LES CAHIERS DE L'ANALYSE DES DONNÉES

J.-P. BENZÉCRI
F. BENZÉCRI

Sources de programmes d'analyse de données en langage PASCAL : (V) : élaboration et analyse de textes (V A) : liste de mots dans l'ordre du texte et liste alphabétique de mots

Les cahiers de l'analyse des données, tome 22, n° 4 (1997), p. 443-454

<http://www.numdam.org/item?id=CAD_1997__22_4_443_0>

Les Cahiers de l'Analyse des Données
Vol. XXII - 1997/8 - n°4 - pp.443-454

SOURCES DE PROGRAMMES D'ANALYSE DE DONNÉES EN LANGAGE PASCAL: (V) : ÉLABORATION ET ANALYSE DE TEXTES (V A) : LISTE DE MOTS DANS L'ORDRE DU TEXTE ET LISTE ALPHABÉTIDUE DE MOTS

[SOURCES PASCAL (V A)]

J.-P. & F. BENZÉCRI

0 Introduction : fonction du programme 'trigalat§'

Depuis 1990 (cf. [LING. TRI], in *CAD*, Vol.XV, n°1, pp.59-82; 1990) ont paru, dans *CAD*, de nombreuses analyses de textes en diverses langues: {Arabe, Espagnol, Français, Grec, Latin, Russe}, fondées sur le dénombrement automatique, au sein d'un corpus de textes, des formes d'un lexique, ou liste de mots, choisis suivant divers critères; au premier rang desquels est la fréqunece de leur emploi.

Il faut rappeler ici que le corpus est toujours considéré, initialement, comme un texte unique; divisé en chapitres; eux-mêmes subdivisés en versets; (le choix de ce terme, pour désigner des alinéas ou des phrases, s'expliquant parce que le premier texte traité a été celui de l'Évangile selon Saint JEAN). Le fichier de texte commence par une ligne de titre, non prise en compte dans la statistique; puis, après un 'aller à la ligne', le texte écrit sans autre contrainte que celle de ne comporter aucun chiffre, hormis ceux qui délimitent les chapitres et versets.

De façon précise, la présence du chiffre 1, signale le début d'un chapitre; les nombres: {2, 3, 4, 5,... ', 10, 11, ...} pouvant venir ensuite pour subdiviser le chapitre en versets. Dans la version des programmes considérée ici, il n'est pas prévu de signes, crochets ou parenthèses, pour inclure dans le texte des titres, sous-titres ou commentaires, non pris en compte dans le traitement statistique. Toutefois, pour une langue déterminée, seuls certains caractères sont considérés comme des lettres composant des mots; les autres caractères (non numériques) n'ont que valeur de séparateur, au même titre qu'u signe de ponctuation ou un espace blanc; et ils peuvent servir à libeller des sigles ou de brefs commentaires.

Les cahiers de l'analyse des données 0339-3097/97/04/

On vient de faire allusion à l'alphabet propre à chaque langue. Ce prblème est particulièrement complexe si la langue, comme l'Arabe ou le Russe, ne s'écrit pas, d'ordinaire, en caractètes latins; ou même comporte, comme le Grec, de nombreux signes diacritiques, essentiels à la distinction des formes de mots. Mais, même en Français, il est préférable de tenir compte des accents; et, en Espagnol, figurent dans l'ordre de l'alphabet, comme des caractères séparés, les couples 'ch', 'll', ou le 'ñ', lettre marquée d'un tilde.

Les articles rendant compte de l'analyse de textes en diverses langues, précisent d'abord les principes alphabétiques adoptés. Et, présentement, pour créer les listes de formes, nous utilisons un programme particulier propre à chaque langue. Pour l'Arabe, le Grec, le Russe, ce programme est strictement lié à une police de caractères que, mettant à profit la commodité du MacIntosh, nous avons créée pour la langue.

Mais en Latin (ainsi qu'en Anglais) il n'y a ni caractère particulier ni signe diacritique; et en Français, le rôle des accents peut être délimité suivant des règles accessibles à tous. C'est pourquoi, le programme 'trigalat§' expliqué ci-après, offre à l'utilisateur de ne considérer strictement qu'un alphabet de 26 lettres, comme en Latin; ou de ne retenir, des accents, que ce qui nous a paru suffire pour analyser des textes en Français. Les règles choisies apparaîtront dans le cours du programme.

Reste à dire la fonction de celui-ci. Ici encore, nous renvoyons au commentaire des sources. Il suffira de dire qu'outre un fichier numérique spécifiant le nombre des chapitres et versets, 'trigalat§' crée des listes d'occurrences, rangées soit dans l'ordre du texte traité, soit dans l'ordre alphabétique; et où chaque forme, écrite sur une ligne, est suivie, sur une autre ligne, de son adresse (numéros de chapitre et de verset).

