NF16 - TP 4 – Les Arbres Binaires de Recherche

Indexation d'un texte

Introduction

Dans ce TP, nous utiliserons les arbres binaires de recherche pour implémenter un exemple d'indexation et de recherche sur un fichier contenant un texte quelconque.

L'arbre binaire de recherche contiendra tous les mots présents dans le texte à indexer :

- Chaque nœud de l'arbre contient un mot, ainsi que la liste de ses positions dans le texte.
- Une position correspond au numéro de ligne dans laquelle se trouve le mot, son « ordre » dans la ligne (1 pour le premier mot de la ligne, 2 pour le deuxième mot, …etc.), ainsi que le numéro de la phrase dans laquelle il se trouve (1ère phrase du texte, 2ème, 3ème, …etc.).

Dans le cadre de ce TP, on se contentera d'indexer des textes qui ne contiennent que des lettres majuscules, des minuscules et des points (séparateur de phrases).

A. Structures de données

Implémenter les structures de données suivantes :

- La structure **`t_position`** (dont on définira le type synonyme **`t_Position`**) qui comporte les champs :
  - **`numero_ligne`** de type **`int`**
  - **`ordre`** de type **`int`**
  - **`numero_phrase`** de type **`int`**
  - **`suivant`** de type **`t_Position*`**
- La structure **`t_listePositions`** (dont on définira le type synonyme **`t_ListePositions`**) qui représente la liste des positions d'un mot dans le texte. Elle comporte les champs :
  - **`debut`** de type **`t_Position*`**
  - **`nb_elements`** de type **`int`**
- La structure **`t_noeud`** (dont on définira le type synonyme **`t_Noeud`**) qui représente un nœud de l'ABR. Elle comporte les champs :
  - **`mot`** de type **`char*`**
  - **`nb_occurences`** de type **`int`**
  - **`positions`** de type **`t_ListePositions`**
  - **`filsGauche`** de type **`t_Noeud`***
  - **`filsDroit`** de type **`t_Noeud`***
- La structure **`t_index`** (dont on définira le type synonyme **`t_Index`**) qui représente l'ABR dans lequel sont stockés les mots. Cette structure comporte les champs :
  - **`racine`** de type **`t_Noeud`***
  - **`nb_mots_differents`** de type **`int`**
  - **`nb_mots_total`** de type **`int`**

B. Fonctions de base

1. Implémenter une fonction qui crée une liste de positions vide (renvoie NULL si échec).

```
t_ListePositions* creer_liste_positions()
```

2. Implémenter une fonction qui permet d'ajouter un élément dans une liste de positions triées. Les positions d'une liste doivent être uniques et ordonnées dans l'ordre croissant du numéro de ligne et de l'ordre du mot dans la ligne. Cette fonction renvoie 1 en cas de succès, 0 sinon.

```
int ajouter_position(t_ListePositions *listeP, int ligne, int ordre, int
num_phrase)
```

3. Implémenter une fonction qui crée un index vide (renvoie NULL en cas d'échec) :

```
t_Index* creer_index()
```

4. Ecrire une fonction qui recherche un mot dans un index (renvoie NULL si non trouvé) :

```
t_Noeud* rechercher_mot(t_Index *index, char *mot)
```

5. Implémenter une fonction qui permet d'ajouter un nœud dans l'index en respectant les règles d'insertion dans les ABR. Elle renvoie 1 en cas de succès, 0 sinon.

```
int ajouter_noeud(t_Index *index, t_Noeud *noeud)
```

Remarque : *On se basera sur l'ordre lexicographique pour déterminer la position d'un nœud dans l'arbre. Par exemple : avion < bagage < bateau < cabane < cabine <voiture.*
Il faudra faire attention à la casse des mots (lettres minuscules et capitales) : « voiture », « Voiture » et « VoiTURE » doivent être considérés comme étant le même mot.

