Algorithmique
TD n° 6 : Programmation dynamique

Exercice 1 : Calcul de la distance de Levensthein

La distance de Levensthein (ou distance d'édition), mesure le nombre minimal de modifications nécessaires pour transformer une chaîne en une autre, grâce à des opérations d'insertion, de suppression ou de substitution. Par exemple, la distance entre A = "banane" et B = "banana" est de 1 car il suffit de substituer la dernière lettre de A par un a.
On définit $L(i, j)$ ($i <= |A|$ et $j <= |B|$), le nombre d'opérations nécessaires pour passer du sous mot de A contenant ses i premières lettres au sous mot de B contenant ses j premières lettres. Ainsi sur notre exemple $L(3, 4)$ représente la distance de "ban" à "bana" et vaut 1.
Pour calculer cette distance, on peut utiliser la récurence suivante :

$$L(i,j) = \begin{cases} max(i,j) & \text{si } min(i,j) == 0 \\ L(i-1,j-1) & \text{si } A_i == B_i \\ 1 + \min \begin{cases} L(i-1,j-1) \\ L(i-1,j) \\ L(i,j-1) \end{cases} \end{cases}$$

Chacune des 3 dernières situations correspond à une opération de modification, d'insertion, ou de suppression.

1. Appliquer l'algorithme sur les chaînes "banane" et "ananas"
2. Expliciter l'algorithme de programmation dynamique.
3. Donner la complexité asymptotique de cette algorithme.
4. Est-ce qu'on peut l'améliorer ?

Exercice 2 : mise en page et impression d'un texte

On dispose d'un fichier texte sans passage à la ligne (il s'agit d'un seul paragraphe) que l'on veut imprimer sur une feuille de taille donnée. Chaque caractère imprimé occupe la même largeur, y compris les espaces, comme sur une machine à écrire mécanique. Le problème est de découper le texte en lignes, pour minimiser les espaces qui restent à la fin de chaque ligne. Un mot ne peut pas être découpé.
Formellement, le texte est une suite de n mots de longueurs $\ell_1, \ell_2, \ldots, \ell_n$, mesurées en nombre de caractères. Chaque ligne contient exactement M caractères, blancs compris. On suppose que toutes les longueurs $\ell_1, \ell_2, \ldots, \ell_n$ sont inférieures ou égales à M (un mot peut tenir entièrement dans une ligne). Si une ligne contient les mots i à j (inclus), où $i \leqslant j$, et qu'on laisse exactement un espace entre deux mots, le nombre de caractères blancs à la fin de la ligne est

$$B := M - j + i - \sum_{k=i}^{j} \ell_k.$$

On veut minimiser la somme, sur toutes les lignes, des cubes [1] des B (nombres de caractères blancs présents à la fin de la ligne). Cette somme est appelé coût. On dira alors que la présentation du texte est « équilibrée ».
Voici un exemple du problème avec longueur de la ligne $M = 14$. Soit donné le texte suivant :

1. des grandes espaces à la fin de la ligne sont fortement pénaliser

```
La récursion est un concept important
```

On a $l_1 = 2$, $l_2 = 9$, $l_3 = 3$, $l_4 = 2$, $l_5 = 7$ et $l_7 = 9$. Pour l'alignement

```
La récursion  | B = 2
est un concept| B = 0
important     | B = 5
```

on a succesivement des valeurs de B de 2, 0 et 5 et donc une somme des cubes de 133.

1. Montrez que l'algorithme (glouton) qui ajoute un par un les mots aux lignes tant qu'ils tiennent dans la ligne ne donne pas toujours le meilleur résultat en l'appliquant sur l'exemple suivant pour des lignes de longueur 17 :

```
La programmation dynamique est une des bases algorithmiques
```

et donner la solution optimale.

2. Donner un algorithme de programmation dynamique qui résout le problème.
Indication : donner d'abord une relation de récurrence pour `cout(i)` qui dénote le coût de placer les mots de 1 à i. Pour cela définir `cout(j,i)` le coût d'une ligne avec les mots j à i (pour $j \leqslant i$)

3. Donner la complexité en temps et en espace de l'algorithme précédent.

4. Comment faut-il modifier l'algorithme pour que la dernière ligne ne soit pas prise en compte dans la somme des cubes des nombres de caractères blancs ?

Exercice 3 : le jeu de Double-Nim

On considère le jeu à deux joueurs suivant. Au début de la partie il y a n allumettes sur la table. Les deux joueurs jouent à tour de rôle en enlevant des allumettes. Celui qui enlève la dernière allumette a gagné. Les règles sont :

- au premier tour, le premier joueur ne peut pas prendre toutes les allumettes d'un coup ;
- aux tours suivants, un joueur doit prendre au minimum une allumette et au maximum le double du nombre pris par l'autre joueur au tour précédent.

1. Donnez une récurrence $J(n, m)$ disant si une position donnée est gagnante ou perdante selon le nombre n d'allumettes restantes et le nombre maximum m d'allumettes que l'on peut prendre.

2. Donner la table de J pour $n = 8$

3. Programmez cette récurrence sous forme d'une fonction récursive. Affichez pour tout n si le premier joueur a une stratégie gagnante, et combien d'appels récursifs ont été faits.

4. Interpréter l'affichage :

- proposer une hypothèse sur les nombres n pour lesquels le premier joueur gagne ;
- constater que le nombre d'appels récursifs est exponentiel en n.

5. Améliorez votre programme à l'aide de la programmation dynamique.

6. Affichez le déroulement d'une partie où un joueur qui peut gagner prend le plus petit nombre possible d'allumettes tel qu'il gagne, et un joueur qui ne peut pas gagner fait n'importe quoi.