

## A COLLECTION OF GRAPH ANALYSIS APL FUNCTIONS

E. Girard, D. Bastin and J. C. Rault  
Laboratoire Central de Recherches  
Thomson-CSF  
Domaine de Corbeville, B. P. 10  
(91) Orsay, France

### Summary

A set of functions dealing with graph theory is presented: graph description, modifications, k-connectivity analysis, and search for paths with given properties. Graph coding coherence and the modularity of APL functions enable one to link these different procedures at will and to use them in such different problems as digital circuit implementation, fault detection, and I. C. mask layout.

The APL functions are given in Appendix 1 and detailed examples in Appendix 2.

### 1 - Introduction

Graph theory is encountered in many fields of application. Graphs are very useful for modeling and describing processes and systems. Thus there is a constant need for algorithms dealing with graphs and leading to efficient computer programs.

During the development of several projects concerning simulation of digital circuits, generation of fault detection and location sequences, and layout of printed and/or integrated circuits, we had an opportunity to experiment with many graph theoretical algorithms. This led us to write a collection of APL functions which we intend to describe in this paper.

The main data structures and general functions for handling them are first described. Then a collection of APL functions dealing with many aspects of graph theory are detailed.

### 2 - Graph Description

The choice of a good description for graphs is not a trivial task. Two main constraints, usually conflicting, prevail:

- keeping memory occupation to a minimum,
- providing efficient execution.

Several coding schemes are generally used. On the one hand, there are list structures which meet the first requirement but have the drawback of being unwieldy to handle; on the other hand there is the connection matrix in which the  $i$ th row contains the numbers of the vertices connected (next neighbors, predecessors or successors as the case may be) to vertex numbered  $i$ . This coding scheme meets the second requirement but has the objectionable feature of wasting memory unnecessarily.

This state of affairs means that one should not have a single coding scheme but rather the capability of several schemes with the appropriate routines for switching easily from one to the other.

In what follows we use mainly two coding schemes:

- \* In the first method, akin to list processing, graphs are described by means of a single vector whose components are either zero or integer indices. The indices for the vertices connected to vertex labeled  $i$  (adjacent vertices predecessors or successors accordingly) are the components of this vector comprised between the  $i$ th and the  $i$ -1th zero. This way of describing graphs is convenient for APL for it keeps memory occupation to a minimum while it is well suited to APL operators. The only disadvantage in certain applications such as graph reduction, is the requirement that vertices be numbered from 1 to  $N$ , with  $N$  the total number of vertices in the graph under consideration.
- \* In the second method, graphs are described by means of the so-called arc matrix (or edge matrix in the case of unoriented graphs). This matrix has two columns and as many lines as there are arcs (or edges) in the graph. This scheme leads to memory occupation usually larger than that in the first scheme but it is better fitted to the array capabilities of APL operators.

## General Functions

According to the preceding coding schemes it is necessary to handle vectors consisting of groups of indices included between two separators, namely zeros.

V PR N                   extracts the Nth group from vector V.

Example:

```
2 3 0 1 3 0 1 2 4 0 3 0 PR 2
1 3
```

V PP N                   gives the components stored in a given vector V between the zero whose index in V is N, and the preceding zero.

Example:

```
2 3 0 1 3 0 1 2 4 0 3 0 PP 10
1 2 4
```

V SN N                   indicates which groups contain a component of value N.

Example:

```
2 3 0 1 3 0 1 2 4 0 3 0 SN 3
1 2 4
```

V SR N                   gives the indices of each 0 immediately following the components of value N in a given vector V

Example:

```
2 3 0 1 3 0 1 2 4 0 3 0 SR 3
3 6 12
```

V MODIFY W               replaces in the vector V the group of number W[1] by the group formed by W[2], ...XXXX

Example:

```
2 3 0 1 3 0 1 2 4 0 3 0 C MODIFY 2 5 6 7
2 3 0 5 6 7 0 1 2 4 0 3 0
```

N PERL V                 exchanges in vector V components whose values are N and N[2]

Example:

```
3 4 PERL 2 3 0 1 3 0 1 2 4 0 3 0
2 4 0 1 4 0 1 2 3 0 4 0
```

N PERB V                 exchanges in vector V the two blocks whose ranks are N[1] and N[2]

Example:

```
3 4 PERB 2 3 0 1 3 0 1 2 4 0 3 0
2 3 0 1 3 0 3 0 1 2 4 0
```

Three special functions for sorting and deletion are constantly used. These are:

- TTRIC V: sorts a given vector V in ascending order with deletion of multiple occurrences.
- GOM V: deletes multiple occurrences in a given vector V.
- TRI V: merges groups from a describing vector having common components, into a single group.
- CODE DECODE V: decodes the vector V according to the code provided by the matrix CODE where the first row corresponds to the new numbering and the second one to the first numbering.

Example:

```

M
0 1 2 3 4
0 3 2 4 1

N DECCDE 2 3 0 1 3 0 1 2 4 0 3 0
2 1 0 4 1 0 4 2 3 0 1 0

```

#### 4 - Graph-Describing Functions

This niladic function simply builds up, in a conversational mode, the graph describing vector for both oriented and unoriented graphs.

- Input: for each vertex the list of its successors (or of the adjacent vertices)
- Output: the resulting describing vector.

In the case of unoriented graphs the description is checked for inconsistencies and all described vertices are printed out.

An example is given in Appendix 2 and will be used throughout this paper for illustrating the different APL functions.

#### Functions OBTPRE and OBTSUC:

These two complementary monadic functions allow, in the case of oriented graphs, one to obtain the predecessor vector PREDEC from the successor vector SUCCES and vice-versa.

Example:

```

SUCCES
3 0 3 0 4 0 5 0 6 0 4 7 0 5 6 0 9 0 7 10 0 8 11 0 12
0 13 0 14 15 0 12 0 16 17 0 14 18 0 0 20 0 22 0 22 0
11 0

[*PREDEC←OBTPRE SUCCES
0 0 1 2 0 3 0 0 4 7 0 5 0 6 9 0 7 10 0 8 0 9 0 10 22
0 11 14 0 12 0 13 16 0 13 0 15 0 15 0 16 0 0 19 0 0
20 21 0

[*OBTSUC PREDEC
3 0 3 0 4 0 5 0 6 0 4 7 0 5 6 0 9 0 7 10 0 8 11
0 12 0 13 0 14 15 0 12 0 16 17 0 14 18 0 0 0 20 0
22 0 11 0 11 0

```

**Function OBTAJ:**

This is for deriving the unoriented graph associated with an oriented graph.

**Example:**

[+ADJAC+OBTADJ SUCCES

3	0	3	0	1	2	4	0	3	5	0	0	4	6	7	0	4	5	7	0	5	6	8	9
0	7	9	10	0	7	8	10	0	8	9	11	0	10	12	22	0	11	13	14	0			
12	14	15	0	12	13	16	0	13	16	17	0	14	15	18	0	15	0						
16	0	20	0	19	22	0	22	0	11	20	21	0											

**Functions ARETE and VECT:**

The arc matrix ARCMAT or the edge matrix EDGMAT are derived from the corresponding describing vectors SUCCES or ADJAC by means of the function ARETE (French for "edge"):

ARCMAT ← ARETE SUCCES  
 EDGMAT ← ARETE ADJAC

In fact two different functions are used: one, ARETE, if multiple edges are not considered; the other, ARET, when multiple edges are present.

Conversely, the describing vectors SUCCES and ADJAC are derived from the corresponding matrices:

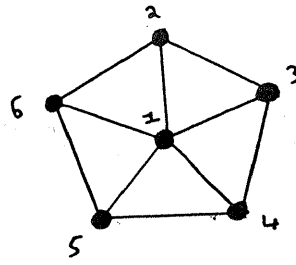
SUCCES ← VECT ARCMAT  
 ADJAC ← VECT EDGMAT

**Wheels**

Function ROUE (French for "wheel") builds up the describing vector ADJAC for a wheel of given order. The "hub" is the vertex labeled 1 and the other ones are on the rim.

ADJAC ← ROUE N

**Example:**



ROUE 5

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

### Adjacent Vertices

Vertices adjacent to a given vertex labeled N are provided by the dyadic function ADJA:

ADJA ← SUCCES ADJA N

#### Example:

SUCCES ADJA 11  
10 12 22

SUCCES ADJA 1  
3

### Ancestors of a Given Vertex

Function ASCEND gives the set of vertices from which a given vertex labeled N may be reached.

