Plus longue sous-liste commune – Tris

Samuel Mimram

4 février 2006

1 Plus longue sous-liste commune

Soit $l = [x_0; \ldots; x_{n-1}]$ une liste. Une liste l' est sous-liste de l si et seulement si il existe un entier $k \leq n$ et une fonction σ strictement croissante de $[\![0, k-1]\!]$ dans $[\![0, n-1]\!]$ telle que $l' = [x_{\sigma(0)}; \ldots; x_{\sigma(k-1)}]$: l' est obtenue à partir de l en supprimant certains éléments et en conservant l'ordre des éléments restants. Par exemple `[2 ; 2 ; 1 ; 3]` est une sous-liste de `[3 ; 2 ; 1 ; 2 ; 1 ; 5 ; 3]`.

Le but de cet exercice est de déterminer la longueur d'une plus longue sous-liste commune à deux listes (i.e. qui est sous-liste des deux listes à la fois). Pouquoi cette longueur est-elle toujours bien définie ?

Cette longueur induit une « distance » entre deux mots et pourrait par exemple être utilisée par des correcteurs orthographiques pour déterminer si deux mots se ressemblent ou en bio-informatique pour mesurer la similarité entre deux séquences de bases.

On rappelle que la concaténation de deux listes se fait à l'aide de l'opérateur `@`.

1.1 Première approche

▶ Écrivez une fonction `sublist` de type `'a list -> 'a list -> bool` qui détermine si une liste est sous-liste d'une autre.

▶ Trouvez une formule de récurrence caractérisant l'ensemble des sous-listes d'une liste donnée. Quel est le cardinal de l'ensemble des sous-liste d'une liste l en fonction de la longueur de l ?

Déduisez-en une fonction `sublists` de type `'a list -> 'a list list` qui renvoie la liste des sous-listes d'une liste. Vous pourrez utiliser la fonction `List.map` de type `('a -> 'b) -> 'a list -> 'b list` qui à une fonction f et à une liste $[x_0; \ldots; x_{n-1}]$ associe la liste $[f(x_0); \ldots; f(x_{n-1})]$.

▶ Écrivez une fonction `mem` de type `'a -> 'a list -> bool` qui détermine si un élément fait partie d'une liste (cette fonction existe en réalité déjà dans la librairie standard de caml et s'appelle `List.mem`).

▶ Écrivez une fonction récursive `inter` de type `'a list -> 'a list -> 'a list` qui renvoie la liste des éléments communs à deux listes.

▶ La librairie standard de caml comporte une fonction `List.fold_left` dont la spécification est la suivante

```
val fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a

List.fold_left f a [b1; ...; bn] is f (... (f (f a b1) b2) ...) bn
```

Reprogrammez la fonction `inter` en une ligne en utilisant `List.fold_left`.

▶ En utilisant les fonctions `List.fold_left` et `max`, écrivez une fonction `max_len` de type `'a list list -> int` qui renvoie la longueur maximale d'une liste dans une liste de listes.

▶ Déduisez-en une fonction `plslc` de type `'a list -> 'a list -> int` qui calcule la longueur d'un plus longue sous-liste commune à deux listes.

1.2 Équation de récurrence

Soient $l = [x_0; \ldots; x_p]$ et $l' = [y_0; \ldots; y_p]$ des listes. On note $\lambda(i, j)$ la longueur de la plus longue sous-suite commune à $l = [x_0; \ldots; x_{i-1}]$ et $l' = [y_0; \ldots; y_{j-1}]$. Montrez que $\lambda(i, j)$ vérifie la relation de récurrence suivante :

$$\lambda(i, j) = \max(\lambda(i-1, j-1) + \delta(x_{i-1}, y_{j-1}), \lambda(i, j-1), \lambda(i-1, j))$$

où $\delta(i, j)$ vaut 1 si $x_i = x_j$ et 0 sinon.

► Implémentez une fonction `plslc2` de type `'a list -> 'a list -> int` qui calcule la longueur d'une plus longue sous-suite commune à deux listes en utilisant cette équation. En quoi cette version est-elle meilleure que la précédente ?

1.3 Résolution de l'équation de récurrence par programmation dynamique

Le temps mis par la fonction précédente est exponentiel en la taille des deux listes en entrée (pourquoi ?).

Nous allons maintenant calculer le plus long sous-tableau commun (on utilise des tableaux au lieu des listes pour des questions d'efficacité et de simplicité de la mise en pratique mais la définition est bien évidemment similaire).

► En stockant les $\lambda(i, j)$ dans une matrice, utilisez la formule de récurrence donnée précédemment pour calculer les $\lambda(i, j)$ en ne calculant qu'une seule fois chaque élément de la matrice.

On rappelle qu'on peut créer une matrice à l'aide de la fonction `make_matrix` de type `int -> int -> 'a -> 'a array array` qui prend en arguments la largeur, la hateur et la valeur initiale des éléments de la matrice. Si `m` est une matrice `m.(i).(j)` permet d'accéder à l'élément d'indice (i, j). Les matrices sont représentées en caml par des tableaux de tableaux, les indices commencent donc aussi à 0.

