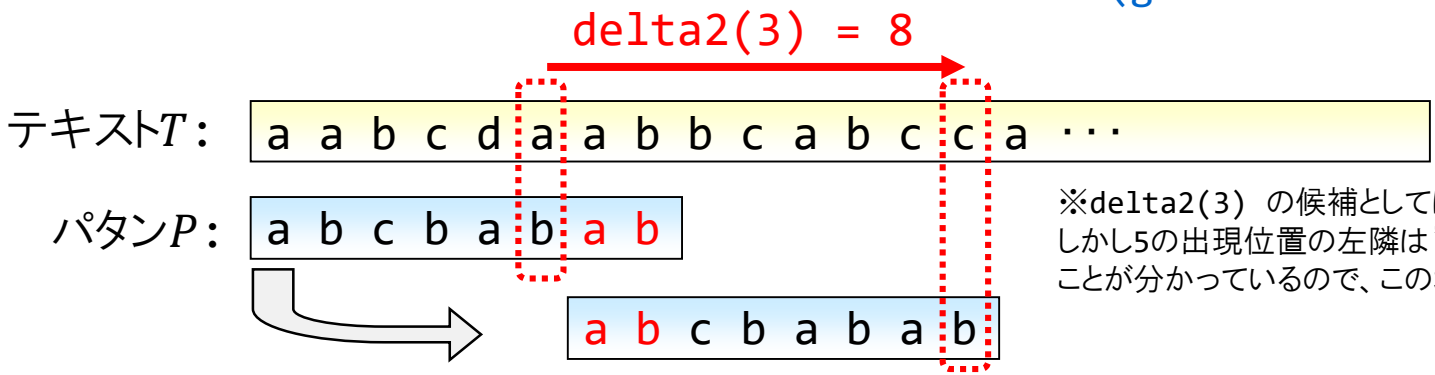


‘c’の位置を合わせるようにシフトする　　Δ=delta1(char)−j+1=5−0=5

delta2(j)

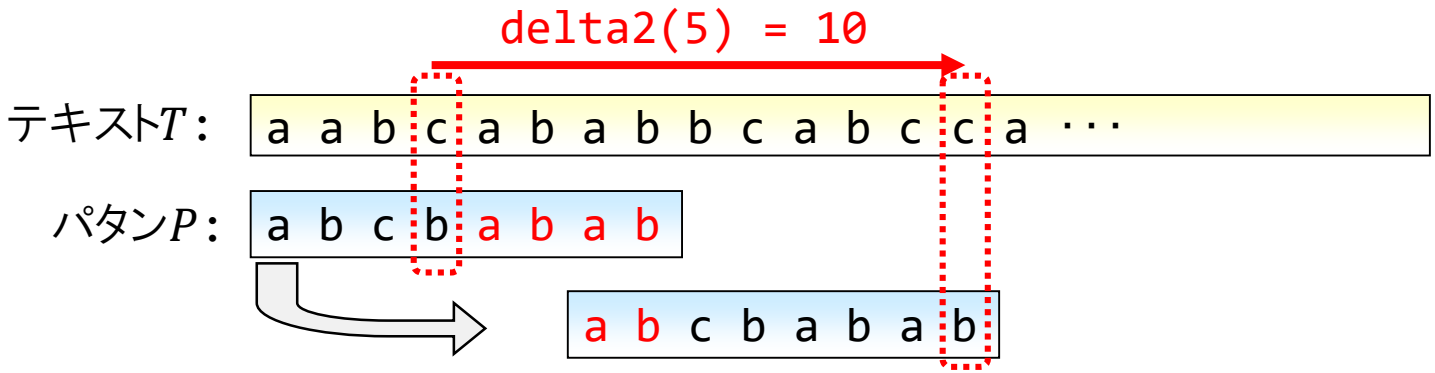
delta2(j):=パタンPの長さ j-1 のsuffixのP中の他の出現位置(あるいは最長一致するPrefix)に合わせた際のポインタのとび幅 (出現しない場合はパタン長)

