Rapport Projet offline

Ivan Delgado
Elisabeth Abbas Zadeb

13 octobre 2019

Première partie
Introduction

Ce document est le Rapport du projet *offline* dans le cadre de l'UE DARR.

Le but de ce projet est de réaliser la partie *offline* d'un moteur de recherche ; a savoir la recherche d'un motif sous le format RegEx dans un texte issu du Projet Guttemberg.

Pour cela nous allons faire une copie de la commande *egrep* en nous restreignant pour les éléments de RegEx aux parenthèses, l'alternative, la concaténation, l'opération étoile, le point , et les lettre ASCII.

Deuxième partie
Mode d'emplois

Le code du projet a été réalisé en Java, il est fourni avec un fichier *build.xml* pour *apache ant* qui permet les commandes suivantes :

- *ant compile*
  pour compiler le projet
- *ant run -Darg0="<motif a rechercher>" -Darg1="<fichier texte>*
  pour lancer la recherche du motif sur un texte
- *ant clean*
  pour nettoyer le répertoire du projet

Troisième partie
Stratégies implémentés

Dans un premier temps nous considérons que le texte est une matrice, c'est un tableau de lignes qui chacune contient un tableau de caractères, pour déter-

miner la position d'un match il suffi donc d'un numéro de ligne et d'une position dans la ligne, cette information est dans la classe *TextPosition.java*.

1 Recherche par Automates

Le traitement d'un regex passe par différentes opérations :

1. Dans un premier temps nous allons parser le regex pour en construire un arbre de syntaxe.
2. Construire a partir de l'arbre construire un automate fini non-déterministe avec-transitions selon la méthode Aho-Ullman [1].
3. Déterminiser l'automate en enlevant les *epsilon* transitions avec la méthode des sous-ensembles.
4. Minimiser l'automate en construisant un automate équivalent avec un nombre minimum d' états.
5. Enfin l'automate est utilisé pour tester si un suffixe d'une ligne du fichier textuel donné initialement est reconnaissable par cet automate.

La structure de données représentant un automate est semblable a celle de la liste d'adjacence d'un graphe, il y a une ligne par état, chacune contenant un tableau de 256 entiers, un par caractère ASCII, la valeur de la case est l'état résultant de la transition par ce caractère, la valeur est *null* s'il n'y a pas de transition.

On ajoute a cela trois tableau, un pour savoir les états de départ, un pour les états acceptés et un pour les *epsilon* transitions.

Le code de la structure de donnée, ainsi que des algorithmes de détermination et de minimisations se trouvent dans le fichier Automate.java.

2 Algorithme de Knuth-Morris-Prat

Si le RegEx recherché n'est composé que de concaténations, il est alors superflu de d'en construire un automate, une représentation en chaîne de caractères suffit.

Nous pouvons alors utiliser l'Algorithme de Knuth-Morris-Pratt (KMP) qui est un algorithme de recherche de sous-chaîne, permettant de trouver les occurrences d'une chaîne P dans un texte S avec une complexité linéaire $O(|P|+|S|)$. KMP se base sur un prétraitement de P qui donne un tableau d'indices appelé retenue qui donne la position de la chaîne ou reprendre les comparaisons en cas de non-concordance, permettant de ne pas avoir a ré-examiner les répétitions de caractères et de préfixes, limitant ainsi les comparaison nécessaires.

KMP est un algorithme simple ne nécessitant pas de structure de données particulière, son code se trouve dans le fichier *KMP.java* .

1. http ://infolab.stanford.edu/~ullman/focs/ch10.pdf

3 Index et Radix Tree

Un texte peut être décomposé en une liste de mots (une suite de caractères uniquement alphabétiques), avec *awk* nous pouvons facilement créer une liste des mots uniques d'un texte dans un fichier *.txt* dans le répertoire *index,* puis en utilisant l'algorithme KMP sur chacun de ces mots nous pouvons trouver la position de l'ensemble des occurrences, qui triées par ordre décroissant d'occurrences, serons mises dans un fichier *.index ; cette suite d'opérations est coûteuse mais elle n'est a faire qu'une fois par fichiers.*

Ensuite avec cet index nous pouvons en construire un *radix tree* ce qui permet de dire rapidement si le mot recherché est dans le texte.

L'arbre contient pour chaque nœud, une liste positions si le nœud correspond a un mot de l'index, et une liste des fils pour la lettre suivante correspondante. Si il n'y a plus de lettre les positions retournées serons celle du noeud et de ses fils récursivement.

L'indexation se fait dans *Indexing.java* et le *radix tree* est défini dans *IndexTree.java.*

La stratégie que nous allons appliquer est la suivante :

Si le motif est un mot, nous allons le chercher avec la méthode de l'index, si il n'y a pas de résultats ou si le motif n'as que des concaténations on utilise KMP.

Si c'est un regex nous utilisons la méthode de l'automate.

Quatrième partie
Test et Analyse

A History of Babylon (124 304 mots) :

- Recherche de « Sargon » :
  temps par egrep 15 ms
  temps par index :
  - construction de l'index : 37 s
  - chargement de l'arbre 300 ms
  - recherche dans l'arbre : <1µs
- Recherche de « S(a|g|r)*on » :
  temps par egrep : 18 ms
  temps par l'automate : 161 ms
- Recherche de « argon »
  temps egrep : 23 ms
  temps KMP : 40 ms

Dracula (164 424 mots) :

- Recherche de « Dracula » :
  temps par egrep 21 ms
  temps par index :
  - construction de l'index : 44 s
  - chargement de l'arbre 500 ms
  - recherche dans l'arbre : <1µs
- Recherche de « D(r|a|c)*ula » :
  temps par egrep : 10 ms
  temps par l'automate : 125 ms
- Recherche de « racula »
  temps egrep : 25 ms
  temps KMP : 50 ms

Don Quixote (430 267 mots) :

- Recherche de « Quixote » :
  temps par egrep 10 ms
  temps par index :
  - construction de l'index : 223 s
  - chargement de l'arbre 650 ms
  - recherche dans l'arbre : <1µs
- Recherche de « Q(u|i|x)*ote » :
  temps par egrep : 10 ms
  temps par l'automate : 600 ms
- Recherche de « uixote »
  temps egrep : 10 ms
  temps KMP : 150 ms

Dans un premier temps on peut constater la constance et l'optimisation de egrep fruit d'années de maturation.
On peut constater que Kmp est bien plus rapide que l'automate, tiens mieux le passage a l'échelle mais est beaucoup moins versatile.
L'index quand a lui est extrêmement coûteux dans le calcul de l'index et dans la construction en mémoire du *radix tree*, mais une fois en place, les recherches dans l'arbre sont proche de l'instantanée et de bien loin plus rapides que les autres méthodes.

Cinquième partie
Conclusions

Dans l'exercice de proposer un clone de la commande egrep, qui est rarement utilisée plusieurs fois sur le même fichier, l'automate et KMP suffisent. L'auto-

mate permet de prendre en charge l'ensemble des motifs possibles et KMP viens accélérer la recherche sur les mots naturels, qui sont de loin les plus fréquemment recherchées.

Néanmoins si on regarde la situation d'un point de vue plus global, celui d'un moteur de recherche sur le projet gunttemberg, on seras plus enclin a lancer plusieurs fois des recherches dans un même document, et l'investissement payé a construire et charger des index pour ces documents se rentabilise rapidement. Plus les recherches sur un meme document sont fréquentes plus la méthode de l'index est payante.