1 Déclarations et structure générale du programme 'trigalat'

La valeur des types, variables et tableaux déclarés, apparaîtra dans le cours de l'algorithme.

Dans les déclarations, ne figure qu'une seule procédure: 'nuxer'; dont la fonction est de traduire un nombre entier: nn, en une suite de chiffres: nux; puis de concaténer cette suite, après un blanc, à une chaine: nox, existant lors de l'appel de 'nuxer'.

Le programme consiste une boucle:

```
while not(rpc='N') do begin er1:=0;rpc:='N';...
```

laquelle s'exécute autant de fois que le choix du texte à traiter se fait sans erreur; et que l'utilisateur désire poursuivre le traitement de nouveaux textes.

```
program trigalat;
uses memtypes,quickdraw,osintf,toolintf,sane,uver5;

const karp=4095;karb=32767;
type taw=array[0..karp]of ptr;ptaw=^taw;
       tbb=packed array[0..karb]of byte;ptbb=^tbb;
var teza,tezt:array[0..255]of ptaw;tbxx:array[0..255]of ptbb;
pont:ptr;pons:stringptr;
er1,lga,lgb,aaa,bbb,u,uu,w,asb,acb,
za,zz,zmx,wa,ww,wmx,tt,tmx:integer;
ma,mb,qa:string;
cr,ca,cb,ina,la,lb,n,na,nb,nc,x,bxx:longint;
nomf,nom,nox,nux:string;
oct:packed array[0..1]of byte;
cri,crm,crz,i,il,i2,j2,cap,ver,cp1,vr1,vrs,lox,mux,aig:integer;
rpc,ch1,rpa,rptx,rptr,rlng:char;
fxu:packed file of byte;nom2:string;fx:text;
nom1:string;fint:file of integer;nom3:string;
or2,dup,acc:array[0..255] of integer;

procedure nuxer(nn:integer);begin nux:='';mux:=10;uu:=nn;
while not(nn<mux) do mux:=10*mux;
while not(mux=1) do begin mux:=mux div 10;
  nux:=concat(nux,chr(48+(uu div mux)));uu:=uu mod mux end;
nox:=concat(nox,' ',nux);end;

begin benzecri;rpc:='O';
while not(rpc='N') do begin er1:=0;rpc:='N';
```

A l'intérieur de la boucle générale, sont comprises, après la boucle de désignation du texte à traiter:

while not((rpc='O')or(er1=6)) do begin...

deux boucles: if (rpc='O') then begin, dont la première offre à l'utilateur de spécifier les listes qu'il veut créer; et la seconde exécute les demandes ainsi formulées. Celle-ci s'achève en s'enquerrant du traitement éventuel d'un autre texte:

write('faut-il transformer un autre texte O ou N ');readln(rpc);end;
end;end.

Les trois: 'end', qui terminent le programme, marquent successivement, la fin de la procédure de création des listes; la fin de la boucle générale; et la fin du programme lui-même.

2 Demandes de l'utilisateur et spécifications de format

Avant la lecture du texte et la création des listes, trois blocs d'instructions fixent: le texte à traiter; les fichiers à créer; les conventions alphabétiques .

```
while not((rpc='O')or(er1=6)) do begin
 write('le nom du texte (français; ou latin etc.) à transformer est ');
 readln(nomf);
 rpc:=dialof(stringptr(@nomf));
 if (rpc='O') then begin
     write('ce nom est-il confirmé O ou N ');readln(rpc) end
   else er1:=er1+1 end;
```

2.1 Choix du texte à traiter

```
if (rpc='O') then begin rpc:='N';
 while not (rpc ='O') do begin
   writeln('NB on peut tenir compte des accents, comme pour un texte français (F)');
   writeln('ou éliminer tous les accents, comme pour un texte latin (L)');
   write('faut-il prendre en compte les accents(F) ou les éliminer(L) ');
   readln(rlng);
   writeln('ce programme peut créer une liste des mots dans l''ordre du texte');
   writeln('et/ou une liste triée dans l''ordre alphabétique');
   write('faut-il créer une liste dans l''ordre du texte O ou N ');
   readln(rptx);
   write('faut-il créer une liste triée alphabétique O ou N ');
   readln(rptr);
   if not(rptx='N') then rptx:='O';if not(rptr='N') then rptr:='O';
   if (rptx='N') and (rptr='N') then rptx:='O';
   if (rlng='F') then writeln('on prendra en compte des accents')
       else writeln('on éliminera tous les accents');
   if (rptx='O') then writeln('on créera une liste dans l''ordre du texte');
   if (rptr='O') then writeln('on créera une liste triée alphabétique');
   write('ces choix sont-ils confirmés O ou N ');readln(rpc);end;end;
```

2.2 Choix des listes demandées

Ainsi qu'on l'a annoncé, l'utilisateur peut demander, pour le texte choisi, une liste des formes dans l'ordre où elles se succèdent dans celui-ci; et une liste triée alphabétique. Toute réponse autre que:'N', est prise pour une demande effective; si rien n'est demandé, le programme crée une liste dans l'ordre du texte: rptx='O'.

