AKADEMIA GÓRNICZO-HUTNICZA im. Stanisława Staszica w Krakowie

# Programowanie dynamiczne – liniowe zagadnienie załadunku

Stanisław Olech - 412023

Automatyka i Robotyka

EAIiIB

# Zad. 1

Kod. 1 Zaimplementowany przez mnie algorytmu rozwiązywania Całkowitoliczbowego problemu liniowego.

```
#include <iostream>
#include <vector>

template <typename type>
type chose_size(std::vector<type> cost, type free_weight, type weight, bool
maximalize){
    // Funkcja wybiera liczbę przedmiotów do wpisania, znając ich: wagi,
funkcje celu i liczbę wolnych miejsc.

    int ans = 0;
    for (int i = 1; i != cost.size(); i++){
        if (maximalize and i * weight <= free_weight and cost[i] >
cost[ans]){
            ans = i;
        }
        if (not maximalize and i * weight <= free_weight and cost[i] <
cost[ans]){
            ans = i;
        }
    }
    return ans;
}

template <typename type>
std::vector<int> backpack_problem(std::vector<std::vector<type>> costs,
std::vector<type> weights, type max_weight, bool maximalize){
    // inicjacja oraz definicja potrzebnych tablic
    int m = max_weight + 1, n = weights.size() * 2;
    std::vector<std::vector<type>> tab(m, std::vector<type> (n, 0));
    std::vector<type> ans;


    // wpisanie do tablicy pierwszego obiektu
    for (int free_weigh = 0; free_weigh != max_weight + 1; free_weigh++) {
        type i = chose_size(costs[0], free_weigh, weights[0], maximalize);
        tab[free_weigh][0] = i;
        tab[free_weigh][1] += costs[0][i];
    }

    // wpisanie pozostałych obiektów
    for(int object_num = 1; object_num != weights.size(); object_num++) {
        for (int free_weigh = 0; free_weigh != max_weight + 1;
free_weigh++) {

            // liczenie nowych wag, które biorą pod uwagę poprzednie
elementy
            std::vector<type> new_cost = costs[object_num];
            for (int i = 0; i != new_cost.size(); i++){
                if (free_weigh - i * weights[object_num] < 0){break;}
                new_cost[i] += tab[free_weigh - i * weights[object_num]][2
* object_num - 1];
            }

            // wpisanie reszty elementów
            type i = chose_size(new_cost, free_weigh, weights[object_num],
```

```
maximalize);
            tab[free_weigh][2 * object_num] = i;
            tab[free_weigh][2 * object_num + 1] += new_cost[i];
        }
    }

    // odzyskiwanie liczby obiektów
    int sum = max_weight;
    for(int object_num = weights.size() - 1; object_num != -1; object_num--
) {
        ans.insert(ans.begin(), tab[sum][2 * object_num]);
        sum -= tab[sum][2 * object_num] * weights[object_num];

    }

    return ans;
}



int main() {
    std::vector<int> cost1 = {20, 18, 14, 11, 7, 2, 0};
    std::vector<int> cost2 = {9, 6, 3, 0};
    std::vector<int> cost3 = {6, 2, 0};
    std::vector<std::vector<int>> costs = {cost3, cost2, cost1};
    std::vector<int> weights = {3, 2, 1};
    int max_weight = 7;


    std::vector<int> ans = backpack_problem(costs, weights, max_weight,
false);
    for(int i = 0; i != ans.size(); i++){
        std::cout <<  ans[i] << " elementow o rozmiarze " << weights[i] <<
std::endl;
    }

    return 0;
}
```

kod źródłowy mojego algorytmu.

```
std::vector<int> cost1 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
std::vector<int> cost2 = {2, 2, 2, 2 ,2, 2};
std::vector<int> cost3 = {0, 3, 6, 12};
std::vector<int> cost4 = {1, 5, 25, 125};
std::vector<int> cost5 = {2, 2, 2, 2, 2};
std::vector<int> cost6 = {0, 5, 10};
std::vector<int> cost7 = {10, 20, 30, 40};
std::vector<int> cost8 = {0, 10, 20, 30};
std::vector<int> cost9 = {0, 50};
std::vector<int> cost10 = {0, 60, 120};
std::vector<int> weights = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
```

Rys. 1 Definicja mojego problemu z  dziesięcioma zmiennymi.

