Politechnika Wrocławska

Platformy programistyczne .Net i Java

Dokumentacja programu rozwiązującego problem plecakowy

Prowadzący: Mgr. inż. Michał Jaroszczuk

Grupa Środa, 18:55-20:40

Jan Klisowski 263485

13.03.2024

Spis treści

1. Opis Projektu

Projekt dotyczy implementacji algorytmu rozwiązującego problem plecakowy (*Knapsack Problem*) w języku C#. Program pozwala na rozwiązanie problemu plecakowego dla zadanej liczby przedmiotów, pojemności plecaka oraz ziarna losowości. Wykorzystuje on algorytm zachłanny do wyboru odpowiednich przedmiotów do umieszczenia w plecaku w taki sposób, aby maksymalizować wartość przedmiotów przy ograniczeniu wagi plecaka.

2. Opis Klas i Metod

2.1. Klasa `Program`

Klasa `Program` zawiera jedną metodę - `Main`. Metoda ta pobiera od użytkownika parametry niezbędne do rozwiązania problemu plecakowego, takie jak liczba przedmiotów, ziarno losowości oraz pojemność plecaka. Następnie tworzony jest obiekt klasy `Problem`, a algorytm rozwiązujący problem plecakowy jest uruchamiany.

2.2. Klasa `Problem`

Klasa `Problem` jest główną klasą reprezentującą problem plecakowy. Odpowiada za generowanie instancji problemu, przechowywanie listy przedmiotów oraz rozwiązywanie problemu z wykorzystaniem algorytmu zachłannego.

2.2.1. Pola

- `_numberOfItems`: Prywatne pole przechowujące liczbę przedmiotów w problemie.
- `_items`: Prywatne pole przechowujące listę przedmiotów w problemie.

2.2.2. Konstruktor

- `Problem(int numberOfItems, int seed)`: Konstruktor klasy `Problem`. Przyjmuje liczbę przedmiotów `numberOfItems` oraz ziarno losowości `seed`. Wewnątrz konstruktora generowane są losowo przedmioty o zadanych wartościach i wagach, które są przechowywane w liście `_items`.

2.2.3. Metody

- `GetItem(int index): Item`: Metoda zwracająca przedmiot znajdujący się pod wskazanym indeksem `index` w liście `_items`. Została użyta w teście jednostkowym.
- `AddItem(Item item): void`: Metoda dodająca nowy przedmiot `item` do listy `_items`. Została użyta w teście jednostkowym.
- `Solve(int capacity): Result`: Metoda rozwiązująca problem plecakowy. Przyjmuje pojemność plecaka `capacity` jako argument. Algorytm wybiera przedmioty o najwyższym stosunku wartości do wagi (sortowanie malejące), dopóki suma wag wybranych przedmiotów nie przekracza pojemności plecaka. Zwraca obiekt klasy `Result` zawierający listę numerów wybranych przedmiotów, całkowitą wartość oraz wagę wybranych przedmiotów.

Metoda `Solve` jest kluczowym elementem klasy `Problem`, gdyż implementuje algorytm rozwiązujący problem plecakowy. Zastosowanie sortowania przedmiotów według stosunku wartości do wagi pozwala na efektywne wybieranie najlepszych przedmiotów dla danego problemu.

2.3. Klasa `Item`

Klasa `Item` przechowuje informacje o pojedynczym przedmiocie, takie jak jego wartość (`Value`) i wagę (`Weight`).

2.4. Klasa `Result`

Klasa `Result` przechowuje wynik działania algorytmu rozwiązującego problem plecakowy. Zawiera listę indeksów wybranych przedmiotów (`ItemNumbers`), całkowitą wartość (`TotalValue`) oraz wagę (`TotalWeight`) wybranych przedmiotów.

