7. Enumérations dans l'ordre alphabétique

Prenons un dictionnaire. Comment savoir si un mot se trouve avant ou après un autre ? On commence par comparer la première lettre de ces deux mots. Si elles sont différentes, on sait comment classer les deux mots. Si elles sont identiques, on compare la deuxième lettre des deux mots. Si elles sont différentes, on sait quel est le mot placé devant l'autre. On continue ainsi tant que les deux mots ont un bloc initial identique, jusqu'à ce que l'on trouve deux lettres différentes. Par exemple prenons les deux mots 001110001 et 001110111 fabriqués à partir des deux lettres (ou chiffres) 0 et 1. Nous avons souligné leur bloc initial identique, les premières lettres différentes étant 0 pour l'un et 1 pour l'autre, ce qui permet d'affirmer que le premier mot est avant le deuxième (puisque 0 est avant 1, ou encore $0 < 1$). C'est ce procédé classique qui constitue le fondement de l'algorithme d'énumération que nous allons développer. Avec cette particularité : nous allons nous-mêmes fabriquer le dictionnaire des mots concernés.

1. Premier exemple : Ecriture de nombres en binaire

Imaginons que l'on veuille énumérer tous les nombres écrits en binaire (avec des 0 et des 1) et ayant N chiffres, soit une longueur N, N étant donné. Pour $N = 4$ par exemple, cela donne :

0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111.

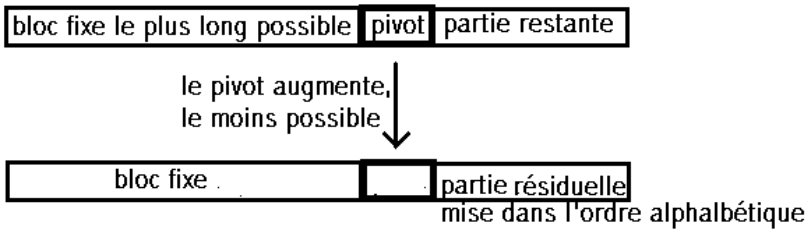
En valeur, ces nombres sont dans l'ordre croissant. Ils sont aussi dans l'ordre alphabétique lorsqu'on les traite comme des mots. Cela évite notamment de répéter deux fois le même mot. Le problème est de trouver la règle permettant de passer d'un mot au suivant, en évitant d'oublier des mots situés entre les deux. Par exemple on sait que 0100 est avant 0110, mais si l'on passe directement de l'un à l'autre, on oublie le mot 0101. Pour éviter ces oublis, on cherche à garder un bloc fixe le plus long possible pour passer d'un mot au suivant. Par exemple si l'on part de 0100, on peut garder le bloc fixe 010, la première lettre qui change est juste après : c'est 0 qui va être transformé en 1. On passe ainsi de 010**0** à 010**1**, et il n'y a aucun mot perdu entre les deux.

2. Principe de l'énumération de mots dans l'ordre alphabétique

Lorsque l'on doit construire un ensemble de mots, on va les écrire les uns après les autres dans l'ordre alphabétique (ou encore ordre lexicographique). On commence par fabriquer le premier mot, le plus petit dans l'ordre alphabétique. Il s'agit ensuite de trouver la règle du jeu qui permet de passer d'un mot au suivant dans l'ordre alphabétique. Enfin, on définit le test d'arrêt qui marque la fin de l'énumération.

Comment avoir la règle du jeu permettant de passer d'un mot à son successeur dans l'ordre alphabétique ? Dans le mot concerné, on cherche à garder un début du mot (un facteur gauche) inchangé, qui soit le plus long possible, de façon à ne pas perdre des mots intermédiaires lors du passage d'un mot au suivant. Derrière ce début inchangé vient la première lettre que l'on doit changer, en la remplaçant par une lettre plus grande, plus précisément la plus petite possible parmi les plus grandes lettres possibles. Cette lettre qui joue un rôle clé, nous l'appelons pivot. Enfin à droite de cette lettre pivot, on met le reste du mot de façon qu'il soit le plus petit possible.

La méthode de passage se résume ainsi :



Nous allons appliquer cela pour notre exemple des nombres en binaire.

