

付録 A

文字列の比較と照合

2つの文字列を比較したり照合したりするアルゴリズムについて学ぶ。

A.1 文字列の同一判定

与えられた2つの文字列が等しいかどうかを調べる。

前から順に比較していき、一致しない文字があれば、その時点で文字列が異なることがわかる。最後まですべて一致すれば、文字列は完全に等しい。

```
アルゴリズム 1: string_eqaul( $s, t$ )
begin
   $i \leftarrow 1$ ;
  while ( $s[i]$  と  $t[i]$  が存在する) do
    begin
      if ( $s[i] \neq t[i]$ ) return 0;
       $i \leftarrow i + 1$ ;
    end;
  return 1;
end;
```

A.2 文字列の比較

与えられた2つの文字列 s と t のうち、どちらが辞書順 (コード順) で前であるかを調べる。

具体例 文字列をソートするために不可欠。

次の3つの場合に分けられる。

- s の方が前である場合は、正の数を返す
- s と t が同一である場合は、0を返す。
- s の方が後である場合は、負の数を返す。

アルゴリズム1の一部を変更し、一致しないことがわかったとき、どちらが前であるかを調べるようにする。

アルゴリズム2: `string_compare(s, t)`

```
begin
   $i \leftarrow 1$ ;
  while (1) do
    begin
      if( $s[i]$ と $t[i]$ が存在する) do
        if ( $s[i] \neq t[i]$ ) return  $s[i] - t[i]$ ;
        else if ( $t[i]$ が存在する) return 1;
        else if ( $s[i]$ が存在する) return -1;
        else return 0;
       $i \leftarrow i + 1$ ;
    end;
  end;
```

A.3 照合

ある文字列 p が、別の文字列 t に、部分文字列として含まれているかどうかを調べる。ここでは、文字列 p をパターン、文字列 t をテキストと呼ぶ。

具体例 テキストエディタやブラウザの「このページの検索」機能（「編集メニュー」の中にあることが多い）。そのファイル（実体は文字列）の中から、指定された文字列が存在する場所を探し、表示する。

パターン p は比較的短い文字列であるのに対し、テキスト t は（非常に）長い文字列であることが多い。

素朴なアルゴリズム

文字列の同一性を判定するアルゴリズム 1 を、テキスト t の先頭から順に適用する。一致しないことがわかったならば、 t を一文字分だけ左にずらして (つまり、パターン p を右に 1 文字分だけ移動させて)、再度適用する。

アルゴリズム 3: `string_match(p, t)`

```
begin
   $m \leftarrow$  “ $p$  の長さ”;
   $k \leftarrow 1$ ;
  while ( $t[k]$  が存在する) do
    begin
       $succ \leftarrow 1$ ;
      for  $i \leftarrow 1$  until  $m$  do
        if ( $t[k + i]$  が存在しない) return 0;
        else if ( $p[i] \neq t[k + i]$ )  $succ \leftarrow 0$ ; last;
      if ( $succ = 1$ ) return  $k$ ;
       $k \leftarrow k + 1$ ;
    end;
  end;
```

比較回数は、テキスト t の長さを n とすると、最悪の場合、 $m \times n$ (つまり、 $O(mn)$) だ。しかし、比較はほとんどの場合失敗するので、実際的には、 n 回程度。

(簡易版) ボイヤー・ムーア法

より巧妙な方法。

基本的なアイデアは、「パターン p の後方からテキストと比較する」こと。

パターンのある文字 $p[i]$ とテキストのある文字 $t[j]$ を比較し、それが一致しなかった状況を考える。

1. もし、 $t[j]$ と同じ文字がパターン p に含まれていない場合は、パターン p をその長さ分だけ右に移動 (シフト) させてよい。

2. 一方、 $t[j]$ と同じ文字がパターン p に含まれていた場合は、その文字がちょうど $t[j]$ と重なるようにパターンを移動 (シフト) させる必要がある。

