5. ストリングマッチング

ストリングマッチング（string matching）

比較的長い文字列（テキストストリング）t の中に，別のある文字列（パターン）p が現れているかどうかを判定し，もし現れているならばその位置を見つける処理．文字列照合とも呼ばれる．

－－－－－－－－－－－－－－－－－－－－

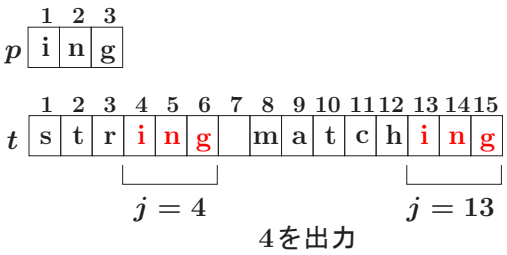
定義

- $t = t_1 t_2 \cdots t_n$，$p = p_1 p_2 \cdots p_m$ $(n \geq m)$．各 t_i, p_i はあるアルファベットから選ばれた文字．
- $p_1 p_2 \cdots p_m = t_j t_{j+1} \cdots t_{j+m-1}$ となる j があれば，p は位置 j で t に照合する（match）という．

[例 5.1]　アルファベットを { , a, b, c, ⋯, z} とする．t =string matching, p =ing とすると，p は位置 $j = 4$ と $j = 13$ で t に照合する．

－－－－－－－－－－－－－－－－－－－－

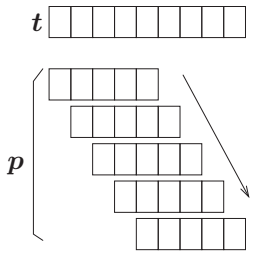
- p, t はそれぞれ配列 $p[1..m], t[1..n]$ に入っているものとする．
- p が t 中に現れるとき，照合する位置の中で最小のものを求める．



－－－－－－－－－－－－－－－－－－－－

5.1 素朴なアルゴリズム

基本的なアイデア



t と p を並べて，1文字ずつ比較していく．

文字の不一致が起これば，p 全体を1文字分ずらす．

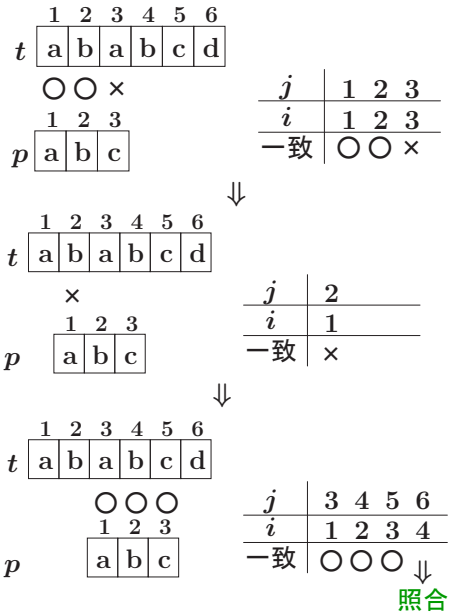
照合が見つかれば停止．

－－－－－－－－－－－－－－－－－－－－

文字 $t[j]$ と $p[i]$ を比較する．最初は $j = i = 1$．j, i の値は更新していく．

t が p を含む場合の例

t =ababcd, p =abc



－－－－－－－－－－－－－－－－－－－－

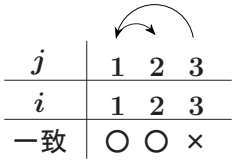
比較した文字 $t[j], p[i]$ が一致した場合の処理

- $j \leftarrow j + 1$; $i \leftarrow i + 1$ とする．
- その結果 $i = m + 1$ となれば，照合位置 $j - m$ を出力して終了．

$t[j] \neq p[i]$ となった場合の処理

- $j \leftarrow j - i + 2$; $i \leftarrow 1$ とする．

j は $i - 1$ 減らして，1増やす

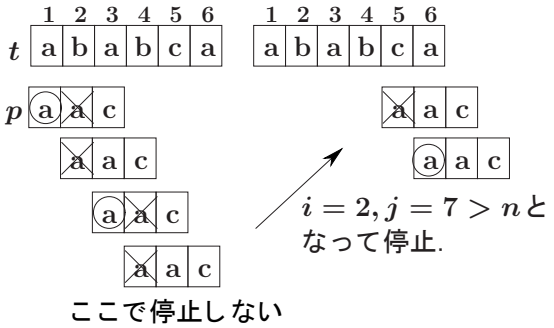


素朴なアルゴリズム（教科書 図 5.2）

```
関数 stringmatching()
int i = 1, j = 1;
while (i <= m && j <= n)
    if (p[i] == t[j]){
        i++; j++;
    }
    else{
        j = j − i + 2; i = 1;
    }
if (i == m + 1)
    printf("Found at %d\n", j − m);
else printf("Not found\n");
```

