Projet d'Algorithmes Avancés

Athène Rousseau Rambach

Adrien Verstrepen

Table des matières

Problème

Le projet que nous avons réalisé répond à plusieurs objectifs. Le premier est de lire un fichier avec l'extension .phy ou .txt donné par l'utilisateur afin de vérifier que l'arbre contenu dedans suit bien la grammaire précise qui nous a été donnée.

Ensuite, le second objectif est de construire l'arbre phylogénétique correspondant à l'arbre extrait du fichier en utilisant une structure définie en C que nous aurions créée.

Le dernier objectif est de sauvegarder l'arbre ainsi créé sous le format reconnu par la grammaire dans un nouveau fichier et d'afficher dans la console l'arbre suivant le format de la grammaire, ainsi qu'une autre représentation de notre choix.

L'objectif principal est donc de développer une application en langage C qui réalise des arbres phylogénétiques à partir d'un document en utilisant une programmation par la grammaire.

Équipe

L'équipe pour ce projet se compose d'Athène Rousseau Rambach et d'Adrien Verstrepen, étudiants de deuxième année de Licence Sciences du Numérique, de l'École du Numérique de la FGES.

Athène s'est impliquée dans la partie création de la structure d'arbre ainsi que la partie affichage car ce sont deux éléments qu'elle avait beaucoup aimé réaliser lors du projet de création d'arbres phylogénétiques que nous avons eu l'occasion de réaliser lors de notre premier semestre en cours de Structures de Données Avancées.

Adrien s'est impliqué dans les parties de vérification de la grammaire ainsi que les tests des différentes fonctions. Il a choisi ces parties car il avait bien compris la partie sur la vérification de grammaire en cours et avait envie d'approfondir ses compétences sur ce sujet en assurant cette partie du projet.

Méthode de travail

Pour les méthodes de travail, nous avons appris de notre expérience en binôme en projet de Structures de Données Avancées, où nous avions commencé notre projet tard dans le semestre et en ayant mis en place trop peu de communication. Pour éviter de répéter ces erreurs, nous avons commencé le projet très tôt, avec des réunions régulières, à des intervalles de maximum 3 semaines. C'est notamment en commençant le projet tôt que nous nous sommes rendu compte qu'il y avait des erreurs dans les données de test qui nous avaient été fournies en début de semestre.

Nous avons d'abord mis en place la vérification de la grammaire du fichier donné en entrée. Ceci se fait grâce à la bibliothèque verification_grammaire, qui va analyser le fichier pour s'assurer que la grammaire utilisée correspond bien à celle que l'on suit. Nous avons fait de nombreux tests à cette étape pour nous assurer qu'il n'y ait pas de problème dès le début de notre code.

Ensuite, nous avons mis en place la structure de nœud. Cette partie nous a pris beaucoup de temps car il était important de trouver les bonnes structures pour représenter nos arbres en structures en C. Cette partie représente les manipulations des nœuds, sans qu'ils forment encore des arbres. Nous avons réalisé des tests avec les premières fonctions.

Pour continuer, nous avons mis en place les fonctions sur les arbres, comme le fait de calculer la hauteur d'un arbre, ou la construction d'un arbre. Une fois que ces fonctions là de réalisation de la structure de l'arbre ont été mises en place, nous avons conçu les fonctions qui vont récupérer les informations dans le fichier, comme le nom de l'espèce ou la distance.

Finalement, nous avons élaboré les fonctions d'affichage et de sauvegarde dans un fichier de l'arbre. Une fois ces fonctions testées, nous avons fait beaucoup de tests, avec des fichiers corrects comme incorrects en termes de grammaire.

Technologie

Structures de données principales

Notre première structure de données est t_nom_espece. Cette structure représente le nom d'une espèce sous forme de chaîne de caractères. Cette structure sera utilisée plus tard dans notre type représentant un nœud d'un arbre phylogénétique pour les nœuds de type feuille, nommés nœuds espèce.

Le second type est t_distance. Cela représente la distance entre deux espèces ou sous-arbres. Ce type est un double.