## Zad. 2

```
00 |0 00 0 00 0 00 0 10 0 10 0 12 0 13 0 13 0 15 0 15
01 |0 00 0 00 0 00 0 10 0 10 0 12 0 13 0 13 0 15 1 16
02 |0 00 0 00 0 00 0 10 0 10 0 12 0 13 0 13 0 15 2 17
03 |0 00 0 00 0 00 0 10 0 10 0 12 0 13 1 16 0 18 0 18
04 |0 00 0 00 0 00 0 10 0 10 0 12 1 17 0 17 0 19 0 19
05 |0 00 0 00 0 00 0 10 0 10 0 12 1 17 0 17 0 19 1 20
06 |0 00 0 00 0 00 0 10 1 15 0 17 0 18 2 19 0 21 0 21
07 |0 00 0 00 0 00 1 20 0 20 0 22 0 23 0 23 0 25 0 25
08 |0 00 0 00 1 10 0 20 0 20 0 22 2 37 0 37 0 39 0 39
09 |0 00 1 50 0 50 0 60 0 60 0 62 0 63 0 63 0 65 0 65
10 |1 30 1 50 0 50 0 60 0 60 0 62 0 63 0 63 0 65 1 66
11 |1 30 1 50 0 50 0 60 0 60 0 62 0 63 0 63 0 65 2 67
12 |1 30 1 50 0 50 0 60 0 60 0 62 0 63 1 66 0 68 0 68
13 |1 30 1 50 0 50 0 60 0 60 0 62 1 67 0 67 0 69 0 69
14 |1 30 1 50 0 50 0 60 0 60 0 62 1 67 0 67 0 69 1 70
15 |1 30 1 50 0 50 0 60 1 65 0 67 0 68 2 69 0 71 0 71
16 |1 30 1 50 0 50 1 70 0 70 0 72 0 73 0 73 0 75 0 75
17 |1 30 1 50 1 60 0 70 0 70 0 72 2 87 0 87 0 89 0 89
18 |1 30 1 50 1 60 0 70 0 70 0 72 2 87 0 87 0 89 1 90
19 |1 30 1 80 0 80 0 90 0 90 0 92 0 93 0 93 0 95 0 95
20 |2 60 1 80 0 80 0 90 0 90 0 92 0 93 0 93 0 95 1 96
```

Rys. 2 Tablica wyborów i wartości funkcji dla poszczególnych stanów.

```
1 elementow o rozmiarze 10
1 elementow o rozmiarze 9
0 elementow o rozmiarze 8
0 elementow o rozmiarze 7
0 elementow o rozmiarze 6
0 elementow o rozmiarze 5
0 elementow o rozmiarze 4
0 elementow o rozmiarze 3
0 elementow o rozmiarze 2
1 elementow o rozmiarze 1
```

Rys. 3. Wynik działania programu.

Uzyskana wartość celu 96.

# Zad. 3

- **Jakie założenia muszą być spełnione dla wag i zysków ?**
  - Maksymalna waga musi być nieujemna.
- **Co się stanie jeśli te założenia nie spełnimy (modyfikacja sposobu rozwiązania zadania)**
  - Program się nie wykona (nieskończona pętla)
- **Jaka jest złożoność obliczeniowa algorytmu?**
  - Algorytm ma złożoność zależną od kilku parametrów. Liczba maszyn (m), liczba maksymalnych stanów (u mnie są to kolejne liczby całkowite – s) oraz liczby przedmiotów oraz wag każdego typu. Wartość przedmiotów oraz wag jest trudna do wyliczenia więc oszacuje ją z góry jako s. Nigdy nie będzie więcej przedmiotów niż jest stanów, pod warunkiem, że nie ma przedmiotów z ujemną lub zerową wagą. W takim przypadku mój algorytm ma złożoność:

$$O(m \cdot s^2)$$

# Wnioski

Implementacja algorytmu programowania dynamicznego dla problemu załadunku wymaga odpowiedniego zdefiniowania wag, zysków oraz ograniczeń zasobowych. Użyłem do tego biblioteki std::vector w c++ by zminimalizować problemy w związku z odczytywaniem rozmiarów tabeli. Złożoność obliczeniowa algorytmu programowania dynamicznego zależy od liczby pod problemów i operacji wykonywanych dla każdego pod problemu. W przypadku problemu załadunku, złożoność może być zależna od liczby przedmiotów, pojemności plecaka oraz liczby dostępnych stanów lub kombinacji. Jednakże nie jest to duża złożoność. Ćwiczenie okazało się proste i satysfakcjonujące. Jest to kolejne zagadnienie z programowania dynamicznego które jest przydatne w problemach związanych z optymalizacją rozmaitych procesów.