インターフェイスの街角 (50)—展開テキストの検索

増井俊之

検索の難しさ

計算機のヘルプ機能が便利だと感じている人は、あまり多くないのではないでしょうか。たとえば、Windows のスタートメニューには〝ヘルプ〟がありますが、すくなくとも私は、問題が起きたり使い方が分からなかったりしたときに役に立ったと実感したことはほとんどありません。

ヘルプやマニュアルに記述がなければあきらめもつきますが、該当する項目があるにもかかわらず、検索キーワードが一致しないために肝腎の説明にいきつけないこともかなり多いように思います。

たとえば、Windows のヘルプでは〝時間をセットする〟方法はみつからず、「時刻を設定する」で検索する必要があります。同様に、〝計算機をリブートする〟方法はヘルプにはなく、「コンピュータを再起動する」で検索する必要があります。このように、用語がほんのすこし違うだけでも検索できなくなる点が使いにくいと感じる大きな原因の 1 つでしょう。

これはヘルプやマニュアルに限った話ではなく、情報検索における長年の懸案ともいうべき問題であり、その解決のために数多くの試みがおこなわれてきました。

これを根本的に解決するには、文字列としてではなく内容にもとづいて検索する手法が必要です。しかし、上例のように単純なケースでも、〝計算機の内蔵時計の時刻を設定する〟という概念を計算機上で表現したり、その概念を検索する操作をおこなうのはそれほど容易なことではありません。そのため、現実には文字列としての検索になんらかの工夫を加える方法がよく利用されています。

そのなかでもっとも簡単なのは〝シソーラス〟を使う方法です。シソーラスを利用すれば、〝リブート＝再起動〟や〝時間　時刻〟などの、意味の似た別のキーワードによる検索も可能になるので、多少は状況が改善されるかもしれません。たとえば、〝リブート〟というキーワードに対して〝再起動〟も該当するようにしておけば、ヘルプに「再起動」しかなくても求める項目を探し出せます。

このような手法を応用すれば、〝リブート〟というキーワードから「コンピュータを再起動する」という項目を検索することはできます。しかし、ユーザーが〝リブート〟という言葉にとらわれていると、〝リブート＝再起動〟であることに気づかず、そのヘルプには〝リブート〟に関する記述はないと判断してしまう可能性もあります。〝計算機〟と〝リブート〟というキーワードで検索した場合は、「計算機をリブートするには」という項目がみつかるほうがより自然でしょう。

展開テキストを用いた検索

上で述べたのは、指定されたキーワードの処理に工夫を加える方法です。これに対し、検索対象のほうをあらゆるキーワードに対応できるようにしておけば、どんなキーワードが指定されても、それに対応する項目がみつかるはずです。たとえば、時刻の設定に関する記述でいえば、〝時間を設定する〟や〝時計を合わせる〟、あるいは〝時刻をセットする〟など、考えうるあらゆる組合せの表現を用意しておくわけです。

〝あらゆる組合せ〟などというと、ヘルプ項目が膨大なサイズになってしまうような気がします。しかし、現実にはヘルプの項目はせいぜい数百個程度であり、それぞれの長さはおおむね 20 文字どまりなので、項目だけなら数十 KB といったところでしょう。項目ごとに 3 つのキーワー

ドが含まれており、各キーワードに 3 種類の言換え表現があると仮定すると、それぞれ $3*3*3 = 27$ 通りの表現がある計算になります。しかし、それでも 1MB 程度の大きさにしかならないので、最近の計算機なら楽にメモリ上で扱うことができます。

完全に展開したテキストを検索対象とする場合には、Migemo[1]を用いたローマ字によるインクリメンタル検索も可能です。さきほどの〝リブート＝再起動〟のようなシソーラスを利用する場合には、〝リブート〟というキーワードを確定させたあとで、はじめて〝再起動〟のようなパターンを対象とした検索がおこなえるようになります。一方、「計算機をリブートする」「コンピュータを再起動する」といった表現に展開したテキストを検索するときは、Migemo を使えば、〝rib〟と入力した時点で「計算機をリブートする」に絞り込めます。

このように、検索対象がそれほど広大ではなく、かつ検索キーワードの表記に揺れがある場合には、あらかじめ検索対象をさまざまなかたちに言い換えておいてから検索する方法が有効ではないでしょうか。

