人　工　知　能　99−17
（１９９５．３．７）

イラストロジックパズルを解くアルゴリズム

菊池 浩明

東海大学工学部

本稿では，連続する１の要素の数を指定した二種類の数列に基づいて，$n \times m$ の行列を０と１で埋めていく「イラストロジックパズル」を考察している．
イラストロジックパズルのいくつかの性質が明らかにされた後，任意のパズルを解くいくつかのアルゴリズムが提案されている．また，アルゴリズムの重要な性質である完全性と健全性，すなわち，いかなるパズルもそのアルゴリズムで解くことが出来，かつ，アルゴリズムのいかなる出力も正しい解になっていることが証明されている．最後に，アルゴリズムの理論的な計算量の評価と，計算機上での実際の実行結果が示されている．

Algorithms of solving the illust-logic puzzles

Hiroaki Kikuchi

Tokai university, Faculty of Engineering
1117 Kitakaname, Hiratsuka, 259-12 Japan

This paper studies the "illust-logic puzzle," which is of a $n \times m$ matrix to be filled with 1 and 0 based on two sequences specifying run-lengths of 1-elements.
After some of properties of the illust-logic puzzle are clarified, some algorithms to solve any given puzzle are proposed. Also, the completeness and the soundness, properties of algorithms that provide an assurance that every puzzle can be solved and every answer is correct in the algorithm, are proved. Finally, the computational time used by algorithm is estimated, and an actual execution time on a computer are shown.

1 はじめに

イラストロジックは，$n \times m$ のマス目に描かれた図柄を，縦と横について連続して塗りつぶすマス目の数から，同定していくパズルである．「ののぐらむ」「お絵かきロジック」「イラストロジック」などと呼ばれ，最近人気を集めている．1988年ごろ，パズル作家のいしだのん氏と西尾徹也氏がそれぞれ独自に考案したといわれる。いしだ氏は，新宿の高層ビルの窓明かりで絵を描くコンクールをきっかけにパズルとして完成し，1990年から英国の日曜紙「サンデー・テレグラフ」，および翌年から毎日新聞で連載を始めた[1].

パズルの例[2]を図1に示す．基本的なルールは次の通りである．

1. パズルの縦と横について，連続して塗りつぶすマスの数の列が与えられる(図の，左と上の数列)．ただし，空いているマス目の数はわからない．
2. 塗りつぶすマス目とマス目の間には，必ず一つ以上空きマスがなくてはならない．
3. 行と列の両端は，空きマスより始めてもよい．

ルール1で定められているように，空いているマス目の数がわからないため，縦と横の数列のどちらか一つだけではイラストは定まらない．しかし，9,10行のように，マス目の数と塗りつぶす数が同じ場合は，その行は一意に決まる．また，ルール2により，8行目も一通りの塗り方しかないため，一意に決まる．すると，d列も，10行目が既に塗られているため，1行目を除いたのこりが全て塗りつぶされる．こうして，縦と横の数列から，全てのマス目を決定していく．塗りつぶしの過程は，勘にたよらず(少しはたよる？)，論理的に進んでいくため，イラスト「ロジック」と呼ばれているものと思われる．

		a	b	c	d	e	f	g	h	i	j
			2								
		3	1	1		1	2	3			
		3	2	5	9	5	4	3	2	3	4
1	2 2										
2	2 1 2										
3	1 1 1										
4	1 1 1										
5	1										
6	3										
7	5 1										
8	1 5 2										
9	10										
10	10										

図 1: パズル1

この例の答を図2に示す．潮をふいているクジラのイラストが描かれている．

イラストロジックを解く問題は，不完全なランレングス符号から情報を同定する問題と解釈できる．しかし，たとえ計算機で総当たりをするにも，場合の数が膨大であり，そのままでは解けない．ヒューリスティックな解法や，遺伝アルゴリズムなどの適用も考えられるが，大きなパズルとなると，実時間で処理することは困難であろう．

