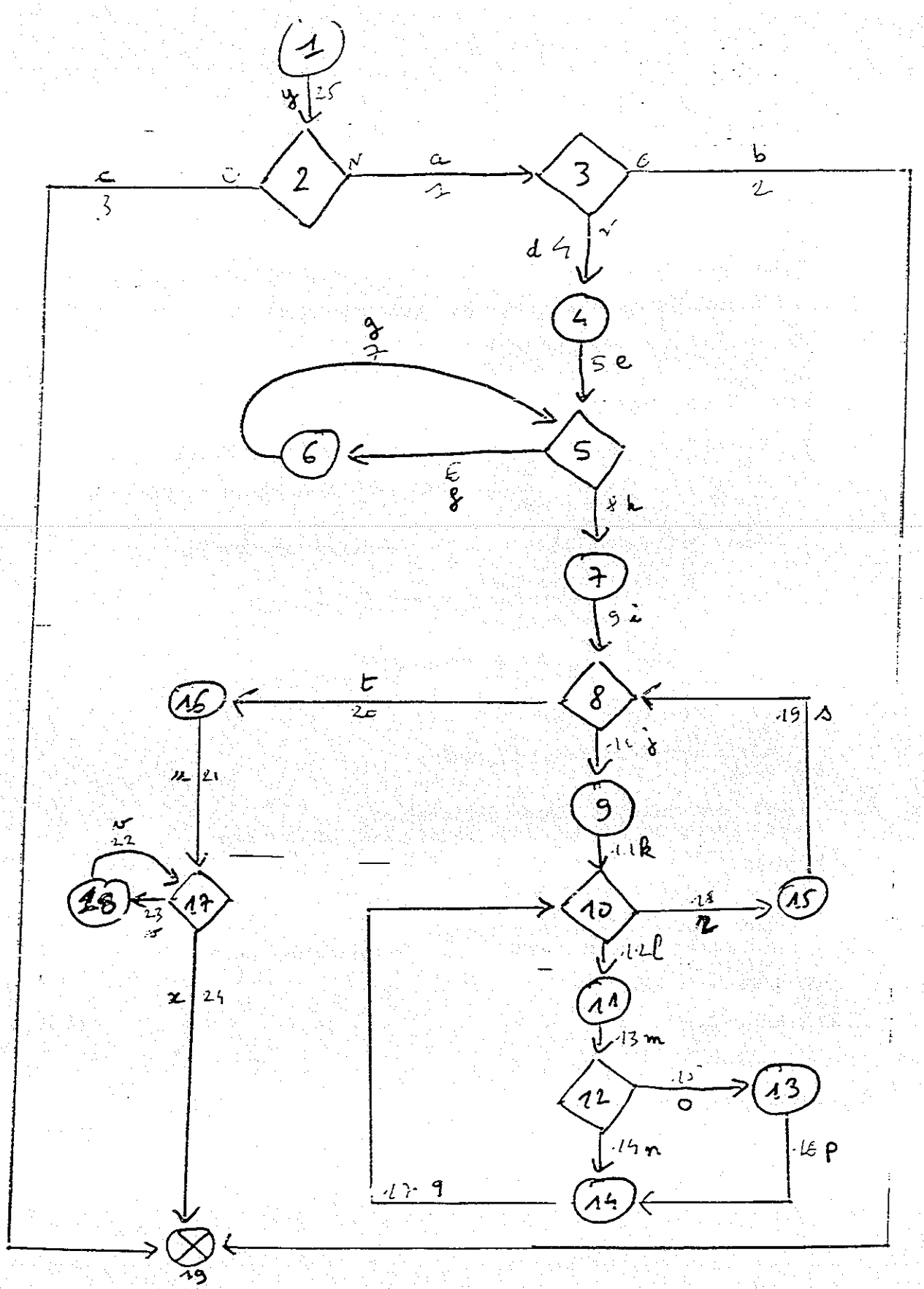
```
      Init(nb_lig,nb_col);          -- dim. effectives requises.
      IF mdest.max_lig<nb_lig OR mdest.max_col<nb_col THEN
          RAISE dim_incomp;
      END IF;
     DECLARE                        -- bloc (sequence+declarations locales)
          ptlig:ptab(base..nb_lig);
     BEGIN
      r_a_z(mdest);                              -- RAZ. matrice resultat.

      -- Initialisations relatives aux tetes de lignes:

      FOR lig IN base..nb_lig LOOP
          ptlig(lig):=ALLOC((NULL,NULL,1,1,zero));
      END LOOP;
      mdest.tetlig:=ptlig;

      p0:=ALLOC((NULL,NULL,1,1,zero));      -- maillon artificiel.
      FOR col IN base..nb_col LOOP
          -- constr. sequent. de la liste des cases de la COLonne
          p:=p0; p.pligsuiv:=NULL;
          Initcol(col,EoCol);
          WHILE NOT Eocol LOOP                    -- parcours COLonne.
                Getval(lig,val);
                IF val /= zero THEN
                   p_1_c:=ALLOC((NULL,NULL,lig,col,val));
                   p.pligsuiv:=p_1_c;
                   p:=p_1_c;
                   ptlig(lig).pcolsuiv:=p_1_c;
                   ptlig(lig):=p_1_c;       -- MAJ @ fin de ligne.
                END IF;
                Avancer_col(Eocol);
          END LOOP;
          mdest.tetcol(col):=p0.pligsuiv;
      END LOOP;

      -- Def. des tetes de lignes definitives,
      -- et liberation des maillons artificiels introduits:

      FREE(p0);
      FOR lig IN base..nb_lig LOOP
          p0:=mdest.tetlig(lig);
          mdest.tetlig(lig):=p0.pcolsuiv;
          ptlig(lig).pcolsuiv:=NULL;        -- marquage F.d.L.
          FREE(p0);
      END LOOP;  -- lig.
     END;
    END constr_seq;

    ---------------

   END matrice_creuse;
```