Le troisième type est la structure t_noeud, qui représente un nœud de l'arbre phylogénétique. On a deux types de nœuds : les « nœuds espèce » qui représentent les feuilles de l'arbre, et les « nœuds arbre » qui représentent un sous-arbre. Pour chaque arbre on a un attribut est_espece integer, s'il vaut 1 cela veut dire que le nœud est un nœud espèce, s'il vaut 0 c'est donc un nœud arbre. Les nœuds espèce contiennent un nom de type t_nom_espece. Les nœuds arbre possèdent deux enfants fils_gauche et fils_droit de type t_noeud et deux éléments t_distance distance_gauche et

distance_droite. On a une représentation récursive de l'arbre avec des pointeurs vers les sous-nœuds. Les nœuds espèce n'ont pas de fils, et les nœuds arbre n'ont pas de nom d'espèce.

Le dernier type est a_noeud, qui est un type pour un pointeur vers un nœud. Cela représente un arbre au sens large, donc par exemple la racine d'un sous-arbre ou de l'arbre complet.

Risques de ces structures et outils mis en place pour réduire ces risques

Parmi les risques liés à nos structures, on peut compter les fuites mémoire, les pointeurs qui pointent vers un espace qui a été libéré, les accès invalides et les erreurs de type.

Pour les fuites de mémoire, lors de l'allocation de nouveaux nœuds, si la libération n'est pas gérée la mémoire n'est jamais rendue. Pour réduire ce risque, nous avons créé une fonction spécifique pour libérer la mémoire.

Pour les pointeurs dans le vide, si on a libéré un nœud sans libérer le pointeur qui pointait vers lui on peut avoir des pointeurs qui renvoient vers un élément qui n'a pas de sens avec des accès illégaux. Pour empêcher cela, nous avons pensé à faire une fonction qui libère un nœud et met à NULL le pointeur qui pointait vers ce nœud.

Pour l'accès invalide, si on manipule un arbre sans vérifier l'intégrité de l'arbre, on peut avoir de nombreuses erreurs qui en découlent, comme des segmentation fault. Pour éviter cela, nous faisons des tests systématiques dans nos fonctions pour vérifier que les arbres donnés en entrée ne soient pas NULL. L'accès à la mémoire est donc encapsulé dans des fonctions sûres. Nous avons également pensé à créer une librairie qui vérifie que le fichier d'entrée contient un arbre correct suivant la grammaire avant de faire des opérations sur l'arbre du document, qui serait donc incorrect.

Avec nos types de structures vraiment spécifiques, on pourrait également faire des erreurs entre le traitement des nœuds espèce et des nœuds arbre. Pour éviter cela, nous avons créé des fonctions précises dédiées pour manipuler différents types de structures, avec des noms explicites, comme « initialiser_noeud ».

Nous avons également fréquemment utilisé l'encapsulation comment outil de protection, avec des fonctions utilitaires. Nous avons des fonctions dédiées pour gérer les flux de lecture, comme lire_caractere.

Affichage alternatif de l'arbre dans la console

Nous avons pensé à différentes approches pour réaliser l'affichage dans la console, mais beaucoup étaient difficiles à mettre en place, à cause de la récursivité. En effet, après de nombreux tests, nous avons regardé différentes manières de faire un affichage d'une structure avec des parents et des enfants.

Parmi ces exemples, nous avons découvert des manières de représenter des dossiers avec des fichiers ou des sous-dossiers. La représentation récursive et très lisible de dossiers d'un utilisateur StackOverflow nous a particulièrement intéressés, et nous avons décidé d'implanter cette représentation, avec des modifications. Cette représentation, que voici ci-dessous, représente des dossiers nommés avec des lettres, qui peuvent comprendre des sous-dossiers ou des fichiers.

```
A
+-B
| +-C
| +-D
|   +-E
+-F
  +-G
```

Nous avons adapté cette représentation pour notamment incorporer la notion de distances afin d'obtenir un résultat comme ci-dessous.

```
+--(0.00)
    +--(3.00)
    |   +--EspeceUn:1.00
    |   +--EspeceDeux:2.00
    +--EspeceTrois:0.50
```

Sauvegarde automatique dans un fichier standardisé

Lors de nos tests, nous étions embêtés par le fait de devoir changer manuellement le nom du fichier de sauvegarde de l'arbre créé avec notre algorithme, en nommant par exemple un fichier arbre1.txt, puis un fichier arbre2.txt, etc. les uns après les autres, à chaque test de notre code. Ces tests étaient parfois espacés de seulement quelques secondes.