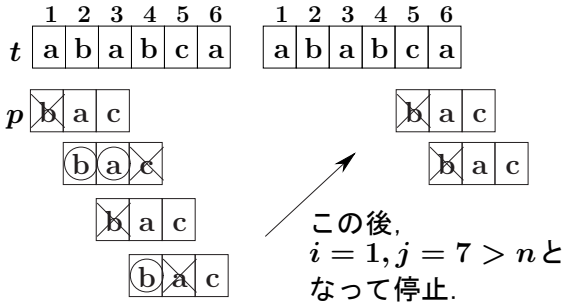
t が p を含まない場合の例 1

t =ababca, p =aac



t が p を含まない場合の例 2

t =ababca, p =bac



時間計算量

- （不必要な移動も行われるが,）p を右にずらす回数は高々 $O(n)$ 回.
- p を一回右にずらすごとにかかる時間は高々 $O(m)$ 時間.

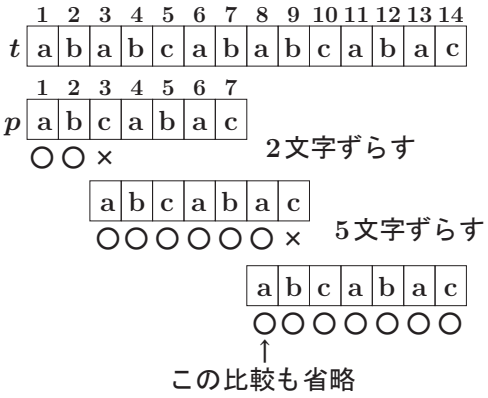
⇒ 素朴なアルゴリズム（図 5.2）の時間計算量は $O(mn)$.

5.2 クヌース・モーリス・プラットのアルゴリズム

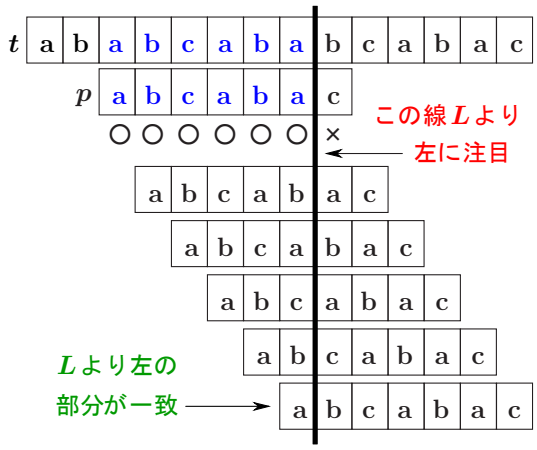
素朴なアルゴリズムにおいて，$p[i] \neq t[j]$ となったとき，p を（一文字分だけでなく）もっとずらすことができれば，文字の比較を減らすことができる.

⇓

効率が改善するはず.



移動位置をどのように決めているか.



－－－－－－－－－－－－－－－－－－－－

クヌース・モーリス・プラットのアルゴリズムでは，$p_i \neq t_j$ となったとき，

> 線 L より左の部分が一致しているような最初の（最も左の）位置に p を移動させる．

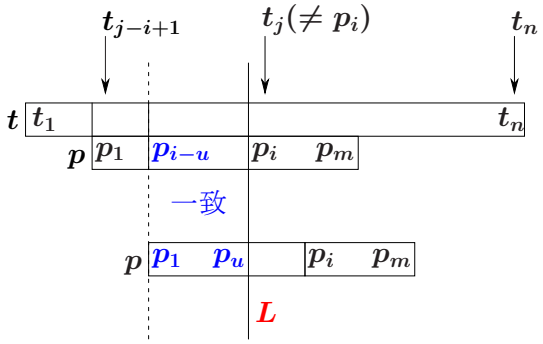
線 L より左の部分で文字の不一致があるような位置に p を移動させても照合しない．

