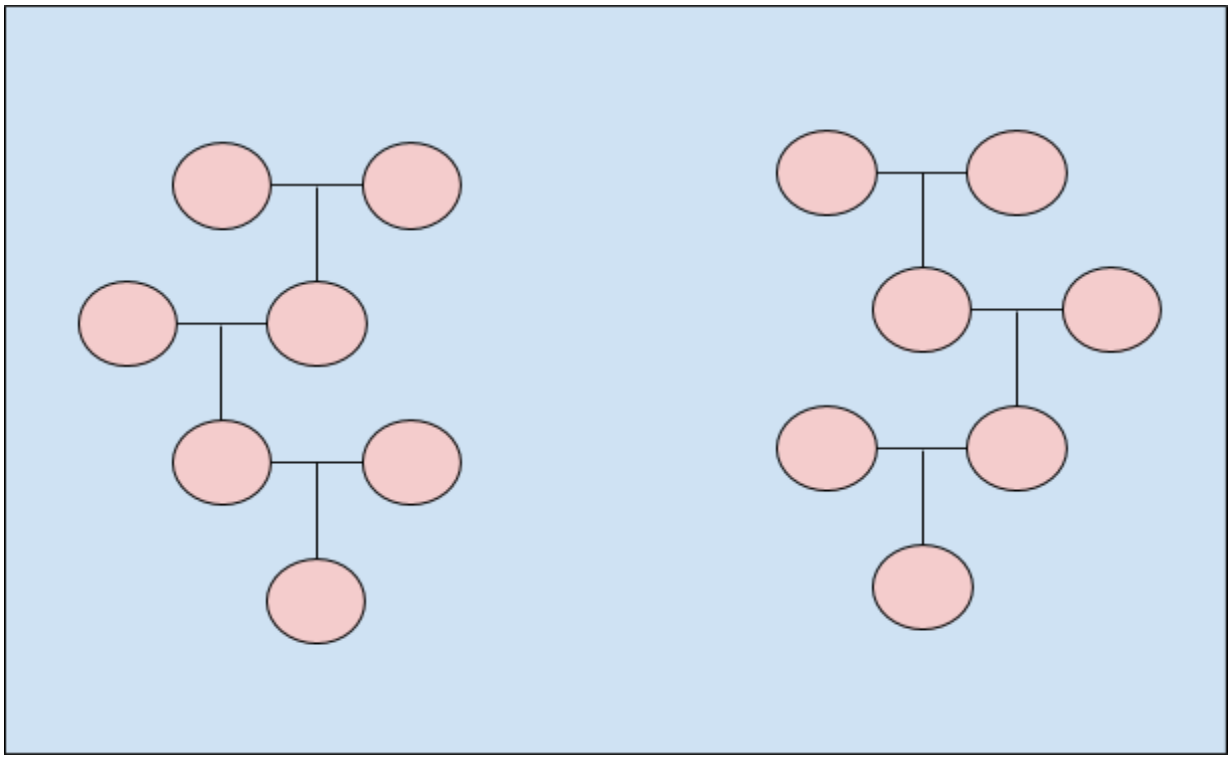
Programmation impérative

Rapport de projet

Arbre Généalogique



Introduction

Un arbre généalogique est une figure représentant un arbre dont les ramifications montrent la filiation des diverses branches d'une même famille. L'objectif de ce projet est d'implanter un programme permettant de construire un arbre généalogique.

Sommaire

Architecture de l'application en modules

Afin de réaliser notre objectif nous avons mis en place plusieurs modules :

- deux modules génériques : module Graphe et un module Registre.
- un module Arbre_Genealogique instanciant les modules graphes et registres, utilisé par le programme principal.

Les graphes

Les graphes sont un modèle abstrait de représentation de liens entre des objets. Ils sont constitués de sommets liés entre eux par des arêtes. Ainsi, notre module de graphe fournit des fonctionnalités d'ajout de sommet, d'ajout d'arête, d'accès aux arêtes existantes, etc. Dans ce projet, l'utilisation de graphes est très intéressante car elle nous permet de réaliser des arbres généalogiques très complets, avec différents types de relations, que nous ne pourrions pas implémenter avec un arbre binaire.

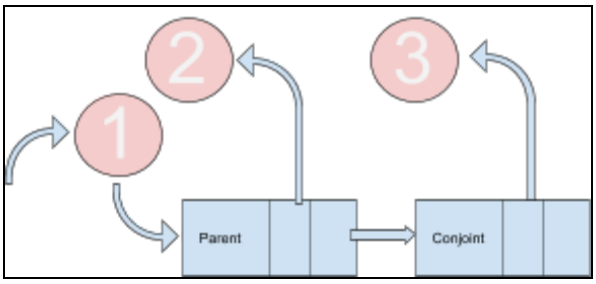


Schéma représentant une liste d'adjacence

Le registre

Un graphe n'est pas une structure optimale pour accéder efficacement aux données des personnes. Par conséquent, nous avons alors opté pour une table de hachage. Une table de hachage est un tableau dont chaque cellule est occupée par une pile d'éléments. Nous accédons à une donnée en rentrant sa clé, elle est alors transformée, à l'aide d'une fonction de hachage (ici le modulo), en un index qui est la numéro de la cellule qui contient la pile qui elle-même contient la donnée cherchée.

