Projet : Analyse d'un fichier texte

Consignes :
- Lisez bien tout le sujet avant de commencer à coder.
- Déposez avant le lundi **9 mai** une archive au format `.tar.gz`, contenant votre code (fichiers `.ml`, `.mli`, etc.) et un rapport au **format PDF**, dans le dépôt `iprf_projet` sur `http://exam.ensiie.fr`.
- Pensez à commenter votre code. Au minimum, chaque déclaration de fonction devra être précédée d'un commentaire expliquant ce que la fonction est censée faire.

L'objectif de ce projet est de concevoir un programme permettant d'analyser un fichier texte. Le fichier ne sera lu qu'une seule fois pour extraire la liste de ses mots. Ensuite, il s'agira de déterminer les éléments suivants :

1. nombre de mots,
2. nombre de mots différents,
3. mot le plus long et sa taille,
4. mot le plus utilisé et son nombre d'occurrences,
5. liste des mots de plus de n lettres,
6. liste des mots répétés plus de k fois,
7. liste des mots commençant par un préfixe donné.

Dans tout le sujet, on désignera par *lettre valide* tout caractère qui est soit une lettre minuscule, soit une lettre majuscule, soit -, soit '. Vous pouvez ajouter d'autres lettres valides (notamment é) pour traiter plus de mots. Dans ce cas, méfiez vous des problèmes liés à l'encodage du fichier, et précisez dans votre rapport vos ajouts et les conditions d'utilisation.

Enfin, on utilisera dans tout le sujet le type OCaml suivant :

```
type word = char list ;;
```

Attention, il ne s'agit pas du type `string` de OCaml. Nous utilisons ici un type reposant sur une liste car c'est ce dont on aura besoin à la partie 3.

1 Échauffement

Question 1. Écrire une fonction `is_a_letter: char -> bool` qui, sur la donnée d'un caractère `c`, retourne `true` si `c` est une *lettre valide*, et `false` sinon.

Question 2. Écrire une fonction `is_valid: word -> bool` qui, sur la donnée d'un mot `w`, retourne `true` si `w` est non vide et uniquement composé de lettres valides, et `false` sinon.

Question 3. Écrire une fonction `to_lower: word -> word` qui, sur la donnée d'un mot `w`, retourne une copie de `w` où les majuscules ont été remplacées par les minuscules correspondantes.

note : Utiliser la fonction `Char.lowercase` de OCaml.

Question 4. Écrire une fonction `trim: word -> word` qui prend un mot `w` et retourne une copie de `w` où on a retiré tout caractère suivant la dernière lettre valide.

Par exemple, on doit obtenir :
- `trim ['e'; 't'; 'c'; '.'; '.'; '.']` ⇝ `['e'; 't'; 'c']`
- `trim ['a'; '.'; 'b']` ⇝ `['a'; '.'; 'b']`

2 Récupération de la liste des mots d'un fichier texte

Vous trouverez un code permettant d'obtenir une première liste de mots à partir d'un fichier ici : `http://www.ensiie.fr/~christophe.mouilleron/Teaching/IPRF/projet/textproc.ml`

Question 5. Écrire une fonction `print_word: word -> unit` permettant d'afficher un mot. Pour tester cette fonction et le code fourni, essayer d'afficher tous les mots d'un fichier texte de votre choix.

Question 6. Expliquer dans le rapport comment fonctionne le code fourni.

Question 7. Écrire une fonction `nub: 'a list -> 'a list` qui, sur la donnée d'une liste ℓ retourne la liste des éléments de ℓ sans doublon.

Question 8. Écrire une fonction `count_words: string -> (int * int)` qui, sur la donnée du chemin d'un fichier texte, retourne un couple d'entier (m, n) où :
– m est le nombre de mots valides dans le fichier,
– n est le nombre de mots valides différents dans le fichier.

Question 9. Tester la fonction `count_words` sur de gros fichiers[1], et expliquer dans le rapport pourquoi cette approche est inefficace.

3 La structure de Trie

La clé pour obtenir rapidement des réponses aux 7 questions de l'introduction est de stocker l'information dans un trie. Il s'agit d'une structure de données efficace pour réaliser l'association entre des suites de caractères et des valeurs. Dans notre cas, nous allons associer à chaque mot valide (de type `word = char list`) son nombre d'occurrences dans le fichier.