⇓

上記のようにしても，照合を見逃すことはない．

－－－－－－－－－－－－－－－－－－－－

失敗関数（failure function）：p の移動位置を高速に求めるための関数．前処理で値を計算する．



$$p_1\ p_2 \cdots p_u = p_{i-u}\ p_{i-u+1} \cdots p_{i-1}$$

－－－－－－－－－－－－－－－－－－－－

失敗関数

$$f(i) = 1 + \max\{u \mid 0 \leq u < i-1,\ p_1\ p_2 \cdots p_u = p_{i-u}\ p_{i-u+1} \cdots p_{i-1}\}$$

ただし $f(1) = 0$ とする．明らかに $f(i) \leq i-1$.

p を t に重ねて文字の一致を調べていったとき，p_i と t_j で不一致が起こったとする．

⇓

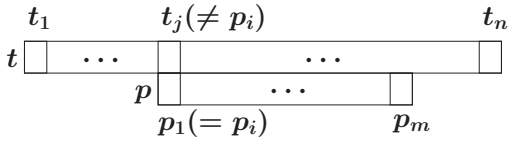
$i \leftarrow f(i)$ とし，文字の一致の検査を p_i と t_j から続ける（j の値の更新は必要ない）．

－－－－－－－－－－－－－－－－－－－－

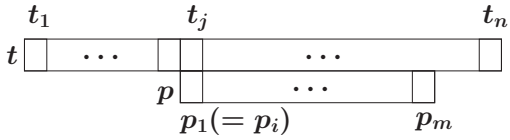
```
関数 kmp()
int i = 1, j = 1;
compf;                    ・・・ 失敗関数の計算
while (i <= m && j <= n)
    if (i == 0 || p[i] == t[j]){
        i++; j++;
    }
    else i = f[i];
if (i == m + 1)
    printf("Found at %d\n", j - m);
else printf("Not found\n");
```

－－－－－－－－－－－－－－－－－－－－

kmp において $i = 0$ となる状況



一旦 $i \leftarrow f(1)(= 0)$ とされる．
次の if 文の実行時に，
$i \leftarrow i+1(= 1)$, $j \leftarrow j+1$ とされる．

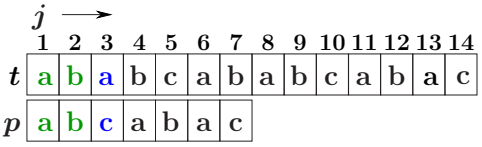


－－－－－－－－－－－－－－－－－－－－

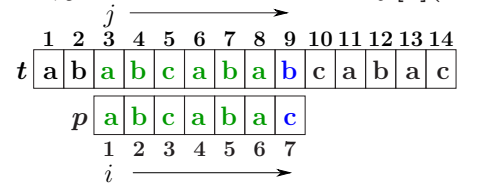
［例］ t =ababcababcabac, p =abcabac
失敗関数の計算（compfの実行）結果

i	1	2	3	4	5	6	7
$f(i)$	0	1	1	1	2	3	2

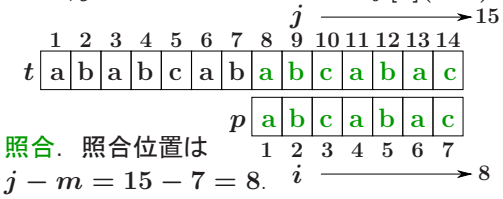
最初は $i = 1, j = 1$.



$i = 3, j = 3$ で不一致 ⇒ $i \leftarrow f[3](= 1)$.



$i = 7, j = 9$ で不一致 ⇒ $i \leftarrow f[7](= 2)$



照合．照合位置は $j - m = 15 - 7 = 8$.

－－－－－－－－－－－－－－－－－－－－

時間計算量（compf の実行時間を除く）

```
while (i <= m && j <= n)
    if (i == 0 || p[i] == t[j]){
        i++; j++;
    }
    else i = f[i];
```

- while ループの実行時間は，i の値が動く回数に比例.
- j の値は減らない．⇒ 増加する回数は $O(n)$.
- $i \leftarrow f(i)$ とすると，i の値は減る．⇒ i の値が増加する回数は，j と同じであるから $O(n)$.
- i の値は 1 回に 1 ずつしか増えないので，i が減少する回数は増加する回数をこえない.
  ⇒ i の値が減る回数も $O(n)$.

⇓

compf の実行時間を除けば，クヌース・モーリス・プラットのアルゴリズムの時間計算量は $O(n)$ である.

