

Programowanie dynamiczne

Sprawozdanie z laboratorium 9 – Piotr Sarna LK1

Cel ćwiczenia

Podczas zajęć zapoznaliśmy się z techniką programowania dynamicznego. Następnie wykorzystaliśmy ją rozwiązania problemu plecakowego.

Wstęp teoretyczny

Programowanie dynamiczne to technika rozwiązywania problemów, poprzez rozwiązanie problemu dla mniejszego zestawu danych, zapamiętaniu go, a następnie wykorzystaniu go przy wyznaczaniu powiększonego zestawu danych, do momentu rozwiązania problemu dla całego zestawu danych.

Jednak aby rozwiązać problem techniką programowania dynamicznego, problem musi być sformułowany rekurencyjnie. Dodatkową wadą jest fakt, że w celu rozwiązania problemu musimy zająć dodatkową pamięć, na tablicę przechowującą wyniki dla mniejszych zestawów danych.

Jeśli jednak możemy rozwiązać problem techniką programowania dynamicznego, możemy wykorzystać jego następujące zalety:

- Problemy o strukturze rekurencyjnej podproblemów można tą metodą rozwiązać w czasie wielomianowym.
- Po wyznaczeniu rozwiązań wszystkich podproblemów czas wyznaczenia głównego problemu jest liniowy.

Programowanie dynamiczne możemy wykorzystać do rozwiązania następujących problemów:

- Wyznaczenie n-tego wyrazu ciągu Fibonacciego
- Rozwiązanie problemu plecakowego
- Wyznaczenie wartości silni
- Wyznaczenie współczynnika dwumianowego
- Problem podziału zbioru

Opis algorytmu

Problem plecakowy polega na wyznaczeniu takiego ułożenia elementów w plecaku przedmiotów (każdy o określonej wadze i wartości), aby uzyskać maksymalną wartość plecaka nie przekraczającą określonej maksymalnej wagi plecaka.

W rozwiązaniu tego problemu, skorzystamy z macierzy kosztów. Będzie ona przechowywać informacje o najwyższej możliwej wartości plecaka, dla danego podproblemu.

Podproblemem będzie plecak o mniejszej pojemności, oraz zmniejszony zestaw przedmiotów. Dzięki temu wyznaczymy maksymalne ułożenie elementów w plecaku dla każdej pojemności i każdego zestawu elementów, na bazie czego zbudujemy ostateczny wynik.

$V[0, j] = 0$  Dodatkowy, zerowy wiersz w macierzy PD (wyzerowany).

$V[i, 0] = 0$  Dodatkowa, zerowa kolumna w macierzy PD (wyzerowana).

Wiersze i kolumny o macierzy indeksie „0” pozostają wyzerowane, ponieważ nie będziemy z nich korzystać.

Dzieje się tak, ponieważ iterację zaczynamy od „1”. 1 to minimalna możliwa maksymalna waga oraz minimalna możliwa liczba wykorzystanych elementów.

$$V[i, j] = \begin{cases} V[i-1, j] & \text{if } w_i > j \\ \max\{V[i-1, j], V[i-1, j-w_i] + p_i\} & \text{if } w_i \leq j \end{cases}$$

Jeśli i -ty element nie mieści się w plecaku, weź rozwiązanie optymalne obliczone, gdy tego elementu nie było. $x_i = 0$

Jeśli i -ty element mieści się w plecaku, weź maksimum z wartości:

- Plecaka bez i -tego elementu ($x_i = 0$)
- Plecaka z i -tym elementem ($x_i = 1$). Wówczas w plecaku musi być miejsce na ten element (zrób miejsce: pomniejsz j - aktualny rozmiar plecaka o wagę i -tego elementu i do rozwiązania optymalnego dodaj wartość i -tego elementu).

Gdzie w_i oznacza wagę elementu i , a p_i oznacza jego wartość.

Ostatnia komórka macierzy będzie zawierała rozwiązanie głównego problemu:

Dane wejściowe

$c = 8$

$n = 4$

id	w_i	p_i
1	2	4
2	1	3
3	4	6
4	4	8

Macierz programowania dynamicznego V

$i \backslash j$	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	4	4	4	4	4	4	4
2	0	3	4	7	7	7	7	7	7
3	0	3	4	7	7	9	10	13	13
4	0	3	4	7	8	11	12	15	15

To jest rozwiązanie optymalne:
maksymalna wartość funkcji celu f^*
(max. suma wartości elementów w plecaku).

