Structure de Données et Algorithmes 2
L2 Informatique - S4 Automne 2020–2021
Projet de développement

Détection automatique de langue

Introduction

1 Modalités

Le TP est à rendre **au plus tard le 13 décembre 2020 à 23h59** via la plateforme *Moodle*. Il est à réaliser **en binôme exclusivement**, durant le temps libre.

Travail à rendre

Il vous est demandé un document synthétique tenant sur **au maximum trois feuilles recto-verso, soit six pages**, où vous ferez figurer les réponses aux questions ainsi que tout élément que vous jugerez bon de préciser. Vous déposerez sur *Moodle*, dans une archive au format `.tar.gz` :

- le document synthétique au format électronique `.pdf` ;
- les sources **dûment commentées** de votre réalisation en langage C, ce qui inclut notamment un fichier `Makefile` mais pas les fichiers objets `.o`. **Un code qui ne compile pas ne sera pas accepté** ;

L'ensemble des programmes ayant servi à l'élaboration du projet doit être dûment commenté. Essayez d'attacher une grande importance à la lisibilité du code : il est indispensable de respecter les conventions d'écriture (*coding style*) du langage C et de bien présenter le code (indentation, saut de ligne, etc.) ; les identificateurs (nom de types, variables, fonctions, etc.) doivent avoir des noms cohérents documentant leur rôle ; indiquez les pré-conditions, post-conditions, paramètres, la sortie et un descriptif de chaque fonction. Un commentaire se doit d'être sobre, concis et aussi précis que possible.

Pensez à utiliser des outils comme `gdb`, `valgrind` ou `gprof` pour faciliter votre développement et optimiser votre programme.

Barème indicatif

- Code (programmation et algorithmique) : 12 points
  - Structures de données : 9 points
  - Mise en application : 3 points
- Code (forme, commentaires) : 2 points
- Rapport : 6 points
- Bonus/Malus rendu des TPs : 2 points

2 Problème

On se propose d'étudier dans ce projet une méthode de détection automatique d'une langue. Pour simplifier cette étude, seuls l'allemand, l'anglais ainsi que le français seront considérés. Les accents et la ponctuation que l'on peut retrouver dans la langue française et allemande ne seront quant à eux pas utilisés.

3 Approche proposée

Afin de résoudre le problème de détection automatique d'une langue, une approche par correspondance de dictionnaire sera utilisée. La méthode est décrite ci-dessous :

1. Analyse des dictionnaires allemand, anglais et français
2. Lecture du texte à identifier
3. Pour chaque mot du texte à identifier tester la correspondance du mot par rapport à chacun des dictionnaires
4. La langue du texte est celle avec le plus grand nombre de correspondances positives

4 Structures de données

Trie

Un **trie**, parfois nommé **arbre préfixe**, est une structure de données sous forme d'arbre qui permet le stockage de chaînes de caractères. Souvent utilisé pour l'implémentation d'une table associative, nous utiliserons une forme légèrement modifiée du trie afin de stocker au mieux les dictionnaires.
La structure permettant de désigner un noeud de l'arbre est composée des éléments suivants :

— Un **tableau de pointeurs vers ses fils**, de taille 26 (pour les lettres allant de a à z). Une case i de ce tableau est **un pointeur vers son noeud fils** si il est possible de continuer un mot avec la lettre d'indice i à partir du préfixe encodé par le noeud courant, ou un pointeur **NULL sinon**.

— Un attribut permettant de savoir si le mot formé en partant de la racine et en allant jusqu'au noeud courant est un mot du dictionnaire.

