JAPLA 研究会資料　2008/5/24

Ｊによる８クイーンーその２
解の探索プログラム
(ＪのLisp風プログラミング)

西川　利男

１．はじめにーパズルの探索と木構造

前回、８クイーンを人手により楽しんでみようという、Ｊの簡単なプログラムを紹介した[1]。８クイーン・パズルは人工知能の問題として、よく取り上げられる。

一般に、パズルやゲームはその進行の段階で、駒などを動かす「手」をコンピュータ上でどう表現するかが問題である。原理的には木(tree)と呼ばれるデータ構造で表すことになるが、それを使ってパズルの解にどう到達するか、という木の探索のアルゴリズムをプログラミングすることになる。

木構造はコンピュータ科学の基本のテーマであり、ふつうはPascal で説明されている。また、パズルは人工知能の問題であることから、Lisp によるプログラミングがよく行われる。

パズルにおける木構造の難しさは次のようにあげられよう。

① パズルの「手」の進行とともに、木構造が動的に作られ、ときには変化する。
② データ構造としての木とそのリンクをプログラミング言語でどう表現するか。
③ 最終ゴールに向けての木の探索のアルゴリズムをどうするか。

などいろいろな問題がある。ふつう教科書にある二分木の探索法(preorder, inorder, postorder traverse)だけで解決できる問題ではない。一般的にＪで木構造とその探索などをどう扱うかは興味深いテーマであるが、あらためて検討したい。

ここではもっと実用的に、以前筆者の著書の中で紹介したLisp による８クイーンのプログラム[2]を参考にしつつ、その機能をＪで「翻訳して」プログラミングしてみた。

２．８クイーンのＪによるLisp風プログラミング

Lisp で最も特徴となる機能は次の３つの基本関数である。そしてこれらはＪのプリミティブで容易に行うことができる。この考え方が非常に重要である。

```
car  <==> {.      リストの最初の要素を取り出す
cdr  <==> }.      リストの最初の要素を取り除いた残りを返す
cons <==> ,       ２つのリストを結合して、新しいリストを作る
```

８クイーンの盤面の位置を０オリジンで(行,列)と表す。
例えばクイーンを盤面の(0,0)、(1,2)、(2,4)に置いた状態は次のように表す。

```
   BDA =: (0,0);(1,2);(2,4)
   BDA
+---+---+---+
|0 0|1 2|2 4|
+---+---+---+
```

文献

[1] 西川利男「Ｊによる８クイーンーその１」JAPLA 研究会資料　2008/4/26
[2] 西川利男「プログラミング言語入門」p.284-288, HBJ 出版局(1990).

そして、動詞 conflict（中で動詞 threat をよんでいる）により次に置く場所が可能かどうかをチェックする。（プログラムリストを参照のこと）

```
   (3, 0) conflict BDA
1                       ==> ぶつかる
   (3, 1) conflict BDA
0                       ==> ぶつからない、置くことができる
   (3, 2) conflict BDA
1                       ==> ぶつかる
   (3, 3) conflict BDA
1                       ==> ぶつかる
   (3, 4) conflict BDA
1                       ==> ぶつかる
   (3, 5) conflict BDA
1                       ==> ぶつかる
   (3, 6) conflict BDA
0                       ==> ぶつからない、置くことができる
   (3, 7) conflict BDA
0                       ==> ぶつからない、置くことができる
```

このようにして、次の「手」として盤面(3, 1), (3, 6), (3, 7)には置くことが可能である。そこで、盤面のデータベース BDA にこの「手」を追加して、更新する。

```
   BDA =: BDA, <(3, 1)
```

可能な「手」の数は複数個あることが多いから、枝分れの木構造となる。この木構造を次々とたどって、盤面データベース BD の要素が８に達したらひとつの解が得られる。しかし、その途中で木の先端がなくなったときは、前に戻るバックトラックが必要になるなど、処理はかなり複雑である。また、木の traverse と同時にその経路の記録（これは盤面データベース BD で行われる）が重要なポイントになる。このような考え方のアルゴリズムに沿って以下の探索プログラムを作成した。