Pour optimiser nos tests, nous avons donc mis en place un système supplémentaire qui suit une structure « arbre{nombre}.txt ». Ce code part de 1 en tant que nombre, et va incrémenter l'élément nombre jusqu'à trouver un fichier correspondant arbre{nombre}.txt qui n'existe pas encore. Une fois ce fichier trouvé, le code sauvegarde l'arbre voulu dans ce fichier. C'est un élément de debugging qui nous a été très utile, car cela nous permettait de pouvoir faire plusieurs tests à la suite, sans temps parasite, et nous pouvions avoir tout de même un suivi facile des fichiers que nous venions de créer par le fait qu'ils soient numérotés dans l'ordre, et non de manière aléatoire.

Difficultés rencontrées

Nous avons rencontré trois grandes difficultés : des données erronées, la réalisation de la structure de l'arbre, ainsi que l'affichage alternatif dans la console.

Données erronées

La première difficulté a été le fait que les premières données de test fournies ne suivaient pas la grammaire indiquée par le sujet. En effet, en testant notre algorithme de vérification de la grammaire d'un fichier donné en entrée qui était censé être correct, notre code indiquait toujours en sortie que le fichier était incorrect. Après avoir testé de nombreuses versions de notre avec toujours le même résultat, nous avons analysé à la main les fichiers de test, et nous avons alors découvert que les arbres ne suivaient effectivement pas la grammaire donnée. Une fois les données corrigées à la main, notre code acceptait bien les arbres modifiés. Ayant commencé le projet en début de semestre, nous avons pu remonter cette erreur très tôt, et proposer aux autres étudiants nos documents corrigés.

Structure de l'arbre

La seconde difficulté a été la réalisation de la structure de l'arbre binaire. En effet, nous avons mis en place au tout début une structure de nœud qui suivait deux cas :

- Une structure nœud espèce, qui représentait une espèce avec une chaine de caractères qui représentait son nom, ainsi qu'un attribut pour sa distance ;
- Une structure nœud arbre, qui représentait un nœud arbre avec deux fils de type nœud, ainsi que sa propre distance.

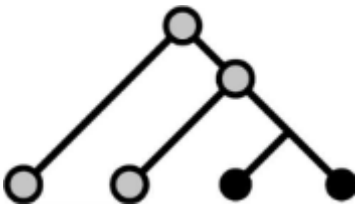
Les problèmes liés à cette structure étaient nombreux, les deux principaux étaient :

- Scinder en deux types les deux types de nœuds rendait la définition du type nœud arbre très complexe car ses enfants pouvaient être une combinaison de deux nœuds espèces, deux nœuds arbre, ou deux nœuds de type différents. Cela posait donc de très gros problème de définition du type nœud arbre car les deux types de nœuds étaient distincts.
- La gestion de la récursivité était extrêmement complexe à cause du fait que l'on ne gardait pas la distance des nœuds au niveau du nœud père, mais au niveau du nœud fils. Cela rendait l'exploration récursive très dur à mettre en place.

Une structure arbre_phylo à part a été mise en place au départ pour le traitement de la racine, mais elle n'accordait aucune valeur ajoutée au fait de manipuler directement des nœuds, et cela apportait des complications au niveau des paramètres d'entrée donnés aux fonctions.

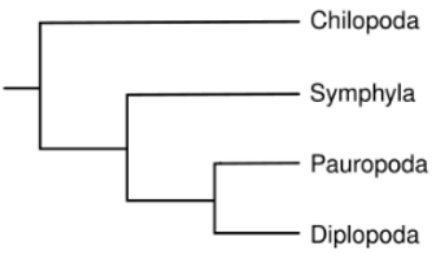
Affichage alternatif dans la console

Nous avons eu beaucoup de mal à trouver une approche pour faire l'affichage dans la console. En effet, à cause de l'aspect récursif à traiter dans l'affichage de l'arbre, nous avons eu beaucoup de mal à trouver un affichage qui n'engendre pas de bugs visuels à cause de la récursivité. La première approche à laquelle nous avions pensé était le fait de concevoir un arbre « vers le bas », avec des / et des \ pour représenter les différentes directions.



