Pour commencer ce TP, il faut télécharger le fichier `fenelon_tp07.zip` sur la page web du cours.

# 1 Lecture de fichier et découpe de texte

Le but de ce TP est d'implémenter un générateur de texte. Pour cela, notre programme va d'abord devoir analyser un texte existant. Pour le modèle que nous allons utiliser dans les parties suivantes, cette analyse consiste à découper le texte en une liste de mots, au niveau des espaces.

1. Écrire une fonction `indice_premier_espace : string -> int option` prenant en argument une chaine de caractères, et renvoyant une valeur `Some i` où $i$ est l'indice du premier espace apparaissant dans la chaine. On renverra `None` si la chaine ne contient pas d'espace, et on utilisera l'`exception Trouve of int` pour interrompre la recherche dès qu'un espace est trouvé.

Exemple

```
# indice_premier_espace "ceci est un test";;
- : int option = Some 4
# indice_premier_espace "ceci_est_un_test";;
- : int option = None
```

2. Écrire une fonction `decoupe : string -> string * string` telle que `decoupe texte` renvoie un couple `(mot, texte_suivant)` tel que :
   - `mot` ne contienne pas d'espace ;
   - `texte = mot ^ " " ^ texte_suivant`.

   Si `texte` ne contient pas d'espace, on renverra le couple `(texte, "")`.

Exemple

```
# decoupe "ceci est un test" ;;
- : string * string = ("ceci", "est un test")
# decoupe "test" ;;
- : string * string = ("test", "")
```

3. Écrire une fonction `lire : string -> string list` prenant en argument un nom de fichier et renvoyant la liste de ses mots.

Exemple

```
# lire "textes/exemple.txt" ;;
- : string list =
["Le"; "chat"; "noir"; "aux"; "yeux"; "verts"; "dormait"; "paisiblement";
 ...
 "se"; "raconter"; "des"; "histoires."]
```

**Remarque :** si le fichier n'est pas trouvé, on pourra utiliser la commande suivante dans la console OCaml : `#cd "chemin/vers/le/bon/dossier/" ;;`

# 2 Modèle de langage de Markov d'ordre 1

Un **modèle de langage de Markov** est un modèle probabiliste qui génère des textes en s'inspirant d'un texte source qu'il a au préalable analysé. Ce type de modèle repose sur l'**hypothèse de Markov**, qui stipule que, lorsqu'un début de texte a déjà été généré, la valeur du mot suivant ne dépend que d'un nombre fixe de mots précédents. Un **modèle de langage de Markov d'ordre** 1 utilise uniquement le dernier mot généré pour prédire le mot suivant.

On souhaite donc analyser un texte source (mis sous la forme d'une liste de mots), et, pour chaque mot du texte, on veut calculer la liste des mots suivants possibles. Pour stocker le résultat de cette analyse, on va utiliser une table de hachage redimensionnable, comme implémentée dans le TP précédent.

Une telle structure est en fait déjà implémentée en OCaml par le module `Hashtbl`, qui fournit un type `('a, 'b) Hashtbl.t` (où `'a` est le type des clés, et `'b` et le type des valeurs), ainsi que les fonctions suivantes :

- `Hashtbl.create : int -> ('a, 'b) Hashtbl.t` :
  ↪ `Hashtbl.create w` renvoie une nouvelle table de hachage vide dont la largeur initiale est $\omega$.

  On peut choisir une bonne valeur de $\omega$ si l'on sait à peu près combien de clés va contenir la table, ou prendre une valeur arbitraire (par exemple $\omega = 42$) et OCaml va agrandir cette valeur si besoin (comme on l'a fait dans le TP précédent).
- `Hashtbl.mem : ('a, 'b) Hashtbl.t -> 'a -> bool` :
  ↪ `Hashtbl.mem table cle` renvoie `true` si `cle` est une clé de `table` (et `false` sinon).
- `Hashtbl.find : ('a, 'b) Hashtbl.t -> 'a -> 'b` :
  ↪ `Hashtbl.find table cle` renvoie la valeur associée à `cle` dans `table`.
- `Hashtbl.add : ('a, 'b) Hashtbl.t -> 'a -> 'b -> unit` :
  ↪ `Hashtbl.add table cle v` ajoute l'association clé-valeur `(cle,v)` à `table`.

Dans cette partie, on va construire une table de hachage dont les clés sont les mots apparaissant dans le texte source, et la valeur associée à une clé est la liste des mots suivants possibles après le mot clé dans le texte source.

4. Écrire une fonction `ajout : ('a, 'b list) Hashtbl.t -> 'a -> 'b -> unit` prenant en argument une table de hachage et deux mots `cle` et `suivant`, et ajoutant `suivant` dans la liste des mots associés à `cle`. On prendra garde à gérer le cas où `cle` n'apparait pas encore dans la table.

Dans sa suite de cette partie, pour abréger les types des fonctions, on définit le type suivant :

```
type modele_1 = (string, string list) Hashtbl.t
```

5. Écrire une fonction `gen_modele : 'a list -> modele_1` prenant en argument une liste de mots et renvoyant une table de hachage représentant un modèle de Markov d'ordre 1 pour cette liste de mots.