ANC ← SUCCES ASCEND N

#### Example:

SUCCES ASCEND 22  
20 21 19

SUCCES ASCEND 3  
1 2

### Descendants of a Given Vertex

Conversely the function DESCEN gives the set of vertices which may be reached from vertex labeled N.

DES ← SUCCES DESCEN N

#### Example:

SUCCES DESCEN 12  
13 14 15 12 16 17 18

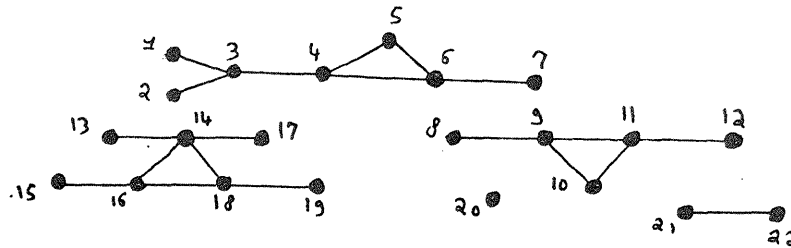
SUCCES DESCEN 8  
9 7 10 5 6 11 6 12 4 13 14 15 16 17 18

**Connected Components:**

Function COMCO derives the weakly-connected component to which the vertex labeled N belongs:

CSC ← ADJAC COMCO N

**Example:**



ADJAC COMCO 12  
12 11 9 10 8

ADJAC COMCO 20  
20

Function COFCO in a similar way gives the strongly-connected component to which the vertex labeled N belongs.

CFC ← SUCCES COFCO N

**Example:**

SUCCES COFCO 3  
3  
SUCCES COFCO 7  
5 8 6 9 4 7 10  
SUCCES COFCO 10  
6 9 7 10 5 6 4

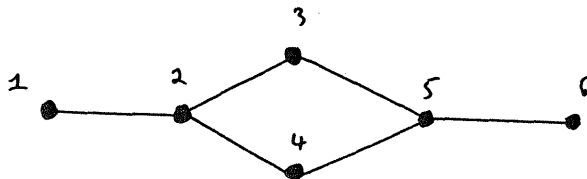
**5 - Graph-Structuring Functions**

Here are gathered several functions used for modifying graphs by removal, addition or duplication of arcs and/or vertices. The functions given below in this paper concern unoriented graphs only. Similar functions exist for oriented graphs.

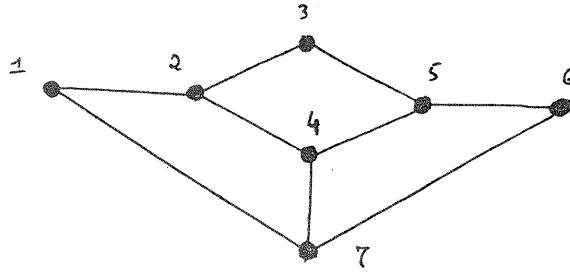
**Addition of a Vertex**

The added vertex is labeled with an index equal to the highest vertex index in the graph plus one. It is sufficient to indicate by the vector N which of the vertices should be adjacent to the new vertex.

**Example:**



GRAF2  
2 0 1 3 4 0 2 5 0 2 5 0 3 4 6 0 5 0



```

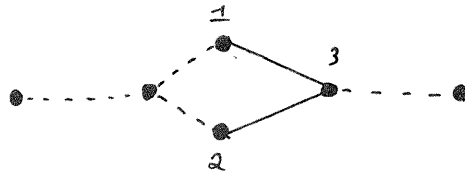
1 4 6 AJOUTS GRAF2
7 2 0 1 3 4 0 2 5 0 7 2 5 0 3 4 6 0 7 5
0 1 4 6 0

```

**Deletion of a Vertex**

Several vertices may be removed from a graph by means of the function ENLEVS. The vertices to be removed are given in vector N. Vertices are relabeled.

Example:



```

1 2 6 ENLEVS GRAF2
3 0 3 0 1 2 0

```

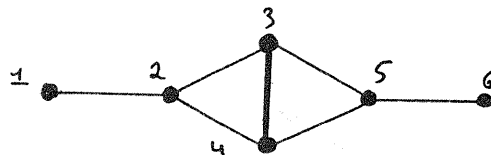
**Addition of an Edge**

The addition of an edge is performed through the dyadic function AJOUTA:

$$ADJAC \leftarrow N \quad AJOUTA \quad ADJAC$$

Here N is a two-component vector indicating the two vertices incident to the added edge.

Example:



```

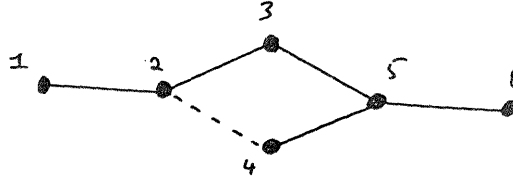
3 4 AJOUTA GRAF2
2 0 1 3 4 0 2 5 4 0 2 5 3 0 3 4 6 0 5 0

```

Deletion of an Edge

Similarly an edge is deleted with the function ENLEVA:

Example:



```
2 4 ENLEVA GRAF2
2 0 1 3 0 2 5 0 5 0 3 4 6 0 5 0
```

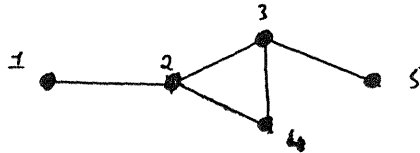
Merging of Two Vertices

On merging two vertices, the resulting vertex is incident to all the edges incident to the two initial vertices. Edges which may link these two vertices are deleted. Vertices are relabeled.

```
ADJAC ← N      CONTRA      ADJAC
```

N is a two-component vector giving indices of vertices to be merged.

Example:



```
3 5 CONTRA GRAF2
2 0 1 3 4 0 2 4 5 0 2 3 0 3 0
```

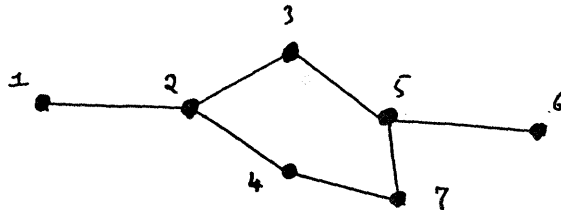
Vertex Splitting

A given vertex may be split so as to generate two vertices. One of these vertices keeps the initial index, the other is labeled with index N + 1 where N is the total number of vertices in the initial graph.

Edges initially incident to the considered vertex are assigned to two resulting vertices according to the user's choice. The way the splitting is performed is fixed in the left argument of the corresponding function dubbed MITOSE for obvious reasons. This left argument is a vector W whose first component has for its value the index for the vertex to be split. The other components are the indices for the vertices which should be kept adjacent to the first resulting vertex.

```
ADJAC ← W      MITOSE      ADJAC
```

Example:



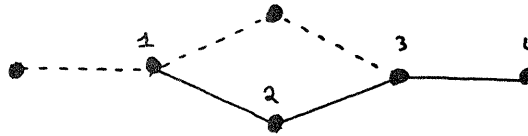
```
5 3 6 MITOSE GRAF2
2 0 1 3 4 0 2 5 0 7 2 0 3 6 7 0 5 0 4 5 0
```

Maximal Subgraph Extraction

Extracting a maximal subgraph from a graph (which means simply keeping in the graph a set of N vertices along with all the associated edges in the given graph) is performed with the function SSGMAX. Vertices of the subgraph obtained in this way are then relabeled with indices ranging from 1 to N in correspondence with the initial order.

GRAPH ← SG SSGMAX GRAPH

Example:



2 4 5 6 SSGMAX GRAF2  
2 0 1 3 0 2 4 0 3 0

6 - Graph Analyzing Functions

In this section we give a non-exhaustive set of functions for the determination of the characteristic components of, or the reduction of, graphs.

Determination of In-Degrees and Out-Degrees

The function DEMDEG determines the in- and out-degrees of a subset, SET, of vertices in a given graph.

