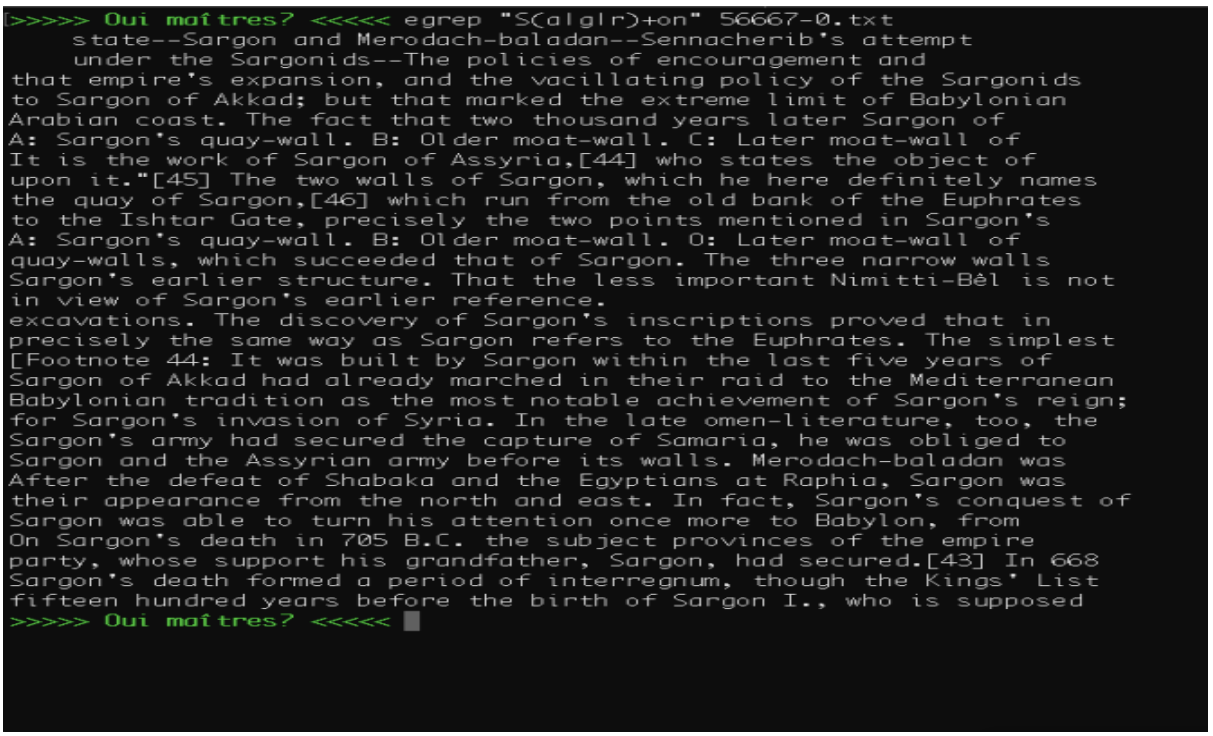
Rapport

pour le projet Clone de Egrep avec support partiel des ERE

rédigé par

Gianni-Alessandro NGAMY et
Anas YAHYAOUI



Octobre 2022

Table des matières

1 Introduction

Grep est un programme de recherche de chaîne de caractères à partir d'une expression régulière. Sa limite est qu'il ne supporte que les expressions régulières de base. Néanmoins, une autre version nommée "Egrep" supporte les expressions régulières étendues et offre donc plus de puissance. Dans ce projet, nous devrons développer un clone de Egrep en java en utilisant l'algorithme Aho-Ullman, nécessitant l'utilisation d'un arbre syntaxique qui nous permettra d'en déduire un automate déterministe, dont on minimise le nombre d'états, qui vérifiera la conformité de chaque mot du texte avec l'expression régulière, c'est l'algorithme Aho-Ullman.

