デジタルサーチャー講座

検索を科学する

塩田 紳二

第7回　検索の技法

今回は、検索しているときにコンピュータの中でどのような処理が行われているかについて解説する。この連載でも概念的な部分については解説したが、今回は実際にプログラムがどのような動きをするのかについて解説してみることにする。

解説の前に

解説する前に、少し用語などについてはっきりさせておこう。コンピュータ関連の用語には、きちんと定義されているものもあれば、曖昧なものもある。しかし、コンピュータの動作のような論理的なものを解説する場合には、個々の用語の意味は、その文章の中でははっきりしている。ただ、人によって用語の定義などに多少の違いがあり、そのために、複数の解説を見ると、用語定義が曖昧なように感じることがある。

まずは「文字列」。この連載でも時折出る単語だが、これは、文字の並びを意味する。文章などと同じように文字が並んでいるものだが、文字列といったときには、文法などには影響されない。たとえば、未完成の文章であったり、文法的には間違っていたり、意味のない羅列である可能性もある。

次に「文字」だが、これは、コンピュータ内で一定数のビットから構成されるパターンで、文字コードによって現実の文字との対応が行われる。たとえばASCIIと呼ばれる文字コードでは、大文字のAを1000001という7bitで表現する。これを10進数で表すと65になる(図1)。

コンピュータは、メモリーを8bit単位、16bit単位といったまとまりで読み書きするため、実際には、7bitで定義されたASCIIコードは、8bitのデータとして「01000001」とメモリーに書き込まれる。

そうなると、文字列とはメモリー内の連続した領域に書き込まれた文字コードのつながりということになる。

文字の種類(文字が何種類あるのか)を「アルファベット」と呼ぶ。一般にアルファベットとは、英語のabcを意味するが、コンピュータ関連で文字を扱うときにアルファベットというと、文字の種類(文字集合)を指すことが多い。これもコンピュータのメモリー上で考えれば、ビットパターンの種類ということになる。

コンピュータの文字列の場合、必ずしも現実の文字と対応していないこともある。たとえば、DNAから特定のパターンを見つけるなんて場合にも利用される。

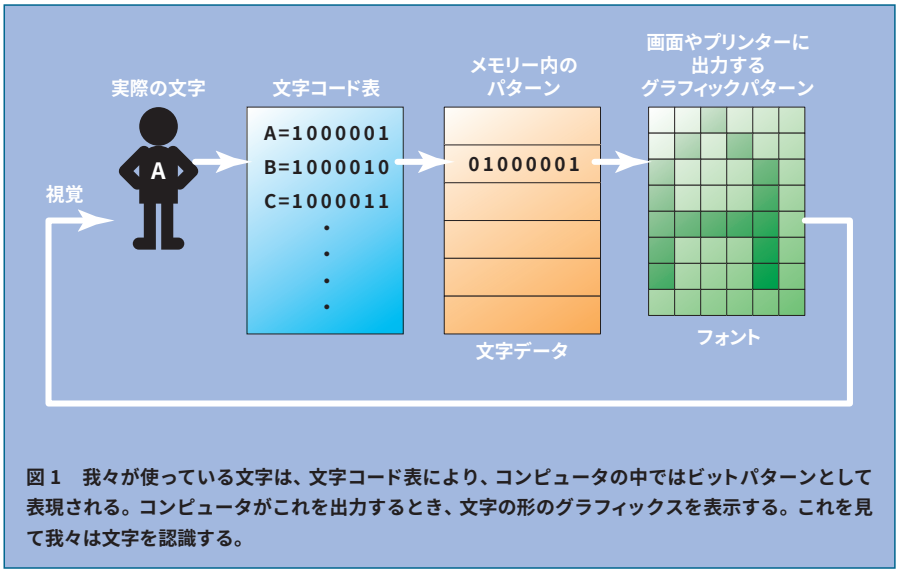

図1　我々が使っている文字は、文字コード表により、コンピュータの中ではビットパターンとして表現される。コンピュータがこれを出力するとき、文字の形のグラフィックスを表示する。これを見て我々は文字を認識する。

このようなとき、アルファベットはAGCTの4種類しかない。しかし、対象となるDNAは、長さが30億程度（人間の場合）あり、たった4種類の文字しかないとはいえ、コンピュータを使わないと検索は不可能である。

