QI QUAESTIONES INFORMATICAE

Vol. 3 No. 3

August, 1985

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUAESTIONES INFORMATICAE

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Editor: Prof G. Wiechers
INFOPLAN
Private Bag 3002
Monument Park 0105

Editorial Advisory Board

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R10	$7	£5
Institutions	R15	$14	£10

Circulation and Production Manager

MR. C. S. M. Mueller
Department of Computer Science
University of the Witwatersrand
1 Jan Smuts Avenue
Johannesburg 2001

Quaestiones Informaticae is prepared by the Computer Centre and Central Graphics Unit, and printed by the Central Printing Unit, all of the University of the Witwatersrand, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

Editorial Note

I am happy to present the current issue of Quaestiones Informaticae. As indicated in the last issue, Vol.3 No.2, it was necessary to look for a new method to produce the journal. Thanks to the efforts of Prof. Judy Bishop we have found a way to continue at an acceptable cost, and the result is in your hands. Wits University's Computer Centre will do the 'typesetting' using a laser printer. Judy will use Computer Science students to do the proof reading, and Wits will also do the printing. On behalf of myself, the Computer Society of SA and the SA Institute of Computer Scientists, I want to express my appreciation for all their efforts.

Conrad Mueller, of Wits' Computer Science department has agreed to become the circulation manager for QI. He will handle the business side of our publication. A hearty welcome to him!

G. WIECHERS
Editor.

STOP PRESS

QI is now on the official supplementary list of journals considered by the National Department of Education for subsidy purposes.

A Comparison of Methods used to Represent Graphs on a Computer

GEOFF SUTCLIFFE

Department of Computer Science
University of Natal
King George V Avenue
Durban
South Africa
4001

ABSTRACT

In general graph theorists have a limited knowledge of computers and computing, and are unaware of the possible variations of representation available, and in what circumstances which representation would be better. To assist in this regard a comparative study of currently used methods for representing graphs on a computer was made. The comparison was over four regions of interest, a) time efficiency, b) representation flexibility, c) space efficiency, d) language flexibility. The comparison was made for five classes of graph theoretic problems, a) graph creation, b) path finding, c) structure finding, d) graph traversal, e) graph analysis.

A minimal set of graph operations for graph manipulation was also provided in the course of the comparison.

KEYWORDS : Data structure, Graph, Graph representation.

Computing Review catagories : E 1. DATA STRUCTURES, Graphs.

NOTATION

The following notation applies throughout this paper:

a) v(i) - vertex with label i,
b) e(i) - edge with label i,
c) V - the number of vertices in the graph,
d) E - the number edges in the graph,
e) D - the graph is directed,
f) U - the graph is undirected.
g) Average adjacency - this is the average number of vertices any vertex in the graph is adjacent to.

THE REPRESENTATIONS STUDIED

There are four generalised graph representation types currently in use. The basic representation types have various specific implementations designed to meet specific needs. Twelve graph representations were studied, and under their type classifications are as follows :

1. Adjacency matrices. Three adjacency matrix representations were studied. They are:

   a. Adjacency matrices - Normal; [4] : The graph is represented by a V x V matrix A where A(i,j) is an attribute of the edge running from v(i) to v(j), 1<=i,j<=V. If there is no such edge a null attribute is entered.

   b. Adjacency matrices - Labelled; [1] : The graph is represented by a V x V matrix A where A(i,j) is the label of the edge running from v(i) to v(j), 1<=i,j<=V. If there is no such edge a null label is entered.

   c. Adjacency matrices - Boolean; [1,2,4,6,7,8,9,10,12] : The graph is represented by a V x V boolean matrix B where B(i,j) is TRUE iff there is an edge running from v(i) to v(j), 1<=i,j<=V.