D ← SUCCESS DEMDEG SET

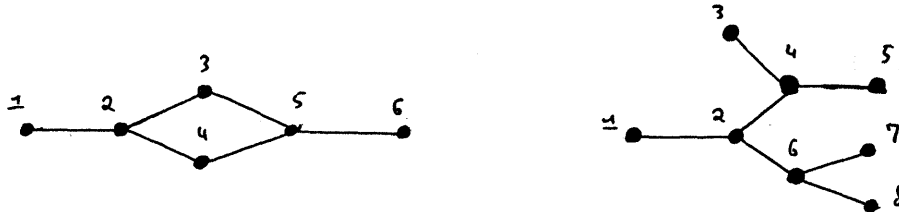
D is a two-component vector where D[1] and D[2] are the in-degree and the out-degree of SET.

1 2 SUCCE DEMDEG 12 13 14 15 16

Checking for Cycles

Cycles in a graph (assumed to be connected) are detected with the function CYCLE. The procedure used here consists in deleting pendent vertices from the graph. Then pendent vertices are deleted from the resulting graph and the procedure is iterated until no more vertices can be deleted. If all vertices have been considered, no cycle is present.

Example:



CYCLE 2 0 1 3 4 0 2 5 0 2 5 0 3 4 6 0 5 0  
LE GRAPHE POSSEDE AU MOINS UN CYCLE.

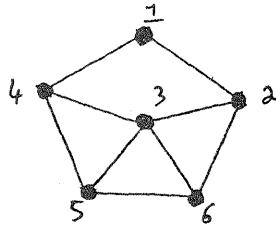
CYCLE 2 0 1 4 6 0 4 0 2 3 5 0 4 0 2 7 8 0 6 0 6 0  
LE GRAPHE EST SANS CYCLE.

Checking Whether a Graph is a Wheel

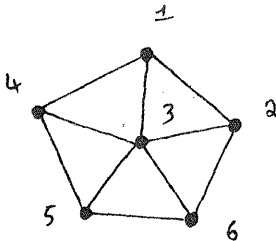
The monadic function WHEEL returns zero if the tested graph is not a wheel, and N if it is a wheel of order N.

N ← WHEEL GRAPH

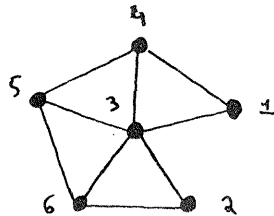
Example:



0  
WHEEL 2 4 0 1 3 6 0 2 4 5 6 0 1 3 5 0 4 3 6 0 5 3 2 0



5  
WHEEL 2 3 4 0 1 3 6 0 1 2 4 5 6 0 1 3 5 0 4 3 6 0 5 3 2 0



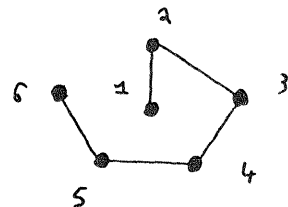
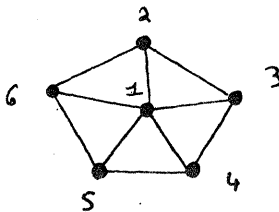
0  
WHEEL 3 4 0 3 6 0 1 2 4 5 6 0 1 3 5 0 4 3 6 0 5 3 2 0

Determination of a Spanning Tree

This is a classical algorithm implemented by function ARBCOV. The result is simply a describing vector for the spanning tree. GRAPH must be renumbered by the function NEWNOT:

SPT ← OBTADJ CODE DECODE ARBCOV NEWNOT GRAPH

Example:



OBTADJ CODE DECODE ARBCOV NEWNOT ROUE 5

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

### Determination of Elementary Circuits

Function CIRELM implements an algorithm devised by J. C. Terman[9].

ELCIR ← CIRELM GRAPH

Example:

```

                CIRELM SUCCES
            4 5 6
            5 6 7
            7 8 9
            8 9 10
            12 13 14
            12 13 15 16 14
4 5 6 0 5 6 7 0 7 8 9 0 8 9 10 0 12 13 14
0 12 13 15 16 14 0

```

### Number of Spanning Trees In A Wheel

Function NSTW returns the number of spanning trees in a wheel of order N.

NUMBER ← NSTW N

The algorithm is described by B. R. Myers[5].

In fact, two different functions are used, one is recurrent (NSTW), the other not (WSTN):

Example:

```

                WSTN 3
16
                WSTN 4
45
                WSTN 5
121
                WSTN 3
16
                WSTN 4
45
                WSTN 5
121

```

### Graph Decomposition Into Connected Components

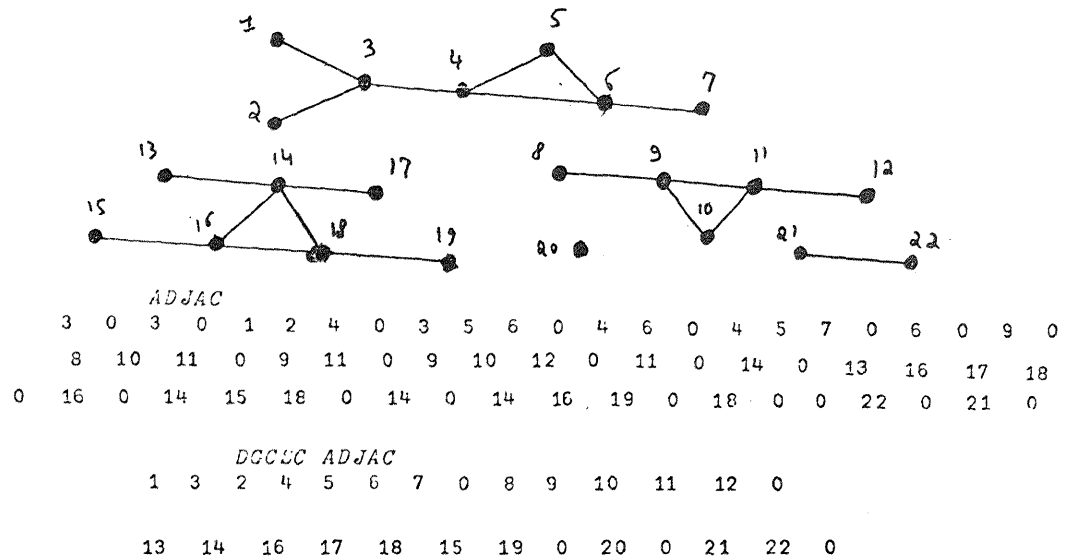
Reduction of graphs into their weakly-connected components is performed with the function DGCSC which uses a classical procedure. The set of vertices connected to vertex 1 is first

derived and extracted from the initial graph, then the procedure is iterated on the remaining graph.

WCCOMP ← DGCSC GRAPH

WCOMP is a vector formed by the sets of indices of the different components delimited by zeros.

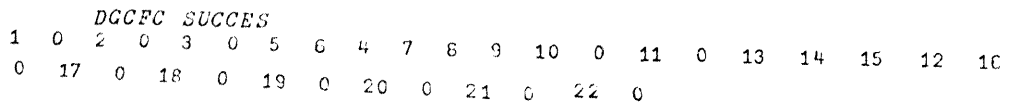
Example:



Graph Decomposition Into Strongly Connected Components

Graphs are reduced in their strongly connected components by the function DGCCFC which uses a procedure similar to that described above:

Example:



Determination of Pendent Vertices

Indices for pendent vertices appear only once in the describing vector

$$PENVER \leftarrow SOMPEN \quad GRAPH$$

PENVER is a vector whose components are the indices of pendent vertices.

Determination of Rooted Trees

Vertices belonging to rooted trees are determined by considering first pendent vertices and then determining paths from them to the roots. For convenience, roots are not included in their corresponding trees but are considered as articulation points. Their determination is performed by the function REARBO:

input argument:           describing vector

result:                    a vector containing the sets of indices of the different rooted trees.

In the following, graphs are assumed to be connected.

#### Determination of Cycles

Independent cycles are determined while building a spanning tree. Rooted trees are first detected and deleted from the graph:

input data:	vectors describing respectively the graph and its possible rooted trees.
result:	a describing vector for independent cycles whose number is equal to the graph cyclomatic number.

#### Determination of Lobes

A lobe (2-connected component) is a set of cycles in which two cycles share a common edge. Lobes are determined with the function RELOBE whose input argument is the cycle-describing vector and whose output is the lobe-describing vector.

