Structures de données et algorithmes

Projet de seconde session 2019-2020:
Résolution de problèmes

Pascal Fontaine – Jean-Michel Begon – Romain Mormont

17 juillet 2020

Description du problème

Soit une chaîne de caractères `s[1..n]` qui correspond à un texte corrompu dont on aurait enlevé tous les signes de ponctuation. Par exemple, en anglais :

"wheninthecourseofhumaneventsitbecomesnecessary"

On souhaite écrire une fonction permettant de séparer la chaîne de caractères en ses différents mots successifs. Le problème est rendu difficile notamment par le fait qu'une même chaîne peut être séparée de plusieurs manières possibles en une suite de mots corrects de la langue sans pour autant avoir du sens (la sous-chaîne "eventsitbecomes" pourrait par exemple correspondre à "event sit be comes" ou à "events it becomes"). Pour lever une partie de l'ambiguïté, on supposera qu'on dispose d'une structure de données contenant pour chaque mot de la langue ciblée un poids strictement positif proportionnel à sa fréquence d'apparition dans des textes de cette langue. Cette structure de données sera accessible par une fonction `weight(w)` prenant en entrée un mot `w` (une chaîne de caractère) et renvoyant soit `0.0` si le mot `w` ne se trouve pas dans la structure, soit le poids du mot s'il s'y trouve. La procédure de séparation des mots d'une chaîne cherchera alors à trouver la séparation maximisant le produit des poids des mots résultants (ou de manière équivalente la somme du logarithme de ces poids).

Plus formellement, soit $T[1 \dots L]$ le tableau contenant le texte à segmenter de longueur L. Il s'agit de déterminer le nombre de coupures de ce texte, noté l, et la position de ces l coupures, notées $c_1, c_2, \dots, c_l$, avec $1 < c_1 < c_2 < \dots < c_l \leq L$, qui maximisent la somme suivante :

$$\sum_{i=0}^{l} \log\left(\texttt{weight}(T[c_i \dots c_{i+1} - 1])\right), \tag{1}$$

où $c_0 = 1$ et $c_{l+1} = L + 1$.

Analyse

Avant d'implémenter votre solution, répondez aux questions suivantes dans votre rapport.

(a) En supposant que la structure de données utilisée pour stocker les mots et leurs poids ne devra pas être mise à jour une fois créée, discutez les différentes structures de données possibles pour implémenter `weight(w)` ? Précisez la complexité de `weight(w)` en fonction du nombre de mots dans le dictionnaire. Quelle structure vous apparaît comme étant la plus indiquée pour ce problème ?

(b) Proposez une approche gloutonne pour résoudre le problème et montrez par un contre-exemple qu'elle ne fonctionnera pas dans le cas général.

(c) Proposez une solution par programmation dynamique en passant par les étapes suivantes :
   (a) Soit `M[n]` le produit des poids des mots d'une séparation optimale de la chaîne `s[1..n]`. Formulez `M[n]` de façon récursive. Précisez le cas de base.
   (b) Donnez le pseudo-code d'une implémentation *efficace* permettant de calculer `M[n]`.
   (c) Analysez la complexité de votre solution en fonction de la taille `n` de la chaîne de caractères, dans le pire et le meilleur cas.

Implémentation

On demande de programmer un outil, qui prend sur son entrée standard une chaîne de caractères, et qui renvoie sur sa sortie standard la séparation des mots résultants (*cf.* `main.c`).

Vous devez implémenter les éléments suivants.

— Dans le cadre de ce projet, nous utiliserons une table de hachage comme structure permettant d'accéder au poids d'un mot. Celle-ci est à implémenter dans le fichier `hashtable.c` et doit répondre à l'interface définie dans le fichier `hashtable.h`.
Remarque : la fonction `getvalue` de l'interface est à mettre en lien avec la fonction `weight` de cet énoncé.
— L'implémentation de la solution par programmation dynamique est à rendre dans un fichier `dpseparate.c` qui répond à l'interface de `separate.h`.
— On vous demande également d'implémenter l'approche gloutonne dans `greedyseparate.c`, qui répond à l'interface de `separate.h`.

L'implémentation de la segmentation des mots se choisira au moment de la compilation.

Pour vous guider, nous vous fournissons les fichiers `parse.c` et `main.c`. Le premier permet de charger le fichier `words.csv` qui liste des mots de la langue anglaise avec le nombre d'apparitions dans un ensemble sélectionné de textes. Le second correspond à l'outil de séparation qui fait appel aux structures et fonctions que vous devez implémenter.

Remarques :

— Le poids d'un mot est le nombre d'apparition dans le corpus ci-dessus `words.csv`, divisé par la somme des nombres d'appariation des éléments dans le corpus entier.
— Utilisez le type `double` pour les poids.
— Attention aux dépassements de capacité.

Deadline et soumission

Le projet est à réaliser **individuellement** pour le **24 août 2020, 23h59** au plus tard. Aucun délai ne sera toléré. Il doit être soumis via la plateforme de soumission (`http://submit.montefiore.ulg.ac.be/`).

Le projet doit être rendu sous la forme d'une archive `tar.gz` contenant :

(a) votre rapport (4 pages maximum) au format PDF. Soyez bref mais précis. Le rapport doit contenir les pseudocodes et études de complexité pour les deux algorithmes de séparation.
(b) les fichiers `hashtable.c`, `dpseparate.c` et `greedyseparate.c`.

Vos fichiers seront compilés sur les machines ms8xx de la manière suivante (en substituant `greedyseparate.c` pour l'implémentation gloutonne) :

```
gcc main.c hashtable.c parse.c dpseparate.c --std=c99 --pedantic -Wall -Wextra -Wmissing-prototypes
-DNDEBUG -lm -o separate
```

Un projet non rendu à temps recevra automatiquement une cote nulle. En cas de plagiat avéré l'étudiant se verra affecter une cote nulle également. Les critères de correction sont précisés sur la page web des projets.

Bon travail !