TP 2
PARTITIONNEMENT HIERARCHIQUE DE DONNÉS

Introduction

Vous allez construire un logiciel qui construit un résumé d'un texte. Pour résoudre ce problème, le texte sera divisé en phrase et votre logiciel devra extraire certaines phrases pour construire le résumé. La sélection des phrases pour le résumé va utiliser un algorithme de partitionnement (clustering) hiérarchique des phrases : les phrases seront regroupées en sous-ensembles de phrases similaires. L'algorithme de partitionnement hiérarchique est une forme généralisée de l'algorithme de Kruskal pour la construction de l'arbre sous-tendant minimum.

Description des entrées

Corpus

Nous allons utiliser le même corpus que pour le tp2. Le code lisant le corpus vous est donné. Ce code va diviser les textes (histoires) en `Phrase`, une nouvelle classe qui vous sera aussi données. Cette classe va contenir une méthode qui retourne la similarité entre deux phrases et une autre méthode qui retournera la distance entre deux phrases. Deux phrases sont distantes une de l'autre si elles sont peut similaire, inversement, elles sont rapprochées si elles sont similaires. Pour ce tp, vous allez utiliser la méthode de distance, vous n'utilisez pas celle de similarité, elle est utilisée par la méthode de distance. La classe `Histoire` va contenir un itérateur parcourant les `Phrase`s de l'histoire.

Ligne de commande

Votre logiciel va accepter en entrées, sur la ligne de commande, des sémaphores indiquant son comportement. Un seul des sémaphores peut être utilisé. Votre logiciel doit déclarer une erreur si plus d'un sémaphore est utilisé ou si le sémaphore est incorrect.

- -c : complete-linkage clustering, utilisation du maximum des distances.
- -s : single-linkage clustering, utilisation du minimum des distances.
- -a : average-linkage clustering, utilisation de la moyenne des distances.

Par défaut, s'il n'y a pas de sémaphore pour le type d'algorithme, considérez que l'utilisateur à utiliser -s. Aussi, la ligne de commande doit contenir le titre de l'histoire à résumer et le sémaphore -n suivit d'un nombre. Ce nombre doit être une valeur entière plus grande que 0, elle indique le nombre de partitions recherché. Par défaut, utilisez la valeur 5. Si le titre indique une histoire en plusieurs parties, vous devez chercher toutes les parties de l'histoire et construire le résumé de l'histoire complète.

Calcule

Pour construire les partitions C_k d'un texte T, vous devez construire un graphe $G =< S, A, w >$ représentant les données. Chaque nœud du graphe représente une partition C_k. Une partition est un ensemble de phrases $p_k \in T$. Initialement, chaque partition contient une seule phrase $C_k = \{p_k\}$. Il y aura donc autant de partitions qu'il y a de phrase. Un arc $a \in A$ est placé entre chaque partition. Cet arc représente la **distance** entre les partitions. Initialement, puisqu'il y a une seule phrase par partition, les arcs représentent la **distance** entre les phrases contenues dans les partitions. Par exemple, pour l'arc $a = (C_k, C_h)$ où $C_k = \{p_k\}$ et $C_h = \{p_h\}$, la distance $w(C_k, C_h)$ sur cet arc sera la distance entre les phrases p_k et p_h.

Lorsque le graphe G initial a été construit, l'algorithme va exécuter un travail similaire à celui de Kruskal, il va choisir l'arc ayant la distance le plus petit et unir les deux partitions aux extrémités de cet arc. L'union des deux partitions va placer les nœuds de chacune des partitions ensemble. Donc les deux nœuds vont être fusionnés en un seul. Cela va modifier le graphe sur lequel votre algorithme travaille. Cette fusion entraine la fusion des arcs vers le nœud fusionné. Soit trois nœuds : C_k, C_i et C_j et les arcs (C_k, C_i) ayant une distance $w(C_k, C_i)$ et (C_k, C_j) ayant une distance $w(C_k, C_j)$. Lors de la fusion des nœuds C_i et C_j, le nouveau nœud $C_h = C_i \cup C_j$ sera ajouté au graphe et les nœuds C_i et C_j seront supprimés du graphe. Les arcs (C_k, C_i) et (C_k, C_j) seront remplacé par un arc (C_k, C_h) ayant une distance $w(C_k, C_h)$. Le calcul de la nouvelle distance de l'arc dépend du sémaphore qui vous a été donné.

