Electronic computation of squared rectangles

ELECTRONIC COMPUTATION OF SQUARED RECTANGLES

ELECTRONIC COMPUTATION OF SQUARED RECTANGLES

PROEFSCHRIFT

TER VERKRIJGING VAN DE GRAAD VAN DOCTOR IN DE TECHNISCHE WETENSCHAP AAN DE TECHNISCHE HOGESCHOOL TE EINDHOVEN, OP GEZAG VAN DE RECTOR MAGNIFICUS Dr K. POSTHUMUS, HOOGLERAAR IN DE AFDELING DER SCHEIKUNDIGE TECHNOLOGIE, VOOR EEN COMMISSIE UIT DE SENAAT TE VERDEDIGEN OP VRIJDAG, 29 JUNI 1962, DES NAMIDDAGS TE 4 UUR

DOOR

ADRIANUS JOHANNES WILHELMUS DUIJVESTIJN

ELECTROTECHNISCH INGENIEUR

GEBOREN TE 's-GRAVENHAGE

DIT PROEFSCHRIFT IS GOEDGEKEURD DOOR DE PROMOTOR
PROF. DR C. J. BOUWKAMP

Summary

This thesis considers problems that arise when the calculation of squared rectangles is automatized on an electronic computer. After the introductory chap. 1, it is indicated in chap. 2 how a so-called c-net is coded such that it can be processed by the computer. In particular, properties of the net in connection with its planarity are easy to recognize using this code. It is shown how, from the code of the original net, the code of the dual net can be obtained. Also described is how the branches and the vertex-vertex incidence matrix of the net can be found from the code of the net. In chap. 3, a set of codes Σ_{N+1} representing c-nets of order $N+1$ is generated from the set of codes T_N of all different c-nets of order N, by addition of a wire in the latter c-nets or their duals. The set Σ_{N+1} may contain codes representing one and the same net or its dual. Therefore a method is described how to each net a number can be assigned that characterizes the net uniquely. Sorting with respect to this characteristic number gives the set of codes T_{N+1} of all different c-nets of order $N+1$. Additional information as to whether the net is selfdual or not is provided, and the number of its symmetry axes is calculated. In chaps 4 and 5 all squared rectangles obtainable from one given c-net are computed. It is also determined whether the squared rectangle is perfect or imperfect and, in the latter case, whether the imperfection is trivial or not. Finally, chap. 6 shows a few typical results. In particular, we mention some typical output of the computer PASCAL and a list of squared squares of orders up to and including nineteen.

Résumé

Cette thèse examine les problèmes qui se posent lorsque le calcul des rectangles divisés en carrés est automatisé sur une calculatrice électronique. Après un chapitre 1 en guise d'introduction, le chapitre 2 montre comment un graphe complet est codifié de manière à pouvoir être traité par la calculatrice. Les propriétés du graphe par rapport à la question de savoir s'il est planaire sont particulièrement aisées à identifier en utilisant ce code. Il est montré comment, à partir du code établi pour le graphe original, on peut obtenir le code pour son dual. On y voit aussi comment, à partir du code du graphe, on peut trouver la matrice d'incidence aux sommets et les arrêtes du graphe. Au chapitre 3, un ensemble de codes Σ_{N+1}, représentant des graphes complets d'ordre $N+1$, est issu de l'ensemble des codes T_N de tous les différents graphes complets d'ordre N, en ajoutant une arête dans ces derniers graphes ou leurs duals. L'ensemble Σ_{N+1} peut contenir des codes représentant un seul et même graphe ou son dual. Pour cette raison, une méthode est décrite et montre comment un nombre peut être assigné pour chaque graphe complet, ce nombre caractérisant uniquement le graphe. Un classement effectué en tenant compte de ce nombre caractéristique donne l'ensemble des codes T_{N+1} de tous les différents graphes complets d'ordre $N+1$. Des informations supplémentaires quant à savoir si le graphe et son dual sont identiques ou non, sont fournies et le nombre de ses axes de symétrie y est calculé. Aux chapitres 4 et 5, tous les rectangles divisés en carrés pouvant être obtenus à partir d'un graphe donné sont calculés, de même que leurs codes Bouwkamp tels qu'ils seront imprimés par la calculatrice. Il y est aussi établi si la dissection est parfaite ou non, et dans ce dernier cas, si l'imperfection est triviale ou non. Enfin le chapitre 6 expose quelques résultats caractéristiques. Nous mentionnons particulièrement quelques réponses typiques données par la calculatrice PASCAL, ainsi qu'une liste des carrés divisés en carrés pour des valeurs jusqu'à et y compris dix-neuf.

Zusammenfassung

Diese Arbeit behandelt die bei der elektronischen Rechenmaschine bei der Automatisierung der Berechnung der in Quadrate unterteilten Rechtecke auftretenden Probleme. Nach der Einleitung in Kapitel 1 zeigt Kapitel 2 die Verschlüsselung eines sogenannten c-Netzes für die Ver-

arbeitung in einer Rechenmaschine. Durch diese Verschlüssung lassen sich besonders leicht die Eigenschaften des Netzes, was seine Ebenheit betrifft, erkennen. Dann wird die Gewinnung der Verschlüsselung des Dualnetzes aus der Verschlüssung des Originalnetzes gezeigt. Auch wird beschrieben, wie sich die Knoten-Knoten Inzidenz Matrix und die Zweige des Netzes aus der Verschlüsselung des Netzes finden läßt. In Kapitel 3 wird eine Menge Σ_{N+1} von Verschlüsselungen, die c-Netze der $(N+1)$-ten Ordnung darstellen, aus der Menge T_N der Verschlüsselungen aller verschiedenen c-Netze N-ter Ordnung durch Hinzufügung eines Drahtes in den c-Netzen oder ihren Dualen gewonnen. Die Menge Σ_{N+1} kann Verschlüssungen enthalten, die ein und dasselbe Netz oder sein Dual darstellen. Es wird daher eine Methode angegeben, durch die sich jedem Netz eine Zahl zuordnen läßt, die das Netz eindeutig kennzeichnet. Das Sortieren nach dieser kennzeichnenden Zahl fürht zur Menge T_{N+1} der Verschlüsselungen aller verschiedenen c-Netze $(N+1)$-ter Ordnung. Weiterhin wird angegeben, ob das Netz selbst-dual ist oder nicht. Es wird auch die Zahl seiner Symmetrieachsen berechnet. Im 4. und 5. Kapitel werden alle in Quadrate unterteilten Rechtecke für ein gegebenes c-Netz und ebenso ihre Bouwkamp-Verschlüsselungen, wie sie von der Rechenmaschine gedruckt werden, errechnet. Es wird auch bestimmt, ob das in Quadrate unterteilte Rechteck vollkommen oder unvollkommen ist. Im letzteren Fall wird festgestellt, ob die Unvollkommenheit trivial ist oder nicht. Kapitel 6 zeigt schließlich einige typische Beispiele. Im besonderen werden einige typische Ausgaben der Rechenmaschine PASCAL und eine Liste der in Quadrate unterteilten Quadrate bis zur neunzehnten Ordnung angeführt.

CHAPTER 1

INTRODUCTION

This thesis is concerned with the problem of dissecting a rectangle into a finite number of non-overlapping squares. In particular, we study the problems that arise when one wants to calculate these dissections by an electronic computer.

The terminology of Brooks, Smith, Stone and Tutte [1]) and Bouwkamp [2]) will be used. A dissection of a rectangle into a finite number $N > 1$ of non-overlapping squares is called a *squared rectangle* or a *squaring* of *order N*. The N squares are called the *elements* of the dissection. The term "elements" is also used for the (lengths of the) sides of the elements.

If the elements are all unequal, the squaring is called *perfect* and the rectangle is called a *perfect rectangle*; otherwise the squaring or rectangle is *imperfect*. Examples of perfect and imperfect squarings are given in figs 1 and 2; the numbers inscribed denote the lengths of the sides of the corresponding squares.

A squared rectangle that contains a smaller squared rectangle in its interior is called *compound*. All other squared rectangles are *simple*. Apparently, the squarings given in figs 1 and 2 are simple. An example of a compound squaring is given in fig. 3.

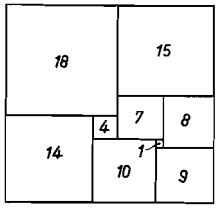


Fig. 1.

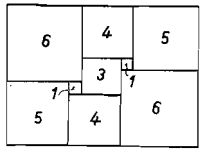


Fig. 2.

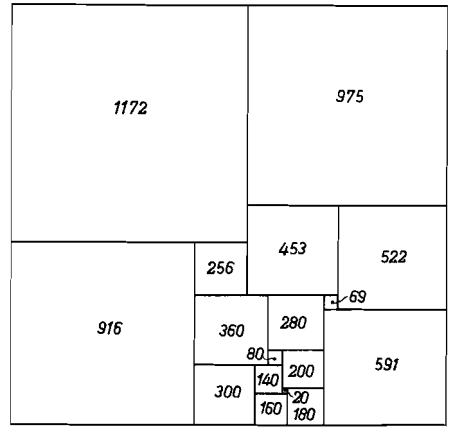


Fig. 3.

Fig. 1. Example of a perfect squaring of order 9.

Fig. 2. Example of an imperfect squaring of order 9.

Fig. 3. Example of a compound perfect squaring of order 17.

The squaring is called *trivially imperfect* if it contains equal elements that touch each other along a common side.

In 1903 Dehn [3]) initiated the study of a somewhat more general problem, namely, that of the (non-trivial) dissection of a rectangle into a finite number of non-overlapping smaller rectangles. He proved the following theorem: If each sub-rectangle has commensurable sides, then so has the original rectangle and, moreover, all the sides of the rectangle and the sub-rectangles are mutually commensurable.

In particular, by taking the sub-rectangles to be squares, Dehn found as a corollary: Any squared rectangle has commensurable sides and elements.

Dehn did not go beyond announcing (and proving) this theorem and its corollary. It remained an open question whether a perfect squared rectangle did exist at all. However, such a squared rectangle was given in 1925 by Moroń [4]), when he gave the example of fig. 1.

Considerable progress was made by Brooks, Smith, Stone and Tutte [1]) in 1940. They succeeded in separating the topological part from the metrical part of the problem. The topological part of the problem appeared to be related to the theory of linear graphs, while the metrical part proved to be connected to the theory of current flow in electrical networks. They also gave a short table of low-order squared rectangles.

The relation of the squared rectangles with electrical networks was also considered by Bouwkamp [2]) who gave a more-physical approach to the problem. In Bouwkamp's paper a table was given of all simple squared rectangles of orders up to and including 13. For that purpose, Bouwkamp introduced a concise and efficient code for the squared rectangle. He supposed the rectangle to be drawn in such a manner that its larger side is horizontal. Further, the element at the upper-left corner should not be smaller than the three remaining corner elements. After this orientation of the rectangle, the upper-left corner of each element is taken as its representative point. The length of the sides of the elements for which the representative points lie in the same horizontal segment are assembled within parentheses in the order from left to right, the elements within parentheses being separated by commas. The parentheses read in order from top to bottom of the rectangle. Collinear horizontal segments are taken in order from left to right. This code will be called the *Bouwkamp code* of the squaring. For example, the codes pertaining to the squarings given in figs 1 and 2 are as follows:

(18,15)(7,8)(14,4)(10,1)(9) and (6,4,5)(3,1)(6)(5,1)(4).

Brooks, Smith, Stone and Tutte [1]) proved that there are no perfect rectangles of order less than 9. The minimum number of elements necessary to divide a square simply is also known [2]). It is a simple imperfect squared square of order 13. Its code reads as follows: (12,11)(1,3,7)(11,2)(5)(2,5)(4,1)(3). Other examples

of simple imperfect squares were found by Bouwkamp, Duijvestijn and Medema [5]). There are none of order 14; those of order 15 are:

(20,8,11)(5,3)(2,12)(7)(19,8)(5,7)(11,2)(9),
(20,19)(1,3,8,7)(19,2)(5)(2,5)(12,1)(3)(8),
(23,18)(7,11)(18,3,2)(1,5,3)(4)(2,1)(12)(11).

Simple squared squares of higher order are given in chap. 6.

Willcocks [3]) has constructed a perfect squared square of order 24, with code as follows:

(81,51,43)(8,35)(30,29)(2,33)(31)(39,14,20,38)(9,5)(4,1)(3,18)(16)(64)(56)(55).

However, this square is compound in that it is built up of one square and two squared rectangles. At present it is not known whether 24 is the minimum number of elements necessary to divide a square perfectly. As to perfect and simple squares, the best result known sofar is also due to Willcocks [7]), who has found a simple perfect square of order 37, with code as follows:

(728,378,406,435)(350,28)(405,29)(464)(648,347,83)(184,206,98)(10,454)(108)
(162,22)(336)(245,102)(20,142)(122)(210,54)(56,189)(250,594)(571,133)(438,94)
(344).

The existing tables of low-order squarings have been useful for the construction of squarings of special type (cf. the 24-order squared square of Willcocks). For that reason, Ellis [8]) started to extend the tables of perfect squarings so as to include those of order 14. These calculations were entirely done by hand, that is, without the use of electronic calculating machinery. It is practically impossible to continue in this way to orders 15 and higher. Further extension can only be carried out with electronic computation.

In trying to solve the problem of generating squared rectangles automatically with an electronic computer, one meets a number of new problems. Especially, how can the computer deal with the topolocigal aspects of the problem?

In the present thesis it will be described how the necessary new networks can be generated automatically. A characteristic of the network will be calculated, by which it can be judged whether two networks are different or the same. Furthermore, it will be described how the Bouwkamp codes of all dissections arising from a given network can be obtained automatically; it is also possible to let the computer indicate whether a squaring is perfect or imperfect, and in the latter case whether the imperfection is trivial or not. The first results have been published by Bouwkamp, Duijvestijn and Medema [5])[9]), where all simple squarings of orders up to and including 15 were given.

In describing the programmes occurring in this paper we closely followed the rules of the ALGOL-60 language [10]). In the ALGOL programmes it is assumed that non-local variables have been introduced previously, unless stated other-

wise. The programmes written in ALGOL were translated into PASCAL (Philips automatic sequence calculator) code. With the aid of these programmes all networks of orders up to and including 19 were generated on PASCAL. Further, for all possible networks of orders up to and including 20, possible squared squares following from these were determined; the Bouwkamp codes of these squared squares were printed by PASCAL.

Some of the results will be given in chap. 6. For example, in contrast with early expectation, there does not exist a simple perfect squared square of order less than 20.

CHAPTER 2

PROBLEMS OF CODING

2.1. Introduction

The relation of squared rectangles to planar electrical networks will now be considered. It was shown in papers already referred to [1)2)] that each element of the squared rectangle corresponds to a wire or branch, while each horizontal line segment corresponds to a vertex, and each vertical line segment to a mesh not containing other parts of the network in its interior. The vertices corresponding to the upper and lower horizontal sides are the poles of the network.

The network constructed in this way is called the *normal polar network* or *normal p-net* [1)]; see the example of fig. 4.

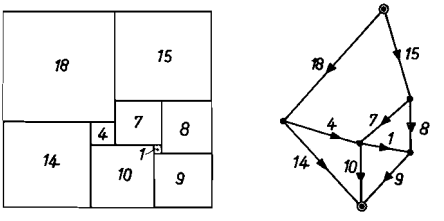


Fig. 4. Example of a squaring and its associated normal p-net; ⊙ = pole of the net.

A planar network (with more than one vertex) that is connected is called a *net*. If two vertices of the net are assigned as poles, and no circuit is enclosing the poles, the net is called a *polar net* or a *p-net*.

A *c-net* is a net that has no parts (consisting of more than one wire, and less than all but one wire) joined to the rest at less than three vertices. Joining the poles of a normal p-net by a wire gives a c-net (c) if the squaring corresponding to the normal p-net is simple.

Now before the normal p-net is constructed, the rectangle can be turned through 90 degrees. Then joining the two poles by a wire again produces a c-net (c'). The net c' is called the *dual* of the net c. Obviously, c is also the dual of c'. Therefore, c and c' form a pair of dual c-nets.

Dual nets can be drawn in such a way that the vertices of either of them lie inside the corresponding meshes of the other, while corresponding branches, and only these, cross each other. Brooks, Smith, Stone and Tutte [1)] proved that the dual of a c-net is a c-net. Apparently, any simple squaring can be obtained from an appropriate c-net.

To illustrate the various concepts, consider the c-net of fig. 5, obtained from the normal polar net of fig. 4. For later purposes this c-net will be called the *reference net*. The reference net and its dual (dashed lines) are drawn in fig. 6.

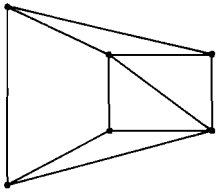


Fig. 5. The reference net.

First some notations will be introduced. The number of vertices of a net will be denoted by K, that of its dual by K', while the number of meshes is denoted by M and M' respectively. The number of wires is denoted by B. Henceforth B is called the *order of the net*. Apparently one has $M' = K$, $K' = M$, while according to the theorem of Euler the following relation holds:

$$K + M = K' + M' = B + 2.$$

Let N be a net with vertices $V_1, \ldots, V_K$, $K \geqslant 2$, and let INC$[i, j]$ be a matrix such that

$$\left.\begin{array}{l} \text{INC } [i,j] = 0, \text{ if } V_i \text{ and } V_j \text{ are not connected,} \\ \text{INC } [i,j] = -1, \text{ if } V_i \text{ and } V_j \text{ are connected,} \end{array}\right\} \ (i \neq j)$$
$$\text{INC } [i,i] = \text{the number of wires at } V_i.$$

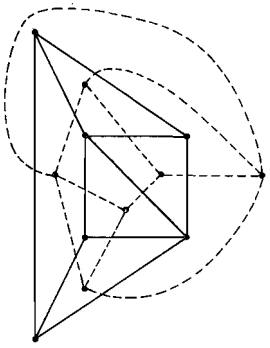


Fig. 6. The reference net and its dual.

It was shown by Brooks, Smith, Stone and Tutte [1]) that all first cofactors of INC are the same, except for the sign. Their common absolute value is called the *complexity* of the net; it is denoted by C. It can be shown that dual nets have equal complexities; furthermore the complexity equals the number of complete trees of the net [1]).

Simple squarings can be obtained from a c-net by placing an electromotive force of value C in one of the wires (all wires have unit resistance). The current flow caused in the network is called the *full flow*, while the currents are called the *full currents*. The highest common factor (HCF) of the full currents is

called the *reduction factor*, denoted by RF. If instead an electromotive force of value $C/(\mathrm{HCF})$ is placed in the wire under consideration, one obtains the *reduced flow* and the *reduced currents*.

The sides of the squaring obtained in this way are the *full sides* and *reduced sides* respectively. The full horizontal side equals the current caused by an electromotive force of value C in its own wire, while the full vertical side is equal to the potential difference between the two ends. The reduced horizontal and reduced vertical sides are obtained if instead an electromotive force of value $C/(\mathrm{HCF})$ is applied.

A squared rectangle that contains a squared rectangle of lower order in its interior and any corresponding p-net are called *compound*; all other squared rectangles and p-nets are *simple*. If a p-net has a part not containing a pole joined to the rest by only two wires, or if it has a pair of vertices joined by two (or more) wires, then these wires will have equal currents. If these currents are not zero, the resulting imperfection is said to be *trivial*.

2.2. Code of the c-net

Next we come to the question of how a general network can be stored into an electronic computer. Obviously the vertex-vertex incidence matrix INC can be used for this purpose; the network is determined uniquely by the matrix INC, and vice versa.

However, it is quite difficult to find out whether the network is planar or not if only the matrix INC is given. In addition, even if the network is known to be planar, it is still difficult to draw the net without crossings from the knowledge of INC alone.

In order to overcome these difficulties, a new code is introduced. It is assumed that the planar network is drawn on the sphere. The vertices are numbered arbitrarily from 1 to K.

The boundary of a mesh contains a set of vertices. A *code of a mesh* is obtained as follows: While walking in the positive sense along the boundary of the mesh, starting with V_i, we encounter V_j, V_k, V_l, ..., until we return to V_i. The sequence V_i, V_j, V_k, ..., V_i is a code of the mesh.

Example:

A possible code of mesh 1 of the reference net is 1 2 6 5 1, as can be seen from fig. 7; but we can also take 2 6 5 1 2, 6 5 1 2 6 or 5 1 2 6 5.

A *code of a net* is the sequence of codes of all its meshes separated by zeros. At the end two more zeros are added. Hence this code of the net can be considered as a vector $V[t]$, $t = 1, 2, \ldots, 2(B + M) + 1$.

Example:

A code of the reference net is as follows:

1 2 6 5 1 0 2 3 6 2 0 3 5 6 3 0 3 4 5 3 0 1 5 4 1 0 1 4 3 2 1 0 0.

It should be noticed that a different code would have been obtained if the vertices were enumerated in another way. Furthermore, the chosen codes of the meshes may be permuted in the code considered. Any of the codes so obtained is sufficient to characterize the net topologically.

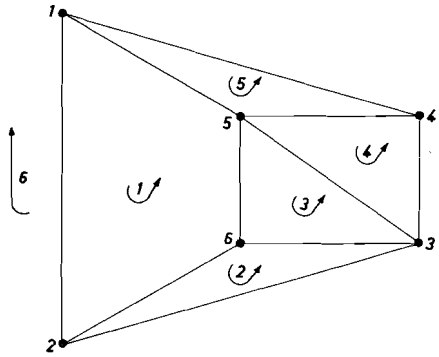


Fig. 7. The reference net.

2.3. Determination of the branches of the c-net

A wire contains two vertices of the net. To each pair of vertices V_i and V_j of a wire two arrows are associated. The first is directed from V_i to V_j and the second from V_j to V_i. The wire with the arrow directed from V_i to V_j will be called *branch* V_iV_j, the other is *branch* V_jV_i. A branch is therefore an oriented wire; it contains two vertices, which are denoted by *branch 1* and *branch 2*. If only one of the two branches V_iV_j and V_jV_i is used to indicate the associated wire, then the net has B branches.

Therefore the branch i is denoted by its two vertices, namely, branch 1[i] and branch 2[i], with $i = 1, 2, \ldots, B$. In the same way the branches of the dual net are denoted by branchdual 1[i] and branchdual 2[i]. It is further assumed that the meshes are numbered from 1 to M in the same sequence as their codes occur in the code of the net.

