

Jouer au petit génie logiciel avec chatGPT

Adam SAID 21905365

Pour chacune des activités vous pouvez retrouver le code complet sur [le git du projet](#).

Activité 1 (8h15 - 10h00)

Pour cette première activité j'ai décidé de d'abord expliquer chaque partie du code et dans la dernière partie montrer l'utilisation de GPT pour cette activité.

1. Objectif

L'objectif principal de ce programme est de développer une application Java répondant à deux fonctions distinctes :

- a. Vérifier si un terme est un mot composé appartenant à un ensemble F et obtenir son identifiant.
- b. Vérifier si un terme entré par l'utilisateur est un préfixe d'un ou plusieurs mots composés et obtenir la liste des paires (mot composé, id) correspondantes.

Le programme doit être optimisé en termes de temps de réponse et d'utilisation de la mémoire, tout en étant capable de traiter un fichier de termes composés de grande taille.

2. Structure de Données Utilisée

La structure de données principale utilisée pour stocker les termes composés est un **Trie**. Un Trie est une structure arborescente où chaque nœud représente une lettre d'un terme. Cette structure permet une recherche efficace des termes, en particulier pour les préfixes.

TermTrie :

- Le Trie est construit avec une classe appelée **TermTrie**.
- Chaque nœud du Trie a une liste de sous-nœuds représentant les lettres suivantes dans les termes composés.
- Chaque feuille du Trie stocke l'identifiant associé au terme complet.

Cette structure offre une recherche rapide, en particulier pour les préfixes, tout en minimisant l'utilisation de la mémoire.

3. Méthode de Chargement du Fichier

Le fichier de termes composés est chargé à l'aide d'un **BufferedReader** pour optimiser la lecture ligne par ligne. Les lignes vides et les commentaires (commençant par "//") sont ignorés. Chaque ligne est ensuite analysée et insérée dans le Trie.

Pour optimiser l'utilisation de la mémoire, le Trie est périodiquement redémarré après le traitement d'un nombre défini de lignes (**CHUNK_SIZE**). Cela permet de libérer la mémoire utilisée par le Trie précédent et de ne pas atteindre la limite de la RAM.

4. Mesures de Performance

Le programme mesure le temps total pris pour charger le fichier, ainsi que le temps pris pour chaque recherche d'identifiant ou de préfixe. Les résultats sont affichés en millisecondes pour une meilleure lisibilité.

De plus, l'utilisation de la mémoire est mesurée et affichée en mégaoctets (Mb) :

Temps total du parsing : 4 secondes

Temps de recherche pour 'avant toute chose' : 0 millisecondes

Résultat : ID = 9

Temps de recherche par préfixe pour 'la' : 7 millisecondes

Résultat : [(la barbe, 316), (la conception des appareils, 1246), (la Famille Cokalane, 524), (la Femme libre, 4448), (la Grande Pâque russe, 160), (la Jungle, 3301), (la Libre Parole, 3268), (la Méridienne, 803), (la même chose, 3184), (la Planète sauvage, 775), (la Revue parisienne, 3148), (la ramener, 3911), (la Saison des amours, 2065), (la Vieille Fille, 4292), (la Ville des prodiges, 3234), (lapin de garenne, 1508), (labour anguleux, 1295), (langages de programmation, 213), (langue allemande, 3944), (langue néolatine, 4091)]

Mémoire totale de la JVM : 615.0 Mb

Mémoire libre : 546.77777 Mb

Mémoire utilisée : 68.22222 Mb

5. Utilisation de GPT

J'ai utilisé GPT afin de me donner des idées de structure de données en Java pour implémenter un système de stockage et de recherche de terme efficace. Le premier prompt a été le suivant :

"Je souhaite réaliser un programme en Java permettant de rechercher et récupérer

l'ID d'un terme stocké en indiquant le terme entier ou un préfixe. Je veux que cette recherche soit efficace, rapide et avec un coût réduit en RAM. Donne moi des exemples de structures efficaces en Java pour réaliser cela"

J'ai choisi d'écrire mes prompts en français car suivant des études que j'ai vu par le passé, écrire à GPT en français ou en anglais n'a pas d'influence car ce sont deux langues qui ont été très utilisées durant le développement du modèle de GPT. Ce premier prompt m'a bien aidé car j'ai pu directement avoir une structure de fichier efficace sans avoir à réfléchir si je partais sur un arbre, des listes ou bien des types déjà fait par des bibliothèques Java.

