JAPLA 研究会資料　2013/12/7

幅優先探索－考え方とＪのプログラム
ゴールへのルート探索を例として

西川　利男

はじめに

「箱入り娘」という古くからのパズルがある。大きさの異なる大小の小片をスライドして決められた解のパターンにもって行くパズルである。以前、JAPLA の夏の合宿で紹介した[1]。そのときは、Ｊのプログラムとしては小片を移動させるグラフィックスだけであった。

久しぶりにこれを見直して、今度は解を探索するプログラムを作ろうと試みた。実はこの課題は Perl の正規表現のパターン・マッチング処理の例として Perl の本[2]にあり、Perl のプログラムコードが載せられている。

このような探索 Search はコンピュータ・アルゴリズムの基本であり、２つの方法がある。

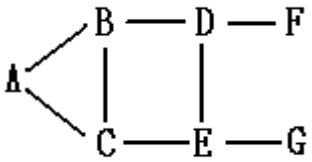
・深さ優先の探索（Depth First Search）＝　縦型探索
・幅優先の探索（Breadth First Search）＝　横型探索

Perl のプログラムは幅優先の方式であり、これを参考にＪでプログラミングしようとした。ところが、これに仲々てこずっている。

今回はそのための腕慣らしとして、決められた経路のゴールへのルート探索を例として、Ｊでの幅優先探索アルゴリズムの考え方とそのプログラムについて述べる。

１．経路のゴール探索

課題例　次のような連結した経路をとりあげる[3]。



ここで、起点 A から終点 G に至る経路をすべて求めること。

なお、正式にはこのような図形をグラフ（graph）といい、A,B などを頂点（vertex）または節点（node）、区間 AB, BC などは辺（edge）または弧（arc）と呼ぶようだが、ここでは分かり易い用語を使った。

[1] 西川利男「Ｊの正規表現プログラミング III 箱入り娘パズル」
JAPLA 研究会資料（蓼科夏の合宿）2005/8/6
[2] 増井俊之「Perl 書法」アスキー出版(1993), p.112-117
[3] 広井誠「パズルでプログラミング－第２回　幅優先探索と 15 パズル（前編）」
M. Hiroi's Home Page, http://www.geocities.jp/m_hiroi/

