ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

ФАКУЛТЕТ ПО КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ

КОМПЮТЪРНО И СОФТУЕРНО ИНЖЕНЕРСТВО

Изследване на операциите и приложно програмиране

Курсов проект

„ Реализация на динамично програмиране – задача за натоварване“

Съставил: Ангел Любомиров Стойнов

Факултетен номер: 121222150

Група: 40

Съдържание

Постановка на задачата

Общ капацитет на кораб – $\boldsymbol{Q}$ тона.

За всеки контейнер са известни теглото $\boldsymbol{q_i}$ и стойността $\boldsymbol{c_i}$.

Трябва да се определи по колко броя да се вземат от всеки тип контейнер, така че да не се надвиши общата товароподемност на кораба, а стойността на натоварената стока да е максимална.

Нека общият капацитет на кораба е Q = 10, контейнерите са от 4 различни типа, а теглото и стойността на всеки контейнер са зададени на таб. 1:

Контейнер	**Тегло q_i**	**Стойност c_i**
1	2	4
2	8	10
3	3	6
4	4	8

(Таблица 1 – показва примерни стойности)

Дефиниране на задачата като задача от динамичното програмиране

Управлението на стъпка i ще е количеството предмети x_i, които да се вземат на тази стъпка. Задачата има същата постановка, както задачата за разпределение на ресурсите, само функциите $f_i(x)$ се определят от стойността на $c_i x_i$;

Управляемата система S е количеството останал капацитет до запълване на кораба. Рекурентната зависимост е $W_i(S, x_i) = c_i x_i + W_{i+1}(S - q_i x_i)$.

Условната оптимална печалба на всяка стъпка е:

$$Wi(S) = max \{c_i x + W_{i+1}(S - q_i x)\},$$

а условното оптимално управление $x_i(S)$ е тази стойност на x, за която е достигната максималната стойност на $W_i(S)$.

Реализация

За реализация на задачата е използван програмния език **java** и **apache** библиотека за логване на резултата - **org.apache.logging.log4j**.

Контейнерите са изразени по следния начин (фиг. 1):

```java
package com.uni.angel.container;

public record Container(int weight, int value) { } 20 usages

```

(Фиг 1, показва дефиницията на контейнерите, съдържаща тегло и стойност)

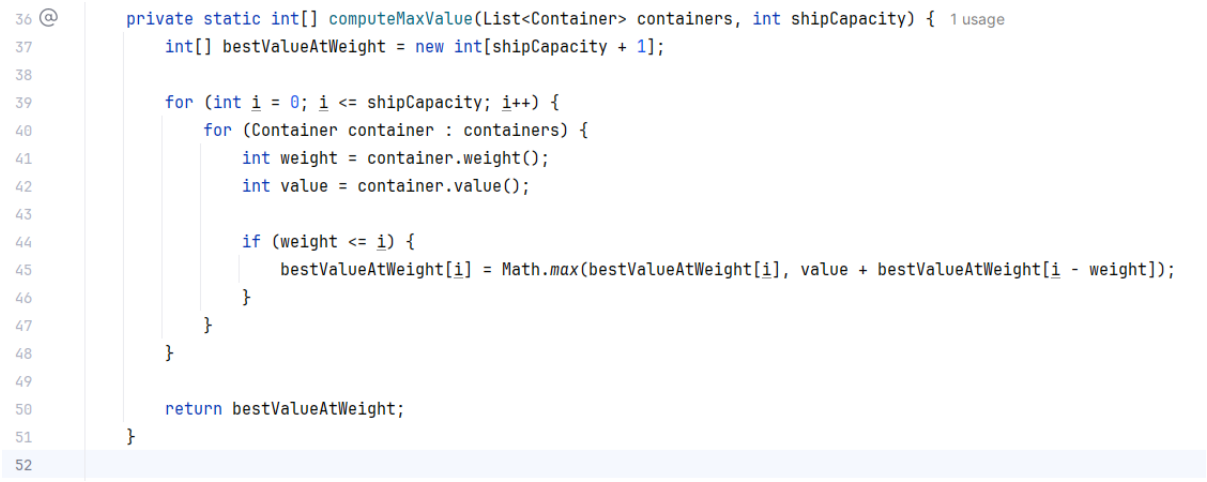
Използвани са примерните стойности (таб. 1):

```java
package com.uni.angel.container;

import java.util.List;

public class ExampleContainers { 4 usages

    private ExampleContainers() { throw new UnsupportedOperationException("Cannot instantiate"); }

    public static List<Container> getContainers() { 4 usages
        return List.of(
                new Container( weight: 2,  value: 4),
                new Container( weight: 8,  value: 10),
                new Container( weight: 3,  value: 6),
                new Container( weight: 4,  value: 8)
        );
    }
}

```

(Фиг 2, показва списък от четири различни контейнера)

Реализацията на алгоритъма е обемна, за това ще се разгледа на отделни стъпки:

1. Намиране на максималната стойност, която може да се получи при всеки възможен капацитет.
2. Намиране на всички възможни решения.
3. Визуализация на получените резултати.



```java
private static int[] computeMaxValue(List<Container> containers, int shipCapacity) {  1 usage
    int[] bestValueAtWeight = new int[shipCapacity + 1];

    for (int i = 0; i <= shipCapacity; i++) {
        for (Container container : containers) {
            int weight = container.weight();
            int value = container.value();

            if (weight <= i) {
                bestValueAtWeight[i] = Math.max(bestValueAtWeight[i], value + bestValueAtWeight[i - weight]);
            }
        }
    }

    return bestValueAtWeight;
}
```

(Фигура 3, показва метода за намиране на максимална стойност)

Методът приема два аргумента – списък от контейнери и товароспособността на кораба. Масивът bestValueAtWeight представлява максималната стойност, която можем да получим при общо тегло ***w***. Външният for цикъл обхожда всяко тегло i. Вътрешният обхожда всеки контейнер от списъка. Идеята е да се провери дали даден контейнер може да се използва, стига той да се побира.