1. Algorithme de recherche
“Approximation d’un nom” :

Description de l’algorithme : Il faut pouvoir chercher une personne par son nom, son prénom ou un film par son titre, en recherche simple, et proposer une recherche avancée qui permet de sélectionner par d’autres critères (plage de temps pour la naissance d’une personne ou pour la création d’un film, par exemple). La recherche avancée doit pouvoir permettre de trouver une ressemblance approximative.
Les résultats pourront être triés selon différents critères (en particulier : date, ordre alphabétique, etc.)

Description du code et explication :

Ce code est écrit en PHP et contient deux fonctions principales qui sont “rechercheFilm” et “recherchepersonne”.
Voici une explication simple du fonctionnement de chacune de ces fonctions :

Fonction “rechercheFilm” :
La fonction sert à rechercher des films dans notre base de données en fonction de différents critères comme le titre, le type, la date de sortie, la durée, le genre, la note, et le nombre de votes pour que l’on puisse retrouver le film simplement.
Elle prend plusieurs paramètres, tels que $titre, $types, ou $genres qui sont les informations que l’utilisateur pourra fournir pour retrouver le film.

La fonction commence par construire une requête SQL qui sélectionne des données de films (comme le titre, le type, l'année, etc.) et leurs évaluations (note moyenne, nombre de votes) de deux tables liées, tout en comptant le nombre total de lignes. Ensuite, selon que les paramètres sont nuls ou non, elle ajoute différentes conditions à la requête SQL (comme “AND tb.primaryTitle ILIKE :titre` si `$titre” n'est pas nul).
Après avoir construit la requête, elle la prépare et y associe les valeurs des paramètres.
La fonction exécute ensuite la requête préparée et renvoie les résultats sous forme d'un tableau associatif.

Fonction “recherchepersonne” :
Cette fonction est similaire à `rechercheFilm`, mais elle est utilisée pour rechercher des personnes (comme des acteurs) dans la base de données.
Elle prend des paramètres tels que “$nom”, “$dateNaissance” ou “$dateDeces” pour filtrer les résultats de recherche.
La requête SQL et l’ajout des conditions est similaire à celle de “rechercheFilm”.
Après la construction de la requête, la fonction prépare et exécute la requête, puis renvoie les résultats.

Dans les deux fonctions, la préparation des requêtes SQL et le binding de paramètres sont utilisés pour la sécurité et l'efficacité de notre programme. La pagination est aussi gérée en

limitant le nombre de résultats renvoyés et en calculant un décalage avec l'attribut "offset" pour séparer les différents résultats dans des pages.

2. Algorithme Trouver des films ou personnes en commun "trouver des éléments en commun dans deux listes de façon optimale " :

Description de l'algorithme : Après avoir sélectionné deux films ou deux personnes, trouver les personnes (ou les films) en commun pour les deux. Par exemple, Thierry Lhermitte et Gérard Jugnot ont joués tous les deux dans Fallait Pas, Grosse Fatigue, Le père Noël est une ordure, Les 1001 nuits, Les Bronzés, Les Bronzés font du ski, Les Bronzés 3, Les Secrets professionnels du Dr Apfelglück, Papy fait de la résistance, Trafic d'influence, Vous n'aurez pas l'Alsace et la Lorraine.

Description du code et explication :

Ce code contient deux fonctions PHP, `FilmEnCommun` et `ActeurEnCommun`, qui servent à rechercher des informations dans une base de données de films et d'acteurs. Voici une explication simple pour chaque fonction :

Fonction "FilmEnCommun":
Le but de cette fonction est de trouver des films dans lesquels deux acteurs spécifiés ont joué ensemble.
Elle prend en paramètre un tableau "$data" contenant les noms des deux acteurs ("PrimaryName1" et "PrimaryName2").
La fonction vérifie d'abord si les deux noms d'acteurs sont fournis. Si ce n'est pas le cas, elle retourne un message d'erreur.
Ensuite, elle construit une requête SQL pour sélectionner les titres des films où les deux acteurs ont joué (en joignant plusieurs tables comme "title_principals", "title_basics", "name_basics" sur les attributs "actor" et "primaryName" pour trouver les acteurs et leurs noms).
La fonction prépare et exécute la requête, en reliant les noms des acteurs aux paramètres de la requête.
Les résultats sont retournés sous forme d'un tableau associatif.

