Mezouar Mikael Université Gustave Eiffel Yasser Hathat

Rapport DM : Algorithmique des arbres

12/03/2021 L2 informatique

Mezouar Mikael Université Gustave Eiffel Yasser Hathat

Introduction :

Dans ce DM , nous allons implanté en langage C un arbre ternaire lexicographique pour que l'on puisse à la fin construire le lexique d'un texte contenu dans un fichier fournit comme argument de la ligne de commande. L'exécutable aura pour nom Lexique et nécessitera toujours un argument : le nom du fichier à traiter . Le Programme aura les fonctionnalités suivantes : il devra afficher les mots du lexique en ordre alphabétique , sauvegarder les mots du lexique en ordre alphabétique (Le nom du fichier de sauvegarde est le nom du fichier d'entrée suivi du suffixe (.LEX)) ,indiquer si le mot est présent dans le texte et sauvegarder l'arbre dans un fichier, le nom du fichier de sauvegarde est le nom du fichier d'entrée suivi du suffixe (.DIC) (format décrit plus loin) . Dans ce rapport nous verrons comment nous avons implanté le programme, puis il y aura un guide utilisateur et enfin la conclusion.

Structure imposé :

Notre structure imposer était une structure avec 4 champs :
- une lettre (type char)
- 3 pointeurs : fils, freredroit, freregauche (type tree ou Tree)

Chaque nœud représente une lettre , un mot est donc chainée par le biais des fils . La première lettre du mot est un nœud, les lettres suivantes sont accessibles grâce au champs fils, la fin d'un mot est caractérisé par un nœud '\0'.

Documentation :

Tree *alloueNoueud(char x) ;
Fonction qui alloue un nœud , elle prend en paramètre un caractère de type char .

---

void clean_tree(Tree *tr) ;
Fonction qui nettoie l'arbre , le vide . Elle prend un arbre en paramètre.

---

int rechercher(Tree *tr char mot[]) ;
fonction qui recherche si le mot est dans l'arbre , si il y est la fonction return 1, sinon elle retournera 0. Elle prend un arbre et un mot en paramètre.

```c
int rechercher(Tree *tr, char mot[]){
    if(tr == NULL){
        return 0;
    }
    if (mot[0] == tr->value){
        if(mot[0]=='\0'){
            return 1;
        }
        else{

            return rechercher(tr->fils,mot+1);
        }
    }
    else{
        if(mot[0] < tr->value){
            return rechercher(tr->freregauche, mot);
        }
        else{
            return rechercher(tr->freredroit, mot);
        }
    }

}
```

---

int insertion(Tree **tr, char mot[]) ;
Fonction qui insere un mot dans un arbre. Elle prend un arbre et un mot en paramètre.

```
int insertion(Tree **tr, char mot[]){
    if(*tr == NULL){
        *tr = alloueNoeud(*mot);

    }
    if (*mot == (*tr)->value){
        if(*mot == '\0')
            return 1;

        else{
            return insertion(&((*tr)->fils), mot+1);
        }
    }
    else if((*tr)->value < *mot)
        return insertion(&((*tr)->freredroit),mot);

    else
        return insertion(&((*tr)->freregauche),mot);
}
```

Les fonctions suivantes affichent l'arbre :
void afficher(Tree *lexique, char *mot, int i) ;
void affiche_arbre(Tree *lexique) ;
La première prend en paramètre un arbre ,un mot et un int . La deuxième prend en paramètre l'arbre.

```c
void afficher(Tree *lexique, char *mot, int i){
    if (lexique->value == '\0'){
        printf("%s\n", mot);
    }
    if (lexique->freregauche != NULL){
        afficher(lexique->freregauche, mot, i);
    }
    if (lexique->fils != NULL){
        mot[i] = lexique->value;
        mot[i+1] = '\0';
        afficher(lexique->fils, mot, i+1);
    }
    if (lexique->freredroit != NULL){
        afficher(lexique->freredroit, mot, i);
    }
}
void affiche_arbre(Tree *lexique){
    char * mot = malloc(sizeof(char)*512);
    int i = 0;
    afficher(lexique, mot, i);

}
```

