Sprawozdanie
Ćwiczenie 2 - PD dla WiTi

Krzysztof Górski 245079
prowadzący Dr inż. Mariusz Makuchowski

26.03.2021

Politechnika
Wrocławska

Proponowana ocena: 3.0
projekt realizowany w ramach kursu Sterowanie Procesami Dyskretnymi
grupa piątek 11:15

1 Wstęp

Celem zadania było opracowanie i przetestowanie algorytmu programowania dynamicznego WiTi dla zadanych zestawów danych.

2 Algorytm

Programowanie dynamiczne opiera się na podziale rozwiązywanego problemu na podproblemy względem kilku parametrów. W przypadku omawianym przez nas problem rysuje się w następujący sposób:

- Mamy do wykonania n zadań na pojedynczej maszynie.
- Jeśli dane zadanie jest spóźnione naliczana jest kara.
- Szukamy uszeregowania o minimalnej sumie kara.

Każde zadanie opisane jest trzema parametrami:

- pi - czas trwania,
- wi - waga,
- di - porządany termin zakończenia.

Celem algorytmu jest znalezienie optymalnego rozwiązania z uwzględnieniem podproblemów:

$$F(I) = min_{kI} F\{(I/\{k\}) + K_k(C(I))\}$$

Gdzie

- I - zbiór zadań
- F(I) - kara optymalnego uszerwgowania zadań I
- Kk (t) - kara zadania k zakończonego w czasie t
- C(I) - długość uszeregowania zadań I

3 Kod

```
#include <iostream>
#include <fstream>
#include <algorithm>
using namespace std;
//****************************************
//definicje globalne
//****************************************
// definicja typu danych
#define DATA_TYPE int
// zmienne i rozmiary
#define SIZE = 10
int LOOP = 11;
int I_size[11];
// sciezka pliku
string PATH = "./witi.data.txt";
string SOLUTION_PATH = "./solution.txt";
// rozpoznawanie zestawu danych
string DATAN[11] = { "data.10:","data.11:", "data.12:" ,
"data.13:", "data.14:","data.15:", "data.16:" ,
"data.17:", "data.18:", "data.19:", "data.20:" };
//****************************************
// KLASY
//****************************************
template<typename TYPE>
class DATA {
public:
    TYPE t;
    TYPE w;
```

