2次元パターン高速探索アルゴリズム

3V-2

浦谷則好
(NHK放送技術研究所)

1. はじめに

土地利用区分図などのメッシュデータのコンピュータ利用や、ディスクトップパブリッシングの発展に伴ない、2次元パターンの高速な探索法が求められている。1次元パターンの探索アルゴリズムを2次元に拡張した例としては、Birdの仕事があるが、[1] この方法では探索されるデータ空間の大きさに比例した時間がかかる。筆者らはすでに複数の1次元パターンを同時に高速に探索するアルゴリズム(FAST法)を開発している。[2] このFAST法を2次元パターンの探索に拡張することによって、より高速なアルゴリズムを考案したので報告する。

2. 2次元FAST法のアルゴリズム

FAST法はAho-Corasick法[3]のパターン照合機械の考え方をBoyer-Moore 法[4]に取り入れることによって、複数の1次元パターンを同時に高速に探索できるようにしたものである。

FAST法を2次元に拡張するための基本的な考え方は以下のとおりである。2次元パターンとしては、説明の便宜上、ここでは2次元文字列を例にとる。

今、図1のようなパターンを(2次元の)テキストから探索する問題を考える。このパターン探索に2次元FAST法では2段階の探索を行う。まずパターンを列方向と行方向に分解して考える。すなわちパターンの各行を1次元パターンと見なしてパターン(行パターン)それぞれに固有の記号(行パターン記号:RP記号)を割り当てる。例えば図1の行パターン“abcde”にはA、“fghij”にはB……とRP記号を付ける。このとき、同じ行パターンには同じRP記号が付くようにする。このRP記号を列方向に並べたものを列パターンと呼ぶことにする。図2は図1の列パターンである。探索の第一段階では行パターンを(通常の)FAST法で探索する。行パターンが見つかれば、列方向にFAST法を用いて列パターンを探索する。列パターンが見つかったときに元の2次元パターンが存在していることがわかる。FAST法と同様にテキストの照合位置を決めるためにgoto関数を用いるが、この作成法はFAST法と同じなので説明を省略する。ただしgoto関数は行パターン、列パターンそれぞれに用意しておく。表1に列パターンに対するgoto関数を示す。2次元FAST法ではまずテキストのcmin行目(cminは列パターンの最小の長さ)から行パターン探索を始める。この行パターン探索の結果としてRP記号を2次元配列mapに記憶する。後述の列パターン探索はこのmap上で行う。このとき注意すべき点は同じ位置で2つの行パターンが検出されうることである。これは1つの行パターンが他の行パターンのprefixである場合に起こる。例えば図1の“abcde”が検出されるときには必ず“abcd”も検出される。この場合にはmap上に2つのRP記号AとGを記憶しなければならない。2次元FAST法ではこうした場合、RP記号を“未探索”に変えることで対処している。このmapの容量は
テキストの行長(m)×(列パターンの最大長(s)+1)
で済む。1行の行パターンの探索が終わった後で次にどの行を探索するかは、どの行パターンが検出されたかによって決まる。例えば行パターン探索で`B'と`I'のパターンが検出されたならば、表1のgoto関数から次の行パターン探索は2行先から始めれば良いことがわかる。もし表1のgoto関数が正ならば列パターンの最後の記号が見つかったことになるのでmapを対象に列パターン探索を開始する。対象となりうる全ての行の行パターン探索が済んでいれば単にFAST法を列方向に行うだけで良いが、前述のように全ての行が探索済みであるとは限らない。この場合mapには

```
abcde      mnop
fghij      abcd      A B C D E
pijkl      efgh
klmno      ijkl      F G H I G H
cdeij      abcd
           efgh
```

図2. 図1の列パターン

図1. 2次元パターンの一例

表1　列パターン用goto関数

RP記号＼状態	0	1	2	3	4	5	6	7	8	9	10	11
A	-4	-6	-7	-8	5	-10	-6	-7	-8	-9	-10	-11
B	-3	-6	-7	4	-9	-10	-6	-7	-8	-9	-10	-11
C	-2	-6	3	-8	-9	-10	-6	-7	-8	-9	-10	-11
D	-1	2	-7	-8	-9	-10	-6	-7	-8	-9	-10	-11
E	1	-6	-7	-8	-9	-10	-6	-7	-8	-9	-10	-11
F	-5	-6	-7	-8	-9	-10	-6	-5	-8	-9	11	-11
G	-1	-6	-7	-8	-9	-10	7	-7	-8	10	-10	-11
H	6	-6	-7	-8	-9	-10	-6	-7	9	-9	-10	-11
I	-2	-6	-7	-8	-9	-10	-6	8	-8	-9	-10	-11
(other)	-5	-6	-7	-8	-9	-10	-6	-7	-8	-9	-10	-11

A Fast Matching Algorithm for Two Dimensional Patterns

Noriyoshi URATANI
NHK Science and Technical Research Laboratories

“未探索”であることを示す記号が入っているので、この値を求めるために部分的に行パターンの探索を行う。

２次元ＦＡＳＴ法のアルゴリズム（照合部分のみ）を図３に示す。ただし変数の宣言等は省略してあり、ｍａｐは初期化されているものとする。$a_{i,j}$ はテキストのｉ行ｊ列目の文字を示す。

