Option Informatique en Spé MP et MP*
Programmation Caml : débit de l'eau, le corrigé

Question 1 • Une situation est parfaitement définie par les contenus b et c des deux «petits» bidons ; le «grand» bidon contient alors $8-b-c$ litres. $b \in [\![0,5]\!]$ et $c \in [\![0,3]\!]$, donc b peut prendre 6 valeurs différentes, et c peut en prendre 4. Le graphe possède en tout 24 sommets.

Question 2 • Il suffit de reverser (au besoin) les contenus des deux «petits» bidons dans le «grand» bidon.

Question 3 • La fonction `état_initial` nous servira à nouveau, pour la question 7 :

```
let état_initial(ma,ca,mb,cb,mc,cc) =
 let bidona = { maxi = ma ; actu = ca }
 and bidonb = { maxi = mb ; actu = cb }
 and bidonc = { maxi = mc ; actu = cc }
 in Etat (bidona,bidonb,bidonc) ;;

let graphe_initial(ma,ca,mb,cb,mc,cc) =
 { sommets = [état_initial(ma,ca,mb,cb,mc,cc)] ; arêtes = [] } ;;
```

Question 4 • La fonction `transfert` se charge de transférer q litres du bidon s vers le bidon d. Il reste à déterminer q en fonction des contenus actuels de s et d, et de la capacité de d :

```
let transfert (s,d) q =
 ({maxi = s.maxi ; actu = s.actu - q} ,
  {maxi = d.maxi ; actu = d.actu + q}) ;;

let opération_élémentaire (s,d) =
 if s.actu + d.actu > d.maxi
  then transfert (s,d) (d.maxi - d.actu)
  else transfert (s,d) s.actu ;;
```

Question 5 • L'utilisation d'un type énuméré à trois constructeurs constants autorise une écriture qui me semble assez claire.

```
type position = G | M | D ;;

let verse (x,y,z) = function
 | (G,M) -> let (x',y') = opération_élémentaire (x,y) in (x',y',z)
 | (M,D) -> let (y',z') = opération_élémentaire (y,z) in (x,y',z')
 | (D,G) -> let (z',x') = opération_élémentaire (z,x) in (x',y,z')
 | (M,G) -> let (y',x') = opération_élémentaire (y,x) in (x',y',z)
 | (D,M) -> let (z',y') = opération_élémentaire (z,y) in (x,y',z')
 | (G,D) -> let (x',z') = opération_élémentaire (x,z) in (x',y,z')
 | (_,_) -> failwith "erreur de programmation" ;;

let successeurs_état (Etat (x,y,z)) =
 map (fun s -> Etat(verse (x,y,z) s)) [(G,M);(M,D);(D,G);(M,G);(D,M);(G,D)] ;;
```

Question 6 • La définition de `filtre` est supposée connue :

```
let élection g t = let r1 = successeurs_état t
 in let r = filtre (fun x -> not(mem x g.sommets)) r1 in
   ({sommets = t::g.sommets ; arêtes = g.arêtes},r) ;;
```

Question 7 • La fonction `clore_graphe` applique l'algorithme de parcours en largeur de la composante connexe, tel qu'il est décrit dans l'énoncé.

```
let rec clore_graphe g en_attente = match en_attente with
 | [] -> g
 | t::q when mem t g.sommets -> clore_graphe g q
 | t::q -> let (g',r) = élection g t in clore_graphe g' (en_attente @ r) ;;
```

```
let construire_graphe (ma,ca,mb,cb,mc,cc) =
 let i = état_initial(ma,ca,mb,cb,mc,cc)
 and gv = {sommets = [] ; arêtes = []}
 in clore_graphe gv [i] ;;
```

Question 8 • Le programme suivant répond à la question ; `identiques` permet de savoir si deux triplets sont égaux à l'ordre près.