Quant au format, celui du Français n'est pris que pour la réponse: rlng='F'; sinon, est pris, par défaut, le format sans accents du Latin.

2.3 Conventions alphabétiques

```
if (rpc='O') then begin
 for i:=0 to 255 do begin or2[i]:=0;dup[i]:=0;acc[i]:=0 end;
```

La valeur d'un caractère (de l'alphabet ASCII étendu) est spécifiée par trois tableaux: or2; dup, acc; qui sont initialement mis à zéro, puis remplis de certaines valeurs utiles.

Si: or2[i], reste fixé à zéro, l'octet 'i' est assimilé à un séparateur ou à un blanc; sinon, il est, dans la suite des formes, représenté par le caractère or2[i] (à moins que: or2[i] ne soit compris entre 48 et 57; auquel cas l'octet 'i' sert à délimiter chapitres et versets. Pour certains octets, qui dans l'ASCII ont valeur de lettre double, on a dup[i]≠0; et 'i' est lu comme: or2[i], suivi de dup[i].

```
  if (rlng='F') then begin
   acc[$E7]:=1;acc[$83]:=1;acc[$EA]:=1;acc[$EE]:=1;acc[$F2]:=1;
   acc[$87]:=1;acc[$8E]:=1;acc[$92]:=1;acc[$97]:=1;acc[$9C]:=1;
   acc[$CB]:=2;acc[$E9]:=2;acc[$ED]:=2;acc[$F1]:=2;acc[$F4]:=2;
   acc[$88]:=2;acc[$8F]:=2;acc[$93]:=2;acc[$98]:=2;acc[$9D]:=2;
   acc[$E5]:=3;acc[$E6]:=3;acc[$EB]:=3;acc[$EF]:=3;acc[$F3]:=3;
   acc[$89]:=3;acc[$90]:=3;acc[$94]:=3;acc[$99]:=3;acc[$9E]:=3;end;
```

Quant à l'accent; son absence est notée: acc[i]=0; et {1, 2, 3} valent, repectivement: aigu, grave, circonflexe. Si: rlng≠'F', il n'y a pas d'accent. On notera que selon les cas, les octets ont été notés en numération décimale (usuelle); ou hexadécimale; auquel cas, 'PASCAL' veut le caractère initial: '$'.

```
for i:=97 to 122 do begin or2[i]:=i;or2[i-32]:=i end;
```

Les 26 minuscules de l'alphabet sont prises telles quelles; les 26 capitales sont assimilées aux minuscules.

```
dup[174]:= 97;or2[174]:=101;dup[190]:= 97;or2[190]:=101;
dup[206]:=111;or2[206]:=101;dup[207]:=111;or2[207]:=101;
dup[222]:=102;or2[222]:=105;dup[223]:=102;or2[223]:=108;
dup[38] :=101;or2[38] :=116;
```

Sont prises comme doubles lettres et traduites en minuscules: {Æ, æ, Œ, œ, fi, fl}; le symbole '&' est lu comme 'et'.

NB: dup, donne le numéro de la *première* des deux lettres: e.g: 'f' pour 'fl'.

```
or2[$80]:=$61;or2[$81]:=$61;
for i:=$87 to $8C do or2[i]:=$61;
or2[$CB]:=$61;or2[$CC]:=$61;or2[$E5]:=$61;or2[$E7]:=$61;
or2[$82]:=$63;or2[$8D]:=$63;
or2[$83]:=$65;for i:=$8E to $91 do or2[i]:=$65;
or2[$E6]:=$65;or2[$E8]:=$65;or2[$E9]:=$65;
for i:=$92 to $95 do or2[i]:=$69;
for i:=$EA to $ED do or2[i]:=$69;
or2[$84]:=$6E;or2[$96]:=$6E;
or2[$85]:=$6F;for i:=$97 to $9B do or2[i]:=$6F;
or2[$BF]:=$6F;or2[$CD]:=$6F;or2[$EE]:=$6F;or2[$EF]:=$6F;or2[$F1]:=$6F;
or2[$86]:=$75;
for i:=$9C to $9F do or2[i]:=$75;for i:=$F2 to $F4 do or2[i]:=$75;
or2[$D8]:=$79;or2[$D9]:=$79;
```

On élimine tous les signes diacritiques autres que les accents; même la cédille du 'ç'. Cette règle convient pour le Français, le Latin, l'Anglais. On a rappelé qu'en Espagnol le '~' a une valeur essentielle; on sait qu'en Allemand le tréma (signe de l'umlaut) définit plutôt une double lettre: 'ä'≈'ae'.