3. Programme d'énumération des nombres en binaire de longueur *N*

Ces nombres vont être successivement placés dans le tableau *a*[*N*] indexé de 0 à *N*-1, *N* étant donné. Au départ, on met uniquement des 0 dans ce tableau, ce qui donne le premier nombre. Puis une boucle répétitive permet de passer d'un nombre au suivant. Quelle est la règle de passage? Le pivot ne peut être qu'un 0, et c'est celui qui est le plus loin possible. Il suffit donc de chercher le premier 0 à partir de la droite. C'est lui le pivot. A sa gauche reste le bloc invariant le plus long possible. On va changer le pivot de 0 en 1. Puis tous les chiffres après lui sont mis à 0. Par exemple quel est le mot qui suit 0011**0**111 ? Il s'agit de 0011**1**000, où nous avons fait ressortir la lettre pivot. Le processus de passage d'un mot au suivant est arrêté lorsque le pivot est en position -1. Par exemple lorsque le dernier mot 11111 est obtenu, on va de droite à gauche à la recherche du premier 0, mais on n'en trouve pas, on bloque alors le mouvement lorsque l'on atteint la position -1 : on est sorti du tableau sans trouver de pivot, et c'est fini.

D'où le programme:

```
mettre a[N] à 0
for(; ;)  { afficher le tableau a[]
          i = N-1;   while(i >= 0 && a[i] = =1)  i- - ; /* recherche de la position du pivot */
          if (i= = -1) break;          /*  i  est la position de la lettre pivot  */
          a[i]=1;   for(j= i+1; j<N; j++) a[j]=0;
        }
```

Remarques:

- on a utilisé une boucle *for(;;)* infinie, mais on la bloque par un *break* lorsque le dernier mot a été affiché.
- dans la boucle *while* de recherche du pivot on doit ajouter la contrainte *i* >= 0 pour empêcher que la position *i* dans le tableau *a*[*i*] ne tombe en-dessous de 0, ce qui provoquerait une erreur fatale. Ainsi, lorsque l'on obtient le dernier mot qui est affiché, la boucle *while* s'arrête lorsque *i* prend la valeur -1.

4. Enumération des permutations de *N* objets

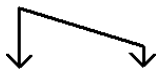
Une permutation de *N* objets est une certaine façon de les mettre dans un certain ordre. Pour nous les objets vont être les nombres entiers pris entre 0 et *N*-1. Par exemple une permutation des cinq objets 0, 1, 2, 3, 4 est 24103. Prenons maintenant toutes les permutations de trois objets, écrites dans l'ordre alphabétique (0<1<2) : 012, 021, 102, 120, 201, 210.

Combien existe-t-il de permutations de *N* objets ? Pour cela, on commence par choisir l'objet à placer en premier, ce qui fait *N* possibilités. A chaque fois que l'on a choisi le premier, on a *N*-1 possibilités pour le choix du deuxième. Puis à chaque fois, on a *N*-2 cas pour le troisième, etc. D'où *N*.(*N*-1).(*N*-2)…. 3.2.1= *N*! permutations. Le nombre de permutations de *N* objets est ***N***!.

Pour nous, le problème essentiel n'est pas de les compter, mais de les énumérer, c'est-à-dire de les obtenir toutes grâce à un procédé répétitif de passage de l'une à la suivante. Pour cela nous allons les

écrire dans l'ordre alphabétique. Prenons un exemple, celui des permutations de 8 lettres 1,2,3,...,8. Considérons la permutation 48657321, et cherchons celle qui va lui succéder. Le bloc gauche le plus long qui n'a pas à changer est 486. Par contre si l'on prenait 4865, il faudrait changer la lettre suivante 7 en une lettre supérieure, et il faudrait prendre celle-ci à sa droite (car à gauche cela reste inchangé), or il n'en existe pas. De même avec les blocs gauches encore plus longs. Avec le bloc inchangé 486, la lettre pivot est le 5. On va la remplacer par une lettre supérieure, la plus petite possible, et située à sa droite. C'est le 7. Les lettres restantes sont 321 ainsi que le 5 qui était la lettre pivot, on les écrit dans l'ordre croissant pour avoir le mot le plus petit possible: 1235. Finalement la permutation qui suit est 48671235.

