LAB10: Programowanie dynamiczne – liniowe zagadnienie załadunku

Zadanie 1

Implementacja Binarnego problemu plecakowego 0/1:

In [50]:

```
import numpy as np

def Loading_problem(T, N, a, c):
    """
    T - ładowność
    N - ilość rodzajów ładunków
    a - lista wag ładunków
    c - lista zysków łądunków
    """

    #inicjalizacja tablic wyjściowych zawierających ilości ładunków
    #oraz wartości funckji
    x = np.zeros((T+1, N))
    f = np.zeros((T+1, N))
    f.fill(-np.inf)

    #iteracja po etapach i= <0, N-1>, ale etapy wykonują się od końca (reverse_idx)
    for i in range(N): #etap
        for j in range(T+1): #waga yi <0, T>

            #pominięcie części obliczeń pierwszego etapu
            if i == N-1 and j < T:
                continue

            #index pomocniczy do iteracji od końca
            reverse_idx = N - i - 1

            #maksymalna ilość przedmiotu o wadze a[] dla wagi j
            ymax = 1 if  j // a[reverse_idx] else 0
            #dla ostatniego etapu ustawiamy wartości bo nie ma wcześniejszych
            if i == 0:
                x[j][i] = ymax
                f[j][i] = ymax*c[reverse_idx]

            else:
                # iteracja po wartościach xi do ymax
                for xi in range(ymax+1):
                    #obliczenie maksymalnego zysku dla xi
                    gain = xi*c[reverse_idx] + f[j - a[reverse_idx]*xi][i - 1]

                    #sprawdzenie nowy zysk jest większy od obecnego dla innej wartości xi
                    if gain > f[j][i]:
                        x[j][i] = xi
                        f[j][i] = gain

    #formatowanie strategi optymalnej
    optimal_strategy = ""
    y = T
    for i in range(N-1, -1, -1):
        optimal_strategy += f'x{N - i - 1}={x[y, i]}, f(y{N - i})={f[y, i]},\n'
        y -= int(x[y, i]*a[N - i - 1])

    return x, f, optimal_strategy

c = np.array([1, 3, 2, 2])
a = np.array([1, 4, 3, 3])
T = 7
N = 4

#Testowy zestaw
x, f, optimal_strategy =  Loading_problem(T, N, a, c)
print(f"Macierz deyczji optymalnych\n{x}")
print(f"Macierz wartości funckji f(yi)\n{f}")
print(optimal_strategy)
```

```
Macierz deyczji optymalnych
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [1. 0. 0. 0.]
 [1. 0. 1. 0.]
 [1. 0. 1. 0.]
 [1. 1. 0. 0.]
 [1. 1. 1. 0.]]
Macierz wartości funckji f(yi)
[[   0.    0.    0. -inf]
 [   0.    0.    0. -inf]
 [   0.    0.    0. -inf]
 [   2.    2.    2. -inf]
 [   2.    2.    3. -inf]
 [   2.    2.    3. -inf]
 [   2.    4.    4. -inf]
 [   2.    4.    5.    5.]]
x0=0.0, f(y1)=5.0,
x1=1.0, f(y2)=5.0,
x2=0.0, f(y3)=2.0,
x3=1.0, f(y4)=2.0,
```

Zadanie 2

Wykonanie algorytmu dla przykładowych 10 zmiennych:

In [45]:

```
T = 20
N = 10
a = [1, 2, 3, 3, 2, 2, 2, 4, 2, 3]

c = [1, 2, 4, 4, 3, 3, 3, 6, 4, 7]


x, f, optimal_strategy =  Loading_problem(T, N, a, c)
print(f"Macierz deyczji optymalnych\n{x}")
```

```
Macierz deyczji optymalnych
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 0. 0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 0. 0. 1. 0. 0. 0.]
 [1. 1. 1. 0. 0. 0. 0. 0. 0. 0.]
 [1. 1. 1. 0. 0. 0. 1. 0. 0. 0.]
 [1. 1. 1. 1. 0. 0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 0. 0. 1. 0. 0. 0.]
 [1. 1. 1. 1. 1. 0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 1. 0. 1. 0. 0. 0.]
 [1. 1. 1. 1. 1. 1. 0. 0. 0. 0.]
 [1. 1. 1. 1. 1. 1. 1. 0. 0. 0.]
 [1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [1. 1. 1. 1. 1. 1. 1. 0. 0. 0.]
 [1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]]
```

