In [4]:
```
import numpy as np
```

Dominik Matracki 408558

Problem plecakowy 0-1 KP

Mamy plecak o maksymalnej pojemnosci B oraz zbior N elementow. Kazdy element charakteryzuje sie zyskiem oraz waga. Dyskretny problem plecakowy:

- Maksymalizuj zysk $\sum_{j=1}^{n} c_j x_j$
- Przy ograniczeniach wagi $B \sum_{j=1}^{n} a_j x_j \leq B, x_j \in 0, 1$

Przyklad problemu plecakowego

$x_1 + 3x_2 + 2x_3 + 2x_4 - > max$

$x_1 + 4x_2 + 3x_3 + 3x_4 \leq 7$

In [5]:
```
from typing import List

def maxArg(array: List[int]):
    if len(array) == 0:
        return

    maximum = -np.inf
    arg = None
    for i, el in enumerate(array):
        if el > maximum:
            arg = i
            maximum = el
    return arg, maximum


def binaryKP(weights, prices, capacity):
    n = len(weights)
    table = [[0 for _ in range(capacity+2)] for _ in range(n+1)]

    # Tworze tabele zyskow
    for i in range(1, n+1):
        for w in range(1, capacity+2):
            if 0 < w-weights[i-1] < capacity + 1:
                table[i][w] = max(table[i-1][w], table[i-1][w-weights[i-1]] + prices[i-1
                continue
            table[i][w] = table[i-1][w]

    # Find max in last row
    arg, maximum = maxArg(table[-1])
    print("Tablica: \n", np.array(table))
    print("Maksymalna wartosc zysku: ", maximum)

    # Ukladam sciezke
    currentPrice = maximum
    i = n
    result = [0 for _ in range(n)]
    while i > 0:
        if table[i][arg] != table[i-1][arg]:
```

```
            result[i-1] = 1
            currentPrice -= p[i-1]
            arg = table[i-1].index(currentPrice)
        i -= 1
    return result
```

In [6]:
```
w = [1, 4, 3, 3, 1, 2, 2, 5, 3, 5]
p = [1, 3, 2, 2, 9, 3, 4, 4, 2, 4]
binaryKP(w, p, 7)
```

```
Tablica:
 [[ 0  0  0  0  0  0  0  0  0]
 [ 0  0  1  1  1  1  1  1  1]
 [ 0  0  1  1  1  3  4  4  4]
 [ 0  0  1  1  2  3  4  4  5]
 [ 0  0  1  1  2  3  4  4  5]
 [ 0  0  9 10 10 11 12 13 13]
 [ 0  0  9 10 12 13 13 14 15]
 [ 0  0  9 10 13 14 16 17 17]
 [ 0  0  9 10 13 14 16 17 17]
 [ 0  0  9 10 13 14 16 17 17]
 [ 0  0  9 10 13 14 16 17 17]]
Maksymalna wartosc zysku:  17
```

Out[6]:
```
[1, 0, 0, 0, 1, 1, 1, 0, 0, 0]
```

In [49]:
```
def shipProblem(a, d, q, capacity):
    n = len(a)
    result = np.zeros((capacity+1, n*2+1))
    f_col = 2
    x_col = 1
    j = 2
    while f_col < n*2+1 and x_col < n*2:
        i = 0
        while i < capacity+1:

            minimal = np.inf
            amount = 0
            # Find minimal value in values
            for x in range(min(i//a[j], d[j]) + 1):
                if i - a[j]*x < 0:
                    value = q[x][j]
                else:
                    value = q[x][j] + result[i-a[j]*x][f_col-2]
                if minimal > value:
                    minimal = value
                    amount = x

            result[i][f_col] = minimal
            result[i][x_col] = amount
            i += 1
        f_col += 2
        x_col += 2
        j -= 1
    print("Tablica wynikow: \n", result)
    # Znalezienie rozwiazania

    items = [0 for _ in range(n)]
    # Znalezienie minimum w ostatniej kolumnie
    i = 0
    column = n*2
    arg = np.argmin(result[:,column])
    currentCost = result[arg][column] - q[int(result[arg][column-1])][i]
    items[i] = result[arg][column-1]
    i += 1
```

```
    column -= 2
    while i < n:
        # Find arg with currentCost in column
        for k in range(capacity):
            if result[k][column] == currentCost:
                arg = k
                break
        currentCost = result[arg][column] - q[int(result[arg][column-1])][i]
        items[i] = result[arg][column-1]
        i += 1
        column -= 2
    return items
```

In [50]:
```
a = [1, 2, 3]
d = [6, 3, 2]

q = np.array([
    [20, 9, 6],
    [18, 6, 2],
    [14, 3, 0],
    [11, 0, 0],
    [7, 0, 0],
    [2, 0, 0],
    [0, 0, 0],
])
shipProblem(a, d, q, 7)
```

```
Tablica wynikow:
 [[ 0.  0.  6.  0. 15.  0. 35.]
 [ 0.  0.  6.  0. 15.  1. 33.]
 [ 0.  0.  6.  1. 12.  2. 29.]
 [ 0.  1.  2.  0. 11.  3. 26.]
 [ 0.  1.  2.  2.  9.  4. 22.]
 [ 0.  1.  2.  1.  8.  5. 17.]
 [ 0.  2.  0.  3.  6.  6. 15.]
 [ 0.  2.  0.  2.  5.  5. 14.]]
```

Out[50]:
```
[5.0, 1.0, 0.0]
```

Zadanie 3

Jakie zalozenia musza byc spelnione dla wag oraz zyskow? Co stanie sie jesli te zalozenia nie spelnimy?

Wagi oraz zyski musza byc dodatnie, poniwaz gdyby wagi byly ujemne to mozliwe by bylo dodawanie wszystkich przedmiotow oraz natychmiastowe uzyskanie najlepszego zysku. Natomiast gdyby zyski byly ujemne to nieoptymlane byloby dodawanie jakiegokolwiek przemiotu.

Jaka jest zlozonosc obliczeniowa algorytmu?

Zlozonosc obliczeniowa algorytmu dla problemu plecakowego wynosi $O(B \cdot n)$, gdzie n - liczba przedmiotow, B - pojemnosc.