Considérons les permutations des *N* éléments notés 0, 1, 2, …, *N*-1. La première permutation est justement 0,1,2,…,*N*-1 que l'on met dans un tableau *a*[]. Pour passer d'une permutation à la suivante, on cherche l'élément pivot, laissant à sa gauche un bloc fixe le plus long possible. Pour cela, on part de la droite et l'on va vers la gauche, à la recherche du premier élément qui a un nombre plus grand que lui à sa droite. Autrement dit, on va de droite à gauche tant que ça monte, en attendant la première descente, car tant qu'il se produit des montées, il n'y a pas de lettre qui puisse être remplacée par une autre plus grande située à sa droite. C'est à la première descente que se trouve le pivot. Le nombre qui va remplacer le pivot est le plus petit des nombres situés à sa droite, et supérieur à lui. Pour l'obtenir, il suffit d'aller de droite à gauche, où les nombres sont dans le sens croissant, en attendant d'avoir le premier nombre supérieur au pivot. On procède alors à l'échange entre le pivot et ce nombre. Après cela, la partie droite est encore dans le sens croissant de droite à gauche. Il suffit de la mettre à l'envers pour avoir le plus petit mot dans l'ordre alphabétique. Par exemple, pour *N* = 10, prenons la permutation : 6 0 8 2 5 9 7 4 3 1. On procède comme indiqué :



6 0 8 2 5 9 7 4 3 1 → 6 0 8 2 7 <u>9 5 4 3 1</u> → 6 0 8 2 7 1 3 4 5 9 qui est la permutation suivante.

Programme :

```
On se donne N, pour avoir toutes les permutations des éléments de 0 à N-1
for(i=0 ; i<N ;i++) a[i]=i ;  /* la première permutation */
for(;;)
  { afficher le tableau a[]  où se trouve chaque permutation

    /* Recherche du pivot de droite à gauche. Notons que dans la boucle while on doit faire i>0, pour
empêcher que l'indice i-1 dans a[i-1] ne devienne négatif, ce qui provoquerait une erreur fatale. Par la même
occasion, lorsque l'on  obtenu la dernière permutation avec toutes ses lettres dans l'ordre décroissant, la boucle
while s'arrête pour la première fois avec  i qui prend la valeur 0, la position du pivot devient –1, et l'on arrête
alors  la boucle for( ;;)  par un break */
      i=N-1; while (i>0 && a[i]<a[i-1]) i - - ;    pospivot= i-1;   if (pospivot = = -1) break ;

   /* Recherche de l'élément qui va remplacer le pivot */
      i=N-1; while (a[i]<a[pospivot]) i- - ;  posremplacant= i;

   /* Echange des éléments pivot et son remplaçant */
     aux=a[pospivot] ; a[pospivot]=a[posremplacant] ; a[posremplacant]=aux ;

   /* Mise à l'envers de la partie droite */
     gauche=pospivot+1 ; droite=N-1 ;
     while(gauche<droite) { aux=a[gauche] ; a[gauche]=a[droite] ; a[droite]=aux ;
                            gauche++ ; droite-- ;}
  }
```

Remarquons que dans ce programme, toutes les permutations sont placées à tour de rôle dans le même tableau *a*[]. Même s'il y a des milliards de milliards de permutations, elles seront toutes

affichées sur l'écran. Signalons que 20 !, nombre de permutations de 20 objets, est de l'ordre de deux milliards de milliards.

5. Enumération des combinaisons de *P* objets pris parmi *N*

A partir d'un ensemble de *N* objets numérotés de 0 à *N*-1, on en prend *P* parmi ces *N*, sans tenir compte de l'ordre dans lequel on les prend. On obtient ce que l'on appelle une « combinaison » de *P* objets pris parmi *N*. Par exemple 3,5,8 est une combinaison de trois objets pris parmi 10 si l'on veut, et c'est la même que 5,8,3 ou 8,5,3. Parmi toutes les façons d'écrire cette combinaison, on a le droit - et on a intérêt - de choisir celle qui paraît la plus simple, 3,5,8, où les nombres sont dans l'ordre croissant. Parmi tous les mots qui représentent une même combinaison, cela revient aussi à prendre le mot le plus petit dans l'ordre alphabétique.

Ce que nous voulons, c'est énumérer toutes les combinaisons de *P* objets parmi *N*, en profitant de l'écriture de chacune avec ses lettres mises dans l'ordre alphabétique croissant. Par exemple les combinaisons de 2 objets parmi 4 sont 01, 02, 03, 12, 13, 23. Cela peut être vu autrement. Prendre 2 objets parmi 4 objets placés dans des cases, cela veut dire si l'on préfère mettre deux cases à 0, et laisser les autres à 1. Par exemple, pour la combinaison 13, on met les cases 1 et 3 à 0, et les deux autres à 1, ce qui donne 1010. Ainsi les combinaisons précédentes peuvent s'écrire 0011, 0101, 0110, 1001, 1010, 1100. On obtient ainsi les mots de longueur *N*, écrits en binaire, avec la présence de *P* zéros.

