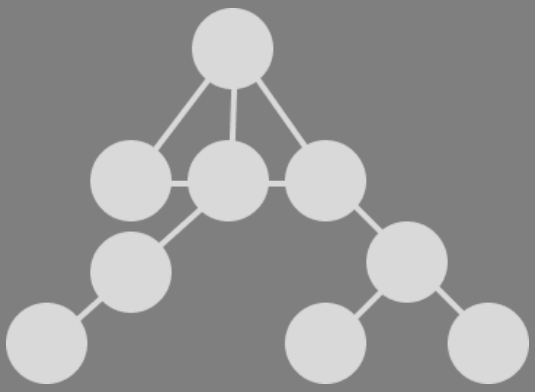
Générateur automatique de phrases

Projet C – Algorithmique et structure de données 2



Mattéo BONNET

Anthony CAO

Matthieu BRANDAO

PARIS PANTHÉON-ASSAS UNIVERSITÉ

SOMMAIRE

Introduction

Durant ce semestre de 2eme année à l'EFREI, nous avons suivi un cours intitulé « Algorithmique et structure de données 2 ». Durant celui-ci nous avons appris une toute nouvelle structure de données : Les arbres. Une structure demandant des acquis sur les pointeurs ainsi que sur le chainage de structures défini préalablement.

Afin de mettre en avant nos nouvelles compétences dans le langage C et d'appliquer les cours suivis, le département d'informatique a demandé, par équipes de 3 étudiants, de construire un projet nommé « Générateur de phrases ».

Ce projet possède sa note de cadrage dont sort quelques directions principales : Le projet doit être codé en C, Utiliser des arbres binaires ou n-aires, Utiliser et extraire les données d'un fichier texte donné en même temps que le projet, et enfin être capable de stocker et chercher dans les arbres pour créer des phrases d'après un modèle.

Nous avons donc créé une équipe constituée de Mattéo Bonnet, Anthony Cao et Matthieu Brandao. Et nous sommes lancés pour la durée d'un mois sur ce projet.

Présentation fonctionnelle du projet

I. Fonctionnalités réalisées

La recherche d'un mot dans la base de données :

Principe : Cette fonctionnalité permet de rechercher dans les arbres (notre base de données) si un mot existe.

Utilisation : Nous entrons le mot souhaité. Le programme parcourt les arbres et nous retourne ce qu'il trouve.

Le retour : Dans le cas où le mot a été trouvé, liste de pointeurs sur une structure de type « node » sera retourné. On pourra alors accéder au mot quand on en a besoin en utilisant juste le pointeur récupéré. Dans le cas contraire, mot inexistant, un pointeur à NULL est retourné.

Recherche aléatoire :

Principe : Cette fonctionnalité permet de rechercher dans la base de données un mot aléatoirement.

Utilisation : Il suffit de lancer la fonctionnalité. Le programme recherche seul. Aucune entrée est demandée.

Le retour : Un mot de type quelconque est retourné sous forme d'un pointeur sur une structure de type « node » Il pourra être récupéré ou affiché.

Génération de phrases constituées de formes de bases :

Principe : Cette fonctionnalité permet de construire une phrase à partir des mots dans la base de données.

Utilisation : Il suffit de lancer la fonctionnalité. Le programme, d'après des modèles prédéfinis, cherchera aléatoirement, un par un, les mots de la phrase dans l'arbre correspondant à leur type. Une phrase complète, constitué de tous les types d'un modèle, est retournée. Attention les mots sont tous dans leur forme de base.

Le retour : Une phrase, de type chaine de caractères, pouvant être récupérée et affichée.

Génération de phrases constituées de formes de fléchis :

Principe : Cette fonctionnalité permet de construire une phrase à partir des mots dans la base de données.

Utilisation : Il suffit de lancer la fonctionnalité. Le programme, d'après des modèles prédéfinis, cherchera aléatoirement, un par un, les mots de la phrase dans l'arbre correspondant à leur type. Il finira par appliquer aléatoirement une forme fléchie du mot. Une phrase complète, constitué de tous les types d'un modèle, est retournée.

Le retour : Une phrase, de type chaine de caractères, pouvant être récupérée et affichée.

Spécificité : Ici on pourra récupérer des phrases constituées de nouveaux types de mots, par rapport à la fonctionnalité précédente. Par exemple les déterminants.

Détails d'un mot fléchi :

Principe : Cette fonctionnalité a pour but de renseigner les caractéristiques d'un mot fléchi à la condition qu'il soit présent dans la base de données.

Utilisation : Après lancement de la fonctionnalité. Il est demandé d'entrer un mot. Le programme cherchera le mot dans tous les arbres. Et finira par retourner les détails sur ce mot.

Le retour : Dans le cas où le mot est trouvé, une ou plusieurs chaines de caractères (contenant forme de base, conjugaison, genre, type du mot) seront retournées dans la console. Dans le cas contraire NULL sera retourné.