#### Determination of Cut-Edges

Cut-edges (bridges) are edges which belong neither to a lobe nor to a rooted tree. Cut-edges sharing a common vertex are considered as a single one.

#### Determination of Cut-Vertices

This section is concerned with the determination of cut-vertices between two lobes, one lobe and one rooted tree, or one rooted tree and one cut-edge. Cut-vertices within a rooted tree or a cut-edge are not considered here. Function REPOAR is used for this purpose.

For each cut-vertex three sets of data are provided as follows:

- a. cut-vertex between two lobes:
  1. index of the cut-vertex
  2. & 3. ranks of the two lobes in the lobe describing vector.
- b. cut-vertex between a lobe and a rooted tree:
  1. index for the cut-vertex
  2. rank of the lobe in the lobe-describing vector.
  3. negative of the rank of the corresponding rooted tree in the rooted-tree-describing vector.
- c. Cut-vertex between a rooted tree and a cut-edge:
  1. index of the cut-vertex
  2. negative of the rank of the corresponding cut-edge in the cut-edge-describing vector
  3. rank of the corresponding rooted tree in the rooted-tree-describing vector.

This way of representing cut-vertices (or articulation points) is convenient for finding the components they connect.

Several of the above functions are gathered in a single function, DECOMP, for the reduction and the determination of characteristic elements in a graph.

An example is detailed in Appendix 2.

Eulerian Circuits

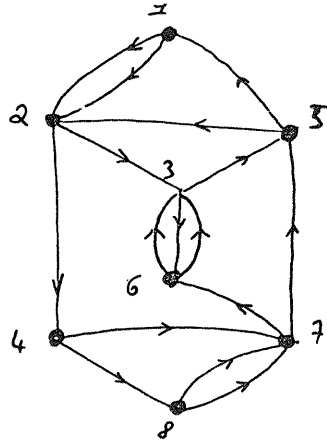
Eulerian circuits are determined through a "minirecoil" procedure [7,8] implemented by function EULER.

In the case where the graph under consideration is not Eulerian it is modified by duplicating a minimum number of edges in the graph. Function ACHEM performs this operation.

data input for EULER: graph describing vector  
 result: a message indicating whether the given graph is Eulerian or not.

If not, the list of the paths to be duplicated is printed out. The Eulerian circuit is described by a vector formed by the indices of the vertices, given in the order they are encountered along the circuit.

Example:



```

EULER [←V
2 2 0 4 3 0 5 6 0 7 8 0 1 2 0 3 3 0 5 6 0 7 7 0

LE GRAPHE N'EST PAS EULERIEN; CHEMINS AJOUTES :
2 4
3 5 1
7 5 2 4 8

CIRCUIT EULERIEN :
1 2 4 8 7 5 2 4 8 7 5 2 4 7 6 3 6 3 5 1 2 3 5 1
  
```

Decomposition Into 2- and 3-Connect Components

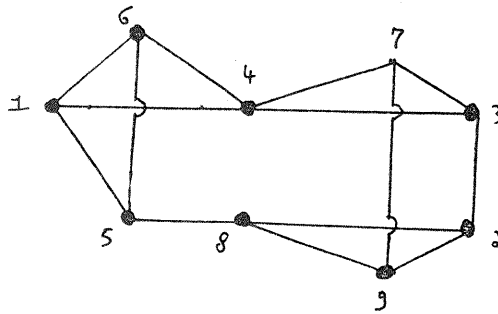
This decomposition takes advantage of results, established by Kleitman[4], for minimizing the number of pairs of vertices for which either two- or three-vertex disjoint paths are sought.

The graph is 2-connected (or 3-connected) if these two (or three) disjoint paths are found. In the case that no such paths exist, the cut-vertex (or cut-set) linking two sets of 2-connected (or 3-connected) components is provided.

This procedure is then iterated on the two resulting sets.

A special labeling procedure is performed to find the vertex disjoint paths[3].

Example:



```

CODE DEC3C []+A1
4 5 6 0 3 8 9 0 2 4 7 0 1 3 5 6 7 0 1 4 6 8 0
1 4 5 0 3 4 9 0 2 5 9 0 2 7 8 0
COMPOSANTE 3-CONNEXE: 1 4 5 6
COMPOSANTE 3-CONNEXE: 2 3 4 7 8 9
ELEMENT NON 3-CONNEXE: 4 5 8

```

7 - Conclusion

The use of this set of functions, which is constantly undergoing improvement and extension, is illustrated in Appendix 1.

It has been proved to be very useful due to modularity, extensibility and interaction capabilities provided by the APL system.

Interaction is desirable for problems which cannot reasonably be solved in a fully automatic manner. This is the case for problems encountered in graph theory.

Until now, however, in this study APL has been considered more as a tool for establishing rapidly and economically the merits of different algorithms dealing with graphs.

As soon as an algorithm or a set of algorithms are declared suitable, they are turned over to professional programmers for translation into another language (mainly FORTRAN) in order to produce a more efficient program which is of easier access to the whole engineering community.

At the present time this system is intended for development purposes; but with the spread of APL and the imminent availability of computing systems built around APL this situation may be reversed. In this case the use of such design automation tools could be contemplated at any stage in the design process.

The transfer of algorithms from the designer to the engineering program developer usually requires no flowcharting. The APL listing itself is considered here as a reference document.

We feel also that the use of APL language could be extended as a convenient vehicle for communication. We suggest generalizing its use to the formal description of algorithms dealing with graphs.

ACKNOWLEDGMENTS

The authors wish to express their gratitude to Mr. P. Rosenstiehl of l'Ecole Pratique des Hautes Etudes (PARIS) for his advice and many stimulating discussions, and to the SESCOSEM Company for their financial support.

BIBLIOGRAPHY

1. C. Berge. The Theory of Graphs. Methuen and Co.: London, 1962.

2. C. Berge. Graphes et Hypergraphes (DUNOD), Paris, 1970.
3. I. T. Frisch. "An algorithm for vertex-pair connectivity", Int. Jl. Control, 1967, Vol. 6, No. 6, p. 579-593.
4. D. J. Kleitmann. "Method for investigating connectivity of large graphs", IEEE Trans. on Circuit Theory, May 1969, p. 232-233.
5. B. R. Myers. "Number of spanning trees in a wheel", IEEE Trans. on Circuit Theory, Marcy 1971, p. 280-82.
6. O. Ore. Theory of Graphs. American Mathematical Society: Providence, R. I., 1962.
7. P. Rosenstiehl. Graphes leurs vecteurs et leurs mots. Cours a l'Ecole Polytechnique, Paris, (avec la collaboration de F. Moniez et J. C. Bermond).
8. P. Rosenstiehl. "Labyrinthologie Mathematique" Mathematiques et Sciences Humaines (9eme annee No. 33, 1971, p. 5-32.
9. J. C. Terman. "An efficient search algorithm to find the elementary circuits of a graph", Com. of the ACM, Vol. 13, No. 12, p. 722-726, December 1970.

APPENDIX 1

APL Functions

Statistics showing the frequencies of occurrence for the different APL operators is given below. It has been used for a quantitative comparison between APL and FORTRAN programs performing the same operation.