(good-suffix heuristic)



※delta2(3) の候補としては1と5の二つある。
しかし5の出現位置の左隣は'b'で既に一致しないことが分かっているので、この場合は1となる。

'ab'の位置を合わせるようにシフトする　Δ = delta2(3)−3+1 = 8−2 = 6



'ab'の位置を合わせるようにシフトする　Δ = delta2(5)−5+1 = 10−4 = 6

BM法の問題点

delta2(j)関数を計算するのは意外と面倒

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の値を比較しなければならず，手間がかかる

delta1だけを用いる方法がよく用いられている

（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

例えば，$T = a^n$，　$P = ba^m$ の場合

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！

Galil アルゴリズム

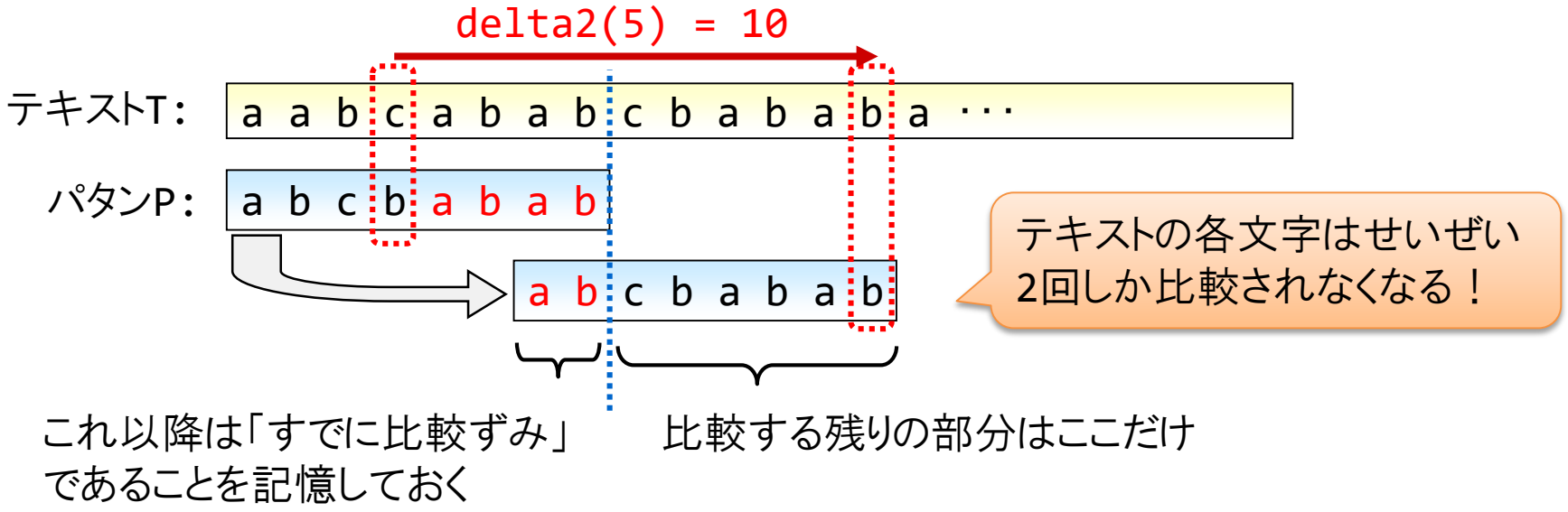
Z. Galil. On improving the worst case running time of the Boyer-Moore string searching algorithm. *Communications of the ACM*, 22(9):505-508, 1979.

