écrites dans *Opt*. Une fois ces matrices remplies en suivant la formule de récurrence décrite ci-haut, un parcours des éléments en ordre inverse permet d'extraire de la matrice *Sel* l'ensemble des éléments ayant produit la valeur optimale.

```
def knapsack(p, v, cmax):
    n = len(p)
    Opt = [[0] * (cmax + 1) for _ in range(n + 1)]
    Sel = [[False] * (cmax + 1) for _ in range(n + 1)]
    #                                   --- cas de base
    for cap in range(p[0], cmax + 1):
        Opt[0][cap] = v[0]
        Sel[0][cap] = True
    #                                   --- cas d'induction
    for i in range(1, n):
        for cap in range(cmax + 1):
            if cap >= p[i] and Opt[i-1][cap - p[i]] + v[i] > Opt[i-1][cap]:
                Opt[i][cap] = Opt[i-1][cap - p[i]] + v[i]
                Sel[i][cap] = True
            else:
                Opt[i][cap] = Opt[i-1][cap]
                Sel[i][cap] = False
    #                                   --- lecture solution
    cap = cmax
    sol = []
    for i in range(n-1, -1, -1):
        if Sel[i][cap]:
            sol.append(i)
            cap -= p[i]
    return (Opt[n - 1][cmax], sol)
```

11.2 Rendu de monnaie

À présent, on souhaite obtenir un montant R avec des pièces de monnaie ou des billets qui valent respectivement $x_0, \ldots, x_{n-1}$ centimes. Le problème consiste donc à déterminer s'il existe une combinaison linéaire positive de $x_0, \ldots, x_{n-1}$ qui vaut R. Vous rigolez, mais en Birmanie (aujourd'hui Myanmar), il y avait des billets de 15, 25, 35, 45, 75 et de 90 kyats (voir figure 11.2).

Pour ce problème, une valeur x_i peut contribuer plusieurs fois à une somme.

```
def coin_change(x, R):
    b = [False] * (R + 1)
    b[0] = True
    for xi in x:
        for s in range(xi, R + 1):
            b[s] |= b[s - xi]
    return b[R]
```

Variante Dans le cas où il existe une solution, on peut s'intéresser à celle qui utilise un nombre minimal de pièces.

Observation On pourrait penser qu'il suffirait de considérer les pièces par valeur décroissante et de donner à tout moment celle qui ne dépasse pas le montant restant.

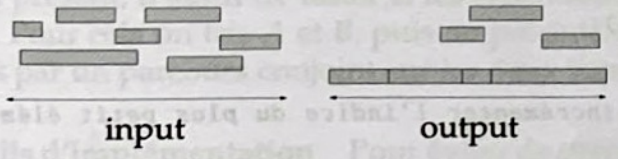
FIGURE 11.2 – Un billet de 45 kyats.

Cela réalise un nombre minimal de pièces pour le système monétaire européen, mais si on considère le système suivant : 1, 3, 4 et 10 et que l'on souhaite rendre la somme de 6, cette approche gloutonne résulte en une solution de 3 pièces composée de 4 + 1 + 1 tandis que l'unique solution optimale est 3 + 3.

Algorithme en $O(nR)$ Soit $A[i][m]$ le nombre minimal de pièces pour rendre un montant $0 \leq m \leq R$ parmi le système monétaire $x_0, \ldots, x_{i-1}$ et $A[i][m] = \infty$ s'il n'y a pas de solution. On dérive une relation de récurrence similaire à celle qu'on obtient pour le sac à dos : pour tout montant m, $A[0][m]$ vaut m/x_0 si x_0 divise m et ∞ sinon. Pour $i = 1, \ldots, n-1$ on a la relation :

$$A[i][m] = \max \begin{cases} A[i][m - x_i] + 1 & \text{cas où l'on choisit la pièce} \\ & \text{à condition que } m \geq x_i \\ A[i-1][m] & \text{cas où l'on ne choisit pas la pièce.} \end{cases}$$

11.3 Sous-ensemble de valeur totale donnée



Définition Étant donné n entiers positifs $x_0, \ldots, x_{n-1}$, on veut savoir s'il en existe un sous-ensemble dont la somme vaut une valeur R donnée. Ce problème est NP-difficile.

Algorithme en $O(nR)$ On maintient un tableau de booléens qui indique pour chaque indice i et chaque $0 \leq s \leq R$ s'il existe un sous-ensemble des entiers $x_0, x_1, \ldots, x_i$ dont la somme vaut s.

Initialement ce tableau n'est vrai qu'à l'indice 0, pour l'ensemble vide. Puis pour chaque $i \in \{0, \ldots, n-1\}$ et tout $s \in \{0, \ldots, R\}$, on peut produire un sous-ensemble de somme s avec les entiers $x_0, \ldots, x_i$, si et seulement s'il existe un sous-ensemble de $x_0, \ldots, x_{i-1}$ de somme s ou $s - x_i$.

Notez l'ordre de la boucle sur s dans l'implémentation.