いずれにしても、1文字以上のシフトが可能である。ほとんどの場合が前者であれば、比較回数は、 n/m 程度となる (オーダーは変わらない)。

本来のボイヤー・ムーア (Boyer-Moore) ・アルゴリズムは、さらなる工夫があるが、ここでは、上記の考えに基づく簡易版のアルゴリズムを示す。

アルゴリズム 4: `simplified_BM(p, t)`

```

begin
   $m \leftarrow$  “ $p$  の長さ”;
  すべての文字  $c$  に対して、
     $shift[c] = m$ ;
  for  $i \leftarrow 1$  until  $m - 1$  do
     $shift[p[i]] = m - i$ ;
   $k \leftarrow m$ ;
  while ( $t[k]$  が存在する) do
    begin
       $succ \leftarrow 1$ ;
      for  $i \leftarrow 0$  until  $m - 1$  do
        if ( $t[k - i]$  が存在しない) return 0;
        else if ( $p[m - i] \neq t[k - i]$ )  $succ \leftarrow 0$ ; last;
      if ( $succ = 1$ ) return  $k - m + 1$ ;
       $k \leftarrow k + shift[t[k - i]]$ ;
    end;
  end;
end;
```

A.4 文字列の近似照合

2つの文字列がどのくらいよく似ているかを計算する。

具体例 英語のスペルチェック (エラー訂正)。辞書に存在しないスペルが見つかったとき、正しいスペル (単語) を推定する。誤りが些細であれば、誤っ

たスペルと正しいスペルは、よく似ている（はずである）。

これを実現するために、2つの文字列間に距離（または、類似度）を定義する。

ここで、1文字削除、1文字挿入、文字の置換、という3つの操作を考える。2つの文字列が与えられた時、一方の文字列にこれらの操作を何回か適用することにより、他方の文字列を作り出すことができる。その回数の下限を2つの文字列の編集距離と定義する。

ここでは、これを一般化して、文字 c 、および、2つの文字の組 $\langle c_1, c_2 \rangle$ に対して、以下のような非負の値が定義されている場合を考える。（上記の編集距離は、これらがすべて1の場合）

$del(c)$ 文字 c を削除する

$ins(c)$ 文字 c を挿入する

$rep(c_1, c_2)$ 文字 c_1 を c_2 で置き換える ($c_1 \neq c_2$)

2つの文字列 s と t の（一般化された）編集距離は、動的計画法によって計算できる。漸化式は、以下のとおり。

$$D(i, j) = \min \left\{ \begin{array}{l} D(i-1, j) + del(s[i]), \\ D(i-1, j-1) + d(s[i], t[j]), \\ D(i, j-1) + ins(t[j]) \end{array} \right\} \quad (i, j \geq 2) \quad (A.1)$$

$$d(i, j) = \begin{cases} 0 & \text{if } s[i] = t[j] \\ rep(s[i], t[k]) & \text{otherwise} \end{cases} \quad (A.2)$$

i または j が 1 の場合に注意すると、以下のアルゴリズムが得られる。

アルゴリズム 5: distance(s, t)

begin

$m \leftarrow$ “ s の長さ”;

$n \leftarrow$ “ t の長さ”;

$dist[1, 1] \leftarrow 0$;

for $j \leftarrow 2$ **until** n **do**

$dist[1, j] \leftarrow dist[1, j-1] + ins(t[j])$;

for $i \leftarrow 2$ **until** m **do**

$dist[i, 1] \leftarrow dist[i-1, 1] + del(s[i])$;

for $j \leftarrow 1$ **until** n **do**

```
dist[i, j] ← min(dist[i - 1, j] + del(s[i]),
                  dist[i - 1, j - 1] + d(s[i], t[j]),
                  dist[i, j - 1] + ins(t[j]));
return dist[m, n];
end;
```

念のために 文字列の実装法は、プログラミング言語によって異なる。例えば、プログラミング言語 C では、文字列は、文字の配列（但し、最後に NULL を付加したもの）として実装されている。アルゴリズムのプログラム化においては、それぞれの実装法に対応した形に変更する必要がある。（より巧妙に実現できる場合もある。）

参考文献

1. 長尾, 黒橋, 佐藤, 池原, 中野. 言語情報処理. 岩波講座「言語の科学」9, 岩波書店, 1998. (1.6 節「文字列の照合」)
2. 茨木俊秀. C によるアルゴリズムとデータ構造. 昭晃堂, 1999. (6.3 節「文字列の照合」)

(2003 年 5 月 28 日 - 佐藤理史)