Université de Technologie de Compiègne

NF16

Rapport de TP

4 - Les ABR

Automne 2014

Romain PELLERIN - Brianda PIZAÑO
Groupe de TP 2
5 janvier 2015

Table des matières

1 Introduction

1.1 Résumé

L'objectif de ce TP est d'écrire un programme en C qui utilisera les propriétés des arbres binaires de recherche dans le but d'implémenter un exemple d'indexation et de recherche sur un fichier contenant un texte quelconque.

L'arbre binaire contiendra tous les mots présents dans le texte.

— chaque nœud de l'arbre contient un mot, ainsi que la liste des positions du mot dans le texte

— une position correspond à un numéro de ligne, un ordre **dans la phrase** (1 pour le premier mot de la phrase, 2 pour le deuxième mot, etc...), un numéro de phrase

— le texte ne contient que des majuscules, des minuscules et des points (séparateur de phrases)

1.2 Structuration du projet

Nous avons choisi de structuer notre projet en 7 fichiers pour une meilleure clarté :

— **main.c** : le fichier source contenant le programme principal

— **liste.h et liste.c** : les fichiers contenant les structures et fonctions propres aux listes de positions des mots

— **outils.h et outils.c** : les fichiers contenant les fonctions de lecture du fichier (ajout d'un mot dans l'arbre,) et les fonctions qui permettent des créer des listes chaînées de `struct phrase` et des listes chaînées de `struct mot`, utilisées pour retrouver les phrases contenant deux mots choisis par l'utilisateur (option 6. du menu proposé à l'utilisateur)

— **arbre.h et arbre.c** : les fichiers contenant les structures et fonctions propres aux arbres et aux nœuds.

1.3 Code source

Le code source est fourni dans l'archive ZIP contenant ce rapport.

2 Implémentation

Les structures et fonctions demandées dans le sujet ont été implémentées, avec quelques spécificités selon la compréhension du sujet. Par exemple, certains vérifications ont été ajoutées (vérification qu'un arbre ou qu'un nœud n'est pas `NULL`). De plus, un mot est caractérisé par son numéro de ligne, son numéro de phrase et son ordre. **Nous avons implémenté cela de telle sorte que l'ordre corresponde à l'ordre dans la phrase, et non à l'ordre dans la ligne.**

2.1 Programme principal

Le programme principal affiche à l'utilisateur un menu qui lui donne le choix parmi **7 possibilités** :

1. Créer un ABR
2. Charger un fichier dans l'ABR
3. Afficher les caractéristiques de l'ABR
4. Afficher tous les mots distincts par ordre alphabétique
5. Rechercher un mot
6. Afficher les phrases contenant deux mots
7. Quitter le programme

L'utilisateur tape au clavier le numéro de son choix. Un `switch` permet d'effectuer les actions correspondantes à l'entrée clavier.

main.c

```
// ...
afficherMenu();
while((inputUser = getchar()) != '7') {
  // ...
  switch(inputUser) {
    case '1':
        // ...
        break;
    case '2':
        // ...
        break;
    case '3':
        // ...
        break;
    case '4':
        // ...
        break;
    case '5':
        // ...
        break;
    case '6':
        // ...
```

```
            break;
        default:
            printf("Mauvaise entrée. Veuillez taper 1, 2, 3, 4, 5, 6 ou 7.\n");
            continue; // On recommence la boucle while
    }
    // ...
    afficherMenu();
}
// ...
```

Une boucle `while` permet d'afficher à nouveau le menu après être sorti du `switch`. L'invariant de boucle est `(inputUser = getchar())!= '7'`.

2.2 Autres fichiers

Les autres fonctions ont été écrites comme demandées.. Pour l'affichage de l'arbre, l'algorithme de **parcours infixe** vu en cours a été implémenté. Il en va de même pour les algorithmes **d'ajout d'un nœud à l'arbre** en tenant compte de l'ordre des mots, de **recherche d'un nœud dans l'abre** et pour l'algorithme qui **retourne la hauteur d'un arbre**.