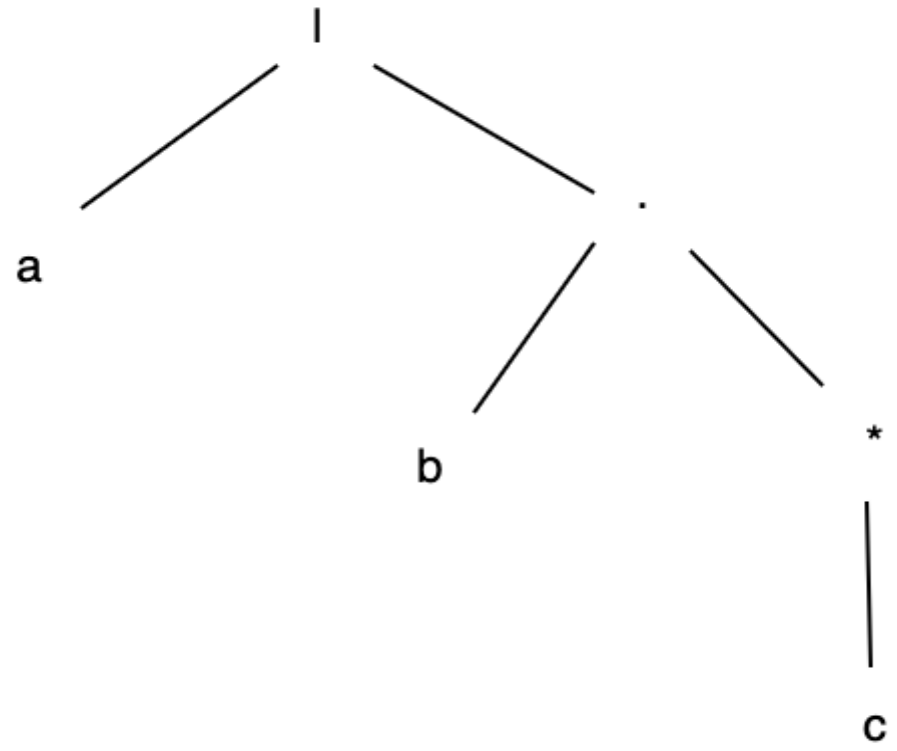
Nous développerons ensuite l'algorithme KMP (Knuth-Morris-Pratt) qui aura le même principe, qui est la recherche de chaîne de caractères mais dans ce cas à partir d'une chaîne de caractères et non une expression régulière. Nous utiliserons des listes afin de stocker les résultats de la recherche.

Nous comparerons ces deux algorithmes ayant un but commun mais des structures et méthodes différentes.

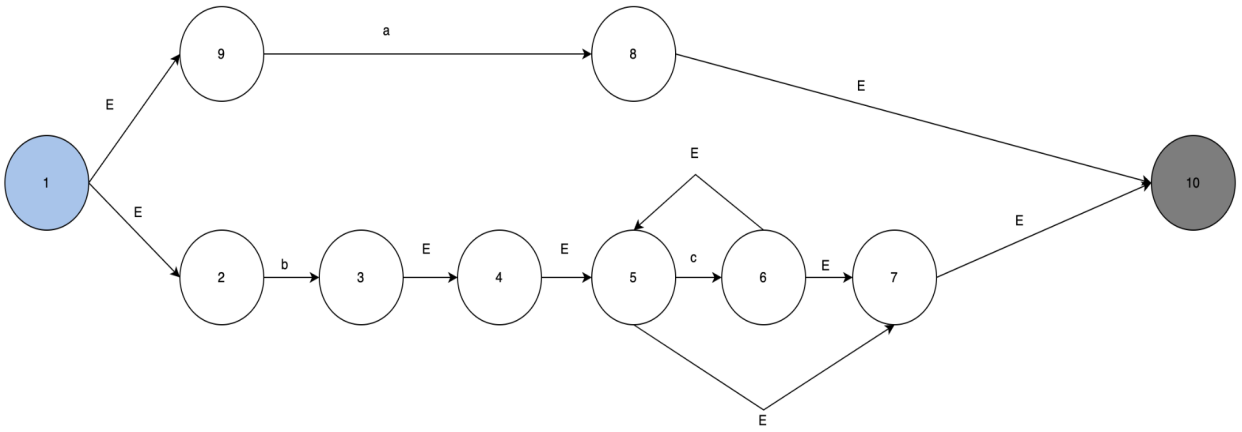
2 Principe

Algorithme Aho-Ullman :

