Un examen d'algorithmique (niveau L2)

I- On a la suite (u_n) définie par la relation de récurrence :

$u_n = 3\ u_{n-1} - 2\ u_{n-2}$, et par les conditions initiales $u_0=0$ et $u_1 = 1$. On se donne un nombre N (par exemple N=20) et on veut connaître u_N. Faire le programme correspondant sans utiliser de tableau ni de récursivité.

II- Anagrammes

A- Combien existe-t-il de mots de longueur 9 constitués de 5 lettres 0 et 4 lettres 1, comme par exemple 010011100 ?

B- On considère les mots de longueur 9 constitués de 4 lettres 0, de 3 lettres 1 et de 2 lettres 2. Combien existe-t-il de tels mots ?

C- Généralisation. L'objectif est d'énumérer tous les mots à base de K lettres, notées de 0 à K-1, sachant que le mot doit contenir *nb*[0] lettres 0, *nb*[1] lettres 1, …, *nb*[K-1] lettres K-1. Par exemple pour l'exemple précédent du B-, on se donne K=3, *nb*[0]=4, *nb*[1]=3, *nb*[2]=2.

1) On veut connaître la longueur N de ces mots. Faire le programme qui calcule N (étant entendu que K et les *nb*[] sont donnés quelconques au préalable).

2) Comme on veut énumérer tous les mots dans l'ordre alphabétique, on commence par fabriquer le premier mot. Dans l'exemple précédent, c'est 000011122. On mettra ce mot dans un tableau *a*[] sur la longueur N.

Faire le programme qui permet d'avoir ce premier mot (K et les *nb*[] étant donnés quelconques).

3) La suite du programme consiste à faire tourner une boucle *for*(;;) qui permet de passer d'un mot au suivant. Dans l'exemple, on obtient la succession des mots : 000011122, 000011212, 000011221, 000012112, 000012121…

a) En prenant l'exemple du mot 0200211100, quel est le mot qui est juste derrière lui ?

b) On vous redonne ci-dessous le programme des permutations de longueur N formées des N lettres 0, 1, 2, ….N-1. Rappelons que pour trouver le pivot, la première lettre qui change derrière un bloc fixe le plus long possible, on va de la droite vers la gauche en attendant la première descente. Puis on cherche le remplaçant du pivot, on fait l'échange, et enfin, derrière le pivot, on met tout à l'envers.

```
on entre la première permutation 012… dans le tableau a[N]
for( ;;)
{
afficher le tableau a[N]
i=N-1 ;
while (i>0 && a[i]<a[i-1]) i--;
pospivot=i-1;  if (pospivot==-1) break;
pivot=a[pospivot];

i=N-1;
while (a[i]<pivot) i--;
posremplacant= i;

aux=a[pospivot]; a[pospivot]=a[posremplacant]; a[posremplacant]=aux;

g=pospivot+1 ; d= N-1 ;
while(g<d) { aux=a[g] ; a[g]=a[d] ; a[d]=aux ;  g++ ; d-- ;}
}
```

Passons maintenant aux anagrammes. Comment avoir le pivot ? Puis faire les (toutes) petites modifications dans le programme des permutations pour avoir celui

des anagrammes (on se contentera d'indiquer ce qu'il convient de changer, sans tout réécrire).

III- On a un paquet de *N* cartes avec *N* pair. En vue de la programmation, on les dispose dans un paquet *a*[*N*], par exemple pour *N*=6 on pourra prendre

0	1	2	3	4	5
0	1	2	3	4	5

On coupe le paquet en deux parts égales. Puis on met en succession les cartes de la première part dans les positions paires, puis celles de la deuxième part dans les positions impaires. Dans l'exemple choisi cela donne 012345 -> 031425.

Faire le programme du mélange.

2) On répète cette opération de mélange des *N* cartes jusqu'à retrouver l'ordre initial des cartes. On appelle période *T*(*N*) le nombre de fois correspondant. Par exemple 012345 -> 031425 -> 043215 -> 024135 -> 012345. D'où la période *T*(6) = 4. Programmer.

IV- On a deux listes déjà triées, l'une de longueur *N*1 mise dans un tableau *a*[*N*1], l'autre de longueur *N*2 dans un tableau *b*[*N*2]. On veut afficher les éléments qui sont communs dans ces deux tableaux. Par exemple :

1
4
5
6
9
11

2
4
4
5
7

⟶ 4 5

On s'inspirera de l'algorithme de fusion de deux listes déjà triées.