La réponse de GPT a été la suivante :

"Pour résoudre ce problème, nous pouvons utiliser une structure de données appelée trie (arbre préfixe) pour stocker les termes composés de manière efficace. Cela permettra de réduire l'empreinte en RAM et d'accélérer les recherches de termes.

```
class TrieNode {
    Map<Character, TrieNode> children;
    int id; // L'identifiant associé au terme composé

    public TrieNode() {
        this.children = new HashMap<>();
        this.id = -1; // Initialise l'identifiant à -1 par défaut
    }
}

public class TermTrie {
    private TrieNode root;

    public TermTrie() {
        this.root = new TrieNode();
    }

    // Insère un terme composé dans le trie
    public void insert(String term, int id) {
        TrieNode node = root;
        for (char c : term.toCharArray()) {
            node.children.putIfAbsent(c, new TrieNode());
            node = node.children.get(c);
        }
        node.id = id;
    }

    // Vérifie si un terme est un mot composé et retourne son identifiant
    public int isCompoundWord(String term) {
```

```

        TrieNode node = root;
        for (char c : term.toCharArray()) {
            if (!node.children.containsKey(c)) {
                return -1; // Le terme n'est pas dans le
trie
            }
            node = node.children.get(c);
        }
        return node.id;
    }

    // Recherche des termes composés ayant le préfixe donné
    public List<Pair<String, Integer>> searchByPrefix(String
prefix) {
        List<Pair<String, Integer>> result = new
ArrayList<>();
        TrieNode node = root;
        for (char c : prefix.toCharArray()) {
            if (!node.children.containsKey(c)) {
                return result; // Aucun terme avec ce
préfixe
            }
            node = node.children.get(c);
        }
        findAllWordsWithPrefix(node, prefix, result);
        return result;
    }

    // Fonction récursive pour trouver tous les termes avec
un préfixe donné
    private void findAllWordsWithPrefix(TrieNode node, String
currentPrefix, List<Pair<String, Integer>> result) {
        if (node.id != -1) {
            result.add(new Pair<>(currentPrefix, node.id));
        }
        for (char c : node.children.keySet()) {
            findAllWordsWithPrefix(node.children.get(c),
currentPrefix + c, result);
        }
    }
}

```

”

Ensuite, j'ai eu des problèmes pour lire le fichier qui était très grand et GPT m'a aidé à changer mon code pour séparer le fichier en plusieurs parties (**chunk**) et utiliser un **BufferReader**. Je n'avais pas eu l'idée du **bufferReader** et mon code n'affichait rien

et je ne comprenais pas pourquoi. Grâce à GPT j'ai eu des indications de pourquoi cela ne marchait pas en utilisant la librairie **Scanner** et une possibilité pour contourner mon problème. Mon prompt a été le suivant :

“J'ai des problèmes pour lire le fichier car il semble être trop grand et j'ai une erreur “Java Heap size”. Comment pourrais-je faire pour diviser le fichier en parties pour simplifier sa lecture et son parsing ?”

Réponse : “L'erreur “Java heap space” indique que votre programme a atteint la limite de mémoire allouée par la machine virtuelle Java (JVM). Cela se produit généralement lorsque vous essayez de traiter un gros fichier qui nécessite plus de mémoire que celle allouée par défaut à la JVM.

Pour résoudre ce problème, vous pouvez augmenter la taille de la mémoire allouée à la JVM en utilisant les options -Xms et -Xmx lors de l'exécution de votre programme.

Par exemple, vous pouvez exécuter votre programme avec une mémoire minimale de 512 Mo et une mémoire maximale de 2 Go comme suit :

[Code inutile]

Cependant, il est important de noter que l'augmentation de la mémoire n'est qu'une solution temporaire et ne résoudra pas nécessairement le problème de manière définitive. Si le fichier est extrêmement volumineux, il peut être plus efficace de traiter le fichier par parties, plutôt que de charger l'ensemble du fichier en mémoire.

Pour traiter de manière plus efficace les fichiers volumineux, nous devrions diviser le traitement du fichier en plusieurs parties et libérer la mémoire entre chaque partie.

Voici une approche qui pourrait mieux gérer la mémoire pour les fichiers volumineux. Elle divise le fichier en parties (lignes) et les traite séparément :

[Code avec CHUNK_SIZE et un BufferedReader]”

Mais dans son dernier message GPT n'a pas utilisé **CHUNK_SIZE** à part pour définir la variable, je lui ai donc fait remarquer et il a changé sa réponse pour ajouter l'utilisation de la variable lors du parsing du fichier.

