Compte Rendu de Projet

Sommaire :

1. Les choix effectués durant l'implémentation de ce projet

Durant la mise en place du programme permettant de détecter un potentiel plagiat entre deux fichiers contenant un code source en langage C nous avons dû faire un certain nombre de choix afin d'obtenir un programme fonctionnel et efficace.

Tout d'abord pour la partie dite de traduction nous avons décidé de traduire les fichiers source caractère par caractère, cela nous a permis de traiter au mieux les différents cas particuliers et de rendre notre programme plus précis dès les premières étapes. Néanmoins cela a pu rendre plus complexe le cas des lignes vides par exemple. C'est en partie pour cela que nous avons choisi de stocker la version traduite du code source dans un fichier texte. En plus de faciliter le traitement des lignes vides sans risquer des problèmes dus à la mémoire, ces fichiers permettent de visualiser au mieux la traduction et les potentielles erreurs qu'elle a pu générer lorsque nous l'avons développée. Ces fichiers qui prennent une place conséquente dans la mémoire sont pour la plupart supprimés à la fin de l'exécution, on ne conserve que la version finale où toute la traduction a été effectuée. Toujours dans un objectif de précision nous avons fait le choix de supprimer les lignes vides ainsi que les lignes ne contenant que le caractère «{ » ou « } ». En effet ces lignes bien qu'elles ne génèrent pas de valeurs incorrectes sont communes à bon nombre de programmes ainsi elles risquent de réduire la distance entre deux fichiers par simple coïncidence. Enfin, nous avons remarqué que le plagiat des codes commence le plus souvent par le changement de nom des variables, ainsi le caractère « _ » a été considéré au même titre que les caractères alphanumériques car il peut être utilisé lors du nommage des variables ainsi « var1 » et « var_1 » sont toutes deux traduites « w ».

Pour calculer la similarité entre chaque segment nous avons choisi de stocker les digrammes dans des listes chainées, c'est une structure que nous maitrisons grâce aux travaux et exercices réalisés lors du semestre et qui a facilité le calcul de la distance de Dice par la suite.

Enfin pour la partie concernant le calcul de la matrice finales via une opération de convolution nous avons testé plusieurs solutions. Nous avons d'abord recopié les valeurs des extrémités pour complètement remplir la matrice F, qui n'avait pas été remplie au début du à la formule de calcul de

ses coefficients on obtenait la sous-matrice F1 aux dimensions [longueur.fichier1-2][longueur.fichier2-2]. Cette solution n'apportait pas satisfaction car les fichiers identiques ne donnaient pas un résultat nul. Nous avons alors pris le parti d'ajouter des zéros sur la diagonale et des 1 sur le reste des extrémités. Le résultat est alors correct pour des fichiers identiques et n'est pas significativement modifié dans les autres cas, les valeurs ajoutées étant négligeables face aux reste des coefficients.

Pour les fonctionnalités dites bonus nous avons pu implémenter celle permettant de traduire les mots-clés du langage par un « m » et non un « w ». Par la suite nous avons testé plusieurs valeurs pour la longueur de n-grammes mais la valeur 2 donnant plus de satisfaction elle a été conservée. Enfin, nous avons essayé d'afficher des matrices colorées mais par manque d'expérience avec ce genre de fichier les résultats n'étaient pas toujours lisibles ou parfois même incohérents et nous n'avons pas pu utilisé cette fonctionnalité. Nous avons également implémenté la distance de Levenshtein il s'agit du code qui a été proposé dans la page Wikipédia mentionnée. Et après avoir un peu cherché en ligne, on a trouvé que pour normaliser la distance obtenue on devrait diviser par le maximum du la longueur des chaînes qui vont être comparées. Au début du programme, l'utilisateur peut sélectionner la distance qu'il souhaite utiliser.

2. Algorithmes et complexités

Dans cette partie nous expliciterons le fonctionnement de certains algorithmes ainsi que leur complexité.

Pour les deux algorithmes, « create_digraph » et « create_segment » il s'agit de la création de deux structures contenant des chaînes de caractères avec une taille N comme composantes et de remplir chaque chaîne en la recopiant de l'argument de la fonction, sans oublier le '\0' à la fin. la complexité est: θ(2) pour la création des diagrammes et θ(longueur de la ligne qu'on traite dans le fichier traduit) pour la création des segments.

Pour l'ajout des segments et des digrammes dans les listes chainées ces deux fonctions prennent des listes chaînées en argument et test si jamais elle est vide, dans ce cas les digrammes ou segments créés par les argument seront les premiers éléments de la liste. Sinon, la fonction parcourt la liste tout entière et pour retrouver le dernier élément et l'ajouter à la suite de celui-ci grâce à la composante "next" des structures. la complexité est: θ(longueur des listes).

Le découpage en digrammes d'un segment suit le principe suivant : on parcourt toute la chaîne de caractères passée en argument et remplit une chaîne de caractères créée « d[2] » pour obtenir tous les digrammes possibles. Ensuite on ajoute chaque digramme à la liste chaînée passée en

argument grâce à (Add_Digraph). Sa complexité est alors : θ(longueur de la chaîne passée en argument -1).

