TER BRENDA

Extraction de Réactions d'une BD

Shutong ZHENG
shutong.zheng@universite-paris-saclay.fr
29 avril 2023

I. INTRODUCTION

1. PRÉSENTATION DU PROJET

Le projet consiste en la création d'un extracteur de réactions enzymatiques à partir d'une base de données BRENDA. L'extracteur doit être capable de trouver tous les chemins partant d'un réactif donné et aboutissant à un produit donné, en passant par une distance spécifiée.

2. OBJECTIFS DU PROJET

Le projet vise à permettre aux étudiants de se familiariser avec les réactions biochimiques, les méthodes d'analyses lexicale et syntaxique, l'algorithmique des graphes, la programmation parallèle, et la conception modulaire de logiciels. Il s'agit également de mettre en œuvre une méthode efficace pour extraire les réactions enzymatiques à partir de la base de données BRENDA en utilisant une approche parallèle pour accélérer le temps de calcul. Les étudiants doivent également concevoir une interface utilisateur conviviale et un manuel d'utilisation pour le logiciel extracteur.

II. ANALYSEUR LEXICAL

1. PRINCIPE DE L'ANALYSEUR LEXICAL

L'analyseur lexical est la première étape de l'analyse des réactions décrites dans un fichier texte. Son rôle est de séparer le texte en différents éléments tels que les noms d'enzymes, les substrats, les produits, les constantes de vitesse et les inhibiteurs. Le principe de l'analyseur lexical est de reconnaître les motifs qui correspondent à chacun de ces éléments et de les stocker dans une structure de données appropriée.

2. IMPLÉMENTATION DE L'ANALYSEUR LEXICAL

L'analyseur lexical est implémenté en utilisant des expressions régulières pour reconnaître les différents motifs du langage de description des réactions. Chaque motif est associé à un identifiant qui permet de le stocker dans une structure de données appropriée. Les données sont stockées dans des objets qui représentent les réactions, les enzymes, les substrats, les produits, les constantes de vitesse et les inhibiteurs.

III. ANALYSEUR SYNTAXIQUE

1. PRINCIPE DE L'ANALYSEUR SYNTAXIQUE

Le principe de base de l'analyseur est d'extraire des informations sur les réactions chimiques et les inhibitions à partir d'un fichier d'entrée donné. L'analyseur lit d'abord chaque ligne du fichier et crée des objets Réaction et Inhibition à partir de son contenu. Il ajoute ensuite ces objets aux structures de données correspondantes pour un traitement et des calculs ultérieurs.

2. SYNTAXE DU LANGAGE DE DESCRIPTION

Les réactions et les inhibitions dans le fichier d'entrée sont décrites à l'aide d'une syntaxe spécifique. Pour les réactions chimiques, la syntaxe est la suivante :

Enzyme : Substrats -> Produits | Unités ; Kcat

Enzyme : nom de l'enzyme, une chaîne entourée de guillemets doubles.

Substrats : une liste de réactifs, séparés par des signes plus (+), chacun représenté par une chaîne entre guillemets doubles.

Produits : Une liste de produits, séparés par des signes plus (+), chaque produit est représenté par une chaîne entre guillemets doubles.

Unités : une liste d'unités, séparées par des virgules (,), chaque unité étant constituée d'une valeur de concentration (nombre à virgule flottante) et d'un nom (chaîne entre guillemets doubles).

Kcat : Constante de vitesse de réaction de l'enzyme, exprimée sous forme de nombre à virgule flottante.

Pour la suppression, la syntaxe est la suivante :

Inhibiteur : Substrat | Unité

Inhibiteur : nom de l'inhibiteur, une chaîne entre guillemets doubles.

Substrat : Substrat à supprimer, représenté par une chaîne entre guillemets doubles.

Unité : Unité composée d'une valeur de concentration (nombre à virgule flottante) et d'un nom (chaîne entre guillemets doubles).

3. IMPLÉMENTATION DE LA SYNTAXE DE L'ANALYSEUR

L'implémentation de l'analyseur est basée sur le code fourni, qui comprend principalement les parties suivantes :

Définissez les classes Espace, Réaction et Inhibition pour représenter les informations dans le fichier d'entrée.

