Projet Programmation Avancée

Réalisation d'un correcteur orthographique

Objectif :

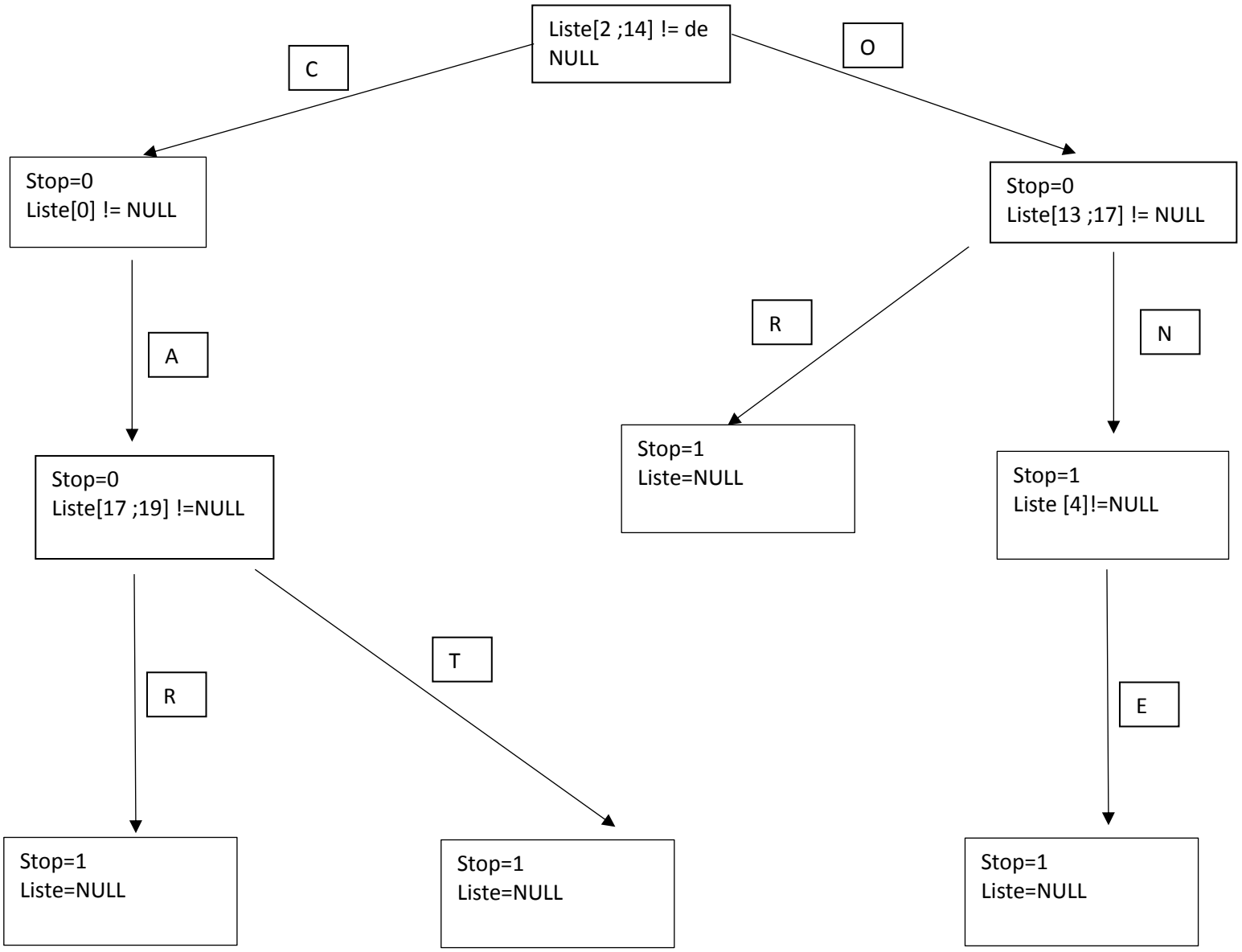
L'objectif de ce projet est de réaliser un programme qui permet de détecter dans un texte tous les mots mal orthographiés (que le dictionnaire utilisé ne connaît pas). Pour cela on utilisera un dictionnaire qui sera construit à partir d'un texte ou d'un ensemble de mots de référence.

