République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
ECOLE SUPÉRIEURE EN INFORMATIQUE
8 Mai 1945 - Sidi-Bel-Abbès

الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
المدرسة العليا للإعلام الآلي
8 ماي 1945 - سيدي بلعباس

Ecole Supérieure en Informatique
08-MAI-1945 Sidi Bel Abbes

Rapport de mini projet

Système de Recommandation de Livres

Présenté par :

- Ghandouz Amina
- Benghenima Hafsa
- Benahmed Firdaws

Encadrant :
Dr. Nassima DIF

Date de publication : Décembre 2024

Table des matières

1 Introduction

Les systèmes de recommandation font partie intégrante de nos vies, ils nous aident à mieux acheter, trouver du nouveau contenu, etc. Pour les entreprises, ces systèmes sont un outil essentiel pour augmenter leurs ventes et la satisfaction des clients.
La majorité des bibliothèques n'ont pas de systèmes de recommandation de livres, ce qui les empêche de faire découvrir aux utilisateurs des livres qui pourraient les intéresser. Par conséquent, si les bibliothèques pouvaient recommander automatiquement des livres, elles pourraient mieux promouvoir la lecture.

Le but de ce projet est de créer un système de recommandation de livres basé sur les livres que les utilisateurs lisent et qui ont des intérêts similaires.

Les retombées du projet seraient d'avoir un système fonctionnel pouvant être intégré aux bibliothèques, permettant aux utilisateurs de découvrir des livres intéressants qu'ils ne connaissaient pas.

Dans ce projet, nous explorons les techniques d'apprentissage automatique non supervisé, en particulier les algorithmes d'association, pour développer un système de recommandation de livres. Nous avons expérimenté trois algorithmes : **Apriori**, **ECLAT**, et **FP-Growth**, afin de découvrir les associations les plus pertinentes entre les livres. Après une analyse comparative, nous avons sélectionné le meilleur modèle et l'avons déployé pour fournir des recommandations efficaces.

Le code complet du projet est disponible sur GitHub via le lien :

`https://github.com/hafsabn/Book-Recommendation-System`

2 Type de Système de Recommandation

Le système de recommandation que nous avons développé utilise des algorithmes basés sur les règles d'association, en particulier Apriori, Eclat, et FP-Growth, qui sont des techniques de découverte d'ensembles fréquents.

Dans ce type de système, nous identifions des ensembles de livres fréquemment lus ensemble par plusieurs utilisateurs. Lorsque qu'un utilisateur a lu un livre qui fait partie de ces ensembles fréquents, nous lui recommandons l'autre livre du même ensemble. Par exemple, si un ensemble fréquent est formé des livres A et B (lisent ensemble souvent par les utilisateurs), et qu'un utilisateur a déjà lu le livre A, le système recommande automatiquement le livre B.

3 Sources de Données Utilisées

Nous avons utilisé deux datasets principaux pour ce projet : `books_df` et `ratings_df`.

- Le dataset `book_df` contient 271 360 lignes et 8 colonnes, représentant les informations relatives aux livres, telles que le titre, l'auteur...
- Le dataset `ratings_df` contient 1 149 780 lignes et 3 colonnes, représentant les évaluations des utilisateurs. Chaque ligne associe un identifiant d'utilisateur User-ID, un identifiant de livre ISBN, et une évaluation (rating).

Nous avons filtré le dataset `ratings_df` pour ne conserver que les évaluations dont la note est supérieure à 5. Cette étape permet de se concentrer uniquement sur les livres ayant reçu des évaluations relativement positives, afin d'éviter d'inclure des évaluations faibles qui pourraient biaiser les recommandations. Ensuite, nous avons fusionné ce sous-ensemble avec le dataset `books_df` en utilisant l'ISBN comme clé de jointure. Cela nous a permis d'obtenir un dataset final contenant à la fois les titres des livres et leurs évaluations positives, essentiel pour générer des recommandations précises.

