Arbre de complétion

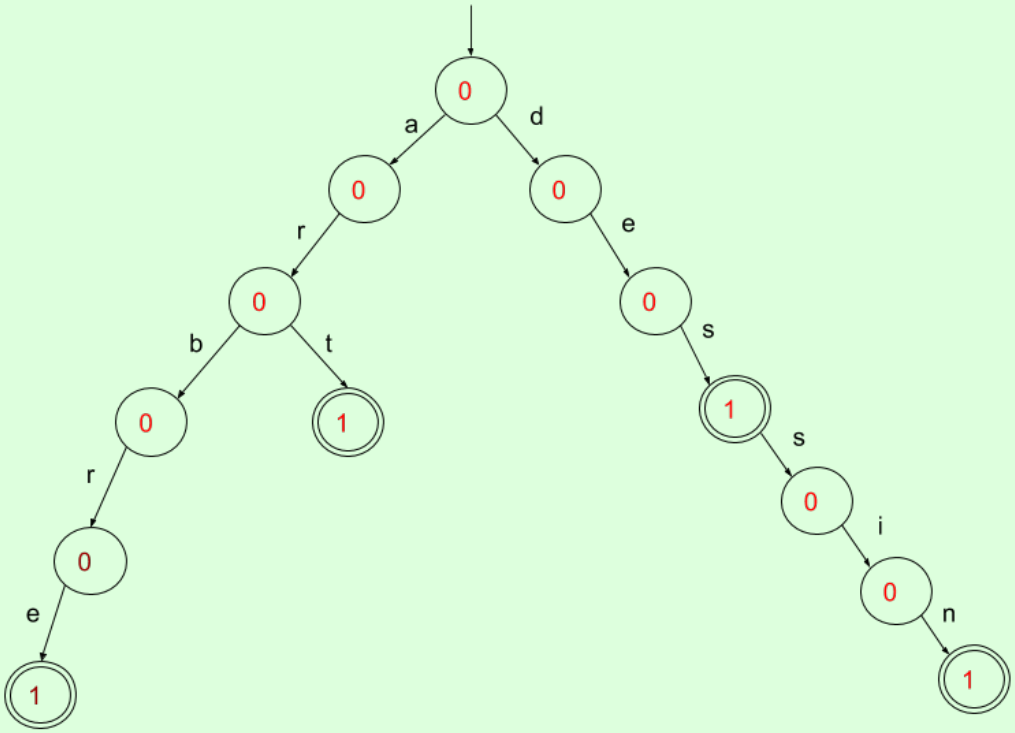
Un algorithme de complétion est un mécanisme logique permettant d'anticiper la saisie et de proposer des mots automatiquement pour faciliter les recherches dans un formulaire sur une page web par exemple.

On utilise pour cela une structure de données arborescente, où chaque nœud de l'arbre est une étape de lecture et chaque arête correspond à la lecture d'une lettre. Les nœuds sont indexés par les lettres suivantes possibles du mot, avec un compteur par nœud pour savoir si celui-ci est final ou non (le nœud est final si le compteur est >0).

Le but de cet exercice est de construire un arbre de complétion à partir de mots de vocabulaire, puis de l'utiliser pour compléter un début de mot proposé par l'utilisateur.

Vous devez implémenter un arbre de complétion en Java pour gérer une base de mots et répondre à différentes requêtes, telles que l'insertion de mots, l'affichage de l'arbre, la suggestion de mots complets à partir d'un préfixe donné, etc.

V = {art, arbre, des, dessin}



1. Définition des Classes

a. Définissez une classe Noeud avec les attributs suivants :

1. `terminal` : un booléen indiquant si le nœud représente la fin d'un mot.
2. `compteur` : un entier indiquant le nombre d'occurrences du mot.
3. `aretes` : une liste d'objets de type `Arete`.

b. Définissez une classe `Arete` avec les attributs suivants :

1. `caractere` : un caractère représentant la relation entre deux nœuds.
2. `noeud` : un objet de type `Noeud` représentant le nœud suivant.

c. Pour initialiser un arbre de complétion, créer un nœud vide dont le `compteur` vaut 0 et dont la liste des arêtes est une liste (`ArrayList`) vide.