V- Fabrication d'un mot étape par étape

Au départ à l'étape 0 le mot est réduit à une lettre : 0. Puis étape après étape, on procède aux remplacements suivants : 0 est remplacé par les deux lettres 1 2, 1 est remplacé par 2, et 2 est remplacé par 0. Cela donne :

étape 0 : 0, étape 1 : 1 2, étape 2 : 2 0, étape 3 : 0 1 2, étape 4 : 1 2 2 0, étape 5 : 2 0 0 1 2, etc. Programmer la fabrication de ce mot jusqu'à ce que sa longueur dépasse un nombre donné *N*. Pour cela utiliser une liste doublement chaînée circulaire, chaque cellule contenant une lettre du mot.

Corrigé

I- u=0 ; v=0 ; for(i=2 ; i<=N ; i++) {w=3*v-2*u ; u=v ; v=w ;}

II-

A) On a un casier de neuf cases à remplir. On commence par placer les quatre lettres 1. Cela revient à choisir quatre cases parmi neuf, soit C_9^4 =(9.8.7.6)/(4.3.2) = 126. A chaque fois, les 0 prennent les cases restantes. D'où 126 mots.

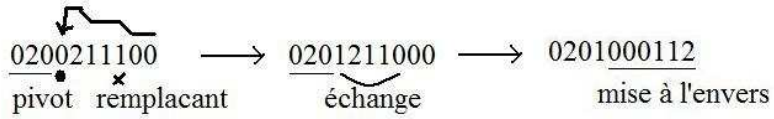
B) On a 9 cases à remplir. On commence par placer les quatre 0, soit C_9^4 cas. A chaque fois, on place les trois 1 dans trois cases choisies parmi les cinq restées vides, soit C_5^3 cas, et les 2 prennent les cases restantes. D'où C_9^4 C_5^3 = 126.10 = 1260 mots.

C1)
```
N=0 ;
for(lettre=0 ;lettre<K ; lettre++)  N+= nb[lettre] ;
```

C2)
```
k=0 ;
for(lettre=0 ; lettre<K ; lettre++)
for(j=0 ;j<nb[lettre] ; j++)
```

a[k++]=lettre ;

C3a)



C3b) Pour avoir le pivot, on va de droite à gauche. Tant que ça monte ou que l'on a un palier, on continue. On s'arrête à la première descente. Idem pour le remplaçant. D'où deux lignes à modifier légèrement dans le programme des permutations pour avoir celui des anagrammes :

while(i>0 && a[i]<=a[i-1]) ….. while(a[i]<=pivot)

III-

1) On entre les cartes de 0 à N-1 dans les cases correspondantes (avec a[i]=i).

On met la deuxième part dans un tableau *b*[] auxiliaire :

```
k=0 ;
for(i=N/2 ;i<N ;i++) b[k++]=a[i] ;
```

On écarte les cartes de la première part (toujours dans a[]) en les mettant en position paire. On fait cela de droite à gauche pour éviter les perturbations :

```
for(i=N/2-1 ; i>=0 ; i--) a[2*i]=a[i] ;
```

On entre b[] dans les cases impaires de a[] :

```
k=0 ;
for(i=1 ;i<N ;i+=2) a[i]=b[k++] ;
```

2)

```
T=0 ; for(i=0 ;i<N ;i++) a[i]=i ;
do { fini=OUI ;
     programme du mélange
    T++ ;
     for(i=0 ; i<N ;i++) if (a[i] !=i) { fini=NON ; break ;}
   }
while(fini==NON) ;
afficher la période T
```

IV- On aménage l'algorithme des têtes coupées

```
i=0 ; j=0 ;
while( !(i==N1 || j==N2)
if (a[i]<b[j]) i++ ;
else if (a[i]>b[j]) j++ ;
else {afficher a[i] ; i++ ; j++ ;}  /* ici  a[i]==b[j]  */
```

Avantage : un seul parcours des listes, pas de double boucle.

V- Déclaration préliminaire :

```
struct cell {int n ; struct cell * s ; struct cell *p ;} ;
```

Embryon de la liste :

```
debut= (struct cell *) malloc(sizeof(struct cell)) ;
debut->n=0 ; debut->s=debut ; debut->p=debut ;
```

Boucle des étapes, jusqu'à ce que la longueur du mot dépasse *N* :

```
longueur=1 ; etape=0 ;
while (longueur<=N)
{
etape++ ;
```

```
ptr=debut ;
do{  apres = ptr->s ;
       if (ptr->n==0)
         { ptr->n=1 ; longueur++ ;
           newcell= …malloc …
           newcell->n=2 ; newcell->s=apres ; newcell->p=ptr ;
           ptr->s=newcell ; apres->p=newcell ;
         }
      else if (ptr->n==1) ptr->n=2 ;
      else ptr->n=0 ;
      ptr=apres ;
   }
while (ptr !=debut) ;
}
```