Créez la classe analyzeLexicale, contenant des propriétés et des méthodes privées (extract et buildGraph) et des propriétés publiques (espaces, inhibitions et réactions).

Implémentez le constructeur et le destructeur de la classe analyzeLexicale.

Implémentez la méthode d'extraction de la classe analyzeLexicale pour analyser le fichier d'entrée et créer des objets de réaction et d'inhibition.

Définissez les fonctions auxiliaires split et extractContent pour gérer les opérations de chaîne.

IV. ALGORITHME D'EXTRACTION

1. PRINCIPE DE L'ALGORITHME D'EXTRACTION

L'algorithme d'extraction est utilisé pour rechercher et identifier tous les chemins possibles entre deux nœuds d'un graphe, en respectant une distance spécifique. Dans le code fourni, plusieurs méthodes de recherche ont été implémentés pour effectuer cette tâche, notamment DFS (Depth-First Search), BFS (Breadth-First Search) et IDDFS (Iterative Deepening Depth-First Search).

2. ARCHITECTURE DE L'ALGORITHME D'EXTRACTION

L'architecture de l'algorithme d'extraction consiste en une classe Graph qui contient les fonctions de recherche, ainsi que des méthodes pour ajouter des arêtes entre les nœuds et pour rechercher les chemins entre les nœuds. Les principales fonctions de recherche sont les suivantes:

1. dfs(): Recherche en profondeur (DFS) pour trouver tous les chemins possibles entre deux nœuds.
2. bfs(): Recherche en largeur (BFS) pour trouver tous les chemins possibles entre deux nœuds.
3. dls(): Recherche en profondeur limitée (DLS) pour trouver tous les chemins possibles entre deux nœuds à une profondeur spécifique.
4. iddfs(): Recherche itérative en profondeur (IDDFS) pour trouver tous les chemins possibles entre deux nœuds jusqu'à une profondeur spécifique.

3. IMPLÉMENTATION DE L'ALGORITHME D'EXTRACTION

L'implémentation de l'algorithme d'extraction comprend la définition de la classe Graph et de ses fonctions associées. Les fonctions de recherche ont été implémentés en utilisant des techniques de programmation récursive pour DFS et DLS, et une approche itérative à l'aide d'une file d'attente pour BFS. IDDFS utilise DLS avec une profondeur croissante jusqu'à atteindre la distance souhaitée.

4. COMPARAISON DES PERFORMANCES ENTRE BFS, DFS ET IDDFS

DFS : La recherche en profondeur peut être inefficace dans certains cas, car elle explore profondément les branches du graphe avant de passer aux autres branches. Ainsi, elle pourrait passer beaucoup de temps à explorer des chemins qui ne correspondent pas à la distance souhaitée.

BFS : La recherche en largeur explore tous les nœuds à une profondeur donnée avant de passer à la profondeur suivante. Bien qu'elle trouve les chemins les plus courts en premier, elle explore également tous les chemins possibles jusqu'à la distance souhaitée, ce qui peut être inefficace si le graphe est dense.

IDDFS : La recherche itérative en profondeur combine les avantages de DFS et BFS. Elle explore les nœuds à différentes profondeurs en utilisant DFS, tout en augmentant progressivement la profondeur de recherche jusqu'à atteindre la distance souhaitée. Cela permet à IDDFS d'éliminer les chemins trop longs ou trop courts rapidement, et de trouver efficacement les chemins correspondant à la distance souhaitée.

D'après mes observations, IDDFS montre une meilleure performance que BFS et DFS pour rechercher des chemins d'une distance spécifique entre deux nœuds. Cette différence de performance peut être attribuée à la manière dont

IDDFS combine les avantages de DFS et BFS pour explorer efficacement les chemins possibles.

V. MODULE DE SORTIE

1. PRINCIPE DU MODULE DE SORTIE

Le module de sortie est responsable de la conversion des résultats de l'analyse lexicale en une représentation lisible et compréhensible pour l'utilisateur. Il génère une chaîne de caractères formatée qui décrit les réactions, les inhibitions et les informations associées, telles que les enzymes, les substrats, les produits, les unités et les constantes de vitesse.