Les fonctions suivantes qui permettent la sauvegarde de l'arbre dans l'ordre alphabétique :
void sauvegarde(FILE* fichier, Tree *lexique, char *mot, int i) ;
void save(Tree *lexique, FILE* fichier) ;
l'une prend en paramètre un arbre , un mot et un int
l'autre prend en paramètre un arbre et un fichier FILE.

```c
void sauvegarde(FILE* fichier, Tree *lexique, char *mot, int i){
    if (lexique->value == '\0'){
        strcat(mot, "\n");
        fputs(mot, fichier);
    }
    if (lexique->freregauche != NULL){
        sauvegarde(fichier, lexique->freregauche, mot, i);
    }
    if (lexique->fils != NULL){
        mot[i] = lexique->value;
        mot[i+1] = '\0';
        sauvegarde(fichier, lexique->fils, mot, i+1);
    }
    if (lexique->freredroit != NULL){
        sauvegarde(fichier, lexique->freredroit, mot, i);
    }
}

void save(Tree *lexique, FILE* fichier ){
    char * mot = malloc(sizeof(char)*512);
    int i = 0;
    sauvegarde(fichier, lexique, mot, i);

}
```

---

void save_tree_in_file(Tree *lexique, FILE* fichier) ;
Fonction qui sauvegarde l'arbre sous forme (.DIC). Elle prend paramètre un arbre et un fichier FILE.

```c
void save_tree_in_file(Tree *lexique, FILE* fichier){
    if (lexique == NULL){
       fputc(' ', fichier);
       return ;
    }
    if (lexique->value != '\0'){
        fputc(lexique->value, fichier);
        save_tree_in_file(lexique->freregauche, fichier);
        save_tree_in_file(lexique->fils, fichier);
    }
    else{
        fputc('\n', fichier);

    }
    save_tree_in_file(lexique->freredroit, fichier);

}
```

Guide utilisateur :

Pour faire fonctionné le programme il faut lancé via les commande suivante :
./a.out -S lexique fichier1 (pour créé un nouveau ficher nommé fichier1 contenant l'arbre)
./a.out -l lexique (pour afficher les mots de lexique dans l'ordre)
./a.out -s lexique fichier1 (pour sauvegarder les mots de lexique dans fichier1 en ordre alphabétique)
./a.out -r lexique mot (pour savoir si le mot est présent dans le fichier ou non)

Piste d'Améliorations :

Nous aurions pu faire un menu afin que le programme soit plus propre mais par manque de temps nous avons pas eu le temps.

Difficultés rencontrés :

Le sujet nous pris beaucoup de temps , car nous avons du revoir les notions vu au CM et en TD/TP afin de pouvoir commencer le projet dans les bonne condition.
Nous avons eu du mal à comprend le point 4 du sujet par rapport à la représentation de l'arbre dans le sujet .

Enfin nous avons rencontrer quelque erreur avec la manipulation des arguments mais nous étions de plus en plus à l'aise au fil du projet .

Conclusion :

Pour conclure , ce DM nous a permis de manipuler les arbres pour la première fois dans le contexte d'un mini-projet , cela nous a permis approfondir nos connaissances par rapport à ce qu'on a vu en TP/TD et en cour. Au niveau de l'organisation du travail, nous avons travailler en partage d'écran via discord , cela nous a permis d'échanger facilement sur le projet, et au niveau de la répartition du travail Mikael s'est plus occupé des fonctions de la manipulation de l'arbre tandis que Yasser s'est plus occupé du main et de mettre en place les options. Mais nous avons toujours travaillé ensemble sur discord durant tout le projet . Nous avons trouvé sur projet très intéressant dans la réflexion .