Nous avons en input un texte et une expression régulière que l'on transformera en arbre syntaxique.



Dans l'exemple ci-dessus, nous avons l'arbre syntaxique de l'expression régulière a|bc*. Le "." représente la concaténation.

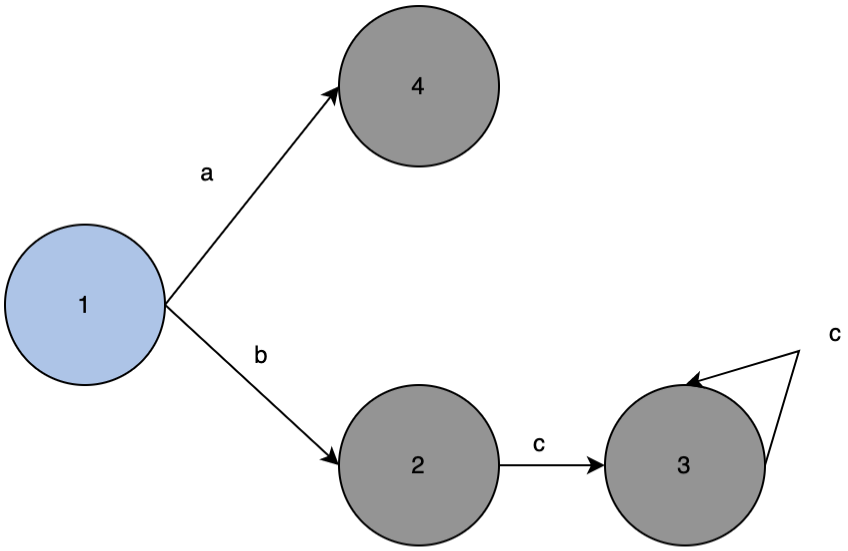


Automate fini non-déterministe avec E-transitions selon Aho-Ullman

A partir de l'arbre syntaxique précédent, nous avons l'automate ci-dessus avec en bleu l'état initial et en gris l'état final. Il est fini mais non-déterministe, la prochaine étape est de le rendre déterministe avec la méthode des sous-ensembles.

Tableau de la méthode des sous-ensembles

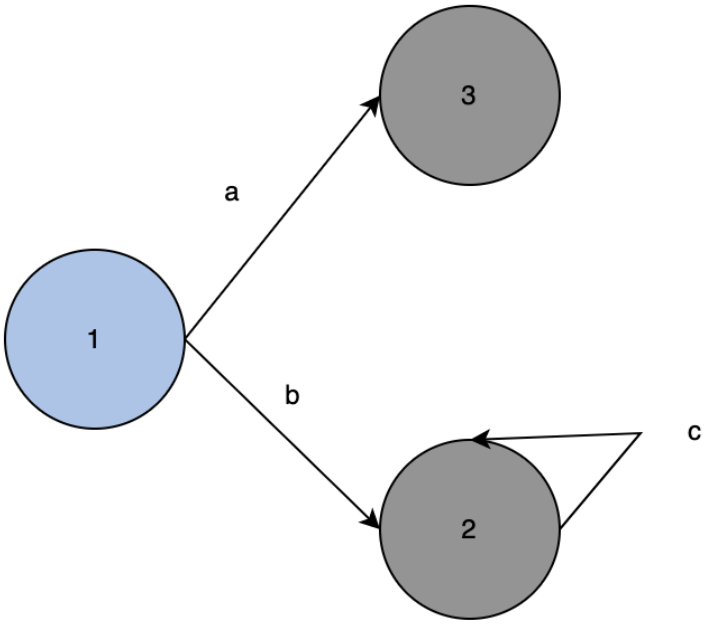
Nouvel état	Anciens états	« a »	« b »	« c »	État initial ?	État final ?
1	1, 2, 9	8, 10	3, 4, 5, 7, 10	Ø	T	F
2	8, 10	Ø	Ø	Ø	F	T
3	3, 4, 5, 7, 10	Ø	Ø	5, 6, 7, 10	F	T
4	5, 6, 7, 10	Ø	Ø	6, 5, 7, 10	F	T