OPERATOR	NUMBER OF TIMES OPERATOR OCCURS	FREQUENCY
----------	---------------------------------	-----------

OPERATEUR	NOMBRE D'OCCURRENCES	FREQUENCE
+	528	20.5527
,	293	11.4052
/	241	9.38108
[	233	9.06968
+	181	7.04554
p	177	6.88984
→	167	6.50058
=	128	4.98248
v	116	4.51538
l	68	2.64694
-	63	2.45232
x	58	2.25769
e	53	2.06306
~	37	1.44025
:	35	1.3624
<	33	1.28455
>	31	1.2067
≠	31	1.2067
⌈	15	0.583885
≥	14	0.544959
∧	12	0.467108
⌊	12	0.467108
⌋	11	0.428182
.	7	0.27248
v	5	0.194628
⌊	5	0.194628
⊕	3	0.116777
⊖	3	0.116777
∖	2	0.0778513
	2	0.0778513
÷	1	0.0389257
≤	1	0.0389257
⌊	1	0.0389257
∘	1	0.0389257
∕	1	0.0389257
*	0	0
⊗	0	0
!	0	0
∧	0	0
⊙	0	0
⊥	0	0
T	0	0

```

VACHEM;D;DN;V;H;I;J
[1] D←1I←0
[2] →(2+I=[/M[;1]),D←D,(+/M[;2]=I)-+/M[;1]=I+I+1
[3] →(Λ/D=0)/14
[4] (1 5)ρ' '; 'LE GRAPHE N'EST PAS EULERIEN; CHEMINS AJOUTES : '
[5] DN←(D<0)/1ρD
[6] V←(H+(D>0)/1ρD),J←0
[7] →(√/DN∈H+TTRIC(M[;1]∈H)/M[;2])/9
[8] →7,V←H,0,V
[9] H←I+((DN∈H)/DN)[1]
[10] →(10+J=+/V=0),H←(I+((I∈V PR J←J+1)/I+(I=M[;2])/M[;1])[1]),H
[11] D[H[1,ρH]]←(D[H[1]]-1),D[H[ρH]]+I+1
[12] →(12+(I+I+1)=ρH),M←((1+(ρM)[1]),2)ρ(,M[ΛJ;],H[I],H[I+1],,M[J+1(ρM)[1]-J←1+M[;1]ΛH[I];)
[13] →(5×~Λ/0=D),T←T,(1+ρ[]+H)ρ0
[14] (1 5)ρ' '; 'LE GRAPHE EST EULERIEN.'
[15] ∇

```

```

VZ←V ADJA N
[1] Z←TTRIC(V PR N),V SN N
[2] ∇

```

```

VZ←N AJOUTA V;A;B
[1] →(2×~0∈N),Z←V
[2] Z←((V[B]-1)ρZ),A,((1+V[B])DROP(V[A]-1)ρZ),(B+[/N],(1+(V+(Z=0)/1ρZ)[A+[/N])DROP Z
[3] ∇

```

```

VZ←N AJOUTS V;I;K
[1] K←1++/V=I←0
[2] →((I+I+1)>ρN)/4
[3] →2,V←V MODIFY N[I],K,V PR N[I]
[4] Z←V,N,0
[5] ∇

```

```

VZ←ARBCOV W;I;A
[1] I←1+Z←0
[2] Z←Z,([/(A<I)/A+W PR I←I+1]),0
[3] →2×I<+/W=0
[4] ∇

```

```

VM←ARET V;A;I;F
[1] F←+/V=I+ρM←10
[2] →(2+I≥F),M←M,(I×A=0)+A+((2×ρA)ρ0,1)\A+V PR I←I+1
[3] M←((-F-ρV),2)ρM
[4] ∇

```

```

VM←ARETE V;A;B;I;F
[1] F←+/V=I+ρM←10
[2] →(2+I≥F),M←M,(I×B=0)+B+((2×ρA)ρ0,1)\A+TTRIC V PR I←I+1
[3] M←((-F-ρV),2)ρM
[4] ∇

```

```

VZ←V ASCEND N;I;A
[1] →(0=ρZ←V SN N)/I←0
[2] →2×I=ρZ←Z,(~A∈Z)/A+V SN Z[I+I+1]
[3] ∇

```

```

VZ+NC CHEMIN V;J;T;TT;SME;SMS;VOIS;I;L
[1] →(NC≥+/V=0)/ρZ+L+10
[2] I+I1|/I++/M[;1]°.=1|/(M+ARETE V)[;1+J+0]
[3] MARK+((ρSFLOT),2)ρAFLOT+(ρM)[1]ρSFLOT+(|/M[;1])ρT+TT+0
[4] →((J>ρSFLOT),I=J+J+1)/9 4
[5] MARQUE I,J
[6] →(T=TT)/11
[7] TT+T
[8] →((NC=TT),1)/3 5
[9] →10×0≠NC+NC-1
[10] →2,(L+L,I++/L≤I),V+I ENLEVS V
[11] SME+(MARK[;1]≠0)/1(ρMARK)[1]
[12] SMS+(MARK[;2]≠0)/1(ρMARK)[1]
[13] Z+TTRIC((~SME∈SMS)/SME),(~VOIS∈SME)/VOIS+(M[;1]∈SMS)/M[;2]
[14] →(0=ρL)/0
[15] Z+2ρL,Z+(1≠ρL)×Z≥|/L)+Z+Z+Z≥|/L
[16] V

```

```

VZ+CIRELM G;P;H;I;J;K;GPRK
[1] P+Nρ(K+1),H+(N++/G=0)ρJ+ρZ+10
[2] →((GPRK[J]>P[1])^(~GPRK[J]∈P)^(~(GPRK+(G PR P[K]),0)[J+J+1]∈H PR P[K]))/10
[3] →(J<ρGPRK)/2
[4] →(~P[1]∈GPRK)/6
[5] Z+Z,(|+(P≠0)/P),0
[6] →(K=1)/12
[7] H+H MODIFY,P[K]
[8] H+H MODIFY(P[K-1]),(H PR P[K-1]),P[K]
[9] →2,K+K-1+J+P[K]+0
[10] K+K+1
[11] →2,(J+0),P[K]+(G PR P[K-1])[J]
[12] →(P[1]=N)/0
[13] →2,(P[1]+P[1]+K+1),H+NρJ+0
[14] V

```

```

VZ+V COFCO N
[1] Z+((~N∈Z)/N),Z+(Z∈V ASCEND N)/Z+V DESCEN N
[2] V

```

```

VZ+V COMCO I;K;A
[1] Z+,I+K+0
[2] →(K<ρZ+Z,(~A∈Z)/A+V PR Z[K+K+1])/2
[3] V

```

```

VZ+N CONTRA V;A
[1] V+V MODIFY N[1],((A≠N[2])/A+V PR N[1]),(A≠N[1])/A+Z+V PR N[2]
[2] Z+((A-1+ρZ)ρV),(A+(V=0)/1ρV)[N[2]]DROP V+(V×V≠N[2])+(N[1]×V=N[2])-V>N[2]
[3] V

```

```

VCYCLE V;T;W
[1] W[W1T+1|/W+V]+0
[2] +3+0=ρT+(~T∈W)/T
[3] +1+3×1≥ρV+T ENLEVS V
[4] (25×0=ρT)DROP(25+36×0=ρT)ρ'LE GRAPHE EST SANS CYCLE.LE GRAPHE POSSEDE AU MOINS UN CYCLE.'
[5] V