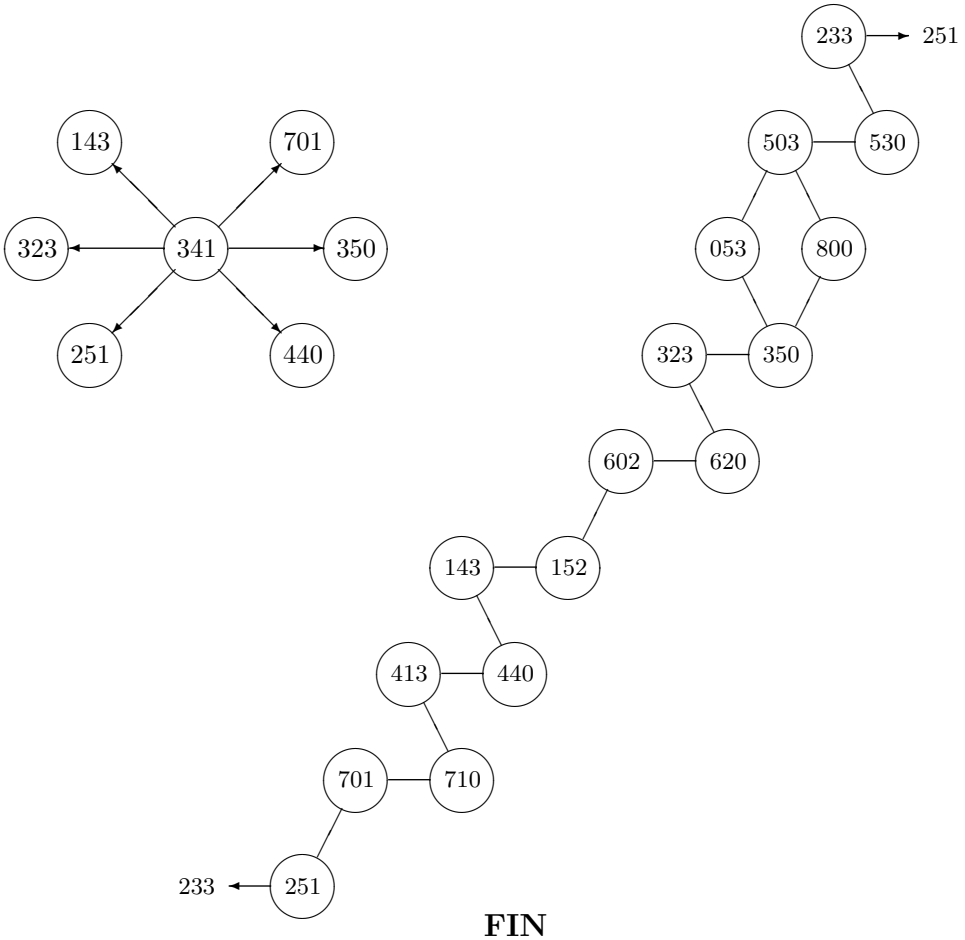
```
let identiques r s = (sort__sort (prefix <) r) = (sort__sort (prefix <) s) ;;

let projection (Etat(u,v,w)) = map (fun x -> x.actu) [u;v;w] ;;

let liste_accessibles (ma,ca,mb,cb,mc,cc) =
 let g = construire_graphe (ma,ca,mb,cb,mc,cc)
 in map projection g.sommets ;;

let est_accessible (x,y,z) (ma,ca,mb,cb,mc,cc) =
 let composante = liste_accessibles (ma,ca,mb,cb,mc,cc) in
 filtre (identiques [x;y;z]) composante <> [] ;;
```

La fonction `liste_accessibles` dresse la liste des sommets de la composante connexe de i ; en appliquant cette fonction aux données du problème, nous constatons qu'il y en a seize.

Question 9 • $(8,0,0) \to (3,5,0) \to (3,2,3) \to (6,2,0) \to (6,0,2) \to (1,5,2) \to (1,4,3) \to (4,4,0)$.

Question 10 • La composante connexe est un graphe partiellement orienté ; seules les arêtes ont été représentées.

Question 11 • Les huit états inaccessibles sont $(3,4,1)$, $(4,3,1)$, $(5,2,1)$, $(6,1,1)$, $(2,4,2)$, $(3,3,2)$, $(4,2,2)$ et $(5,1,2)$. De chacun de ces états partent six arcs qui mènent tous à la ≪grande≫ composante connexe.



FIN