The branches of the net and its dual can be derived from the code $V[t]$ of the net by the following programme:

```
procedure form branches (V, branch 1, branch 2, branchdual 1, branchdual 2,
        K, M);
        integer K, M;
        integer array branch 1, branch 2, branchdual 1, branchdual 2, V;
begin   integer m, t, tt, i;
    t: = m: = 1; tt: = 0;
begin: for i: = 1 step 1 until tt do
        begin
            if V[t+1] = branch 1[i] ∧ V[t] = branch 2[i]
```

```
                then
                    begin
                        branchdual 2[i]: = m; go to next
                    end
            end i;
        tt: = tt + 1; branch 1[tt]: = V[t]; branch 2[tt]: = V[t+1];
        branchdual 1[tt]: = m;
next:   t: = t + 1;
        if V[t+1] = 0
        then
            begin
                if V[t+2] = 0
                then go to end;
                m: = m + 1; t: = t + 2
            end;
        go to begin;
end:    B: = tt; M: = m; K: = B + 2 − M
end form branches
```

Example:

Applying the **procedure** form branches to the code of the reference net one obtains a set of branches which are given below:

i	branch 1[i]	branch 2[i]	branchdual 1[i]	branchdual 2[i]
1	1	2	1	6
2	2	6	1	2
3	6	5	1	3
4	5	1	1	5
5	2	3	2	6
6	3	6	2	3
7	3	5	3	4
8	3	4	4	6
9	4	5	4	5
10	4	1	5	6

2.4. Dualization of the c-net

From the code $V[t]$ of the net it is possible to obtain the code of the dual net. To see this, it is first necessary to define the concept of adjacent vertex and adjacent branch of a vertex. An *adjacent vertex* V_2 of a vertex V_1 is a vertex that is connected to V_1. The branch V_1V_2 will be called an *adjacent branch* of V_1.

A mesh is said to be *left* of a branch V_iV_j of its boundary, if the sequence V_iV_j occurs in the code of the mesh. In that case the branch is said to be right of the

mesh. The mesh is said to be *right* of a branch V_iV_j if the sequence V_jV_i occurs in the code of the mesh; if so the branch is left of the mesh.

Example:

In the reference net the mesh 1 is left of branch 12, but right of branch 21.

Now it is known that the vertices of the dual net are corresponding to the meshes of the original net and that the meshes of the dual net are corresponding to the vertices of the original net. Assuming again that the vertices are numbered from 1 to K, while the meshes are numbered from 1 to M, according to the occurrence of their codes in the code of the net, we choose the enumeration of the meshes of the dual net equal to the enumeration of their corresponding vertices of the original net. The same is done for the vertices of the dual net and the meshes of the original net.

Next we consider an arbitrary vertex V_0. To this vertex V_0 a set of *left-cyclic-ordered adjacent branches* will be associated in the following way: Take an arbitrary adjacent vertex V_1 of V_0; then search the *left* mesh L_1 of V_0V_1; then search the other adjacent vertex V_2 of V_0 in L_1; then search the *left* mesh L_2 of V_0V_2; and so on, until V_1 has been reached again.

The set V_0V_1, V_0V_2, . . . , V_0V_k, where $k \geqslant 3$, will be called the left-cyclic-ordered adjacent branches of V_0. If *left* is replaced by *right*, then the right-cyclic-ordered adjacent branches of V_0 are obtained.

The sequence L_1, L_2, . . . , L_k, L_1 is precisely a code of the mesh V_0 of the dual net.

If this procedure is carried out for all vertices of the net, the code of the dual net is obtained. How it can be done in an automatic way is described by the **procedure** dualize, as follows:

```
procedure dualize (branch 1, branch 2, branchdual 1, branchdual 2, K, V);
        integer K;
        integer array branch 1, branch 2, branchdual 1, branchdual 2, V;
begin integer i, j, l, h, t, search, remember;
    integer array vector 1, vector 2[1:B];
    t: = 0; i: = 1;
start: l: = 1;
    for j: = 1 step 1 until B do
      begin
          if branch 1[j] = i
          then
              begin
                  vector 2[l]: = branchdual 1[j];
                  vector 1[l]: = branchdual 2[j]; l: = l + 1
              end;
          if branch 2[j] = i
```

```
                then
                    begin
                        vector 1[l]: = branchdual 1[j];
                        vector 2[l]: = branchdual 2[j]; l: = l + 1
                    end
            end;
        t: = t + 1; V[t]: = vector 1[1]; search: = remember: = vector 2[1];
begin: for h: = 1 step 1 until l—1 do
            begin
                if vector 1[h] = search
                then
                    begin
                        t: = t + 1; V[t]: = search; search: = vector 2[h];
                        if search = remember
                        then go to continue;
                        go to begin
                    end
            end;
continue: t: = t + 1; V[t]: = 0; i: = i + 1; if i = K + 1
                                                then go to end;
            go to start;
end: t: = t + 1; V[t]: = 0
end dualize
```

Example:

The code of the dual of the reference net will become

6 1 5 6 0 1 6 2 1 0 2 6 4 3 2 0 4 6 5 4 0 1 3 4 5 1 0 1 2 3 1 0 0.

2.5. Determination of the vertex-vertex incidence matrix of the c-net

The vertex-vertex incidence matrix INC is easily determined from the branches of the original net. It is described by the following programme:

```
begin  integer i, j, k, m, n;
       for i: = 1 step 1 until K do
           for j: = 1 step 1 until K do INC[i, j]: = 0;
       for k: = 1 step 1 until B do
           begin
               m: = branch 1[k]; n: = branch 2[k];
               INC[n, m]: = INC[m, n]: = —1;
               INC[m, m]: = INC[m, m] + 1;
               INC[n, n]: = INC[n, n] + 1
           end
end
```

2.6. Wheels

Finally, a set of special nets are worth mentioning, namely the so-called wheels. A *wheel* is a c-net with an even number B of branches, with one vertex $p_{\frac{1}{2}B}$ and $B-1$ vertices p_3, where p_k means a vertex incident with k branches (see fig. 8).

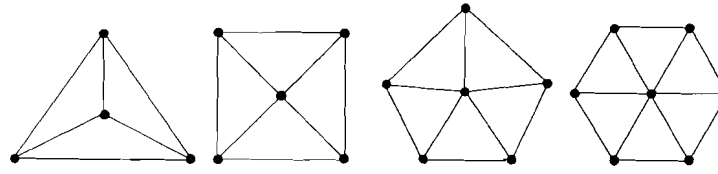


Fig. 8. The first few low-order wheels.

The code of a wheel having B branches is determined by the **procedure** wheel. It is supposed that W has been declared as **integer array** variable.

```
procedure wheel (B); value B;
        integer B;
begin integer MDP, t, l;
      MDP: = B ÷ 2 +1; t: = 1;
      for l: = 1 step 1 until MDP−2 do
         begin
               W[t]: = W[t+3]: = l; W[t+1]: = MDP; W[t+2]: = l + 1;
               W[t+4]: = 0; t: = t + 5
         end;
      W[t]: = W[t+3]: = MDP − 1; W[t+1]: = MDP; W[t+2]: = 1;
      W[t+4]: = 0; t: = t + 5;
      for l: = 1 step 1 until MDP − 1 do
         begin
               W[t]: = l; t: = t + 1
         end;
      W[t]: = 1; W[t+1]: = W[t+2]: = 0
end wheel
```

CHAPTER 3

IDENTIFICATION PROBLEM

3.1. Introduction

Consider the set S_B of c-nets having B wires. Let s be an element of S_B and s' its dual. Then, according to Tutte [11]), we have the following theorem: If s is not a wheel, then at least one of the nets s and s' can be constructed from an element σ of S_{B-1} by addition of a wire joining two vertices of σ.

With the aid of this theorem, the set S_B can be constructed from the set S_{B-1}. To this end, we start with the set T_{B-1} of the codes of the $(B-1)$-wire c-nets; for each element of S_{B-1}, we have one element of T_{B-1}. Take one element of the set T_{B-1}; it represents a certain $(B-1)$-wire c-net. Add a wire in this c-net, in so far as the result is a B-wire c-net, and construct a code representing the latter c-net. If this procedure is carried out for all elements of T_{B-1} and for all possibilities of adding wires, a set Σ_B of codes is obtained, of which each code represents a B-wire c-net. Let s be an element of S_B, then either s or its dual s' is represented by an element of Σ_B.

In the set Σ_B there may be many codes representing the same net. Now two questions arise:

(1) How can the set Σ_B be constructed in an automatic way on an electronic computer?

(2) How (if Σ_B is available) can equal nets represented by different codes be identified, and how can this be done on a computer, so as to obtain the set T_B?

3.2. Generation of nets by means of their codes

Apparently, addition of a wire to a c-net s, by joining two of its vertices, gives a non-planar network unless these two vertices belong (before joining) to one and the same mesh of s.

Let the net s contain a mesh R of b wires ($b > 3$) and let $V_1\ V_2\ V_3 \ldots V_b\ V_1$ be the code of R. Apparently a net s^* is obtained if two vertices V_i and V_j, not being adjacent vertices, are joined by a wire. This can be done in $b(b-3)/2$ different ways, and in any of these ways the mesh R is split into two smaller meshes, R_1 and R_2.

For example, if V_1 is joined to V_3 two new meshes having the following codes are obtained: $V_1\ V_2\ V_3\ V_1$ and $V_3\ V_4 \ldots V_b\ V_1\ V_3$. The total number of elements of these codes exceeds the number of elements in the code of mesh R by 3. This is true if two arbitrary non-adjacent vertices of R are joined, for any R of s. Hence the number of elements of the code of the new net s^* exceeds the number of elements of the code of the original net s by 4 (in the code of s^*, the codes of R_1 and R_2 are separated by the element 0).

In the following programme it is described how, starting from a code representing a certain net, the codes of the new nets are obtained (addition of a wire in the original net). If the original net is not selfdual, then the dual net is constructed and the procedure repeated (addition of a wire in the dual net).

```
procedure generate nets (W); integer array W;
        comment if the net from which the new nets are generated is selfdual,
                it is assumed that a Boolean variable selfdual is true, other
                wise selfdual is false;
begin   Boolean dualized;
        dualized: = false; go to con 2;
con 1:  if dualized ∨ selfdual
        then go to finished;
        form branches (W, branch 1, branch 2, branchdual 1, branchdual 2, K, M);
        dualize (branch 1, branch 2, branchdual 1, branchdual 2, K, W);
          dualized: = true;
con 2:  begin integer i, ii, m, s, t, MM, p, q, a, b, l;
            integer array sum [1 : M + 1], multiplicity [1 : M];
            m: = t: = sum [1]: = i: = 1;
        label: if W[t+2] = 0
               then
                   begin
                       t: = t + 3; i: = i + 1; sum [i]: = t;
                       multiplicity [i−1]: = m; m: = 1; if W[t] = 0
                                                        then go to follow
                   end;
               t: = t + 1; m: = m + 1; go to label;
        follow: MM: = i − 1;
               for ii: = 1 step 1 until MM do
                   begin
                       if multiplicity [ii] > 3
                       then
          begin
            q: = sum [ii] − 1;
            for a: = 1 step 1 until sum [ii] − 1 do V[a]: = W[a];
            for b: = sum [ii + 1] step 1 until sum[MM+1] do V[b+4]: = W[b];
            for s: = 1 step 1 until multiplicity [ii] − 2 do
              begin
                for l: = s + 2 step 1 until if s = 1 then multiplicity [ii] − 1
                                             else multiplicity [ii] do
                  begin
                    p: = q + 1;
```

```
                    for m: = s step 1 until l do
                        begin
                          V[p]: = W[m+q]; p: = p+1
                        end;
                    V[p]: = W[s+q]; p: = p+1; V[p]: = 0; p: = p+1;
                    for   m: = l step 1 until multiplicity [ii] do
                          begin
                            V[p]: W[m+q]; p: = p+1
                          end;
                    for   m: = 1 step 1 until s do
                          begin
                            V[p]: = W[m+q]; p: = p+1
                          end;
                    V[p]: = W[l+q]; V[p+1]: =0;
                        comment at this point the net can be identified,
                        the procedure form TNSTAR will be explained
                        later; form TNSTAR;
                     end l
                end s
            end if
                    end ii
            end block con 2; go to con 1;
finished:
end generate net
```

Example:

From the code of the reference net (which is selfdual) four new codes can be generated. The new codes are denoted by $V_k[t]$ ($k=1,2,3,4$), while that of the reference net is denoted by $W[t]$.

t = 1(1)37

$W[t]$ = 1 2 6 5 1 0 2 3 6 2 0 3 6 5 3 0 3 4 5 3 0 1 5 4 1 0 1 4 3 2 1 0 0

$V_1[t]$ = 1 2 6 1 0 6 5 1 6 0 2 3 6 2 0 3 6 5 3 0 3 4 5 3 0 1 5 4 1 0 1 4 3 2 1 0 0

$V_2[t]$ = 2 6 5 2 0 5 1 2 5 0 2 3 6 2 0 3 6 5 3 0 3 4 5 3 0 1 5 4 1 0 1 4 3 2 1 0 0

$V_3[t]$ = 1 2 6 5 1 0 2 3 6 2 0 3 6 5 3 0 3 4 5 3 0 1 5 4 1 0 1 4 3 1 0 3 2 1 3 0 0

$V_3[t]$ = 1 2 6 5 1 0 2 3 6 2 0 3 6 5 3 0 3 4 5 3 0 1 5 4 1 0 4 3 2 4 0 2 1 4 2 0 0

3.3. Identification problem

We now return to question (2) of sec. 3.1, which may be phrased somewhat differently as follows: How can we find out whether or not two different codes represent one and the same net? What is more, how can we uniquely characterize the net if and when it is represented by one of its many possible codes? This set of problems is henceforth referred to by the expression "identification

problem". To solve this identification problem is of course much more complicated than the construction of the set of codes Σ_B.

Two nets are equal if an enumeration of the vertices can be found such that the vertex-vertex incidence matrices INC of the two nets are equal. In principle it is possible to run through all $K!$ permutations of the vertices of one net and compare the corresponding incidence matrices with that of the other net. However, such a procedure takes a long time, even on a fast computer.

It would be much better if from the code there could be found a characteristic of the net determining the latter in a unique way. In a first attempt to find such a characteristic, we tried several simple and obvious possibilities. However, already at an early stage it became apparent that these characteristics did fail to characterize the net uniquely.

The characteristics can be divided into two types: Type 1 of characteristic is such that two nets having different characteristics are different. Type 2 of characteristic is such that two nets having equal characteristics are equal. Apparently a characteristic of both type 1 and type 2 determines the net uniquely.

3.4. Type-1 characteristics

We will see in how far the identification problem can be solved if use is made of a characteristic that is of type 1 and not of type 2. If the generation process of sec. 3.2 is applied to the set T_{B-1} the set Σ_B is obtained. The set Σ_B has many more elements than the set T_B. That means, many nets corresponding to codes in Σ_B are equal. With the characteristic under consideration, nets having different characteristics can be discriminated. However, nets having equal characteristics need not be equal; that is, the remaining undiscriminated nets represented by elements of Σ_B have to be tested in a different way. This causes much extra labour if the set Σ_B is much larger than the set T_B.

Some simple examples of characteristics of type 1 will be discussed now. The first example is a vector A of which the elements $A[k]$ denote the number of vertices incident with k wires, $k \geqslant 3$. The reference net consists of 2 p_4's and 4 p_3's. Hence $A[3] = 4$ and $A[4] = 2$. A short notation is $A = 3^4 4^2$.

Another example is the combination of the characteristic A of a net with A' of its dual, (A, A'). For example, in the case of the reference net we have $(3^4 4^2, 3^4 4^2)$.

That the characteristic (A,A') is not of type 2 can be seen from fig. 9, where two different nets with the same characteristic (A,A') are shown.

A third and last example of characteristics of type 1 is due to Bouwkamp. He considered a matrix D of which the elements $D[k,j]$ denote the number of wires that connect a vertex p_k to a vertex p_j. Apparently D is symmetric. Furthermore, D has the property that the sum of the elements to the right of the main diagonal plus the trace equals the number of wires B of the net. For the reference

net and its dual the matrices D and D' are as follows:

$$D = \begin{Vmatrix} 3 & 6 \\ 6 & 1 \end{Vmatrix}, \qquad D' = \begin{Vmatrix} 3 & 6 \\ 6 & 1 \end{Vmatrix}.$$

The combination of the four characteristics A,A',D,D' will be denoted by $I = (A,A',D,D')$. It is easy for the computer to determine I from the code of the net, but I is by no means fully discriminating. For example, S_{16} has 249 elements, except for duals, but there are in this case only 169 different characteristics I (see also fig. 9).

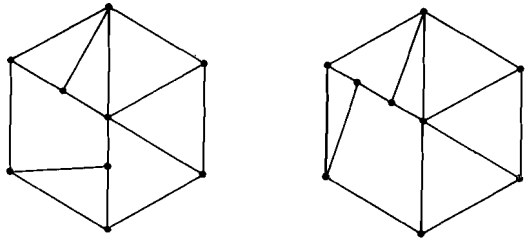


Fig. 9. Example of two distinct c-nets with the same characteristic I.

3.5. Type-2 characteristics

A characteristic of type 2 can be used as a sieve. Nets having equal characteristics can be omitted. Especially if the characteristic is selective not much extra work has to be done. First the remaining nets having different characteristics can be classified according to their complexity. Only nets having equal complexities have to be investigated. Now the characteristic I of type 1 can be applied. If this does not discriminate either, then at last the Bouwkamp codes can be calculated; with the aid of these codes two nets can always be discriminated.

3.6. A characteristic of both type 1 and type 2

Consider the vertex-vertex incidence matrix INC of the net as obtained from the code of the latter. The off-diagonal elements of INC are either zero or minus one. We replace the off-diagonal elements by their absolute values. Then an element on the diagonal is the sum of the off-diagonal elements in the same row (or column). The new matrix will be denoted by X, with elements X_{ij} $(i,j = 1, 2, \ldots, K)$.

To X an integer $G(X)$ will be associated. The binary notation of $G(X)$ is obtained by writing the elements of X to the right of the main diagonal in the sequence $X_{12}\, X_{13} \ldots X_{1K}\, X_{23}\, X_{24} \ldots X_{2K} \ldots X_{K-1,K}$ so that its decimal value is given by

$$G(X) = \sum_{i=1}^{K-1} \sum_{j=i+1}^{K} X_{ij}\; 2^{\frac{1}{2}(K-i)(K-i-1)+K-j}.$$

$G(X)$ is called the identification number of the net in relation to the code of the net under consideration. If $G(X)$ is known, X and INC are known, and vice versa: From the identification number the upper triangle of X can be constructed while the lower triangle follows from the symmetry of X; the diagonal elements may be found from the sums of the off-diagonal elements in the same row.

Let now the matrix X be transformed by interchanging the kth row with the lth row and at the same time the kth column with the lth column. This transformation is nothing but a new enumeration of the vertices; such an enumeration is called a permutation. For every permutation we have an X and the corresponding $G(X)$. Let GM be the maximum of $G(X)$ on the group of permutations. The number GM is independent of the particular choice of the code of the net. Hence GM is a characteristic of both type 1 and type 2; it is called *identification magnitude*.

A permutation (there may be more than one) for which $G(X)$ is maximum on the group of permutations of X brings the matrix in the maximal form, say. The matrix can be brought into this maximal form by running through all possible permutations (there are $K!$ of them) and by testing which permutation gives the maximal G. If K is large this process is time consuming.

Instead of considering the full permutation group one can consider a subgroup of the group of all permutations (by imposing enough requirements on X) and maximize $G(X)$ on this subgroup.

Let p be a permutation of the full permutation group of the net. To each p there corresponds an identification number $G(X_p)$. If $G(X_{p1}) = G(X_{p2}) = \ldots = G(X_{pi})$ we identify the elements $p_1, \ldots, p_i$ to an element h. These new elements h form the set H.

If there exist p_1 and p_2 such that $G(Xp_1) = G(Xp_2)$ it is possible to deform the net topologically, after having fixed the enumeration (corresponding to the permutation p_1) to the vertices, such that the deformed net can be considered as the non-deformed net with an enumeration corresponding to the permutation p_2.

Next, let the set $H^* \subset H$ be such that if $h^* \epsilon H^*$ the permutations corresponding to h^* are satisfying certain criteria $CR_1, CR_2, \ldots, CR_s$. Then the identification problem is solved if enough criteria can be found (i.e. s just so large) that the set H^* contains only one element. If H^* contains more than one element the identification number G can be maximized on H^*, and the work involved may be considerably less compared to the maximization on the full permutation group.

Another possibility to determine the maximum of G on a certain permutation group H^{**} is to construct certain paths through H^{**}, of which it is known that they lead to the maximal G on H^{**} (steepest ascent).

3.7. Example of a type-2 characteristic

Instead of maximizing the identification number G one can maximize other

numbers defined on the permutation group. For example, the following procedure was attempted. In a maximalization process tested on one of the available computers the number G^* was maximized where G^* is defined by

$$G^* = \sum_{i=1}^{K-1} \sum_{j=i+1}^{K} X_{ij} \{2^{(K-j)(K-1)+K-i-1} + 2^{(K-i)(K-1)+K-j}\}.$$

It is assumed that the main diagonal elements of the matrix X are non-increasing and remain so in the maximalization process. The transformation applied to X was the interchange of two rows and the corresponding columns. The columns j and k (and the corresponding rows) were tried for interchange when $X[i,j] = 0$ and $X[i,k] = 1$, $k > j$, while the main diagonal elements remained non-increasing. The process was stopped when no i, j and k could be found such that G^* increased when the columns j and k were interchanged. The reason why this process works only as a sieve is that there are cases where more than two rows and columns have to be interchanged simultaneously in order to increase G^*. The sieve works much better if the method is applied to both the original net and its dual. This was tested on those codes of Σ_{16} that are representing nets for which $K = M = 7$ and it gave perfect discrimination. The method is still unsatisfactory even when both the original and dual nets are "maximized" because a special programme is necessary for identifying the nets as soon as the identification numbers corresponding to the "maximum" permutation have been calculated: one has also to remember which nets are dual. The method that can be used is that of drawing chains in the set Σ_B. A chain can be drawn either when two codes correspond to nets having equal identification numbers or when it is known that the nets are dual. The process of drawing chains has been carried out on a computer.

3.8. Weights and scores

With the method of "weights and scores" a sequence of importance of the vertices of the net is calculated that does not depend on the particular code representing the net. As soon as a sequence of importance (this is a permutation) is known the identification number corresponding to that permutation is calculated. This identification number is used to characterize the net.

To each vertex of the net a weight is assigned; all weights are assembled in a vector: weight [i], $i = 1(1)K$. The weights can change during the process; the process of weights and scores is ready when the weights of all vertices are different.