Pour terminer je lui ai demandé d'ajouter l'affichage du temps de parsing et de recherche ainsi que le calcul de la RAM utilisée dans mon main en lui fournissant mon code :

*“Voici le code actuel :
[Classe Main]*

Rajoute y le temps pris pour rechercher chaque ID ou terme et aussi la RAM utilisée”

Réponse :

“Pour mesurer le temps pris pour rechercher chaque ID ou terme et la RAM utilisée, vous pouvez utiliser `System.currentTimeMillis()` pour mesurer le temps et `Runtime.getRuntime().totalMemory()` et `Runtime.getRuntime().freeMemory()` pour obtenir des informations sur la mémoire. Voici comment vous pouvez les intégrer dans votre programme :

[Code Main modifié convenablement]

Cette version affiche le temps total pris pour le chargement du fichier, le temps pris pour chaque recherche d'ID ou terme, ainsi que des informations sur l'utilisation de la mémoire. Assurez-vous de personnaliser et adapter ces mesures en fonction de vos besoins.”

Et cette modification rendait exactement ce que je voulais alors je n'ai pas fais de retour supplémentaire. Cela m'a donc permis un gain de temps plutôt intéressant car sur une chose aussi simple les erreurs sont peu fréquentes et cela m'évite d'avoir à chercher quelle bibliothèque utiliser et à faire les différents “print” dans la console.

Activité 2 (22h30 - 0h30)

Vous pouvez retrouver le code généré pour chacune des spécifications dans le dossier approprié du Git. Je n'ai pas voulu le mettre ici pour ne pas alourdir le rapport.

Spécification 1

Pour ajouter les spécificités demandées j'ai voulu donner tout l'énoncé à GPT afin de voir ce dont il était capable. Je lui ai donc copié et collé l'énoncé en lui demandant de réaliser le code répondant aux spécifications données. Sa réponse n'a pas été fameuse, bien qu'il ait fait les classes Cible, Arme, EtoileDeLaMort... Cela manquait de justesse, par exemple on préférera faire de Cible une interface car elle ne peut pas être implémentée d'elle-même. De plus, la classe VaisseauSpatial n'utilisait pas la classe Arme comme demandé.

J'ai alors décidé de séparer chaque point en un prompt afin de réaliser les tâches petit à petit. En faisant cela, GPT a été beaucoup plus précis sur sa production et a répondu selon mes attentes au premier, troisième et quatrième points du premier coup. Mais pour le second il m'a fallu préciser plusieurs fois que Vaisseau devait étendre la classe Arme et cela devait être contre-intuitif pour lui à la vue de l'insistance que j'ai dû mettre.

Toujours est-il que GPT m'a quand même bien aidé pour implémenter assez rapidement ces différentes classes et même refactoriser celles déjà faites afin d'y inclure les modifications.

Spécification 2

Dans cette suite de l'activité, j'ai continué à utiliser GPT pour implémenter les modifications demandées. Pour commencer je lui ai donné l'énoncé ainsi que le premier point et GPT m'a créé le code correspondant qui satisfaisait les spécifications. Pour cela il a créé trois classes pour les trois types d'objets (Arme, Documents et AutreObjet) ainsi qu'une classe Humanoïde pour étendre la classe Personnage sans la modifier. Il y a ajouté la possibilité de détenir un objet ainsi que les méthodes pour acquérir, se séparer et donner un objet.

Pour le second point, de la même manière que précédemment j'ai simplement copié collé la spécification et GPT a fait les modifications de lui-même et elles sont convenables du premier coup. Il a de lui-même modifié la classe Arme précédemment créée pour y ajouter un attribut de puissance et il a ajouté à la classe Humanoïde une fonction pour tirer sur un Personnage tout en vérifiant qu'il possède bien une arme et en modifiant les points de vie de l'adversaire.

Pour le troisième point, j'ai donné directement la spécification est l'IA à modifier la classe Document pour y ajouter les nouvelles spécificités en ajoutant la capacité de lecture qui augmente les connaissances du détenteur et une gestion de l'état de lecture du document. Cependant il a halluciné que Humanoïde possédait les getter et setter d'objet pour vérifier que le personnage est bien le détenteur du document. Il a donc fallu que je les rajoute manuellement.

