X : MP-PC 2007

Proposition de correction

Correction réalisée avec Maple V release 5.1

Note : il y a dans ce qui suit plusieurs instructions de "test" non affichées, et à 2 reprises elles provoquent des modifications de variables globales ; donc : être prudent à l'exécution.

```
> restart:
```

```
> tabtest:=array(1..11,[0,0,3,2,3,3,3,3,3,3,5]):n:=11:
```

Question1

```
> occurences:=proc(t,n) local r,i,j,compteur;
  r:=array(1..256);
  for i from 1 to 256 do
      compteur:=0;
      for j from 1 to n do
             if t[j]=i then compteur:=compteur+1 fi;
      r[i]:=compteur     od;    od;
  eval(r)
  end:
```

```
> occurences(tabtest,n):
```

Question 2

```
> mini:=proc(t,n) local marc,rloc,nb,x,i;
  marc:=1;rloc:=occurences(t,n);nb:=rloc[1];
  for i from 2 to 256 do
               x:=rloc[i];
               if x<nb then marc:=i;nb:=x fi   od;
  marc
  end:
```

```
> mini(tabtest,n):
```

Question 3

Nous allons d'abord écrire une procédure qui calcule le nombre d'occurences contigues d'une lettre à partir d'une position donnée.

```
> nbrepet:=proc(t,i,n) local nbr,j;
  nbr:=1; j:=0;
  while (i+j+1<=n) and (t[i+j+1]=t[i]) do nbr:=nbr+1;j:=j+1 od;
  nbr
  end:
```

```
> nbrepet(tabtest,3,11):
```

```
> tailleCodage:=proc(t,n) local nprime,i,pos,rep;
  nprime:=1;  # le marqueur prend une place
  pos:=1;
  while pos<=n do rep:=nbrepet(t,pos,n);
                  if rep=1 then nprime:=nprime+1;pos:=pos+1
                           else nprime:=nprime+3;pos:=pos+rep fi  od;
  nprime
  end:
```

```
> tailleCodage(tabtest,n):
```

Remarque : le processus n'est pas avantageuxsi la plupart des répétitions sont courtes.

Question 4

```
> codage:=proc(t,n) local tprime,pos,i,rep,marc;
  tprime:=array(1..tailleCodage(t,n));
  marc:=mini(t,n);tprime[1]:=marc;pos:=1;i:=2;
  while pos<=n do
      rep:=nbrepet(t,pos,n);
      if rep=1 then tprime[i]:=t[pos];pos:=pos+1;i:=i+1
               else tprime[i]:=marc;tprime[i+1]:=rep-1;
                    tprime[i+2]:=t[pos];pos:=pos+rep;i:=i+3 fi   od;
  eval(tprime)
  end:
```

```
> codage(tabtest,n):
```

Question 5

Nous écrivons d'abord une procédure qui effectue i rotations.

```
> tabtest2:=array(1..8,["c","o","n","c","o","u","r","s"]):n:=8:
```

```
> tourner:=proc(t,n,i) local k,tb;
  tb:=array(1..n);
  for k from i+1 to n do tb[k-i]:=t[k] od;
  for k from 1 to i do tb[n-i+k]:=t[k] od;
  eval(tb)
  end:
```

```
> tourner(tabtest2,n,4):
```

```
> comparerRotations:=proc(t,n,i,j) local ti,tj,comp,k;
  ti:=tourner(t,n,i); tj:=tourner(t,n,j);
  comp:=0; k:=1;
  while (comp=0) and (k<=n) do
     if ti[k]<tj[k] then comp:=-1 elif ti[k]>tj[k] then comp:=1
                    else k:=k+1 fi od;
  comp
  end:
```

```
> comparerRotations(tabtest2,n,2,4):
```

Voici une version plus élégante mais qui nécessite de bien connaitre Maple :

```
> cR:=proc(t,n,i,j) local ti,tj,comp,k;
  ti:=cat(seq(tourner(t,n,i)[k],k=1..n));
  tj:=cat(seq(tourner(t,n,j)[k],k=1..n));
  if ti=tj then 0 elif ti<tj then -1 else 1 fi
  end:
```

```
> cR(tabtest2,n,1,3):
```

Préalable à la question 6 : il s'agit de construire la fonction *triRotations,* sans laquelle on ne peut pas continuer la programmation. Signalons que *Maple* n'est vraiment pas fait pour cela!