－－－－－－－－－－－－－－－－－－－－

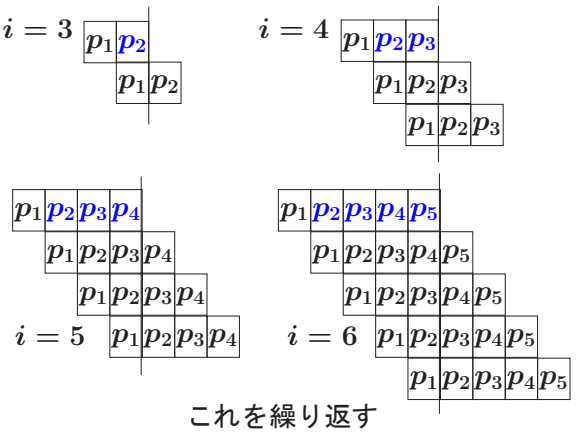
失敗関数の定義（再掲）

$$f(i) = 1 + \max\{u \mid 0 \le u < i-1,\ p_1\ p_2 \cdots p_u = p_{i-u}\ p_{i-u+1} \cdots p_{i-1}\}$$

ただし $f(1) = 0$. また，$f(2) = 1$.

$f(i)$ $(3 \le i \le m)$ のすぐ思いつく計算法

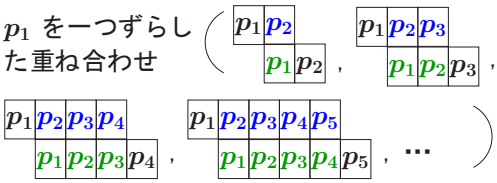
$p_1\ p_2 \cdots p_{i-1}$ と $p_1\ p_2 \cdots p_{i-1}$ をずらしながら重ね合わせていく.



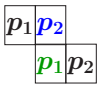
これを繰り返す

－－－－－－－－－－－－－－－－－－－－

（実行時間を考えたときの）この方法の問題点



を独立に行うのは無駄.

例えば，  において $p_1 \ne p_2$ であることが分かれば，他の重ね合わせは不要.

例えば，  において $p_1 = p_2$ であることが分かれば，他の重ね合わせで p_1 と p_2 を比較する必要がない.

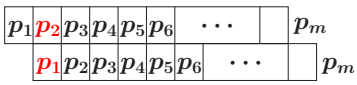
同様の無駄が，p_1 を二つずらした重ね合わせ，三つずらした重ね合わせ，$\cdots$ でも起こる.

－－－－－－－－－－－－－－－－－－－－

クヌース・モーリス・プラットの方法における失敗関数の計算（compf）

$f(1) = 0$，$f(2) = 1$ となるようにする.

p_1 を一つずらした重ね合わせは，p と p に対して 1 回だけ行う.



- $p_1 \ne p_2$ であれば，この重ね合わせについては，それ以降の要素の比較をしない.
- $p_1 = p_2$ であれば，$f(3) \leftarrow 2$ として，p_2 と p_3 の比較を行う.
  - $p_2 \ne p_3$ であれば，この重ね合わせについては，もう要素の比較をしない.
  - $p_2 = p_3$ であれば，$f(4) \leftarrow 3$ として，p_3 と p_4 の比較を行う.

－－－－－－－－－－－－－－－－－－－－

比較した要素の不一致が見つかるまで，同様の処理を続ける．不一致が見つかれば，p_1 をより大きくずらした重ね合わせを考える．

compf では，$f(i)$ の計算を，$i = 1, 2, 3, \cdots$ の順に行う．

$f(1), \cdots, f(j)$ まで既に求まっているとして，次に $f(j+1)$ を計算するものとする．

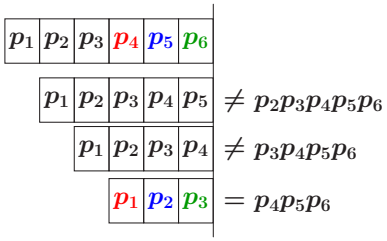
> $f(1), \cdots, f(j)$ の値を利用して，無駄な重ね合わせや要素の無駄な比較を避ける．

－－－－－－－－－－－－－－－－－－－－

例えば，$f(7)$ まで求めたところだとする．
$f(7) = 4$, $f(4) = 2$ であるとする．

$$f(7) = 1 + \max\{u \mid 0 \le u < 6, \\ p_1\ p_2 \cdots p_u = p_{7-u}\ p_{8-u} \cdots p_6\}.$$