2. FORMATS DE SORTIE PRIS EN CHARGE

Le format de sortie principal est une chaîne de caractères textuelle, qui peut être facilement affichée à l'écran ou écrite dans un fichier. La structure de la chaîne de caractères est conçue pour faciliter la compréhension des réactions et des inhibitions par l'utilisateur.

3. IMPLÉMENTATION DU MODULE DE SORTIE

L'implémentation du module de sortie se fait principalement à travers la fonction operator << qui prend en paramètre un flux de sortie (ostream) et une instance de la classe analyseLexicale. La fonction parcourt les réactions et les inhibitions stockées dans l'instance d'analyse lexicale et construit une chaîne de caractères formatée pour représenter les informations associées. Cette chaîne est ensuite écrite dans le flux de sortie.

La méthode printReaction est une fonction membre de la classe analyseLexicale qui permet d'afficher une réaction spécifique en générant une chaîne de caractères formatée à partir de l'indice de la réaction dans la liste des réactions. Cette méthode est utile pour afficher individuellement les réactions à des fins de déboggage ou d'analyse.

En résumé, le module de sortie est conçu pour présenter les informations extraites par l'analyse lexicale sous une forme lisible et compréhensible pour l'utilisateur. Il génère une chaîne de caractères formatée à partir des réactions et des inhibitions stockées dans l'instance d'analyse lexicale, en utilisant la fonction operator<< et la méthode printReaction.

VI. DÉMO

1. PRÉSENTATION DE LA DÉMO

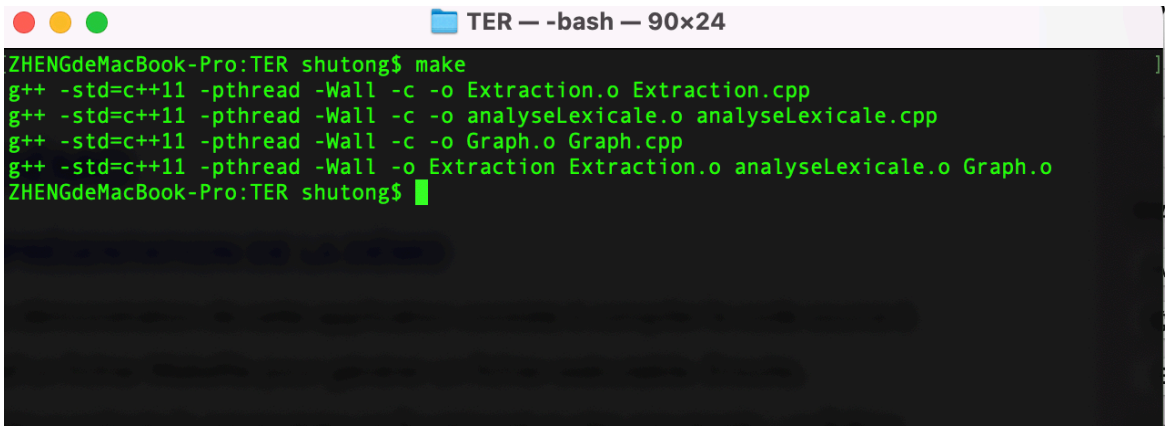
La démonstration de cette application consiste à compiler le code source à l'aide d'un fichier Makefile pour générer un fichier exécutable. Ensuite, l'utilisateur exécute le programme en fournissant quatre arguments : un fichier de réactions au format .ssa, un substrat de réaction, un produit de réaction et une distance. Le programme analysera les réactions du fichier, appliquera l'algorithme d'extraction et enregistrera les résultats dans un fichier output.txt situé dans le répertoire courant.

2. SCÉNARIOS DE DÉMONSTRATION

1. Compiler le programme :

Ouvrez un terminal et accédez au répertoire contenant le code source et le fichier Makefile.

Exécutez la commande 'make' pour compiler le code et générer le fichier exécutable.



