**Projet de Recherches : complétion (semi-)automatique**

**Définition :**

**Complétion automatique :** Processus par lequel un système prédit et complète une entrée en fonction de données ou de contextes disponibles.

**Complétion semi-automatique :** Assistance où le système propose des options, mais laisse l'utilisateur faire la sélection finale.

**Application :**

Saisie sur les claviers, complétion de code (VS Code), moteurs de recherche, assistants virtuels, etc…

**Recherche sur les approches**

1. Utilisation d’algorithmes avec des règles préprogrammées (N-gramme (séquence donnée), dictionnaires statiques ou listes)

Implémentation rapide et simple mais pas/peu de flexibilité

1. Utilisation de statistiques pour prédire les séquences (Modèle de Markov (2ème partie), Méthode TD-IDF)

Suggestions rapides mais pas de contexte donc pas de compréhension sémantique)

1. Utilisation de l’intelligence artificielle et réseaux neuronaux (RNN ou LSTM, GPT ou BERT)

Efficaces même face à des problèmes complexes mais demandes bcp de calculs et de ressources

1. Utilisation de plusieurs approches pour améliorer les résultats (1 avec 3 pour vitesse et contexte)

Compromis entre performance, vitesse et précision mais très compliqué à implémenter

1. Utilisation d’un système qui s’améliore en temps réel grâce à l’utilisateur (fait selon les choix des utilisateurs)

Très adaptable mais complexe et dépend des données collectées

**Recherche sur les algorithmes de distance**

**Définition :**

Un algorithme de calcul de distance mesure la similarité (ou la différence) entre deux objets comme du texte, des vecteurs, des chaînes de caractères, etc...

**Application :**

Correction de l’orthographe, traitement de texte, alignement de séquences ADN, recherche d’information, etc…

**Distance d’édition (ou de Levenshtein) :**

1. **Définition :**

On recherche le nombre minimal d'opérations élémentaires comme les insertions, les suppressions, les substitutions nécessaires pour transformer une chaîne en une autre.

1. **Complexité en temps** :

O(n×m) (où n et m sont les longueurs des deux chaînes).

1. **Complexité dans l’espace :**

O(n×m) (optimisé à O(min (n, m)) dans des implémentations avancées).

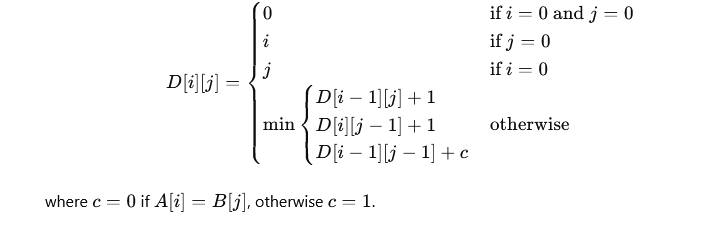
1. **Avantages/ Limites :**

Implémentation simple et intuitive mais ne prend pas en compte les permutations ou les erreurs plus complexes.

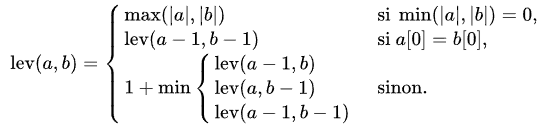
Pour calculer la distance de Levenshtein il faut inclure :

1. **Insertion** : Ajouter un caractère.
2. **Suppression**: Supprimer un caractère.
3. **Substitution**: Remplacer un caractère par un autre.

Formule de récurrence de la distance :



**Implémentation simple en python :**

****

(à ecrire en latex)

def levenshtein\_distance(a, b):

n, m = len(a), len(b)

dp = [[0] \* (m + 1) for \_ in range(n + 1)]

for i in range(n + 1):

for j in range(m + 1):

if i == 0:

dp[i][j] = j

elif j == 0:

dp[i][j] = i

elif a[i - 1] == b[j - 1]:

dp[i][j] = dp[i - 1][j - 1]

else:

dp[i][j] = 1 + min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1])

return dp[n][m]

**Implémentation avec gestion de la mémoire :**

def optimized\_levenshtein(a, b):

n, m = len(a), len(b)

prev = list(range(m + 1))

curr = [0] \* (m + 1)

for i in range(1, n + 1):

curr[0] = i

for j in range(1, m + 1):

insert = curr[j - 1] + 1

delete = prev[j] + 1

substitute = prev[j - 1] + (0 if a[i - 1] == b[j - 1] else 1)

curr[j] = min(insert, delete, substitute)

prev, curr = curr, prev

return prev[m]

A partir de cette algorithme, il nous faut une base de donnée avec laquelle on va pouvoir comparer les mots et la saisie de l’utilisateur .

On obtient donc un algorithme qui doit avec :

* **En Entrée :** une saisie partielle de l’utilisateur (préfixe) et une base de données de mots ou de phrases.
* **Le Calcul :** évaluer la distance de Levenshtein entre la saisie partielle et chaque mot de la base.
* **Filtrer :** ne conserver que les mots avec la plus faible distance (nombre de mots que l’on veut afficher)
* **En Sortie :** retourner les suggestions classées par pertinence.