A = 25
N = 19
p = 2

M = A − N + 2p
= 25 − 19 + 2 = 8

M < 10 ⇒ SIMPLE % Test

CHEMINS	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y
1 y c			X																						X
2 y a b	X	X																							X
3 y a d e h i t u x Matr. Vide : nblig = nbcol = 0	X			X	X			X	X											X	X			X	X
4 y a d e { f g } h i t u { w v } x	X			X	X	X	X	X	X											X	X	X	X	X	X
5 y a d e h i { j k r s } t u x Matr. Vide : nblig = 0, nbcol > 0	X			X	X			X	X	X	X							X	X	X	X			X	X
6 y a d e, { f g } h i j k l m n q r s t u { w v } x. Matr. Non Vide - Nbcol = 1, Val = 0	X			X	X	X	X	X	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X
7 y a d e { f g } h i j k l m o p q r s t u { w v } x. Matr. Non Vide - Nbcol = 1, Val ≠ 0	X			X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X

EXAMEN DE GÉNIE LOGICIEL

Cours BO

session du 12 avril 1996

Tous documents et calculettes sont autorisés.

5 POINTS	Exercice de tests

La procédure ADA suivante (AJOUT-LIGNE, dans le module matrices carées) implémente l'addition de deux lignes dans une matrice creuse (contenant beaucoup de zéros) représentée sous forme de multiliste chaînée. Cette procédure utilise une procédure interne ADD-CASE.

1) Établir son graphe de commande

2) Calculer son nombre cyclomatique. Comment peut-on qualifier la complexité de cette procédure?

3) En déduire une politique de tests permettant une couverture C2 (100% des branches). Préciser la sémantique de chaque donnée utilisée pour le test (par exemple : matrice vide, etc....)

Qu'en conclure par rapport au nombre cyclomatique?

NB : on assimilera un RAISE (susciter une exception) à un RETURN simple

P.J. : le programme Ada (module Matrice Carrées)

```
 1|--*********************************************************
 2|--*   S T R U C T U R E   D E   M A T R I C E   C R E U S E  *
 3|--*********************************************************
 4|
 5|generic
 6|   type ELEMENT is private;
 7|   with function "+" (e1,e2:ELEMENT) return ELEMENT is <>;
 8|   ZERO:ELEMENT;                          -- elt neutre de "+".
 9|
10|package MATRICES_CARREES is
11|
12|   type MATRICE (ORDRE: positive) is limited private;
13|
14|   -- N.B.:Les matrices ont la valeur ZERO initialement.
15|
16|   subtype INDICE is integer range 1..integer'last;
17|
18|
19|     -- Ajout d'une ligne a une autre:
20|     -- -------------------------------
21|        -- Realise: MAT (Ligne,:) := MAT (Ligne,:) + MAT (Plus,:);
22|        -- Leve ERR_MAT si l'indice LIGNE ou PLUS sort de l'intervalle.
23|
24|   procedure AJOUT_LIGNE (MAT:in out MATRICE; LIGNE,PLUS: INDICE);
25|
26|
27|
28|   -- PRIMITIVES GENERIQUES:
29|   -------------------------
30|
31|   -- Application du traitement TRAITER a tous les elts non nuls
32|   -- dans la ligne MAT (LIGNE,:).
33|   -- TRAITER recupere la COLonne de l'elt, et sa valeur:
34|   -- MAT_l_c = MAT (ligne,col) /= Zero).
35|
36|   generic
37|     with procedure TRAITER (col:INDICE; Mat_l_c: ELEMENT);
38|   procedure TRAITER_LIGNE (MAT:in MATRICE; LIGNE: in INDICE);
39|
40|   ERR_MAT:EXCEPTION;
41|
42|PRIVATE
43|   type cellule;
44|   type ptr is access cellule;
45|   type vec_ptr is array (INDICE range<>) of ptr;
46|
47|   type MATRICE (ORDRE: positive) is record
48|          L_tete,C_tete: vec_ptr (INDICE'first..ORDRE);
49|        end record;
50|
51|end MATRICES_CARREES;
52|
53|
54|--*************************************************
55|--*   IMPLEMENTATION  DES  MATRICES  CREUSES   *
56|--*************************************************
57|
58|with UNCHECKED_DEALLOCATION;
59|package body MATRICES_CARREES is
60|
61|   type CELLULE is record
62|         lig,col: INDICE;          -- coordonnees logiques.
63|         contenu: ELEMENT;
64|         lig_succ,col_succ:ptr;
65|        end record;
66|
67|   procedure DISPOSE is new unchecked_deallocation (cellule,ptr);
68|
69|
```

