SPRAWOZDANIE 5

Wprowadzenie:
Celem niniejszego sprawozdania jest zbadanie złożoności obliczeniowej, oraz omówienie zasad działania poszczególnych algorytmów rozwiązywania problemu plecakowego. Wszystkie algorytmy były sprawdzano za pomocą, wcześniej wczytanych wartości z pliku tekstowego zawierającego informacje o liczbie przedmiotów, ich wartości, wadze oraz pojemność plecaka.

Problem plecakowy – jeden z najczęściej poruszanych problemów optymalizacyjnych. Nazwa zagadnienia pochodzi od maksymalizacyjnego problemu wyboru przedmiotów, tak by ich wartość sumaryczna była jak największa i jednocześnie mieściły się w plecaku.

```
def przedmioty_plik(file_path):
   with open(file_path, 'r') as file:
       liczba_przedmiotow, pojemnosc = map(int,
file.readline().split())
       przedmioty = []
       for _ in range(liczba_przedmiotow):
           przedmiot1 = []
           line = file.readline().split()
           rozmiar, waga = map(int, line)
           przedmiot1.append(rozmiar)
           przedmiot1.append(waga)
           przedmioty.append(przedmiot1)
       return pojemnosc, przedmioty
```

PRZYKŁADOWO:
4 4
1 2
2 1
4 2
4 1
Pierwsza linia oznacza, że lista przedmiotów zawiera “4 przedmioty a plecak ma wolne 4 miejsca (wagę) dostępnych na przedmioty”, każda kolejna oznacza przedmiot “r w” gdzie r to rozmiar przedmiotu a w jego wartość. Złożoność obliczeniowa została sprawdzona i sporządzona na wykresach za pomocą zewnętrznej biblioteki *timeit* służącej do wykonywania wykresów, oraz w excelu. W sprawozdaniu zostały zawarte trzy algorytmy, rozwiązywania problemu plecakowego w sposób:
-dynamiczny
-zachłanny

-siłowy
W pierwszej sekcji zostały zawarte wykresy
zależności czasu od liczby przedmiotów dla poszczególnych algorytmów. Oraz krótkie omówienie jak działają poszczególne funkcje.

SEKCJA 1

Algorytm dynamiczny, został zrealizowany za pomocą techniki projektowania algorytmów, polegającej na rozwiązaniu podproblemów, i zapamiętaniu ich wyniku, jest to bardzo podobny sposób do metody “dziel i zwyciężaj”.
Algorytm rozwiązuje problem przez zastosowanie macierzy kosztów która przechowuje rozwiązania podproblemów (kolejne wartości funkcji celu) Macierz decyzyjna (opcjonalna) – przechowuje wartości zmiennych decyzyjnych.

```
def dynamiczny(pojemnosc_plecaka, przedmioty):
    n = len(przedmioty)
    # Inicjalizacja tablicy dp o wymiarach (n+1) x (pojemnosc_plecaka+1)
wypełnionej zerami
    # +1 bo kolumny z 0 pozwala na przechowywanie wyników dla przypadków gdy
mamy 0 przedmiotów
    dp = [[0] * (pojemnosc_plecaka + 1) for _ in range(n + 1)]

    # Przechodzenie przez każdy przedmiot
    for i in range(1, n + 1):
        # Przechodzenie przez każdą możliwą wagę plecaka od 1 do
pojemnosc_plecaka
        for w in range(1, pojemnosc_plecaka + 1):
            # Sprawdzenie, czy aktualny przedmiot może być dodany do plecaka
(czy jego waga jest mniejsza lub równa bieżącej pojemności)
            if przedmioty[i - 1][0] <= w:
                # Wybór maksymalnej wartości między:
                # 1. Nie dodaniem aktualnego przedmiotu (wartość bez tego
przedmiotu)
                # 2. Dodaniem aktualnego przedmiotu (wartość z uwzględnieniem
tego przedmiotu)
                #max{V[i - 1,j] , V[i - 1,j - wi] + pi}
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - przedmioty[i -
1][0]] + przedmioty[i - 1][1])
            else:
                # Jeśli przedmiot nie może być dodany, zachowujemy poprzednią
wartość (bez tego przedmiotu)
                dp[i][w] = dp[i - 1][w]

    # Zwrócenie maksymalnej wartości plecaka (znajduje się w
dp[n][pojemnosc_plecaka])
    # oraz całej tablicy dp do dalszej analizy lub debugowania
    return dp[n][pojemnosc_plecaka], dp
```

Zasada działania jest opisana za pomocą komentarzy w kodzie.