2.2.1 Ajouts par rapport au sujet

`struct mot` **et** `struct phrase`

Par rapport à ce qui a été demandé, nous avons choisi d'ajouter deux structures (deux listes chaînées) :

outils.h

```
typedef struct mot {
    char* mot;
    struct mot *suivant;
} Mot;

typedef struct phrase {
    int numero;
    Mot *debut;
    struct phrase *suivante;
} Phrase;
```

Ces deux listes chaînées sont remplies au fur et à mesure que l'on *parse* (parcourt) le fichier texte lu, grâce à plusieurs **fonctions que nous avons ajoutées** dans les fichiers **outils.c** et **outils.h**. Ainsi, lorsque l'utilisateur choisi deux mots (option 6 du menu), grâce à la fonction `NoeudABR *rechercher_noeud(ArbreBR *arbre, char *mot)` on peut aisément retrouver les numéros de phrase où les deux mots sont présents. Puis l'affichage de la ou des phrases est rendu très simple grâce à nos deux listes chaînées.

outils.c

```
void afficher_phrase(Phrase *phrase, int numero) { // La phrase 1 est passée en
    paramètre, elle pointe sur la 2, qui pointe sur la 3, etc
  while(phrase != NULL) {
    if (phrase->numero == numero) { // Si c'est la bonne phrase, on l'affiche
      Mot *mot = phrase->debut;
      while (mot != NULL) {
```

```
        printf("%s ",mot->mot);
        mot = mot->suivant;
      }
    }
    phrase = phrase->suivante;
  }
  printf(".\n");
}
```

Autres ajouts

Dans **arbre.h** et **arbre.c**, nous avons ajouté d'autres fonctions, dont le nom est suffisamment évoquant pour ne pas être expliquéees :

- `void parcours_infixe(NoeudABR *n)`
- `void supprimerNoeud(NoeudABR *noeud)`
- `int max (int a, int b)`
- `int getHauteur(NoeudABR *noeud)`
- `int isEquilibre(NoeudABR *n)`

Quant aux fichiers **liste.h** et **liste.c**, seule la fonction `void supprimerPosition(Position *pos)` a été ajoutée. Elle supprime toutes les positions d'un mot.

2.3 Difficulté rencontrée

La difficulté majeure rencontrée dans ce TP a été l'implémentation de la fonction qui lit un fichier et stocke chaque mot dans un nœud de l'arbre. Il faut en effet être capable de détecter la fin d'un mot, lorsque l'on rencontre soit un espace, soit un point, soit un retour à la ligne. De plus, si je rencontre un espace ou un retour à la ligne alors que le caractère précédent était un point, il ne faut pas ajouter de mot. Il fallait également prendre en compte le fait qu'un fichier ne finit pas nécessairement par un point. Enfin, sur Windows, le caractère de saut de ligne est `\r\n` alors que sur les systèmes UNIX c'est `\n`. Il fallait donc prendre en considération tous ces aspects.

ATTENTION : L'ordre d'un mot correspond à l'ordre dans la phrase, et non à l'ordre dans la ligne.

Au final, la fonction est plutôt conséquente :

outils.c

```
int charger_fichier(ArbreBR *arbre, char *filename) {
    if (arbre == NULL)
        return 0;

    FILE *fp;
    int c, c_old;

    int ligne = 1, ordre = 1, phrase = 1;
    char *mot = (char*) malloc(sizeof(char)*26);
    int counter = 0, total = 0;

    fp = fopen(filename, "r");
    if (fp == NULL) {
```

```
        perror("Le fichier ne peut pas être ouvert.");
        return 0;
    }
    do {
        c_old = c;
        c = fgetc(fp);
        if (feof(fp)) // END OF FILE
            break;

        if (c == EOF || c == '\n') {
            if (c_old != '.' && c_old != ' ') {
                total++;
                mot[counter] = 0;
                saveWord(arbre,mot,ligne,ordre,phrase);
                ordre++;
            }
            ligne++;

            counter = 0;
            mot = (char*) malloc(sizeof(char)*26);
        }
        else if (c == ' ') {
            if (c_old != '.') {
                total++;
                mot[counter] = 0;
                saveWord(arbre,mot,ligne,ordre,phrase);
                ordre++;
            }

            counter = 0;
            mot = (char*) malloc(sizeof(char)*26);
        }
        else if (c == '.') {
            mot[counter] = 0;
            saveWord(arbre,mot,ligne,ordre,phrase);
            counter = 0;
            total++;
            ordre = 1;
            phrase++;
        }
        else {
            if (c == '\r') {
                c = c_old;
                continue; // WINDOWS
            }
            mot[counter] = c;
            counter++;
        }
    }while(1);

    if (c_old != '.' && c_old != ' ' && c_old != '\n') {
```

```
        saveWord(arbre,mot,ligne,ordre,phrase);
        total++;
    }

    fclose(fp);
    return total;
}
```