The process starts with the weights of all vertices equal to 2. New weights are assigned after "scores" have been calculated. The scores are given by a vector: score [i], $i = 1(1)K$. Depending on the value of a **Boolean** variable: fromdual, scores are calculated with the aid of the weights of the original or the dual net. The scores are calculated by the following programme:

```
begin integer i;
      Boolean fromdual;
      integer array weight original, score [1:K], weight dual [1:M];
      for i: = 1 step 1 until K do score [i]: = 0;
      if ¬ fromdual
      then
        for i: = 1 step 1 until B do
        begin
        score [branch 1 [i]]: = score [branch 1 [i]] + weight original [branch 2 [i]];
        score [branch 2[i]]: = score [branch 2 [i]] + weight original [branch 1 [i]]
        end
      else
        for i: = 1 step 1 until B do
        begin
        score [branch 1[i]]: = score [branch 1[i]]+weight dual [branchdual 2[i]];
        score [branch 2[i]]: = score [branch 2[i]]+weight dual [branchdual 1[i]]
        end
end
```

Example:
For the reference net the start is as follows:

vertex i	weight $[i]$
1	2
2	2
3	2
4	2
5	2
6	2

When the scores are calculated (fromdual is **false**) one obtains

i	score $[i]$
1	6
2	6
3	8
4	6
5	8
6	6

With the aid of the score a new weight is calculated. One tries to discriminate between vertices that have equal weights so far, by means of their scores. First, all vertices of weight 2 are searched; those among them having the lowest score get a new weight equal to 2. The vertices having the next lowest score get a new weight twice as large, and so on. Then all vertices of (old) weight 4 are searched;

those among them having the lowest score get a new weight twice as large as the last given weight, and so on, until all vertices have got new weights.

The maximum weight is remembered. Then again new scores are calculated. The process is stopped if the maximum weight assigned equals 2^K or if the maximum weight has not been changed. In the latter case we shall say there is "no gain". Consequently, in continuing the discussion of our example, the next step for the reference net is

i	weight [i]	score [i]
1	2	8
2	2	8
3	4	10
4	2	10
5	4	10
6	2	10

Maximum weight = 4.

Then

i	weight [i]	score [i]
1	2	14
2	2	14
3	8	18
4	4	18
5	8	18
6	4	18

Maximum weight = 8.

Finally

i	weight [i]
1	2
2	2
3	8
4	4
5	8
6	4

Maximum weight = 8, hence there is no gain. The reason that the process stops on no gain, if it is applied to the reference net, is that the reference net has certain symmetry properties. Apparently in any permutation of the reference net the vertices 1 and 2, 4 and 6, 3 and 5 can be interchanged without changing the identification number corresponding to that permutation.

3.9. Procedure identify

Now the **procedure** identify will be described. When the branches of the original and dual nets have been found, this procedure calculates a sequence of

importance of the vertices. This sequence is given by a vector: location [i], $i = 1(1)K$. If, entering the procedure, fromdual is **false**, scores are calculated with the aid of the original net alone. If fromdual is **true**, scores are calculated the first time with the aid of the dual net, and later with the aid of the original net. If, after coming back from the procedure, the **Boolean** variable nogain is **false**, the identification process is ready; if nogain is **true**, the identification process is not yet ready. The maximum weight assigned is indicated by the procedure. It is assumed that weight original [$K+1$] is equal to zero. The **procedure** identify is given by the following programme:

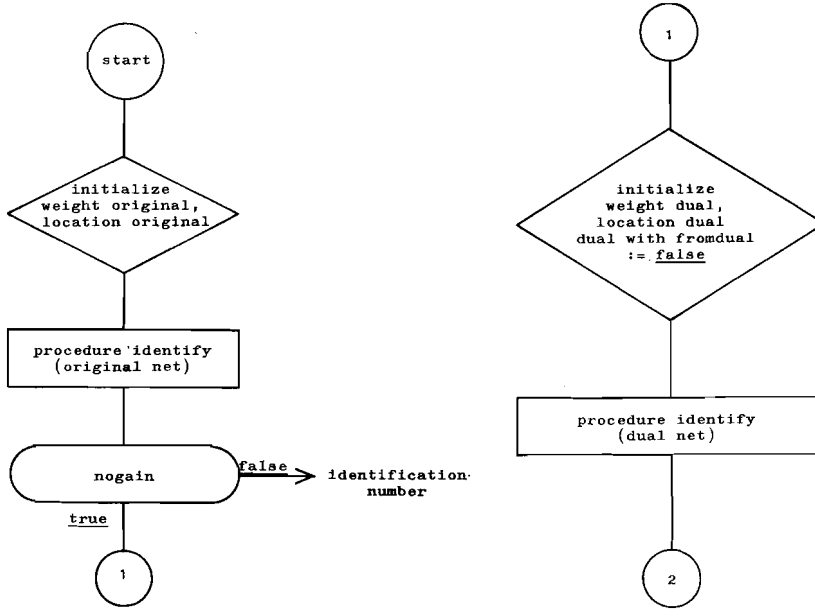
```
procedure identify (K, weight original, weight dual, location, maxweight, n,
        branch 1, branch 2, branchdual 1, branchdual 2);
integer K, n, maxweight;
integer array weight original, weight dual, location, branch 1, branch 2, branch
        dual 1, branchdual 2;
begin integer z, weightstorage, t, i, k, q, s, l, min;
    Boolean ready;
    integer array new location, score [1:K];
    nogain: = false;
start: z: = 1; weightstorage: = 1; t: = 1;
    for i: = 1 step 1 until K do score [i]: = 0;
    if fromdual
    then
        for i: = 1 step 1 until B do
        begin
          score [branch 1 [i]]: = score [branch 1 [i]] + weight dual [branch
          dual 2[i]];
          score [branch 2 [i]]: = score [branch 2 [i]] + weight dual [branch
          dual 1 [i]];
          fromdual: = false
        end
    else
        for i: = 1 step 1 until B do
         begin
          score [branch 1 [i]]: = score [branch 1 [i]] + weight original
          [branch 2 [i]];
          score [branch 2 [i]]: = score [branch 2 [i]] + weight original
          [branch 1 [i]];
         end;
label 1: for i: = z step 1 until K do
         begin
          if weight original [location [i]] ≠ weight original [location [i+1]]
```

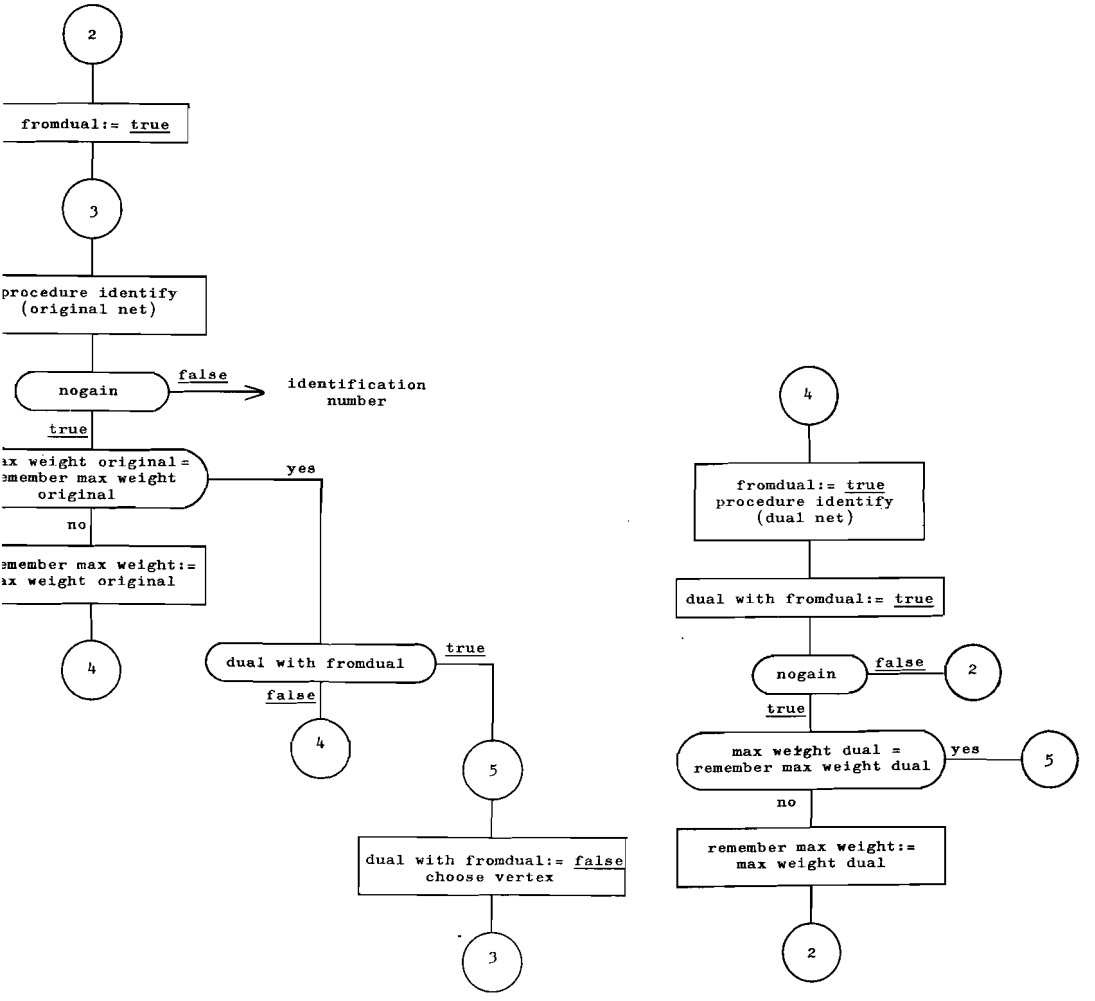
```
            then go to continue
           end;
continue: if i > z
          then
           begin
            for k: = z step 1 until i do weight original [location [k]]: = 0;
            label 2: min: = M * 2↑K; ready: = true;
            for l: = z step 1 until i do
             begin
              if score [location [l]] < min ∧ weight original [location [l]] = 0
              then
               begin
                min: = score [location [l]]; ready: = false
               end
             end;
            if ready
            then go to continue i;
            weightstorage: = 2 * weightstorage;
            for n: = z step 1 until i do
             begin
              if score [location [n]] = min
              then
               begin
                weight original [location [n]]: = weightstorage;
                new location [t]: = location [n]; t: = t + 1
               end
             end n;
            n: = i; go to label 2
           end of then of i > z
          else
           begin
            weightstorage: = 2 * weightstorage;
            weight original [location [i]]: = weightstorage;
            new location [t]: = location [i]; t: = t + 1
           end else;
continue i: z: = i + 1; if z ≦ K
                        then go to label 1;
          if weightstorage ≠ 2↑K
          then
           begin
            if weightstorage = maxweight
            then
```

```
        begin
          nogain: = true; go to finish
        end
      else
        begin
          for s: = 1 step 1 until K do location [s]: = new location [s];
          maxweight: = weightstorage; go to start
        end else;
    for s: = 1 step 1 until K do location [s]: = new location [s];
finish:
end identify
```

3.10. Procedure identification

In the case of no gain the **procedure** identify is applied to the dual net. Then it is applied again to the original net, now calling the **procedure** with fromdual is **true**. It may happen that after coming back from the procedure there is still no gain. If the maximum weight is not increased, it is investigated whether the dual net has been used with the aid of the original net (if so, then dual with fromdual is **true**). If the maximum weight has been increased, but there is no gain, the dual net is identified (application of the **procedure** identify) with fromdual is **true**. When no further improvement can be made due to the symmetry, one of the vertices having equal weights (one with the maximum possible weight) is chosen; its weight is increased by unity. Then the **procedure** identify is called again for the original net, with fromdual is **false**. The number of choices





may be more than one; it determines the degree of symmetry of the net. If nogain is **false** the identification number corresponding to the permutation location [*i*] of the vertices of the original net is calculated.

The calculation of this identification number is shown in the accompanying flow chart. The associated programme is given by the **procedure** identification.

procedure identification (*V*, identificationnumber);
  integer identificationnumber;
  integer array *V*;
begin integer *i*, *j*, workstorage, maxweight original, maxweight dual, remember
    maxweight original, remember maxweight dual, *n* original, *n* dual;
  Boolean dual with fromdual, fromdual, nogain, *l*;

```
integer array weight original, location original [1:K+1], inverse location
              [1:K], weight dual, location dual [1:M+1];
number of choices: = 0;
for i: = 1 step 1 until K do
                begin
                    weight original [i]: = 2; location original [i]: = i
                end;
location original [K+1]: = K + 1; weight original [K+1]: = 0; from
dual: = false; maxweight original: = 2;
identify (K, weight original, weight dual, location original, maxweight
original, n original, branch 1, branch 2, branchdual 1, branchdual 2);
remember maxweight original: = maxweight original;
if ¬ nogain
then go to form identificationnumber;
for i: = 1 step 1 until M do
                begin
                    weight dual [i]: = 2; location dual [i]: = i
                end;
location dual [M+1]: = M + 1; weight dual [M + 1]:=0; fromdual
: = false; maxweight dual: = 2; dual with fromdual: = false;
identify (M, weight dual, weight original, location dual, maxweight dual,
n dual, branchdual 1, branchdual 2, branch 1, branch 2);
remember maxweight dual: = maxweight dual;
two: fromdual: = true;
three: identify (K, weight original, weight dual, location original, maxweight
    original, n original, branch 1, branch 2, branchdual 1, branchdual 2);
    if ¬ nogain
    then go to form identificationnumber;
    if maxweight original = remember maxweight original
    then
        begin
            if dual with fromdual
            then
                begin
                five: weight original [location [n original]]: =
                    weight original [location [n original]] + 1;
                    number of choices: = number of choices + 1; dual
                    with fromdual: = false; go to three
                end; go to four
        end;
    remember maxweight original: = maxweight original;
four: fromdual: = true;
```

```
        identify (M, weight dual, weight original, location dual, maxweight dual,
                n dual, branchdual 1, branchdual 2, branch 1, branch 2);
        dual with fromdual: = true;
        if ¬ nogain
        then go to two;
        if maxweight dual = remember maxweight dual
        then go to five;
        remember maxweight dual: = maxweight dual; go to two;
form identificationnumber:
        for i: = 1 step 1 until K do inverse location [location original [i]]: = i;
        identificationnumber: = 0;
        for l: = 1 step 1 until B do
            begin
             i: = K + 1 — inverse location [branch 1 [l]];
             j: = K + 1 — inverse location [branch 2 [l]];
             if i > j
             then
                 begin
                     workstorage: = i; i: = j; j: = workstorage
                 end;
             identificationnumber: = identificationnumber +
             2↑((K↑2+K+i*(i—2*K+1)—2*j)÷2)
            end
end identification
```

3.11. Input and output procedures

It is assumed that the **procedure** identification calculates an invariant of the net. Let s again be an element of S_N and s' its dual. Then the set $S_N{}^*$ is built up as follows: If the number K of vertices of s is smaller than the number M of meshes, s is put in $S_N{}^*$. If $K > M$ then s' is put in $S_N{}^*$. If $K = M$ then of the nets s and s' that with the smaller identification number is put in $S_N{}^*$. If the net is selfdual s is put in $S_N{}^*$.

Each element of $S_N{}^*$ is represented by one of its possible codes. This code is called a representative of the element of $S_N{}^*$. The set of representatives of all elements of $S_N{}^*$ form the set $T_N{}^*$. Now taking one element of $T_N{}^*$, new nets are generated with the aid of the **procedure** generate nets. As soon as a new net is generated, the **procedure** form TNSTAR is called. In this procedure the identification number is calculated using the **procedure** identify, while with the aid of the **procedure** new net test it is determined whether this element of $T_{N+1}{}^*$ was already found. The parameter H, which is **integer**, denotes the number of new codes of $T_{N+1}{}^*$ found so far.

In the **procedure** form TNSTAR the **procedure** WRITE is used which is

described below. It writes on magnetic tape the code of a new element of T_{N+1}*, the number of choices, an indication whether the net is selfdual or not, and an indication whether the element is the last element of T_{N+1}* or not. In this procedure it is assumed that two new standard functions are added to the ALGOL-60 language. The first one is the **procedure** write (*E*), where *E* is an expression. This procedure writes an **integer** or **real** on magnetic tape. The second procedure is the parameterless **procedure** read, which reads the next number from magnetic tape. The format on tape determines whether the result is **integer** or **real**.

The procedures form TNSTAR, WRITE and new net test are given below.

```
procedure WRITE (W, number of choices, selfdual); integer number of choices;
                                                  Boolean selfdual;
                                                  integer array W;
        begin integer i; write (W[1]),
            for i: = 2 step 1 until i do
                begin
                    write (W[i]); if W[i—1] = 0 ∧ W[i] = 0
                            then go to end
                end;
        end: write (number of choices);
             if selfdual
             then write (1)
             else write (0)
        end WRITE;

procedure new net test (V, storage); integer storage;
                                     integer array V;
begin integer p;
      own integer array id number [1:4↑(B—9)];
      for p: = 1 step 1 until H do
                begin
                    if storage = id number [p]
                    then go to end
                end;
      H: = H + 1; id number [H]: = storage; WRITE (V, number of choices,
      selfdual);
end:
end new net test;
procedure form TNSTAR;
begin integer array U [1:2*(2*B÷3+B)+2)];
   form branches (V, branch 1, branch 2, branchdual 1, branchdual 2, K, M);
   if K = M
```

```
then
    begin
        identification (V, identificationnumber);
        storage: = identificationnumber;
        dualize (branch 1, branch 2, branchdual 1, branchdual 2, K, U);
        form branches (U, branch 1, branch 2, branchdual 1, branchdual 2,
        K, M);
        identification (U, identificationnumber);
        if identificationnumber < storage
        then
            begin
                selfdual: = false; new net test (U, identificationnumber)
            end
        else
            begin
              if identificationnumber > storage
              then
                begin
                  selfdual: = false; new net test (V, storage)
                end
              else
                begin
                  selfdual: = true; new net test (V, storage)
                end
            end
    end
else
    begin
      if K > M
      then
          begin
            dualize (branch 1, branch 2, branchdual 1, branchdual 2,
            K, U);
            form branches (U, branch 1, branch 2, branchdual 1, branch
            dual 2, K, M);
            identification (U, identificationnumber);
            selfdual: = false;
            new test net (U, identificationnumber)
          end
      else
            begin
              identification (V, identificationnumber);
              selfdual: = false; new net test (V, identificationnumber)
```

```
                end
        end
end form TNSTAR
```

With the aid of procedure READ the code of a net, the number of choices, an indication whether the net is selfdual or not, and an indication whether the net is the last net of T_N* or not, are read from magnetic tape. The programme is given below.

```
procedure READ (W, number of choices, selfdual, end of file);
        integer end of file, number of choices;
        Boolean selfdual;
        integer array W;
begin integer i, j;
      W[1]: = read;
      for i: = 1 step 2 until i do
          begin
              W[i+1]: = read; W[i+2]: = read;
              if W[i+1] = 0 ∧ W[i+2] = 0
              then go to end
          end;
end:  number of choices: = read;
      j: = read; if j = 0
                 then selfdual: = false
                 else selfdual: = true;
      end of file: = read
end READ
```

3.12. Complete generation and identification programme

We start with the set S_8* consisting of one element. This element is generated by the **procedure** wheel (8). Its code is written on magnetic tape. From the set S_8* the set S_9* is formed, and so on. Finally the complete programme is given in programme I. It is assumed that a **procedure** stop is added to the ALGOL language. This procedure stops the machine.

CHAPTER 4

DETERMINATION OF NETWORK CURRENTS

4.1. Introduction

In chap. 2 it was mentioned that the rectangle dissections can be obtained from the branch currents of a net after placing an electromotive force equal to the complexity in one of the branches of the net. In a net having N branches an electromotive force can be placed in N different ways, which will lead to N dissections (possibly all different). The currents in the branches follow uniquely from Kirchhoff's laws:

(1) The sum of the currents at any vertex is zero.

(2) In each electrical mesh, the sum of the electromotive forces is equal to $\Sigma I_s R_s$, where I_s and R_s denote the branch currents and the branch resistances respectively in the mesh under consideration.

4.2. The branch-mesh incidence matrix

It is clear that there are $M-1$ independent electrical meshes of the net. For these electrical meshes a choice will be made from the M meshes of the net. Apparently there are M possible choices. In an electrical mesh a current $i[m]$, $m = 1(1)M-1$, will be assumed. The positive direction of a mesh current is that of the positive sense of the mesh. The branch currents and mesh currents are connected by the relation $I = \Gamma i$. Here I is the vector of the branch currents having the elements $I[k]$, $k = 1(1)B$, while i is the vector of the mesh currents having the elements $i[m]$, $m = 1(1)M-1$, and Γ is the branch-mesh incidence matrix having B rows and $M-1$ columns. Furthermore we consider the vector E with elements $E[k]$, $k = 1(1)B$, denoting the electromotive force in branch $1[k]$, branch $2[k]$ and the vector e with elements $e[m]$, $m = 1(1)M-1$, denoting the sum of the electromotive forces in mesh m. The vectors E and e are connected [12)13)] by the relation $e = \Gamma' E$, where Γ' denotes the transpose of Γ. Now writing Z for $\Gamma'\Gamma$, it can be shown [12)13)] that $e = Zi$ and $I = \Gamma Z^{-1}\Gamma' E$, where Z^{-1} means the inverse of Z. The matrix Z has $M-1$ rows and columns. Furthermore, Z is symmetric and non-singular.

From the definition it follows immediately that $Z' = (\Gamma'\Gamma)' = \Gamma'\Gamma = Z$. Hence Z is symmetric. That the matrix is non-singular follows from the fact that the branch currents are determined uniquely by the electromotive forces and the resistances in the branches of the net and from the fact that the set of mesh currents $i[m]$, $m = 1(1)M-1$, is a maximal set of linearly independent mesh currents.

Now another matrix which is denoted by γ will be considered. It is obtained as follows: Consider the mesh currents in the M meshes of the net and let these currents form a vector j. Hence j has the elements $j = j[m]$, $m = 1(1)M$; one

of these elements is linearly dependent on the other elements. The matrix γ is defined by $I = \gamma j$. Apparently Γ can be obtained from γ by omitting a suitable column in γ. In fact M different Γ's can be obtained from γ. Since only planar networks will be considered, it is easy to see that in each row of the matrix γ two and only two elements are different from zero: in fact in a planar network each branch occurs in exactly two meshes. The sum of these elements is zero. The number of non-zero elements in a column is equal to the number of branches in the mesh corresponding to that column.

