Découpe d'une corde

par David Latreyte

Un magasin achète des cordes d'escalade de longueur L et les découpe (soigneusement) en cordes plus petites pour les vendre à ses clients. On souhaite déterminer un découpage optimal pour maximiser le revenu, sachant que les prix de ventes p_i d'une corde de i mètres sont donnés.
Par exemple, supposons qu'on dispose d'une corde de $L=10$ mètres, avec les prix de ventes indiqués dans le tableau suivant :

i	0	1	2	3	4	5	6	7	8	9	10
p_i	0	1	5	8	9	10	17	18	20	24	26

1 Stratégie gloutonne

1) Rappeler en quoi consiste une stratégie gloutonne.

▶ Lorsqu'on met en œuvre une stratégie gloutonne, on cherche, à chaque étape, le meilleur choix « du moment », c'est à dire un choix optimum local.

2) On définit la *densité* d'une corde de longueur i comme étant le rapport p_i/i, c'est-à-dire son prix au mètre. Expliquer en quoi cette grandeur pourrait être pertinente dans la mise en œuvre d'un algorithme glouton.
Remarque : une grandeur comparable a été utilisée dans le problème du sac à dos.

▶ Puisqu'on cherche à maximiser le profit, l'idée est de commencer à découper la corde par les longueurs qui rapportent le plus à l'instant où l'on effectue la découpe, c'est à dire les segments dont la densité est la plus grande.

3) Quelle instruction mathématique permettrait de déterminer par combien de segments de longueur i une corde de longueur L peut être découpée ?

▶ Division entière : $L//i$

4) Quelle instruction mathématique permettrait de déterminer la longueur de corde restante après la découpe de segments de longueur i ?

▶ Modulo : $L\%i$

5) Écrire le code de la fonction `sol_gloutonne` dont la spécification est :

```
def sol_gloutonne(L, s, p):
    """
    Détermine grâce à une stratégie gloutonne la découpe optimale
    d une corde, connaissant les longueurs des segments possibles
```

```
    et les valeurs associées.

    Paramètres
    ----------
    L : int
        Longueur de la corde
    s : list[int]
        Liste des segments possibles
    p : list[int]
        Prix associés aux segments

    Valeur retournée
    ----------------
    (somme, {segment : nombre})
        Tuple formé de la somme gagnée "optimale"
        et du dictionnaire des segments de découpe accompagnés de leur nombre.
    """
```

Utiliser les deux instructions suivantes dans le corps de la fonction (se renseigner sur l'influence du paramètre `key` dans la définition de la fonction `sorted`) :

```
donnees = [(s[i], p[i], p[i] / s[i]) for i in range(1, len(s))]
donnees = sorted(donnees, key=lambda donnee: donnee[2])
```

▶
```
def sol_gloutonne(L, s, p):
    """
    Détermine grâce à une stratégie gloutonne la découpe optimale
    d une corde, connaissant les longueurs des segments possibles
    et les valeurs associées.

    Paramètres
    ----------
    L : int
        Longueur de la corde
    s : list[int]
        Liste des segments possibles
    p : list[int]
        Prix associés aux segments

    Valeur retournée
    ----------------
    (somme, {segment : nombre})
        Tuple formé de la somme gagnée "optimale"
        et du dictionnaire des segments de découpe accompagnés de leur nombre.
    """
    # Préparation des données en vue du tri sur la densité
    donnees = [(s[i], p[i], p[i] / s[i]) for i in range(1, len(s))]

    # Tri des données par densité croissante
    donnees = sorted(donnees, key=lambda donnee: donnee[2])

    # Valeurs retournées
```

```python
    decoupe = {}  # Couples (segment : nbre)
    somme = 0

    i = 0
    while i < len(donnees) and L != 0:
        segment = donnees[-1 - i][0]
        # Pas la peine de considérer les segments trop grands
        if segment <= L:
            # Nbre de segments
            nbre = L // segment
            # Longueur de corde restante
            L = L % segment
            # Ajout au dictionnaire résultat
            decoupe[segment] = nbre
            # Ajout à la somme totale
            somme += nbre * donnees[-1 - i][1]
        i += 1

    return (somme, decoupe)
```

2 Utilisation de la programmation dynamique

6) Rappeler quel est l'objectif de la programmation dynamique.

▶ La programmation dynamique est une technique dont le principe consiste à découper un problème en sous-problèmes (non indépendants les uns des autres), à les résoudre et à stocker les résultats de ces sous-problèmes dans un tableau afin de les réutiliser.

7) Le somme maximale obtenue, lorsqu'on utilise le $i^{\text{ème}}$ segment (de longueur i donc) pour découper une corde de longueur j est donnée par la relation :

$$S[i][j] = \begin{cases} 0 & \text{si } j = 0 \\ 0 & \text{si } i = 0 \\ \max\left(S[i-1][j]; p[i] + S[i][j-i]\right) & \text{si } j \geqslant i \\ S[i-1][j] & \text{si } j < i \end{cases}$$

où p est le prix d'une découpe de longueur i. Justifier cette relation.

Remarque : une relation comparable a été donnée dans le problème du sac à dos.

▶ – Il est évident que si la longueur de la corde est nulle, la somme obtenue l'est également. La conclusion est identique si la longueur du segment est nulle.

– Dans le cas général, *deux choix sont possibles : on utilise le segment de longueur i ou on ne l'utilise pas. La solution est alors la valeur maximale entre ces deux choix.*

  – Si on n'utilise pas le segment de longueur i, la solution est alors la somme maximale déterminée en utilisant tous les segments de longueurs inférieures à i, pour une même longueur de corde.

