Sorbonne Université

Faculté de Sciences et Ingénierie Master II Stl
DAAR

Clone de egrep unix

Étudiants
Lyes Lounis
Arezki Galou

Table des matières

1 Introduction

La commande egrep de Unix permet de chercher un motif dans un texte et afficher toutes les lignes contenant ce motif .
le motif peut etre soit une chaine de caractere simple ("Babylone") ou une expression reguliére : une représentation d'un langage algébrique reconnaissable par un automate fini déterministe .
Le but du projet est d'implementer différents algorithme afin de reproduire egrep, ici pour les éxpressions reguliere on considere seulement les symboles suivants :

— concatenation : "ab" .

— etoile : "c*" la lettre c repeter zero ou plusieurs fois ("ccccc ...." ou "") .

— alternative : "motif1" | "motif2" soit le motif1 ou le motif2

Dans notre implementation si le motif se reduit a une chaine de caractere simple (pas une regE) on réalise la recherche avec l'algorithme de : Knuth-Morris-Pratt .

Dans le cas ou le motif a cherché est une regE on réalise la recherche avec l'algorithme de : Aho Ulman

2 Algorithmes de recherches

2.1 Methode Aho Ulman

On utilise cette méthode afin de créer un automate déterministe qu'on utilsera lors de la recherhce du motif dans un texte, pour cela on passe par les étapes suivantes :

— On transforme la regEx en un arbre syntaxique .

— A partir de cet arbre construire un automate avec ε-transitions .

— On enléve les ε-transitions

— Enfin construire notre automate détreminisite .

2.1.1 Transformation en automate avec ε transitions

On parcourt l'arbre syntaxique et on applique cet algorithme :
.Si le noeud est une **feuille** (caractére) : On construit deux noeud (initial et finale) avec une transition de ce caractere depuis le noeud initial vers le noeud finale

.Si le noeud est une **Alternative** : on construit deux automates récursivement avec ces deux fils, on construit un noeud initial, on ajoute deux ε transitions vers les deux nouvelx automates depuis le noeud initial, on ajoute ε transition depuis les deux noeud final des nouvelx automates vers un nouvelx noeud finale .

.Si le noeud est une **Concaténation** : On construit deux automates avec ces deux fils, on ajoute une ε-transition depuis le noeud finale du premier fils vers le noeud initiale de l'autre automate

.Si le noeud est une **Etoile** : On construit un automate avec son fils , on crée deux noeud (initial ni et finale nf), on ajoute deux ε transitions depuis l'initial(ni) vers le finale(nf) et vers le noeud initial du nouvel automate (fils), on ajoute ε transition depuis le noeud finale du nouvel automate vers le noeud finale (nf) .

2.1.2 Transformation en automate sans ε transitions

Dans cette etape on parcourt l'automate avec ε transitions et on construit entierement un nouvel automate de cette maniére :
algo (courant : node, dernierImportant : node)
On garde une réference du dernier noeud important visité (un noeud qui a une transition entrante autre que ε plus le noeud initial), au debut on initialise avec le noeud initial(dernierImportant = initial)
Pour chaque noeud visité on parcourt ses transitions :
Soit une transition α vers un noeud n :

- Si $\alpha = \varepsilon$: on fait un appel recursif (on visite n)
- Sinon : Dans ce cas le noeud n est un noeud important alors :
  - Créer un nouvel noeud (newNode)si ce n'est déja créer qui représente ce noeud important .
  - On ajoute une transition (dernierImportant, α, newNode).
  - Enfin mettre a jour dernierImportant = newNode et visiter n.

les etats acceptables : Si le noeud qu'on visite (courant) est un noeud acceptable alors dernierImportant est acceptable aussi .

2.1.3 Transformation en automate deterministe

Dans cette étape on parcourt l'automate sans ε transitions et on construit entierement un nouvel automate déterministe de cette maniére :
Principe : Pour chaque nouvel noeud crée on garde une reference vers l'ensemble des noeuds regroupé .
initialisation : Créer un nouvel noeud n, qui a comme ensemble le noeud initial de notre automate sans ϵ .
Parcours : Pour chaque noeud visité on parcourt son ensemble de noeud regroupé : et pour chaque noeud n dans cet ensemble on parcourt ses transitions sortantes pour construire une map (caractere : ensemble de noeud) .

On parcourt la map et pour chaque ensemble de noeuds faire :

— Si on a déja créer un noeud contenant cet ensemble alors ne rien faire .