	User-ID	ISBN	Book-Rating	Book-Title
0	276729	0521795028	6	The Amsterdam Connection : Level 4 (Cambridge ...
1	276744	038550120X	7	A Painted House
2	276747	0060517794	9	Little Altars Everywhere
3	276747	0671537458	9	Waiting to Exhale
4	276747	0679776818	8	Birdsong: A Novel of Love and War

FIGURE 1 – Extrait du dataset final fusionné.

4 Prétraitement et Nettoyage des Données

Après la fusion des datasets, nous avons effectué un nettoyage pour préparer les données avant de les utiliser pour les recommandations. Plusieurs étapes ont été réalisées pour garantir la qualité du dataset fusionné :

- Doublons : Des doublons ont été identifiés, notamment des évaluations répétées pour un même utilisateur et un même livre. Ces doublons ont été supprimés afin de garantir l'intégrité des données.
- Valeurs manquantes : Aucune valeur manquante n'a été trouvée dans les colonnes essentielles du dataset. Par conséquent, ce problème n'a pas nécessité de traitement particulier.
- Filtrage des évaluations : Nous avons sélectionné uniquement les évaluations supérieures à 5, afin de nous concentrer sur les livres ayant reçu des évaluations positives.

Ces étapes ont permis d'obtenir un dataset propre et prêt à être utilisé pour les recommandations.

Afin de visualiser les effets du nettoyage, nous avons comparé la distribution des évaluations avant et après le prétraitement :