- Si on utilise le segment de longueur i, la somme est le prix de la découpe de ce segment auquel il faut ajouter la somme maximale obtenue en découpant une corde de longueur $j-i$.
  Ici, on conserve l'indice i car il est tout à fait possible que l'on soit amené à découper la corde de longueur $j-i$ par la longueur i.

– La solution générale impose $j \geqslant i$. Si la longueur du segment i est supérieure à la longueur de la corde, il ne reste que $S[i-1][j]$.

8) Représenter le tableau S pour une longueur de corde $L = 4\,\text{m}$.

▶

	0	**1**	**2**	**3**	**4**	L
0	0	0	0	0	0	
1	0	1	2	3	4	
2	0	1	5	6	10	
3	0	1	5	8	10	
4	0	1	5	8	10	
i						

$S[1][1] = \max(S[0][1]; 1 + S[1][0]) = \max(0; 1+0) = 1$

$S[1][2] = \max(S[0][2]; 1 + S[1][1]) = \max(0; 1+1) = 2$

$S[1][3] = \max(S[0][3]; 1 + S[1][2]) = \max(0; 1+2) = 3$

$S[1][4] = \max(S[0][4]; 1 + S[1][3]) = \max(0; 1+3) = 4$

$S[2][1] = S[1][1] = 1$

$S[2][2] = \max(S[1][2]; 5 + S[2][0]) = \max(2; 5+0) = 5$

9) Quelle est la valeur de la somme maximale obtenue en utilisant la programmation dynamique ?

▶ La somme est celle obtenue en considérant dans le tableau la longueur de la corde (ici $L = 4\,\text{m}$) et tous les types de segments disponibles (ici 4).

C'est donc 10.

10) Comparer le résultat précédent à celui obtenu grâce à la stratégie gloutonne. Cette dernière a-t-elle conduit à un découpage optimal ?

▶ Le choix d'optima locaux (stratégie gloutonne) ne conduit pas toujours à la solution optimale.

11) Écrire le code de la fonction `sol_dynamique` dont la spécification est :

```python
def sol_dynamique(L, s, p):
    """
    Détermine en utilisant la programmation dynamique la découpe optimale
    d une corde, connaissant les longueurs des segments possibles et les
    valeurs associées.

    Paramètres
    ----------
    L : int
        Longueur de la corde
    s : List[int]
        Liste des segments possibles
    p : List[int]
        Prix associés aux segments
```

```
    Valeur retournée
    ----------------
    S : List[List[int]]
        Tableau des solutions maximales des sous problèmes.
    """
```

12) Quelle est la complexité de cet algorithme ?

▶ La complexité est en $\theta(n \times L)$ où n est le nombre de longueurs possibles.

13) À partir du tableau de la question 8, indiquer comment retrouver le découpage conduisant à la solution.

▶ – À chaque étape deux choix sont possibles, on sélectionne le segment de longueur i ou on ne le sélectionne pas. Dans ce second cas, la solution est $S[i][j] = S[i-1][j]$; il suffit de remonter d'une case verticalement dans le tableau si ces valeurs sont égales. Dans le cas contraire, la solution est $S[i][j] = p[i] + S[i][j-i]$; il faut donc se déplacer horizontalement vers la gauche d'une valeur égale à $j - i$.

– $S[4][4] = S[3][4]$: pas de segment 4 ;

– $S[3][4] = S[2][4]$: pas de segment 2 ;

– $S[2][4] \neq S[1][4]$: un segment 2 et maintenant, il faut considérer $S[2][4-2] = S[2][2]$;

– $S[2][2] \neq S[1][2]$: un segment 2 et maintenant il faut considérer $S[2][2-2] = S[2][0] = 0$

Il faut donc découper la corde de longueur 4 en deux segments de longueur 2.

14) Écrire la fonction `recherche_sol` dont la spécification est

```
def recherche_sol(L, s, S):
    """
    Recherche la solution et toutes les découpes correspondant à cette solution.

    Paramètres
    ----------
    L : int
        Longueur de la corde
    s : List[int]
        Liste des segments possibles
    S : List[List[int]]
        Tableau des solutions optimales des sous problèmes.

    Valeur retournée
    ----------------
    (somme, {segment : nombre})
        Tuple formé de la somme gagnée maximale
        et du dictionnaire des segments de découpe accompagnés de leur nombre.
```

```
    """
```

▶
```python
def recherche_sol(L, s, S):
    """
    Recherche la solution et toutes les découpes correspondant à cette solution.

    Paramètres
    ----------
    L : int
        Longueur de la corde
    s : List[int]
        Liste des segments possibles
    S : List[List[int]]
        Tableau des solutions optimales des sous problèmes.

    Valeur retournée
    ----------------
    (somme, {segment : nombre})
        Tuple formé de la somme gagnée maximale
        et du dictionnaire des segments de découpe accompagnés de leur nombre.
    """
    # Case en bas à droite du tableau
    j = L  # indice de la colonne
    i = len(s) - 1  # indice de la ligne
    # Somme optimale
    somme = S[i][j]

    # Recherche des découpes
    decoupe = {}

    while j > 0 or i > 0:
        if S[i][j] == S[i - 1][j]:
            # Segment i pas utilisé
            i = i - 1
        else:
            # Segment i utilisé
            decoupe[i] = 1 if i not in decoupe.keys() else decoupe[i] + 1
            j = j - s[i]

    return (somme, decoupe)
```