Projet d'algorithmique du texte

Algorithme Aho-Corasick

Projet réalisé par

- MEDDAH Amine
-

Sommaire

1 Description du projet

Description du projet

L'algorithme d'Aho-Corasick est un algorithme de recherche de chaîne de caractères (ou motif) dans un texte.

Il consiste à avancer dans une structure de données abstraite appelée dictionnaire qui contient le ou les mots recherchés en lisant les lettres du texte T une par une. La structure de données est implantée de manière efficace, ce qui garantit que chaque lettre du texte n'est lue qu'une seule fois. Généralement le dictionnaire est implanté à l'aide d'un trie ou arbre digital auquel on rajoute des liens suffixes. Une fois le dictionnaire implanté, l'algorithme a une complexité linéaire en la taille du texte T et des chaînes recherchées.

L'algorithme extrait toutes les occurrences des motifs. Il est donc possible que le nombre d'occurrences soit quadratique, comme pour un dictionnaire a, aa, aaa, aaaa et un texte aaaa . Le motif a apparaît à quatre reprises, le motif aa à trois reprises, etc.

Le but de ce projet est d'implémenté Aho-Corasick en fonction de trois types de représentation du *Trie* :

- Une matrice de transition.
- Une liste d'adjacence
- Un mix entre matrice et liste.

2 Description des structures de données

Description des structures de données

1. *Structure du trie pour la représentation sous forme de matrice*

Cette structure contient :

- Un entier « ***maxNode*** » qui définit le nombre maximal de nœuds que l'arbre peut contenir. Il servira à la création de l'arbre. (Une matrice avec *maxNode* ligne, est composée de*256* colonnes et représente le code ASCII des caractères).
- Un entier « ***nextNode*** » qui nous indique le prochain emplacement libre de l'arbre (matrice) pour y ajouter un nouveau nœud.
- Une matrice d'entier « ***transition*** » qui contiendra l'arbre. (Chaque nœud aura une dépendance avec un autre nœud en fonction d'une lettre de l'alphabet).
- Un tableau d'entier « ***finite*** » de taille *maxNode*indique pour chaque nœud s'il est final ou non ('0' si l'état n'est pas final, sinon il est final).

2. *Structure du trie pour la représentation sous forme de liste*

On distingue deux structures,

- List
  - Un entier « ***targetNode*** » qui représente le nœud actuel dans l'arbre.
  - Un caractère « ***lettre*** » qui représente la transition à un autre nœud.
  - Un pointeur « ***next*** » qui pointe vers le nœud qui le suit.
- Trie
  - Un entier « ***maxNode*** » qui définit le nombre maximal de nœuds que l'arbre peut contenir. Il servira à la création de l'arbre. (Une liste de nœud de maximum *maxNode* éléments).
  - Un entier « ***nextNode*** » qui indique le prochain emplacement libre de l'arbre (liste) pour y ajouter un nouveau nœud.
  - Une liste de type **List**« ***transition*** » qui formera l'arbre (chaque élément contient le nœud actuel, la transition et le nœud qui le suit de par la transition).
  - Un tableau d'entier « ***finite*** » de taille *maxNode* indique pour chaque nœud s'il est final ou non ('0' si l'état n'est pas final, sinon il est final).

3. *Structure du trie pour la représentation sous forme de mix entre la matrice et la liste*

Elle contient la structure « List » vue précédemment ainsi que la structure « Trie » à un détail près qui concerne la représentation de l'arbre.

- Un tableau d'entier « ***transitionRoot*** » pour indiquer les nœuds qui sont reliés directement à la racine de l'arbre.
- Une liste de **List** « transitionOthers » qui représente le branchement avec les éléments du tableau c'est-à-dire la suite de la liste ***transitionRoot***.

4. *Structure node pour la construction des suppléances*

Cette structure est une simple file où on va enfiler les éléments au cours de la recherche de leurs dépendances en regardant les suffixes existant de ce dernier.

- Un entier « ***target*** », l'élément courant de la file.
- Un pointeur « ***next*** » vers le prochaine élément de la file.