Algorytm zachłanny - to rodzaj algorytmu, który podejmuje decyzje sekwencyjne, wybierając lokalnie najlepszą opcję w danym momencie, bez względu na

konsekwencje dalszych decyzji. Kluczową cechą algorytmu zachłannego jest to, że wybiera on rozwiązanie optymalne dla danego kroku, mając nadzieję, że prowadzi ono do optymalnego rozwiązania całego problemu.

Cechy algorytmu zachłannego:

- **Prostota**:
  - Algorytmy zachłanne są zazwyczaj łatwe do zrozumienia i zaimplementowania.
- **Szybkość**:
  - Mają zazwyczaj mniejszą złożoność obliczeniową w porównaniu do algorytmów, które przeszukują całe przestrzenie rozwiązań, takich jak algorytmy dynamiczne czy wyczerpujące (brute force).
- **Lokalna optymalność**:
  - Każda decyzja jest lokalnie optymalna w nadziei, że doprowadzi to do globalnego optimum.

```
def zachlanny(przedmioty1, pojemnosc_plecaka):
   przedmioty = copy.deepcopy(przedmioty1)

   # Dodaj wartość na jednostkę masy do każdej z list
   for przedmiot in przedmioty:
       wjm = przedmiot[1] / przedmiot[0]  # Oblicz wartość na jednostkę masy
(wartość / rozmiar)
       przedmiot.append(wjm)  # Dodaj tę wartość jako trzeci element w każdej
zagnieżdżonej liście

   # Posortuj przedmioty względem wartości na jednostkę masy malejąco
   posortowane_przedmioty = sorted(przedmioty, key=lambda x: x[2],
reverse=True)  # Sortuj według wartości na jednostkę masy (trzecia wartość w
każdej liście), malejąco

   plecak = []                             # Lista do przechowywania wybranych
przedmiotów
   waga_w_plecaku = 0                      # Całkowita waga przedmiotów w
plecaku
   wartosc_w_plecaku = 0                   # Całkowita wartość przedmiotów w
plecaku
   dostepne_miejsce = pojemnosc_plecaka  # Dostępne miejsce w plecaku

   for przedmiot in posortowane_przedmioty:    # Przejdź przez posortowaną
listę przedmiotów
       if przedmiot[0] <= dostepne_miejsce:    # Jeśli przedmiot mieści się w
pozostałym miejscu plecaka
           plecak.append(przedmiot)            # Dodaj przedmiot do plecaka
           dostepne_miejsce -= przedmiot[0]    # Zmniejsz dostępne miejsce w
plecaku o wagę przedmiotu
           waga_w_plecaku += przedmiot[0]      # Dodaj wagę przedmiotu do
całkowitej wagi w plecaku
```

```
            wartosc_w_plecaku += przedmiot[1]    # Dodaj wartość przedmiotu do
całkowitej wartości w plecaku
        elif dostepne_miejsce == 0:              # Jeśli nie ma już dostępnego
miejsca w plecaku
            break  # Przerwij pętlę

    return wartosc_w_plecaku, waga_w_plecaku,  plecak  # Zwróć listę wybranych
przedmiotów, całkowitą wagę, pozostałe dostępne miejsce i całkowitą wartość
```

Algorytm siłowy, znany również jako algorytm wyczerpujący (ang. brute-force algorithm), to metoda rozwiązywania problemów polegająca na przeszukiwaniu wszystkich możliwych rozwiązań i wybieraniu najlepszego. Jest to podejście niewymagające żadnych heurystyk ani zaawansowanych technik optymalizacyjnych, co sprawia, że jest uniwersalny, ale jednocześnie może być bardzo kosztowny pod względem czasu i zasobów.

Cechy algorytmu siłowego:

1. **Prostota**:
    - Algorytm siłowy jest zazwyczaj prosty do zaimplementowania, ponieważ polega na przeszukiwaniu wszystkich możliwych rozwiązań.
2. **Uniwersalność**:
    - Może być zastosowany do szerokiej gamy problemów, niezależnie od ich specyfiki.
3. **Dokładność**:
    - Gwarantuje znalezienie optymalnego rozwiązania, jeśli takie istnieje, ponieważ sprawdza wszystkie możliwe kombinacje.
4. **Wysoka złożoność obliczeniowa**:
    - Główną wadą algorytmów siłowych jest ich złożoność obliczeniowa. Dla problemów o dużej liczbie możliwych rozwiązań algorytmy te mogą być bardzo nieefektywne i czasochłonne.
    - Dla problemu plecakowego z n przedmiotami, liczba możliwych kombinacji to 2^n.
5. **Złożoność czasowa i pamięciowa**:
    - W najgorszym przypadku algorytm siłowy może mieć wykładniczą złożoność czasową i pamięciową, co sprawia, że jest praktycznie bezużyteczny dla dużych problemów.

```
def sum_rozwiazan(przedmioty):
    wartosc, waga = 0, 0              # Inicjalizacja zmiennych przechowujących
łączną wartość i wagę
    for przedm in przedmioty:         # Iteracja przez każdy przedmiot w rozwiązaniu
        wartosc += przedm[1]          # Dodaj wartość przedmiotu do łącznej wartości
        waga += przedm[0]             # Dodaj wagę przedmiotu do łącznej wagi
    return wartosc, waga              # Zwróć łączną wartość i wagę jako krotkę
(wartosc, waga)

# Funkcja rozwiązująca problem plecakowy metodą siłową
```

```python
def silowy(pojemnosc_plecaka, przedmioty):
    res = []  # Inicjalizacja listy do przechowywania możliwych rozwiązań

    # Iteracja przez wszystkie możliwe liczby przedmiotów (od 0 do liczby
przedmiotów)
    for przedmiot in range(len(przedmioty) + 1):
        # Generowanie wszystkich możliwych kombinacji 'przedmiot' przedmiotów
        for rozwiazania in itertools.combinations(przedmioty, przedmiot):
            # Oblicz łączną wartość i wagę dla danej kombinacji przedmiotów
            wartosc, waga = sum_rozwiazan(rozwiazania)
            # Sprawdź, czy łączna waga jest mniejsza lub równa pojemności plecaka
            if waga <= pojemnosc_plecaka:
                # Jeśli tak, dodaj kombinację do listy rozwiązań
                res.append((wartosc, waga, rozwiazania))

    # Sortuj listę rozwiązań w porządku malejącym według łącznej wartości
    res.sort(reverse=True, key=lambda x: x[0])

    # Zwróć posortowaną listę rozwiązań
    return res
```

SEKCJA 2

ALGORYTM DYNAMICZNY
wykres t=f(n) zależności czasu obliczeń t od liczby n przedmiotów, przy stałej pojemności plecaka b.

ALGORYTM ZACHŁANNY
wykres t=f(n) zależności czasu obliczeń t od liczby n przedmiotów, przy stałej pojemności plecaka b.

ALGORYTM SIŁOWY
wykres t=f(n) zależności czasu obliczeń t od liczby n przedmiotów, przy stałej pojemności plecaka b.

SKALA LOGARYTMICZNA
wykres t=f(n) zależności czasu obliczeń t od liczby n przedmiotów, przy stałej pojemności plecaka b, dla wszystkich trzech algorytmów plecakowych.

ALGORYTM DYNAMICZNY
wykres t=f(b) zależności czasu obliczeń t od pojemności plecaka b, przy stałej liczbie przedmiotów n.

ALGORYTM ZACHŁANNY
wykres t=f(b) zależności czasu obliczeń t od pojemności plecaka b, przy stałej liczbie przedmiotów n.

ALGORYTM SIŁOWY
wykres t=f(b) zależności czasu obliczeń t od pojemności plecaka b, przy stałej liczbie przedmiotów n.

ALGORYTM DYNAMICZNY
wykres t=f(n,b) zależności czasu obliczeń t od liczby n przedmiotów i pojemności plecaka b.

ALGORYTM ZACHŁANNY
wykres t=f(n,b) zależności czasu obliczeń t od liczby n przedmiotów i pojemności plecaka b.

ALGORYTM SIŁOWY
wykres t=f(n,b) zależności czasu obliczeń t od liczby n przedmiotów i pojemności plecaka b.

SEKCJA 3

Wnioski

- Algorytm Dynamiczny: Skuteczny i dokładny, ale może być czasochłonny dla dużych danych.
- Algorytm Zachłanny: Szybki i prosty, ale nie zawsze daje optymalne rozwiązanie.
- Algorytm Siłowy: Gwarantuje optymalne rozwiązanie, ale jest bardzo nieefektywny dla dużej liczby przedmiotów ze względu na wykładniczą złożoność obliczeniową.

Z względu na swój sposób działania, poszczególne algorytmy mają **złożoność obliczeniową:**

Zachłanny - $O(n*\log_2*n)$

Wyczerpujący/siłowy - $O(2^n)$

Dynamiczny $O(n*c)$

Złożoność pamięciową:

Zachłanny - O(n)

Wyczerpujący/siłowy - O(2)

Dynamiczny O(n*c)