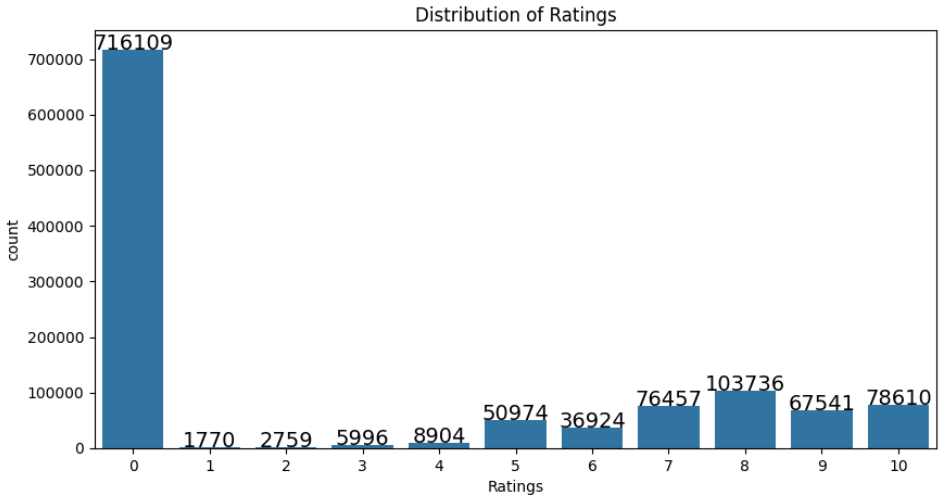


FIGURE 2 – Distribution des évaluations avant le prétraitement

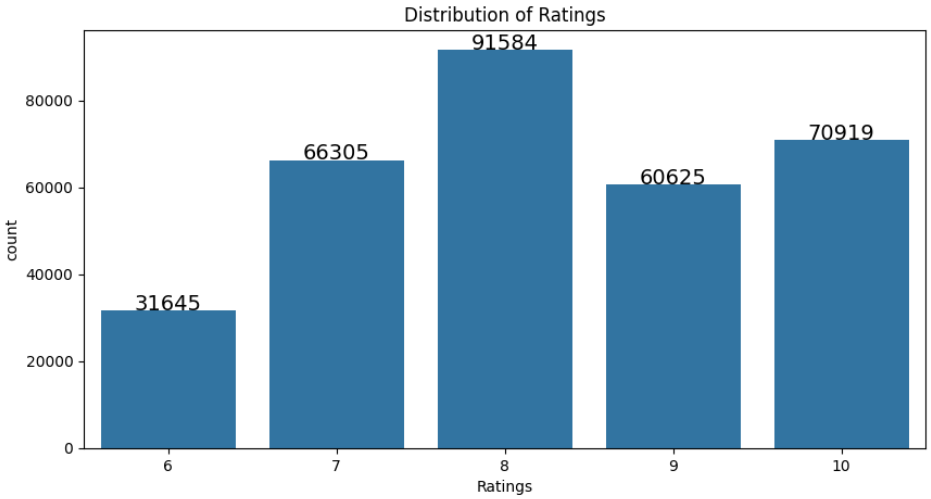


FIGURE 3 – Distribution des évaluations après le prétraitement

5 Extraction de motifs fréquents

5.1 Concept et importance dans l'exploration de données

L'extraction de motifs fréquents (Frequent Pattern Mining) est un processus clé en exploration de données (data mining) qui consiste à identifier des motifs, des séquences ou des ensembles d'éléments qui apparaissent fréquemment dans un ensemble de données. Ces motifs peuvent révéler des tendances cachées dans les données et être utilisés pour diverses applications, comme les systèmes de recommandation, l'analyse de panier d'achat, ou même la détection d'anomalies. L'extraction de motifs fréquents est cruciale car elle permet de comprendre les relations entre les données, d'améliorer la prise de décision, et d'optimiser les processus dans de nombreux domaines, notamment le marketing, la vente, et l'analyse comportementale.

5.2 Définitions

5.2.1 Transactions, items et ensembles d'éléments fréquents (frequent itemsets) :

- **Transactions** : Une transaction dans un contexte d'exploration de données est un enregistrement individuel dans une base de données, souvent représenté par un ensemble d'éléments ou d'objets associés. Par exemple, dans notre système de recommandation de livres, une transaction est la lecture d'un livre par un utilisateur.
- **Items (éléments)** : Un item est un objet individuel ou un élément d'une transaction. Dans le contexte d'une boutique en ligne, un item pourrait être un produit spécifique, comme un livre particulier.
- **Ensembles d'éléments fréquents** : Un ensemble d'éléments fréquent est un groupe d'items qui apparaissent ensemble dans un nombre suffisamment élevé de transactions. L'extraction de ces ensembles est au cœur de l'extraction de motifs fréquents, car ils permettent de découvrir des relations entre les éléments.
- **Ensembles d'éléments candidats** : Un ensemble d'éléments candidats est un groupe d'items qui pourrait potentiellement être fréquent. Ces ensembles sont générés pendant le processus d'exploration de motifs fréquents et nécessitent une validation en calculant leur support pour déterminer s'ils sont effectivement fréquents. Par exemple, si les items A et B apparaissent souvent séparément dans les transactions, $\{A, B\}$ pourrait être un ensemble candidat à évaluer.

5.2.2 Support, confiance et lift (mesures clés) :

- **Support** : Le support mesure la fréquence d'apparition d'un ensemble d'éléments dans l'ensemble des transactions. Il est calculé comme le pourcentage de transactions dans lesquelles un ensemble d'éléments donné apparaît. Le support est utilisé pour évaluer la pertinence d'un motif en fonction de sa fréquence dans les données.

$$\text{Support}(A) = \frac{\text{Nombre de transactions contenant } A}{\text{Total des transactions}}$$

— **Confiance (Confidence)** : La confiance est une mesure de la probabilité qu'un élément apparaisse dans une transaction, étant donné qu'un autre élément y est présent. Elle est utilisée pour évaluer la force d'une règle d'association. Par exemple, la confiance d'une règle "Si l'utilisateur achète le livre X, alors il achètera aussi le livre Y" est la probabilité d'acheter Y, sachant que X a été acheté.

