Devoir en temps libre n° 2

Ce devoir est à rendre pour le mardi 9 mai 2017, sous forme électronique uniquement. Les questions comportant le symbole ⊛ sont à chercher au brouillon (je vous fais confiance). Le code CAML devra être envoyé à `nicolas.pecheux@cpge.info` avant 07h59 le 9 mai 2017. Vous enverrez un unique fichier `.ml` dont le nom sera obligatoirement et exactement `<login>.ml` où `<login>` est votre nom d'utilisateur du laboratoire d'informatique (tout en minuscules avec les tirets).

Votre programme devra respecter scrupuleusement les noms et les types indiqués (ou un type plus général ou équivalent bien entendu) car il sera en partie corrigé automatiquement. Le code fourni devra compiler sans erreurs ni aucun message d'avertissement. L'encodage du fichier devra être UTF-8 et vous n'utiliserez que des caractères ASCII pour les noms de variables. Il est impératif de veiller au strict respect de ces consignes.

La clarté et la simplicité du code et l'utilisation judicieuse de commentaires seront pris en compte dans l'évaluation. On prendra bien soin de tester toutes les fonctions sur des cas triviaux, puis un peu plus complexes.

Lorsque l'on pose des hypothèses sur les arguments (par exemple qu'un argument est un entier naturel), il n'est pas nécessaire dans vos fonctions de le vérifier ni de prévoir le comportement si l'hypothèse est violée (par exemple si l'argument est négatif). On pourra cependant lever des exceptions (`failwith "Argument non valable"` par exemple) si on le souhaite.

Les trois parties sont indépendantes.

1 Des tableaux vers les listes et inversement

QUESTION 1 Écrire les fonctions `list_of_vect : 'a vect -> 'a list` et `vect_of_list : 'a list -> 'a vect` qui permettent de passer d'un tableau à une liste et inversement. On n'utilisera pas de références de listes et on aura donc recours à des fonctions récursives.

2 Tris de tableaux

Dans cette partie, on se propose d'implémenter pour les tableaux le tri insertion, le tri rapide et éventuellement le tri fusion, que nous avons déjà rencontrés pour les listes. On fera bien attention aux différents cas particuliers (tableaux vides, notamment).

Avant de commencer cette partie, vous devez chercher la partie III du DS n° 1, puis vérifier vos réponses avec le corrigé.

QUESTION 2 ⊛ Avez-vous réellement cherché la partie III du DS n° 1, puis — et uniquement après cela — confronté vos réponses avec celles du corrigé ?

QUESTION 3 Écrire une fonction `echange : 'a vect -> int -> int -> unit` tel que `echange tab i j` échange le contenu des cases d'indices `i` et `j` du tableau `tab`. On pourra supposer que les indices sont valables, mais je vous encourage plutôt à lever une erreur pour faciliter la détection de bogues par la suite.

QUESTION 4 Écrire une fonction `tri_insertion : 'a vect -> unit` qui trie un tableau en insérant les éléments à la bonne place dans la partie du tableau déjà triée, en les comparant aux autres. Le tri devra se faire sur place, c'est-à-dire sans créer un autre tableau.

QUESTION 5 Écrire une fonction `partition : 'a vect -> int -> int -> int` tel que `partition tab i j` sur un tableau $(t_k)_{0\leq k\leq n-1}$ avec $0 \leq i < j \leq n-1$ réarrange le sous-tableau $(t_k)_{i\leq k\leq j}$ suivant le pivot t_i et renvoie l'indice p du pivot dans le tableau réarrangé. Après l'appel, on doit avoir $\forall k\,(i \leq k < p \Rightarrow t_k \leq t_p) \wedge (p < k \leq j \Rightarrow t_p < t_k)$ dans le tableau modifié (par effets de bord !).

QUESTION 6 Écrire une fonction `tri_rapide_aux` de type `'a vect -> int -> int -> unit` tel que l'appel `tri_rapide_aux tab i j` sur un tableau $(t_k)_{0\leq k\leq n-1}$ avec $0 \leq i < j \leq n-1$ trie sur place le sous-tableau $(t_k)_{i\leq k\leq j}$.

QUESTION 7 En déduire la fonction `tri_rapide : 'a vect -> unit` qui trie sur place un tableau.

QUESTION 8 (*Facultatif et plus difficile*) Implémenter une fonction `tri_fusion : 'a vect -> unit` qui trie un tableau en utilisant la même idée que le tri fusion pour les listes. La division est évidente puisqu'il suffit de calculer l'indice du milieu. En revanche la fonction `fusion : 'a vect -> int -> int -> int -> unit` tel que `fusion tableau debut milieu fin` fusionne les sous-tableaux triés $tableau[debut..milieu]$ et $tableau[(milieu+1)..fin]$ dans $tableau[debut..fin]$ est plus délicate. On commencera par créer deux copies[1] des deux sous-tableaux avant de fusionner leurs éléments dans le tableau initial $tableau[debut..fin]$.

1. Ce n'est donc pas entièrement un tri sur place

3 Programmation dynamique

Le but de cette partie est d'implémenter les trois exemples d'algorithmes du chapitre sur la programmation dynamique, qu'il faut donc évidemment avoir lus et bien compris.

Pour être sûr que vous avez bien compris, nous allons ajouter une toute petite difficulté : nous prendrons la convention[a] que les indices commencent à 0 et non à 1 comme dans le polycopié. Il faudra donc faire très attention à ce point et ne pas hésiter à réécrire les algorithmes avant de les implémenter. On initialisera les tableaux avec la valeur `-1` que l'on utilisera également pour représenter ∞. Vous vérifierez bien que vos fonctions ont exactement le bon type et exactement la même sortie que dans les exemples ci-dessous.

a. Qui est la convention standard en informatique.