途中経過も分かるようにした探索のメインプログラム qx をコーディングごとに説明してみよう。

```
qx =: 3 : 0
BD =. ''                              盤面データベースの初期設定
I =. 0                                行の初期値
J =. 0                                列の初期値
K =. 0                                ステップ数の初期値
label_LOOP.
T =. (I, J) conflict BD               ぶつかるかどうかのテスト
if. T = 0                             ぶつからないとき
  do.
     BD =. BD, (<I, J)                盤面データベースに「手」を追加する
     if. 8 = #BD do.
        wr '==============='         盤面データベースが８個に達したら完成
        wr 'Congratulation!'
        BD
```

```
          return.
        end.
        wr 'Step:', ': K =. K + 1
        wr BD
        I =. I + 1
        J =. 0
        goto_LOOP.
    else.                               ぶつかったとき
        if. J < 7                       １行が終わっていないとき
          do. J =. J + 1                列を増やして、
              goto_LOOP.                  はじめに戻って繰り返す
          else.                         １行が終わってしまったとき
          label_AGAIN.
              YN =. rd 1
              if. 0 < #YN do. return. end.
              I =. {. > {: BD       盤面データベースから列を増やし,
              J =. 1 + }. > {: BD       (右横に進み)
              BD =. }: BD           盤面データベースから１つ落とす
              if. J > 7 do. goto_AGAIN. end. また繰り返す
              goto_LOOP.            はじめに戻って繰り返す
        end.
end.
)
```

実行すると次のようになる。

```
   qx ''
Step:1
+---+
|0 0|
+---+
Step:2
+---+---+
|0 0|1 2|
+---+---+
Step:3
+---+---+---+
|0 0|1 2|2 4|
+---+---+---+
Step:4
+---+---+---+---+
|0 0|1 2|2 4|3 1|
+---+---+---+---+
Step:5
+---+---+---+---+---+
|0 0|1 2|2 4|3 1|4 3|
```

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

```
Step:6
+---+---+---+---+---+
|0 0|1 2|2 4|3 1|4 7|
+---+---+---+---+---+
```

```
Step:7
+---+---+---+---+
|0 0|1 2|2 4|3 6|
+---+---+---+---+
Step:8
+---+---+---+---+---+
|0 0|1 2|2 4|3 6|4 1|
+---+---+---+---+---+
Step:9
+---+---+---+---+---+---+
|0 0|1 2|2 4|3 6|4 1|5 3|
+---+---+---+---+---+---+
Step:10
+---+---+---+---+---+---+---+
|0 0|1 2|2 4|3 6|4 1|5 3|6 5|
+---+---+---+---+---+---+---+
```

（途中省略）

```
Step:109
+---+---+---+---+
|0 0|1 4|2 7|3 5|
+---+---+---+---+
Step:110
+---+---+---+---+---+
|0 0|1 4|2 7|3 5|4 2|
+---+---+---+---+---+
Step:111
+---+---+---+---+---+---+
|0 0|1 4|2 7|3 5|4 2|5 6|
+---+---+---+---+---+---+
Step:112
+---+---+---+---+---+---+---+
|0 0|1 4|2 7|3 5|4 2|5 6|6 1|
+---+---+---+---+---+---+---+
===============
Congratulation!
+---+---+---+---+---+---+---+---+
```

```
|0 0|1 4|2 7|3 5|4 2|5 6|6 1|7 3|
+---+---+---+---+---+---+---+---+
```

ひとつの解を見つけるまでにはかなりのステップを要する。

３．８クイーンの解－実行例

queen ’’

```
No. 1
+-+-+-+-+-+-+-+-+
|*| | | | | | | |
+-+-+-+-+-+-+-+-+
| | | | |*| | | |
+-+-+-+-+-+-+-+-+
| | | | | | | |*|
+-+-+-+-+-+-+-+-+
| | | | | |*| | |
+-+-+-+-+-+-+-+-+
| | |*| | | | | |
+-+-+-+-+-+-+-+-+
| | | | | | |*| |
+-+-+-+-+-+-+-+-+
| |*| | | | | | |
+-+-+-+-+-+-+-+-+
| | | |*| | | | |
+-+-+-+-+-+-+-+-+
```

```
No. 2
+-+-+-+-+-+-+-+-+
|*| | | | | | | |
+-+-+-+-+-+-+-+-+
| | | | | |*| | |
+-+-+-+-+-+-+-+-+
| | | | | | | |*|
+-+-+-+-+-+-+-+-+
| | |*| | | | | |
+-+-+-+-+-+-+-+-+
| | | | | | |*| |
+-+-+-+-+-+-+-+-+
| | | |*| | | | |
+-+-+-+-+-+-+-+-+
| |*| | | | | | |
+-+-+-+-+-+-+-+-+
| | | | |*| | | |
+-+-+-+-+-+-+-+-+
```

```
No. 3
+-+-+-+-+-+-+-+-+
|*| | | | | | | |
+-+-+-+-+-+-+-+-+
| | | | | | |*| |
+-+-+-+-+-+-+-+-+
| | | |*| | | | |
+-+-+-+-+-+-+-+-+
| | | | | |*| | |
+-+-+-+-+-+-+-+-+
| | | | | | | |*|
+-+-+-+-+-+-+-+-+
| |*| | | | | | |
+-+-+-+-+-+-+-+-+
| | | | |*| | | |
+-+-+-+-+-+-+-+-+
```

```
|  |  |*|  |  |  |  |  |
+--+--+--+--+--+--+--+--+
```