２．幅優先探索のアルゴリズムの考え方

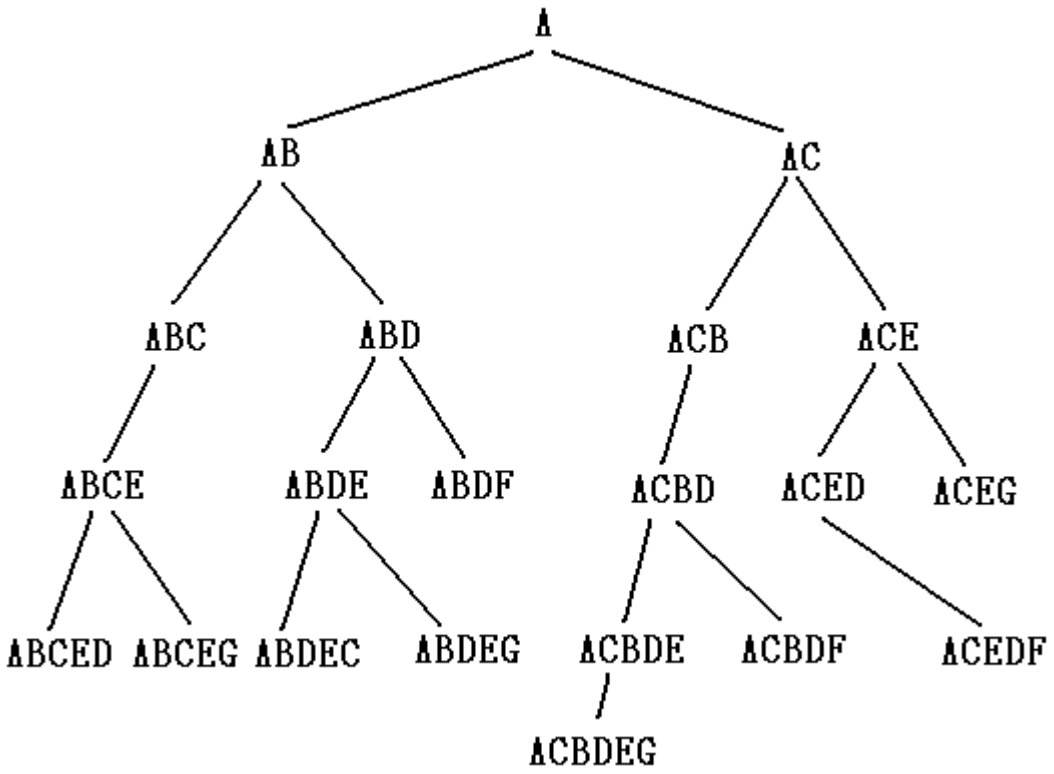
ふつう考えるように順次経路をたどっていくということから、発想を転換する。

それにはまず、区間経路を列挙することから始める。

AB, AC, BC, CB, BD, DE, CE, EC, DE, ED, DF, EG

そして経路探索とは、このような区間を記したカードがあったとして、これらを次々と取り出して、連続するように並べる、というように考える。

これから、下図のような経路探索の木を作る。



この経路探索の木をたどっていき、最後がGで終わるものがゴールへの経路である。

つまり解は

ACEG
ABCEG
ABDEG
ACBDEG

の４通りである。

このように、経路探索のアルゴリズムとは上の木構造について、階層レベルを定めて、そのレベルでは幅優先＝横方向に調べて、ゴールを見つける。そして階層レベルを１つずつ増やして行きつつ、次々と進めて行く。

これから分かるように、幅優先探索では、解は最短の経路から順次求められる。

３．Ｊのプログラム

３．１　準備

Ｊのプリミティブに名前をつけて読みやすくする。

```
head =: {.
behead =: }.
tail =: {:
curtail =: }:
区分経路データベース
DC1 =: 'AB';'AC';'BC';'CB';'BG'
DA1 =: 'AB';'AC';'BC';'CB';'BD';'DE';'CE';'EC';'DE';'ED';'DF';'EG'
```

３．２　make 探索の木の根を元に、葉を生成するプログラム

```
make =: 3 : 0"(1 0)
:
DB =. x.
if. 0 = # > y. do. '' return. end.  NB. revised 2013/11/12
Key =. tail > y.
Lk =. Key -.~ (> Key e. L:0 DB)#DB
LL =. Lk -. y.
LLL =. (Key = > head L:0 LL) # LL
LR =. ~. (< curtail > y.) ,L:0 (LLL)
tdouble =. */ L:0 ~: L:0 LR  NB. test whether duplicate ?
LR =.  (> tdouble) # LR      NB. remove the duplicated items
)
```

プログラムは左引数を区分の経路データベースとして、右引数にはそこにいたる経路を指定する。経路の最後をキーとして、経路データベースからキーを含むその他の項目を取り出し、それ以前の経路の後ろに追加したものを返す。このとき、以前に現れたものを排除することがポイントである。さもないと堂々巡りして先に進めない！

次のように実行される

```
   DA1 make <'A'
+--+--+
|AB|AC|
+--+--+

   DA1 make <'AB'
+---+---+
|ABC|ABD|
+---+---+

   DA1 make <'AC'
+---+---+
|ACB|ACE|
+---+---+

   DA1 make 'AB';'AC'
+---+---+
|ABC|ABD|
+---+---+
|ACB|ACE|
+---+---+

   DA1 make DA1 make <'A'
+---+---+
|ABC|ABD|
+---+---+
|ACB|ACE|
```

```
+---+---+
```

３．３　breadth　階層レベルを指定して、探索の途中経過を出力するプログラム

```
breadth =: 3 : 0
:
DB =. x.
Level =. y.
D =. <'A'
j =. 0
while. j < Level
  do.
    wr 'Level: ', ": j
    D =. , DB make D
    wr D
    wr testD  =. # L:0 D
    wr tt =. (j + 2) = L:0 testD
    wr (, > tt) # D
    j =. j + 1
  end.
'*** end. ***'
)
```