Aby odczytać, które przedmioty zostały zapakowane do plecaka, ustawiamy się w ostatniej komórce macierzy, i porównujemy jej wartość z komórką bezpośrednio wyżej. Jeśli komórka wyżej ma mniejszą wartość, oznacza to że element o indeksie wiersza macierzy, w którym się znajdujemy jest w plecaku.

Jeśli tak jest, przesuwamy się o wiersz w górę, oraz o w_i kolumn w lewo i dokonujemy tego samego porównania.

W przeciwnym wypadku przesuwamy się o wiersz w górę i wykonujemy to samo porównanie.

Czynność powtarzamy tak długo, aż dojdziemy do najmniejszego indeksu macierzy, co oznacza przejście przez jej wszystkie elementy lub aż waga plecaka, którą zmniejszamy po każdym napotkaniu elementu w plecaku będzie równa 0, co będzie oznaczało wypakowanie wszystkich jego elementów.

$i \backslash j$	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	4	4	4	4	4	4	4
2	0	3	4	7	7	7	7	7	7
3	0	3	4	7	7	9	10	13	13
4	0	3	4	7	8	11	12	15	15

- Idziemy do $V[i-1, j-w_i]$, na podstawie której wyznaczyliśmy $V[i, j]$.
- Czy x_3 jest w rozwiązaniu? $V[3, 4] = V[2, 4]$, więc x_3 nie jest w rozwiązaniu. Przesuwamy się o wiersz w górę, sprawdzamy czy x_2 jest w rozwiązaniu.

Implementacja rozwiązania problemu plecakaowego poprzez technikę programowania dynamicznego w C++.

Znajdywanie maksymalnej wartości plecaka:

```
for (int i = 1; i <= count; i++) {
    for (int j = 1; j <= weight; j++) {
        if (itemsArray[i - 1].weight <= j) {
            knapsackArray[i][j] = std::max(knapsackArray[i - 1][j], knapsackArray[i - 1][j - itemsArray[i - 1].weight] + itemsArray[i - 1].value);
        }
        else {
            knapsackArray[i][j] = knapsackArray[i - 1][j];
        }
    }
}

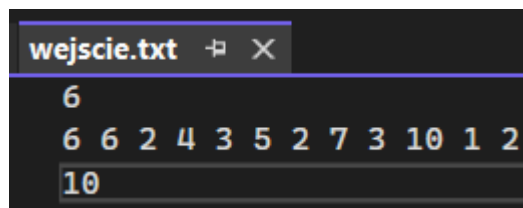
std::cout << "Maksymalna wartosc plecaka wynosi: " << knapsackArray[count][weight] << std::endl;
```

Oraz wypisywanie elementów plecaka o najwyższej wartości:

```
std::cout << "Zawartosc plecaka: \n";
int remainingWeight = weight;
for (int i = count; i > 0 && remainingWeight > 0; i--) {
    if (knapsackArray[i][remainingWeight] != knapsackArray[i - 1][remainingWeight]) {
        std::cout << itemsArray[i - 1].weight << " " << itemsArray[i - 1].value << std::endl;
        remainingWeight -= itemsArray[i - 1].weight;
    }
}
```

Prezentacja działania mojej implementacji w C++

Dla następujących danych:



```
wejście.txt
6
6 6 2 4 3 5 2 7 3 10 1 2
10
```

Otrzymujemy następujące wyniki:

```
Programowanie dynamiczne
1. Pobierz dane z klawiatury
2. Pobierz dane z pliku wejście.txt
Wybierz opcje:
2
Maksymalna wartosc plecaka wynosi: 26
Zawartosc plecaka:
3 10
2 7
3 5
2 4
```

Wnioski

Algorytm cechuje złożoność czasowa oraz pamięciowa $O(p * w)$

W celu rozwiązania problemu musimy przejść przez wszystkie podproblemy dla każdej kombinacji ilości elementów „p”, oraz wagi „w”, w wyniku czego złożoność jest ilorazem ilości elementów oraz maksymalnej wagi plecaka.

Bibliografia

<https://www.cs.put.poznan.pl/arybarczyk/TeoriaAiSD3.pdf>

https://www.cs.put.poznan.pl/mszachniuk/mszachniuk_files/lab_aisd/Szachniuk-ASD-t5.pdf