Automate fini déterministe en utilisant la méthode des sous-ensembles

Le tableau ci-dessus nous montre la méthode employée pour rendre le précédent automate déterministe par la méthode des sous-ensembles.

Puis, nous devons minimiser l'automate afin d'avoir le moins d'états possibles.



Automate déterministe avec un nombre minimum d'états

À partir de cet automate, nous pouvons tester si une chaîne de caractères suit la règle de notre expression régulière. Et c'est ce que nous ferons en parcourant entièrement le texte donné et en testant chaque chaîne de caractère.

Algorithme KMP :

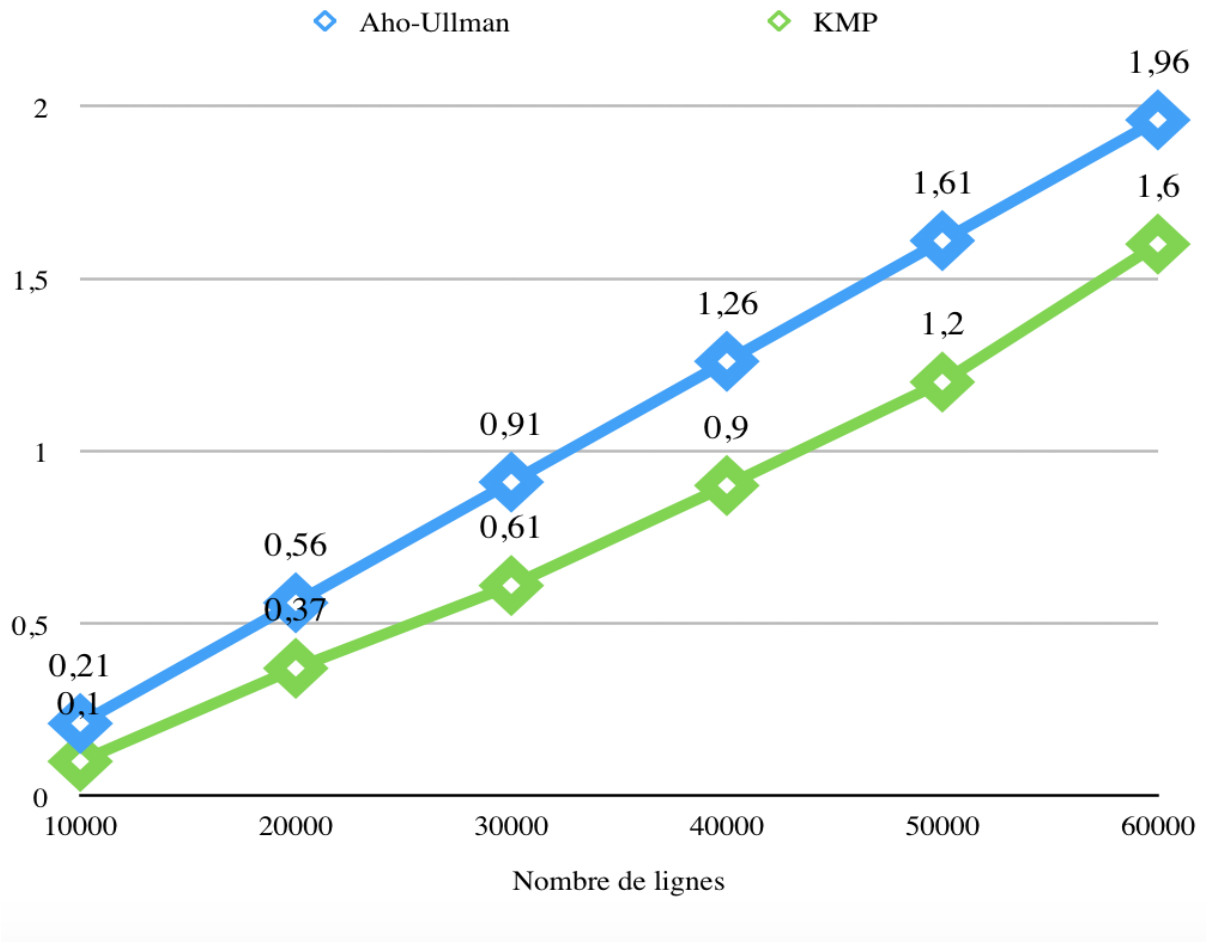
Nous avons en input un texte et une chaîne de caractères. Nous déclarons une liste qui comportera tous les caractères du texte et une liste qui comportera pour chaque position la prochaine position potentielle de la chaîne recherchée. Après avoir rempli cette deuxième liste, la création d'une nouvelle liste est nécessaire pour contenir les indices de départ des réelles occurrences de la chaîne recherchée dans le texte. Nous avons alors en output, une liste de position des occurrences de la chaîne de caractères recherchée dans le texte ainsi que le nombre d'occurrences de cette chaîne de caractères.

