

Knight-tour のプログラム

東大 計数工学科 寛 捷彦

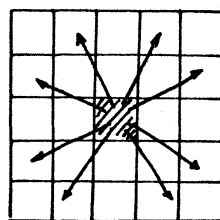
§ 1. 序

パズルを計算機で解こうという場合の多くは、理論によ、て裏付けられてうまい方法があるわけではなくて、結局しらみつぶしに場合を尽くして解を求めることに落ちつくようである。あとはいかに無駄な場合の調査を省くかに工夫を求めることになる。

Knight-tour の問題に興味をもったのは、知人のみやげにケンブリッジの TSS のプログラムを貰ったのがきっかけであった。BCPL で書かれたそのプログラムを解説して、このパズルの存在を知り、しばらくの間取組んでみた顛末を報告する。

§ 2. パズル

Chess の Knight の動き方は他の駒に比べ特異であり、この動き方に基づいたパズルは



いくつか考えられるが、その一つが Knight-tour である。

与えられた盤上で、各マス目をただ一度ずつ通過して全部のマス目を動く Knight の道を見出すこと。

$n \times n$ の盤について調べてみると、 $n=2, 3, 4$ の時は解が存在しない。 $n=5, 6$ では解が存在する事がわかった。また $n=8$ (chess board) にも解は存在する。

§3. しらみつぶし

人手で調べて解の存在がわかったので、まずは計算機で唯一つでもよいから解を出させてみようと、「しらみつぶし」方法でプログラムを組んでみた結果が下図である。($n=8$)

Chess board のように 白黒塗り分けると、Knight は 必ず 異なる色のマス目へ動くから、高々 32 個の区別がつけば $n=8$ の時の通過点の番号はわかる。1~9A~W を使って 32 進ダンプリシのが下図で、0 は未通過点を表わす。

```

10000000 00100000 20000000 00200000 30000000 00300000 40000000 00000000
10008000 00100070 20007000 00200060 30006000 00300050 40005000 00400000
10C9800A 09108A70 20087000 00200060 30006000 00300050 40005000 00400000
10C9800A C9108A70 20087000 00200060 30006000 EG300050 40F05000 F0400000
10C9801A C91JBA70 2JD871RH D02K0460 30EG6000 EG300050 40F05000 F0400000
10C9801A C91JBA70 2JD871RH D02K0460 3KEG6NP0 EG3L005N 4LF05M00 F04M0000
10C9801A C91JBA70 2JD871RH D02KPH60 3KEG6NPQ EG3L005N 4LFR5MQQ FS4M0R0Q
10C9801A C91JBA70 2JD871RH D02KPH60 3KEG6NPQ EG3L005N 4LFR5MQQ FS4M0R0Q

```

第7行目をマス目に書き入れてみると、Knight は 8 進動いてはいるが、⑤の所へ進んだところで地点αを孤立させてしまっているので、既に道を誤っていることになる。東大大型ヒータで30秒走らせた時、この誤った道のしらみつぶしで

1	⊗	C	9	8		I	A
C	9	1	J	B	A	7	
2	J	①	8	7	I	B	H
D		Z	K	P	H	6	
3	K	E	G	6	N	P	
E	G	3	L	O	Q	5	N
4	L	F	Ⓜ	5	M	Ⓞ	Q
F	S	4	M		R		

や、と J の所まで戻って来たところである。これではよほどの事が無い限りは唯一つの解に達することさえできそうにもない。

§ 4. Feasible test

そこで、こうした孤立点が生じないように、すべての未通過点に現在地から Knight の動きで行きつくことができることを確かめながら step を進めることにした。たまたま倍長のレジスタを使うと 64 bits で各 bit に各マス目を対応させることができるから、上の条件を満たすか否かを調べるプログラムをアセンブラで組込んで走らせてみたのが下図である。

```

10000000 00100000 20000000 00200000 30000000 00300000 40000000 00000000
10000000 00100000 20000000 00200000 30006000 00300050 40005000 00400000
10008000 00100070 20007000 00200060 30006000 00300050 40005000 00400000
10008000 00100070 20007000 00200060 30006000 00300050 40005000 00400000
10C9800A 0910BA70 20087000 00200060 30006000 00300050 40005000 00400000
10C9800A 0910BA70 20087000 00200060 30006000 00300050 40005000 00400000
10C9800A C910BA70 20D87000 00200060 30E06000 00300050 40005000 00400000
10C9800A C910BA70 20D87000 00200060 30E06000 00300050 40E05000 00400000
10C9800A C910BA70 20D87000 00200060 30E06000 00300050 40E05000 00400000

