10/10/2021

Projet 1 – Clone de egrep

Lounes BRAHIMI
MASTER 2 STL DAAR

1. Définition du problème :

1.1 Introduction :

Le projet a pour objectif la recherche de chaînes de caractères au sein d'un fichier textuel, semblable à la commande « egrep », ce dernier devra donc supporter des expressions régulières restreintes aux éléments suivants : « les parenthèses », « l'alternative », « la concaténation », « l'opération étoile », « le point » et « la lettre ASCII ».

1.2 Contraintes :

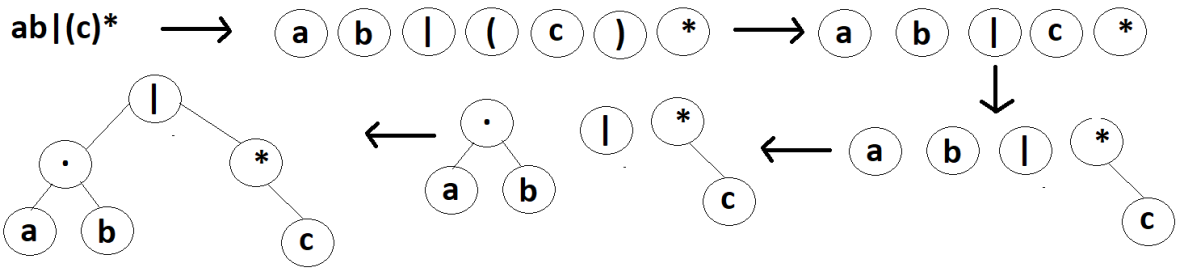
- Si la chaine de caractère recherchée est une expression régulière non réduite à une simple concaténation, cette dernière sera transformée en un arbre syntaxique dans un premier temps. Puis en un automate fini non déterministe contenant des epsilons transitions. Ce dernier sera déterminé. Finalement chaque suffixe d'une ligne du fichier textuel sera examiné afin de vérifier si elle est reconnaissable par l'automate obtenu.
- Si l'expression régulière recherchée est réduite à une simple concaténation, une autre méthode n'utilisant pas les automates sera employée, cette dernière s'appuie sur l'algorithme Knuth-Morris-Pratt.

2. Principaux algorithmes :

2.1. Recherche en utilisant un automate :

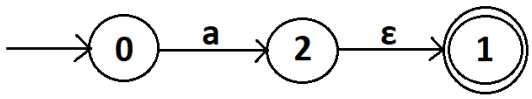
- Transformation de l'expression régulière en arbre syntaxique :

  Au début tous les éléments de l'expression régulière deviennent des arbres à un seul nœud, puis l'algorithme supprime les nœuds à parenthèses en prenant soin de traiter récursivement les nœuds qu'ils encadrent. Ensuite, il détecte la feuille étoile et l'arbre sur laquelle elle doit s'appliquer, ce dernier deviendra ainsi son seul sous arbre. Après cela, il détecte les concaténations, et crée leurs arbres respectifs de gauche à droite. Enfin, dans le même sens de parcours que la concaténation, l'algorithme détecte le sous-arbre gauche et droit de chaque nœud alternation, à la fin du parcours on aura plus qu'une seule structure arborescente représentant notre arbre syntaxique souhaité.



Transformation de « ab|(c) » en arbre syntaxique*

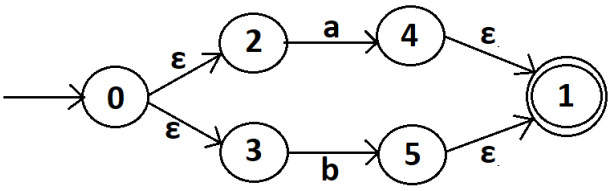
- Transformation d'un arbre syntaxique en un automate fini non déterministe avec epsilons transitions :
  - L'état initiale sera toujours l'état numéroté « 0 ».
  - L'état finale sera toujours l'état numéroté « 1 ».
  - Si l'arbre est constitué que d'une feuille on crée la transition suivie d'une epsilon transition, exemple « a » :



	a	b	ε	I	F
0	[2]	null	null	[1]	null
1	null	null	null	null	[1]
2	null	null	[1]	null	null

