TP D'OPTION INFORMATIQUE 3
Implémentation de structures de données

1 Implémentation des piles

On rappelle la liste d'opérations de la structure mutable de pile :

- `creer_pile : unit -> 'a pile`
- `est_vide : 'a pile -> bool`
- `empiler : 'a pile -> 'a -> unit`
- `depiler : 'a pile -> 'a`

Implémentation par référence de liste

On considère d'abord une implémentation par références de liste :
`type 'a pile = 'a list ref`
Le sommet de la pile est alors l'élément en tête de la liste référencée. Implémenter dans ce type `'a pile` les opérations de la structure de pile en Ocaml. Déterminer leur complexité.

Implémentation par tableau

Dans ce choix d'implémentation, on fixe une **capacité maximale** stockée dans une variable globale `c`, et on représente une pile par un tableau de taille $c+1$, dont la première case contient la taille t actuelle de la pile, et les t cases suivantes contiennent les valeurs de la pile, la dernière (ie d'indice t) contenant le sommet. les valeurs contenues dans les $c-t$ cases restantes n'ont aucun impact sur la pile représentée.
Puisque le type des éléments du tableau est fixé à la création, on conviendra que cette implémentation ne traite que des piles d'entiers :

```
let c = 300;;
type pile = int array;;
```

1. Donner, pour $c = 10$, un exemple de tableau représentant dans ce type la pile $\begin{pmatrix}3\\5\\2\end{pmatrix}$
2. Implémenter les opérations de la structure de pile dans cette représentation. On traitera dans `empiler` le cas du dépassement de la capacité maximale en provoquant une erreur.
3. Déterminer la complexité de ces opérations.

Implémentation par enregistrements

On cherche à adapter l'implémentation précédente pour que le type des valeurs de la pile puisse être quelconque. On utilise à cet effet un **type enregistrement** :
`type 'a pile = {mutable taille : int ; tab : 'a option array};;`
Si p est un élément de ce type, p a deux champs `p.taille` et `p.tab`. Le mot-clé `mutable` indique que l'attribut `taille` peut être modifié, en écrivant par exemple `p.taille <- p.taille + 1`.
Le type `'a option` indique que chaque case du tableau `tab` contient soit la valeur `None`, soit une valeur `Some x` avec `x` de type `'a`. On pourra utiliser la fonction `Option.get : 'a option -> 'a` pour extraire la valeur d'une case non vide.
On fixe toujours une capacité maximale c et on représente une pile par un enregistrement contenant la taille t actuelle de la pile, et un tableau de c cases dont les t premières cases contiennent les valeurs de la pile. Lors de la création d'une pile vide, on initialisera les cases du tableau avec la valeur `None`.
Implémenter les opérations des piles pour cette représentation et donner leur complexité (l'opération `<-` est une opération élémentaire, et la fonction `Option.get` est en $O(1)$).

2 Implémentation des files

On rappelle la liste d'opérations de la structure mutable de file :

- `creer_file : unit -> 'a file`
- `est_vide : 'a file -> bool`
- `enfiler : 'a file -> 'a -> unit`
- `defiler : 'a file -> 'a`

Par couple de références de listes

On représente une file $(a_1, \dots, a_n)$ par une référence contenant deux listes $[a_1, \dots, a_r], [a_n, \dots, a_{r+1}]$, de sorte de pouvoir accéder efficacement à la tête et à la queue :
`type 'a file = ('a list * 'a list) ref;;`
Les éléments enfilés vont dans la seconde liste, et ceux défilés viennent de la première. Dans le cas où celle-ci est vide, on transvase d'abord tous les éléments de la seconde vers la première.
Implémenter les opérations des files pour cette représentation et donner leur complexité. Que peut-on dire de la moyenne de la complexité de n appels successifs à `defiler` ?

Par enregistrement

On adapte la représentation dans un type enregistrement au cas des files :
`type 'a file = {mutable taille : int; mutable tete: int; tab :'a option array};;`
L'attribut `tete` donne la position de la tête de la file dans le tableau. Cette case et les suivantes contiennent les *taille* valeurs de la file. La représentation est circulaire : s'il n'y a pas assez de cases à droite de la position de la tête, on reprend au début du tableau.

1. Déterminer la file représentée par :

```
{taille = 4;
 tete = 5;
 tab = [|Some 3; Some 5; Some 1; Some 2; Some 6; Some 6; Some 0|]
}
```

2. Implémenter les opérations des files pour cette représentation et donner leur complexité.

3 Implémentation des dictionnaires

On rappelle la liste d'opérations de la structure mutable de dictionnaire :

- `creer_dictionnaire : unit -> ('a, 'b) dico`
- `ajouter_couple : ('a, 'b) dico -> 'a -> 'b -> unit`
- `cle_presente : ('a, 'b) dico -> 'a -> bool`
- `valeur_associee : ('a, 'b) dico -> 'a -> 'b`
- `supprimer : ('a, 'b) dico -> 'a -> unit`
- `modifier_valeur : ('a, 'b) dico -> 'a -> 'b -> unit`

Par une référence de liste

Proposer une implémentation représentant un dictionnaire par une référence de liste de couples. Préciser la complexité de chaque opération.

Par enregistrement

Proposer une implémentation des dictionnaires stockant la taille et un tableau de couples dans un enregistrement. Le tableau sera trié selon les clés, afin que la recherche d'une clé puisse être dichotomique. Préciser la complexité de chaque opération.