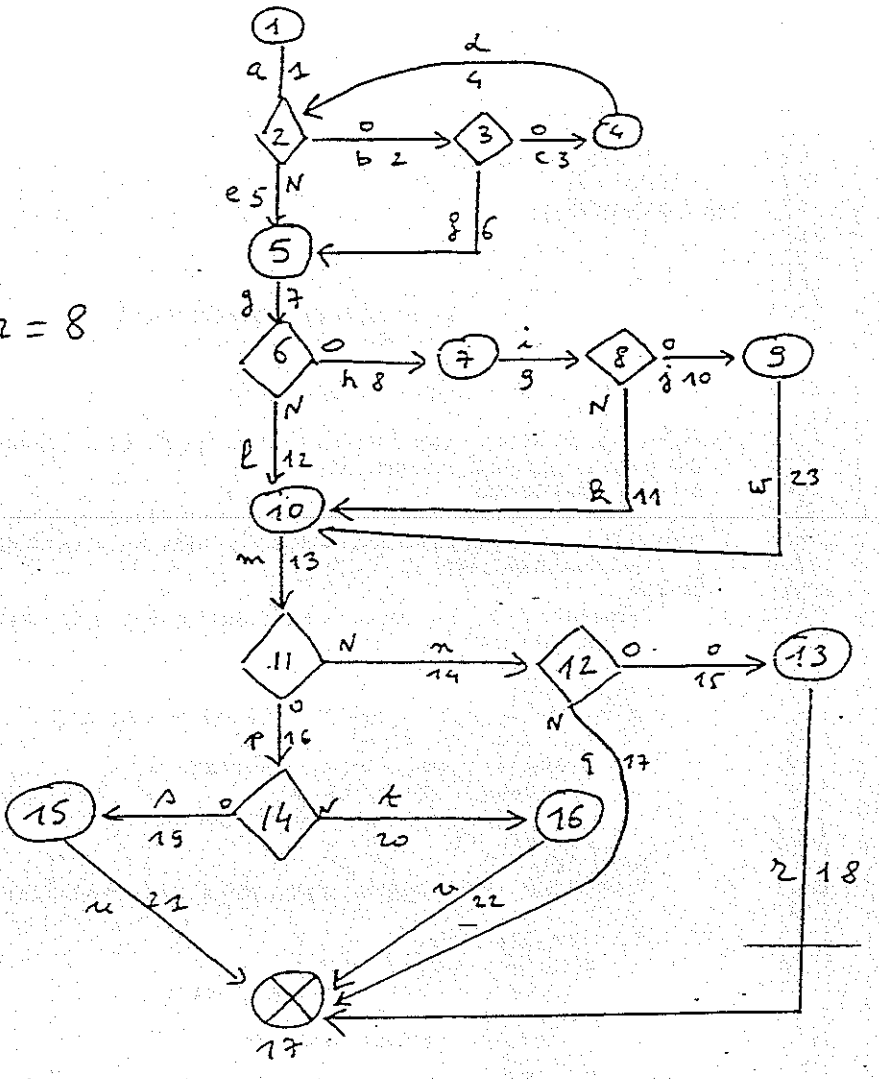


```
70|                -- SOUS-PROGRAMMES LOCAUX:
71|                -- ------------------------
72|
73|   procedure CTRL_BORNES (MAT: matrice; LIG,COL:indice) is
74|   begin
75|     if LIG > MAT.Ordre or COL > MAT.Ordre then
76|        raise ERR_MAT;
77|     end if;
78|   end;
79|
80|
81|   -- Transforme les coord. logiques (LIG,COLONNE) en coord.
82|   -- physiques dans la liste de cases correspondant a la COLONNE
83|   -- specifiee, designant la 1ere case dont le no. de ligne
84|   -- est >= LIG.
85|   -- Au retour, PLIG = null s'il n'existe pas de telle case.
86|
87|   procedure LIG_OE (MAT: matrice; lig,colonne: INDICE;
88|                     plig,prec: in out ptr) is
89|   begin
90|   ---
91|   end;
92|
93|
94|
95|   -- Implemente l'affectation  MAT(lig,col):=contenu;
96|   -- Preassertion: PC_SUCC.col >= col,
97|   --              ou PC_SUCC = null (MAT(lig,col)=0 lors de l'appel);
98|   --              et PC_PRED pointe sur la case precedente dans la
99|   --                        LIGne (si une telle case existe).
100|
101|  procedure AFFECTER (MAT: in out matrice; lig,col:INDICE;
102|                      contenu:ELEMENT; pc_pred,pc_succ:ptr) is
103|    pl_succ,pl_pred,p_l_c:ptr;
104|  begin
105|
106|    if pc_succ = null or else pc_succ.col > col then
107|
108|       -- Premier cas: MAT(lig,col)=Zero avant l'appel.
109|       -- On cree une nlle case si CONTENU <> Zero.
110|
111|       if contenu /= Zero then
112|          lig_OE (mat,lig,col,pl_succ,pl_pred);
113|          p_l_c:=new Cellule'(lig,col,contenu,pl_succ,pc_succ);
114|
115|          IF pc_pred /= null then          -- Insertion sur la LIGne.
116|             pc_pred.col_succ:=p_l_c;
117|          else
118|             Mat.L_tete(lig):=p_l_c;
119|          end if;
120|
121|          if pl_pred /= null then          -- Insertion sur la COLonne.
122|             pl_pred.lig_succ:=p_l_c;
123|          else
124|             Mat.C_tete(col):=p_l_c;
125|          end if;
126|       end if;
127|
128|    elsif contenu /= Zero then             -- 2eme cas: chgt de contenu.
129|       pc_succ.contenu:=contenu;
130|
131|    else                                   -- 3eme cas: RAZ de la case.
132|       lig_OE (mat,lig,col,p_l_c,pl_pred);
133|
134|       if pc_pred /= null then             -- Nouv. chainagee LIGne.
135|          pc_pred.col_succ:=p_l_c.col_succ;
136|       else
137|          Mat.L_tete(lig):=p_l_c.col_succ;
138|       end if;
```

```
139|
140|       if pl_pred /= null then          -- Nouv. chainages COLonne.
141|          pl_pred.lig_succ:=p_l_c.lig_succ;
142|       else
143|          Mat.C_tete(col):=p_l_c.lig_succ;
144|       end if;
145|       DISPOSE (p_l_c);
146|    end if;
147| end AFFECTER;
148|
149|
150|
151|                -- SOUS-PROGRAMMES EXPORTES:
152|                -- ***************************
153|
154|    -- Ajout d'une ligne a une autre:
155|
156| procedure AJOUT_LIGNE (MAT:in out MATRICE; LIGNE,PLUS: INDICE)is
157|   prec1,pcol1,pc1_succ:ptr;
158|   new_llc:BOOLEAN;
159|
160|   procedure ADD_CASE (col:INDICE; Mat_Lp_c:ELEMENT) is
161|       somme:ELEMENT:=Mat_Lp_c;
162|   begin
163|       while pcol1 /= null and then pcol1.col<col loop
164|         prec1:=pcol1;
165|         pcol1:=pcol1.col_succ;
166|       end loop;
167|       new_llc:=pcol1 = null or else pcol1.col > col;
168|       if not new_llc then
169|          somme:=somme + pcol1.contenu;
170|          if somme = Zero then          -- Mat[Ligne,col]:=Zero;
171|             pc1_succ:=pcol1.col_succ;
172|          end if;
173|       end if;
174|       AFFECTER (Mat,ligne,col,somme,prec1,pcol1);
175|
176|       if new_llc then
177|          if prec1 = null then
178|             prec1:= MAT.L_tete(ligne);
179|          else
180|             prec1:= prec1.col_succ;
181|          end if;
182|       elsif somme = Zero then
183|          pcol1:= pc1_succ;
184|       end if;
185|   end ADD_CASE;
186|
187|   procedure ADD_PLUS is new TRAITER_LIGNE (add_case);
188|
189| begin
190|   ctrl_bornes (Mat,ligne,INDICE'first);
191|   pcol1:=MAT.L_tete(ligne);
192|   ADD_PLUS (Mat,plus);
193| end AJOUT_LIGNE;
194|
195|
196|
197| -- PRIMITIVES GENERIQUES:
198|
199| procedure TRAITER_LIGNE (MAT:in MATRICE; LIGNE: in INDICE)is
200| begin
201| ------------------
202| end TRAITER_LIGNE;
203|
204|end MATRICES_CARREES;
```

