Sprawozdanie Programowanie dynamiczne – liniowe zagadnienie załadunku

Jakub Ochman grupa 3. AiR

Zadanie 1

Implementacja metody programowania dynamicznego dla całkowitoliczbowego problemu liniowego:

In [ ]:
```
def integer_knapsack(weights, profits, availability, capacity):
    # weights - wektor wag przedmiotów
    # profits - macierz zysków, gdzie profits[i][k] to zysk za (k+1)-szą sztukę i-tego przedmiotu
    # availability - wektor dostępności, czyli maksymalna liczba sztuk danego przedmiotu
    # capacity - maksymalna pojemność plecaka (ograniczenie wagowe)

    n = len(weights)  # liczba przedmiotów

    # dp[i][w] przechowuje maksymalny zysk dla przedmiotów od i do końca przy pojemności w
    dp = [[0] * (capacity + 1) for _ in range(n + 1)]

    # decisions[i][w] przechowuje liczbę wybranych sztuk i-tego przedmiotu przy pojemności w
    decisions = [[0] * (capacity + 1) for _ in range(n)]

    # iteracja po przedmiotach od ostatniego do pierwszego
    for i in range(n - 1, -1, -1):
        # iteracja po pojemnościach od 0 do capacity
        for w in range(capacity + 1):
            max_val = dp[i + 1][w]  # zysk bez wybrania i-tego przedmiotu
            max_k = 0
            total_profit = 0

            # sprawdzenie możliwych ilości i-tego przedmiotu do wybrania
            for k in range(1, availability[i] + 1):
                total_weight = k * weights[i]
                if total_weight <= w:
                    total_profit += profits[i][k - 1]  # suma zysków z k sztuk
                    val = total_profit + dp[i + 1][w - total_weight]  # całkowity zysk po
                    # dodaniu pozostałych przedmiotów
                    if val > max_val:
                        max_val = val
                        max_k = k
                else:
                    break  # dalsze zwiększanie k przekracza pojemność

            dp[i][w] = max_val  # zapis maksymalnego zysku
            decisions[i][w] = max_k  # zapis optymalnej liczby sztuk i-tego przedmiotu

    strategy = [0] * n
    remaining_capacity = capacity

    # odtworzenie strategii na podstawie macierzy decyzji
    for i in range(n):
        k = decisions[i][remaining_capacity]
        strategy[i] = k
        remaining_capacity -= k * weights[i]

    max_profit = dp[0][capacity]  # maksymalny zysk dla pełnej pojemności i wszystkich przedmiotów
    return dp, decisions, max_profit, strategy
```

Zadanie 2

Poniżej przedstawiono przykład działania algorytmu dla 10 rodzajów przedmiotów z ograniczeniami liczbowymi, wagami, zyskami i ograniczoną pojemnością ustawioną na 9.

