ALLIO
BARRY

DEVOIRS N°2 – COMPTE RENDU

Résumé de l'exercice :

A partir de l'algorithme d'AhoCorassick étudié en cours, nous devons en adapter les différentes étapes en code. On relève 3 étapes importantes : la construction d'un graphe contenant les mots à rechercher, la création de transition d'échec de façon à optimiser au maximum le parcours du graphe et le parcours simultané du graphe et du texte (dans lequel on recherche) afin de mettre en évidence les mots clés trouvés.

Les difficultés :

Le langage

Etant donné la non trivialité de l'algorithme et le fait qu'une interface graphique soit très utile pour présenter les résultats, nous avons décidé de faire le projet en Java.

Insérer les mots dans un graphe

AhoCorassick
|-Tree
|-State

Nous avons utilisé cette hiérarchie pour représenter notre graphe

- State : contiens des fonctions pour créer des états et changer leur attribut (final, état d'échec…)
- Tree : contiens des fonctions pour créer des graphes et les parcourir à partir de mots. On y trouve aussi la fonction qui construit les transitions d'échecs.
- AhoCorassick contient des fonctions globales (ajout d'un tableau de mot, recherche de l'état suivant en prenant compte du cas d'échec…) qui utilisent des méthodes de Tree.

Transitions d'échecs

Cette fonction a été la plus dure à concevoir à cause des nombreuses subtilités qu'elle impose. L'algorithme consiste à prendre un suffixe d'un mot clé et d'essayer de retrouver ce suffixe parmi le préfixe d'autres mots clés.

Prenons un exemple :

Mots clés
T O R T U E
T U E U R
U R N E

(1)
T O R T U E
T U E U R
U R N E

(2)
T O R T U E
T U E U R
U R N E

(3)
T O R T U E
T U E U R
U R N E

(4)
T O R T U E
T U E U R
U R N E

(5)
T O R T U E
T U E U R
U R N E

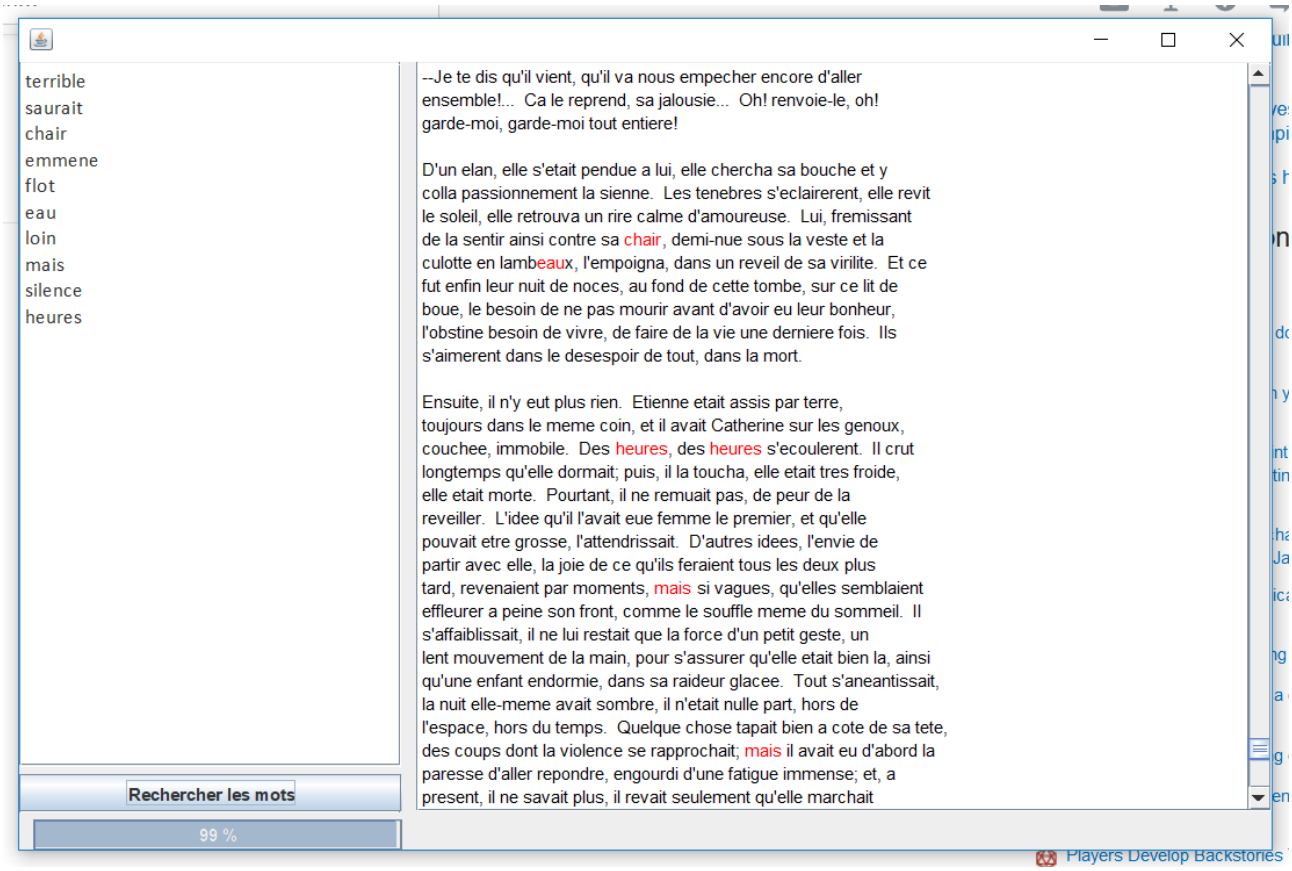
…

Ici, au bout de la 3ème itération, une transition d'échec se crée entre le 2ème T de tortue et le T de tueur. Ainsi de suite pour le U et le E. En répétant pour les autres mots clés, on s'aperçoit qu'on peut relier les 2 dernières lettres U R de tueur aux lettres U R de urne.

Dans le code, on utilise 2 variables : une pour représenter le curseur orange (suffixe) et l'autre pour représenter le curseur vert (préfixe). On parcourt l'ensemble des mots clés et l'ensemble de leur lettre en partant de la seconde (puisqu'il s'agit sinon d'un préfixe par définition).

- Si en lisant le suffixe, on ne détecte pas de préfixe correspondant dans la liste des mots clés, pas la peine de tester plus loin donc. On va donc réduire la taille du suffixe à rechercher en partant d'un caractère plus loin.
- Si on détecte un préfixe correspondant, on peut relier l'état actuel avec l'état du préfixe correspondant grâce à une fonction prévue pour cela

La représentation / Résultat obtenu



Résultats de recherche de quelques mots clés sur le texte de « Germinal » transmit avec le devoir (temps de recherche=environ 2 sec).

On utilise la librairie graphique de base de Java.

AOForm.Java contient à la fois le code pour faire apparaître l'interface mais aussi l'algorithme de recherche qui utilise les méthodes de AhoCorassick.java.

On extrait les mots clés de la partie gauche (une ligne=1 mot clé). On construit un graphe avec les éventuelles transitions d'échec.

Puis on parcourt tout le texte de la zone de droite caractère par caractère. On a implémenté une fonction qui nous dit à quel état on en est par rapport à l'état précédent et le caractère actuellement lu. Il suffit donc de 2 variables pour la mise en forme : l'une pour connaître la position de départ (start) et l'autre la position d'arrivée (end).

Dès que l'on sort de l'état 0, on est forcément en train de trouver au moins un caractère, donc on peut initialiser start à partir de ce caractère. Et end sera défini quand on aura trouvé un état final.;