$$\text{Confiance}(A \rightarrow B) = \frac{\text{Support}(A \cup B)}{\text{Support}(A)}$$

— **Lift** : Le lift mesure l'augmentation de la probabilité de trouver deux éléments ensemble par rapport à ce qui serait attendu si les éléments étaient indépendants. Un lift supérieur à 1 indique une forte association entre les éléments, tandis qu'un lift inférieur à 1 suggère qu'ils sont moins susceptibles d'apparaître ensemble que prévu par hasard.

$$\text{Lift}(A \rightarrow B) = \frac{\text{Confiance}(A \rightarrow B)}{\text{Support}(B)}$$

5.3 Défis dans l'extraction de motifs

L'extraction de motifs fréquents présente plusieurs défis importants :

— **Problème de complexité computationnelle** : L'extraction de motifs fréquents peut devenir très coûteuse en termes de temps et de mémoire, en particulier pour de grandes bases de données. La croissance combinatoire du nombre d'ensembles d'éléments possibles rend les calculs inefficaces.

— **Manipulation de données volumineuses et variables** : Les grandes bases de données avec un grand nombre de transactions et d'items rendent difficile l'extraction de motifs fréquents. De plus, les données peuvent être complexes et variées, rendant l'identification de motifs significatifs plus difficile.

— **Sélection des paramètres** : Le choix des seuils de support, de confiance et de lift peut avoir un impact important sur les résultats. Des seuils trop élevés peuvent entraîner la perte de motifs potentiellement intéressants, tandis que des seuils trop faibles peuvent entraîner des motifs trop généraux ou non significatifs.

— **Variabilité des motifs** : Les motifs peuvent varier en fonction des contextes ou des utilisateurs. Il peut être difficile de distinguer les motifs pertinents pour différents segments d'utilisateurs ou contextes d'utilisation.

6 Techniques et Algorithmes Utilisés

Pour les recommandations, nous avons utilisé trois techniques populaires basées sur les règles d'association : **Apriori**, **Eclat**, et **FP-Growth**.

6.1 Eclat

L'algorithme ECLAT est une technique utilisée en fouille de données pour identifier des ensembles d'éléments fréquents. Il est couramment utilisé dans les systèmes de recommandation, comme dans notre projet, ainsi que dans les suggestions de films ou de chansons, ou encore dans l'analyse de marché pour déterminer les produits que les gens achètent souvent ensemble (exemples : lait et pain, burger et frites, fruits et légumes...).

6.1.1 Théorie de l'Algorithme

Etape01 : Représentation verticale :

Contrairement aux tables traditionnelles, ECLAT travaille avec un format de données vertical. Chaque élément est représenté par une liste de transactions (TID-list).

Exemple :

Trans-ID	A	B
T1	✓	✗
T2	✗	✓
T3	✓	✓
T4	✓	✗
T5	✗	✓

- Item A : {T1, T3, T4}
- Item B : {T2, T3, T5}

Etape02 : Intersection des TID-lists

Pour trouver des combinaisons d'éléments (par exemple {A, B}), ECLAT intersecte leurs TID-lists.

- {A} = {T1, T3, T4}, {B} = {T2, T3, T5}
- {A, B} = {T3} (car A et B apparaissent tous deux dans la transaction T3).

Etape03 :Support Count

L'algorithme compte le nombre de transactions contenant chaque combinaison (support).

Si le support atteint un seuil prédéfini (appelé support minimum), la combinaison est considérée comme fréquente.

(Exemple : Dans notre exemple, le support minimum est de 1).

Enfin, ce processus est répété pour toutes les combinaisons. Les combinaisons avec un support inférieur au seuil défini sont éliminées.

