Problème du rendu de monnaie

Programmation dynamique

Le problème du rendu de monnaie avec les pièces monnaie = (1, 2, 5) et la somme à rendre s = 13. Nous allons détailler l'approche **bottom-up** (tabulation) de la programmation dynamique pour résoudre ce problème.

Objectif

Trouver le nombre minimum de pièces nécessaires pour rendre la somme s = 13 en utilisant les pièces de valeurs 1, 2 et 5.

Approche bottom-up (tabulation)

1. **Initialisation** :
   - Créez un tableau dp de taille s + 1 = 14 (indices de 0 à 13).
   - Initialisez toutes les cases à +∞ (une valeur représentant l'infini), sauf dp[0] = 0 car aucune pièce n'est nécessaire pour rendre une somme de 0.

dp = [0, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞]

2. **Remplissage du tableau** :
   Pour chaque somme x de 1 à 13, et pour chaque pièce c dans (1, 2, 5), mettez à jour dp[x] comme suit :

$$dp[x]=min(dp[x],1+dp[x-c])dp[x]=min(dp[x],1+dp[x-c])$$

Cela signifie que si on peut rendre la somme x - c avec dp[x - c] pièces, alors on peut rendre x avec 1 + dp[x - c] pièces.

Détail des étapes

Étape 1 : x = 1

- Pièces disponibles : 1
  - x - 1 = 0 : dp[1] = min(+∞, 1 + dp[0]) = min(+∞, 1 + 0) = 1
  - dp = [0, 1, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞]

Étape 2 : x = 2

- Pièces disponibles : 1, 2
  - x - 1 = 1 : dp[2] = min(+∞, 1 + dp[1]) = min(+∞, 1 + 1) = 2
  - x - 2 = 0 : dp[2] = min(2, 1 + dp[0]) = min(2, 1 + 0) = 1
  - dp = [0, 1, 1, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞]

Étape 3 : x = 3

- Pièces disponibles : 1, 2
  - x - 1 = 2 : dp[3] = min(+∞, 1 + dp[2]) = min(+∞, 1 + 1) = 2
  - x - 2 = 1 : dp[3] = min(2, 1 + dp[1]) = min(2, 1 + 1) = 2
  - dp = [0, 1, 1, 2, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞]

Étape 4 : x = 4

- Pièces disponibles : 1, 2
  - x - 1 = 3 : dp[4] = min(+∞, 1 + dp[3]) = min(+∞, 1 + 2) = 3
  - x - 2 = 2 : dp[4] = min(3, 1 + dp[2]) = min(3, 1 + 1) = 2
  - dp = [0, 1, 1, 2, 2, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞]

Étape 5 : x = 5

- Pièces disponibles : 1, 2, 5
  - x - 1 = 4 : dp[5] = min(+∞, 1 + dp[4]) = min(+∞, 1 + 2) = 3
  - x - 2 = 3 : dp[5] = min(3, 1 + dp[3]) = min(3, 1 + 2) = 3
  - x - 5 = 0 : dp[5] = min(3, 1 + dp[0]) = min(3, 1 + 0) = 1
  - dp = [0, 1, 1, 2, 2, 1, +∞, +∞, +∞, +∞, +∞, +∞, +∞, +∞]

Étape 6 : x = 6

- Pièces disponibles : 1, 2, 5
  - x - 1 = 5 : dp[6] = min(+∞, 1 + dp[5]) = min(+∞, 1 + 1) = 2
  - x - 2 = 4 : dp[6] = min(2, 1 + dp[4]) = min(2, 1 + 2) = 2
  - x - 5 = 1 : dp[6] = min(2, 1 + dp[1]) = min(2, 1 + 1) = 2
  - dp = [0, 1, 1, 2, 2, 1, 2, +∞, +∞, +∞, +∞, +∞, +∞, +∞]

Étape 7 : x = 7

- Pièces disponibles : 1, 2, 5
  - x - 1 = 6 : dp[7] = min(+∞, 1 + dp[6]) = min(+∞, 1 + 2) = 3
  - x - 2 = 5 : dp[7] = min(3, 1 + dp[5]) = min(3, 1 + 1) = 2
  - x - 5 = 2 : dp[7] = min(2, 1 + dp[2]) = min(2, 1 + 1) = 2
  - dp = [0, 1, 1, 2, 2, 1, 2, 2, +∞, +∞, +∞, +∞, +∞, +∞]