Le premier mot est 00..011..1, formé de *P* zéros suivis de *N*-*P* uns. Il reste à déterminer la règle de passage d'un mot au suivant. Le pivot est le premier 0 à partir de la droite que l'on peut transformer en 1. Cela sous-entend que ce 0 a un 1 au moins à sa droite. Ce 0 ne peut pas être suivi d'autres 0 avant de tomber sur le 1, car sinon on aurait choisi le dernier de ces 0 comme pivot. Ainsi, il suffit de chercher le premier bloc 01 à partir de la droite. On va le transformer en 10. Tout ce qui est à sa gauche reste fixe. Dans ce qui est à sa droite, il n'y a aucun bloc 01. Cela signifie que cette partie droite est formée d'un bloc de 1 suivi d'un bloc de 0, l'un de ces blocs pouvant être vide. Il suffit de mettre à l'envers cette partie droite pour avoir le plus petit mot dans l'ordre alphabétique. Par exemple la combinaison 001**01**1100 est suivie de 001**10**0011. Le processus de passage d'une combinaison à la suivante se termine lorsque le pivot que l'on trouve est en position -1.

D'où le programme:

```
for(i=0; i<N-P;i++) a[N-1-i]=1;   /*  on place les 1 … la fin, le reste étant déjà à 0  */
compteurc=1;    /* cette variable va compter le nombre des combinaisons */
afficher la première combinaison
for(;;)
  {
  compteurc++; i =N-1;
  while (a[i]!=1 || a[i-1]!=0 && i>0) i --;
  pospivot = i-1;     if (pospivot==-1) break;       a[pospivot]=1; a[pospivot+1]=0;
  gauche=pospivot+2; droite=N-1 ;
  while(gauche<droite) { aux=a[g] ; a[g]=a[d] ; a[d]=aux ; g++ ; d-- ; }
  afficher la combinaison       /* parcourir le tableau a[], et si a[i] vaut 0 afficher i  */
  }
}
```

6. Exercice complémentaire

Enumérer les mots de longueur N à base de trois lettres 0, 1, 2 (cela pourrait aussi bien être a, b, c), tels que deux lettres successives de ces mots ne sont jamais identiques.

Ces mots sont, dans l'ordre alphabétique :
pour *N*=1 : 0, 1, 2.

pour *N*=2 : 01, 02, 10, 12, 20, 21.
Pour *N*=3 : 010, 012, 020, 021, 101, 102, 120, 121, 201, 202, 210, 212.
Pour *N*=4 : 0101, 0102, 0120, 0121, 0201, 0202, 0210, 0212, 1010, 1012, 1020, 1021, 1201, 1202, 1210, 1212, 2010, etc. (on a souligné le pivot, qui sera utilisé dans ce qui suit).

a) *A partir d'un mot de longueur K, combien y a-t-il de mots de longueur K+1 obtenus en rajoutant une lettre à la fin. En déduire la formule qui donne le nombre de mots de longueur N.*

Derrière la dernière lettre d'un mot de longueur *K*, on a le choix entre deux lettres pour former un mot de longueur *K*+1. Le nombre de mots double lors de ce passage. Si l'on appelle $u(K)$ le nombre de mots de longueur *K*, on a la relation de récurrence $u(K+1) = 2\,u(K)$ avec au départ $u(1) = 3$, d'où

$$\boldsymbol{u(N) = 3 \cdot 2^{N-1}}.$$

b) *Le pivot est la lettre la plus loin possible, derrière un bloc fixe, qui doit changer pour passer au mot suivant. Indiquer comment obtenir le pivot, en partant de la droite.*

Le pivot ne peut pas être le chiffre 2. Il peut être soit le chiffre 0, soit le chiffre 1 pourvu que ce 1 soit précédé d'un 0 car s'il était précédé de 2 on ne pourrait pas faire passer le pivot à 2.

c) *Par quelle lettre doit-on remplacer le pivot ?*

Si le pivot est 0, et que devant lui se trouve un 2, il passe à 1, et si devant lui se trouve un 1, il passe à 2. Si le pivot est 1 (il est alors précédé de 0) il passe à 2.

d) *Quel est le premier mot de longueur N donnée ?*

Il s'agit du mot 0101....

