PR3-S2

Ayoub Bouchama

15 Juin 2023

1 Projet PIM: Session 2

1.1 Implémentation de l'arbre préfixe

Dans mon implémentation, chaque nœud de l'arbre contient :

1. Une valeur d'arête (arête au-dessus).
2. Une valeur de nœud.
3. Une clé.
4. Un booléen qui indique si le contient contient une information.
5. Les fils du nœud qui sont des arbres (stockés dans une file).

J'ai choisi d'utiliser une file pour stocker la clé, car c'était la meilleure solution pour cette implémentation. Cela permet de diviser la clé en de petits morceaux de clés. Chaque clé est constituée de la concaténation des valeurs d'arêtes de chaque nœud qu'on a parcouru à partir de la racine jusqu'au nœud correspondant à la clé.

Les fils sont stockés dans une file car, initialement, j'avais prévu de les stocker dans un tableau. Cependant, étant donné que l'insertion n'impose pas nécessairement un ordre et que le module des files a déjà été implémenté dans le projet pr2-s2, j'ai utilisé les fonctions et procédures existantes dans ce module.

1.2 Utilisation du compteur d'occurrence

Pour utiliser le programme, il faut d'abord compiler les fichiers `files.adb` et `arbre.adb` (ou compiler directement le fichier `test_arbre.adb`).

Le fichier `test_arbre.adb` contient des tests du module Arbre, principalement des tests pour la fonction `Ajouter` qui insère un nœud dans l'arbre, la fonction `Valeur` qui retourne la valeur du nœud associée à une clé, et des affichages de l'arbre à chaque étape d'insertion pour observer les différences. Le test contient chaque affichage où l'on affiche les nœuds (arêtes, clés, valeur du nœud)

de chaque niveau.

Ensuite, il faut compiler les fichiers `CLI.adb` et `main.adb` (ou utiliser `gnatmake main.adb` pour compiler tous les fichiers).

Il faut ensuite disposer du fichier dans le même répertoire et lancer la commande :

```
./main <mot> <fichier>
```

où `mot` est le mot à rechercher et `fichier` est le nom du fichier de recherche.

Si le mot existe, le nombre d'occurrences du mot s'affichera, sinon le programme signalera que ce mot n'existe pas.

Le fichier `ayoub.txt` existe pour le test ; Vous pouvez par exemple essayer la commande :

- `./main ayoub ayoub.txt` pour voir que le programme affiche que le mot 'ayoub' existe 6 fois dans le fichier.
- `./main n7 ayoub.txt` pour voir que le programme affiche que le mot 'n7' n'existe pas dans le fichier 'ayoub.txt'.

1.3 Réponse à la question 4 (Suggestion des mots)

Ma proposition d'application efficace de l'arbre préfixe est la suggestion de mots similaires ou la complétion de mots, telle qu'on les trouve dans les suggestions des claviers numériques des téléphones.

En utilisant un arbre préfixe, il est possible de stocker un large ensemble de mots et de préfixes correspondants. Lorsque l'utilisateur commence à saisir un mot, l'arbre préfixe est consulté pour trouver tous les mots qui ont le même préfixe. Ces mots peuvent être présentés à l'utilisateur comme des suggestions, l'aidant ainsi à compléter le mot ou à choisir une option similaire.

Ceci serait plus visible et évident en détenant une interface graphique pour rafraîchir les suggestions à chaque fois que l'utilisateur tape un mot. Dans mon programme, je me suis contenté d'insérer un petit dictionnaire qui contient une dizaine de mots (je n'ai pas pu en mettre un plus grand qui contient presque tous les mots du français à cause des erreurs de mémoire STACK OVERFLOW). On demande à l'utilisateur s'il souhaite une suggestion de mot ou de quitter. S'il choisit la première option, on lui demande d'entrer les premières lettres de son mot, puis on affiche tous les mots de l'arbre dont les préfixes correspondent à ce qu'il a entré. Pour tester ceci, vous pouvez lancer la commande (après avoir compilé le fichier `suggestion.adb`) :

```
./suggestion
```

1.4 Réponse à la question 5

Je pense que la structure arborescente est très efficace dans les algorithmes de recherche grâce à sa capacité à organiser les données de manière hiérarchique. Elle permet une recherche rapide car elle réduit le nombre de comparaisons nécessaires entre nœuds à chaque étape.
Cependant, je pense que la suppression d'une donnée dans l'arbre peut être complexe, car lorsque nous devons supprimer un nœud, nous devons également supprimer tous ses descendants (ses fils, leurs fils, etc.). Cependant, si nous souhaitons conserver les descendants du nœud que nous supprimons, cela peut poser un problème. Il peut être difficile de gérer cette situation et de maintenir la structure de l'arbre cohérente tout en préservant les fils du nœud supprimé. Cela nécessite une manipulation délicate de l'arbre et peut potentiellement affecter les performances de l'opération de suppression.

J'ai pensé qu'une solution possible pour remédier à ce problème serait de désactiver le nœud au lieu de le supprimer, afin de préserver le chemin d'accès vers ses fils.

Une amélioration possible pour réduire le temps de recherche dans l'arbre préfixe est d'indexer les éléments de l'arbre. Plutôt que de parcourir tous les nœuds pour effectuer une recherche, on peut utiliser des indices associés à chaque nœud pour accéder directement aux fils correspondants. Cela permet d'obtenir rapidement la valeur associée au nœud sans avoir à parcourir l'ensemble de l'arbre. Par exemple, dans notre application de l'arbre préfixe, au lieu d'insérer les fils de chaque nœud de manière aléatoire, on peut utiliser le code de chaque morceau de clé (arête) pour accéder directement aux fils appropriés, ce qui facilite la récupération des valeurs des nœuds.

Bilan

Le projet etait vraiment interessant meme si je n'avais pas compris le concept de la mise en oeuvre de ce type d'arbre en premier temps. J'avais renconté plusieurs difficultés, je m'etais bloqué plusieurs fois pour des erreurs betes mais j'ai pu débogué mon code à l'aide des affichages sur la console pour détecter d'où vient une erreur.