```

1	⊗	C	9	8			A
C	9	1		B	A	7	
2		①	8	7		B	
D		Z				6	
3		E		6			
E	G	3				5	
4		F		5			
F		4					③

同じく第7行目をマス目に記入してみると左図のようになる。孤立点は無くな、たが、⊗, ③の点と共に端点の形に強制されていて、このまま step を進めることが無意味であることがわかる。③が端

点に強制されたのは'5'を選んだ時であるから、⑤の次には、
 ④が端点として残らぬように④自身を次の通過点として選ば
 ねばならなかったことがわかる。同じく30秒走らせた時、
 やつとE点の選択変更点戻ってきたところであった。

§3,4の方法から得られた結果を反省してみると

- ・ 誤った直に入ったことをなるべく早く検出すること。

D stepで既に誤った時は各step 8方向あるとして $2^{3 \times (64-18)}$
 $= 2^{108} \approx 10^{30}$ 回ムダな調査を進めることになる。(実際はもう少しが)

- ・ *feasible test* にしても孤立点が生じる可能性は、1step
 進めたとき、そのマス目から knightの動きの関係にある
 高々8個の点であるから、前回の状態を上手に覚えてお
 けば、かなりの手間が省ける。

§5. Algorithm

各マス目を $\langle x, y \rangle$, $\langle x, y \rangle$ から knight が動きうるマス
 目の数を l_{xy} としよう。knight が次に動く点として $\langle x, y \rangle$
 を選ぶ条件は次の様になる。

1. $\langle x, y \rangle$ は現地点から桂馬の関係にあり (1-1)

盤内かつ未通過点であること。 (1-2)

現地点につき(2-3)の条件があるときは

それを満す点であること。 (1-3)

2. $\langle x, y \rangle$ から桂馬の関係にある点を $\langle x^*, y^* \rangle$ として

$lx^*y^* := lx^*y^* - 1$ とする。

- $lx^*y^* = 0$ なる $\langle x^*, y^* \rangle$ が存在するとき

$\langle x, y \rangle \xrightarrow{\text{行止り}} \langle x^*, y^* \rangle$

$\langle x, y \rangle$ が tour の 最終点の1つ手前の時のみ

可. (この時解に達する) 他は不可 (2-1)

- $lx^*y^* = 1$ なる $\langle x^*, y^* \rangle$ が存在するとき

$\langle x, y \rangle \xrightarrow{\text{1本道}} \langle x^*, y^* \rangle$

$\left[\begin{array}{l} \langle x, y \rangle \text{ の次点として } \langle x^*, y^* \rangle \text{ を選ばない限り} \\ \langle x^*, y^* \rangle \text{ は 端点となる。} \end{array} \right]$

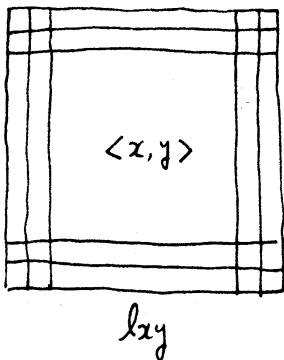
- $(l_{uv} = 1 \text{ なる点の数}) \geq 3$ 不可 (2-2)

- $(l_{uv} = 1 \text{ なる点の数}) = 2$

$\langle x, y \rangle$ の次の点として $lx^*y^* = 1$ なる点 $\langle x^*, y^* \rangle$ の

1つを選ぶ (2-3)

