Programmation dynamique
TP découpe d'une barre

Christophe Viroulaud

Terminale - NSI

Algo 26

Un ferrailleur découpe des barres métalliques pour la revente. Les barres, plus petites, obtenues sont utilisées dans diverses constructions (garde-corps, châssis, fenêtre...). Le prix de revente des barres n'est donc pas proportionnel à leur dimension.

Longueur	1	2	3	4	5	6	8	10
Prix	2	5	8	10	11	14	17	20

Comment maximiser le prix de vente d'une barre ?

Sommaire

Approche naïve

Une formalisation mathématique du débitage d'une barre de longueur **`longueur`** permet d'écrire pour une première découpe de taille **`taille`** :

$$prix\_max(longueur) = prix(taille) + prix\_max(longueur - taille)$$

Si on effectue une première découpe **`taille`**, il reste une barre **`longueur-taille`**. L'objectif est de trouver le prix pour chaque découpe possible et ne garder que le prix maximum.

Activité 1 :

1. Déterminer *à la main* le prix maximal pour une barre de longueur 10.
2. Construire le début de l'arbre des possibilités de découpe permettant de calculer les prix de vente.
3. Construire un dictionnaire associant la longueur de barre à son prix.
4. Déterminer une fonction **récursive** permettant de calculer le prix maximal obtenu pour la découpe d'une barre. La fonction parcourra toutes les branches de l'arbre pour déterminer le prix maximum.

Avant de regarder la correction

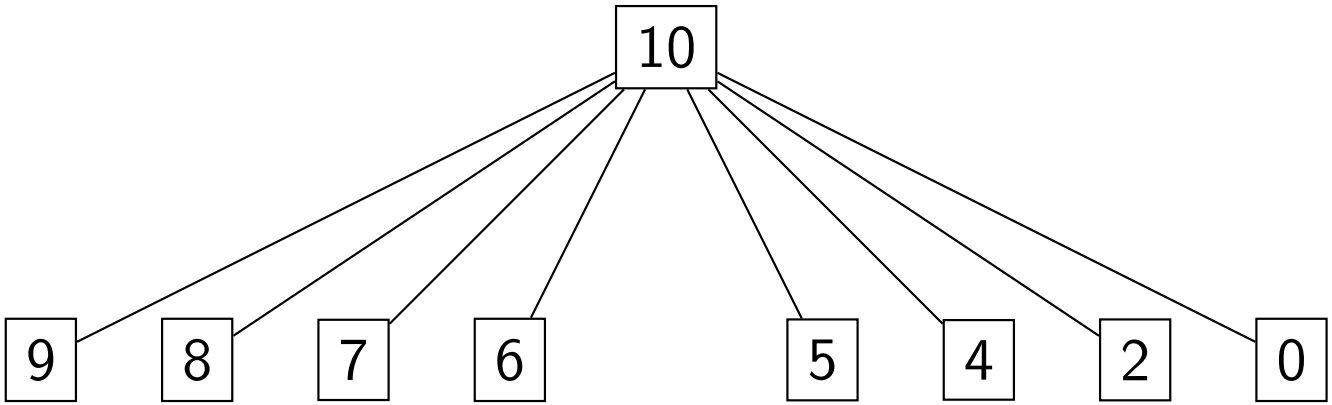
- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

- 3 découpes de taille 3 : $8 \times 3 = 24$
- 1 découpe de taille 1 : $2 \times 1 = 2$

Le prix maximal obtenu est 26.

Longueur	1	2	3	4	5	6	8	10
Prix	2	5	8	10	11	14	17	20



```
1 prix = {1: 2, 2: 5,
2         3: 8, 4: 10,
3         5: 11, 6: 14,
4         8: 17, 10: 20}
```

Code 1 – Chaque taille est associée à un prix.

```python
def decoupe_naif(l: int, prix: dict) -> int:
    if l == 0:
        return 0
    val_max = 0
    for taille, valeur in prix.items():
        if l >= taille:
            # descente dans chaque branche
            val_calc = decoupe_naif(l-taille, prix)
   + valeur
            # ne conserve que la meilleure branche
            if val_calc > val_max:
                val_max = val_calc
    return val_max
```

Sommaire

Programmation dynamique - Approche descendante

Pour éviter de parcourir les branches de l'arbre déjà calculées, on maintient un tableau des valeurs maximales pour chaque découpe.

```
1 track = [0 for _ in range(longueur+1)]
```

> **Activité 2 :** Écrire la fonction `decoupe_TD(l: int, prix: dict, track: int)` $\rightarrow$ `int` qui reprend l'algorithme naïf mais évite de calculer des valeurs déjà connues.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction


```
1  def decoupe_TD(l: int, prix: dict, track: int) -> int:
2      # valeur déjà calculée
3      if track[l] > 0:
4          return track[l]
5      if l == 0:
6          return track[0]
7      val_max = 0
8      for taille, valeur in prix.items():
9          if l >= taille:
10             # descente dans l'arbre
11             val_calc = decoupe_TD(l-taille, prix, track)
       + valeur
12             if val_calc > val_max:
13                 val_max = val_calc
14     # conservation du meilleur prix
15     track[l] = val_max
16     return track[l]
```

Sommaire

Approche ascendante

Une approche ascendante calcule le prix optimal des petites découpes et utilise ses valeurs pour déterminer les prix des découpes plus grandes.
L'algorithme s'écrit alors :

- Construire un tableau qui conserve les prix optimum pour chaque découpe, initialisés à 0.
- Pour chaque taille :
  - Initialiser le prix maximal à 0.
  - Pour chaque **`découpe`** possible :
    - Calculer le prix en ajoutant la valeur de la **`découpe`** et le prix optimal du reste de la barre déjà calculé.
    - Si le prix calculé est meilleur, mettre à jour le prix maximal.
  - Mettre à jour le tableau des prix optimum.
- Renvoyer le prix optimal pour la longueur de la barre.

Activité 3 : Écrire la fonction itérative `decoupe_BU(l: int, prix: dict)` $\rightarrow$ `int` qui implémente l'algorithme précédent.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

```
1  def decoupe_BU(l: int, prix: dict) -> int:
2      track = [0 for _ in range(l+1)]
3      for i in range(1, l+1):
4          val_max = 0
5          for taille, valeur in prix.items():
6              if i >= taille:
7                  val_calc = track[i-taille] + valeur
8                  if val_calc > val_max:
9                      val_max = val_calc
10         track[i] = val_max
11     return track[l]
```