— Sinon créer un nouvel noeud (newNode) contenant cet ensemble, puis on ajoute une transitions (noeud courant, clé, newNode), enfin on visite newNode .

construire un nouvel noeud contenant cet ensemble de noeud (On vérifie d'abord si le noeud existe pas déja)
Les etats acceptables : Un etat est acceptable si au moins un etat dans son ensemble regroupé est un etat acceptable

2.1.4 Structure finale

Aprés avoir construit notre automate déterministe, on le transforme en une matrice (nombre de noeud x 256)
matrice[i] [α] = k : transition depuis l'etat i vers l'etat j avec α

2.2 Méthode Knuth-Morris-Prat

L'algorithme KMP est conçu pour améliorer la recherche naive d'une sous chaine de caractére, la particularité de cet algorithme réside dans son pré-traitement de la sous chaine, ainsi on a 2 etapes a effectuer :

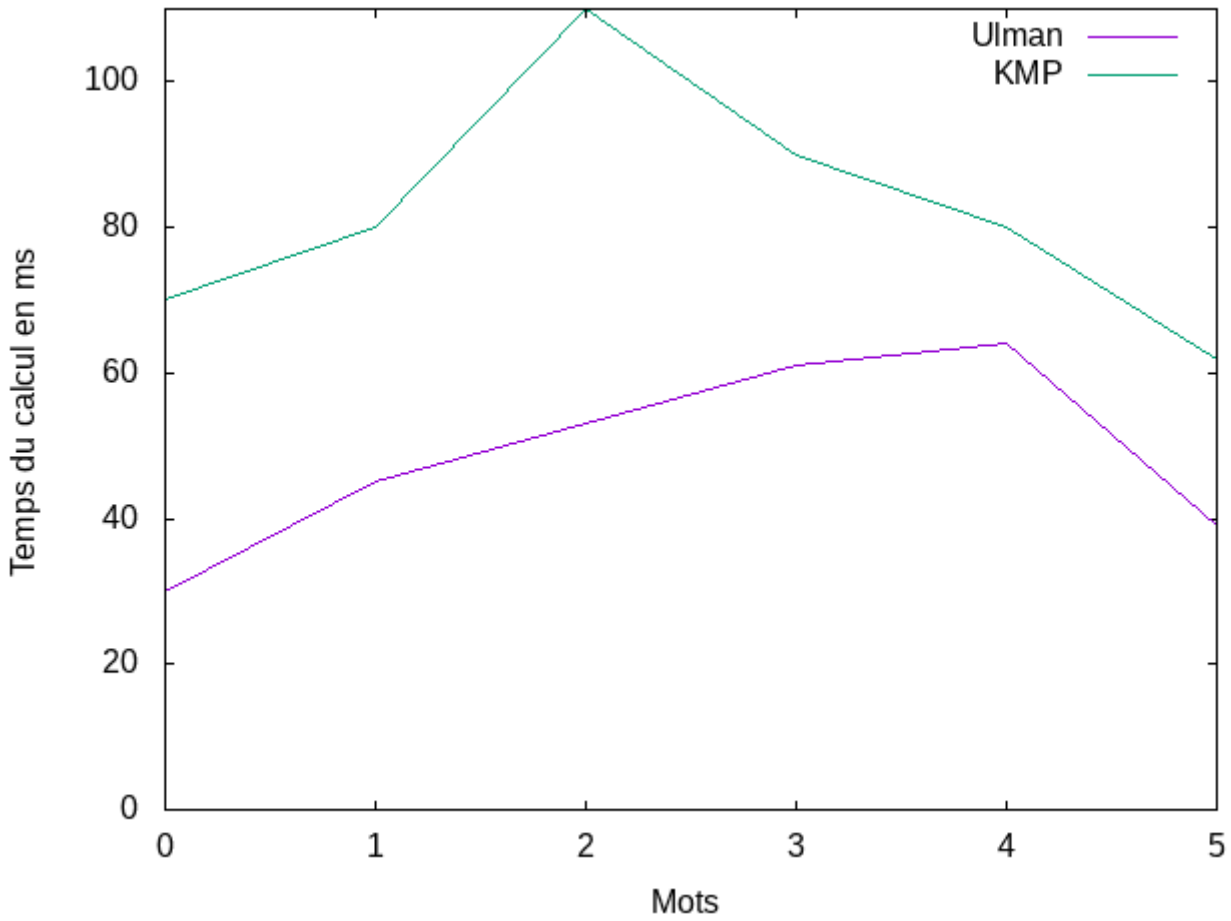
— **Calcule de la retenue** : on calacule un tableau de taille égale a la sous chaine, chaque case indique la prochaine position

— **La recherche** : on se servant du tableau **retenue** on évite de revenir a chaque echec a la position i+1 .

