La méthode de compression R.L.E. (*Run Length Encoding*) permet de coder des suites de caractères, avec l'espoir de faire diminuer la taille de la place mémoire occupée par ces données. L'idée de R.L.E. est simple et consiste à remplacer toute séquence de n caractères c identiques consécutifs par les deux informations n et c. Par exemple, le codage R.L.E. de la suite `aaabbaccccbbb` produira la suite `3a2b1a4c3b`.

Dans cet exercice, on suppose que l'on dispose des deux fonctions suivantes :

- la fonction `enleve`, de type `int -> 'a list -> 'a list`, qui étant donnés un entier naturel n et une liste ℓ rend la liste ℓ privée de ses n premiers éléments – si n est supérieur à la longueur de ℓ, cette fonction rend la liste vide.

- la fonction `lg_prefix`, de type `'a list -> int`, qui étant donnée une liste ℓ rend le nombre d'éléments égaux consécutifs apparaissant en début de ℓ.

```
# enleve 0 [1;2;3];;
- : int list = [1; 2; 3]
# enleve 5 [1;2];;
- : int list = []
# enleve 3 [1;2;3;4;5;6;7];;
- : int list = [4; 5; 6; 7]
```

```
# lg_prefix [];;
- : int = 0
# lg_prefix [1;2;3];;
- : int = 1
# lg_prefix [1;1;1;2;2;3];;
- : int = 3
```

1. Puisque tous les éléments d'une liste doivent avoir le même type, le type des éléments de la liste obtenue après codage de la liste de caractères initiale doit permettre de regrouper deux informations : le caractère considéré et le nombre de fois où il apparaît (avant le caractère différent suivant). Définir ce type par un type enregistrement (type `encode`).

2. Définir une fonction `codage`, de type `char list -> encode list` qui permet de coder une liste de caractères avec la méthode R.L.E. .

En fait, un entier est codé sur 2 octets et un caractère est codé sur 1 octet. Aussi, le codage R.L.E. n'apporte pas systématiquement un gain de place mémoire. Par exemple, la suite `abcde` devient, une fois codée, `a1b1c1d1e1`. Initialement, cette suite occupe donc 5×1 octets et une fois codée, elle en occupe $5 \times (1+2)$ octets. On se propose donc de modifier la fonction `codage` de manière à ce que la place mémoire qu'occupe une suite codée ne puisse pas être supérieure à celle qu'elle occupait initialement. Pour cela, on effectuera le codage uniquement lorsque le nombre de fois où apparaît un caractère est strictement supérieur à 3, dans le cas contraire, le codage n'apportant aucun gain, on conservera la séquence du caractère considéré. Par exemple, la suite `aaabbaccccbbb` sera dorénavant codée par `aaabba4cbbb`.

3. En adoptant cette stratégie, il donc nécessaire de modifier le type des éléments de la liste représentant la suite codée. Pour cela, on souhaite regrouper en un seul type les listes de caractères et le type `encode` défini à la question 1. Proposer un type somme `info` dont les deux constructeurs permettent une telle définition.

4. Définir une fonction `repete` de type `'a -> int -> 'a list` qui étant donnés un entier n et un élément e construit une liste contenant n occurrences de e.

5. Définir et donner le type d'une fonction `codage2` qui permet de mettre en oeuvre le codage décrit.

6. Définir une fonction `decodage` qui étant donnée une liste ℓ de type `info list` retourne la liste ℓ' telle que $\ell = \texttt{codage2}(\ell')$.

7. Définir et donner le type d'une fonction `tailles`, qui étant donnée une liste codée retourne la paire d'entiers (k_1, k_2) où k_1 (resp. k_2) correspond au nombre d'octets occupés par la suite codée (resp. la suite initiale). Par exemple, en appliquant la fonction `tailles` à la suite `aaabba4cbbb` (obtenue en codant la suite `aaabbaccccbbb`), on obtient la paire (12,13).

8. Le taux de compression apporté par un codage correspond au rapport :

$$\frac{\text{nombre d'octets occupés par la suite codée}}{\text{nombre d'octets occupés par la suite initiale}}$$

   Définir et donner le type d'une fonction `taux` permettant de calculer le taux de compression apporté par la fonction `codage2`. Cette fonction prend en argument la suite codée. Par exemple, le taux de compression obtenu en codant la suite `aaabbaccccbbb` par `aaabba4cbbb` est de `0.923076923077`.