éxécuté avec le projet

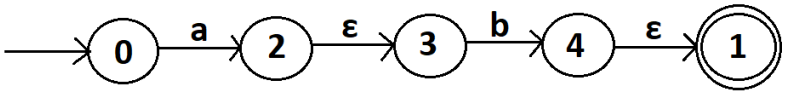
  - Dans le cas d'une alternation, l'état avant celle-ci crée deux epsilons transitions vers deux nouveaux états l'un appliquera récursivement l'algorithme sur le fils gauche de l'alternation, l'autre sur le fil droit, les deux états finaux de ses deux branches feront respectivement une dernière epsilon transition pour se rejoindre sur l'état final de l'automate ou l'état qui permettra la continuation de la transformation s'il existe, exemple « a|b » :



	a	b	ε	I	F
0	null	null	[2, 3]	[1]	null
1	null	null	null	null	[1]
2	[4]	null	null	null	null
3	null	[5]	null	null	null
4	null	null	[1]	null	null
5	null	null	[1]	null	null

éxécuté avec le projet

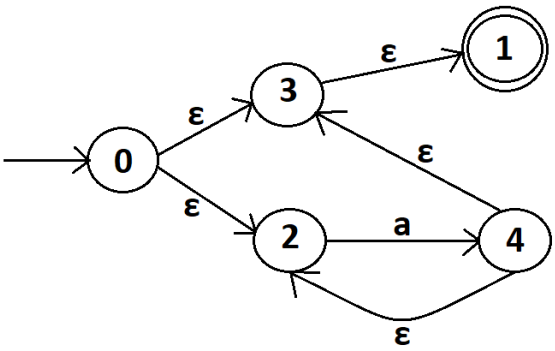
- Dans le cas d'une concaténation, l'algorithme est appliqué récursivement sur le fils gauche et le fils droit, les deux résultats se lient avec une epsilon transition depuis la gauche vers la droite, exemple « a.b » :



	a	b	ε	I	F
0	[2]	null	null	[1]	null
1	null	null	null	null	[1]
2	null	null	[3]	null	null
3	null	[4]	null	null	null
4	null	null	[1]	null	null

éxécuté avec le projet

- Dans le cas d'une étoile, l'état courant fait sortir deux epsilons transitions vers deux nouveaux nœuds, le premier signifie qu'on peut ne pas effectuer ce qui est à l'intérieur de la boucle, ce dernier peut donc atterrir avec une autre transition sur l'état final, le deuxième est le corps de la boucle, ce dernier une fois pris peut atterrir avec une epsilon transition sur un état menant à un état final ou bien atterrir sur l'état de début pour refaire le corps de la boucle, exemple « (a)* » :



	a	b	ε	I	F
0	null	null	[2, 3]	[1]	null
1	null	null	null	null	[1]
2	[4]	null	null	null	null
3	null	null	[1]	null	null
4	null	null	[3, 2]	null	null

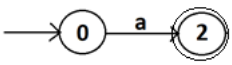
exécuté avec le projet

- Suppression des epsilons transitions de l'automate fini non déterministe avec epsilon transitions obtenue à l'étape suivante :

  L'algorithme parcours tous les états de l'automate, si l'état ne fait pas d'epsilon transition alors l'état avec ses transitions sortantes sont gardés tel quel dans le nouvel automate. Sinon la liste des états pointés avec des epsilons transitions est détectée, si un état de cette dernière liste à lui aussi à son tour une epsilon transition vers un nouveau nœud, ce nouveau nœud sera ajouté à la liste (cela résout le problème des epsilons transitions successives), ensuite les transitions (sans epsilons) des états appartenant à la liste détectée deviennent les transitions propres de l'état courant. Enfin, pendant le processus les états pointés par des transitions sans epsilon sont marqués comme non-puit, ce qui permet à la fin de supprimer les états puits.

  Suppression des epsilons sur les exemples de la section précédente :

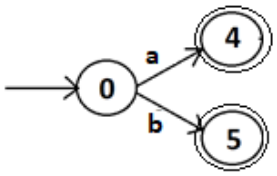
  - « a » :



	a	b	ε	I	F
0	[2]	null	null	[1]	null
1	null	null	null	null	null
2	null	null	null	null	[1]

exécuté avec le projet

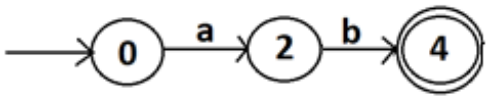
  - « a|b » :