Next the matrix $\zeta = \gamma'\gamma$ is formed. The matrix Z follows from ζ by omitting one row and the corresponding column. Obviously the matrix ζ is singular. The elements $\zeta[r, s]$ of ζ are either zero or minus one for $r \neq s$. This element is obtained by multiplying the rth column of γ by the sth column of γ. Now r and s are denoting meshes. If r and s have no branch in common this product is zero. However if r and s are incident this product equals minus one. The meshes r and s can only have one branch in common, and the positive directions of the mesh currents is such that the mesh currents in the common branches are opposite. The elements $\zeta[i, i]$ are equal to the number of branches in mesh i. Hence it is clear that ζ is the vertex-vertex incidence matrix of the dual net.

It was shown by Brooks, Smith, Stone and Tutte [1]) that the absolute value of all first cofactors of ζ are equal to the complexity C of the net. This also implies that Z is non-singular.

4.3. Calculation of the currents

From the relation $I = \Gamma Z^{-1}\Gamma' E$ one can obtain all possible dissections from the net. Any particular dissection is obtained by placing an electromotive force of value C in a particular branch of the net. In that case the vector E contains only one non-zero element, and the resulting vector I hence is one column of $R = \Gamma Z^{-1}\Gamma'$ multiplied by the complexity. Therefore each column (or row) of R determines the elements of a rectangle.

The inverse of Z is obtained by using Gaussian elimination and backsubstitution. It is described in programme II and can be traced through the comments.

The matrix R can be obtained from ZINV $= Z^{-1}$ using the following programme, where it is assumed that R and ZINV are declared as **integer array** variables; the bounds of the subscripts follow from $R[1{:}B, 1{:}B]$ and ZINV $[1{:}M, 1{:}M]$.

```
begin integer i, r, s;
    for i: = 1 step 1 until M do
                                    begin
                                        ZINV (i, M): = 0; ZINV (M, i): = 0
                                    end;
```

```
    for r: = 1 step 1 until B do
        begin
            for s: = 1 step 1 until B do
                R[r, s]: = ZINV [branchdual 1[r], branchdual 1[s]]
                         — ZINV [branchdual 1[r], branchdual 2[s]]
                         — ZINV [branchdual 2[r], branchdual 1[s]]
                         + ZINV [branchdual 2[r], branchdual 2[s]]
        end
end
```

When the branch currents are known the imperfection can be tested. It is described in the following programme, where it is assumed that the variable imperfection is **Boolean.**

```
begin integer i, j;
    imperfection: = false;
    for i: = 1 step 1 until B—1 do
        for j: = i + 1 step 1 until B do
                                        begin
                                            if R[r, i] = R[r, j]
                                            then imperfection: = true
                                        end
end imperfection
```

Furthermore zero currents can be counted. It is assumed in the following programme that the variable zero currents is declared as **integer.**

```
begin integer i;
    zero currents: = 0;
    for i: = 1 step 1 until B do
                                begin
                                    if R[r, i] = 0
                                    then zero currents: = zero currents + 1
                                end
end zero currents
```

Finally we describe the calculation of the reduction factor RF for row r of $R[r, s]$. The programme that calculates RF uses the **procedure** HCF(x, y) which determines the highest common factor of two integers x and y. The variable RF is **integer.**

```
begin integer l, hcf;
    procedure HCF(x, y); integer x, y;
    begin integer RN1, RN2;
        RN1: = x; hcf: = y;
```

```
        algorithm: RN2: = RN1 − hcf*(RN1 ÷ hcf);
                   if RN2 ≠ 0
                   then
                       begin
                           RN1: = hcf; hcf: = RN2; go to algorithm
                       end;
                   hcf: = abs(hcf)
        end HCF;
        HCF(R[r, 1], R[r, r]);
        for l: = 2 step 1 until B do HCF(R[r, l]), hcf);
        RF: = hcf
end determination RF
```

CHAPTER 5

CONSTRUCTION OF BOUWKAMP CODES

5.1. Introduction

After having calculated the matrix R, it will be described in the sequel how the Bouwkamp codes of all dissections belonging to R can be obtained. The kth row or column of R is representing the currents in the branches of the original net after an electromotive force of value C has been placed in branch: branch 1[k], branch 2[k].

5.2. The vector ordered current

Of all vertices $V_1, \ldots, V_K$ of the original net the respective left-cyclic-ordered adjacent branches are considered. Their currents are considered as elements of a vector "ordered current". The sequence of the elements of ordered current is as follows: The currents through the left-cyclic-ordered adjacent branches of vertex 1 are put into ordered current first; the currents of the left-cyclic-ordered adjacent branches of vertex 2 are put into ordered current next; and so on. Apparently ordered current has $2B$ elements.

After a column of R has been calculated, it is necessary to know where, in ordered current, a particular element of this column has to be stored positive, and where, again in ordered current, it has to be stored negative. This information is given by two vectors, namely, positive [k] and negative [k], $k = 1(1)B$. Hence the current in branch: branch 1[k], branch 2[k] is given by the element: ordered current [positive [k]], while the current in branch: branch 2[k], branch 1[k] is given by the element: ordered current [negative [k]].

If an element of ordered current is given we also want to know to which branch this current belongs. This information can be obtained from a vector: from [k], $k = 1(1)2B$. The current: ordered current [k] is flowing in branch: branch 1 [from [k]], branch 2 [from [k]]. The following relation holds: $k =$ from [negative [k]] = from [positive [k]]. Finally we need to know for any vertex V_i the smallest l such that the branch belonging to ordered current [l] is an adjacent branch of V_i. Let this smallest l be l_i. The vector: address [k], $k = 1(1)K$, is defined by: address [k] = l_k.

In the next programme it is described how the vectors positive, negative, from, and address can be obtained assuming that the vectors branch 1, branch 2, branchdual 1 and branchdual 2 are given.

```
procedure left cyclic ordered adjacent vertices (branch 1, branch 2, branchdual 1,
        branchdual 2, positive, negative, address, from, K);
begin integer h, i, j, k, remember, meshsearch;
    k: = 1; i: = 1; address [0]: = 0; address [1]: = 1;
```

```
search first branch:
      for j: = 1 step 1 until B do
       begin
            if branch 1[j] = i
            then
              begin
                remember: = meshsearch: = branchdual 1[j]; from [k]: = j;
                positive [j]: = k;
                go to go on searching
              end;
            if branch 2 [j] = i
            then
              begin
                remember: = meshsearch: = branchdual 2[j]; from [k]: = j;
                negative [j]: = k;
                go to go on searching
              end
      end j;
go on searching:
      k: = k + 1;
      for h: = 1 step 1 until B do
        begin
          if branch 1[h] = i ∧ branchdual 2[h] = meshsearch
          then
              begin
                if branchdual 1[h] = remember
                then go to continue;
                from [k]: = h; positive [h] = k; meshsearch: = branchdual
                1[h];
                go to go on searching
              end;
          if branch 2[h] = i ∧ branchdual 1[h] = meshsearch
          then
              begin
                if branchdual 2[h] = remember
                then go to continue;
                from [k]: = h; negative [h]: = k; meshsearch: = branchdual
                2[h];
                go to go on searching
              end
        end h;
continue:
```

```
        i: = i + 1; address [i]: = k; if i ≠ K + 1
                                        then go to search first branch
end left cyclic ordening adjacent vertices
```

Example:

After applying the **procedure** left cyclic ordening adjacent vertices to the reference net we find:

i	branch 1[*i*]	branch 2[*i*]	branchdual 1[*i*]	branchdual 2[*i*]	positive [*i*]	negative [*i*]
1	1	2	1	6	1	4
2	2	6	1	2	6	18
3	6	5	1	3	20	14
4	5	1	1	5	17	2
5	2	3	2	6	5	7
6	3	6	2	3	10	19
7	3	5	3	4	9	15
8	3	4	4	6	8	11
9	4	5	4	5	13	16
10	4	1	5	6	12	3

i	from [*i*]	address [*i*]	*i*	from [*i*]
1	1	1	11	8
2	4	4	12	10
3	10	7	13	9
4	1	11	14	3
5	5	14	15	7
6	2	18	16	9
7	5	21	17	4
8	8		18	2
9	7		19	6
10	6		20	3

Next we calculate a vector: reduced ordered current. The elements of reduced ordered current are equal to the corresponding elements of ordered current divided by the reduction factor RF: reduced ordered current [*k*] = ordered current [*k*] ÷ RF. The following programme determines the vector: reduced ordered current. It should be noted that, to simplify notation, the vector: current [*s*] is identical with $R[r, s]$ for fixed r and $s = 1(1)B$.

```
begin integer i;
      for i: = 1 step 1 until B do
          begin
              reduced ordered current [positive [i]]: = current [i] ÷ RF;
              reduced ordered current [negative [i]]: = −current [i] ÷ RF
          end
end
```

Example:
Calculating the reduced ordered currents of the reference net, for $r = 3$, one obtains:

i	reduced ordered current [i]	i	reduced ordered current [i]
1	10	11	7
2	— 9	12	1
3	— 1	13	— 8
4	—10	14	—32
5	— 4	15	15
6	14	16	8
7	4	17	9
8	— 7	18	—14
9	—15	19	—18
10	18	20	32

5.3. Determination of the Bouwkamp codes

Now we consider the left-cyclic-ordered adjacent branches of a vertex V_0 and their currents:

V_0V_1	current V_0V_1
.	.
.	.
.	.
V_0V_k	current V_0V_k

In these (cycle of) currents the first positive current following some negative current or other is searched (there are at least one positive and one negative current). The corresponding branch, V_0V_l say, is put in class C_{pos}. All successive branches V_0V_{l+1}, V_0V_{l+2}, . . . , V_0V_s that carry a positive current are put in class C_{pos}. The branch V_0V_{s+1}, carrying a negative current, is put in class C_{neg}, while all successive branches V_0V_{s+2}, V_0V_{s+3}. . . . , V_0V_t that carry a negative current are put in class C_{neg}. All indices of the second vertex are taken mod(k). Then the following theorem can be formulated:

All branches V_0V_1, ... V_0V_k are belonging either to C_{pos} or C_{neg} if the network is planar.

Now a begin can be made with building up a Bouwkamp code of a dissection originating from a net after having placed an electromotive force in one of the branches of the net. This branch is called the accumulator branch. Starting from the accumulator branch we follow the current in the positive direction. If in the case of the reference net the third row of R is used, we find that the accumulator branch 6,5 is carrying a reduced ordered current equal to 32. Then one of the vertices V_a of the accumulator branch will be passed. In the reference net this is vertex 5. The next step is to consider the left-cyclic-ordered adjacent branches of V_a. In particular the branches of C_{pos} of V_a are considered. The sequence in

which they occur in C_{pos} is just the way in which the corresponding squares have to be drawn. Notice that the reduced currents of the left-cyclic-ordered adjacent branches of V_a are given by the elements: reduced ordered current [address $[V_a]$], ..., reduced ordered current [address $[V_a+1]-1$].

In the reference net one has:

branch	C_{pos}	reduced ordered current
56		
53	53	15
54	54	8
51	51	9

The Bouwkamp code can be started with the reduced ordered currents belonging to the branches of C_{pos} of V_a. In the example a part of the Bouwkamp code is as follows: (15, 8, 9).

The next step is to find the vertex with which the process has to be continued. To this end a vector "contour" is defined. It is assumed that the vertex V_a has a level zero. After having drawn the Bouwkamp code so far, the vector contour contains the levels of the adjacent vertices of V_a belonging to branches of C_{pos} of V_a. The level of V_i equals the level of V_j plus the absolute value of the reduced current of branch V_jV_i. The adjacent vertices corresponding to the elements of contour are forming the vector "vertex contour". In the example of the reference net one has:

contour [1] = 15	vertex contour [1] = 3
contour [2] = 8	vertex contour [2] = 4
contour [3] = 9	vertex contour [3] = 1

The next step is to find the minimum of contour $[i]$. In the case of more than one element equal to the minimum, the element with the smallest subscript is considered first. Let this element be contour $[q]$. In the example the minimum of contour equals 8, while the corresponding vertex, namely, vertex contour [2] equals 4.

The class C_{pos} of vertex contour $[q]$ determines which squares can be drawn next. In the example one has:

branch	C_{pos}	reduced ordered current
43	43	7
41	41	1
45		

The Bouwkamp code can be extended with the reduced ordered currents of the branches of the class C_{pos} of vertex contour $[q]$. In the example one has: (15,8,9)(7,1). A right parenthesis will be added only if contour $[q+1] \neq$ contour $[q]$.

Then the vectors contour and vertex contour are updated. The vector contour is determined as follows: The element contour [q] is replaced by the levels of the adjacent vertices of vertex contour [q]. The element vertex contour [q] is replaced by the just-mentioned adjacent vertices of vertex contour [q]. The example therefore gives:

contour [1] = 15	vertex contour [1] = 3
contour [2] = 8 + 7 = 15	vertex contour [2] = 3
contour [3] = 8 + 1 = 9	vertex contour [3] = 1
contour [4] = 9	vertex contour [4] = 1

The following step is the condensation of the vectors contour and vertex contour. If for any i contour [i] = contour [$i+1$] and vertex contour [i] = vertex contour [$i+1$], then the elements contour [$i+1$] and vertex contour [$i+1$] are omitted. The new vectors contour and vertex contour then have one element less than the old vectors. This process is repeated until no more elements can be omitted. Then the minimum of contour is searched again, and so on. The whole process may be stopped when both vectors contour and vertex contour have only one element. The element contour [1] will then be equal to contour [1] = (complexity—accumulator current) ÷ RF while vertex contour [1] will be the other vertex of the accumulator branch.

The example of the reference net is running through the following steps.

After condensation one has

contour [1] = 15	vertex contour [1] = 3
contour [2] = 9	vertex contour [2] = 1

The minimum of contour is contour [2] and is equal to 9, while vertex contour [2] = 1. The left-cyclic-ordered adjacent branches of vertex 1 and their currents are:

branch	C_{pos}	reduced ordered current
12	12	10
15		
14		

The Bouwkamp code can be extended to (15,8,9)(7,1)(10). Updating contour and vertex contour gives

contour [1] = 15	vertex contour [1] = 3
contour [2] = 19	vertex contour [2] = 2

There is no condensation necessary. The minimum of contour is contour [1] and is equal to 15; vertex contour [1] = 3. The left-cyclic-ordered adjacent branches of vertex 3 and their currents are:

branch	C_{pos}	reduced ordered current
32	36	18
34	32	4
35		
36		

The Bouwkamp code can be extended to (15,8,9)(7,1)(10)(18,4). After updating and condensation one obtains

contour [1] = 33 vertex contour [1] = 6
contour [2] = 19 vertex contour [2] = 2

The minimum of contour is now contour [2] and is equal to 19, while vertex contour [2] = 2. The left-cyclic-ordered adjacent branches of vertex 2 and their currents become:

branch	C_{pos}	reduced ordered current
21	26	14
23		
26		

The Bouwkamp code can be extended to (15,8,9)(7,1)(10)(18,4)(14). After updating one has:

contour [1] = 33 vertex contour [1] = 6
contour [2] = 33 vertex contour [2] = 6

After condensation one obtains:

contour [1] = 33 vertex contour [2] = 6

Now the process is ready.

It is clear that another Bouwkamp code of the same dissection would have been obtained if, instead of the left-cyclic direction, the right-cyclic direction was chosen. Furthermore other Bouwkamp codes are obtained by starting with the other vertex of the accumulator branch either using the left or the right-cyclic direction; it is then necessary to use C_{neg} instead of C_{pos}.

If we want to code the dissection with the restriction given in Bouwkamp's paper [2]), that the larger side is horizontal and that the left upper corner element should not be smaller than the three remaining corner elements, it is then sometimes necessary to consider the dual net also. This is so if the complexity is greater than twice the current through the accumulator branch in the original net.

The currents of the dual net can be obtained as follows. Assuming a current in the accumulator branch of the dual net equal to the current in the corresponding accumulator branch in the original net minus the complexity, the

current in branch: branch 1[i], branch 2[i] of the original net is equal to the current in branch: branchdual 1[i], branchdual 2[i] of the dual net. Then the same procedure as described before can be used for obtaining Bouwkamp codes corresponding to the dual net.

The four corner elements can be obtained from the first and the last element of the set C_{pos} of the accumulator vertex V_a and from the first and the last element of the set C_{neg} of the other vertex of the accumulator branch. The four corner elements are denoted by former first, next first, former second and next second, respectively. If in a Bouwkamp code corresponding to the original or dual net two consecutive elements are equal, a **Boolean** variable: trivial imperfection is assigned **true**. The complete procedure is given in programme II. In this programme it is assumed that two new procedures are added to the ALGOL language, namely stop and punch (E). Depending on the result of expression E, the **procedure** punch (E) punches the result in the next free columns of the punch card. The **procedure** stop stops the computer.

CHAPTER 6

SOME RESULTS

From the wheel S_8 we obtained the sets $S_9, S_{10}, \ldots, S_{19}$ using the electronic computers PASCAL and STEVIN of the Philips computing centre. The programmes were so arranged that the generated and identified nets could be written on magnetic tape, punched on cards or punched on paper tape. For orders up to and including 16, the list of identification numbers was stored in the core memory while for higher orders it was stored on the magnetic drum. In the latter case we applied the following sorting method.

The drum has a capacity of 16384 words of 42 bits. The identification number needs at least two words for orders higher than 16. We can therefore store 8192 identification numbers on the magnetic drum. Let the identification number I consist of the bits $a_{45}, \ldots, a_3, a_2, a_1$; then four numbers are formed, namely, $\sum_{k=1}^{13} a_k 2^{k-1}$, $\sum_{k=1}^{13} a_{k+13} 2^{k-1}$, $\sum_{k=1}^{13} a_{k+26} 2^{k-1}$ and $\sum_{k=1}^{6} a_{k+39} 2^{k-1}$. Let $\frac{1}{2}A$ be the sum (modulo 2^{13}) of these four numbers. If locations A and $A+1$ of the magnetic drum contain zeros, then the identification number is new and is stored in these two locations. If the locations A and $A+1$ contain non-zero numbers, it is investigated whether the contents of A and $A+1$ is equal to I. If so, the net represented by I was already found. If not, the contents of the next two locations, namely, $A+2$ and $A+3$, are compared with I, and so on. If I is not found on the drum, I is stored in the first two locations containing zeros and following upon the locations A and $A+1$.

We found that the sets S_k* have the following number of elements. The computing time on PASCAL is also given below:

k	number of c-nets except for duals	computing time
8	1	
9	1	
10	2	
11	2	
12	8	
13	11	
14	37	
15	79	5 minutes
16	249	15 minutes
17	671	50 minutes
18	2182	2·5 hours
19	6692	7 hours

The codes and identification numbers of the nets of S_{15}* and S_{16}* were punched on cards. The cards were sorted with respect to the identification number and were listed on one of the available printers. In table I we give a fotographic copy of this output. The format is as follows: code of the net, number of choices, selfdual (1 means selfdual), identification number.

As soon as the nets were available, we investigated whether perfect or imperfect simple squared squares could be obtained from these nets. To that end we used programme II of the determination of Bouwkamp codes. However, the code was only punched if the reduced sides were equal. The codes were sorted according to increasing reduced sides.

The nets of orders 20 were generated. They were kept in the computer. The nets having a complexity satisfying the relation $C = 2kA^2$, where k and A are integers, $A \geqslant 15$, were punched on paper tape after they had been identified and had passed the **procedure** new net test. This programme took 30 hours of computing time.

The reason why we considered only complexities equal to $2kA^2$ with $A \geqslant 15$ was the following. We wanted to know whether perfect squared squares of order 19 exist. Now the largest element of a perfect squaring is greater than 18. If a perfect squaring exists then its reduced side is certainly greater than 19. Hence by taking $A \geqslant 15$ we have not missed any simple perfect squaring of order 19. On the other hand we did not want too many nets as computer output so we chose A not too small. From experience of low-order squared squares we expect that no other simple imperfect squared squares of order 19 exist than those contained in table II.

From these nets the Bouwkamp codes of the squared squares were punched on cards. In table II we give a fotographic reproduction of the codes of the imperfect squared squares of orders up to and including 19. The format is as follows: C = complexity, S reduced horizontal side * reduced vertical side, * or blank (* means imperfect, blank means perfect), RF reduction factor, Bouwkamp code, number of choices.

At last we give all Bouwkamp codes of a few nets (of orders 10, 20, 21 and 22) as typed by the on-line typewriter of PASCAL. We did not use the on-line printer because only 92 print wheels are available which is too few for the Bouwkamp codes. The same format is used as above. There are only two extra characters, namely, trivial imperfection and the number of zero currents. A reproduction of this output is given in table III.

REFERENCES

1) R. L. Brooks, C. A. B. Smith, A. H. Stone and W. T. Tutte, Duke math. J. **7**, 312-340, 1940.

2) C. J. Bouwkamp, Proc. Acad. Sci. Amst. **49**, 1176-1188, 1946; **50**, 58-78, 1296-1299, 1947 (= Indag. math. **8**, 724-736, 1946; **9**, 43-63, 622-625, 1947).

3) M. Dehn, Math. Ann. **57**, 314-332, 1903.

4) Z. Moroń, Przeglad Mat. Fiz. **3**, 152-153, 1925.

[5]) C. J. Bouwkamp, A. J. W. Duijvestijn and P. Medema, Catalogue of simple squared rectangles of orders nine through fourteen and their elements, Department of Mathematics, Technische Hogeschool, Eindhoven (Netherlands), May 1960, 50 pp.

[6]) T. H. Willcocks, Fairy Chess Rev. 7, Aug/Oct. 1948.

[7]) T. H. Willcocks, private communication, July 1961.

[8]) R. C. Ellis, private communication, November 1959-October 1960; The perfectable rectangles of order 14, and List of 16-wire c-nets, as yet unpublished manuscripts.

[9]) C. J. Bouwkamp, A. J. W. Duijvestijn and P. Medema, Tables relating to simple squared rectangles of orders nine through fifteen, Department of Mathematics and Mechanics, Technische Hogeschool, Eindhoven (Netherlands), August 1960, 360 pp.

[10]) J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, P. Naur, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden and M. Woodger, Numerische Math. **2**, 106-136, 1960.

[11]) W. T. Tutte, Proc. Acad. Sci. Amst. **64 A** (= Indag. Math. **23**), 441-455, 1961.

[12]) W. Cauer, Theorie der linearen Wechselstromschaltungen, Akademie-Verlag Berlin, 2nd. ed. (1954), pp. 56-91.

[13]) G. Kron, Tensor analysis of networks, John Wiley & Sons, Inc., New York (1939).