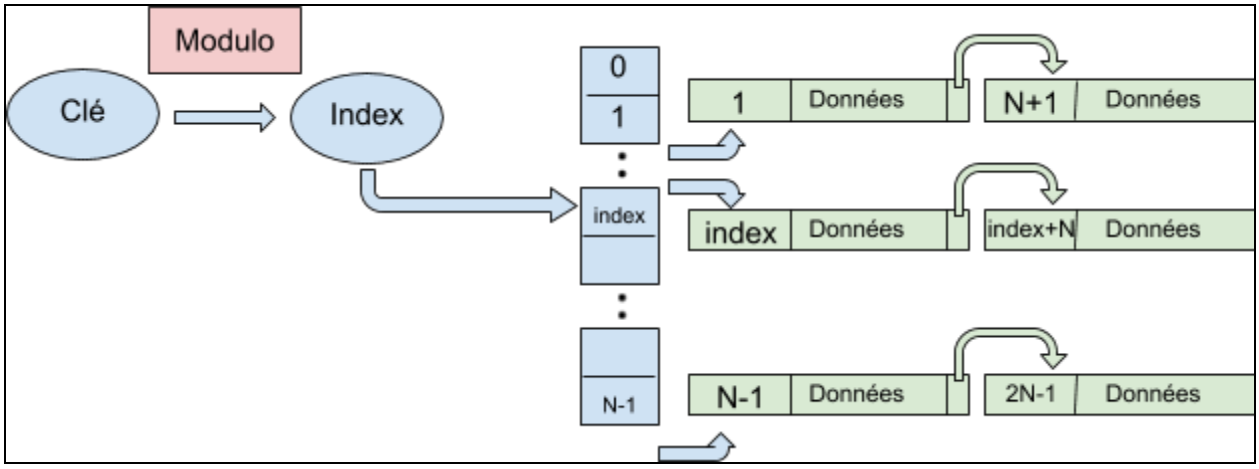


Schéma représentant notre table de hachage

Les tests

Pour tester notre programme nous avons tout d'abord tester nos deux modules génériques, en développant des scénarios de test puis des programmes testant toutes les fonctionnalités de nos modules. Enfin, pour le module principal, nous avons seulement réalisé un scénario pour vérifier que nos procédures "ponts" des modules déjà testés fonctionnent .

Les difficultés et solutions envisagées

La visibilité est assez compliquée à manier en Ada ; pour cette raison, le module Arbre_Genealogique contient des types qui sont des alias des types privés. La gestion des chaînes de caractère de longueur variable est elle aussi relativement complexe, et nous nous sommes régulièrement référés au site Wikibooks pour leur usage.

Répartition du travail

Vous pouvez consulter la liste des commits ici : github.com/GauBen/Arbre-Genealogique.

En résumé :

- Signatures et code du registre : Thibaud Sarlat
- Signatures et code des graphes : Gautier Ben Aïm
- Tests : Thibaud Sarlat et Gautier Ben Aïm
- Code qui lie les deux modules : Thibaud Sarlat
- Conception de l'interface : Gautier Ben Aïm
- Interface de gestion du registre : Thibaud Sarlat
- Interface de gestion du graphe : Gautier Ben Aïm
- Fonctionnalité de recherche dans le registre : Gautier Ben Aïm

Bilan du projet

Finalement, nous avons réussi à implanter un programme qui remplit sa fonction première c'est à dire construire des arbres. Nous pouvons actuellement ajouter des gens dans le registre puis construire des arbres en créant des relations entre eux. Nous avons aussi rajouté une fonctionnalité de statistiques fournissant diverses informations sur notre arbre. Nous obtenons alors un programme très complet. Néanmoins, nous pourrions ajouter quelques fonctionnalités subsidiaires comme celle de retourner le nombre d'ancêtres connus d'un individu.

Bilan personnel

Thibaud Sarlat : j'ai beaucoup apprécié ce projet qui m'a permis de consolider les connaissances acquises en cours et en travaux pratiques notamment sur la visibilité et la notion de privé. Aussi, ce projet m'a permis de découvrir la théorie des graphes qui est tout de même très pratique.

Au niveau du temps de travail sur chaque partie :

- 7h signature et code du registre
- 5h Tests

- 4h code liant les modules
- 7h interface de gestion du graphe.