Nous allons adapter une procédure de *tri-rapide* ; commençons par construire l'ensemble des rotations et appliquons la procédure classique (ce n'est évidemment pas le moyen le plus économique).

```
> lesrotations:=proc(t,n) local k,LesRt;
  LesRt:=array(0..n-1); LesRt[0]:=eval(t);
  for k from 1 to n-1 do LesRt[k]:=tourner(t,n,k)  od;
  eval(LesRt)
  end:
```

```
> Trot:=lesrotations(tabtest2,n):
```

```
> rtest:=array(0..n-1,[seq(i,i=0..n-1)]):
```

```
> echange:=proc(T,r,i,j) local x,y,rbis,Tbis;
  rbis:=eval(r); Tbis:=eval(T);
  x:=rbis[i];rbis[i]:=rbis[j];rbis[j]:=x;
  y:=eval(Tbis[i]);Tbis[i]:=eval(Tbis[j]);Tbis[j]:=eval(y);
  end:
```

Nous introduisons une petite variante de cR, de façon à utiliser directement le tableau des rotations.

```
> cR2:=proc(T,n,i,j) local ti,tj,comp,k;
  ti:=cat(seq(eval(T[i])[k],k=1..n));
  tj:=cat(seq(eval(T[j])[k],k=1..n));
  if ti=tj then 0 elif ti<tj then -1 else 1 fi
  end:
```

```
> partition:=proc(T,r,n,deb,fin) local m,i;
  m:=deb;
  for i from deb+1 to fin do
    if cR2(T,n,deb,i)=1 then echange(T,r,m+1,i);m:=m+1 fi od;
  echange(T,r,deb,m);
  m
  end:
```

```
> partition(Trot,rtest,n,1,6):eval(rtest):eval(Trot):
```

```
> Trot:=lesrotations(tabtest2,n): #remettre à niveau le tableau initial si
          #l'on a testé la procédure partition
  rtest:=array(0..n-1,[seq(i,i=0..n-1)]):
```

```
> triRotations:=proc(T,r,deb,fin) local m,Tbis,rbis;
  Tbis:=eval(T);rbis:=eval(r);
  if fin>deb then m:=partition(eval(Tbis),eval(rbis),n,deb,fin);
                  triRotations(eval(Tbis),eval(rbis),deb,m-1);
                  triRotations(eval(Tbis),eval(rbis),m+1,fin)   fi;
  eval(rbis)
  end:
```

```
> triRotations(Trot,rtest,0,7):
```

Remarquer que ce processus fournit également la liste triée des rotations, ce qui simplifie la question suivante.
Question 6 Ne sachant pas numéroter les caractères alphanumériques en Maple, je suis contraint d'en passer par la procédure suivante (évidemment, il serait plus commode d'employer un modulo ...).

```
> numalpha:=proc(car)  local lettre, nb ,j,i;
  lettre:=array(1..26,
  ["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"]);
  nb:=array(1..26,[seq(i,i=1..26)]);
  j:=1;
```

```
  while car<>lettre[j] do j:=j+1    od;
  j
  end:
```

```
> codageBW1:=proc(t,n) local lesrot,code,i;
  code:=array(0..n);
  for i from 0 to n-1 do code[i]:=eval(Trot[i])[n] od;
  code[n]:=numalpha(t[1]);
  eval(code)
  end:
```

```
> codageBW1(tabtest2,n):
```