6.1.2 Application dans la vie réelle

Étape 1 : Table des transactions

Trans-ID	Pain	Lait	Beurre
T1	✓	✓	✗
T2	✓	✗	✓
T3	✓	✓	✓
T4	✗	✓	✓

Étape 2 : TID-lists

- Lait : {T1, T3, T4}
- Pain : {T1, T2, T3}
- Beurre : {T2, T3, T4}

Étape 3 : Recherche des ensembles fréquents (intersections)

1. {Lait, Pain} :
   - Lait = {T1, T3, T4}
   - Pain = {T1, T2, T3}
   - Intersection = {T1, T3}
   - Support = 2
2. {Lait, Beurre} :
   - Lait = {T1, T3, T4}
   - Beurre = {T2, T3, T4}
   - Intersection = {T3, T4}
   - Support = 2
3. {Pain, Beurre} :
   - Pain = {T1, T2, T3}
   - Beurre = {T2, T3, T4}
   - Intersection = {T2, T3}
   - Support = 2
4. {Lait, Pain, Beurre} :
   - Lait = {T1, T3, T4}
   - Pain = {T1, T2, T3}

- Beurre = {T2, T3, T4}
- Intersection = {T3}
- Support = 1

Étape 4

Avec un support minimum de 2, la combinaison {Lait, Pain, Beurre} est éliminée.

6.1.3 Avantages de l'algorithme ECLAT

- **Efficacité avec des données éparses** : Convient aux ensembles de données contenant de nombreux éléments dans relativement peu de transactions.
- **Implémentation simple** : Se concentre sur l'intersection des listes plutôt que sur la génération et le test de toutes les combinaisons possibles.
- **Utilisation de la mémoire** : Plus efficace en mémoire grâce à l'utilisation de la représentation verticale.

6.1.4 Inconvénients de l'algorithme ECLAT

- **Exigence en mémoire élevée** : Pour les grands ensembles de données, le stockage des TID-lists pour tous les éléments et leurs combinaisons peut être gourmand en mémoire.
- **Manque de scalabilité** : Les opérations d'intersection peuvent devenir coûteuses en termes de calcul avec l'augmentation du nombre d'éléments.
- **Données statiques** : Non adapté aux ensembles de données dynamiques où les transactions sont fréquemment ajoutées ou mises à jour, car les TID-lists doivent être recalculées, ce qui est chronophage.

6.2 Apriori

L'algorithme Apriori est un outil puissant pour découvrir des relations fréquentes entre des éléments dans des bases de données transactionnelles. Dans notre contexte, il permet d'identifier les livres lus ensemble par les utilisateurs et de déduire des règles d'association prédictives.

6.2.1 Théorie de l'Algorithme Apriori

1. **Génération de la table des transactions** :
   - La première étape consiste à construire une table de transactions où chaque ligne représente un utilisateur et contient la liste des livres qu'il a lus.
2. **Itération avec $K = 1$ (Candidats de taille 1)** :
   - Identifier tous les éléments uniques (livres dans ce cas).
   - Calculer le support de chaque élément (à savoir la proportion des transactions contenant l'élément).
   - Retenir uniquement les éléments dont le support est supérieur ou égal à $min\_support$. Ces éléments deviennent les ensembles fréquents de taille 1 ($itemsets$).
3. **Itération pour $K = 2, K = 3, \ldots$** :
   - Combiner les condidats de taille K.
   - Calculer le support de chaque candidat.
   - Retenir les ensembles ayant un support supérieur ou égal à $min\_support$.
   - Répéter jusqu'à ce qu'aucun candidat n'ait un support suffisant.
4. **Génération des règles d'association** :
   - Pour chaque ensemble fréquent, générer toutes les règles possibles.
   - Calculer la confiance($Confidence$) de chaque règle (proportion des transactions contenant l'anticident qui contiennent aussi le consequent).
   - Retenir uniquement les règles ayant une confiance supérieure ou égale au $min\_confidence$.

6.2.2 Exemple Pratique

Table des Transactions

Utilisateur	Livres lus
U1	{A, B, C}
U2	{A, B}
U3	{A, C}
U4	{A, B, C}
U5	{B, C}

Paramètres :

- *min_ support* = 0,7 (70%).
- *confidence* = 50%.