Pour ce dernier point j'ai comme à l'accoutumée donné à GPT le point à réaliser mais également l'interface Monstre à utiliser. Et il n'a pas dérogé à la règle et a parfaitement réalisé la classe Troll qui étend Personnage et implémente Monstre en ajoutant un attribut de puissance et une méthode d'attaque, ce qui convient très bien à ce qui est demandé.

Spécification 3

Pour commencer le refactoring j'ai voulu voir si GPT pouvait tout faire tout seul en lui donnant uniquement la consigne : "On veut jouer dans un univers mixte Star Wars - LOTR. Factoriser les 2 spécifications ci-dessus dans un même code." Mais comme je le craignais, GPT a fait un peu n'importe quoi, par exemple il a refactorisé la classe Arme en créant ArmeMixte qui étrangement étend la classe personnage pour donner à une arme la capacité de se déplacer. Et il a fait de même pour la classe DocumentMixte. Je donc plutôt décider de faire des prompts pour chaque classe en commençant par la classe Personnage. Mon prompt a été le suivant : "Je veux que tu refactor la classe personnage de Star Wars et LOTR pour qu'elles puissent exister dans un même programme Java [Code des deux classes]".

Sa réponse : "vous pouvez créer une classe `Personnage` générale avec les attributs communs, puis étendre cette classe pour chaque univers spécifique. Voici comment vous pouvez refactoriser les classes" Suivi d'une classe Personnage qui étend Cible ainsi que deux classes qui étendent la première en y ajoutant respectivement les attributs de précision et de connaissances. Cela répond parfaitement au refactoring voulu.

En suite j'ai demandé le refactoring des classes Armes, Objet et Document afin qu'elles puissent fonctionner dans les deux milieux. GPT m'a répondu avec les trois classes modifiées sans problème.

La suite du refactoring a été faite de la même manière sans poser aucun problème à GPT pour effectuer les changements. Au final on se retrouve avec un package contenant toutes les classes des deux milieux et des `extend` pour la classe `Personnage`. Les autres étant spécifiques au deux univers, il n'y avait pas grand chose à changer.

Activité 3 (15h00 - 16h00)

Je me suis servi de GPT afin de corriger le code en C qui contenait des erreurs. Pour ce faire je lui ai donné tout le code fourni avec l'énoncé expliquant les AVL et j'ai terminé mon prompt en demandant "ce code marche-t-il ?".

Ce à quoi il m'a répondu avec la liste des erreurs à corriger et comment les corriger :

"Erreurs de syntaxe dans la fonction `insérer` :

La condition `racine->gauche` dans la fonction `insérer` devrait être `racine->gauche`.

[Code corrigé]

Erreur dans la condition de comparaison `==` dans la fonction `supprimer` :

La condition `if (racine->gauche = NULL || racine->droite == NULL)` devrait être `if (racine->gauche == NULL || racine->droite == NULL)`.

Erreur dans la fonction `equilibrerArbre` :

Dans la condition `if (equilibre > 1 && valeur > racine->gauche->valeur)`, vous utilisez `faireSimpleRotationAGauche` au lieu de `faireDoubleRotationAGauche`."

Bien que les deux premières erreurs soient bien corrigées, GPT ne m'a pas donné de code corrigé pour la troisième et puis j'avais un doute car il corrigeait l'un des deux cas de rotations double mais pas les deux. Je lui ai donc demandé s'il était sûr de lui en lui donnant la fonction `equilibrerArbre()`. Ce à quoi il m'a répondu :

Je m'excuse pour la confusion dans ma réponse précédente. Vous avez raison, il y a une erreur dans la partie de code que vous avez citée. La rotation double à droite devrait utiliser `faireDoubleRotationAGauche` au lieu de deux rotations simples. Voici la correction : [Code corrigé]"

Cette fois-ci le code était bien corrigé. Cette première passe m'a donc permis de corriger une partie du code sans même essayer de le lancer. Mais en exécutant le programme une première fois je m'aperçois qu'il reste une erreur de typo lors de l'appel de `libererMémoire()`. Je pourrais très bien faire la correction moi même mais je décide de donner uniquement la fonction à GPT afin de voir s'il en voit l'erreur :

"Oui, vous avez raison, et je m'excuse pour la confusion. Il y a une erreur dans le nom de la fonction. La fonction s'appelle `libererMemoire`, mais dans son corps, elle appelle `libererMemoire` au lieu de `libereMemoire`. Voici la correction : [Code corrigé]"

