AKADEMIA GÓRNICZO-HUTNICZA im. Stanisława Staszica w Krakowie

**Programowanie dynamiczne –**

**Wyznaczanie optymalnej wielkości partii**

**produkcyjnej**

Stanisław Olech - 412023

Automatyka i Robotyka

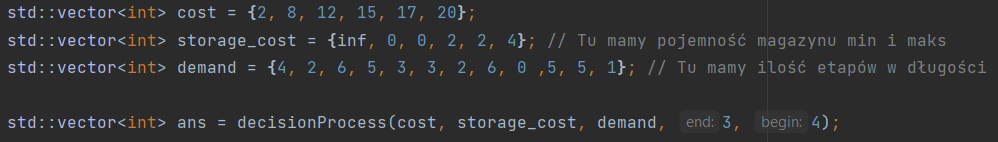
EAIiIB

**Zad. 1**

Kod. 1 Zaimplementowany przez mnie algorytmu

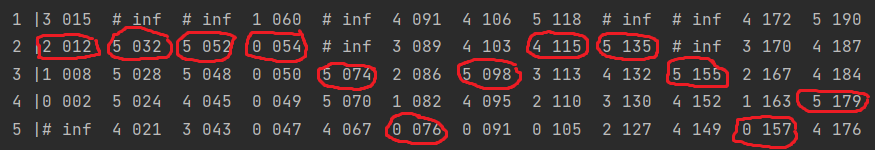
#include <iostream>  
#include <vector>  
#include <iomanip>  
#include <limits>  
int inf = std::numeric\_limits<int>::max();  
  
template <typename type>  
std::tuple<int, int> chose(std::vector<std::vector<type>>& tab, std::vector<type>& cost, std::vector<type>& storage\_cost, std::vector<type>& demand, type process, type state, type min\_storage){  
 int min = inf;  
 int min\_ind;  
 for(int i = 0; i != cost.size(); i++){  
 type next = tab[state][0] - demand[process] + i - min\_storage;  
 if (cost.size() - min\_storage - 1 < next or next < 0){continue;}  
 int x = (demand.size() - 1 - process) \* 2;  
 type next\_cost = tab[next][x];  
 type new\_cost = cost[i] + storage\_cost[next] + next\_cost;  
 if (next\_cost == inf or storage\_cost[next] == inf){new\_cost = inf;}  
  
 if (new\_cost < min){  
 min = new\_cost;  
 min\_ind = i;  
 }  
 }  
 if (min == inf){  
 return {-1, inf};  
 }  
 else{  
 return {min\_ind, min};  
 }  
}  
  
template <typename type>  
std::vector<type> decisionProcess(std::vector<type>& cost, std::vector<type>& storage\_cost, std::vector<type>& demand, type end, type begin){  
 // inicjacja oraz definicja potrzebnych tablic  
 int min\_storage = storage\_cost.size();  
 for(int i = 0; i != storage\_cost.size(); i++){  
 if (storage\_cost[i] != inf){  
 min\_storage = i;  
 break;  
 }  
 }  
  
 int m = storage\_cost.size() - min\_storage, n = demand.size() \* 2 + 1;  
 std::vector<std::vector<type>> tab(m, std::vector<type> (n, 0));  
 std::vector<type> ans;  
  
 for(int i = 0; i != tab.size(); i++){  
 tab[i][0] = min\_storage + i;  
 }  
  
 // wpisanie do tablicy pierwszego obiektu  
 for(int i = 0; i != tab.size(); i++){  
 int num = end - tab[i][0] + demand[demand.size()-1];  
 if (num < 0 or num > cost.size() - 1){  
 tab[i][1] = -1;  
 tab[i][2] = inf;  
 }  
 else{  
 tab[i][1] = num;  
 tab[i][2] = cost[tab[i][1]];  
 }  
  
 }  
  
 // dla każdego etapu od przedostatniego do drugiego  
 for(int y = demand.size() - 2; y != -1; y--){  
 int ind = (demand.size() - 1 - y) \* 2 + 1;  
 for(int i = 0; i != tab.size(); i++){  
 std::tuple<int, int> temp = chose(tab, cost, storage\_cost, demand, y, i, min\_storage);  
 tab[i][ind] = std::get<0>(temp);  
 tab[i][ind + 1] = std::get<1>(temp);  
 }  
 }  
  