Étape 1 : $K = 1$

- **Candidats** : {A}, {B}, {C}.
- **Calcul du support** :
  - Support($\{A\}$) = $\frac{4}{5} = 0,8$.
  - Support($\{B\}$) = $\frac{4}{5} = 0,8$.
  - Support($\{C\}$) = $\frac{4}{5} = 0,8$.
- **Ensembles fréquents** : {A}, {B}, {C} (tous à *min_ support* $\geq 0,7$).

Étape 2 : $K = 2$

- **Candidats** : {A, B}, {A, C}, {B, C}.
- **Calcul du support** :
  - Support($\{A, B\}$) = $\frac{3}{5} = 0,6$.
  - Support($\{A, C\}$) = $\frac{3}{5} = 0,6$.
  - Support($\{B, C\}$) = $\frac{4}{5} = 0,8$.
- **Ensembles fréquents** : {B, C}.

Étape 3 : $K = 3$

- Pas de candidats (aucun ensemble plus grand que {B, C} n'atteint le support minimum).

Étape 4 : Génération des règles d'association

- **Ensemble fréquent** : {B, C}.
- **Règles possibles** :
  - {B} → {C} : Confiance = $\frac{\text{Support}(\{B,C\})}{\text{Support}(\{B\})} = \frac{0,8}{0,8} = 1$ (100%).
  - {C} → {B} : Confiance = $\frac{\text{Support}(\{B,C\})}{\text{Support}(\{C\})} = \frac{0,8}{0,8} = 1$ (100%).
- **Règles retenues** : {B} → {C}, {C} → {B}.

6.3 FP-Growth

6.3.1 Algorithme FP-Growth : Une Version Améliorée de l'Apriori :

L'algorithme **FP-Growth** (Frequent Pattern Growth) est une version améliorée de l'algorithme **Apriori**, spécialement conçu pour surmonter certaines des limitations d'Apriori, en particulier son inefficacité en termes de coûts computationnels et d'utilisation de mémoire. FP-Growth est plus efficace car il évite la génération de candidats d'ensembles d'éléments fréquents (itemsets) comme le fait l'algorithme Apriori. Au lieu de cela, FP-Growth utilise une structure de données compacte appelée **l'arbre FP** (Frequent Pattern Tree) et extrait directement les ensembles d'éléments fréquents à partir de cet arbre. Voici une explication détaillée de l'algorithme FP-Growth avec un exemple à partir d'un jeu de données aléatoire.

6.3.2 Différences Clés entre Apriori et FP-Growth :

- **Génération de Candidats** :
  - **Apriori** génère des candidats d'ensembles d'éléments fréquents à chaque itération et les taille ensuite en fonction du seuil de support minimum.
  - **FP-Growth** évite la génération de candidats et utilise une structure d'arbre efficace pour découvrir les ensembles fréquents d'éléments.
- **Efficacité** :
  - **Apriori** nécessite plusieurs passages sur l'ensemble des données, à chaque fois en générant des ensembles candidats de plus en plus grands.
  - **FP-Growth** ne nécessite que deux passages sur le jeu de données : un pour construire l'arbre FP et un autre pour exploiter cet arbre.

6.3.3 Explication de l'Algorithme FP-Growth

Étape 1 : Construction de l'Arbre FP

L'arbre FP est une représentation compressée du jeu de données. Il stocke les ensembles d'éléments fréquents dans une structure en arbre, où les chemins de l'arbre représentent les combinaisons d'ensembles d'éléments.

- **Nœud Racine** : Représente un ensemble vide.
- **Nœuds Enfants** : Représentent les éléments du jeu de données.
- **Arêtes** : Représentent les fréquences des éléments.