Graphes de Commande :

ADD_CASE



N.C. = 7 + 1 = 8
= A - N + 2
= 23 - 17 + 2 = 8

AJOUT_LIGNE

NC = 0 + 1 = 1
= A - N + 2
= 1 - 2 + 2 = 1

N.C. global = NC(Add_case) + NC(Ajout_ligne) = 8 + 1 = 9

Procédure que l'on peut qualifier de simple (NC < 10)

CHEMINS	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
1) LIGNE vide a e g l m p s u	X				X		X					X	X			X			X		X		
2) 1ère colonne ≠ 0 de LIGNE > 1ère col. ≠ 0 de ligne PLUS a b f g l m p s u	X	X				X	X					X	X			X			X		X		
3) 1ère col. ≠ 0 de LIGNE < 1ère col. ≠ 0 de PLUS, (LIGNE a 1 col.) a b c d e g l m p t V	X	X	X	X	X		X					X	X			X				X		X	
4) 1ère col ≠ 0 de LIGNE = 1ère col. ≠ 0 de PLUS, Somme ≠ 0 a b f g h i k m n q	X	X				X	X	X	X		X		X	X			X						
5) idem, mais Somme = 0 a b f g h i j w m n o r	X	X				X	X	X	X	X			X	X	X			X						X

COUVERTURE 100 % des ARCS (OU BRANCHES) EN 5 TESTS (< N.C. = 8)

EXAMEN DE GÉNIE LOGICIEL
Cours B5

Deuxième Session 11 mars 1997
Locaux : ENSAM

La question de cours et l'exercice doivent être traités sur deux copies distinctes.

Prenez le temps de bien lire l'énoncé.

Tous documents et calculettes sont autorisés.

EXERCICE DE TESTS (8 points)

La procédure ADA ci-jointe (CHARGEMENT) incluse dans le module Graphe- Habille, charge en mémoire les données d'une structure de graphe valué a partir d'un fichier de données.

1) Etablir son graphe de commande.

N.B.: On ignorera les 2 petites procédures internes (lignes 83-93), ainsi que la gestion d'exception (lignes 130-131). On assimilera un RAISE (ligne 99, etc.) a un RETURN ordinaire.

2) Calculer son nombre cyclomatique. Comment peut-on qualifier la complexité de cette procédure?

3) En déduire une politique de tests permettant une couverture C2 (100 % des arcs ou branches). Préciser la sémantique de chaque jeu de données utilisé pour le test (par exemple: graphe vide, etc.).

Qu'en concluez-vous par rapport au nombre cyclomatique?

Comment procéderiez-vous si l'on vous demandait ensuite une couverture 100 %des instructions?