プログラムとしては、条件(1-2)も条件(2)と同等にし
らべられる様に、与えられた盤よりるまわり大きな配列とし



て lx_y を用意し、通過点及び盤外点に
対応する lx_y の値を負にとるようにし
てある。

この *algorithm* を用いてプログラムを作り、第1の点として $\langle 1, 1 \rangle$ を与え、 $l_{11} < 0$, $l_{11}^{**} := l_{11}^{**} - 1$ に初期値を設定して解かせてみたところ、10秒(印刷枚数の方で打切られた)で98個の解がみつかった。丁度第1解の34番目より先だけを変更して得られる解を尽くしているところで打切られていた。この調子で解が存在するとすれば、 $\langle 1, 1 \rangle$ を第1点とするものだけで(各 *step* でそれぞれえつつ解に達する選び方があるとして) $2^{34} \times 100 \approx 10^{12}$, 更に第1点の選び方が対称性を考慮しても16通りあるから、気の遠くなるような数になる。

§ 6. Variation

そこで解の形に特徴のあるものを求めてみることにした。

a. 閉じた道

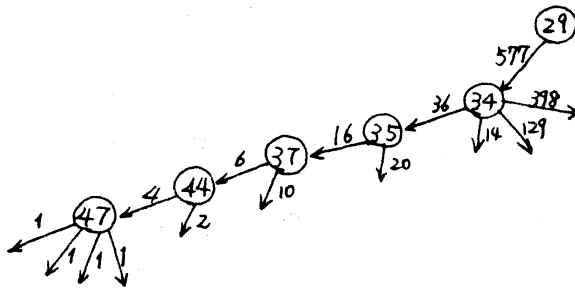
§ 5 の *algorithm* を用いて、初期値設定を変える。即ち、第1の点として $\langle 1, 1 \rangle$ を与え、 l_{11} , l_{11}^{**} は変更しないままにしておく。 $l_{11} = 2$ であることを利用して、条件として $l_{uv} = 1$ なる点の数が1の時(これは第2点を選んだ時にのみ生じる)は、 $\langle u, v \rangle$ を次の点として選んではならぬ(2-4)を付加することで閉じた道を探し出すことができる。図-1に第1番目に見つかった解を示す。30秒で600個の解がみつかり、29番目より先だけを変更して得られる解が577個

あった。

ANSWER(64)	1	8	11	22	3	18	13	16
	10	23	2	7	12	15	4	19
	55	64	9	24	21	6	17	14
	30	35	54	63	40	25	20	5
	47	56	29	36	53	62	41	26
	34	31	46	39	28	37	52	61
	57	48	33	44	59	50	27	42
	32	45	58	49	38	43	60	51

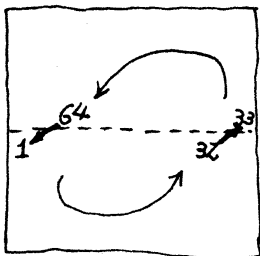
図-1

解の存在状況は下の様であった。



b. 点対称な解

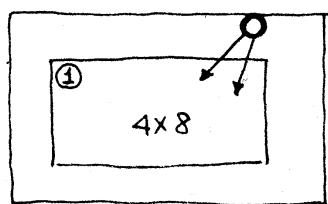
閉じた道もまたあまりに多くの解が存在するので更に条件を付加してみた。盤の半分だけをまずうめつくし、次に残りの



の半分をうめつくす解で、閉じた道でありかつ、中心に対して対称な点の通過番号の差が32であるようなものを求めてみた。

これも同じ algorithm を用いて行ってみた。

解に対する制約から、 4×8 の盤を考えて、第1点を与えた

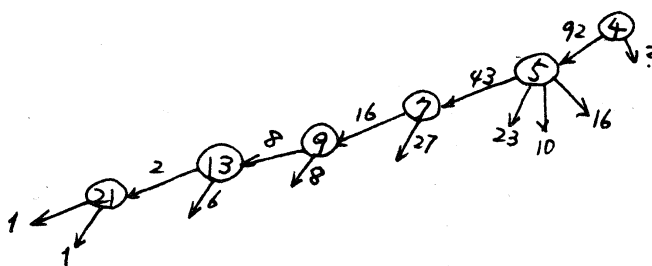


時、その対称点に対応する lij を(番外)
 あたかもそこへも動ける様子に与え(従って
 $li*j^*$ も変化させておく)る。条件として
 この lij へ行かぬ様子は $(2-4')$ として求めた第1解は下の様なもの
 である。

1 TH ANSWER AT 5724

①	22	1	28	9	③	12	18
6	25	4	21	14	17	12	31
23	2	27	8	29	10	19	16
20	3	24	5	20	15	30	11