3 Lecture et mise en mémoire du texte à traiter

```
cr:=0;za:=0;wa:=0;nox:='';tt:=0;bxx:=0;
new(teza[0]);new(tezt[0]);new(tbxx[0]);
```

En tête du programme sont déclarés deux types: taw, et: tbb, destinés, respectivement, à contenir des pointeurs et du texte:

```
const karp=4095;karb=32767;
type taw=array[0..karp]of ptr;ptaw=^taw;
     tbb=packed array[0..karb]of byte;ptbb=^tbb;
```

l'un et l'autre type occupe une zone 32k de mémoire. Les zones elle-mêmes étant affectées par des pointeurs, sont définis les types: ptaw, et: ptbb. Et, en tête du tableau des variables, on a les déclarations:

```
var teza,tezt:array[0..255]of ptaw;tbxx:array[0..255]of ptbb;
```

par les 256 pointeurs: tbxx[tt], on peut accéder à un texte fragmenté dont la longueur totale est: 256×32k = 8 megaoctets. Plus précisément, quand le texte est lu en mémoire centrale, est affectée, à chacune de ses formes (i.e.: mots), une chaîne: 'nox', (codée en utilisant, comme on l'explique ci-après, la convention alphabétique définie par les tableaux: or2, dup, acc, du §2.3); à laquelle sont ajoutés, par la procédure 'nuxer' (cf. *supra*, §1), le numéro: cp1, du chapitre; puis celui: vr1, du verset.

Une fois ainsi enregistrés, les mots ne sont plus modifiés ni déplacés, même pour le tri alphabétique; mais le programme y accède par des pointeurs, contenus dans les tableaux: teza[za], tezt[za]; où, d'une part, l'entier: za (ou: zz...) prend une valeur comprise entre: 0, et un maximum: zmx, (≤255) qui s'accroît de 1, chaque fois qu'un nouveau pointeur est initialisé; et, d'autre part, l'accès, au texte entier, par les deux voies: teza, et: tezt, permet de simuler le tri par fusion de deux copies du texte en comparant les enregistrements vers lesquels pointent deux pointeurs mobiles (i.e.:variables, cf. *infra*, §4.2).

Avant d'entreprendre de lire le texte, on crée, en mémoire centrale, deux tableaux de pointeurs: teza[0], tezt[0]; et un tableau de texte: tbbx[0].

```
reset(fxu,nomf);nom:='';i1:=32;
while not (eof(fxu) or (i1=13)) do begin nom:=concat(nom,chr(i1));
  read(fxu,oct[1]);i1:=oct[1];i1:=i1 mod 256;if (i1<0) then i1:=i1+256;end;
write(nom);readln(rpc);
```

Ainsi qu'on l'a dit au §0, le texte à traiter commence par une ligne de titre, destinée seulement à l'utilisateur; ligne dont ne gardent pas trace les fichiers créés par 'trigalat'; ni par les autres programmes, après celui-ci. Le titre est simplement lu, et affiché à l'écran (après un blanc). Ainsi qu'on le voit au §1, le fichier du texte à traiter a été déclaré: fxu:packed file of byte; on a pris ce type, de préférence à: 'text', parce que, pour: 'text', les procédures usuelles de lecture n'offrent pas un accès sûr aux octets individuels. On supposera désormais que le nom: nomf, du fichier à traiter est: Disq:Texte.

```
rewrite(fint,concat(nomf,'§num'));
cri:=0;cap:=0;ver:=0;vrs:=0;crz:=0;crm:=0;aig:=0;
```

On crée un fichier: 'Disq:Texte§num', destiné à contenir une suite d'entiers, donnant, dans l'ordre des chapitres, le nombre de versets que comprend chacun de ceux-ci. Reste à expliquer les entiers mis à zéro...

cri : vaut 1, si un mot est en cours de lecture; et: zéro, sinon;
cap: numéro du chapitre en cours d'écriture;
ver: numéro du verset en cours de lecture;
vrs: sert à loger un numéro de verset qu'on lit dans: 'Disq:Texte';
crz: vaut 1, si un numéro de verset est en cours de lecture; et: zéro, sinon.
crm: initialement nul, prend et garde la valeur 1, dès qu'un mot a été effectivement lu;
aig : prend une valeur différente de zéro quand est lue une lettre qui, selon le tableau: acc, cf. §2.3, porte l'un des accents numérotés: {1, 2, 3}; est mis à zéro après lecture d'une lettre non accentuée; sauf si (cf. *infra*) celle-ci est un 's' (codé: 113=$73, en ASCII), susceptible de marquer le pluriel d'un mot accentué; ou d'être la dernière lettre de: 'dès', à distinguer de: 'des'...

