Jakub Rynio | nr albumu: 117323
Informatyka niestacjonarna
semestr: 3
Grupa laboratoryjna: 2

1 Opis problemu plecakoweg

Problem plecakowy (ang. *knapsack problem*) jest klasycznym problemem optymalizacyjnym w informatyce i teorii decyzji. Polega na wyborze przedmiotów z określonymi wagami w_i i wartościami v_i, tak aby zmieściły się w plecaku o pojemności W i maksymalizowały łączną wartość.

Formalna definicja problemu

Dany jest zbiór n przedmiotów, z których każdy i ma wagę w_i i wartość v_i. Pojemność plecaka wynosi W. Celem jest znalezienie podzbioru przedmiotów $S \subseteq \{1, 2, \ldots, n\}$, takiego, że:

$$\sum_{i \in S} w_i \leq W \quad \text{oraz} \quad \sum_{i \in S} v_i \text{ jest maksymalna.}$$

Warianty problemu plecakowego

- **Problem plecakowy 0-1:** Każdy przedmiot może być wybrany albo odrzucony ($x_i \in \{0, 1\}$).
- **Problem plecakowy z podzielnością:** Przedmioty mogą być dzielone ($0 \leq x_i \leq 1$).

Metody rozwiązania

Metoda zachłanna

Stosowana głównie do problemu z podzielnością. Polega na sortowaniu przedmiotów według stosunku wartości do wagi $\frac{v_i}{w_i}$ i dodawaniu ich do plecaka w tej kolejności, aż pojemność się wyczerpie.

Algorytm:

1. Sortuj przedmioty według $\frac{v_i}{w_i}$ w porządku malejącym.
2. Inicjalizuj $W_{res} = W$.
3. Dla każdego przedmiotu i:
   - Jeśli $w_i \leq W_{res}$, dodaj cały przedmiot do plecaka.
   - Inaczej dodaj jego część $\frac{W_{res}}{w_i}$.

Metody dokładne

- Programowanie dynamiczne
- Przeszukiwanie z nawrotami

Metody przybliżone

- Heurystyki
- Algorytmy ewolucyjne

2 Algorytm rozwiązujący problem plecakowy metodą zachłanną

```python
import os

def error():
    print("Nieprawidlowe dane wejsciowe")
    input()
    os.system("cls")

def get_number(prompt, return_type="float", only_poz = False, zero = True):
    while True:
        try:
            if return_type == "float":
                output = float(input(prompt))

            elif return_type == "int":
                output = int(input(prompt))
        except ValueError:
            error()
            continue

        if output < 0 and only_poz or output == 0 and zero == False:
            error()
            continue
        else:
            return output

def greedy(values, weights, names, capacity):
    # Oblicza stosunek warto ci do wagi i sortuje w kolejno ci malej cej
    items = sorted(enumerate(zip(values, weights, names)), key=lambda x: x[1][0] / x
        [1][1], reverse=True)

    totalValue = 0
    selectedItems = {}
    remainingCapacity = capacity

    for i, (value, weight, name) in items:
        if weight <= remainingCapacity:
            if name not in selectedItems:
                selectedItems[name] = 0
            selectedItems[name] += 1
            totalValue += value
            remainingCapacity -= weight

    return totalValue, selectedItems

capacity = get_number("Podaj maksymalna pojemnosc plecaka: ", "float", True, False)
items_number = get_number("Podaj ilosc przedmiotow: ", "int", True, False)

values, weights, count, names = [], [], [], []
weight_for_print, value_for_print, name_for_print, count_for_print = [], [],[], []

for i in range(items_number):
    print(f"Przedmiot {i + 1} z {items_number}: \n")

    while True:
        name = str(input("Podaj nazwe przedmiotu: "))
        if name in names:
```

```
56                print("Ten przedmiot juz zostal dodany, nie mozna go dodac ponownie!")
57                input()
58                os.system("cls")
59                continue
60            else: break
61        while True:
62            weight = get_number(f"Podaj wage przedmiotu {name}: ", "float", True, False)
63            if weight == capacity:
64                error()
65                continue
66            else: break
67        value = get_number(f"Podaj wartosc przedmiotu {name}: ", "float", True, True)
68        count_value = get_number(f"Podaj ilosc przedmiotu {name}: ", "int", True, False)
69
70        weight_for_print.append(weight)
71        value_for_print.append(value)
72        name_for_print.append(name)
73        count_for_print.append(count_value)
74
75        for j in range(count_value):
76            weights.append(weight)
77            values.append(value)
78            names.append(name)
79            count.append(count_value)
80
81    input()
82    os.system("cls")
83
84    odstep = 10
85    print(f"{'Nazwa':<{odstep}}{'Waga':<{odstep}}{'Wartosc':<{odstep}}{'Ilosc':<{odstep
          }}")
86    print(40 * "-")
87
88    for i in range(items_number):
89        print(f"{name_for_print[i]:<{odstep}}{weight_for_print[i]:<{odstep}}{
              value_for_print[i]:<{odstep}}{count_for_print[i]:<{odstep}}")
90        print(40 * "-")
91
92    maxValue, selectedItems = greedy(values, weights, names, capacity)
93
94    print("\n\nMaksymalna warto    :", maxValue)
95    print("\nWybrane przedmioty:")
96    for name, count in selectedItems.items():
97        print(f"  Przedmiot {name}: {count} szt.")
98    input()
```

Opis algorytmu:

Linia 8-24 (get_number) – funkcja walidacyjna,
Linia 26-42 (greedy) – główny algorytm zachłanny:

- *Linia 28:* Tworzy listę przedmiotów posortowaną według stosunku wartości do wagi (najpierw przedmioty, które oferują najwięcej wartości za jednostkę wagi).
- *Linia 34-40:* Wybiera przedmioty z posortowanej listy, zaczynając od tych o najwyższym stosunku, dopóki nie wyczerpie się pojemności plecaka.

- *Linia 42:* Zwraca całkowitą wartość wybranych przedmiotów oraz ich ilość w plecaku.

Linia 44-75 – Pobieranie danych od użytkownika
Linia 80-86 – Wyświetlenie tabeli z nazwami, wagami, wartościami i ilościami przedmiotów.
Linia 88-93 – Po wykonaniu algorytmu zachłannego, wyświetlana jest maksymalna wartość, którą można uzyskać, oraz lista wybranych przedmiotów wraz z ich ilością, które mieszczą się w plecaku.

3 Wnioski

Wnioski
Algorytm rozwiązujący problem plecakowy metodą zachłanną skutecznie dobiera przedmioty, maksymalizując wartość przy ograniczonej pojemności plecaka. Wykorzystuje prostą heurystykę sortowania przedmiotów według stosunku wartości do wagi i wybiera je w tej kolejności, aż pojemność plecaka zostanie wyczerpana. Choć algorytm jest szybki i łatwy do implementacji, nie gwarantuje optymalnego rozwiązania w każdym przypadku, szczególnie w problemach 0-1. Dla dokładniejszych wyników warto rozważyć metody programowania dynamicznego lub algorytmy przeszukiwania z nawrotami.