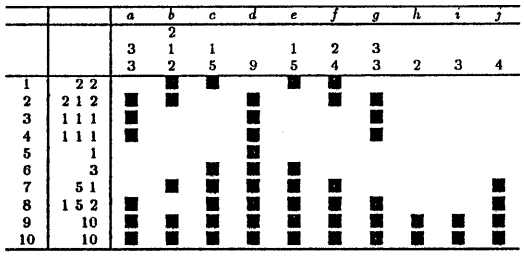


図 2: パズル1の解

そこで，本稿では，可能性を考慮することで，効率的にイラストロジックを解くいくつかのアルゴリズムを提案する．また，それらの処理速度などについて評価を行なう．

2 準備

イラストロジックパズルを定式化し，いくつかの性質を示す．

パズルとは，各要素が0(空きマス)，1(黒マス)，$\emptyset$(未定)の三つの値をとる $n \times m$ の行列

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

と，縦と横について各々連続する1の数を示した数列 X, Y

$$X = \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix} = \begin{pmatrix} x_{11}, \ldots, x_{1k_1} \\ \vdots \\ x_{m1}, \ldots, x_{mk_m} \end{pmatrix},$$

$$Y = (y_1, \ldots, y_n) = \begin{pmatrix} y_{11} & & y_{n1} \\ \vdots & \cdots & \vdots \\ y_{1k_1} & & y_{nk_n} \end{pmatrix}$$

から成っている．行と列に対する操作は本質的に変わらないので，行あるいは列の一つを語と呼び，$a = (a_1, \ldots, a_n)$, $a_i \in \{0, 1, \emptyset\}$ と表す．ここで，n を語の長さと呼ぶ．語 a の符号化とは，その行(列)の対応する数列 $x(y)$ であり，これを，$f(a) = b = (b_1, \ldots, b_k)$ と表す．符号化 $f(a)$ の各要素は，自然数を取り，その要素の数 k を符号化の長さと呼ぶ．

符号化は，写像 $f : \{0, 1, \emptyset\} \to I^k$ と表すことができるが，単射ではない．従って，一つの符号化に対して，対応する語は一般に複数存在する．例えば，$f(a) = (3, 1, 1)$ で，$n = 8$ の場合，語 a は (11101010), (11101001), (11100101), (01110101) の4通り存在する．そこで，f の逆関数 $f^{-1}(b)$ を，可能な語の集合として定める．すなわち，$f^{-1} : I^k \to \{0, 1\}^{\{0,1\}^n}$ である．

符号化 b の逆関数の要素の数 $|f^{-1}(b)|$ をその符号化の 自由度と呼ぶ．先の例では，自由度は 4 となる．全ての行 (列) についての自由度の数の積が，そのパズル A の $X(Y)$ を満たす組合せの数を表している．この値は，たとえ 20×20 程度のパズルでも 10 の数十乗にも達し，総当たりが不可能であることを裏付ける．

定理 2.1 長さ n の語の符号化を $b=(b_1,\ldots,b_k)$ とする．この符号化の自由度は，次式で与えられる．

$$|f^{-1}(b)| = \left(\begin{array}{c} n-b^*+1 \\ k \end{array} \right) = \frac{(n-b^*+1)!}{(n-b^*+1-k)!k!}$$

ここで，$b^* = \sum_{i=1}^{k} b_i$ である．

証明 $n-b^*$ は連続する長さが不明な空きマスの総数であり，これを k 個に分割する数が自由度となる．従って，これに先頭の一マスの 1 を足して，k 個の仕切りを入れる組合せの数を求めて，定理が得られる．(証明終)

先の例をもう一度考えよう．$n=8, b^*=3+1+1=5$, $k=3$ である．この自由度は，

$$|f^{-1}(3,1,1)| = \left(\begin{array}{c} 8-5+1 \\ 3 \end{array} \right) = \frac{4\cdot 3\cdot 2}{3\cdot 2\cdot 1} = 4$$

