



Politechnika
Wrocławska

Wydział Informatyki i
Telekomunikacji



Platformy programistyczne .Net i Java - LAB1-2

Politechnika Wrocławska

Wydział Informatyki i telekomunikacji

Kierunek: Informatyczne systemy automatyki

grupa nr 2

github.com/wernexnrs/264254-.NET-i-Java

Dawid Popławski - 264254

Termin zajęć: Środa godz. 17⁰⁵ - 18⁴⁵

Prowadzący: mgr inż. Michał Jaroszczuk

Spis treści

1	Opis problemu	2
2	Aplikacja konsolowa	3
2.1	Drzewo projektu	3
2.2	Problem.cs	3
3	Testy jednostkowe	4
3.1	Drzewo projektu	4
4	Aplikacja okienkowa	6
4.1	Drzewo projektu	6

Opis problemu

Problem plecakowy to klasyczny problem optymalizacyjny, który polega na wybraniu zestawu przedmiotów o maksymalnej wartości, tak aby suma ich wag nie przekraczała określonej pojemności plecaka.

Dla każdego przedmiotu należało określić wartość binarną $x_i \in \{0, 1\}$, gdzie "0" oznaczało, że przedmiot nie zmieścił się do plecaka. Pojemność plecaka jest ograniczona pewną ustaloną na początku wartością, gdzie:

$$\sum_{i=1}^n x_i w_i \leq C$$

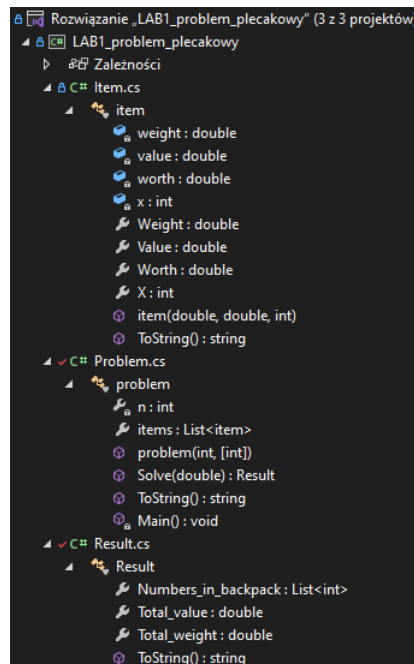
Zadaniem optymalizacyjnym było:

$$\max \sum_{i=1}^n x_i v_i, \quad \text{gdzie } v_i - \text{wartość}, w_i - \text{waga}$$

Na potrzeby projektu należało założyć, że wartość i waga pojedynczego przedmiotu będzie mieścić się w zakresie $v_i, w_i \in \langle 1, 10 \rangle$

Aplikacja konsolowa

Drzewo projektu



Rys. 2.1: Drzewo projektu aplikacji konsolowej.

Problem.cs

W tym pliku definiujemy klasę problem, która reprezentuje i rozwiązuje problem plecakowy.

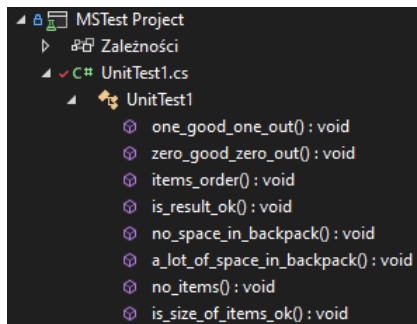
```
Odwolań: 10 | 7/7 sprawdzanie
public Result Solve(double capacity)
{
    items = [... items.OrderByDescending(item => item.Worth)];
    Result result = new();
    foreach (item element in items)
    {
        if (element.Weight <= capacity)
        {
            element.X = 1;
            capacity -= element.Weight;
            result.Total_value += element.Value;
            result.Total_weight += element.Weight;
            result.Numbers_in_backpack.Add(items.IndexOf(element));
        }
        if (capacity < items.Min(item => item.Weight)) break;
    }
    return result;
}
```

Rys. 2.2: Kod rozwiązujący problem plecakowy.