言換え辞書

検索対象をさまざまな表現に展開するには、まず言換えのための辞書を用意します。

本来ならば、ヘルプ項目を文法的に解析して言換え表現を生成する方法をとるべきかもしれません。しかし、これを完璧におこなうのはかなり難しいので、単純な単語の置換によって展開することにします。

言換え辞書としては図 1 のようなものを使用します。これは、左側の単語を右側の単語列で言換え可能であることを示しています。〝ロゴ〟は〝絵〟に置き換えてもかまいませんが、〝絵〟を〝ロゴ〟に置き換えるとおかしなことになってしまうので、言換えには方向性をもたせています。また、〝位置〟と〝場所〟はたいていの場合は言換えが可能ですが、〝置き場所〟を〝置き位置〟に言い換えるのはおかしいので、「置き場所」は別に定義しています。

展開スクリプト

テキストの表現を言い換えるには、図 2 のプログラム (dicexpand) を使います。検索対象のテキストのなかに

1 http://migemo.namazu.org/

図 1　言換え辞書

```
#
# 言換え不能単語
#
置き場所
#
# 可換単語
#
位置                場所
場所                位置
サイズ              大きさ
大きさ              サイズ
追加する            加える
加える              追加する
変える              変更する
変更する            変える
作る                作成する
作成する            作る
#
# 計算機関連単語
#
計算機              コンピュータ, マシン
コンピュータ        計算機, マシン
リスタート          再起動, リブート
再起動              リスタート, リブート
リブート            再起動, リスタート
エクステンション    拡張子, ファイルタイプ
拡張子              エクステンション, ファイルタイプ
ファイルタイプ      拡張子, エクステンション
IME                 日本語入力システム
日本語入力システム  IME
消す                削除する, 消去する
削除する            消す, 消去する
消去する            消す, 削除する

トラブル            問題
ロゴ                絵
アイコン            絵
ビットマップ        絵
バッテリー          電池
```

図 1 の辞書エントリにマッチする単語が現れた場合は、それを複数の表現で言い換える処理を再帰的に実行し、できるかぎり幅広い表現を生成するようになっています。

展開の実行

dicexpand の引数に図 1 の辞書 (dict) とヘルプテキスト tips（図 3）を指定し、

```
% ./dicexpand dict help.txt
```

のように実行すると、help.txt の表現を言い換えたテキスト（図 4）が出力されます。

このようにして展開したテキストに対し、〝拡張子〟および〝変える〟といったキーワードで検索すると、「拡張子の関連付けを変える」という項目がみつかります。同様

図 2　テキスト展開スクリプト dicexpand

```
#!/usr/local/bin/perl

$dictfile = shift;
if($dictfile eq ''){
    die "% dicexpand dict [text]\n";
}

open(dict,$dictfile) || die "Can't open dictfile\n";
while(<dict>){
    next if /^#/ || /^\s/;
    chop;
    ($word,$subst) = split;
    if($subst eq ''){
        $subst = $word;
    }
    $dict{$word} = $subst;
    push(@dict,$word);
}
close(dict);

while(<>){
    next if /^#/ || /^\s/;
    chop;
    ($number,$data) = split;
    &expand($number,"",$data);
}

sub expand {
    local($number,$prefix,$data) = @_;
    local(@s,$w,$s,$r);
    if($data eq ''){
        print "$number\t$prefix\n" unless $done{$prefix};
        $done{$prefix} = 1;
    }
    else {
        for $w (@dict){
            if($data =~ /^$w(.*)$/){
                $r = $1;
                &expand($number,$prefix.$w,$r);
                @s = split(/,/,$dict{$w});
                for $s (@s){
                    &expand($number,$prefix.$s,$r);
                }
                return;
            }
        }
        ($s,$r) = ($data =~ /^([\x80-\xff].|[\x00-\x7f])(.*)/);
        &expand($number,$prefix.$s,$r);
    }
}
```

に、〝タイプ〟と〝変更〟というキーワードで検索した場合は、「ファイルタイプの関連付けを変更する」という項目がみつかることになります。

Generate and Filter

以上の例では検索する前にテキストの全文を展開していますが、展開しながら検索すれば用途はさらにひろがるでしょう。

たとえば、〝次の 100 行を 4 文字インデントする〟とか、〝ここ 1 週間で更新したファイルをすべて CD-R にバックアップする〟といった操作の手順を調べようとしても、普通のヘルプシステムでは〝行を x 文字ぶんインデントする方法〟〝同じ操作を y 回実行する方法〟〝z 時間以内に更新されたファイルのリストを得る方法〟〝指定されたファイルを CD-R に焼く方法〟などの項目しかみつか

