TRAVAUX PRATIQUES XXII

Distance d'édition

Ce TP se fait en C.

A Distance d'édition

A.1 Opérations

Dans un mot, une **addition en position i** consiste à insérer une lettre de sorte à ce qu'elle soit en position i.

Exemple. Si on insère un A en position 3 dans GARD, on obtient GARAD

Réciproquement, une **délétion en position i** consiste à supprimer la lettre d'indice i.

Exemple. Si on supprime en position 2 dans GARAD, on obtient GAAD

Enfin, une **substitution en position i** consiste à remplacer la lettre d'indice i par une autre

Exemple. Si on subtitute dans GAAD en position 0 par L, on obtient LAAD

A.2 Distance

La distance d'édition (ou distance de Levenshtein) entre deux mots `u` et `v` est le nombre minimal d'opérations (additions/délétions/substitions) pour transformer `u` en `v` . On la note $d(u, v)$.

On peut la calculer par récurrence : on va lire les deux mots de droite à gauche, et se demander quelle opération il faut effectuer à chaque fois. Pour transformer u en v :

- Soit la dernière lettre de u et la dernière lettre de v sont égales : dans ce cas, on peut enlever la dernière lettre des deux mots, et égaliser le reste.
- Sinon, on peut :
  - Enlever la dernière lettre de u et tenter de transformer le reste de u en v.
    Par exemple, pour égaliser ALGOR et ALTRU, on peut enlever le R de ALGOR puis essayer d'égaliser ALGO et ALTRU.
  - Ajouter à la fin de u la dernière lettre de v. On revient alors au tout premier cas : il faut ensuite transformer u (moins cette nouvelle dernière lettre) avec le reste de v.
    Par exemple, pour égaliser ALGOR et ALTRU, on peut transformer ALGOR en ALGORU puis égaliser sans les U finaux, donc égaliser ALGOR et ALTR.
  - Substituer à la dernière lettre de u la dernière lettre de v. On est encore ramenés au tout premier cas.
    Par exemple, pour égaliser ALGOR et ALTRU, on peut transformer ALGOR en ALGOU puis égaliser sans les U finaux, donc égaliser ALGO et ALTR.

0. En déduire une formule de récurrence $lev_{u,v}(i, j)$ qui trouve la distance d'édition entre els i premières lettres de u et les j premières lettres de v.

1. Quels sont ses cas de base ?

B Codons !

On travaillera en C, en utilisant des `char*` pour représenter les mots.

B.1 Calcul de la distance d'édition

2. Implémenter un algorithme efficace pour calculer la distance d'édition.

   Remarque. Il n'y a pas d'options en C. À la place, vous pouvez utiliser la valeur `INT_MIN` pour représenter le fait qu'une distance n'a pas encore été calculée. Comme elle est (très très) négative, il n'y a pas de risque de la confondre avec la valeur d'une distance.

Pour plus de confort [1], nous allons lire les deux mots u et v... depuis la ligne du terminal qui lance le code ! Modifiez votre main pour qu'il prenne ces arguments-ci :

```c
int main(int argc, char* argv[])
```

Le premier argument, `argc`, est le nombre de mots sur la ligne du terminal qui lance le code (les « mots » sont définis comme étant séparés par des espaces). Le second argument, `argv`, est le tableau de ces mots (il est donc de longueur `argc`).

Exemple. Dans la ligne de terminal ci-dessous, il y a trois mots : `./a.out` et `MAGIE` et `MANGUE`

Terminal

```
./a.out MAGIE MANGUE
```

Ainsi, on aurait `argc` valant 3, et `argv[0]` valant `./a.out`, `argv[1]` valant `MAGIE` et `argv[2]` valant `MANGUE`
Vous pouvez utiliser la fonction `strlen` de `string.h` pour calculer la longueur des mots stockés dans `argv`.

3. Modifiez votre code pour qu'il lise les deux mots dont on cherche la distance d'édition sur la ligne de commande.

B.2 Reconstruction de la solution optimale

On rappelle qu'un `char*` est une succession de caractères terminée par le caractère spécial `\0`

4. Écrire une fonction `void sub(char* u, int i, char x)` qui dans `u` substitue en position `i` par `x`. On supposera sans vérifier que la position est valide.
5. Écrire une fonction `void del(char* u, int i)` qui dans `u` supprime en position `i`. On supposera sans vérifier que la position est valide.
6. Écrire une fonction `void add(char* u, int i, char x)` qui dans `u` insère en position `i` le caractère `x`. On supposera sans vérifier que la position est valide ; et que l'espace mémoire alloué pour `u` est assez grand pour supporter une lettre de plus.
7. En déduire une fonction `void edition(char* u, len_u, char* v, len_v)` qui affiche un chemin optimal d'un mot à un autre. On commencera par recopier les deux mots dans une zone allouée de longueur la somme des longueur des deux mots plus un (cela garantit qu'aucun addition ne « débordera » de la zone mémoire).

1. et parce que c'est au programme