II. Fonctionnalités ajoutées

Libération de tous les arbres :

Principe : Cette fonctionnalité a été conçue pour vider notre base de données. Elle peut être utile dans le cas d'une refonte de la base, lorsqu'on s'aperçoit d'une erreur de stockage dans la base ou à la fermeture du programme pour ne pas laisser de mémoire inutilisable.

Utilisation : Il suffit de lancer la fonctionnalité. Le programme parcourra l'ensemble des arbres en libérant (free()) chaque nœud et les formes fléchies associées.

Présentation technique du projet

I. Choix des structures de données

Tout d'abord un « root » principal (structure de type « trees ») a été créé, il contient les pointeurs vers les « root » de chaque arbre correspondant aux types des mots (Adjectifs, Noms, Verbes, Adverbes, Pronoms, Déterminants, Conjonctions, Prépositions, Interjections).

Les nœuds suivants seront tous du même type (structure « node ») constitué de la lettre, d'un tableau de fléchis, du nombre de fléchis et d'un tableau* listant tous les nœuds enfants.

Les racines de chaque arbre pointé par le root principal n'ont pas de lettre, ils servent juste comme point d'entrée.

La structure d'un mot fléchi se décrit par le mot lui-même, son mot de base, ses caractéristiques et le nombre de caractéristique.

* Il est constitué de 28 cellules, toutes à NULL par défaut. Au fur et à mesure du remplissage des mots, les cellules du tableau se remplissent en fonction du code ASCII du caractère à ajouter. Les nouvelles cellules seront des pointeurs vers le nœud correspondant à la lettre.

Afin de déterminer un mot de base, il suffit de parcourir l'un des arbres et de s'arrêter lorsqu'un nœud à un nombre de fléchis supérieur à 0.

Pour plus de clarté :

```
typedef struct s_flechie {
   char *word;
   char *baseword;
   char **tab_cara; int n_cara;
} flechie;

typedef struct s_node {
   char letter;
   flechie **tab;
   int n_flechies;
   struct s_node **alphabet;
} node, *p_node;
```

```
typedef p_node tree;

typedef struct s_trees {
   tree tree_adj;
   tree tree_nom;
   tree tree_ver;
   tree tree_adv;
   tree tree_pro;
   tree tree_pre;
   tree tree_det;
   tree tree_int;
   tree tree_con;
   tree tree_qpro;
} trees;
```

Une schématisation dynamique de la structure est aussi disponible sur ce lien :

https://www.figma.com/proto/824NEXGF4k6YFVmYpihzkr/Tree---C-Project?page-id=0%3A1&node-id=22%3A14&viewport=259%2C59%2C0.14&scaling=scale-down&starting-point-node-id=22%3A14

II. Les principales fonctions

Recherche de mots

search_word() : En utilisant une entrée donnée, retourne une liste de « p_node » dont le mot de base est égal à l'entrée.

- Déclaration d'un « p_node » temporaire (pour naviguer dans chaque arbre).
- Parcoure des nœuds (caractères du mot) autant de fois que la taille de la recherche.
- Arrêt de la recherche si on ne peut pas continuer de naviguer dans l'arbre.

Extraction d'une forme de base

random_word() : Tout en sachant l'arbre à parcourir, retourne un « p_node » aléatoire valide (ayant un mot de base et des formes fléchies).

- Navigue dans l'arbre jusqu'à arriver à un « node » avec des mots fléchis.
- Si l'on peut continuer, on donne 1/2 chance de continuer
- Sinon on arrête la navigation.

is_alphabet_empty() : Reçoit l'alphabet d'un « node », et retourne une valeur en fonction de si le tableau est vide.

- Parcourt chaque case de l'alphabet et vérifie si chacune vaut NULL.
- Retourne 1 si toutes les valeurs de l'alphabet sont NULL (donc impossible de naviguer plus profondément dans l'arbre), sinon 0.

Génération des phrases

gen_phrase_nodes() : Crée une liste dynamique de « p_node ». La liste et le type de ses « p_node » suit un des modèles prédéfinis, mais ces « p_node » sont aléatoires.

- On choisit un nombre aléatoire pour le modèle prédéfini.
- Ajoute ensuite les « p_node » choisis aléatoirement en fonction du type de chaque mot du modèle.

gen_phrase_flechie() : Génère une phrase correcte grammaticalement, en suivant un modèle prédéfini de phrase. La fonction annonce les formes fléchies potentiellement manquantes.

- Pour chaque « p_node » de la liste, on ajoutera un/deux mot.s fléchi.s selon le type du mot :
  - Nom : déterminant + nom accordés
  - Adjectif : accord avec le nom précédent
  - Verbe : temps à l'indicatif et accord avec le nom précédent
- Sinon récupère le premier mot fléchi (interjection par exemple).