6. Ecrire une fonction qui permet d'indexer un fichier texte .i.e. lit le fichier et ajoute tous les mots qu'il contient dans un index. Elle renvoie le nombre de mots lus.

```
int indexer_fichier(t_Index *index, char *filename)
```

7. Ecrire une procédure qui affiche la liste des mots classés par ordre alphabétique.

```
void afficher_index(t_Index *index)
```

```
Q
|-- Quis
|---- (l:2, o:16, p:8)
|
|-- Quisque
|---- (l:1, o:0, p:3)
|

R
|-- Risus
|---- (l:1, o:2, p:3)
|

S
|-- Sit
|---- (l:0, o:3, p:0)
|---- (l:2, o:8, p:7)
|---- (l:3, o:6, p:10)
|
```

Figure 1 Exemple d'affichage pour l'index

Remarque : *votre affichage devra se rapprocher le plus possible de l'exemple fourni.*

8. Ecrire une procédure qui permet d'afficher l'ensemble des phrases dans lesquelles se trouve un mot.

```
Void afficher_occurences_mot(t_Index *index, char *mot)
```

Indications :

- *Vous êtes libre de procéder de la façon dont vous souhaitez pour faire cette fonction, mais essayez d'optimiser votre algorithme.*
- *Si vous voulez vous pouvez ajouter des structures supplémentaires pour vous aider à améliorer votre algorithme. Vous pouvez même modifier les structures demandées plus haut.*
- *Par exemple, vous pouvez décider d'ajouter un second index pour trouver plus rapidement les phrases.*

```
Mot = "Rhoncus"
Occurences = 5
| Ligne 6, mot 30 :  Ut turpis est maximus non turpis et elementum rhoncus arcu.
| Ligne 8, mot 8 :  Duis rhoncus bibendum ornare.
| Ligne 10, mot 6 : Pellentesque non quam eu odio convallis rhoncus sed eget massa.
| Ligne 10, mot 13 :  Mauris sit amet rhoncus ipsum vel viverra eros.
| Ligne 11, mot 5 : Morbi pretium quam nec luctus rhoncus.
```

Figure 2 : Exemple d'affichages pour la question 8

9. Ecrire une fonction qui permet d'équilibrer l'arbre d'un index. Elle renvoie un pointeur vers le nouvel index équilibré.

```
t_Index* equilibrer_index(t_Index *index)
```

Remarques et indications :

- *On ne demande pas d'équilibrer directement l'index donné en paramètre (en faisant des rotations par exemple). La fonction utilise l'index donné « index » pour en construire un second « index2 » qui lui est équilibré à la construction.*
- *Si l'index est déjà équilibré, il ne faut rien faire.*
- *N'oubliez pas libérer la mémoire de l'ancien index.*
- *Vous pouvez regarder du côté de la recherche dichotomique pour écrire cet algorithme.*

C. Programme Principal :

Programmer un menu qui propose les fonctionnalités suivantes :

1. **Charger un fichier**: lire un fichier et le charger dans un index puis afficher le nombre de mots lus dans le fichier.
2. **Caractéristiques de l'index** : Afficher les caractéristiques de l'index et si oui ou non il est équilibré.
3. **Afficher index :** Afficher les mots contenus dans l'index en respectant le format demandé.
4. **Rechercher un mot** : Afficher le n° de ligne, l'ordre dans la ligne, le n° de phrase pour chaque occurrence.
5. **Afficher les occurrences d'un mot :** Recherche un mot dans l'index et affiche toutes les phrases dans lesquelles il apparait.
6. **Equilibrer l'index**
7. **Quitter :** La mémoire allouée dynamiquement doit être libérée.

NF16 - TP 4 – Les Arbres Binaires de Recherche

Consignes générales :

Sources

- À la fin du programme, les blocs de mémoire dynamiquement alloués doivent être proprement libérés. Vous devrez également être attentifs à la complexité des algorithmes implémentés.
- L'organisation MINIMALE du projet est la suivante :
  - Fichier d'en-tête tp4.h, contenant la déclaration des structures/fonctions de base,
  - Fichier source tp4.c, contenant la définition de chaque fonction,
  - Fichier source main.c, contenant le programme principal.

Rapport

Votre rapport de quatre pages maximum contiendra :

- La liste des structures et des fonctions supplémentaires que vous avez choisi d'implémenter et les raisons de ces choix.
- La description de la solution choisie et des algorithmes implémentés pour les questions B.8 et B.9
- Un exposé succinct de la complexité de chacune des fonctions implémentées.
- Votre rapport et vos fichiers source feront l'objet d'une remise sur Moodle dans l'espace qui sera ouvert à cet effet quelques jours suivant votre démonstration au chargé de TP (un seul rendu de devoir par binôme).