Voici à quoi doit ressembler la requête :

```
$sql = "SELECT DISTINCT
            m.primaryTitle
        FROM
        title_principals p1
        JOIN title_principals p2
```

```
                ON
                    p1.tconst = p2.tconst
            JOIN title_basics m
                ON
                    p1.tconst = m.tconst
            JOIN name_basics a1
                ON
                    p1.nconst = a1.nconst
            JOIN name_basics a2
                ON
                    p2.nconst = a2.nconst
            WHERE
                a1.primaryName = :actor1
                AND
                    a2.primaryName = :actor2
                AND
                    p1.category = 'actor'
                AND
                    p2.category = 'actor'";
```

(Version test changement possible)

2. Fonction “ActeurEnCommun”:

L’objectif de cette fonction est de trouver des acteurs qui ont joué dans deux films spécifiés. Elle prend en paramètres un tableau “$data” contenant les titres de deux films (“primaryTitle1” et “primaryTitle2”).
Comme dans “FilmEnCommun”, la fonction commence par vérifier la présence des titres des films. Si l'un des titres est absent, elle renvoie un message d'erreur.
La requête SQL est construite un peu de la même façon que pour la fonction pour trouver les noms des acteurs qui ont joué dans les deux films spécifiés.
Après avoir préparé la requête, les titres des films sont reliés en tant que paramètres, et la requête est exécutée.
Les résultats sont retournés sous forme d'un tableau associatif.

Dans les deux fonctions, l'utilisation de `htmlspecialchars` et `PDO::PARAM_STR` lors de la liaison des paramètres aide à prévenir les injections SQL et assure la sécurité des données. Ces fonctions permettent une recherche efficace et sûre dans notre base de données de films et d'acteurs dans le cadre de notre projet.

Voici à quoi doit ressembler la requête :

```
        $sql = "SELECT DISTINCT
            a.primaryName
        FROM
```

```
            title_principals p1
        JOIN
            title_principals p2
                ON
                    p1.nconst = p2.nconst
        JOIN
            title_basics m1
                ON
                    p1.tconst = m1.tconst
        JOIN
            title_basics m2
                ON
                    p2.tconst = m2.tconst
        JOIN
            name_basics a
                ON
                    p1.nconst = a.nconst

        WHERE
            m1.primaryTitle = :movie1
            AND
            m2.primaryTitle = :movie2
            AND
            p1.category = 'actor'
            AND
            p2.category = 'actor'";
```

(Version test changement possible)

3. Algorithme Rapprochement de films

“Trouver la plus courte chaîne entre deux éléments dans le graphe “a joué dans”” :

Description de l’algorithme : On veut pouvoir relier deux films ou personnes par une chaîne de personnes et films la plus courte possible suivant la relation “X a participé au film Y”. Par exemple, on peut passer de John Travolta à Harrison Ford en passant par Pulp Fiction, Rosanna Arquette, Le Grand Bleu, Jean Reno, Léon, Natalie Portman, Star Wars III : Revenge of the Sith, Ewan McGregor, Star Wars VII : The Force Awakens, Harrison Ford. Est-ce qu’il y a plus court ?

Description du code et explication simplifiée:
Pour cette description nous allons aborder le cas d’un rapprochement entre 2 acteurs, à noter que l’algorithme de rapprochement entre deux acteurs ou deux films sont similaires dans la globalité la différence se traduit par :

Pour le rapprochement entre deux acteurs, nous allons créer un graphe où les nœuds seront les films et les acteurs seront les branches.
Pour le rapprochement entre deux films, nous allons créer un graphe où les nœuds seront les acteurs et les films seront les branches.

Concernant la procédure, pour chaque rapprochement, nous allons créer trois processus (à l'aide du module threading en python) P1 , P2 et P3 où P1 et P2 créerons un graph à partir de leur point de départ et P3 se chargera de la validation de traitement.

<u>Processus P1 et P2 :</u>
Les deux processus P1 et P2 possèdent un algorithme similaire qui a pour but de retourner à chaque étape les noeuds suivis des branches sous la forme d’un dictionnaire. A l’initialisation, les processus vont récupérer les noeuds à partir de leurs points de départ (soit acteur soit film) c’est à dire pour un rapprochement d’acteur nous allons lister les oeuvres auquels la personne à participé, et pour un rapprochement de films nous allons lister les personnes qui ont participé à cette oeuvre.
Voici un exemple de code de l’initialisation de départ pour le rapprochement entre deux acteurs :

```
    #Initialisation
    sql = "SELECT tconst FROM title_principals WHERE nconst =
%(nconst_debut)s;"
    value = {"nconst_debut": nconst_debut}
    cur.execute(sql, value)
```

```
    tabs_tconst_noeud = [noeud[0] for noeud in cur.fetchall()]
```

Ensuite, ils rentrent dans une boucle tant que le traitement n'est pas validé (validation gérer par le processus P3), ils vont récupérer les branches à partir des noeuds actuels (récupérés lors de l'initialisation) et ils envoient sous la forme d'un dictionnaire leurs réponses à P3 dans lesquels les clés seront les noeuds et les valeurs seront un tableau de branches et se mettent en attente de la réponse de P3. Si la validation est ok, arrêt du traitement sinon, il réinitialise le dictionnaire pour récupérer les nœuds à partir des branches actuelles et ils réitèrent ce fonctionnement jusqu'à validation.

