Franck Caron, Pierre Dechamps
M1 GIL

Projet d'Algorithmique du Texte

Introduction

Ce document a pour but de décrire synthétiquement les structures et les test réalisés qui nous ont permis de produire le contenu du projet.

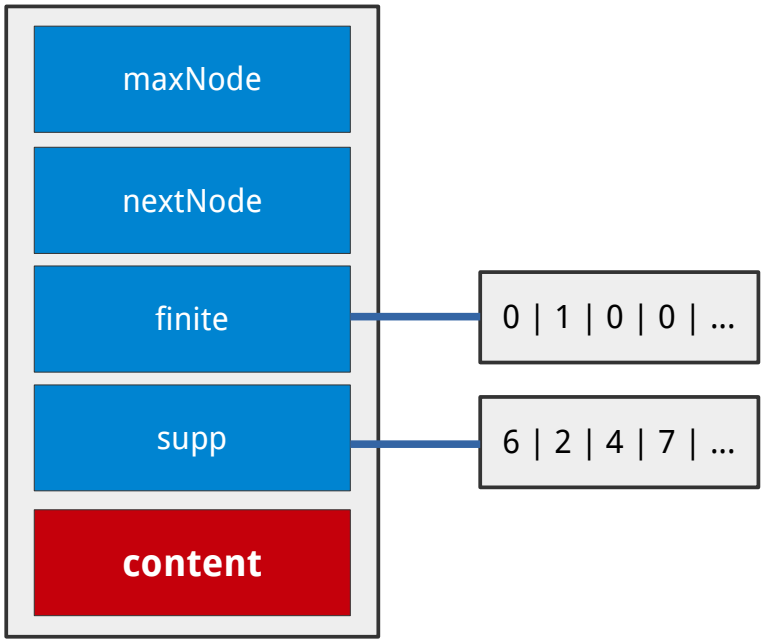
Ce dernier utilise les différentes structures employées lors du TP, afin d'implanter l'algorithme d'Aho-Corasick, pour la recherche d'un ensemble fini de mots dans un texte donné.

Le projet se décompose donc en 5 sous - programmes:

- ac-matrice : qui implante l'algorithme avec une structure matricielle.
- ac-liste : qui l'implante avec une structure basée sur des listes.
- ac-mixte : qui l'implante avec une structure mixte, basée sur les 2 précédentes.
- genere-mots : qui permet de construire des listes de mots aléatoires à tester.
- genere-texte : qui permet de créer des textes dans lesquels effectuer les recherches.

Mise en place de l'algorithme

Nous avons défini 3 structures de trie, selon les indications données en séances de TP. Structurellement, ces tries se schématisent globalement de la manière suivante :



Attributs :

- maxNode (int) : La capacité du trie, en nombre de nœuds.
- nextNode (int) : Le numéro du prochain état ajouté.
- finite (char *) : Le tableau indiquant si un état représente la fin d'un mot.
- supp (int *) : L'état suppléant associé à cet état, pour l'exécution de l'algorithme d'Aho-Corasick

Le conteneur des transitions associé à ce trie (content) à un type qui dépend du type de trie utilisé :

- Trie list : Une liste de cellules (état/ lettre).
- Trie matrice : Un tableau à deux dimensions ([états][lettre]) pour le trie matrice.
- Trie mixte : Un tableau de transitions pour le nœud racine et un tableau de listes pour les autres états.

Comme imposé par le sujet, nous n'avons pas effectué d'autres modifications sur ces structures, mis à part l'ajout du tableau d'entiers supp, afin de conserver l'état suppléant d'un état donné.

Chaque trie dispose de fonctions d'insertion, ce création et d'existance (réalisées durant le TP1), auxquelles nous avons ajouté les fonction prepareAC et executeAC.
La fonction prepareAC(trie) permet de préparer l'exécution de l'algorithme en mettant à jour le tableau des suppléants du trie.
La fonction executeAC(trie, q, l) permet de réaliser l'étape de lecture de la lettre l depuis l'état q, afin de mettre à jour la valeur de l'état (après lecture), et d'indiquer le nombre de mots trouvés lors de cette étape.

Grâce à ces fonctions, le flux d'exécution du programme principal est relativement simple :

1. On initialise un nouveau trie*.
2. On lit le fichier dictionnaire fourni en paramètre et on en insère chaque mot dans le trie.
3. On lit le texte caractère par caractère. Pour chacun d'entre eux, on exécute l'algorithme depuis l'état actuel (en employant executeAC) et on ajoute le nombre de mots trouvés au compteur.
4. On renvoie le nombre de mots comptés.

(* Note : La taille du trie utilisée est fixée par 2 constantes permettant de définir le nombre de mots à traiter (MAX_WORD_COUNT), et la taille maximale de ces mots (MAX_WORD_SIZE). Si vous souhaitez utiliser le programme des tests plus « coûteux » que ceux figurant dans le sujet, Il faudra les modifier manuellement (la taille par défaut est de 6400 états.)).

Génération de mots et de texte

Les procédures de génération s'effectuent au moyen de la structure charbuffer. (Définie dans charbuffer.h et charbuffer.c).

Ces structures permettent d'initialiser des chaînes de caractères d'une certaine taille (ne pouvant dépasser une certaine capacité maximale fixée à la création du buffer), dont le contenu est encodé sur un alphabet donné.

Le but de cette structure réside dans sa fonction randomize_buffer(buffer, size), qui modifie de manière aléatoire les size premiers octets présent dans le buffer. C'est cette fonction qui permettra la création du contenu aléatoire des mots et des textes.