3 Capture d'écran

Voici un aperçu de notre programme :

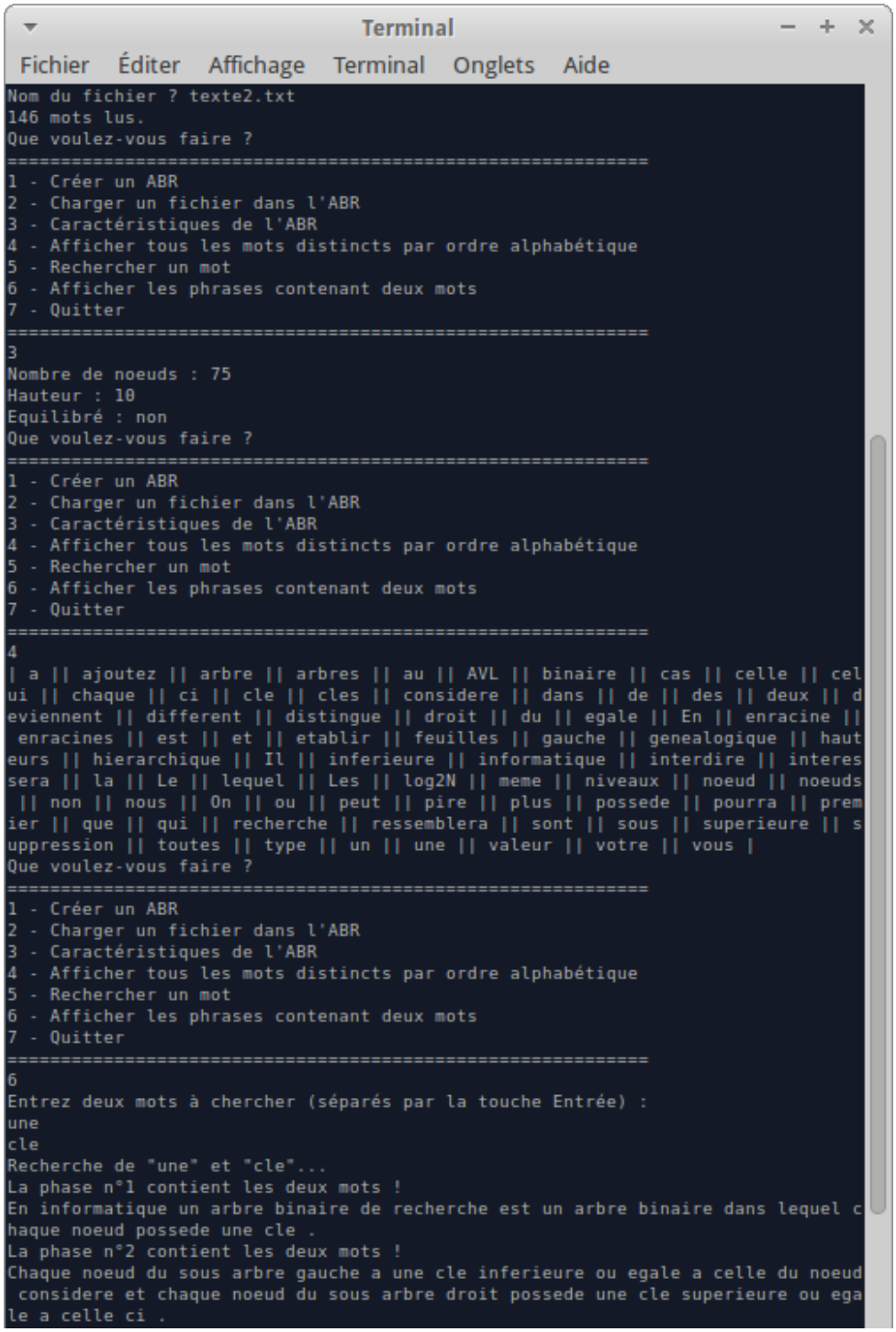

FIGURE 3.1 – L'utilisateur a choisi 1, 2, 3, 4 et enfin 6 dans le menu d'options