2. Linked list representations. Five linked list representations were studied. Of these three are restricted to either directed or undirected graphs only:

   a. Adjacency lists - Normal; [1,2,6,7,9,10,12] : The graph is represented by V linked lists and a vector of V list headers. Each list header represents a vertex, and each node in a list refers to a vertex adjacent from the vertex represented by the list header, thus implicitly representing a directed edge.

   b. Adjacency lists - Inverted; [9] : The graph is represented by V linked lists and a vector of V list headers. Each list header represents a vertex and each node in a list refers to a vertex adjacent to the vertex represented by the list header, thus implicitly representing a directed edge.

   c. Adjacency lists - Modified (Undirected graphs only) : The graph is represented by V linked lists and of a vector of V list headers. Each list header represents a vertex and each linked list contains a node for each vertex adjacent from the vertex represented by the list header, provided that the label of the vertex represented by the list header is greater than or equal to the label of the vertex referred to in the list node. Thus each list node implicitly represents an undirected edge.

   d. Orthogonal adjacency lists (Directed graphs only); [9] : The graph is represented by 2*V linked lists and a 2 x V array of list headers. Each pair of list headers represents a vertex and each list node represents a directed edge. One element of each list header pair heads a list of edges incident from the vertex represented by the header pair, and the other element heads a list of edges incident to the vertex represented by the header pair. Each list node contains references to its head and tail vertices and has two link fields, allowing it to be linked into both the "edges incident from" list associated with its tail vertex and the "edges incident to" list associated with its head vertex. This representation is a combination of a) and b).

e. Adjacency multilists (Undirected graphs only); [9]: The graph is represented by V linked lists and a vector of V list headers. Each list header represents a vertex and each list node represents an undirected edge. Each list header heads a list of edges incident on the vertex represented by the list header. Each list node contains references to both its end vertices and has two link fields, allowing it to be linked into the "edges incident" lists associated with both its end vertices.

3. Incidence matrices. Three incidence matrix representations were studied. They are:

   a. Incidence matrices - Labelled; [8] : The graph is represented by a V x E matrix A where A(i,k) = j if v(i) is adjacent to v(j) via e(k), and 0 otherwise, 1<=i,j<-V,1<=k<=E.

   b. Incidence matrices - Inverted : The graph is represented by a V x E matrix A where A(i,k) = j if v(i) is adjacent from v(j) via e(k), and 0 otherwise, 1<=i,j<=V,1<=k<=E.

   c. Incidence matrices - Boolean; [4,7] : The graph is represented by a V x E boolean matrix B where B(i,j) is true iff e(j) is incident on v(i), 1<=i<=V,1<=j<=E.

4. Edge lists. These fall into a class of their own, having nothing in common with any of the other representations. [5]: The edge e(i) running from v(j) to v(k) is represented by the entries A(i) = j and B(i) = k, 1<=i<=E, 1<=j,k<=V. Vertices are thus only represented implicitly. Isolates are represented by a dummy edge running from the isolate to a vertex labelled 0.

TECHNIQUE OF COMPARISON

The aim of the analysis was to find standards by which the representations could be compared. Given these standards a set of demands on them could be created and a comparison of the performance of the representations under these demands could be made. The standards chosen were based on requirements of graph representation and manipulation. The demands were a representative set of graph algorithms, divided into five classes.

THE STANDARDS

1. *Graph operations*

The primary standard upon which the comparison was made was a set of graph operations with which all graphical aspects of graph algorithms could be performed. Non-graphical aspects, such as algebraic computations, were ignored on the grounds that all representations would be equally affected by such aspects.

The set of graph operations decided on are as follows:

Class 1 - Graph Structure Access

a) Add a new vertex.	ADV(v)
b) Add an edge between vertices v and v'.	ADE(v,v')
c) Remove vertex v.	REV(v)
d) Remove edge w.	REE(w)
e) Set attribute n of vertex v to x.	SVA(v,n,x)
f) Set attribute m of edge w to y.	SEA(w,m,y)
g) Retrieve attribute n of vertex v.	x:=RVA(v,n)
h) Retrieve attribute m of edge w.	y:=REA(w,m)

Class 2 - Control Structures

a) For each vertex v do	FEV(v)
b) For each edge w do	FEE(w)
c) For each vertex v' adjacent from vertex v (via edge w) do	FVAF(v,v1,w)
d) For each vertex v' adjacent to vertex v (via edge w) then	FVAT(v,v1,w)
e) If vertex v is adjacent to vertex v' (via edge w) do	VAD(v,v',w)
f) If edges w and w' are co-incident (through vertex v) then	EAD(w,w',v)
g) If edge w is incident on vertex v (to vertex v') then	EVIT(w,v,v')
h) If edge w is incident on vertex v (from vertex v') then	EVIF(w,v,v')

Note that for undirected graphs operations 2c and 2d, 2g and 2h have the same effect. Where necessary a reference to an edge by a variable w may be replaced by the two end vertices of the edge. These operations were found to be sufficient to implement all graphical aspects of graph algorithms.

