TP 2
COMPARAISON DE PLUSIEURS TEXTES

1. Introduction

Pour le deuxième TP, vous devez construire un logiciel C++ qui compare plusieurs textes. Un corpus de texte est fourni et l'utilisateur veut trouver les textes pareils, ou différents. Cette recherche doit tenir compte du nombre d'occurrences d'un mot et de l'importance qu'a ce mot. La prochaine section va décrire les entrées du logiciel. La troisième section contient une description des calculs que le logiciel doit faire. La quatrième section décrit les sorties. Finalement, la cinquième section contient les directives.

2. Description des entrées

Corpus

Une première entrée utilisée par votre logiciel est un corpus de texte. Ce corpus sera divisé en plusieurs fichiers contenant les textes et un autre fichier décrivant le corpus. Votre programme va prendre le nom du fichier contenant la description du corpus sur la ligne de commande. Pour l'exemple sur Moodle, le fichier `listeDocument.xml` contient la description du corpus et les fichiers `janvier1930.xml` à `avril1930.xml` contiennent les textes. Pour le reste de la description, nous utiliserons D_i pour désigner le $i^{ième}$ textes. Tous ces fichiers sont en format xml. Le code permettant la lecture de ces fichiers est déjà donné. Le code donné est composé de 18 fichiers :

- Des classes pour décrire un document XML :
    - AttributNonDefinieXML (hpp et cpp) : une exception.
    - DocumentXML (hpp et cpp) : la base du document XML. Une structure d'arbre généralisé est utilisée.
    - NoeudXML (hpp et cpp) : les nœuds qui composent l'arbre généralisé.
        - ElementXML (hpp et cpp) : un type de nœud représentant les éléments (tag) XML.
        - TexteXML (hpp et cpp) : un type de nœud représentant le texte entre les éléments XML.

  Cette structure est spécialisée pour le TP et n'est pas complète pour représenter des documents XML génériques. Vous n'avez pas à vous occuper de ce code pour le TP. Il est appelé au début pour lire les fichiers et construire la structure de base.

- Des classes pour représenter la structure de base.
    - Histoire (hpp et cpp) : décris une histoire (dans le code, le mot « **histoire** » est utilisé plutôt que le mot « **texte** » pour décrire les textes). Cette structure contient le titre de l'histoire et les phrases qui composent l'histoire. Un itérateur permet de parcourir les phrases de l'histoire.
    - Phrase (hpp et cpp) : décris une phrase de l'histoire. Elle contient le texte original et les mots qui composent la phrase. Un itérateur permet de parcourir les mots (string) de la phrase. Les mots sont déjà en minuscule.

  Ces fichiers contiennent la structure que vous devrez consulter pour faire vos calculs. **Vous pouvez ajouter du code dans ces fichiers.**
- Lecteur (hpp et cpp) : contient le lecteur de fichier XML pour construire la structure XML contenant l'information d'un fichier. Un automate simple est utilisé comme algorithme de base avec des pointeurs de fonction (callback).
    - Automate : vous allez peut-être voir cela dans vos études, cela dépend des cours que vous suivez.
    - Pointeurs de fonction : habituellement montré dans le cours inf3135.

  REMARQUE : vous n'avez pas à comprendre ou modifier ce code pour le TP.
- TP2.cpp : contient la fonction `main`. En ce moment, la fonction `main` contient du code montrant comment utiliser les structures de base (Histoire et Phrase). Pour votre projet, vous devez, entre autres, remplacer ce code par votre code. **Vous pouvez ajouter des fonctions.**
    - Ce fichier contient aussi la fonction `lireDocuments`. Cette fonction lit le corpus et construit la structure de base pour le logiciel. Elle est appelée par la première ligne du programme principal. Elle retourne un vecteur d'histoires. Votre code doit utiliser ce vecteur pour faire l'analyse. Cela vous demande d'être capable d'utiliser les méthodes des classes Histoire et Phrase.
- makefile : pour compiler le tous.
    - make : compile le projet, l'exécutable portera le nom de TP2.
    - make clean : nettoie le répertoire.
    - make archive : construit une archive pour la remise.