2.5. Testy Jednostkowe w Klasie `UnitTest1`

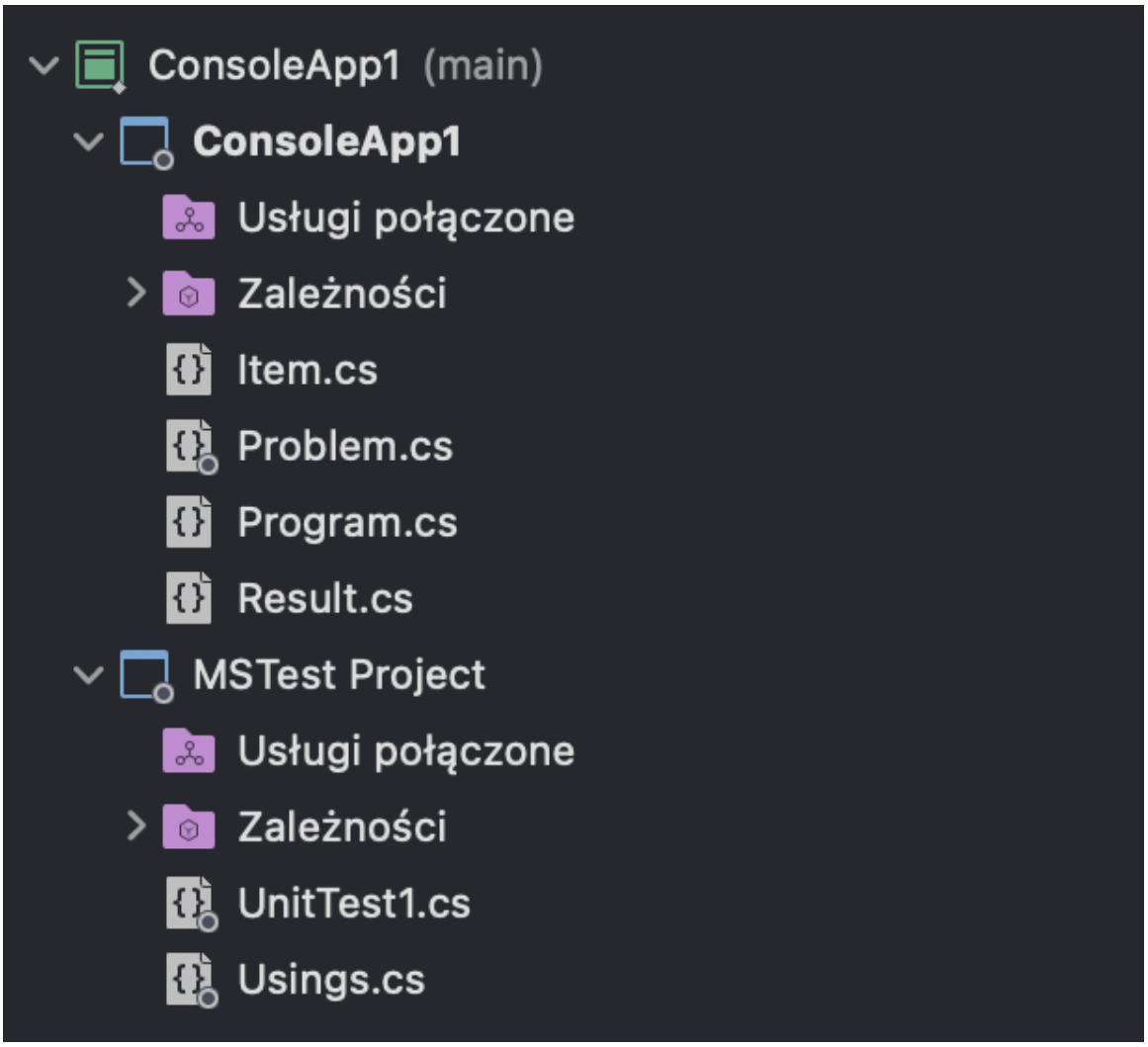
- **TestCoNajmniejJeden**: Sprawdzenie, czy jeśli co najmniej jeden przedmiot spełnia ograniczenia, to zwrócono co naj- mniej jeden element.
- **TestNiePasuje**: Sprawdzenie, czy jeśli żaden przedmiot nie spełnia ograniczeń, to zwrócono puste rozwiązanie.
- **TestKolejnoscBezZnaczenia**: Sprawdzenie, czy kolejność przedmiotów ma wpływa na znalezione rozwiązanie.
- **TestKonkretnaInstancja**: Sprawdzenie poprawności wyniku dla konkretnej instancji.
- **TestKolejnoscSortowania**: Sprawdza, czy przedmioty są wybierane w kolejności malejącej wartości do wagi.
- **TestMaxPojemnosc**: Sprawdza, czy wszystkie przedmioty zostaną wybrane, gdy pojemność plecaka jest większa lub równa sumie wag przedmiotów.
- **TestUjemnaPojemnosc**: Sprawdza, czy dla ujemnej pojemności plecaka nie zostanie wybrany żaden przedmiot.
- **TestDuzaPojemnosc**: Sprawdza, czy dla bardzo dużej pojemności plecaka zostaną wybrane co najmniej niektóre przedmioty.

3. Repozytorium

Link do repozytorium projektu: *https://github.com/jachoofrachoo/netjavaKlis*

4. Zdjęcia

4.1. Drzewo Projektu



Rys. 1: Drzewo Projektu

4.2. Kluczowy Fragment Programu

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.CompilerServices;

[assembly: InternalsVisibleTo("MSTest Project")]
namespace ConsoleApp1
{

    public class Problem
    {
        private int _numberOfItems;
        private List<Item> _items;

        public Problem(int numberOfItems, int seed)
        {
            _numberOfItems = numberOfItems;
            _items = new List<Item>();

            Random random = new Random(seed);
            for (int i = 0; i < _numberOfItems; i++)
            {
                int value = random.Next(1, 11);
                int weight = random.Next(1, 11);
                _items.Add(new Item(value, weight));
            }
        }

        public Item GetItem(int index)
        {
            return _items[index];
        }


        public void AddItem(Item item)
        {
            _items.Add(item);
        }

```

Rys. 2: Kluczowy Fragment Programu

```
        public Result Solve(int capacity)
        {
            List<Item> sortedItems = _items.OrderByDescending(item => (double)item.Value / item.Weight).ToList();

            List<int> selectedItems = new List<int>();
            int totalValue = 0;
            int totalWeight = 0;

            Console.WriteLine("Przedmioty wygenerowane:");
            foreach (var item in _items)
            {
                Console.WriteLine($"Wartosc: {item.Value}, Waga: {item.Weight}");
            }
            Console.WriteLine();

            foreach (var item in sortedItems)
            {
                if (totalWeight + item.Weight <= capacity)
                {
                    selectedItems.Add(_items.IndexOf(item) + 1);
                    totalValue += item.Value;
                    totalWeight += item.Weight;
                }
                else
                {
                    break;
                }
            }

            return new Result
            {
                ItemNumbers = selectedItems,
                TotalValue = totalValue,
                TotalWeight = totalWeight
            };
        }
    }
}

```

Rys. 3: Kluczowy Fragment Programu