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++){
```

```cpp
        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.

```
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,  end: 3,  begin: 4);
```

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

```
1 |3 015  # inf  # inf  1 060  # inf  4 091  4 106  5 118  # inf  # inf  4 172  5 190
2 |2 012  5 032  5 052  0 054  # inf  3 089  4 103  4 115  5 135  # inf  3 170  4 187
3 |1 008  5 028  5 048  0 050  5 074  2 086  5 098  3 113  4 132  5 155  2 167  4 184
4 |0 002  5 024  4 045  0 049  5 070  1 082  4 095  2 110  3 130  4 152  1 163  5 179
5 |# inf  4 021  3 043  0 047  4 067  0 076  0 091  0 105  2 127  4 149  0 157  4 176
```

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

```
5 produkcja w 1 etapie
0 produkcja w 2 etapie
5 produkcja w 3 etapie
5 produkcja w 4 etapie
4 produkcja w 5 etapie
5 produkcja w 6 etapie
0 produkcja w 7 etapie
5 produkcja w 8 etapie
0 produkcja w 9 etapie
5 produkcja w 10 etapie
5 produkcja w 11 etapie
2 produkcja w 12 etapie
```

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 e, stanów s, liczby wyborów e.

$$O(e \cdot s \cdot w)$$

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.