TP Option Info MP/MP* : Révisions de MPSI

Listes et récursivité

Écrire des fonctions réalisant les opérations suivantes sur les listes :

1. Longueur d'une liste.
2. Appartenance d'un élément `x` à une liste `l`.
3. Déterminer le ième élément d'une liste.
4. Concaténation de deux listes.
5. "Aplatir" une liste de listes.
6. Appliquer une fonction `f` à tous les éléments d'une liste `l`.
7. Retourner une liste : essayer de le faire en temps linéaire.

Vecteurs et programmation impérative

1. Écrire une fonction calculant le n-ième terme de la suite de Fibonacci en utilisant un vecteur où seront stockées les valeurs successives. Quelle est sa complexité temporelle ? spatiale ? La réécrire en utilisant des références.
2. Écrire une fonction `appartient x a` d'appartenance d'un élément `x` à un vecteur `a`. Quelle est sa complexité ? Peut-on faire mieux si l'on suppose le tableau trié ? Programmer une version efficace dans ce cas.
3. Écrire une fonction `concatene a1 a2` qui concatène deux tableaux unidimensionnels (il faudra créer un nouveau tableau). L'adapter ensuite aux matrices ayant le même nombre de lignes.
4. Écrire une fonction `map_array f a` qui, sur le modèle de la fonction `map` pour les listes, renvoie un nouveau vecteur contenant les images des éléments de `a` par la fonction `f`.

Diviser pour régner et programmation dynamique

1. **Tri fusion:**
   - Écrire une fonction qui sépare une liste en deux.
   - Écrire une fonction qui réalise la fusion de deux listes triées en une seule liste triée.
   - Écrire la fonction de tri fusion.

   En bonus : refaire les autres algorithmes de tris vus l'année dernière.

2. **Sac à dos:** Étant donnés n objets de valeurs $c_1, \ldots, c_n$ et de volume $v_1, \ldots, v_n$ et un sac à dos de volume W_{max}, on souhaite remplir le sac en maximisant la valeur $\sum c_i$ tout en respectant la contrainte $\sum v_i \leqslant W_{max}$.

   On note $f(i, V)$ la valeur maximale qu'il est possible d'atteindre avec les i premiers objets et le volume maximal V. On cherche $f(n, W_{max})$. On rappelle la formule de récurrence:

$$f(i,V) = \begin{cases} \max(c_i + f(i-1, V - v_i), f(i-1, V)) \text{ si } v_i \leqslant V \\ f(i-1, V) \text{ sinon} \end{cases}$$

   En utilisant un tableau bi-dimensionnel de taille $(n+1) \times (W_{max}+1)$ destiné à contenir les valeurs de $f(i, w)$ pour $0 \leqslant i \leqslant n$ et $0 \leqslant w \leqslant W_{max}$, programmer la résolution de ce problème. On écrira une fonction `sacados c v wmax` prenant en entrée deux vecteurs `c` et `v` contenant les valeurs $c_1, \ldots c_n$ et les volumes $v_1, \ldots, v_n$, ainsi que le poids maximal du sac `wmax`, et qui renverra $f(n, W_{max})$ après avoir rempli le tableau des $f(n, i)$.

Structures de données et types

On considère la définition récursive du type `'a arbre`:

```
type 'a arbre = Nil | Noeud of 'a arbre * 'a * 'a arbre ;;
```

Écrire des fonctions réalisant les opérations suivantes:
- Déterminer la hauteur d'un arbre binaire.
- Déterminer le nombre total de nœuds d'un arbre binaire.
- Déterminer le nombre de feuilles d'un arbre binaire.
- Déterminer le nombre de nœuds internes d'un arbre binaire.
- Ajouter un élément à la fin de la branche la plus à gauche (renverra un nouvel arbre).