	a	b	ε	I	F
0	[4]	[5]	null	[1]	null
1	null	null	null	null	null
2	null	null	null	null	null
3	null	null	null	null	null
4	null	null	null	null	[1]
5	null	null	null	null	[1]

exécuté avec le projet

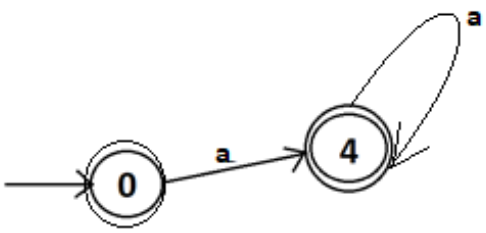
- « a.b » :



	a	b	ε	I	F
0	[2]	null	null	[1]	null
1	null	null	null	null	null
2	null	[4]	null	null	null
3	null	null	null	null	null
4	null	null	null	null	[1]

éxécuté avec le projet

- « (a)* » :



	a	b	ε	I	F
0	[4]	null	null	[1]	[1]
1	null	null	null	null	null
2	null	null	null	null	null
3	null	null	null	null	null
4	[4]	null	null	null	[1]

éxécuté avec le projet

- Déterminer l'automate sans epsilons transitions :

L'algorithme commence par parcourir les états pour détecter les finaux et les stocker dans une liste.

Chaque indice d'état a une chaine de caractère qui représente son nom unique qui l'identifie, ce dernier est généré en rencontrant un nouveau groupe d'états pointés, il se nomme par la concaténation de ces derniers dans un ordre croissant. Dans un premier lieu, l'état initiale se nommera toujours « 0 ».

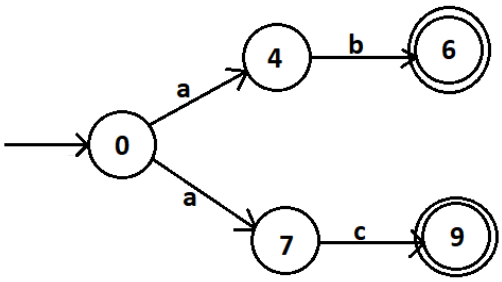
Premièrement, l'algorithme traite l'état initial, il examine la liste des nœuds pointés. Si cette dernière ne se compose que d'un seul élément, si c'est la première fois que cet élément est rencontré, alors on lui crée un nom afin de l'indexé, ensuite, le nœud est pointé dans le nouvel automate, si ce n'est pas la première fois qu'il est rencontré, il est directement pointé dans le nouvel automate. Sinon si la liste d'éléments pointés par l'état initial est aux pluriels, alors le nom symbolisant cette liste de nœuds cibles est généré, si ce dernier a déjà été rencontré alors une transition vers lui est créé dans le nouvel automate depuis l'état initiale, sinon, le nouvel état au nom combiné est créé et pointé dans le nouvel automate depuis le nœud initial.

À cette étape, l'algorithme vérifie si de nouveaux nœuds (en dehors du nœud initial) ont été créé dans le nouvel automate. Si c'est le cas, il les parcourt un à un. Il vérifie l'ensemble des nœuds pointés. Si elle est réduite à un seul élément et que ce dernier a déjà été créé, alors il sera directement pointé dans la nouvelle matrice, sinon il est généré et pointé dans le nouvel automate. Si l'ensemble est composé de plusieurs éléments, le nom symbolisant cette collection de nœuds est produit, si ce dernier a déjà été rencontré alors une transition vers lui est créé dans le nouvel automate depuis l'état courant, sinon, le nouvel état au nom combiné est créé et pointé dans le nouvel automate.

Enfin, les états qui ont un nom composé d'un nœud parmi ceux qui sont finaux dans l'ancienne matrice, deviennent finaux dans le nouvel automate fini déterministe.

Exemple de détermination :

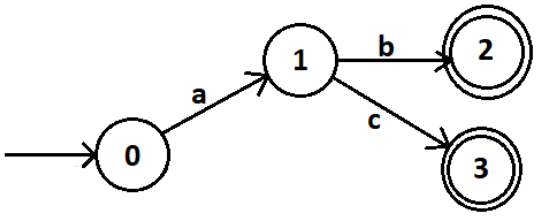
- NDFA pour « ab|ac » sans epsilons transitions :



	a	b	c	ε	I	F
0	[4, 7]	null	null	null	[1]	null
1	null	null	null	null	null	null
2	null	null	null	null	null	null
3	null	null	null	null	null	null
4	null	[6]	null	null	null	null
5	null	null	null	null	null	null
6	null	null	null	null	null	[1]
7	null	null	[9]	null	null	null
8	null	null	null	null	null	null
9	null	null	null	null	null	[1]

exécuté avec le projet

- DFA pour « ab|ac » :



	a	b	c	ε	I	F
0	[1]	null	null	null	[1]	null
1	null	[2]	[3]	null	null	null
2	null	null	null	null	null	[1]
3	null	null	null	null	null	[1]

exécuté avec le projet

- La recherche :