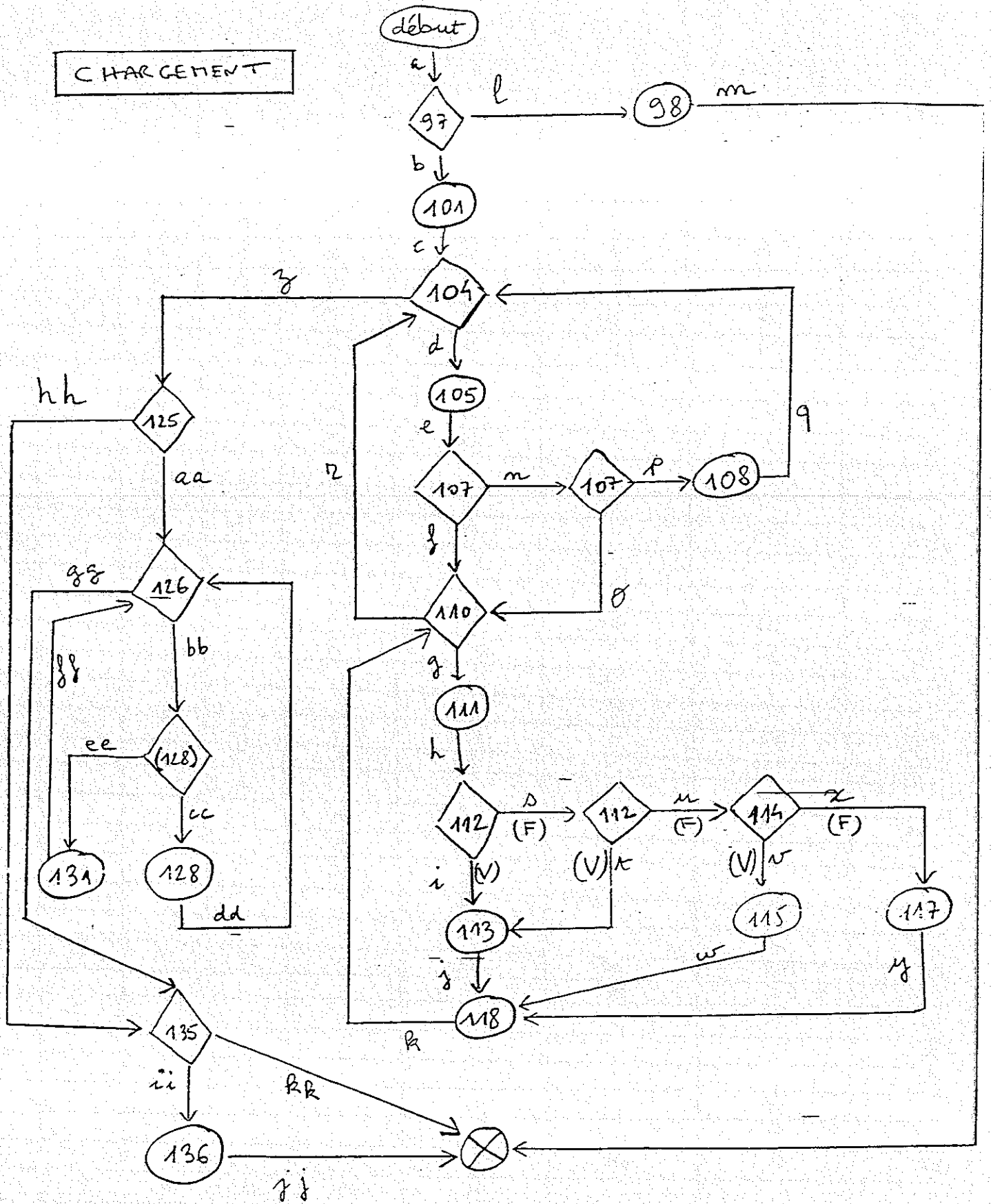
```
--=============================================
--=   Ajout de primitives d'E/S aux graphes.  =
--=============================================

WITH graphe_value,TEXT_IO;
PACKAGE graphe_habille IS

  USE graphe_value,TEXT_IO;

  alfalen :CONSTANT:=8;          -- long. des noms des sommets.
  titrelen:CONSTANT:=40;         -- longueur du titre.
  TYPE nom_des_sommets IS ARRAY(POSITIVE RANGE<>)
                          OF STRING (1..alfalen);

  TYPE setude (nb_sommets:POSITIVE) IS RECORD
              titre:STRING(1..titrelen);  -- titre de l'etude
              nomsom:nom_des_sommets(1..nb_sommets);
              g:graphe (nb_sommets);
        END RECORD;
  TYPE etude IS ACCESS setude;

  err_data:EXCEPTION;    -- suscitee par CHARGEMENT.


  -- LECTURE DU FICHIER INPUT ET CONSTITUTION DU GRAPHE EN MEMOIRE,
  -- AVEC CONTROLE DES DONNEES LUES.

  -- L'utilisateur fournit 1 fichier de donnees (Std_Input par defaut)
  -- contenant les donnees d'1 graphe sans boucle, et ayant la
  -- structure suivante:

  -- Ligne 1: Nb. de sommets (n)   Titre de l'etude(<=40 car.)

  -- Puis n groupes de lignes, contenant, pour le groupe d:

  --     Nom du sommet d (8 car.), cadre en tete de ligne;
  --     Nb. de successeurs de d (soit s ce nb.);
  --     puis s doublets: No du sommet successeur (soit j);
  --                      Valeur de l'arc d -> j (>0).

  -- Exemple:
  -----------

  -- 4     GRAPHE-TEST POUR FORD
  -- S1         3    2 3  4 6  3 8
  -- S2         1    4 2
  -- S3         0
  -- S4         1    3 2

  -- N.B.: Si FICDON n'est pas Std_Input, il doit avoir ete ouvert
  -- par le pgm. utilisateur avant l'appel de Chargement.

  -- CHARGEMENT controle la validite des donnees lues, et affiche
  -- 1 message d'erreur quand il y a lieu.
  --    La donnee RESEAU influe sur le niveau du controle.
  -- Reseau = True (valeur par defaut) indique que le graphe est
  -- 1 reseau de transport. Dans ce cas, il doit avoir 1 entree
  -- (le sommet 1) et 1 sortie (le sommet n) uniques.

  -- CHARGEMENT suscite l'anomalie ERR_DATA si les donnees
  -- contiennent 1 anomalie. Sinon, IMPGRAF est appele pour
  -- controle par l'utilisateur.

  PROCEDURE chargement(et:IN OUT etude; reseau:BOOLEAN:=TRUE;
                       ficdon:FILE_TYPE:=STANDARD_INPUT);
END graphe_habille;
```



```
WITH environnement,
PACKAGE BODY graphe_habille IS
  USE environnement, USE es_entier;
--*****************   C H A R G E M E N T   *****************
-- Lecture du fichier FICDON et constitution du graphe en memoire,
-- avec controle des donnees lues.

PROCEDURE chargement(et:IN OUT etude; reseau:BOOLEAN:=TRUE;
                     ficdon:FILE_TYPE:=STANDARD_INPUT) IS
    nbarc,nbsom,ima( ),capacite,len:INTEGER;
    erreur:BOOLEAN:=FALSE;
    normal:EXCEPTION;

  PROCEDURE imperr (numsom:INTEGER; msg:STRING) IS
  BEGIN
    PUT("*** Sommet no.:"); PUT(numsom,3); PUT(' ');
    PUT_LINE (msg);
    erreur:=TRUE;
  END;
  PROCEDURE bidon (&pred:POSITIVE; capacite:INTEGER) IS
  BEGIN
    RAISE normal;
  END;
  PROCEDURE chercher_pred IS NEW GAMMA_PRED (bidon);

BEGIN
    GET (ficdon,nbsom);
    IF nbsom <= 0 THEN
       imperr(0,"Nb. de sommets invalide.");
       RAISE err_data;
    END IF;
    et:=NEW setude(nbsom);
    GET_LINE (ficdon,et.Titre,len);
    et.Titre(len+1..titrelen):=(len+1..titrelen=>' ');
    FOR i in 1 .. nbsom LOOP
        SET_COL (ficdon,1);
        GET(ficdon,et.nomsom(i)); GET(ficdon,nbarc);
        IF reseau AND THEN ((nbarc<1)+(i<nbsom)) THEN
           imperr(i,"Erreur demi-degre exterieur.");
        ELSE
           FOR j in 1 .. nbarc LOOP
              GET(ficdon,image); GET(ficdon,capacite);
              IF (image NOT IN 1..nbsom)OR image=i THEN
                 imperr(i,"Extremite terminale incorrecte.");
              ELSIF capacite<=0 THEN
                 imperr(i,"Valeur d'arc <= 0");
              ELSE
                 SET (et.g,i,image, capacite);
              END IF;
           END LOOP;
        END IF;
    END LOOP;    -- fin lecture des donnees.

    --  VERIFICATION QUE LE SOMMET 1 EST BIEN SEUL PT. D'ENTREE.

    IF reseau THEN
       FOR I IN 2 .. NBSOM LOOP
           BEGIN
             chercher_pred (et.g,i);
             imperr(i,"est point d'entree interdit.");
           EXCEPTION
             WHEN normal => NULL;          -- I a >= 1 pred.
           END;
       END LOOP;
    END IF;    -- fin controle.
    IF erreur THEN RAISE err_data; END IF;
    impgraf (et);
END chargement;
END graphe_habille;
```