あるいは効率のため、アルファベットを制限することもある。たとえば、検索を簡単にするために小文字だけを使うといったことである。これは、検索漏れをなくすためにも有効な方法だ。大文字、小文字の違いが意味にかかわってこなければ、このような制限を加えることで検索漏れを防ぐこともできるわけだ。

次に「終端記号」だが、これは、1つのデータの終わりを意味するもので、具体的にはメモリー上のビットパターンに対応する。たとえば、文字列をメモリーに置くときにその終わりを示すために最後に終端記号を置く。このとき、終端記号のパターンには、文字と簡単に区別できるようなパターンが使われる。前述のASCIIコードでは、改行などを意味するコントロールコード（値が0から31までの文字で、機器の制御などに使われる）を使うことが多い。

なお、解説では、文字列の検索について行うが、実際には、文字だけでなく、さまざまなデータが対象になることがある。

メモリー内の文字列から検索

いま、検索対象となる文字列と探したい文字列（検索文字列）の2つがあるとする。実際には、メモリーのどこかに格納されていて、ともに終端記号で終わりが示されている。話を簡単にするために、どちらの文字列もASCIIコードからなる文字で構成されているとしよう。日本語の場合、1文字が16bitで表され、この中に8bitの文字などが混在してくると処理がちょっと面倒になるからだ。ただし基本的な考え方は一緒である。

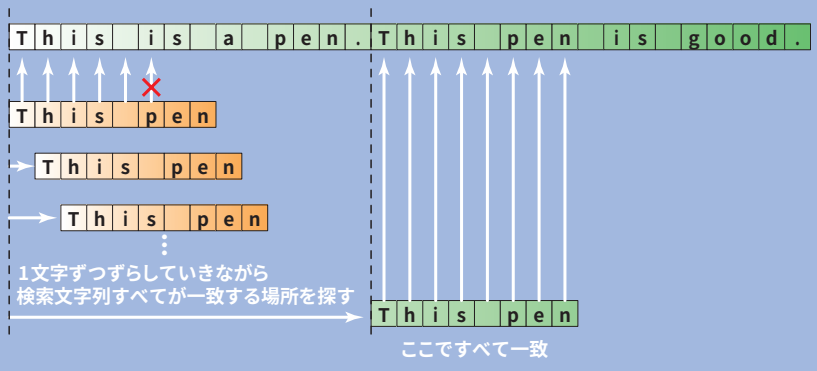

図2　最も簡単な文字列検索は、1文字ずつ比較を行い、不一致があれば、位置を1つずらして再度比較を行っていく方法である。

この場合の検索とは、文字列の中に検索文字列と一致するところがあれば、その場所（文字列先頭からの文字数）を答えとするものとする。この場合の答えは、必ず0以上（最初が1文字目だから）となる。もし見つからない場合には、0を答えとする。また、検索の条件として、検索対象文字列の中に含まれるすべての検索文字列を見つけることとする。これは、どんなときでも検索対象文字列を最後まで調べる必要があることを意味する。

一番簡単な方法は、1文字ずつ比較する方法である。検索文字列すべてが一致すれば、検索は終了である。一致していなければ1文字ずらして、一致を調べる、これを繰り返す（図2）。便宜的に不一致が起こったときに比較する位置をずらすことを「スライド」と呼ぶことにしよう。比較の開始位置を動かす量（文字数）をスライド量とする。この方法はコンピュータの性能に頼ったやり方で、「力ずく」（Brute Force）の方法ともいわれる。

検索文字列がすべて検索対象文字列に含まれない、つまり1文字も一致しない検索において、n文字の中からm文字を検索しようとすると、n-m+1回の比較を行うだけでいい。

細かな解説は省くが、mに対してnが十分大きな値なら、この処理の実行時間はnに比例する。このため、上記のやり方をそのまま実装しているシステムも少なくない。

ワープロの検索機能などのように、対象データもそれほど大きくなく、また、検索がたまにしか使われないのであれば、簡単にプログラムを作ることができるため、Brute Force方式が使われることもある。しかし、検索が頻繁に使われるアプリケーションや対象データが巨大になる場合には、もう少し工夫する必要がある。

もう少し効率的な方法

検索は、古くからコンピュータが行う処理の1つであったため、いろいろなやり方（アルゴリズム）が提案されてきた。こうしたメモリー上の文字列検索についても、効率化する方法がいくつかある。