 // odzyskanie  
 int state = begin;  
 for(int y = 0; y != demand.size(); y++){  
 int ind = (demand.size() - 1 - y) \* 2 + 1;  
 ans.push\_back(tab[state - min\_storage][ind]);  
 state = state + tab[state - min\_storage][ind] - demand[y];  
 }  
  
 // wyświetlanie  
 for(int i = 0; i != tab.size(); i++){  
 std::cout << tab[i][0] << " |";  
 for(int j = 1; j != tab[i].size(); j++){  
 if (j % 2 == 1){  
 if (tab[i][j] == -1){  
 std::cout << "#" << " ";  
 }  
 else{  
 std::cout << tab[i][j] << " ";  
 }  
 }  
 else{  
 if (tab[i][j] == inf){  
 std::cout << "inf" << " ";  
 }  
 else{  
 std::cout << std::setfill('0') << std::setw(3) << tab[i][j] << " ";  
 }  
 }  
  
 }  
 std::cout << std::endl;  
 }  
 std::cout << std::endl;  
 return ans;  
}  
  
  
  
int main() {  
 std::vector<int> cost = **{**2, 8, 12, 15, 17, 20**}**;  
 std::vector<int> storage\_cost = **{**inf, 0, 0, 2, 2, 4**}**; // Tu mamy pojemność magazynu min i maks  
 std::vector<int> demand = **{**4, 2, 6, 5, 3, 3, 2, 6, 0 ,5, 5, 1**}**; // Tu mamy ilość etapów w długości  
  
 std::vector<int> ans = decisionProcess(cost, storage\_cost, demand, 3, 4);  
 for(int i = 0; i != ans.size(); i++){  
 std::cout << ans[i] << " produkcja w " << i + 1 << " etapie"<< std::endl;  
 }  
  
 return 0;  
}

kod źródłowy mojego algorytmu.

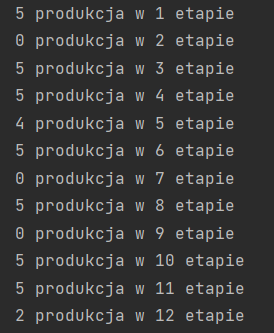


Rys. 1 Definicja mojego problem. Od góry: wektor kosztów produkcji, macierz kosztów składowania (inf oznacza że algorytm zawsze będzie mieć produkt w zapasie), macierz kolejnych zamówień. W wywołaniu funkcji kryje się wartość początkowa 4 produkty i wartość końcowa 3.

**Zad. 2**



Rys. 2 macierz decyzji optymalnych, wartości funkcji dla każdego etapu i rozważanego stanu. Na czerwono zostały pokazane decyzje.



Rys. 3. Wynik działania programu.

Wartość funkcji celu to 179.

**Zad. 3**

* **Jakie modyfikacje zagadnienia można dodać, aby rozszerzyć i bardziej dostosować model problemu do rzeczywistych uwarunkowań produkcyjnych?**
  + Możemy dodać więcej parametrów i więcej ograniczeń. Oznacza to jednak znaczące powiększenie złomności problemu.
* **Jaka jest złożoność obliczeniowa algorytmu?**
  + Algorytm ma złożoność zależną od ilości: etapów , stanów , liczby wyborów .

**Wnioski**

Implementacja algorytmu programowania dynamicznego dla problemu Wyznaczania optymalnej wielkości partii produkcyjnej tak jak 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 etapów, możliwych stanów i możliwych decyzji. Dla pojedynczego problemu nie jest to duża złożoność ale uwzględniając wiele parametrów dochodzimy do problemu wielowymiarowego z dużą liczbą wyborów oraz stanów. Ćwiczenie okazało się proste i satysfakcjonujące. Jest to kolejne zagadnienie z programowania dynamicznego które jest przydatne w problemach związanych z optymalizacją rozmaitych procesów.