TABLE I

Codes and identification numbers of S_{15}* and S_{16}*

617601271023720734707467045640165105325021520354300	3	0	3777423
317301271023720134101521025620632603643046540514500	2	0	7566764
617601271023720347304574056750465401641032130143100	1	0	7775522
617601271023720347304574056750465404324026420216200	2	0	7775530
617601271023720347304574056750465401641043240214200	2	0	7777422
617860127102372034730458740568504654064326021600	0	0	767524302
628560218202672027320765470437404584014810123410 0	1	0	767524501
617860127102372034730458740568504654064360321630 0	1	0	767544122
182102832038430487540576508167803453052350125610 0	0	0	767560203
284201382103483024520163104367404754061260257620 0	0	0	1365564302
856802186202672027320765470437404584014810123410 0	0	0	1473447250
185610162102672027320765470437404584081480123410 0	0	0	1473465411
618601278102372034730457405687504654016431032130 0	1	0	1476425530
618601278102372034730458740568504654016431032130 0	1	0	1522533622
418401281023482014510162102672073270375430157610 0	1	0	1563526700
856802186202672027320376543045840148102342041240 0	0	0	1563564302
718701281023820358304685406786047640453403217430 0	1	0	1573253401
718701281023820358304685406786047640453240217420 0	0	0	1573464501
718701281023820354830468406786045764017510532150 0	1	0	1577050322
718701281023820358304685406786047640174045321400	0	0	1577051203
617860127102372034873045840568504654016431032130 0	2	0	1625215552
618601278102372034730458740568504654064326021600	1	0	1661633310
182102832038430475404874057650816780234562061260 0	0	0	1665113630
284201382103483024520163104367404754061256057650 0	0	0	1665407350
617860127102372034873045840568504654064326021600	0	0	1665413330
182102832038430475404874057650816780123451056150 0	0	0	1665423314
618601278102372034730458740568504654016410432140 0	0	0	1665423341
182102832038430475404874057650816780234520561250 0	0	0	1665423611
287202348204584056785016510153103213035430612760 0	0	0	1665425511
418340128102382014510162103267303754305765061560 0	0	0	1666130370
1821028320384304754048740576508167803456306123600	0	0	1666131134
617860127102387203483045840568504654016431032130 0	0	0	1666431131
856806186016210267202732037654304584081480123410 0	0	0	1667030351
182102832038430475404874057650816780345205612600	0	0	1667031115
418408348012810238201451016210326730375430157610 0	1	0	1667031213
856802186202672027320376543045840148101231034130 0	1	0	1706222656
185610162102672027320765705437504584081480123410 0	1	0	1706224556
856802186202672027320765705437504584014810123410 0	1	0	1706424536
485402145202563201367107417084780586506876012310 0	1	0	1706431217
712870234820458406786085680165101532103543017600 0	0	0	1715650611
617860127102387203483045840568504654064326021600 0	0	0	1725530701
418403283081380248201451016310236742047540157610 0	0	0	1725550611
182102832038430487540576508167802345202562061260 0	0	0	1741764302
418401328102482014510163102367207427047540157610 0	1	0	1741764501
712870234820458405678501651015310321303543017610 0	0	0	1745314710
856802182086280267202732037654304584014810123410 0	0	0	1745315501
182102832038430487540576508167801234101451056150 0	0	0	1745344522
185610162102672027320765430454840814802342041240 0	0	0	1745344541
182102832038430475404874057650816780456406123460 0	0	0	1745350611
287203483082380458405678501651015321035430612760 0	0	0	1745360603

617860127102387203483045840568504654043240216420	0	0	1746622701
617860127102387203483045840568504654043240216420	0	0	1746660303
148521026320256203673074137047840586506876012310	1	0	1746222651
185610162102672027320376543045840814801231034130	0	0	1746224532
418401281023482014510162102673203754305765061560	1	0	1746224551
182102832038430475404874057650167107817012345610	2	0	1746324541
182102832038430475404874057650816780123103456130	0	0	1746424531
128108718023482045840567850165101532103543017610	0	0	1746627003
182102832038430475404874057650816780123410456140	0	0	1747212341
284208238034830512502452016321036743047540157610	1	0	1747222603
182102832038430475404874057650816780234204561240	0	0	1747230303
284203823021320348302452016310436740475406125760	1	0	1747252023
418401281023420482401451016210267320375430157610	2	0	1747254013
617860127102387203483045840568504654064360321630	1	0	1747260107
618601281023820348304584056850476540274320167210	2	0	1761526700
718701281023820354304834046840678604576401753210	0	0	1765113304
718701281023820358304685406786047640453250217452	0	0	1765422701
718701281023820358306856054650678604764017453210	1	0	1766122524
718701281023820358304685406786047640174510532150	0	0	1767012341
718701281023820354830468406786045764017531032130	0	0	1767022505
718701281023820358304685406786047640174531032130	0	0	1767030303
718701281023820358304685406786047640745053217500	0	0	1767112214
718701281023820354830468406786045764075370321730	0	0	1767112421
718701281023820358304685406786047640745370321730	0	0	1767114411
718701281023820354830468406786045740764701753210	0	0	1767142045
718701281023820358304685406786047640745327021720	0	0	1767150015
718701281023820354830468406786045764075327021720	0	0	1767150023
821808328084380854808658087680817801234104567140	1	0	1776425003
821808328084380854808658087680817801231034567130	1	0	1777050023
182102832038430475404874057650816780345630623606126	0	0	767350254
182102832038430475404874057650816780234520512505615	2	0	767524612
182102832038430475404874057650816780345305123505615	1	0	767560216
182102832038430475404874057650816780345305235056125	1	0	767560423
718701281023820358304685406786047640174104534032143	0	0	777650130
718701281023820358304685406786047640174104532402142	0	0	777660411
182102832038430475404874057650816780234520612602562	1	0	1532771422
418401281023482014510162102672073270754704374015761	2	0	1563566700
718701281023820358304685406786047640453403743032173	1	0	1577253401
718701281023820358304685406786047640453240274202172	0	0	1577464501
856806186016210267202732037654304584081480234204124	1	0	1633673041
856802182086280267202732076547043740458401481012341	0	0	1663537210
284201382103483024520163104367404754061260576502565	1	0	1663566160
182102832038430475404874057650816780456406234606126	0	0	1663566430
856802182086280267202732076570543750458401481012341	0	0	1665361416
182102832038430475404874057650816780345306123603563	0	0	1665361443
617860127102372034873045840568504654043240264202162	1	0	1666527430
182102832038430475404874057650816780234206126024562	0	0	1666535130
718701281023820358304685406786047640453250245202174	0	0	1675631310
718701281023820358304685406786047640453403174303213	0	0	1675651130
718701281023820358304685406786047640174103213014531	0	0	1677424530
718701281023820358304685406786047640174105321501451	0	0	1677425122
718701281023820354830468406786045740764701751053215	0	0	1677431211
856802186202672027320765705437504584014810123103413	2	0	1706424537
182102832038430475404874057650816780345305623506126	0	0	1716430635
185610162102672027320765704374075470458408148012341	0	0	1724743462
182102832038430475404874057650816780123105615013451	0	0	1724745452
618601278102372034730458740568504654016410321301431	1	0	1726522654
182102832038430475404874057650167107817023452056125	1	0	1726522662
182102832038430475404874057650816780123104564061346	1	0	1726524562
182102832038430475404874057650167107817023456206126	1	0	1726530362
182102832038430475404874057650816780123103456306136	0	0	1726624552
182102832038430475404874057650816780123103453056135	1	0	1726630352
718701281023820354304834046840678604576401751053215	1	0	1734613650
718701281023820358306856054650678604764045320217420	1	0	1736414570
718701281023820358304685406786047640174105325021452	0	0	1736424562
718701281023820358306856054650678604764017410453214	1	0	1736430651
182102832038430487540576508167803453052350256206126	0	0	1743356250
182102832038430475404874057650816780456406340606126	0	0	1743666411
856806186016210267202732076570543750458401481012341	0	0	1746224653
856806186016210267202732037654304584081480123103413	1	0	1746324562
182102832038430475404874057650167107817012341045614	2	0	1746364522
182102832038430475404874057650816780123410561501451	0	0	1746627114
182102832038430475404874057650167107817012310345613	0	0	1746627411
287203483082380458405678501651015310321303543061276	1	0	1746633211
182102832038430475404874057650816780234205612502452	1	0	1746635111
617860127102387203483045840568504654043240264202162	0	0	1747232701
182102832038430475404874057650816780123104564134100	0	0	1747252431
718701281023820354304834046840678604576407532702172	0	0	1764751130
718701381034830458405685067860576502543202752017231	1	0	1765526700

7187012810238203583068560546506786047640174531032130 0	1	0	1766224552
7187012810238203583068560546506786047640745327021720 0	0	0	1766522630
1821028320384304754048740576508167802342045640612460 0	0	0	1766524530
7187012810238203583068560546506786047640174510532150 0	0	0	1766525114
7187012810238203583046854067860476407457053175032130 0	0	0	1766525122
7187012810238203543048340468406786047640753703217300 0	0	0	1766527104
7187012810238203543048340468406786047640175310321300 0	0	0	1766530330
7187012810238203583046854067860476407401745105325021520 0	0	0	1766560431
7187012810238203583046854067860476407457053250217520 0	0	0	1767117401
7187012810238203543048340468406786047457064701753210 0	0	0	1767132304
71870128102382035830685605465067860476407457053217500 1	1	0	1767152124
7187012810238203583046854067860476407457051750532150 0	0	0	1767152222
7187012810238203543048406786047640753703173032130 0	0	0	1767153005
7187012810238203583046854067860476405325027452021720 0	0	0	1767426501
7187012810238203583046854067860476407453703173032130 0	0	0	1767447003
7187012810238203583046854067860476407401745103213015310 0	0	0	1767450131
7187012810238203583046854067860476407457021720753270 0	0	0	1767524302
7187012810238203548304684067860457407647075327021720 0	1	0	1767524501
7187012810238203583046854067860476407457032173075370 0	1	0	1767544122
7187012810238203583046854067860476407453702172073270 0	0	0	1767560203
8218083280843808548086580876808178012310456406713460 0	1	0	1776424530
82180832808438085480865808768081780123104564067134700 0	1	0	1776431211
8218083280843808548086580876808178012310456406713600 0	0	0	1777424302
8218083280843808548086580876808178012310345056713500 0	1	0	1777424501
8218083280843808548086580876808178012341045674071470 0	1	0	1777444122
8218083280843808548086580876808178012310345673071370 0	0	0	1777460203
7197012891023820354830468406798604576401751015321000 1	1	0	77706406022
719701289102382035830468540679860467601751015321000 1	1	0	77706424003
1892102932039430758704985405765081678023456201261000 0	0	0	331341252301
1892102932039430758704985405765081678012341014561000 1	1	0	361641212322
7189701281023820354830469840679604576403753017321000 0	0	0	363661420203
7189701281023820354830469840679604576402753201721000 0	0	0	363665000423
7197012891023820354830468406798604576403753017321000 1	0	0	365651404114
1921029320394307587049854057650916789012345101561000 1	1	1	365651404122
7197012891023820358304685406798604760457401753210 0	0	0	365651420203
1921029320394307587049854057650916789034530123561000 0	0	0	365655000423
7197012891023820358304685406798604760374530173210 1	0	0	371631404122
1921029320394307587049854057650916789023452012561000 0	0	0	371631420203
7197012891023820354830468406798604574047640173210 0	0	0	371635000423
7197012910238920354303483046984067960457640173210 0	0	0	474257126004
8129802349204694067896017651015310132103564301871000 1	0	0	474316526004
7189701281023820358305698505465067960476401745321000 0	0	0	475116152024
2982092345905695067896017610164310465402832071387000 0	1	0	475215116420
1921029320394530486540498406876091789056750123571000 0	0	0	475215152024
7189701281023982035930469540679606476024532017421000 0	0	0	475217016041
7197012891023820354830468406798604576401753101321000 0	0	0	475217032005
1892102932039843075870485405765081678023456201261000 0	1	0	475247026003
1921029320394307587049854057650916789034563012361000 0	0	0	475304546041
2982092349046945078965017510153210456403543071287000 0	1	0	475306132005
1892102932039843075870485405765081678034563012361000 0	0	0	475306426003
2192029320398430485405865068760178910123101345671000 1	0	0	475416122424
7189701281023820358304698540679606476017410145321000 0	0	0	475605112222
3983082380934590569506789601761016432104654071287000 0	0	0	475605113005
7189701281023982035493046940679604576401751015321000 0	0	0	475605407003
7129670173103783038430658760485405695025920134521000 0	0	0	475606122045
1967101732103783038430658760485405695091259023454200 0	0	0	475606422023
7189701281023982035930469540679606476017410145321000 0	0	0	475607012023
1897108128023982034930459405695067960172102765432000 0	1	0	575413110214
8197801291023920354930469406796045840487640185321000 0	0	0	575413140045
1921029320394306986049654056750687609178901234571000 0	0	0	575416101045
1897108128023982034930459405695067960176210265432000 0	0	0	575601504122
1921029320394306986049654056875091789023452012571000 0	0	0	575601520105
1897108128023982034930459405695067960176521025432000 0	0	0	575601520203
2982013921034930469405796507897075870128510315643000 0	0	0	575605100145
1897108128023982034930459405695067960176542102432000 0	1	0	575605100415
1921029320394306986049654056875091789023457201271000 0	0	0	575605100423
1921029320398430758704854057650916789034563012361000 0	0	0	630345252301
5285021482025632013671017410479840589650697601231000 1	1	0	630522517300
1892102932039430758704985405765081678034563012361000 1	0	0	630523252501
1892102932039843048754057650816780345302356201261000 2	1	0	630524507700
1892102932039430758704985405765081678012310134561000 0	0	0	644545130701
7189701281023820354303498304694067960457640175321000 0	0	0	660544315501
1921029832038430758970485405765091679034563012361000 0	1	0	660545212701
1921029320394530486540498406876091789012356101671000 0	0	0	664645412211
1892102932039843048754057650816780345302352012561000 0	0	0	665240572021
6189601281023982034930459405695047654027432016721000 0	0	0	670322336002
9569067960219720276820283203865430459401491012341000 1	0	0	670322536001
1892102932039430758704854057650816780123101345610 0	0	0	670433252022
1921029320394530486540498406876091789035673012371000 0	0	0	670526114411
9567902197202768202832054865038430459401491012341000 0	0	0	670611664003

19567101721027682028320548650384304594091490123410	0	0	670615244043
19210298320384304875405765091678901231013451015610	0	1	670621316410
61789601271023720348730459840569504654026432016210	0	0	670621346042
19210298320384307597048954057650916790123103134561	0	1	670621352022
19210298320384304897540576509167901231034530135610	0	1	670621352041
18921029320394304987540576508167801231013451015610	0	0	670621354012
96790712970173103783038430478654056950259201345210	0	0	670621546022
19210293203984304754047405765091678902342012456100	0	0	670621564003
23982034593056950678960176103243016421046540712870	0	0	670622326042
19210298320384307587048540576509167890123101345610	0	0	670623116014
71296701731037830384304876540569502592023452013210	0	0	670623116041
71970128910238203543034983046940679604576401753210	0	0	670623216012
18921029320394307587049854057650816780234520125610	0	0	670624322414
18971081280238203498304594056950679601765410143210	0	1	670624322441
71897012810238203549830469406796045764027552017210	0	0	670625144016
18921029320394307587049854057650816780345301235610	0	0	670625144043
71897012810239820359304695406796064760345031743210	1	0	670625160007
18971081280238203498304594056950679601762102654320	0	1	670704306422
18971081280238203498304594056950679601765101543210	0	1	670704314411
19210293203945304865404984068760917890235620126710	0	0	670704606212
71970129102389203548304684067986045740476401753210	0	0	670704642015
19210293203945304865404984068760917890235672012710	0	0	670704644013
18921029320394307598704954057650816780345660123610	0	0	670706206016
19210293203984307587048540576509167890345301235610	0	0	670706206043
19210298320384307587048540576509167890345630123610	0	0	670706222007
71970128910238203548304684067960457404764017532100	1	1	670706406023
19210293203943075870498540576509167890123101345610	0	0	671302236003
71897012810238204984035483094690679604576401753210	0	0	671402347042
71970128910238203549830469406796045764017531013210	0	0	671422126422
18910192102932039430758704985405765081678012345610	0	0	671422126441
18921029320398430758704854057650816780234012456100	0	1	671422134122
71970289201291082380354830468406798604576401753210	0	1	671422226142
19210293203984307587048540576509167907897012345610	1	0	671424107422
18921029320394307598704954057650816780123410145610	0	0	671600335003
61796012710238972034830459840569504654016410143210	0	0	671600364043
18921029320394307587049854057650816780123451015610	0	1	671601217003
39630215932023620267420147851058950697607987012410	0	0	671601246043
71970129102389203583034698540679606476017453101321	0	0	671601252023
71970129102389203548304698406796045764017531013210	0	0	671601254013
71897012810238203548304698406796045764017531013210	0	0	671602226016
71897012810238203583046985406796064760174531013210	0	0	671603016023
19310423940495405865059850686760917890245620132671	0	0	671604206162
18921029320394307598704954057650816780456401234610	0	1	671604207016
19210293203943075870498404854057650916789012345610	0	0	671604207043
71970128910238203549830469406796045764017510153210	0	0	671604223007
61960128910237820347304598740569504654016410143210	0	1	671604242063
71970128910238203598304695406796064760174014532100	0	0	671604260017
71970129102389203548304698406796045764017510153210	1	0	671605006216
19210293203943047854049740586509168790345301235610	0	0	671605006423
71970129102389203583046985406796064760174014532100	0	0	671605044017
95679021972027682028320586503854304594014910123410	0	1	705141155501
61896012810238203498304594056950476540274320167210	1	0	720261526700
71296701731037830384304876540569502592013421024520	0	0	721060371501
19210293203984307587048540576509167890123101345610	0	0	721141252701
71897012810238203549830469406796045764017510153210	1	0	721240364522
71970128910238203583046985406796064760174531013210	0	0	721240565203
18921029320394304754049874057650816780234520125610	0	0	721240665103
71970129102389203548304684067986045764017531013210	0	1	721300670303
18921029320394307587049854057650816780234012456100	0	0	721301454303
81280298202349204694067896017651015321035643018710	0	1	721412425603
19210298320384304754048740576509167890123101345610	0	0	721442222714
19210293203984304754048740576509167890345301235610	0	0	721442231303
71970128910238203583046985406796064760174014532100	0	0	721443024703
97890718701281023820349830459405695067960176543210	1	0	725256020423
18910192102932039843047540487405765081678012345610	1	0	740522515114
71897012810238203598305695054650679604764017453210	1	0	741501424703
95679071970172102768202832038654304594014910123410	0	0	741502230523
19210293203983038430475404874057650916789012345610	1	0	741502424523
94590529502392039430612560173210378430486540168710	2	1	741502444163
19210298320384304754048740576509167890345301235610	1	0	741503050217
71870789701281023820354983046940679604576401753210	0	1	745542120145
71297023489204584056798501651015310321035430176100	0	1	745543010213
19210293203945305865048540498406876091789012356710	0	0	750537000423
71970128910238203549830469406796045764027532017210	0	0	750725100216
71897012810239820354303049304694067960457640175321	0	0	751301262007
71870789701281023982035493049406796045764017532100	0	0	751522120145
18921029320394305985058750495405765081678012345610	0	0	751522120423
31293039430495405865059850687601782102892013456710	1	0	751523002423
23982034593056950789706796017610164321046540712870	0	0	751523010115
29820459409234905695067896017610164310546507132870	1	0	751524101423

7189701281023982035493046940679604574047640175321 00	0	0	751526001017
1921029832038430487540576509167890123103453013561 00	0	1	760262332041
7197012910239203548930468406798604574047640175321 00	0	0	760473404411
1291019710234892045840567985016510215320354301761 00	0	1	760473440023
3963021593202367202742014785105895069760798701241 00	0	1	760661510211
1921029320394304784048540497405865091687901234561 00	0	0	760665100415
9567902192029720276820283203865430459401491012341 00	0	0	760665400115
7189701281039830238209354904694067960457640175321 00	0	1	761060547022
1891019210293203984307587048540576508167801234561 00	0	0	761060564212
7197012891023820498403548309469067960457640175321 00	0	1	761240370023
9289021820293203943075870498540576501678101234561 00	0	0	761240626114
6196012789102372034730458740569850465401641014321 00	0	1	761240626122
1921029320398303843075870485405765091678901234561 00	0	0	761240662015
1897108128023820349830459405695067960476540174321 00	1	1	761241217003
2192029320394304985405865068760178910123101345671 00	0	0	761302416023
7128970234820459840679605695016510153210354301761 00	1	1	761440613211
7197012910238920354830698604684096790457640175321 00	0	1	761442115023
1892102932039430758704984048540576508167801234561 00	0	0	761442212232
1892102932039430475404984048740576508167801234561 00	1	0	761442222216
1921029320394304785404974058650168710917901234561 00	1	0	761442223007
1921029832038430758704854057650916790789701234561 00	0	0	761442407023
7189701281023982035930569505465067960476401745321 00	0	0	761442414131
5195032930139102459201561017310237842048654016871 00	0	1	761442424115
1921029832038430589059750485405765091679012345610 0	0	0	761462120145
7197012910239203548930468406798604574047531013210 0	0	1	761602414033
1921029832038430758704854057650916789034530123561 00	0	1	761644201017
7197012891023820354983046940679604574047640175321 00	0	1	761645000217
9289021820293203984307587048540576501678101234561 00	1	1	770522502222
1921029820283203843075870485405765091678901234561 00	1	0	770522540023
1921029832038430758704854057650167810918901234561 00	0	1	770621310212
1921029320945903953048654049840687609178901235671 00	0	1	770624301043
1892102932039830384307587048540576508167801234561 00	1	0	770625100216
9789071870128102398203493045940569506796017654321 00	0	0	771420341043
1921029320394306986049640465405687509178901234571 00	0	1	771422220143
1921029320394307587059850495405765091678901234561 00	0	1	771432005043
1897108128023982034930459405695067960576543210 0	0	1	771604201017
1897108128023982034930459405695067960476432100	1	1	771605000217
1891019710812802398203493045940569506796017654321 00	0	1	775412003045
1981012910239203493045940569506796078970218765432 00	2	1	777402020423