In [46]:

```
print(f"Macierz wartości funckji f(yi)\n{f}")
```

```
Macierz wartości funckji f(yi)
[[   0.    0.    0.    0.    0.    0.    0.    0.    0. -inf]
 [   0.    0.    0.    0.    0.    0.    0.    0.    0. -inf]
 [   0.    4.    4.    4.    4.    4.    4.    4.    4. -inf]
 [   7.    7.    7.    7.    7.    7.    7.    7.    7. -inf]
 [   7.    7.    7.    7.    7.    7.    7.    7.    7. -inf]
 [   7.   11.   11.   11.   11.   11.   11.   11.   11. -inf]
 [   7.   11.   11.   11.   11.   11.   11.   11.   11. -inf]
 [   7.   11.   13.   14.   14.   14.   14.   14.   14. -inf]
 [   7.   11.   13.   14.   14.   14.   15.   15.   15. -inf]
 [   7.   11.   17.   17.   17.   17.   17.   17.   17. -inf]
 [   7.   11.   17.   17.   17.   17.   18.   18.   18. -inf]
 [   7.   11.   17.   20.   20.   20.   20.   20.   20. -inf]
 [   7.   11.   17.   20.   20.   20.   21.   21.   21. -inf]
 [   7.   11.   17.   20.   23.   23.   23.   23.   23. -inf]
 [   7.   11.   17.   20.   23.   23.   24.   24.   24. -inf]
 [   7.   11.   17.   20.   23.   26.   26.   26.   26. -inf]
 [   7.   11.   17.   20.   23.   26.   27.   27.   27. -inf]
 [   7.   11.   17.   20.   23.   26.   27.   28.   28. -inf]
 [   7.   11.   17.   20.   23.   26.   30.   30.   30. -inf]
 [   7.   11.   17.   20.   23.   26.   30.   31.   31. -inf]
 [   7.   11.   17.   20.   23.   26.   30.   31.   32.   32.]]
```

In [47]:

```
print(f"Strategia optymalna, w której zawiera się wektor decyzyjny (kolumna xi)\noraz wartość uzyskanej funkcji
```

```
Strategia optymalna, w której zawiera się wektor decyzyjny (kolumna xi)
oraz wartość uzyskanej funkcji celu:

x0=0.0, f(y1)=32.0,
x1=1.0, f(y2)=32.0,
x2=0.0, f(y3)=30.0,
x3=1.0, f(y4)=30.0,
x4=1.0, f(y5)=26.0,
x5=1.0, f(y6)=23.0,
x6=1.0, f(y7)=20.0,
x7=1.0, f(y8)=17.0,
x8=1.0, f(y9)=11.0,
x9=1.0, f(y10)=7.0,
```

W ten sposób udało się uzyskać wartość funkcji celu f jako 32.

Zadanie 3

1. Jakie założenia muszą być spełnione dla wag i zysków? Wartości wag i zysków powinny być dodatnie. Dodtkowo bierzemy pod uwagę wagę w tej samej jednostce miary dla każdego przedmiotu.

2. Co się stanie jeśli te założenia nie spełnimy? Jeżeli wartości wag lub zysków będą ujemne to możemy otrzymać ujemny wynik zysku całkowitego co oznacza stratę. W tym przypadku algorytm normalnie się wykona zwracając ujemny wynik.

3. Jaka jest złożoność obliczeniowa algorytmu? W teorii złożoność takiego algorytmu powinna wynosić (zgodnie z oznaczeniami powyżej T- pojemność, N - liczba produktów) O(N*T), gdyż wykonywana jest iteracja po całej macierzy rozmiaru TxN.