```

```

VZ+M DECODE V
[1] Z+M[1;M[2;]1V]
[2] V

```

```

VDECOMP V
[1] ''
[2] '
[3] ' *****'
[4] ' * ELEMENTS REMARQUABLES DU GRAPHE *'
[5] ' *****'
[6] ARBO←REARBO V
[7] LOBES←RELOBE RECYCL V
[8] PONTS←REPOINT V
[9] POAK←REPOAR V
[10] ''
[11] ' *****'
[12] ' * ENCHAINEMENT DES ELEMENTS REMARQUABLES DU GRAPHE *'
[13] ' *****'
[14] ''
[15] DECRIR
[16] V

VDECRIR;I;A;V;W1;W2
[1] I←0
[2] E1:→((I+I+1)>+/POAR=0)/E2
[3] →(A[2]<0),(A+POAR PR I)[3]<0)/E9,E3
[4] W1←' AU LOBE '
[5] →E4,W2←LOBES PR A[3]
[6] E3:W1←' A L'ARBORESCENCE ';W2←ARBO PR-A[3]
[7] E4:→E1,[←'LE POINT D'ARTICULATION ';A[1];' RELIE LE LOBE ';LOBES PR A[2];W1;W2
[8] E9:→E1,[←'LE POINT D'ARTICULATION ';A[1];' RELIE LE PONT ';PONTS PR-I[2];' A L
[9] E2:I←0 'ARBORESCENCE ';ARBO PR A[3]
[10] E5:→((I+I+1)>+/PONTS=0)/ρW1←V←0
[11] A←PONTS PR I
[12] E6:→((V+V+1)>ρA)/E7
[13] →E6,W1←W1,(~W2∈W1)/W2←LOBES SN A[V]
[14] E7:V←1
[15] E8:→((V+V+1)>ρW1)/E5
[16] →E8,[←'LE PONT ';A;' RELIE LE LOBE ';LOBES PR W1[1];' AU LOBE ';LOBES PR W1[V]
[17] V

VCK DEC2C V;I;A;K;C1
[1] →((I+0)=ρ,A+2 CHEMIN V)/5
[2] C1←C1+(C1←DGCSC A ENLEVS V)≥A
[3] CK[K]DEC2C(K←TTRIC A,C1 PR I+I+1)SSGMAX V
[4] →3×I<+/C1=0
[5] (22×2≥ρCK)DROP(22+23×2≥ρCK)ρ'COMPOSANTE 2-CONNEXE: ELEMENT NON 2-CONNEXE: ';CK
[6] V

VCK DEC3C V;I;A;K;C1
[1] →((I+0)=ρ,A+3 CHEMIN V)/5
[2] C1←(C1≥I/A)+C1+C1+(C1←DGCSC A ENLEVS V)≥I/A
[3] CK[K]DEC3C(K←TTRIC A,C1 PR I+I+1)SSGMAX(A×~A[1]∈V PR A[2])AJOUTA V
[4] →3×I<+/C1=0
[5] (22×3≥ρCK)DROP(22+23×3≥ρCK)ρ'COMPOSANTE 3-CONNEXE: ELEMENT NON 3-CONNEXE: ';CK
[6] V

VZ←DEGRAF;ORIENT;I;SOM;FIN;J;SOMMET;A
[1] 20ρ' ';23ρ'*'
[2] 20ρ' ';*DESCRIPTION DU GRAPHE*'
[3] 20ρ' ';23ρ'*'
[4] ''
[5] 'LE GRAPHE EST-IL ORIENTE ? (REPONDRE OUI OU NON)'
[6] DON1:ORIENT←3ρ□
[7] →((^/ORIENT='OUI')∨^/ORIENT='NON')/SUITE1
[8] COR1:→DON1,[←'MAUVAISE REPONSE, RECOMMENCER'
[9] SUITE1:ORIENT←^/ORIENT='OUI'
[10] ''
[11] 20ρ' ';23ρ'*';(4×1-ORIENT)ρ'*';8ρ'*'
[12] 20ρ' ';*DESCRIPTION DU GRAPHE ';(4×1-ORIENT)ρ'NON ';'ORIENTE*'
[13] 20ρ' ';23ρ'*';(4×1-ORIENT)ρ'*';8ρ'*'
[14] ''
[15] 'LISTE DES ';(12×ORIENT)ρ'SUCESSEURS ';(18×1-ORIENT)ρ'SOMMETS ADJACENTS '

```

```

[16] ''
[17] '          INSTRUCTIONS '
[18] '          -----'
[19] 'EN CAS D'ERREUR DE DESCRIPTION POUR LE SOMMET I,TAPER:  SOMMET,I  A LA PLACE'
[20] 'DE LA DESCRIPTION D'UN SOMMET ULTERIEUR.'
[21] 'POUR TERMINER LA DESCRIPTION TAPER: FIN'
[22] '          -----'
[23] ''
[24] Z+ \FIN+SOMMET+I+J+0
[25] SUITE2: 'SOMMET ';I+I+1
[26] SOM+,[]
[27] +(0=ρSOM)/TEST1
[28] →MODIF×\J
[29] →SUITE2,Z+Z,0
[30] TEST1:→((1=ρSOM)∧SOM[1]=0)/TESTF
[31] →(SOM[1]=0)/SUITE3
[32] →MODIF×\J
[33] →SUITE2,Z+Z,SOM,0
[34] SUITE3:J+1
[35] I+SOM[2]-1
[36] →SUITE2
[37] MODIF:Z+Z MODIFY,I,SOM
[38] J+0
[39] I++/Z=0
[40] →SUITE2
[41] ''
[42] TESTF:'NOMBRE DE SOMMETS DU GRAPHE : ';I-1
[43] +(ORIENT=1)/0
[44] TESTCO:A+VERIFY Z
[45] →(0=ρA)/I+0
[46] SUITE5:'SOMMET ';A[I+I+1]
[47] Z+Z MODIFY A[I],[]
[48] →(I<ρA)/SUITE5
[49] →TESTCO
[50] ∇

∇Z+V DEMDEG A;I
[1] Z+2ρI+0
[2] →(2×I<ρA),(Z[2]+Z[2]++/~(V PR A[I])εA),Z[1]+Z[1]++/~(V SN A[I+I+1])εA
[3] ∇

∇Z+V DESCEN N;I;A
[1] →(0=ρZ+V PR N)/I+0
[2] →2×I=ρZ+Z,(~AεZ)/A+V PR Z[I+I+1]
[3] ∇

∇Z+DGCFC V;I
[1] Z+ \I+0
[2] →((I>+/V=0),(I+I+1)εZ)/0 2
[3] →2,Z+Z,(V COFCO I),0
[4] ∇

∇Z+DGCSC V;I
[1] Z+ \I+0
[2] →((I>+/V=0),(I+I+1)εZ)/0 2
[3] →2,Z+Z,(V COMCO I),0
[4] ∇

∇Z+N DROP V
[1] Z+((Nρ0),(((ρV)-N)ρ1))/V
[2] ∇

∇Z+N ENLEVA V;A
[1] Z+(V MODIFY N[1],(A=A[A \N[2]]+0)/A+V PR N[1])MODIFY N[2],(A=A[A \N[1]]+0)/A+V PR N[2]
[2] ∇

∇Z+N ENLEVS V;I
[1] Z+(~VεN+TTRIC,N)/V+I+0
[2] →(2×I=ρN+N-1),Z+((V-1+ρZ PR N[I])ρZ),(V+((Z=0)/\ρZ)[N[I]])DROP Z+Z-N[I+I+1]<Z
[3] ∇

```

```

∇Z←EULER V;I;T;M;L
[1] T←(ρM←ARET V)[L←Z←,1]ρ0
[2] ACHEM
[3] →(0∈T[I←(Z[1]=M[;2])]/ι(ρM)[1]])/5
[4] →3,(Z[2]←-Z[2]),(T[I]←1),L←L,(Z←M[I←((0=T[I])/I)[1];1],Z)[1]
[5] →(1=ρL)/8
[6] Z←L[I←1+ρL],Z
[7] →3,L←IρL
[8] (1 5)ρ' '; 'CIRCUIT EULERIEN :
[9] Z←φ(Z>0)/Z
[10] ∇

∇Z←GOM V;K
[1] →(0=ρV←,V)/ρZ←ιK←0
[2] →(2×K<ρV),Z←Z,(~V[K]∈Z)/V[K←K+1]
[3] ∇

∇Z←M INDIC V
[1] Z←(MΛ.=V)/ι(ρM)[1]
[2] ∇

∇Z←LS N
[1] →(N=1 2)/3 4
[2] →0,Z←(LS N-1)+LS N-2
[3] →0,Z←1
[4] Z←3
[5] ∇

∇MARQUE V;SM;MT;R;RT;P;I
[1] MARK[SM;2]←SM←V[I←1]
[2] MT←φ(7,ρR)ρ(,φM[R;]),(,φMARK[M[R;2];]),SFLOT[M[R;2]],AFLOT[(R←(M[;1]=SM[I])/ι(ρM)[1])],
[3] →((P←MARK[SM[I];]≠0)[2]=0)/5
[4] →(5+P[1]=0),MT[;3]←MT[;3]+MT[;1]×Λ/MT[;3 6]=0 RT←(M[;2]=SM[I])/ι(ρM)[1]
[5] MT[;4]←MT[;4]+MT[;1]×</MT[;4 7]≠0
[6] →7,(SM←SM,MT[(v/MARK[M[R;2];]≠MT[;3 4])/ι(ρMT)[1];2]),(MT[;3]←MT[;3]+MT[;2]×2=-/MT[;
MT[;4 3 5]=0
[7] →(8+v/MARK[V[2];]≠0),MARK[M[R;2];]←MT[;3 4] 4 3 5≠0),MT[;4]←MT[;4]+MT[;2]×2=-/
[8] →(2+7×(ρSM)<I←I+1)
[9] P←I+V[2]
[10] →(0=ρI←(~MARK[I;]∈0,I)/MARK[I;])/0
[11] →(10+2×I=V[1]),P←P,I←I[1]
[12] T←T+I+1,0ρSFLOT[I]←~SFLOT[I←(~P∈V)/P+φP]
[13] →(AFLOT[R←M INDIC V←P[I,I+1]]=1)/16
[14] →(AFLOT[RT←M INDIC φV]=0)/16
[15] →17,AFLOT[R,RT]←SFLOT[V]←0
[16] AFLOT[R]←~AFLOT[R]
[17] →((ρP)>I←I+1)/13
[18] MARK←(ρMARK)ρ0
[19] ∇

∇Z←W MITOSE V;W1;U;N;I
[1] Z←(V MODIFY W1,W,N←1++/V=I←0),(U←(~U∈W←K[1+ι1+ρV])/U←V PR W1),(W1←W[1]),0
[2] →((I←I+1)>ρU)/0
[3] →2,Z←Z MODIFY U[I],N,(W≠W1)/W←V PR U[I]
[4] ∇

∇Z←W MODIFY NV;M
[1] Z←((M-1+ρW PR NV[1])ρW),(1 DROP NV),0,(M←((W=0)/ιρW)[NV[1]])DROP W
[2] ∇

∇Z←NEWNOT W;I;J
[1] Z←CODE←,W;I←1
[2] Z←(I,J)PERL(I,J←((Z≥I+I+1)/Z)[1])PERB Z
[3] CODE←(I,J)PERB CODE
[4] →(I←+/Z=0)/2
[5] CODE←(2,1+ρCODE)ρ0,(CODE←GOM(CODE≠0)/CODE),0,GOM(Z≠0)/Z
[6] ∇

∇Z←NSTW N
[1] Z←((LS N)*2)+4×1+2|N
[2] ∇