と定まり，前述の場合の数に一致している．

ここで提案するアルゴリズムの基本原理となる定理を示す．

定理 2.2 長さ n の語 $a=(a_1,\ldots,a_n)$ の符号化を $b=f(a)$ とする．逆関数 $f^{-1}(b)$ の要素の積集合が

$$c=(c_1,\ldots,c_n) = \bigcap_{a'\in f^{-1}(b)} a'$$

である時，次が成立する．

$$a_i = \begin{cases} 1 & \text{if} \quad c_i=\{1\} \\ 0 & \text{if} \quad c_i=\{0\} \end{cases}$$

証明 $c_i=\{1\}$ であることは，そのマス目に 0 が来る可能性がないこと，すなわち，必然的に 1 であることを示している．よって，$a_i=1$. $c_i=\{0\}$ の場合も同様である．(証明終)

例として，$n=5, f(a)=(3), k=1$ のパズルを考えよう．この時，

$$\begin{aligned} f^{-1}(3) &= \left\{ \begin{array}{l} (11100), \\ (01110), \\ (00111) \end{array} \right\} \\ c &= (\{0,1\},\{0,1\},\{1\},\{0,1\},\{0,1\}) \end{aligned}$$

となり，定理より，$a_3=1$ が言える．

次に部分的に語の要素が既知である場合を考える．これは，パズルを解いていく過程で生じる問題である．語 a の一部が既知である時，それを制約とみなし，対応する符号化 b の逆関数 $f^{-1}(b)$ を，a を満たす語の集合として定める．例えば，a_2 が既知である語 $a=(*1**)$ の符号化 $b=(3)$ の逆関数は，$f^{-1}(b)=\{(11100),(01110)\}$ となる．この様に定めると，部分的に既知の語についても，明らかに，定理 2.2が成立する．

3 アルゴリズム

提案するアルゴリズムの基本戦略は，定理 2.2を縦，横，縦，横と逐次的に適用していく というものである．

アルゴリズム cube2

input $n\times m$ のパズル A, およびその符号化 X, Y.

step 1 各 i 行について，続く 2, 3, 4 を繰り返す．

step 2 各 j 列について，続く 3, 4 を繰り返す．

step 3 A の i 行目 a_i の j 列目を $a_{ij}=1$ と置き，$f(a_i)=X_i$ を満たす $a_{i1},\ldots,a_{im}$ の 組が存在するか確かめる．同様の操作を，$a_{ij}=0$ にも適用する．

step 4 $a_{ij}=1$ が可能で，$a_{ij}=0$ が不可能ならば，$a'_{ij}=1$ とする．逆に，$a_{ij}=0$ が可能で，$a_{ij}=1$ が不能ならば，$a'_{ij}=0$ とする．

step 5 各 i 列について，続く 6, 7, 8 を繰り返す．

step 6 各 j 行について，続く 7, 8 を繰り返す．

step 7 A の i 列目 a_i の j 行目を $a_{ji}=1$ と置き，$f(a_i)=Y_i$ を満たす $a_{1i},\ldots,a_{ni}$ の 組が存在するか確かめる．同様の操作を，$a_{ji}=0$ にも適用する．

step 8 $a_{ji}=1$ が可能で，$a_{ji}=0$ が不可能ならば，$a'_{ji}=1$ とする．逆に，$a_{ji}=0$ が可能で，$a'_{ji}=1$ が不能ならば，$a_{ji}=0$ とする．

step 9 $A^{k+1}=\left(\begin{array}{c} a'_{11},\ldots,a'_{1m} \\ a'_{n1},\ldots,a'_{nm} \end{array} \right)$ と置き，$A^{k+1}=A^k$ ならば (4 と 8 のどちらかで更新がなければ) 終了．さもなければ，1 へ戻る．