ただし、こうした方法は、検索対象の文字の種類（アルファベットの数）や文字列の長さ、検索文字列の最大長といった条件により効率に違いが出てくる。このため、よいとされている手法であっても、検索対象に合わないと必ずしも効率的ではなくなってしまうことがある。

最初に解説した方法が非効率なのは2つ理由がある。1つは、スライド量が毎回1と決まっていることである。

もう1つは、前半部分は一致するが、後ろの方で失敗する場合である。最初の1文字が一致しなければ、すぐにスライドさせて次の場所から開始できるのに、途中まで一致しているから、その分スライドを行うまでの比較数が増えてしまう(図3)。

これを解消するために考えられたのがBM(Boyer-Moore)法と呼ばれる方法である(ちなみにBoyer、Mooreは考案者の名前)。

また、この方法は最初に解説した方法によく似ているが、比較を検索文字列の先頭からではなく、後ろから行う(図4)。このようにすると、文字列の前半部分が一致しているような場合であっても、無駄な比較を行わずにスライドさせることができる。

しかし、後半部分が一致したら、やはり結果は同じである。しかし、このとき、事前に作っておいた表を使うことで、1文字だけでなく、数文字スライドさせることが可能になるのである。

検索を行う前に、検索文字列について、各アルファベットが最後から何文字目に最初に現れるかを表にしておく(図5)。この表を「文字スライド表」と呼ぶことにする。

この表から、比較で一致しなかった文字(検索対象文字列中の文字)を探す。あれば、表にある数字だけスライドさせることができる(図6)。つまり、文字xが一致しなかったら、検索文字列の中のxの位置がそこに来るようにスライドさせることができるのだ。もし、文字xが検索文字列に含まれていなければ、その部分を飛ばして、xの次の文字に検索文字列の先頭が来るまでスライドさせることができる。

検索文字列の長さをmとし、先頭からi文字目で不一致が起こったとき、一致した部分の長さはm-iなので、文字スライド表から調べた値から引けば、スライド量を求めることができる。

このようにすることで、途中まで一致した場合でも、不一致のあとに大きくスライドさせることが可能になる。

さらに効率的にするには、検索文字列の中に含まれる繰り返しパターンに着目する。もし、後半部分と同じような文字のパターンが前半にあるなら、一致していた部分が重なるようにスライドさせることで、より大きくスライドさせることが可能になる。

たとえば、aabcabという検索文字列で、最後の「ab」まで一致し、cで一致しなかったとき、3文字ずらして、2文字目からのabというパターンがそのとき一致していたところまでずらすことが可能になる(図7)。