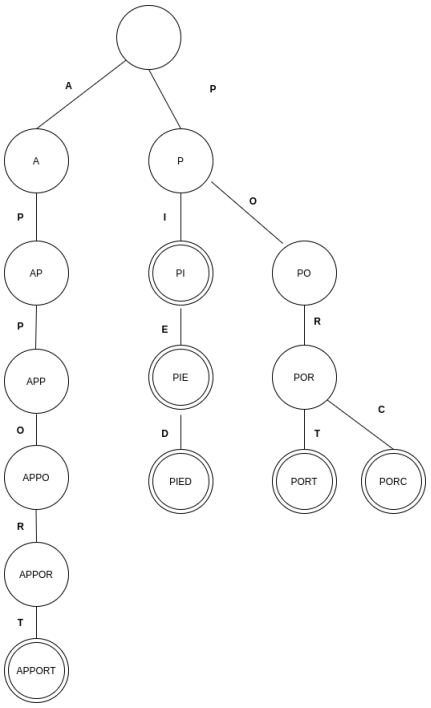

FIGURE 1 – Un trie représentant les mots PI, PIE, PIED, PORT, PORC et APPORT. Les noeuds de l'arbre qui sont double-cerclés sont les mots appartenant aux dictionnaires.

Directed Acyclic Word Graph

Un **Directed Acyclic Word Graph**, abrégé **DAWG** dans le reste du document, est un **graphe orienté acyclique** qui permet, à l'instar du trie vu précédemment, le stockage de chaînes de caractères. Les applications d'une telle structure sont également similaires à celles d'un trie.

La structure des sommets et des arêtes du graphe sera de type **récursive**, la représentation par matrice d'adjacence n'étant pas adaptée à la construction itérative du DAWG.

Voici les attributs composant la structure :

— Un id **unique** représentant le label associé au sommet courant.

— Un **tableau de 26 éléments** (pour les lettres allant de a à z) étant soit un **pointeur** vers le sommet suivant, soit **vide**. Ce tableau représente les arêtes sortantes du sommet courant.

— Un attribut permettant de savoir si le mot formé en partant de la racine et en allant jusqu'au sommet courant est un mot du dictionnaire.

En plus de la structure d'un sommet, voici les autres structures de données qui seront utilisées durant la phase de génération du DAWG :

— Une structure représentant une **arête** contenant trois attributs : le label de l'arête, un pointeur vers le sommet gauche, un pointeur vers le sommet droit.

— Une **hashmap** qui est un tableau associatif permettant une association **clé/valeur** contenant les sommets enregistrés du graphe. Pour ajouter un sommet dans la hashmap, sa clé devra comprendre les informations suivantes : s'il s'agit d'un sommet **final** et les **arêtes sortantes** du sommet (label + id du sommet droit de l'arête).

— Une **pile** contenant les arêtes dont le sommet droit n'a pas encore été enregistré.

Petite précision : un sommet est considéré comme enregistré lorsque celui-ci a été minimisé.

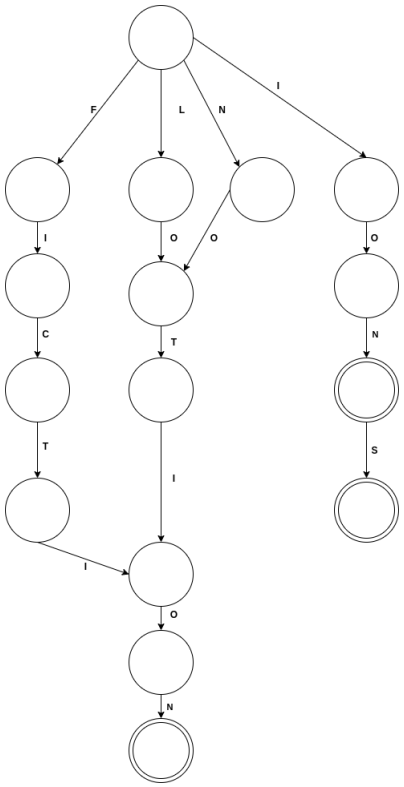

FIGURE 2 – Un DAWG représentant les mots FICTION, LOTION, NOTION, ION et IONS. Les sommets du graphe qui sont double-cerclés sont les mots appartenant aux dictionnaires.

Minimisation : La minimisation est une étape qui consiste à supprimer les noeuds intermédiaires inutiles et donc de réduire le nombre de noeud au strict minimum tout en conservant le même fonctionnement de notre graphe. Nous utiliserons cette étape de manière à maintenir un DAWG le plus compact possible en mémoire. C'est pour cette étape que la hashmap sera utile, pour faire des recherches efficaces dans le graphe.

