Distance d'Édition

Mofan Zhang

Novembre 2020

Distance de Levenshtein

Équation de Bellman

On étudie la distance de Levenshtein entre le string **A** et le string **B**, respectivement de longueur de m et n. On peux définir une fonction **distLevs[i][j]** dont la valeur est la distance de Levenshtein entre le préfixe $\mathbf{A_i}$ et le préfixe $\mathbf{B_j}$ [1]. La matrice **distLevs** est de dimension $(m+1)*(n+1)$.

Si l'un des deux préfixes est vide, la distance entre les préfixes est évidemment la longueur de l'autre préfixe. On a donc :

$$\begin{aligned} distLevs[i][0] &= i, \quad \forall i \\ distLevs[0][j] &= j, \quad \forall j \end{aligned} \tag{1}$$

Pour $i > 0$ et $j > 0$, on considère toutes les modifications à effectuer pour passer de $\mathbf{A_i}$ à $\mathbf{B_j}$. En effet, la suite de modifications peuvent être interprétées de manière suivante :

- Cas 1 : Les modifications pour passer de $\mathbf{A_{i-1}}$ à $\mathbf{B_j}$, et puis supprimer le dernier caractère de $\mathbf{A_i}$;
- Cas 2 : Les modifications pour passer de $\mathbf{A_i}$ à $\mathbf{B_{j-1}}$, et puis ajouter le $j^{ème}$ caractère de $\mathbf{B_j}$ à la fin;

[1] Le préfixe $\mathbf{S_i}$ est un substring de **S** constitué des i premiers caractères de **S**.

- Cas 3 : Les modifications pour passer de $\mathbf{A_{i-1}}$ à $\mathbf{B_{j-1}}$, et puis remplacer le dernier caractère de $\mathbf{A_i}$ par le dernier caractère de $\mathbf{B_j}$ si les deux ne sont pas identiques;
- Cas 4 : Les modifications pour passer de $\mathbf{A_{i-1}}$ à $\mathbf{B_{j-1}}$, et ne faire plus rien si le dernier caractère de $\mathbf{A_i}$ est identique que le dernier caractère de $\mathbf{B_j}$.

On choisit à chaque fois le meilleur (en minimisant le nombre de modifications) de ces 4 cas et on obtient ainsi la distance de Levenshtein pour passer de $\mathbf{A_i}$ à $\mathbf{B_j}$. L'équation de Bellman de ce problème peut donc être présentée comme ceci :

$$distLevs[i][j] = \min \begin{cases} 0 & i,j = 0 \\ distLevs[i-1][j] + 1 & i > 0 \\ distLevs[i][j-1] + 1 & j > 0 \\ distLevs[i-1][j-1] + coût & i,j > 0 \end{cases} \tag{2}$$

où :

$$coût = \begin{cases} 0 & \text{si } A[i-1] = B[j-1] \\ 1 & \text{sinon} \end{cases} \tag{3}$$

Complexité

Ensuite, on fait une comparaison de complexité entre différentes solutions (récursive, récursive avec mémoïsation externe, et itérative). Dans notre cas, le conteneur utilisé pour la mémoïsation externe est un **std::map** (table de hachage). Dans la solution itérative, c'est un **std::vector** de deux dimensions qui est utilisé.

Table 1: Comparaison de la complexité entre différentes solutions

	Complexité en temps	Complexité en espace
Solution récur. sans mém.	$\Omega(2^{max(m,n)})$ (expon.)	expon.
Solution récur. avec mém.	$O(mn)$ (polyn.)	$O(mn)$ (polyn.)
Solution itérative	$O(mn)$ (polyn.)	$O(mn)$ (polyn.)

Évidemment, la solution récursive sans mémoïsation externe est très exhaustive. Pour calculer la distance de Levenshtein entre le string "ecoles" et le string "eclose", cette solution prend 9.644 secondes dans ma machine, alors que la solution récursive avec mémoïsation et la solution itérative ne prennent que 0.001 seconde.

Distance de Damerau-Levenshtein

La distance de Damerau-Levenshtein est un développement de la distance de Levenshtein montrée ci-dessus. On autorise dans ce cas la transposition de deux caractères successifs. Il faut faire attention que maintenant la solution optimale de préfixe ne génère pas forcément la solution optimale de string initial comme précédent, parce que l'opération de transposition peut impacter la structure de string (préfixe) passé et donc rendre la solution ne plus valide. Un cas très particulier à considérer est l'opération sous forme de $\mathbf{abc} -> \mathbf{cda}$ dont le coût est donné par $|b| + |d| + 1$ [1]. Compte tenu de ce dernier, on peut écrire l'équation de Bellman du problème de distance de Damerau-Levenshtein :

$$
distDL[i][j] = \min \begin{cases} 0 & i,j = 0 \\ distDL[i-1][j] + 1 & i > 0 \\ distDL[i][j-1] + 1 & j > 0 \\ distDL[i-1][j-1] + coût & i,j > 0 \\ \\ \min\ distDL[k-1][l-1] + ... & \\ \qquad ... + (i-k) + (j-l) - 1 & pour \begin{cases} 0 < k < i \\ 0 < l < j \\ A[i-1] = B[l-1] \\ A[k-1] = B[j-1] \end{cases} \end{cases} \tag{4}
$$

où :

$$
coût = \begin{cases} 0 & \text{si } A[i-1] = B[j-1] \\ 1 & \text{sinon} \end{cases} \tag{5}
$$

References

[1] Leonid Boytsov. Indexing methods for approximate dictionary searching: Comparative analysis. *Journal of Experimental Algorithmics (JEA)*, 16:1–1, 2011.