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

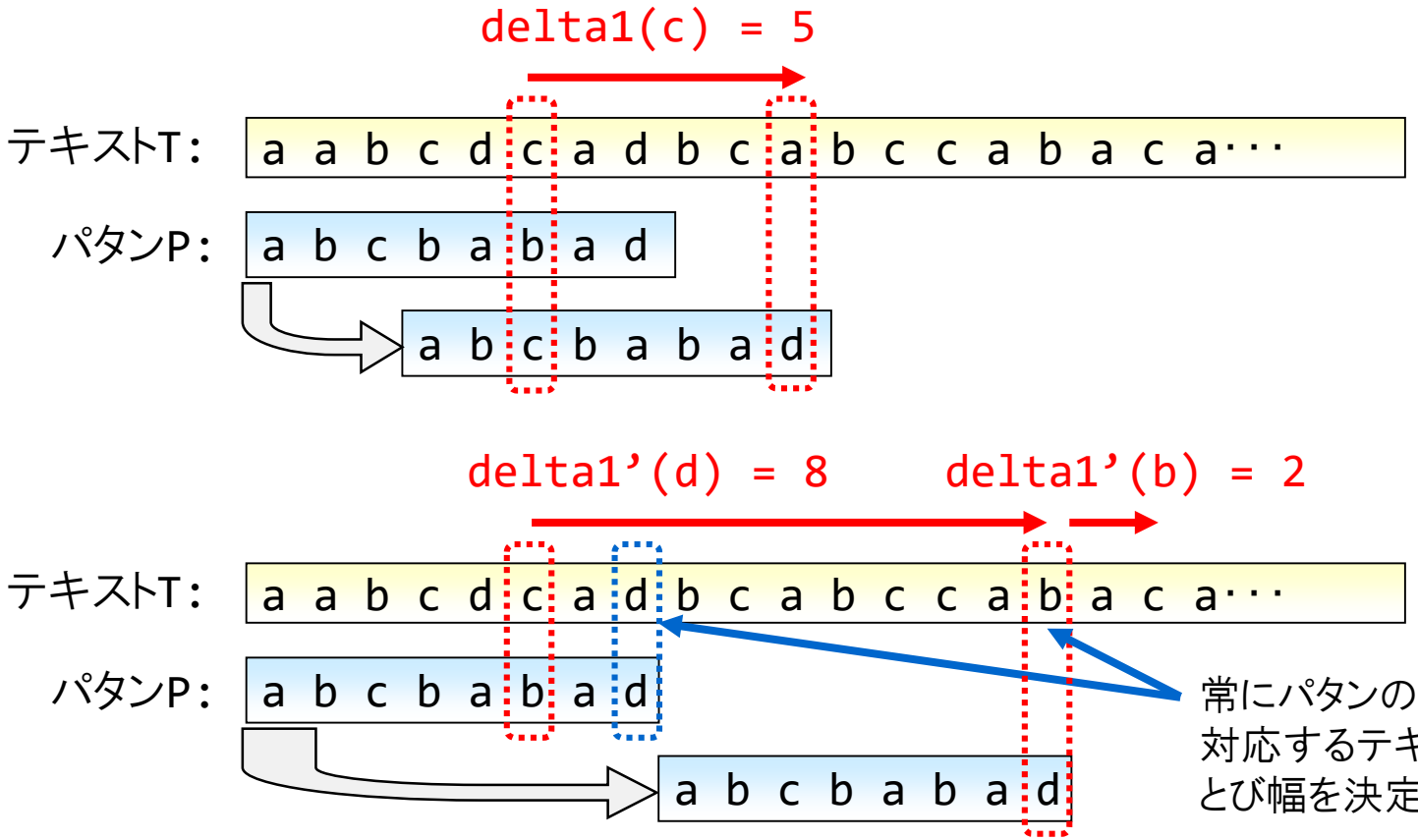


Horspool アルゴリズム

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

∑が十分に大きい場合は、delta1関数(bad-character heuristic)が大抵の場合に一番よいシフト量を与えることが知られている

→ 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∈∑ 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Horspool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
- delta1を引く際、パタン末尾の右隣に位置するテキスト上の文字を使う

Horspoolよりもとび幅は長くなる傾向がある