X2010 – PSI Épreuve d'info, corrigé par Laure Gonnord 2012

Maple V9

Partie 0 – Préparation

```
> restart;
```

```
> interface(version);
```

Maple Worksheet Interface, Maple 9.01, Linux, July 10 2003, 137236

```
> texte :="quelbonbonbon";
```

$$texte := \text{"quelbonbonbon"}$$

```
> mot1:= "bon";
```

$$mot1 := \text{"bon"}$$

```
> mot2:="bonbon";
```

$$mot2 := \text{"bonbon"}$$

```
> mot3:="bonnet";
```

$$mot3 := \text{"bonnet"}$$

```
> allouer:=proc(n) ## au secours comme c'est moche !
  [0$n];
  end;
```

$$allouer := \mathbf{proc}(n)\ [\text{`\$`}(0, n)]\ \mathbf{end\ proc}$$

```
> allouer(6);
```

$$[0, 0, 0, 0, 0, 0]$$

```
> chaine2intlist :=proc (mot)
  local l,resu;
  l:=length(mot);
  resu:=convert( mot, 'bytes' );
  map(x->x-96,resu);
  end;
```

chaine2intlist := **proc**(*mot*)
local *l*, *resu*;
l := *length*(*mot*);
resu := *convert*(*mot*, `bytes`);
map(**proc**(*x*) **option** *operator*, *arrow*; *x* – 96 **end proc** , *resu*)
end proc

```
> afficherMot:= proc(tab,i,k)
   print(tab[i..i+k-1]);
  end:
```

```
> afficherSuffixek := proc(tab,k)
  local tailleT;
  tailleT := taille(tab);
  print(tab[k..tailleT]);
  end:
```

```
> imot1:=chaine2intlist(mot1);
```

$$imot1 := [2, 15, 14]$$

```
> imot2:=chaine2intlist(mot2);
```

$$imot2 := [2, 15, 14, 2, 15, 14]$$

```
> imot3:=chaine2intlist(mot3);
```

$$imot3 := [2, 15, 14, 14, 5, 20]$$

```
> itexte:=chaine2intlist(texte);
```

$$itexte := [17, 21, 5, 12, 2, 15, 14, 2, 15, 14, 2, 15, 14]$$

```
> taille:=t->nops(t);
```

$$taille := t \rightarrow nops(t)$$

```
> taille(chaine2intlist(mot2));
```

$$6$$

```
> afficherMot(itexte,3,4);
```

$$[5, 12, 2, 15]$$

Partie I

Question 1

On réalise un parcours de mot et texte, en s'arrêtant lorsque mot[i] est différent de texte[i+k-1] ou que l'on a atteint la fin du mot

Attention à prévoir le cas où le mot dépasse du texte.

```
> enTetedeSuffixe:=proc (mot,texte,k) #vu comme des tableaux d'entiers
  local resu,i,m;
  m:=taille(mot);
  #print(m);
  if (k+m > 1+taille(texte))
     then
       resu :=false; #ca depasse!
     else
       resu:=true;i:=1;                          #chaines indexées de 1 à
  N=length
       while(resu and i<=m) do
          if mot[i] <> texte [i+k-1]
              then resu:=false
          end if;
          i:=i+1;
       end do;
```

```
  end if;
  #if resu then print("trouve"); endif;
  resu;
  end:
```

```
> enTetedeSuffixe(imot1,itexte,5);
```

true

```
> enTetedeSuffixe(imot2,itexte,5);
```

true

```
> itexte[5..10];
```

[2, 15, 14, 2, 15, 14]

```
> imot2;
```

[2, 15, 14, 2, 15, 14]

Question 2

Pour rechercher le mot dans le texte, on texte successivement si il est en tête d'un suffixe de tab. On s'arrête lorsqu'on dépasse ou que l'on a trouvé le mot

```
> rechercherMot :=proc (mot,tab)
  local tailleM, tailleT,i,trouve;
  tailleM := taille(mot);
  tailleT := taille(tab);

  trouve:=false;
  i:=1;
  while (not(trouve) and (i<=tailleT-tailleM+1)) do #arret qd on est a la
  fin OU qu'on a trouve.
    trouve:=enTetedeSuffixe(mot,tab,i);
    i:=i+1;
    #print(i);
  end do;
  trouve;
  end:
>
```

```
> rechercherMot(imot1,itexte);
```

