第5部 パズル

巡回数

問題 1

2倍して桁が巡回する最小の自然数nを求めてください.

1234の2倍が, 2341, 3412, 4123のいずれかになれば, 1234は2倍して桁が巡回するといえます. また, 123の巡回として, 2301も, 23001も, 認めることにします.

問題 2

ある自然数について, $\forall n \in \{2,3,4,\cdots,9\}$に対し, n倍して桁数が巡回するものが存在することを示してください. もちろん, その数が求まれば, それで存在の証明になります. もし, 存在しないなら, それを証明してください.

上位桁に0があるのを認めるのは前問と同じです.

問題1について, 計算機で探索する方法をとった場合には, その探索が停止することを保証してください. 発見できない場合に, 永久に停止しないような探索は行なわないようにしてください. 探索の範囲が有限であることを保証する説明を必要とします.

第5部 パズル

ナンバーリンク

以前,「図26でAとa, Bとb, CとcをA交わらないように結べ」という(初めて見た人は大人でも結構楽しめる) パズル風の問題が幼稚園の入園問題に出たという話を聞いたことがあります(真偽はわかりません). 人間はどうしても最短距離で結ばれる線の周辺に集中して考える傾向があるようで, この傾向に

固執する人は図26の問題でもむずかしく感じるのでしょう．しらみつぶしが得意な計算機ならこんな盲点はないといえるのですが，少し大きな問題となると全探索では所要時間が気になります．

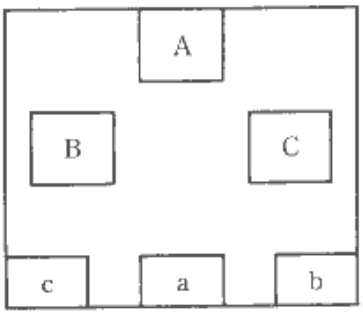

図 26

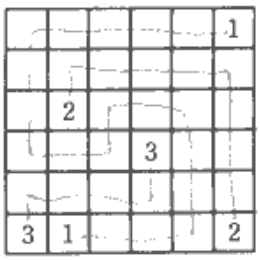

図 27

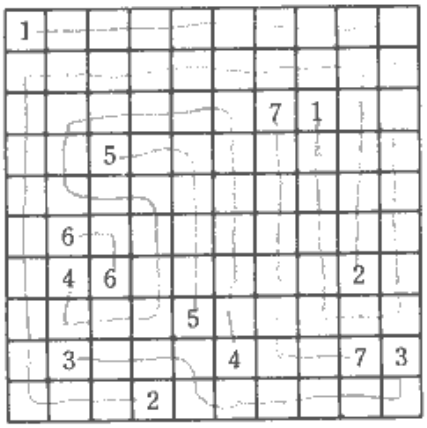

図 28

図27，図28は，雑誌「ニコリ」にナンバーリンクという名前で出ている問題です．解くルールは以下のとおりです．

（1）　同じ数字どうしを線でつなげる．

（2）　ワクの外に線を出してはいけない．

（3）　1マスに1本だけ通過できる（交差はダメ）．

（4）　線は縦横に引く．

「ニコリ」には全マスを使うということも書かれていますが，これは解くルールというより問題としての条件（解がユニークで，そのとき使われていないマスはない）になりますので，ここでは省いておきます．

問題1

図27程度の規模の問題を解く簡潔なプログラムを示してください．とりあえずは効率よりスマートさを重視して書いてください．

問題2

図28の程度の規模の問題を妥当な時間内に解く簡潔なプログラムを示してください．できれば人間が使っているようなヒューリスティクスを取り入れていて，かつスマートなプログラムを示してください．

表 11 {2,3,…,9}で巡回する数

n	0588235294117647
×2	1176470588235294
3	1764705882352941
4	2352941176470588
5	2941176470588235
6	3529411764705882
7	4117647058823529
8	4705882352941176
9	5294117647058823

表 12 {2,3,…,9}で巡回する最小桁数の数

n	
2	142857
3	076923
4	025641
5	142857
6	142857
7	010752688172043
8	0126582278481
9	010989

が巡回します．このような数は，1/19，1/23 など多数あります．

●補足

前述の定理を使用しますと，2 倍，3 倍,…, 9 倍のそれぞれについて最小の巡回数を求めることができます．それを**表 12** に示します．

巡回数はダイヤル数として，中公新書「続・数理パズル」に記載があります．

巡回数が，循環小数と深い関係があり，巡回数を求めることが最大公約数を求める問題に帰着するのは，美しい関係であると思います．前述の定理で数は 10 進数に限らず，何進数でも同じような定理が成り立ちますので，何かのときに，この定理が役に立つかもしれません．

