

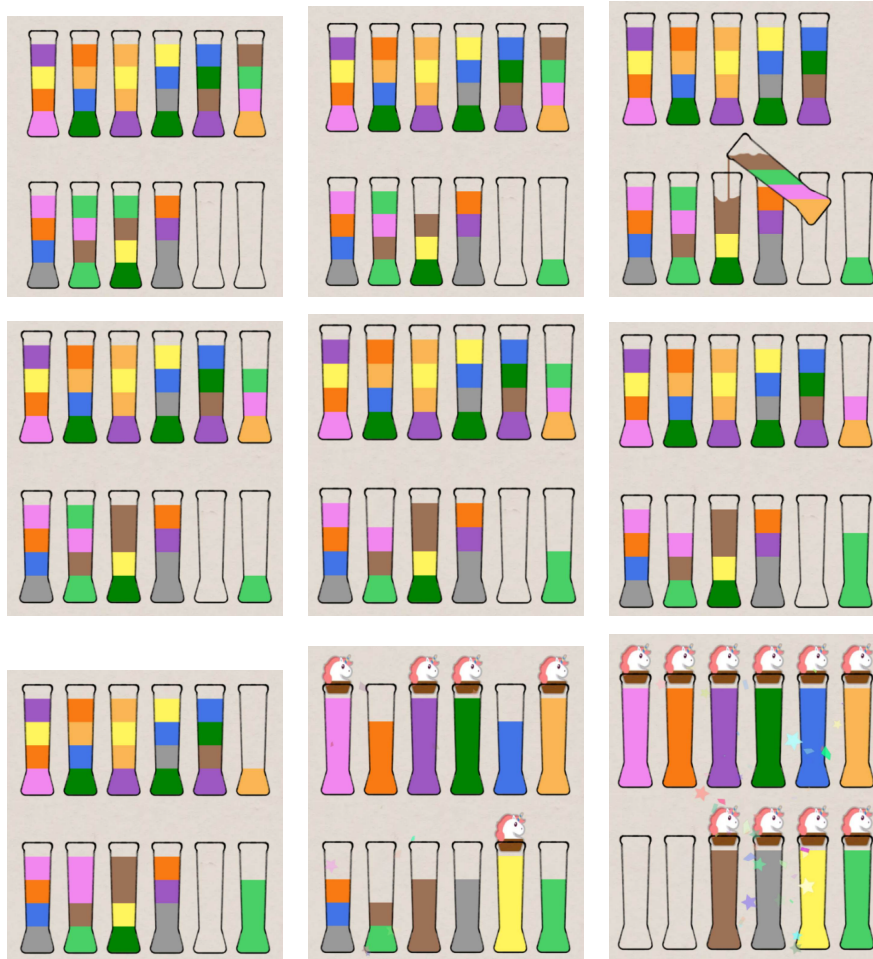
# TP - files de priorité et tri de liquides

Florian Bourse

Dans ce sujet, nous nous intéressons au jeu suivant :

Étant donné des fioles mélangées, on souhaite les trier en utilisant la seule opération suivante : on choisit deux fioles et on transvide autant que possible de la première dans la seconde.

Une contrainte supplémentaire impose que la seconde fiole soit vide ou que les deux fioles contiennent le même liquide en phase supérieure.



Généralisons ce jeu en deux problèmes : un problème de décision et un problème d'optimisation :

#### WaterSort :

**Instance :**  $n$  piles de capacité  $c$ .

**Solution :**  $V$  si il existe une suite de coup permettant d'arriver dans un état où chaque pile est soit remplie avec une seule couleur, soit vide.  $F$  sinon.

#### WaterSort\* :

**Instance :** Une instance de WaterSort dont la solution est  $V$ .

**Solution :** Une suite de coup permettant d'arriver dans un état où chaque pile est soit remplie avec une seule couleur, soit vide.

**Optimisation :** Minimiser le nombre de coups.

## 1 Résolution naïve du problème WaterSort

*Dans cette section, nous ignorerons le fichier heap.ml pour se concentrer sur le fichier watersort.ml.*

Dans un premier temps, on se propose d'implémenter un algorithme de retour-sur-trace. Pour ce faire, la brique de base est une fonction qui à partir d'un état donné de la partie, donne la liste des états que l'on peut atteindre en 1 étape. Le fichier watersort.ml fournit une telle fonction, mais en supposant l'existence des fonctions `peux_verser` et `verse` qu'il vous reste à coder.

□ 1 – Implémenter les fonctions `peux_verser` et `verse`. La première détermine si il est possible de verser une fiole  $f_1$  dans une fiole  $f_2$  et la seconde renvoie les nouvelles fioles  $f'_1$  et  $f'_2$  obtenues après cette opération en supposant qu'elle est possible.

□ 2 – Implémenter une fonction permettant de décider WaterSort par retour-sur-trace, et observer que cette technique ne permet pas de résoudre le problème quelque soit l'entrée.

*En effet, il est possible d'avoir une suite infinie d'états dont chacun est successeur du précédent.*

□ 3 – Rappeler la définition d'une structure de donnée persistante (ou immuable) et nommer un avantage de la persistance.