C ommence alors la boucle de lecture du contenu de: 'Disq:Texte', (après le titre), octet par octet; lecture faite dans le premier octet du tableau: oct, afin d'éviter toute confusion avec le format de lecture des entiers.

```
while not eof(fxu) do begin
  read(fxu,oct[1]);i1:=oct[1];i1:=i1 mod 256;if (i1<0) then i1:=i1+256;
  i2:=or2[i1];j2:=dup[i1];ch1:=chr(i1);
  if not((i2=$73) or (i2=0)) then aig:=0;
  if not(acc[i1]=0) then aig:=acc[i1];
  if (i1 in [48..57]) then begin
     if (crz=0) then vrs:=0;
      crz:=1;vrs:=(10*vrs)+i1-48 end;
  if (crz=1) and not(i1 in [48..57]) then begin
      if (vrs=1) then begin cap:=cap+1;
          if (crm=1) and (1<cap) and (0<ver) then write(fint,ver) end;
      if not((vrs=ver+1) or (vrs=1)) then begin writeln;
           write('ZZZ: cap =',cap:4,' ; ver =',ver:4,' ; vrs =',vrs:4);
           readln(rpc) end;
      ver:=vrs;crz:=0 end;
```

Si le caractère lu est un chiffre, celui-ci sert soit à créer un numéro de verset (cas: crz=0); soit à modifier le numéro déjà en lecture (cas: crz=1). Si un numéro est en cours de lecture (crz=1) et que le caractère lu n'est pas un chiffre, c'est que la lecture du numéro: vrs, est achevée. Si: vrs=1, c'est que commence un chapitre; le nombre: ver, des versets du chapitre précédent (si toutefois il y en a un) est à écrire dans: 'Disq:Texte§num'. Si est enfreint l'ordre régulier de numérotage des versets, ceci est signalé à l'utulisateur; qui doit entrer un caractère: rpc, mais la lecture, de: Disq:Texte, se poursuit.

```
  if not(j2=0) then begin
        nox:=concat(nox,chr(j2));end;
  if not(i2=0) then begin
     nox:=concat(nox,chr(i2));cri:=1 end
    else if (cri=1) then begin
```

Si le caractère lu est double, la 1-ère lettre du doublet: chr(j2), est ajoutée au mot: 'nox', qui est en cours de lecture. Si le caractère qu'on peut appeler principal: chr(i2), est une véritable lettre, celle-ci est ajoutée à: 'nox'; et on enregistre (ou confirme) qu'un mot est en cours de lecture. Mais si: (i2=0), et qu'un mot est en cours de lecture: (cri=1), il faut, par une suite d'opératons, qu'introduit (then begin...), prendre acte de ce que cette lecture est achevée.

```
      if (aig=1) then nox:=concat(nox,'/');
      if (aig=2) then nox:=concat(nox,'"');
      if (aig=3) then nox:=concat(nox,'>');
      lox:=length(nox);
       cri:=0;crm:=1;aig:=0;
       cpl:=cap;if (cpl=0) then cpl:=1;
       vrl:=ver;if (vrl=0) then vrl:=1;
       nuxer(cpl);nuxer(vrl);
```

Après adjonction éventuelle d'un accent, la composition du mot est arrêtée, sa longueur: lox, est connue. Les paramètres: {cri, aig} sont remis à zéro; crm, est mis, ou confirmé, à 1. Au mot proprement dit, on ajoute les numéros {cpl, vrl} de chapitre et verset. Mais il faut prendre garde que le texte en cours de lecture peut être dépourvu de numérotage; en ce cas, cap et ver sont restés à zéro; et {cpl, vrl} sont pris, par défaut, égaux à 1.

Reste maintenant à ranger la chaîne: nox, dans une zone de mémoire désignée par un pointeur: tbxx[tt]; en une position qui elle-même sera gardée

```
if (wa=4096) then begin
        wa:=0;za:=za+1;new(teza[za]);new(tezt[za]);end;
if (bxx+length(nox)>32767) then begin
     tt:=tt+1;bxx:=0;new(tbxx[tt]);end;
pont:=@tbxx[tt]^[bxx];bxx:=bxx+length(nox)+1;
    teza[za]^[wa]:=pont;tezt[za]^[wa]:=pont;
    pons:=stringptr(pont);pons^:=nox;
pons^[0]:=chr(lox);pons^[lox+1]:=chr(length(nox)-(lox+1));
cr:=cr+1;wa:=wa+1;nox:=''; end end;
```

dans les tableaux de pointeurs: teza[za], tezt[za]. Encore faut-il que les zones de mémoire déjà créées suffisent aux informations qu'on doit y ranger: si tel n'est pas le cas, l'espace nécessaire est affecté par l'opération: 'new'. Il faut encore remarquer que la chaine: nox, n'est pas gardée exactement telle quelle; mais, comme deux chaînes succesives; dant la 1-ère esy le mot proprement dit; et la 2-nde est formée des deux numéros, {cp1, vr1}, séparés par un blanc; la longueur de chacune des chaîne étant logée suivant la norme de l'ALGOL.