この移動量は、最初の表による移動量(この場合は、1)よりも大きく、表を使っ

図3　前半が一致し、後半で不一致の場合、複数の比較を繰り返すため、検索に時間がかかってしまうことになる。

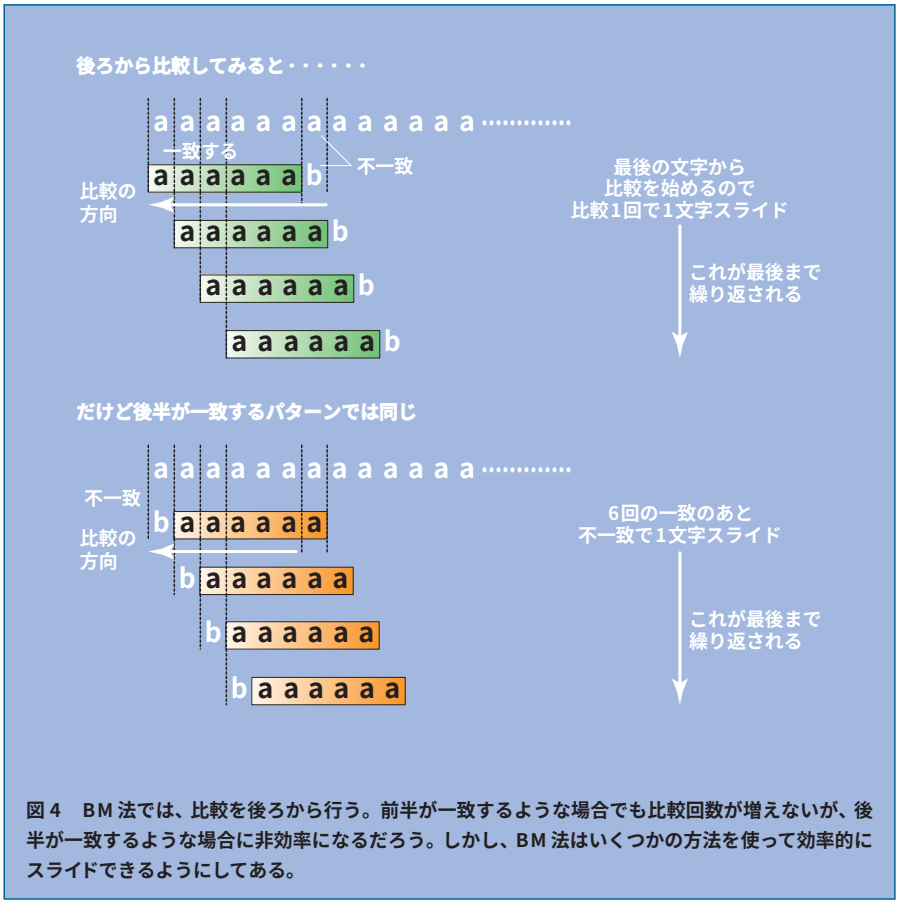

図4　BM法では、比較を後ろから行う。前半が一致するような場合でも比較回数が増えないが、後半が一致するような場合に非効率になるだろう。しかし、BM法はいくつかの方法を使って効率的にスライドできるようにしてある。

た場合よりも効率的になる。

このためには、やはり表を作らねばならない（こちらの表はパターンスライド表と呼ぶことにする）。このパターンスライド表は、検索文字列と同じ長さで、一致しなかった場所とそのときのスライド量を表すものだ。

実際のプログラムでは、不一致が起こったときに、文字スライド表とパターンスライド表からそれぞれスライド量を求め、大きなほうの値を採用する。

このようにすることで、毎回のスライド量を大きくできるため、最初に紹介した単純な方法よりも速く検索できる可能性が高くなる。また、最悪の場合でも、スライド量は1となり、最初の方法よりも悪くなる可能性もない。

文字列を最後まで調べるには、検索文字列の最初まで比較が行われたときの位置を結果として出力し、そこからさらにスライド量を求めて検索を続行する。そしてスライドしたときに検索文字列が後ろにはみ出すようならば、そこで検索が終了する。

その他の方法

このBM法は効率がよく、多くのプログラムで利用されている。また、これをベースに改良したアルゴリズムもあるようだ。たとえば、繰り返しパターンをもう少し詳細に分析しておいて、スライド量を増やすやり方がある。

検索はコンピュータの基本的な処理でもあり、これまでさまざまな研究が行われてきた。しかし、インターネットやデスクトップ検索が最近になって脚光を浴びたように、これからもさまざまな分野に応用されていく技術である。また、コンピュータの性能などが向上することで、過去には不可能だった方法が実現できたり、新しい方法が日々開発されたりしている。新しい技術が今後もさまざまな検索を可能にしていくことだろう。

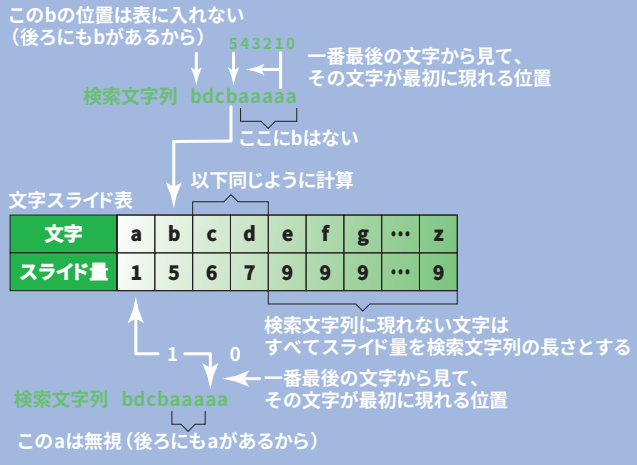

図5　文字スライド表は、検索文字列に現れる文字の種類とその最後の出現位置を記録したもの。これを使うことで、不一致が起こったときの文字を見て、スライド量を計算できる。