► Déduisez-en une fonction `plstc` de type `'a array -> 'a array -> int` qui calcule la longueur du plus long sous-tableau commun à deux tableaux. Quelle est la complexité de cette nouvelle fonction ?

► Auriez vous une idée sur comment adapter cet algorithme pour renvoyer un plus long sous-tableau commun (on ne demande pas de l'implémenter) ?

2 Quelques tris

Comme il faut l'avoir fait au moins une fois dans sa vie, je vous propose ici d'implémenter quelques tris courants.

Comme vous avez vu le tri-fusion en cours, je vous conseille de commencer par le quicksort.

2.1 Tri-fusion

► Programmez une fonction `length` de type `'a list -> int` qui renvoie la longueur d'une liste (je vous demande de la reporgrammer à titre d'exercice car elle existe dans la librairie standard de caml sous le nom `List.length`).

► Programmez un fonction `split` de type `'a list -> ('a list * 'a list)` qui prend en entrée une liste de longueur n et la « coupe » en deux en renvoyant la liste des $n/2$ premiers éléments et la liste des éléments restant.

```
# split [1; 2; 3; 4; 5];;
- : int list * int list = ([1; 2], [3; 4; 5])
```

Vous pourrez utiliser une fonction récursive intermédiaire.

▶ Programmez une fonction `merge` de type `'a list -> 'a list -> 'a list` qui prend en arguments deux listes d'entiers supposées triées et revoie une liste triée contenant les éléments des deux listes données en arguments.

▶ Implémentez une fonction `mergesort` de type `'a list -> 'a list` qui prend une liste `l` et renvoie une listes triée contenant les éléments de `l`. L'algorithme est le suivant :
- si la liste `l` contient un ou zéro élément alors elle est déjà triée, on peut donc la renvoyer directement ;
- sinon utilisez la fonction `split` sur `l` pour obtenir deux sous-listes `l1` et `l2`, triez ces listes en faisant un appel récursif à votre fonction et fusionnez-les en utilisant `merge`.

Vérifiez que votre fonction marche sur un exemple.

Quelle est sa complexité ?

2.2 Quicksort

L'algorithme précédent n'a pas une bonne *complexité en espace* (que cela signifie-t-il ? pourquoi ?). L'algortihme quicksort que nous allons présenter a lui l'avantage de pouvoir se faire *en place*, c'est-à-dire sans avoir à créer de listes (ou de tableaux) intermédiaires. Son inconvénient est que sa complexité dans le pire des cas est en $O(n^2)$ mais sa complexité moyenne reste en $O(n \cdot \ln(n))$.

▶ Programmez une fonction `swap` de type `'a array -> int -> int -> unit` telle que `swap arr i j` échange les élements d'indice `i` et `j` dans le tableau `arr`.

▶ Programmez une fonction `partition` de type `'a array -> int -> int -> int` telle que `partition arr start end` qui partitionne le sous-tableau entre les indices `start` et `end` (inclus) en prenant l'élément `arr.(start)` comme pivot et renvoie le nouvel indice du pivot dans le tableau.

Plus exactement, appelons p la valeur de `arr.(start)` (le pivot). Vous devrez échanger certaines cases du tableau, en utilisant `swap`, dont les indices sont compris entre `start` et `end` et renvoyer un entier k de sorte que :

$$\forall i, \quad \texttt{start} \leq i < k \Rightarrow \texttt{arr.}(i) < p$$

et

$$\forall i, \quad k \leq i \leq \texttt{end} \Rightarrow \texttt{arr.}(i) \geq p$$

La complexité de votre fonction devra être linéaire en $\texttt{end} - \texttt{start}$. N'hésitez pas à me demander si vous avez du mal à comprendre ce que doit faire cette fonction ou comment l'implémenter.

▶ Implémentez l'algorithme quicksort. Votre fonction `quicksort` sera de type `'a array -> int -> int -> unit`. L'idée est que `quicksort arr start end` va trier les éléments du tableau `arr` dont les indices sont compris entre `start` et `end`.
- Si $\texttt{start} \geq \texttt{end} - 1$ alors il n'y a rien à faire et vous pouvez renvoyer directement le tableau `arr`.
- Sinon utilisez `partition` pour partitionner le tableau entre `start` et `end`. Si i est l'indice du pivot après le partitionnement, il faudra ensuite appeler récursivement `quicksort arr start` $i-1$ et `quicksort arr` i `end`.

Testez votre fonction sur un exemple.

Donnez un exemple de liste où la complexité est en $O(n^2)$.

2.3 Tri linéaire (!)

▶ Supposons que vous ayez à trier un tableau d'entiers qui sont tous inférieurs à un entier k. Comment feriez-vous pour faire un tri ayant une complexité linéaire en la taille du tableau à trier ? Quelle serait alors la complexité en espace ?