true

```
> rechercherMot(imot3,itexte);
```

false

```
> rechercherMot(imot2,itexte);
```

true

Question 3

Pour compter le nombre d'occurrences, cette fois on teste tous les suffixes du texte.

```
> compterOccurrences :=proc (mot,tab)
  local tailleM, tailleT,i,nbocc;
  tailleM := taille(mot);
  tailleT := taille(tab);
  i:=1;
  nbocc :=0;
  while (i<=tailleT-tailleM+1) do #Cette fois on teste tout
       #print("test du decalage = ",i);
       if (enTetedeSuffixe(mot,tab,i)) then
            nbocc :=nbocc+1;
       end if;
       i:=i+1;
  end do;
  nbocc;
  end:
```

```
> compterOccurrences (imot1,itexte);
```

3

```
> compterOccurrences(imot2,itexte);
```

2

```
> compterOccurrences(imot3,itexte);
```

0

Question 4

On commence par créer un "tableau" resu de taille 26 dont les cases sont initialisées à 0. Ensuite on parcourt le texte "tab" lettre par lette et on incrémente la case correspondante dans le tableau resu. Enfin on retourne resu

```
> frequenceLettres:=proc (tab) #utiliser compterOccurrences est une
  heresie algorithmique ici
  local resu,tailleT,c,i;
  tailleT:=taille(tab);

  resu := allouer(26);
  for i from 1 to 26 do #inutile mais demandé par l'énoncé
     resu[i] :=0;
  end do;


  for i from 1 to tailleT do #un seul parcourt du texte suffit !
      c := tab[i]; # "caractere" courant du texte
      resu[c] := resu[c]+1;
  end do;

  resu;
  end:
```

```
> frequenceLettres(itexte);
```

[0, 3, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 3, 3, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0]

Question 5

Prendre les paires mot[i],mot[i+1] et regarder le nb d'occ dans le texte à gauche de i, en passant en parametre de la fonction compteOccurrences le sous tableau mot[0..i]. Atroce algorithmiquement car plein de copies, mais l'enoncé dit ok.

```
> afficherFrequenceBigrammes :=proc(tab)
  local bigramme,texte,i,tailleT;

  tailleT:=taille(tab);

  for i from 1 to tailleT-1 do
      bigramme := tab[i..(i+1)];
      texte := tab[1..i];
      if compterOccurrences(bigramme,texte) = 0 then #je sais qu'il existe
  et que j'ai la premiere occ sous les yeux
           afficherMot(tab,i,2);
           print(compterOccurrences(bigramme,tab));
      end if;
  end do;

  end:
>
```

```
> afficherFrequenceBigrammes(itexte);
                                  [17, 21]
                                     1
                                  [21, 5]
                                     1
                                  [5, 12]
                                     1
                                  [12, 2]
                                     1
                                  [2, 15]
                                     3
                                  [15, 14]
                                     3
                                  [14, 2]
                                     2
```

```
> itexte;
                  [17, 21, 5, 12, 2, 15, 14, 2, 15, 14, 2, 15, 14]
```

Partie II Tableau des suffixes

Question 6

Pour comparer des suffixes on parcourt parallèlement les deux sous-tableaux. dès qu'une lettre tab[k1+i] est différente de tab[k2+i] on peut conclure. on s'arrête dans ce cas, ou alors lorsque l'on a dépassé la borne de droite du mot. dans ce cas c'est le mot le plus petit qui est inférieur. Ici je n'utilise pas de retour anticipé (return), mais on aurait pu.

```
> comparerSuffixes := proc (tab,k1,k2)
  local i,r,tailleT,fini;

  tailleT:=taille(tab);
  fini :=false;
  i:=0;
  r:=0;

  #print("comparaison de ",tab[k1..tailleT],"et",tab[k2..tailleT]);

  if (k1=k2) then r:=0;
  else
     while (k1+i <= tailleT) and (k2+i <= tailleT) and not(fini) do
          if tab[k1+i] < tab[k2+i] then r:=-1; fini:=true;
          elif tab[k1+i] > tab[k2+i] then r:=1; fini:=true;
          else i:=i+1;
          end if;
     end do;
     if not(fini) then
         if (k1+i <=tailleT) then r:=1;
         elif (k2+i <= tailleT) then r:=-1;
         else print("should not happen");
         end if;
     end if;
  end if;
  r;
  end:
```

réponses sucessives attendues, -1, 0 et 1

```
> comparerSuffixes(itexte,5,1);
                                    -1

> comparerSuffixes(itexte,4,4);
                                     0

> comparerSuffixes(itexte,12,11);
                                     1
```

Question 7

Comme suggéré dans l'énoncé, on commence par créer un "tableau" [1,2,....,tailleMot] puis on le trie selon l'ordre induit par la fonction de comparaison précédente.

