INF233-TP2 : Énumerer les permutations et autres joyeusetés

NB : Ecrire tout dans un meme fichier (certaines fonctions seront utilisées pour plusieurs exercices). Ecrire les questions qui ne demandent pas de programme en commentaires.

Envoyer le TP à la fin avec la commande :

`mailx -s ''TP2-NOM1-NOM2'' aline.parreau@ujf-grenoble.fr < fichier.c`

Exercice 1 : Enumerer les permutations d'un ensemble à n éléments

Une permutation d'un ensemble à n éléments peut être définie comme une suite de n nombres $u_1, ..., u_n$ compris entre 1 et n où chaque élément est présent une seule fois (de manière équivalente, tous les u_i sont differents). Une permutation peut être uniquement représentée par un tableau de taille n contenant des entiers de 1 à n. Chaque élément étant présent exactement une fois dans le tableau.

Pour éviter des problèmes de mémoire en C, on pourra utiliser des tableaux de taille `NMAX`, où `NMAX` est une constante définie en début de fichier (qu'on pourra mettre égale à 15 dans un premier temps). On n'utilisera qu'une petite partie du tableau (indices de 1 à n). (Attention à bien passer la valeur n en argument de chaque fonction qui utilise des tableaux!)

Le but de l'exercice est d'énumérer toutes les permutations de l'ensemble $\{1, ..., n\}$. Nous allons construire deux solutions pour ce problème : une solution itérative et une solution récursive.

Question 1 : Combien y a-t-il de permutations de $\{1, ..., n\}$?

Solution itérative :

Pour cette solution, nous allons énumérer les permutations dans l'ordre *lexicographique* : On a $(u_1, ..., u_n) <_{lex} (v_1, ..., v_n)$ si il existe un indice $1 \leq j \leq n$ tel que :

- $u_i = v_i$ pour $i < j$,
- $u_j < v_j$.

Question 2 : Quelle est la plus petite permutation pour cet ordre? La plus grande?

Question 3 : Ecrire une fonction qui initialise le tableau avec la première permutation.

Question 4 : Donnez la permutation qui suit les permutations suivantes ($n = 5$) : $(1, 2, 3, 4, 5)$, $(1, 2, 5, 4, 3)$, $(1, 4, 2, 3, 5)$, $(1, 4, 5, 3, 2)$.

Pour passer à la permutation suivante on suit l'algorithme suivant (en êtes-vous convaincus?) :

1. Trouver le plus grand indice i tel que `tab[i]<tab[i+1]`
2. Trouver l'indice j du plus petit élément plus grand que `tab[i]` situé après i.
3. Echanger les éléments d'indice i et j
4. Trier le tableau dans l'ordre croissant, après l'indice i.

Question 5 : Ecrire une fonction `void echange(int tab[NMAX], int i, int j)` qui échange les éléments d'indice i et j du tableau donné en argument.

Question 6 : Ecrire une fonction `int minimum(int tab[NMAX], int deb, int fin, int valeur)` qui trouve l'indice du plus petit élément plus grand que `valeur`, compris entre les indices `deb` et `fin`. (On pourra supposer que `tab[deb]>valeur`).

Question 7 : Ecrire une fonction qui prend un tableau en entrée, et qui le modifie pour donner la permutation suivante. On suivra l'algorithme précédent.

Rappel : Pour la dernière étape, on fournit la fonction qui trie le tableau entre les indices `deb` et n :

```
void tri(int tab[NMAX], int deb, int n)
   {int i,min;
    for (i=deb; i<n; i++){
       /*on cherche la plus petite valeur entre i et n*/
```

```
      min=minimum(tab,i,n,0);
      /*on met le minimum en position i*/
      echange(tab,min,i);
  }}
```

La fonction finale a la structure suivante :

1. On part de la plus petite permutation
2. On affiche la permutation
3. On passe à la permutation suivante
4. On répète les étape 2 et 3 jusqu'à atteindre la dernière permutation.

Question 8 : Ecrire la fonction qui énumère toutes les permutations de $\{1, ..., n\}$. La tester.

Solution récursive : La solution utilisée précédement est bien laborieuse à mettre en oeuvre. Nous allons voir ici une solution récursive, plus efficace à écrire mais dont l'on contrôle moins bien le comportement...

Le but de cette partie est d'écrire une fonction `int permutation(int tab[NMAX], int deb, int n)` qui va énumérer toutes les permutations des éléments de 1 à n du tableau `tab` pour lesquelles les éléments entre 1 et `deb -1` sont fixés. (On permute les éléments après l'indice `deb`).

Question 9 : Supposer que la fonction précédente est écrite. Quelle commande permet de lister toutes les permutations de 1 à n ?

Question 10 : Donner les permutations que doit afficher la commande `permutation(tableau,3,5)` lorsque `tableau` est le tableau `[1,4,2,3,5]` (On n'a donné ici que les éléments utiles du tableau).

Question 11 : (*Terminaison*) Lorsque `deb=n`, la fonction `permutation` affiche le tableau. (Il n'y a qu'une seule permutation possible). Ecrire une fonction `afficher` qui affiche les éléments de 1 à n du tableau.

L'idee du programme récursif est de mettre en première position toutes les valeurs possibles et d'énumérer les permutations sur ce qu'il reste. On donne la structure de la fonction `permutation` :

```
Si deb=n, afficher le tableau
Sinon :
   Pour i allant de deb à n, faire:
     1) Echanger les elements d'indice i et deb
     2) Appeler récursivement permutation avec tab, deb+1, n
     3) Echanger les elements d'indice i et deb
```

Question 12 : Comprendre et justifier cet algorithme. A quoi sert l'étape 3 ? Que garantit la fonction sur le tableau à la fin ? Montrer qu'après un appel à la fonction `permutation`, le tableau donné en entrée est **dans le même état qu'avant l'appel**. C'est ce qu'on appelle un *invariant*.

Question 13 : Ecrire la fonction `permutation` et la tester pour écrire toutes les permutations de 1 à n avec différentes valeur de n.

Exercice 2 : Énumerer les sous-ensembles d'un ensemble à n éléments

Question 1 : En suivant la méthode de l'exercice précédent, écrire des fonctions itératives et récursives qui énumèrent tous les sous-ensembles de $\{1, ..., n\}$.

(Indice : on représentera un sous ensemble par un tableau de taille n contenant des 0 et des 1. La case i vaut 1 si et seulement si l'élément est dans l'ensemble.)

Question 2 : Modifier le fonction recursive précédente pour n'afficher que les sous-ensembles de taille k.

Exercice 3 : Pour s'entrainer...

Faire de même avec les mots, les sous-ensembles ordonnés,...