La fonction permettant de comparer deux digrammes fonctionne de la manière suivante : cette fonction prend en argument deux chaînes de caractères et crée deux listes de leurs digrammes grâce aux fonctions évoquées plus tôt, elle parcourt ensuite la liste la plus courte, pour ne pas compter un même digramme deux fois, et pour chaque digramme de cette liste la plus courte, elle parcourt la deuxième liste en comparant les digrammes deux à deux : si jamais ils sont identiques on incrémente notre compteur i et on fait un break, puisqu'un même diagramme peut se répéter deux fois dans la liste la plus longueur, pour ne pas compter un même digramme commun plusieurs fois ce qui a pu créer des valeurs incohérentes. Sa complexité est O(m -1) où m=max(longueur de la première chaîne passée en argument ; longueur de la deuxième chaîne passée en argument).

Enfin pour l'algorithme glouton on recherche chaque fois le plus petit élément de la matrice et on remplace toutes les valeurs de la « croix » formée par la colonne et la ligne du plus petit coefficient. Pour cela on a recopiée cette dernière dans une deuxième matrice M1 pour pouvoir changer les valeurs sans toucher à la matrice originale. Dans chaque boucle, on recherche le minimum de M1, une fois trouvé on le copie dans C et on sauvegarde sa ligne et sa colonne. Toutes les valeurs sur la "croix" créé par la ligne et colonne du plus petit élément seront ensuite remplacées par des 10 dans la matrice M1: pour ne pas être détecté comme plus petit élément dans la boucle qui suit (on rappelle que la distance de Dice est comprise entre 0 et 1). On reprend ce raisonnement, et notre boucle while s'arrête dès que le plus petit élément vaut 10, on s'assure ainsi que tous les plus petits éléments ont été recopiés dans la matrice C, qui a elle aussi été remplie par des 1 sur la "croix".

3. Difficultés rencontrées durant la mise en place du projet

Pendant l'élaboration de notre programme et des différentes fonctions nous avons dû faire face à plusieurs difficultés. Ces difficultés peuvent de classer en deux catégories, des obstacles techniques dus à la difficulté d'implémentation ou aux nombres importants de cas particuliers et des difficultés de gérer un code d'une longueur conséquente par rapport à nos habitudes ainsi qu'un travail en binôme différents du travail que l'on a pu réaliser au long de notre semestre.

Lors de la traduction dans la première étape, pour le traitement des chaînes d'échappement de type '\'' dans le code, nous avons alors laissé leurs traductions en '' . En effet ce sont des cas très particuliers et nous avons remarqué que cela ne change pas grande chose pour le résultat final, comme il y a uniformité du processus de traduction dans le

programme, on pourrait alors quand même détecter si deux chaînes d'échappement ont été plagiées.

Le calcul de la distance de Dice ainsi que certains cas particuliers qui y sont liés ont été le principal problème et une importante source de valeurs incohérentes. Nous avons rencontré des "nan"(not a number) lors du calcul de distance pour de longs fichiers, il y avait alors plusieurs sources d'erreurs possibles comme les espaces ou encore les lignes vides. Grâce à plusieurs expériences nous avons déterminé que le problème venait des espaces et nous avons alors pu perfectionner notre fonction de traduction. Ensuite, il a fallu traiter le cas de valeurs négatives résolu grâce à l'ajout d'un test sur la longueurs des chaînes, l'ordre des while diffère selon la longueur des chaînes passées en argument. Enfin, la présence d'un même digramme plusieurs fois dans un segment faussait certains résultats nous avons donc ajouté un « if » et un « break ».

Dans l'avant-dernier bonus: nous avons créé une autre liste chaînée à partir du fichier "inter" pour sauvegarder les lignes originales. Nous avons ensuite parcouru cette nouvelle liste pour calculer la distance entre chaque deux segments des deux fichiers, sans prendre en compte les lignes vides et les lignes qui contiennent seulement des accolades, mais nous avons trouvé un peu plus compliqué de traiter le cas des segments qui commencent par des accolades: nous avons alors abandonné ce cas pour ne pas avoir beaucoup de tests "ifs" imbriqués.

Au-delà des difficultés techniques, nous avons parfois eu du mal à interpréter nos résultats par manque de résultats théoriques les distances de Dice et les distances entre les fichiers ne pouvant pas toujours être calculées à la main. De plus, notre manque d'expérience avec des codes aussi longs rendait parfois difficiles l'identification et la correction d'erreurs. Enfin le travail en binôme peut parfois être difficile notamment lors de la lecture et de la compréhension du code de l'autre personne.

4. Répartition et méthodes de travail au sein du binôme

Nous avons décidé pour commencer de se répartir le travail sur les trois premières parties ainsi Hania a réalisé le découpage des segments et du calcul de la distance de Dice pendant que Emile a implémenté la fonction de traduction du code. Pour la suite du projet nous avions eu plus l'occasion de travailler ensemble et nous avons pu toutes les semaines se retrouver afin de réfléchir à la suite du projet, la correction d'éventuels bugs ainsi que la réalisation de tests et la préparation du rendu du projet.