Épreuve d'informatique de l'X - 2007 - MP/PC

Compression `bzip`

Question 1 : On commence par initialiser le tableau `r` des résultats (lignes 4 à 6), puis on parcourt le tableau à compresser et pour chaque lettre rencontrée, on incrémente le contenu de la case du tableau résultat correspondante (ligne 8).

```
1  occurrences := proc(t::array,n::integer)
2  local r, i;
3  r := array(0..256);
4  for i from 0 to 256 do
5    r[i] := 0
6  od;
7  for i from 0 to n-1 do
8     r[t[i]] := r[t[i]] + 1
9  od;
10 return(r);
11 end;
```

Question 2 : On commence par créer le tableau des occurrences, puis ensuite, il s'agit d'un parcours de tableau pour trouver l'indice correspondant au plus petit élément du tableau.

```
1  mini := proc(t::array,n::integer)
2  local r, i, i0, m;
3  r := occurrences(t,n);
4  m := r[0];
5  i0 := 0;
6  for i from 1 to n do
7    if r[i]<m then m:=r[i]; i0 := i fi;
8  od;
9  return(i0);
10 end;
```

Question 3 : La variable `compt` permet de calculer le nombre d'apparitions successives du caractère courant (`CaracCourant`). Si ce nombre est égal à 1, le caractère sera représenté par lui même et occupera un case dans le tableau du texte compressé (ligne 9) sinon il faudra trois caractères supplémentaires pour représenter la redondance du caractère courant dans le tableau du texte compressé (ligne 10). Les lignes 15 à 18 permettent de prendre en compte le dernier caractère présent dans le texte non compressé.

```
1  tailleCodage := proc(t::array,n::integer)
2  local taille, compt, CaracCourant, i;
3  taille := 1;
4  compt := 1;
5  CaracCourant := t[0];
6  for i from 1 to n-1 do
7    if t[i-1] <> t[i]
8     then
9        if compt = 1 then taille := taille+1
10                    else taille := taille + 3; compt:=1;
11       fi;
12    else compt := compt+1;
13   fi;
14 od;
15 if compt = 1
16    then taille := taille + 1
17    else taille := taille + 3
18 fi;
19 return(taille);
20 end;
```

Question 4 : après avoir récupéré la taille du tableau du texte compressé (variable `nprime`) et avoir créé le tableau correspondant (variable `tprime`), on initialise le tableau `tprime` (ligne 7 à 9), puis on exécute un parcours similaire à celui de la fonction `tailleCodage` en remplaçant les éléments du tableau `tprime`.

```
1  codage := proc(t::array,n::integer)
2  local Marqueur, compt, tprime, pos, i, nprime;
3  Marqueur := mini (t,n);
4  compt := 1;
5  nprime := tailleCodage (t,n);
6  tprime := array(0..nprime-1);
7  for i from 0 to nprime -1 do
8     tprime[i] := Marqueur
9  od;
10 pos := 1;
11 for i from 1 to n-1 do
12   if t[i-1] <> t[i]
13    then
14       if compt = 1 then tprime[pos] := t[i-1]; pos := pos+1
15                    else tprime[pos+1] := compt-1;
16                         tprime[pos+2] := t[i-1];
17                         pos := pos + 3; compt:=1
18       fi;
19    else compt := compt+1;
20   fi;
21 od;
22 if compt = 1
23     then tprime[pos] := t[n-1]
24     else tprime[pos+1] := compt-1; tprime[pos+2] <- t[n-1];
25 fi;
26 return(tprime);
27 end;
```

Question 5 : Pour réaliser cette fonction, on utilise deux pointeurs , `iprime` et `jprime` que l'on déplace jusqu'à ce que les caractères pointés soient distincts ou bien que l'on fait un tour complet (donné par le test `compt < n`). Si on a fait un tour complet, les deux mots sont identiques, on retourne 0 (ligne 12) sinon il suffit de comparer les deux caractères pointés d'indice `iprime` et `jprime` (lignes 13 et 14).

```
1  comparerRotations := proc(t::array,n::integer,i::integer,j::integer)
2  local iprime, jprime, compt;
3  iprime := i;
4  jprime := j;
5  compt := 0;
6  while t[iprime] = t[jprime] and compt < n do
7    iprime := (iprime + 1) mod n;
8    jprime := (jprime + 1) mod n;
9    compt := compt+1;
10 od;
11 if compt = n
12   then return(0)
13   else if t[iprime] > t[jprime] then return(1)
14                                 else return(-1)
15        fi;
16 fi;
17 end;
```

Question 6 :

```
1  codageBW := proc (t::array,n::integer)
2  local rot, res, i;
3  rot := triRotations(t,n);
4  res := array(0..n);
5  for i from 0 to n do res[i] := 0 od;
6  for i from 0 to n-1 do
7    res[i] := t[(rot[i]+n-1) mod n];
8    if rot[i] = 1 then res[n] := i; fi;  # on a trouvé la clef
9  od;
10 return(res);
11 end;
```

Question 7 : Pour calculer la complexité, on somme les coûts du tri ($O(n \ln n)$) et du balayage du tableau fait par la boucle de la ligne 5 et par la boucle des lignes 6 à 9, ce qui nous donne une complexité en $O(nln(n)) + O(n)$ soit en $O(n \ln n)$.

Question 8 : Ne pas oublier que la clé est dans la dernière case et donc ne compte pas comme une lettre ;

```
1  frequence := proc (tprime::array,nprime::integer)
2  local occur, i, j;
3  occur := array(0..255);
4  for i from 0 to 255 do occur[i] := 0 od;
5  for i from 0 to nprime-2 do
6     j := tprime[i];
7     occur[j] :=  occur[j]+1;
8  od;
9  return(occur);
10 end;
```

Question 9 :

```
1  triCarsDe := proc(tprime::array,nprime::integer)
2  local occur, triCars, pointeurs, i, k, pointeur;
3   occur := frequence(tprime,nprime);
4   triCars := array (0..nprime-2);
5   pointeur :=  0;
6  for i from 0 to 255 do
7     for k from 1 to occur[i] do
8         triCars[pointeur] := i;
9         pointeur := pointeur+1;
10     od;
11 od;
12 triCars;
13 end;
```

Question 10 :

```
1  trouverIndices := proc(tprime::array,nprime::integer)
2  local indices, i, triCars, lettre, pos;
3  indices := array (0..nprime-2);
4  for i from 0 to nprime -2 do  indices[i] := 0 od;
5  triCars :=triCarsDe(tprime,nprime);
6  lettre :=  triCars[0];
7  pos :=  0;
8  for i from 0 to nprime-2 do
9    if triCars[i] <> lettre then pos :=  0; lettre := triCars[i] fi;
10   while tprime[pos] <> lettre do pos:=pos+1 od;
11   indices[i] := pos; pos := pos+1;
12 od;
13 return(indices);
14 end;
```

Question 11 :

```
1  decodageBW := proc (tprime::array,nprime::integer)
2  local pos, indices, texte, i;
3   pos := tprime[nprime-1];
4   indices := trouverIndices(tprime,nprime);
5   texte := array (0..nprime-2);
6   for i from 0 to nprime-2 do texte[i] :=  0 od;
7   for i from 0 to nprime-2 do
8        texte[i] := tprime[pos];
9        pos := indices[pos];
10  od;
11  return(texte);
12 end;
```