図 3　Windows のヘルプエントリ例

```
1       拡張子の関連付けを変更する
2       IMEのツールバーを消す
3       計算機を再起動する
4       ごみ箱のアイコンを変更する
5       ファイルの置き場所を変える
```

図 4　展開結果

```
1       拡張子の関連付けを変更する
1       拡張子の関連付けを変える
1       エクステンションの関連付けを変更する
1       エクステンションの関連付けを変える
1       ファイルタイプの関連付けを変更する
1       ファイルタイプの関連付けを変える
2       IMEのツールバーを消す
2       IMEのツールバーを削除する
2       IMEのツールバーを消去する
2       日本語入力システムのツールバーを消す
2       日本語入力システムのツールバーを削除する
2       日本語入力システムのツールバーを消去する
3       計算機を再起動する
3       計算機をリスタートする
3       計算機をリブートする
3       コンピュータを再起動する
3       コンピュータをリスタートする
3       コンピュータをリブートする
3       マシンを再起動する
3       マシンをリスタートする
3       マシンをリブートする
4       ごみ箱のアイコンを変更する
4       ごみ箱のアイコンを変える
4       ごみ箱の絵を変更する
4       ごみ箱の絵を変える
5       ファイルの置き場所を変える
5       ファイルの置き場所を変更する
```

りません。このような操作を実行する場合は、機能をうまく組み合わせたり、パラメータの値を与えたりする必要があります。

1 文字インデントする方法、2 文字インデントする方法、…… というふうに、すべてをヘルプ項目にするわけにはいかないので、これらを一般化して〝n 文字インデントする方法〟が挙げられているわけです。結果として、ユーザーがおこないたいのは〝4 文字インデントする〟といった具体的な作業であるにもかかわらず、一種の抽象化をしなければその方法が調べられないことになります。あることをしたいときに知りたいのはその具体的な操作手順であり、その種の作業の一般的な知識ではありません。

このような場合、〝100〟や〝4〟〝インデント〟などの具体的なキーワードから予想される機能をすべて展開し、それらを対象として絞込み検索を実行すれば、目的とする項目に直接たどりつけるでしょう。たとえば、上例のキーワードであれば、〝4 パラグラフを 100 文字インデントする〟とか〝4 行インデントを 100 回繰り返す〟、あるいは〝4 文字インデントを 100 回繰り返す〟といった項目への展開が可能です。これらのなかから該当する項目を選択するか、さらに異なるキーワードを追加していけば、目的とする操作手順を知ることができます。

図 5　生成後検査法のアルゴリズム

```
for(;;){
    解候補Sを生成;
    解候補が存在しなければ終了;
    Sが正しい解であるかを検査;
    正しい解であればSを印刷;
}
```

あらゆる解の候補を順に生成してから、それが解になっているかを調べていく方法は**生成後検査 (Generate and Test) 法**と呼ばれ（図 5）、8-Queen 問題[2]などのパズルを解くためのプリミティブなアルゴリズムとしてよく知られています。ヘルプの検索やコマンド起動の場合も、候補の生成と手作業によるフィルタリングを組み合わせることにより、目的とする解答が素早く得られるのではないでしょうか。このような、いわば〝Generate and Filter〟方式にもとづいてヘルプ項目の検索やコマンドの実行をうまく制御できれば、頭に浮かんだ要求をそのまま入力することで計算機を操作する理想的なインターフェイスに一歩近づけるような気がします。

おわりに

計算機の高速化やメモリの低価格化により、今回のシステムのように検索対象のテキストを展開してから検索を実行したり、あるいは Migemo のようにキーワードの展開後に検索する方法も十分実用的なレベルに達しつつあります。今後も、計算機のもてるパワーをふんだんに使ってユーザーの環境を快適にする**富豪的アプローチ**[3]をさらに追求していきたいと思います。

（ますい・としゆき　ソニー CSL）

2 チェス盤上に 8 個のクイーンを置き、各クイーンがたがいに経路を妨害しないように動ける配置を求めるパズルです。

3 http://www.csl.sony.co.jp/person/masui/fugo.html