TP Option Info MP/MP* : Révisions avant les concours

Les exercices proposés sont extraits de sujets de concours et constituent des exercices classiques à savoir refaire rapidement. Vous pouvez aussi faire les exercices proposés dans la feuille "Conseils pour les concours".

Fonctions sur les listes

1. Programmer en Caml une fonction `longueur : 'a list -> int` qui prend en entrée une liste et qui renvoie la longueur de cette liste. Quelle est la complexité de cette fonction, en terme du nombre n d'éléments de la liste ?

2. Écrire une fonction `appartient` d'appartenance d'un élément à une liste.

3. Écrire une fonction `supprime` de suppression de toutes les occurrences d'un élément dans une liste.

4. Écrire une fonction `ajoute` d'ajout d'un élément dans une liste sans redondance (si l'élément appartient déjà à la liste, on renvoie la liste sans la modifier).

5. Coder une fonction `union` de signature `'a list -> 'a list -> 'a list` telle que `union l1 l2`, où `l1` et `l2` sont deux listes d'éléments sans doublon dans un ordre arbitraire, renvoie une liste sans doublon contenant l'union des éléments des deux listes, dans un ordre arbitraire. Donner la complexité de cette fonction.

6. En utilisant la fonction `union`, coder une fonction `fusion` de signature `'a list list -> 'a list` telle que `fusion l`, où `l` est une liste de listes d'éléments, chacune de ces listes étant sans doublon, renvoie une liste de tous les éléments contenus dans au moins une des listes de la liste l, sans doublon et dans un ordre arbitraire. Donner sa complexité.

7. Coder fonction `produit` de signature `'a list -> 'b list -> ('a * 'b) list`, telle que `produit l1 l2` renvoie une liste de tous les couples (x, y) avec x un élément de `l1` et y un élément de `l2`. On supposera les listes `l1` et `l2` sans doublon. La liste résultante doit avoir pour longueur le produit des longueurs des deux listes. Donner la complexité de cette fonction.

Tri par insertion

8. Écrire une fonction `insertion x q` de signature `'a -> 'a list -> 'a list` prenant en entrée un élément `x` et une liste `q` triée dans l'ordre croissant, et renvoyant une liste triée dans l'ordre croissant, constituée des éléments de `q` et `x`.

9. En déduire une fonction `tri_insertion q` de signature `'a list -> 'a list` permettant de trier une liste dans l'ordre croissant.

10. Rappeler la complexité de ce tri dans le pire et le meilleur cas. Que peut-on dire de la complexité si dans la liste `q`, tous les éléments excepté peut-être un sont dans l'ordre croissant ?

11. Écrire une fonction `mem1 x q` de signature `'a -> ('a * 'b) list -> bool` testant l'appartenance d'un couple dont le premier élément est `x` dans la liste `q`.

12. Écrire une fonction `assoc x q` de signature `'a -> ('a * 'b) list -> 'b` renvoyant, s'il existe, l'élément `y` du premier couple `(x,y)` appartenant à la liste `q`.

13. Écrire une fonction `plus_petit_absent` de signature `int list -> int` telle que l'appel à `plus_petit_absent l` renvoie le plus petit entier naturel non présent dans `l`.

14. Coder une fonction `aplatir` de signature `('a * 'a list) list -> 'a list`, telle que, si `liste` est une liste de couples $[(x_1, l_{x1}); ...; (x_n, l_{xn})]$, où chaque x_i est un élément de type `'a`, et l_{xi} une liste d'éléments de type `'a` de la forme $[y_{i1}; ...; y_{ik_i}]$, `aplatir liste` est une liste d'éléments de type `'a` :

$$[x_1; y_{11}; ...; y_{1k_1}; x_2; y_{21}; ...; y_{2k_2}; ...; x_n; y_{n1}; ...; y_{nk_n}].$$

15. Coder une fonction `tri_fusion` de signature `('a * 'b) list -> ('a * 'b) list` triant une liste de couples (x, y) par ordre décroissant de la valeur de la seconde composante y de chaque couple. Quelle est la complexité de cet algorithme ?

16. Écrire une fonction récursive `circulaire` de signature `'a list -> 'a list` qui réalise une permutation circulaire des éléments (en décalant vers la droite).

Fonctions sur les tableaux/vecteurs

1. Écrire une fonction `dichotomie a t` de signature `int -> int array -> int` telle que si `t` est un tableau d'entiers strictement croissants et `a` un élément supérieur ou égal au premier élément du tableau et strictement inférieur au dernier, la fonction renvoie l'unique indice i tel que `t.(i)` $\leqslant$ `a` $<$ `t.(i+1)`. La fonction doit avoir une complexité logarithmique en la taille du tableau.