Pour le développement, on écrit d'abord un tri (selection ici), on teste, puis ensuite, on remplace la fonction de comparaison sur les entiers par celle écrite plus haut sur les suffixes de tab correspondants. Évidement sur une copie on ne met pas la version sur les entiers !

```
> calculerSuffixes_aux :=proc(tab)
  local tabS,tailleT,i,j,imin,temp;
```

```
  tailleT := taille(tab);
  tabS:=allouer(tailleT);

  #initialisation avec les indices 1 à tailleT

  for i from 1 to tailleT do
      tabS[i] :=tailleT-i+1;
  end do;

  print(tabS);
  #tri selection : je prend le min et je le mets à la case 1, puis le
  2ième min et je le mets à la case 2, etc

  for i from 1 to tailleT -1 do
       imin:=i;
       for j from i+1 to tailleT  do #selection du ième min
         if tabS[j] < tabS[imin] then imin:=j ;
         end if;
       end do;

       #swap de tab[i] avec tab[imin]
       temp := tabS[i];
       tabS[i] := tabS[imin];
       tabS[imin] := temp;

  end do;
  tabS;
  end:
```

Vérification que cela trie bien, au moins sur un tableau dans l'ordre inverse.

```
> calculerSuffixes_aux(itexte);
```

[13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

Maintenant on copie-colle et on change la fonction de comparaison

```
> calculerSuffixes :=proc(tab)
  local tabS,tailleT,i,j,imin,temp;

  tailleT := taille(tab);
  tabS:=allouer(tailleT);

  #initialisation avec les indices 1 à tailleT

  for i from 1 to tailleT do
      tabS[i] :=i;
  end do;

  #print(tabS);
  #tri selection : je prend le min et je le mets à la case 1, puis le
  2ième min et je le mets à la case 2, etc

  for i from 1 to tailleT -1 do
       imin:=i;
       for j from i+1 to tailleT   do #selection du ième min au sens lex
  !!
         if (comparerSuffixes(tab,tabS[j],tabS[imin]) < 0)
             then imin:=j ;
         end if;
       end do;

       #swap de tabS[i] avec tabS[imin]
  #     print ("swap de
  ",afficherSuffixek(tab,tabS[i]),"avec",afficherSuffixek(tab,tabS[imin]))
  ;
       temp := tabS[i];
       tabS[i] := tabS[imin];
       tabS[imin] := temp;

  end do;
  tabS;
  end:
```

```
> calculerSuffixes(itexte);
```

[11, 8, 5, 3, 4, 13, 10, 7, 12, 9, 6, 1, 2]

```
> itexte;
```

[17, 21, 5, 12, 2, 15, 14, 2, 15, 14, 2, 15, 14]

Et on vérifie qu'on obtient bien celui de l'énoncé !

PARTIE III

Question 8 .

Comparaison d'un mot avec un suffixe, presque un copier coller de la fonction comparaison de suffixes... ici on compare mot[i] avec tab[k+i-1], sauf si ça déborde !

```
> comparerMotSuffixe := proc (mot,tab,k)
  local i,r,tailleT,tailleM,fini;

  tailleT:=taille(tab);
  tailleM :=taille(mot);
  fini :=false;
  i:=1;
  r:=0;

  #print (mot,tab[k..tailleT]);

  while (k+i-1 <= tailleT) and (i<=tailleM) do
      if mot[i] < tab[k+i-1] then return -1 ;
      elif mot[i] > tab[k+i-1] then return 1 ;
      else
         i:=i+1;
      end if;
  end do;
```

```
  #print(i);
  if i<tailleM then return 1; end if; #on avait pas fini de lire le mot a
  la fin du texte, le mot est donc supérieur
  return 0;

  end:
>
> totomot:=[15,14];itexte;
                                totomot := [15, 14]
                    [17, 21, 5, 12, 2, 15, 14, 2, 15, 14, 2, 15, 14]
> comparerMotSuffixe(totomot,itexte,1);
                                        -1
> comparerMotSuffixe(totomot,itexte,3);
                                         1
> comparerMotSuffixe(totomot,itexte,12);

                                         0
> comparerMotSuffixe(imot2,itexte,11);
                                         1
```

Question 9.