Główny algorytm iteruje po wszystkich elementach listy i sprawdza, czy waga danego przedmiotu jest mniejsza bądź równa aktualnej pojemności plecaka. W przypadku spełnienia warunku pole "X" tego obiektu zmieniane jest na 1, od pojemności plecaka odejmowana zostaje waga wrzuconego przedmiotu oraz zliczania jest suma całkowitej wartości, całkowitej wagi, do tego zapisywany jest indeks przedmiotu i wstawiany do osobnej listy.

Testy jednostkowe

Drzewo projektu



Rys. 3.1: Drzewo projektu testów jednostkowych.

Test	Czas trwa...
MSTest Project (8)	45 ms
MSTest_Project (8)	45 ms
UnitTest1 (8)	45 ms
a_lot_of_space_in_backpack	42 ms
is_result_ok	3 ms
is_size_of_items_ok	< 1 ms
items_order	< 1 ms
no_items	< 1 ms
no_space_in_backpack	< 1 ms
one_good_one_out	< 1 ms
zero_good_zero_out	< 1 ms

Rys. 3.2: Wynik testów jednostkowych.

Zrealizowanie testy jednostkowe:

- Sprawdzenie, czy jeśli co najmniej jeden przedmiot spełnia ograniczenia, to zwrócono co najmniej jeden element.
- Sprawdzenie, czy jeśli żaden przedmiot nie spełnia ograniczeń, to zwrócono puste rozwiązanie.
- Sprawdzenie, czy kolejność przedmiotów ma wpływa na znalezione rozwiązanie.
- Sprawdzenie poprawności wyniku dla konkretnej instancji.
- Sprawdzenie, czy jeżeli plecak nie ma miejsca to nie będzie w nim przedmiotów.
- Sprawdzenie, czy jeżeli plecak ma bardzo dużo miejsca to wszystkie przedmioty do niego trafiły.
- Sprawdzenie, czy algorytm działa gdy nie ma żadnych przedmiotów.
- Sprawdzenie, czy ilość wylosowanych przedmiotów zgadza się z ilością obiektów w liście.

```
[TestMethod]
[Odwolania: 0]
public void one_good_one_out()
{
    /*Sprawdzenie, czy jeśli co najmniej jeden
    przedmiot spełnia ograniczenia,
    to zwrócono co najmniej jeden element.*/

    problem problem = new(15,2);
    int capacity = 10;
    Result result = problem.Solve(capacity);
    Debug.Assert(problem.items.Any(item => item.Weight <= capacity) && result.Numbers_in_backpack.Count >= 1);
}

[TestMethod]
[Odwolania: 0]
public void zero_good_zero_out()
{
    /*Sprawdzenie, czy jeśli
    żaden przedmiot nie spełnia ograniczeń,
    to zwrócono puste rozwiązanie.*/

    problem problem = new(15, 2);
    double capacity = 0.0001;
    Result result = problem.Solve(capacity);
    Debug.Assert(problem.items.All(item => item.Weight > capacity) && result.Numbers_in_backpack.Count == 0);
}

[TestMethod]
[Odwolania: 0]
public void items_order()
{
    /*Sprawdzenie, czy kolejność przedmiotów
    ma wpływa na znalezione rozwiązanie.*/
    double capacity = 10;
    problem problem = new(15, 2);
    Result result = problem.Solve(capacity);
    problem problem_reversed = problem;
    problem_reversed.items.OrderBy(item => item.Weight);
    Result result_reversed = problem_reversed.Solve(capacity);
    Debug.Assert(result_reversed != result);
}
```

