AKADEMIA GÓRNICZO-HUTNICZA im. Stanisława Staszica w Krakowie

**Programowanie dynamiczne – liniowe**

**zagadnienie załadunku**

Stanisław Olech - 412023

Automatyka i Robotyka

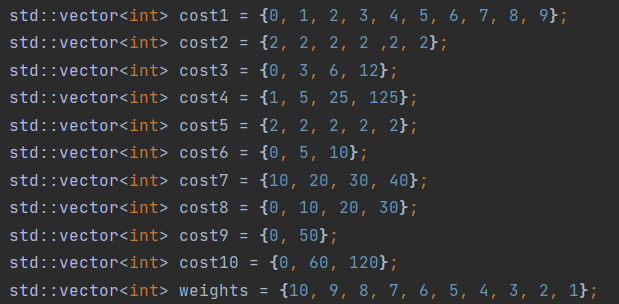
EAIiIB

**Zad. 1**

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

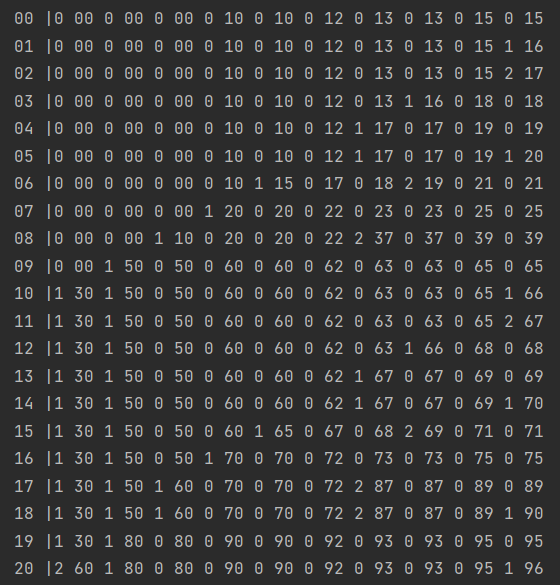
#include <iostream>  
#include <vector>  
  
template <typename type>  
type chose\_size(std::vector<type> cost, type free\_weight, type weight, bool maximalize){  
 // Funkcja wybiera liczbę przedmiotów do wpisania, znając ich: wagi, funkcje celu i liczbę wolnych miejsc.  
  
 int ans = 0;  
 for (int 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.

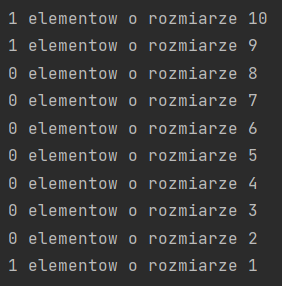


Rys. 1 Definicja mojego problemu z dziesięcioma zmiennymi.

**Zad. 2**



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



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:

**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.