Recherche dicho dans "tableau" trié, comme dans la question 8 on va d'abord écrire l'algorithme sur un tableau d'entiers. On fait attention à bien traiter les petits cas. Pareil, sur une copie on ne met que la version avec la fonction de comparaison écrite plus haut.

```
> rechercheTabTrie_aux:=proc(tab,num)

  local imin,imax,imilieu;
  imin:=1;
  imax:=taille(tab);

  while(imin <imax) do
      imilieu := iquo(imax + imin,2);
    #  print(imilieu);
      if (num = tab[imilieu]) then return true;
      elif num < tab[imilieu] then imax:=imilieu ;
      else imin:=imilieu +1;
      end if;
  end do;
  return false;

  end:
> test:=[-2,-1,4,23,71,80,900];
                          test := [-2, -1, 4, 23, 71, 80, 900]
> rechercheTabTrie_aux(test,80);
                                      true
> rechercheTabTrie_aux(test,4);
                                      true
```

Et maintenant recherche dicho dans le tableau tabS en utilisant toujours la fonction de comparaison sur les mots.

```
> rechercherMot2:=proc(mot,tab,tabS)

  local imin,imax,imilieu,r;
  imin:=1;
  imax:=taille(tab);

  while(imin <=imax) do
      imilieu := iquo(imax + imin,2);
      r := comparerMotSuffixe(mot,tab,tabS[imilieu]);
      if (r=0) then #print("trouve a",imilieu);
                    return true;
      elif r < 0 then imax:=imilieu -1
      else imin:=imilieu +1;
      end if;
  end do;
  return false;
  end:
> tabS:=calculerSuffixes(itexte);
                    tabS := [11, 8, 5, 3, 4, 13, 10, 7, 12, 9, 6, 1, 2]
> rechercherMot2(imot1,itexte,tabS);
                                      true
> rechercherMot2(imot3,itexte,tabS);
                                      false
> rechercherMot2(imot2,itexte,tabS);
                                      true
```

Question 11

Cette fonction est très similaire à la précédente. Au lieu de s'arrêter lorsqu'on a trouvé, on stocke l'indice et on continue sur le sous-tableau de gauche, pour éventuellement trouver un indice plus petit.

```
> rechercherPremierSuffixe:=proc(mot,tab,tabS)

  local imin,imax,imilieu,r,indicetrouve;
  imin:=1;
  imax:=taille(tabS);

  indicetrouve := -1;

  while(imin <=imax) do
      imilieu := iquo(imax + imin,2);
```

```
      #print(imin,imax,imilieu,tabS[imilieu]);
      r := comparerMotSuffixe(mot,tab,tabS[imilieu]);
      if r=0 then indicetrouve :=imilieu; end if;
      if r <= 0 then    #meme si r=0 on cherche a gauche un éventuel
  indice plus petit
           imax:=imilieu -1; #print("a gauche");
      else
           imin:=imilieu +1; #print("a droite");
      end if;
  end do;
  return indicetrouve;
  end:
```

```
> rechercherPremierSuffixe(imot1,itexte,tabS);#should be 1
```

1

```
> rechercherPremierSuffixe(imot2,itexte,tabS);#should be 2
```

2

```
> imot1,imot2,imot3,itexte;tabS;
```

[2, 15, 14], [2, 15, 14, 2, 15, 14], [2, 15, 14, 14, 5, 20],
[17, 21, 5, 12, 2, 15, 14, 2, 15, 14, 2, 15, 14]

[11, 8, 5, 3, 4, 13, 10, 7, 12, 9, 6, 1, 2]

```
> comparerMotSuffixe(imot2,itexte,8);
```