5. *Structure ACMatricepour la construction des d'AC-Corasick_ Matrice*

Elle regroupe :

- La structure d'un **Trie.**
- Un tableau d'entier qui représente les suppléances.

6. *Structure ACListepour la construction des d'AC-Corasick_ Liste*

Elle regroupe :

- La structure d'un **Trie.**
- Un tableau d'entier qui représente les suppléances.

7. *Structure ACMixtpour la construction des d'AC-Corasick_ Mixt*

Elle regroupe :

- La structure d'un **Trie.**
- Un tableau d'entier qui représente les suppléances.

Description des fonctions

- Trie createTrie(intmaxNode);

Réalise la création du ***Trie*** en:

- allouant l'espace mémoire (***maxNode***) de pour l'arbre (Matrice de transition, Liste d'adjacence, ou les deux pour Mixt) ;
- affectant la valeur '0' au ***nextNode*** pour dire qu'il n'y a pas d'éléments dans l'arbre ;
- allouant l'espace mémoire(***maxNode***) pour le tableau des états finaux ;
- initialisant le tableau des états finaux à '0' pour chaque état.

- voidinsertInTrie(Trie trie, unsigned char *w);

Réalise l'insertion dans la matrice en :

- vérifiantsi le ***nextNode***est égale à '0' pour savoir si on ajoute à la racine ;
- Si ce n'est pas le cas, vérifier :
  - si la racine avec le caractère de transition contient un état, on avance au prochain nœud on sauvegarde le chemin, puis on avance au prochain caractère du mot ;
  - l'opération précédente se reproduit pour chaque ajout jusqu'à ne plus trouver de chemin, alors on insère le mot, on incrémente le ***nextNode*** et on met le dernier état en état final en affectant '1' dans le tableau des états finaux lui correspondant.

Réalise l'insertion dans la liste en :

- vérifiant que tant que ce n'est pas la fin du mot :
  - si le caractère de transition correspond au caractère du mot :
    - si c'est le cas, on avance au prochain élément de la liste et on avance dans le mot ;
    - dès qu'on ne trouve plus de correspondance, on ajoute un élément à la suite de ce dernier et on incrémente le ***nextNode*** pour signifier que le prochain nœud libre est ici.

Réalise l'insertion pour le mixt en :

- vérifiant si la transition n'existe pas dans le ***transtionRoot*** ;si c'est le cas :
    - se positionner sur le premier élément de la liste qui correspond à cette racine ;
    - sinon créer la liste pour cette liste ;
    - reprendre exactement le même processus de recherche et d'ajout définit pour la liste d'adjacence.

❖ Void addListeNode(List* liste, unsigned char c, int index)

Réalise l'ajout d'un élément à la liste en vérifiant :

- si la liste est vide ; ajouter en premier sinon ajouter l'élément en fin de liste.

❖ enfiler (inttarget)

Elle permet d'enfiler un élément dans la file.

❖ intdefiler()

Elle permet de défiler un élément de la file.

❖ intisEmpty()

Elle permet de vérifier si la file est vide.

❖ ACMatricecreateACMatrice()

Réalise la création du ***ACMatrice***en:

- allouant l'espace mémoire pour l'arbre (Matrice de transition, Liste d'adjacence, ou les deux pour Mixt) ;
- allouant l'espace mémoire (***maxNode***) pour le tableau des suppléants et initialiser tous ses éléments avec '-1'.

Initialisant le tableau des états finaux à '0' pour chaque état.