169

CHARGEMENT



$M = 12 + 1 = 13$

$= A - N + 2P = 37 - 26 + 2.1 = 13$

$10 < M < 40 \Rightarrow$ PGM RELATIVEMENT COMPLEXE

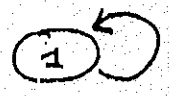
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	aa	bb	cc	dd	ee	ff	gg	hh
1) a r m GRAPHE VIDE	X											X	X																					
2) a b c d e f r z (hh) (ii) (jj)	X	X	X	X	X	X												X								X						X	X	X
3) a b c d e f g h i j k r z (hh) (kk)	X	X	X	X	X	X	X	X	X	X	X							X								X								X
4) a b c d e f g h s t j k r z (hh) (kk)	X	X	X	X	X	X	X	X		X	X							X	X	X						X								X
5) a b c d e f g h s u v w k r z d e f r z (hh) (kk)	X	X	X	X	X	X	X	X			X							X	X		X	X	X			X								X
6) a b c d e f g k s u x y k r z d e f r z (hh) (ii) (jj)	X	X	X	X	X	X	X	X			X							X	X		X			X	X	X						X	X	X
7) a b c d e n p q z (ac) (gg) (kk)	X	X	X	X	X								X		X	X										X	X								X
8) a b c d e n o g h s u x y k r z d e n o r z (aa) (bb) (ee) (ff) (gg) (ii) (jj)	X	X	X	X	X		X	X			X			X	X			X	X		X			X	X	X	X	X		X	X	X		X	X
9) a b c d e n o g h s u x y k r z d e n o r z	X	X	X	X	X		X	X			X			X	X			X	X		X			X	X	X	X	X	X	X		X			X

1) Graphe vide

2) Graphe Non Réseau à 1 Sommet, 0 arc :

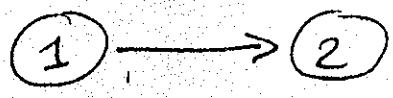
3) G.N.R. à 1 sommet, 1 arc invalide

4) G.N.R. à 1 sommet, 1 boucle :

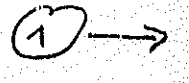


5) G.N.R. à 2 sommets, 1 arc de capacité = 0

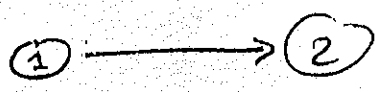
6) G.N.R. à 2 sommets, 1 arc ordinaire :



7) "Réseau" à 1 sommet (1 arc) :



8) Réseau correct à 2 sommets :



9) Réseau à 2 sommets Non Connexe :

EXAMEN DE GÉNIE LOGICIEL
Cours B5

1ère Session 28 février 1998
Locaux : maison des examens à Arcueil

La question de cours et l'exercice doivent être traités sur deux copies distinctes.