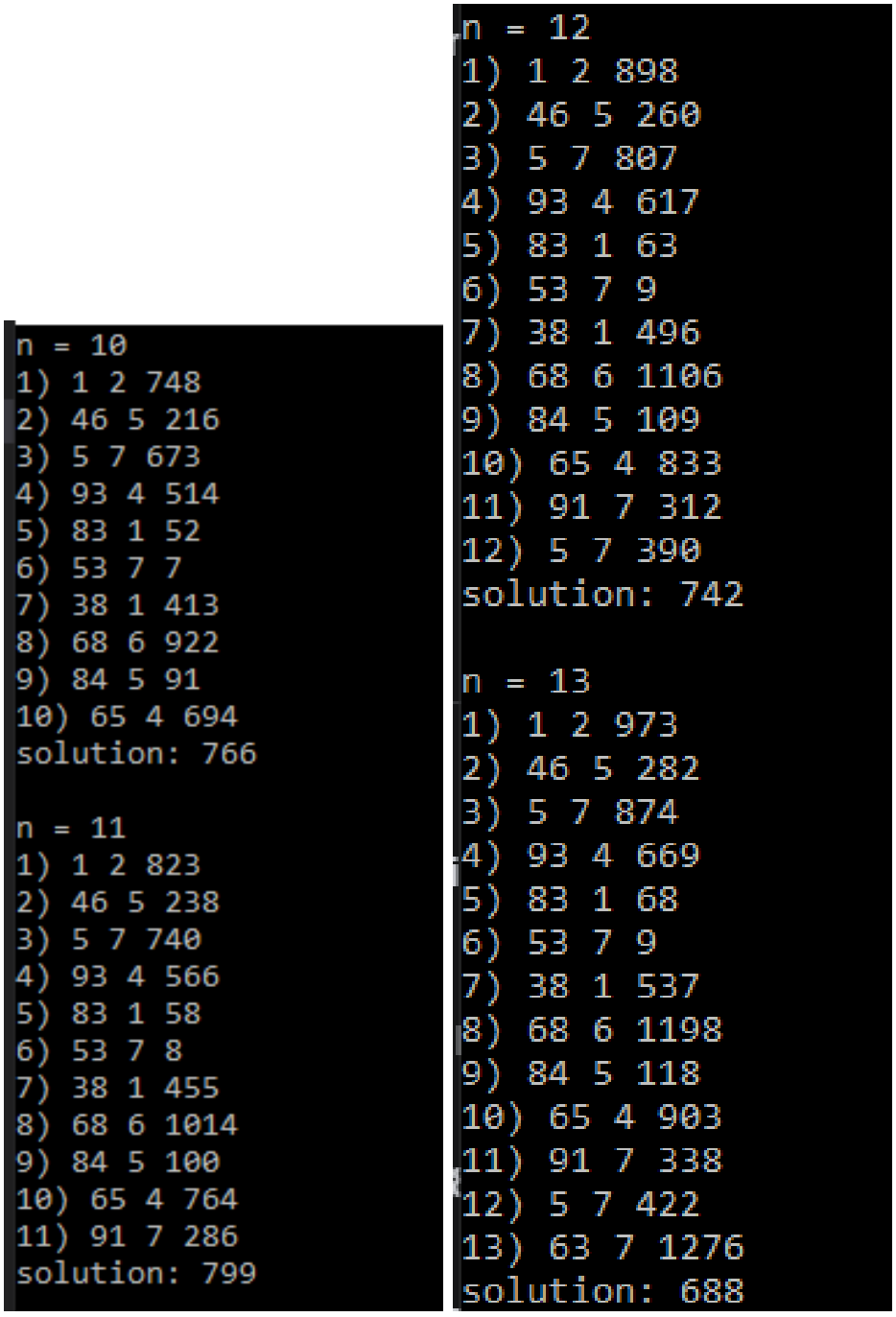
```
    TYPE r;
    int x; // kolejnosc
};
//****************************************
// FUNKCJE
//****************************************
void print_data(int I, DATA<DATA_TYPE>** data) {
    cout << "n = " << I_size[I] << endl;
        for (int i = 0; i < I_size[I]; i++)
    {
        cout << data[I][i].x+1 << ") " << data[I][i].t <<
        " " << data[I][i].w << " " << data[I][i].r << " "
        << endl;
    }
}
void read_from_file(int I, DATA<DATA_TYPE> **data,
fstream &file){
    string s;
    while (s != DATAN[I]) file >> s;
    file >> I_size[I];
    data[I] = new DATA<DATA_TYPE>[I_size[I]];
    for (int i = 0; i < I_size[I]; i++)
    {
    data[I][i].x = i;
    file >> data[I][i].t >> data[I][i].w >> data[I][i].r;
    }

}
void algorytm(int I, DATA<DATA_TYPE>** data)
```

```
{
    int bit = 1 << I_size[I];
    int* F = new int[bit];
    F[0] = 0;
    for (int bit_count = 1; bit_count < bit; bit_count++)
    {
       int pomtime = 0;
        for (int i = 0, b = 1; i < I_size[I]; i++, b *= 2)
        {
            if (bit_count & b)
            {
                pomtime += data[I][i].t;
            }
        }
        F[bit_count] = 999999;
        for (int i = 0, b = 1; i < I_size[I]; i++, b *= 2)
        {
            if (bit_count & b)
            {
                F[bit_count] = min(F[bit_count],
                F[bit_count - b] + data[I][i].w
                * max(pomtime - data[I][i].r, 0));
            }
        }
    }
    cout << "solution: " << F[bit - 1] << endl;
    cout << endl;
    delete[] F;
}
```