TABLE II

List of simple imperfect squared squares of orders up to and including 19

C= 1058	S	23*	23*	RF 23	(12,11)(1,3,7)(11,2)(5)(2,5)(4,1)(3)0
C= 3042	S	39*	39*	RF 39	(20,8,11)(5,3)(2,12)(7)(19,8)(5,7)(11,2)(9)0
C= 3042	S	39*	39*	RF 39	(20,19)(1,3,8,7)(19,2)(5)(2,5)(12,1)(3)(8)0
C= 3362	S	41*	41*	RF 41	(23,18)(7,11)(18,3,2)(1,5,3)(4)(2,1)(12)(11)0
C= 4608	S	48*	48*	RF 48	(28,20)(7,5,8)(2,3)(9)(20,8)(11)(12,5)(2,9)(7)0
C= 4608	S	48*	48*	RF 48	(28,20)(8,12)(20,9,7)(5,7)(2,5)(11)(3,2)(9)(8)0
C= 4608	S	48*	48*	RF 48	(28,20)(11,9)(20,8)(2,7)(8,5)(5,3)(12)(2,9)(7)0
C= 5202	S	51*	51*	RF 51	(22,14,15)(13,1)(16)(13,9)(9,4)(4,5)(20)(16,1)(15)0
C= 5408	S	52*	52*	RF 52	(28,24)(7,9,8)(24,4)(1,6)(5)(1,7)(4,6)(15)(13)0
C=10890	S	11*	11*	RF495	(4,3,4)(1,2)(3,2)(1,3)(1,2)(1,2)(4)(1,4)(3)1
C= 9248	S	34*	34*	RF136	(19,15)(4,5,6)(15,7,1)(5,1)(7)(1,4)(8)(1,6)(5)0
C= 7688	S	62*	62*	RF 62	(33,29)(4,5,20)(29,7,1)(6)(13)(7,13)(9,4)(1,6)(5)0
C= 8192	S	64*	64*	RF 64	(36,28)(9,8,11)(28,8)(3,5)(7,2)(5)(2,9)(7)(20)(16)0
C= 8192	S	64*	64*	RF 64	(36,28)(9,11,8)(28,8)(3,5)(7,2)(5,9,2)(7)(20)(16)0
C= 8450	S	65*	65*	RF 65	(33,32)(1,3,8,20)(32,2)(5)(13)(2,5,13)(12,1)(3)(8)0
C= 8450	S	65*	65*	RF 65	(36,29)(16,13)(14,13,9)(4,9)(4,20,1)(5)(1,16)(15)(14)0
C= 8978	S	67*	67*	RF 67	(39,28)(12,7,9)(5,2)(1))(28,8,3)(2,7,8)(5)(20)(19)0
C= 9248	S	68*	68*	RF 68	(25,20,23)(5,12,3)(26)(23,7)(19)(20,3)(7,19)(17,5)(12)0
C= 9248	S	68*	68*	RF 68	(36,32)(4,6,7,15)(32,8)(5,1)(8)(1,4)(9)(6,21)(15)0
C= 9248	S	68*	68*	RF 68	(36,32)(8,9,15)(32,4)(11,1)(10)(4,11)(17,8)(1,10)(9)0
C= 9522	S	69*	69*	RF 69	(39,30)(7,12,11)(2,5)(30,11)(3,8)(8,7,2)(5)(20)(19)0
C= 9800	S	70*	70*	RF 70	(38,32)(6,9,17)(32,8,4)(1,8)(5)(7,1)(6)(4,21)(17)0
C= 9800	S	70*	70*	RF 70	(39,31)(5,7,19)(3,2)(1,8)(31,12)(5,3)(2,1)(20)(19)0
C= 9800	S	70*	70*	RF 70	(41,29)(11,18)(2,5,4)(29,11,1)(3)(1,3)(7,2)(23)(18)0

C=15488	S	44*	44*	RF176	(24,20)(8,12)(20,4)(7,5)(4,3,5)(2,3)(9)(1,2)(8)(7)0
C=12800	S	80*	80*	RF 80	(44,36)(8,28)(36,16)(9,7)(5,7,16)(2,5)(11)(3,2)(9)(8)0
C=12800	S	80*	80*	RF 80	(44,36)(11,9,16)(36,8)(2,7)(8,5)(5,3)(28)(2,9)(7)(16)0
C=12800	S	80*	80*	RF 80	(44,36)(16,7,5,8)(2,3)(9)(36,8)(11)(28,5)(2,9)(7)(16)0
C=12800	S	80*	80*	RF 80	(51,29)(14,15)(13,1)(16)(29,13,9)(9,4)(4,5)(20)(16,1)(15)0
C=13448	S	82*	82*	RF 82	(47,35)(12,23)(35,12,5,7)(3,2)(1,5,3)(4)(2,1)(24)(23)0
C=13448	S	82*	82*	RF 82	(51,31)(15,16)(9,5,1)(4,13)(31,16,4)(9)(13)(22)(15,1)(14)0
C=14112	S	84*	84*	RF 84	(44,40)(7,13,20)(40,4)(1,6)(5)(11,6,7)(5,1)(4,24)(20)0
C=14112	S	84*	84*	RF 84	(48,36)(9,7,20)(2,5)(8,3)(36,7,5)(8)(2,11)(9)(28)(20)0
C=14450	S	85*	85*	RF 85	(39,25,21)(4,17)(15,14)(1,16)(21,17,1)(15)(16)(31)(4,29)(25)0
C=14450	S	85*	85*	RF 85	(46,39)(9,11,19)(39,7)(7,2)(5,8)(5,2)(3,11)(8)(27)(19)0
C=14792	S	86*	86*	RF 86	(47,39)(7,5,8,19)(2,3)(9)(39,8)(11)(12,5)(2,28)(7)(19)0
C=14792	S	86*	86*	RF 86	(47,39)(8,12,19)(39,9,7)(5,7)(2,5)(11)(3,2)(28)(8)(19)0
C=14792	S	86*	86*	RF 86	(47,39)(11,8,20)(39,8)(3,5)(5,7,2)(7)(11,2)(9)(27)(20)0
C=14792	S	86*	86*	RF 86	(51,35)(15,20)(1,5,9)(35,13,4)(9)(4,16)(13)(22)(1,15)(14)0
C=15138	S	87*	87*	RF 87	(48,39)(7,12,20)(2,5)(39,11)(9,8)(8,3)(28)(2,7)(5)(20)0
C=15138	S	87*	87*	RF 87	(48,39)(8,11,20)(3,5)(39,7,2)(5)(2,9)(7)(12)(8,28)(20)0
C=15138	S	87*	87*	RF 87	(51,36)(14,22)(1,13)(36,16)(9,13)(4,9)(20)(5,4)(1,16)(15)0
C=16562	S	91*	91*	RF 91	(52,39)(15,24)(39,7,6)(4,6,5)(1,9)(8)(1,4)(7)(28)(24)0

C=29160	S	18*	18*	RF810	(11,7)(3,4)(1,2)(7,3,2)(1,3)(1,2)(1,2)(4)(1,4)(3)0
C=28800	S	30*	30*	RF480	(13,8,9)(7,1)(10)(8,3,2)(1,5,3)(4)(2,1)(11)(1,10)(9)0
C=24200	S	55*	55*	RF220	(28,27)(7,8,12)(16,12)(6,1)(5,4)(16)(8,15)(11,5)(1,7)(6)0
C=25088	S	56*	56*	RF224	(24,14,18)(10,4)(7,15)(14,10,9,1)(8)(1,8,23)(4,7)(18)(15)0
C=25088	S	56*	56*	RF224	(30,12,14)(7,3,2)(1,15)(4)(11)(4,11)(26,12,3)(7)(2,16)(14)0
C=25992	S	57*	57*	RF228	(30,27)(11,16)(11,7,12)(4,3)(4,7)(1,2)(16)(15,3)(2,14)(12)0
C=26912	S	58*	58*	RF232	(30,28)(2,6,7,13)(28,4)(9,1)(8)(4,9)(15,2)(6)(1,8)(7)0
C=28800	S	60*	60*	RF240	(32,28)(4,5,6,13)(28,7,1)(5,1)(7)(1,4)(8)(1,19)(5)(13)0
C=30752	S	62*	62*	RF248	(33,29)(8,9,12)(29,4)(11,1)(7,3)(4,11)(3,8)(14)(1,10)(9)1
C=30752	S	62*	62*	RF248	(34,28)(5,8,15)(1,4)(28,7)(1,7)(5)(6,1)(5,1)(4,19)(15)0
C=27556	S	83*	83*	RF166	(31,22,30)(9,13)(7,23)(23,17)(11,2)(9)(13,7)(6,11)(30)(29)(24)0
C=22472	S	106*	106*	RF106	(55,51)(4,5,11,31)(51,7,1)(6)(24)(7,24)(20,4)(1,6)(5)(11)0
C=22472	S	106*	106*	RF106	(55,51)(4,11,36)(51,8)(1,10)(9)(19)(8,9,19)(15,4)(11,1)(10)0
C=22472	S	106*	106*	RF106	(59,47)(12,35)(47,24)(2,5,4,24)(23,1)(3)(1,3)(7,2)(5)(12)0
C=22472	S	106*	106*	RF106	(59,47)(24,23)(47,12)(2,5,4,12)(35,1)(3)(1,3)(7,2)(5)(24)0
C=22898	S	107*	107*	RF107	(54,53)(1,3,8,21,20)(53,2)(5)(13)(2,5,13)(33,1)(3)(8)(21)0
C=22898	S	107*	107*	RF107	(59,48)(11,13,24)(48,8,10,4)(2,11)(6)(6,2)(4,14)(35)(10)(24)0
C=22898	S	107*	107*	RF107	(59,48)(11,13,24)(48,14,6,2)(4,11)(10)(35)(10,4)(2,8)(6)(24)0
C=23328	S	108*	108*	RF108	(64,44)(16,11,17)(5,6)(20,1)(24)(44,20)(4,11,5)(24)(6,23)(17)0
C=23328	S	108*	108*	RF108	(64,44)(20,24)(44,24,11,5)(1,23)(6)(17)(20,4)(6,17)(16,5)(11)0
C=23762	S	109*	109*	RF109	(58,51)(7,10,11,23)(51,11,3)(12,1)(12)(7,4)(3,13)(35)(10)(23)0
C=24200	S	110*	110*	RF110	(57,21,32)(10,11)(9,1)(8,36)(17)(53,21)(10,9,17)(1,8)(32)(25)0
C=24200	S	110*	110*	RF110	(57,24,29)(13,6,5)(1,33)(7)(20)(53,24)(13,20)(6,7)(29,1)(28)0
C=24200	S	110*	110*	RF110	(57,53)(4,5,24,20)(53,7,1)(6)(13)(7,13)(33,4)(1,6)(5)(24)0
C=24200	S	110*	110*	RF110	(57,53)(8,9,36)(53,4)(11,1)(10)(21)(4,11,21)(17,8)(1,10)(9)0
C=24200	S	110*	110*	RF110	(60,27,23)(4,19)(16,15)(1,33)(17)(50,27)(16,17)(23,4)(19,1)(18)0
C=24642	S	111*	111*	RF111	(60,51)(9,8,15,19)(1,7)(51,19)(13,9)(6,13)(8,1)(7)(32)(28)0
C=24642	S	111*	111*	RF111	(60,51)(11,15,25)(51,9)(7,4)(19)(16)(9,16)(26,11,7)(4,19)(15)0
C=24642	S	111*	111*	RF111	(62,49)(20,29)(49,6,7)(5,1)(4,13,6,5)(9)(1,4)(7)(33)(29)0
C=24642	S	111*	111*	RF111	(68,43)(23,20)(3,17)(7,19)(43,20,5)(12)(5,12)(3,26,7)(23)(19)0
C=25088	S	112*	112*	RF112	(60,52)(8,13,15,16)(52,16)(11,2)(9,8)(7,9)(13,2)(36)(11)(24)0
C=25088	S	112*	112*	RF112	(64,48)(7,5,8,28)(2,3)(9)(11)(48,20,5)(2,9)(7)(8,36)(28)0
C=25088	S	112*	112*	RF112	(64,48)(20,28)(48,9,7)(5,7,8)(2,5)(11)(3,2)(9)(8)(36)(28)0
C=25538	S	113*	113*	RF113	(66,47)(23,24)(47,19)(5,7,1)(6,19)(13)(28,6)(1,8,23)(7)(15)0
C=25538	S	113*	113*	RF113	(68,45)(26,19)(7,12)(45,20,3)(12,19,5)(17)(5,7)(25)(23,3)(20)0
C=25992	S	114*	114*	RF114	(41,39,34)(5,29)(2,11,31)(34,9)(20)(9,20)(5,44,2)(11)(39)(31)0
C=25992	S	114*	114*	RF114	(60,29,25)(4,21)(16,17)(15,1)(39)(54,21)(17,4)(14,29)(16,1)(15)0
C=25992	S	114*	114*	RF114	(63,51)(12,11,28)(1,10)(51,13,12)(3,7)(4,11)(10,3)(7)(35)(28)0
C=26450	S	115*	115*	RF115	(65,50)(16,20,14)(5,9)(50,14,1)(13,4)(1,4)(9,16)(13)(36)(29)0
C=26450	S	115*	115*	RF115	(67,48)(11,9,28)(2,7)(8,5)(12)(48,20,7)(5,2)(3,39)(8)(28)0
C=26450	S	115*	115*	RF115	(68,47)(15,32)(6,9)(47,15,8,4)(1,8)(5)(7,1)(6)(4,36)(32)0
C=26912	S	116*	116*	RF116	(60,56)(4,12,6,7,27)(56,8)(5,1)(8)(1,4)(21)(12)(6,33)(27)0
C=26912	S	116*	116*	RF116	(60,56)(4,12,13,27)(56,8)(19,1)(14)(8,33)(15,4)(12)(14,1)(13)0
C=26912	S	116*	116*	RF116	(60,56)(4,12,15,11,14)(56,8)(8,3)(17,3)(17)(14,4)(12)(31)(29)0
C=26912	S	116*	116*	RF116	(60,56)(4,12,15,25)(56,8)(10,7,3)(8,10)(3,4)(13)(12)(35)(25)0
C=26912	S	116*	116*	RF116	(65,51)(9,13,29)(5,4)(1,16)(51,20)(9,36)(16,4)(13)(15,1)(14)0
C=26912	S	116*	116*	RF116	(68,48)(23,25)(48,17,3)(19,7)(5,20)(12)(12,5)(7,26,3)(23)(19)0
C=27378	S	117*	117*	RF117	(65,52)(8,12,32)(5,3)(2,1)(13)(52,20)(8,5)(3,2)(1,33)(32)0
C=27378	S	117*	117*	RF117	(68,49)(26,23)(49,12,7)(3,20)(5,19,12)(17)(7,5)(25)(3,23)(20)0
C=27378	S	117*	117*	RF117	(70,47)(18,29)(3,4,11)(2,1)(5)(47,18,7)(3,2)(1,4)(11)(29)0
C=27848	S	118*	118*	RF118	(62,56)(12,8,7,29)(56,6)(1,6)(4,5)(21,1)(12)(8,4)(33)(29)0
C=27848	S	118*	118*	RF118	(69,49)(8,11,30)(5,3)(2,12)(7)(49,19,8)(5,7)(11,2)(39)(30)0
C=28322	S	119*	119*	RF119	(70,49)(17,32)(4,6,7)(49,17,8)(5,1)(8)(1,4)(9)(6,38)(32)0
C=28800	S	120*	120*	RF120	(62,58)(8,9,15,26)(58,4)(11,1)(10)(4,11)(17,8)(1,36)(9)(26)0
C=28800	S	120*	120*	RF120	(70,50)(19,31)(1,3,8,7)(50,19,2)(5)(2,5)(12,1)(3)(39)(31)0
C=29768	S	122*	122*	RF122	(52,33,37)(29,4)(41)(29,13,10)(3,23,13)(16)(10,3)(44)(41,4)(37)0
C=29768	S	122*	122*	RF122	(66,56)(12,13,31)(56,10)(8,4)(3,10)(7)(15,3)(12,8)(4,35)(31)0
C=31752	S	126*	126*	RF126	(68,58)(11,15,32)(58,9,1)(8,4)(10,9)(17)(1,8)(11)(4,36)(32)0

TABLE III

A few examples of squarings obtained from various nets

c=	130 s	15*	11*	RF	5	(6,4,5)(3,1)(6)(5,1)(4) 1 0 0
c=	130 s	69*	61			(36,33)(5,28)(25,9,2)(7)(16) 1 0 0
c=	130 s	33*	32	RF	2	(18,15)(7,8)(14,4)(10,1)(9) 1 0 0
c=	130 s	33*	32	RF	2	(18,15)(7,8)(14,4)(10,1)(9) 1 0 0
c=	130 s	33*	32	RF	2	(18,15)(7,8)(14,4)(10,1)(9) 1 0 0
c=	130 s	69*	61			(36,33)(5,28)(25,9,2)(7)(16) 1 0 0
c=	130 s	15*	11*	RF	5	(6,4,5)(3,1)(6)(5,1)(4) 1 0 0
c=	130 s	33*	32	RF	2	(18,15)(7,8)(14,4)(10,1)(9) 1 0 0
c=	130 s	69*	61			(36,33)(5,28)(25,9,2)(7)(16) 1 0 0
c=	130 s	69*	61			(36,33)(5,28)(25,9,2)(7)(16) 1 0 0

c=	31752 s	664*	659	RF	24	(357,307)(47,107,153)(3,44)(302,58)(17,11,16)(6,5)(88,46)(81)(199)(163) 0 0 0
c=	31752 s	5665*	4919	RF	3	(2480,1408,1777)(538,501,369)(596,1550)(37,464)(575)(148,912)(2439,41)(764)(1634,42)(1592) 0 0 0
c=	31752 s	5737*	4847	RF	3	(2552,1408,1777)(1039,369)(842,1304)(143,724,172)(2295,219,38)(181)(552,462)(400)(1766)(1676) 0 0 0
c=	31752 s	2866*	2426	RF	6	(1218,744,904)(484,260)(100,165,639)(295,65)(230)(1208,10)(423,71)(352,244)(885)(775) 0 0 0
c=	31752 s	591*	585*	RF	27	(320,271)(41,94,136)(8,33)(265,41,22)(19,3)(16,20)(76)(72,42)(178)(148) 0 0 0
c=	31752 s	670*	653	RF	24	(360,310)(46,88,176)(4,42)(293,71)(33,97)(104)(9,167)(40,66)(118,26)(92) 0 0 0
c=	31752 s	617*	559*	RF	27	(304,150,163)(137,13)(176)(25,60,52)(255,41,8)(33)(8,220)(72,2)(70)(142) 0 0 0
c=	31752 s	662*	661	RF	24	(342,156,164)(148,6)(172)(38,78,32)(46,158)(319,61)(21,103)(82)(176,9)(167) 0 0 0
c=	31752 s	5855*	4729	RF	3	(2385,1570,1900)(856,714)(384,1516)(142,956)(2344,41)(1039)(225,555,176)(379,1313)(1264)(954) 0 0 0
c=	31752 s	6083*	4561	RF	3	(2390,1576,944,1313)(575,369)(334,1348)(1233,143)(590,128)(462)(500,558)(2171,219)(1952)(1900) 0 0 0
c=	31752 s	2686*	2606	RF	6	(1411,1275)(136,464,675)(1195,352)(24,210,230)(376)(190,20)(19,656)(269)(467,99)(368) 0 0 0
c=	31752 s	5591*	4993	RF	3	(2822,2769)(117,428,2224)(2171,379,208,64)(181)(171,37)(88,130)(46,42)(600)(596)(1196) 0 0 0
c=	31752 s	1773*	1755*	RF	9	(923,850)(73,185,176,416)(832,164)(9,167)(52,142)(216)(126,16)(110,73)(489)(452) 0 0 0
c=	31752 s	733*	590	RF	24	(312,198,223)(148,50)(25,42,156)(58,17)(59)(40,18)(278,34)(22,55)(214)(211) 0 0 0
c=	31752 s	1366*	1280	RF	12	(495,419,452)(76,194,149)(116,336)(339,232)(45,220)(239)(107,125)(64,492)(446)(428) 0 0 0
c=	31752 s	1450*	1196	RF	12	(600,428,422)(123,299)(130,181,117)(64,176)(42,88)(596,46)(37,208)(171)(475)(379) 0 0 0
c=	31752 s	2846*	2446	RF	6	(1228,817,801)(16,785)(421,412)(1218,10)(9,213,190)(440)(23,167)(236)(98,860)(768) 0 0 0
c=	31752 s	1408*	1238	RF	12	(682,336,430)(242,94)(191,333)(64,135,43)(92,142)(596,110)(39,188)(149)(475)(337) 0 0 0
c=	31752 s	234*	207*	RF	72	(107,64,63)(1,62)(27,38)(16,11)(26,23)(100,23)(3,82)(2,27)(25)(52) 0 0 0
c=	31752 s	126*	126*	RF	126	(68,58)(11,15,32)(58,9,1)(8,4)(10,9)(17)(1,8)(11)(4,36)(32) 0 0 0

