TP n° 4 d'Informatique
Programmation dynamique
Distance de Levenshtein

Objectif du TP

La programmation dynamique est une technique évoluée de conception d'algorithmes. Ces algorithmes ont pour but de résoudre des problèmes de calculs ou d'optimisation et le font souvent de manière récursive, mais sans refaire plusieurs fois le même calcul.

Pour que l'on puisse utiliser la programmation dynamique, il faut avoir :
- la **propriété de sous-structure optimale** du problème (principe d'optimalité de Bellman) : une partie de la solution optimale du problème global est elle-même la solution optimale du sous-problème correspondant.
- le **chevauchement des sous-problèmes** : le calcul de la solution optimale d'un sous-problème fait intervenir le calcul de celle d'un sous-problème différent ; les sous-problèmes ne sont pas indépendants. Remarque : s'ils le sont, les algorithmes de type *diviser pour régner* sont à considérer.

On appelle « équation de Bellman » l'équation de récurrence qui permet de calculer le coût d'un sous-problème à partir des sous-problèmes déjà traités.

On souhaite mettre en place un calcul efficace de « distance d'édition » entre deux mots, afin d'aboutir à un système de correction automatisée d'une saisie de texte.

Distance d'édition

La « distance d'édition » ou « distance de Levenshtein » (Vladimir Levenshtein, 1965) est la distance qui sépare deux chaînes de caractères : il s'agit du nombre minimal d'opérations élémentaires à réaliser pour transformer une chaîne de caractères en une autre.

Les opérations élémentaires considérées sont au nombre de trois :
- substitution d'un caractère (« `abcde` » → « `abfde` »)
- insertion d'un caractère (« `abcde` » → « `abcfde` »)
- suppression d'un caractère (« `abcde` » → « `abde` »)

La distance de Levenshtein est une distance au sens mathématique du terme. Pour tout couple de chaînes de caractères (s,t), elle vérifie en effet les propriétés :
- $d(s,t) \geq 0$
- $d(s,t) = 0 \iff s = t$
- $d(s,t) = d(t,s)$
- $d(s,t) \geq d(s,u) + d(u,t) \quad \forall u$

1 Équation de Bellman et programmation

Afin d'obtenir l'équation de Bellman, on cherche à décomposer le problème en sous-problèmes.

On note n et p les nombres respectifs de caractères des chaînes s et t.

On remarque qu'un sous-problème consistant à prendre en compte les i premiers caractères de s, notés s_i, et les j premiers caractères de t, notés t_j, peut être résolu (si $i > 1$ et $j > 1$) à l'aide des sous-problèmes de taille directement inférieure :
– si on connaît $d(s_{i-1},t_j)$, on peut réaliser une suppression de s_i à s_{i-1} et obtenir $d(s_i,t_j)$.
– si on connaît $d(s_i,t_{j-1})$, on peut réaliser une insertion de t_{j-1} à t_j et obtenir $d(s_i,t_j)$.
– si on connaît $d(s_{i-1},t_{j-1})$, on peut réaliser une substitution entre le i^{e} caractère de s et le j^{e} de t si besoin, puis obtenir $d(s_i,t_j)$. On note $\delta_{i,j} = 0$ si les deux caractères sont identiques, 1 s'ils sont différents.
Pour simplifier la suite, on note $d(i,j)$ la distance $d(s_i,t_j)$.

1. Déterminer l'équation de Bellman vérifiée par d, pour $i > 1$ et $j > 1$.
2. Combien vaut $d(0,0)$? $d(i,0)$? $d(j,0)$?
3. Écrire une fonction `levenshtein` qui prend en argument deux chaînes de caractères `s` et `t` et qui retourne la distance de Levenshtein les séparant.
   Quelle est la distance en les mots « rouge » et « rose » ?

2 Autocorrection

On souhaite utiliser ce calcul de distance pour déterminer, parmi des mots d'une liste donnée, le mot le plus proche d'un mot saisi par l'utilisateur. Pour ce faire, on imagine utiliser une structure de dictionnaire, ayant pour clé chaque mot de la liste et pour valeur la distance avec le mot saisi.

4. Écrire une fonction `mindico` ayant comme argument un dictionnaire et retournant, sous forme de liste, la ou les clés correspondant à la valeur minimale présente dans le dictionnaire.
   La tester ensuite avec le dictionnaire `{"a":2, "b":4, "c":1, "d":2, "e":1}`.
5. Écrire la fonction `autocorrection_simple` ayant comme argument un mot `mot` et une liste `liste` de mots, qui retourne la liste des mots de `liste` les plus proches de `mot` au sens de la distance de Levenshtein.
   Parmi les mots `essai`, `test`, `encore`, `maison`, lequel est le plus proche de `esai` ?

On peut facilement récupérer l'ensemble des mots de la langue française. On se propose d'utiliser ici un dictionnaire de 20000 mots environ. Il est à récupérer sur le site de la classe, sous forme texte.

Rappel : pour lire un fichier ligne à ligne, la syntaxe est la suivante :

```
fichier = open("TP4-liste.txt")
for mot in fichier:
    [ instructions utilisant la variable mot ]
fichier.close()
```

De plus, la variable `mot` contient alors toute une ligne, y compris le caractère `\n` « retour à la ligne », qu'il faut supprimer en écrivant par exemple `mot = mot[:-1]`.

6. Écrire une fonction `autocorrection` prenant un unique argument `mot` de type chaîne de caractère et retournant la liste des mots les plus proches de `mot`.