実行すると、次のようになされる

```
   DA1 breadth 5
Level: 0
+--+--+
|AB|AC|
+--+--+
+-+-+
|2|2|
+-+-+
+-+-+
|1|1|
+-+-+
+--+--+
|AB|AC|
+--+--+
Level: 1
+---+---+---+---+
|ABC|ABD|ACB|ACE|
+---+---+---+---+
+-+-+-+-+
|3|3|3|3|
+-+-+-+-+
+-+-+-+-+
|1|1|1|1|
+-+-+-+-+
+---+---+---+---+
|ABC|ABD|ACB|ACE|
+---+---+---+---+
Level: 2
+----++----+----+----++----+----+
|ABCE||ABDE|ABDF|ACBD||ACED|ACEG|
+----++----+----+----++----+----+
+-+-+-+-+-+-+-+-+
|4|0|4|4|4|0|4|4|
+-+-+-+-+-+-+-+-+
```

```
+-+-+-+-+-+-+-+-+
|1|0|1|1|1|0|1|1|
+-+-+-+-+-+-+-+-+
+----+----+----+----+----+----+
|ABCE|ABDE|ABDF|ACBD|ACED|ACEG|
+----+----+----+----+----+----+
Level: 3
+-----+-----+++-----+-----+---++-----+-----+++-----++---++
|ABCED|ABCEG|||ABDEC|ABDEG|ABD||ACBDE|ACBDF|||ACEDF||ACE||
+-----+-----+++-----+-----+---++-----+-----+++-----++---++
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|5|5|0|0|5|5|3|0|5|5|0|0|5|0|3|0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|1|0|0|1|1|0|0|1|1|0|0|1|0|0|0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-----+-----+-----+-----+-----+-----+-----+
|ABCED|ABCEG|ABDEC|ABDEG|ACBDE|ACBDF|ACEDF|
+-----+-----+-----+-----+-----+-----+-----+

Level: 4
+------++----++++++++----++----+----+++------++----++++++----++++----+----+++
|ABCEDF||ABCE||||||||ABDE||ABDE|ABDF|||ACBDEG||ACBD||||||ACED||||ACED|ACEG|||
+------++----++++++++----++----+----+++------++----++++++----++++----+----+++
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|6|0|4|0|0|0|0|0|0|0|4|0|4|4|0|0|6|0|4|0|0|0|0|0|4|0|0|0|4|4|0|0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+------+------+
|ABCEDF|ACBDEG|
+------+------+
*** end. ***
```

４．ゴール探索のプログラム search と実行例

```
search =: 3 : 0
:
DB =. x.
Level =. y.
D =. <'A'
j =. 0
while. j < Level
  do.
NB.     wr 'Level: ', ": j
    D =. , DB make"(1 0) D
    testD  =. # L:0 D
    tt =. (j + 2) = L:0 testD
NB.     wr (#D), (#> tt)
    if. (#D) ~: (#> tt) do. '*** end ***' return. end.
    D =. (, > tt) # D
NB.     wr D
NB.     wr 'G' e. L:0 D
    wr > (> 'G' e. L:0 D) # D
    j =. j + 1
  end.
'*** end ***'
)
```

```
   DA1 search 5
ACEG
ABCEG
ABDEG
ACBDEG
*** end ***
```

５．２分木グリッドのゴール探索

昨年、日本科学未来館の情報部門で、「巨大な数」と題するコーナーがあった。縦横・グリッド状に並べた２分木のゴール探索の場合の数が、次数が大きくなるにしたがって巨大な数になるというものである。

今回のＪの幅優先探索のプログラムを使ってこの問題に適用してみた。ここで必要になったのは、区間連結のデータベースを自動的に生成することで、次のプログラム condata でおこなう。

```
condata =: 3 : 0
C0 =. , 2 <\ "(1) y.
C1 =. , 2 <\ "(1) |: y.
C0, C1
(] , (|. L:0)) C0, C1
)
```