0

```
> mot2;
```

"bonbon"

copier –coller pour rechercher dernier (non demandé dans l'énoncé)

```
> rechercherDernierSuffixe :=proc(mot,tab,tabS)
  local imin,imax,imilieu,r,indicetrouve;
  imin:=1;
  imax:=taille(tabS);

  indicetrouve := -1;

  while(imin <=imax) do
      imilieu := iquo(imax + imin,2);
      #print(imin,imax,imilieu,tabS[imilieu]);
      r := comparerMotSuffixe(mot,tab,tabS[imilieu]);
      if r=0 then indicetrouve :=imilieu; end if;
      if r < 0 then    #meme si r=0 on cherche a gauche un éventuel indice
  plus petit
           imax:=imilieu -1; #print("a gauche");
      else
           imin:=imilieu +1; #print("a droite");
      end if;
  end do;
  return indicetrouve;
  end:
```

```
> rechercherDernierSuffixe(imot1,itexte,tabS);#should be 3
```

3

```
> rechercherDernierSuffixe(imot2,itexte,tabS);#should be 3
```

3

Question 12 :

Le tableau des préfixes est trié par ordre alphabétique. Donc, si i1 est la case du premier suffixe dont mot est un prefixe, et i2 la case du dernier suffixe, toutes les cases entre i1 et i2 ont aussi mot comme préfixe. L'algo n'est pas optimal, on aurait pu rechercher les deux indices avec une seule fonction de recherche dicho.

```
> compterOccurrences :=proc (mot,tab,tabS)
  local i1,i2;
  i1 := rechercherPremierSuffixe(mot,tab,tabS);
  i2 := rechercherDernierSuffixe(mot,tab,tabS);

  if i1=-1 then return 0 ; else
     return i2-i1+1;
  end if;
  end:
```

```
> compterOccurrences(imot3,itexte,tabS);
```

0

```
> compterOccurrences(imot2,itexte,tabS);
```

2

```
> compterOccurrences(imot3,itexte,tabS);
```

0

Question 13

Calcul de l'ensemble des sous-mots de taille k d'un mot donné. La lecture du tableau des suffixes donne directement les occurrences des kgrammes. Exemple pour k=2, les 3 premières cases donnent 3*"bo", ensuite, 1*" el", etc.

```
> afficherFrequencesKGrammes:= proc(tab,tabS,k)
  local kgramme,i,j,nbocc,tailleS,dec;

  tailleS :=taille(tabS);
  kgramme := allouer(k); #inutile mais imposé par l'énoncé
  j:=1; # j va être l'indice de parcours de tabS

  while(j<=tailleS) do  #parcours de tab
       dec := tabS[j]; #suffixe numero dec du mot.
       if (dec+k > tailleS) then j:=j+1;  #ce decalage ne correspond pas à
  un kgramme, ça dépasse !
       else
          kgramme := tab[dec..dec+k]; #copie du kgramme;
          afficherMot(tab,dec,k);
          nbocc:=1;
          j:=j+1;
          while(j<= tailleS and comparerMotSuffixe(kgramme,tab,tabS[j])=0)
```

```
  do #kgramme est un prefixe de tab[tabS[j]]...
             j:=j+1;
             nbocc:=nbocc+1;
          end do;
          #ici on a le compte du nb d'occurrences de kgramme
          print("apparait",nbocc,"fois")
       end if;
  end do;

  end:
```

```
> afficherFrequencesKGrammes(itexte,tabS,2);
                                    [2, 15]
                                "apparait" , 3, "fois"
                                    [5, 12]
                                "apparait" , 1, "fois"
                                    [12, 2]
                                "apparait" , 1, "fois"
                                    [14, 2]
                                "apparait" , 2, "fois"
                                    [15, 14]
                                "apparait" , 2, "fois"
                                    [17, 21]
                                "apparait" , 1, "fois"
                                    [21, 5]
                                "apparait" , 1, "fois"
```

```
> afficherFrequencesKGrammes(itexte,tabS,4);
                                  [2, 15, 14, 2]
                                "apparait" , 2, "fois"
                                  [5, 12, 2, 15]
                                "apparait" , 1, "fois"
                                  [12, 2, 15, 14]
                                "apparait" , 1, "fois"
                                  [14, 2, 15, 14]
                                "apparait" , 1, "fois"
                                  [15, 14, 2, 15]
                                "apparait" , 2, "fois"
                                  [17, 21, 5, 12]
                                "apparait" , 1, "fois"
                                  [21, 5, 12, 2]
                                "apparait" , 1, "fois"
```

Et voilà !