QUESTION 9 ⊛ Résoudre l'exercice 15.1.2.

QUESTION 10 ⊛ Résoudre l'exercice 15.1.3.

QUESTION 11 Implémenter en CAML la fonction `plus_rapide_chemin : int vect vect -> int vect vect -> int vect -> int vect -> int -> (int * int * int vect vect * int vect vect)` qui prend en entrée a, t, e, x et n, et qui renvoie f, f^*, l et l^*. L'exemple de la figure 15.2 s'écrira :

```
let a =
  let a1 = [|7; 9; 3; 4; 8; 4|] in
  let a2 = [|8; 5; 6; 4; 5; 7|] in
  [|a1; a2|]
;;
let t =
  let t1 = [|2; 3; 1; 3; 4|] in
  let t2 = [|2; 1; 2; 2; 1|] in
  [|t1; t2|]
;;
let e = [|2; 4|]
let x = [|3; 2|];;
let n = 6;;
```

On aura alors :

```
let fstar, lstar, l, f = plus_rapide_chemin a t e x n;;
f : int vect vect = [|[|9; 18; 20; 24; 32; 35|]; [|12; 16; 22; 25; 30; 37|]|]
fstar : int = 38
l : int vect vect = [|[|-1; 0; 1; 0; 0; 1|]; [|-1; 0; 1; 0; 1; 1|]|]
lstar : int = 0
```

QUESTION 12 Résoudre l'exercice 15.1.1.

QUESTION 13 Implémenter en CAML la fonction `afficher_postes : int -> int vect vect -> int -> unit` de l'exercice 15.1.1. On aura donc :

```
afficher_postes lstar l n;;
chaine 1, poste 1
chaine 2, poste 2
chaine 1, poste 3
chaine 2, poste 4
chaine 2, poste 5
chaine 1, poste 6
- : unit = ()
```

QUESTION 14 Implémenter la fonction `multiplier_matrices : int vect vect -> int vect vect -> int vect vect`.

QUESTION 15 ⊛ Résoudre l'exercice 15.2.3.

QUESTION 16 Implémenter la fonction `ordre_chaine_matrices : int vect -> int vect vect * int vect vect`. Les cases non utilisées auront pour valeur `-1`. Par exemple :

```
let m, s = ordre_chaine_matrices [|30; 35; 15; 5; 10; 20; 25|];;
m : int vect vect =
 [|[|0; 15750; 7875; 9375; 11875; 15125|];
   [|-1; 0; 2625; 4375; 7125; 10500|];
   [|-1; -1; 0; 750; 2500; 5375|];
   [|-1; -1; -1; 0; 1000; 3500|];
   [|-1; -1; -1; -1; 0; 5000|];
   [|-1; -1; -1; -1; -1; 0|]
 |]
s : int vect vect =
 [|[|-1; 0; 0; 2; 2; 2|];
   [|-1; -1; 1; 2; 2; 2|];
   [|-1; -1; -1; 2; 2; 2|];
   [|-1; -1; -1; -1; 3; 4|];
   [|-1; -1; -1; -1; -1; 4|];
   [|-1; -1; -1; -1; -1; -1|]
 |]
```

QUESTION 17 Implémenter la fonction `affichage_parenthesage_optimal : int vect vect -> unit`.

```
affichage_parenthesage_optimal s;;
((A1(A2A3))((A4A5)A6))
- : unit = ()
```

QUESTION 18 Implémenter la fonction `memorisation_chaine_matrices : int vect -> int`.

```
memorisation_chaine_matrices [|30; 35; 15; 5; 10; 20; 25|];;
- : int = 15125
```

QUESTION 19 Implémenter la fonction `plus_longue_sous_sequence : string vect -> string vect -> int vect vect * string vect vect`. La matrice `b` sera initialisée avec `"none"` et on utilisera `"up"`, `"diag"` et `"left"` pour les symboles ↑, ↖ et ←, respectivement.

```
let x = [|"a"; "b"; "c"; "b"; "d"; "a"; "b"|];;
let y = [|"b"; "d"; "c"; "a"; "b"; "a"|];;
let c, b = plus_longue_sous_sequence x y;;
c : int vect vect =
 [|[|0; 0; 0; 0; 0; 0; 0|]; [|0; 0; 0; 0; 1; 1; 1|]; [|0; 1; 1; 1; 1; 2; 2|];
   [|0; 1; 1; 2; 2; 2; 2|]; [|0; 1; 1; 2; 2; 3; 3|]; [|0; 1; 2; 2; 2; 3; 3|];
   [|0; 1; 2; 2; 3; 3; 4|]; [|0; 1; 2; 2; 3; 4; 4|]|]
b : string vect vect =
 [|[|"none"; "none"; "none"; "none"; "none"; "none"; "none"|];
   [|"none"; "up"; "up"; "up"; "diag"; "left"; "diag"|];
   [|"none"; "diag"; "left"; "left"; "up"; "diag"; "left"|];
   [|"none"; "up"; "up"; "diag"; "left"; "up"; "up"|];
   [|"none"; "diag"; "up"; "up"; "up"; "diag"; "left"|];
   [|"none"; "up"; "diag"; "up"; "up"; "up"; "up"|];
   [|"none"; "up"; "up"; "up"; "diag"; "up"; "diag"|];
   [|"none"; "diag"; "up"; "up"; "up"; "diag"; "up"|]|]
```

QUESTION 20 Implémenter la fonction `imprimer_plsc : string vect vect -> string vect -> string vect -> unit`. Les éléments seront séparés par un simple `;` sans se soucier de retours à la ligne.

```
imprimer_plsc b x y;;
b;c;b;a;- : unit = ()
```