Rapport projet Dictionnaire Algorithmique Avancée L3-S5

But du projet

Ecrire un un dictionnaire en C, c'est à dire un programme permettant de stocker et chercher des mots, en utilisant un arbre n-aire, modélisé par un arbre binaire de recherche.

On doit pouvoir disposer des fonctionnalités suivantes :

- Ajout d'un mot
- Suppression d'un mot
- Recherche d'un mot
- Sauvegarde du dictionnaire dans un fichier
- Restitution d'un dictionnaire à partir d'un fichier
- Quitter le programme

Code

libdico.c

`libdico.c` contient les fonctions de manipulation du dictionnaire.

```
#include "./libdico.h"
#define DEBUG 1
/*
Algo  :

Check lettre par lettre :
Si left->car = word[c], c++ et currentNode = currentNode->left;
Si left-car après currentNode->left, currentNode=currentNode->right;
Si left->car avant currentNode->left, 'faut mettre leftNode->left dans ->right et mettre le caractere actuel en tant que nouveau left
*/
Dictionnary addWord(Dictionnary d, char* word){
  if(word[0] != '\0'){
    if(NULL == d){
      d = malloc(sizeof(Dictionnary)); //On alloue la memoire a d.
      d->car = word[0]; //On y met le premier caractere de la chaine
      word++; //on enlève le premier caractere de la chaine
      d->left = addWord(d->left, word); //On fait la meme chose sur le fils droit.
    }else
    if(word[0] < d->car){
      char tmp = d->car;
      d->car = word[0];

      Dictionnary newNode = createDictionnary(); //New 'old' node
      newNode = malloc(sizeof(Dictionnary));
      newNode->car = tmp;
      newNode->left = d->left;
      newNode->right = d->right;
      word++;
      d->right = newNode;
      d->left = NULL;
      d->left = addWord(d->left, word);

    }else
    if(word[0] > d->car){
      d->right = addWord(d->right, word);
    }else
    if(word[0] == d->car){
      word++;
      d->left = addWord(d->left, word);
    }
  }else{
    d = malloc(sizeof(Dictionnary));
    d->car = '*';
    return d;
  }
return d;
}

Dictionnary createDictionnary(){
  return NULL;
}
```

```c
int dictionnaryEmpty(Dictionnary d){
  return  d==NULL;
}
void displayLeft(Dictionnary d){ //Retourne le mot le plus "à gauche"
  Dictionnary cur = d;
  int i=0;
  while(cur->right != NULL ){
    printf("Caractere no %d : %c\n", i, cur->car);
    cur = cur->right;
    i++;
  }
}
void prefix(Dictionnary d){ //Parcours préfixe du dictionnaire
   printf("%c\n", d->car);
  if(d->left!=NULL){
    prefix(d->left);
  }
  if(d->right!=NULL){
    prefix(d->right);
  }

}
int getHeight(Dictionnary d){
  if(d == NULL){
    return -1;
  }

  int left = getHeight(d->left);
  int right = getHeight(d->right);

  if(left > right){
    return left+1;
  }else{
    return right+1;
  }
}

int belongs(Dictionnary d, char* word){ //Voit si un mot appartient ou non au dictionnaire.
  printf("Lettre courante du dico %c, lettre du mot %c", d->car, word[0]);
  if(word[0] != '*'){
    if(d==NULL){
      return 0;
    }
    if(d->car == word[0]){
      word++;
      return belongs(d->left, word);
    }
    if(word[0] < d->car){
      return 0;
    }
    if(word[0] > d->car){
      return belongs(d->right, word);
    }
  }
  return d->car == word[0];
}
}
```

libdico.h

```c
/*PROJET Dictionnary : "L'épicerie des mots"*/
/*Prototype des fonctions, include et autres structures*/
#ifndef LIBDICO_H
#define LIBDICO_H

#include <stdio.h>
#include <stdlib.h>

typedef struct node{
  char car;
  struct node *left, *right;


}*Dictionnary;

/*debug*/
void prefix(Dictionnary);
void displayLeft(Dictionnary);
//Prototypes des fonctions d'opérations sur le Dictionnary
Dictionnary createDictionnary();
Dictionnary addWord(Dictionnary, char*);
int dictionnaryEmpty(Dictionnary);
int getHeight(Dictionnary);
int belongs(Dictionnary, char*);

//Prototypes des autres fonctions relatives au Dictionnary
void leave();

//"L'Asie est un bretzel, la vie n'est pas un bretzel." - Arthur Rimbaud

#endif
```

