Kacper Malinowski, 263518

Automatyka i Robotyka, 6. Sem, spec. ARP

LAB 1 – Knapsack Problem

Link do repozytorium: https://github.com/kacper-malinowski/LAB1Malinowski

1. Opis projektu

Program służy do generowania i rozwiązywania problemu plecakowego metodą aproksymacyjną. Algorytm losowo generuje zestaw przedmiotów na podstawie danych wejściowych użytkownika, następnie sortuje je względem ich „efektywnej wartości", czyli ilorazu wartości i wagi, po czym zabiera przedmioty z góry listy dopóki plecak nie zostanie wypełniony lub przedmioty się nie skończą.

1.1 Klasy:

a) Item

```
public class Item
{
    public int ID;
    public int weight;
    public int value;
    public float effectiveValue;
}
```

Rysunek 1 Klasa Item

Klasa Item zawiera dane odnośnie konkretnego przedmiotu, jego ID, wagę, wartość oraz efektywną wartość, według której lista przedmiotów będzie sortowana.

b) Problem

```
public class Problem {
    public List<Item> items;
    IEnumerable<Item> ordered;
    Result knapsack;

    3 references
    public void GenerateItems(int itemAmount, int seed) ...

    6 references
    public void ViewItems() ...

    8 references
    public void sortItems() ...

    8 references
    public Result stealItems(int maxWeight) ...

    3 references
    public Result Solve(TerminalInput input) ...

}
```

Rysunek 2 Klasa Problem

Klasa problem zawiera listy służące przechowywaniu listy przedmiotów – posortowanej i nieposortowanej, oraz zmienną typu Result w której pojawi się rozwiązanie wygenerowane przez algorytm. Klasa ta zawiera także metody odpowiedzialne za tworzenie instancji problemu oraz jej rozwiązywanie.

d) Result

```
public class Result {
    public List<Item> stolenItems;
    public int maxWeight;
    public int totalValue = 0;
    public int totalWeight = 0;

    12 references
    public Result() ...

    1 reference
    public Result(int var) ...

    0 references
    public override string ToString() ...
}
```

Rysunek 3 Klasa Result

Klasa Result przechowuje rozwiązanie problemu plecakowego – listę „ukradzionych" przedmiotów, maksymalną i wykorzystaną pojemność plecaka oraz wartość przedmiotów w nim się znajdujących. Klasa zawiera konstruktor oraz przeciążenie funkcji ToString(), aby uprościć wyświetlanie wyników problemu.

e) TerminalInput

```
public class TerminalInput {
    public int itemAmount;
    public int seed;
    public int maxWeight;
    1 reference
    public TerminalInput downloadInput() {
        TerminalInput input = new TerminalInput();

        Console.WriteLine("Enter the amount of items");
        input.itemAmount = Convert.ToInt32(Console.ReadLine());

        Console.WriteLine("Enter the seed");
        input.seed = Convert.ToInt32(Console.ReadLine());

        Console.WriteLine("Enter the knapsack capacity");
        input.maxWeight = Convert.ToInt32(Console.ReadLine());

        return input;
    }
}
```