3 Attentes

Nous avons un premier algorithme qui utilise des structures de données plus élaborées que le second, néanmoins après avoir déterminé l'automate à partir de l'expression régulière nous ne parcourons le texte qu'une seule fois. Alors que dans le second, nous devons le faire au moins 3 fois, 2 fois pour préparer les structures de données et une dernière fois pour la recherche. De ce fait, nous pensons que le temps d'exécution de la phase de préparation peut-être équivalent mais dans la phase de recherche le premier semble plus rapide car moins complexe.

4 Résultats

Nombre de lignes	Aho-Ullman	KMP	Var.
10000	0,21	0,1	-52
20000	0,56	0,37	-34
30000	0,91	0,61	-33
40000	1,26	0,9	-29
50000	1,61	1,2	-25
60000	1,96	1,6	-18



5 Analyse

Nos résultats sont exprimés en seconde et la colonne Var. représente la différence de temps entre les 2 algorithmes (exprimé en pourcentage). Cette différence suit la formule : ((KMP/Aho-Ullman)-1)*100.

Les résultats nous montrent que l'algorithme KMP est plus rapide que l'algorithme Aho-Ullman, on remarque qu'il est en moyenne 31% plus rapide en fonction de la longueur du texte donné, mais la différence de temps des algorithmes semblent se rapprocher plus le texte est long.

7 Conclusion

Pour conclure, l'algorithme bien que plus simple à mettre en place, tant dans la définition des structures de données utilisées et dans l'exécution de la boucle principale est plus efficace que celui de Aho-Ullman sur des textes à grand nombre de lignes. Cependant, on remarque que plus la longueur du texte est grande, plus les deux algorithmes tendent vers le même temps d'exécutions. Cela peut s'expliquer par le fait que KMP parcourt plus de fois le texte entier que Aho-Ullman. Pour avoir un avis plus général sur les forces du chaque algorithme, il aurait fallu tester les performances sur des textes bien plus grands et bien plus petits. La limite du KMP est qu'il ne permet pas de rechercher des expressions régulières, ainsi le problème n'est pas exactement le même. On en conclut que pour des recherches d'une chaîne de caractères connues, il vaut mieux utiliser KMP sinon Aho-Ullman.