libio.c

`libio.c` contient les fonctions d'input/output, de sauvegarde et de chargement du dictionnaire.

```c
#include "libio.h"
```

```c
#define NULL_MARKER '#'
int save(Dictionnary d, char *path){ //Sauve l'arbre d dans le path donné
  FILE *fp = fopen(path, "w");
  if(fp == NULL){
    return 1;
  }
  saveDictionnary(d, fp);
  return 0;
}

Dictionnary load(Dictionnary d, char *path){ //Charge le dictionnaire depuis le fichier
  FILE *fp = fopen(path, "r");
  if(fp == NULL){
    return NULL;
  }
  loadDictionnary(d, fp);
  return d;
}

Dictionnary loadDictionnary(Dictionnary d, FILE *fp){
  char buf;
  if(d == NULL){
    d=createDictionnary();
    d=malloc(sizeof(Dictionnary));
  }
  if(!fscanf(fp, "%c", &buf) || buf == '#'){
    return NULL;
  }else{
    d->car = buf;
    printf("SV: %c\n", d->car);
    d->left = loadDictionnary(d->left, fp);
    d->right = loadDictionnary(d->right, fp);
    return d;
  }
}

void saveDictionnary(Dictionnary d, FILE *fp){
  if(d==NULL){
    fprintf(fp, "#");
  }else{
    fprintf(fp, "%c", d->car);
    saveDictionnary(d->left, fp);
    saveDictionnary(d->right, fp);
  }
}
```

libio.h

```c
#ifndef LIBIO_H
#define LIBIO_H
#include "./libdico.h"
#include "./libstack.h"
#include <string.h>

Dictionnary load(Dictionnary d, char *path);
int save(Dictionnary d, char *path);
Dictionnary parseStr(Dictionnary d, char *str, Stack s);
char *serialize(Dictionnary d);
void readTree(Dictionnary d);
Dictionnary writeTree(Dictionnary d, char *str);
void saveDictionnary(Dictionnary d, FILE *fp);
Dictionnary loadDictionnary(Dictionnary d, FILE *fp);
#endif
```

libstack.c

Nous avons pensé à utiliser une pile afin de pouvoir proposer un parcours efficace pour la sauvegarde de l'arbre. Par faute de temps, et la solution optimisée n'étant pas fonctionnelle, celle-ci n'a pas été utilisée.

```c
#include "libstack.h"

Stack createStack(){
  return NULL;
}

Stack push(Stack s, Dictionnary d){
  if(s->dict == NULL){
    s->dict = d;
    return s;
  }
  Stack newNode = createStack();
  newNode = malloc(sizeof(Stack));
  newNode->dict = d;
  newNode->nxt = s;
  return newNode;
}
Stack pop(Stack s){
  return s->nxt;
}
```

libstack.h

```
#ifndef LIBSTACK_H
#define LIBSTACK_H

#include "libdico.h"
typedef struct nodestack{
  Dictionnary dict;
  struct nodestack *nxt;
} *Stack;

Stack createStack();
Stack push(Stack s, Dictionnary d);
Stack pop(Stack s);

#endif
```

mainCli.c

Le menu est codé ici.

```
#include <stdio.h>
#include "libdico.h"
#include "libio.h"

int main(){
    Dictionnary d = createDictionnary();
    d = malloc(sizeof(Dictionnary));
    //d = addWord(d, "vomi");
    //d = addWord(d, "relent");
    //prefix(d);
    while(1){
        char word[20];
        int choice;
        printf("1Ajouter mot\n2Afficher contenu du dico\n3Testerl'appartenance d'un mot\n4Vider le dico\n5Sauvegarder contenu dico dans un fichier\n6Charger un dico à partir d'un

        scanf("%d", &choice);
        switch(choice){
            case 1:
                scanf("%s", word);
                d = addWord(d, word);
                printf("cbon\n");
                break;
            case 2:
                if((d==NULL) || (d->car=='*')){
                    printf("le tableau il est vide\n");
                }else{
                prefix(d);}
                break;
            case 3:
                scanf("%s", word);
                //strcpy(word, "*");
                printf("%d", belongs(d, word));
                break;
            case 4:
                d->car='*';
                d->left = d->right = NULL;
            case 5:
                //scanf("%s", word);
                save(d, "./test");
                printf("Le dico est save ++\n");
                //return 0;
                break;
            case 6:
                load(d, "./test");
                break;
            case 7:
                return 0;
            default:
                return 0;
        }

    }
}
```

Makefile

```
#Makefile du projet : "L'épicerie des mots"

all : epicerieDesMots

epicerieDesMots: libdico.o libstack.o libio.o mainCli.o
    gcc -o epicerieDesMots libdico.o libstack.o libio.o mainCli.o

libdico.o: libdico.c libdico.h
    gcc -o libdico.o -c libdico.c

libstack.o: libstack.c libstack.h
    gcc -o libstack.o -c libstack.c

libio.o: libio.c libio.h
    gcc -o libio.o -c libio.c

mainCli: mainCli.c libstack.h libio.h libdico.h
        gcc -o mainCli.o -c mainCli.c
```

```
clean:
	rm -f *.o epicerieDesMots
```

Problèmes rencontrés

Nous avous éprouvé des difficultés sur la suppression des mots, ce pourquoi cette fonction n'a au final pas été implémentée. Nous avons passé du temps à essayer d'optimiser les algorithmes et à les réfléchir, ce qui a eu pour conséquence de nous prendre de court. Le projet Dictionnaire a été un des plus complexes des trois, car les algorithmes étaient à adapter afin de pouvoir implémenter les arbres n-aires à l'aide d'arbres binaires de recherche.