e) *Faire le programme qui donne tous les mots de longueur N.*

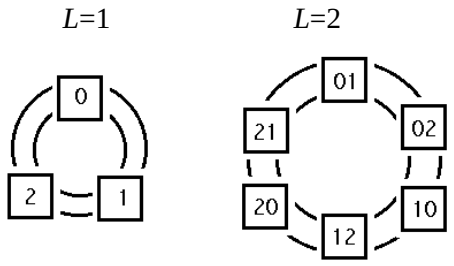
On fabrique le premier mot, que l'on place dans le tableau a[N] où vont se trouver à tour de rôle tous les mots.

```
for(i=0;i<N;i++) if (i%2==0) a[i]=0; else a[i]=1;
compteur=0;     chaque mot est numéroté par cette variable compteur
Puis on fait la grande boucle du programme qui permet de passer d'un mot au suivant.

for(;;)
  {
     affichage du mot
    compteur++;  printf("\n%d: ",compteur);  for(i=0; i<N;i++) printf("%d ",a[i]);
   /* Recherche de la position du pivot à partir de la droite*/
  i=N-1;  while(i>=1 &&  a[i]= =2 || (a[i]= =1 && a[i-1]==2)) i - -;
  pospivot=i;
 /* Si la position du pivot tombe à 0, et que la lettre en position 0 est un 2, on a obtenu le
  dernier mot, et la boucle est arrêtée */
  if (pospivot==0 && a[i]==2) break;
  /* On augmente le pivot, en distinguant plusieurs cas, notamment celui où la position du
 pivot est tombée à 0 */
  if (pospivot==0) a[pospivot]++;
  else if (a[pospivot]==0 && a[pospivot-1]==1 ) a[pospivot]=2;
  else if (a[pospivot]==0 && a[pospivot-1]==2) a[pospivot]=1;
  else if (a[pospivot]==1) a[pospivot]=2;
 /* Derrière le pivot, on place une alternance de 0 et de 1*/
  k=0; for(i=pospivot+1;i<N;i++)  {a[i]=k%2; k++; }
  }
```

Autre méthode, utilisant des listes chaînées, au lieu de tableaux

La succession des mots de longueur donnés va être placée dans une liste circulaire doublement chaînée, chaque cellule contenant un mot. On commence par la liste des mots de longueur *L*=1, puis à partir de là on fabrique la liste des mots de longueur *L*=2, et ainsi de suite jusqu'à la liste finale des mots de longueur *N*. Chaque cellule contient un mot, placé dans un tableau *a*[*N*] dont on n'utilise qu'une partie de longueur *L* (de la case 0 à la case *L*-1) à chaque étape. Cela donne l'évolution suivante :



On constate que pour passer d'une étape à la suivante, un mot est remplacé par deux mots, une cellule est remplacée par deux cellules. Pour cela, il faudra modifier le contenu d'une cellule, en ajoutant la dernière lettre (distinguer plusieurs cas), et insérer une nouvelle cellule après elle, en reprenant le contenu de la précédente jusqu'à une certaine case, et en ajoutant une lettre. On en déduit le programme complet, construit morceau par morceau.

```
struct mot { int a[N];  struct mot * s;  struct mot * p;  };

main()
{
L=1;    /* embryon de la liste, avec trois cellules pour les trois mots de longueur 1 */
debut = (struct mot *) (malloc(sizeof(struct mot)));
cell1 = (struct mot *)(malloc( sizeof (struct mot)));
cell2 = (struct mot *)(malloc( sizeof (struct mot)));
debut->a[0]=0; debut->s=cell1; debut->p=cell2;
cell1->a[0]=1; cell1->s=cell2; cell1->p=debut;
cell2->a[0]=2; cell2->s=debut; cell2->p=cell1;

for(L=2; L<=N; L++)  /* boucle des étapes, pour chaque longueur des mots  */
  {
  ptr=debut;     /* à chaque fois, on fait le tour de la liste en ajoutant des cellules */
  do
    {
    ptra=ptr->s;
    newcell= (struct mot *)(malloc( sizeof (struct mot)));
    for(i=0;i<L-1;i++) newcell->a[i]=ptr->a[i];
    if (ptr->a[L-2]==0) { ptr->a[L-1]=1;  newcell->a[L-1]=2; }
    else if (ptr->a[L-2]==1) { ptr->a[L-1]=0; newcell->a[L-1]=2; }
    else if (ptr->a[L-2]==2) { ptr->a[L-1]=0; newcell->a[L-1]=1;}
    newcell->s=ptra; newcell->p=ptr; ptr->s=newcell;ptra->p=newcell;
    ptr=ptra;
    }
  while (ptr!=debut);
  }
pptr=debut; n=0;
do  { n++; printf("\n%d: ",n);  for(i=0;i<L;i++) printf ("%d ",pptr->a[i]); pptr=pptr->s; }
while (pptr!=debut);
getchar();
}
```