Plus fort que papy au mot le plus long

Le mot le plus long est un jeu télévisé populaire et ininterrompu depuis le 19 septembre 1965 (https://www.youtube.com/watch?v=u2UVm1Dcms8). Ce jeu a pour but de construire les mots les plus longs à partir d'un tirage aléatoire de 7 lettres.

Le but de cette activité est d'établir un programme capable de générer la liste des mots les plus longs selon un tirage.

1. Construction d'un dictionnaire de référence.

Pour établir des mots à partir d'un tirage, le programme a besoin d'un dictionnaire de référence contenant l'ensemble des mots disponibles. Cet ensemble de mot étant conséquent, il est nécessaire d'optimisé le temps de recherche stockant l'ensemble des mots dans un **trie** (arbre préfixe)

Arbre préfixe

Un trie ou arbre préfixe est un arbre numérique ordonné qui est utilisé pour stocker une table associative où les clés sont généralement des chaînes de caractères. Contrairement à un arbre binaire de recherche, aucun nœud dans le trie ne stocke la chaîne à laquelle il est associé. C'est la position du nœud dans l'arbre qui détermine la chaîne correspondante.

Pour tout nœud, ses descendants ont en commun le même préfixe. La racine est associée à la chaîne vide. Des valeurs ne sont pas attribuées à chaque nœud, mais uniquement aux feuilles et à certains nœuds internes se trouvant à une position qui désigne l'intégralité d'une chaîne correspondant à une clé.

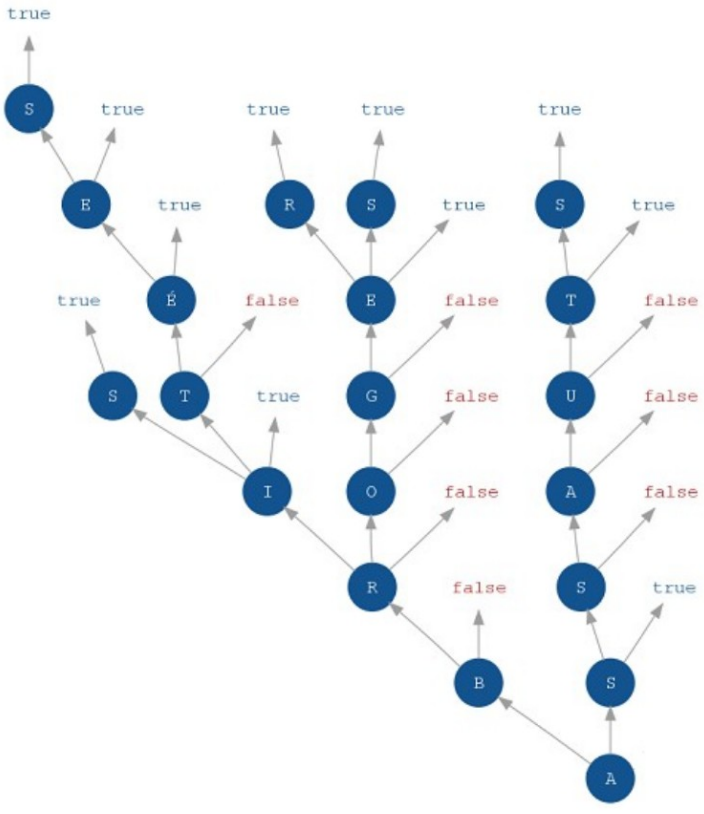

Figure 1: Trie construit avec les mots *abri, abris, abrité, abritée, abritées, abroge, abroger, abroges, as, assaut et assauts.*

Les logiciels de traitement de texte présentent souvent une fonctionnalité d'auto-complétion des mots au cours de la frappe. Certains éditeurs de texte l'étendent aux commandes d'un langage de programmation. Dans tous les cas, l'auto-complétion est grandement facilitée par la représentation des mots sous forme de trie : il suffit en effet d'extraire le sous-arbre correspondant à un préfixe donné pour connaître toutes les suggestions en rapport avec le texte saisi.