3 Teste

3.0.1 Aho Ulman VS KMP

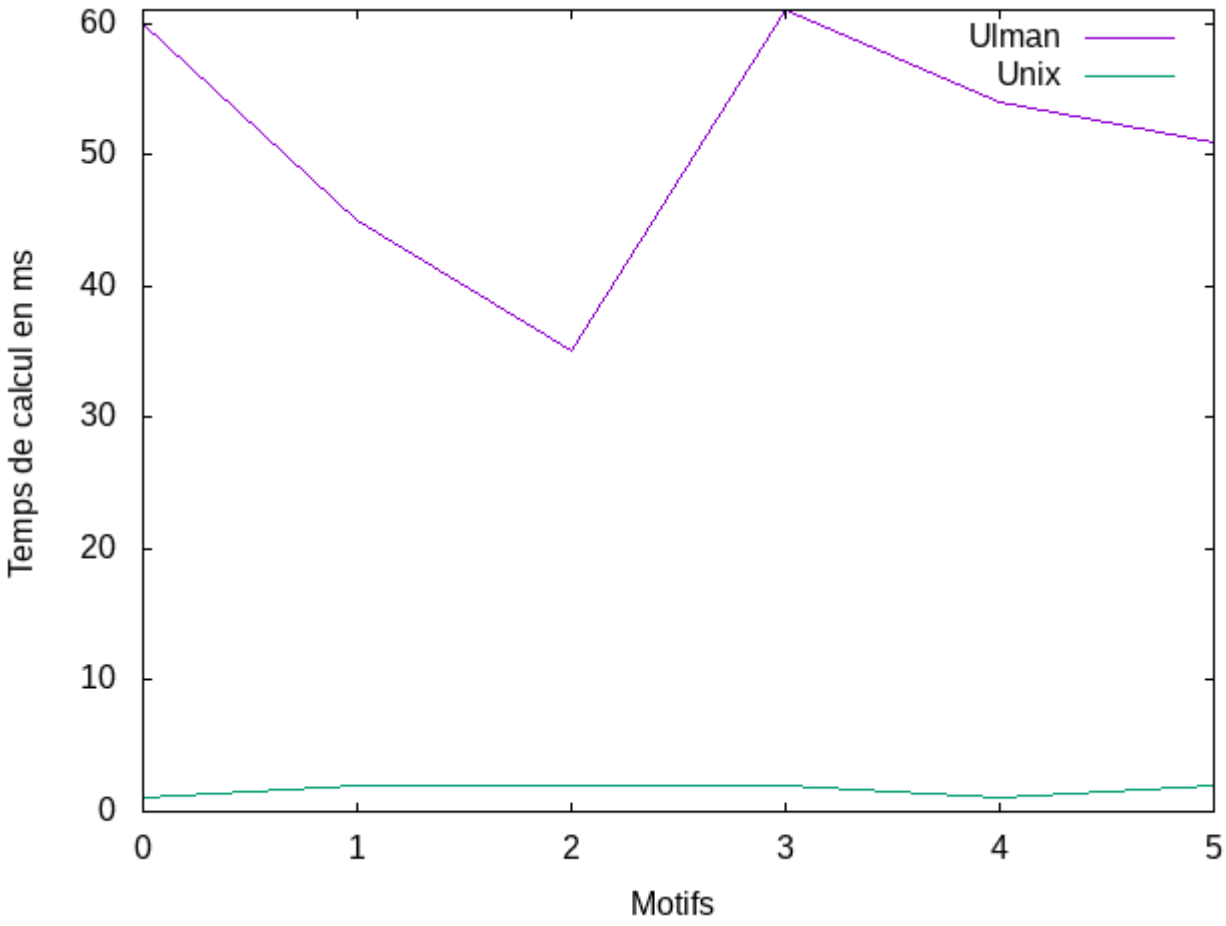
On compare le temps d'exécutions entre les algorithme Aho Ulman et KMP avec différents mots .



On remarque que la methode Ulman est plus efficace que KMP, le temps de construction et de recherche est beaucoup moins important dans Ulman

3.0.2 Aho Ulman vs Egrep linux

On compare le temp d'exécution entre les algorithmes Aho Ulman et egrep de Unix avec différents motifs



On remarque que egrep de Unix est beaucoup plus efficace, le temps de construction de notre automate est important ce qui augmente le temps de calcul .

4 Conclusion

Unix reste plus rapide que nos deux implementation, par contre la méthode de Ulman est plus efficace que celle de KMP, donc on prefere s'en passer de kmp, et utiliser la méthode de Ulman meme dans le cas d'une chaine caractere simple .