また、先の search を多少修正したプログラム route で検索を行う。左引数に区間データベース、右引数には起点と終点を指定して実行する。

まず3 x 3 の場合である。

```
 A ― B ― C
 |   |   |
 D ― E ― F
 |   |   |
 G ― H ― I
```

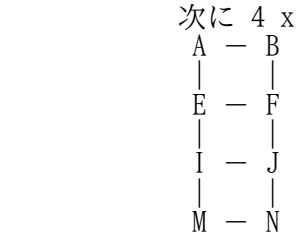
```
RD3 =: 3 3$'ABCDEFGI'
WAY3 =: condata RD3
WAY3
```

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|AB|BC|DE|EF|GH|HI|AD|DG|BE|EH|CF|FI|BA|CB|ED|FE|HG|IH|DA|GD|EB|HE|FC|IF|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

```
   WAY3 route 'AI'
No.1 ABCFI
No.2 ABEFI
No.3 ABEHI
No.4 ADEFI
No.5 ADEHI
No.6 ADGHI
No. 7 ABCFEHI
No. 8 ABEDGHI
No. 9 ADEBCFI
No.10 ADGHEFI
No.11 ABCFEDGHI
No.12 ADGHEBCFI
*** 12 routes found ***
```

3 x 3 の場合には、12 の経路が得られた。

次に 4 x 4 の場合をやってみよう。




```
   RD4 =: 4 4$'ABCDEFGHIJKLMNOP'
   WAY4 =: condata RD4
   WAY4 route 'AP'
No. 1 ABCDHLP
No. 2 ABCGHLP
No. 3 ABCGKLP
No. 4 ABCGKOP
No. 5 ABFGHLP
No. 6 ABFGKLP
No. 7 ABFGKOP
No. 8 ABFJKLP
No. 9 ABFJKOP
No.10 ABFJNOP
No.11 AEFGHLP
No.12 AEFGKLP
No.13 AEFGKOP
No.14 AEFJKLP
No.15 AEFJKOP
No.16 AEFJNOP
No.17 AEIJKLP
No.18 AEIJKOP
No.19 AEIJNOP
No.20 AEIMNOP
   (途中省略)
No.180 AEIMNJFGCDHLKOP
No.181 AEIMNJFBCDHLKOP
No.182 AEIMNJFBCDHGKLP
No.183 AEIMNJFBCDHGKOP
No.184 AEIMNJFBCGHLKOP
*** 184 routes found ***
```

さらに、5 x 5 ではかなり時間がかかった末、次のようになる。

```
   WAY5 route 'AY'
*** 8512 routes found ***
```