第5部 パズル

ナンバーリンク

もちろん規模が異なる同じ種類の問題ですので，1つのプログラムで問題1，問題2の解答とすることができます．しかし，この問題は典型的なバックトラッキング問題で，しかも，出題の図28程度の規模の問題になると，その計算量は，単純に考えても，バックトラッキング・プログラムの演習としてよく知られている8クイーン問題での計算量をはるかに越えるものです．

そこで計算量を極力抑えるべく枝刈りが必要になってきますが，できるだけ簡潔な基本プログラムを書いておくことが，その後の枝刈りの工夫を組み込む際にその正しさ，試行作業の効率化を助けるはずです．

ということで，問題1はバックトラッキングの基本プログラムを簡潔に書いてもらおうというものでした．そして出題の図27程度の小さな問題でも，単純な方法ではかなりの計算量が必要であることを認識してもらい，問題2への心構えを期待したのでした．

以下の説明で用いるプログラムはModula-2/86で書かれています．

バックトラッキングの基本プログラム

一般的なバックトラッキングのアルゴリズムは，プログラミングの教科書にもよくでてくるように，再帰呼出しを使って，**プログラム42**のように簡潔に記述することができます．手続きSolve(P)は局面Pに対して解をすべて求めるためのものです．したがって，メインプログラムではSolve(初期局面)とすること

```
procedure Solve(P:position);   (*positionは局面を表わすデータ型*)
 var M: move;                  (*moveは試みを表わすデータ型*)
 begin
  if goal(P) then found        (*1つの解が見つかった*)
  else          (*goal()は解か否かの判定手続き，foundはその処理手続き*)
   for each possible move M for P do
    Solve(moved(P,M))   (*moved(P, M)は局面Pに試みMを施した新しい局面*)
 end;
```

プログラム 42

ですべての解が見つけられます.

この基本型をもとに問題に応じてプログラムを組み立てていけばよいのですが, その際,

・局面, 試みのためのデータ構造(position, move)を問題に適したものとし,

・P が解局面かどうかの判定方法(関数 goal())

・局面 P で可能な試み M の集合を求める方法

・局面 P に試み M を適用する方法(手続き moved)

のアルゴリズムが, プログラムの効率, スマートさを左右する重要な要素となります.

この基本型をナンバーリンクに適用したときの 1 つのナイーブなプログラムの例を**プログラム 43** に示します.

ここでは局面を Board という 2 次配列(0 で空点, 1～MaxElm で始点/経路点

```
MODULE NumLink;
CONST MaxSize   =10;      (*盤サイズ  の可能な最大数*)
      MaxElmNum=10;       (*結ぶべき数の可能な最大数*)
TYPE Coordinate     =[0..MaxSize+1];
     BoardType      =ARRAY Coordinate, Coordinate OF INTEGER;
     EndPointsType  =RECORD SX,SY,TX,TY: Coordinate END;
     EndPointsArray=ARRAY [0..MaxElmNum] OF EndPointsType;
VAR MaxElm: INTEGER;              (*問題における結ぶべき数の最大*)
    Board: BoardType;             (*問題を表わす盤*)
    EndPoints: EndPointsArray;(*結ぶべき各数の始点, 終点*)

PROCEDURE Dist(X1,Y1,X2,Y2:INTEGER): INTEGER;
 BEGIN RETURN ABS(X1-X2)+ABS(Y1-Y2) END Dist;

PROCEDURE Solve(Elm:INTEGER; FX,FY:Coordinate);
 PROCEDURE Try(X,Y:Coordinate);
  BEGIN
   IF Board[X,Y]=0 THEN          (* (b) *)
    Board[X,Y]:=Elm; Solve(Elm,X,Y); Board[X,Y]:=0
   END
  END Try;
 BEGIN (*Solve*)                 (* (a) *)
  IF Dist(FX,FY,EndPoints[Elm].TX,EndPoints[Elm].TY)=1 THEN
   IF Elm=MaxElm THEN Found
   ELSE WITH EndPoints[Elm+1] DO Solve(Elm+1,SX,SY) END
   END
  ELSE Try(FX+1,FY); Try(FX,FY+1); Try(FX-1,FY); Try(FX,FY-1)
  END
 END Solve;

BEGIN (*main*)
 InputData; Solve(1,EndPoints[1].SX,EndPoints[1].SY)
END NumLink.
```