  Vous allez devoir modifier ce fichier pour ajouter votre classe d'arbre AVL.

3. Calcule

Premier calcul

Pour chaque texte $\boldsymbol{D_i}$ vous devrez calculer le $\mathbf{tf}(\boldsymbol{D_i}, \boldsymbol{m})$ de chaque mot $\boldsymbol{m}$ contenu dans le texte. Il y aura donc une valeur $\mathbf{tf}$ pour chaque mot dans chaque texte. Pour ce calcul vous devez utiliser un arbre AVL qui contiendra les mots comme clef et le nombre d'occurrences de chaque mot comme définition (donc une structure associative). Il y aura donc un arbre AVL par texte. Le $\mathbf{tf}$ (*terme frequency*) d'un mot dans un texte est simplement le nombre de fois que ce mot apparait dans le même texte. Un mot aura donc un $\mathbf{tf}$ différent pour chaque texte.

Aussi, pour chaque mot, vous devrez calculer son $\mathbf{idf}(\boldsymbol{m})$. Il y a un seul $\mathbf{idf}$ pour tous les textes. L' $\mathbf{idf}$ (*inverse document frequency*) est inversement proportionnelle au logarithme du nombre de textes qui contient ce mot. Par exemple, pour un mot donné, s'il apparait dans 3 textes sur 26, alors il aura un $\mathbf{idf}$ de $\mathbf{log_2}(\mathbf{26/3})$. Donc, si vous avez un mot $\boldsymbol{m}$ qui apparait dans $\boldsymbol{x}$ des $\boldsymbol{N}$ textes, vous avez $\mathbf{idf}(\boldsymbol{m}) = \mathbf{log_2}(\boldsymbol{N/x})$. Vous devez aussi utiliser un arbre AVL pour placer les résultats de ce calcul.

Deuxième calcul

Maintenant que vous avez la fréquence de chaque mot (**tf**) pour chaque texte et que vous avez l' $\mathbf{idf}$ pour l'ensemble des textes, il reste à utiliser ces valeurs pour calculer la similarité des textes. La similarité entre deux textes ($\boldsymbol{D_i}$ et $\boldsymbol{D_j}$) est donnée par la formule suivante.

$$\boldsymbol{Sim}(\boldsymbol{D_i}, \boldsymbol{D_j}) = \frac{\boldsymbol{p}(\boldsymbol{D_i}, \boldsymbol{D_j})}{\boldsymbol{n}(\boldsymbol{D_i}) \cdot \boldsymbol{n}(\boldsymbol{D_j})}$$

Pour faire ce calcul, il faut utiliser les formules suivantes.

$$\boldsymbol{s}(\boldsymbol{m_k}, \boldsymbol{D_i}) = \boldsymbol{tf}(\boldsymbol{D_i}, \boldsymbol{m_k}) \cdot \boldsymbol{idf}(\boldsymbol{m_k})$$

$$\boldsymbol{p}(\boldsymbol{D_i}, \boldsymbol{D_j}) = \sum_{\boldsymbol{m_k} \in (\boldsymbol{D_i} \cup \boldsymbol{D_j})} \boldsymbol{s}(\boldsymbol{m_k}, \boldsymbol{D_i}) \cdot \boldsymbol{s}(\boldsymbol{m_k}, \boldsymbol{D_j})$$

Dans la formule pour $\boldsymbol{p}$, $\boldsymbol{m_k}$ prends, tour à tour, tous les mots possibles. Remarquez qu'il est seulement nécessaire de prendre les mots qui sont présents dans l'**union** des deux textes comparés.

$$\boldsymbol{n}(\boldsymbol{D_i}) = \sqrt{\sum_{\boldsymbol{m_k} \in \boldsymbol{D_i}} \boldsymbol{s}(\boldsymbol{m_k}, \boldsymbol{D_i})}$$

4. Production des sorties

Votre logiciel doit placer les résultats dans un fichier nommé `res.txt`. Ce fichier contiendra les résultats de la comparaison entre chaque texte du corpus($\boldsymbol{Sim}(\boldsymbol{D_i}, \boldsymbol{D_j})$). Ces valeurs seront placées comme suit.