L’algorithme commence par décomposer le texte du fichier en une liste de chaines de caractères représentant les lignes respectives.

L’automate tournera donc pour chaque ligne de la liste obtenue.

Pour toute ligne, il fera tourner l’automate déterministe sur tous ses suffixes successivement du plus long au plus court, si ce dernier valide l’automate, dans ce cas il a terminé de traiter la ligne courante et le renvoie.

L’automate est exécuté de façon successive sur chaque caractère du suffixe, il prend un état courant (l’état initiale au départ) et le caractère représentant la transition qui doit être effectuée, si la transition existe dans l’automate à partir du nœud courant, alors il renvoie l’état cible et ce dernier devient à son tour le nouvel état courant.

```
_____________________________________________
regEx : S(a|r|g)*on
_____________________________________________
    state--Sargon and Merodach-baladan--Sennacherib's attempt
    under the Sargonids--The policies of encouragement and
that empire's expansion, and the vacillating policy of the Sargonids
to Sargon of Akkad; but that marked the extreme limit of Babylonian
Arabian coast. The fact that two thousand years later Sargon of
A: Sargon's quay-wall. B: Older moat-wall. C: Later moat-wall of
It is the work of Sargon of Assyria,[44] who states the object of
upon it."[45] The two walls of Sargon, which he here definitely names
the quay of Sargon,[46] which run from the old bank of the Euphrates
to the Ishtar Gate, precisely the two points mentioned in Sargon's
A: Sargon's quay-wall. B: Older moat-wall. O: Later moat-wall of
quay-walls, which succeeded that of Sargon. The three narrow walls
Sargon's earlier structure. That the less important Nimitti-BÃªl is not
in view of Sargon's earlier reference.
excavations. The discovery of Sargon's inscriptions proved that in
precisely the same way as Sargon refers to the Euphrates. The simplest
[Footnote 44: It was built by Sargon within the last five years of
Sargon of Akkad had already marched in their raid to the Mediterranean
Babylonian tradition as the most notable achievement of Sargon's reign;
for Sargon's invasion of Syria. In the late omen-literature, too, the
```

exécuté avec le projet

2.2. Recherche en utilisant l'algorithme Knuth-Morris-Pratt :

L'algorithme KMP permet de trouver les occurrences d'une chaîne de caractères (expression régulière réduite à une suite de concaténations) N dans un texte M avec une complexité linéaire en O(|N|+|M|) dans le pire cas.

L'algorithme a été développé en 1970 par Knuth et Pratt, et dans un autre contexte, par Morris, il est publié conjointement en 1977.

L'algorithme commence par créer le « Factor », qui est une liste ordonnée des éléments de la chaine caractère recherchée.

```
regEx : mami
Factor : [m, a, m, i]
```

exécuté avec le projet

Ensuite, L'algorithme génère une liste nommée « Carry Over » qui indique pour chaque position un décalage à effectuer si la chaine de caractère ne matche pas à partir d'un certain indice.

Le premier indice est toujours initialisé à « -1 ».

Ensuite nous avançons un à un sur les éléments de « Factor », et pour chaque indice « i » nous cherchons la taille du plus long suffixe propre qui est aussi un préfixe de là sous chaine commençant à l'indice « 0 » et qui finit à l'indice « i-1 », cette longueur est le décalage de l'indice « i » dans « Carry Over ».