このように、場合の数は巨大な数になる。

```
NB. Breadth-first Search  by Toshio Nisikawa 2013/12/3

wr =: 1!:2&2
rd =: 1!:1

NB. Data ==============================
NB.   A - B - D - F
NB.   |   |   |
NB.   + - C - E - G

head =: {.
behead =: }.
tail =: {:
curtail =: }:

index_of =: i. ~

NB. Interval Data Base
DA =: 'AB';'AC';'BC';'BD';'CE';'DE';'DF';'EG'

DC =: 'AB';'AC';'BC';'BG'



NB. New Version / OK! - 2013/11/9 --------------------------------

DC1 =: 'AB';'AC';'BC';'CB';'BG'
DA1 =: 'AB';'AC';'BC';'CB';'BD';'DE';'CE';'EC';'DE';'ED';'DF';'EG'


NB. e.g. DC1 make <'AB' =>
NB. +---+---+
NB. |ABC|ABG|
NB. +---+---+
NB.      DC1 make <'AC' =>
NB. +---+
NB. |ACB|
NB. +---+
make =: 3 : 0"(1 0)
:
DB =. x.
if. 0 = # > y. do. '' return. end.  NB. revised 2013/11/12
Key =. tail > y.
Lk =. Key -.~ (> Key e. L:0 DB)#DB
LL =. Lk -. y.
LLL =. (Key = > head L:0 LL) # LL
LR =. ~. (< curtail > y.) ,L:0 (LLL)
tdouble =. */ L:0 ~: L:0 LR  NB. test of double
LR =.  (> tdouble) # LR
)

breadth =: 3 : 0
:
DB =. x.
Level =. y.
D =. <'A'
j =. 0
while. j < Level
  do.
```

```
    wr 'Level: ', ": j
    D =. , DB make D
    wr D
    wr testD  =. # L:0 D
    wr tt =. (j + 2) = L:0 testD
    wr (, > tt) # D
    j =. j + 1
  end.
'*** end. ***'
)

search =: 3 : 0
:
DB =. x.
'Start Goal' =. y.
D =. Start
Level =. 10
j =. 0
while. j < Level
  do.
NB.    wr 'Level: ', ": j
    D =. , DB make"(1 0) D
    testD  =. # L:0 D
    tt =. (j + 2) = L:0 testD
NB.    wr (#D), (#> tt)
    if. (#D) ~: (#> tt) do. '*** end ***' return. end.
    D =. (, > tt) # D
NB.    wr D
    Hit =. (> Goal e. L:0 D) # D
    if. 0 < #Hit do. wr > Hit end.
    j =. j + 1
  end.
'*** end ***'
)

NB. rename for comparing Prolog calc. 2013/11/30 ============================
WAY =:
'ab';'bc';'ae';'ef';'fe';'bf';'fb';'fg';'gf';'cg';'gc';'eh';'hi';'ih';'fi';'i
f';'ij';'gj'

NB. Hiro's Home Page ========================================
WAYH =:
'ab';'bc';'cb';'af';'ae';'eh';'he';'hi';'ih';'ij';'ji';'jk';'ef';'fe';'fi';'i
f';'fg';'gf';'fb';'bf';'fj';'jf';'gj';'jg';'gc';'cg';'cd'

NB. (way data) route ('Start, Goal')
route =: 3 : 0
:
DB =. x.
Level =. 50
'Start Goal' =. y.
D =. Start
HN =. 1
j =. 0
while. j < Level
  do.
NB.    wr 'Level: ', ": j
    D =. , DB make D
    testD  =. # L:0 D
```

```
    tt =. (j + 2) = L:0 testD
NB.     wr (#D), (#> tt)
    if. (#D) ~: (#> tt) do. goto_fin. end.
    D =. (, > tt) # D
    if. 0 = #D do. goto_fin. end.
    Hit =. (> Goal = L:0 (tail @ ,) L:0 D) # D
    if. 0 < #Hit
      do.
        HN0 =. HN + i. # Hit
        HNN =. ,. HN0
        NN =. ((# Hit),3)$'No.'
        NNN =. NN ,. ": HNN
        NB. wr > Hit
        HH =. ' ',"(0 1) > Hit
NB.         wr '-------'
        wr NNN ,. HH
    end.
    HN =. HN + # Hit
    j =. j + 1
  end.
label_fin.
'*** ', (": {: HN0), ' routes found ***'
)

RD3 =: 3 3$'ABCDEFGHI'
NB.     , 2 <¥ "(1) RD3
NB. +--+--+--+--+--+--+
NB. |AB|BC|DE|EF|GH|HI|
NB. +--+--+--+--+--+--+
NB.     <"(1) |: ,. > |: (L:0) 2 <¥ RD3
NB. +--+--+--+--+--+--+
NB. |AD|DG|BE|EH|CF|FI|
NB. +--+--+--+--+--+--+

NB. connect vertex to interval data ============================
condata =: 3 : 0
C0 =. , 2 <\ "(1) y.
C1 =. , 2 <\ "(1) |: y.
C0, C1
(] , (|. L:0)) C0, C1
)

WAY3 =: condata RD3

RD4 =: 4 4$'ABCDEFGHIJKLMNOP'
WAY4 =: condata RD4

RD5 =: 5 5$'ABCDEFGHIJKLMNOPQRSTUVWXY'
WAY5 =: condata RD5
```