

Base du Développement Logiciel

21 janvier 2022

1 Sujet

L'objectif de ce TP est de réaliser un compresseur (un peu naïf) de texte. Chaque mot sera remplacé par une séquence de caractères et celle-ci sera d'autant plus courte que le mot apparaît souvent dans le texte. Ainsi les mots les plus courants n'occuperont plus qu'un seul caractère.

Attention : la gestion des fichiers n'est pas la même sous Windows et Unix. Pour vous simplifier le travail, faites votre projet sous Unix.

L'une des structures de données utilisée sera `std::map<key,value>`. C'est une table associative qui, à chaque élément de type `key`, associe un élément de type `value`. Parmi les fonctions membres intéressantes, il y a :

- `insert(std::pair<key,value> const &),`
- `erase(iterator),`
- `begin(),`
- `size(),`
- `operator[] (key const &).`

L'opérateur `[]` permet d'accéder à la valeur associée à une clé donnée. La fonction `begin` renvoie un itérateur qui, une fois déréférencé, donne une paire `<key,value>`. Cette paire est celle de la table qui a le membre de gauche le plus petit (pour un certain ordre qui dépend du type `key`). L'itérateur renvoyé par `begin` peut aussi être passé à la fonction `erase` pour supprimer la paire en question de la table. La fonction `size` renvoie le nombre de paires présentes dans la table.

Le type `std::multimap<key,value>` propose le même genre d'interface, si ce n'est que plusieurs paires de la table peuvent maintenant posséder le même membre de gauche. L'opérateur `[]` n'a donc plus aucun intérêt mais les autres fonctions restent utilisables.

Aussi bien pour `map` que pour `multimap`, il est possible d'utiliser la syntaxe suivante de boucle pour itérer sur toutes les paires `e` contenues dans la table `m`.

```
for(std::pair<key,value> const &e : m) { ... }
```

2 Lecture de fichier

Implémentez la fonction `load` qui lit le fichier `1984.txt` fourni et renvoie un vecteur de chaînes de caractères contenant un mot par case. Notez que ce fichier a été nettoyé de toute sa ponctuation et qu'il contient un seul mot par ligne afin de simplifier le travail.

Les fonctions et méthodes suivantes pourront être utiles pour lire le fichier :

- `std::ifstream(char const *),`
- `std::getline(std::istream &, std::string &).`

3 Comptage des occurrences

Le type suivant définit une table qui associe à chaque mot son nombre d'occurrences :

```
typedef std::map<std::string, int> occs;
```

Implémentez la fonction `count` qui prend en argument un vecteur de mots et renvoie une table associative qui associe à chaque mot son nombre d'occurrences dans le vecteur.

4 Création des codes

On associera à chaque mot un encodage qui est, dans un premier temps, une chaîne de ‘0’ et de ‘1’. Le type suivant décrit une telle association :

```
typedef std::map<std::string, std::string> codes;
```

Implémentez la fonction `prefix` qui prend en argument une paire clé-valeur de type `codes` et une chaîne de caractère. Elle renvoie alors une paire clé-valeur contenant la même clé que son argument d’entrée mais une valeur contenant la concaténation du 2e argument et de la valeur de son deuxième argument d’entrée.

Implémentez la fonction `merge` qui prend deux arguments de type `codes` et en renvoie un. La table renvoyée doit associer à chaque mot du premier argument son code préfixé par ‘0’ et à chaque mot du second argument son code préfixé par ‘1’. Par exemple, si les arguments sont les tables $\langle \text{“baz”} \mapsto \text{“x”} \rangle$ et $\langle \text{“bar”} \mapsto \text{“y”}; \text{“foo”} \mapsto \text{“z”} \rangle$, alors la table renvoyée doit être $\langle \text{“bar”} \mapsto \text{“1y”}; \text{“baz”} \mapsto \text{“0x”}; \text{“foo”} \mapsto \text{“1z”} \rangle$. Vous utiliserez la fonction `prefix` pour effectuer cette tâche.

Le type suivant stocke de façon ordonnée des paires contenant un entier et une table de type `codes` :

```
typedef std::multimap<int, codes> partial_codes;
```

Implémentez la fonction `extract` qui prend un argument de type `partial_codes`. Elle supprime le plus petit élément et en renvoie une copie.

Implémentez la fonction `reduce` qui prend un argument de type `partial_codes` et le en supprimant les deux plus petits éléments et ajoute un nouvel élément à leur place via des appels à la fonction `extract` définie précédemment. Ce nouvel élément a pour membre de gauche la somme des membres de gauche des éléments supprimés et pour membre de droite le résultat de `merge` appliquée aux membres de droite des éléments supprimés.

Implémentez la fonction `create` qui prend un argument de type `occs` et renvoie une table de type `codes` obtenue de la façon suivante. On part d’une table de type `partial_codes` créée de telle sorte que, si l’argument passé à la fonction contient une paire $(\text{“foo”}, 42)$, alors la table créée contient une paire $(42, \langle \text{“foo”} \mapsto \text{“”} \rangle)$.

On applique ensuite la fonction `reduce` à cette table jusqu’à ce qu’elle ne contienne plus qu’une seule paire. Le membre de droite de cette paire est le résultat de la fonction `create`.

5 Compression de texte

Implémentez la fonction `compress` qui prend un vecteur de mots et une table de type `codes` en argument et crée un nouveau fichier qui contient les mots du vecteur remplacés par leur code. Retournez un string concaténant tout le contenu qui a été mis dans le fichier. Testez la fonction sur le contenu du fichier initial. Remarquez la taille du fichier obtenu.

Implémentez `lashorten` qui prend une table de type `codes` et en renvoie une autre dont les membres de droite peuvent utiliser d’autres caractères que ‘0’ et ‘1’. Testez à nouveau sur le fichier initial. Remarquez la taille du nouveau fichier obtenu.

Expliquez comment vous feriez pour décompresser le fichier obtenu (Écrivez votre réponse au début de votre fichier).