アルゴリズム及び演習 第 11 回演習解答

小野 孝男*

2007 年 7 月 2 日

1. パターンは $\pi = abcabcab$ なので, それぞれ以下のようになる:
   (a) 例えば $i = 5$ に対しては, その直前の a と先頭の a が等しいので「パターンの 5 文字目で失敗した場合, 次はその文字をパターンの 2 文字目と比較してみればよい」ことがわかる. つまり $f(5) = 2$ である. 全ての i に対して $f(i)$ を計算すると表 1 の通りになる.

i	1	2	3	4	5	6	7	8
$f(i)$	0	1	1	1	2	3	4	5

表 1　パターン π に対する失敗関数 $f(i)$

   (b) オートマトンは図 1 のようになる. 但し, 状態 3, 5, 6, 8 で入力が a のときは状態 2 へ, その他図にない全ての入力に対しては状態 1 へ戻るものとする.

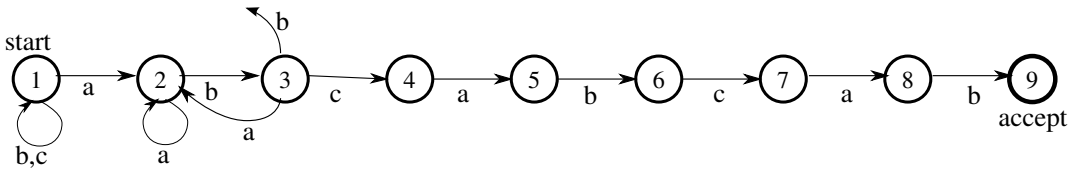

図 1　π にマッチするオートマトン

2. 文字列 β から, 無限長の文字列 $\beta' = b'_0 b'_1 \dots b'_k \dots$ を

$$b'_i = b_{i \bmod n}$$

   によって作る. この β' については次の性質が成り立つ:

   命題 1. β' の中に α が存在する iff 全ての $i \in \{0, 1, \dots, n-1\}$ に対して $a_i = b_{(k+i) \bmod n}$ を満たす k が存在する.

   証明. ($\Rightarrow$) β' の中に α が存在するので, 全ての i に対し $a_i = b'_{k+i}$ であるような k が存在する. β' の作り方から, これは全ての i に対し $a_i = b_{(k+i) \bmod n}$ であることを意味する.
   ($\Leftarrow$) 全ての i に対し $a_i = b_{(k+i) \bmod n}$ と仮定する. この k に対し, $b'_{k+i} = b_{(k+i) \bmod n} = a_i$ である. つまり, $b'_{k+1} \dots b'_{k+n}$ は α と全く同じであるので, β' の中に α が存在する.

   □

* ono@is.nagoya-u.ac.jp

さらに, $a_i = b_{(k+i) \bmod n}$ を満たす k は, 0 以上 n 未満と仮定してよい. つまり, β' において参照される文字を $b'_0 b'_1 \ldots b'_{2n-1}$ までに制限してもかまわないことがわかる.

この結果から, β を 2 つ連結した文字列をテキスト, α をパターンとして KMP (あるいは BM) の文字列照合アルゴリズムを実行すればよいことがわかる. これらのアルゴリズムはテキストとパターンの長さの和の線形時間で実行できるので, 全ての i に対して $a_i = b_{(k+i) \bmod n}$ であるような k が存在するかどうかも $2n + n = 3n$ の線形時間, つまり $O(n)$ 時間で判定することができる.

3. 最長共通部分列の長さを求める際に, 同時に「その長さがどのように計算されてきたのか」という履歴を記録すればよい. 実際には「直前はどこか」を記録しておくだけで十分である.

   具体的には, 「直前はどこか」を記録する配列 `prev[][]` を用いてテキスト p. 117 の 2 重ループ部分の内部を次のように変更する:

```
if (a[i] == b[j]) {
    lcs[i][j] = lcs[i-1][j-1] + 1;
    prev[i][j] = LEFTUP;
} else if (lcs[i][j-1] < lcs[i-1][j]) {
    lcs[i][j] = lcs[i-1][j];
    prev[i][j] = LEFT;
} else {
    lcs[i][j] = lcs[i][j-1];
    prev[i][j] = UP;
}
```

   そして, この配列 `prev[][]` に基づいて (m, n) から $(0, 0)$ へとたどってゆけばよい:

```
i = m; j = n;
while (i > 0 && j > 0) {
    switch (prev[i][j]) {
    case UP:
        --j;
        break;

    case LEFT:
        --i;
        break;

    default:
        printf("%c", a[i]);
        --i; --j;
        break;
    }
}
```

   これは逆順に表示するので, 必要であれば再帰呼び出しなどによって正順に表示するよう変更すればよい.