Voici un exemple de résultat pour chaque itération des processus P1,P2 :

```
#P1: {"type": "debut", noeud1 : ['Acteur','Acteur'], noeud2 :
['Acteur']}
#P2: {"type": "fin", noeud1 : ['Acteur'], noeud2 : ['Acteur']}
```

Trois conditions permettent de rendre les résultats de P1 et P2 lisible pour les traitement de P3, une branche ne doit plus apparaître pour plus de 2 itérations, la branche ne doit pas être égal au point de départ et les noeuds n'apparaissent qu'une seule fois dans tout le traitement (voir exemple pour résultat ci-dessous).

Processus 3 :
Le processus P3 récupère les résultats de P1 et P2 au fur est à mesure de l'exécution et stocks ces résultats dans les dictionnaires finaux (dic_debut et dic_fin). Le processus teste si il y a un noeud en commun ou une branche en commun, si c'est le cas il appelle la fonction chemin qui prend les deux dictionnaires dic_debut et dic_fin pour retracer le chemin et arrête la recherche, si aucune condition est avérée alors la recherche continue.

```
322         with condition:
323             clee = clee_commun(dic_debut, dic_fin, i)
324             result = acteur_commun(dic_debut, dic_fin, i)
325             if clee:
326                 Chemin_f = chemin_test(dic_debut, dic_fin, {}, clee)
327                 traitement_accepter = True
328                 condition.notify_all()
329                 break
330             elif result:
331                 Chemin_f = chemin_test(dic_debut, dic_fin, result, [])
332                 traitement_accepter = True
333                 condition.notify_all()
334                 break
335             else :
336                 traitement_accepter = False
337                 condition.notify_all()
```

Voici un exemple des deux dictionnaires créer par P3 :

- Cas où noeud en commun (t30) traitement arréter à l'itération 3 car 't30' est présent dans les 2 dictionnaires :

```
```

```
dic_debut = {
    0 : {'t01' : ['n01'], 't02' : ['n03']},
    1 : {'t07' : ['n08','n01']},
    2 : {'t12' : ['n21','n08']},
    3 : {'t30' : ['n21','n00','n22']}

}


dic_fin = {
    0 : {'t03' : ['n04']},
    1 : {'t10' : ['n20','n04']},
    2 : {'t20' : ['n22','n20']},
    3 : {'t30' : ['n21','n00','n22']}
```

- Cas branche en commun (n9) traitement arréter à l'itération 1 car 'n9' est présent dans les 2 dictionnaires :

```
dic_debut_test = {
    0 : {'t20': ['n2','n1']},
    1 : { 't4' : ['n2', 'n9', 'n10']}
}

dic_fin_test = {
    0: {'t24' : ['n5', 'n7']},
    1: {'t3' : ['n12', 'n5', 'n9']}
}
```

Si le programme dépasse un certain nombre d'itérations définies au préalable, alors le programme sera stoppé pour éviter les cas de boucle infinies.

Chemin :

Concernant la construction du chemin gérér par la fonction chemin. Le principe est de séparer le traitement en 2 pour créer 2 chemin, un chemin du début jusqu'au point de connexion et du point de connexion à la fin. Le but est de se positionner à la valeur de la dernière itération du programme (soit les clés des dictionnaires début et fin) et de comparer si une valeur de la clé du sous dictionnaire est présente dans la clé du sous dictionnaire à l'itération n-1.

Par exemple pour le cas du noeud en commun (t30) le résultat sera :

Les données sont sous format tuple dans lequel l'indice 0 (gauche) correspond au chemin du début au point de connexion et à l'indice 1 (droite) correspond au chemin du point de connexion à la fin. De plus ceci n'est qu'une version test la mise en forme pourra changer comme l'ajout des extrémités et des jointures (film ou acteur) entre chaque branches.
Pour créer ces chemins du début au point de connexion nous utiliserons le dictionnaire dic_debut créé au fur et à mesure par P3 et du point de connexion à la fin nous utiliserons dic_fin créer aussi par P3.

API :

Etant donné que ce code fait partie d'un environnement extérieur à notre site web, nous allons mettre en place une API qui permet de créer la communication entre le back-end de notre site web et l'algorithme rapprochement, par l'intermédiaire de requêtes HTTP contenant des flux de données sous format Json.