Cette fois-ci le code était bien corrigé et fonctionnel. Pour poursuivre et utiliser ce code j'ai demandé à GPT de me créer un main() exemple pour utiliser ce programme et voila sa réponse :

“Bien sûr, voici un exemple simple de `main` pour utiliser les fonctions de l'arbre AVL que vous avez définies :

```
int main() {
    Noeud* racine = NULL;

    // Insérer des valeurs dans L'arbre AVL
    racine = inserer(racine, 10);
    racine = inserer(racine, 20);
    racine = inserer(racine, 30);
    racine = inserer(racine, 15);
    racine = inserer(racine, 5);

    // Afficher L'arbre AVL
    printf("Arbre AVL après insertion :\n");
    afficherArbre(racine, 0);

    // Supprimer une valeur de L'arbre AVL
    racine = supprimer(racine, 20);

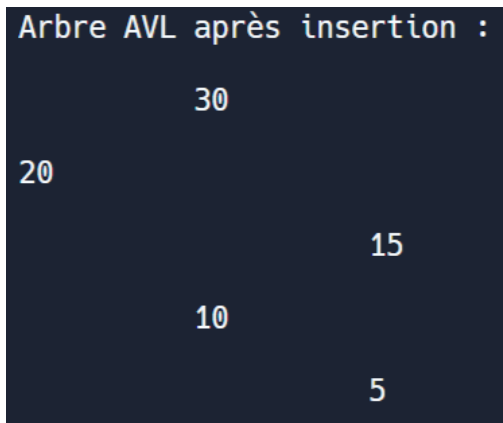
    // Afficher L'arbre AVL après suppression
    printf("\nArbre AVL après suppression de la valeur 20 :\n");
    afficherArbre(racine, 0);

    // Libérer la mémoire utilisée par L'arbre AVL
    libererMemoire(racine);

    return 0;
}
```

Cet exemple crée un arbre AVL, y insère quelques valeurs, affiche l'arbre après l'insertion, supprime une valeur, puis affiche à nouveau l'arbre après la suppression. Enfin, la mémoire utilisée par l'arbre est libérée à la fin du programme. Vous pouvez ajuster les valeurs et les opérations en fonction de vos besoins.”

Ce qui m’a après exécution rendu l’arbre suivant :



Pour terminer je lui ai demandé de me faire exactement la même chose mais avec un programme en Java et voila sa réponse :

“Voici une version équivalente du code en Java. Notez que Java utilise une gestion automatique de la mémoire (garbage collector), donc il n'est pas nécessaire d'appeler explicitement la fonction `free` comme dans C. [Code en Java]”

Après exécution le programme semble fonctionnel à première vue car il renvoi le même arbre qu'en C mais tout de même on a une erreur “NullPointerException”

```
Arbre AVL après insertion :

    30
   /  \
  20    15
 /      \
10        5

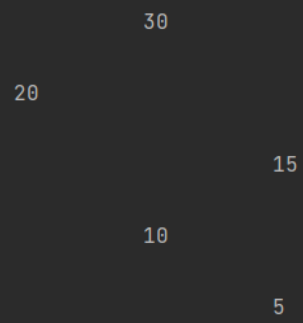
Exception in thread "main" java.lang.NullPointerException Create breakpoint
    at AVLTree.faireSimpleRotationADroite(AVLTree.java:31)
    at AVLTree.faireDoubleRotationAGauche(AVLTree.java:68)
    at AVLTree.equilibrerArbre(AVLTree.java:86)
    at AVLTree.supprimer(AVLTree.java:164)
    at AVLTree.main(AVLTree.java:209)

    5

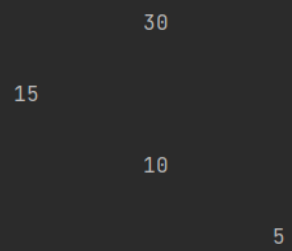
Process finished with exit code 1
```

Pour corriger cela j'ai simplement donné le nom de l'erreur ainsi que le code à la ligne où survient l'erreur et GPT m'a répondu avec un correctif en ajoutant des conditions avant d'essayer de récupérer les fils des nœuds. Et cela a corrigé le problème :

Arbre AVL après insertion :



Arbre AVL après suppression de la valeur 20 :



Process finished with exit code 0