プログラム 43

を表わします．なお，その後のプログラムでの処理の都合上，終点は本来の数＋100で表わしています)と，EndPointsという始点終点を記録する配列とで表わしています．この種のプログラムでの常套手段として，Boardには本来の大きさの外側に，枠外を示すデータを含む領域をもたせています．(これはいわゆる番兵アルゴリズムにも通じる技法で，わずかな一定領域を用意することで効率とプログラムの簡潔さを得るものです．)

このナイーブプログラムを実行してみると，図27の問題に対して2分17秒でした．この時間は，解をすべて(ここでは1つ)見つけこれ以上の解がないことを知るまでのものです．(PC-9801 VM 2の下での実行です．以下の実験結果も同じです．)

図28程度の規模の問題の複雑さが，図27程度の規模の問題の複雑さの何倍あるかは予測できませんが，36から100に大きく増えているマスの数に関して指数関数的に増加していくであろうことから，きわめて大きな倍率になると想像できます．図28程度の問題をナイーブプログラムで解こうとすれば，超高速のスーパーコンピュータといえども長い時間がかかりそうです．

ナイーブプログラム（プログラム43）の動きをトレースしてみると，非常に無駄なことをしていることがすぐにわかります．ある数を閉じ込めた状態で延々と探索を続ける，遠回りをしている，などです．このような“無駄”な状態になったときに，それ以上探索しないようにするのが枝刈りです．

ここでの枝刈りは大きく2つの種類に分けられます．

（1） これ以上探索を続けても解がないことが明らかとなった．

（2） その状態から解が生成できるかもしれないが，その場合はその解と同等あるいはよりスマートな解が1つ以上あって，それらの別解の少なくとも1つは探索される．

すべての解を列挙しなければならないとすれば，(2)の枝刈りを行なうことはできません．しかし，解がユニークであることが問題として要求されるナンバーリンクというパズルの性格上，種々の別解はよりスマートな解で代表させることにしてもかまわないでしょう．

(1)のタイプの枝刈りで最初に気づくのは，連結できない数を含む局面に達した場合でょしう．これは未連結の各数について，始点から空点をたどって終点に達するか否かにより判定できます．**プログラム44**(a)に局面Boardにおけ

る数nのための判定手続きIsConnect(n, Board)を示します*.

連結できない数がある場合の枝刈りは，手続きSolve本体の先頭（プログラム43の(a)部分）に

```
FOR I:=Elm+1 TO MaxElm DO
  IF NOT IsConnect(I, Board) THEN RETURN END
END;
```

を挿入することで行ないます（枝刈り(a)）.

(2)のタイプの枝刈りでは，団子形を作る道，ループを作る道のように，遠回りしている経路であることが判明したときにそれ以上の探索をしないとするのが最も容易な判定でしょう．これは，始点からたどってきてこれから調べようとしている空点の周囲の4点に自身の数がちょうど1個か否かで判定できます(終点を別の数にしているのはこのためです)．プログラム44(b)に示す手続きAdjacentを用意し，プログラム43(b)のIF文の条件式を

```
PROCEDURE IsConnect(Elm:INTEGER; Bd:BoardType): BOOLEAN;
 PROCEDURE Go(X,Y:Coordinate): BOOLEAN;
  BEGIN
   IF Bd[X,Y]=0 THEN
    Bd[X,Y]:=Elm;
    RETURN Go(X+1,Y) OR Go(X,Y+1) OR Go(X-1,Y) OR Go(X,Y-1)
   ELSE RETURN Bd[X,Y]=Elm+100
   END
  END Go;
 BEGIN (*IsConnext*)
  WITH EndPoints[Elm] DO
   RETURN Go(SX+1,SY) OR Go(SX,SY+1) OR Go(SX-1,SY) OR Go(SX,SY-1)
  END
 END IsConnect;
```

（a） ある局面で連続可能かどうかの検査

```
PROCEDURE Adjacent(X,Y:Coordinate; Elm:INTEGER): CARDINAL;
 BEGIN
  RETURN ORD(Board[X+1,Y]=Elm)+ORD(Board[X,Y+1]=Elm)
        +ORD(Board[X-1,Y]=Elm)+ORD(Board[X,Y-1]=Elm)
 END Adjacent;
```

（b） 指定した数の入っている隣接マスの数を求める

プログラム 44 いくつかの枝刈りの工夫

---

* Modula-2の論理式は

P AND Q == IF P THEN Q ELSE FALSE

P OR Q == IF P THEN TRUE ELSE Q

の形で評価されます.