```

```

VZ←OBTADJ V;I
[1] Z←1I+0
[2] →(2×I<+/V=0),Z+Z,(TTRIC(V PR I),V SN I+I+1),0
[3] ∇

VZ←OBTPRE V;I
[1] Z←1I+0
[2] →(2×I<+/V=0),Z+Z,(V SN I+I+1),0
[3] ∇

VZ←OBTSUC V;I
[1] Z←1I+0
[2] →(2×I<+/V=0),Z+Z,(V SN I+I+1),0
[3] ∇

VZ←N PERB W;V
[1] V←(W=0)/1ρZ+W
[2] →(1=ρN+TTRIC N)/0
[3] Z←(V[N[1]-1]ρW),(W PR N[2]),0,((V[N[2]-1]-V[N[1]])ρV[N[1]] DROP W),(W PR N[1])
[4] ∇,0,V[N[2]] DROP W

VZ← N PERL W
[1] Z←W
[2] →(N[1]=N[2])/0
[3] Z←(N[2]×W=N[1])+(N[1]×W=N[2])+W×~W∈N
[4] ∇

∇W+V PP N;Z
[1] →(((Z+(V=0)/1ρV)1N)≠1)/3
[2] →0,W+(N-1)ρV
[3] W+Z[(Z1N)-1]DROP(N-1)ρV
[4] ∇

∇W+V PR N;Z
[1] →(N≠1)/3
[2] →0,W+(((V=0)/1ρV)[1]-1)ρV
[3] W+Z[N-1]DROP((Z+(V=0)/1ρV)[N]-1)ρV
[4] ∇

VZ←REARBO V;W;I;A;B;K
[1] W←SOMPEN V,Z←1I+K+0
[2] →((K+K+1)>ρW)/6
[3] →(0=ρB+(~B∈W)/B+V PR W[K])/2
[4] →(1<ρ(~A∈W)/A+V PR B)/2
[5] →2,W+W,B
[6] →((I+I+1)>ρW)/K+0
[7] →((B+,W[I])∈Z)/6
[8] →(8+K=ρB),B+B,(~A∈B)/A+(A∈W)/A+V PR B[K+K+1]
[9] →6,(Z+Z,B,0),□+'ARBORESCENCE : ';B
[10] ∇

VZ←RECYCL V;K;I;A;N;NC;F;ARB;AA;TREE
[1] →(0=NC+1+((+/V≠0)÷2)-+/V=0)/ρZ←TREE←1N+I+K+0
[2] A←(~A∈ARBO)/A+V PR ARB←,((~A∈F+(ARBO≠0)/ARBO)/A+1+/V=0)[1]
[3] →(A[I+I+1]∈ARB)/5
[4] →3,(I+0),(A←(~A∈F,ARB[-1+ρARB])/A+V PR A[I]),ARB←ARB,A[I]
[5] →((NC>N+N+1)×6-3×I<ρA),(Z+Z,AA,0),□+'CYCLE : ';AA←(-1+ARB1A[I])DROP ARB
[6] →(0=ρA+(~A∈ARB,F,TREE)/A+V PR ARB[(ρARB)-K+K+1])/6
[7] →3,(I+K+0),(ARB←((ρARB)-K)ρARB),TREE←TREE,KρΦARB
[8] ∇

```

```

    VZ+RELOBE V;K;J;U;W
[1] K+0
[2] K+K+1
[3] →((J+K)>+/V=0)/10
[4] W+V PR K
[5] →((J+J+1)>+/V=0)/2
[6] →(2>+/WεU+V PR J)/5
[7] →8+K=1
[8] →3,V+(Z[K-1]ρV),(Z[K]DROP Z[J-1]ρV),(GOM W,U),((Z+(V=0)/1ρV)[J]-1)DROP V
[9] →3,V+(Z[1]DROP Z[J-1]ρV),(GOM W,U),((Z+(V=0)/1ρV)[J]-1)DROP V
[10] Z+V+K+0
[11] →((K+K+1)>+/Z=0)/0
[12] →11,□←'LOBE : ';Z PR K
[13] V

```

```

    VZ+REPOAR V;A;I;K;B;C;J
[1] Z+1I+K+0
[2] →((I+I+1)=+/LOBES=C)/7
[3] A+LOBES PR K+I
[4] →((K+K+1)>+/LOBES=0)/2
[5] →(1≠ρB+(AεC+LOBES PR K)/A)/4
[6] →4,Z+Z,B,I,K,0
[7] I+0
[8] →((I+I+1)>+/ARBO=0)/K+0
[9] A+ARBO PR I
[10] →((K+K+1)>ρA)/8
[11] →(0=ρB+(~BεARBO)/B+V PR A[K])/10
[12] →((J+0)=ρC+LOBES SN B)/15
[13] →((J+J+1)>ρC)/8
[14] →13,Z+Z,B,C,-I,0
[15] →8,Z+Z,B,(-PONTS SN B),I,0
[16] V

```

```

    VZ+REPONT V;I;A;B;K;U
[1] Z+1I+0
[2] →((I+I+1)>+/V=0)/9
[3] →(IεARBO)/2
[4] →(0=ρB+LOBES SN I)/2
[5] U+1K+0
[6] →(6+K=ρB),J+U,LOBES PR B[K+K+1]
[7] →(0=ρA+(~AεARBO,U)/A+V PR I)/2
[8] →2,Z+Z,I,A,0
[9] Z+TRI Z
[10] V