En se référant à la lettre de l'énoncé, on utilise le tableau r donné par la procédure *triRotations* ainsi qu'une procédure de construction d'une rotation (qui existe forcément puisqu'on en a besoin pour *comparerRotations),* telle que la procédure *tourner* . J'utilise ci-dessous la procédure *triRotations* telle que constituée plus haut, un peu différemment du texte.

```
> codageBW2:=proc(t,n) local lesrot,unerot,code,rtest,i,r;
  code:=array(0..n); unerot:=array(1..n);
  rtest:=array(0..n-1,[seq(i,i=0..n-1)]):
  lesrot:=eval(lesrotations(t,n));
  r:=eval(triRotations(lesrot,rtest,0,n-1));
  lesrot:=eval(lesrotations(t,n)); # je remets le tableau des rotations à
  sa                                                                    #
  valeur initiale
  for i from 0 to n-1 do
      unerot:=eval(lesrot[r[i]]); #emploi du tableau r
      code[i]:=unerot[n]       od;
  code[n]:=numalpha(t[1]);
  eval(code)
  end:
```

```
> codageBW2(tabtest2,n):
```

Question 7

la procédure *triRotations* est en temps O(n.ln(n)) *appels à* *comparerRotations* qui est en temps linéaire, donc triRotations est en temps O(n^2.ln(n)). Pour le reste il faut essentiellement constituer n rotations d'un mot de n lettres, ce qui réclame un temps en O(n^2). Globalement, le temps nécessaire pour *codageBW* est donc en O(n^2.ln(n) .

Question 8

Je considère dans la suite que le texte t' est donné sous forme d'un tableau de caractères. Ne disposant pas d'un couple codage/décodage ASCII en Maple V Release 5.1 , je me contente des 26 lettres de l'alphabet, numérotées de 1 à 26.

```
> n:=8:np:=n+1: tptest:=array(1..np,["s","n","o","c","c","u","r","o",3]):
```

```
> alphanum:=proc(m) local alphabet;
           #transformer un nombre en lettre
  alphabet:=array(1..26,["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"]);
  eval(alphabet[m])
  end:
```

-

```
> frequences:=proc(tp,np) local tabfreq,i,j,compt;
  tabfreq:=array(1..26);
  for i from 1 to 26 do
       compt:=0;
       for j from 1 to np-1 do
           if numalpha(tp[j])=i then compt:=compt+1 fi  od;
       tabfreq[i]:=compt    od;
  eval(tabfreq)
  end:
```

```
> frequences(tptest,np):
```

Question 9

```
> triCarsDe:=proc(tpr,npr) local tabfreq,pos,i,j,tabDe;
  tabfreq:=eval(frequences(tpr,npr));
  tabDe:=array(1..npr-1);
  pos:=1;
  for i from 1 to 26 do
     if tabfreq[i]<>0 then
       for j from 0 to tabfreq[i]-1 do tabDe[pos+j]:=alphanum(i) od;
       pos:=pos+tabfreq[i]  fi     od;
  eval(tabDe)
  end:
```

```
> triCarsDe(tptest,np):
```

Question 10

```
> trouverIndices:=proc(tpr,npr) local tDe,indices,freq,i,j,num,pos,rep;
  tDe:=eval(triCarsDe(tpr,npr));
  indices:=array(1..npr-1);
  freq:=eval(frequences(tpr,npr));
  pos:=1;
  while pos<=npr-1 do
     i:=1;
     while tpr[i]<>tDe[pos] do i:=i+1  od; indices[pos]:=i;
     rep:=freq[numalpha(tDe[pos])];
     for j from 2 to rep do
           i:=i+1; pos:=pos+1;
           while eval(tpr[i])<>eval(tDe[pos]) do i:=i+1 od;
           indices[pos]:=i od;
     pos:=pos+1;
  od;
  eval(indices)
  end:
```

```
> trouverIndices(tptest,np):
```

Remarque: ici, le tableau *indices* est numéroté à partir de 1 et non à partir de 0 comme donné dans l'énoncé sur l'exemple .

Question 11

```
> decodageBW:=proc(tpr,npr)  local taborigine,indices,nl,j,j1;
  taborigine:=array(1..npr-1);
  indices:=eval(trouverIndices(tpr,npr));
  j:=indices[1]; taborigine[1]:=tpr[j];
  for nl from 2 to npr-1 do
```

```
    j1:=indices[j]; taborigine[nl]:=tpr[j1]; j:=j1   od;
eval(taborigine)
end:
> decodageBW(tptest,np):
```