Question : Soit **C[i]** le coût **minimal** de la répartition des **i** premiers mots du texte. Si **n** est le nombre de mots du texte, on cherche donc à calculer **C[n]**. Etablir une formule de récurrence permettant de calculer **C[i]** pour **i** compris entre **1** et **n**. Justifier précisément son raisonnement.

Réponse : avec 1<=i<=j

Admettons une matrice de coût pour « sss uu pp iiiii » de taille maximale égale à 6 :

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 0 | INF | INF |
| - | 4 | 1 | INF |
| - | - | 4 | INF |
| - | - | - | 1 |

Afin d’avoir une matrice exploitable, on met tous les éléments à la puissance de 3 :

|  |  |  |  |
| --- | --- | --- | --- |
| 27 | 0 | INF | INF |
| - | 64 | 1 | INF |
| - | - | 64 | INF |
| - | - | - | 1 |

Afin de trouver l’optimisation exacte dans cette matrice, on doit trouver pour j entre i et la taille de la matrice le minimum pour le coût actuel à l’aide de la formule C[i-1] la valeur du coût précédent + la matrice[i][j-1] ce qui correspond au coût de la ligne actuelle.

Voici comment se constitue le tableau permettant d’avoir le coût minimal :

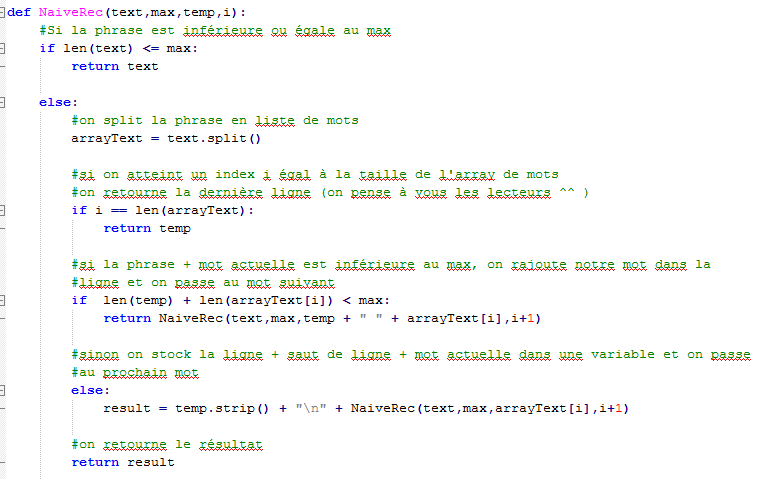
|  |  |  |  |
| --- | --- | --- | --- |
| 29 | 2 | 65 | 1 |

En employant la formule de récurrence, ici on peut voir que le coût minimal est de 29.

Ce qui confirme davantage notre formule de récurrence ci-dessus.

Question : Estimer la complexité de cet algorithme (algorithme naïf).

Ici, on a :



Donc d’après le master Théorème, on prend le **meilleur des cas** donc lorsque la phrase est coupée en deux donc b = 2 et a = 1 car on ne fait qu’une récursive. Vu qu’à la fin, nous fusionnons les mots, on a 1 fusion pour le meilleur des cas donc **a< b^d** donc nous avons **O(n^d)** mais vu que l’on est dans le **meilleur des cas**, on a donc **O(n^2)** donc une complexité **quadratique**.

Question : Estimer la complexité de cet algorithme (algorithme Bottom-Up).

D’après la question précédente, on un **O(n^d),** or ici tous nos éléments de la matrice des coûts sont ramenés au cube. On peut donc en déduire qu’on a **O(n^3)** donc une complexité cubique.

Question : Proposer et implémenter en Python une méthode gloutonne pour résoudre ce problème. Est-elle optimale ? Si oui justifier pourquoi, sinon proposer un contre-exemple.

Prenons par exemple notre célèbre phrase sss uu pp iiiii avec une taille 6 maximum par ligne.

La sortie de cette formule, on a :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| s | s | s |  | u | u |
| p | p |  |  |  |  |
| i | i | i | i | i |  |

Nous avons un coût de 4^3 + 1^3, ce qui fait 65 or dans la première question. Pour cette même phrase, nous avons un coût minimum de 29. Nous pouvons donc en conclure que la méthode gloutonne n’est pas la solution la plus optimale.