Étape 2 : Extraction des Ensembles Fréquents à Partir de l'Arbre FP

Une fois l'arbre FP construit, l'algorithme utilise cet arbre pour trouver les ensembles d'éléments fréquents. À partir du bas de l'arbre (les éléments ayant les plus faibles supports), il trace des chemins jusqu'à la racine pour découvrir les ensembles fréquents d'éléments.

6.3.4 Exemple Pratique

Prenons un **jeu de données aléatoire** :

ID Transaction	Livres
T1	{A, B, D, E}
T2	{B, D, E}
T3	{A, B, D}
T4	{B, E}
T5	{A, D, E, F}

Étape 1 : Prétraitement du Jeu de Données

— **Calcul des fréquences des éléments** (compte de support pour chaque élément) :

ID Transaction	A	B	D	E	F
T1	1	1	1	1	0
T2	0	1	1	1	0
T3	1	1	1	0	0
T4	0	1	0	1	0
T5	1	0	1	1	1

$$\text{Support(livre)} = \sum_{i=1}^{n} \text{Présence du livre dans la transaction } T_i$$

avec :

— n représente le nombre total de transactions dans le jeu de données.
— La présence d'un livre dans une transaction est indiquée par 1, et l'absence par 0.

Exemple avec le livre A :

$$\text{Support(A)} = \text{Présence dans } T1 + \text{Présence dans } T2 + ... + \text{Présence dans } T5$$

$$\text{Support(A)} = 1 + 0 + 1 + 0 + 1 = 3$$

De la même manière, nous pouvons calculer les supports pour les autres livres :

$$\text{Support(B)} = 1 + 1 + 1 + 1 + 0 = 4$$
$$\text{Support(D)} = 1 + 1 + 1 + 0 + 1 = 4$$
$$\text{Support(E)} = 1 + 1 + 0 + 1 + 1 = 4$$
$$\text{Support(F)} = 0 + 0 + 0 + 0 + 1 = 1$$

— **Application du seuil de support minimum** (disons que le support minimum = 3). Les éléments dont le support est inférieur à 3 sont éliminés, ce qui donne :
  — A, B, D, E.
— **Tri des éléments** : On trie les éléments selon leur fréquence dans les transactions. Le tri donne l'ordre suivant (du plus fréquent au moins fréquent) :
  — B (4), D (4), E (4), A (3).

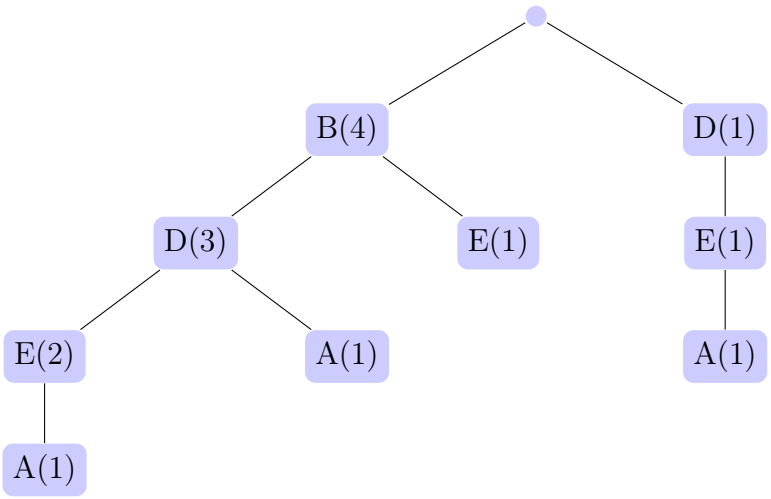
Étape 2 : Construction de l'Arbre FP

L'arbre FP est construit en ajoutant les transactions de manière compressée. Chaque transaction est ajoutée de manière à ce que les éléments soient triés dans l'ordre des éléments les plus fréquents. Les transactions peuvent partager des préfixes communs dans l'arbre, ce qui permet une compression des données.