Pour les fonctions :

❖ ACListepreAC(char** mots, intnbrMots);

❖ voidcompleter(ACListeacListe);

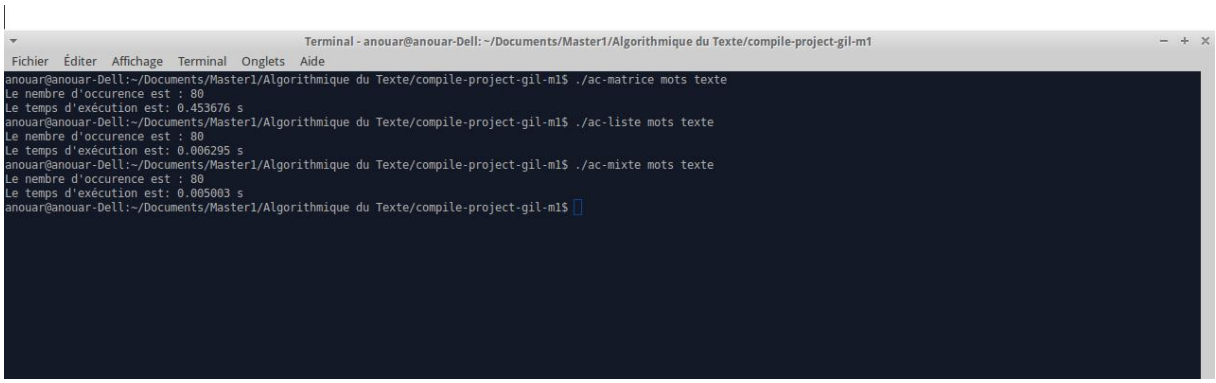
❖ voidac(char** mots, intnbrMots,unsigned char* text, inttextLenght);

Elles sont implémentées exactement comme décrit en cours et respecter scrupuleusement toutes les étapes.

4 Résultats d’exécution

Résultat d’exécution

- Aho-Corasick avec l’exemple du sujet :