Deux symboles: end, marquent la fin de la boucle:

else if (cri=1) then begin...

d'enregistrement d'un mot; et de la boucle générale:

while not eof(fxu) do begin...

de lecture du fichier: Disq:Texte.

```
if ((crm=1) and (0<ver)) then write(fint,ver);
if (cap=0) then write(fint,1);
close(fxu);close(fint);
zmx:=za;wmx:=wa-1;tmx:=tt;
writeln('cr =',cr:10,'  ;  zmx=',za:6,'  ;  wmx=',wmx:10,'  ;  tmx=',tmx:6);
```

On notera que l'enregistrement du dernier mot ne se fait que si le dernier caractère n'est pas accepté comme une lettre (i.e. si: i2=0). Si du texte a effectivement été lu, le nombre de versets du chapitre en cours est mis dans: 'Disq:Texte§nuù'. On ferme les fichiers: fxu, fint (puverts, respectivement en lecture et écriture); l'utilisateur est averti du nombre, cr, des mots lus; et de l'étendue de mémoire qu'occupent les zones initialisées par: new.

4 Tri alphabétique et création des listes de formes

On considérera successivement: la création d'une liste dans l'ordre du texte (§4.1); le tri alphabétique des formes (§4.2); la création d'une liste alphabétique (§4.3). Seul l'algorithme de tri offre matière à un commentaire.

4.1 Création d'une liste dans l'ordre du texte

```
if (rptx='O') then begin
 writeln('On crée la liste dans l''ordre du texte');
 rewrite(fx,concat(nomf,'$'));
 for zz:=0 to zmx do begin
  wa:=karp;if (zz=zmx) then wa:=wmx;
  for ww:=0 to wa do begin
    pont:=tezt[zz]^[ww];pons:=stringptr(pont);
    ma:=pons^;writeln(fx,ma);
    pont:=@pons^[length(ma)+1];pons:=stringptr(pont);
    qa:=pons^;writeln(fx,qa);end;end;
 close(fx);end;
```

Les chaînes à écrire sur le fichier: Disq:Texte§, sont lues en mémoire centrale au moyen du pointeur: pont:=tezt[zz]^[ww]; la variable: zz, qui désigne le bloc de pointeurs, varie de 0 à zmx; au sein de chaque bloc, l'indice: ww, varie de 0 à un maximum qui est: karp; sauf pour le dernier bloc, qui peut être incomplétement utilisé; auquel cas, le maximum est wmx<karp.

4.2 Tri alphabétique des formes

L'algorithme consiste en une imbrication de boucles dont nous expliquons la stucture hiérarchique. Le paramètre supérieur est: la, longueur des blocs de formes déjà bien rangés et que l'on interclasse: au départ: la=1. On interclasse le 1-er bloc avec le 2-nd;... le (2.p-1)-ème avec le 2.p-ème. Quand ces couples de blocs ont été bien classés, on a des blocs bien classés de longueur: 2*la. Donc, on double: la; et ainsi de suite jusqu'à ce que: la, soit ≥ au nombre total: cr, des formes à classer.

```
if (rptr='0') then begin
 writeln('On trie, puis on crée la liste alphabétique');
 la:=1;ina:=0;
 while (ina+la<cr) do begin
  x:=cr-(ina+la);if(x<la)then lb:=x else lb:=la;
  na:=ina;zz:=na div (karp+1);ww:=na mod (karp+1);
  pons:=stringptr(teza[zz]^[ww]);
  ma:=pons^;
  nb:=ina+la;zz:=nb div (karp+1);ww:=nb mod (karp+1);
  pons:=stringptr(teza[zz]^[ww]);
  mb:=pons^;
```

Au sein de la boucle de fusion:while (ina+la<cr) do begin..., on interclasse deux blocs: (a) = [ina... ina+la-1], et: (b) = [ina+la... ina+la+lb-1]. Quand ce classement est fait, on remet à jour: {ina, la, lb}. Les formes à comparer sont lues par des pointeur de la forme: teza[zz]^[ww] ; et, après comparaison, les formes bien classées sont désignées des pointeurs de la forme : tezt[zz]^[ww], à prendre consécutivement.

Dans la fusion de deux blocs (a) et (b), les formes considérées sont celles que désignent ceux des cr pointeurs: teza[zz]^[ww], dont le rang global: n , varie de: n=ina, à: n=(ina+la+lb)-1; il faut que ces pointeurs soient recopiés dans le bloc, de même amplitude, des pointeurs tezt; c'est pourquoi l'essentiel de la fusion consiste en une boucle: for n:= ina to...