"Titre texte1" "Titre texte 2" … "Titre texte n"

"Titre texte 1" $\boldsymbol{Sim}(\boldsymbol{D_1}, \boldsymbol{D_1})$ $\boldsymbol{Sim}(\boldsymbol{D_1}, \boldsymbol{D_2})$ $\boldsymbol{Sim}(\boldsymbol{D_1}, \boldsymbol{D_n})$

…

"Titre texte n" $\boldsymbol{Sim}(\boldsymbol{D_n}, \boldsymbol{D_1})$ $\boldsymbol{Sim}(\boldsymbol{D_n}, \boldsymbol{D_2})$ $\boldsymbol{Sim}(\boldsymbol{D_n}, \boldsymbol{D_n})$

Dans ce contexte, "Titre texte1" est le titre du premier texte entre guillemets et $\boldsymbol{Sim}(\boldsymbol{D_1}, \boldsymbol{D_2})$ est la différence entre les textes 1 et 2, qui sont une valeur de type double, non arrondi.

5. Directive

1. Le TP est à faire seul ou en équipe de deux (maximum).
2. Code :
    a. Pas de `goto`.
    b. Pour une fonction :
        i. Additionnez le nombre de `if`, `for`, `while`, `switch` et `try`. Ce nombre ne doit pas dépasser 9.
        ii. Un seul `return`.
    c. Vous DEVEZ avoir une classe pour les arbres AVL et l'utiliser pour le calcul des tf, idf si vous voulez pouvoir avoir 20/20.
    d. Vous POUVEZ utiliser la classe `map` de la stl c++ pour replacer les arbres AVL, dans ce cas, votre note ne dépassera pas 16/20.
3. Commentaires
    - Commentez l'entête de chaque classe et fonction.
        - Vos fonctions et classes, pas besoin de commenter mon code.
    - Une ligne contient soit un commentaire, soit du code, pas les deux.
    - Utilisez des noms d'identificateur significatif.
    - Une ligne de commentaire ne devrait pas dépasser 80 caractères. Continuez sur la ligne suivante au besoin.
    - Plus en détail
        - La première ligne d'un commentaire doit contenir une description courte (1 phrase) de la fonction ou la classe.
            - Courte.
            - Complète.
            - Commencez la description avec un verbe.
            - Assurez-vous de ne pas simplement répéter le nom de la méthode, donnez plus d'information.
        - Ensuite, au besoin, une description détaillée de la fonction ou classe va suivre.
            - Indépendant du code. Les commentaires d'entêtes décrivent ce que la fonction fait, ils ne décrivent pas comment c'est fait.
            - Si vous avez besoin de mentionner l'objet courant, utilisez le mot 'this'.
        - Ensuite, placez une ligne vide.
        - Décrivez les paramètres de la fonction.
        - Écrivez les commentaires à la troisième personne, EN FRANÇAIS.

Remise

Remettre le TP par l'entremise de Moodle. Vous devez remettre un fichier compressé du projet en utilisant exactement la méthode suivante sur le serveur Java :

Dans le répertoire de votre projet, où seulement les fichiers `.hpp`, `.cpp` et le `makefile` sont présent,

- `make archive`

Cela va produire un fichier `TP2.tar.gz` que vous allez remettre par l'entremise de Moodle.

Le TP est à remettre avant le 21 juillet 23:55.

Évaluation

- Fonctionnalité (14 pts) : Votre TP doit compiler sans erreur (il peut y avoir des warnings). J'utilise la ligne de compilation suivante sur les serveurs Java de l'université : `make.`
    - Des tests seront donnés.
- Structures (3 pts) : Il faut avoir **plusieurs** fonctions. Construisez un code bien structuré. (Voir les directives.)
- Lisibilité (3 pts) : commentaire, indentation et noms d'identificateur significatif. (Voir les directives.)