Given the set of graph operations, these needed to be allocated real times in order to do numerical comparisons. For each graphical operation for each representation, the computer operations (e.g. array accesses, **for** loops etc) required to execute the graph operation (if possible) were calculated. In cases where the number of computer operations were V and/or E dependent, an average case was derived. Hence the time needed by each representation to perform could be calculated.

The fundamental computer operations which were used to construct the graph operations for each representation are as follows:

Computations	*Controls*
Set variable	Comparison >=
Array insert	Comparison >
Array retrieve	Comparison =
Vector insert	Comparison <>
Vector retrieve	For loop
Boolean array insert	Exit/Goto
Boolean array test	
Create list node	
Link list nodes	
Move to next node in list	
Set list node field	
Retrieve list node field	

The times required to do these computer operations were obtained from a bench marking program run on the Z80 Zilog chip. These times were assumed to be representative of many CPUs.

2. *Representation flexibility*

There are many types of graphs, with different structures. Each representation can represent a sub-set of all possible graph structures. This standard refers to the ability of the representations to represent different graph structures, and to access elements of a represented graph. The representation and access standards by which the representations were compared are as follows:

Representation

Can represent both directed and undirected graphs/
Can represent only directed or undirected graphs

Multiple edges	No multiple edges
Loops	No loops
Vertex labels	No vertex labels
Edge labels	No edge labels
Vertex attributes	No vertex attributes
Edge attributes	No edge attributes

Access

Edges defined by label and/or end vertices.
Vertices accessed by label,pointer etc.
Vertices accessible randomly and/or sequentially.
Edges accessed by label, pointer, end vertices etc.
Edges accessible randomly and/or sequentially.
Vertex attributes accessed via edge label, pointer etc.
Edge attributes accessed via edge labels, end vertex labels, direct access etc.

These capabilities and features were sufficient to define how a graph may be stored and accessed in each representation.

3. *Space requirements*

The RAM space required by each representation was calculated by defining the basic data constructs of each representation, and working out how many of each construct would be required to represent a graph of given parameters. The parameters were :

a) V,
b) E,
c) number of vertex attributes,
d) number of edge attributes,
e) size of a vertex attribute,
f) size of an edge attribute,
g) size of vertex labels (if relevant),
h) size of edge labels (if relevant).

To determine the size of each construct the following sizes for common data structures were used:

Data type	*Space required*
integer	2 bytes
real	4 bytes
character	1 byte
pointer	2 bytes
packed boolean	1 bit
arrays	as per array type, no dope vector.

4. *Language flexibility*

Some of the representations required specialised language features such as pointers, dynamic variables etc, which could only be supplied by certain languages. Only high level languages were considered, as these are the languages commonly used for graph processing. For each representation a list of some common languages that could support it was made.

THE DEMANDS

The demands placed on the representations were a set of graph algorithms. The algorithms were divided into five classes and examined under the following headings:

a. Algorithm description,

b. Algorithm features. This was in terms of the capabilities as discussed in 2),

c. Each algorithm was converted so that all graphical aspects of the algorithm were performed using the basic set of graph operations as described in 1),

d. An operation count of the graph operations used in the converted algorithm was done. For every algorithm an average case count was found for each graph operation. For some operations in some algorithms a best/average/worst case count was justified and calculated also.

Given the ability of each representation to perform the graph operations and to represent different graph structures, b) and d) above determined which representations could execute which algorithms. The efficiency of each representation in performing the algorithms that it could perform was measured using the times calculated for each graph operation for each representation, substituted into d) above.

The following algorithms were studied (divided into the five classes):

Class 1 - Graph creation algorithms

a) Harary graphs [4].
b) Create graph.
c) Line graph.
d) Pruned graph.
e) Regularisation of a planar graph [5].

Class 2 - Path finding algorithms

a) Dijkstra's algorithm [9].
b) All pairs shortest paths [9].
c) Fleury's algorithm [4].

Class 3 - Structure finding algorithms

a) Matching in a bi-partite graph.
b) Krusal's algorithm [9].
c) Prim's algorithm [1].
d) Stable marriage algorithm [10].
e) Detect cycle [12].

Class 4 - Graph traversal algorithms

a) Depth first search [9].
b) Breadth first search [9].
c) Post order tree search [11].

