Dictionnaire (4 séances, à rendre)

1. Introduction

On désire implémenter un dictionnaire, c'est à dire une structure de données non ordonnée avec unicité. Pour simplifier le problème, les éléments à stocker seront des chaînes de caractères, mais tout type disposant d'un ordre lexicographique conviendrait. À chaque mot est associé une valeur (par exemple le même mot dans une autre langue, ou la nature grammaticale — nom, verbe, adjectif, etc)

Les méthodes à définir sont (le type Valeur est générique) :

```
bool contientMot(String mot)
// vrai ssi la chaîne mot figure dans le dictionnaire

void ajouterMot(String mot, Valeur v)
// ajoute la chaîne mot au dictionnaire, avec la valeur v,
// mot étant supposé absent du dictionnaire

void associerMot(String mot, Valeur v)
// associe la valeur v à la chaîne mot dans le dictionnaire,
// mot pouvant être présent ou absent du dictionnaire

void supprimerMot(String mot)
// supprime l'éventuelle chaîne mot du dictionnaire

Valeur valeurAssociée(String mot)
// donne la valeur correspondant à la chaîne mot
// (supposée figurer dans le dictionnaire)
```

2. Hashage

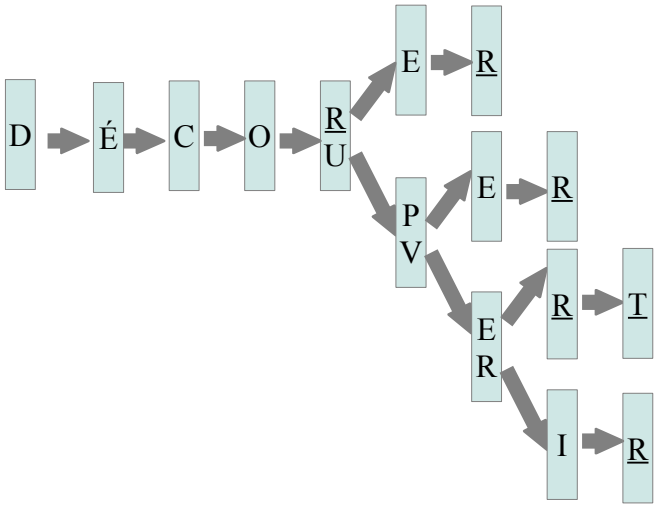
Implémenter un dictionnaire en utilisant une table de hashage. Calculer les coûts et l'encombrement mémoire en remarquant pour les comparaisons (equals) et pour la fonction de hashage que beaucoup de mots en Français ont un début commun (par exemple découvrir, découper, décorer, découvert, découver). Les coûts seront donc rédigés en fonction du nombre de mots stockés et de la longueur des mots.

3. Arbre des préfixes

Il est possible de gagner de l'encombrement mémoire en ne stockant qu'une fois les préfixes communs. On va donc construire une structure arborescente dans laquelle chaque cellule contient la « liste » des lettres possibles pour la suite du mot, et pour chaque lettre, une nouvelle cellule du même type.

Par exemple, avec les mots « décor, découvrir, découper, décorer, découvert, découver », le dictionnaire pourrait ressembler au schema à droite (les lettres soulignées correspondent à une fin de mot).

Chaque cellule comporte une suite de lettre, chacune associée à une nouvelle cellule. Proposer plusieurs implémentations de ce type selon la façon de stocker la suite de lettres (tableau statique, A-liste, table de hashage) en discutant des avantages et inconvénients de chacune. En implémenter effectivement une.



4. Utilisation

En utilisant un dictionnaire, écrire un programme principal affichant les dix mots les plus fréquents dans un texte (donné par le nom du fichier qui le contient) avec leurs fréquences. Ce programme doit compiler indifféremment avec les deux implémentations (et fournir des résultats similaires).

5. Rendu

Un document de travail doit être rédigé **au préalable et en parallèle**, précisant les ambiguïtés du sujet. Ce document présentera aussi les choix effectués (constantes, comportement des méthodes non précisé dans l'énoncé, …) et les limites d'utilisation. Il est aussi, si ce n'est plus important, que le code. Il devra comporter un comparatif de vos deux implémentations en terme de complexité temporelle (l'ordre de grandeur du coût de chaque méthode doit être renseigné) et d'encombrement mémoire.

Le travail est à déposer sur MADOC au plus tard le 25 avril 2014 à 23h55 sous forme d'une archive zip contenant le rapport (pdf) et le code.

Le code doit être clair et concis, commenté, avec en particulier les pré- et post-conditions. Le rapport doit être bien écrit (français correct, attention à l'orthographe!) et agréable à lire, il doit comporter introduction et conclusion et des annexes pour les détails techniques (graphiques, code, etc.)

Vous pouvez proposer quelques méthodes supplémentaires si leur coûts sont très différents d'une implémentation à l'autre (mais impérativement les mêmes méthodes dans toutes les implémentations). Vous pouvez les implémenter ou seulement les évoquer dans le rapport, avec leurs coûts.