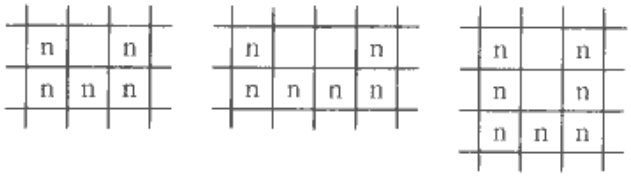

図 80 遠回りとなっている他の例

(Board[X,Y]=0) AND (Adjacent(X,Y,Elm)=1)

とします（枝刈り(b)）.

この2つの比較的単純な枝刈りでもかなりの“無駄”な探索を省くことができます．図27の問題に対しては1.4秒で解けるようになります.

しかし図28規模の問題に対してはまだまだ力不足で，後述のプログラムとの動作比較から推測すると，なお25時間強の実行が必要のようです．これは妥当な時間内とはいえないでしょう.

もう少し枝刈りの方法を検討しましょう.

枝刈り(b)は明らかな遠回りを除外していたのですが，図80のような経路も遠回りになっています．経路で囲まれた袋小路は他の数の経路にはなり得ないので，短絡させても解が消滅することはありません．これを枝刈り(c)とします．ただし，実際のプログラムでは効率よく実現できる条件に限られるでしょう.

さらに，探索の範囲を狭める有力な方法として，

(3) 事前に決定できる要素はできるだけ早く求めておく

があります．この方法に関連してナンバーリンクでは，「ある数nを結ぶすべての経路がマスpを通るならば，マスpには数nが入る」という事実が使えます．たとえば，図27の左下の3のすぐ上のマスは3以外にはありえないので，あらかじめ3とできます．もちろん初期局面だけでなく，途中の局面でもこの操作を行なうことで，探索範囲の減少，解不在の早期発見につながります.

実際のプログラムで「すべての経路の共通マス」を求めるのは非現実的ですので，「数nが連結可能である局面において，マスpに他の数を置くと連結不可能となる」か否かで判定します.

プログラム45のプログラムは，(a)，(b)の枝刈りのほかに，1つの空点を囲むような遠回りに対する枝刈り(c)と，(3)の事前決定を組み込んだものです．Uniqueにより事前に決定したマス中の数は負数で表わしてバックトラッキ

ングで定めてきた数と区別しています.

このプログラムを図28の問題に対して実行させると4時間22分でした. なお, 図28の問題は「ニコリ」では説明用の例題として使われていたもので, 人間には比較的やさしい部類に属するのではないかと思われますが, しらみつぶしの計算機には苦手のほうに入るのかもしれません. 実際,「ニコリ」(No.20)にある6つの問題をプログラム45で解かせたところ, 最短33分, 最長393分, 平均125分というところでした.

枝刈りの効果とオーバーヘッドとの兼合いですが, 有力な方法である(3)はここでは"重い"手続きです. そこでプログラム45では, ある数が結ばれたときにのみ実行しています. また, (a)の判定もこのアルゴリズムでは重く, 2回の試行に1回の割で行なうようにしたほうが実は3割以上高速になります.

おわりに

出題文に「できれば人間が使っているようなヒューリスティクスを取り入れて」と書きました. 出題者が自分で解くときのやり方を分析(?)してみると, まずは(3)のようにして一意に決まるマスを埋めます. 次いで, 辺上にある数に主に注目しつつ, (マスのことはあまり考えずに)各数の位置関係だけを考慮してすべてを結ぶ方法を求め, その後経路がマスにおさまるかどうかを調べているようです. この点から考えると, プログラム45でヒューリスティクスらしい点は(3)の手法を取り入れていることぐらいでしょうか.

マスのことは考えないでとにかく結べるかどうかを調べるという手法の近似として, 出題時には実は10×10の問題を5×5に縮退して概略解を見つけ, その後概略解に基づいて詳細な解を見つけるという方法のアイディアだけはもっていたのです. しかし, いざプログラムを作ろうとすると, ナノピコ教室にはふさわしくない場合分けの多いプログラムとならざるをえない, ということで日の目を見ていません. この種の方法を用いた効率的で簡潔なプログラムは未解決ということにしておきましょう.