Class 5 - Graph analysis algorithms

a) Connected components [9].
b) Topological sort [9].
c) Boolean adjacency matrix.
d) Check for reflexivity [3].
e) Degree.

COMPARISON RESULTS

This section gives the results obtained from comparing representations under the standards and demands specified in the preceding sections.

1. *Comparison by representation flexibility*

Firstly for each algorithm class it was necessary to decide which representations are suitable for that class. This was done by looking at the representation suitability for each algorithm in each class. There are three possible reasons for a representation being unable to perform an algorithm:

a) cannot do a required graph operation,
b) cannot supply a required feature,
c) cannot represent the graph type the algorithm is aimed at
(i.e. cannot represent directed/undirected graphs).

Representations were not penalised for being unable to represent a graph type, but were excluded from a class if they could not perform most of the algorithms for one of the other two reasons.

The most flexible representations appear to be Adjacency lists - Normal, Adjacency lists - Inverted and to a slightly lesser extent Edge lists. These representations can cope with both directed and undirected graphs, and are suitable for all classes of algorithms. Of the specialised representations Orthogonal adjacency lists are suitable for representing directed graphs, and Adjacency multilists are suitable for undirected graphs.

2. *Comparison by time efficiency*

Once the comparison by representation flexibility had been done the ability of each representation to perform each graph operation was examined. This was done for V running from 10 to 10 5 and at two edge densities for each V. The two edge densities considered in undirected graphs were:

a) the edge density of a tree,
b) an average edge density, half way between a tree and a
complete simple graph.

Originally a complete graph was also considered but it was found that the results are almost exactly the same as for b). a) is called the low edge density and E=O(V). b) is called the high edge density and E=O(V 2). In the case of directed graphs twice the number of edges were used to give the same edge density, since directed simple graphs can have twice the number of edges of undirected graphs.

To compare representations within a class, some standard demand of that class had to be established. The operation counts of each algorithm in each class were added to provided a measure of the importance of each graph operation in each class. Algorithms that could be applied to both undirected and directed graphs had their operation counts added in for both cases. Having established a "graph operation count" for each class the "times" for each representation for each class were calculated from the operation times previously determined. The results of these calculations are not actually times although they are referred to as such. Actually they are a measure of the representations' abilities to perform algorithms in each algorithm class. The times were compared and the results are summarised in Table 1.

TABLE 1

Preferred representations in each algorithm class for various graph types

Algorithm Class	Graph Type	Edge Density	Preferred Representations
1	U	Low	Adjacency multilists Adjacency lists - Normal
1	U	High	Adjacency multilists Adjacency lists - Normal
1	D	Low	Adjacency lists - Normal Orthogonal adjacency lists
1	D	High	Adjacency lists- Normal Orthogonal adjacency lists
2	U	Low	Adjacency lists - Normal Adjacency multilists Any linked list Any adjacency matrix
2	U	High	Adjacency matrices - Labelled
2	D	Low	Adjacency lists - Normal Orthogonal adjacency lists Any linked list Any adjacency matrix
2	D	High	Adjacency matrices - Labelled
3	U	Low	Adjacency lists - Modified Any linked list
3	U	High	Adjacency lists - Modified Any non-adjacency matrix
3	D	Low	Any linked list
3	D	High	Any non-adjacency matrix
4	U	Low	Adjacency multilists Adjacency lists - Normal
4	U	High	Adjacency multilists Adjacency lists - Normal
4	D	Low	Orthogonal adjacency lists Adjacency lists - Normal
4	D	High	Orthogonal adjacency lists Adjacency lists - Normal
5	U	Low	Adjacency matrices - Boolean Any adjacency matrix Any linked list is tolerable
5	U	High	Adjacency matrices - Boolean Any adjacency matrix
5	D	Low	Adjacency matrices - Boolean Any adjacency matrix Any linked list is tolerable
5	D	High	Adjacency matrices - Boolean Any adjacency matrix

The linked list representations emerge as the overall best. At low density their supremacy is unrivalled, while at high edge density they are the top performer in three of the five algorithm classes. Of the linked lists Adjacency lists - Normal give the best overall performance, being suited to both directed and undirected graphs. Adjacency multilists and Orthogonal adjacency lists give performances that equal, and in some cases improve upon, that of Adjacency lists - Normal. However they are both limited to one type of graph. The adjacency matrices give top performance at high density in two of the algorithm classes. Of the adjacency matrices, Adjacency matrices - Labelled appear to be the best, followed by Adjacency matrices - Boolean. The incidence matrices and edge lists do not have the power of the other two groups of representations.