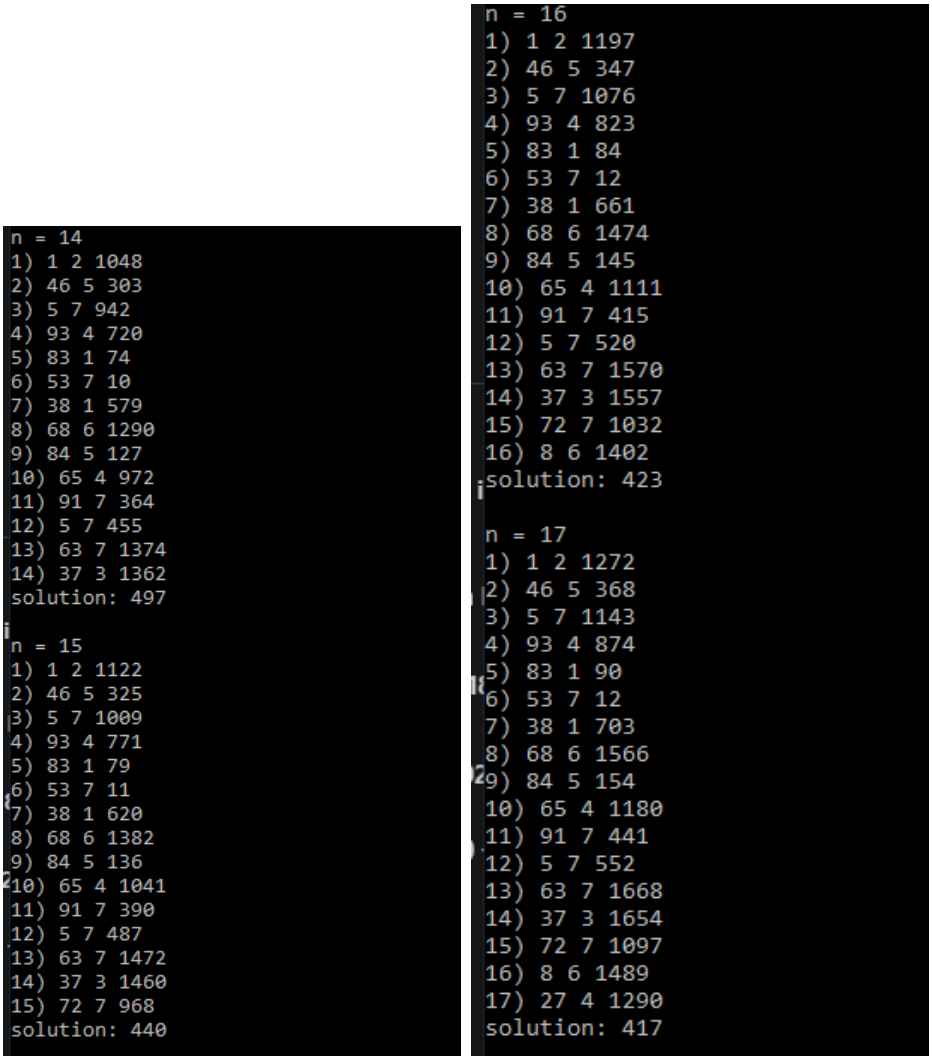
```
//*****************************************
// MAIN
//*****************************************
int main()
{
    DATA<DATA_TYPE>**data = new DATA<DATA_TYPE>*[10];
    fstream file;
    file.open("data.txt");
    for (int i = 0; i < LOOP; i++)
    {
        read_from_file(i, data, file);
        print_data(i, data);
        algorytm(i, data);

    }
    file.close();
    return 0;
}
```

