langage C
TP 2-3_4-5 : Énoncé

Objectifs :

Le TP permet de démarrer un projet que vous devrez continuer en dehors des séances programmées. Ce projet réalisera, à l'aide de structures de données diverses, un lexique du vocabulaire employé dans un texte.

Le lien avec le module SIT152 est fait par les choix de structure de données et les algorithmes à utiliser et les commentaires que vous aurez à rédiger sur les complexités de vos algorithmes.

L'objectif est de maîtriser la programmation en langage C (même si tous les aspects ne peuvent être couverts par un seul projet) et les aspects algorithmiques et complexités associés à une problématique donnée.

Introduction :

Nous aimerions connaître les occurrences des mots employés dans un texte. Pour ce faire nous réaliserons un lexique contenant tous les mots du texte ainsi que leurs occurrences. Ce lexique réalisé, il nous serait ensuite possible de répondre à la question : combien de fois se retrouve le mot « commodity » dans ce texte.

N'hésitez pas à demander des éclaircissements à votre encadrant de TP.

1 Mise en place environnement.

Les fichiers à récupérer se trouvent sous **`~lib-src/FIP/SIT151/TP2-3_4-5`**

- Copiez le dans un répertoire vous appartenant.

2 Les textes à utiliser

Les textes qui serviront à l'évaluation des structures de données et algorithmes associés sont : fic0.txt, fic1.txt, fic2.txt, ...

Vous pouvez consulter le contenu de l'un d'entre eux en l'ouvrant avec un éditeur de texte.

exemple de texte (fic0.txt):

Karl Marx
Capital Volume One

Part One
Commodities and Money

SECTION ONE

THE TWO FACTORS OF A COMMODITY:
USE-VALUE AND VALUE
(THE SUBSTANCE OF VALUE AND THE MAGNITUDE OF VALUE)

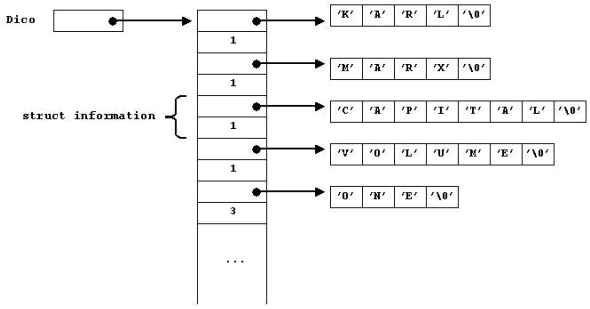
The wealth of those societies in which the capitalist mode of production
prevails, presents itself as "an immense accumulation of commodities," [1]
its unit being a single commodity. Our investigation must therefore begin
with the analysis of a commodity.

3 Rôle de la fonction lireMot

- Ouvrez `appli_initiale.c` à l'aide de jGrasp.
- Compilez et faites l'édition de lien de `appli_initiale.c`
- Exécutez le code exécutable obtenu. Attention il faut un paramètre pour l'exécutable : le nom d'un fichier texte.
- Remarquez le résultat obtenu, et comprenez le rôle de la fonction `lireMot`.

4 Création d'un lexique sous la forme d'un tableau.

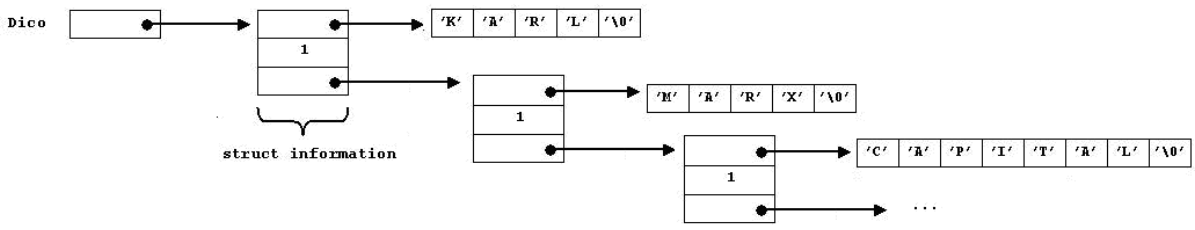
Une idée simple pour traiter le problème choisi (connaître pour tous les mots d'un texte, le nombre d'occurrences de ces mots) est de construire et utiliser une structure de données de type tableau d'informations. Une information est composée de deux champs, l'un est un pointeur sur le premier caractère du mot de l'information et l'autre est le nombre d'occurrences du mot dans le texte. Dans l'exemple qui suit le mot « ONE » a été trouvé 3 fois dans le texte, alors que les 4 autres mots ne s'y trouvent qu'une fois :



- **Mettez en œuvre cette idée. Pensez à décomposer fonctionnellement le problème (approche top-down).**
- **Validez en utilisant comme texte le fichier fic0.txt.**
- **Si fonctionnellement le résultat obtenu est satisfaisant, essayez de refaire un Dico en utilisant le texte du fichier fic9.txt.**

5 Création d'un lexique sous la forme d'une liste.

Une idée tout aussi simple pour traiter le problème choisi (connaître pour tous les mots d'un texte, le nombre d'occurrences de ces mots) est de construire une structure de données de type liste d'informations. Une information (⇔ à un élément de cette liste) est composée de trois champs, le premier est un pointeur sur le premier caractère du mot de l'information, le deuxième le nombre d'occurrences du mot dans le texte, le troisième un pointeur sur l'information suivante de la liste :



- **Mettez en œuvre cette idée.**
- **Validez en utilisant comme texte le fichier fic0.txt.**
- **Si fonctionnellement le résultat obtenu est satisfaisant, essayez de refaire un Dico en utilisant le texte du fichier fic9.txt.**