Nous avons écarté très rapidement cette représentation, car nous devions afficher sur chaque ligne, les décalages ainsi que les noms d'espèces qui étaient une feuille à ce niveau. Cela rendait les bugs d'affichage extrêmement majoritaires.

Notre seconde approche a été de représenter l'arbre de manière verticale avec une combinaison de | et de - pour imiter le résultat du schéma ci-dessous.



Nous aurions pu poursuivre cette approche, mais quand nous avons trouvé l'affichage que nous avons finalement choisi, nous nous sommes dit que le potentiel d'insertion de la distance de manière lisible était bien plus élevé, donc nous avons opté pour la représentation inspirée par l'affichage de dossiers, sous-dossiers et fichiers.

Expérience

Nous avons tous les deux beaucoup appris de ce projet. Nous avons vraiment pris en main le concept de grammaire, d'analyse de grammaire et renforcé nos compétences sur les arbres.

Nous avons également fait de grands progrès par rapport au projet « frère » que nous avions réalisé au cours du premier semestre. En effet, nous avons vraiment appris de notre méthodologie de travail du premier semestre, qui n'avait pas été efficace, et nous avons mis en place une méthodologie de travail avec plus de travail en amont et de communication avec des points réguliers. Nous sommes également très heureux d'avoir réussi à faire une implantation vraiment efficace de l'affichage de l'arbre phylogénétique, dont nous n'étions pas entièrement satisfais au premier semestre.

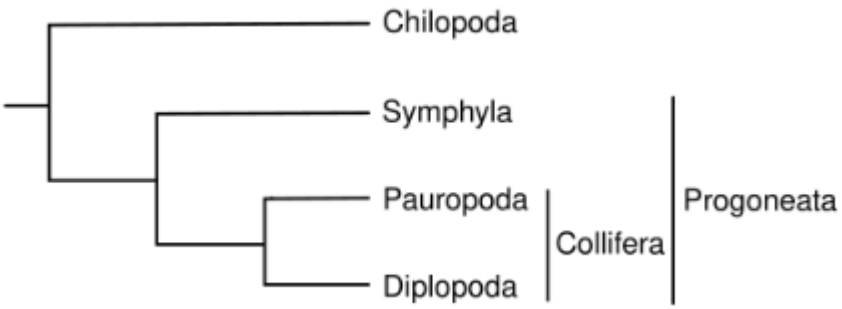
Nous trouvons notre projet complet, fonctionnel, et nous sommes heureux des efforts que nous avons fait pour les détails.

Prolongation

Il y a différents types de fichiers qui permettent de sauvegarder un arbre phylogénétique. Il pourrait être intéressant de sauvegarder notre arbre créé dans l'un de ces types de fichier de sauvegarde d'arbre phylogénétique. Parmi les extensions de fichier, nous avions repéré trois types que voici :

- En premier, il y a les fichiers de format Newick, qui produit des arbres qui représentent à partir d'un arbre de cette forme un arbre associé :

  (Chilopada,(Symphyla,(Pauropoda,Diplopoda))) ;



- En second, nous avions pensé au format Nexus. Pour vous donner un exemple, si l'on représente l'arbre donné dans l'exemple juste au-dessus, la représentation de cet arbre serait la suivante :

  ```
  #NEXUS
  Begin taxa;
    Dimensions ntax=4;
  ```

```
  Taxlabels
    Chilopada
    Symphyla
    Pauropoda
    Diplopoda;
End;
Begin trees;
  Tree arbre1 = (Chilopada,(Symphyla,(Pauropoda,Diplopoda)));
End;
```

- Le dernier format auquel nous avions pensé est une représentation avec des graphes avec un format DOT. Si l'on veut réaliser notre même arbre avec ce format, nous aurions fait une sauvegarde de ce type :

```
digraph ArbrePhylo {
  node [shape=ellipse];

  node0 [label=""];
  node1 [label="Chilopada"];
  node2 [label=""];
  node3 [label="Symphyla"];
  node4 [label=""];
  node5 [label="Pauropoda"];
  node6 [label="Diplopoda"];

  node0 -> node1;
  node0 -> node2;
  node2 -> node3;
  node2 -> node4;
  node4 -> node5;
  node4 -> node6;
}
```