Politechnika Wrocławska

Platformy programistyczne .Net i Java

Projekt aplikacji okienkowej Java na przykładzie prostego problemu optymalizacyjnego

Prowadzący: Mgr. inż. Michał Jaroszczuk

Grupa Środa, 18:55-20:40

Jan Klisowski 263485

18.06.2024

Spis treści

1. Opis Projektu - Algorytm Plecakowy

Projekt polegał na implementacji algorytmu plecakowego (ang. knapsack problem) w języku Java. Algorytm ten służy do rozwiązywania problemu optymalizacji wyboru przedmiotów o określonej wadze i wartości tak, aby maksymalizować łączną wartość przedmiotów mieszczących się w plecaku o określonej pojemności.

2. Opis Klas i Metod

2.1. Klasa `Main`

Klasa `Main` jest punktem wejścia do aplikacji. Umożliwia użytkownikowi wprowadzenie parametrów takich jak liczba przedmiotów, ziarno generatora losowego oraz pojemność plecaka. Następnie generuje instancję problemu i wyświetla wygenerowane przedmioty oraz rozwiązanie problemu.

2.2. Klasa `Problem`

Klasa `Problem` odpowiada za generowanie przedmiotów o losowych wartościach i wagach oraz rozwiązywanie problemu plecakowego przy użyciu algorytmu zachłannego.

```
public class Problem {
    private int n;
    private int seed;
    private int lowerBound;
    private int upperBound;
    private List<Item> items;

    public Problem(int n, int seed, int lowerBound, int upperBound) {
        this.n = n;
        this.seed = seed;
        this.lowerBound = lowerBound;
        this.upperBound = upperBound;
        this.items = new ArrayList<>();
        generateItems();
    }

    public Problem(List<Item> items) {
        this.items = items;
        this.n = items.size();
    }



    private void generateItems() {
        Random random = new Random(seed);
        for (int i = 0; i < n; i++) {
            int value = lowerBound + random.nextInt(upperBound - lowerBound + 1);
```

```
            int weight = lowerBound + random.nextInt(upperBound - lowerBound +
                1);
            items.add(new Item(value, weight));
        }
    }

    public Result solve(int capacity) {
        items.sort((a, b) -> Double.compare((double) b.value / b.weight,
            (double) a.value / a.weight));

        int remainingCapacity = capacity;
        int totalValue = 0;
        int totalWeight = 0;
        List<Integer> itemCounts = new ArrayList<>();

        for (int i = 0; i < items.size(); i++) {
            itemCounts.add(0);
        }

        for (int i = 0; i < items.size(); i++) {
            Item item = items.get(i);
            while (remainingCapacity >= item.weight) {
                remainingCapacity -= item.weight;
                totalValue += item.value;
                totalWeight += item.weight;
                itemCounts.set(i, itemCounts.get(i) + 1);
            }
        }

        return new Result(itemCounts, totalValue, totalWeight);
    }

    public List<Item> getItems() {
        return items;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (Item item : items) {
            sb.append(item.toString()).append("\n");
        }
        return sb.toString();
    }
}
```

2.3. Klasa `Item`

Klasa `Item` reprezentuje pojedynczy przedmiot z określoną wartością i wagą.

```
class Item {
    int value;
    int weight;

    Item(int value, int weight) {
        this.value = value;
        this.weight = weight;
    }

    @Override
    public String toString() {
        return "Item{" +
                "value=" + value +
                ", weight=" + weight +
                '}';
    }
}
```

2.4. Klasa `Result`

Klasa `Result` przechowuje wynik rozwiązania problemu plecakowego, w tym listę liczników przedmiotów, łączną wartość oraz wagę przedmiotów umieszczonych w plecaku.

```
class Result {
    List<Integer> itemCounts;
    int totalValue;
    int totalWeight;

    Result(List<Integer> itemCounts, int totalValue, int totalWeight) {
        this.itemCounts = itemCounts;
        this.totalValue = totalValue;
        this.totalWeight = totalWeight;
    }

    @Override
    public String toString() {
        return "Result{" +
                "itemCounts=" + itemCounts +
                ", totalValue=" + totalValue +
                ", totalWeight=" + totalWeight +
                '}';
    }
}
```

2.5. Testy Jednostkowe

Testy jednostkowe zostały przeprowadzone przy użyciu biblioteki JUnit w celu weryfikacji poprawności działania algorytmu.