Un trie en Python

Une classe incomplète arbreTrie() est disponible sur Moodle dans le fichier code.zip (.\code\trie.py)

Q1. **Compléter** la classe arbreTrie() afin que les méthodes incomplètes répondent à leur spécification

Un module runtest sur .\code\runtest.py . Ce module permet de tester l'ensemble des méthodes de la classe arbreTrie()

Q2. **Tester** et **valider** le fonctionnement de la classe arbreTrie()

Utilisation d'un trie comme dictionnaire

On dispose d'une base de données qui contient 137 800 mots français entre 2 et 16 lettres, sans tiret, ni espace ni apostrophe.

L'accès à cette base s'effectue avec les paramètres suivants :

- ✗ Adresse IP : **'141.94.247.234'**
- ✗ Port : **3306**
- ✗ Utilisateur : **'TNSI'**
- ✗ Mot de passe : **'eleve_NSI'**
- ✗ Nom de la base : **'DICTIONNAIRE'**

Cette base de données DICTIONNAIRE est constituée de deux tables :

lexico qui contient l'ensemble des mots du dictionnaire, ses attributs sont :

- ✗ **id** : identifiant unique à usage interne
- ✗ **mot** : ensemble des mots du dictionnaire. Cet ensemble contient des verbes conjugués ainsi que les noms et adjectifs dans leur différentes formes (genre et pluriel).
- ✗ **lemme** : indique l'id de la racine du mot. Par exemple, l'entrée du verbe conjugué **chantait** (attribut mot) indique dans l'attribut lemme, l'id du mot **chanter** qui correspond à la forme à l'infinitif du mot.
- ✗ **cgram** : clé étrangère associée à la clé primaire de la table gram. Cet attribut indique la nature du mot (nom, verbe ...)
- ✗ **freq** : indique la fréquence d'apparition des mots dans un ensemble d'œuvres littéraires.

gram qui modélise la nature des mots :

- ✗ **id_cg** : identifiant unique
- ✗ **cgram** : nature des mots (NOM, AUX : auxiliaire, VER : verbe, ADJ, adjectif ...).

Q3. **Créer** un fichier Python lexico.py. puis **établir** un programme qui demande la saisie d'un mot français et indique s'il est présent ou non dans le dictionnaire réduit à l'ensemble des mots sans les verbes conjugués.

Remarque : Utiliser bien évidemment la classe arbreTri() dans cette recherche.

Constater que la création du trie prend un peu de temps (quelques secondes). Par contre, une fois le trie créé, la réponse à la question est instantanée.

2.Construction de l'algorithme de construction de mots.

But de cette partie : Etablir l'ensemble des combinaisons de mots de 2 à 7 lettres à partir d'un tirage de 7 lettres.

La recherche de l'ensemble des mots possibles peut être réalisé avec un algorithme de backtracking identique à celui étudié lors de l'activité sur le jeu du morpion.

Le tirage des lettres est une donnée composite, deux solutions pour stocker ce tirage sont possibles. Le stockage peut être fait dans une liste ou dans une chaîne de caractères.

Q4. **Justifier** que dans l'algorithme de backtracking, le stockage dans une liste permet d'effectuer la recherche dans un espace mémoire constant contrairement à la chaîne de caractère qui réclamera davantage d'espace.

La recherche de mots valides à partir d'un tirage sera programmée dans une classe mlpl. Le prototype de cette classe est disponible dans le fichier mlpl.py

Q5. **Définir** des jeux de test dans les spécifications des méthodes.

Q6. **Programmer** la recherche de mots valides et **tester** au fur et à mesure chaque méthode.

3.Synthèse

Q7. Par groupe de 2, **préparer** un oral de présentation de ce projet

Format et évaluation de l'oral

Temps de présentation : 6 minutes (3 minutes / élève) + 4 minutes de questions. La deuxième partie sera composée de deux questions (1 par candidat) qui porteront sur les deux parties

Support de présentation : un diaporama de présentation de 6 pages maxi sera présenté. Ce diaporama fera apparaître les structures de données et les principes algorithmiques mise en œuvre. Aucune ligne de code ne doit être présenté durant cet oral.

Critère d'évaluation :

- **Oral de présentation :**
  - ➔ Qualité du diaporama
  - ➔ Qualité et précision de l'oral de présentation
  - ➔ Qualité et précision du vocabulaire utilisé
  - ➔ Précision et justesse de la réponse à la question.
- **Codage** :
  - ➔ Fonctionnement des classes
  - ➔ Nom des variables utilisées
  - ➔ Jeux de tests définis
  - ➔ Clarté et lisibilité du code