c=	52893 s	30869*	22624			(11464,7398,11407)(4066,3332)(1111,1544,677)(867,11217)(11160,3268,1102)(725,386)(339,47)(2458)(2166)(7892) 0 0 0
c=	52893 s	3528*	2549	RF	9	(1322,966,1240)(312,360,274)(405,1109)(44,200,68)(1027,339)(149,299)(185,17)(166)(704)(688) 0 0 0
c=	52893 s	28706*	24187			(12289,7530,8887)(4759,1739,1032)(707,325)(2828,6384)(2446)(1718,3556)(11898,4006,1144)(2862)(1024,8916)(7892) 0 0 0
c=	52893 s	9609*	8022	RF	3	(4283,2551,2775)(1435,892,224)(724,2275)(836,56)(780)(297,1138)(845,771)(3739,841)(74,2972)(2898) 0 0 0
c=	52893 s	31361*	21532			(12289,8491,10581)(3576,3440,1475)(860,615)(245,1095)(1105)(136,4409)(222,3490)(2843,3268)(5975,783)(5192) 0 0 0
c=	52893 s	9348*	8283	RF	3	(4756,2201,2391)(2011,190)(2581)(461,790,760)(83,378)(3527,1017,295)(30,3311)(49,771)(722)(2510) 0 0 0
c=	52893 s	27311*	25582			(9933,8491,8887)(1442,3799,2854,396)(2458,6825)(6756,4619)(945,4367)(4744)(2137,2482)(1322,9870)(8895)(8548) 0 0 0
c=	52893 s	28637*	24256			(12849,7458,8330)(4367,2219,872)(1565,7637)(2001,218)(1783)(147,3637)(1024,3490)(11407,2466)(8941,652)(8889) 0 0 0
c=	52893 s	990*	969	RF	27	(560,430)(74,124,232)(43,31)(12,19)(13,42)(35,108)(409,135,29)(106)(33,307)(274) 0 0 0
c=	52893 s	31448*	21445			(11464,8941,11043)(2523,3675,2745)(641,10402)(9981,4006)(932,2452)(2854,1753)(233,2219)(1986)(5975,885)(5090) 0 0 0
c=	52893 s	27371*	25522			(15120,12251)(3521,4880,4450)(10402,4066,652)(3414,759)(3799,1240)(1070,3380)(2310)(6336,1144)(849,5441)(5192) 0 0 0
c=	52893 s	27674*	25219			(14268,6558,6843)(6268,290)(1738)(1442,3666,1160)(2506,5792)(10951,4759)(2535,3637)(6198,1102)(351,5441)(5090) 0 0 0
c=	52893 s	26776*	26117			(14110,12666)(1444,4619,6603)(12007,3547)(372,2508,1739)(3919)(1494,245)(6848)(1785,725)(2219)(4541,1161)(3380) 0 0 0
c=	52893 s	8838*	8799	RF	3	(4610,4228)(388,878,770,2186)(4189,809)(108,662)(319,667)(1128)(113,549)(780)(344,205)(2391)(2252) 0 0 0
c=	52893 s	29038*	23861			(12251,5792,4941,6448)(2634,1967)(4409,1383)(727,7626)(4744)(2691,1718)(4305,2157)(11610,641)(3332)(9783)(7637) 0 0 0
c=	52893 s	27940*	24953			(14110,13830)(570,2137,11123)(10843,1907,1070,890)(860)(857,233)(153,707)(386)(218,168)(3012)(2962)(5974) 0 0 0
c=	52893 s	29203*	23690			(12567,7743,8893)(5268,1475)(325,2172,6596)(1800)(1753,47)(2219)(11125,1444)(1240,513)(727,2005)(9679)(8401) 0 0 0
c=	52893 s	30043*	22850			(12007,7138,4450,6448)(2632,1956)(1565,6581)(1341,1111)(6033,1105)(2676)(2446)(2482,2640)(10843,1164)(9679)(9521) 0 0 0
c=	52893 s	28997*	23896			(12048,8548,8401)(2427,5974)(2828,2440,2280)(1150,3547)(672,1544,612)(932,4280)(11848,872)(3343)(9521)(7628) 0 0 0
c=	52893 s	9438*	8195	RF	3	(4177,2486,2775)(972,1225,289)(936,2128)(719,253)(1222,1192)(4016,880)(30,3690)(124,1128)(1004)(2132) 0 0 0
c=	52893 s	28514*	24579			(12531,7653,8330)(3536,3420,677)(2745,6254)(136,2506,3521)(1322,2370)(11848,2005)(957,3919)(2962)(9786)(6381) 0 0 0

c=	59508 s	1682*	1644*	RF	18	(896,766)(152,225,389)(748,126,22)(104,70)(67,3)(64,164)(197,33)(164)(64,489)(425) 0 0 0
c=	59508 s	891*	762*	RF	36	(383,299,269)(148,91)(61,208)(61,91)(379,4)(148,4)(65)(35,56)(79,21)(285)(227) 0 0 0
c=	59508 s	1654*	1652*	RF	18	(896,758)(149,191,418)(756,129,11)(118,42)(118,115)(205,42)(3,112)(163)(54,476)(422) 0 0 0
c=	59508 s	1814*	1492*	RF	18	(748,501,565)(197,240,64)(240,389)(50,104,43)(219,64)(744,54)(158)(155,149)(538)(532) 0 0 0
c=	59508 s	1768*	1538*	RF	18	(794,498,476)(22,454)(243,277)(53,156,34)(156,155)(744,103)(225,34)(1,608)(191)(416) 0 0 0
c=	59508 s	45*	42*	RF	684	(16,14,15)(2,7,5)(4,11)(11,7)(2,7)(2,7)(4,5)(2,16)(15)(14) 0 0 0
c=	59508 s	901*	752*	RF	36	(378,249,274)(133,91,25)(98,201)(59,32)(27,105)(374,4)(120,17)(103)(27,277)(250) 0 0 0
c=	59508 s	1764*	1542	RF	18	(785,425,554)(296,129)(243,440)(64,156,76)(757,92)(122,197)(206,42)(164)(89,548)(459) 0 0 0
c=	59508 s	1774*	1532	RF	18	(785,500,489)(11,478)(234,277)(51,140,43)(130,190)(747,89)(196,33)(163)(103,565)(462) 0 0 0
c=	59508 s	1802*	1504	RF	18	(757,497,548)(206,240,51)(197,402)(54,113,34)(266,5)(205)(747,64)(182)(53,554)(501) 0 0 0
c=	59508 s	1662*	1644*	RF	18	(896,766)(152,225,389)(748,126,22)(104,70)(67,3)(64,164)(197,33)(164)(64,489)(425) 0 0 0
c=	59508 s	891*	762*	RF	36	(383,299,269)(148,91)(61,208)(61,91)(379,4)(148,4)(65)(35,56)(79,21)(285)(227) 0 0 0
c=	59508 s	1774*	1532	RF	18	(785,500,489)(11,478)(234,277)(51,140,43)(130,190)(747,89)(196,33)(163)(103,565)(462) 0 0 0
c=	59508 s	1814*	1492*	RF	18	(748,501,565)(197,240,64)(240,389)(50,104,43)(219,64)(744,54)(158)(155,149)(538)(532) 0 0 0
c=	59508 s	1768*	1538*	RF	18	(794,498,476)(22,454)(243,277)(53,156,34)(156,155)(744,103)(225,34)(1,608)(191)(416) 0 0 0
c=	59508 s	882*	771*	RF	36	(397,211,274)(148,63)(117,220)(38,78,32)(46,103)(374,61)(21,103)(82)(46,277)(231) 0 0 0

C= 99072 S 19023* 14001 RF 3 (7071,5000,6952)(2071,1481,1448)(33,911,504)(916,598)(407,7049)(6930,1886,526)(318,280)(1598)(1560)(5044) 0 0 0
C= 99072 S 3306* 2198* RF 18 (1244,907,1155)(299,360,248)(360,1043)(38,200,61)(954,328)(173,243)(166,34)(132,75)(683)(626) 0 0 0
C= 99072 S 4508* 3748* RF 12 (1882,1224,1402)(658,380,186)(8,429,965)(194)(347,227)(120,536)(1866,605,69)(536)(120,1381)(1261) 0 0 0
C= 99072 S 18015* 15009 RF 3 (7960,4800,5247)(2456,1913,439)(1448,4258)(26,1422)(543,1396)(696,2303)(7049,1607)(1450,1368)(82,5524)(5442) 0 0 0
C= 99072 S 9886* 6626* RF 6 (3764,2690,3432)(1033,1041,616)(490,126)(364,3194)(41,984,8)(984,65)(2862,943)(919)(1919,8)(1911) 0 0 0
C= 99072 S 744* 632* RF 72 (546,204,194)(10,184)(103,111)(39,56,8)(56,63)(286,82,17)(65,8)(57,7)(254)(204) 0 0 0
C= 99072 S 8542* 7970* RF 6 (3048,2690,2804)(358,1305,913,114)(799,2119)(2082,1324)(392,1320)(377,1320)(758,943)(392,3047)(2840)(2655) 0 0 0
C= 99072 S 2264* 1864* RF 24 (995,605,664)(330,216,59)(235,488)(138,78)(50,253)(60,246,24)(222)(869,186)(29,712)(683) 0 0 0
C= 99072 S 2232* 1896* RF 24 (1001,567,664)(268,202,97)(241,320)(66,136)(166,168)(98,279)(895,270,2)(268)(87,712)(625) 0 0 0
C= 99072 S 19695* 13389 RF 3 (7160,5464,7071)(1656,2201,1607)(2420,6258)(40,1071,545)(6169,1031)(920,1826)(1708,394)(1314)(48,3838)(3430) 0 0 0
C= 99072 S 16847* 14177 RF 3 (8008,5272,5567)(2504,2473,295)(2010,3852)(168,1842)(31,2610)(238,2303)(6169,2071)(936,4758)(4098,276)(3822) 0 0 0
C= 99072 S 9166* 7346* RF 6 (4152,2202,2812)(1592,610)(1267,2155)(358,949,285)(664,888)(3194,1316)(725,888)(664,2379)(1878,165)(1715) 0 0 0
C= 99072 S 4508* 3748* RF 12 (1878,1224,1406)(662,380,182)(198,427,963)(349,229)(120,536)(1870,8)(603,67)(536)(120,1379)(1259) 0 0 0
C= 99072 S 2762* 2742* RF 18 (1490,1272)(245,333,694)(1252,211,27)(184,68)(184,237)(307,88)(219,53)(166,124)(42,776)(734) 0 0 0
C= 99072 S 2264* 1864* RF 24 (941,659,664)(225,246,186,5)(181,408)(60,307)(60,138,24)(330)(923,78)(216)(83,712)(629) 0 0 0
C= 99072 S 17631* 15393 RF 3 (7865,4850,5516)(2908,1266)(2478,4310)(631,1597,754)(7528,968)(1388,1838)(2024,543)(1481,450)(1031,5567)(4536) 0 0 0
C= 99072 S 8302* 8210* RF 6 (4470,3832)(758,1133,1941)(3740,610,120)(490,388)(13,312,808)(102,299)(1005,197)(808)(312,2437)(2185) 0 0 0
C= 99072 S 2216* 1912* RF 24 (958,589,669)(373,216)(136,533)(136,216)(21,115)(954,4)(398)(35,181)(150)(4,710)(154)(552) 0 0 0
C= 99072 S 17775* 15249 RF 3 (7865,5036,4874)(162,4812)(2534,2084)(295,1071,1053,130)(908,1886)(7384,776)(33,1913)(1880)(955,5653)(4728) 0 0 0
C= 99072 S 18127* 14897 RF 3 (7465,5310,5352)(1716,2082,1512)(1470,582)(570,2412)(439,911,566)(543,2473)(7432,472)(1908)(531,5663)(3032) 0 0 0
C= 99072 S 17885* 15201 RF 3 (7736,4840,5247)(2232,2201,407)(1794,3560)(31,1898,2056)(664,1509)(7465,935)(2866,268)(1998,168)(6094)(4264) 0 0 0
C= 99072 S 2232* 1896* RF 24 (967,601,664)(268,270,63)(207,520)(98,168,2)(166,313)(299,136)(66,268)(202)(121,712)(591) 0 0 0

```
begin integer N, x ;N: = 8;
  begin integer number of choices;
        Boolean selfdual;
        integer array W[1:2*(2*N÷3+N)+1];
        procedure wheel (B); value B;
                             integer B;
        begin integer MDP, t, l;
              MDP: = B ÷ 2 + 1; t: = 1;
              for l: = 1 step 1 until MDP—2 do
                  begin
                    W[t]: = W[t+3]: = l; W[t+1]: = MDP; W[t+2]: = l + 1; W[t+4]: = 0: t: = t + 5
                  end:
              W[t]: = W[t+3]: = MDP—1; W[t+1]: = MDP; W[t+2]: = 1; W[t+4]: = 0; t: = t + 5;
              for l: = 1 step 1 until MDP - 1 do
                  begin
                    W[t]: = l; t: = t + 1
                  end;
              W[t]: = 1; W[t+1]: = W[t+2]: = 0
        end wheel;
        procedure WRITE (W, number of choices, selfdual); integer number of choices;
                                                          Boolean selfdual;
                                                          integer array W;
        begin integer i; write (0); write (W[1]);
              for i: = 2 step 1 until i do
                  begin
```

```
                write (W[i]); if W[i−1] = 0 ∧ W[i] = 0
                                then go to end
            end;
    end: write (number of choices);
         if selfdual
         then write (1)
         else write (0)
    end WRITE;
write (N); wheel (N); number of choices: = 2; selfdual: = true ;WRITE (W, number of choices, selfdual); write (−1);
go to finish;
start: begin integer K, M, B, H, end of file, identificationnumber, storage;
         integer array branch 1, branch 2, branchdual 1, branchdual 2 [1:N+1], W[1:2*(2*N÷3+N)+1],
                       V[1:2*(2*N÷3+N)+5];
         procedure READ (W, number of choices, selfdual, end of file); integer number of choices, end of file;
                                                                       Boolean selfdual;
                                                                       integer array W;
         begin integer i, j;
               W[1]: = read;
               for i: = 1 step 2 until i do
                  begin
                    W[i+1]: = read; W[i+2]: = read; if W[i+1] = 0 ∧ W[i+2] = 0
                                                    then go to end
                  end;
         end: number of choices: = read;
               j: = read; if j = 0
                          then selfdual: = false
```

```
                else selfdual: = true;
        end of file: = read
end READ;
procedure form branches (V, branch 1, branch 2, branchdual 1, branchdual 2, K, M);
        integer K, M;
        integer array branch 1, branch 2, branchdual 1, branchdual 2, V;
        begin integer m, t, tt, i;
            t: = m: = 1; tt: = 0;
        begin:
            for i: = 1 step 1 until tt do
                begin
                  if V[t+1] = branch 1 [i] ∧ V[t] = branch 2[i]
                  then
                    begin
                      branchdual 2[i]: = m; go to next
                    end
                end i;
            tt: = tt + 1; branch 1[tt]: = V[t]; branch 2[tt]: = V[t+1]; branchdual 1[tt]: = m;
        next: t: = t + 1; if V[t+1] = 0
                        then
                          begin
                            if V[t+2] = 0
                            then go to end;
                            m: = m + 1; t: = t + 2
                          end;
            go to begin;
```

```
        end: B:= tt; M:= m; K:= B + 2 - M
        end form branches;
procedure dualize (branch 1, branch 2, branchdual 1, branchdual 2, K, V);
        integer K;
        integer array branch 1, branch 2, branchdual 1, branchdual 2, V;
        begin integer i, j, l, h, t, search, remember;
            integer array vector 1, vector 2[1:B];
            t:= 0; i:= 1;
        start: l:= 1;
            for j:= 1 step 1 until B do
                begin
                  if branch 1[j] = i
                  then
                    begin
                      vector 2[l]:= branchdual 1[j];
                      vector 1[l]:= branchdual 2[j]; l:= l + 1
                    end;
                  if branch 2[j] = i
                  then
                    begin
                      vector 1[l]:= branchdual 1[j];
                      vector 2[l]:= branchdual 2[j]; l:= l + 1
                    end
                end;
            t:= t + 1; V[t]:= vector 1[1]; search:= remember:= vector 2[1];
        begin:for h:= 1 step 1 until l - 1 do
```

```
                    begin
                      if vector 1[h] = search
                      then
                        begin
                          t: = t + 1; V[t]: = search; search: = vector 2[h];
                          if search = remember
                          then go to continue;
                          go to begin
                        end
                    end;
            continue:
                t: = t + 1; V[t]: = 0; i: = i + 1; if i = K + 1
                                                   then go to end;
                go to start;
            end: t: = t + 1; V[t]: = 0
            end dualize;
procedure identification (V, identificationnumber); integer identificationnumber;
                                                    integer array V;
begin integer i, j, workstorage, maxweight original, maxweight dual, remember maxweight dual, remember max
          weight original, n original, n dual, l;
    Boolean dual with fromdual, fromdual, nogain;
    integer array weight original, location original [1:K+1], inverse location [1:K], weight dual, location dual
          [1:M+1];
    procedure identify (K, weight original, weight dual, location, max weight, n, branch 1, branch 2, branchdual
          1, branchdual 2);
        integer K, n, maxweight;
```

```
        integer array weight original, weight dual, location, branch 1, branch 2, branchdual 1, branch
                dual 2;
begin integer z, weightstorage, t, i, k, q, s, l, min;
    Boolean ready;
    integer array new location, score [1:K];
    nogain: = false;
start: z: = 1; weightstorage: = 1; t: = 1;
    for i: = 1 step 1 until K do score [i]: = 0;
    if fromdual
    then for i: = 1 step 1 until B do
            begin
              score [branch 1 [i]]: = score [branch 1 [i]] + weight dual [branchdual 2 [i]];
              score [branch 2 [i]]: = score [branch 2 [i]] + weight dual [branchdual 1 [i]];
              fromdual: = false
            end
    else for i: = 1 step 1 until B do
            begin
              score [branch 1 [i]]: = score [branch 1 [i]] + weight original [branch 2 [i]];
              score [branch 2 [i]]: = score [branch 2 [i]] + weight original [branch 1 [i]]
            end;
label 1:
    for i: = z step 1 until K do
      begin
        if weight original [location [i]] ≠ weight original [location [i + 1]]
        then go to continue
      end;
```

```
continue:
    if i > z
    then
      begin
        for k: = z step 1 until i do weight original [location [k]]: = 0;
        label 2:
        min: = M*2↑K; ready: = true;
        for l: = z step 1 until i do
         begin
           if score [location [l]] < min ∧ weight original [location [l]] = 0
           then
             begin
               min: = score [location [l]]; ready: = false
             end
         end;
        if ready
        then go to continue i;
        weightstorage: = 2*weightstorage;
        for n: = z step 1 until i do
         begin
           if score [location [n]] = min
           then
             begin
               weight original [location [n]]: = weightstorage;
               new location [t]: = location [n]; t: = t + 1
             end
```

```
                    end n;
                  n: = i; go to label 2
                  end of then i > z
                else
                begin
                  weightstorage: = 2*weightstorage;
                  weight original [location [i]]: = weightstorage;
                  new location [t]: =location [i]; t: = t + 1
                end else;
continue i:
        z: = i + 1; if z ≦ K
                    then go to label 1;
        if weightstorage ≠ 2↑K
        then if weightstorage = maxweight
             then
               begin
                 nogain: = true; go to finish
               end
             else
               begin
                 for s: = 1 step 1 until K do location [s]: = new location [s];
                 maxweight: = weightstorage; go to start
               end else;
        for s: = 1 step 1 until K do location [s]: = new location [s];
finish:
end identify;
```

```
        number of choices: = 0;
        for i: = 1 step 1 until K do
            begin
              weight original [i]: = 2; location original [i]: = i
            end;
        location original [K+1]: = K + 1; weight original [K+1]: = 0; fromdual: = false;
        maxweight original: = 2;
        identify (K, weight original, weight dual, location original, maxweight original, n original, branch 1, branch 2,
                branchdual 1, branchdual 2);
        remember maxweight original: = maxweight original;
        if ¬ nogain
        then go to form identificationnumber;
        for i: = 1 step 1 until M do
            begin
              weight dual [i]: = 2; location dual [i]: = i
            end;
        location dual [M+1]: = M+1; weight dual [M+1]: = 0; fromdual: = false; maxweight dual: = 2;
        dual with fromdual: = false;
        identify (M, weight dual, weight original, location dual, maxweight dual, n dual, branchdual 1, branchdual 2,
                branch 1, branch 2);
        remember maxweight dual: = maxweight dual;
two:    fromdual: = true;
three:  identify (K, weight original, weight dual, location original, maxweight original, n original, branch 1, branch 2,
                branchdual 1, branchdual 2);
        if ¬ nogain
        then go to form identificationnumber;
```

```
        if maxweight original = remember maxweight original
        then
          begin
            if dual with fromdual
            then
              begin
              five: weight original [location [n original]]: =
                    weight original [location [n original]] + 1; dual with fromdual: = false;
                    number of choices: = number of choices + 1; go to three
              end;
            go to four
          end;
        remember maxweight original: = maxweight original;
four:   fromdual: = true;
        identify (M, weight dual, weight original, location dual, maxweight dual, n dual, branchdual 1, branchdual 2,
                  branch 1, branch 2);
        dual with fromdual: = true;
        if nogain
        then go to two;
        if maxweight dual = remember maxweight dual
        then go to five;
        remember maxweight dual: = maxweight dual; go to two;
form    identificationnumber:
        for i: = 1 step 1 until K do inverse location [location original [i]]: = i;
        identificationnumber: = 0;
        for l: = 1 step 1 until B do
```

```
    begin
      i: = K + 1—inverse location [branch 1 [l]]; j: = K + 1—inverse location [branch 2 [l]];
      if i > j
      then
        begin
          workstorage: = i; i: = j; j: = workstorage
        end;
      identificationnumber: = identificationnumber + 2↑((K↑2+K+i*(i—2*K+1)—2*j)÷2)
    end
end identification;
procedure form TNSTAR;
begin integer array U [1:2*(2*N÷3+N)+5];
      procedure new net test (V, storage); integer storage;
                                           integer array V;
      begin integer p;
            own integer array id number [1:4↑(B—9)]
            for p: = 1 step 1 until H do
              begin
                if storage = id number [p]
                then go to end
              end;
            H: = H + 1; id number [H]: = storage; WRITE (V, number of choices, selfdual);
end:
end new net test;
form branches (V, branch 1, branch 2, branchdual 1, branchdual 2, K, M);
if K = M
```

```
then
  begin
    identification (V, identificationnumber);
    storage: = identificationnumber;
    dualize (branch 1, branch 2, branchdual 1, branchdual 2, K, U);
    form branches (U, branch 1, branch 2, branchdual 1, branchdual 2, K, M);
    identification (U, identificationnumber);
    if identificationnumber < storage
    then
      begin
        selfdual: = false; new net test (U, identificationnumber)
      end
    else
      begin
        if identificationnumber > storage
        then
          begin
            selfdual: = false; new net test (V, storage)
          end
        else
          begin
            selfdual: = true; new net test (V, storage)
          end
      end
  end
else
```

```
        begin
          if K > M
          then
            begin
              dualize (branch 1, branch 2, branchdual 1, branchdual 2, K, U);
              form branches (U, branch 1, branch 2, branchdual 1, branchdual 2, K, M);
              identification (U, identificationnumber);
              selfdual: = false;
              new net test (U, identificationnumber)
            end
          else
            begin
              identification (V, identificationnumber); selfdual: = false; new net test (V, identificationnumber)
            end
        end
end form TNSTAR;
procedure generate nets (W); integer array W;
begin Boolean dualized;
      dualized: = false; go to con 2;
con 1: if dualized ∨ selfdual
       then go to finished;
       form branches (W, branch 1, branch 2, branchdual 1, branchdual 2, K, M);
       dualize (branch 1, branch 2, branchdual 1, branchdual 2, K, W);
       dualized: = true;
con 2: begin integer i, ii, m, s, t, MM, p, q, a, b, l;
             integer array sum [1:N], multiplicity [1:N];
```

```
      m: = t: = sum [1]: = i: = 1;
label: if W[t+2] = 0
      then
        begin
          t: = t + 3; i: = i + 1; sum [i]: = t; multiplicity [i−1]: = m; m: = 1;
          if W[t] = 0
          then go to follow
        end;
      t: = t + 1; m: = m + 1; go to label;
follow:
      MM: = i − 1;
      for ii: = 1 step 1 until MM do
        begin
          if multiplicity [ii] > 3
          then
            begin
              q: = sum [ii] − 1;
              for a: = 1 step 1 until sum [ii] − 1 do V[a]: = W[a];
              for b: = sum [ii+1] step 1 until sum [MM+1] do V[b+4]: = W[b];
              for s: = 1 step 1 until multiplicity [ii] − 2 do
                  begin
                    for l: = s + 2 step 1 until if s = 1 then multiplicity [ii] − 1
                                                else multiplicity [ii] do
                        begin
                          p: = q + 1;
                          for m: = s step 1 until l do
```

```
                                    begin
                                        V[p]: = W[m+q]; p: = p + 1
                                    end;
                                V[p]: = W[s+q]; p: = p + 1; V[p]: = 0; p: = p + 1;
                                for m: = l step 1 until multiplicity [ii] do
                                    begin
                                        V[p]: = W[m+q]; p: = p + 1
                                    end;
                                for m: = 1 step 1 until s do
                                    begin
                                        V[p]: = W[m+q]; p: = p+1
                                    end;
                                V[p]: = W[l+q]; V[p+1]: = 0;
                                form TNSTAR;
                            end l
                        end s
                    end if
                end ii
            end block con 2;
            go to con 1;
        finished:
        end generate nets;
    H: = 0; write (N+1);
next net:
    READ (W, number of choices, selfdual, end of file);
    generate nets (W);
```

```
      if end of file ≧ 0
      then
        go to next net;
      N: = N + 1; if N − 2*(N÷2) = 0
                  then
                    begin
                      wheel (N); selfdual: = true; number of choices: = 2; WRITE (W, number of choices, selfdual)
                    end;
          write (−1);
  end;
finish: stop; N: = read; x: = read; go to start;
end
```