Pseudo-code :

- Vous trouverez ci-dessous l'algorithme à utiliser afin de générer le DAWG.

```
# Lecture du dictionnaire
POUR tous les mots m du dictionnaire trié par ordre alphabétique FAIRE :
    Insérer le mot m dans le DAWG
FIN POUR
Minimiser le DAWG jusqu'à la profondeur 0
```

- Vous trouverez ci-dessous l'algorithme détaillant l'insertion d'un mot dans le DAWG.

```
Étape 1 :
Trouver la taille n du plus grand préfixe commun
entre le mot précédemment inséré et le nouveau mot à insérer
```

```
Étape 2 :
Minimiser le DAWG jusqu'à la profondeur n

Étape 3 :
Ajouter le suffixe non commun au graphe à partir,
soit de la racine si la pile est vide
soit à partir du sommet droit de la dernière arête empilée.

Pour chaque lettre du suffixe ajoutée au graphe empiler l'arête correspondante.

Étape 4 :
Marquer le dernier sommet ajouté comme étant final.
```

- Vous trouverez ci-dessous l'algorithme détaillant la minimisation du DAWG pour une profondeur p (passé en paramètre de la fonction).

```
TANT QUE la taille de la pile est supérieure à p FAIRE :
    Dépiler la pile, l'arête dépilée sera nommée a.

    Vérifier si un sommet équivalent au sommet droit
    de l'arête a est déjà présent dans la hashmap.

    SI c'est le cas
    ALORS :
        Relier le sommet gauche de l'arête a à ce sommet
    SINON
        Enregistrer ce sommet dans la hashmap.
    FIN SI
FIN TANT QUE
```

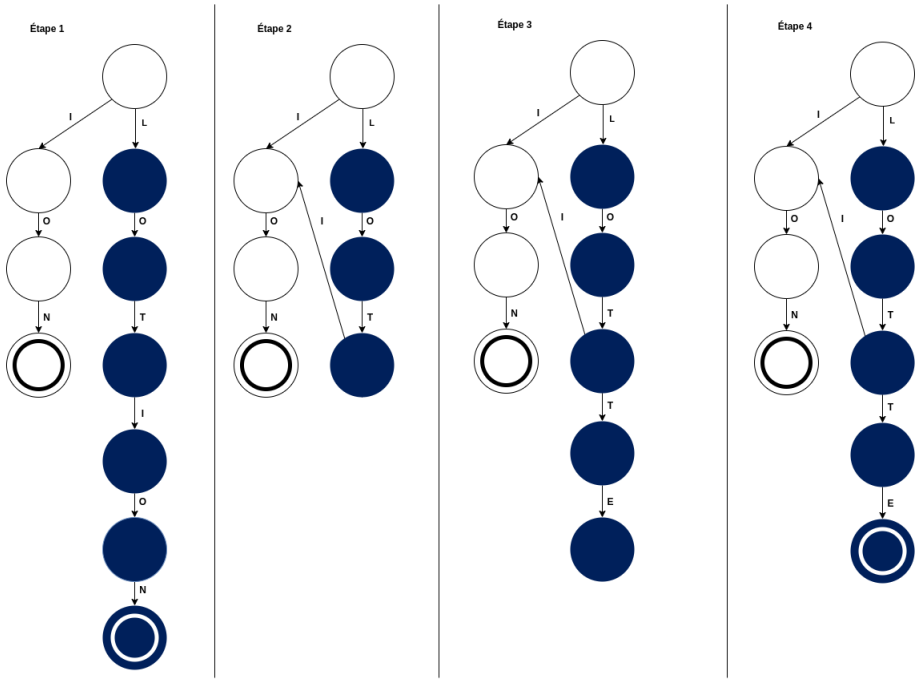

FIGURE 3 – Les différentes étapes d'insertion du mot LOTTE dans le DAWG contenant les mots suivants : ION, LOTION. Les sommets blanc sont les sommets déjà enregistrés dans le graphe et dont contenus dans la hashmap. Les sommets bleu marine sont les sommets qui ne sont pas enregistrés dans le graphe et dont les arêtes entrantes sont contenues dans la pile.