2. Fonction d'Insertion

Écrivez une méthode `insererMot` dans la classe `Noeud` pour insérer un mot dans l'arbre de complétion.

- La fonction est récursive
- Elle prend comme argument une chaîne de caractères

- **Cas terminal** :

Si le mot est vide, cela signifie que tous les caractères du mot ont été insérés. Dans ce cas, on met à jour l'attribut estTerminal du nœud actuel à true (pour indiquer que c'est la fin d'un mot) et on incrémente le compteur associé.

- **Cas Récursif** :

On prend le premier caractère du mot (la première lettre) et on cherche une arête dans le nœud actuel qui a ce caractère.

- S'il existe une telle arête, on appelle récursivement `insererMot` avec le nœud correspondant à cette arête et le reste du mot (tous les caractères sauf le premier).
- Si l'arête n'existe pas, on crée une nouvelle arête avec ce caractère et un nouveau nœud associé. Ensuite, on appelle récursivement `insererMot` avec le nouveau nœud et le reste du mot.

Extraire le Premier Caractère : Utilisez la méthode `charAt(index)` de la classe `String` pour obtenir le caractère à l'index spécifié.

Exemple :

```
String mot = "bonjour";
char premierCaractere = mot.charAt(0);
```

Obtenir la Chaîne Initiale Privée du Premier Caractère : Utilisez la méthode `substring(beginIndex)` de la classe `String` pour obtenir la sous-chaîne à partir

de l'index spécifié.

Exemple

```
String mot = "bonjour";
String resteDuMot = mot.substring(1);
```

3. Fonction d'Affichage

Écrivez une méthode `affiche` dans la classe `Noeud` pour afficher la totalité de l'arbre de complétion. La fonction doit respecter les conditions suivantes :

- Elle doit parcourir l'arbre de manière récursive.
- Elle doit afficher chaque mot complet trouvé ainsi que son compteur.

4. Fonction d'Appartenance

a. Écrivez une méthode `appartient` dans la classe `Noeud` qui retourne `true` si le mot est présent dans l'arbre de complétion et `false` sinon. La fonction doit respecter les conditions suivantes :

- Vous pouvez au choix utiliser une approche itérative ou récursive pour parcourir l'arbre.
- La fonction doit retourner `true` uniquement si le mot complet est présent dans l'arbre (c'est à dire si la lecture de la dernière lettre conduit à un noeud terminal).

5. Fonction de Suggestion

Écrivez une méthode `suivant` dans la classe `Noeud` pour retourner une liste de mots complets à partir d'un préfixe donné. La fonction doit respecter les conditions suivantes :

- Elle doit utiliser une approche itérative de parcours de l'arbre jusqu'au nœud correspondant au préfixe.
- Elle doit faire appel à la fonction d'affichage du noeud atteint pour afficher l'ensemble des noeuds suivants

6. Tests

a. Créez un arbre de complétion vide.

b. Insérez les mots `"chat"`, `"chien"`, `"chapeau"`, `"bonjour"`, `"bonsoir"`, `"bon"`, `"jour"`, `"soir"` dans l'arbre de complétion.

c. Affichez l'arbre pour vérifier que les mots sont correctement insérés.

d. Testez la fonction `suivant` avec le préfixe `"bon"` et affichez les résultats.

e. Testez la fonction `appartient` avec les mots `"chat"`, `"maison"`, et `"bon"`, affichez le résultat pour chaque test.

—

From:
https://wiki.centrale-med.fr/informatique/ - **WiKi informatique**

Permanent link:
https://wiki.centrale-med.fr/informatique/tc_info:2023-tp-texte-corr

Last update: **2023/12/04 09:47**