アルゴリズム cube2 では，step 3 と 7 で，仮定した値が符号化 $X(Y)$ を満たすことができるかどうかのチェックを行なう．従って，その様なパターンがあれば処理は早いのだが，存在しない場合は，結局，全ての場合，すなわち，f^{-1} の全ての要素を求めることとなる．しかも，処理はマス目ワイズで行なわれるため，同一の行でも同じ語を作り出して検査することがあり，極めて効率が悪い．そこで，行と列について予め全ての可能な語を書きだすことにより，探索の効率化を図るアルゴリズムを考える．

アルゴリズム cube3

step 1 各 i 行について，続く 2, 3 を繰り返す．

step 2 $f^{-1}(x_i) = f^{-1}(x_{i1},\ldots,x_{ik_i})$ を求め，$c = \bigcap_{a\in f^{-1}(x_i)} a$ と置く．

step 3 i 行の各 j 列について，$c_j = \{1\}$ ならば，$a'_{ij} = 1$，$c_j = \{0\}$ ならば，$a'_{ij} = 0$ とする．

step 4 各 i 列について，続く 5, 6 を繰り返す．

step 5 $f^{-1}(y_i) = f^{-1}(y_{i1},\ldots,y_{ik_i})$ を求め，$c = \bigcap_{a\in f^{-1}(y_i)} a$ と置く．

step 6 i 列の各 j 行について，$c_j = \{1\}$ ならば，$a'_{ji} = 1$，$c_j = \{0\}$ ならば，$a'_{ji} = 0$ とする．

step 7 $A^{k+1} = \begin{pmatrix} a'_{11},\ldots,a'_{1m} \\ a'_{n1},\ldots,a'_{nm} \end{pmatrix}$ と置き，$A^{k+1} = A^k$ ならば (3 と 6 のどちらかで更新がなければ) 終了．さもなければ，1 へ戻る．

アルゴリズム cube3 は，cube2 に比べて格段に早い．しかし，それでも専門誌などに掲載されている難度の高い 問題は解くことが出来ない．f^{-1} を求める計算量が，問題の大きさに対して指数関数的に大きくなるからである．では，人はどのようにしてこのパズルを解いているのだろうか？観察してみると，どうやら，真面目に端から順番通り解いていくのではなく，場合の数の少なそうな行や列を見つけて，そこから同定してパズルを解いているようである．そこで，f^{-1} を計算する前に，その処理にかかる処理時間を見積り，ある敷居値を越えたものはその回は見送る枝刈り処理を加える．敷居値を越えた行や列は，たとえ処理を行なったとしても，自由度が多き過ぎてほとんど同定が進まないことが予想される．妥当な枝刈りであるといえる．ここで，自由度は定理2.1の式により求められる．

アルゴリズム cube4

step 2, step 5 を除いて cube3 に同じ．変更点だけを示す．1 と 2 との間に 1.5 を，4 と 5 の間に 4.5 を追加する．

step 1.5 自由度 $|f^{-1}(x_i)|$ を求め，その値が敷居値 θ 以内であれば 2 へ，そうでなければ，次の $i+1$ 行に移る．

step 4.5 自由度 $|f^{-1}(y_i)|$ を求め，その値が敷居値 θ 以内であれば 4 へ，そうでなければ，次の $i+1$ 列に移る．

4 評価

アルゴリズムを，完全性，計算量，実行時間の観点から評価する．

4.1 完全性

ここでの興味は，提案アルゴリズムがどんなパズルでも解くことができるか，という点にある．従って，処理時間は重要でなく十分な処理時間を仮定する．

まず，パズル $<A,X,Y>$ が解を持つ，すなわち，符号化 X,Y を満たすような A が一つ以上存在することを，そのパズルは 可解である，という．更に，解が一意に決まることを，パズルは一意であるという．一般に，正常なパズルは可解で一意でなくてはならない．