The main inadequacy of the linked list representations is their inefficiency with the VAD operation. If this operation is used in graphs of high edge density, the inherent ability of the adjacency matrices to do this operation makes them far superior to the linked list representations. Whereas the adjacency matrices have a constant time for the VAD operation, the linked list representation times are average adjacency dependent, i.e. proportional to E and inversely proportional to V.

The adjacency matrices have their problems with the FVAF and FVAT operations. Of the linked list representations Adjacency lists - Normal, Adjacency multilists and Orthogonal adjacency lists perform the important FVAF operation in a time related to average adjacency, while the adjacency matrices perform it in a time proportional to V. This means that the linked list representations' time complexities for FVAF are at worst equal to those of the adjacency matrices, and at best are an order of exponentiation on V better (assuming $V<=O(E)<=V^2$). It is only the mentioned linked list sentatirepre ons that do perform FVAF so well.

Another facet of interest is that while Adjacency lists - Normal perform the FVAF operation well, they perform the FVAT operation very badly. The converse is true for Adjacency matrices - Inverted. These limitations are inherent in the structure of these two representations. However, Adjacency multilists and Orthogonal adjacency lists perform both operations well, in their respective graph types.

3. *Comparison by space requirements*

The adjacency matrix representations require space proportional to V^2, and independent of E and hence perform equally at high and low density and in directed and undirected graphs.

The space used by the linked list representations is dependent primarily on E at high edge density. The space used by each header is at most half that of an edge node, and $E>V-1$ means that the size of the edge nodes is of primary importance. For Adjacency lists - Normal and Adjacency lists - Inverted this is especially true in undirected graphs as each edge is represented by two edge nodes.

The incidence matrix representations fare poorly in terms of space requirements, especially the non-boolean ones. The space used by an incidence matrix representation is proportional to $E*V$, and thus increases both with V and with edge density. Comparison of the boolean and non-boolean incidence matrix representations with the boolean and non-boolean adjacency matrix representations shows clearly how edge density affects the space requirements of the two matrix type representations.

E is the controlling factor in the space used by the Edge lists. Edge lists are the most efficient users of space as there is no extra data to give the representation structure (e.g. pointer fields), nor wasted data area (e.g. non-existent edges are catered for in adjacency matrices). Also the implicit representation of vertices helps to save space.

Overall Edge lists are the best in terms of space requirements. At high edge densities with no edge attributes Adjacency matrices - Boolean are the best, but only for this limited situation. The linked list representations perform nearly as well as Edge lists and their space requirements are within an order of magnitude of that of Edge lists. Incidence matrices are not space efficient and should not be used for large graphs, especially where high densities are expected.

4. *Comparison by language flexibility*

Edge lists are most easily supported and only highly specialised languages such as LISP and FORTH cannot support them directly. The non-boolean matrix representations are also suitable for most high level languages. Whereas Edge lists can be stored in one dimensional arrays, these representations require two dimensional arrays. The boolean matrix representations have less language flexibility as a boolean data type is required.

The linked list representations can only be represented by PASCAL-like languages, such as PASCAL, ALGOL 68 and PL/1. This is because dynamic variables and a pointer data structure are necessary to implement linked structures effectively. Linked structures can be implemented using arrays and/or vectors, but this is not true to the spirit of linked data structures and there are limitations in such techniques.

The PASCAL-like languages are the best for graph theory applications. They can cope with all types of representation structure, notably the linked list representations. The only failure of this type of languages is that some do not have boolean data structures. The FORTRAN-like languages can support all the matrix representations and Edge lists, which is essentially a matrix representation anyway. Specialised languages like FORTH and LISP are not suitable for any of the studied graph representations. LISP would be suitable for a representation that used only linked structures, e.g. a linked list representation with linked headers.

CONCLUSION

Overall the linked list representations give the best performances. In terms of flexibility and time efficiency they are the best performing representations. Their space requirements are low, and the space required is proportional to the size of the graph, i.e. proportional to V + E. Adjacency and incidence matrices have space requirements proportional to V^2
and V*E respectively. The only drawback of linked list representations is that they can only be effectively implemented by languages that supply dynamic variables and pointer structures. All the other representations only need array or vector data types.