2. Exécuter le programme avec des arguments :

Dans le terminal, exécutez le fichier exécutable avec les arguments requis. Par exemple :

```
g++ -std=c++11 -pthread -Wall -o Extraction Extraction.o analyseLexicale.o Graph.o
ZHENGdeMacBook-Pro:TER shutong$ ./Extraction reaction.ssa entre sortie distance
```

où :

reaction.ssa est le fichier contenant les réactions.

entre est le substrat de la réaction.

sortie est le produit de la réaction.

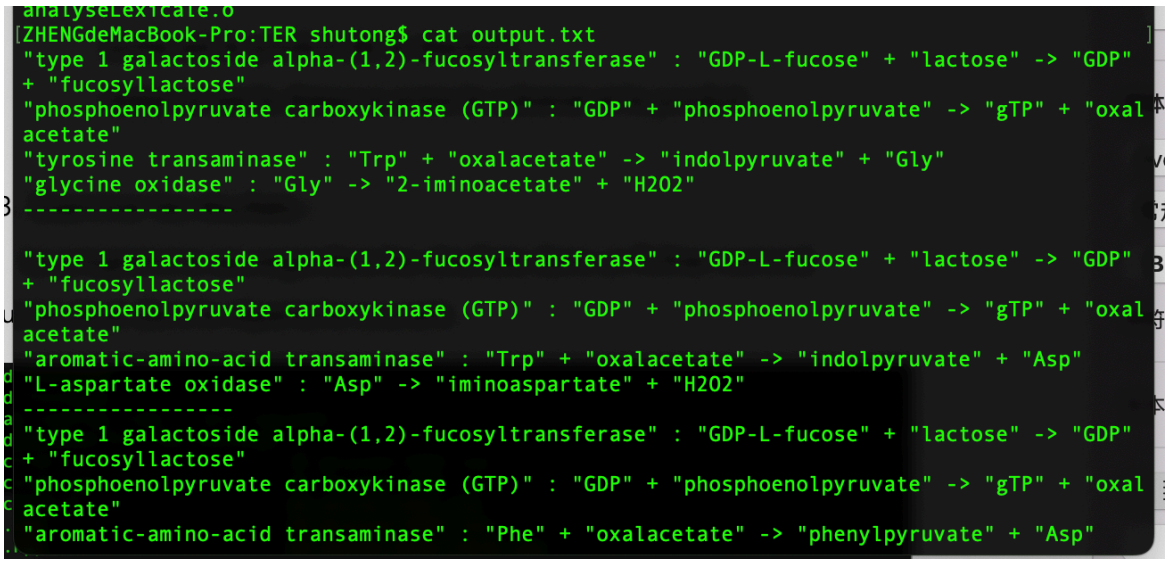
distance est la distance entre le substrat et le produit.

3. Examiner les résultats :

Le programme sauvegardera les résultats dans un fichier nommé output.txt situé dans le répertoire courant.

```
thread finished
thread finished
Program running time: 7 milliseconds
ZHENGdeMacBook-Pro:TER shutong$ ls
Extraction                  d2-glucose-H2O2.ssa
Extraction.cpp              d2-lactose-H2O2.ssa
Extraction.o                d3-aceton-H2O2.ssa
Graph.cpp                   d3-glucose-H2O2.ssa
Graph.hpp                   d4-aceton-H2O2+inhib.ssa
Graph.o                     d4-aceton-H2O2.ssa
Makefile                    d4-lactose-H2O2.ssa
analyseLexicale.cpp         output.txt
analyseLexicale.hpp         structData.hpp
analyseLexicale.o
ZHENGdeMacBook-Pro:TER shutong$
```

Ouvrez le fichier output.txt pour consulter les résultats générés par le programme.



3. RÉSULTATS DE LA DÉMO

Après avoir exécuté le programme avec les arguments fournis, les résultats de l'analyse des réactions, de l'application de l'algorithme d'extraction et de la génération du module de sortie seront sauvegardés dans le fichier output.txt. Les résultats incluront les informations sur les réactions, les inhibitions, les enzymes, les substrats, les produits, ainsi que les chemins trouvés entre le substrat et le produit avec la distance spécifiée. Les résultats démontreront la capacité du programme à analyser les réactions biochimiques, à extraire des informations pertinentes et à les présenter de manière lisible et compréhensible pour l'utilisateur.