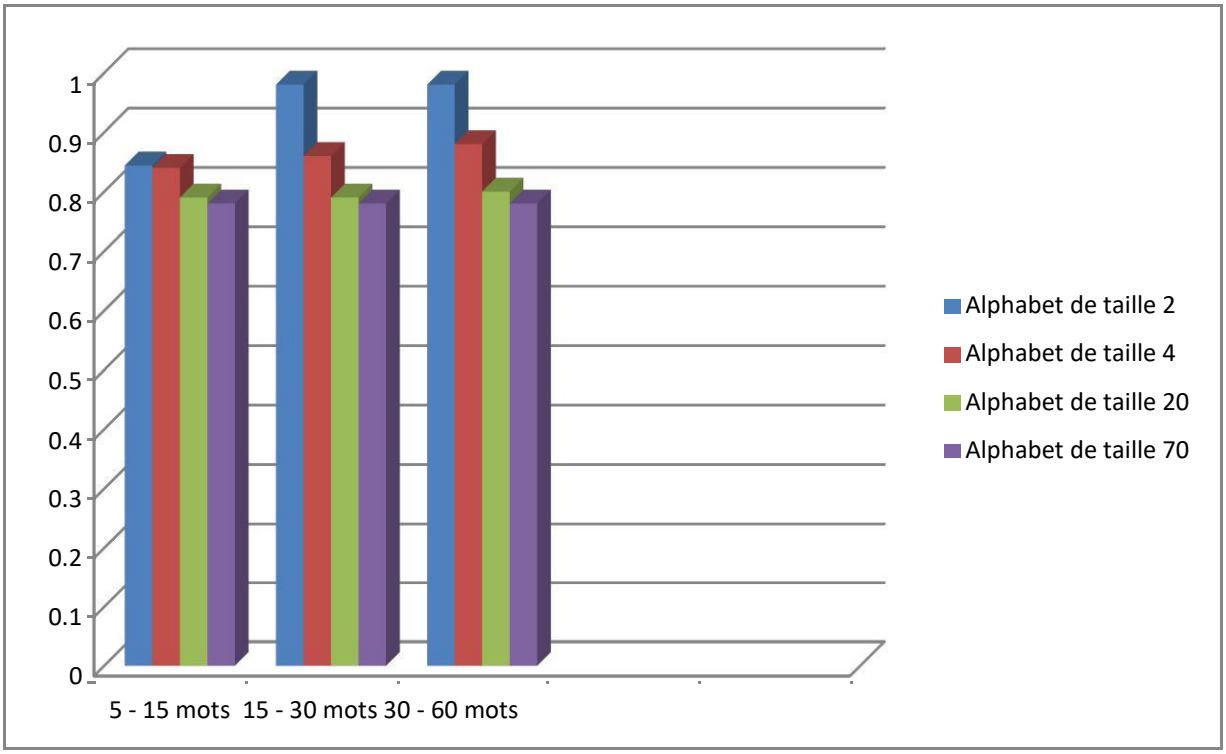


Nombre d’occurrences trouvées : 80.
Temps d’exécution avec Aho-Corasick Matrice : 0,453 s
Temps d’exécution avec Aho-Corasick Liste : 0,006 s
Temps d’exécution avec Aho-Corasick Mixt : 0,005 s

❖ Aho-Corasick Matrice

Taille du mot / Texte et mot d'alphabet	5 - 15	15 - 30	30 - 60
2	0,844	0,98	0,98
4	0,84	0,86	0,88
20	0,79	0,79	0,80
70	0,78	0,78	0,78

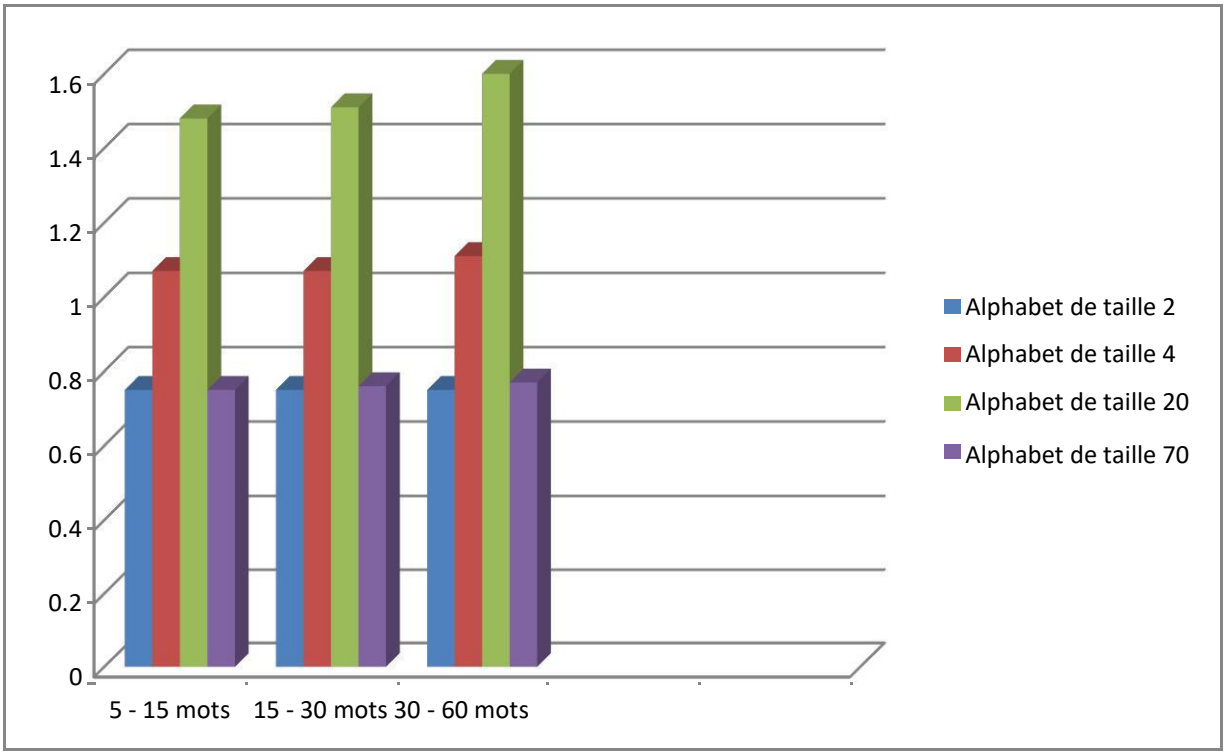
Graphe de comparaison Aho-Corasick Matrice