次に，あるアルゴリズムの出力が，必ずパズルの解の一つになっていることを，アルゴリズムが健全であるという．そして，ある可解なパズルがあれば，必ずそのアルゴリズムで解けることを，アルゴリズムが完全であるという．

さて，実は前節で提案したアルゴリズムは全く同一の結果を出力する 本質的に同一のものであり，これをアルゴリズム cube と呼ぶ．このアルゴリズムについて，すぐに次が言える．

定理 4.1 ある可解で一意なパズル A,X,Y について，アルゴリズム cube は健全であり，かつ，完全である，すなわち，cube の出力は必ず X,Y を満たし，如何なるパズルも cube で解くことが出来る．

証明 健全性は，定理2.2から明らか．完全性を示す．パズル A,X,Y について，アルゴリズムが停止し，しかも，出力 A^k に $a_{ij} = \emptyset$ となる要素が 残っていると仮定する．このような要素 a_{ij} は A の中で一つだけとなることはない．なぜならば，アルゴリズムが停止したということは，$a_{ij} = 1$ と 0 の両方を取る可能性があることに等しく，これは，符号化 X と Y の両方を変えるため，パズルが可解であることに矛盾する．同様にして，$\emptyset$ は二つでも足りないことがわかる．結局，少なくとも，行と列を共通にする 4 箇所 a_{ij}, a_{kj}, a_{il}, a_{kl} が必要である．しかし，これら 4 つは，

$$\begin{pmatrix} a_{ij} & a_{il} \\ a_{kj} & a_{kl} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

この二種類の値を取ることとなり，結局，パズルが一意であることと矛盾する．(証明終)

4.2 計算量

アルゴリズム cube3 を代表として，パズルの大きさに関する計算量を求めよう．まず，$n\times m$ の A, n 行の符号化 X, m 列の符号化 Y から成るパズルを考える．step 2 で必要な語の数は，

$$\phi_i = |f^{-1}(x_i)| = \frac{(m-x_i^*+1)!}{(m-x_i^*+1-k_i)!(k_i)!},$$

ただし，k_i は i 行目の符号化の長さ，$x_i^* = x_{i1}+\cdots+x_{ik_i}$ とする．更に，各行についてこの処理が行なわれ

るため，step 1-3 全体では，$\sum_{i=1}^{n}\phi_i$ 個の語を求める計算となる．列についても同様に，列 i の自由度を ψ_i と置いて表すと，step 4-7 にかけて $\sum_{i=1}^{m}\psi_i$ 個の語が必要となる．更に，以上の処理を収束するまで繰り返すため，繰り返し回数を定数 c で表すと，アルゴリズム全体で必要とする語の総数は，

$$(\sum_{i=1}^{m}\phi_i+\sum_{i=1}^{n}\psi_i)c$$

と表される．簡単化のため，$n=m$, $\phi_i=\psi_i$ と置くと，アルゴリズムのオーダーは $O(n!)$ である．なお，総当たりの場合は，次に示す数となる．

$$\prod_{i=1}^{n}\phi_i=\left(\frac{n!}{(n-k)!k!}\right)^n$$

4.3 実行時間

Sun Sparcstation 2(30MIPS) での，各種処理時間を示す．まず，行単位での自由度 ϕ_i と実際にそこでかかる処理時間を図 3に示す．サンプルは，40×45 の大きさのパズルを用いた．これより，自由度と処理時間がほぼ比例関係にあることがわかる．従って，cube4 の枝刈りの信頼性も得られる．