- -c : $w(C_k, C_h) = \max(\, w(C_k, C_i), w(C_k, C_j))$
- -s : $w(C_k, C_h) = \min(\, w(C_k, C_i), w(C_k, C_j))$
- -a : $w(C_k, C_h) = \frac{w(C_k, C_i) \times |C_i| + w(C_k, C_j) \times |C_j|}{|C_i| + |C_j|}$

L'union est terminée lorsque cette opération a été répétée pour tous les nœuds du graphe. Votre algorithme va continuer à unir des nœuds jusqu'à ce que le nombre n de partitions recherchées soit atteint. Lorsque ce nombre est atteint, il ne reste plus qu'à trouver une phrase par partition, cette phrase va représenter la partition.

Pour chaque partition C_i, vous devez trouver une phrase représentante. Soit $p_j \in C_i$ les phrases de la partition. Pour chacune des phrases p_j, vous allez calculer la distance moyenne m_j de ces phrases vers les autres phrases de la partition.

$$m_j = \sum_{p_x \in C_i \setminus \{p_j\}} distance(\, p_x, p_j) / (\, |C_i| - 1\,)$$

La phrase ayant la moyenne la plus petite sera choisie pour représenter la partition.

Voici l'algorithme complet :

Partitionnement(T, n)
  Construire le graphe initial $G =< S, A, w >$:
    S = { {p} | $p \in T$ }
    A = { (C_i, C_j) | $C_i \in S, C_j \in S, i \neq j$ }
    $w(C_i, C_j)$ = distance entre p_i et p_j où $C_i = p_i$ et $C_i = p_i$.
  Calculer le partitionnement :
    Tant que $|S| > n$
      (C_i, C_j) = arc ayant la distance $w(C_i, C_j)$ minimum.
      C_h = construire un nouveau nœud où $C_h = C_i \cup C_j$
      Enlever C_i et C_j de l'ensemble S des nœuds.
      $\forall C_k \in S$, soit $(C_k, C_i) \in A$ et $(C_k, C_j) \in A$:
        Enlever (C_k, C_i) de A.
        Enlever (C_k, C_j) de A.
        Ajouter l'arc (C_k, C_h) à A.
        Calculer la nouvelle distance w(C_k, C_h) selon la formule demandée (-c, -s, -a).
      Ajouter le nœud C_h à l'ensemble S des nœuds.
  Trouver une phrase représentante par partition :
    $\forall C_i \in S$,
      $\forall p_j \in C_i$,
        $m_j = \sum_{p_x \in C_i \setminus \{p_j\}} distance(p_x, p_j)/(|C_i| - 1)$
      Le m_j le plus petit indique la phrase p_j qui devra être affiché pour cette partition.

Production des sorties

Afficher simplement les phrases représentant les partitions, l'ordre n'a pas d'importance. Une phrase par ligne.

Directive

1. Le tp est à faire seul ou en équipe de deux (maximum).
2. Commentaire :
   a. Commentez l'entête de chaque fonction. Ces commentaires doivent contenir la description de la fonction et le rôle de ces paramètres.
   b. Une ligne contient soit un commentaire, soit du code, pas les deux.
   c. Utilisez des noms d'identificateur significatif.
   d. Utilisez le français.
3. Code :
   a. Pas de `goto`, `continue`.
   b. Les `break` ne peuvent apparaître que dans les `switch`.
   c. Un seul `return` par méthode.
4. Indentez votre code. Assurez-vous que l'indentation est faite avec des espaces.

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 Malt :

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

- `tar cvf tp3.tar *`
- `gzip tp3.tar`

Cela va produire un fichier `tp3.tar.gz` que vous allez remettre par l'entremise de Moodle.

Le tp est à remettre avant le 30 avril 23:59.

Évaluation

- Fonctionnalité (10 pts) : Votre tp doit compiler sans erreur (il peut y avoir des warnings). J'utilise la ligne de compilation suivante : `g++ *.cpp -lm -std=c++11`
- Structures (2 pt) : Il faut avoir **plusieurs** fonctions. Construisez un code bien structuré. Vous pouvez utiliser la librairie **stl**.
- Lisibilité (2 pts) : commentaire, indentation et noms d'identificateur significatif.