Tri rapide

2. Écrire une fonction `echange` réalisant l'échange de deux éléments d'un tableau.

3. Écrire une fonction `separation` prenant en entrée un vecteur `v` et deux indices `i1` et `i2` ayant pour fonction de séparer le sous-tableau `v[i1+1..i2]` selon le pivot `p = v(i1)`, et retournant l'indice du tableau correspondant à la position de `p` dans `v` après séparation. La fonction devra modifier directement le tableau (tri en place) et ne pas en créer un nouveau.

4. Écrire une fonction `tri_rapide` réalisant le tri d'un vecteur/tableau.

5. Donner un ordre de grandeur du nombre de comparaisons effectuées lorsque le tableau est déjà trié dans l'ordre croissant (respectivement décroissant).

6. On suppose ici que n est de la forme 2^k et que lors d'une exécution du tri rapide, chaque séparation du tableau a coupé le tableau en deux parts égales. Donner la complexité dans ce cas.

7. On ne suppose plus, ici, que n est de la forme 2^k. Montrer que, si on suppose que la complexité est croissante, alors elle est en $O(n \log(n))$.

Fonctions sur les arbres binaires

On représente les arbres binaires par le type:
`type arbre = Vide | Noeud of arbre*int*arbre ;;`

1. Écrire une fonction calculant la hauteur d'un arbre binaire, avec la convention que l'arbre vide a pour hauteur -1.
2. Écrire une fonction renvoyant le nombre total de nœuds d'un arbre binaire.
3. Écrire une fonction renvoyant le nombre de feuilles d'un arbre binaire.
4. On travaille maintenant avec des arbres binaires de recherche (ABR). Écrire une fonction qui vérifie si un arbre est bien un ABR (et renvoie donc un booléen).
5. Écrire des fonctions de recherche du minimum et du maximum dans un ABR.
6. Écrire une fonctions d'insertion (aux feuilles) d'un élément dans un ABR.

Fonctions sur les automates

On représente les automates déterministes complets par le type:
`type automate = { n: int ; i : int ; t : int list ; gamma : int -> char -> int } ;;`

où n est le nombre d'états (numérotés de 0 à $n-1$), i l'état initial, t la liste des états finaux et *gamma* est la fonction de transition.

1. Écrire une fonction `delta_etoile` de signature `automate -> int -> string -> int` qui, prenant en entrée un automate `a`, un état `i` et un mot `m`, renvoie l'état atteint par la machine `a` en partant de l'état `i` et en lisant le mot `m`.
2. Écrire une fonction `est_acceptant` de signature `automate -> string -> bool` telle que `est_acceptant a m` indique si le mot `m` est reconnu par l'automate `a`.

Fonctions sur les graphes

1. On représente un graphe par des listes d'adjacence (un graphe est donc de type `int list array`). Écrire une fonction `est_clique` de signature `int list array -> int list -> bool` telle que `est_clique aretes xs` renvoie `true` si et seulement si la liste `xs` est une liste d'indices de sommets formant une clique (c'est-à-dire qu'ils sont tous reliés entre eux) dans le graphe décrit par `aretes`.
2. Avec les mêmes conventions que précédemment, écrire une fonction `voisins_inferieurs` de signature `int list array -> int -> int list` telle que `voisins_inferieurs aretes x` renvoie la liste des voisins du sommet d'indice `x` dont l'indice est strictement inférieur à `x`.

Fonctions sur les chaînes de caractères

1. Programmer en Caml une fonction `est_present` de signature `string -> string -> int -> bool` prenant en entrée deux chaînes de caractères `s` et `t` et un entier `s` et testant si `s` figure dans `t` à la position `s`. Quelle est la complexité de cette fonction, en terme des longueurs k et n de `s` et `t` ?

2. À l'aide de la fonction `est_present`, programmer une fonction `recherche_naive` de signature `string -> string -> int list`, telle que si `s` est un motif et `t` une chaîne de caractères, `recherche_naive s t` renvoie la liste des indices de début des occurrences de `s` dans `t`.

3. Quelle est la complexité de la fonction `recherche_naive`, en terme des longueurs k et n de ses deux paramètres `s` et `t` ?

Matrices booléennes

Une matrice booléenne est une matrice dont les coefficients prennent uniquement les valeurs faux et vrai (`false` et `true` en langage de programmation). Le produit de matrices booléennes s'obtient selon la formule habituelle en prenant comme somme de deux valeurs booléennes le « ou logique » et comme produit de deux valeurs booléennes le « et logique ».

Écrire en Caml une fonction nommée `mult` telle que, si `A` et `B` sont deux matrices carrées booléennes de même taille (de type `bool array array`) alors `mult A B` renvoie une matrice codant leur produit.