On doit prendre garde que cet indice: n, est celui du pointeur: tezt, qui reçoit une nouvelle valeur; laquelle ne peut être que le 1-er pointeur: πa, non déjà recopié du bloc (a); ou le 1-er pointeur: πb, non recopié de (b). Les mots devant être rangés dans l'ordre alphabétique croissant, on doit comparer les deux mots: {ma, mb}, vers lesquels pointent: {πa, πb}.

```
  for n:=ina to (ina+la+lb)-1 do begin
   if(na<ina+la) and (nb<ina+la+lb)
     then begin
```

Encore faut-il, toutefois que ni l'un ni l'autre des deux blocs (a) et (b) ne soit déjà entièrement copié; d'où la condition: if(na<..., par laquelle débute le bloc des instructions de comparaison, en tête de la boucle: for n:=...

```
if (ma=mb) then acb:=1 else acb:=0;
lga:=length(ma);lgb:=length(mb);u:=0;w:=1;asb:=0;
if (acb=0) then while not(w=0) do begin w:=0;
   if (lga=u) then aaa:=0 else aaa:= ord(ma[u+1]);
   if (lgb=u) then bbb:=0 else bbb:= ord(mb[u+1]);
   if (aaa>bbb) then asb:=1;
   if (aaa=bbb) and not(aaa=0) then begin w:=1;u:=u+1 end end;
```

Plus précisément, la comparaison est toute faite si: ma=mb, ce qu'on code: acb=1, pour dire: "a comme b". Sinon, on prend la suite des lettres, le rang: u, croissant jusqu'à ce que soit trouvée une différence entre: ma, et: mb.

```
if(asb=0)
  then begin
    nc:=na;na:=na+1;
    if not(na=ina+la) then begin
      zz:=na div (karp+1);ww:=na mod (karp+1);
      pons:=stringptr(teza[zz]^[ww]);
      ma:=pons^ end end
  else begin
    nc:=nb;nb:=nb+1;
    if not(nb=ina+la+lb) then begin
      zz:=nb div (karp+1);ww:=nb mod (karp+1);
      pons:=stringptr(teza[zz]^[ww]);
      mb:=pons^ end end end
```

Le résultat est: asb=1, si: "a est supérieur à b"; sinon: asb=0, et le pointeur: πa est recopié, comme c'est aussi le cas quand: ma=mb.

```
else if(ina+la=na)
  then begin
    nc:=nb;nb:=nb+1;
    if not(nb=ina+la+lb) then begin
      zz:=nb div (karp+1);ww:=nb mod (karp+1);
      pons:=stringptr(teza[zz]^[ww]);
      mb:=pons^ end end
  else begin
    nc:=na;na:=na+1;
    if not(na=ina+la) then begin
       zz:=na div (karp+1);ww:=na mod (karp+1);
      pons:=stringptr(teza[zz]^[ww]);ma:=pons^  end end;
```

Reste à considérer le cas où l'un des blocs: (a), ou: (b); la 1-ère de ces deux éventualités étant reconnue par la condition: (ina+la=na).

```
zz:=nc div (karp+1);ww:=nc mod (karp+1);pont:=teza[zz]^[ww];
zz:=n div (karp+1);ww:=n mod (karp+1);tezt[zz]^[ww]:=pont;end;
```

Le rang: nc, du pointeur: teza, à copier, étant déterminé, d'une manière ou d'une autre, on calcule son adresse: {zz,ww} pour le mettre dans tezt.

```
ina:=ina+la+lb;
if (cr-1<ina+la) then begin
 la:=2*la;ina:=0;writeln('la =',la:8);
 if (cr>la) then
    for zz:=0 to zmx do for ww:=0 to karp do
      teza[zz]^[ww]:=tezt[zz]^[ww];
end;end;
```

La boucle de fusion de blocs: while (ina+la<cr) do begin..., se termine en fixant les paramètres: {ina, la} pour rentrer dans la boucle; si, après un

doublement de: 'la', le tri doit effectivement se pousrsuivre, l'ordre auquel on s'est arrêté dans: tezt, est recopié dans: teza; si le tri est achevé, il est inutile de mettre à jour: teza; car seul: tezt, sert pour créer la liste alphabétique.

4.3 Création d'une liste alphabétique

```
 writeln('cr =',cr:10,'  ;  zmx=',za:6,'  ;  wmx=',wmx:10,'  ;  tmx=',tmx:6);
 rewrite(fx,concat(nomf,'$t'));
 for zz:=0 to zmx do begin
  wa:=karp;if (zz=zmx) then wa:=wmx;
  for ww:=0 to wa do begin
    pont:=tezt[zz]^[ww];pons:=stringptr(pont);
    ma:=pons^;writeln(fx,ma);
    pont:=@pons^[length(ma)+1];pons:=stringptr(pont);
    qa:=pons^;writeln(fx,qa);end;end;
 close(fx);end;
```