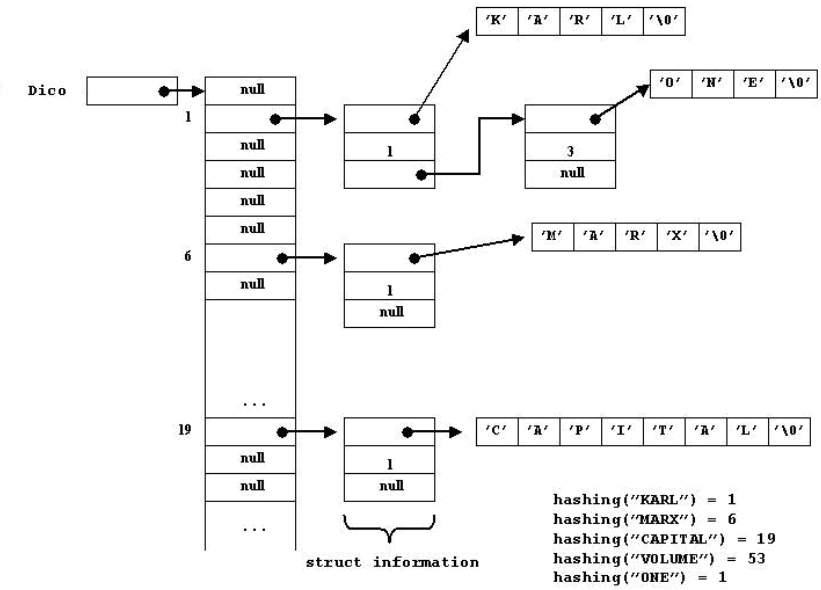
6 Analyse des deux réalisations précédentes.

Comparez les deux solutions précédentes… [complexité temporelle théorique/mesurée, complexité mémoire, …]

7 Création d'un lexique sous la forme d'une table de hashing.

Pour ne pas avoir à rechercher « longuement » si un mot est déjà présent dans le lexique, l'idée consiste à associer à chaque mot possible un index dans un tableau d'informations. Ceci est réalisé à l'aide d'une fonction (fonction de hashing) de l'ensemble des mots dans l'ensemble des indices possibles du tableau d'information (i.e. l'image de la fonction de hashing). Hélas cette fonction n'est généralement pas bijective et plusieurs mots peuvent avoir la même image par la fonction de hashing choisie.

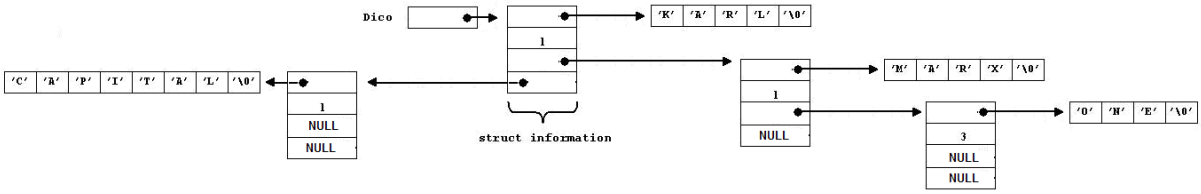
Comment faire s'il existe deux mots du vocabulaire du texte qui correspondent à la même valeur de hashing ? Une solution consiste à chaîner les informations correspondantes dans une liste. Le tableau d'information s'en trouve modifié: ce n'est plus un tableau d'informations, mais un tableau de liste d'informations. Une information (⇔ à un élément de ces listes) est composée de trois champs, le premier est un pointeur sur le premier caractère du mot de l'information, le deuxième le nombre d'occurrences du mot dans le texte, le troisième un pointeur sur l'information suivante de la liste.



- **Mettez en œuvre cette idée. (Déterminez une fonction de hashing)**
- **Validez en utilisant comme texte le fichier fic0.txt.**
- **Si fonctionnellement le résultat obtenu est satisfaisant, essayez de refaire un Dico en utilisant le texte du fichier fic9.txt.**
- **Refaites des tests avec différentes fonctions de hashing.**
- **Analysez et justifiez les résultats obtenus avec ces différentes fonctions de hashing.**

8 Création d'un lexique sous la forme d'un arbre binaire.

L'idée consiste à réaliser un arbre binaire ordonné d'informations. Une information (⇔ à un nœud de cet arbre) est composée de quatre champs, le premier est un pointeur sur le premier caractère du mot associé à ce nœud, le deuxième le nombre d'occurrences du mot dans le texte, le troisième un pointeur sur un arbre binaire ordonné d'informations dont les mots sont tous alphabétiquement inférieurs au mot de l'information courante, le quatrième un pointeur sur un arbre binaire ordonné d'informations dont les mots sont tous alphabétiquement supérieurs au mot de l'information courante.



- **Mettez en œuvre cette idée.**
- **Validez en utilisant comme texte le fichier fic0.txt.**
- **Si fonctionnellement le résultat obtenu est satisfaisant, essayez de refaire un Dico en utilisant le texte du fichier fic9.txt.**

9 Analyse de la complexité des différentes implémentations.

Mesurez les différents temps d'exécution de vos algorithmes en utilisant les fichiers fournis (fic0.txt, …, fic9.txt). Pour une mesure des temps d'exécution il est pratique d'utiliser la commande `time`. Faites `man time` pour comprendre comment l'utiliser.

Réalisez une représentation graphique des mesures obtenues. En abscisse vous aurez le nombre de mots traités et en ordonnée les temps d'exécution. Ces courbes doivent permettre de se faire une idée des performances relatives des différents algorithmes et de leurs complexités.

Votre analyse des complexités (constatées et théoriques et constatées) pourra être confortée par l'utilisation de textes à traiter différents de ceux fournis.