Exemple

```
# let modele_exemple = gen_modele (lire "textes/exemple.txt") ;;
val modele_exemple : modele_1 = <abstr>
# Hashtbl.find modele_exemple "Le" ;;
- : string list = ["feu"; "vent"; "soleil"; "chien,"; "chat"]
```

6. Écrire une fonction `valeur_indice : 'a list -> int -> 'a` telle que `valeur_indice l i` renvoie la $i$-ème valeur de la liste `l` (on lèvera une erreur si la valeur $i$ est trop grande).

Exemple

```
# valeur_indice (Hashtbl.find modele_exemple "Le") 4 ;;
- : string = "chat"
```

7. Écrire une fonction `valeur_hasard : 'a list -> 'a` prenant en argument une liste `l` et renvoyant une valeur de `l` au hasard (avec probabilité uniforme).

Exemple

```
# valeur_hasard (Hashtbl.find modele_exemple "Le") ;; (* À tester plusieurs fois *)
- : string = "soleil"
```

8. Écrire une fonction `avec_point : string -> bool` qui teste si une chaine de caractères termine par un point.

Exemple

```
# avec_point "MP2I" ;;
- : bool = false
# avec_point "MP2I." ;;
- : bool = true
```

9. Écrire une fonction `gen_txt : modele_1 -> string -> int -> string list` prenant en arguments un modèle de Markov d'ordre 1, un mot de départ, et un nombre de phrases, et renvoyant une liste de mots générés par le modèle.
On s'arrêtera de générer des mots quand le nombre de phrases désiré sera atteint.

Exemple
```
# gen_txt modele_exemple "Le" 1 ;;
- : string list = ["Le"; "chat"; "endormi"; "non"; "loin"; "de"; "là."]
```

10. Écrire une fonction `ecrire : string -> string list -> unit` prenant en arguments un nom de fichier et une liste de mots, et écrivant dans ce fichier le texte représenté par la liste de mots.
On séparera chaque mot par un espace, on écrira chaque phrase sur une même ligne, et on reviendra à la ligne à la fin de chaque phrase.

Exemple
```
# ecrire "resultat.txt" (gen_txt modele_exemple "Le" 5) ;;
- : unit
```

resultat.txt
```
Le chat endormi non loin de fromage qu'elle avait trouvé dans le jardin.
La lune, pleine et invitant les collines.
Le feu crépitait joyeusement dans la cheminée, répandant une mélodie joyeuse depuis
la pièce et invitant les yeux.
Le chat endormi non loin de tous les fenêtres.
La lune, pleine et à couper le jardin.
```

11. Tester votre fonction en utilisant le Comte de Monte-Cristo d'Alexandre Dumas comme texte source.

## 3 Modèle de langage de Markov d'ordre $n$

Les résultats obtenus dans la partie précédente peuvent ne pas être très convaincants. Dans cette partie, on généralise ce qu'on a fait dans la partie 2.

Un **modèle de Markov d'ordre** $n$ utilise les $n$ derniers mots générés pour prédire le mot suivant. On appelle **contexte** la liste des $n$ derniers mots générés.

12. Écrire une fonction `nouveau_contexte : 'a list -> 'a -> 'a list` prenant en argument un contexte ainsi qu'un nouveau mot qu'on vient de généré, et renvoyant le nouveau contexte.

Exemple
```
# nouveau_contexte ["ceci"; "est"; "un"] "test" ;;
- : string list = ["est"; "un"; "test"]
```

13. Écrire une fonction `contexte_depart : int -> string list` renvoyant une liste contenant $n$ fois la chaine de caractères vide.

Exemple
```
# contexte_depart 5 ;;
- : string list = ["";"";"";"";""]
```

Dans sa suite de cette partie, pour abréger les types des fonctions, on définit le type suivant :

```
type modele_n = (string list, string list) Hashtbl.t
```

14. Écrire une fonction `gen_modele_ordre : int -> string list -> modele_n` prenant en arguments un entier $n$ et une liste de mots, et renvoyant une table de hachage représentant un modèle de Markov d'ordre $n$ pour cette liste de mots.

Exemple

```
# let modele_exemple2 = gen_modele_ordre 2 (lire "textes/exemple.txt") ;;
val modele_exemple2 : modele_n = <abstr>
# Hashtbl.find modele_exemple2 ["le"; "ciel"] ;;
- : string list = ["nocturne,"; "bleu"]
```

15. Écrire une fonction `gen_txt_ordre : int -> modele_n -> string list -> int -> string list` prenant en arguments un entier $n$, un modèle de Markov d'ordre $n$, un contexte de départ, et un nombre de phrases, et renvoyant une liste de mots générés par le modèle.
On s'arrêtera de générer des mots quand le nombre de phrases désiré sera atteint.

Exemple

```
# test_ordre2 () ;;
- : unit
```

resultat.txt

```
Le chat noir aux yeux verts dormait paisiblement sur le toit en tuiles rouges,
créant une mélodie apaisante qui berçait les habitants de la maison.
Le feu crépitait joyeusement dans la cuisine, se méfiant du chat endormi non loin
de là.
```

16. Comparer les résultats obtenus sur Le Comte de Monte-Cristo avec un modèle d'ordre 2 ou 3, par rapport au modèle d'ordre 1.
17. Tester le modèle d'ordre 1 ou 2 sur le discours d'investiture de Donald Trump.

Vous pouvez également tester vos programmes avec d'autres textes sources, que vous pouvez par exemple trouver sur le site du **Projet Gutenberg** : `https://www.gutenberg.org/`