Barème de notation :
Question de cours 12 points
- Question 1 : 4
- Question 2 : 2
- Question 3 : 2
- Question 4 : 2
- Question 5 : 2
- Bonus pour les questions 6, 7, 8 : 1 point par question traitée.

Exercice noté sur 8 points

Prenez le temps de bien lire l'énoncé.

Tous documents et calculettes sont autorisés.

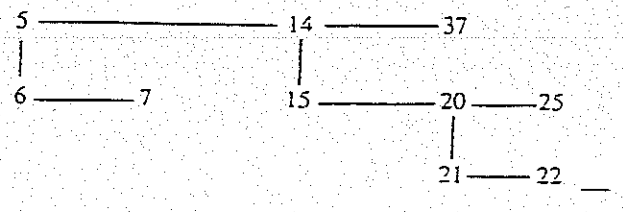
Une file d'attente à priorité est une file d'attente dont les clients sont servis dans l'ordre des priorités attachées à chacun d'eux : est pris en charge le client le plus prioritaire présent dans la file au moment où le « guichet » se libère.

L'une des organisations internes possible est celle dite de l'arbre P, dans laquelle :

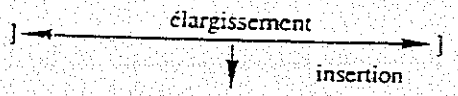
- la racine De l'arbre contient l'élément le *moins* prioritaire ;
- le sommet le plus à droite(en fait, une feuille) contient l'élément le plus prioritaire ;
- la branche droite de l'arbre est ordonnée selon les priorités croissantes ; d'éventuels éléments de même priorité sont rangés selon l'ordre croissant des durées de séjour en file (les plus anciens sont situés le plus bas) :
- le SAG (Sous-Arbre Gauche) de tout sommet s contient des éléments dont la priorité est supérieure (strictement) à celle de s, et inférieure ou égale à celle du fils droit de s : les éléments du SAG de priorité égale à celle de ce dernier sommet sont plus récents dans la file.

  Corollaire : comme le sommet le plus à droite contient le « maximum », il ne peut avoir de SAG : c'est donc bien une feuille.
- La structure se définit récursivement : tout SAG a les mêmes propriétés que l'arbre global.

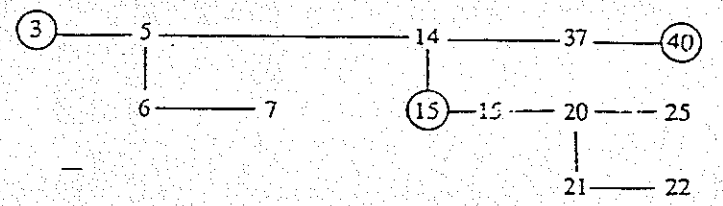
L'organisation d'un arbre-P se comprend assez bien si l'on fait subir à l'arbre une rotation de 45° vers la droite ; exemple :



L'axe horizontal est un axe d'*élargissement* de l'intervalle des priorités présentes dans le (sous-) arbre ; le demi-axe vertical, l'axe d'*insertion* à l'intérieur de cet intervalle.



Par exemple si la file précédente reçoit successivement des clients de priorité 40, 3, 15, elle devient :



La procédure AJOUTER ci-jointe, incluse dans le module PFILE-STABLE, range un nouveau client de priorité donnée dans une file d'attente (le paramètre F).

1)- Etablir son graphe de commande.
2)- Calculer son nombre cyclomatique. Comment peut-on qualifier la complexité de cette procédure ?
3)- En déduire une politique de test permettant une couverture à 100% des arcs (ou branches). Préciser la sémantique de chaque jeu de données utilisé pour le test (par exemple : file F vide ; etc...)

Comment procéderiez-vous si l'on vous demandait maintenant une couverture 100% des nœuds (ou sommets) ?