4 Wyniki

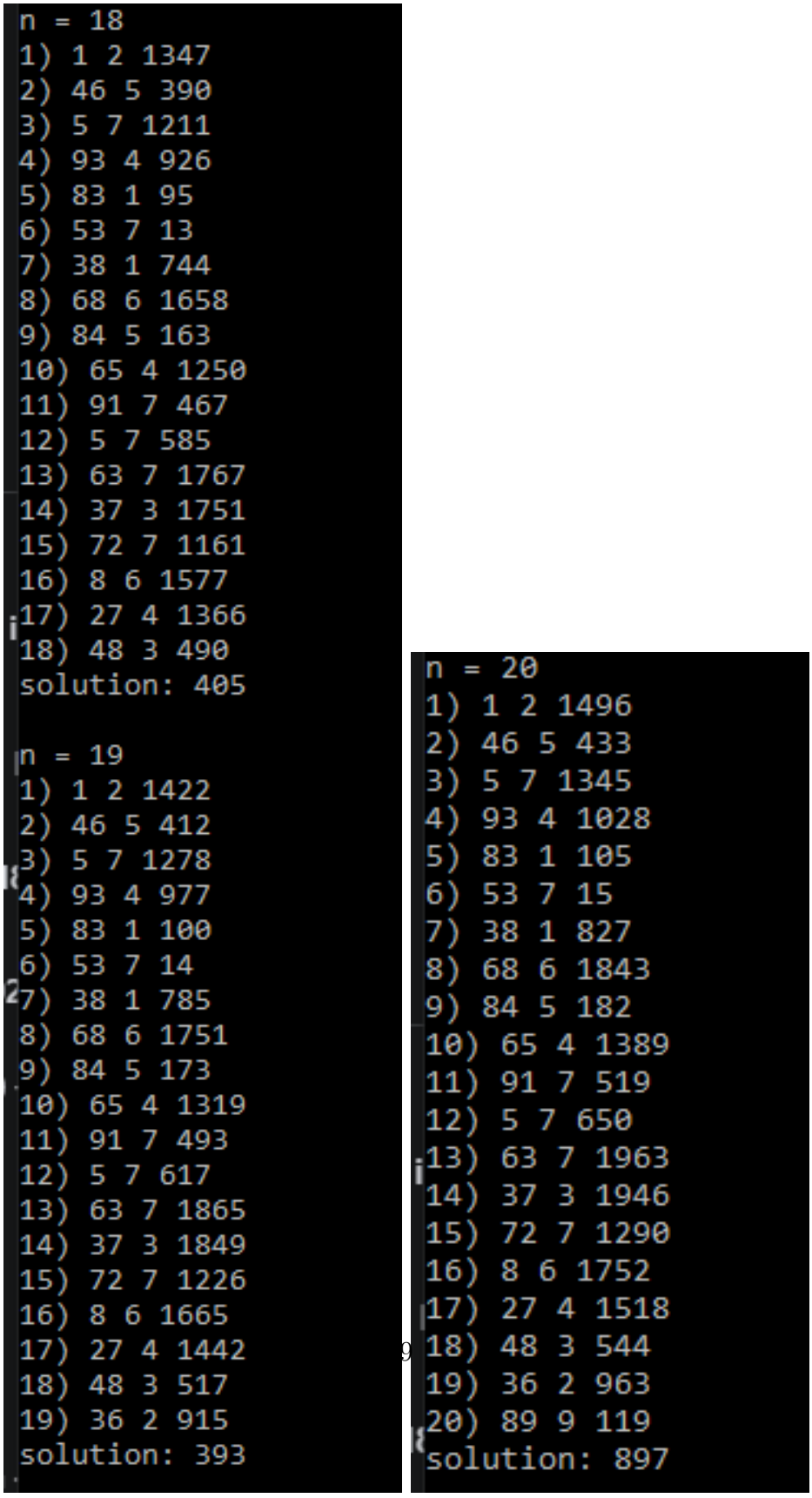
Algorytm zadziałał zgodnie z założeniem i dla każdego zestawu danych znalazł rozwiązanie optymalne.



Rysunek 1: wyniki dla zestawów 10-13



Rysunek 2: wyniki dla zestawów 14-18



Rysunek 3: wyniki dla zestawów 18-20

5 Wnioski

Algorytmy programowania dynamicznego pomimo skomplikowanych założeń dają się dość prosto zaimplementować w iteracyjnej formie. Większy problem stanowi zaimplementowanie permutacji, których nie udało się zastosować w tym rozwiązaniu.

Rozwiązanie programu na ocenę: 3.0

Spis rysunków