La figure 3 représente les différentes étapes de l'algorithme de génération du DAWG. L'état initial en est le suivant :

— Le mot ION a déjà été ajouté et minimiser par l'algorithme.
— Le mot LOTION est le dernier mot à avoir été ajouté et n'a donc pas encore été minimisé.

La figure 3 représente les différentes étapes lors de l'ajout du mot LOTTE :

1. Le mot LOTION est le dernier mot qui a été ajouté au DAWG. Comme il ne partage pas de préfixe commun avec le mot ION, le mot a été rajouté à la racine et toutes les arêtes du mot LOTION sont ajoutées dans la pile des arêtes non enregistrées.
2. Après avoir récupéré la taille du plus grand préfixe commun, le processus de minimisation commence. Les lettres à minimiser du mot LOTION sont donc dans cet ordre N, O, I. Ces trois sommets sont ensuite dépilés de la pile des arêtes non enregistrées. La figure 4 explique en détail la procédure sous forme de schéma.
3. Lors de cette étape, on ajoute le suffixe non commun à LOTION et LOTTE (à savoir **TE**) dans la pile des arêtes non enregistrées.
4. On marque le dernier sommet du mot LOTTE comme étant final.

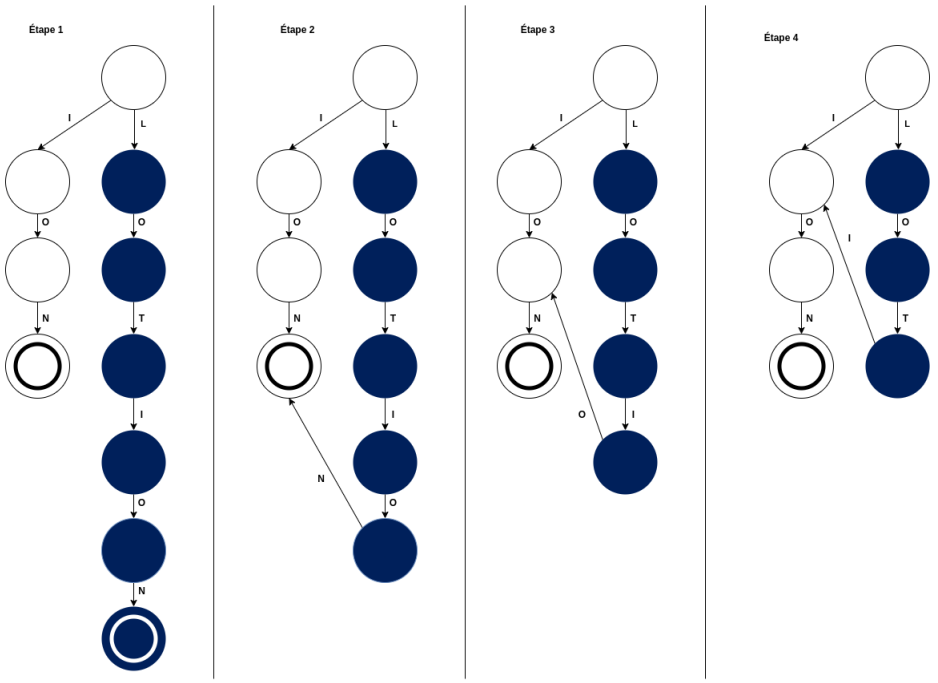

FIGURE 4 – Les différentes étapes de la minimisation du suffixe ION. Les sommets bleu marine sont les sommets qui ne sont pas enregistrés dans le graphe et dont les arêtes entrantes sont contenues dans la pile.

Voici quelques informations complémentaires qui sont hors du scope de l'UE de SDA2 mais qui sont relatives à ce sujet.