```

```

    VZ+ROUE N;I
[1] Z+(1+1N),0,1,(N+1),(1+I+2),0
[2] →(2+I=N),Z+Z,1,(1+I+I+1),I,0
[3] Z+Z,1,I,2,0
[4] V

```

```

    VH+SL N;Z;I
[1] Z+1,3;I+2
[2] Z+Z,Z[I-2]+Z[-1+I+I+1]
[3] →2×1I<N
[4] H+Z[N]
[5] V

```

```

    VZ+V SN N;I;A
[1] →(0=ρA+(V=N)/1ρV)/ρZ+1I+0
[2] →(2×I<ρA),Z+Z,1++/0=V[1A[I+I+1]]
[3] V

```

```

    VZ+SOMPEN V
[1] V[V11[V/V]+C
[2] Z+(~(1[V]εV)/1[V/V
[3] V

```

```

∇Z←V SR N;U;A
[1] →(0=ρA+(V=N)/1ρV)/ρZ+10
[2] →(2×0≠ρA+(~A∈U)/A),Z←Z,U←(0=V[A])/A+A+1
[3] ∇

∇Z←G SSGMAX V;J
[1] V←(V∈G,0)/V,Z+1I+0
[2] →((I+I+1)∈G)/4
[3] →(2×I<⌈/G),(V+V-V>+/Z=0),Z+Z-Z>+/Z=0
[4] →(2×I<⌈/G),Z+Z,(V PR I),0
[5] ∇

∇Y←TRI V;K;J;Z;W;U
[1] Y←V+K+0
[2] K←K+1
[3] →((J+K)>+/Y=0)/0
[4] W←Y PR K
[5] →((J+J+1)>+/Y=0)/2
[6] →(0=V/W∈U+Y PR J)/5
[7] →(K=1)/9
[8] →3,Y←(Z[K-1]ρY),(Z[K]DROP Z[J-1]ρY),(GOM W,U),((Z+(Y=0)/1ρY)[J]-1)DROP Y
[9] →3,Y←(Z[1]DROP Z[J-1]ρY),(GOM W,U),((Z+(Y=0)/1ρY)[J]-1)DROP Y
[10] ∇

∇Z←TRIC W;K
[1] →(0=ρZ+W←,W)/K+0
[2] →(2×K<ρW),Z+((Z<W[K])/Z),((Z=W[K])/Z),(Z>W[K+K+1])/Z
[3] 4  >IH⊙
CHARACTER ERROR
4  >IH⊙
      ^
[3] )CLEAR
      )CARD

∇Z←TRIC W;K
[1] →(0=ρZ+W←,W)/K+0
[2] →(2×K<ρW),Z+((Z<W[K])/Z),((Z=W[K])/Z),(Z>W[K+K+1])/Z
[3] ∇

∇Z←TTRIC W;K
[1] →(0=ρZ+W←,W)/K+0
[2] →(2×K<ρW),Z+((Z<W[K])/Z),W[K],(Z>W[K+K+1])/Z
[3] ∇

∇Z←VECT M;I;A
[1] A←⌈/(,M),Z←1I+0
[2] →(2×I<A),Z+Z,((M[;1]=I+I+1)/M[;2]),0
[3] ∇

∇Z←VERIFY W;I;V1;V2
[1] Z←1I+0
[2] A0:→(^(V1+W PR I)∈V2+W SN I←I+1)/A1
[3] Z←Z,I,(~V1∈V2)/V1
[4] A1:→(^(V2∈V1)/A2
[5] Z←Z,I,(~V2∈V1)/V2
[6] A2:→(I<+/W=0)/A0
[7] →(0=ρZ+TTRIC Z)/A3
[8] →0,[←'LA DESCRIPTION DES SOMMETS ADJACENTS AUX SOMMETS ' ;Z;' EST INEXACTE'
[9] A3:'LA DESCRIPTION DU GRAPHE EST COHERENTE.'
[10] ∇

∇Z←WHEEL V;I;N
[1] N←+/V=I←Z+0
[2] →((N<I),(3=+/V=I←I+1),Z≠0)/5 2 4
[3] →2×0≠Z+(N-1)×(N-1)=+/V=I
[4] →Z+0
[5] Z←(3×(N=4)×Z=0)+(Z≠0)×N-1
[6] ∇

∇Z←WSTN N
[1] Z←((SL N)*2)+4×-1+2|N
[2] ∇

```

EXAMPLES

SUCCES+DEGRAF

\*\*\*\*\*  
\*DESCRIPTION DU GRAPHE\*  
\*\*\*\*\*

LE GRAPHE EST-IL ORIENTE ? (REPONDRE OUI OU NON)  
OUI

\*\*\*\*\*  
\*DESCRIPTION DU GRAPHE ORIENTE\*  
\*\*\*\*\*

LISTE DES SUCCESSEURS

INSTRUCTIONS

-----  
EN CAS D'ERREUR DE DESCRIPTION POUR LE SOMMET I, TAPER: SOMMET, I A LA PLACE  
DE LA DESCRIPTION D'UN SOMMET ULTERIEUR.  
POUR TERMINER LA DESCRIPTION TAPER: FIN  
-----

SOMMET 1

[ ]:

3

SOMMET 2

[ ]:

3

SOMMET 3

[ ]:

4

SOMMET 4

[ ]:

5

SOMMET 5

[ ]:

6

SOMMET 6

[ ]:

4 7

SOMMET 7

[ ]:

5 8

SOMMET 8

[ ]:

9

SOMMET 9

[ ]:

7 10

SOMMET 10

[ ]:

8 11

SOMMET 11

[ ]:

12

SOMMET 12

[ ]:

13

SOMMET 13  
 □: 14 15  
 SOMMET 14  
 □: 12  
 SOMMET 15  
 □: 16 17  
 SOMMET 16  
 □: 14 18  
 SOMMET 17  
 □: 10  
 SOMMET 18  
 □: 10  
 SOMMET 19  
 □: 20  
 SOMMET 20  
 □: 22  
 SOMMET 21  
 □: 22  
 SOMMET 22  
 □: 11  
 SOMMET 23  
 □: FIN

NOMBRE DE SOMMETS DU GRAPHE : 22

SUCCES

3 0 3 0 4 0 5 0 6 0 4 7 0 5 8 0 9 0 7 10 0 8 11 0 12  
 17 0 14 18 0 0 0 20 0 22 0 22 0 11 0

0 13 0 14 15 0 12 0 16

[←PREDEC←OBTPRE SUCCES

0 0 1 2 0 3 6 0 4 7 0 5 0 6 9 0 7 10 0 8 0 9 0 10 22  
 13 0 15 0 15 0 16 0 0 19 0 0 20 21 0

0 11 14 0 12 0 13 16 0

□←OBT SUC PREDEC

3 0 3 0 4 0 5 0 6 0 4 7 0 5 8 0 9 0 7 10 0 8 11 0  
 17 0 14 18 0 0 0 20 0 22 0 22 0 11 0

12 0 13 0 14 15 0 12 0 16

□+ADJAC+OBTADJ SUCCES  
 3 0 3 0 1 2 4 0 3 5 6 0 4 6 7 0 4 5 7 0 5 6 8 9 0 7 9 10  
 11 0 10 12 22 0 11 13 14 0 12 14 15 0 12 13 16 0 13 16 17  
 16 0 20 0 19 22 0 22 0 11 20 21 0  
 0 7 8 10 0 8 9  
 0 14 15 18 0 15 0

SUCCES ADJA 11  
 10 12 22

SUCCES ADJA 1  
 3

SUCCES ASCEND 22  
 20 21 19

SUCCES ASCEND 3  
 1 2

SUCCES DESCEN 12  
 13 14 15 12 16 17 18

SUCCES DESCEN 8  
 9 7 10 5 8 11 6 12 4 13 14 15 16 17 18

SUCCES COFCO 3  
 3

SUCCES COFCO 7  
 5 8 6 9 4 7 10

SUCCES COFCO 10  
 8 9 7 10 5 6 4

\*\*\*\*\*  
\* ELEMENTS REMARQUABLES DU GRAPHE \*  
\*\*\*\*\*

ARBORESCENCE : 1 3 2  
ARBORESCENCE : 17  
ARBORESCENCE : 18  
ARBORESCENCE : 19 20 22 21  
CYCLE : 4 5 6  
CYCLE : 5 6 7  
CYCLE : 7 8 9  
CYCLE : 8 9 10  
CYCLE : 12 13 14  
CYCLE : 13 14 16 15  
LOBE : 4 5 6 7  
LOBE : 7 8 9 10  
LOBE : 12 13 14 16 15

\*\*\*\*\*  
\* ENCHAINEMENT DES ELEMENTS REMARQUABLES DU GRAPHE \*  
\*\*\*\*\*

LE POINT D'ARTICULATION 7 RELIE LE LOBE 4 5 6 7 AU LOBE 7 8 9 10  
LE POINT D'ARTICULATION 4 RELIE LE LOBE 4 5 6 7 A L'ARBORESCENCE 1 3 2  
LE POINT D'ARTICULATION 15 RELIE LE LOBE 12 13 14 16 15 A L'ARBORESCENCE 17  
LE POINT D'ARTICULATION 16 RELIE LE LOBE 12 13 14 16 15 A L'ARBORESCENCE 18  
LE POINT D'ARTICULATION 11 RELIE LE PONT 10 11 12 A L'ARBORESCENCE 19 20 22 21  
LE PONT 10 11 12 RELIE LE LOBE 7 8 9 10 AU LOBE 12 13 14 16 15