

CIR2 - Algorithmique Avancée

Sujet TD&P n°1 - Séance n°1

Exercice 1

Algo. Un peu de math.

i. Soit T un simple tableau de n entiers. Ecrire l'algorithme qui recherche une monotonie croissante maximale (i.e., qui recherche la suite de valeurs croissantes¹ la plus longue) dans un tableau à 1 dimension, et renvoie l'indice de départ et la longueur de cette suite avec les variables *deb* et *long*. On minimisera le nombre de cases visitées.

Exercice 2

Algo & C++. Un peu de combinatoire.

i. Algo. Objectif. Écrire deux algorithmes *Suivant* et *Generateur*.

Nous allons travailler sur la génération de mots. Les lettres disponibles sont tirées d'un mot donné en entrée de l'algorithme. Il faut faire attention à ce que chaque lettre du mot en entrée n'apparaît qu'une seule fois (comme dans le mot "arc" et non pas comme dans le mot "flèche" où le 'e' apparaît 2 fois).

On établit un ordre des lettres selon celui de notre alphabet ($a < b < \dots < z$). Ainsi, à partir du mot "arc", nous disposons d'un court alphabet ordonné tel que $a < c < r$. Tous les mots générés auront la même taille que le mot en entrée (soit une taille de 3 pour le mot "arc"), on peut les lister dans l'ordre :

| | | |
|------------------|----------------|----------------|
| - aaa, aac, aar, | aca, acc, acr, | ara, arc, arr, |
| - caa, cac, car, | cca, ccc, ccr, | cra, crc, crr, |
| - raa, rac, rar, | rca, rcc, rcr, | rra, rrc, rrr. |

Deux algorithmes sont à écrire : *Suivant* prend un mot en paramètre et renvoie le mot suivant dans l'ordre lexicographique. Par exemple *Suivant*("aaa") renvoie "aac", *Suivant*("aar") renvoie "aca" ou encore *Suivant*("crr") renvoie "raa". On peut penser aux reports des retenues d'une addition en imaginant *Suivant*(099) renvoyant 100 à l'image de *Suivant*("arr") renvoie "caa".

Generateur va générer et afficher l'ensemble des mots.

Vous disposez des constantes et fonctions suivantes :

- les constantes 'premiereLettre' et 'derniereLettre' (e.g., 'a' et 'r'),
- la constante k correspond au nombre de lettres de chaque mot (soit 3 pour notre exemple),

¹ Non strictement.

CIR2 - Algorithmique Avancée

- la fonction *succ*(lettre) renvoie la lettre suivante de l'alphabet ordonné (e.g., si 'a', 'c' et 'r' forment l'alphabet, *succ*('a') retourne 'c', *succ*('c') retourne 'r', mais *succ*('r') retourne une erreur : il faut alors faire en sorte de ne pas en arriver là). A vous de gérer :

- la ou les "retenues",
- les constantes "premierMot" et "dernierMot" de k lettres (e.g., aaa et rrr),
- la fonction *AfficherMot*(mot) affiche "mot" donné en paramètre.

L'accès aux lettres d'un mot se fait à la manière d'un tableau classique disposant de l'opérateur '['].

ii. C++². Objectif. Développer un code C++ qui va générer tous les mots à partir des lettres de l'alphabet issues du mot "algorithm" et qui font la même taille que ce dernier.

Les lettres d'un mot sauvegardées dans un `std::vector` de `char` peuvent être triées dans l'ordre lexicographique avec la méthode `sort` :

- <https://en.cppreference.com/w/cpp/algorithm/sort>. Vous obtiendrez ce type d'instruction :
 - `sort(vecLettresDuMotAlgo.begin(), vecLettresDuMotAlgo.end());`

`sort` peut être utilisée en l'important avec l'instruction `#include <algorithm>`. Implémenter pour cela les algorithmes de la question (i.) et afficher tous les mots obtenus avec la méthode *Generateur*. Tester sur d'autres mots, plus petits que "Algorithm", afin de vérifier si le nombre de mots générés est bien celui attendu.

Bonus.

a. Comprendre l'importance des algorithmes combinatoires par cet exercice. Imaginer que chaque mot ne peut avoir au maximum qu'une seule occurrence de chaque lettre ("aaa" est interdit contrairement à "acr" qui devient le premier mot et "rca" devient le dernier). Si le problème du voyageur de commerce est compris³, développer un programme qui va générer l'ensemble des tournées du voyageur de commerce en se basant sur les algorithmes de cet exercice. L'idée est que chaque ville à traverser par le voyageur peut correspondre à une seule lettre.

b. Au niveau de la STL et des algorithmes, essayez d'utiliser d'autres méthode de `<algorithm>` :

- `next_permutation`(https://en.cppreference.com/w/cpp/algorithm/next_permutation),
ou
- `prev_permutation`(https://en.cppreference.com/w/cpp/algorithm/prev_permutation).

c. Il est possible d'aller plus loin en considérant des situations réelles en ajoutant à votre programme des distances pour chaque couple de villes afin de cibler la (ou 'une des') tournée du voyageur la (ou 'les') plus courte.

Bon courage.

² Si vous n'avez pas encore de compilateur c++ d'installé, vous pouvez utiliser un service web tel que : https://www.onlinegdb.com/online_c++_compiler.

³ https://fr.wikipedia.org/wiki/Probl%C3%A8me_du_voyageur_de_commerce