PROGRAMME II

```
begin integer B, x;
next B:
      B: = read; x: = read;
start: begin integer end of file, number of choices;
            integer array V[1:2*(B+2*B÷3)+1];
            procedure READ (W, number of choices, end of file); integer end of file, number of choices;
                                                                integer array W;
            begin integer i, j;
                  W[1]: = read;
```

```
        for i: = 1 step 2 until i do
            begin
              W[i+1]: = read; W[i+2]: = read; if W[i+1] = 0 ∧ W[i+2] = 0
                                                    then go to end
            end;
end: number of choices: = read; j: = read; end of file: = read
end READ;
READ (V, number of choices, end of file);
begin integer K, M, complexity, hcf;
      integer array branch 1, branch 2, branchdual 1, branchdual 2[1: B], INC [1:2*B÷3—1, 1:4*B÷3—2],
                  ZINV [1:2*B÷3, 1:2*B÷3];
      procedure form branches (V, branch 1, branch 2, branchdual 1, branchdual 2, K, M);
                integer K, M;
                integer array branch 1, branch 2, branchdual 1, branchdual 2, V;
      begin integer m, t, tt, i;
            t: = m: = 1; tt: = 0;
      begin: for i: = 1 step 1 until tt do
                begin
                  if V[t+1] = branch 1[i] ∧ V[t] = branch 2[i]
                  then
                      begin
                        branchdual 2[i]: = m; go to next
                      end
                end i;
             tt: = tt + 1; branch 1[tt]: = V[t]; branch 2[tt]: = V[t+1]; branchdual 1[tt]: = m;
      next:  t: = t + 1; if V[t+1] = 0
```

```
                    then
                      begin
                        if V[t+2] = 0
                        then go to end;
                        m: = m + 1; t: = t + 2
                      end;
        go to begin;
end:  B: = tt; M: = m; K: = B + 2 − M
end form branches;
procedure dualize (branch 1, branch 2, branchdual 1, branchdual 2, K, V);
          integer K;
          integer array branch 1, branch 2, branchdual 1, branchdual 2, V;
begin     integer i, j, l, h, t, search, remember;
          integer array vector 1, vector 2 [1:B];
          t: = 0; i: = 1;
start:    l: = 1;
          for j: = 1 step 1 until B do
              begin
                if branch 1[j] = i
                then
                  begin
                    vector 2[l]: = branchdual 1[j];
                    vector 1[l]: = branchdual 2[j]; l: = l + 1
                  end;
                if branch 2[j] = i
                then
```

```
                    begin
                      vector 1[l]: = branchdual 1[j];
                      vector 2[l]: = branchdual 2[j]; l: = l + 1
                    end
                end;
            t: = t + 1; V[t]: = vector 1[1]; search: = remember: = vector 2[1];
begin:      for h: = 1 step 1 until l − 1 do
                begin
                  if vector 1[h] = search
                  then
                    begin
                      t: = t + 1; V[t]: = search; search: = vector 2[h];
                      if search = remember
                      then go to continue;
                      go to begin
                    end
                end;
continue:   t: = t + 1; V[t]: = 0; i: = i + 1; if i = K + 1
                                                then go to end;
                                                go to start;
end:        t: = t + 1; V[t]: = 0
end dualize;
procedure HCF(x, y); integer x, y;
begin integer RN1, RN2;
      RN1: = x; hcf: = y;
algorithm:
```

```
      RN2: = RN1 − hcf*(RN1 ÷ hcf);
      if RN2 ≠ 0
      then
        begin
          RN1: = hcf; hcf: = RN2;
          go to algorithm
        end;
      hcf: = abs(hcf)
end HCF;
form branches (V, branch 1, branch 2, branchdual 1, branchdual 2, K, M);
if K < M
then
  begin
    dualize (branch 1, branch 2, branchdual 1, branchdual 2, K, V);
    form branches (V, branch 1, branch 2, branchdual 1, branchdual 2, K, M)
  end;
comment initialize matrix INC;
begin integer i, j;
      for i: = 1 step 1 until M − 1 do
          begin
            for j: = i + 1 step 1 until M + i − 2 do INC [i, j]: = 0
          end
end initialize matrix;
comment form upper triangle;
begin integer i:
      for i: = 1 step 1 until B do
```

```
            begin
              if branchdual 2[i] ≠ M
              then
                begin
                  INC [branchdual 1[i], branchdual 2[i]]: = −1;
                  INC [branchdual 2[i], branchdual 2[i]]: = INC [branchdual 2[i], branchdual 2[i]] + 1
                end;
              INC [branchdual 1[i], branchdual 1[i]]: = INC [branchdual 1[i], branchdual 1[i]] + 1
            end
end form upper triangle;
comment initialize inverse of INC;
begin integer i;
      for i: = 1 step 1 until M − 1 do INC [i, M + i − 1]: = 1
end initialize inverse of INC;
comment Gaussian elimination;
begin integer i, j, k, l, f, g, h;
      for i: = 1 step 1 until M − 2 do
         for j: = i + 1 step 1 until M − 1 do
            begin
              if INC [i,j] ≠ 0
              then
                begin
                  HCF (INC [i,j]*INC [j,j + M − 1], INC [i, i]*INC [i, i + M − 1]);
                  f: = INC [i,j]*INC [j,j + M − 1] ÷ hcf;
                  g: = INC [i, i]*INC [i, i + M − 1] ÷ hcf;
                  INC [j,j + M − 1]: = g*INC [j,j + M − 1];
```

```
                    for k: = j step 1 until i + M − 1 do INC [j, k]: = g*INC [j, k] − f*INC [i, k];
                    HCF (INC [j, j], INC [j, j + M − 1]);
                    for h: = j + 1 step 1 until i + M − 1 do
                        begin
                          if INC [j, h] ≠ 0
                          then
                            begin
                              HCF (hcf, INC [j, h]);
                              if hcf = 1
                              then go to continue
                            end
                        end;
                    for l: = j step 1 until i + M − 1 do
                        begin
                          if INC [j, l] ≠ 0
                          then INC [j, l]: = INC [j, l] ÷ hcf
                        end;
                      INC [j, j + M − 1]: = INC [j, j + M − 1] ÷ hcf;
                  continue:
                  end then
              end j
end Gaussian elimination;
comment calculation of the complexity;
begin integer N, D, i;
      N: = INC[1,1]; D: = INC [1,M];
      for i: = 2 step 1 until M − 1 do
```

```
            begin
              HCF (N*INC [i, i], D*INC [i, i + M − 1]);
              N: = INC [i, i]*N ÷ hcf;
              D: = INC [i, i + M − 1]*D ÷ hcf
            end;
        complexity: = N
end calculation of the complexity;
comment backsubstitution;
begin integer i, j, k, l, m, f, g;
        for i: = 1 step 1 until M − 1 do
            begin
              if INC [M − i, M − i] = complexity
              then go to for j;
              HCF(INC [M − i, M − i], complexity);
              f: = complexity ÷ hcf;
              g: = INC [M − i, M − i] ÷ hcf;
              for k: = M step 1 until 2*M − i − 1 do INC [M − i, k]: = INC [M − i, k]*f ÷ g;
            for j:
              for j: = i + 1 step 1 until M − 1 do
                  begin
                    if INC [M − j, M − i] ≠ 0
                    then
                      begin
                        HCF (INC [M − j, M − i], complexity);
                        f: = INC [M − j, M − i] ÷ hcf;
                        g: = complexity ÷ hcf;
```

```
                    for l: = M step 1 until 2*M—1—j do INC[M—j, l]:=g* INC[M—j, l]—f* INC[M—i,l];
                    if g ≠ 1
                    then for m: = M — j step 1 until M — 1 — i do INC [M — j, m]: = g*INC [M — j, m];
                  end then
              end j
          end i
end backsubstitution;
comment put final touch to the inverse of INC;
begin integer i, j;
      for i: = 1 step 1 until M — 1 do
          begin
            for j: = 1 step 1 until i do ZINV [i,j]: = INC [i, M — 1 + j]
          end lower triangle ZINV;
      for i: = 1 step 1 until M — 1 do
          for j: = i + 1 step 1 until M — 1 do ZINV [i,j]: = ZINV [j, i];
      for i: = 1 step 1 until M do
          begin
            ZINV [i, M]: = 0; ZINV [M, i]: = 0
          end
end final touch of the inverse of INC;
begin integer r, zero currents, RF, vertical, horizontal, b;
      Boolean original, second time, imperfection, trivial imperfection;
      integer array current, positive original, positive dual, negative original, negative dual [1:B], from original,
                  from dual [1:2*B], Bouwkamp code [1:2*B—1], address original [0:K+1], address dual
                  [0:M+1];
```

```
procedure left cyclic ordening adjacent vertices (branch 1, branch 2, branchdual 1, branchdual 2, positive,
        negative, address, from, K);
        integer K;
        integer array branch 1, branch 2, branchdual 1, branchdual 2, positive, negative, address,
                from;
begin integer h, i, j, k, remember, meshsearch;
    k: = 1; i: = 1; address [0]: = 0; address [1]: = 1;
search first branch:
    for j: = 1 step 1 until B do
        begin
          if branch 1 [j] = i
          then
            begin
              remember: = meshsearch: = branchdual 1[j]; from [k]: = j; positive [j]: = k;
              go to go on searching
            end;
          if branch 2 [j] = i
          then
            begin
              remember: = meshsearch: = branchdual 2[j]; from [k]: = j; negative [j]: = k;
              go to go on searching
            end
        end j;
go on searching:
        k: = k + 1;
        for h: = 1 step 1 until B do
```

```
                begin
                  if branch 1[h] = i ∧ branchdual 2[h] = meshsearch
                  then
                    begin
                      if branchdual 1[h] = remember
                      then go to continue;
                      from [k]: = h; positive [h]: = k; meshsearch: = branchdual 1[h];
                      go to go on searching
                    end;
                  if branch 2[h] = i ∧ branchdual 1[h] = meshsearch
                  then
                    begin
                      if branchdual 2[h] = remember
                      then go to continue;
                      from [k]: = h; negative [h]: = k; meshsearch: = branchdual 2[h];
                      go to go on searching
                    end
                end h;
continue:
        i: = i + 1; address [i]: = k; if i ≠ K + 1
                                      then go to search first branch
end  left cyclic ordening adjacent vertices;
procedure form code (branch 1, branch 2, positive, negative, address, from);
            integer array branch 1, branch 2, positive, negative, address, from;
begin integer next first, former first, next second, former second, place, increment, signum, first vertex;
        integer array reduced ordered current [1:2*B];
```

```
procedure reduce address (u, t);
          integer u, t;
begin
  place: = if u < address [t]
           then adress [t+1] − 1
           else if u = address [t+1]
                then address [t]
                else u
end reduce address, where u is to be reduced and t the vertex;
begin integer i;
     for i: = 1 step 1 until B do
         begin
           reduced ordered current [positive [i]]: = current [i] ÷ RF;
           reduced ordered current [negative [i]]: = −current [i] ÷ RF
         end i
end;
if second time
then go to first and third way of forming code;
reduce address (positive [r]+1, branch 1[r]);
next first: = abs (reduced ordered current [place]);
reduce address (positive [r]−1, branch 1[r]);
former first: = abs (reduced ordered current [place]);
reduce address (negative [r]+1, branch 2[r]);
next second: = abs (reduced ordered current [place]);
reduce address (negative [r]−1, branch 2[r]);
former second: = abs (reduced ordered current [place]);
```

```
        if next first ≧ next second ∧ next first ≧ former first ∧ next first ≧ former second
        then
          begin
            increment: = 1; go to first and third way of forming code
          end;
        if next second ≧ former first ∧ next second ≧ former second
        then
          begin
            increment: = 1; go to second and fourth way of forming code
          end;
        increment: = −1;
        if former first ≧ former second
        then go to first and third way of forming code
        else go to second and fourth way of forming code;
first and third way of forming code:
        first vertex: = branch 1[r];
        if current [r] > 0
        then signum: = −1
        else signum: = 1;
        go to start Bouwkamp code;
second and fourth way of forming code:
        first vertex: = branch 2[r];
        if current [r] > 0
        then signum: = 1
        else signum: = −1;
start Bouwkamp code:
```

```
begin integer end, min, where, i, j, k, l, p, q, ii, count, s, v, t, number of squares;
    integer array contour, vertex contour, save contour, save vertex contour, new squares,
                vertices new squares [1:B];
    procedure fetch new squares (branch 1, branch 2, address, from);
            integer array branch 1, branch 2, address, from;
            comment ii has to be initialized, vertex t has to be given, b is a running variable;
    begin integer l;
        Boolean T, S;
        l:= 0; T:= true; place:= address [t]; S:= true;
    label: place:= place + increment;
        reduce address (place, t)
        if reduced ordered current [place] = 0
        then go to label;
        if sign (reduced ordered current [place]) = sign (if T then signum else −signum)
        then
          begin
            if ¬ (T ∧ (¬ S))
            then go to label;
          label 1:
            Bouwkamp code [b]:= abs (reduced ordered current [place]); b:= b + 1;
            new squares [ii]:= abs (reduced ordered current [place]);
            vertices new squares [ii]:= if branch 1 [from [place]] = t
                                        then branch 2 [from [place]]
                                        else branch 1 [from [place]];
            ii:= ii + 1; go to label
          end;
```

```
                    if T ∧ S
                    then
                      begin
                        T: = false; go to label
                      end;
                    if ¬(T ∧ (¬ S))
                    then
                      begin
                        T: = true; S: = false; go to label 1
                      end;
                    number of squares: = ii − 1
    end fetch new squares;
    Bouwkamp code [1]: = − 1; b: = 2; vertex contour [1]: =first vertex; end: =1; contour [1]:=0;
back: min: = contour [1]; where: = 1;
    for i: = 2 step 1 until end do
      begin
        if contour [i] < min
        then
          begin
            min: = contour [i]; where: = i
          end
      end;
    count: = 0;
    for j: = where + 1 step 1 until end do
      begin
        if min ≠ contour [j]
```

```
            then go to next
            else count: = count + 1
          end;
next: ii: = 1;
      for k: = 0 step 1 until count do
          begin
            t: = vertex contour [where + k];
            fetch new squares (branch 1, branch 2, address, from)
          end;
      for i: = 1 step 1 until number of squares—1 do
          begin
            if new squares [i] = new squares [i+1]
            then
              begin
                trivial imperfection: = true; go to follow
              end
          end;
follow:
      Bouwkamp code [b]: = — 1; b: = b + 1;
      for l: = where + count + 1 step 1 until end do
          begin
            save contour [l]: = contour [l];
            save vertex contour [l]: = vertex contour [l]
          end;
      for p: = where step 1 until where + number of squares — 1 do
          begin
```

```
                contour [p]: = new squares [p+1 — where] + min;
                vertex contour [p]: = vertices new squares [p+1 — where]
            end;
        for q: = where + count + 1 step 1 until end do
            begin
                contour [number of squares — count + q — 1]: = save contour [q];
                vertex contour [number of squares — count + q — 1]: = save vertex contour [q]
            end;
        if where > 1
        then s: = where — 1
        else s: = 1
        for v: = s + 1 step 1 until end + number of squares — count — 1 do
            begin
                if ¬(contour [s] = contour [v] ∧ vertex contour [s] = vertex contour [v])
                then
                    begin
                        s: = s + 1; contour [s]: = contour [v]; vertex contour [s]: = vertex contour [v]
                    end
            end;
        end: = s;
        if end ≠ 1
        then go to back
    end Bouwkamp code
end form code;
left cyclic ordening adjacent vertices (branch 1, branch 2, branchdual 1, branchdual 2, positive original, nega
                                                    tive original, address original, from original, K);
```

```
left cyclic ordening adjacent vertices (branchdual 1, branchdual 2, branch 1, branch 2, positive dual, negative
                                        dual, address dual, from dual, M);
for r: = 1 step 1 until B do
    begin integer s;
        for s: = 1 step 1 until B do
            current [s]: = ZINV [branchdual 1[r], branchdual 1[s]]
                         — ZINV [branchdual 1[r], branchdual 2[s]]
                         — ZINV [branchdual 2[r], branchdual 1[s]]
                         + ZINV [branchdual 2[r], branchdual 2[s]];
        comment test imperfection;
        imperfection: = false; second time: = false; trivial imperfection: = false;
        begin integer i;
            for i: = 1 step 1 until B— 1 do
                for j: = i + 1 step 1 until B do
                    begin
                      if abs (current [i]) = abs (current[j])
                      then
                        begin
                          imperfection: = true; go to count zero currents
                        end
                    end
        end;
    count zero currents:
        zero currents: = 0;
        begin integer k;
            for k: = 1 step 1 until B do
```

```
                    begin
                      if current [k] = 0
                      then zero currents: = zero currents + 1
                    end
                end;
                HCF (current [1], complexity);
                begin integer l;
                    for l: = 2 step 1 until B do HCF (current [l], hcf);
                    RF: = hcf
                end;
                if (complexity − current [r]) > current [r]
                then
                  begin
                    vertical: = current [r] ÷ RF;
                    current [r]: = current [r] − complexity;
                    horizontal: = − current [r] ÷ RF;
                    original: = false; go to dual net
                  end
                else
                  begin
                    original: = true;
                    vertical: = (complexity − current [r]) ÷ RF;
                    horizontal: = current [r] ÷ RF
                  end;
original net:
        form code (branch 1, branch 2, positive original, negative original, address original, from original);
```

```
procedure left cyclic ordening adjacent vertices (branch 1, branch 2, branchdual 1, branchdual 2, positive,
        negative, address, from, K);
        integer K;
        integer array branch 1, branch 2, branchdual 1, branchdual 2, positive, negative, address,
                from;
begin integer h, i, j, k, remember, meshsearch;
    k: = 1; i: = 1; address [0]: = 0; address [1]: = 1;
search first branch:
    for j: = 1 step 1 until B do
        begin
          if branch 1 [j] = i
          then
            begin
              remember: = meshsearch: = branchdual 1[j]; from [k]: = j; positive [j]: = k;
              go to go on searching
            end;
          if branch 2 [j] = i
          then
            begin
              remember: = meshsearch: = branchdual 2[j]; from [k]: = j; negative [j]: = k;
              go to go on searching
            end
        end j;
go on searching:
        k: = k + 1;
        for h: = 1 step 1 until B do
```

```
                    begin
                      if current [k] = 0
                      then zero currents: = zero currents + 1
                    end
                end;
                HCF (current [1], complexity);
                begin integer l;
                    for l: = 2 step 1 until B do HCF (current [l], hcf);
                    RF: = hcf
                end;
                if (complexity − current [r]) > current [r]
                then
                  begin
                    vertical: = current [r] ÷ RF;
                    current [r]: = current [r] − complexity;
                    horizontal: = − current [r] ÷ RF;
                    original: = false; go to dual net
                  end
                else
                  begin
                    original: = true;
                    vertical: = (complexity − current [r]) ÷ RF;
                    horizontal: = current [r] ÷ RF
                  end;
original net:
        form code (branch 1, branch 2, positive original, negative original, address original, from original);
```

```
            go to dummy point;
dual net:
            form code (branchdual 1, branchdual 2, positive dual, negative dual, address dual, from dual);
dummy point:
            punch (complexity);
            punch (horizontal); punch (vertical);
            if imperfection
            then punch (1)
            else punch (0);
            punch (RF);
            begin integer i;
                  for i: = 1 step 1 until b − 1 do punch (Bouwkamp code [i])
            end;
            punch (number of choices); punch (zero currents);
            if ¬ imperfection
            then go to next r
            else if trivial imperfection
                 then go to next r
                 else if second time
                      then go to next r
                      else
                        begin
                          second time: = true; current [r]: = − current [r];
                          if original
                          then go to dual net
                          else go to original net
                        end;
```

```
                    next r:
                        if trivial imperfection
                        then punch (1)
                        else punch (0)
                    end r;
                end
            end;
            if end of file ≧ 0
            then go to start
            else
              begin
                stop; go to next B
              end
        end
end
```

Acknowledgement

I am indebted to P. Medema who wrote the first programmes concerning the squaring problem for the I.B.M. 650. Also his help in debugging the PASCAL programmes and his assistance with the production runs are highly appreciated. The many discussions we had and his valuable remarks were a great help.
Furthermore I want to acknowledge Mr van der Sloot and Mr van der Giezen who spent several nights in taking care of part of the production runs on PASCAL.
To H. C. J. A. Nunnink I owe an example of a net (with code 145104695 403796303273028972015981018236410O) where the dual net is necessary to improve the maximum weight of the original net in the **procedure** indentification.
Finally I want to thank N.V. Centrex and N.V. Eindhovensche Drukkerij for the care with which they have printed this thesis.

Curriculum vitae

Born in the Hague, 10 Dec. 1927. H.B.S.b, Sint Janscollege, the Hague, 1946; Electrotechnisch ingenieur, Technological University, Delft, the Netherlands, 1950; Doctoraal examen, mathematics and physics, University of Amsterdam, 1955. Worked at the Mathematical Centre in Amsterdam in the computation department under the direction of prof. dr ir A. van Wijngaarden from 1953 to 1956. Joined Philips Research Laboratories, Eindhoven, the Netherlands, in 1956. Since 1960 at Philips Computing Centre.

Address of the author: Philips Computing Centre,
Eindhoven, The Netherlands.