Politechnika Wrocławska

Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Platformy programistyczne JAVA i .NET

Zadanie 5

Autorstwa
Jakub Dołharz 272561

1 OPIS ZADANIA

Zadanie polegało na rozwiązaniu nieograniczonego problemu plecakowego w języku java. Ten problem różni się od zwykłego problemu plecakowego tym, że do plecaka można wstawic nielimitowaną ilość tego samego przemsiotu (o ile pojemność na to pozwala)

2 WYKONANIE ZADANIA

2.1 Polecenie 1 i 2

Pierwsze zadanie polegało na implementacji problemu, drugie na jego rozwiązaniu. Dlatego oba te zadania zostaną omówione razem. Implementacja problemu i jego rozwiązanie, składa się z trzech klas: Problem, Przedmiot i Result. Za generowanie przedmiotów odpowiedzialna jest klasa Problem. Przedmioty mają losowe wartości i wagi z określonego zakresu. Metoda solve(int capacity) (należąca do klasy Problem) implementuje algorytm zachłanny, sortując przedmioty według stosunku wartości do wagi, a następnie wybierając maksymalną liczbę najcenniejszych przedmiotów mieszczących się w plecaku. Wynik jest zwracany jako obiekt klasy Result, zawierający listę wybranych przedmiotów, ich ilości, summę wartości i wagę. Implementacja zapewnia kontrolę poprawności danych wejściowych poprzez walidację w konstruktorze i metodzie solve().

```
Give number of items:
13
Give seed:
5
Give knapsack capacity:
77
Generated items:
No: 0 value: 8 weight: 3
No: 1 value: 5 weight: 5
No: 2 value: 7 weight: 6
No: 3 value: 5 weight: 2
No: 4 value: 3 weight: 2
No: 5 value: 2 weight: 4
No: 6 value: 8 weight: 1
No: 7 value: 9 weight: 1
No: 8 value: 9 weight: 6
No: 9 value: 8 weight: 8
No: 10 value: 9 weight: 6
No: 11 value: 9 weight: 7
No: 12 value: 2 weight: 3

No: 7 Ilosc: 77

Weight: 77
Value: 693
```

Obraz 1: Przykład poprawnie rozwiązanego problemu

```
Give number of items:
4
Give seed:
5
Give knapsack capacity:
71
Generated items:
No: 0 value: 8 weight: 3
No: 1 value: 5 weight: 5
No: 2 value: 7 weight: 6
No: 3 value: 5 weight: 2

No: 0 Ilosc: 23
No: 3 Ilosc: 1

Weight: 71
Value: 189
```

Obraz 2: Kolejny przykład poprawnie rozwiązanego problemu

```
Give number of items:
2
Give seed:
99
Give knapsack capacity:
33
Generated items:
No: 0 value: 8 weight: 9
No: 1 value: 10 weight: 2

No: 1 Ilosc: 16

Weight: 32
Value: 160

Process finished with exit code 0
```

Obraz 3: Ostatni przykład poprawnie rozwiązanego problemu

3 POLECENIE 3

Trzecia część zadania polegała na napisaniu testów jednostkowych. nakazane były 4 konkretne testy, których wykonanie jest opisane poniżej.

```
private final int TEST_SEED = 123;
private final int TEST_ITEMS_COUNT = 5;
private final int LOWER_BOUND = 1;
private final int UPPER_BOUND = 10;

@BeforeEach
void setUp() {
    problem = new Problem(TEST_ITEMS_COUNT, TEST_SEED, LOWER_BOUND, UPPER_BOUND);
}
```

Obraz 4: Przygotowanie tworzące przykładowy problem, jest wywoływane przed każdym testem.

```
@Test
void solveShouldReturnAtLeastOneItemWhenCapacityIsSufficient() {
    Result result = problem.solve(20);
    assertFalse(result.getNumeryPrzedmiotow().isEmpty(),
            "Powinien zwrócić przynajmniej jeden przedmiot gdy pojemność jest wystarczająca");
    assertTrue(result.getSumaWag() > 0,
            "Sumaryczna waga powinna być większa od zera");
}
```

Obraz 5: Test sprawdzający czy otrzymamy rozwiązanie przy chociaż jednym spełniającym wymagania przedmiocie. Problem jest generowany z użyciem wyżej wspomnianego przygotowania, a za pojemność plecaka przyjęte jest 20. Test przebiegł prawidłowo.

```java
@Test
void solveShouldReturnEmptyResultWhenCapacityIsZero() {
    Result result = problem.solve(0);
    assertTrue(result.getNumeryPrzedmiotow().isEmpty(),
            "Nie powinien zwrócić żadnych przedmiotów gdy pojemność jest zerowa");
    assertEquals(0, result.getSumaWag(),
            "Sumaryczna waga powinna być zerowa");
    assertEquals(0, result.getSumaWartosci(),
            "Sumaryczna wartość powinna być zerowa");
}
```

Obraz 6: Ten test sprawdza czy otrzymamy puste rozwiązanie, jeśli żaden przedmiot nie spełnia wymagań. Używamy tego samego problemu, ale za ograniczenie pojemności plecaka ustawiamy 0. W ten sposób żaden przedmiot nie może się zmieścić i nie spełnia wymagań. Test przebiegł prawidłowo

```java
@Test
void itemsShouldHaveValuesAndWeightsWithinBounds() {
    List<Przedmiot> items = problem.getPrzedmioty();
    for (Przedmiot item : items) {
        assertTrue(item.getWartosc() >= LOWER_BOUND && item.getWartosc() <= UPPER_BOUND,
                "Wartość przedmiotu poza zakresem");
        assertTrue(item.getWaga() >= LOWER_BOUND && item.getWaga() <= UPPER_BOUND,
                "Waga przedmiotu poza zakresem");
    }
}
```

Obraz 7: Ten test sprawdza czy wagi i wartości przedmiotu znajdują się w założonym przedziale. Test przebiegł prawidłowo.

```java
@Test
void solveShouldReturnCorrectResultForSpecificInstance() {
    Problem specificProblem = new Problem(3, 42, 1, 10);

    // Dla ziarna 42 i 3 przedmiotów powinniśmy dostać:
    // No: 0 v: 1 w: 4
    // No: 1 v: 9 w: 5
    // No: 2 v: 1 w: 6

    Result result = specificProblem.solve(10);



    assertEquals(1, result.getNumeryPrzedmiotow().size());
    assertEquals(2, (int)result.getIlosci().get(0));
    assertEquals(10, result.getSumaWag());
    assertEquals(18, result.getSumaWartosci());
}
```

Obraz 8: Ten test sprawdza czy zadanie jest poprawnie rozwiązane dla konkretnych wartości. Aby go wykonać tworzymy konkretny obiekt klasy problem o 3 przedmiotach z seedem 42. Test przebiegł prawidłowo.