Après cela, l'algorithme effectue le traitement suivant :

```
for (i=1; i < Factor.size(); i++){
        if ( ( Factor[i] = Factor[0] )
                ET    ( CarryOver[i] == 0 ) )  alors {
                CarryOver[i] = -1
        }
}
```

Enfin, l’algorithme finit cette partie « Carry Over » avec ce traitement :

```
for (i=0; i < Factor.size(); i++){
    if ( ( CarryOver[i] != -1 )
        ET ( Factor[i] == Factor[CarryOver[i]] ) )  alors {
        CarryOver[i] = CarryOver[CarryOver[i]]
    }
}
```

Le dernier élément de Carry Over est toujours un 0.

```
regEx : mamamia
Factor : [m, a, m, a, m, i, a]
CarryOver[-1, 0, -1, 0, -1, 3, 0, 0]
```

exécuté avec le projet

A l’étape de la recherche, l’algorithme commence par décomposer le texte en une liste de lignes. Pour chaque ligne de la collection il cherche dans tous ses suffixes en prenant soin de faire les décalages dans les cas de crash en utilisant le Carry Over. Enfin, il renvoie cette dernière ligne si elle match et s'il n’y a pas de crash.

```
regEx : Sargon
    state--Sargon and Merodach-baladan--Sennacherib's attempt
    under the Sargonids--The policies of encouragement and
that empire's expansion, and the vacillating policy of the Sargonids
to Sargon of Akkad; but that marked the extreme limit of Babylonian
Arabian coast. The fact that two thousand years later Sargon of
A: Sargon's quay-wall. B: Older moat-wall. C: Later moat-wall of
It is the work of Sargon of Assyria,[44] who states the object of
upon it."[45] The two walls of Sargon, which he here definitely names
the quay of Sargon,[46] which run from the old bank of the Euphrates
to the Ishtar Gate, precisely the two points mentioned in Sargon's
A: Sargon's quay-wall. B: Older moat-wall. O: Later moat-wall of
quay-walls, which succeeded that of Sargon. The three narrow walls
Sargon's earlier structure. That the less important Nimitti-BÃªl is not
in view of Sargon's earlier reference.
excavations. The discovery of Sargon's inscriptions proved that in
precisely the same way as Sargon refers to the Euphrates. The simplest
[Footnote 44: It was built by Sargon within the last five years of
Sargon of Akkad had already marched in their raid to the Mediterranean
Babylonian tradition as the most notable achievement of Sargon's reign;
```

exécuté avec le projet

3. Améliorations :

L'utilisation de l'algorithme KMP à l'étape de la recherche avec l'automate pourrait être une amélioration. Ainsi quand l'automate n'accepte pas le suffixe d'une ligne, on le réappliquera sur un nouveau suffixe en utilisant le décalage « Carry Over ».

4. Banc de tests :

Un banc de test est exécuté par l'algorithme, elle contient 14 expressions régulières ayant une complexité qui suffit pour examiner tous les cas qui peuvent être saisis. Ses expressions régulières sont appliquées sur 3 livres téléchargés sur la plateforme « Gutenberg ». Les tests s'exécutent avec succès en une moyenne de 40 secondes.

Néanmoins plusieurs tests ont été effectué à la construction du projet étape par étape (visualisation de la matrice de l'automate fini non déterministe avec et sans epsilons transitions, visualisation de la matrice de l'automate déterministe, vérification de plusieurs cas d'expressions régulières à la main pour la recherche avec l'automate et avec l'algorithme KMP).

5. Test de performance des deux algorithmes :

Un fichier se nommant « tests_kmp_seulement » a été créé, il contient seulement des recherches de motifs réduites à des concaténations.

Un autre fichier nommé « test_automate_seulement » a été conçu. Ce dernier est identique au premier fichier sauf que tous les motifs recherchés sont les fils d'une étoile, ce qui oblige l'algorithme à utiliser la méthode avec automate.

L'exécution du fichier « tests_kmp_seulement » dure en moyenne 20 secondes.

L'exécution du fichier « test_automate_seulement » quant à lui prend environ 40 secondes.

Les résultats obtenus, prouvent que l'utilisation de l'algorithme kmp, permet à cette échelle d'aller deux fois plus rapide, tel cité dans la section « Améliorations », il serait intéressant d'inclure les décalages que permet l'algorithme kmp pendant la recherche dans la méthode avec automate.

6. Conclusion :

Le projet fut très constructif, car il m'a permis de mettre en pratique des notions de théories qui ont été abordés plusieurs fois pendant mon cursus (notamment les automates).

Les algorithmes conçus s'appliquent sur des livres, de façon récursive et en considérant les livres tels que des objets représentant le champ d'application de la recherche. Ce champ d'application peut être généralisé et devenir générique. Le projet pourrait ainsi évoluer pour pouvoir effectuer une recherche sur une bibliothèque (une base de données de livres).