（途中省略）

```
No. 91
+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |*|
+--+--+--+--+--+--+--+--+
|  |  |*|  |  |  |  |  |
+--+--+--+--+--+--+--+--+
|*|  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+
|  |  |  |  |  |*|  |  |
+--+--+--+--+--+--+--+--+
|  |*|  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+
|  |  |  |  |*|  |  |  |
+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |*|  |
+--+--+--+--+--+--+--+--+
|  |  |  |*|  |  |  |  |
+--+--+--+--+--+--+--+--+

No. 92
+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |*|
+--+--+--+--+--+--+--+--+
|  |  |  |*|  |  |  |  |
+--+--+--+--+--+--+--+--+
|*|  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+
|  |  |*|  |  |  |  |  |
+--+--+--+--+--+--+--+--+
|  |  |  |  |  |*|  |  |
+--+--+--+--+--+--+--+--+
|  |*|  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |*|  |
+--+--+--+--+--+--+--+--+
|  |  |  |  |*|  |  |  |
+--+--+--+--+--+--+--+--+

*** end ***
```

完成したプログラム queen では動詞 disp_8Q を用いて盤面図で表した。

このようにして、解として 92 例が得られた。イギリスの Glaisher*)により回転、鏡像の対称、点対称から重複したものを除いたユニークな解は 12 通りといわれている。

*)「エイト・クーン」http://ja.wikipedia.org/wiki/

```
NB. J-Lisp / Lisp_styled Programming in J
NB. 8-Queen Back_Track Program
NB.   by Toshio Nishikawa 2008/4/14

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

NB. Lisp Functions in J
car =: {.
cdr =: }.
cons =: ,

NB. Test Program in J-Lisp Style ======================================
member =: 3 : 0
L =. car y.
M =. cdr y.
if. 0 = #M
  do. 0
  else.
    if. L = car M
       do. 1
       else. member L, cdr M
    end.
end.
)

append =: 3 : 0
L =. car y.
M =. cdr y.
if. 0 = #L
  do. M
  else. cons (car L), (append (cdr L), M)
end.
)

NB. 8-Queens Program in J-Lisp Style ====================================
BDA =: (0,0);(1,2);(2,4)

threat =: 3 : 0
'I J K L' =. y.
if. +. /  (I=K), (J=L), ((|I-K) = (|J-L)), ((|I-K) = (|L-J))
  do. 1
  else. 0
end.
)
```

```
conflict =: 3 : 0
:
'N M' =. x.
D =. y.
i =. 0
T =. 0
while. i < #y.
  do.
    CARD =: > car D
    T =. T +. threat N, M, ({. CARD), (}. CARD)
    D =. cdr D
    i =. i + 1
end.
T
)
NB. Graphic Display BD
NB. eg. disp_8Q QGoal
disp_8Q =: 3 : 0
<"(0) '*' y. } (8 8)$' '
)


NB. 8-Queen
NB. usage: queen ''
queen =: 3 : 0
BD =. ''
NOQ =. 1
I =. 0
J =. 0
label_LOOP.
T =. (I, J) conflict BD
if. T = 0
  do.
      BD =. BD, (<I, J)
      if. 8 = #BD do.
         wr 'No. ', (": NOQ)
NB.      wr BD
         wr disp_8Q BD
         QGoal =: BD
         YN =. rd 1 NB. CR => continue, 1 CR => end
         if. 0 < #YN do. (I, J) return. end.
         NOQ =. NOQ + 1
         I =. 0
         goto_NEXT.
```

```
      end.
NB.       wr BD
      I =. I + 1
      label_NEXT.
      J =. 0
      goto_LOOP.
  else.
      if. J < 7
        do. J =. J + 1
            goto_LOOP.
        else.
            label_AGAIN.
NB.              I =. {. > {: BD
              try. I =. {. > {: BD catch. '*** end ***' return. end.
              J =. 1 + }. > {: BD
              BD =. }: BD
              if. J > 7 do. goto_AGAIN. end.
            goto_LOOP.
      end.
end.
)
```