Explication de la structure de dictionnaire choisie :

```
struct cell {
  int stop;
  struct cell * liste[26];
};
```

Nous avons choisi comme structure de donnée une ‘sorte’ de liste chainée. Contenant une liste de pointeur de cell , qui correspond aux 26 lettre de l’alphabet rangé dans l’ordre alphabétique. Le int stop sert à savoir si la structure pointée correspond à la fin d’un mot.

Exemple schématique de notre structure contenant les mots (on, or , car , cat , one) :



Explication des différents programmes du projet :

-int num(char c) ; renvoie le numéro alphabétique de la lettre rentrée en argument , si le caractère n'est pas une lettre de l'alphabet renvoie -1.

ex :num(b) renvoie 1

-int fin_mot(char c) ;renvoie 1 si le caractère lue représente la fin d'un mot (. , ? ; …) sino renvoie 0

- int ajout_mot (struct cell ** d , char *m) ; ajoute un mot dans la structure qui nous sert de dictionnaire. Renvoie 1 si pas d'erreur sinon 0.

-int creation_dico (FILE * fp, struct cell ** d) ;crée la structure de notre dictionnaire en lisant les mot contenue dans le fichier donné en argument .

-int comparaison(char *,int x , int conjug) ;regarde si le caractère m[x] correspond à la fin du mot si oui renvoie 1 sinon revoie 0.L'arguement conjug (0 désactivé , 1 activé) est une implémentation permettant de prendre en compte la conjugaison des verbes ou l'accord des noms (mais le programme n'est pas optimal car il conjugue aussi les noms et accorde les verbes).

-int reconaissance(struct cell * d , char * m) ;Ce programme vérifie si le mot m excite dans notre dictionnaire , si il existe il renvoie 0 sinon revoie 1 .

- int lecture(FILE * fd, struct cell * d) ;lit les mots d'un texte donné en paramètre (FILE * fd),et regarde si ce mot existe dans notre dictionnaire .Ce programme renvoie le nombre de mot non reconnue .

-void suprime_dico(strcut cell **d) ;supprime le dictionnaire rentré en paramètre en libérant l'espace alloué.

Respect du cahier des charges :

Notre programme permet d'implémenter et de manipuler un dictionnaire choisi sous la forme d'une structure comme décrit précédemment. Puis d'analyser une phrase mot par mot afin de relever les mots inconnus pour le dictionnaire. Comme nous avions remarquer que le dictionnaire utilisé au départ ne comprenait pas la conjugaison et les pluriels, nous avons rajouté une fonction qui reconnaît certains suffixes (-ing, -ed, -s). Mais cette fonction accepte chaque suffixe derrière n'importe quoi, ainsi les suffixes -ing et -ed peuvent se retrouvaient derrière des noms. Notre dictionnaire ne prend pas en compte les accents et autres caractère spéciaux, pour cela il aurait fallut modifier les fonctions afin quelles prennent en compte les caractère du code UTF-8. Nous aurions alors dus augmenté la taille de notre tableau considérablement.

Efficacité de notre structure :

Nous sacrifions l'espace vis à vis de la rapidité de recherche.

En effet, notre structure assigne à chaque fois un int plus un tableau de 26 pointeurs, il est donc très coûteux en mémoire par rapport à un dictionnaire sous forme d'une liste de mots (pour le dictionnaire que nous utilisions la taille était multiplié par 100) . Mais cette différence de taille diminue lorsque l'on augmente les mots avec des préfixes similaires. En revanche, le fait que pour une lettre il n'existe qu'une unique branche correspondante dans la structure rend la recherche de mot au sein du dictionnaire plus rapide car indépendante de la taille du dictionnaire.

Nous avions, à certain moments du projet, envisagé d'autres structures :

La première fût d'utiliser, à la place de notre tableau, une liste chaînée. Cela aurait permis d'obtenir un dictionnaire bien moins volumineux, mais lors de la recherche ne sachant pas quelle lettre existe à chaque étape nous aurions été obligés de tester chaque informations de la liste. Nous aurions alors eu un dictionnaire peu volumineux mais bien plus lent à la recherche.

Nous avions aussi envisagé une table de hachage basée sur les lettres les plus utilisées de la langue anglaise.