Of the linked list representations, Adjacency lists - Normal and Adjacency lists - Inverted are the most flexible, and give good performance in most situations. The specialised linked list representations, Adjacency multilists and Orthogonal adjacency lists, give better performances than Adjacency lists - Normal and Adjacency lists - Inverted, but are limited to undirected and directed graphs respectively.

The use of extra "structure" items in representing graphs is justified, despite the increased space requirements. One envisages an improved linked list representation with some extra structure to improve the ability to check for the adjacency of any two vertices in the graph. Other extra "structure" items that could be added are separate linking of edge nodes and linked headers rather than an array.

ACKNOWLEDGEMENTS

This research was made possible by grants from the University of Natal and the C.S.I.R.

REFERENCES

1. Aho A.V., Hopcroft J.E., Ullman J.D., *Data Structures and algorithms*, Addison-Wesley Publishing Company, 1983.

2. Aho A.V., Hopcroft J.E., Ullman J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

3. Berztiss A.T., *Data Structures, Theory and practice*, Academic Press, 1971.

4. Bondy J.A., Murty U.S.R., *Graph Theory with Applications*, Macmillan and American Elsevier, 1976.

5. Deo N., *Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall Inc., 1974.

6. Golumbic M.C., *Algorithm Graph Theory and Perfect Graphs*, Academic Press Inc., 1980.

7. Gotlieb C.C., Gotlieb L.R., *Data types and Structures*, Prentice-Hall Inc., 1978.

8. Hall P.A.V., *Computational Structures*, Macdonald and Jane's, Ltd, 1975.

9. Horowitz E., Sahni S., *Fundamentals of Data Structures*, The Pitman Press, 1976.

10. Sedgewick R., *Algorithms*, Addison-Wesley Company, 1983.

11. Standish T.A., *Data Structure Techniques*, Addison-Wesley Company, 1980.

12. Tremblay J-P, Sorenson P.G., *An Introduction to Data Structures with Applications*, McGraw-Hill Book Company, 1976.

NOTES FOR CONTRIBUTORS

The purpose of this journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles, exploratory articles of general interest to readers of the Journal. The preferred languages of the Journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to:
Prof. G. Wiechers at:
INFOPLAN
Private Bag 3002
Monument Park 0105

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. The original ribbon copy of the typed manuscript should be submitted. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name, and the affiliation and address. Each paper must be accompanied by an abstract less than 200 words which will be printed immediately below the title at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Illustrations and tables should not be included in the text, although the author should indicate the desired location of each in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Illustrations should also be supplied on separate sheets, and each should be clearly identified on the back in pencil with the Author's name and figure number. Original line drawings (not photoprints) should be submitted and should include all relevant details. Drawings etc., should be submitted and should include all relevant details. Drawings etc., should be about twice the final size required and lettering must be clear and "open" and sufficiently large to permit the necessary reduction of size in block-making.

Where photographs are submitted, glossy bromide prints are required. If words or numbers are to appear on a photograph, two prints should be sent, the lettering being clearly indicated on one print only. Computer programs or output should be given on clear original printouts and preferably not on lined paper so that they can be reproduced photographically.

Figure legends should be typed on a separate sheet and placed at the end of the manuscript.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters; between the letter O and zero; between the letter l, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetical order of author's name, and cited in the text by number in square brackets. Journal references should be arranged thus:

1. ASHCROFT,E. and MANNA,Z.(1972).The Translation of 'GOTO' Programs to 'WHILE' Programs, in *Proceedings of IFIP Congress 71*,North-Holland, Amsterdam, 250-255.

2. BOHM,C. and JACOPINI,G.(1966).Flow Diagrams, Turing Machines and Languages with only Two Formation Rules, *Comm.ACM*, 9,366-371.

3. GINSBURG,S.(1966). *Mathematical Theory of context-free Languages*, McGraw Hill, New York.

Proofs and reprints

Galley proofs will be sent to the author to ensure that the papers have been been correctly set up in type and not for the addition of new material or amendment of texts. Excessive alterations may have to be disallowed or the cost charged to the author. Corrected galley proofs, together with the original typescript, must be returned to the editor within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.

Hierdie notas is ook in Afrikaans verkrygbaar.