Les programmes genere-mots et genere-texte font alors appel à ce module afin d'en produire le résultat attendu.

Tests d'utilisation

Comme demandé, nous avons exécuté les 3 différents programmes sur des ensembles de mots avec des alphabets d'une certaine taille. Pour chaque taille d'alphabet utilisé, la taille du texte de test était fixée à 5M de caractères.

Voici nos résultats :

ac-matrice

Mots\Alphabet	2	4	20	70
5 ... 15	0.270s	0.170s	0.115s	0.102s
15 ... 30	0.270s	0.169s	0.115s	0.111s
30 ... 60	0.287s	0.175s	0.118s	0.107s

(Matériel de test : Processeur i7 4810-MQ)

ac-liste

Mots\Alphabet	2	4	20	70
5 ... 15	0.305s	0.290s	0.620s	1.841s
15 ... 30	0.313s	0.289s	0.627s	1.805s
30 ... 60	0.322s	0.306s	0.644s	1.867s

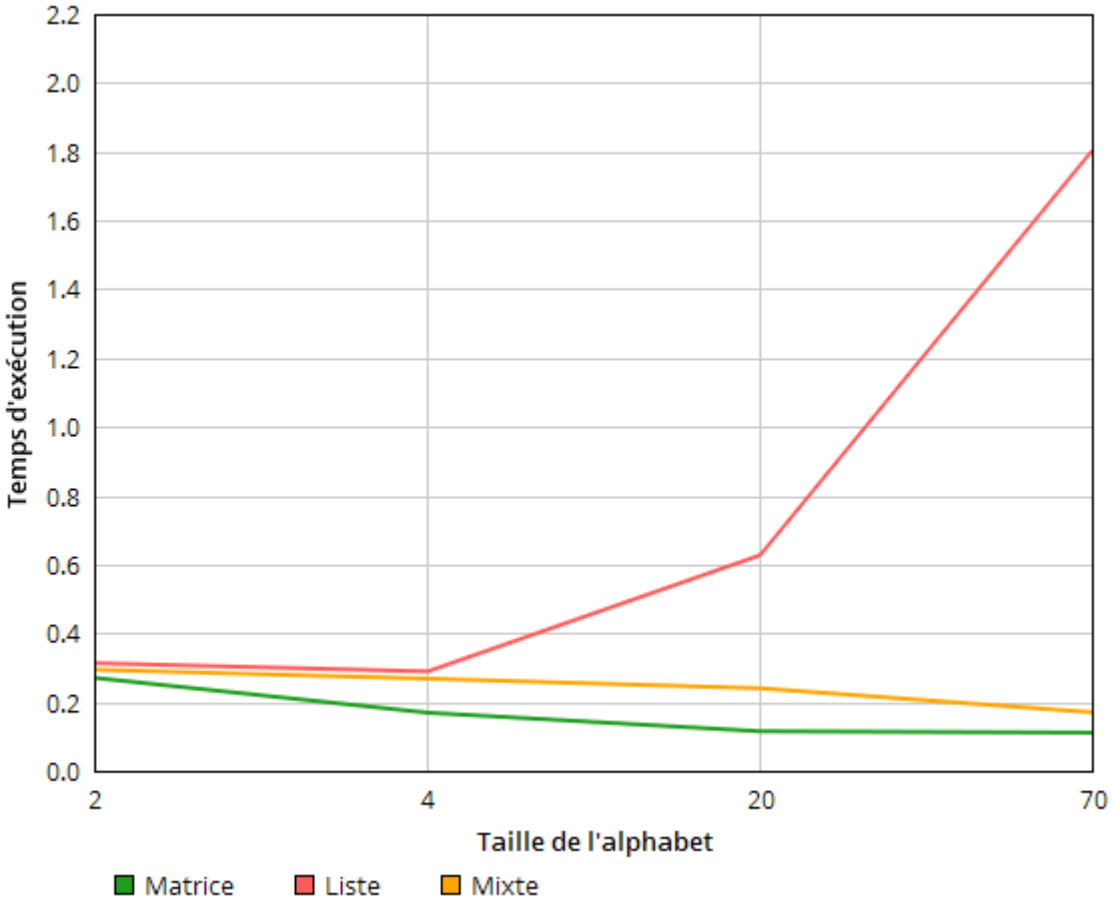
(Matériel de test : Processeur i7 4810-MQ)

ac-mixte

Mots\Alphabet	2	4	20	70
5 ... 15	0.296s	0.265s	0.244s	0.166s
15 ... 30	0.294s	0.268s	0.240s	0.170s
30 ... 60	0.317s	0.285s	0.252s	0.159s

(Matériel de test : Processeur i7 4810-MQ)

Observations :



Cette courbe représente la courbe des valeurs moyennes des temps d'exécution pour chaque taille d'alphabet donné.

Un classement sur la rapidité des structures est trivial : la matrice est plus rapide que la structure mixte, qui est elle-même plus rapide que la liste.

On peut constater que le temps d'exécution augmente évolue de manière quasi-linéaire avec la taille de l'alphabet, pour l'utilisation du trie liste.
Les tries mixtes et matrices sont quant à eux plus efficaces sur des alphabets de plus grande taille.

De ce fait, il est recommandé de préférer le trie mixte au trie liste, dans tous les cas (du fait qu'il n'y ait pas de complexité en espace supérieure entre les deux).

En revanche, au vu des ressources mémorielles élevées requises par le trie matrice. Il ne doit être utilisé que si la vitesse de calcul est prioritaire sur la consommation de mémoire.