Détails d'un mot (type, conjugaison, genre, ..)

search_flechie() : Recherche toutes les caractéristiques du mot cherché. La fonction retourne une liste de pointeurs vers une structure « fléchie » correspondants chacun au mot fléchi donné.

- Pour chaque arbre, la fonction utilise une autre fonction récursive qui parcourra l'arbre et donnera les structures fléchies (qui ont été déclarées dans tree.h) correspondantes au mot fléchi donné.

III. Difficultés rencontrées et solutions

Tout d'abord nous avons éprouvé des difficultés pour comprendre le sujet. Effectivement la complexité du projet et la compréhension de la structure de donnée attendue fut un obstacle au début. Nous avons finalement pu programmer la structure comme présentée précédemment.

Lors du lancement du programme sur nos différents ordinateurs, nous avons pu remarquer des différences de résultats entre eux. Pour certains un affichage défaillant, pour d'autres une mémoire ne supportant pas les opérations. Un travail de recherche et de débogages fut donc appliqué. En cette fin de projet le programme marche sur n'importe quel support/IDE.

Nous avons commencé, comme grand nombre de nos camarades, à essayer notre code avec le dictionnaire de mots non accentués. Or nous avons quand même rencontré des mots possédants des caractères spéciaux. Aujourd'hui le programme marche avec ces caractères (sauf le ö), le problème ne se pose donc plus.

Enfin, nous avons trouvé que la date de rendu était trop courte. Certaines fonctionnalités demandent beaucoup de temps à développer, ce qui a précipité l'écriture du programme pouvant altérer la qualité finale du projet.

Présentation des résultats

I. Présentation de l'interface utilisateur

Menus :

```
0) Leave
1) Generate a complete sentence
2) Search with flechi words
3) Other
```

->

```
0) Back to main menu
1) Search with base words
2) Random search into noun tree
3) Generate a sentence with base words
```

Exemples de retours utilisateur :

Recherche d'un mot fléchi donné :

```
search:belle
Results:
belle : Adj beau, Fem SG
belle : Adj bel, Fem SG
belle : Nom belle, Fem SG
```

Ou

```
search:abc
No results found
```

Création d'une phrase (avec des mots fléchis) :

```
cette maizena noetique kidnappait lesquels b
```

Recherche d'un mot de base donné :

```
search:programmation
Array of p_node:
Pointer[@] -> Node {letter: 'n', tab_flechis: Pointer[@ = 000002cb486c8290] -> Array of flechi (size: 2), alphabet: []}
```

Recherche aléatoire d'un mot :

```
Pointer[@] -> Node {letter: 'a', tab_flechis: Pointer[@ = 000002cb3e19ec00] -> Array of flechi (size: 1), alphabet: [[@] -> Node {'b'}; [@]
 -> Node {'c'}; [@] -> Node {'d'}; [@] -> Node {'e'}; [@] -> Node {'f'}; [@] -> Node {'g'}; [@] -> Node {'h'}; [@] -> Node {'i'}; [@] -> No
de {'j'}; [@] -> Node {'k'}; [@] -> Node {'l'}; [@] -> Node {'m'}; [@] -> Node {'n'}; [@] -> Node {'o'}; [@] -> Node {'p'}; [@] -> Node {'q
'}; [@] -> Node {'r'}; [@] -> Node {'s'}; [@] -> Node {'t'}; [@] -> Node {'u'}; [@] -> Node {'v'}; [@] -> Node {'x'}; [@] -> Node {'y'}; [@
] -> Node {'z'}; ]}
```

Création d'une phrase (avec des mots de bases) :

```
nu gothique piauler c
```

II. Données de reproduction des tests

Création d'une phrase (avec des mots fléchis) :	Run/1
Recherche d'un mot fléchi donné :	Run/2/{word to find}
Recherche d'un mot de base donné :	Run/3/1/{word to find}
Recherche aléatoire d'un mot :	Run/3/2
Création d'une phrase (avec des mots de bases) :	Run/3/3

Conclusion

Ce projet fut formateur sur deux principaux axes.

Le premier est l'organisation en équipe. Effectivement nous avons dû créer une stratégie pour coder bien et surtout atteindre nos objectifs. Nous avons commencé par essayer de se répartir les tâches avant de se rendre compte que nous serions plus productifs en « session de code » ensemble. Nous avons alors commencé à nous retrouver régulièrement pour faire un point d'avancement et programmer un maximum en groupe.

Le second est le langage C et la structure en arbre. Le fait de devoir traiter, découper, et stocker un fichier texte extérieur à notre code fut une nouveauté pour nous en C. Nous nous sommes donc formés sur internet. De plus la structure en arbres utilisée pour la base de données était à la fois nouvelle et encore abstraite pour nous. À travers ce projet nous avons donc mieux compris son fonctionnement.

Matthieu Brandao

Mattéo Bonnet

Anthony Cao