Rysunek 4 Klasa TerminalInput

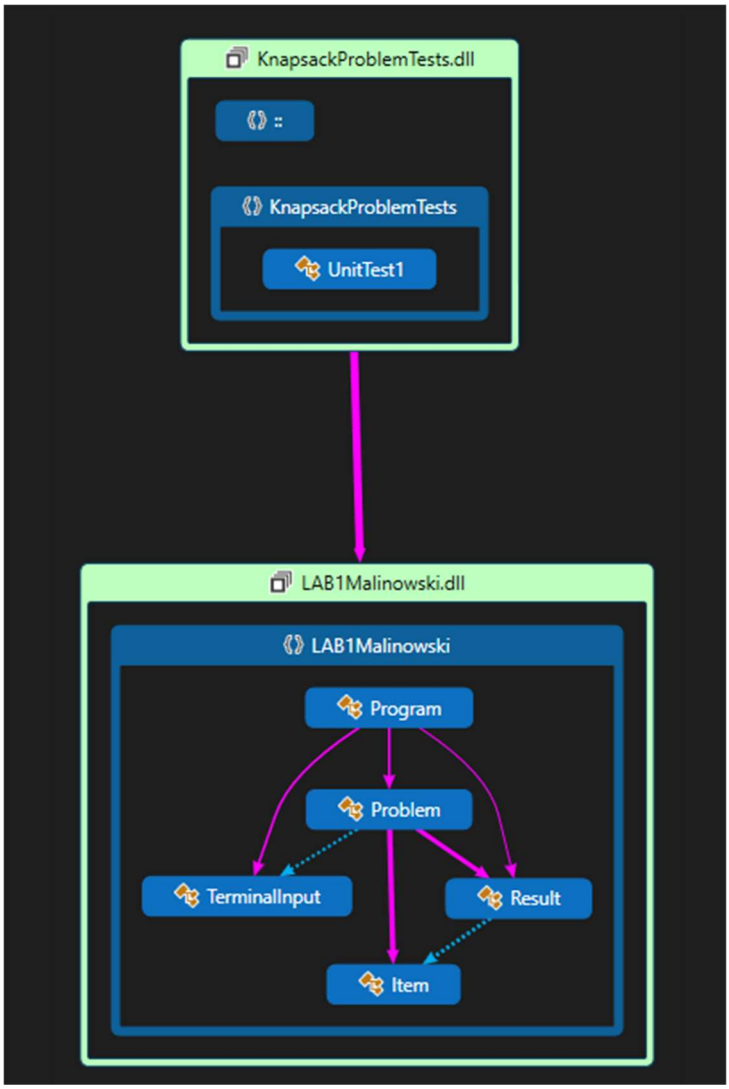
Klasa TerminalInput obsługuję wprowadzanie danych wejściowych przez użytkownika poprzez terminal. Metoda wczytująca działa niezależnie od algorytmu rozwiązywania co umożliwia jego zmianę na przykładowo interfejs graficzny.

2. Opis testów

Program zawiera 8 testów sprawdzających poprawność działania algorytmu.

- CommonInstance() – Typowy przypadek z poprawnymi danymi
- EmptyInstance() – Przypadek w którym nie mamy żadnych przedmiotów do umieszczenia w plecaku.
- DoesTheItemOrderMatter() – Przypadek zawierający dwie listy z tymi samymi przedmiotami jednak ułożonymi w innej kolejności.
- CustomInstance() – Instancja problemu z ręcznie wpisanymi danymi, dla których ręcznie obliczono prawidłowe rozwiązanie.
- NoSpaceInBackpack() – Plecak nie maksymalną pojemność równą 0 – nie można w nim umieścić żadnego przedmiotu.
- NoItemsFitBackpack() – Maksymalna pojemność plecaka jest mniejsza niż masa jakiegokolwiek przedmiotu na liście.
- BackpackTooLarge() – Pojemność plecaka jest znacznie większa niż suma przedmiotów na liście.

3. Drzewo projektu



4. Kluczowa część algorytmu

```
3 references | 2/2 passing
public void GenerateItems(int itemAmount, int seed) {
    items = new List<Item>();
    Random random = new Random(seed);
    for (int i = 0; i < itemAmount; i++) {
        Item item = new Item();
        item.ID = i;
        item.weight = random.Next(1, 10);
        item.value = random.Next(1, 10);
        item.effectiveValue = (float)item.value / (float)item.weight;
        items.Add(item);
    }
}

8 references | 6/6 passing
public void sortItems() {
    ordered = items.OrderByDescending(item => item.effectiveValue);
}

8 references | 6/6 passing
public Result stealItems(int maxWeight) {
    knapsack = new Result(maxWeight);
    foreach (Item item in ordered) {
        if(knapsack.totalWeight == knapsack.maxWeight) {break; }
        if (knapsack.totalWeight + item.weight <= knapsack.maxWeight) {
            knapsack.stolenItems.Add(item);
            knapsack.totalWeight += item.weight;
            knapsack.totalValue += item.value;
        }
    }
    return knapsack;
}
```

```
3 references | 2/2 passing
public Result Solve(TerminalInput input) {
    Result knapsack = new Result();
    GenerateItems(input.itemAmount, input.seed);
    sortItems();
    knapsack = stealItems(input.maxWeight);
    return knapsack;
}
```