

## 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  $A = \text{"banane"}$  et  $B = \text{"banana"}$  est de 1 car il suffit de substituer la dernière lettre de  $A$  par un  $a$ .

On définit  $L(i, j)$  ( $0 \leq i \leq |A|$  et  $0 \leq j \leq |B|$ ), 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 et  $L(0, 3)$  représente la distance du mot vide à "ban" et vaut 3.

Pour calculer cette distance, on peut utiliser la récurrence suivante (où  $A_i$  (pour  $1 \leq i \leq |A|$ ) désigne la  $i$ -ème lettre de  $A$ ) :

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

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" en remplissant un tableau adéquat.
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  $\ell_1, \ell_2, \dots, \ell_n$ , mesurées en nombre de caractères. Chaque ligne contient exactement  $M$  caractères, blancs compris. On suppose que toutes les longueurs  $\ell_1, \ell_2, \dots, \ell_n$  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  $i \leq j$ , et qu'on laisse exactement un espace entre deux mots, le nombre de caractères blancs à la fin de la ligne  $u$  est

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

On souhaite avoir une répartition du texte minimisant les blancs à la fin de chaque ligne. En élevant les  $B_u$  à la puissance 3, on crée un score de pénalité qu'on peut appliquer sur l'ensemble

des lignes. Ainsi, en minimisant la somme des pénalités, on limitera le nombre de vides à la fin des lignes. On appellera cette somme le coût.

Voici un exemple du problème avec longueur de la ligne  $M = 14$ . Soit donné le texte suivant :

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 | B0 = 2
est un concept| B1 = 0
important     | B2 = 5
```

on a successivement des valeurs de  $B_u$  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 la formule pour connaître la pénalité à appliquer pour une ligne contenant les mots de  $i$  à  $j$ . On appellera la fonction liée  $penalite(i, j)$ .
3. En utilisant la fonction  $penalite$ , donner une formule de récurrence pour calculer  $cout(i)$ , le coût pour le texte jusqu'au  $i$ -ème mot.
4. Écrire un algorithme de programmation dynamique à partir de cette récurrence. Quelle est sa complexité ?
5. On ne veut pas que la dernière ligne soit pénalisée par les vides restants. Comment faut-il modifier l'algorithme pour qu'aucune pénalité ne soit appliqué à cette ligne ?

### 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.