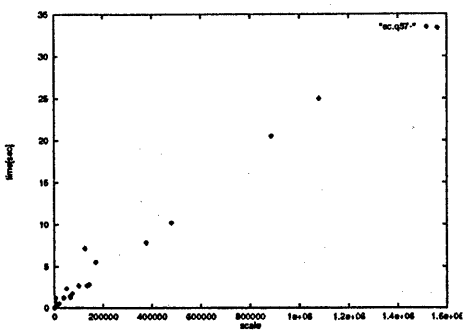


図 3: 自由度と処理時間の関係

パズルの大きさ $n\times m$ と，パズルの難易度 (total time)，すなわち，$\prod_{i=1}^{n}\phi_i$ との関係を図 4に示す．y 軸が対数スケールになっており，大きさが難易度に対して指数関数的に増加することを表している．

各種アルゴリズムの処理時間を図 5に示す．実装に問題があるのか，cube3 にまれに処理時間がかかる例が生じている．ここで示されているように，50 × 50 の大きさのパズルまで 現実的な処理時間で解けることが確認された．また，実際の実行の様子を付録に示している．行と列の検査により，パズルが同定されていく様子がよく現れている．

5 おわりに

イラストロジックパズルについて考察し，いくつかの性質を明らかにした．パズルを解く二三のアルゴリズムを提案し，理論的な計算量と 実際の実行時間を示した．提案アルゴリズムの完全性，すなわち，任意の健全で一意なパズルは 本アルゴリズムで解くことが出来ることを証明した．

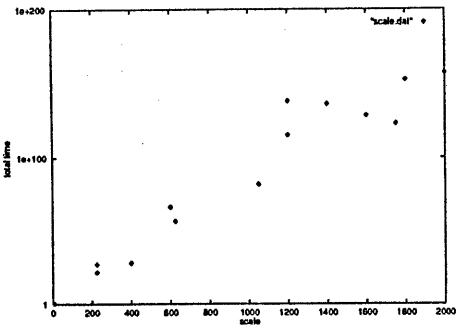


図 4: 大きさと難易度

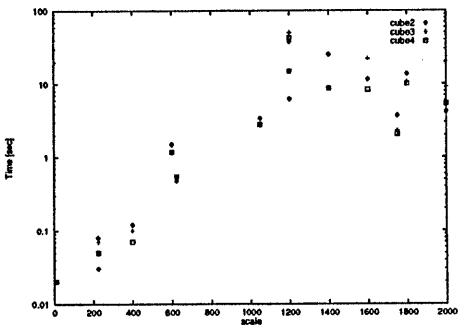


図 5: 実行時間

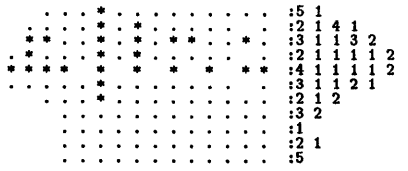
参考文献

[1] “経済特集「パズル景気 専門誌・単行本が好調」”, 朝日新聞 11 月 26 日（夕刊），pp.10, 1994

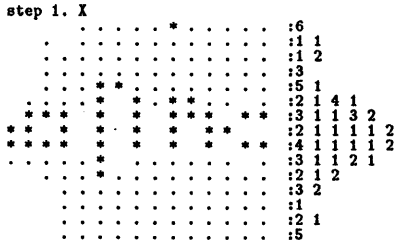
[2] “イラストロジックのルールと解き方”，イラストロジックパラダイス, vol.7, pp. 90, 1995