３．　実験結果

テキスト中の文字の平均照合回数で２次元ＦＡＳＴ法の効率Ｐを評価すると、図３のアルゴリズムからわかるように最良時には　　Ｐ＝１／（ｃｍｉｎ×ｒｍｉｎ）
となる。ここでｃｍｉｎ（ｒｍｉｎ）は列（行）パターンの最小の長さである。

２次元ＦＡＳＴ法の効率はテキストやパターンの性質と大きさによって大きく変化する。テキストもパターンも一様乱数によって生成し、実験した結果を図４に示す。図中でパラメータはｑが文字種、ｍがパターンの大きさ（ｍ×ｍのパターン）、ｋがパターンの数である。テキストの平均文字照合回数だけでは評価として不足であると思われるので、

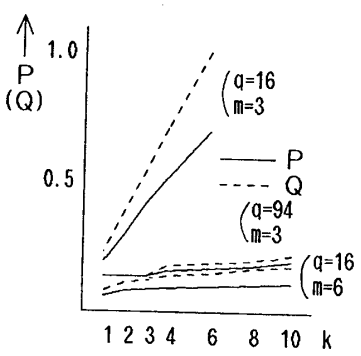

図４．　２次元ＦＡＳＴ法の効率

（文字照合回数＋ｍａｐの照合回数）／テキスト文字数による効率Ｑも同じ図中に破線で示す。

これを見ると予想されることながら文字種が多いほど、あるいはパターンが大きいほど２次元ＦＡＳＴ法の効率が良いことがわかる。

４．　おわりに

ＦＡＳＴ法を２次元に拡張した探索アルゴリズムを考案し、実験によって複数の２次元パターンを同時に高速に探索できることを確認した。この方法はアルゴリズムから容易に想像できるように一般のｎ次元パターンの探索に拡張することが可能である。

今後はさらに実験を進めて２次元ＦＡＳＴ法の性質を明らかにしていきたいと考えている。

＜参考文献＞

（１）R.S.Bird: “Two Dimensional Pattern Matching”
Inf.Process.Lett., Vol.6, No.5(1977),pp.168-170

（２）浦谷則好：“複数文字列照合アルゴリズム”
信学技報ＳＳ８７－２７(1988)

（３）Aho 他：“Efficient String Matching: An Aid to Bibliographic Search”
Comm. ACM, Vol.18, No.6(1975),pp.333-340

（４）Knuth 他：“Fast Pattern Matching in Strings”
SIAMJ. COMPUT., Vol.6, No.2(1977),pp.323-350

```
begin
  cst:=0;  qc:=cmin;  (最小の列パターン長)
  while qc≦n do begin
    mm:=cmax;  (最大の列パターン長)
    rst:=0;  dm:=cmin;
    qr:=rmin;  (最小の行パターン長)
    while qr≦m do begin
      if rgo(rst, a[qc,qr])<0 then begin
        qr:=qr-rgo(rst, a[qc,qr]); rst:=0
       end
      else begin
        rst:=rgo(rst, a[qc,qr])
        if rout(rst)>0 then begin
          if map[mm, qr]=0 then
            map[mm, qr]:=rout(rst);
          else if map[mm, qr]>0 then
            map[mm, qr]:=URS  (未探索に変更)
         end
        cst:=cgo(0, rout(rst));
        if cst<0 then begin
          if (-cst)<dm then dm:=-cst
         end
        else begin  (列パターン探索)
          len:=length(rout(rst));
          (行パターンの長さ)
          dm:=csearch()
         end
        qr:=qr-1
     end  end
    qc:=qc+dm;
    maprpl(map, dm)  (mapの並べ直しと初期化)
end  end
```

図３．　２次元ＦＡＳＴ法のアルゴリズム

```
function csearch()
  begin
    qs:=qc;  mc:=mm;
    while mm≦cmax do begin
      if cout(cst)≠empty then
        print qr, qs, cout(cst);
      mc:=mc-1;  qs:=qs-1;
      if map[mc, qr]=URS then
       begin  (mapの部分計算)
        j:=qr+len-1;  tst:=0;
        while j≦(qr+len-1) do begin
          tst:=rgo(tst, a[qs,j]);
          if tst<0 then return(dm)
          else begin
            if j=qr and rout(tst)>0
             then begin
               cst:=cgo(cst, cout(tst));
               if cst<0 then begin
                 if (mc-cst-cmin)<dm
                   then
                     return(mc-cst-cmax)
                   else return(dm)
                end
               else break
             end
            j:=j-1;
            if j<qr then return(dm)
       end  end  end
      else begin  (map上の探索)
        cst:=cgo(cst, map[mc, qr]);
        if cst<0 then begin
          if (mc-cst-cmax)<dm
            then return(mc-cst-cmax)
            else return(dm)
end  end  end  end
```