Transactions après suppression des éléments peu fréquents (F) et triées par fréquence des éléments :

ID Transaction	Livres
T1	{B, D, E, A}
T2	{B, D, E}
T3	{B, D, A}
T4	{B, E}
T5	{D, E, A}

Construction de l'arbre FP :



Étape 3 : Extraction des Ensembles Fréquents à partir de l'Arbre FP

Une fois l'arbre FP construit, on peut en extraire les ensembles fréquents d'éléments en suivant les chemins de l'arbre. Chaque chemin correspond à un ensemble d'éléments

fréquents, et les combinaisons possibles sont extraites en remontant l'arbre. En appliquant cet algorithme à notre ensemble de données, nous serons capables d'identifier des règles d'association réelles basées sur les habitudes de lecture des utilisateurs. Ces règles associent des livres qui sont fréquemment lus ensemble, ce qui peut être utilisé pour générer des recommandations. Par exemple, si une règle fréquente de type Livre A, Livre B → Livre C est extraite, cela signifie qu'un utilisateur ayant lu les livres A et B est susceptible de lire également le livre C.

6.3.5 Pseudo-Code de l'Algorithme FP-Growth

Voici un pseudo-code simple de l'algorithme FP-Growth :

Algorithm 1: Algorithme FP-Growth

Data: Un ensemble de transactions avec des éléments
Result: Ensembles d'éléments fréquents
Input: *min_support* : Seuil minimum de support
Output: Ensembles d'éléments fréquents ayant un support supérieur ou égal à *min_support*

```
for élément i dans l'ensemble de données do
    Compter la fréquence de i
end
for ensemble d'éléments I do
    Construire l'arbre FP en insérant les ensembles d'éléments dans la structure de l'arbre selon la fréquence et l'ordre des éléments
end
for modèle fréquent dans l'arbre FP do
    Générer la base de motifs conditionnels en extrayant les sous-ensembles des modèles fréquents
    Générer l'arbre FP conditionnel en appliquant de manière récursive l'algorithme FP-Growth sur la base de motifs conditionnels
end
```

7 Étude Comparative et Sélection du Meilleur Modèle

Après avoir appliqué les algorithmes Apriori, Eclat, et FP-Growth, nous avons comparé leurs performances afin de sélectionner le modèle le plus adapté pour notre système de recommandation. Cette comparaison s'est basée sur plusieurs critères essentiels :

Temps d'exécution : Chaque algorithme a été évalué en termes de temps nécessaire pour traiter le dataset final et générer les règles d'association.

Nombre de règles extraites : Nous avons analysé la quantité de règles générées par chaque algorithme, en prenant en compte leur pertinence et leur diversité.

Simplicité des règles : Un autre critère a été la lisibilité et la simplicité des règles d'association pour leur interprétation par les utilisateurs.

Support et confiance minimaux : Les valeurs de support et de confiance minimaux influencent directement la qualité des recommandations. Ces paramètres ont été ajustés pour chaque algorithme afin de garantir des résultats optimaux.

7.1 Résultats Comparatifs

Apriori : Cet algorithme a généré des règles d'association pertinentes, mais il était le plus lent en termes de temps d'exécution, surtout avec des datasets de grande taille.

Eclat : Bien que plus rapide qu'Apriori, Eclat a parfois produit des règles légèrement redondantes, nécessitant un filtrage supplémentaire.

FP-Growth : Cet algorithme s'est distingué par son efficacité, générant rapidement des règles pertinentes avec une utilisation optimale des ressources.

7.2 Meilleur Modèle Sélectionné

Sur la base des résultats obtenus, FP-Growth a été identifié comme le meilleur modèle. Les critères principaux pour ce choix ont été :

- Son temps d'exécution rapide sur des datasets volumineux.
- La pertinence des règles générées.
- Sa capacité à s'adapter aux variations des paramètres comme le support et la confiance minimaux.