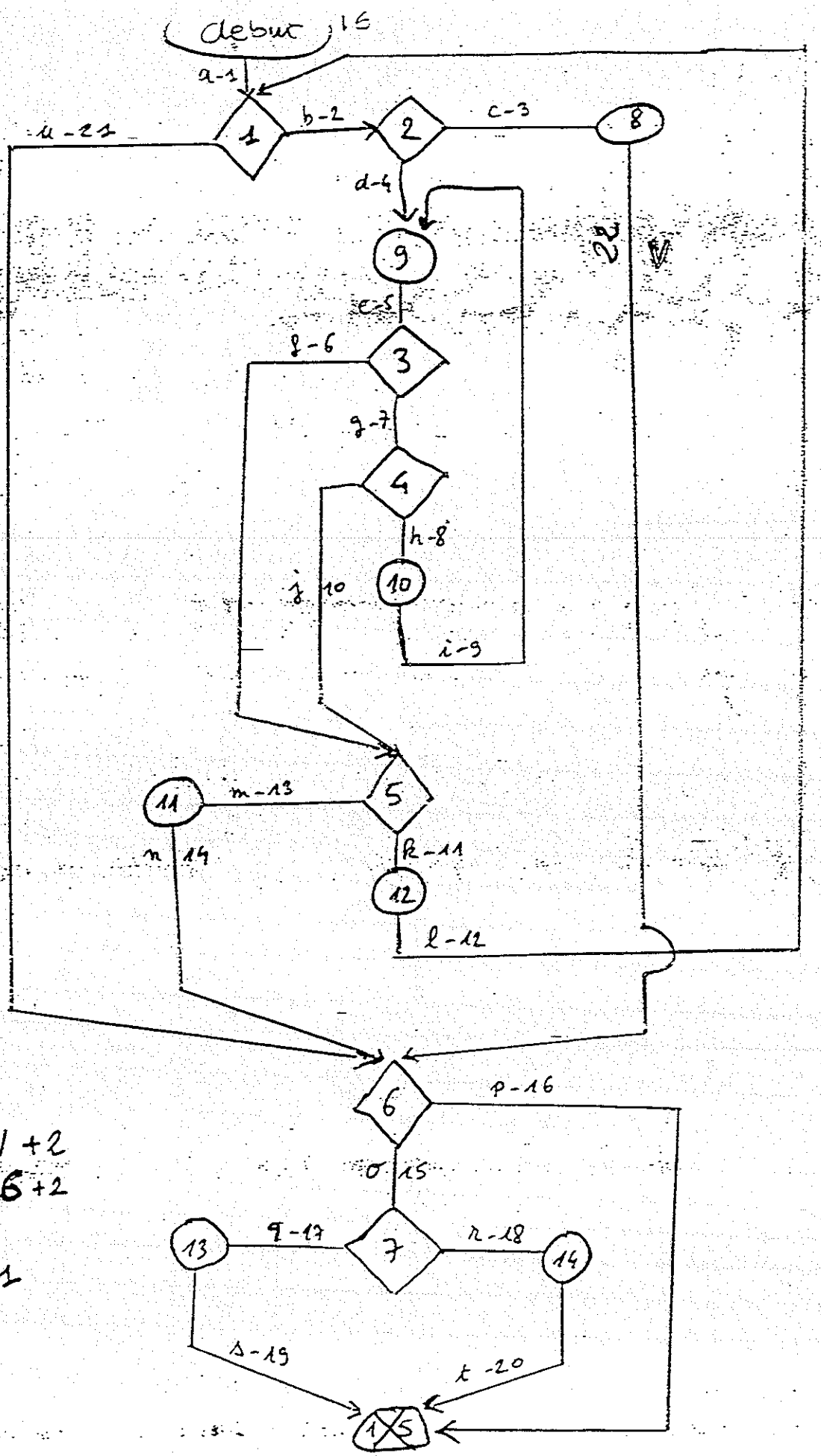
P.J : Le texte de la procédure Ada AJOUTER

```
--=====================================================
--    FILE D'ATTENTE A PRIORITE ILLIMITEE (STABLE)    --
--=====================================================

WITH arbin;                -- (reemploi des arbres binaires)

GENERIC
  TYPE element  IS PRIVATE;
  TYPE priorite IS PRIVATE;
  WITH FUNCTION "<" (p1,p2:priorite) RETURN BOOLEAN IS <>;

PACKAGE pfile_stable IS

  TYPE p_file IS LIMITED PRIVATE;


-- Test de l'etat de la p_file F:

  FUNCTION  vide (f:p_file) RETURN BOOLEAN;


-- Un nouvel element rejoint la p_file F:

  PROCEDURE ajouter (f:IN OUT p_file; e:element; p:priorite);

  -----------------------------------------------------------

PRIVATE
  TYPE client IS RECORD
                 elem:element;
                 prio:priorite;
        END RECORD;

  PACKAGE gest_mem IS NEW arbin (client);
  USE gest_mem;

  TYPE p_file IS RECORD          -- le type File d'Attente.
          racine:arbre:=NULL;
      END RECORD;

END pfile_stable;


-- Corps du module (implementation):

PACKAGE BODY pfile_stable IS               -- Corps du module:


  -- Mise en file d'attente d'1 nouvel arrivant:

  procedure AJOUTER (f:IN OUT p_file; e:element; p:priorite) IS
            -------

    nouv:arbre:=NEW((e,p));      -- adresse nouveau maillon cree.
    pf:arbre:=f.racine;
    q :arbre;
    pere_newrac:arbre:=NULL;
    newrac:BOOLEAN:=TRUE;

  BEGIN

    LOOP         -- boucle de descente dans l'arbre (a gauche).
       IF VIDE(pf) THEN          -- le S.-A. de racine PF est vide.
          EXIT;
       ELSIF NOT (pf.Cval.prio < p) THEN        --new client=new racine
          nouv.Dfils:=pf;
          EXIT;
       END IF;
```



```
       -- { Invariant: pf.prio < p }
       LOOP              -- boucle de parcours d'1 branche droite.
          q:=SADR(pf);
          EXIT WHEN VIDE(q) OR ELSE NOT (q.Cval.prio < p);
          -- { q.prio < p }
          pf:=q;
       END LOOP;

       IF VIDE (q) THEN          -- new Max du (Sous-)Arbre.
          pf.Dfils:=nouv;
          newrac:=FALSE;         -- elargissement a droite.
          EXIT;
       END IF;

       pere_newrac:=pf;          -- { pf.prio < p <= q.prio }
       pf:=pf.Gfils;
    END LOOP;

    IF newrac THEN     -- on a insere en tete d'1 (Sa-)Arbre.
      IF pere_newrac=NULL THEN          -- nlle racine globale.
         f.Racine:=nouv;
      ELSE
         pere_newrac.Gfils:=nouv;
      END IF;
    END IF;

  END ajouter;

-----------------------------------------------------------

END pfile_stable;
```

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
a m o q s (file vide)	X														X		X		X		X	
a b c v o q s client le – prioritaire	X	X	X												X		X		X			X
a b d e f m n p client + prioritaire ds. file à 1 élt	X	X		X	X	X							X	X		X						
a b d e g j k l u o r t (insertion dans une sous-file vide)	X	X		X	X		X			X	X	X			X			X		X	X	
a b d e g h i e f m n p client + prioritaire ds. file à 2 élts	X	X		X	X		X	X	X				X	X		X						



= A - N + 2
= 22 - 16 + 2
= 8
= 7 + 1