— Un DAWG est un type particulier de graphes appelé **"automate"** étudié en **théorie des langages**.
— La construction du DAWG se base sur le papier de recherche disponible à l'adresse suivante : https ://www.aclweb.org/anthology/J00-1002.pdf

5 Ressources et aides

Vous avez à votre disposition les fichiers suivants :

— L'implémentation d'une **hashmap** (*hashmap.h*). Vous pouvez trouver la documentation d'utilisation à cette adresse : https ://github.com/sheredom/hashmap.h
— L'implémentation d'une **pile** (*stack.h*, *stack.c*)
— Des fonctions pour la manipulation de caractères (*utils.h*, *utils.c*)
— Un *main* à compléter !
— 3 dictionnaires allemand, anglais, français.

Vous aurez à utiliser des fonctions de la librairie standard. Vous pourrez inclure *<stdbool.h>* pour utiliser les booléens en C. Pour la manipulation de chaînes de caractères, vous utiliserez *<string.h>* :

— **strlen()** permet de récupérer la longueur d'une chaîne.
— **strtok()** permet de découper une chaîne en sous-chaînes selon le token donné en entrée.

Certaines fonctions d'*utils.h*, de *stack.h* et de *hashmap.h* pourront aussi vous être utiles :

- *utils.h* :
  - **ascii_to_index()** qui permet de changer les cararactères de a à z en indice de tableau (a -> 0, b -> 1, c -> 2, ..., z -> 25)
  - **concat()** qui permet de créer une nouvelle chaîne de caractère concaténant les deux chaînes passées en paramètres
- *stack.h* :
  - **new_stack(n)** pour créer une pile de taille n
  - **stack_size(s)** pour récupérer la taille de la pile s
  - **stack_push(s, e)** pour empiler l'élément e dans la pile s
  - **stack_pop(s)** pour dépiler la pile s et récupérer l'élément
  - **stack_peek(s)** pour accéder au premier élément de la pile s mais sans le retirer de la pile
  - **is_stack_empty(s)** pour savoir si la pile s est vide ou non
- *hashmap.h* :
  - **hashmap_create()** qui permet d'initialiser une hashmap avec une taille qui est une puissance de 2
  - **hashmap_get()** qui permet de récupérer la valeur pour une clé donnée si la clé existe, NULL sinon
  - **hashmap_put()** qui permet d'enregistrer la valeur pour une certaine clé
  - **hashmap_iterate()** qui permet, à l'aide d'un pointeur de fonction, d'itérer à travers chacun des éléments contenus dans le dictionnaire
  - **hashmap_destroy()** qui permet de libérer l'espace mémoire alloué à la hashmap, **attention** l'espace mémoire des éléments qui ont été insérés dans la hashmap ne sont quant à eux pas libérés

  Des exemples concrets d'utilisation de la hashmap sont disponibles à l'adresse suivante : https ://github.com/sheredom/hashmap.h

6 Questions

Question 1 Implémentation de la structure Trie : vous devez proposer une implémentation complète de la structure d'arbre Trie contenant les fonctions d'**initialisation**, d'**insertion** d'un élément dans l'arbre, et de **recherche** d'un mot.

Question 2 Implémentation de la structure DAWG : vous devez proposer une implémentation complète de la structure de graphe DAWG contenant les fonctions d'**initialisation**, d'**insertion** d'un élément dans le graphe, et de **recherche** d'un mot. La phase d'insertion comprend une étape de **minimisation** du graphe, dont la fonction est également à implémenter.

Question 3 Proposer une **mise en application** de la détection automatique de langue dans le *main*. A partir d'une phrase en entrée, saisie par l'utilisateur, vous devrez être capable de pouvoir détecter la langue de cette phrase en utilisant l'une ou l'autre structure au choix en les alimentant avec les dictionnaires fournis.

Question 4 Proposer des **tests** permettant de valider le bon fonctionnement de votre approche et de l'utilisation des deux structures. Une partie de vos tests doit comprendre une **évaluation des performances**. L'ensemble du déroulement de vos tests ainsi que leurs résultats doivent être expliqués et détaillés dans votre rapport.

Question 5 Dans votre rapport, vous proposerez une **analyse théorique et pratique** sur les différences d'utilisation des deux structures implémentées.