In [37]:
```
weights =  [1, 3, 1, 1, 2, 1, 3, 2, 5, 3] # wektor wag
availability = [3, 2, 5, 2, 3, 4, 1, 2, 1, 3] # wektor dostępnosci

profits = [ # macierz zysków
    [3, 2, 1],
    [4, 3],
    [2, 2, 1, 1, 1],
    [1, 1],
    [3, 2, 2],
    [4, 3, 2, 1],
    [4],
    [3, 2],
    [5],
    [4, 3, 2]
]

capacity = 9 # ograniczenie pojemności

# uzupełnianie tabeli profits zerami, tak aby utworzyła
# sensowną macierz prostokątną
max_availability = max(availability)
profits_fixed = []
for p, avail in zip(profits, availability):
```

```python
    row = p + [0] * (max_availability - avail)
    profits_fixed.append(row)

print("Macierz zysków za wybranie poszczególnych przedmiotów")
for row in profits_fixed:
    print(" ".join(f"{val:>3}" for val in row))
print()

dp, decisions, max_profit, strategy = integer_knapsack(weights, profits_fixed, availability, capacity)

print(f"Maksymalizowana wartość zysku: {max_profit}\n")

print("Macierz decyzji:")
for row in decisions:
    print(" ".join(f"{val:>3}" for val in row))

print("\nMacierz maksymalnych zysków")
for row in dp:
    print(" ".join(f"{val:>3}" for val in row))

print("\nStrategia optymalna (liczba sztuk każdego przedmiotu):")
print(strategy)
```

```
Macierz zysków za wybranie poszczególnych przedmiotów
  3   2   1   0   0
  4   3   0   0   0
  2   2   1   1   1
  1   1   0   0   0
  3   2   2   0   0
  4   3   2   1   0
  4   0   0   0   0
  3   2   0   0   0
  5   0   0   0   0
  4   3   2   0   0

Maksymalizowana wartość zysku: 21

Macierz decyzji:
  0   0   0   1   1   1   1   2   1   2
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   1   2   1   2   1   2
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   1   0   0
  0   1   2   3   2   3   3   2   3   3
  0   0   0   0   0   0   1   0   1   1
  0   0   1   0   2   1   0   2   1   1
  0   0   0   0   0   1   0   0   1   0
  0   0   0   1   1   1   2   2   2   3

Macierz maksymalnych zysków
  0   4   7  10  12  14  16  18  19  21
  0   4   7   9  11  13  14  16  17  19
  0   4   7   9  11  13  14  16  17  19
  0   4   7   9  10  12  13  15  16  17
  0   4   7   9  10  12  13  15  16  17
  0   4   7   9  10  12  13  14  16  17
  0   0   3   4   5   7   8   9  11  11
  0   0   3   4   5   7   7   9  10  10
  0   0   0   4   4   5   7   7   9   9
  0   0   0   4   4   4   7   7   7   9
  0   0   0   0   0   0   0   0   0   0

Strategia optymalna (liczba sztuk każdego przedmiotu):
[2, 0, 2, 0, 0, 3, 0, 1, 0, 0]
```

Zadanie 3

Jakie zakłada się założenia odnośnie wartości wag i zysków przedmiotów?

    W algorytmie programowania dynamicznego zakłada się, że zarówno wagi, jak i zyski przedmiotów są wartościami
    całkowitymi i nieujemnymi. Jest to konieczne, ponieważ tablica dynamiczna (macierz stanu) indeksowana jest po wagach,
    a więc wartości muszą być liczbami całkowitymi. Dodatkowo, zakłada się sens fizyczny danych – nie ma przedmiotów o
    ujemnej wadze lub ujemnym zysku, ponieważ nie mają one praktycznego znaczenia w kontekście problemu załadunku.

Co się stanie, jeśli te założenia nie są spełnione? Jaka modyfikacja sposobu rozwiązania zadania?

    Jeżeli wagi lub zyski nie byłyby liczbami całkowitymi, nie da się bezpośrednio zastosować programowania dynamicznego,
    ponieważ nie można indeksować tablicy zmiennoprzecinkowymi wartościami.Jednym z możliwych rozwiązań jest
    przeskalowanie wartości poprzez przemnożenie przez potęgę liczby 10 wszystkich wartości. Jeśli zyski byłyby ujemne,
    należałoby zmodyfikować sposób interpretacji rozwiązania lub wykluczyć takie przedmioty na etapie przygotowania
    danych.

Jaka jest złożoność obliczeniowa algorytmu?

    Złożoność obliczeniowa algorytmu wynosi O(n * C * a), gdzie n to liczba przedmiotów, C to pojemność, a a to średnia
    liczba dostępnych sztuk przedmiotu. W praktyce, dla danych o ograniczonej dostępności, czas działania może być
    znacznie lepszy niż w najgorszym przypadku.

Źródła:

- Na podstawie materiałów z zajęć oraz wykładu
- Na podstawie opisu problemu: https://en.wikipedia.org/wiki/Knapsack_problem

---

Środowisko:

Jupiter Notebook w Visual Studio Code z rozszerzeniem Jupiter, Python