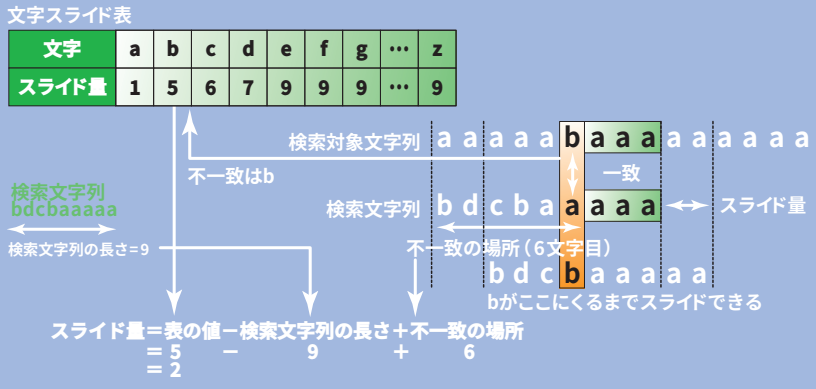

図6　検索中で不一致が起こったら、そのときの検索対象文字列側の文字から文字スライド表を引く。得られた数字と検索文字列の長さ、検索文字列内の不一致が起こった位置からスライド量を計算できる。

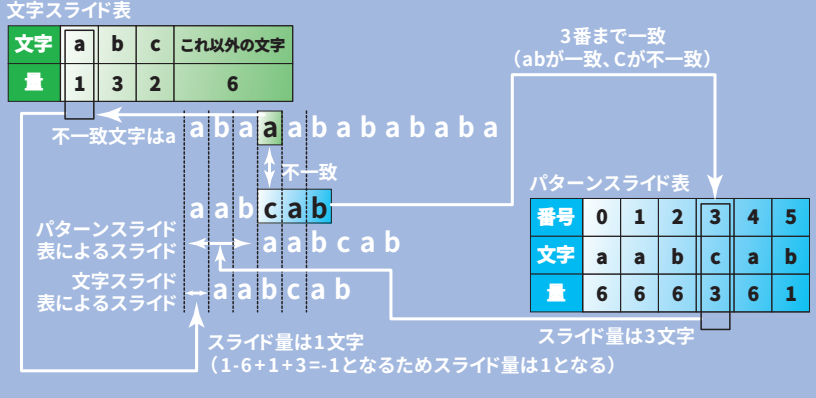

図7　パターンスライド表を使うと、文字スライド表よりもさらに大きくスライドさせることが可能になる。

［インターネットマガジン　バックナンバーアーカイブ］ ご利用上の注意

このPDFファイルは、株式会社インプレスR&D（株式会社インプレスから分割）が1994年～2006年まで発行した月刊誌『インターネットマガジン』の誌面をPDF化し、「インターネットマガジン　バックナンバーアーカイブ」として以下のウェブサイト「All-in-One INTERNET magazine 2.0」で公開しているものです。

http://i.impressRD.jp/bn

このファイルをご利用いただくにあたり、下記の注意事項を必ずお読みください。

- 記載されている内容（技術解説、URL、団体・企業名、商品名、価格、プレゼント募集、アンケートなど）は発行当時のものです。
- 収録されている内容は著作権法上の保護を受けています。著作権はそれぞれの記事の著作者（執筆者、写真の撮影者、イラストの作成者、編集部など）が保持しています。
- 著作者から許諾が得られなかった著作物は収録されていない場合があります。
- このファイルやその内容を改変したり、商用を目的として再利用することはできません。あくまで個人や企業の非商用利用での閲覧、複製、送信に限られます。
- 収録されている内容を何らかの媒体に引用としてご利用する際は、出典として媒体名および月号、該当ページ番号、発行元（株式会社インプレスR&D）、コピーライトなどの情報をご明記ください。
- オリジナルの雑誌の発行時点では、株式会社インプレス R&D（当時は株式会社インプレス）と著作権者は内容が正確なものであるように最大限に努めましたが、すべての情報が完全に正確であることは保証できません。このファイルの内容に起因する直接的および間接的な損害に対して、一切の責任を負いません。お客様個人の責任においてご利用ください。

このファイルに関するお問い合わせ先
株式会社インプレスR&D
All-in-One INTERNET magazine 編集部
im-info@impress.co.jp