第1点を隅に与えた時、30秒で丁度100個の解が求まった。
 第4点以後を変化させた解の個数は92個。存在状況は次の様
 であつた。



§7. まとめ

始めは解があるか否か確かめよう位のつもりでとりかかった
 が、あまりに解の数が多すぎて興味がそがれてしまった。一
 度 8×8 の解の総数を計算してみたいとは思っているが、今

のままの *algorithm* では計算時間があまりにかかりすぎるので更に工夫を要する。

今後は、「人間が学習し」ながら *algorithm* を作り出してゆくのではなく、「計算機自身に学習させ」るプログラムを考えてみたいと思っている。そうやってこそ計算機によるパズル解きの面白みもでてくるのではないだろうか。

付録として §6.b に用いたプログラムのこの *algorithm* 部分をつけておいた。

SUBROUTINE KNIGHT
 COMMON /P/ ENT, X, Y, U, IS *trial count*
 COMMON /Q/ C, IC, IT, SC *trial count*
 COMMON /R/ B, L, DX, DY *law=1 の点の数*
 COMMON /S/ SP, MAX, SX, SY, SD, SL, stack *陣の分布をいろう*
 INTEGER ENT, X, Y, U
 INTEGER C
 INTEGER SC(64)
 INTEGER B(8,8), L(12,12), DX(8), DY(8) *lxy*
 INTEGER SP, SX(64), SY(64), SD(64), SL(64) *knightの動き方を入れてある*
 GO TO (10, 20, 30), ENT

→10 D=1 *第1の方向にset*
 B(X,Y)=SP *<x,y> ← movementの番号*
 IT=IT+1

→11 X=SX(SP)+DX(D)
 Y=SY(SP)+DY(D) *{ 次の点の候補 <x',y'> (1-1)*
 LL=L(X+2,Y+2) *画面点, 除外 (1-2)*
 IF (LL.LT.0) GO TO 13 *除外 (2-4')*
 IF (Y.LE.0) GO TO 13 *(1-3)*
 IF (IS.EQ.2 .AND. LL.NE.1) GO TO 13
 NF=0
 DO 12 I=1,8
 M=X+DX(I)
 N=Y+DY(I)
 K=L(M+2,N+2)-1
 L(M+2,N+2)=K
 IF (K.EQ.0) NF=NF+1
 IF (K.EQ.1) IS=IS+1

12 CONTINUE
 IF (LL.EQ.1) IS=IS-1
 IF (NF.NE.0) GO TO 32 *(2-1) → 32*
 IF (IS.GT.2) GO TO 30 *(2-2) → 30*
 SD(SP)=D *<x,y> の方向 push-down*
 SP=SP+1
 SX(SP)=X
 SY(SP)=Y
 SL(SP)=LL
 SC(SP)=0
 L(X+2,Y+2)=-1 *lxy' < 0*
 GO TO 10

→13 IF (D.GE.8) GO TO 14
 D=D+1
 GO TO 11 *方向尽くした場合は*

14 IF (SP.LT.2) GO TO 33 *stack の空 (終了)*
 X=SX(SP)
 Y=SY(SP)
 IC=SC(SP)
 LL=SL(SP)
 L(X+2,Y+2)=LL
 ENT=2
 RETURN *途中経過印刷用の return*

20 SP=SP-1
 D=SD(SP)
 SC(SP)=SC(SP)+IC
 30 IF (LL.EQ.1) IS=IS+1
 DO 31 I=1,8
 M=X+DX(I)
 N=Y+DY(I)

31 *pop up 方向の復元*
lxy の復元*
付録-1

```

K=L(M+2,N+2)
L(M+2,N+2)=K+1
IF( K.EQ.1 ) IS=IS-1
31 CONTINUE
GO TO 13
③ IF( SP.NE.MAX-1 .OR. NF.NE.1 ) GO TO 30 条件(2-1)の検査→
R(X,Y)=MAX
SC(SP)=SC(SP)+1 ↓ 解に到達
C=C+1
ENT=3
RETURN } 解の印刷用の return
33 ENT=4
RETURN } すべての命令の尽くしおわり
END

```