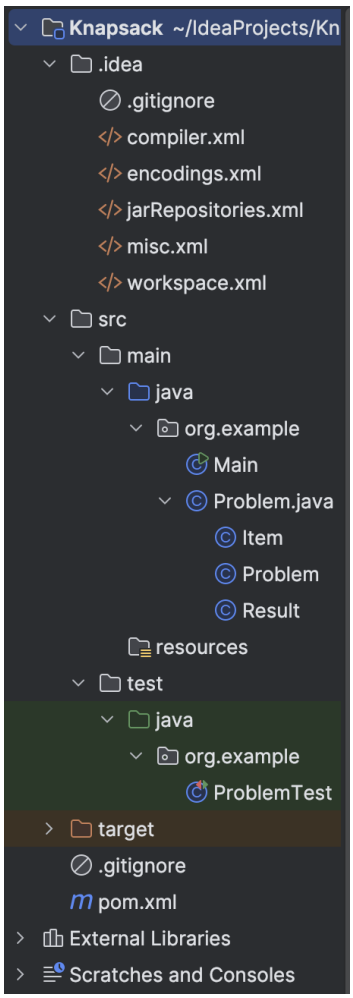
```
public class ProblemTest {

    @Test
    public void testCoNajmniejJedenPasuje() {
        Problem problem = new Problem(5, 123, 1, 10);
        int capacity = 20;
        Result result = problem.solve(capacity);
        assertTrue(result.totalWeight > 0);
    }

    @Test
    public void testNIcNIePasuje() {
        Problem problem = new Problem(5, 1, 11, 20);
        int capacity = 0;
        Result result = problem.solve(capacity);
        assertEquals(0, result.totalWeight);
        assertEquals(0, result.totalValue);
    }

    @Test
    public void testSprawdzanieWagiWartosci() {
        Problem problem = new Problem(5, 123, 1, 10);
        List<Item> items = problem.getItems();
        for (Item item : items) {
            assertTrue(item.weight >= 1 && item.weight <= 10);
            assertTrue(item.value >= 1 && item.value <= 10);
        }
    }

    @Test
    public void testDlaInstancji() {
        List<Item> items = new ArrayList<>();
        items.add(new Item(5, 2));
        items.add(new Item(10, 3));
        items.add(new Item(8, 1));
        items.add(new Item(3, 5));
        Problem problem = new Problem(items);
        int capacity = 6;
        Result result = problem.solve(capacity);
        assertEquals(6, result.totalWeight);
        assertEquals(48, result.totalValue);
        assertEquals(List.of(6, 0, 0, 0), result.itemCounts);
```

```
    }
}
```

3. Repozytorium

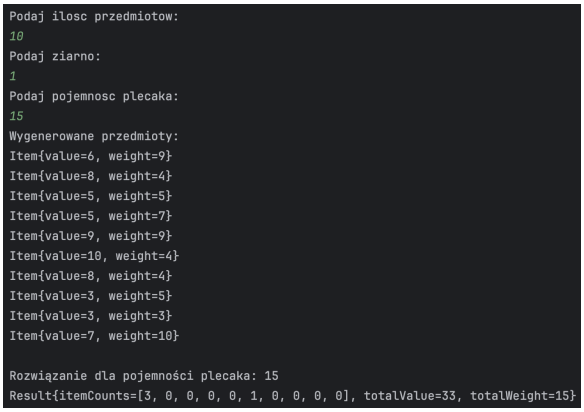
Link do repozytorium projektu: *https://github.com/jachoofrachoo/netjavaKlis*

3.1. Drzewo Projektu



Rysunek 1: Drzewo Projektu

3.2. Działanie aplikacji



Rysunek 2: Prezentacja działania