Algorithme en python :

def suggestions\_completion(partiel, dictionnaire, seuil=3, max\_suggestions=5):

distances = []

for mot in dictionnaire:

if abs(len(mot) - len(partiel)) <= seuil: # Filtrage rapide sur la longueur

dist = distance\_levenshtein(partiel, mot)

if dist <= seuil:

distances.append((mot, dist))

# Trier par distance et renvoyer les meilleures suggestions

distances.sort(key=lambda x: x[1])

return [mot for mot, \_ in distances[:max\_suggestions]]

**Distance de Damerau-Levenshtein:**

1. **Définition :**

C’est la distance de Levenshtein mais qui inclut une opération supplémentaire : la transposition de deux caractères adjacents.

1. **Complexité en temps** :

O(n×m) mais un peu plus couteux que la distance de Levenshtein en pratique

1. **Complexité dans l’espace :**

O(n×m)

1. **Avantages/ Limites :**

On a une meilleure gestion des erreurs typiques (« ab » avec « ba ») mais plus couteux de Levenshtein.

**Distance de Hamming :**

1. **Définition :**

On recherche le nombre de positions où deux chaînes de même longueur vont différer

1. **Complexité en temps** :

O(n) (où n est la longueur des deux chaînes).

1. **Complexité dans l’espace :**

O(n)

1. **Avantages/ Limites :**

Algorithme très rapide mais ne peut fonctionner que sur des chaînes de même longueur.

Si nécessaire, d’autres façons de calculer la distance existent (donc je peux développer) et s’il n’y a pas assez de détails, je peux développer les 3 parties ci-dessus.

**Recherche sur chaînes de Markov pour l’historique**

**Définition :**

Une chaîne de Markov est un modèle mathématique qui représente un système où la probabilité de passer d’un état à un autre dépend uniquement de l’état actuel.

**États :** Ce sont les différents éléments du système comme les pages web ou les actions possibles.

**Transitions :** Probabilité de passer d’un état à un autre.

**Matrice de transition :** Une table qui regroupe les probabilités de transition entre tous les états.

**Applications :**

Modélisation de processus aléatoires, analyse de séquences, prédictions, historique de navigation.

Ici on parlera de l’historique (navigation web, actions utilisateurs).

Les chaînes de Markov vont nous permettre de modéliser les transitions entre les pages visitées ou entre les actions effectuées par l’utilisateur.

Si un utilisateur navigue entre trois pages web A, B et C alors une chaîne de Markov pourrait ressembler à :

* Depuis A,

1. Probabilité d’aller à B : 0,6.
2. Probabilité d’aller à C : 0,4.

* Depuis B,

1. Probabilité d’aller à A : 0,3.
2. Probabilité d’aller à C : 0,7.

* Depuis C,

1. Probabilité d’aller à A : 0,5.
2. Probabilité d’aller à B : 0,5.

La matrice de transition serait :

[[0 0.6 0.4]

[0.3 0 0.7]

[0.5 0.5 0]]

Le but serait d’utiliser l’historique de navigation ou l’historique des actions pour modéliser les transitions probables entre différentes étapes.

Pour cela :

1. On collecte les données de navigation ou d’historique.
2. On compte les transitions observées entre les états pour calculer les probabilités de transition.
3. On construit la matrice de transition.

**Exemple :**

On a l’historique suivant : A→C→B→A→B→C.

* 1. Compter les transitions observées
* A→C : 1 fois.
* C→B : 1 fois.
* B→A : 1 fois.
* A→B : 1 fois.
* B→C : 1 fois.
  1. Calculer les probabilités de transition

Pour calculer la probabilité d’un état X vers un état Y on va diviser le nombre de transitions X→Y par toutes les transitions venant de X.

Transitions depuis A :

* Total des transitions depuis A : A→ C et A→ B donc 2.
* P(A→C) = 1/2 = 0.5
* P(A→B) = 1/2 = 0.5

Transitions depuis C :

* Total des transitions depuis C : C→ B donc 1.
* P(C→B) = 1/1 = 1

Transitions depuis B :

* Total des transitions depuis B : B→ A et B→C donc 2.
* P(B→A) = 1/2 = 0.5
* P(B→C) = 1/2 =0.5
  1. Résumé sous forme de matrice de transition

[[0 0.5 0.5]  
[0.5 0 0.5]  
[0 1 0 ]]

**Avantages et limites des Chaînes de Markov pour l’Historique**

Les chaînes de Markov permettent de prédire l’étape suivante en se basant sur les actions précédentes. Cela va permettre par la suite de personnaliser les suggestions pour chaque utilisateur.   
De plus, ce système est facile à mettre en place et à implémenter.

Cependant il existe certaines limites : une chaîne de Markov ne prend en compte que l’état actuel et pas l’historique complet. On a donc un problème de mémoire qui apparait. De plus, pour obtenir des probabilités fiables, il faut un très grand historique.