TD 12

Automates émondés

On définit le type automate par :

```
type automate = { initial : int list; finals : int list ; transition : (char* int) list array };
```

Exercice 1 : Écrire les fonctions utiles au TD :

(-a-) La fonction reconnait la présence d'une valeur dans une liste :

```
mem : 'a -> 'a list -> bool
```

(-b-) La fonction donnant le second membre associé à une valeur dans une liste de couples :

```
assoc : 'a -> ('a * 'b) list -> 'b
```

(-c-) la fonction reconnait la présence du premier terme dans une liste de couples :

```
mem_assoc : 'a -> ('a * 'b) list -> bool
```

(-d-) La fonction appliquant à l'ensemble de la liste la fonction f :

```
map : ('a -> 'b) -> 'a list -> 'b list
```

(-e-) La fonction renvoyant l'union sans doublon de deux listes :

```
union : 'a list -> 'a list -> 'a list
```

(-f-) La fonction renvoyant l'intersection de deux listes :

```
intersection : 'a list -> 'a list -> 'a list
```

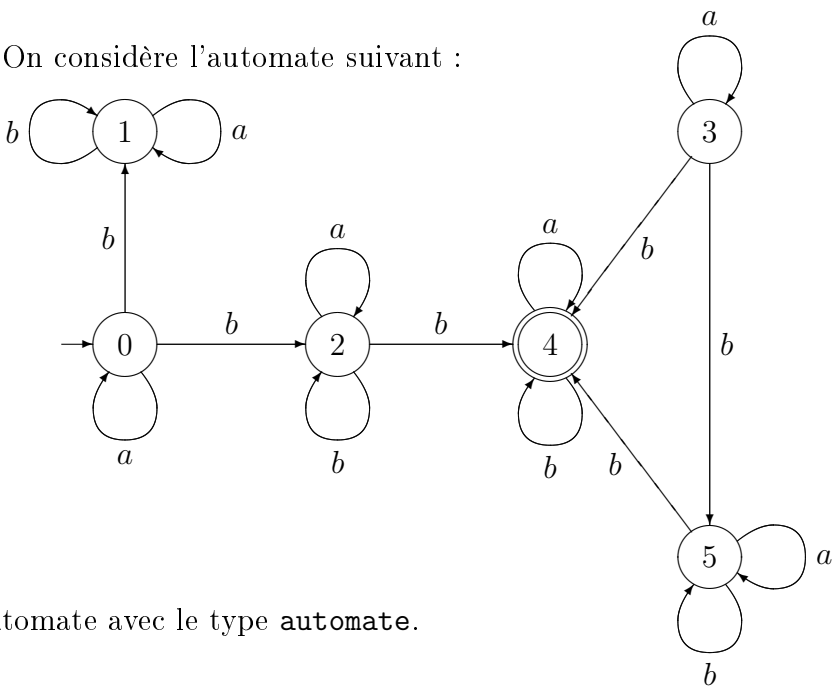
(-g-) La fonction renvoyant pour deux listes l_1 et l_2 en entrée, les éléments de l_2 qui ne sont pas dans l_1 :

```
difference : 'a list -> 'a list -> 'a list
```

(-h-) La fonction qui transforme une liste de liste en une liste sans doublon :

```
list_list : 'a list list -> 'a list
```

Exercice 2 : On considère l'automate suivant :



Définir l'automate avec le type `automate`.

Exercice 3 : Écrire une fonction `trans` qui renvoie pour un r donné, la liste des états s tels que : $\exists a \in A, (r, a, s) \in E$.

```
trans : automate -> int -> int list
```

Exercice 4 : **États accessibles :**
On désire écrire une fonction qui renvoie la liste des états accessibles d'un automate.
Le principe de l'algorithme est simple, on commence par poser $Q_{acc,0} = I$ (les états initiaux sont accessibles par définition).
Si n est un entier et si nous avons déjà calculé $Q_{acc,n}$ à la nième itération, alors on a :

$$Q_{acc,n+1} = Q_{acc,n} \cup \bigcup_{\substack{q \in Q_{acc,n} \\ a \in A}} \{q' \in Q | \, (q, a, q') \in E\}$$

L'algorithme se termine quand cette suite est constante.

Écrire une fonction `accessible` qui renvoie la liste des états accessibles d'un automate donnée.

```
accessible : automate -> int list
```

Exercice 5 : **États co-accessibles :**

(a) Écrire une fonction `miroir_automate` qui renvoie le miroir d'un automate donné.

```
miroir_automate : automate -> automate
```

(b) Écrire une fonction `co_accessible` qui renvoie la liste des états co-accessibles d'un automate donné.

```
co_accessible : automate -> int list
```

Exercice 6 : **États utiles : :**

En déduire une fonction `utile` qui renvoie la liste des états utiles d'un automate donnée :

```
utile : automate -> int list
```

Exercice 7 :

(a) Écrire une fonction `sous_graphe` qui renvoie l'automate privé de l'état i donné (on gardera la même taille pour le vecteur transition dont on remplacera la composante i par la liste vide, et on effacera toutes les transitions avec i) :

```
sous_graphe : automate -> int -> automate
```

(b) En déduire la fonction `emonde` qui renvoie l'automate émondé :

```
emonde : automate -> automate
```