❖ Aho-Corasick Liste d’adjacence

Taille du mot / Texte et mot d’alphabet	5 - 15	15 - 30	30 - 60
2	0,75	0,75	0,75
4	1,07	1,07	1,11
20	1,48	1,51	1,60
70	0,75	0,76	0,77

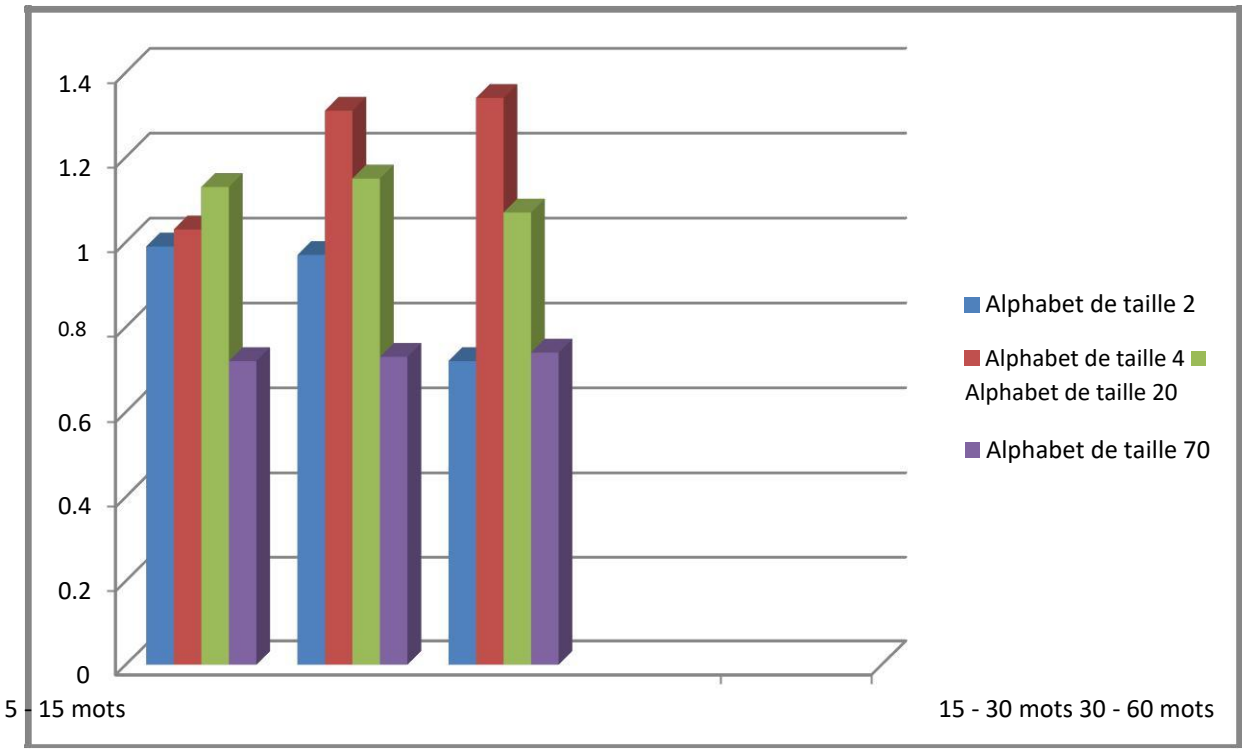
Graphe de comparaison Aho-Corasick Liste d’adjacence



❖ Aho-Corasick Mixt

Taille du mot / Texte et mot d'alphabet	5 - 15	15 - 30	30 - 60
2	0,99	0,97	0,97
4	1,03	1,31	1,34
20	1,13	1,15	1,07
70	0,72	0,73	0,74

Graphe de comparaison Aho-Corasick Matrice



❖ Comparaison entre Aho-CorasickMatrice, Aho-CorasickListe et Aho-CorasickMixt

Graphe de comparaison Aho-Corasick Matrice, Aho-Corasick Liste d'adjacence et Aho-Corasick Mixt,