Rys. 3.3: Testy jednostkowe.

```

[TestMethod]
O | Odwołania: 0
public void is_result_ok()
{
    /*Sprawdzenie poprawności wyniku
    dla konkretnej instancji.*/

    double capacity = 10;
    List<item> items = [
        new item(1, 1, 0),
        new item(1, 1, 0),
        new item(1, 1, 0),
        new item(1, 1, 0),
        new item(1, 1, 0),
        new item(1, 1, 0),
        new item(1, 1, 0),
        new item(1, 1, 0),
        new item(1, 1, 0),
        new item(1, 1, 0)];

    problem problem_test = new(8, 0){ items = items};

    Result result = problem_test.Solve(capacity);
    List<int> good_result = [0, 1, 2, 3, 4, 5, 6, 7];
    CollectionAssert.AreEqual(good_result, result.Numbers_in_backpack);
    Assert.AreEqual(8, result.Total_value);
    Assert.AreEqual(8, result.Total_weight);
}

[TestMethod]
O | Odwołania: 0
public void no_space_in_backpack()
{
    /*Sprawdzenie, czy jeżeli plecak nie ma miejsca to nie będzie w nim przedmiotów.*/

    problem problem = new(15, 2);
    double capacity = 0;
    Result result = problem.Solve(capacity);
    Debug.Assert(result.Numbers_in_backpack.Count == 0);
}

[TestMethod]
O | Odwołania: 0
public void a_lot_of_space_in_backpack()
{
    /*Sprawdzenie, czy jeżeli plecak ma bardzo dużo miejsca to wszystkie przedmioty do niego trafiły.*/

    problem problem = new(1000, 2);
    double capacity = int.MaxValue;
    Result result = problem.Solve(capacity);
    Debug.Assert(result.Numbers_in_backpack.Count == 1000);
}

```

Rys. 3.4: Testy jednostkowe.

```

[TestMethod]
O | Odwołania: 0
public void no_items()
{
    /*Sprawdzenie, czy algorytm działa gdy nie ma żadnych przedmiotów.*/

    problem problem = new(0, 2);
    double capacity = 20;
    Result result = problem.Solve(capacity);
    Debug.Assert(result.Numbers_in_backpack.Count == 0 && result.Total_weight == 0 && result.Total_value == 0);
}

[TestMethod]
O | Odwołania: 0
public void is_size_of_items_ok()
{
    /*Sprawdzenie, czy ilość wylosowanych przedmiotów zgadza się z ilością obiektów w liście.*/

    problem problem = new(15, 2);
    Debug.Assert(problem.items.Count == 15);
}

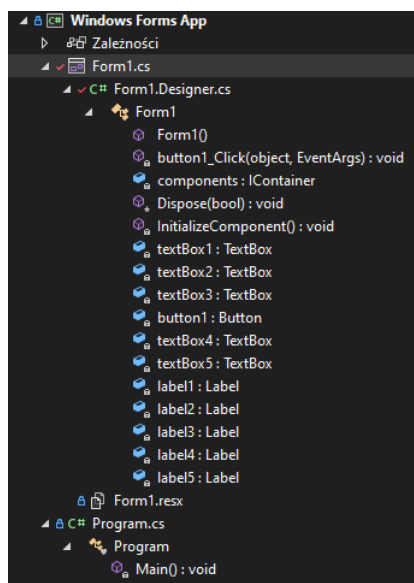
```

Rys. 3.5: Testy jednostkowe.

Aplikacja okienkowa

Aplikacja posiada proste zabezpieczenie w przypadku wpisania do textboxa innej wartości niż liczby.

Drzewo projektu



Rys. 4.1: Drzewo projektu aplikacji okienkowej.

```
1 using LAB1_problem_plecakowy;
2
3 namespace Windows_Forms_App
4 {
5     Odwołania: 3
6     public partial class Form1 : Form
7     {
8         1 odwołanie
9         public Form1()
10         {
11             InitializeComponent();
12         }
13
14         1 odwołanie
15         private void button1_Click(object sender, EventArgs e)
16         {
17             bool textBox1Valid = int.TryParse(textBox1.Text, out int n);
18             textBox1.BackColor = textBox1Valid ? System.Drawing.Color.Green : System.Drawing.Color.Red;
19
20             bool textBox2Valid = int.TryParse(textBox2.Text, out int seed);
21             textBox2.BackColor = textBox2Valid ? System.Drawing.Color.Green : System.Drawing.Color.Red;
22
23             bool textBox3Valid = int.TryParse(textBox3.Text, out int capacity);
24             textBox3.BackColor = textBox3Valid ? System.Drawing.Color.Green : System.Drawing.Color.Red;
25
26             if (!textBox1Valid || !textBox2Valid || !textBox3Valid)
27                 return;
28
29             problem mc = new(n, seed);
30             Result result = mc.Solve(capacity);
31             Console.WriteLine(mc.ToString());
32             Console.WriteLine(result.ToString());
33
34             textBox4.Text = result.ToString();
35             textBox5.Text = mc.ToString();
36         }
37     }
38 }
```

Rys. 4.2: Kod aplikacji okienkowej.