LAB11: Programowanie dynamiczne – Wyznaczanie optymalnej wielkości partii produkcyjnej

Zadanie 1

In [1]:
```
import numpy as np

def Loading_problem(N, g, h, q, Y_max, Y_min, y0, yk):
    """
    N - liczba miesięcy
    g - koszt produkcji
    h - koszt składowania
    q -ilosc produktów w danym miesiącu
    Y_min - pojemność minimalna magazynu
    Y_max - pojemność maksymalna magazynu
    y0 - stan na początku
    yk - stan na końcu
    """

    yn = Y_max - Y_min + 1
    #inicjalizacja tablic wyjściowych zawierających ilości ładunków
    #oraz wartości funckji
    x = np.zeros((Y_max+1, N))
    f = np.zeros((Y_max+1, N))
    f.fill(np.inf)
    yi = np.array(range(Y_min, Y_max))
    XMAX = len(g)-1
    XMIN = 0
    #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(Y_min, Y_max+1): #waga yi <ymin, ymax>

            #pominięcie części obliczeń pierwszego etapu
            if i == N-1 and j > Y_min:
                x[j, i] = np.NaN
                f[j, i] = np.inf
                continue

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

            #dla ostatniego etapu ustawiamy wartości bo nie ma wcześniejszych
            if i == 0:
                x_min = x_max = yk + q[reverse_idx] - j
            # elif i == N-1:
            #     x_min = x_max = y0 + q[reverse_idx] - j
            #     print(x_min, x_max)
            else:
                x_min = Y_min + q[reverse_idx] - j if Y_min + q[reverse_idx] - j >
```

```python
                x_max = Y_max + q[reverse_idx] - j if Y_max + q[reverse_idx] - j <


            if x_min > x_max or XMIN >  x_max or x_min < XMIN or x_max > XMAX:
                x[j, i] = np.NaN
                f[j, i] = np.inf
                continue

            # else:
            # iteracja po wartościach xi do ymax
            for xi in range(x_min, x_max+1):
                if i == 0:
                    cost = g[xi] + h[j + xi - q[reverse_idx]]
                else:
                    #obliczenie maksymalnego zysku dla xi
                    cost = g[xi] + h[j + xi - q[reverse_idx] - Y_min] + f[j + xi -
                #sprawdzenie nowy zysk jest większy od obecnego dla innej wartości
                if cost < f[j][i]:
                    x[j][i] = xi
                    f[j][i] = cost

    #formatowanie strategi optymalnej
    optimal_strategy = ""
    yi = y0
    for i in range(N-1, -1, -1):
        optimal_strategy += f'x{N - i - 1}={x[yi - Y_min, i] - Y_min if i == N-1 el
        yi = int(yi + x[yi - Y_min, i] - q[N - i - 1])
    s = ' '
    optimal_strategy += f'{s :8}y{N - i}={yi - Y_min},\n'
    optimal_strategy += f'Całkowity zminimalizowany koszt wynosi: {f[Y_min][N-1]}'

    return x, f, optimal_strategy

def print_all(x, f, optimal_strategy):
    print(f"Macierz deyczji optymalnych\n{x}")
    print(f"Macierz wartości funckji f(yi)\n{f}")
    print(f'Strategia optymalna:\n{optimal_strategy}')

N = 6
q = np.array([3, 3, 3, 3, 3, 3])
h = np.array([0, 2, 4, 6, 8, 10])
g = np.array([0, 15, 18, 19, 20, 24])

# #Testowy zestaw z wykładu
print_all(*Loading_problem(N, g, h, q, Y_max=4, Y_min=0, y0=0, yk=0))
```

```
Macierz deyczji optymalnych
[[ 3.  3.  4.  3.  3.  4.]
 [ 2.  5.  5.  5.  5. nan]
 [ 1.  4.  4.  4.  4. nan]
 [ 0.  0.  0.  0.  0. nan]
 [nan  0.  0.  0.  0. nan]]
Macierz wartości funckji f(yi)
[[ 19.  38.  52.  71.  90. 104.]
 [ 18.  30.  49.  68.  82.  inf]
 [ 15.  26.  45.  64.  78.  inf]
 [  0.  19.  38.  52.  71.  inf]
 [ inf  20.  32.  51.  70.  inf]]
Strategia optymalna:
x0=4.0, y0=0,
x1=5.0, y1=1,
x2=0.0, y2=3,
x3=4.0, y3=0,
x4=5.0, y4=1,
x5=0.0, y5=3,
        y6=0,
Całkowity zminimalizowany koszt wynosi: 104.0
```