最後になりましたが, このパズルは, ICやプリント基板上の配線問題と密接な関係があります. 2点間を交差しないように結ぶことはまさに1平面上の配線問題です. 現実には2層の配線層が用いられることが多く, また, パズルのように解がユニークということはまずないので, 経路長和最短であるとか, 折れ

```
PROCEDURE Solve(Elm:INTEGER; FX,FY:Coordinate);
 VAR UniqNum,I: CARDINAL;
     UniqRec: ARRAY [1..MaxSize*MaxSize] OF RECORD UX,UY: Coordinate END;

 PROCEDURE Unique(): BOOLEAN;
  VAR E: INTEGER; X,Y: Coordinate; Exist: BOOLEAN;
  BEGIN
   FOR Y:=1 TO Size DO FOR X:=1 TO Size DO
    IF (Board[X,Y]=0) AND (Adjacent(X,Y,0)<=2) THEN
     Exist:=FALSE; Board[X,Y]:=-999;
     FOR E:=Elm+1 TO MaxElm DO
      IF NOT IsConnect2(E,Board) THEN
       IF Exist THEN Board[X,Y]:=0; RETURN FALSE END;
       Exist:=TRUE; Board[X,Y]:=-E;
       INC(UniqNum); WITH UniqRec[UniqNum] DO UX:=X; UY:=Y END;
     END END;
     IF NOT Exist THEN Board[X,Y]:=0 END
    END (*IF*)
   END END; (*FOR*)
   RETURN TRUE
  END Unique;

 PROCEDURE Try(X,Y:Coordinate);
  PROCEDURE Check(X,Y:Coordinate): BOOLEAN;
   BEGIN RETURN (Board[X,Y]<>0) OR (Adjacent2(X,Y,Elm)<=1) END Check;
  BEGIN (*Try*)
   IF Board[X,Y]=0 THEN
    IF (Adjacent(X,Y,Elm)=1) AND Check(X+1,Y) AND Check(X,Y+1)
                             AND Check(X-1,Y) AND Check(X,Y-1) THEN
     Board[X,Y]:=Elm; Solve(Elm,X,Y); Board[X,Y]:=0
    END
   ELSIF (Board[X,Y]=-Elm) AND (Adjacent(X,Y,Elm)=1) THEN
    Board[X,Y]:=Elm; Solve(Elm,X,Y); Board[X,Y]:=-Elm
   END
  END Try;

 BEGIN (*Solve*)
  FOR I:=Elm+1 TO MaxElm DO IF NOT IsConnect2(I,Board) THEN RETURN END END;
  IF Dist(FX,FY,EndPoints[Elm].TX,EndPoints[Elm].TY)=1 THEN
   IF Elm=MaxElm THEN Found2
   ELSE
    UniqNum:=0;
    IF Unique() THEN WITH EndPoints[Elm+1] DO Solve(Elm+1,SX,SY) END END;
    FOR I:=1 TO UniqNum DO WITH UniqRec[I] DO Board[UX,UY]:=0 END END
   END
  ELSE Try(FX+1,FY); Try(FX,FY+1); Try(FX-1,FY); Try(FX,FY-1)
  END
 END Solve;

BEGIN (*main*)
 InputData;
 WITH EndPoints[0] DO SX:=0; SY:=0; TX:=1; TY:=0; Solve(0,SX,SY) END
END NumLink.
```

プログラム 45 図27程度の規模の問題を“妥当な”時間に解くプログラム（プログラムヘッダ，定数定義，型定義，変数宣言は，プログラム43に同じ．手続き Dist, Adjacent は，プログラム43，44に同じ．手続 IsConnect 2 は -Elm のマスも使ってたどることを除き IsConnect に同じ．手続き Adjacent 2 は，-Elm のマスも数えることを除き Adjacent に同じ．）

数最小とかの優劣度による最適配線を求めることになります．現実社会での利用価値は絶大ですので多くのアルゴリズム研究がなされているとは思われますが，残念ながら出題者はCADには素人であり，このパズルに適用できるような研究成果アルゴリズムを知りません．しかし，CADでも(問題の規模が違うとはいえ)やはり計算量が膨大なため，現実のシステムでは人間との対話型環境による準最適解(最適であることは保証されないが，一定以上の基準を満たしている解)を求める方法をとることが多いようです．

第5部　パズル

拾い碁

この問題から一筆書きを連想された人もいらっしゃるでしょうがむしろ，石取りゲームの一種です．

もっと問題を発展させ，「向きを変える回数の一番少ないものを求める」とか，「取り始めてから取り終わるまでの道のりの最小のものを求める」とかを考えることもでき，そうなると何かおもしろい理論もでてくるかもしれません．

が，とにかく答を出すことを考えてみましょう．だから，…理論というほどのものは，ありません．とにかく，すべての取り方を試してみることになります．

この問題は「端から取る」という“端”がはっきりしません．パズルやクイズを解くプログラムを作ろうとすると，こんな問題によく出くわします．なんでもない表現なのにきちんと定義しようとするとわけがわからなくなってしまうのです．