Créer une table de hachage en OCaml

Le module `Hashtbl` d'OCaml implémente le type abstrait Dictionnaire qui permet la gestion d'ensembles dynamiques d'associations clés-valeurs, dont les clés sont 2 à 2 distinctes.
Les `Hashtbl` permettent ainsi de représenter :

→ des ensembles dynamiques (les éléments sont les clés, on peut leur associer une valeur booléenne indiquant la présence) ;
→ des multi-ensembles (en remplaçant la présence booléenne par un nombre d'occurrences entier) ;
→ ou plus généralement une fonction sur un ensemble fini qu'on n'arrive pas à mettre en bijection avec $[\![0, n[\![$, sans quoi on ferait simplement un tableau et pas une `Hashtbl`.

Une table dont les clés sont de type `'a` et dont les valeurs sont de type `'b` est de type `('a, 'b) Hashtbl.t`, cependant ce type n'est fixé que lors du premier ajout d'une association clé-valeur à la table.

On crée une table de hachage comme suit, en précisant `n` une estimation du nombre de clés que contiendra la table. Cette valeur est donnée seulement en vue d'améliorer les performances de la table, elle ne limite pas le nombre de clés.

```
let tbl = Hashtbl.create n
```

Pour manipuler la table, on dispose des fonctions élémentaires suivantes.

- `Hashtbl.mem : ('a, 'b) Hashtbl.t -> 'a -> bool` qui teste la présence d'une clé dans la table.
- `Hashtbl.find : ('a, 'b) Hashtbl.t -> 'a -> 'b` qui donne la valeur associée à une clé présente dans la table. Si la clé n'est pas présente dans la table, l'exception `Not_found` est levée.
- `Hashtbl.add : ('a, 'b) Hashtbl.t -> 'a -> 'b -> unit` qui permet d'ajouter une association clé-valeur à la table. Si la clé était déjà présente, la valeur associée est masquée par la nouvelle valeurs.
- `Hashtbl.replace : ('a, 'b) Hashtbl.t -> 'a -> 'b -> unit` agit comme `add` sauf que si la clé était déjà présente, la valeur qui lui était associée est écrasée.

De plus on dispose aussi d'une fonctionnelle `fold` pour itérer sur les associations de la table.

```
Hashtbl.fold : ('a -> 'b -> 'c -> 'c) -> ('a, 'b) Hashtbl.t -> 'c -> 'c
```

On remarque que l'ordre des arguments dans la fonction d'accumulation diffère de celui pour la fonctionelle `List.fold_left`, en effet ici l'accumulateur est passé en troisième argument.