この値が 4 であるから，

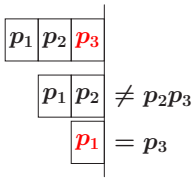


－－－－－－－－－－－－－－－－－－－－

同様に，

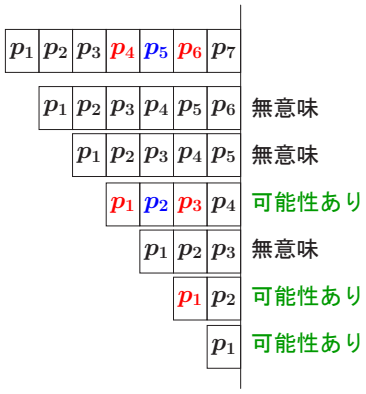
$$f(4) = 1 + \max\{u \mid 0 \le u < 3, \\ p_1 \cdots p_u = p_{4-u} \cdots p_3\}.$$

この値が 2 であるから，

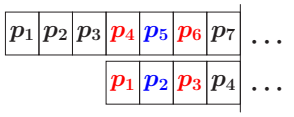


－－－－－－－－－－－－－－－－－－－－

$f(8)$ を計算するときの重ね合わせとして，

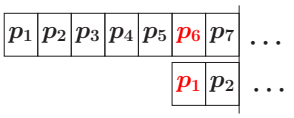


－－－－－－－－－－－－－－－－－－－－

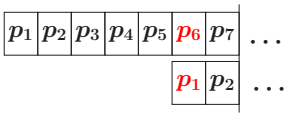


p_7 と p_4 を比較する（$j = 7$, $i = 4 = f(7)$）

- $p_7 = p_4$ であれば $f(8) \leftarrow 5$.
  （$j \leftarrow j + 1$; $i \leftarrow i + 1$; $f(j) \leftarrow i$;）
- $p_7 \neq p_4$ であれば，以下のようにずらして，p_7 と p_2 を比較する．（$i \leftarrow 2 = f(4) = f(i)$）

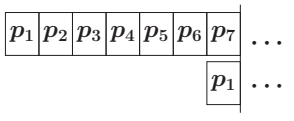


－－－－－－－－－－－－－－－－－－－－



p_7 と p_2 を比較する（$j = 7$, $i = 2$）

- $p_7 = p_2$ であれば $f(8) \leftarrow 3$.
  （$j \leftarrow j + 1$; $i \leftarrow i + 1$; $f(j) \leftarrow i$;）
- $p_7 \neq p_2$ であれば，以下のようにずらして，p_7 と p_1 を比較する．（$i \leftarrow 1 = f(2) = f(i)$）



－－－－－－－－－－－－－－－－－－－－

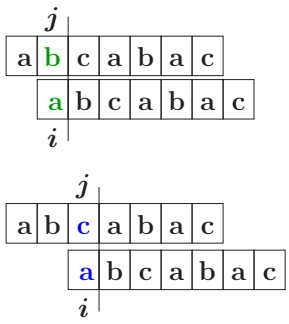
関数 compf()

```
int i = 0, j = 1;
f[1] = 0;
while (j < m)
  if (i == 0 || p[i] == p[j])
    f[++j] = ++i;
  else i = f[i];
```

－－－－－－－－－－－－－－－－－－－－

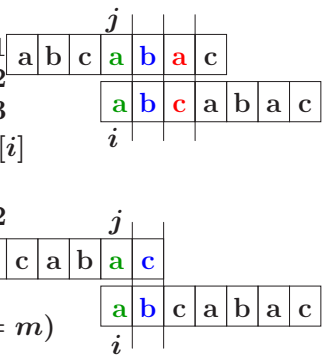
compf の動作例（p = abcabac）

j	i	
1	0	$f[1] \leftarrow 0$
2	1	$f[2] \leftarrow 1$
		$p[j] \neq p[i]$
2	0	
3	1	$f[3] \leftarrow 1$
		$p[j] \neq p[i]$



－－－－－－－－－－－－－－－－－－－－

j	i	
3	0	
4	1	$f[4] \leftarrow 1$
5	2	$f[5] \leftarrow 2$
6	3	$f[6] \leftarrow 3$
		$p[j] \neq p[i]$
6	1	
7	2	$f[7] \leftarrow 2$
		停止 $(j = m)$



－－－－－－－－－－－－－－－－－－－－

compf の時間計算量は $O(m)$.

kmp の時間計算量（compf の部分を除く）の評価におけるのと同様の理由による.

- while ループの実行時間は，i の値が動く回数に比例.
- i の値が増加する回数は，j と同じで $O(m)$.
- i が減少する回数は増加する回数をこえない.
  $\Rightarrow i$ の値が減る回数も $O(m)$.

kmp 全体の時間計算量は $O(n+m)$.
（$m \leq n$ と仮定しているので，$O(n)$ としてよい.）

－－－－－－－－－－－－－－－－－－－－