Visuellement, un trie est un arbre où chaque nœud (interne ou feuille) contient une valeur, et où chaque fils est associé à un caractère. Les nœuds internes peuvent ainsi avoir autant de fils que de caractères valides différents. La recherche de la valeur associée à une suite de caractères se fait en partant de la racine et en choisissant à chaque étape le fils correspondant à le prochain caractère à lire. Lorsqu'on a consommé tous les caractères, on prend la valeur stockée dans le nœud courant.

La figure 1 illustre le trie qu'on obtient à partir de la suite de mots `«le la les le un»`. La figure 2 montre que la valeur associée au mot `le` est bien la valeur 2.

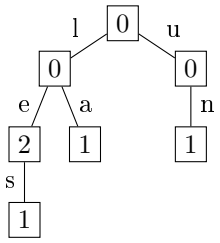

FIGURE 1 – Exemple de trie.

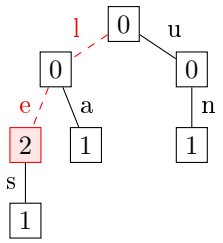

FIGURE 2 – La valeur associée à `[ 'l'; 'e' ]` est 2.

En OCaml, on définit les trie à l'aide du type inductif suivant :

```
module CharMap = Map.Make (Char) ;;
type trie = T of int * trie CharMap.t ;;
```

Un trie est donc un nœud de la forme `T (v, m)` où `v` est la valeur stockée dans le nœud, et où `m` est une association qui permet de récupérer (si possible) un fils en fonction d'un caractère. Le trie *vide* est donc

```
let empty_trie = T (0, CharMap.empty) ;;
```

1. Certaines pages sur Wikipedia ont un code source de plusieurs centaines de kilooctets.

et le trie de la figure 1 peut être construit (péniblement) via le code suivant :

```
let ns      = T (1, CharMap.empty) ;;
let ne      = T (2, CharMap.add 's' ns CharMap.empty) ;;
let na      = T (1, CharMap.empty) ;;
let nl      = T (0, CharMap.add 'a' na (CharMap.add 'e' ne CharMap.empty)) ;;
let nn      = T (1, CharMap.empty) ;;
let nu      = T (0, CharMap.add 'n' nn CharMap.empty) ;;
let example = T (0, CharMap.add 'l' nl (CharMap.add 'u' nu CharMap.empty)) ;;
```

Question 10. Écrire la fonction `trie_get: word -> trie -> int` qui, sur la donnée d'un mot `w` et d'un trie `t`, renvoie la valeur associée à `w` dans `t`.

Si on ne peut pas lire le mot `w` (cas où il n'y a aucun fils correspondant à la lettre qu'on cherche à lire), on renverra la valeur `0`.

Question 11. Écrire la fonction `trie_incr: word -> trie -> trie` qui, sur la donnée d'un mot `w` et d'un trie `t`, renvoie un nouveau trie `t'` dans lequel la valeur associée à `w` a été augmentée de 1.

On utilisera pour ce faire l'algorithme suivant :

Algorithme 1 : `trie_incr`

Entrée : un mot w et un trie $t = T\ (v,\ m)$
Sortie : un trie t' où la valeur associée à w a été augmentée de 1

si *w est la liste vide* **alors** **retourner** $T\ (v+1,\ m)$
sinon
  Découper w en $x : xs$
  si *x est une clé valide de m* **alors**
    $s \leftarrow$ valeur associée à x dans m
  sinon
    $s \leftarrow$ trie vide
  $s' \leftarrow$ `trie_incr`$(xs,\ s)$
  $m' \leftarrow m$ où on a changé la valeur associée à x par s'
  retourner $T\ (v,\ m')$

Question 12. En vous aidant du code fourni à la section 2, écrire une fonction `trie_words: string -> trie` qui, sur la donnée du chemin vers un fichier, renvoie le trie construit à partir des mots de ce fichier.

Question 13. Écrire une fonction `trie_card: trie -> int` qui compte le nombre de nœuds avec une valeur non nulle dans un trie donné.

Question 14. Écrire une fonction `trie_sum: trie -> int` qui retourne la somme des valeurs contenues dans les nœuds d'un trie donné.

Question 15. Écrire une fonction `count_words_v2: string -> (int * int)`, qui retourne la même chose que `count_words` mais en faisant appel au code des deux questions précédente.

Vérifier en faisant des tests sur de gros fichiers que cette nouvelle version est beaucoup plus rapide.

Question 16. Proposer des fonctions pour répondre, à partir du trie associé à un fichier, aux 5 autres questions posées dans l'introduction.