IFT2125-6001 TA: Maëlle Zimmermann

Démonstration 10

1

Question: Donner un algorithme pour rendre la monnaie avec le moins de pièces possibles, en supposant que chaque type de pièce existe en quantité illimitée.

Solution: Soient $c_1, c_2, ..., c_n > 0$ la valeur des pièces disponibles en quantité illimitée. Supposons que l'on désire rendre la monnaie sur k unités. Nous construisons un tableau T de taille $n \times (k+1)$ tel que $T[i, j]$ est le nombre de pièces minimum afin de rendre la monnaie sur j unités en utilisant seulement les i premières pièces.

Notons d'abord que $T[i, 0] = 0$ pour tout $1 \leq i \leq n$ puisqu'il n'y a aucune pièce à rendre. De plus,

$$T[i,j] = \min(\underbrace{T[i-1,j]}_{\text{ne pas prendre pièce } c_i}, \underbrace{T[i, j-c_i]+1}_{\text{prendre pièce } c_i})$$

en supposant que chaque case à l'extérieur du tableau vaut implicitement $+\infty$. En effet, si on n'utilise pas la $i$ème pièce, le nombre de pièces rendues sera identique au nombre de pièces nécessaires pour rendre le même montant j en utilisant seulement les $i-1$ premières pièces. Si on utilise la $i$ème pièce, on utilisera une pièce de plus que le nombre de pièces nécessaires pour rendre le montant restant, de valeur $j - c_i$. On choisira d'utiliser ou non la $i$ème pièce selon laquelle de ces deux options nécessite le moins de pièces.

Le nombre minimum de pièces nécessaires afin de rendre la monnaie sur k unités sera donc $T[n, k]$. Il faut encore une étape supplémentaire afin d'identifier les pièces à rendre. Pour cela, il suffit de débuter à la case $T[n, k]$ du tableau et retrouver le cheminement qui a été pris depuis $T[0, 0]$. Si $T[n, k]$ provient de $T[i-1, j]$, alors la pièce i n'est jamais utilisée. Si $T[n, k]$ provient de $T[i, j-c_i]$ alors la pièce i a été utilisée. Ce processus est répété itérativement jusqu'à arriver à $T[0, 0]$.

Voici une implémentation de cet algorithme:

```
def nb_pieces(c, k):
```

```
    T = [[0]*(k+1) for i in range(len(c))]

    for i in range(len(c)):
        for j in range(1,k+1):
            a = T[i-1][j] if i > 0 else float("inf")
            b = T[i][j-c[i]] if j >= c[i] else float("inf")
            T[i][j] = min(a, b+1)

    return T

def monnaie(c, k):
    p = [0] * len(c)
    T = nb_pieces(c, k)
    i, j = len(c)-1, k

    while (i, j) != (0, 0):
        a = T[i-1][j] if i > 0 else float("inf")
        b = T[i][j-c[i]] if j >= c[i] else float("inf")

        if T[i][j] == a:
            i -= 1
        else:
            j -= c[i]
            p[i] += 1

    return p
```

Le temps d'exécution exact de `monnaie` est dans $\theta(nk)$.

2

Question: Donner un algorithme qui rend la monnaie même lorsque le nombre de pièces disponibles est limité.

Solution: Il suffit de créer une ligne pour chaque occurrence d'une pièce puis d'utiliser la règle:

$$T[i,j] = \min(\underbrace{T[i-1,j]}_{\text{ne pas prendre pièce } c_i}, \underbrace{T[i-1,j-p_i]+1}_{\text{prendre pièce } c_i})$$

où p_i est la pièce associée à la ligne i. Toutes les cases sont initialisées à $+\infty$ à l'exception de la première colonne qui est initialisée à 0. Voici une implémentation possible:

```
def nb_pieces(c, s, k):
    T = [[float("inf")] * (k+1) for i in range(sum(s)+1)]
```

```python
    P = [0] + [p for i in range(len(c)) for p in [c[i]] * s[i]]

    for i in range(len(T)):
        T[i][0] = 0

    for i in range(1, len(T)):
        for j in range(1, k+1):
            a = T[i-1][j] if i > 0 else float("inf")
            b = T[i-1][j-P[i]] if j >= c[i] else float("inf")
            T[i][j] = min(a, b+1)

    return T

def monnaie(c, s, k):
    T = nb_pieces(c, s, k)
    P = [None] + [x for i in range(len(c)) for x in [i] * s[i]]
    p = [0] * len(c)
    j = k

    for i in reversed(range(1, len(T))):
        a = T[i-1][j]
        b = T[i-1][j-P[i]] if j >= P[i] else float("inf")
    if T[i][j] != a:
        p[P[i]] += 1
        j -= c[P[i]]
    return p
```