La création de la liste alphabétique: Disq:Texte§t, ne diffère en rien de celle de la liste dans l'ordre du texte (cf. §4.1).

```
 for zz:=0 to zmx do begin dispose(tezt[zz]);dispose(teza[zz]);end;
 for tt:=0 to tmx do dispose(tbxx[tt]);
 write('faut-il transformer un autre texte O ou N ');readln(rpc);end;
end;end.
```

Le traitement d'un texte s'achève en libérant les blocs de mémoire; et demandant à l'utilisateur s'il veut poursuivre. Les trois: end, qui terminent le programme sont, respectivement: pour le traitement d'un texte, commandé par l'instruction conditionnelle: if (rpc='O') then begin rpc:='N';...; pour la boucle: while not(rpc='N') do begin er1:=0;rpc:='N';... qui constitue l'essentiel du programme; et pour le programme lui-même.

5 Enchaînement des programmes de traitement et analyse des textes

Le présent cahier ne donne la source que du programme 'trigalat', qui réalise la première étape de l'analyse d'un texte: la création de la liste des formes, étiquetées par chapitre et verset, et rangées dans l'ordre du texte: Disq:Texte§; et de la liste alphabétique: Disq:Texte§t. De nombreux articles parus dans *CAD*, et dont le dernier est: [ASSOC. VILLE NOUV.], offrent des exemples des traitements ultérieurs, dont il reste à publier les programmes.

Le programme 'tridic' construit un tableau de correspondance, entre, d'une part, fragments du texte (chapitres, versets ou autres); et, d'autre part, une suite de formes, communément appelée: 'lexique'; laquelle doit être présentée, dans l'ordre alphabétique, avec un mot par ligne. Le tableau de correspondance étant construit, on rentre dans les programmes usuels d'analyse factorielle et de classification automatique (cf. [SOURCES PASCAL, I, II])

Même si, en toute rigueur, le choix du lexique peut se faire au gré du linguiste, il est conforme aux principes de l'analyse distributionnelle, qu'il soit fondé sur un inventaire de fréquences, fait sur le texte même qu'on analyse; ou sur un corpus plus étendu. À cette fin, le programme: 'qamus' (dont le nom signifie en arabe: dictionnaire), produit, à partir de: Disq:Texte§t, une liste:

Disque:Texte$§t, où chaque forme attestée figure une seule fois, quel qu'en soit le nombre d'occurrences; mais précédée de ce nombre, écrit sur une ligne distincte.

Telle quelle, la liste: Disque:Texte$§, ne suffit pas à offrir une vue globale de l'utilisation des formes de la langue dans le corpus. Par le programme: 'trimu§', cette liste est triée de telle sorte que les mots y soient rangés dans l'ordre croissant du nombre d'occurrences: c'est la liste: Disq:Texte$§t. En fait, le programme: 'trimu§' a été conçu, initialement, par passer de: Disq:Texte§, à: Disq:Texte§t; mais compte tenu de la présentation de la liste: Texte$§, avec les mots précédés de leur fréquence, elle a bien ici l'effe désiré.

D'après: Disq:Texte$§t, l'utilisateur peut, à loisir choisir un lexique: . Toutefois, celui-ci, soit: Disq:Dic, n'est pas présenté dans l'ordre alphabétique que requiert: 'triduc'. Le rangement peut être fait par un programme: 'trimu', fondé, comme 'trimu§', sur l'algorigthme expliqué au §4.2.

Reste qu'une même forme de mot, peut prendre, selon le contexte, des sens différents; ce qui compromet la clarté de l'analyse. En lisant le fichier: Disq:Texte§, le programme: 'prCcrd' crée une liste: Disq:Text(x§, qui ne diffère de: Texte§, qu'en ce qu'après l'adresse de chaque occurrence, celle-ci est répétée, sur la même ligne, avec un contexte, comprenant, avant et après, un même nombre: x, de formes; nombre qu'on peut choisir de 2 à 5.

En soumettant: Disq:Texte(x§, à: 'trimu§', on obtient une liste: Disq:texte(x§t, dans laquelle, on a, pour les formes rangées dans l'ordre alphabétique, le bloc des occurrences de chacune, avec leur contexte. Une telle *concordance* permet d'apprécier la diversité des emplois d'une forme; qui peut non seulement dériver quant au sens, mais relever de plusieurs mots différents: e.g.: 'porte', nom; ou diverses places dans la conjugaison du verbe: 'porter' (je porte, il porte, qu'il porte...).

Restent donc, au moins, à publier, dans [SOURCES PASCAL V], les programmes: 'trimu§, 'trimu', 'tridic', 'qamus', 'prCcrd'...

Références bibliographiques

On trouve dans [LING. TRI 2], in *CAD*, Vol. XVI, n°2, pp. 133-160, (1991), des références quant à la conception et à l'enchaînement du système de programmes dont 'trigalat' fait partie.