

Exercice 1 : Concepts

(a) Un trie (arbre de préfixes) est surtout adapté pour :

- A. Rechercher rapidement des mots par préfixe
- B. Trier des entiers en place
- C. Calculer des distances euclidiennes
- D. Compresser des tableaux de nombres réels

(b) Considérez un trie contenant les mots: **chat**, **chien**, **chou**, **choix**. Quel est le plus long préfixe commun à tous ces mots? (a) _____

.....

(c) Donnez la complexité en temps (Big-O) d'une recherche d'un mot de longueur L dans un trie, en supposant un alphabet de taille fixe.

.....

Exercice 2 : Définition de structures

(a) Proposez une structure C pour un noeud de trie contenant :
— un tableau de pointeurs vers enfants (alphabet ASCII minuscule a..z),
— un indicateur `is_word` qui vaut 1 si le noeud termine un mot.

.....
.....
.....
.....
.....
.....
.....
.....
.....

Exercice 3 : Programmation

(a) Écrire une fonction `trie_new` qui alloue un noeud vide et initialise tous les pointeurs enfants à NULL et `is_word` à 0.


```
int trie_search(const TrieNode *root, const char *word);
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

(d) Écrire une fonction `trie_starts_with` qui retourne 1 si un préfixe existe dans le trie, 0 sinon. Prototype :

```
int trie_starts_with(const TrieNode *root, const char *prefix);
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Exercice 4: Réflexion

(a) Expliquez un inconvénient mémoire des tries, et donnez une amélioration possible.

.....

.....

.....