Remarquons qu'il nous suffit de chercher un chemin sans cycle car l'existence d'un chemin (contenant ou non un cycle) implique l'existence d'un chemin sans cycle. Remarquons également qu'il est très courant que lorsque plusieurs coups sont possibles, leur ordre ne soit pas important.

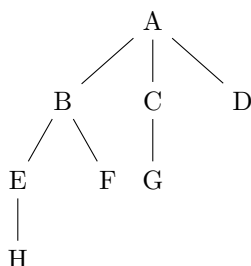
□ 4 – À l'aide d'un parcours du graphe dont les sommets sont les états possibles et les arêtes sont données par listes d'adjacence par la fonction `succ`, écrire un algorithme décidant WaterSort.

## 2 Tas binomiaux

### Arbres binomiaux

Les arbres binomiaux sont une famille d'arbres d'arité quelconque définie par induction :

- Un arbre binomial d'ordre 0 est réduit à sa racine ;
- Un arbre binomial d'ordre  $k$  est constitué d'une racine et de  $k$  sous-arbres qui sont des arbres binomiaux d'ordres respectifs  $k-1, k-2, \dots, 0$ , où les ordres des sous-arbres sont décroissant de gauche à droite.



*Exemple d'arbre binomial d'ordre 3*

- 5 – Montrer que l'on peut construire un arbre binomial d'ordre  $k$  à partir d'arbres binomiaux d'ordre  $k-1$ .
- 6 – Donner le nombre de nœuds dans un arbre binomial d'ordre  $k$ . Justifier.

On utilise les types suivants pour représenter les arbres binomiaux en OCaml :

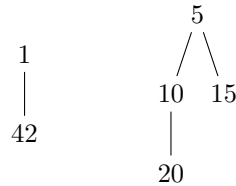


```
(* Les arbres sont de la forme Node (ordre, racine, fils) *)
type 'a arbre = Node of int * 'a * 'a foret
and 'a foret = 'a arbre list
```

- 7 – Écrire une fonction `union_arbre : 'a arbre -> 'a arbre -> 'a arbre` prenant en entrée deux arbres binomiaux d'ordre  $k$  et renvoyant un arbre binomial d'ordre  $k+1$  contenant les mêmes éléments, avec pour racine la plus petite des deux racines des arbres d'entrée.
- 8 – Écrire une fonction `est_binomial : int -> 'a arbre -> bool` qui teste si un arbre est bien un arbre binomial d'ordre  $k$ .
- 9 – Calculer la complexité de cette fonction dans le cas où l'entrée est bien un arbre binomial.

## Implémentation des tas binomiaux

Un tas binomial est une forêt d'arbres binomiaux d'ordres distincts, dont chacun des arbres vérifie la propriété de tas : tout nœud qui n'est pas une racine possède une étiquette plus grande que celle de son père. On les représentera en OCaml par le type `'a foret` défini précédemment, les arbres étant triés par ordre croissant : les arbres de plus petit ordre sont à gauche de la liste.



*Exemple de tas binomial contenant 6 éléments*

- 10 – Montrer qu'un tas binomial contenant  $n$  éléments possède au plus  $O(\log n)$  arbres.
- 11 – Implémenter l'ajout d'un arbre binomial dans un tas binomial et en déduire l'ajout d'un élément dans un tas binomial `add : 'a -> 'a foret -> 'a foret`.
- 12 – Calculer la complexité de cette fonction.
- 13 – Écrire une fonction `union_tas` prenant en entrée deux tas binomiaux et renvoyant un tas binomial contenant les éléments présents dans ces deux tas. En déduire une fonction `extract_min : 'a foret -> 'a * 'a foret` prenant en entrée un tas binomial et renvoyant l'élément minimal ainsi que le tas privé de cet élément.
- 14 – Calculer la complexité de ces opérations.
- 15 – Quelle est la structure de donnée abstraite implémentée par les tas binomiaux ?

## 3 WaterSort\*

- 16 – Donner un algorithme qui permet de déterminer en inspectant une fois chaque pile un minorant du nombre de coups requis pour arriver dans un état où chaque pile est soit remplie avec une seule couleur, soit vide.
- 17 – En déduire un algorithme permettant de résoudre WaterSort\* et l'implémenter.

## 4 Pour aller plus loin : difficulté du problème

Étudier la difficulté du problème de décision WaterSort. On pourra utiliser le résultat suivant : le problème 3-Partition défini ci-dessous est NP-difficile, même si les entrées sont données en unaire.

### 3-Partition :

**Instance :** Un ensemble de  $3m$  entiers  $a_1, \dots, a_{3m}$ , tels que  $\sum_{j=1}^{3m} a_j = mB$ .

**Solution :**  $V$  si il existe une partition de  $\llbracket 1; 3m \rrbracket$  en  $m$  triplets  $(j_{i,1}, j_{i,2}, j_{i,3})_{i=1\dots m}$  tels que  $a_{j_{i,1}} + a_{j_{i,2}} + a_{j_{i,3}} = B$ .  $F$  sinon.