Étape 8 : x = 8

- Pièces disponibles : 1, 2, 5

    - x - 1 = 7 : dp[8] = min(+∞, 1 + dp[7]) = min(+∞, 1 + 2) = 3
    - x - 2 = 6 : dp[8] = min(3, 1 + dp[6]) = min(3, 1 + 2) = 3
    - x - 5 = 3 : dp[8] = min(3, 1 + dp[3]) = min(3, 1 + 2) = 3
    - dp = [0, 1, 1, 2, 2, 1, 2, 2, 3, +∞, +∞, +∞, +∞, +∞]

Étape 9 : x = 9

- Pièces disponibles : 1, 2, 5
    - x - 1 = 8 : dp[9] = min(+∞, 1 + dp[8]) = min(+∞, 1 + 3) = 4
    - x - 2 = 7 : dp[9] = min(4, 1 + dp[7]) = min(4, 1 + 2) = 3
    - x - 5 = 4 : dp[9] = min(3, 1 + dp[4]) = min(3, 1 + 2) = 3
    - dp = [0, 1, 1, 2, 2, 1, 2, 2, 3, 3, +∞, +∞, +∞, +∞]

Étape 10 : x = 10

- Pièces disponibles : 1, 2, 5
    - x - 1 = 9 : dp[10] = min(+∞, 1 + dp[9]) = min(+∞, 1 + 3) = 4
    - x - 2 = 8 : dp[10] = min(4, 1 + dp[8]) = min(4, 1 + 3) = 4
    - x - 5 = 5 : dp[10] = min(4, 1 + dp[5]) = min(4, 1 + 1) = 2
    - dp = [0, 1, 1, 2, 2, 1, 2, 2, 3, 3, 2, +∞, +∞, +∞]

Étape 11 : x = 11

- Pièces disponibles : 1, 2, 5
    - x - 1 = 10 : dp[11] = min(+∞, 1 + dp[10]) = min(+∞, 1 + 2) = 3
    - x - 2 = 9 : dp[11] = min(3, 1 + dp[9]) = min(3, 1 + 3) = 3
    - x - 5 = 6 : dp[11] = min(3, 1 + dp[6]) = min(3, 1 + 2) = 3
    - dp = [0, 1, 1, 2, 2, 1, 2, 2, 3, 3, 2, 3, +∞, +∞]

Étape 12 : x = 12

- Pièces disponibles : 1, 2, 5
    - x - 1 = 11 : dp[12] = min(+∞, 1 + dp[11]) = min(+∞, 1 + 3) = 4
    - x - 2 = 10 : dp[12] = min(4, 1 + dp[10]) = min(4, 1 + 2) = 3
    - x - 5 = 7 : dp[12] = min(3, 1 + dp[7]) = min(3, 1 + 2) = 3
    - dp = [0, 1, 1, 2, 2, 1, 2, 2, 3, 3, 2, 3, 3, +∞]

Étape 13 : x = 13

- Pièces disponibles : 1, 2, 5

- x - 1 = 12 : dp[13] = min(+∞, 1 + dp[12]) = min(+∞, 1 + 3) = 4
- x - 2 = 11 : dp[13] = min(4, 1 + dp[11]) = min(4, 1 + 3) = 4
- x - 5 = 8 : dp[13] = min(4, 1 + dp[8]) = min(4, 1 + 3) = 4
- dp = [0, 1, 1, 2, 2, 1, 2, 2, 3, 3, 2, 3, 3, 4]

Résultat final

- dp[13] = 4
- La somme 13 peut être rendue avec **4 pièces** (par exemple, 5 + 5 + 2 + 1).

Tableau final dp

dp=[0,1,1,2,2,1,2,2,3,3,2,3,3,4]

L'algorithme :

- On utilise un tableau dp pour stocker les résultats des sous-problèmes.
- On remplit le tableau de manière itérative, en partant des petites sommes jusqu'à amount.

```
Fonction coinChangeIteratif(coins, amount):
    dp = tableau de taille (amount + 1) initialisé à +∞
    dp[0] = 0

    Pour x de 1 à amount:
Pour chaque c dans coins:
        Si x - c >= 0:
            dp[x] = min(dp[x], 1 + dp[x - c])

    Si dp[amount] == +∞:
        Retourner -1
    Sinon:
        Retourner dp[amount]
```

Conclusion

En utilisant la programmation dynamique bottom-up, nous avons déterminé que le nombre minimum de pièces pour rendre la somme 13 est **4**. Cette approche est efficace et évite les calculs redondants en stockant les résultats intermédiaires dans un tableau.

Exercice : Compléter le tableau dans le cas monnaie = (1, 3, 4) et s = 10.