A. 実行結果

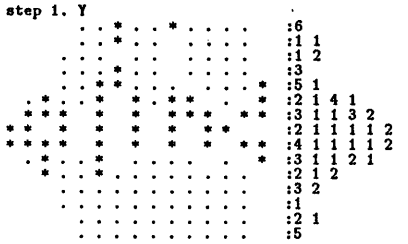
```
step 0. X
. . . . . . . . . . . . . . . . . . . . :6
. . . . . . . . . . . . . . . . . . . . :1 1
. . . . . . . . . . . . . . . . . . . . :1 2
. . . . . . . . . . . . . . . . . . . . :3
. . . . . . . . . . . . . . . . . . . . :5 1
. . . . . . . . . . . . . . . . . . . . :2 1 4 1
. * * . . . . . * * . . * . . . . . . . :3 1 1 3 2
. . . . . . . . . . . . . . . . . . . . :2 1 1 1 1 2
* * * *   *   *   *   *   * *           :4 1 1 1 1 2
. . . . . . . . . . . . . . . . . . . . :3 1 1 2 1
. . . . . . . . . . . . . . . . . . . . :2 1 2
. . . . . . . . . . . . . . . . . . . . :3 2
. . . . . . . . . . . . . . . . . . . . :1
. . . . . . . . . . . . . . . . . . . . :2 1
. . . . . . . . . . . . . . . . . . . . :5

step 0. Y
    . . . . . . . . . . . . . . . . . . :6
    . . . . . . . . . . . . . . . . . . :1 1
    . . . . . . . . . . . . . . . . . . :1 2
    . . . . . . . . . . . . . . . . . . :3
```



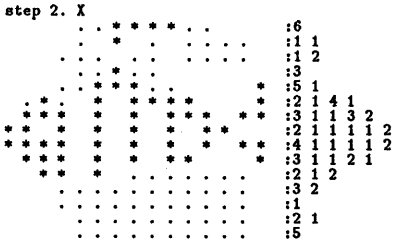
step 1. X



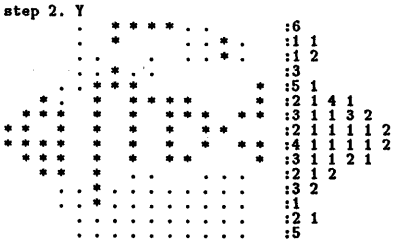
step 1. Y



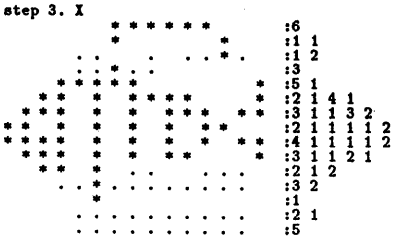
step 2. X



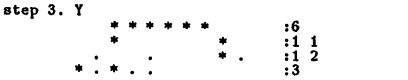
step 2. Y

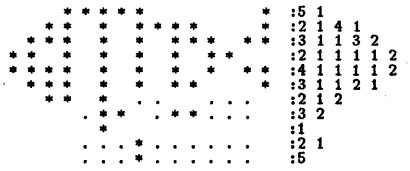


step 3. X

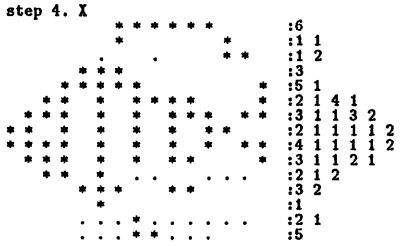


step 3. Y

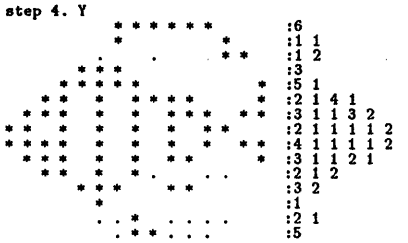




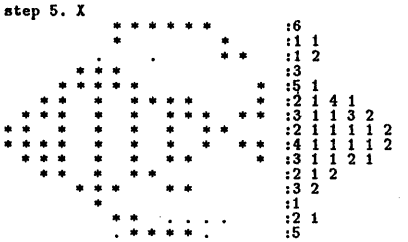
step 4. X



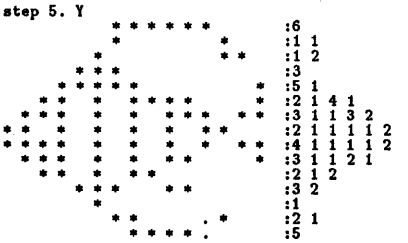
step 4. Y



step 5. X



step 5. Y



step 6. X